Seminar-Unterlagen

8-Bit Systeme

KONTRON ELEKTRONIK GRUPPE

KONTRON ELEKTRONIK

8-Bit Systeme

Inhaltsverzeichnis

1. Überblick über ein Z80-Mikroprozessorsystem

1.1 Z80-Bus-Organisation

2. Beschreibung Z80-CPU

2.1 Interner CPU-Aufbau
2.2 Registersatz der Z80-CPU
2.3 Hauptregistersatz
2.4 Zweitregistersatz
2.5 I-Register
2.6 R-Register
2.7 IX und IY-Register
2.8 SP-Register
2.9 PC-Register
2.10 Anschlüsse der Z80-CPU
2.11 Zeitverhalten

3. Einführung in den Z80-Befehlssatz

4. Periphere Bausteine des Z80-Systems

4.1 Prinzipschaltung eines Z80-Systems
4.2 Allgemeines Funktionsprinzip der I/O-Bausteine
4.3 PIO
4.4 CTC
4.5 SIO
4.6 DMA

5. Interrupttechnik in Z80-Systemen
5.1 Definition
5.2 Nicht maskierbarer Interrupt
5.3 Maskierbarer Interrupt

5.3.1 Mode 0
5.3.2 Mode 1
5.3.3 Mode 2

5.4 Verlassen einer ISR
5.5 Prioritäten
5.6 Abarbeiten eines Interrupts

6. Vorgehensweise bei der Programmerstellung

6.1 Hauptpunkte bei der Programmentwicklung
6.2 Problemanalyse
6.3 Erarbeitung eines Lösungsweges
6.4 Kodierung

7. Techn. Manual CPU

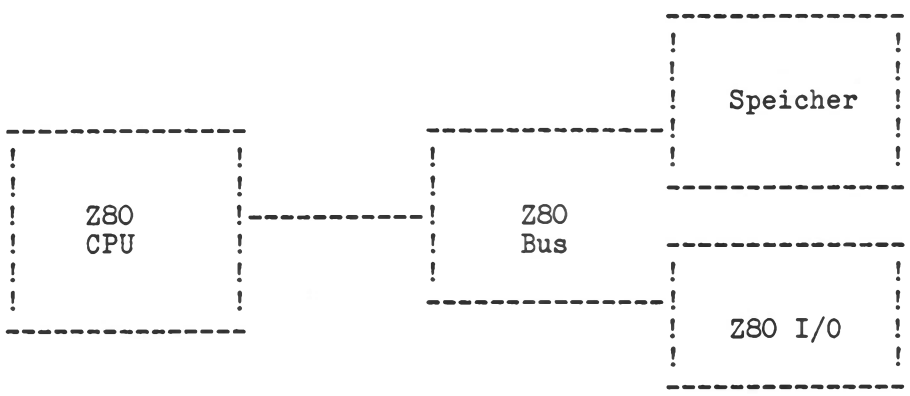
8. Techn. Manual PIO

9. Techn. Manual CTC

10. Techn. Manual SIO

11. Beispielprogramme

1. Überblick über ein Z80-Mikroprozessorsystem



Die Zentraleinheit besteht aus einer universellen Logikschaltung, die vorher definierte Befehle einziehen, verstehen und ausführen kann.

Die I/O-Einheiten übernehmen die Kommunikation mit der Außenwelt.

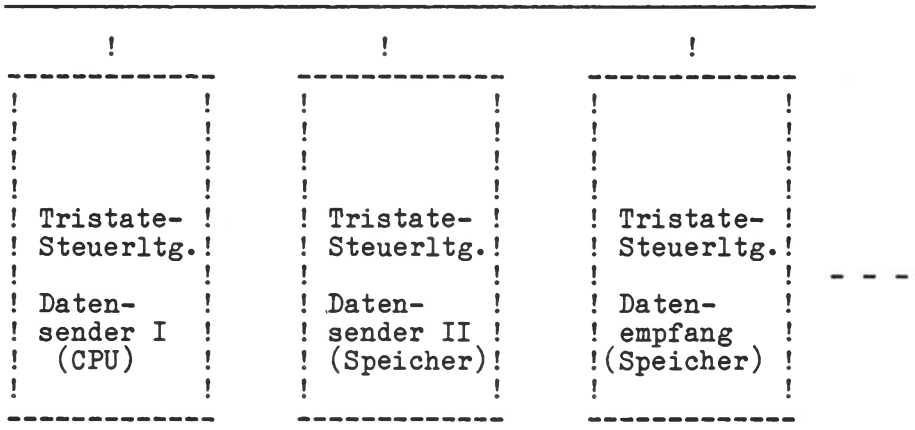
CPU und I/O-Bausteine sind architektonisch aufeinander abgestimmt und können nur **zusammen** ihre volle Leistungsfähigkeit (Interrupt) entwickeln.

Der Speicher kann im Gegensatz dazu aus Bausteinen der verschiedensten Hersteller aufgebaut sein, sofern eine geschwindigkeitsmäßige (Zugriffszeit), sowie pegelmäßige Anpassung sichergestellt ist.

Der Bus stellt die Summe aller Verbindungsleitungen zwischen den drei Elementen dar.

1.1 Z80-Bus-Organisation

Alle Teilnehmer liegen über Tristate-Puffer parallel am Datenbus.



Nur je zwei Einheiten dürfen am Datenverkehr teilhaben, alle anderen müssen abgeschaltet (im hochomigen Zustand) sein.

a) Adreßbus

Die CPU gibt die Adresse der Einheit aus, mit der sie in Verbindung treten will
(Unibus)

b) Datenbus

Auf ihm erfolgt der Datenverkehr zwischen CPU, Speichern und I/O-Einheiten.
(Bidirektionaler Bus).

c) Steuerbus

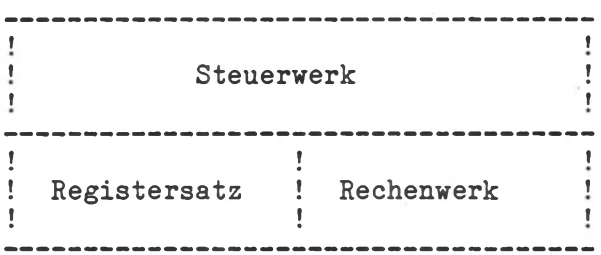
Die CPU beeinflußt die angesprochene Einheit durch Steuerbus-Signale (RD, WR etc.). Die CPU wird beeinflußt durch Steuerbus-Signale (RESET, WAIT, INT, NMI etc.).

2. Beschreibung der Z80-CPU

Wesentliche Merkmale:

* 8 Datenleitungen D0....D7 ---> 8-Bit-Maschine
* 16 Adreßleitungen ---> Speicher bis 64k (65536)
* Alle Steuersignale 'aktiv low', dadurch höhere Treiberleistung und bessere Störfestigkeit
* single-5V-Versorgung
* alle Anschlüsse TTL-kompatibel
* zwei Interrupteingänge NMI/INT
* Refreshlogik für dynam. Speicher im Prozessor integriert
* 5V Einphasentakt

2.1 Interner Aufbau

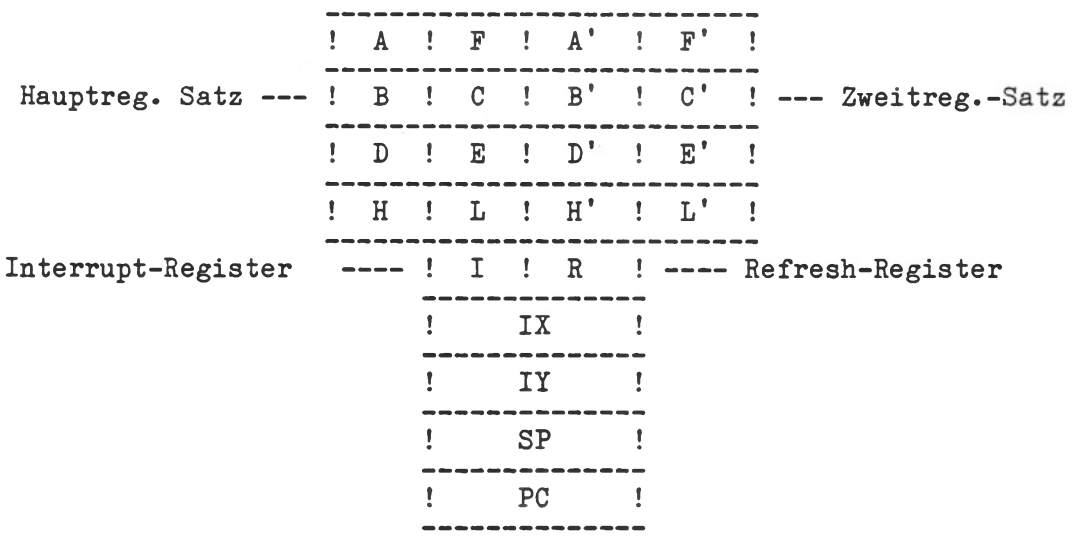


Steuerwerk: Holt, decodiert und führt die Befehle aus

Rechenwerk: Führt Verknüpfungen arithm. und logischer Art durch

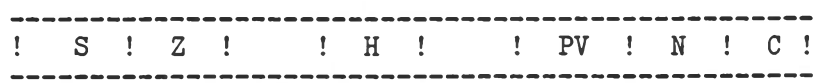
Registersatz: Dient der Zwischenspeicherung von Werten in der CPU ---> dadurch **schneller** Zugriff möglich.

2.2 Registersatz der Z80-CPU



2.3 Hauptregistersatz

- besteht aus acht 8-Bit-Registern
- das Register A (Akkumulator) nimmt eine Sonderstellung ein. Bei Verknüpfungen logischer (AND OR usw.) oder arithmetischer (AND SUB usw.) Art steht sowohl ein Operand als auch das Ergebnis im Register A (Einadreßmaschine!).
- Das R-Register enthält die Flags, die je nach Ausgang des vorhergegangenen Befehls gesetzt bzw. rückgesetzt worden sind.



NB: Nicht jeder Befehl beeinflußt die Flags!

- Von den acht zur Verfügung stehenden Bits werden nur sechs bentutz.

Kurz-bezeichnung	Flagname	Bedeutung
Z	Zero-Flag	Ist gesetzt, wenn eine Operation das Ergebnis 0 hat.
C	Carry/Link-Flag	Ist gesetzt bei Entstehen eines Übertrages bezüglich des höchstwertigen Bits.
S	Sign-Flag	Ist gesetzt, wenn das Ergebnis einer Operation im höchstwertigen Bit eine 1 aufweist. (Rechnen mit Vorzeichen).
H	Half-Carry-Flag	Ist gesetzt, wenn eine Addition oder Subtraktion einen Übertrag bzeüglich Bit 4 des Akkumulators erzeugt hat. (BCD-Rechnung).
N	Add/Subtract-Flag	Ist gesetzt, wenn die vorausgegangene Operation eine Subtraktion war.
P/V	Parity/Overflow-Flag log. Operationen: Parity-Flag	Ist gesetzt, wenn das Resultat einer logischen Verknüpfung 'even' ist.
	arithm. Operationen: Overflow-Flag	Ist gesetzt, wenn ein Übertrag bezüglich Bit 6 stattgefunden hat (Vorzeichenrechnung).

Das F-Register kann nicht durch direktes Eingeben eines Wertes beeinflußt werden. Die Beeinflussung erfolgt nur durch den Ausgang bestimmter Befehle.

Die Register B, C, D, E, H und L sind frei verwendbar.

Um auch mit 16-bit-Einheiten möglichst einfach operieren zu können (z.B. Adressen!), ist es möglich, je zwei Register zu einem Registerpaar zusammenzufassen und als eine Größe zu behandeln. Zusammenfaßbar sind so B und C, D und E, sowie H und L.

2.4 Zweitregistersatz (gestrichener Registersatz)

- besitzt den gleichen Aufbau wie der Hauptregistersatz
- kann **nicht direkt** benützt werden
- mit zwei Befehlen kann der Inhalt des Zweitregistersatzes gegen den Inhalt des Hauptregistersatzes **getauscht** werden
- dient in erster Linie zur superschnellen Rettung der Hauptregisterwerte bei dringenden, zeitkritischen Programmunterbrechungen (Interrupts).

2.5 I-Register

- Das Interrupt-Vektorregister dienst zur Auffindung der Einsprungadresse der entsprechenden Interrupt-Service-Routine (siehe 'Interrupt in Z80-Systemen').
- Hier liegt das höherwertige Byte einer Adresse, die in die sog. Interrupttabelle zeigt, in der dann die Startadresse der entsprechenden Interrupt-Service-Routine zu finden ist.

2.6 R-Register

- Das Refresh-Register enthält die jeweilige aktuelle 7-Bit-Refreshadresse.
- In jedem Befehlsholzyklus wird der Refresh in einer der insgesamt 128 mögliche Zeilen durchgeführt.
- Der Refresh wird während der Decodierphase durchgeführt (Takt 3 und 4); dadurch kein Zeitverlust! Nach jeder Refresh-Aktion wird das R-Register automatisch inkrementiert. Nach Erreichen des maximalen Wertes (= 127) wird das R-Register automatisch wieder auf 0 gesetzt (Ringzähler).
- Selbst bei Verwendung längstmöglicher Befehle bleibt der Refresh gesichert.
- Im HALT-Zustand führt die CPU automatisch NOP-Befehle aus und sichert somit den Refresh.

2.7 IX und IX-Register

IX und IY sind zwei 16-Bit-Register, die vornehmlich zur indizierten Adressierung verwendet werden.

IX bzw. IY enthält dabei eine (feste) Basisadresse, die durch einen, im **Befehl enthaltenen** Offset ergänzt wird.

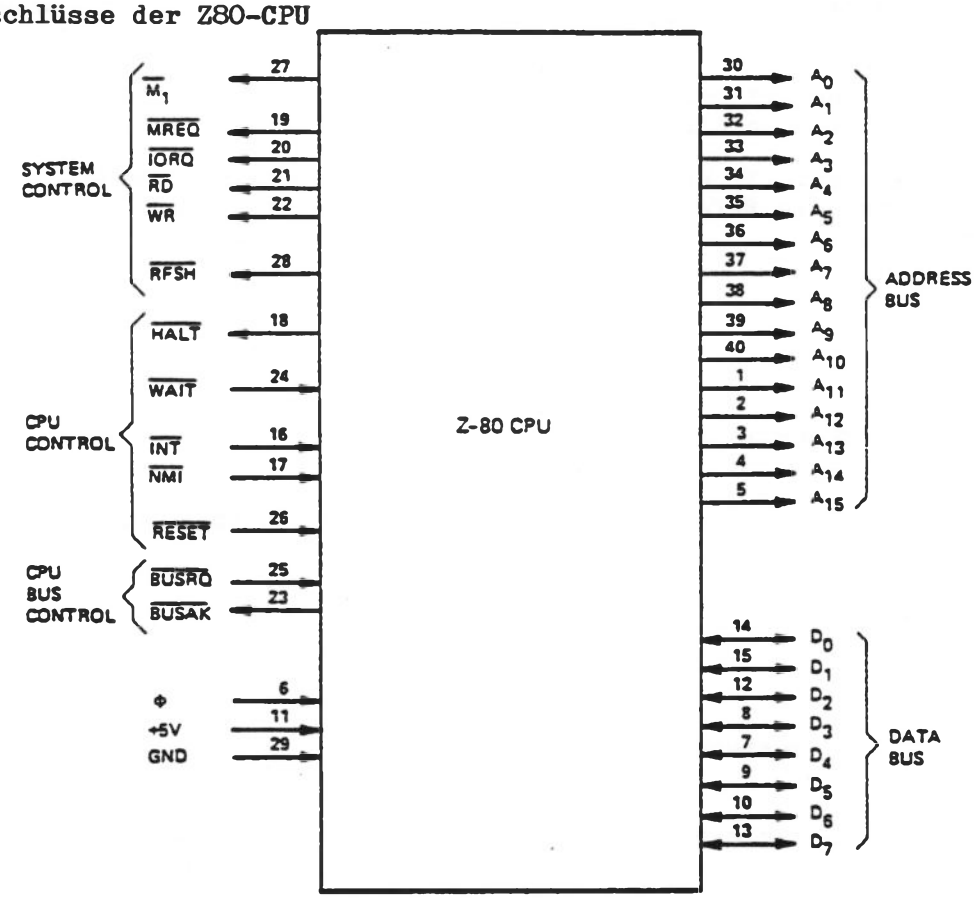
2.8 SP-Register

Das SP-Register enthält den aktuellen Stackpointer.

2.9 PC-Register

Das PC-Register enthält den aktuellen Befehlszählerstand.

2.10 Anschlüsse der Z80-CPU



a) **Adreßleitungen:** 16 Bit Breite, maximal ansprechbarer Speicherbereich = 64 kByte lineare Adressierung.
Unidirektional

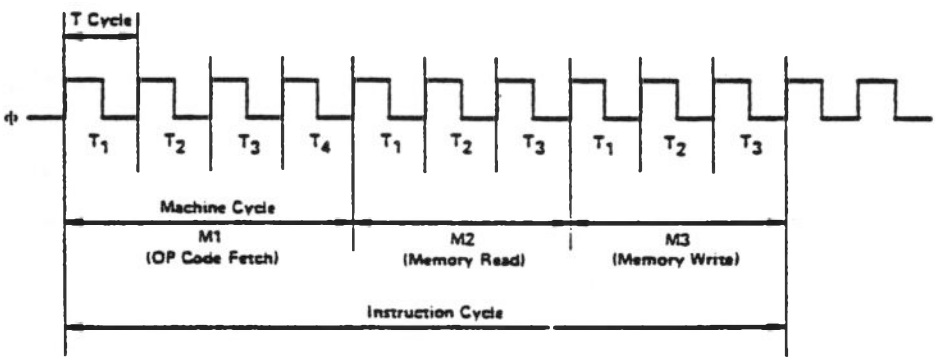
b) **Datenleitungen:** 8 Bit breite (8-Bit-Prozessor!)
Bidirektional

c) Steuerleitungen:

System-steuer-	M1	Während der Befehlsholphase (Takt 1 und 2) aktiv
	MREQ, IORQ	Unterscheidung von Speicher- und Kanal-Zugriffen
	RD, WR	Unterscheidung von Lese- und Schreibzugriffen
	RFSH	während der Befehlsholphase (Takt 3 und 4) aktiv
CPU-Steuer-Leitungen	HALT	Anzeige des HALT-Zustandes
	WAIT	Möglichkeit zur Verzögerung der Programmverarbeitung Der zweite Takt eines Befehls wird so lange wiederholt, so lange WAIT aktiv ist. ACHTUNG REFRESH!
	INT	Interruptleitung, wird durch Ein-/Ausgabekanäle aktiviert
	NMI	Interruptleitung des Nicht Maskierbaren Interrupts Ein Aktivieren dieser Leitung führt in **jedem** Fall zum sofortigen Einsprung in die entsprechende Interruptroutine. Ein Plus von mindestens 3 Taktlängen.
	RESET	Setzt den PC auf Wert 0000 und bringt das Steuerwerk in den Grundzustand. Dazu: Reg. R u I = 00 Inter.Mode = 0 Interrupt = disabled
CPU-BUS-Steuer-leitungen	BUSRQ	Möglichkeit, der CPU die Kontrolle über den BUS zu entziehen (z.B. DMA). CPU schaltet sich in den hochohmigen Zustand.
	BUSACK	Rückmeldung für den erfolgten Übergang in den hochohmigen Zustand

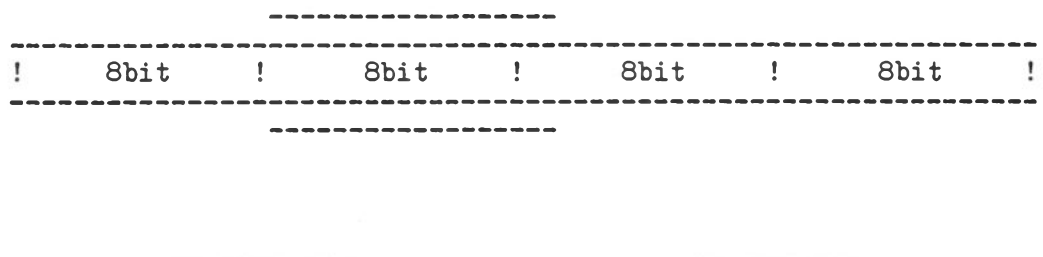
2.11 Zeitverhalten

	Z80-CPU	Z80A-CPU	Z80B-CPU
Taktfrequenz	2,5 MHz	4 MHz	6 MHz
Taktzeit	400 ns	250 ns	167 ns
Dauer M1 Zyklus	1.6 us	1 us	0.67 us
Dauer Mem. Read	1.2 us	0.75 us	0.5 us
Dauer Mem. Write	1.2 us	0.75 us	0.5 us



3. Einführung in den Z80-Befehlssatz

Befehlsaufbau des Mikroprozessors Z80



kann 1 oder 2 Byte umfassen.
Bei Befehlen mit zwei Anweisungsteilen treten zwei M1-Zyklen hintereinander auf.

kann kein, ein oder zwei Byte umfassen.

Folgende Befehle sind möglich :

1 Byte-Befehle	besteht nur aus einem Anweisungsteil	z.B. LD A,B SUB A etc.
2 Byte-Befehle	kann aus einem Doppel-anweisungsteil oder aus einem Einfachanweisungs-teil + einem Operanden bestehen	z.B. RLC B SRL H z.B. LD A,n ADD A,n
3 Byte-Befehle	kann aus einem Doppel-anweisungsteil + einem Operanden oder aus einem Einfachanweisungs-teil + zwei Operanden bestehen	z.B. ADD A,(IX+d) LD(IX+d),C z.B. CALL nn LD A,n
4 Byte-Befehle	besteht aus einem Doppelanweisungsteil + zwei Operanden	z.B. RLC (IX+d) BIT b,(IY+d)

4. Periphere Bausteine des Z80-Systems

Aufgaben:

- Ermöglichung der Kommunikation zwischen Computer (MC) und dessen Umgebung. Zeitliche Koordination der Verbindung.
  Umgebung = Maschine, Mensch

  Beispiele: Tastatur für Eingabe, Bildschirm oder Drucker für Ausgabe.

- Enlastung der CPU durch spezifische Hardware in den Peripheriebausteinen; dezentrale technische Intelligenz.

  Beispiel: Parallel- -----> Serienwandler im SIO

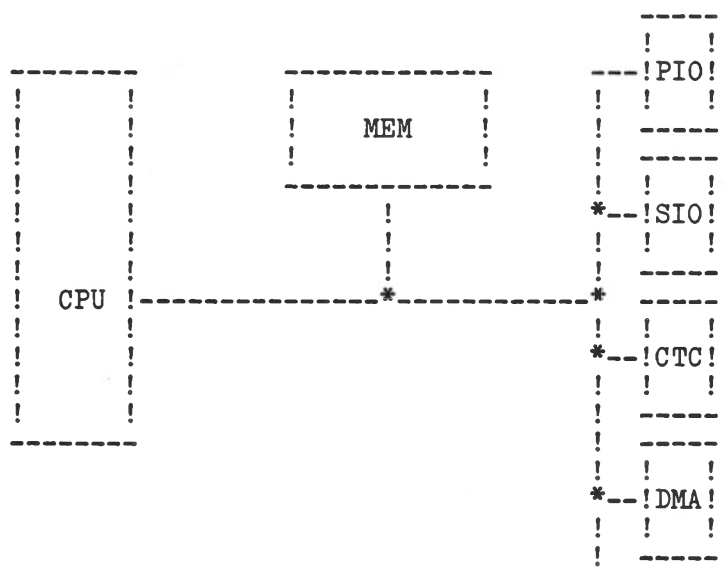
- Entlastung der CPU durch die Interruptfähigkeit der Peripheriebausteine

  (Unterschied: Polling ------ Interrupt)

- Entlastung der CPU durch Übernahme spezieller Aufgaben

  Beispiel: Ereigniszählung

4.1. Prinzipschaltung eines Z80-Systems



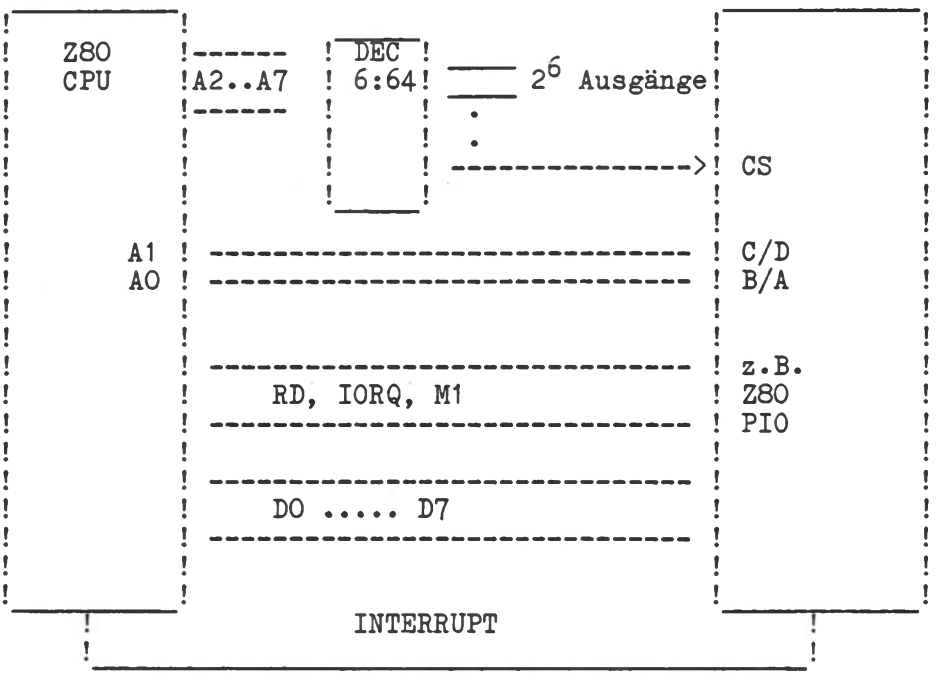
4.2 Allgemeines Funktionsprinzip der I/O Bausteine

- Unterschied: Peripheriebaustein <-----> konventionelle Hardware

  konventionelle Hardware = starre Hardware
  Peripheriebaustein = programmierte Hardware,
  d. h., kann für mehrere Aufgaben (z.B. INPUT, OUTPUT, ETC.) verwendet werden.

- Folge: a. Unterscheidung zwischen Anweisung und eigentlichen Daten notwendig!

  b. Programmierung muß vor Funktionsbeginn erfolgen! (Initialisierung)

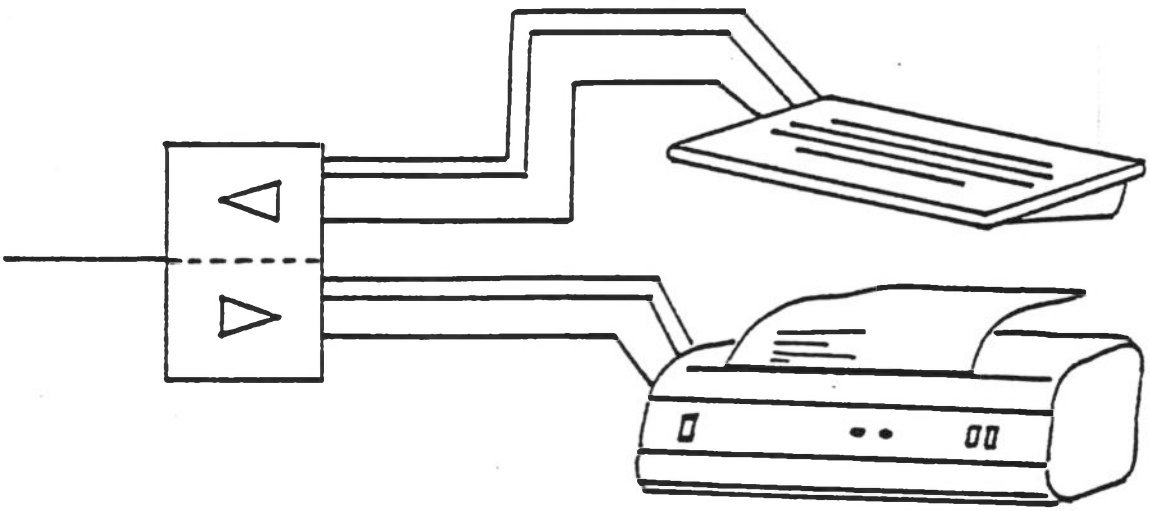


4.3 Z80-PIO (parallel I/O)

* 2 Kanäle A und B in einem Baustein (40 Pins)
* je 8 Datenleitungen und 2 Handshakeleitungen
* Betriebarten
  - Output-Mode
  - Input-Mode
  - bidirektionaler-Mode (nur Kanal A)
  - Bit-Mode

Beispiele:

Anschluß einer Tastatur (Input-Mode) oder eines Druckers (Output-Mode).

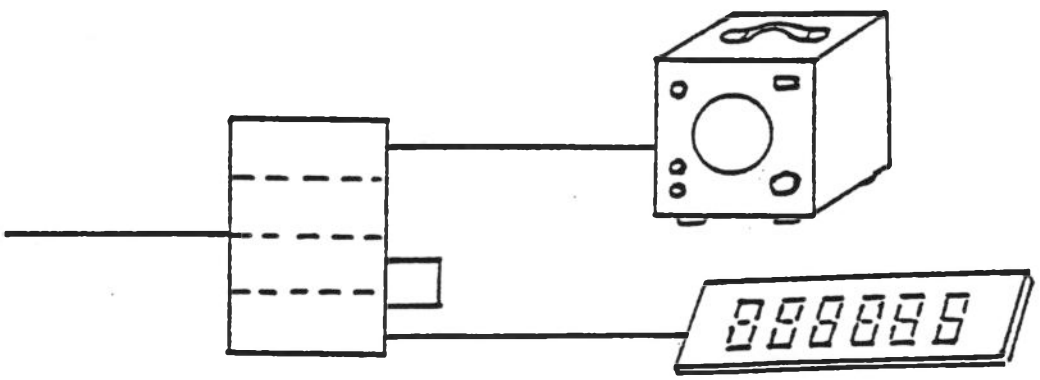


4.3 Z80-CTC (counter-timer-circuit)

* 4 Kanäle pro Baustein (28 Pin)
* je ein vorsetzbares Zählregister von 8 Bit, das durch den Systemtakt oder durch externe Impulse dekrementiert wird.
* je ein Zähl-/Start-Eingang, sowie ein Zerocount-Ausgang
* Betriebsarten:
  - Zähler-Mode
  - Zeitgeber-Mode

Beispiele:

Zählen extern erzeugter Pulse
Erzeugen eines Zeitrasters



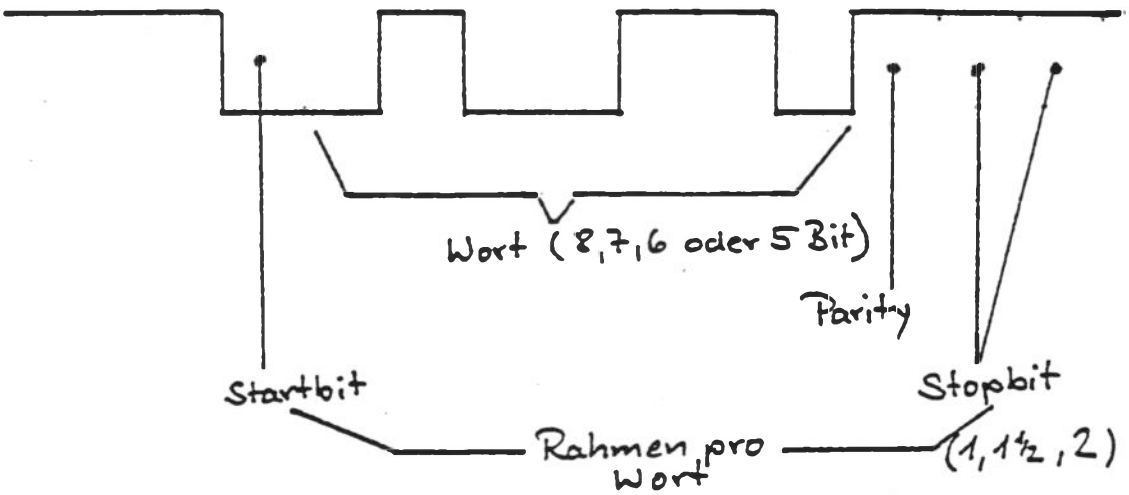
4.4 Z80-SIO

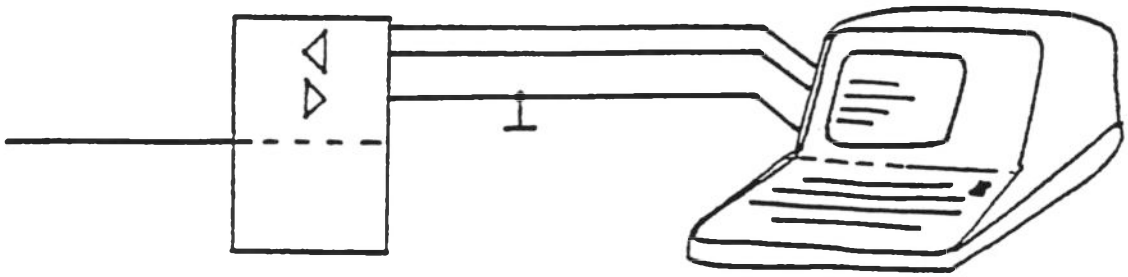
* 2 Kanäle (A und B) in einem Baustein (40 Pin)
* je zwei Übertragungsleitungen (full duplex), sowie 4 Handshakeleitungen
* automatische Wandlung seriell/parallel und parallel/seriell
* Betriebsarten:

  a.) asynchron (wortweise)

  Beispiel:

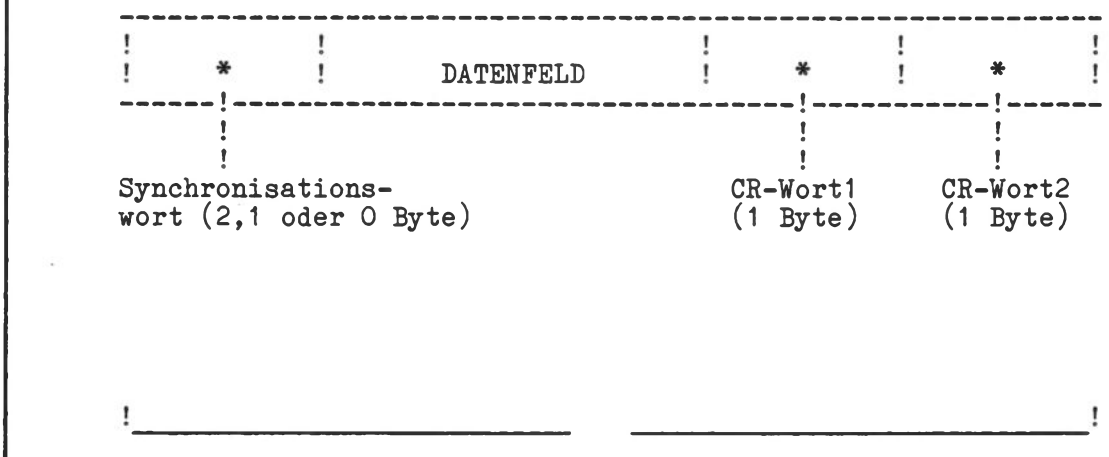
  Anschluß eines seriellen Datensichtgerätes



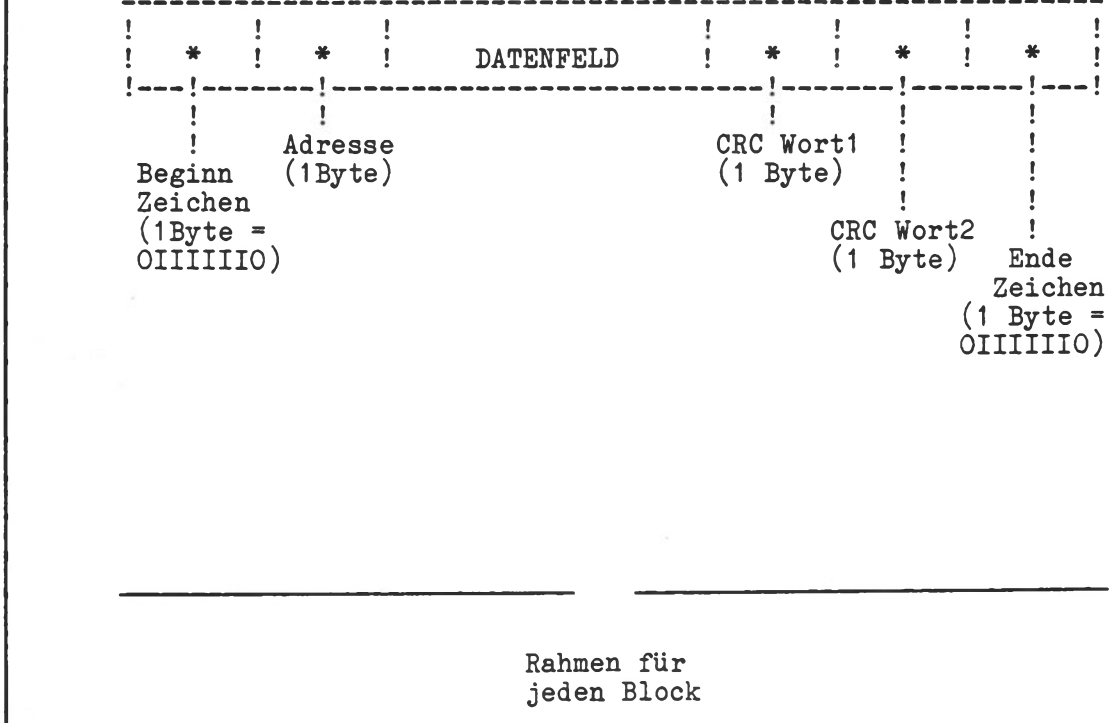


b.) **synchron** (blockweise)

monosync
bisync



SDLC
HDLC

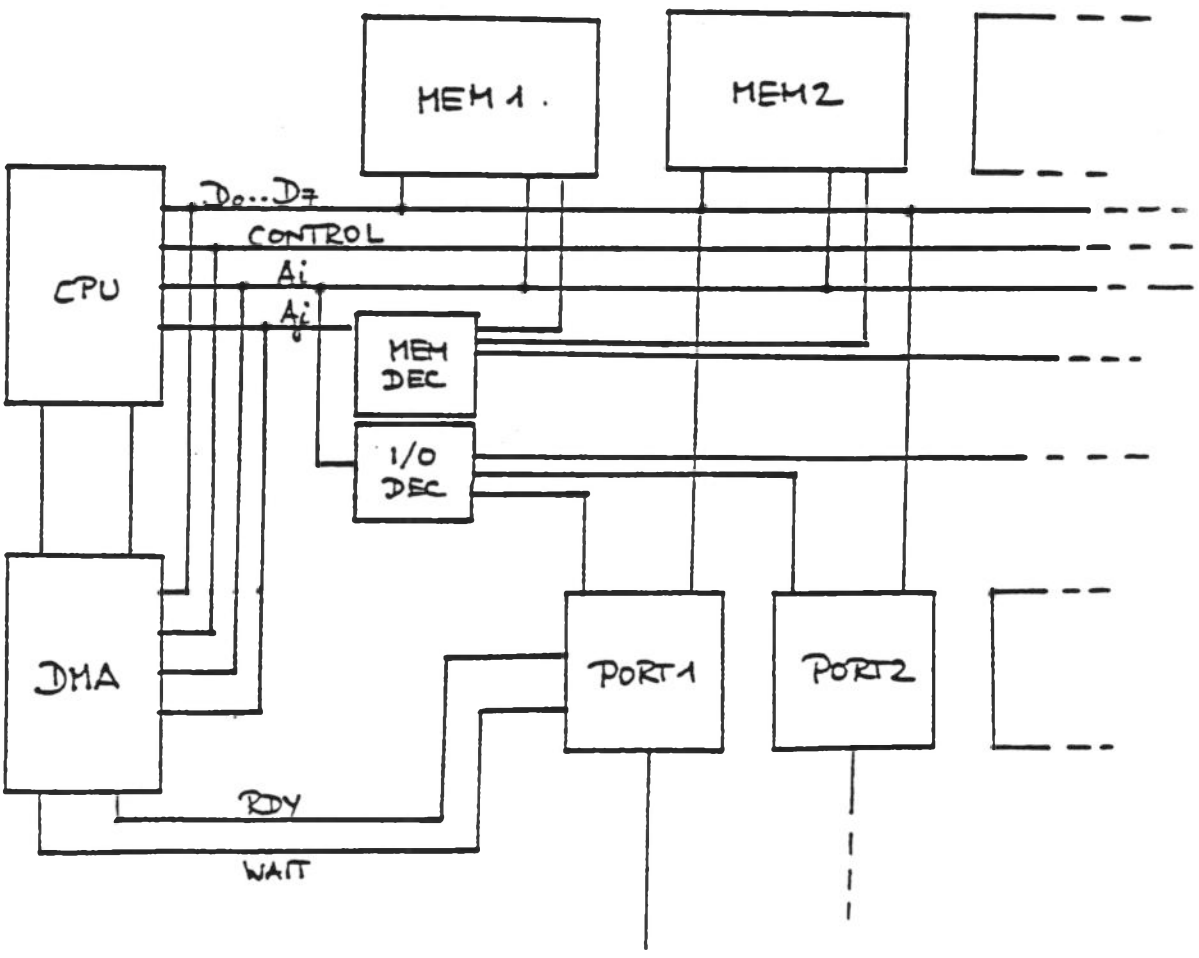


4.5 Z80-DMA (direct memory access)

* 1 Kanal pro Baustein (40 Pin)
* Anschlüsse zur Übernahme von Adreß-, Daten- und Steuerbusleitungen
* Handshake-Anschlüsse WAIT und RDY für evtl. Destination Port
* Funktionsweisen:
  - Übertragung
  - Suche
  - Übertagung und Suche
* Betriebsarten:
  - Byte at a time
  - Burst
  - Continuous

Beispiel:

Ausgabe Speicherblock auf Peripheriegerät (z.B. Magnetplatte)



5. Interrupttechnik in Z80-Systemen

5.1. Definition

Ein Interrupt ist ein asynchron zum laufenden Programm auftretendes Ergebnis.

Die reguläre Ausführung eines Interrupts bedeutet:

- Unterbrechen des laufenden Programms nach Abschluß des gerade in Ausführung begriffenen Befehls
- Retten des aktuellen Befehlszählerstandes auf den Stack
- Anlauf eines anderen Programms der Interrupt-Service-Routine (ISR).

5.2. Nicht-maskierbarer Interrupt

Der Z80 verfügt über einen Eingang NMI (activ low): Non-maskable Interrupt.

Dieser Interrupteingang kann nicht gesperrt (maskiert) werden und dient zur Meldung von Katastrophen, wie z.B. Netzausfall.

Die CPU unterbricht nach der Ausführung des gerade aktuellen Befehls die Abarbeitung des Programms, rettet den aktuellen PC auf den Stack, lädt PC mit der Adresse 066H und fährt mit dem dort hinterlegten Befehl fort.

Die Wirkung des NMI ist demnach fast identisch mit einem

```
CALL 066h
```

-Befehl.

5.3. Maskierbarer Interrupt

Der Eingang INT (activ low) wird für alle anderen Unterbrechungsanforderungen, herkommend z.B. von Peripherieelementen wie PIO, CTC, SIO, DMA, verwendet. Er kann gesperrt (maskiert) werden durch den Befehl DI (Disable Interrupt) oder durch den Hardware-RESET. Der Befehl EI (Enable Interrupt) hebt die Maskierung wieder auf.

Durch die Befehle IM0, IM1, IM2 wird die Betriebsart (Interrupt Mode) ausgewählt.

5.3.1. Mode 0

8080-kompatible Betriebsart. Die CPU sendet die Signale M1 und IORQ aus. Diese Signale werden in der Peripherie UND-verknüpft (M1 u IORQ = INTA, Interrupt Acknowledged).

Die CPU erwartet in diesem Zyklus ein Byte auf dem Datenbus, das sie als Befehl interpretieren wird. Üblicherweise sendet die Peripherie die Codierungen RST p (Restart auf Adresse p x 8, 1-Byte-Befehl) oder CALL (Aufruf eines Programms, 3-Byte-Befehl) aus.

Nach Erhalt des vollständigen Befehls wird dieser von der CPU ausgeführt.

Bei CALL erfolgen 1 INTA-Zyklus und 2 Memory-Read-Zyklen, auf die jedoch die Peripherie reagiern muß (zusätzliche Hardware!).

5.3.2. Mode 1

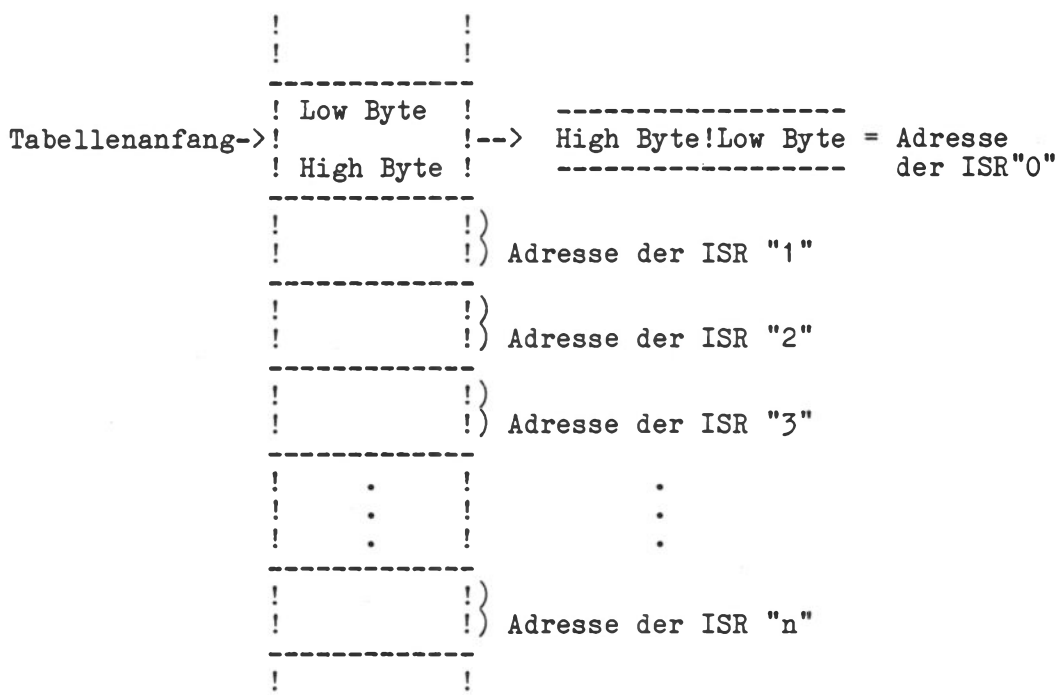
Einfachste Interruptart. Diese Betriebsart entspricht dem Vorgang beim nicht-maskierbaren Interrupt; es wird ein RST 038H ausgeführt.

5.3.3. Mode 2

Z80-Standard-Betriebsart. Es wird mit einem Vektor über eine Tabelle von ISR-Anfangsadresse verzweigt.

Im einzelnen:

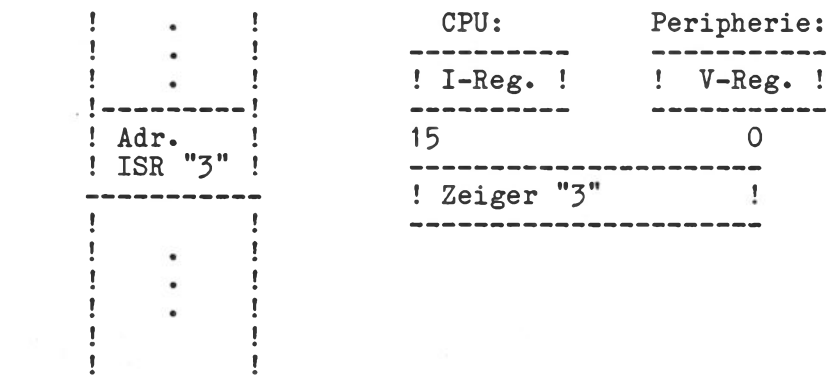
Der Programmierer hinterlegt im Speicher eine Tabelle mit den Einsprungstellen seiner Interrupt-Service-Routinen (ISR):



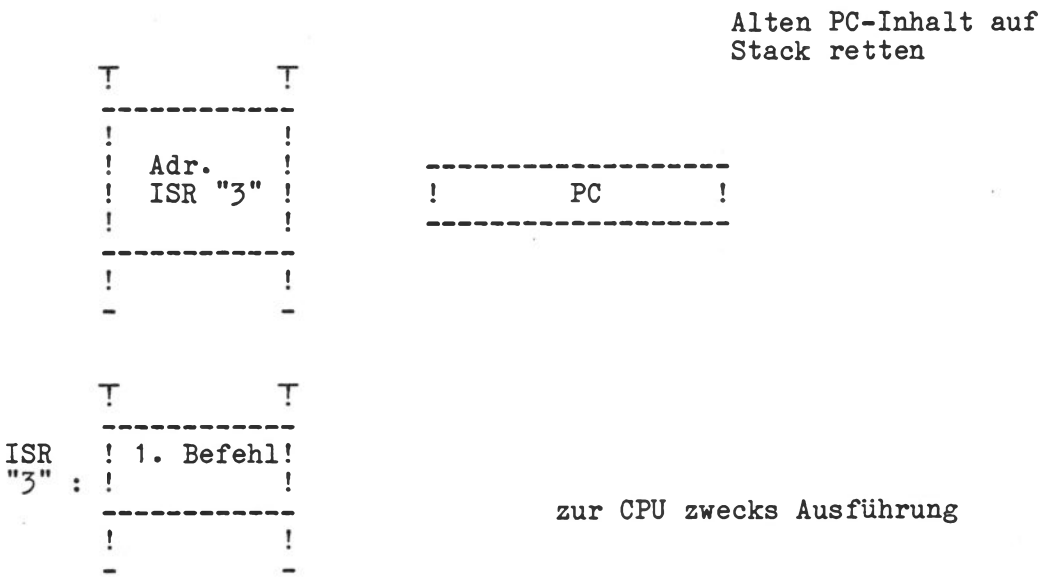
Desweiteren hinterlegt der Programmierer (im Rahmen der System initialisierung) das High-Byte der Adresse des Tabellenanfangs im CPU-Register I (Interrupt Vektor - High-Byte). Damit ist es möglich, die Tabelle in jeden beliebigen Speicherbereich zu legen, sie darf nur nicht über eine 256-Byte-Grenze hinausragen.

Zur Vervollständigung des Interruptvektors wird in den Interruptvektor-Registern der Z80-Peripheriebausteine ein Low-Byte der Einsprungstelle hinterlegt und zwar für jede Interruptmöglichkeit ein spezifischer Wert.

Das I-Register der CPU und ein Vektorregister eines Periphierie-bausteins werden zusammengesetzt und bilden dann einen Zeiger auf die Stelle, in der die Anfangsadresse der zugehörigen ISR steht:



Damit ist die Adresse der Stelle bestimmt, aus der bei einem Interrupt der PC mit der Anfangsadresse der ISR geladen werden kann:



Bei einem Interrupt, von einem Z80-Peripheriebaustein ausgelöst, (INT geht "low") arbeitet die CPU den laufenden Befehl noch bis zu seinem Ende ab und erkennt dann die Interrupt-Anforderung. Die CPU reagiert mit den Aussenden der Signale M1 und IORQ.

Der Peripheriebaustein (mit der höchsten Priorität), der einen Interrupt ausgesendet hat, erkennt M1 IORQ = INTA und reagiert mit dem Aussenden seines Vektors, des Low Bytes, auf den Datenbus.

Die CPU rettet den aktuellen PC, liest den Low-Byte-Vektor, setzt ihn mit dem Inhalt ihres I-Registers zusammen und transferiert die durch den Interruptvektor adressierte Zelle in den PC und maskiert den Interrupteingang INT.

Es schließt sich eine normale Befehlsholphase an: Die CPU arbeitet die ISR ab.

5.4. Verlassen einer ISR

Die Interrupt-Service-Routine arbeitet in der Art eines Unterprogramms und muß deswegen mittels RETURN verlassen werden. Dadurch wird der vorher gerettete PC vom Stack geholt und wieder als neuer PC-Inhalt installiert: die Verarbeitung des unterbrochenen Programms wird nahtlos fortgesetzt.

Neben der RETURN-Funktion ist jedoch noch der intern und zum Teil auch extern gespeicherte Interruptzustand rückzusetzten.

Da bei einem Non-Maskable-Interrupt (NMI) nur die CPU betroffen ist, genügt hier der Return über RETN. Neben der Stackfunktion stellt dieser den Maskierungszustand des INT Eingangs wieder auf den vor Eintritt von NMI vorhandenen Zustand (durch NMI war INT maskiert worden).

Bei einer über INT aufgerufenen ISR erfolgt der Return über RETI.

Die zusätzliche Funktion von RETI ist das Zurücksetzen des Peripherieelementes, dessen Interrupt gerade abgearbeitet worden ist: Der in der Priorität höchste interruptaktive Baustein (IEI = high, IEO = low) wartet auf das Erscheinen des Codes RETI (ED,40) während einer Befehlsholphase. Mit der Bedingung RETI M1 MREQ RD verläßt der Peripheriebaustein mit IEI = high und IEO = low den Interruptzustand und meldet seinerseits IEO = high.

Damit ist dieser Interrupt bedient und es wird zum unterbrochenen Programm verzweigt, das selbst wieder eine ISR niederer Priorität sein kann.

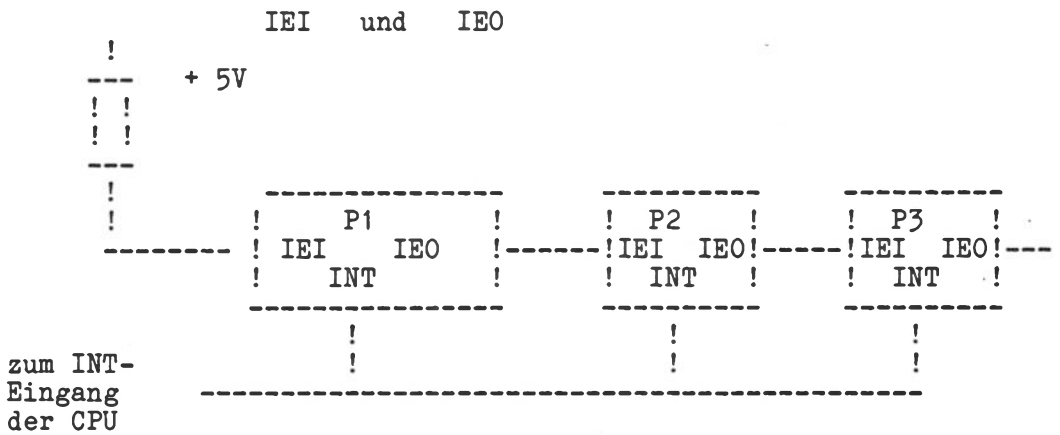
5.5. Prioritäten

Höchste Priorität in der CPU hat der Hardware-"Interrupt" BUS-REQUEST, der allerdings ein reiner Hardware-Vorgang ist.

Von den echten Interrupteingängen dominiert RESET über NMI und NMI über INT.

In der Peripherie werden Interrupts nach dem Daisy-Chain-Prinzip gekettet:

Die Bausteine sind bezüglich der Priorität in Serie geschaltet über die Signale

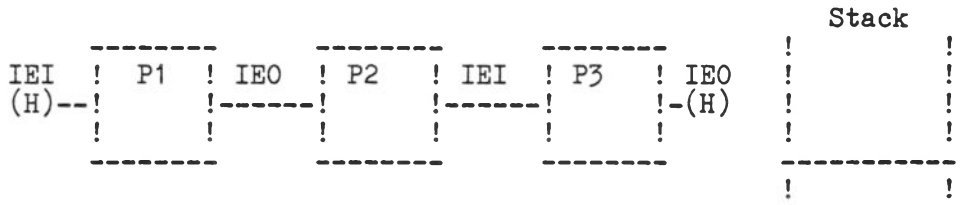


Peripheriebaustein P1 ist in der Prioritätskette über P2 und P3, P2 ist über P3.

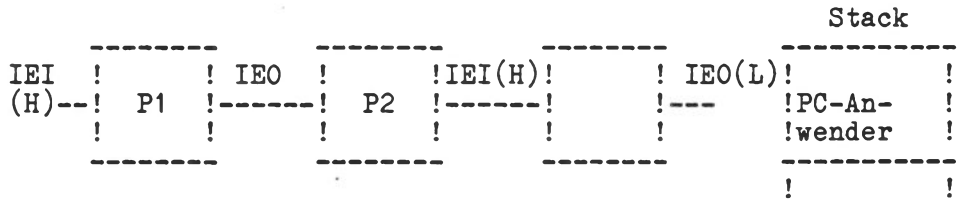
Beispiel siehe Bild "Abarbeitung von Interrupts"

5.6. Abarbeitung von Interrupts

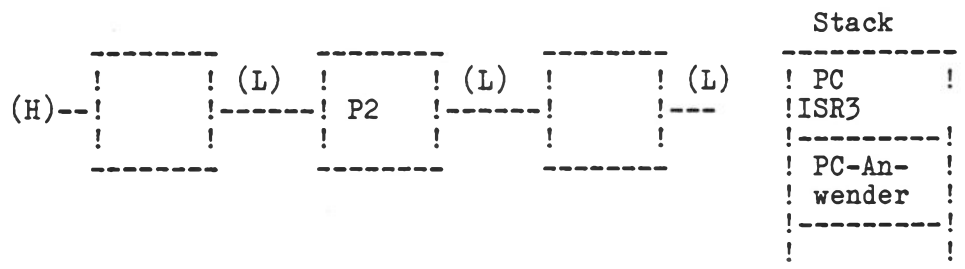
1. Kein Interrupt aktiv.
   Die CPU arbeitet im Programm ANWENDER



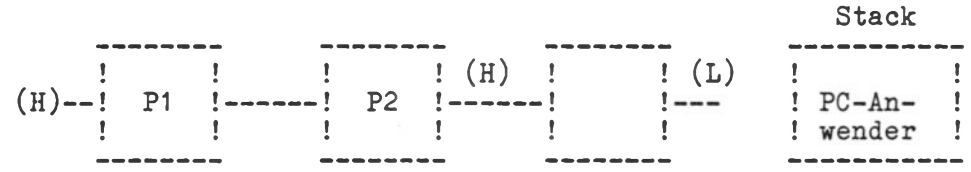
2. P3 hat Interrupt gemeldet und seinen Vektor ausgesandt. Die CPU hat ANWENDER verlassen und arbeitet in der ISR 3



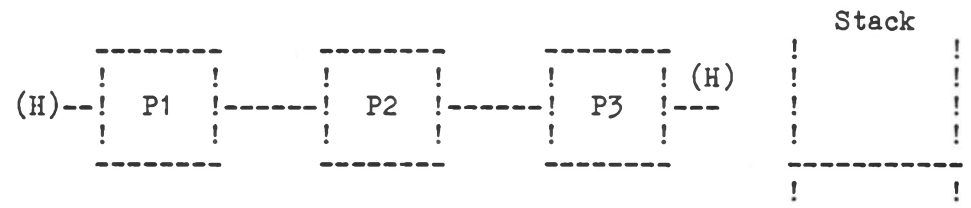
3. P1 hat Interrupt gemeldet und seinen Vektor ausgesandt. Die CPU hat ISR 3 verlassen und arbeitet in ISR 1.



4. P1 hat RETI erkannt. Die CPU hat ISR 3 wieder aufgenommen.



5. P3 hat RETI erkannt. Die CPU hat ANWENDER wieder aufgenommen



6. Vorgehensweise bei der Programmerstellung

6.1 Hauptpunkte der Programmentwicklung

Man muß sich bei einer Programmentwicklung darüber klar sein, daß es sich dabei nicht nur um das "Notieren" handelt, d.h. um das Niederschreiben des Programmes auf Papier. Vielmehr besteht eine solche Entwicklung aus folgenden Hauptkomponenten:

- Problemanalyse
- Erarbeitung eines Lösungsvorschlages (Ablaufplan)
- Codieren in der gewünschten Sprache
- Eingeben und Übersetzen (Entwicklungssystem)
- Testen (Entwicklungssystem)

6.2 Problemanalyse

Einer der wichtigsten Punkte ist die Problemanalyse, mit der ein vorliegendes Problem bis in die Einzelheiten ausgeleuchtet wird. Versäumnisse die hier durch mangelnde Sorgfalt oder Vorausplanung entstehen, sind zu einem späteren Zeitpunkt in vielen Fällen nur noch unter unvertretbarem Aufwand korrigierbar.

Speziell die fehlende Betrachtung von Randwertbedingungen kann einen Lösungsvorschlag im nachhinein als ungeeignet und damit als unbrauchbar qualifizieren. Bei einer Entwicklung sollte deshalb diesem Stadium besondere Beachtung geschenkt werden!

6.3 Erarbeitung eines Lösungsweges

Kaum weniger wichtig ist die "professionelle" Erarbeitung eines Lösungsweges. Die Erfahrung zeigt jedoch, daß in vielen Fällen dieser Punkt schmählich vernachlässigt wurde.

Verständlich wird dies aus dem Drang, möglichst bald "Vorzeigbares" in Form von möglichst vielen Statements (= Befehlszeilen) zu produzieren. Logische Fehler, uneffiziente Programme und ein Mangel an Übersicht sind die nahezu unausweichliche Folge.

Bedenkt man, daß bei einer Programmentwicklung bis zu 50 % der Kosten beim Testen und korrigieren entstehen, gewinnt gerade diese Entwicklungsphase (zusammen mit der Problemanalyse) an Bedeutung. Die Erarbeitung eines Lösungsweges sollte unbedingt mit Hilfe von Flußdiagrammen bzw. Struktogrammen bei strukturierter Programmierung durchgeführt werden. Letzteres sind Programmablaufpläne, die nicht nur den Programmfluß festlegen, sondern auch seinen Aufbau (Struktur).

Eine Erleichterung ergibt sich weiterhin durch schrittweise Verfeinerung des Lösungsentwurfs: Man konzentriert sich zunächst auf die "große Linie" oder den "Rahmen", dann auf die Details.

Von Fall zu Fall, und in Abhängigkeit des Problems und des Programmierens kann die Detaillierung so weit gehen, daß ein direktes eins-zu-eins-mäßiges Übertragen des Programmflußplanes in die entsprechenden Befehle möglich wird.

Wesentliche Vorteile der geschilderten Methode sind:

- Verringerung des Zeitbedarfs zur Kodierung auf ein Minimum
- Große Übersichtlichkeit des Programmes
- Größtmögliche Vermeidung von logischen Fehlern
- Exakte Dokumentation
- Optimale Nachprüfbarkeit

6.4 Kodierung

Die Kodierung des Lösungsweges, d.h. die Umsetzung des Programmflußplanes in entsprechende Befehle sollte auf die Verwendung optimaler Befehle bzw. Befehlsfolgen ausgerichtet sein. Der Erfolg dieser Bemühung ist ein Programm, das bezüglich Speicherbedarf und Laufzeit ein Optimum darstellt. Eine möglichst weitgehende Detaillieferung des Programmablaufplanes erleichtert dabei die Kodierung, da zusätzliche Denkprozesse zur Aufstellung von Teilabläufen entfallen.

Zusammenfassend kann festgehalten werden, daß im Hinblick auf die Verringerung der Software-Erstellungs-Kosten der Problemanalyse, sowie der Erarbeitung des Lösungsweges größte Bedeutung zugemessen werden sollte. Investitionen in diese Entwicklungsphasen ergeben ein Mehrfaches an Zeitgewinn durch Verkürzung der Testphase.

Beispiel: - Differenzwertanzeige

Geplante Vorgehensweise:

1. Problemanalyse	a) Problem definieren
	b) Randbedingungen festlegen
2. Erarbeitung Lösungsvorschlag	a) Entwurf eines Rahmen-Ablauf-Planes
	b) Herauslösen von Einzel-problemen
	c) Aufstellung von Flußplänen pro Teilaufgabe
	d) Verfeinerung der Ablaufpläne
3. Codieren (in Assembler)	a) Umsetzung in Assembler-befehle
	b) Bildung eines Gesamt-programmes

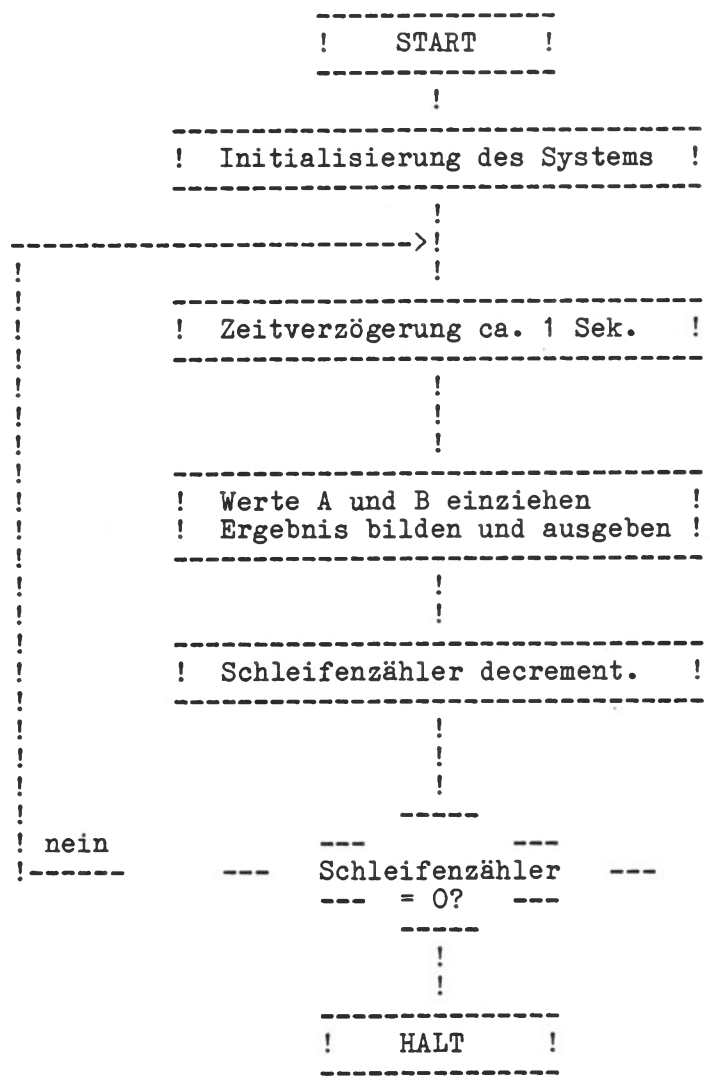
zu 1a Problemstellung:

Zwei Meßwerte A und B, die im binären (8bit) Format statisch vorliegen, sollen über zwei parallele I/O-Schnittstellen abgefragt und voneinander subtrahiert werden (C=A-B). Das Ergebnis wird anschließend über eine, das Vorzeichen über eine weitere parallele I/O-Schnittstelle ausgegeben.

zu 1b Randbedingungen:

- Wert A Eingabe über PORT 1 (Adr. 08/0A)
- Wert B Eingabe über PORT 2 (Adr. 09/0B)
- Wert C Ausgabe über PORT 3 (Adr. 2C/2E)
- Vorzeichen Ausg. über PORT 4 (Adr. 2D7/F)
- 10malige Wiederholung des Vergleich-Vorgangs
- Abstand von Messung zu Messung ca. 1 Sek.
- Erste Messung ca. 1 Sek. nach dem Einschalten
- Start durch RESET
- Ende durch Auflaufen auf HALT
- Anzeige des letzten Ergebnisses bis zum nächsten START
- Stackbereich beginnend bei 2000H

zu 2a **Gesamtablaufplan:**

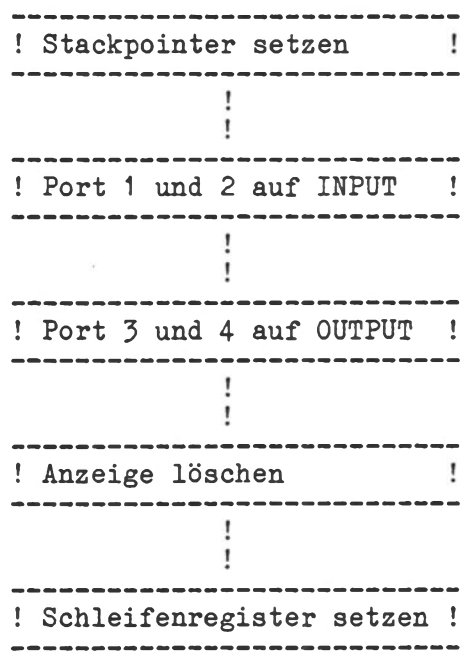


zu 2B Herauslösung von Einzelproblemen aus dem Rahmenplan:

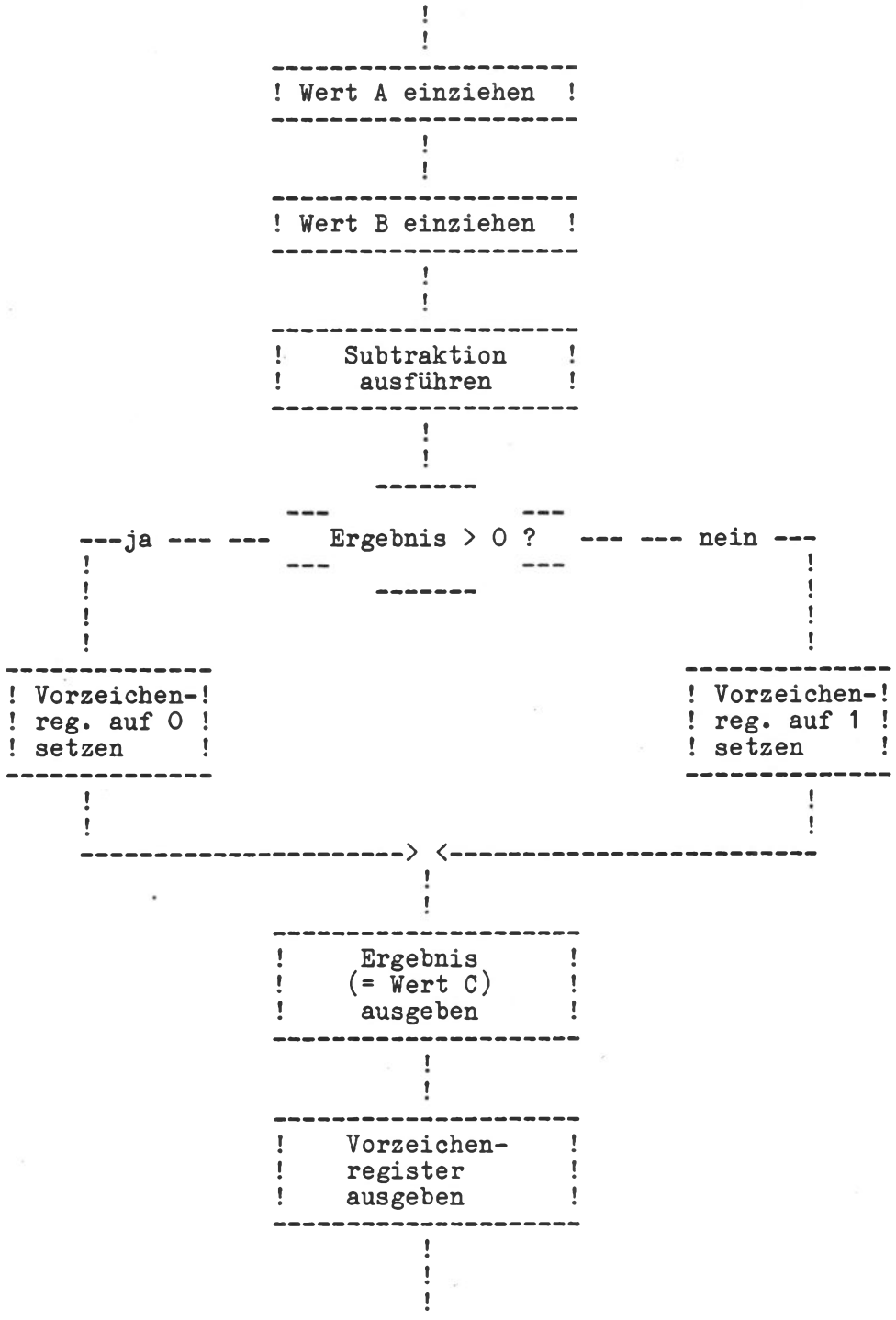
- Initialisierung
- Zeitverzögerung
- Wertmanipulation

zu 2C Aufstellung von Ablaufplänen pro Teilbereich:

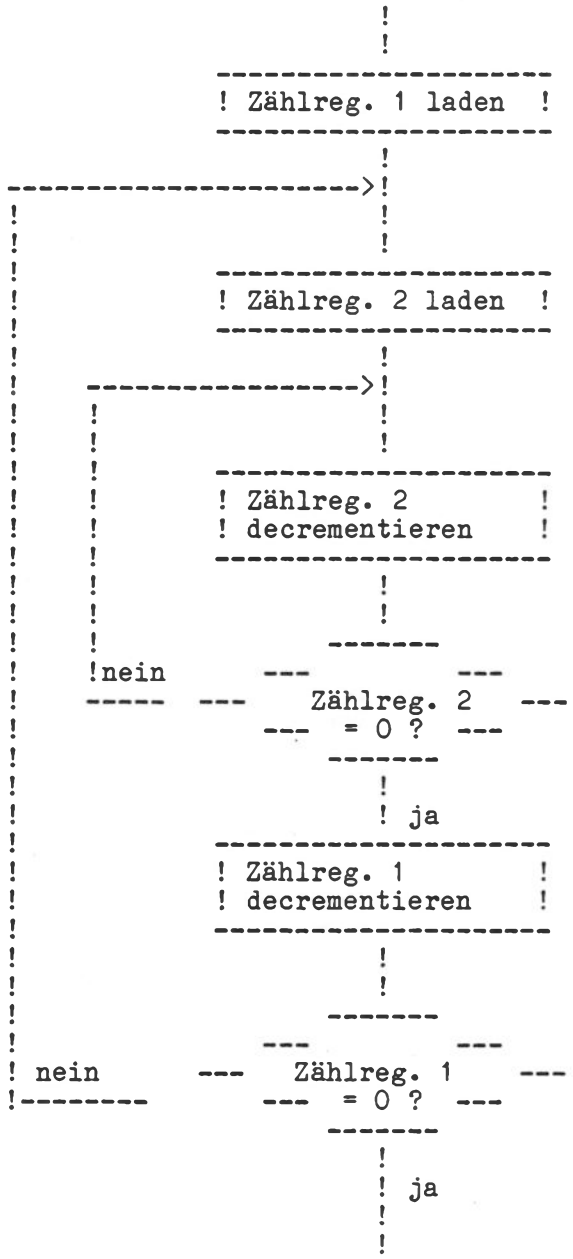
- Initialisierung



- Wertmanipulation



- Zeitverzögerung



zu 3a Umsetzung in Assemblerbefehle:

- Initialisierung

```
LD SP, 2000H                Stackpointer laden
LD A, 7FH
OUT (0AH),A                 Port 1 auf Input
OUT (0BH),A                 Port 2 auf Input
LD A, 0FH
OUT (2EH),A                 Port 3 auf Output
OUT (2FH),A                 Port 4 auf Output
LD A, 0
OUT (2CH), A                Anzeige löschen
OUT (2DH), A                Anzeige löschen
```

- Wertmanipulation

```
        IN A, (09H)         Wert B lesen
        LD B,A              Wert B ins Reg. B
        IN A, (08H)         Wert A lesen
        SUB B               Wert A - Wert B = Wert C
        JRNC, PLUS          wenn Ergebnis > 0
        LD D,1              wenn Ergebnis < 0
                                 Vorzeichen = negativ
        JR WEITER
PLUS:   LD, D,0             Vorzeichen = positiv
WEITER: OUT (2CH)?A         Ausgabe Ergebnis
        LD A,D
        OUT (2DH),A         Ausgabe Vorzeichen
```

- Zeitverzögerung

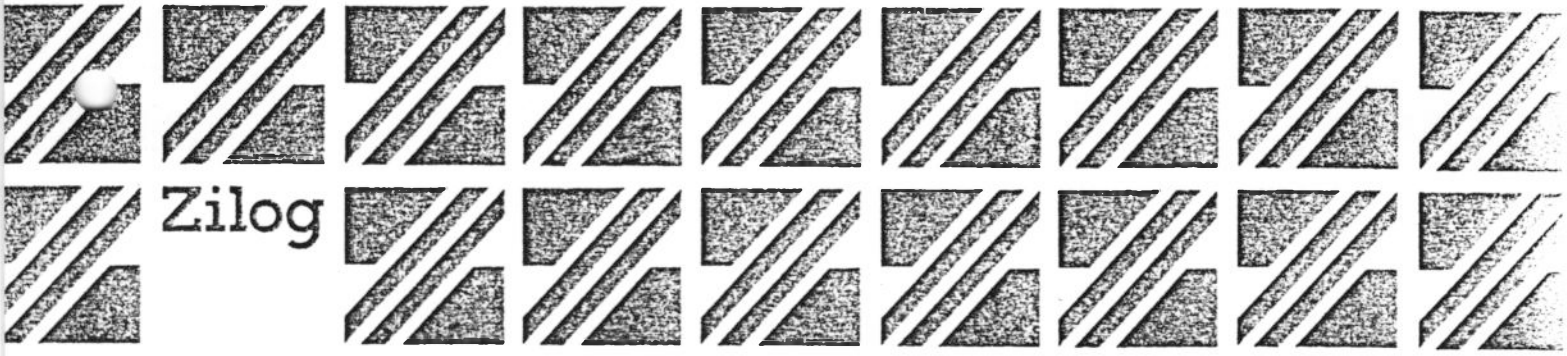
```
        LD B,0FFH           Zählregister 1 laden
MARKE1: PUSH BC             Zählregister 1 retten
        LD B,80H            Zählregister 2 laden
MARKE2: DJNZ MARKE2         Zählregister 2 decrementieren
                            falls = 0 zurück zu MARKE2
        POP BC              sonst Zählreg. 1 zurückholen
        DJNZ MARKE1         Zählreg. 1 decrementieren,
                            falls = 0 zurück zu MARKE1
```

zu 3b Zusammensassung:

```
                LD SP,2000H
                LD A,7FH
                OUT (0AH),A
                OUT (0BH),A
                LD A,0FH
                OUT (2FH),A
                OUT (2FH),A
                LD A,0
                OUT (2CH),A
                OUT (2DH),A
                LD E,10

        NEU:
                LD B,0FFH

        MARKE1:
                PUSH BC
                LD B,80H

        MARKE2:
                DJNZ MARKE2
                POP BC
                DJNZ MARKE1
                IN A,(09H)
                LD B,A
                IN A,(08H)
                SUB B
                IRNC PLUS
                LD D,1
                JR WEITER

        PLUS:
                LD D,0
        WEITER: OUT (2CH),A
                LD A,D
                OUT (2DH),A
                DEC E
                JP NZ,NEU
                HALT
```

Z80™-CPU
Z80A™-CPU

Technical Manual



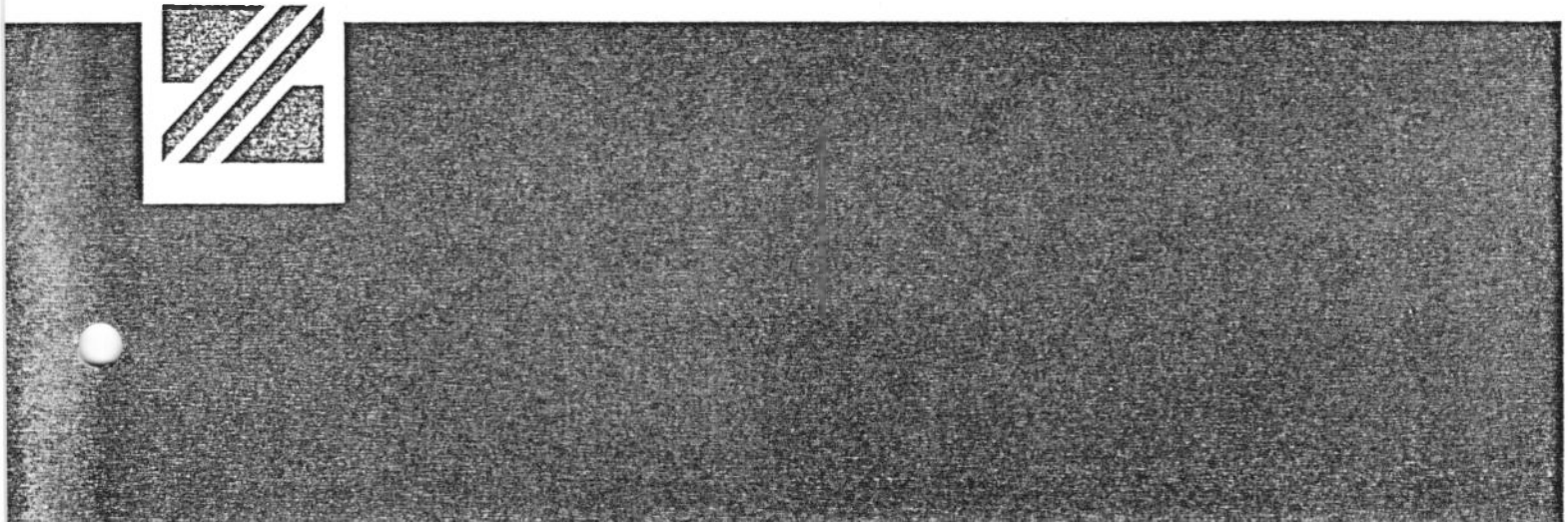


TABLE OF CONTENTS

Chapter		Page
1.0	Introduction	1
2.0	Z80-CPU Architecture	3
3.0	Z80-CPU Pin Description	7
4.0	CPU Timing	11
5.0	Z80-CPU Instruction Set	19
6.0	Flags	39
7.0	Summary of OP Codes and Execution Times	43
8.0	Interrupt Response	55
9.0	Hardware Implementation Examples	59
10.0	Software Implementation Examples	63
11.0	Electrical Specifications	69
12.0	Z80-CPU Instruction Set Summary	73

1.0 INTRODUCTION

The term "microcomputer" has been used to describe virtually every type of small computing device designed within the last few years. This term has been applied to everything from simple "microprogrammed" controllers constructed out of TTL MSI up to low end minicomputers with a portion of the CPU constructed out of TTL LSI "bit slices." However, the major impact of the LSI technology within the last few years has been with MOS LSI. With this technology, it is possible to fabricate complete and very powerful computer systems with only a few MOS LSI components.

The Zilog Z-80 family of components is a significant advancement in the state-of-the art of microcomputers. These components can be configured with any type of standard semiconductor memory to generate computer systems with an extremely wide range of capabilities. For example, as few as two LSI circuits and three standard TTL MSI packages can be combined to form a simple controller. With additional memory and I/O devices a computer can be constructed with capabilities that only a minicomputer could previously deliver. This wide range of computational power allows standard modules to be constructed by a user that can satisfy the requirements of an extremely wide range of applications.

The major reason for MOS LSI domination of the microcomputer market is the low cost of these few LSI components. For example, MOS LSI microcomputers have already replaced TTL logic in such applications as terminal controllers, peripheral device controllers, traffic signal controllers, point of sale terminals, intelligent terminals and test systems. In fact the MOS LSI microcomputer is finding its way into almost every product that now uses electronics and it is even replacing many mechanical systems such as weight scales and automobile controls.

The MOS LSI microcomputer market is already well established and new products using them are being developed at an extraordinary rate. The Zilog Z-80 component set has been designed to fit into this market through the following factors:

1. The Z-80 is fully software compatible with the popular 8080A CPU offered from several sources. Existing designs can be easily converted to include the Z-80 as a superior alternative.
2. The Z-80 component set is superior in both software and hardware capabilities to any other microcomputer system on the market. These capabilities provide the user with significantly lower hardware and software development costs while also allowing him to offer additional features in his system.
3. For increased throughput the Z80A operating at a 4 MHZ clock rate offers the user significant speed advantages over competitive products.
4. A complete product line including full software support with strong emphasis on high level languages and a disk-based development system with advanced real-time debug capabilities is offered to enable the user to easily develop new products.

Microcomputer systems are extremely simple to construct using Z-80 components. Any such system consists of three parts:

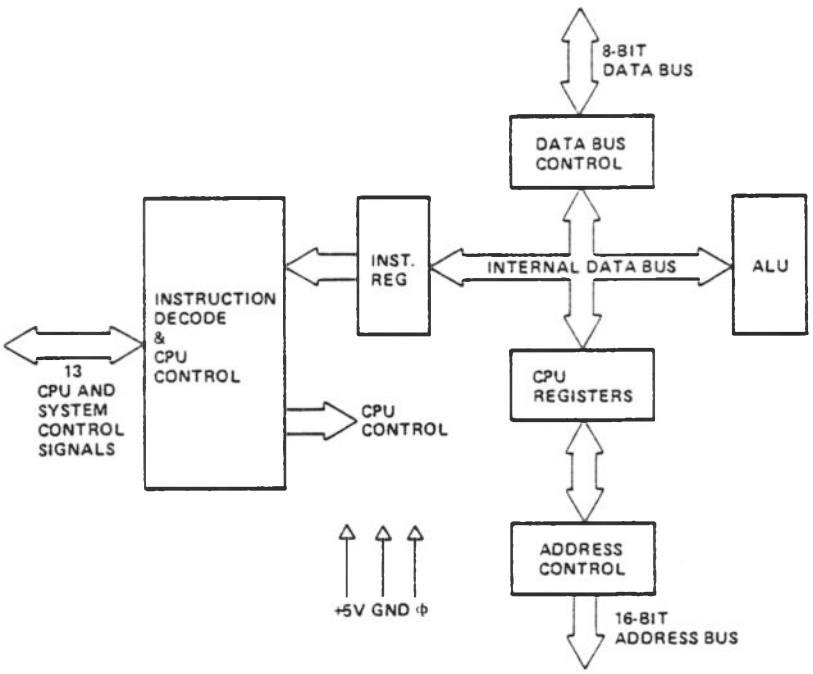
1. CPU (Central Processing Unit)
2. Memory
3. Interface Circuits to peripheral devices

The CPU is the heart of the system. Its function is to obtain instructions from the memory and perform the desired operations. The memory is used to contain instructions and in most cases data that is to be processed. For example, a typical instruction sequence may be to read data from a specific peripheral device, store it in a location in memory, check the parity and write it out to another peripheral device. Note that the Zilog component set includes the CPU and various general purpose I/O device controllers, while a wide range of memory devices may be used from any source. Thus, all required components can be connected together in a very simple manner with virtually no other external logic. The user's effort then becomes primarily one of software development. That is, the user can concentrate on describing his problem and translating it into a series of instructions that can be loaded into the microcomputer memory. Zilog is dedicated to making this step of software generation as simple as possible. A good example of this is our

assembly language in which a simple mnemonic is used to represent every instruction that the CPU can perform. This language is self documenting in such a way that from the mnemonic the user can understand exactly what the instruction is doing without constantly checking back to a complex cross listing.

2.0 Z-80 CPU ARCHITECTURE

A block diagram of the internal architecture of the Z-80 CPU is shown in figure 2.0-1. The diagram shows all of the major elements in the CPU and it should be referred to throughout the following description.



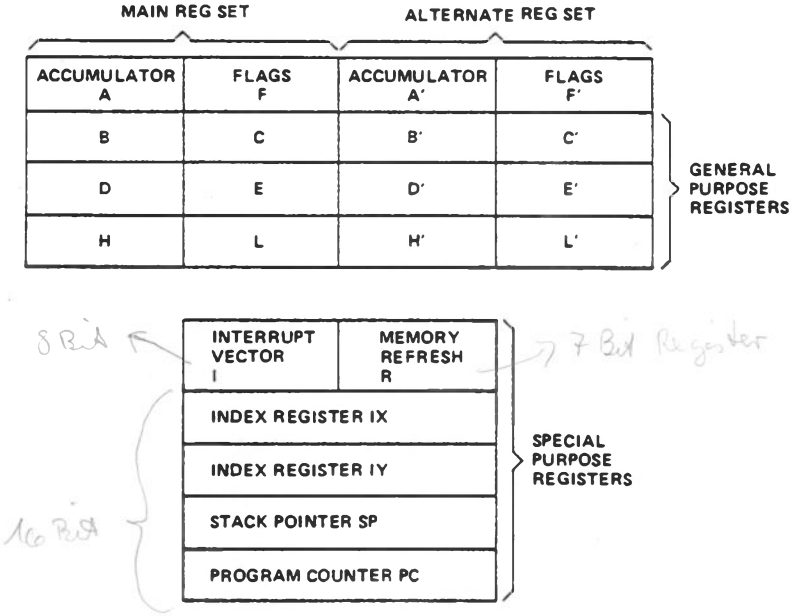
Z-80 CPU BLOCK DIAGRAM
FIGURE 2.0-1

2.1 CPU REGISTERS

The Z-80 CPU contains 208 bits of R/W memory that are accessible to the programmer. Figure 2.0-2 illustrates how this memory is configured into eighteen 8-bit registers and four 16-bit registers. All Z-80 registers are implemented using static RAM. The registers include two sets of six general purpose registers that may be used individually as 8-bit registers or in pairs as 16-bit registers. There are also two sets of accumulator and flag registers.

Special Purpose Registers

1. **Program Counter (PC).** The program counter holds the 16-bit address of the current instruction being fetched from memory. The PC is automatically incremented after its contents have been transferred to the address lines. When a program jump occurs the new value is automatically placed in the PC, overriding the incrementer.
2. **Stack Pointer (SP).** The stack pointer holds the 16-bit address of the current top of a stack located anywhere in external system RAM memory. The external stack memory is organized as a last-in first-out (LIFO) file. Data can be pushed onto the stack from specific CPU registers or popped off of the stack into specific CPU registers through the execution of PUSH and POP instructions. The data popped from the stack is always the last data pushed onto it. The stack allows simple implementation of multiple level interrupts, unlimited subroutine nesting and simplification of many types of data manipulation.



Z-80 CPU REGISTER CONFIGURATION
FIGURE 2.0-2

3. **Two Index Registers (IX & IY).** The two independent index registers hold a 16-bit base address that is used in indexed addressing modes. In this mode, an index register is used as a base to point to a region in memory from which data is to be stored or retrieved. An additional byte is included in indexed instructions to specify a displacement from this base. This displacement is specified as a two's complement signed integer. This mode of addressing greatly simplifies many types of programs, especially where tables of data are used.

4. **Interrupt Page Address Register (I).** The Z-80 CPU can be operated in a mode where an indirect call to any memory location can be achieved in response to an interrupt. The I Register is used for this purpose to store the high order 8-bits of the indirect address while the interrupting device provides the lower 8-bits of the address. This feature allows interrupt routines to be dynamically located anywhere in memory with absolute minimal access time to the routine.

5. **Memory Refresh Register (R).** The Z-80 CPU contains a memory refresh counter to enable dynamic memories to be used with the same ease as static memories. This 7-bit register is automatically incremented after each instruction fetch. The data in the refresh counter is sent out on the lower portion of the address bus along with a refresh control signal while the CPU is decoding and executing the fetched instruction. This mode of refresh is totally transparent to the programmer and does not slow down the CPU operation. The programmer can load the R register for testing purposes, but this register is normally not used by the programmer.

Accumulator and Flag Registers

The CPU includes two independent 8-bit accumulators and associated 8-bit flag registers. The accumulator holds the results of 8-bit arithmetic or logical operations while the flag register indicates specific conditions for 8 or 16-bit operations, such as indicating whether or not the result of an operation is equal to zero. The programmer selects the accumulator and flag pair that he wishes to work with with a single exchange instruction so that he may easily work with either pair.

General Purpose Registers

There are two matched sets of general purpose registers, each set containing six 8-bit registers that may be used individually as 8-bit registers or as 16-bit register pairs by the programmer. One set is called BC, DE and HL while the complementary set is called BC', DE' and HL'. At any one time the programmer can select either set of registers to work with through a single exchange command for the entire set. In systems where fast interrupt response is required, one set of general purpose registers and an accumulator/flag register may be reserved for handling this very fast routine. Only a simple exchange commands need be executed to go between the routines. This greatly reduces interrupt service time by eliminating the requirement for saving and retrieving register contents in the external stack during interrupt or subroutine processing. These general purpose registers are used for a wide range of applications by the programmer. They also simplify programming, especially in ROM based systems where little external read/write memory is available.

2.2 ARITHMETIC & LOGIC UNIT (ALU)

The 8-bit arithmetic and logical instructions of the CPU are executed in the ALU. Internally the ALU communicates with the registers and the external data bus on the internal data bus. The type of functions performed by the ALU include:

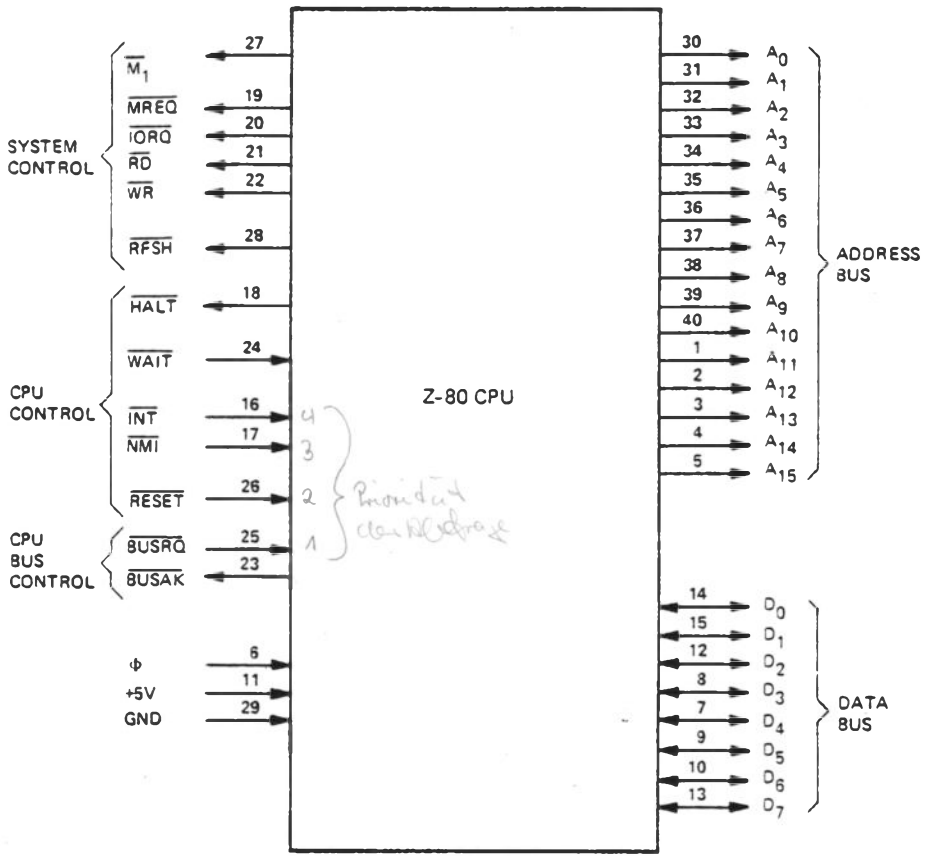
Add	Left or right shifts or rotates (arithmetic and logical)
Subtract	Increment
Logical AND	Decrement
Logical OR	Set bit
Logical Exclusive OR	Reset bit
Compare	Test bit

2.3 INSTRUCTION REGISTER AND CPU CONTROL

As each instruction is fetched from memory, it is placed in the instruction register and decoded. The control sections performs this function and then generates and supplies all of the control signals necessary to read or write data from or to the registers, control the ALU and provide all required external control signals.

3.0 Z-80 CPU PIN DESCRIPTION

The Z-80 CPU is packaged in an industry standard 40 pin Dual In-Line Package. The I/O pins are shown in figure 3.0-1 and the function of each is described below.



Z-80 PIN CONFIGURATION
FIGURE 3.0-1

A_0-A_{15}
(Address Bus)

Tri-state output, active high. A_0-A_{15} constitute a 16-bit address bus. The address bus provides the address for memory (up to 64K bytes) data exchanges and for I/O device data exchanges. I/O addressing uses the 8 lower address bits to allow the user to directly select up to 256 input or 256 output ports. A_0 is the least significant address bit. During refresh time, the lower 7 bits contain a valid refresh address.

D_0-D_7
(Data Bus)

Tri-state input/output, active high. D_0-D_7 constitute an 8-bit bidirectional data bus. The data bus is used for data exchanges with memory and I/O devices.

$\overline{M_1}$
(Machine Cycle one)

Output, active low. $\overline{M_1}$ indicates that the current machine cycle is the OP code fetch cycle of an instruction execution. Note that during execution of 2-byte op-codes, $\overline{M1}$ is generated as each op code byte is fetched. These two byte op-codes always begin with CBH, DDH, EDH or FDH. $\overline{M1}$ also occurs with $\overline{IORQ}$ to indicate an interrupt acknowledge cycle.

$\overline{MREQ}$
(Memory Request)

Tri-state output, active low. The memory request signal indicates that the address bus holds a valid address for a memory read or memory write operation.

$\overline{\text{IORQ}}$ (Input/Output Request)	Tri-state output, active low. The $\overline{\text{IORQ}}$ signal indicates that the lower half of the address bus holds a valid I/O address for a I/O read or write operation. An $\overline{\text{IORQ}}$ signal is also generated with an $\overline{\text{M1}}$ signal when an interrupt is being acknowledged to indicate that an interrupt response vector can be placed on the data bus. Interrupt Acknowledge operations occur during M_1 time while I/O operations never occur during M_1 time.
$\overline{\text{RD}}$ (Memory Read)	Tri-state output, active low. $\overline{\text{RD}}$ indicates that the CPU wants to read data from memory or an I/O device. The addressed I/O device or memory should use this signal to gate data onto the CPU data bus.
$\overline{\text{WR}}$ (Memory Write)	Tri-state output, active low. $\overline{\text{WR}}$ indicates that the CPU data bus holds valid data to be stored in the addressed memory or I/O device.
$\overline{\text{RFSH}}$ (Refresh)	Output, active low. $\overline{\text{RFSH}}$ indicates that the lower 7 bits of the address bus contain a refresh address for dynamic memories and the current $\overline{\text{MREQ}}$ signal should be used to do a refresh read to all dynamic memories.
$\overline{\text{HALT}}$ (Halt state)	Output, active low. $\overline{\text{HALT}}$ indicates that the CPU has executed a HALT software instruction and is awaiting either a non maskable or a maskable interrupt (with the mask enabled) before operation can resume. While halted, the CPU executes NOP's to maintain memory refresh activity.
$\overline{\text{WAIT}}$ (Wait)	Input, active low. $\overline{\text{WAIT}}$ indicates to the Z-80 CPU that the addressed memory or I/O devices are not ready for a data transfer. The CPU continues to enter wait states for as long as this signal is active. This signal allows memory or I/O devices of any speed to be synchronized to the CPU.
$\overline{\text{INT}}$ (Interrupt Request)	Input, active low. The Interrupt Request signal is generated by I/O devices. A request will be honored at the end of the current instruction if the internal software controlled interrupt enable flip-flop (IFF) is enabled and if the $\overline{\text{BUSRQ}}$ signal is not active. When the CPU accepts the interrupt, an acknowledge signal ($\overline{\text{IORQ}}$ during M_1 time) is sent out at the beginning of the next instruction cycle. The CPU can respond to an interrupt in three different modes that are described in detail in section 5.4 (CPU Control Instructions).
$\overline{\text{NMI}}$ (Non Maskable Interrupt)	Input, negative edge triggered. The non maskable interrupt request line has a higher priority than $\overline{\text{INT}}$ and is always recognized at the end of the current instruction, independent of the status of the interrupt enable flip-flop. $\overline{\text{NMI}}$ automatically forces the Z-80 CPU to restart to location 0066_H. The program counter is automatically saved in the external stack so that the user can return to the program that was interrupted. Note that continuous $\overline{\text{WAIT}}$ cycles can prevent the current instruction from ending, and that a $\overline{\text{BUSRQ}}$ will override a $\overline{\text{NMI}}$.

$\overline{\text{RESET}}$

Input, active low. $\overline{\text{RESET}}$ forces the program counter to zero and initializes the CPU. The CPU initialization includes:

1) Disable the interrupt enable flip-flop
2) Set Register I = 00_H
3) Set Register R = 00_H
4) Set Interrupt Mode 0

During reset time, the address bus and data bus go to a high impedance state and all control output signals go to the inactive state.

$\overline{\text{BUSRQ}}$
(Bus Request)

Input, active low. The bus request signal is used to request the CPU address bus, data bus and tri-state output control signals to go to a high impedance state so that other devices can control these buses. When $\overline{\text{BUSRQ}}$ is activated, the CPU will set these buses to a high impedance state as soon as the current CPU machine cycle is terminated.

$\overline{\text{BUSAK}}$
(Bus Acknowledge)

Output, active low. Bus acknowledge is used to indicate to the requesting device that the CPU address bus, data bus and tri-state control bus signals have been set to their high impedance state and the external device can now control these signals.

Φ

Single phase TTL level clock which requires only a 330 ohm pull-up resistor to +5 volts to meet all clock requirements.

4.0 CPU TIMING

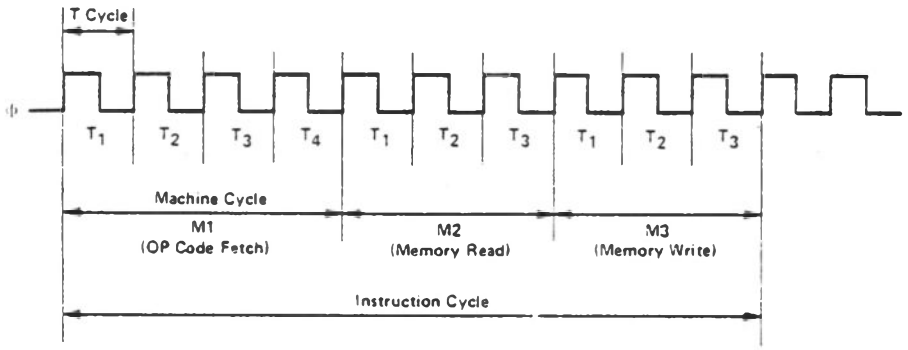
The Z-80 CPU executes instructions by stepping through a very precise set of a few basic operations. These include:

Memory read or write

I/O device read or write

Interrupt acknowledge

All instructions are merely a series of these basic operations. Each of these basic operations can take from three to six clock periods to complete or they can be lengthened to synchronize the CPU to the speed of external devices. The basic clock periods are referred to as T cycles and the basic operations are referred to as M (for machine) cycles. Figure 4.0-0 illustrates how a typical instruction will be merely a series of specific M and T cycles. Notice that this instruction consists of three machine cycles (M1, M2 and M3). The first machine cycle of any instruction is a fetch cycle which is four, five or six T cycles long (unless lengthened by the wait signal which will be fully described in the next section). The fetch cycle (M1) is used to fetch the OP code of the next instruction to be executed. Subsequent machine cycles move data between the CPU and memory or I/O devices and they may have anywhere from three to five T cycles (again they may be lengthened by wait states to synchronize the external devices to the CPU). The following paragraphs describe the timing which occurs within any of the basic machine cycles. In section 10, the exact timing for each instruction is specified.



BASIC CPU TIMING EXAMPLE
FIGURE 4.0-0

All CPU timing can be broken down into a few very simple timing diagrams as shown in figure 4.0-1 through 4.0-7. These diagrams show the following basic operations with and without wait states (wait states are added to synchronize the CPU to slow memory or I/O devices).

4.0-1. Instruction OP code fetch (M1 cycle)

4.0-2. Memory data read or write cycles

4.0-3. I/O read or write cycles

4.0-4. Bus Request/Acknowledge Cycle

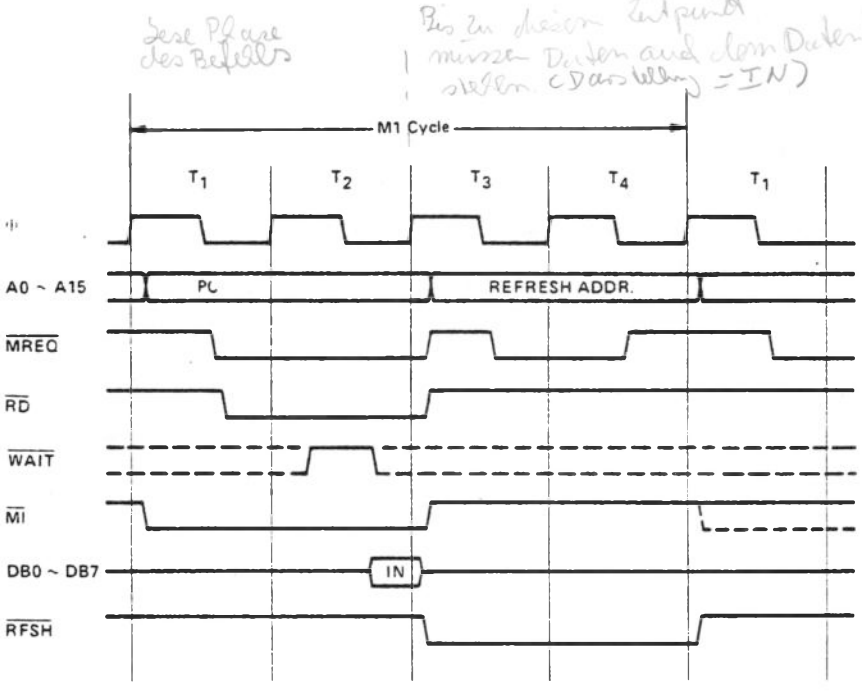
4.0-5. Interrupt Request/Acknowledge Cycle

4.0-6. Non maskable Interrupt Request/Acknowledge Cycle

4.0-7. Exit from a HALT instruction

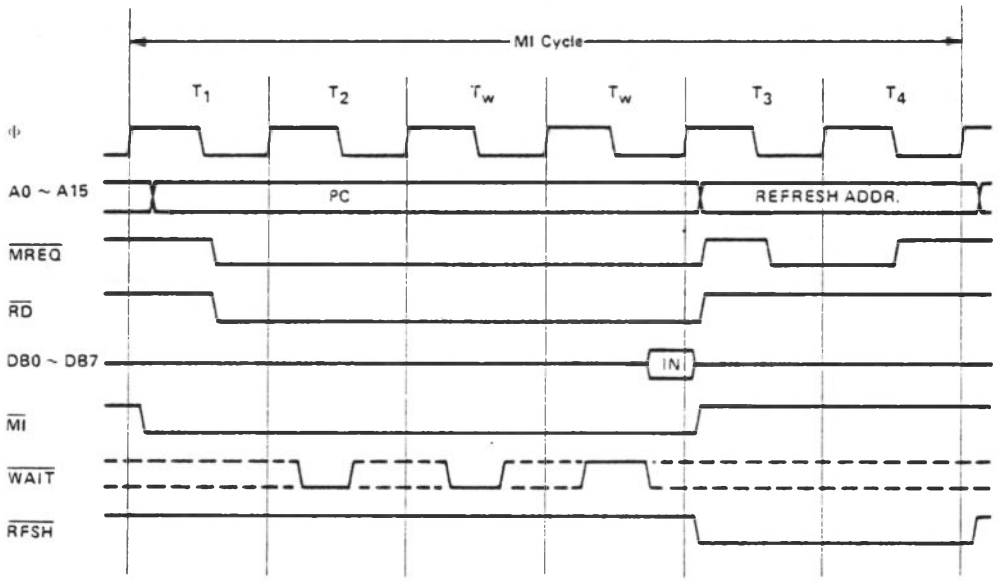
INSTRUCTION FETCH

Figure 4.0-1 shows the timing during an M1 cycle (OP code fetch). Notice that the PC is placed on the address bus at the beginning of the M1 cycle. One half clock time later the $\overline{MREQ}$ signal goes active. At this time the address to the memory has had time to stabilize so that the falling edge of $\overline{MREQ}$ can be used directly as a chip enable clock to dynamic memories. The $\overline{RD}$ line also goes active to indicate that the memory read data should be enabled onto the CPU data bus. The CPU samples the data from the memory on the data bus with the rising edge of the clock of state T3 and this same edge is used by the CPU to turn off the $\overline{RD}$ and $\overline{MRQ}$ signals. Thus the data has already been sampled by the CPU before the $\overline{RD}$ signal becomes inactive. Clock state T3 and T4 of a fetch cycle are used to refresh dynamic memories. (The CPU uses this time to decode and execute the fetched instruction so that no other operation could be performed at this time). During T3 and T4 the lower 7 bits of the address bus contain a memory refresh address and the $\overline{RFSH}$ signal becomes active to indicate that a refresh read of all dynamic memories should be accomplished. Notice that a $\overline{RD}$ signal is not generated during refresh time to prevent data from different memory segments from being gated onto the data bus. The $\overline{MREQ}$ signal during refresh time should be used to perform a refresh read of all memory elements. The refresh signal can not be used by itself since the refresh address is only guaranteed to be stable during $\overline{MREQ}$ time.



INSTRUCTION OP CODE FETCH
FIGURE 4.0-1

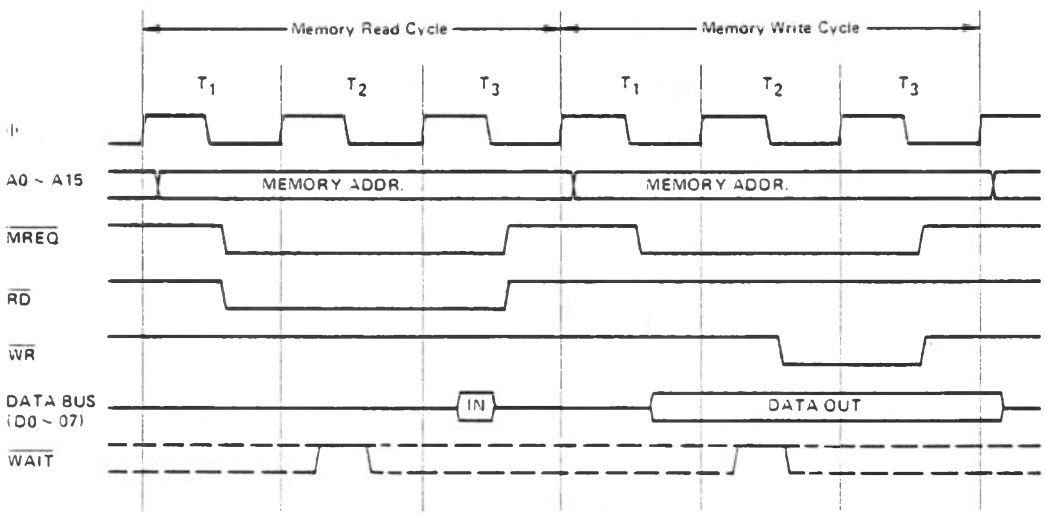
Figure 4.0-1A illustrates how the fetch cycle is delayed if the memory activates the $\overline{WAIT}$ line. During T2 and every subsequent Tw, the CPU samples the $\overline{WAIT}$ line with the falling edge of Φ. If the $\overline{WAIT}$ line is active at this time, another wait state will be entered during the following cycle. Using this technique the read cycle can be lengthened to match the access time of any type of memory device.



INSTRUCTION OP CODE FETCH WITH WAIT STATES
FIGURE 4.0-1A

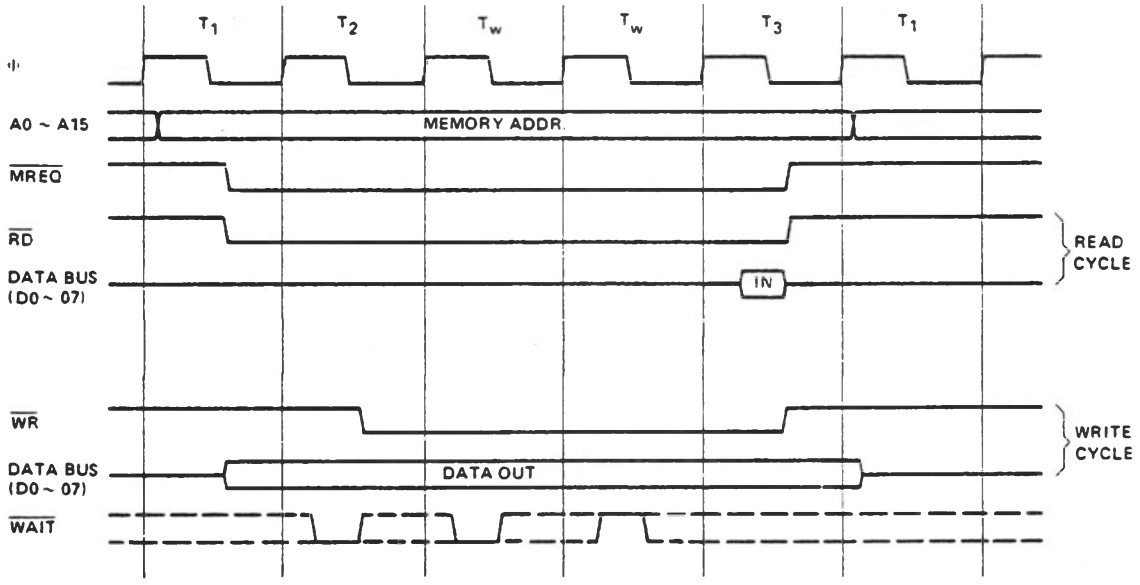
MEMORY READ OR WRITE

Figure 4.0-2 illustrates the timing of memory read or write cycles other than an OP code fetch (M1 cycle). These cycles are generally three clock periods long unless wait states are requested by the memory via the WAIT signal. The MREQ signal and the RD signal are used the same as in the fetch cycle. In the case of a memory write cycle, the MREQ also becomes active when the address bus is stable so that it can be used directly as a chip enable for dynamic memories. The WR line is active when data on the data bus is stable so that it can be used directly as a R/W pulse to virtually any type of semiconductor memory. Furthermore the WR signal goes inactive one half T state before the address and data bus contents are changed so that the overlap requirements for virtually any type of semiconductor memory type will be met.



MEMORY READ OR WRITE CYCLES
FIGURE 4.0-2

Figure 4.0-2A illustrates how a $\overline{WAIT}$ request signal will lengthen any memory read or write operation. This operation is identical to that previously described for a fetch cycle. Notice in this figure that a separate read and a separate write cycle are shown in the same figure although read and write cycles can never occur simultaneously.



MEMORY READ OR WRITE CYCLES WITH WAIT STATES
FIGURE 4.0-2A

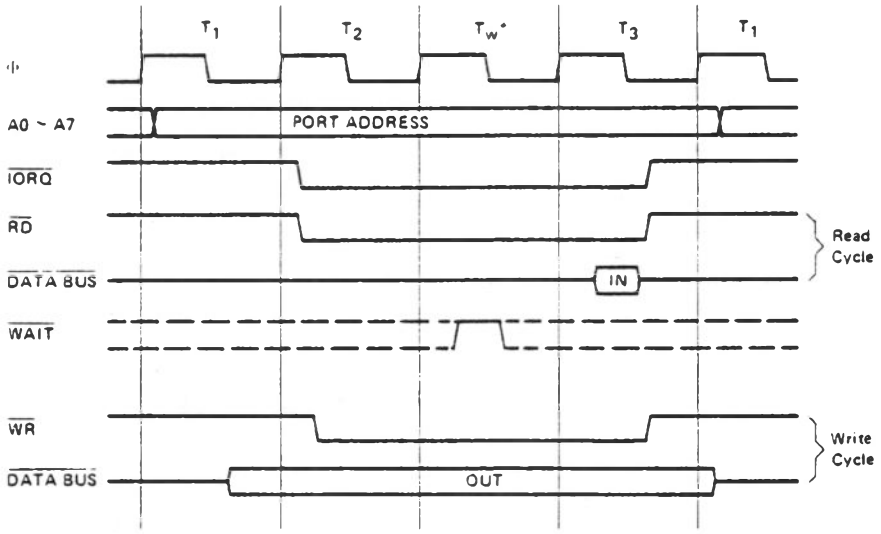
INPUT OR OUTPUT CYCLES

Figure 4.0-3 illustrates an I/O read or I/O write operation. Notice that during I/O operations a single wait state is automatically inserted. The reason for this is that during I/O operations, the time from when the $\overline{IORQ}$ signal goes active until the CPU must sample the $\overline{WAIT}$ line is very short and without this extra state sufficient time does not exist for an I/O port to decode its address and activate the $\overline{WAIT}$ line if a wait is required. Also, without this wait state it is difficult to design MOS I/O devices that can operate at full CPU speed. During this wait state time the $\overline{WAIT}$ request signal is sampled. During a read I/O operation, the $\overline{RD}$ line is used to enable the addressed port onto the data bus just as in the case of a memory read. For I/O write operations, the $\overline{WR}$ line is used as a clock to the I/O port, again with sufficient overlap timing automatically provided so that the rising edge may be used as a data clock.

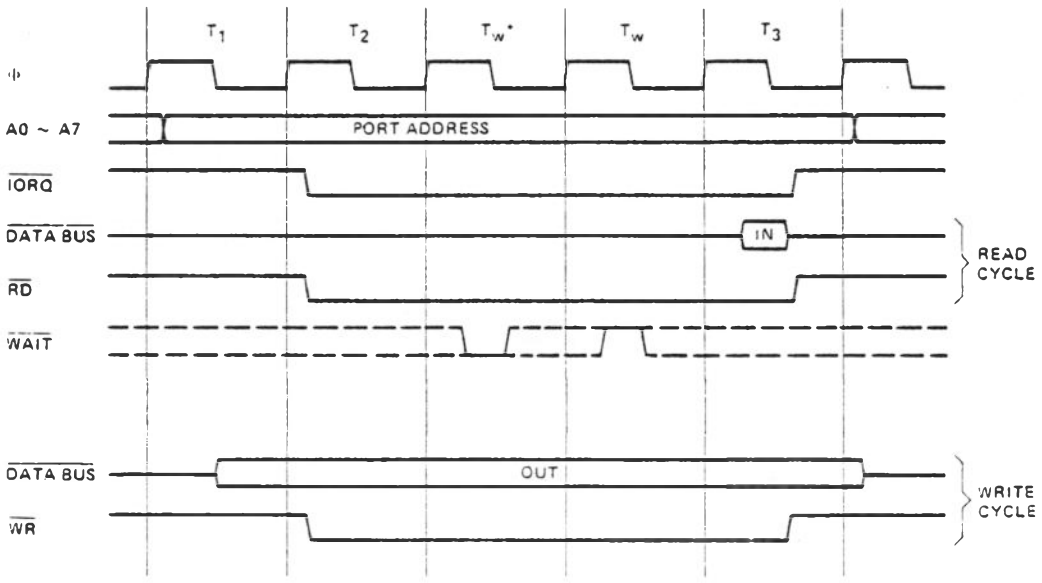
Figure 4.0-3A illustrates how additional wait states may be added with the $\overline{WAIT}$ line. The operation is identical to that previously described.

BUS REQUEST/ACKNOWLEDGE CYCLE

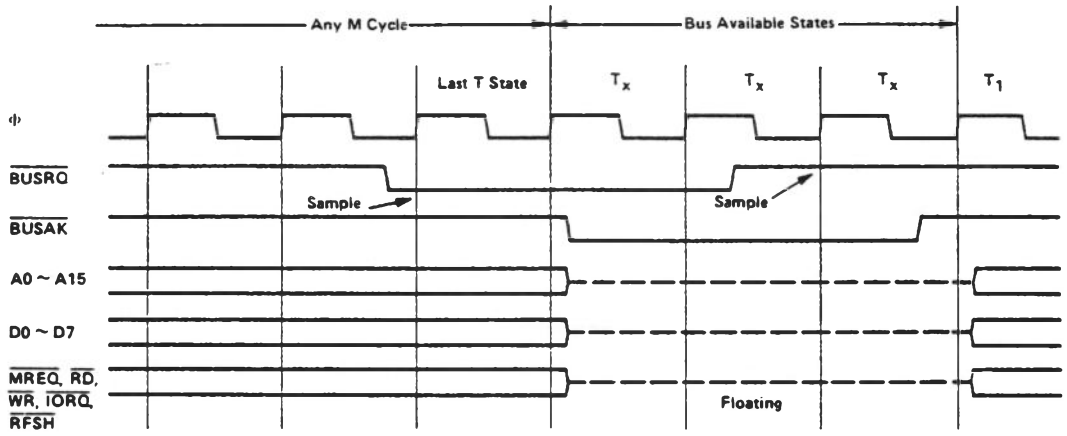
Figure 4.0-4 illustrates the timing for a Bus Request/Acknowledge cycle. The $\overline{BUSRQ}$ signal is sampled by the CPU with the rising edge of the last clock period of any machine cycle. If the $\overline{BUSRQ}$ signal is active, the CPU will set its address, data and tri-state control signals to the high impedance state with the rising edge of the next clock pulse. At that time any external device can control the buses to transfer data between memory and I/O devices. (This is generally known as Direct Memory Access [DMA] using cycle stealing). The maximum time for the CPU to respond to a bus request is the length of a machine cycle and the external controller can maintain control of the bus for as many clock cycles as is desired. Note, however, that if very long DMA cycles are used, and dynamic memories are being used, the external controller must also perform the refresh function. This situation only occurs if very large blocks of data are transferred under DMA control. Also note that during a bus request cycle, the CPU cannot be interrupted by either a $\overline{NMI}$ or an $\overline{INT}$ signal.



INPUT OR OUTPUT CYCLES
FIGURE 4.0-3



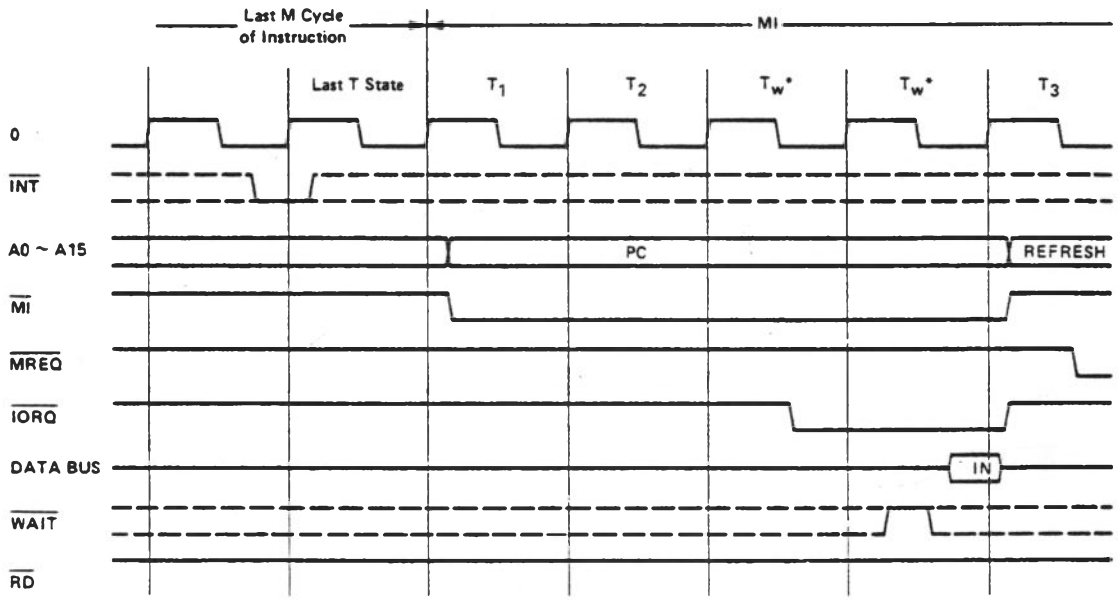
INPUT OR OUTPUT CYCLES WITH WAIT STATES
FIGURE 4.0-3A



BUS REQUEST/ACKNOWLEDGE CYCLE
FIGURE 4.0-4

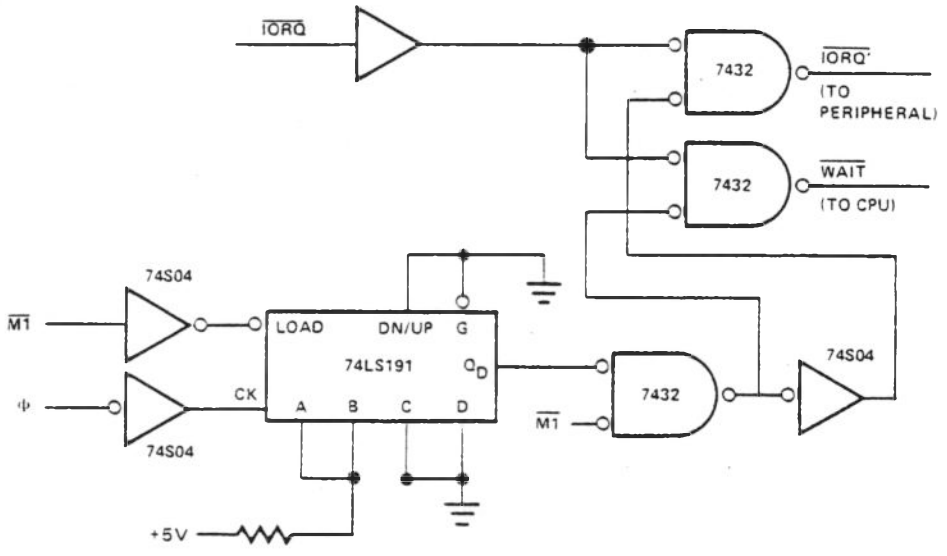
INTERRUPT REQUEST/ACKNOWLEDGE CYCLE

Figure 4.0-5 illustrates the timing associated with an interrupt cycle. The interrupt signal ($\overline{INT}$) is sampled by the CPU with the rising edge of the last clock at the end of any instruction. The signal will not be accepted if the internal CPU software controlled interrupt enable flip-flop is not set or if the $\overline{BUSRQ}$ signal is active. When the signal is accepted a special M1 cycle is generated. During this special M1 cycle the $\overline{IORQ}$ signal becomes active (instead of the normal $\overline{MREQ}$) to indicate that the interrupting device can place an 8-bit vector on the data bus. Notice that two wait states are automatically added to this cycle. These states are added so that a ripple priority interrupt scheme can be easily implemented. The two wait states allow sufficient time for the ripple signals to stabilize and identify which I/O device must insert the response vector. Refer to section 8.0 for details on how the interrupt response vector is utilized by the CPU.

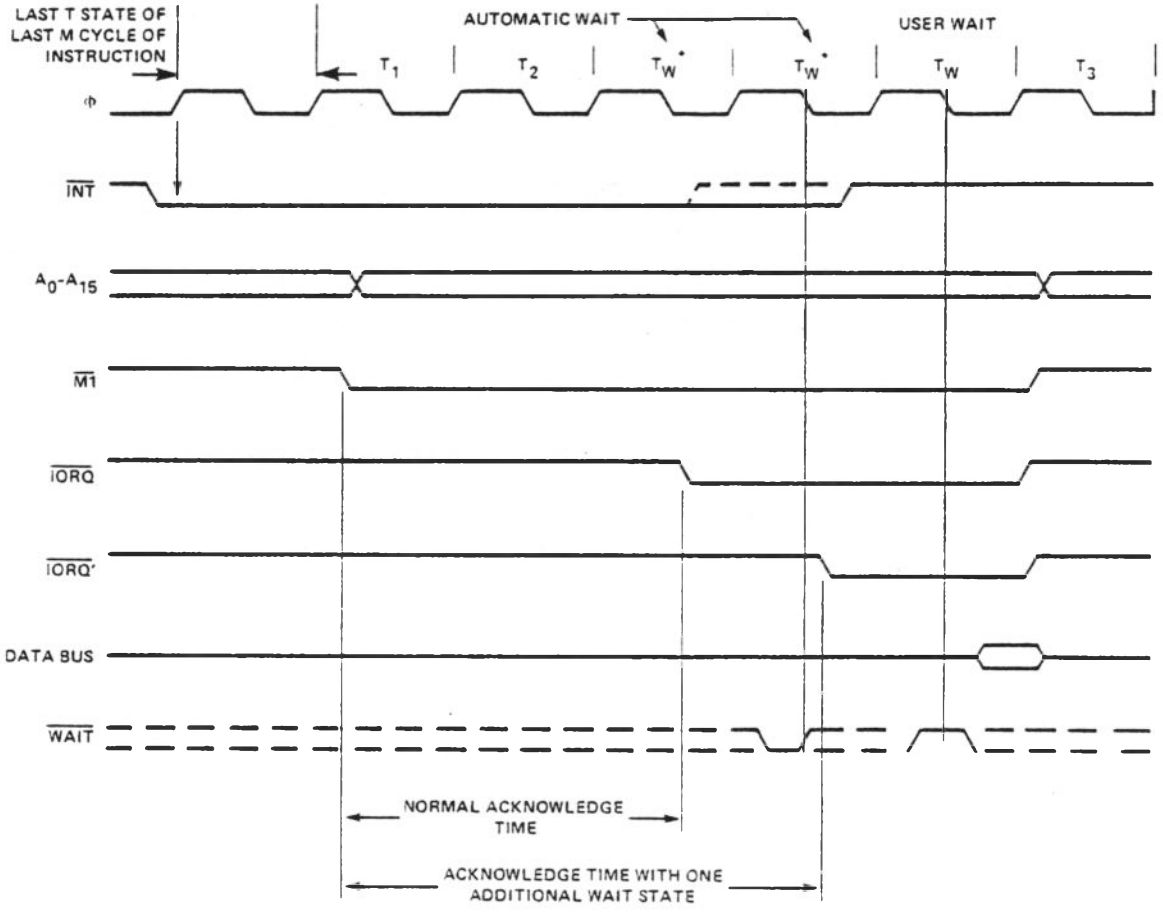


INTERRUPT REQUEST/ACKNOWLEDGE CYCLE
FIGURE 4.0-5

Figures 4.0-5A and 4.0-5B illustrate how a programmable counter can be used to extend interrupt acknowledge time. (Configured as shown to add one wait state)



EXTENDING INTERRUPT ACKNOWLEDGE TIME WITH WAIT STATE
FIGURE 4.0-5A



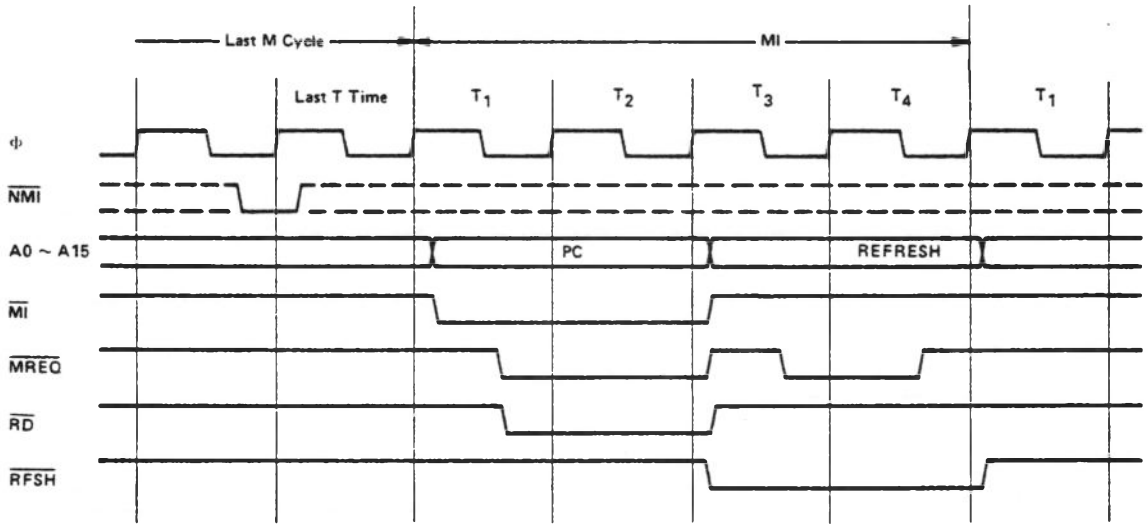
REQUEST/ACKNOWLEDGE CYCLE WITH ONE ADDITIONAL WAIT STATE
FIGURE 4.0-5B

NON MASKABLE INTERRUPT RESPONSE

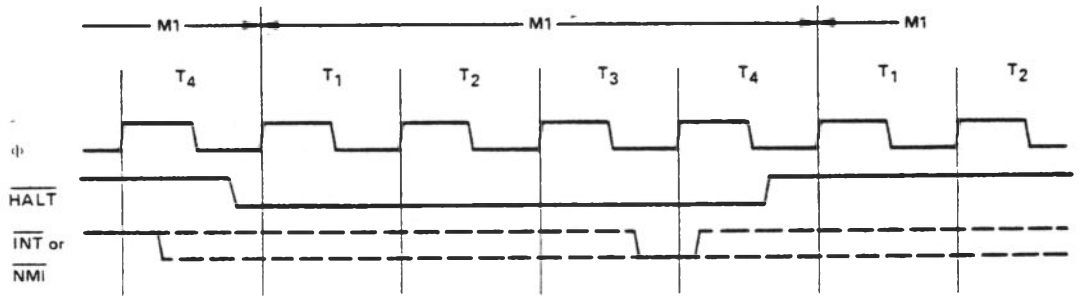
Figure 4.0-6 illustrates the request/acknowledge cycle for the non maskable interrupt. This signal is sampled at the same time as the interrupt line, but this line has priority over the normal interrupt and it can not be disabled under software control. Its usual function is to provide immediate response to important signals such as an impending power failure. The CPU response to a non maskable interrupt is similar to a normal memory read operation. The only difference being that the content of the data bus is ignored while the processor automatically stores the PC in the external stack and jumps to location 0066_H. The service routine for the non maskable interrupt must begin at this location if this interrupt is used.

HALT EXIT

Whenever a software halt instruction is executed the CPU begins executing NOP's until an interrupt is received (either a non maskable or a maskable interrupt while the interrupt flip flop is enabled). The two interrupt lines are sampled with the rising clock edge during each T4 state as shown in figure 4.0-7. If a non maskable interrupt has been received or a maskable interrupt has been received and the interrupt enable flip-flop is set, then the halt state will be exited on the next rising clock edge. The following cycle will then be an interrupt acknowledge cycle corresponding to the type of interrupt that was received. If both are received at this time, then the non maskable one will be acknowledged since it has highest priority. The purpose of executing NOP instructions while in the halt state is to keep the memory refresh signals active. Each cycle in the halt state is a normal M1 (fetch) cycle except that the data received from the memory is ignored and a NOP instruction is forced internally to the CPU. The halt acknowledge signal is active during this time to indicate that the processor is in the halt state.



NON MASKABLE INTERRUPT REQUEST OPERATION
FIGURE 4.0-6



HALT INSTRUCTION
IS RECEIVED
DURING THIS
MEMORY CYCLE

HALT EXIT
FIGURE 4.0-7

5.0 Z-80 CPU INSTRUCTION SET

The Z-80 CPU can execute 158 different instruction types including all 78 of the 8080A CPU. The instructions can be broken down into the following major groups:

- Load and Exchange
- Block Transfer and Search
- Arithmetic and Logical
- Rotate and Shift
- Bit Manipulation (set, reset, test)
- Jump, Call and Return
- Input/Output
- Basic CPU Control

5.1 INTRODUCTION TO INSTRUCTION TYPES

The load instructions move data internally between CPU registers or between CPU registers and external memory. All of these instructions must specify a source location from which the data is to be moved and a destination location. The source location is not altered by a load instruction. Examples of load group instructions include moves between any of the general purpose registers such as move the data to Register B from Register C. This group also includes load immediate to any CPU register or to any external memory location. Other types of load instructions allow transfer between CPU registers and memory locations. The exchange instructions can trade the contents of two registers.

A unique set of block transfer instructions is provided in the Z-80. With a single instruction a block of memory of any size can be moved to any other location in memory. This set of block moves is extremely valuable when large strings of data must be processed. The Z-80 block search instructions are also valuable for this type of processing. With a single instruction, a block of external memory of any desired length can be searched for any 8-bit character. Once the character is found or the end of the block is reached, the instruction automatically terminates. Both the block transfer and the block search instructions can be interrupted during their execution so as to not occupy the CPU for long periods of time.

The arithmetic and logical instructions operate on data stored in the accumulator and other general purpose CPU registers or external memory locations. The results of the operations are placed in the accumulator and the appropriate flags are set according to the result of the operation. An example of an arithmetic operation is adding the accumulator to the contents of an external memory location. The results of the addition are placed in the accumulator. This group also includes 16-bit addition and subtraction between 16-bit CPU registers.

The rotate and shift group allows any register or any memory location to be rotated right or left with or without carry either arithmetic or logical. Also, a digit in the accumulator can be rotated right or left with two digits in any memory location.

The bit manipulation instructions allow any bit in the accumulator, any general purpose register or any external memory location to be set, reset or tested with a single instruction. For example, the most significant bit of register H can be reset. This group is especially useful in control applications and for controlling software flags in general purpose programming.

The jump, call and return instructions are used to transfer between various locations in the user's program. This group uses several different techniques for obtaining the new program counter address from specific external memory locations. A unique type of call is the restart instruction. This instruction actually contains the new address as a part of the 8-bit OP code. This is possible since only 8 separate addresses located in page zero of the external memory may be specified. Program jumps may also be achieved by loading register HL, IX or IY directly into the PC, thus allowing the jump address to be a complex function of the routine being executed.

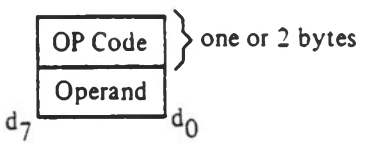
The input/output group of instructions in the Z-80 allow for a wide range of transfers between external memory locations or the general purpose CPU registers, and the external I/O devices. In each case, the port number is provided on the lower 8 bits of the address bus during any I/O transaction. One instruction allows this port number to be specified by the second byte of the instruction while other Z-80 instructions allow it to be specified as the content of the C register. One major advantage of using the C register as a pointer to the I/O device is that it allows different I/O ports to share common software driver routines. This is not possible when the address is part of the OP code if the routines are stored in ROM. Another feature of these input instructions is that they set the flag register automatically so that additional operations are not required to determine the state of the input data (for example its parity). The Z-80 CPU includes single instructions that can move blocks of data (up to 256 bytes) automatically to or from any I/O port directly to any memory location. In conjunction with the dual set of general purpose registers, these instructions provide for fast I/O block transfer rates. The value of this I/O instruction set is demonstrated by the fact that the Z-80 CPU can provide all required floppy disk formatting (i.e., the CPU provides the preamble, address, data and enables the CRC codes) on double density floppy disk drives on an interrupt driven basis.

Finally, the basic CPU control instructions allow various options and modes. This group includes instructions such as setting or resetting the interrupt enable flip flop or setting the mode of interrupt response.

5.2 ADDRESSING MODES

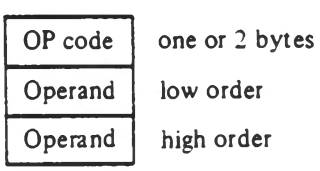
Most of the Z-80 instructions operate on data stored in internal CPU registers, external memory or in the I/O ports. Addressing refers to how the address of this data is generated in each instruction. This section gives a brief summary of the types of addressing used in the Z-80 while subsequent sections detail the type of addressing available for each instruction group.

Immediate. In this mode of addressing the byte following the OP code in memory contains the actual operand.



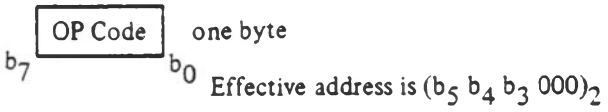
Examples of this type of instruction would be to load the accumulator with a constant, where the constant is the byte immediately following the OP code.

Immediate Extended. This mode is merely an extension of immediate addressing in that the two bytes following the OP codes are the operand.



Examples of this type of instruction would be to load the HL register pair (16-bit register) with 16 bits (2 bytes) of data.

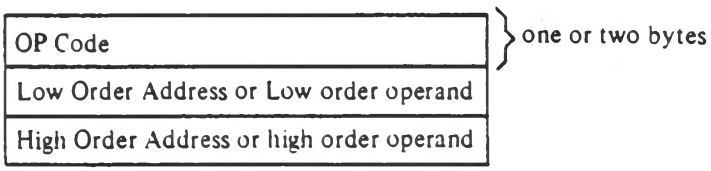
Modified Page Zero Addressing. The Z-80 has a special single byte call instruction to any of 8 locations in page zero of memory. This instruction (which is referred to as a restart) sets the PC to an effective address in page zero. The value of this instruction is that it allows a single byte to specify a complete 16-bit address where commonly called subroutines are located, thus saving memory space.



Relative Addressing. Relative addressing uses one byte of data following the OP code to specify a displacement from the existing program to which a program jump can occur. This displacement is a signed two's complement number that is added to the address of the OP code of the following instruction.

The value of relative addressing is that it allows jumps to nearby locations while only requiring two bytes of memory space. For most programs, relative jumps are by far the most prevalent type of jump due to the proximity of related program segments. Thus, these instructions can significantly reduce memory space requirements. The signed displacement can range between +127 and -128 from A + 2. This allows for a total displacement of +129 to -126 from the jump relative OP code address. Another major advantage is that it allows for relocatable code.

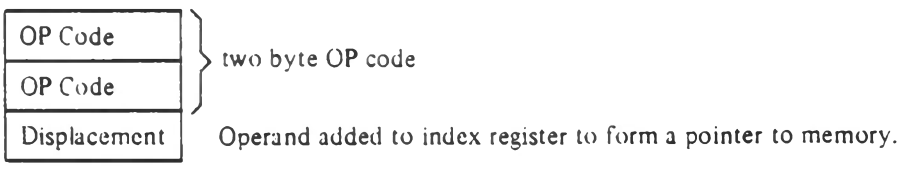
Extended Addressing. Extended Addressing provides for two bytes (16 bits) of address to be included in the instruction. This data can be an address to which a program can jump or it can be an address where an operand is located.



Extended addressing is required for a program to jump from any location in memory to any other location, or load and store data in any memory location.

When extended addressing is used to specify the source or destination address of an operand, the notation (nn) will be used to indicate the content of memory at nn, where nn is the 16-bit address specified in the instruction. This means that the two bytes of address nn are used as a pointer to a memory location. The use of the parentheses always means that the value enclosed within them is used as a pointer to a memory location. For example, (1200) refers to the contents of memory at location 1200.

Indexed Addressing. In this type of addressing, the byte of data following the OP code contains a displacement which is added to one of the two index registers (the OP code specifies which index register is used) to form a pointer to memory. The contents of the index register are not altered by this operation.



An example of an indexed instruction would be to load the contents of the memory location (Index Register + Displacement) into the accumulator. The displacement is a signed two's complement number. Indexed addressing greatly simplifies programs using tables of data since the index register can point to the start of any table. Two index registers are provided since very often operations require two or more tables. Indexed addressing also allows for relocatable code.

The two index registers in the Z-80 are referred to as IX and IY. To indicate indexed addressing the notation:

(IX+d) or (IY+d)

is used. Here d is the displacement specified after the OP code. The parentheses indicate that this value is used as a pointer to external memory.

Register Addressing. Many of the Z-80 OP codes contain bits of information that specify which CPU register is to be used for an operation. An example of register addressing would be to load the data in register B into register C.

Implied Addressing. Implied addressing refers to operations where the OP code automatically implies one or more CPU registers as containing the operands. An example is the set of arithmetic operations where the accumulator is always implied to be the destination of the results.

Register Indirect Addressing. This type of addressing specifies a 16-bit CPU register pair (such as HL) to be used as a pointer to any location in memory. This type of instruction is very powerful and it is used in a wide range of applications.

An example of this type of instruction would be to load the accumulator with the data in the memory location pointed to by the HL register contents. Indexed addressing is actually a form of register indirect addressing except that a displacement is added with indexed addressing. Register indirect addressing allows for very powerful but simple to implement memory accesses. The block move and search commands in the Z-80 are extensions of this type of addressing where automatic register incrementing, decrementing and comparing has been added. The notation for indicating register indirect addressing is to put parentheses around the name of the register that is to be used as the pointer. For example, the symbol

(HL)

specifies that the contents of the HL register are to be used as a pointer to a memory location. Often register indirect addressing is used to specify 16-bit operands. In this case, the register contents point to the lower order portion of the operand while the register contents are automatically incremented to obtain the upper portion of the operand.

Bit Addressing. The Z-80 contains a large number of bit set, reset and test instructions. These instructions allow any memory location or CPU register to be specified for a bit operation through one of three previous addressing modes (register, register indirect and indexed) while three bits in the OP code specify which of the eight bits is to be manipulated.

ADDRESSING MODE COMBINATIONS

Many instructions include more than one operand (such as arithmetic instructions or loads). In these cases, two types of addressing may be employed. For example, load can use immediate addressing to specify the source and register indirect or indexed addressing to specify the destination.

5.3 INSTRUCTION OP CODES

This section describes each of the Z-80 instructions and provides tables listing the OP codes for every instruction. In each of these tables the OP codes in bold type are identical to those offered in the 8080A CPU. Also shown is the assembly language mnemonic that is used for each instruction. All instruction OP codes are listed in hexadecimal notation. Single byte OP codes require two hex characters while double byte OP codes require four hex characters. The conversion from hex to binary is repeated here for convenience.

Hex		Binary		Decimal	Hex		Binary		Decimal
0	=	0000	=	0	8	=	1000	=	8
1	=	0001	=	1	9	=	1001	=	9
2	=	0010	=	2	A	=	1010	=	10
3	=	0011	=	3	B	=	1011	=	11
4	=	0100	=	4	C	=	1100	=	12
5	=	0101	=	5	D	=	1101	=	13
6	=	0110	=	6	E	=	1110	=	14
7	=	0111	=	7	F	=	1111	=	15

Z-80 instruction mnemonics consist of an OP code and zero, one or two operands. Instructions in which the operand is implied have no operand. Instructions which have only one logical operand or those in which one operand is invariant (such as the Logical OR instruction) are represented by a one operand mnemonic. Instructions which may have two varying operands are represented by two operand mnemonics.

LOAD AND EXCHANGE

Table 5.3-1 defines the OP code for all of the 8-bit load instructions implemented in the Z-80 CPU. Also shown in this table is the type of addressing used for each instruction. The source of the data is found on the top horizontal row while the destination is specified by the left hand column. For example, load register C from register B uses the OP code 48H. In all of the tables the OP code is specified in hexadecimal notation and the 48H (=0100 1000 binary) code is fetched by the CPU from the external memory during M1 time, decoded and then the register transfer is automatically performed by the CPU.

The assembly language mnemonic for this entire group is LD, followed by the destination followed by the source (LD DEST., SOURCE). Note that several combinations of addressing modes are possible. For example, the source may use register addressing and the destination may be register indirect; such as load the memory location pointed to by register HL with the contents of register D. The OP code for this operation would be 72. The mnemonic for this load instruction would be as follows:

LD (HL), D

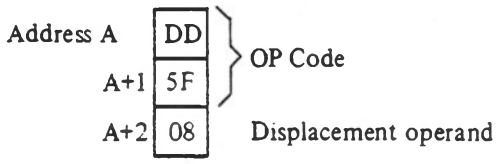
The parentheses around the HL means that the contents of HL are used as a pointer to a memory location. In all Z-80 load instruction mnemonics the destination is always listed first, with the source following. The Z-80 assembly language has been defined for ease of programming. Every instruction is self documenting and programs written in Z-80 language are easy to maintain.

Note in table 5.3-1 that some load OP codes that are available in the Z-80 use two bytes. This is an efficient method of memory utilization since 8, 16, 24 or 32 bit instructions are implemented in the Z-80. Thus often utilized instructions such as arithmetic or logical operations are only 8-bits which results in better memory utilization than is achieved with fixed instruction sizes such as 16-bits.

All load instructions using indexed addressing for either the source or destination location actually use three bytes of memory with the third byte being the displacement d. For example a load register E with the operand pointed to by IX with an offset of +8 would be written:

LD E, (IX + 8)

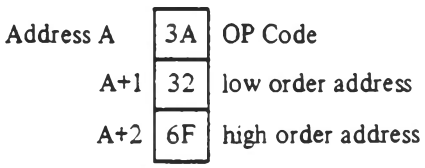
The instruction sequence for this in memory would be:



The two extended addressing instructions are also three byte instructions. For example the instruction to load the accumulator with the operand in memory location 6F32H would be written:

LD A, (6F 32H)

and its instruction sequence would be:

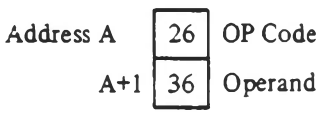


Notice that the low order portion of the address is always the first operand.

The load immediate instructions for the general purpose 8-bit registers are two-byte instructions. The instruction load register H with the value 36H would be written:

LD H, 36H

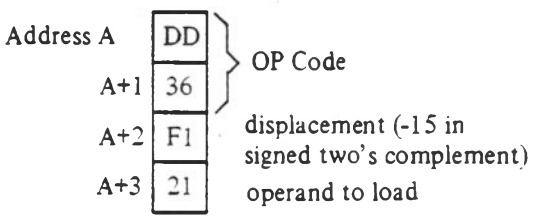
and its sequence would be:



Loading a memory location using indexed addressing for the destination and immediate addressing for the source requires four bytes. For example:

LD (IX - 15), 21H

would appear as:



Notice that with any indexed addressing the displacement always follows directly after the OP code.

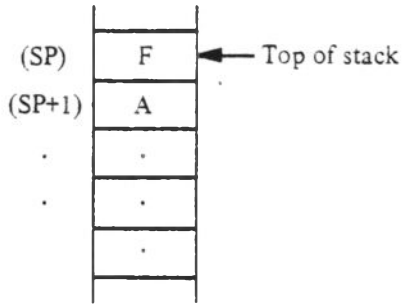
Table 5.3-2 specifies the 16-bit load operations. This table is very similar to the previous one. Notice that the extended addressing capability covers all register pairs. Also notice that register indirect operations specifying the stack pointer are the PUSH and POP instructions. The mnemonic for these instructions is "PUSH" and "POP." These differ from other 16-bit loads in that the stack pointer is automatically decremented and incremented as each byte is pushed onto or popped from the stack respectively. For example the instruction:

PUSH AF

is a single byte instruction with the OP code of F5H. When this instruction is executed the following sequence is generated:

Decrement SP
LD (SP), A
Decrement SP
LD (SP), F

Thus the external stack now appears as follows:



			SOURCE																
			IMPLIED		REGISTER							REG INDIRECT			INDEXED		EXT. ADDR.	IMME.
			I	R	A	B	C	D	E	H	L	(HL)	(BC)	(DE)	(IX + d)	(IY + d)	(nn)	n
DESTINATION	REGISTER	A	ED 57	ED 5F	7F	78	79	7A	7B	7C	7D	7E	0A	1A	DD 7E d	FD 7E d	3A n n	3E n
		B			47	40	41	42	43	44	45	46			DD 46 d	FD 46 d		06 n
		C			4F	48	49	4A	4B	4C	4D	4E			DD 4E d	FD 4E d		0E n
		D			57	50	51	52	53	54	55	56			DD 56 d	FD 56 d		16 n
		E			5F	58	59	5A	5B	5C	5D	5E			DD 5E d	FD 5E d		1E n
		H			67	60	61	62	63	64	65	66			DD 66 d	FD 66 d		26 n
		L			6F	68	69	6A	6B	6C	6D	6E			DD 6E d	FD 6E d		2E n
	REG INDIRECT	(HL)			77	70	71	72	73	74	75							36 n
		(BC)			02													
		(DE)			12													
	INDEXED	(IX+d)			DD 77 d	DD 70 d	DD 71 d	DD 72 d	DD 73 d	DD 74 d	DD 75 d							DD 36 d n
		(IY+d)			FD 77 d	FD 70 d	FD 71 d	FD 72 d	FD 73 d	FD 74 d	FD 75 d							FD 36 d n
	EXT. ADDR	(nn)			32 n n													
	IMPLIED	I			ED 47													
		R			ED 5F													

8 BIT LOAD GROUP
'LD'
TABLE 5.3–1

The POP instruction is the exact reverse of a PUSH. Notice that all PUSH and POP instructions utilize a 16-bit operand and the high order byte is always pushed first and popped last. That is a:

PUSH BC is PUSH B then C
PUSH DE is PUSH D then E
PUSH HL is PUSH H then L
POP HL is POP L then H

The instruction using extended immediate addressing for the source obviously requires 2 bytes of data following the OP code. For example:

LD DE, 0659H

will be:

Address A	11	OP Code
A+1	59	Low order operand to register E
A+2	06	High order operand to register D

In all extended immediate or extended addressing modes, the low order byte always appears first after the OP code.

Table 5.3-3 lists the 16-bit exchange instructions implemented in the Z-80. OP code 08H allows the programmer to switch between the two pairs of accumulator flag registers while D9H allows the programmer to switch between the duplicate set of six general purpose registers. These OP codes are only one byte in length to absolutely minimize the time necessary to perform the exchange so that the duplicate banks can be used to effect very fast interrupt response times.

BLOCK TRANSFER AND SEARCH

Table 5.3-4 lists the extremely powerful block transfer instructions. All of these instructions operate with three registers.

HL points to the source location.
DE points to the destination location.
BC is a byte counter.

After the programmer has initialized these three registers, any of these four instructions may be used. The LDI (Load and Increment) instruction moves one byte from the location pointed to by HL to the location pointed to by DE. Register pairs HL and DE are then automatically incremented and are ready to point to the following locations. The byte counter (register pair BC) is also decremented at this time. This instruction is valuable when blocks of data must be moved but other types of processing are required between each move. The LDIR (Load, increment and repeat) instruction is an extension of the LDI instruction. The same load and increment operation is repeated until the byte counter reaches the count of zero. Thus, this single instruction can move any block of data from one location to any other.

Note that since 16-bit registers are used, the size of the block can be up to 64K bytes (1K = 1024) long and it can be moved from any location in memory to any other location. Furthermore the blocks can be overlapping since there are absolutely no constraints on the data that is used in the three register pairs.

The LDD and LDDR instructions are very similar to the LDI and LDIR. The only difference is that register pairs HL and DE are decremented after every move so that a block transfer starts from the highest address of the designated block rather than the lowest.

SOURCE

DESTINATION		REGISTER							IMM. EXT.	EXT. ADDR.	REG. INDIR.
		AF	BC	DE	HL	SP	IX	IY	nn	(nn)	(SP)
REGISTER	AF										F1
	BC								01 n n	ED 48 n n	C1
	DE								11 n n	ED 58 n n	D1
	HL								21 n n	2A n n	E1
	SP				F9		DD F9	FD F9	31 n n	ED 78 n n	
	IX								DD 21 n n	DD 2A n n	DD E1
	IY								FD 21 n n	FD 2A n n	FD E1
EXT. ADDR.	(nn)		ED 43 n n	ED 53 n n	22 n n	ED 73 n n	DD 22 n n	FD 22 n n			
REG. IND.	(SP)	F5	C5	D5	E5		DD E5	FD E5			

PUSH INSTRUCTIONS →

POP INSTRUCTIONS ↑

NOTE: The Push & Pop Instructions adjust the SP after every execution

16 BIT LOAD GROUP
'LD'
'PUSH' AND 'POP'
TABLE 5.3–2

		IMPLIED ADDRESSING				
		AF'	BC', DE' & HL'	HL	IX	IY
IMPLIED	AF	08				
	BC, DE & HL		D9			
	DE			EB		
REG. INDIR.	(SP)			E3	DD E3	FD E3

EXCHANGES
'EX' AND 'EXX'
TABLE 5.3–3

			SOURCE: REG. INDIR. (HL)	
DESTINATION: REG. INDIR.	(DE)	ED A0	'LDI' – Load (DE)←(HL) Inc HL & DE, Dec BC
		ED B0	'LDIR,' – Load (DE)←(HL) Inc HL & DE, Dec BC, Repeat until BC = 0
		ED A8	'LDD' – Load (DE)←(HL) Dec HL & DE, Dec BC
		ED B8	'LDDR' – Load (DE)←(HL) Dec HL & DE, Dec BC, Repeat until BC = 0

Reg HL points to source
Reg DE points to destination
Reg BC is byte counter

BLOCK TRANSFER GROUP
TABLE 5.3–4

Table 5.3-5 specifies the OP codes for the four block search instructions. The first, CPI (compare and increment) compares the data in the accumulator, with the contents of the memory location pointed to by register HL. The result of the compare is stored in one of the flag bits (see section 6.0 for a detailed explanation of the flag operations) and the HL register pair is then incremented and the byte counter (register pair BC) is decremented.

The instruction CPIR is merely an extension of the CPI instruction in which the compare is repeated until either a match is found or the byte counter (register pair BC) becomes zero. Thus, this single instruction can search the entire memory for any 8-bit character.

The CPD (Compare and Decrement) and CPDR (Compare, Decrement and Repeat) are similar instructions, their only difference being that they decrement HL after every compare so that they search the memory in the opposite direction. (The search is started at the highest location in the memory block).

It should be emphasized again that these block transfer and compare instructions are extremely powerful in string manipulation applications.

ARITHMETIC AND LOGICAL

Table 5.3-6 lists all of the 8-bit arithmetic operations that can be performed with the accumulator. also listed are the increment (INC) and decrement (DEC) instructions. In all of these instructions, except INC and DEC, the specified 8-bit operation is performed between the data in the accumulator and the source data specified in the table. The result of the operation is placed in the accumulator with the exception of compare (CP) that leaves the accumulator unaffected. All of these operations affect the flag register as a result of the specified operation. (Section 6.0 provides all of the details on how the flags are affected by any instruction type). INC and DEC instructions specify a register or a memory location as both source and destination of the result. When the source operand is addressed using the index registers the displacement must follow directly. With immediate addressing the actual operand will follow directly. For example the instruction:

AND 07H

would appear as:

Address A	E6	OP Code
A+1	07	Operand

SEARCH LOCATION REG. INDIR. (HL)	
ED A1	'CPI' Inc HL, Dec BC
ED B1	'CPIR', Inc HL, Dec BC repeat until BC = 0 or find match
ED A9	'CPD' Dec HL & BC
ED B9	'CPDR' Dec HL & BC Repeat until BC = 0 or find match

HL points to location in memory to be compared with accumulator contents
BC is byte counter

BLOCK SEARCH GROUP
TABLE 5.3–5

Assuming that the accumulator contained the value F3H the result of 03H would be placed in the accumulator:

Acc before operation	1111 0011 = F3H
Operand	0000 0111 = 07H
Result to Acc	0000 0011 = 03H

The Add instruction (ADD) performs a binary add between the data in the source location and the data in the accumulator. The subtract (SUB) does a binary subtraction. When the add with carry is specified (ADC) or the subtract with carry (SBC), then the carry flag is also added or subtracted respectively. The flags and decimal adjust instruction (DAA) in the Z-80 (fully described in section 6.0) allow arithmetic operations for:

multiprecision packed BCD numbers

multiprecision signed or unsigned binary numbers

multiprecision two's complement signed numbers

Other instructions in this group are logical and (AND), logical or (OR), exclusive or (XOR) and compare (CP).

There are five general purpose arithmetic instructions that operate on the accumulator or carry flag. These five are listed in table 5.3-7. The decimal adjust instruction can adjust for subtraction as well as addition, thus making BCD arithmetic operations simple. Note that to allow for this operation the flag N is used. This flag is set if the last arithmetic operation was a subtract. The negate accumulator (NEG) instruction forms the two's complement of the number in the accumulator. Finally notice that a reset carry instruction is not included in the Z-80 since this operation can be easily achieved through other instructions such as a logical AND of the accumulator with itself.

Table 5.3-8 lists all of the 16-bit arithmetic operations between 16-bit registers. There are five groups of instructions including add with carry and subtract with carry. ADC and SBC affect all of the flags. These two groups simplify address calculation operations or other 16-bit arithmetic operations.

SOURCE

	REGISTER ADDRESSING							REG. INDIR.	INDEXED		IMMED.
	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)	n
'ADD'	87	80	81	82	83	84	85	86	DD 86 d	FD 86 d	C6 n
ADD w CARRY 'ADC'	8F	88	89	8A	8B	8C	8D	8E	DD 8E d	FD 8E d	CE n
SUBTRACT 'SUB'	97	90	91	92	93	94	95	96	DD 96 d	FD 96 d	D6 n
SUB w CARRY 'SBC'	9F	98	99	9A	9B	9C	9D	9E	DD 9E d	FD 9E d	DE n
'AND'	A7	A0	A1	A2	A3	A4	A5	A6	DD A6 d	FD A6 d	E6 n
'XOR'	AF	A8	A9	AA	AB	AC	AD	AE	DD AE d	FD AE d	EE n
'OR'	B7	B0	B1	B2	B3	B4	B5	B6	DD B6 d	FD B6 d	F6 n
COMPARE 'CP'	BF	B8	B9	BA	BB	BC	BD	BE	DD BE d	FD BE d	FE n
INCREMENT 'INC'	3C	04	0C	14	1C	24	2C	34	DD 34 d	FD 34 d	
DECREMENT 'DEC'	3D	05	0D	15	1D	25	2D	35	DD 35 d	FD 35 d	

8 BIT ARITHMETIC AND LOGIC
TABLE 5.3–6

Decimal Adjust Acc, 'DAA'	27
Complement Acc, 'CPL'	2F
Negate Acc, 'NEG' (2's complement)	ED 44
Complement Carry Flag, 'CCF'	3F
Set Carry Flag, 'SCF'	37

GENERAL PURPOSE AF OPERATIONS
TABLE 5.3-7

		REGISTER ADDRESSING							REG. INDIR.	INDEXED	
	BIT	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)
TEST 'BIT'	0	CB 47	CB 40	CB 41	CB 42	CB 43	CB 44	CB 45	CB 46	DD CB d 46	FD CB d 46
	1	CB 4F	CB 48	CB 49	CB 4A	CB 4B	CB 4C	CB 4D	CB 4E	DD CB d 4E	FD CB d 4E
	2	CB 57	CB 50	CB 51	CB 52	CB 53	CB 54	CB 55	CB 56	DD CB d 56	FD CB d 56
	3	CB 5F	CB 58	CB 59	CB 5A	CB 5B	CB 5C	CB 5D	CB 5E	DD CB d 5E	FD CB d 5E
	4	CB 67	CB 60	CB 61	CB 62	CB 63	CB 64	CB 65	CB 66	DD CB d 66	FD CB d 66
	5	CB 6F	CB 68	CB 69	CB 6A	CB 6B	CB 6C	CB 6D	CB 6E	DD CB d 6E	FD CB d 6E
	6	CB 77	CB 70	CB 71	CB 72	CB 73	CB 74	CB 75	CB 76	DD CB d 76	FD CB d 76
	7	CB 7F	CB 78	CB 79	CB 7A	CB 7B	CB 7C	CB 7D	CB 7E	DD CB d 7E	FD CB d 7E
RESET BIT 'RES'	0	CB 87	CB 80	CB 81	CB 82	CB 83	CB 84	CB 85	CB 86	DD CB d 86	FD CB d 86
	1	CB 8F	CB 88	CB 89	CB 8A	CB 8B	CB 8C	CB 8D	CB 8E	DD CB d 8E	FD CB d 8E
	2	CB 97	CB 90	CB 91	CB 92	CB 93	CB 94	CB 95	CB 96	DD CB d 96	FD CB d 96
	3	CB 9F	CB 98	CB 99	CB 9A	CB 9B	CB 9C	CB 9D	CB 9E	DD CB d 9E	FD CB d 9E
	4	CB A7	CB A0	CB A1	CB A2	CB A3	CB A4	CB A5	CB A6	DD CB d A6	FD CB d A6
	5	CB AF	CB A8	CB A9	CB AA	CB AB	CB AC	CB AD	CB AE	DD CB d AE	FD CB d AE
	6	CB B7	CB B0	CB B1	CB B2	CB B3	CB B4	CB B5	CB B6	DD CB d B6	FD CB d B6
	7	CB BF	CB B8	CB B9	CB BA	CB BB	CB BC	CB BD	CB BE	DD CB d BE	FD CB d BE
SET BIT 'SET'	0	CB C7	CB C0	CB C1	CB C2	CB C3	CB C4	CB C5	CB C6	DD CB d C6	FD CB d C6
	1	CB CF	CB C8	CB C9	CB CA	CB CB	CB CC	CB CD	CB CE	DD CB d CE	FD CB d CE
	2	CB D7	CB D0	CB D1	CB D2	CB D3	CB D4	CB D5	CB D6	DD CB d D6	FD CB d D6
	3	CB DF	CB D8	CB D9	CB DA	CB DB	CB DC	CB DD	CB DE	DD CB d DE	FD CB d DE
	4	CB E7	CB E0	CB E1	CB E2	CB E3	CB E4	CB E5	CB E6	DD CB d E6	FD CB d E6
	5	CB EF	CB E8	CB E9	CB EA	CB EB	CB EC	CB ED	CB EE	DD CB d EE	FD CB d EE
	6	CB F7	CB F0	CB F1	CB F2	CB F3	CB F4	CB F5	CB F6	DD CB d F6	FD CB d F6
	7	CB FF	CB F8	CB F9	CB FA	CB FB	CB FC	CB FD	CB FE	DD CB d FE	FD CB d FE

BIT MANIPULATION GROUP
TABLE 5.3–10

Disable Interrupt	– prevent interrupt before routine is exited.
LD A, n OUT n, A	– notify peripheral that service routine is complete
Enable Interrupt	
Return	

This seven byte sequence can be replaced with the one byte EI instruction and the two byte RETI instruction in the Z80. This is important since interrupt service time often must be minimized.

To facilitate program loop control the instruction DJNZ e can be used advantageously. This two byte, relative jump instruction decrements the B register and the jump occurs if the B register has not been decremented to zero. The relative displacement is expressed as a signed two's complement number. A simple example of its use might be:

Address	Instruction	Comments
N, N + 1	LD B, 7	; set B register to count of 7
N + 2 to N + 9	(Perform a sequence of instructions)	; loop to be performed 7 times
N + 10, N + 11	DJNZ -8	; to jump from N + 12 to N + 2
N + 12	(Next Instruction)	

			CONDITION									
			UN-COND.	CARRY	NON CARRY	ZERO	NON ZERO	PARITY EVEN	PARITY ODD	SIGN NEG	SIGN POS	REG B≠0
JUMP 'JP'	IMMED. EXT.	nn	C3 n n	DA n n	D2 n n	CA n n	C2 n n	EA n n	E2 n n	FA n n	F2 n n	
JUMP 'JR'	RELATIVE	PC+e	18 e-2	38 e-2	30 e-2	28 e-2	20 e-2					
JUMP 'JP'	REG. INDIR.	(HL)	E9									
JUMP 'JP'		(IX)	DD E9									
JUMP 'JP'		(IY)	FD E9									
'CALL'	IMMED. EXT.	nn	CD n n	DC n n	D4 n n	CC n n	C4 n n	EC n n	E4 n n	FC n n	F4 n n	
DECREMENT B, JUMP IF NON ZERO 'DJNZ'	RELATIVE	PC+e										10 e-2
RETURN 'RET'	REGISTER INDIR.	(SP) (SP+1)	C9	D8	D0	C8	C0	E8	E0	F8	F0	
RETURN FROM INT 'RETI'	REG. INDIR.	(SP) (SP+1)	ED 4D									
RETURN FROM NON MASKABLE INT 'RETN'	REG. INDIR.	(SP) (SP+1)	ED 45									

NOTE—CERTAIN FLAGS HAVE MORE THAN ONE PURPOSE. REFER TO SECTION 6.0 FOR DETAILS

JUMP, CALL and RETURN GROUP
TABLE 5.3–11

		SOURCE					
DESTINATION		BC	DE	HL	SP	IX	IY
'ADD'	HL	09	19	29	39		
	IX	DD 09	DD 19		DD 39	DD 29	
	IY	FD 09	FD 19		FD 39		FD 29
ADD WITH CARRY AND SET FLAGS 'ADC'	HL	ED 4A	ED 5A	ED 6A	ED 7A		
SUB WITH CARRY AND SET FLAGS 'SBC'	HL	ED 42	ED 52	ED 62	ED 72		
INCREMENT 'INC'		03	13	23	33	DD 23	FD 23
DECREMENT 'DEC'		0B	1B	2B	3B	DD 2B	FD 2B

16 BIT ARITHMETIC
TABLE 5.3–8

ROTATE AND SHIFT

A major capability of the Z-80 is its ability to rotate or shift data in the accumulator, any general purpose register, or any memory location. All of the rotate and shift OP codes are shown in table 5.3-9. Also included in the Z-80 are arithmetic and logical shift operations. These operations are useful in an extremely wide range of applications including integer multiplication and division. Two BCD digit rotate instructions (RRD and RLD) allow a digit in the accumulator to be rotated with the two digits in a memory location pointed to by register pair HL. (See figure 5.3-9). These instructions allow for efficient BCD arithmetic.

BIT MANIPULATION

The ability to set, reset and test individual bits in a register or memory location is needed in almost every program. These bits may be flags in a general purpose software routine, indications of external control conditions or data packed into memory locations to make memory utilization more efficient.

The Z-80 has the ability to set, reset or test any bit in the accumulator, any general purpose register or any memory location with a single instruction. Table 5.3-10 lists the 240 instructions that are available for this purpose. Register addressing can specify the accumulator or any general purpose register on which the operation is to be performed. Register indirect and indexed addressing are available to operate on external memory locations. Bit test operations set the zero flag (Z) if the tested bit is a zero. (Refer to section 6.0 for further explanation of flag operation).

JUMP, CALL AND RETURN

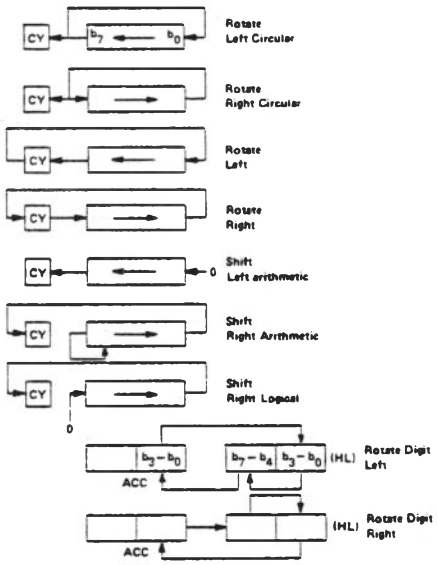
Figure 5.3-11 lists all of the jump, call and return instructions implemented in the Z-80 CPU. A jump is a branch in a program where the program counter is loaded with the 16-bit value as specified by one of the three available addressing modes (Immediate Extended, Relative or Register Indirect). Notice that the jump group has several different conditions that can be specified to be met before the jump will be made. If these conditions are not met, the program merely continues with the next sequential instruction. The conditions are all dependent on the data in the flag register. (Refer to section 6.0 for details on the flag register). The immediate extended addressing is used to jump to any location in the memory. This instruction requires three bytes (two to specify the 16-bit address) with the low order address byte first followed by the high order address byte.

Source and Destination

TYPE OF ROTATE OR SHIFT

	A	B	C	D	E	H	L	(HL)	(IX + d)	(IY + d)
'RLC'	CB 07	CB 00	CB 01	CB 02	CB 03	CB 04	CB 05	CB 06	DD CB d 06	FD CB d 06
'RRC'	CB 0F	CB 08	CB 09	CB 0A	CB 0B	CB 0C	CB 0D	CB 0E	DD CB d 0E	FD CB d 0E
'RL'	CB 17	CB 10	CB 11	CB 12	CB 13	CB 14	CB 15	CB 16	DD CB d 16	FD CB d 16
'RR'	CB 1F	CB 18	CB 19	CB 1A	CB 1B	CB 1C	CB 1D	CB 1E	DD CB d 1E	FD CB d 1E
'SLA'	CB 27	CB 20	CB 21	CB 22	CB 23	CB 24	CB 25	CB 26	DD CB d 26	FD CB d 26
'SRA'	CB 2F	CB 28	CB 29	CB 2A	CB 2B	CB 2C	CB 2D	CB 2E	DD CB d 2E	FD CB d 2E
'SRL'	CB 3F	CB 38	CB 39	CB 3A	CB 3B	CB 3C	CB 3D	CB 3E	DD CB d 3E	FD CB d 3E
'RLD'								ED 6F		
'RRD'								ED 67		

	A
RLCA	07
RRCA	0F
RLA	17
RRA	1F



ROTATES AND SHIFTS
TABLE 5.3–9

For example an unconditional Jump to memory location 3E32H would be:

Address A	C3	OP Code
A+1	32	Low order address
A+2	3E	High order address

The relative jump instruction uses only two bytes, the second byte is a signed two's complement displacement form the existing PC. This displacement can be in the range of +129 to -126 and is measured from the address of the instruction OP code.

Three types of register indirect jumps are also included. These instructions are implemented by loading the register pair HL or one of the index registers IX or IY directly into the PC. This capability allows for program jumps to be a function of previous calculations.

A call is a special form of a jump where the address of the byte following the call instruction is pushed onto the stack before the jump is made. A return instruction is the reverse of a call because the data on the top of the stack is popped directly into the PC to form a jump address. The call and return instructions allow for simple subroutine and interrupt handling. Two special return instructions have been included in the Z-80 family of components. The return from interrupt instruction (RETI) and the return from non maskable interrupt (RETN) are treated in the CPU as an unconditional return identical to the OP code C9H. The difference is that (RETI) can be used at the end of an interrupt routine and all Z-80 peripheral chips will recognize the execution of this instruction for proper control of nested priority interrupt handling. This instruction coupled with the Z-80 peripheral devices implementation simplifies the normal return from nested interrupt. Without this feature the following software sequence would be necessary to inform the interrupting device that the interrupt routine is completed:

Table 5.3-12 lists the eight OP codes for the restart instruction. This instruction is a single byte call to any of the eight addresses listed. The simple mnemonic for these eight calls is also shown. The value of this instruction is that frequently used routines can be called with this instruction to minimize memory usage.

		OP CODE	
CALL ADDRESS	0000_H	C7	'RST 0'
	0008_H	CF	'RST 8'
	0010_H	D7	'RST 16'
	0018_H	DF	'RST 24'
	0020_H	E7	'RST 32'
	0028_H	EF	'RST 40'
	0030_H	F7	'RST 48'
	0038_H	FF	'RST 56'

RESTART GROUP
TABLE 5.3–12

INPUT/OUTPUT

The Z-80 has an extensive set of Input and Output instructions as shown in table 5.3-13 and table 5.3-14. The addressing of the input or output device can be either absolute or register indirect, using the C register. Notice that in the register indirect addressing mode data can be transferred between the I/O devices and any of the internal registers. In addition eight block transfer instructions have been implemented. These instructions are similar to the memory block transfers except that they use register pair HL for a pointer to the memory source (output commands) or destination (input commands) while register B is used as a byte counter. Register C holds the address of the port for which the input or output command is desired. Since register B is eight bits in length, the I/O block transfer command handles up to 256 bytes.

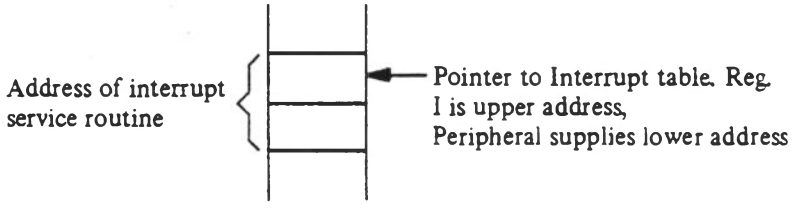
In the instructions IN A, n and OUT n, A the I/O device address n appears in the lower half of the address bus (A_0-A_7) while the accumulator content is transferred in the upper half of the address bus. In all register indirect input output instructions, including block I/O transfers the content of register C is transferred to the lower half of the address bus (device address) while the content of register B is transferred to the upper half of the address bus.

INPUT DESTINATION				PORT ADDRESS: IMMED. (n)	PORT ADDRESS: REG. INDIR. (C)	
INPUT 'IN'	REG ADDRESSING	A		DB	ED 78	
		B			ED 40	
		C			ED 48	
		D			ED 50	
		E			ED 58	
		H			ED 60	
		L			ED 68	
'INI' – INPUT & Inc HL, Dec B	REG. INDIR	(HL)			ED A2	BLOCK INPUT COMMANDS
'INIR'– INP, Inc HL, Dec B, REPEAT IF B≠0					ED B2	
'IND'– INPUT & Dec HL, Dec B					ED AA	
'INDR'– INPUT, Dec HL, Dec B, REPEAT IF B≠0					ED BA	

INPUT GROUP
TABLE 5.3–13

CPU CONTROL GROUP

The final table, table 5.3-15 illustrates the six general purpose CPU control instructions. The NOP is a do-nothing instruction. The HALT instruction suspends CPU operation until a subsequent interrupt is received, while the DI and EI are used to lock out and enable interrupts. The three interrupt mode commands set the CPU into any of the three available interrupt response modes as follows. If mode zero is set the interrupting device can insert any instruction on the data bus and allow the CPU to execute it. Mode 1 is a simplified mode where the CPU automatically executes a restart (RST) to location 0038H so that no external hardware is required. (The old PC content is pushed onto the stack). Mode 2 is the most powerful in that it allows for an indirect call to any location in memory. With this mode the CPU forms a 16-bit memory address where the upper 8-bits are the content of register I and the lower 8-bits are supplied by the interrupting device. This address points to the first of two sequential bytes in a table where the address of the service routine is located. The CPU automatically obtains the starting address and performs a CALL to this address.



			SOURCE							
			REGISTER							REG. IND.
			A	B	C	D	E	H	L	(HL)
'OUT'	IMMED.	(n)	D3							
	REG. IND.	(C)	ED 79	ED 41	ED 49	ED 51	ED 59	ED 61	ED 69	
'OUTI' – OUTPUT Inc HL, Dec b	REG. IND.	(C)								ED A3
'OTIR' – OUTPUT, Inc HL, Dec B, REPEAT IF B≠0	REG. IND.	(C)								ED B3
'OUTD' – OUTPUT Dec HL & B	REG. IND.	(C)								ED AB
'OTDR' – OUTPUT, Dec HL & B, REPEAT IF B≠0	REG. IND.	(C)								ED B8

BLOCK OUTPUT COMMANDS

PORT DESTINATION ADDRESS

OUTPUT GROUP
TABLE 5.3–14

'NOP'	00	
'HALT'	76	
DISABLE INT '(DI)'	F3	
ENABLE INT '(EI)'	FB	
SET INT MODE 0 'IM0'	ED 46	8080A MODE
SET INT MODE 1 'IM1'	ED 56	CALL TO LOCATION 0038_H
SET INT MODE 2 'IM2'	ED 5E	INDIRECT CALL USING REGISTER I AND 8 BITS FROM INTERRUPTING DEVICE AS A POINTER.

MISCELLANEOUS CPU CONTROL
TABLE 5.3–15

6.0 FLAGS

Each of the two Z-80 CPU Flag registers contains six bits of information which are set or reset by various CPU operations. Four of these bits are testable; that is, they are used as conditions for jump, call or return instructions. For example a jump may be desired only if a specific bit in the flag register is set. The four testable flag bits are:

1) Carry Flag (C) – This flag is the carry from the highest order bit of the accumulator. For example, the carry flag will be set during an add instruction where a carry from the highest bit of the accumulator is generated. This flag is also set if a borrow is generated during a subtraction instruction. The shift and rotate instructions also affect this bit.

2) Zero Flag (Z) – This flag is set if the result of the operation loaded a zero into the accumulator. Other wise it is reset.

3) Sign Flag (S) – This flag is intended to be used with signed numbers and it is set if the result of the operation was negative. Since bit 7 (MSB) represents the sign of the number (A negative number has a 1 in bit 7), this flag stores the state of bit 7 in the accumulator.

4) Parity/Overflow Flag (P/V) – This dual purpose flag indicates the parity of the result in the accumulator when logical operations are performed (such as AND A, B) and it represents overflow when signed two's complement arithmetic operations are performed. The Z-80 overflow flag indicates that the two's complement number in the accumulator is in error since it has exceeded the maximum possible (+127) or is less than the minimum possible (-128) number than can be represented in two's complement notation. For example consider adding:

```
+120 =    0111 1000
+105 =    0110 1001
-----    ----------
  C  =  0 1110 0001  = -95 (wrong) Overflow has occured
```

Here the result is incorrect. Overflow has occurred and yet there is no carry to indicate an error. For this case the overflow flag would be set. Also consider the addition of two negative numbers:

```
 -5 =    1111 1011
-16 =    1111 0000
-----    ----------
  C  =  1 1110 1011  = -21 correct
```

Notice that the answer is correct but the carry is set so that this flag can not be used as an overflow indicator. In this case the overflow would not be set.

For logical operations (AND, OR, XOR) this flag is set if the parity of the result is even and it is reset if it is odd.

There are also two non-testable bits in the flag register. Both of these are used for BCD arithmetic. They are:

1) Half carry (H) – This is the BCD carry or borrow result from the least significant four bits of operati When using the DAA (Decimal Adjust Instruction) this flag is used to correct the result of a previous packed decimal add or subtract.

2) Subtract Flag (N) – Since the algorithm for correcting BCD operations is different for addition or subtraction, this flag is used to specify what type of instruction was executed last so that the DAA operation will be correct for either addition or subtraction.

The Flag register can be accessed by the programmer and its format is as follows:

S	Z	X	H	X	P/V	N	C

X means flag is indeterminate.

Table 6.0-1 lists how each flag bit is affected by various CPU instructions. In this table a '•' indicates that the instruction does not change the flag, an 'X' means that the flag goes to an indeterminate state, a '0' means that it is reset, a '1' means that it is set and the symbol '↕' indicates that it is set or reset according to the previous discussion. Note that any instruction not appearing in this table does not affect any of the flags.

Table 6.0-1 includes a few special cases that must be described for clarity. Notice that the block search instruction sets the Z flag if the last compare operation indicated a match between the source and the accumulator data. Also, the parity flag is set if the byte counter (register pair BC) is not equal to zero. This same use of the parity flag is made with the block move instructions. Another special case is during block input or output instructions, here the Z flag is used to indicate the state of register B which is used as a byte counter. Notice that when the I/O block transfer is complete, the zero flag will be reset to a zero (i.e. B=0) while in the case of a block move command the parity flag is reset when the operation is complete. A final case is when the refresh or I register is loaded into the accumulator, the interrupt enable flip flop is loaded into the parity flag so that the complete state of the CPU can be saved at any time.

Instruction	C	Z	P/V	S	N	H	Comments
ADD A, s; ADC A,s	↕	↕	V	↕	0	↕	8-bit add or add with carry
SUB s; SBC A, s, CP s, NEG	↕	↕	V	↕	1	↕	8-bit subtract, subtract with carry, compare and negate accumulator
AND s	0	↕	P	↕	0	1	Logical operations
OR s; XOR s	0	↕	P	↕	0	0	And set's different flags
INC s	•	↕	V	↕	0	↕	8-bit increment
DEC m	•	↕	V	↕	1	↕	8-bit decrement
ADD DD, ss	↕	•	•	•	0	X	16-bit add
ADC HL, ss	↕	↕	V	↕	0	X	16-bit add with carry
SBC HL, ss	↕	↕	V	↕	1	X	16-bit subtract with carry
RLA; RLCA, RRA, RRCA	↕	•	•	•	0	0	Rotate accumulator
RL m; RLC m; RR m; RRC m SLA m; SRA m; SRL m	↕	↕	P	↕	0	0	Rotate and shift location s
RLD, RRD	•	↕	P	↕	0	0	Rotate digit left and right
DAA	↕	↕	P	↕	•	↕	Decimal adjust accumulator
CPL	•	•	•	•	1	1	Complement accumulator
SCF	1	•	•	•	0	0	Set carry
CCF	↕	•	•	•	0	X	Complement carry
IN r, (C)	•	↕	P	↕	0	0	Input register indirect
INI; IND; OUTI; OUTD	•	↕	X	X	1	X	Block input and output
INIR; INDR; OTIR; OTDR	•	1	X	X	1	X	Z = 0 if B ≠ 0 otherwise Z = 1
LDI, LDD	•	X	↕	X	0	0	Block transfer instructions
LDIR, LDDR	•	X	0	X	0	0	P/V = 1 if BC ≠ 0, otherwise P/V = 0
CPI, CPIR, CPD, CPDR	•	↕	↕	X	1	X	Block search instructions Z = 1 if A = (HL), otherwise Z = 0 P/V = 1 if BC ≠ 0, otherwise P/V = 0
LD A, I; LD A, R	•	↕	IFF	↕	0	0	The content of the interrupt enable flip-flop (IFF) is copied into the P/V flag
BIT b, s	•	↕	X	X	0	1	The state of bit b of location s is copied into the Z flag
NEG	↕	↕	V	↕	1	↕	Negate accumulator

The following notation is used in this table:

Symbol	Operation
C	Carry/link flag. C=1 if the operation produced a carry from the MSB of the operand or result.
Z	Zero flag. Z=1 if the result of the operation is zero.
S	Sign flag. S=1 if the MSB of the result is one.
P/V	Parity or overflow flag. Parity (P) and overflow (V) share the same flag. Logical operations affect this flag with the parity of the result while arithmetic operations affect this flag with the overflow of the result. If P/V holds parity, P/V=1 if the result of the operation is even, P/V=0 if result is odd. If P/V holds overflow, P/V=1 if the result of the operation produced an overflow.
H	Half-carry flag. H=1 if the add or subtract operation produced a carry into or borrow from into bit 4 of the accumulator.
N	Add/Subtract flag. N=1 if the previous operation was a subtract.
	H and N flags are used in conjunction with the decimal adjust instruction (DAA) to properly correct the result into packed BCD format following addition or subtraction using operands with packed BCD format.
↕	The flag is affected according to the result of the operation.
•	The flag is unchanged by the operation.
0	The flag is reset by the operation.
1	The flag is set by the operation.
X	The flag is a "don't care."
V	P/V flag affected according to the overflow result of the operation.
P	P/V flag affected according to the parity result of the operation.
r	Any one of the CPU registers A, B, C, D, E, H, L.
s	Any 8-bit location for all the addressing modes allowed for the particular instruction. .
ss	Any 16-bit location for all the addressing modes allowed for that instruction.
ii	Any one of the two index registers IX or IY.
R	Refresh counter.
n	8-bit value in range <0, 255>
nn	16-bit value in range <0, 65535>
m	Any 8-bit location for all the addressing modes allowed for the particular instruction.

SUMMARY OF FLAG OPERATION
TABLE 6.0-1

7.0 SUMMARY OF OP CODES AND EXECUTION TIMES

The following section gives a summary of the Z-80 instructions set. The instructions are logically arranged into groups as shown on tables 7.0-1 through 7.0-11. Each table shows the assembly language mnemonic OP code, the actual OP code, the symbolic operation, the content of the flag register following the execution of each instruction, the number of bytes required for each instruction as well as the number of memory cycles and the total number of T states (external clock periods) required for the fetching and execution of each instruction. Care has been taken to make each table self-explanatory without requiring any cross reference with the test or other tables.

Mnemonic	Symbolic Operation	Flags C	Z	P/V	S	N	H	OP-Code 76 543 210	No. of Bytes	No. of M Cycles	No. of T Cycles	Comments r, r'	Reg.
LD r, r'	r ← r'	•	•	•	•	•	•	01 r r'	1	1	4		
LD r, n	r ← n	•	•	•	•	•	•	00 r 110	2	2	7	000	B
								← n →				001	C
LD r, (HL)	r ← (HL)	•	•	•	•	•	•	01 r 110	1	2	7	010	D
LD r, (IX+d)	r ← (IX+d)	•	•	•	•	•	•	11 011 101	3	5	19	011	E
								01 r 110				100	H
								← d →				101	L
LD r, (IY+d)	r ← (IY+d)	•	•	•	•	•	•	11 111 101	3	5	19	111	A
								01 r 110					
								← d →					
LD (HL), r	(HL) ← r	•	•	•	•	•	•	01 110 r	1	2	7		
LD (IX+d), r	(IX+d) ← r	•	•	•	•	•	•	11 011 101	3	5	19		
								01 110 r					
								← d →					
LD (IY+d), r	(IY+d) ← r	•	•	•	•	•	•	11 111 101	3	5	19		
								01 110 r					
								← d →					
LD (HL), n	(HL) ← n	•	•	•	•	•	•	00 110 110	2	3	10		
								← n →					
LD (IX+d), n	(IX+d) ← n	•	•	•	•	•	•	11 011 101	4	5	19		
								00 110 110					
								← d →					
								← n →					
LD (IY+d), n	(IY+d) ← n	•	•	•	•	•	•	11 111 101	4	5	19		
								00 110 110					
								← d →					
								← n →					
LD A, (BC)	A ← (BC)	•	•	•	•	•	•	00 001 010	1	2	7		
LD A, (DE)	A ← (DE)	•	•	•	•	•	•	00 011 010	1	2	7		
LD A, (nn)	A ← (nn)	•	•	•	•	•	•	00 111 010	3	4	13		
								← n →					
								← n →					
LD (BC), A	(BC) ← A	•	•	•	•	•	•	00 000 010	1	2	7		
LD (DE), A	(DE) ← A	•	•	•	•	•	•	00 010 010	1	2	7		
LD (nn), A	(nn) ← A	•	•	•	•	•	•	00 110 010	3	4	13		
								← n →					
								← n →					
LD A, I	A ← I	•	‡	IFF	‡	0	0	11 101 101	2	2	9		
								01 010 111					
LD A, R	A ← R	•	‡	IFF	‡	0	0	11 101 101	2	2	9		
								01 011 111					
LD I, A	I ← A	•	•	•	•	•	•	11 101 101	2	2	9		
								01 000 111					
LD R, A	R ← A	•	•	•	•	•	•	11 101 101	2	2	9		
								01 001 111					

Notes: r, r' means any of the registers A, B, C, D, E, H, L

IFF the content of the interrupt enable flip-flop (IFF) is copied into the P/V flag

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,

‡ = flag is affected according to the result of the operation.

8-BIT LOAD GROUP
TABLE 7.0-1

Mnemonic	Symbolic Operation	Flags C	Z	P/V	S	N	H	Op-Code 76 543 210	No. of Bytes	No. of M Cycles	No. of T States	Comments
LD dd, nn	dd ← nn	•	•	•	•	•	•	00 dd0 001 ← n → ← n →	3	3	10	dd Pair 00 BC 01 DE
LD IX, nn	IX ← nn	•	•	•	•	•	•	11 011 101 00 100 001 ← n → ← n →	4	4	14	10 HL 11 SP
LD IY, nn	IY ← nn	•	•	•	•	•	•	11 111 101 00 100 001 ← n → ← n →	4	4	14	
LD HL, (nn)	H ← (nn+1) L ← (nn)	•	•	•	•	•	•	00 101 010 ← n → ← n →	3	5	16	
LD dd, (nn)	dd_H ← (nn+1) dd_L ← (nn)	•	•	•	•	•	•	11 101 101 01 dd1 011 ← n → ← n →	4	6	20	
LD IX, (nn)	IX_H ← (nn+1) IX_L ← (nn)	•	•	•	•	•	•	11 011 101 00 101 010 ← n → ← n →	4	6	20	
LD IY, (nn)	IY_H ← (nn+1) IY_L ← (nn)	•	•	•	•	•	•	00 101 010 00 101 010 ← n → ← n →	4	6	20	
LD (nn), HL	(nn+1) ← H (nn) ← L	•	•	•	•	•	•	00 100 010 ← n → ← n →	3	5	16	
LD (nn), dd	(nn+1) ← dd_H (nn) ← dd_L	•	•	•	•	•	•	11 101 101 01 dd0 011 ← n → ← n →	4	6	20	
LD (nn), IX	(nn+1) ← IX_H (nn) ← IX_L	•	•	•	•	•	•	11 011 101 00 100 010 ← n → ← n →	4	6	20	
LD (nn), IY	(nn+1) ← IY_H (nn) ← IY_L	•	•	•	•	•	•	11 111 101 00 100 010 ← n → ← n →	4	6	20	
LD SP, HL	SP ← HL	•	•	•	•	•	•	11 111 001	1	1	6	
LD SP, IX	SP ← IX	•	•	•	•	•	•	11 011 101 11 111 001	2	2	10	
LD SP, IY	SP ← IY	•	•	•	•	•	•	11 111 101 11 111 001	2	2	10	qq Pair
PUSH qq	(SP-2) ← qq_L (SP-1) ← qq_H	•	•	•	•	•	•	11 qq0 101	1	3	11	00 BC 01 DE
PUSH IX	(SP-2) ← IX_L (SP-1) ← IX_H	•	•	•	•	•	•	11 011 101 11 100 101	2	4	15	10 HL 11 AF
PUSH IY	(SP-2) ← IY_L (SP-1) ← IY_H	•	•	•	•	•	•	11 111 101 11 100 101	2	4	15	
POP qq	qq_H ← (SP+1) qq_L ← (SP)	•	•	•	•	•	•	11 qq0 001	1	3	10	
POP IX	IX_H ← (SP+1) IX_L ← (SP)	•	•	•	•	•	•	11 011 101 11 100 001	2	4	14	
POP IY	IY_H ← (SP+1) IY_L ← (SP)	•	•	•	•	•	•	11 111 101 11 100 001	2	4	14	

Notes: dd is any of the register pairs BC, DE, HL, SP
qq is any of the register pairs AF, BC, DE, HL
$(PAIR)_H$, $(PAIR)_L$ refer to high order and low order eight bits of the register pair respectively.
E.g. BC_L = C, AF_H = A

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown.
↕ flag is affected according to the result of the operation.

16-BIT LOAD GROUP
TABLE 7.0-2

Mnemonic	Symbolic Operation	C	Z	P/V	S	N	H	Op-Code 76 543 210	No. of Bytes	No. of M Cycles	No. of T States	Comments
EX DE, HL	DE ↔ HL	•	•	•	•	•	•	11 101 011	1	1	4	
EX AF, AF'	AF ↔ AF'	•	•	•	•	•	•	00 001 000	1	1	4	
EXX	(BC) (DE) (HL) ↔ (BC') (DE') (HL')	•	•	•	•	•	•	11 011 001	1	1	4	Register bank and auxiliary register bank exchange
EX (SP), HL	H ← (SP+1) L ← (SP)	•	•	•	•	•	•	11 100 011	1	5	19	
EX (SP), IX	IX_H ↔ (SP+1) IX_L ↔ (SP)	•	•	•	•	•	•	11 011 101 11 100 011	2	6	23	
EX (SP), IY	IY_H ↔ (SP+1) IY_L ↔ (SP)	•	•	•	•	•	•	11 111 101 11 100 011	2	6	23	
LDI	(DE) ← (HL) DE ← DE+1 HL ← HL+1 BC ← BC-1	•	•	↕ ①	•	0	0	11 101 101 10 100 000	2	4	16	Load (HL) into (DE), increment the pointers and decrement the byte counter (BC)
LDIR	(DE) ← (HL) DE ← DE+1 HL ← HL+1 BC ← BC-1 Repeat until BC = 0	•	•	0 ①	•	0	0	11 101 101 10 110 000	2 2	5 4	21 16	If BC ≠ 0 If BC = 0
LDD	(DE) ← (HL) DE ← DE-1 HL ← HL-1 BC ← BC-1	•	•	↕ ①	•	0	0	11 101 101 10 101 000	2	4	16	
LDDR	(DE) ← (HL) DE ← DE-1 HL ← HL-1 BC ← BC-1 Repeat until BC = 0	•	•	0 ①	•	0	0	11 101 101 10 111 000	2 2	5 4	21 16	If BC ≠ 0 If BC = 0
CPI	A - (HL) HL ← HL+1 BC ← BC-1	•	↕ ②	↕ ①	↕	1	↕	11 101 101 10 100 001	2	4	16	
CPIR	A - (HL) HL ← HL+1 BC ← BC-1 Repeat until A = (HL) or BC = 0	•	↕ ②	↕ ①	↕	1	↕	11 101 101 10 110 001	2 2	5 4	21 16	If BC ≠ 0 and A ≠ (HL) If BC = 0 or A = (HL)
CPD	A - (HL) HL ← HL-1 BC ← BC-1	•	↕ ②	↕ ①	↕	1	↕	11 101 101 10 101 001	2	4	16	
CPDR	A - (HL) HL ← HL-1 BC ← BC-1 Repeat until A = (HL) or BC = 0	•	↕ ②	↕ ①	↕	1	↕	11 101 101 10 111 001	2 2	5 4	21 16	If BC ≠ 0 and A ≠ (HL) If BC = 0 or A = (HL)

Notes: ① P/V flag is 0 if the result of BC-1 = 0, otherwise P/V = 1
② Z flag is 1 if A = (HL), otherwise Z = 0.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown.
↕ = flag is affected according to the result of the operation.

EXCHANGE GROUP AND BLOCK TRANSFER AND SEARCH GROUP

TABLE 7.0-3

Mnemonic	Symbolic Operation	Flags C	Z	P/V	S	N	H	Op-Code 76 543 210	No. of Bytes	No. of M Cycles	No. of T States	Comments
ADD A, r	A ← A + r	↕	↕	V	↕	0	↕	10 [000] r	1	1	4	r Reg. 000 B 001 C 010 D 011 E 100 H 101 L 111 A
ADD A, n	A ← A + n	↕	↕	V	↕	0	↕	11 [000] 110 ← n →	2	2	7	
ADD A, (HL)	A ← A + (HL)	↕	↕	V	↕	0	↕	10 [000] 110	1	2	7	
ADD A, (IX+d)	A←A + (IX+d)	↕	↕	V	↕	0	↕	11 011 101 10 [000] 110 ← d →	3	5	19	
ADD A, (IY+d)	A←A+(IY+d)	↕	↕	V	↕	0	↕	11 111 101 10 [000] 110 ← d →	3	5	19	
ADC A, s	A ← A + s + CY	↕	↕	V	↕	0	↕	[001]				s is any of r, n, (HL), (IX+d), (IY+d) as shown for ADD instruction
SUB s	A ← A − s	↕	↕	V	↕	1	↕	[010]				
SBC A, s	A ← A − s − CY	↕	↕	V	↕	1	↕	[011]				
AND s	A ← A ∧ s	0	↕	P	↕	0	1	[100]				The indicated bits replace the 000 in the ADD set above.
OR s	A ← A ∨ s	0	↕	P	↕	0	0	[110]				
XOR s	A ← A ∘ s	0	↕	P	↕	0	0	[101]				
CP s	A − s	↕	↕	V	↕	1	↕	[111]				
INC r	r ← r + 1	•	↕	V	↕	0	↕	00 r [100]	1	1	4	
INC (HL)	(HL) ← (HL)+1	•	↕	V	↕	0	↕	00 110 [100]	1	3	11	
INC (IX+d)	(IX+d) ← (IX+d)+1	•	↕	V	↕	0	↕	11 011 101 00 110 [100] ← d →	3	6	23	
INC (IY+d)	(IY+d) ← (IY+d) + 1	•	↕	V	↕	0	↕	11 111 101 00 110 [100] ← d →	3	6	23	
DEC m	m←m−1	•	↕	V	↕	1	↕	[101]				m is any of r, (HL), (IX+d), (IY+d) as shown for INC Same format and states as INC. Replace 100 with 101 in OP code.

Notes: The V symbol in the P/V flag column indicates that the P/V flag contains the overflow of the result of the operation. Similarly the P symbol indicates parity. V = 1 means overflow, V = 0 means not overflow. P = 1 means parity of the result is even, P = 0 means parity of the result is odd.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown.
↕ = flag is affected according to the result of the operation.

8-BIT ARITHMETIC AND LOGICAL GROUP
TABLE 7.0-4

Mnemonic	Symbolic Operation	Flags C	Z	P/V	S	N	H	Op-Code 76 543 210	No. of Bytes	No. of M Cycles	No. of T States	Comments
DAA	Converts acc. content into packed BCD following add or subtract with packed BCD operands	‡	‡	P	‡	•	‡	00 100 111	1	1	4	Decimal adjust accumulator
CPL	$A \leftarrow \overline{A}$	•	•	•	•	1	1	00 101 111	1	1	4	Complement accumulator (one's complement)
NEG	A ← 0 − A	‡	‡	V	‡	1	‡	11 101 101 01 000 100	2	2	8	Negate acc. (two's complement)
CCF	$CY \leftarrow \overline{CY}$	‡	•	•	•	0	X	00 111 111	1	1	4	Complement carry flag
SCF	CY ← 1	1	•	•	•	0	0	00 110 111	1	1	4	Set carry flag
NOP	No operation	•	•	•	•	•	•	00 000 000	1	1	4	
HALT	CPU halted	•	•	•	•	•	•	01 110 110	1	1	4	
DI	IFF ← 0	•	•	•	•	•	•	11 110 011	1	1	4	
EI	IFF ← 1	•	•	•	•	•	•	11 111 011	1	1	4	
IM 0	Set interrupt mode 0	•	•	•	•	•	•	11 101 101 01 000 110	2	2	8	
IM 1	Set interrupt mode 1	•	•	•	•	•	•	11 101 101 01 010 110	2	2	8	
IM2	Set interrupt mode 2	•	•	•	•	•	•	11 101 101 01 011 110	2	2	8	

Notes: IFF indicates the interrupt enable flip-flop
CY indicates the carry flip-flop.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
‡ = flag is affected according to the result of the operation.

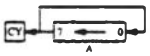
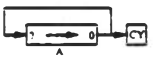
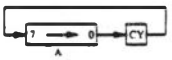
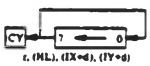
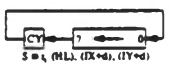
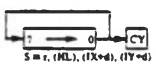
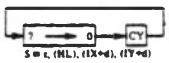
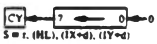
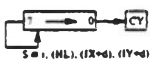
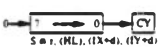
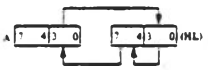
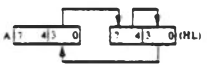
GENERAL PURPOSE ARITHMETIC AND CPU CONTROL GROUPS
TABLE 7.0-5

Mnemonic	Symbolic Operation	Flags						Op-Code	No. of Bytes	No. of M Cycles	No. of T States	Comments
		C	Z	P/V	S	N	H	76 543 210				
ADD HL, ss	HL ← HL+ss	↕	•	•	•	0	X	00 ss1 001	1	3	11	ss Reg. 00 BC 01 DE 10 HL 11 SP
ADC HL, ss	HL←HL+ss+CY	↕	↕	V	↕	0	X	11 101 101 01 ss1 010	2	4	15	
SBC HL, ss	HL←HL-ss-CY	↕	↕	V	↕	1	X	11 101 101 01 ss0 010	2	4	15	
ADD IX, pp	IX ← IX + pp	↕	•	•	•	0	X	11 011 101 00 pp1 001	2	4	15	pp Reg. 00 BC 01 DE 10 IX 11 SP
ADD IY, rr	IY←IY+rr	↕	•	•	•	0	X	11 111 101 00 rr1 001	2	4	15	rr Reg. 00 BC 01 DE 10 IY 11 SP
INC ss	ss ← ss + 1	•	•	•	•	•	•	00 ss0 011	1	1	6	
INC IX	IX ← IX + 1	•	•	•	•	•	•	11 011 101 00 100 011	2	2	10	
INC IY	IY ← IY + 1	•	•	•	•	•	•	11 111 101 00 100 011	2	2	10	
DEC ss	ss ← ss - 1	•	•	•	•	•	•	00 ss1 011	1	1	6	
DEC IX	IX ← IX - 1	•	•	•	•	•	•	11 011 101 00 101 011	2	2	10	
DEC IY	IY ← IY - 1	•	•	•	•	•	•	11 111 101 00 101 011	2	2	10	

Notes: ss is any of the register pairs BC, DE, HL, SP
pp is any of the register pairs BC, DE, IX, SP
rr is any of the register pairs BC, DE, IY, SP.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown.
↕ = flag is affected according to the result of the operation.

16-BIT ARITHMETIC GROUP
TABLE 7.0-6

Mnemonic	Symbolic Operation	Flags C	Z	P/V	S	N	H	Op-Code 76 543 210	No. of Bytes	No. of M Cycles	No. of T States	Comments
RLCA		‡	•	•	•	0	0	00 000 111	1	1	4	Rotate left circular accumulator
RLA		‡	•	•	•	0	0	00 010 111	1	1	4	Rotate left accumulator
RRCA		‡	•	•	•	0	0	00 001 111	1	1	4	Rotate right circular accumulator
RRA		‡	•	•	•	0	0	00 011 111	1	1	4	Rotate right accumulator
RLC r		‡	‡	P	‡	0	0	11 001 011 00 [000] r	2	2	8	Rotate left circular register r
RLC (HL)		‡	‡	P	‡	0	0	11 001 011 00 [000] 110	2	4	15	r Reg. 000 B 001 C 010 D 011 E 100 H 101 L 111 A
RLC (IX+d)		‡	‡	P	‡	0	0	11 011 101 11 001 011 ← d → 00 [000] 110	4	6	23	
RLC (IY+d)		‡	‡	P	‡	0	0	11 111 101 11 001 011 ← d → 00 [000] 110	4	6	23	
RL s		‡	‡	P	‡	0	0	[010]				Instruction format and states are as shown for RLC,s. To form new OP-code replace [000] of RLC,s with shown code
RRC s		‡	‡	P	‡	0	0	[001]				
RR s		‡	‡	P	‡	0	0	[011]				
SLA s		‡	‡	P	‡	0	0	[100]				
SRA s		‡	‡	P	‡	0	0	[101]				
SRL s		‡	‡	P	‡	0	0	[111]				
RLD		•	‡	P	‡	0	0	11 101 101 01 101 111	2	5	18	Rotate digit left and right between the accumulator and location (HL). The content of the upper half of the accumulator is unaffected
RRD		•	‡	P	‡	0	0	11 101 101 01 100 111	2	5	18	

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
‡ = flag is affected according to the result of the operation.

ROTATE AND SHIFT GROUP
TABLE 7.0-7

Mnemonic	Symbolic Operation	Flags C	Z	P/V	S	N	H	Op-Code	No. of Bytes	No. of M Cycles	No. of T States	Comments
BIT b, r	$Z \leftarrow \overline{r}_b$	•	‡	X	X	0	1	11 001 011 01 b r	2	2	8	
BIT b, (HL)	$Z \leftarrow \overline{(HL)}_b$	•	‡	X	X	0	1	11 001 011 01 b 110	2	3	12	
BIT b, (IX+d)	$Z \leftarrow \overline{(IX+d)}_b$	•	‡	X	X	0	1	11 011 101 11 001 011 ← d → 01 b 110	4	5	20	
BIT b, (IY+d)	$Z \leftarrow \overline{(IY+d)}_b$	•	‡	X	X	0	1	11 111 101 11 001 011 ← d → 01 b 110	4	5	20	
SET b, r	$r_b \leftarrow 1$	•	•	•	•	•	•	11 001 011 [11] b r	2	2	8	
SET b, (HL)	$(HL)_b \leftarrow 1$	•	•	•	•	•	•	11 001 011 [11] b 110	2	4	15	
SET b, (IX+d)	$(IX+d)_b \leftarrow 1$	•	•	•	•	•	•	11 011 101 11 001 011 ← d → [11] b 110	4	6	23	
SET b, (IY+d)	$(IY+d)_b \leftarrow 1$	•	•	•	•	•	•	11 111 101 11 001 011 ← d → [11] b 110	4	6	23	
RES b, s	$s_b \leftarrow 0$ s ≡ r, (HL), (IX+d), (IY+d)							[10]				To form new OP-code replace [11] of SET b,s with [10]. Flags and time states for SET instruction

r	Reg.
000	B
001	C
010	D
011	E
100	H
101	L
111	A

b	Bit Tested
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Notes: The notation s_b indicates bit b (0 to 7) or location s.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
‡ = flag is affected according to the result of the operation.

BIT SET, RESET AND TEST GROUP
TABLE 7.0-8

Mnemonic	Symbolic Operation	Flags C	Z	P/V	S	N	H	Op-Code 76 543 210	No. of Bytes	No. of M Cycles	No. of T States	Comments
JP nn	PC ← nn	•	•	•	•	•	•	11 000 011 ← n → ← n →	3	3	10	
JP cc, nn	If condition cc is true PC ← nn, otherwise continue	•	•	•	•	•	•	11 cc 010 ← n → ← n →	3	3	10	cc: 000 NZ non zero; 001 Z zero; 010 NC non carry; 011 C carry; 100 PO parity odd; 101 PE parity even; 110 P sign positive; 111 M sign negative
JR e	PC ← PC + e	•	•	•	•	•	•	00 011 000 ← e-2 →	2	3	12	
JR C, e	If C = 0, continue	•	•	•	•	•	•	00 111 000 ← e-2 →	2	2	7	If condition not met
	If C = 1, PC ← PC+e								2	3	12	If condition is met
JR NC, e	If C = 1, continue	•	•	•	•	•	•	00 110 000 ← e-2 →	2	2	7	If condition not met
	If C = 0, PC ← PC + e								2	3	12	If condition is met
JR Z, e	If Z = 0 continue	•	•	•	•	•	•	00 101 000 ← e-2 →	2	2	7	If condition not met
	If Z = 1, PC ← PC + e								2	3	12	If condition is met
JR NZ, e	If Z = 1, continue	•	•	•	•	•	•	00 100 000 ← e-2 →	2	2	7	If condition not met
	If Z = 0, PC ← PC + e								2	3	12	If condition met
JP (HL)	PC ← HL	•	•	•	•	•	•	11 101 001	1	1	4	
JP (IX)	PC ← IX	•	•	•	•	•	•	11 011 101 11 101 001	2	2	8	
JP (IY) ,	PC ← IY	•	•	•	•	•	•	11 111 101 11 101 001	2	2	8	
DJNZ,e	B ← B-1 If B = 0, continue	•	•	•	•	•	•	00 010 000 ← e-2 →	2	2	8	If B = 0
	If B ≠ 0, PC ← PC + e								2	3	13	IF B ≠ 0

Notes: e represents the extension in the relative addressing mode.

e is a signed two's complement number in the range <-126, 129>

e-2 in the op-code provides an effective address of pc +e as PC is incremented by 2 prior to the addition of e.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
↕ = flag is affected according to the result of the operation.

JUMP GROUP
TABLE 7.0-9

Mnemonic	Symbolic Operation	Flags C	Z	P/V	S	N	H	Op-Code 76 543 210	No. of Bytes	No. of M Cycles	No. of T States	Comments
CALL nn	$(SP-1)\leftarrow PC_H$ $(SP-2)\leftarrow PC_L$ $PC\leftarrow nn$	•	•	•	•	•	•	11 001 101 ← n → ← n →	3	5	17	
CALL cc, nn	If condition cc is false continue, otherwise same as CALL nn	•	•	•	•	•	•	11 cc 100 ← n → ← n →	3 3	3 5	10 17	If cc is false If cc is true
RET	$PC_L\leftarrow(SP)$ $PC_H\leftarrow(SP+1)$	•	•	•	•	•	•	11 001 001	1	3	10	
RET cc	If condition cc is false continue, otherwise same as RET	•	•	•	•	•	•	11 cc 000	1 1	1 3	5 11	If cc is false If cc is true
RETI	Return from interrupt	•	•	•	•	•	•	11 101 101 01 001 101	2	4	14	
RETN	Return from non maskable interrupt	•	•	•	•	•	•	11 101 101 01 000 101	2	4	14	
RST p	$(SP-1)\leftarrow PC_H$ $(SP-2)\leftarrow PC_L$ $PC_H\leftarrow 0$ $PC_L\leftarrow P$	•	•	•	•	•	•	11 t 111	1	3	11	

cc	Condition	
000	NZ	non zero
001	Z	zero
010	NC	non carry
011	C	carry
100	PO	parity odd
101	PE	parity even
110	P	sign positive
111	M	sign negative

t	P
000	00H
001	08H
010	10H
011	18H
100	20H
101	28H
110	30H
111	38H

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown
↕ = flag is affected according to the result of the operation.

CALL AND RETURN GROUP
TABLE 7.0-10

Mnemonic	Symbolic Operation	Flags C	Z	P/V	S	N	H	Op-Code 76 543 210	No. of Bytes	No. of M Cycles	No. of T States	Comments
IN A, (n)	A ← (n)	•	•	•	•	•	•	11 011 011 ← n →	2	3	11	n to A_0 ~ A_7 Acc to A_8 ~ A_{15}
IN r, (C)	r ← (C) if r = 110 only the flags will be affected	•	‡ ①	P	‡	0	‡	11 101 101 01 r 000	2	3	12	C to A_0 ~ A_7 B to A_8 ~ A_{15}
INI	(HL) ← (C) B ← B - 1 HL ← HL + 1	•	‡ ①	X	X	1	X	11 101 101 10 100 010	2	4	16	C to A_0 ~ A_7 B to A_8 ~ A_{15}
INIR	(HL) ← (C) B ← B - 1 HL ← HL + 1 Repeat until B = 0	•	1 ①	X	X	1	X	11 101 101 10 110 010	2 2	5 (If B ≠ 0) 4 (If B = 0)	21 16	C to A_0 ~ A_7 B to A_8 ~ A_{15}
IND	(HL) ← (C) B ← B - 1 HL ← HL - 1	•	‡ ①	X	X	1	X	11 101 101 10 101 010	2	4	16	C to A_0 ~ A_7 B to A_8 ~ A_{15}
INDR	(HL) ← (C) B ← B - 1 HL ← HL - 1 Repeat until B = 0	•	1 ①	X	X	1	X	11 101 101 10 111 010	2 2	5 (If B ≠ 0) 4 (If B = 0)	21 16	C to A_0 ~ A_7 B to A_8 ~ A_{15}
OUT (n), A	(n) ← A	•	•	•	•	•	•	11 010 011 ← n →	2	3	11	n to A_0 ~ A_7 Acc to A_8 ~ A_{15}
OUT (C), r	(C) ← r	•	• ①	•	•	•	•	11 101 101 01 r 001	2	3	12	C to A_0 ~ A_7 B to A_8 ~ A_{15}
OUTI	(C) ← (HL) B ← B - 1 HL ← HL + 1	•	‡ ①	X	X	1	X	11 101 101 10 100 011	2	4	16	C to A_0 ~ A_7 B to A_8 ~ A_{15}
OTIR	(C) ← (HL) B ← B - 1 HL ← HL + 1 Repeat until B = 0	•	1 ①	X	X	1	X	11 101 101 10 110 011	2 2	5 (If B ≠ 0) 4 (If B = 0)	21 16	C to A_0 ~ A_7 B to A_8 ~ A_{15}
OUTD	(C) ← (HL) B ← B - 1 HL ← HL - 1	•	‡ ①	X	X	1	X	11 101 101 10 101 011	2	4	16	C to A_0 ~ A_7 B to A_8 ~ A_{15}
OTDR	(C) ← (HL) B ← B - 1 HL ← HL - 1 Repeat until B = 0	•	1 ①	X	X	1	X	11 101 101 10 111 011	2 2	5 (If B ≠ 0) 4 (If B = 0)	21 16	C to A_0 ~ A_7 B to A_8 ~ A_{15}

Notes: ① If the result of B - 1 is zero the Z flag is set, otherwise it is reset.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
‡ = flag is affected according to the result of the operation.

INPUT AND OUTPUT GROUP
TABLE 7.0-11

8.0 INTERRUPT RESPONSE

The purpose of an interrupt is to allow peripheral devices to suspend CPU operation in an orderly manner and force the CPU to start a peripheral service routine. Usually this service routine is involved with the exchange of data, or status and control information, between the CPU and the peripheral. Once the service routine is completed, the CPU returns to the operation from which it was interrupted.

INTERRUPT ENABLE – DISABLE

The Z80 CPU has two interrupt inputs, a software maskable interrupt and a non maskable interrupt. The non maskable interrupt (NMI) can *not* be disabled by the programmer and it will be accepted whenever a peripheral device requests it. This interrupt is generally reserved for very important functions that must be serviced whenever they occur, such as an impending power failure. The maskable interrupt (INT) can be selectively enabled or disabled by the programmer. This allows the programmer to disable the interrupt during periods where his program has timing constraints that do not allow it to be interrupted. In the Z80 CPU there is an enable flip flop (called IFF) that is set or reset by the programmer using the Enable Interrupt (EI) and Disable Interrupt (DI) instructions. When the IFF is reset, an interrupt can not be accepted by the CPU.

Actually, for purposes that will be subsequently explained, there are two enable flip flops, called IFF_1 and IFF_2.

IFF_1	IFF_2
Actually disables interrupts from being accepted.	Temporary storage location for IFF_1.

The state of IFF_1 is used to actually inhibit interrupts while IFF_2 is used as a temporary storage location for IFF_1. The purpose of storing the IFF_1 will be subsequently explained.

A reset to the CPU will force both IFF_1 and IFF_2 to the reset state so that interrupts are disabled. They can then be enabled by an EI instruction at any time by the programmer. When an EI instruction is executed, any pending interrupt request will not be accepted until after the instruction following EI has been executed. This single instruction delay is necessary for cases when the following instruction is a return instruction and interrupts must not be allowed until the return has been completed. The EI instruction sets both IFF_1 and IFF_2 to the enable state. When an interrupt is accepted by the CPU, both IFF_1 and IFF_2 are automatically reset, inhibiting further interrupts until the programmer wishes to issue a new EI instruction. Note that for all of the previous cases, IFF_1 and IFF_2 are always equal.

The purpose of IFF_2 is to save the status of IFF_1 when a non maskable interrupt occurs. When a non maskable interrupt is accepted, IFF_1 is reset to prevent further interrupts until reenabled by the programmer. Thus, after a non maskable interrupt has been accepted, maskable interrupts are disabled but the previous state of IFF_1 has been saved so that the complete state of the CPU just prior to the non maskable interrupt can be restored at any time. When a Load Register A with Register I (LD A, I) instruction or a Load Register A with Register R (LD A, R) instruction is executed, the state of IFF_2 is copied into the parity flag where it can be tested or stored.

A second method of restoring the status of IFF_1 is thru the execution of a Return From Non Maskable Interrupt (RETN) instruction. Since this instruction indicates that the non maskable interrupt service routine is complete, the contents of IFF_2 are now copied back into IFF_1, so that the status of IFF_1 just prior to the acceptance of the non maskable interrupt will be restored automatically.

Figure 8.0-1 is a summary of the effect of different instructions on the two enable flip flops.

Action	IFF_1	IFF_2	
CPU Reset	0	0	
DI	0	0	
EI	1	1	
LD A, I	•	•	$IFF_2 \rightarrow$ Parity flag
LD A, R	•	•	$IFF_2 \rightarrow$ Parity flag
Accept NMI	0	•	
RETN	IFF_2	•	$IFF_2 \rightarrow IFF_1$

"•" indicates no change

FIGURE 8.0-1
INTERRUPT ENABLE/DISABLE FLIP FLOPS

CPU RESPONSE

Non Maskable

A nonmaskable interrupt will be accepted at all times by the CPU. When this occurs, the CPU ignores the next instruction that it fetches and instead does a restart to location 0066H. Thus, it behaves exactly as if it had received a restart instruction but, it is to a location that is not one of the 8 software restart locations. A restart is merely a call to a specific address in page 0 of memory.

Maskable

The CPU can be programmed to respond to the maskable interrupt in any one of three possible modes.

Mode 0

This mode is identical to the 8080A interrupt response mode. With this mode, the interrupting device can place any instruction on the data bus and the CPU will execute it. Thus, the interrupting device provides the next instruction to be executed instead of the memory. Often this will be a restart instruction since the interrupting device only need supply a single byte instruction. Alternatively, any other instruction such as a 3 byte call to any location in memory could be executed.

The number of clock cycles necessary to execute this instruction is 2 more than the normal number for the instruction. This occurs since the CPU automatically adds 2 wait states to an interrupt response cycle to allow sufficient time to implement an external daisy chain for priority control. Section 5.0 illustrates the detailed timing for an interrupt response. After the application of RESET the CPU will automatically enter interrupt Mode 0.

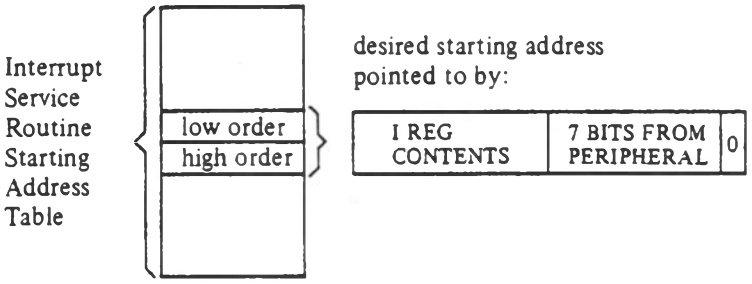
Mode 1

When this mode has been selected by the programmer, the CPU will respond to an interrupt by executing a restart to location 0038H. Thus the response is identical to that for a non maskable interrupt except that the call location is 0038H instead of 0066H. Another difference is that the number of cycles required to complete the restart instruction is 2 more than normal due to the two added wait states.

Mode 2

This mode is the most powerful interrupt response mode. With a single 8 bit byte from the user an indirect call can be made to any memory location.

With this mode the programmer maintains a table of 16 bit starting addresses for every interrupt service routine. This table may be located anywhere in memory. When an interrupt is accepted, a 16 bit pointer must be formed to obtain the desired interrupt service routine starting address from the table. The upper 8 bits of this pointer is formed from the contents of the I register. The I register must have been previously loaded with the desired value by the programmer, i.e. LD I, A. Note that a CPU reset clears the I register so that it is initialized to zero. The lower eight bits of the pointer must be supplied by the interrupting device. Actually, only 7 bits are required from the interrupting device as the least significant bit must be a zero. This is required since the pointer is used to get two adjacent bytes to form a complete 16 bit service routine starting address and the addresses must always start in even locations.



The first byte in the table is the least significant (low order) portion of the address. The programmer must obviously fill this table in with the desired addresses before any interrupts are to be accepted.

Note that this table can be changed at any time by the programmer (if it is stored in Read/Write Memory) to allow different peripherals to be serviced by different service routines.

Once the interrupting devices supplies the lower portion of the pointer, the CPU automatically pushes the program counter onto the stack, obtains the starting address from the table and does a jump to this address. This mode of response requires 19 clock periods to complete (7 to fetch the lower 8 bits from the interrupting device, 6 to save the program counter, and 6 to obtain the jump address.)

Note that the Z80 peripheral devices all include a daisy chain priority interrupt structure that automatically supplies the programmed vector to the CPU during interrupt acknowledge. Refer to the Z80-PIO, Z80-SIO and Z80-CTC manuals for details.

9.0 HARDWARE IMPLEMENTATION EXAMPLES

This chapter is intended to serve as a basic introduction to implementing systems with the Z80-CPU.

MINIMUM SYSTEM

Figure 9.0-1 is a diagram of a very simple Z-80 system. Any Z-80 system must include the following five elements:

1) Five volt power supply
2) Oscillator
3) Memory devices
4) I/O circuits
5) CPU

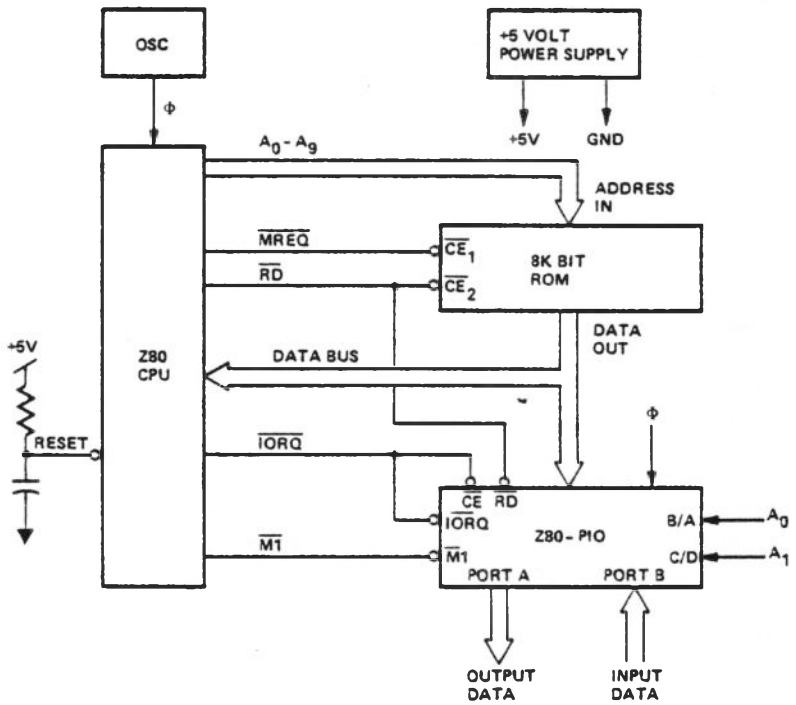


FIGURE 9.0-1
MINIMUM Z80 COMPUTER SYSTEM

Since the Z80-CPU only requires a single 5 volt supply, most small systems can be implemented using only this single supply.

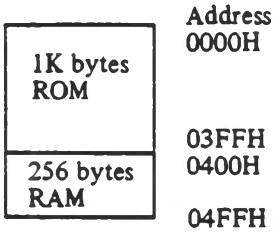
The oscillator can be very simple since the only requirement is that it be a 5 volt square wave. For systems not running at full speed, a simple RC oscillator can be used. When the CPU is operated near the highest possible frequency, a crystal oscillator is generally required because the system timing will not tolerate the drift or jitter that an RC network will generate. A crystal oscillator can be made from inverters and a few discrete components or monolithic circuits are widely available.

The external memory can be any mixture of standard RAM, ROM, or PROM. In this simple example we have shown a single 8K bit ROM (1K bytes) being utilized as the entire memory system. For this example we have assumed that the Z-80 internal register configuration contains sufficient Read/Write storage so that external RAM memory is not required.

Every computer system requires I/O circuits to allow it to interface to the "real world." In this simple example it is assumed that the output is an 8 bit control vector and the input is an 8 bit status word. The input data could be gated onto the data bus using any standard tri-state driver while the output data could be latched with any type of standard TTL latch. For this example we have used a Z80-PIO for the I/O circuit. This single circuit attaches to the data bus as shown and provides the required 16 bits of TTL compatible I/O. (Refer to the Z80-PIO manual for details on the operation of this circuit.) Notice in this example that with only three LSI circuits, a simple oscillator and a single 5 volt power supply, a powerful computer has been implemented.

ADDING RAM

Most computer systems require some amount of external Read/Write memory for data storage and to implement a "stack." Figure 9.0-2 illustrates how 256 bytes of static memory can be added to the previous example. In this example the memory space is assumed to be organized as follows:



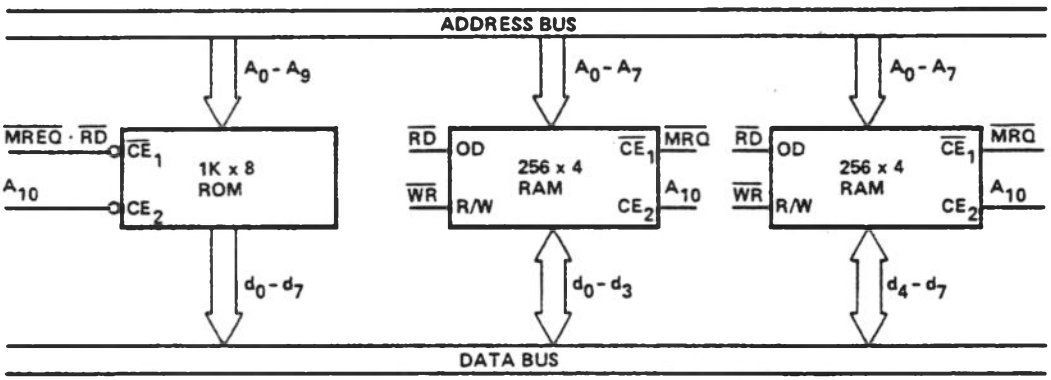


FIGURE 9.0-2
ROM & RAM IMPLEMENTATION EXAMPLE

In this diagram the address space is described in hexidecimal notation. For this example, address bit A_{10} separates the ROM space from the RAM space so that it can be used for the chip select function. For larger amounts of external ROM or RAM, a simple TTL decoder will be required to form the chip selects.

MEMORY SPEED CONTROL

For many applications, it may be desirable to use slow memories to reduce costs. The $\overline{WAIT}$ line on the CPU allows the Z-80 to operate with any speed memory. By referring back to section 4 you will notice that the memory access time requirements are most severe during the M1 cycle instruction fetch. All other memory accesses have an additional one half of a clock cycle to be completed. For this reason it may be desirable in some applications to add one wait state to the M1 cycle so that slower memories can be used. Figure 9.0-3 is an example of a simple circuit that will accomplish this task. This circuit can be changed to add a single wait state to any memory access as shown in Figure 9.0-4.

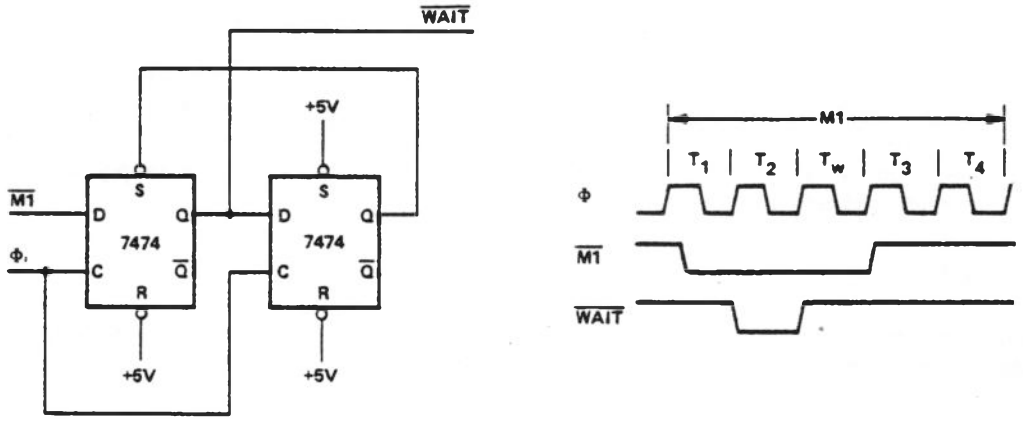


FIGURE 9.0-3
ADDING ONE WAIT STATE TO AN M1 CYCLE

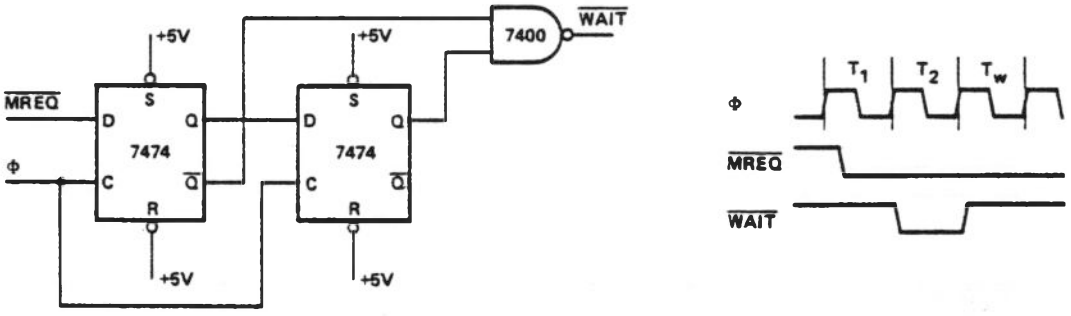


FIGURE 9.0-4
ADDING ONE WAIT STATE TO ANY MEMORY CYCLE

INTERFACING DYNAMIC MEMORIES

This section is intended only to serve as a brief introduction to interfacing dynamic memories. Each individual dynamic RAM has varying specifications that will require minor modifications to the description given here and no attempt will be made in this document to give details for any particular RAM. Separate application notes showing how the Z80-CPU can be interfaced to most popular dynamic RAM's are available from Zilog.

Figure 9.0-5 illustrates the logic necessary to interface 8K bytes of dynamic RAM using 18 pin 4K dynamic memories. This figure assumes that the RAM's are the only memory in the system so that A_{12} is used to select between the two pages of memory. During refresh time, all memories in the system must be read. The CPU provides the proper refresh address on lines A_0 through A_6. To add additional memory to the system it is necessary to only replace the two gates that operate on A_{12} with a decoder that operates on all required address bits. For larger systems, buffering for the address and data bus is also generally required.

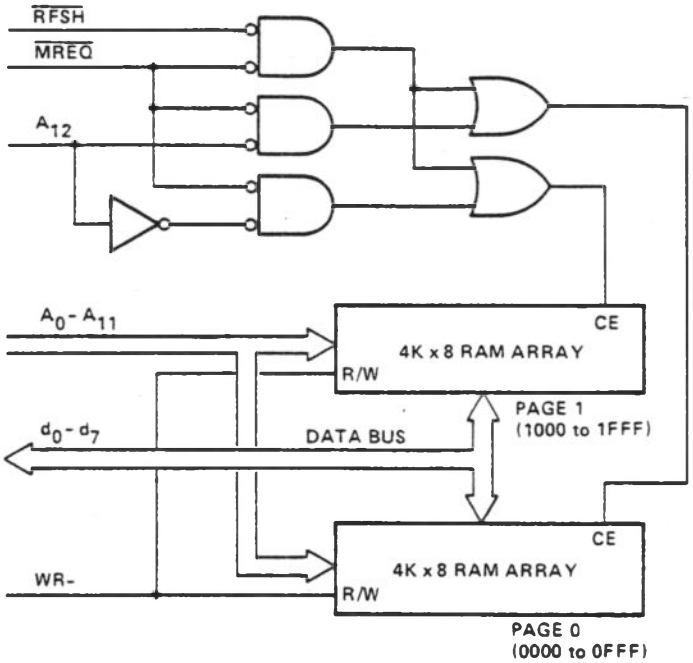


FIGURE 9.0-5
INTERFACING DYNAMIC RAMS

10.0 SOFTWARE IMPLEMENTATION EXAMPLES

10.1 METHODS OF SOFTWARE IMPLEMENTATION

Several different approaches are possible in developing software for the Z-80 (Figure 10.1). First of all, Assembly Language or PL/Z may be used as the source language. These languages may then be translated into machine language on a commercial time sharing facility using a cross-assembler or cross-compiler or, in the case of assembly language, the translation can be accomplished on a Z-80 Development System using a resident assembler. Finally, the resulting machine code can be debugged either on a time-sharing facility using a Z-80 simulator or on a Z-80 Development System which uses a Z80-CPU directly.

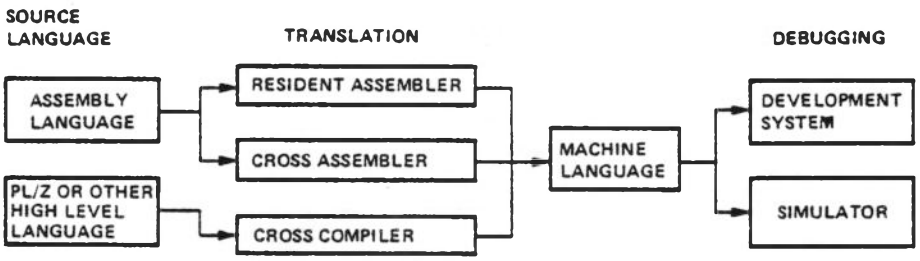


FIGURE 10.1

In selecting a source language, the primary factors to be considered are clarity and ease of programming vs. code efficiency. A high level language such as PL/Z with its machine independent constructs is typically better for formulating and maintaining algorithms, but the resulting machine code is usually somewhat less efficient than what can be written directly in assembly language. These tradeoffs can often be balanced by combining PL/Z and assembly language routines, identifying those portions of a task which must be optimized and writing them as assembly language subroutines.

Deciding whether to use a resident or cross assembler is a matter of availability and short-term vs. long-term expense. While the initial expenditure for a development system is higher than that for a time-sharing terminal, the cost of an individual assembly using a resident assembler is negligible while the same operation on a time-sharing system is relatively expensive and in a short time this cost can equal the total cost of a development system.

Debugging on a development system vs. a simulator is also a matter of availability and expense combined with operational fidelity and flexibility. As with the assembly process, debugging is less expensive on a development system than on a simulator available through time-sharing. In addition, the fidelity of the operating environment is preserved through real-time execution on a Z80-CPU and by connecting the I/O and memory components which will actually be used in the production system. The only advantage to the use of a simulator is the range of criteria which may be selected for such debugging procedures as tracing and setting breakpoints. This flexibility exists because a software simulation can achieve any degree of complexity in its interpretation of machine instructions while development system procedures have hardware limitations such as the capacity of the real-time storage module, the number of breakpoint registers and the pin configuration of the CPU. Despite such hardware limitations, debugging on a development system is typically more productive than on a simulator because of the direct interaction that is possible between the programmer and the authentic execution of his program.

10.2 SOFTWARE FEATURES OFFERED BY THE Z80-CPU

The Z-80 instruction set provides the user with a large and flexible repetoire of operations with which to formulate control of the Z80-CPU.

The primary, auxiliary and index registers can be used to hold the arguments of arithmetic and logical operations, or to form memory addresses, or as fast-access storage for frequently used data.

Information can be moved directly from register to register; from memory to memory; from memory to registers; or from registers to memory. In addition, register contents and register/memory contents can be exchanged without using temporary storage. In particular, the contents of primary and auxilary registers can be completely exchanged by executing only two instructions, EX and EXX. This register exchange procedure can be used to separate the set of working registers between different logical procedures or to expand the set of available registers in a single procedure.

Storage and retrieval of data between pairs of registers and memory can be controlled on a last-in first-out basis through PUSH and POP instructions which utilize a special stack pointer register, SP. This stack register is available both to manipulate data and to automatically store and retrieve addresses for subroutine linkage. When a subroutine is called, for example, the address following the CALL instruction is placed on the top of the push-down stack pointed to by SP. When a subroutine returns to the calling routine, the address on the top of the stack is used to set the program counter for the address of the next instruction. The stack pointer is adjusted automatically to reflect the current "top" stack position during PUSH, POP, CALL and RET instructions. This stack mechanism allows pushdown data stacks and subroutine calls to be nested to any practical depth because the stack area can potentially be as large as memory space.

The sequence of instruction execution can be controlled by six different flags (carry, zero, sign, parity/overflow, add-subtract, half-carry) which reflect the results of arithmetic, logical, shift and compare instructions. After the execution of an instruction which sets a flag, that flag can be used to control a conditional jump or return instruction. These instructions provide logical control following the manipulation of single bit, eight-bit byte (or) sixteen-bit data quantities.

A full set of logical operations, including AND, OR, XOR (exclusive - OR), CPL (NOR) and NEG (two's complement) are available for Boolean operations between the accumulator and 1) all other eight-bit registers, 2) memory locations or 3) immediate operands.

In addition, a full set of arithmetic and logical shifts in both directions are available which operate on the contents of all eight-bit primary registers or directly on any memory location. The carry flag can be included or simply set by these shift instructions to provide both the testing of shift results and to link register/register or register/memory shift operations.

10.3 EXAMPLES OF USE OF SPECIAL Z80 INSTRUCTIONS

A. Let us assume that a string of data in memory starting at location "DATA" is to be moved into another area of memory starting at location "BUFFER" and that the string length is 737 bytes. This operation can be accomplished as follows:

```
LD      HL , DATA       ; START ADDRESS OF DATA STRING
LD      DE , BUFFER     ; START ADDRESS OF TARGET BUFFER
LD      BC , 737        ; LENGTH OF DATA STRING
LDIR                    ; MOVE STRING – TRANSFER MEMORY POINTED TO
                        ; BY HL INTO MEMORY LOCATION POINTED TO BY DE
                        ; INCREMENT HL AND DE, DECREMENT BC
                        ; PROCESS UNTIL BC = 0.
```

11 bytes are required for this operation and each byte of data is moved in 21 clock cycles.

B. Let's assume that a string in memory starting at location "DATA" is to be moved into another area of memory starting at location "BUFFER" until an ASCII $ character (used as string delimiter) is found. Let's also assume that the maximum string length is 132 characters. The operation can be performed as follows:

```
        LD      HL , DATA       ; STARTING ADDRESS OF DATA STRING
        LD      DE , BUFFER     ; STARTING ADDRESS OF TARGET BUFFER
        LD      BC , 132        ; MAXIMUM STRING LENGTH
        LD      A ,'$'          ; STRING DELIMITER CODE
LOOP:CP         (HL)            ; COMPARE MEMORY CONTENTS WITH DELIMITER
        JR      Z , END         ; GO TO END IF CHARACTERS EQUAL
        LDI                     ; MOVE CHARACTER (HL) to (DE)
                                ; INCREMENT HL AND DE, DECREMENT BC
        JP      PE , LOOP       ; GO TO "LOOP" IF MORE CHARACTERS
END:                            ; OTHERWISE, FALL THROUGH
                                ; NOTE: P/V FLAG IS USED
                                ; TO INDICATE THAT REGISTER BC WAS
                                ; DECREMENTED TO ZERO.
```

19 bytes are required for this operation.

C. Let us assume that a 16-digit decimal number represented in packed BCD format (two BCD digits/ byte) has to be shifted as shown in the Figure 10.2 in order to mechanize BCD multiplication or division. The operation can be accomplished as follows:

```
        LD      HL , DATA       ; ADDRESS OF FIRST BYTE
        LD      B , COUNT       ; SHIFT COUNT
        XOR     A               ; CLEAR ACCUMULATOR
ROTAT: RLD                      ; ROTATE LEFT LOW ORDER DIGIT IN ACC
                                ; WITH DIGITS IN (HL)
        INC     HL              ; ADVANCE MEMORY POINTER
        DJNZ    ROTAT           ; DECREMENT B AND GO TO ROTAT IF
                                ; B IS NOT ZERO, OTHERWISE FALL THROUGH
```

11 bytes are required for this operation.

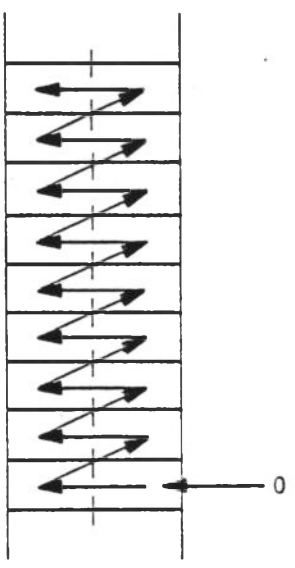


FIGURE 10.2

D. Let us assume that one number is to be subtracted from another and a) that they are both in packed BCD format, b) that they are of equal but varying length, and c) that the result is to be stored in the location of the minuend. The operation can be accomplished as follows:

```
        LD      HL , ARG1       ; ADDRESS OF MINUEND
        LD      DE , ARG2       ; ADDRESS OF SUBTRAHEND
        LD      B , LENGTH      ; LENGTH OF TWO ARGUMENTS
        AND     A               ; CLEAR CARRY FLAG
SUBDEC: LD      A , (DE)        ; SUBTRAHEND TO ACC
        SBC     A , (HL)        ; SUBTRACT (HL) FROM ACC
        DAA                     ; ADJUST RESULT TO DECIMAL CODED VALUE
        LD      (HL) , A        ; STORE RESULT
        INC     HL              ; ADVANCE MEMORY POINTERS
        INC     DE
        DJNZ    SUBDEC          ; DECREMENT B AND GO TO "SUBDEC" IF B
                                ; NOT ZERO, OTHERWISE FALL THROUGH
```

17 bytes are required for this operation.

10.4 EXAMPLES OF PROGRAMMING TASKS

A. The following program sorts an array of numbers each in the range ⟨0,255⟩ into ascending order using a standard exchange sorting algorithm.

```
LOC     OBJ CODE    STMT    SOURCE STATEMENT

                      1  ;  *** STANDARD EXCHANGE (BUBBLE) SORT ROUTINE ***
                      2  ;
                      3  ;  AT ENTRY:  HL CONTAINS ADDRESS OF DATA
                      4                C CONTAINS NUMBER OF ELEMENTS TO BE SORTED
                      5                (1<C<256)
                      6  ;
                      7  ;  AT EXIT: DATA SORTED IN ASCENDING ORDER
                      8  ;
                      9  ;  USE OF REGISTERS
                     10  ;
                     11  ;  REGISTER   CONTENTS
                     12  ;
                     13  ;  A          TEMPORARY STORAGE FOR CALCULATIONS
                     14  ;  B          COUNTER FOR DATA ARRAY
                     15  ;  C          LENGTH OF DATA ARRAY
                     16  ;  D          FIRST ELEMENT IN COMPARISON
                     17  ;  E          SECOND ELEMENT IN COMPARISON
                     18  ;  H          FLAG TO INDICATE EXCHANGE
                     19  ;  L          UNUSED
                     20  ;  IX         POINTER INTO DATA ARRAY
                     21  ;  IY         UNUSED
                     22  ;
0000    222600       23  SORT:   LD      (DATA), HL      ;SAVE DATA ADDRESS
0003    CB84         24  LOOP:   RES     FLAG, H         ;INITIALIZE EXCHANGE FLAG
0005    41           25          LD      B, C            ;INITIALIZE LENGTH COUNTER
0006    05           26          DEC     B               ;ADJUST FOR TESTING
0007    DD2A2600     27          LD      IX, (DATA)      ;INITIALIZE ARRAY POINTER
000B    DD7E00       28  NEXT:   LD      A, (IX)         ;FIRST ELEMENT IN COMPARISON
000E    57           29          LD      D, A            ;TEMPORARY STORAGE FOR ELEMENT
000F    DD5E01       30          LD      E, (IX+1)       ;SECOND ELEMENT IN COMPARISON
0012    93           31          SUB     E               ;COMPARISON FIRST TO SECOND
0013    3008         32          JR      NC, NOEX        ;IF FIRST > SECOND, NO JUMP
0015    DD7300       33          LD      (IX), E         ;EXCHANGE ARRAY ELEMENTS
0018    DD7201       34          LD      (IX+1), D
001B    CBC4         35          SET     FLAG, H         ;RECORD EXCHANGE OCCURRED
001D    DD23         36  NOEX:   INC     IX              ;POINT TO NEXT DATA ELEMENT
001F    10EA         37          DJNZ    NEXT            ;COUNT NUMBER OF COMPARISONS
                     38                                  ;REPEAT IF MORE DATA PAIRS
0021    CB44         39          BIT     FLAG, H         ;DETERMINE IF EXCHANGE OCCURRED
0023    20DE         40          JR      NZ, LOOP        ;CONTINUE IF DATA UNSORTED
0025    C9           41          RET                     ;OTHERWISE, EXIT
                     42  ;
0026                 43  FLAG:   EQU     0               ;DESIGNATION OF FLAG BIT
0026                 44  DATA:   DEFS    2               ;STORAGE FOR DATA ADDRESS
                     45          END
```

B. The following program multiplies two unsigned 16 bit integers and leaves the result in the HL register pair.

```
01/22/76     11:32:36                MULTIPLY LISTING                PAGE 1
LOC    OBJ CODE    STMT    SOURCE STATEMENT

0000                 1  MULT:;   UNSIGNED SIXTEEN BIT INTEGER MULTIPLY.
                     2  ;        ON ENTRANCE:  MULTIPLIER IN DE.
                     3  ;                      MULTIPLICAND IN HL.
                     4  ;
                     5  ;        ON EXIT:  RESULT IN HL.
                     6  ;
                     7  ;        REGISTER USES:
                     8  ;
                     9  ;
                    10  ;        H      HIGH ORDER PARTIAL RESULT
                    11  ;        L      LOW ORDER PARTIAL RESULT
                    12  ;        D      HIGH ORDER MULTIPLICAND
                    13  ;        E      LOW ORDER MULTIPLICAND
                    14  ;        B      COUNTER FOR NUMBER OF SHIFTS
                    15  ;        C      HIGH ORDER BITS OF MULTIPLIER
                    16  ;        A      LOW ORDER BITS OF MULTIPLIER
                    17  ;
0000   0610         18           LD     B, 16;        NUMBER OF BITS- INITIALIZE
0002   4A           19           LD     C, D;         MOVE MULTIPLIER
0003   7B           20           LD     A, E;
0004   EB           21           EX     DE, HL;       MOVE MULTIPLICAND
0005   210000       22           LD     HL, 0;        CLEAR PARTIAL RESULT
0008   CB39         23  MLOOP:   SRL    C;            SHIFT MULTIPLIER RIGHT
000A   1F           24           RR     A;            LEAST SIGNIFICANT BIT IS
                    25  ;                             IN CARRY.
000B   3001         26           JR     NC, NOADD     IF NO CARRY, SKIP THE ADD.
000D   19           27           ADD    HL, DE;       ELSE ADD MULTIPLICAND TO
                    28  ;                             PARTIAL RESULT.
000E   EB           29  NOADD:   EX     DE, HL;       SHIFT MULTIPLICAND LEFT
000F   29           30           ADD    HL, HL;       BY MULTIPLYING IT BY TWO.
0010   EB           31           EX     DE, HL;
0011   10F5         32           DJNZ   MLOOP         REPEAT UNTIL NO MORE BITS.
0013   C9           33           RET;
                    34           END;
```

Absolute Maximum Ratings

Temperature Under Bias	Specified operating range.
Storage Temperature	-65°C to +150°C
Voltage On Any Pin with Respect to Ground	-0.3V to +7V
Power Dissipation	1.5W

*Comment

Stresses above those listed under "Absolute Maximum Rating" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other condition above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

Note: For Z80-CPU all AC and DC characteristics remain the same for the military grade parts except I_{cc}.

I_{cc} = 200 mA

Z80-CPU D.C. Characteristics

T_A = 0°C to 70°C, V_{cc} = 5V ± 5% unless otherwise specified

Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Condition
V_{ILC}	Clock Input Low Voltage	-0.3		0.45	V	
V_{IHC}	Clock Input High Voltage	V_{cc} -.6		V_{cc}+.3	V	
V_{IL}	Input Low Voltage	-0.3		0.8	V	
V_{IH}	Input High Voltage	2.0		V_{cc}	V	
V_{OL}	Output Low Voltage			0.4	V	I_{OL}=1.8mA
V_{OH}	Output High Voltage	2.4			V	I_{OH} = -250μA
I_{CC}	Power Supply Current			150	mA	
I_{LI}	Input Leakage Current			10	μA	V_{IN}=0 to V_{cc}
I_{LOH}	Tri-State Output Leakage Current in Float			10	μA	V_{OUT}=2.4 to V_{cc}
I_{LOL}	Tri-State Output Leakage Current in Float			-10	μA	V_{OUT}=0.4V
I_{LD}	Data Bus Leakage Current in Input Mode			±10	μA	$0 \leq V_{IN} \leq V_{cc}$

Capacitance

T_A = 25°C, f = 1 MHz,
unmeasured pins returned to ground

Symbol	Parameter	Max.	Unit
C_ϕ	Clock Capacitance	35	pF
C_{IN}	Input Capacitance	5	pF
C_{OUT}	Output Capacitance	10	pF

Z80-CPU Ordering Information

C – Ceramic
P – Plastic
S – Standard 5V ±5% 0° to 70°C
E – Extended 5V ±5% -40° to 85°C
M – Military 5V ±10% -55° to 125°C

Z80A-CPU D.C. Characteristics

T_A = 0°C to 70°C, V_{cc} = 5V ± 5% unless otherwise specified

Symbol	Parameter	Min.	Typ.	Max.	Unit	Test Condition
V_{ILC}	Clock Input Low Voltage	-0.3		0.45	V	
V_{IHC}	Clock Input High Voltage	V_{cc} -.6		V_{cc}+.3	V	
V_{IL}	Input Low Voltage	-0.3		0.8	V	
V_{IH}	Input High Voltage	2.0		V_{cc}	V	
V_{OL}	Output Low Voltage			0.4	V	I_{OL}=1.8mA
V_{OH}	Output High Voltage	2.4			V	I_{OH} = -250μA
I_{CC}	Power Supply Current		90	200	mA	
I_{LI}	Input Leakage Current			10	μA	V_{IN}=0 to V_{cc}
I_{LOH}	Tri-State Output Leakage Current in Float			10	μA	V_{OUT}=2.4 to V_{cc}
I_{LOL}	Tri-State Output Leakage Current in Float			-10	μA	V_{OUT}=0.4V
I_{LD}	Data Bus Leakage Current in Input Mode			±10	μA	$0 \leq V_{IN} \leq V_{cc}$

Capacitance

T_A = 25°C, f = 1 MHz,
unmeasured pins returned to ground

Symbol	Parameter	Max.	Unit
C_ϕ	Clock Capacitance	35	pF
C_{IN}	Input Capacitance	5	pF
C_{OUT}	Output Capacitance	10	pF

Z80A-CPU Ordering Information

C – Ceramic
P – Plastic
S – Standard 5V ±5% 0° to 70°C

T_A = 0°C to 70°C, Vcc = +5V ± 5%, Unless Otherwise Noted.

Signal	Symbol	Parameter	Min	Max	Unit	Test Condition
Φ	t_c	Clock Period	.4	[12]	μsec	
	t_w (ΦH)	Clock Pulse Width, Clock High	180	[E]	nsec	
	t_w (ΦL)	Clock Pulse Width, Clock Low	180	2000	nsec	
	$t_{r,f}$	Clock Rise and Fall Time		30	nsec	
A_{0-15}	$t_{D(AD)}$	Address Output Delay		145	nsec	C_L = 50pF
	$t_{F(AD)}$	Delay to Float		110	nsec	
	t_{acm}	Address Stable Prior to $\overline{MREQ}$ (Memory Cycle)	[1]		nsec	
	t_{aci}	Address Stable Prior to $\overline{IORQ}$, $\overline{RD}$ or $\overline{WR}$ (I/O Cycle)	[2]		nsec	
	t_{ca}	Address Stable from $\overline{RD}$, $\overline{WR}$, $\overline{IORQ}$ or $\overline{MREQ}$	[3]		nsec	
	t_{caf}	Address Stable From $\overline{RD}$ or $\overline{WR}$ During Float	[4]		nsec	
D_{0-7}	$t_{D(D)}$	Data Output Delay		230	nsec	C_L = 50pF
	$t_{F(D)}$	Delay to Float During Write Cycle		90	nsec	
	$t_{S\Phi(D)}$	Data Setup Time to Rising Edge of Clock During M1 Cycle	50		nsec	
	$t_{S\overline{\Phi}(D)}$	Data Setup Time to Falling Edge of Clock During M2 to M5	60		nsec	
	t_{dcm}	Data Stable Prior to $\overline{WR}$ (Memory Cycle)	[5]		nsec	
	t_{dci}	Data Stable Prior to $\overline{WR}$ (I/O Cycle)	[6]		nsec	
	t_{cdf}	Data Stable From $\overline{WR}$	[7]			
	t_H	Any Hold Time for Setup Time	0		nsec	
$\overline{MREQ}$	$t_{DL\overline{\Phi}(MR)}$	$\overline{MREQ}$ Delay From Falling Edge of Clock, $\overline{MREQ}$ Low		100	nsec	C_L = 50pF
	$t_{DH\Phi(MR)}$	$\overline{MREQ}$ Delay From Rising Edge of Clock, $\overline{MREQ}$ High		100	nsec	
	$t_{DH\overline{\Phi}(MR)}$	$\overline{MREQ}$ Delay From Falling Edge of Clock, $\overline{MREQ}$ High		100	nsec	
	t_w ($\overline{MRL}$)	Pulse Width, $\overline{MREQ}$ Low	[8]		nsec	
	t_w ($\overline{MRH}$)	Pulse Width, $\overline{MREQ}$ High	[9]		nsec	
$\overline{IORQ}$	$t_{DL\Phi(IR)}$	$\overline{IORQ}$ Delay From Rising Edge of Clock, $\overline{IORQ}$ Low		90	nsec	C_L = 50pF
	$t_{DL\overline{\Phi}(IR)}$	$\overline{IORQ}$ Delay From Falling Edge of Clock, $\overline{IORQ}$ Low		110	nsec	
	$t_{DH\Phi(IR)}$	$\overline{IORQ}$ Delay From Rising Edge of Clock, $\overline{IORQ}$ High		100	nsec	
	$t_{DH\overline{\Phi}(IR)}$	$\overline{IORQ}$ Delay From Falling Edge of Clock, $\overline{IORQ}$ High		110	nsec	
$\overline{RD}$	$t_{DL\Phi(RD)}$	$\overline{RD}$ Delay From Rising Edge of Clock, $\overline{RD}$ Low		100	nsec	C_L = 50pF
	$t_{DL\overline{\Phi}(RD)}$	$\overline{RD}$ Delay From Falling Edge of Clock, $\overline{RD}$ Low		130	nsec	
	$t_{DH\Phi(RD)}$	$\overline{RD}$ Delay From Rising Edge of Clock, $\overline{RD}$ High		100	nsec	
	$t_{DH\overline{\Phi}(RD)}$	$\overline{RD}$ Delay From Falling Edge of Clock, $\overline{RD}$ High		110	nsec	
$\overline{WR}$	$t_{DL\Phi(WR)}$	$\overline{WR}$ Delay From Rising Edge of Clock, $\overline{WR}$ Low		80	nsec	C_L = 50pF
	$t_{DL\overline{\Phi}(WR)}$	$\overline{WR}$ Delay From Falling Edge of Clock, $\overline{WR}$ Low		90	nsec	
	$t_{DH\overline{\Phi}(WR)}$	$\overline{WR}$ Delay From Falling Edge of Clock, $\overline{WR}$ High		100	nsec	
	t_w ($\overline{WRL}$)	Pulse Width, $\overline{WR}$ Low	[10]		nsec	
$\overline{M1}$	$t_{DL(M1)}$	$\overline{M1}$ Delay From Rising Edge of Clock, $\overline{M1}$ Low		130	nsec	C_L = 50pF
	$t_{DH(M1)}$	$\overline{M1}$ Delay From Rising Edge of Clock, $\overline{M1}$ High		130	nsec	
$\overline{RFSH}$	$t_{DL(RF)}$	$\overline{RFSH}$ Delay From Rising Edge of Clock, $\overline{RFSH}$ Low		180	nsec	C_L = 50pF
	$t_{DH(RF)}$	$\overline{RFSH}$ Delay From Rising Edge of Clock, $\overline{RFSH}$ High		150	nsec	
$\overline{WAIT}$	$t_{S(WT)}$	$\overline{WAIT}$ Setup Time to Falling Edge of Clock	70		nsec	
$\overline{HALT}$	$t_{D(HT)}$	$\overline{HALT}$ Delay Time From Falling Edge of Clock		300	nsec	C_L = 50pF
$\overline{INT}$	$t_{S(IT)}$	$\overline{INT}$ Setup Time to Rising Edge of Clock	80		nsec	
$\overline{NMI}$	t_w ($\overline{NML}$)	Pulse Width, $\overline{NMI}$ Low	80		nsec	
$\overline{BUSRQ}$	$t_{S(BQ)}$	$\overline{BUSRQ}$ Setup Time to Rising Edge of Clock	80		nsec	
$\overline{BUSAK}$	$t_{DL(BA)}$	$\overline{BUSAK}$ Delay From Rising Edge of Clock, $\overline{BUSAK}$ Low		120	nsec	C_L = 50pF
	$t_{DH(BA)}$	$\overline{BUSAK}$ Delay From Falling Edge of Clock, $\overline{BUSAK}$ High		110	nsec	
$\overline{RESET}$	$t_{S(RS)}$	$\overline{RESET}$ Setup Time to Rising Edge of Clock	90		nsec	
	$t_{F(C)}$	Delay to Float ($\overline{MREQ}$, $\overline{IORQ}$, $\overline{RD}$ and $\overline{WR}$)		100	nsec	
	t_{mr}	$\overline{M1}$ Stable Prior to $\overline{IORQ}$ (Interrupt Ack.)	[11]		nsec	

[12] $t_c = t_{w(\Phi H)} + t_{w(\Phi L)} + t_r + t_f$

[1] $t_{acm} = t_{w(\Phi H)} + t_f - 75$

[2] $t_{aci} = t_c - 80$

[3] $t_{ca} = t_{w(\Phi L)} + t_r - 40$

[4] $t_{caf} = t_{w(\Phi L)} + t_r - 60$

[5] $t_{dcm} = t_c - 210$

[6] $t_{dci} = t_{w(\Phi L)} + t_r - 210$

[7] $t_{cdf} = t_{w(\Phi L)} + t_r - 80$

[8] $t_{w(MRL)} = t_c - 40$

[9] $t_{w(MRH)} = t_{w(\Phi H)} + t_f - 30$

[10] $t_{w(\overline{WRL})} = t_c - 40$

[11] $t_{mr} = 2t_c + t_{w(\Phi H)} + t_f - 80$

NOTES:

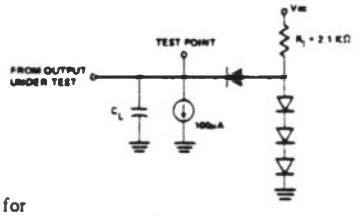
A. Data should be enabled onto the CPU data bus when $\overline{RD}$ is active. During interrupt acknowledge data should be enabled when $\overline{M1}$ and $\overline{IORQ}$ are both active.

B. All control signals are internally synchronized, so they may be totally asynchronous with respect to the clock.

C. The $\overline{RESET}$ signal must be active for a minimum of 3 clock cycles.

D. Output Delay vs. Loaded Capacitance
TA = 70°C Vcc = +5V ±5%
Add 10nsec delay for each 50pf increase in load up to a maximum of 200pf for the data bus & 100pf for address & control lines

E. Although static by design, testing guarantees $t_{w(\Phi H)}$ of 200 μsec maximum

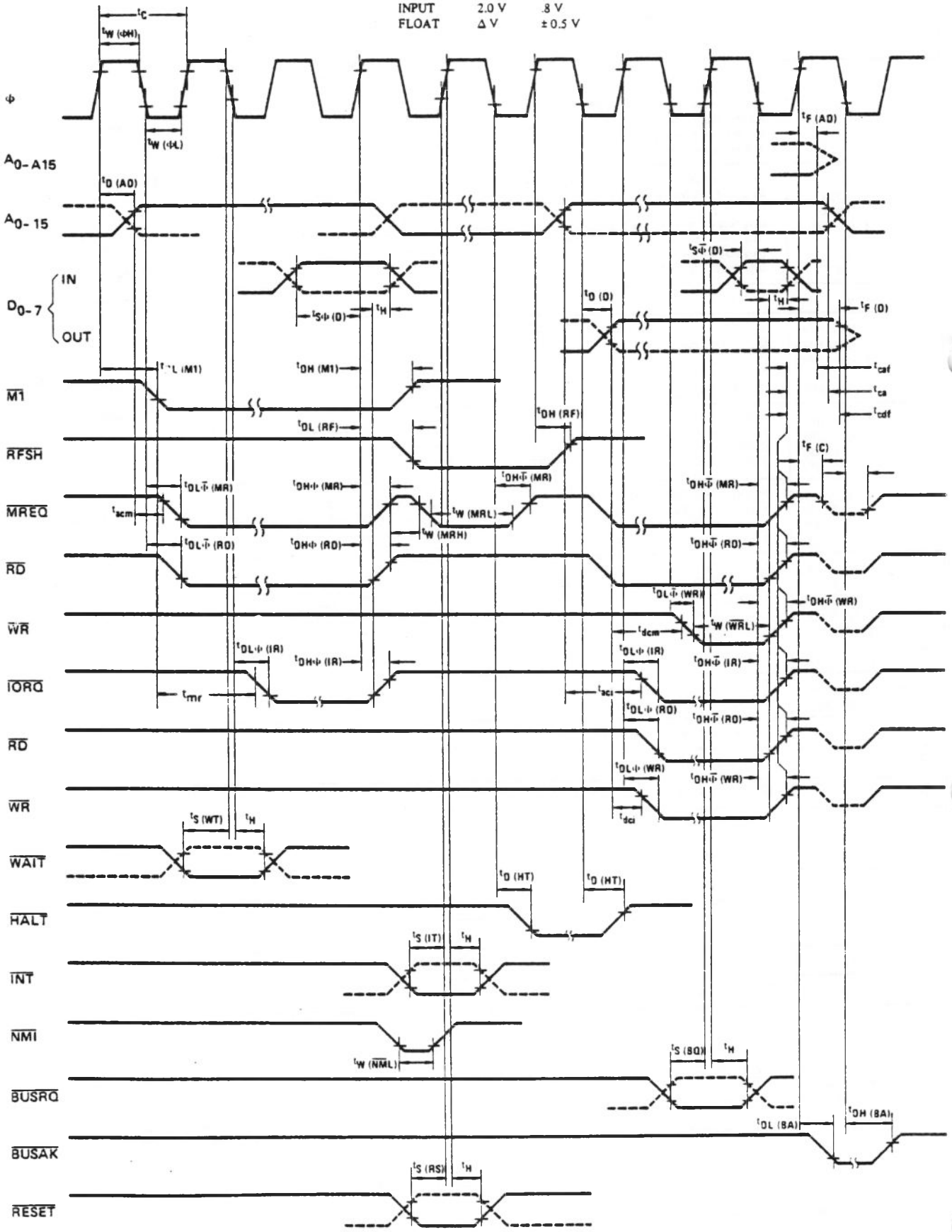


Load circuit for Output

A.C. Timing Diagram

Timing measurements are made at the following voltages, unless otherwise specified:

	"1"	"0"
CLOCK	V_{cc} -.6V	.45V
OUTPUT	2.0 V	.8 V
INPUT	2.0 V	.8 V
FLOAT	Δ V	± 0.5 V



T_A = 0°C to 70°C, Vcc = +5V ± 5%, Unless Otherwise Noted.

Signal	Symbol	Parameter	Min	Max	Unit	Test Condition
Φ	t_c	Clock Period	.25	[12]	μsec	
	$t_{w(\Phi H)}$	Clock Pulse Width, Clock High	110	[E]	nsec	
	$t_{w(\Phi L)}$	Clock Pulse Width, Clock Low	110	2000	nsec	
	t_r, t_f	Clock Rise and Fall Time		30	nsec	
A_{0-15}	$t_{D(AD)}$	Address Output Delay		110	nsec	C_L = 50pF
	$t_{F(AD)}$	Delay to Float		90	nsec	
	t_{acm}	Address Stable Prior to MREQ (Memory Cycle)	[1]		nsec	
	t_{aci}	Address Stable Prior to IORQ, RD or WR (I/O Cycle)	[2]		nsec	
	t_{ca}	Address Stable from RD, WR, IORQ or MREQ	[3]		nsec	
	t_{caf}	Address Stable From RD or WR During Float	[4]		nsec	
D_{0-7}	$t_{D(D)}$	Data Output Delay		150	nsec	C_L = 50pF
	$t_{F(D)}$	Delay to Float During Write Cycle		90	nsec	
	$t_{S\Phi(D)}$	Data Setup Time to Rising Edge of Clock During M1 Cycle	35		nsec	
	$t_{S\bar{\Phi}(D)}$	Data Setup Time to Falling Edge of Clock During M2 to M5	50		nsec	
	t_{dcm}	Data Stable Prior to WR (Memory Cycle)	[5]		nsec	
	t_{dci}	Data Stable Prior to WR (I/O Cycle)	[6]		nsec	
	t_{cdf}	Data Stable From WR	[7]			
	t_H	Any Hold Time for Setup Time		0	nsec	
MREQ	$t_{DL\bar{\Phi}(MR)}$	MREQ Delay From Falling Edge of Clock, MREQ Low		85	nsec	C_L = 50pF
	$t_{DH\Phi(MR)}$	MREQ Delay From Rising Edge of Clock, MREQ High		85	nsec	
	$t_{DH\bar{\Phi}(MR)}$	MREQ Delay From Falling Edge of Clock, MREQ High		85	nsec	
	$t_{w(\overline{MRL})}$	Pulse Width, MREQ Low	[8]		nsec	
	$t_{w(\overline{MRH})}$	Pulse Width, MREQ High	[9]		nsec	
IORQ	$t_{DL\Phi(IR)}$	IORQ Delay From Rising Edge of Clock, IORQ Low		75	nsec	C_L = 50pF
	$t_{DL\bar{\Phi}(IR)}$	IORQ Delay From Falling Edge of Clock, IORQ Low		85	nsec	
	$t_{DH\Phi(IR)}$	IORQ Delay From Rising Edge of Clock, IORQ High		85	nsec	
	$t_{DH\bar{\Phi}(IR)}$	IORQ Delay From Falling Edge of Clock, IORQ High		85	nsec	
RD	$t_{DL\Phi(RD)}$	RD Delay From Rising Edge of Clock, RD Low		85	nsec	C_L = 50pF
	$t_{DL\bar{\Phi}(RD)}$	RD Delay From Falling Edge of Clock, RD Low		95	nsec	
	$t_{DH\Phi(RD)}$	RD Delay From Rising Edge of Clock, RD High		85	nsec	
	$t_{DH\bar{\Phi}(RD)}$	RD Delay From Falling Edge of Clock, RD High		85	nsec	
WR	$t_{DL\Phi(WR)}$	WR Delay From Rising Edge of Clock, WR Low		65	nsec	C_L = 50pF
	$t_{DL\bar{\Phi}(WR)}$	WR Delay From Falling Edge of Clock, WR Low		80	nsec	
	$t_{DH\bar{\Phi}(WR)}$	WR Delay From Falling Edge of Clock, WR High		80	nsec	
	$t_{w(\overline{WRL})}$	Pulse Width, WR Low	[10]		nsec	
M1	$t_{DL(M1)}$	M1 Delay From Rising Edge of Clock, M1 Low		100	nsec	C_L = 50pF
	$t_{DH(M1)}$	M1 Delay From Rising Edge of Clock, M1 High		100	nsec	
RFSH	$t_{DL(RF)}$	RFSH Delay From Rising Edge of Clock, RFSH Low		130	nsec	C_L = 50pF
	$t_{DH(RF)}$	RFSH Delay From Rising Edge of Clock, RFSH High		120	nsec	
WAIT	$t_{s(WT)}$	WAIT Setup Time to Falling Edge of Clock	70		nsec	
HALT	$t_{D(HT)}$	HALT Delay Time From Falling Edge of Clock		300	nsec	C_L = 50pF
INT	$t_{s(IT)}$	INT Setup Time to Rising Edge of Clock	80		nsec	
NMI	$t_{w(NML)}$	Pulse Width, NMI Low	80		nsec	
BUSRQ	$t_{s(BQ)}$	BUSRQ Setup Time to Rising Edge of Clock	50		nsec	
BUSAK	$t_{DL(BA)}$	BUSAK Delay From Rising Edge of Clock, BUSAK Low		100	nsec	C_L = 50pF
	$t_{DH(BA)}$	BUSAK Delay From Falling Edge of Clock, BUSAK High		100	nsec	
RESET	$t_{s(RS)}$	RESET Setup Time to Rising Edge of Clock	60		nsec	
	$t_{F(C)}$	Delay to Float (MREQ, IORQ, RD and WR)		80	nsec	
	t_{mr}	M1 Stable Prior to IORQ (Interrupt Ack.)	[11]		nsec	

[12] $t_c = t_{w(\Phi H)} + t_{w(\Phi L)} + t_r + t_f$

[1] $t_{acm} = t_{w(\Phi H)} + t_f - 65$

[2] $t_{aci} = t_c - 70$

[3] $t_{ca} = t_{w(\Phi L)} + t_r - 50$

[4] $t_{caf} = t_{w(\Phi L)} + t_r - 45$

[5] $t_{dcm} = t_c - 170$

[6] $t_{dci} = t_{w(\Phi L)} + t_r - 170$

[7] $t_{cdf} = t_{w(\Phi L)} + t_r - 70$

[8] $t_{w(\overline{MRL})} = t_c - 30$

[9] $t_{w(\overline{MRH})} = t_{w(\Phi H)} + t_f - 20$

[10] $t_{w(\overline{WRL})} = t_c - 30$

[11] $t_{mr} = 2t_c + t_{w(\Phi H)} + t_f - 65$

NOTES:

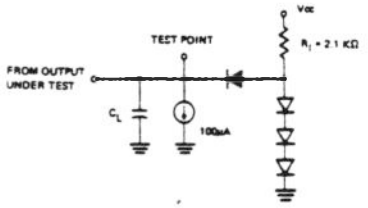
A. Data should be enabled onto the CPU data bus when RD is active. During interrupt acknowledge data should be enabled when M1 and IORQ are both active.

B. All control signals are internally synchronized, so they may be totally asynchronous with respect to the clock.

C. The RESET signal must be active for a minimum of 3 clock cycles.

D. Output Delay vs. Loaded Capacitance
TA = 70°C Vcc = +5V ±5%
Add 10nsec delay for each 50pf increase in load up to maximum of 200pf for data bus and 100pf for address & control lines.

E. Although static by design, testing guarantees $t_{w(\Phi H)}$ of 200 μsec maximum



Load circuit for Output

ADC HL, ss	Add with Carry Reg. pair ss to HL
ADC A, s	Add with carry operand s to Acc.
ADD A, n	Add value n to Acc.
ADD A, r	Add Reg. r to Acc.
ADD A, (HL)	Add location (HL) to Acc.
ADD A, (IX+d)	Add location (IX+d) to Acc.
ADD A, (IY+d)	Add location (IY+d) to Acc.
ADD HL, ss	Add Reg. pair ss to HL
ADD IX, pp	Add Reg. pair pp to IX
ADD IY, rr	Add Reg. pair rr to IY
AND s	Logical 'AND' of operand s and Acc.
BIT b, (HL)	Test BIT b of location (HL)
BIT b, (IX+d)	Test BIT b of location (IX+d)
BIT b, (IY+d)	Test BIT b of location (IY+d)
BIT b, r	Test BIT b of Reg. r
CALL cc, nn	Call subroutine at location nn if condition cc if true
CALL nn	Unconditional call subroutine at location nn
CCF	Complement carry flag
CP s	Compare operand s with Acc.
CPD	Compare location (HL) and Acc. decrement HL and BC
CPDR	Compare location (HL) and Acc. decrement HL and BC, repeat until BC=0
CPI	Compare location (HL) and Acc. increment HL and decrement BC
CPIR	Compare location (HL) and Acc. increment HL, decrement BC repeat until BC=0
CPL	Complement Acc. (1's comp)
DAA	Decimal adjust Acc.
DEC m	Decrement operand m
DEC IX	Decrement IX
DEC IY	Decrement IY
DEC ss	Decrement Reg. pair ss
DI	Disable interrupts
DJNZ e	Decrement B and Jump relative if B≠0
EI	Enable interrupts
EX (SP), HL	Exchange the location (SP) and HL
EX (SP), IX	Exchange the location (SP) and IX
EX (SP), IY	Exchange the location (SP) and IY
EX AF, AF'	Exchange the contents of AF and AF'
EX DE, HL	Exchange the contents of DE and HL
EXX	Exchange the contents of BC, DE, HL with contents of BC', DE', HL' respectively
HALT	HALT (wait for interrupt or reset)
IM 0	Set interrupt mode 0
IM 1	Set interrupt mode 1
IM 2	Set interrupt mode 2
IN A, (n)	Load the Acc. with input from device n
IN r, (C)	Load the Reg. r with input from device (C)
INC (HL)	Increment location (HL)
INC IX	Increment IX
INC (IX+d)	Increment location (IX+d)
INC IY	Increment IY
INC (IY+d)	Increment location (IY+d)
INC r	Increment Reg. r
INC ss	Increment Reg. pair ss
IND	Load location (HL) with input from port (C), decrement HL and B
INDR	Load location (HL) with input from port (C), decrement HL and decrement B, repeat until B=0
INI	Load location (HL) with input from port (C); and increment HL and decrement B

INIR	Load location (HL) with input from port (C), increment HL and decrement B, repeat until B=0
JP (HL)	Unconditional Jump to (HL)
JP (IX)	Unconditional Jump to (IX)
JP (IY)	Unconditonal Jump to (IY)
JP cc, nn	Jump to location nn if condition cc is true
JP nn	Unconditional jump to location nn
JP C, e	Jump relative to PC+e if carry=1
JR e	Unconditional Jump relative to PC+e
JP NC, e	Jump relative to PC+e if carry=0
JR NZ, e	Jump relative to PC+e if non zero (Z=0)
JR Z, e	Jump relative to PC+e if zero (Z=1)
LD A, (BC)	Load Acc. with location (BC)
LD A, (DE)	Load Acc. with location (DE)
LD A, I	Load Acc. with I
LD A, (nn)	Load Acc. with location nn
LD A, R	Load Acc. with Reg. R
LD (BC), A	Load location (BC) with Acc.
LD (DE), A	Load location (DE) with Acc.
LD (HL), n	Load location (HL) with value n
LD dd, nn	Load Reg. pair dd with value nn
LD HL, (nn)	Load HL with location (nn)
LD (HL), r	Load location (HL) with Reg. r
LD I, A	Load I with Acc.
LF IX, nn	Load IX with value nn
LD IX, (nn)	Load IX with location (nn)
LD (IX+d), n	Load location (IX+d) with value n
LD (IX+d), r	Load location (IX+d) with Reg. r
LD IY, nn	Load IY with value nn
LD IY, (nn)	Load IY with location (nn)
LD (IY+d), n	Load location (IY+d) with value n
LD (IY+d), r	Load location (IY+d) with Reg. r
LD (nn), A	Load location (nn) with Acc.
LD (nn), dd	Load location (nn) with Reg.
LD (nn), HL	Load location (nn) with HL
LD (nn), IX	Load location (nn) with IX
LD (nn), IY	Load location (nn) with IY
LD R, A	Load R with Acc.
LD r, (HL)	Load Reg. r with location (Hl
LD r, (IX+d)	Load Reg. r with location (IX
LD r, (IY+d)	Load Reg. r with location (IY
LD r, n	Load Reg. r with value n
LD r, r'	Load Reg. r with Reg. r'
LD SP, HL	Load SP with HL
LD SP, IX	Load SP with IX
LD SP, IY	Load SP with IY
LDD	Load location (DE) with loca (HL), decrement DE, HL and
LDDR	Load location (DE) with loca (HL), decrement DE, HL and repeat until BC=0
LDI	Load location (DE) with loca (HL), increment DE, HL, decrement BC
LDIR	Load location (DE) with loca (HL), increment DE, HL, decrement BC and repeat unt BC=0
NEG	Negate Acc. (2's complement
NOP	No operation
OR s	Logical 'OR' or operand s and
OTDR	Load output port (C) with lo (HL) decrement HL and B, re until B=0
OTIR	Load output port (C) with loc (HL), increment HL, decreme repeat until B=0
OUT (C), r	Load output port (C) with Re
OUT (n), A	Load output port (n) with Acc
OUTD	Load output port (C) with loc (HL), decrement HL and B
OUTI	Load output port (C) with loc (HL), increment HL and decre B

POP IX	Load IX with top of stack
POP IY	Load IY with top of stack
POP qq	Load Reg. pair qq with top of stack
PUSH IX	Load IX onto stack
PUSH IY	Load IY onto stack
PUSH qq	Load Reg. pair qq onto stack
RES b, m	Reset Bit b of operand m
RET	Return from subroutine
RET cc	Return from subroutine if condition cc is true
RETI	Return from interrupt
RETN	Return from non maskable interrupt
RL m	Rotate left through carry operand m
RLA	Rotate left Acc. through carry
RLC (HL)	Rotate location (HL) left circular
RLC (IX+d)	Rotate location (IX+d) left circular
RLC (IY+d)	Rotate location (IY+d) left circular
RLC r	Rotate Reg. r left circular
RLCA	Rotate left circular Acc.
RLD	Rotate digit left and right between Acc. and location (HL)
RR m	Rotate right through carry operand m
RRA	Rotate right Acc. through carry
RRC m	Rotate operand m right circular
RRCA	Rotate right circular Acc.
RRD	Rotate digit right and left between Acc. and location (HL)
RST p	Restart to location p
SBC A, s	Subtract operand s from Acc. with carry
SBC HL, ss	Subtract Reg. pair ss from HL with carry
SCF	Set carry flag (C=1)
SET b, (HL)	Set Bit b of location (HL)
SET b, (IX+d)	Set Bit b of location (IX+d)
SET b, (IY+d)	Set Bit b of location (IY+d)
SET b, r	Set Bit b of Reg. r
SLA m	Shift operand m left arithmetic
SRA m	Shift operand m right arithmetic
SRL m	Shift operand m right logical
SUB s	Subtract operand s from Acc.
XOR s	Exclusive 'OR' operand s and Acc.

Mnemonic Comparison between System Z80 and System 8080A

Opcode	8080A		Z80		Opcode	8080A		Z80	
00	NOP		NOP		30	----		JR	NC,disp
01	LXI	B,dddd	LD	BC,dddd	31	LXI	SP,dddd	LD	SP,dddd
02	STAX	B	LD	(BC),A	32	STA	adr	LD	(adr),A
03	INX	B	INC	BC	33	INX	SP	INC	SP
04	INR	B	INC	B	34	INR	M	INC	(HL)
05	DCR	B	DEC	B	35	DCR	M	DEC	(HL)
06	MVI	B,dd	LD	B,dd	36	MVI	M,dd	LD	(HL),dd
07	RLC		RLCA		37	STC		SCF	
08			EX	AF,AF'	38	----		JR	C,disp
09	DAD	B	ADD	HL,BC	39	DAD	SP	ADD	HL,SP
0A	LDAX	B	LD	A,(BC)	3A	LDA	adr	LD	A,(adr)
0B	DCX	B	DEC	BC	3B	DCX	SP	DEC	SP
0C	INR	C	INC	C	3C	INR	A	INC	A
0D	DCR	C	DEC	C	3D	DCR	A	DEC	A
0E	MVI	C,dd	LD	C,dd	3E	MVI	A,dd	LD	A,dd
0F	RRC		RRCA		3F	CMC		CCF	
10	----		DJNZ	disp	40	MOV	B,B	LD	B,B
11	LXI	D,dddd	LD	DE,dddd	41	MOV	B,C	LD	B,C
12	STAX	D	LD	(DE),A	42	MOV	B,D	LD	B,D
13	INX	D	INC	DE	43	MOV	B,E	LD	B,E
14	INR	D	INC	D	44	MOV	B,H	LD	B,H
15	DCR	D	DEC	D	45	MOV	B,L	LD	B,L
16	MVI	D,dd	LD	D,dd	46	MOV	B,M	LD	B,(HL)
17	RAL		RLA		47	MOV	B,A	LD	B,A
18	----		JR	disp	48	MOV	C,B	LD	C,B
19	DAD	D	ADD	HL,DE	49	MOV	C,C	LD	C,C
1A	LDAX	D	LD	A,(DE)	4A	MOV	C,D	LD	C,D
1B	DCX	D	DEC	DE	4B	MOV	C,E	LD	C,E
1C	INR	E	INC	E	4C	MOV	C,H	LD	C,H
1D	DCR	E	DEC	E	4D	MOV	C,L	LD	C,L
1E	MVI	E,dd	LD	E,dd	4E	MOV	C,M	LD	C,(HL)
1F	RAR		RRA		4F	MOV	C,A	LD	C,A
20	----		JR	NZ,disp	50	MOV	D,B	LD	D,B
21	LXI	H,dddd	LD	HL,dddd	51	MOV	D,C	LD	D,C
22	SHLD	adr	LD	(adr),HL	52	MOV	D,D	LD	D,D
23	INX	H	INC	HL	53	MOV	D,E	LD	D,E
24	INR	H	INC	H	54	MOV	D,H	LD	D,H
25	DCR	H	DEC	H	55	MOV	D,L	LD	D,L
26	MVI	H,dd	LD	H,dd	56	MOV	D,M	LD	D,(HL)
27	DAA		DAA		57	MOV	D,A	LD	D,A
28	----		JR	Z,disp	58	MOV	E,B	LD	E,B
29	DAD	H	ADD	HL,HL	59	MOV	E,C	LD	E,C
2A	LHLD	adr	LD	HL,(adr)	5A	MOV	E,D	LD	E,D
2B	DCX	H	DEC	HL	5B	MOV	E,E	LD	E,E
2C	INR	L	INC	L	5C	MOV	E,H	LD	E,H
2D	DCR	L	DEC	L	5D	MOV	E,L	LD	E,L
2E	MVI	L,dd	LD	L,dd	5E	MOV	E,M	LD	E,(HL)
2F	CMA		CPL		5F	MOV	E,A	LD	E,A

Opcode	8080A		Z80		Opcode	8080A		Z80	
60	MOV	H,B	LD	H,B	90	SUB	B	SUB	B
61	MOV	H,C	LD	H,C	91	SUB	C	SUB	C
62	MOV	H,D	LD	H,D	92	SUB	D	SUB	D
63	MOV	H,E	LD	H,E	93	SUB	E	SUB	E
64	MOV	H,H	LD	H,H	94	SUB	H	SUB	H
65	MOV	H,L	LD	H,L	95	SUB	L	SUB	L
66	MOV	H,M	LD	H,(HL)	96	SUB	M	SUB	(HL)
67	MOV	H,A	LD	H,A	97	SUB	A	SUB	A
68	MOV	L,B	LD	L,B	98	SBB	B	SBC	A,B
69	MOV	L,C	LD	L,C	99	SBB	C	SBC	A,C
6A	MOV	L,D	LD	L,D	9A	SBB	D	SBC	A,D
6B	MOV	L,E	LD	L,E	9B	SBB	E	SBC	A,E
6C	MOV	L,H	LD	L,H	9C	SBB	H	SBC	A,H
6D	MOV	L,L	LD	L,L	9D	SBB	L	SBC	A,L
6E	MOV	L,M	LD	L,(HL)	9E	SBB	M	SBC	A,(HL)
6F	MOV	L,A	LD	L,A	9F	SBB	A	SBC	A,A
70	MOV	M,B	LD	(HL),B	A0	ANA	B	AND	B
71	MOV	M,C	LD	(HL),C	A1	ANA	C	AND	C
72	MOV	M,D	LD	(HL),D	A2	ANA	D	AND	D
73	MOV	M,E	LD	(HL),E	A3	ANA	E	AND	E
74	MOV	M,H	LD	(HL),H	A4	ANA	H	AND	H
75	MOV	M,L	LD	(HL),L	A5	ANA	L	AND	L
76	HLT		HALT		A6	ANA	M	AND	(HL)
77	MOV	M,A	LD	(HL),A	A7	ANA	A	AND	A
78	MOV	A,B	LD	A,B	A8	XRA	B	XOR	B
79	MOV	A,C	LD	A,C	A9	XRA	C	XOR	C
7A	MOV	A,D	LD	A,D	AA	XRA	D	XOR	D
7B	MOV	A,E	LD	A,E	AB	XRA	E	XOR	E
7C	MOV	A,H	LD	A,H	AC	XRA	H	XOR	H
7D	MOV	A,L	LD	A,L	AD	XRA	L	XOR	L
7E	MOV	A,M	LD	A,(HL)	AE	XRA	M	XOR	(HL)
7F	MOV	A,A	LD	A,A	AF	XRA	A	XOR	A
80	ADD	B	ADD	A,B	B0	ORA	B	OR	B
81	ADD	C	ADD	A,C	B1	ORA	C	OR	C
82	ADD	D	ADD	A,D	B2	ORA	D	OR	D
83	ADD	E	ADD	A,E	B3	ORA	E	OR	E
84	ADD	H	ADD	A,H	B4	ORA	H	OR	H
85	ADD	L	ADD	A,L	B5	ORA	L	OR	L
86	ADD	M	ADD	A,(HL)	B6	ORA	M	OR	(HL)
87	ADD	A	ADD	A,A	B7	ORA	A	OR	A
88	ADC	B	ADC	A,B	B8	CMP	B	CP	B
89	ADC	C	ADC	A,C	B9	CMP	C	CP	C
8A	ADC	D	ADC	A,D	BA	CMP	D	CP	D
8B	ADC	E	ADC	A,E	BB	CMP	E	CP	E
8C	ADC	H	ADC	A,H	BC	CMP	H	CP	H
8D	ADC	L	ADC	A,L	BD	CMP	L	CP	L
8E	ADC	M	ADC	A,(HL)	BE	CMP	M	CP	(HL)
8F	ADC	A	ADC	A,A	BF	CMP	A	CP	A

Opcode	8080A		Z80	
C0	RNZ		RET	NZ
C1	POP	B	POP	BC
C2	JNZ	adr	JP	NZ,adr
C3	JMP	adr	JP	adr
C4	CNZ	adr	CALL	NZ,adr
C5	PUSH	B	PUSH	BC
C6	ADI	dd	ADD	A,dd
C7	RST	0	RST	0
C8	RZ		RET	Z
C9	RET		RET	
CA	JZ	adr	JP	Z,adr
CB	----		see below	
CC	CZ	adr	CALL	Z,adr
CD	CALL	adr	CALL	adr
CE	ACI	dd	ADC	A,dd
CF	RST	1	RST	8
D0	RNC		RET	NC
D1	POP	D	POP	DE
D2	JNC	adr	JP	NC,adr
D3	OUT	port	OUT	port,A
D4	CNC	adr	CALL	NC,adr
D5	PUSH	D	PUSH	DE
D6	SUI	dd	SUB	dd
D7	RST	2	RST	10H
D8	RC		RET	C
D9	----		EXX	
DA	JC	adr	JP	C,adr
DB	IN	port	IN	A,port
DC	CC	adr	CALL	C,adr
DD	----		see below	
DE	SBI	dd	SBC	A,dd
DF	RST	3	RST	18H
E0	RPO		RET	PO
E1	POP	H	POP	HL
E2	JPO	adr	JP	PO,adr
E3	XTHL		EX	(SP),HL
E4	CPO	adr	CALL	PO,adr
E5	PUSH	H	PUSH	HL
E6	ANI	dd	AND	dd
E7	RST	4	RST	20H
E8	RPE		RET	PE
E9	PCHL		JP	(HL)
EA	JPE	adr	JP	PE,adr
EB	XCHG		EX	DE,HL
EC	CPE	adr	CALL	PE,adr
ED	----		see below	
EE	XRI	dd	XOR	dd
EF	RST	5	RST	28H
F0	RP		RET	P
F1	POP	PSW	POP	AF
F2	JP	adr	JP	P,adr
F3	DI		DI	
F4	CP	adr	CALL	P,adr
F5	PUSH	PSW	PUSH	AF
F6	ORI	dd	OR	dd
F7	RST	6	RST	30H
F8	RM		RET	M
F9	SPHL		LD	SP,HL
FA	JM	adr	JP	M,adr
FB	EI		EI	
FC	CM	adr	CALL	M,adr
FD	----		see below	
FE	CPI	dd	CP	dd
FF	RST	7	RST	38H

The following instructions are unique to Z80 :-

Opcode

CB00	RLC	B
CB01	RLC	C
CB02	RLC	D
CB03	RLC	E
CB04	RLC	H
CB05	RLC	L
CB06	RLC	(HL)
CB07	RLC	A
CB08	RRC	B
CB09	RRC	C
CB0A	RRC	D
CB0B	RRC	E
CB0C	RRC	H
CB0D	RRC	L
CB0E	RRC	(HL)
CB0F	RRC	A
CB10	RL	B
CB11	RL	C
CB12	RL	D
CB13	RL	E
CB14	RL	H
CB15	RL	L
CB16	RL	(HL)
CB17	RL	A
CB18	RR	B
CB19	RR	C
CB1A	RR	D
CB1B	RR	E
CB1C	RR	H
CB1D	RR	L
CB1E	RR	(HL)
CB1F	RR	A
CB20	SLA	B
CB21	SLA	C
CB22	SLA	D
CB23	SLA	E
CB24	SLA	H
CB25	SLA	L
CB26	SLA	(HL)
CB27	SLA	A
CB28	SRA	B
CB29	SRA	C
CB2A	SRA	D
CB2B	SRA	E
CB2C	SRA	H
CB2D	SRA	L
CB2E	SRA	(HL)
CB2F	SRA	A
CB38	SRL	B
CB39	SRL	C
CB3A	SRL	D
CB3B	SRL	E
CB3C	SRL	H
CB3D	SRL	L
CB3E	SRL	(HL)
CB3F	SRL	A
CB40	BIT	0,B
CB41	BIT	0,C
CB42	BIT	0,D
CB43	BIT	0,E
CB44	BIT	0,H
CB45	BIT	0,L
CB46	BIT	0,(HL)
CB47	BIT	0,A
CB48	BIT	1,B
CB49	BIT	1,C
CB4A	BIT	1,D
CB4B	BIT	1,E
CB4C	BIT	1,H
CB4D	BIT	1,L
CB4E	BIT	1,(HL)
CB4F	BIT	1,A
CB50	BIT	2,B
CB51	BIT	2,C
CB52	BIT	2,D
CB53	BIT	2,E
CB54	BIT	2,H
CB55	BIT	2,L
CB56	BIT	2,(HL)
CB57	BIT	2,A
CB58	BIT	3,B
CB59	BIT	3,C
CB5A	BIT	3,D
CB5B	BIT	3,E
CB5C	BIT	3,H
CB5D	BIT	3,L
CB5E	BIT	3,(HL)
CB5F	BIT	3,A

Opcode			Opcode		
CB60	BIT	4,B	CB90	RES	2,B
CB61	BIT	4,C	CB91	RES	2,C
CB62	BIT	4,D	CB92	RES	2,D
CB63	BIT	4,E	CB93	RES	2,E
CB64	BIT	4,H	CB94	RES	2,H
CB65	BIT	4,L	CB95	RES	2,L
CB66	BIT	4,(HL)	CB96	RES	2,(HL)
CB67	BIT	4,A	CB97	RES	2,A
CB68	BIT	5,B	CB98	RES	3,B
CB69	BIT	5,C	CB99	RES	3,C
CB6A	BIT	5,D	CB9A	RES	3,D
CB6B	BIT	5,E	CB9B	RES	3,E
CB6C	BIT	5,H	CB9C	RES	3,H
CB6D	BIT	5,L	CB9D	RES	3,L
CB6E	BIT	5,(HL)	CB9E	RES	3,(HL)
CB6F	BIT	5,A	CB9F	RES	3,A
CB70	BIT	6,B	CBA0	RES	4,B
CB71	BIT	6,C	CBA1	RES	4,C
CB72	BIT	6,D	CBA2	RES	4,D
CB73	BIT	6,E	CBA3	RES	4,E
CB74	BIT	6,H	CBA4	RES	4,H
CB75	BIT	6,L	CBA5	RES	4,L
CB76	BIT	6,(HL)	CBA6	RES	4,(HL)
CB77	BIT	6,A	CBA7	RES	4,A
CB78	BIT	7,B	CBA8	RES	5,B
CB79	BIT	7,C	CBA9	RES	5,C
CB7A	BIT	7,D	CBAA	RES	5,D
CB7B	BIT	7,E	CBAB	RES	5,E
CB7C	BIT	7,H	CBAC	RES	5,H
CB7D	BIT	7,L	CBAD	RES	5,L
CB7E	BIT	7,(HL)	CBAE	RES	5,(HL)
CB7F	BIT	7,A	CBAF	RES	5,A
CB80	RES	0,B	CBB0	RES	6,B
CB81	RES	0,C	CBB1	RES	6,C
CB82	RES	0,D	CBB2	RES	6,D
CB83	RES	0,E	CBB3	RES	6,E
CB84	RES	0,H	CBB4	RES	6,H
CB85	RES	0,L	CBB5	RES	6,L
CB86	RES	0,(HL)	CBB6	RES	6,(HL)
CB87	RES	0,A	CBB7	RES	6,A
CB88	RES	1,B	CBB8	RES	7,B
CB89	RES	1,C	CBB9	RES	7,C
CB8A	RES	1,D	CBBA	RES	7,D
CB8B	RES	1,E	CBBB	RES	7,E
CB8C	RES	1,H	CBBC	RES	7,H
CB8D	RES	1,L	CBBD	RES	7,L
CB8E	RES	1,(HL)	CBBE	RES	7,(HL)
CB8F	RES	1,A	CBBF	RES	7,A

Opcode		
CBC0	SET	0,B
CBC1	SET	0,C
CBC2	SET	0,D
CBC3	SET	0,E
CBC4	SET	0,H
CBC5	SET	0,L
CBC6	SET	0,(HL)
CBC7	SET	0,A
CBC8	SET	1,B
CBC9	SET	1,C
CBCA	SET	1,D
CBCB	SET	1,E
CBCC	SET	1,H
CBCD	SET	1,L
CBCE	SET	1,(HL)
CBCF	SET	1,A
CBD0	SET	2,B
CBD1	SET	2,C
CBD2	SET	2,D
CBD3	SET	2,E
CBD4	SET	2,H
CBD5	SET	2,L
CBD6	SET	2,(HL)
CBD7	SET	2,A
CBD8	SET	3,B
CBD9	SET	3,C
CBDA	SET	3,D
CBDB	SET	3,E
CBDC	SET	3,H
CBDD	SET	3,L
CBDE	SET	3,(HL)
CBDF	SET	3,A
CBE0	SET	4,B
CBE1	SET	4,C
CBE2	SET	4,D
CBE3	SET	4,E
CBE4	SET	4,H
CBE5	SET	4,L
CBE6	SET	4,(HL)
CBE7	SET	4,A
CBE8	SET	5,B
CBE9	SET	5,C
CBEA	SET	5,D
CBEB	SET	5,E
CBEC	SET	5,H
CBED	SET	5,L
CBEE	SET	5,(HL)
CBEF	SET	5,A

Opcode		
CBF0	SET	6,B
CBF1	SET	6,C
CBF2	SET	6,D
CBF3	SET	6,E
CBF4	SET	6,H
CBF5	SET	6,L
CBF6	SET	6,(HL)
CBF7	SET	6,A
CBF8	SET	7,B
CBF9	SET	7,C
CBFA	SET	7,D
CBFB	SET	7,E
CBFC	SET	7,H
CBFD	SET	7,L
CBFE	SET	7,(HL)
CBFF	SET	7,A
DD09	ADD	IX,BC
DD19	ADD	IX,DE
DD21	LD	IX,dddd
DD22	LD	(adr),IX
DD23	INC	IX
DD29	ADD	IX,IX
DD2A	LD	IX,(adr)
DD2B	DEC	IX
DD34	INC	(IX+offset)
DD35	DEC	(IX+offset)
DD36	LD	(IX+offset),dd
DD39	ADD	IX,SP
DD46	LD	B,(IX+offset)
DD4E	LD	C,(IX+offset)
DD56	LD	D,(IX+offset)
DD5E	LD	E,(IX+offset)
DD66	LD	H,(IX+offset)
DD6E	LD	L,(IX+offset)
DD70	LD	(IX+offset),B
DD71	LD	(IX+offset),C
DD72	LD	(IX+offset),D
DD73	LD	(IX+offset),E
DD74	LD	(IX+offset),H
DD75	LD	(IX+offset),L
DD77	LD	(IX+offset),A
DD7E	LD	A,(IX+offset)
DD86	ADD	A,(IX+offset)
DD8E	ADC	A,(IX+offset)

Opcode		
DD96	SUB	(IX+offset)
DD9E	SBC	A,(IX+offset)
DDA6	AND	(IX+offset)
DDAE	XOR	(IX+offset)
DDB6	OR	(IX+offset)
DDBE	CP	(IX+offset)
DDCBof06	RLC	(IX+offset)
DDCBof0E	RRC	(IX+offset)
DDCBof16	RL	(IX+offset)
DDCBof1E	RR	(IX+offset)
DDCBof26	SLA	(IX+offset)
DDCBof2E	SRA	(IX+offset)
DDCBof3E	SRL	(IX+offset)
DDCBof46	BIT	0,(IX+offset)
DDCBof4E	BIT	1,(IX+offset)
DDCBof56	BIT	2,(IX+offset)
DDCBof5E	BIT	3,(IX+offset)
DDCBof66	BIT	4,(IX+offset)
DDCBof6E	BIT	5,(IX+offset)
DDCBof76	BIT	6,(IX+offset)
DDCBof7E	BIT	7,(IX+offset)
DDCBof86	RES	0,(IX+offset)
DDCBof8E	RES	1,(IX+offset)
DDCBof96	RES	2,(IX+offset)
DDCBof9E	RES	3,(IX+offset)
DDCBofA6	RES	4,(IX+offset)
DDCBofAE	RES	5,(IX+offset)
DDCBofB6	RES	6,(IX+offset)
DDCBofBE	RES	7,(IX+offset)
DDCBofC6	SET	0,(IX+offset)
DDCBofCE	SET	1,(IX+offset)
DDCBofD6	SET	2,(IX+offset)
DDCBofDE	SET	3,(IX+offset)
DDCBofE6	SET	4,(IX+offset)
DDCBofEE	SET	5,(IX+offset)
DDCBofF6	SET	6,(IX+offset)
DDCBofFE	SET	7,(IX+offset)

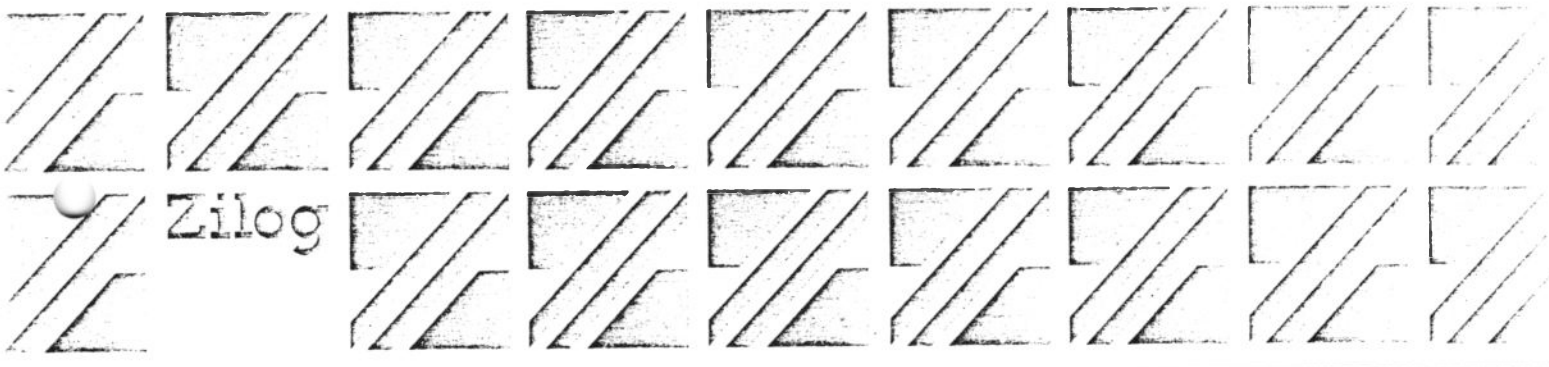
Opcode		
DDE1	POP	IX
DDE3	EX	(SP),IX
DDE5	PUSH	IX
DDE9	JP	(IX)
DDF9	LD	SP,IX
ED40	IN	B,(C)
ED41	OUT	(C),B
ED42	SBC	HL,BC
ED43	LD	(dddd),BC
ED44	NEG	
ED45	RETN	
ED46	IM	0
ED47	LD	I,A
ED48	IN	C,(C)
ED49	OUT	(C),C
ED4A	ADC	HL,BC
ED4B	LD	BC,(adr)
ED4D	RETI	
ED50	IN	D,(C)
ED51	OUT	(C),D
ED52	SBC	HL,DE
ED53	LD	(adr),DE
ED56	IM	1
ED57	LD	A,I
ED58	IN	E,(C)
ED59	OUT	(C),E
ED5A	ADC	HL,DE
ED5B	LD	DE,(adr)
ED5E	IM	2
ED60	IN	H,(C)
ED61	OUT	(C),H
ED62	SBC	HL,HL
ED67	RRD	
ED68	IN	L,(C)
ED69	OUT	(C),L
ED6A	ADC	HL,HL
ED6F	RLD	
ED72	SBC	HL,SP
ED73	LD	(adr),SP
ED78	IN	A,(C)
ED79	OUT	(C),A
ED7A	ADC	HL,SP
ED7B	LD	SP,(adr)
EDA0	LDI	
EDA1	CPI	
EDA2	INI	
EDA3	OUTI	
EDA8	LDD	
EDA9	CPD	
EDAA	IND	
EDAB	OUTD	

Opcode		
ED80	LDIR	
ED81	CPIR	
ED82	INIR	
ED83	OTIR	
ED88	LDDR	
ED89	CPDR	
ED8A	INDR	
ED8B	OTDR	
FD09	ADD	IY,BC
FD19	ADD	IY,DE
FD21	LD	IY,dddd
FD22	LD	(adr),IY
FD23	INC	IY
FD29	ADD	IY,IY
FD2A	LD	IY,(adr)
FD2B	DEC	IY
FD34	INC	(IY+offset)
FD35	DEC	(IY+offset)
FD36	LD	(IY+offset),dd
FD39	ADD	IY,SP
FD46	LD	B,(IY+offset)
FD4E	LD	C,(IY+offset)
FD56	LD	D,(IY+offset)
FD5E	LD	E,(IY+offset)
FD66	LD	H,(IY+offset)
FD6E	LD	L,(IY+offset)
FD70	LD	(IY+offset),B
FD71	LD	(IY+offset),C
FD72	LD	(IY+offset),D
FD73	LD	(IY+offset),E
FD74	LD	(IY+offset),H
FD75	LD	(IY+offset),L
FD77	LD	(IY+offset),A
FD7E	LD	A,(IY+offset)
FD86	ADD	A,(IY+offset)
FD8E	ADC	A,(IY+offset)
FD96	SUB	(IY+offset)
FD9E	SBC	A,(IY+offset)
FDA6	AND	(IY+offset)
FDAE	XOR	(IY+offset)
FDB6	OR	(IY+offset)
FDBE	CP	(IY+offset)

Opcode		
FDCB0f06	RLC	(IY+offset)
FDCB0f0E	RRC	(IY+offset)
FDCB0f16	RL	(IY+offset)
FDCB0f1E	RR	(IY+offset)
FDCB0f26	SLA	(IY+offset)
FDCB0f2E	SRA	(IY+offset)
FDCB0f3E	SRL	(IY+offset)
FDCB0f46	BIT	0,(IY+offset)
FDCB0f4E	BIT	1,(IY+offset)
FDCB0f56	BIT	2,(IY+offset)
FDCB0f5E	BIT	3,(IY+offset)
FDCB0f66	BIT	4,(IY+offset)
FDCB0f6E	BIT	5,(IY+offset)
FDCB0f76	BIT	6,(IY+offset)
FDCB0f7E	BIT	7,(IY+offset)
FDCB0f86	RES	0,(IY+offset)
FDCB0f8E	RES	1,(IY+offset)
FDCB0f96	RES	2,(IY+offset)
FDCB0f9E	RES	3,(IY+offset)
FDCB0fA6	RES	4,(IY+offset)
FDCB0fAE	RES	5,(IY+offset)
FDCB0fB6	RES	6,(IY+offset)
FDCB0fBE	RES	7,(IY+offset)
FDCB0fC6	SET	0,(IY+offset)
FDCB0fCE	SET	1,(IY+offset)
FDCB0fD6	SET	2,(IY+offset)
FDCB0fDE	SET	3,(IY+offset)
FDCB0fE6	SET	4,(IY+offset)
FDCB0fEE	SET	5,(IY+offset)
FDCB0fF6	SET	6,(IY+offset)
FDCB0fFE	SET	7,(IY+offset)
FDE1	POP	IY
FDE3	EX	(SP),IY
FDE5	PUSH	IY
FDE9	JP	(IY)
FDF9	LD	SP,IY

Zilog

Z80™-PIO
Z80A™-PIO
Technical Manual



Z80- PIO TECHNICAL MANUAL
TABLE OF CONTENTS

1.0 Introduction 1

2.0 Architecture 3

3.0 Pin Description 5

4.0 Programming the PIO 9
- 4.1 Reset 9
- 4.2 Loading the Interrupt Vector 9
- 4.3 Selecting an Operating Mode 10
- 4.4 Setting the Interrupt Control Word 11

5.0 Timing 13
- 5.1 Output Mode Timing 13
- 5.2 Input Mode Timing 13
- 5.3 Bidirectional Mode Timing 14
- 5.4 Control Mode 14

6.0 Interrupt Control 15

7.0 Applications 17
- 7.1 Interrupt Daisy Chain 17
- 7.2 I/O Device Interface 18
- 7.3 Control Interface 19

8.0 Programming Summary 21
- 8.1 Load Interrupt Vector 21
- 8.2 Set Mode 21
- 8.3 Set Interrupt Control 21

9.0 Electrical Specifications 23
- 9.1 Absolute Maximum Ratings 23
- 9.2 D.C. Characteristics 23
- 9.3 Clock Driver 23
- 9.4 A.C. Characteristics 24
- 9.5 Capacitance 24

10.0 Timing Chart 25

1.0 INTRODUCTION

The Z-80 Parallel I/O (PIO) Circuit is a programmable, two port device which provides a TTL compatible interface between peripheral devices and the Z80-CPU. The CPU can configure the Z80-PIO to interface with a wide range of peripheral devices with no other external logic required. Typical peripheral devices that are fully compatible with the Z80-PIO include most keyboards, paper tape readers and punches, printers, PROM programmers, etc. The Z80-PIO utilizes N channel silicon gate depletion load technology and is packaged in a 40 pin DIP. Major features of the Z80-PIO include:

- Two independent 8 bit bidirectional peripheral interface ports with 'handshake' data transfer control
- Interrupt driven 'handshake' for fast response
- Any one of four distinct modes of operation may be selected for a port including:

  Byte output

  Byte input

  Byte bidirectional bus (Available on Port A only)

  Bit control mode

  All with interrupt controlled handshake
- Daisy chain priority interrupt logic included to provide for automatic interrupt vectoring without external logic
- Eight outputs are capable of driving Darlington transistors
- All inputs and outputs fully TTL compatible
- Single 5 volt supply and single phase clock are required.

One of the unique freatures of the Z80-PIO that separates it from other interface controllers is that all data transfer between the peripheral device and the CPU is accomplished under total interrupt control. The interrupt logic of the PIO permits full usage of the efficient interrupt capabilities of the Z80-CPU during I/O transfers. All logic necessary to implement a fully nested interrupt structure is included in the PIO so that additional circuits are not required. Another unique feature of the PIO is that it can be programmed to interrupt the CPU on the occurrence of specified status conditions in the peripheral device. For example, the PIO can be programmed to interrupt if any specified peripheral alarm conditions should occur. This interrupt capability reduces the amount of time that the processor must spend in polling peripheral status.

2.0 PIO ARCHITECTURE

A block diagram of the Z80-PIO is shown in figure 2.0-1. The internal structure of the Z80-PIO consists of a Z80-CPU bus interface, internal control logic, Port A I/O logic, Port B I/O logic, and interrupt control logic. The CPU bus interface logic allows the PIO to interface directly to the Z80-CPU with no other external logic. However, address decoders and/or line buffers may be required for large systems. The internal control logic synchronizes the CPU data bus to the peripheral device interfaces (Port A and Port B). The two I/O ports (A and B) are virtually identical and are used to interface directly to peripheral devices.

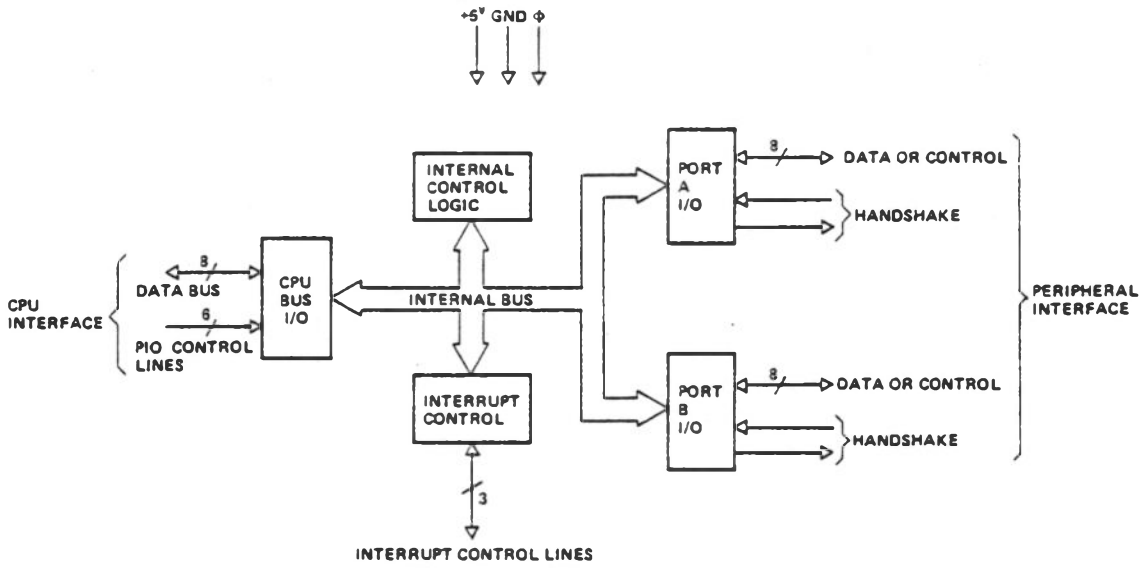


FIGURE 2.0-1
PIO BLOCK DIAGRAM

The Port I/O logic is composed of 6 registers with "handshake" control logic as shown in figure 2.0-2. The registers include: an 8 bit data input register, an 8 bit data output register, a 2 bit mode control register, an 8 bit mask register, an 8 bit input/output select register, and a 2 bit mask control register.

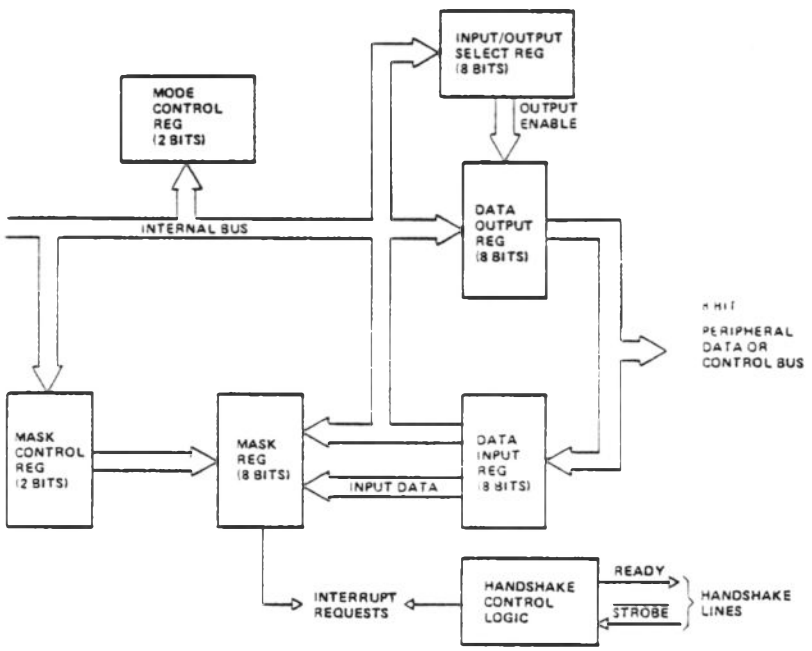


FIGURE 2.0-2
PORT I/O BLOCK DIAGRAM

The 2-bit mode control register is loaded by the CPU to select the desired operating mode (byte output, byte input, byte bidirectional bus, or bit control mode). All data transfer between the peripheral device and the CPU is achieved through the data input and data output registers. Data may be written into the output register by the CPU or read back to the CPU from the input register at any time. The handshake lines associated with each port are used to control the data transfer between the PIO and the peripheral device.

The 8-bit mask register and the 8-bit input/output select register are used only in the bit control mode. In this mode any of the 8 peripheral data or control bus pins can be programmed to be an input or an output as specified by the select register. The mask register is used in this mode in conjunction with a special interrupt feature. This feature allows an interrupt to be generated when any or all of the unmasked pins reach a specified state (either high or low). The 2-bit mask control register specifies the active state desired (high or low) and if the interrupt should be generated when *all* unmasked pins are active (AND condition) or when *any* unmasked pin is active (OR condition). This feature reduces the requirement for CPU status checking of the peripheral by allowing an interrupt to be automatically generated on specific peripheral status conditions. For example, in a system with 3 alarm conditions, an interrupt may be generated if any one occurs or if all three occur.

The interrupt control logic section handles all CPU interrupt protocol for nested priority interrupt structures. The priority of any device is determined by its physical location in a daisy chain configuration. Two lines are provided in each PIO to form this daisy chain. The device closest to the CPU has the highest priority. Within a PIO, Port A interrupts have higher priority than those of Port B. In the byte input, byte output or bidirectional modes, an interrupt can be generated whenever a new byte transfer is requested by the peripheral. In the bit control mode an interrupt can be generated when the peripheral status matches a programmed value. The PIO provides for complete control of nested interrupts. That is, lower priority devices may not interrupt higher priority devices that have not had their interrupt service routine completed by the CPU. Higher priority devices may interrupt the servicing of lower priority devices.

When an interrupt is accepted by the CPU in mode 2, the interrupting device must provide an 8-bit interrupt vector for the CPU. This vector is used to form a pointer to a location in the computer memory where the address of the interrupt service routine is located. The 8-bit vector from the interrupting device forms the least significant 8 bits of the indirect pointer while the I Register in the CPU provides the most significant 8 bits of the pointer. Each port (A and B) has an independent interrupt vector. The least significant bit of the vector is automatically set to a 0 within the PIO since the pointer must point to two adjacent memory locations for a complete 16-bit address.

The PIO decodes the RETI (Return from interrupt) instruction directly from the CPU data bus so that each PIO in the system knows at all times whether it is being serviced by the CPU interrupt service routine without any other communication with the CPU.

3.0 PIN DESCRIPTION

A diagram of the Z80-PIO pin configuration is shown in figure 3.0-1. This section describes the function of each pin.

D_7-D_0 — Z80-CPU Data Bus (bidirectional, tristate)
This bus is used to transfer all data and commands between the Z80-CPU and the Z80-PIO. D_0 is the least significant bit of the bus.

B/A Sel — Port B or A Select (input, active high)
This pin defines which port will be accessed during a data transfer between the Z80-CPU and the Z80-PIO. A low level on this pin selects Port A while a high level selects Port B. Often Address bit A_0 from the CPU will be used for this selection function.

C/D Sel — Control or Data Select (input, active high)
This pin defines the type of data transfer to be performed between the CPU and the PIO. A high level on this pin during a CPU write to the PIO causes the Z-80 data bus to be interpreted as a *command* for the port selected by the B/A Select line. A low level on this pin means that he Z-80 data bus is being used to transfer data between the CPU and the PIO. Often Address bit A_1 from the CPU will be used for this function.

$\overline{CE}$ — Chip Enable (input, active low)
A low level on this pin enables the PIO to accept command or data inputs from the CPU during a write cycle or to transmit data to the CPU during a read cycle. This signal is generally a decode of four I/O port numbers that encompass port A and B. data and control.

Φ — System Clock (input)
The Z80-PIO uses the standard Z-80 system clock to synchronize certain signals internally. This is a single phase clock.

$\overline{M1}$ — Machine Cycle One Signal from CPU (input, active low)
This signal from the CPU is used as a sync pulse to control several internal PIO operations. When $\overline{M1}$ is active and the $\overline{RD}$ signal is active, the Z80-CPU is fetching an instruction from memory. Conversely, when $\overline{M1}$ is active and $\overline{IORQ}$ is active, the CPU is acknowledging an interrupt. In addition, the $\overline{M1}$ signal has two other functions within the Z80-PIO.

1. $\overline{M1}$ synchronizes the PIO interrupt logic.
2. When $\overline{M1}$ occurs without an active $\overline{RD}$ or $\overline{IORQ}$ signal the PIO logic enters a reset state.

$\overline{IORQ}$ — Input/Output Request from Z80-CPU (input, active low)
The $\overline{IORQ}$ signal is used in conjunction with the B/A Select, C/D Select, $\overline{CE}$, and $\overline{RD}$ signals to transfer commands and data between the Z80-CPU and the Z80-PIO. When $\overline{CE}$, $\overline{RD}$ and $\overline{IORQ}$ are active, the port addressed by B/A will transfer data to the CPU (a read operation). Conversely, when $\overline{CE}$ and $\overline{IORQ}$ are active but $\overline{RD}$ is not active, then the port addressed by B/A will be written into from the CPU with either data or control information as specified by the C/D Select signal. Also, if $\overline{IORQ}$ and $\overline{M1}$ are active simultaneously, the CPU is acknowledging an interrupt and the interrupting port will automatically place its interrupt vector on the CPU data bus if it is the highest priority device requesting an interrupt.

$\overline{RD}$ — Read Cycle Status from the Z80-CPU (input, active low)
If $\overline{RD}$ is active a MEMORY READ or I/O READ operation is in progress. The $\overline{RD}$ signal is used with B/A Select, C/D Select, $\overline{CE}$, and $\overline{IORQ}$ signals to transfer data from the Z80-PIO to the Z80-CPU.

IEI — Interrupt Enable In (input, active high)
This signal is used to form a priority interrupt daisy chain when more than one interrupt driven device is being used. A high level on this pin indicates that no other devices of higher priority are being serviced by a CPU interrupt service routine.

IEO — Interrupt Enable Out (output, active high)
The IEO signal is the other signal required to form a daisy chain priority scheme. It is high only if IEI is high and the CPU is not servicing an interrupt from this PIO. Thus this signal blocks lower priority devices from interrupting while a higher priority device is being serviced by its CPU interrupt service routine.

$\overline{INT}$ — Interrupt Request (output, open drain, active low)
When $\overline{INT}$ is active the Z80-PIO is requesting an interrupt from the Z80-CPU.

$A_0 - A_7$ — Port A Bus (bidirectional, tristate)
This 8 bit bus is used to transfer data and/or status or control information between Port A of the Z80-PIO and a peripheral device. A_0 is the least significant bit of the Port A data bus.

$\overline{A\ STB}$ — Port A Strobe Pulse from Peripheral Device (input, active low)
The meaning of this signal depends on the mode of operation selected for Port A as follows:

1) Output mode: The positive edge of this strobe is issued by the peripheral to acknowledge the receipt of data made available by the PIO.
2) Input mode: The strobe is issued by the peripheral to load data from the peripheral into the Port A input register. Data is loaded into the PIO when this signal is active.
3) Bidirectional mode: When this signal is active, data from the Port A output register is gated onto Port A bidirectional data bus. The positive edge of the strobe acknowledges the receipt of the data.
4) Control mode: The strobe is inhibited internally.

A RDY — Register A Ready (output, active high)
The meaning of this signal depends on the mode of operation selected for Port A as follows:

1) Output mode: This signal goes active to indicate that the Port A output register has been loaded and the peripheral data bus is stable and ready for transfer to the peripheral device.
2) Input mode: This signal is active when the Port A input register is empty and is ready to accept data from the peripheral device.
3) Bidirectional mode: This signal is active when data is available in the Port A output register for transfer to the peripheral device. In this mode data is not placed on the Port A data bus unless $\overline{A\ STB}$ is active.
4) Control mode: This signal is disabled and forced to a low state.

$B_0 - B_7$ — Port B Bus (bidirectional, tristate)
This 8 bit bus is used to transfer data and/or status or control information between Port B of the PIO and a peripheral device. The Port B data bus is capable of supplying 1.5ma @ 1.5V to drive Darlington transistors. B_0 is the least significant bit of the bus.

$\overline{B\ STB}$ — Port B Strobe Pulse from Peripheral Device (input, active low)
The meaning of this signal is similar to that of $\overline{A\ STB}$ with the following exception:
In the Port A bidirectional mode this signal strobes data from the peripheral device into the Port A input register.

B RDY — Register B Ready (output, active high)
The meaning of this signal is similar to that of A Ready with the following exception:
In the Port A bidirectional mode this signal is high when the Port A input register is empty and ready to accept data from the peripheral device.

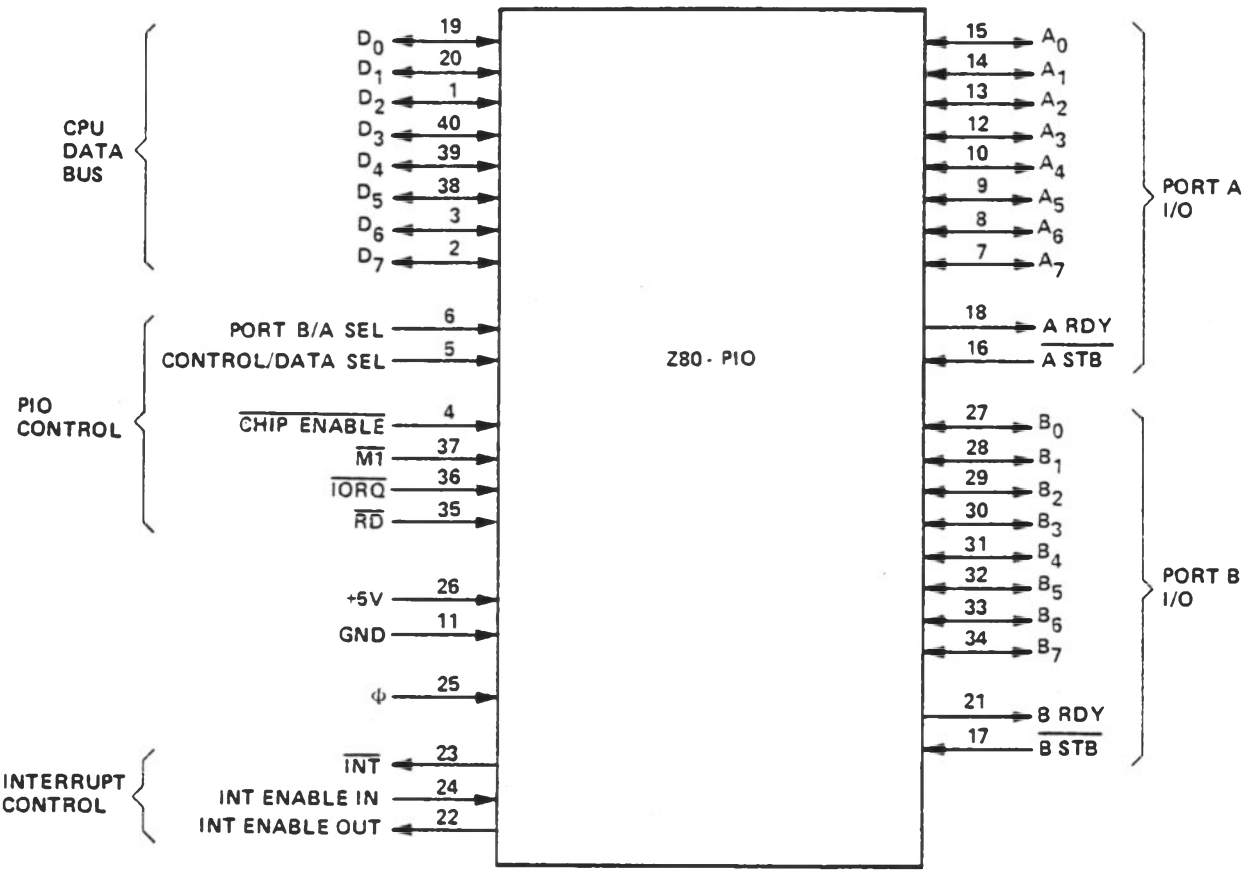


FIGURE 3.0-1
PIO PIN CONFIGURATION

4.0 PROGRAMMING THE PIO

4.1 RESET

The Z80-PIO automatically enters a reset state when power is applied. The reset state performs the following functions:

1) Both port mask registers are reset.
2) Port data bus lines are set to a high impedance state and the Ready "handshake" signals are inactive (low).
3) The vector address registers are *not* reset.
4) Both port interrupt enable flip flops are reset.
5) Both port output registers are reset.

In addition to the automatic power on reset, the PIO can be reset by applying an $\overline{M1}$ signal without the presence of a $\overline{RD}$ or $\overline{IORQ}$ signal. If no $\overline{RD}$ or $\overline{IORQ}$ is detected during $\overline{M1}$ the $\overline{PIO}$ will enter the reset state immediately after the $\overline{M1}$ signal goes inactive. The purpose of this reset is to allow a single external gate to generate a reset without a power down sequence. This approach was required due to the 40 pin packaging limitation.

Once the PIO has entered the internal reset state it is held there until the PIO receives a control word from the CPU.

4.2 LOADING THE INTERRUPT VECTOR

The PIO has been designed to operate with the Z80-CPU using the mode 2 interrupt response. This mode requires that an interrupt vector be supplied by the interrupting device. This vector is used by the CPU to form the address for the interrupt service routine of that port. This vector is placed on the Z-80 data bus during an interrupt acknowledge cycle by the highest priority device requesting service at that time. (Refer to the Z80-CPU Technical Manual for details on how an interrupt is serviced by the CPU). The desired interrupt vector is loaded into the PIO by writing a control word to the desired port of the PIO with the following format:

D7	D6	D5	D4	D3	D2	D1	D0
V7	V6	V5	V4	V3	V2	V1	0

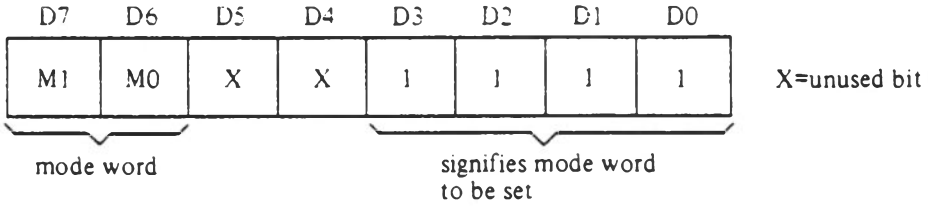
signifies this control word is an interrupt vector

D0 is used in this case as a flag bit which when low causes V7 thru V1 to be loaded into the vector register. At interrupt acknowledge time, the vector of the interrupting port will appear on the Z-80 data bus exactly as shown in the format above.

4.3 SELECTING AN OPERATING MODE

Port A of the PIO may be operated in any of four distinct modes: Mode 0 (output mode), Mode 1 (input mode), Mode 2 (bidirectional mode), and Mode 3 (control mode). Note that the mode numbers have been selected for mnemonic significance: i.e. 0=Out, 1=In, 2=Bidirectional. Port B can operate in any of these modes except Mode 2.

The mode of operation must be established by writing a control word to the PIO in the following format:



Bits D7 and D6 from the binary code for the desired mode according to the following table:

D7	D6	Mode
0	0	0 (output)
0	1	1 (input)
1	0	2 (bidirectional)
1	1	3 (control)

Bits D5 and D4 are ignored. Bits D3-D0 must be set to 1111 to indicate "Set Mode".

Selecting Mode 0 enables any data written to the port output register by the CPU to be enabled onto the port data bus. The contents of the output register may be changed at any time by the CPU simply by writing a new data word to the port. Also the current contents of the output register may be read back to the Z80-CPU at any time through the execution of an input instruction.

With Mode 0 active, a data write from the CPU causes the Ready handshake line of that port to go high to notify the peripheral that data is available. This signal remains high until a strobe is received from the peripheral. The rising edge of the strobe generates an interrupt (if it has been enabled) and causes the Ready line to go inactive. This very simple handshake is similar to that used in many peripheral devices.

Selecting Mode 1 puts the port into the input mode. To start handshake operation, the CPU merely performs an input read operation from the port. This activates the Ready line to the peripheral to signify that data should be loaded into the empty input register. The peripheral device then strobes data into the port input register using the strobe line. Again, the rising edge of the strobe causes an interrupt request (if it has been enabled) and deactivates the Ready signal.

Mode 2 is a bidirectional data transfer mode which uses all four handshake lines. Therefore only Port A may be used for Mode 2 operation. Mode 2 operation uses the Port A handshake signals for output control and the Port B handshake signals for input control. Thus, both A RDY and B RDY may be active simultaneously. The only operational difference between Mode 0 and the output portion of Mode 2 is that data from the Port A output register is allowed on to the port data bus only when $\overline{\text{A STB}}$ is active in order to achieve a bidirectional capability.

Mode 3 operation is intended for status and control applications and does not utilize the handshake signals. When Mode 3 is selected, the next control word sent to that port defines which of the port data bus lines are to be inputs and which are outputs. The format of the control word is shown below:

D7	D6	D5	D4	D3	D2	D1	D0
I/O_7	I/O_6	I/O_5	I/O_4	I/O_3	I/O_2	I/O_1	I/O_0

If any bit is set to a one, then the corresponding data bus line will be used as an input. Conversely, if the bit is reset, the line will be used as an output.

During Mode 3 operation the strobe signal is ignored and the Ready line is held low. Data may be written to a port or read from a port by the Z80-CPU at any time during Mode 3 operation. When reading a port, the data returned to the CPU will be composed of input data from port data bus lines assigned as inputs plus port output register data from those lines assigned as outputs.

4.4 SETTING THE INTERRUPT CONTROL WORD

The interrupt control word for each port has the following format:

D7	D6	D5	D4	D3	D2	D1	D0
Enable Interrupt	AND/ OR	High/ Low	Masks follows	0	1	1	1

used in Mode 3 only — signifies interrupt control word

If bit D7=1 the interrupt enable flip flop of the port is set and the port may generate an interrupt. If bit D7=0 the enable flag is reset and interrupts may not be generated. If an interrupt is pending when the enable flag is set, it will then be enabled onto the CPU interrupt request line. Bits D6, D5, and D4 are used only with Mode 3 operation. They are disregarded for all other modes. These three bits are used to allow for interrupt operation in Mode 3 when any group of the I/O lines go to certain defined states. Bit D6 (AND/ OR) defines the logical operation to be performed in port monitoring. If bit D6=1 an AND function is specified and if D6=0, an OR function is specified. For example, if the AND function is specified, all bits must go to a specified state before an interrupt will be generated while the OR function will generate an interrupt if any specified bit goes to the active state.

Bit D5 defines the active polarity of the port data bus line to be monitored. If bit D5=1 the port data lines are monitored for a high state while if D5=0 they will be monitored for a low state.

If bit D4=1 the next control word sent to the port will be interpreted as a mask as follows:

D7	D6	D5	D4	D3	D2	D1	D0
MB_7	MB_6	MB_5	MB_4	MB_3	MB_2	MB_1	MB_0

Only those port lines whose mask bit is zero will be monitored for generating an interrupt.

5.0 TIMING

5.1 OUTPUT MODE (MODE 0)

Figure 5.0-1 illustrates the timing associated with Mode 0 operation. An output cycle is always started by the execution of an output instruction by the CPU. A $\overline{WR}$* pulse is generated by the PIO during a CPU I/O write operation and is used to latch the data from the CPU data bus into the addressed port's (A or B) output register. The rising edge of the $\overline{WR}$* pulse then raises the Ready flag after the next falling edge of Φ to indicate that data is available for the peripheral device. In most systems the rising edge of the Ready signal can be used as a latching signal in the peripheral device if desired. The Ready signal will remain active until: (1) a positive edge is received from the strobe line indicating that the peripheral has taken the data, or (2) if already active, Ready will be forced low 1½ Φ cycles after the leading edge of $\overline{IORQ}$ if the port's output register is written into. Ready will return high on the first falling edge of Φ after the trailing edge of $\overline{IORQ}$. This guarantees that Ready is low when port data is changing. The Ready signal will not go inactive until a falling edge occurs on the clock (Φ) line. The purpose of delaying the negative transition of the Ready signal until after a negative clock transition is that it allows for a very simple generation scheme for the strobe pulse. By merely connecting the Ready line to the Strobe line, a strobe with a duration of one clock period will be generated with no other logic required. The positive edge of the strobe pulse automatically generates an $\overline{INT}$ request if the interrupt enable flip flop has been set and this device is the highest priority device requesting an interrupt.

If the PIO is not in a reset state, the output register may be loaded before mode 0 is selected. This allows the port output lines to become active in a user defined state.

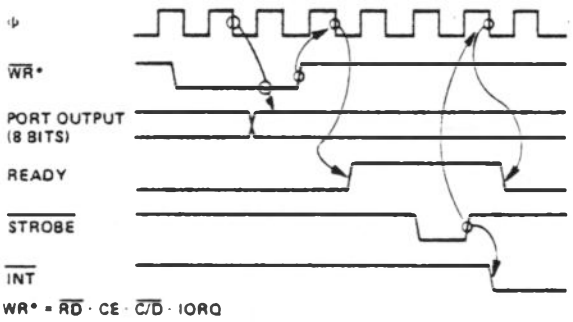


FIGURE 5.0-1
MODE 0 (OUTPUT) TIMING

5.2 INPUT MODE (MODE 1)

Figure 5.0-2 illustrates the timing of an input cycle. The peripheral initiates this cycle using the strobe line after the CPU has performed a data read. A low level on this line loads data into the port input register and the rising edge of the strobe line activates the interrupt request line ($\overline{INT}$) if the interrupt enable is set and this is the highest priority requesting device. The next falling edge of the clock line (Φ) will then reset the Ready line to an inactive state signifying that the input register is full and further loading must be inhibited until the CPU reads the data. The CPU will in the course of its interrupt service routine, read the data from the interrupting port. When this occurs, the positive edge from the CPU $\overline{RD}$ signal will raise the Ready line with the next low going transition of Φ, indicating that new data can be loaded into the PIO. If already active, Ready will be forced low one and one-half Φ periods following the leading edge of $\overline{IORQ}$ during a read of a PIO port. If the user strobes data into the PIO only when Ready is high, the forced state of Ready will prevent input register data from changing while the CPU is reading the PIO. Ready will go high again after the trailing edge of the $\overline{IORQ}$ as previously described.

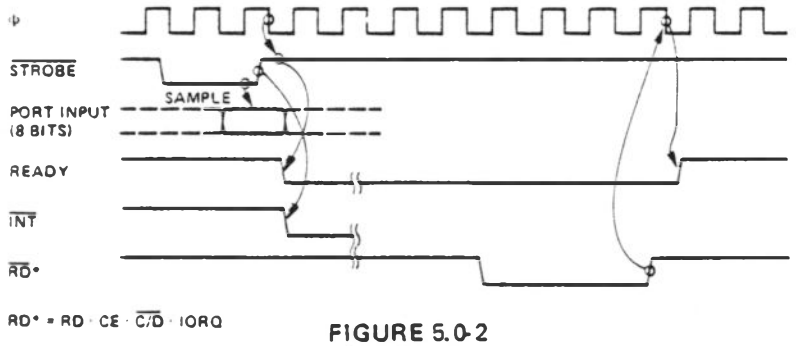


FIGURE 5.0-2
MODE 1 (INPUT) TIMING

5.3 BIDIRECTIONAL MODE (MODE 2)

This mode is merely a combination of Mode 0 and Mode 1 using all four handshake lines. Since it requires all four lines, it is available only on Port A. When this mode is used on Port A, Port B must be set to the Bit Control Mode. The same interrupt vector will be returned for a Mode 3 interrupt on Port B and an input transfer interrupt during Mode 2 operation of Port A. Ambiguity is avoided if Port B is operated in a polled mode and the Port B mask register is set to inhibit all bits.

Figure 5.0-3 illustrates the timing for this mode. It is almost identical to that previously described for Mode 0 and Mode 1 with the Port A handshake lines used for output control and the Port B lines used for input control. The difference between the two modes is that, in Mode 2, data is allowed out onto the bus only when the A strobe is low. The rising edge of this strobe can be used to latch the data into the peripheral since the data will remain stable until after this edge. The input portion of Mode 2 operates identically to Mode 1. Note that both Port A and Port B must have their interrupts enabled to achieve an interrupt driven bidirectional transfer.

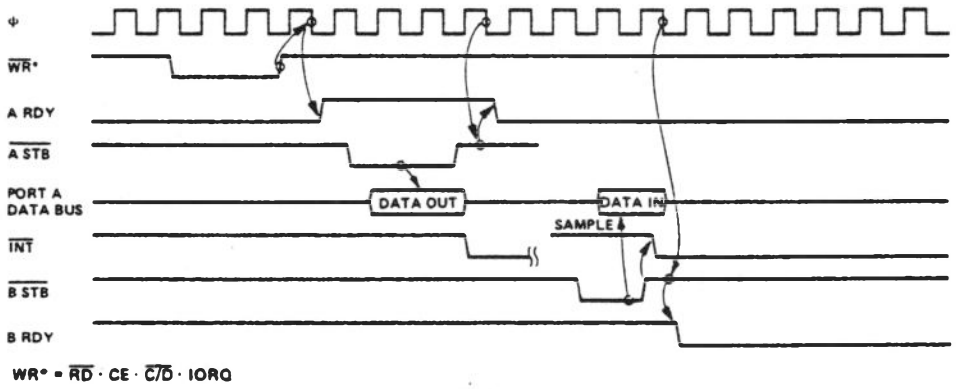


FIGURE 5.0-3
PORT A, MODE 2 (BIDIRECTIONAL) TIMING

The peripheral must not gate data onto a port data bus while $\overline{\text{A STB}}$ is active. Bus contention is avoided if the peripheral uses $\overline{\text{B STB}}$ to gate input data onto the bus. The PIO uses the $\overline{\text{B STB}}$ low level to latch this data. The PIO has been designed with a zero hold time requirement for the data when latching in this mode so that this simple gating structure can be used by the peripheral. That is, the data can be disabled from the bus immediately after the strobe rising edge.

5.4 CONTROL MODE (MODE 3)

The control mode does not utilize the handshake signals and a normal port write or port read can be executed at any time. When writing, the data will be latched into output registers with the same timing as Mode 0. A RDY will be forced low whenever Port A is operated in Mode 3. B RDY will be held low whenever Port B is operated in Mode 3 unless Port A is in Mode 2. In the latter case, the state of B RDY will not be affected.

When reading the PIO, the data returned to the CPU will be composed of output register data from those port data lines assigned as outputs and input register data from those port data lines assigned as inputs. The input register will contain data which was present immediately prior to the falling edge of $\overline{\text{RD}}$. See Figure 5.0-4.

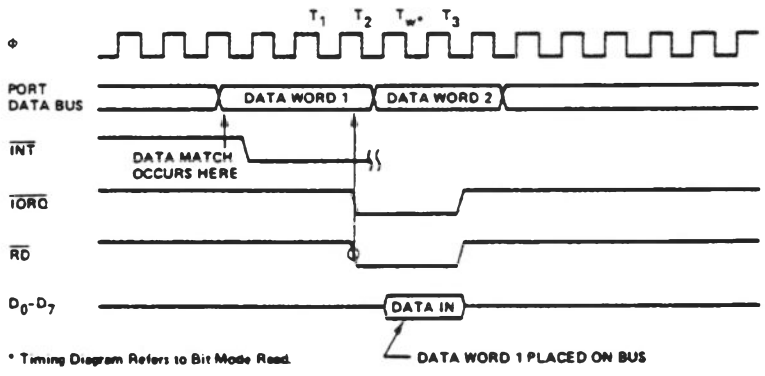


FIGURE 5.0-4

An interrupt will be generated if interrupts from the port are enabled and the data on the port data lines satisfies the logical equation defined by the 8-bit mask and 2-bit mask control registers. Another interrupt will not be generated until a change occurs in the status of the logical equation. A Mode 3 interrupt will be generated only if the result of a Mode 3 logical operation changes from false to true. For example, assume that the Mode 3 logical equation is an "OR" function. An unmasked port data line becomes active and an interrupt is requested. If a second unmasked port data line becomes active concurrently with the first, a new interrupt will not be requested since a change in the result of the Mode 3 logical operation has not occurred.

If the result of a logical operation becomes true immediately prior to or during $\overline{M1}$, an interrupt will be requested after the trailing edge of $\overline{M1}$.

6.0 INTERRUPT SERVICING

Some time after an interrupt is requested by the PIO, the CPU will send out an interrupt acknowledge ($\overline{M1}$ and $\overline{IORQ}$). During this time the interrupt logic of the PIO will determine the highest priority port which is requesting an interrupt. (This is simply the device with its Interrupt Enable Input high and its Interrupt Enable Output low). To insure that the daisy chain enable lines stabilize, devices are inhibited from changing their interrupt request status when $\overline{M1}$ is active. The highest priority device places the contents of its interrupt vector register onto the Z80 data bus during interrupt acknowledge.

Figure 6.0-1 illustrates the timing associated with interrupt requests. During $\overline{M1}$ time, no new interrupt requests can be generated. This gives time for the Int Enable signals to ripple through up to four PIO circuits. The PIO with IEI high and IEO low during $\overline{INTA}$ will place the 8-bit interrupt vector of the appropriate port on the data bus at this time.

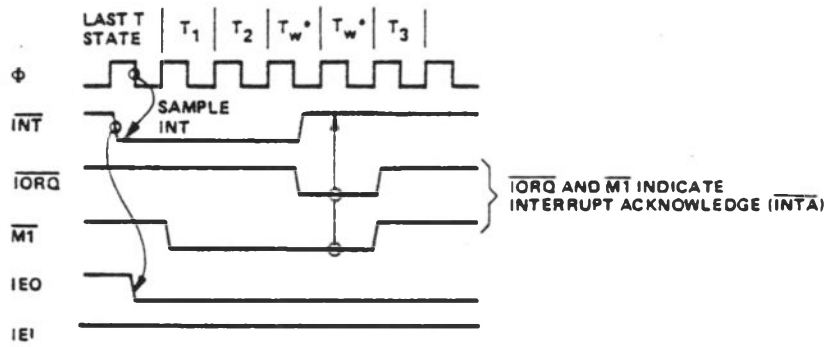


FIGURE 6.0-1
INTERRUPT ACKNOWLEDGE TIMING

If an interrupt requested by the PIO is acknowledged, the requesting port is 'under service'. IEO of this port will remain low until a return from interrupt instruction (RETI) is executed while IEI of the port is high. If an interrupt request is not acknowledged, IEO will be forced high for one $\overline{M1}$ cycle after the PIO decodes the opcode 'ED'. This action guarantees that the two byte RETI instruction is decoded by the proper PIO port. See Figure 6.0-2.

Figure 6.0-3 illustrates a typical nested interrupt sequence that could occur with four ports connected in the daisy chain. In this sequence Port 2A requests and is granted an interrupt. While this port is being serviced, a higher priority port (1B) requests and is granted an interrupt. The service routine for the higher priority port is completed and a RETI instruction is executed to indicate to the port that its routine is complete. At this time the service routine of the lower priority port is completed.

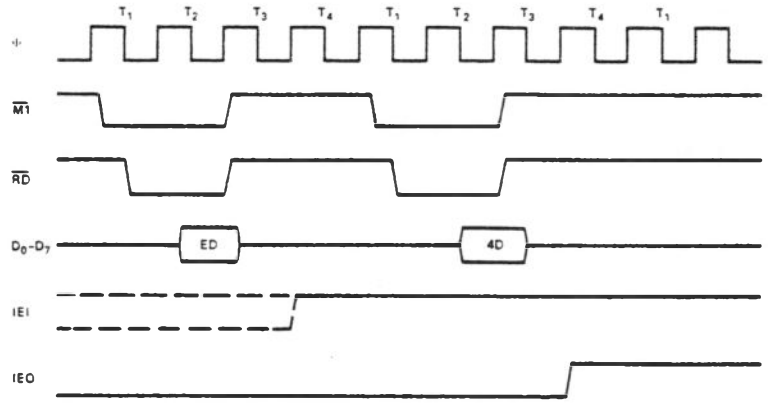
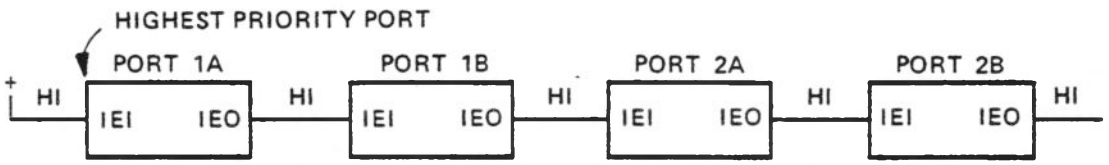
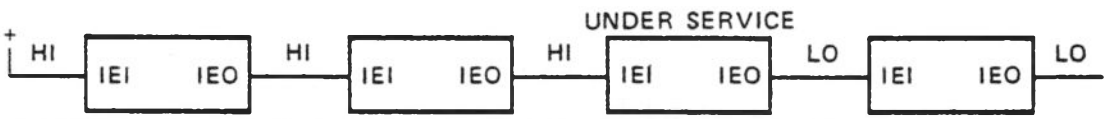


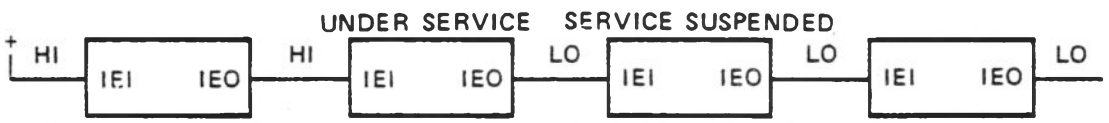
FIGURE 6.0-2
RETURN FROM INTERRUPT CYCLE



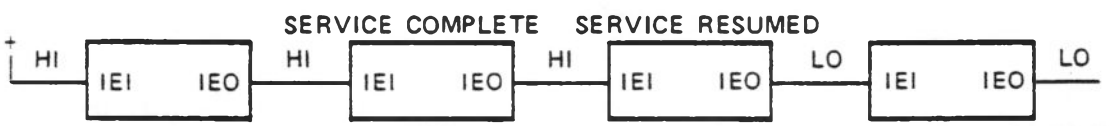
1. PRIORITY INTERRUPT DAISY CHAIN BEFORE ANY INTERRUPT OCCURS.



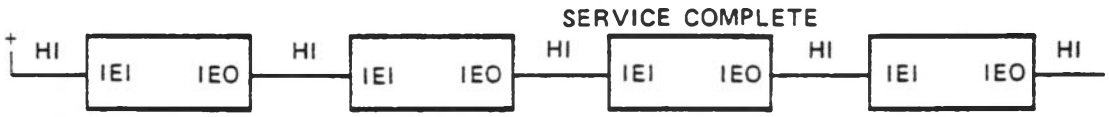
2. PORT 2A REQUESTS AN INTERRUPT AND IS ACKNOWLEDGED.



3. PORT 1B INTERRUPTS, SUSPENDS SERVICING OF PORT 2A.



4. PORT 1B SERVICE ROUTINE COMPLETE, "RETI" ISSUED, PORT 2A SERVICE RESUMED.



5. SECOND "RETI" INSTRUCTION ISSUED ON COMPLETION OF PORT 2A SERVICE ROUTINE.

FIGURE 6.0-3
DAISY CHAIN INTERRUPT SERVICING

7.0 APPLICATIONS

7.1 EXTENDING THE INTERRUPT DAISY CHAIN

Without any external logic, a maximum of four Z80-PIO devices may be daisy chained into a priority interrupt structure. This limitation is required so that the interrupt enable status (IEO) ripples through the entire chain between the beginning of $\overline{M1}$, and the beginning of $\overline{IORQ}$ during an interrupt acknowledge cycle. Since the interrupt enable status cannot change during $\overline{M1}$, the vector address returned to the CPU is assured to be from the highest priority device which requested an interrupt.

If more than four PIO devices must be accommodated, a "look-ahead" structure may be used as shown in figure 7.0-1. With this technique more than thirty PIO's may be chained together using standard TTL logic.

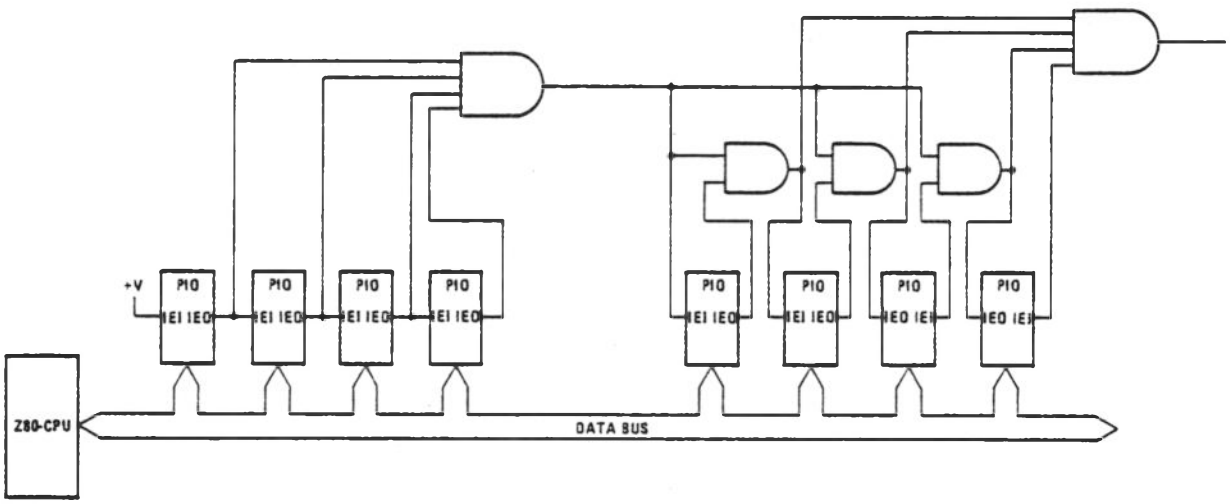


FIGURE 7.0-1
A METHOD OF EXTENDING THE INTERRUPT PRIORITY DAISY CHAIN

7.2 I/O DEVICE INTERFACE

In this example, the Z80-PIO is connected to an I/O terminal device which communicates over an 8 bit parallel bidirectional data bus as illustrated in figure 7.0-2. Mode 2 operation (bidirectional) is selected by sending the following control word to Port A:

D7	D6	D5	D4	D3	D2	D1	D0
1	0	X	X	1	1	1	1

Mode Control

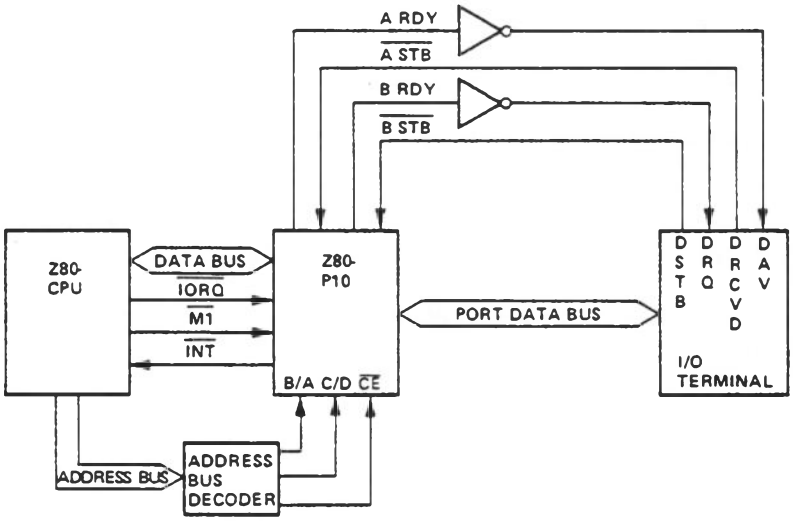


FIGURE 7.0-2

EXAMPLE I/O INTERFACE

Next, the proper interrupt vector is loaded (refer to CPU Manual for details on the operation of the interrupt).

V7	V6	V5	V4	V3	V2	V1	0

Interrupts are then enabled by the rising edge of the first $\overline{M1}$ after the interrupt mode word is set unless that $\overline{M1}$ defines an interrupt acknowledge cycle. If a mask follows the interrupt mode word, interrupts are enabled by the rising edge of the first $\overline{M1}$ following the setting of the mask.

Data can now be transferred between the peripheral and the CPU. The timing for this transfer is as described in Section 5.0.

7.3 CONTROL INTERFACE

A typical control mode application is illustrated in figure 7.0-3. Suppose an industrial process is to be monitored. The occurrence of any abnormal operating condition is to be reported to a Z80-CPU based control system. The process control and status word has the following format:

D7	D6	D5	D4	D3	D2	D1	D0
Special Test	Turn On Power	Power Failure Alarm	Halt Process-ing	Temp. Alarm	Turn Heaters On	Pressur-ize System	Pressure Alarm

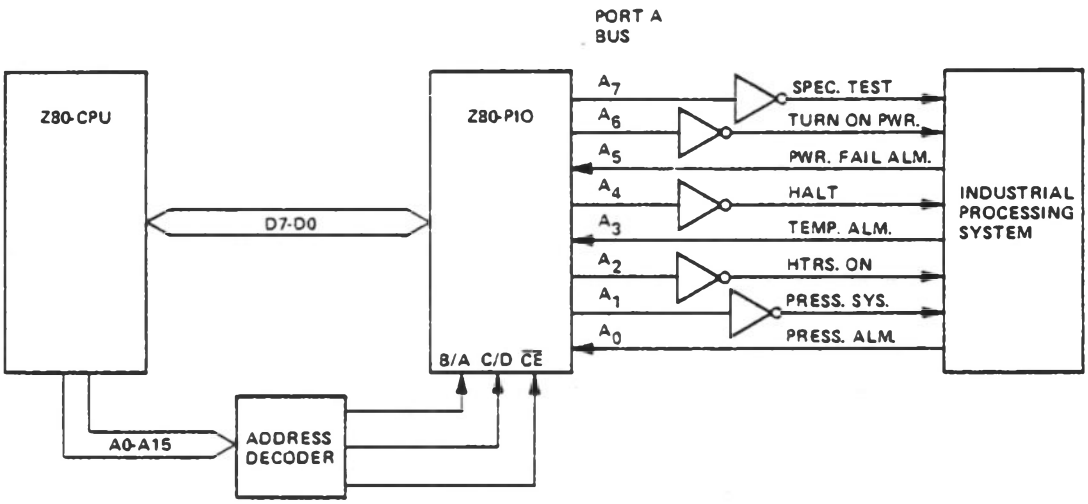


FIGURE 7.0-3
CONTROL MODE APPLICATION

The PIO may be used as follows. First Port A is set for Mode 3 operation by writing the following control word to Port A.

D7	D6	D5	D4	D3	D2	D1	D0
1	1	X	X	1	1	1	1

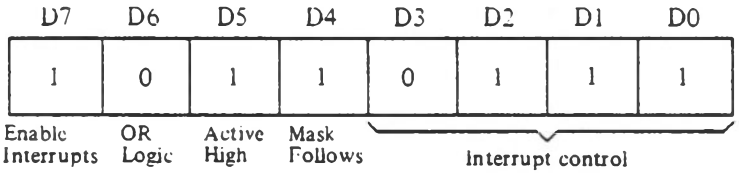
Whenever Mode 3 is selected, the next control word sent to the port must be an I/O select word. In this example we wish to select port data lines A5, A3 and A0 as inputs and so the following control word is written:

D7	D6	D5	D4	D3	D2	D1	D0
0	0	1	0	1	0	0	1

Next the desired interrupt vector must be loaded (refer to the CPU manual for details);

D7	D6	D5	D4	D3	D2	D1	D0
V7	V6	V5	V4	V3	V2	V1	0

An interrupt control word is next sent to the port:



The mask word following the interrupt mode word is:

D7	D6	D5	D4	D3	D2	D1	D0
1	1	0	1	0	1	1	0

Selects A5, A3 and A0 to be monitored

Now, if a sensor puts a high level on line A5, A3, or A0, an interrupt request will be generated. The mask word may select any combination of inputs or outputs to cause an interrupt. For example, if the mask word above had been:

D7	D6	D5	D4	D3	D2	D1	D0
0	1	0	1	0	1	1	0

then an interrupt request would also occur if bit A7 (Special Test) of the output register was set.

Assume that the following port assignments are to be used:

$E0_H$= Port A Data
$E1_H$= Port B Data
$E2_H$= Port A Control
$E3_H$= Port B Control

All port numbers are in hexadecimal notation. This particular assignment of port numbers is convenient since A_0 of the address bus can be used as the Port B/A Select and A_1 of the address bus can be used as the Control/Data Select. The Chip Enable would be the decode of CPU address bits A_7 thru A_2 (1110 00). Note that if only a few peripheral devices are being used, a Chip Enable decode may not be required since a higher order address bit could be used directly.

8.0 PROGRAMMING SUMMARY

8.1 LOAD INTERRUPT VECTOR

V7	V6	V5	V4	V3	V2	V1	0

8.2 SET MODE

M1	M0	X	X	1	1	1	1

M_1	M_0	Mode
0	0	Output
0	1	Input
1	0	Bidirectional
1	1	Bit Control

When selecting Mode 3, the next word must set the I/O Register:

I/O_7	I/O_6	I/O_5	I/O_4	I/O_3	I/O_2	I/O_1	I/O_0

I/O = 1 Sets bit to Input

I/O = 0 Sets bit to Output

8.3 SET INTERRUPT CONTROL

Int Enable	AND/ OR	High/ Low	Mask Follows	0	1	1	1

Used in Mode 3 only

If the "mask follows" bit is high, the next control word written to the port must be the mask:

MB_7	MB_6	MB_5	MB_4	MB_3	MB_2	MB_1	MB_0

MB = 0, Monitor bit

MB = 1, Mask bit from being monitored

Also, the interrupt enable flip flop of a port may be set or reset without modifying the rest of the interrupt control word by using the following command:

Int Enable	X	X	X	0	0	1	1

Absolute Maximum Ratings

Temperature Under Bias	Specified operating range.
Storage Temperature	-65° C to +150° C
Voltage On Any Pin With Respect To Ground	-0.3 V to +7 V
Power Dissipation	6 W

*Comment

Stresses above those listed under "Absolute Maximum Rating" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other condition above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

Note: All AC and DC characteristics remain the same for the military grade parts except I_{CC}:

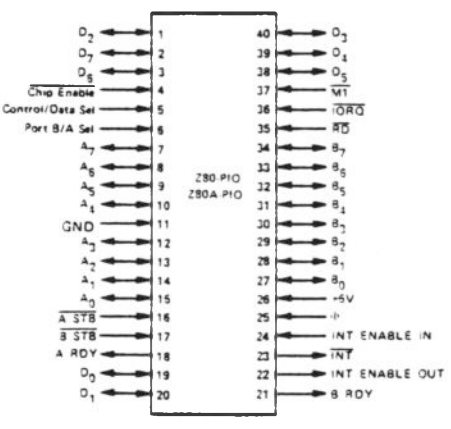
I_{CC} = 130 mA

Z80-PIO and Z80A-PIO D.C. Characteristics

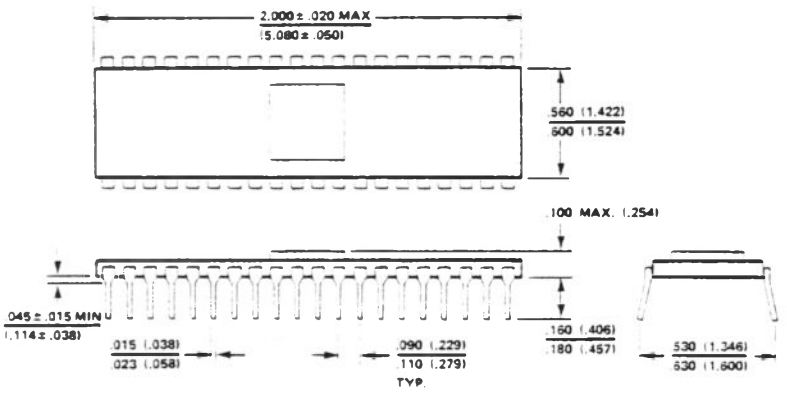
T_A = 0° C to 70° C, V_{CC} = 5 V ± 5% unless otherwise specified

Symbol	Parameter	Min.	Max.	Unit	Test Condition
V_{ILC}	Clock Input Low Voltage	-0.3	.45	V	
V_{IHC}	Clock Input High Voltage	V_{CC}-.6	V_{CC}+.3	V	
V_{IL}	Input Low Voltage	-0.3	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC}	V	
V_{OL}	Output Low Voltage		0.4	V	I_{OL} = 2.0 mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = -250 μA
I_{CC}	Power Supply Current		70	mA	
I_{LI}	Input Leakage Current		±10	μA	V_{IN} = 0 to V_{CC}
I_{LOH}	Tri-State Output Leakage Current in Float		10	μA	V_{OUT} = 2.4 to V_{CC}
I_{LOL}	Tri-State Output Leakage Current in Float		-10	μA	V_{OUT} = 0.4 V
I_{LD}	Data Bus Leakage Current in Input Mode		±10	μA	$0 \leq V_{IN} \leq V_{CC}$
I_{OHD}	Darlington Drive Current	-1.5	3.8	mA	V_{OH} = 1.5 V R_{EXT} = 390 Ω Port B Only

Package Configuration



Package Outline



NOTE: Dimensions in parentheses are for metric system (cm).

TA = 0° C to 70° C, Vcc = +5 V ± 5%, unless otherwise noted

SIGNAL	SYMBOL	PARAMETER	MIN	MAX	UNIT	COMMENTS
Φ	t_c	Clock Period	400	[1]	nsec	
	$t_{W(\Phi H)}$	Clock Pulse Width, Clock High	170	2000	nsec	
	$t_{W(\Phi L)}$	Clock Pulse Width, Clock Low	170	2000	nsec	
	t_r, t_f	Clock Rise and Fall Times		30	nsec	
	t_H	Any Hold Time for Specified Set-Up Time	0		nsec	
CS, $\overline{CE}$ ETC.	$t_{S\Phi}$ (CS)	Control Signal Set-Up Time to Rising Edge of Φ During Read or Write Cycle	280		nsec	
D_0-D_7	t_{DR} (D)	Data Output Delay from Falling Edge of $\overline{RD}$		430	nsec	[2]
	$t_{S\Phi}$ (D)	Data Set-Up Time to Rising Edge of ΦDuring Write or $\overline{M1}$ Cycle	50		nsec	C_L = 50 pF
	t_{DI} (D)	Data Output Delay from Falling Edge of $\overline{IORQ}$ During INTA Cycle		340	nsec	[3]
	t_F (D)	Delay to Floating Bus (Output Buffer Disable Time)		160	nsec	
IEI	t_S (IEI)	IEI Set-Up Time to Falling Edge of $\overline{IORQ}$ During INTA Cycle	140		nsec	
IEO	t_{DH} (IO)	IEO Delay Time from Rising Edge of IEI		210	nsec	[5]
	t_{DL} (IO)	IEO Delay Time from Falling Edge of IEI		190	nsec	[5] C_L = 50 pF
	t_{DM} (IO)	IEO Delay from Falling Edge of $\overline{M1}$ (Interrupt Occurring Just Prior to $\overline{M1}$) See Note A.		300	nsec	[5]
$\overline{IORQ}$	$t_{S\Phi}$ (IR)	$\overline{IORQ}$ Set-Up Time to Rising Edge of Φ During Read or Write Cycle	250		nsec	
$\overline{M1}$	$t_{S\Phi}$ (M1)	$\overline{M1}$ Set-Up Time to Rising Edge of Φ During INTA or $\overline{M1}$ Cycle. See Note B.	210		nsec	
$\overline{RD}$	$t_{S\Phi}$ (RD)	$\overline{RD}$ Set-Up Time to Rising Edge of Φ During Read or $\overline{M1}$ Cycle	240		nsec	
A_0-A_7, B_0-B_7	t_S (PD)	Port Data Set-Up Time to Rising Edge of $\overline{STROBE}$ (Mode 1)	260		nsec	
	t_{DS} (PD)	Port Data Output Delay from Falling Edge of $\overline{STROBE}$ (Mode 2)		230	nsec	[5]
	t_F (PD)	Delay to Floating Port Data Bus from Rising Edge of $\overline{STROBE}$ (Mode 2)		200	nsec	C_L = 50 pF
	t_{DI} (PD)	Port Data Stable from Rising Edge of $\overline{IORQ}$ During WR Cycle (Mode 0)		200	nsec	[5]
$\overline{ASTB}$, $\overline{BSTB}$	t_W (ST)	Pulse Width, $\overline{STROBE}$	150		nsec	
			[4]		nsec	
$\overline{INT}$	t_D (IT)	$\overline{INT}$ Delay Time from Rising Edge of $\overline{STROBE}$		490	nsec	
	t_D (IT3)	$\overline{INT}$ Delay Time from Data Match During Mode 3 Operation		420	nsec	
ARDY, BRDY	t_{DH} (RY)	Ready Response Time from Rising Edge of $\overline{IORQ}$		t_c+ 460	nsec	[5] C_L = 50 pF
	t_{DL} (RY)	Ready Response Time from Rising Edge of $\overline{STROBE}$		t_c+ 400	nsec	[5]

NOTES:

A. 2.5 t_c > (N-2) t_{DL} (IO) + t_{DM} (IO) + t_S (IEI) + TTL Buffer Delay, if any

B. $\overline{M1}$ must be active for a minimum of 2 clock periods to reset the PIO.

[1] $t_c = t_{W(\Phi H)} + t_{W(\Phi L)} + t_r + t_f$

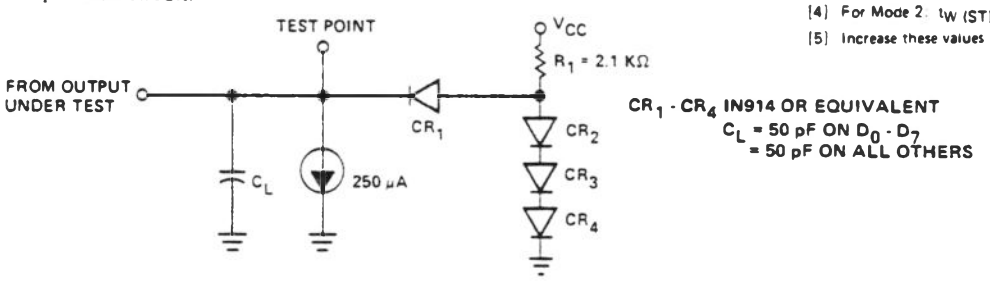
[2] Increase t_{DR} (D) by 10 nsec for each 50 pF increase in loading up to 200 pF max.

[3] Increase t_{DI} (D) by 10 nsec for each 50 pF increase in loading up to 200 pF max.

[4] For Mode 2: t_W (ST) > t_S (PD)

[5] Increase these values by 2 nsec for each 10 pF increase in loading up to 100 pF max.

Output load circuit.



CR_1 - CR_4 IN914 OR EQUIVALENT
C_L = 50 pF ON D_0 - D_7
= 50 pF ON ALL OTHERS

Capacitance

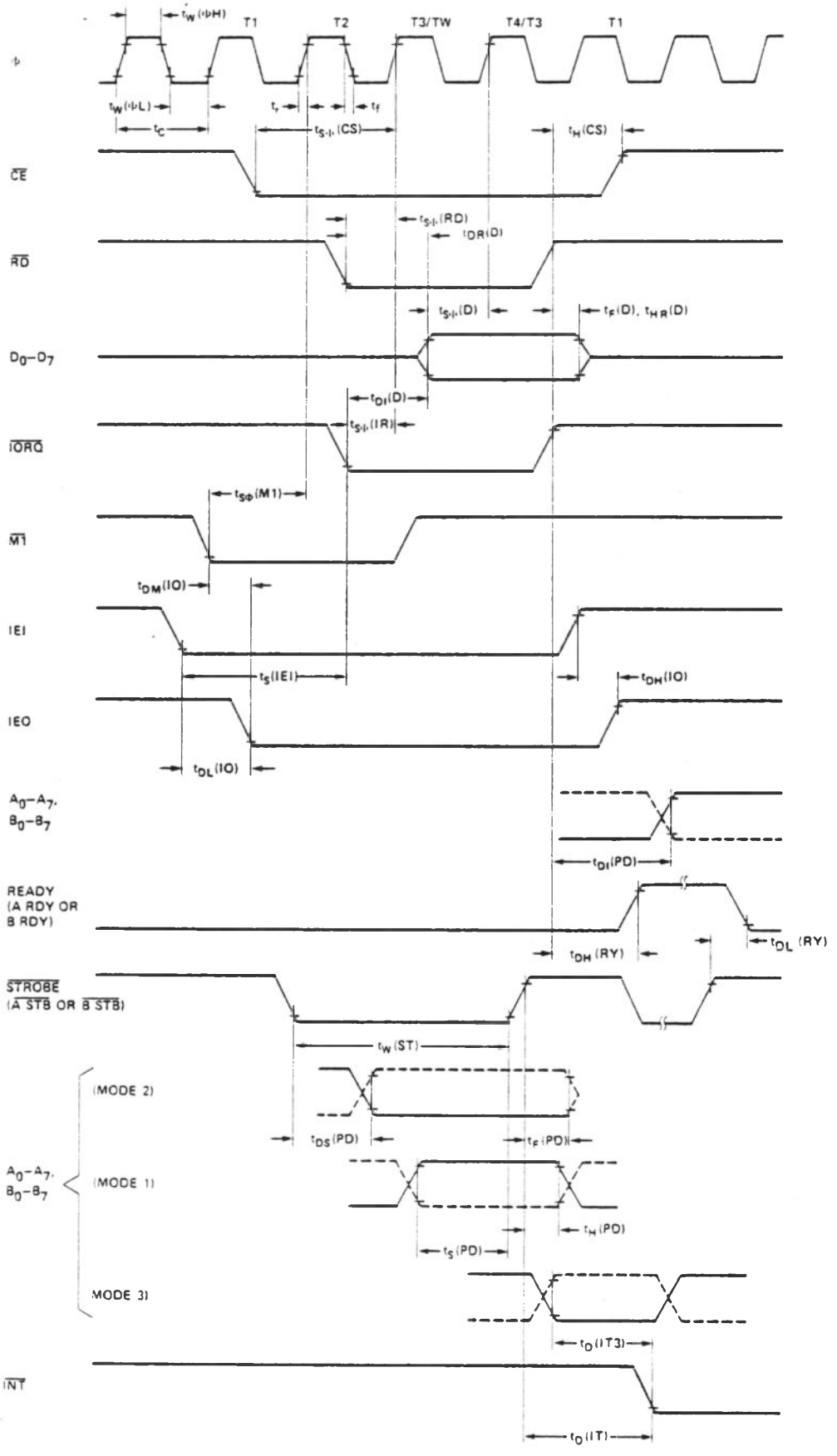
TA = 25° C, f = 1 MHz

Symbol	Parameter	Max.	Unit	Test Condition
C_Φ	Clock Capacitance	10	pF	Unmeasured Pins Returned to Ground
C_{IN}	Input Capacitance	5	pF	
C_{OUT}	Output Capacitance	10	pF	

A.C. Timing Diagram

Timing measurements are made at the following voltages, unless otherwise specified:

	"1"	"0"
CLOCK	$V_{CC}-.6$	.45V
OUTPUT	2.0V	0.8V
INPUT	2.0V	0.8V
FLOAT	ΔV =	+0.5V



T_A = 0° C to 70° C, Vcc = +5 V ± 5%, unless otherwise noted

SIGNAL	SYMBOL	PARAMETER	MIN	MAX	UNIT	COMMENTS
Φ	t_c	Clock Period	250	[1]	nsec	
	$t_{W(\Phi H)}$	Clock Pulse Width, Clock High	105	2000	nsec	
	$t_{W(\Phi L)}$	Clock Pulse Width, Clock Low	105	2000	nsec	
	t_r, t_f	Clock Rise and Fall Times		30	nsec	
	t_h	Any Hold Time for Specified Set-Up Time	0		nsec	
CS, $\overline{CE}$ ETC.	$t_{S\Phi(CS)}$	Control Signal Set-Up Time to Rising Edge of Φ During Read or Write Cycle	145		nsec	
D_0-D_7	$t_{DR(D)}$	Data Output Delay From Falling Edge of $\overline{RD}$		380	nsec	[2]
	$t_{S\Phi(D)}$	Data Set-Up Time to Rising Edge of Φ During Write or $\overline{M1}$ Cycle	50		nsec	C_L = 50 pF
	$t_{DI(D)}$	Data Output Delay from Falling Edge of $\overline{IORQ}$ During INTA Cycle		250	nsec	[3]
	$t_{F(D)}$	Delay to Floating Bus (Output Buffer Disable Time)		110	nsec	
IEI	$t_{S(IEI)}$	IEI Set-Up Time to Falling edge of $\overline{IORQ}$ During INTA Cycle	140		nsec	
IEO	$t_{DH(IO)}$	IEO Delay Time from Rising Edge of IEI		160	nsec	[5]
	$t_{DL(IO)}$	IEO Delay Time from Falling Edge of IEI		130	nsec	[5] C_L = 50 pF
	$t_{DM(IO)}$	IEO Delay from Falling Edge of $\overline{M1}$ (Interrupt Occurring Just Prior to $\overline{M1}$) See Note A.		190	nsec	[5]
$\overline{IORQ}$	$t_{S\Phi(IR)}$	$\overline{IORQ}$ Set-Up Time to Rising Edge of Φ During Read or Write Cycle.	115		nsec	
$\overline{M1}$	$t_{S\Phi(M1)}$	$\overline{M1}$ Set-Up Time to Rising Edge of Φ During INTA or $\overline{M1}$ Cycle See Note B	90		nsec	
$\overline{RD}$	$t_{S\Phi(RD)}$	$\overline{RD}$ Set-Up Time to Rising Edge of Φ During Read or $\overline{M1}$ Cycle	115		nsec	
A_0-A_7, B_0-B_7	$t_{S(PD)}$	Port Data Set-Up Time to Rising Edge of $\overline{STROBE}$ (Mode 1)	230		nsec	
	$t_{DS(PD)}$	Port Data Ourput Delay from Falling Edge of $\overline{STROBE}$ (Mode 2)		210	nsec	[5]
	$t_{F(PD)}$	Delay to Floating Port Data Bus from Rising Edge of $\overline{STROBE}$ (Mode 2)		180	nsec	C_L = 50 pF
	$t_{DI(PD)}$	Port Data Stable from Rising Edge of $\overline{IORQ}$ During WR Cycle (Mode 0)		180	nsec	[5]
$\overline{ASTB}$, $\overline{BSTB}$	$t_{W(ST)}$	Pulse Width, $\overline{STROBE}$	150 [4]		nsec nsec	
$\overline{INT}$	$t_{D(IT)}$	$\overline{INT}$ Delay time from Rising Edge of $\overline{STROBE}$		440	nsec	
	$t_{D(IT3)}$	$\overline{INT}$ Delay Time from Data Match During Mode 3 Operation		380	nsec	
ARDY, BRDY	$t_{DH(RY)}$	Ready Response Time from Rising Edge of $\overline{IORQ}$		t_c+ 410	nsec	[5] C_L = 50 pF
	$t_{DL(RY)}$	Ready Response Time from Rising Edge of $\overline{STROBE}$		t_c+ 360	nsec	[5]

NOTES:

A. 2.5 t_c > (N-2) $t_{DL(IO)}$ + $t_{DM(IO)}$ + $t_{S(IEI)}$ + TTL Buffer Delay, if any

B. $\overline{M1}$ must be active for a minimum of 2 clock periods to reset the PIO.

[1] $t_c = t_{W(\Phi H)} + t_{W(\Phi L)} + t_r + t_f$

[2] Increase $t_{DR(D)}$ by 10 nsec for each 50 pF increase in loading up to 200 pF max.

[3] Increase $t_{DI(D)}$ by 10 nsec for each 50 pF increase in loading up to 200 pF max.

[4] For Mode 2: $t_{W(ST)} > t_{S(PD)}$

[5] Increase these values by 2 nsec for each 10 pF increase in loading up to 100 pF max.

Z80™-CTC
Z80A™-CTC

Technical Manual

TABLE OF CONTENTS

1.0 Introduction 1

2.0 CTC Architecture 2
2.1 Overview 2
2.2 Structure of Channel Logic 3
2.2.1 The Channel Control 3
2.2.2 The Prescaler 4
2.2.3 The Time Constant Register 4
2.2.4 The Down Counter 4
2.3 Interrupt Control Logic 5

3.0 CTC Pin Description 6

4.0 CTC Operating Modes 9
4.1 CTC Counter Mode 9
4.2 CTC Timer Mode 10

5.0 CTC Programming 11
5.1 Loading the Channel Control Register 11
5.2 Loading the Time Constant Register 14
5.3 Loading the Interrupt Vector Register 15

6.0 CTC Timing 16
6.1 CTC Write Cycle 16
6.2 CTC Read Cycle 17
6.3 CTC Counting and Timing 18

7.0 CTC Interrupt Servicing 19
7.1 Interrupt Acknowledge Cycle 19
7.2 Return from Interrupt Cycle 20
7.3 Daisy Chain Interrupt Servicing 21

8.0 Absolute Maximum Ratings 22
8.1 D.C. Characteristics 22
8.2 Capacitance 22
8.3 A.C. Characteristics 23
8.4 A.C. Timing Diagram 24
8.5 A.C. Characteristics 25
8.6 Package Configuration and Package Outline 26

1.0 INTRODUCTION

The Z80-Counter Timer Circuit (CTC) is a progammable component with four independent channels that provide counting and timing functions for microcomputer systems based on the Z80-CPU. The CPU can configure the CTC channels to operate under various modes and conditions as required to interface with a wide range of devices. In most applications, little or no external logic is required. The Z80-CTC utilizes N-channel silicon gate depletion load technology and is packaged in a 28-pin DIP. The Z80-CTC requires only a single 5 volt supply and a one-phase 5 volt clock. Major features of the Z80-CTC include:

- All inputs and outputs fully TTL compatible.
- Each channel may be selected to operate in either Counter Mode or Timer Mode.
- Used in either mode, a CPU-readable Down Counter indicates number of counts-to-go until zero.
- A Time Constant Register can automatically reload the Down Counter at Count Zero in Counter and Timer Mode.
- Selectable positive or negative trigger initiates time operation in Timer Mode. The same input is monitored for event counts in Counter Mode.
- Three channels have Zero Count/Timeout outputs capable of driving Darlington transistors.
- Interrupts may be programmed to occur on the zero count condition in any channel.
- Daisy chain priority interrupt logic included to provide for automatic interrupt vectoring without external logic.

2.0 CTC ARCHITECTURE

2.1 OVERVIEW

A block diagram of the Z80-CTC is shown in figure 2.0-1. The internal structure of the Z80-CTC consists of a Z80-CPU bus interface, Internal Control Logic, four sets of Counter/Timer Channel Logic, and Interrupt Control Logic. The four independent counter/timer channels are identified by sequential numbers from 0 to 3. The CTC has the capability of generating a unique interrupt vector for each separate channel (for automatic vectoring to an interrupt service routine). The 4 channels can be connected into four contiguous slots in the standard Z80 priority chain with channel number 0 having the highest priority. The CPU bus interface logic allows the CTC device to interface directly to the CPU with no other external logic. However, port address decoders and/or line buffers may be required for large systems.

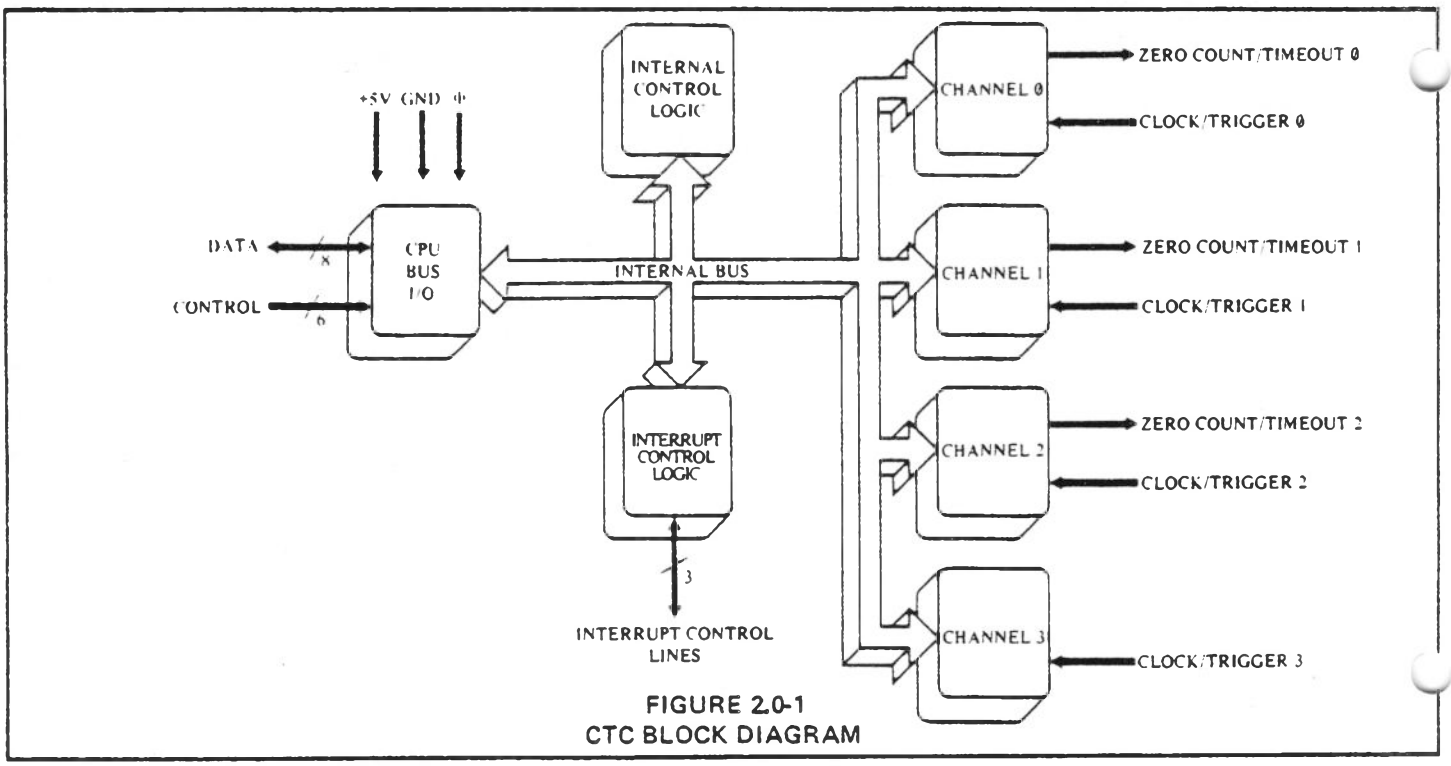


FIGURE 2.0-1
CTC BLOCK DIAGRAM

2.2 STRUCTURE OF CHANNEL LOGIC

The structure of one of the four sets of Counter/Timer Channel Logic is shown in figure 2.0-2. This logic is composed of 2 registers, 2 counters and control logic. The registers are an 8-bit Time Constant Register and an 8-bit Channel Control Register. The counters are an 8-bit CPU-readable Down Counter and an 8-bit Prescaler.

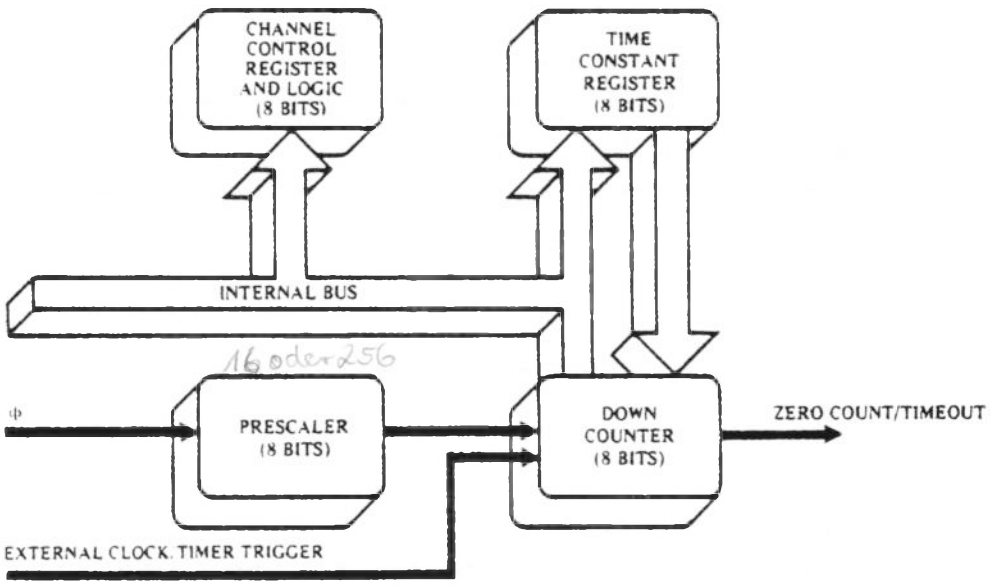


FIGURE 2.0-2
CHANNEL BLOCK DIAGRAM

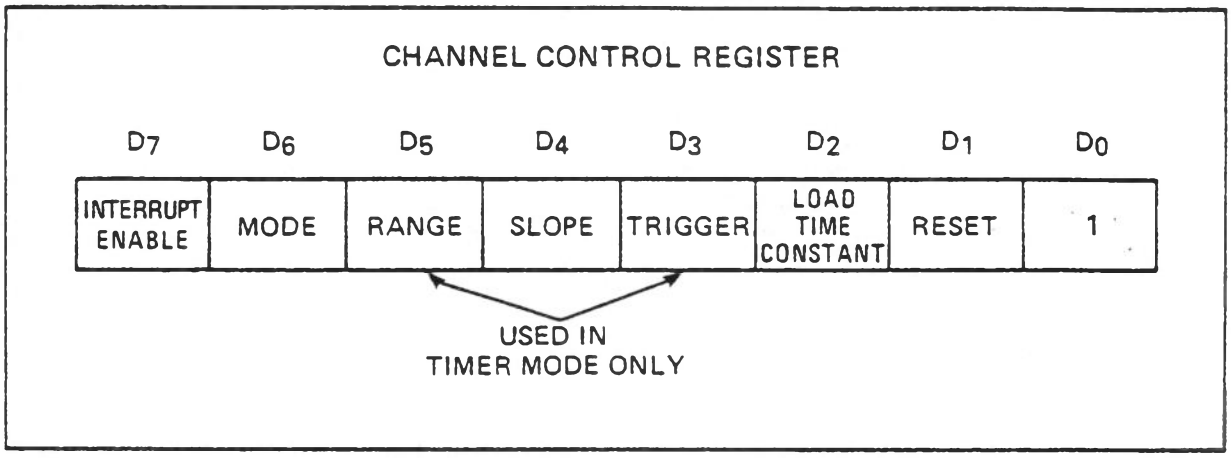
2.2.1 THE CHANNEL CONTROL REGISTER AND LOGIC

The Channel Control Register (8-bit) and Logic is written to by the CPU to select the modes and parameters of the channel. Within the entire CTC device there are four such registers, corresponding to the four Counter/Timer Channels. Which of the four is being written to depends on the encoding of two channel select input pins: CS0 and CS1 (usually attached to A0 and A1 of the CPU address bus). This is illustrated in the truth table below:

	CS1	CS0
Ch 0	0	0
Ch 1	0	1
Ch 2	1	0
Ch 3	1	1

2.2.1 CONTINUED

In the control word written to program each Channel Control Register, bit 0 is always set, and the other 7 bits are programmed to select alternatives on the channel's operating modes and parameters, as shown in the diagram below. (For a more complete discussion see section 4.0: "CTC Operating Modes" and section 5.0: "CTC Programming.")



2.2.2 THE PRESCALER

Used in the Timer Mode only, the Prescaler is an 8-bit device which can be programmed by the CPU via the Channel Control Register to divide its input, the System Clock (Φ), by 16 or 256. The output of the Prescaler is then fed as an input to clock the Down Counter, which initially, and every time it clocks down to zero, is reloaded automatically with the contents of the Time Constant Register. In effect this again divides the System Clock by an additional factor of the time constant. Every time the Down Counter counts down to zero, its output, Zero Count/Timeout (ZC/TO), is pulsed high.

2.2.3 THE TIME CONSTANT REGISTER

The Time Constant Register is an 8-bit register, used in both Counter Mode and Timer Mode, programmed by the CPU just after the Channel Control Word with an integer time constant value of 1 through 256. This register loads the programmed value into the Down Counter when the CTC is first initialized and reloads the same value into the Down Counter automatically whenever it counts down thereafter to zero. If a new time constant is loaded into the Time Constant Register while a channel is counting or timing, the present down count will be completed before the new time constant is loaded into the Down Counter. (For details of how a time constant is written to a CTC channel, see section 5.0: "CTC Programming.")

2.2.4 THE DOWN COUNTER

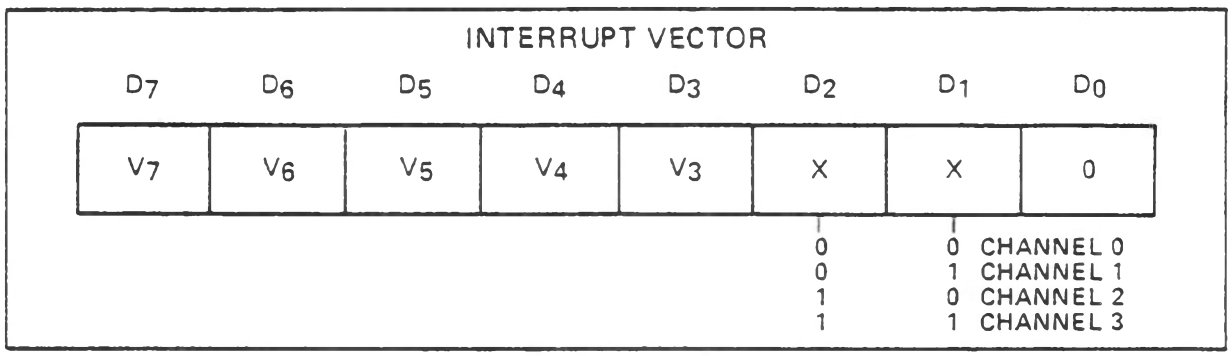
The Down Counter is an 8-bit register, used in both Counter Mode and Timer Mode, loaded initially, and later when it counts down to zero, by the Time Constant Register. The Down Counter is decremented by each external clock edge in the Counter Mode, or in the Timer Mode, by the clock output of the Prescaler. At any time, by performing a simple I/O Read at the port address assigned to the selected CTC channel, the CPU can access the contents of this register and obtain the number of counts-to-zero. Any CTC channel may be programmed to generate an interrupt request sequence each time the zero count is reached.

In channels 0, 1, and 2, when the zero count condition is reached, a signal pulse appears at the corresponding ZC/TO pin. Due to package pin limitations, however, channel 3 does not have this pin and so may be used only in applications where this output pulse is not required.

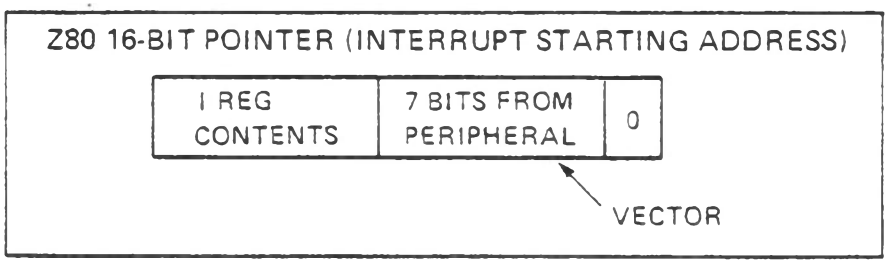
2.3 INTERRUPT CONTROL LOGIC

The Interrupt Control Logic insures that the CTC acts in accordance with Z80 system interrupt protocol for nested priority interrupting and return from interrupt. The priority of any system device is determined by its physical location in a daisy chain configuration. Two signal lines (IEI and IEO) are provided in CTC devices to form this system daisy chain. The device closest to the CPU has the highest priority; within the CTC, interrupt priority is predetermined by channel number, with channel 0 having highest priority down to channel 3 which has the lowest priority. The purpose of a CTC-generated interrupt, as with any other peripheral device, is to force the CPU to execute an interrupt service routine. According to Z80 system interrupt protocol, lower priority devices or channels may not interrupt higher priority devices or channels that have already interrupted and have not had their interrupt service routines completed. However, high priority devices or channels may interrupt the servicing of lower priority devices or channels.

A CTC channel may be programmed to request an interrupt every time its Down Counter reaches a count of zero. (To utilize this feature requires that the CPU be programmed for interrupt mode 2.) Some time after the interrupt request, the CPU will send out an interrupt acknowledge, and the CTC's Interrupt Control Logic will determine the highest-priority channel which is requesting an interrupt within the CTC device. Then if the CTC's IEI input is active, indicating that it has priority within the system daisy chain, it will place an 8-bit Interrupt Vector on the system data bus. The high-order 5 bits of this vector will have been written to the CTC earlier as part of the CTC initial programming process; the next two bits will be provided by the CTC's Interrupt Control Logic as a binary code corresponding to the highest-priority channel requesting an interrupt; finally the low-order bit of the vector will always be zero according to a convention described below.



This interrupt vector is used to form a pointer to a location in memory where the address of the interrupt service routine is stored in a table. The vector represents the least significant 8 bits, while the CPU reads the contents of the I register to provide the most significant 8-bits of the 16-bit pointer. The address in memory pointed to will contain the low-order byte, and the next highest address will contain the high-order byte of an address which in turn contains the first opcode of the interrupt service routine. Thus in mode 2, a single 8-bit vector stored in an interrupting CTC can result in an indirect call to any memory location.



There is a Z80 system convention that all addresses in the interrupt service routine table should have their low-order byte in an even location in memory, and their high-order byte in the next highest location in memory, which will always be odd so that the least significant bit of any interrupt vector will always be even. Hence the least significant bit of any interrupt vector will always be zero.

The RETI instruction is used at the end of any interrupt service routine to initialize the daisy chain enable line IEO for proper control of nested priority interrupt handing. The CTC monitors the system data bus and decodes this instruction when it occurs. Thus the CTC channel control logic will know when the CPU has completed servicing an interrupt, without any further communication with the CPU being necessary.

3.0 CTC PIN DESCRIPTION

A diagram of the Z80-CTC pin configuration is shown in figure 3.0-1. This section describes the function of each pin.

D7 – D0

Z80-CPU Data Bus (bi-directional, tri-state)

This bus is used to transfer all data and command words between the Z80-CPU and the Z80-CTC. There are 8 bits on this bus, of which D0 is the least significant.

CS1 – CS0

Channel Select (input, active high)

These pins form a 2-bit binary address code for selecting one of the four independent CTC channels for an I/O Write or Read. (See truth table below.)

	CS1	CS0
Ch 0	0	0
Ch 1	0	1
Ch 2	1	0
Ch 3	1	1

$\overline{CE}$

Chip Enable (input, active low)

A low level on this pin enables the CTC to accept control words, Interrupt Vectors, or time constant data words from the Z80 Data Bus during an I/O Write cycle, or to transmit the contents of the Down Counter to the CPU during an I/O Read cycle. In most applications this signal is decoded from the 8 least significant bits of the address bus for any of the four I/O port addresses that are mapped to the four Counter/Timer Channels.

Clock (Φ)

System Clock (input)

This single-phase clock is used by the CTC to synchronize certain signals internally.

$\overline{M1}$

Machine Cycle One Signal from CPU (input, active low)

When $\overline{M1}$ is active and the $\overline{RD}$ signal is active, the CPU is fetching an instruction from memory. When $\overline{M1}$ is active and the $\overline{IORQ}$ signal is active, the CPU is acknowledging an interrupt, alerting the CTC to place an Interrupt Vector on the Z80 Data Bus if it has daisy chain priority and one of its channels has requested an interrupt

$\overline{IORQ}$

Input/Output Request from CPU (input, active low)

The $\overline{IORQ}$ signal is used in conjunction with the $\overline{CE}$ and $\overline{RD}$ signals to transfer data and Channel Control Words between the Z80-CPU and the CTC. During a CTC Write Cycle, $\overline{IORQ}$ and $\overline{CE}$ must be true and $\overline{RD}$ false. The CTC does not receive a specific write signal, instead generating its own internally from the inverse of a valid $\overline{RD}$ signal. In a CTC Read Cycle, $\overline{IORQ}$, $\overline{CE}$ and $\overline{RD}$ must be active to place the contents of the Down Counter on the Z80 Data Bus. If $\overline{IORQ}$ and $\overline{M1}$ are both true, the CPU is acknowledging an interrupt request, and the highest-priority interrupting channel will place its Interrupt Vector on the Z80 Data Bus.

3.0 CTC PIN DESCRIPTION (CONT'D)

$\overline{RD}$

Read Cycle Status from the CPU (input, active low)

The $\overline{RD}$ signal is used in conjunction with the $\overline{IORQ}$ and $\overline{CE}$ signals to transfer data and Channel Control Words between the Z80-CPU and the CTC. During a CTC Write Cycle, $\overline{IORQ}$ and $\overline{CE}$ must be true and $\overline{RD}$ false. The CTC does not receive a specific write signal, instead generating its own internally from the inverse of a valid $\overline{RD}$ signal. In a CTC Read Cycle, $\overline{IORQ}$, $\overline{CE}$ and $\overline{RD}$ must be active to place the contents of the Down Counter on the Z80 Data Bus.

IEI

Interrupt Enable In (input, active high)

This signal is used to help form a system-wide interrupt daisy chain which establishes priorities when more than one peripheral device in the system has interrupting capability. A high level on this pin indicates that no other interrupting devices of higher priority in the daisy chain are being serviced by the Z80-CPU.

IEO

Interrupt Enable Out (output, active high)

The IEO signal, in conjunction with IEI, is used to form a system-wide interrupt priority daisy chain. IEO is high only if IEI is high and the CPU is not servicing an interrupt from any CTC channel. Thus this signal blocks lower priority devices from interrupting while a higher priority interrupting device is being serviced by the CPU.

$\overline{INT}$

Interrupt Request (output, open drain, active low)

This signal goes true when any CTC channel which has been programmed to enable interrupts has a zero-count condition in its Down Counter.

$\overline{RESET}$

Reset (input, active low)

This signal stops all channels from counting and resets channel interrupt enable bits in all control registers, thereby disabling CTC-generated interrupts. The ZC/TO and $\overline{INT}$ outputs go to their inactive states, IEO reflects IEI, and the CTC's data bus output drivers go to the high impedance state.

CLK/TRG3 – CLK/TRG0

External Clock/Timer Trigger (input, user-selectable active high or low)

There are four CLK/TRG pins, corresponding to the four independent CTC channels. In the Counter Mode, every active edge on this pin decrements the Down Counter. In the Timer Mode, an active edge on this pin initiates the timing function. The user may select the active edge to be either rising or falling.

ZC/TO2 – AC/TO0

Zero Count/Timeout (output, active high)

There are three ZC/TO pins, corresponding to CTC channels 2 through 0. (Due to package pin limitations channel 3 has no ZC/TO pin.) In either Counter Mode or Timer Mode, when the Down Counter decrements to zero an active high going pulse appears at this pin.

3.0 CTC PIN DESCRIPTION

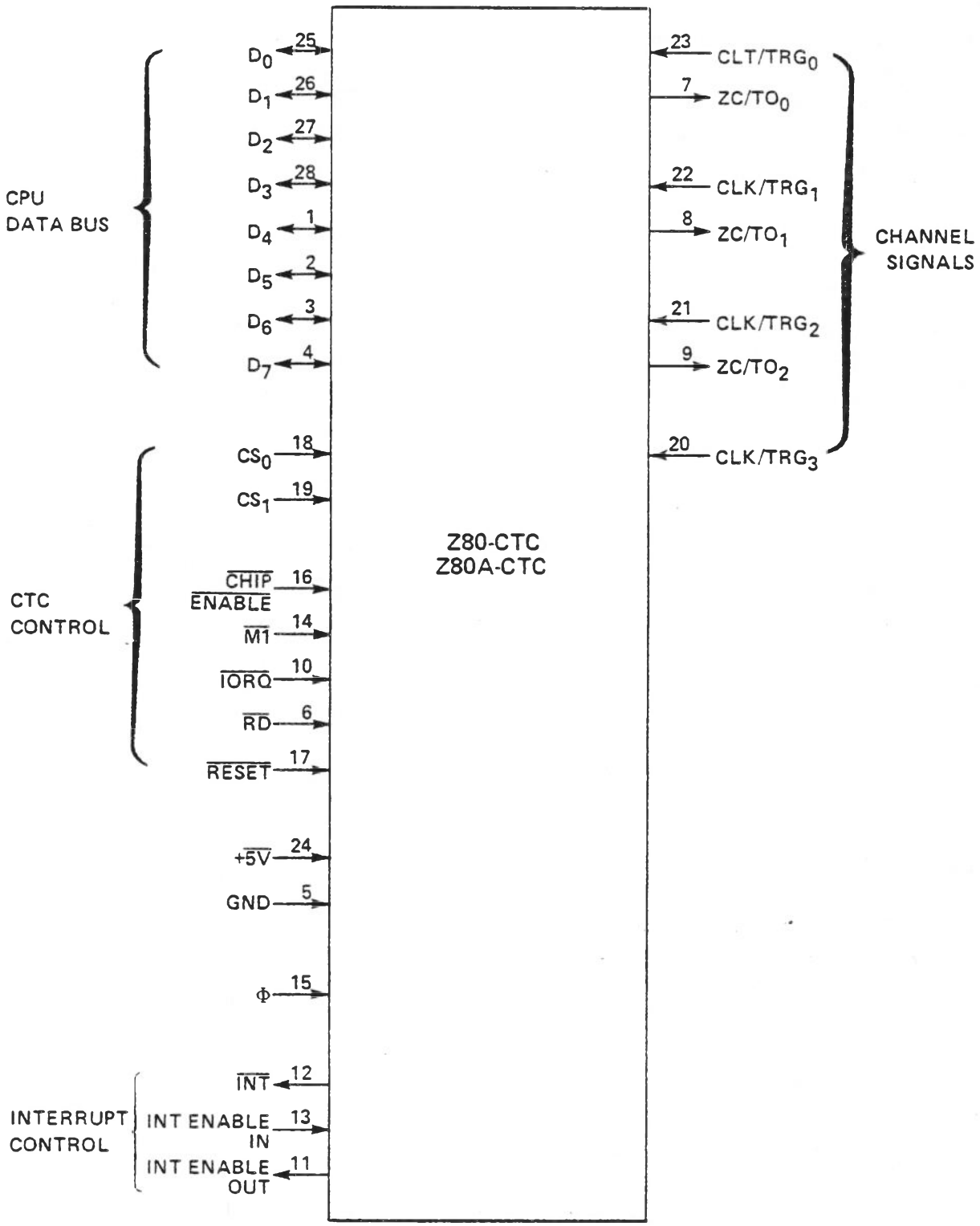


FIGURE 3.0-1
CTC PIN CONFIGURATION

4.0 CTC OPERATING MODES

At power-on, the Z80-CTC state is undefined. Asserting $\overline{\text{RESET}}$ puts the CTC in a known state. Before any channel can begin counting or timing, a Channel Control Word and a time constant data word must be written to the appropriate registers of that channel. Further, if any channel has been programmed to enable interrupts, an Interrupt Vector word must be written to the CTC's Interrupt Control Logic. (For further details, refer to section 5.0: "CTC Programming.") When the CPU has written all of these words to the CTC all active channels will be programmed for immediate operation in either the Counter Mode or the Timer Mode.

4.1 CTC COUNTER MODE

In this mode the CTC counts edges of the CLK/TRG input. The Counter Mode is programmed for a channel when its Channel Control Word is written with bit 6 set. The Channel's External Clock (CLK/TRG) input is monitored for a series of triggering edges; after each, in synchronization with the next rising edge of Φ (the System Clock), the Down Counter (which was initialized with the time constant data word at the start of any sequence of down-counting) is decremented. Although there is no set-up time requirement between the triggering edge of the External Clock and the rising edge of Φ, (Clock), the Down Counter will not be decremented until the following Φ pulse. (See the parameter ts(CK) in section 8.3: "A.C. Characteristics.") A channels's External Clock input is pre-programmed by bit 4 of the Channel Control Word to trigger the decrementing sequence with either a high or a low going edge.

In any of Channels 0, 1, or 2, when the Down Counter is successively decremented from the original time constant until finally it reaches zero, the Zero Count (ZC/TO) output pin for that channel will be pulsed active (high). (However, due to package pin limitations, channel 3 does not have this pin and so may only be used in applications where this output pulse is not required.) Further, if the channel has been so pre-programmed by bit 7 of the Channel Control Word, an interrupt request sequence will be generated. (For more details, see section 7.0: "CTC Interrupt Servicing.")

As the above sequence is proceeding, the zero count condition also results in the automatic reload of the Down Counter with the original time constant data word in the Time Constant Register. There is no interruption in the sequence of continued down-counting. If the Time Constant Register is written to with a new time constant data word while the Down Counter is decrementing, the present count will be completed before the new time constant will be loaded into the Down Counter.

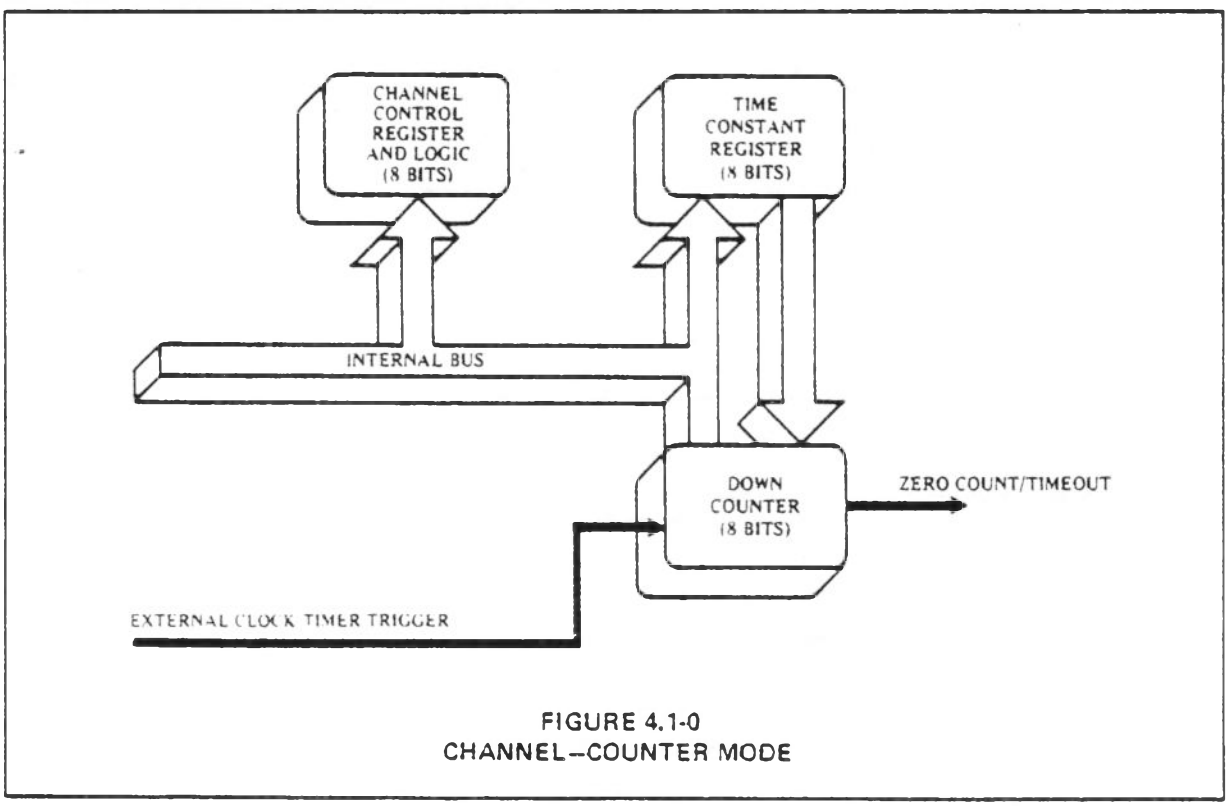


FIGURE 4.1-0
CHANNEL–COUNTER MODE

4.2 CTC TIMER MODE

In this mode the CTC generates timing intervals that are an integer value of the system cl-iod. The Timer Mode is programmed for a channel when its Channel Control Word is written with bit 6 the chan-nel then may be used to measure intervals of time based on the System Clock period. The Syste-k is fed through two successive counters, the Prescaler and the Down Counter. Depending on the pre-pr-med bit 5 in the Channel Control Word, the Prescaler divides the System Clock by a factor of either 16 or The out-put of the Prescaler is then used as a clock to decrement the Down Counter, which may be pre-mmed with any time constant integer between 1 and 256. As in the Counter Mode, the time constant i-matically reloaded into the Down Counter at each zero-count condition, and counting continues. Also at ount, the channel's Time Out (ZC/TO) output (which is the output of the Down Counter) is pulsed, resul- a uni-form pulse train of precise period given by the product.

$$t_c * P * TC$$

where t_c is the System Clock period, P is the Prescaler factor of 16 or 256 and TC is the pre-pro-ned time constant.

Bit 3 of the Channel Control Word is pre-programmed to select whether timing will be autc-cally initiated, or whether it will be initiated with a triggering edge at the channel's Timer Trigger (CL-G) input. If bit 3 is reset the timer automatically begins operation at the start of the CPU cycle following tlO Write machine cycle that loads the time constant data word to the channel. If bit 3 is set the timer begi-peration on the second succeeding rising edge of Φ after the Timer Trigger edge following the loading of tme constant data word. If no time constant data word is to follow then the timer begins operation on second succeeding rising edge of Φ after the Timer Trigger edge following the control word write cycle. B of the Channel Control Word is pre-programmed to select whether the Timer Trigger will be sensitive to -ing or falling edge. Although there is no set-up requirement between the active edge of the Timer Triggerd the nex rising edge of Φ. If the Timer Trigger edge occurs closer than a specified minimum set-up time to trising ed- of Φ, the Down Counter will not begin decrementing until the following rising edge of Φ. (See the-rameter ts(TR) in section 8.3: "A.C. Characteristics".)

If bit 7 in the Channel Control Word is set, the zero-count condition in the Down Counter, b-des causing a pulse at the channel's Time Out pin, will be used to initiate an interrupt request sequence. For more details, see section 7.0: "CTC Interrupt Servicing.".)

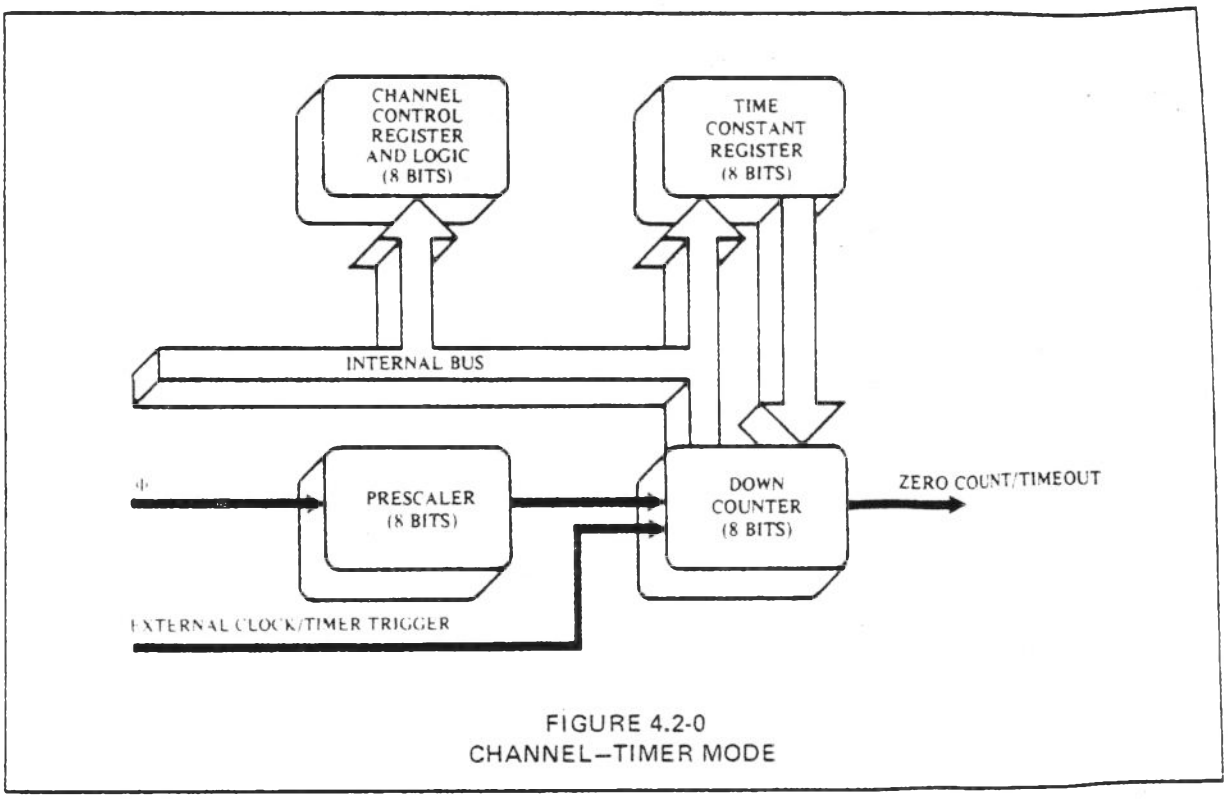


FIGURE 4.2-0
CHANNEL–TIMER MODE

5.0 CTC PROGRAMMING

Before a Z80-CTC channel can begin counting or timing operations, a Channel Control Word and a Time Constant data word must be written to it by the CPU. These words will be stored in the Channel Control Register and the Time Constant Register of that channel. In addition, if any of the four channels have been programmed with bit 7 of their Channel Control Words to enable interrupts, an Interrupt Vector must be written to the appropriate register in the CTC. Due to automatic features in the Interrupt Control Logic, one pre-programmed Interrupt Vector suffices for all four channels.

5.1 LOADING THE CHANNEL CONTROL REGISTER

To load a Channel Control Word, the CPU performs a normal I/O Write sequence to the port address corresponding to the desired CTC channel. Two CTC input pins, namely CS0 and CS1, are used to form a 2-bit binary address to select one of four channels within the device. (For a truth table, see section 2.2.1: "The Channel Control Register and Logic".) In many system architectures, these two input pins are connected to Address Bus lines A0 and A1, respectively, so that the four channels in a CTC device will occupy contiguous I/O port addresses. A word written to a CTC channel will be interpreted as a Channel Control Word, and loaded into the Channel Control Register, its bit 0 is a logic 1. The other seven bits of this word select operating modes and conditions as indicated in the diagram below. Following the diagram the meaning of each bit will be discussed in detail.

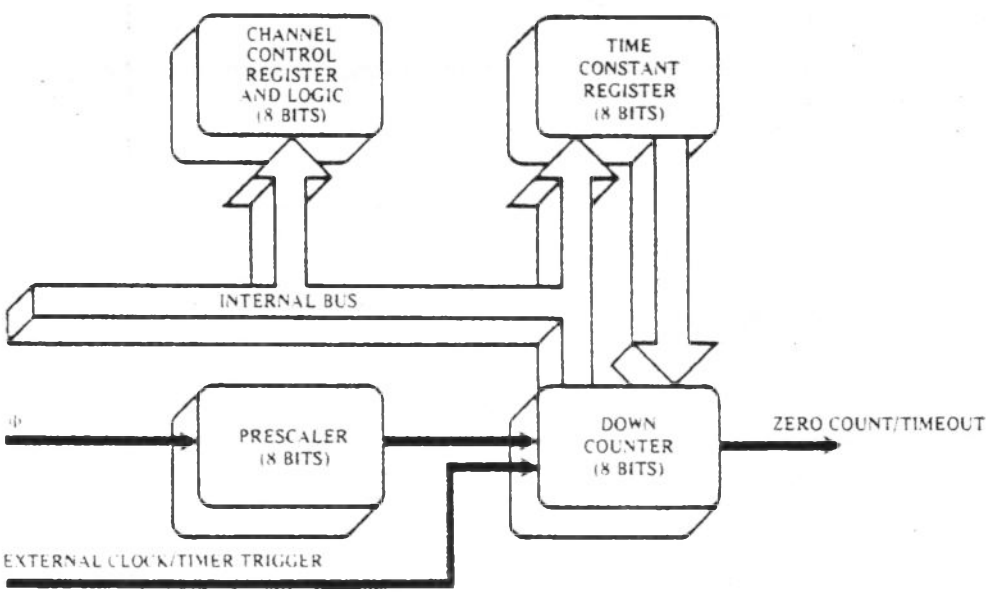
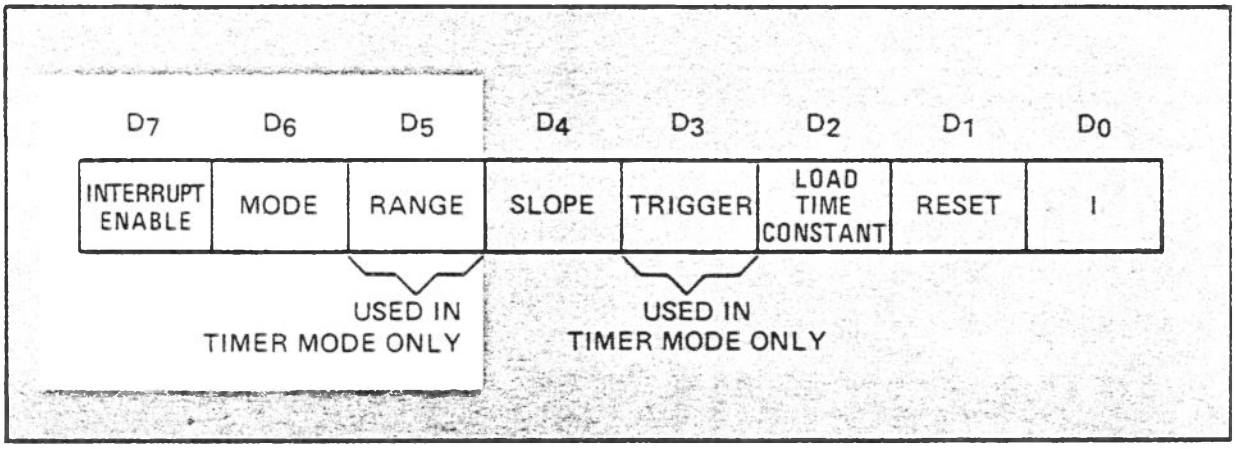


FIGURE 5.1-0
CHANNEL BLOCK DIAGRAM

5.1 LOADING THE CHANNEL CONTROL REGISTER (CONT'D)



Bit 7 = 1

The channel is enabled to generate an interrupt request sequence every time the Down Counter reaches a zero-count condition. To set this bit to 1 in any of the four Channel Control Registers necessitates that an Interrupt Vector also be written to the CTC before operation begins. Channel interrupts may be programmed in either Counter Mode or Timer Mode. If an updated Channel Control Word is written to a channel already in operation, with bit 7 set, the interrupt enable selection will not be retroactive to a preceding zero-count condition.

Bit 7 = 0

Channel interrupts disabled.

Bit 6 = 1

Counter Mode selected. The Down Counter is decremented by each triggering edge of the External Clock (CLK/TRG) input. The Prescaler is not used.

Bit 6 = 0

Timer Mode selected. The Prescaler is clocked by the System Clock Φ, and the output of the Prescaler in turn clocks the Down Counter. The output of the Down Counter (the channel's ZC/TO output) is a uniform pulse train of period given by the product

$$t_c * P * TC$$

where t_c is the period of System Clock Φ, P is the Prescaler factor of 16 or 256, and TC is the time constant data word.

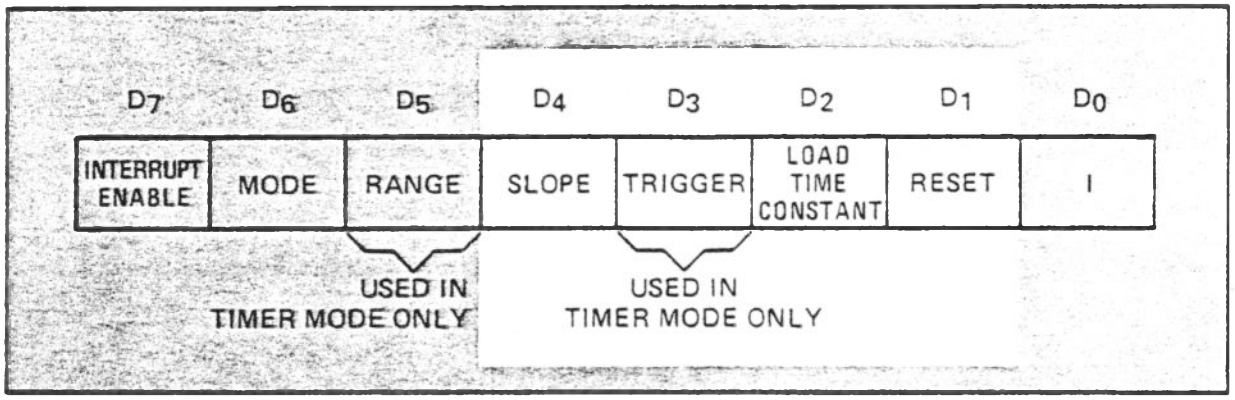
Bit 5 = 1

(Defined for Timer Mode only.) Prescaler factor is 256.

Bit 5 = 0

(Defined for Timer Mode only.) Prescaler factor is 16.

5.1 LOADING THE CHANNEL CONTROL REGISTER (CONT'D)



Bit 4 = 1

TIMER MODE – positive edge trigger starts timer operation.

COUNTER MODE – positive edge decrements the down counter.

Bit 4 = 0

TIMER MODE – negative edge trigger starts timer operation.

COUNTER MODE – negative edge decrements the down counter.

Bit 3 = 1

Timer Mode Only – External trigger is valid for starting timer operation after rising edge of T_2 of the machine cycle following the one that loads the time constant. The Prescaler is decremented 2 clock cycles later if the setup time is met, otherwise 3 clock cycles.

Bit 3 = 0

Timer Mode Only – Timer begins operation on the rising edge of T_2 of the machine cycle following the one that loads the time constant.

Bit 2 = 1

The time constant data word for the Time Constant Register will be the next word written to this channel. If an updated Channel Control Word and time constant data word are written to a channel while it is already in operation, the Down Counter will continue decrementing to zero before the new time constant is loaded into it.

Bit 2 = 0

No time constant data word for the Time Constant Register should be expected to follow. To program bit 2 to this state implies that this Channel Control Word is intended to update the status of a channel already in operation, since a channel will not operate without a correctly programmed data word in the Time Constant Register, and a set bit 2 in this Channel Control Word provides the only way of writing to the Time Constant Register.

Bit 1 = 1

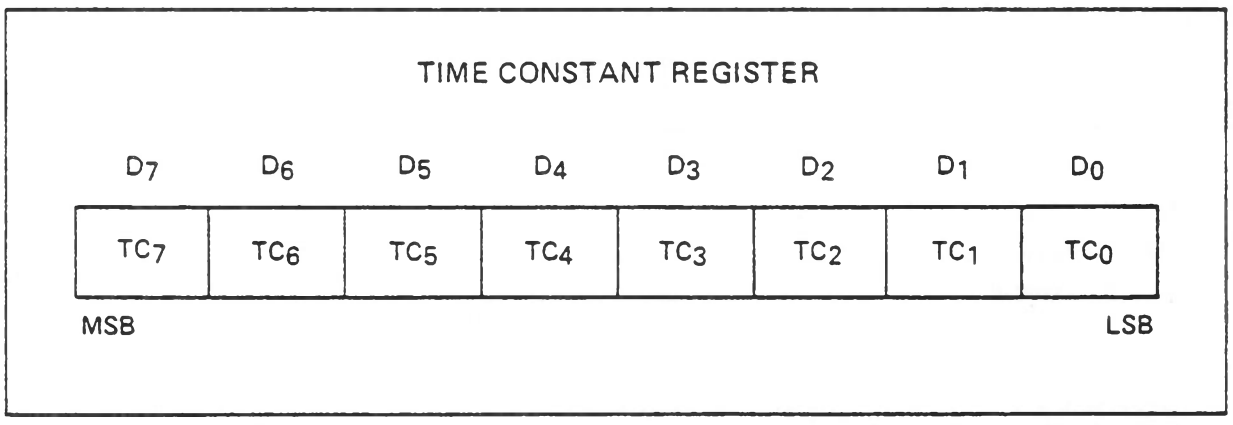
Reset channel. Channel stops counting or timing. This is not a stored condition. Upon writing into this bit a reset pulse discontinues current channel operation, however, none of the bits in the channel control register are changed. If both bit 2 = 1 and bit 1 = 1 the channel will resume operation upon loading a time constant.

Bit 1 = 0

Channel continues current operation.

5.2 LOADING THE TIME CONSTANT REGISTER

A channel may not begin operation in either Timer Mode or Counter Mode unless a time constant data word is written into the Time Constant Register by the CPU. This data word will be expected on the next I/O Write to this channel following the I/O Write of the Channel Control Word, provided that bit 2 of the Channel Control Word is set. The time constant data word may be any integer value in the range 1-256. If all eight bits in this word are zero, it is interpreted as 256. If a time constant data word is loaded to a channel already in operation, the Down Counter will continue decrementing to zero before the new time constant is loaded from the Time Constant Register to the Down Counter.



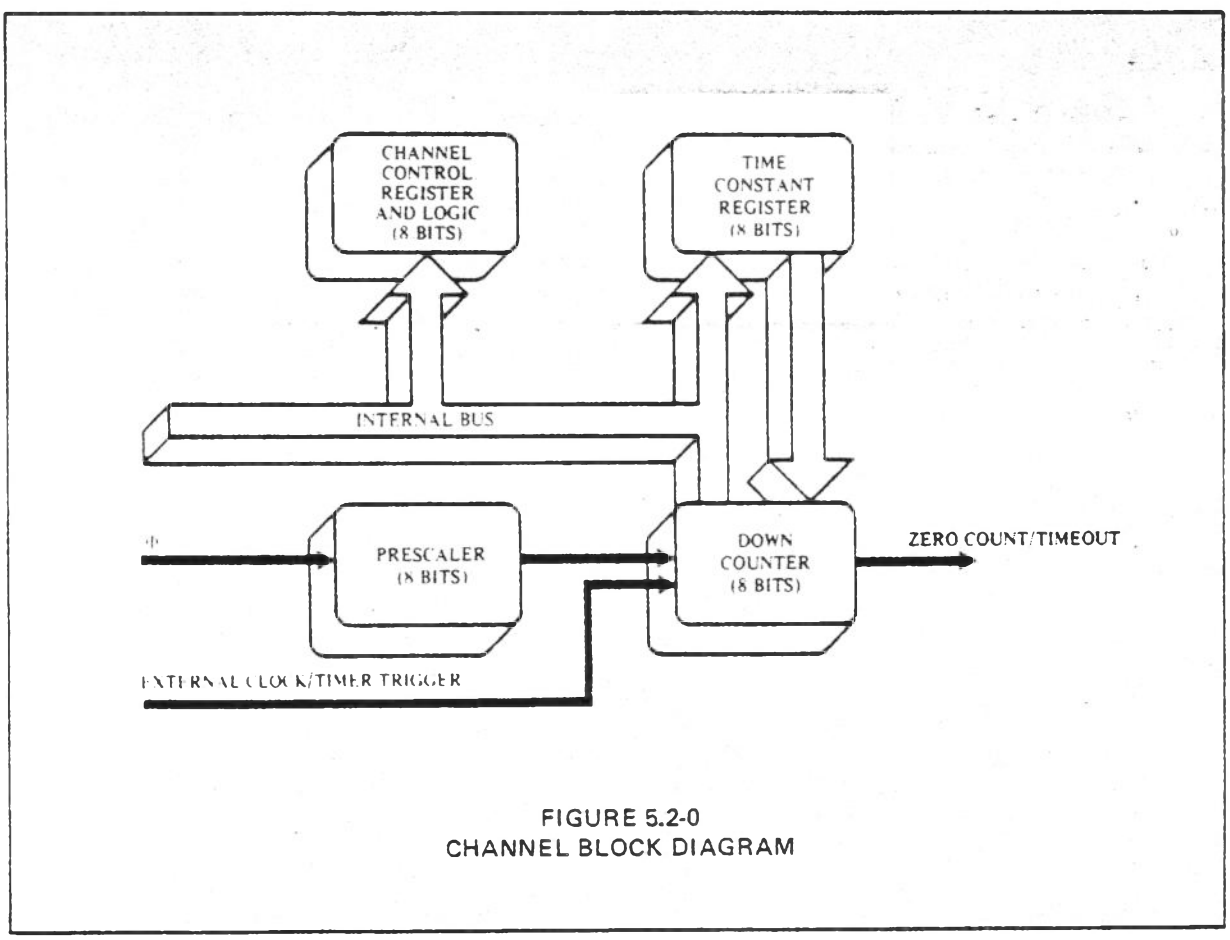
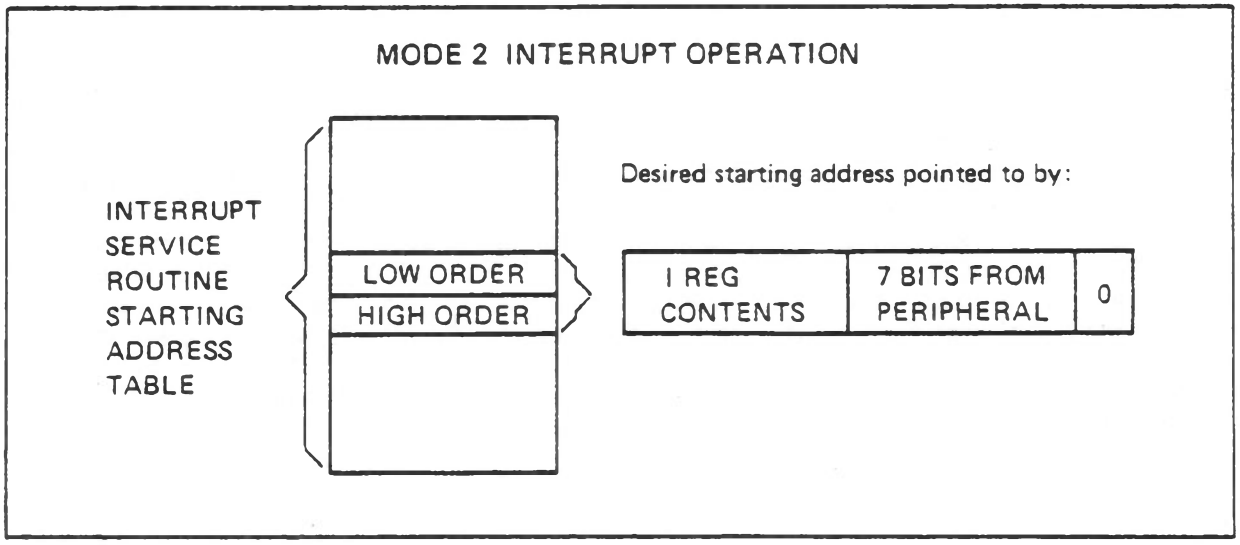


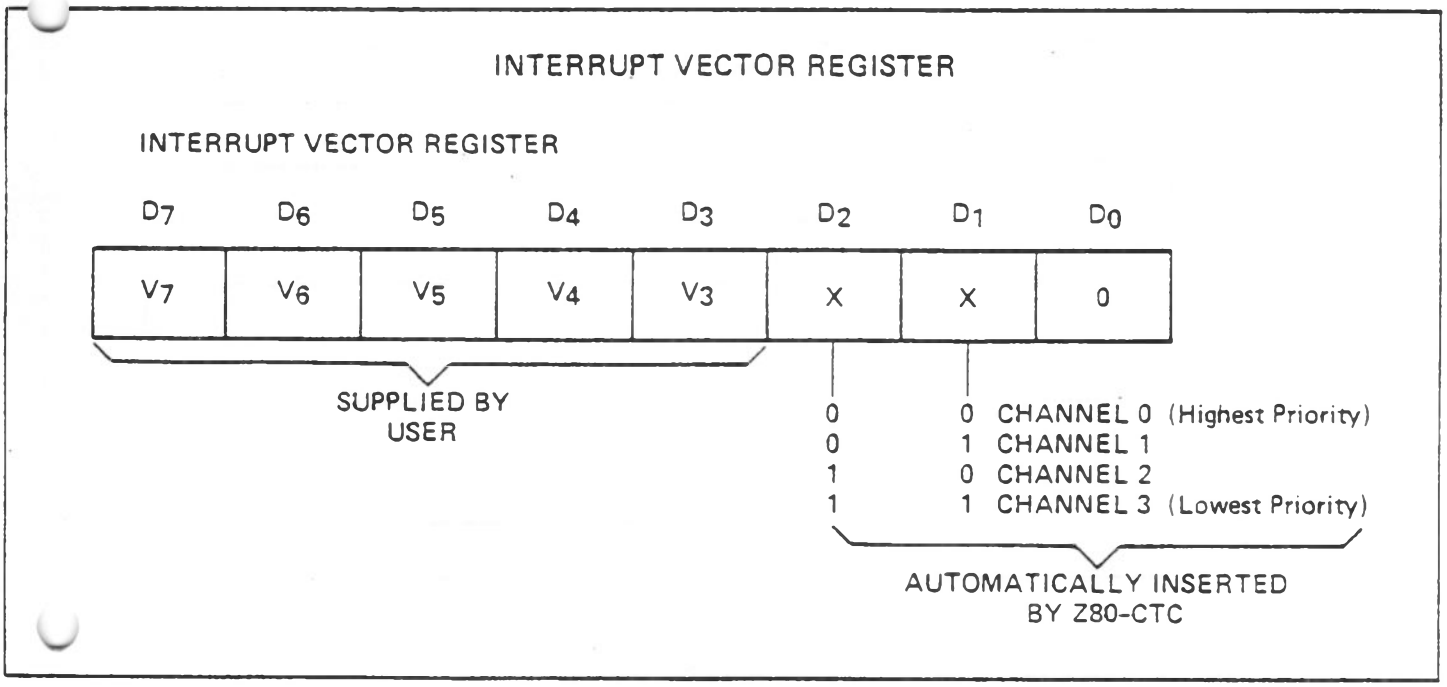
FIGURE 5.2-0
CHANNEL BLOCK DIAGRAM

5.3 LOADING THE INTERRUPT VECTOR REGISTER

The Z80-CTC has been designed to operate with the Z80-CPU programmed for mode 2 interrupt response. Under the requirements of this mode, when a CTC channel requests an interrupt and is acknowledged, a 16-bit pointer must be formed to obtain a corresponding interrupt service routine starting address from a table in memory. The upper 8 bits of this pointer are provided by the CPU's I register, and the lower 8 bits of the pointer are provided by the CTC in the form of an Interrupt Vector unique to the particular channel that requested the interrupt. (For further details, see section 7.0: "CTC Interrupt Servicing".)



The high order 5 bits of this Interrupt Vector must be written to the CTC in advance as part of the initial programming sequence. To do so, the CPU must write to the I/O port address corresponding to the CTC channel 0, just as it would if a Channel Control Word were being written to that channel, except that bit 0 of the word being written must contain a 0. (As explained above in section 5.1, if bit 0 of a word written to a channel were set to 1, the word would be interpreted as a Channel Control Word, so a 0 in bit 0 signals the CTC to load the incoming word into the Interrupt Vector Register.) Bits 1 and 2, however, are not used when loading this vector. At the time when the interrupting channel must place the Interrupt Vector on the Z80 Data Bus, the Interrupt Control Logic of the CTC automatically supplies a binary code in bits 1 and 2 identifying which of the four CTC channels is to be serviced.



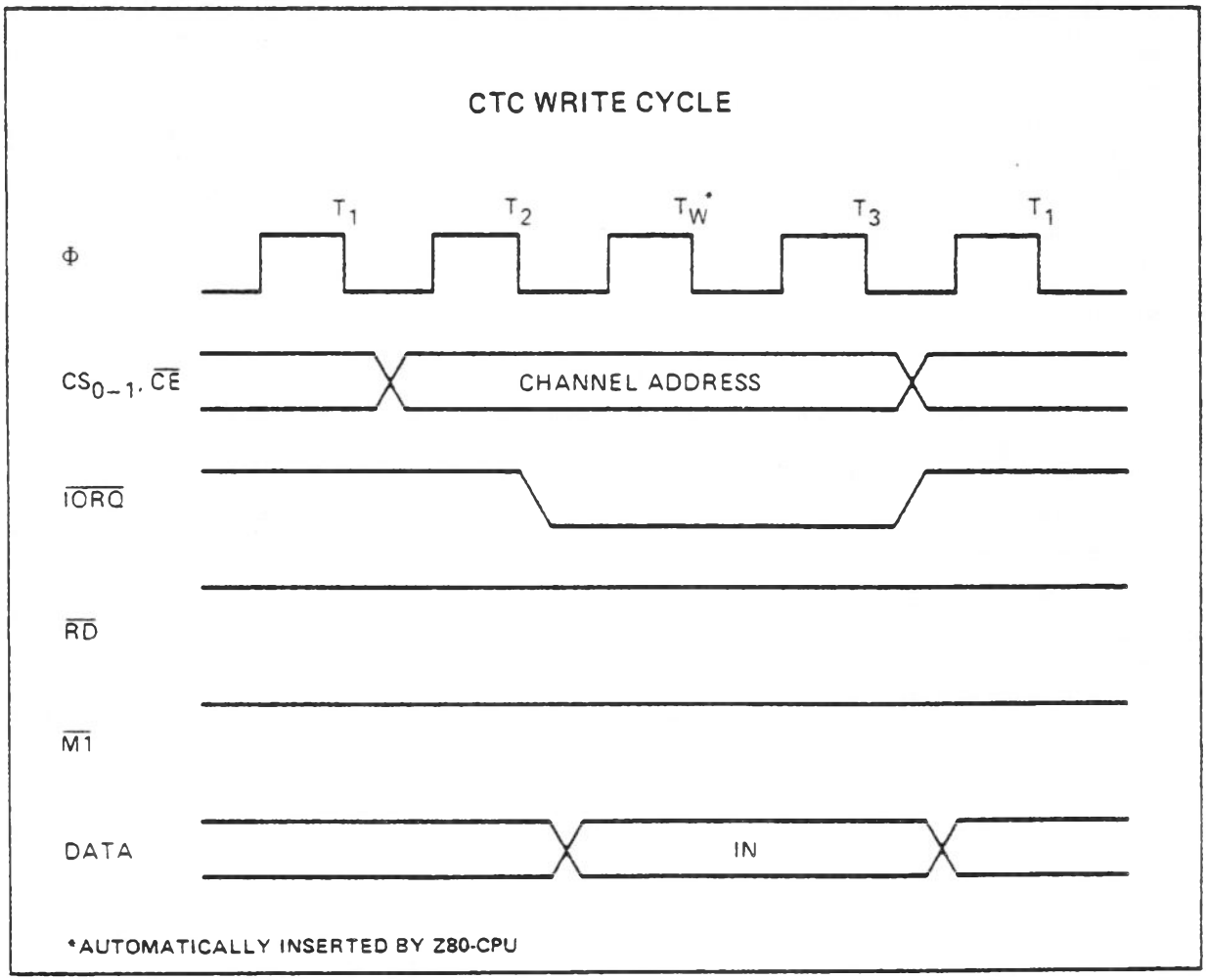
6.0 CTC TIMING

This section illustrates the timing relationships of the relevant CTC pins for the following types of operation: writing a word to the CTC, reading a word from the CTC, counting, and timing. Elsewhere in this manual may be found timing diagrams relating to interrupt servicing (section 7.0) and an A.C. Timing Diagram which quantitatively specifies the timing relationships (section 8.4).

6.1 CTC WRITE CYCLE

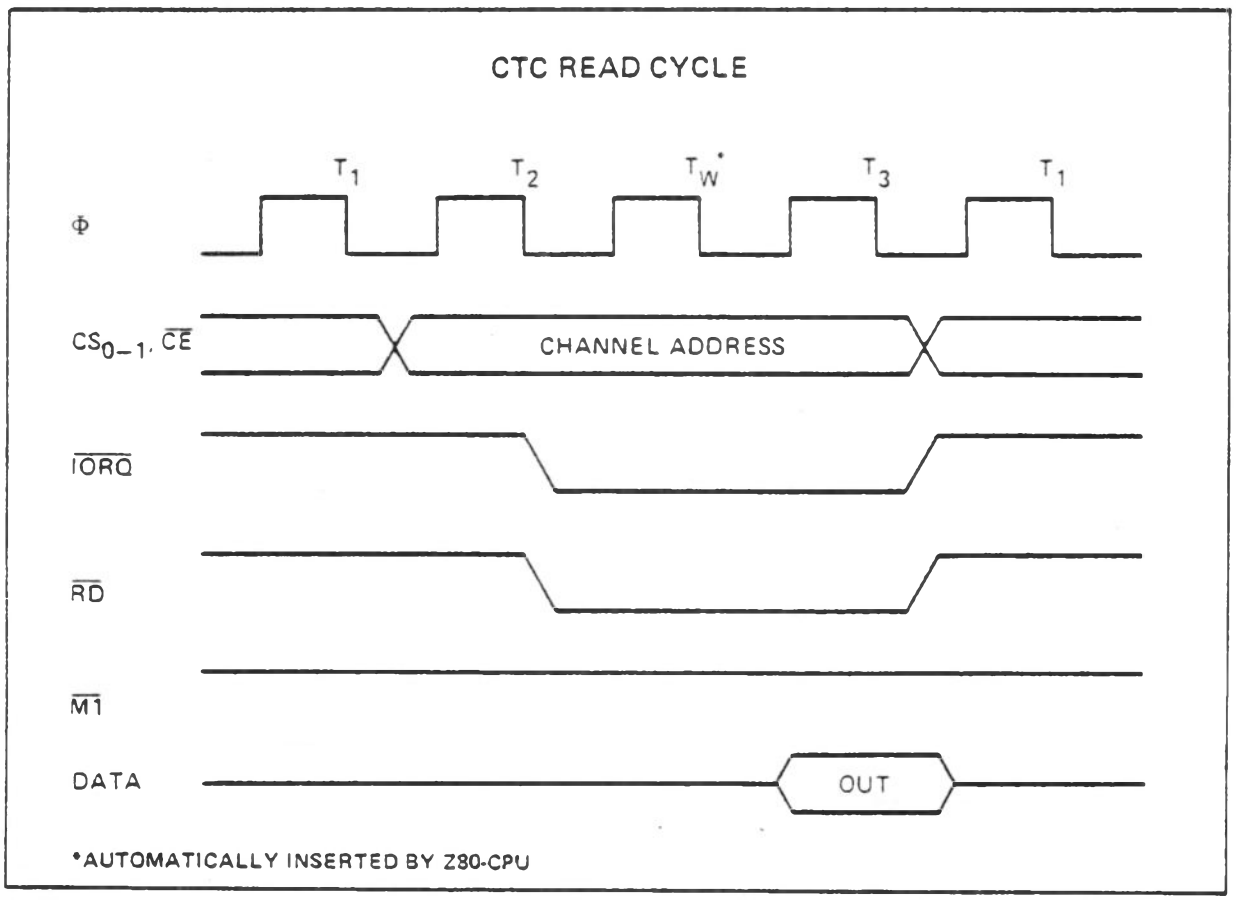
Figure 6.0-1 illustrates the timing associated with the CTC Write Cycle. This sequence is applicable to loading either a Channel Control Word, an Interrupt Vector, or a time constant data word.

In the sequence shown, during clock cycle T_1, the Z80-CPU prepares for the Write Cycle with a false (high) signal at CTC input pin $\overline{RD}$ (Read). Since the CTC has no separate Write signal input, it generates its own internally from the false $\overline{RD}$ input. Later, during clock cycle T_2, the Z80-CPU initiates the Write Cycle with true (low) signals at CTC input pins $\overline{IORQ}$ (I/O Request) and $\overline{CE}$ (Chip Enable). (Note: $\overline{M1}$ must be false to distinguish the cycle from an interrupt acknowledge.) Also at this time a 2-bit binary code appears at CTC inputs CS1 and CS0 (Channel Select 1 and 0), specifying which of the four CTC channels is being written to, and the word being written appears on the Z80 Data Bus. Now everything is ready for the word to be latched into the appropriate CTC internal register in synchronization with the rising edge beginning clock cycle T_3. No additional wait states are allowed.



6.2 CTC READ CYCLE

Figure 6.0-2 illustrates the timing associated with the CTC Read Cycle. This sequence is used any time the CPU reads the current contents of the Down Counter. During clock cycle T_2, the Z80-CPU initiates the Read Cycle with true signals at input pins RD (Read), IORQ (I/O Request), and CE (Chip Enable). Also at this time a 2-bit binary code appears at CTC inputs CS1 and CS0 (Channel Select 1 and 0), specifying which of the four CTC channels is being read from. (Note: M1 must be false to distinguish the cycle from an interrupt acknowledge.) On the rising edge of the cycle T_3 the valid contents of the Down Counter as of the rising edge of cycle T_2 will be available on the Z80 Data Bus. No additional wait states are allowed.

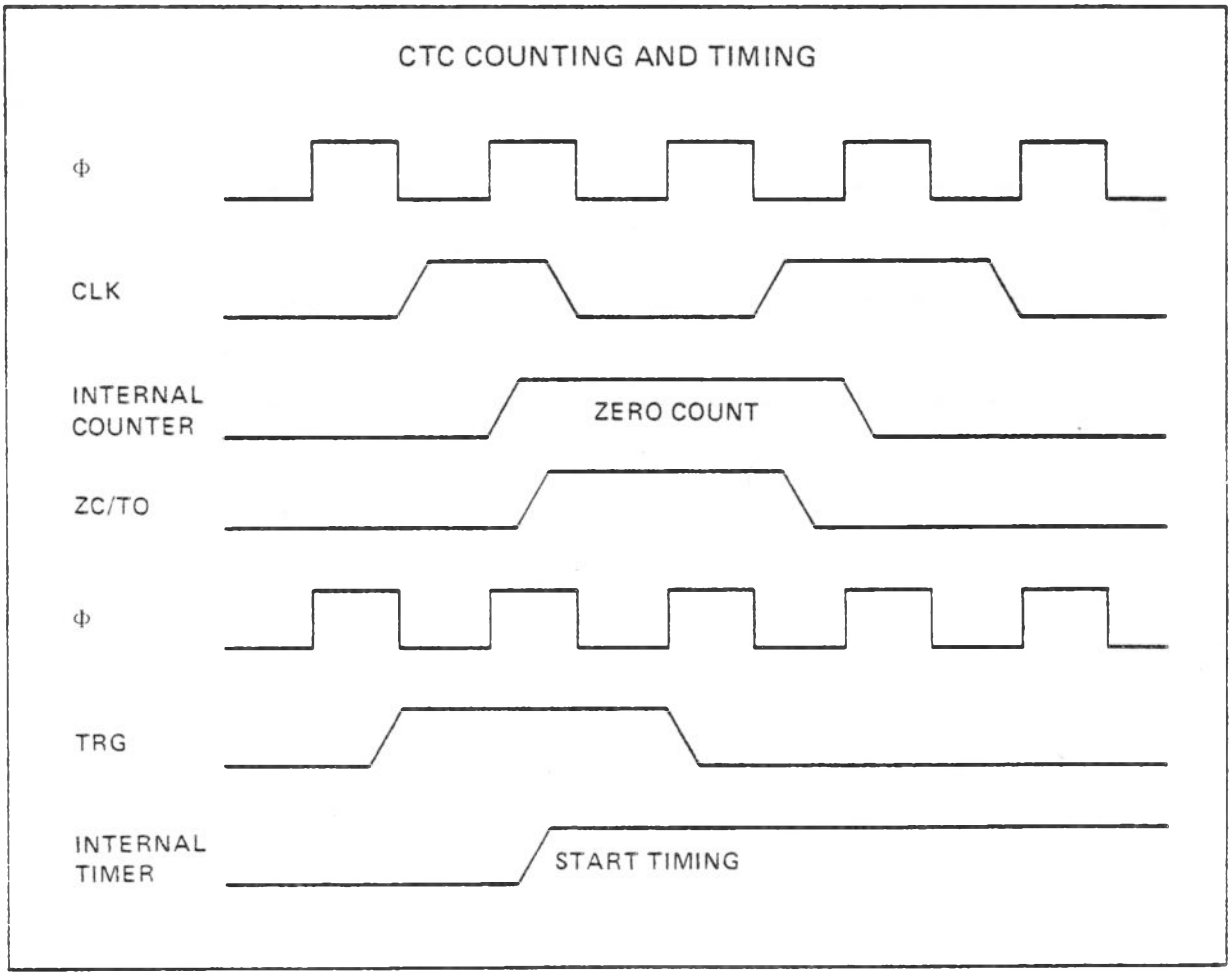


6.3 CTC COUNTING AND TIMING

Figure 6.0-3 illustrates the timing diagram for the CTC Counting and Timing Modes.

In the Counter Mode, the edge (rising edge is active in this example) from the external hardware connected to pin CLK/TRG decrements the Down Counter in synchronization with the System Clock Φ. As specified in the A.C. Characteristics (Section 9.1) this CLK/TRG pulse must have a minimum width and the minimum period must not be less than twice the system clock period. Although there is no set-up time requirement between the active edge of the CLK/TRG and the rising edge of Φ if the CLK/TRG edge occurs closer than a specified minimum time, the decrement of the Down Counter will be delayed one cycle of Φ. Immediately after the decrement of the Down Counter, 1 to 0, the ZC/TO output is pulsed true.

In the Timer Mode, a pulse trigger (user-selectable as either active high or active low) at the CLK/TRG pin enables timing function on the second succeeding rising edge of Φ. As in the Counter Mode, the triggering pulse is detected asynchronously and must have a minimum width. The timing function is initiated in syncronization with Φ, and a minimum set-up time is required between the active edge of the CLK/TRG and the next rising edge of Φ. If the CLK/TRG edge occurs closer than this, the initiation of the timer function will be delayed one cycle of Φ.



7.0 CTC INTERRUPT SERVICING

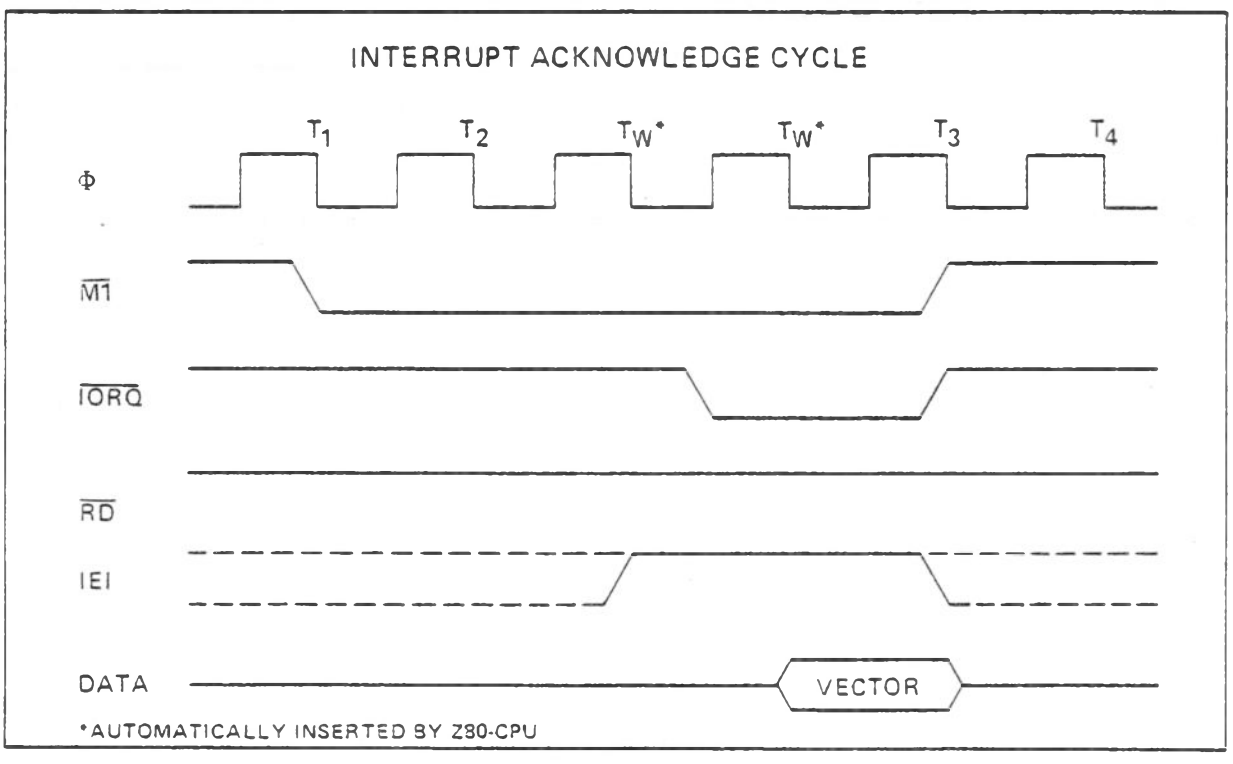
Each CTC channel may be individually programmed to request an interrupt every time its Down Counter reaches a count of zero. The purpose of a CTC-generated interrupt, as for any other peripheral device, is to force the CPU to execute an interrupt service routine. To utilize this feature the Z80-CPU must be programmed for mode 2 interrupt response. Under the requirements of this mode, when a CTC channel requests an interrupt and is acknowledged, a 16-bit pointer must be formed to obtain a corresponding interrupt service routine starting address from a table in memory. The lower 8 bits of the pointer are provided by the CTC in the form of an Interrupt Vector unique to the particular channel that requested the interrupt. (For further details, refer to chapter 8.0 of the Z80-CPU Technical Manual.)

The CTC's Interrupt Control Logic insures that it acts in accordance with Z80 system interrupt protocol for nested priority interrupt and proper return from interrupt. The priority of any system device is determined by its physical location in a daisy chain configuration. Two signal lines (IEI and IEO) are provided in the CTC and all Z80 peripheral devices to form the system daisy chain. The device closest to the CPU has the highest priority; within the CTC, interrupt priority is predetermined by channel number, with channel 0 having highest priority. According to Z80 system interrupt protocol, low priority devices or channels may not interrupt higher priority devices or channels that have already interrupted and not had their interrupt service routines completed. However, high priority devices or channels may interrupt the servicing of lower priority devices or channels. (For further details, see section 2.3: "Interrupt Control Logic".)

Sections 7.1 and 7.2 below describe the nominal timing relationships of the relevant CTC pins for the Interrupt Acknowledge Cycle and the Return from Interrupt Cycle. Section 7.3 below discusses a typical example of daisy chain interrupt servicing.

7.1 INTERRUPT ACKNOWLEDGE CYCLE

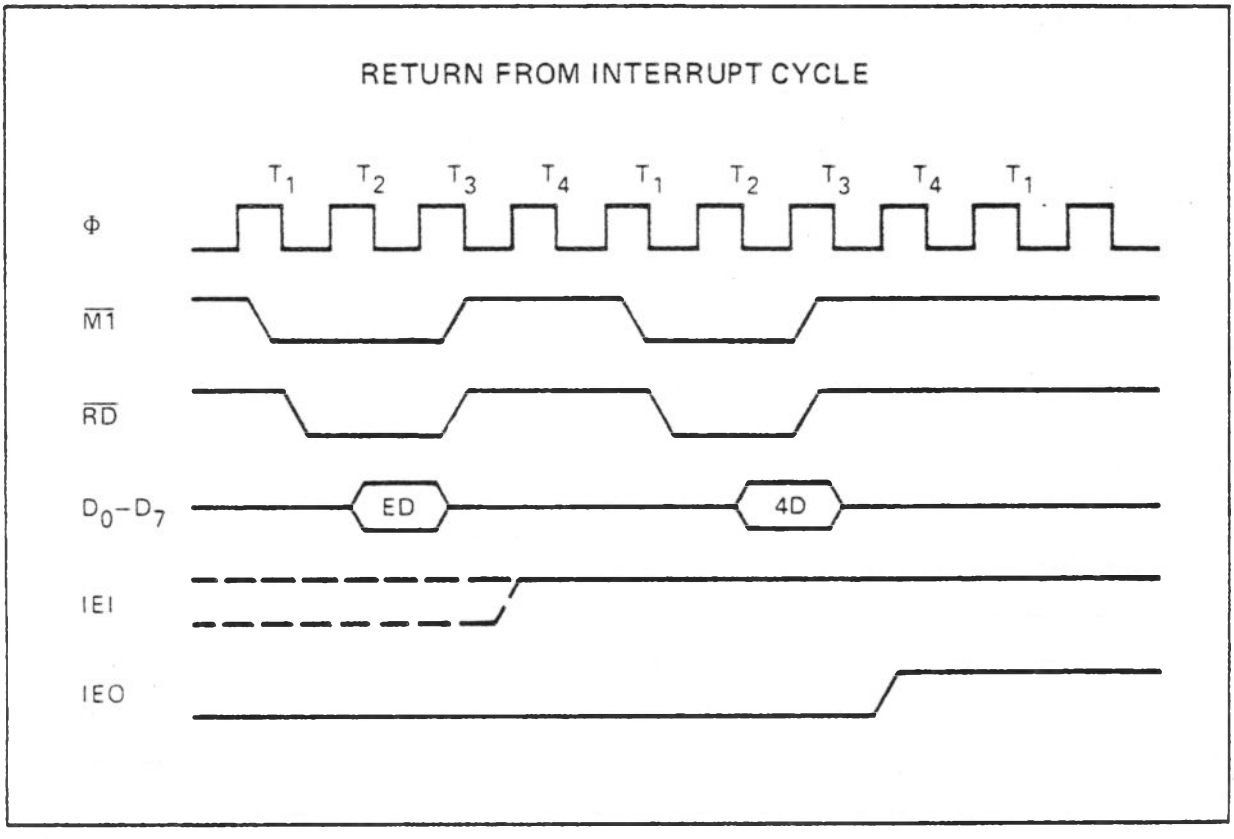
Figure 7.0-1 illustrates the timing associated with the Interrupt Acknowledge Cycle. Some time after an interrupt is requested by the CTC, the CPU will send out an interrupt acknowledge ($\overline{M1}$ and $\overline{IORQ}$). To insure that the daisy chain enable lines stabilize, channels are inhibited from changing their interrupt request status when $\overline{M1}$ is active. $\overline{M1}$ is active about two clock cycles earlier than $\overline{IORQ}$, and $\overline{RD}$ is false to distinguish the cycle from an instruction fetch. During this time the interrupt logic of the CTC will determine the highest priority channel requesting an interrupt. If the CTC Interrupt Enable Input (IEI) is active, then the highest priority interrupting channel within the CTC places its Interrupt Vector onto the Data Bus when $\overline{IORQ}$ goes active. Two wait states (T_{W*}) are automatically inserted at this time to allow the daisy chain to stabilize. Additional wait states may be added.



7.2 RETURN FROM INTERRUPT CYCLE

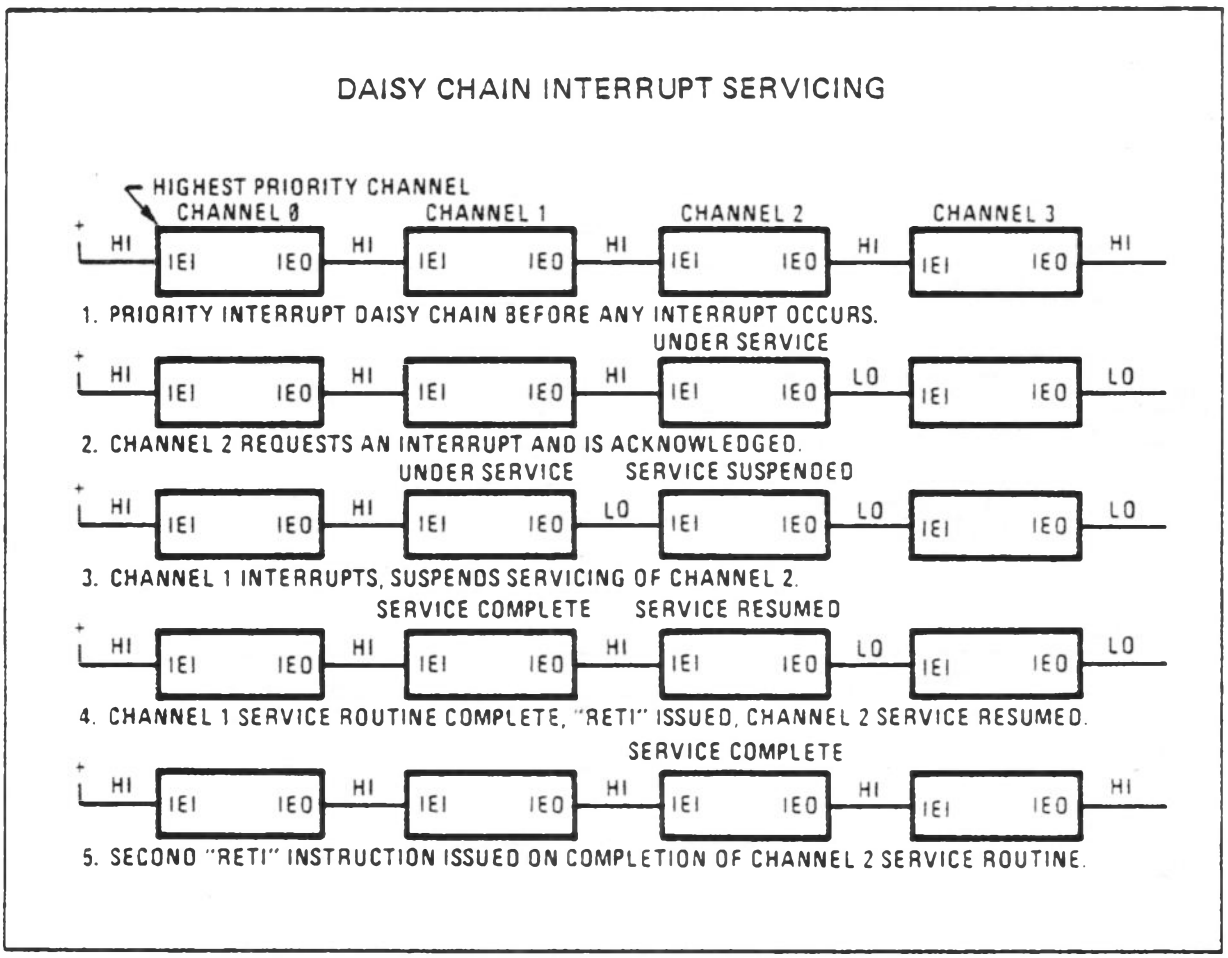
Figure 7.0-2 illustrates the timing associated with the RETI Instruction. This instruction is used at the end of an interrupt service routine to initialize the daisy chain enable lines for proper control of nested priority interrupt handling. The CTC decodes the two-byte RETI code internally and determines whether it is intended for a channel being serviced.

When several Z80 peripheral chips are in the daisy chain IEI will become active on the chip currently under service when an EDH opcode is decoded. If the following opcode is 4DH, the peripheral being serviced will be re-initialized and its IEO will become active. Additional wait states are allowed.



7.3 DAISY CHAIN INTERRUPT SERVICING

Figure 7.0-3 illustrates a typical nested interrupt sequence which may occur in the CTC. In this example, channel 2 interrupts and is granted service. While this channel is being serviced, higher priority channel 1 interrupts and is granted service. The service routine for the higher priority channel is completed, and a RETI instruction (see section 7.2 for further details) is executed to signal the channel that its routine is complete. At this time, the service routine of the lower priority channel 2 is resumed and completed.



8.0 ABSOLUTE MAXIMUM RATINGS

Temperature Under Bias	0° C to 70° C
Storage Temperature	-65° C to + 150° C
Voltage On Any Pin With Respect To Ground	-0.3 V to +7 V
Power Dissipation	0.8 W

***Comment**
Stresses above those listed under "Absolute Maximum Rating" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other condition above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

8.1 D.C. CHARACTERISTICS

TA = 0°C to 70°C, Vcc = 5V ± 5% unless otherwise specified

Z80-CTC

Symbol	Parameter	Min	Max	Unit	Test Condition
V_{ILC}	Clock Input Low Voltage	-0.3	.45	V	
V_{IHC}	Clock Input High Voltage [1]	V_{CC} - .6	V_{CC} + .3	V	
V_{IL}	Input Low Voltage	-0.3	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC}	V	
V_{OL}	Output Low Voltage		0.4	V	I_{OL} = 2 mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = -250 μA
I_{CC}	Power Supply Current		120	mA	T_C = 400 nsec
I_{LI}	Input Leakage Current		10	μA	V_{IN} = 0 to V_{CC}
I_{LOH}	Tri-State Output Leakage Current in Float		10	μA	V_{OUT} = 2.4 to V_{CC}
I_{LOL}	Tri-State Output Leakage Current in Float		-10	μA	V_{OUT} = 0.4V
I_{OHD}	Darlington Drive Current	-1.5		mA	V_{OH} = 1.5V R_{EXT} = 390Ω

Z80A-CTC

Symbol	Parameter	Min	Max	Unit	Test Condition
V_{ILC}	Clock Input Low Voltage	-0.3	.45	V	
V_{IHC}	Clock Input High Voltage [1]	V_{CC} - .6	V_{CC} + .3	V	
V_{IL}	Input Low Voltage	-0.3	0.8	V	
V_{IH}	Input High Voltage	2.0	V_{CC}	V	
V_{OL}	Output Low Voltage		0.4	V	I_{OL} = 2 mA
V_{OH}	Output High Voltage	2.4		V	I_{OH} = -250 μA
I_{CC}	Power Supply Current		120	mA	T_C = 250 nsec
I_{LI}	Input Leakage Current		10	μA	V_{IN} = 0 to V_{CC}
I_{LOH}	Tri-State Output Leakage Current in Float		10	μA	V_{OUT} = 2.4 to V_{CC}
I_{LOL}	Tri-State Output Leakage Current in Float		-10	μA	V_{OUT} = 0.4V
I_{OHD}	Darlington Drive Current	-1.5		mA	V_{OH} = 1.5V R_{EXT} = 390Ω

8.2 CAPACITANCE

TA = 25° C, f = 1 MHz

Symbol	Parameter	Max.	Unit	Test Condition
C_Φ	Clock Capacitance	20	pF	Unmeasured Pins Returned to Ground
C_{IN}	Input Capacitance	5	pF	
C_{OUT}	Output Capacitance	10	pF	

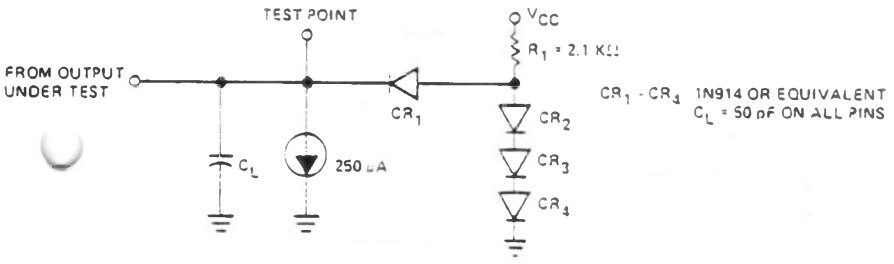
8.3 A.C. CHARACTERISTICS Z80-CTC

T_A = 0° C to 70° C, V_{CC} = +5 V ± 5%, unless otherwise noted

Signal	Symbol	Parameter	Min	Max	Unit	Comments
Φ	t_C	Clock Period	400	[1]	ns	
	$t_W(\Phi H)$	Clock Pulse Width, Clock High	170	2000	ns	
	$t_W(\Phi L)$	Clock Pulse Width, Clock Low	170	2000	ns	
	t_r, t_f	Clock Rise and Fall Times		30	ns	
	t_H	Any Hold Time for Specified Setup Time	0		ns	
CS, $\overline{CE}$, etc.	$t_{S\Phi}(CS)$	Control Signal Setup Time to Rising Edge of Φ During Read or Write Cycle	160		ns	
D_0–D_7	$t_{DR}(D)$	Data Output Delay from Rising Edge of $\overline{RD}$ During Read Cycle		480	ns	[2]
	$t_{S\Phi}(D)$	Data Setup Time to Rising Edge of Φ During Write or M1 Cycle	60		ns	
	$t_{DI}(D)$	Data Output Delay from Falling Edge of IORQ During INTA Cycle		340	ns	[2]
	$t_F(D)$	Delay to Floating Bus (Output Buffer Disable Time)		230	ns	
IEI	$t_S(IEI)$	IEI Setup Time to Falling Edge of $\overline{IORQ}$ During INTA Cycle	200		ns	
IEO	$t_{DH}(IO)$	IEO Delay Time from Rising Edge of IEI		220	ns	[3]
	$t_{DL}(IO)$	IEO Delay Time from Falling Edge of IEI		190	ns	[3]
	$t_{DM}(IO)$	IEO Delay from Falling Edge of $\overline{M1}$ (Interrupt Occurring just Prior to $\overline{M1}$)		300	ns	[3]
$\overline{IORQ}$	$t_{S\Phi}(IR)$	$\overline{IORQ}$ Setup Time to Rising Edge of Φ During Read or Write Cycle	250		ns	
$\overline{M1}$	$t_{S\Phi}(M1)$	$\overline{M1}$ Setup Time to Rising Edge of Φ During INTA or M1 Cycle	210		ns	
$\overline{RD}$	$t_{S\Phi}(RD)$	$\overline{RD}$ Setup Time to Rising Edge of Φ During Read or M1 Cycle	240		ns	
$\overline{INT}$	$t_{DCK}(IT)$	$\overline{INT}$ Delay Time from Rising Edge of CLK/TRG		$2t_C(\Phi)$ + 200		Counter Mode
	$t_{D\Phi}(IT)$	$\overline{INT}$ Delay Time from Rising Edge of Φ		$t_C(\Phi)$ + 200		Timer Mode
CLK/TRG_{0-3}	$t_C(CK)$	Clock Period	$2t_C(\Phi)$			Counter Mode
	t_r, t_f	Clock and Trigger Rise and Fall Times		50		
	$t_S(CK)$	Clock Setup Time to Rising Edge of Φ for Immediate Count	210			Counter Mode
	$t_S(TR)$	Trigger Setup Time to Rising Edge of Φ for Enabling of Prescaler on Following Rising Edge of Φ	210			Timer Mode
	$t_W(CTH)$	Clock and Trigger High Pulse Width	200			Counter and Timer Modes
	$t_W(CTL)$	Clock and Trigger Low Pulse Width	200			Counter and Timer Modes
ZC/TO_{0-2}	$t_{DH}(ZC)$	ZC/TO Delay Time from Rising Edge of Φ, ZC/TO High		190		Counter and Timer Modes
	$t_{DL}(ZC)$	ZC/TO Delay Time from Falling Edge of Φ, ZC/TO Low		190		Counter and Timer Modes

Notes: [1] $t_C = t_W(\Phi H) + t_W(\Phi L) + t_r + t_f$.
[2] Increase delay by 10 nsec for each 50 pF increase in loading, 200 pF maximum for data lines and 100 pF for control lines.
[3] Increase delay by 2 nsec for each 10 pF increase in loading, 100 pF maximum
[4] $\overline{RESET}$ must be active for a minimum of 3 clock cycles.

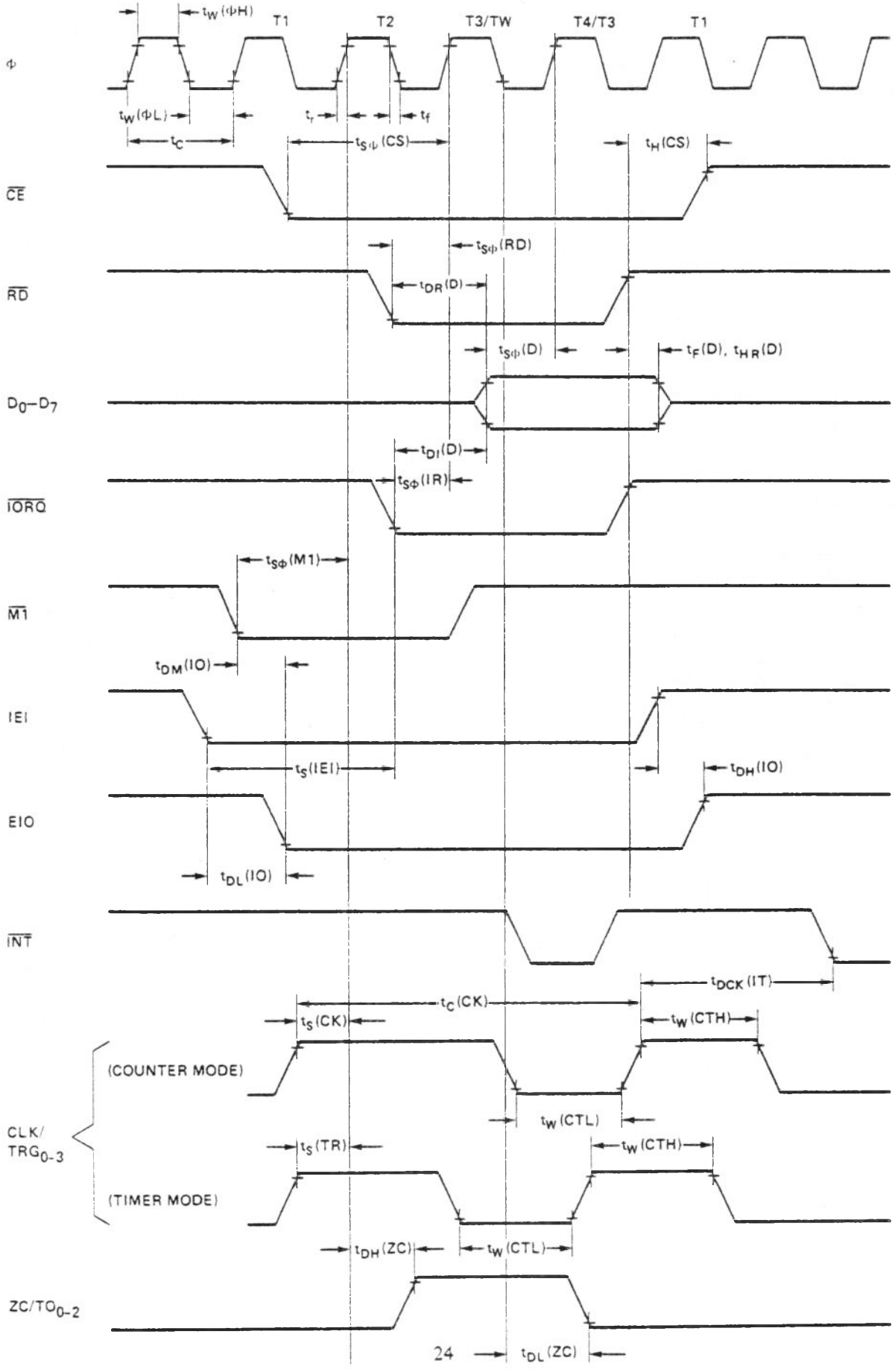
OUTPUT LOAD CIRCUIT



8.4 A.C. TIMING DIAGRAM

Timing measurements are made at the following voltages, unless otherwise specified:

	"1"	"0"
CLOCK	V_{CC} - .6V	.45V
OUTPUT	2.0V	.8V
INPUT	2.0V	.8V
FLOAT	ΔV	±0.5V



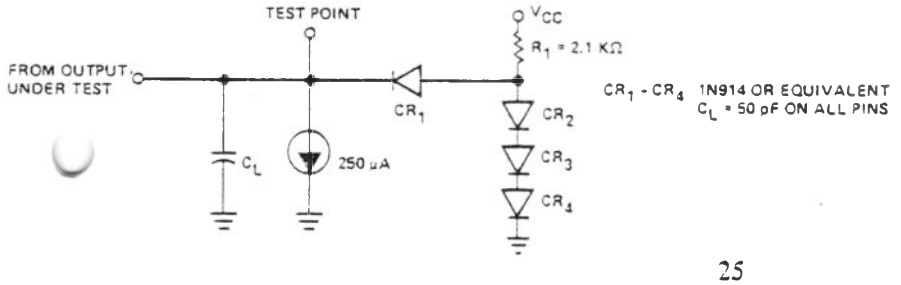
8.5 A.C. CHARACTERISTICS Z80A-CTC

T_A 0° C to 70° C, V_{CC} = +5 V ± 5%, unless otherwise noted

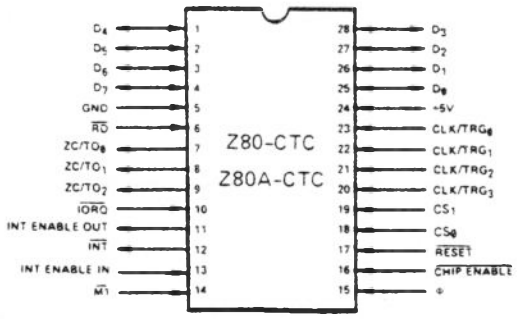
Signal	Symbol	Parameter	Min	Max	Unit	Comments
Φ	t_C	Clock Period	250	[1]	ns	
	$t_W(\Phi H)$	Clock Pulse Width, Clock High	105	2000	ns	
	$t_W(\Phi L)$	Clock Pulse Width, Clock Low	105	2000	ns	
	t_r, t_f	Clock Rise and Fall Times		30	ns	
	t_H	Any Hold Time for Specified Setup Time	0		ns	
CS, $\overline{CE}$, etc	$t_{S\Phi}(CS)$	Control Signal Setup Time to Rising Edge of Φ During Read or Write Cycle	60		ns	
D_0–D_7	$t_{DR}(D)$	Data Output Delay from Falling Edge of $\overline{RD}$ During Read Cycle		380	ns	[2]
	$t_{S\Phi}(D)$	Data Setup Time to Rising Edge of Φ During Write or M1 Cycle	50		ns	
	$t_{DI}(D)$	Data Output Delay from Falling Edge of IORG During INTA Cycle		160	ns	[2]
	$t_F(D)$	Delay to Floating Bus (Output Buffer Disable Time)		110	ns	
IEI	$t_S(IEI)$	IEI Setup Time to Falling Edge of $\overline{IORQ}$ During INTA Cycle	140		ns	
IEO	$t_{DH}(IO)$	IEO Delay Time from Rising Edge of IEI		160	ns	[3]
	$t_{DL}(IO)$	IEO Delay Time from Falling Edge of IEI		130	ns	[3]
	$t_{DM}(IO)$	IEO Delay from Falling Edge of $\overline{M1}$ (Interrupt Occurring just Prior to $\overline{M1}$)		190	ns	[3]
$\overline{IORQ}$	$t_{S\Phi}(IR)$	$\overline{IORQ}$ Setup Time to Rising Edge of Φ During Read or Write Cycle	115		ns	
$\overline{M1}$	$t_{S\Phi}(M1)$	$\overline{M1}$ Setup Time to Rising Edge of Φ During INTA or M1 Cycle	90		ns	
$\overline{RD}$	$t_{S\Phi}(RD)$	$\overline{RD}$ Setup Time to Rising Edge of Φ During Read or M1 Cycle	115		ns	
$\overline{INT}$	$t_{DCK}(IT)$	$\overline{INT}$ Delay Time from Rising Edge of CLK/TRG		$2t_C(\Phi) + 140$		Counter Mode
	$t_{D\Phi}(IT)$	$\overline{INT}$ Delay Time from Rising Edge of Φ		$t_C(\Phi) + 140$		Timer Mode
CLK/TRG_{0-3}	$t_C(CK)$	Clock Period	$2t_C(\Phi)$			Counter Mode
	t_r, t_f	Clock and Trigger Rise and Fall Times		30		
	$t_S(CK)$	Clock Setup Time to Rising Edge of Φ for Immediate Count	130			Counter Mode
	$t_S(TR)$	Trigger Setup Time to Rising Edge of Φ for enabling of Prescaler on Following Rising Edge of Φ	130			Timer Mode
	$t_W(CTH)$	Clock and Trigger High Pulse Width	120			Counter and Timer Modes
	$t_W(CTL)$	Clock and Trigger Low Pulse Width	120			Counter and Timer Modes
ZC/TO_{0-2}	$t_{DH}(ZC)$	ZC/TO Delay Time from Rising Edge of Φ, ZC/TO High		120		Counter and Timer Modes
	$t_{DL}(ZC)$	ZC/TO Delay Time from Rising Edge of Φ, ZC/TO Low		120		Counter and Timer Modes

Notes: [1] $t_C = t_W(\Phi H) + t_W(\Phi L) + t_r + t_f$.
[2] Increase delay by 10 nsec for each 50 pF increase in loading, 200 pF maximum for data lines and 100 pF for control lines.
[3] Increase delay by 2 nsec for each 10 pF increase in loading, 100 pF maximum.
[4] $\overline{RESET}$ must be active for a minimum of 3 clock cycles.

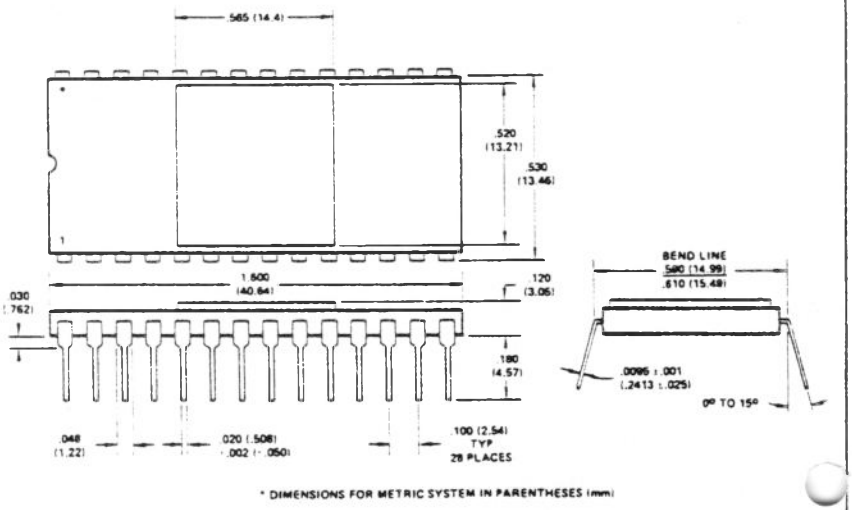
OUTPUT LOAD CIRCUIT



8.6 PACKAGE CONFIGURATION



PACKAGE OUTLINE



* DIMENSIONS FOR METRIC SYSTEM IN PARENTHESES (mm)

Ordering Information

C – Ceramic
P – Plastic
S – Standard 5V±5%, 0° to 70°C
E – Extended 5V±5%, -40° to 85°C
M – Military 5V±10%, -55° to 125°C

Example:

Z80-CTC CS (Ceramic–Standard Range)
Z80A-CTC PS (Plastic–Standard Range, 4 MHz)

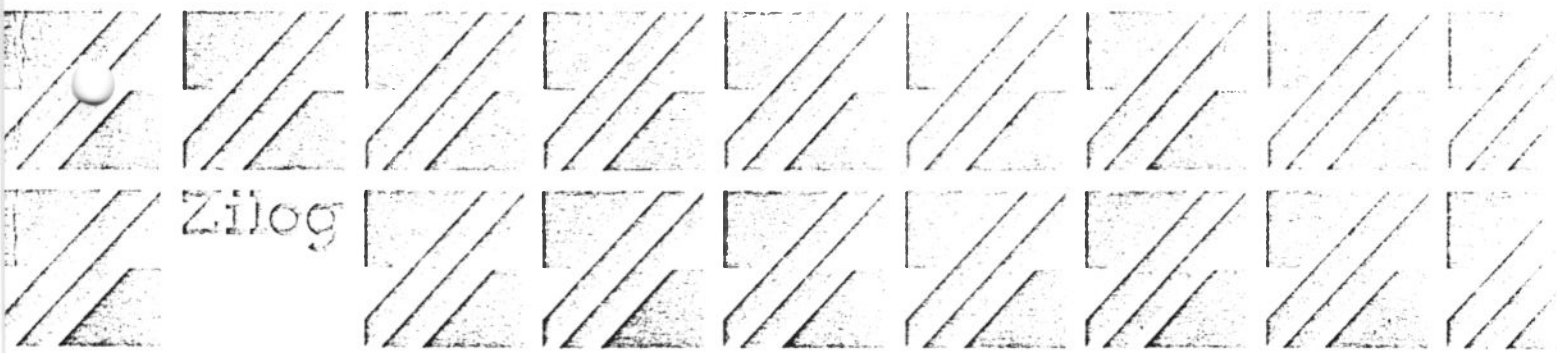
ZILOG Z80 MICROCOMPUTER SYSTEM COMPONENT FAMILY

- Z80, Z80A-CPU — CENTRAL PROCESSOR UNIT
- Z80, Z80A-PIO — PARALLEL I/O
- Z80, Z80A-CTC — COUNTER/TIMER CIRCUIT
- Z80, Z80A-DMA — DIRECT MEMORY ACCESS
- Z80, Z80A-SIO — SERIAL I/O
- Z6104 — 4K x 1 STATIC RAM
- Z6116 — 16K x 1 DYNAMIC RAM

10460 Bubb Road
Cupertino, California 95014
Telephone (408) 446-4666
TWX 910-338-7621

3-0036-00

Z-80® SIO
TECHNICAL MANUAL



Z80-SIO Technical Manual

Contents

General Information 1
Pin Description 2
Architecture 5
The Data Path 5
Functional Description 7
Asynchronous Operation 9
Asynchronous Transmit 9
Asynchronous Receive 10
Synchronous Operation 13
Synchronous Transmit 14
Synchronous Receive 17
SDLC (HDLC) Operation 21
SDLC Transmit 21
SDLC Receive 25
Z80-SIO Programming 29
Write Registers 29
Read Registers 34
Applications 59
Timing 41

General Information

The Z80-SIO (Serial Input/Output) is a dual-channel multi-function peripheral component designed to satisfy a wide variety of serial data communications requirements in microcomputer systems. Its basic function is a serial-to-parallel, parallel-to-serial converter/controller, but—within that role—it is configurable by systems software so its "personality" can be optimized for a given serial data communications application.

The Z80-SIO is capable of handling asynchronous and synchronous byte-oriented protocols such as IBM Bisync, and synchronous bit-oriented protocols such as HDLC and IBM SDLC. This versatile device can also be used to support virtually any other serial protocol for applications other than data communications (cassette or floppy disk interfaces, for example).

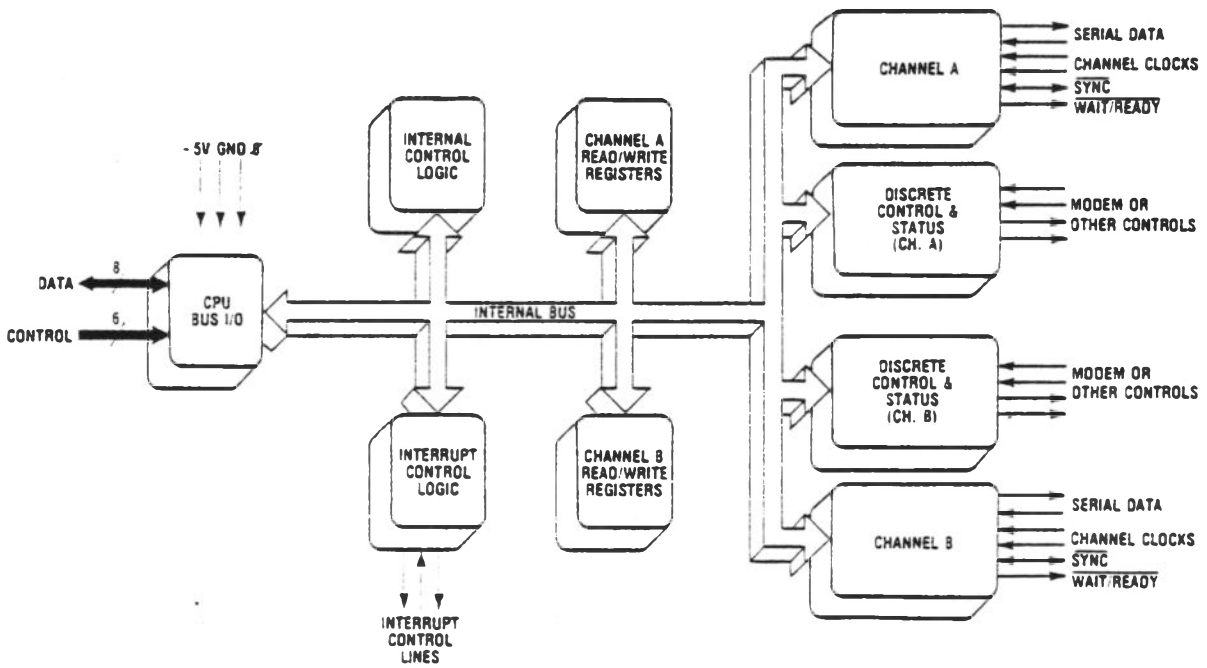
The Z80-SIO can generate and check CRC codes in any synchronous mode and can be programmed to check data integrity in various modes. The device also has facilities for modem controls in both channels. In applications where these controls are not needed, the modem controls can be used for general-purpose I/O.

STRUCTURE

- N-channel silicon-gate depletion-load technology
- 40-pin DIP
- Single 5 V power supply
- Single-phase 5 V clock
- All inputs and outputs TTL compatible

FEATURES

- Two independent full-duplex channels
- Data rates in synchronous or isosynchronous modes:
  - 0–550K bits/second with 2.5 MHz system clock rate
  - 0–880K bits/second with 4.0 MHz system clock rate
- Receiver data registers quadruply buffered; transmitter doubly buffered.
- Asynchronous features:
  - 5, 6, 7 or 8 bits/character
  - 1, 1½ or 2 stop bits
  - Even, odd or no parity
  - ×1, ×16, ×32 and ×64 clock modes
  - Break generation and detection
  - Parity, overrun and framing error detection



Z80-SIO BLOCK DIAGRAM

- Binary synchronous features:
  - Internal or external character synchronization
  - One or two sync characters in separate registers
  - Automatic sync character insertion
  - CRC generation and checking
- HDLC and IBM SDLC features:
  - Abort sequence generation and detection
  - Automatic zero insertion and deletion
  - Automatic flag insertion between messages
  - Address field recognition
  - I-field residue handling
  - Valid receive messages protected from overrun
  - CRC generation and checking
- Separate modem control inputs and outputs for both channels
- CRC-16 or CRC-CCITT block check
- Daisy-chain priority interrupt logic provides automatic interrupt vectoring without external logic
- Modem status can be monitored

Pin Description

D_0–D_7. *System Data Bus* (bidirectional, 3-state). The system data bus transfers data and commands between the CPU and the Z80-SIO. D_0 is the least significant bit.

B/$\overline{A}$. *Channel A Or B Select* (input, High selects Channel B). This input defines which channel is accessed during a data transfer between the CPU and the Z80-SIO. Address bit A_0 from the CPU is often used for the selection function.

C/$\overline{D}$. *Control Or Data Select* (input, High selects Control). This input defines the type of information transfer performed between the CPU and the Z80-SIO. A High at this input during a CPU write to the Z80-SIO causes the information on the data bus to be interpreted as a command for the channel selected by B/$\overline{A}$. A Low at C/$\overline{D}$ means that the information on the data bus is data. Address bit A_1 is often used for this function.

$\overline{CE}$. *Chip Enable* (input, active Low). A Low level at this input enables the Z80-SIO to accept command or data inputs from the CPU during a write cycle, or to transmit data to the CPU during a read cycle.

ϕ. *System Clock* (input). The Z80-SIO uses the standard Z80A System Clock to synchronize internal signals. This is a single-phase clock.

$\overline{M1}$. *Machine Cycle One* (input from Z80-CPU, active Low). When $\overline{M1}$ is active and $\overline{RD}$ is also active, the Z80-CPU is fetching an instruction from memory; when $\overline{M1}$ is active while $\overline{IORQ}$ is active, the Z80-SIO accepts $\overline{M1}$ and $\overline{IORQ}$ as an interrupt acknowledge if the Z80-SIO is the highest priority device that has interrupted the Z80-CPU.

$\overline{IORQ}$. *Input/Output Request* (input from CPU, active Low). $\overline{IORQ}$ is used in conjunction with B/$\overline{A}$, C/$\overline{D}$, $\overline{CE}$ and $\overline{RD}$ to transfer commands and data between the CPU and the Z80-SIO. When $\overline{CE}$, $\overline{RD}$ and $\overline{IORQ}$ are all active,

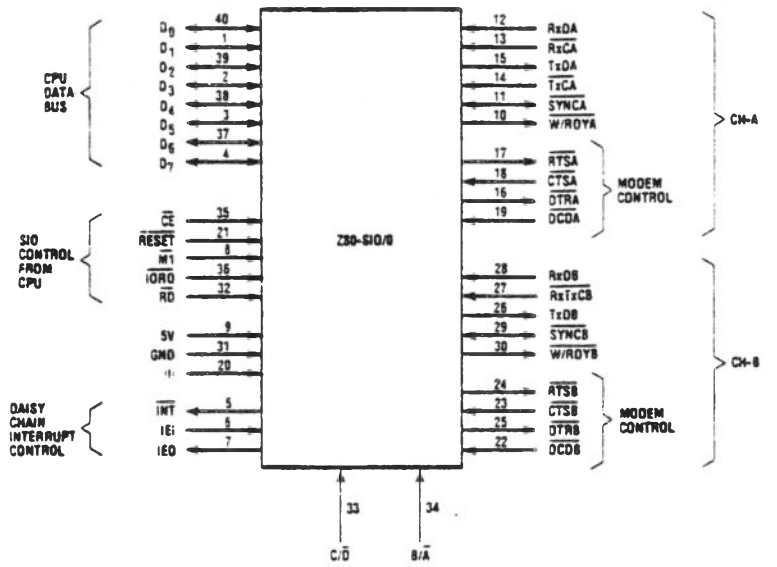


Figure 1. Z80-SIO/0 Pin Configuration

the channel selected by B/$\overline{A}$ transfers data to the CPU (a read operation). When $\overline{CE}$ and $\overline{IORQ}$ are active, but $\overline{RD}$ is inactive, the channel selected by B/$\overline{A}$ is written to by the CPU with either data or control information as specified by C/$\overline{D}$. As mentioned previously, if $\overline{IORQ}$ and $\overline{M1}$ are active simultaneously, the CPU is acknowledging an interrupt and the Z80-SIO automatically places its interrupt vector on the CPU data bus if it is the highest priority device requesting an interrupt.

$\overline{RD}$. *Read Cycle Status.* (input from CPU, active Low). If $\overline{RD}$ is active, a memory or I/O read operation is in progress. $\overline{RD}$ is used with B/$\overline{A}$, $\overline{CE}$ and $\overline{IORQ}$ to transfer data from the Z80-SIO to the CPU.

$\overline{RESET}$. *Reset* (input, active Low). A Low $\overline{RESET}$ disables both receivers and transmitters, forces TxDA and TxDB marking, forces the modem controls High and disables all interrupts. The control registers must be rewritten after the Z80-SIO is reset and before data is transmitted or received.

IEI. *Interrupt Enable In* (input, active High). This signal is used with IEO to form a priority daisy chain when there is more than one interrupt-driven device. A High on this line indicates that no other device of higher priority is being serviced by a CPU interrupt service routine.

IEO. *Interrupt Enable Out* (output, active High). IEO is High only if IEI is High and the CPU is not servicing an interrupt from this Z80-SIO. Thus, this signal blocks lower priority devices from interrupting while a higher priority device is being serviced by its CPU interrupt service routine.

$\overline{INT}$. *Interrupt Request* (output, open drain, active Low). When the Z80-SIO is requesting an interrupt, it pulls $\overline{INT}$ Low.

$\overline{W/RDYA}$, $\overline{W/RDYB}$. *Wait/Ready A, Wait/Ready B* (outputs, open drain when programmed for Wait function, driven High and Low when programmed for Ready function). These dual-purpose outputs may be programmed as Ready lines for a DMA controller or as Wait lines that synchronize the CPU to the Z80-SIO data rate. The reset state is open drain.

$\overline{CTSA}$, $\overline{CTSB}$. *Clear To Send* (inputs, active Low). When programmed as Auto Enables, a Low on these inputs enables the respective transmitter. If not programmed as Auto Enables, these inputs may be programmed as general-purpose inputs. Both inputs are Schmitt-trigger buffered to accommodate slow-risetime inputs. The Z80-SIO detects pulses on these inputs and interrupts the CPU on both logic level transitions. The Schmitt-trigger inputs do not guarantee a specified noise-level margin.

$\overline{DCDA}$, $\overline{DCDB}$. *Data Carrier Detect* (inputs, active Low). These signals are similar to the CTS inputs, except they can be used as receiver enables.

RxDA, RxDB. *Receive Data* (inputs, active High).

TxDA, TxDB. *Transmit Data* (outputs, active High).

$\overline{RxCA}$, $\overline{RxCB}$.* *Receiver Clocks* (inputs). See the following section on bonding options. The Receive Clocks may be 1, 16, 32 or 64 times the data rate in asynchronous modes. Receive data is sampled on the rising edge of $\overline{RxC}$.

*See footnote on next page.

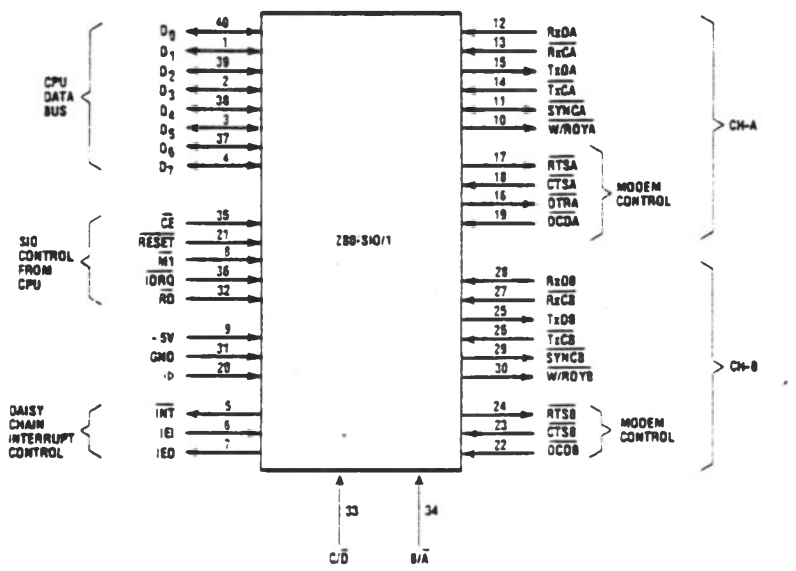


Figure 2. Z80-SIO/1 Pin Configuration

$\overline{\text{TxCA}}$, $\overline{\text{TxCB}}$.* *Transmitter Clocks* (inputs). See section on bonding options. In asynchronous modes, the Transmitter clocks may be 1, 16, 32 or 64 times the data rate. The multiplier for the transmitter and the receiver must be the same. Both the $\overline{\text{TxC}}$ and $\overline{\text{RxC}}$ inputs are Schmitt-trigger buffered for relaxed rise- and fall-time requirements (no noise margin is specified). TxD changes on the falling edge of $\overline{\text{TxC}}$.

$\overline{\text{RTSA}}$, $\overline{\text{RTSB}}$. *Request To Send* (outputs, active Low). When the RTS bit is set, the $\overline{\text{RTS}}$ output goes Low. When the RTS bit is reset in the Asynchronous mode, the output goes High after the transmitter is empty. In Synchronous modes, the $\overline{\text{RTS}}$ pin strictly follows the state of the RTS bit. Both pins can be used as general-purpose outputs.

$\overline{\text{DTRA}}$, $\overline{\text{DTRB}}$. *Data Terminal Ready* (outputs, active Low). See note on bonding options. These outputs follow the state programmed into the DTR bit. They can also be programmed as general-purpose outputs.

$\overline{\text{SYNC}}$ A, $\overline{\text{SYNC}}$ B. *Synchronization* (inputs/outputs, active Low). These pins can act either as inputs or outputs. In the Asynchronous Receive mode, they are inputs similar to $\overline{\text{CTS}}$ and $\overline{\text{DCD}}$. In this mode, the transitions on these lines affect the state of the Sync/Hunt status bits in RR0. In the External Sync mode, these lines also act as inputs. When external synchronization is achieved, $\overline{\text{SYNC}}$ must be driven Low on the second rising edge of $\overline{\text{RxC}}$ after that rising edge of $\overline{\text{RxC}}$ on which the last bit of the sync character was received. In other words, after the sync pattern is detected, the external logic must wait for two full Receive Clock cycles to activate the $\overline{\text{SYNC}}$ input. Once $\overline{\text{SYNC}}$ is forced Low, it is wise to keep it Low until the CPU informs the external sync logic that synchronization has been lost or a new message is about to start. Character assembly begins on the rising edge of $\overline{\text{RxC}}$ that immediately precedes the falling edge of $\overline{\text{SYNC}}$ in the External Sync mode.

In the Internal Synchronization mode (Monosync and Bisync), these pins act as outputs that are active during the part of the receive clock ($\overline{\text{RxC}}$) cycle in which sync characters are recognized. The sync condition is not latched, so these outputs are active each time a sync pattern is recognized, regardless of character boundaries.

BONDING OPTIONS

The constraints of a 40-pin package make it impossible to bring out the Receive Clock, Transmit Clock, Data Terminal Ready and Sync signals for both channels. Therefore, Channel B must sacrifice a signal or have two signals bonded together. Since user requirements vary, three bondings options are offered:

- Z80-SIO/0 has all four signals, but $\overline{\text{TxCB}}$ and $\overline{\text{RxCB}}$ are bonded together (Fig. 1).
- Z80-SIO/1 sacrifices $\overline{\text{DTRB}}$ and keeps $\overline{\text{TxCB}}$, $\overline{\text{RxCB}}$ and $\overline{\text{SYNCB}}$ (Fig. 2).
- Z80-SIO/2 sacrifices $\overline{\text{SYNCB}}$ and keeps $\overline{\text{TxCB}}$, $\overline{\text{RxCB}}$ and $\overline{\text{DTRB}}$ (Fig. 3).

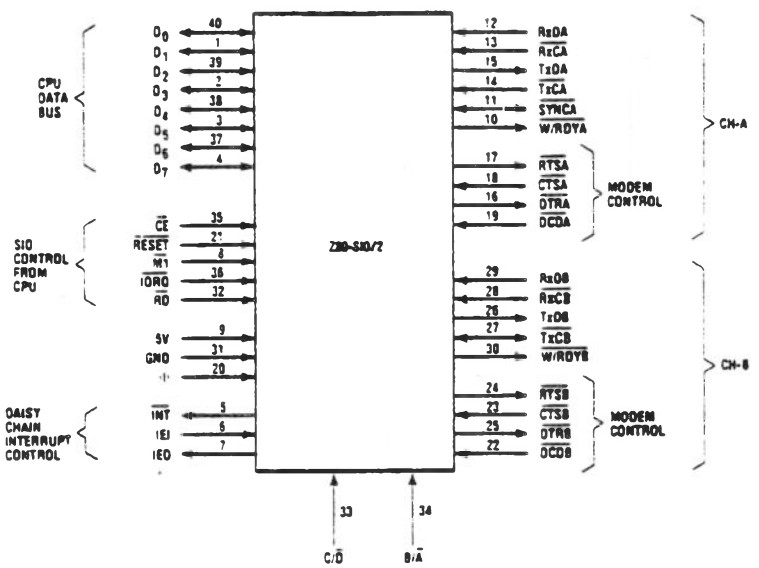


Figure 3. Z80-SIO/2 Pin Configuration

*These clocks may be directly driven by the Z80-CTC (Counter Timer Circuit) for fully programmable baud rate generation.

Architecture

The device internal structure includes a Z80-CPU interface, internal control and interrupt logic, and two full-duplex channels. Associated with each channel are read and write registers, and discrete control and status logic that provides the interface to modems or other external devices.

The read and write register group includes five 8-bit control registers, two sync-character registers and two status registers. The interrupt vector is written into an additional 8-bit register (Write Register 2) in Channel B that may be read through Read Register 2 in Channel B. The registers for both channels are designated in the text as follows:

WR0-WR7 — Write Registers 0 through 7
RR0-RR2 — Read Registers 0 through 2

The bit assignment and functional grouping of each register is configured to simplify and organize the programming process. Table 1 illustrates the functions assigned to each read or write register.

WR0	Register pointers, CRC initialize, initialization commands for the various modes, etc.
WR1	Transmit/Receive interrupt and data transfer mode definition.
WR2	Interrupt vector (Channel B only)
WR3	Receive parameters and controls
WR4	Transmit/Receive miscellaneous parameters and modes
WR5	Transmit parameters and controls
WR6	Sync character or SDLC address field
WR7	Sync character or SDLC flag

(a) Write Register Functions

RR0	Transmit/Receive buffer status, interrupt status and external status
RR1	Special Receive Condition status
RR2	Modified interrupt vector (Channel B only)

(b) Read Register Functions

Table 1. Functional Assignments of Read and Write Registers

The logic for both channels provides formats, synchronization and validation for data transferred to and from the channel interface. The modem control inputs Clear to Send (CTS) and Data Carrier Detect (DCD) are monitored by the discrete control logic under program control. All the modem control signals are general purpose in nature and can be used for functions other than modem control.

For automatic interrupt vectoring, the interrupt control logic determines which channel and which device within the channel has the highest priority. Priority is fixed with Channel A assigned a higher priority than Channel B; Receive, Transmit and External/ Status interrupts are prioritized in that order within each channel.

Data Path

The transmit and receive data path for each channel is shown in Figure 4. The receiver has three 8-bit buffer registers in a FIFO arrangement (to provide a 3-byte delay) in addition to the 8-bit receive shift register. This arrangement creates additional time for the CPU to service an interrupt at the beginning of a block of high-speed data. The receive error FIFO stores parity and framing errors and other types of status information for each of the three bytes in the receive data FIFO.

Incoming data is routed through one of several paths depending on the mode and character length. In the Asynchronous mode, serial data is entered in the 3-bit buffer if it has a character length of seven or eight bits, or is entered in the 8-bit receive shift register if it has a length of five or six bits.

In the Synchronous mode, however, the data path is determined by the phase of the receive process currently in operation. A Synchronous Receive operation begins with the receiver in the Hunt phase, during which the receiver searches the incoming data stream for a bit pattern that matches the preprogrammed sync characters (or flags in the SDLC mode). If the device is programmed for Monosync Hunt, a match is made with a single sync character stored in WR7. In Bisync Hunt, a match is made with dual sync characters stored in WR6 and WR7.

In either case the incoming data passes through the receive sync register, and is compared against the programmed sync character in WR6 or WR7. In the Monosync mode, a match between the sync character programmed into WR7 and the character assembled in the receive sync register establishes synchronization.

In the Bisync mode, however, incoming data is shifted to the receive shift register while the next eight bits of the message are assembled in the receive sync register. The match between the assembled character in the receive sync registers with the programmed sync character in WR6 and WR7 establishes synchronization. Once synchronization is established, incoming data by-

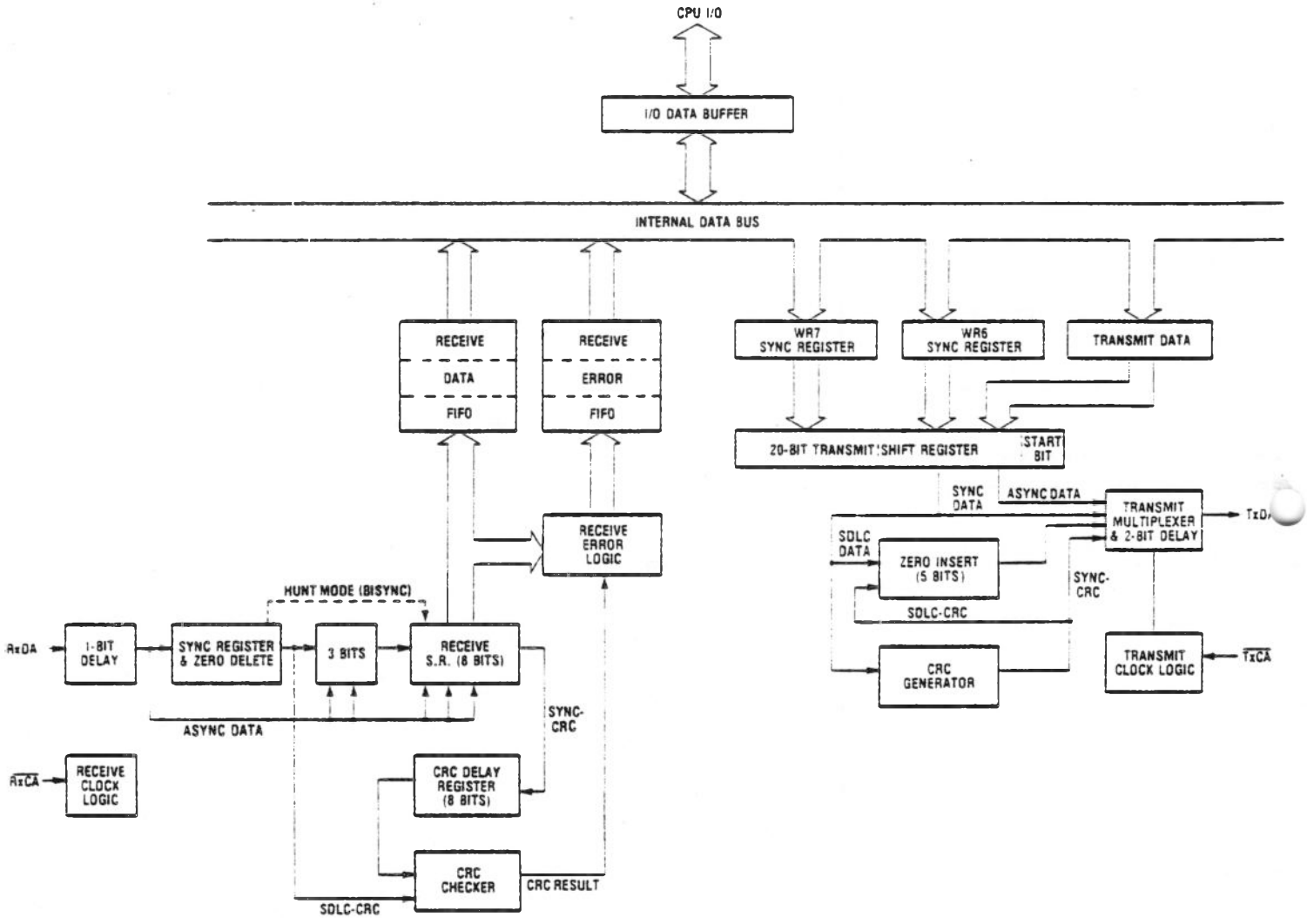


Figure 4. Transmit and Receive Data Path

passes the receive sync register and directly enters the 3-bit buffer.

In the SDLC mode, incoming data first passes through the receive sync register, which continuously monitors the receive data stream and performs zero deletion when indicated. Upon receiving five contiguous 1's, the sixth bit is inspected. If the sixth bit is a 0, it is deleted from the data stream. If the sixth bit is a 1, the seventh bit is inspected. If that bit is a 0, a Flag sequence has been received; if it is a 1, an Abort sequence has been received.

The reformatted data enters the 3-bit buffer and is transferred to the receive shift register. Note that the SDLC receive operation also begins in the Hunt phase, during which the Z80-SIO tries to match the assembled character in the receive shift register with the flag pattern in WR7. Once the first flag character is recognized, all subsequent data is routed through the same path, regardless of character length.

Although the same CRC checker is used for both SDLC and synchronous data, the data path taken for each mode is different. In Bisync protocol, a byte-oriented operation requires that the CPU decide to include the data character in CRC. To allow the CPU ample time to make this decision, the Z80-SIO provides an 8-bit delay for synchronous data. In the SDLC mode, no delay is provided since the Z80-SIO contains logic that determines the bytes on which CRC is calculated.

The transmitter has an 8-bit transmit data register that is loaded from the internal data bus and a 20-bit transmit shift register that can be loaded from WR6, WR7 and the transmit data register. WR6 and WR7 contain sync characters in the Monosync or Bisync modes, or address field (one character long) and flag respectively in the SDLC mode. During Synchronous modes, information contained in WR6 and WR7 is loaded into the transmit shift register at the beginning of the message and, as a time filler, in the middle of the message if a Transmit Underrun condition occurs. In the SDLC mode, the flags are loaded into the transmit shift register at the beginning and end of message.

Asynchronous data in the transmit shift register is formatted with start and stop bits and is shifted out to the transmit multiplexer at the selected clock rate. Synchronous (Monosync or Bisync) data is shifted out to the transmit multiplexer and also to the CRC generator at the $\times 1$ clock rate.

SDLC/HDLC data is shifted out through the zero insertion logic, which is disabled while the flags are being sent. For all other fields (address, control and frame check) a 0 is inserted following five contiguous 1's in the data stream. The CRC generator result for SDLC data is also routed through the zero insertion logic.

Functional Description

The functional capabilities of the Z80-SIO can be described from two different points of view: as a data communications device, it transmits and receives serial data, and meets the requirements of various data communications protocols; as a Z80 family peripheral, it interacts with the Z80-CPU and other Z80 peripheral circuits, and shares their data, address and control busses, as well as being a part of the Z80 interrupt structure. As a peripheral to other microprocessors, the Z80-SIO offers valuable features such as non-vectored interrupts, polling and simple handshake capabilities.

The first part of the following functional description describes the interaction between the CPU and Z80-SIO; the second part introduces its data communications capabilities.

I/O CAPABILITIES

The Z80-SIO offers the choice of Polling, Interrupt (vectored or non-vectored) and Block Transfer modes to transfer data, status and control information to and from the CPU. The Block Transfer mode can be implemented under CPU or DMA control.

Polling. The Polled mode avoids interrupts. Status registers RR0 and RR1 are updated at appropriate times for each function being performed (for example, CRC Error status valid at the end of the message). All the interrupt modes of the Z80-SIO must be disabled to operate the device in a polled environment.

While in its Polling sequence, the CPU examines the status contained in RR0 for each channel; the RR0 status bits serve as an acknowledge to the Poll inquiry. The two RR0 status bits D_0 and D_2 indicate that a receive or transmit data transfer is needed. The status also indicates Error or other special status conditions (see "Z80-SIO Programming"). The Special Receive Condition status contained in RR1 does not have to be read in a Polling sequence because the status bits in RR1 are accompanied by a Receive Character Available status in RR0.

Interrupts. The Z80-SIO offers an elaborate interrupt scheme to provide fast interrupt response in real-time applications. As mentioned earlier, Channel B registers WR2 and RR2 contain the interrupt vector that points to an interrupt service routine in the memory. To service operations in both channels and to eliminate the necessity of writing a status analysis routine, the Z80-SIO can modify the interrupt vector in RR2 so it points directly to one of eight interrupt service routines. This is done under program control by setting a program bit (WR1, D_2) in Channel B called "Status Affects Vector." When this bit is set, the interrupt vector in WR2 is modified according to the assigned priority of the various interrupting conditions. The table in the Write Register 1 description (Z80-SIO Programming section) shows the modification details.

Transmit interrupts, Receive interrupts and External/Status interrupts are the main sources of interrupts (Figure 5). Each interrupt source is enabled under program control with Channel A having a higher priority than Channel B, and with Receiver, Transmit and External/Status interrupts prioritized in that order within each channel. When the Transmit interrupt is enabled, the CPU is interrupted by the transmit buffer *becoming* empty. (This implies that the transmitter must have had a data character written into it so it can become empty.) When enabled, the receiver can interrupt the CPU in one of three ways:

- Interrupt on first receive character
- Interrupt on all receive characters
- Interrupt on a Special Receive condition

Interrupt On First Character is typically used with the Block Transfer mode. Interrupt On All Receive Characters has the option of modifying the interrupt vector in the event of a parity error. The Special Receive Condition interrupt can occur on a character or message basis (End Of Frame interrupt in SDLC, for example). The Special Receive condition can cause an interrupt only if the Interrupt On First Receive Character or Interrupt On All Receive Characters mode is selected. In Interrupt On First Receive Character, an interrupt can occur from Special Receive conditions (except Parity Error) after the first receive character interrupt (example: Receive Overrun interrupt).

The main function of the External/Status interrupt is to monitor the signal transitions of the $\overline{\text{CTS}}$, $\overline{\text{DCD}}$ and $\overline{\text{SYNC}}$ pins; however, an External/Status interrupt is also caused by a Transmit Underrun condition or by the detection of a Break (Asynchronous mode) or Abort (SDLC mode) sequence in the data stream. The interrupt caused by the Break/Abort sequence has a special feature that allows the Z80-SIO to interrupt when the Break/Abort sequence is detected or terminated. This feature facilitates the proper termination of the current message, correct initialization of the next message, and the accurate timing of the Break/Abort condition in external logic.

CPU/DMA Block Transfer. The Z80-SIO provides a Block Transfer mode to accommodate CPU block transfer functions and DMA controllers (Z80-DMA or other designs). The Block Transfer mode uses the $\overline{\text{WAIT}}/\overline{\text{READY}}$ output in conjunction with the Wait/Ready bits of Write Register 1. The $\overline{\text{WAIT}}/\overline{\text{READY}}$ output can be defined under software control as a WAIT line in the CPU Block Transfer mode or as a READY line in the DMA Block Transfer mode.

To a DMA controller, the Z80-SIO READY output indicates that the Z80-SIO is ready to transfer data to or from memory. To the CPU, the WAIT output indicates that the Z80-SIO is not ready to transfer data, thereby requesting the CPU to extend the I/O cycle. The programming of bits 5, 6 and 7 of Write Register 1 and the logic states of the $\overline{\text{WAIT}}/\overline{\text{READY}}$ line are defined in the Write Register 1 description (Z80-SIO Programming section.)

DATA COMMUNICATIONS CAPABILITIES

In addition to the I/O capabilities previously discussed, the Z80-SIO provides two independent full-duplex channels as well as Asynchronous, Synchronous and SDLC (HDLC) operational modes. These modes facilitate the implementation of commonly used data communications protocols.

The specific features of these modes are described in the following sections. To preserve the independence and completeness of each section, some information common to all modes is repeated.

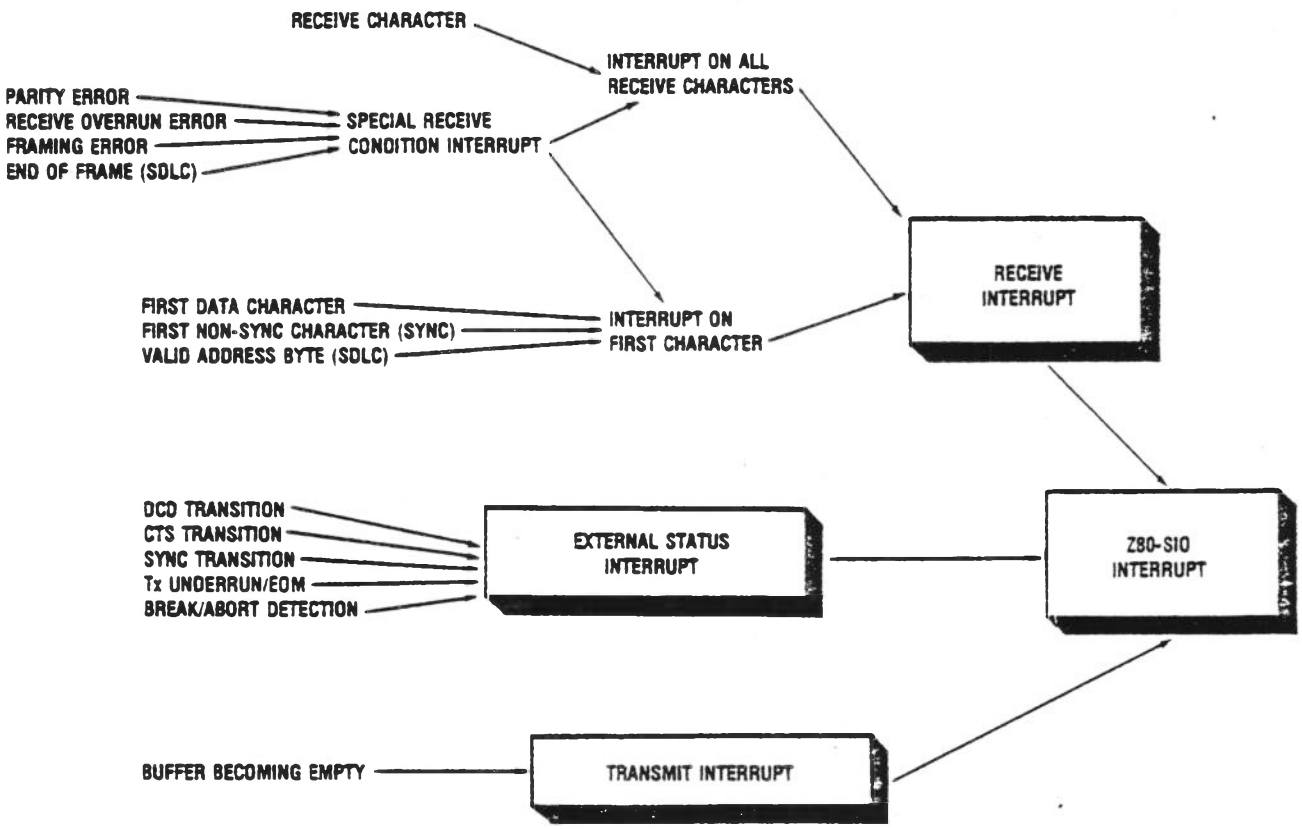


Figure 5. Interrupt Structure

Asynchronous Operation

To receive or transmit data in the Asynchronous mode, the Z80-SIO must be initialized with the following parameters: character length, clock rate, number of stop bits, even or odd parity, interrupt mode, and receiver or transmitter enable. The parameters are loaded into the appropriate write registers by the system program. WR4 parameters must be issued before WR1, WR3 and WR5 parameters or commands.

If the data is transmitted over a modem or RS232C interface, the REQUEST TO SEND ($\overline{RTS}$) and DATA TERMINAL READY ($\overline{DTR}$) outputs must be set along with the Transmit Enable bit. Transmission cannot begin until the Transmit Enable bit is set.

The Auto Enables feature allows the programmer to send the first data character of the message to the Z-SIO without waiting for $\overline{CTS}$. If the Auto Enables bit is set, the Z80-SIO will wait for the $\overline{CTS}$ pin to go Low before it begins data transmission. $\overline{CTS}$, $\overline{DCD}$ and $\overline{SYNC}$ are general-purpose I/O lines that may be used for functions other than their labeled purposes. If $\overline{CTS}$ is used for another purpose, the Auto Enables Bit must be programmed to 0.

Figure 6 illustrates asynchronous message formats; Table 2 shows WR3, WR4 and WR5 with bits set to indicate the applicable modes, parameters and commands in asynchronous modes. WR2 (Channel B only) stores the interrupt vector; WR1 defines the interrupt modes and data transfer modes. WR6 and WR7 are not used in asynchronous modes. Table 3 shows the typical program steps that implement a full-duplex receive/transmit operation in either channel.

Asynchronous Transmit

The Transmit Data output (TxD) is held marking (High) when the transmitter has no data to send. Under program control, the Send Break (WR5, D_4) command can be issued to hold TxD spacing (Low) until the command is cleared.

The Z80-SIO automatically adds the start bit, the programmed parity bit (odd, even or no parity) and the programmed number of stop bits to the data character to be transmitted. When the character length is six or seven bits, the unused bits are automatically ignored by the Z80-SIO. If the character length is five bits or less, refer to the table in the Write Register 5 description (Z80-SIO Programming section) for the data format.

Serial data is shifted from TxD at a rate equal to 1, 1/16th, 1/32nd or 1/64th of the clock rate supplied to the Transmit Clock input ($\overline{TxC}$). Serial data is shifted out on the falling edge of ($\overline{TxC}$).

If set, the External/Status Interrupt mode monitors the status of $\overline{DCD}$, $\overline{CTS}$ and $\overline{SYNC}$ throughout the transmission of the message. If these inputs change for a period of time greater than the minimum specified pulse width, the interrupt is generated. In a transmit operation, this feature is used to monitor the modem control signal $\overline{CTS}$.

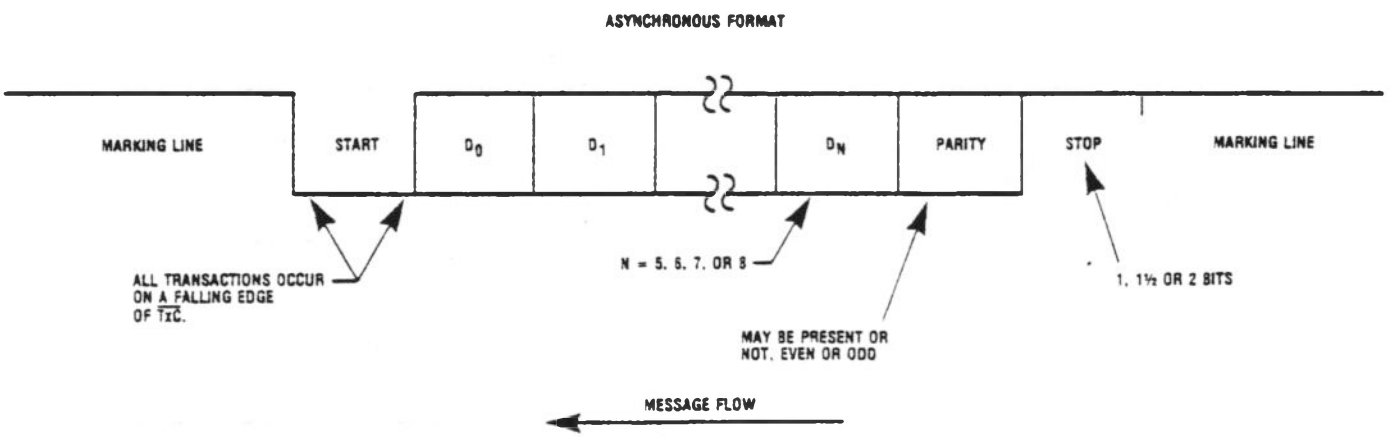


Figure 6. Asynchronous Message Format

Asynchronous Receive

An Asynchronous Receive operation begins when the Receive Enable bit is set. If the Auto Enables option is selected, $\overline{DCD}$ must be Low as well. A Low (spacing) condition on the Receive Data input (RxD) indicates a start bit. If this Low persists for at least one-half of a bit time, the start bit is assumed to be valid and the data input is then sampled at mid-bit time until the entire character is assembled. This method of detecting a start bit improves error rejection when noise spikes exist on an otherwise marking line.

If the ×1 clock mode is selected, bit synchronization must be accomplished externally. Receive data is sampled on the rising edge of RxC. The receiver inserts 1's when a character length of other than eight bits is used. If parity is enabled, the parity bit is not stripped from the assembled character for character lengths other than eight bits. For lengths other than eight bits, the receiver assembles a character length of the required number of data bits, plus a parity bit and 1's for any unused bits. For example, the receiver assembles a 5-bit character with the following format: 1 1 P D_4 D_3 D_2 D_1 D_0.

Since the receiver is buffered by three 8-bit registers in addition to the receive shift register, the CPU has enough time to service an interrupt and to accept the data character assembled by the Z80-SIO. The receiver also has three buffers that store error flags for each data character in the receive buffer. These error flags are loaded at the same time as the data characters.

After a character is received, it is checked for the following error conditions:

- When parity is enabled, the Parity Error bit (RR1, D_4) is set whenever the parity bit of the character does not match with the programmed parity. Once this bit is set, it remains set until the Error Reset Command (WR0) is given.

- The Framing Error bit (RR1, D_6) is set if the character is assembled without any stop bits (that is, a Low level detected for a stop bit). Unlike the Parity Error bit, this bit is set (and not latched) only for the character on which it occurred. Detection of framing error adds an additional one-half of a bit time to the character time so the framing error is not interpreted as a new start bit.

- If the CPU fails to read a data character while more than three characters have been received, the Receive Overrun bit (RR1, D_5) is set. When this occurs, the fourth character assembled replaces the third character in the receive buffers. With this arrangement, only the character that has been written over is flagged with the Receive Overrun Error bit. Like Parity Error, this bit can only be reset by the Error Reset command from the CPU. Both the Framing Error and Receive Overrun Error cause an interrupt with the interrupt vector indicating a Special Receive condition (if Status Affects Vector is selected).

Since the Parity Error and Receive Overrun Error flags are latched, the error status that is read reflects an error in the current word in the receive buffer plus any Parity or Overrun Errors received since the last Error Reset command. To keep correspondence between the state of the error buffers and the contents of the receive data buffers, the error status register must be read before the data. This is easily accomplished if vectored

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
WR3	00 = Rx 5 BITS/CHAR 10 = Rx 6 BITS/CHAR 01 = Rx 7 BITS/CHAR 11 = Rx 8 BITS/CHAR		AUTO ENABLES	0	0	0	0	Rx ENABLE
WR4	00 = ×1 CLOCK MODE 01 = ×16 CLOCK MODE 10 = ×32 CLOCK MODE 11 = ×64 CLOCK MODE		0	0	00 = NOT USED 01 = 1 STOP BIT/CHAR 10 = 1½ STOP BITS/CHAR 11 = 2 STOP BITS/CHAR		EVEN/$\overline{ODD}$ PARITY	PARITY ENABLE
WR5	DTR	00 = Tx 5 BITS (OR LESS)/CHAR 10 = Tx 6 BITS/CHAR 01 = Tx 7 BITS/CHAR 11 = Tx 8 BITS/CHAR		SEND BREAK	Tx ENABLE	0	RTS	0

Table 2. Contents of Write Registers 3, 4 and 5 in Asynchronous Modes

FUNCTION	TYPICAL PROGRAM STEPS		COMMENTS
	REGISTER:	*INFORMATION LOADED:*	
	WR0	CHANNEL RESET	Reset SIO
	WR0	POINTER 2	
	WR2	INTERRUPT VECTOR	Channel B only
	WR0	POINTER 4, RESET EXTERNAL STATUS INTERRUPT	
	WR4	ASYNCHRONOUS MODE, PARITY INFORMATION, STOP BITS INFORMATION, CLOCK RATE INFORMATION	Issue parameters
INITIALIZE	WR0	POINTER 3	
	WR3	RECEIVE ENABLE, AUTO ENABLES, RECEIVE CHARACTER LENGTH	
	WR0	POINTER 5	
	WR5	REQUEST TO SEND, TRANSMIT ENABLE, TRANSMIT CHARACTER LENGTH, DATA TERMINAL READY	Receive and Transmit both fully initialized. Auto Enables will enable Transmitter if $\overline{CTS}$ is active and Receiver if $\overline{DCD}$ is active.
	WR0	POINTER 1, RESET EXTERNAL STATUS INTERRUPT	
	WR1	TRANSMIT INTERRUPT ENABLE, STATUS AFFECTS VECTOR, INTERRUPT ON ALL RECEIVE CHARACTERS, DISABLE WAIT/READY FUNCTION, EXTERNAL INTERRUPT ENABLE	Transmit/Receive interrupt mode selected. External interrupt monitors the status of the $\overline{CTS}$, $\overline{DCD}$ and $\overline{SYNC}$ inputs and detects the Break sequence. Status Affects Vector in Channel B only.
	TRANSFER FIRST DATA BYTE TO SIO		This data byte must be transferred or no transmit interrupts will occur.
IDLE MODE	EXECUTE HALT INSTRUCTION OR SOME OTHER PROGRAM		Program is waiting for an interrupt from the SIO.
DATA TRANSFER AND ERROR MONITORING	Z80 INTERRUPT ACKNOWLEDGE CYCLE TRANSFERS RR2 TO CPU		When the interrupt occurs, the interrupt vector is modified by: 1. Receive Character Available; 2. Transmit Buffer Empty; 3. External/Status change; and 4. Special Receive condition.
	IF A CHARACTER IS RECEIVED: • TRANSFER DATA CHARACTER TO CPU • UPDATE POINTERS AND PARAMETERS • RETURN FROM INTERRUPT		Program control is transferred to one of the eight interrupt service routines.
	IF TRANSMITTER BUFFER IS EMPTY: • TRANSFER DATA CHARACTER TO SIO • UPDATE POINTERS AND PARAMETERS • RETURN FROM INTERRUPT		If used with processors other than the Z80, the modified interrupt vector (RR2) should be returned to the CPU in the Interrupt Acknowledge sequence.
	IF EXTERNAL STATUS CHANGES: • TRANSFER RR0 TO CPU • PERFORM ERROR ROUTINES (INCLUDE BREAK DETECTION) • RETURN FROM INTERRUPT		
	IF SPECIAL RECEIVE CONDITION OCCURS: • TRANSFER RR1 TO CPU • DO SPECIAL ERROR (E.G. FRAMING ERROR) ROUTINE • RETURN FROM INTERRUPT		
	REDEFINE RECEIVE/TRANSMIT INTERRUPT MODES		When transmit or receive data transfer is complete.
TERMINATION	DISABLE TRANSMIT/RECEIVE MODES		
	UPDATE MODEM CONTROL OUTPUTS (E.G. RTS OFF)		In Transmit, the All Sent status bit indicates transmission is complete.

Table 3. Asynchronous Mode

interrupts are used, because a special interrupt vector is generated for these conditions.

While the External/Status interrupt is enabled, break detection causes an interrupt and the Break Detected status bit (RR0, D_7) is set. The Break Detected interrupt should be handled by issuing the Reset External/Status Interrupt command to the Z80-SIO in response to the first Break Detected interrupt that has a Break status of 1 (RR0, D_7). The Z80-SIO monitors the Receive Data input and waits for the Break sequence to terminate, at which point the Z80-SIO interrupts the CPU with the Break status set to 0. The CPU must again issue the Reset External/Status Interrupt command in its interrupt service routine to reinitialize the break detection logic.

The External/Status interrupt also monitors the status of $\overline{\text{DCD}}$. If the $\overline{\text{DCD}}$ pin becomes inactive for a period greater than the minimum specified pulse width, an interrupt is generated with the DCD status bit (RR0, D_3) set to 1. Note that the $\overline{\text{DCD}}$ input is inverted in the RR0 status register.

If the status is read after the data, the error data for the next word is also included if it has been stacked in the buffer. If operations are performed rapidly enough so the next character is not yet received, the status register remains valid. An exception occurs when the Interrupt On First Character Only mode is selected. A special interrupt in this mode holds the error data and the character itself (even if read from the buffer) until the Error Reset command is issued. This prevents further data from becoming available in the receiver until the Reset command is issued, and allows CPU intervention on the character with the error even if DMA or block transfer techniques are being used.

If Interrupt On Every Character is selected, the interrupt vector is different if there is an error status in RR1. If a Receiver Overrun occurs, the most recent character received is loaded into the buffer; the character preceding it is lost. When the character that has been written over the other characters is read, the Receive Overrun bit is set and the Special Receive Condition vector is returned if Status Affects Vector is enabled.

In a polled environment, the Receive Character Available bit (RR0, D_0) must be monitored so the Z80-CPU can know when to read a character. This bit is automatically reset when the receive buffers are read. To prevent overwriting data in polled operations, the transmit buffer status must be checked before writing into the transmitter. The Transmit Buffer Empty bit is set to 1 whenever the transmit buffer is empty.

Synchronous Operation

Before describing synchronous transmission and reception, the three types of character synchronization—Monosync, Bisync and External Sync—require some explanation. These modes use the ×1 clock for both Transmit and Receive operations. Data is sampled on the rising edge of the Receive Clock input ($\overline{RxC}$). Transmitter data transitions occur on the falling edge of the Transmit Clock input ($\overline{TxC}$).

The differences between Monosync, Bisync and External Sync are in the manner in which initial character synchronization is achieved. The mode of operation must be selected before sync characters are loaded, because the registers are used differently in the various modes. Figure 7 shows the formats for all three of these synchronous modes.

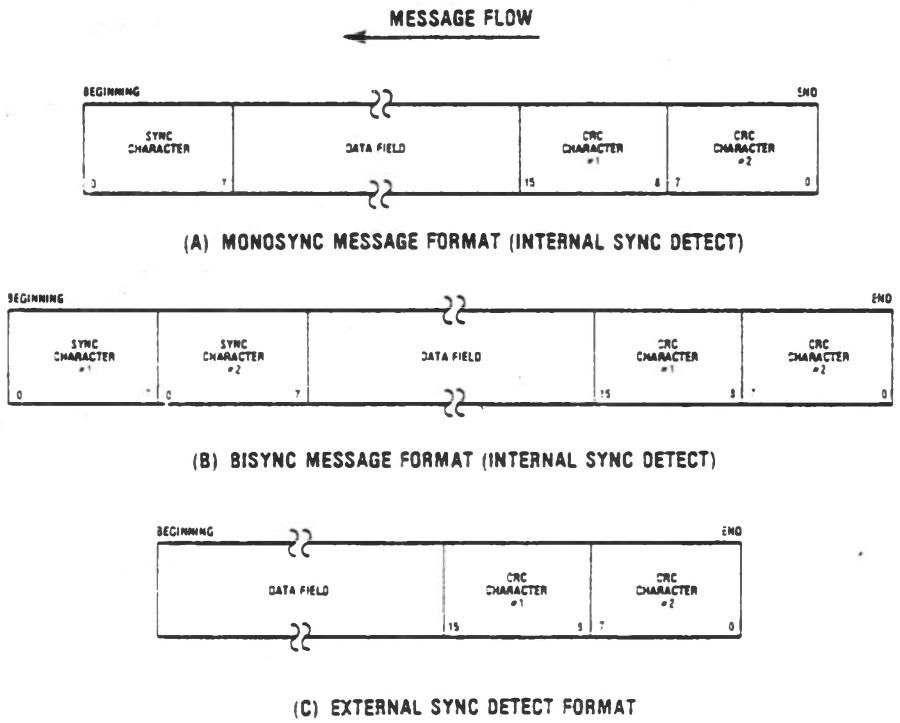
Monosync. In a Receive operation, matching a single sync character (8-bit sync mode) with the programmed sync character stored in WR7 implies character synchronization and enables data transfer.

Bisync. Matching two contiguous sync characters (16-bit sync mode) with the programmed sync characters stored in WR6 and WR7 implies character synchronization. In both the Monosync and Bisync modes, $\overline{SYNC}$ is used as an output, and is active for the part of the receive clock that detects the sync character.

External Sync. In this mode, character synchronization is established externally; $\overline{SYNC}$ is an input that indicates external character synchronization has been achieved. After the sync pattern is detected, the external logic must wait for two full Receive Clock cycles to activate the $\overline{SYNC}$ input. The SYNC input must be held Low until character synchronization is lost. Character assembly begins on the rising edge of $\overline{RxC}$ that precedes the falling edge of $\overline{SYNC}$.

In all cases after a reset, the receiver is in the Hunt phase, during which the Z80-SIO looks for character synchronization. The hunt can begin only when the receiver is enabled, and data transfer can begin only when character synchronization has been achieved. If character synchronization is lost, the Hunt phase can be re-entered by writing a control word with the Enter Hunt Phase bit set (WR3, D_4). In the Transmit mode, the transmitter always sends the programmed number of sync bits (8 or 16). In the Monosync mode, the transmitter transmits from WR6; the receiver compares against WR7.

In the Monosync, Bisync and External Sync modes, assembly of received data continues until the Z80-SIO is reset, or until the receiver is disabled (by command or by $\overline{DCD}$ in the Auto Enables mode), or until the CPU sets the Enter Hunt Phase bit.



(A) MONOSYNC MESSAGE FORMAT (INTERNAL SYNC DETECT)

(B) BISYNC MESSAGE FORMAT (INTERNAL SYNC DETECT)

(C) EXTERNAL SYNC DETECT FORMAT

Figure 7. Synchronous Formats

After initial synchronization has been achieved, the operation of the Monosync, Bisync and External Sync modes is quite similar. Any differences are specified in the following text.

Table 4 shows how WR3, WR4 and WR5 are used in synchronous receive and transmit operations. WR0 points to other registers and issues various commands, WR1 defines the interrupt modes, WR2 stores the interrupt vector, and WR6 and WR7 store sync characters. Table 5 illustrates the typical program steps that implement a half-duplex Bisync transmit operation.

Synchronous Transmit

INITIALIZATION

The system program must initialize the transmitter with the following parameters: odd or even parity, $\times 1$ clock mode, 8- or 16-bit sync character(s), CRC polynomial, Transmitter Enables, Request To Send, Data Terminal Ready, interrupt modes and transmit character length. WR4 parameters must be issued before WR1, WR3, WR5, WR6 and WR7 parameters or commands.

One of two polynomials—CRC-16 ($X^{16}+X^{15}+X^2+1$) or SDLC ($X^{16}+X^{12}+X^5+1$)—may be used with synchronous modes. In either case (SDLC mode not selected), the CRC generator and checker are reset to all 0's. In the transmit initialization process, the CRC generator is initialized by setting the Reset Transmit CRC Generator command bits (WR0). Both the transmitter and the receiver use the same polynomial.

Transmit Interrupt Enable or Wait/Ready Enable can be selected to transfer the data. The External/Status interrupt mode is used to monitor the status of the $\overline{\text{CLEAR TO SEND}}$ input as well as the Transmit Underrun/EOM latch. Optionally, the Auto Enables feature can be used to enable the transmitter when $\overline{\text{CTS}}$ is active. The first data transfer to the Z80-SIO can begin when the External/Status interrupt occurs (CTS status bit set) or immediately following the Transmit Enable command (if the Auto Enables modes is set).

Transmit data is held marking after reset or if the transmitter is not enabled. Break may be programmed to generate a spacing line that begins as soon as the Send Break bit is set. With the transmitter fully initialized and enabled, the default condition is continuous transmission of the 8- or 16-bit sync character.

DATA TRANSFER AND STATUS MONITORING

In this phase, there are several combinations of interrupts and Wait/Ready.

Data Transfer Using Interrupts. If the Transmit Interrupt Enable bit (WR1, D_1) is set, an interrupt is generated each time the transmit buffer becomes empty. The interrupt can be satisfied either by writing another character into the transmitter or by resetting the Transmitter Interrupt Pending latch with a Reset Transmitter Pending command (WR0, CMD_5). If the interrupt is satisfied with this command and nothing more is written into the transmitter, there can be no further Transmit Buffer Empty interrupts, because it is the process of the buffer becoming empty that causes the interrupts and the buffer cannot become empty when it is already empty. This situation does cause a Transmit Underrun condition, which is explained in the "Bisync Transmit Underrun" section.

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
WR3	00 = Rx 5 BITS/CHAR 10 = Rx 6 BITS/CHAR 01 = Rx 7 BITS/CHAR 11 = Rx 8 BITS/CHAR		AUTO ENABLES	ENTER HUNT MODE	Rx CRC ENABLE	0	SYNC CHAR LOAD INHIBIT	RX ENABLE
WR4	0	0	00 = 8-BIT SYNC CHAR 01 = 16-BIT SYNC CHAR 10 = SDLC MODE 11 = EXT SYNC MODE		0	0 SELECTS SYNC MODES	EVEN/$\overline{\text{ODD}}$ PARITY	PARITY ENABLE
WR5	DTR	00 = Tx 5 BITS (OR LESS)/CHAR 10 = Tx 6 BITS/CHAR 01 = Tx 7 BITS/CHAR 11 = Tx 8 BITS/CHAR		SEND BREAK	Tx ENABLE	1 SELECTS CRC-16	RTS	Tx CRC ENABLE

Table 4. Contents of Write Registers 3, 4 and 5 in Synchronous Modes

Data Transfer Using WAIT/READY. To the CPU, the activation of $\overline{\text{WAIT}}$ indicates that the Z80-SIO is not ready to accept data and that the CPU must extend the output cycle. To a DMA controller, READY indicates that the transmit buffer is empty and that the Z80-SIO is ready to accept the next data character. If the data character is not loaded into the Z80-SIO by the time the transmit shift register is empty, the Z80-SIO enters the Transmit Underrun condition.

Bisync Transmit Underrun. In Bisync protocol, filler characters are inserted to maintain synchronization when the transmitter has no data to send (Transmit Underrun condition). The Z80-SIO has two programmable options for solving this situation: it can insert sync characters, or it can send the CRC characters generated so far, followed by sync characters.

These options are under the control of the Reset Transmit Underrun/EOM command in WR0. Following a channel or channel reset, the Transmit Underrun/EOM status bit (RR0, D_6) is in a set condition and allows the insertion of sync characters when there is no data to send. CRC is not calculated on the automatically inserted sync characters. When the CPU detects the end of message, a Reset Transmit Underrun/EOM command can be issued. This allows CRC to be sent when the transmitter has no data. In this case, the Z80-SIO sends CRC, followed by sync characters, to terminate the message.

There is no restriction as to when in the message the Transmit Underrun/EOM bit can be reset. If Reset is issued after the first data character has been loaded the 16-bit CRC is sent and followed by sync characters the first time the transmitter has no data to send. Because of the Transmit Underrun condition, an External/Status interrupt is generated whenever the Transmit Underrun/EOM bit becomes set.

In the case of sync insertion, an interrupt is generated only after the first automatically inserted sync character has been loaded. The status indicates the Transmit Underrun/EOM bit and the Transmit Buffer Empty bit are set.

In the case of CRC insertion, the Transmit Underrun/EOM bit is set and the Transmit Buffer Empty bit is reset while CRC is being sent. When CRC has been completely sent, the Transmit Buffer Empty status bit is set and an interrupt is generated to indicate to the CPU that another message can begin (this interrupt occurs because CRC has been sent and sync has been loaded). If no more messages are to be sent, the program can terminate transmission by resetting RTS, and disabling the transmitter (WR5, D_3).

Pad characters may be sent by setting the Z80-SIO to 8 bits/transmit character and writing FF to the transmitter while CRC is being sent. Alternatively, the sync characters can be redefined as pad characters during this time. The following example is included to clarify this point.

The Z80-SIO interrupts with the Transmit Buffer Empty bit set.

The CPU recognizes that the last character (ETX) of the message has already been sent to the Z80-SIO by examining the internal program status.

To force the Z80-SIO to send CRC, the CPU issues the Reset Transmit Underrun/EOM Latch command (WR0) and satisfies the interrupt with the Reset Transmit Interrupt Pending command. (This command prevents the Z80-SIO from requesting more data.) Because of the transmit underrun caused by this command, the Z80-SIO starts sending CRC. The Z80-SIO also causes an External/Status interrupt with the Transmit Underrun/EOM latch set.

The CPU satisfies this interrupt by loading pad characters into the transmit buffer and issuing the Reset External/Status Interrupt command.

With this sequence, CRC is followed by a pad character instead of a sync character. Note that the Z80-SIO will interrupt with a Transmit Buffer Empty interrupt when CRC is completely sent and that the pad character is loaded into the transmit shift register.

From this point on the CPU can send more pad characters or sync characters.

Bisync CRC Generation. Setting the Transmit CRC enable bit (WR5, D_0) initiates CRC accumulation when the program sends the first data character to the Z80-SIO. Although the Z80-SIO automatically transmits up to two sync characters (16-bit sync), it is wise to send a few more sync characters ahead of the message (before enabling Transmit CRC) to ensure synchronization at the receiving end.

The transmit CRC Enable bit can be changed on the fly any time in the message to include or exclude a particular data character from CRC accumulation. The Transmit CRC Enable bit should be in the desired state when the data character is loaded from the transmit data buffer into the transmit shift register. To ensure this bit is in the proper state, the Transmit CRC Enable bit must be issued before sending the data character to the Z80-SIO.

Transmit Transparent Mode. Transparent mode (Bisync protocol) operation is made possible by the ability to change Transmit CRC Enable on the fly and by the additional capability of inserting 16-bit sync characters. Exclusion of DLE characters from CRC calculation can be achieved by disabling CRC calculation immediately preceding the DLE character transfer to the Z80-SIO.

In the case of a Transmit Underrun condition in the Transparent mode, a pair of DLE-SYN characters are sent. The Z80-SIO can be programmed to send the DLE-SYN sequence by loading a DLE character into WR6 and a sync character into WR7.

Transmit Termination. The Z80-SIO is equipped with a special termination feature that maintains data integrity and validity. If the transmitter is disabled while a data or sync character is being sent, that character is sent as usual, but is followed by a marking line rather than CRC or sync characters. When the transmitter is disabled, a

FUNCTION	TYPICAL PROGRAM STEPS		COMMENTS
	REGISTER:	*INFORMATION LOADED:*	
	WR0	CHANNEL RESET. RESET TRANSMIT CRC GENERATOR	Reset SIO, initilize CRC generator.
	WR0	POINTER 2	
	WR2	INTERRUPT VECTOR	Channel B only
	WR0	POINTER 3	
	WR3	AUTO ENABLES	Transmission begins only after $\overline{CTS}$ is detected.
	WR0	POINTER 4	
	WR4	PARITY INFORMATION, SYNC MODES INFORMATION, ×1 CLOCK MODE	Issue transmit parameters.
	WR0	POINTER 6	
	WR6	SYNC CHARACTER 1	
	WR0	POINTER 7, RESET EXTERNAL/STATUS INTERRUPTS	
INITIALIZE	WR7	SYNC CHARACTER 2	
	WR0	POINTER 1, RESET EXTERNAL/STATUS INTERRUPTS	
	WR1	STATUS AFFECTS VECTOR, EXTERNAL INTERRUPT ENABLE, TRANSMIT INTERRUPT ENABLE OR WAIT/READY MODE ENABLE	External Interrupt mode monitors the status of $\overline{CTS}$ and $\overline{DCD}$ input pins as well as the status of Tx Underrun/EOM latch. Transmit Interrupt Enable interrupts when the Transmit buffer becomes empty; the Wait/Ready mode can be used to transfer data using DMA or CPU Block Transfer.
	WR0	POINTER 5	Status Affects Vector (Channel B only).
	WR5	REQUEST TO SEND, TRANSMIT ENABLE, BISYNC CRC, TRANSMIT CHARACTER LENGTH	Transmit CRC Enable should be set when first non-sync data is sent to Z80-SIO.
	FIRST SYNC BYTE TO SIO		Need several sync characters in the beginning of message. Transmitter is fully initialized.
IDLE MODE	EXECUTE HALT INSTRUCTION OR SOME OTHER PROGRAM		Waiting for interrupt or Wait/Ready output to transfer data.
DATA TRANSFER AND STATUS MONITORING	*WHEN INTERRUPT (WAIT/READY) OCCURS:* • INCLUDE/EXECLUDE DATA BYTE FROM CRC ACCUMULATION (IN SIO). • TRANSFER DATA BYTE FROM CPU (OR MEMORY) TO SIO. • DETECT AND SET APPROPRIATE FLAGS FOR CONTROL CHARACTERS (IN CPU). • RESET Tx UNDERRUN/EOM LATCH (WR0) IF LAST CHARACTER OF MESSAGE IS DETECTED. • UPDATE POINTERS AND PARAMETERS (CPU). • RETURN FROM INTERRUPT.		Interrupt occurs (Wait/Ready becomes active) when first data byte is being sent. Wait mode allows CPU block transfer from memory to SIO; Ready mode allows DMA block transfer from memory to SIO. The DMA chip can be programmed to capture special control characters (by examining only the bits that specify ASCII or EBCDIC control characters), and interrupt CPU.
	IF ERROR CONDITION OR STATUS CHANGE OCCURS: • TRANSFER RR0 TO CPU. • EXECUTE ERROR ROUTINE. • RETURN FROM INTERRUPT.		Tx Underrun/EOM indicates either transmit underrun (sync character being sent) or end of message (CRC-16 being sent).
TERMINATION	REDEFINE INTERRUPT MODES. UPDATE MODEM CONTROL OUTPUTS (E.G., TURN OFF RTS). DISABLE TRANSMIT MODE		Program should gracefully terminate message.

Table 5. Bisync Transmit Mode

character in the buffer remains in the buffer. If the transmitter is disabled while CRC is being sent, the 16-bit transmission is completed, but sync is sent instead of CRC.

A programmed break is effective as soon as it is written into the control register; characters in the transmit buffer and shift register are lost.

In all modes, characters are sent with the least significant bits first. This requires right-hand justification of transmitted data if the word length is less than eight bits. If the word length is five bits or less, the special technique described in the Write Register 5 discussion (Z80-SIO Programming section) must be used for the data format. The states of any unused bits in a data character are irrelevant, except when in the Five Bits Or Less mode.

If the External/Status Interrupt Enable bit is set, transmitter conditions such as "starting to send CRC characters," "starting to send sync characters," and $\overline{CTS}$ changing state cause interrupts that have a unique vector if Status Affects Vector is set. This interrupt mode may be used during block transfers.

All interrupts may be disabled for operation in a Polled mode, or to avoid interrupts at inappropriate times during the execution of a program.

Synchronous Receive

INITIALIZATION

The system program initiates the Synchronous Receive operation with the following parameters: odd or even parity, 8- or 16-bit sync characters, ×1 clock mode, CRC polynomial, receive character length, etc. Sync characters must be loaded into registers WR6 and WR7. The receivers can be enabled only after all receive parameters are set. WR4 parameters must be issued before WR1, WR3, WR5, WR6 and WR7 parameters or commands.

After this is done, the receiver is in the Hunt phase. It remains in this phase until character synchronization is achieved. Note that, under program control, all the leading sync characters of the message can be inhibited from loading the receive buffers by setting the Sync Character Load Inhibit bit in WR3.

DATA TRANSFER AND STATUS MONITORING

After character synchronization is achieved, the assembled characters are transferred to the receive data FIFO. The following four interrupt modes are available to transfer the data and its associated status to the CPU.

No Interrupts Enabled. This mode is used for a purely polled operation or for off-line conditions.

Interrupt On First Character Only. This mode is normally used to start a polling loop or a Block Transfer instruction using $\overline{WAIT/READY}$ to synchronize the CPU or the DMA device to the incoming data rate. In this mode, the Z80-SIO interrupts on the first character and thereafter interrupts only if Special Receive conditions are detected. The mode is reinitialized with the Enable Interrupt On Next Receive Character command to allow the next character received to generate an interrupt. Parity errors do not cause interrupts in this mode, but End Of Frame (SDLC mode) and Receive Overrun do.

If External/Status interrupts are enabled, they may interrupt any time $\overline{DCD}$ changes state.

Interrupt On Every Character. Whenever a character enters the receive buffer, an interrupt is generated. Error and Special Receive conditions generate a special vector if Status Affects Vector is selected. Optionally, a Parity Error may be directed not to generate the special interrupt vector.

Special Receive Condition Interrupts. The Special Receive Condition interrupt can occur only if either the Receive Interrupt On First Character Only or Interrupt On Every Receive Character modes is also set. The Special Receive Condition interrupt is caused by the Receive Overrun error condition. Since the Receive Overrun and Parity error status bits are latched, the error status—when read—reflects an error in the current word in the receive buffer in addition to any Parity or Overrun errors received since the last Error Reset command. These status bits can only be reset by the Error reset command.

CRC Error Checking and Termination. A CRC error check on the receive message can be performed on a per character basis under program control. The Receive CRC Enable bit (WR3, D_3) must be set/reset by the program before the next character is transferred from the receive shift register into the receive buffer register. This ensures proper inclusion or exclusion of data characters in the CRC check.

To allow the CPU ample time to enable or disable the CRC check on a particular character, the Z80-SIO calculates CRC eight bit times after the character has been transferred to the receive buffer. If CRC is enabled before the next character is transferred, CRC is calculated on the transferred character. If CRC is disabled before the time of the next transfer, calculation proceeds on the word in progress, but the word just transferred to the buffer is not included. When these requirements are satisfied, the 3-byte receive data buffer is, in effect, unusable in Bisync operation. CRC may be enabled and disabled as many times as necessary for a given calculation.

In the Monosync, Bisync and External Sync modes, the CRC/Framing Error bit (RR1, D_6) contains the comparison result of the CRC checker 16 bit times (eight bits delay and eight shifts for CRC) after the character has been transferred from the receive shift register to the buffer. The result should be zero, indicating an error-

free transmission. (Note that the result is valid only at the end of CRC calculation. If the result is examined before this time, it usually indicates an error.) The comparison is made with each transfer and is valid only as long as the character remains in the receive FIFO.

Following is an example of the CRC checking operation when four characters (A, B, C and D) are received in that order.

Character A loaded into buffer
Character B loaded into buffer

If CRC is disabled before C is in the buffer, CRC is not calculated on B.

Character C loaded into buffer

After C is loaded, the CRC/Framing Error bit shows the result of the comparison through character A.

Character D loaded into buffer

After D is in the buffer, the CRC Error bit shows the result of the comparison through character B whether or not B was included in the CRC calculations.

Due to the serial nature of CRC calculation, the Receive Clock ($\overline{RxC}$) must cycle 16 times (8-bit delay plus 8-bit CRC shift) after the second CRC character has been loaded into the receive buffer, or 20 times (the previous 16 plus 3-bit buffer delay and 1-bit input delay) after the last bit is at the RxD input, before CRC calculation is complete. A faster external clock can be gated into the Receive Clock input to supply the required 16 cycles. The Transmit and Receive Data Path diagram (Figure 4) illustrates the various points of delay in the CRC path.

The typical program steps that implement a half-duplex Bisync Receive mode are illustrated in Table 6. The complete set of command and status bit definitions are explained under "Z80-SIO Programming."

FUNCTION		TYPICAL PROGRAM STEPS	COMMENTS
	REGISTER:	*INFORMATION LOADED*	
	WR0	CHANNEL RESET, RESET RECEIVE CRC CHECKER	Reset SIO; initialize Receive CRC checker.
	WR0	POINTER 2	
	WR2	INTERRUPT VECTOR	Channel B only
	WR0	POINTER 4	
	WR4	PARITY INFORMATION, SYNC MODES INFORMATION, ×1 CLOCK MODE	Issue receive parameters.
	WR0	POINTER 5, RESET EXTERNAL STATUS INTERRUPT	
	WR5	BISYNC CRC-16, DATA TERMINAL READY	
	WR0	POINTER 3	
INITIALIZE	WR3	SYNC CHARACTER LOAD INHIBIT, RECEIVE CRC ENABLE; ENTER HUNT MODE, AUTO ENABLES, RECEIVE CHARACTER LENGTH	Sync character load inhibit strips all the leading sync characters at the beginning of the message. Auto Enables enables the receiver to accept data only after the $\overline{DCD}$ input is active.
	WR0	POINTER 6	
	WR6	SYNC CHARACTER 1	
	WR0	POINTER 7	
	WR7	SYNC CHARACTER 2	
	WR0	POINTER 1, RESET EXTERNAL/STATUS INTERRUPT	
	WR1	STATUS AFFECTS VECTOR, EXTERNAL INTERRUPT ENABLE, RECEIVE INTERRUPT ON FIRST CHARACTER ONLY	In this interrupt mode, only the first non-sync data character is transferred to the CPU. All subsequent data is transferred on a DMA basis; however Special Receive Condition interrupts will interrupt the CPU. Status Affects Vector used in Channel B only.

Table 6. Bisync Receive Mode

FUNCTION	TYPICAL PROGRAM STEPS	COMMENTS
INITIALIZE (CONTINUED)	WR0 POINTER 3, ENABLE INTERRUPT ON NEXT RECEIVE CHARACTER	Resetting this interrupt mode provides simple program loopback entry for the next transaction.
	WR3 RECEIVE ENABLE, SYNC CHARACTER LOAD INHIBIT, ENTER HUNT MODE, AUTO ENABLE, RECEIVE WORD LENGTH	WR3 is reissued to enable receiver. Receive CRC Enable must be set after receiving SOH or STX character.
IDLE MODE	EXECUTE HALT INSTRUCTION OR SOME OTHER PROGRAM	Receive mode is fully initialized and the system is waiting for interrupt on first character.
DATA TRANSFER AND STATUS MONITORING	*WHEN INTERRUPT ON FIRST CHARACTER OCCURS, THE CPU DOES THE FOLLOWING:* • TRANSFERS DATA BYTE TO CPU • DETECTS AND SETS APPROPRIATE FLAGS FOR CONTROL CHARACTERS (IN CPU) • INCLUDES/EXCLUDES DATA BYTE IN CRC CHECKER • UPDATES POINTERS AND OTHER PARAMETERS • ENABLES WAIT/READY FOR DMA OPERATION • ENABLES DMA CONTROLLER • RETURNS FROM INTERRUPT	During the Hunt mode, the SIO detects two contiguous characters to establish synchronization. The CPU establishes the DMA mode and all subsequent data characters are transferred by the DMA controller. The controller is also programmed to capture special characters (by examining only the bits that specify ASCII or EBCDIC control characters) and interrupt the CPU upon detection. In response, the CPU examines the status or control characters and takes appropriate action (e.g. CRC Enable Update).
	WHEN WAIT/READY BECOMES ACTIVE, THE DMA CONTROLLER DOES THE FOLLOWING: • TRANSFERS DATA BYTE TO MEMORY • INTERRUPTS CPU IF A SPECIAL CHARACTER IS CAPTURED BY THE DMA CONTROLLER • INTERRUPTS THE CPU IF THE LAST CHARACTER OF THE MESSAGE IS DETECTED	
	FOR MESSAGE TERMINATION, THE CPU DOES THE FOLLOWING: • TRANSFERS RR1 TO THE CPU • SETS ACK/NAK REPLY FLAG BASED ON CRC RESULT • UPDATES POINTERS AND PARAMETERS • RETURNS FROM INTERRUPT	The SIO interrupts the CPU for error condition, and the error routine aborts the present message, clears the error condition, and repeats the operation.
TERMINATION	REDEFINE INTERRUPT MODES AND SYNC MODES UPDATE MODEM CONTROLS DISABLES RECEIVE MODE	

Table 6. Bisync Receive Mode (Continued)

SDLC (HDLC) Operation

The Z80-SIO is capable of handling both High-level Synchronous Data Link Control (HDLC) and IBM Synchronous Data Link Control (SDLC) protocols. In the following text, only SDLC is referred to because of the high degree of similarity between SDLC and HDLC.

The SDLC mode is considerably different than Synchronous Bisync protocol because it is bit oriented rather than character oriented and, therefore, can naturally handle transparent operation. Bit orientation makes SDLC a flexible protocol in terms of message length and bit patterns. The Z80-SIO has several built-in features to handle variable message length. Detailed information concerning SDLC protocol can be found in literature published on this subject, such as IBM document GA27-3093.

The SDLC message, called the frame (Figure 8), is opened and closed by flags that are similar to the sync characters in Bisync protocol. The Z80-SIO handles the transmission and recognition of the flag characters that mark the beginning and end of the frame. Note that the Z80-SIO can receive shared-zero flags, but cannot transmit them. The 8-bit address field of an SDLC frame contains the secondary station address. The Z80-SIO has an Address Search mode that recognizes the secondary station address so it can accept or reject the frame.

Since the control field of the SDLC frame is transparent to the Z80-SIO, it is simply transferred to the CPU. The Z80-SIO handles the Frame Check sequence in a manner that simplifies the program by incorporating features such as initializing the CRC generator to all 1's, resetting the CRC checker when the opening flag is detected in the Receive mode, and sending the Frame Check/Flag sequence in the Transmit mode. Controller hardware is simplified by automatic zero insertion and deletion logic contained in the Z80-SIO.

Table 7 shows the contents of WR3, WR4 and WR5 during SDLC Receive and Transmit modes. WR0 points to other registers and issues various commands. WR1 defines the interrupt modes. WR2 stores the interrupt vector. WR7 stores the flag character and WR6 the secondary address.

SDLC Transmit

INITIALIZATION

Like Synchronous operation, the SDLC Transmit mode must be initialized with the following parameters: SDLC mode, SDLC polynomial, Request To Send, Data Terminal Ready, transmit character length, transmit interrupt modes (or Wait/Ready function), Transmit Enable, Auto Enables and External/Status interrupt.

Selecting the SDLC mode and the SDLC polynomial enables the Z80-SIO to initialize the CRC Generator to all 1's. This is accomplished by issuing the Reset Transmit CRC Generator command (WR0). Refer to the Synchronous Operation section for more details on the interrupt modes.

After reset, or when the transmitter is not enabled, the Transmit Data output is held marking. Break may be programmed to generate a spacing line. With the transmitter fully initialized and enabled, continuous flags are transmitted on the Transmit Data output.

An abort sequence may be sent by issuing the Send Abort command (WR0, CMD_1). This causes at least eight, but less than fourteen, 1's to be sent before the line reverts to continuous flags. It is possible that the Abort sequence (eight 1's) could follow up to five continuous 1 bits (allowed by the zero insertion logic) and thus cause up to thirteen 1's to be sent. Any data being transmitted and any data in the transmit buffer is lost when an abort is issued.

When required, an extra 0 is automatically inserted when there are five contiguous 1's in the data stream. This does not apply to flags or aborts.

DATA TRANSFER AND STATUS MONITORING

There are several combinations of interrupts and the Wait/Ready function in the SLDC mode.

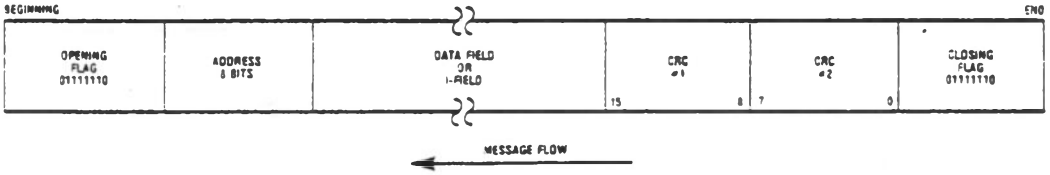


Figure 8. Transmit/Receive SDLC/HDLC Message Format

Data Transfer Using Interrupts. If the Transmit Interrupt Enable bit is set, an interrupt is generated each time the buffer becomes empty. The interrupt may be satisfied either by writing another character into the transmitter or by resetting the Transmit Interrupt Pending latch with a Reset Transmitter Pending command (WR0, CMD_5). If the interrupt is satisfied with this command and nothing more is written into the transmitter, there are no further transmitter interrupts. The result is a Transmit Underrun condition. When another character is written and sent out, the transmitter can again become empty and interrupt the CPU. Following the flags in an SDLC operation, the 8-bit address field, control field and information field may be sent to the Z80-SIO using the Transmit Interrupt mode. The Z80-SIO transmits the Frame Check sequence using the Transmit Underrun feature.

When the transmitter is first enabled, it is already empty and obviously cannot then become empty. Therefore, no Transmit Buffer Empty interrupts can occur until after the first data character is written.

When the transmitter is first enabled, it is already empty and cannot then become empty. Therefore, no Transmit Buffer Empty interrupts can occur until after the first data character is written.

Data Transfer Using Wait/Ready. If the Wait/Ready function has been selected, WAIT indicates to the CPU that the Z80-SIO is not ready to accept the data and the CPU must extend the I/O cycle. To a DMA controller, READY indicates that the transmitter buffer is empty and that the Z80-SIO is ready to accept the next character. If the data character is not loaded into the Z80-SIO by the time the transmit shift register is empty, the Z80-SIO enters the Transmit Underrun condition. Address, control and information fields may be transferred to the Z80-SIO with this mode using the Wait/Ready function. The Z80-SIO transmits the Frame Check sequence using the Transmit Underrun feature.

SDLC Transmit Underrun/End Of Message. SDLC-like protocols do not have provisions for fill characters within a message. The Z80-SIO therefore automatically terminates an SDLC frame when the transmit data buffer and output shift register have no more bits to send. It does this by first sending the two bytes of CRC and following these with one or more flags. This technique allows very high-speed transmissions under DMA or CPU block I/O control without requiring the CPU to respond quickly to the end of message situation.

The action that the Z80-SIO takes in the underrun situation depends on the state of the Transmit Underrun/EOM command. Following a reset, the Transmit Underrun/EOM status bit is in the set state and prevents the insertion of CRC characters during the time there is no data to send. Consequently, flag characters are sent. The Z80-SIO begins to send the frame as data is written into the transmit buffer. Between the time the first data byte is written and the end of the message, the Reset Transmit Underrun/EOM command must be issued. Thus the Transmit Underrun/EOM status bit is in the reset state at the end of the message (when underrun occurs), which automatically sends the CRC characters. The sending of CRC again sets the Transmit/Underrun/EOM status bit.

Although there is no restriction as to when the Transmit Underrun/EOM bit can be reset within a message, it is usually reset after the first data character (secondary address) is sent to the Z80-SIO. Resetting this bit allows CRC and flags to be sent when there is no data to send which gives additional time to the CPU for recognizing the fault and responding with an abort command. By resetting it early in the message, the entire message has the maximum amount of CPU response time in an unintentional transmit underrun situation.

	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
WR3	00 = Rx 5 BITS/CHAR 10 = Rx 6 BITS/CHAR 01 = Rx 7 BITS/CHAR 11 = Rx 8 BITS/CHAR		AUTO ENABLES	ENTER HUNT MODE (IF INCOMING DATA NOT NEEDED)	Rx CRC ENABLE	ADDRESS SEARCH MODE	0	Rx ENABLE
WR4	0	0	1 0 SELECTS SDLC MODE		0	0	0	0
WR5	DTR	00 = Tx 5 BITS (OR LESS)/CHAR 10 = Tx 6 BITS/CHAR 01 = Tx 7 BITS/CHAR 11 = Tx 8 BITS/CHAR		0	Tx ENABLE	0 SELECTS SDLC CRC	RTS	Tx CRC ENABLE

Table 7. Contents of Write Registers 3, 4 and 5 in SDLC Modes

When the External/Status interrupt is set and while CRC is being sent, the Transmit Underrun/EOM bit is set and the Transmit Buffer Empty bit is reset to indicate that the transmit register is full of CRC data. When CRC has been completely sent, the Transmit Buffer Empty status bit is set and an interrupt is generated to indicate to the CPU that another message can begin. This interrupt occurs because CRC has been sent and the flag has been loaded. If no more messages are to be sent, the program can terminate transmission by resetting $\overline{RTS}$, and disabling the transmitter.

In the SDLC mode, it is good practice to reset the Transmit Underrun/EOM status bit immediately after the first character is sent to the Z80-SIO. When the Transmit Underrun is detected, this ensures that the transmission time is filled by CRC characters, giving the CPU enough time to issue the Send Abort command. This also stops the flags from going on the line prematurely and eliminates the possibility of the receiver accepting the frame as valid data. The situation can happen because it is possible that—at the receiving end—the data pattern immediately preceding the automatic flag insertion could match the CRC checker, giving a false CRC check result. The External/Status interrupt is generated whenever the Transmit Underrun/EOM bit is set because of the Transmit Underrun condition.

The transmit underrun logic provides additional protection against premature flag insertion if the proper response is given to the Z80-SIO by the CPU interrupt service routine. The following example is given to clarify this point:

The Z80-SIO raises an interrupt with the Transmit Buffer Empty status bit set.

The CPU does not respond in time and causes a Transmit Underrun condition.

The Z80-SIO starts sending CRC characters (two bytes).

The CPU eventually satisfies the Transmit Buffer Empty interrupt with a data character that follows the CRC character being transmitted.

The Z80-SIO sets the External/Status interrupt with the Transmit Underrun/EOM status bit set.

The CPU recognizes the Transmit Underrun/EOM status and determines from its internal program status that the interrupt is not for "end of message".

The CPU immediately issues a Send Abort Command (WR0) to the Z80-SIO.

The Z80-SIO sends the Abort sequence by destroying whatever data (CRC, data or flag) is being sent.

This sequence illustrates that the CPU has a protection of 22 minimum and 30 maximum transmit clock cycles.

SDLC CRC Generation. The CRC generator must be reset to all 1's at the beginning of each frame before CRC accumulation can begin. Actual accumulation begins when the program sends the address field (eight bits) to the Z80-SIO. Although the Z80-SIO automatically transmits one flag character following the Transmit Enable, it may be wise to send a few more flag characters ahead of the message to ensure character synchronization at the receiving end. This can be done by externally timing out after enabling the transmitter and before loading the first character.

The Transmit CRC Enable (WR5, D_0) should be enabled prior to sending the address field. In the SDLC mode all the characters between the opening and closing flags are included in CRC accumulation, and the CRC generated in the Z80-SIO transmitter is inverted before it is sent on the line.

Transmit Termination. If the transmitter is disabled while a character is being sent, that character (data or flag) is sent in the normal fashion, but is followed by a marking line rather than CRC or flag characters.

A character in the buffer when the transmitter is disabled remains in the buffer; however, a programmed Abort sequence is effective as soon as it is written into the control register. Characters being transmitted, if any, are lost. In the case of CRC, the 16-bit transmission is completed if the transmitter is disabled; however, flags are sent in place of CRC.

In all modes, characters are sent with the least-significant bits first. This requires right-hand justification of data to be transmitted if the word length is less than eight bits. If the word length is five bits or less, the special technique described in the Write Register 5 section ("Z80-SIO Programming" chapter; "Write Registers" section) must be used.

Since the number of bits/character can be changed on the fly, the data field can be filled with any number of bits. When used in conjunction with the Receiver Residue codes, the Z80-SIO can receive a message that has a variable I-field and retransmit it exactly as received with no previous information about the character structure of the I-field (if any). A change in the number of bits does not affect the character in the process of being shifted out. Characters are sent with the number of bits programmed at the time that the character is loaded from the transmit buffer to the transmitter.

If the External/Status Interrupt Enable is set, transmitter conditions such as "starting to send CRC characters," "starting to send flag characters," and $\overline{CTS}$ changing state cause interrupts that have a unique vector if Status Affects Vector is set. All interrupts can be disabled for operation in a polled mode.

Table 8 shows the typical program steps that implement the half-duplex SDLC Transmit mode.

FUNCTION	TYPICAL PROGRAM STEPS		COMMENTS
	REGISTER:	*INFORMATION LOADED:*	
	WR0	CHANNEL RESET	Reset SIO.
	WR0	POINTER 2	
	WR2	INTERRUPT VECTOR	Channel B only
	WR0	POINTER 3	
	WR3	AUTO ENABLES	Transmitter sends data only after $\overline{CTS}$ is detected.
	WR0	POINTER 4, RESET EXTERNAL/STATUS INTERRUPTS	
	WR4	PARITY INFORMATION, SDLC MODE, ×1 CLOCK MODE	
	WR0	POINTER 1, RESET EXTERNAL/STATUS INTERRUPTS	
INITIALIZE	WR1	EXTERNAL INTERRUPT ENABLE, STATUS AFFECTS VECTOR, TRANSMIT INTERRUPT ENABLE *OR* WAIT/READY MODE ENABLE	The External Interrupt mode monitors the status of the $\overline{CTS}$ and $\overline{DCD}$ inputs, as well as the status of Tx Underrun/EOM latch. Transmit Interrupt interrupts when the Transmit buffer becomes empty; the Wait/Ready mode can be used to transfer data on a DMA or Block Transfer basis. The first interrupt occurs when $\overline{CTS}$ becomes active, at which point flags are transmitted by the Z80-SIO. The first data byte (address field) can be loaded in the Z80-SIO after this interrupt. Flags cannot be sent to the Z80-SIO as data. Status Affects Vector used in Channel B only.
	WR0	POINTER 5	
	WR5	TRANSMIT CRC ENABLE, REQUEST TO SEND, SDLC-CRC, TRANSMIT ENABLE, TRANSMIT WORD LENGTH, DATA TERMINAL READY	SDLC-CRC mode must be defined before initializing transmit CRC generator.
	WR0	RESET TRANSMIT CRC GENERATOR	Initialize CRC generator to all 1's.
IDLE MODE	EXECUTE HALT INSTRUCTION OR SOME OTHER PROGRAM		Waiting for Interrupt or Wait/Ready output to transfer data.
DATA TRANSFER AND STATUS MONITORING	*WHEN INTERRUPT (WAIT/READY) OCCURS, THE CPU DOES THE FOLLOWING:* • CHANGES TRANSMIT WORD LENGTH (IF NECESSARY) • TRANSFERS DATA BYTE FROM CPU (MEMORY) TO SIO • RESETS Tx UNDERRUN/EOM LATCH (WR0)		Flags are transmitted by the SIO as soon as Transmit Enable is set and $\overline{CTS}$ becomes active. The $\overline{CTS}$ status change is the first interrupt that occurs and is followed by transmit buffer empty for subsequent transfers.
	IF LAST CHARACTER OF THE I-FIELD IS SENT, THE SIO DOES THE FOLLOWING: • SENDS CRC • SENDS CLOSING FLAG • INTERRUPTS CPU WITH BUFFER EMPTY STATUS		Word length can be changed "on the fly" for variable I-field length. The data byte can contain address, control, or I-field information (never a flag). It is a good practice to reset Tx Underrun/EOM latch in the beginning of the message to avoid a false end-of-frame detection at the receiving end. This ensures that, when underrun occurs, CRC is transmitted and underrun interrupt (Tx Underrun/EOM latch active) occurs. Note that "Send Abort" can be issued to the SIO in response to any interrupting continuing to abort the transmission.
	CPU DOES THE FOLLOWING: • ISSUES RESET Tx INTERRUPT PENDING COMMAND TO THE Z80-SIO • UPDATES NS COUNT • REPEATS THE PROCESS FOR NEXT MESSAGE, ETC.		
	IF THE VECTOR INDICATES AN ERROR, THE CPU DOES THE FOLLOWING: • SENDS ABORT • EXECUTES ERROR ROUTINE • UPDATES PARAMETERS, MODES, ETC. • RETURNS FROM INTERRUPT		
TERMINATION	REDEFINE INTERRUPT MODES		Terminate gracefully.
	UPDATE MODEM CONTROL OUTPUTS		
	DISABLE TRANSMIT MODE		

Table 8. SDLC Transmit Mode

SDLC Receive

INITIALIZATION

The SDLC Receive mode is initialized by the system with the following parameters: SDLC mode, × 1 clock mode, SDLC polynomial, receive word length, etc. The flag characters must also be loaded in WR7 and the secondary address field loaded in WR6. The receiver is enabled only after all the receive parameters have been set. After all this has been done, the receiver is in the Hunt phase and remains in this phase until the first flag is received. While in the SDLC mode, the receiver never re-enters the Hunt phase, unless specifically instructed to do so by the program. The WR4 parameters must be issued prior to the WR1, WR3, WR5, WR6 and WR7 parameters.

Under program control, the receiver can enter the Address Search mode. If the Address Search bit (WR3, D-) set, a character following the flag (first non-flag character) is compared against the programmed address in WR6 and the hardwired global address (11111111). If the SDLC frame address field matches either address, data transfer begins.

Since the Z80-SIO is capable of matching only one address character, extended address field recognition must be done by the CPU. In this case, the Z80-SIO simply transfers the additional address bytes to the CPU as if they were data characters. If the CPU determines that the frame does not have the correct address field, it can set the Hunt bit, and the Z80-SIO suspends reception and searches for a new message headed by a flag. Since the control field of the frame is transparent to the Z80-SIO, it is transferred to the CPU as a data character. Extra zeros inserted in the data stream are automatically deleted; flags are not transferred to the CPU.

DATA TRANSFER AND STATUS MONITORING

After receipt of a valid flag, the assembled characters are transferred to the receive data FIFO. The following four interrupt modes are available to transfer this data and its associated status.

No Interrupts Enabled. This mode is used for purely polled operations or for off-line conditions.

Interrupt On First Character Only. This mode is normally used to start a software polling loop or a Block Transfer instruction using $\overline{\text{WAIT}}$/READY to synchronize the CPU or DMA device to the incoming data rate. In this mode, the Z80-SIO interrupts on the first character and thereafter only interrupts if Special Receive conditions are detected. The mode is reinitialized with the Enable Interrupt On Next Receive Character Command.

e first character received after this command is issued causes an interrupt. If External/Status interrupts are enabled, they may interrupt any time the $\overline{\text{DCD}}$ input changes state. Special Receive conditions such as End Of Frame and Receiver Overrun also cause interrupts. The End Of Frame interrupt can be used to exit the Block Transfer mode.

Interrupt On Every Character. An interrupt is generated whenever the receive FIFO contains a character. Error and Special Receive conditions generate a special vector if Status Affects Vector is selected.

Special Receive Condition Interrupts. The Special Receive Condition interrupt is not, as such, a separate interrupt mode. Before the Special Receive condition can cause an interrupt, either Interrupt On First Receive Character Only or Interrupt On Every Character must be selected. The Special Receive Condition interrupt is caused by a Receive Overrun or End Of Frame detection. Since the Receive Overrun status bit is latched, the error status read reflects an error in the current word in the receive buffer in addition to any errors received since the last Error Reset command. The Receive Overrun status bit can only be reset by the Error Reset command. The End Of Frame status bit indicates that a valid ending flag has been received and that the CRC Error and Residue codes are also valid.

Character length may be changed on the fly. If the address and control bytes are processed as 8-bit characters, the receiver may be switched to a shorter character length during the time that the first information character is being assembled. This change must be made fast enough so it is effective before the number of bits specified for the character length have been assembled. For example, if the change is to be from the 8-bit control field to a 7-bit information field, the change must be made *before* the first seven bits of the I-field are assembled.

SDLC Receive CRC Checking. Control of the receive CRC checker is automatic. It is reset by the leading flag and CRC is calculated up to the final flag. The byte that has the End Of Frame bit set is the byte that contains the result of the CRC check. If the CRC/Framing Error bit is not set, the CRC indicates a valid message. A special check sequence is used for the SDLC check because the transmitted CRC check is inverted. The final check must be 0001110100001111. The 2-byte CRC check characters must be read by the CPU and discarded because the Z80-SIO, while using them for CRC checking, treats them as ordinary data.

SDLC Receive Termination. If enabled, a special vector is generated when the closing flag is received. This signals that the byte with the End Of Frame bit set has been received. In addition to the results of the CRC check, RR1 has three bits of Residue code valid at this time. For those cases in which the number of bits in the I-field is not an integral multiple of the character length used, these bits indicate the boundary between the CRC check bits and the I-field bits. For a detailed description of the meaning of these bits, see the description of the residue codes in RR1 under "Z80-SIO Programming."

Any frame can be prematurely aborted by an Abort sequence. Aborts are detected if seven or more 1's occur

and cause an External/Status interrupt (if enabled) with the Break/Abort bit in RR0 set. After the Reset External/Status interrupts command has been issued a second interrupt occurs when the continuous 1's condition has been cleared. This can be used to distinguish between the Abort and Idle line conditions.

Unlike the synchronous mode, CRC calculation in SDLC does not have an 8-bit delay since all the characters are included in CRC calculation. When the second CRC character is loaded into the receive buffer, CRC calculation is complete.

Table 9 shows the typical steps required to implement a half-duplex SDLC receive mode. The complete set of command and status bit definitions is found in the next section.

FUNCTION	TYPICAL PROGRAM STEPS		COMMENTS
	REGISTER:	*INFORMATION LOADED:*	
	WR0	CHANNEL 2	Reset SIO
	WR0	POINTER 2	
	WR2	INTERRUPT VECTOR	Channel B only
	WR0	POINTER 4	
	WR4	PARITY INFORMATION, SYNC MODE, SDLC MODE, x1 CLOCK MODE	
	WR0	POINTER 5, RESET EXTERNAL/STATUS INTERRUPTS	
	WR5	SDLC-CRC, DATA TERMINAL READY	
	WR0	POINTER 3	
	WR3	RECEIVE CRC ENABLE, ENTER HUNT MODE, AUTO ENABLES, RECEIVE CHARACTER LENGTH, ADDRESS SEARCH MODE	'Auto Enables' enables the receiver to accept data only after $\overline{DCD}$ becomes active. Address Search Mode enables SIO to match the message address with the programmed address or the global address.
	WR0	POINTER 6	
INITIALIZE	WR6	SECONDARY ADDRESS FIELD	This address is matched against the message address in an SDLC poll operation.
	WR0	POINTER 7	
	WR7	SDLC FLAG 01111110	This flag detects the start and end of frame in an SDLC operation.
	WR0	POINTER 1, RESET EXTERNAL/STATUS INTERRUPTS	In this interrupt mode, only the Address Field (1 character only) is transferred to the CPU. All subsequent fields (Control, Information, etc.) are transferred on a DMA basis. Status Affects Vector in Channel B only.
	WR1	STATUS AFFECTS VECTOR, EXTERNAL INTERRUPT ENABLE, RECEIVE INTERRUPT ON FIRST CHARACTER ONLY.	
	WR0	POINTER 3, ENABLE INTERRUPT ON NEXT RECEIVE CHARACTER	Used to provide simple loop-back entry point for next transaction.
	WR3	RECEIVE ENABLE, RECEIVE CRC ENABLE, ENTER HUNT MODE, AUTO ENABLES, RECEIVER CHARACTER LENGTH, ADDRESS SEARCH MODE	WR3 reissued to enable receiver.
IDLE MODE	EXECUTE HALT INSTRUCTION OR SOME OTHER PROGRAM		SDLC Receive Mode is fully initialized and SIO is waiting for the opening flag followed by a matching address field to interrupt the CPU.

Table 9. SDLC Receive Mode

FUNCTION	TYPICAL PROGRAM STEPS	COMMENTS
	WHEN INTERRUPT ON FIRST CHARACTER OCCURS, THE CPU DOES THE FOLLOWING: • TRANSFERS DATA BYTE (ADDRESS BYTE) TO CPU • DETECTS AND SETS APPROPRIATE FLAG FOR EXTENDED ADDRESS FIELD • UPDATES POINTERS AND PARAMETERS • ENABLES DMA CONTROLLER • ENABLES WAIT/READY FUNCTION IN SIO • RETURNS FROM INTERRUPT	During the Hunt phase, the SIO interrupts when the programmed address matches the message address. The CPU establisnes the DMA mode and all subsequent data characters are transferred by the DMA controller to memory.
	WHEN THE READY OUTPUT BECOMES ACTIVE, THE DMA CONTROLLER DOES THE FOLLOWING: • TRANSFERS THE DATA BYTE TO MEMORY • UPDATES THE POINTERS	During the DMA operation, the SIO monitors the $\overline{DCD}$ input and the Abort sequence in the data stream to interrupt the CPU with External Status error. The Special Receive condition interrupt is caused by Receive Overrun error.
DATA TRANSFER AND STATUS MONITORING	*WHEN END OF FRAME INTERRUPT OCCURS, THE CPU DOES THE FOLLOWING:* • EXITS DMA MODE (DISABLES WAIT/READY) • TRANSFERS RR1 TO THE CPU • CHECKS THE CRC ERROR BIT STATUS AND RESIDUE CODES • UPDATES NR COUNT • ISSUES 'ERROR RESET' COMMAND TO SIO	Detection of End of Frame (Flag) causes interrupt and deactivates the Wait/Ready function. Residue codes indicate the bit structure of the last two bytes of the message, which were transferred to memory under DMA. 'Error Reset' is issued to clear the special condition.
	WHEN 'ABORT SEQUENCE DETECTED' INTERRUPT OCCURS. THE CPU DOES THE FOLLOWING: • TRANSFERS RR0 TO THE CPU • EXITS DMA MODE • ISSUES THE RESET EXTERNAL STATUS INTERRUPT COMMAND TO THE SIO • ENTERS THE IDLE MODE	Abort sequence is detected when seven or more 1's are found in the data stream. CPU is waiting for Abort Sequence to terminate. Termination clears the Break/Abort status bit and causes interrupt.
	WHEN THE SECOND ABORT SEQUENCE INTERRUPT OCCURS. THE CPU DOES THE FOLLOWING: • ISSUES THE RESET EXTERNAL STATUS INTERRUPT COMMAND TO THE SIO	At this point, the program proceeds to terminate this message.
TERMINATION	REDEFINE INTERRUPT MODES, SYNC MODE AND SDLC MODES DISABLE RECEIVE MODE	

Table 9. SDLC Receive Mode (Continued)

Z80-SIO Programming

To program the Z80-SIO, the system program first issues a series of commands that initialize the basic mode of operation and then other commands that qualify conditions within the selected mode. For example, the Asynchronous mode, character length, clock rate, number of stop bits, even or odd parity are first set, then the interrupt mode and, finally, receiver or transmitter enable. The WR4 parameters must be issued before any other parameters are issued in the initialization routine.

Both channels contain command registers that must be programmed via the system program prior to operation. The Channel Select input ($B/\overline{A}$) and the Control/Data input ($C/\overline{D}$) are the command structure addressing controls, and are normally controlled by the CPU address bus. Figure 14 illustrates the timing relationships for programming the write registers, and transferring data and status.

$C/\overline{D}$	$B/\overline{A}$	Function
0	0	Channel A Data
0	1	Channel B Data
1	0	Channel A Commands/Status
1	1	Channel B Commands/Status

Write Registers

The Z80-SIO contains eight registers (WR0-WR7) in each channel that are programmed separately by the system program to configure the functional personality of the channels. With the exception of WR0, programming the write registers requires two bytes. The first byte contains three bits (D_0-D_2) that point to the selected register; the second byte is the actual control word that is written into the register to configure the Z80-SIO.

Note that the programmer has complete freedom, after pointing to the selected register, of either reading to test the read register or writing to initialize the write register. By designing software to initialize the Z80-SIO in a modular and structured fashion, the programmer can use powerful block I/O instructions.

WR0 is a special case in that all the basic commands (CMD_0-CMD_2) can be accessed with a single byte. Reset (internal or external) initializes the pointer bits D_0-D_2 to point to WR0.

The basic commands (CMD_0-CMD_2) and the CRC controls (CRC_0, CRC_1) are contained in the first byte of any write register access. This maintains maximum flexibility and system control. Each channel contains the following control registers. These registers are addressed as commands (not data).

WRITE REGISTER 0

WR0 is the command register; however, it is also used for CRC reset codes and to point to the other registers.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
CRC Reset Code 1	CRC Reset Code 0	CMD 2	CMD 1	CMD 0	PTR 2	PTR 1	PTR 0

Pointer Bits (D_0-D_2). Bits D_0-D_2 are pointer bits that determine which other write register the next byte is to be written into or which read register the next byte is to be read from. The first byte written into each channel after a reset (either by a Reset command or by the external reset input) goes into WR0. Following a read or write to any register (except WR0), the pointer will point to WR0.

Command Bits (D_3-D_5). Three bits, D_3-D_5, are encoded to issue the seven basic Z80-SIO commands.

Command	CMD_2	CMD_1	CMD_0	
0	0	0	0	Null Command (no effect)
1	0	0	1	Send Abort (SDLC Mode)
2	0	1	0	Reset External/Status Interrupts
3	0	1	1	Channel Reset
4	1	0	0	Enable Interrupt on next Rx Character
5	1	0	1	Reset Transmitter Interrupt Pending
6	1	1	0	Error Reset (latches)
7	1	1	1	Return from Interrupt (Channel A)

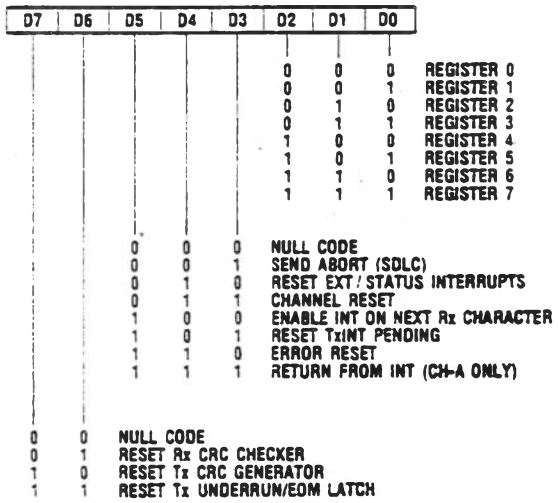
Command 0 (Null). The Null command has no effect. Its normal use is to cause the Z80-SIO to do nothing while the pointers are set for the following byte.

Command 1 (Send Abort). This command is used only with the SDLC mode to generate a sequence of eight to thirteen 1's.

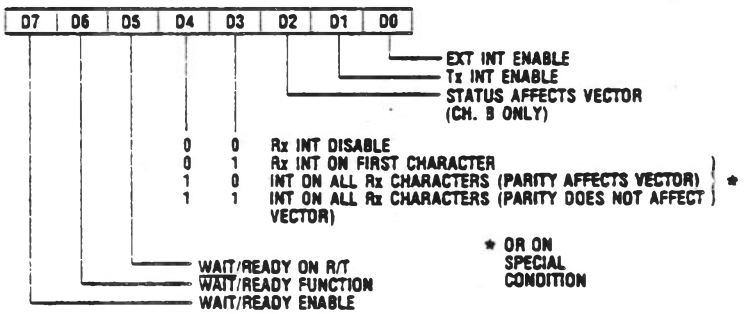
Command 2 (Reset External/Status Interrupts). After an External/Status interrupt (a change on a modem line or a break condition, for example), the status bits of RR0 are latched. This command re-enables them and allows interrupts to occur again. Latching the status bits captures short pulses until the CPU has time to read the change.

Command 3 (Channel Reset). This command performs the same function as an External Reset, but only on a single channel. Channel A Reset also resets the interrupt prioritization logic. All control registers for the channel must be rewritten after a Channel Reset command.

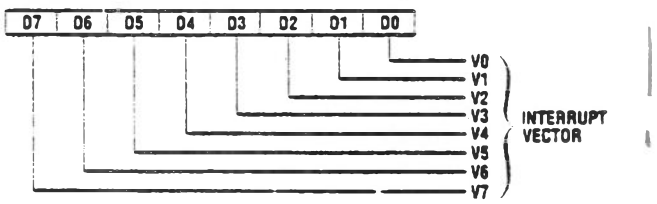
WRITE REGISTER 0



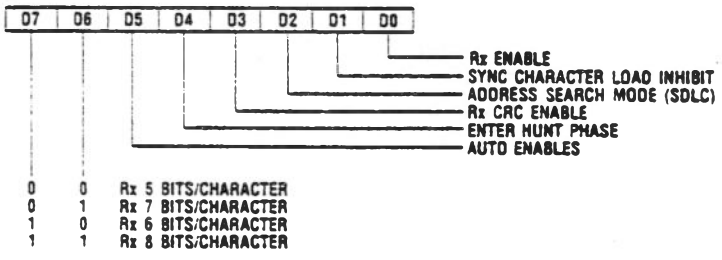
WRITE REGISTER 1



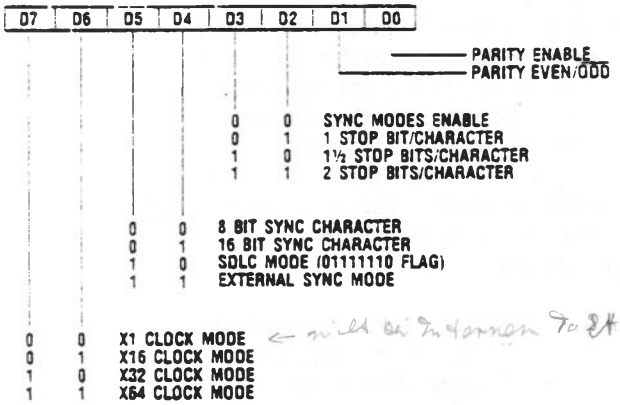
WRITE REGISTER 2 (CHANNEL B ONLY)



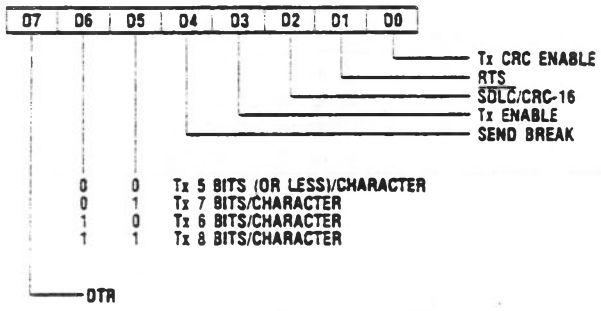
WRITER REGISTER 3



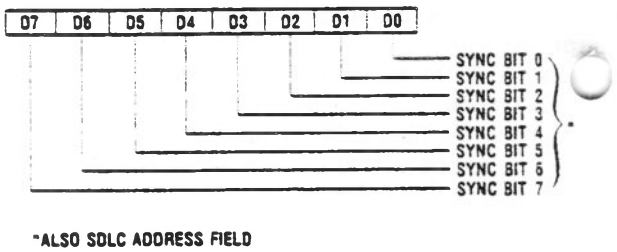
WRITE REGISTER 4



WRITE REGISTER 5

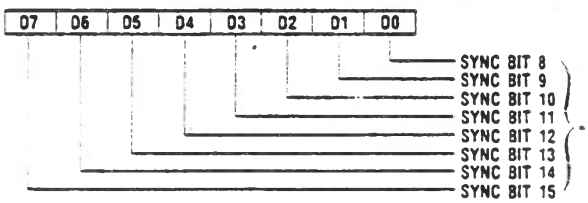


WRITE REGISTER 6



*ALSO SDLC ADDRESS FIELD

WRITE REGISTER 7



*FOR SDLC IT MUST BE PROGRAMMED TO "01111110" FOR FLAG RECOGNITION

Figure 9. Write Register Bit Functions

After a Channel Reset, four extra system clock cycles should be allowed for Z80-SIO reset time before any additional commands or controls are written into that channel. This can normally be the time used by the CPU to fetch the next op code.

Command 4 (Enable Interrupt On Next Receive Character). If the Interrupt On First Receive Character mode is selected, this command reactivates that mode after each complete message is received to prepare the Z80-SIO for the next message.

Command 5 (Reset Transmitter Interrupt Pending). The transmitter interrupts when the transmit buffer becomes empty if the Transmit Interrupt Enable mode is selected. In those cases where there are no more characters to be sent (at the end of message, for example), issuing this command prevents further transmitter interrupts until after the next character has been loaded into the transmit buffer or until CRC has been completely sent.

Command 6 (Error Reset). This command resets the error latches. Parity and Overrun errors are latched in RR1 until they are reset with this command. With this scheme, parity errors occurring in block transfers can be examined at the end of the block.

Command 7 (Return From Interrupt). This command must be issued in Channel A and is interpreted by the Z80-SIO in exactly the same way it would interpret an RETI command on the data bus. It resets the interrupt-under-service latch of the highest-priority internal device under service and thus allows lower priority devices to interrupt via the daisy chain. This command allows use of the internal daisy chain even in systems with no external daisy chain or RETI command.

CRC Reset Codes 0 and 1 (D_6 and D_7). Together, these bits select one of the three following reset commands:

CRC Reset Code 1	CRC Reset Code 0	
0	0	Null Code (no affect)
0	1	Reset Receive CRC Checker
1	0	Reset Transmit CRC Generator
1	1	Reset Tx Underrun/End Of Message latch

The Reset Transmit CRC Generator command normally initializes the CRC generator to all 0's. If the SDLC mode is selected, this command initializes the CRC generator to all 1's. The Receive CRC checker is also initialized to all 1's for the SDLC mode.

WRITE REGISTER 1

WR1 contains the control bits for the various interrupt and Wait/Ready modes.

D_7	D_6	D_5	D_4
Wait/Ready Enable	$\overline{\text{Wait}}$ Or Ready Function	Wait/Ready On Receive/ Transmit	Receive Interrupt Mode 1
D_3	D_2	D_1	D_0
Receive Interrupt Mode 0	Status Affects Vector	Transmit Interrupt Enable	External Interrupts Enable

External/Status Interrupt Enable (D_0). The External/Status Interrupt Enable allows interrupts to occur as a result of transitions on the $\overline{\text{DCD}}$, $\overline{\text{CTS}}$ or $\overline{\text{SYNC}}$ inputs, as a result of a Break/Abort detection and termination, or at the beginning of CRC or sync character transmission when the Transmit Underrun/EOM latch becomes set.

Transmitter Interrupt Enable (D_1). If enabled, interrupts occur whenever the transmitter buffer becomes empty.

Status Affects Vector (D_2). This bit is active in Channel B only. If this bit is not set, the fixed vector programmed in WR2 is returned from an interrupt acknowledge sequence. If this bit is set, the vector returned from an interrupt acknowledge is variable according to the following interrupt conditions:

	V_3	V_2	V_1	
Ch B	0	0	0	Ch B Transmit Buffer Empty
	0	0	1	Ch B External/Status Change
	0	1	0	Ch B Receive Character Available
	0	1	1	Ch B Special Receive Condition*
Ch A	1	0	0	Ch A Transmit Buffer Empty
	1	0	1	Ch A External/Status Change
	1	1	0	Ch A Receive Character Available
	1	1	1	Ch A Special Receive Condition*

*Special Receive Conditions: Parity Error, Rx Overrun Error, Framing Error, End Of Frame (SDLC).

Receive Interrupt Modes 0 and 1 (D_3 and D_4). Together these two bits specify the various character-available conditions. In Receive Interrupt modes 1, 2 and 3, a Special Receive Condition can cause an interrupt and modify the interrupt vector.

D_4 Receive Interrupt Mode 1	D_3 Receive Interrupt Mode 0	
0	0	0. Receive Interrupts Disabled
0	1	1. Receive Interrupt On First Character Only
1	0	2. Interrupt On All Receive Characters—parity error is a Special Receive condition
1	1	3. Interrupt On All Receive Characters—parity error is not a Special Receive condition

Wait/Ready Function Selection (D_5–D_7). The Wait and Ready functions are selected by controlling D_5, D_6, and D_7. Wait/Ready function is enabled by setting Wait/Ready Enable (WR1, D_7) to 1. The Ready function is selected by setting D_6 (Wait/Ready function) to 1. If this bit is 1, the $\overline{\text{WAIT/READY}}$ output switches from High to Low when the Z80-SIO is ready to transfer data. The Wait function is selected by setting D_6 to 0. If this bit is

0, the $\overline{\text{WAIT/READY}}$ output is in the open-drain state and goes Low when active.

Both the Wait and Ready functions can be used in either the Transmit or Receive modes, but not both simultaneously. If D_5 (Wait/Ready on Receive/Transmit) is set to 1, the Wait/Ready function responds to the condition of the receive buffer (empty or full). If D_5 is set to 0, the Wait/Ready function responds to the condition of the transmit buffer (empty or full).

The logic states of the $\overline{\text{WAIT/READY}}$ output when active or inactive depend on the combination of modes selected. Following is a summary of these combinations:

If $D_7 = 0$	
And $D_6 = 1$	And $D_6 = 0$
$\overline{\text{READY}}$ is High	$\overline{\text{WAIT}}$ is floating

If $D_7 = 1$			
And $D_5 = 0$		And $D_5 = 1$	
$\overline{\text{READY}}$	Is High when transmit buffer is full.	$\overline{\text{READY}}$	Is High when receive buffer is empty.
$\overline{\text{WAIT}}$	Is Low when transmit buffer is full and an SIO data port is selected.	$\overline{\text{WAIT}}$	Is Low when receive buffer is empty and an SIO data port is selected.
$\overline{\text{READY}}$	Is Low when transmit buffer is empty.	$\overline{\text{READY}}$	Is Low when receive buffer is full.
$\overline{\text{WAIT}}$	Is floating when transmit buffer is empty.	$\overline{\text{WAIT}}$	Is floating when receive buffer is full.

The $\overline{\text{WAIT}}$ output High-to-Low transition occurs with the delay time $t_{DIC(WR)}$ after the I/O request. The Low-to-High transition occurs with the delay $t_{DH\phi(WR)}$ from the falling edge of ϕ. The $\overline{\text{READY}}$ output High-to-Low transition occurs with the delay $t_{DL\phi(WR)}$ from the rising edge of ϕ. The $\overline{\text{READY}}$ output Low-to-High transition occurs with the delay $t_{DIC(WR)}$ after $\overline{\text{IORQ}}$ falls.

The Ready function can occur any time the Z80-SIO is not selected. When the $\overline{\text{READY}}$ output becomes active (Low), the DMA controller issues $\overline{\text{IORQ}}$ and the corresponding B/$\overline{\text{A}}$ and C/$\overline{\text{D}}$ inputs to the Z80-SIO to transfer data. The $\overline{\text{READY}}$ output becomes inactive as soon as $\overline{\text{IORQ}}$ and $\overline{\text{CS}}$ become active. Since the Ready function can occur internally in the Z80-SIO whether it is addressed or not, the $\overline{\text{READY}}$ output becomes inactive when any CPU data or command transfer takes place. This does not cause problems because the DMA controller is not enabled when the CPU transfer takes place.

The Wait function—on the other hand—is active only if the CPU attempts to read Z80-SIO data that has not yet been received, which occurs frequently when block transfer instructions are used. The Wait function can also become active (under program control) if the CPU tries to write data while the transmit buffer is still full. The fact that the $\overline{\text{WAIT}}$ output for either channel can become active when the opposite channel is addressed (because the Z80-SIO is addressed) does not affect operation of software loops or block move instructions.

WRITE REGISTER 2

WR2 is the interrupt vector register; it exists in Channel B only. V_4–V_7 and V_0 are always returned exactly as written; V_1–V_3 are returned as written if the Status Affects Vector (WR1, D_2) control bit is 0. If this bit is 1, they are modified as explained in the previous section.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
V_7	V_6	V_5	V_4	V_3	V_2	V_1	V_0

WRITE REGISTER 3

WR3 contains receiver logic control bits and parameters.

D_7	D_6	D_5	D_4
Receiver Bits/ Char 1	Receiver Bits/ Char 0	Auto Enables	Enter Hunt Phase
D_3	**D_2**	**D_1**	**D_0**
Receiver CRC Enable	Address Search Mode	Sync Char Load Inhibit	Receiver Enable

Receiver Enable (D_0). A 1 programmed into this bit allows receive operations to begin. This bit should be set only after all other receive parameters are set and receiver is completely initialized.

Sync Character Load Inhibit (D_1). Sync characters preceding the message (leading sync characters) are not loaded into the receive buffers if this option is selected. Because CRC calculations are not stopped by sync character stripping, this feature should be enabled only at the beginning of the message.

Address Search Mode (D_2). If SDLC is selected, setting this mode causes messages with addresses not matching the programmed address in WR6 or the global (11111111) address to be rejected. In other words, no receive interrupts can occur in the Address Search mode unless there is an address match.

Receiver CRC Enable (D_3). If this bit is set, CRC calculation starts (or restarts) at the beginning of the last character transferred from the receive shift register to the buffer stack, regardless of the number of characters in the stack. See "SDLC Receive CRC Checking" (SDLC Receive section) and "CRC Error Checking" (Synchronous Receive section) for details regarding when this bit should be set.

Enter Hunt Phase (D_4). The Z80-SIO automatically enters the Hunt phase after a reset; however, it can be re-entered if character synchronization is lost for any reason (Synchronous mode) or if the contents of an incoming message are not needed (SDLC mode). The Hunt phase is re-entered by writing a 1 into bit D_4. This sets the Sync/Hunt bit (D_4) in RR0.

Auto Enables (D_5). If this mode is selected, $\overline{DCD}$ and $\overline{CTS}$ become the receiver and transmitter enables, respectively. If this bit is not set, $\overline{DCD}$ and $\overline{CTS}$ are simply inputs to their corresponding status bits in RR0.

Receiver Bits/Characters 1 and 0 (D_7 and D_6). Together, these bits determine the number of serial receive bits assembled to form a character. Both bits may be changed during the time that a character is being assembled, but they must be changed before the number of bits currently programmed is reached.

D_7	D_6	Bits/Character
0	0	5
0	1	7
1	0	6
1	1	8

WRITE REGISTER 4

WR4 contains the control bits that affect both the receiver and transmitter. In the transmit and receive initialization routine, these bits should be set before issuing WR1, WR3, WR5, WR6, and WR7.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Clock Rate 1	Clock Rate 0	Sync Modes 1	Sync Modes 0	Stop Bits 1	Stop Bits 0	Parity Even/$\overline{\text{Odd}}$	Parity

Parity (D_0). If this bit is set, an additional bit position (in addition to those specified in the bits/character control) is added to transmitted data and is expected in receive data. In the Receive mode, the parity bit received is transferred to the CPU as part of the character, unless 8 bits/character is selected.

Parity Even/$\overline{\text{Odd}}$ (D_1). If parity is specified, this bit determines whether it is sent and checked as even or odd (1 = even).

Stop Bits 0 and 1 (D_2 and D_3). These bits determine the number of stop bits added to each asynchronous character sent. The receiver always checks for one stop bit. A special mode (00) signifies that a synchronous mode is to be selected.

D_3 Stop Bits 1	D_2 Stop Bits 0	
0	0	Sync modes
0	1	1 stop bit per character
1	0	1½ stop bits per character
1	1	2 stop bits per character

Sync Modes 0 and 1 (D_4 and D_5). These bits select the various options for character synchronization.

Sync Mode 1	Sync Mode 0	
0	0	8-bit programmed sync
0	1	16-bit programmed sync
1	0	SDLC mode (01111110 flag pattern)
1	1	External Sync mode

Clock Rate 0 and 1 (D_6 and D_7). These bits specify the multiplier between the clock ($\overline{TxC}$ and $\overline{RxC}$) and data rates. For synchronous modes, the $\times 1$ clock rate must be specified. Any rate may be specified for asynchronous modes; however, the same rate must be used for both the receiver and transmitter. The system clock in all modes must be at least 4.5 times the data rate. If the $\times 1$ clock rate is selected, bit synchronization must be accomplished externally.

Clock Rate 1	Clock Rate 0	
0	0	Data Rate × 1 = Clock Rate
0	1	Data Rate × 16 = Clock Rate
1	0	Data Rate × 32 = Clock Rate
1	1	Data Rate × 64 = Clock Rate

WRITE REGISTER 5

WR5 contains control bits that affect the operation of transmitter, with the exception of D_2, which affects the transmitter and receiver.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
DTR	Tx Bits/ Char 1	Tx Bits/ Char 0	Send Break	Tx Enable	CRC-16/ $\overline{\text{SDLC}}$	RTS	Tx CRC Enable

Transmit CRC Enable (D_0). This bit determines if CRC is calculated on a particular transmit character. If it is set at the time the character is loaded from the transmit buffer into the transmit shift register, CRC is calculated on the character. CRC is not automatically sent unless this bit is set when the Transmit Underrun condition exists.

Request To Send (D_1). This is the control bit for the $\overline{RTS}$ pin. When the $\overline{RTS}$ bit is set, the $\overline{RTS}$ pin goes Low; when reset, $\overline{RTS}$ goes High. In the Asynchronous mode, $\overline{RTS}$ goes High only after all the bits of the character are transmitted and the transmitter buffer is empty. In Synchronous modes, the pin directly follows the state of the bit.

CRC-16/$\overline{\text{SDLC}}$ (D_2). This bit selects the CRC polynomial used by both the transmitter and receiver. When set, the CRC-16 polynomial ($X^{16} + X^{15} + X^2 + 1$) is used; when reset the SDLC polynomial ($X^{16} + X^{12} + X^5 + 1$) is used. If the SDLC mode is selected, the CRC generator and checker are preset to all 1's and a special check sequence is used. The SDLC CRC polnomial must be selected when the SDLC mode is selected. If the SDLC mode is not selected, the CRC generator and checker are preset to all 0's (for both polynomials).

Transmit Enable (D_3). Data is not transmitted until this bit is set, and the Transmit Data output is held marking. Data or sync characters in the process of being transmitted are completely sent if this bit is reset after transmission has started. If the transmitter is disabled during the transmission of a CRC character, sync or flag characters are sent instead of CRC.

Send Break (D_4). When set, this bit immediately forces the Transmit Data output to the spacing condition, regardless of any data being transmitted. When reset, TxD returns to marking.

Transmit Bits/Characters 0 and 1 (D_5 and D_6). Together, D_6 and D_5 control the number of bits in each byte transferred to the transmit buffer.

D_6 Transmit Bits/ Character 1	D_5 Transmit Bits/ Character 0	Bits/Character
0	0	Five or less
0	1	7
1	0	6
1	1	8

Bits to be sent must be right justified, least-significant bits first. The Five Or Less mode allows transmission of one to five bits per character; however, the CPU should format the data character as shown in the following table.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
1	1	1	1	0	0	0	D	Sends one data bit
1	1	1	0	0	0	D	D	Sends two data bits
1	1	0	0	0	D	D	D	Sends three data bits
1	0	0	0	D	D	D	D	Sends four data bits
0	0	0	D	D	D	D	D	Sends five data bits

Data Terminal Ready (D_7). This is the control bit for the $\overline{DTR}$ pin. When set, $\overline{DTR}$ is active (Low); when reset, $\overline{DTR}$ is inactive (High).

WRITE REGISTER 6

This register is programmed to contain the transmit sync character in the Monosync mode, the first eight bits of a 16-bit sync character in the Bisync mode, or a transmit sync character in the External Sync mode. In the SDLC mode, it is programmed to contain the secondary address field used to compare against the address field of the SDLC frame.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Sync 7	Sync 6	Sync 5	Sync 4	Sync 3	Sync 2	Sync 1	Sync 0

WRITE REGISTER 7

This register is programmed to contain the receive sync character in the Monosync mode, a second byte (last eight bits) of a 16-bit sync character in the Bisync mode, or a flag character (01111110) in the SDLC mode. WR7 is not used in the External Sync mode.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Sync 15	Sync 14	Sync 13	Sync 12	Sync 11	Sync 10	Sync 9	Sync 8

Read Registers

The Z80-SIO contains three registers, RR0-RR2 (Figure 10), that can be read to obtain the status information for each channel (except for RR2—Channel B only). The status information includes error conditions, interrupt vector and standard communications-interface signals.

To read the contents of a selected read register other than RR0, the system program must first write the pointer byte to WR0 in exactly the same way as a write register operation. Then, by executing an input instruction, the contents of the addressed read register can be read by the CPU.

The status bits of RR0 and RR1 are carefully grouped to simplify status monitoring. For example, when the interrupt vector indicates that a Special Receive Condition interrupt has occurred, all the appropriate error bits can be read from a single register (RR1).

READ REGISTER 0

This register contains the status of the receive and transmit buffers; the $\overline{DCD}$, $\overline{CTS}$ and $\overline{SYNC}$ inputs; the Transmit Underrun/EOM latch; and the Break/Abort latch.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
Break/ Abort	Transmit Underrun/ EOM	CTS	Sync/ Hunt	DCD	Transmit Buffer Empty	Interrupt Pending (Ch. A only)	Receive Character Available

Receive Character Available (D_0). This bit is set when at least one character is available in the receive buffer; it is reset when the receive FIFO is completely empty.

Interrupt Pending (D_1). Any interrupting condition in the Z80-SIO causes this bit to be set; however, it is readable only in Channel A. This bit is mainly used in applications that do not have vectored interrupts available. During the interrupt service routine in these applications, this bit indicates if any interrupt conditions are present in the Z80-SIO. This eliminates the need for analyzing all the bits of RR0 in both Channels A and B. Bit D_1 is reset when all the interrupting conditions are satisfied. This bit is always 0 in Channel B.

Transmit Buffer Empty (D_2). This bit is set whenever the transmit buffer becomes empty, except when a CRC character is being sent in a synchronous or SDLC mode. The bit is reset when a character is loaded into the transmit buffer. This bit is in the set condition after a reset.

Data Carrier Detect (D_3). The DCD bit shows the state of the $\overline{DCD}$ input at the time of the last change of any of the five External/Status bits (DCD, CTS, Sync/Hunt, Break/Abort or Transmit Underrun/EOM). Any transition of the $\overline{DCD}$ input causes the DCD bit to be latched

and causes an External/Status interrupt. To read the current state of the DCD bit, this bit must be read immediately following a Reset External/Status Interrupt command.

Sync/Hunt (D_4). Since this bit is controlled differently in the Asynchronous, Synchronous and SDLC modes, its operation is somewhat more complex than that of the other bits and therefore requires more explanation.

In asynchronous modes, the operation of this bit is similar to the DCD status bit, except that Sync/Hunt shows the state of the $\overline{\text{SYNC}}$ input. Any High-to-Low transition on the $\overline{\text{SYNC}}$ pin sets this bit and causes an External/Status interrupt (if enabled). The Reset External/Status Interrupt command is issued to clear the interrupt. A Low-to-High transition clears this bit and sets the External/Status interrupt. When the External/Status interrupt is set by the change in state of any other input or condition, this bit shows the inverted state of the $\overline{\text{SYNC}}$ pin at the time of the change. This bit must be read immediately following a Reset External/Status Interrupt command to read the current state of the $\overline{\text{SYNC}}$ input.

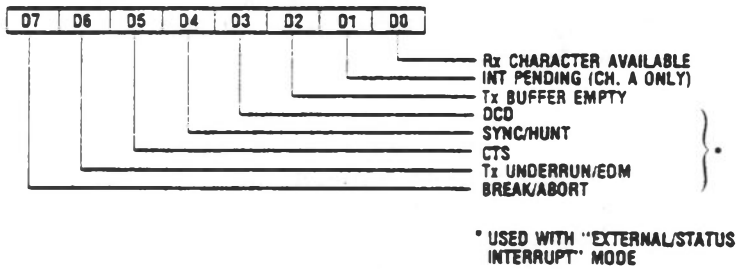
In the External Sync mode, the Sync/Hunt bit operates in a fashion similar to the Asynchronous mode, except the Enter Hunt Mode control bit enables the external sync detection logic. When the External Sync Mode and Enter Hunt Mode bits are set (for example, when the receiver is enabled following a reset), the $\overline{\text{SYNC}}$ input must be held High by the external logic until external character synchronization is achieved. A High at the $\overline{\text{SYNC}}$ input holds the Sync/Hunt status bit in the reset condition.

When external synchronization is achieved, $\overline{\text{SYNC}}$ must be driven Low on the second rising edge of $\overline{\text{RxC}}$ after that rising edge of $\overline{\text{RxC}}$ on which the last bit of the sync character was received. In other words, after the sync pattern is detected, the external logic must wait for two full Receive Clock cycles to activate the $\overline{\text{SYNC}}$ input. Once $\overline{\text{SYNC}}$ is forced Low, it is a good practice to keep it Low until the CPU informs the external sync logic that synchronization has been lost or a new message is about to start. Refer to Figure 18 for timing details. The High-to-Low transition of the $\overline{\text{SYNC}}$ input sets the Sync/Hunt bit, which—in turn—sets the External/Status interrupt. The CPU must clear the interrupt by issuing the Reset External/Status Interrupt command.

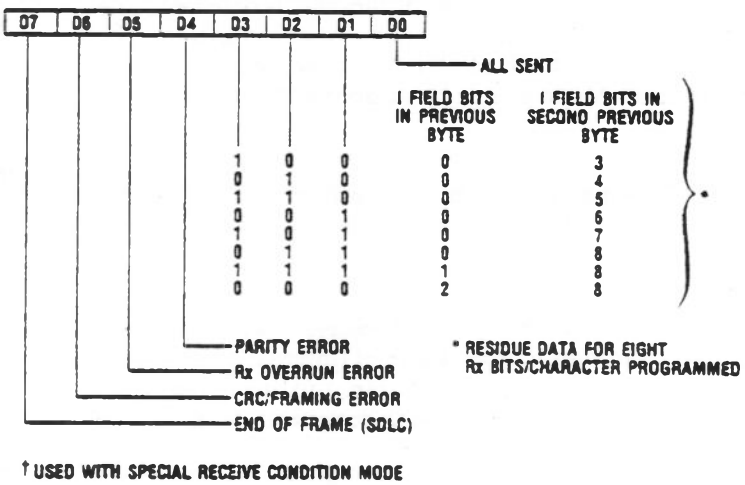
When the $\overline{\text{SYNC}}$ input goes High again, another External/Status interrupt is generated that must also be cleared. The Enter Hunt Mode control bit is set whenever character synchronization is lost or the end of message is detected. In this case, the Z80-SIO again looks for a High-to-Low transition on the $\overline{\text{SYNC}}$ input and the operation repeats as explained previously. This implies the CPU should also inform the external logic that character synchronization has been lost and that the Z80-SIO is waiting for $\overline{\text{SYNC}}$ to become active.

In the Monosync and Bisync Receive modes, the Sync/Hunt status bit is initially set to 1 by the Enter Hunt Mode bit. The Sync/Hunt bit is reset when the Z80-SIO establishes character synchronization. The

READ REGISTER 0



READ REGISTER 1†



READ REGISTER 2

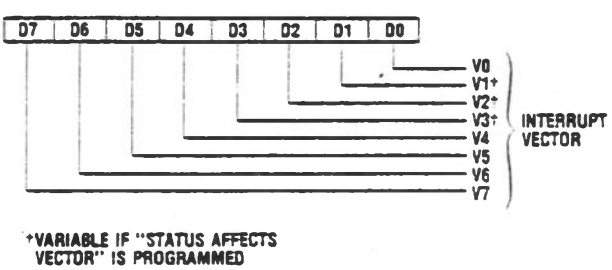


Figure 10. Read Register Bit Functions

High-to-Low transition of the Sync/Hunt bit causes an External/Status interrupt that must be cleared by the CPU issuing the Reset External/Status Interrupt command. This enables the Z80-SIO to detect the next transition of other External/Status bits.

When the CPU detects the end of message or that character synchronization is lost, it sets the Enter Hunt Mode control bit, which—in turn—sets the Sync/Hunt bit to 1. The Low-to-High transition of the Sync/Hunt bit sets the External/Status interrupt, which must also be cleared by the Reset External/Status Interrupt command. Note that the $\overline{\text{SYNC}}$ pin acts as an output in this mode and goes Low every time a sync pattern is detected in the data stream.

In the SDLC mode, the Sync/Hunt bit is initially set by the Enter Hunt mode bit, or when the receiver is disabled. In any case, it is reset to 0 when the opening flag of the first frame is detected by the Z80-SIO. The External/Status interrupt is also generated, and should be handled as discussed previously.

Unlike the Monosync and Bisync modes, once the Sync/Hunt bit is reset in the SDLC mode, it does not need to be set when the end of message is detected. The Z80-SIO automatically maintains synchronization. The only way the Sync/Hunt bit can be set again is by the Enter Hunt Mode bit, or by disabling the receiver.

Clear To Send (D_5). This bit is similar to the DCD bit, except that it shows the inverted state of the $\overline{\text{CTS}}$ pin.

Transmit Underrun/End Of Message (D_6). This bit is in a set condition following a reset (internal or external). The only command that can reset this bit is the Reset Transmit Underrun/EOM Latch command (WR0, D_6 and D_7). When the Transmit Underrun condition occurs, this bit is set; its becoming set causes the External/Status interrupt, which must be reset by issuing the Reset External/Status Interrupt command bits (WR0). This status bit plays an important role in conjunction with other control bits in controlling a transmit operation. Refer to "Bisync Transmit Underrun" and "SDLC Transmit Underrun" for additional details.

Break/Abort (D_7). In the Asynchronous Receive mode, this bit is set when a Break sequence (null character plus framing error) is detected in the data stream. The External/Status interrupt, if enabled, is set when Break is detected. The interrupt service routine must issue the Reset External/Status Interrupt command (WR0, CMD_2) to the break detection logic so the Break sequence termination can be recognized.

The Break/Abort bit is reset when the termination of the Break sequence is detected in the incoming data stream. The termination of the Break sequence also causes the External/Status interrupt to be set. The Reset External/Status Interrupt command must be issued to enable the break detection logic to look for the next Break sequence. A single extraneous null character is present in the receiver after the termination of a break; it should be read and discarded.

In the SDLC Receive mode, this status bit is set by the detection of an Abort sequence (seven or more 1's). The External/Status interrupt is handled the same way as in the case of a Break. The Break/Abort bit is not used in the Synchronous Receive mode.

READ REGISTER 1

This register contains the Special Receive condition status bits and Residue codes for the I-field in the SDLC Receive Mode.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
End Of Frame (SDLC)	CRC/ Framing Error	Receiver Overrun Error	Parity Error	Residue Code 2	Residue Code 1	Residue Code 0	All Sent

All Sent (D_0). In asynchronous modes, this bit is set when all the characters have completely cleared the transmitter. Transitions of this bit do not cause interrupts. It is always set in synchronous modes.

Residue Codes 0, 1 and 2 (D_1–D_3). In those cases of the SDLC receive mode where the I-field is not an integral multiple of the character length, these three bits indicate the length of the I-field. These codes are meaningful only for the transfer in which the End Of Frame bit is set (SDLC). For a receive character length of eight bits per character, the codes signify the following:

Residue Code 2	Residue Code 1	Residue Code 0	I-Field Bits In Previous Byte	I-Field Bits In Second Previous Byte
1	0	0	0	3
0	1	0	0	4
1	1	0	0	5
0	0	1	0	6
1	0	1	0	7
0	1	1	0	8
1	1	1	1	8
0	0	0	2	8

I-Field bits are right-justified in all cases.

If a receive character length different from eight bits is used for the I-field, a table similar to the previous one may be constructed for each different character length. For no residue (that is, the last character boundary coincides with the boundary of the I-field and CRC field), the Residue codes are:

Bits per Character	Residue Code 2	Residue Code 1	Residue Code 0
8 Bits per Character	0	1	1
7 Bits per Character	0	0	0
6 Bits per Character	0	1	0
5 Bits per Character	0	0	1

Parity Error (D_4). When parity is enabled, this bit is set for those characters whose parity does not match the programmed sense (even/odd). The bit is latched, so once an error occurs, it remains set until the Error Reset command (WR0) is given.

Receive Overrun Error (D_5). This bit indicates that more than three characters have been received without a read from the CPU. Only the character that has been written over is flagged with this error, but when this character is read, the error condition is latched until reset by the Error Reset command. If Status Affects Vector is enabled, the character that has been overrun interrupts with a Special Receive Condition vector.

CRC/Framing Error (D_6). If a Framing Error occurs (asynchronous modes), this bit is set (and not latched) for the receive character in which the Framing Error occurred. Detection of a Framing Error adds an additional one-half of a bit time to the character time so the Framing Error is not interpreted as a new start bit. In synchronous and SDLC modes, this bit indicates the result of comparing the CRC checker to the appropriate check value. This bit is reset by issuing an Error Reset command. The bit is not latched, so it is always updated when the next character is received. When used for CRC error and status in synchronous modes, it is usually set since most bit combinations result in a non-zero CRC except for a correctly completed message.

End Of Frame (D_7). This bit is used only with the SDLC mode and indicates that a valid ending flag has been received and that the CRC Error and Residue codes are also valid. This bit can be reset by issuing the Error Reset command. It is also updated by the first character of the following frame.

READ REGISTER 2 (Ch. B Only)

This register contains the interrupt vector written into WR2 if the Status Affects Vector control bit is not set. If the control bit is set, it contains the modified vector shown in the Status Affects Vector paragraph of the Write Register 1 section. When this register is read, the vector returned is modified by the highest priority interrupting condition at the time of the read. If no interrupts are pending, the vector is modified with $V_3 = 0$, $V_2 = 1$ and $V_1 = 1$. This register may be read only through Channel B.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
V_7	V_6	V_5	V_4	V_3	V_2	V_1	V_0
				Variable if Status Affects Vector is enabled			

Applications

The flexibility and versatility of the Z80-SIO make it useful for numerous applications, a few of which are included here. These examples show several applications that combine the Z80-SIO with other members of the Z80 family.

Figure 11 shows simple processor-to-processor communication over a direct line. Both remote processors in this system can communicate to the Z80-CPU with different protocols and data rates. Depending on the complexity of the application, other Z80 peripheral circuits (Z80-CTC, for example) may be required. The unused channel of the Z80-SIO can be used to control other peripherals or they can be connected to other remote processors.

Figure 12 illustrates how both channels of a single Z80-SIO are used with modems that have primary and secondary, or reverse channel options. Alternatively, two modems without these options can be connected to the Z80-SIO. A suitable baud-rate generator (Z80-CTC) must be used for asynchronous modems.

Figure 13 shows the Z80-SIO in a data concentrator, a relatively complex application that uses two Z80-SIOs to perform a variety of functions. The data concentrator can be used to collect data from many terminals over low-speed lines and transmit it over a single high-speed line after editing and reformatting.

The Z80-DMA controller circuit is used with Z80-SIO #2 to transmit the reformatted data at high speed with the required protocol. The high-speed modem provides the transmit clock for this channel. The Z80-CTC counter-timer circuit supplies the transmit and receive clocks for the low-speed lines and is also used as a time-out counter for various functions.

Z80-SIO #1 controls local or remote terminals. A single intelligent terminal is shown within the dashed lines. The terminal employs a Z80-SIO to communicate to the data concentrator on one channel while providing the interface to a line printer over its second channel. The intelligent terminal shown could be designed to operate interactively with the operator.

Depending on the software and hardware capabilities built into this system, the data concentrator can employ store-and-forward or hold-and-forward methods for regulating information traffic between slow terminals and the high-speed remote processor. If the high-speed channel is provided with a dial-out option, the channel can be connected to a number of remote processors over a switched line.

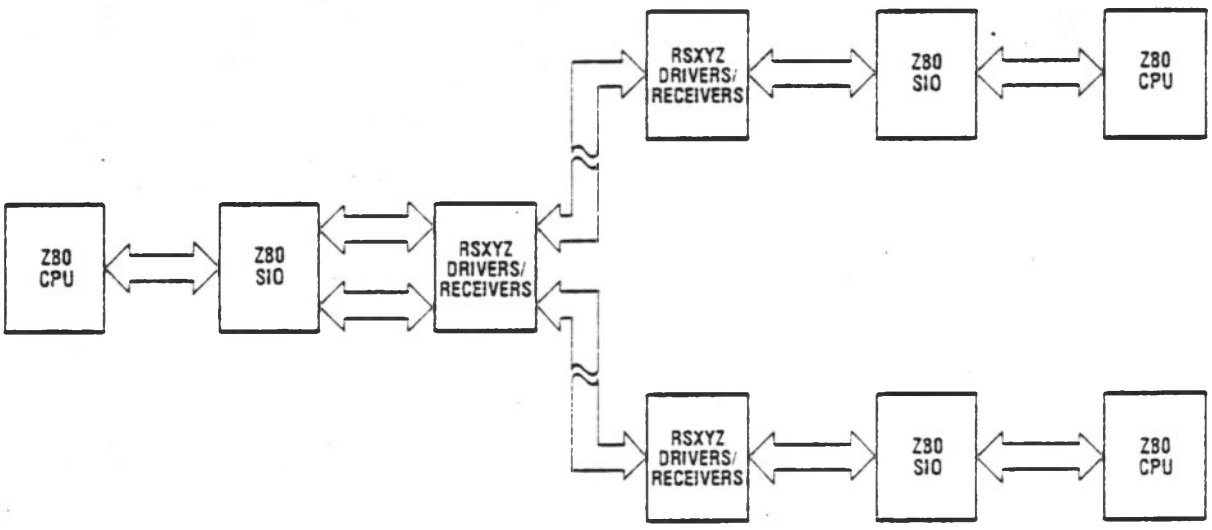


Figure 11. Synchronous/Asynchronous Processor-to-Processor Communcation (Direct Wire to Two Remote Locations)

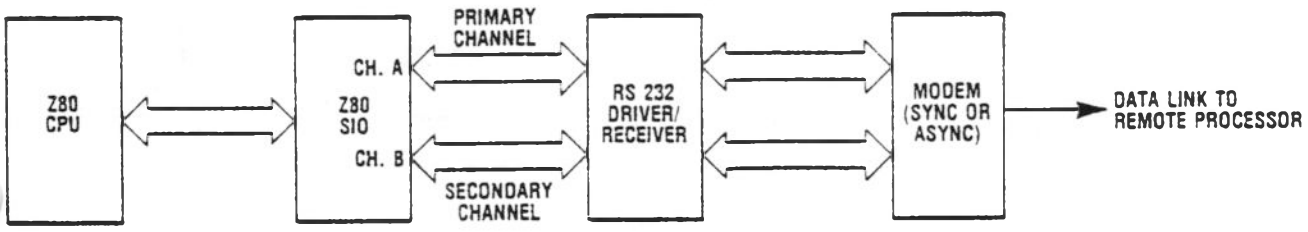


Figure 12. Synchronous/Asynchronous Processor-to-Processor Communication (Using Telephone Line)

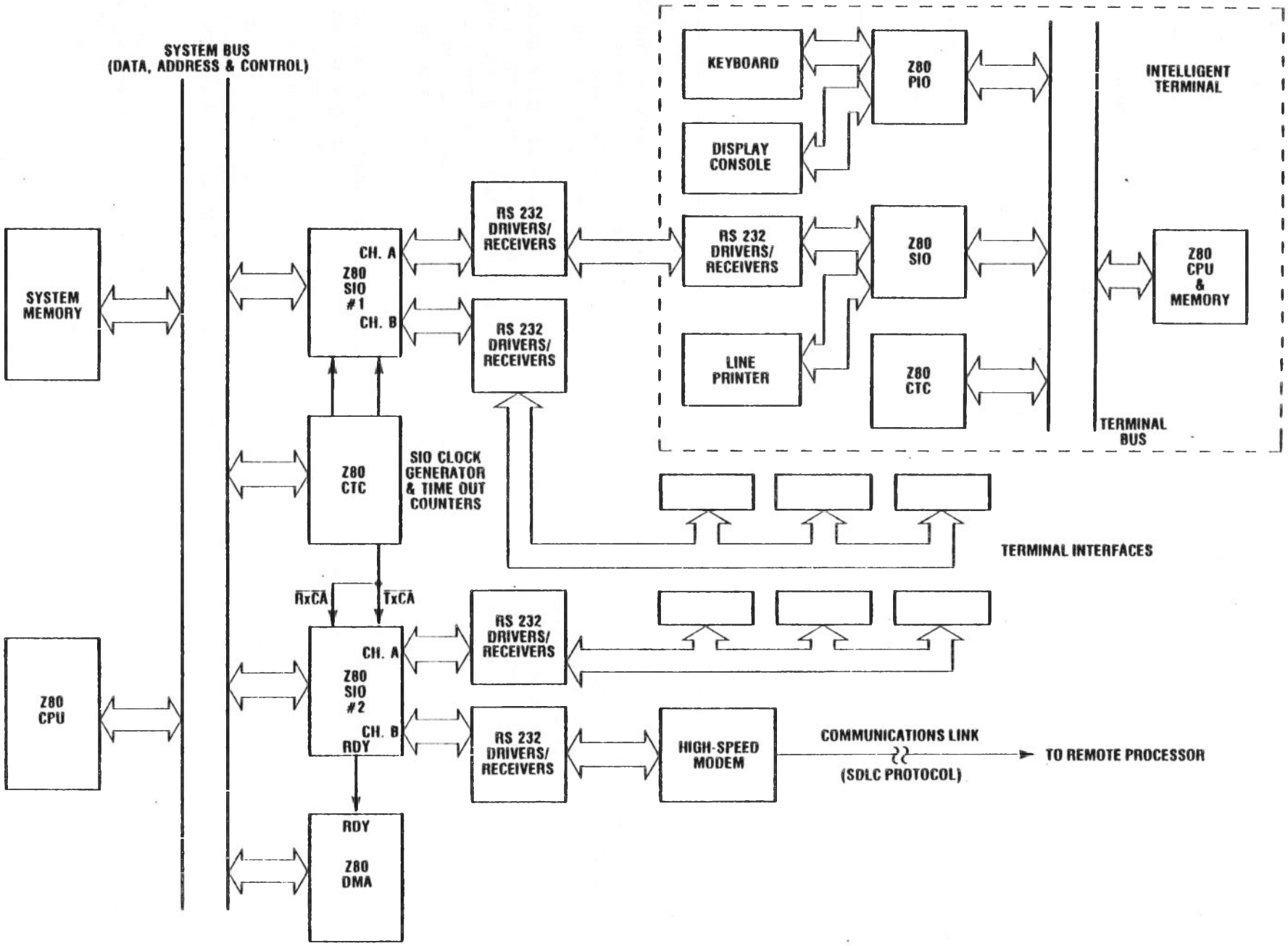


Figure 13. Data Concentrator

Timing

READ CYCLE

The timing signals generated by a Z80-CPU input instruction to read a Data or Status byte from the Z80-SIO are illustrated in Figure 14a.

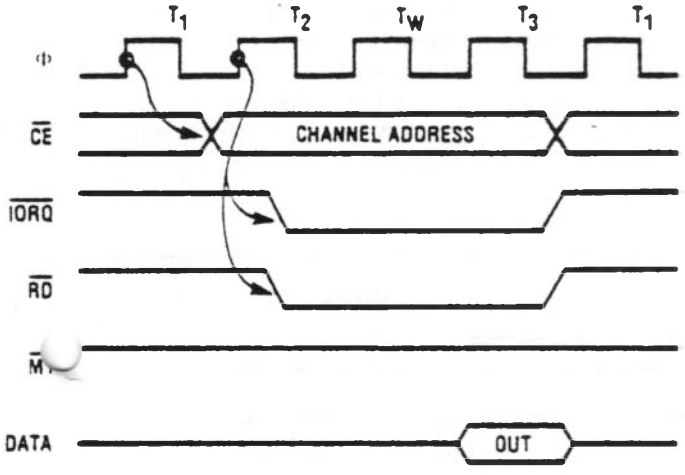


Figure 14a. Read Cycle

WRITE CYCLE

Figure 14b illustrates the timing and data signals generated by a Z80-CPU output instruction to write a Data or Control byte into the Z80-SIO.

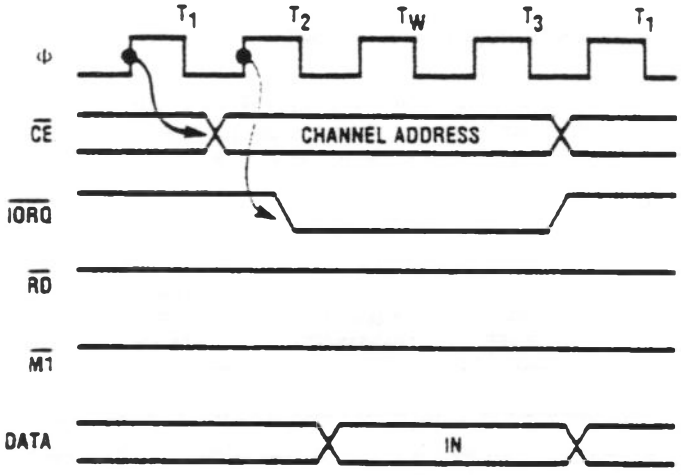


Figure 14b. Write Cycle

INTERRUPT ACKNOWLEDGE CYCLE

After receiving an Interrupt Request signal ($\overline{INT}$ pulled Low), the Z80-CPU sends an Interrupt Acknowledge signal ($\overline{M1}$ and $\overline{IORQ}$ both Low). The daisy-chained interrupt circuits determine the highest priority interrupt requestor. The IEI of the highest priority peripheral is terminated High. For any peripheral that has no interrupt pending or under service, IEO = IEI. Any peripheral that does have an interrupt pending or under service forces its IEO Low.

To insure stable conditions in the daisy chain, all interrupt status signals are prevented from changing while $\overline{M1}$ is Low. When $\overline{IORQ}$ is Low, the highest priority interrupt requestor (the one with IEI High) places its interrupt vector on the data bus and sets its internal interrupt-under-service latch.

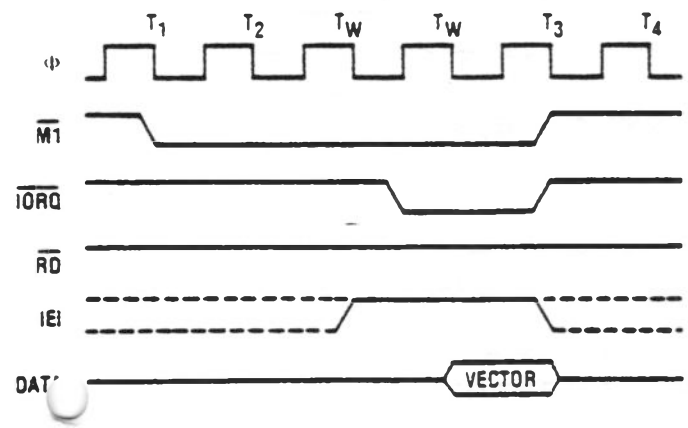


Figure 14c. Interrupt Acknowledge Cycle

RETURN FROM INTERRUPT CYCLE

Normally, the Z80-CPU issues a RETI (RETurn from Interrupt) instruction at the end of an interrupt service routine. RETI is a 2-byte opcode (ED-4D) that resets the interrupt-under-service latch to terminate the interrupt that has just been processed. This is accomplished by manipulating the daisy chain in the following way.

The normal daisy chain operation can be used to detect a pending interrupt; however, it cannot distinguish between an interrupt under service and a pending unacknowledged interrupt of a higher priority. Whenever "ED" is decoded, the daisy chain is modified by forcing High the IEO of any interrupt that has not yet been acknowledged. Thus the daisy chain identifies the device presently under service as the only one with an IEI High and an IEO Low. If the next opcode byte is "4D," the interrupt-under-service latch is reset.

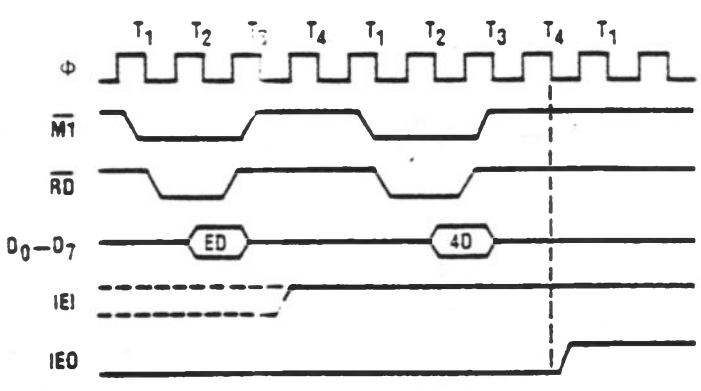


Figure 14d. Return from Interrupt Cycle

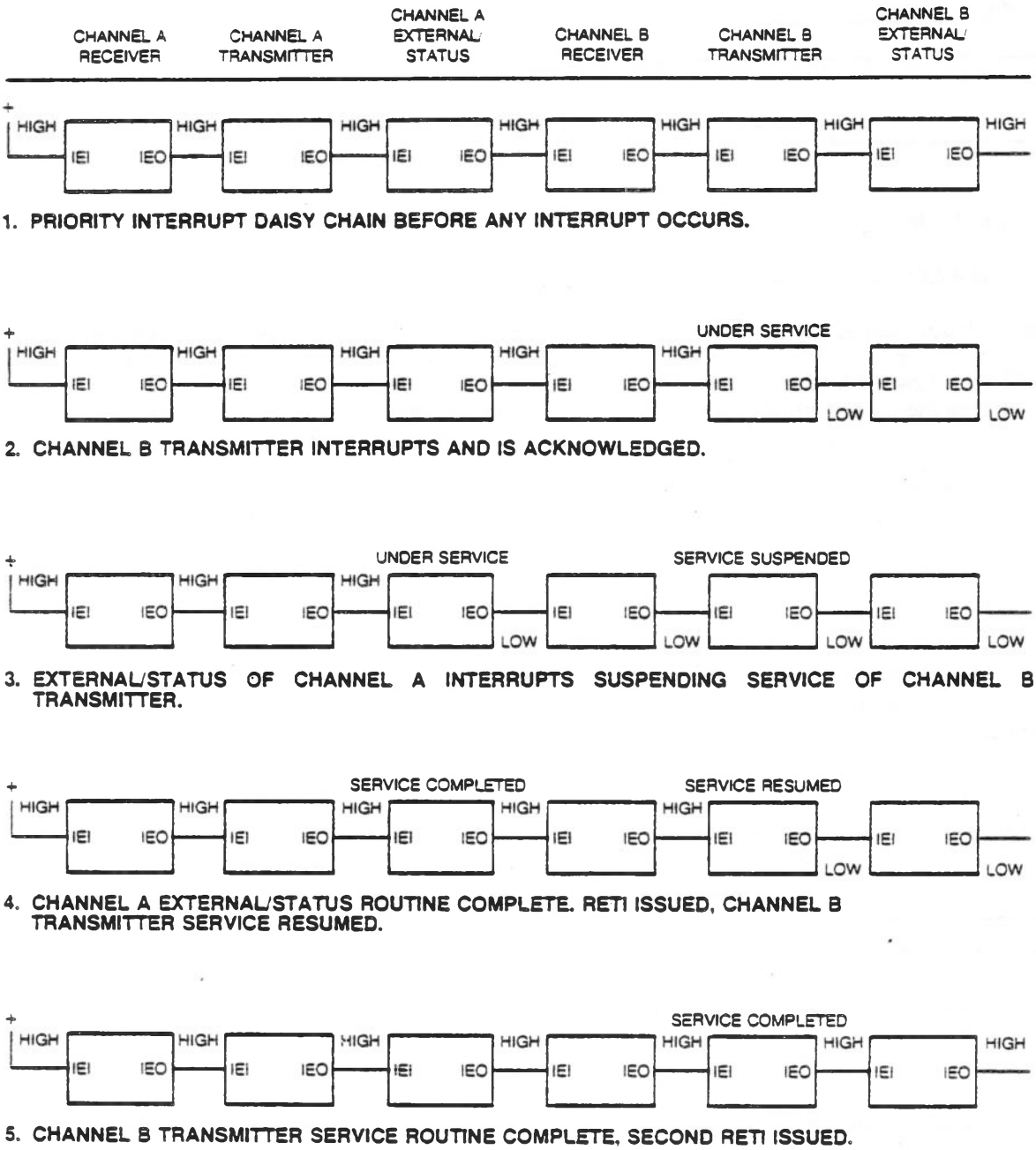
The ripple time of the interrupt daisy chain (both the High-to-Low and the Low-to-High transitions) limits the number of devices that can be placed in the daisy chain. Ripple time can be improved with carry-look-ahead, or by extending the interrupt acknowledge cycle. For further information about techniques for increasing the number of daisy-chained devices, refer to Zilog Application Note 03-0041-01 (*The Z80 Family Program Interrupt Structure*).

DAISY CHAIN INTERRUPT NESTING

Figure 15 illustrates the daisy chain configuration of interrupt circuits and their behavior with nested interrupts (an interrupt that is interrupted by another with a higher priority).

Each box in the illustration could be a separate external Z80 peripheral circuit with a user-defined order of interrupt priorities. However, a similar daisy chain structure also exists inside the Z80-SIO, which has six interrupt levels with a fixed order of priorities.

The case illustrated occurs when the transmitter of Channel B interrupts and is granted service. While this interrupt is being serviced, it is interrupted by a higher priority interrupt from Channel A. The second interrupt is serviced and—upon completion—a RETI instruction is executed or a RETI command is written into the Z80-SIO, resetting the interrupt-under-service latch of the Channel A interrupt. At this time, the service routine for Channel B is resumed. When it is completed, another RETI instruction is executed to complete the interrupt service.



1. PRIORITY INTERRUPT DAISY CHAIN BEFORE ANY INTERRUPT OCCURS.

2. CHANNEL B TRANSMITTER INTERRUPTS AND IS ACKNOWLEDGED.

3. EXTERNAL/STATUS OF CHANNEL A INTERRUPTS SUSPENDING SERVICE OF CHANNEL B TRANSMITTER.

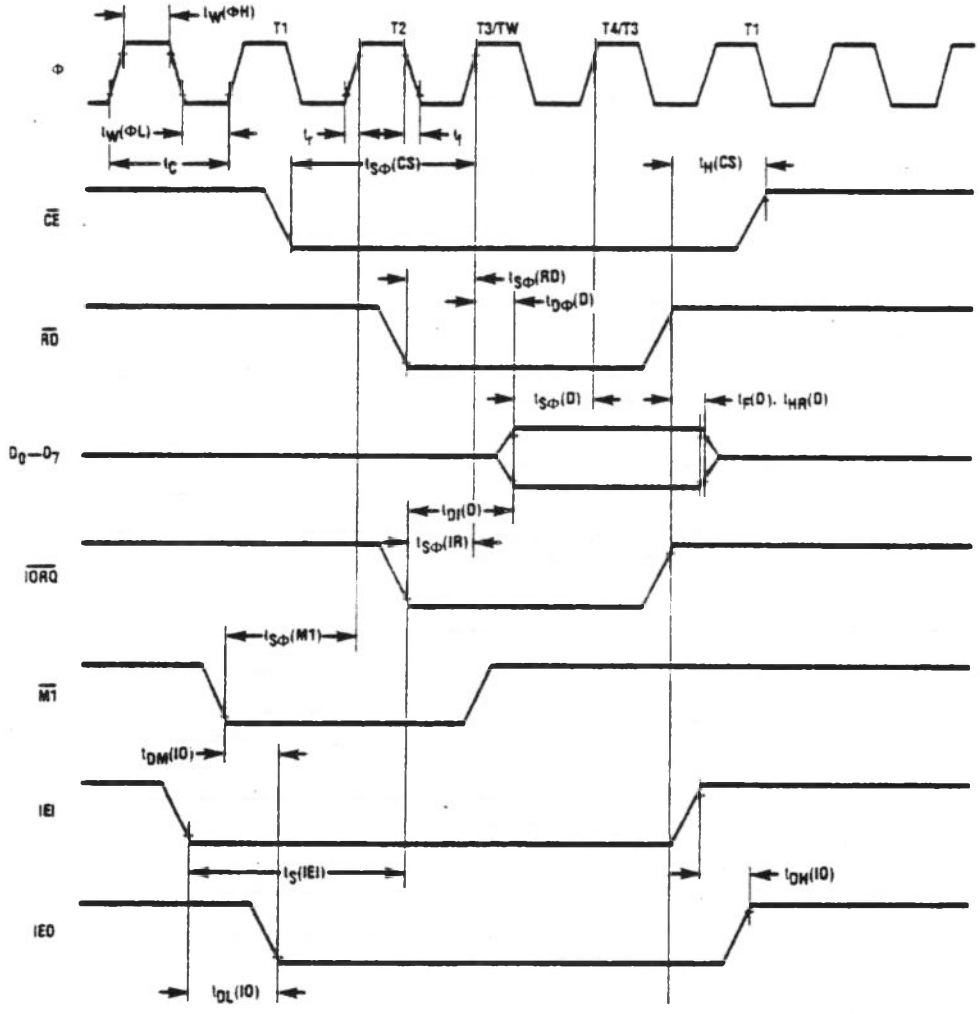
4. CHANNEL A EXTERNAL/STATUS ROUTINE COMPLETE. RETI ISSUED, CHANNEL B TRANSMITTER SERVICE RESUMED.

5. CHANNEL B TRANSMITTER SERVICE ROUTINE COMPLETE, SECOND RETI ISSUED.

Figure 15. Typical Interrupt Sequence

AC Characteristics

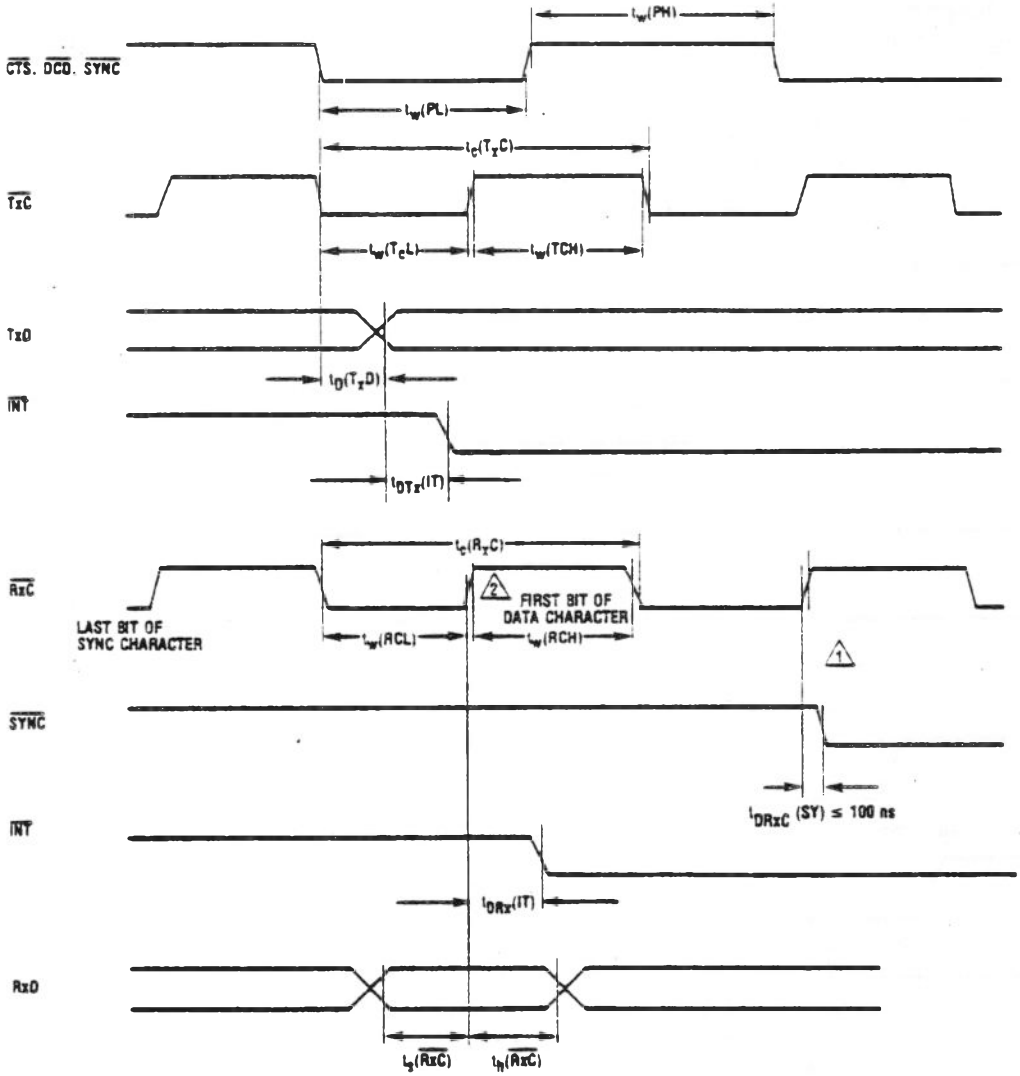
$T_A = 0°C$ to $70°C$, $V_{CC} = +5V, \pm 5\%$



Signal	Symbol	Parameter	Z80-SIO Min	Z80-SIO Max	Z80A-SIO Min	Z80A-SIO Max	Unit
Φ	$t_{c(\Phi)}$	Clock Period	400	4000	250	4000	ns
	$t_{w(\Phi H)}$	Clock Pulse Width, clock HIGH	170	2000	105	2000	ns
	$t_{w(\Phi L)}$	Clock Pulse Width, clock LOW	170	2000	105	2000	ns
	t_r, t_f	Clock Rise and Fall Times	0	30	0	30	ns
	t_H	Any Unspecified Hold Time for setup times specified below	0		0		ns
CE, B/A C/D, IORQ	$t_{s\Phi}(CS)$	Control Signal Setup Time to rising edge of Φ during Read or Write Cycle	160		145		ns
D_0-D_7	$t_{d\Phi}(D)$	Data Output Delay from rising edge of Φ during Read Cycle		240		220	ns
	$t_{s\Phi}(D)$	Data Setup Time to rising edge of Φ during Write or M1 Cycle	50		50		ns
	$t_{dI}(D)$	Data Output Delay from falling edge of IORQ during INTA Cycle		340		160	ns
	$t_z(D)$	Delay to Floating Bus (output buffer disable time)		230		110	ns
IEI	$t_s(IEI)$	IEI Setup Time to falling edge of IORQ during INTA Cycle	200		140		ns
IEO	$t_{dm}(IO)$	IEO Delay Time from rising edge of IEI (after 'ED' decode)		150		100	ns
	$t_{dI}(IO)$	IEO Delay Time from falling edge of IEI		150		100	ns
	$t_{dm}(IO)$	IEO Delay Time from falling edge of M1 (interrupt occurring just prior to M1)		300		190	ns
M1	$t_{s\Phi}(M1)$	M1 Setup Time to rising edge of Φ during INTA or M1 Cycle	210		90		ns
RD	$t_{s\Phi}(RD)$	RD Setup Time to rising edge of Φ during Read or M1 Cycle	240		115		ns

*If WAIT from the SIO is to be used, CE, IORQ, C/D and M1 must be valid for as long as the Wait condition is to persist.

AC Characteristics (Continued)



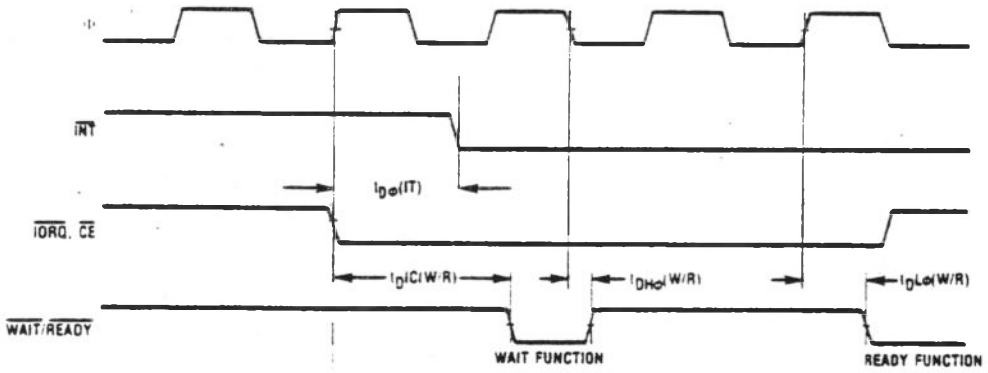
NOTES:

1. The SYNC input must be driven Low on the rising edge of RxC delayed two complete clock cycles from the last bit of the sync character.
2. Data character assembly begins on the next Receive Clock cycle after the last bit of the sync character is received.

Signal	Symbol	Parameter	Z80-SIO Min	Z80-SIO Max	Z80A-SIO Min	Z80A-SIO Max	Unit
INT	$t_{dm}(IT)$	INT Delay Time from rising edge of RxC	10	13	10	13	φ periods
	$t_{dTx}(IT)$	INT Delay Time from transition of Xmit Data Bit	5	9	5	9	φ periods
CTSA. CTSB. DCDA. DCDB. SYNCA. SYNCB	$t_w(PH)$	Minimum HIGH Pulse Width for latching state into register and generating interrupt	200		200		ns
	$t_w(PL)$	Minimum LOW Pulse Width for latching state into register and generating interrupt	200		200		ns
SYNCA. SYNCB	$t_{dx}(SY)$	Sync Pulse Delay Time from rising edge of RxC. Output Modes	4	7	4	7	φ periods
	$t_{dxec}(SY)$	Sync Pulse Delay Time from rising edge of RxC External Sync Mode		100		100	ns
TxCA. TxCB	$t_c(TxC)$	Transmit Clock Period	400	∞	400	∞	ns
	$t_w(TCH)$	Transmit Clock Pulse Width, clock HIGH	180	∞	180	∞	ns
	$t_w(TCL)$	Transmit Clock Pulse Width, clock LOW	180	∞	180	∞	ns
TxDA. TxDB†	$t_d(TxD)$	TxD Output Delay from falling Edge of TxC (x1 Clock Mode)		400		300	ns
RxCA. RxCB	$t_c(RxC)$	Receive Clock Period	400	∞	400	∞	ns
	$t_w(RCH)$	Receive Clock Pulse Width, clock HIGH	180	∞	180	∞	ns
	$t_w(RCL)$	Receive Clock Pulse Width, clock LOW	180	∞	180	∞	ns
RxDA. RxDB†	$t_s(RxC)$	Setup Time to rising edge of RxC, x1 mode	0		0		ns
	$t_h(RxC)$	Hold Time from rising edge of RxC, x1 mode	140		140		ns

†In all modes, the system clock (φ) rate must be at least 4.5 times the maximum data rate. RESET must be active a minimum of one complete φ cycle.

AC Characteristics (Continued)



Signal	Symbol	Parameter	Z80-SIO Min	Z80-SIO Max	Z80A-SIO Min	Z80A-SIO Max	Unit
INT	$t_{\phi I(IT)}$	INT Delay Time from rising edge of ϕ		200		200	ns
WAIT/READY	$t_{\phi IC(W/R)}$	WAIT/READY Delay Time from IORQ or CE in Wait Mode		180		130	ns
	$t_{\phi H\phi(W/R)}$	WAIT/READY Delay Time from falling edge of ϕ, WAIT/READY HIGH, Wait Mode		150		130	ns
	$t_{\phi Rx(W/R)}$	WAIT/READY Delay Time from rising edge of RxC Data Bit, Ready Mode	10	13	10	13	ϕ periods
	$t_{\phi Tx(W/R)}$	WAIT/READY Delay Time from center of Transmit Data Bit, Ready Mode	5	9	5	9	ϕ periods
	$t_{\phi L\phi(W/R)}$	WAIT/READY Delay Time from rising edge of ϕ, WAIT/READY LOW, Ready Mode		120		120	ns

DC Characteristics

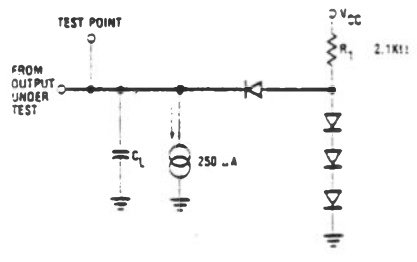
$T_A = 0°C$ to $70°C$, $V_{CC} = +5V, \pm 5\%$

Symbol	Parameter	Min.	Max.	Unit	Test Condition
V_{ILC}	Clock Input Low Voltage	−0.3	+0.45	V	
V_{IHC}	Clock Input High Voltage	$V_{CC}-0.6$	+5.5	V	
V_{IL}	Input Low Voltage	−0.3	+0.8	V	
V_{IH}	Input High Voltage	+2.0	+5.5	V	
V_{OL}	Output Low Voltage		+0.4	V	$I_{OL} = 2.0$ mA
V_{OH}	Output High Voltage	+2.4		V	$I_{OH} = -250$ μA
I_{LI}	Input Leakage Current	−10	+10	μA	$0 \le V_{IN} \le V_{CC}$
I_Z	3-State Output/Data Bus Input Leakage Current	−10	+10	μA	$0 \le V_{IN} \le V_{CC}$
I_{LSY}	SYNC Pin Leakage Current	−40	−10	μA	
I_{CC}	Power Supply Current		100	mA	

Capacitance

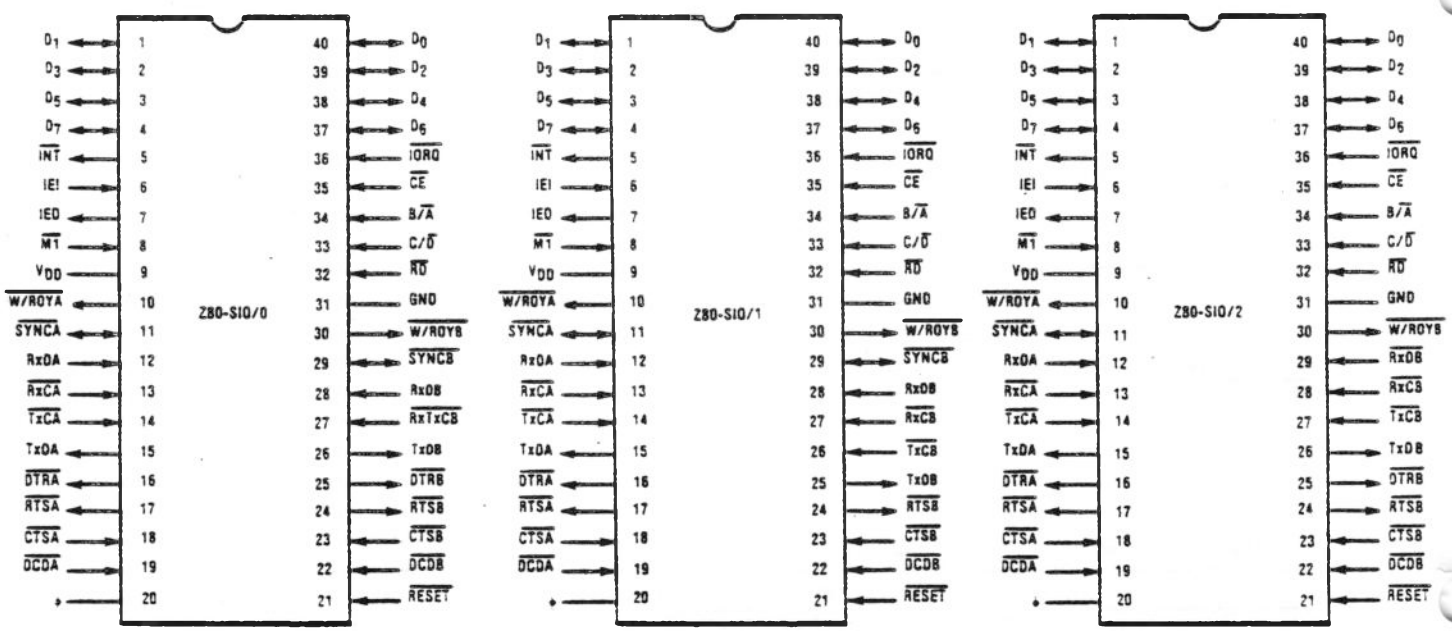
$T_A = 25°C$, $f = 1$ MHz

Symbol	Parameter	Min.	Max.	Unit	Test Condition
C	Clock Capacitance		40	pF	Unmeasured pins returned to ground
C_{IN}	Input Capacitance		5	pF	
C_{OUT}	Output Capacitance		10	pF	

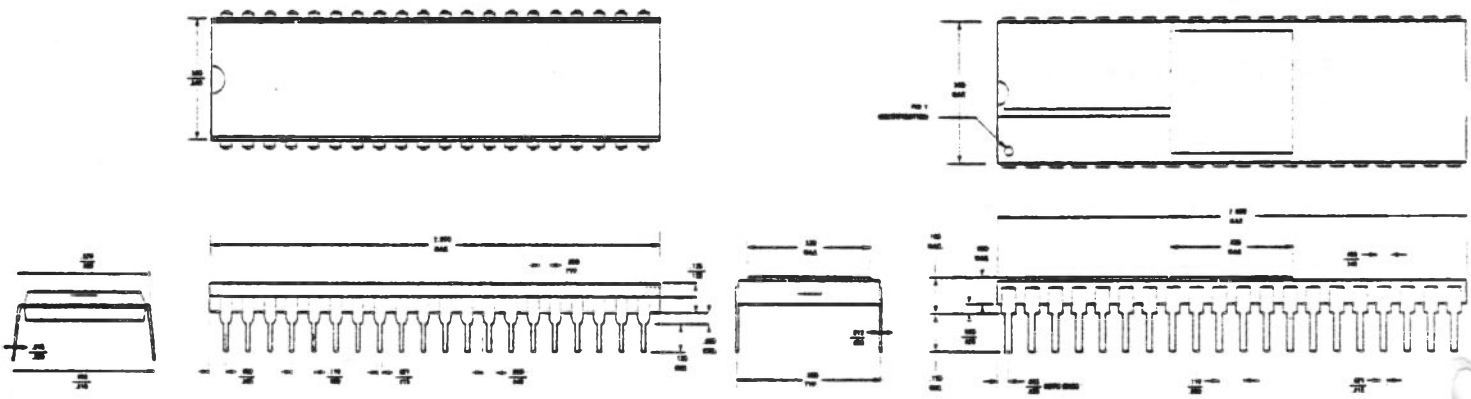


$C_L = 50$ pF. Increase delay by 10 ns for each 50 pF increase in C_L, up to 200 pF maximum.

Package Configurations



Package Outlines



40-Pin Plastic

40-Pin Ceramic

Ordering Information

C — Ceramic
P — Plastic
S — Standard 5V ±5%, 0° to 70°C
E — Extended 5V ±5%, −40° to 85°C
M — Military 5V ±10%, −55° to 125°C
/0 — Type 0 Bonding
/1 — Type 1 Bonding
/2 — Type 2 Bonding

Example:

Z80-SIO/1 CS (Ceramic—Standard Range—Type 1 Bonding)

Z80-SIO/0 PS (Plastic — Standard Range — Type 0 Bonding)

READER'S COMMENTS

Your feedback about this document is important to us: only in this way can we ascertain your needs and fulfill them in the future. Please take the time to fill out this questionnaire and return it to us. This information will be helpful to us, and, in time, to the future users of Zilog systems. Thank you.

Your Name: _____

Company Name: _____

Address: _____

Title of this document: _____

What software products do you have? _____

_____

_____

What is your hardware configuration (including memory size)? _____

_____

_____

Does this publication meet your needs? ☐ Yes ☐ No

If not, why not? _____

_____

_____

How do you use this publication? (Check all that apply)

- ☐ As an introduction to the subject?
- ☐ As a reference manual?
- ☐ As an instructor or student?

How do you find the material?

	Excellent	Good	Poor
Technicality	☐	☐	☐
Organization	☐	☐	☐
Completeness	☐	☐	☐

What would have improved the material? _____

_____

_____

Other comments, suggestions or corrections: _____

_____

_____

If you found any mistakes in this document, please let us know what and where they were:

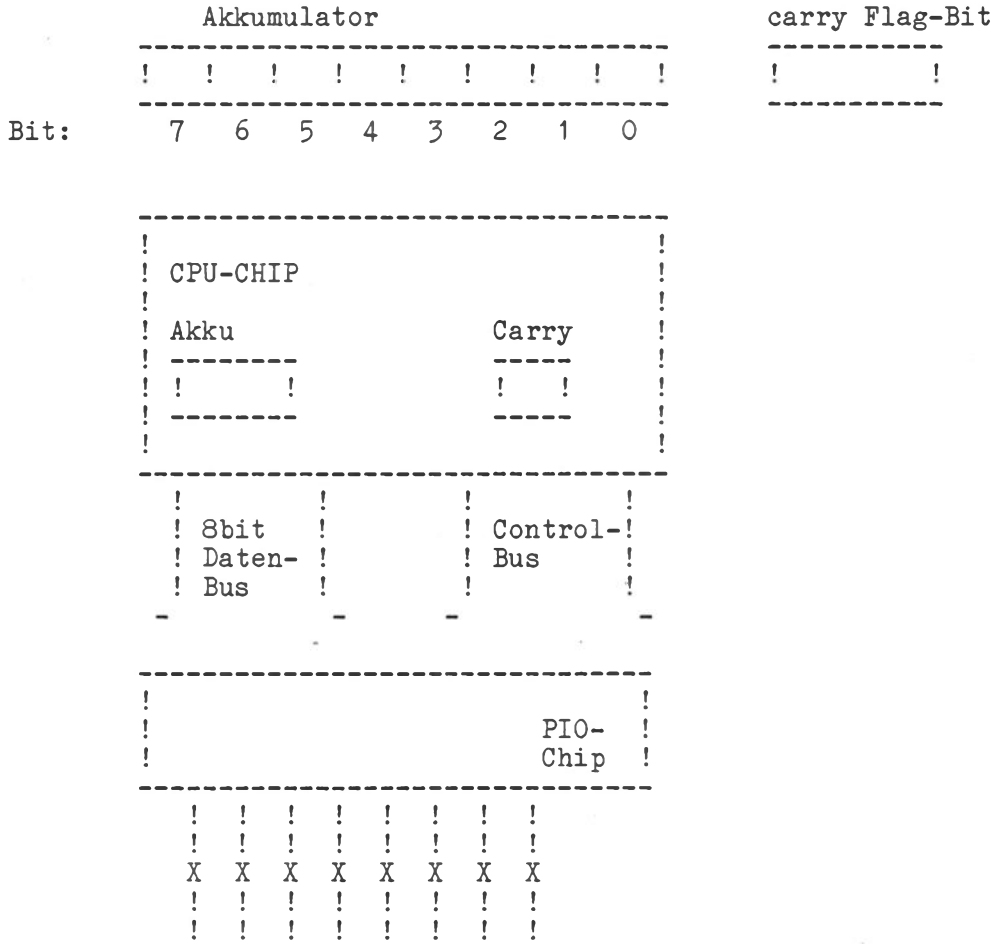
_____

_____

11. BEISIELPROGRAMME

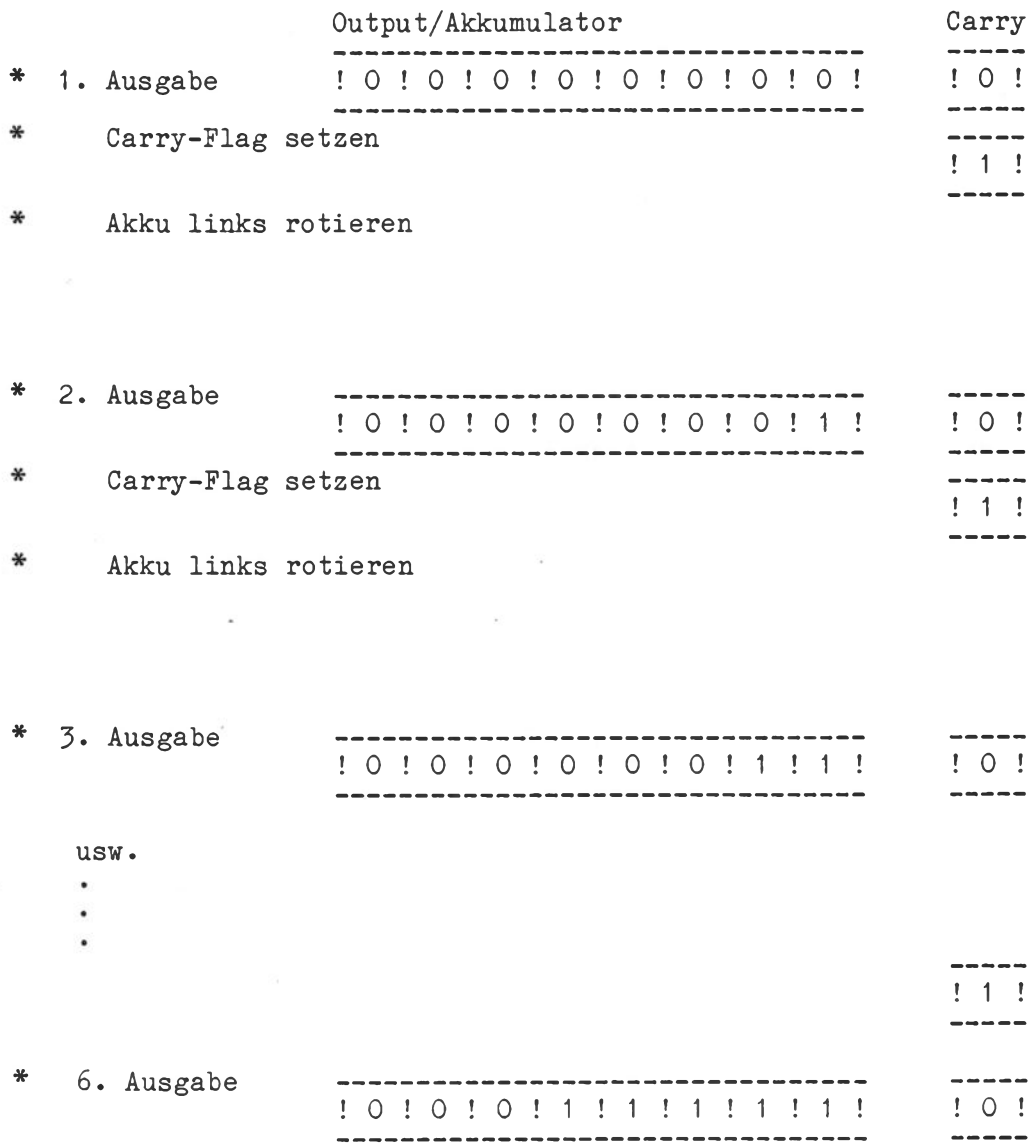
VORFÜHRUNGSPROGRAMM LAMPENSTEUERUNG

Datenfluß



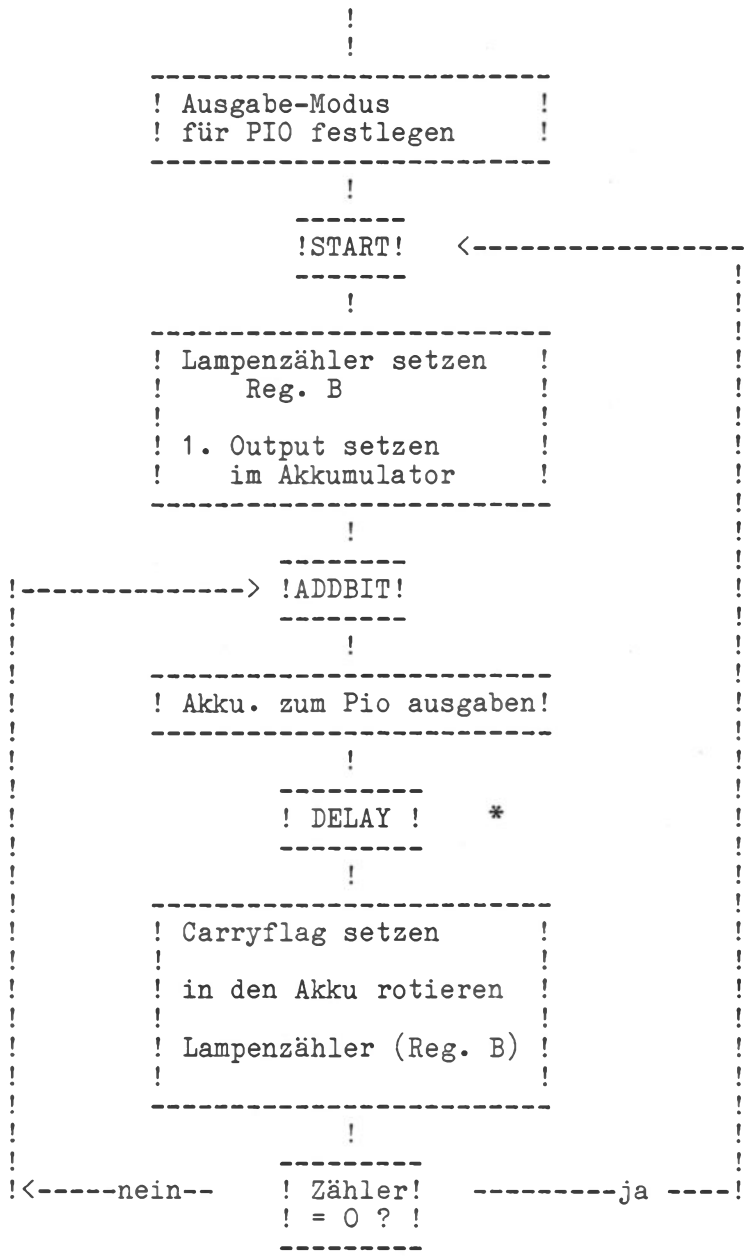
VORFÜHRUNGSPROGRAMM LAMPENSTEUERUNG

Datenmanipulation in der CPU



VORFÜHRUNGSPROGRAMM LAMPENSTEUERUNG

Programm Flußdiagramm



VORFÜHRUNGSPROGRAMM LAMPENSTEUERUNG

Programm Flußdiagramm
für DELAY


```
LOC   OBJ CODE M STMT SOURCE STATEMENT
                    1
                    2
                    3  ; Vorfuehrprogramm fuer ECB oder KIT und Lampstecker
                    4  ; ================================================
                    5
                    6  ; Ansteuerung von 8 LEDs ueber einen PIO-Port
                    7  ; Mode : Output
                    8  ;
                   11  *LIST ON
                   12
                   13
                   14
                   15  INI:
3C50   3E0F        16          LD    A,0FH     ; PORT A IN
3C52   D30A        17          OUT   (0AH),A   ; OUTPUT MODE
                   18
                   19  START:
3C54   0609        20          LD    B,9       ; LAMPENZAEHLER INITIALISIEREN
3C56   3E00        21          LD    A,0       ; AUSZUGEB. BYTE = 00
                   22                          ; ==> ANZEIGE LOESCHEN
                   23
                   24  ADDBIT:
3C58   D308        25          OUT   (08H),A   ; NAECHSTES BYTE AUSGEBEN
3C5A   16FF        26          LD    D,255     ; AEUSS. ZAEHLSCHLEIFE INITIAL
                   27
                   28  DEL1:
3C5C   1E32        29          LD    E,50      ; INNERE ZAEHLSCHLEIFE INITIAL
                   30
                   31  DEL2:
3C5E   1D          32          DEC   E         ; INNERE SCHLEIFE RUNTERZ.
3C5F   20FD        33          JR    NZ,DEL2
3C61   15          34          DEC   D         ; AEUSSERE SCHL. RUNTERZ.
3C62   20F8        35          JR    NZ,DEL1
3C64   37          36          SCF             ; CARRY FLAG SETZEN
3C65   CB17        37          RL    A         ; ACCU NACH LINKS SCHIEBEN UND
                   38                          ; CARRY BIT AUF BIT 0 VON ACCU
3C67   10EF        39          DJNZ  ADDBIT    ; LAMPENZAEHLER RUNTERZAEHLEN
3C69   C3543C      40          JP    START
                   41  END
```

AUFGABENSTELLUNG:

Die folgenden Programme sind der Reihe nach erweiterbar, wobei Teile aus bereits erstellten Programmteilen wiederverwendet werden können, so daß der Programmieraufwand in Grenzen gehalten wird.

Eine nähere Spezifizierung der Aufgabenstellung entfällt. Dies hat den Vorteil, daß bei der Programmerstellung andere Wege als beim vorgefertigten Beispielprogramm beschritten werden können und somit auch der Lerneffekt größer ist.
Als Hilfestellung können jedoch die entsprechenden Flußdiagramme verwendet werden.

LAEMPLI.1

Es ist ein Programm zu entwerfen, das eine Leuchtdiodenzeile (8 LED) über den PIO Kanal A der ECB/C8-Karte ansteuert.

PIO-Kanal-A-Daten = Adresse-00H
PIO-Kanal-A-Control-Adresse 02H

Zu realisieren ist ein Lauflicht mit einem durchlaufenden Lichtpunkt. Lineare Programmierung im Hauptprogramm. Keine Verwendung eines Unterprogrammes.
Verwendung einer Zeitschleife.

LAEPMLI.2

Aufgabenstellung siehe LAEMPLI.1

Abweichend davon soll die Zeitschleife mit Verschiebeoperation durch ein Unterprogramm realisiert werden. Die Verzögerungszeit der Zeitschleife soll über eine Variable (über DEFB) einstellbar sein.

LAEMPLI.3

Aufgabenstellung siehe LAEMPLI.2

Abweichend davon soll sich das Lauflicht nicht nur in eine Richtung bewegen, sondern beim Erreichen des Endes der Zeile in die jeweils andere Richtung weiterlaufen.
Realisierung durch zwei Unterprogramme.

LAEMPLI.4

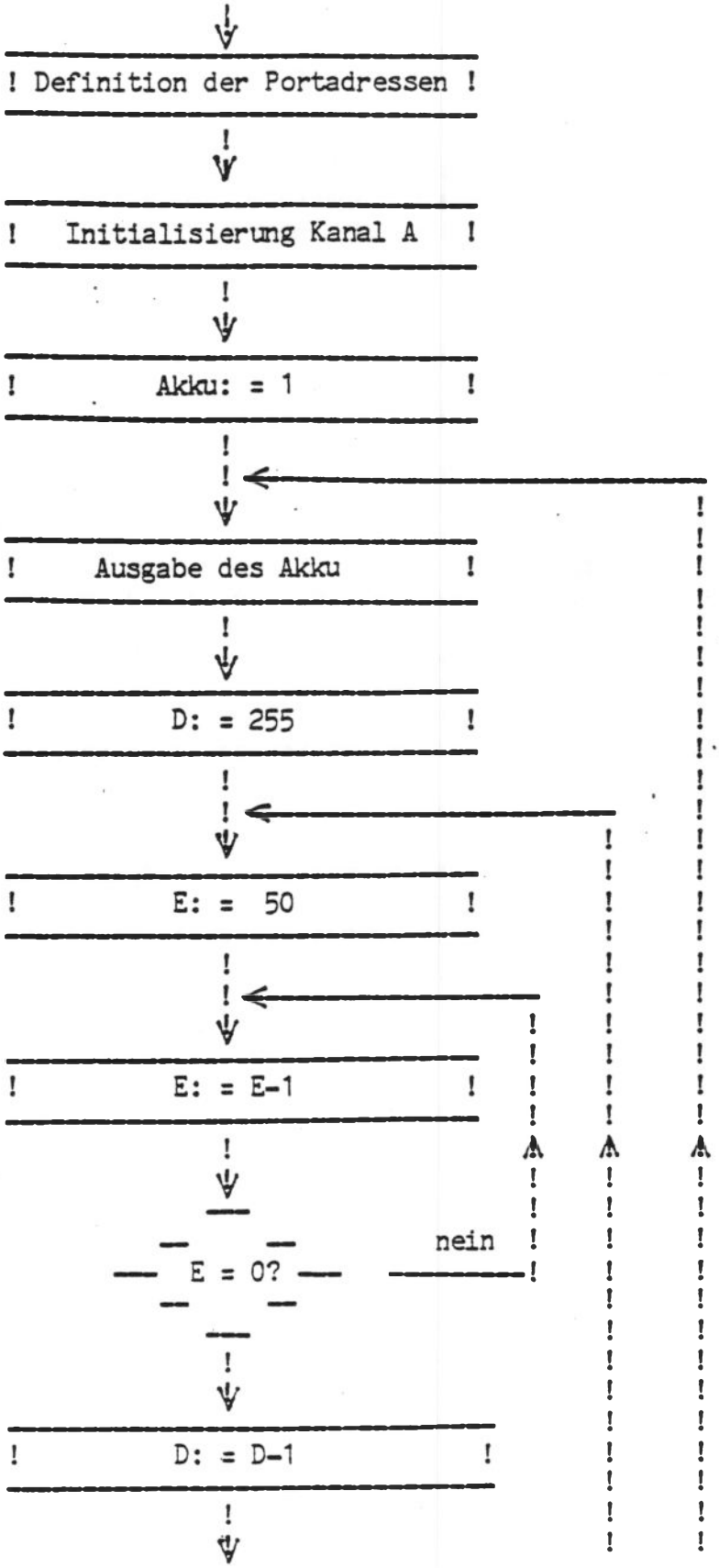
Aufgabenstellung siehe LAEMPLI.3

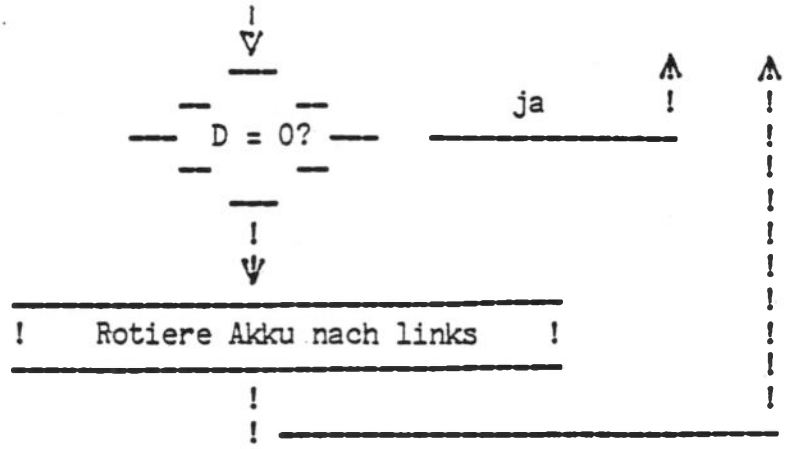
Abweichend davon soll nach jedem Anstoßen des Leuchtpunktes am Ende der Zeile die Fortbewegungsgeschwindigkeit zunehmen, bis ein Maximum erreicht ist. Dann verringert sich die Geschwindigkeit wieder bis zu einem Minimum usw.

Dies soll durch ein weiteres Unterprogramm ACC realisiert werden, das aus dem Unterprogramm UP bzw. DOWN aufgerufen wird.

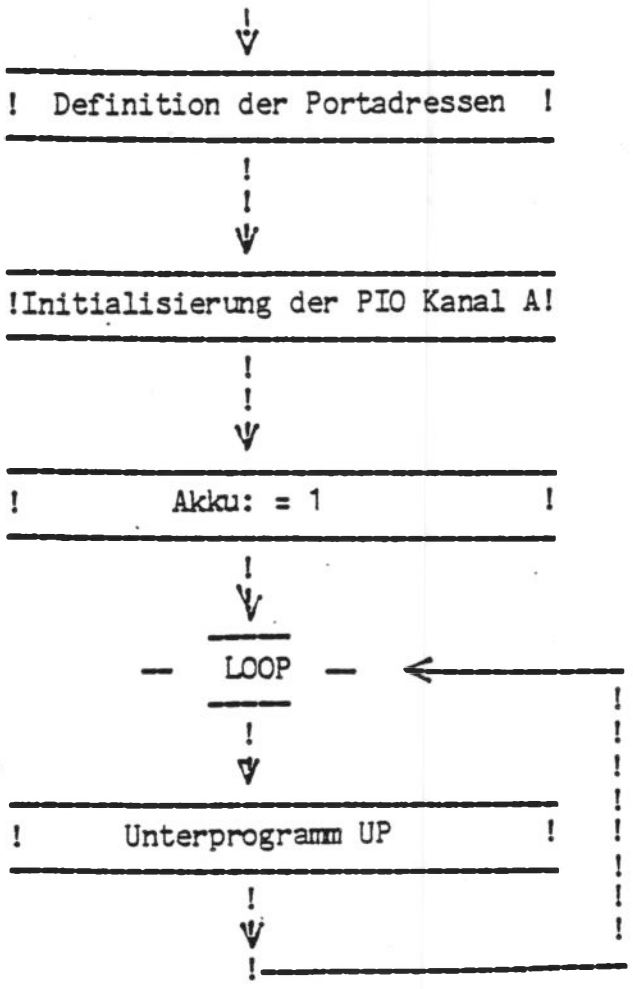
Anmerkung: Durch verschiedene Inhalte des Akkus A (wird über Leuchtdiodenzeile angezeigt) am Anfang des Programmes, können verschiedene Durchlaufmuster erzeugt werden (Eingabe des Anfangsakkuinhaltes über Programm).

F L U S S D I A G R A M M : LAEMPLI.1

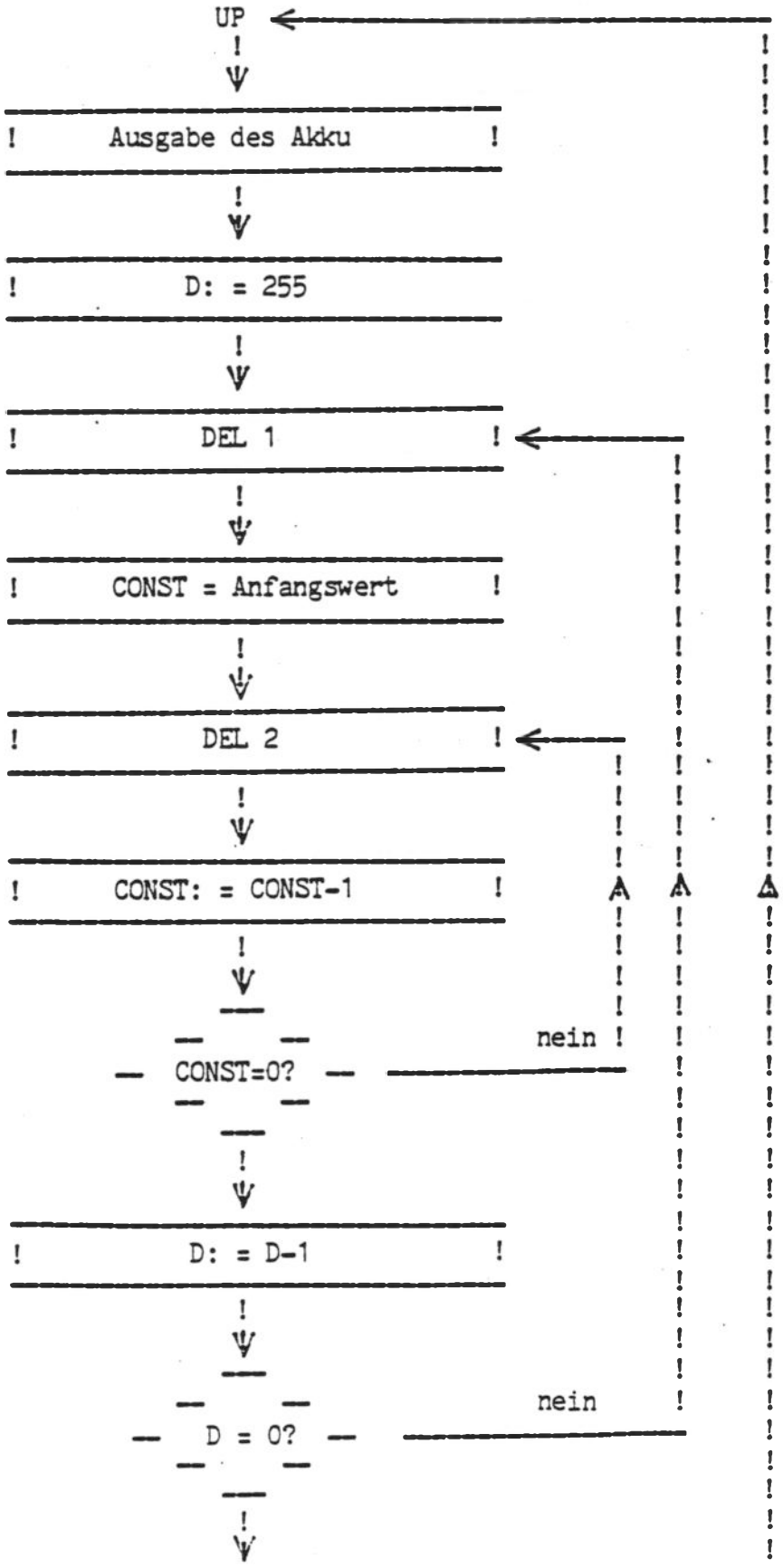


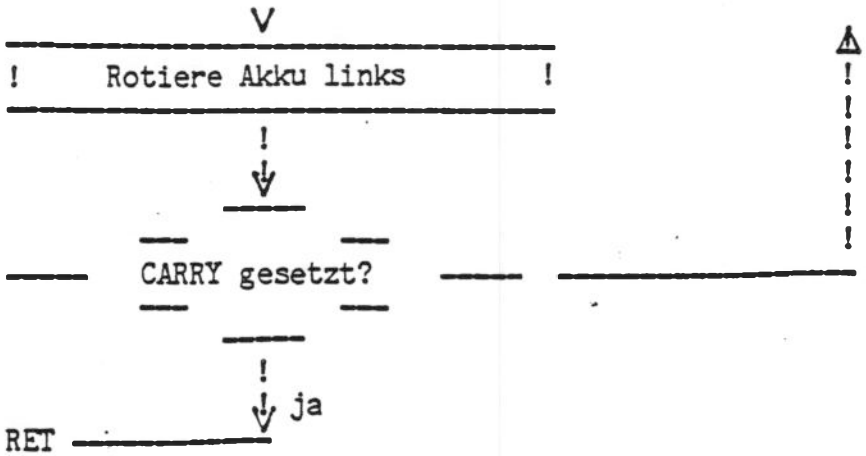

D = 0?
ja
Rotiere Akku nach links

F L U S S D I A G R A M M : LAEMPLI.2

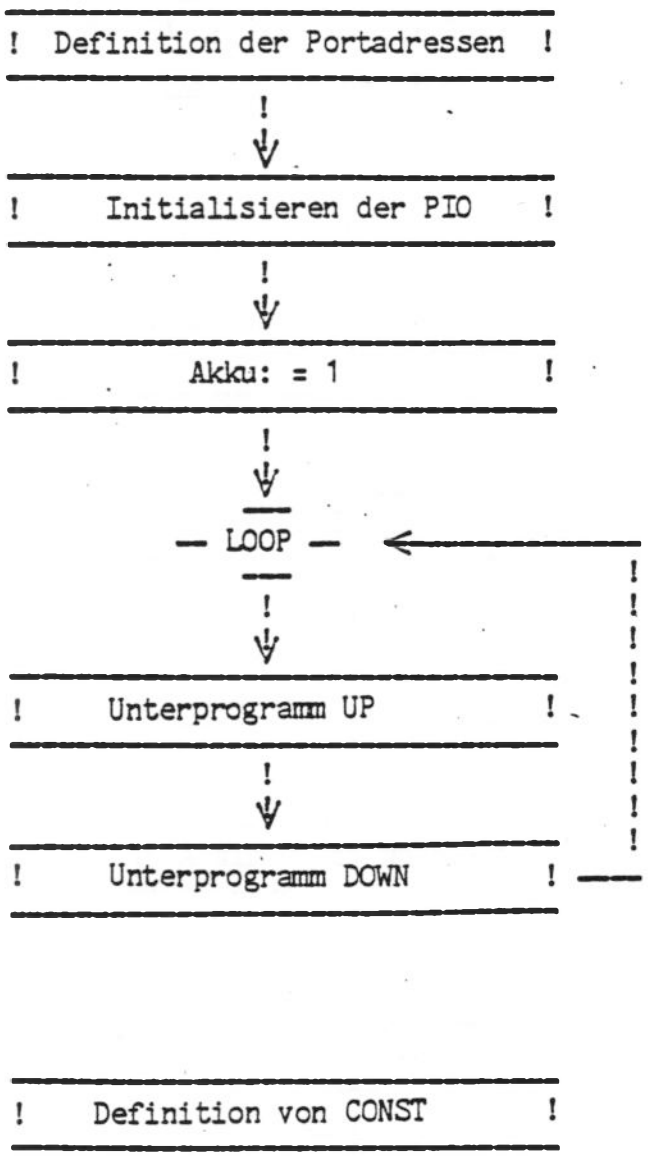


Definition von CONST


UP
Ausgabe des Akku
D: = 255
DEL 1
CONST = Anfangswert
DEL 2
CONST: = CONST-1
CONST=0?
nein
D: = D-1
D = 0?
nein


Rotiere Akku links
CARRY gesetzt?
ja
RET

F L U S S D I A G R A M M : LAEMPLI.3



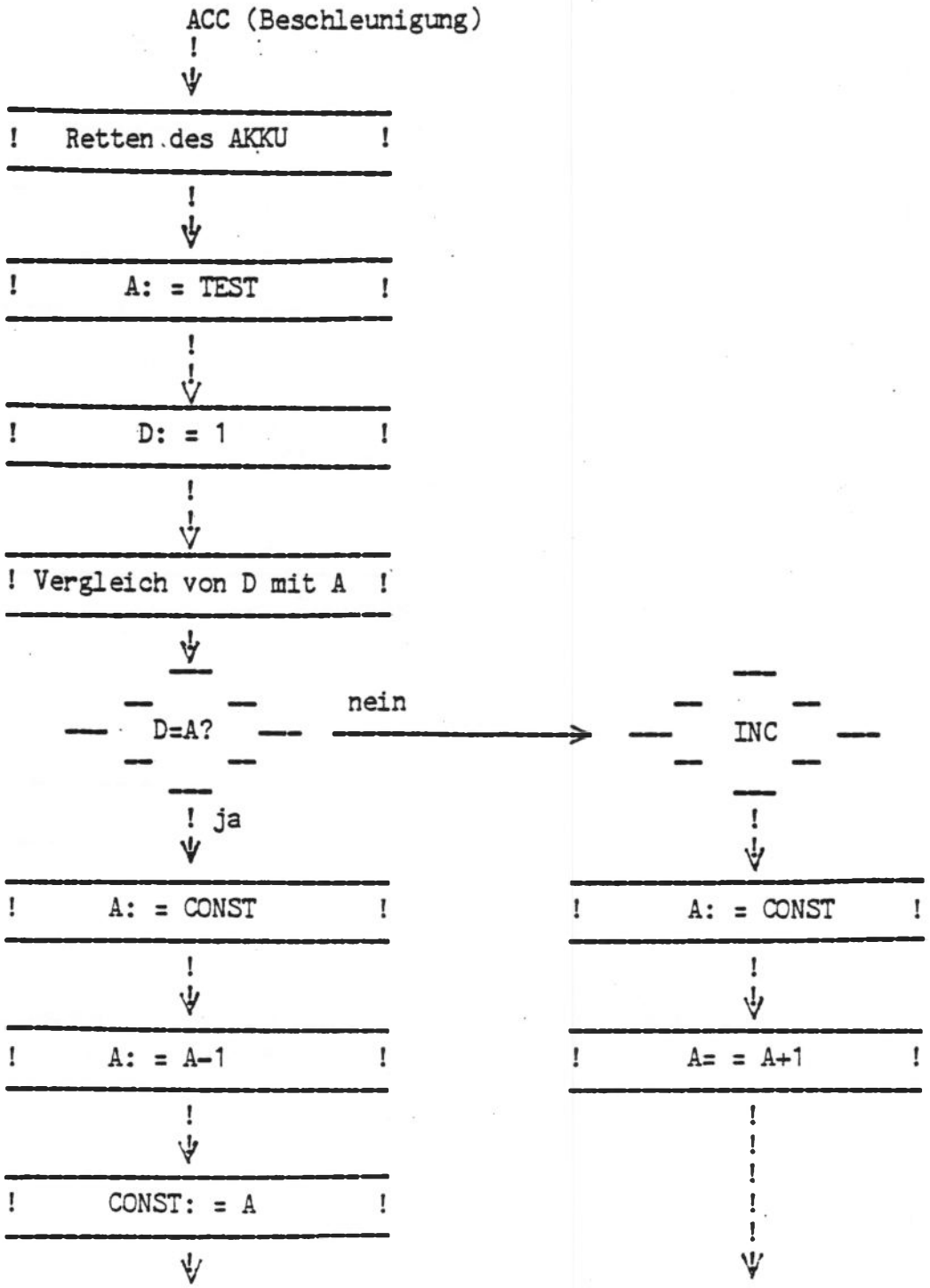
Unterprogramm UP identisch mit dem unter LAEMPLI.2
Unterprogramm DOWN identisch mit dem Unterprogramm UP, mit folgenden Abweichungen:

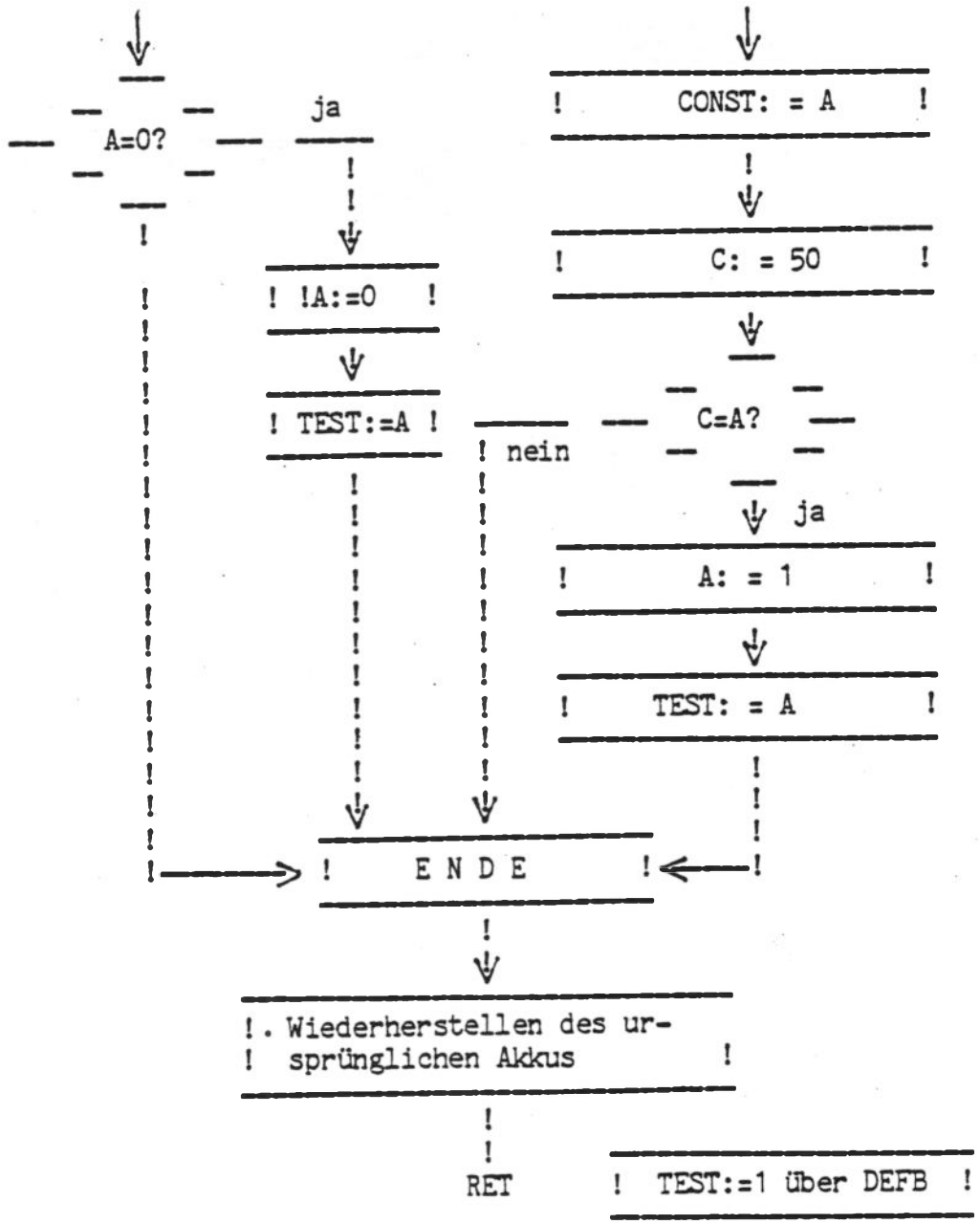
* anstatt RLA steht RRA
* DEL1 ——> DEL3
* DEL2 ——> DEL4

F L U S S D I A G R A M M : LAEMPLI.4

Hauptprogramm identisch mit dem unter LAEMPLI.3
Unterprogramme UP und DOWN identisch mit denen unter LAMEPLI.3, außer:

* vor RET jeweils Sprung ins Unterprogramm ACC




A=0?
ja
!A:=0
TEST:=A
nein
CONST: = A
C: = 50
C=A?
ja
A: = 1
TEST: = A
E N D E
. Wiederherstellen des ur-
sprünglichen Akkus
RET
TEST:=1 über DEFB

```
                        1  *H DEMO-PROGRAMM FUER ECB/C8
                        2  ;==========================
                        3  PAC:    EQU 02H          ;I/O PORT A CONTROL INFO-ADR
                        4  PAD:    EQU 00H          ;I/O PORT A DATEN - ADR
0000   3E0F             5          LD    A,0FH      ;PORT A IN
0002   D302             6          OUT   (PAC),A    ;OUTPUT MODE
0004   3E01             7          LD    A,1        ;AUSZUGEB. BYTE = 01
                        8  ADDBIT:
0006   D300             9          OUT   (PAD),A    ;NAECHSTES BYTE AUSGEBEN
0008   16FF            10          LD    D,255      ;AEUSS. ZAEHLSCHLEIFE INITIAL
                       11  DEL1:
000A   1E32            12          LD    E,50       ;INNERE ZAEHLSCHLEIFE INITIAL
                       13  DEL2:
000C   1D              14          DEC   E          ;INNERE SCHLEIFE RUNTERZ.
000D   20FD            15          JR    NZ,DEL2
000F   15              16          DEC   D          ;AEUSSERE SCHL. RUNTERZ.
0010   20F8            17          JR    NZ,DEL1
0012   17              18          RLA              ;ACCU NACH LINKS SCHIEBEN UND
0013   18F1            19          JR ADDBIT        ;CARRY BIT AUF BIT 0 VON ACCU
```

```
DEMO-PROGRAMM FUER ECB/C8      LAEMPLI.2                          PAGE   1
 LOC   OBJ CODE M STMT SOURCE STATEMENT                           ASM 5.9

                      1  *H DEMO-PROGRAMM FUER ECB/C8
                      2  ;============================
                      3  PAD:    EQU     0       ;I/O-PORT A DATA ADRESSE

                      4  PAC:    EQU     2       ;I/O-PORT A CONTROL ADRESSE
0000   310065         5          LD SP,6500H     ;Laden des Stackpointers
                      6                          ;INITIALISIERUNG der PIO
0003   3E0F           7          LD      A,0FH   ;Port A in Output-Mode
0005   D302           8          OUT (PAC) ,A
0007   3E01           9          LD A,1          ;Erstes auszugebendes Muster
                     10                          ;am Lampenstecker
                     11  LOOP:                   ;Beginn des Hauptprogrammes
0009   CD0E00   R    12          CALL UP         ;Zum Unter.pgm UP(herauf-
                     13                          ;zaehlen)
000C   18FB         14          JR LOOP         ;Sprung auf LOOP---->Endlos-
                     15                          ;schleife
                     16                          ;******************************
                     17  UP:                     ;Linkslauf-Unter.pgm(herauf)
000E   D300         18          OUT (PAD),A     ;naechstes Byte ausgeben
0010   16FF         19          LD D,255        ;INITIALISIERUNG der aeusse-
                     20                          ;ren Zeitschleife(Anfangswert)
0012   4F           21          LD C,A          ;Abspeichern des Akkuinhaltes
                     22  DEL1
0013   3A2100   R    23          LD A,(CONST)    ;INIT. innere Zeitschleife
                     24  DEL2
0016   3D           25          DEC A           ;Innere Zeitschleife herunter-
                     26                          ;zaehlen
0017   20FD         27          JR NZ,DEL2      ;Sprung,wenn Akku nicht 0
0019   15           28          DEC D           ;Aeussere Zeitschleife her-
                     29                          ;unterzaehlen
001A   20F7         30          JR NZ,DEL1      ;Sprung,wenn D nicht 0
001C   79           31          LD A,C          ;Urspruenglichen Akkuinhalt
                     32                          ;wiederherstellen
001D   17           33          RLA             ;Linksrotieren des Akku
001E   30EE         34          JR NC,UP        ;Sprung,wenn Carry nicht ge-
                     35                          ;setzt
0020   C9           36          RET             ;Ende des Unter.pgm UP
                     37                          ;******************************
0021   32           38  CONST:  DEFB    50      ;Anfangswert fuer Zeit-
                     39                          ;schleife Innen
```

```
DEMO-PROGRAMM FUER ECB/C8     LAEMPLI.3                          PAGE   1
 LOC   OBJ CODE M STMT SOURCE STATEMENT                           ASM 5.9

                      1  *H DEMO-PROGRAMM FUER ECB/C8
                      2  ;==========================
                      3  PAD:    EQU     0        ;I/O-Port Data Adresse
                      4  PAC:    EQU     2        ;I/O-Port Control Adresse
0000   310065         5          LD SP,6500H      ;Laden des Stackpointers
                      6                           ;INITIALISIERUNG der PIO
0003   3E0F           7          LD A,0FH         ;Port A in Output-Mode
0005   D302           8          OUT (PAC) ,A
0007   3E01           9          LD A,1           ;erstes auszugebendes Muster
                      10 LOOP:
0009   CD1100    R    11         CALL UP          ;zum Sub.pgm Linksrotieren
000C   CD2400    R    12         CALL DOWN        ;zum Sub.pgm Rechtsrotieren
000F   18F8           13         JR LOOP          ;Endlosschleife
                      14                          ;*****************************
                      15 UP:                      ;Linksrotieren-Sub.pgm
0011   D300           16         OUT (PAD),A      ;naechstes Byte ausgeben
0013   16FF           17         LD D,255         ;Initialisierung der aeusse-
                      18                          ;ren Zeitschleife(Anfangswert)
0015   4F             19         LD C,A           ;Abspeichern des Akkuinhaltes
                      20 DEL1
0016   3A3700    R    21         LD A,(CONST)     ;Anfangswert der Zeitschleife
                      22 DEL2
0019   3D             23         DEC A            ;innere Zeitschleife herunter-
                      24                          ;zaehlen
001A   20FD           25         JR NZ,DEL2       ;Sprung,wenn Akku nicht Null
001C   15             26         DEC D            ;aeussere Zeitschleife
                      27                          ;herunterzaehlen
001D   20F7           28         JR NZ,DEL1       ;Sprung,wenn d nicht NullD
001F   79             29         LD A,C           ;Urspruenglichen Akkuinhalt
                      30                          ;wiederherstellen
0020   17             31         RLA              ;Linksrotieren des Akku
0021   30EE           32         JR NC,UP         ;Sprung,wenn Carry nicht
                      33                          ;gesetzt
0023   C9             34         RET              ;Ende des Sub.pgm UP
                      35                          ;*****************************
                      36
```

```
                        37
                        38
                        39                          ;*****************************
                        40  DOWN:                   ;Sub.pgm Rechtsrotieren
                        41                          ;Initialisierung der PIO
0024   D300             42          OUT (PAD),A     ;naechstes Byte ausgeben
0026   16FF             43          LD D,255        ;Initialisiere aeussere Zeit-
                        44                          ;schleife(Anfangswert)
0028   4F               45          LD C,A          ;Retten des Akkuinhaltes
                        46  DEL3
0029   3A3700    R      47          LD A,(CONST)    ;INIT. innere Zeitschleife
                        48  DEL4
002C   3D               49          DEC A           ;Innere Zeitschleife herunter-
                        50                          ;zaehlen
002D   20FD             51          JR NZ,DEL4      ;Sprung,wenn A nicht 0
002F   15               52          DEC D           ;Aeussere Zeitschleife
                        53                          ;herunterzaehlen
0030   20F7             54          JR NZ,DEL3      ;Sprung,wenn D nicht 0
0032   79               55          LD A,C          ;Zurueckholen des Akkuinhalts
0033   1F               56          RRA             ;Rechtsrotieren des Akku A
0034   30EE             57          JR NC,DOWN      ;Sprung nach DOWN,wenn Carry
                        58                          ;nicht gesetzt
```



```
0036   C9               59          RET             ;Zurueck zum Haupt.pgm
                        60                          ;*****************************
0037   32               61  CONST:  DEFB    50      ;Anfangswert fuer Zeit-
                        62                          ;schleife innen
```

```
DEMO-PROGRAMM FUER ECB/C8      LAEMPLI.4                          PAGE   1
 LOC   OBJ CODE M STMT SOURCE STATEMENT                           ASM 5.9

                        1  *H DEMO-PROGRAMM FUER ECB/C8
                        2  ;==========================
                        3  PAD:    EQU     0        ;I/O-Port Data Adresse
                        4  PAC:    EQU     2        ;I/O-Port Control Adresse
0000   310065           5          LD SP,6500H      ;Laden des Stackpointers
                        6                           ;INITIALISIERUNG der PIO
0003   3E0F             7          LD A,0FH         ;Port A in Output-Mode
0005   D302             8          OUT (PAC) ,A
0007   3E01             9          LD A,1           ;erstes auszugebendes Muster
                       10  LOOP:
0009   CD1100    R     11          CALL UP          ;zum Sub.pgm Linksrotieren
000C   CD2700    R     12          CALL DOWN        ;zum Sub.pgm Rechtsrotieren
000F   18F8            13          JR LOOP          ;Endlosschleife
                       14                           ;*****************************
                       15  UP:                      ;Linksrotieren-Sub.pgm
0011   D300            16          OUT (PAD),A      ;naechstes Byte ausgeben
0013   16FF            17          LD D,255         ;Initialisierung der aeusse-
                       18                           ;ren Zeitschleife(Anfangswert)
0015   4F              19          LD C,A           ;Abspeichern des Akkuinhaltes
                       20  DEL1
0016   3A6B00    R     21          LD A,(CONST)     ;Anfangswert der Zeitschleife
                       22  DEL2
0019   3D              23          DEC A            ;innere Zeitschleife herunter-
                       24                           ;zaehlen
001A   20FD            25          JR NZ,DEL2       ;Sprung,wenn Akku nicht Null
001C   15              26          DEC D            ;aeussere Zeitschleife
                       27                           ;herunterzaehlen
001D   20F7            28          JR NZ,DEL1       ;Sprung,wenn d nicht NullD
001F   79              29          LD A,C           ;Urspruenglichen Akkuinhalt
                       30                           ;wiederherstellen
0020   17              31          RLA              ;Linksrotieren des Akku
0021   30EE            32          JR NC,UP         ;Sprung,wenn Carry nicht
                       33                           ;gesetzt
0023   CD3D00    R     34          CALL ACC         ;zum Sub.pgm Beschleunigung
0026   C9              35          RET              ;Ende des Sub.pgm UP
                       36                           ;*****************************
                       37
```

```
                        38
                        39
                        40                      ;*******************************
                        41  DOWN:               ;Sub.pgm Rechtsrotieren
                        42                      ;Initialisierung der PIO
0027   D300             43          OUT (PAD),A ;naechstes Byte ausgeben
0029   16FF             44          LD D,255    ;Initialisiere aeussere Zeit-
                        45                      ;schleife(Anfangswert)
002B   4F               46          LD C,A      ;Retten des Akkuinhaltes
                        47  DEL3
002C   3A6B00   R        48          LD A,(CONST) ;INIT. innere Zeitschleife
                        49  DEL4
002F   3D               50          DEC A       ;Innere Zeitschleife herunter-
                        51                      ;zaehlen
0030   20FD             52          JR NZ,DEL4  ;Sprung,wenn A nicht 0
0032   15               53          DEC D       ;Aeussere Zeitschleife
                        54                      ;herunterzaehlen
0033   20F7             55          JR NZ,DEL3  ;Sprung,wenn D nicht 0
0035   79               56          LD A,C      ;Zurueckholen des Akkuinhalts
0036   1F               57          RRA         ;Rechtsrotieren des Akku A
0037   30EE             58          JR NC,DOWN  ;Sprung nach DOWN,wenn Carry
```

```
 LOC   OBJ CODE M STMT SOURCE STATEMENT                              ASM 5.9

                   59                      ;nicht gesetzt
0039   CD3D00   R  60          CALL ACC    ;zum Sub.pgm Beschleunigung
003C   C9          61          RET         ;Zurueck zum Haupt.pgm
                   62                      ;******************************
                   63
                   64
                   65                      ;******************************
                   66  ACC:                ;Unterprogramm Beschleunigung
003D   F5          67          PUSH AF     ;Retten des Akkus
003E   3A6A00   R  68          LD A,(TEST) ;Vergleichsoperator. Bei
0041   1601        69          LD D,1      ;A=1--->schneller/ A=0--->
0043   BA          70          CP D        ;--->langsamer
0044   2011        71          JR NZ,INC   ;Sprung,wenn A nicht D
0046   3A6B00   R  72          LD A,(CONST)
0049   3D          73          DEC A       ;neuer Anfangszahlenwert
                   74                      ;von (CONST)
004A   326B00   R  75          LD (CONST),A
004D   3D          76          DEC A
004E   2005        77          JR NZ,ENDE  ;Sprung,wenn CONST
0050   3E00        78          LD A,0      ;noch nicht 1
0052   326A00   R  79          LD (TEST),A ;Aenderung von TEST falls A=1
                   80  ENDE
0055   F1          81          POP AF
0056   C9          82          RET         ;Ende des Sub.pgm
                   83                      ;Beschleunigung
                   84
                   85
                   86  INC:
0057   3A6B00   R  87          LD A,(CONST)
005A   3C          88          INC A       ;Erhoehen des Wertes
005B   326B00   R  89          LD (CONST),A ;von CONST
005E   0E32        90          LD C,50
0060   B9          91          CP C        ;Abfrage auf Erreichen des
                   92                      ;maximalen Wertes
0061   20F2        93          JR NZ,ENDE  ;Sprung,wenn A nicht C
0063   3E01        94          LD A,1
0065   326A00   R  95          LD (TEST),A
0068   18EB        96          JR ENDE
                   97                      ;******************************
006A   01          98  TEST    DEFB   01
006B   32          99  CONST   DEFB   50
                  100
```

```
                                WUERFEL         09.12.80..BA        PAGE   1
LOC   OBJ CODE M STMT SOURCE STATEMENT                               ASM 5.8

                          1
                          2          ; Spielprogramm fuer ECB-C8 und Lampstecker
                          3          ; ==========================================
                          4
                          5
                          6          ; Vereinbarung externer Adressen
                          7          ; ------------------------------
                          8
                          9          EXTERNAL TABLE  ; Interrupt-Tabelle in einem
                         10                          ; anderem Programm realisiert
                         11          EXTERNAL MUSTER ; Tabelle MUSTER in einem
                         12                          ; anderem Programm
                         13
                         14
                         15          ; Vereinbarung allgemeiner Groessen
                         16          ; ----------------------------------
                         17
                         18 PAC      EQU     02      ; Kontrol-Adresse PIO-Port A
                         19 PAD      EQU     00      ; Daten-Adresse PIO-Port A
                         20
                         21 STACK    EQU   1400H     ; Stack-Oberkante
                         22
                         23
                         24
                         25 INIT:                    ; Initialisierung
                         26
0000   DD212A00 R       27          LD IX,ISR       ; Adr. der Interrupt-Service-
                         28                          ; Routine ins Register IX !
0004   DD220000 X       29          LD (TABLE),IX   ; diese Adr. im Speicher bei
                         30                          ; TABLE ablegen !
                         31                          ; Damit ist die Interrupt-
                         32                          ; Tabelle geladen.
0008   210000   X       33          LD HL,TABLE     ; Nun die Adr. der Tabelle ins
                         34                          ; HL-Registerpaar
000B   7C               35          LD A,H          ; Den hoeherwertigen Adressteil
000C   ED47             36          LD I,A          ; ins Register I der CPU
000E   7D               37          LD A,L          ; und den niederwertigen Teil
000F   D302             38          OUT (PAC),A     ; ins I-Vektor-Reg. des Ports
                         39
0011   3EFF             40          LD A,0FFH       ; Anweisung "FF" = BIT-Mode
0013   D302             41          OUT (PAC),A     ; an den Port ausgeben.
0015   3EC0             42          LD A,0C0H       ; Maske zur Festlegung der
                         43                          ; Leitungen auf IN oder OUT
0017   D302             44          OUT (PAC),A     ; an den Port ausgeben
                         45                          ; ---> D0 ... D5 = OUTPUT
                         46                          ;      D6 und D7 = INPUT
```

```
                        47
0019    3E97            48          LD A,97H        ; Interrupt-Steuerwort
001B    D302            49          OUT (PAC),A     ; an den Port ausgeben
                        50                          ; ---> Interrupt enabled
                        51                          ;      OR-Funktion
                        52                          ;      Ausloesung durch LOW
                        53                          ;      Maske folgt
001D    3E7F            54          LD A,7FH        ; Maske zur Auswahl der inter=
                        55                          ; ruptfaehigen Input-Leitungen
001F    D302            56          OUT (PAC),A     ; an den Port ausgeben
                        57                          ; ---> nur Leitung D7 ist
                        58                          ;      interruptfaehig.
```

```
LOC   OBJ CODE  M STMT SOURCE STATEMENT

                    59
                    60
0021  310014        61          LD SP,STACK     ; Stackpointer setzen
0024  ED5E          62          IM 2            ; Interrupt-Mode 2 waehlen
0026  FB            63          EI              ; CPU fuer Interrupt freigeben
                    64
                    65
                    66
                    67  HAUPT:                  ; Hauptprogramm
                    68
0027  76            69          HALT            ; Warten im HALT-Zustand
0028  18FD          70          JR HAUPT        ; wenn aus Interrupt-Service-
                    71                          ; Routine zurueck, erneut
                    72                          ; warten !
                    73
                    74
                    75
                    76
                    77  ISR:                    ; Interrupt-Service-Routine
                    78
002A  ED5F          79          LD A,R          ; Refresh-Reg. = Zufallswert
002C  0607          80          LD B,07H        ; da nur 6 Zahlen erwuenscht,
002E  A0            81          AND B           ; Maskierung mit 00000III.
                    82                          ; ---> Rest = 8 Moeglichkeiten
                    83                          ; ---> z.B. 06 und 07  weg!
002F  FE06          84          CP 06H          ; 06 ?
0031  2004          85          JR NZ,NSEX      ; wenn nicht
0033  3E00          86          LD A,0          ; wenn ja, tausche in 0 um
0035  1806          87          JR AUS          ; und ueberspringe Abfrage
                    88
                    89  NSEX:
0037  FE07          90          CP 07H          ; 07 ?
0039  2002          91          JR NZ,AUS       ; wenn nicht
003B  3E01          92          LD A,1          ; wenn ja, tausche gegen 1
                    93
                    94  AUS:
003D  110000    X   95          LD DE,MUSTER    ; Lade das Reg.paar DE mit der
                    96                          ; Adresse MUSTER (siehe unten)
0040  2600          97          LD H,0          ; loesche das H-Register
0042  6F            98          LD L,A          ; Zufallswert (0..5) ins L-Reg.
0043  19            99          ADD HL,DE       ; Adersse MUSTER + Zufallswert
0044  7E           100          LD A,(HL)       ; Inhalt der Speicherzelle
                    101                         ; mit obig errechneter Adr.
                    102                         ; ins A-Register,
0045  D300          103          OUT (PAD),A     ; und von dort zum Port !
                    104
0047  FB            105          EI              ; erneute Freigabe des Interr.
0048  ED4D          106          RETI            ; Rueckkehr ins Hauptprogramm
                    107
                    108
                    109
                    110
```

```
                                WUERFELTAB      06.02.80..BA        PAGE   1
LOC   OBJ CODE M STMT SOURCE STATEMENT                               ASM 5.8

                        1
                        2           ; Zusatzprogramm zu WUERFEL
                        3           ; ==========================
                        4
                        5           ; enthaelt die Interrupt-Tabelle TABLE
                        6           ; sowie die Tabelle MUSTER
                        7
                        8
                        9           ; Vereinbarung globaler Adressen
                       10           ; ================================
                       11
                       12           GLOBAL  TABLE
                       13           GLOBAL  MUSTER
                       14
                       15
                       16   TABLE:                  ; Interrupt-Tabelle
0000                   17           DEFS 2          ; zwei reservierte Plaetze
                       18
                       19
                       20
                       21   MUSTER:                 ; Tabelle der Ausgabemuster
0002   01              22           DEFB    01H
0003   03              23           DEFB    03H
0004   07              24           DEFB    07H
0005   0F              25           DEFB    0FH
0006   1F              26           DEFB    1FH
0007   3F              27           DEFB    3FH     ; entspricht 1,2....6 auf HIGH
                       28                           ; gesetzten Output-Leitungen
```

```
                                SIO.ASYNC       14.01.80..BA        PAGE   1
LOC   OBJ CODE M STMT SOURCE STATEMENT                               ASM 5.9

                        1
                        2           ;  Asynchrone serielle SIO Uebertragung
                        3           ;  ====================================
                        4           ;  Polling ohne Quitungsbetrieb
                        5
                        6
```

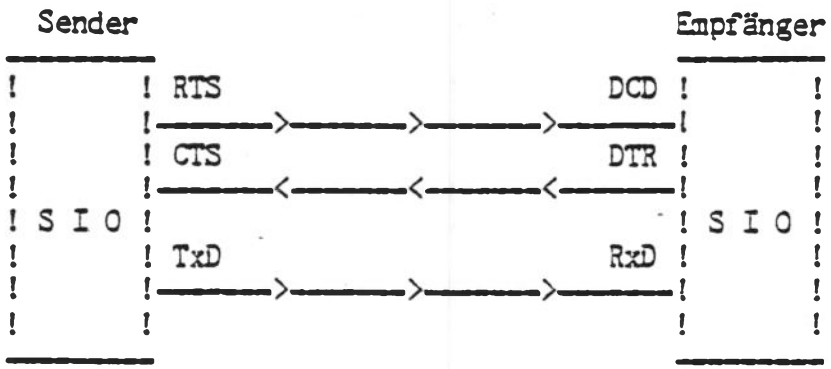
```
                        7  ; Initialisierung
                        8  ; ---------------
                        9
                       10  INISIO:
0000   210A00   R       11        LD HL,SIOTAB     ; HL mit Adresse von SIOTAB laden
0003   0609            12        LD B,TABLEN      ; B mit Laenge
0005   0E07            13        LD C,SIOCHB+2    ; C mit Kontrolladresse
                       14
0007   EDB3            15        OTIR             ; Ausgabe der Steuerworte an den
                       16                         ; SIO-Kanal B
0009   C9              17        RET
                       18
                       19  SIOTAB:
000A   18              20        DEFB    18H      ; -> Reg.0 -> RESET
000B   04              21        DEFB    04H      ; -> Reg.0 -> naechstes Reg.= Reg.4
000C   CC              22        DEFB    0CCH     ; -> Reg.4 -> 2 Stopbits, no Parity
                       23                         ;                Clock x 64, 7 Bit/Char.
000D   05              24        DEFB    05H      ; -> Reg.0 -> naechstes Reg.= Reg.5
000E   68              25        DEFB    68H      ; -> Reg.5 -> 8 Bit/Char., Sender frei
000F   03              26        DEFB    03H      ; -> Reg.0 -> naechstes Reg.= Reg.3
0010   C1              27        DEFB    0C1H     ; -> Reg.3 -> 8 Bit/Char., Empf. frei
0011   01              28        DEFB    01H      ; -> Reg.0 -> naechstes Reg.= Reg.1
0012   00              29        DEFB    00H      ; -> Reg.1 -> kein Interr. u Handshake
                       30  TABLEN  EQU $-SIOTAB
                       31
                       32
                       33  ; Ausgabe eine Zeichens
                       34  ; ---------------------
                       35
                       36  SIOOUT:
0013   F5              37        PUSH AF          ; A und F retten
                       38
                       39  WAIT:
0014   DB07            40        IN A,(SIOCHB+2)  ; Statusregister 0 lesen
0016   CB57            41        BIT 2,A          ; Sendepuffer leer ?
0018   28FA            42        JR Z,WAIT        ; wenn nicht, erneute Abfrage
                       43
001A   F1              44        POP AF           ; A u F zurueckholen
001B   D305            45        OUT (SIOCHB),A   ; und A aussenden
001D   C9              46        RET
                       47
                       48  ; Empfang eines Zeichens
                       49  ; ----------------------
                       50
                       51  SIOIN:
001E   DB07            52        IN A,(SIOCHB+2)  ; Statusregister 0 lesen
0020   CB47            53        BIT 0,A          ; ist ein Zeichen angekommen ?
0022   28FA            54        JR Z,SIOIN       ; wenn nicht, warten
0024   DB05            55        IN A,(SIOCHB)    ; wenn ja, abholen
0026   CBBF            56        RES 7,A          ; oberstes Bit loeschen
0028   C9              57        RET
                       58  SIOCHB EQU 05H         ; SIO Adresse Kanal B ECB/C8
```

.COMMENT ?

asynchrone, interruptgesteuerte SIO-Übertragung

Bei dem vorliegenden Programmbeispiel handelt es sich um eine serielle Übertragung, bei der nicht nur der reine Datentransfer, sondern auch das Herstellen bzw. Lösen der Verbindung interruptgesteuert verläuft.

Folgende Schaltung liegt zugrunde:



Die Verbindung TxD (Transmit Data) RxD (Receive Data) stellt die Datenleitung dar, auf der die Übertragung abläuft.
Das Handshake-Signal RTS (Request To Send) dient dazu, den Wunsch Daten zu senden zu signalisieren. Dazu wird RTS mit DCD (Data Carrier Detect) verbunden.
Seine Bereitschaft zum Empfang gibt der Empfänger durch Setzen des DTR-Signals (Data Terminal Ready) zu verstehen, welches mit CTS (Clear To Send) verbunden ist.

PAGE

Der Ablauf im vorliegenden Beispiel kann wie folgt beschrieben werden:

1) Setzen des RTS-Signals durch das Hauptprogramm

2) --> Interrupt beim Empfänger wegen Änderung der DCD-Leitung:
die DTR-Leitung weird gesetzt (Interrupt Service Routine 1).

3) --> Interrupt beim Sender wegen Änderung der DTS-Leitung:
erstes Zeichen wird dem Sender übergeben (ISR 2).

4) --> Interrupt beim Empfänger wegen Ankunft eines Zeichens und Interrupt beim Sender wegen Leerwerden des Sendepuffers:
empfangenes Zeichen wird abgeholt und gespeichert (ISR 3), neues Zeichen wird in den Sendepuffer gebracht (ISR 4).

Schritt 4) wird nun so lange wiederholt, bis alle Zeichen der Sendung übertragen worden sind. Anschließend wird das Signal RTS zurückgesetzt (innerhalb ISR 2!).

5) --> Interrupt beim Empfänger wegen Änderung der DCD-Leitung:
Rücksetzen des DTR-Signals (ISR 1).

6) --> Interrupt beim Sender wegen Änderung der CTS-Leitung:
Ausgabe der Endemeldung (ISR 2).

PAGE

Insgesamt werden bei fehlerfreiem Ablauf vier verschiedene Interrupts verwendet, die auf folgende Ursachen zurückzuführen sind:

- Sendepuffer leer
- Zeichen im Empfangspuffer verfügbar
- Zustandsänderung der DCD-Leitung (Empfänger)
- Zustandsänderung der CTS-Leitung (Sender)

Dies gilt für eine undirektionale Verbindung, wie sie in diesem Beispiel verwendet wurde. Bei einem bidirektionalen Betrieb erhöht sich die Zahl der insgesamt möglichen Interrupts entsprechend.

Das vorliegende Verhalten wird durch die Fähigkeit des Z80-SIO ermöglicht, auf verschiedene Situationen mit verschieden modifizierten Interruptvektoren zu reagieren. Bei der Initialisierung ist dafür das Bit "status affects vektor" zu setzen.

Pro Kanal sind folgende vier interruptauslösende Situationen unterscheidbar:

a) - Sendepuffer leer

b) - Änderung auf den Handshake-Leitungen (CTS oder DCD)

c) - Zeichen im Empfänger verfügbar

d) - Sondersituation beim Empfang (Fehler)

Im vorliegenden Beispiel wurden für den Sender die Situationen a) und b) zugelassen, für den Empfänger die Situationen c) und b), wobei bezüglich b) jeweils nur die Änderung von CTS bzw. DCD zum Tragen kommt.

Als Voprbereitung für den bidirektionalen Betrieb (full duplex) wurden die beiden Interrupt Service Routinen 1 und 2 so ausgelegt, daß sowohl eine Änderung des CTS-Signals als auch des DTR-Signals erkannt wird.
Bei Auftragen einer noch nicht implementierten Situation wird jedoch lediglich eine entsprechende Meldung ausgegeben.

?
PAGE

```
; externe Referenzen
; ------------------

        EXTERNAL inttab         ; Interrupttabelle


; globale Definitionen
; --------------------

        GLOBAL te_int,exintb,rf_int,exinta,notimp

PAGE
```

```
                ; Vereinbarungen
                ; ==============

                ; Flag-Bits
                ; ----------

0000            noresp  EQU 0           ; "keine Antwort Bit"
0003            dcd     EQU 3           ; DCD Status Bit (siehe SIO Manual)
0005            cts     EQU 5           ; CTS Status Bit (siehe SIO Manual)

                ; Kommandoworte
                ; --------------

                                        ; Kommandoworte
006A            rtson   EQU 06AH        ; zum Setzen von RTS
0068            rtsoff  EQU 068H        ; zum Rücksetzen von RTS
00E8            dtron   EQU 0E8H        ; zum Setzen von DTR
0068            dtroff  EQU 068H        ; zum Rücksetzen von DTR
0028            respin  EQU 028H        ; für die Aufhebung eines
                                        ; anstehenden Interrupts
0030            reserr  EQU 030H        ; für das Löschen der
                                        ; Fehleranzeige
0047            ctcmod  EQU 047H        ; zum Einstellen des CTC
0008            ctccon  EQU 008H        ; incl. Zeitkonstante

                ; Ein/Ausgabeadressen
                ; -------------------

0005            sioc8b  EQU 05H         ; SIO C8, Kanal b, Datenadr.
                                        ; (Monitorausgabe)
0080            siox1a  EQU 80H         ; SIO X, Kanal a, Datenadr.
                                        ; (Empfänger)
0081            siox1b  EQU 81H         ; SIO X, Kanal b, Datenadr.
                                        ; (Sender)
0088            ctcx10  EQU 88H         ; CTC1 X, Kanal 0
0089            ctcx11  EQU 89H         ; CTC1 X, Kanal 1
                                        ; (Erzeugung des Taktes
                                        ; für die Übertragung)

                PAGE
```

```
                        ; Übertragungsparameter
                        ; ----------------------

9000                    s_data  EQU 9000H       ; Startadr. des Quellblocks
9080                    r_data  EQU 9080H       ; Startadr. des Zielblocks
0080                    length  EQU 0080H       ; Länge des Blocks


                        ; Einsprungtabelle
                        ; -----------------

0000'  C3 0275'                 JP ini          ; Initialisierung

0003'  C3 0300'                 JP main         ; Hauptprogramm

                        PAGE
```

```
                                ; Meldungen
                                ; ---------

0006'  0C0A                     hmsg01: DEFW 0C0AH
0008'  48 50 3A 20                      DEFM 'HP: asynchrone, interruptgesteuerte
000C'  61 73 79 6E
0010'  63 68 72 6F                              SIO-Übertragung
0014'  6E 65 2C 20
0018'  69 6E 74 65
001C'  72 72 75 70
0020'  74 67 65 73
0024'  74 65 75 65
0028'  72 74 65 20
002C'  53 49 4F 2D
0030'  5D 62 65 72
0034'  74 72 61 67
0038'  75 6E 67
003B'  0A0D                             DEFW 0A0DH
003D'  00                               DEFB 00
003E'  0A                       hmsg02: DEFB 0AH
003F'  48 50 3A 20                      DEFM 'HP: Start durch Aktivieren von RTS'
0043'  53 74 61 72
0047'  74 20 64 75
004B'  72 63 68 20
004F'  41 6B 74 69
0053'  76 69 65 72
0057'  65 6E 20 76
005B'  6F 6E 20 52
005F'  54 53
0061'  0A0D                             DEFW 0A0DH
0063'  00                               DEFB 00
0064'  0A                       hmsg03: DEFB 0AH
0065'  48 50 3A 20                      DEFM 'HP: Empfänger gibt keine Antwort'
0069'  45 6D 70 66
006D'  7B 6E 67 65
0071'  72 20 67 69
0075'  62 74 20 6B
0079'  65 69 6E 65
007D'  20 41 6E 74
0081'  77 6F 72 74
0085'  0A0D                             DEFW 0A0DH
0087'  00                               DEFB 00
                                PAGE
```

```
0088'   0A                          smsg04: DEFB 0AH
0089'   53 2D 49 53                        DEFM 'S-ISR2: Empfänger reagiert;
008D'   52 32 3A 20
0091'   45 6D 70 66                                      CTS wurde aktiviert'
0095'   7B 6E 67 65
0099'   72 20 72 65
009D'   61 67 69 65
00A1'   72 74 3B 20
00A5'   43 54 53 20
00A9'   77 75 72 64
00AD'   65 20 61 6B
00B1'   74 69 76 69
00B5'   65 72 74
00B8'   0A0D                                DEFW 0A0DH
00BA'   00                                  DEFB 00
00BB'   0A                          smsg05: DEFB 0AH
00BC'   53 2D 49 53                        DEFM 'S-ISR4: Sendung abgesclossen;
00C0'   52 34 3A 20
00C4'   53 65 6E 64                                      RTS wurde deaktiviert'
00C8'   75 6E 67 20
00CC'   61 62 67 65
00D0'   73 63 6C 6F
00D4'   73 73 65 6E
00D8'   3B 20 52 54
00DC'   53 20 77 75
00E0'   72 64 65 20
00E4'   64 65 61 6B
00E8'   74 69 76 69
00EC'   65 72 74
00EF'   0A0D                                DEFW 0A0DH
00F1'   00                                  DEFB 00
00F2'   0A                          smsg06: DEFB 0AH
00F3'   53 2D 49 53                        DEFM 'S-ISR2: Emfänger reagiert;
00F7'   52 32 3A 20
00FB'   45 6D 66 7B                                      CTS wurde deaktiviert'
00FF'   6E 67 65 72
0103'   20 72 65 61
0107'   67 69 65 72
010B'   74 3B 20 43
010F'   54 53 20 77
0113'   75 72 64 65
0117'   20 64 65 61
011B'   6B 74 69 76
011F'   69 65 72 74
0123'   0A0D                                DEFW 0A0DH
0125'   0A                                  DEFB 0AH
0126'   45 6E 64 65                        DEFM 'Ende der Übertragung'
012A'   20 64 65 72
012E'   20 5D 62 65
0132'   72 74 72 61
0136'   67 75 6E 67
013A'   0A0D                                DEFW 0A0DH
013C'   00                                  DEFB 00
```

```
013D'   0A                          emsg07: DEFB 0AH
013E'   45 2D 49 53                         DEFM 'E-ISR1: Aufforderung erkannt;
0142'   52 31 3A 20                                       DTR wurde aktiviert'
0146'   41 75 66 66
014A'   6F 72 64 65
014E'   72 75 6E 67
0152'   20 65 72 6B
0156'   61 6E 6E 74
015A'   3B 20 44 54
015E'   52 20 77 75
0162'   72 64 65 20
0166'   61 6B 74 69
016A'   76 69 65 72
016E'   74
016F'   0A0D                                DEFW 0A0DH
0171'   00                                  DEFB 00
0172'   0A                          emsg08: DEFB 0AH
0173'   45 2D 49 53                         DEFM 'E-ISR1: Endemeldung erkannt;
0177'   52 31 3A 20                                       DTR wurde deaktiviert'
017B'   45 6E 64 65
017F'   6D 65 6C 64
0183'   75 6E 67 20
0187'   65 72 6B 61
018B'   6E 6E 74 3B
018F'   20 44 54 52
0193'   20 77 75 72
0197'   64 65 20 64
019B'   65 61 6B 74
019F'   69 76 69 65
01A3'   72 74
01A5'   0A0D                                DEFW 0A0DH
01A7'   00                                  DEFB 00
01A8'   0A                          emsg09: DEFB 0AH
01A9'   45 2D 49 53                         DEFM 'E-ISR1: Fehler: keine Statusänderung'
01AD'   52 31 3A 20
01B1'   46 65 68 6C
01B5'   65 72 3A 20
01B9'   6B 65 69 6E
01BD'   65 20 53 74
01C1'   61 74 75 73
01C5'   7B 6E 64 65
01C9'   72 75 6E 67
01CD'   0A0D                                DEFW 0A0DH
01CF'   00                                  DEFB 00
                                    PAGE
```

```
01D0'   0A                      smsg10: DEFB 0AH
01D1'   53 2D 49 53                     DEFM 'S-ISR2: Fehler: keine Statusänderung'
01D5'   52 32 3A 20
01D9'   46 65 68 6C
01DD'   65 72 3A 20
01E1'   6B 65 69 6E
01E5'   65 20 53 74
01E9'   61 74 75 73
01ED'   7B 6E 64 65
01F1'   72 75 6E 67
01F5'   0A0D                            DEFW 0A0DH
01F7'   00                              DEFB 00
01F8'   0A                      emsg11: DEFB 0AH
01F9'   45 2D 49 53                     DEFM 'E-ISR1: Fehler: falsche Statusänderung'
01FD'   52 31 3A 20
0201'   46 65 68 6C
0205'   65 72 3A 20
0209'   66 61 6C 73
020D'   63 68 65 20
0211'   53 74 61 74
0215'   75 73 7B 6E
0219'   64 65 72 75
021D'   6E 67
021F'   0A0D                            DEFW 0A0DH
0221'   00                              DEFB 00
0222'   0A                      smsg12: DEFB 0AH
0223'   53 2D 49 53                     DEFM 'S-ISR2: Fehler: falsche Statusänderung'
0227'   52 32 3A 20
022B'   46 65 68 6C
022F'   65 72 3A 20
0233'   66 61 6C 73
0237'   63 68 65 20
023B'   53 74 61 74
023F'   75 73 7B 6E
0243'   64 65 72 75
0247'   6E 67
0249'   0A0D                            DEFW 0A0DH
024B'   00                              DEFB 00
024C'   0A                      msg13:  DEFB 0AH
024D'   49 53 52 35                     DEFM 'ISR5: Fehler: ISR nicht implementiert'
0251'   3A 20 46 65
0255'   68 6C 65 72
0259'   3A 20 49 53
025D'   52 20 6E 69
0261'   63 68 74 20
0265'   69 6D 70 6C
0269'   65 6D 65 6E
026D'   74 69 65 72
0271'   74
0272'   0A0D                            DEFW 0A0DH
0274'   00                              DEFB 00
                        PAGE
```

```
                        ; Programm
                        ; ========

                        ; Initialisierung
                        ; ---------------

0275'                   ini:
0275'                   inicpu:
0275'  31 0539'                 LD SP,stack
0278'  21 0000*                 LD HL,inttab
027B'  7C                       LD A,H              ; Int.vektor oberes Byte
027C'  ED 47                    LD I,A              ; ins I-Register

027E'                   inictc:
027E'  3E 47                    LD A,ctcmod         ; Herstellung des Taktes
0280'  D3 88                    OUT (ctcx10),A      ; für die Übertragung
0282'  D3 89                    OUT (ctcx11),A      ; siehe ECB/X Beschreibung
0284'  3E 08                    LD A,ctccon
0286'  D3 88                    OUT (ctcx10),A
0288'  D3 89                    OUT (ctcx11),A

028A'                   inisio:                     ; Sendekanal
028A'  21 0000*                 LD HL,inttab        ; ----------
028D'  7D                       LD A,L              ; Int.vektor unteres Byte
028E'  32 02E5'                 LD (vektor),A       ; in die SIO-Tabelle

0291'  21 02E3'                 LD HL,bsiotab       ; Adr. der Kommandosequenz
0294'  06 0B                    LD B ,btablen       ; Länge der Sequenz
0296'  0E 83                    LD C ,siox1b+2      ; Kontrolladr. SIO
0298'  ED B3                    OTIR                ; Ausgabe der Sequenz

                                                    ; Empfangskanal
029A'  21 02EE'                 LD HL,asiotab       ; ----------
029D'  06 09                    LD B ,atablen
029F'  0E 82                    LD C ,siox1a+2
02A1'  ED B3                    OTIR

                                                    ; Monitorkanal
02A3'  21 02F7'                 LD HL,c8tab         ; ----------
02A6'  06 09                    LD B ,c8len
02A8'  0E 07                    LD C ,sioc8b+2
02AA'  ED B3                    OTIR

                        PAGE
```

```
02AC'                           inialg:
02AC'   FD 21 0549'                     LD IY,flagb
02B0'   FD CB 00 9E                     RES dcd,(IY)            ; Grundzustand für die
02B4'   FD CB 00 AE                     RES cts,(IY)            ; erzeugen
02B8'   FD 21 054A'                     LD IY,flaga
02BC'   FD CB 00 9E                     RES dcd,(IY)
02C0'   FD CB 00 AE                     RES cts,(IY)
02C4'   21 9000                         LD HL,s_data
02C7'   22 054B'                        LD (source),HL
02CA'   21 9080                         LD HL,r_data
02CD'   22 054D'                        LD (target),HL
02D0'   21 0080                         LD HL,length
02D3'   22 054F'                        LD (range),HL

02D6'   DD 21 0006'                     LD IX,hmsg01            ; Meldung
 _DA'   CD 0333'                        CALL putmsg             ; ausgeben

02DD'   ED 5E                           IM 2                    ; Int.mode 2
02DF'   FB                              EI

02E0'   C3 0300'                        JP main                 ; zum Hauptprogramm

                                PAGE
```

```
                        ; Kommandosequenzen
                        ; ------------------
                                                ; für Sendekanal
02E3'                   bsiotab:                ; ---------------
02E3'  18                       DEFB 18H        ; -> Reg0 : Reset
02E4'  12                       DEFB 12H        ; -> Reg0 : Zeiger auf Reg2
                                                ;           Reset Ext.Stat.Int.
02E5'                   vektor: DEFS  1         ; Platz für unteres Int.Vek.Byte
02E6'  14                       DEFB 14H        ; -> Reg0 : Zeiger auf Reg4
                                                ;           Reset Ext.Stat.Int.
02E7'  CC                       DEFB 0CCH       ; -> Reg4 : 2 Stopb. oHne Parity
                                                ;           Clock * 64
02E8'  15                       DEFB 15H        ; -> Reg0 : Zeiger auf Reg5
                                                ;           Reset Ext.Stat.Int.
02E9'  68                       DEFB 68H        ; -> Reg5 : 8 Bit/Z. Sender frei
02EA'  13                       DEFB 13H        ; -> Reg0 : Zeiger auf Reg3
                                                ;           Reset Ext.Stat.Int.
02EB'  C1                       DEFB 0C1H       ; -> Reg3 : 8 Bit/Z. Empf. frei
02EC'  11                       DEFB 11H        ; -> Reg0 : Zeiger auf Reg1
                                                ;           Reset Ext.Stat.Int.
02ED'  1F                       DEFB 1FH        ; -> Reg1 : Int. frei Handshake
                                                ;           Status affects Vektor
000B                    btablen EQU $-bsiotab

                        PAGE
```

```
                                        ; für Empfangskanal
02EE'                   asiotab:        ; ----------------
02EE'  18                       DEFB 18H        ; asiotab ist identisch zu
02EF'  14                       DEFB 14H        ; bsiotab. Ledigilch der
02F0'  CC                       DEFB 0CCH       ; Int.vektor fehlt, da dieser
02F1'  15                       DEFB 15H        ; nur bei Kanal b eingegeben
02F2'  68                       DEFB 68H        ; werden kann.
02F3'  13                       DEFB 13H
02F4'  C1                       DEFB 0C1H
02F5'  11                       DEFB 11H
02F6'  1F                       DEFB 1FH

0009                    atablen EQU $-asiotab

                                        ; für den Monitorkanal
 2F7'                   c8tab:          ; ----------------
 2F7'  18                       DEFB 18H
)2F8'  04                       DEFB 04H
)2F9'  4C                       DEFB 4CH
)2FA'  05                       DEFB 05H
)2FB'  68                       DEFB 68H
)2FC'  03                       DEFB 03H
)2FD'  C1                       DEFB 0C1H
)2FE'  01                       DEFB 01H
02FF'  00                       DEFB 00H

0009                    c8len   EQU $-c8tab
                        PAGE
```

```
                        ; Hauptprogramm
                        ; ----------

0300'                   main:
0300'  FD 21 054A'              LD IY,flaga             ; Vorbereitung für
0304'  FD CB 00 C6              SET noresp,(IY)         ; timeout

0308'  DD 21 003E'              LD IX,hmsg02
030C'  CD 0333'                 CALL putmsg             ; Startmeldung ausgeben

030F'  3E 15                    LD A,15H
0311'  D3 83                    OUT (siox1b+2),A        ; RTS Signal
0313'  3E 6A                    LD A,rtson              ; des Senders
0315'  D3 83                    OUT (siox1b+2),A        ; setzen

0317'  06 00                    LD B,0
0319'  10 FE                    DJNZ $                  ; warten auf Reaktion

031B'  FD CB 00 46              BIT noresp,(IY)         ; hat Empf. geantwortet ?
031F'  28 0F                    JR Z,traend             ; wenn ja

0321'  DD 21 0064'              LD IX,hmsg03            ; wenn nicht,
0325'  CD 0333'                 CALL putmsg             ; Meldung machen
0328'  3E 15                    LD A,15H                ; und
032A'  D3 83                    OUT (siox1b+2),A        ; RTS Signal
032C'  3E 68                    LD A,rtsoff             ; des Senders
032E'  D3 83                    OUT (siox1b+2),A        ; zurücksetzen

0330'                   traend:
0330'                   stop:
0330'  76                       HALT
0331'  18 FD                    JR stop


                        PAGE
```

```
                                ; Unterprogramm
                                ; ---------------

0333'                           putmsg:
0333'   DD 7E 00                       LD A,(IX)
0336'   FE 00                          CP 0                    ; Meldung fertig ?
0338'   28 0E                          JR Z,putend             ; wenn ja

033A'   F5                              PUSH AF                 ; wenn nicht, rette A

033B'                           wait:
033B'   DB 07                          IN A,(sioc8b+2)         ; und prüfe
033D'   CB 57                          BIT 2,A                 ; ob Kanal frei ist
033F'   28 FA                          JR Z,wait               ; wenn nicht, warten

0341'   F1                            POP AF                  ; wenn ja
 342'   D3 05                         OUT (sioc8b),A          ; Ausgabe des Zeichens
 344'   DD 23                         INC IX                  ; Zeiger auf nächstes
                                                               ; Zeichen stellen
0346'   18 EB                        JR putmsg

0348'                           putend:
0348'   C9                              RET


                                PAGE
```

```
                        ; Interrupt Service Routinen
                        ; ==========================

                        ; ISR1: External Status Change Empfänger
                        ; ---------------------------------------

0349'                   exinta:
0349'  F5                       PUSH AF
034A'  C5                       PUSH BC
034B'  D5                       PUSH DE
034C'  E5                       PUSH HL
034D'  DD E5                    PUSH IX
034F'  FD E5                    PUSH IY

0351'  3E 10                    LD A,10H                ; muß sein,wegen
0353'  D3 82                    OUT (siox1a+2),A        ; Reset Ext.Stst.Int
  55'  DB 82                    IN A,(siox1a+2)         ; lesen Statusreg.0

0357'                   achkcts:
0357'  FD 21 054A'              LD IY,flaga
035B'  FD CB 00 6E              BIT cts,(IY)            ; prüfe alten CTS Zustand
035F'  20 15                    JR NZ,aocts1            ; wenn gesetzt

0361'                   aocts0:                         ; wenn nicht gesetzt,
0361'  CB 6F                    BIT cts,A               ; prüfe jetzigen Zustand
0363'  28 22                    JR Z,achkdcd            ; wenn gleich, prüfe DCD

0365'                   acts0_1:                        ; wenn ungleich,
0365'  FD CB 00 EE              SET cts,(IY)            ; speichere neues CTS
0369'  FD CB 00 86              RES noresp,(IY)         ; verhindere timeout
036D'  DD 21 01F8'              LD IX,emsg11            ; und gib
0371'  CD 0333'                 CALL putmsg             ; Warnung "nicht impl."
0374'  18 54                    JR achkend

0376'                   aocts1:                         ; CTS gesetzt
 376'  CB 6F                    BIT cts,A               ; prüfe jetziges CTS
 378'  20 0D                    JR NZ,achkdcd           ; wenn gleich, prüfe DCD

037A'                   acts1_0:
037A'  FD CB 00 AE              RES cts,(IY)            ; wenn nicht, speichere
037E'  DD 21 01F8'              LD IX,emsg11            ; neues CTS und gib
0382'  CD 0333'                 CALL putmsg             ; Warnung "nicht impl."
0385'  18 43                    JR achkend
                        PAGE
```

```
0387'                           achkdcd:                ; wenn CTS unverändert,
                                                        ; muß DCD verändert sein!
0387'  FD CB 00 5E                      BIT dcd,(IY)   ; prüfe altes DCD
038B'  20 1D                             JR NZ,aodcd1   ; wenn es gesetzt war

038D'                           aodcd0:                 ; wenn nicht gesetzt,
038D'  CB 5F                             BIT dcd,A      ; prüfe jetziges DCD
038F'  28 32                             JR Z,achkerr   ; wenn gleich, Fehler!

0391'                           adcd0_1:                ; wenn nicht,
0391'  FD CB 00 DE                       SET dcd,(IY)   ; speichere neues DCD,
0395'  FD CB 00 86                       RES noresp,(IY); verhindere timeout
0399'  DD 21 013D'                       LD IX,emsg07   ; und melde
039D'  CD 0333'                          CALL putmsg    ; "Aufforderung erkannt"
03A0'  3E 05                             LD A,05H
03A2'  D3 82                             OUT (siox1a+2),A
  A4'  3E E8                             LD A,dtron     ; aktiviere anschließend
 3A6'  D3 82                             OUT (siox1a+2),A ; die DTR-Leitung
03A8'  18 20                             JR achkend

03AA'                           aodcd1:                 ; altes DCD ist gesetzt
03AA'  CB 5F                             BIT dcd,A      ; prüfe jetziges DCD
03AC'  20 15                             JR NZ,achkerr  ; wenn gleich, Fehler!

03AE'                           adcd1_0:                ; wenn nicht,
03AE'  FD CB 00 9E                       RES dcd,(IY)   ; speichere neues DCD,
03B2'  DD 21 0172'                       LD IX,emsg08   ; und melde "Ende=
03B6'  CD 0333'                          CALL putmsg    ; meldung erkannt"
03B9'  3E 05                             LD  A,05H
03BB'  D3 82                             OUT (siox1a+2),A
03BD'  3E 68                             LD A,dtroff    ; deaktiviere anschl.
03BF'  D3 82                             OUT (siox1a+2),A ; DTR-Leitung
03C1'  18 07                             JR achkend

03C3'                           achkerr:                ; wenn keinerlei Änderung
 C3'  DD 21 01A8'                        LD IX,emsg09   ; Ausgabe "Fehler bei
 C7'  CD 0333'                           CALL putmsg    ; bei Statusänderung"

03CA'                           achkend:
03CA'  FD E1                             POP IY
03CC'  DD E1                             POP IX
03CE'  E1                                POP HL
03CF'  D1                                POP DE
03D0'  C1                                POP BC
03D1'  F1                                POP AF

03D2'  FB                                EI
03D3'  ED 4D                             RETI
                                PAGE
```

```
                        ; ISR2: External Status Change Sender
                        ; ------------------------------------

03D5'                   exintb:
03D5'  F5                       PUSH AF
03D6'  C5                       PUSH BC
03D7'  D5                       PUSH DE
03D8'  E5                       PUSH HL
03D9'  DD E5                    PUSH IX
03DB'  FD E5                    PUSH IY

03DD'  3E 10                    LD A,10H                ; muß sein, wegen
03DF'  D3 83                    OUT (siox1b+2),A        ; Reset Ext.Stat.Int.
03E1'  DB 83                    IN A,(siox1b+2)         ; lesen Statusreg.0

03E3'                   bchkcts:
03E3'  FD 21 0549'              LD IY,flagb
03E7'  FD CB 00 6E              BIT cts,(IY)            ; prüfe alten CTS Zustand
03EB'  20 1B                    JR NZ,bocts1            ; wenn gesetzt

03ED'                   bocts0:                         ; wenn nicht gesetzt,
03ED'  CB 6F                    BIT cts,A               ; prüfe jetzigen Zustand
03EF'  28 28                    JR Z,bchkdcd            ; wenn gleich, prüfe DCD

03F1'                   bcts0_1:                        ; wenn ungleich,
03F1'  FD CB 00 EE              SET cts,(IY)            ; speichere neues CTS
03F5'  FD CB 00 86              RES noresp,(IY)         ; verhindere timeout
03F9'  DD 21 0088'              LD IX,smsg04            ; und gib
03FD'  CD 0333'                 CALL putmsg             ; Meldung "CTS aktiv"

0400'  2A 054B'                 LD HL,(source)          ; anschließend
0403'  7E                       LD A,(HL)               ; erstes Zeichen an
0404'  D3 81                    OUT (siox1b),A          ; Sender ausgeben

0406'  18 40                    JR bchkend

0408'                   bocts1:
0408'  CB 6F                    BIT cts,A               ; prüfe jetziges CTS
040A'  20 0D                    JR NZ,bchkdcd           ; wenn gleich, prüfe DCD

040C'                   bcts1_0:
040C'  FD CB 00 AE              RES cts,(IY)            ; wenn nicht, speichere
0410'  DD 21 00F2'              LD IX,smsg06            ; neues CTS und gib
0414'  CD 0333'                 CALL putmsg             ; Meldung "CTS deaktiv"
0417'  18 2F                    JR bchkend
                        PAGE
```

```
0419'                           bchkdcd:                ; wenn CTS unverändert,
                                                        ; muß DCD verändert sein!
0419'  FD CB 00 5E                      BIT dcd,(IY)   ; prüfe altes DCD
041D'  20 11                            JR NZ,bodcd1   ; wenn es gesetzt war

041F'                           bodcd0:                 ; wenn nicht gesetzt,
041F'  CB 5F                            BIT dcd,A      ; prüfe jetziges DCD
0421'  28 1E                            JR Z,bchkerr   ; wenn gleich, Fehler!

0423'                           bdcd0_1:                ; wenn nicht,
0423'  FD CB 00 DE                      SET dcd,(IY)   ; speichere neues DCD
0427'  DD 21 0222'                      LD IX,smsg12   ; und melde
042B'  CD 0333'                         CALL putmsg    ; "nicht impl. ISR"
042E'  18 18                            JR bchkend

0430'                           bodcd1:                 ; altes DCD ist gesetzt
 430'  CB 5F                            BIT dcd,A      ; prüfe jetziges DCD
 432'  20 0D                            JR NZ,bchkerr  ; wenn gleich, Fehler!

0434'                           bdcd1_0:                ; wenn nicht,
0434'  FD CB 00 9E                      RES dcd,(IY)   ; speichere neues DCD,
0438'  DD 21 0222'                      LD IX,smsg12   ; und melde
043C'  CD 0333'                         CALL putmsg    ; "nicht impl. ISR"
043F'  18 07                            JR bchkend

0441'                           bchkerr:                ; wenn keinerlei Änderung,
0441'  DD 21 01D0'                      LD IX,smsg10   ; Ausgabe "Fehler bei
0445'  CD 0333'                         CALL putmsg    ; bei Statusänderung"

0448'                           bchkend:
0448'  FD E1                            POP IY
044A'  DD E1                            POP IX
044C'  E1                               POP HL
044D'  D1                               POP DE
 44E'  C1                               POP BC
 44F'  F1                               POP AF

0450'  FB                               EI
0451'  ED 4D                            RETI
                                PAGE
```

```
                        ; ISR3 Zeichen beim Empfänger verfügbar
                        ; ---------------------------------------

0453'                   rf_int:
0453'  F5                       PUSH AF
0454'  E5                       PUSH HL

0455'  DB 80                    IN A,(siox1a)          ; Zeichen holen
0457'  2A 054D'                 LD HL,(target)         ; Speicherzeiger abrufen
045A'  77                       LD (HL),A              ; Zeichen ablegen
045B'  23                       INC HL                 ; Zeiger weiterschalten
045C'  22 054D'                 LD (target),HL         ; und rückspeichern

045F'  E1                       POP HL
0460'  F1                       POP AF

 461'  FB                       EI
 462'  ED 4D                    RETI


                        ; ISR4 Sendepuffer leer
                        ; ---------------------

0464'                   te_int:
0464'  F5                       PUSH AF
0465'  C5                       PUSH BC
0466'  D5                       PUSH DE
0467'  E5                       PUSH HL
0468'  DD E5                    PUSH IX
046A'  FD E5                    PUSH IY

046C'  ED 5B 054F'              LD DE,(range)          ; Zeichenzähler abrufen
0470'  1B                       DEC DE                 ; decrementieren
0471'  ED 53 054F'              LD (range),DE          ; und rückspeichern
 475'  7B                       LD A,E
 476'  B2                       OR D                   ; alle Zeichen übertr.?
0477'  28 0C                    JR Z,resrts            ; wenn ja

0479'  2A 054B'                 LD HL,(source)         ; wenn nicht, hole
047C'  23                       INC HL                 ; Zeiger, incrementiere
047D'  22 054B'                 LD (source),HL         ; speichere zurück
0480'  7E                       LD A,(HL)              ; hole Zeichen
0481'  D3 81                    OUT (siox1b),A         ; und gebe es aus
0483'  18 13                    JR te_end
                        PAGE
```

```
0485'                          resrts:
0485'  3E 28                           LD A,respin
0487'  D3 83                           OUT (siox1b+2),A        ; storniere evtl. Int.
0489'  3E 05                           LD A,05H
048B'  D3 83                           OUT (siox1b+2),A
048D'  3E 68                           LD A,rtsoff
048F'  D3 83                           OUT (siox1b+2),A        ; setze RTS zurück

0491'  DD 21 00BB'                     LD IX,smsg05            ; und melde
0495'  CD 0333'                        CALL putmsg             ; "RTS deaktiviert"

0498'                          te_end:
0498'  FD E1                           POP IY
049A'  DD E1                           POP IX
049C'  E1                              POP HL
049D'  D1                              POP DE
049E'  C1                              POP BC
049F'  F1                              POP AF

04A0'  FB                              EI
04A1'  ED 4D                           RETI


                               ; ISR5 Nicht implementierte ISR
                               ; ---------------------------

04A3'                          notimp:
04A3'  F5                              PUSH AF
04A4'  DD E5                           PUSH IX

04A6'  3E 30                           LD A,reserr
04A8'  D3 83                           OUT (siox1b+2),A        ; storniere Fehler
04AA'  D3 82                           OUT (siox1a+2),A
04AC'  DD 21 024C'                     LD IX,msg13
04B0'  CD 0333'                        CALL putmsg             ; melde "nicht
                                                               ; impl. Funktion"
04B3'  DD E1                           POP IX
04B5'  F1                              POP AF

04B6'  FB                              EI
04B7'  ED 4D                           RETI
                               PAGE
```

```
                        ; Stackbereich und Datenbereich
                        ; ---------------------------------

04B9'                           DEFS 80H
0539'                   stack:
0539'                           DEFS 10H


0549'                   flagb:  DEFS  1H
054A'                   flaga:  DEFS  1H

054B'                   source: DEFS  2H
054D'                   target: DEFS  2H
054F'                   range:  DEFS  2H


                                END
```

Macros:

Symbole:

0357'	ACHKCTS	0387'	ACHKDCD	03CA'	ACHKEND
03C3'	ACHKERR	0365'	ACTS0_1	037A'	ACTS1_0
0391'	ADCD0_1	03AE'	ADCD1_0	0361'	AOCTS0
0376'	AOCTS1	038D'	AODCD0	03AA'	AODCD1
02EE'	ASIOTAB	0009	ATABLEN	03E3'	BCHKCTS
0419'	BCHKDCD	0448'	BCHKEND	0441'	BCHKERR
03F1'	BCTS0_1	040C'	BCTS1_0	0423'	BDCD0_1
0434'	BDCD1_0	03ED'	BOCTS0	0408'	BOCTS1
041F'	BODCD0	0430'	BODCD1	02E3'	BSIOTAB
000B	BTABLEN	0009	C8LEN	02F7'	C8TAB
0008	CTCCON	0047	CTCMOD	0088	CTCX10
0089	CTCX11	0005	CTS	0003	DCD
0068	DTROFF	00E8	DTRON	013D'	EMSG07
0172'	EMSG08	01A8'	EMSG09	01F8'	EMSG11
0349I'	EXINTA	03D5I'	EXINTB	054A'	FLAGA
0549'	FLAGB	0006'	HMSG01	003E'	HMSG02
0064'	HMSG03	0275'	INI	02AC'	INIALG
0275'	INICPU	027E'	INICTC	028A'	INISIO
028B*	INTTAB	0080	LENGTH	0300'	MAIN
024C'	MSG13	0000	NORESP	04A3I'	NOTIMP
0348'	PUTEND	0333'	PUTMSG	054F'	RANGE
0030	RESERR	0028	RESPIN	0485'	RESRTS
0453I'	RF_INT	0068	RTSOFF	006A	RTSON
9080	R_DATA	0005	SIOC8B	0080	SIOX1A
0081	SIOX1B	0088'	SMSG04	00BB'	SMSG05
00F2'	SMSG06	01D0'	SMSG10	0222'	SMSG12
054B'	SOURCE	0539'	STACK	0330'	STOP
9000	S_DATA	054D'	TARGET	0498'	TE_END
0464I'	TE_INT	0330'	TRAEND	02E5'	VEKTOR
033B'	WAIT				

Keine Fehler

```
                        TITLE SIO
                        SUBTTL 17.03.82..BA

                ; Interrupttabelle des Programms SIO
                ; ===================================

                ; externe Referenzen
                ; ------------------

                        EXTERNAL te_int,exintb,rf_int,exinta,notimp

                ; globale Definitionen
                ; --------------------

                        GLOBAL inttab


0000'                   inttab:
0000'                   intabb:                 ; Tabelle für Kanal b (Sender)
0000'   0000*                   DEFW te_int     ; Sendepuffer leer
0002'   0000*                   DEFW exintb     ; Statusänderung
0004'   0000*                   DEFW notimp     ; nicht implementiert
0006'   0000*                   DEFW notimp     ; nicht implementiert

0008'                   intaba:                 ; Tabelle für Kanal a (Empänger)
0008'   0000*                   DEFW notimp     ; nicht implementiert
000A'   0000*                   DEFW exinta     ; Statusänderung
000C'   0000*                   DEFW rf_int     ; Zeichen im Empfänger verfügbar
000E'   0000*                   DEFW notimp     ; nicht implementiert


                        END
```

Macros:

Symbole:

000A*	EXINTA	0002*	EXINTB	0008'	INTABA
0000'	INTABB	0000I'	INTTAB	000E*	NOTIMP
000C*	RF_INT	0000*	TE_INT		

Keine Fehler