Indholdsfortegnelse

Programmeringsøvelse: Port-test.

Formål: Formålet med øvelsen er at indlære, hvorledes man kan lave en " go / no go " test af en almindelig port, samt at indøve brugen af udviklingsudstyret.

Udstyr: Udviklingsudstyr PMDS.
EPROM 2732 (slettet).
Comet 3400 med MPS-10 kort (32 bit I/O).
Dummy-kabel til at forbinde ind og udgange sammen.
Stik med lysdioder til indikering.

Hjælpemidler: Diagrammer til Comet 3400.
Datablad over Z80 PIO.
Oversigt over nødvendige portnumre og controlord til PIO.

Gennemførelse: Programmet skal på MPS-10 kortet teste port A i IC-3 og port A i IC-5.

1) Tegn et struktugram der udfører følgende:
Tænd lysdiode 1 (indikation af test startet).
Sæt IC-3 port A op som udgang og IC-5 port A som indgang,hvorefter data sendes til udgangen og hentes på indgangen for at sammenligne. Hvis dataerne er forskellige skal programmet stoppe.
Der skal prøves med alle kombinationer af data.
Nu vendes dataretningen ved at IC-3 port A sættes op som indgang og IC-5 port A som udgang, hvorefter der på samme måde prøves med alle datakombinationer.
Hvis det er i orden tændes også lysdiode 2 (indikation af test ok), hvorefter programmet stoppes.

2) Programmet skrives i Z80 assembler ud fra struktugrammet:
Ved hjælp af editoren åbnes filen: porttest.s
og programmet indtastes i denne fil.
Efter assemblering kan det eventuelt afprøves i udviklingsudstyret v.h.a. debuggeren, hvorefter det programmeres i en EPROM (type 2732).

3) Klargør Comet 3400:

MPS-10 kort: forbind dummykabel mellem PIO'ernes A-porte.

MPS-23A kort: forbind stikket med lysdioderne

MPS-27 kort: fjern EPROM'en (IC 12) og i test-EPROM'en i stedet for.

4) Tænd for Comet 3400 og afprøv testprogrammet imens indikationen på lysdioderne iagttages. De afprøves både med og uden fejl på I/O kortet.

Oversigt over nødvendige portnumre:

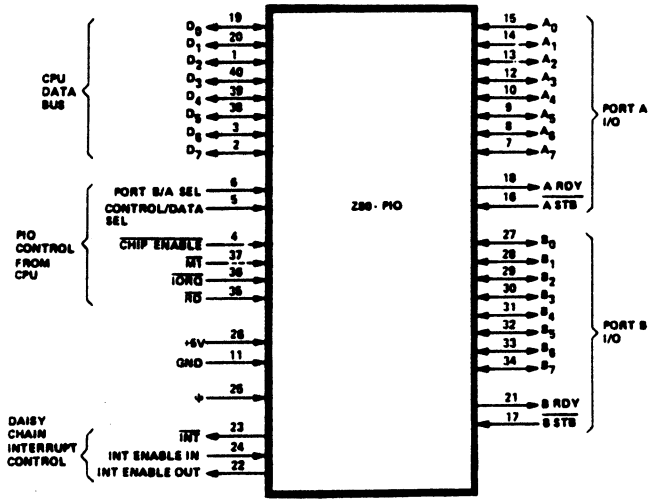
Kreds	Register	Symbolsk navn	Port nr.
PIO:			
IC-3 (MPS-10)	data-A	DATA3A	58 hex
IC-3 (MPS-10)	control-A	CONT3A	5A hex
IC-5 (MPS-10)	data-A	DATA5A	5C hex
IC-5 (MPS-10)	control-A	CONT5A	5E hex
Lysdioder:			
IC-24 (MPS-23A)	output	LED	F0 hex

Control-ord til PIO

Funktion	Symbolsk navn	Control ord
Byte output	MODE0	0F hex
Byte input	MODE1	4F hex

Zilog

Z-80 PIO Pin Description

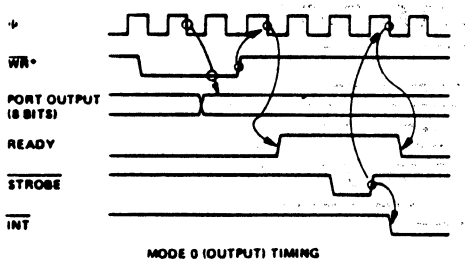


D_7-D_0	Z80-CPU Data Bus (bidirectional, tristate)
B/A Sel	Port B or A Select (input, active high)
C/D Sel	Control or Data Select (input, active high)
$\overline{CE}$	Chip Enable (input, active low)
Φ	System Clock (input)
$\overline{M1}$	Machine Cycle One Signal from CPU (input, active low)
$\overline{IORQ}$	Input/Output Request from Z80-CPU (input, active low)
$\overline{RD}$	Read Cycle Status from the Z80-CPU (input, active low)
IEI	Interrupt Enable In (input, active high)
IEO	Interrupt Enable Out (output, active high). IEI and IEO form a daisy chain connection for priority interrupt control.
$\overline{INT}$	Interrupt Request (output, open drain, active low)
A_0-A_7	Port A Bus (bidirectional, tristate)
$\overline{A\ STB}$	Port A Strobe Pulse from Peripheral Device (input, active low)
A RDY	Register A Ready (output, active high)
B_0-B_7	Port B Bus (bidirectional, tristate)
$\overline{B\ STB}$	Port B Strobe Pulse from Peripheral Device (input, active low)
B RDY	Register B Ready (output, active high)

Timing Waveforms

OUTPUT MODE

An output cycle is always started by the execution of an output instruction by the CPU. The $\overline{WR}$ pulse from the CPU latches the data from the CPU data bus into the selected port's output register. The write pulse sets the ready flag after a low going edge of Φ, indicating data is available. Ready stays active until the positive edge of the strobe line is received indicating that data was taken by the peripheral. The positive edge of the strobe pulse generates an $\overline{INT}$ if the interrupt enable flip flop has been set and if this device has the highest priority.

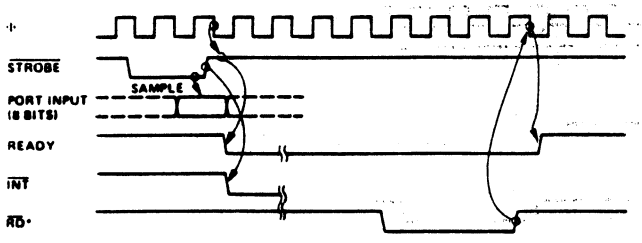


MODE 0 (OUTPUT) TIMING

$\overline{WR}^* = \overline{RD} \cdot CE \cdot \overline{C/D} \cdot IORQ$

INPUT MODE

When $\overline{STROBE}$ goes low data is loaded into the selected port input register. The next rising edge of strobe activates $\overline{INT}$ if interrupt enable is set and this is the highest priority requesting device. The following falling edge of Φ resets Ready to an inactive state, indicating that the input register is full and cannot accept any more data until the CPU completes a read. When a read is complete the positive edge of $\overline{RD}$ will set Ready at the next low going transition of Φ. At this time new data can be loaded into the PIO.

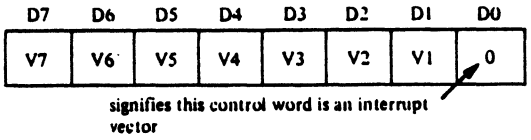


$\overline{RD}^* = \overline{RD} \cdot CE \cdot \overline{C/D} \cdot IORQ$ MODE 1 (INPUT) TIMING

PIO Programming

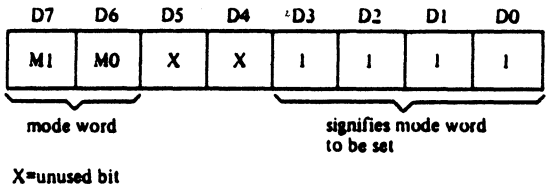
LOAD INTERRUPT VECTOR

The Z80-CPU requires an 8-bit interrupt vector be supplied by the interrupting device. The CPU forms the address for the interrupt service routine of the port using this vector. During an interrupt acknowledge cycle the vector is placed on the Z-80 data bus by the highest priority device requesting service at that time. 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.



SELECTING AN OPERATING MODE

When selecting an operating mode, the 2-bit mode control register is set to one of four values. These two bits are the most significant bits of the register, bits 7 and 6; bits 5 and 4 are not used while bits 3 through 0 are all set to 1111 to indicate "set mode."



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

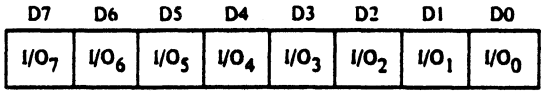
MODE 0 active indicates that data is to be written from the CPU to the peripheral.

MODE 1 active indicates that data is to be read from the peripheral to the CPU.

MODE 2 allows data to be written to or read from the peripheral device.

MODE 3 is intended for status and control applications. When selected, the next control word must set the I/O Register to indicate which lines are to be input and which lines are to be output.

I/O = 1 sets bit to input.
I/O = 0 sets bit to output.



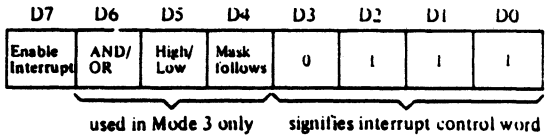
INTERRUPT CONTROL

Bit 7 = 1 — interrupt enable is set—allowing interrupt to be generated.

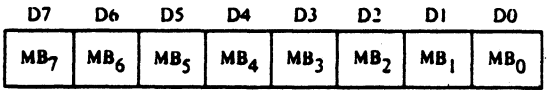
Bit 7 = 0 — indicates the enable flag is reset and interrupts may not be generated.

Bits 6,5,4 — are used in the bit mode interrupt operations; otherwise they are disregarded.

Bits 3,2,1,0 — signify that this command word is an interrupt control word.

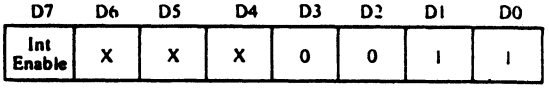


If the "mask follows" bit is high (D4 = 1), the next control word written to the port must be the mask.



Only those port lines whose mask bit is a 0 will be monitored for generating an interrupt.

The interrupt enable flip-flop of a port may be set or reset without modifying the rest of the interrupt control word by the following command.



Programmeringsøvelse: Signatur Ram-test.

Formål: Formålet med øvelsen er at indøve hvorledes en beskrevet test-algoritme kan omsættes til et program, skrevet i assembler, samt at indse hvilke ting der skal tages højde for, når programmet skal bruges til stimulering af en signaturmåling.

Udstyr:
Udviklingsudstyr PMDS.
EPROM 2732 (slettet).
Comet 3400.
Signaturmultimeter.
2 stk. 16-ben IC-clips.

Hjælpemidler: Diagrammer til Comet 3400.

Beskrivelse af opgaven:
Programmet skal på MPS-23A kortet teste ramlageret der adresseres fra adr.0000 til adr.F000. Ramlageret skal testes for: skrive/læse-fejl, samt decodnings- (adresse-) fejl.
Testens resultat skal kunne undersøges ved 8 signatur-målinger på databussen og sammenligning med kendte ok-signaturer for disse. Til brug ved fejlfinding skal der kunne måles entydige signaturer på adresse- og chipselect-kredsløbene omkring ramlageret.
Testprogrammet skal generere de nødvendige start/-stop signaler til signaturmeteret.
For at få entydige signaturer kræves det, at den del af testprogrammet, der ligger imellem start og stop, skal gennemkøres ens hver gang, uanset om ramlageret er iorden eller ej. Det betyder at der i denne del af programmet ikke må forekomme betingede jump´s, hvor betingelsen er afhængig af de læste data fra ramlageret.
Da programmet jo skal teste hele ramlageret, må der ikke benyttes rampladser til lagring af mellemresultater o.lign., man må klare sig med CPU´ens interne registre. Af samme årsag må man heller ikke benytte subroutiner.

Gennemførelse:

1) Tegn et struktugram der udfører følgende:

a) Generer en startpuls til signaturmeteret.
(Det kan gøres ved at der skrives et "1"tal til portnr. F1 hex, hvorved der på MPS-23A kortet

kommer en positiv flanke ud på stikket J1 ben 1. Der skal selvfølgelig vælges en udgang og dermed et port nr. der i testprogrammet ikke benyttes til andet, end til denne generering af et start/-stop signal).

b) Fyld hele ramlageret med dataerne skiftevis AA hex og 55 hex (checkerboard eller skakbrætsmønster d.v.s. skiftevis "1" og "0").

c) Læs hele ramlageret. (Dette gøres udelukkende for at man med signaturmålingerne på databussen kan teste dataerne fra ramlageret; d.v.s. dataerne skal altså ikke bruges til noget i CPU'en).

d) Nu fyldes hele ramlageret med det modsatte mønster, altså skiftevis AA hex og 55 hex.

e) Igen læses hele ramlageret, for at danne signaturer på databussen.

1. gennemløb		2. gennemløb	
RAM	adr.	RAM	adr.
55	EFFF	AA	EFFF
AA	EFFE	55	EFFE
55	EFFD	AA	EFFD
AA	EFFC	55	EFFC
55	0003	AA	0003
AA	0002	55	0002
55	0001	AA	0001
AA	0000	55	0000

(Det er nu testet om hver hukommelses-celle henholdsvis kan "1" og "0" stilles; men ikke om der eventuelt er decodningsfejl).

f) Ramlageret fyldes nu med adresse-afhængige data, d.v.s. adressen skrives ud på rampladsen som data. (For at få plads til de 16 bit som adressen består af skal man betragte rampladserne som hængende sammen to og to, som vist i følgende skitse):

3. gennemløb

RAM	adr.
! EF !	EFFF
! FE !	EFFE
! EF !	EFFD
! FC !	EFFC
! 00 !	0003
! 02 !	0002
! 00 !	0001
! 00 !	0000

g) Ramlageret læses igen for at danne signaturer på databussen, og dermed teste det for decodningsfejl.

h) Generer en stoppuls til signaturmeteret.
(Her kan der skrives et "0" til portnr. F1 hex, hvorved der på MPS-23A kortet kommer en negativ flanke ud på stikket J1 ben 1).

i) Programmet laves som en endeløs sløjfe, så der hoppes tilbage for at generere ny startpuls.

---

2) Skriv programmet ud fra struktugrammet og indtast det i filen **ramtest.s**.
Foretag assemblering og eventuelt prøvekørsel i debuggeren.

---

3) Den endelige objektkode programmeres i en EPROM.

---

4) Klargør Comet 3400 og signaturmeter:

MPS-27 kort: fjern EPROM'en (IC 12) og isæt test-EPROM'en i stedet for.

Forbind signaturmeteret til MPS-23A kortet:

Clock: IC 14 ben 11 $\overline{RD}$

Qualifier: IC 14 ben 1 $\overline{M1}$ opcode fetch

Start/stop: J 1 ben 1 portnr. F1 bit 0

NB! CPU-kort (MPS-27) OG RAM-kort (MPS-23A) må

ikke sidde for langt fra hinanden, pga. timings-problemer i COMET'en.

---

5) Indstil signatumeteret:

stilling: SIGNATUR QUALIFIED

CLOCK: positiv flanke.

QUALIFIER: "1".

START: positiv flanke.

STOP: negativ flanke.

---

6) Tænd for COMET'en og kontroller at der kommer start, stop og clock signaler (GATE skal blinke på signaturmeteret). Da det er et ramlager på 60-k der testes, kan gennemløbstiden være helt oppe på et par sekunder.
Optag nogle signaturer på et OK-ramlager. De kan evt. indspilles i signaturmeterets hukommelse (Vcc signaturen bør dog ikke indspilles, da den let kan opstå som en fejl-signatur på andre målepunkter).
Målepunkterne bør omfatte: Vcc, databussen, adressebussen samt CS-logikken.

---

7) Afprøv testen på ramlageret med fejl og undersøg hvilke signaturer der ændrer sig.
Vcc signaturen skal være uforandret, ellers er det et tegn på, at testprogrammet gennemløbes forskelligt afhængig af om kredsløbet er iorden eller ej.
Hvis Vcc signaturen har ændret sig, betyder det at samtlige andre signaturer også vil have ændret sig, og det vil ikke være muligt, ved hjælp af disse, at lokalisere fejlen nærmere.
På mindst en af databussens ledninger skal der komme en fejl-signatur, for at indikere ramfejlen.
Fejl-signaturer på adresse og chip-select ledninger skal kunne lede en hen til punktet, hvor fejlen befinder sig.

Programmeringsøvelse: Guided Probning.

Formål: Formålet med øvelsen er at se hvorledes, man kan programmere "FLUKE 9010A Troubleshooter" til at måle med proben, og hvorledes disse målinger kan anvendes i et "Guided Fejlfindings program".

Udstyr:
- Fluke 9010A Troubleshooter.
- Bånd (minicassette) med program-moduler.
- Comet 3400.

Hjælpemidler:
- 9010A Programming Manual.
- Dokumentation over program-moduler til "Guided Fejlfinding".
- Diagram til Comet 3400 (MPS-23A kortet).

Beskrivelse af opgaven:

I første omgang skal der laves et program, så der med proben kan måles signaturer, samt et program der stimulerer det lille udsnit af kredsløbet (på MPS-23A kortet) der skal testes.
Ved hjælp af disse, skal der optages OK-signaturer til senere brug i fejlfindingsprogrammet.
I anden del af øvelsen skal der laves et "Guided Fejlfindingsprogram" ved hjælp af nogle færdige program-moduler fra båndet, jeres signatur- og stimulerings-programmerer samt evt. et modul mere der mangler. OK-signaturerne skal også bruges til dette.

Gennemførelse:

1) HUSK!!
Inden I begynder at indtaste jeres programmer på "Fluken", skal båndet med de øvrige programmoduler læses ind. På A-siden (der er skrive-beskyttet) ligger de moduler I får færdige; mens I skal bruge B-siden af båndet til at gemme jeres programmer og rettelser på.
Båndet læses ved indtastningen:

```
READ TAPE
ENTER/YES
```

---

2) Skriv "STIMULER" programmet:

Programmet skal stimulere indgangene på det viste

diagramudsnit; dvs. adresse bit 4, 5, 6 og 7 skal udsendes vha. CPU-pod'en, således at der prøves med deres forskellige kombinations-muligheder.

a) Til at udsende adressen kan følgende funktion bruges:

READ REG F

Herved udsendes indholdet af register F som adresse.

b) Ved hjælp af arithmetiske funktioner på register F kan de forskellige adresse-kombinationer laves.

c) Med IF funktionen kan det laves i en "gentag indtil" sløjfe (se Programming Manual side 4-8).

---

3) Indtast programmet som program nr: 10.
Vedrørende åbning og lukning af nye programmer se Programming Manual side 3-1.
Efter indtastningen kan der laves en "back up" på båndets B-side med indtastningen:

WRITE TAPE
ENTER/YES

---

4) Skriv "SIGNATUR" programmet:
Programmet skal med proben udføre en måling på kredsløbet mens det stimuleres, og ud fra probens response skal signaturen isoleres, og i register A returneres til hovedprogrammet.

Se i Programming Manual side 5-12 til side 5-16 sammen med følgende beskrivelse:

a) For at proben kan måle signatur, skal den først synkroniseres, enten til adresse eller data mode. Dette gøres med en af funktionerne:

SYNC A eller: SYNC D

b) Som det næste skal proben nulstilles, vha. funktionen:

READ PROBE

c) Efter at have stimuleret kredsløbet, ved at eksekvere underprogrammet "STIMULER", kan proben nu læses med samme funktion som før:

READ PROBE

Probens response befinder sig nu i register 0, hvor signaturen ligger som bit 8 til 23 (se figur

5-1 i Programming Manual).

d) Inden programmet returnerer til hovedprogrammet skal signaturen befinde sig i register A, liggende fra bit 0 til 15 og med de øvrige bit nulstillet.

---

5) Indtast programmet som program nr: 9. og lav evt. en "back-up" på båndet.

---

6) For at kunne afprøve de to programmer, indtastes følgende lille hjælpeprogram, der kalder jeres "SIGNATUR" program, for derefter at vise register A (signaturen) på displayet.

```
PROGRAM 20
LABEL 0
EXECUTE PROGRAM 9
DPY- SIGNATUR = $A
GOTO 0
```

---

7) Optag OK-signaturer på kredsløbet og noter i skemaet:

Målepunkt	Signatur
IC 18 pin 6	~~78~~ FE72
IC 18 pin 4	F
IC 18 pin 5	113A
IC 16 pin 4	FE
IC 16 pin 5	F16
IC 16 pin 9	334F
IC 16 pin 10	55D1

---

8) Indfør OK-signaturerne i hovedprogrammet (program 1). Find først i programdokumentationen, hvilke "IF"-sætninger det er, der skal rettes. Rettelsen foregår ved at man først sletter sætningen, for derefter at indtaste den pány, med rettelsen (OK-signaturen), se Programming Manual side 3-3.

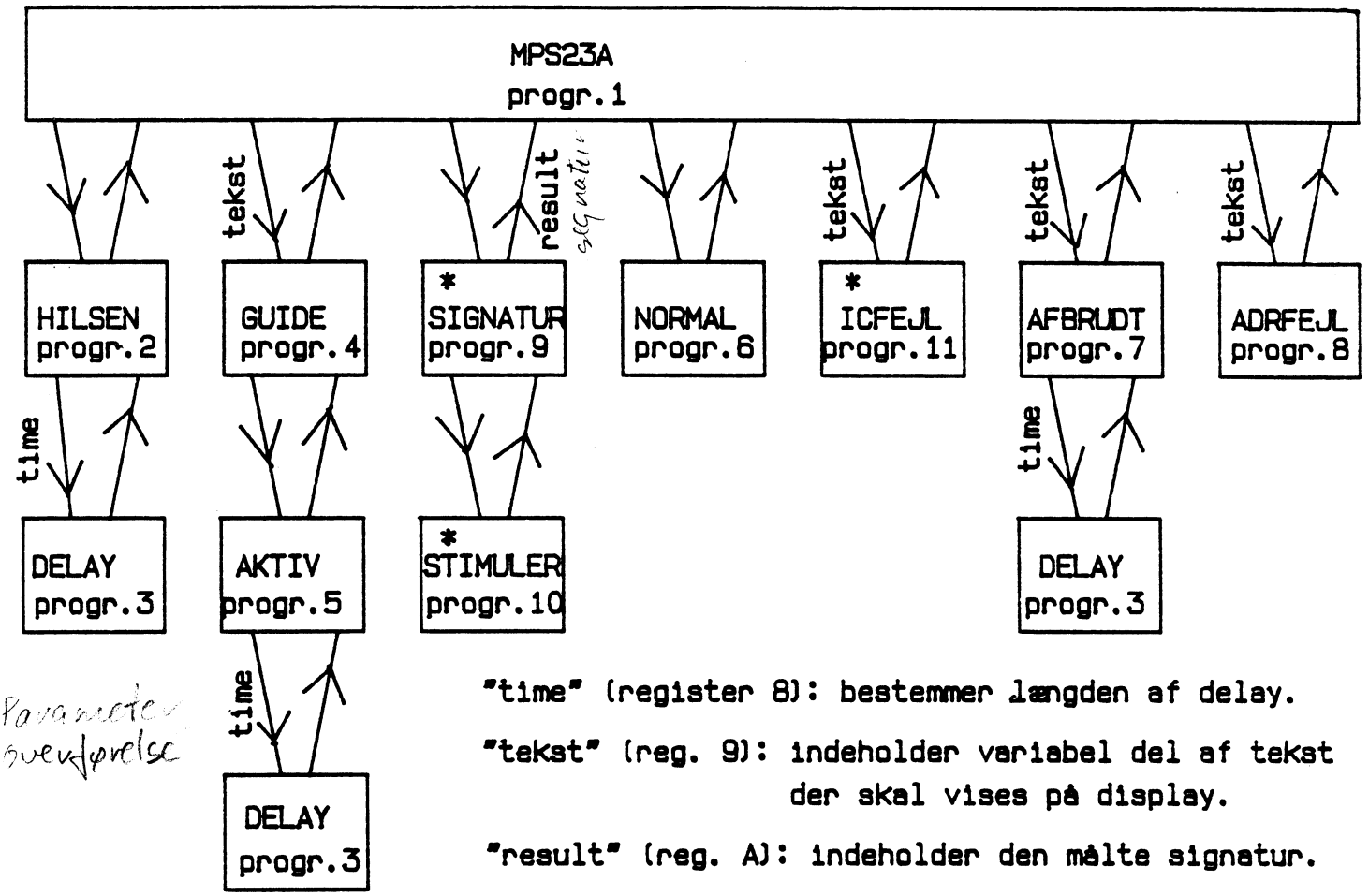
9) Start program 1 og afprøv den guidede fejlfinding på kredsløbet, både uden fejl og med forskellige fejl indført på det.

10) Som I måske har observeret kommer der ingen fejlmelding ved fejl i IC´erne. Underprogrammet "IC-FEJL" (program 11) mangler nemlig, så det er nu jeres opgave at skrive det, så der på displayet kommer teksten:

IC-xx DEFEKT MELLEM PIN-yy OG zz

De variable xx, yy og zz overføres fra hovedprogrammet ved hjælp af et register, (se program dokumentationen).

OVERSIGT OVER PROGRAM-MODULER.

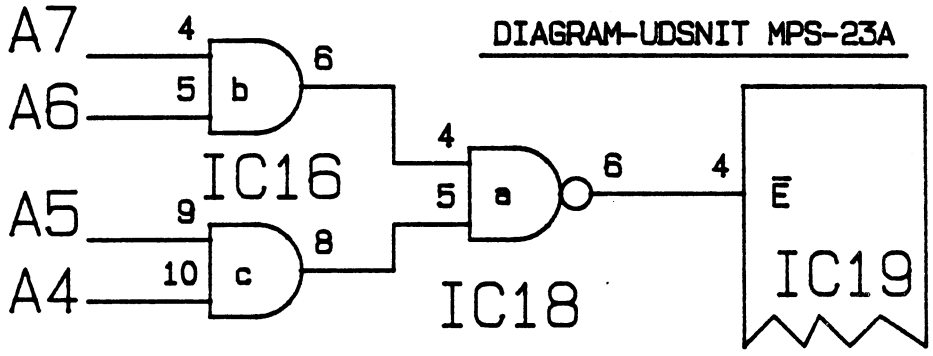


"time" (register 8): bestemmer længden af delay.

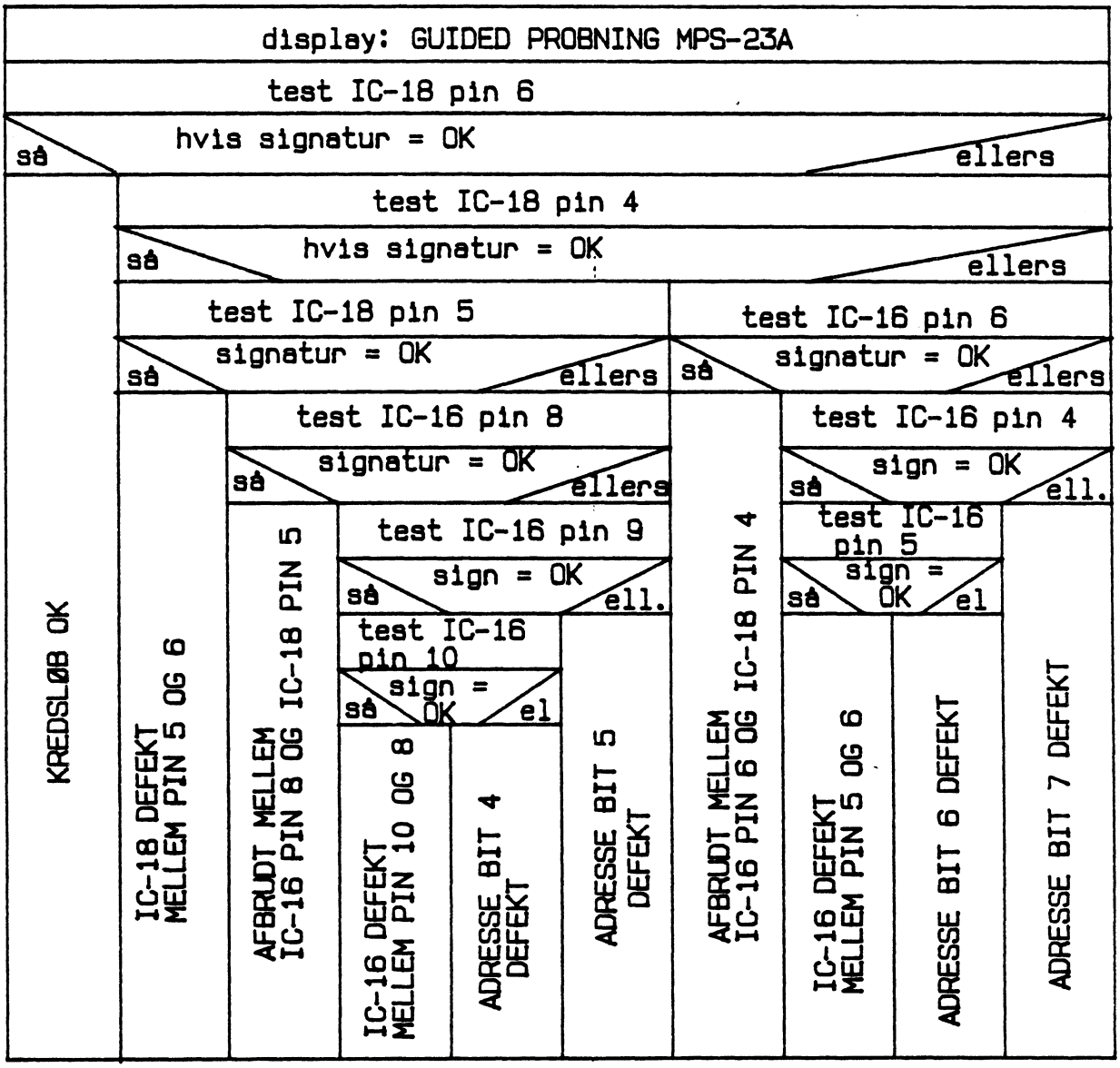
"tekst" (reg. 9): indeholder variabel del af tekst der skal vises på display.

"result" (reg. A): indeholder den målte signatur.

"*" moduler der mangler.



Fejlfindings-træ



AKTIV PROGRAM 5

SYNC FREE-RUN		
gentag	gentag	READ PROBE
	indtil PROBE = "3-state"	
	DELAY: 100 msek.	
	READ PROBE	
indtil PROBE ≠ "high" or "low"		
gentag	gentag	READ PROBE
	indtil PROBE = "high" or "low"	
	DELAY: 200 msek.	
	READ PROBE	
indtil PROBE ≠ "3-state"		
RETURN		

HIGHZ = 2000000 "3-state"
HIGHLOW = 5000000 "high" or "low"

```
1  !        Programmerings opgave i guided probning,
2  !        til FLUKE 9010A.
3  !        ----------------------------------------
4  INCLUDE "Z80.POD"                  ! inkluderer Z80-pod informationer.

5  DECLARATIONS
6   assign REG8 to TIME
7   assign REG9 to TEKST
8   assign REGA to RESULT
9
10 PROGRAM 0
11
12 !_____________________________________________________________________
13
14  !!!!!!!! MPS23A !!!!!!!!
15  !-----------------------
16 PROGRAM 1                  ! her begynder hovedprogrammet.
17
18  EXECUTE 2                 ! skriv "GUIDED PROBNING MPS-23A"
19  REG9 = 1806               ! skriv "PROBE IC-18 PIN-6" og
20  EXECUTE 4                 ! vent til proben er paa plads.
21  EXECUTE 9                 ! maal signaturen med stimulering.
22  IF REGA = 0000 GOTO 0           ! virker kredsloebet?
23  REG9 = 1804               ! ellers test IC-18, pin 4.
24  EXECUTE 4
25  EXECUTE 9
26  IF REGA = 0000 GOTO 1     ! hvis ok: test IC18, pin 5
27
28  REG9 = 1606               ! ellers test IC-16, pin 6.
29  EXECUTE 4
30  EXECUTE 9
31  IF REGA = 0000 GOTO 2     ! hvis ok: saa fandt vi fejlen:
32                            ! afbrudt mellem IC-16p6 og IC-18p4.
33  REG9 = 1604               ! ellers test IC-16, pin 4.
34  EXECUTE 4
35  EXECUTE 9
36  IF REGA = 0000 GOTO 3     ! hvis ok: test IC-16, pin 5
37
38  REG9 = 7                  ! ellers fandt vi fejlen: adresse bit 7
39  EXECUTE 8                 ! defekt.
40  GOTO 4
41
42 3: REG9 = 1605             ! test IC-16, pin 5
43  EXECUTE 4
44  EXECUTE 9
45  IF REGA = 0000 GOTO 5     ! hvis ok: saa fandt vi fejlen:
46                            ! IC-16 defekt mellem pin 5 og 6.
47  REG9 = 6                  ! ellers  fandt vi fejlen: adresse
48  EXECUTE 8                 ! bit 6 defekt.
49  GOTO 4
50
51 5: REG9 = 160506           ! udskriv fejlen: IC-16 defekt mellem
52  EXECUTE 11                ! pin 5 og 6.
53  GOTO 4
54
```

```
 55  2: REG9 =16061804          ! udskriv fejlen: afbrudt mellem
 56   EXECUTE 7                 ! IC-16 pin 6 og IC-18 pin 4.
 57   GOTO 4
 58
 59  1: REG9 = 1805             ! test IC-18 pin 5
 60   EXECUTE 4
 61   EXECUTE 9
 62   IF REGA = 0000 GOTO 6     ! hvis ok: saa fandt vi fejlen:
 63                             ! IC-18 defekt mellem pin 5 og 6.
 64   REG9 = 1608               ! ellers test IC-16 pin 8
 65   EXECUTE 4
 66   EXECUTE 9
 67   IF REGA = 0000 GOTO 7     ! hvis ok: saa fandt vi fejlen:
 68                             ! afbrudt mellem IC-16p8 og IC-18p5.
 69   REG9 = 1609               ! test IC-16 pin 9
 70   EXECUTE 4
 71   EXECUTE 9
 72   IF REGA = 0000 GOTO 8     ! hvis ok: test IC-16 pin 10.
 73
 74   REG9 = 5                  ! ellers fandt vi fejlen: adresse
 75   EXECUTE 8                 ! bit 5 defekt.
 76   GOTO 4
 77
 78  8: REG9 = 1610             ! test IC-16 pin 10
 79   EXECUTE 4
 80   EXECUTE 9
 81   IF REGA = 0000 GOTO 9     ! hvis ok: saa fandt vi fejlen:
 82                             ! IC-16 defekt mellem pin 10 og 8.
 83   REG9 = 4                  ! ellers er fejlen: adresse bit 4
 84   EXECUTE 8                 ! defekt.
 85   GOTO 4
 86
 87  9: REG9 = 161008           ! udskriv fejlen: IC-16 defekt mellem
 88   EXECUTE 11                ! pin 10 og 8.
 89   GOTO 4
 90
 91  7: REG9 = 16081805         ! udskriv fejlen: afbrudt mellem
 92   EXECUTE 7                 ! IC-16 pin 8 og IC-18 pin 5.
 93   GOTO 4
 94
 95  6: REG9 = 180506           ! udskriv fejlen: IC-18 defekt mellem
 96   EXECUTE 11                ! pin 5 og 6.
 97   GOTO 4
 98
 99  0: EXECUTE 6               ! udskriv OK-melding: fungerer normalt.
100
101  4:
102
103
104  !-------------------------------------------------------------------
105
106   !!!!!!!! HILSEN !!!!!!!!
107   !----------------------
108  PROGRAM 2
109                                  ! underprogram der skriver paa
                                     ! display og venter 3 sek.
```

```
110   DPY-GUIDED PROBNING MPS-23A
111   REG8 = 80
112   EXECUTE 3
113
114  !_____________________________________________________________________
115
116   !!!!!!!! DELAY !!!!!!!!
117   !----------------------
118  PROGRAM 3                              ! underprogram der giver et delay
119                                         ! afhaengig af vaerdien i "TIME".
120  0:       IF REG8 = 0 GOTO 1
121   DEC REG8
122   GOTO 0
123  1:
124
125  !_____________________________________________________________________
126
127   !!!!!!!! GUIDE !!!!!!!!
128   !----------------------
129  PROGRAM 4                              ! underprogram der anmoder brugeren
130                                         ! om at probe et punkt og venter
131  DECLARATIONS                           ! til proben er paa plads.
132   assign REG1 to ICnr
133   assign REG2 to PINnr
134
135   REG1 = REG9 SHR SHR SHR SHR SHR SHR SHR SHR AND FF
136   REG2 = REG9 AND FF
137   DPY- PROBE IC-$1   PIN-$2 #
138   EXECUTE 5
139
140  !_____________________________________________________________________
141
142   !!!!!!!! AKTIV !!!!!!!!
143   !----------------------
144  PROGRAM 5                              ! underprogram der undersoeger om
145                                         ! proben er paa plads ( ikke
146  DECLARATIONS                           ! 3-statet i mindst 200 m.sek ).
147   assign REG0 to PROBEDATA
148   assign REG1 to HIGHZ
149   assign REG2 to HIGHLOW
150
151   REG1 = 2000000                        ! maske for "3-state"
152   REG2 = 5000000                        ! maske for "high" eller "low"
153
154   SYNC FREE-RUN
155  0:    READ PROBE                      ! vent til proben har sluppet
156        REG0 = REG0 AND REG1            ! det forrige maalepunkt
157     IF REG0 = 0 GOTO 0
158     REG8 = 8                          ! 100 m.sek. delay
159     EXECUTE 3
160     READ PROBE                        ! er proben stadig "3-state"?
161     REG0 = REG0 AND REG2
162   IF REG0 > 0 GOTO 0
163  1:    READ PROBE                     ! vent til proben er paa nyt
164        REG0 = REG0 AND REG2            ! maalepunkt.
```

```
165     IF REG0 = 0 GOTO 1
166     REG8 = 10                       ! 200 m.sek. delay
167     EXECUTE 3
168     READ PROBE                      ! er proben stadig paa nyt
169     REG0 = REG0 AND REG1            ! maalepunkt?
170   IF REG0 > 0 GOTO 1
171   DPY-+ VENT                        ! saa er vi klar til at maale.
172
173  !_____________________________________________________________________
174
175   !!!!!!!! NORMAL !!!!!!!!
176   !-----------------------
177  PROGRAM 6                          ! underprogram der skriver paa display
178
179   DPY- FUNGERER NORMALT#
180
181  !_____________________________________________________________________
182
183   !!!!!!!! AFBRUDT !!!!!!!!
184   !----------------------
185  PROGRAM 7                         ! underprogram der skriver "AFBRUDT
186                                    ! MELLEM IC-nr PIN-nr OG IC-nr PIN-nr"
187  DECLARATIONS
188   assign REG1 to ICnr1
189   assign REG2 to PINnr1
190   assign REG3 to ICnr2
191   assign REG4 to PINnr2
192
193   REG4 = REG9 AND FF                          ! split "TEKST" op.
194   REG9 = REG9 SHR SHR SHR SHR SHR SHR SHR SHR
195   REG3 = REG9 AND FF
196   REG9 = REG9 SHR SHR SHR SHR SHR SHR SHR SHR
197   REG2 = REG9 AND FF
198   REG9 = REG9 SHR SHR SHR SHR SHR SHR SHR SHR
199   REG1 = REG9 AND FF
200                                       ! saa er det klar til display.
201   DPY-AFBRUDT PRINTBANE MELLEM #
202   REG8 = 80                           ! delay inden naeste linie
203   EXECUTE 3                           ! displayes.
204   DPY-IC-$1, PIN-$2, OG
205   DPY-+  IC-$3, PIN-$4#
206
207  !_____________________________________________________________________
208
209   !!!!!!!! ADRFEJL !!!!!!!!
210   !------------------------
211  PROGRAM 8                         ! underprogram der skriver
212                                    ! "ADRESSE BIT-nr DEFEKT" paa display.
213  DECLARATIONS
214   assign REG1 to BITnr
215
216   REG1 = REG9 AND FF
217   DPY- ADRESSE BIT-$1 DEFEKT #
218
219  !_____________________________________________________________________
```

Programmeringsøvelse: Key-buffer test.

Formål: Formålet med øvelsen er at indlære hvorledes man kan skrive et testprogram, der er i stand til at teste nogle perifere kredsløb. Programmet skal kunne bruges, både til en "go / no go" test, samt til stimulering af en signaturmåling i forbindelse med fejlfinding på kredsløbet.

Udstyr:
Udviklingsudstyr PMDS.
EPROM 2732 (slettet).
Comet 3400 med MPS-10 kort (32 bit I/O).
Dummy-kabel til at forbinde fra MPS-10 til key-board-fatning.
Stik med lysdioder til indikering.
Signaturmultimeter.

Hjælpemidler:
Diagrammer til Comet 3400.
Princip-diagram og beskrivelse af key-buffer.

Beskrivelse af opgaven:

Programmet skal på MPS-27 kortet teste de kredsløb der indgår i den buffer der har til opgave at modtage karakterer fra et keyboard.

Med fjernet keyboard etableres en forbindelse fra en udgangsport på MPS-10 kortet til indgangen af bufferen, således at programmet via denne udgangsport kan stimulere bufferen med forskellige testmønstre. Disse testmønstre kan fra bufferen, der har plads til max. 9 bytes af gangen, læses ind på databussen, hvor CPU'en kan checke dem og på denne måde teste bufferens funktion.

Da programmet også skal kunne bruges til fejlfinding på kredsløbet, ved hjælp af signaturmålinger, er det nødvendigt at der genereres start og stop signaler til signaturmeteret.

Endvidere skal det sikres at programforløbet, i den del af programmet hvor der optages signatur, gennemløbes ens, uanset om der er fejl på det testede kredsløb eller ej. Forgreninger i programmet må altså først forekomme efter, at der er genereret stop-signal.

KEY-BUFFER TEST.

Gennemførelse:

1) Inden man kan begynde at skrive testprogrammet er det nødvendigt først at gennemgå kredsløbets virkemåde. Se derfor først på "Princip-diagram og beskrivelse" af key-bufferen og sammenhold denne beskrivelse med det rigtige diagram over MPS-27 kortet.

---

2) Tegn et struktugram evt. ud fra følgende hjælp:

a) Start med at tænde LED 1 for at indikere at testen er startet.

b) Initialiser PIO (se beskrivelsen over externe forbindelser), og sæt stroben "high".

c) Udsend startbit til signaturmeteret.

d) Nu kan selve testen starte: Bufferen fyldes med de forskellige testmønstre ved hjælp af PIO'en, hvor der efter hver byte skal sendes en impuls på stroben.

Testmønstrene kan f.eks. laves ved en binær optælling af de udsendte data, så der testes med samtlige 256 kombinationer af disse. Men da bufferen maksimum kan indeholde 9 bytes af gangen, må det antal bytes der fyldes i den, ikke overstige disse 9, inden bufferen tømmes igen. F.eks. kan dataerne sendes i "klumper" på 8 bytes af gangen, så vil det passe at man efter 32 "klumper" har prøvet samtlige 256 kombinationer.

For hver "klump" tømmes bufferen, imens der testes for det rigtige indhold. Da denne testning ikke må indeholde betingede hop, pga. signaturmålingen, kan den laves som vist, når man skal til at sætte kode på sit struktugram:

```
in a,(BUFFER)   ; hent data der skal testes.
xor *OKDATA*    ; afleverer "1" i acc. hvis
                ; testdata ikke svarer til
                ; forventet data (*OKDATA*).
or e            ; testresultat bliver OR'et
ld e,a          ; sammen med tidligere test-
                ; resultat.
```

Det forudsættes at register E, der bruges til opsummering af testresultat, er nulstillet inden testen startes. Efter testen vil eventuelle "1"-taller i register E fortælle, hvilke bitpositioner der har været fejl på. De forventede data *0KDATA*

kan evt. være indholdet i et CPU-register.

Husk på, at selv om der kun er fyldt 8 bytes i bufferen, skal den hver gang læses 9 gange for at være tømt, det medfører at man første gang den læses skal finde "FF".

Sæt det forannævnte ind i de nødvendige "gentage sløjfer", så der bliver overført og testet de nævnte datakombinationer.

e) Udsend stoppuls til signaturmeteret.

f) Undersøg testresultatet og tænd LED 2 hvis testen viser at bufferen var iorden, mens der i modsat fald slukkes for LED 2.

g) Testprogrammet gentages kontinuerligt, for at det kan bruges til signaturmålingen.

---

3) Skriv programmet ud fra struktugrammet og indtast det i filen: **buftest.s**

Foretag assemblering og eventuelt afprøvning i debuggeren.

---

4) Den endelige objektkode programmeres i en EPROM, type 2732.

---

5) Klargør Comet 3400 og signaturmeter:

a) MPS-27 kort: fjern EPROM'en (IC 12) og isæt test-EPROM'en i stedet for.

b) MPS-23A kort: isæt stikket med lysdioder.

c) Forbind "dummy kabel" mellem MPS-10 og indgang af key-buffer (MPS-27). Se tegning over externe forbindelser.

d) Forbind signaturmeter:

Clock: IC-25 ben 3 (MPS-27) systemclock

Start: J 1 ben 1 (MPS-23A) portnr. F1 bit 0

Stop: J 1 ben 1 (MPS-23A) portnr. F1 bit 0

KEY-BUFFER TEST.

6) Indstil signaturmeter:

stilling:	SIGNATUR NORMAL.
CLOCK:	negativ flanke.
START:	positiv flanke.
STOP:	negativ flanke.

---

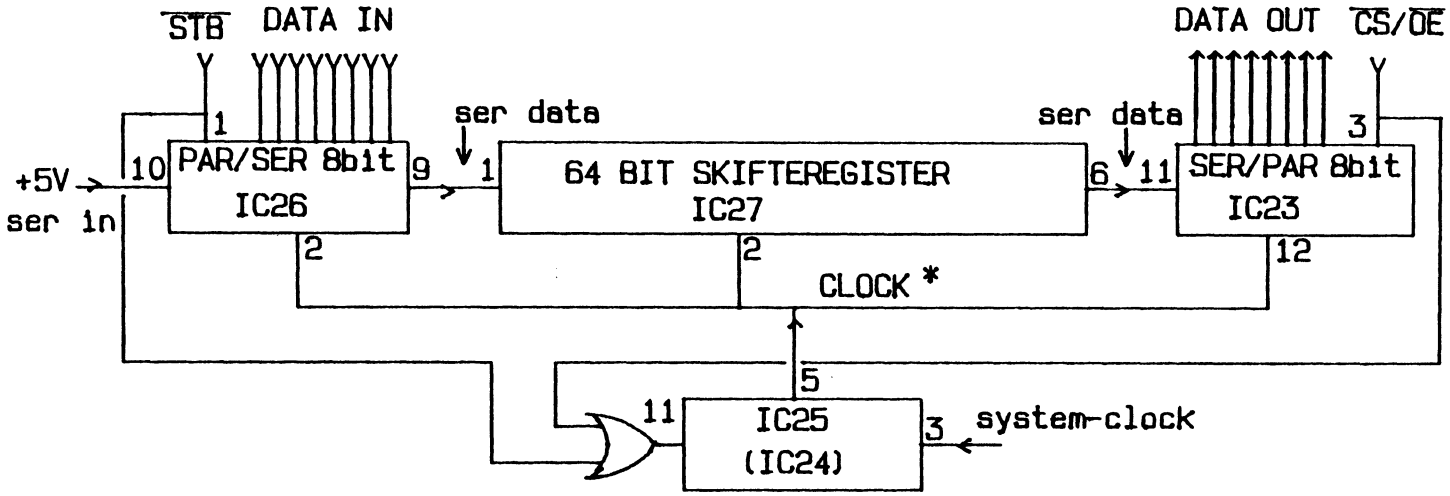
7) Tænd for COMET'en og afprøv testprogrammet først med indikeringerne på lysdioderne, henholdsvis med og uden fejl på kredsløbet.

Mål Vcc signaturen uden fejl på kredsløbet, og kontroller at den ikke ændrer sig når der indføres fejl.

Optag nogle ok-signaturer i key-bufferen og afprøv hvilke der ændrer sig med forskellige fejl på kredsløbet.

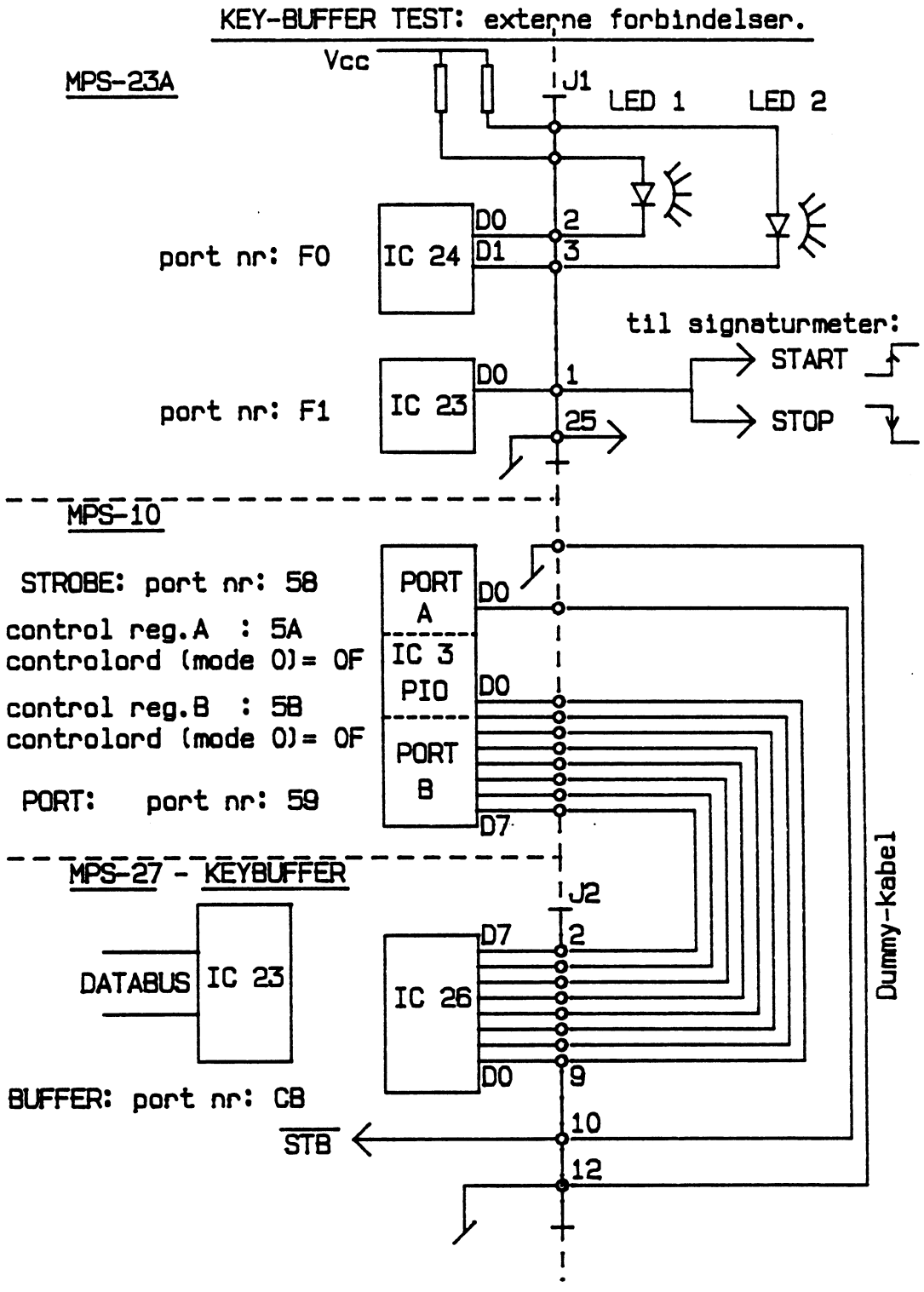
Key-buffer. MPS-27

Princip-diagram og beskrivelse.



* Kredsløbet med IC25, IC24 genererer 8 clock-pulser efter hver puls på enten STB eller CS/OE (frekvensen er det halve af "system clock").

Når stroben STB har parralel-indlæst data i IC26, vil dataerne automatisk blive skiftet 8 bit til højre, når STB igen går high. Efter yderligere 9 pulser på enten STB eller CS/OE vil dataerne ligge i IC23 parat til at blive parralel-udlæst.
Der er således plads til 9 data-bytes i bufferen.
Hvis det er CS/OE der skifter dataerne til højre, vil der blive seriel-indlæst databyten "FF" i IC26; dvs. en tom buffer indeholder udelukkende databytene "FF".


KEY-BUFFER TEST: externe forbindelser.
MPS-23A
Vcc
J1
LED 1
LED 2
port nr: F0
IC 24
D0
D1
2
3
til signaturmeter:
START
STOP
port nr: F1
IC 23
D0
1
25
MPS-10
STROBE: port nr: 58
control reg.A : 5A
controlord (mode 0)= 0F
control reg.B : 5B
controlord (mode 0)= 0F
PORT: port nr: 59
PORT A
IC 3 PIO
PORT B
D0
D0
D7
MPS-27 - KEYBUFFER
J2
DATABUS
IC 23
IC 26
D7
2
D0
9
BUFFER: port nr: CB
STB
10
12
Dummy-kabel

Programmeringsøvelse: Rom-test.

Formål:

Formålet med øvelsen er at indøve hvorledes man kan teste en rom eller eprom for dens korrekte bitmønster indhold, samt at se hvorledes dette testprogram skal skrives for at kunne køre i en "In-Circuit Emulator", Computest uSA.

Udstyr:

Udviklingsudstyr PMDS.
EPROM 2716 (slettet).
Computest uSA.
Comet 3400.

Hjælpemidler:

User Manual til uSA.
Struktugram over Cyclic Redundancy Check.
Udskrift af filen: **romtest.s**

Beskrivelse af opgaven:

Der skal laves en "Diagnostic Prom" til uSA'en, d.v.s. en eprom (indeholdende et eller flere testprogrammer) der sættes i forpladen på uSA'en. uSA'ens cpu-pod sættes ind istedet for cpu'en i computeren der skal testes (Comet 3400), og ad denne vej kan diagnostic-programmet nu teste Comet'en, mens testresultater o.lign. kan vises på displayet i uSA'en.

Der skal laves et testprogram der måler en signatur (Cyclic Redundancy Check) for Comet'ens ROM og viser denne 8-bits signatur på displayet.
Diagnostic Prom'en skal udover selve testprogrammet, der skrives i Z80-kode, også indeholde oplysninger til uSA'en (Diagnostic Header, Subtest Header, tekst linier m.m.). De sidstnævnte control koder skrives på uSA'ens præmisser og er beskrevet i dens User Manual.

På udviklingsudstyret ligger en fil (**romtest.s**) hvor en del af disse ting er lavet på forhånd, så jeres opgave bliver at tilføje de resterende oplysninger sammen med selve testprogrammet, der måler signaturen.

I filen **romtest.s** ligger en beskrivelse af forskellige "macro-kald" der kan bruges til at generere control koderne til uSA'en.

ROM-TEST.

Gennemførelse:

1) Inden vi begynder at skrive selve testprogrammet er det vigtigt først at se på, hvordan det skal præsentere sig overfor brugeren, dvs. hvad skal der displayes:

a) Når diagnostic prom'en selectes med "PROM/MEM" knappen skrives der en initial-tekst på displayet f.eks.:

* COMET DIAGNOSTIC *

b) Der kan laves op til 99 testprogrammer i diagnostic prom'en, så det er en fordel, hvis hver af disse subteste starter med at præsentere sig på displayet, når de vælges med "SUBSEL" knappen. Vores testprogram, der får subtest nr. 01, kan starte med at skrive:

01* ROM-TEST

c) Når rom-testen er færdig, skal resutatet dvs. rom'ens signatur displayes. Ud over denne 8-bits signatur, der displayes som variable data, skal der være en tekst der fortæller, hvad der vises.

Skriv herunder hvad der skal displayes, når testen er færdig. Der er plads til max. 20 karakterer og de gyldige karakterer kan findes i User Manual side A-1:

display: _

---

2) ASCII MESSAGE:

a) Ved hjælp af editoren åbnes filen: romtest.s. Find i denne fil afsnittet: ASCII MESSAGE: der beskriver, hvordan man med et macro-kald laver en tekst-string.

Der er i forvejen lavet 2 tekst-stringe:

* COMET DIAGNOSTIC *

og: 01* ROM-TEST

b) Tilfør den eller de tekst-stringe, der skal til for at skrive jeres tekst.

---

3) MESSAGE LIST:

a) Det der displayes kan bestå af en eller flere tekst-stringe, samt eventuelt nogle variable data. De forskellige dele bindes sammen med en "message list". Denne "message list" består af lige så mange "message list entry", som der er af tekst-stringe og data, der ønskes på displayet af gangen.

b) Find afsnittet: MESSAGE LIST: der beskriver syntaxen for det macro-kald, der skal bruges til at danne de enkelte "message list entry".
Der er i forvejen lavet "message list" til de 2 første tekster der display´es.

c) Tilføj "message listen" til det I ønsker display´et, når rom-testen er færdig.

---

4) TEST-PROGRAM:
Find sidst i filen afsnittet: TEST-PROGRAMMER: hvor selve programmet skrives.

a) De 2 første instruktioner er lavet på forhånd, hvor linien:

```
reque(S1LIST,20,0x1A)
```

er et macro-kald der genererer "service request", for at starte programmet med at display´e teksten:

```
01*   ROM-TEST
```

Syntaxen for dette macro-kald er beskrevet i afsnittet: SERVICE REQUEST:

b) Resten af testprogrammet, som I nu skal til at lave, tilføjes sidst i filen hvor det står anvist. Lad labelen ENDING: stå efter jeres sidste instruktion, for at markere programslut.

Rom-testen skal laves efter "Cyclic Redundancy Check" princippet og som resultat vise en 8-bits signatur. Skriv programmet ud fra struktugrammet over "Cyclic Redundancy Check".

Signaturen afleveres til sidst ud i "shadow ram men" på den hukommelsesplads, som I har angivet med dens label i "message listen".

I skal huske at reservere denne hukommelsesplads i "shadow rammen". Dette kan, i begyndelsen af sub-test-programmet, gøres med sætningen:

XXXX = SEREQ + 5

XXXX erstattes med jeres label og "SEREQ + 5" er første ledige plads i "shadow rammen" (de første pladser bruges til "service request" i forbindelse med macro-kaldet "reque").

---

5) Assemblering:

a) Når programmet er skrevet i filen **romtest.s** assembleres denne og rettes for eventuelle syntax fejl.

b) Objektkoden programmeres i en eprom, type 2716, vha. programmeringsprogrammet der startes op med:

sprom proc16

Programmet programmeres fra adresse 0000 i eprom-men.

---

6) Klargør Comet 3400 og Computest uSA:

a) Med Comet og uSA slukket fjernes CPU'en i Cometen og erstattes af CPU-poden fra uSA'en.

b) Jeres Diagnostic-Prom sættes i soklen PROM 1 på uSA'en, med ben 1 op mod udløsestangen.

c) Tænd først for Comet'en, og dernæst for uSA'en. Hvis der slukkes skal det ske i modsat rækkefølge, for at beskytte CPU-poden.

---

7) Afprøvning af Diagnostic-Prom:

a) Med knappen: PROM/MEM kobles Diagnostic-Prom'en ind i memmory-mappen og initial-teksten:

* COMET DIAGNOSTIC *

skal nu vise sig på displayet.

b) Rom-testen vælges med følgende indtastning:

SUBSEL
0
1
ENTER

c) Det valgte testprogram kan nu startes med knappen: RUN, hvorefter der på display'et skulle komme teksten:

01* ROM-TEST

Efter et par sekunder (der er bestemt af "WAIT TIME" i forbindelse med "service request"), starter selve test-programmet, der skulle slutte med at vise den resulterende signatur på displayet.

d) Afprøv test-programmet med forskellige rom'er i Cometen, og kontroller at de giver forskellig signatur.

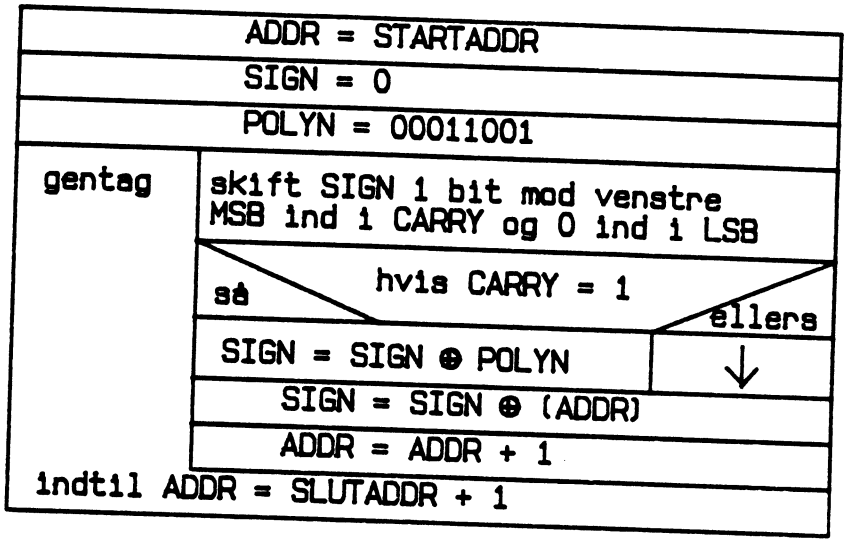
```
! Modulname:              romtest.s
! Mainprogram:            none
! Programtype:            assembler
! Date:
! Programmer:
! Function:               diagnostic prom
! Register used
! as inputs:              none
! Register used
! as outputs:             none
! Subroutine calls:       none
! Destroys reg.:          all
! Language:               PCP m4 + z80 assembler
! Debug:
! Debugger:
! Last correction:
!*********************************************************
! CONTROL INFORMATION *
!**********************

ORIGEN  = 0xA000        ! Diagnostic Proms start adr.
!
        .sect   .text
        .base   ORIGEN
        include(mac)
!*********************************************************
! DIAGNOSTIC HEADER: *
!**********************
                        ! PROGRAM ORIGEN: start adr.for
                        ! diagnostic eprom:
hilo(                           ORIGEN)
!---------------------------------------------------------
                        ! ENDING ADDRESS: slut adr. + 1
                        ! for diagnostic program:
hilo(                           ORIGEN+0x400)
!---------------------------------------------------------
                        ! NUMBER OF PROMS: antal eprom
                        ! (1 eller 2):
.data1                          1
!---------------------------------------------------------
                        ! MESSAGE LIST POINTER: skriv
                        ! label for Message List, der
                        ! skal vises som initial tekst:
hilo(                           ORIGEN+0x200)
!---------------------------------------------------------
                        ! NUMBER OF SUBTESTS:
                        ! (mellem 1 og 99):
.data1                          1
!---------------------------------------------------------
SEREQ   = ORIGEN        ! SERVICE REQUEST BYTE POINTER:
                        ! angiver hvor service request
                        ! ligger i shadow ram.
hilo(                           SEREQ)
!---------------------------------------------------------
                        ! FILTER CRITERIA POINTER:
                        ! (0 = ingen filter):
```

```
hilo(                              0)
!-------------------------------------------------------
                          ! SUBTEST1 POINTER: skriv
                          ! "HEADx" hvor "x" erstattes
                          ! med subtest nummer:
hilo(                                HEAD1)
!-------------------------------------------------------

        ! evt. flere subtest pointere:

.align  0x80
!*********************************************************
! SUBTEST HEADERS: *
!******************                                  ifdef(x,`
! Syntax for macro-kald der genererer Subtest
! Header:
!       subhead(nr,map,filter)
!       ^^^^^^^^^^^^^^^^^^^^^^
!            Parametre:
!       nr       = subtest nummer ( mellem 1 og 99 ).
!       map      = FUNCTIONS SPECIFIED MAP:
!                  0 = memory mapped til UUT, og ingen
!                  "fault isolation" maalinger.
!       filter   = FILTER CRITERIA POINTER:
!                  0 = ingen filter.
!
!          for yderligere oplysninger om "map
!          og filter" se side 6-6 i User Manual.
! ( NB! dette macro-kald kan kun bruges til Subtest
!   Headers uden "fault isolation" maalinger.)
!
! skriv macro-kald med parametre herunder:
!*************************************************')
        subhead(1,0,0)


!*********************************************************
! PROGRAM BODY: *
!****************
.align  0x100
!*********************************************************
! ASCII MESSAGE: *
!*****************                                    ifdef(x,`
! Syntax for macro-kald der genererer
! Ascii Message ( tekst-string ):
!       string(nr,tekst)
!       ^^^^^^^^^^^^^^^^
!          Parametre:
!       nr       = nummer paa tekst-string.
!       tekst    = tekst-string; se side A-1 i User
!                  Manual for gyldige karakterer.
!
! skriv macro-kald med parametre herunder:
!*************************************************')

        string(1,* COMET DIAGNOSTIC *)
```

```
        string(2,01*   ROM-TEST)


!*******************************************************
! MESSAGE LIST: *
!***************
.align  0x100
!*********************************************ifdef(x,`
! Syntax for macro-kald der genererer
! Message List Entry:
!       messlist(label,opcod,lenth,point)
!       ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
!          Parametre:
!       label   = symbolsk navn der skal bruges som
!                 "MESSAGE LIST POINTER" naar message
!                 skal displayes.
!       opcod   = OPCODE: angiver operation type
!                 og data format.
!       lenth   = FIELD LENGTH: antal karakterer
!                 i tekst.
!       point   = ADDRESS POINTER:
!                 Ved Ascii Message: skriv "STRINGx"
!                 hvor "x "er nr. paa tekst-string.
!                 Ved data Message: skriv label for
!                 hukommelsespladsen hvor data ligger.
!
!          for yderligere oplysninger se side
!          9-4 i User Manual.
!
! skriv macro-kald med parametre herunder:
!*****************************************************')

! initial tekst: vises naar PROM/MEM aktiveres
        messlist(DIALIST,0x80,20,STRING1)

! subtest1 tekst:
        messlist(S1LIST,0x80,14,STRING2)


!*************************************************ifdef(x,`
! SERVICE REQUEST: *
!******************
! Syntax for macro-kald der via Shadow Ram anmoder
! uSAen om at udfoere en funktion; f.eks at dis-
! playe noget:
!       reque(message,wait,service)
!       ^^^^^^^^^^^^^^^^^^^^^^^^^^
!          Parametre:
!       message = MESSAGE LIST POINTER: skriv label
!                 for Message List, der skal vises
!                 paa displayet.
!       wait    = WAIT TIME: tid = vaerdi * 0,1 sek.
!       service = SERVICE REQUEST BYTE: bestemmer
!                 funktionen der skal udfoeres.
!
```

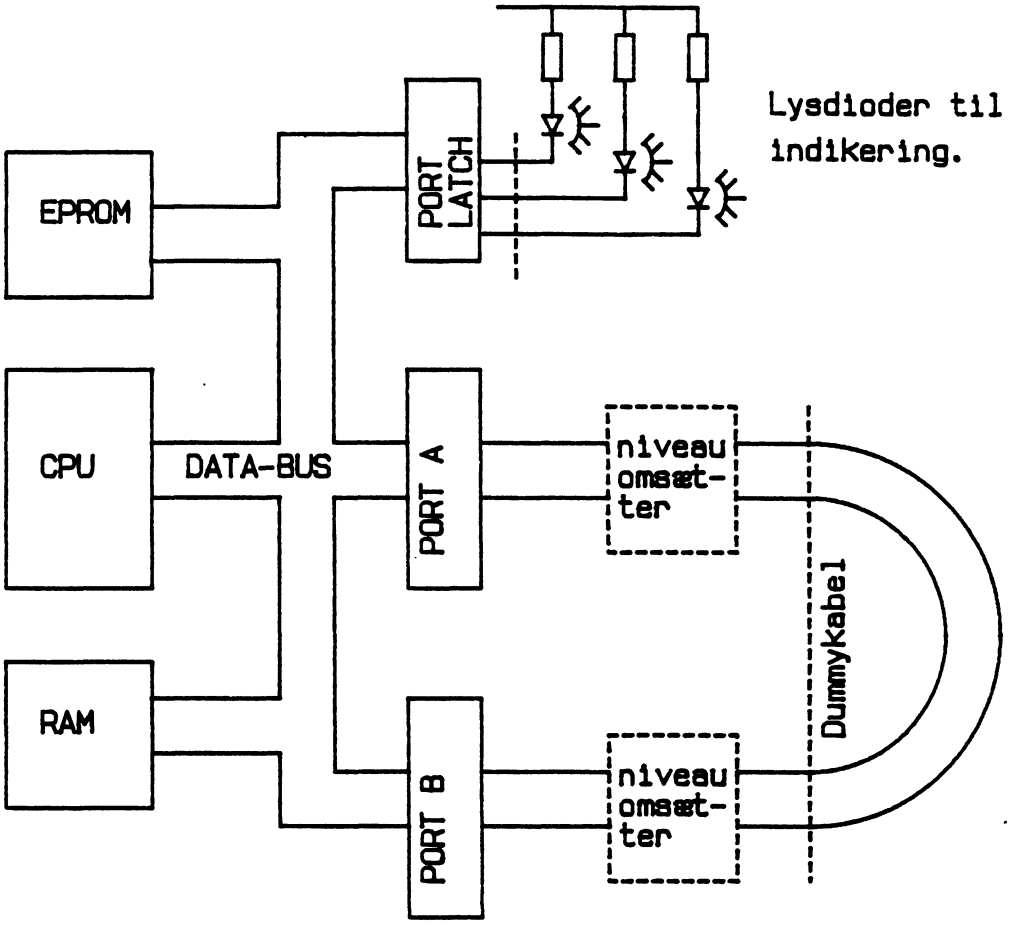
```
!             for yderligere oplysninger se side
!             8-2 i User Manual.
!
! macro-kald med parametrene skrives de aktuelle
! steder i test-programmet.
!*****************************************************')
! TEST-PROGRAMMER: *
!*******************
!
! SUBTEST 1:
!***********
.align  0x80
! definitioner:
        ROMB    = 0xF800        ! ROM start adresse
        ROMSL   = 0x0000        ! ROM slut adresse + 1

! reservering af rampladser i shadow ram:


! program:
SUBT:   ld sp,ORIGEN+0xFF       ! init stackpointer

                        ! skriv initial-tekst: 01*  ROM-TEST
        reque(S1LIST,20,0x1A)

! Fortsaet med resten af testprogrammet her:
!

ENDING:
```

Cyclic Redundancy Check



⊕ = bit for bit EXCL-OR

Princip for PORT-TEST.



Indiker: test startet
Sæt testparametre op
Udfør test
Indiker: test kører endnu
Udfør test
Indiker: testresultat

RAM-TEST

Fejl-typer:

Read/Write Error (Skrive/læse fejl):
let af
Hukommelsesceller kan ikke 0- eller 1-stilles.
finde
Kan optræde på enkelte databit eller i hele databredden.

Decod Error (Adresse fejl):
Fejl ved adressebussen eller decodningslogikken.
Der er tilgang til samme hukommelsesplads på to eller flere forskellige adresser.

Bitafsmitnings fejl:
Der er overhøring mellem hukommelsescellerne.
Kan være mønsterafhængig.

Forskellige test-algoritmer:

1) March:

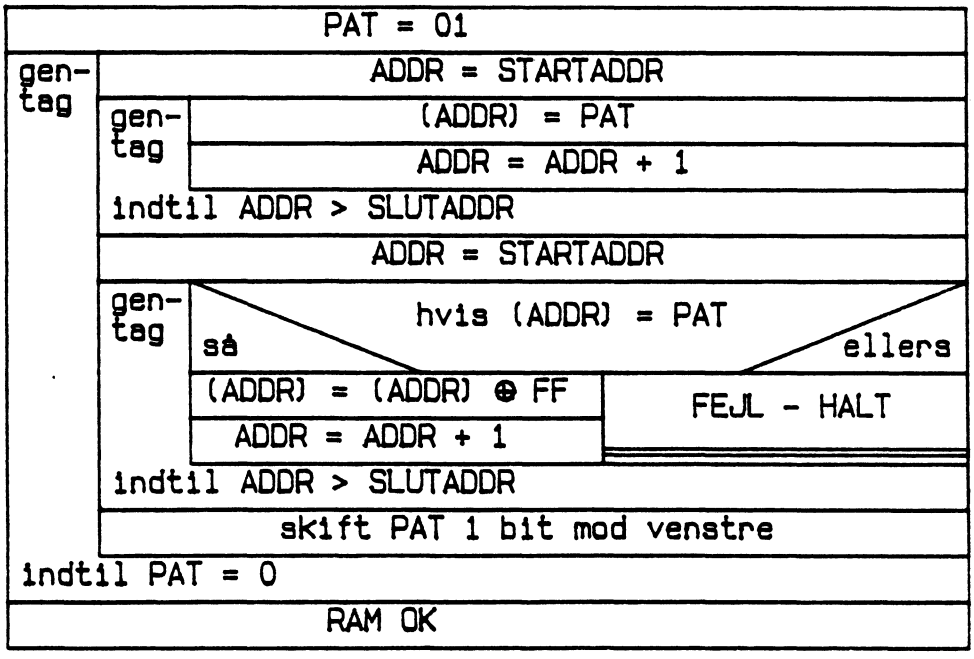
Fordele:

God til at undersøge for Read/Write Error, og Decod Error.
Rimelig kort tidsforbrug.

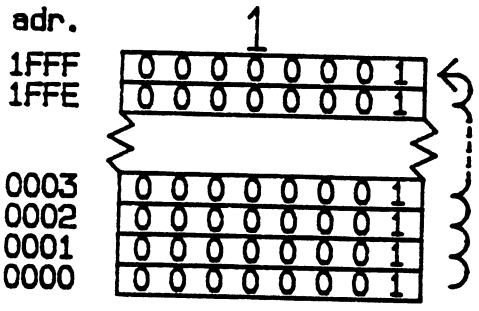
Ulemper:

Dårlig til at finde Bitafsmitnings fejl.
Kan kun bruges til Go/No Go test (giver ikke noget testresultat der ved analyse kan fortælle noget om fejlens nærmere pladsering).
Destruerer det oprindelige indhold i ramlageret (testen er ikke transparent).

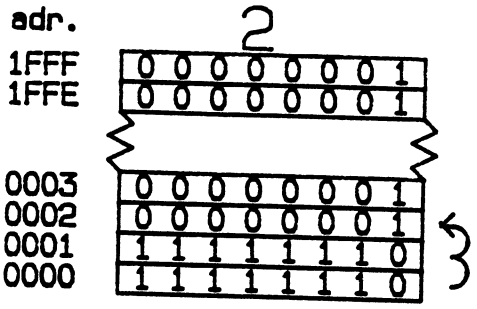
RAM-test: MARCH



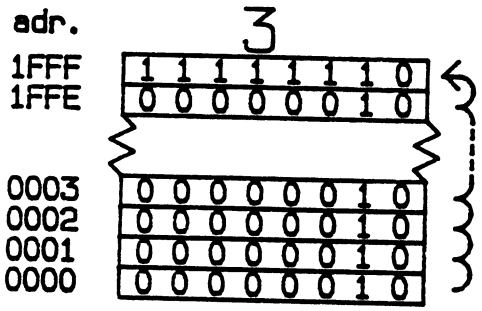
MARCH



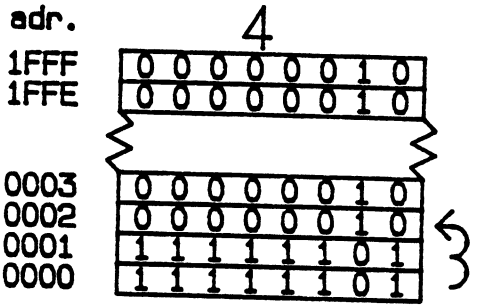
Rammen fyldes med mønstret 01



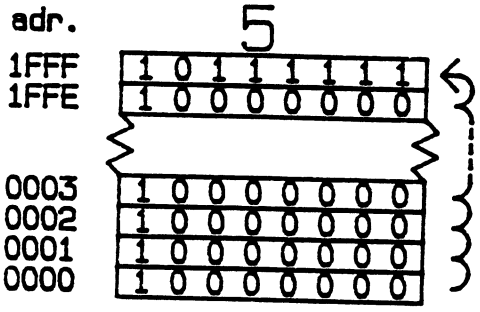
Adresseområdet læses og checkes for korrekt indhold, hvorefter indholdet inverteres.



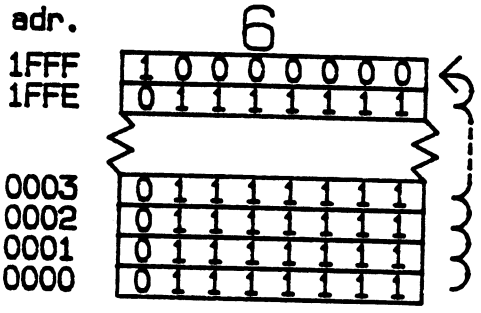
Testmønstet skiftes 1 plads mod venstre, og skrives i adresseområdet.



Det nye indhold checkes og inverteres igen.



Sidste testmønster der skrives i rammen



og checkes på samme måde.

RAM-TEST.

```
! Modulname:              march
! Mainprogram:           none
! Programtype:           assembler
! Date:                  8-4-86
! Programmer:            Svend Erik Bertelsen
! Function:              ram-test
! Register used
! as inputs:             none
! Register used
! as outputs:            none
! Subroutine calls:      none
! Destroys reg.:         all
! Language:              PCP z80 assembler
! Debug:
! Debugger:
! Last correction:
!-------------------------------------------------

        ! Definitioner:
PAT      =  0x01        ! testmoenster.
startadr =  0           ! startadresse  for ramlager.
slutadr  =  0x2000      ! slutadresse + 1 for ramlager.

        ! Startadresse for program:
        .sect   .text
        .base   0xF800

        ld  de,slutadr
        ld  b,PAT                   ! PAT = 01
skift:  ld  hl,startadr             ! addr = startaddr
fyld:   ld  (hl),b                  ! (addr) = PAT
        inc hl                      ! addr = addr + 1

        ld  a,l                     ! gentag "fyld" indtil
        cp  e                       ! addr > slutaddr.
        jp  nz,fyld
        ld  a,h
        cp  d
        jp  nz,fyld

        ld  hl,startadr             ! addr = startaddr
test:   ld  a,(hl)                  ! test om: (addr) = PAT
        cp  b
        jp  z,ok
        halt                        ! stop hvis (addr) <> PAT
ok:     xor 0xFF                    ! hvis (addr) = PAT:
        ld  (hl),a                  ! (addr) = (addr) excl-or FF
        inc hl
        ld  a,l                     ! gentag "test" indtil
        cp  e                       ! addr > slutaddr.
        jp  nz,test
        ld  a,h
        cp  d
        jp  nz,test

        sla b                       ! skift PAT 1 bit mod venstre.
        jp  nc,skift                ! gentag "skift" indtil PAT = 0

        ! RAM OK
```

2) Galpat:

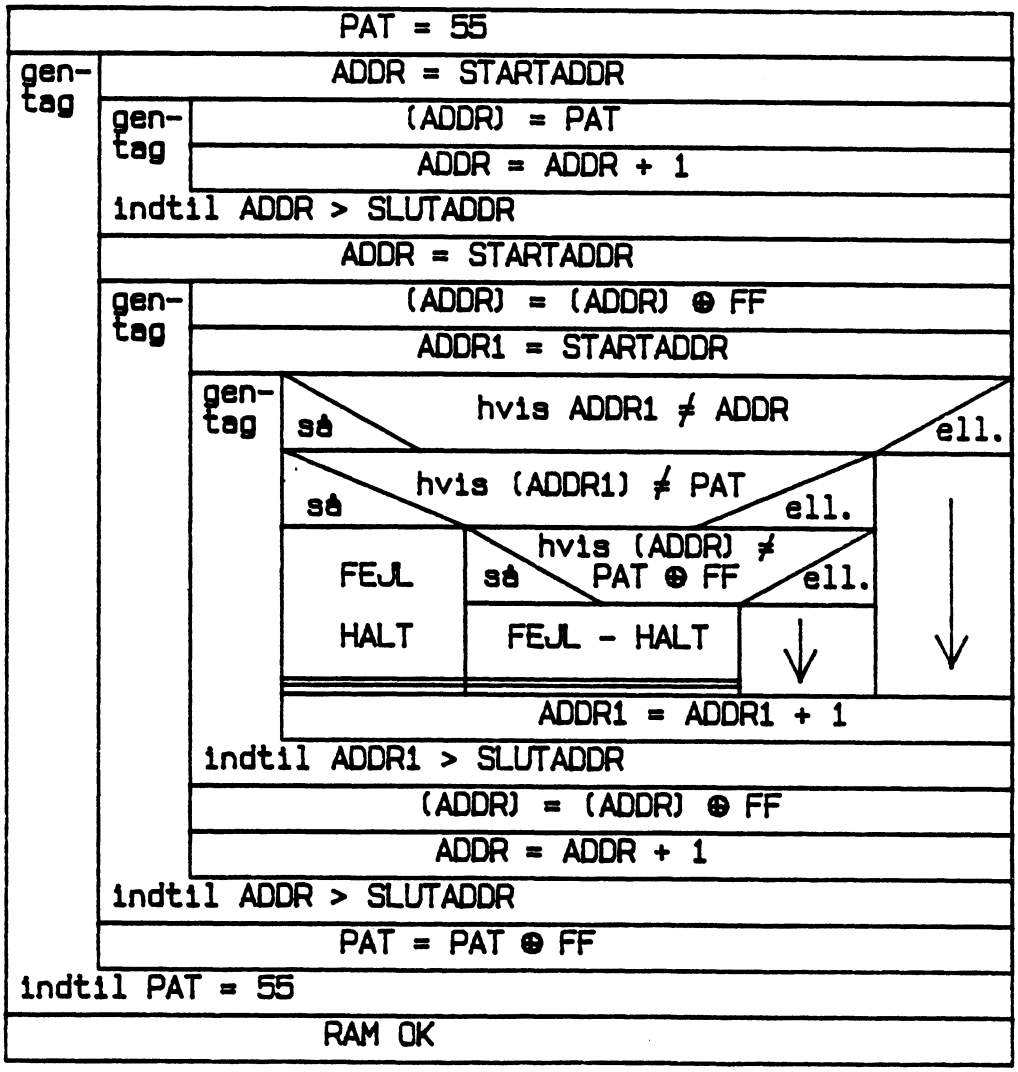
Fordele:

God til at undersøge for Read/Write Error, og Decod Error.
Rimelig god til at undersøge for Bitafsmitningsfejl.

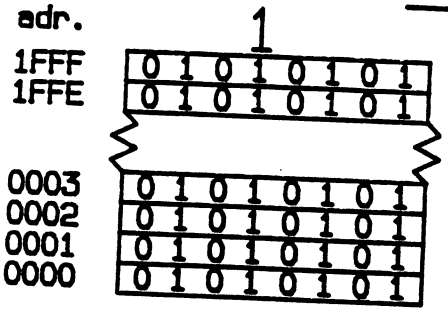
Ulemper:

Meget lang tidsforbrug (bruges bedst i mindre adresse-afsnit af gangen).
Destruerer det oprindelige indhold i ramlageret (testen er ikke transparent), testen kan laves i en variant der er transparant, da den arbejder på det oprindelige indhold af ramlageret, og efter testen afleverer det uændret.

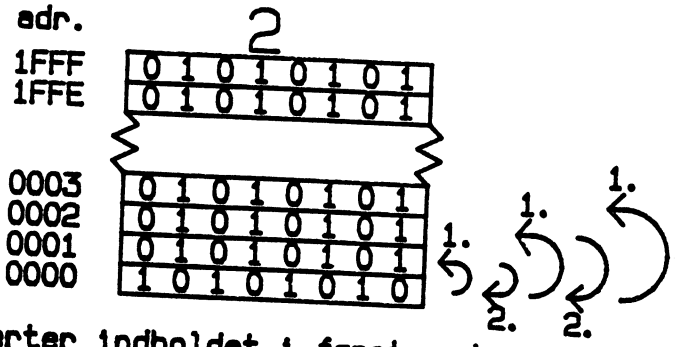
RAM-test: GALPAT



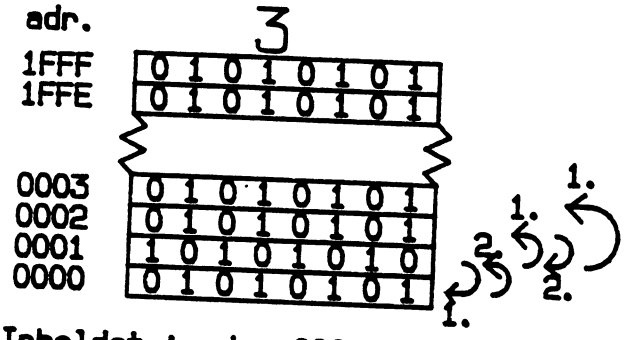
GALPAT.



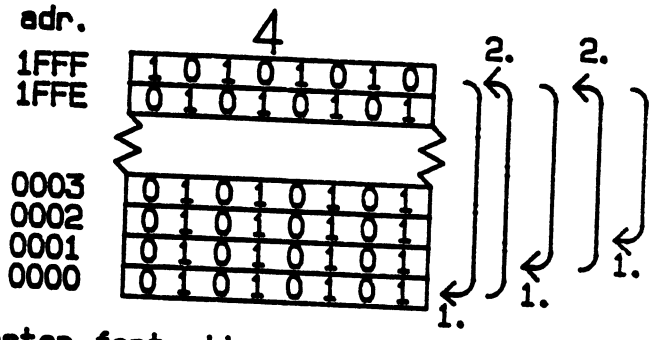
Adresseområdet fyldes med mønstret 55.



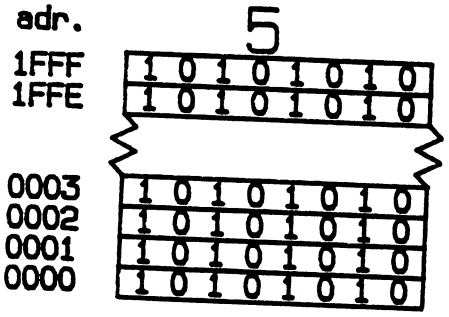
Inverter indholdet i første adr.
Gennemløb med stigende adr.:
1. check næste adr.
2. check de inveterede data



Inholdet i adr. 0001 inverteres og de øvrige adr. checkes som før.



Testen fortsætter indtil indholdet af alle adr. har været inverteret.



Adresseområdet fyldes nu med mønstret AA og hele testen gentages med dette mønster.

R/W og DECOD-test:

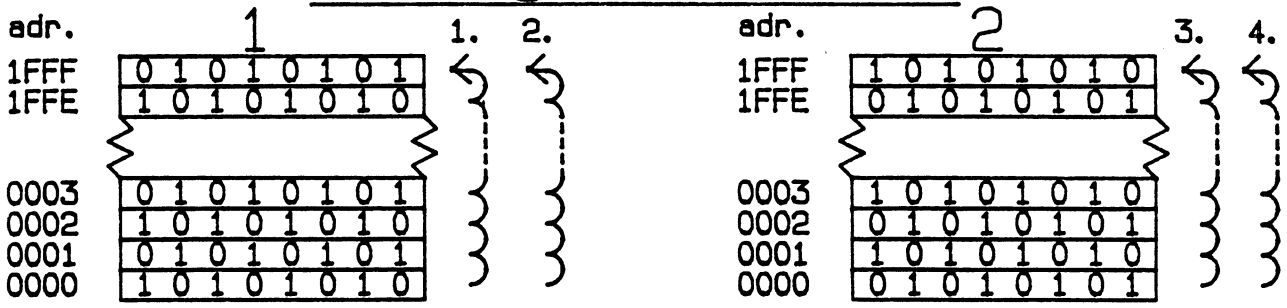
Fordele:

Testmønsteret er enkelt og overskueligt, hvorved testen bliver relativ nem at lave.
Rimelig kort tidsforbrug.
Ved analyse af testresultat, kan der drages slutninger om fejlens beliggenhed.

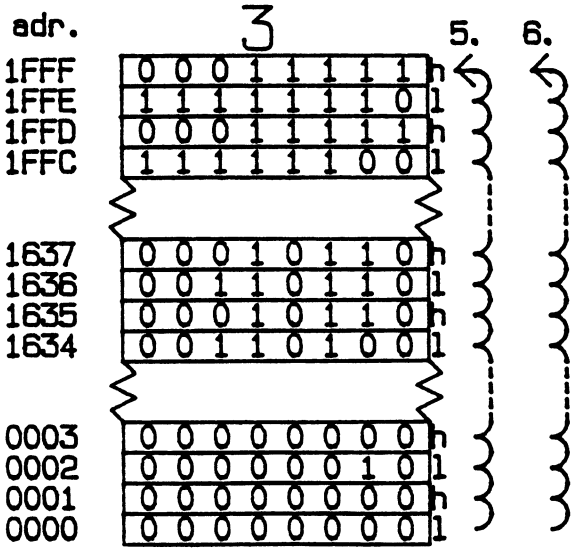
Ulemper:

Der testes ikke for bitafsmitning.
Destruerer det oprindelige indhold (ikke transparent).

R/W og DECOD-TEST



1. Adresseområdet fyldes med skakbræt-mønster, skiftevis AA og 55.
2. Adresseområdet læses og checkes for korrekt indhold.
3. Adresseområdet fyldes med det inverterede mønster, skiftevis 55 og AA.
4. Adresseområdet læses og checkes for korrekt indhold.

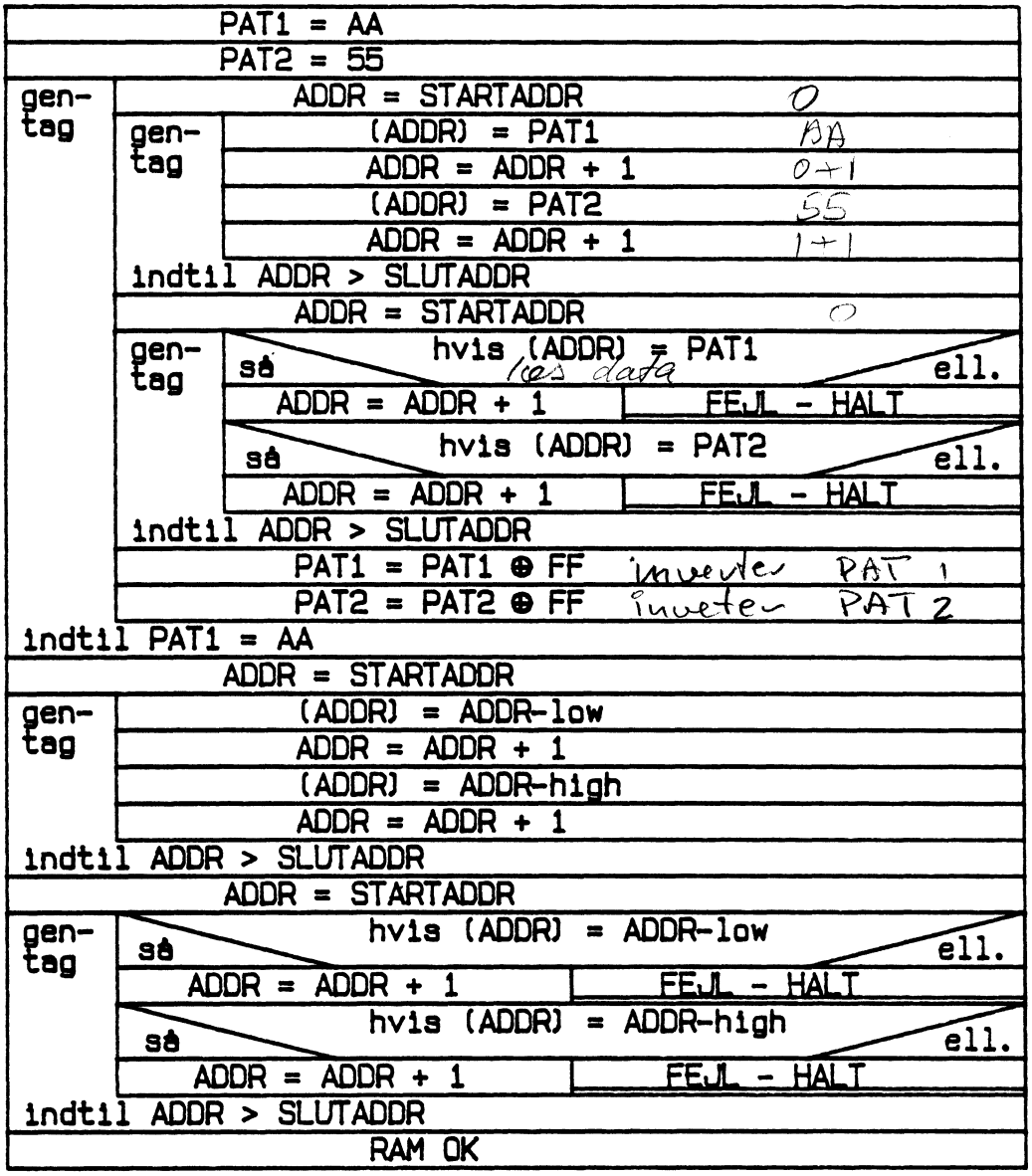


5. Adresseområdet fyldes med adr. afhængige data.
6. Adresseområdet læses og checkes for korrekt indhold.

h: viser fejl ved adr. bit 8-15

l: viser fejl ved adr. bit 0-7

RAM-test:


PAT1 = AA
PAT2 = 55
gen-tag
ADDR = STARTADDR
0
gen-tag
(ADDR) = PAT1
AA
ADDR = ADDR + 1
0+1
(ADDR) = PAT2
55
ADDR = ADDR + 1
1+1
indtil ADDR > SLUTADDR
ADDR = STARTADDR
0
gen-tag
hvis (ADDR) ≠ PAT1
så
les data
ell.
ADDR = ADDR + 1
FEJL - HALT
hvis (ADDR) = PAT2
så
ell.
ADDR = ADDR + 1
FEJL - HALT
indtil ADDR > SLUTADDR
PAT1 = PAT1 ⊕ FF
inverter PAT 1
PAT2 = PAT2 ⊕ FF
inverter PAT 2
indtil PAT1 = AA
ADDR = STARTADDR
gen-tag
(ADDR) = ADDR-low
ADDR = ADDR + 1
(ADDR) = ADDR-high
ADDR = ADDR + 1
indtil ADDR > SLUTADDR
ADDR = STARTADDR
gen-tag
hvis (ADDR) = ADDR-low
så
ell.
ADDR = ADDR + 1
FEJL - HALT
hvis (ADDR) = ADDR-high
så
ell.
ADDR = ADDR + 1
FEJL - HALT
indtil ADDR > SLUTADDR
RAM OK

ROM-TEST

4 principper til fejldetektering af ROM / EPROM:

1) Checksum: (aritmetisk checksum)

Aritmetisk addition af indholdet af samtlige adresser i ROM'en, hvor "carry" smides væk.

Adr.	ROM	addition af indholdet:
0000	! 43 !	43
0001	! 2B !	+2B = 6E
0002	! C3 !	+C3 = 31
0003	! 21 !	+21 = 51
0004	! 0A !	+0A = 5B
0005	! 62 !	+62 = 8D

checksum = 8D

---

2) Modificeret Checksum:

Som aritmetisk checksum, men efter hver addition roteres summen 1 bitposition mod venstre.

Adr.	ROM	addition af indholdet:
0000	! 43 !	43 <= 86
0001	! 2B !	+2B = B1 <= 63
0002	! C3 !	4C +C3 = 26 <= 4C
0003	! 21 !	+21 = 6D <= DA
0004	! 0A !	+0A = E4 <= C9
0005	! 62 !	+62 = 2B

2B <= 56

modificeret checksum = 56

---

3) Lodret paritet: (EX-OR CHECK)

Der laves lige (eller ulige) paritets-check mellem alle bit i samme bitposition i ROM'ens indhold.

Adr.	D7	D6	D5	D4	D3	D2	D1	D0
0000	0	1	0	0	0	0	1	1
0001	0	0	1	0	1	0	1	1
0002	1	1	0	0	0	0	1	1
0003	0	0	1	0	0	0	0	1
0004	0	0	0	0	1	0	1	0
0005	0	1	1	0	0	0	1	0
lodret paritet:	1	1	1	0	0	0	1	0

checksum = E2

Z80 program til de 3 første test principper:

```
          ld   hl,ROMSTART
          ld   de,ROMSLUT
          ld   b,0              ! checksum := 0
LOOP:     ld   a,b
-----------------------------------------
! Checksum:   !  Modificeret !  Lodret   !
!             !  Checksum:   !  Paritet: !
! add a,(hl)  !  add a,(hl)  !  xor (hl) !
!             !  rlc a       !           !
-----------------------------------------
          ld   b,a
          inc  hl               ! næste adresse
          ld   a,l
          cp   e                ! adr.low = slut?
          jr   nz,LOOP
          ld   a,h
          cp   d                ! adr.high = slut?
          jr   nz,LOOP
          ld   a,b              ! aflever checksum
          ld   (RESULT),a       ! i RESULT

          --------------------
```

4) Cyclic Redundancy Check (CRC) eller Signatur Check:

Signaturen dannes som en rest fra en division af ROM´ens data med et polynomium.

Princip for "CRC" vist med eksempel der giver en 4 bit signatur:

Data: 1011110010
Polinomium: 11001 = x4 + x3 + 1

```
 1011110010 : 11001 = 110111
-11001
 11101
-11001
   10000
  -11001
    10011
   -11001
     10100
    -11001
      1101
```

Rest: 111000101

4 bit signatur: 0101

Normalt bruges en 8 eller 16 bit signatur for at øge fejl-detekterings sikkerheden.

Valget af polynomiet har betydning for fejl-detekterings sikkerheden, og afgør tillige om signaturen bliver på 8 eller 16 bit.

Eksempel på anvendte polynomier:

8 bit sign:

x8 + x4 + x3 + 1 (1)00011001

16 bit sign:

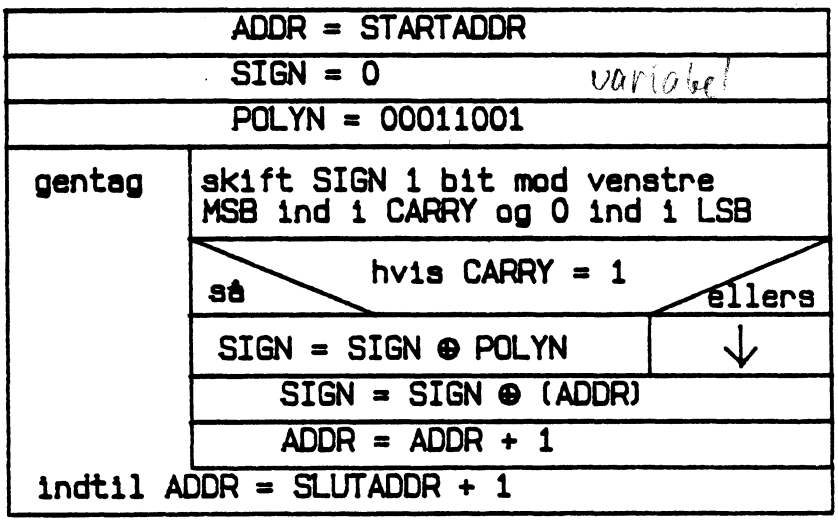
CRC-16	=	x16 + x15 + x2 + 1	(1)1000000000000101
SDLC (CCITT-16)	=	x16 + x12 + x5 + 1	(1)0001000000100001
Signatur-meter	=	x16 + x9 + x7 + x4 + 1	

Signaturen kan laves både som en hardware-løsning (f.eks. signatur meter) og en software-løsning.

Struktugram over program til Cyclic Redundancy Check:

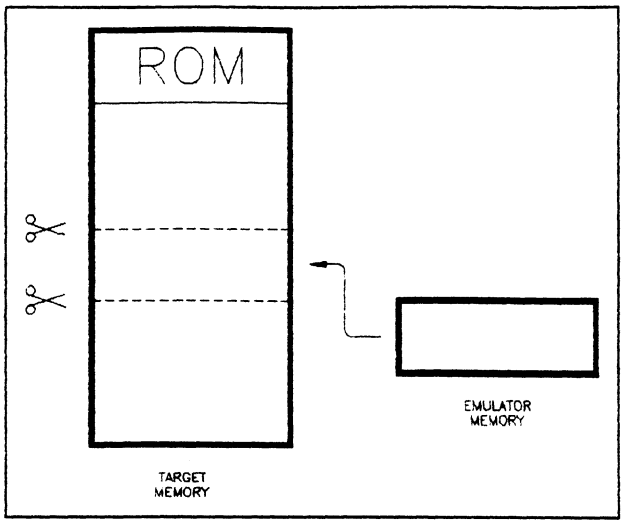
POLYN = 00011001
(polynomium til 8 bit signatur: x8 + x4 + x3 + 1)

Cyclic Redundancy Check



⊕ = bit for bit EXCL-OR

Testprogrammer i emulator



Man kan ved hjælp af en emulator indlægge en shadow memory et vilkårligt sted i memorymappen.

Dette kan være velegnet til, at udføre testprogrammer som tester ROM'er i et givet target system, hvor man ikke kan fjerne disse mens testen foregår.

Metoden man anvender består i, at "klippe" et passende stykke af den eksisterende memorymap ud, og i stedet her anbringe emulatorens shadow memory.

Dette foretages i PMDS-II anlægget ved, at man tilslutter debuggerens POD i CPU-soklen i stedet for selve CPU'en, op opsætter debuggeren til at emulere et stykke af memory mappen.

Dette foretages ved i debuggerens opstartsfil, at ændre 'map' kommandoerne til følgende linier:

```
MAP F800 FFFF PROTO ROM
MAP 0000 0FFF EMEM ROM
```

Derved instrueres emulatoren om, at der befinder sig en ROM i target systemet på adresse F800-FFFF.

Selve emulatorens shadow memory placeres i memory mappen i adresseområdet 0000-0FFF.

Selve testprogrammet skal derfor blot nu assembleres til at blive udført fra adresse 0000:

```
            .sect     .text
            .base     0x0000

start:      ......
```

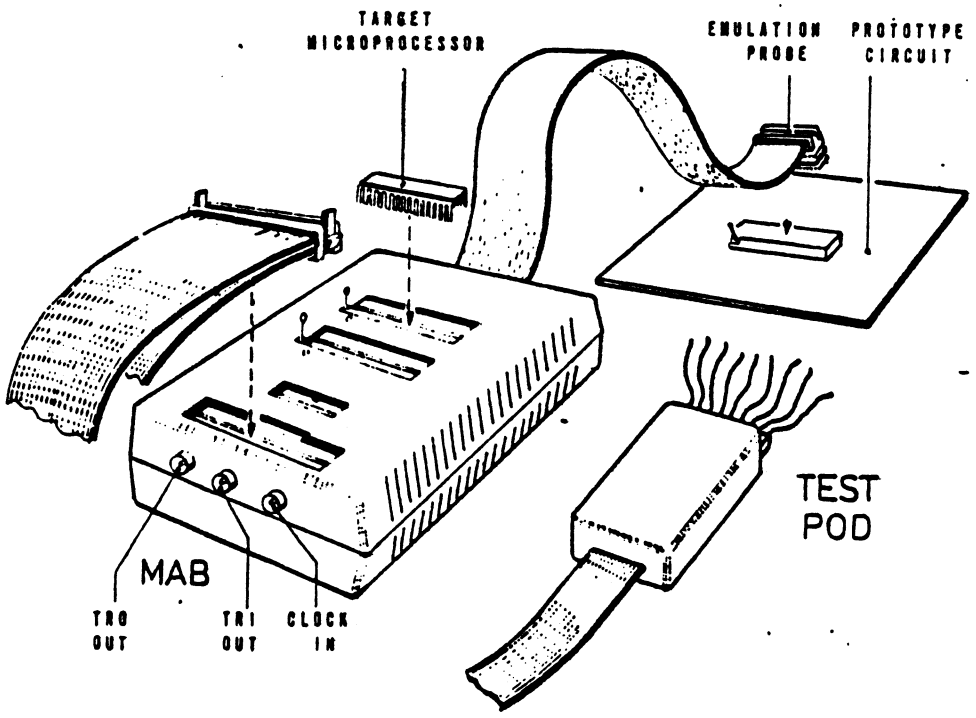
Selve test programmet, som nu er placeret i emulatoren og udføres i det aktuelle target system, kan f.eks. give sine fejl meddelser i et af CPU'ens registre, hvor man via debuggeren kan se indholdet af disse.

For at indikere over for debuggeren at testen er slut kan man afslutte sit test program med en 'halt' instruktion. Dette bevirker nemlig at debuggeren giver kontrollen tilbage til brugeren.

Når testprogrammet skal udføres skal man blot starte debuggeren op, og load'e testprogrammet for derefter at skrive:

```
RUN 0000
```

THE MICROPROCESSOR ADAPTER BOX AND TEST POD



PHILIPS

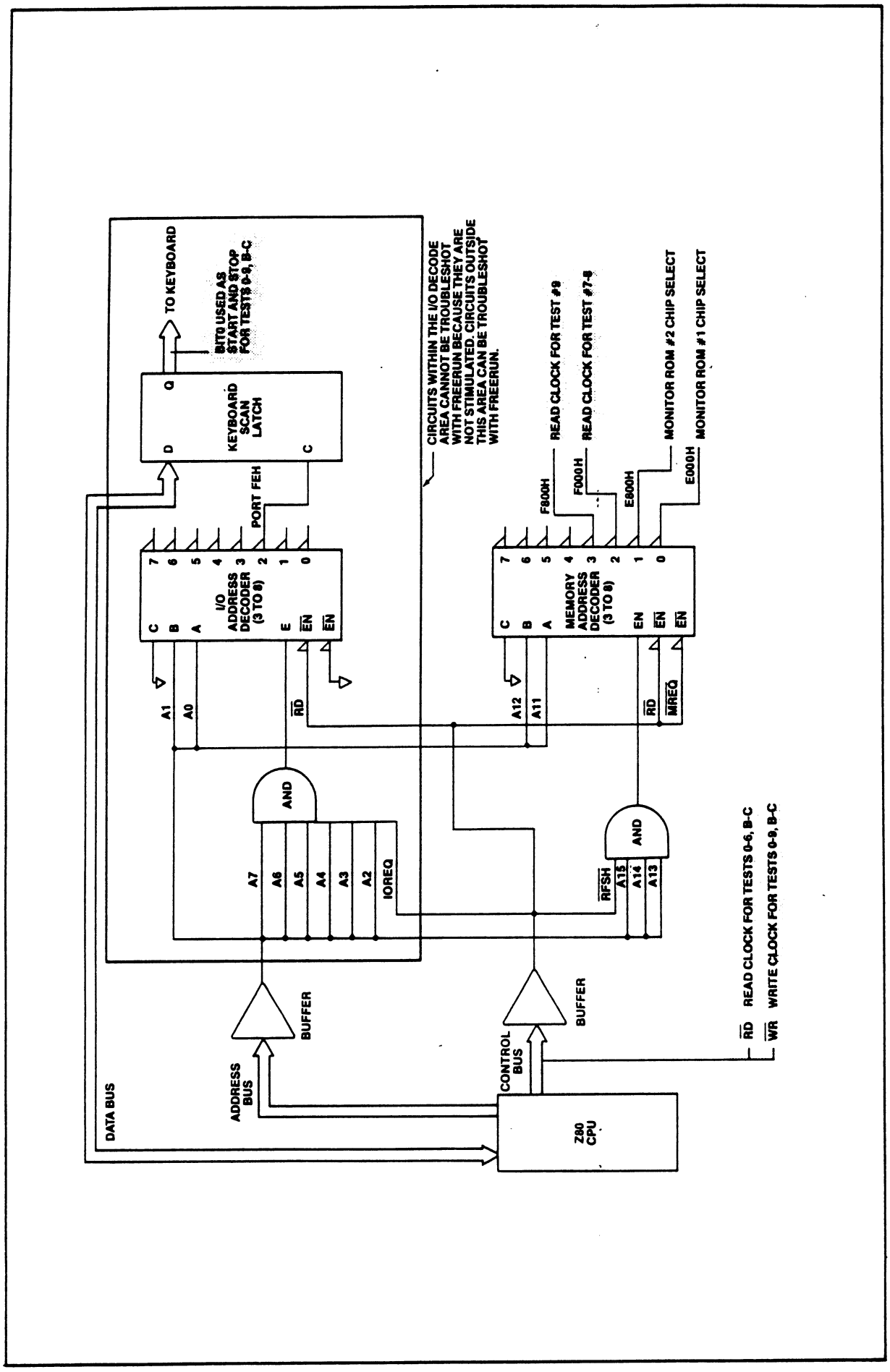


Figure 9. For all SA tests (except #A), START and STOP are both connected to bit 0 in the KEYBOARD SCAN LATCH addressed as I/O port FEH. The programs set and reset the bit to create edges that can be detected by the signature analyzer's START and STOP inputs. START is selected to trigger on a rising edge and STOP triggers on a falling edge. Setting the bit at the beginning of the test opens the GATE while resetting the bit immediately after the last test step closes the GATE. Figures 10 and 12 show the Z80 code that creates the START and STOP edges. The CLOCK connection depends on which test is run as discussed in figures 10-12. The START, STOP and CLOCK connections for test #A are shown in the separate description of that test.

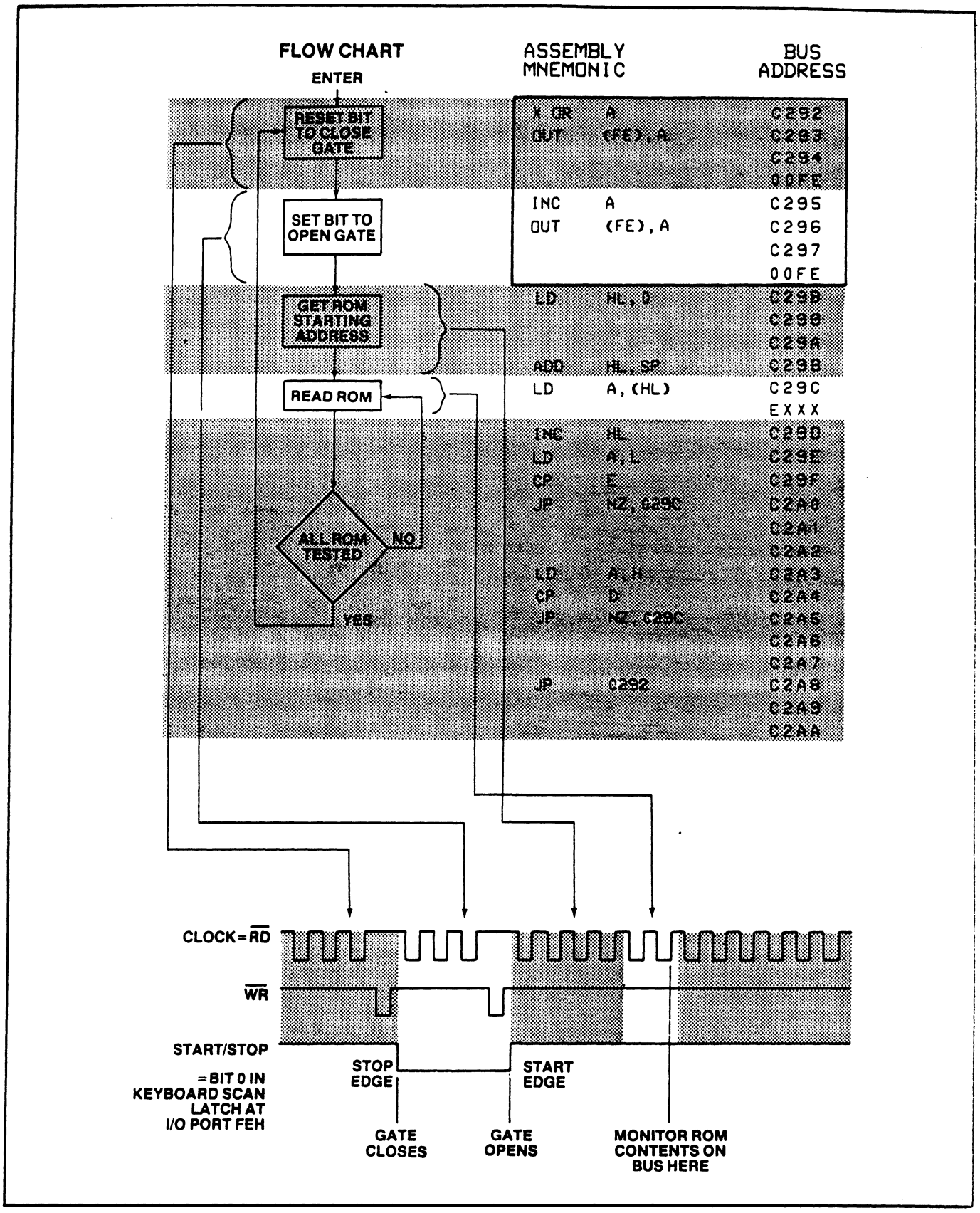


Figure 10. Here is the Z80 code that creates the START and STOP edges by setting and resetting bit 0 in the KEYBOARD SCAN LATCH at I/O port FEH. For tests 0-6 and B-C, two CLOCKS ($\overline{RD}$ and $\overline{WR}$) are used to detect the START and STOP edges. A $\overline{RD}$ CLOCK occurs with every op-code fetch of the Z80 which is more than adequate to detect the START and STOP edges. (A clock edge must occur both before and after both the START and STOP edges to ensure detection.) A $\overline{WR}$ CLOCK occurs when the I/O port FEH and the device being tested are written. The code shown is for MONITOR ROM tests 0-1. A $\overline{WR}$ CLOCK does not occur for these tests. Similar code applies to tests 0-6 and B-C.

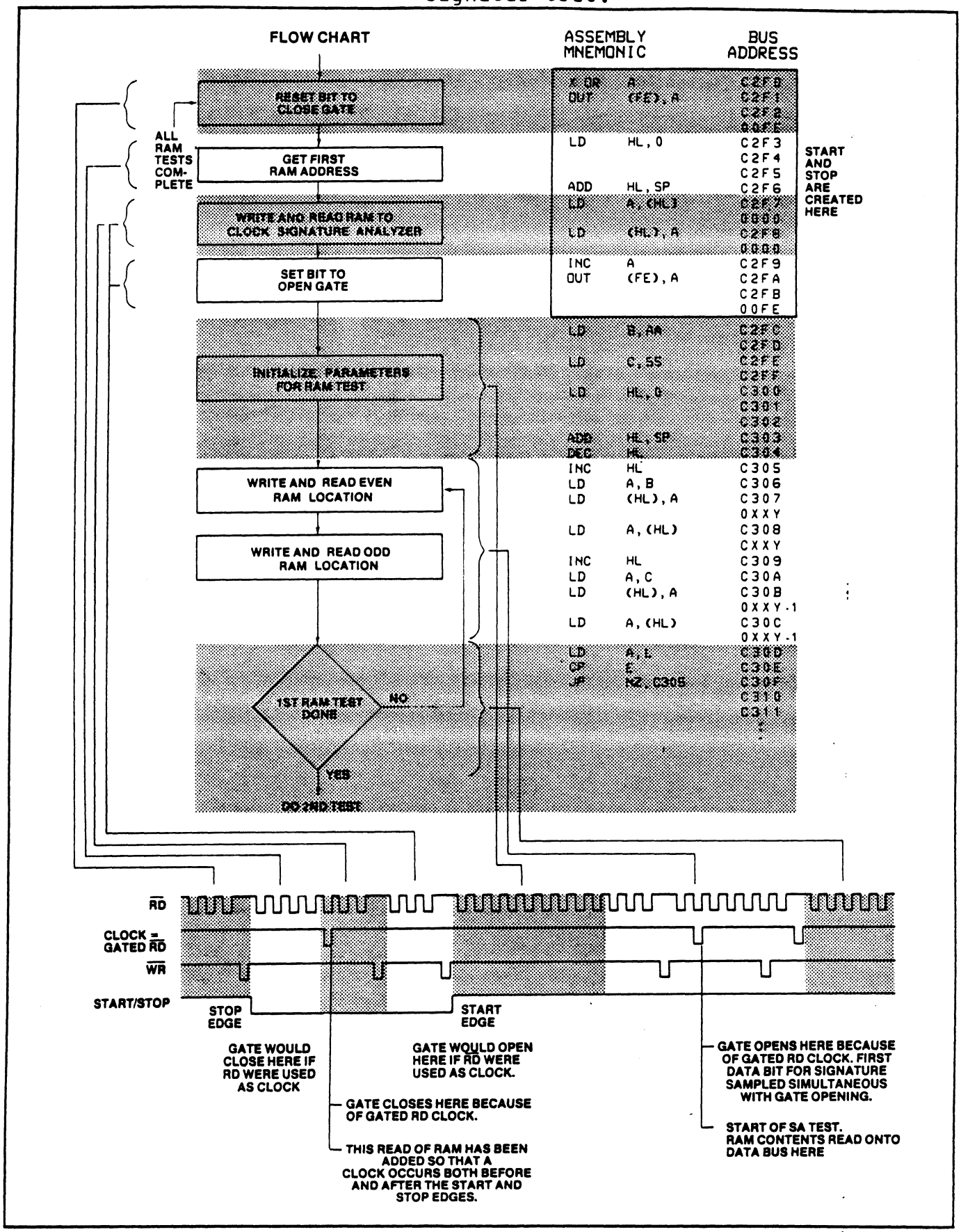


Figure 12. Generation of the START and STOP edges for tests 7-9 is similar to tests 0-6 and B-C of figure 10. In this case, extra code has been added between the generation of START and STOP because of the gated CLOCK. The gated $\overline{RD}$ CLOCK used in these tests occurs only when the device being tested is read. The extra code adds both a read and a write access to the device under test between the STOP and START edges to ensure detection.

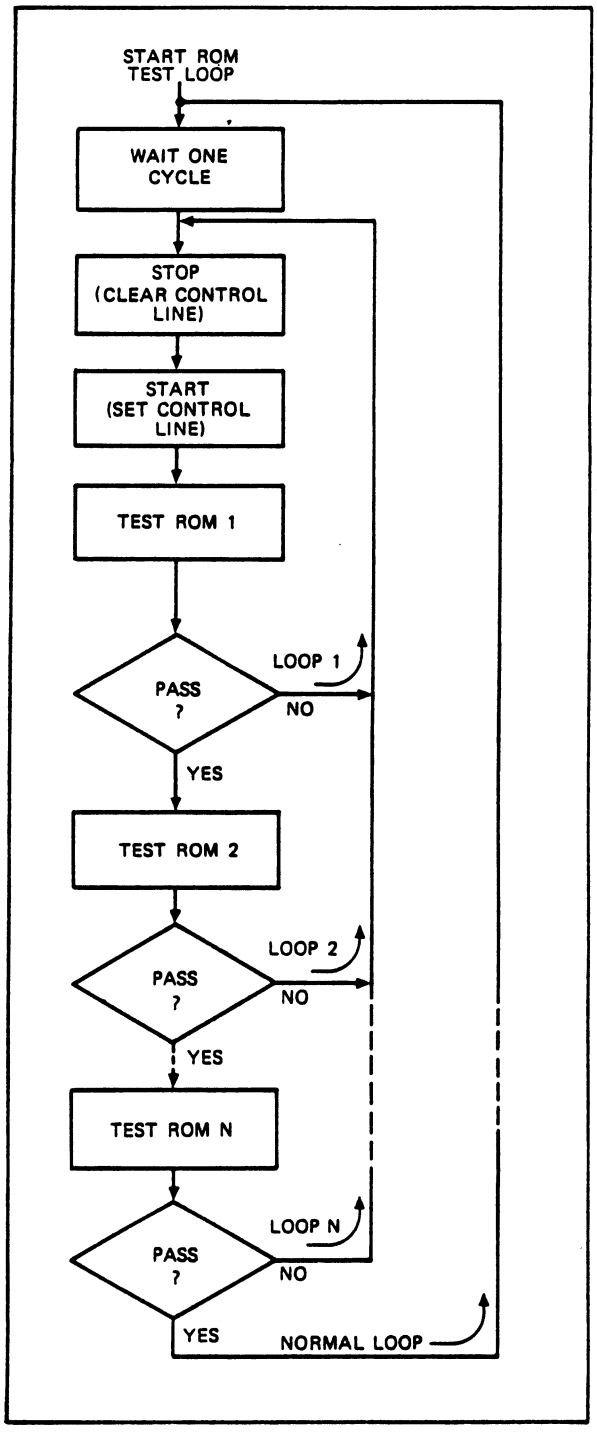


Figure 7. Loop on Fail Flowchart. SA test cycle length varies with first fault location to produce a characteristic Vcc signature.

Table 1

Motorola 6800 Code for performing a checksum on two ROMs ("ROM1" and "ROM2"). Address line A15 is toggled at the start. A variable length time window is created by again toggling A15 when an incorrect checksum is found, or when the test is completed. The V_{cc} signature will take on one of only three possible values depending on whether 1) ROM1 is bad, 2) ROM2 is bad, or 3) both ROMs are good.

MEM. ADDRESS LOCATION	LABEL	MNEMONIC	LOCATION OR #	EXPLANATION
3800	CSR1	FCB	$D2	Check sum of ROM1
3801	CSR2	FCB	$8C	Check sum of ROM2
	STSP	EQU	$8000	Label used to toggle A15 for SA start and stop
	BESA	STA A	STSP	
		LDX	#$3801	Load index register with the starting address of ROM1
		CLR A		Clear accumulator for checksum
	ROM1	ADD A	0,X	Add contents of ROM to Accumulator A.
		INX		Increment index register to check next location
		CPX	#$4000	Compare the index register with the end of ROM1.
		BNE	ROM1	Branch to "ROM1" if all ROM1 locations have not been checked
		STA A	$00	Output checksum onto data bus
		CMP A	CSR1	Compare accumulator A with ROM1 checksum
		BNE	BESA	Branch to "BESA" to end time window if not equal
	ROM2	LDX	#$4000	Same as above except ROM1's are changed to ROM2's
		CLR A		
		ADD A	0,X	
		INX		
		CPX	#$4200	
		BNE	ROM2	
		STA A	$00	
		CMP A	CSR2	
		BNE	BESA	
		NOP		Extend time window one step
		BRA	BESA	If both checksums are correct then end time window

SECTION F—STATIC VIDEO PATTERN, TEST A

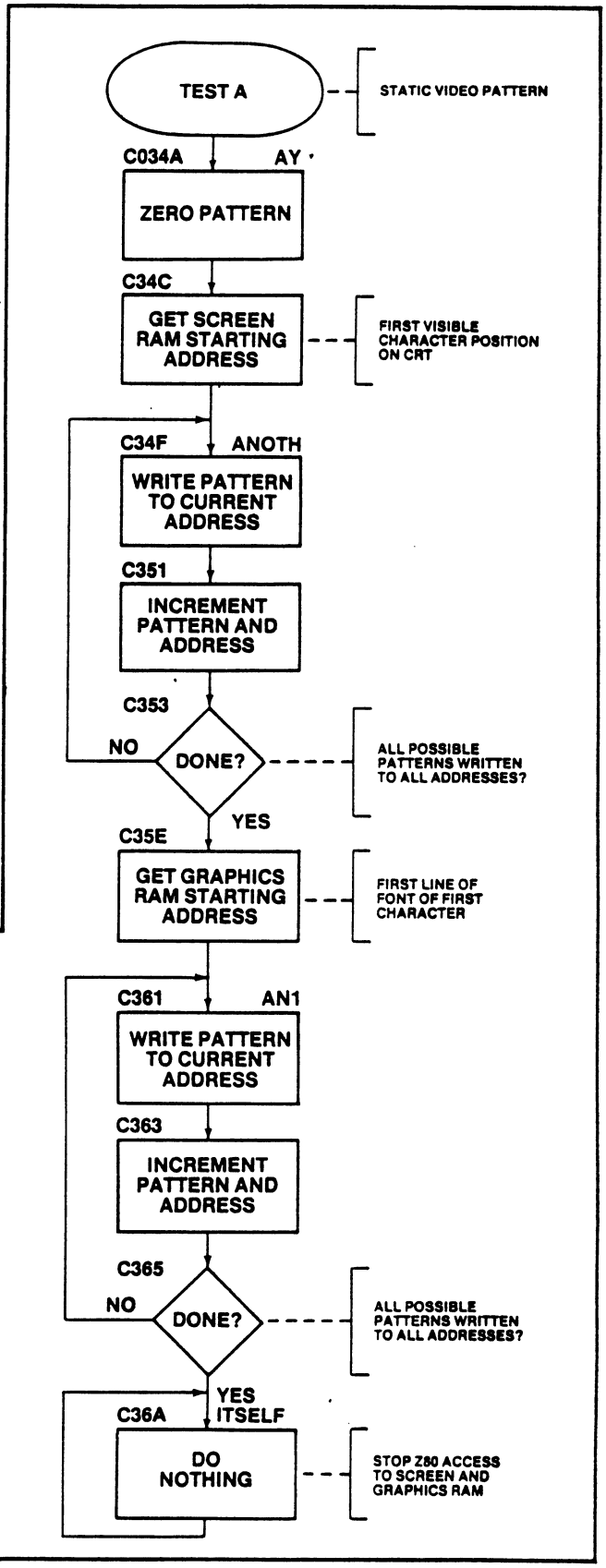


Figure 14. The static video pattern is written into RAM by this program. All possible characters and all possible patterns of user-defined font are written into SCREEN RAM and GRAPHICS RAM respectively. The program then enters a small loop that keeps the Z80 from further interaction with either RAM. This keeps the node activity limited to the CRT refresh process so that signatures are stable.

HEX ADRS	CONTENTS	LABEL	INSTRUCTION	
C34A		AY		
C34A	0600		LD	B, 0
C34C	2180F0		LD	HL, 0F080H
C34F		ANOTH:		
C34F	78		LD	A, B
C350	77		LD	(HL), A
C351	23		INC	HL
C352	04		INC	B
C353	7D		LD	A, L
C354	B7		OR	A
C355	C24FC3		JP	NZ, ANOTH
C358	7C		LD	A, H
C359	FEF8		CP	0F8H
C35B	C24FC3		JP	NZ, ANOTH
C35E	21000FC		LD	HL, 0FC00H
C361		AN1:		
C361	78		LD	A, B
C362	77		LD	(HL), A
C363	23		INC	HL
C364	04		INC	B
C365	7D		LD	A, L
C366	B4		OR	H
C367	C261C3		JP	NZ, AN1
C36A	C36AC3	ITSELF	JP	ITSELF

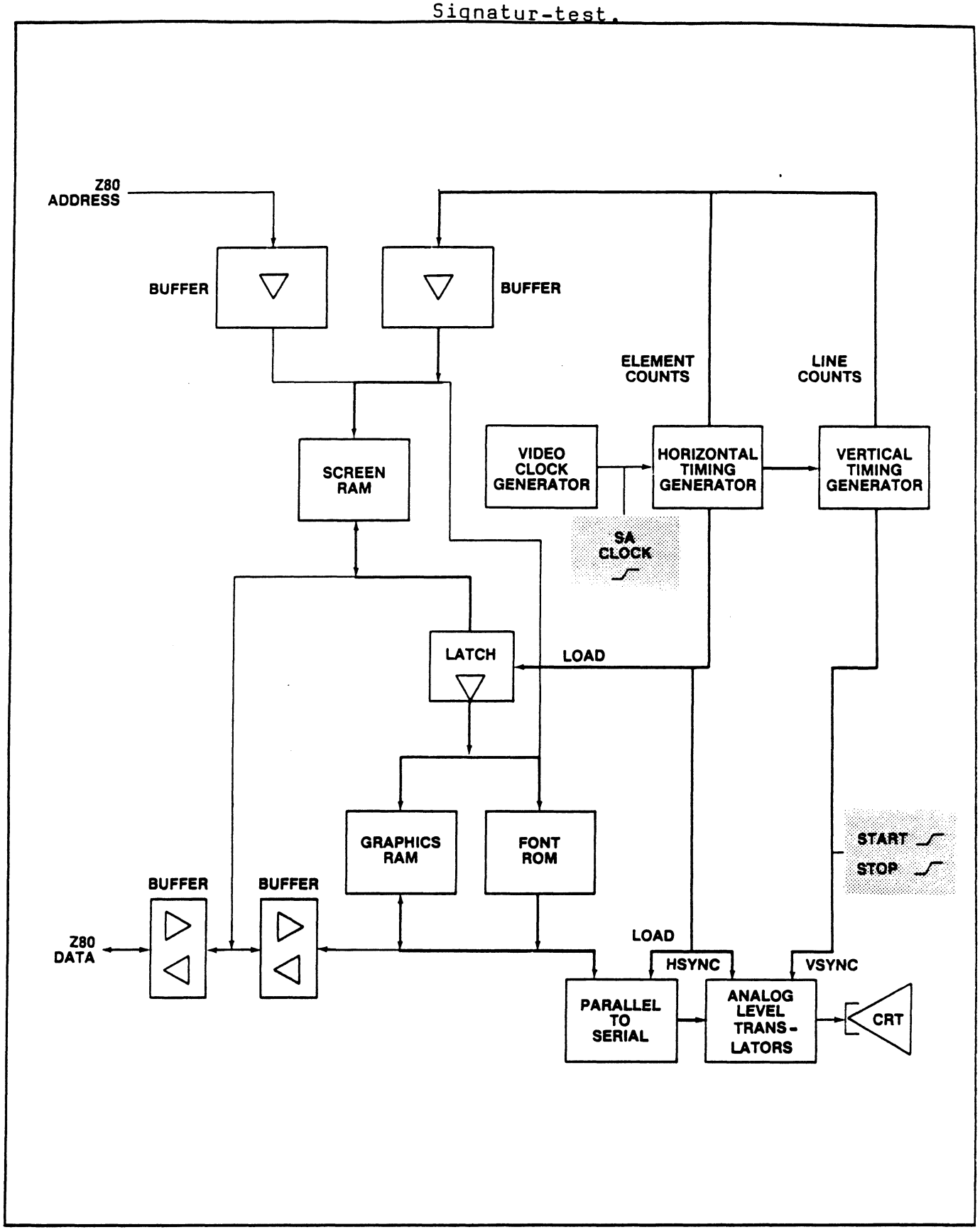


Figure 15. These nodes are stimulated by the CRT refresh process while the static video pattern is displayed on the CRT. These START, STOP and CLOCK connections form a GATE that is open for one complete refresh cycle of the CRT, allowing the signature analyzer to detect errors in any of the circuits including all the font patterns in ASCII FONT ROM. When the TIMING GENERATORS fail, START and STOP are no longer generated. In that case, START and STOP are moved to the connections shown in the next figures, closer to the kernel circuits of the VIDEO TEXT GENERATOR.

SECTION G—PARALLEL PORT, TEST B

External hardware was required to stimulate the INPUT PORT. Wires on an external connector loop the patterns that are written to the OUTPUT PORT back to the INPUT PORT. Timing requirements of the HANDSHAKE control lines made it impossible to simply loop their outputs back to the inputs with similar wires on the connectors. It was decided not to add the ICs that would be required to fully stimulate the HANDSHAKE circuits. Two things determined this. First, the extra hardware would not be available to the distributors or field service personnel. Second, the HANDSHAKE circuits consist of one IC that can be easily troubleshot with other means such as logic probes.

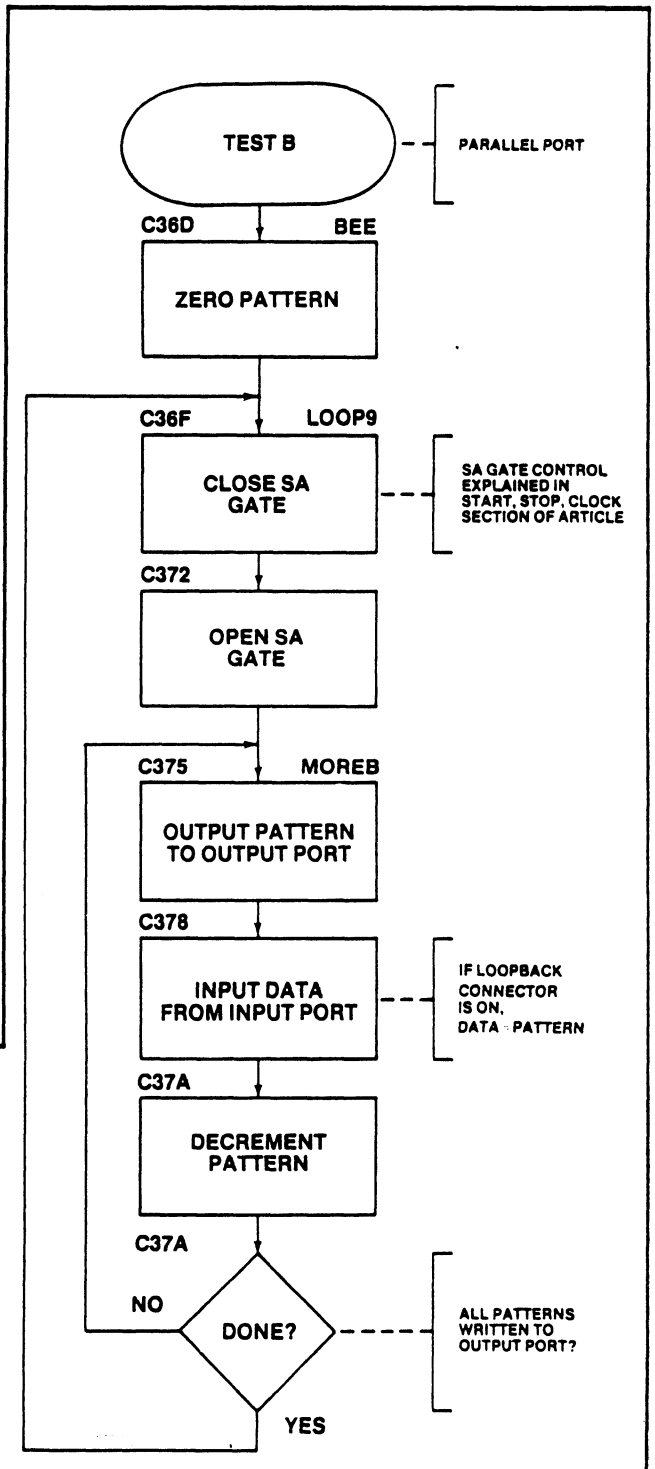


Figure 18. This program stimulates both the OUTPUT and INPUT PARALLEL PORTS by continuously writing all possible patterns to the OUTPUT PORT. The program also reads the INPUT PORT whether it is stimulated or not. The INPUT PORT is stimulated by looping the OUTPUT PORT back to the INPUT PORT using the connector shown in the following figure.

```
HEX
ADRS   CONTENTS   LABEL    INSTRUCTION

C36D              BEE:
C36D   0600                LD    B,0
C36F              LOOP9:
C36F   AF                  XOR   A
C370   D3FE                OUT   (0FEH),A
C372   3C                  INC   A
C373   D3FE                OUT   (0FEH),A
C375              MOREB:
C375   78                  LD    A,B
C376   D3FF                OUT   (0FFH),A
C378   DBFF                IN    A,(0FFH)
C37A   10F9                DJNZ  MOREB-$
C37C   D36FC3              JP    LOOP9
```

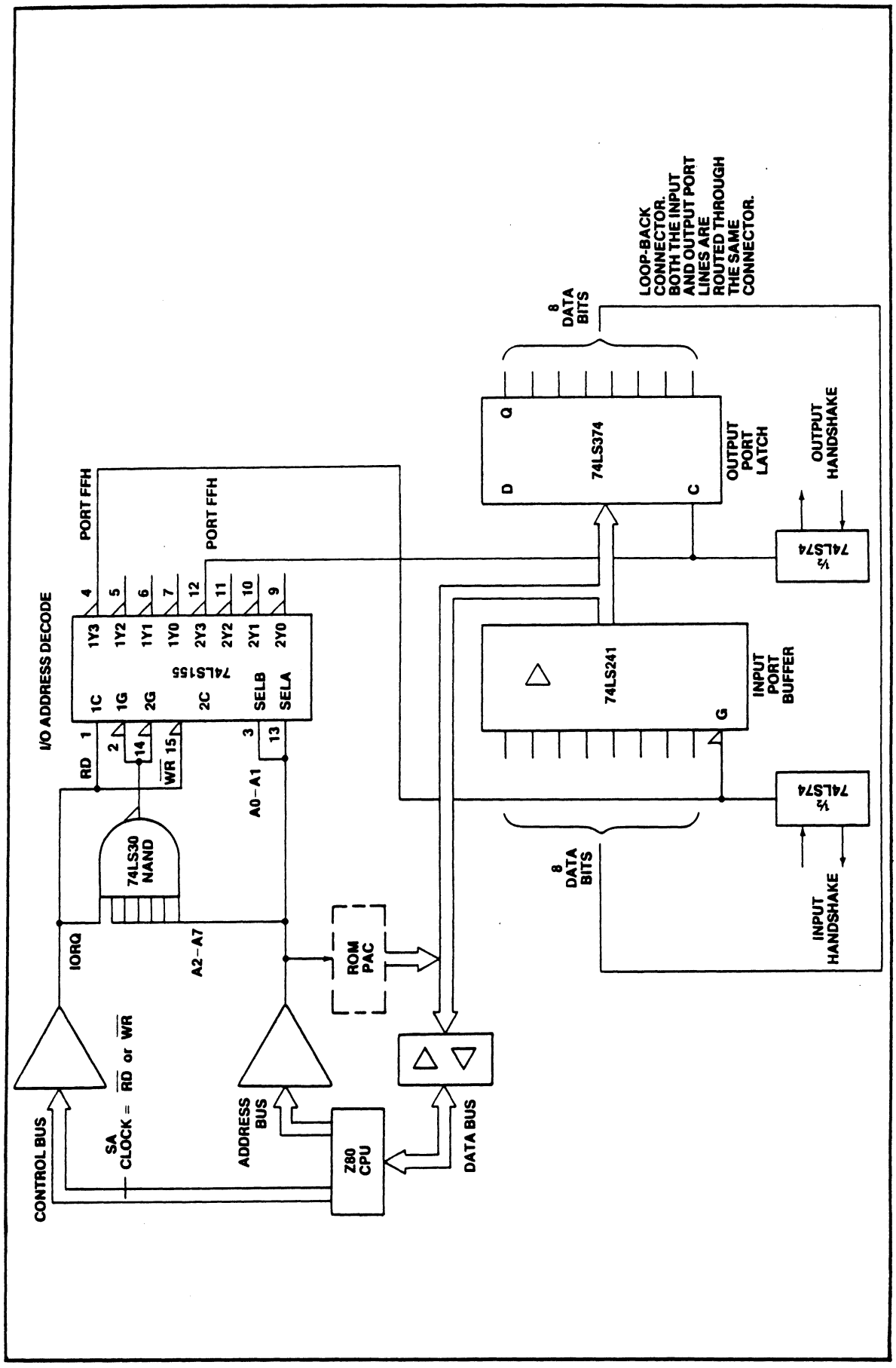


Figure 19. These circuits are stimulated by the PARALLEL PORT test. START and STOP are connected to a bit in the KEYBOARD SCAN LATCH and are controlled by the program as shown in the section called THE HARDWARE AND SOFTWARE BEHIND START, STOP AND CLOCK. With the loop-back connector on, signatures are taken on the data bus using $\overline{RD}$ as a CLOCK. If signatures are correct, then both the OUTPUT and INPUT PORT are operating correctly. If signatures are incorrect, the connector is removed and signatures are taken on the OUTPUT PORT using $\overline{WR}$ as the CLOCK. Repairs are made as required. The loopback connector is then replaced to check the INPUT PORT for problems associated with reading it. The text explains why the HANDSHAKE circuitry is not stimulated by this test.

SECTION H—SERIAL RS-232 PORT, TEST C

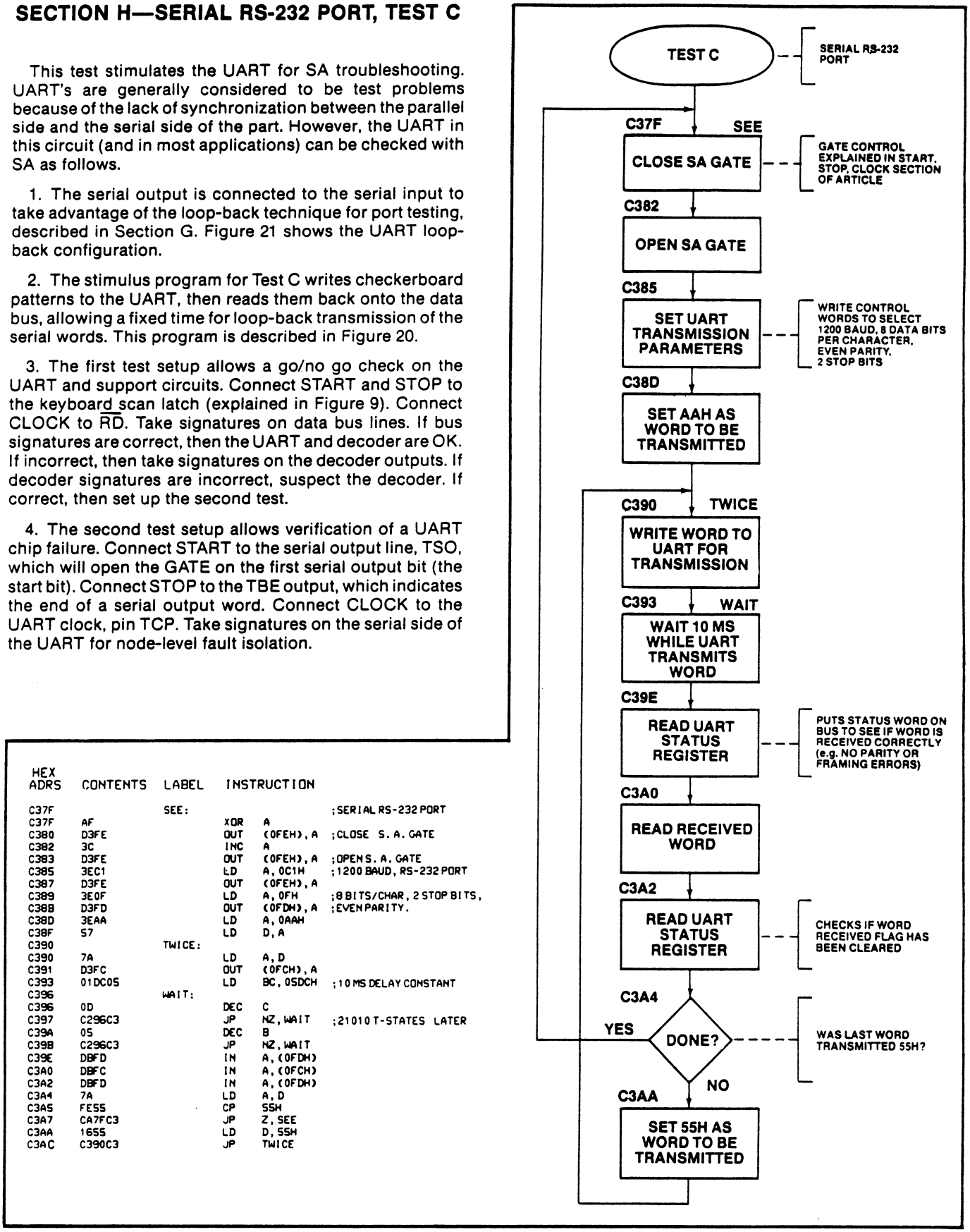
This test stimulates the UART for SA troubleshooting. UART's are generally considered to be test problems because of the lack of synchronization between the parallel side and the serial side of the part. However, the UART in this circuit (and in most applications) can be checked with SA as follows.

1. The serial output is connected to the serial input to take advantage of the loop-back technique for port testing, described in Section G. Figure 21 shows the UART loop-back configuration.

2. The stimulus program for Test C writes checkerboard patterns to the UART, then reads them back onto the data bus, allowing a fixed time for loop-back transmission of the serial words. This program is described in Figure 20.

3. The first test setup allows a go/no go check on the UART and support circuits. Connect START and STOP to the keyboard scan latch (explained in Figure 9). Connect CLOCK to $\overline{RD}$. Take signatures on data bus lines. If bus signatures are correct, then the UART and decoder are OK. If incorrect, then take signatures on the decoder outputs. If decoder signatures are incorrect, suspect the decoder. If correct, then set up the second test.

4. The second test setup allows verification of a UART chip failure. Connect START to the serial output line, TSO, which will open the GATE on the first serial output bit (the start bit). Connect STOP to the TBE output, which indicates the end of a serial output word. Connect CLOCK to the UART clock, pin TCP. Take signatures on the serial side of the UART for node-level fault isolation.




```
HEX
ADRS   CONTENTS   LABEL   INSTRUCTION

C37F              SEE:                        ;SERIAL RS-232 PORT
C37F   AF                 XOR   A
C380   D3FE               OUT   (0FEH),A      ;CLOSE S.A. GATE
C382   3C                 INC   A
C383   D3FE               OUT   (0FEH),A      ;OPEN S.A. GATE
C385   3EC1               LD    A,0C1H        ;1200 BAUD, RS-232 PORT
C387   D3FE               OUT   (0FEH),A
C389   3E0F               LD    A,0FH         ;8 BITS/CHAR, 2 STOP BITS,
C38B   D3FD               OUT   (0FDH),A      ;EVEN PARITY.
C38D   3EAA               LD    A,0AAH
C38F   57                 LD    D,A
C390              TWICE:
C390   7A                 LD    A,D
C391   D3FC               OUT   (0FCH),A
C393   01DC05             LD    BC,05DCH      ;10 MS DELAY CONSTANT
C396              WAIT:
C396   0D                 DEC   C
C397   C296C3             JP    NZ,WAIT       ;21010 T-STATES LATER
C39A   05                 DEC   B
C39B   C296C3             JP    NZ,WAIT
C39E   DBFD               IN    A,(0FDH)
C3A0   DBFC               IN    A,(0FCH)
C3A2   DBFD               IN    A,(0FDH)
C3A4   7A                 LD    A,D
C3A5   FE55               CP    55H
C3A7   CA7FC3             JP    Z,SEE
C3AA   1655               LD    D,55H
C3AC   C390C3             JP    TWICE
```

Figure 20. The serial inputs and outputs of a UART are stimulated by this program along with the control and status registers. The UART is first set to send and receive serial words with eight data bits, even parity and two stop bits at 1200 baud. Next, the program writes the word AAH into the UART and waits for its transmission to complete. Then the status word is read onto the data bus along with the serial word received, and then the status word again. Finally 55H is loaded for transmission similar to the word AAH. The program jumps back to the beginning of the loop upon completion.

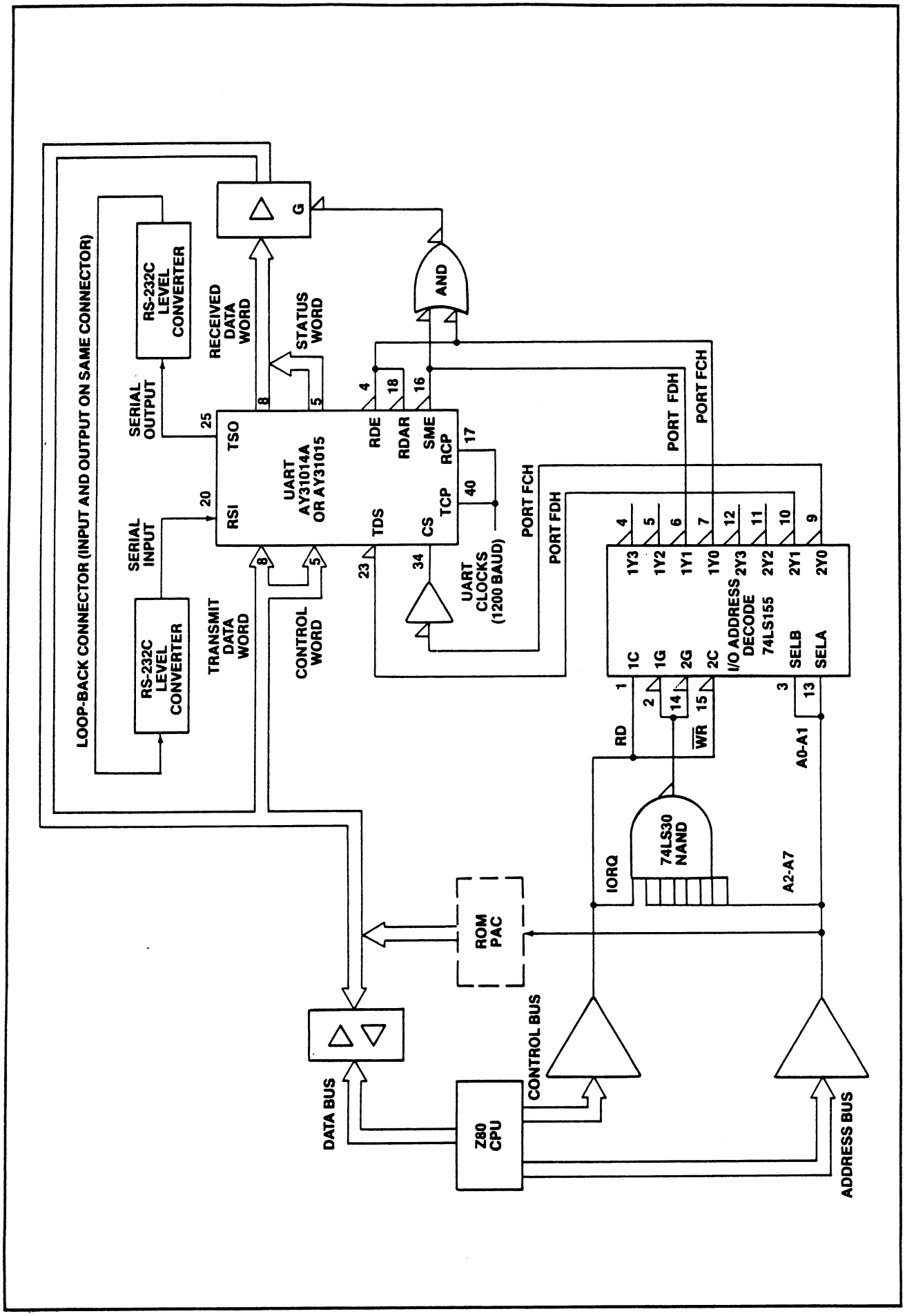


Figure 21. In Test C, the UART serial output is looped back into the serial input, as shown in this schematic. The stimulus exercises the entire loop. START, STOP and CLOCK connections for troubleshooting both the parallel and serial sides of the UART are outlined in the text.

SECTION I—A DESIGNER'S CHECKLIST FOR GETTING STARTED WITH SA

Here's a summary of this article that can be used as a checklist when designing or retrofitting SA into a microprocessor based system. It is not limited to Z80-based systems or personal computers.

1. Provide a means to FREERUN the microprocessor by using a FREERUN fixture or by designing in a way to open the data bus and force the NOP or FREERUN instruction into the processor. Find START, STOP and CLOCK connections on the processor.
2. Provide a means to store, access, and select the SA stimulus programs. Try to find a way that depends only upon circuits that can be troubleshot with FREERUN or logic probes in a simple manner.
3. Create START and STOP using software to control hardware that's already available or hardware that is specially designed into the product for SA testing. Use circuits that can be checked by FREERUN, a previous SA test, or other easy means such as logic probes.
4. Choose a CLOCK that is synchronous to START, STOP and DATA on all nodes being tested. Be sure there's a CLOCK edge both before and after the START and STOP edges. Avoid CLOCKing DATA from a node when it's in the 3rd state.
5. Create software test loops that can give both a go/no-go indication of all bused devices (like a diagnostic) and also allow fault isolation of a bad component or process fault even with bused devices. The tests can be separate.
6. Be sure your test can isolate a failure because of either a read or write problem with the device (e.g. RAM).
7. Provide a way to stimulate uncontrolled inputs of I/O devices in a synchronous fashion using loop-back connectors or external stimuli.
8. Provide means to open feedback loops. Usually only a concern in circuits that are independent of the processor.
9. One-shots and UART serial outputs generally cannot be tested with SA, so find a way to bypass them (eliminate their effect on other circuit elements) during the SA test so that all nodes operate synchronous to the CLOCK.
10. Be sure your tests don't depend upon circuits working that are being tested. Usually done by running the SA tests open-loop (i.e. the tests only stimulate the devices but don't check the results to see if it was accomplished. The signature analyzer will check the results.). Sometimes can happen inadvertently when controlling START and STOP with software.

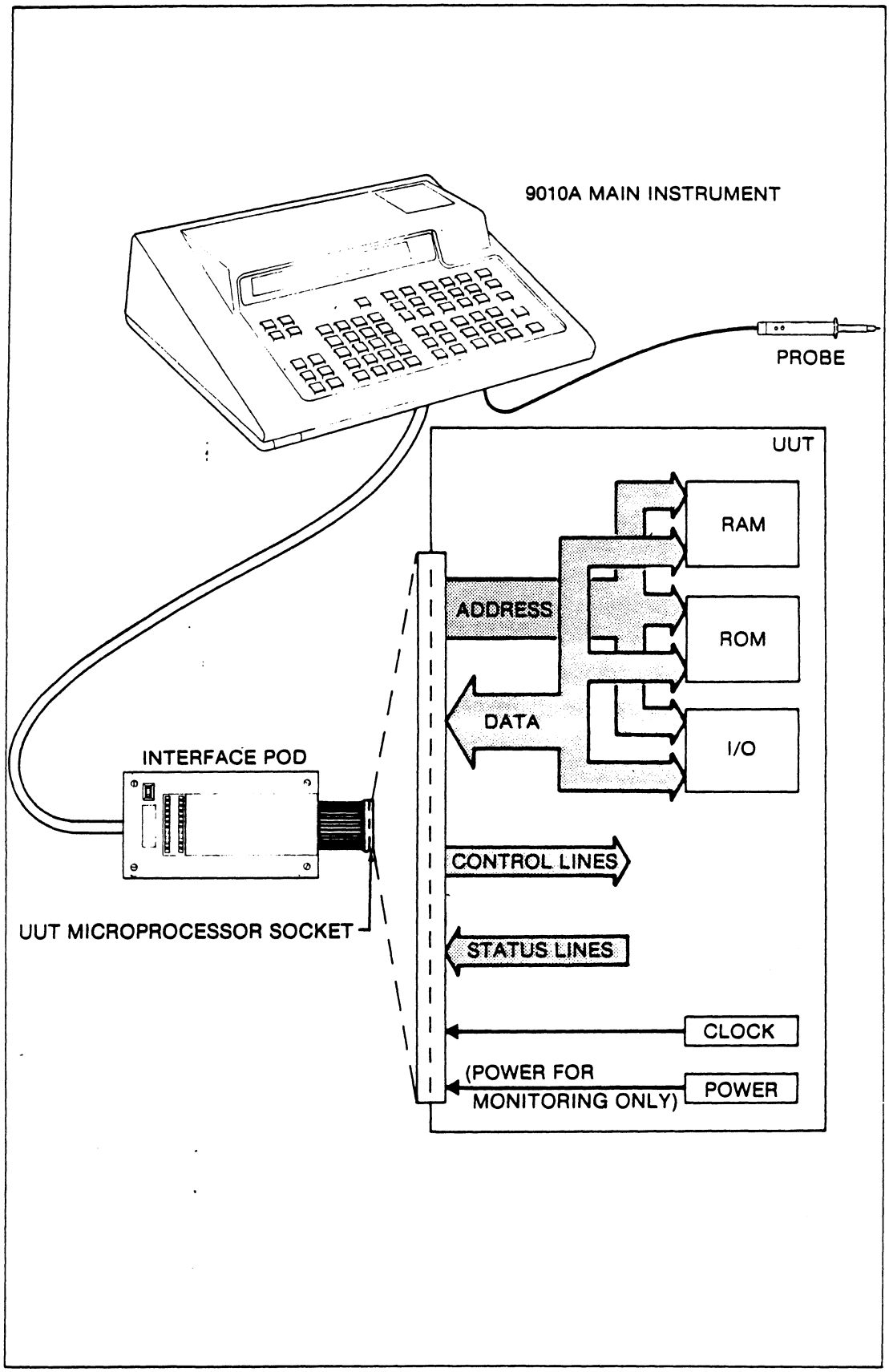


Figure 3A-1. Communication Between the 9010A and the UUT

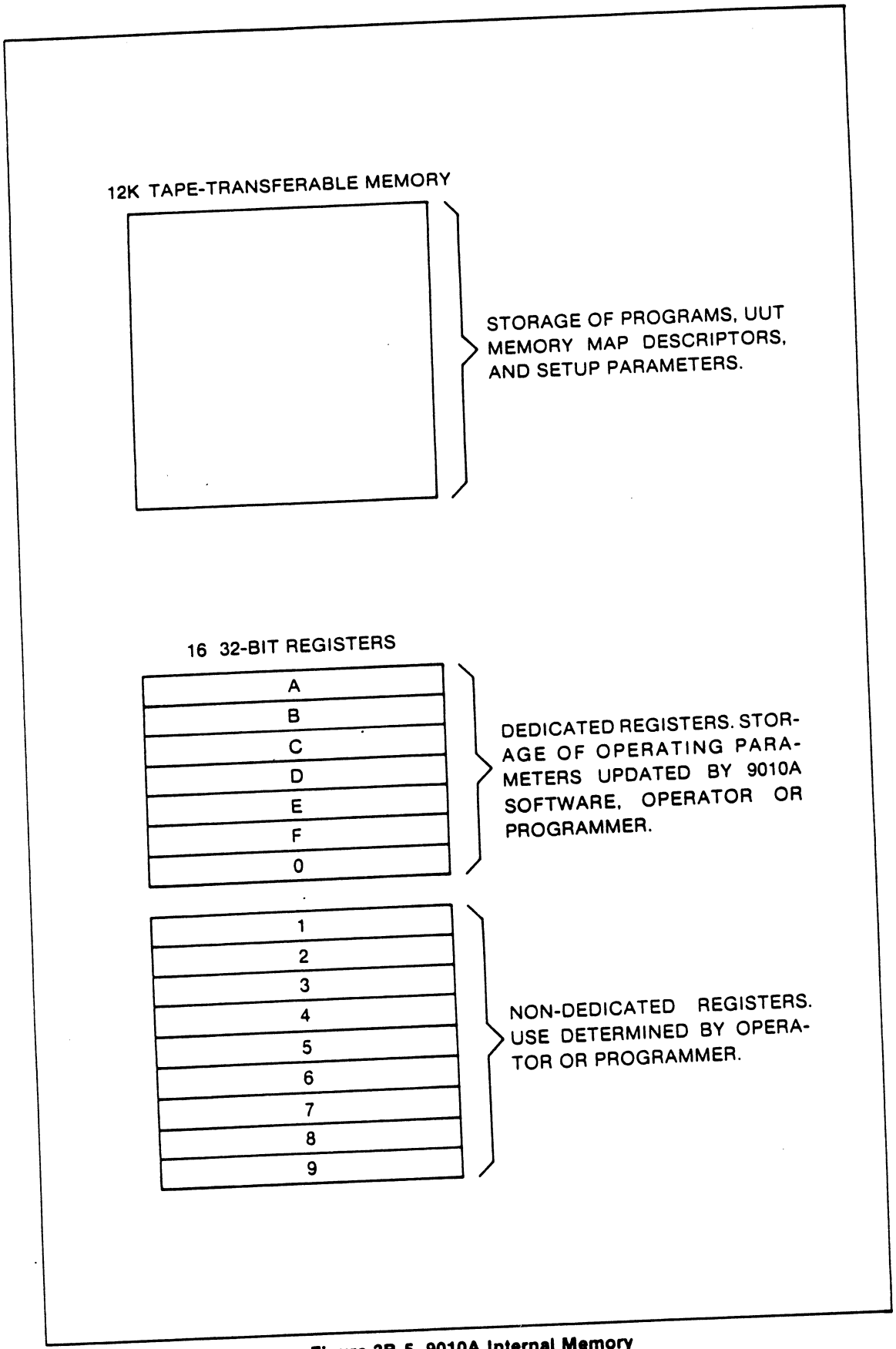


Figure 3B-5. 9010A Internal Memory

Table 4-1. Function of the Sixteen 32-bit Registers

TYPE OF REGISTER	REGISTER	FUNCTION
DEDICATED	A	Stores the last bit mask specified by the programmer. Also, if the UUT I/O address descriptors are invoked as default values when the I/O Test is performed, the bit mask specified by the last I/O address descriptor is stored in Register A.
	B	Stores the last ROM signature specified by the programmer. Also, if the UUT ROM address descriptors are invoked as default values when the ROM Test is performed, the ROM signature specified by the last ROM address descriptor is stored in Register B.
	C	Stores the last status/control information specified by the programmer for the Write Control or Toggle Data Control functions, or generated by the 9010A during performance of the Read Status function.
	D	Stores the last bit number (in the range 0-31) specified by the programmer for the Toggle Address, Toggle Data, or Toggle Data Control functions.
	E	Stores the last data specified by the programmer or generated by the 9010A during operation.
	F	Stores the last address specified by the programmer or generated by the 9010A during an operation involving the interface pod.
	0	Stores data accumulated during the Read Probe operation.
NON-DEDICATED	1-9	Non-dedicated registers for sole use by programmer for storage and manipulation of data as specified by the programmer.

NOTES:

1. Dedicated Registers A through F and 0 are also available to the programmer for storage and manipulation of data.

2. Registers 0 through 7 are local registers whose values are local to the currently executing program.

3. Registers 8 through F are global registers and unaffected by passing between called and calling programs.

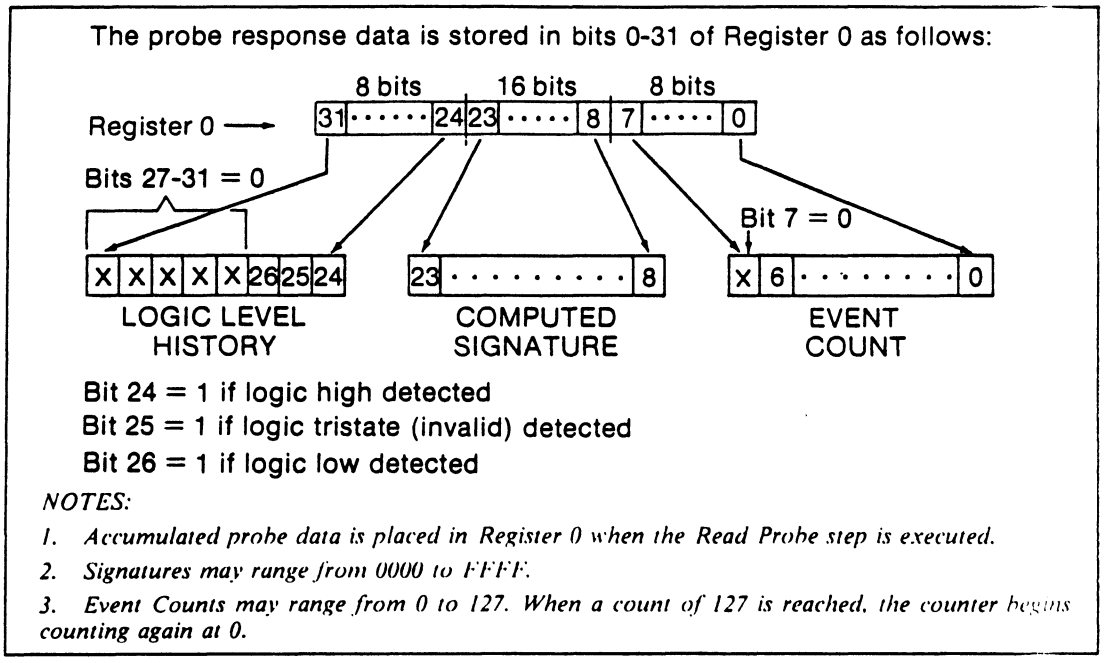


Figure 5-1. Probe Data Format for Register 0

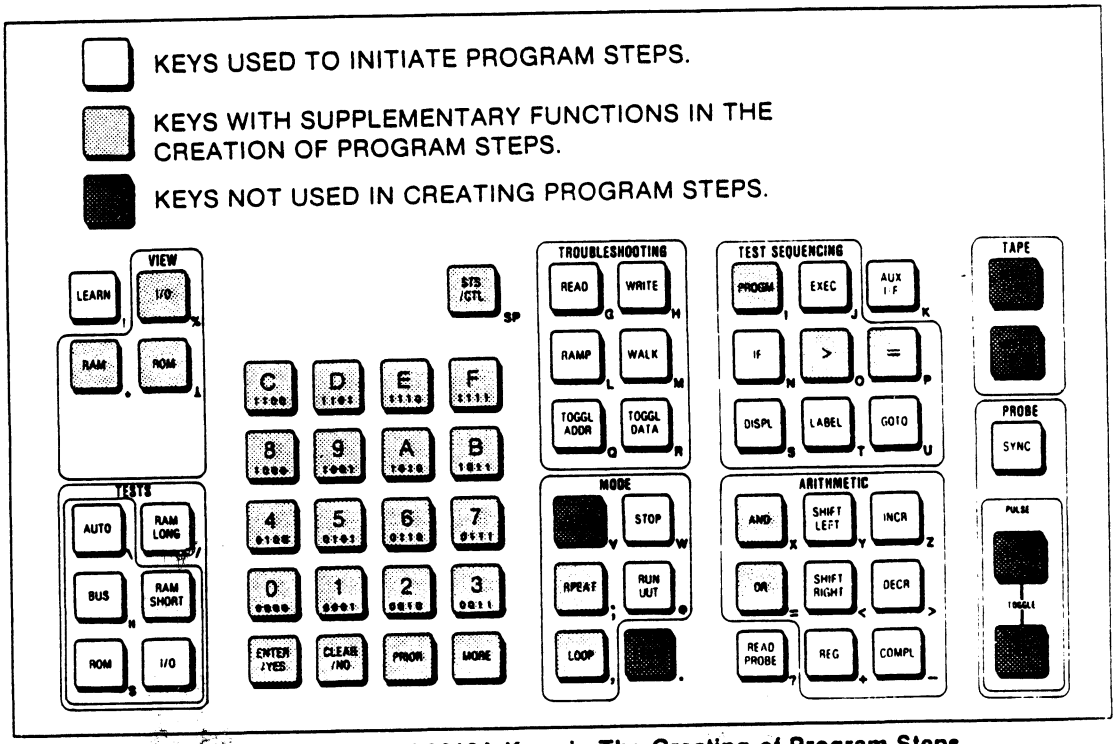


Figure 4-1. Function of 9010A Keys in The Creating of Program Steps

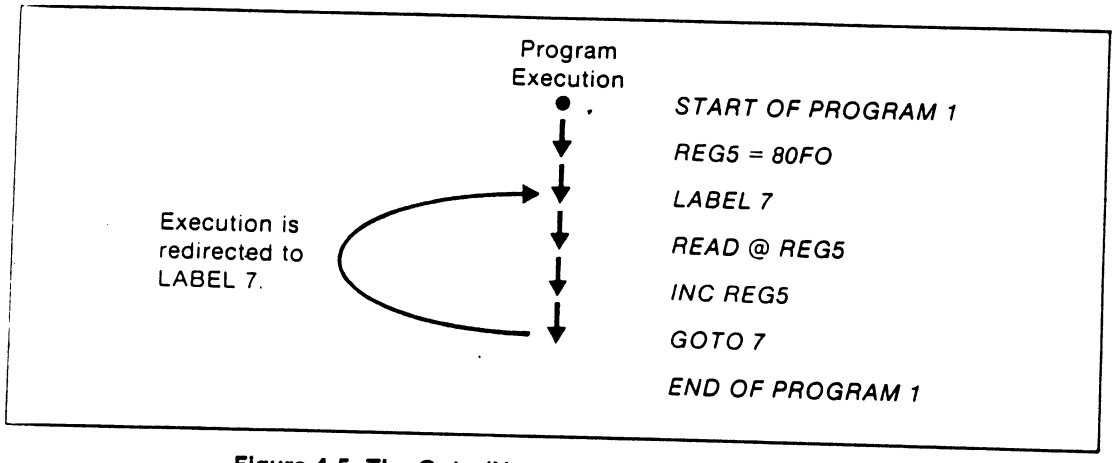


Figure 4-5. The Goto (Unconditional Branch) Step

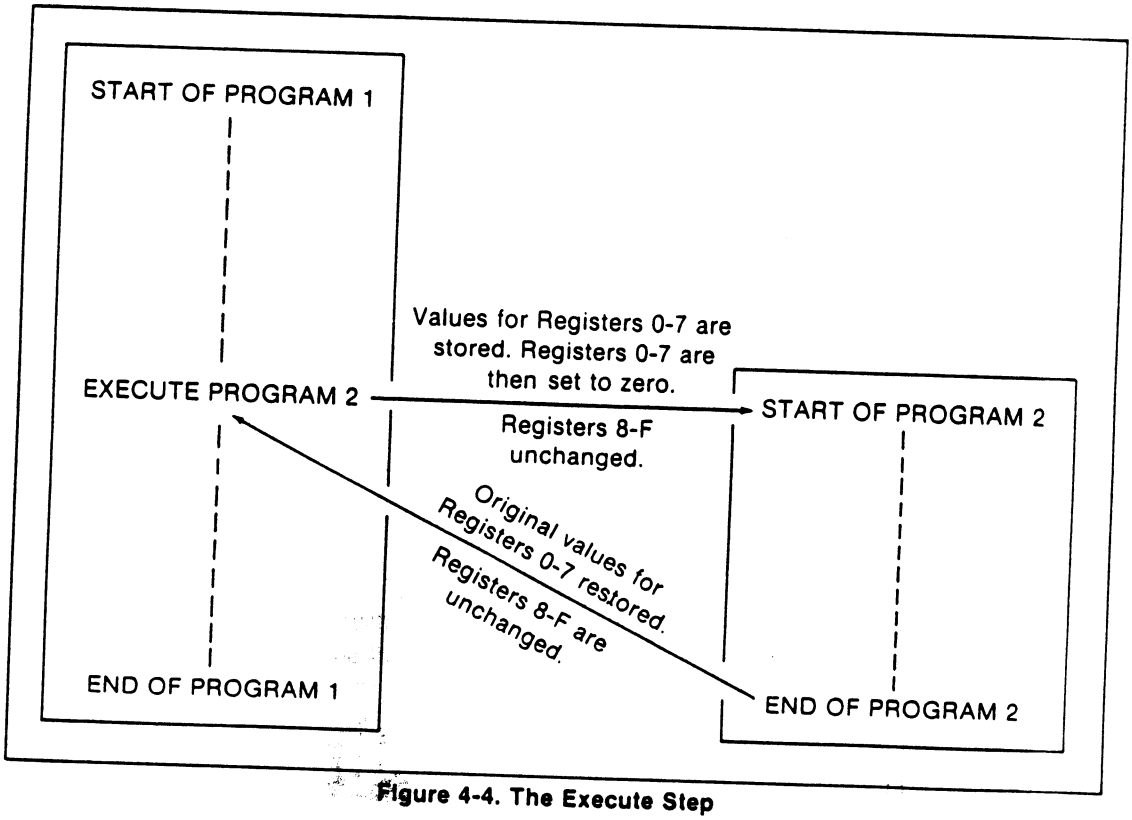
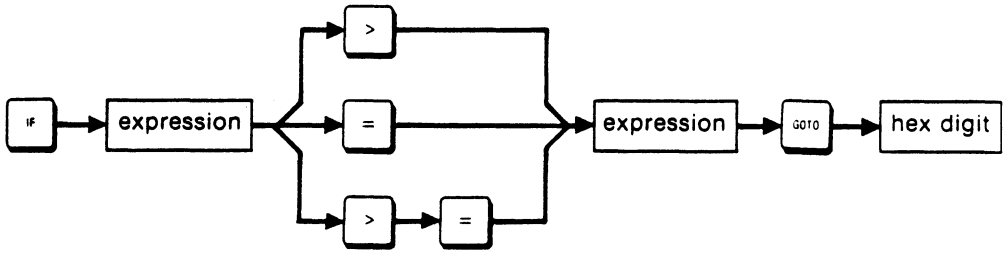


Figure 4-4. The Execute Step



EXAMPLES: *IF REG4 > 2000 GOTO 5*

IF REGE OR FFF7 = FFFF GOTO B

IF 8000 > REG6 GOTO 9

IF REGE CPL >= REG 3 GOTO A

NOTES:

expression = a sequence of keystrokes consisting of numeric values and/or registers. It may also include Arithmetic operations.

hex digit = single hexadecimal digit which designates the label where the branch redirects program execution.

Figure 4-6. The Syntax for the Specification of the If Step

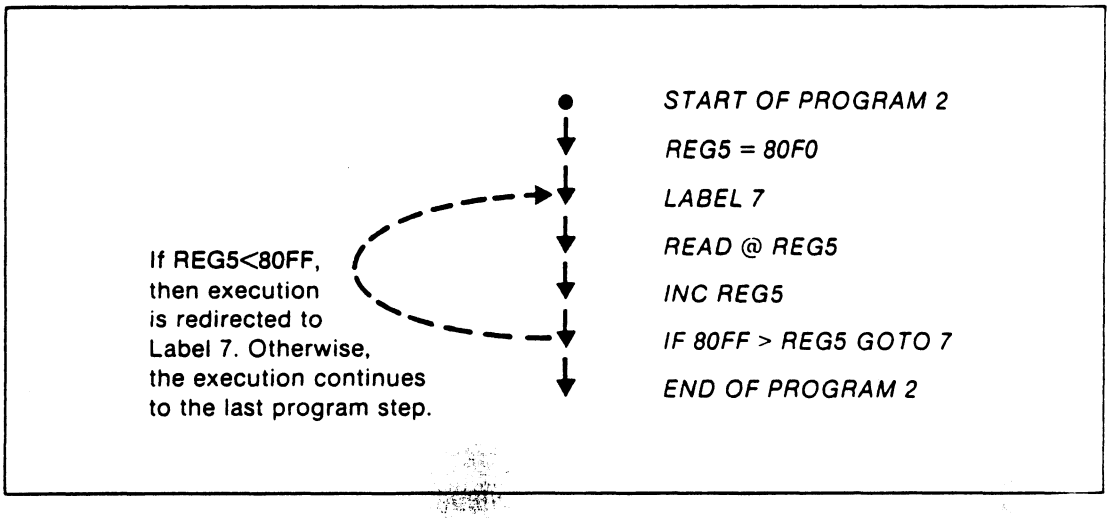


Figure 4-7. The If (Conditional Branch) Step

Table 4-3. Function of Special Symbols in the Display Step

CHARACTER OR SYMBOL	KEY NAME	ACTION CAUSED
◭ (bell)	ROM VIEW	Causes the 9010A to beep when the Display step is executed. The bell symbol does not appear on the display when the step is executed.
$	ROM TEST	When followed by a hexadecimal digit, it causes contents of the register designated by the digit to be displayed in hexadecimal on the display.
@	RUN UUT	The same as for the $ symbol, except that the contents are displayed in decimal.
/	RAM LONG	When followed by a hexadecimal digit, it suspends program execution and prompts for input. When the operator enters a hexadecimal value terminated by ENTER, the 9010A places the value in the register designated by the digit and resumes program execution. Pressing ENTER without specifying a hexadecimal value causes the value to default to the previous contents of the register.
	AUTO TEST	The same as for the / symbol, except that the 9010A accepts only a decimal entry.
?	READ PROBE	When followed by a hexadecimal digit, it suspends program execution and displays the question mark (?). If the operator presses the ENTER/YES key, the 9010A places a 1 in the designated register. If the operator presses the CLEAR/NO key, the 9010A places a 0 in the designated register. After the 1 or 0 is placed in the register, the 9010A removes the question mark and then resumes program execution.
%	I/O VIEW	When followed by a hexadecimal digit, it enables or disables asynchronous input from the operator during execution. Asynchronous input is stored in the register designated by the hexadecimal digit. Asynchronous input is described in Section 5.
+	REG	When it is the first character in the specification, it causes the following characters in the specification to be appended to the text that is on the display at the time the Display step is executed.

NOTE: In order to causes the $, /,\ , or % symbols to appear in the displayed text when the Display step is executed, the symbols must appear twice in the specification.

FLUKE®

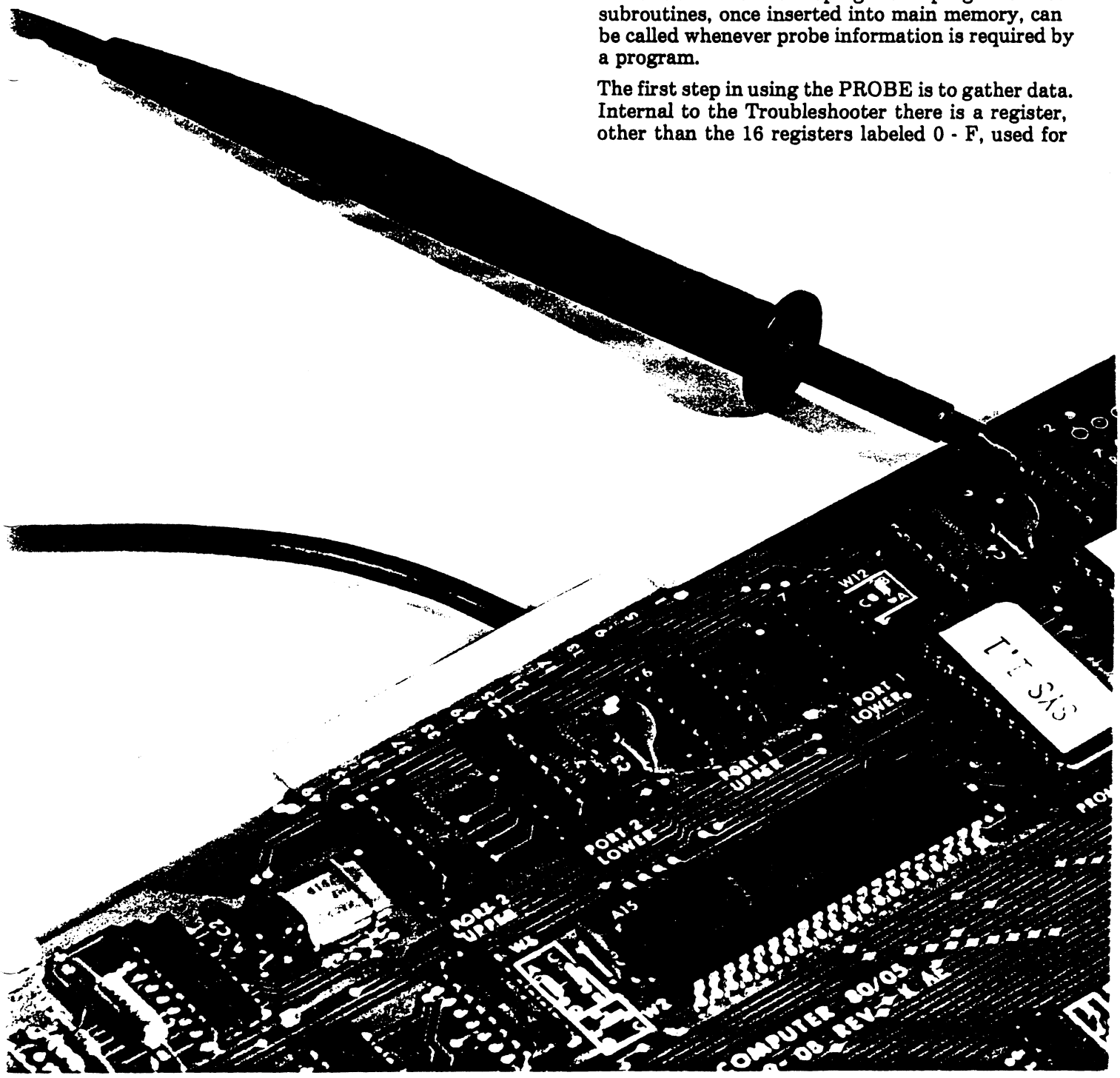
Technical Data

Application Information B0163

9010A Micro-System Troubleshooter: The Probe and the Program

The 9000-series Micro-System Troubleshooter's probe is used extensively when designing a Guided Fault Isolation (GFI) program. This bulletin will show how the probe can be made to interact with a GFI program and provide some subroutines that can be used when developing a GFI program. These subroutines, once inserted into main memory, can be called whenever probe information is required by a program.

The first step in using the PROBE is to gather data. Internal to the Troubleshooter there is a register, other than the 16 registers labeled 0 - F, used for

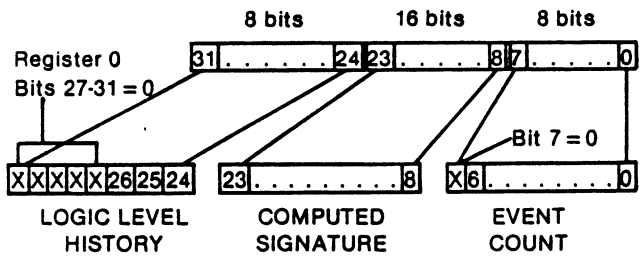


accumulating PROBE data. This bulletin will refer to this register as the probe register. Whether in the immediate or programming mode, the procedure for using this register is always the same. The three basic steps are:

READ PROBE	To clear the probe register
STIMULUS	Run stimulus by the probe. This could take the form of any Troubleshooter function i.e. RAMP, WALK, TOGGLE, etc.
READ PROBE	To terminate the gathering of data by dumping the data from the probe register into the display and register 0. In the executing mode, only register 0 is affected and not the display.

The only way that you can get probe data into a program is through Register 0. The following table shows how the probe data is formatted inside Register 0.

The probe response data is stored in bits 0-31 of Register 0 as follows:



Bit 24 = 1 if logic high detected
Bit 25 = 1 if logic tristate (invalid) detected
Bit 26 = 1 if logic low detected

NOTES:
1. Accumulated probe data is placed in Register 0 when the Read Probe step is executed.
2. Signatures may range from 0000 to FFFF.
3. Event Counts may range from 0 to 127. When a count of 127 is reached, the counter begins counting again at 0.

Figure 1

Once the data is in Register 0, the three pieces of data, LEVEL HISTORY, SIGNATURE, and COUNT have to be separated into their individual parts to be of any use.

To show this, using the Troubleshooter, assume that a READ PROBE operation has caused the value of 596EC55 to be entered into register 0.

Register 0 looks like this:

```
 0    5    9    6    E    C    5    5
0000 0101 1001 0110 1110 1100 0101 0101
```

Figure 2

Using the AND function of the Troubleshooter as a mask, unwanted data from Register 0 can be masked out leaving only the desired information as a result. If, for instance, only the count is required, then all bits EXCEPT 0 thru 6 need to be masked out. To do this you can AND Register 0 with a number that has bits 0 - 6 set to one and all other bits set to zero.

```
 0    5    9    6    E    C    5    5
0000 0101 1001 0110 1110 1100 0101 0101
ANDed with:
0000 0000 0000 0000 0000 0000 0111 1111
Resulting in:
 0    0    0    0    0    0    5    5
0000 0000 0000 0000 0000 0000 0101 0101
```

Figure 3

The Troubleshooter will only accept the AND value in hexadecimal form, so 1111111 converted tc hex is 7F.

To demonstrate this operation, perform the following keystrokes.

First set register 0 equal to 596EC55:

KEYSTROKE	DISPLAY
REG	REG_
0	REG0 = _
596EC55	REG0 = 596EC55_
ENTER	REG0 = 596EC55

Now to perform the MASK operation without changing register 0:

KEYSTROKE	DISPLAY
REG	REG_
1	REG1 = _
REG	REG1 = REG_
0	REG1 = 596EC55 _
AND	REG1 = 596EC55 AND _
7F	REG1 = 596EC55 AND 7F_
ENTER	REG1 = 55

This leaves the count in Register 1 and register 0 is unchanged.

To get the signature information, you can mask ou unwanted bits by ANDing with FFFF00.

KEYSTROKE	DISPLAY
REG	REG_
1	REG1 = _
REG	REG1 = REG_
0	REG1 = 596EC55 _
AND	REG1 = 596EC55 AND _
FFFF00	REG1 = 596EC55 AND FFFF00_
ENTER	REG1 = 96EC00

That leaves the signature followed by eight bits set to zero. To remove these trailing zeros, the shift function should be used. After performing eight shift right operations, the signature will be left as a result.

The keystrokes for this operation are:

REG	REG_
1	REG1 = _

Press SHIFT RIGHT key eight times.

ENTER	REG1 = 96EC

Bits 24-26 are used for the logic level history in Register 0.

To separate level history, AND register 0 with 7000000. Then shift register 1, right 24 times.

This completes the procedure for separating the probe data into its three parts. The next step is to put this procedure into a program.

If this procedure is to be a subroutine which will be called by another program, then global registers will have to be used to transfer the data back to the calling program. The global registers are 8 - F. So this program uses register 8 for the COUNT, register 9 for the SIGNATURE and register A for the LEVEL.

First write the program so it only performs the separating operation and not the read probe and stimulus operations.

KEYSTROKE	DISPLAY
PROGRAM	PROGRAM _
1	PROGRAM 1_
ENTER	PROGRAM 1 CREATED
REG 8	REG8 = _
REG 0	REG8 = REG0 _
AND	REG8 = REG0 AND _
7F	REG8 = REG0 AND 7F_
ENTER	REG8 = REG0 AND 7F
REG 9	REG9 = _
REG 0	REG9 = REG0 _
AND	REG9 = REG0 AND _
FFFF00	REG9 = REG0 AND FFFF00_
ENTER	REG9 = REG0 AND FFFF00
REG 9	REG9 = _
SHIFT RIGHT	REG9 = REG9 SHR _

Press the SHIFT RIGHT key seven more times:

ENTER	SHR SHR SHR SHR SHR SHR SHR SHR
REG A	REGA = _

KEYSTROKE	DISPLAY
REG 0	REGA = REG0 _
AND	REGA = REG0 AND _
7000000	REGA = REG0 AND 7000000_
ENTER	REGA = REG0 AND 7000000
REG A	REGA = _
SHIFT RIGHT	REGA = REGA SHR _

Press the SHIFT RIGHT key 23 more times:

ENTER	SHR SHR SHR SHR SHR SHR SHR SHR

For test purposes add a display statement that displays the LEVEL HISTORY and SIGNATURE in hexadecimal, and the COUNT in decimal. The display step should look something like this:

ENTER	DPY-LVL = $A SIG = $9 CNT = @8

Also add a step at the very beginning setting register 0 to a test value.

Press the PRIOR key until display =

KEYSTROKE	DISPLAY
PRIOR	START OF PROGRAM 1

Then enter:

REG 0	REG0 = _
596EC55	REG0 = 596EC55_
ENTER	REG0 = 596EC55

Now close the program:

PROGRAM	PROGRAM 1 CLOSED-XXXX BYTES LEFT

Now execute the program:

EXEC 1	EXECUTE PROGRAM 1_
ENTER	

Almost immediately, the display should read:

LVL = 5 SIG = 96EC CNT = 85

If the display is different, then check your program. Often there is a mistake in the number of Shift Right commands. Remember that when the SHR steps are displayed, every time "MORE" is pressed, eight characters are added to the display - less if it is the end of the program line.

Once this program is working properly with the test value, you should remove the first step which sets REG 0 to 596EC55 and the last step for displaying the values. Now you could add the three basic steps for probe operation.

Read Probe - Stimulus - Read Probe

As mentioned in the beginning of the bulletin, the probe operation should be used as a subroutine, for it is performed many times in different places throughout a GFI program. Putting these three steps into this program would make it very inflexible.

Fairchild 333

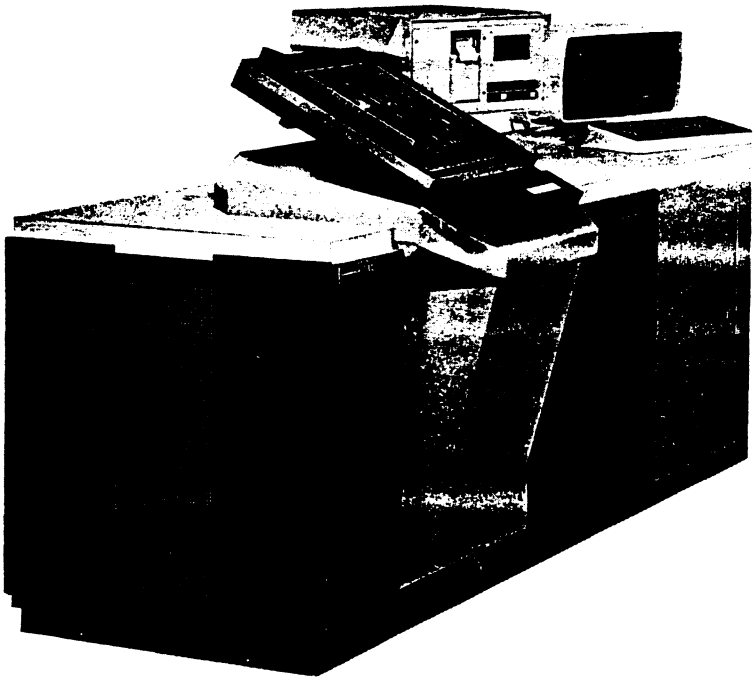
In Circut Tester.

ICT

Produktionstest av kretskort blir ofta en flaskhals i produktionen. Vi har resurser att lösa sådana problem.

Sammanfattning av tester som utförs vid incircuittest.

Komponenttyp	**Tester**
Strapp	Avbrottstest
Lödbryggor etc	Kortslutningstest
Motstånd	R värde + tolerans
Potentiometer	R värde + tolerans
Kondensator	C värde + tolerans
Pol. kondensator	C värde + tolerans, läckström
Spole	R värde + tolerans alternativt L värde + tolerans
Diod	Vf + tolerans, I_R
Transistor	Som två dioder, I_{CES}
OP-förstärkare	+sving, —sving, offset
Regulatorer	Utspänning med toleranser
Digitalkretsar	Sanningstabell med nivåer
PROM—ROM	Signaturanalys (referens riktigt kort)
RAM	Vandrande 1—0
Processor	Singelstep av utvalda kommandon
Korttest:	Varje kort förses med testremsa med ev felaktiga komponenter (tester).
Kortslutningar:	Alla kortslutningar tas bort och en omtest sker.
Statistik:	Felen sorteras och rangordnas så att snabb överblick blir möjlig.

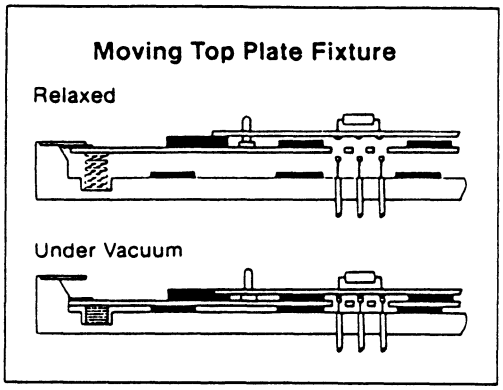


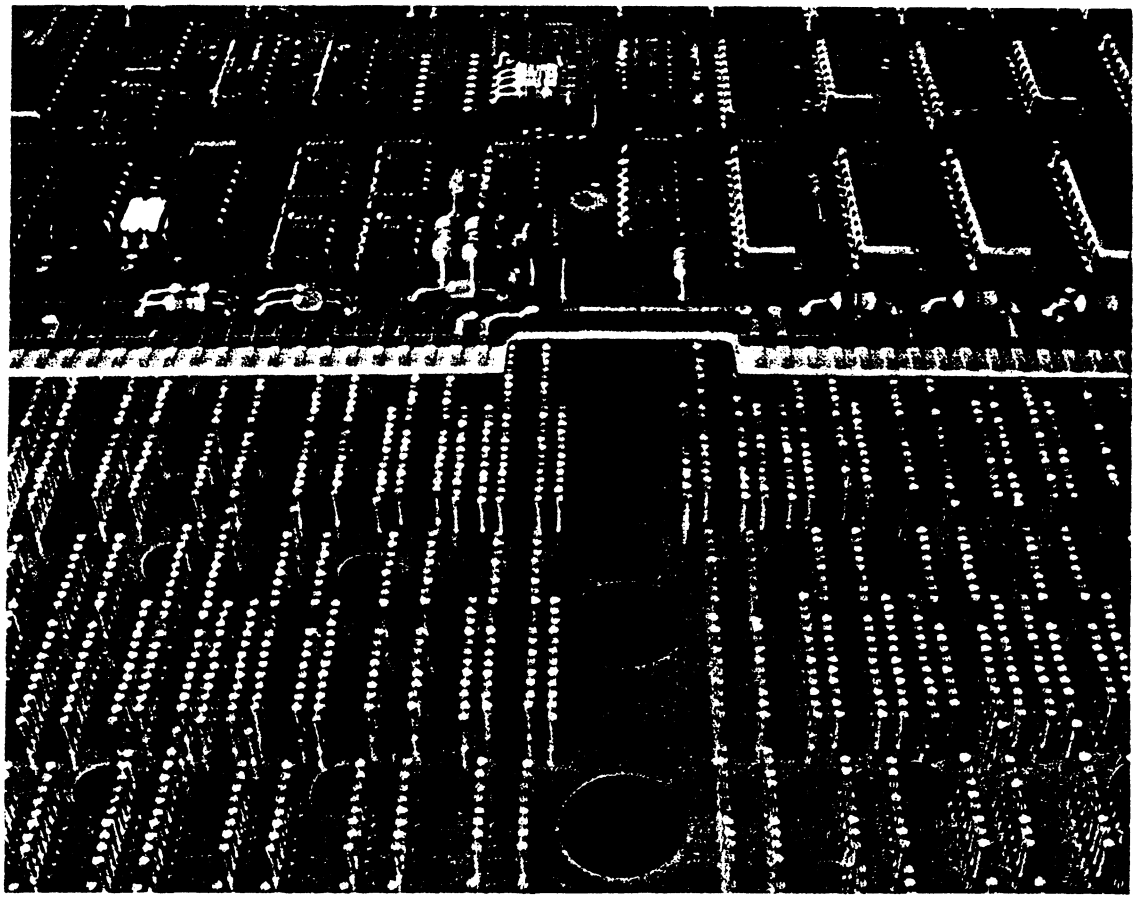
Programmering	För att få god feltäckning krävs ett omfattande programbibliotek och goda tekniska kunskaper.
Dataset	Varje komponent definieras med identitet, värden, toleranser, anslutningspunkter.
Automatisk programmering	Basickod genereras efter syntaxkontroll.
Debugg	Kontroll av varje test i programmet. Vi lägger till nödvändiga tester och tar bort omöjliga. Vi granskar kretsanvändningen utan kostnad: Vid debugg av program görs en kontroll om något ej är riktigt, t ex öppna ingångar, saknade skyddsdioder od. Fel i ovanstående meddelas kunden omgående.

FIXTURTILLVERKNING

Fixturer är ofta en svag länk i test. Genom att tillverka bra fixturer underlättas testen och onödiga fel undviks.

Nodplacering:	Nålarna i fixturen definieras mekaniskt på kortet och i kretsschemat. Eventuella extra noder bestämms. Vi granskar schemor utan kostnad. Vid vår nodplacering utgår vi från mönsterkortet för att kontrollera scheman.
Borrning:	Borrning av hål i botten respektive topplatta.
Montering:	Hylsor och nålar monteras på rätt höjd. Styrpinnar sätts fast.
Virning:	Anslutning av nålarna till kontaktpanelen.
Kontroll:	Mätning av kontakt till varje nål, kontroll av fixturen efter praktisk test 500 ggr.
Dokumentation:	Kretsschema med nodplacering, folie för snabb felsökning av kortslutningar.





— Nordens första och enda Testcenter för kretskort —

Chapter 5

PROGRAM MEMORY

5.1 INTRODUCTION

This section describes in detail the formats of the memory used to execute diagnostic programs on a UUT.

The uSA supplies up to 4K bytes of PROM and 256 bytes of RAM (Shadow RAM) to execute the diagnostic programs that exercise the UUT. Understanding the organization, operation, and interaction of these two memory areas provides the basis for all subsequent diagnostic efforts.

The uSA executes diagnostics on the UUT from the diagnostic program resident in the PROMs plugged into the front of the uSA, or from the program in the UUT memory. Because the memory of the UUT can occupy any address location, the diagnostic program could overlay the UUT memory or it could extend it. For example, the UUT memory could extend from 0000 to 2FFF and the diagnostic PROM could begin at 2000. In this case, the diagnostic program would overlay the UUT memory. If, however, the UUT memory was from 0000 to 1FFF and the diagnostic program began at 2000, the diagnostic program would extend the program memory space.

In addition to the mapping of the diagnostic program to UUT memory, the uSA contains a 256-byte block of RAM, called "Shadow RAM", that dynamically overlays the control information of the diagnostic program in PROMs. The diagnostic program uses the Shadow RAM during execution.

Figure 5-1 illustrates how the diagnostic program can overlay UUT memory, and how the Shadow RAM overlays the diagnostic Program.

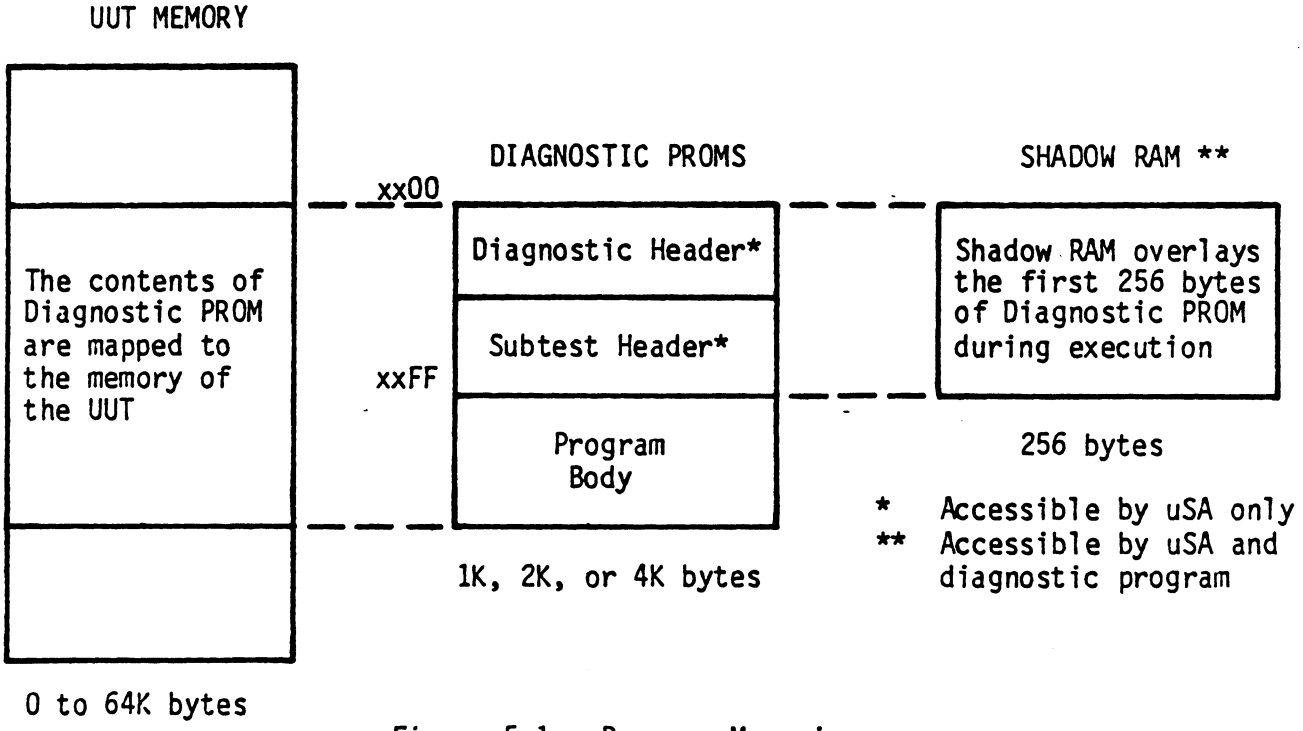


Figure 5-1. Program Memories

5.2 DIAGNOSTIC PROM

The diagnostic program in the PROM (or pair of PROMs) consists of control information and a program body as shown in figure 5-2.

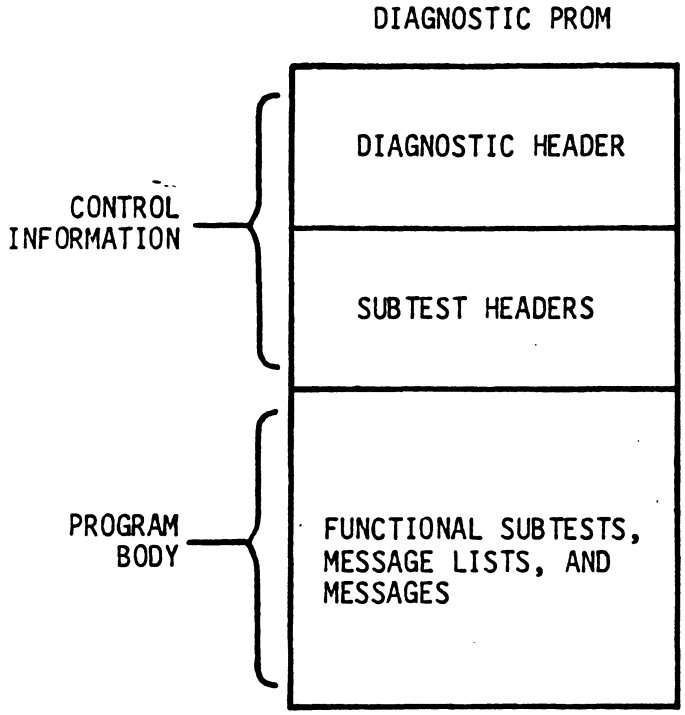


Figure 5-2. Diagnostic Program in PROMs

5.2.1 Control Information

The control information in the PROM consists of a Diagnostic Header followed by the subtest pointers and the Subtest Headers. If this information occupies less than 256 bytes, the leftover bytes can not be used because the uSA overlays the first 256 bytes of PROM 1 of the diagnostic program with Shadow RAM during program execution (see figure 5-1). The uSA provides the Shadow RAM to the users program at execution time. The uSA can switch the RAM in and out of the working memory space as required, for access to the diagnostic program control information and also to service special requests made by the diagnostic program. The information contained in the diagnostic header and subtest header areas is not accessible to the program during execution.

5.2.2 Program Body

The bytes following the first 256 bytes are reserved for the body of the program. The body consists of functional subtests, message lists and messages and usually begins at the starting location of the PROM plus 256 bytes (100H). However, if the diagnostic program requires more than 256 bytes of control information, the body starts in the next contiguous byte after the last byte of the control information. Figure 5-3 illustrates an example in which control information occupies less than 256 bytes, and figure 5-4 illustrates an example in which control information occupies more than 256 bytes.

5.3 SHADOW RAM

The uSA operating system uses shadow RAM for data transfer and service requests between the uSA and the diagnostic program. Five bytes of the Shadow RAM are used for making a service request. When data is stored in the service request byte, the uSA interrupts, pauses the UUT and determines what action is to be taken.

The data stored at the Service Request byte communicates to the system: 1) when the subtest is finished, 2) when the diagnostic program is finished with all of the subtests, and 3) if there is a message to be displayed. The Service Request byte also defines what action to take after servicing the requests. The options are: 1) pause the microprocessor under test, 2) wait a period of time, or 3) continue.

If the wait option is selected, the wait time is stored in the Wait Time byte in the Shadow RAM (see figure 5-5). If the request was to display a message, then the two bytes following the service request byte point to the message list. The status byte tells the diagnostic program which, if any, fault isolation function is active. The rest of the shadow RAM is available to the programmer for data storage and stack pointer.

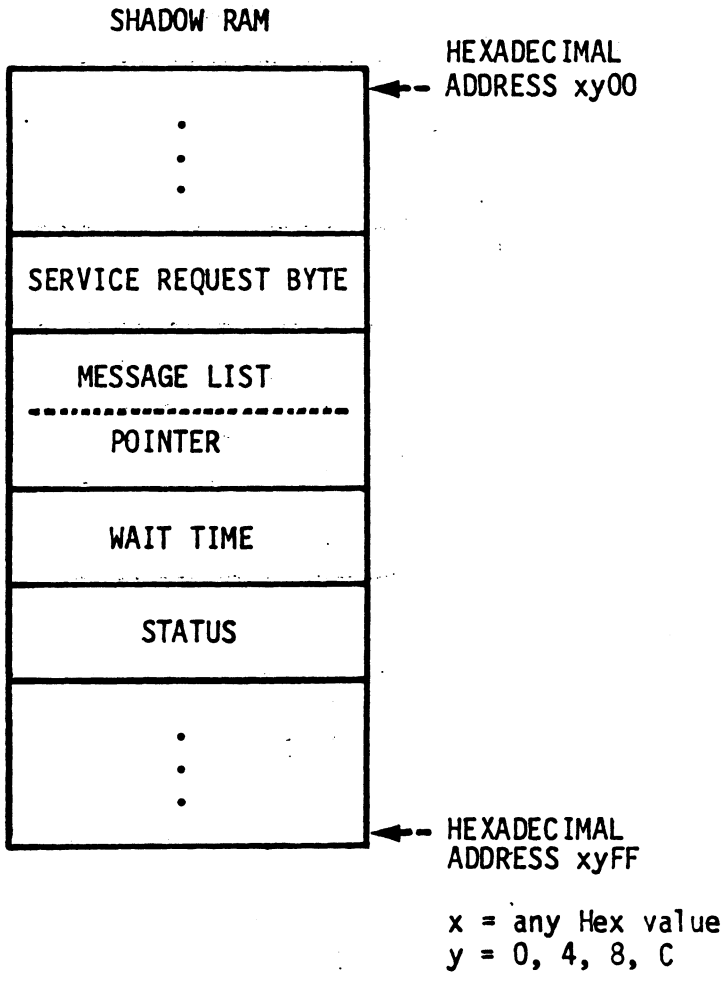


Figure 5-5. Shadow RAM Format

6.1 DIAGNOSTIC HEADER

The Diagnostic Header contains information about the operating environment of the diagnostic program. The Diagnostic Header must start at the first address in the diagnostic PROM. Figure 6-2 shows the format of the diagnostic header block and the position of the diagnostic header block in the diagnostic program.

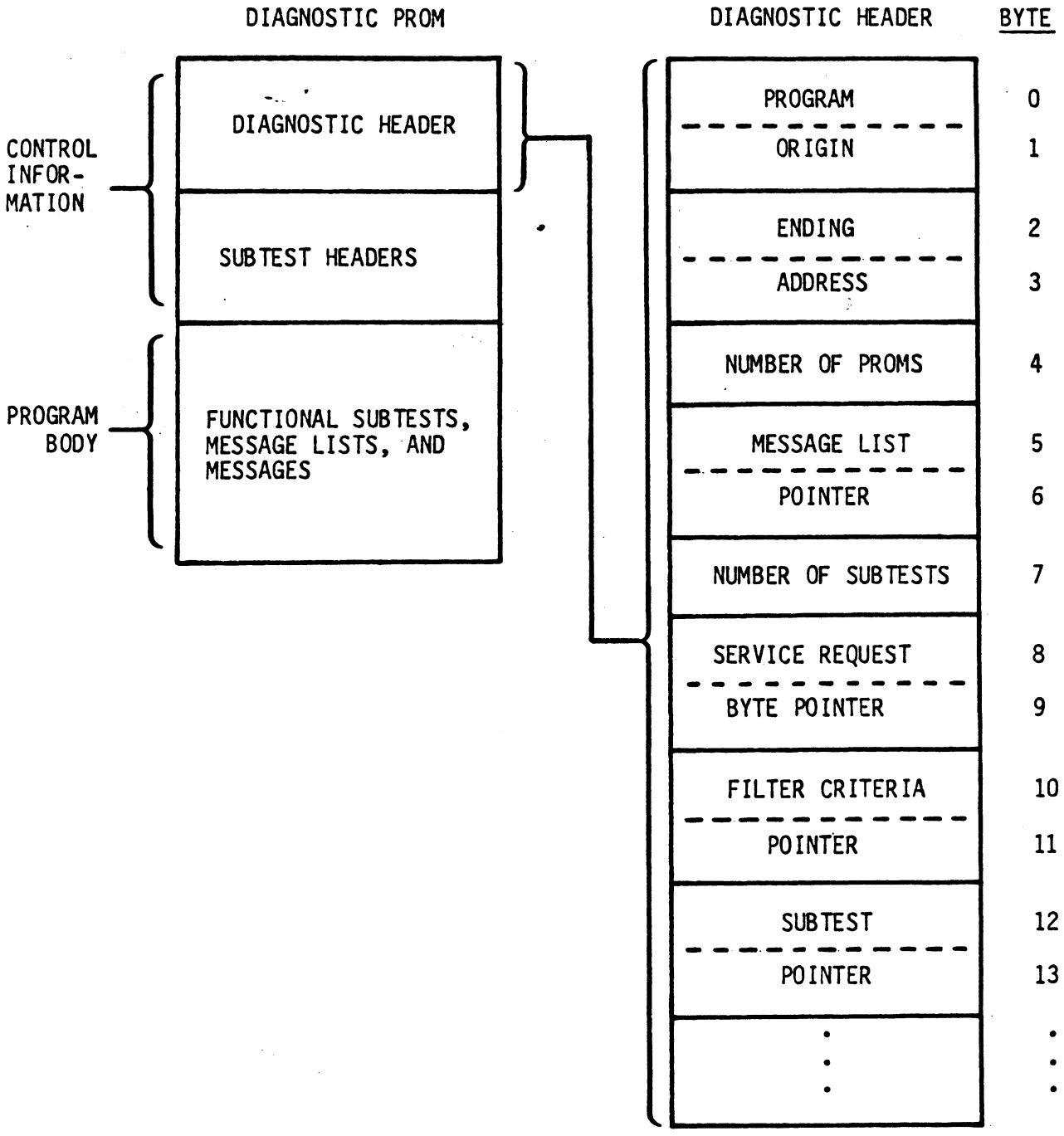
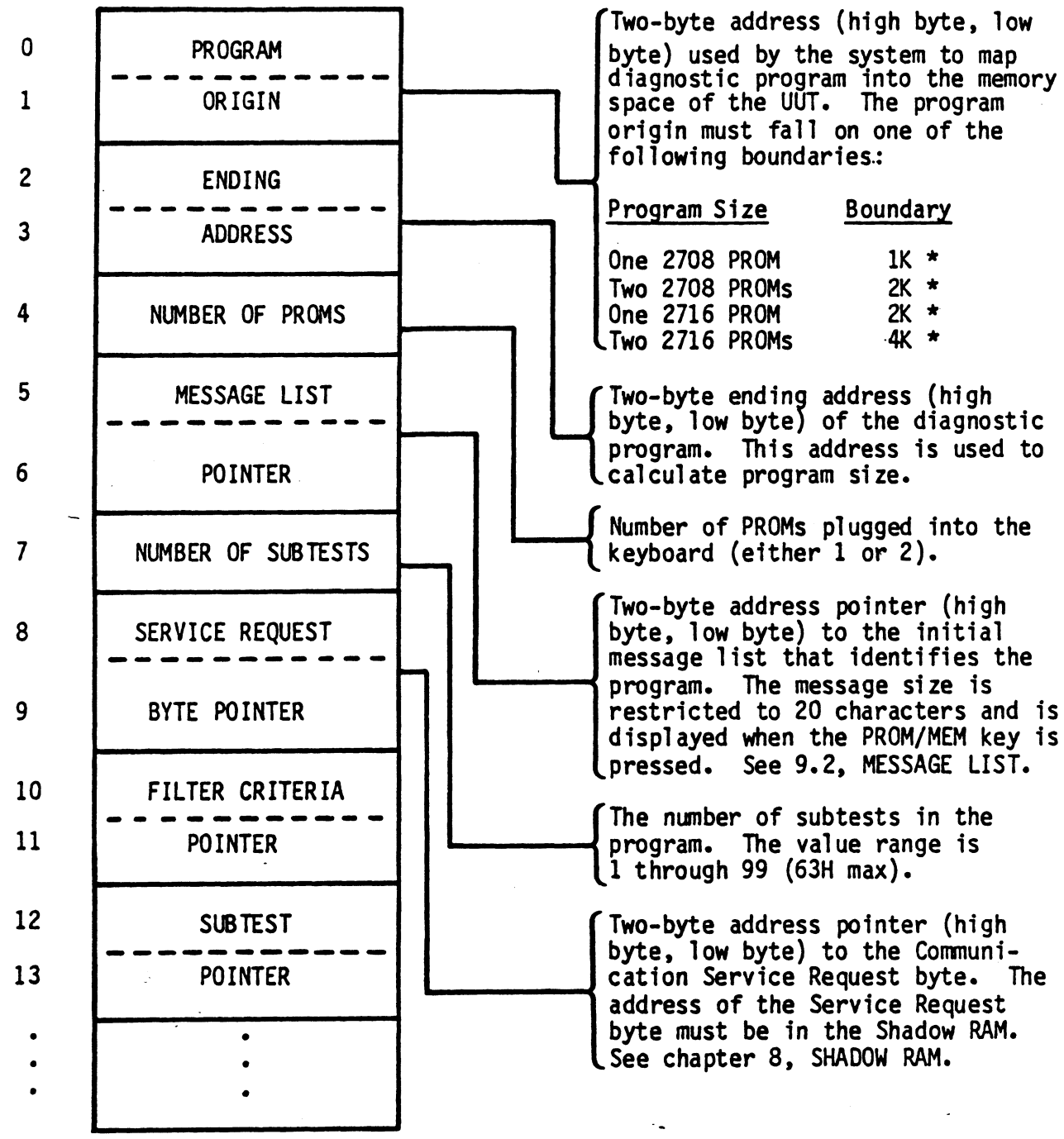


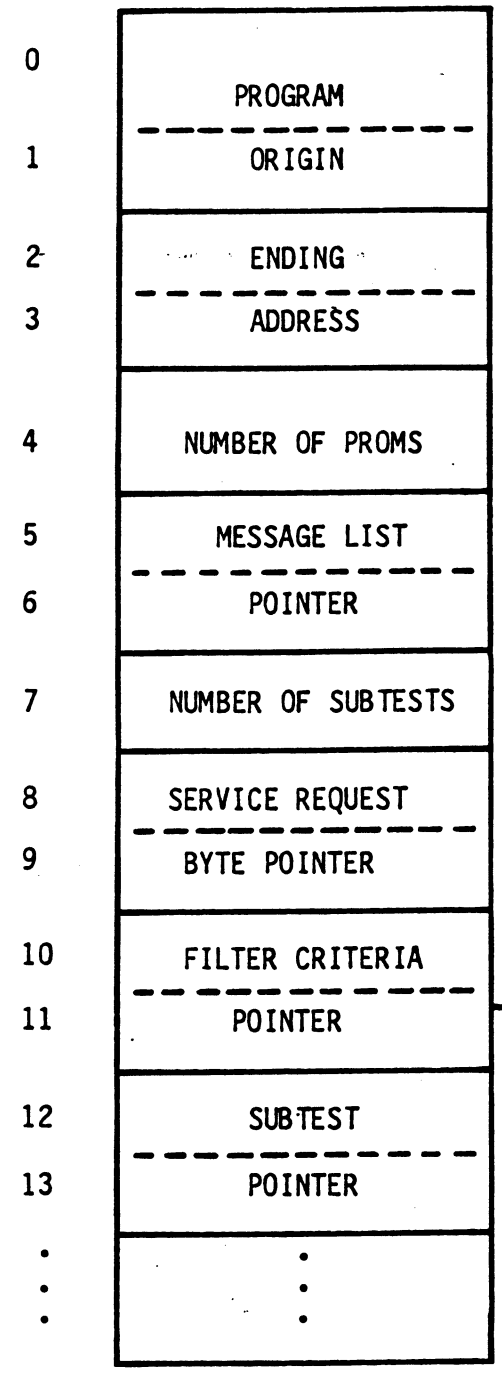
Figure 6-2. Diagnostic Header

A description of the bytes of the diagnostic header follows:



* Boundary 1K = x000, x400, x800, xC00
2K = x000, x800
4K = x000

Diagnostic Header description (continued).



DESCRIPTION

Two-byte address pointer (high byte, low byte) to the filter criteria. This data is optional. If no filtering is specified, the value of the address pointer must be 0. See chapter 7, DIAGNOSTIC PROM FILTER CRITERIA.

Two-byte address pointers (high byte, low byte) to subtest header blocks. A subtest pointer is required for each subtest, and a diagnostic program may have as many as 99 subtest pointers.

6.2 SUBTEST HEADER

The Subtest Header defines the control for the individual subtest. The most efficient placement of the subtest control blocks is immediately following the Diagnostic Header block, as this area is overlayed by the shadow RAM.

In addition to the information required in the subtest header, the programmer has the option of specifying criteria for any or all of the fault detection functions: SIG, FREQ, COUNT, INTRVL.

When the optional data is present and the operator presses a fault detection key, the uSA determines how to take the measurement from the information in the subtest header. Figure 6-3 shows the position of the Subtest Header Block in the diagnostic program and the format of the Subtest Header.

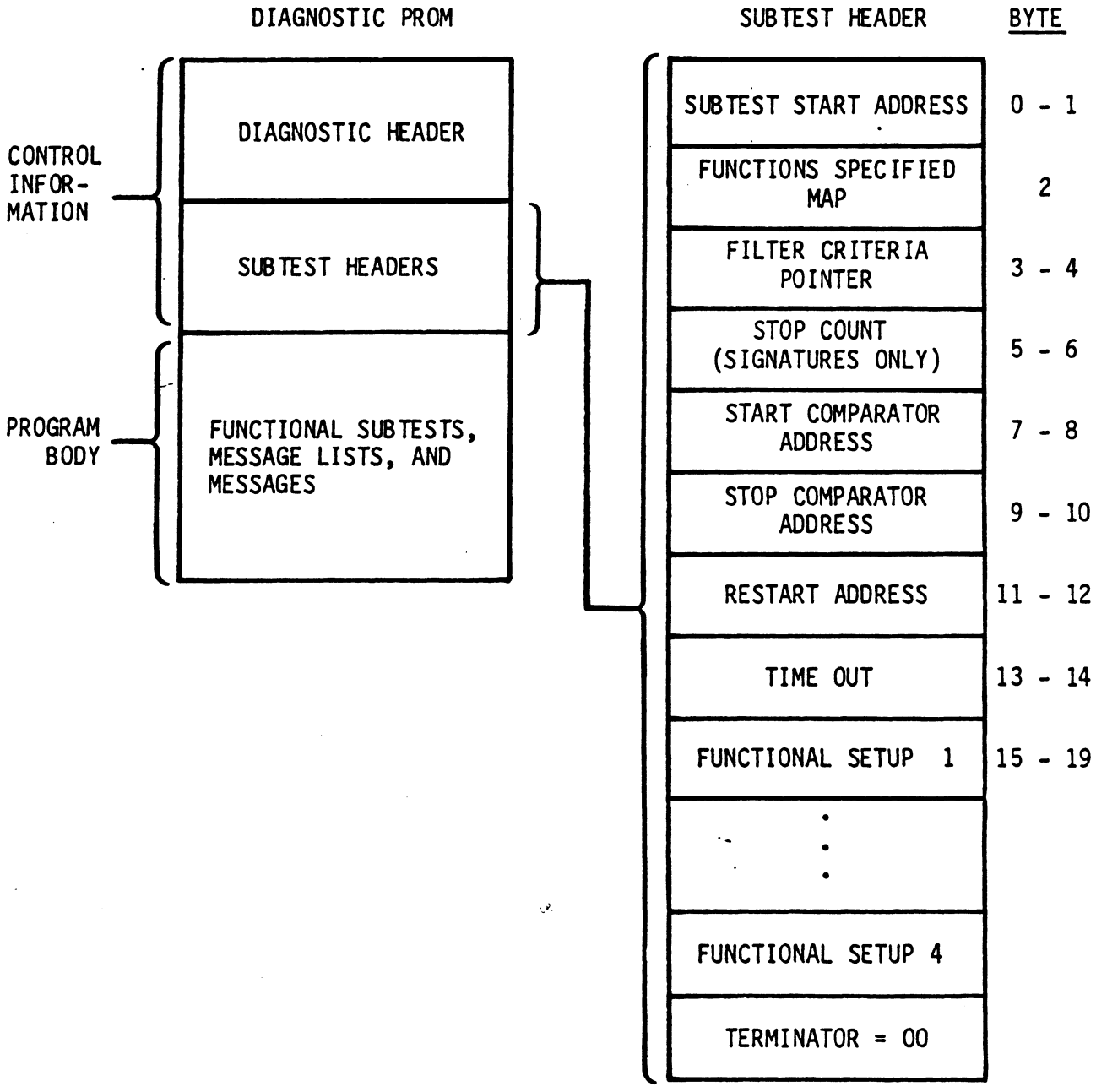
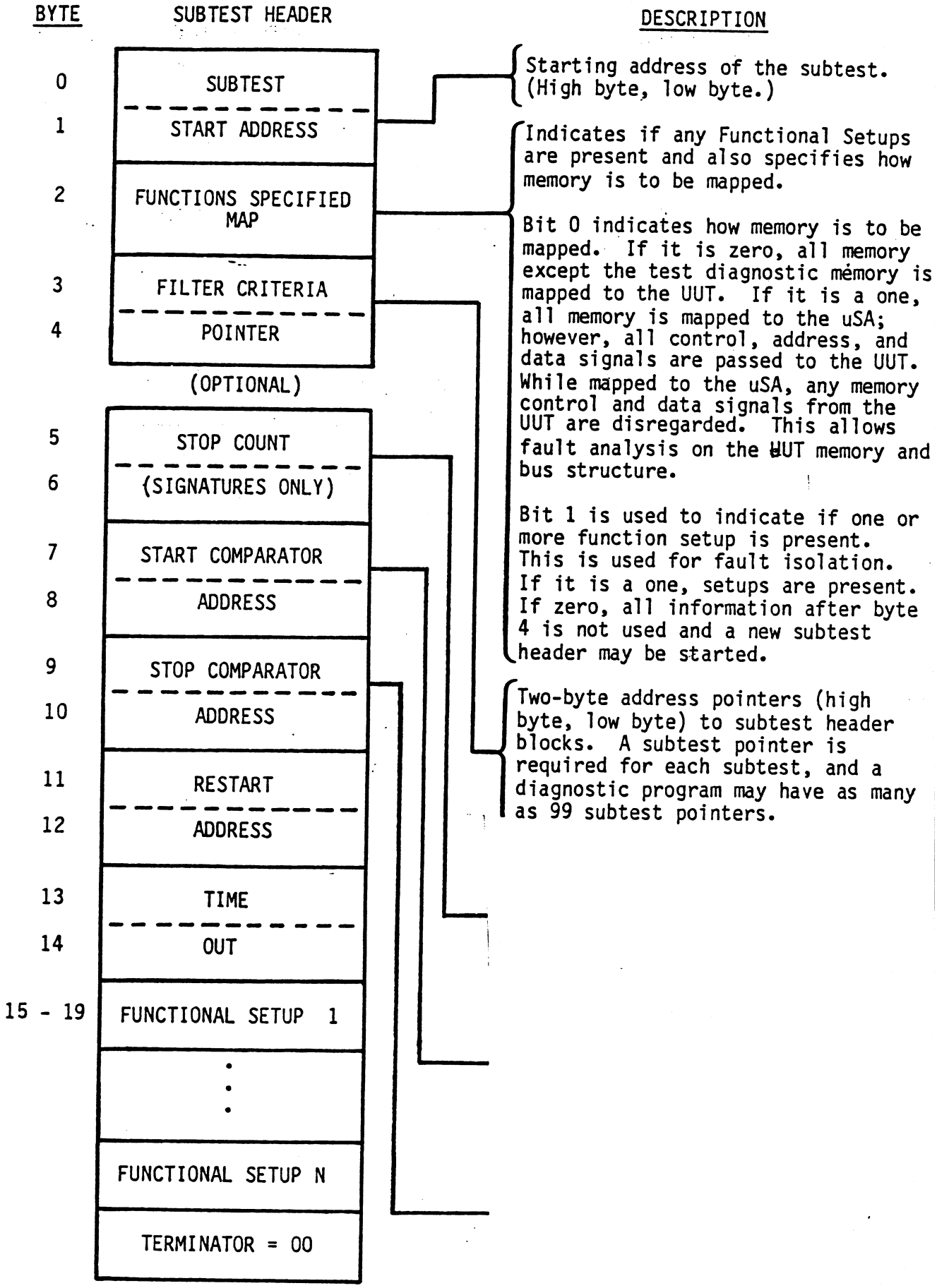


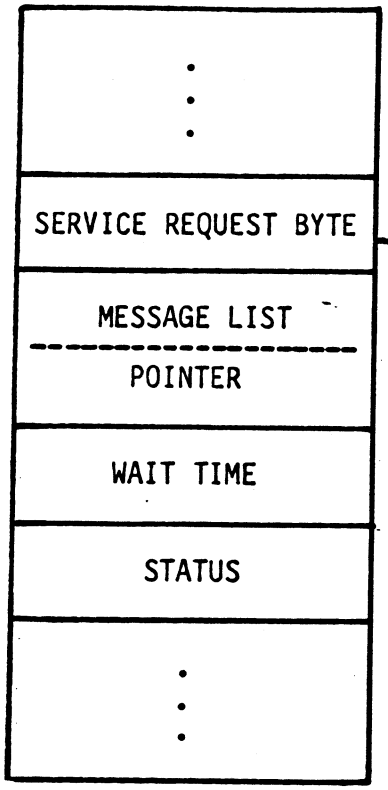
Figure 6-3. Subtest Header

A description of the bytes of the subtest header follows:



Each byte of the Shadow RAM is described below:

SHADOW RAM



DESCRIPTION

The uSA uses service requests to provide the diagnostic program with the functions listed below. Service is requested by storing the hex value for the function in the Service Request Byte. The hex values may be ORed together for multiple requests. Storing the coded data in the Service request Byte causes the uSA to perform the request. The request types are defined below.

Hex Value	Description
01	END SUBTEST

This signifies the end of the subtest. The program is not paused. One of the following actions will occur, depending on the operator selected subtest:

If SUBSEL = 0 at the end of each subtest, the subtest number is incremented and the next subtest is executed until the last subtest has been run. At this point, the subtest number is set to 1 and the program continues to loop.

If SUBSEL does not equal zero, the program loops on the selected subtest.

02 MESSAGE OUTPUT

The Message Pending LED indicator is illuminated and the program is paused for operator inspection of the message. If input is allowed, it can be entered from the appropriate keypad at this time. See the Message List description in chapter 9 for details.

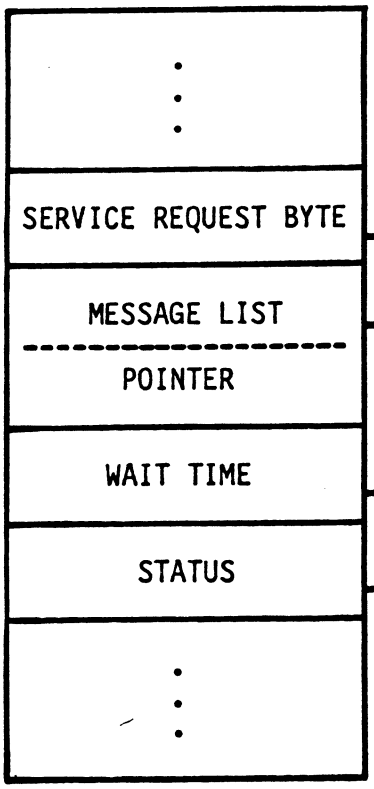
04 END TEST

The program is paused and one of the following actions will occur, depending on the operator selected subtest.

If SUBSEL = 0 the program will be restarted at subtest 1 when the RUN key is pushed.

If SUBSEL does not equal zero, the selected subtest will be repeated when the RUN key is pushed.

Shadow RAM description (continued)



SERVICE REQUEST (continued)

Hex Value	Description
08	HALT OVERRIDE (or CONTINUE) This value, which can be ORed with any other service request, causes the uSA to perform the requested action and then to resume operation. This action can be used for diagnostic loop control and message output.
10	WAIT This value can be used by itself or ORed with the Message Output request. After displaying the message, the uSA waits a specified amount of time before resuming. The time is specified by the Wait Time Byte.

For message output to the uSA via a service request, the two bytes immediately following the Service Request Byte must contain the absolute address of the associated Message List. (High byte, low byte.)

This value specifies the amount of time to wait. Each hexadecimal digit represents approximately 100 milliseconds.

The Status Byte, the fourth byte after the Service Request Byte, is set by the uSA to indicate which fault detection measurement is currently being made. The Status Byte is updated whenever a function key is pressed.

The format of the Status Byte is:

Hex Value	Description
00	No Fault Detection Measurement
10	Signature
20	Count
40	Interval
80	Frequency

Chapter 9

DIAGNOSTIC PROM MESSAGES

9.1 INTRODUCTION

Diagnostic programs can initiate message displays and request data input from the operator. The Message List is a control block in the body of the Diagnostic PROM that is used to control and specify display data and input operations. The Message List is addressed by a pointer in the Shadow RAM, as shown in figure 9-1, or by a pointer in the Diagnostic Header. See section 6.1.

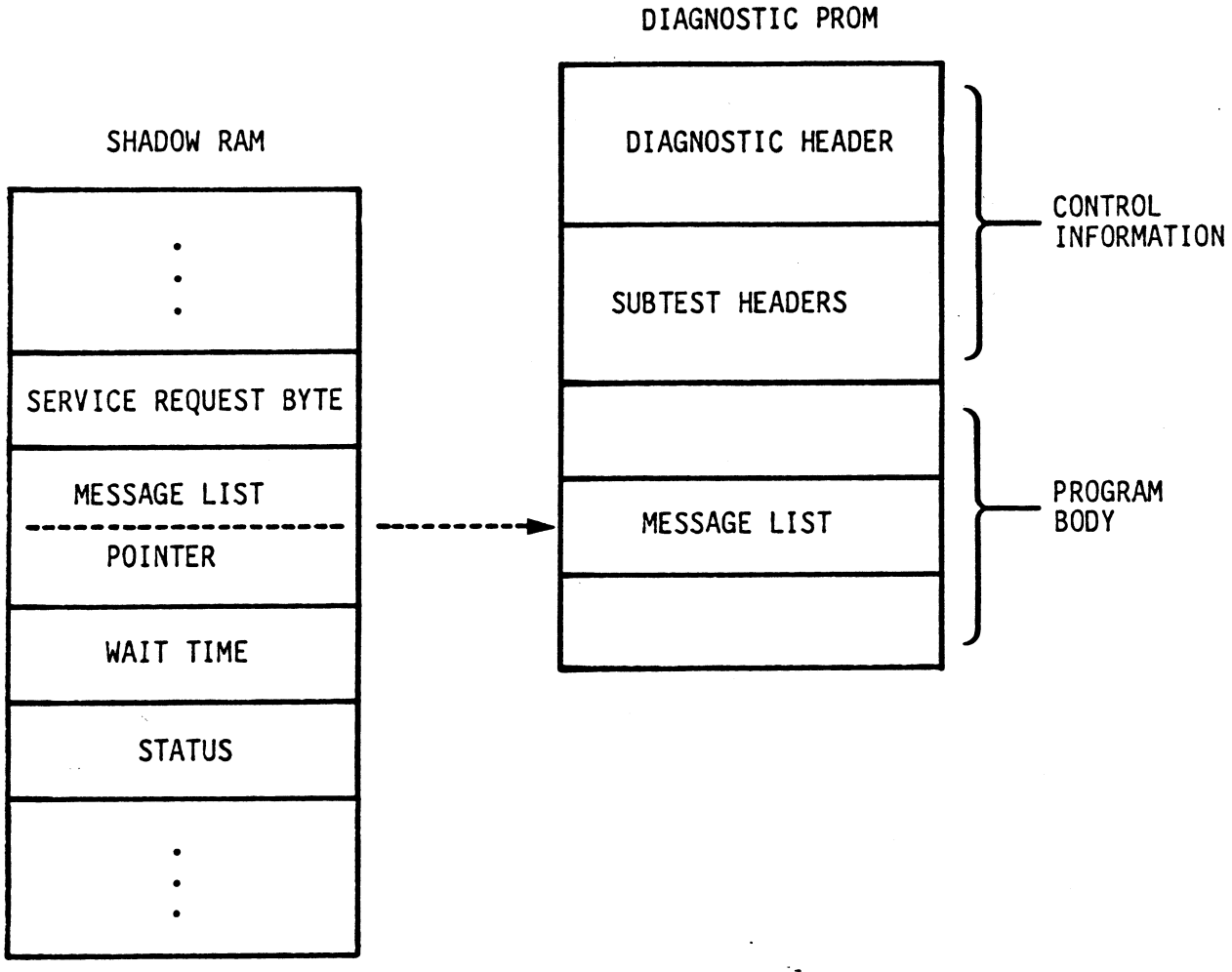


Figure 9-1. Message List

Messages can be from 1 to 20 characters in length. If more than 20 characters are required, messages may be linked, in which case multiple lines are displayed. The test program initiates a message output by storing a data byte in the Service Request Byte. A message can be one of two types: "Display and Pause" or "Display and Continue."

9.1.1 Display and Halt Messages

The MESSAGE PENDING LED indicator is illuminated, a beep is sounded, and the test program execution is paused. If the display is not being used to view the register, a memory or I/O message will be displayed. If the display is being used, the MSG key must be pushed to view the message. When the MSG key is pressed, the message will be displayed. Once the MSG key has been pressed, messages will be automatically displayed until a key is pressed that results in a display change. If more than one display line is associated with the message (linked message) the operator can step through the message by pressing the INCR key. To resume test execution, press the RUN key.

9.1.2 Display and Continue Messages

Similar to above except that test execution automatically resumes after the message is displayed. Note that a delay may be necessary to guarantee display stability long enough for human perception. To output a message of this type, the Service Request Byte consists of the message output bit ORed with the halt override bit or the wait bit.

To display a message, the diagnostic program will normally store the wait time, if any, in the Wait Time Byte, then store the address of the Message List in the Message List Pointer, and lastly, store the Message Output Request in the Service Request Byte.

9.2 MESSAGE LIST

A Message List consists of one or more message list entries, as shown in figure 9-2.

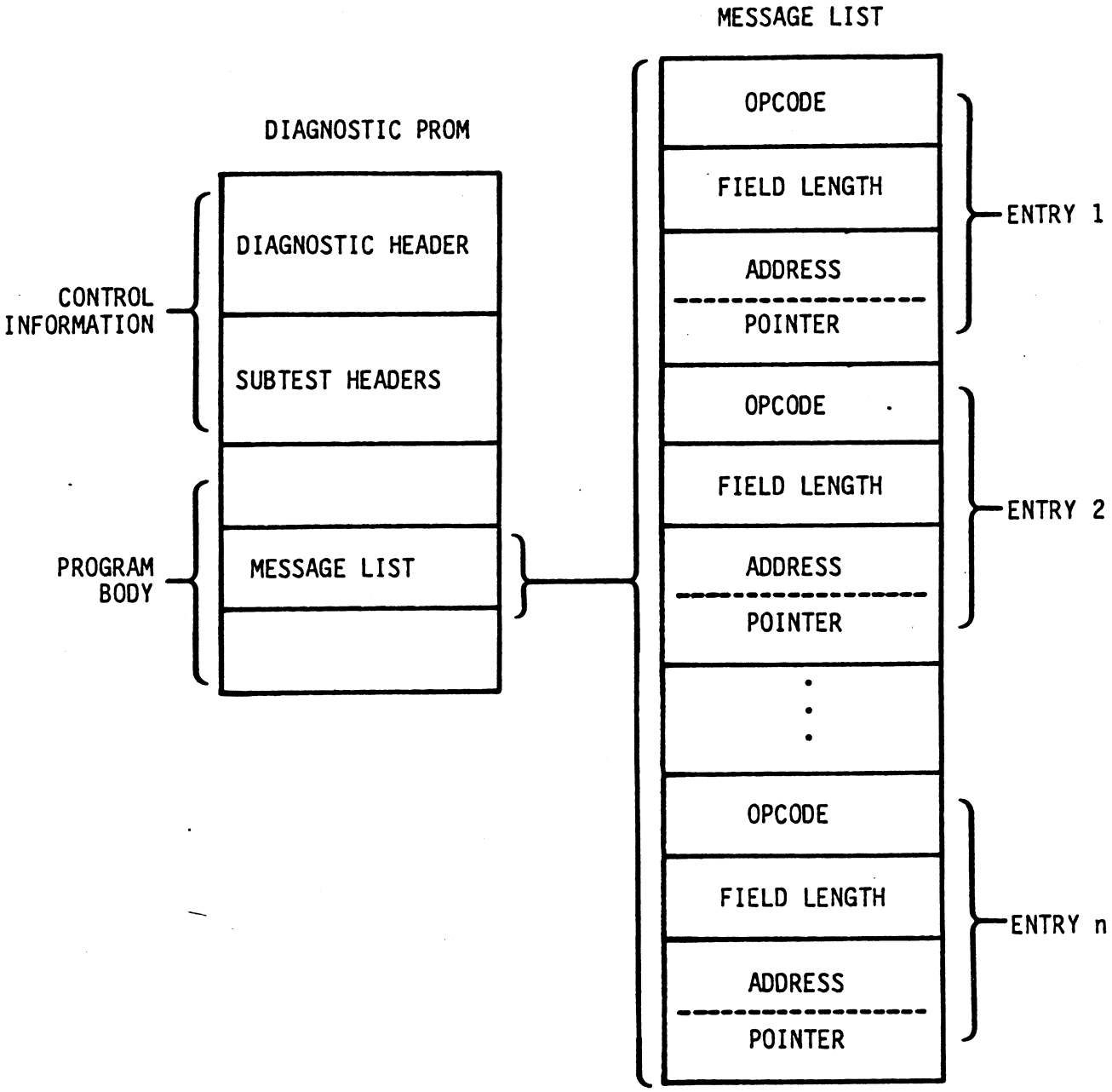
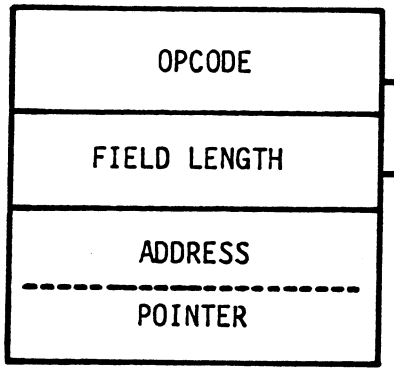


Figure 9-2. Message List

A description of the bytes of the Message List follows:

BYTE	MESSAGE LIST ENTRY	DESCRIPTION
0	OPCODE	Specifies the operation type and data format. The definition of the values for this byte are shown below:
1	FIELD LENGTH	Length of the field displayed. The maximum size of the field for each data type is defined below:
2	ADDRESS	
3	POINTER	



HEX VALUE	MEANING
00	Output only, input not allowed.
01	Field allows operator input.
02	The data displayed or input is in hexadecimal format.
04	Linked message. One or more additional messages.
0A	The data displayed or input is in binary format.
20	The data displayed or input is in decimal format.
40	The data displayed or input is signed (+ or -).
80	Last entry of the Message List.

Any of the above may be ORed together.

DATA TYPE	SIZE OF FIELD
ASCII	Number of characters to be displayed.
Hexa-decimal	Number of digits to be displayed.
Binary	Number of bits to be displayed.
Decimal	1 to 5 decimal digits.

(See section 9.3 also.)

Message List description (continued):

BYTE	MESSAGE LIST ENTRY	DESCRIPTION
0	OPCODE	Address of the associated information. (High byte, low byte.) The types of information pointed to are: ASCII MESSAGE LINKED MESSAGE LIST DATA STORED IN SHADOW RAM
1	FIELD LENGTH	
2	ADDRESS	
3	POINTER	

Figure 9-3 shows an example of a simple ASCII message.

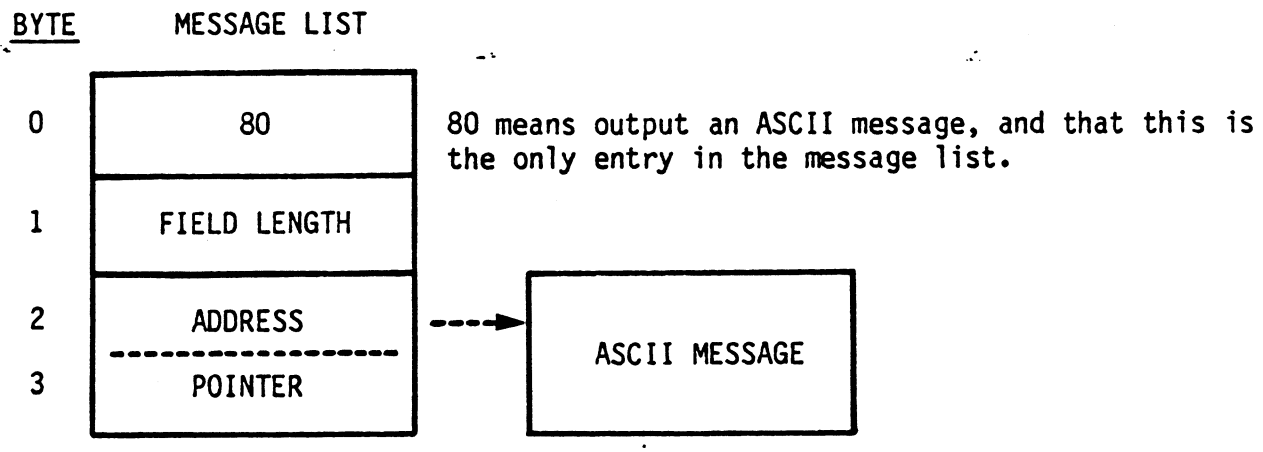


Figure 9-3. Simple ASCII Messages

Figure 9-4 shows a Message List with an ASCII message and an hexadecimal data input storage area.

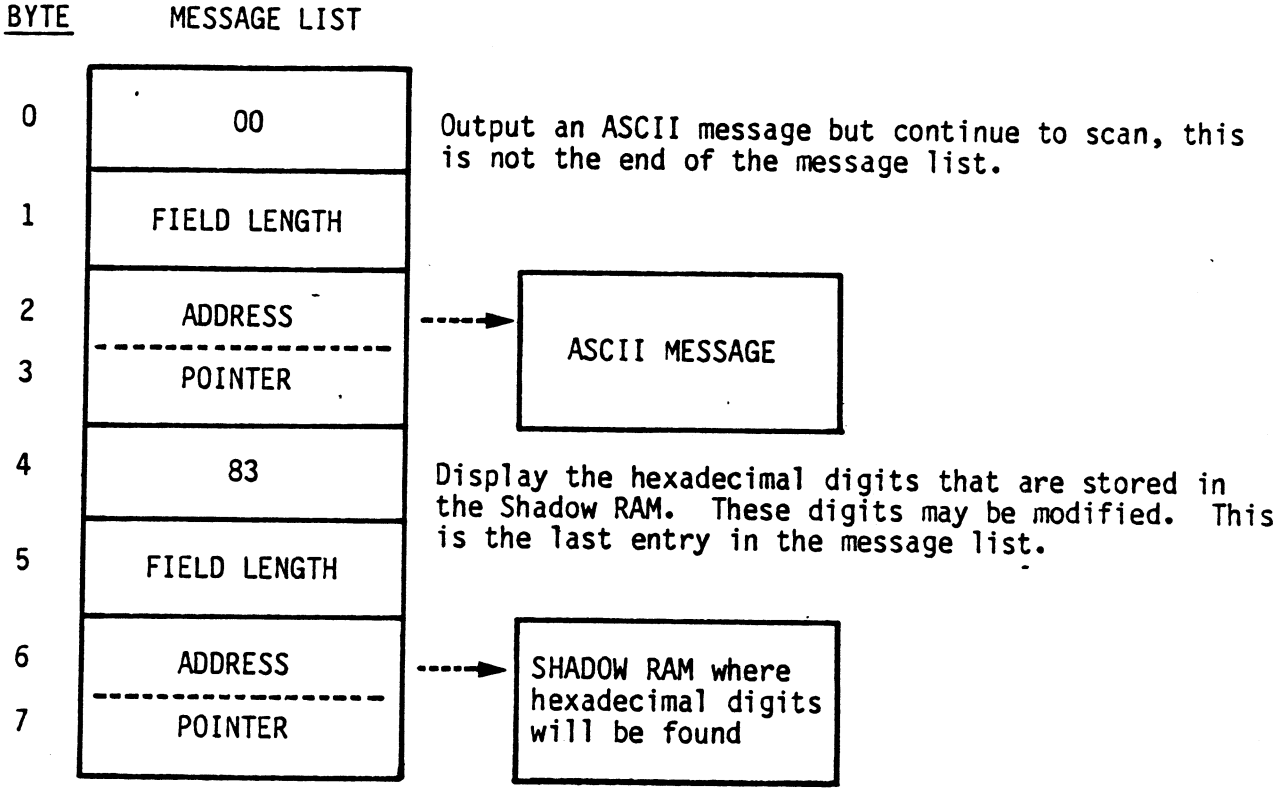


Figure 9-4. ASCII Message and Hexadecimal Data

The last entry on a Message List must always be designated as last, even if the last entry is a link entry. See figure 9-5.

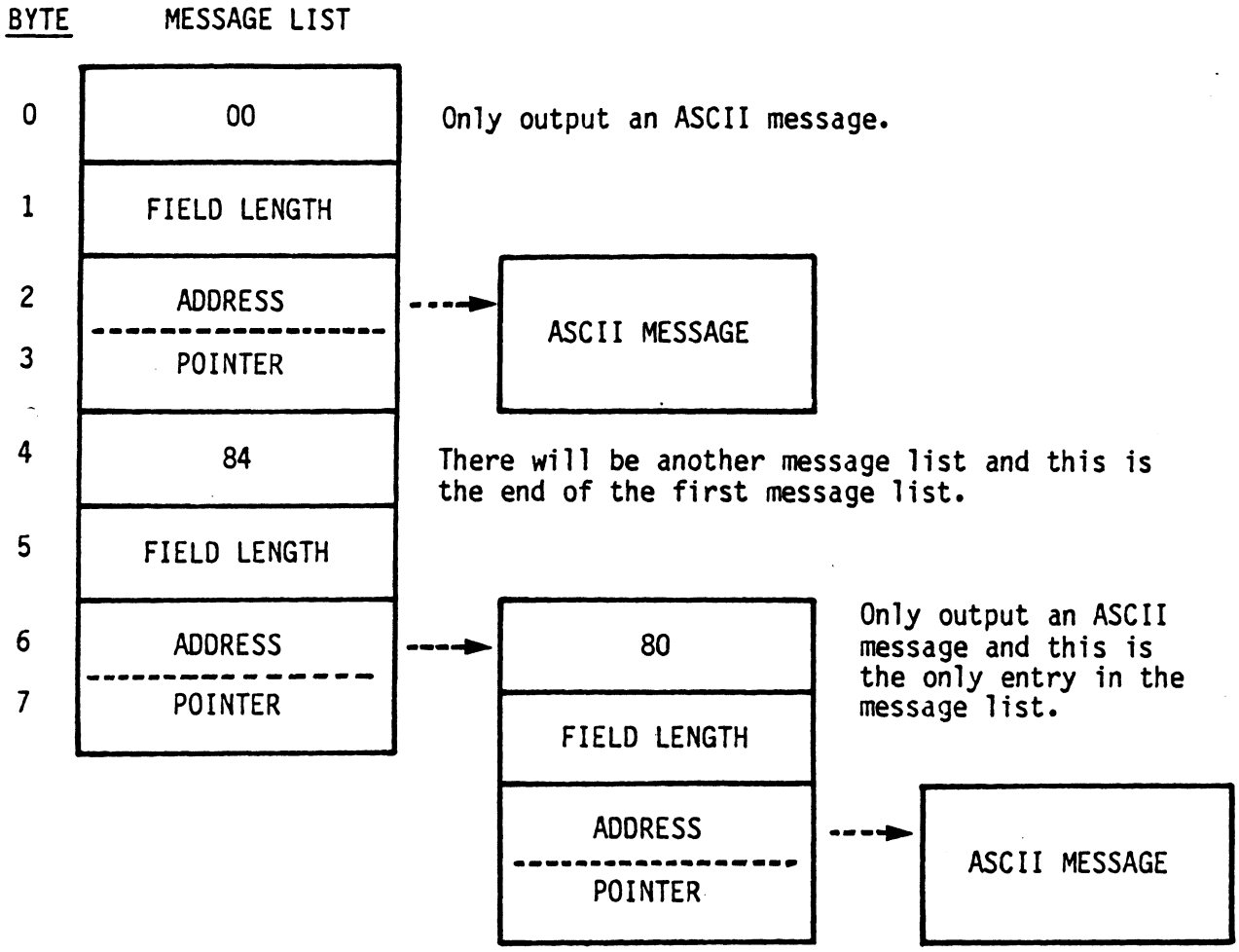


Figure 9-5. Linked List of Two ASCII Messages

To view the second ASCII message, the operator would push the INCR key.

The last Message List of a group of linked Message Lists may also be a linked list and may point to the first (or any other) Message List. This allows the operator to continually "scroll" a number of messages. An example of this is the REGISTER display.

9.3 DATA FORMATS

The internal format of data associated with Message Lists is in either ASCII or hexadecimal format. The internal data types carried in binary are as follows:

9.3.1 Hexadecimal

Each hexadecimal character is represented internally by four bits. The number of characters allowed is limited by the display length and is specified by the Field Length entry of the Message List. The data is carried right-justified internally. In the case of an odd number of hexadecimal characters the most significant nibble (4 bits) of the internal field is set to zero by the uSA.

9.3.2 Binary

Binary is a variation of the hexadecimal format, in that, before the BINARY key is pressed, the data is input/output in hexadecimal format. The data is still right-justified hexadecimal internally. The Length Field entry of the Message List must specify the number of binary characters. The number of characters displayed for binary will be the nearest MOD 4 number. For example, if a 10-bit binary field is specified, then three hexadecimal characters will be displayed. If a hexadecimal character is entered that exceeds the specified binary field, then it is treated as an illegal input character and is not accepted.

Binary data is always displayed in the last "n" character position of the display (right-justified). Therefore, a binary field must be the last field of display.

9.3.3 Decimal

Decimal data is converted to hexadecimal when the ENTER key is pressed. Decimal data is limited to one to five characters (0 - 65,535) in length and is always stored in a two-byte field (right-justified) as an unsigned 16-bit hexadecimal number. When a decimal field is to be displayed, it is converted from hexadecimal format to decimal format. If the value exceeds the range of digits specified in the message list, "ERR" will be displayed (right-justified) and a warning beep will be sounded.

9.3.4 Signed

A signed field is stored internally as an ASCII plus (+) or minus (-).

Appendix A

ASCII CHARACTER SET

Table A-1 contains the 7-bit ASCII hexadecimal code for each character that can be viewed on the uSA front panel display.

Table A-1. ASCII Character Set in Hexadecimal Representation

7-bit Hexadecimal Number	Character	7-bit Hexadecimal Number	Character
20	SPACE	40	@
21	!	41	A
22	RIGHT DOUBLE "	42	B
23	ALL SEGMENTS ON	43	C
24	$	44	D
25	%	45	E
26	MU	46	F
27	'	47	G
28	(	48	H
29	)	49	I
2A	*	4A	J
2B	+	4B	K
2C	,	4C	L
2D	-	4D	M
2E	.	4E	N
2F	/	4F	O
30	0	50	P
31	1	51	Q
32	2	52	R
33	3	53	S
34	4	54	T
35	5	55	U
36	6	56	V
37	7	57	W
38	8	58	X
39	9	59	Y
3A	:	5A	Z
3B	;	5B	Σ
3C	°	5C	\
3D	=	5D	]
3E	LEFT DOUBLE "	5E	->
3F	?	5F	<-

4.3 PROCESSOR CONTROL

The Processor Control keys listed and described below control the operation of the UUT:

RUN
RUN/DISP
STEP
RESET
FILL

4.3.1 RUN Key

This key starts execution at the address contained in the program counter status register PN. The PN can be changed using the REGISTER key. When the RUN key is pressed, the Run LED status indicator is illuminated.

4.3.2 RUN/DISP Key

In addition to the functions of the RUN key, the RUN/DISP key causes the selected Memory, Register, or I/O displays to be updated ten times per second when they are selected.

4.3.3 STEP Key

This key enables the operator to single-step program execution. Each time the step key is pressed, one instruction is executed. If any of the Memory, Register, or I/O displays are active, they are updated after each instruction is executed. This key is also used to stop execution when RUN or RUN/DISP is active.

4.3.4 RESET Key

This key forces a hardware Reset signal to the emulator microprocessor. The signal occurs once each time the key is pressed. This key does not effect the UUT. The processor registers are not altered by the reset signal. However, if interrupts are enabled, they will be reset.

4.4 DIAGNOSTIC SELECT KEYS

Diagnostic Select keys listed and described below are used to map a PROM into the UUT memory and to select subtests from the PROM:

PROM/MEM
SUBSEL
OPT/MEM KEY

4.4.1 PROM/MEM Key

This key maps diagnostic PROMs located on the keyboard into the UUT memory space. The PROM Memory Enable LED status indicator is illuminated and the initial message is displayed. The subtest is initialized to the first subtest. When the PROM/MEM key is pressed the operator sees

initialize message

and the subfunction INCR is enabled. The message that is displayed is the diagnostic initialization message as defined by the Diagnostic PROM Header. If the Message Pending LED status indicator is on, the INCR key is used to view multi-line messages. After the PROM/MEM key is pressed, the REGISTER key can be used to find the PROM origin, since the Program-Counter Last (PL) contains the starting address of the PROM overlay.

4.4.2 SUBSEL Key

The SUBSEL key selects one of 99 possible subtests as defined in the Diagnostic Header (see section 6.1). Selecting Subsel=00 causes all the diagnostic subtests to be executed. Selecting subroutine "n" causes only the designated subtest to execute. If the subtest number selected does not exist in the program, an input error will occur. This will cause the Input Error LED status indicator to light and will cause an audio error alert to be sounded.

When the SUBSEL key is pressed the operator sees

SUBSELECT=dd

dd = Subtest number in decimal.

and the subfunction Decimal keypad is enabled. After the decimal number is entered, press the ENTER key to terminate the input mode.

4.4.3 OPT/MEM Key

The OPT/MEM key is used by the Communication Option. See the Communication Option manual for details.

PMDS-II kommandoer.

Login: hold3 Password: (hemmelig)	login under holdnummer med (hemmelig) password.
$ CTRL d	logout fra systemet.
$ l $ ls -l	list directory (vis filer) kort liste. " " " " lang liste.

$ mv glnavn nynavn
$ cp glnavn nynavn
$ rm navn
	rename.
	kopier fil.
	slet fil.

$ cat navn $ lpr navn	vis fil på skærm. udskriv fil på printer.
NO SCROLL	en udskrift af tekst, der fylder mere end 24 linier på skærmen, stoppes ved at trykke NO SCROLL tasten og fortsættes ved at trykke NO SCROLL tasten igen.
DEL	enhver aktivitet kan afbrydes med DEL tasten.
$ e navn	editer filen "navn" (se næste side).

$ m80 navn
$ m80 navn -d1700
$ m80 navn -d1700|lpr
	assembler filen "navn".
	assembler og vis assembler-dokumentation på skærm.
	assembler og udskriv assembler-dokumentation på printer.

$ debug (nr)	debug (afprøv) program, (nr) = 0 , 1 , 2 ell. 3 (se sider om program afprøvning).
$ sprom proc16 $ sprom proc32	programmer EPROM type 2716. " " " 2732A.
p: l a.out p: p stadr sladr epradr p: d p: q	hent filen "a.out". programmer fra startadr til slutadr i EPROM startende fra adr epradr. verificer programmering. afslut programmering.

Philips Screen Editor

ikke en word processer

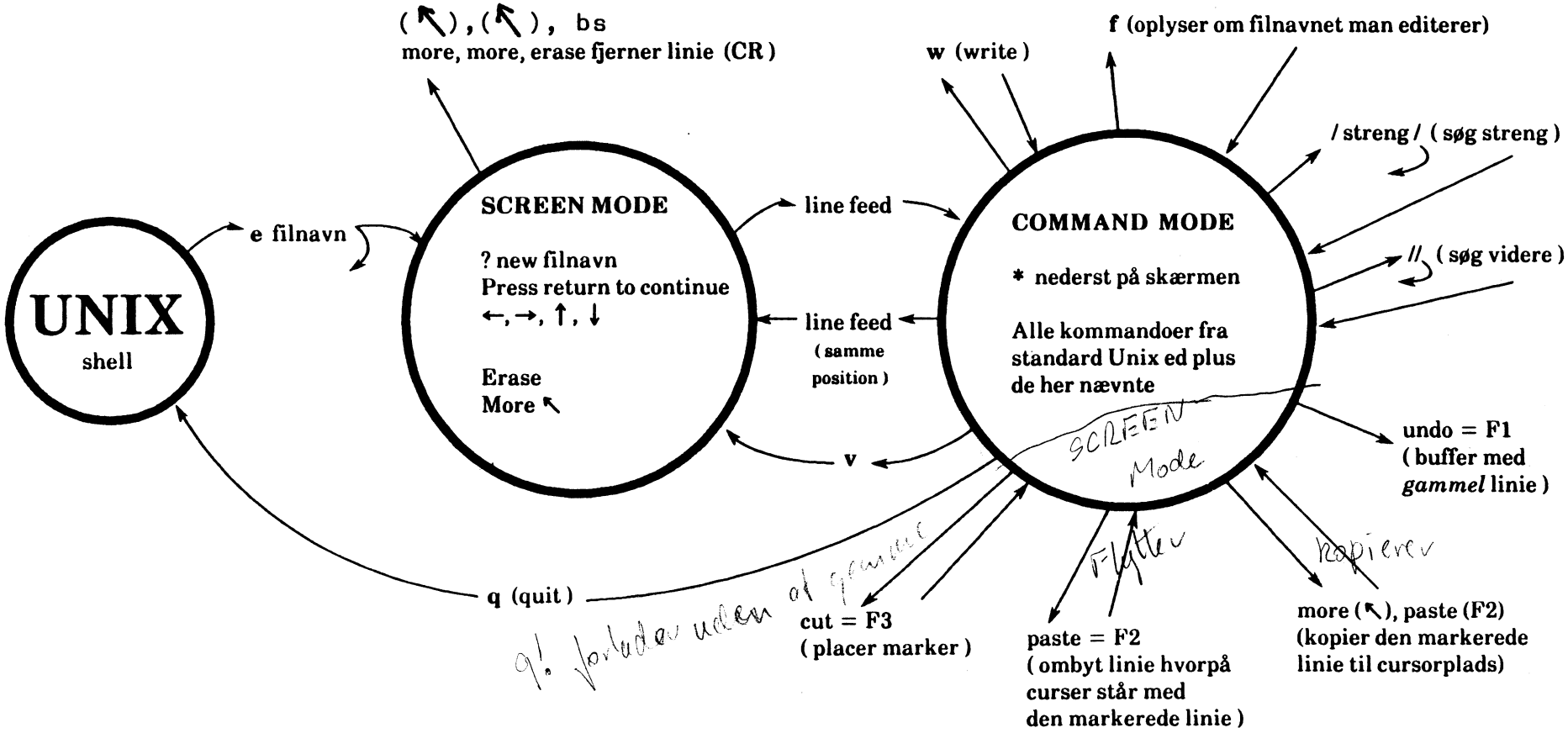
24 linier à 80 karakterer maks. 160

ESC
BREAK
DELETE
SET UP
} Not used by editor
(difficult to restoreyour text)

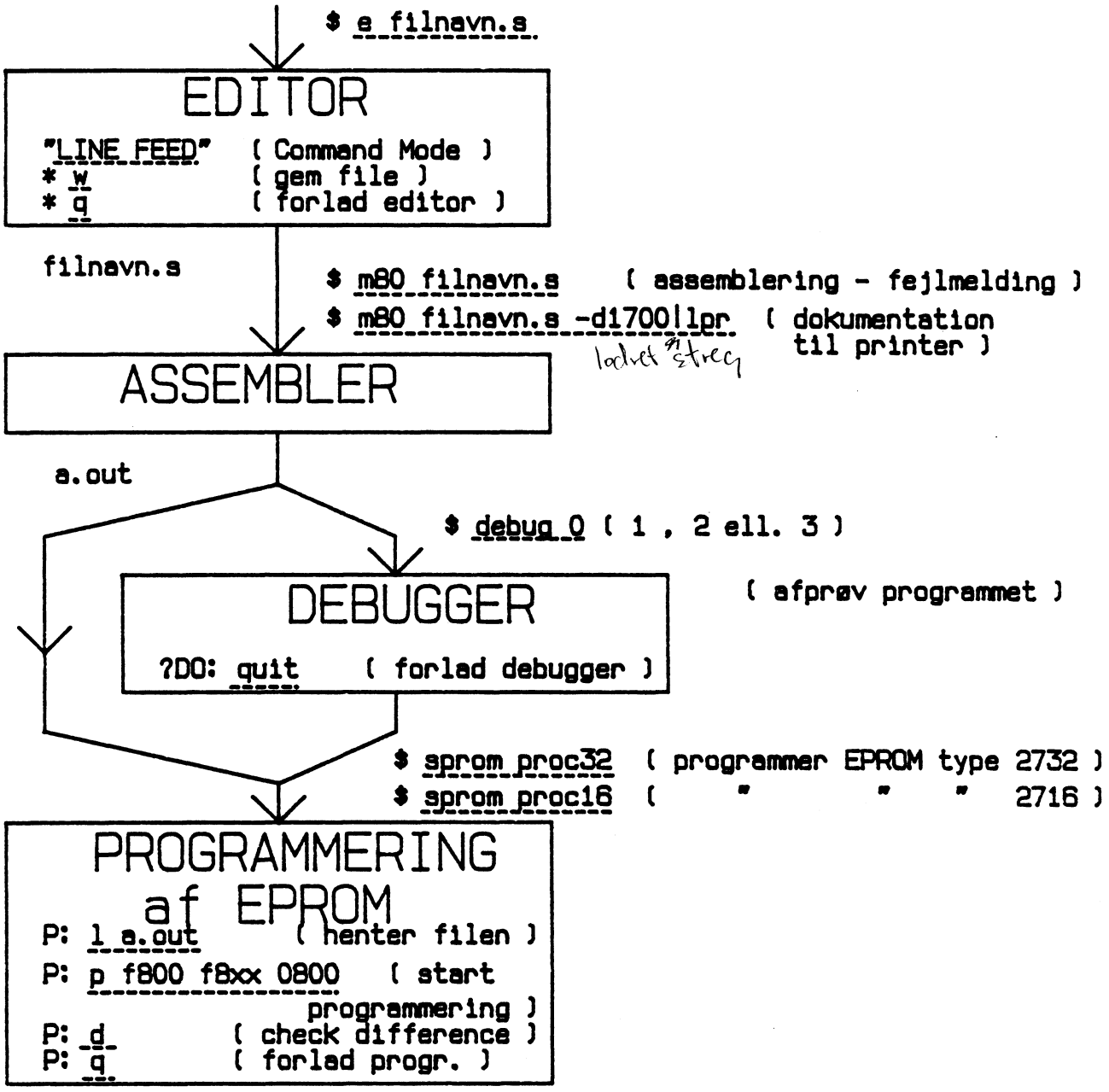
Philips TN100 Terminal

Erase = backspace
Command = line feed
Undo = F1
Paste = F2
Cut = F3

Cursor: ↑ ↓ → ←: enkelttegn eller linie
more, (↑ ↓ → ←): flere tegn (ord) eller linier

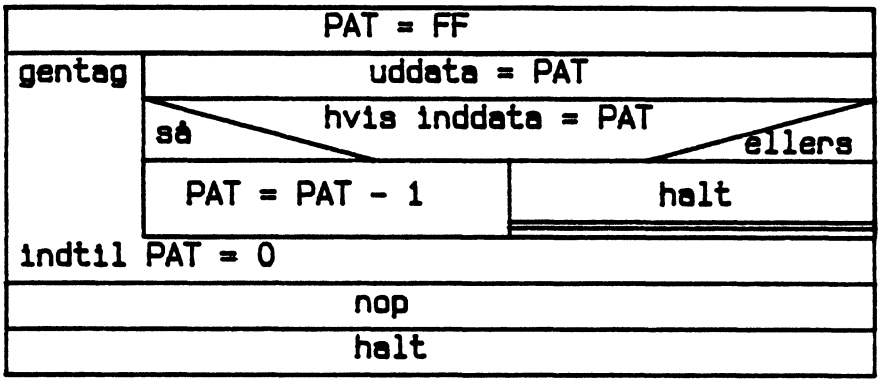


Program-udvikling.



```
                        !!!!!!  Program: demo  !!!!!!
                        !!!!!!!!!!!!!!!!!!!!!!!!!!!!!
                        ! Program til demonstration af forskel-
                        ! lige faciliteter og kommandoer i de-
                        ! buggeren.
                        !-----------------------------------------
                                ! Definitioner:

0058                    uddata  =       0x58    ! output-port.
005A                    inddata =       0x5a    ! input-port.
00FF                    PAT     =       0xff

                                .sect   .text   ! startadresse
                                .base   0xf800  ! for program.

F800 06FF                       ld  b,PAT       ! PAT = FF
F802 78                 loop:   ld  a,b
F803 D358                       out (uddata),a  ! output PAT

F805 DB5A                       in  a,(inddata) ! er input = PAT?
F807 B8                         cp  b
F808 CA0CF8                     jp  z,videre    ! saa fortsaet

F80B 76                         halt            ! ellers stop progr.

F80C 05                 videre: dec b           ! PAT = PAT - 1.

F80D C202F8                     jp  nz,loop     ! gentag "loop"
                                                ! indtil PAT = 0.
F810 00                         nop
F811 76                         halt            ! afslut program.
```

demo



DEBUGGER.

Efter assemblering af sourcefile med kommandoen:
("demo.s" erstattes af det aktuelle filnavn)

```
m80 demo.s
```

kan programmet afproeves i debuggeren, der startes med kommandoen:

```
debug x
```

x erstattes af 0 , 1 , 2 eller 3 afhaengig af hvilken debugger der skal arbejdes paa (eller hvilken der er ledig).

I opkaldet ligger der oplysning om memory og in-out map; denne opsaetning vises paa displayet:

```
 Log : opened     date : 86-04-11 time : 10:45:50
?map F800 FFFF emem 0 rom
?map 0 7FF emem 1000 ram
?imap 58 5C /ci
?imap CB /ci
?omap 58 5E z80pio:s
?omap F0 F1 lysdio:s
```

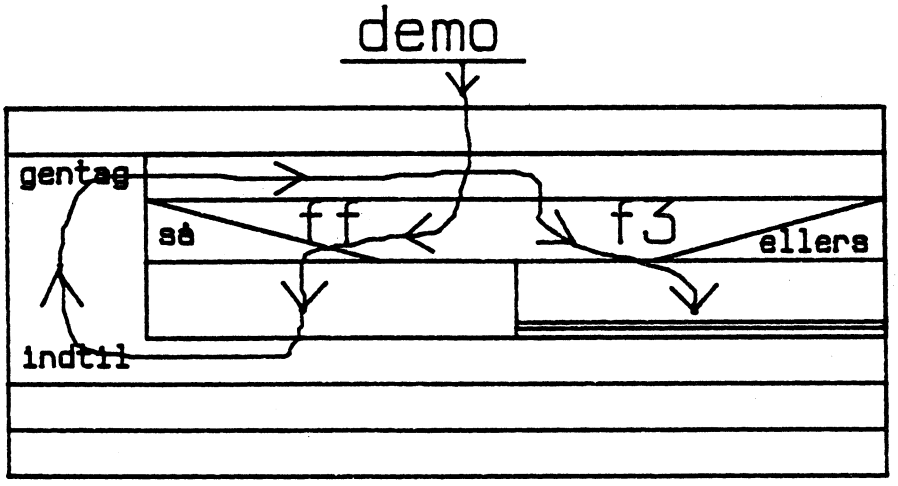
Ved "imap" betyder "/ci" at der anmodes om input fra tastaturet.
"omap 58 5E z80pio:s" betyder at data til portnr. omraadet skrives ud i en fil "z80pio:s". I denne fil kan man senere gaa ind og kontrollere de udsendte data.

Programmet loades og de foerste 16 koder vises disassembleret:

```
?load target
 D0     target:m LOADED, START ADDRESS = 0000
?mem F800 F810 disassemble
 D0     DISASSEMBLY
 ADDR   DATA      OPC   OPERAND
 F800   06FF      LD    B,H'FF'
 F802   78        LD    A,B
 F803   D358      OUT   (H'58'),A
 F805   DB5A      IN    A,(H'5A')
 F807   B8        CP    B
 F808   CA0CF8    JP    Z,H'F80C'
 F80B   76        HALT
 F80C   05        DEC   B
 F80D   C202F8    JP    NZ,H'F802'
 F810   00        NOP
?D0:
```

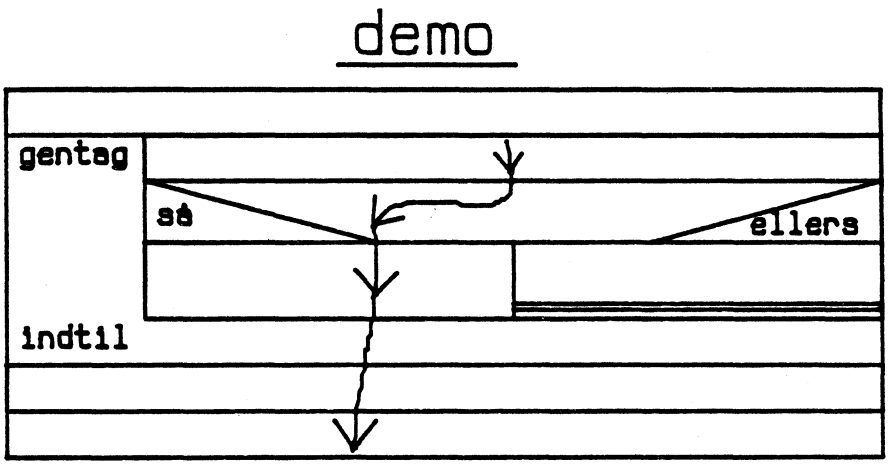
Herefter kan de forskellige kommandoer indtastes.

```
[   Eksempel paa afproevning af program "demo"  ]

[   ********* betyder indtastning fra keyboard  ]


[   Kommando for start af emulering fra adresse F800  ]
[   med ny indspilling i trace  ]

?DO: run f800 new
     ************
?
 DO     EMULATION STARTED
 DO     F803  D358     OUT (H'58'),A   OUTPUT-PORT 58 = FF  TO  z80pio:s
?
 DO     F805  DB5A     IN  A,(H'5A')    INPUT-PORT 5A = ff
                                                      **
                  [  med "ff" proeves "hvis-saa"  ]
?
 DO     F803  D358     OUT (H'58'),A   OUTPUT-PORT 58 = FE  TO  z80pio:s
?
 DO     F805  DB5A     IN  A,(H'5A')    INPUT-PORT 5A = f3
                                                      **
                  [  med "f3" proeves "hvis-ellers"  ]

[  emuleringen stopper med melding om aarsagen: ( HALT INSTRUCTION )  ]

?
 DO     EMULATION HALTED       : HALT INSTRUCTION

 [  sidst koerte instruktion og register-indhold vises:  ]

 DO          DISASSEMBLY   REGISTERS     FFFFFF
                           PC    SP    A  SZHPNC  B  C  D  E  H  L  IF RA
 ADDR DATA   OPC OPERAND   IX    IY  A' AA SZHPNC B' BB CC' DD' EE' HH' LL' IM' IV'

                           F80C  82BE  F3 101011  FE 00 03 04 F8 15  0 57
 F80B 76     HALT          EFBF  FFDD  00 000001  FF FF FD FF EF F9  0 00
```



```
[  Er der problemer undersoeges programmet naermere:  ]
[  med kommandoen "trace -19 b vises  ]
[  de 20 sidst koerte maskin-cycles:  ]

?DO: trace -19 b
     ***********

 DO     TRACE-MEMORY
 LINE   ADDR  DATA     OPC  OPERAND          CNTRL       ECTL PODO  POD1
 - 19   F807  B8       CP   B                MR  OPC     0000
 - 18   F808  CA0CF8   JP   Z,H'F80C'        MR  OPC     0000
 - 17   F809  0C                            MR          1000
 - 16   F80A  F8                            MR          1000
 - 15   F80C  05       DEC  B                MR  OPC     0000
 - 14   F80D  C202F8   JP   NZ,H'F802'       MR  OPC     0000
 - 13   F80E  02                            MR          1000
 - 12   F80F  F8                            MR          1000
 - 11   F802  78       LD   A,B              MR  OPC     0000
 - 10   F803  D358     OUT  (H'58'),A        MR  OPC     0000
 -  9   F804  58                            MR          1000
 -  8   FE58  FE                            IOW         1000
 -  7   F805  DB5A     IN   A,(H'5A')        MR  OPC     0000
 -  6   F806  5A                            MR          1000
 -  5   FE5A  F3                            IOR         1000
 -  4   F807  B8       CP   B                MR  OPC     0000
 -  3   F808  CA0CF8   JP   Z,H'F80C'        MR  OPC     0000
 -  2   F809  0C                            MR          1000
 -  1   F80A  F8                            MR          1000
    0   F80B  76       HALT                  MR  OPC     0000

 [ med kommandoen "step f800 jump forever" koeres der singlestep paa ]
 [ alle betingede jump-instruktioner, og register-indholdet vises:   ]

?DO: step f800 jump forever
     **********************

 DO     EMULATION STARTED
 DO     F803  D358   OUT (H'58'),A    OUTPUT-PORT 58 = FF  TO  z80pio:s
 DO     F805  DB5A   IN  A,(H'5A')     INPUT-PORT 5A = ff
 DO           DISASSEMBLY    REGISTERS    FFFFFF       **
                             PC   SP   A  SZHPNC  B  C  D  E  H  L  IF RA
 ADDR DATA    OPC OPERAND    IX   IY   AA SZHPNC  BB CC DD EE HH LL IM IV

                             F80C 82BE>FF 010010<>FF 00 03 04 F8 0F< 0 2E
 F808 CA0CF8  JP Z,H'F80C'   EFBF FFDD 00 000001  FF FF FD FF EF F9  0 00

                             F802 82BE>FF 100010<>FE 00 03 04 F8 0F< 0 13
 F80D C202F8  JP NZ,H'F802'  EFBF FFDD 00 000001  FF FF FD FF EF F9  0 00

 DO     F803  D358   OUT (H'58'),A    OUTPUT-PORT 58 = FE  TO  z80pio:s
 DO     F805  DB5A   IN  A,(H'5A')     INPUT-PORT 5A = f3
                                                    **
                             F80B 82BE>F3 101011<>FE 00 03 04 F8 0F< 0 4D
 F808 CA0CF8  JP Z,H'F80C'   EFBF FFDD 00 000001  FF FF FD FF EF F9  0 00

 DO     EMULATION HALTED      : HALT INSTRUCTION
                             F80C 82BE>F3 101011<>FE 00 03 04 F8 0F< 0 02
 F80B 76      HALT           EFBF FFDD 00 000001  FF FF FD FF EF F9  0 00
```

```
[  "gentag indtil" sloejfen afproeves:     ]
[  med kommandoen "register b=01" aendres  ]
[  indholdet af register B til "01":       ]

?DO: register b=01
     ************   Sætter værdi i Register

[  programmet startes fra adresse F802:    ]

?DO: run f802 new
     ************   starter fra adr. F802

?
 DO     EMULATION STARTED
 DO     F803  D358   OUT (H'58'),A     OUTPUT-PORT 58 = 01  TO  z80pio:s
?
 DO     F805  DB5A   IN  A,(H'5A')     INPUT-PORT 5A = 01
                                                      **
        [   med "01" vaelges "hvis-saa" vejen   ]
?
 DO     EMULATION HALTED       : HALT INSTRUCTION
 DO           DISASSEMBLY     REGISTERS     FFFFFF
                              PC   SP   A  SZHPNC  B  C  D  E  H  L  IF RA
 ADDR DATA    OPC  OPERAND    IX   IY   AA SZHPNC  BB CC DD EE HH LL IM IV

                              F812 82BE>01 010010<>00 00 03 04 F8 0F< 0 03
 F811 76      HALT            EFBF FFDD 00 000001  FF FF FD FF EF F9  0 00
```



```
[  de sidste maskincycler vises:   ]

?DO: trace -19 b
     **********

 DO     TRACE-MEMORY
 LINE   ADDR   DATA     OPC   OPERAND          CNTRL       ECTL PODO  POD1
 - 16   F802   78       LD    A,B              MR  OPC     0000
 - 15   F803   D358     OUT   (H'58'),A        MR  OPC     0000
 - 14   F804   58                              MR          1000
 - 13   0158   01                              IOW         1000
 - 12   F805   DB5A     IN    A,(H'5A')        MR  OPC     0000
 - 11   F806   5A                              MR          1000
 - 10   015A   01                              IOR         1000
 -  9   F807   B8       CP    B                MR  OPC     0000
 -  8   F808   CA0CF8   JP    Z,H'F80C'        MR  OPC     0000
 -  7   F809   0C                              MR          1000
 -  6   F80A   F8                              MR          1000
 -  5   F80C   05       DEC   B                MR  OPC     0000
 -  4   F80D   C202F8   JP    NZ,H'F802'       MR  OPC     0000
 -  3   F80E   02                              MR          1000
 -  2   F80F   F8                              MR          1000
 -  1   F810   00       NOP                    MR  OPC     0000
    0   F811   76       HALT                   MR  OPC     0000


 [  med kommandoen "step f800 all 3" single-steppes  ]
 [  der 3 instruktioner, mens register-indhold vises: ]

?DO: step f800 all 3
     ***************

 DO     EMULATION STARTED
 DO             DISASSEMBLY      REGISTERS     FFFFFF
                                 PC   SP   A   SZHPNC  B  C  D  E  H  L  IF RA
 ADDR DATA      OPC OPERAND      IX   IY   AA  SZHPNC  BB CC DD EE HH LL IM IV

                                 F802 82BE>01 010010<>FF 00 03 04 F8 0F< 0 25
 F800 06FF      LD  B,H'FF'      EFBF FFDD 00  000001  FF FF FD FF EF F9  0 00

                                 F803 82BE>FF 010010<>FF 00 03 04 F8 0F< 0 73
 F802 78        LD  A,B          EFBF FFDD 00  000001  FF FF FD FF EF F9  0 00

 DO     F803   D358     OUT (H'58'),A    OUTPUT-PORT 58 = FF  TO   z80pio:s
                                 F805 82BE>FF 010010<>FF 00 03 04 F8 0F< 0 1B
 F803 D358      OUT (H'58'),A    EFBF FFDD 00  000001  FF FF FD FF EF F9  0 00


 [  med kommandoen "quit" forlades debuggeren:   ]

?DO: quit
     ****
```

Keybuffer-test

tænd LED1 test startet				
GENTAG	init. PIO: A (STROBE) og B (PORT) = ud			
	(STROBE) ← high			
	start signatur-måling			
	DATAUD = 0			
	DATAIND = 0			
	TEST = 0			
	GENTAG	LOOP = 8		
		GENTAG	(PORT) ← DATAUD	
			(STROBE) ← ⊓	
			DATAUD = DATAUD + 1	
			LOOP = LOOP - 1	
		INDTIL: LOOP = 0		
		test (BUFFER) = FF		
		LOOP = 8		
		GENTAG	test (BUFFER) = DATAIND	
			DATAIND = DATAIND + 1	
			LOOP = LOOP - 1	
		INDTIL: LOOP = 0		
	INDTIL: DATAUD = 0			
	stop signatur-måling			
	læs (BUFFER)			
	så HVIS TEST = ok			ellers
	tænd LED2	sluk LED2		

0 - 256

ingen start stop puls til signatur mete.

? Bufcode1.s

? Bufcode.s

Zilog

Product Specification

Z-80® CPU
Z-80A CPU

The Zilog Z80 product line is a complete set of microcomputer components, development systems and support software. The Z80 microcomputer component set includes all of the circuits necessary to build high-performance microcomputer systems with virtually no other logic and a minimum number of low cost standard memory elements.

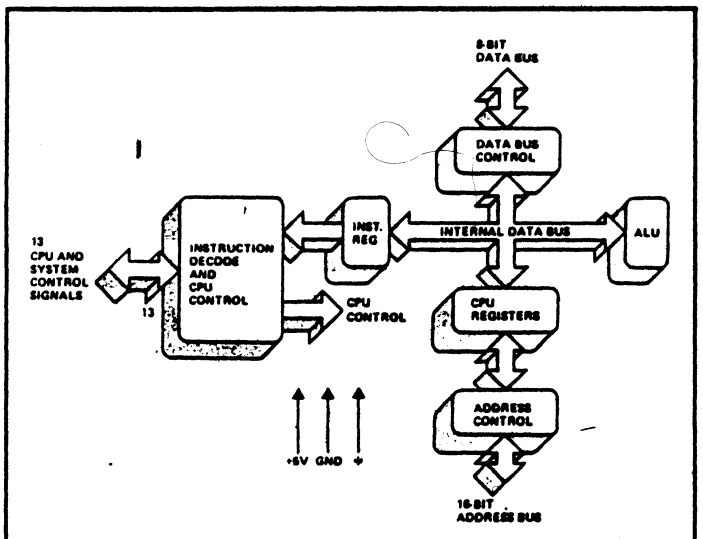
The Z80 and Z80A CPU's are third generation single chip microprocessors with unrivaled computational power. This increased computational power results in higher system through-put and more efficient memory utilization when compared to second generation microprocessors. In addition, the Z80 and Z80A CPU's are very easy to implement into a system because of their single voltage requirement plus all output signals are fully decoded and timed to control standard memory or peripheral circuits. The circuit is implemented using an N-channel, ion implanted, silicon gate MOS process.

Figure 1 is a block diagram of the CPU, Figure 2 details the internal register configuration which contains 208 bits of Read/Write memory that are accessible to the programmer. The registers include two sets of six general purpose registers that may be used individually as 8-bit registers or as 16-bit register pairs. There are also two sets of accumulator and flag registers. The programmer has access to either set of main or alternate registers through a group of exchange instructions. This alternate set allows foreground/background mode of operation or may be reserved for very fast Interrupt response. Each CPU also contains a 16-bit stack pointer which permits simple implementation of multiple level interrupts, unlimited subroutine nesting and simplification of many types of data handling.

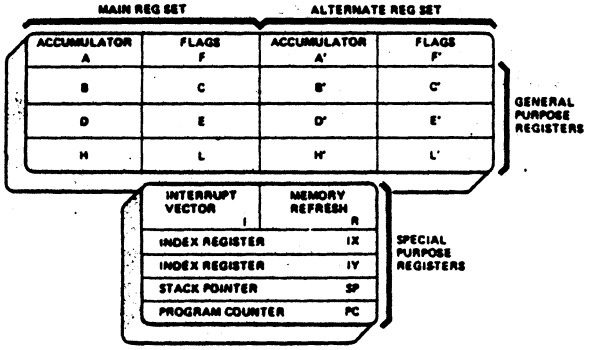
The two 16-bit index registers allow tabular data manipulation and easy implementation of relocatable code. The Refresh register provides for automatic, totally transparent refresh of external dynamic memories. The I register is used in a powerful interrupt response mode to form the upper 8 bits of a pointer to a interrupt service address table, while the interrupting device supplies the lower 8 bits of the pointer. An indirect call is then made to this service address.

FEATURES

- Single chip, N-channel Silicon Gate CPU.
- 158 instructions–includes all 78 of the 8080A instructions with total software compatibility. New instructions include 4-, 8- and 16-bit operations with more useful addressing modes such as indexed, bit and relative.
- 17 internal registers.
- Three modes of fast interrupt response plus a non-maskable interrupt.
- Directly interfaces standard speed static or dynamic memories with virtually no external logic.
- 1.0 μs instruction execution speed.
- Single 5 VDC supply and single-phase 5 volt Clock.
- Out-performs any other single chip microcomputer in 4-, 8-, or 16-bit applications.
- All pins TTL Compatible
- Built-in dynamic RAM refresh circuitry.



Z80, Z80A CPU BLOCK DIAGRAM



Z80, Z80A CPU REGISTERS

Zilog

Instruction Set

The following is a summary of the Z80, Z80A instruction set showing the assembly language mnemonic and the symbolic operation performed by the instruction. A more detailed listing appears in the Z80-CPU technical manual, and assembly language programming manual. The instructions are divided into the following categories:

8-bit loads	Miscellaneous Group
16-bit loads	Rotates and Shifts
Exchanges	Bit Set, Reset and Test
Memory Block Moves	Input and Output
Memory Block Searches	Jumps
8-bit arithmetic and logic	Calls
16-bit arithmetic	Restarts
General purpose Accumulator & Flag Operations	Returns

In the table the following terminology is used.

b ≡ a bit number in any 8-bit register or memory location

cc ≡ flag condition code
- NZ ≡ non zero
- Z ≡ zero
- NC ≡ non carry
- C ≡ carry
- PO ≡ Parity odd or no over flow
- PE ≡ Parity even or over flow
- P ≡ Positive
- M ≡ Negative (minus)

d ≡ any 8-bit destination register or memory location

dd ≡ any 16-bit destination register or memory location

e ≡ 8-bit signed 2's complement displacement used in relative jumps and indexed addressing

L ≡ 8 special call locations in page zero. In decimal notation these are 0, 8, 16, 24, 32, 40, 48 and 56

n ≡ any 8-bit binary number

nn ≡ any 16-bit binary number

r ≡ any 8-bit general purpose register (A, B, C, D, E, H, or L)

s ≡ any 8-bit source register or memory location

s_b ≡ a bit in a specific 8-bit register or memory location

ss ≡ any 16-bit source register or memory location

subscript "L" ≡ the low order 8 bits of a 16-bit register

subscript "H" ≡ the high order 8 bits of a 16-bit register

( ) ≡ the contents within the ( ) are to be used as a pointer to a memory location or I/O port number

8-bit registers are A, B, C, D, E, H, L, I and R

16-bit register pairs are AF, BC, DE and HL

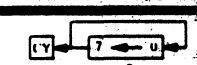
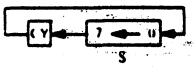
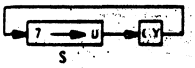
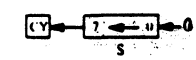
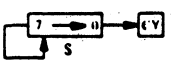
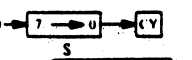
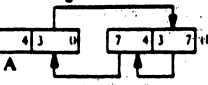
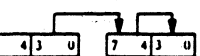
16-bit registers are SP, PC, IX and IY

Addressing Modes implemented include combinations of the following:

Immediate	Indexed
Immediate extended	Register
Modified Page Zero	Implied
Relative	Register Indirect
Extended	Bit

	Mnemonic	Symbolic Operation	Comments
8-BIT LOADS	LD r, s	r ← s	s ≡ r, n, (HL), (IX+e), (IY+e)
	LD d, r	d ← r	d ≡ (HL), r (IX+e), (IY+e)
	LD d, n	d ← n	d ≡ (HL), (IX+e), (IY+e)
	LD A, s	A ← s	s ≡ (BC), (DE), (nn), I, R
	LD d, A	d ← A	d ≡ (BC), (DE), (nn), I, R
16-BIT LOADS	LD dd, nn	dd ← nn	dd ≡ BC, DE, HL, SP, IX, IY
	LD dd, (nn)	dd ← (nn)	dd ≡ BC, DE, HL, SP, IX, IY
	LD (nn), ss	(nn) ← ss	ss ≡ BC, DE, HL, SP, IX, IY
	LD SP, ss	SP ← ss	ss = HL, IX, IY
	PUSH ss	$(SP-1) \leftarrow ss_H$; $(SP-2) \leftarrow ss_L$	ss = BC, DE, HL, AF, IX, IY
	POP dd	$dd_L \leftarrow (SP)$; $dd_H \leftarrow (SP+1)$	dd = BC, DE, HL, AF, IX, IY
EXCHANGES	EX DE, HL	DE ↔ HL	
	EX AF, AF'	AF ↔ AF'	
	EXX	$\begin{pmatrix} BC \\ DE \\ HL \end{pmatrix} \leftrightarrow \begin{pmatrix} BC' \\ DE' \\ HL' \end{pmatrix}$	
	EX (SP), ss	$(SP) \leftrightarrow ss_L$, $(SP+1) \leftrightarrow ss_H$	ss ≡ HL, IX, IY

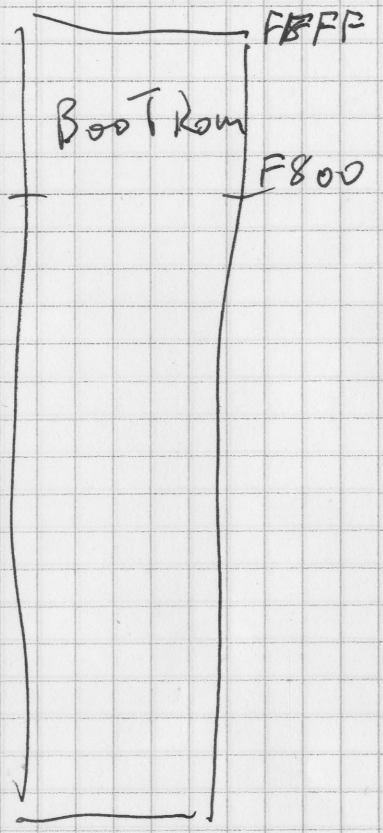
	Mnemonic	Symbolic Operation	Comments
MEMORY BLOCK MOVES	LDI	(DE) ← (HL), DE ← DE+1 HL ← HL+1, BC ← BC-1	
	LDIR	(DE) ← (HL), DE ← DE+1 HL ← HL+1, BC ← BC-1 Repeat until BC = 0	
	LDD	(DE) ← (HL), DE ← DE-1 HL ← HL-1, BC ← BC-1	
	LDDR	(DE) ← (HL), DE ← DE-1 HL ← HL-1, BC ← BC-1 Repeat until BC = 0	
MEMORY BLOCK SEARCHES	CPI	A-(HL), HL ← HL+1 BC ← BC-1	
	CPIR	A-(HL), HL ← HL+1 BC ← BC-1, Repeat until BC = 0 or A = (HL)	A-(HL) sets the flags only. A is not affected
	CPD	A-(HL), HL ← HL-1 BC ← BC-1	
	CPDR	A-(HL), HL ← HL-1 BC ← BC-1, Repeat until BC= 0 or A = (HL)	
8-BIT ALU	ADD s	A ← A + s	
	ADC s	A ← A + s + CY	CY is the carry flag
	SUB s	A ← A - s	
	SBC s	A ← A - s - CY	s ≡ r, n, (HL) (IX+e), (IY+e)
	AND s	A ← A ∧ s	
	OR s	A ← A ∨ s	
	XOR s	A ← A ⊕ s	

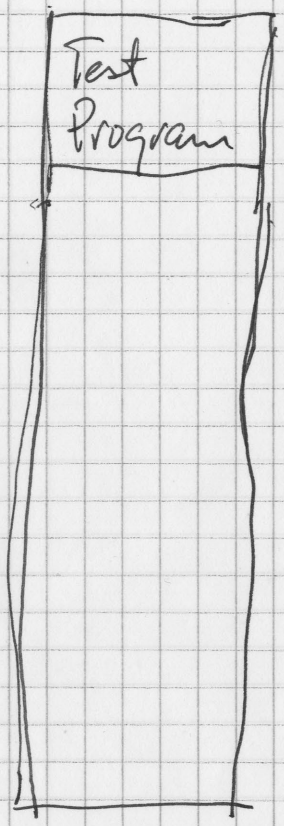
	Mnemonic	Symbolic Operation	Comments
8-BIT ALU	CP s	A - s	s = r, n (HL) (IX+e), (IY+e)
	INC d	d ← d + 1	d = r, (HL) (IX+e), (IY+e)
	DEC d	d ← d - 1	
16-BIT ARITHMETIC	ADD HL, ss	HL ← HL + ss	ss ≡ BC, DE HL, SP
	ADC HL, ss	HL ← HL + ss + CY	
	SBC HL, ss	HL ← HL - ss - CY	
	ADD IX, ss	IX ← IX + ss	ss ≡ BC, DE, IX, SP
	ADD IY, ss	IY ← IY + ss	ss ≡ BC, DE, IY, SP
	INC dd	dd ← dd + 1	dd ≡ BC, DE, HL, SP, IX, IY
	DEC dd	dd ← dd - 1	dd ≡ BC, DE, HL, SP, IX, IY
GP ACC. & FLAG	DAA	Converts A contents into packed BCD following add or subtract.	Operands must be in packed BCD format
	CPL	$A \leftarrow \overline{A}$	
	NEG	A ← 00 - A	
	CCF	$CY \leftarrow \overline{CY}$	
	SCF	CY ← 1	
MISCELLANEOUS	NOP	No operation	
	HALT	Halt CPU	
	DI	Disable Interrupts	
	EI	Enable Interrupts	
	IM 0	Set interrupt mode 0	8080A mode
	IM 1	Set interrupt mode 1	Call to 0038_H
	IM 2	Set interrupt mode 2	Indirect Call
ROTATES AND SHIFTS	RLC s		s ≡ r, (HL) (IX+e), (IY+e)
	RL s		
	RRC s		
	RR s		
	SLA s		
	SRA s		
	SRL s		
	RLD		
	RRD		

	Mnemonic	Symbolic Operation	Comments
BIT S, R, & T	BIT b, s	$Z \leftarrow \overline{s_b}$	Z is zero flag
	SET b, s	$s_b \leftarrow 1$	s ≡ r, (HL) (IX+e), (IY+e)
	RES b, s	$s_b \leftarrow 0$	
INPUT AND OUTPUT	IN A, (n)	A ← (n)	
	IN r, (C)	r ← (C)	Set flags
	INI	(HL) ← (C), HL ← HL + 1 B ← B - 1	
	INIR	(HL) ← (C), HL ← HL + 1 B ← B - 1 Repeat until B = 0	
	IND	(HL) ← (C), HL ← HL - 1 B ← B - 1	
	INDR	(HL) ← (C), HL ← HL - 1 B ← B - 1 Repeat until B = 0	
	OUT(n), A	(n) ← A	
	OUT(C), r	(C) ← r	
	OUTI	(C) ← (HL), HL ← HL + 1 B ← B - 1	
	OTIR	(C) ← (HL), HL ← HL + 1 B ← B - 1 Repeat until B = 0	
	OUTD	(C) ← (HL), HL ← HL - 1 B ← B - 1	
	OTDR	(C) ← (HL), HL ← HL - 1 B ← B - 1 Repeat until B = 0	
JUMPS	JP nn	PC ← nn	cc ≡ NZ PO, Z PE, NC P, C M
	JP cc, nn	If condition cc is true PC ← nn, else continue	
	JR e	PC ← PC + e	kk ≡ NZ NC, Z C
	JR kk, e	If condition kk is true PC ← PC + e, else continue	
	JP (ss)	PC ← ss	ss = HL, IX, IY
	DJNZ e	B ← B - 1, if B = 0 continue, else PC ← PC + e	
CALLS	CALL nn	$(SP-1) \leftarrow PC_H$ $(SP-2) \leftarrow PC_L$, PC ← nn	cc ≡ NZ PO, Z PE, NC P, C M
	CALL cc, nn	If condition cc is false continue, else same as CALL nn	
RESTARTS	RST L	$(SP-1) \leftarrow PC_H$ $(SP-2) \leftarrow PC_L$, $PC_H \leftarrow 0$ $PC_L \leftarrow L$	
RETURNS	RET	$PC_L \leftarrow (SP)$, $PC_H \leftarrow (SP+1)$	cc ≡ NZ PO, Z PE, NC P, C M
	RET cc	If condition cc is false continue, else same as RET	
	RETI	Return from interrupt, same as RET	
	RETN	Return from non-maskable interrupt	

MPS 27 Boot Rom 2732

IC12





Adr 800

Prom i 28 Bens sokkel længst nede

jump F803

jump start

start:

$ sprom proc-32 2732 husk at forlade

L load L q, out

+ ram indhold

P program F800 F8XX afset 0800

b Blank check b

d diferens check F800 F8XX 0800

q quit forlader sprom program

8^{00} – 9^{30} fys på Lundin

9^{50} – 11^{15}

12^{00} – 13^{30} 5827.01.91

13^{50} – 15^{15}

Test programmer bør ikke bruge ram med mindre Ram er testet og fundet i orden

0x15 Hex 0x

oct 0

ingen parantes talværdi

parantes (indehold af det der peges på)

Slut altid testprogram med Halt og kan da kun opstartes med reset eller interrupt

MREG ORG

ld out hvor (register til index

ld ind

0x Hexadecimal

0 octal

uden Decimal

Dokumenter og afprøv Testprogrammer grundigt

Hold 3 | password password

L

ls - l

W space Nyt filnavn

Q quit forlader Debugger

Subrutine uden Call og Return

Sub

ld ix iy HL ret

jump

sub

–
–
–
–
–
–

jmp (ix)

E 84 C |

|||| || |

Port Test Program

MPS 23

Tænd Lysdiode 1

Nåleprøve 1

Initialicer Port PC3 = UD
IC5 = IND

MPS10
Porttest.S

Sæt Data = 00

Gentag

Send Data til output

Hent input

Hvis input = Data

SÅ

ellers

Data = Data+1

Halt

indtil Data = FF

Tænd Lysdiode 2 (OK)

Nåleprøve 2

Initialice P10 Port

IC3 = Ind
IC5 = Ud

Sæt Data = 00

Gentag

Send Data til output

Hent Data til input

Hvis input = Data

SÅ

ellers

DATA = DATA+1

Halt

indtil Data = FF

Nåleprøve 3

Tænd lysdiode 3

HALT 3