ASSEMBLY
PROGRAMMING
8080/8085
en INTERFACING

AP map 2.

INTERRUPT.

Interrupt

1. INLEIDING

In deze les worden de voorwaarden en mogelijkheden voor het werken met interrupts behandeld. Er wordt eerst ingegaan op de interrupt-mogelijkheden van de afzonderlijke 8080- en 8085-microprocessoren. Daarna bespreken we de interrupt-mogelijkheden, die in de SDK 85 door de gebruiker zijn toe te passen.

We gaan er in deze les van uit, dat u de stof uit de les "I/O-Interfacing" van de cursus "Microprocessors/Microcomputers" beheerst. Lees deze les, of het overeenkomstige hoofdstuk (21) uit het boek, daarom nog eens nauwkeurig door.

2. WAT IS INTERRUPT ?

In elke computertoepassing is het noodzakelijk, dat er communicatie (= uitwisseling van informatie) tussen computer en buitenwereld optreedt. De computer geeft niet alleen informatie en besturingssignalen op de uitgangen af, maar moet ook rekening houden met signalen, die van de buitenwereld (b.v. een proces of een toetsenbord) afkomstig zijn.

Het initiatief tot een uitwisseling van informatie kan zowel van de computer als van de buitenwereld uitgaan. Als de computer het tijdstip van informatie-uitwisseling bepaalt, dan spreken we van geprogrammeerde I/O (I/O = Input/Output = uitwisseling van informatie). Hierbij test de computer steeds de status van de buitenwereld, om te bepalen of deze in staat is tot het plegen van I/O.
Komt het startsein voor een I/O-handeling van de buitenwereld, dan spreken we interrupt I/O. De computer is dan bezig met de uitvoering van een programma.
Wanneer een interrupt-signaal van de buitenwereld wordt ontvangen, dan wordt de programma-uitvoering onderbroken t.b.v. een informatie-uitwisseling.

> Vraag 1: Bij geprogrammeerde I/O wordt de informatie-uitwisseling gestart door de computer/buitenwereld.
> Bij interrupt I/O wordt de informatie-uitwisseling gestart door de computer/buitenwereld.

a. Geprogrammeerde I/O

Bij geprogrammeerde I/O moet de computer, op het moment dat de informatie-uitwisseling zou moeten beginnen, eerst de statussignalen testen (fig. 1) om te bepalen of ook de buitenwereld in staat is tot het plegen van I/O.

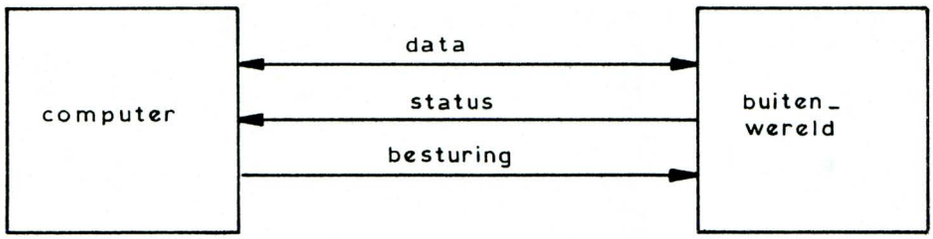


fig.1

Het testen van de statussignalen gebeurt onder besturing van instructies, die deel uitmaken van het werkprogramma.
Wanneer uit de statussignalen is gebleken, dat de informatie-uitwisseling plaats kan vinden, dan wordt deze uitgevoerd.
Dit gebeurt ook weer onder besturing van instructies uit het werkprogramma.

In een systeem met geprogrammeerde I/O moeten dus op een aantal plaatsen in het werkprogramma instructies t.b.v. het testen van de statussignalen worden opgenomen. Dit kost geheugenruimte en verwerkingstijd.

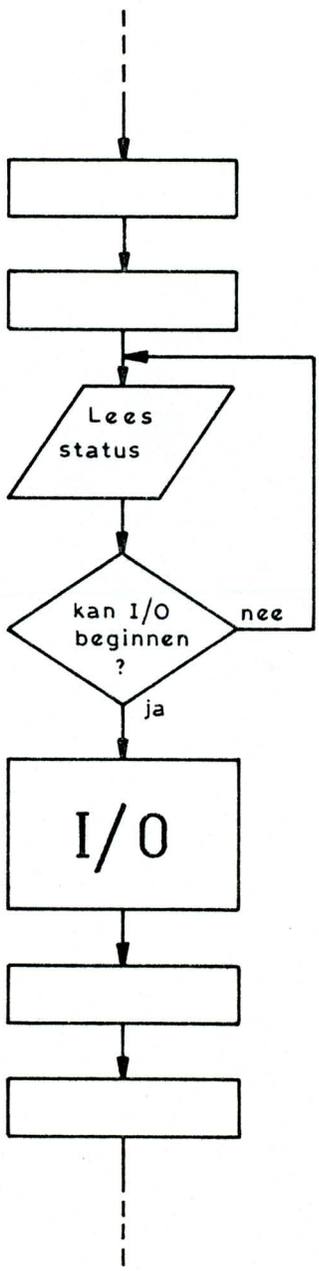


fig.2

b. Interrupt I/O

In systemen met interrupt I/O bepaalt niet de computer maar de buitenwereld het moment, waarop de informatie-uitwisseling moet beginnen. De buitenwereld zendt een z.g. interrupt request (interrupt = onderbreking; request = verzoek, aanvraag) naar de computer (fig. 3).

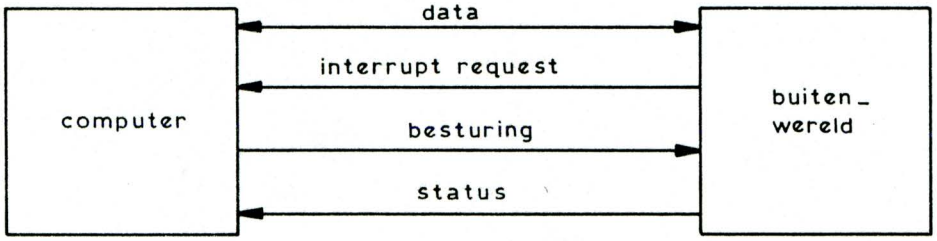


fig.3

De computer zal dan de programma-uitvoering onderbreken en een instructie tussenvoegen. Door deze instructie wordt vanuit het werkprogramma naar een z.g. interrupt service routine (ISR) gesprongen.
Een interrupt service routine is te beschouwen als een subroutine, die niet door een CALL-instructie, maar door een interrupt request wordt aangeroepen (fig. 4).

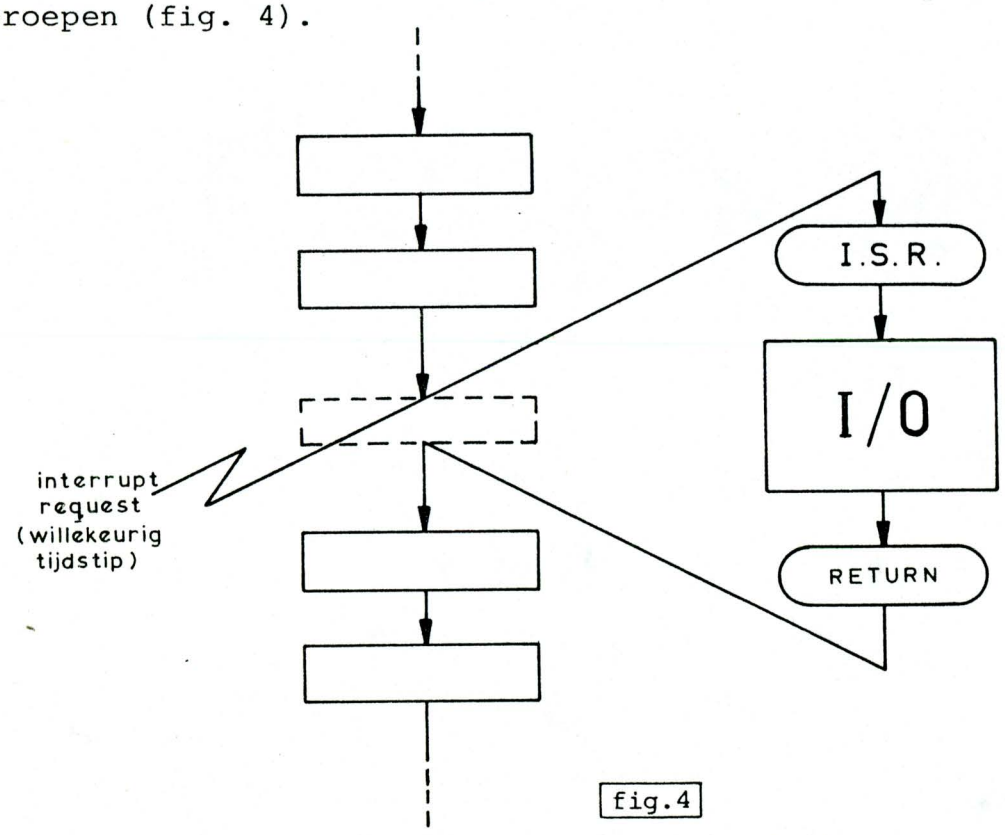


fig.4

In de interrupt service routine bevinden zich de instructies, die de data-uitwisseling besturen. Als deze instructies zijn uitgevoerd, dan moet worden teruggekeerd naar die plaats in het hoofdprogramma, waar dit door de interrupt request werd onderbroken.

Vraag 2: De laatste instructie van een ISR is een .....-instructie.

Daarom moet een ISR altijd met een RETURN-instructie worden afgesloten. Een voorwaarde daarbij is, dat na het ontvangen van de interrupt request, de inhoud van de programmateller in de stack wordt opgeslagen.

Opmerking 1:

De data-uitwisseling tussen computer en buitenwereld wordt zowel bij geprogrammeerde I/O als bij interrupt I/O bestuurd door instructies, dus software-matig.
Het verschil tussen beide I/O-methodes ligt in het starten van de informatie-uitwisseling.

Opmerking 2:

Als derde I/O-methode kennen we direct memory access (DMA). Hierop gaan we in deze les niet in.

SAMENVATTING 1

1. Zowel bij geprogrammeerde I/O als bij interrupt I/O wordt de data-uitwisseling tussen de computer en de buitenwereld softwarematig, dus door instructies, geregeld.

2. Bij systemen met geprogrammeerde I/O moet de computer door het herhaaldelijk testen van statussignalen bepalen of de buitenwereld in staat is tot het plegen van I/O.

3. Bij systemen met interrupt I/O geeft de buitenwereld d.m.v. een interrupt request te kennen, dat er zo snel mogelijk een data-uitwisseling moet plaatsvinden.

3. INTERRUPT-SIGNALEN BIJ DE 8080

Bij het afhandelen van interrupt requests onderscheiden we:
a. accepteren van een interrupt request, dit is het opslaan van een ontvangen interrupt request, zodat deze verder afgehandeld kan worden.
b. honoreren van een interrupt request, dit is het afhandelen van een geaccepteerde interrupt request, d.w.z. dat de CPU actie onderneemt om een extra instructie tussen te voegen en uit te voeren. Als gevolg van deze instructie wordt naar de juiste interrupt service routine gesprongen.

De 8080-microprocessor kent t.b.v. het werken met interrupt I/O 3 besturingssignalen, nl. INT, INTE en INTA.

Vraag 3: Op de INT/INTE/INTA-ingang komen de interrupt requests binnen.
Op de INT/INTE/INTA-uitgang wordt aangegeven, of de CPU in staat is een interrupt request te accepteren.
Op de INT/INTE/INTA-uitgang wordt aangegeven, dat de CPU een interrupt request heeft geaccepteerd.

Op de INT (= interrupt)-ingang kunnen interrupt requests worden ontvangen.

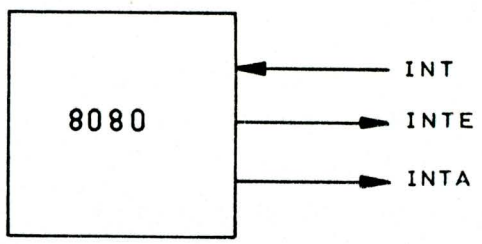


fig.5

Op de uitgang INTE (= interrupt enable) geeft de 8080 aan, of deze wel (INTE = 1) of niet (INTE = 0) in staat is een interrupt request te accepteren. Dit INTE-signaal is bij het werken met interrupt I/O eigenlijk alleen noodzakelijk voor de timing and control unit in de CPU.
Bij 8080-microprocessoren is het INTE-signaal echter wel op één van de aansluitpunten beschikbaar.
Als een interrupt request is geaccepteerd, dan plaatst de 8080 gedurende 1 state een 1 op de uitgang INTA (=interrupt acknowledge).
Dit INTA-signaal fungeert als READ-signaal, om de extra instructie binnen te halen, zodat naar een interrupt service routine kan worden gesprongen.

De INTE- en INTA-signalen worden opgewekt door de timing and control unit en een z.g. interrupt enable flip-flop, ook wel INTE FF genoemd (fig.6).

De uitgang van deze flip-flop is de INTE-uitgang van de CPU. We zullen de INTE- en INTA-signalen aan de hand van een timing-diagram (fig.7) bespreken.

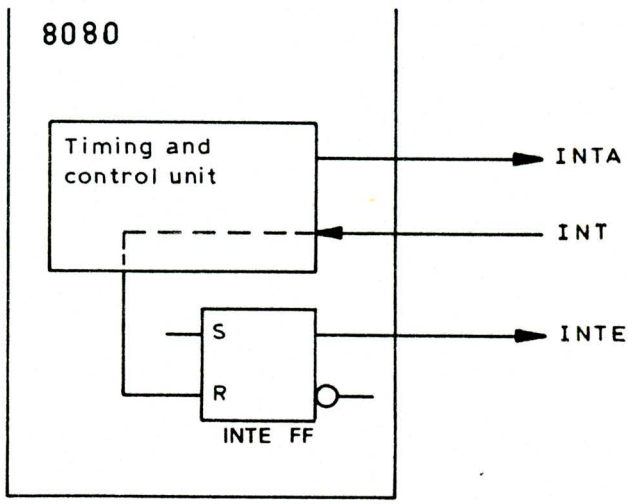


fig.6

 Antw.3: INT; INTE; INTA.

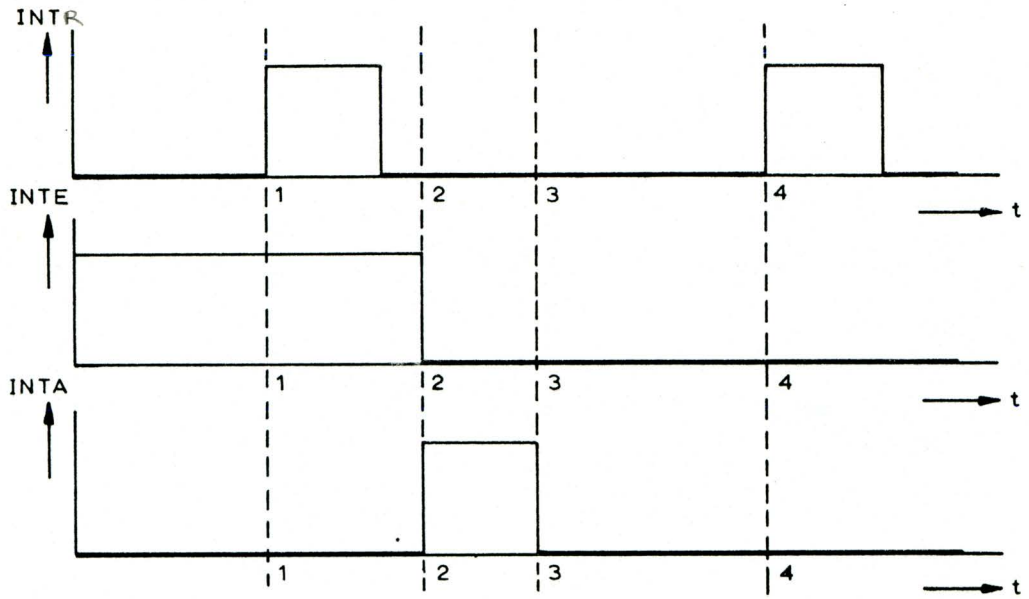


fig.7

Vraag 4: Voor t = 1 is de CPU wel/niet in staat een interrupt request te accepteren.

Voor t = 1 geldt INTE = 1. De CPU kan dus een eventuele interrupt request accepteren.

Op t = 1 wordt een interrupt request ontvangen. De timing and control unit blijft deze even onthouden, totdat de instructie, tijdens welke de interrupt request binnenkwam, volledig is uitgevoerd.
Dit is dus het accepteren van de interrupt request.

Op t = 2 is de lopende instructie afgewerkt. De timing and control unit geeft de interrupt request door aan de reset-ingang van de INTE FF. De INTE-uitgang wordt laag. Tegelijkertijd wordt gedurende 1 state (van t = 2 tot t = 3) op de INTA-uitgang een 1 afgegeven. Wat er verder gebeurt, is afhankelijk van een extern aan te sluiten interrupt controller. Deze moet nl. op commando van het INTA-signaal de extra tussen te voegen instructie aan de CPU afgeven. Dit wordt in de volgende paragraaf besproken.

Op t = 4 wordt er weer een interrupt request ontvangen.

Vraag 5: Deze interrupt request wordt wel/niet geaccepteerd.

Deze interrupt request kan niet worden geaccepteerd. Immers, de interrupt enable flip-flop is nog steeds gereset (INTE = 0).
Dit houdt in, dat na het accepteren van een interrupt request een volgende interrupt request niet kan worden geaccepteerd, tenzij we zorgen dat de interrupt enable flip-flop eerst wordt geset.

Vraag 6: De interrupt enable flip-flop wordt geset door het uitvoeren van een EI/DI-instructie.

Antw.4: wel. Antw.5: niet. Antw.6: EI.

Dit kunnen we bereiken door een EI-instructie (= enable interrupt) uit te laten voeren.

Omgekeerd kunnen we ook voorkomen, dat een bepaald deel van het werkprogramma (b.v. een wachtlus) door een ongewenste interrupt wordt onderbroken.

Vraag 7: Aan het begin van een wachtlus moet dan een EI/DI-instructie worden uitgevoerd.

Aan het begin van deze wachtlus plaatsen we dan een DI-instructie (fig. 8).
Hierdoor wordt de interrupt enable flip-flop gereset.
Totdat er weer een EI-instructie volgt zijn interrupts verboden, d.w.z. dat er geen interrupt requests worden geaccepteerd.

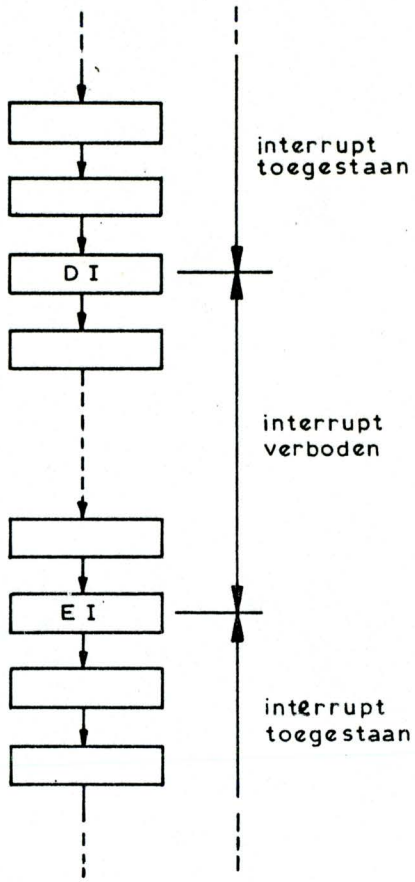


fig.8

 Antw.7: DI.

SAMENVATTING 2

4. T.b.v. het werken met interrupt I/O kent de 8080 de besturingssignalen INT, INTE en INTA.

5. Op de INT-ingang kunnen interrupt requests worden ontvangen.

6. Op de INTE-uitgang geeft de CPU aan, dat interrupt requests kunnen worden geaccepteerd.

7. Op de INTA-uitgang geeft de CPU aan, dat een ontvangen interrupt request is geaccepteerd en dat er een extra instructie moet worden binnengehaald.

8. De interrupt enable flip-flop wordt geset door een EI-instructie.

9. De interrupt enable flip-flop wordt gereset door het accepteren van een interrupt request of door een DI-instructie.

4. INTERRUPT CONTROLLER VOOR DE 8080

In de vorige paragraaf is beschreven, welke interrupt-signalen in 8080-systemen kunnen optreden. De 8080 heeft slechts één INT-ingang, terwijl er soms meer randapparaten, elk met een eigen interrupt request en bijbehorende interrupt service routine, zijn aangesloten. Nu kunnen er in zo'n systeem de volgende problemen optreden:

a. Hoe weet de CPU van welk randapparaat een bepaalde interrupt request afkomstig is ?
b. Hoe weet een randapparaat, dat een hiervan afkomstige interrupt request is geaccepteerd en kan worden gehonoreerd ? (Er is nl. slechts één INTA-signaal.)
c. Hoe kunnen enkele interrupt requests verboden worden, terwijl het accepteren van andere interrupt requests wel is toegestaan ?
d. Welke interrupt request moet worden geaccepteerd, als er tegelijkertijd meer interrupt requests optreden ?
e. Hoe wordt bij elke geaccepteerde interrupt request de bijbehorende interrupt service routine aangeroepen ?

Om al deze problemen op te lossen, moet een extra stuk hardware in het systeem worden opgenomen. Dit is de z.g. interrupt controller (fig. 9).

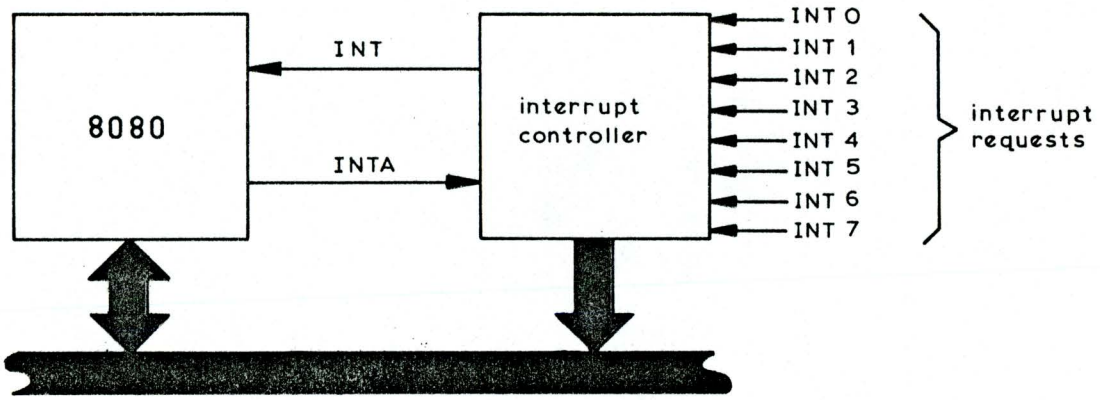


fig.9

De taken van deze interrupt controller zijn:

a. Het onderscheid kunnen maken tussen een aantal verschillende interrupt requests (in fig. 9 zijn dit er maximaal 8).
b. Het verbieden of toestaan van elke afzonderlijke interrupt request. Dit is het z.g. maskeren van interrupt requests.
c. Het regelen van de prioriteit, d.w.z. dat bij het ontvangen van meer interrupt requests eerst de belangrijkste wordt geaccepteerd.
d. Het, na ontvangst van het INTA-signaal, doorgeven van de juiste RESTART-instructie aan de CPU.

Opmerking bij punt a en punt c

Als er, zoals in fig. 9, acht interrupt requests met een bepaalde prioriteit van elkaar kunnen worden onderscheiden, dan zeggen we vaak dat er 8 interrupt-niveaus aanwezig zijn.

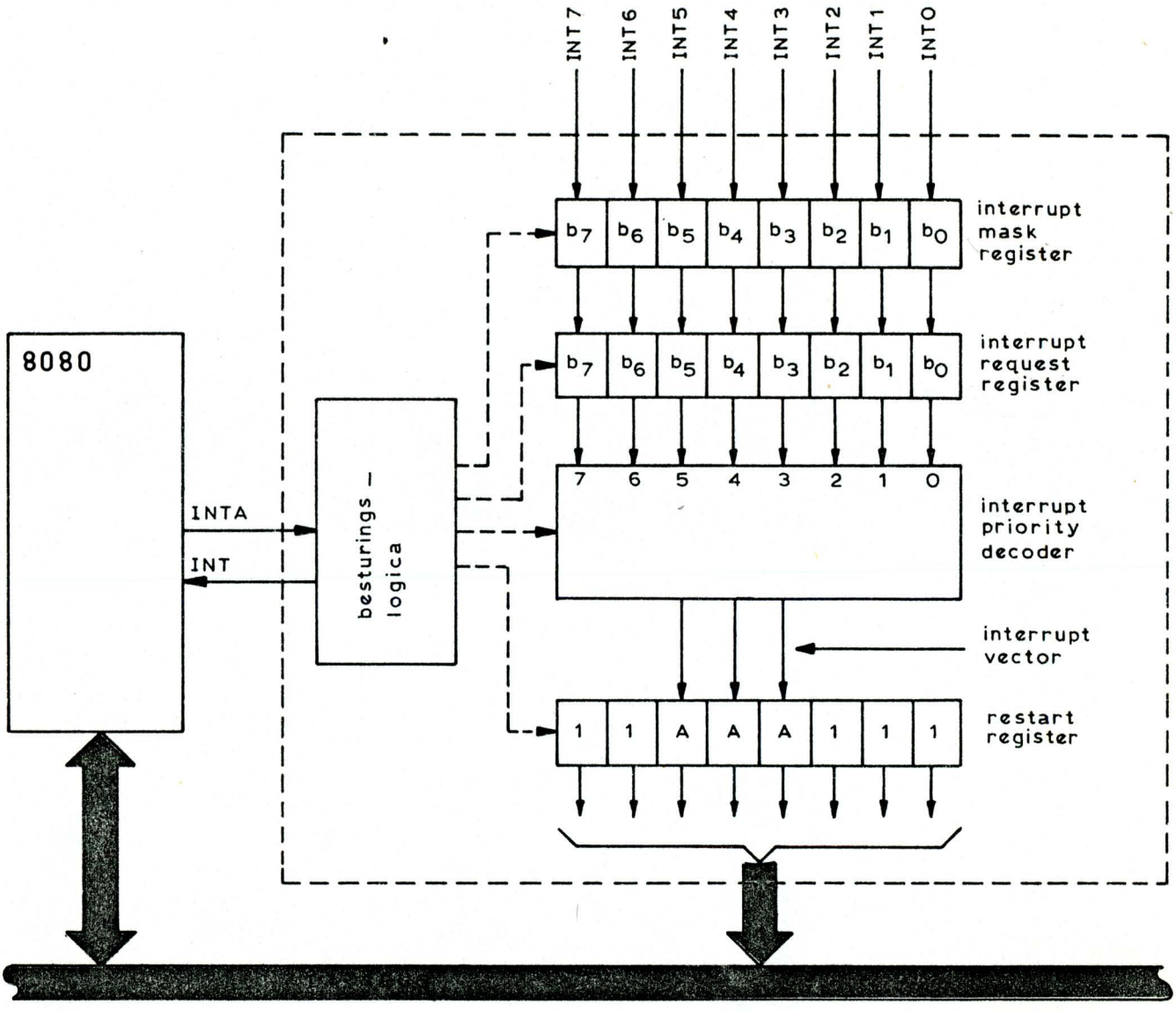

INT 7
INT 6
INT 5
INT 4
INT 3
INT 2
INT 1
INT 0
b7 b6 b5 b4 b3 b2 b1 b0
interrupt mask register
b7 b6 b5 b4 b3 b2 b1 b0
interrupt request register
7 6 5 4 3 2 1 0
interrupt priority decoder
interrupt vector
1 1 A A A 1 1 1
restart register
8080
INTA
INT
besturings - logica

fig 10

In fig. 10 is het blokschema van een interrupt controller voor 8080-systemen getekend.

Deze interrupt controller kent 8 interrupt-niveaus. INT0 heeft de hoogste prioriteit, INT7 heeft de laagste prioriteit.

Deze interrupt controller bestaat uit de volgende delen:
a. Besturingslogica.
b. Interrupt mask register.
c. Interrupt request register.
d. Interrupt priority decoder.
e. Restart register.

a. Besturingslogica

De besturingslogica (engels: control logic) regelt de totale gang van zaken binnen de interrupt controller, waarbij rekening wordt gehouden met het INTA-signaal van de CPU. Als er een te accepteren interrupt request wordt ontvangen, dan geeft de besturingslogica een INT-signaal voor de CPU af.

b. Interrupt mask register

In het interrupt mask register kan de programmeur aangeven welke interrupt requests wel en welke niet mogen worden geaccepteerd.
Als een bepaalde bit met 0 wordt gevuld, dan wordt een op deze bit binnenkomende interrupt request geaccepteerd.
Is een bit in het interrupt mask register 1, dan wordt de overeenkomstige interrupt request niet geaccepteerd.

> Vraag 8: Als het interrupt mask register wordt gevuld met $F1_{16}$, dan worden de interrupt requests,die binnenkomen op de lijnen ....., ... en ..... geaccepteerd.
> Interrupt requests, die binnenkomen op de lijnen ....., ..., ..., ... en ..... worden verboden.

Stel dat het interrupt mask register wordt gevuld met $F1_{16} = 11110001_2$, dan geldt dat b_3, b_2 en b_1 "0" zijn. Interrupt requests, die op deze bits binnenkomen, dit zijn INT3, INT2 en INT1, zullen dan worden geaccepteerd.
De overige bits (b_7, b_6, b_5, b_4 en b_0) zijn dan "1", d.w.z. dat de interrupt requests INT7, INT6, INT5, INT4 en INT0 niet worden geaccepteerd.

Opmerking:

In deze beschrijving is er van uitgegaan, dat een interrupt request wordt geaccepteerd, als de bijbehorende bit in het interrupt mask register "0" is. Er zijn echter ook interrupt controllers, waarbij een interrupt request wordt geaccepteerd door een "1" in het interrupt mask register.

Antw.8: INT3; INT2; INT1; INT7; INT6; INT5; INT4; INT0.

c. Interrupt request register

In het interrupt request register wordt een interrupt request opgeslagen (in de vorm van "1") als deze (tengevolge van de inhoud van het interrupt mask register) mag worden geaccepteerd.

Stel dat het interrupt mask register is gevuld met $A7_{16}$ en er komt een interrupt request binnen op INT4.

Vraag 9: De inhoud van het interrupt request register wordt dan .....$_2$.

Deze interrupt request mag worden geaccepteerd, want b_4 van het interrupt mask register is 0 ($A7_{16} = 10100111_2$). b_4 van het interrupt request register wordt dan 1. De overige bits blijven 0. De inhoud van het interrupt request register wordt dan 00010000_2.

d. Interrupt priority decoder

De taak van de interrupt priority decoder is het afgeven van een identificatiecode, die de eerstvolgend te honoreren interrupt request aangeeft. Omdat de interrupt controller van fig. 10 acht interrupt-niveaus heeft, bestaat de identificatiecode uit 3 bits (= 2^3 = 8 mogelijkheden).

Vraag 10: De identificatiecode, die bij de interrupt request INT5 behoort, is .....$_2$.

Als INT5 moet worden gehonoreerd, dan is de identificatiecode 101_2.

Tabel 1 is de waarheidstabel voor de interrupt priority decoder. De identificatiecode wordt opgebouwd m.b.v. de inhoud van het interrupt request register (IRR), waarbij rekening wordt gehouden met de prioriteit (INT0 heeft de hoogste prioriteit).

Inhoud IRR b_7.....b_0	Identificatie-code	Eerstvolgend te honoreren interrupt request
00000000	---	-
XXXXXXX1	000	INT0
XXXXXX10	001	INT1
XXXXX100	010	INT2
XXXX1000	011	INT3
XXX10000	100	INT4
XX100000	101	INT5
X1000000	110	INT6
10000000	111	INT7

Tabel 1

Als de inhoud van het interrupt request register 00000000_2 is, dan is er geen interrupt request ontvangen, die moet worden geaccepteerd en gehonoreerd. Er is dan geen sprake van een identificatiecode.

In de overige 8 regels van tabel 1 komen don't cares (X) voor. Deze geven aan, dat het niets uitmaakt of deze bits 0 of 1 zijn. Van een aantal gelijktijdig geaccepteerde interrupt requests wordt nl. alleen die met de hoogste prioriteit uitgevoerd.

Stel b.v., dat de inhoud van het interrupt request register 01000100_2 is. Van de geaccepteerde interrupt requests INT6 en INT2 heeft INT2 een hogere prioriteit. De identificatiecode is dan 010_2. Als INT2 is gehonoreerd, zorgt de besturingslogica ervoor, dat b_2 van het interrupt request register 0 wordt. De inhoud hiervan wordt dan 01000000_2, zodat nu INT6 kan worden gehonoreerd. (Behalve wanneer ondertussen één van de interrupt requests INT0 t/m INT5 is geaccepteerd. Deze worden dan voor INT6 gehonoreerd i.v.m. hun hogere prioriteit.)

e. Restart register

M.b.v. de identificatiecode wordt een RESTART-instructie (RST) gevormd. Deze RST-instructie wordt opgeslagen in het z.g. restart register, door sommige fabrikanten ook wel interrupt address register genoemd.

De algemene vorm van de RST-instructie is 11AAA111. Hierin is AAA de identificatiecode of interrupt vector.

Vraag 11: De instructie RST1 luidt in machinetaal $\ldots\ldots_{16}$.

Zo komt RST1 overeen met $11001111_2 = CF_{16}$.

Na het ontvangen van een INTA-signaal wordt de in het restart register aanwezige RST-instructie op de databus geplaatst, zodat deze door de CPU kan worden uitgevoerd.

SAMENVATTING 3

10. Een interrupt controller regelt het accepteren van interrupt requests.

11. Geaccepteerde interrupt requests worden met een bepaalde prioriteit gehonoreerd, d.w.z. de belangrijkste eerst.

12. Na het ontvangen van een INTA-signaal zet de interrupt controller een RESTART-instructie op de databus. Deze wordt dan door de CPU uitgevoerd.

Antw.11: CF.

5. UITVOERING VAN RESTART-INSTRUCTIES

Het doel van een RESTART-instructie is het vullen van de programmateller met het beginadres van de juiste interrupt service routine (ISR). Omdat de CPU na het beëindigen van deze ISR weer verder moet gaan op die plaats in het hoofdprogramma, waar dit werd onderbroken door een interrupt, moet aan de volgende voorwaarden worden voldaan:

a. Door het uitvoeren van een RESTART-instructie moet de inhoud van de programmateller worden opgeslagen in de stack.
b. De interrupt service routine moet eindigen met een RETURN-instructie, waardoor naar de juiste plaats in het hoofdprogramma wordt teruggesprongen.

Vraag 12: Een RST-instructie is te vergelijken met een .....-instructie.

Een RST-instructie heeft dezelfde taken als een CALL-instructie. Tussen beide instructies is echter één groot verschil.

Vraag 13: Een CALL-instructie bestaat uit 1/2/3 bytes.
Een RST-instructie bestaat uit 1/2/3 bytes.

Een CALL-instructie bestaat altijd uit 3 bytes, waarvan er 2 het beginadres van de aan te roepen subroutine bevatten.
Een RST-instructie bestaat uit slechts 1 byte. Voordat de programmateller kan worden gevuld met het beginadres van de interrupt service routine, moet de timing en control unit dit beginadres bepalen aan de hand van de interrupt vector (deze ligt immers in de RST-instructie besloten).

De programmateller wordt dan gevuld met $0000000000AAA000_2$. Hierin is AAA weer de interrupt vector.

Vraag 14: Door een RST3-instructie wordt de programmateller gevuld met ..0018..$_{16}$.

Na het optreden van een RST3-instructie wordt de programmateller dus gevuld met $0000000000011000_2 = 0018_{16}$. Dit adres wordt vaak aangeduid met restart-adres.
In tabel 2 zijn de 8 mogelijke RST-instructies met de bijbehorende restart-adressen vermeld.

interrupt vector	RST-instructie (hexadec.)	restart-adres (hexadec.)
000	C7	0000
001	CF	0008
010	D7	0010
011	DF	0018
100	E7	0020
101	EF	0028
110	F7	0030
111	FF	0038

Tabel 2

Vraag 15: De RST-instructie RST6 komt overeen met het resetten van de CPU.

Vraag 16: Bij elk van de restart-blokken in tabel 2 behoort een blok van ..... geheugenplaatsen.

Uit tabel 2 kunnen we de volgende conclusies trekken:

1. RST0 komt overeen met het geven van een RESET-impuls aan de CPU, met dat verschil, dat bij RST0 de inhoud van de programmateller in de stack wordt geplaatst.
2. Bij elk restart-adres behoren 8 bytes, waarin een sprong naar een interrupt service routine kan worden geplaatst.

In de meeste gevallen is 8 bytes te weinig om een complete ISR te bevatten. Daarom bevinden zich op de restart-adressen vaak sprongopdrachten naar de feitelijke ISR (fig. 11). We zullen het totale samenspel tussen CPU en interrupt controller aan de hand van een voorbeeld beschrijven.

Opmerking:

De ISR7 kan direct op adres 0038_{16} beginnen, omdat dit het laatste van de 8 restart-adressen is.

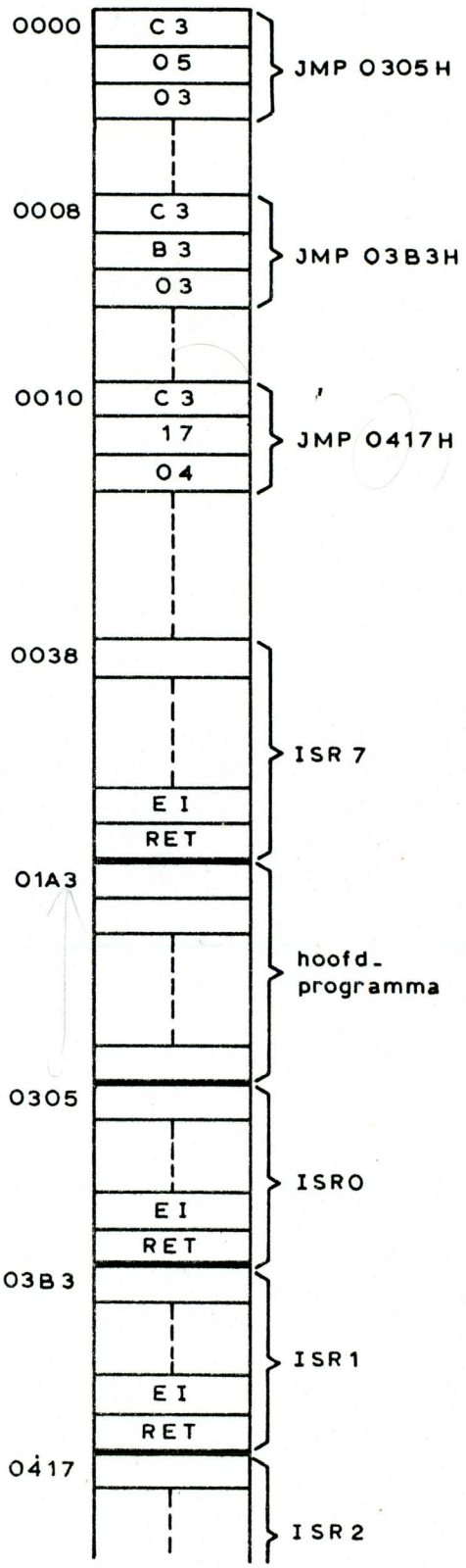


fig.11

Antw.15: RST0. Antw.16: 8.

6. VOORBEELD

Gegeven: Een microcomputer met als CPU een 8080, bevat een interrupt controller volgens fig. 10. Een deel van de memory map is zoals in fig. 11 is getekend. Op adres 0248_{16} (in het hoofdprogramma) bevindt zich de instructie DCR A.
De INTE FF in de CPU is gereset. De inhoud van het interrupt mask register in de interrupt controller is $F0_{16}$.
Op adres 0267_{16} (in het hoofdprogramma) staat een EI-instructie.

Gevraagd: Beschrijf wat er achtereenvolgens gebeurt, als er tijdens de instruction fetch van de instructie DCR A (adres 0248_{16}) een interrupt request INT2 wordt ontvangen.

GEEF EERST ZELF DE GEVRAAGDE BESCHRIJVING.
ALS U VOORGAANDE PARAGRAFEN GOED HEEFT BESTUDEERD, DAN MOET DIT U ZONDER MEER LUKKEN. VERGELIJK DAARNA UW BESCHRIJVING MET DE ONZE.

a. De CPU is bezig met de uitvoering van het hoofdprogramma. Tijdens de instruction fetch van de instructie DCR A wordt op de interrupt controller een interrupt request INT2 ontvangen. Deze wordt in het interrupt request register opgeslagen (b_2 van het interrupt mask register is immers 0).

b. De interrupt controller zet een 1 op de INT-lijn, plaatst een RST2-instructie in het restart-register en blijft dan op een INTA-signaal van de CPU wachten.

c. De INTE FF is gereset, dus de CPU reageert niet op het INT-signaal. Er wordt dan ook geen INTA-signaal afgegeven.
De interrupt controller laat de 1 op de INT-lijn staan en houdt de RST2-instructie in het restart-register.

d. De CPU blijft doorgaan met de programma-uitvoering. Op een bepaald moment zal dan de EI-instructie op adres 0267_{16} worden uitgevoerd. Daardoor wordt de INTE FF geset. De CPU is dan in staat interrupt requests te accepteren.

e. Op de INT-lijn staat nog steeds een 1. De CPU reset de INTE FF en zendt een INTA-signaal naar de interrupt controller.

f. De interrupt controller zet de RST2-instructie op de databus en maakt de INT-lijn 0. (De interrupt request is nu immers door de CPU geaccepteerd). RST2 wordt in het instructieregister van de CPU geplaatst en uitgevoerd. De inhoud van de programmateller wordt dus naar de stack overgebracht. De nieuwe inhoud van de programmateller wordt 0010_{16}.

g. De CPU voert de instructie JMP 0417 uit en springt dus naar het begin van ISR2. De instructies van deze interrupt service routine worden alle uitgevoerd.

h. Aan het eind van ISR2 bevindt zich een EI-instructie. Hierdoor wordt de INTE FF in de CPU weer geset.
Eventueel volgende interrupt requests kunnen nu weer worden geaccepteerd.

i. Daarna wordt de RET-instructie uitgevoerd. De programmateller krijgt vanuit de stack weer de oorspronkelijke inhoud. De CPU gaat dan verder met de uitvoering van het hoofdprogramma, op die plaats, waar dit door de interrupt request INT2 werd onderbroken.

Opmerking:

In dit voorbeeld staat de EI-instructie aan het eind van ISR2. Tijdens het uitvoeren van deze interrupt service routine kunnen er dus geen interrupt requests worden geaccepteerd. Als ISR2 wel mag worden onderbroken door andere interrupt requests, dan moet de EI-instructie aan het begin van de ISR staan. We spreken dan van het nesten van interrupts. Dit is het accepteren en honoreren van een interrupt request, tijdens het uitvoeren van een ISR, die bij een vorige interrupt request behoort.

SAMENVATTING 4

13. Een RST-instructie bergt de inhoud van de programmateller op in de stack en vult de programmateller dan met een restart-adres. De RST-instructie is te beschouwen als een 1-byte CALL-instructie.

14. De algemene vorm van een RST-instructie is $11AAA111_2$.

15. De algemene vorm van een restart-adres is $0000000000AAA000_2$.

16. Onder het nesten van interrupts verstaan we het accepteren en honoreren van een interrupt request, voordat de ISR van een vorige interrupt request volledig is uitgevoerd.

7. INTERRUPT SIGNALEN BIJ DE 8085

De 8085-microprocessor kent, voor wat betreft het werken met interrupts, een aantal uitbreidingen t.o.v. de 8080.
Deze zijn in fig. 12 getekend.

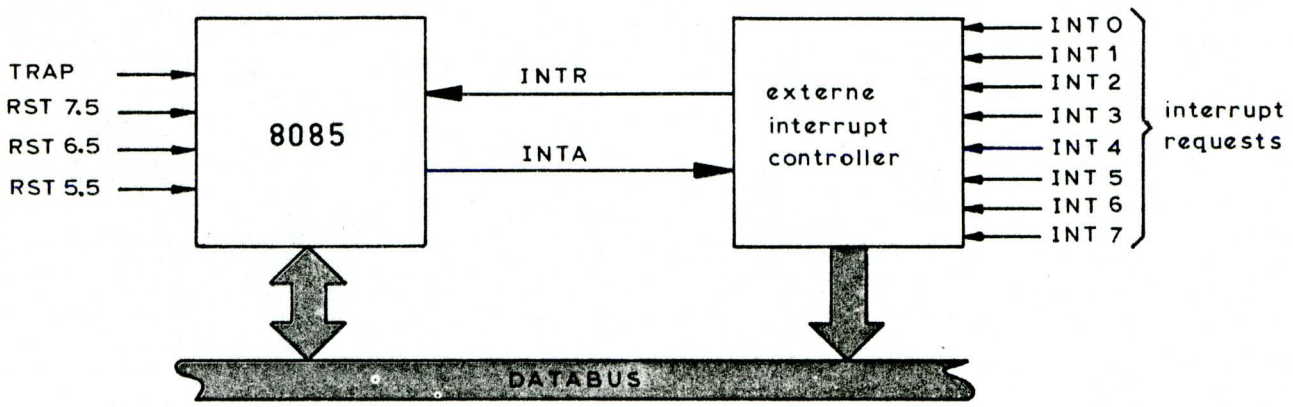


fig.12

De 8085 bezit een interne interrupt controller, die de interrupt requests TRAP, RST7.5, RST6.5 en RST5.5 verwerkt.
De interrupt requests INT0 t/m INT7, die ook bij 8080-systemen voorkomen, moeten door een externe interrupt controller worden afgehandeld.

Vraag 17: Een 8080-systeem volgens fig. 9 kent maximaal ..... interruptniveaus.
Een 8085-systeem volgens fig.12 kent maximaal ..... interruptniveaus.

We zien, dat 8085-systemen 4 interrupt niveaus extra hebben t.o.v. systemen met een 8080-CPU.
We kunnen de interrupt-afhandeling in 8085-systemen in twee afzonderlijke delen splitsen:
1. De externe interrupt controller (paragraaf 8).
2. De interne interrupt controller (paragraaf 9 en verder).

8. DE EXTERNE INTERRUPT CONTROLLER VOOR DE 8085

De externe interrupt controller voor 8085-systemen is vrijwel gelijk aan die voor 8080-systemen (fig. 10). Er zijn echter twee kleine verschillen.

1. Het INT-signaal (bij de 8080) wordt in 8085-systemen meestal INTR genoemd. Dit verschil ligt alleen in de naam, dus niet in de functie van dit signaal.

2. De 8085 geeft geen INTE-signaal af. In de 8085 is wel een interrupt enable flip-flop aanwezig, maar de uitgang hiervan is niet met een aansluitpunt van de CPU verbonden.
   Dit INTE-signaal is echter niet nodig voor de externe interrupt controller.

Wanneer er één van de interrupt requests INT0 t/m INT7 wordt ontvangen, dan geeft de externe interrupt controller steeds een signaal op de INTR-lijn naar de CPU af (tenminste als er een 0 in de betreffende bit van het interrupt mask register staat).
De CPU meldt dan via een INTA-signaal of deze interrupt request is geaccepteerd en dat er een RST-instructie op de databus geplaatst dient te worden.

Als er geen INTA-signaal van de CPU komt, dan zal de externe interrupt controller geen RST-instructie op de databus plaatsen.

RST-instructies worden door de 8085 op dezelfde wijze uitgevoerd als door de 8080. Ook bij de 8085 moet na het accepteren van een interrupt request een EI-instructie zijn uitgevoerd, voordat een volgende interrupt request kan worden geaccepteerd.

SAMENVATTING 5

17. De interne interrupt controller in de 8085 kan de interrupt requests TRAP, RST5.5, RST6.5 en RST7.5 afhandelen.

18. Als er meer dan 4 interrupt-niveaus gewenst zijn, dan moet de 8085 worden uitgebreid met een externe interrupt controller.

19. De besturingssignalen tussen de 8085 en de externe interrupt controller zijn INTR en INTA.

20. INTA fungeert als READ-signaal bij het binnenhalen van een RST-instructie.

9. DE INTERNE INTERRUPT CONTROLLER

De interne interrupt controller in de 8085 maakt deel uit van de timing and control unit. Daarom is de interne interrupt controller eenvoudiger dan de externe. De interne interrupt controller behoeft nl. geen interrupt vector en RST-instructies te genereren.
Als er nl. een te accepteren interrupt request wordt ontvangen, kan de interrupt controller (in samenwerking met de rest van de timing and control unit) rechtstreeks de programmateller, de instructie-decoder en de stackpointer besturen.

In fig. 13 is het blokschema van de interne interrupt controller getekend.

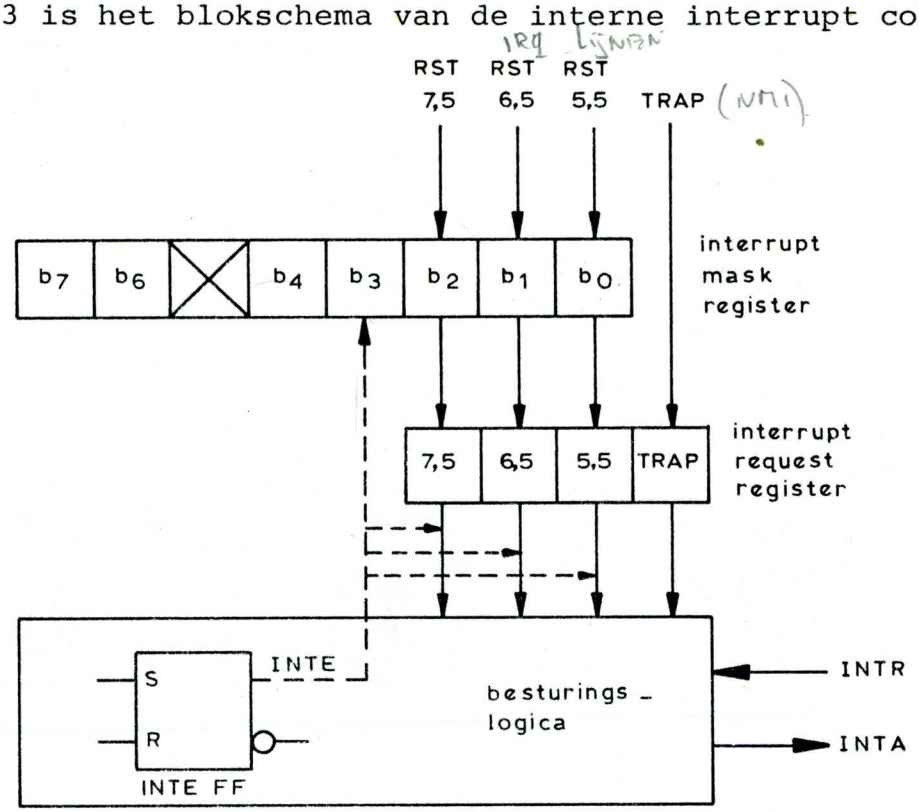


fig.13

De interne interrupt controller bestaat uit de volgende hoofddelen:
a. Interrupt mask register.
b. Interrupt request register.
c. Besturingslogica.

a. Interrupt mask register

Het interrupt mask register is een 8-bits register, waarvan echter maar 5 bits (b_4 t/m b_0) door de interrupt controller worden gebruikt.

b_5 wordt nl. helemaal niet gebruikt (= undefined) en b_6 en b_7 worden aangesproken bij serial I/O (zie de les "Systeemeigenschappen hardware").

b_0, b_1 en b_2 zijn echte interrupt masks. Een 0 in deze bits, laat de bijbehorende interrupt requests accepteren. Een 1 in één van deze bits verbiedt het accepteren van de bijbehorende interrupt request.

b_3 heeft een dubbele functie. We kunnen het interrupt mask register nl. zowel vullen met een nieuwe waarde als uitlezen. Dit gebeurt met de z.g. SIM- en RIM-instructies. (Deze instructies worden in paragraaf 10 behandeld.)

Op het moment dat we het interrupt mask register met een nieuwe waarde vullen (om b.v. b_0, b_1 en b_2 te wijzigen), dan moet $b_3 = 1$ zijn.
In dit geval worden alle 8 bits van het interrupt mask register met de nieuwe waarde gevuld. Als $b_3 = 0$ was geweest, dan waren alleen b_3 t/m b_7 veranderd. b_0, b_1 en b_2 zouden dan ongewijzigd zijn gebleven.

Op het moment, dat we het interrupt mask register uitlezen, fungeert b_3 als INTE-signaal.

b_4 fungeert als een tweede mask bit voor de interrupt request RST7.5. Als $b_4 = 1$, dan kan RST7.5 niet worden geaccepteerd, onafhankelijk van b_2. Als $b_4 = 0$, dan is het wel of niet accepteren van RST7.5 afhankelijk van de toestand van b_2.

We zullen de werking van b_0 t/m b_4 van het interrupt mask register toelichten aan de hand van enkele voorbeelden.
Deze voorbeelden bevinden zich aan het eind van paragraaf 10, omdat u voor het werken met het interrupt mask register de instructies RIM en SIM dient te kennen.

Vraag 18: D.m.v. het interrupt mask register kunnen we kiezen of de interrupt requests ....., ... en ..... wel of niet mogen worden geaccepteerd.

b. Interrupt request register

In het interrupt request register worden die interrupt requests opgeslagen, die geaccepteerd mogen worden. Behalve RST5.5, RST6.5 en RST7.5, die via het interrupt mask register lopen, is er nog een vierde interrupt request TRAP mogelijk. TRAP gaat buiten het interrupt mask register om, d.w.z. dat we niet kunnen aangeven of TRAP wel of niet moet worden geaccepteerd.
Ook door het resetten van de INTE FF kan het accepteren van TRAP niet worden verboden. TRAP wordt nl. altijd geaccepteerd, ook na een DI-instructie.

TRAP wordt daarom vaak een nonmaskable interrupt request genoemd. (nonmaskable = niet maskeerbaar).

Het is van belang te weten, aan welke (elektrische) eisen een interrupt request moet voldoen, om te kunnen worden gedetecteerd.

INTR, RST5.5 en RST6.5 zijn high level sensitive. D.w.z. dat op deze signalen kan worden gereageerd als ze 1 zijn (high level = hoog niveau).

Antw.18: RST5.5; RST6.5; RST7.5.

RST7.5 is rising edge sensitive. D.w.z. er wordt alleen gereageerd op de voorflank van dit signaal (rising edge = stijgende flank).

TRAP is rising edge and high level sensitive. D.w.z. dat zowel op een voorflank als op een continue 1 wordt gereageerd.

c. Besturingslogica

De taak van de besturingslogica ligt al in de naam besloten, nl. het besturen van de interrupt-afhandeling in de 8085.
De besturingslogica zal in ieder geval met de externe interrupt controller moeten samenwerken (via de INTR- en INTA-signalen). Eventuele RST-instructies (RST0 t/m RST7), die via de databus binnenkomen, moeten op de juiste wijze worden uitgevoerd.

Bovendien moeten de interrupt requests, die in het interrupt request register staan (RST5.5, RST6.5, RST7.5 en TRAP), worden afgehandeld. D.w.z. dat de juiste restart-adressen in de programmateller moeten worden geplaatst.

In tabel 3 zijn de restart-adressen voor de 12 mogelijke interrupt requests vermeld. De interrupt requests RST5.5, RST6.5, RST7.5 en TRAP zijn tussen haken geplaatst, omdat hierbij geen "echte" RESTART-instructies behoren. Deze interrupt requests worden nl. rechtstreeks door de interne interrupt controller afgehandeld, zonder dat er een operatiecode in het instructie-register hoeft te staan (zoals bij RST0 t/m RST7).

interrupt request	restart-adressen (hexadecimaal)
RST0	0000
RST1	0008
RST2	0010
RST3	0018
RST4	0020
(TRAP)	0024
RST5	0028
(RST5.5)	002C
RST6	0030
(RST6.5)	0034
RST7	0038
(RST7.5)	003C

Tabel 3

Uit tabel 3 blijkt, waarom de naam RST5.5 is gekozen. Het restart-adres van RST5.5 ligt precies tussen de restart-adressen van RST5 en RST6 in. Om dezelfde reden wordt TRAP ook wel RST4.5 genoemd.

Een belangrijke taak van de besturingslogica is het handhaven van de prioriteit.
Hieronder zijn de interrupt requests in volgorde van prioriteit vermeld.

TRAP (hoogste prioriteit)
RST7.5
RST6.5
RST5.5
INTR (laagste prioriteit)

De onderlinge prioriteit van de interrupt requests INT0 t/m INT7 hangt van de externe interrupt controller af.

SAMENVATTING 6

21. De interne interrupt controller in de 8085 bestaat uit de volgende delen:
    a. Interrupt mask register.
    b. Interrupt request register.
    c. Besturingslogica.

22. De interrupt requests RST5.5., RST6.5 en RST7.5 lopen via het interrupt mask register.

23. De interrupt request TRAP (RST4.5) wordt altijd geaccepteerd. TRAP is een nonmaskable interrupt request.

24. Na het accepteren van één van de interrupt requests RST5.5, RST6.5, RST7.5 of TRAP wordt geen RESTART-instructie gegenereerd. De interne interrupt controller kan rechtstreeks de programmateller en de stackpointer besturen.

10. DE RIM- en SIM-INSTRUCTIES

De RIM- en SIM-instructies dienen om het interrupt mask register uit te lezen resp. te vullen. Bij de uitvoering van beide instructies is de accumulator betrokken.

De RIM-instructie (read interrupt masks) brengt de inhoud van het interrupt mask register over naar de accumulator.
In b_3 van de accumulator komt dan de uitgang van de INTE FF (INTE) terecht.

De SIM-instructie (set interrupt masks) vult het interrupt mask register vanuit de accumulator. Om de inhoud van b_0, b_1 en b_2 van het interrupt mask register te wijzigen, moet b_3 van de accumulator 1 zijn.
Als $b_3 = 0$, dan worden door de SIM-instructie alleen b_4 t/m b_7 van de accumulator naar het interrupt mask register overgebracht.

We zullen nu enkele voorbeelden van het werken met het interrupt mask register bespreken. Hierbij is vooral de stof uit paragraaf 9a van belang. Lees deze zonodig nog eens door.

VOORBEELD 1

Gegeven: Na het uitvoeren van een RIM-instructie is de inhoud van de accumulator 00_{16}.

Gevraagd: Welke interrupt requests kunnen nu worden geaccepteerd.

Vraag 19: RST5.5 kan wel/niet worden geaccepteerd.
RST6.5 kan wel/niet worden geaccepteerd.
RST7.5 kan wel/niet worden geaccepteerd.
TRAP kan wel/niet worden geaccepteerd.

Oplossing:

b_3 van de accumulator (= INTE) is 0. Alleen TRAP kan dus worden geaccepteerd.

VOORBEELD 2

Gegeven: Na het uitvoeren van een RIM-instructie is de inhoud van de accumulator $1A_{16}$.

Gevraagd: Welke interrupt requests kunnen nu worden geaccepteerd ?

Vraag 20: RST5.5 kan wel/niet worden geaccepteerd.
RST6.5 kan wel/niet worden geaccepteerd.
RST7.5 kan wel/niet worden geaccepteerd.
TRAP kan wel/niet worden geaccepteerd.

Oplossing:

De inhoud van de accumulator is $1A_{16} = 00011010_2$. b_3 = INTE = 1, dus RST5.5 kan worden geaccepteerd, want $b_0 = 0$. RST6.5 kan niet worden geaccepteerd, want $b_1 = 1$. $b_4 = 1$, dus onafhankelijk van de toestand van

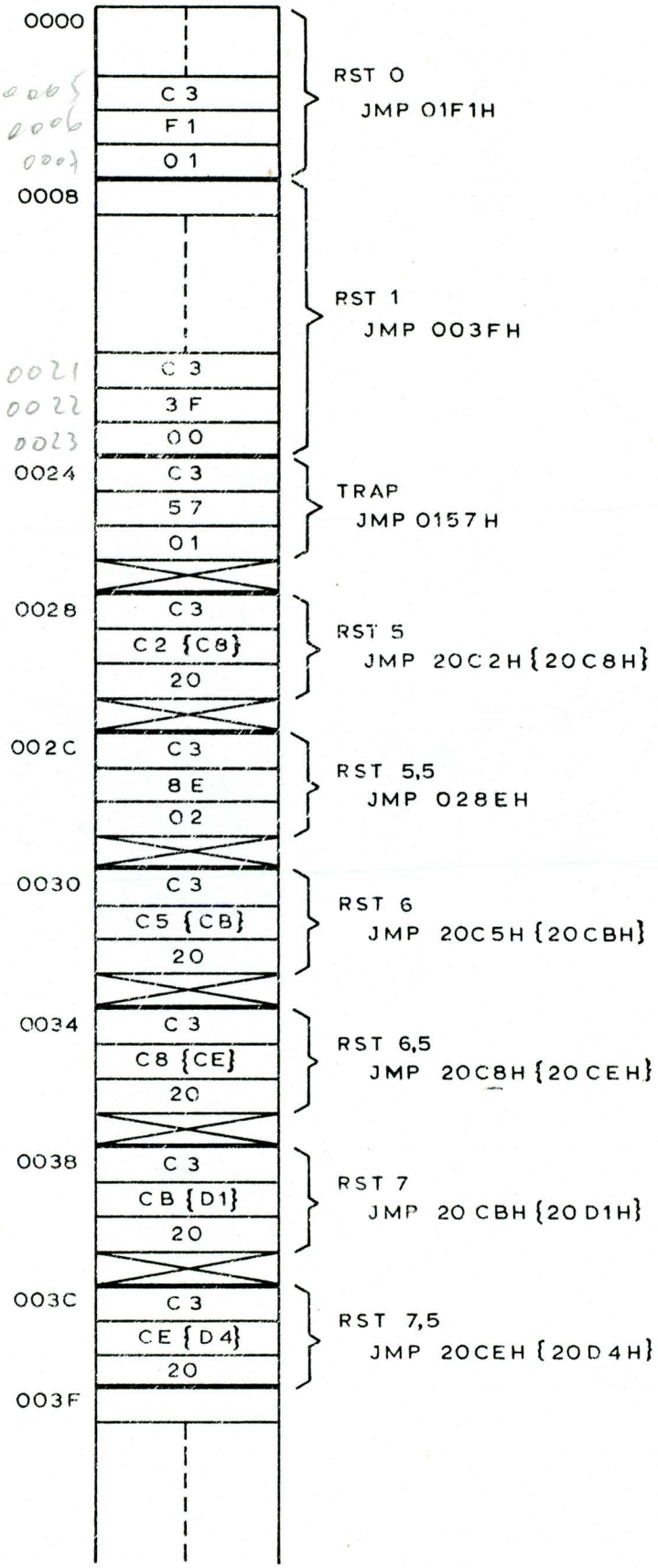

0000
C 3
F 1
0 1
0008
C 3
3 F
0 0
0024
C 3
5 7
0 1
0028
C 3
C2 {C8}
20
002C
C 3
8 E
0 2
0030
C 3
C5 {CB}
20
0034
C 3
C8 {CE}
20
0038
C 3
CB {D1}
20
003C
C 3
CE {D4}
20
003F
RST 0
JMP 01F1H
RST 1
JMP 003FH
TRAP
JMP 0157H
RST 5
JMP 20C2H {20C8H}
RST 5,5
JMP 028EH
RST 6
JMP 20C5H {20CBH}
RST 6,5
JMP 20C8H {20CEH}
RST 7
JMP 20CBH {20D1H}
RST 7,5
JMP 20CEH {20D4H}

fig.14

b_2 zal RST7.5 niet worden geaccepteerd.

VOORBEELD 3

Gegeven: In een 8085-systeem wordt nevenstaand programma uitgevoerd.

```
EI
MVI  A,09H
SIM
XRA  A
SIM
-
-
```

Gevraagd: Welke interrupt requests kunnen na de eerste SIM-instructie worden geaccepteerd ? En na de tweede SIM-instructie ?

Vraag 21: Na de eerste SIM-instructie kan
RST5.5 wel/niet worden geaccepteerd.
RST6.5 wel/niet worden geaccepteerd.
RST7.5 wel/niet worden geaccepteerd.

Oplossing:

Door de EI-instructie wordt de INTE FF geset, dus INTE wordt 1.
TRAP kan in ieder geval worden geaccepteerd.
Door de MVI- en SIM-instructies wordt de inhoud van het interrupt mask register $09_{16} = 00001001_2$ ($b_3 = 1$, dus ook b_0, b_1 en b_2 worden gevuld).
Nu is $b_0 = 1$, dus RST5.5 wordt niet geaccepteerd.
RST6.5 en RST7.5 wel, want b_4, b_2 en b_1 zijn alle 0.

Daarna wordt de instructie XRA A uitgevoerd, gevolgd door een tweede SIM-instructie.

Vraag 22: Na deze tweede SIM-instructie kan
RST5.5 wel/niet worden geaccepteerd.
RST6.5 wel/niet worden geaccepteerd.
RST7.5 wel/niet worden geaccepteerd.

Door XRA A worden alle bits in de accumulator, dus ook b_3, 0.
Als dit d.m.v. een SIM-instructie naar het interrupt mask register wordt overgebracht, dan zullen b_0, b_1 en b_2 dus niet wijzigen.
De toestand blijft gelijk aan die na de eerste SIM-instructie.

SAMENVATTING 7

25. Door de RIM-instructie wordt de inhoud van het interrupt mask register binnen de 8085 naar de accumulator gebracht.

26. Door de SIM-instructie wordt de inhoud van de accumulator in het interrupt mask register geplaatst.

27. Tijdens het uitvoeren van een RIM-instructie fungeert b_3 van het interrupt mask register als INTE-signaal.
Bij een SIM-instructie dient b_3 om de bits b_0, b_1 en b_2 wel of niet te laten wijzigen.

Antw.21: niet; wel; wel. Antw.22: niet; wel; wel.

11. INTERRUPT MOGELIJKHEDEN IN DE SDK 85

In de SDK 85 bevindt zich geen externe interrupt controller. De interrupt requests INT0 t/m INT7 kunnen dus niet voorkomen. Dit houdt echter niet in, dat geen der RESTART-instructies RST0 t/m RST7 kunnen optreden. Immers, wanneer we een RESTART-instructie in het objectprogramma opnemen, dan zal deze tijdens de programma-uitvoering in het instructieregister worden geplaatst en uitgevoerd.

De interne interrupt controller is wel aanwezig. Deze zit nl. in de 8085-CPU. Aan de hierop binnenkomende vier interrupt requests zijn, door het systeem bepaalde, vaste functies toegekend.

In fig. 14 zijn de inhouden van de eerste 64 bytes van de in de SDK 85 aanwezige ROM (type 8355) weergegeven.

Belangrijk:

U ziet, dat in fig. 14 een aantal adressen tussen accolades, b.v. {20C8}, staan. Er zijn nl. voor wat betreft de beginadressen van interrupt service routines twee verschillende uitvoeringen van de SDK 85 op de markt gebracht.

In de eerste uitvoering gelden de adressen, die niet tussen accolades staan. Deze adressen worden in de rest van deze cursus aangehouden. De adressen, die in de tweede uitvoering gelden, worden in deze les steeds tussen accolades vermeld, tenminste als deze adressen afwijken van die van de eerste uitvoering.
U dient dus te weten, welke van de adressen in uw systeem moeten worden aangehouden. Dit kunt u eenvoudig bepalen, door m.b.v. "SUBST MEM" de inhoud van het geheugenadres $003D_{16}$ te bekijken.

Als de inhoud van dit adres CE_{16} is, dan gelden in uw systeem de adressen van tabel 4. Is de inhoud van dit adres $D4_{16}$, dan moet u de adressen van tabel 5 gebruiken (dit zijn dus de adressen, die in deze les tussen accolades staan).

RST5	20C2
RST6	20C5
RST6.5	20C8
RST7	20CB
RST7.5	20CE

Tabel 4

RST5	{20C8}
RST6	{20CB}
RST6.5	{20CE}
RST7	{20D1}
RST7.5	{20D4}

Tabel 5

We zullen nu de interrupt mogelijkheden in de SDK 85 kort beschrijven.

a. RST0

Als in het objectprogramma de instructie RST0 (= $C7_{16}$) wordt uitgevoerd, dan wordt de programmateller gevuld met 0000_{16}. Op dit adres begint de monitor. Aangezien deze meer dan 8 bytes beslaat, staat op de adressen 0005_{16}, 0006_{16} en 0007_{16} de instructie JMP 01F1H. Dit is een sprong naar de rest van het monitor-programma.

b. RST1

Als de instructie RST1 (= CF_{16}) wordt uitgevoerd, dan wordt de programmateller gevuld met 0008_{16}. Op dit adres begint een programmadeel met de naam "SAVE REGISTERS", dat de inhouden van alle CPU-registers in het RAM-geheugen (adressen $20E9_{16}$ t/m $20F5_{16}$) veilig stelt. Van deze RST1-instructie kunt u nuttig gebruik maken, door deze aan het eind van uw programma te plaatsen (i.p.v. een HLT-instructie). Als uw programma is uitgevoerd, dan kunt u naderhand controleren, wat door uw programma in de registers is gewijzigd.

Het eerste deel van het programmadeel "SAVE REGISTERS" bevindt zich op de adressen 0008_{16} t/m 0020_{16}. Daarna volgt de instructie JMP 003FH. Dit is een sprong naar de rest van dit programmadeel, zodat de adressen 0024_{16} t/m $003E_{16}$ worden vrijgehouden voor de overige RST-instructies en interrupt requests.

Vraag 23: De RESTART-instructies ....., ... en ..... mogen niet worden gebruikt.

Omdat het programmadeel "SAVE REGISTERS" de adressen 0008_{16} t/m 0023_{16} in beslag neemt, mogen de instructies RST2, RST3 en RST4 niet worden gebruikt. Immers, de bij deze instructies behorende restart-adressen bevinden zich binnen dit programmadeel.

c. TRAP

Als de non-maskable interrupt request TRAP wordt geaccepteerd, dan wordt de programmateller gevuld met de waarde 0024_{16}. Op dit adres bevindt zich de instructie JMP 0157H. Dit is een sprong naar het SINGLE STEP-programma van de monitor.

De interrupt request TRAP wordt veroorzaakt door de timer uit de basic RAM 8155. Omdat deze timer door de monitor is toegewezen aan de single step-functie, mag u deze timer in uw programma niet gebruiken.

Vraag 24: De interrupt request TRAP wordt ook wel ..... genoemd.

De interrupt request TRAP wordt ook wel met RST4.5 aangeduid, omdat het bijbehorende restart-adres zich tussen die van RST4 en RST5 bevindt.

d. RST5

Als de instructie RST5 (= EF_{16}) wordt uitgevoerd, dan wordt de programmateller gevuld met 0028_{16}. Op dit adres bevindt zich een spronginstructie naar adres $20C2_{16}$ {$20C8_{16}$} in het RAM-geheugen. Op dit adres, en de twee volgende adressen, kunt u een spronginstructie plaatsen naar het adres van de door u gewenste interrupt service routine.

e. RST5.5

Door het accepteren van een interrupt request RST5.5 wordt de programmateller gevuld met $002C_{16}$. Op dit adres bevindt zich de instructie JMP 028EH. Dit is een sprong naar de interrupt service routine "ININT".

Antw.23: RST2; RST3; RST4. Antw.24: RST4.5.

Door deze ISR wordt de hexadecimale waarde van een op het toetsenbord ingedrukte toets op adres $20FE_{16}$ in het RAM-geheugen geplaatst.

De interrupt service routine "ININT" behoort in feite bij de monitor-subroutine "RDKBD". Deze subroutine blijft nl. in een programmalus rondgaan, totdat via RST5.5 en "ININT" adres $20FE_{16}$ is gevuld met de bij een ingedrukte toets behorende hexadecimale waarde.
Het laatste deel van "RDKBD" brengt de inhoud van $20FE_{16}$ over naar de accumulator.
RST5.5 wordt veroorzaakt door de keyboard display controller 8279.

Vraag 25: Om RST5.5 te laten accepteren, moet $b_0/b_1/b_2/b_3$ van het interrupt mask register worden gevuld met 0.
Hiervoor gebruiken we een RIM/SIM-instructie.

Om RST5.5 te kunnen accepteren, moet b_0 van het interrupt mask register worden gevuld met 0. We doen dit m.b.v. een SIM-instructie.
Daarmee wordt de inhoud van de accumulator naar het interrupt mask register overgebracht.

Vraag 26: $b_0/b_1/b_2/b_3$ van de accumulator moet dan 1 zijn.

We moeten dan zorgen, dat b_3 van de accumulator 1 is, want anders worden b_0, b_1 en b_2 van het interrupt mask register niet gewijzigd (zie paragraaf 9a). De te gebruiken instructies zijn dan b.v.

```
MVI A,08H
SIM
```

Als we voor de eerste keer in ons programma de monitor-subroutine "RDKBD" aanroepen, dan moeten deze instructies worden gebruikt om RST5.5 toe te staan (behalve als b_0 van het interrupt mask register al 0 is).

We roepen "RDKBD" b.v. aan met

```
MVI  A,08H
SIM
CALL RDKBD
```

Hierbij behoeft voor de CALL-opdracht geen EI-instructie te worden opgenomen. Deze staat nl. in de subroutine. Vlak voor de RET-instructie staat echter wel een DI-instructie. Als RST6.5 of RST7.5 ook moeten kunnen worden geaccepteerd, dan moet na de CALL-instructie wel een EI-instructie staan.

f. RST6

Als de instructie RST6 (= $F7_{16}$) wordt uitgevoerd, dan wordt de programmateller gevuld met 0030_{16}. Op dit adres bevindt zich een spronginstructie naar adres $20C5_{16}$ {$20CB_{16}$} in het RAM-geheugen. Op de adressen $20C5_{16}$, $20C6_{16}$ en $20C7_{16}$ {$20CB_{16}$, $20CC_{16}$ en $20CD_{16}$}, kunt u een spronginstructie plaatsen naar het begin van de bij RST6 behorende interrupt service routine.

g. RST6.5

Door het accepteren van een interrupt request RST6.5 wordt de programmateller gevuld met 0034_{16}. Op dit adres bevindt zich een spronginstructie naar adres $20C8_{16}$ {$20CE_{16}$} in het RAM-geheugen. Hier kunt u een spronginstructie naar de bijbehorende interrupt service routine plaatsen.

RST6.5 wordt veroorzaakt door de timer uit de expansion RAM 8155. (als het goed aangesloten is!)
Omdat u het beginadres van de bijbehorende interrupt service routine zelf kunt bepalen, kunt u in uw programma gebruik maken van deze timer.

h. RST7

Door het uitvoeren van de instructie RST7 (= FF_{16}) wordt de programmateller gevuld met 0038_{16}. Op dit adres bevindt zich een spronginstructie naar adres $20CB_{16}$ {$20D1_{16}$} in het RAM-geheugen. Hier kunt u een spronginstructie naar de bij RST7 behorende interrupt service routine plaatsen.

i. RST7.5

Een interrupt request RST7.5 wordt veroorzaakt door het indrukken van de toets "VECT INTR" (vector interrupt). Als RST7.5 wordt geaccepteerd, dan wordt de programmateller gevuld met $003C_{16}$. Vanaf hier vindt een sprong naar adres $20CE_{16}$ {$20D4_{16}$} plaats. Op deze adressen kunt u een spronginstructie naar de bij RST7.5 behorende interrupt service routine plaatsen.

Vraag 27: Om RST7.5 te kunnen accepteren, moeten de bits ..... en .... van het interrupt mask register worden gevuld met 0.

Als de interrupt request RST7.5 geaccepteerd moet worden, dienen b_4 en b_2 van het interrupt mask met 0 te worden gevuld (zie paragraaf 9a).

Opmerking:

RST0, RST1, RST5, RST6 en RST7 noemen we vaak software interrupts, omdat deze worden gerealiseerd door een instructie in het programma.

TRAP, RST5.5, RST6.5 en RST7.5 zijn hardware interrupts. Deze interrupt requests komen nl. via aparte lijnen op de CPU binnen.

Antw.27: b_4; b_2.

SAMENVATTING 8

28. In een objectprogramma in de SDK 85 mogen de instructies RST0, RST1, RST5, RST6 en RST7 worden gebruikt.
    De instructies RST2, RST3 en RST4 mogen in de SDK 85 niet worden toegepast.

29. TRAP (= RST4.5) is door de monitor voor de single step-functie gereserveerd.

30. RST5.5 dient voor de communicatie tussen CPU (8085) en keyboard display controller (8279).

31. RST6.5 wordt veroorzaakt door de timer uit de expansion RAM (8155).
    Deze timer kunt u in uw programma gebruiken.

32. RST7.5 wordt veroorzaakt door het indrukken van de toets "VECT INTR".

PROGRAMMAVOORBEELDEN-3.

Programmavoorbeelden - 3

1. INLEIDING

In deze les behandelen we twee programma's, die voornamelijk dienen om de stof uit voorgaande lessen te herhalen.

Elk voorbeeld bestaat uit probleemomschrijving, probleemanalyse, stroomdiagram en programma.
Probeer bij elk voorbeeld eerst zelf het programma te schrijven. Test (en verbeter) uw programma op de SDK 85. Bestudeer daarna de door ons gegeven oplossing.

De programma's hebben alle betrekking op de SDK 85. U kunt de gevraagde programma's echter ook voor andere computersystemen schrijven. U volgt dan steeds de in deze les gegeven ontwikkelingsmethode, maar gaat hierbij uit van de eigenschappen van een andere computer en eventueel een andere instructieset.

2. VOORBEELD 1

a. Probleemomschrijving:

Gegeven: Op het toetsenbord worden twee hexadecimale getallen ingetypt. Elk getal bestaat uit twee hexadecimale cijfers. Het meest significante cijfer wordt eerst ingetypt.

Gevraagd: Schrijf een programma, dat deze twee getallen met elkaar vermenigvuldigt en het product in hexadecimale vorm op de displays 1 t/m 4 weergeeft.

Aanvullende eisen t.b.v. de SDK 85:

a. Het programma moet beginnen op adres 2000_{16}.
b. Er moet zoveel mogelijk gebruik worden gemaakt van in vorige lessen beschreven subroutines.
c. De displays 5 en 6 en de decimale punten moeten uit zijn.

SCHRIJF NU ZELF HET GEVRAAGDE PROGRAMMA EN TEST DIT OP UW SDK 85.
BESTUDEER DAARNA ONZE OPLOSSING.

b. Probleemanalyse:

Uit de probleemomschrijving volgt, dat er achtereenvolgens moet gebeuren:
a. invoeren van twee getallen;
b. vermenigvuldigen van deze getallen;
c. uitvoeren van het resultaat.

We kunnen het probleem dus splitsen in drie deelproblemen (fig.1).
Voor elk deelprobleem gaan we bepalen, hoe dit opgelost kan worden.

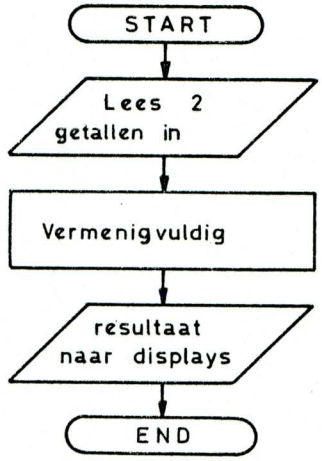


fig.1

Vraag 1: Voor het invoeren van de getallen gebruiken we subroutine .....

Bij het invoeren van de twee getallen maken we gebruik van de monitor-subroutine "RDKBD".
Deze subroutine plaatst de bij een ingedrukte toets behorende hexadecimale waarde in de accumulator.

Vraag 2: Om een volledig getal in te lezen, moet "RDKBD" 1/2/4 maal worden aangeroepen.

De getallen bestaan elk uit twee hexadecimale cijfers. Om een getal in te voeren, dient tweemaal een toets te worden ingedrukt.
"RDKBD" moet voor elk getal dus tweemaal worden aangeroepen.

We moeten de twee afzonderlijk ingetypte cijfers samenvoegen tot één getal. We duiden de 4 bits van het eerst ingetypte cijfer aan met XXXX, de bits van het tweede cijfer noemen we YYYY.

Vraag 3: Direct nadat "RDKBD" voor de eerste maal is uitgevoerd, is de inhoud van de accumulator .....
Direct na de tweede maal is de inhoud van de accumulator.....

Als "RDKBD" voor de eerste maal is uitgevoerd, het eerste cijfer is dus ingetypt, is de inhoud van de accumulator 0000XXXX. Nadat "RDKBD" voor de tweede maal is uitgevoerd, is de inhoud van de accumulator 0000YYYY geworden.

We kunnen het getal, dat uit twee cijfers wordt samengesteld, aangeven met XXXXYYYY. Het eerst ingetypte cijfer staat dus vóór het tweede.

Vraag 4: Na het invoeren van het eerste cijfer moeten de bits van dit cijfer 1/4/8 plaatsen naar links/rechts worden geschoven.

We moeten de cijfers van het eerste cijfer dus 4 plaatsen naar links schuiven en de bits van het tweede cijfer hierbij optellen (fig.2).

Antw.1: RDKBD. Antw.2: 2. Antw.3: 0000XXXX; 0000YYYY. Antw.4: 4; links.

```
0000XXXX ⟶    XXXX0000 (1e cijfer)
              0000YYYY (2e cijfer)
              +
              XXXXYYYY (getal)
```

fig.2

In fig.3a is het algemeen stroomdiagram voor het invoeren van één getal weergegeven.
Op deze wijze moeten er echter twee getallen worden ingevoerd. Het is dus zinvol om het invoeren van een volledig getal in een subroutine onder te brengen. Om twee getallen in te voeren wordt deze subroutine dan tweemaal aangeroepen.

De subroutine geven we b.v. de naam "INBYTE".
Fig.3a gaat dan over in fig.3b.

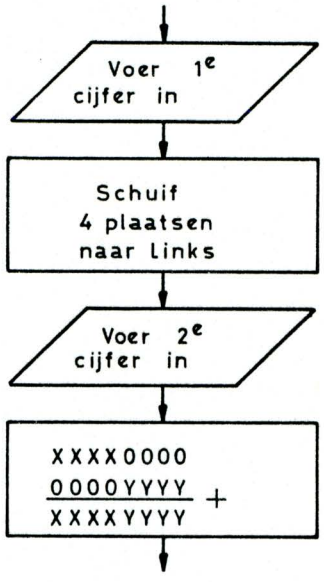


fig.3a

Door het tweemaal aanroepen van "INBYTE" worden twee getallen, elk van 8 bits, ingelezen. Deze getallen moeten met elkaar worden vermenigvuldigd.

Vraag 5: Voor het vermenigvuldigen maken we gebruik van subroutine ".....".

Voor het vermenigvuldigen maken we gebruik van de subroutine "MULTI", die in de les "Binaire vermenigvuldiging" is ontwikkeld.
Deze subroutine vermenigvuldigt twee 8-bits getallen tot een 16-bits resultaat. Dit is geheel overeenkomstig de probleemomschrijving. Dus ook het deelprobleem vermenigvuldigen is opgelost.

Het resultaat (product) moet in hexadecimale vorm op de displays 1 t/m 4 worden weergegeven.

Vraag 6: Hiervoor gebruiken we subroutine ".....".

De monitor-subroutine "UPDAD" beantwoordt precies aan deze eis, dus ook dit deelprobleem is opgelost.

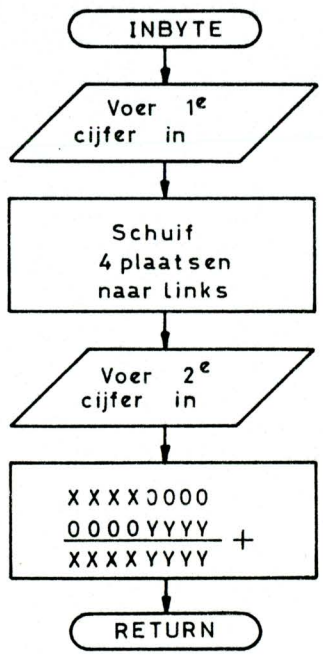


fig.3b

c. Stroomdiagram:

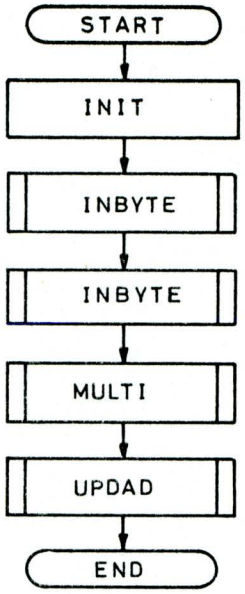


fig.4

Het algemeen stroomdiagram voor het gehele probleem is nu eenvoudig te tekenen, door de in de probleemanalyse gevonden deeloplossingen achterelkaar te plaatsen. Zo ontstaat fig.4. Hierin is tevens een blok voor de initialisatie aangebracht.

We moeten nu voor het programma en de subroutine "INBYTE" een gedetailleerd stroomdiagram ontwikkelen.

We beginnen met "INBYTE", omdat we de eigenschappen van deze subroutine moeten kennen, wanneer we deze vanuit het hoofdprogramma aanroepen.
Wat we van de subroutine moeten weten, is o.a. de parameteroverdracht, m.a.w. in welke registers of geheugenwoorden moeten we getallen plaatsen, voordat de subroutine wordt aangeroepen en waar worden door de subroutine de resultaten neergezet.
Verder moeten we weten welke registers door de subroutine worden gebruikt en hoe eventuele interrupts moeten worden afgehandeld.

We kunnen het stroomdiagram van fig.3b omzetten in dat van fig.5.

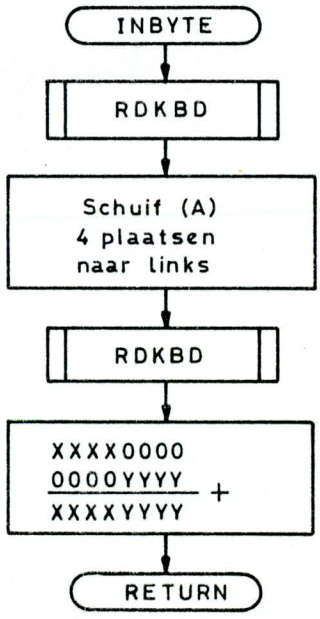


fig.5

Het eerst ingetypte cijfer wordt d.m.v. de subroutine "RDKBD" in de accumulator geplaatst en daarna 4 plaatsen naar links geschoven.

Vraag 7: Dit resultaat mag wel/niet in de accumulator blijven staan.

Het zo ontstane resultaat mag niet in de accumulator blijven staan, omdat het dan verloren zou gaan, als "RDKBD" voor de tweede maal wordt aangeroepen.
We brengen de inhoud van de accumulator daarom over naar een van de hulpregisters, b.v. register B.

Het gedetailleerd stroomdiagram van subroutine "INBYTE" is in fig.6b weergegeven.

Omdat "RDKBD" wordt aangeroepen, moet het interrupt mask register worden geinitialiseerd.

Antw.7: niet.

Vraag 8: "RDKBD" maakt gebruik van RST 5.5/6.5/7.5.

Om "RDKBD" te laten functioneren, moet interrupt request RST5.5 worden toegestaan (zie paragraaf 11e van de les "Interrupt").

Vraag 9: $b_2/b_1/b_0$ van het interrupt mask register moet 0 zijn.
b_3 moet 0/1 zijn.
Het interrupt mask register wordt gevuld m.b.v. de ..S.I.M...-instructie.

Het interrupt mask register wordt m.b.v. de SIM-instructie vanuit de accumulator gevuld. Daarbij moet gelden $b_0 = 0$, om RST5.5 toe te staan (enable RST5.5) en $b_3 = 1$, om b_2, b_1 en b_0 van het interrupt mask register te laten wijzigen (zie paragraaf 9a van de les "Interrupt").

Het initialiseren van het interrupt mask register moet plaatsvinden, voordat "INBYTE", dus "RDKBD", wordt aangeroepen.

We gaan nu het algemeen stroomdiagram van fig.4 omzetten in een gedetailleerd stroomdiagram.
We moeten er nl. voor zorgen, dat het resultaat van een subroutine door een volgende subroutine kan worden verwerkt.

Vraag 10: "INBYTE" plaatst een getal in .....
"MULTI" vereist dat er twee getallen in de registers .....
en ..... staan.

Door het tweemaal aanroepen van "INBYTE" wordt er tweemaal een getal in de accumulator geplaatst.
"MULTI" vermenigvuldigt twee getallen, die zich in de registers D en E bevinden. We moeten dus zorgen, dat het eerste getal in register D en het tweede getal in register E terechtkomt.

Vraag 11: "MULTI" plaatst het product van de twee getallen in registerpaar .....
"UPDAD" geeft de inhoud van registerpaar ..... op de displays weer.

Het product van de beide getallen wordt door "MULTI" in registerpaar D,E geplaatst. Door "UPDAD" wordt de inhoud van ditzelfde registerpaar op de displays weergegeven. Het product behoeft dus niet verplaatst te worden.

We moeten er wel voor zorgen, dat de displays 5 en 6 en de decimale punten uit zijn. De displays 5 en 6 zijn al uit, want ze worden door geen van de gebruikte subroutines aangestuurd.

Vraag 12: De decimale punt van display 4 zal uit zijn, als register .... met 0 wordt gevuld.

We dienen b_0 van register B 0 te maken. "UPDAD" dooft dan de decimale punt van display 4.

Antw.8: 5.5. Antw.9: b_0; 1; SIM. Antw.10: de accumulator; D; E.
Antw.11: D,E; D,E. Antw.12: B.

Door het aanbrengen van de nu gevonden voorwaarden in fig.4, ontstaat het gedetailleerd stroomdiagram van fig.6a.

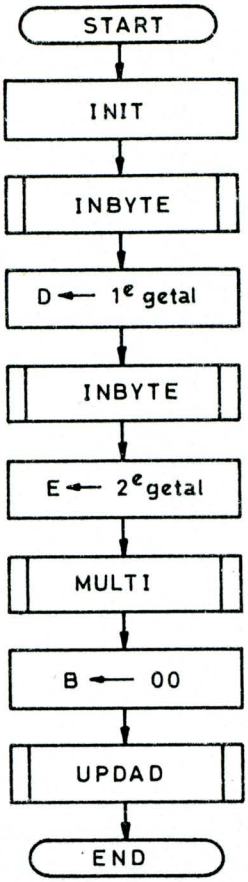


fig.6a

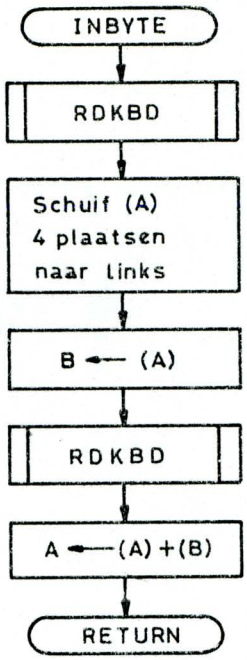


fig.6b

De initialisatie bestaat uit het vullen van het interrupt mask register en de stackpointer, b.v. met $20C2_{16}$. Dit is de maximale waarde, die is toegestaan als we de stack in de basic RAM-module onderbrengen.

d. Programma:

Uit fig.6a en fig.6b is nu rechtstreeks een programma te schrijven (fig.7).

ADRES	HEXADEC.	LABEL	MNEM.	OPERAND	COMMENT
			ORG	2000H	
		MULTI	EQU	2840H	
		UPDAD	EQU	0363H	
		RDKBD	EQU	02E7H	
		MASK	EQU	08H	
2000	3E 08		MVI	A,MASK	;RST5.5 WORDT
2002	30		SIM		; TOEGESTAAN
2003	31 C2 20		LXI	SP,20C2H	;INITIALISEER STACKPOINTER
2006	CD 17 20		CALL	INBYTE	;GETAL 1
2009	57		MOV	D,A	; NAAR D
200A	CD 17 20		CALL	INBYTE	;GETAL 2
200D	5F		MOV	E,A	; NAAR E
200E	CD 40 28		CALL	MULTI	;PRODUCT IN D,E
2011	06 00		MVI	B,00H	;DEC.PUNT UIT
2013	CD 63 03		CALL	UPDAD	;(D,E) NAAR DISPLAYS
2016	76		HLT		
2017	CD E7 02	INBYTE:	CALL	RDKBD	;EERSTE CIJFER
201A	07		RLC		;SCHUIF
201B	07		RLC		; (ACCU)
201C	07		RLC		; 4 PLAATSEN
201D	07		RLC		; NAAR LINKS
201E	47		MOV	B,A	
201F	CD E7 02		CALL	RDKBD	;TWEEDE CIJFER
2022	80		ADD	B	
2023	C9		RET		
			END		

fig.7

Test dit programma op uw SDK 85. Behalve de instructies uit fig.7, moet u natuurlijk ook de subroutine "MULTI" uit de les "Binaire vermenigvuldiging" in het RAM-geheugen plaatsen.

Opmerking:

De assembler directives ORG, EQU en END kunt u niet in het RAM-geheugen plaatsen. Dit zijn nl. aanwijzingen voor het assembly-vertaalprogramma, die noodzakelijk zijn, als het bronprogramma op een ontwikkelingssysteem wordt vertaald.
De assembler directives resulteren niet in machinecodes in het objectprogramma (zie de les "Syntax en subroutines" uit de cursus "Microprocessers/Microcomputers").

3. VOORBEELD 2

a. Probleemomschrijving:

Schrijf een programma met de naam "COUNT", dat aan de volgende eisen voldoet.

a. Een 16-bits software teller (b.v. een registerpaar) wordt ca. elke ms (milliseconde) met 1 verhoogd.
b. De inhoud van deze teller moet steeds op de LED's van de 2 output-poorten worden weergegeven.
c. Wanneer op de toets "VECT INTR" wordt gedrukt, moet de teller worden gevuld met de op de 2 input-poorten aangeboden waarde (16 bits). Daarna moet het tellen worden voortgezet, vanaf de nieuw ingevoerde waarde.

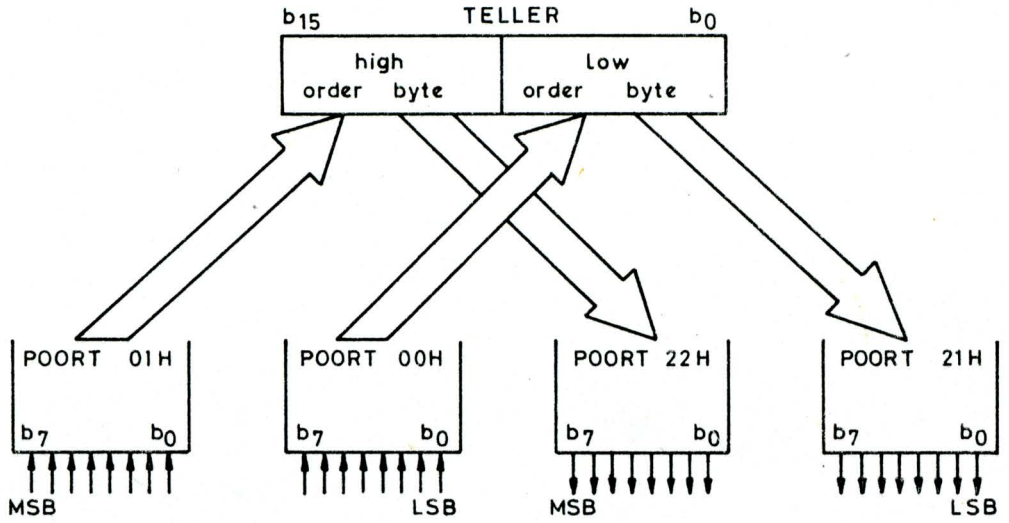


fig.8

Aanvullende eisen t.b.v. de SDK 85:

a. De tijd van ca. 1 ms, bij het verhogen van de teller, dient te worden gerealiseerd m.b.v. de timer uit de expansion RAM-I/O module.
b. Het programma moet beginnen op adres 2000_{16}.
c. De interrupt service routine voor de timer moet beginnen op adres 2800_{16}.
d. De interrupt service routine voor "VECT INTR" moet in het geheugen direct achter de interrupt service routine voor de timer staan.
e. Geen van beide interrupt service routines mag worden onderbroken door een nieuwe interrupt request.
f. De interrupt service routines krijgen de namen "TMINT" (= timer interrupt) en "VCINT" (= vector interrupt).
g. Direct na het starten van het programma moet de inhoud van de teller 0 zijn.

h. Er mogen geen andere, dan in deze probleemomschrijving genoemde, interrupt requests worden geaccepteerd of gehonoreerd.

SCHRIJF NU ZELF HET GEVRAAGDE PROGRAMMA EN TEST DIT OP UW SDK 85.
BESTUDEER DAARNA ONZE OPLOSSING.

b. Probleemanalyse:

Het te ontwikkelen programma valt in drie delen uiteen:
"COUNT", het hoofdprogramma.
"TMINT", de interrupt service routine voor het verhogen van de teller.
"VCINT", de interrupt service routine voor het vullen van de teller met een nieuwe waarde.

We zullen deze drie delen afzonderlijk behandelen.

"COUNT"

De functie van het hoofdprogramma is vrij eenvoudig. Na de gebruikelijke initialisatie moet het programma in een lus blijven rondgaan, totdat er een interrupt request wordt ontvangen. Na het afhandelen van een interrupt request moet weer in de bovengenoemde lus worden teruggekeerd.
Het algemeen stroomdiagram voor "COUNT" is in fig.9 weergegeven.

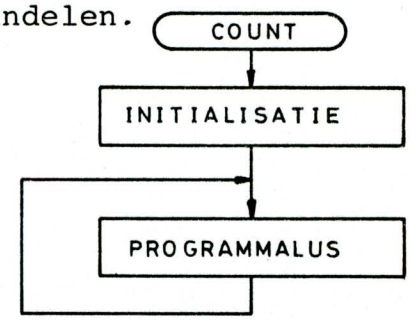


fig.9

De initialisatie heeft in dit probleem betrekking op de input- en output-poorten, de teller, de stackpointer, het toestaan van de juiste interrupt requests en het starten van de timer (fig.10).

De programmalus bestaat uit de instructies, die de tellerinhoud naar de output-poorten overbrengen.

Het initialiseren van de input-poorten 00_{16} en 01_{16} en de output-poorten 21_{16} en 22_{16} gebeurt op de in vorige lessen aangehouden wijze.

De stackpointer wordt b.v. gevuld met $20C2_{16}$, het hoogste adres, dat binnen de basic RAM-I/O module is toegestaan.
De teller moet worden gevuld met 0 (zie punt g van de probleemomschrijving).

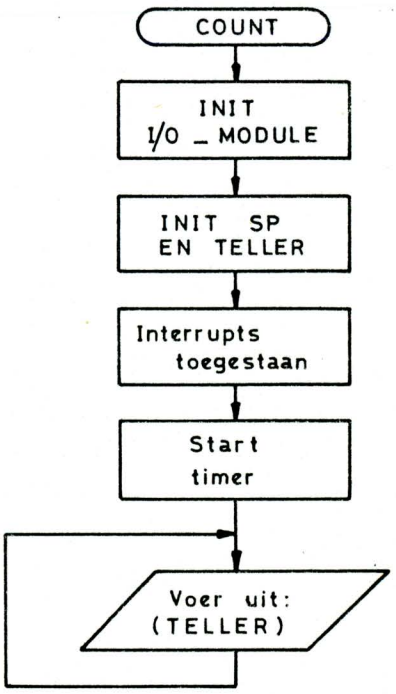


fig.10

Voor de teller moeten we nog een registerpaar of eventueel 2 geheugenwoorden aanwijzen. Omdat we nog niet weten, welke registers door de interrupt service routines worden gebruikt, wachten we hiermee tot deze routines zijn gedefinieerd.

Vraag 13: Het honoreren van de juiste interrupt requests bereiken we door het vullen van het ..... register.

Om het honoreren van interrupt requests toe te staan, moeten we het interrupt mask register vullen.

Vraag 14: In het interrupt mask register moet gelden:
b_0 = 0/1; b_1 = 0/1; b_2 = 0/1; b_3 = 0/1; b_4 = 0/1.

RST5.5 mag niet worden geaccepteerd. b_0 van het interrupt mask register moet dus 1 zijn (zie paragraaf 9a van de les "Interrupt").
RST6.5, afkomstig van de timer, moet wel kunnen worden geaccepteerd, dus b_1 moet 0 zijn.
Om RST7.5, afkomstig van de toets "VECT INTR", te kunnen accepteren, moet gelden $b_2 = b_4 = 0$. b_3 moet 1 zijn om b_0, b_1 en b_2 te kunnen wijzigen. We vullen het interrupt mask register dan met $00001001_2 = 09_{16}$.

Voordat de timer kan worden gestart, moet eerst het count length register van deze timer worden gevuld.

Vraag 15: Het count length register moet worden gevuld met een decimale waarde van ca. 1000/3000/6000.

De timer moet ca. 1 ms = 1000 µs = 3000 klokimpulsen blijven tellen. Het count length register dient dan te worden gevuld met de dubbele waarde, dus 6000 (zie paragraaf 8d van de les "Systeemeigenschappen hardware").

Hierbij verwaarlozen we de tijd, die door de interrupt service routine "TMINT" in beslag wordt genomen. Dit mag, omdat er in de probleemomschrijving ca. 1 ms is vermeld. (Als er gevraagd was naar exact 1 ms, dan had het aantal states van "TMINT" in mindering moeten worden gebracht op de waarde, die naar het count length register wordt gestuurd.)

In paragraaf 8d van de les "Systeemeigenschappen hardware" is precies beschreven, hoe de timer gestart en gestopt dient te worden en hoe we het count length register moeten vullen. Lees deze paragraaf nog eens goed door.

Als de timer is gestart, moet het programma in een lus blijven rondgaan, totdat het door interrupt wordt onderbroken.
In deze programmalus wordt de inhoud van de teller naar de outputpoorten gestuurd.

Het stroomdiagram, dat uit het voorgaande volgt, is in fig.11 weergegeven.

Antw.13: interrupt mask. Antw.14: 1; 0; 0; 1; 0. Antw.15: 6000.

"TMINT"

Als een interrupt request van de timer (RST6.5) wordt ontvangen, dan moet de interrupt service routine "TMINT" worden uitgevoerd.
In deze routine moet achtereenvolgens:
a. de timer worden gestopt.
b. de teller met 1 worden verhoogd.
c. de timer opnieuw worden gestart.
d. teruggekeerd worden naar het hoofdprogramma "COUNT".

Deze handelingen zijn direct in een stroomdiagram (fig.11) weer te geven.

"VCINT"

Als een interrupt request van de "VECT INTR"-toets (RST7.5) wordt ontvangen, dan moet de interrupt service routine "VCINT" worden uitgevoerd.
In deze routine moet achtereenvolgens
a. de timer worden gestopt.
b. de teller worden gevuld met de waarde, die op twee input-poorten wordt aangeboden.
c. de timer opnieuw worden gestart.
d. teruggekeerd worden naar het hoofdprogramma "COUNT".

Dit is weergegeven in het stroomdiagram van fig.11.

c. Stroomdiagram:

Uit de probleemanalyse volgt het stroomdiagram voor "COUNT"; "TMINT" en "VCINT".

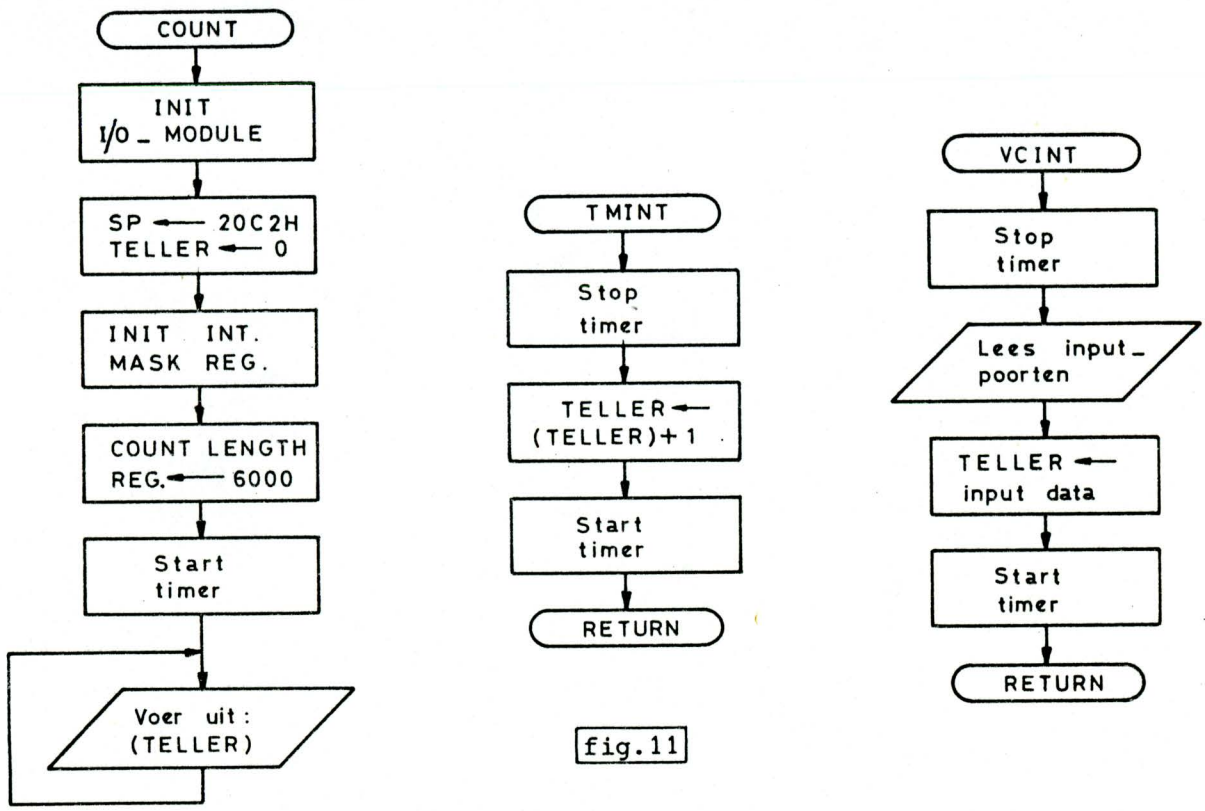


fig.11

Voordat we vanuit fig.11 het programma gaan schrijven, moeten we weten welk registerpaar of welke geheugenwoorden voor de teller gebruikt gaan worden. Daartoe bepalen we eerst welke CPU-registers nog vrij zijn.

Vraag 16: De input en output verloopt via de .....
Verder wordt er wel/niet beslag gelegd op andere registers.

Uit fig.11 blijkt, dat van de registers alleen de accumulator wordt gebruikt. Immers, via dit register verloopt elke input- en output-actie. Er is verder geen blok in fig.11 aanwezig, waarin één van de registers B, C, D, E, H of L wordt gebruikt.
Daarom kiezen we voor de teller een registerpaar. De inhoud hiervan is nl. met een INX-instructie eenvoudig te verhogen. We kiezen b.v. registerpaar B,C.

Als u registerpaar D,E of H,L kiest, denk er dan bij het intypen van het programma uit fig.12 aan, dat u alle instructies wijzigt, die betrekking hebben op de teller.

Omdat het de bedoeling is, dat het te schrijven programma op de SDK 85 wordt getest, moeten voor wat betreft het aanroepen van de interrupt service routines "TMINT" en "VCINT" nog extra voorzieningen worden getroffen.

Vraag 17: Het bij RST6.5 behorende restart-adres is $\ldots\ldots\ldots_{16}$.
Bij RST7.5 behoort restart-adres $\ldots\ldots\ldots_{16}$.

Door de interne interrupt controller van de 8085 wordt na het honoreren van een interrupt request RST6.5 de programmateller gevuld met het restart-adres 0034_{16}. Op dit adres bevindt zich een sprongopdracht naar $20C8_{16}$ {$20CE_{16}$} in het RAM-geheugen. Hier moet u dus een sprongopdracht naar de bijbehorende interrupt service routine plaatsen, b.v. JMP TMINT.

Opmerking:

Zie voor de verklaring van het adres tussen accolades, paragraaf 11 van de les "Interrupt".

Bij RST7.5 behoort het restart adres $003C_{16}$. Hier bevindt zich een sprongopdracht naar adres $20CE_{16}$ {$20D4_{16}$} in het RAM-geheugen. Op deze, en de volgende twee locaties, dient de sprongopdracht naar de interrupt service routine "VCINT" te staan.

Een punt, waarop we nog moeten letten, is het feit, dat een interrupt service routine niet mag worden onderbroken door een nieuwe interrupt request. M.a.w. aan het begin van een interrupt service routine moet het accepteren van interrupt requests worden verboden. Aan het eind van een interrupt service routine moet dit weer worden toegestaan.

Vraag 18: Aan het begin van een interrupt service routine moet wel/geen DI-instructie worden opgenomen.
Aan het eind van een interrupt service routine moet wel/geen EI-instructie worden opgenomen.

Antw.16: accumulator; niet. Antw.17: 0034; 003C.
Antw.18: geen; wel.

Na het honoreren van een interrupt request wordt de interrupt enable flipflop (zie de les "Interrupt") gereset.
We hoeven aan het begin van een interrupt service routine dus niet een DI-instructie te plaatsen.
Aan het eind van de interrupt service routines moet wel een EI-instructie staan, om interrupts weer toe te laten.

d. Programma

Uit het stroomdiagram van fig.11 en de daarbij behorende toelichtingen, ontstaat het programma van fig.12.

Let hierin vooral op het gebruik van de symbolische namen en de assembler directives.

ADRES	HEXADEC.	LABEL	MNEM.	OPERAND	COMMENT
			ORG	2000H	
		IMASK	EQU	09H	
		CLRHI	EQU	57H	
		CLRLO	EQU	70H	
		START	EQU	C0H	
		STOP	EQU	40H	
2000	3E 03	COUNT:	MVI	A,03H	;INITIALISATIE
2002	D3 20		OUT	20H	;POORT 21H = OUTPUT
2004	32 FF 20		STA	20FFH	;POORT 22H = OUTPUT
2007	AF		XRA	A	
2008	D3 02		OUT	02H	;POORT 00H = INPUT
200A	D3 03		OUT	03H	;POORT 01H = INPUT
200C	31 C2 20		LXI	SP,20C2H	
200F	01 00 00		LXI	B,0000H	;INIT TELLER op 0000
2012	3E 09		MVI	A,IMASK	;RST6.5 EN
2014	30		SIM		; RST7.5 WORDEN
2015	FB		EI		; TOEGESTAAN
2016	3E 57		MVI	A,CLRHI	;VUL
2018	D3 2D		OUT	2DH	; COUNT
201A	3E 70		MVI	A,CLRLO	; LENGTH
201C	D3 2C		OUT	2CH	; REGISTER
201E	3E C0		MVI	A,START	;START REGISTER A
2020	D3 28		OUT	28H	; TIMER
2022	78	LUS:	MOV	A,B	
2023	D3 22		OUT	22H	;VOER UIT
2025	79		MOV	A,C	; (TELLER)
2026	D3 21		OUT	21H	
2028	C3 22 20		JMP	LUS	
			ORG	2800H	
2800	3E 40	TMINT:	MVI	A,STOP	;STOP
2802	D3 28		OUT	28H	; TIMER
2804	03		INX	B	;VERHOOG TELLER
2805	3E C0		MVI	A,START	;START
2807	D3 28		OUT	28H	; TIMER
2809	FB		EI		
280A	C9		RET		
280B	3E 40	VCINT:	MVI	A,STOP	;STOP
280D	D3 28		OUT	28H	; TIMER
280F	DB 01		IN	01H	;VUL HIGH ORDER
2811	47		MOV	B,A	; BYTE VAN TELLER
2812	DB 00		IN	00H	;VUL LOW ORDER
2814	4F		MOV	C,A	; BYTE VAN TELLER
2815	3E C0		MVI	A,START	;START
2817	D3 28		OUT	28H	; TIMER
2819	FB		EI		
281A	C9		RET		
			ORG	20C8H	;{20CEH}
20C8	C3 00 28		JMP	TMINT	
			ORG	20CEH	;{20D4H}
20CE	C3 0B 28		JMP	VCINT	
			END		;TERUG NAAR MONITORPROGRAMMA

fig.12

e. Testen

Test dit programma op uw SDK 85. Hierbij zult u ervaren, dat de hoge frequentie, waarmee wordt geteld, het bijzonder moeilijk maakt om te controleren of er correct wordt geteld en of de teller na het indrukken van "VECT INTR" ook daadwerkelijk met de juiste waarde wordt gevuld.

Een oplossing hiervoor is tijdens het testen gebruik te maken van de monitor-subroutine "DELAY", om de telfrequentie te verlagen.

Vraag 19: De instructie CALL DELAY moet dan worden opgenomen in het hoofdprogramma/"TMINT"/"VCINT".

We moeten "DELAY" dan vanuit de interrupt service routine "TMINT" laten aanroepen. Daarin wordt immers de timer gestopt, de teller verhoogd en de timer direct weer gestart. Tussen het stoppen en starten van de timer laten we dan "DELAY" aanroepen. Dit mag zowel direct voor als direct na de instructie INX B.

Denk eraan, dat door het tussenvoegen van b.v. de instructies LXI D,FFFFH en CALL DELAY de instructies 6 plaatsen dienen te worden opgeschoven.

De interrupt service routine "VCINT" zal dan op adres 2811_{16} beginnen, zodat u tevens de instructie JMP VCINT op adres $20CE_{16}$ {$20D4_{16}$} moet wijzigen.

PROGRAMMEERBARE CHIPS.

Programmeerbare chips

1. INLEIDING

Microcomputers hebben sinds hun introductie in veel applicaties (= toepassingen) de z.g. "wilde logica" vervangen. Onder wilde logica verstaan we uit losse delen opgebouwde digitale schakelingen, die bepaalde functies verrichten, maar geen regelmatige structuur hebben.

Door het toepassen van microprocessoren veranderde dit aanzienlijk. Het hart van de wilde logica werd vervangen door een systeem met een ordelijke structuur. Een bussysteem waaraan de microprocessor, ROM, RAM en I/O-poorten verbonden waren, maakte een ontwerp veel overzichtelijker.

Echter niet alle wilde logica verdween. Veel schakelingen, die zich altijd rondom het microprocessorsysteem bevinden, bleven over. Dit waren voornamelijk de schakelingen, die de interface met de buitenwereld of het proces verzorgden. Deze lieten zich nl. niet in de structuur van een microprocessorsysteem inpassen. Ze kwamen óf te weinig voor óf verschilden telkens van applicatie tot applicatie. Dit weerhield een tijd lang het omzetten van deze schakelingen in één z.g. custom LSI chip. Door het toepassen van custom LSI chips zou men tot een geringer aantal componenten kunnen komen, waardoor een eenvoudiger en overzichtelijker structuur zou ontstaan.

Een custom LSI chip is een geïntegreerde uitvoering van een speciale schakeling, die door de ontwerper eerst uit losse elementen is opgebouwd. Alleen bij zeer grote aantallen is het economisch verantwoord om een willekeurige schakeling te integreren in een chip. (Dit was ook het bestaansrecht van de microprocessor, omdat deze in grote aantallen in veel verschillende toepassingen voorkwam.)

Om dus de resterende wilde logica om te zetten in één of slechts enkele chips, moest men dus de aantallen verhogen. Dit gebeurde na verloop van tijd als gevolg van twee oorzaken:

1. Een groot aantal applicaties kon veel goedkoper m.b.v. een microprocessor worden gerealiseerd, waardoor de benodigde aantallen chips vanzelf toenamen. Hierdoor werd het toepassen van een custom LSI chip voor de rest van de schakeling gerechtvaardigd.
2. Langzamerhand ontstond de gedachte, om een aantal veel voorkomende schakelingen, die vrijwel nooit tegelijkertijd werkzaam moeten zijn, samen te voegen. Door besturingssignalen zou men de gewenste schakeling dan kunnen activeren en tegelijkertijd de overige schakelingen het zwijgen opleggen. Een dergelijke samenvoeging zou men dan in een chip kunnen integreren. Omdat men dan kan kiezen uit één bepaalde schakeling in de chip, is zo'n chip in uiteenlopende applicaties te gebruiken, waardoor de toe te passen aantallen enorm worden verhoogd.

Door de genoemde oorzaken werd het fabriceren en toepassen van custom LSI chips een economisch haalbare kaart.

In deze les bespreken we twee van deze custom LSI chips, nl. de ROM-I/O chip 8355 en de RAM-I/O-timer chip 8155. De keyboard display controller

8279 komt in de les "Interfacing-2" aan de orde.

Custom LSI chips worden vaak programmeerbare chips of multifunctional chips genoemd. Programmeerbaar omdat software-matig een bepaalde functie binnen de chip is te selecteren. Multifunctional omdat de chip een aantal functies bevat.

2. OPBOUW VAN EEN CUSTOM LSI CHIP

In een custom LSI chip moet aan bepaalde voorwaarden worden voldaan.

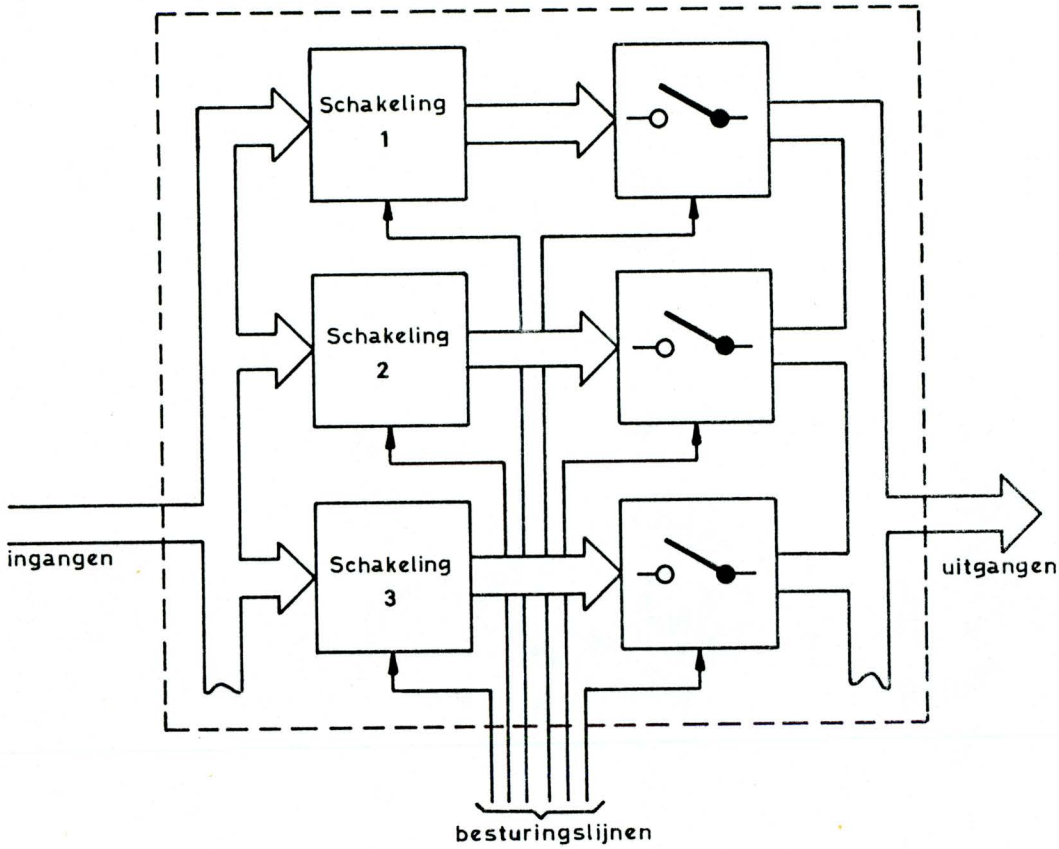


fig.1

De schakelingen binnen de chip gebruiken alle dezelfde pins (= aansluitpunten van de chip). De uitgangen van elke schakeling moeten tri-state worden uitgevoerd, zodat de uitgangssignalen van de niet-geselecteerde schakelingen de uitgangssignalen van de wel-geselecteerde schakeling niet verstoren (fig.1).

De ingangen van de afzonderlijke schakelingen mogen natuurlijk wel zijn doorverbonden. Ook mogen de in- en uitgangen met elkaar worden doorverbonden (het principe van de bi-directionele databus), mits de schakelingen elkaar maar niet beïnvloeden.

Een andere voorwaarde is, dat een dergelijke chip op eenvoudige wijze in een microprocessorsysteem moet kunnen worden opgenomen. D.m.v. input- en

output-poorten moet de chip op het bussysteem van de microcomputer worden aangesloten. Via deze I/O-poorten kunnen data en besturingssignalen worden uitgewisseld.
Eén of meer output-poorten dienen tevens om besturingssignalen vast te houden, zodat een bepaalde schakeling gedurende een zekere tijd kan worden geselecteerd. D.m.v. instructies, dus software-matig, kunnen we deze output-poort(en) met énen en nullen vullen, m.a.w. we programmeren de chip.

Als nu ook de benodigde I/O-poorten in de custom LSI chip worden ondergebracht, dan verkrijgt men één enkele chip, die rechtstreeks op het bussysteem kan worden aangesloten én tevens de taken van de wilde logica, b.v. de interface met de buitenwereld, overneemt.

Deze methode is voor het eerst toegepast in een chip, die alleen een I/O-module bevatte. Voor elk van de hierin aanwezige I/O-lijnen kon gekozen worden of deze als input- of als output-lijn dienst moest doen. Later is men veel verder gegaan en heeft men ook complete interfaces voor verschillende soorten visual display units (= beeldschermeenheden), floppy disk controllers, enz., op één chip ondergebracht.
Deze chips behoeven niet meer op één, strikt door de fabrikant bepaalde, manier te worden gebruikt. Zij kunnen geprogrammeerd worden door middel van software, waardoor de schakelaars (uitgangen van ingebouwde output-poorten) een bepaalde stand krijgen. We spreken hier van programmeerbare chips. Ze worden in steeds grotere aantallen gebruikt, vooral omdat men erin slaagt telkens meer functies in één chip samen te voegen.
Sommige van deze programmeerbare chips zijn complexer van opbouw dan de meeste 8-bits microprocessoren.

Naast deze trend om wilde logica zoveel mogelijk te vervangen en in een nette structuur onder te brengen, is er nog een andere. Het aantal componenten, waaruit een compleet microcomputersysteem is opgebouwd, wil men steeds verder terugbrengen. Uiteindelijk leidt dit tot de single chip microcomputers. Deze hebben echter maar beperkte mogelijkheden.
Een tussenvorm is dat men b.v. het geheugen en enkele I/O-functies bij elkaar neemt. Dit is b.v. gedaan in de ROM-I/O chip 8355 en de RAM-I/O-timer chip 8155. Samen met de microprocessor 8085 vormen ze een complete microcomputer, bestaande uit CPU, RAM, ROM en I/O-module.

Het voordeel van deze methode voor de ontwerper is, dat hij zowel voor kleine als grote applicaties van dezelfde microprocessor gebruik kan maken en niet voor elk project een nieuwe machinegerichte taal behoeft aan te leren.
Tussen de meeste single chip microcomputers en "gewone" microprocessoren bestaan nl. grote verschillen in architectuur en programmering.

De genoemde multifunctional chips bieden dus vooral voor kleine systemen grote voordelen. Met enkele chips is een complete microcomputer te realiseren.
Het I/O-gedeelte moet dan wel flexibel toegepast kunnen worden. Dit is dan ook programmeerbaar uitgevoerd. Dit soort chips bevat én een programmeerbaar I/O-gedeelte én een geheugenfunctie. Het programmeerbare I/O-gedeelte kan d.m.v. input- en output-instructies (I/O mapped I/O) of d.m.v. geheugentransportinstructies (memory mapped I/O) worden aangesproken.

SAMENVATTING 1

1. Onder wilde logica verstaan we de uit losse elementen opgebouwde schakelingen, die oorspronkelijk noodzakelijk waren om de interface tussen microprocessorsysteem en buitenwereld te verzorgen.

2. Door in één chip een aantal uiteenlopende schakelingen onder te brengen, is de wilde logica vrijwel geheel vervangen door custom LSI chips.

3. In veel toepassingen verzorgt een custom LSI chip de interface tussen het microprocessorbussysteem en de buitenwereld.

4. Door een aantal verschillende functies in één custom LSI chip onder te brengen, kan een complete microcomputer d.m.v. slechts enkele chips worden gerealiseerd.

3. DE ROM-I/O-chip 8355

Deze ROM-I/O chip is speciaal ontwikkeld voor toepassing in 8085-systemen. De 8355 is nl. volledig aangepast aan de gemultiplexte data/adresbus. Verder werkt de 8355 met dezelfde status- en besturingssignalen als de 8085.

Vraag 1: In de 8355 zitten ..... bytes ROM en ..... 8-bits I/O-poorten.

In de 8355 bevinden zich 2048 bytes = 2K bytes ROM en 16 I/O-lijnen, die naar believen als input- of als output-lijn kunnen fungeren. Dit, en het feit dat er vrijwel geen extra logica nodig is om deze chip op het bussysteem aan te sluiten, maakt de toepassing in kleine systemen erg economisch.

De 8355 is in een 40-pins dual in line-behuizing ondergebracht. In fig.2 zijn de aansluitmogelijkheden van deze chip weergegeven.

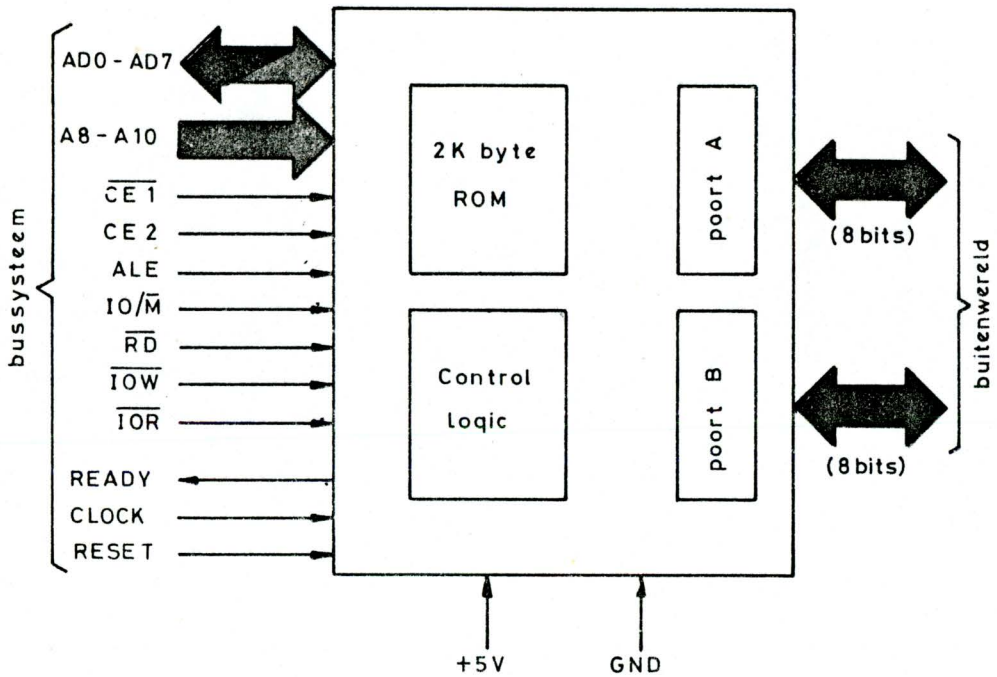


fig.2

Door de control logic worden de van het bussysteem afkomstige signalen omgezet in de juiste interne besturingssignalen. We bespreken nu achtereenvolgens
a. Control logic (paragraaf 4)
b. ROM-gedeelte (paragraaf 5)
c. I/O-gedeelte (paragraaf 6).

Daarna bespreken we in paragraaf 7 op welke wijze de 8355 in de SDK 85 is opgenomen.

4. CONTROL LOGIC (8355)

In fig.3 is een gedetailleerder blokschema van de control logic (besturingslogica) weergegeven.

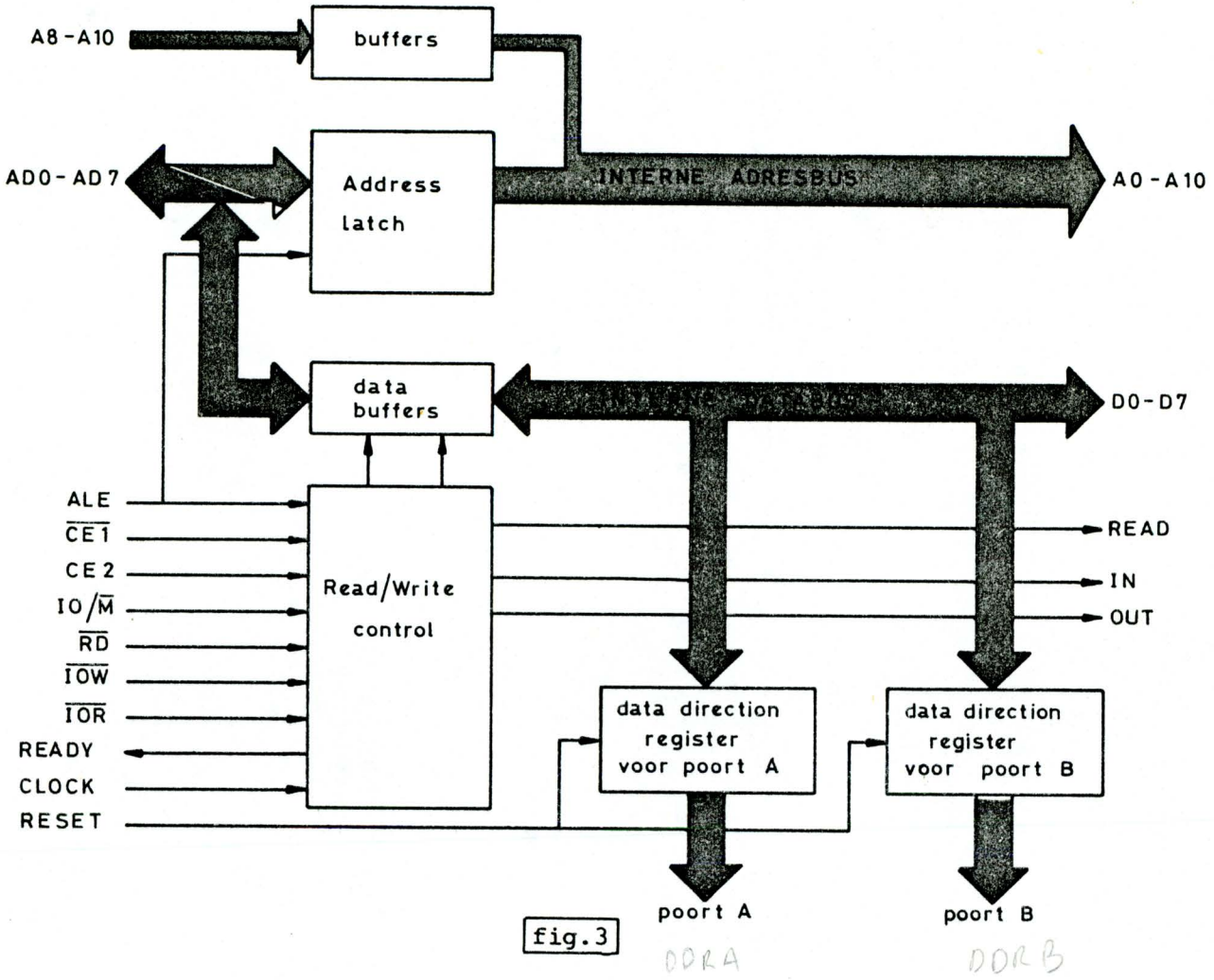


fig.3

We gaan nu in op de afzonderlijke delen van de control logic.

a. Address latch

De address latch dient om onder besturing van het ALE-signaal de data/adresbus te demultiplexen (zie paragraaf 6 van de les "Systeemeigenschappen hardware"). De 8 uitgangen van de address latch vormen de low order byte van de interne adresbus. Omdat er binnen de 8355 slechts 11 adreslijnen noodzakelijk zijn, worden van de high order byte alleen de lijnen A8, A9 en A10 via buffers hieraan toegevoegd. De interne adresbus bestaat dus uit de lijnen A0 t/m A10.

b. Databuffers

De databuffers zijn bi-directionele tri-state buffers. Door de read/write logic worden ze alleen in geleidende toestand gebracht, op het moment dat er data van de interne op de externe databus (of omgekeerd) moet worden geplaatst. In de overige gevallen verkeren de databuffers in de high impedance (floating) toestand.

c. Data direction registers

Dit zijn de command registers voor de I/O-poorten A en B. Elk data direction register bestaat uit 8 flip flops. M.b.v. een output- of een memory write-instructie kan een data direction register worden gevuld. Omdat de data direction registers rechtstreeks met de I/O-poorten zijn verbonden, gaan we hierop in paragraaf 6 dieper in.

d. Read/write logic

De taak van deze schakeling is het omzetten van de ontvangen besturingssignalen in de intern benodigde READ-, IN- en OUT-commando's. Deze schakeling geeft alleen interne besturingssignalen af als voor wat betreft de chip enable-signalen $\overline{CE1}$ en CE2 aan bepaalde voorwaarden is voldaan.

Vraag 2: Om de 8355 te selecteren moet gelden $\overline{CE1}$ = 0/1 en CE2 = 0/1.

Alleen als $\overline{CE1}$ = 0 én CE2 = 1, kan de read/write control interne besturingssignalen afgeven en hiermee de 8355 laten functioneren.

Als niet aan de genoemde voorwaarden is voldaan, worden er geen interne besturingssignalen opgewekt en worden de databuffers in de high impedance-toestand gebracht. M.a.w. de 8355 is dan niet geselecteerd.

De interne besturingssignalen worden als volgt opgewekt:

READ: Er wordt een READ-commando opgewekt als IO/$\overline{M}$ = 0 en er een $\overline{RD}$-signaal van de CPU wordt ontvangen. Het adres op de lijnen A0 t/m A10 bepaalt dan, uit welke ROM-locatie data op de externe databus wordt geplaatst.

OUT: Er wordt een OUT-commando opgewekt, als er een signaal (0 = actief, 1 = rusttoestand) op de $\overline{IOW}$-ingang wordt ontvangen. Hierbij wordt geen rekening gehouden met de toestand van IO/$\overline{M}$. Dit is ook niet noodzakelijk, omdat een ontvangen write-signaal natuurlijk nooit betrekking kan hebben op het ROM-gedeelte van de chip.

IN: Een IN-commando kan door twee oorzaken worden opgewekt.
1. Als IO/$\overline{M}$ = 1 (dus het I/O-gedeelte wordt geselecteerd) en er wordt een $\overline{RD}$-signaal van de CPU ontvangen.
2. Als er een signaal op de $\overline{IOR}$-ingang (0 = actief, 1 = rusttoestand) wordt ontvangen. Hierbij heeft de toestand van IO/$\overline{M}$ geen invloed.

Conclusies:

1. Het $\overline{RD}$-signaal kan fungeren als memory read (bij IO/$\overline{M}$ = 0) of als I/O read, dus als input-commando (bij IO/$\overline{M}$ = 1).

2. $\overline{IOR}$ en $\overline{IOW}$ werken als read (IN) resp. write (OUT) voor het I/O-gedeelte. Hierbij heeft IO/$\overline{M}$ geen invloed.

Deze constructie houdt in, dat het I/O-gedeelte binnen de 8355 zowel volgens memory mapped I/O als I/O mapped I/O kan worden aangesproken.

Behalve de interne besturingssignalen, geeft de read/write logic ook een ready signaal t.b.v. de CPU af. Dit ready signaal is hoog gedurende de tijd, die de adresselectieschakeling in het ROM-gedeelte nodig heeft, om een geadresseerd geheugenwoord te selecteren. (Dit is dus de access-tijd).

SAMENVATTING 2

5. De ROM-I/O chip 8355 bevat
   a. control logic,
   b. een ROM-gedeelte,
   c. een I/O-gedeelte.

6. De control unit bestaat uit
   a. een demultiplexer latch voor de data/adresbus,
   b. bi-directionele tri-state buffers t.b.v. de databus,
   c. read/write logic, voor het opwekken van de interne besturingssignalen en een READY-signaal,
   d. 2 data direction registers voor de initialisatie van de 16 I/O-lijnen.

7. De 8355 is geschikt voor zowel memory mapped I/O als I/O mapped I/O.

5. HET ROM-GEDEELTE (8355)

Het ROM-gedeelte van de 8355 heeft een capaciteit van 2048 = 2K bytes.

Vraag 3: Hiervoor zijn ..... adreslijnen noodzakelijk.

Om deze 2048 geheugenwoorden van elkaar te kunnen onderscheiden, zijn 11 adreslijnen noodzakelijk (immers $2048 = 2^{11}$). Dit zijn de lijnen A0 t/m A10 van de interne adresbus. Als IO/$\overline{M}$ laag is en het $\overline{RD}$-signaal is actief, dan wordt via de interne databus de inhoud van het geadresseerde geheugenwoord aan de externe databus afgestaan.
De snelheid waarmee dit gebeurt, is geheel aangepast aan de snelheid van de microprocessor 8085.
De accesstijd is 400 ns voor gebruik bij de standaard 8085. Een speciale versie, de 8355-2, heeft een accesstijd van 300 ns, zodat deze zonder wait states te introduceren, met de snellere 8085-2 kan samenwerken.
Het ROM-gedeelte is mask programmeerbaar en dus alleen voor grote series interessant. Een UV-wisbare versie is de 8755, die alle eigenschappen van de 8355 bezit, plus de mogelijkheid om meer malen te wissen en te programmeren.

6. HET I/O-GEDEELTE (8355)

In fig.4 is het blokschema van het I/O-gedeelte van de 8355 getekend.

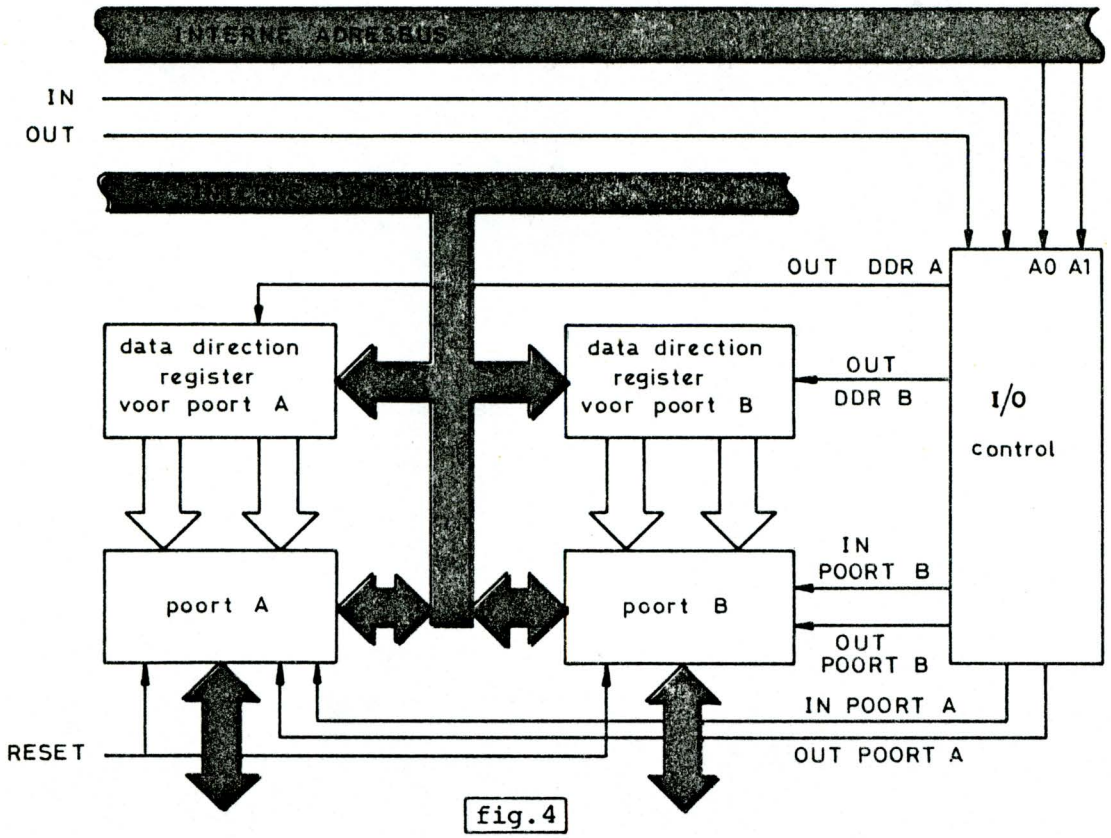


fig.4

In dit I/O-gedeelte bevinden zich o.a. 2 data direction registers en 2 I/O-poorten.

Vraag 4: Om deze locaties te kunnen adresseren, zijn ..... adreslijnen nodig.

Dit zijn de 4 locaties, die door de gebruiker moeten kunnen worden geadresseerd. Hiervoor zijn 2 adreslijnen (want $2^2 = 4$) noodzakelijk. In de 8355 worden hiertoe A0 en A1 van de interne adresbus gebruikt. De I/O control genereert m.b.v. deze twee adreslijnen en de IN- en OUT-signalen de benodigde besturingssignalen voor de I/O-poorten en data direction registers. Deze signalen zijn in tabel 1 vermeld.

IN/OUT	A1	A0	besturingssignaal
-	X	X	-
OUT	0	0	OUT POORT A
OUT	0	1	OUT POORT B
OUT	1	0	OUT DDR A
OUT	1	1	OUT DDR B
IN	0	0	IN POORT A
IN	0	1	IN POORT B
IN	1	0	-
IN	1	1	-
- = geen actief signaal aanwezig X = don't care			

Tabel 1

Uit tabel 1 zijn de volgende conclusies te trekken.

1. Omdat alleen A0 en A1 voor de adresselectie in het I/O-gedeelte worden gebruikt, zijn de overige lijnen (A2 t/m A10) van de interne adresbus don't cares.

2. De data direction registers kunnen alleen met een waarde worden gevuld. Ze kunnen niet worden uitgelezen.

In fig.5 is weergegeven, hoe we met een bit van een data direction register kunnen kiezen of een bepaalde I/O-lijn als input- of als outputlijn moet fungeren.
Fig.5 geldt voor b_0 van poort A en de overeenkomstige bit van het bijbehorende data direction register. Deze schakeling komt in de 8355 totaal dus 16 maal voor.

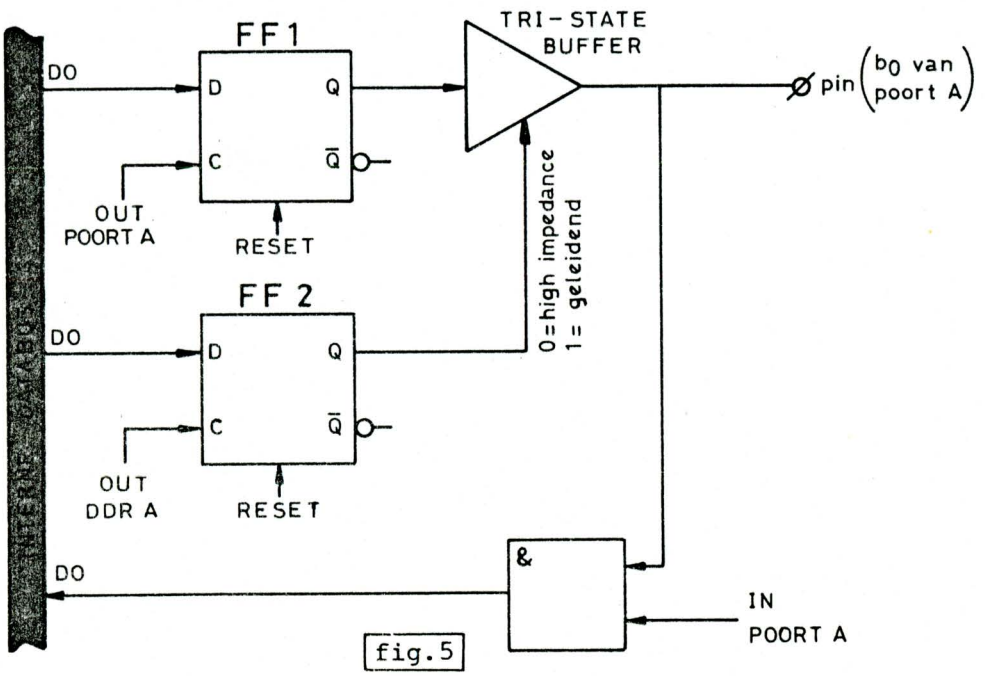


fig.5

De beide flip flops zijn zo geconstrueerd, dat door een impuls op de C-ingang de informatie van de D-ingang wordt opgeslagen en op de Q-uitgang verschijnt.

Vraag 5: Door een reset-impuls wordt b_0 van poort A 0/1/high impedance. Poort A werkt dan als input/output-poort.

Door een reset-impuls wordt de Q-uitgang van FF2 0 en wordt de tri-state buffer in de high impedance toestand gebracht. Wanneer er nu een commando IN POORT A optreedt, dan wordt de op b_0 aangeboden data via de EN-poort op D0 van de interne databus geplaatst.
Dit gebeurt ook voor de overige 7 bits van poort A. Na een reset-impuls worden de poorten A en B dus als input-poorten ingesteld.

Om b.v. b_0 van poort A als output-lijn te laten functioneren, moet FF2 worden geset. Dit kunnen we bereiken, door via de interne adresbus het data direction register voor poort A te selecteren en een commando OUT DDR A af te laten geven.

Vraag 6: D0 van de interne databus moet dan 0/1 zijn.

Omdat de Q-uitgang van FF2 dan 1 moet zijn (om de tri-state buffer te laten geleiden), moet D0 van de interne databus 1 zijn. Deze bit wordt nl. door het commando OUT DDR A in FF2 opgeslagen.

Dit zou b.v. te bereiken zijn met de instructies

```
MVI   A,FFH
OUT   DDRA
```

Hierin is DDRA de symbolische naam voor het poortnummer, waarbij het data direction register voor poort A wordt geselecteerd.

Vraag 7: Als de uitvoering van deze twee instructies direct op een reset impuls volgt, dan wordt door poort A de combinatie $\ldots\ldots\ldots_2$ afgegeven.

Door een reset-impuls worden zowel FF1 als FF2 gereset. Als de genoemde instructies zijn uitgevoerd, wordt door de tri-state buffer de toestand van de Q-uitgang van FF1 aan de pin (b_0 van poort A) doorgegeven.
Op deze (output) lijn verschijnt dus een 0 (FF1 is immers gereset).
Hetzelfde gebeurt op de overige 7 lijnen van poort A. Er wordt dus de combinatie $00000000_2 = 00_{16}$ afgegeven.
M.b.v. output-instructies kunnen we nu de gewenste data via deze output-poort uitvoeren.

Uit fig.5 zijn nog de volgende conclusies te trekken:

1. Als lijnen van een poort als output-lijn werken, dan kan d.m.v. een input-instructie worden getest, welke data zich op de output-lijnen bevindt. Immers door een commando IN POORT A wordt via de EN-poort in fig.5 altijd de toestand van b_0 op de interne databus geplaatst. Zo ook bij de overige 15 lijnen van de 2 I/O-poorten.

Antw.5: high impedance; input. Antw.6: 1. Antw.7: 00000000.

2. Voordat een poort als output-poort wordt geïnitialiseerd, kunnen we de poort vullen met een bepaalde bitcombinatie.
Als de poort daarna als output-poort wordt ingesteld (m.a.w. de tri-state buffers worden in de geleidende toestand gestuurd), dan wordt direct nadat de high impedance toestand (door een reset-impuls) eindigt, deze combinatie afgegeven.
Dit is vooral van belang bij het besturen van snelle processen.
We zijn dan nl. in staat om een output-lijn direct van de high impedance toestand in de 1-toestand over te laten gaan.

SAMENVATTING 3

8. Het ROM-gedeelte van de 8355 heeft een capaciteit van 2K bytes.

9. Het I/O-gedeelte bevat 2 I/O-poorten en 2 data direction registers.

10. Door het vullen van de data direction registers kan voor elk van de 16 I/O-lijnen afzonderlijk worden aangegeven of deze als input- of als output-lijn dienst moet doen.

11. Na een reset-impuls zijn alle 16 I/O-lijnen in de high impedance toestand gebracht en als input-lijn geïnitialiseerd.

12. De data direction registers kunnen niet worden uitgelezen.

13. De toestanden op de output-lijnen kunnen d.m.v. input-instructies weer door de CPU worden ingelezen.

7. DE 8355 IN DE SDK 85

In de voorgaande paragrafen zijn de eigenschappen van de 8355 behandeld. In deze paragraaf wordt beschreven hoe deze eigenschappen in de SDK 85 zijn benut. Hierin is de 8355 aangesloten volgens fig.6.

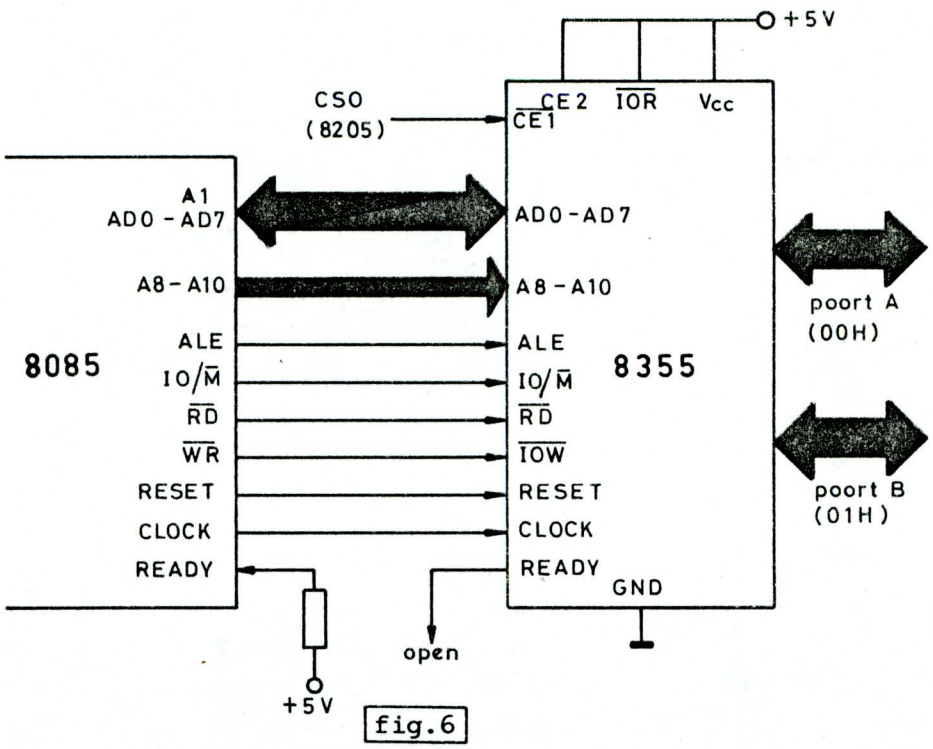


fig.6

Vraag 8: Het ROM-gedeelte in de SDK 8355 beslaat de adressen .....$_{16}$ t/m .....$_{16}$.

Het ROM-gedeelte wordt geselecteerd als geldt $\overline{CE1}$ = 0, CE2 = 1 en IO/$\overline{M}$ = 0. CE2 is vast met +5V verbonden, dus 1.
$\overline{CE1}$ is verbonden met de CS0-uitgang van de adresdecoder 8205. Deze lijn is 0, als zich op de adresbus het adres 00000XXXXXXXXXXX bevindt (X = don't care).
Het ROM-gedeelte wordt dus geselecteerd bij de uitvoering van een geheugentransportinstructie, waarin één van de adressen 0000000000000000_2 t/m 0000011111111111_2 = 0000_{16} t/m $07FF_{16}$ voorkomt.

Het I/O-gedeelte wordt geselecteerd als $\overline{CE1}$ = 0, CE2 = 1 en IO/$\overline{M}$ = 1.

Vraag 9: Door de instructie OUT 00H wordt poort A/poort B/geen poort geselecteerd.

Laten we b.v. de instructie OUT 00H uitvoeren, dan wordt tijdens de instruction execution het adres 0000_{16} op de adresbus geplaatst (het poortnummer 00_{16} wordt immers zowel over de high als over de low order byte verstuurd). A15 t/m A11 zijn dan 0, dus $\overline{CE1}$ = 0. Tijdens een

Antw.8: 0000; 07FF. Antw.9: poort A.

output-instructie is IO/$\overline{M}$ = 1, dus het I/O-gedeelte binnen de 8355 wordt geselecteerd. Op de interne adresbus (zie fig.4) geldt dan A0 = 0 en A1 = 0, dus poort A wordt geselecteerd.

Vraag 10: Door de instructie OUT 04H wordt poort A/poort B/geen poort geselecteerd.

Tijdens de instruction execution van OUT 04H staat op de adresbus het adres 0404_{16} = 0000010000000100_2 en er geldt IO/$\overline{M}$ = 1. Het I/O-gedeelte wordt geselecteerd ($\overline{CE1}$ = 0, want A15 t/m A11 zijn alle 0). Omdat A0 = 0 en A1 = 0 wordt poort A geselecteerd.

Vraag 11: Door de instructie OUT 0CH wordt poort A/poort B/geen poort geselecteerd.

Tijdens de instruction execution van OUT 0CH staat op de adresbus $0C0C_{16}$ = 0000110000001100_2. Nu is A11 = 1, dus de CS0-uitgang van de 8205 is 1, dus $\overline{CE1}$ = 1. Er wordt dus geen enkele functie binnen de 8355 geselecteerd.

Voor poort B en de beide data direction registers vinden we op deze wijze ook 2 I/O-adressen (= poortnummers), waarbij elk wordt geselecteerd. Deze zijn in tabel 2 vermeld. Ga deze voor uzelf na.

Locatie	I/O-adressen
poort A	00_{16} en 04_{16}
poort B	01_{16} en 05_{16}
DDR A	02_{16} en 06_{16}
DDR B	03_{16} en 07_{16}

Tabel 2

Vraag 12: De adressen uit tabel 2 gelden voor memory/I/O mapped I/O.

De poortnummers uit tabel 2 gelden voor gebruik in IN- en OUT-instructies, dus I/O mapped I/O. M.b.v. de 8355 is memory mapped I/O ook mogelijk.

Vraag 13: We moeten dan gebruik maken van de besturingslijnen ..... en .....

Bij memory mapped I/O moeten we de $\overline{IOR}$-lijn als READ en de $\overline{IOW}$-lijn als WRITE toepassen.
Immers dan wordt het I/O-gedeelte geselecteerd, onafhankelijk van het IO/$\overline{M}$-signaal. In de SDK 85 is de $\overline{IOR}$ met +5V verbonden. Input-operaties kunnen dus alleen met IN-instructies worden gepleegd. De $\overline{IOW}$-ingang van de 8355 is met de $\overline{WR}$-uitgang van de 8085 verbonden. D.w.z. dat zowel bij OUTPUT- als bij memory write-operaties een write-impuls op de $\overline{IOW}$-ingang optreedt. Voor output-operaties is dus ook memory mapped I/O mogelijk.

Vraag 14: Door STA 000CH wordt poort A/poort B/DDR A/DDR B geselecteerd

Bij de uitvoering van b.v. STA 000CH zijn van de adresbus A15 t/m A11 alle 0, dus $\overline{CE1}$ = 0. Door de write-impuls op de $\overline{IOW}$-ingang wordt ongeacht de toestand van IO/$\overline{M}$ het I/O-gedeelte geselecteerd. In dit geval is poort A de bestemming, omdat A0 = 0 en A1 = 0. Memory mapped I/O is voor output-operaties in de SDK 85 mogelijk, mits men ervoor zorgt dat de eerste vijf bits van het adres alle 0 zijn.

Omdat de snelheden van de in de SDK 85 gebruikte 8085 en 8355 op elkaar zijn aangepast, is de READY-uitgang van de 8355 niet met de CPU verbonden. De READY-ingang van de 8085 is via een weerstand op de +5V aangesloten, zodat de CPU continu een actief READY-signaal ontvangt en geen wait states gaat tussenvoegen.

SAMENVATTING 4

14. In de SDK 85 beslaat het ROM-gedeelte van de 8355 de adressen 0000_{16} t/m $07FF_{16}$.

15. In het I/O-gedeelte van de 8355 in de SDK 85 heeft elke poort en elk data direction register 2 adressen, waarbij deze wordt geselecteerd.

16. In de SDK 85 moeten input-poorten altijd volgens I/O mapped I/O worden geadresseerd.

17. In de SDK 85 mogen output-poorten zowel volgens I/O mapped I/O als memory mapped I/O worden geadresseerd.

Antw.14: poort A.

8. DE RAM-I/O-TIMER CHIP 8155

Evenals de 8355 is ook de RAM-I/O-timer chip 8155 ontwikkeld voor toepassing in 8085-systemen. De 8155 is nl. volledig aangepast aan de gemultiplexte data/adresbus. Verder werkt de 8155 met dezelfde status- en besturingssignalen als de 8085.

Vraag 15: In de 8155 bevinden zich ..... bytes RAM, ... I/O-poorten met samen ..... I/O-lijnen en een ....-bits timer.

In de 8155 bevindt zich een RAM-gedeelte met een capaciteit van 256 bytes, 2 I/O-poorten met elk 8 I/O-lijnen, een I/O-poort met 6 I/O-lijnen en een 14-bits timer.
Al deze mogelijkheden en het feit, dat vrijwel geen extra logica nodig is om de 8155 op een 8085-bussysteem aan te sluiten, maken de 8155 bijzonder geschikt voor toepassing in kleine systemen.

In fig.7 zijn de aansluitmogelijkheden van deze chip weergegeven.

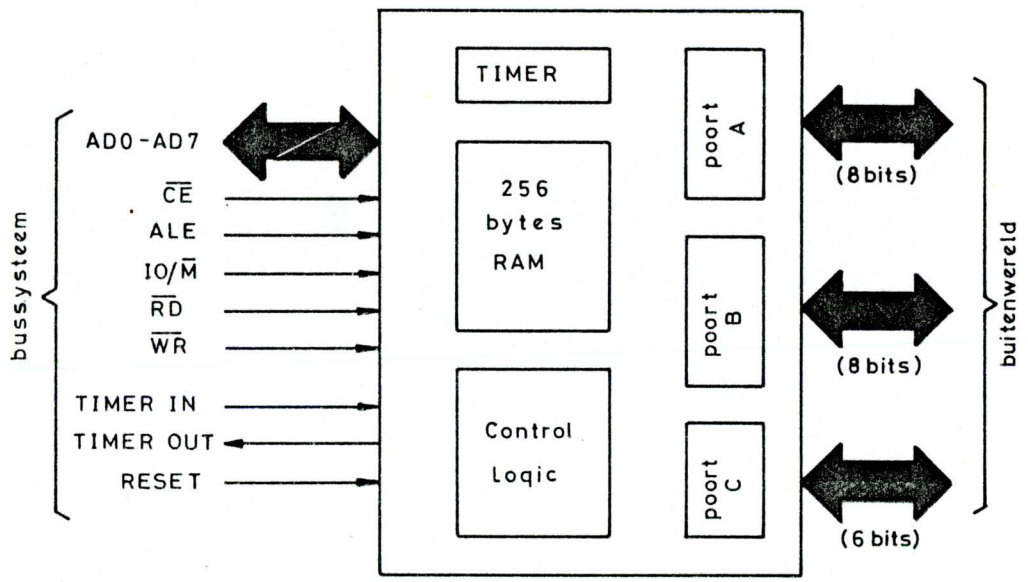


fig.7

We bespreken nu achtereenvolgens
a. Control logic (paragraaf 9)
b. RAM-gedeelte (paragraaf 10)
c. I/O-gedeelte (paragraaf 11)
d. Timer-gedeelte (paragraaf 12).

Daarna bespreken we op welke wijze de basic 8155 (paragraaf 13) en de expansion 8155 (paragraaf 14) in de SDK 85 zijn opgenomen.

9. CONTROL LOGIC (8155)

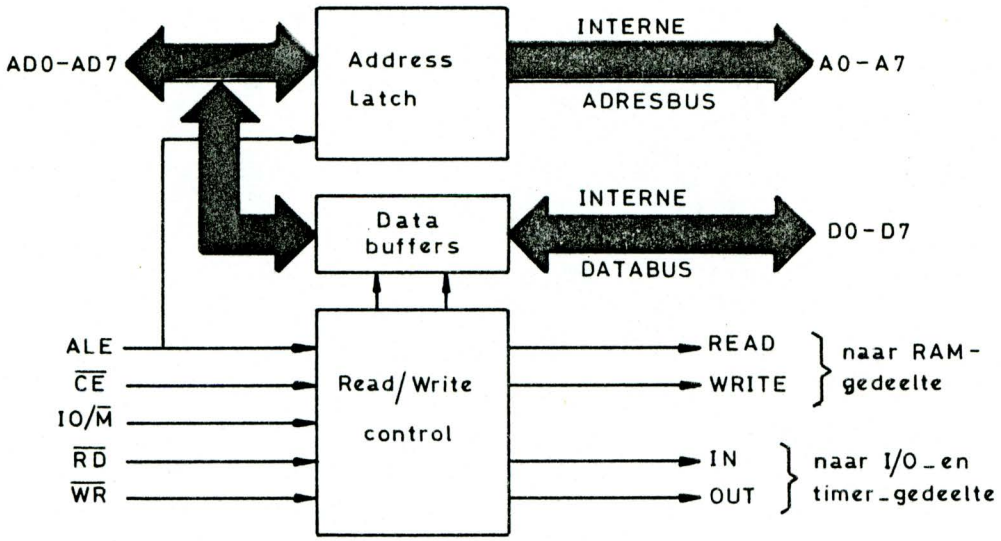


fig.8

In fig.8 is een gedetailleerder blokschema van de control logic weergegeven.

De address latch en de databuffers functioneren op dezelfde wijze als in de 8355. De interne adresbus is nu echter 8 bits breed.

Merk op, dat de 8155 geen READY-signaal afgeeft. Gewoonlijk is dit geen bezwaar, omdat de snelheid van de 8185 op die van de 8085 (waarvoor de chip is ontworpen) is aangepast.

De read/write control geeft de besturingssignalen voor de overige delen van de 8155 af. Hierbij spelen vooral de signalen op $\overline{CE}$, IO/$\overline{M}$, $\overline{RD}$ en $\overline{WR}$ een rol (zie tabel 3).

$\overline{CE}$	IO/$\overline{M}$	$\overline{RD}$ of $\overline{WR}$	afgegeven
1	X	X	-
0	0	$\overline{RD}$	READ
0	0	$\overline{WR}$	WRITE
0	1	$\overline{RD}$	IN
0	1	$\overline{WR}$	OUT
X = don't care. - = geen actief signaal aanwezig.			

Tabel 3

De 8155 wordt alleen geactiveerd als geldt $\overline{CE}$ = 0. Omdat er geen aparte ingangen zijn voor input- en output-signalen (zoals $\overline{IOR}$ en $\overline{IOW}$ bij de 8355) kunnen het I/O-gedeelte en de timer alleen volgens I/O mapped I/O worden geadresseerd.

10. RAM-GEDEELTE (8155)

Vraag 16: Het RAM-gedeelte heeft een capaciteit van ..... bytes.
Hiervoor zijn ..... adreslijnen noodzakelijk.

Het RAM-gedeelte van de 8155 bevat 256 geheugenwoorden van 8 bits. Om deze 256 locaties te kunnen adresseren zijn 8 adreslijnen nodig ($2^8 = 256$). Dit zijn de lijnen A0 t/m A7 van de interne databus.

Omdat er geen READY-signaal wordt afgegeven, moet de access-tijd van de 8155 aangepast zijn aan de snelheid van de CPU. Als de access-tijd voor een bepaalde CPU te lang duurt, kunnen we één van de volgende oplossingen kiezen:

a. De klokfrequentie van de CPU verlagen. Dit houdt wel in, dat elke state langer gaat duren, zodat de gehele programma-uitvoering wordt vertraagd.
b. D.m.v. een extra schakeling, die steeds wanneer de 8155 wordt geselecteerd, na een bepaalde tijd (minimaal de access-tijd) een READY-signaal genereert.

SAMENVATTING 5

18. De RAM-I/O-timer chip 8155 bevat
    a. 256 bytes RAM.
    b. 2 8-bits I/O-poorten.
    c. 1 6-bits I/O-poort.
    d. 1 14-bits timer.
    e. control logic.

19. De control logic bestaat uit
    a. een demultiplexer latch voor de data/adresbus.
    b. bi-directionele tri-state buffers t.b.v. de databus.
    c. read/write logic voor het opwekken van de interne besturingssignalen.

20. De 8155 geeft geen READY-signaal af.

11. I/O-GEDEELTE (8155)

Binnen het I/O-gedeelte van de 8155 komen 4 locaties voor, die door de gebruiker kunnen worden geadresseerd.
In fig.9 zijn de adresdecoder en I/O-control weergegeven. Deze twee schakelingen zorgen voor de juiste besturingssignalen voor de 4 locaties (tabel 4).

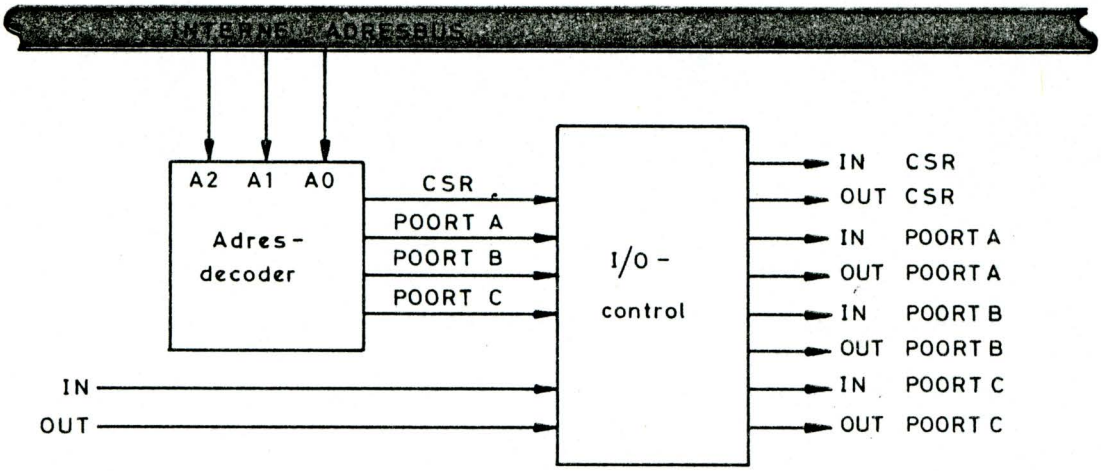


fig.9

A2	A1	A0	IN of OUT	afgegeven
0	0	0	IN	IN CSR
0	0	0	OUT	OUT CSR
0	0	1	IN	IN POORT A
0	0	1	OUT	OUT POORT A
0	1	0	IN	IN POORT B
0	1	0	OUT	OUT POORT B
0	1	1	IN	IN POORT C
0	1	1	OUT	OUT POORT C
1	X	X	X	-
X	X	X	-	-

X = don't care.
\- = geen actief signaal aanwezig.

Tabel 4

Vraag 17: Er wordt alleen een besturingssignaal afgegeven als A2 = 0/1.

Uit tabel 4 blijkt, dat er alleen een besturingssignaal wordt afgegeven als A2 = 0. Wanneer er zich nl. I/O-adressen met A2 = 1 op de databus bevinden, dan wordt een locatie binnen het timer-gedeelte geselecteerd.

In fig.10 is het blokschema van het I/O-gedeelte weergegeven.
T.b.v. de overzichtelijkheid zijn de besturingslijnen tussen I/O-control, command status register en de 3 I/O-poorten niet alle getekend.

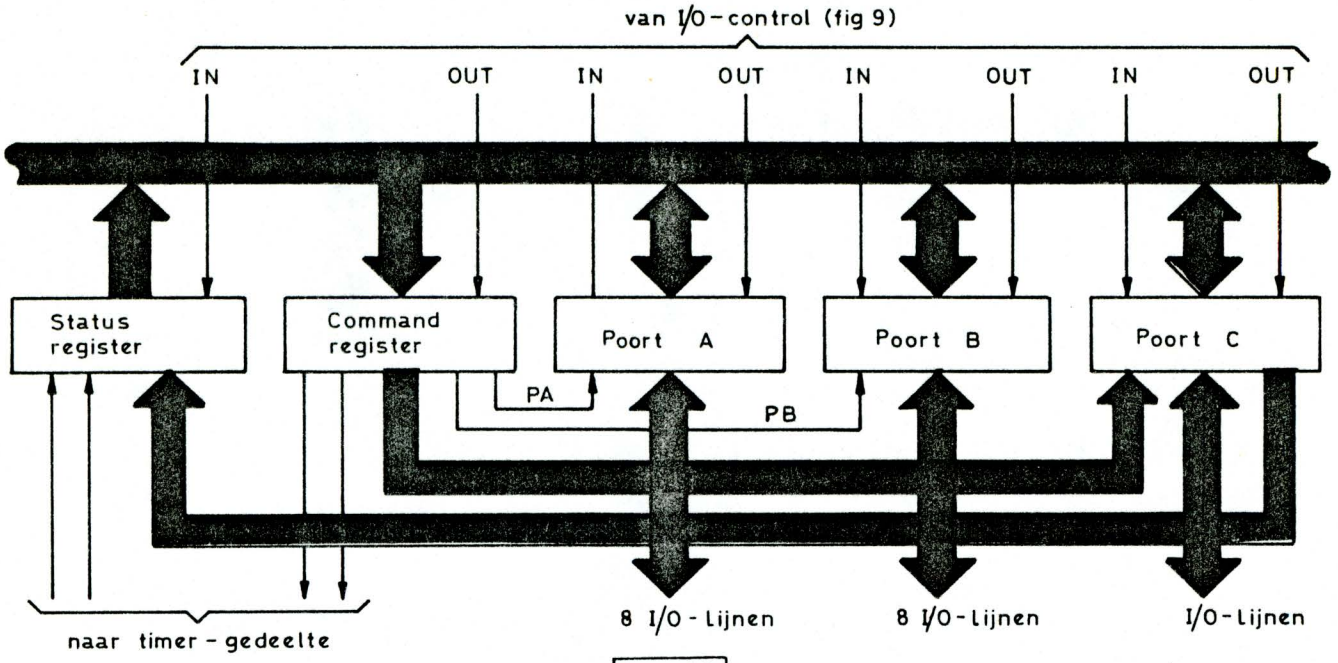


fig.10

We zullen nu achtereenvolgens behandelen
a. Command status register.
b. Strobed I/O.
c. Poorten A en B.
d. Poort C.

a. Command status register

Vraag 18: Om het command status register te selecteren, moet gelden CE = 0/1; IO/$\overline{M}$ = 0/1; A2 = 0/1; A1 = 0/1 en A0 = 0/1.

Het command status register wordt m.b.v. input- en output-instructies aangesproken. Als we het juiste I/O-adres op de adresbus plaatsen, geldt $\overline{CE}$ = 0 (de 8155 wordt dus geactiveerd) en IO/$\overline{M}$ = 1 (het I/O-gedeelte binnen de 8155 wordt geselecteerd). Het juiste I/O-adres is XXXXX000 (zie tabel 4). Hierin zijn A7 t/m A3 don't cares (zie fig.9).

Het command status register is in twee delen te splitsen, nl. het command register en het status register.

Als we het CSR d.m.v. een output-instructie aanspreken, dan wordt het command register (fig.11) geselecteerd. Dit command register is alleen

 Antw.18: 0; 1; 0; 0; 0.

met een bepaalde bitcombinatie te vullen (is dus niet uit te lezen) en dient voor het programmeren van het I/O-gedeelte en de timer.

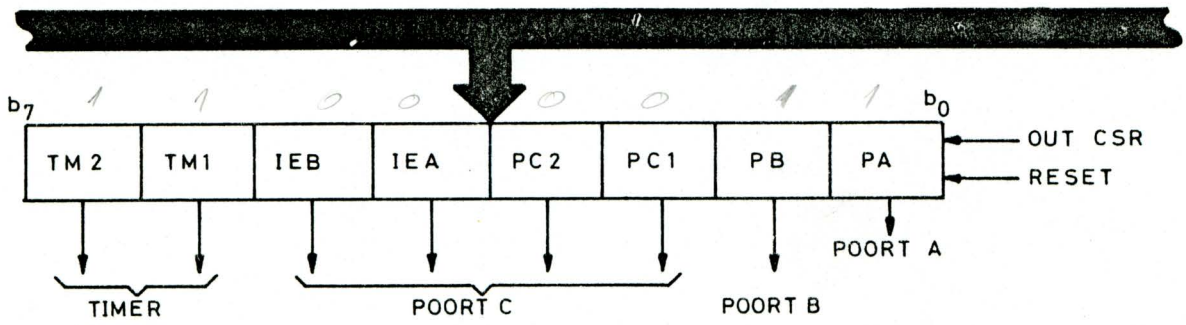


fig.11

Met b_0 en b_1 kunnen we de I/O-poorten A en B als volgt initialiseren.
PA = 0: poort A is input.
PA = 1: poort A is output.
PB = 0: poort B is input.
PB = 1: poort B is output.

Met b_2 t/m b_5 programmeren we poort C. Hierop gaan we in paragraaf 11d dieper in.

Met b_6 en b_7 laten we de timer starten en stoppen. Dit wordt in paragraaf 12 behandeld.

Stel, we willen poort A als input-poort en poort B als output-poort initialiseren.
b_2 t/m b_7 van het command register moeten 0 zijn.

Vraag 19: Het command register moet dan worden gevuld met $\ldots\ldots_{16}$.

Er moet dan gelden PA = 0 en PB = 1. Het command register moet dan worden gevuld met $00000010_2 = 02_{16}$.

Door een impuls op de RESET-ingang van de 8155 wordt het command register gevuld met 01000000_2.

Vraag 20: Na een RESET-impuls geldt poort A is input/output en poort B is input/output.

Na een RESET-impuls zijn PA en PB beide 0, de poorten A en B zijn dan beide als input-poort geïnitialiseerd.

Als we het CSR d.m.v. een input-instructie aanspreken, dan wordt het status register (fig.12) geselecteerd. Dit status register kan alleen worden uitgelezen en dient om de status van het I/O-gedeelte en de timer naar de CPU over te brengen.

Antw.19: 02. Antw.20: input; input.

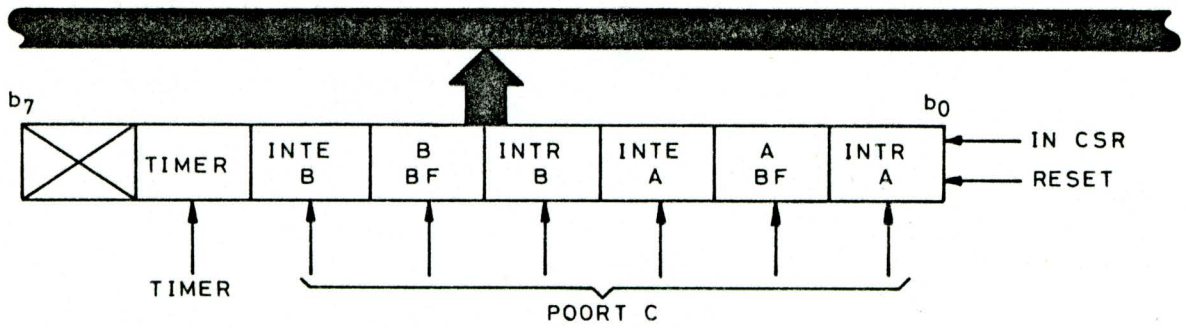


fig.12

b_0 t/m b_5 bevatten steeds informatie over de status van de I/O-poorten. De betekenis van deze bits wordt in paragraaf 11d besproken.

b_6 wordt 1 op het moment dat de timer in de 8155 tot nul is teruggeteld. b_6 wordt 0 (gereset) als het status register wordt uitgelezen of door een impuls op de RESET-ingang van de 8155.

Het uitlezen van het status register wordt voornamelijk toegepast bij geprogrammeerde I/O. De CPU behoeft nu niet een aantal afzonderlijke statuslijnen te testen, maar kan d.m.v. één enkele input-instructie de status van het gehele I/O-gedeelte naar binnen halen.

SAMENVATTING 6

21. In het I/O-gedeelte van de 8155 bevinden zich:
    adresdecoder,
    I/O-control,
    2 8-bits I/O-poorten
    1 6-bits I/O-poort
    command status register.

22. De adresdecoder en de I/O-control zorgen voor de juiste IN- en OUT-commando's voor de 3 I/O-poorten en het command status register.

23. Het command status register is te splitsen in command register en status register.

24. Het command register dient om het I/O-gedeelte en de timer in de 8155 te programmeren. Het command register kan alleen met een 8-bits combinatie worden gevuld.

25. Het status register dient om de status van het I/O-gedeelte en de timer uit te lezen t.b.v. geprogrammeerde I/O. Het status register kan alleen worden uitgelezen.

b. Strobed I/O

Voordat de poorten A, B en C kunnen worden behandeld, moet u iets weten over strobed I/O.

De gang van zaken bij strobed input is in fig.13 weergegeven.

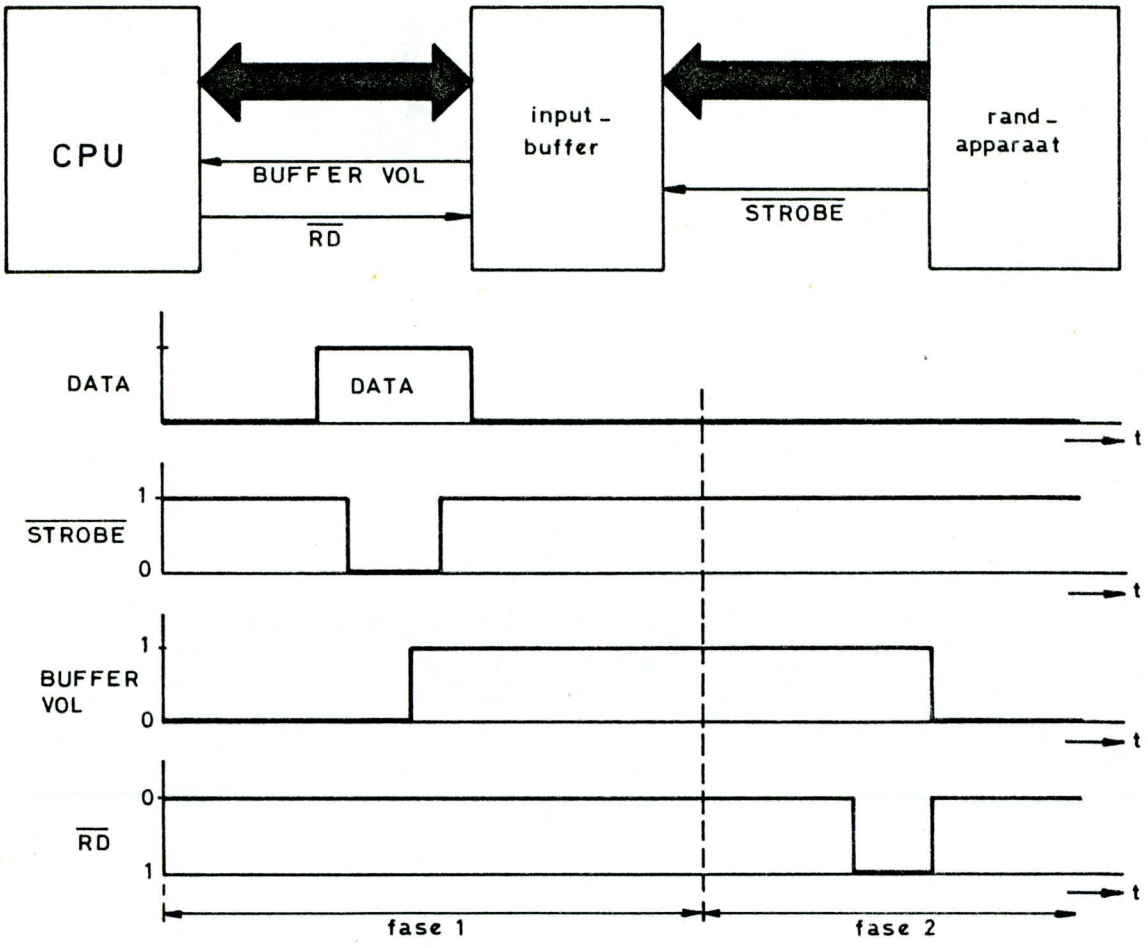


fig.13

Er zijn nl. randapparaten, die kortstondig data afgeven. Op het moment dat de data op de lijnen naar de I/O-module aanwezig is, wordt tevens een z.g. STROBE-signaal afgegeven. Nu zijn er voor het invoeren van de data twee oplossingen.

1. Geprogrammeerde I/O. De CPU moet, m.b.v. input-instructies, het STROBE-signaal continu testen. Is dit signaal actief (in fig.13 geldt 0 = actief), dan kan de data, weer met een input-instructie, worden ingevoerd.

Het testen van het STROBE-signaal kost echter veel tijd, die beter besteed kan worden aan de uitvoering van het eigenlijke programma in de microcomputer. Een nog groter probleem is, dat data en STROBE-signaal in de meeste gevallen veel te kort aanwezig zijn, om op bovengenoemde manier te kunnen worden verwerkt.
Als de CPU een actief STROBE-signaal heeft gedetecteerd, dan is op het moment, dat de data ingevoerd kan worden (enkele µs later) de data al vaak weer verdwenen. Daarom wordt meestal voor de volgende oplossing gekozen.

2. Strobed I/O. In de I/O-module is dan een input-buffer opgenomen (fig.13). Deze input-buffer bestaat uit 8 flip flops (tenminste bij 8-bits microcomputers), die op commando van het STROBE-signaal de data van het randapparaat opslaan. De bijbehorende besturingslogica geeft dan een actief BUFFER VOL-signaal aan de CPU af. Dit signaal blijft actief, totdat de CPU d.m.v. een leesopdracht (b.v. een input-instructie) de inhoud van de input-buffer heeft uitgelezen. Daarna is de input-buffer weer in staat nieuwe data van het randapparaat op te slaan, totdat de CPU deze weer kan overnemen.
De gehele actie is in twee fasen te verdelen, nl. fase 1, de hardware-actie, waarbij de data door het STROBE-signaal in de input-buffer wordt geklokt en fase 2, de software-actie, waarbij de CPU de inhoud van de input-buffer overneemt.

Strobed output via de 8155 verloopt volgens fig.14.

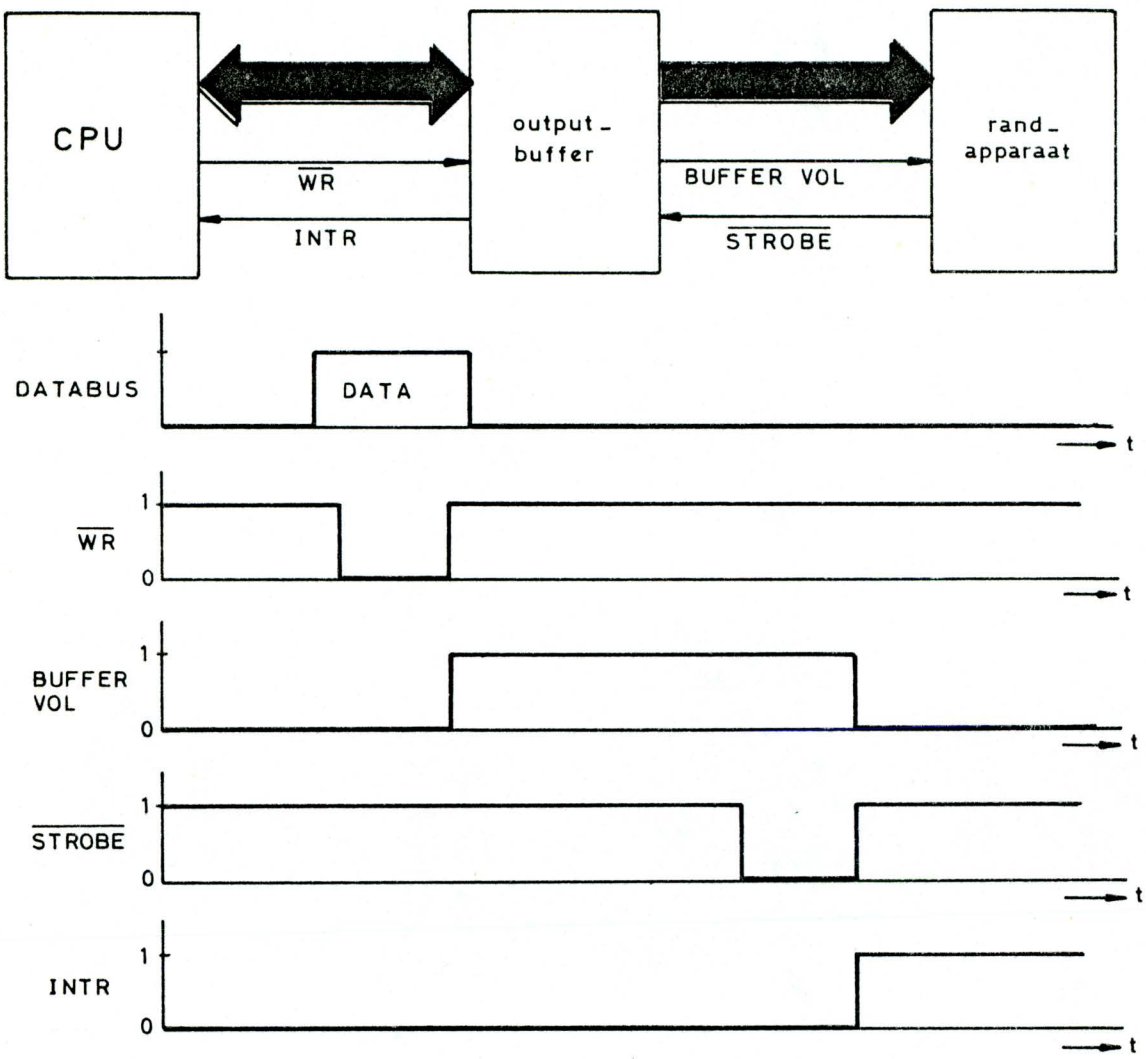


fig.14

De CPU plaatst d.m.v. een output-instructie een uit te voeren karakter in de output-buffer. Deze data staat dan continu op de datalijnen naar het randapparaat.
De bij de output-buffer behorende besturingslogica zendt dan een actief BUFFER VOL-signaal naar het randapparaat. Als deze de data heeft overgenomen, wordt een STROBE-signaal teruggestuurd.
De besturingslogica zal dan
1. het actieve BUFFER VOL-signaal wegnemen.
2. een interrupt request naar de CPU zenden.
Wat er daarna gebeurt, is afhankelijk van de instructies in de bijbehorende interrupt service routine. Er kan b.v. direct een nieuw karakter naar de output-buffer worden gezonden.

SAMENVATTING 7

26. Bij strobed I/O wordt m.b.v. een STROBE-signaal de transmissiesnelheid van de CPU aangepast aan die van een randapparaat, zonder dat er data verloren gaat (van een snel input-apparaat) of de CPU moet blijven wachten (op een traag output-apparaat) totdat er data kan worden uitgevoerd.

27. Bij strobed input klokt een input-apparaat d.m.v. het STROBE-signaal data in de input-buffer. Dan wordt een BUFFER VOL-signaal naar de CPU gezonden.

28. Bij strobed output plaatst de CPU data in de output-buffer. Er wordt dan een BUFFER VOL-signaal naar het output-apparaat gezonden. Als dit de data heeft overgenomen, wordt er een STROBE-signaal teruggestuurd, waardoor er een interrupt request naar de CPU wordt gezonden.

c. Poorten A en B

Fig.15 toont de schakeling voor b_5 van poort A. Deze schakeling komt in de 8155 dus 16 maal voor.

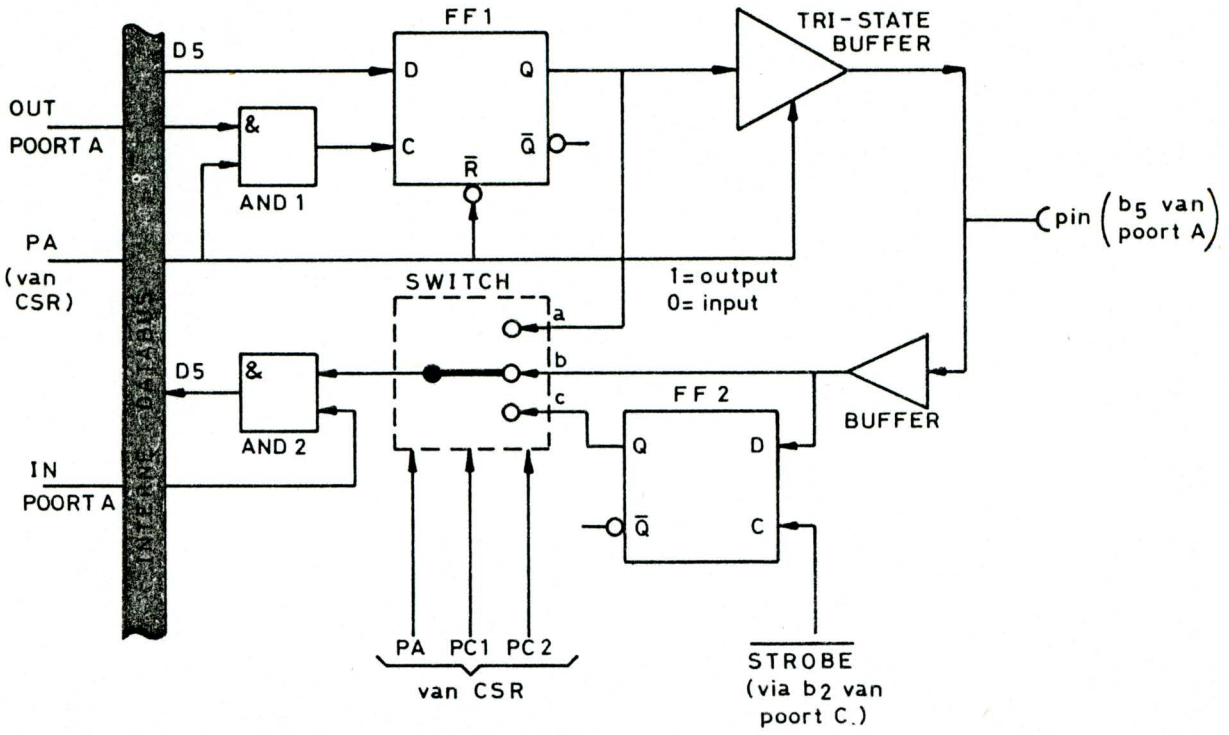


fig.15

In dit schema is FF1 b_5 van de output-buffer en FF2 b_5 van de input-buffer.
Poort A kan als input-poort en als output-poort werken, afhankelijk van PA (dit is b_0 van het command register, zie fig.11).
In beide gevallen kan dan m.b.v. PC1 en PC2 (b_2 en b_3 van het command register) wel of niet voor strobed I/O worden gekozen.
In totaal zijn er dus 4 modes, d.w.z. manieren, waarop poort A kan functioneren.

Input-poort (PA = 0; PC1 = PC2). Omdat geldt PA = 0, spelen FF1, AND1 en de tri-state buffer geen rol. Door de drie besturingssignalen wordt SWITCH (dit is te beschouwen als een elektronische schakelaar met 2 standen) in stand b gezet. FF2 doet dan ook niet mee. Voor dit geval kunnen we fig.15 dan vereenvoudigen tot fig.16a.

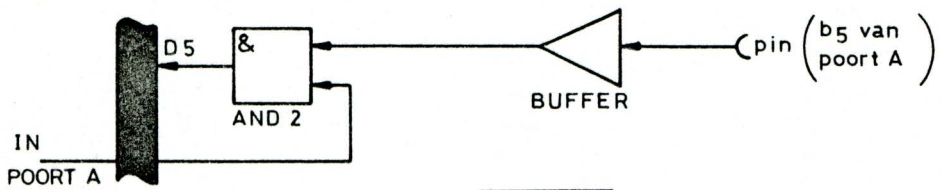


fig.16a

Door een besturingssignaal IN POORT A wordt de op de pin aangeboden informatie via de buffer en EN-poort AND2 op D5 van de interne databus geplaatst. Aangezien in poort A acht van deze schakelingen zitten, hebben we zo een normale 8-bits input-poort verkregen.

Opmerking:

PC1 = PC2 wil zeggen, dat b_2 en b_3 van het command register gelijk zijn. Dus beide 0, of beide 1. Als b_2 en b_3 niet gelijk zijn (PC1 ≠ PC2), dan wordt de poort voor strobed I/O geprogrammeerd.

Output-poort (PA = 1, PC1 = PC2). Omdat geldt PA = 1 geeft AND1 het commando OUT POORT A aan de klokingang van FF1. De reset-ingang is niet actief en de tri-state buffer is continu in de geleidende toestand. SWITCH wordt door PA, PC1 en PC2 in stand a geplaatst.
FF2 speelt dus geen rol. Voor dit geval kunnen we fig.15 dan vereenvoudigen tot fig.16b.

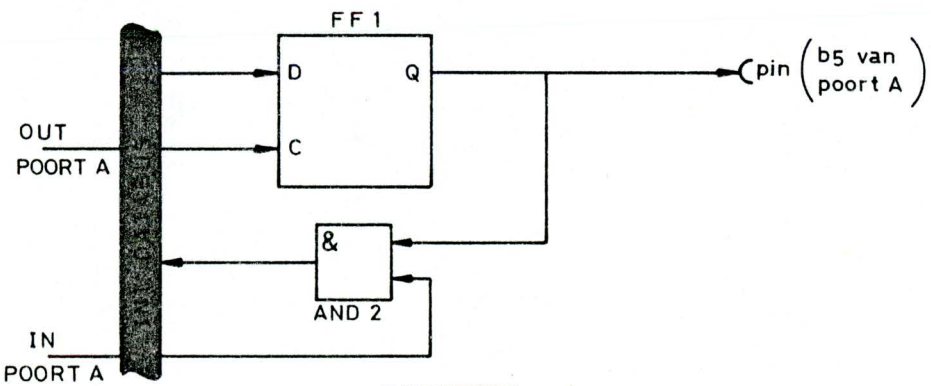


fig.16b

Door een besturingssignaal OUT POORT A wordt D5 van de interne databus in FF1 geklokt en op de Q-uitgang geplaatst. Evenzo voor de overige 7 bits van poort A. Dit is dus een normale output-poort.

Vraag 21: In dit geval kan de uitgevoerde data wel/niet door de CPU worden teruggelezen.

23-09 Antw.21: wel.

Als poort A nu d.m.v. een input-instructie wordt geselecteerd, wordt door het besturingssignaal IN POORT A de uitgevoerde data via de 8 EN-poorten AND2 op de interne databus geplaatst. De CPU kan dus de via een output-poort uitgevoerde data steeds teruglezen.

Strobed input-poort (PA = 0, PC1 ≠ PC2). Omdat geldt PA = 0, spelen FF1, AND1 en de tri-state buffer geen rol. Door PA, PC1 en PC2 wordt SWITCH in stand c geplaatst. Fig.15 is voor dit geval te vereenvoudigen tot fig.16c.

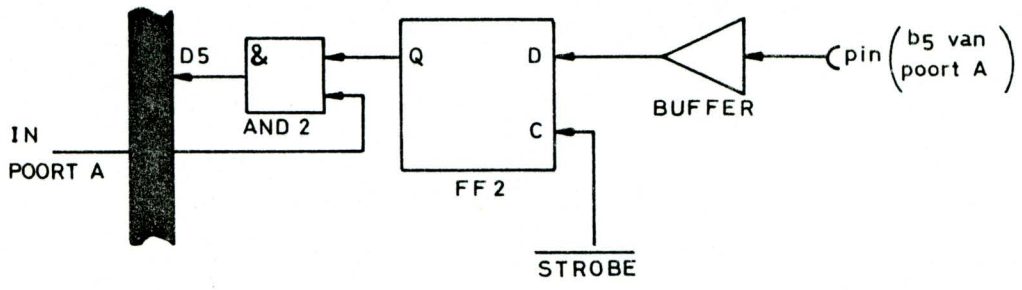


fig 16c

D.m.v. een actief $\overline{\text{STROBE}}$-signaal wordt de toestand van b_5 in FF2 geklokt. Door een commando IN POORT A wordt de inhoud van FF2 op de interne databus geplaatst. Dit gebeurt natuurlijk ook voor de overige 7 bits van poort A. Het bij strobe input behorende besturingssignaal BUFFER VOL wordt door poort C verzorgd.

Strobed output-poort (PA = 1, PC1 ≠ PC2). Door PA, PC1 en PC2 wordt SWITCH in stand a gezet. Fig.15 kan in dit geval weer worden vereenvoudigd tot fig.16b. D.w.z. dat poort A zich als een normale output-poort gedraagt. Wanneer we strobed output willen plegen, moeten de noodzakelijke besturingssignalen BUFFER VOL en INTR (zie fig.14) door de hardware van poort C worden opgewekt.

Alles wat in deze paragraaf over poort A is geschreven, geldt ook voor poort B.

SAMENVATTING 8

29. De poorten A en B kunnen elk voor 4 modes worden geprogrammeerd, nl. als
    normale input-poort,
    normale output-poort,
    strobed input-poort,
    strobed output-poort.

30. Met PA en PB (b_0 en b_1 van het command register) kan de datarichting, d.w.z. input of output, worden geprogrammeerd.

31. Met PC1 en PC2 (b_2 en b_3 van het command register) wordt de keuze tussen normale en strobed I/O gemaakt.

32. Als een poort voor strobed I/O is geprogrammeerd, dan doen 3 lijnen van poort C dienst als besturingslijnen voor de strobed I/O-poort.

33. Als een poort als output-poort is geprogrammeerd, dan kan de CPU steeds de uitgevoerde data m.b.v. input-instructies teruglezen.

d. Poort C

D.m.v. PC1 en PC2 (b_2 en b_3 van het command register, zie fig.11) kan poort C in vier verschillende modes worden geprogrammeerd. We zullen deze 4 modes nu achtereenvolgens kort bespreken.

Mode 1: (PC1 = 0, PC2 = 0). Poort C werkt als een normale 6-bits input-poort. Door een actief besturingssignaal IN POORT C (afkomstig van de I/O-control, zie fig.9), wordt de op de 6 input-lijnen aangeboden data op D0 t/m D5 van de interne databus geplaatst (fig.17a). D6 en D7 zijn dan ongedefiniëerd.

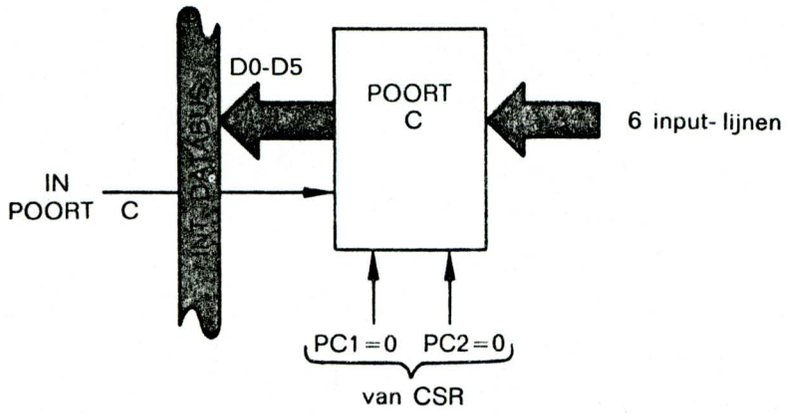


fig. 17a

Mode 2: (PC1 = 1, PC2 = 1). Poort C werkt als een normale 6-bits output-poort. Door een commando OUT POORT C wordt de data van D0 t/m D5 van de interne databus op de 6 output-lijnen geplaatst (fig.17b). De data van D6 en D7 van de databus worden dus niet uitgevoerd.

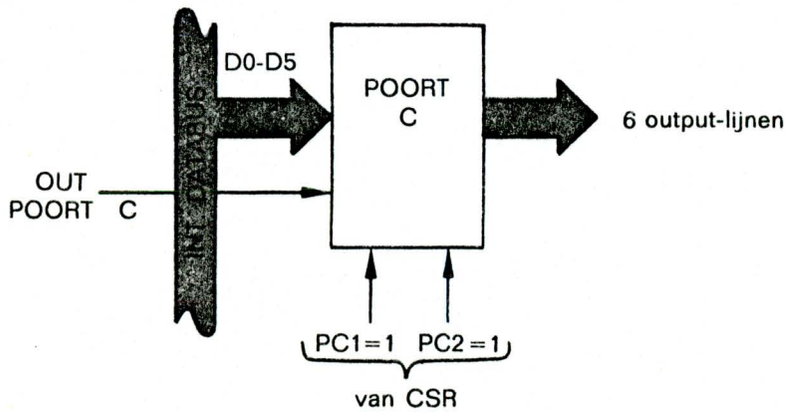


fig. 17 b

Mode 3: (PC1 = 1, PC2 = 0). Poort C werkt nu gedeeltelijk als besturings-poort voor poort A en gedeeltelijk als output-poort (zie tabel 5 en fig.17c).

bits van poort C	MODE 3
b_0	A INTR (= interrupt request t.b.v. poort A)
b_1	A BF (= BUFFER VOL t.b.v. poort A)
b_2	A $\overline{\text{STB}}$ (= $\overline{\text{STROBE}}$ t.b.v. poort A)
b_3	output-lijn
b_4	output-lijn
b_5	output-lijn

Tabel 5

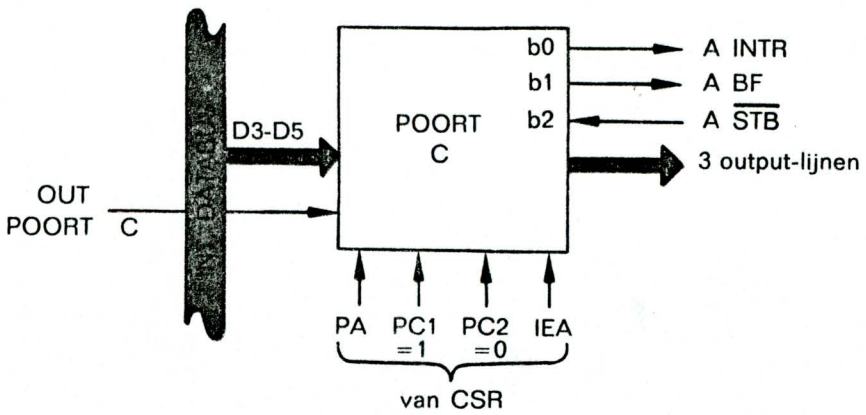


fig. 17c

b_3, b_4 en b_5 van poort C zijn normale output-lijnen. Door een commando OUT POORT C (afkomstig van de I/O-control), wordt alleen de data van D3, D4 en D5 op de drie output-lijnen geplaatst. De overige bits van de databus kunnen in deze mode niet worden uitgevoerd via poort C (natuurlijk wel via poort A of poort B).

Poort A is in deze mode voor strobed input (als PA in het command register 0 is) of voor strobed output (als PA = 1) geprogrammeerd. De drie besturingssignalen voor de strobed I/O via poort A, dus b_0, b_1 en b_2 van poort C moeten hardware-matig met de CPU en het randapparaat worden verbonden. In deze mode werkt poort B dus als normale input-poort (als PB in het command register 0 is) of als normale output-poort (als PB = 1).

Opmerking:

Door het vullen van de interrupt enable bit van IEA (b_4 van het command register) kunnen we aangeven of poort C wel (IEA = 1) of niet (IEA = 0) een interrupt request INTR A aan de CPU moet zenden. Dit is van belang als de programma-uitvoering in de microcomputer gedurende bepaalde tijd niet door interrupt I/O of strobed I/O mag worden onderbroken.

Mode 4: (PC1 = 0, PC2 = 1). Poort C werkt nu geheel als besturingspoort voor strobed I/O via de poorten A en B (zie tabel 6 en fig.17d).

bits van poort C	MODE 4
b_0	A INTR (= interrupt request t.b.v. poort A)
b_1	A BF (= BUFFER VOL t.b.v. poort A)
b_2	A $\overline{\text{STB}}$ (= $\overline{\text{STROBE}}$ t.b.v. poort A)
b_3	B INTR (= interrupt request t.b.v. poort B
b_4	B BF (= BUFFER VOL t.b.v. poort B)
b_5	B $\overline{\text{STB}}$ (= $\overline{\text{STROBE}}$ t.b.v. poort B).

Tabel 6

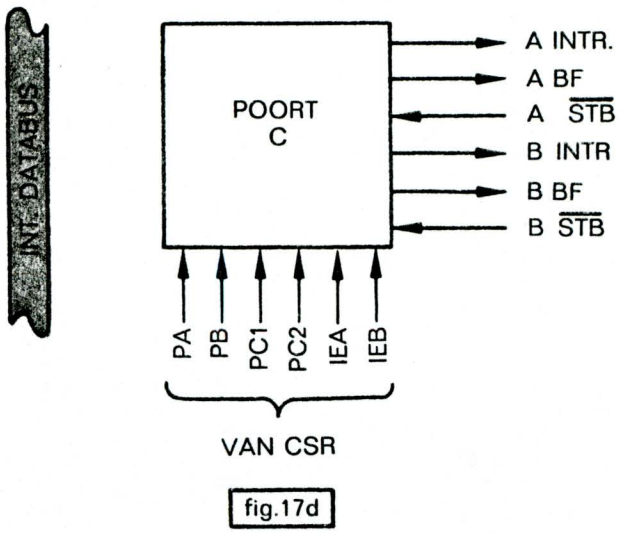


fig.17d

b_0, b_1 en b_2 zijn evenals in mode 3, de besturingssignalen voor strobed I/O via poort A. Nu werken b_3, b_4 en b_5 als de besturingssignalen voor strobed I/O via poort B. In mode 4 werken poort A en poort B dus beide met strobed I/O. Voor elke poort is met PA resp. PB in het command register te programmeren of het strobed input of strobed output betreft. D.m.v. IEA en IEB (interrupt enable bits, b_4 en b_5 van het command register) is het afgeven van interrupt requests A INTR resp. B INTR toe te staan of te verbieden.

Vraag 21: Het is wel/niet mogelijk om poort A als normale input-poort en tegelijk poort B als strobed input-poort te programmeren.

In mode 3 werkt alleen poort A als strobed I/O-poort. In mode 4 werken de poorten A en B beide met strobed I/O. Het is dus niet mogelijk om alleen poort B als strobed I/O-poort en tegelijkertijd poort A als normale I/O-poort te laten functioneren. Als er slechts één strobed I/O-poort vereist is, dan moet dit dus poort A zijn.

M.b.v. de lijnen A INTR en B INTR van poort C is het natuurlijk ook mogelijk om interrupt I/O te plegen.

Antw.21: niet.

De conclusies, die we uit deze en voorgaande paragrafen kunnen trekken, zijn:

1. D.m.v. de 8155 kunnen we 22 (programmeerbare) I/O-lijnen voor normale I/O realiseren.
2. De poorten A en B kunnen ook worden gebruikt voor strobed I/O en interrupt I/O. We moeten dan de 6 I/O-lijnen van poort C opofferen t.b.v. de noodzakelijke besturingssignalen.

e. Voorbeeld

We hebben de mogelijkheden van het I/O-gedeelte van de 8155 in afzonderlijke delen besproken. We zullen nu enkele mogelijkheden samenvoegen in een voorbeeld (fig.18). Hierin zijn alleen de databus en de belangrijkste besturingssignalen weergegeven.

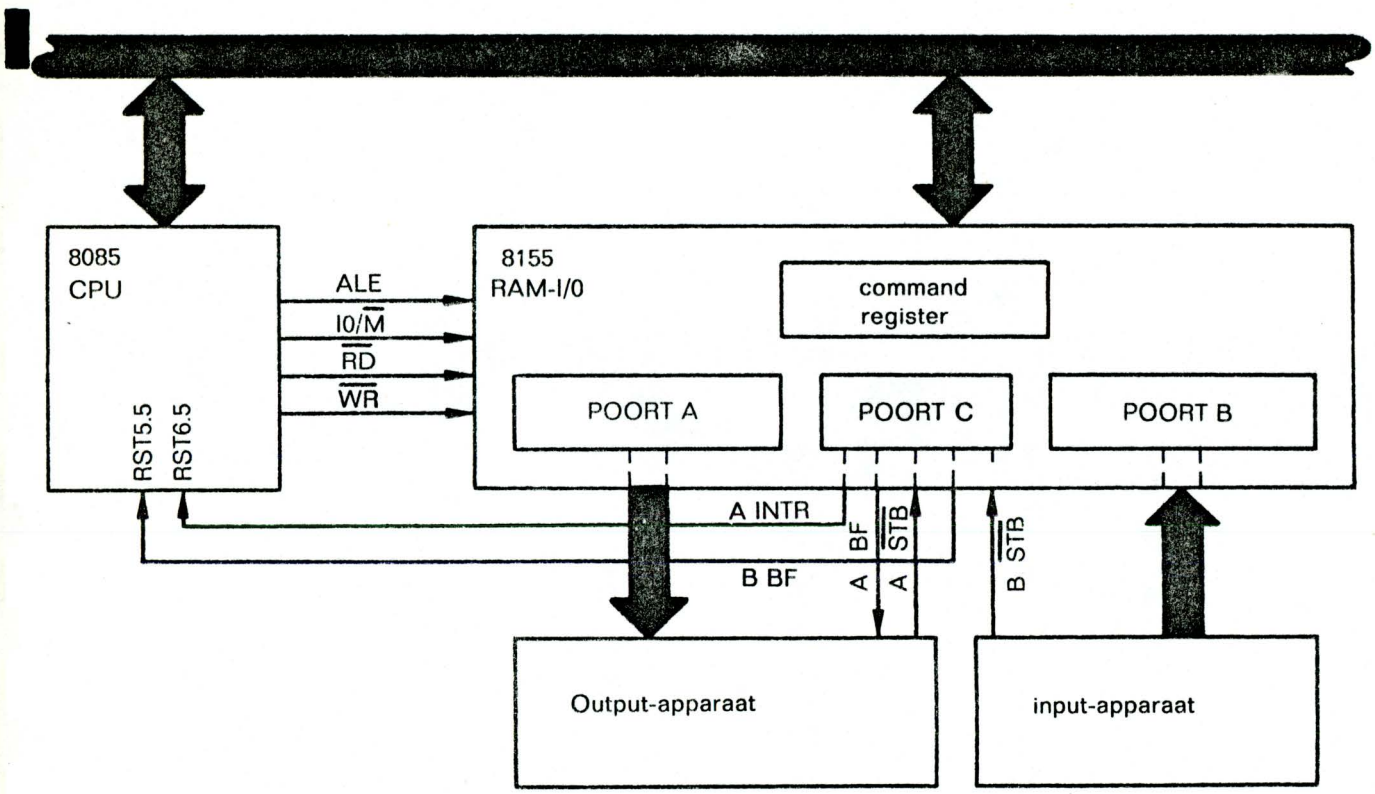


fig. 18

Op de 8155 zijn een input- en een output-apparaat aangesloten, die beide d.m.v. strobed I/O met de rest van het microcomputersysteem communiceren.

Vraag 22: De strobed input vindt plaats op basis van geprogrammeerde/interrupt I/O. De strobed output vindt plaats op basis van geprogrammeerde/interrupt I/O.

De lijnen A INTR en B BF zijn verbonden met interrupt request-ingangen van de CPU. Zowel de strobed output via poort A als de strobed input via poort B vindt plaats op basis van interrupt I/O. In beide gevallen zijn de voordelen van strobed I/O en interrupt I/O dus gecombineerd.

Vraag 23: IEA in het command register moet 0/1 zijn.

Omdat poort C interrupt requests op de lijn A INTR moet plaatsen, moet IEA (b_4 van het command register) 1 zijn. Voor IEB geldt deze voorwaarde niet. De lijn B INTR wordt nl. niet gebruikt, omdat het BUFFER VOL-signaal (B BF) in dit geval als interrupt request dienst doet.

In dit voorbeeld zien we duidelijk het verschijnsel programmeerbare chip naar voren komen. Immers, aan het begin van het uit te voeren programma moet zowel de interne interrupt controller in de CPU als het I/O-gedeelte in de 8155 worden geprogrammeerd. Hiertoe moeten we dus het interrupt mask register en het command register met bepaalde bitcombinaties vullen.

We gaan nu bepalen met welke bitcombinaties we het systeem op de gewenste wijze kunnen laten functioneren. Hierbij gaan we ervan uit, dat de niet-gebruikte bits 0 zijn. Dit zijn b_7, b_6 en b_5 van het interrupt mask register en b_7, b_6 en b_5 van het command register.

Vraag 24: Het interrupt mask register moet worden gevuld met ……$_2$ = ……$_{16}$.

Voor het interrupt mask register moet gelden:
$b_0 = 0$, om RST5.5 te laten honoreren.
$b_1 = 0$, om RST6.5 te laten honoreren.
$b_2 = 1$, $b_4 = 1$ } om RST7.5 niet te laten honoreren.
$b_3 = 1$, om b_0, b_1 en b_2 te kunnen wijzigen.

Het interrupt mask register moet dan m.b.v. een SIM-instructie worden gevuld met $00011100_2 = 1C_{16}$.

Vraag 25: Het command register moet worden gevuld met ……$_2$ = ……$_{16}$.

Voor het command register moet dan gelden:
b_0 = PA = 1 (poort A = output)
b_1 = PB = 0 (poort B = input)
b_2 = PC1 = 0, b_3 = PC2 = 1 } mode 4
b_4 = IEA = 1 (enable A INTR)

Het command register moet dan m.b.v. een OUT-instructie worden gevuld met $00011001_2 = 19_{16}$.

 Antw.23: 1. Antw.24: 00011100; 1C. Antw.25: 00011001; 19.

SAMENVATTING 9

34. Poort C kan voor 4 modes worden geprogrammeerd. Hiertoe dienen PC1 en PC2 (b_2 en b_3 van het command register).

35. In mode 1 werkt poort C als normale 6-bits input-poort. De poorten A en B zijn dan ook normale I/O-poorten.

36. In mode 2 werkt poort C als normale 6-bits output-poort. De poorten A en B zijn dan ook normale I/O-poorten.

37. In mode 3 werkt poort C deels als normale 3-bits output-poort (op b_3, b_4 en b_5) en deels als besturingspoort (op b_0, b_1 en b_2) t.b.v. de strobed I/O-poort A. Poort B werkt dan als normale I/O-poort.

38. In mode 4 werkt poort C geheel als besturingspoort voor de strobed I/O-poorten A en B.

12. TIMER-GEDEELTE (8155)

In veel applicaties moet de tijd nauwkeurig worden bijgehouden. Enerzijds om iets op een bepaald tijdstip te laten gebeuren, anderzijds om te meten welke tijd er tussen twee gebeurtenissen is verlopen. Dit soort tijdmetingen is mogelijk door gebruik te maken van software wachtlussen. In veel gevallen is dit om twee redenen onpraktisch. Ten eerste omdat een werkelijk nauwkeurige tijdmeting niet mogelijk is, en ten tweede omdat de CPU in een dergelijke wachtlus zinloos werk uit voert. De tijd zou beter besteed kunnen worden aan de uitvoering van de instructies uit het feitelijke hoofdprogramma.

Vaak wordt daarom een tijdmeting uitgevoerd m.b.v. een hardware timer. Zo'n timer bestaat uit een digitale teller, met een stabiele (kristal) klok. Als de telfrequentie hoog is en de teller uit voldoende flip flops bestaat, kan én een nauwkeurige én tegelijk lange tijd worden ingesteld. Een dergelijke timer is in de 8155 opgenomen.

In fig.19 is het blokschema van het timer-gedeelte van de 8155 weergegeven.

De taak van de timer control is het juist laten functioneren van het gehele timer-gedeelte. De timer zelf is een 14-bits down counter, waarvan de inhoud door elke klokimpuls met 1 wordt verlaagd.
Als de inhoud van de timer 0 is geworden, wordt er een signaal aan de timer control afgegeven. Deze geeft dan een TIMER OUT-signaal af.
Welke vorm dit signaal heeft, hangt af van de mode, waarin de timer functioneert.
Het count length register bevat steeds de 14-bits waarde, waarmee de counter wordt gevuld bij het starten. b_{14} en b_{15} (M1 en M2) van het count length register dienen om de timer in een van de 4 mogelijke modes te programmeren.

We zullen nu achtereenvolgens de mogelijkheden en eigenschappen van het timer-gedeelte bespreken.

a. Adresselectie en besturingssignalen

De timer control zorgt er o.a. voor dat de juiste besturingssignalen voor de overige delen binnen het timer-gedeelte worden opgewekt.
Hiertoe worden A0, A1, A2 (van de interne adresbus), IN en OUT (afkomstig van de control logic van de 8155, zie fig.8) gebruikt. Dit is in tabel 7 weergegeven.

A2	A1	A0	IN of OUT	afgegeven
X	X	X	-	-
0	X	X	X	-
1	0	0	OUT	WR CLRLO
1	0	0	IN	RD TMLO
1	0	1	OUT	WR CLRHI
1	0	1	IN	RD TMHI
1	1	X	X	-

X = don't care.
- = geen actief signaal aanwezig.

Tabel 7

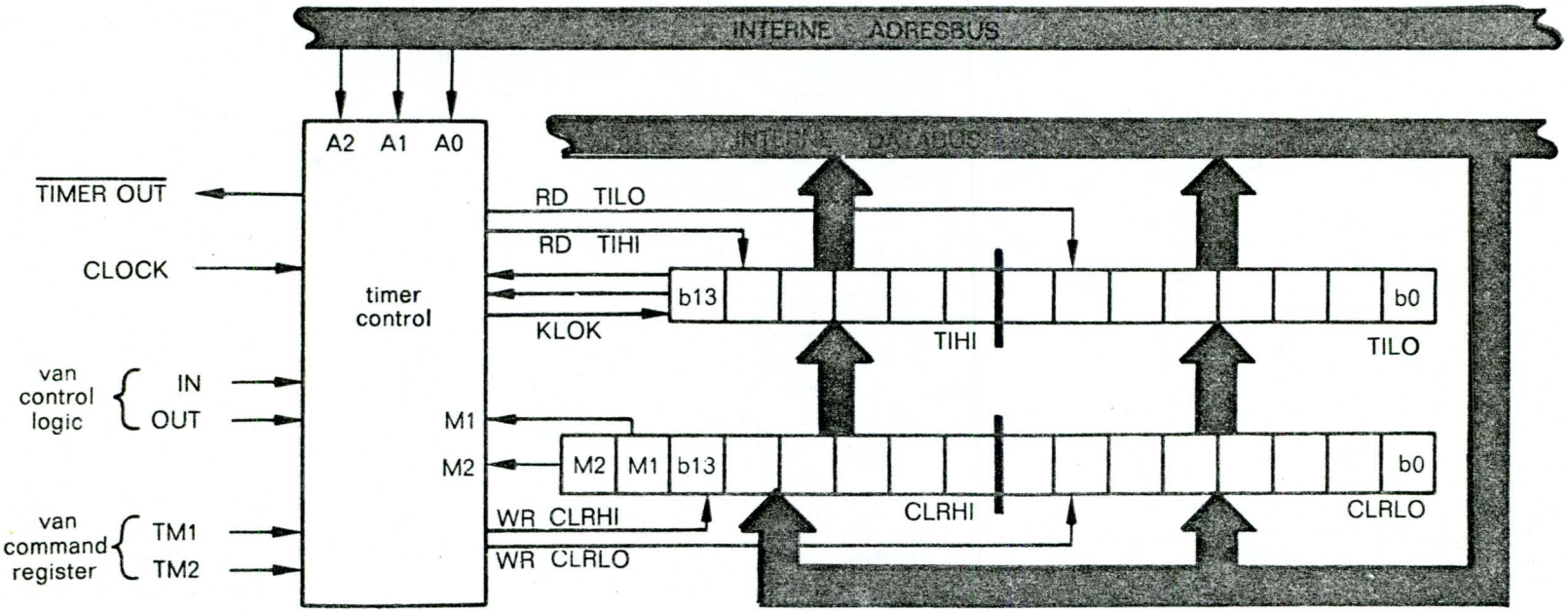


fig.19

In tabel 7 betekent
CLRLO: low order byte van count length register.
CLRHI: high order byte van count length register.
TILO : low order byte van timer.
TIHI : high order byte van timer.

Uit tabel 7 blijkt dat er alleen een locatie binnen het timer-gedeelte wordt geselecteerd als A2, A1 en A0 de combinatie 100_2 of 101_2 vormen. Er zijn in het timer-gedeelte dus maar 2 locaties die gevuld en uitgelezen kunnen worden.

Vraag 26: Als A2 = 0 wordt een locatie binnen het I/O-/timer-gedeelte geselecteerd.

Door I/O-adressen, waarin A2 = 0, wordt een van de 4 locaties binnen het I/O-gedeelte geselecteerd (zie tabel 4). In de 8155 zijn dus totaal 6 locaties d.m.v. I/O mapped I/O te adresseren.

b. Count length register

Vraag 27: Om het gehele count length register te vullen, zijn 1/2/4 output-instructies nodig.

Het count length register bestaat uit 16 bits, dus 2 bytes. b_0 t/m b_{13} bevatten de beginwaarde, die bij het starten van de timer naar de overeenkomstige 14 bits van de down counter worden overgebracht. b_{14} en b_{15} (M1 en M2) dienen om een van de 4 timer-modes te programmeren.
Met M1 wordt aangegeven of de timer, na het bereiken van de waarde 0, gestopt moet blijven (M1 =0) of automatisch opnieuw moet worden gestart (M1 = 1). M2 geeft aan welke vorm het TIMER OUT-signaal moet hebben. Dit kan een korte impuls (M2 = 1) of een symmetrische blokgolf (M2 = 0) zijn. Fig.20 geeft een overzicht van de vormen, die het TIMER OUT-signaal in elk van de 4 timer-modes heeft.

In elk van de 4 timing-diagrammen wordt de timer op t = 1 gestart en is op t = 3 de inhoud van de timer tot 0 gedaald. De tijd tussen t = 1 en t = 3 is dus de tijd, die we d.m.v. b_0 t/m b_{13} van het count length register hebben ingesteld.

In mode 1 is met M2 = 0 de symmetrische blokgolf geselecteerd. Als de timer-inhoud tot de helft van de beginwaarde is teruggeteld, dan wordt het TIMER OUT-signaal actief (t = 2). Dit signaal blijft actief, totdat de timer-inhoud 0 geworden is (t = 3). TIMER OUT wordt dan 1 en de timer stopt.

Mode 2 verloopt op dezelfde wijze, met dat verschil, dat op t = 3, t = 4, enz., de timer opnieuw vanuit het count length register wordt gevuld, en direct weer wordt gestart. Het TIMER OUT-signaal is dus een continu doorgaande blokgolf, totdat de timer d.m.v. een instructie wordt gestopt.

In mode 3 telt de timer gedurende de geprogrammeerde tijd. Als de timer-inhoud 0 geworden is, wordt er een korte impuls op de TIMER OUT-lijn afgegeven en de timer stopt.

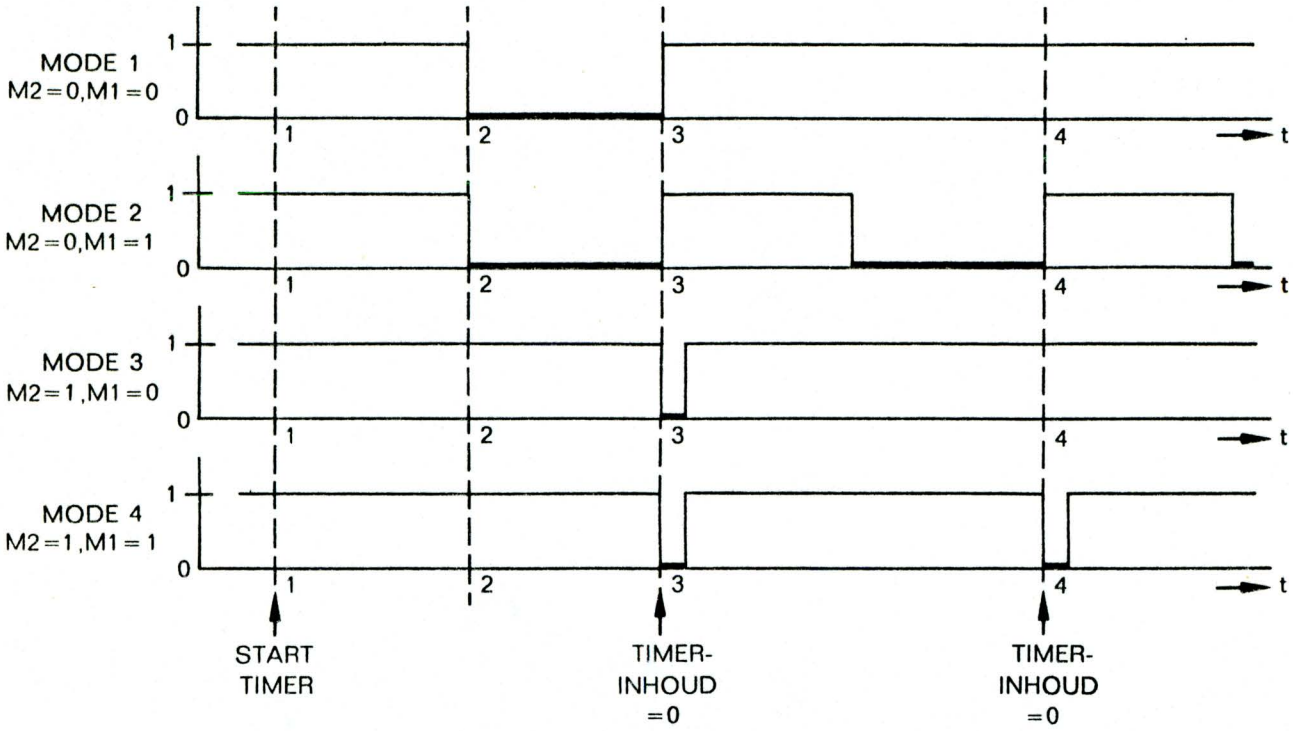


fig. 20

In mode 4 gebeurt hetzelfde, met het verschil, dat op t = 3, t = 4, enz., de timer automatisch opnieuw wordt gevuld en gestart.

Het count length register kan alleen worden gevuld (m.b.v. 2 output-instructies). Wanneer we een input-opdracht met het I/O-adres van CLRLO of CLRHI laten uitvoeren, dan wordt de overeenkomstige byte van de timer uitgelezen.

c. Timer

De timer zelf is een 14-bits down counter, die bij het starten wordt gevuld vanuit b_0 t/m b_{13} van het count length register. Door elke klokimpuls wordt de timer-inhoud met 1 verlaagd. Als de inhoud tot de helft van de beginwaarde of tot 0 is gedaald, worden er signalen aan de timer control afgegeven. Deze bepaalt hiermee de vorm van het TIMER OUT-signaal en of de timer wel of niet direct opnieuw moet worden gestart.

D.m.v. twee input -instructies kan de inhoud van de timer worden uitgelezen. Dit kan van belang zijn, als ergens in een programma moet worden bepaald hoe lang het nog duurt, voordat de timer-inhoud 0 wordt.

d. Starten en stoppen van de timer

Met b_{14} en b_{15} (M1 en M2) van het count length register, bepalen we in welke mode de timer moet opereren. Met b_6 en b_7 (TM1 en TM2) van het command register kunnen we de timer laten starten en stoppen. In tabel 8 zijn de mogelijkheden hiervoor weergegeven.

TM2	TM1	Gevolgen
0	0	Er verandert niets. Als de timer bezig is te tellen, blijft deze doorgaan.
0	1	De timer wordt direct gestopt.
1	0	De timer stopt als de timer-inhoud 0 geworden is. (Als de timer al gestopt was, dan blijft deze gestopt.)
1	1	De timer wordt vanuit het count length register gevuld en gestart. Als de timer al gestart was, dan wordt deze opdracht pas uitgevoerd als de timer-inhoud 0 geworden is.

Tabel 8

We zullen dit verduidelijken in een voorbeeld. Stel, dat de timer al is gestart, maar dat we de timer direct opnieuw willen starten.

Vraag 28: Hiervoor moeten we 1/2/3 maal TM2 en TM1 veranderen.

In tabel 8 komt geen combinatie van TM2 en TM1 voor om de timer direct opnieuw te starten, als deze reeds gestart was. We zullen dus de timer eerst moeten stoppen (met TM2 = 0 en TM1 = 1) en meteen weer starten (met TM2 = 1 en TM1 = 1).

Wanneer we de timer willen starten, moeten we zorgen dat eerst het count length register is gevuld.

Vraag 29: Het starten van de timer beïnvloedt de inhoud van het count length register wel/niet.

Bij het starten van de timer wordt de down counter vanuit het count length register gevuld. De inhoud van het count length register blijft daarbij ongewijzigd. Dit houdt in, dat als de timer een aantal malen gedurende dezelfde tijd moet tellen, we niet steeds het count length register behoeven te vullen. Na de eerste keer kunnen we volstaan met het geven van het commando start timer. Dit doen we dus d.m.v. b_7 en b_6 (TM2 en TM2) van het command register.

Omdat b_0 t/m b_5 van dit command register worden gebruikt voor het programmeren van het I/O-gedeelte, moeten we zorgen dat het starten en stoppen van de timer geen invloed heeft op deze bits. Hoe dit gerealiseerd kan worden, is in paragraaf 13 beschreven.

Antw.28: 2. Antw.29: niet.

SAMENVATTING 10

39. Het gebruik van een hardware timer heeft als voordelen
    a. er is een nauwkeurige tijdmeting mogelijk.
    b. de CPU kan gedurende de tijdmeting doorgaan met het uitvoeren van het programma.

40. De timer in de 8155 kan in 4 modes opereren, nl.
    a. enkele symmetrische blokgolf,
    b. continue blokgolf,
    c. enkele TIMER OUT-impuls,
    d. continue TIMER OUT-impulsen.
    De gewenste mode wordt met M1 en M2 van het count length register geprogrammeerd.

41. De timer kan m.b.v. TM1 en TM2 van het command register worden gestart en gestopt.

42. Bij het starten wordt de down counter vanuit het count length register gevuld.

43. De timer-inhoud is op elk moment m.b.v. 2 input-instructies uit te lezen.

13. DE BASIC 8155 IN DE SDK 85

In de voorgaande paragrafen zijn de eigenschappen en mogelijkheden van de RAM-I/O-timer chip 8155 behandeld. In deze paragraaf wordt beschreven hoe de 8155 op de plaats van de basic RAM-I/O chip is benut. Hierin is de 8155 volgens fig.21 met de CPU verbonden.

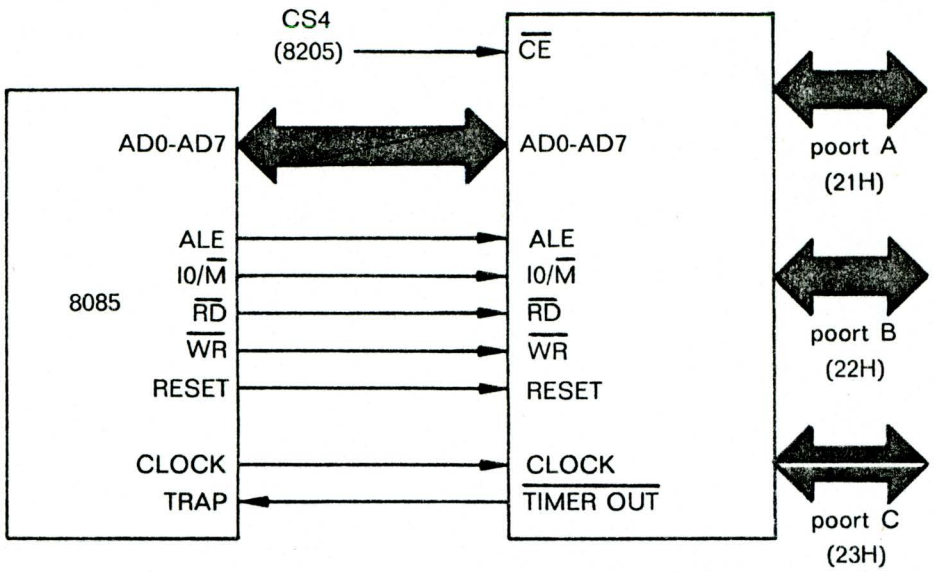


fig. 21

Vraag 30: Er wordt een RAM-locatie geselecteerd als IO/$\overline{M}$ = 0/1.
Het RAM-gedeelte wordt geselecteerd bij de adressen .....$_{16}$
t/m .....$_{16}$. Dit zijn ..... adressen.
De capaciteit van het RAM-gedeelte is ..... bytes.

Om een geheugenwoord in het RAM-gedeelte te selecteren, moet gelden IO/$\overline{M}$ = 0 en $\overline{CE}$ = 0. $\overline{CE}$ is verbonden met de CS4-uitgang van de adres-decoder 8205. Deze uitgang is 0, als er zich op de adresbus een adres bevindt, waarvan A15 t/m A11 de combinatie 00100_2 vormen. Dit zijn de adressen 2000_{16} t/m $27FF_{16}$, dus totaal 2048 verschillende adressen.

Het RAM-gedeelte van de 8155 heeft een capaciteit van 256 bytes. De interne adresbus binnen de 8155 is dan ook 8 bits breed. Intern werkt de 8155 dus met de RAM-adressen 00_{16} t/m FF_{16}.
Doordat A10, A9 en A8 van de adresbus geen rol spelen bij de adresselectie binnen het RAM-geheugen, wordt elk van de RAM-locaties geselecteerd met 8 verschillende adressen. Dit is in tabel 9 vermeld.

Antw.30: 0; 2000; 27FF; 2048; 256.

intern adres	wordt geselecteerd bij de adressen
00	2000, 2100, 2200, 2300, 2400, 2500, 2600 en 2700
01	2001, 2101, 2201, 2301, 2401, 2501, 2601 en 2701
02	2002, 2102, 2202, 2302, 2402, 2502, 2602 en 2702
03	2003, 2103, 2203, 2303, 2403, 2503, 2603 en 2703
.	.
.	.
.	.
FE	20FE, 21FE, 22FE, 23FE, 24FE, 25FE, 26FE en 27FE
FF	20FF, 21FF, 22FF, 23FF, 24FF, 25FF, 26FF en 27FF

Tabel 9

Dit houdt in, dat voor de 256 RAM-locaties in de basic 8155 2048 adressen uit de totale adrescapaciteit van de CPU worden bezet.

Vraag 31: Het I/O- en het timer-gedeelte worden geselecteerd als IO/$\overline{M}$ = 0/1 en $\overline{CE}$ = 0/1.
In de basic 8155 komen de I/O-adressen .....$_{16}$ t/m .....$_{16}$ voor.

Er wordt een locatie binnen het I/O-gedeelte of het timer-gedeelte geselecteerd als IO/$\overline{M}$ = 1. Natuurlijk moet dan de gehele chip zijn geactiveerd met $\overline{CE}$ = 0. Aangezien een I/O-adres zowel via de high als de low order byte van de adresbus wordt verstuurd, is $\overline{CE}$ = 0 bij de I/O-adressen 00100XXX.
Dit zijn dus 00100000_2 t/m 00100111_2 = 20_{16} t/m 27_{16}. Hiervan worden er in de 8155 maar 6 gebruikt, nl. 4 voor het I/O-gedeelte en 2 voor het timer-gedeelte. In tabel 10 zijn deze I/O-adressen met de bijbehorende locaties weergegeven.

I/O-adres	geselecteerde locatie
20_{16}	command status register
21_{16}	I/O-poort A
22_{16}	I/O-poort B
23_{16}	I/O-poort C
24_{16}	low order byte van CLR (timer)
25_{16}	high order byte van CLR (timer)
CLR = count length register.	

Tabel 10

Vraag 32: Door de instructie OUT 20H wordt het command/status register geselecteerd.

Het command status register is verdeeld in het command register, te vullen met OUT 20H, en het status register, uit te lezen met IN 20H.

Vraag 33: De timer in de basic 8155 mag wel/niet door de gebruiker worden aangesproken.

 Antw.31: 1; 0; 20; 25. Antw.32: command. Antw.33: niet.

De timer in de basic 8155 is toegewezen aan de SINGLE STEP-functie van de monitor van de SDK 85. Door een actief TIMER OUT-signaal wordt altijd naar de interrupt service routine van deze SINGLE STEP gesprongen. Dit houdt in dat we deze timer niet voor een ander doel mogen gebruiken.

Hierbij is echter nog een belangrijk punt, dat zowel de monitor als de gebruiker het command register aanspreekt. De monitor moet immers de timer starten en stoppen en de gebruiker wil vaak het I/O-gedeelte programmeren. Dit is mogelijk, mits de monitor alleen b_6 en b_7 van het command register beïnvloedt en de gebruiker zich tot b_0 t/m b_5 beperkt. Aangezien het command register alleen in zijn geheel is te vullen (dus alle 8 bits tegelijk) moeten er twee oplossingen worden gevonden.

a. Als de gebruiker het command register vult, moet hij(zij) ervoor zorgen, dat de werking van de timer niet wordt beïnvloed.

Vraag 34: b_6 moet dan worden gevuld met 0/1 en b_7 met 0/1.

De gebruiker moet b_6 en b_7 van het command register dan vullen met 0. Dit heeft nl. geen invloed op het gedrag van de timer (zie tabel 8).

b. Als de monitor het command register vult, moet in b_0 t/m b_5 die combinatie worden gezet, die de gebruiker er eerder heen had gestuurd. Hiertoe moet de monitor weten welke combinatie dit was.

Vraag 35: Het command register is wel/niet uit te lezen.

De inhoud van het command register is echter niet terug te lezen. De gebruiker moet in dit geval zorgen, dat de bitcombinatie, die naar het command register wordt gestuurd, tevens op een door de monitor adresseerbare locatie wordt geplaatst.

In de SDK 85 is dit adres $20FF_{16}$ in het RAM-geheugen. Dit is de reden, dat in voorgaande lessen bij het initialiseren van de 8155 steeds de instructie STA 20FFH is opgenomen. Wanneer de monitor nu het command register aan moet spreken, "kijkt" deze eerst op adres $20FF_{16}$ welke combinatie de gebruiker in b_0 t/m b_5 wil hebben. Hiervoor wordt dan de combinatie voor b_6 en b_7 geplaatst. Zo voert de monitor onderstaande instructies uit om de timer te starten.

```
LDA  20FFH
ORI  C0H
OUT  20H
```

Door de ORI-instructie worden b_7 en b_6 met 1 gevuld, terwijl b_0 t/m b_5 ongewijzigd blijven.

Antw.34: 0; 0. Antw.35: niet.

14. DE EXPANSION 8155 IN DE SDK 85

In fig.22 is weergegeven hoe de expansion RAM-I/O-timer chip 8155 met de CPU is verbonden.

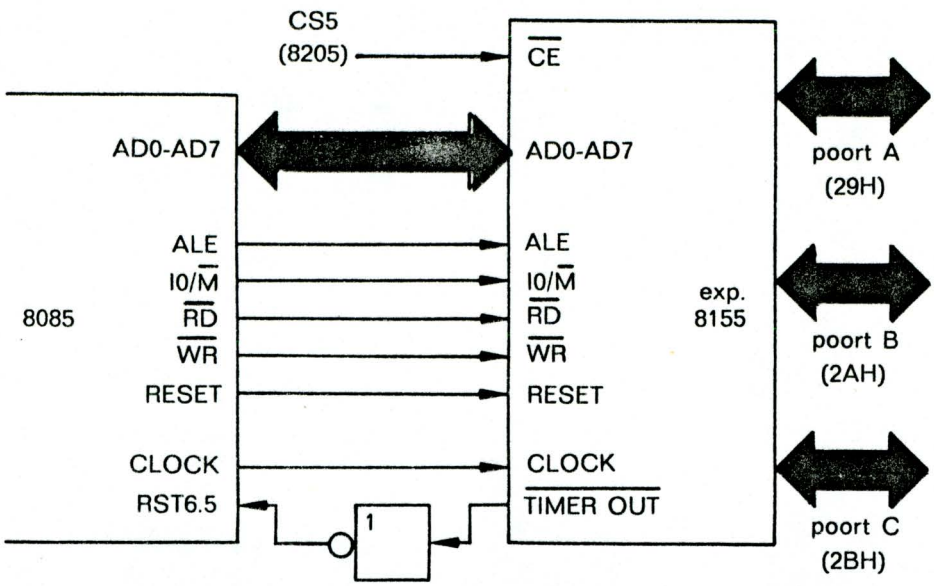


fig.22

Ook in de expansion 8155 wordt een geheugenwoord in het RAM-gedeelte door 8 verschillende adressen geselecteerd. Zo kunnen we b.v. het geheugenwoord 2800_{16} ook adresseren met 2900_{16}, $2A00_{16}$, $2B00_{16}$, $2C00_{16}$, $2D00_{16}$, $2E00_{16}$ en $2F00_{16}$.

De I/O-adressen in de expansion 8155 zijn in tabel 11 vermeld.

I/O-adres	geselecteerde locatie
28_{16}	command status register
29_{16}	I/O-poort A
$2A_{16}$	I/O-poort B
$2B_{16}$	I/O-poort C
$2C_{16}$	low order byte van CLR (timer)
$2D_{16}$	high order byte van CLR (timer)
CLR = count length register	

Tabel 11

Het gebruik van de timer in de expansion 8155 is in de les "Systeemeigenschappen hardware" al besproken. We beperken ons hier tot een verklaring, waarom de timer steeds in mode 2 (zie fig.20) is geprogrammeerd.
Als de timer nl. in mode 3 opereerde, dan zou dit boven mode 2 de volgende voordelen hebben gehad.

a. Na een TIMER OUT-impuls blijft de timer gestopt. We behoeven de timer dan niet m.b.v. 2 extra instructies te stoppen.
b. De maximale tijd tussen starten en de TIMER OUT-impuls zou twee maal zo lang zijn.

De TIMER OUT-uitgang is via een NIET-poort (inverter) met de RST6.5-ingang van de CPU verbonden.
De impulsen, die op de TIMER OUT-lijn in mode 3 (of mode 4) optreden, zijn echter te kort voor de CPU, om gedetecteerd te kunnen worden.
Daarom moet voor mode 1 of 2 worden gekozen.
Omdat de timer na elke actie wordt gestopt, is er in dit geval geen verschil tussen mode 1 en mode 2. U mag dus beide gebruiken.
Het bijbehorende timing diagram is in fig.23 weergegeven.

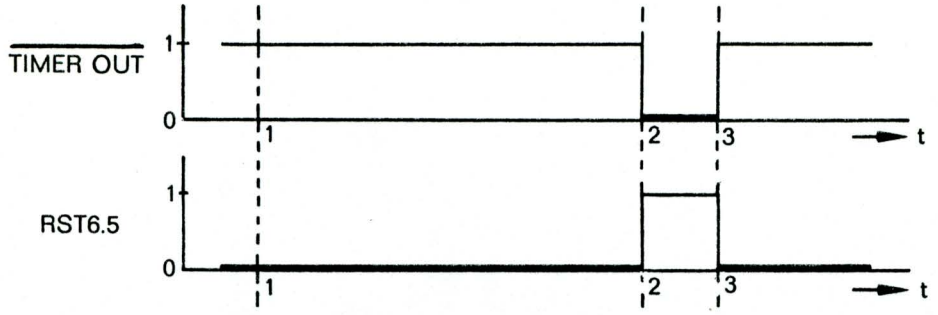


fig. 23

Op t = 1 wordt de timer gestart. Op t = 2 is de timer-inhoud tot de helft van de beginwaarde gedaald. Het TIMER OUT-signaal wordt laag, RST6.5 wordt hoog. De timer blijft nog doortellen, want de inhoud is nog niet 0 geworden. Door het honoreren van de interrupt request RST6.5 wordt naar de bijbehorende interrupt service routine gesprongen. Aan het begin hiervan wordt de timer gestopt (t = 3) en het TIMER OUT-signaal wordt hoog, RST6.5 wordt laag.

Opmerking:

De NIET-poort tussen TIMER OUT en RST6.5 is noodzakelijk, omdat de RST6.5-ingang van de CPU high level sensitive is (zie de les "Interrupt").

14. DE EXPANSION 8155 IN DE SDK 85

In fig.22 is weergegeven hoe de expansion RAM-I/O-timer chip 8155 met de CPU is verbonden.

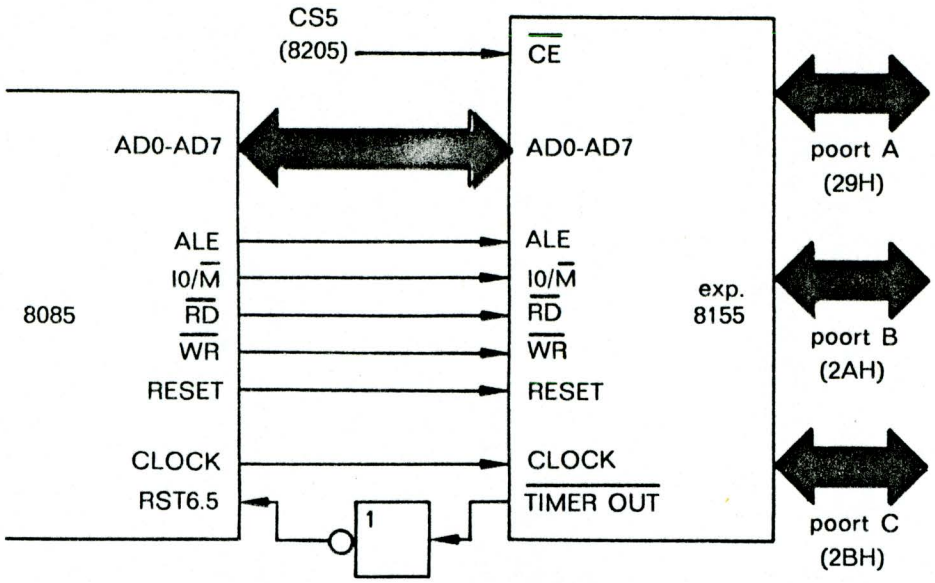


fig.22

Ook in de expansion 8155 wordt een geheugenwoord in het RAM-gedeelte door 8 verschillende adressen geselecteerd. Zo kunnen we b.v. het geheugenwoord 2800_{16} ook adresseren met 2900_{16}, $2A00_{16}$, $2B00_{16}$, $2C00_{16}$, $2D00_{16}$, $2E00_{16}$ en $2F00_{16}$.

De I/O-adressen in de expansion 8155 zijn in tabel 11 vermeld.

I/O-adres	geselecteerde locatie
28_{16}	command status register
29_{16}	I/O-poort A
$2A_{16}$	I/O-poort B
$2B_{16}$	I/O-poort C
$2C_{16}$	low order byte van CLR (timer)
$2D_{16}$	high order byte van CLR (timer)
CLR = count length register	

Tabel 11

Het gebruik van de timer in de expansion 8155 is in de les "Systeemeigenschappen hardware" al besproken. We beperken ons hier tot een verklaring, waarom de timer steeds in mode 2 (zie fig.20) is geprogrammeerd.
Als de timer nl. in mode 3 opereerde, dan zou dit boven mode 2 de volgende voordelen hebben gehad.

a. Na een TIMER OUT-impuls blijft de timer gestopt. We behoeven de timer dan niet m.b.v. 2 extra instructies te stoppen.
b. De maximale tijd tussen starten en de TIMER OUT-impuls zou twee maal zo lang zijn.

De TIMER OUT-uitgang is via een NIET-poort (inverter) met de RST6.5-ingang van de CPU verbonden.
De impulsen, die op de TIMER OUT-lijn in mode 3 (of mode 4) optreden, zijn echter te kort voor de CPU, om gedetecteerd te kunnen worden.
Daarom moet voor mode 1 of 2 worden gekozen.
Omdat de timer na elke actie wordt gestopt, is er in dit geval geen verschil tussen mode 1 en mode 2. U mag dus beide gebruiken.
Het bijbehorende timing diagram is in fig.23 weergegeven.

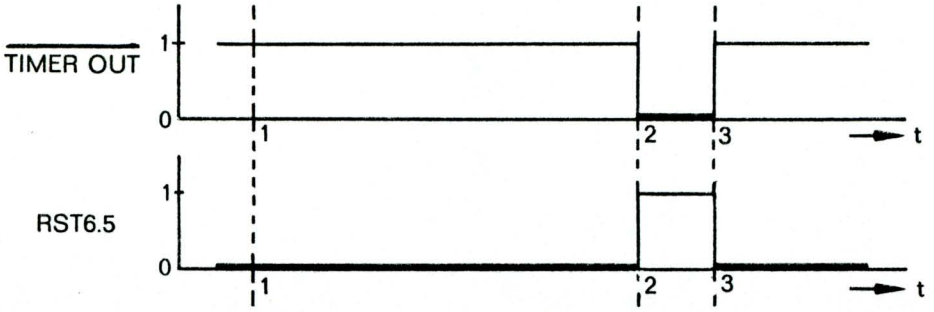


fig. 23

Op t = 1 wordt de timer gestart. Op t = 2 is de timer-inhoud tot de helft van de beginwaarde gedaald. Het TIMER OUT-signaal wordt laag, RST6.5 wordt hoog. De timer blijft nog doortellen, want de inhoud is nog niet 0 geworden. Door het honoreren van de interrupt request RST6.5 wordt naar de bijbehorende interrupt service routine gesprongen. Aan het begin hiervan wordt de timer gestopt (t = 3) en het TIMER OUT-signaal wordt hoog, RST6.5 wordt laag.

Opmerking:

De NIET-poort tussen TIMER OUT en RST6.5 is noodzakelijk, omdat de RST6.5-ingang van de CPU high level sensitive is (zie de les "Interrupt").

SAMENVATTING 11

44. Omdat bij de adresselectie van RAM-geheugenwoorden in de SDK 85 A10, A9 en A8 geen rol spelen, kan elke RAM-locatie door 8 verschillende adressen worden geselecteerd.

45. Omdat zowel de monitor als de gebruiker van de SDK 85 het command register in de basic 8155 aanspreekt, moeten beide rekening houden met de bits, die niet veranderd mogen worden.

46. Omdat de $\overline{\text{TIMER OUT}}$-impuls van de expansion 8155 te kort is voor de RST6.5-ingang van de CPU, moet de timer zodanig worden geprogrammeerd, dat op de $\overline{\text{TIMER OUT}}$-lijn een symmetrische blokgolf optreedt.

15. SLOTOPMERKINGEN

De in deze les beschreven programmeerbare chips 8355 en 8155 zijn slechts voorbeelden uit een grote reeks. Ze kunnen de systeemontwerper veel gemak bieden.
Men moet er evenwel steeds op letten, dat het ook noodzakelijk is zo'n programmeerbare chip te gebruiken.
B.v. wanneer men een eenvoudige serial interface nodig heeft, dan kan dat met een 8251 programmeerbare serial interface chip gebeuren.
Maar misschien is de serie in- en uitgang van de 8085 al voldoende.
Vooropgezet dat de tijd toereikend is.

In het algemeen bevatten alle programmeerbare chips veel hardware, die door het "setten" van "schakelaars" door software commando's geselecteerd en geactiveerd worden.
Daarnaast zijn zij zo intelligent, dat zij eenvoudige tot zeer moeilijke taken zelfstandig, dus zonder tussenkomst van de CPU, kunnen uitvoeren.
Dit leidt zelfs tot het toepassen van microprocessoren in gebieden, waar vroeger minicomputers gebruikt werden.

Voorbeelden van zulke intelligente programmeerbare chips zijn o.a.

- Floppy disk controller
- CRT controller
- Serial interface controller
- Cassette controller
- Dot matrix printer controller.

Er zijn zelfs chips, die eigenlijk een complete microcomputer bevatten.
Deze zijn zodanig opgebouwd, dat ze voor velerlei interface-toepassingen geschikt zijn.
Dit noemt men Universal Peripheral Interface microcomputers (b.v. de 8041). Hiermee kan een speciale intelligente interface worden gemaakt, door het schrijven van een programma, dat in een ROM in deze single chip microcomputer wordt opgeslagen.
Via een databus kan met een gewoon microprocessorsysteem data en commando's worden uitgewisseld net zoals bij b.v. de 8155.

DIGITALE KLOK.

Digitale klok

1. INLEIDING

In de voorgaande lessen bent u twee programma's tegengekomen, waarin sprake was van timing. Het eerste was het programma, waarin m.b.v. de monitor-subroutine "DELAY" een tijd van exact 0,5 s werd gerealiseerd (voorbeeld 3 van de les "Programmavoorbeelden-2"). Het tweede programma was "COUNT", waarin m.b.v. een hardware timer een tijd van ca. 1 ms werd ingesteld (voorbeeld 2 van de les "Programmavoorbeelden-3").

In deze les gaan we m.b.v. een microcomputer een digitale klok realiseren. Hierbij speelt de timing natuurlijk een zeer belangrijke rol.

Uitgaande van de probleemomschrijving zullen we een zo algemeen mogelijke probleemanalyse uitvoeren. Dat wil zeggen de te ontwikkelen oplossingmethode moet voor elk willekeurig type microcomputer gelden. Pas wanneer we overgaan van het algemeen stroomdiagram op een gedetailleerd stroomdiagram, en daarna op een programma, dan moeten we wel rekening houden met de eigenschappen van een bepaalde microcomputer en de instructieset van de hierin aanwezige microprocessor.

In deze les gaan we in de laatste fase van de programma-ontwikkeling uit van de SDK 85 en de instructieset van de 8085. Wanneer u echter de digitale klok m.b.v. een andere microcomputer wilt realiseren, dan dient u op dat moment de machine-gerichte details aan te passen aan de door u gewenste microcomputer. De probleemanalyse en het algemeen stroomdiagram, dus de oplossingsmethode, blijven wel gelijk.

2. PROBLEEMOMSCHRIJVING

Gevraagd wordt een digitale klok te ontwikkelen, die voldoet aan het hardware-schema van fig.1.

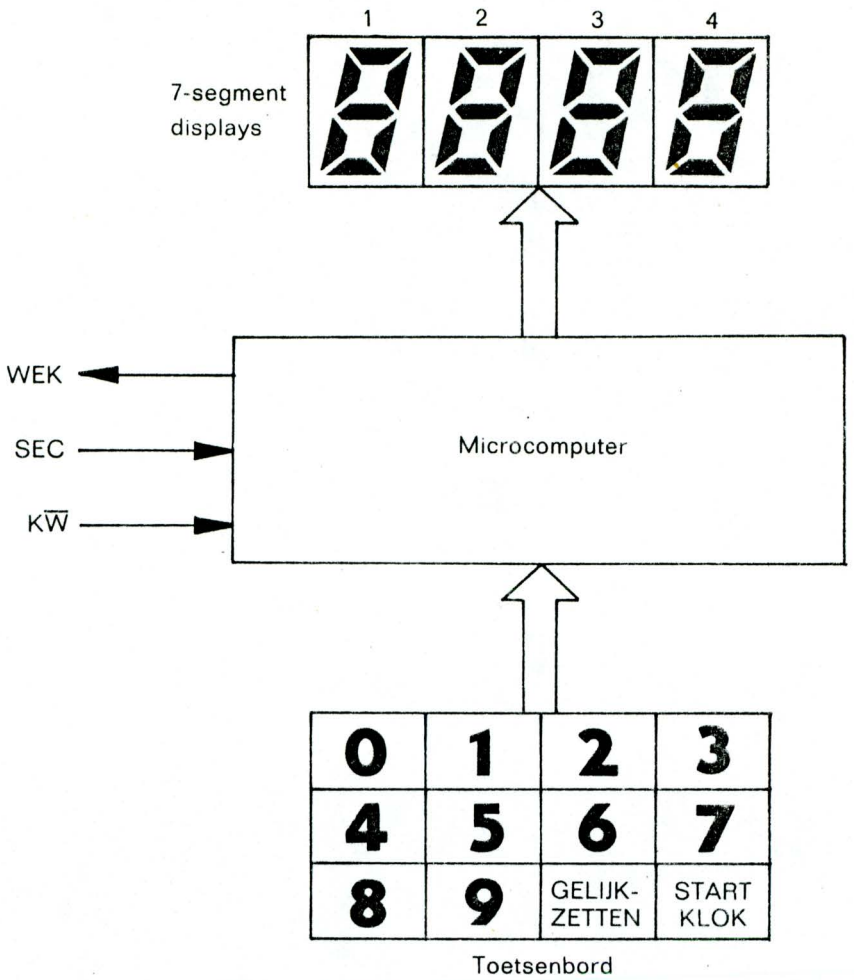


fig.1

De klok bestaat uit een microcomputer, die is voorzien van vier 7-segment_displays, een toetsenbord, een uitgang WEK en twee ingangen SEC en KW̄.

De klok moet een ingebouwde wekkerfunctie hebben, die m.b.v. KW̄ wel of niet kan worden geselecteerd.

Als KW̄ = 1, dan moet de schakeling als digitale klok (zonder wekkerfunctie) werken. Op de displays moet de juiste tijd zichtbaar worden gemaakt, waarbij het signaal op de ingang SEC aangeeft op welke wijze dit gebeurt.
SEC = 0: Op de displays worden de uren (displays 1 en 2) en de minuten (displays 3 en 4) weergegeven.
SEC = 1: Op de displays 3 en 4 worden de seconden weergegeven. De displays 1 en 2 zijn dan gedoofd.

De klok kan d.m.v. het toetsenbord gelijk worden gezet. Dit gelijkzetten moet als volgt verlopen.
1. Er wordt eerst op de toets "GELIJKZETTEN" gedrukt.
2. Daarna wordt m.b.v. de decimale toetsen de juiste tijd ingetypt.
3. Door het indrukken van de toets "START KLOK" moet de klok weer gaan lopen.

Ad 2: De in te voeren tijd bestaat uit de uren en minuten. Het aantal seconden moet na het indrukken van "START KLOK" eerst op nul worden gesteld.

Als $K\overline{W} = 0$, moet de schakeling als digitale wekkerklok functioneren. Het weergeven van de tijd gaat zoals reeds is beschreven. Nu moet echter de weergegeven tijd steeds worden vergeleken met een vooraf ingevoerde wektijd. Als deze wektijd is bereikt, dan moet gedurende 1 seconde een 1 op de uitgang WEK worden geplaatst.

Het invoeren van de wektijd moet als volgt verlopen.
1. Eerst wordt de toets "GELIJKZETTEN" ingedrukt.
2. Dan wordt m.b.v. de decimale toetsen de gewenste wektijd in uren en minuten ingetypt.
3. Door het indrukken van de toets "START KLOK" zal de klok weer verdergaan met het weergeven van de juiste tijd en het vergelijken van deze tijd met de nieuw ingevoerde wektijd.

Tijdens het invoeren van de wektijd moet de klok intern blijven doorlopen, zodat na het indrukken van "START KLOK" weer direct de juiste tijd wordt weergegeven.

Aanvullende eisen

1. De klok moet een 24-uurs cyclus hebben. D.w.z. dat de tijd tot 23 uur 59 minuten wordt aangegeven. Om middernacht springt de klok weer op nul.
2. Na het indrukken van de toets "GELIJKZETTEN" moet op de displays steeds worden weergegeven welke decimale toetsen zijn ingedrukt, m.a.w. welke nieuwe tijd, resp. wektijd, is ingetypt.
3. Als het aantal uren kleiner is dan 10, dan moet i.p.v. dat er een 0 op display 1 wordt weergegeven, dit display gedoofd zijn.
4. Het te ontwikkelen programma krijgt de naam "TIME".

PROBEER NU ZELF EEN PROBLEEMANALYSE UIT TE VOEREN EN EEN ALGEMEEN STROOMDIAGRAM OP TE STELLEN. CONTROLEER DAARNA AAN DE HAND VAN DE VOLGENDE PARAGRAAF OF U DIT OP DE JUISTE WIJZE HEBT GEDAAN, M.A.W. OF U MET ALLE PROBLEMEN REKENING HEBT GEHOUDEN.

3. PROBLEEMANALYSE

Het is zeer eenvoudig om een microcomputer als digitale klok te laten functioneren.
We hoeven in feite alleen de inhoud van een tijdteller op de displays weer te geven en deze inhoud steeds te verhogen. Dit is in fig.2 weergegeven.
Aangezien de klok continu moet lopen, bestaat het programma uit een oneindige lus. De tijdsduur, die door deze lus in beslag wordt genomen, bepaalt de snelheid waarmee de inhoud van de tijdteller wordt verhoogd, dus de kleinste tijdseenheid, die door de klok kan worden aangegeven.

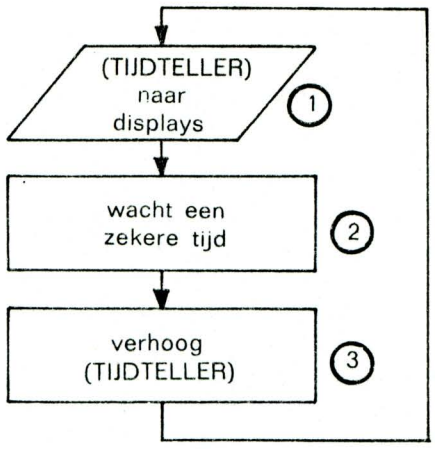


fig.2

Vraag 1: In ons geval moet de tijdteller eenmaal per uur/minuut/seconde worden verhoogd.

De door ons te ontwikkelen klok moet de tijd in seconden nauwkeurig weer kunnen geven. De tijdteller moet dus elke seconde eenmaal worden verhoogd. De "zekere tijd" in blok ② van fig.2 kan dus worden vervangen door "1 seconde".

Een andere wijziging, die we in fig.2 moeten aanbrengen, is het feit dat de inhoud van de tijdteller eventueel moet worden vergeleken met een gewenste wektijd.

Vraag 2: Dit vergelijken moet worden uitgevoerd als $K\overline{W}$ = 0/1.

Deze vergelijking behoeft alleen te worden uitgevoerd als de wekkerfunctie is geselecteerd, dus als $K\overline{W}$ = 0.

Door het aanbrengen van deze uitbreiding gaat fig.2 over in fig.3.

Fig.3 is het algemeen stroomdiagram voor het continu doorlopen van de (wekker)klok. Hierin zijn nog geen voorzieningen aangebracht voor het gelijkzetten en het invoeren van de wektijd.

Vraag 3: Deze functies worden gestart door het indrukken van de toets .....

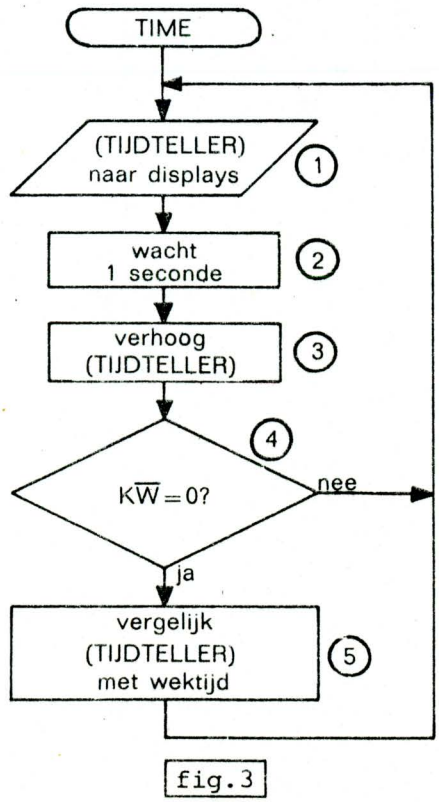


fig.3

Antw.1: seconde. Antw.2: 0.
Antw.3: "GELIJKZETTEN".

Deze beide functies beginnen pas als de toets "GELIJKZETTEN" is ingedrukt. Het detecteren of deze toets is ingedrukt, kan op twee manieren plaatsvinden.

De eerste methode is die, waarbij op een of meer plaatsen in het programma instructies zijn opgenomen, die testen of deze toets is ingedrukt.

Vraag 4: Deze methode komt overeen met geprogrammeerde/interrupt I/O.

De tweede methode is die, waarbij het indrukken van de toets "GELIJKZETTEN" de programma-uitvoering onderbreekt om eerst de klok gelijk te zetten of een nieuwe wektijd in te voeren.

Vraag 5: Deze methode komt overeen met geprogrammeerde/interrupt I/O.

De definitieve keuze tussen een oplossing op basis van geprogrammeerde I/O en een met interrupt I/O stellen we nog even uit, totdat we meer weten over de rest van het programma.

Bovendien wordt deze keuze eventueel bepaald door de eigenschappen van de te gebruiken microcomputer. Als hierin nl. geen interrupt controller aanwezig is, zijn we in de regel minder gauw geneigd een oplossing op basis van interrupt I/O te kiezen.

We gaan daarom eerst de blokken van fig.3 wat gedetailleerder bekijken.

Blok ①

In dit blok moet de juiste tijd op de 7-segment displays worden weergegeven. Hierbij bepaalt de toestand van de ingang SEC op welke wijze dit moet gebeuren.

Vraag 6: Als SEC = 0, dan moeten de uren en minuten/seconden worden weergegeven.

Als SEC = 0, moeten de uren en minuten worden weergegeven.
Als SEC = 1, dan moeten de seconden op display 3 en 4 verschijnen en display 1 en 2 moeten uit zijn.
Dit is in fig.4 weergegeven. Hierin is tevens een voorziening aangebracht om display 1 te laten doven, als hierop een 0 zou verschijnen, omdat het aantal uren kleiner dan 10 is.

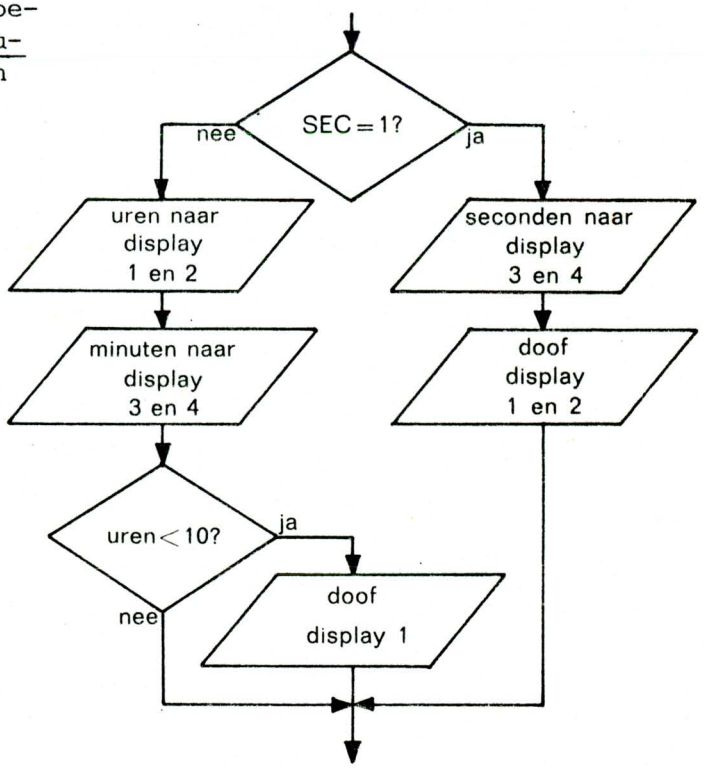


fig.4

Blok ②

In blok ② van fig.3 is aangegeven, dat er een seconde moet worden gewacht.

23-10 Antw.4: geprogrammeerde.
Antw.5: interrupt.
Antw.6: uren en minuten.

Vraag 7: Blok ② moet minder dan/exact/meer dan 1 seconde in beslag nemen.

Dit is niet helemaal correct. De tijdteller moet exact elke seconde worden verhoogd. Alle blokken samen moeten exact 1 seconde in beslag nemen. De tijd, die blok ② in beslag moet nemen, moet gelijk zijn aan 1 seconde verminderd met de tijdsduur, die de overige blokken nemen.

Vraag 8: We kunnen de tijd, die de overige blokken in beslag nemen, nu wel/niet precies berekenen.
Deze tijd is wel/niet altijd gelijk.

De tijd, die door de overige blokken in beslag wordt genomen, is nog niet te bepalen. Dat kunnen we immers pas als we zover zijn, dat we de blokken door instructies hebben vervangen. Wel kunnen we alvast concluderen, dat deze tijd niet steeds gelijk is. In fig.3 is nl. te zien dat blok ⑤ soms wel en soms niet wordt uitgevoerd. Bovendien zien we in fig.4 dat hierin óf de opdrachten uit de linker óf die uit de rechter blokken worden uitgevoerd. Het zou wel toevallig zijn dat beide groepen opdrachten dezelfde tijd in beslag nemen.

Voor het realiseren van een wachttijd zijn twee methodes.
a. Een software wachtlus.
b. Een hardware timer, die regelmatig de programma-uitvoering onderbreekt.

Omdat we geen vaste tijd voor een wachtlus kunnen bepalen, kiezen we voor de oplossing met de timer. Deze dient elke seconde een interrupt request te geven. In de bijbehorende interrupt service routine dient de tijdteller dan verhoogd te worden.
Blok ② en blok ③ van fig.3 moeten dan uit het hoofdprogramma "TIME" naar de interrupt service routine worden overgebracht.
Als we tevens blok ① vervangen door de in fig.4 weergegeven blokken, dan ontstaat het stroomdiagram van fig.5.
Hierin is de interrupt service routine aangeduid met "TMINT" (= timer interrupt).

We hebben dus zo een stroomdiagram verkregen, waarin het tempo waarmee de tijdteller wordt opgehoogd, onafhankelijk is van de tijd, die de overige opdrachten in beslag nemen. In fig.5 moeten we de blokken ③, ④ en ⑤ nog verder onderverdelen.

Blok ③

In de tijdteller moeten de uren, minuten en seconden worden bijgehouden. De tijdteller is daarom in 3 delen te splitsen, nl. de urenteller, de minutenteller en de secondenteller.
Deze tellers geven we in de stroomdiagrammen (en in het uiteindelijke programma) symbolische namen, b.v. UURTL, MINTL en SECTL.

Vraag 9: SECTL moet elke seconde ..... maal met 1 worden verhoogd.
De maximale inhoud van SECTL is ..... seconden.
MINTL moet worden verhoogd als de inhoud van SECTL van ..... naar ..... springt.

Antw.7: minder dan. Antw.8: niet; niet. Antw.9: 1; 59; 59; 0.

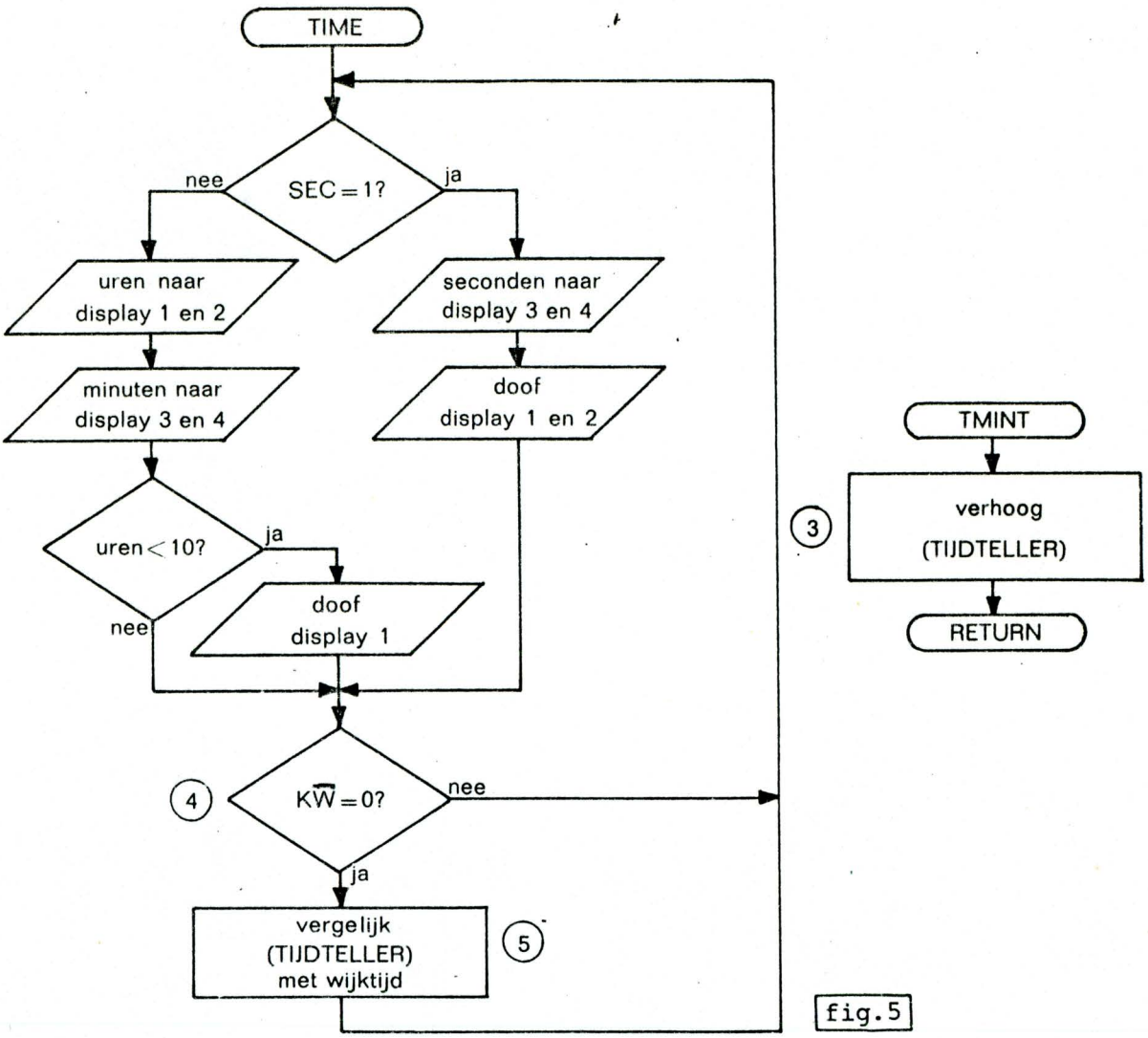


fig.5

SECTL moet elke seconde met 1 worden verhoogd. Als SECTL 60 maal is verhoogd, moet de inhoud van deze teller 0 worden en moet MINTL met 1 worden verhoogd (immers 1 minuut = 60 seconden).

Zo moet UURTL met 1 worden verhoogd, als de inhoud van MINTL van 59 op 0 springt.

Vraag 10: De maximale inhoud van UURTL is .....

10-23 Antw.10: 23.

Aangezien de klok een 24-uurs cyclus moet hebben, is de hoogste weer te geven tijd 23 uur, 59 minuten en 59 seconden.
Daarna moet de klok weer met 0 uur, 0 minuten en 0 seconden (middernacht) verdergaan.
UURTL moet dus van 23 op 0 springen.

In fig.6 is het stroomdiagram voor het verhogen van de drie tellers weergegeven.

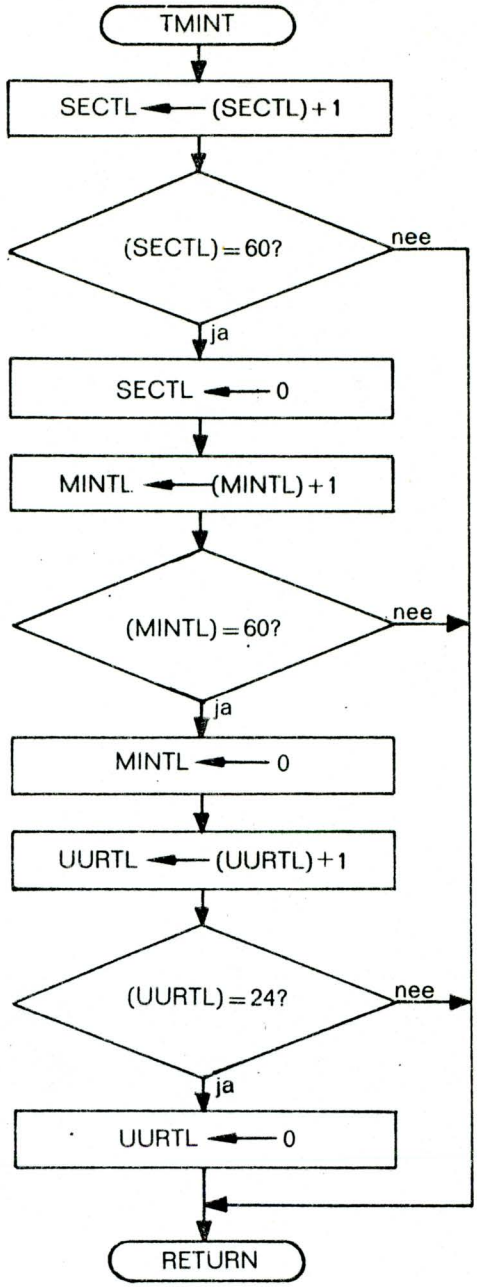


fig.6

Blok ④ en ⑤

In deze twee blokken moet, indien de wekkerfunctie is geselecteerd, de inhoud van de tijdteller worden vergeleken met de wektijd.
Zijn deze gelijk, dan moet gedurende 1 seconde een 1 op de uitgang WEK worden geplaatst.

Vraag 11: De ingevoerde wektijd is uitgedrukt in .... en .....

De wektijd bestaat uit uren en minuten. Deze moeten worden vergeleken met de inhouden van UURTL en MINTL.
We geven ook de uren en minuten van de wektijd symbolische namen, b.v. UURWK en MINWK.

Als er geldt, dat (UURTL) = (UURWK) én (MINTL) = (MINWK), dan moet de uitgang WEK 1 worden.
We zouden hiervoor het stroomdiagram van fig.7 kunnen tekenen.

Vraag 12: Als in fig.7 de inhoud van de tijdteller en de wektijd gelijk zijn, dan is WEK gedurende ..... seconde(n) 1.

Antw.11: uren; minuten. Antw.12: 60.

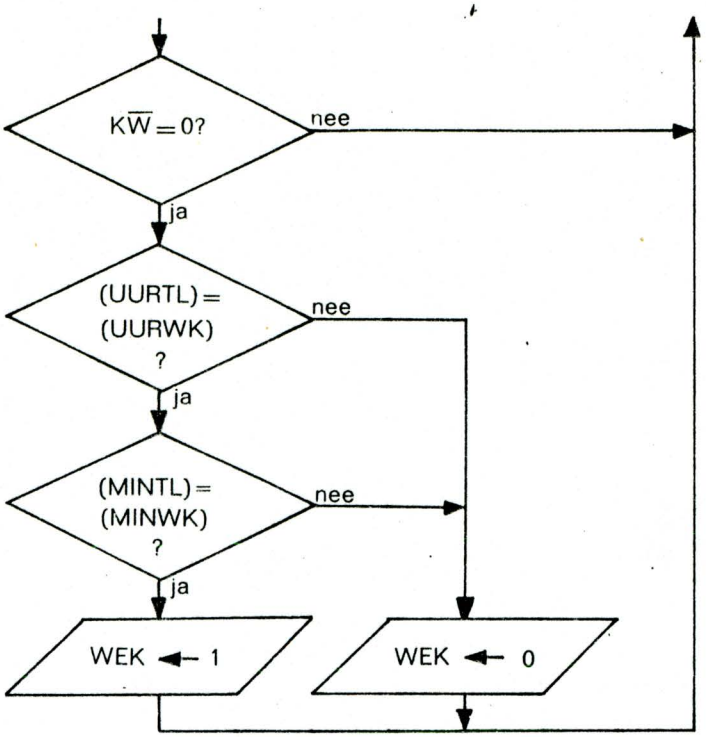


fig.7

In fig.7 zit echter nog een fout.
Als de wektijd en de inhoud van de tijdteller gelijk zijn geworden, dan zal deze toestand 1 minuut gehandhaafd blijven.
Immers, de toestand verandert pas, als de inhoud van MINTL wordt verhoogd.
De bedoeling is, dat WEK gedurende slechts 1 seconde (b.v. de eerste seconde) 1 is.

Vraag 13: WEK mag alleen 1 zijn, als (SECTL) = ...

Als we er nu voor zorgen, dat WEK alleen 1 is, als (UURTL) = (UURWK) én (MINTL) = (MINWK) én (SECTL) = 0, dan zal WEK slechts 1 seconde 1 zijn. Na 1 seconde is (SECTL) nl. niet meer 0.
Door deze voorwaarde gaat fig.7 over in fig.8.

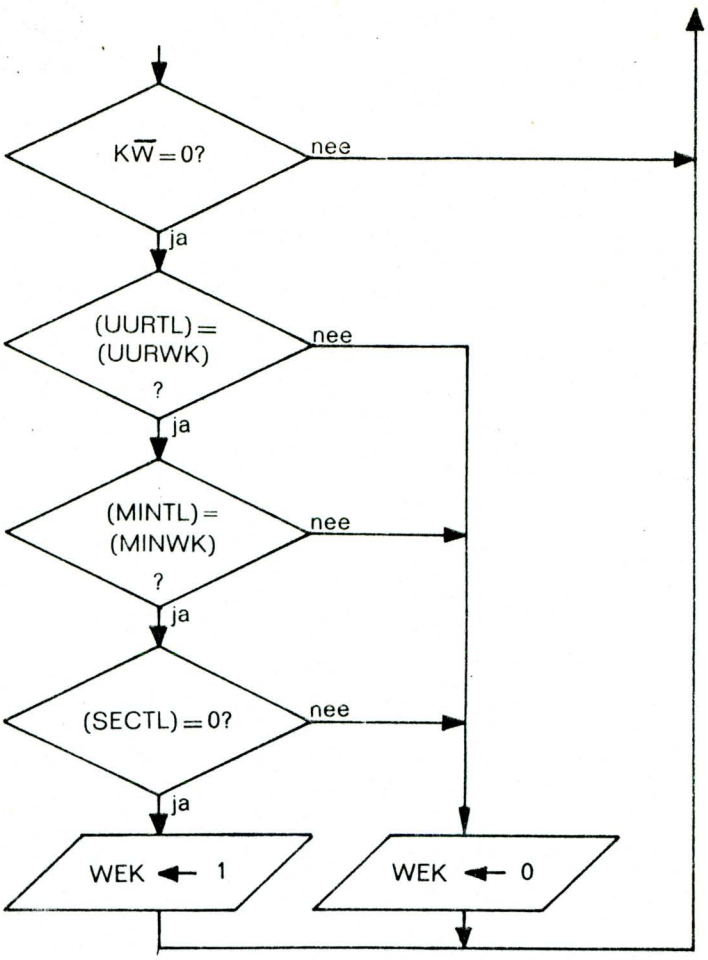


fig.8

Als we blok ④ en ⑤ van fig.5 vervangen door de blokken uit fig.8 dan ontstaat het algemeen stroomdiagram voor het hoofdprogramma "TIME". Fig.6 is het algemeen stroomdiagram voor de interrupt service routine "TMINT".

We moeten nu gaan beslissen of het gelijkzetten van de klok en het invoeren van de wektijd op basis van geprogrammeerde of interrupt I/O plaats gaat vinden.

Omdat we al een interrupt controller nodig hebben (voor de timer), kiezen we in dit geval voor interrupt I/O, dus de methode, waarbij het indrukken van de toets "GELIJKZETTEN" een interrupt request veroorzaakt. We moeten dus nu het algemeen stroomdiagram voor de bijbehorende interrupt service routine ontwikkelen.
Deze geven we b.v. de symbolische naam "GELYK".

Met het indrukken van "GELIJKZETTEN" wordt aangegeven, dat de klok zal worden gelijk gezet of dat een nieuwe wektijd wordt ingevoerd.

Vraag 14: Welk van beide functies gewenst is, wordt aangegeven door het .....-signaal.

Antw.14: KW̄.

Als $K\overline{W}$ = 1, wordt de klok gelijk gezet.
Als $K\overline{W}$ = 0, dan wordt er een nieuwe wektijd ingevoerd.
In de interrupt service routine moet dit signaal dus worden getest (fig.9).

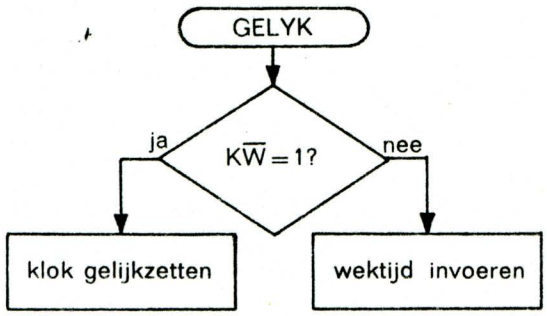


fig.9

In beide gevallen wordt daarna m.b.v. de decimale toetsen een tijd ingetypt, die afhankelijk van $K\overline{W}$ naar UURTL en MINTL of naar UURWK en MINWK moet worden overgebracht, als er op de toets "START KLOK" wordt gedrukt.
Deze toets geeft in feite aan, dat er een volledige tijd (uren en minuten) is ingetypt.

Vraag 15: Bij het gelijkzetten van de klok moet SECTL worden gevuld met .....

In de probleemomschrijving is vermeld, dat bij het gelijkzetten van de klok de seconden op nul gesteld moeten worden. Dit houdt in, dat SECTL dan met nul moet worden gevuld.
In fig.10 is het bijbehorende stroomdiagram weergegeven.

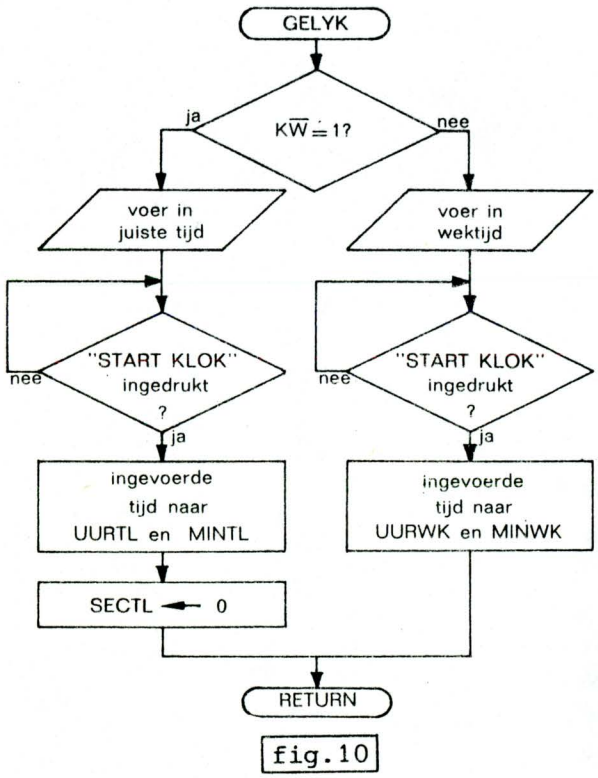


fig.10

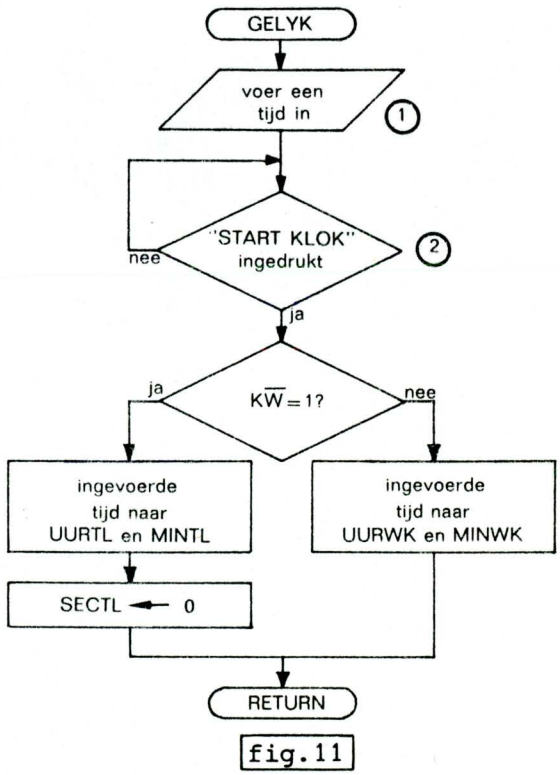


fig.11

Fig.11 is een vereenvoudiging van fig.10. Er wordt eerst een tijd ingevoerd. Pas na het indrukken van "START KLOK" wordt getest of deze ingevoerde tijd naar UURTL en MINTL of naar UURWK en MINWK moet worden overgebracht.

 Antw.15: 0.

Blok ① en ② van fig.11 moeten nog nader worden bekeken. Om een volledige tijd, bestaande uit uren en minuten, in te voeren, moeten drie of vier toetsen worden ingedrukt.
Het detecteren of er een toets is ingedrukt, en zo ja welke dit is, is vrij eenvoudig (b.v. met INPUT-opdrachten). Maar een ingedrukte toets moet resulteren in een combinatie van énen en nullen.
De drie of vier combinaties, die ontstaan door het achtereenvolgens indrukken van de decimale toetsen 0 t/m 9, moeten nog worden samengevoegd tot één volledige tijd. Dit moet bij voorkeur zodanig gebeuren, dat deze tijd rechtstreeks naar UURTL en MINTL resp. UURWK en MINWK kan worden overgebracht.

Vraag 16: De inhouden van UURTL, MINTL en SECTL zijn binaire/BCD/hexadecimale getallen.

De getallen, die zich in UURTL, MINTL en SECTL bevinden, zijn bij voorkeur BCD-getallen. We kunnen dan de inhouden van deze tellers rechtstreeks op de displays weergeven. Het is dan logisch dat de inhouden van UURWK en MINWK ook BCD-getallen zijn, zodat we deze zonder omzettingen direct met UURTL en MINTL kunnen vergelijken.

De ingevoerde tijden moeten dan ook in BCD-getallen worden omgezet.
De gemakkelijkste oplossing is dus een toetsenbord, dat direct de BCD-code voor een ingedrukte toets afgeeft.
Is dit niet het geval, dan moeten de door het toetsenbord afgegeven bitcombinaties m.b.v. een tabel of een berekening worden omgezet in BCD-getallen.

De achtereenvolgens ontvangen BCD-combinaties moeten worden opgeslagen en samengesteld tot een getal, dat de ingevoerde tijd weergeeft.

Hoe dit kan gebeuren, laten we zien aan de hand van een voorbeeld.
Stel, we voeren de tijd 14 uur en 53 minuten in. Daarvoor drukken we achtereenvolgens op de toetsen 1, 4, 5 en 3. De plaats, waar de van het toetsenbord ontvangen data wordt opgeslagen, noemen we TBBUF (= toetsenbord buffer). Als de inhoud van TBBUF 0000 was, dan wordt deze, door het indrukken van de vier toetsen, achtereenvolgens

0000_{BCD}
0001_{BCD}
0014_{BCD}
0145_{BCD}
1453_{BCD}

Door het indrukken van een toets schuift de inhoud van TBBUF één cijferplaats (= 4 bits in BCD) naar links. Op de vrijgekomen plaats wordt het nieuwe cijfer neergezet.

Vraag 17: Als we 9 uur 15 minuten willen invoeren, dan moeten we 3/4 maal een decimale toets indrukken.

Als we b.v. 9 uur en 15 minuten willen invoeren, dan behoeven we alleen de toetsen 9, 1 en 5 in te drukken. De inhoud van TBBUF is dan achtereenvolgens 0000_{BCD}, 0009_{BCD}, 0091_{BCD} en 0915_{BCD}.

Antw.16: BCD. Antw.17: 3.

Het stroomdiagram wordt dan b.v. zoals in fig.12 is weergegeven. Hierin is tevens een blok opgenomen, dat de inhoud van TBBUF op de displays weergeeft. Dit is nl. één van de aanvullende eisen uit de probleemomschrijving.
(Dit stroomdiagram is iets anders opgebouwd dan dat van fig. 11. Er wordt nl. per ingedrukte toets afgevraagd welke functie deze toets heeft.)

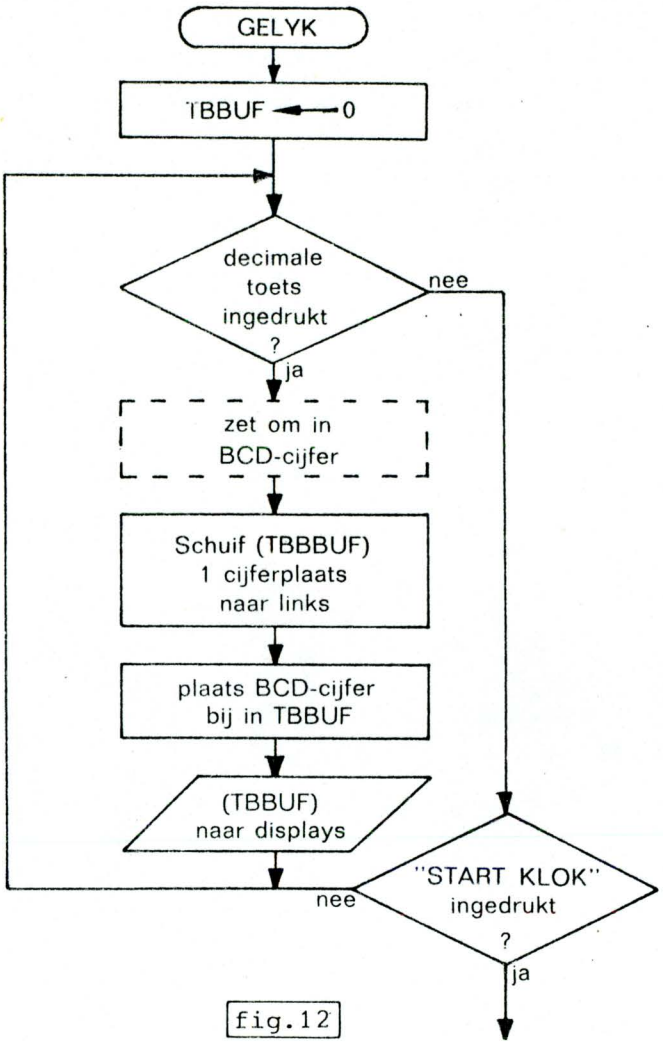


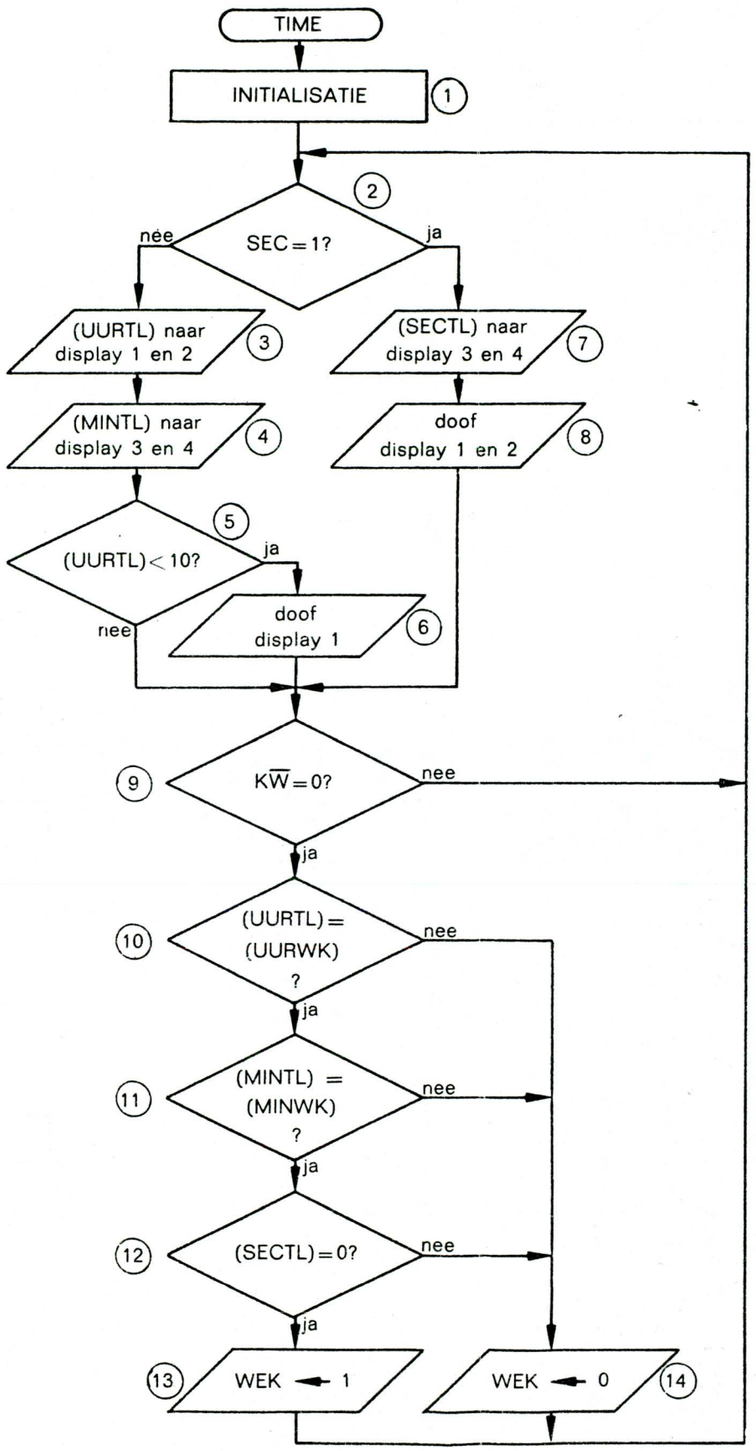
fig.12

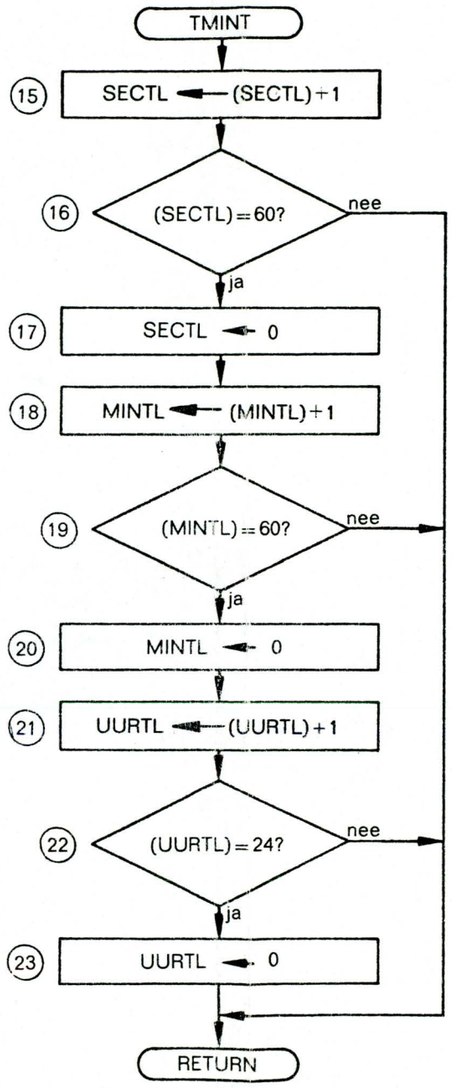
4. ALGEMEEN STROOMDIAGRAM

Door het combineren van de figuren 5, 6, 8, 11 en 12 ontstaat het algemeen stroomdiagram voor het hoofdprogramma "TIME" en de interrupt service routines "TMINT" en "GELYK". Dit is in fig.13 weergegeven.

De bedoeling is nu, dat we hieruit een gedetailleerd stroomdiagram ontwikkelen. Daarbij moeten we uitgaan van de eigenschappen van een bepaalde microcomputer. In deze les is dat natuurlijk weer de SDK 85.

Wanneer u de digitale klok met een ander type microcomputer wilt realiseren, dan zult u, op dezelfde wijze als in de volgende paragraaf is gedaan, te werk moeten gaan. Voor wat betreft de machine-gerichte details zal uw oplossing dan afwijken van de onze.
De oplossingsmethode (fig.13) blijft echter wel gelijk.


TIME
INITIALISATIE
1
2
SEC = 1?
nee
ja
(UURTL) naar display 1 en 2
3
(MINTL) naar display 3 en 4
4
(SECTL) naar display 3 en 4
7
doof display 1 en 2
8
5
(UURTL) < 10?
ja
nee
doof display 1
6
9
KW = 0?
nee
ja
10
(UURTL) = (UURWK) ?
nee
ja
11
(MINTL) = (MINWK) ?
nee
ja
12
(SECTL) = 0?
nee
ja
13
WEK ← 1
WEK ← 0
14


TMINT
15
SECTL ← (SECTL) + 1
16
(SECTL) = 60?
nee
ja
17
SECTL ← 0
18
MINTL ← (MINTL) + 1
19
(MINTL) = 60?
nee
ja
20
MINTL ← 0
21
UURTL ← (UURTL) + 1
22
(UURTL) = 24?
nee
ja
23
UURTL ← 0
RETURN

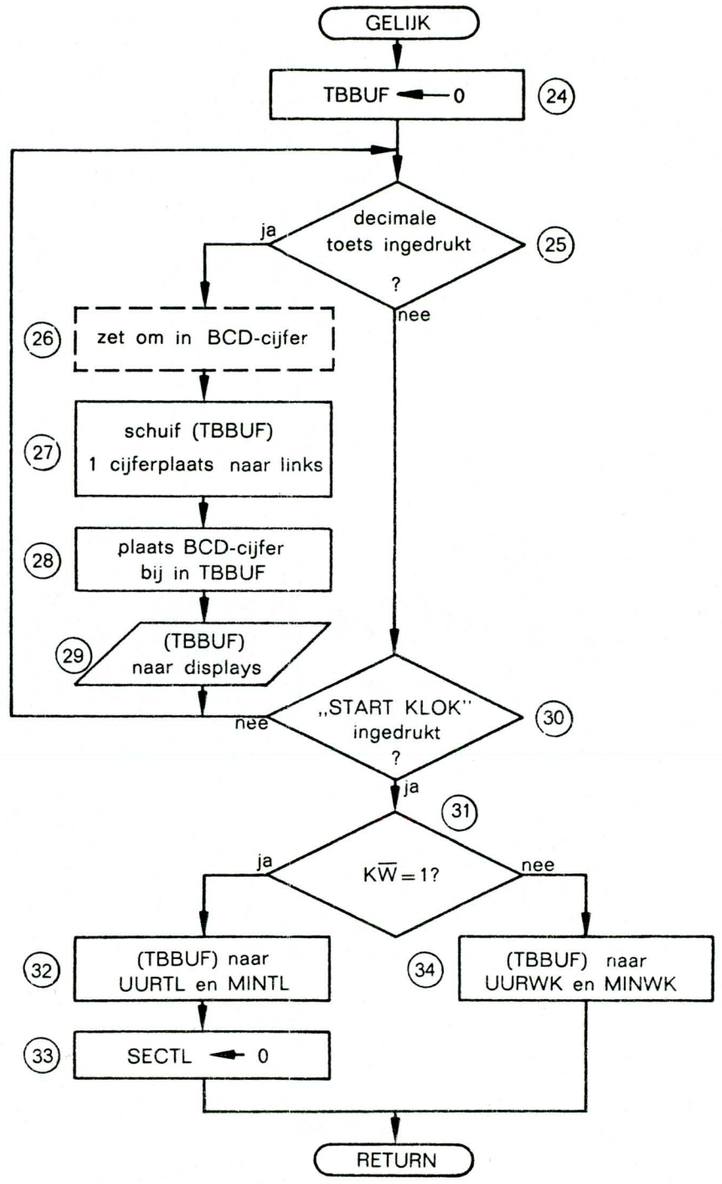

GELIJK
TBBUF ← 0
24
ja
decimale toets ingedrukt ?
25
nee
26
zet om in BCD-cijfer
27
schuif (TBBUF) 1 cijferplaats naar links
28
plaats BCD-cijfer bij in TBBUF
29
(TBBUF) naar displays
nee
„START KLOK" ingedrukt ?
30
ja
31
KW = 1?
ja
nee
32
(TBBUF) naar UURTL en MINTL
33
SECTL ← 0
34
(TBBUF) naar UURWK en MINWK
RETURN

fig.13

5. GEDETAILLEERD STROOMDIAGRAM

We gaan nu de blokken van fig.13 zodanig onderverdelen, dat elk blok in het gedetailleerd stroomdiagram door 1 of enkele instructies kan worden vervangen.

Blok ①

De initialisatie dient om een aantal beginsituaties in te stellen. In ieder geval moet de I/O-module, de stackpointer en de interrupt controller worden geïnitialiseerd. Hierbij komen eventueel nog beginvoorwaarden, waaraan i.v.m. de rest van het programma moet worden voldaan.

Blok ②

In dit blok moet het ingangssignaal SEC worden getest.
Dit signaal wordt aangeboden op een input-poort. Deze poort duiden we voorlopig aan met "INPP".
Blok ② wordt dan vervangen door de blokken van fig.14.

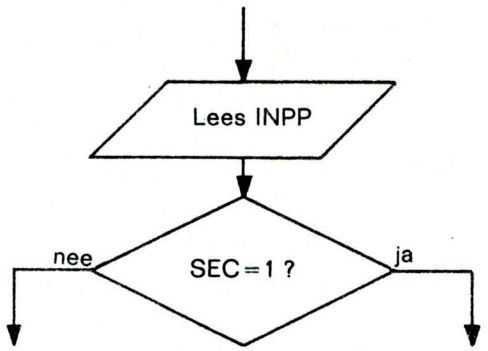


fig.14

Blok ③ en ④

Als we de 7-segment displays 1 t/m 4 van de SDK 85 gebruiken, dan kunnen we deze m.b.v. monitor-subroutines aansturen.

Vraag 18: Om informatie in hexadecimale vorm op deze 4 displays weer te geven, roepen we de monitor-subroutine "....." aan. Daartoe moet de weer te geven informatie zich in de registers ..... en ..... bevinden.

De subroutine "UPDAD" geeft de inhoud van registerpaar D,E in hexadecimale vorm weer op de displays 1 t/m 4.
Voor het aanroepen van "UPDAD" moeten de inhouden van UURTL en MINTL dus naar de registers D en E worden overgebracht.
Blok ③ en ④ gaan daarom over in de blokken van fig.15.

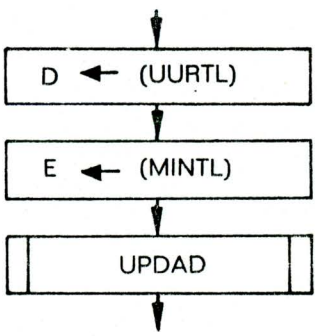


fig.15

Blok ⑤ en ⑥

Deze twee blokken moeten ervoor zorgen, dat display 1 wordt gedoofd, als hierop t.g.v. "UPDAD" een 0 wordt weergegeven.
Hiertoe moeten we instructies opnemen in ons programma.

 Antw.18: UPDAD; D; E.

Vraag 19: Eerst moet het ..... register van de 8279 worden gevuld.
Daarna moet het ..... register worden gevuld.

We moeten eerst het command register (CMREG) van de 8279 vullen met het commando "write display" (WRDPL). Daarna moet het dataregister (DTREG) worden gevuld met een waarde (DOOF), die het geadresseerde display doet doven.
Blok ⑤ en ⑥ worden daarom vervangen door de blokken van fig.16.
De hierin gebruikte namen worden bij het schrijven van het programma omgezet in waarden en adressen.

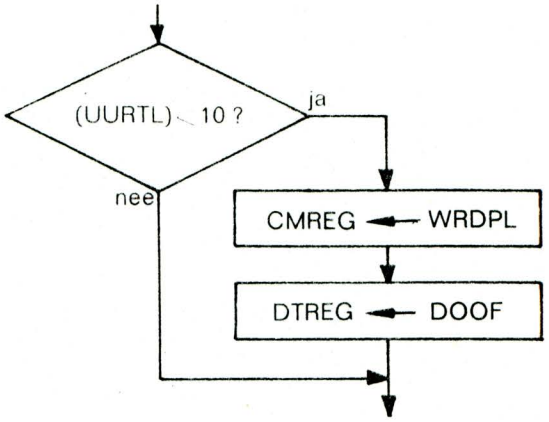


fig.16

Blok ⑦ en ⑧

Blok ⑦ is weer te realiseren door het aanroepen van "UPDAD".

Vraag 20: De inhoud van SECTL moet zich daartoe in register .....
bevinden.

De inhoud van SECTL moet daarbij in register E staan, zodat deze op de displays 3 en 4 wordt weergegeven.
Op de displays 1 en 2 verschijnt de inhoud van register D.
Deze displays moeten worden gedoofd. Dit gebeurt weer door eerst het command register van de 8279 te vullen met het commando "write display" (WRDPL).
Als we zorgen dat de auto increment bit 1 is, dan kunnen we de displays 1 en 2 doven, door tweemaal de waarde DOOF naar het dataregister te sturen.
De blokken ⑦ en ⑧ worden dan vervangen door de blokken van fig.17.

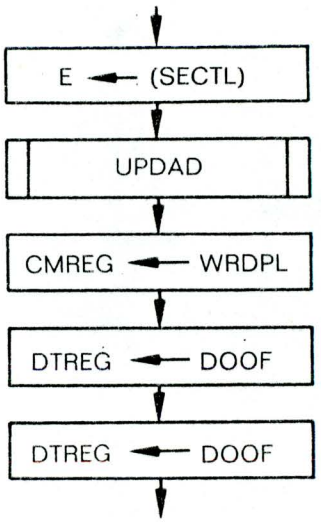


fig.17

Blok ⑨

Voordat $K\overline{W}$ kan worden getest, moet eerst de input-poort, waarop dit signaal wordt aangeboden, worden ingelezen.
We gaan ervan uit, dat $K\overline{W}$ op dezelfde input-poort is aangesloten als SEC. Blok ⑨ wordt dan vervangen door de twee blokken van fig.18.

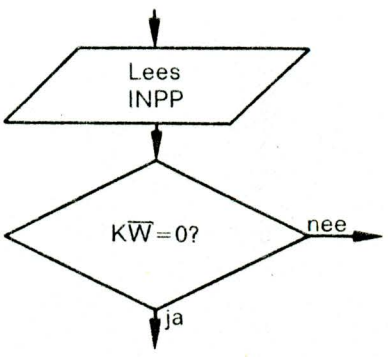


fig.18

Antw.19: command; data. Antw.20: E.

Blok (10), (11) en (12)

Het vergelijken van twee getallen levert geen problemen op.
Deze blokken kunnen ongewijzigd in het gedetailleerd stroomdiagram worden opgenomen.

Blok (13) en (14)

Afhankelijk van de testresultaten van de voorgaande blokken, moet er een 1 of een 0 op de uitgang WEK worden geplaatst.
We noemen de output-poort, waarop WEK is aangesloten, voorlopig OUTPP.

Vraag 21: Een output-operatie verloopt altijd via .....

We moeten daarom eerst de accumulator vullen met de juiste waarde (resp. WEK1 en WEK0).
Deze waarde wordt daarna via OUTPP uitgevoerd.
De blokken (13) en (14) worden dan vervangen door die van fig.19.

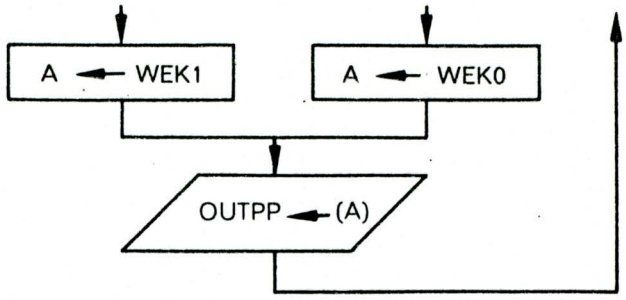


fig.19

Blok (15) t/m (23)

De blokken (15) t/m (23) vormen de interrupt service routine "TMINT". Deze interrupt service routine wordt aangeroepen na het accepteren van een interrupt request van de timer. De blokken zelf behoeven niet verder te worden onderverdeeld, maar voor wat betreft het correct laten functioneren van de timer moeten nog een aantal uitbreidingen worden aangebracht.

Bij de initialisatie moet de timer worden gestart.

Vraag 22: Hiertoe moet eerst het ..... register worden gevuld.

We moeten dan eerst het count length register vullen met een waarde, die de timer gedurende 1 seconde laat tellen.

Vraag 23: Het count length register bestaat uit 8/14/16 bits.

Het count length register bevat 16 bits. Hiervan zijn er twee (b_{15} en b_{14}) voor de timer mode gereserveerd.

Het count length register moet dus met 2 bytes worden gevuld. Deze noemen we voorlopig TIHI (= timer high order byte) en TILO (= timer low order byte). Daarna moet de timer voor de eerste keer worden gestart.
Aan de initialisatie (blok (1)) moeten dan de blokken van fig.20 worden toegevoegd.

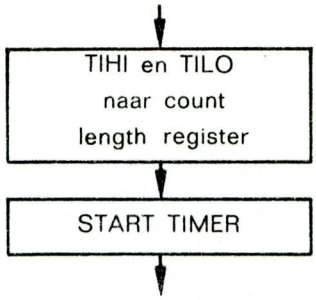


fig 20

In de interrupt service routine moet na elke interrupt request de timer worden gestopt en weer worden gestart. In de voorgaande lessen, waarin de timer werd toegepast, hebben we steeds de timer aan het eind van de interrupt service routine gestart (fig.21).

Vraag 24: De uitvoering van de blokken (15) t/m (23) neemt wel/niet altijd dezelfde tijd in beslag.

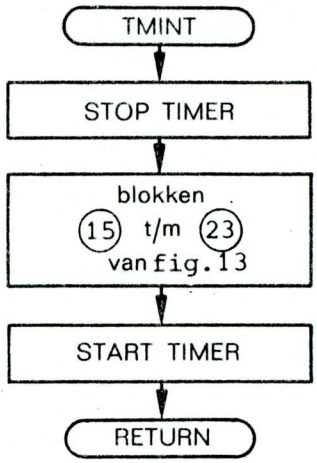


fig.21

Hierbij treedt het probleem op, dat de tijd, die nodig is voor de uitvoering van de blokken (15) t/m (23) sterk kan variëren. Immers in blok (16) kan al direct naar de RETURN-opdracht worden gesprongen.
Dit houdt in, dat bij een oplossing volgens fig.21 de interrupt requests van de timer niet altijd exact 1 seconde uit elkaar liggen. Dit is te verhelpen, door de timer direct na het stoppen weer opnieuw te starten (fig.22).

Er doet zich echter nog een probleem voor bij het gebruik van de timer in de SDK 85.

Vraag 25: De maximale tijd, die met de timer is te realiseren, is ca. 1 ms/2,5 ms/1 s.

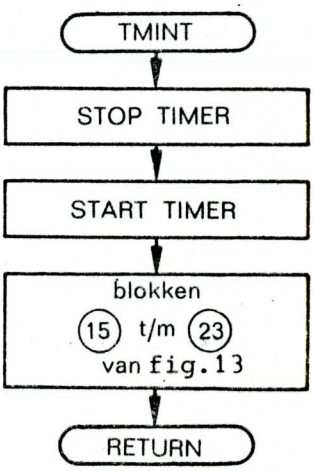


fig.22

Met de timer uit de expansion RAM-I/O chip 8155 is een maximale tijd van ca, 2,5 ms te realiseren.

Antw.24: niet. Antw.25: 2,5 ms.

Vraag 26: SECTL mag pas met 1 worden verhoogd als er ..... maal interrupt request van de timer is geaccepteerd.

Als we de timer exact elke 2,5 ms een interrupt request af laten geven, dan mag SECTL pas na 400 geaccepteerde interrupt requests (400 x 2,5 ms = 1000 ms = 1 s) met 1 worden verhoogd.

Een oplossing hiervoor is in fig.23 weergegeven.
Er zijn twee tellers toegevoegd.

In IRTL wordt het aantal geaccepteerde interrupt requests geteld. Wanneer dit er vier zijn geweest, wordt IRTL met 0 gevuld en wordt HONTL met 1 verhoogd.

Vraag 27: HONTL wordt 1 maal per 1/10/100 ms verhoogd.

Na 4 geaccepteerde interrupt requests wordt HONTL met 1 verhoogd.
Dit is na 4 x 2,5 ms = 10 ms = 0,01 s.
In HONTL worden dus de honderdsten van seconden geteld.

Om na het starten van het programma en na het gelijkzetten van de klok dit tellen van de interrupt requests goed te laten verlopen, moeten in de initialisatie en in de interrupt service routine "GELYK" de tellers IRTL en HONTL met 0 worden gevuld.

Blok (24)

In dit blok moet TBBUF met 0 worden gevuld.
Aangezien we nog niet weten welke registers of geheugenwoorden we voor TBBUF reserveren, kunnen we dit blok voorlopig onveranderd laten.

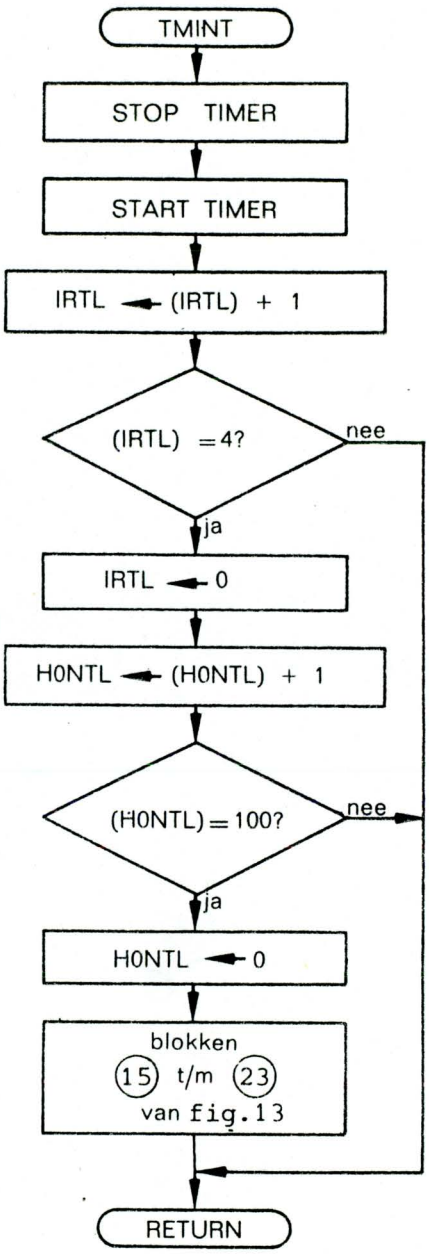


fig.23

 Antw.26: 400. Antw.27: 10.

Blok (25)

In het monitor-programma van de SDK 85 bevindt zich een subroutine, die de accumulator vult met een binaire code voor een ingedrukte toets.

Vraag 28: Dit is de monitor-subroutine ..RDKBD..

In tabel 1 zijn de combinaties vermeld, die "RDKBD" in de accumulator plaatst, na het indrukken van de toetsen, die we bij de digitale klok gebruiken. Voor de toets "START KLOK" nemen we b.v. de GO-toets. (Dit had echter ook een van de overige commando toetsen of een van de toetsen A t/m F mogen zijn.)

toets	(A) (hexadec)
0	00
1	01
2	02
3	03
4	04
5	05
6	06
7	07
8	08
9	09
GO	12

Tabel 1

Blok (25) wordt dan vervangen door de twee blokken uit fig.24.

Blok (26)

Uit tabel 1 blijkt, dat door "RDKBD" direct een ingedrukte decimale toets in BCD-code wordt aangegeven. Blok (26) kan dus voor wat betreft de SDK 85 komen te vervallen.

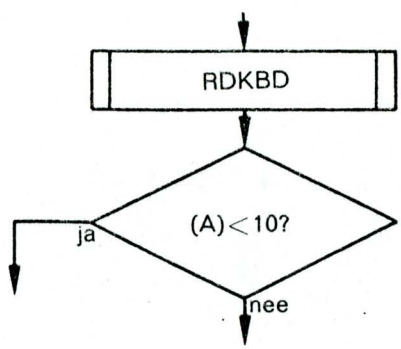


fig.24

Blok (27) en (28)

In blok (27) wordt de inhoud van TBBUF 1 cijferplaats naar links geschoven.

Vraag 29: Een BCD-cijfer bestaat uit ...4... bits.

Dit houdt in, dat de inhoud van TBBUF 4 maal moet worden geschoven. Daarna kan het BCD-cijfer, dat zich achteraan in de accumulator bevindt, achteraan in TBBUF worden geplaatst door de inhoud van de accumulator hierbij op te tellen (b_7 t/m b_4 van de accumulator zijn immers 0). Blok (27) en (28) kunnen dan worden vervangen door de blokken van fig.25.

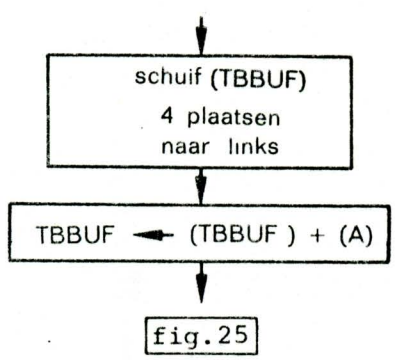


fig.25

Antw.28: RDKBD. Antw.29: 4.

Blok (29)

Vraag 30: We maken weer gebruik van de monitor-subroutine ".....". De inhoud van TBBUF moet dan in registerpaar ..... staan.

Voor het weergeven van de inhoud van TBBUF op de displays roepen we "UPDAD" weer aan.
Een voorwaarde is dan, dat de inhoud van TBBUF eerst naar registerpaar D,E wordt overgebracht.
Blok (29) wordt dan vervangen door de blokken van fig.26.

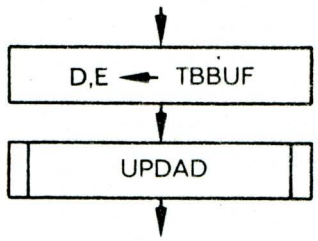


fig.26

Blok (30)

In blok (30) moet worden getest of de toets "START KLOK" is ingedrukt.
Aangezien we hiervoor de GO-toets gebruiken, moet er worden getest of de inhoud van de accumulator gelijk is aan 12_{16}.
Blok (30) wordt daarom vervangen door het testblok van fig.27.

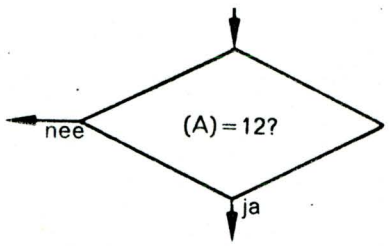


fig.27

Blok (31)

Om dezelfde reden als bij blok (9) is gedaan, moet blok (31) worden vervangen door twee blokken (fig.28).

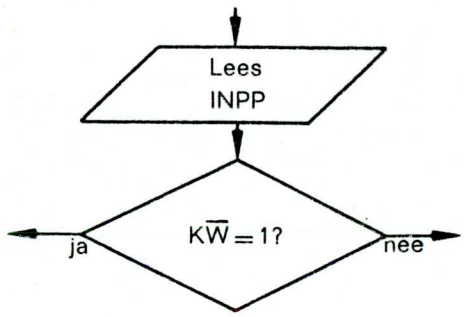


fig.28

Blok (32), (33) en (34)

De blokken (32) en (34) kunnen direct worden overgenomen in het gedetailleerd stroomdiagram.

Vraag 31: Behalve SECTL moeten ook ..... en ..... met 0 worden gevuld.

Blok (33) moet worden uitgebreid.
T.b.v. de interrupt service routine "TMINT" moeten nl. ook IRTL en HONTL met 0 worden gevuld.
Blok (33) gaat dan over in de blokken van fig.29.

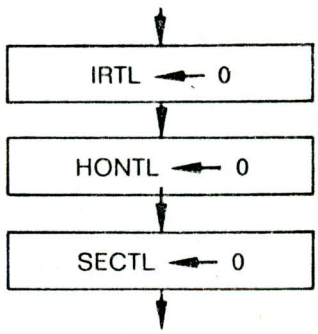
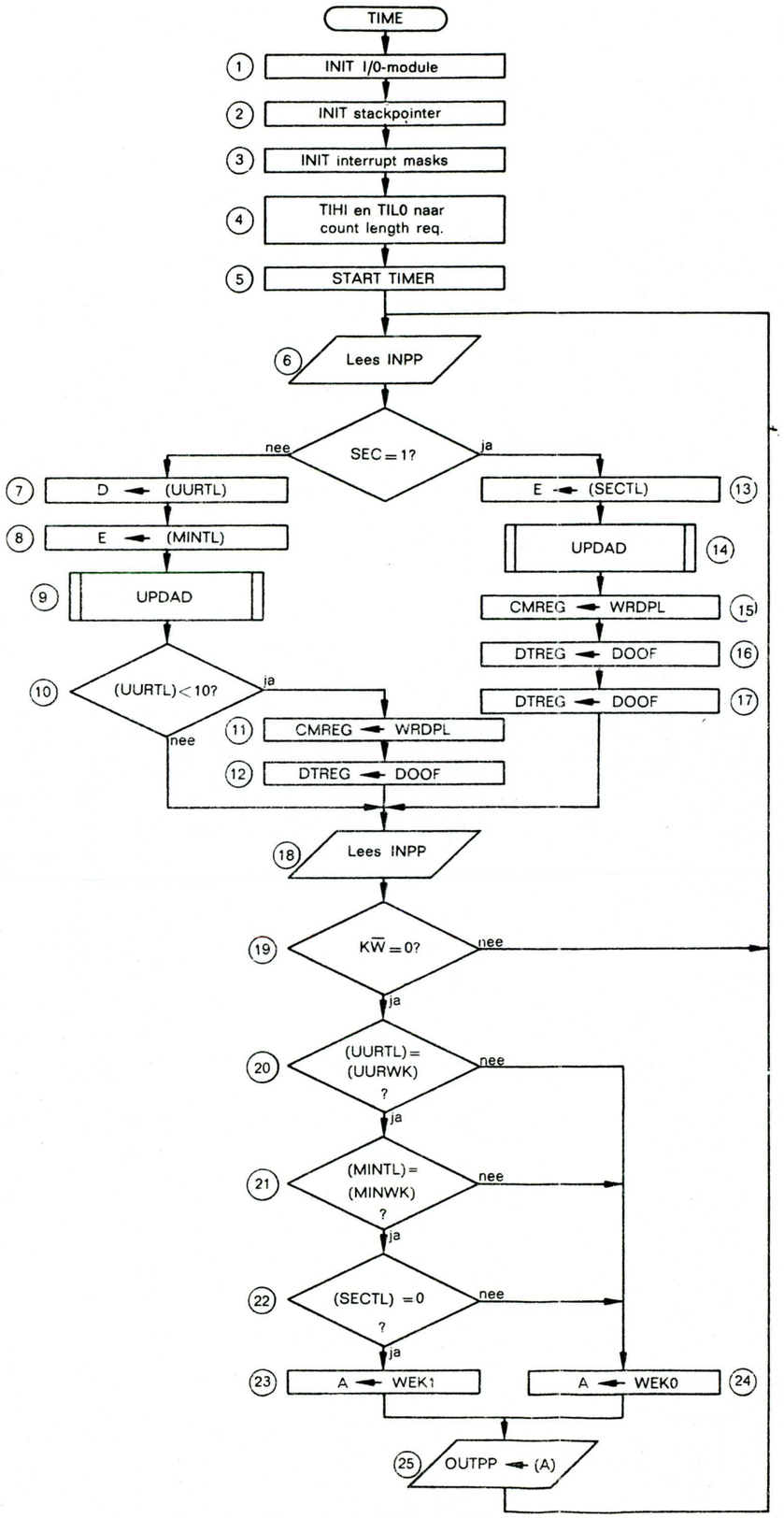


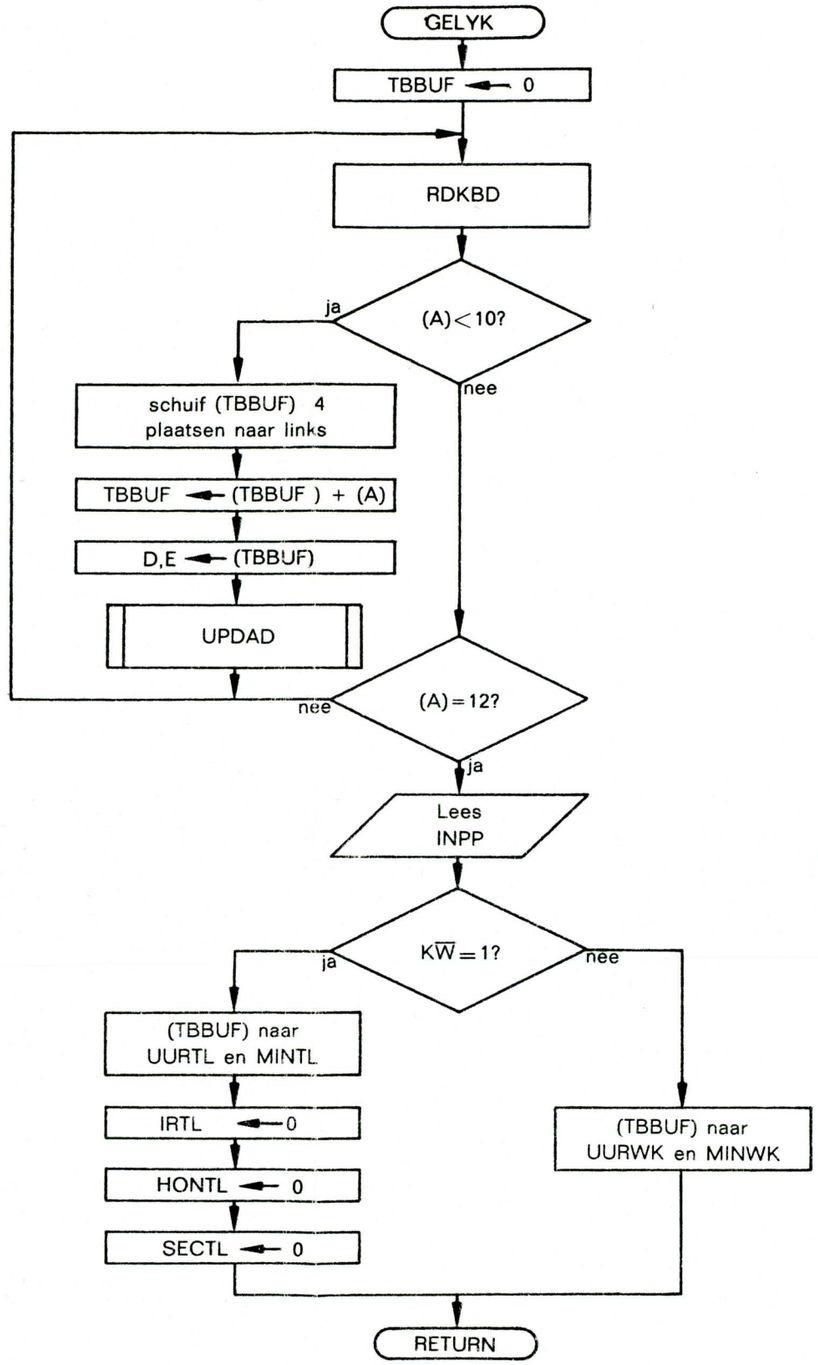
fig. 29

Als we nu alle blokken uit de figuren 14 t/m 29 samenvoegen, dan ontstaat het gedetailleerd stroomdiagram van fig.30.

Uit dit gedetailleerd stroomdiagram is nu een programma te schrijven, mits de symbolische namen worden vervangen door de bijbehorende waarden, resp. adressen.

PROBEER NU ZELF HET PROGRAMMA TE SCHRIJVEN EN TEST DIT OP DE SDK 85.
BESTUDEER DAARNA ONS PROGRAMMA.


TIME
1 INIT I/0-module
2 INIT stackpointer
3 INIT interrupt masks
4 TIHI en TIL0 naar count length req.
5 START TIMER
6 Lees INPP
SEC=1?
nee
ja
7 D ← (UURTL)
8 E ← (MINTL)
9 UPDAD
10 (UURTL)<10?
11 CMREG ← WRDPL
12 DTREG ← DOOF
13 E ← (SECTL)
14 UPDAD
15 CMREG ← WRDPL
16 DTREG ← DOOF
17 DTREG ← DOOF
18 Lees INPP
19 KW=0?
20 (UURTL)= (UURWK) ?
21 (MINTL)= (MINWK) ?
22 (SECTL) =0 ?
23 A ← WEK1
24 A ← WEK0
25 OUTPP ← (A)


GELYK
TBBUF ← 0
RDKBD
(A)<10?
ja
nee
schuif (TBBUF) 4 plaatsen naar links
TBBUF ← (TBBUF) + (A)
D,E ← (TBBUF)
UPDAD
(A)=12?
Lees INPP
KW=1?
(TBBUF) naar UURTL en MINTL
IRTL ← 0
HONTL ← 0
SECTL ← 0
(TBBUF) naar UURWK en MINWK
RETURN

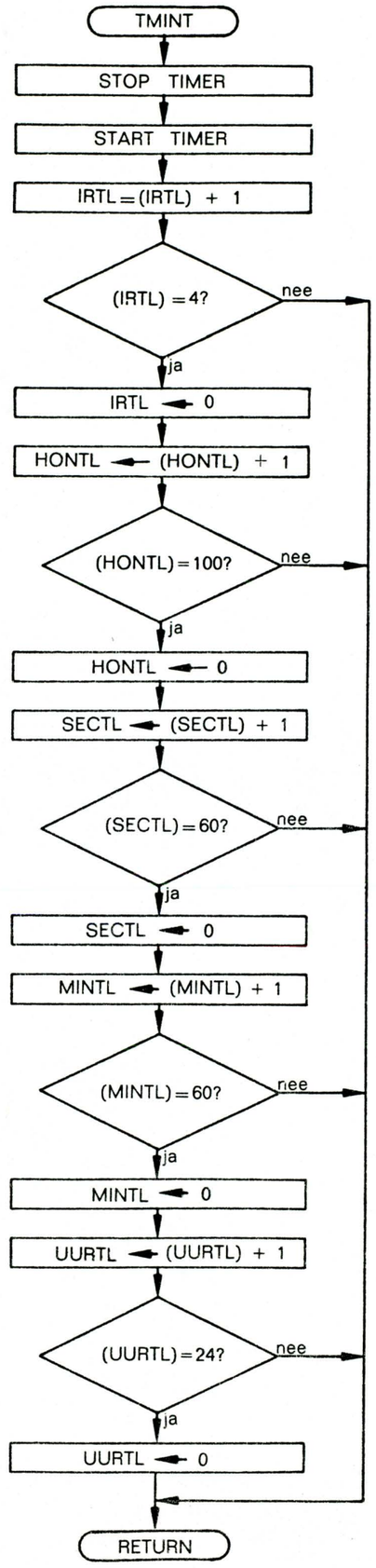

TMINT
STOP TIMER
START TIMER
IRTL=(IRTL) + 1
(IRTL) =4?
nee
ja
IRTL ← 0
HONTL ← (HONTL) + 1
(HONTL)=100?
HONTL ← 0
SECTL ← (SECTL) + 1
(SECTL)=60?
SECTL ← 0
MINTL ← (MINTL) + 1
(MINTL)=60?
MINTL ← 0
UURTL ← (UURTL) + 1
(UURTL)=24?
UURTL ← 0
RETURN

fig. 30

6. PROGRAMMA

Voordat fig.30 in een programma kan worden omgezet, moeten nog een aantal zaken worden uitgezocht, nl.
a. De interrupt-afhandeling moet worden geregeld.
b. De symbolische namen moeten worden vervangen door waarden resp. adressen.
c. De beginadressen van "TIME", "TMINT" en "GELYK" moeten worden vastgesteld.

a. Interrupt-afhandeling

In deze digitale klok is sprake van twee interrupt service routines.

Vraag 32: "TMINT" behoort bij een interrupt request van .....
"GELYK" behoort bij een interrupt request van de toets .....

"TMINT" is de interrupt service routine, die moet worden uitgevoerd na het ontvangen van een interrupt request van de timer uit de expansion RAM-I/O-timer chip.
"GELYK" is de interrupt service routine, die bij de toets "GELIJKZETTEN" behoort.

Vraag 33: De timer geeft de interrupt request RST..... af.
Voor de toets "GELIJKZETTEN" kiezen we de toets "....." van het toetsenbord van de SDK 85. Hierbij behoort de interrupt request RST.....

De timer geeft steeds een interrupt request RST6.5 af.

Omdat het indrukken van de toets "GELIJKZETTEN" een interrupt request moet veroorzaken, kiezen we hiervoor de toets "VECT INTR" van het toetsenbord van de SDK 85. Het indrukken van deze toets veroorzaakt nl. een interrupt request RST7.5.

We moeten dan wel zorgen, dat zich op adres $20CE_{16}$ {$20D4_{16}$} een sprong naar het begin van de interrupt service routine "GELYK" bevindt.
Op adres $20C8_{16}$ {$20CE_{16}$} moet de instructie JMP TMINT staan.

Vraag 34: RST5.5 wordt in het programma voor de digitale klok wel/niet gebruikt.

Behalve RST6.5 en RST7.5 kan RST5.5 ook optreden.
Deze interrupt request wordt nl. gebruikt bij de communicatie tussen de CPU en de keyboard display controller 8279 in de subroutine "RDKBD".

In de initialisatie moet het interrupt mask register zodanig worden gevuld, dat het accepteren van RST5.5, RST6.5 en RST7.5 wordt toegestaan.

Vraag 35: Aan het begin van "TIME" moet wel/niet een EI-instructie worden opgenomen.

Antw.32: de timer; GELIJKZETTEN. Antw.33: 6.5; VECT INTR; 7.5.
Antw.34: wel. Antw.35: wel.

Na het initialiseren van het interrupt mask register moet tevens een EI-instructie worden uitgevoerd, om de INTE FF in de interrupt controller te setten.

Omdat we met 3 interrupt requests te maken hebben, moeten we ook het eventueel nesten van interrupts regelen.

Vraag 36: "TMINT" mag wel/niet worden onderbroken door het honoreren van RST7.5.
"GELYK" mag wel/niet worden onderbroken door het honoreren van RST6.5.

Een interrupt request van de timer moet altijd meteen worden gehonoreerd, anders zou de klok achter gaan lopen. Om dezelfde reden mag de interrupt service routine van de timer niet worden onderbroken door het honoreren van een andere interrupt request.

Vraag 37: In "TMINT" moet de EI-instructie aan het begin/eind worden geplaatst.
In "GELYK" moet de EI-instructie aan het begin/eind worden geplaatst.

Omdat "TMINT" niet mag worden onderbroken, moet de EI-instructie aan het eind van deze interrupt service routine staan.
In "GELYK" moet de EI-instructie direct aan het begin staan, zodat deze interrupt service routine kan worden onderbroken door een interrupt request van de timer.

In de interrupt service routine "GELYK" wordt de monitor-subroutine "RDKBD" aangeroepen. Tijdens het uitvoeren van deze subroutine moet RST5.5 worden geaccepteerd en gehonoreerd.

Vraag 38: Direct voor de instructie CALL RDKBD moet wel/niet een EI-instructie staan.
Direct na CALL RDKBD moet wel/niet een EI-instructie staan.

Direct voor het aanroepen van "RDKBD" behoeft geen EI-instructie te worden opgenomen. Deze zou nl. al aan het begin van "GELYK" staan. Bovendien bevindt zich een EI-instructie in deze subroutine (zie paragraaf 11e van de les "Interrupt").
Direct na de instructie CALL RDKBD moet wel een EI-instructie worden opgenomen, omdat aan het eind van "RDKBD" een DI-instructie staat.

b. Symbolische namen

In tabel 2 zijn de symbolische namen uit fig.30 vermeld, met de adressen, die we hiervoor hebben gekozen.
U mag hiervoor ook andere adressen kiezen (behalve natuurlijk voor CMREG en DTREG).

Antw.36: niet; wel. Antw.37: eind; begin. Antw.38: niet; wel.

symbolische naam	adres	functie
IRTL	2800_{16}	teller voor interrupt requests
HONTL	2801_{16}	teller voor honderdsten van seconden
SECTL	2802_{16}	secondenteller
MINTL	2803_{16}	minutenteller
UURTL	2804_{16}	urenteller
MINWK	2805_{16}	wektijd minuten
UURWK	2806_{16}	wektijd uren
TBBUF	reg.H,L	toetsenbordbuffer
CMREG	1900_{16}	command register 8279
DTREG	1800_{16}	data register 8279
INPP	00_{16}	input-poort t.b.v. SEC en $K\overline{W}$
OUTPP	21_{16}	output-poort t.b.v. WEK

Tabel 2

We moeten tevens bepalen op welke I/O-lijnen de signalen SEC, $K\overline{W}$ en WEK worden ontvangen resp. worden afgegeven.
We maken b.v. onderstaande keuze (u mag hiervan natuurlijk weer afwijken):
SEC = b_0 van input-poort 00_{16}.
$K\overline{W}$ = b_1 van input-poort 00_{16}.
WEK = b_0 van output-poort 21_{16} .

Behalve de symbolische adressen uit tabel 2, zijn er ook een aantal symbolische namen, waarvoor nog een definitieve waarde moet worden berekend, nl. WRDPL, DOOF, WEK0 , WEK1, TIHI en TILO.

WRDPL

Dit is de waarde, die naar het command register van de 8279 moet worden gestuurd om naderhand één of twee displays te laten doven.

Vraag 39: WRDPL moet worden vervangen door $\ldots\ldots_2 = \ldots\ldots\ldots_{16}$.

Het commando "write displays" (zie paragraaf 9 van de les "Systeemeigenschappen hardware") bestaat uit 8 bits.
b_7, b_6 en b_5 vormen de combinatie 100. b_4 (= auto increment) moet 1 zijn, want soms is het noodzakelijk dat achtereenvolgens display 1 en display 2 moeten worden geadresseerd.
b_3 t/m b_0 zijn alle 0. Dit is nl. het adres van display 1. WRDPL moet dus worden vervangen door de waarde $10010000_2 = 90_{16}$.

DOOF

Vraag 40: DOOF komt overeen met de waarde $\ldots\ldots\ldots_{16}$.

Om een segment van een display te laten doven, moeten we een 1 naar de 8279 sturen. Om alle 7-segmenten en de decimale punt van een display te doven, moeten we $11111111_2 = FF_{16}$ in het data-register van de 8279 plaatsen (zie voorbeeld 8 van de les "Systeemeigenschappen hardware").

 Antw.39: 10010000; 90. Antw.40: FF.

WEK0, WEK1

Vraag 41: WEK0 is die waarde, waarbij bit ..... van de accumulator 0 is.

WEK is aangesloten op b_0 van output-poort 21_{16}. D.m.v. OUT-instructies wordt vanuit de accumulator een 0 of een 1 op deze lijn geplaatst. Door WEK0 in de accumulator te zetten, moet deze lijn 0 worden. Voor WEK0 kiezen we b.v. de waarde 00_{16}. Voor WEK1 kiezen we dan 01_{16}. De toestanden van b_7 t/m b_1 doen er in feite niet toe. De overeenkomstige lijnen van output-poort 21_{16} worden nl. niet gebruikt.

TIHI, TILO

TIHI en TILO vormen samen een 16-bits waarde, waarmee het count length register tijdens de initialisatie wordt gevuld. b_{13} t/m b_1 van deze waarde wordt bij het starten van de timer in de down counter geplaatst, die dan omlaag gaat tellen.

Fig.31 is het timing diagram voor het accepteren en honoreren van de interrupt request RST6.5, zoals dit door de interrupt service routine "TMINT" van fig.30 gebeurt.

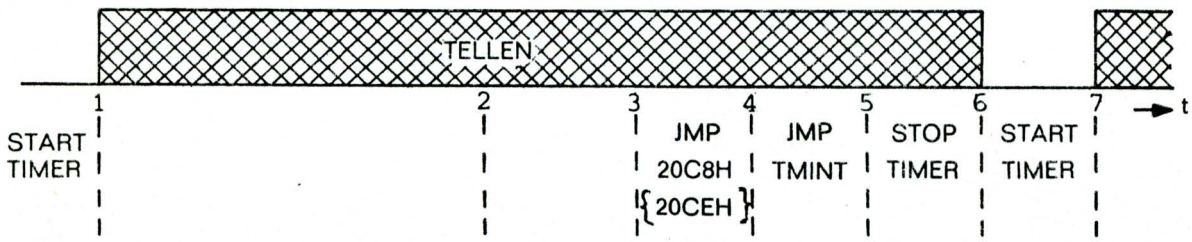


fig. 31

Op t = 1 wordt de timer gestart. De down counter gaat omlaag tellen.

Op t = 2 is de inhoud van de down counter tot de helft van de beginwaarde gedaald. Er verschijnt dan een 1 op de RST6.5-ingang van de CPU. Deze interrupt request wordt wel geaccepteerd (opgeslagen in het interrupt request register), maar niet direct gehonoreerd. De instructie, waarmee de CPU bezig was toen de interrupt request werd ontvangen, moet eerst helemaal worden uitgevoerd.

Op t = 3 is deze instructie uitgevoerd. RST6.5 wordt nu gehonoreerd, d.w.z. dat de programmateller wordt gevuld met het bijbehorende restartadres 0034_{16}. Hier bevindt zich de sprongopdracht JMP 20C8H {20CEH}.

Op t = 4 is deze sprongopdracht uitgevoerd, zodat nu met de instructie JMP TMINT wordt verdergegaan.

Op t = 5 is ook deze sprongopdracht uitgevoerd en kan aan de interrupt service routine worden begonnen. De eerste instructies laten de timer stoppen.

Antw.41: b_0.

Op t = 6 zijn deze instructies uitgevoerd. De timer stopt dan en het signaal op de RST6.5-ingang wordt laag (de interrupt request is dus nu pas weggenomen). Nu volgen de instructies, die de timer weer starten.

Op t = 7 is de timer gestart. Er zal pas weer een interrupt request RST6.5 optreden, als de timer de helft van de beginwaarde heeft bereikt.

Vraag 42: De tijd tussen t = 1 en t = 7 moet exact ..... ms zijn.

Op t = 7 begint dezelfde cyclus als op t = 1. De tijd die tussen t = 1 en t = 7 ligt, moet exact 2,5 ms duren.

Deze 2,5 ms is in drie delen te splitsen, nl.:
- van t = 1 tot t = 2: dit is de tijd tussen het starten van de timer en het accepteren van de interrupt request. Deze tijd is direct afhankelijk van de waarde (TIHI en TILO), die zich in het count length register bevindt.
- van t = 2 tot t = 3: dit is de tijd tussen het accepteren en het honoreren van de interrupt request.
- van t = 3 tot t = 7: dit is de tijd tussen het honoreren van de interrupt request en het opnieuw starten van de timer.

Vraag 43: Tussen t = 3 en t = 7 ligt een tijd van ..... µs.
(1 state = 1/3 µs.)

Tussen t = 3 en t = 7 worden de volgende instructies uitgevoerd.

```
JMP 20C8H {20CEH} = 10 states
JMP TMINT         = 10 states
MVI A,40H         =  7 states
OUT 28H           = 10 states
MVI A,C0H         =  7 states
OUT 28H           = 10 states
```

Deze instructies nemen samen 54 states = 18 µs in beslag.

Vraag 44: De tijd tussen t = 2 en t = 3 is wel/niet exact te bepalen.

De tijd tussen t = 2 en t = 3 is niet exact te bepalen. Deze tijd is nl. afhankelijk van het aantal states, dat nodig is voor het afmaken van de instructie, tijdens welke de interrupt request is ontvangen. Als gemiddelde kunnen we aannemen, dat de interrupt requests precies midden in een instructie worden ontvangen. Om te kunnen bepalen hoeveel states tussen t = 2 en t = 3 liggen, moeten we dan wel weten hoelang de instructie duurt. Aangezien de instructies niet alle evenveel states in beslag nemen, moeten we hiervoor ook weer een gemiddelde instructieduur bepalen.

Vraag 45: De instructies uit de instructieset van de 8085, die de kortste tijd in beslag nemen, duren ..... states.
De langste nemen ..... states in beslag.

```
                        ORG     2000H
                IRTL    EQU     2800H
                HONTL   EQU     2801H
                SECTL   EQU     2802H
                MINTL   EQU     2803H
                UURTL   EQU     2804H
                MINWK   EQU     2805H
                UURWK   EQU     2806H
                CMREG   EQU     1900H
                DTREG   EQU     1800H
                INPP    EQU     00H
                OUTPP   EQU     21H
                WRDPL   EQU     90H
                DOOF    EQU     FFH
                WEK0    EQU     00H
                WEK1    EQU     01H
                TIHI    EQU     79H
                TILO    EQU     F4H
                UPDAD   EQU     0363H
                RDKBD   EQU     02E7H
2000  3E 03     TIME:   MVI     A,03H     ;INIT I/O-MODULE
2002  D3 20             OUT     20H       ;POORT 21H = OUTPUT
2004  32 FF 20          STA     20FFH     ;POORT 22H = OUTPUT
2007  AF               XRA     A
2008  D3 02             OUT     02H       ;POORT 00H = INPUT
200A  D3 03             OUT     03H       ;POORT 01H = INPUT
200C  31 C2 20          LXI     SP,20C2H  ;INIT STACKPOINTER
200F  3E 08             MVI     A,08H     ;INIT INT.MASKS
2011  30                SIM
2012  FB                EI
2013  3E 79             MVI     A,TIHI    ;INIT
2015  D3 2D             OUT     2DH       ; COUNT
2017  3E F4             MVI     A,TILO    ;  LENGTH
2019  D3 2C             OUT     2CH       ;  REGISTER
201B  3E C0             MVI     A,C0H     ;START
201D  D3 28             OUT     28H       ; TIMER
201F  DB 00     TIM1:   IN      INPP      ;TEST SEC
2021  0F                RRC               ;ROTATE RIGHT (ARRY
2022  DA 41 20          JC      TIM2      ;SPRING ALS SEC = 1
2025  2A 03 28          LHLD    H,MINTL
2028  EB                XCHG
2029  06 00             MVI     B,00H     ;(UURTL)EN(MINTL)
202B  CD 63 03          CALL    UPDAD     ; NAAR DISPLAYS
202E  3A 04 28          LDA     UURTL
2031  FE 10             CPI     10H       ;(UURTL)<10 ?
2033  D2 54 20          JNC     TIM3      ;SPRING ALS (UURTL)>9
2036  21 00 19          LXI     H,CMREG   ;(UURTL)<10
2039  36 90             MVI     M,WRDPL   ; DOOF
203B  25                DCR     H         ;  DISPLAY 1
203C  36 FF             MVI     M,DOOF
203E  C3 54 20          JMP     TIM3
2041  21 02 28    TIM2: LXI     H,SECTL   ;SEC = 1
2044  5E               MOV     E,M
2045  06 00             MVI     B,00H     ;(SECTL) NAAR
2047  CD 63 03          CALL    UPDAD     ; DISPLAY 3 EN 4
204A  21 00 19          LXI     H,CMREG
204D  36 90             MVI     M,WRDPL   ;DOOF
204F  25                DCR     H         ; DISPLAY
2050  36 FF             MVI     M,DOOF    ;  1 EN 2
2052  36 FF             MVI     M,DOOF
2054  DB 00     TIM3:   IN      INPP      ;TEST KW
2056  0F                RRC
2057  0F                RRC
2058  DA 1F 20          JC      TIM1      ;SPRING ALS KW = 1
205B  11 06 28          LXI     D,UURWK
205E  21 04 28          LXI     H,UURTL
2061  1A                LDAX    D
2062  BE                CMP     M         ;(UURTL) = (UURWK) ?
2063  C2 78 20          JNZ     TIM4      ;SPRING BIJ ONGELIJK
2066  1B                DCX     D         ;(D,E) = MINWK
2067  2B                DCX     H         ;(H,L) = MINTL
2068  1A                LDAX    D
2069  BE                CMP     M         ;(MINTL) = (MINWK) ?
206A  C2 78 20          JNZ     TIM4      ;SPRING BIJ ONGELIJK
206D  2B                DCX     H         ;(H,L) = SECTL
206E  AF                XRA     A
206F  BE                CMP     M         ;(SECTL) = 0 ?
2070  C2 78 20          JNZ     TIM4      ;SPRING BIJ ONGELIJK
2073  3E 01             MVI     A,WEK1    ;WEK = 1
2075  C3 7A 20          JMP     TIM5
2078  3E 00     TIM4:   MVI     A,WEK0    ;WEK = 0
207A  D3 21     TIM5:   OUT     OUTPP
207C  C3 1F 20          JMP     TIM1
```

INTERRUPT REQUEST

```
                        ORG     2810H
2810  F5        TMINT:  PUSH    PSW
2811  E5                PUSH    H
2812  3E 40             MVI     A,40H     ;STOP
2814  D3 28             OUT     28H       ; TIMER
2816  3E C0             MVI     A,C0H     ;START
2818  D3 28             OUT     28H       ; TIMER
281A  21 00 28          LXI     H,IRTL
281D  34                INR     M         ;(IRTL) + 1
281E  3E 04             MVI     A,04H
2820  BE                CMP     M         ;(IRTL) = 4 ?
2821  C2 56 28          JNZ     TERUG     ;SPRING ALS (IRTL)<4
2824  AF                XRA     A
2825  77                MOV     M,A       ;(IRTL) = 0
2826  23                INX     H         ;(H,L) = HONTL
2827  7E                MOV     A,M
2828  C6 01             ADI     01H       ;(HONTL) + 1
282A  27                DAA
282B  77                MOV     M,A
282C  D2 56 28          JNC     TERUG     ;SPRING ALS (HONTL)<100
282F  23                INX     H         ;(H,L) = SECTL
2830  7E                MOV     A,M
2831  C6 01             ADI     01H       ;(SECTL) + 1
2833  27                DAA
2834  77                MOV     M,A
2835  FE 60             CPI     60H       ;(SECTL) = 60 ?
2837  C2 56 28          JNZ     TERUG     ;SPRING ALS (SECTL)<60
283A  AF                XRA     A
283B  77                MOV     M,A       ;(SECTL) = 0
283C  23                INX     H         ;(H,L) = MINTL
283D  7E                MOV     A,M
283E  C6 01             ADI     01H       ;(MINTL) + 1
2840  27                DAA
2841  77                MOV     M,A
2842  FE 60             CPI     60H       ;(MINTL) = 60 ?
2844  C2 56 28          JNZ     TERUG     ;SPRING ALS (MINTL)<60
2847  AF                XRA     A
2848  77                MOV     M,A       ;(MINTL) = 0
2849  23                INX     H         ;(H,L) = UURTL
284A  FE                MOV     A,M
284B  C6 01             ADI     01H       ;(UURTL) + 1
284D  27                DAA
284E  77                MOV     M,A
284F  FE 24             CPI     24H       ;(UURTL) = 24 ?
2851  C2 56 28          JNZ     TERUG     ;SPRING ALS (UURTL)<24
2854  AF                XRA     A
2855  77                MOV     M,A       ;(UURTL) = 0
2856  E1        TERUG:  POP     H
2857  F1                POP     PSW
2858  FB                EI
2859  C9                RET
                        ORG     2880H
2880  FB        GELYK:  EI
2881  F5                PUSH    PSW
2882  C5                PUSH    B
2883  D5                PUSH    D
2884  E5        GLK1:   PUSH    H
2885  21 00 00          LXI     H,0000H   ;(TBBUF) = 0
2888  E5                PUSH    H
2889  CD E7 02          CALL    RDKBD
288C  FB                EI
288D  E1                POP     H
288E  FE 0A             CPI     0AH       ;(A)<10 ?
2890  D2 A4 28          JNC     GLK2      ;SPRING ALS (A)>9
2893  29                DAD     H         ;SCHUIF
2894  29                DAD     H         ; (TBBUF)
2895  29                DAD     H         ;  4 PLAATSEN
2896  29                DAD     H         ;   NAAR LINKS
2897  85                ADD     L
2898  6F                MOV     L,A       ;(A) NAAR TBBUF
2899  E5                PUSH    H
289A  EB                XCHG
289B  06 00             MVI     B,00H     ;(TBBUF) NAAR
289D  CD 63 03          CALL    UPDAD     ; DISPLAYS
28A0  E1                POP     H
28A1  C3 88 28          JMP     GLK1
28A4  FE 12       GLK2:   CPI     12H       ;(A) = 12 ?
28A6  C2 88 28          JNZ     GLK1      ;SPRING ALS (A) ≠ 12
28A9  DB 00             IN      INPP      ;TEST KW
28AB  0F                RRC
28AC  0F                RRC
28AD  D2 C0 28          JNC     GLK3
28B0  22 03 28          SHLD    MINTL     ;(TBBUF) NAAR UURTL,MINTL
28B3  AF                XRA     A
28B4  32 00 28          STA     IRTL      ;(IRTL) = 0
28B7  32 01 28          STA     HONTL     ;(HONTL) = 0
28BA  32 02 28          STA     SECTL     ;(SECTL) = 0
28BD  C3 C3 28          JMP     GLK4
28C0  22 05 28    GLK3:   SHLD    MINWK     ;(TBBUF) NAAR UURWK,MINWK
28C3  E1          GLK4:   POP     H
28C4  D1                  POP     D
28C5  C1                  POP     B
28C6  F1                  POP     PSW
28C7  C9                  RET
                          ORG     20C8H     ;(20CEH)
20C8  C3 10 28            JMP     TMINT
                          ORG     20CEH     ;(20D4H)
20CE  C3 80 28            JMP     GELYK
                          END
```

INT. REG EINDE

fig. 32

Uit de instructieset van de 8085 blijkt dat de instructieduur van 4 tot 18 states kan variëren. Nu is het zo, dat de instructies van 4, 7 en 10 states het vaakst in het programma voorkomen. Als gemiddelde instructieduur nemen we 8 states. Als tijdens het testen blijkt, dat de klok te snel of te langzaam loopt, dan kunnen we deze waarde naderhand nog aanpassen. Voorlopig gaan we er dan vanuit, dat de instructies gemiddeld 8 states duren. Omdat we aangenomen hebben dat de interrupt request halverwege een instructie optreedt, bevinden zich tussen t = 2 en t = 3 gemiddeld 4 states.

Opmerking:

We hebben hierbij geen rekening gehouden met de subroutine "RDKBD". Deze maakt nl. gebruik van RST5.5 (heeft dus een lagere prioriteit dan RST6.5 van de timer).
Tijdens de uitvoering van RDKBD is het honoreren van andere interrupt requests gedurende enkele tientallen µs verboden. Deze tijd zouden we eigenlijk ook tussen t = 2 en t = 3 moeten voegen.
Aangezien een klok hooguit 1 of 2 maal per week behoeft te worden gelijkgezet, wordt "RDKBD" vrijwel niet aangeroepen.
De invloed van "RDKBD" is daarom te verwaarlozen. Bovendien geldt, dat na het uitvoeren van "GELYK", van waaruit "RDKBD" wordt aangeroepen, de klok is gelijkgezet. Het is dus niet belangrijk of tijdens het uitvoeren van "RDKBD" de klok wel of niet gelijk blijft lopen.

Vraag 46: 2,5 ms komt overeen met ..... states.
Tussen t = 1 en t = 2 moeten zich dan ..... states bevinden.

Tussen t = 1 en t = 7 ligt dan 2,5 ms = 2500 µs = 7500 states.
Tussen t = 2 en t = 7 bevinden zich 4 + 54 = 58 states.
Tussen t = 1 en t = 2 moeten dan 7500 - 58 = 7442 states = 7442 klokimpulsen liggen.

Vraag 47: De timer moet dan worden gevuld met de waarde $.....10$.
Er moet dan gelden TIHI = $.....16$ en TILO = $.....16$.

De timer moet worden gevuld met de dubbele waarde, dus met 14884_{10} = 11101000100100_2. Deze waarde komt uit b_{13} t/m b_0 van het count length register. b_{15} en b_{14} moeten de combinatie 01_{16} vormen (zie paragraaf 8d van de les "Systeemeigenschappen hardware").
In de initialisatie moet het count length register dus worden gevuld met $0111101000100100_2 = 7A24_{16}$. Er geldt dan TIHI = $7A_{16}$ en TILO = 24_{16}.

Naast de symbolische adressen uit tabel 2, hebben we in het programma te maken met de symbolische waarden, die we in deze paragraaf hebben bepaald. Deze zijn in tabel 3 weergegeven.

symbolische naam	waarde	functie
WRDPL	90_{16}	commando "write display"
DOOF	FF_{16}	alle segmenten gedoofd
WEK0	00_{16}	WEK = 0
WEK1	01_{16}	WEK = 1
TIHI	$7A_{16}$	high order byte van count length reg.
TILO	24_{16}	low order byte van count length reg.

Tabel 3

Antw.46: 7500; 7442. Antw.47: 14884; 7A; 24.

c. Beginadressen

Als laatste moeten we de beginadressen van "TIME"; "TMINT" en "GELYK" nog vaststellen. Hiervoor zijn in de probleemomschrijving geen eisen gesteld. We kiezen de adressen uit tabel 4. Dit is een willekeurige keuze. U kunt eventueel andere beginadressen aanhouden.

naam	beginadres
TIME	2000_{16}
TMINT	2810_{16}
GELYK	2880_{16}

Tabel 4

d. Programma

Uitgaande van fig.30 kunnen we nu het programma schrijven.
Dit is in fig.32 weergegeven.

De interrupt service routines "TMINT" en "GELYK" beginnen en eindigen met PUSH- resp. POP-instructies.

Omdat de monitor-subroutines "RDKBD" en "UPDAD" registerinhouden aantasten, moeten voor en na de CALL-instructies ook PUSH- en POP-instructies worden opgenomen. Bij het aanroepen van "UPDAD" moet b_0 van register B 0 zijn, om de decimale punt van display 4 te doven.

Aan het begin van de interrupt service routine "TMINT" zijn twee PUSH-instructies opgenomen. Hiermee is bij de berekening van TIHI en TILO geen rekening gehouden.

Vraag 48: Deze twee PUSH-instructies duren samen ..... states.

Elke PUSH-instructie neemt 12 states in beslag. Dit houdt in, dat in fig.31 tussen t = 2 en t = 7 niet 58 maar 82 states liggen.

Vraag 49: De timer moet dan worden gevuld met .....$_{10}$.
Er moet dan gelden TIHI = ...$_{16}$ en TILO = ...$_{16}$.

Tussen het starten van de timer en het ontvangen van de interrupt request moeten zich dan 7500 - 82 = 7418 states = 7418 klokimpulsen bevinden.
De timer moet met de dubbele waarde worden gevuld, dus met 14836_{10} = 11100111110100_2. Het count length register moet dan tijdens de initialisatie worden gevuld met 0111100111110100_2 = $79F4_{16}$. Er moet dus gelden TIHI = 79_{16} en TILO = $F4_{16}$.

7. TESTEN

De enige manier om dit programma te testen, is dit te laten uitvoeren in de computer waarvoor het programma is geschreven. In dit geval is dit de SDK 85.
Plaats daarom het hoofdprogramma, de interrupt service routines en de spronginstructies JMP TMINT en JMP GELYK in het RAM-geheugen.

Start het programma met "GO";2;0;0;0;"EXEC".
Zet de schakelaar aan b_0 van input-poort 00_{16} in de stand 1.
Er geldt nu SEC = 1. Op de displays 3 en 4 moeten de seconden worden weergegeven. Als u het programma goed heeft ingetypt, zal dit ook gebeuren. Maar de displays 1 en 2, die hadden moeten doven, lichten enigszins op. Er is duidelijk te zien, dat deze displays steeds afwisselend oplichten en weer doven. De oorzaak hiervan is in fig.30 terug te vinden.
In blok (14) worden de displays 1 en 2 aangestuurd. Door de blokken (15), (16) en (17) worden ze weer gedoofd.
Aangezien het programma steeds weer opnieuw wordt doorlopen, zullen de displays ook steeds kortstondig worden aangestuurd en oplichten. Hetzelfde geldt natuurlijk voor display 1 als SEC = 0 en het aantal weer te geven uren is kleiner dan 10.

Om deze fout op te heffen, zijn er diverse mogelijkheden.
We kunnen b.v. direct voor blok (18) (zie fig.30) een wachtlus aanbrengen. Displays worden dan minder vaak aangestuurd. De tijd dat de displays gedoofd zijn, is dan relatief groot t.o.v. de tijd dat ze aangestuurd worden. Immers na het doven van de gewenste displays wordt eerst de wachtlus uitgevoerd.

Een tweede oplossing is het zelf ontwikkelen van een subroutine, die de displays die gedoofd moeten blijven, niet eerst aanstuurt en ze daarna pas weer dooft. Deze subroutine komt dan in de plaats van de blokken (6) t/m (17) in fig.30. Dit zijn de instructies op de adressen $201F_{16}$ t/m 2053_{16} in fig.32.

We kiezen voor de tweede oplossing.
Echter voordat we deze subroutine gaan ontwikkelen (zie paragraaf 8), moet er nog een andere fout uit het programma worden gehaald.
Wanneer u nl. de klok wilt gelijkzetten, en dus op de toets "VECT INTR" drukt, verschijnt er op de displays de tekst "-80 85" of de foutmelding "-Err". Er is dan klaarblijkelijk teruggesprongen naar het monitorprogramma van de SDK 85.

Wanneer we het programma opnieuw starten, dan verschijnt de letter E op de displays en verder gebeurt er niets. Ook de LED's van de outputpoorten blijven alle branden.

> Vraag 50: Door de initialisatie van de I/O-poorten moeten deze LED's blijven branden/uitgaan.

Door de instructies op de adressen 2000_{16} t/m $200B_{16}$ worden de I/O-poorten geïnitialiseerd. Daarbij moeten de LED's van de output-poorten doven.

Omdat de LED's in dit geval niet doven, zal er waarschijnlijk een fout in de initialisatie zitten. M.b.v. "SUBST MEM" controleren we dan de inhouden van de betreffende geheugenwoorden. Er blijkt dan, dat de oorspronkelijke inhouden alle verloren zijn gegaan, m.a.w. het programma is vernietigd.

Vraag 51: Dit komt, doordat de inhoud van ..... te ver is gedaald.

Dit kan alleen worden veroorzaakt door de stackpointer. Als de inhoud hiervan nl. te ver daalt, dan zal het programma worden overschreven door data, die naar de stack wordt gebracht.

Vraag 52: Tegenover elke CALL-instructie moet een .....-instructie staan.
Bij elke PUSH-instructie behoort een .....-instructie.

Dat de stackpointer zover wordt verlaagd, kan de volgende oorzaken hebben.
a. Niet elke subroutine of interrupt service routine is correct afgesloten met een RET-instructie.
b. Er staan meer PUSH- dan POP-instructies in het programma.
c. Er wordt te vaak genest.

Controle van het programma wijst niet op een foutief gebruik van CALL-, RET-, PUSH- en POP-instructies. We moeten dus onderzoeken of er te vaak wordt genest.

Vraag 53: Het vernietigen van het programma werd veroorzaakt door het indrukken van de toets ".....".

Pas nadat we op de toets "VECT INTR" drukten, werd het programma vernietigd. Het is dus logisch te veronderstellen, dat de fout in het accepteren en honoreren van de interrupt request RST7.5 ligt.

Direct aan het begin van de interrupt service routine "GELYK" bevindt zich een EI-instructie. Er kunnen dus meteen nieuwe interrupt requests gehonoreerd worden, dus ook RST7.5. Dat er direct een nieuwe RST7.5 wordt geaccepteerd en gehonoreerd, zal blijken uit de volgende beschouwing.

Elke schakelaar bezit een zekere contactdender, d.w.z. dat bij het indrukken en loslaten van de toets het contact een aantal malen open en dicht gaat (dendert).
In fig.33 zijn de gevolgen hiervan weergegeven.

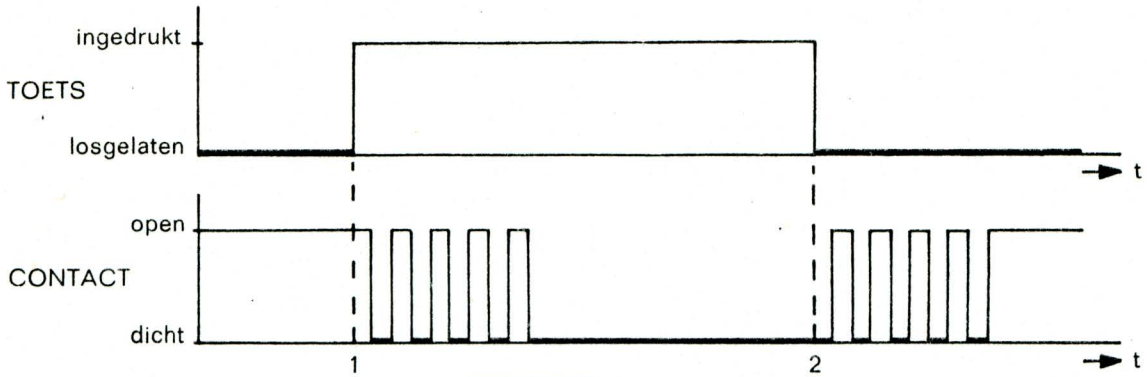


fig. 33

Op t = 1 wordt de toets ingedrukt. Voordat het contact definief wordt gesloten, zal het enkele keren denderen.
Hetzelfde gebeurt als op t = 2 de toets wordt losgelaten.
Het gevolg is dat er i.p.v. één impuls een aantal impulsen ontstaan.

Omdat de "VECT INTR"-toets direct op de RST7.5-ingang van de 8085 is aangesloten, zal het indrukken van "VECT INTR" in een aantal interrupt requests resulteren. Door de EI-instructie aan het begin van de interrupt service routine zullen al deze interrupt requests niet alleen geaccepteerd maar ook gehonoreerd worden.

Vraag 54: Deze EI-instructie mag wel/niet worden weggelaten.

Deze EI-instructie is noodzakelijk om de interrupt request van de timer toe te staan. We moeten dus zorgen dat RST6.5 wel en RST7.5 niet wordt toegestaan.

Vraag 55: Dit kunnen we bereiken door de inhoud van het .....-register te wijzigen.
Dit register moet worden gevuld met $\ldots\ldots_{16}$.

We moeten dus de inhoud van het interrupt mask register wijzigen. RST6.5 en RST5.5, afkomstig van de timer, resp. subroutine "RDKBD", moeten worden toegestaan. b_0 en b_1 van het interrupt mask register moeten dus 0 worden. RST7.5 moet worden verboden, dus b_4 en b_2 moeten 1 zijn. b_3 moet 1 zijn, om b_2, b_1 en b_0 te kunnen wijzigen.
Het interrupt mask register moet dus worden gevuld met $00011100_2 = 1C_{16}$.
Aan het begin van "GELYK" moeten dus de volgende instructies worden opgenomen:

```
MVI A,1CH
SIM
```

Door deze instructies wordt het honoreren van interrupt request RST7.5 verboden. Echter het accepteren van RST7.5 gaat gewoon door. Ontvangen interrupt requests worden nl. wel in het interrupt request register opgeslagen.

Antw.54: niet. Antw.55: interrupt mask; 1C.

Vraag 56: Als we aan het eind van "GELYK" het interrupt mask register weer met 08_{16} vullen, dan wordt er wel/niet direct een RST7.5 gehonoreerd.

Dit houdt in, dat gedurende de uitvoering van "GELYK" een interrupt request RST7.5 in het interrupt register is opgeslagen. Immers, na het honoreren van de eerste interrupt request worden door de contactdender nog een aantal interrupt requests ontvangen.

Als we nu aan het eind van "GELYK" het interrupt mask register met 08_{16} vullen, om alle interrupt requests weer toe te staan, dan is er dus direct een geaccepteerde RST7.5 in het interrupt request register aanwezig, die meteen gehonoreerd zal worden. We moeten dus zorgen, dat deze interrupt request wordt gereset, voordat het interrupt mask register met 09_{16} wordt gevuld (om ook RST5.5 te verbieden, maken we b_0 1).

Dit resetten van RST7.5 doen we door het interrupt mask register nogmaals met $1C_{16}$ te vullen. (Wanneer we nl. d.m.v. de SIM-instructie een of meer interrupt requests verbieden, dan worden de in het interrupt mask register aanwezige, maar nu verboden interrupt request(s) gereset).
We moeten aan het eind van "GELYK" het interrupt mask register dus eerst met $1C_{16}$ en dan met 09_{16} vullen.
"GELYK" moet dus worden afgesloten met de volgende instructies:

```
MVI A,1CH
SIM
MVI A,09H
SIM
RET
```

De interrupt service routine "GELYK" wordt dan zoals in fig.34 is weergegeven.

adres	hexadecimaal	label	mnem.	operand	commentaar
			ORG	287D	
287D	F5	GELYK:	PUSH	PSW	
287E	3E 1C		MVI	A,1CH	;VERBIEDT
2880	30		SIM		; RST7.5
2881	FB		EI		
2882	C5		PUSH	B	
.			.		
.			.		
.			.		
.			.		
28C5	C1		POP	B	
28C6	3E 1C		MVI	A,1CH	;RESET
28C8	30		SIM		; RST7.5
28C9	3E 09		MVI	A,09H	
28CB	30		SIM		
28CC	F1		POP	PSW	
28CD	C9		RET		

fig.34

23-10 Antw.56: wel.

Breng nu in uw programma de wijzigingen van fig.34 aan.
Merk op, dat het beginadres van "GELYK" is veranderd. Dit is gedaan om de extra instructies tussen te voegen.
We behoeven nu nl. niet de gehele interrupt service routine opnieuw in te voeren. De instructies op de adressen 2882_{16} t/m $28C5_{16}$ blijven ongewijzigd. We moeten dan wel de instructie JMP GELYK op adres 20CE $\{20D4_{16}\}$ wijzigen.

Als we nu het hoofdprogramma "TIME" weer hebben ingetypt (dit was immers verloren gegaan), dan kunnen we dit testen. Er blijkt dan, dat het programma correct werkt, afgezien van het aansturen van de displays.
Hiervoor moeten we dus nog een oplossing vinden.
Deze is in de volgende paragraaf beschreven.

```
                        ORG     2000H
                IRTL    EQU     2800H
                HONTL   EQU     2801H
                SECTL   EQU     2802H
                MINTL   EQU     2803H
                UURTL   EQU     2804H
                MINWK   EQU     2805H
                UURWK   EQU     2806H
                INPP    EQU     00H
                OUTPP   EQU     21H
                DOOF    EQU     15H
                WEK0    EQU     00H
                WEK1    EQU     01H
                TIHI    EQU     79H
                TILO    EQU     F4H
                UPDAD   EQU     0363H
                RDKBD   EQU     02E7H
                OUTPT   EQU     02B7H
2000  3E 03     TIME:   MVI     A,03H   ;INIT I/O-MODULE
2002  D3 20             OUT     20H     ;POORT 21H = OUTPUT
2004  32 FF 20          STA     20FFH   ;POORT 22H = OUTPUT
2007  AF               XRA     A
2008  D3 02             OUT     02H     ;POORT 00H = INPUT
200A  D3 03             OUT     03H     ;POORT 01H = INPUT
200C  31 C2 20          LXI     SP,20C2H ;INIT STACKPOINTER
200F  3E 09             MVI     A,09H   ;INIT INT.MASKS
2011  30                SIM
2012  FB                EI
2013  3E 79     7B      MVI     A,TIHI  ;INIT
2015  D3 2D             OUT     2DH     ; COUNT
2017  3E F4     5C      MVI     A,TILO  ;  LENGTH
2019  D3 2C             OUT     2CH     ;    REGISTER
201B  3E C0             MVI     A,C0H   ;START
201D  D3 28             OUT     28H     ; TIMER
201F  21 F0 28    TIM1: LXI     H,28F0H
2022  DB 00             IN      INPP    ;TEST SEC
2024  0F                RRC
2025  DA 53 20          JC      TIM2    ;SPRING ALS SEC = 1
2028  3A 04 28          LDA     UURTL   ;(A) = XXXXYYYY
202B  0F                RRC
202C  0F                RRC
202D  0F                RRC
202E  0F                RRC             ;(A) = YYYYXXXX
202F  E6 0F             ANI     0FH     ;(A) = 0000XXXX
2031  C2 36 20          JNZ     TIM6    ;SPRING ALS (UURTL)>9
2034  3E 15             MVI     A,DOOF  ;DOOF DISPLAY 1
2036  77        TIM6: MOV     M,A
2037  23                INX     H       ;(H,L) = 28F1H
2038  3A 04 28          LDA     UURTL   ;(A) = XXXXYYYY
203B  E6 0F             ANI     0FH     ;(A) = 0000YYYY
203D  77                MOV     M,A
203E  23                INX     H       ;(H,L) = 28F2H
203F  3A 03 28          LDA     MINTL   ;(A) = XXXXYYYY
2042  0F                RRC
2043  0F                RRC
2044  0F                RRC
2045  0F                RRC             ;(A) = YYYYXXXX
2046  E6 0F             ANI     0FH     ;(A) = 0000XXXX
2048  77                MOV     M,A
2049  23                INX     H       ;(H,L) = 28F3H
204A  3A 03 28          LDA     MINTL   ;(A) = XXXXYYYY
204D  E6 0F             ANI     0FH     ;(A) = 0000YYYY
204F  77                MOV     M,A
2050  C3 6A 20          JMP     TIM3
2053  36 15      TIM2: MVI     M,DOOF  ;DOOF DISPLAY 1
2055  23                INX     H       ;(H,L) = 28F1H
2056  36 15             MVI     M,DOOF  ;DOOF DISPLAY 2
2058  23                INX     H       ;(H,L) = 28F2H
2059  3A 02 28          LDA     SECTL   ;(A) = XXXXYYYY
205C  0F                RRC
205D  0F                RRC
205E  0F                RRC
205F  0F                RRC             ;(A) = YYYYXXXX
2060  E6 0F             ANI     0FH     ;(A) = 0000XXXX
2062  77                MOV     M,A
2063  23                INX     H       ;(H,L) = 28F3H
2064  3A 02 28          LDA     SECTL   ;(A) = XXXXYYYY
2067  E6 0F             ANI     0FH     ;(A) = 0000YYYY
2069  77                MOV     M,A
206A  21 F0 28    TIM3: LXI     H,28F0H
206D  AF                XRA     A       ;DISPLAY 1 t/m 4
206E  47                MOV     B,A     ;DEC. PUNT UIT
206F  CD B7 02          CALL    OUTPT
2072  DB 00             IN      INPP    ;TEST KW
2074  0F                RRC
2075  0F                RRC
2076  DA 1F 20          JC      TIM1    ;SPRING ALS KW = 1
2079  11 06 28          LXI     D,UURWK
207C  21 04 28          LXI     H,UURTL
207F  1A                LDAX    D
2080  BE                CMP     M       ;(UURTL) = (UURWK) ?
2081  C2 96 20          JNZ     TIM4    ;SPRING BIJ ONGELIJK
2084  1B                DCX     D       ;(D,E) = MINWK
2085  2B                DCX     H       ;(H,L) = MINTL
2086  1A                LDAX    D
2087  BE                CMP     M       ;(MINTL) = (MINWK) ?
2088  C2 96 20          JNZ     TIM4    ;SPRING BIJ ONGELIJK
208B  2B                DCX     H       ;(H,L) = SECTL
208C  AF                XRA     A
208D  BE                CMP     M       ;(SECTL) = 0 ?
208E  C2 96 20          JNZ     TIM4    ;SPRING BIJ ONGELIJK
2091  3E 01             MVI     A,WEK1  ;WEK = 1
2093  C3 98 20          JMP     TIM5
2096  3E 00     TIM4: MVI     A,WEK0  ;WEK = 0
2098  D3 21      TIM5: OUT     OUTPP
209A  C3 1F 20          JMP     TIM1

                        ORG     2810H
2810  F5        TMINT: PUSH    PSW
2811  E5                PUSH    H
2812  3E 40             MVI     A,40H   ;STOP
2814  D3 28             OUT     28H     ; TIMER
2816  3E C0             MVI     A,C0H   ;START
2818  D3 28             OUT     28H     ; TIMER
281A  21 00 28          LXI     H,IRTL
281D  34                INR     M       ;(IRTL) + 1
281E  3E 04             MVI     A,04H
2820  BE                CMP     M       ;(IRTL) = 4 ?
2821  C2 56 28          JNZ     TERUG   ;SPRING ALS (IRTL)<4
2824  AF                XRA     A
2825  77                MOV     M,A     ;(IRTL) = 0
2826  23                INX     H       ;(H,L) = HONTL
2827  7E                MOV     A,M
2828  C6 01             ADI     01H     ;(HONTL) + 1
282A  27                DAA
282B  77                MOV     M,A
282C  D2 56 28          JNC     TERUG   ;SPRING ALS (HONTL)<100
282F  23                INX     H       ;(H,L) = SECTL
2830  7E                MOV     A,M
2831  C6 01             ADI     01H     ;(SECTL) + 1
2833  27                DAA
2834  77                MOV     M,A
2835  FE 60             CPI     60H     ;(SECTL) = 60 ?
2837  C2 56 28          JNZ     TERUG   ;SPRING ALS (SECTL)<60
283A  AF                XRA     A
283B  77                MOV     M,A     ;(SECTL) = 0
283C  23                INX     H       ;(H,L) = MINTL
283D  7E                MOV     A,M
283E  C6 01             ADI     01H     ;(MINTL) + 1
2840  27                DAA
2841  77                MOV     M,A
2842  FE 60             CPI     60H     ;(MINTL) = 60 ?
2844  C2 56 28          JNZ     TERUG   ;SPRING ALS (MINTL)<60
2847  AF                XRA     A
2848  77                MOV     M,A     ;(MINTL) = 0
2849  23                INX     H       ;(H,L) = UURTL
284A  7E                MOV     A,M
284B  C6 01             ADI     01H     ;(UURTL) + 1
284D  27                DAA
284E  77                MOV     M,A
284F  FE 24             CPI     24H     ;(UURTL) = 24 ?
2851  C2 56 28          JNZ     TERUG   ;SPRING ALS (UURTL)<24
2854  AF                XRA     A
2855  77                MOV     M,A     ;(UURTL) = 0
2856  E1        TERUG: POP     H
2857  F1                POP     PSW
2858  FB                EI
2859  C9                RET
                        ORG     287DH
287D  F5        GELYK: PUSH    PSW
287E  3E 1C             MVI     A,1CH   ;VERBIED
2880  30                SIM             ; RST7.5
2881  FB                EI
2882  C5                PUSH    B
2883  D5                PUSH    D
2884  E5                PUSH    H
2885  21 00 00          LXI     H,0000H ;(TBBUF) = 0
2888  E5        GLK1: PUSH    H
2889  CD E7 02          CALL    RDKBD
288C  FB                EI
288D  E1                POP     H
288E  FE 0A             CPI     0A,H    ;(A)<10 ?
2890  D2 A4 28          JNC     GLK2    ;SPRING ALS (A)>9
2893  29                DAD     H       ;SCHUIF
2894  29                DAD     H       ; (TBBUF)
2895  29                DAD     H       ;  4 PLAATSEN
2896  29                DAD     H       ;   NAAR LINKS
2897  85                ADD     L
2898  6F                MOV     L,A     ;(A) NAAR TBBUF
2899  E5                PUSH    H
289A  EB                XCHG
289B  06 00             MVI     B,00H   ;(TBBUF) NAAR
289D  CD 63 03          CALL    UPDAD   ; DISPLAYS
28A0  E1                POP     H
28A1  C3 88 28          JMP     GLK1
28A4  FE 12       GLK2: CPI     12H     ;(A) = 12 ?
28A6  C2 88 28          JNZ     GLK1    ;SPRING ALS (A) ≠ 12
28A9  DB 00             IN      INPP    ;TEST KW
28AB  0F                RRC
28AC  0F                RRC
28AD  D2 C0 28          JNC     GLK3
28B0  22 03 28          SHLD    MINTL   ;(TBBUF) NAAR UURTL,MINTL
28B3  AF                XRA     A
28B4  32 00 28          STA     IRTL    ;(IRTL) = 0
28B7  32 01 28          STA     HONTL   ;(HONTL) = 0
28BA  32 02 28          STA     SECTL   ;(SECTL) = 0
28BD  C3 C3 28          JMP     GLK4
28C0  22 05 28    GLK3: SHLD    MINWK   ;(TBBUF) NAAR UURWK,MINWK
28C3  E1          GLK4: POP     H
28C4  D1                POP     D
28C5  C1                POP     B
28C6  3E 1C             MVI     A,1CH   ;RESET
28C8  30                SIM             ; RST7.5
28C9  3E 09             MVI     A,09H
28CB  30                SIM
28CC  F1                POP     PSW
28CD  C9                RET
                        ORG     20C8H   ;(20CEH)
20C8  C3 10 28          JMP     TMINT
                        ORG     20CEH   ;(20D4H)
20CE  C3 7D 28          JMP     GELYK
                        END
```

fig. 37

8. DISPLAY-AANSTURING

We moeten een programmadeel ontwikkelen, dat de 7-segment displays op een zodanige wijze aanstuurt, dat de displays, die gedoofd moeten zijn, niet kortstondig oplichten.

Vraag 57: Hiervoor maken we gebruik van de monitor-subroutine "........"

M.b.v. de monitor-subroutine "OUTPT" kunnen we data op de displays weergeven en tegelijkertijd aangeven, welke displays moeten doven.
In tabel 5 zijn de karakters vermeld, die we in ons programma nodig hebben.

Tabel 5 is ontleend aan paragraaf 5d van de les "Systeemeigenschappen software".
Lees deze nog eens nauwkeurig door.

karakter	((H,L))
0	00
1	01
2	02
3	03
4	04
5	05
6	06
7	07
8	08
9	09
geen	15

Tabel 5

Als we de inhoud van UURTL, MINTL of SECTL m.b.v. "OUTPT" willen weergeven, dan moet deze inhoud worden gesplitst in 2 bytes (fig.35).

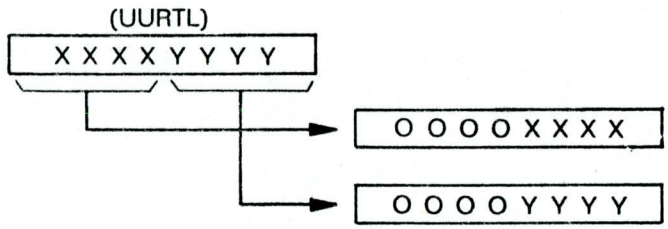


fig. 35

De linker 4 bits moeten 4 plaatsen naar rechts worden geschoven.
Voor de rechter 4 bits moeten 4 nullen worden geplaatst.

Om 4 karakters weer te geven, moeten 4 geheugenwoorden worden gereserveerd. In deze geheugenwoorden moeten wij de gewenste codes uit tabel 5 plaatsen. Voor deze geheugenwoorden kiezen we b.v. de adressen $28F0_{16}$, $28F1_{16}$, $28F2_{16}$ en $28F3_{16}$.

Vraag 58: Voor het aanroepen van "OUTPT" moet H,L worden gevuld met $...._{16}$.

Voordat we "OUTPT" aanroepen, moet H,L worden gevuld met het beginadres van deze 4 geheugenwoorden.

De blokken (6) t/m (17) in fig.30 kunnen dan worden vervangen door die van fig.36.

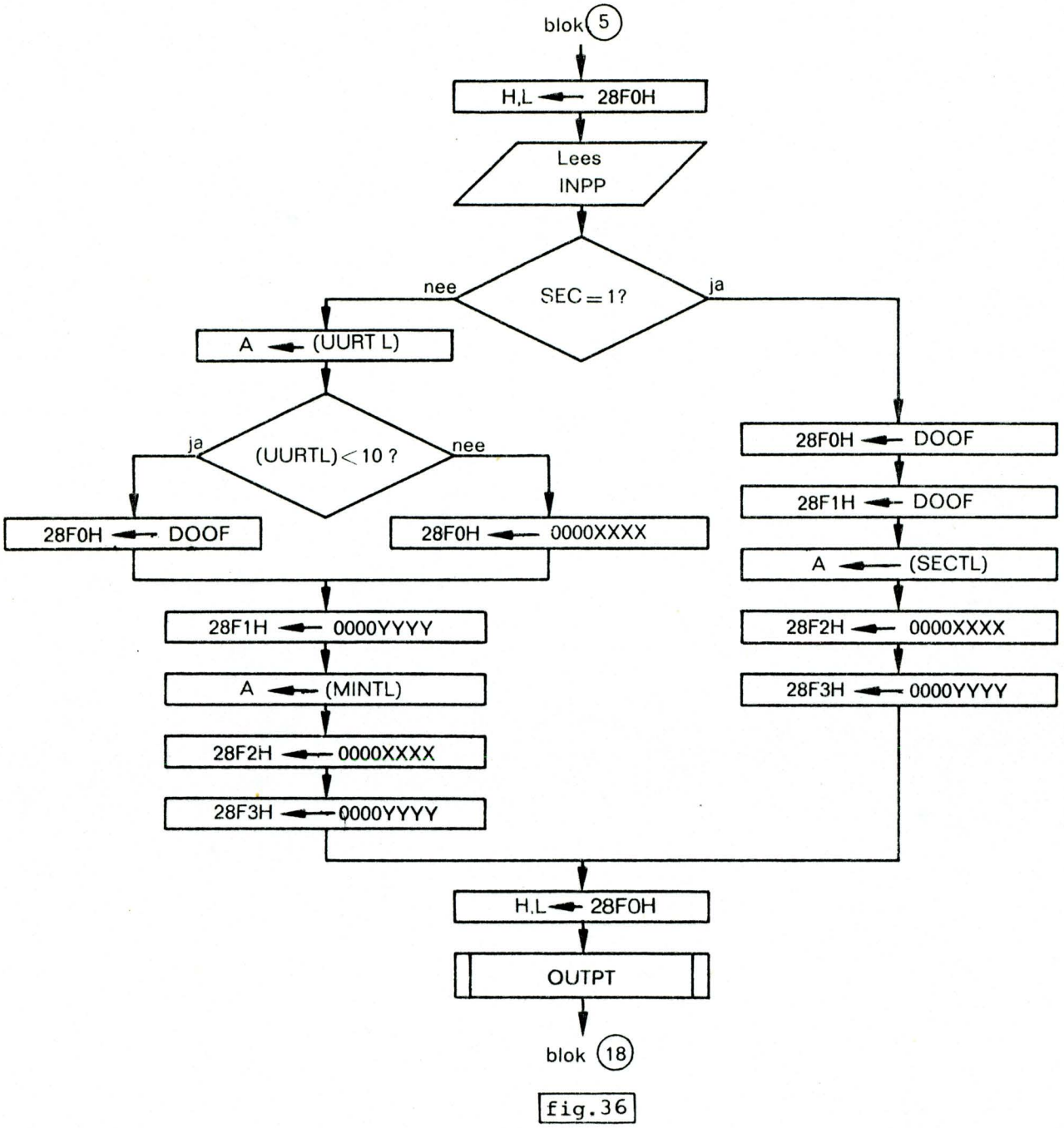


fig.36

Fig.36 moet worden omgezet in een programma. Hierbij moeten we rekening houden met het feit dat DOOF niet meer de waarde FF_{16} heeft, maar 15_{16} (zie tabel 5).

Het totale hoofdprogramma "TIME" is weergegeven in fig.37. Hierin zijn de instructies op de adressen $201F_{16}$ t/m 2071_{16} afkomstig van fig.36.

Test het programma nu op de SDK 85.
Als u alles correct heeft ingetypt, zal de klok correct functioneren.

Zo niet, spoor de fout dan op m.b.v. single step en/of breakpoints.

INTERFACING-1.

Interfacing-1

1. INLEIDING

In de voorgaande lessen is een aantal feiten over de interne structuur (opbouw, organisatie en programmering) van de microcomputer beschreven. Echter met alleen een processor en centraal geheugen zijn we er nog niet. De microcomputer is nl. geen doel op zich, maar is pas zinvol in samenwerking met gebruikersapparatuur. Onder gebruikersapparatuur verstaan we systeemdelen, die door het processorsysteem worden bestuurd en die óf informatie afgeven, die door de buitenwereld kan worden verwerkt, óf de door het systeem afgegeven informatie verder verwerken. Het kunnen apparaten zijn zoals draaibanken, meetapparatuur, computerrandapparatuur (teletype, printer, ponsbandapparaten), regelapparatuur, huishoudelijke apparaten, enz.
Het aansluiten van de genoemde apparaten aan een processorsysteem duiden we aan met de algemene term interfacing. De interface moet zo gemaakt worden, dat de processor de, voor de informatie-overdracht noodzakelijke, taken kan verzorgen.
Deze taken zijn o.a.:

a. Datatransport tussen de CPU en de randapparatuur.
b. De selectie van het gewenste apparaat (dus de adressering) als er meer apparaten aanwezig zijn.
c. Het besturen van de randapparatuur d.m.v. besturingssignalen en het verwerken van statussignalen, die van de randapparatuur komen.
d. Coördinatie van de samenwerking tussen diverse randapparaten.
e. Coördinatie van randapparatuur, CPU en geheugen zoals b.v. bij DMA.

Als we de koppeling tussen de CPU en de buitenwereld willen verzorgen, betekent dit dat er een pad moet worden gemaakt, waarover de informatie-uitwisseling plaats kan vinden. Dit pad kan bestaan uit een verbinding met de drie bussen (adresbus, databus en besturingsbus), welke in de computer voorkomen. In geval van invoer van data zou de koppeling externe data door moeten laten naar de databus en dus naar de CPU. Bij uitvoer van data moet de data vanuit de CPU worden aangeboden aan de buitenwereld. We koppelen de databus aan een of meer output-latches en input-buffers. Deze noemen we dan resp. output- en input-poorten. Het wordt hierdoor mogelijk om met behulp van een input-instructie data vanuit de inputbuffer naar de CPU over te brengen. In de input-instructie zal een adres moeten zijn opgenomen, dat de bewuste buffer (input-poort) zal selecteren.
In geval van output, zal er een output-instructie moeten worden uitgevoerd, waarin d.m.v. een output-adres de gewenste output-poort wordt geselecteerd.

Maar alleen I/O-poorten en I/O-instructies zijn niet voldoende om een correcte data-uitwisseling te garanderen. Er moet een voorschrift voor de samenwerking tussen CPU en gebruikersapparatuur worden vastgelegd. Zo'n voorschrift wordt vaak /protocol/ genoemd. Bij het opstellen van een protocol speelt het begrip interface een belangrijke rol.

SAMENVATTING 1

1. Onder interfacing verstaan we het aansluiten van gebruikersapparatuur op een processorsysteem.
2. Een protocol is een voorschrift, volgens welk de samenwerking tussen CPU en gebruikersapparaat dient te verlopen.

2. HET BEGRIP INTERFACE

Een interface (letterlijk vertaald: scheidingsvlak) is een verzameling regels, die men in acht moet nemen als een processorsysteem met een ander systeem (b.v. een gebruikersapparaat) gekoppeld wordt.
Deze regels hebben betrekking op zowel de hardware als de software van deze koppeling. Immers een interface-schakeling alleen is niet voldoende. Er is ook een programma nodig om de samenwerking tussen CPU en randapparaat te besturen. Behalve dit moeten ook regels worden opgesteld t.b.v. de mechanische verbinding, b.v. aansluittabellen van connectoren, functies van de verschillende draden, niveau's en frequenties van de optredende elektrische signalen, eventueel gebruik van lijnversterkers bij lange leidingen, enz.

Vraag 1: Een interface omvat alleen hardware/alleen software/ hardware en software.

In fig. 1 zijn de signaallijnen aangegeven, die bij interfacing een rol spelen. Dit zijn: a. datalijnen,
b. besturingslijnen,
c. statuslijnen.

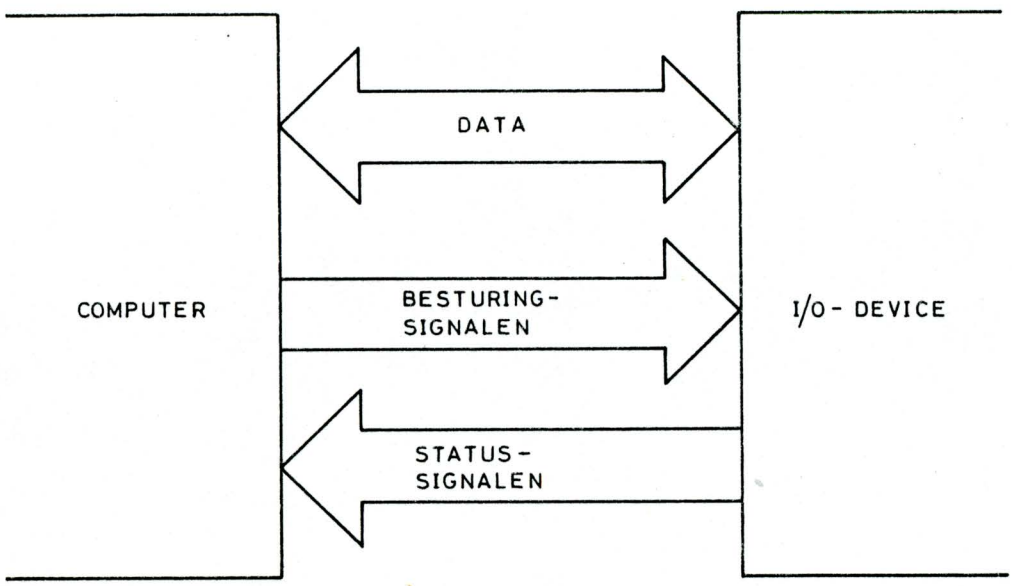


fig.1

Via de datalijnen wordt de feitelijke data getransporteerd. Het zijn meestal input-lijnen of output-lijnen. In sommige toepassingen worden datalijnen voor tweerichtingsverkeer gebruikt (te vergelijken met de bi-directionele databus).

Via de besturingslijnen geeft de CPU opdrachten (commando's) aan een randapparaat. Dit kunnen lees- of schrijfcommando's zijn, maar ook besturingssignalen voor motoren en koppen van magneetband- resp. magneetschijfeenheden. De signalen op deze besturingslijnen gaan altijd vanuit de computer naar het randapparaat.

Antw.1: hardware en software.

Via de statuslijnen meldt een randapparaat in welke toestand het zich bevindt. Het kunnen signalen zijn die aangeven dat het randapparaat data voor de CPU klaar heeft staan, data van de CPU heeft overgenomen, bezig is data te verwerken enz. Vaak komen ook z.g. error-statuslijnen voor. Met een error-signaal deelt een randapparaat mee, dat er een fout is opgetreden, b.v. een mechanische storing of het niet correct ontvangen van de door de CPU uitgezonden data.

Door het op de juiste wijze regelen van het uitsturen van commando's en het verwerken van statussignalen kan een data-uitwisseling tussen CPU en randapparaat worden gerealiseerd, die de volgende eigenschappen bezit:
a. Het optreden van fouten wordt tot een minimum beperkt.
b. Na een eventueel optredende fout, wordt direct actie ondernomen om deze fout te herstellen, of de gebruiker hierover in te lichten.

De I/O-methoden volgens welke zo'n data-uitwisseling kan worden gerealiseerd zijn:
a. geprogrammeerde I/O,
b. interrupt I/O,
c. direct memory access (DMA).

In deze les gaan we op de eerste twee methoden nader in.

SAMENVATTING 2

3. Een interface bevat o.a. voorschriften voor zowel hardware als alle software, die betrokken is bij de samenwerking tussen CPU en randapparaat.

4. De signaallijnen in een interface zijn te verdelen in
   a. datalijnen,
   b. besturingslijnen,
   c. statuslijnen.

5. De methoden, volgens welke de data-uitwisseling tussen computer en randapparatuur verlopen, zijn
   a. geprogrammeerde I/O,
   b. interrupt I/O,
   c. direct memory access.

3. EIGENSCHAPPEN VAN I/O

Uit bestudering van de tegenwoordig in computersystemen toegepaste randapparaten en de hierbij noodzakelijke interfaces blijkt, dat deze naar 3 gezichtspunten zijn in te delen, nl. naar
a. parallel of serie datatransport.
b. lage, middelmatige of hoge datatransportsnelheid (datarate).
c. synchroon of asynchroon datatransport.

De bovengenoemde punten hangen nauw samen met de wijze, waarop de verschillende randapparaten data opslaan of verwerken.

Vraag 2: Het datatransport van en naar magneetschijf verloopt parallel/in serie.

Zo zal de data bij een magneetschijf bit voor bit gelezen worden en wel met een zeer hoge leessnelheid. Bovendien kan een schijf niet worden gestopt, zodat de data synchroon met het aanbod gelezen of geschreven zal moeten worden. Er is voor elke bit een maximale tijd, waarin de data moet worden overgenomen.

Vraag 3: Bij een magneetbandeenheid is het datatransport parallel/serie.

Bij een meersporen magneetbandeenheid (b.v. 9 sporen) wordt de data parallel uitgelezen. Ook hier komt de data in groepen en zeer snel achter elkaar.
Een ander uiterste vinden we bij de ponsbandapparatuur. Bij b.v. een ponsbandlezer wordt de data parallel gelezen en wel byte voor byte. Het transportmechanisme stopt na elk teken totdat weer een stap- (of lees-)commando wordt gegeven. Hierbij kan de CPU in een door hem bepaald maar beperkt tempo, dus byte na byte overnemen en verwerken.
Bij een schijf- en bandeenheid kan door de hoge mechanische snelheid van het medium nauwelijks sprake zijn van overnemen en verwerken op byte-basis. Meestal zal zelfs de CPU te langzaam zijn voor het verwerken van de hoge datarate. Er zal hierbij een speciaal I/O-mechanisme, b.v. een DMA-controller of een aparte I/O-processor, moeten worden ingeschakeld.

Bij I/O-processen kan het ook voorkomen dat de gebeurtenissen elkaar niet snel opvolgen, maar dat een optredende gebeurtenis om een directe actie vraagt. We spreken in deze gevallen van een korte reactietijd. Ook voor dit soort processen is een speciaal I/O-mechanisme nodig.
Een ander soort problematiek zal optreden als het processorsysteem met een groot aantal apparaten te maken krijgt, die alle tegelijk in bedrijf zijn.
Tabel 1 geeft enkele eigenschappen van het datatransport bij een aantal veel voorkomende randapparaten.

De getallen in tabel 1 zijn richtwaarden, om u een idee te geven van de transportsnelheden. Met name magneetband- en magneetschijfeenheden zijn veel te snel voor de huidige microprocessors.
I.p.v. deze I/O-devices (= randapparaten) worden in microcomputers veelal normale cassetterecorders resp. floppy disks toegepast.

Antw.2: in serie. Antw.3: parallel.

Randapparaat	Transport	datarate/ baudrate	Transport van meer karakters
teletype	serie	110 bits/s	per karakter (start-stop)
visual display unit	serie	9600 bits/s	per karakter
line printer	parallel	20.000 kar./s of meer	per blok
magneetband-eenheid	parallel	200.000 kar/s of meer	per blok
magneetschijf-eenheid	serie	10.000.000 bits/s of meer	per blok
floppy disk	serie	250.000 bits/s	per blok
mini floppy disk(diskette)	serie	50.000 bits/s	per blok
mechanische ponsbandlezer/ -ponser	serie	110 bits/s	per karakter (start-stop)
foto-elektrische ponsbandlezer	parallel	500 kar./s	per blok

Tabel 1

De teletype en de mechanische ponsbandlezer/-ponser (deze drie zijn vaak gecombineerd in één terminal) werken op het z.g. start-stop-principe, d.w.z. dat het mechanische gedeelte vóór het uitwisselen van een karakter wordt gestart en na de laatste stopbit weer wordt gestopt. Dit is een van de redenen voor de lage transportsnelheid.

SAMENVATTING 3

6. De mogelijke vormen van datatransport zijn naar 3 gezichtspunten in te delen, nl.
   a. parallel of serie.
   b. transportsnelheid.
   c. synchroon of asynchroon.

7. De vorm van datatransport, die we voor een bepaald randapparaat kiezen, hangt nauw samen met de wijze waarop dit randapparaat data verwerkt of opslaat.

8. We spreken van systemen met een korte reactietijd als op elke actie direct wordt gereageerd. Dit wil nog niet zeggen, dat de afzonderlijke acties elkaar in hoog tempo opvolgen.

4. GEPROGRAMMEERDE I/O

Bij geprogrammeerde I/O wordt de data-uitwisseling volledig bestuurd d.m.v. instructies, die de CPU uitvoert. Dit betekent dat d.m.v. input-instructies data en statussignalen via input-poorten naar de CPU worden overgebracht en dat d.m.v. output-instructies data en commando's naar een I/O-device worden gestuurd.

Vraag 4: Geprogrammeerde I/O is geschikt voor langzame/snelle randapparaten.

Doordat alles door middel van de instructies van het programma wordt bestuurd, kan deze methode slechts voor langzame apparaten worden toegepast. De CPU moet namelijk achtereenvolgens een aantal stappen doorlopen, die samen een relatief lange tijd in beslag nemen.
Bij de geprogrammeerde I/O kunnen we twee soorten onderscheiden, nl.:
a. I/O zonder acknowledge (zonder terugmelding),
b. I/O op handshake basis (met terugmelding).

a. Geprogrammeerde I/O zonder terugmelding

Bij I/O zonder acknowledge (= bevestiging), wordt door het device, dat data afgeeft, samen met de data een z.g. DATA READY-signaal gegenereerd. Dit DATA READY-signaal geeft aan, dat er nieuwe data op de datalijnen aanwezig is.

Fig.2 geeft hiervan een voorbeeld met een input-apparaat. Fig.3 is het bijbehorende timing-diagram.

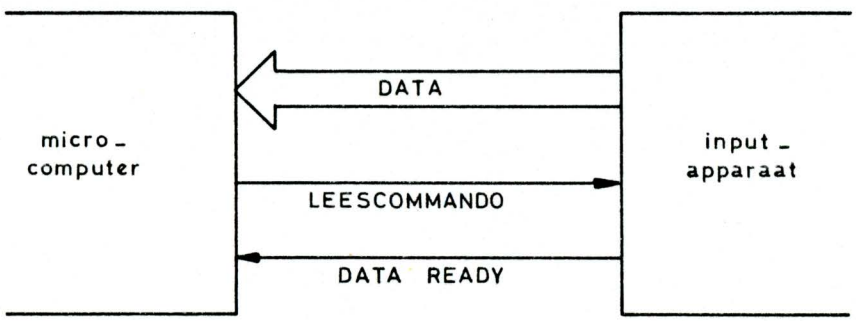


fig.2

Antw.4: langzame.

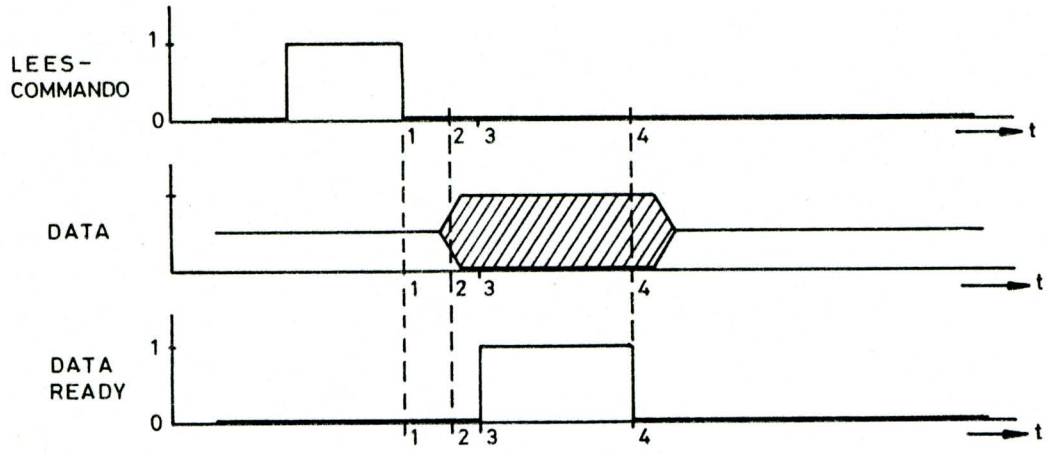


fig.3

In dit voorbeeld is het input-apparaat zo geconstrueerd, dat na een overgang van 1 naar 0 van het leescommando een leesactie wordt gestart. In fig.3 vindt deze achterflank op t = 1 plaats.
Na een (meestal door mechanische acties veroorzaakte) vertragingstijd, zal het input-apparaat data op de datalijnen plaatsen (t = 2).
Even daarna zal DATA READY hoog worden om aan de CPU mee te delen dat de data stabiel is en overgenomen kan worden. De CPU moet dit terstond doen, omdat het DATA READY-signaal maar van korte duur is.
Na een door de eigenschappen van het input-apparaat bepaalde tijd, worden data en DATA READY-signaal weggenomen en zijn we weer in de beginsituatie beland.

Vraag 5: Het wegnemen van de data gebeurt wel/niet in opdracht van de CPU.

De data is dus na een bepaalde tijd verdwenen, ongeacht het feit dat de CPU deze wel of niet heeft overgenomen. De snelheid van de CPU en het input-apparaat moeten dan wel volkomen op elkaar zijn afgestemd.
In fig.4 en fig.5 is aangegeven, wat de gevolgen zijn als bij deze I/O-methode de beide devices niet op elkaar zijn aangepast.

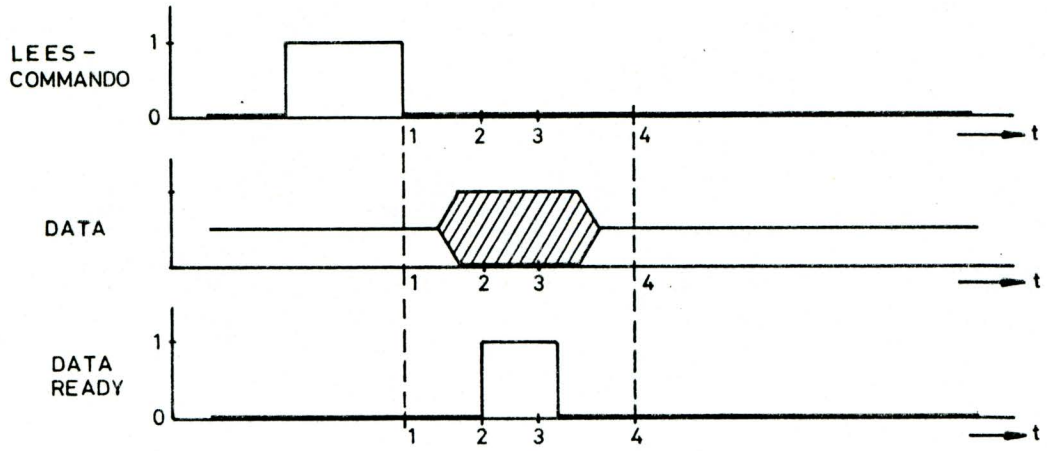


fig.4

In fig.4 is aangegeven wat er gebeurt als het randapparaat sneller is dan de CPU. Op t = 1 wordt d.m.v. een leescommando een leesactie gestart. De CPU begint dan d.m.v. input- en logische instructies te testen of DATA READY al 1 is. Omdat voor de uitvoering van deze instructies een zekere tijd (minimaal enkele µs) nodig is, wordt niet op t = 2, maar pas op t = 3 een 1 gedetecteerd. Dan zal de CPU, ook weer m.b.v. een instructie, de data van de datalijnen inlezen. Omdat ook voor deze instructie een zekere tijd nodig is, gebeurt dit op t = 4. Echter het veel snellere input-apparaat heeft de data en DATA READY al weggenomen. Het gevolg is, dat hetgeen de CPU inleest niet de bedoelde data is. Er is dus een karakter verloren gegaan.

Hetzelfde zou natuurlijk zijn gebeurd, als het randapparaat zoveel sneller was, dat data en DATA READY al zijn weggenomen, op het moment dat de CPU de eerste test uitvoert.

Vraag 6: De CPU leest dan wel/niet een foutief karakter in.

Omdat de CPU in dit geval ook het DATA READY-signaal heeft gemist, zal deze doorgaan met het testen of dit signaal 1 wordt.
Als we geen bijzondere voorzieningen treffen (b.v. het aangeven van een maximaal aantal testen), dan zal er van de rest van de programma-uitvoering weinig terecht komen. De CPU blijft dan immers in de testlus doorgaan.

In fig.5 is het tegenovergestelde weergegeven.
Het randapparaat is nu te traag t.o.v. de CPU.

Antw.6: niet.

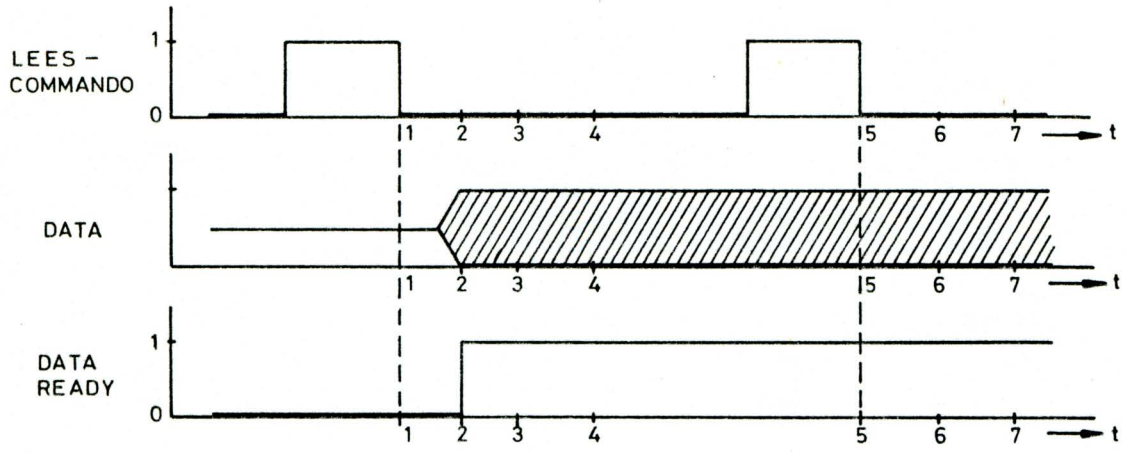


fig.5

Op t = 1 wordt weer een leesactie gestart. Op t = 2 is het DATA READY-signaal dan 1 geworden, wat op t = 3 door de CPU wordt gedetecteerd. Op t = 4 wordt de data dan ingelezen. De CPU kan dan verder gaan met de programma-uitvoering. Als er meteen weer een opdracht tot het inlezen van een karakter volgt (b.v. bij het invoeren van een blok data), dan zal op t = 5 een nieuwe leesopdracht worden gegeven.
De CPU begint dan met het testen van DATA READY. Direct bij de eerste test (t = 6) detecteert de CPU al een 1. Het DATA READY-signaal is nl. door de traagheid van het input-apparaat nog niet weggenomen.
De CPU leest dan op t = 7 nieuwe data in.
Het gevolg is, dat de CPU dezelfde data als twee (of meer) afzonderlijke karakters invoert.

Geprogrammeerde I/O zonder terugmelding, ook wel simple I/O genoemd, kan dus alleen worden toegepast als de snelheden van de CPU en I/O-device volkomen op elkaar zijn afgestemd.
Deze soort interfacing wordt vaak toegepast in gevallen, waar nauwelijks of geen besturingslogica in het betreffende randapparaat aanwezig is.

Het voordeel van deze interfacing is de betrekkelijke eenvoud van de benodigde software (fig.6).

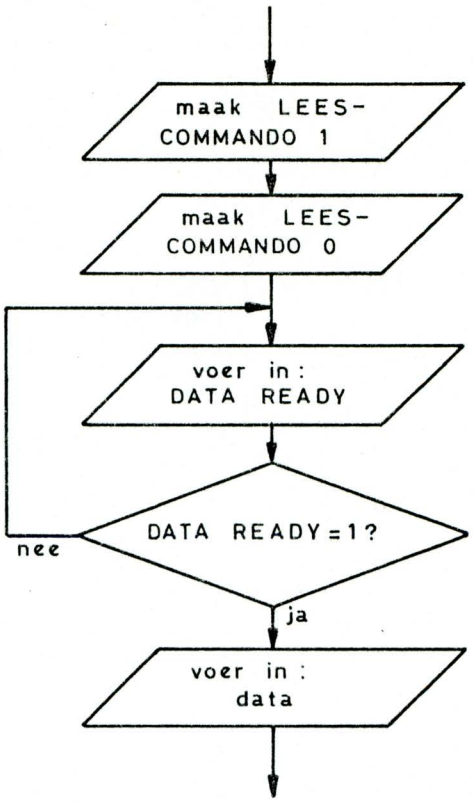


fig.6

SAMENVATTING 4

9. Bij geprogrammeerde I/O wordt de data-uitwisseling tussen CPU en randapparatuur volledig door instructies (dus software-matig) bestuurd.

10. Geprogrammeerde I/O is te splitsen in:
    a. I/O zonder acknowledge.
    b. I/O met handshake.

11. Bij I/O zonder acknowledge wordt samen met de data een z.g. DATA READY-signaal afgegeven. Er wordt echter niet gecontroleerd of de tegenpartij deze data op de juiste wijze overneemt.

12. Een nadeel van I/O zonder acknowledge is dat er gemakkelijk data verloren kan gaan of dubbel wordt gelezen.

13. Een voordeel van I/O zonder acknowledge is dat de benodigde software vrij eenvoudig is.

b. Geprogrammeerde I/O met terugmelding

Wanneer we I/O op basis van handshake plegen, worden een aantal problemen uit de vorige paragraaf voorkomen. Door het handshake-protocol wordt het mogelijk om zowel de CPU als het randapparaat van elkaar te laten weten hoe ver ze zijn gevorderd met het datatransport.
We zullen dit bespreken aan de hand van een voorbeeld, waarin geprogrammeerde output op basis van handshake plaatsvindt. Het output-apparaat is volgens fig.7 met de microcomputer gekoppeld. Fig.8 is het bijbehorende timing-diagram.

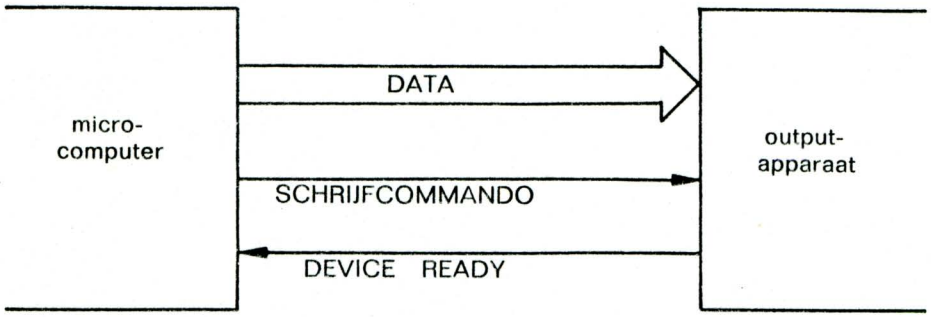


fig.7

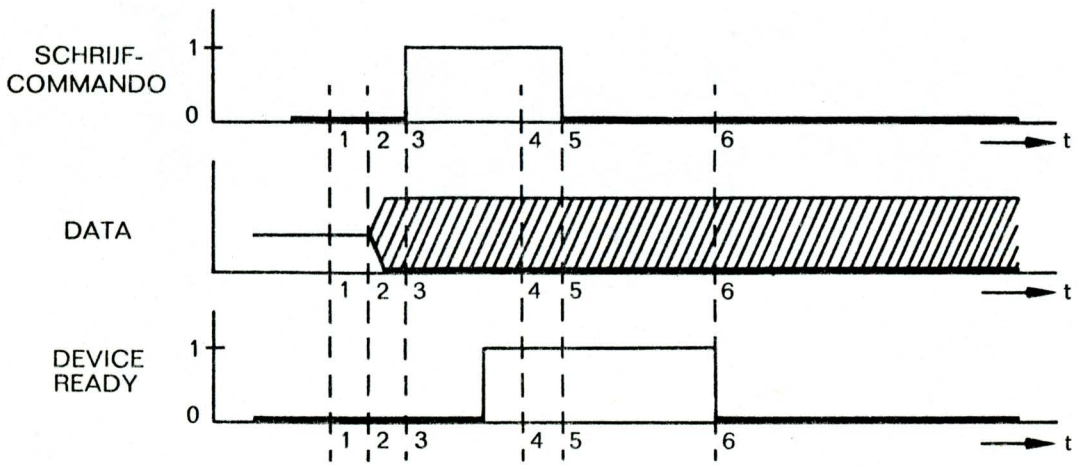


fig.8

Zoals in de meeste praktische toepassingen geeft ook in dit voorbeeld het output-apparaat een 0 op de DEVICE READY (= apparaat klaar)-lijn af, als het device in staat is om aan een I/O-actie deel te nemen.
Dit heeft het volgende voordeel.
Een open lijn wordt vrijwel altijd als een 1 gedetecteerd. Als het be-

treffende output-apparaat niet is ingeschakeld of als er helemaal geen output-apparaat aanwezig is, wordt er geen actief signaal op de DEVICE READY-lijn geplaatst. De lijn is dan open en de (micro)computer detecteert dan een 1.
Op deze wijze is het onmogelijk, dat de CPU een schrijfactie start, als er geen functionerend randapparaat is aangesloten.

In fig.9 is het stroomdiagram weergegeven voor de acties, die plaats moeten vinden, voor het uitvoeren van één karakter.
Deze acties zijn ondergebracht in een subroutine, die voor elk uit te voeren karakter opnieuw moet worden aangeroepen.
In het aanroepende programma (hoofdprogramma) wordt dan bepaald, wanneer en hoe vaak deze subroutine (b.v. bij het uitvoeren van een heel blok data) moet worden aangeroepen.

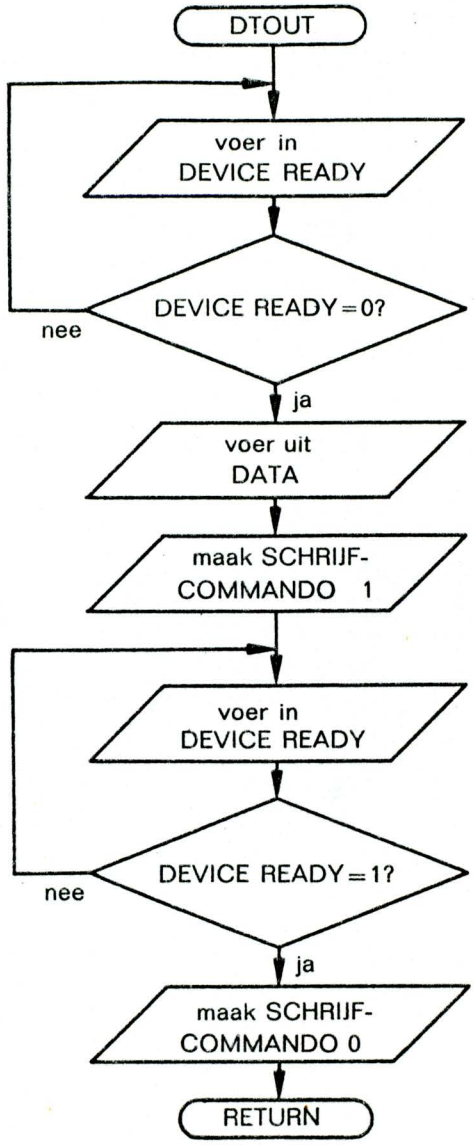


fig.9

Op t = 1 moet als gevolg van instructies uit het programma een schrijfactie starten. De CPU test het DEVICE READY-signaal. Dit is 0 (fig.8). Het output-apparaat staat dus klaar om data over te nemen. De CPU plaatst dan (via een output-poort) data op de datalijnen (t = 2) en maakt direct daarna het schrijfcommando actief (t = 3). Daarna wordt het DEVICE READY-signaal getest. Op t = 4 wordt gedetecteerd, dat dit signaal 1 is. Het randapparaat is dus ergens mee bezig. Dit moet dan wel het overnemen van de zojuist uitgevoerde data zijn. De CPU kan dan het schrijfcommando weer 0 maken (t = 5).
Omdat de data via een output-poort is uitgevoerd, zal deze ongewijzigd op de datalijnen blijven staan.
In antwoord op het wegnemen van het schrijfcommando zal, na een door de (mechanische) traagheid van het output-apparaat bepaalde tijd, het DEVICE READY-signaal weer 0 worden (t = 6). Het device staat dus klaar om een volgend karakter over te nemen.

Vraag 7: De CPU mag tussen t = 5 en t = 6 wel/niet een nieuwe schrijfactie starten.

Antw.7: wel.

Na het wegnemen van het schrijfcommando (op t = 5) mag de CPU direct een nieuwe schrijfactie starten. Door de eerste 2 blokken in fig.9 wordt immers voorkomen, dat er nieuwe data en een schrijfcommando worden uitgevoerd op het moment, dat het output-apparaat nog bezig is. Door deze twee blokken blijft de CPU tot na t = 6 wachten met het uitvoeren van nieuwe data.

Dit voorbeeld toont duidelijk een nadeel van geprogrammeerde I/O. Wanneer de CPU een blok data via een traag output-apparaat uitvoert, dan wordt een groot deel van de tijd besteed aan testen en wachten, totdat er aan een bepaalde voorwaarde is voldaan.

Er kan zich echter nog een probleem voordoen. Het apparaat kan b.v. de opdracht niet met het gewenste resultaat hebben uitgevoerd. Om dit te signaleren zouden we de interface nu moeten uitbreiden met minimaal twee lijnen (fig.10). Een lijn van het apparaat naar de computer, welke b.v. hoog wordt als er een fout is opgetreden en een lijn van de computer naar het apparaat om het een reset te geven.

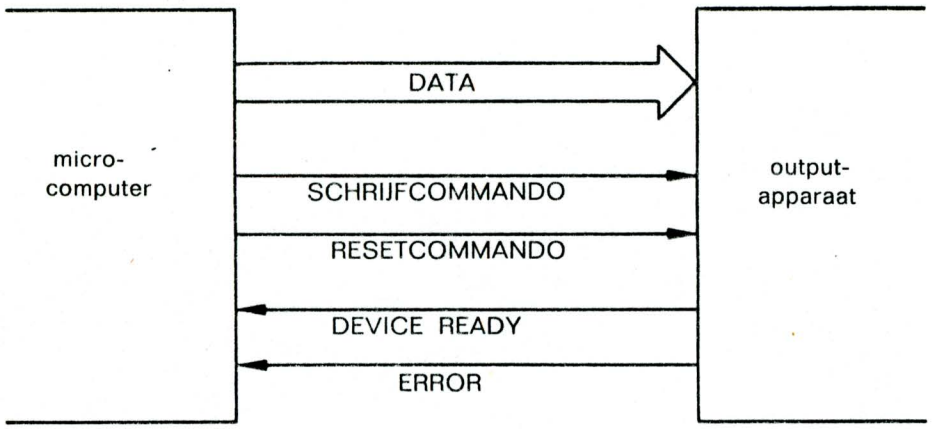


fig.10

Opmerking:

Vaak komen er een reeks commandolijnen en een reeks ERROR-lijnen voor. Een en ander is afhankelijk van het type apparaat. Het output-protocol moet nu als volgt worden aangepast.
Bij het testen of het apparaat klaar is voor gebruik, moet zowel de DEVICE READY- als de ERROR-lijn worden getest. In geval van ready met een error-indicatie moet gestopt worden en worden teruggekeerd naar het aanroepende programma met daarbij de mededeling dat er iets mis is met het I/O-apparaat. Immers tijdens de start van het computersysteem zijn alle apparaten gereset. Wanneer zowel de DEVICE READY-lijn als de ERROR-lijn laag zijn, kan worden begonnen met het datatransport. Dit verloopt op gelijke wijze als in het vorige voorbeeld. Nadat de DEVICE READY-lijn nu weer van hoog naar laag gaat om terug te melden dat de transport-operatie klaar is, moet de CPU nog testen of de ERROR-lijn

actief is. In geval van een fout zal de CPU een resetcommando geven aan het apparaat en terugkeren naar het aanroepende programma, maar nu met een foutmelding. In het aanroepende programma moeten dan voorzieningen worden gemaakt om actie te nemen als er fouten voorkomen.
In de volgende paragraaf behandelen we hiervan een voorbeeld.

SAMENVATTING 5

14. In systemen met I/O op basis van handshake wordt steeds gecontroleerd of de tegenpartij op de juiste wijze op een afgegeven besturingssignaal heeft gereageerd.

15. Door het opstellen van een goed handshake-protocol kan het verloren gaan of dubbel lezen van data worden voorkomen.

16. In systemen met handshake I/O komen vaak ERROR-lijnen voor. Hiermee meldt een randapparaat dat er een fout is opgetreden. In het handshake-protocol zijn dan voorschriften opgenomen voor het opheffen of corrigeren van deze fouten.

5. VOORBEELD 1

Als voorbeeld van geprogrammeerde I/O met terugmelding behandelen we een situatie waarin een microcomputer met een ponsbandponser (papertape punch) en een ponsbandlezer (papertape reader) is gekoppeld.
Om u in staat te stellen delen uit dit voorbeeld te simuleren en te testen, gaan we uit van de SDK 85 met 2 output-poorten (van de 8155) en 2 input-poorten (van de 8355).

Het hardware-schema is dat van fig.11.

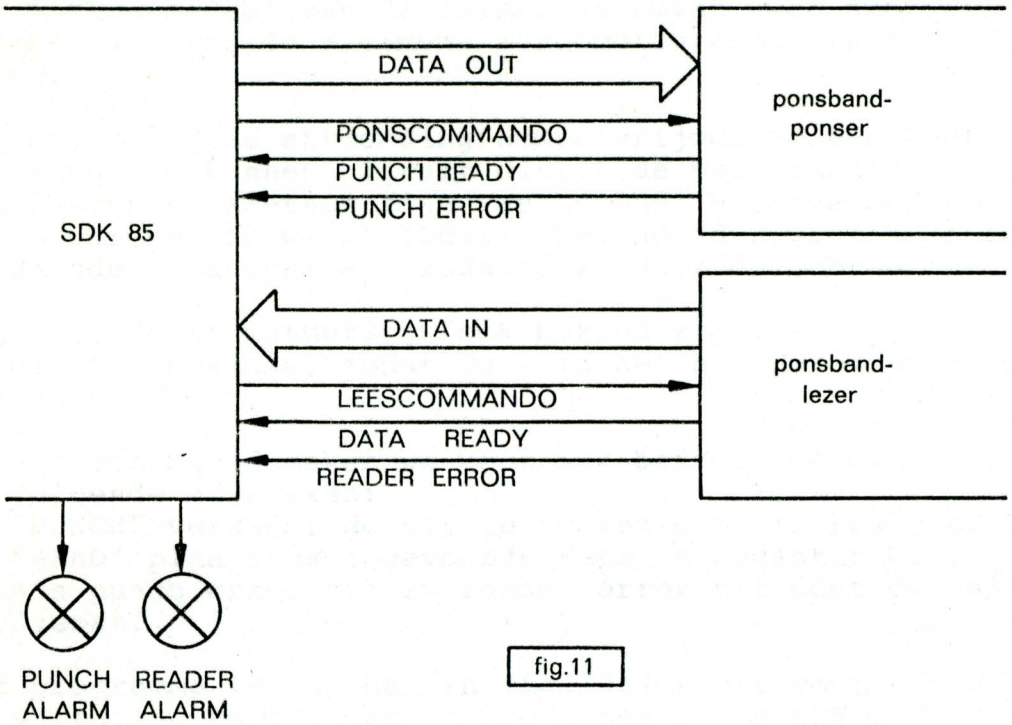
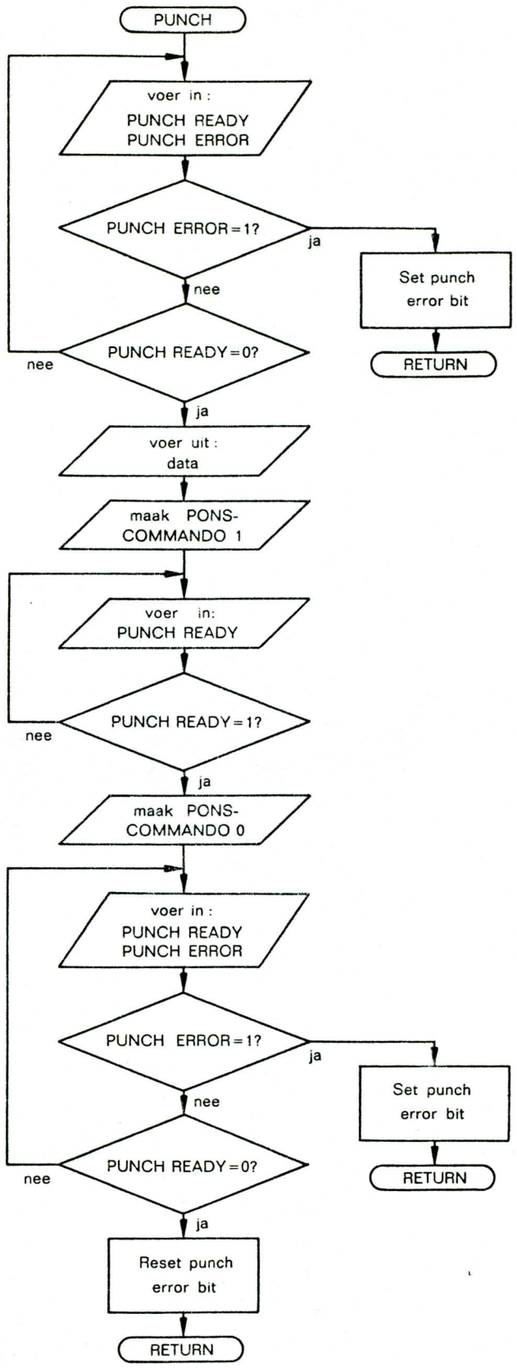


fig.11

In tabel 2 is van elk van de 32 I/O-lijnen de functie beschreven.

Poort	Lijn	Functie	actieve toestand
00	b_0 b_1 b_2 b_3	PUNCH ERROR PUNCH READY READER ERROR DATA READY	1 0 1 0
01	b_0-b_7	input data	-
21	b_0-b_7	output data	-
22	b_0 b_1 b_6 b_7	ponscommando leescommando PUNCH ALARM READER ALARM	1 1 1 1

Tabel 2



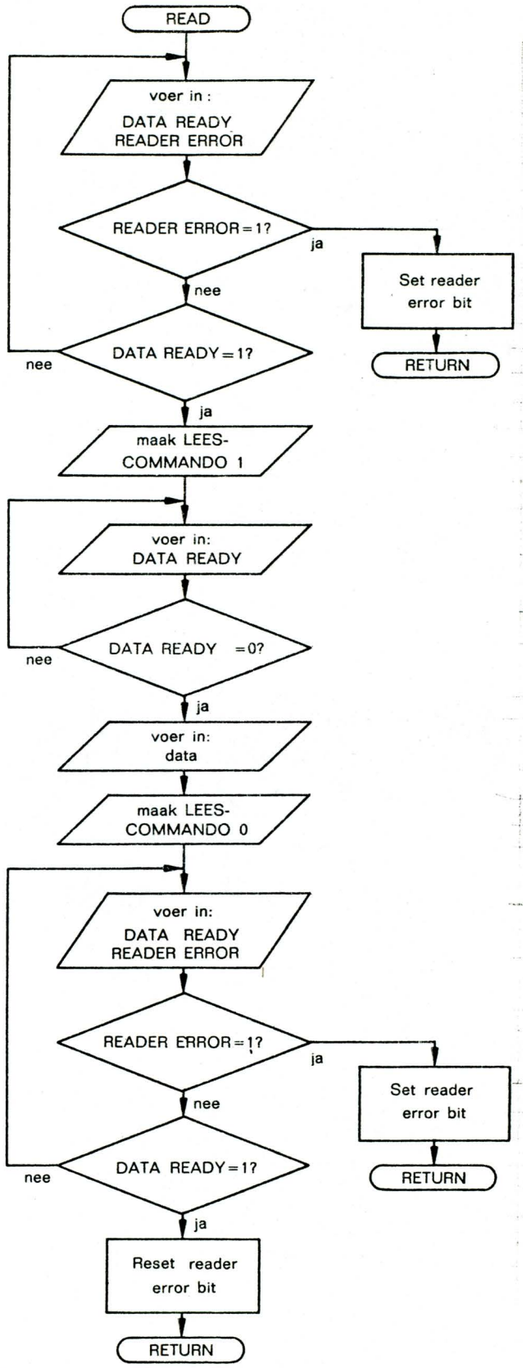


fig.12

```
                        ORG   2800H
2800 DB 00  PUNCH:      IN    00H
2802 0F                 RRC
2803 D8                 RC
2804 0F                 RRC
2805 DA0028             JC    PUNCH     ;SPRING ALS READY ≠ 0
2808 79                 MOV   A,C       ;HAAL DATA OP
2809 D3 21              OUT   21H       ;VOER DATA UIT
280B 3E 01              MVI   A,01H     ;MAAK PONS-
280D D3 22              OUT   22H       ; COMMANDO 1
280F DB 00  PUN1:       IN    00H
2811 0F                 RRC             ;TEST PUNCH READY
2812 0F                 RRC
2813 D20F28             JNC   PUN1      ;SPRING ALS READY ≠ 1
2816 AF                 XRA   A         ;MAAK PONS-
2817 D3 22              OUT   22H       ; COMMANDO 0
2819 DB 00  PUN2:       IN    00H
281B 0F                 RRC             ;TEST PUNCH ERROR
281C D8                 RC              ;RETURN ALS ERROR = 1
2810 0F                 RRC             ;TEST PUNCH READY
281E DA1928             JC    PUN2      ;SPRING ALS READY ≠ 0
2821 C9                 RET             ;ERROR BIT = 0
                        ORG   2840H
2840 DB 00  READ:       IN    00H
2842 0F                 RRC             ;TEST
2843 0F                 RRC             ; READER
2844 0F                 RRC             ;  ERROR
2845 D8                 RC              ;RETURN ALS ERROR = 1
2846 0F                 RRC             ;TEST DATA READY
2847 024028             JNC   READ      ;SPRING ALS READY ≠ 1
284F 3E 02              MVI   A,02H     ;MAAK LEES-
284C D3 22              OUT   22H       ; COMMANDO 1
284E DB 00  RD1:        IN    00H
2850 E6 08              ANI   08H       ;TEST DATA READY
2852 C24E28             JNZ   RD1       ;SPRING ALS READY ≠ 0
2855 DB 01              IN    01H       ;VOER
2857 47                 MOV   B,A       ; DATA IN
2858 AF                 XRA   A         ;MAAK LEES-
2859 D3 22              OUT   22H       ; COMMANDO 0
285B DB 00  RD2:        IN    00H
285D 0F                 RRC             ;TEST
285E 0F                 RRC             ; READER
285F 0F                 RRC             ;  ERROR
2860 D8                 RC              ;RETURN ALS ERROR = 1
2861 0F                 RRC             ;TEST DATA READY
2862 025B28             JNC   RD2       ;SPRING ALS READY ≠ 1
2865 3F                 CMC             ;RESET ERROR BIT
2866 C9                 RET
                        END
```

fig.13

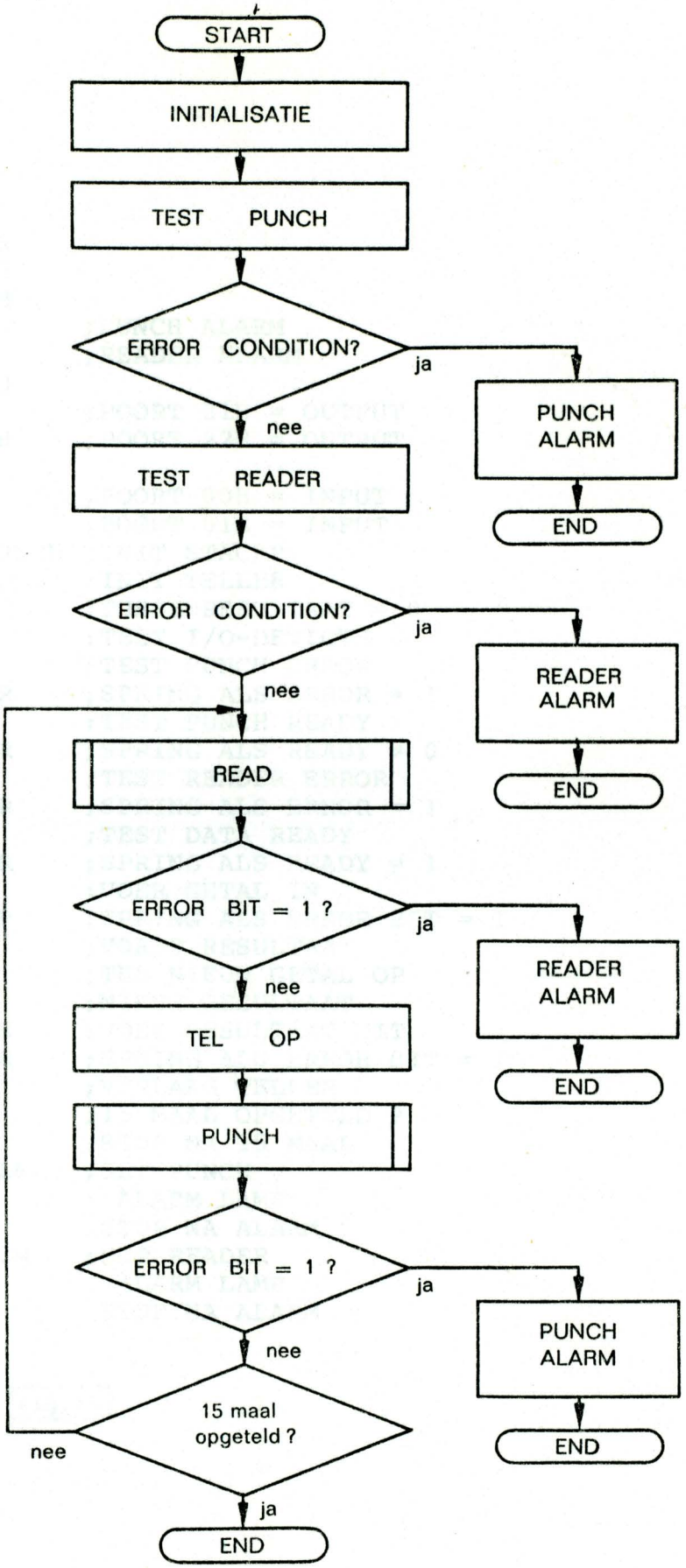

START
INITIALISATIE
TEST PUNCH
ERROR CONDITION?
ja
PUNCH ALARM
END
nee
TEST READER
ERROR CONDITION?
ja
READER ALARM
END
nee
READ
ERROR BIT = 1 ?
ja
READER ALARM
END
nee
TEL OP
PUNCH
ERROR BIT = 1 ?
ja
PUNCH ALARM
END
nee
15 maal opgeteld ?
nee
ja
END

fig.14

carry status flag zijn opgeslagen.
Het stroomdiagram en programma worden dan zoals in fig.14 resp. fig.15 is weergegeven. (Ga deze voor uzelf na.)

Vertaal nu de programma's uit fig.13 en fig.15. Plaats daarna het objectprogramma in het RAM-geheugen van de computer en start het.

Om het programma te kunnen testen, moet u zich als ponsbandponser en -lezer gaan gedragen.
Voor het gemak kunt u een lijstje met de functie van elke I/O-lijn naast de schakelaars en LED's op de uitbreidingsprint leggen (b.v. fig.16).

Om een correcte data-uitwisseling te simuleren, moet u voor het starten van het programma de volgende statussignalen aan de computer aanbieden:

- PUNCH ERROR = 0
- PUNCH READY = 0 (u kunt data van de microcomputer ontvangen)
- READER ERROR = 0
- DATA READY = 1 (zonder een leescommando heeft u geen data voor de microcomputer klaar staan)

Na het starten van het programma ontvangt u een actief leescommando (LED brandt). U stelt met de schakelaars op poort 01_{16} de gewenste data in en geeft dan een actief DATA READY-signaal. De computer neemt dan het leescommando weg en blijft wachten totdat u DATA READY weer 1 maakt.

Daarna gaat de computer de data verwerken en wil het ontstane resultaat door de ponsbandponser op ponsband laten zetten. U ontvangt dan een actief ponscommando. Als u het resultaat op de LED's van poort 21_{16} heeft bekeken (en eventueel opgeschreven), meldt u dit met een 1 op de PUNCH READY-lijn. De computer neemt dan het ponscommando weg en blijft wachten totdat u PUNCH READY weer 0 maakt.

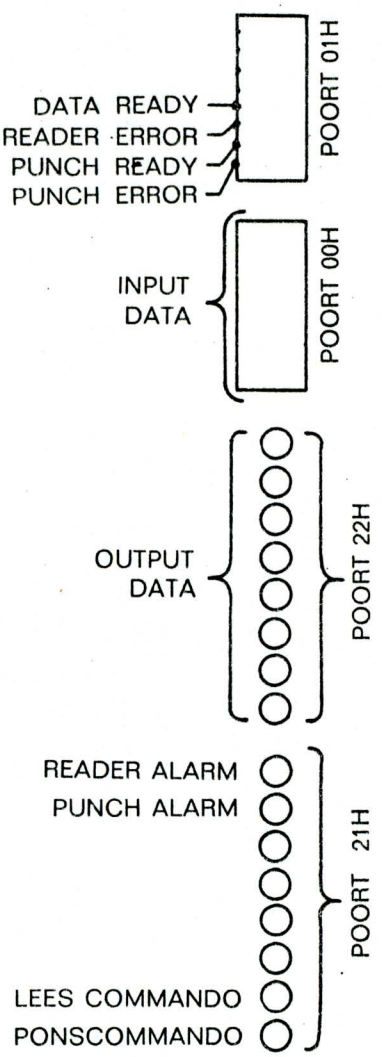


fig.16

Op deze wijze moet u nogmaals 14 maal de ponsbandponser en -lezer simuleren. Daarna stopt de programma-uitvoering. Als u alles correct heeft gedaan, zal er geen alarmlamp gaan branden.

Dirksen Velden.

U ziet, dat hoe traag u als randapparaat ook reageert, er bij deze geprogrammeerde I/O met handshake geen fouten optreden.
Er wordt nl. nooit een volgende actie gestart, wanneer niet beide partijen hiervoor klaar zijn.

Opmerking:

Controleer ook wat er gebeurt als u voor het starten van het programma niet de juiste beginsituatie heeft ingesteld of als u tijdens de data-uitwisseling een actief ERROR-signaal geeft.

6. INTERRUPT I/O

Een nadeel van geprogrammeerde I/O is dat een groot deel van het programma bestaat uit lussen, waarin wordt gewacht op een toestandsverandering in het randapparaat. Het is duidelijk, dat hierbij veel tijd wordt verspeeld aan het uitvoeren van niet functionele instructies.
De CPU zal veel werk moeten verrichten zonder een stap verder te komen.
Als we het I/O-proces vanuit een andere hoek bekijken, zien we dat er alleen maar data-uitwisseling mogelijk is als de CPU naar het randapparaat kijkt. We kunnen concluderen, dat met de normale manier van geprogrammeerde I/O we vaak iets anders zouden willen doen dan het wachten op het I/O-apparaat, maar van de andere kant zouden we dit "stopwerk" moeten kunnen onderbreken, zodra toestandsveranderingen in een I/O-apparaat plaatsvinden.
De "normale" hardware in een CPU is meestal niet geschikt voor dit soort handelingen, maar door enkele uitbreidingen wordt het toch mogelijk.
Het principe van taken onderbreken voor taken met een hogere prioriteit wordt interrupt genoemd.
Wanneer we deze methode voor input- en output-acties toepassen, spreken we van interrupt I/O.

In de rest van deze les gaan we ervan uit, dat u de stof uit de les "Interrupt" beheerst. Lees die les zonodig nog eens door.

Het is mogelijk, dat in een computersysteem meer interruptiebronnen (= devices die interrupt requests afgeven) voorkomen.
Het probleem waarmee we dan te maken krijgen, is de identificatie van de interruptiebron. Om naar de gewenste (bij het interrumperende apparaat behorende) service routine te kunnen springen, moet er een relatie worden gelegd tussen de interruptiebron en het adres van de betreffende routine.
Het vaststellen van het beginadres van de juiste interrupt service routine kan op twee verschillende manieren plaatsvinden, nl. volgens
a. polled interrupt,
b. vectored interrupt.

Beide methodes zullen nu worden besproken.

a. Polled interrupt

In fig.17 is de hardware-configuratie van een systeem met polled interrupt-afhandeling weergegeven (to poll = stemmen behalen, onderzoeken).

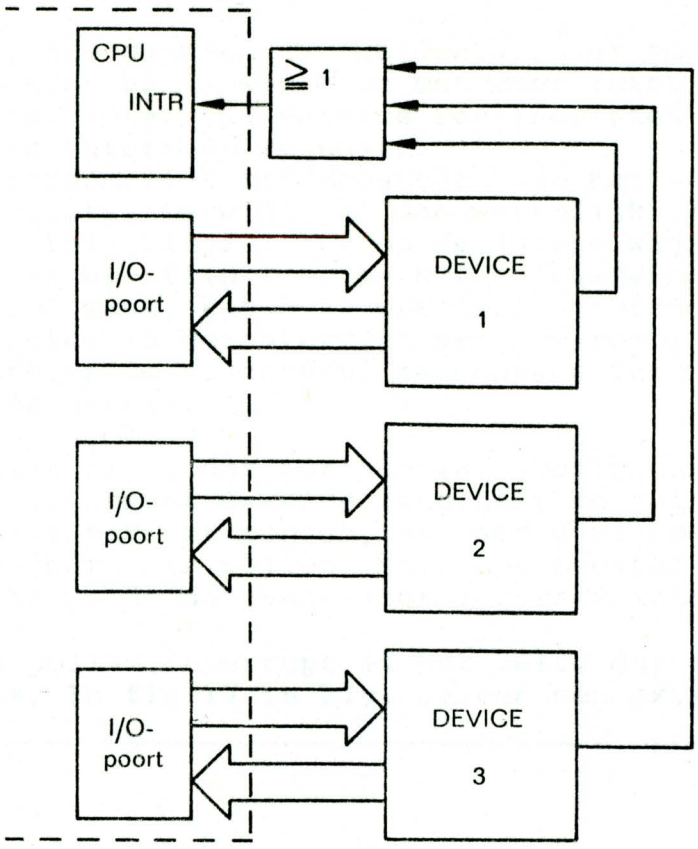


fig. 17

Bij polled interrupt worden de interrupt request-lijnen van alle devices (in fig.17 zijn er slechts 3 getekend) samengebundeld en b.v. via een OF-functie aangesloten op de interrupt ingang van de CPU. Wanneer de CPU een interrupt request binnenkrijgt, zal daarna via de normale I/O-poorten aan elk apparaat moeten worden gevraagd of hij soms de bron van de interrupt was. Het zal duidelijk zijn dat door de volgorde van afvragen, welke er altijd in het programma zal zitten, er een prioriteit is ontstaan. Het apparaat dat het eerst aan de orde komt, heeft een hogere prioriteit dan het volgende, enz.
Nadat zo de bron van de interrupt request is gelocaliseerd kan, nu via een sprongtabel, naar de bijbehorende interrupt service routine worden gesprongen. In de sprongtabel staat voor elk apparaat het adres van de interrupt service routine.
In het geval van het ontvangen en detecteren van een interrupt request door de CPU, is het noodzakelijk dat er een terugmelding komt naar de bron van de interrupt request.

b. Vectored interrupt

In fig.18 is de hardware-configuratie van een systeem met vectored interrupt-afhandeling weergegeven.

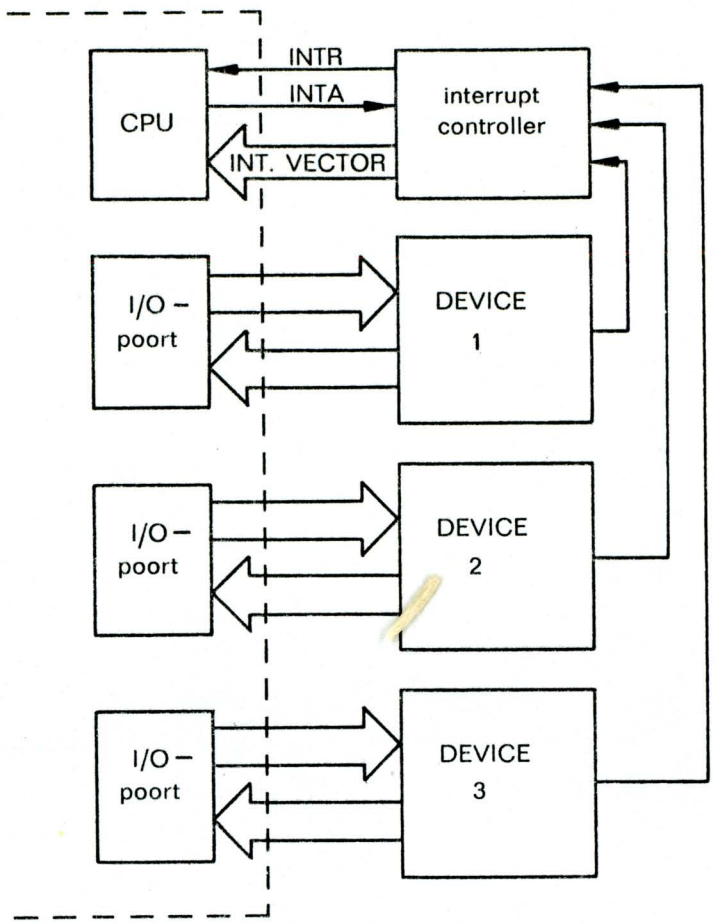


fig.18

Bij vectored interrupt wordt een groot deel van de taken, door de interrupt controller overgenomen.
De interrupt requests van de I/O-devices worden door de interrupt controller gebundeld tot één lijn, die met de interrupt-ingang van de CPU is verbonden. Als een doorgegeven interrupt request is geaccepteerd zendt de CPU een INTA-signaal (= interrupt acknowledge) terug. De interrupt controller geeft dan een interrupt vector af. Dit is het nummer van de interruptbron. De CPU kan hieruit eenvoudig het beginadres van de bij deze interruptbron behorende interrupt service routine afleiden. Er zijn zelfs (programmeerbare) interrupt controllers, die direct het juiste restart-adres (= beginadres van de interrupt service routine) afgeven. Een voorbeeld hiervan is de 8259. De CPU behoeft dan slechts dit restart-adres in te lezen en in de programmateller te plaatsen.

De huidige interrupt controllers zijn vaak programmeerbare chips. Aan het begin van het programma kunnen we zo'n chip dan initialiseren, b.v. voor wat betreft prioriteiten en interrupt masks.

Zie voor de afhandeling van vectored interrupt en de opbouw van interrupt controllers de les "Interrupt".

7. INTERRUPT INTERFACE

In de voorgaande paragraaf is beschreven, hoe een I/O-apparaat een routine kan aanroepen, door het afgeven van een interrupt request. Als volgende stap volgt het schrijven van een programma voor een interface, die op interrupts is gebaseerd.

In het algemeen gaan we er bij het schrijven van zo'n I/O-programma (interrupt service routine) vanuit, dat er een hoofdprogramma is, waarmee de CPU bezig is, als er geen service routines zijn af te werken. Nemen we b.v. een input-apparaat, dan kan dit wel een interrupt request afgeven, als er data klaar is om ingevoerd te worden, maar dan moeten we van te voren dit apparaat starten met een leescommando. Na het geven van dit leescommando kan de CPU doorgaan met het uitvoeren van het hoofdprogramma, totdat het input-apparaat d.m.v. een interrupt request meldt, dat het data beschikbaar heeft (fig.19). Dan kan in de interrupt service routine de data worden ingevoerd en verderop in het hoofdprogramma worden verwerkt.

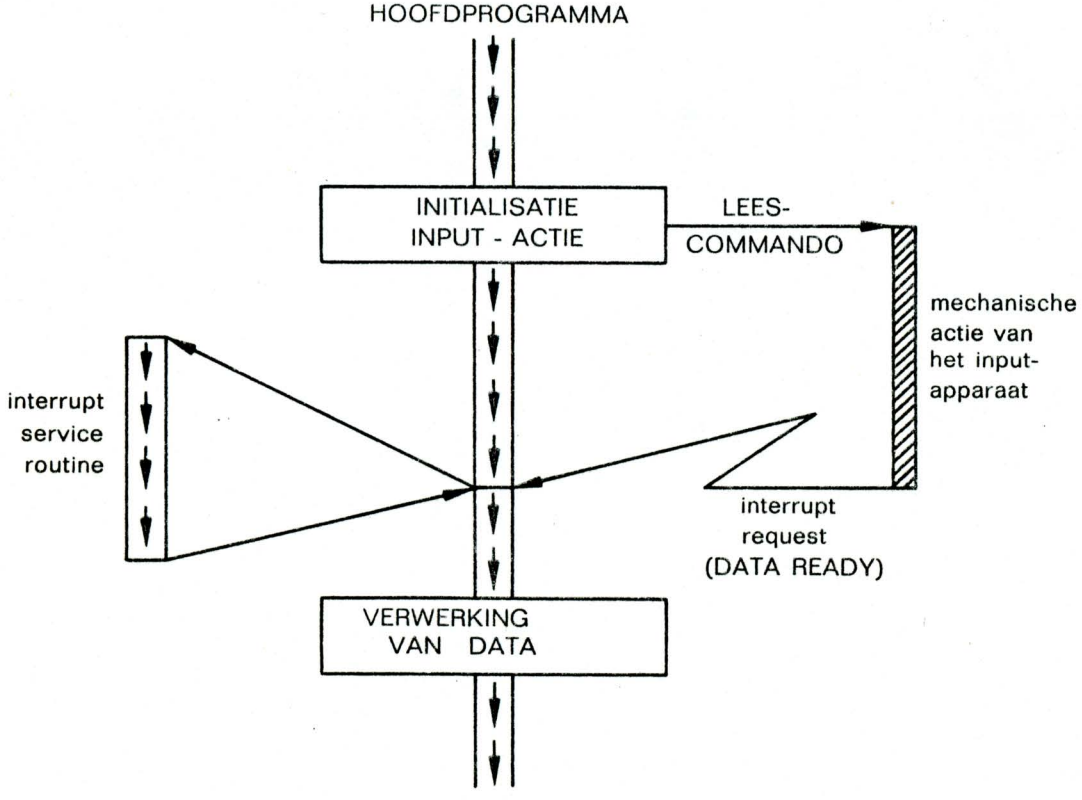


fig.19

Uit fig.19 blijkt, dat na het geven van het leescommando zowel de CPU als het input-apparaat tegelijk werkzaam zijn. T.o.v. geprogrammeerde I/O geeft dit dus een belangrijke tijdwinst.

Nog duidelijker wordt deze tijdwinst, als we fig.20 bekijken.
Hierin zijn 3 I/O-devices op de computer aangesloten.

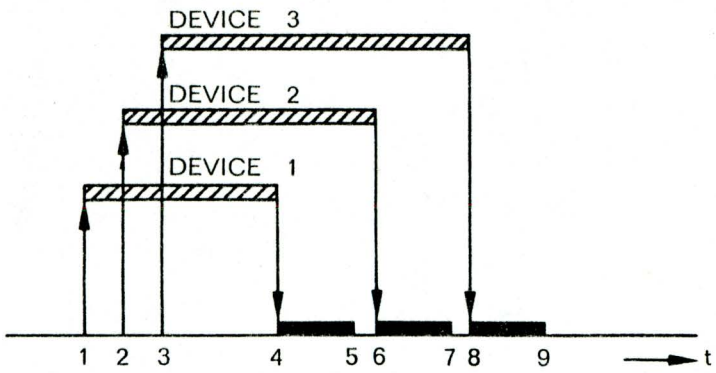


fig.20

Op t = 1, t = 2 en t = 3 worden de startcommando's (lees of schrijf) voor de 3 devices gegeven. Op t = 4 geeft het eerste device een interrupt request. In de daarop volgende interrupt service routine kan dan de feitelijke data-uitwisseling plaatsvinden.
Hetzelfde gebeurt op t = 6 en t = 8. Op t = 9 zijn de 3 I/O-acties achter de rug.

Ter vergelijking is in fig.21 weergegeven welke tijd met deze 3 I/O-acties zou zijn gemoeid, indien voor geprogrammeerde I/O was gekozen.

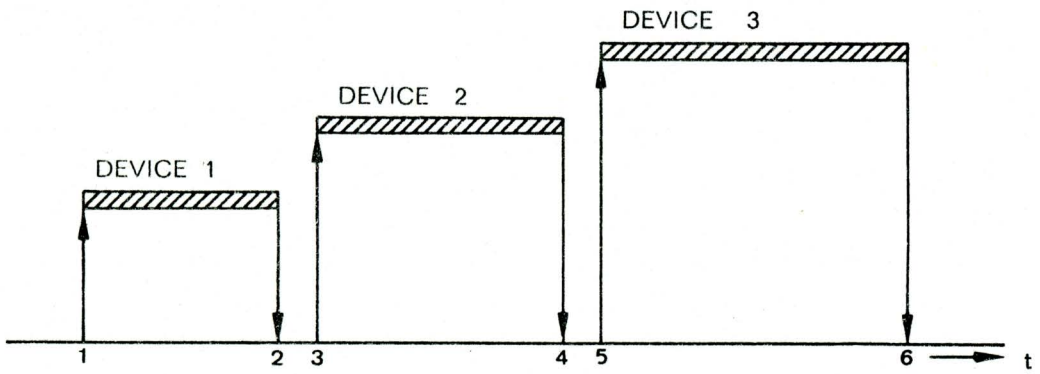


fig.21

Duidelijk is, dat de mechanische acties van de 3 devices nu niet parallel, maar na elkaar worden uitgevoerd.
Op t = 1 wordt device 1 gestart. Op t = 2 is de data beschikbaar.
Deze kan worden ingevoerd. Pas op t = 3 kan device 2 worden gestart, enz.
De totaal benodigde tijd is in dit geval dus veel groter dan bij interrupt I/O.

Het toepassen van interrupt betekent meestal tijdwinst. Bovendien wordt een I/O-apparaat direct geholpen.
Wanneer we een koppeling maken met een I/O-apparaat, zal de bron van de interrupt vele oorzaken kunnen hebben.
De meest voor de hand liggende oorzaak is een verandering van het DATA READY-signaal, een andere oorzaak is de ERROR-lijn.
Voor elk van deze twee interrupt request-bronnen zal meestal dezelfde interrupt service routine worden gebruikt. In deze routine zal dan via een test van de statuslijnen worden bepaald wat er verder moet gebeuren.
In het hoofdprogramma (wat immers om de I/O-actie vroeg), zal er een programmadeel moeten zijn wat de error-handling voor zijn rekening neemt. Het I/O-apparaat kan nl. nooit beslissen wat er moet gebeuren. Dit is een taak van de gebruiker.
Samenvattend kunnen we zeggen dat de data-uitwisseling op interrupt-basis het mogelijk maakt om sneller te reageren op I/O-apparaten en daarnaast een mogelijkheid biedt om I/O-apparaten parallel te laten werken.
Als we gaan rekenen, blijkt al gauw dat de snelheid van datatransport toch volledig wordt bepaald door de snelheid waarmee instructies worden uitgevoerd. De processorsnelheid bepaalt dus globaal de datarate.
Meestal moeten er per te transporteren karakter in de orde van 10 à 100 instructies worden uitgevoerd. Hieruit volgt dat de datarate dan in de orde van maximaal 50.000 woorden per seconde zal liggen.
Voor een schijf- of bandeenheid is dit echter veel te weinig. Bij deze apparaten wordt gesproken van snelheden van een of enkele megabytes per seconde. De zojuist besproken techniek zal hierbij dan ook niet voldoen. We zullen dan b.v. op DMA moeten overgaan.

SAMENVATTING 7

21. In systemen met vectored interrupt worden alle interrupt request-lijnen door een interrupt controller tot een lijn gebundeld, die is aangesloten op de interrupt-ingang van de CPU.

22. Na het ontvangen van een interrupt request stuurt de CPU een INTA-signaal naar de interrupt controller. Deze plaatst dan de bij de betreffende interruptbron behorende interrupt vector (of soms direct het restart-adres) op de databus.

23. Bij vectored interrupt wordt de prioriteit niet software-matig maar hardware-matig (binnen de interrupt controller) geregeld.

24. Een voordeel van vectored interrupt t.o.v. polled interrupt is de korte reactietijd. Een nadeel is de extra benodigde hardware, dus de interrupt controller.

8. VOORBEELD 2

In deze les bespreken we een voorbeeld, waarin een printer op basis van interrupt I/O een blok data uit het geheugen van een microcomputer afdrukt.
Om u weer in staat te stellen deze data-uitwisseling te testen en te simuleren, gaan we weer uit van de SDK 85 met de uitbreidingsprint.
In fig.22 is het hardware-schema van het systeem weergegeven.

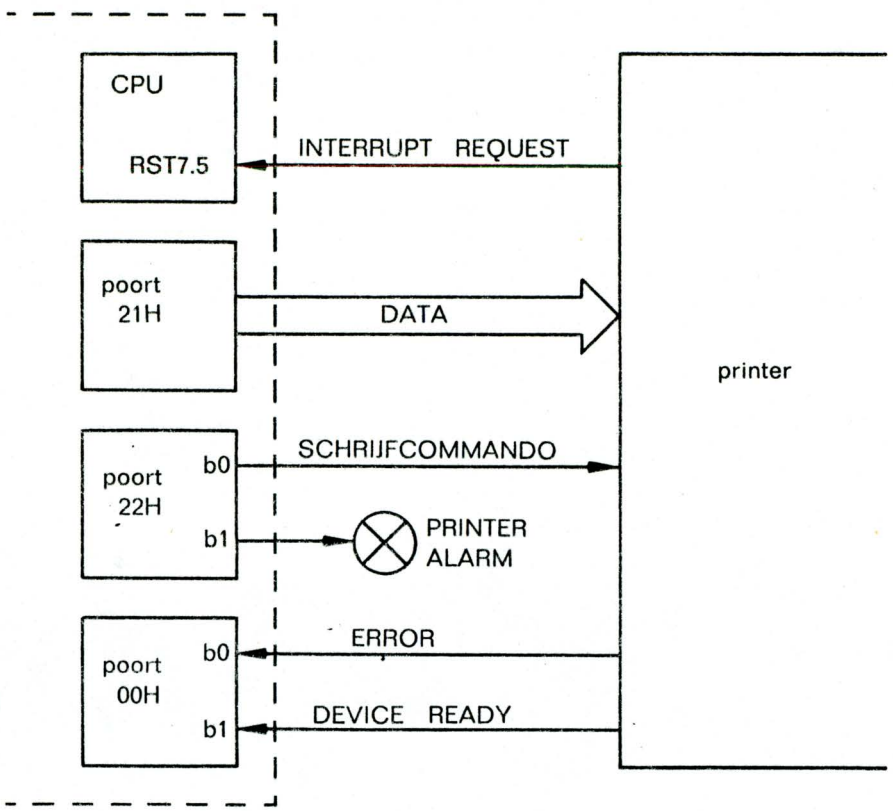


fig.22

Het totale systeem moet als volgt functioneren.
De CPU is bezig met het hoofdprogramma. Hierin wordt data bewerkt. Daarbij ontstaan 256 resultaten, die door het hoofdprogramma in de geheugenwoorden 2800_{16} t/m $28FF_{16}$ worden opgeslagen.
Direct nadat het eerste resultaat in adres 2800_{16} is geplaatst, wordt een pointer gevuld met de waarde 2800_{16}, het eerste resultaat wordt op de datalijnen naar de printer gezet en er wordt een schrijfcommando aan de printer afgegeven. De CPU gaat dan door met de bewerking van de data en het opslaan van de resultaten.

De bedoeling is, dat we een interrupt service routine ontwikkelen, waarin wordt onderzocht door welke oorzaak de interrupt request van de printer is ontstaan en welke actie hierop moet worden ondernomen. Er zijn wat dit betreft drie mogelijkheden, nl.:

a. DEVICE READY is 0 (= actief) geworden. Er is dus een vorige output-actie beëindigd. Voor een volgende actie moet de pointer met 1 worden verhoogd, nieuwe data op de datalijnen worden geplaatst en een actief schrijfcommando worden afgegeven.
b. DEVICE READY is 1 geworden. De printer heeft de data overgenomen en is bezig deze af te drukken. Het schrijfcommando kan worden weggenomen.
c. Er is een error condition (ERROR = 1).
Het gehele systeem moet dan worden gereset. De lamp PRINTER ALARM moet gaan branden.

Uit deze 3 punten blijkt, welke afspraken er gelden voor het actief zijn van de status- en besturingssignalen, nl.

SCHRIJFCOMMANDO:	1 = actief
DEVICE READY:	0 = actief
ERROR:	1 = actief

Uit bovenstaande beschouwing volgt het stroomdiagram voor de interrupt service routine "PRINT". Dit is in fig.23 weergegeven.

Voordat we dit stroomdiagram in een programma kunnen omzetten, moeten we de overdrachtsparameters vastleggen.
In dit geval is dit alleen de plaats van de eerst volgend uit te voeren data. Dit is een van de geheugenadressen 2800_{16} t/m $28FF_{16}$.
Het juiste adres wordt aangegeven door de pointer. Hiervoor kiezen we b.v. registerpaar D,E. Het programma wordt dan dat van fig.24.

Om deze interrupt service routine te kunnen testen, moet er een hoofd-programma worden ontwikkeld, waarin de resultaten voor de adressen 2800_{16} t/m $28FF_{16}$ ontstaan en dat kan worden onderbroken door een interrupt request op de RST7.5-ingang van de CPU.

In dit hoofdprogramma moet achtereenvolgens
a. het systeem worden geïnitialiseerd.
b. worden getest of de printer is aangesloten en ingeschakeld.
c. het eerste resultaat worden berekend en de eerste schrijfactie worden gestart.
d. de volgende 255 resultaten worden berekend.
e. verder worden gegaan met overige door de programmeur bepaalde bewerkingen.

Om het programma eenvoudig te houden, laten we de resultaten ontstaan door de inhoud van een register steeds met 1 te verhogen.
Als dit 256 maal is gebeurd, en alle resultaten op de adressen 2800_{16} t/m $28FF_{16}$ zijn geplaatst, laten we de CPU in een oneindige programmalus rondgaan. Deze lus kan dan steeds door interrupt requests worden onderbroken, totdat alle resultaten zijn uitgevoerd.
Het stroomdiagram voor dit programma is in fig.25 weergegeven.

Voordat we dit stroomdiagram in een programma kunnen omzetten, moeten we eerst weer enkele machinegerichte details vastleggen. In dit geval is de vraag welke locaties we voor REGISTER, TELLER en POINTER reserveren.

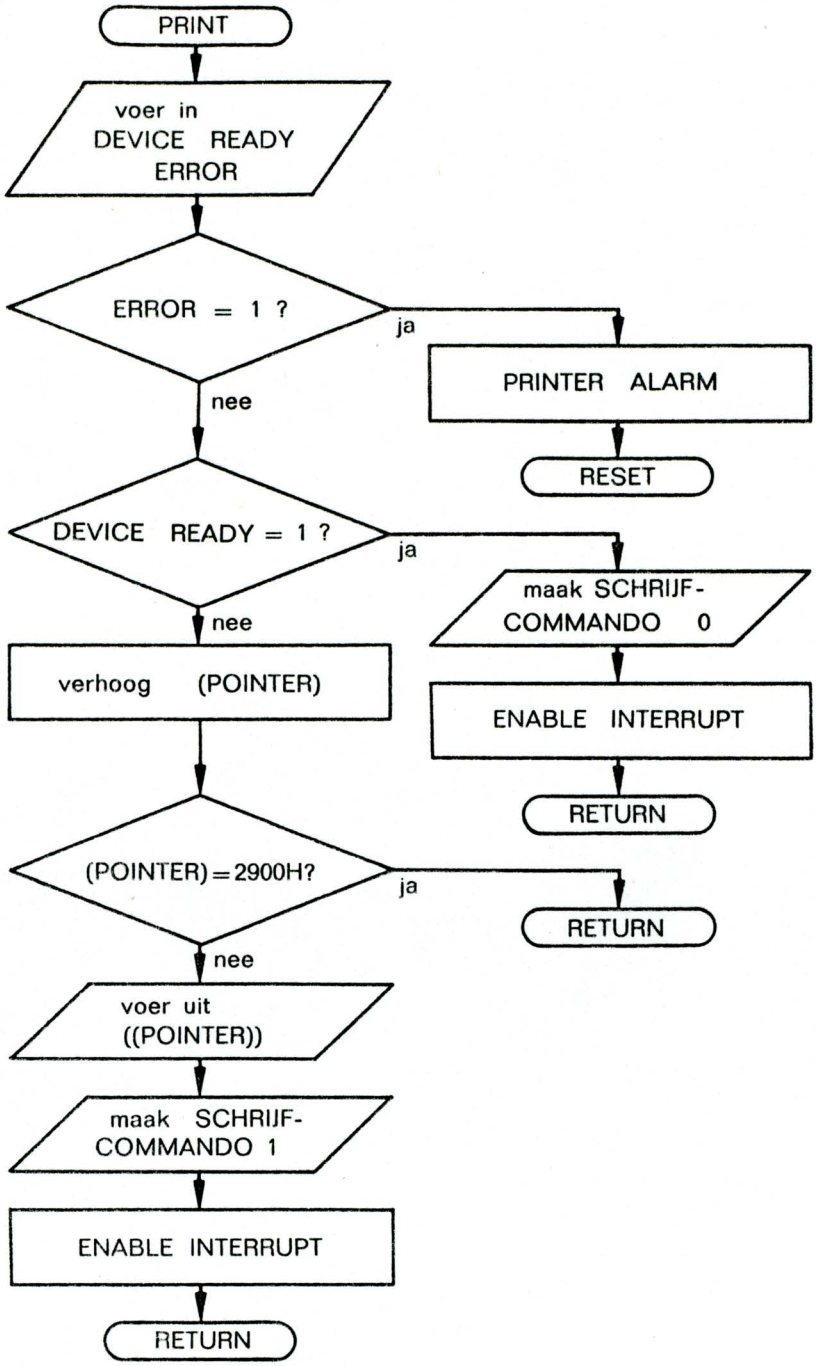

PRINT
voer in
DEVICE READY
ERROR
ERROR = 1 ?
ja
PRINTER ALARM
RESET
nee
DEVICE READY = 1 ?
ja
maak SCHRIJF-
COMMANDO 0
ENABLE INTERRUPT
RETURN
nee
verhoog (POINTER)
(POINTER) = 2900H?
ja
RETURN
nee
voer uit
((POINTER))
maak SCHRIJF-
COMMANDO 1
ENABLE INTERRUPT
RETURN

fig.23

```
                        ORG   2000H
2000 DB 00  PRINT:      IN    00H
2002 0F                 RRC                 ;TEST ERROR
2003 D2 0D 20           JNC   PRT1          ;SPRING ALS ERROR ≠ 1
2006 3E 02              MVI   A,02H         ;PRINTER
2008 D3 22              OUT   22H           ; ALARM
200A C3 00 00           JMP   0000H         ;RESET
200D 0F             PRT1:   RRC             ;TEST DEVICE READY
200E D2 16 20           JNC   PRT2          ;SPRING ALS READY ≠ 1
2011 AF                 XRA   A             ;MAAK SCHRIJF-
2012 D3 22              OUT   22H           ; COMMANDO 0
2014 FB                 EI                  ;VERVOLG
2015 C9                 RET                 ; SCHRIJFACTIE
2016 1C             PRT2:   INR   E         ;VERHOOG POINTER
2017 C8                 RZ                  ;RETURN NA 256 ACTIES
2018 1A                 LDAX  D             ;VOER UIT
2019 D3 21              OUT   21H           ; ((POINTER))
201B 3E 01              MVI   A,01H         ;MAAK SCHRIJF-
201D D3 22              OUT   22H           ; COMMANDO 1
201F 3E 1F              EI/ MVI A,1FH       ;START NIEUWE
2021 30                 RET SIM             ; SCHRIJFACTIE
2022 3E 0B              END MVI A,0BH
2024 30                     SIM
2025 FB                     EI
2026 C9                     RET
                            END
```

fig.24

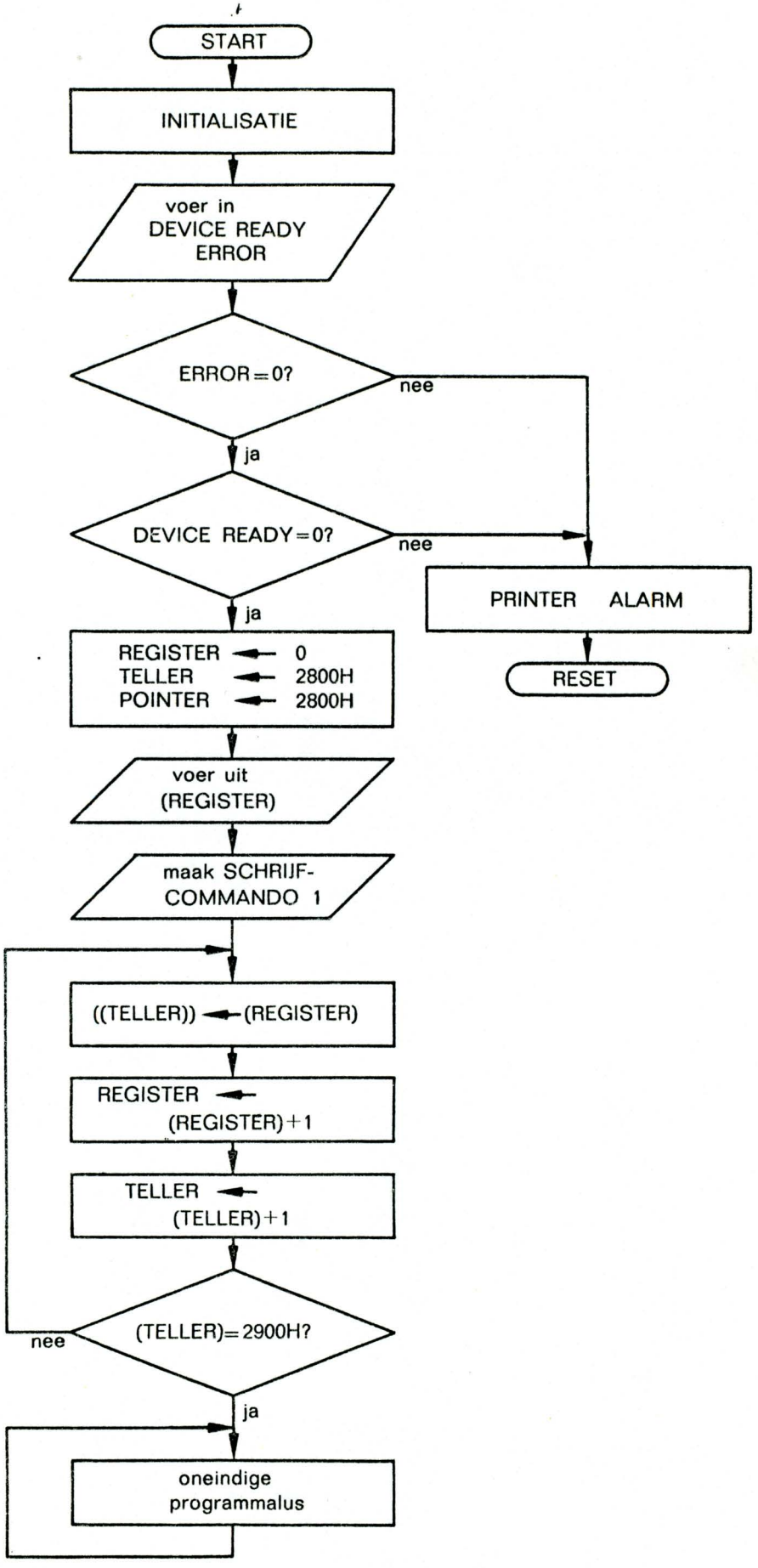

START
INITIALISATIE
voer in
DEVICE READY
ERROR
ERROR=0?
nee
ja
DEVICE READY=0?
nee
ja
PRINTER ALARM
RESET
REGISTER ← 0
TELLER ← 2800H
POINTER ← 2800H
voer uit
(REGISTER)
maak SCHRIJF-
COMMANDO 1
((TELLER)) ← (REGISTER)
REGISTER ←
(REGISTER)+1
TELLER ←
(TELLER)+1
(TELLER)=2900H?
nee
ja
oneindige
programmalus

fig.25

```
                        ORG   2030H
                PRINT   EQU   2000H
                RESET   EQU   0000H
                PRAL    EQU   02H        ;PRINTER ALARM
                MASK    EQU   0BH        ;UNMASK RST7.5
2030  3E 03             MVI   A,03H      ; INITALISEREN.
2032  D3 20             OUT   20H        ;POORT 21H = OUTPUT
2034  32 FF20           STA   20FFH      ;POORT 22H = OUTPUT
2037  AF          INI.REG.  XRA   A
2038  D3 02             OUT   02H        ;POORT 00H = INPUT
203A  03 03             OUT   03H        ;POORT 01H = INPUT
203C  31 C2 20          LXI   SP,20C2H   ; INITIALISEREN STACKPOINTER.
203F  3E 0B             MVI   A,MASK     ;ENABLE (INITIALISEREN)
2041  30           INT REQ  SIM          ; RST7.5
2042  FB                    EI           ; ENABLE INTERRUPT.
2043  DB 00                 IN    00H    ;
2045  0F                    RRC          ;TEST ERROR (Rotate Right carry)
2046  DA 62 20              JC    PRT3     ;SPRING ALS ERROR ≠ 0
2049  0F                    RRC          ;TEST DEVICE READY
204A  DA 62 20              JC    PRT3     ;SPRING ALS READY ≠ 0
204D  21 00 28              LXI   H,2800H  ;INIT TELLER
2050  11 00 28              LXI   D,2800H  ;INIT POINTER
2053  7D                   MOV   A,L      ;VOER UIT
2054  D3 21                 OUT   21H      ;  (REGISTER)
2056  3E 01                 MVI   A,01H    ;MAAK SCHRIJF- (INITIALISEREN)
2058  D3 22                 OUT   22H      ;  COMMANDO 1
205A  75          PRT4:     MOV   M,L      ;STORE DATA
205B  2C                    INR   L        ;(TELLER)+1  L → L+1
205C  C2 5A 20              JNZ   PRT4     ; SPRING NAAR LABEL "PRT4".
205F  C3 5F 20    LUS:      JMP   LUS      ;ONEINDIGE LUS
2062  3E 02       PRT3:    MVI   A,PRAL   ;PRINTER (INITIALISEREN)
2064  D3 22                 OUT   22H      ;  ALARM
2066  C3 00 00              JMP   RESET    ; SPRING NAAR LABEL "RESET".
                            END            ; TERUG NAAR MONITORPROGRAMMA
```

fig.26

Vraag 9: Voor POINTER is registerpaar ..... gereserveerd.
Voor TELLER mogen we wel/niet hetzelfde registerpaar gebruiken.

POINTER is een adresaanwijzer volgens welke in de interrupt service routine resultaten worden uitgevoerd. Hiervoor hebben we registerpaar D,E al gereserveerd.

TELLER is een adresaanwijzer volgens welke in het hoofdprogramma de berekende resultaten in het geheugen worden opgeslagen. Aangezien het berekenen en uitvoeren van de resultaten niet met dezelfde snelheid gebeurt, mogen we voor TELLER dus niet hetzelfde registerpaar D,E gebruiken. We kiezen b.v. registerpaar H,L.

Vraag 10: Voor REGISTER mogen we wel/niet register L gebruiken.

Omdat in dit specifieke voorbeeld de inhoud van REGISTER steeds gelijk is aan de low order byte van TELLER, is de meest eenvoudige oplossing om REGISTER in register L onder te brengen.

In fig.26 is dan het bronprogramma weergegeven. Ga dit voor uzelf na.

Om het programma en de interrupt service routine te kunnen testen, dient u ze eerst in machine-taal om te zetten en in het geheugen te plaatsen.

Daarna moet u, op eenzelfde wijze als in voorbeeld 1 is gedaan, de printer simuleren.
U kunt dan b.v. weer een lijstje (fig.27) naast de schakelaars en de LED's op de uitbreidingsprint leggen.

Door het indrukken van de toets "VECT INTR" kunt u een interrupt request RST7.5 genereren. Om de gevolgen van contactdender te voorkomen, moet u de EI- en de RET-instructies aan het eind van de interrupt service routine vervangen door:

```
3E 1F  MVI A,1FH    ;RESET
30     SIM          ; RST7.5
3E 0B  MVI A,0BH    ;ENABLE
30     SIM          ; RST7.5
FB     EI
C9     RET
```

(Zie paragraaf 7 van de les "Digitale klok".)

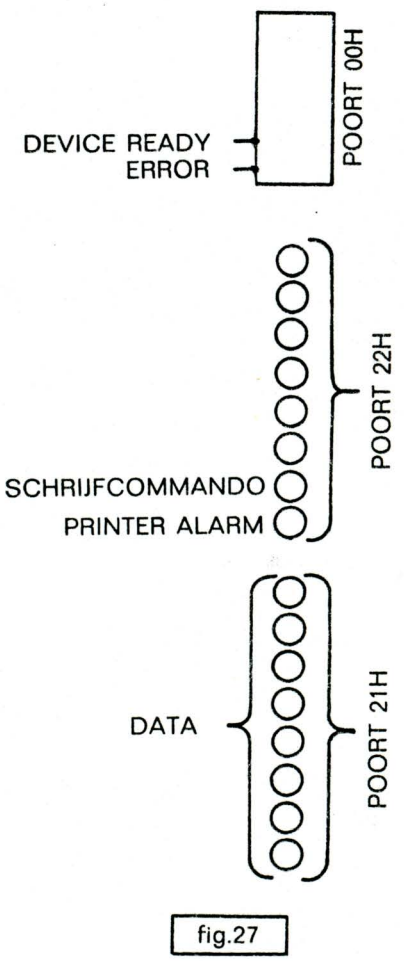


fig.27

Als u de printer op de juiste wijze simuleert, zal op de LED's van poort 21_{16} een binair telpatroon ontstaan.
Als u tussentijds een actief ERROR-signaal aanbiedt, wordt naar adres 0000_{16} gesprongen. Op de 7-segment displays verschijnt de tekst "- 80 85". U dient dan het programma opnieuw op adres 2030_{16} te starten.