Ontwerp van een microprogrammeerbaar computersysteem

Jacobs, J.W.M.; Raedts, P.M.

Award date:
1981

TECHNISCHE HOGESCHOOL EINDHOVEN

AFDELING DER ELECTROTECHNIEK

VAKGROEP DIGITALE SYSTEMEN (EB)

Ontwerp van een Microprogram-meerbaar Computersysteem.

Rapport van het afstudeerwerk, uitgevoerd in de periode october 1980 t/m augustus 1981 door:

J. Jacobs

P. Raedts

Afstudeerhoogleraar : Prof. Ir. A. Heetman

Coaches : Ir. M. Stevens

Ir. J. Kemper

Samenvatting

In dit rapport wordt het ontwerp van een microprogrammeerbaar computersysteem beschreven. Enkele onderdelen hiervan, met name het reken-orgaan en een geheugenkaart, zijn vrij ver uitgewerkt. Andere onderdelen behoeven nog verdere ontwikkeling. Het rekenorgaan staat, wat de hardware betreft, vele getalrepresentaties toe. Van deze mogelijkheden is alleen een klein Floating Point pakket in firmware gerealiseerd.

Inhoudsopgave

Inleiding

Microprogrammeren, een techniek die o.a. gebruikt kan worden bij het bouwen van computers en controllers, is de laatste jaren sterk in belangstelling gestegen. Deze toename in belangstelling is voor een groot gedeelte te danken aan het beschikbaar komen van geintegreerde circuits waarvan de functies microprogrammeerbaar zijn. In feite komt het hierop neer: Het overhevelen van "hardware"-taken naar het "software"-domein. Het grote voordeel is de flexibiliteit van software; daar tegenover staat een tragere response in vergelijking met een systeem waarvan de functies in hardware zijn uitgevoerd. In het algemeen geldt dat een microprogrammeerbaar systeem gestructureerder is van opzet dan een systeem dat is opgebouwd door middel van "wilde logica".

Binnen de vakgroep EB werd medio augustus 1980 besloten een project op te starten, dat tot doel had een microprogrammeerbaar computersysteem te ontwikkelen. Hiertoe werd een tweetal studenten aangetrokken, de schrijvers van het verslag dat voor U ligt, die dit project als gezamelijke afstudeeropdracht kregen. Door het ontbreken van de benodigde financiele middelen werd het echter onmogelijk het systeem te bouwen. Hierdoor kwam de nadruk volledig op het ontwerp te liggen.

Het resultaat van het afstudeerwerk is in vijf hoofdstukken vastgelegd. Na een inleiding in de microprogrammering en de globale systeemopzet, hoofdstuk 1, volgt het hardware ontwerp van de processor, hoofdstuk 2. Hierin komen ondermeer een beschrijving van de gebruikte microprogrammeerbare bouwstenen en het ontwerp van een Arithmetic and Logic Unit (ALU) aan bod. Om het ontwerp van de ALU te completeren is software geschre-

ven. Deze omvat programmatuur voor zowel een register machine als een stack machine, zie hoofdstuk 3. Een geheugenkaart, bedoeld om het systeem te voorzien van een behoorlijke geheugencapaciteit, besluit de reeks ontwerpen, zie hoofdstuk 4. In het laatste hoofdstuk, hoofdstuk 5, worden conclusies getrokken en enkele suggesties voor verbeteringen en uitbreidingen aan de hand gedaan.

1 Functionele Opbouw van de Processor

1.1 Inleiding

Wat is microprogrammeren?

Microprogrammeren is het programmeren van deelacties. Een microprogrammeerbare machine is een machine waarin een voorgeschreven sequentie van micro-instructies afgewerkt moet worden om een bepaalde machine-instructie uit te voeren. Het geheugen waarin de micro-instructies opgeslagen zijn, wordt het microprogrammageheugen of microgeheugen genoemd. Een micro-instructie bestaat normaliter uit twee belangrijke onderdelen:

1) Besturingsvector voor de te besturen hardware.

2) Besturingsvector die het adres van de volgende uit te voeren microinstructie bepaalt.

ad 1) De besturingsvector van de te besturen hardware kan uit meerdere micro-operaties bestaan. Als de te besturen hardware een ALU is dan omvatten deze micro-operaties o.a. zaken als:

- ALU source operand selection
- ALU function
- ALU destination
- Carry control
- Shift control
- etc.

ad 2) De besturingsvector die het adres van de volgende uit te voeren micro-instructie bepaalt kan uit de volgende onderdelen bestaan:

- Branch Address

- Sequencer control
- Condtion code multiplexer control
- etc.

De sequencer is een bouwsteen die de volgorde bepaalt van executie van micro-instructies die zijn opgeslagen in een microprogrammageheugen. In zijn eenvoudigste vorm bestaat de sequencer uit een adresteller die steeds de sequentieel volgende micro-instructie aanwijst. Aan een meer flexibele sequencer kan men de volgende eisen stellen:

Het moet mogelijk zijn het adres van de volgende uit te voeren micro-instructie te kiezen uit een van de volgende adressen:

- Het sequentieel volgende adres.
- Een extern aangeboden adres, bijv. een sprongadres, startadres of interruptadres.
- Een adres afkomstig van een stack. Dit om subroutines in microprogramma's mogelijk te maken.

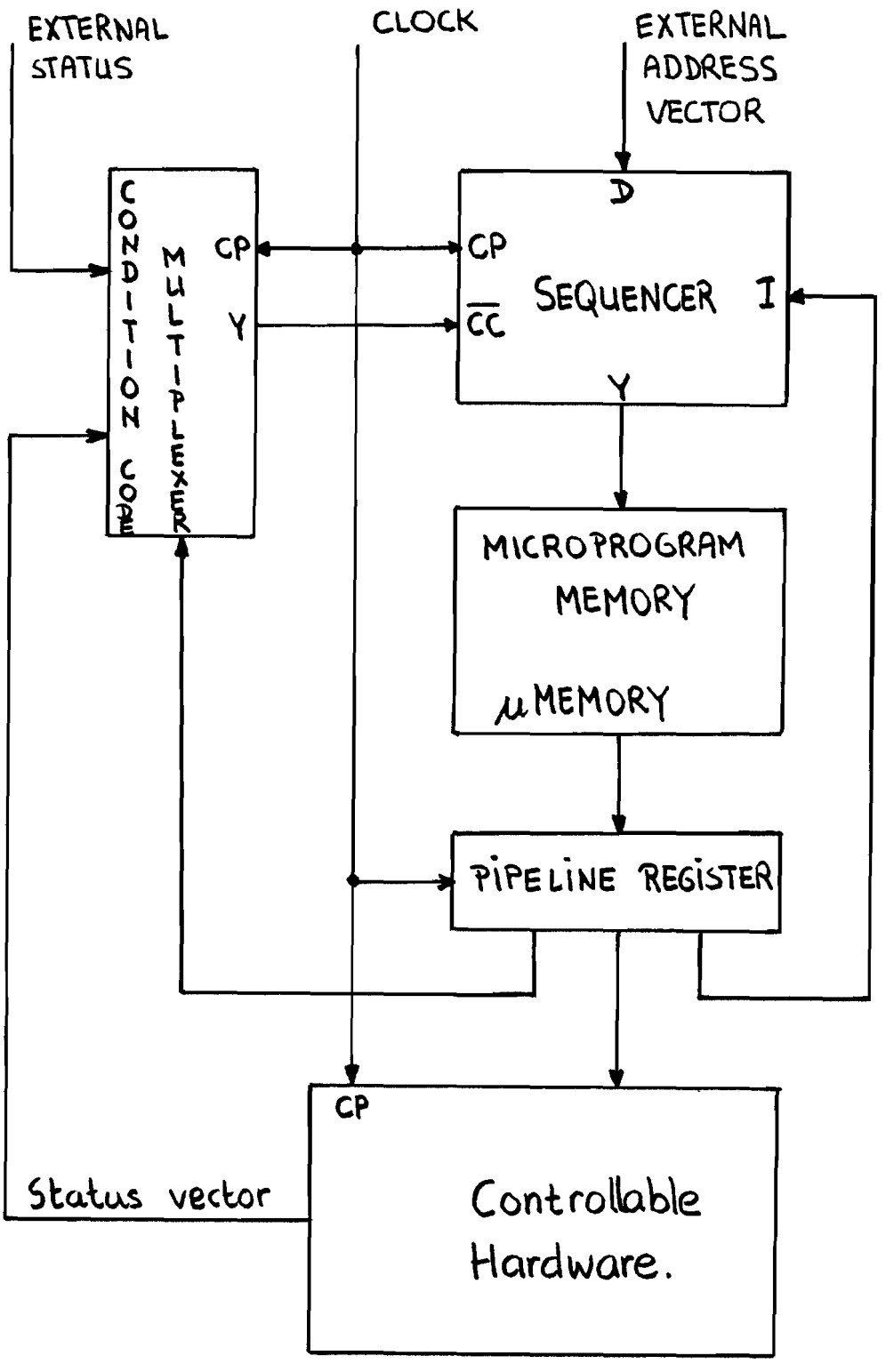
De keuze van het adres moet bepaald kunnen worden door:

- Een aan de sequencer opgedrukte instructie.
- Een selecteerbaar conditie code bit om conditionele opdrachten te realiseren.
- De stand van een teller om lussen in microprogramma's te kunnen realiseren.

Een sequencer die aan al deze eisen voldoet is de Am2910. De Am2910 Microprogram Controller wordt in paragraaf 1.3. uitgebreid besproken.

1.2 Algemene structuur van een microprogrammeerbaar systeem

De algemene structuur van een microprogrammeerbaar systeem is in figuur 1.1 weergegeven. De sequencer vormt het hart van het systeem en wordt bestuurd vanuit het Pipeline Register. Het Pipeline Register bevat altijd de micro-instructie die momentaan wordt uitgevoerd. Na elke klokslag genereert de sequencer een adres dat wordt aangeboden aan het microprogrammageheugen. Enige tijd later (de toegangstijd van het ROM) verschijnt het geselecteerde microwoord aan de uitgang van het geheugen om in het Pipeline Register geklokt te worden. Een gedeelte van het microwoord dient ter besturing van de sequencer en de conditie code multiplexer. De rest wordt gebruikt als besturingsvector voor de te besturen hardware. De sequencer kan via de multiplexer testen op een van de statusbits die door de hardware gegenereerd worden of op een externe conditie. Het Pipeline Register zorgt voor een scheiding van het systeem in twee gedeelten. Het ene gedeelte bepaalt het adres van het volgende te executeren microwoord en zet het betreffende microwoord klaar om in het Pipeline Register te worden geklokt ("fetch"). Het andere gedeelte verwerkt de instructievector uit het Pipeline Register betreffende de te besturen hardware en vormt een statusvector die in een Status Register wordt opgeslagen ("execute"). Het Pipeline Register en het Status Register dragen er toe bij dat de "fetch"- en "execute"-acties parallel uitgevoerd kunnen worden. De actie, die de langste tijdsduur heeft, bepaalt de lengte van de klokcyclus. In een non-pipelined systeem moeten deze acties sequentieel worden uitgevoerd. In een pipelined systeem wordt een micro-instructie dus sneller verwerkt dan in een non-pipelined systeem.



ALGEMENE STRUCTUUR VAN EEN MICROPROGRAMMEERBAAR SYSTEEM.

figuur 1.1

1.3 De Am2910 Microprogram Controller

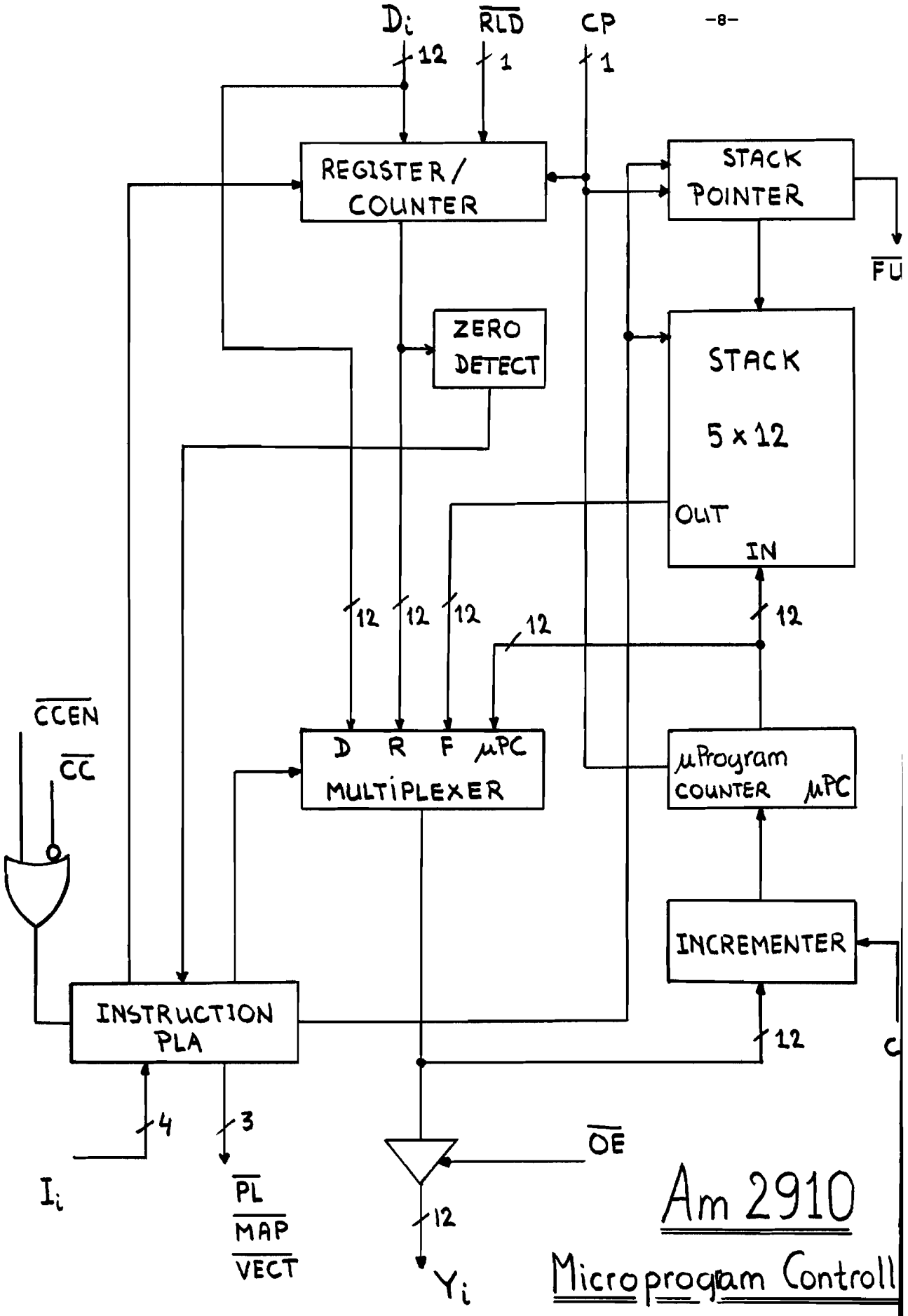
De Am2910 is een address-sequencer die ontworpen is om de volgorde van executie van micro-instructies, die in een micro-programmageheugen zijn opgeslagen, te verzorgen. In figuur 1.2 is de functionele opbouw van de Am2910 schematisch weergegeven. Door middel van een multiplexer selecteert de Am2910 gedurende elke micro-instructie een adres, ter breedte van 12 bits, uit een van de volgende vier bronnen:

1) Microprogram Counter Register
2) Top of Stack
3) External Input D
4) Register/Counter

ad 1) Het Microprogram Counter Register (uPC) bestaat uit een register en een incrementer. De ingang van de incrementer is verbonden met de adres-uitgang Y. Na elke klokcyclus wordt het register geladen met de som van de momentane waarde van de Y-output en de carry-in (CI) van de incrementer. Er geldt: uPC := Y + CI.

ad 2) De 2910 bevat een stack van vijf woorden diep. De stack kan men gebruiken om terugkeeradressen op te slaan, indien men een subroutine op microprogramma-niveau wenst uit te voeren. De Stack Pointer, die als up/down counter is uitgevoerd, wijst altijd naar het laatste woord dat op de stack is weggeschreven. Hier-door kan men een adres van de stack lezen zonder eerst een pop-operatie uit te voeren. Het statussig-naal FULL/ geeft aan of de stack vol is. Een slash (/) achter een besturingssignaal geeft aan dat het signaal "laag" actief is.

ad 3) De derde bron is de externe input D. Deze bron wordt



Am 2910

Microprogram Controll

figuur 1.2

gebruikt om sprongadressen op te drukken. Deze adressen kunnen een verschillende oorsprong hebben. Hiertoe genereert de 2910 een drietal enable-signalen, waarmee externe bronnen geselecteerd worden. Deze drie signalen zijn achtereenvolgens PL/, MAP/ en VECT/. Ze worden actief door een bepaalde instructie aan de 2910 op te drukken. Slechts een van de drie bronnen mag tegelijkertijd actief zijn daar ze allen data leveren aan dezelfde input D.

- PL/ : Pipeline Address Enable
  Selecteert bron nr. 1 als input, dit is normaliter een gedeelte van het Pipeline Register. Hierin staat een sprongadres of de initiele waarde van een teller.
- MAP/ : Map Address Enable
  Selecteert bron nr. 2 als input, dit is normaliter een Mapping PROM of Mapping FPLA. Hierin staan de startadressen van opdrachten die het systeem kan uitvoeren.
- VECT/ : Vector Address Enable
  Selecteert bron nr. 3 als input, bijv. een PROM dat startadressen bevat van Interrupt Service Routines.

ad 4) De vierde bron is de Register/Counter. Dit register kan onder instructie-besturing of m.b.v. het besturingssignaal RLD/ (Register Load) geladen worden met de data die zich op de D-input bevindt. Met behulp van dit register kan men een programmalus opzetten om een instructie meerdere malen uit te voeren. Afbreken van de lus kan gebeuren doordat de teller naar nul is geteld of doordat de CC/ (Condition Code) input "laag" is.

Alle datapaden in de 2910 zijn 12 bits breed. De maximale adresseerbare geheugenruimte bedraagt derhalve 4096 microwoorden (4K). Met behulp van instructievector I (I_3 t/m I_0) selecteert men een van 16 mogelijke instructies die de 2910 kan uitvoeren. In de literatuur (AM29,MICK) worden deze instructies allen uitgebreid toegelicht. Negen van deze instructies zijn conditionele instructies, waarin getest wordt op het conditie code bit (CC/). Als men het besturingssignaal CCEN/ (Condition Code Enable) "hoog" houdt, dan wordt de aangeboden conditie code genegeerd en reageert de 2910 alsof de conditie code input "laag" is (condition true). Door voor de CC/ input van de 2910 een of meerdere multiplexers te plaatsen (bijv. de Am2922) kan het aantal conditie code bits waarop men wil kunnen testen naar believen uitgebreid worden. Voor een meer volledige beschrijving van de Am2910 zie literatuur (AM29,MICK,JACO).

Naast de Am2910 zijn er in de Am2900 familie ook andere sequencers opgenomen zoals de Am2909 en de Am2911. Dit zijn bit-slice sequencers met datapaden ter breedte van 4 bits. Door meerdere slices in cascade te schakelen kan men een groter microprogrammageheugen adresseren. In tegenstelling tot de 2910 kennen de 2909 en de 2911 geen eigen instructieset. Men heeft hier dus nog de vrijheid om een eigen instructieset te implementeren. Met behulp van de Am29811A (Next Address Control Unit) en een teller kan men nagenoeg de hele 2910 instructieset implementeren. Resumerend kan men stellen dat de 2909/2911 boven de 2910 te verkiezen is in een van de volgende gevallen:

- Het te adresseren microprogrammageheugen bevat meer dan 4K microwoorden.
- De instructieset van de 2910 biedt niet voldoende variatie, men wenst complexere instructies te kunnen gebruiken.

1.4 Architectuur van het systeem

Enkele uitgangspunten bij het ontwerp van de architectuur van het microprogrammeerbare processorsysteem waren:

- Deel het systeen op in functionele eenheden, die elk een eigen specifieke taak hebben.

- Geef elk van deze functionele eenheden een eigen onafhankelijke besturing. In het totale systeem zien we dus een gedistribueerde microbesturing. Elke functionele eenheid heeft zijn eigen sequencer met een eigen microprogramma.

- Laat de functionele eenheden op asynchrone basis met elkaar gegevens uitwisselen. Dit houdt in dat elke eenheid met zijn eigen optimale klokfrequentie kan werken, onafhankelijk van andere eenheden.

- Kies een zodanige structuur dat het gemakkelijk mogelijk is het systeem uit te breiden tot een multi-processor systeem (meerdere reken-processoren).

- Het is de bedoeling dat het systeem in de toekomst o.a. wordt gebruikt als APL machine. Hierbij wordt gedacht aan een adresseringsmechanisme voor het Main Memory dat speciaal op APL is afgestemd. Het is dus wenselijk ervoor te zorgen dat het adresseringsmechanisme voor het Main Memory op eenvoudige wijze aangepast kan worden.

- Als systeembus wordt uitgegaan van de Multibus. De Multibus werd door Intel ontwikkeld om te worden toegepast in microprocessorsystemen. Het microprogrammeerbare systeem vormt, naast andere gebruikers zoals de diverse I/O apparatuur, een van de gebruikers van die bus. Om te voorkomen dat deze bus een bottle-neck wordt in het microprogrammeerbare systeem dienen er enkele maatregelen

genomen te worden.

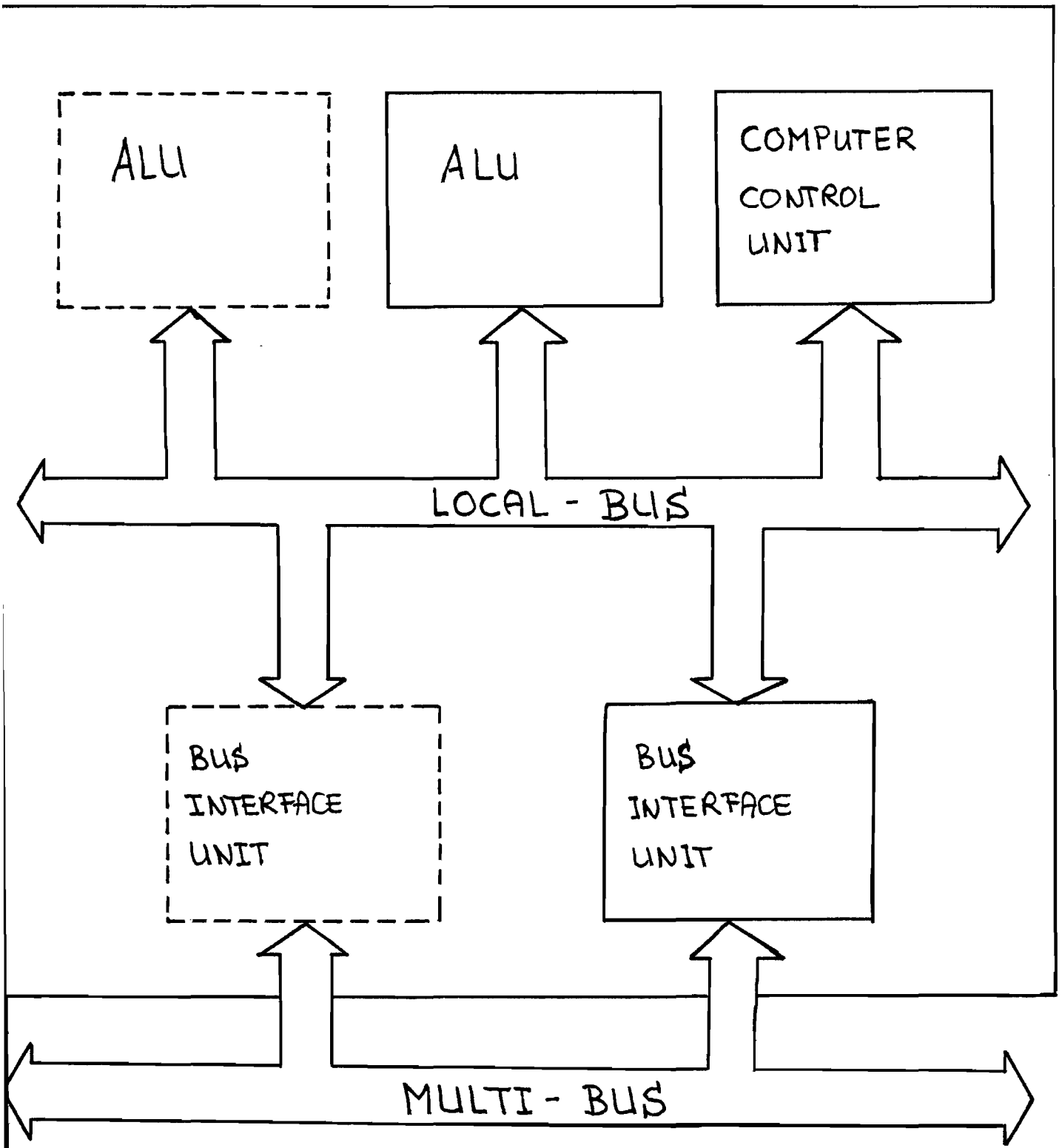
Aan de hand van de gestelde eisen zijn we gekomen tot de architectuur zoals die weergegeven is figuur 1.3.
We kunnen drie typen eenheden onderscheiden: de ALU (Arithmetic and Logic Unit) de CCU (Computer Control Unit) en de BIU (Bus Interface Unit). De Local Bus dient als communicatiekanaal tussen deze eenheden. De functionele taken van deze eenheden worden nu achtereenvolgens besproken.

1.4.1 Arithmetic and Logic Unit

In de ALU dient het echte rekenwerk te gebeuren. Het rekenen dient in verschillende formaten mogelijk te zijn, zowel integer als floating point. Deze ALU moet in staat zijn naast het standaard instructie repertoire van de diverse microprocessoren (zoals 8085, Z80, 6800, 8086, Z8000 en 68000) ook complexere instructies te kunnen verwerken zoals die bijvoorbeeld voorkomen in de 8087 (Numeric Data Processor) en de Am9511A (Arithmetic Processor). Zie lit.(IBSC,AMZ8). Hierbij valt onder andere te denken aan floating point bewerkingen zoals optellen, aftrekken, vermenigvuldigen, delen, goniometrische en logarithmische functies en worteltrekken.

Het ontwerp van de ALU werd gebaseerd op de bit-slice microprocessoren uit de Am2900 familie van Advanced Micro Devices. Met name de Am2903 bleek zeer geschikt te zijn, daar er in deze bouwsteen reeds voorzieningen zijn getroffen om complexe instructies te kunnen realiseren.

Zoals blijkt uit fig. 1.3 is het mogelijk meerdere ALU's parallel te schakelen op de Local Bus om aldus een multiprocessor systeem te realiseren.



ARCHITECTUUR VAN HET SYSTEEM

figuur 1.3

1.4.2 Bus Interface Unit

De Bus Interface Unit verzorgt via de Multibus het contact met de buitenwereld (Main Memory, I/O etc.). Om ervoor te zorgen dat de performance van het systeem niet al te zeer afhankelijk is van de belasting van de Multibus kan men een aantal maatregelen nemen. Hierbij valt bijvoorbeeld te denken aan:

- Instruction Queue
  De BIU kan instructies uit het Main Memory ophalen en deze in een Instruction Queue plaatsen. Hierdoor wordt een voorraad nog uit te voeren instructies opgebouwd.

- RAM of Associatief geheugen
  Hierin kan data worden geplaatst waarop tijdens de executie van het gebruikersprogramma veelvuldig access wordt gepleegd. Ook kan hierin data worden geplaatst die weggeschreven dient te worden naar het Main Memory, maar waarvoor nog geen toestemming verkregen is.

De BIU is ook de plaats waar alle adresberekeningen plaats vinden. Hier bevinden zich ook de registers waar de Stack Pointer en de Program Counter van het gebruikersprogramma zijn opgeslagen. Om deze adresberekeningen te kunnen doen is in de BIU een eenvoudige ALU nodig. Deze ALU hoeft alleen integer getallen te kunnen optellen en aftrekken. De breedte van de data paden in deze ALU moet 24 bits zijn daar het Main Memory met 24 adreslijnen geadresseerd dient te worden. Men zou deze ALU kunnen opbouwen met zes Am2901 bit-slice microprocessoren, die dan in cascade geschakeld dienen te worden.

Bij lees- en schrijfoperaties dient de BIU het berekende adres in het MAR (Memory Address Register) te plaatsen en een van de lees- of schrijfcommandolijnen van de Multibus te activeren. Bij het schrijven van data naar het Main Memory of een I/O device dient de BIU pariteitsbits bij zowel data als adres

te genereren, zie hoofdstuk 4. De data- en adresbytes dienen tesamen met hun pariteitsbits over de Multibus verzonden te worden. Alle met de Multibus verbonden modules controleren de data- en adresbytes die ze ontvangen. Wordt er ergens een fout geconstateerd, dan volgt er een interrupt naar de BIU. De BIU kan dan besluiten tot hertransmissie van dezelfde boodschap. Verdwijnt de fout niet na enige hertransmissies dan dient de CCU gewaarschuwd te worden. Ook bij een leesoperatie worden bij de adresbytes pariteitsbits gegenereerd. De ontvangen databytes worden op hun pariteit gecontroleerd. Als de BIU een fout constateert wordt de leesoperatie enige malen herhaald. Verdwijnt de fout niet dan dient wederom de CCU gewaarschuwd te worden.

De databus gedeelten van de Local Bus en de Multibus hebben niet dezelfde breedte. Hier dient de BIU voor een aanpassing te zorgen. Indien men alternatieve adresseringsmechanismen wenst in te voeren dan dient men de BIU aan te passen of meerdere BIU's in het systeem op te nemen.

1.4.3 Computer Control Unit

De Computer Control Unit bestuurt het totale systeem. De functie van de CCU zullen we aan de hand van een voorbeeld beschrijven. We gaan er in eerste instantie vanuit dat het systeem een ALU, een BIU en een CCU bevat. De CCU kent de instructieset van de ALU. Als de ALU klaar is met de uitvoering van een opdracht dan maakt hij zijn "Operation Complete" lijn actief. Als de CCU dit gezien heeft, dan haalt hij in de BIU een nieuwe machine-instructie op. Vervolgens wordt deze machine-instructie vertaald in een of meerdere instructies voor de ALU en/of de BIU. De instructie, bestemd voor de ALU, wordt op de Local Bus geplaatst en het "Operation Request" signaal voor de ALU wordt geactiveerd. Hierdoor weet de ALU dat er een

voor hem bestemde instructie op de Local Bus staat. Deze wordt binnen gehaald en het "Operation Complete" signaal wordt weggehaald. De ALU start vervolgens de executie van de instructie. De communicatie tussen de BIU en de CCU verloopt op soortgelijke wijze.

Andere taken van de CCU zijn het afhandelen van interrupts op systeemniveau en eventueel het verdelen van de werklast over verschillende ALU's en BIU's.

Alle communicatie tussen de units verloopt via de Local Bus. Indien meerdere ALU's en BIU's in het systeem worden opgenomen is het wenselijk dat elke unit een uniek adres krijgt en dat de Local Bus dus een adresbus krijgt. Wat betreft de Local Bus moet een keuze gemaakt uit twee gevallen:

Single Bus Master

In dit geval is de CCU altijd degene die de Local Bus onder controle heeft en dus ook alle transporten hierover initieert.

Een manier om datatransporten tussen de ALU en de BIU te laten verlopen is met een tussentijdse data opslag in de CCU. Deze methode heeft als nadeel dat een datatransport altijd in twee slagen moet gebeuren en dus extra tijd kost.

Een andere methode is dat de data wel rechtstreeks van BIU naar ALU gaat of omgekeerd, maar dat de CCU de bijbehorende adressen en besturingssignalen genereert. Het nadeel hierbij is, dat bij een transport drie eenheden tegelijkertijd betrokken zijn.

Multiple Bus Master

Bij deze methode kan elke unit master van de bus worden. Elke unit wordt in staat geacht de Local Bus te kunnen besturen. Hierdoor verlopen de datatransporten snel want nu gaan ze rechtstreeks van BIU naar ALU en omgekeerd.

Indien elke unit als gelijkwaardig wordt beschouwd dient er een busarbitrage mechanisme opgenomen te worden. De CCU kan eventueel deze arbitrage verzorgen.

Het afstudeerwerk heeft zich met name geconcentreerd op de Arithmetic and Logic Unit. Deze komt in de volgende hoofdstukken dan ook uitgebreid aan de orde. In hoofdstuk 2 wordt het hardware ontwerp van de ALU behandeld. De bijbehorende software (firmware) komt in hoofdstuk 3 aan de orde.

2 Het Hardware Ontwerp van de ALU

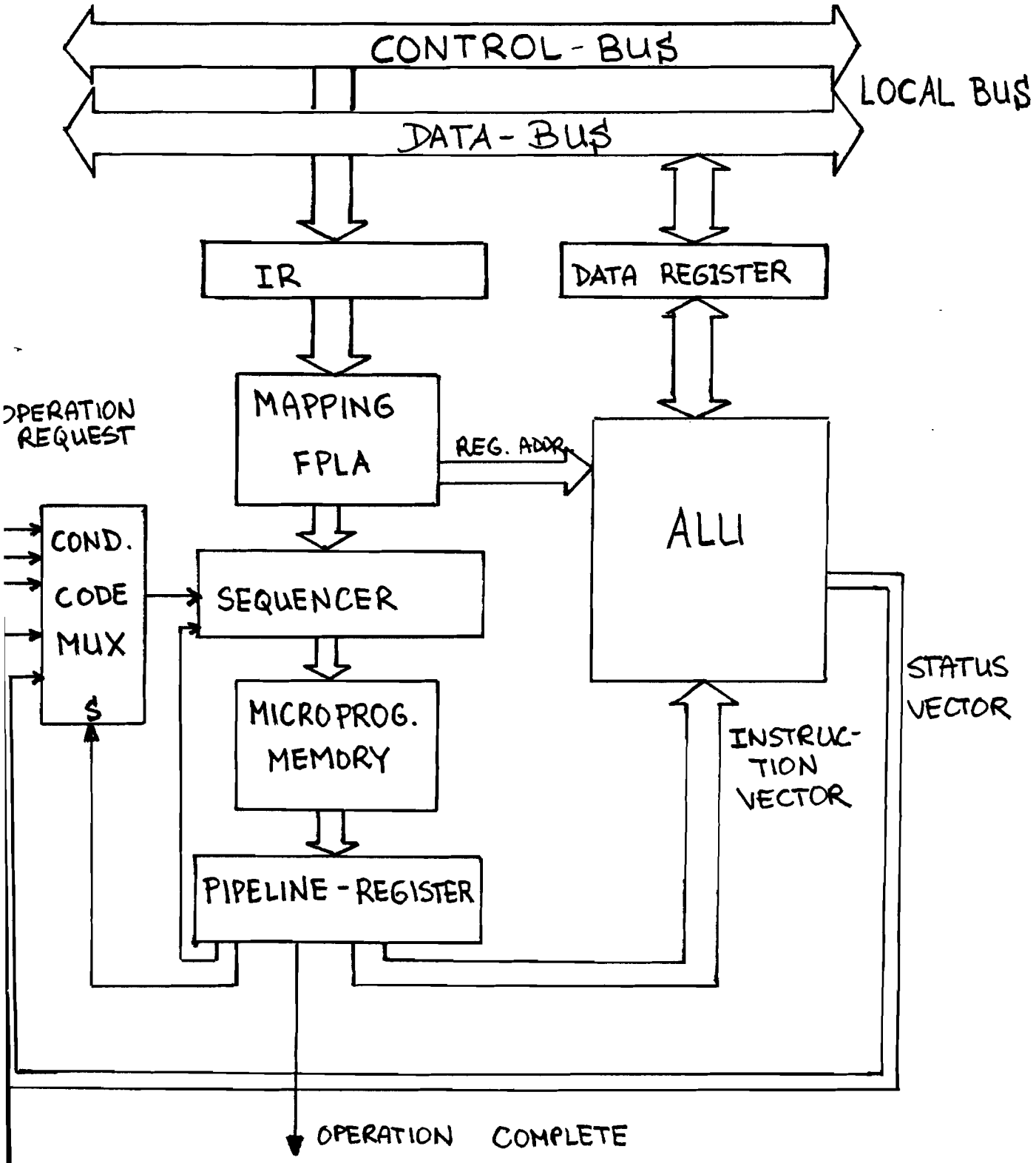
2.1 Inleiding

De ALU (Arithmetic and Logic Unit) is naast de CCU (Computer Control Unit) en de BIU (Bus Interface Unit) een van de drie hoofdbestanddelen van het computersysteem. In de ALU dient het echte rekenwerk verricht te worden. Adresberekeningen vinden niet hier doch in de BIU plaats. Besloten werd de ALU te ontwerpen met behulp van Am2900 bouwstenen. De Am2900-familie bestaat uit een aantal LSI circuits die ontworpen zijn voor gebruik in microprogrammeerbare computers en controllers. Elk van deze IC's is zodanig ontworpen dat de gebruiker een grote mate van vrijheid heeft in zijn ontwerp. Ze zijn dan ook zeer geschikt als bouwstenen voor een computer waarmee men bestaande machines wenst te emuleren. Het kunnen emuleren van deze processoren was een van de uitgangspunten bij het ontwerp van deze processor. Een aantal andere eisen die bij het ontwerp als uitgangspunt golden waren o.a. het mogelijk maken van rekenen in:

- 8 bits
- 16 bits
- 32 bits
- floating point

In figuur 2.1 zien we de architectuur van de microprogrammeerbare ALU weergeven. De algemene structuur van een microprogrammeerbaar systeem zoals dat behandeld werd in paragraaf 1.2 vinden we hier weer duidelijk terug. De ALU "black-box" bevat de Am2903 bit-slice microprocessoren met de benodigde randlogica. Deze black-box wordt in de navolgende paragrafen van dit hoofdstuk uitgebreid besproken. De belangrijkste input- en outputvectoren van deze black-box zijn:

- Datavector
- Adresvector


CONTROL-BUS
LOCAL BUS
DATA-BUS
IR
DATA REGISTER
MAPPING
FPLA
REG. ADDR
ALU
OPERATION
REQUEST
COND.
CODE
MUX
SEQUENCER
MICROPROG.
MEMORY
PIPELINE-REGISTER
STATUS
VECTOR
INSTRUC-
TION
VECTOR
OPERATION COMPLETE

figuur 2.1

- Instructievector
- Statusvector

Data uitwisseling met een andere zelfstandige unit verloopt via het Data Register over het data-bus gedeelte van de Local Bus. Dit Data Register kan worden gevuld vanaf of gelezen door de Local Bus. De ALU wordt bestuurd door middel van de instructievector. Deze instructievector is afkomstig uit het lokale Pipeline Register. Het Pipeline Register bevat het microwoord dat momenteel wordt uitgevoerd. Indien men de ALU een opdracht wil laten uitvoeren dan wordt deze opdracht vanaf de Local Bus in een Instruction Register geklokt. Een FPLA vertaalt deze opdracht in een startadres voor een microprogramma, dat deze opdracht kan uitvoeren. Het startadres wordt door de sequencer doorgegeven aan het microgeheugen. De FPLA verzorgt eveneens de adresvector voor de ALU-black-box. Tijdens uitvoering van het microprogramma kan getest worden op bits uit de statusvector. Afhankelijk van de uitslag van de test kan de sequencer beslissen of een conditionele sprong al dan niet wordt uitgevoerd. Nadat het microprogramma verwerkt is wordt het "Operation Complete" signaal actief opdat de CCU een nieuwe opdracht kan sturen.

2.2 Am2900 bouwstenen

Om de navolgende paragrafen goed te kunnen begrijpen is het noodzakelijk om een goed inzicht te hebben in de werking en de functie van een tweetal Am2900 bouwstenen: de Am2903 en de Am2904. Deze worden hiertoe nu in het kort beschreven. De 2903 en 2904 circuits zijn gerealiseerd m.b.v. de Low-Power Schottky TTL technologie. In de literatuur (AM29,MICK) vindt men een meer volledige beschrijving van deze bouwstenen.

2.2.1 Am2903 The Superslice

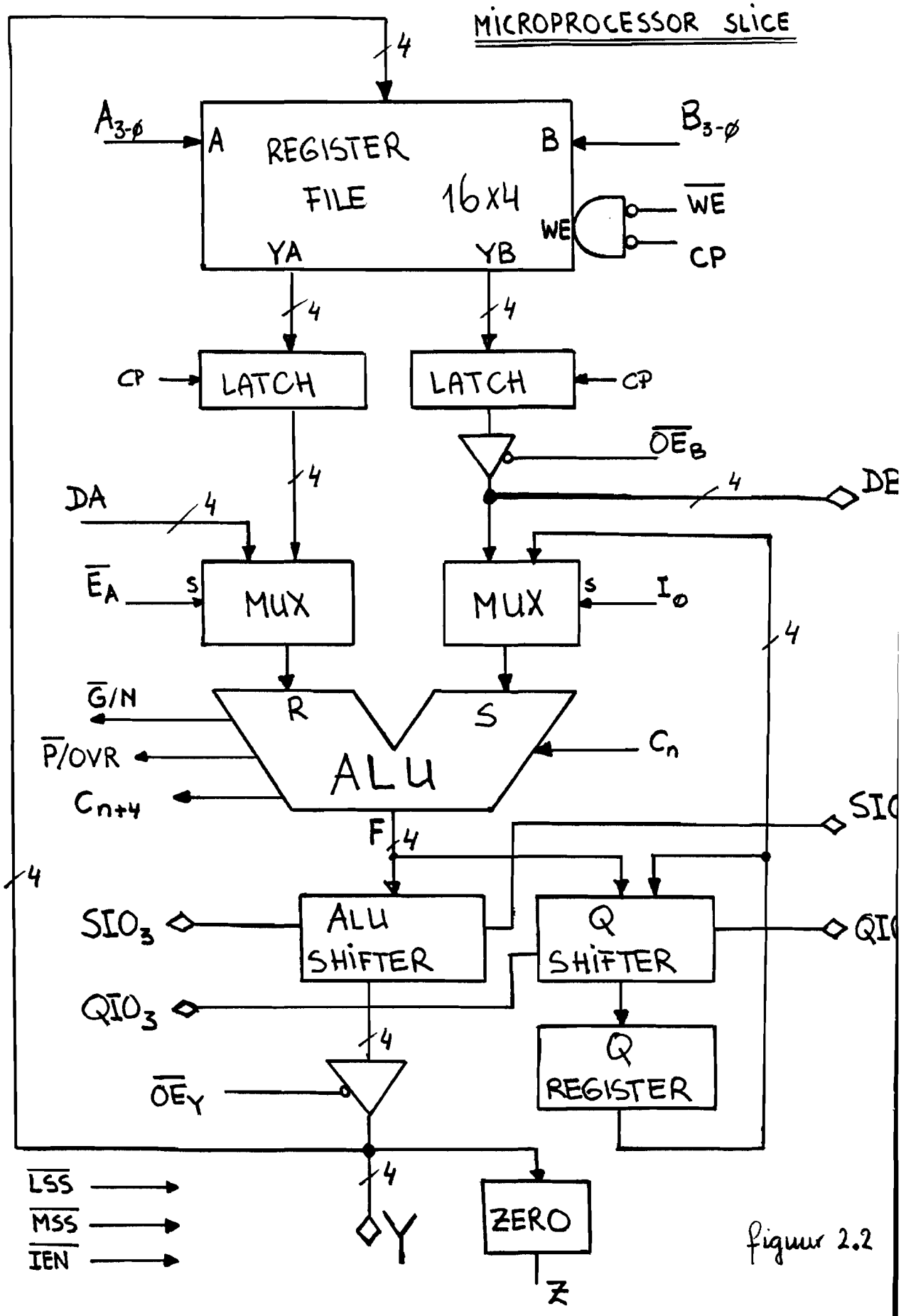
De Am2903 is een bit-slice microprocessor met datapaden ter breedte van vier bits. Zoals men kan zien in figuur 2.2 bevat de 2903 o.a. een dual-port RAM (16 woorden van 4 bits elk; latches op beide output-ports), een ALU, een ALU Shifter, een Q Register en een Q Shifter. Door meerdere 2903's in cascade te schakelen kan men een processor ontwerpen met een databus ter breedte van elk veelvoud van vier. In de navolgende bladzijden worden enkele aspecten van de 2903 nader toegelicht.

Dual-Port RAM

Men kan uit het dual-port RAM (de Register File) simultaan twee woorden lezen. Het woord geadresseerd met behulp van A-adreslijnen verschijnt op de YA output-port, evenzo verschijnt het woord geadresseerd m.b.v. de B-adreslijnen op de YB output-port. Indien de A- en B-adreslijnen gelijk zijn, verschijnt identieke data aan beide output-ports. De latches aan de output-ports zijn transparant gedurende de tijd dat het kloksignaal CP "hoog" is en ze houden de data vast gedurende de "laag"-tijd van CP. Met behulp van het stuursignaal $OE_B/$ kan men de data afkomstig van de YB output-port op de DB-bus plaatsen. De DB-bus is een bidirectioneel datapad waarvan men data kan lezen uit het dual-port RAM en waarover men data kan aanbieden aan de ALU. Schrijven in het RAM gebeurt altijd op het adres aangeboden via de B-adreslijnen. De te schrijven data is afkomstig van de ALU Shifter ($OE_Y/=0$) of van elders via de bidirectionele Y-bus ($OE_Y/=1$). Er wordt alleen geschreven als voldaan is aan de voorwaarden: WE/=0, CP=0 (WE : Write Enable). De WE/ input wordt in het algemeen verbonden met een stuursignaal, dat als schrijfsignaal voor de gehele processor dient.

Am 2903

MICROPROCESSOR SLICE



figuur 2.2

De Arithmetic & Logic Unit

De Am2903 ALU kent naast een aantal speciale functies zeven arithmetische operaties (optellen en aftrekken van twee operanden met verschillende varianten) en negen logische operaties (o.a. AND, OR, NOT, NAND, NOR, EXCLUSIVE OR & EXCLUSIVE NOR). Met behulp van de negen speciale functies kan men de volgende operaties implementeren:

- Single- & Double Length Normalization
- Two's Complement Division
- Unsigned Multiplication
- Two's Complement Multiplication
- Two's Complement - Sign/Magnitude Conversion
- Increment by One or Two

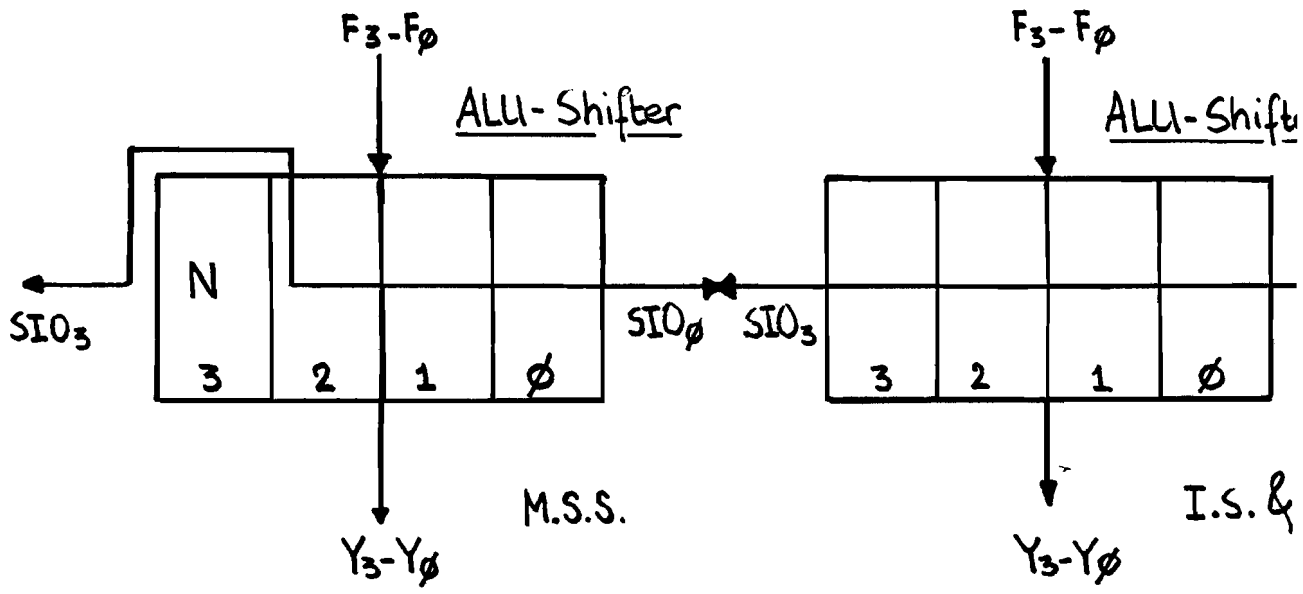
Door middel van een multiplexer voor de R-input van de ALU kan, met de stuurlijn $E_A/$, als ALU operand gekozen worden uit de externe DA input en de YA output-port van het RAM. De stuurlijnen I_0 en $OE_B/$ verrichten dezelfde functie voor ALU operand S en selecteren uit de YB output-port van het RAM, de externe DB-bus en het Q Register.

Bij diverse bewerkingen kan het nodig zijn de ALU een Carry-in aan te bieden. Hierin is voorzien met de C_n input. De 2903 genereert ook een Carry-out signaal, C_{n+4}.

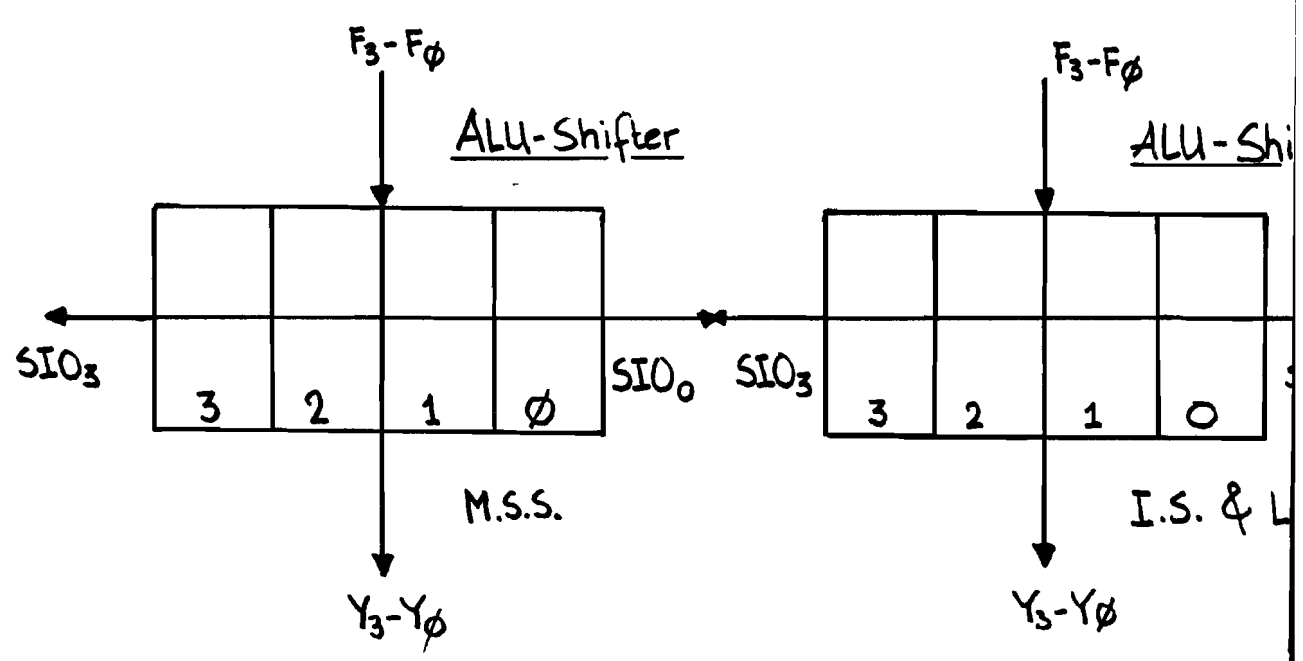
Indien een aantal 2903's in cascade geschakeld zijn dient elke slice gepositioneerd te worden. De positie van elke slice wordt vastgelegd met behulp van de LSS/- en de MSS/-lijnen. Een slice kan worden geprogrammeerd als:

MSS	(Most Significant Slice)	LSS/=1 ; MSS/=0.
IS	(Intermediate Slice)	LSS/=1 ; MSS/=1.
LSS	(Least Significant Slice)	LSS/=0 ; MSS/ is output.

Indien een slice als Intermediate of als Least Significant Slice is geprogrammeerd dan verschijnen op de G/,N en P/,OVR



Am 2903 Arithmetic Shift Path.



Am 2903 Logical Shift Path.

figuur 2.3

lijnen respectievelijk Carry Generate (G/) en Carry Propagate (P/), signalen die nodig zijn indien men een carry lookahead generator gebruikt. Was de slice echter als Most Significant Slice geprogrammeerd dan zouden via dezelfde lijnen respectievelijk de ALU statussignalen N en OVR zijn verschenen. N, negative, is het meest significante bit van de ALU output F en wordt meestal als tekenbit gebruikt. OVR, overflow, is in het algemeen een outputsignaal dat aangeeft dat een arithmetische operatie een resultaat heeft, dat buiten de grenzen van de 2's complement notatie valt. De outputpennen C_{n+4}, P/,OVR en G/,N kunnen afhankelijk van de opgedrukte instructie een verschillende functie hebben. Voor een exacte definitie van deze signalen zie (AM29,MICK).

De ALU Shifter

Afhankelijk van de aan de Am2903 opgedrukte instructie laat de ALU Shifter de F-outputs van de ALU door op een van de volgende manieren:

- non-shifted Y := F
- up-shifted Y := 2F
- down-shifted Y := F/2

Zowel arithmetische als logische schuifoperaties zijn mogelijk. In geval van een arithmetische schuifoperatie schuift men niet door de meest significante bitpositie (d.w.z. men laat het tekenbit staan). Bij een logische schuifoperatie wordt door alle bitposities heengeschoven, dus ook door de meest significante bitpositie. Per microcyclus wordt er ten hoogste over een bitpositie geschoven.

SIO_0 en SIO_3 zijn bidirectionele, seriele, I/O lijnen waarover men door meerdere slices kan schuiven. Zie figuur 2.3 In de literatuur (AM29,MICK) worden enkele andere mogelijkheden die de ALU Shifter nog te bieden heeft zoals "parity checking & generation" en "sign extend" uitgebreid besproken.

Q Register en Q Shifter.

Het Q Register is een vier bit register dat voornamelijk bedoeld is als hulpregister bij delen en vermenigvuldigen. De ALU output F kan al dan niet verschoven (m.b.v. de Q Shifter) in het Q Register geladen worden. De Q Shifter kan, in tegenstelling tot de ALU Shifter, alleen logische schuifoperaties uitvoeren (schuiven door alle bitposities). Hierbij worden QIO_3 en QIO_0 als I/O-lijnen gebruikt. Men kan schuiven over de dubbele lengte van een woord door QIO_3 van de "most significant slice" met SIO_0 van de "least significant slice" te verbinden. Als men rond wil schuiven, dan dient ook QIO_0 van LSS met SIO_3 van MSS verbonden te worden.

Nul Detectie

De Am2903 is voorzien van de Z (Zero) input/output pen. Bij de meeste 2903-instructies is Z als outputlijn geschakeld en kan gebruikt worden als nul detectie signaal van de Y-bus. De Z-pen is uitgevoerd als "open-collector output", hierdoor is het mogelijk een "wired-or" te maken met de Z-outputs van meerdere 2903's en zo met een lijn een nul dectectie over de hele breedte van de Y-bus te realiseren. Als men de speciale functies van de 2903 gebruikt, dan kan de Z-lijn ook als input fungeren bijvoorbeeld om het tekenbit door te geven aan de overige slices. Zie lit. (AM29,MICK).

Am2903 Besturingssignalen

Men kan de 2903 een bepaalde opdracht laten uitvoeren door de juiste instructievector op te drukken. De instructievector I heeft een breedte van negen bits: I_0 t/m I_8.

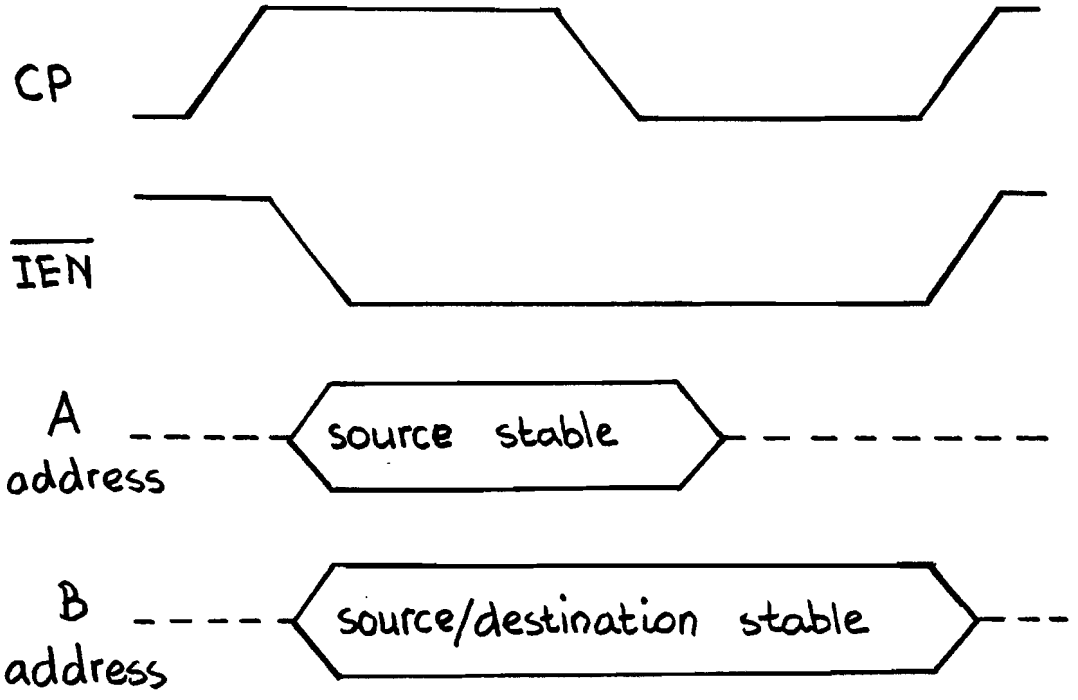
Als een slice als "least significant slice" is geprogrammeerd (LSS/=0) dan wordt de MSS/ pen van de betreffende slice

gepromoveerd tot WRITE/ output-pen. Deze WRITE/ output wordt "laag" als er een instructie wordt uitgevoerd waarbij data in de Register File moet worden weggeschreven. In het algemeen wordt dit WRITE/ signaal gebruikt als schrijf-opdracht voor de hele ALU. Hiertoe wordt de WRITE/ pen verbonden met de WE/ inputs van alle 2903's.

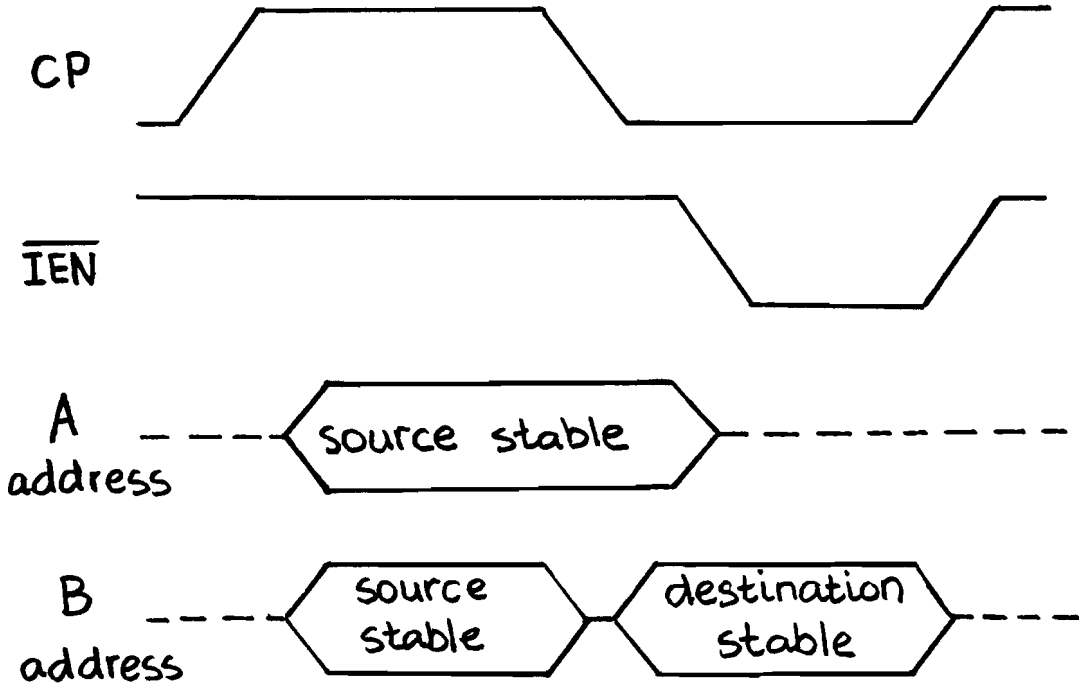
Een belangrijk besturingssignaal is het IEN/ signaal (Instruction Enable). Als de IEN/ input "laag" is, dan wordt de instructie opgedrukt door middel van I_8-I_0 normaal uitgevoerd. Maakt men echter de IEN/ input "hoog" dan heeft dit tot gevolg dat alle geheugen functies van de 2903 (met uitzondering van de Register File) bevroren worden. IEN/ "hoog" houden van een slice die als LSS geprogrammeerd is, heeft ook tot gevolg dat de WRITE/ output van die betreffende slice "hoog" gehouden wordt. Dat wil zeggen dat er geen schrijfpuls voor de Register File verschijnt. Indien nu de WRITE/ output van de LSS met de WE/ input van alle 2903's verbonden is, dan blijft de inhoud van de hele Register File ongewijzigd.

Men kan de IEN/ input bijvoorbeeld gebruiken om een "Compare" opdracht te realiseren. Door middel van instructievector I geeft men de opdracht om de inhoud van twee registers van elkaar af te trekken. Hierbij worden de status flags geset waarop men naderhand kan testen. Als IEN/ "0" is dan wordt de inhoud van het register, dat met het B-adres geaddresseerd was, overschreven met de ALU output F (als OE_Y/ "0" is). De oorspronkelijke inhoud van het register gaat verloren. Had men echter IEN/ "1" gemaakt dan zou de WRITE/ output niet "laag" zijn geworden. Er zou dus ook niets overschreven zijn, terwijl men toch de status flags van de aftrekking tot zijn beschikking heeft. Aldus heeft men een "Compare" opdracht gerealiseerd.

In het algemeen worden de 2903's in een twee-adres structuur gebruikt. Stel we hebben de opdracht ADD RA,RB waarbij RA het register is dat geadresseerd wordt met het A-adres en RB het register dat geadresseerd wordt met het B-adres. Het resultaat van de bewerking wordt in RB geplaatst, waardoor de oor-



TWO ADDRESS MODE



THREE ADDRESS MODE

figuur 2.4

spronkelijke inhoud van RB verloren gaat. Indien men de inhoud van dit register nog nodig heeft moet men dit eerst redden alvorens de opdracht uit te voeren.

Het is echter ook mogelijk de 2903's in een drie-adres structuur te gebruiken. Bij een drie-adres structuur ziet een soortgelijke opdracht er als volgt uit : ADD RA,RB,RC. Hier zijn RA en RB de source-registers en RC het destination-register. We kunnen dit realiseren door voor de B-adres inputs van de 2903's een multiplexer te plaatsen. Op de ingang van de multiplexer plaatst men de adreslijnen van RB en RC. De multiplexer laten we omschakelen op de neergaande flank van CP. Dit heeft tot gevolg dat in het eerste gedeelte van de klokperiode (CP is "1") het source-adres RB en in het tweede gedeelte van de klokperiode (CP is "0") het destination-adres RC wordt aangeboden aan de Register File. Op de neergaande flank van CP worden eveneens de latches op de output-ports van de Register File gesloten en dus blijft gedurende de rest van de klokperiode de juiste source-data aangeboden aan de ALU. Door de WRITE/ output gedurende de CP "hoog"-tijd ook "hoog" te houden heeft men een drie-adres structuur gerealiseerd. Dit kan men bereiken door het IEN/ signaal op de juiste wijze te manipuleren. In figuur 2.4 is het tijd-diagram voor zowel de twee- als de drie-adres structuur weergegeven.

2.2.2 De Am2904 Status & Shift Control Unit

De Am2904 "Status and Shift Control Unit" vervult vier belangrijke functies voor bit-slice microprocessoren zoals de Am2903. Deze vier functies zijn achtereenvolgens:

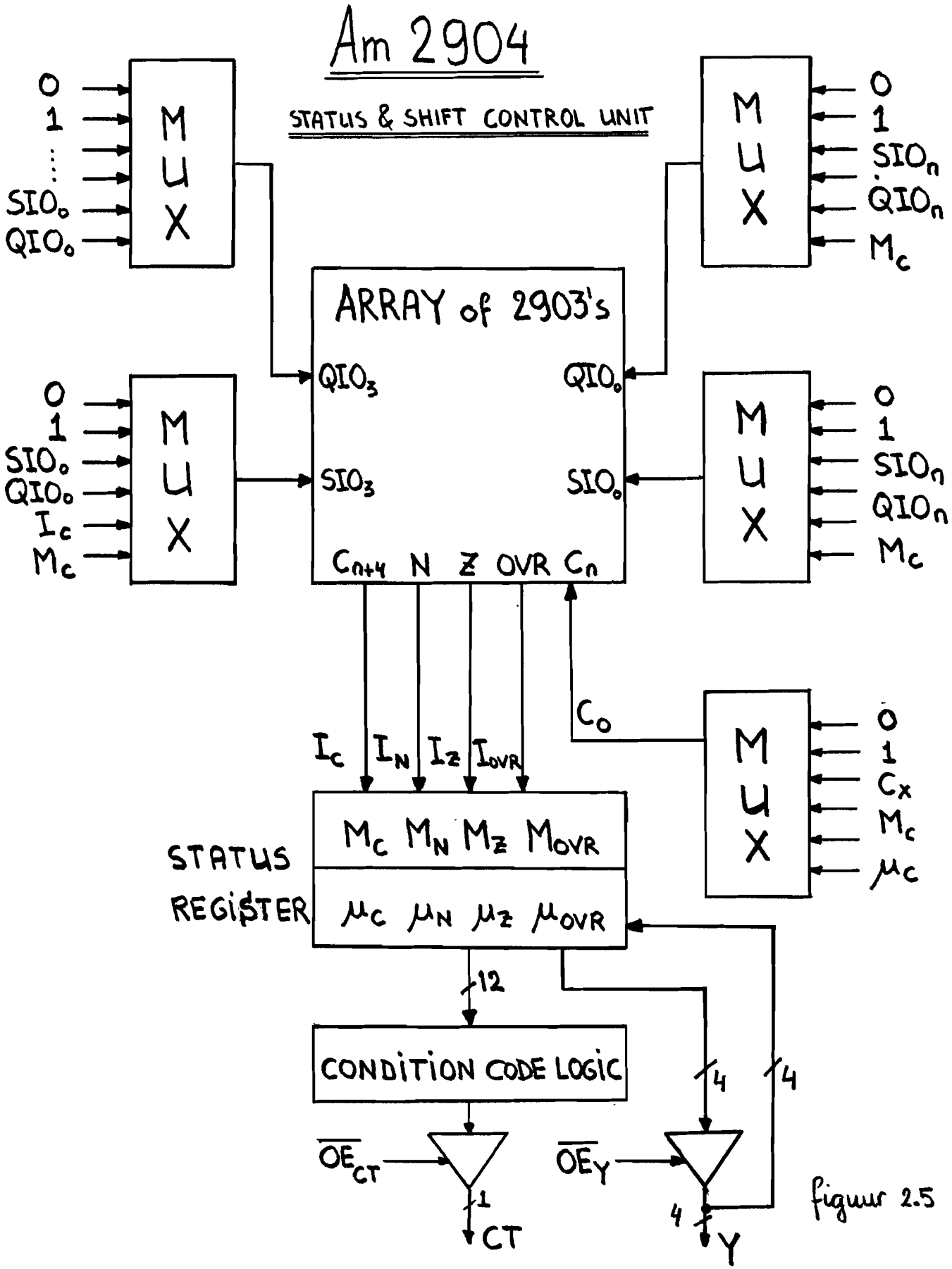
1) Status Register
2) Conditie Code Logica
3) Schuif Verbindingen
4) Carry-in Multiplexer

In figuur 2.5 zien we deze functies weergegeven. De 2904 bevat alle logica die rond de 2903's getekend is. Al deze functies worden bestuurd door middel van dertien instructielijnen, I_0 t/m I_{12}, en negen "enable"-lijnen. Op de volgende bladzijden worden deze functies beknopt besproken. Een meer volledige beschrijving van de mogelijkheden die de Am2904 biedt, vindt men in de literatuur (AM29,MICK).

ad 1) Het Status Register

De Am2904 bevat twee 4-bit registers waarin de status outputs van de ALU, Carry (C) Negative (N) Zero (Z) en Overflow (OVR), kunnen worden opgeslagen. Deze twee registers hebben respectievelijk de namen Micro Status Register (uSR) en Machine Status Register (MSR) meegekregen. Deze registers worden betuurd door middel van de instructie-lijnen I_0 t/m I_5 en een zestal "enable"-lijnen. Het uSR heeft een enable-lijn, CE_u/, waarmee schrijven in het register, onafhankelijk van de opgedrukte instructie, verboden wordt. Dit in tegenstelling tot het MSR dat vijf enable-lijnen kent. Een enable-lijn voor elke bit-positie (E_Z/, E_N/, E_{OVR}/ en E_C/) en een voor het hele MSR (CE_M/).

Het uSR kan worden geladen vanuit de status inputs (I_Z, I_N, I_{OVR} en I_C) of vanaf het MSR. De bits in het uSR kunnen afzonderlijk worden geset of gereset. Het MSR kan worden geladen vanaf de status inputs, vanaf het uSR en vanaf de Y-bus. De bits in het MSR kunnen worden geset, gereset of geinverteerd. Door de bit-enable lijnen van het MSR te gebruiken kunnen de bits in het MSR ook afzonderlijk worden gemanipuleerd. Met behulp van de bidirectionele Y-bus kan men de inhoud van de status registers redden en weer terugschrijven. Deze voorziening kan men bijvoorbeeld gebruiken bij interrupt afhandeling. Hierbij wil men in het algemeen eerst de processorstatus op een stack redden, alvorens de interrupt service routine uit te voeren.


Am 2904
STATUS & SHIFT CONTROL UNIT
ARRAY of 2903's
MUX
0
1
SIO0
QIO0
SIOn
QIOn
Mc
QIO3
QIO0
SIO3
SIO0
Cn+4 N Z OVR Cn
Ic
Co
IC IN IZ IOVR
MC MN MZ MOVR
μC μN μZ μOVR
STATUS REGISTER
Cx
μc
12
CONDITION CODE LOGIC
4
OECT
OEY
CT
Y

figuur 2.5

ad 2) Conditie Code Logica

De conditie code uitgang CT van de 2904 kan worden gebruikt als testingang voor de sequencer. Met behulp van de instructielijnen I_0 t/m I_5 kan men op uitgang CT elk bit uit de statusregisters en van directe ingangen I (al dan niet geinverteerd) laten verschijnen. Ook is een aantal combinaties van deze bits beschikbaar om bijv. het testen mogelijk te maken op het "groter dan", "groter of gelijk dan", "kleiner dan" en "kleiner of gelijk dan" zijn van twee unsigned of two's complement getallen.

ad 3) Schuif Verbindingen

De "Shift Linkage Multiplexers" (de vier bovenste multiplexers in figuur 2.5) zorgen voor de noodzaklijke verbindingen tussen de SIO- en QIO-lijnen van de 2903's. Met behulp van de vijf instructie-lijnen I_6 t/m I_{10} kan men 32 verschillende verbindingen tot stand brengen. Hiermee is o.a. voorzien in schuiven over enkele en dubbele lengte, zowel naar links als naar rechts en met of zonder carry (M_C). Met de SE/-input (Shift Enable) kan men de outputs van de Shift Linkage Multiplexers in three-state toestand zetten.

ad 4) Carry-in Multiplexer

De Carry-in Multiplexer verzorgt de Carry-in input, C_n, van de "least significant ALU slice". Gekozen kan worden uit een zevental bronnen: 0, 1, M_C, M_C/, u_C, u_C/ en C_x. Hierbij is C_x een input pen van de 2904. Om enkele speciale functies van de 2903 uit te kunnen voeren, dient men de Z- output van de 2903 aan te sluiten op de C_x input van de 2904. Met behulp van de instructie-lijnen I_{12}, I_{11}, I_5, I_3, I_2 en I_1 selecteert men de benodigde carry-in bij een opdracht voor de 2903.

2.3 Het Ontwerp

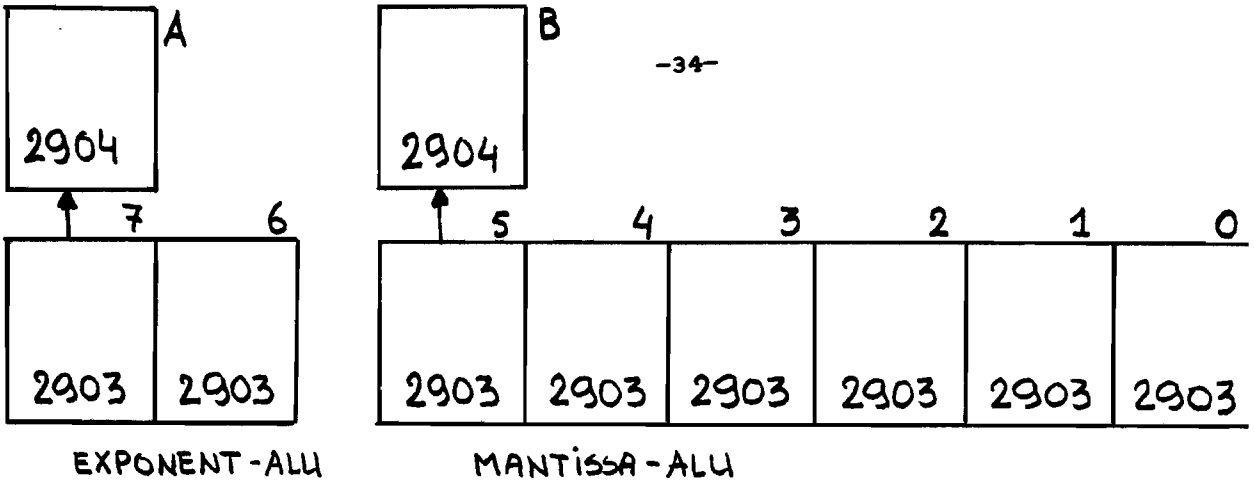
2.3.1 Inleiding

Zoals we in paragraaf 2.1 gezien hebben was een van de eisen voor het ontwerp dat men moet kunnen rekenen in 8, 16, 32 bits of in floating point notatie. De ALU's waarin deze berekeningen plaats vinden noemen we achtereenvolgens de Byte ALU de 2-Bytes ALU, de 4-Bytes ALU en de FP-ALU. In het ontwerp zijn de verschillende ALU's op dezelfde groep 2903's en 2904's geprojecteerd.

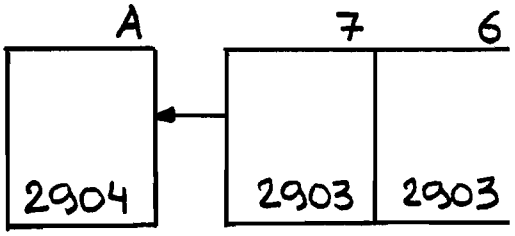
Bij floating point berekeningen gebruiken we 8 bits voor de exponent en 24 bits voor de mantisse, in totaal dus ook 32 bits. Om floating point berekeningen snel te laten verlopen worden de exponent berekeningen en de mantisse berekeningen parallel aan elkaar uitgevoerd in twee gescheiden ALU's: de Exponent-ALU en de Mantisse-ALU. Beide ALU's worden bestuurd door dezelfde sequencer met een microprogramma waarin besturingsvelden voor beide ALU's zijn opgenomen. Beide ALU's hebben ook ieder hun eigen 2904 Status and Shift Control Unit. Interactie tussen beide ALU's kan men bijvoorbeeld realiseren door de sequencer te laten testen op een bit van het status register in de 2904 van de Mantisse-ALU en afhankelijk van de uitslag van deze test de Exponent-ALU al dan niet een actie te laten uitvoeren (bijv. het verhogen van de exponent).

De ALU's zijn opgebouwd met de Am2903 bit-slice microprocessor circuits. Daar elke Am2903 een 4-bit breed data pad heeft, hebben we in totaal 8 van deze circuits nodig om de FP-ALU en de 4-Bytes ALU te kunnen realiseren. De 2903's zijn genummerd van 7 tot en met 0. De 2904's hebben, eveneens ter onderscheid, de letters A, B en C meegekregen. De diverse ALU's zijn als volgt op de 2903's en 2904's geprojecteerd:

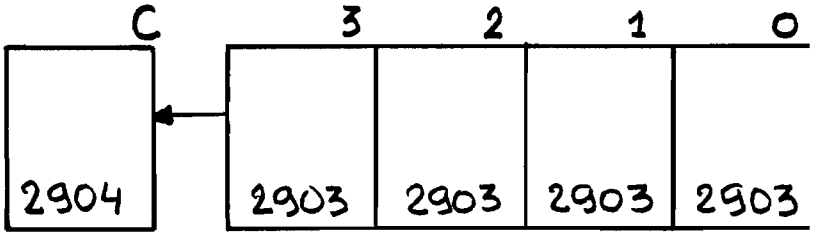
Exponent-ALU : 2903's 7 en 6 ; 2904-A
Mantisse-ALU : 2903's 5 t/m 0 ; 2904-B

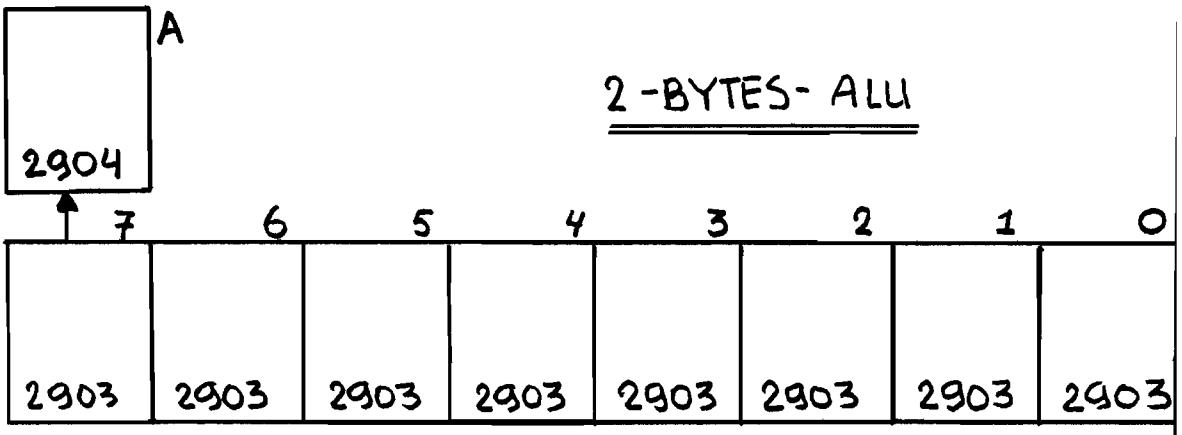


FLOATING POINT-ALU



BYTE-ALU





2-BYTES-ALU

4-BYTES-ALU

figuur 2.

Byte-ALU	:	2903's	7 en 6 ; 2904-A
2-Bytes ALU	:	2903's	3 t/m 0 ; 2904-C
4-Bytes ALU	:	2903's	7 t/m 0 ; 2904-A

In figuur 2.6 zien we dit nog eens schematisch weergegeven. Een andere mogelijk logische plaats voor de Exponent-ALU zouden sliceposities 1 en 0 geweest zijn. Deze keuze echter zou tot een ondoorzichtiger hardware ontwerp geleid hebben, wat betreft het carry lookahead gedeelte. In het huidige ontwerp wordt voor de FP-ALU en de 4-Bytes ALU dezelfde carry lookahead hardware gebruikt. Het ligt voor de hand de Byte-ALU op de Exponent-ALU te projecteren, daar beide dezelfde afmetingen hebben. Hiervoor is dus geen extra hardware nodig.

Het feit dat de Byte ALU en de 2-Bytes ALU niet op dezelfde slices geprojecteerd zijn kan tot gevolg hebben dat men bij emulatie van een 16-bits processor veelvuldig gegevens moet uitwisselen tussen beide ALU's. De meeste 16-bits processoren kunnen namelijk zowel in 8 bits als in 16 bits rekenen. Een voordeel is echter dat we nu de mogelijkheid erbij gekregen hebben de Byte-ALU en de 2-Bytes ALU parallel te bedrijven.

De Mantisse-ALU (3-Bytes ALU) en de 2-Bytes ALU kunnen nooit tegelijkertijd bedreven worden, daar ze gedeeltelijk op dezelfde 2903's geprojecteerd zijn. Het was dus mogelijk geweest om te volstaan met een 2904 die zowel de 2-Bytes als de 3-Bytes ALU zou kunnen verzorgen. Hiervoor is niet gekozen om het ontwerp doorzichtig te houden. Bovendien zou er extra hardware nodig zijn geweest om de 2904 te kunnen overschakelen van de ene naar de andere ALU.

In de volgende paragrafen worden de volgende aspecten van het hardware ontwerp nader toegelicht:

2.3.2 Carry Verbindingen
2.3.3 Nul Detectie
2.3.4 Schuif Verbindingen
2.3.5 Besturingssignalen

2.3.2 Carry Verbindingen

Wat betreft de carry verbindingen kan men onderscheid maken tussen drie categorien:

1) Carry Lookahead
2) Carry-in 2903-0
3) Interslice Carries

Deze drie categorien worden nu achtereenvolgens behandeld.

ad 1) Carry Lookahead

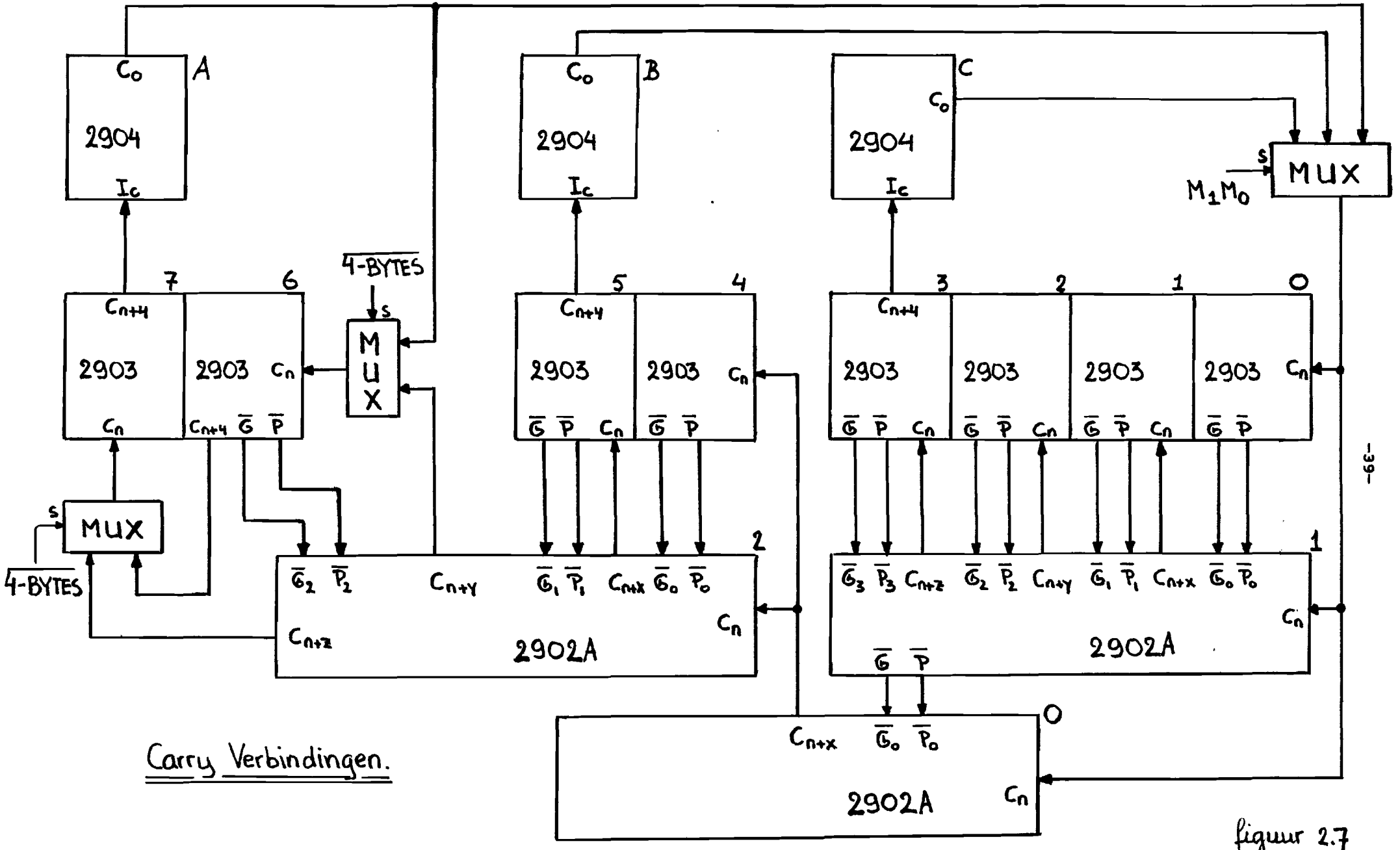
Indien men meerdere 2903's tot een ALU wil vormen dan dient er een route te zijn waarlangs een carry-out zich naar de carry-in van de volgende slice kan voortplanten. Een eenvoudige manier is de "Ripple-Carry" methode. Hierbij verbindt men de C_{n+4}-output van een 2903 met de C_n-input van de 2903, die aan zijn linkerzijde geplaatst is (indien de LSS geheel rechts geplaatst is). Het voordeel van deze methode is dat men hiervoor geen extra hardware nodig heeft. Hier tegenover staat dat deze methode traag is.

We zullen nu een tijdberekening maken voor de Ripple Carry methode (wat betreft de C_{n+4} output van de MSS in de 4-Bytes ALU). De combinatorische vertraging in de 2903 van input C_n naar output C_{n+4} bedraagt 30 nanoseconden. Indien men de ALU een van de standaard functies laat verrichten dan is maximaal 88 ns nadat alle inputs stabiel zijn geworden ook de output C_{n+4} stabiel. Dit geschiedt parallel in alle 2903's. Vervolgens moeten alle carry-out signalen zich nog voortplanten. In het geval van een 4-Bytes ALU (8 slices) duurt het dus: 88 + 7*30 = 298 ns alvorens de C_{n+4} output van de MSS zijn gegarandeerde eindwaarde heeft bereikt.

Om hier een versnelling mogelijk te maken is in de 2903's enige logica opgenomen die het mogelijk maakt om een Carry Lookahead Generator toe te passen. De 2903 genereert hiertoe de signalen P/, Carry Propagate, en G/, Carry Generate. Het Carry Propagate signaal geeft aan dat, als de betreffende slice een carry-in krijgt, de slice ook een carry-out genereert. Het Carry Generate signaal geeft aan of de ALU bewerking intern in de slice een carry-out tot gevolg heeft, onafhankelijk van het carry-in signaal. Het carry-out signaal C_{n+4} kan dus worden voorgesteld door de logische functie: $C_{n+4} = G + P*C_n$.
De Am2902A Carry Lookahead Generator heeft voorzieningen om een viertal Carry Propagate/Generate signaal-paren, P3 t/m P0 en G3 t/m G0, en een carry-in signaal te verwerken tot een drietal carry-in signalen (C_{n+x}, C_{n+y} en C_{n+z}) voor de overeenkomstige 2903's. Bovendien genereert de Am2902A een Carry Propagate/Generate signaal-paar (P,G) om carry lookahead over een groter bereik mogelijk te maken (d.w.z. over meer dan vier 2903's). De logische functies die de Am2902A realiseert zijn als volgt:

$$C_{n+x} = G0 + P0*C_n$$
$$C_{n+y} = G1 + P1*G0 + P1*P0*C_n$$
$$C_{n+z} = G2 + P2*G1 + P2*P1*G0 + P2*P1*P0*C_n$$
$$G = G3 + P3*G2 + P3*P2*G1 + P3*P2*P1*G0 + P3*P2*P1*P0*C_n$$
$$P = P3*P2*P1*P0$$

P,G is een Carry Propagate/Generate signaal-paar dat een groep van vier 2903 slices vertegenwoordigt. In figuur 2.7 zien we hoe drie Am2902A's in het ontwerp zijn toegepast. De carry-in multiplexers voor de slices 7 en 6 zijn nodig om deze twee slices zelfstandig (Byte-ALU) te kunnen maken en dus los te koppelen van de carry lookahead hardware van de overige 2903's. Alleen indien men een 4-Bytes ALU wenst, is het noodzakelijk dat de carry-in multiplexers van de slices 6 en 7 respectievelijk C_{n+y} en C_{n+z} van de 2902A selecteren. Indien men geen 4-Bytes ALU wenst dan moeten deze multiplexers



Carry Verbindingen.

figuur 2.7

respectievelijk de C_0 output van 2904-A en C_{n+4} output van 2903-6 selecteren. Slice 6 en 7 zijn dan dus in Ripple Carry mode geschakeld. Een carry lookahead generatie voor slechts twee 2903 slices levert geen noemenswaardige tijdwinst op (hoogstens enkele ns). Voor de standaard ALU functies kan het zelfs tijdverlies opleveren. De multiplexers worden geschakeld met het stuursignaal 4-Bytes/. Dit stuursignaal is uit het Pipeline Register afkomstig en is als volgt gedefinieerd:

4-Bytes/ = 0 ; 4-Bytes ALU
4-Bytes/ = 1 ; Geen 4-Bytes ALU

Dit houdt in dat de 2903's van de Byte ALU gescheiden worden van de overige 2903's. Men kan dan de Byte ALU (eventueel parallel aan de 2- of 3-Bytes ALU), de 2- of de 3-Bytes ALU bedrijven.

Device Nr.	Device Path	Delay
2903-7 t/m 0	all naar P/,G/	81
2902A-1	Pi/,Gi/ naar P/,G/	12
2902A-0	Po/,Go/ naar C_{n+x}	9
2902A-2	C_n naar C_{n+z}	14
74S157	A naar Y	7,5
2903-7	C_n naar C_{n+4}	30
		+
Total	ns	153,5

Carry-out Delay Time.
4-Bytes ALU met Carry Lookahead Hardware.

figuur 2.8

We kunnen nu weer een tijdberekening opzetten om te vergelijken hoe het Carry Lookahead schema zich gedraagt ten opzichte van de Ripple Carry methode. Ook hier gaan we er van uit dat de 4-Bytes ALU een standaard functie verricht. Alle

2903's berekenen parallel hun P/ en G/ waarden. Deze zijn, nadat alle inputs van de 2903 stabiel zijn, na maximaal 81 ns op hun eindwaarde.

In de tabel van fig 2.8 zien we dat de benodige tijd vergeleken met de Ripple Carry methode ongeveer gehalveerd werd van 298 ns naar 153,5 ns. De Carry Lookahead methode levert dus een behoorlijke winst op.

ad 2) Carry-in voor 2903-0

De Carry-in multiplexer voor 2903-0 kan selecteren uit drie bronnen: de C_0 outputs van de 2904's A, B en C. Dit is nodig om achtereenvolgens de 4-, 3- en 2-Bytes ALU te kunnen realiseren. De multiplexer wordt bestuurd met de selectielijnen M1 en M0. De stuursignalen worden afgeleid uit het al bekende stuursignaal 4-Bytes/ en de nieuwe stuursignalen 3-Bytes/ en 2-Bytes/. Ook deze signalen zijn afkomstig uit het Pipeline Register en zijn als volgt gedefinieerd:

```
3-Bytes/ = 0  ; 3-Bytes ALU
3-Bytes/ = 1  ; Geen 3-Bytes ALU

2-Bytes/ = 0  ; 2-Bytes ALU
2-Bytes/ = 1  ; Geen 2-Bytes ALU
```

Van deze drie stuursignalen mag er nooit meer dan een tegelijkertijd actief ("0") zijn, daar men slechts een van de drie ALU's (de 2- ,3- of 4-Bytes ALU) tegelijkertijd kan bedrijven. Wel mogen meerdere stuursignalen tegelijkertijd non-actief zijn. In de tabel van figuur 2.9 zijn alle toegestane combinaties van deze stuursignalen opgenomen. Men had ook kunnen volstaan met twee stuursignalen, daar het aantal toegestane combinaties slechts 4 bedraagt. Hiervoor is niet gekozen, omdat de fysische betekenis van dergelijke signalen minder doorzichtig zou zijn.

Uit de tabel kan men voor M1 en M0 de volgende logische functies afleiden:

M1 = 3-Bytes + 2-Bytes

M0 = 4-Bytes + 2-Bytes

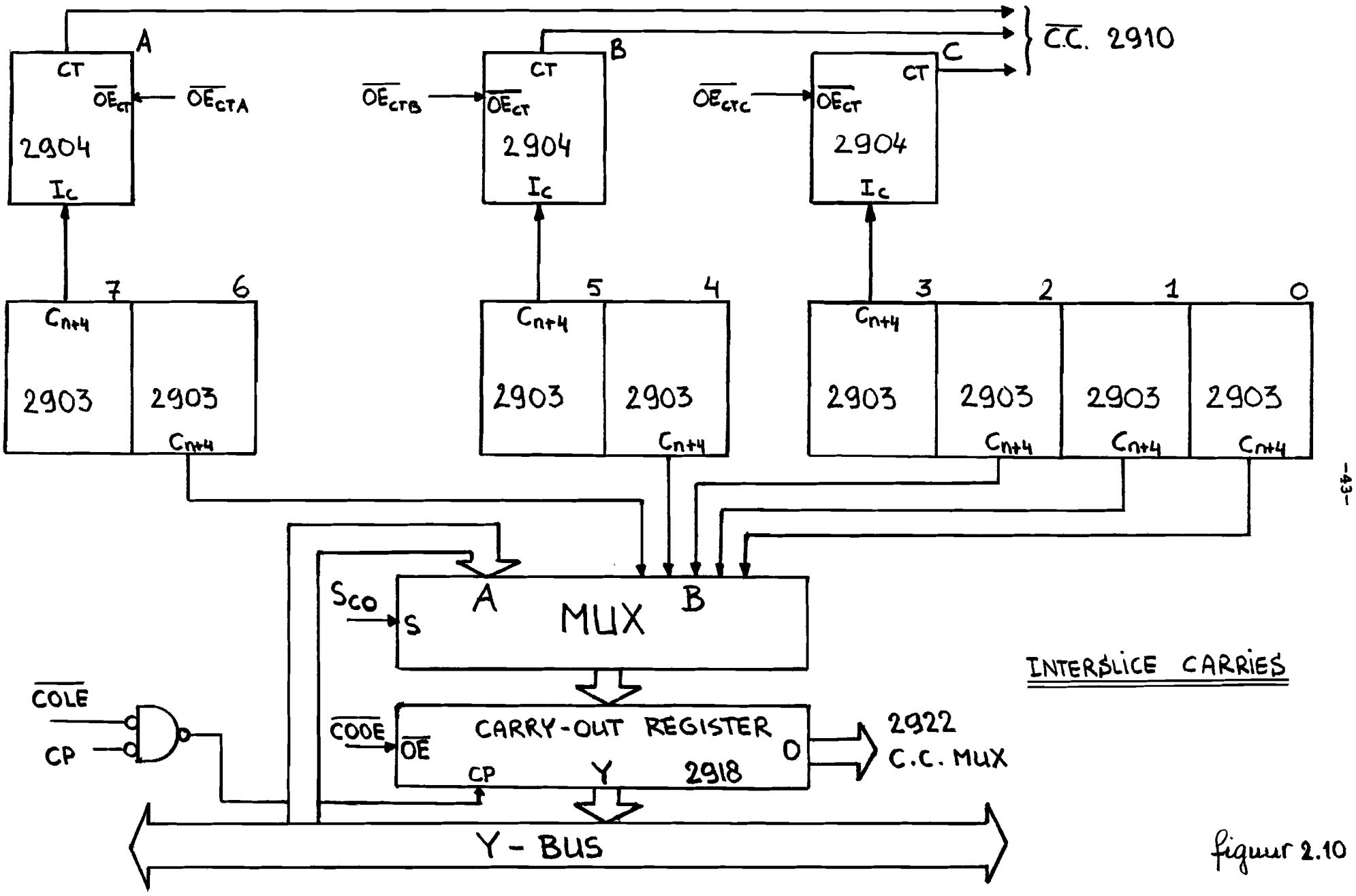
4-Bytes/	3-Bytes/	2-Bytes/	C_n (2903-0)	M1	M0
1	1	1	X	0	0
0	1	1	2904-A	0	1
1	0	1	2904-B	1	0
1	1	0	2904-C	1	1

Multiplexer besturingssignalen M1 en M0.

figuur 2.9

ad 3) Interslice-Carries

Indien men in de BCD (Binary Coded Decimal) getallen representatie wenst te rekenen is het noodzakelijk, dat men kan testen op alle interslice carries (de C_{n+4}'s). Drie van deze carries heeft men via de 2904's A, B en C al beschikbaar.
Voor de resterende vijf carries is extra hardware toegevoegd. In figuur 2.10 zien we dat die hardware bestaat uit een multiplexer en een Carry-out Register. Het Carry-out Register is opgebouwd met behulp van Am2918 circuits. Deze circuits bevatten flip-flops van het D-type en hebben twee soorten outputs: een "totem-pole output" die met de conditie code multiplexer input van de sequencer verbonden is en een "three-state output" die op Y-bus aangesloten is. De three-state output kan gebruikt worden om de inhoud van het Carry-out Register (dat een onderdeel van het PSW - Program Status Word - vormt) te



INTERSLICE CARRIES

figuur 2.10

kunnen redden. Ook kan het worden teruggeschreven via de multiplexer die selecteert tussen de Y-bus en de 2903 Carry-outputs.

Indien men in de BCD representatie rekent, dient men na elke bewerking tussen twee BCD getallen het resultaat weer in een correcte BCD notatie te brengen. Na elke berekening dient dus eerst een correctieslag te volgen, alvorens verder te rekenen. De Carry-out signalen die bij de bewerking tussen twee BCD getallen optreden, dient men gedurende de gehele correctieslag vast te houden. Dit kan men realiseren door het COLE/-signaal (Carry Out Latch Enable) "hoog" te houden waardoor de klokingang van het Carry-out Register "hoog" blijft. De correctiebewerking dient per digit te geschieden (te beginnen met het "least significant digit") daar elke correctiebewerking een carry-out naar een hogere positie tot gevolg kan hebben. Per BCD digit kan men twee gevallen onderscheiden, waarin men actie dient te ondernemen:

1) Vanuit deze digit is een carry-out opgetreden naar de volgende slicepositie. Deze carry-out is opgetreden, ten gevolge van een bewerking tussen twee correcte BCD getallen; dus niet tijdens de correctieslag. Dit soort carry-outs staan in het Carry-Out Register.

2) Het resultaat van de bewerking tussen twee BCD digits levert een van de hexadecimale getallen A t/m F op (ook tijdens de correctieslag) d.w.z. geen correct BCD getal.

De actie, die men in beide gevallen dient te ondernemen, is het verhogen van het resultaat in de betreffende slice met zes. Via de conditie code multiplexer kan de sequencer het Carry-out Register onderzoeken. Een microprogramma onderzoekt per slice afzonderlijk of er al dan niet 6 bij opgetelt dient

te worden. De controle op het groter dan 9 zijn (A t/m F) gebeurt eveneens per slice afzonderlijk d.m.v. een microprogramma (maskeren, aftrekken en testen). De hele correctieslag wordt hierdoor vrij langdurig.

Men zou de zaak enigzins kunnen versnellen door wat extra hardware toe te voegen aan het ontwerp. Deze hardware zou er als volgt uit kunnen zien:

- Een FPLA, dat de vector in het Carry-out Register - in dat geval 8 bits breed - vertaalt in een immediate datavector (bestaande uit zessen en nullen). Deze vector zou op de DA-bus van de 2903's aangeboden moeten worden, om te worden opgeteld bij het resultaat van de BCD bewerking.
- Per 2903 een stuk hardware om te detecteren op het groter dan 9 zijn van de F-output van de ALU.
- Een FPLA dat deze detectie signalen eveneens vertaalt in een soortgelijke immediate datavector. Ook deze vector moet via de DA-bus bij het resultaat opgeteld kunnen worden.

Het bij deze extra hardware behorende microprogramma zou veel korter zijn en zou ook veel sneller verwerkt kunnen worden. Voor deze oplossing is echter niet gekozen vanwege een drietal redenen. Deze waren achtereenvolgens:

1) BCD rekenen had geen hoge prioriteit bij het ontwerp van deze ALU.
2) Om dit te kunnen realiseren is een behoorlijke hoeveelheid extra hardware nodig, op een toch al vrij uitgebreid ontwerp.
3) Er is een nieuwe bit-slice microprocessor aangekondigd, de Am29203, die intern al enkele BCD voorzieningen heeft. De aangekondigde nieuwe functies die de Am29203, naast de functies die ook de Am2903 al kent, zijn o.a. BCD optellen en aftrekken en conversies tussen getallen in BCD en biniare representatie. Op het moment dat dit verslag geschreven werd, waren hiervan nog geen nadere specificaties bekend.

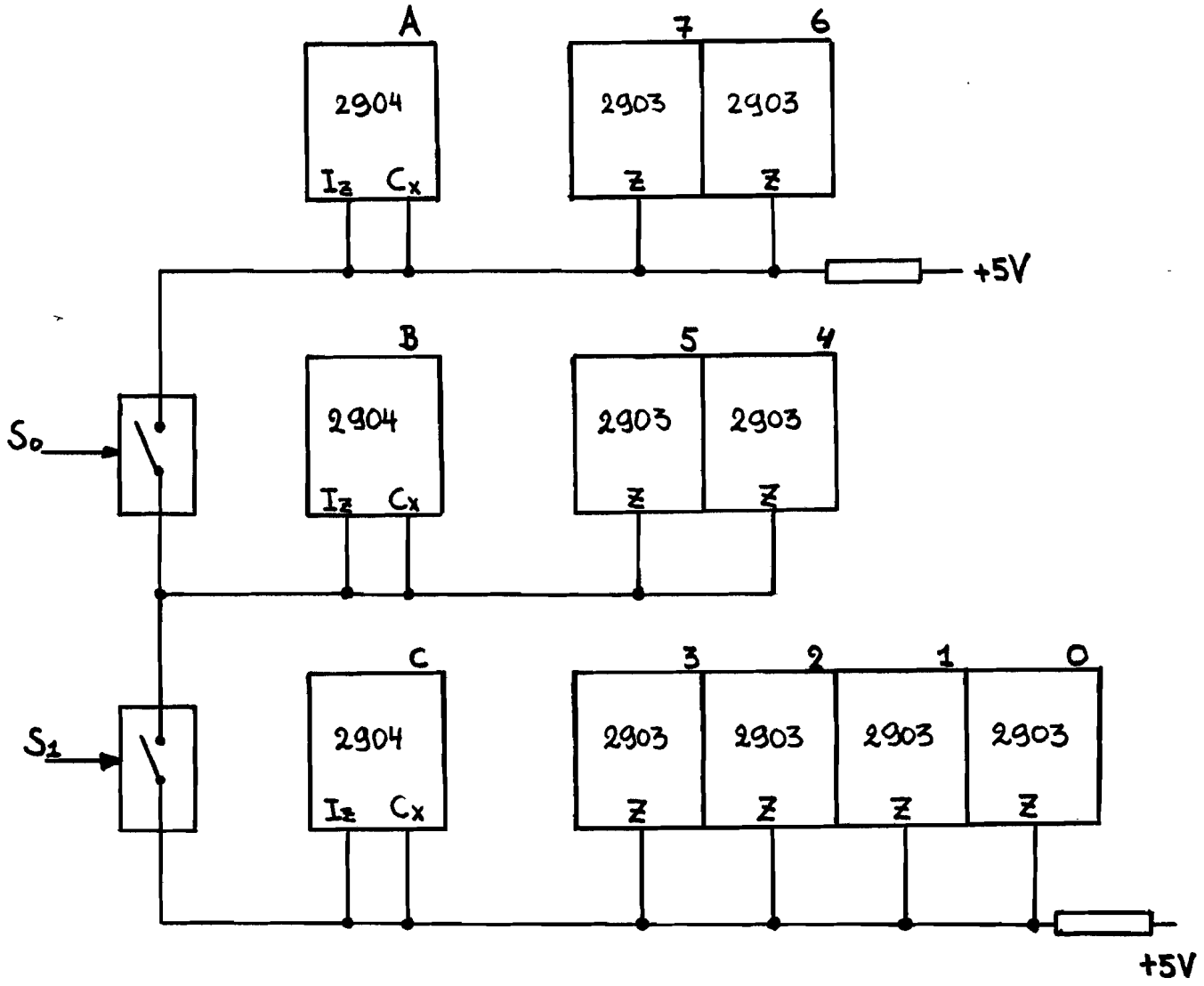
2.3.3 Nul Detectie

Zoals reeds in paragraaf 2.2.1 werd vermeld, bevat de Am2903 intern een stuk logica, dat een nul detectie verricht op de Y-bus. De output van deze logica is de Z-output. De Z-output is als open-collector output uitgevoerd. Het is hierdoor mogelijk een statuslijn Z voor de hele Y-bus te maken, door de Z-outputs van alle 2903's met een "wired-or" met elkaar te verbinden.

In figuur 2.11 zien we hoe de Z-outputs met elkaar verbonden moeten worden om de verschillende ALU configuraties te kunnen realiseren. Schakelaar S_0 is noodzakelijk om de Byte ALU te kunnen scheiden van de overige 2903's. Schakelaar S_1 vervult eenzelfde functie voor de 2-Bytes ALU.
Het is noodzakelijk dat de paden, die geschakeld worden, bidirectioneel zijn. Dit is nodig daar de Z-pen bij gebruik van de speciale functies ook als input kan fungeren, om informatie van de ene naar de andere slice door te kunnen geven. Zie literatuur (AM29,MICK). Een andere eis is dat de "control input" van de schakelaar bestuurd moet kunnen worden vanuit het Pipeline Register. De control input moet dus kunnen werken met signalen op TTL nivo. Als schakelaar werd gekozen de CD4066 van National Semiconductor. Dit circuit bevat vier "bilateral switches" en is gerealiseerd in de CMOS technologie. Enkele relevante technische specificaties van de CD4066 zijn als volgt:

"ON" Resistance (5 V power supply)	:	120 ohm
Propagation Delay	:	25 ns
"ON-OFF" Switch Delay	:	90 ns

Meer uitgebreide gegevens van de CD4066 vindt men in de literatuur (NATI). De "ON-OFF" schakeltijd lijkt met 90 ns aan de lange kant. Men dient echter te bedenken dat er alleen geschakeld wordt, als men overgaat op een andere ALU configura-



ZERO - LIJN VERBINDINGEN

figuur 2.11

tie. In figuur 2.12 vindt men een overzicht van de standen van de schakelaars in de verschillende ALU configuraties. Hierbij geldt de volgende afspraak:

S = 1 : De schakelaar is gesloten.
S = 0 : De schakelaar is open.

4-Bytes/	3-Bytes/	2-Bytes/	S1	S0
1	1	1	X	0
0	1	1	1	1
1	0	1	1	0
1	1	0	0	0

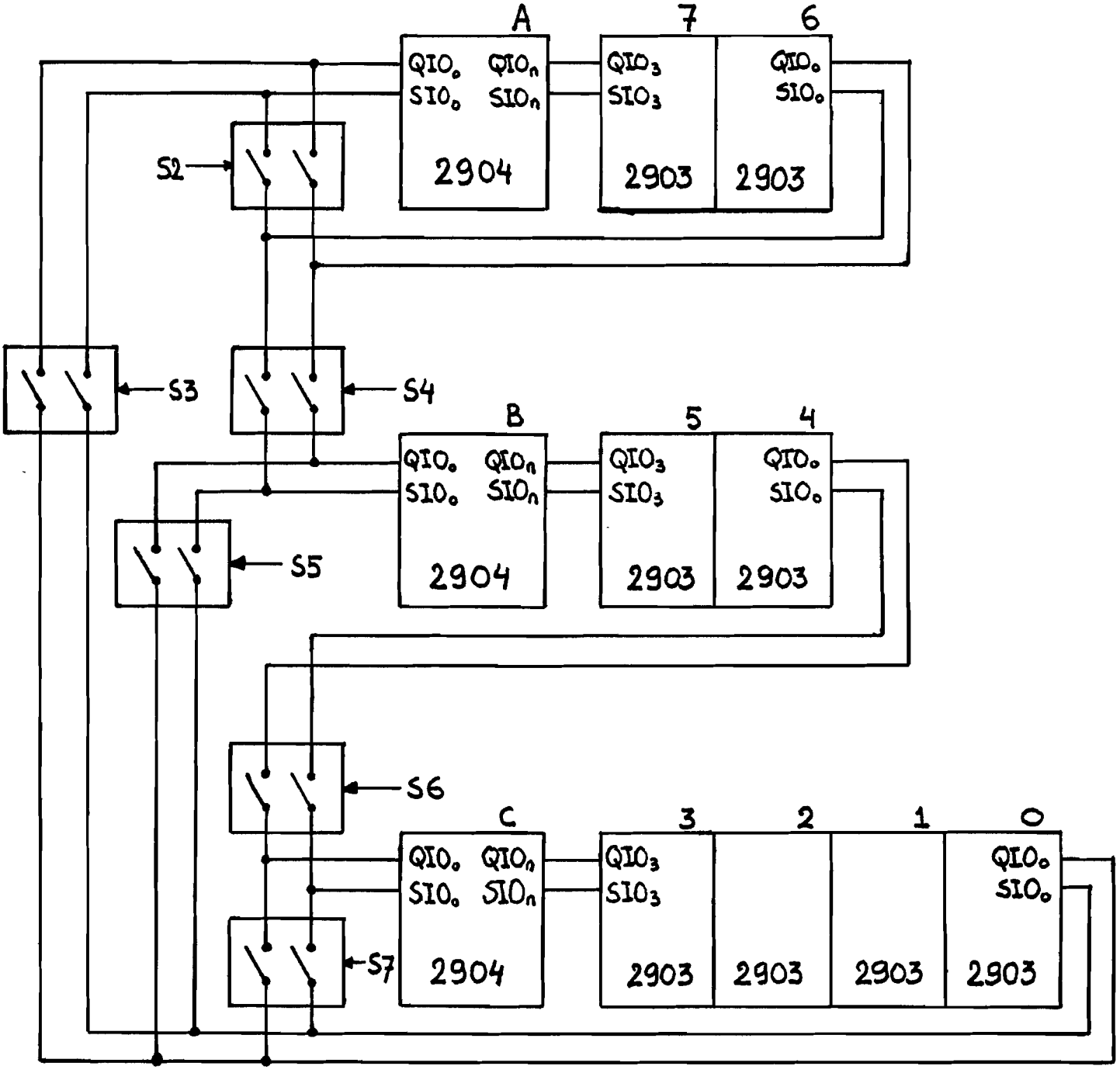
Z-lijn schakelaarstanden bij verschillende ALU configuraties.

figuur 2.12

Uit deze tabel kan men logische functies afleiden voor de control inputs van de schakelaars. Deze zijn als volgt:

S_1 = 4-Bytes + 3-Bytes
S_0 = 4-Bytes

Ook wat betreft de Z-lijn vertoont de aangekondigde Am29203 enig verschil met de Am2903. Bij de 2903 is de nul detectie logica aangesloten achter de $OE_Y/$ buffer, dus op de externe Y-bus. Dit in tegenstelling tot de Am29203, waar de nul detectie voor dit buffer plaats vindt. Een ander verschil is dat men bij de Am29203 de Z-output "hoog" kan houden door het $OE_Y/$ stuursignaal "hoog" te houden. Hierdoor kan men - met behulp van deze stuurlijnen - een nul detectie doen op een



SCHUIF VERBINDINGEN

figuur 2.13

gedeelte van de Y-bus. Bij de 2903's kan men hetzelfde bereiken, zij het iets moeizamer. Men dient hiertoe de $OE_Y/$ stuurlijnen op dezelfde wijze te bedienen; in combinatie hiermee moeten op die plaatsen waar $OE_Y/=1$ nullen geforceerd worden op de externe Y-bus.

2.3.4 Schuif Verbindingen

Om de benodigde schuif verbindingen tot stand te brengen in de verschillende ALU configuraties, zijn een zestal dubbele schakelaars nodig (voor zowel de SIO- als de QIO-lijnen). Zie figuur 2.13. Ook deze schakelaars dienen voor een bidirectionele verbinding te zorgen. Dit is nodig om dat men wenst zowel naar rechts als naar links te kunnen schuiven. De gekozen schakelaars zijn van hetzelfde type als die bij de Z-lijn zijn toegepast: de CD4066. In figuur 2.14 zijn de schakelaarstanden voor de verschillende ALU configuraties weergegeven.

4-Bytes/	3-Bytes/	2-Bytes/	S7	S6	S5	S4	S3	S2
1	1	1	0	0	0	0	0	1
0	1	1	0	1	0	1	1	0
1	0	1	0	1	1	0	0	1
1	1	0	1	0	0	0	0	1

Schakelaarstanden van schuif verbindingen bij verschillende ALU configuraties.

figuur 2.14

Uit deze tabel kan men weer de schakelfuncties voor de control inputs van de schakelaars afleiden. Deze zien er als volgt uit:

S_7 = 2-Bytes	S_4 = 4-Bytes
S_6 = 3-Bytes + 4-Bytes	S_3 = 4-Bytes
S_5 = 3-Bytes	S_2 = 4-Bytes/

In geval van een 4-Bytes ALU dienen zowel 2904-B en C een zodanige instructievector opgedrukt te krijgen, dat ze wat betreft de SIO- en QIO-lijnen transparant zijn. In geval van een 3-Bytes ALU geldt hetzelfde voor 2904-C.

2.3.5 Besturingssignalen

Als men een eenvoudige ALU ontwerpt (bijv. een Byte-ALU) dan liggen alle sliceposities vast. In het algemeen is dan de WRITE/ pen van LSS verbonden met de WE/ inputs van alle 2903's. Ook alle IEN/ inputs zijn dan met elkaar verbonden tot een IEN/ stuurlijn. Met dit stuursignaal kan men alle schrijfoperaties in de hele ALU verbieden.

In ons geval liggen de zaken niet zo eenvoudig. Hiervoor zijn een aantal redenen op te voeren.

1) De sliceposisties liggen niet vast. Dit komt doordat we met behulp van dezelfde slices meerdere ALU configuraties willen realiseren.
2) We hebben een mechanisme nodig dat (bij de verschillende ALU configuraties) ervoor zorgt, dat de schrijfpulsen - afgegeven door de als LSS geprogrammeerde slices - bij de juiste 2903 WE/ inputs komen.
3) We willen het schrijven in delen van de ALU kunnen verbieden. Stel we schrijven data - bestemd voor de Byte, de 2-Bytes of de 3-Bytes ALU - in de Register

File. Dan willen we dat de inhoud van de Register File op de overeenkomstige adressen in de rest van de ALU slices behouden blijft.

ad 1) Positionering van 2903 slices

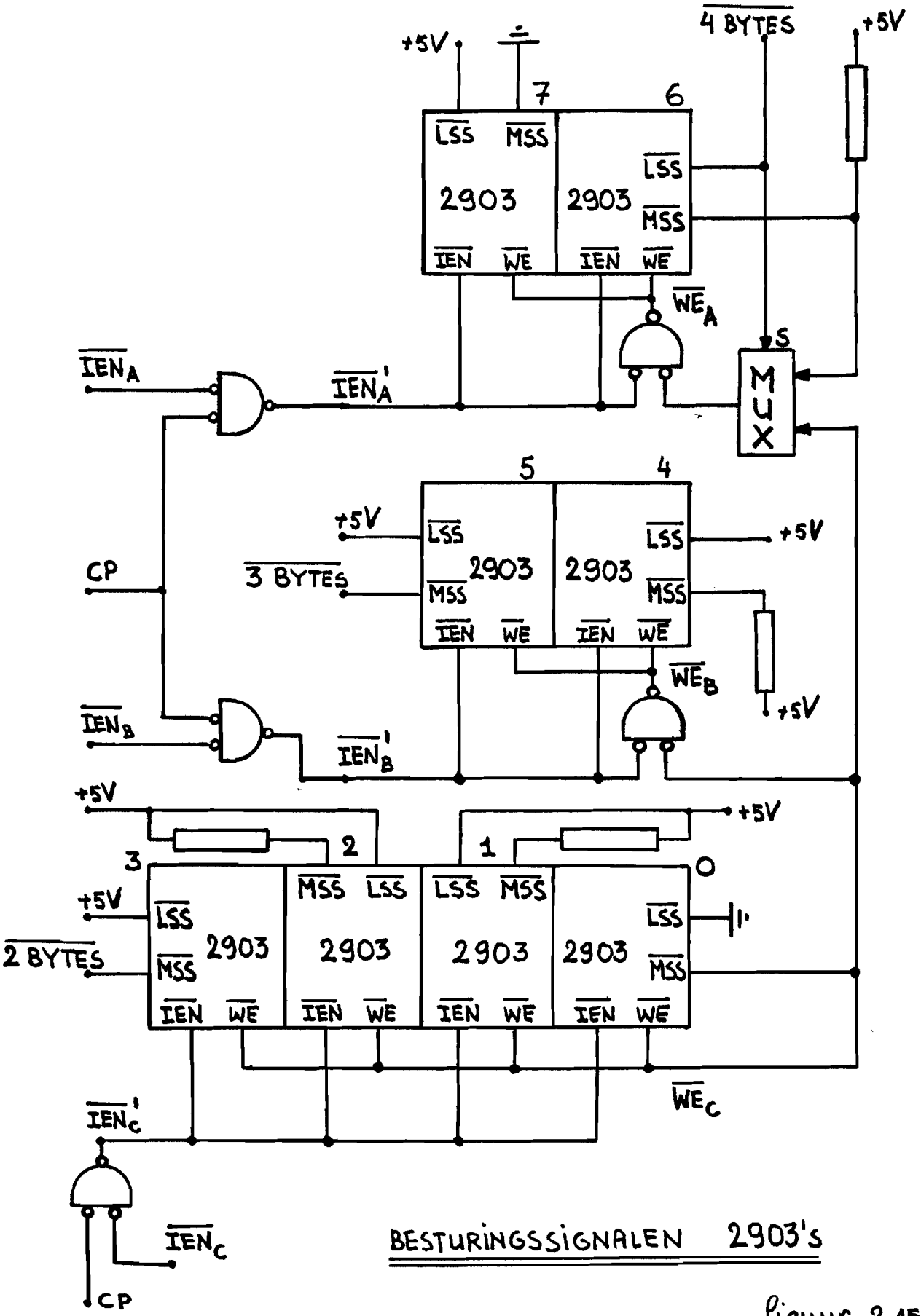
Onder positionering van slices verstaan we het programmeren van de slices als MSS, IS of LSS. Dit programmeren kan gebeuren door de LSS/ en MSS/ input pennen op het juiste nivo te brengen. De positie van vijf van de acht slices ligt vast voor alle ALU configuraties. De positie van 2903-6, 2903-5 en 2903-3 kan veranderen als men een andere configuratie selecteert. De slices worden als volgt gepositioneerd:

2903-7 : MSS.
2903-6 : IS bij 4-Bytes ALU ; LSS bij overige gevallen.
2903-5 : MSS bij 3-Bytes ALU ; IS bij overige gevallen.
2903-4 : IS.
2903-3 : MSS bij 2-Bytes ALU ; IS bij overige gevallen.
2903-2 : IS.
2903-1 : IS.
2903-0 : LSS.

Ook de slice positionering wordt bestuurd met de bekende stuursignalen 2-Bytes/, 3-Bytes/ en 4-Bytes/. Zie figuur 2.15.

ad 2) Distributie Schrijfsignalen

In geval van een 4-Bytes ALU dienen alle acht 2903's hun schrijfpuls van slice 0 te krijgen. Als men de Byte ALU wenst te bedrijven dan moeten de slices 7 en 6 hun schrijfpuls van slice 6 ontvangen. Dit kunnen we bereiken door voor de WE/ inputs van de slices 7 en 6 een multiplexer te plaatsen. Met behulp van het stuursignaal 4-Bytes/ selecteert deze multiplexer de MSS/ output van de slice 6 of slice 0. Zie figuur 2.15.



BESTURINGSSIGNALEN 2903's

figuur 2.15

ad 3) Instruction Enable Signalen

Als men de Byte-ALU met een andere ALU (de 2- of 3-Bytes ALU) parallel wil bedrijven dan dienen alle besturingssignalen van deze ALU's gescheiden te zijn. De Byte-ALU dient dan dus ook een eigen IEN/ signaal te krijgen. Dit signaal heeft de naam IEN_A/ gekregen en stuurt de IEN/ inputs van de 2903's 7 en 6. Ook de 2-Bytes ALU heeft voor de IEN/ inputs van de 2903's 3 t/m 0 een eigen stuursignaal, IEN_C/. De resterende slices 5 en 4 zijn voorzien van het IEN_B/ signaal. Zie figuur 2.15.

In een aantal gevallen wensen we het schrijven in een deel van de ALU te verbieden. Bijvoorbeeld als men de 2-Bytes ALU bedrijft, dan wil men niet dat de inhoud van de Register File of van het Q Register van de 2903's 5 en 4 verandert. De inhoud van het Q Register blijft behouden als men de IEN_B/ input van de 2903's 5 en 4 "hoog" houdt. De inhoud van de Register File is hierdoor echter nog niet beveiligd. Door IEN_B/ "hoog" te houden wordt namelijk niet voorkomen dat slice 0 een schrijfpuls genereert, die ook de WE/ inputs van slice 5 en 4 bereikt. Een als LSS geprogrammeerde 2903 geneert alleen dan geen schrijfpuls, als zijn eigen IEN/ "hoog" gehouden wordt.

Een eenvoudige oplossing hiervoor is: het beveiligen van de WE/ inputs van de slices 5 en 4 met IEN_B/. Als men nu dit signaal "hoog" houdt dan bevriest men alle geheugen functies van de slices 5 en 4. Eenzelfde voorziening is getroffen voor de slices 7 en 6. Voor slices 3 t/m 0 is deze voorziening niet nodig omdat hier het "hoog" houden van IEN_C/ wel tot gevolg heeft dat de schrijfpuls - afkomstig van slice 0 - uitblijft. Let wel, als men IEN_C/ "hoog" houdt dan ontvangen slices 5 en 4 - ongeacht de toestand van het IEN_B/ signaal - geen schrijfpuls. De inhoud van de Register File van slices 5 en 4 blijft nu ongewijzigd.

Ook wat betreft de definitie van het IEN/ signaal wijkt de aangekondigde Am29203 enigszins af van de Am2903. Bij de 29203

betekent "hoog" houden van IEN/ dat alle geheugen functies bevroren worden, onafhankelijk van het feit of er een externe schrijfpuls komt. IEN/ beinvloedt hier de WRITE/ output van de LSS niet. Als we nu 29203's zouden toepassen in dit ontwerp, dan zouden slices 5 en 4 wel een schrijfpuls ontvangen indien $IEN_C/=1$ en $IEN_B/=0$.

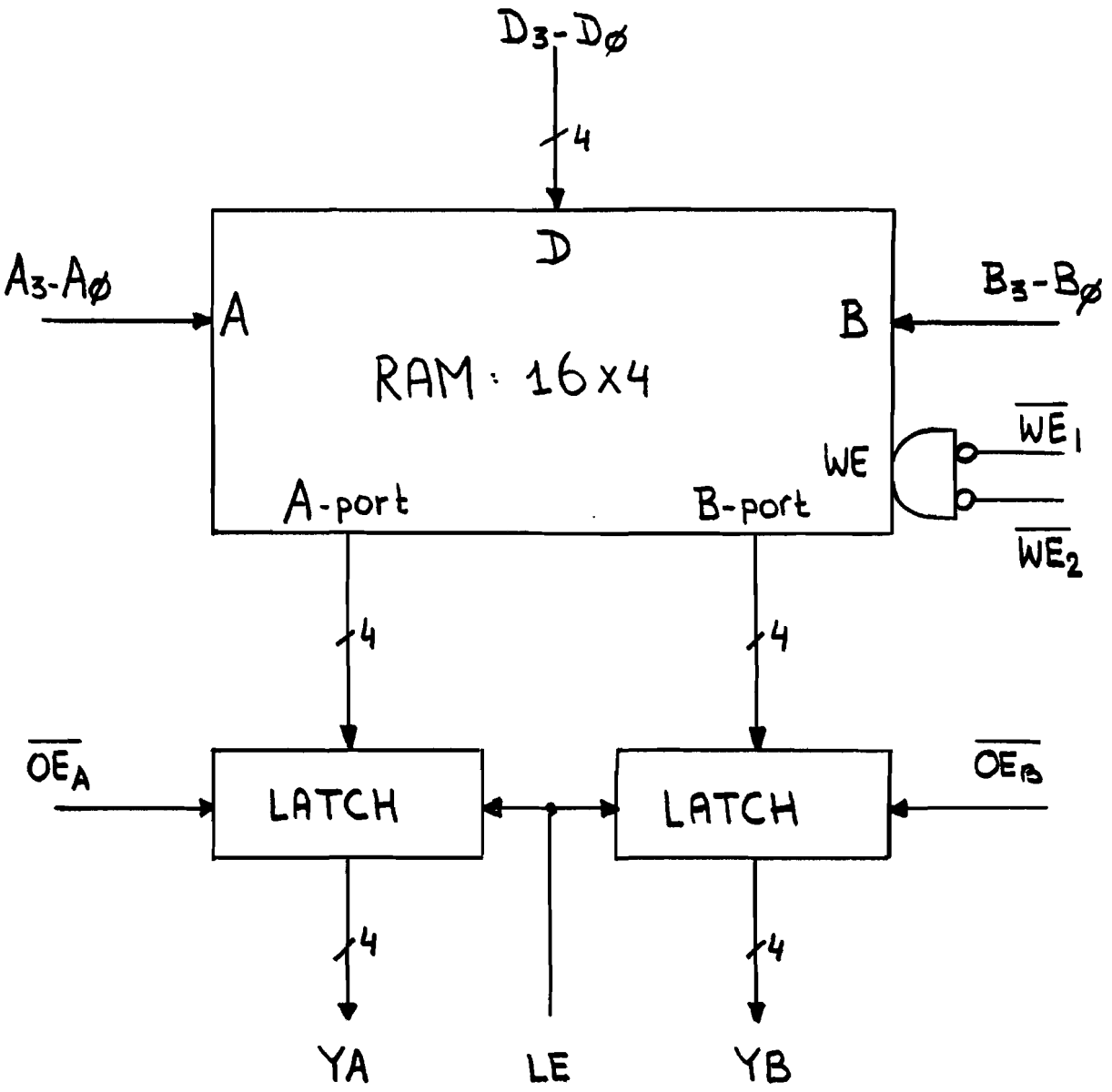
In figuur 2.15 zien we dat de drie verschillende IEN/ signalen gekoppeld zijn met het kloksignaal CP. Dit wordt gedaan om de drie-adres structuur, zoals die in paragraaf 2.2.1 besproken werd, te realiseren. Verder zien we in dezelfde figuur, dat de WE/ inputs eveneens te onderscheiden zijn in drie verschillende WE/ signalen: $WE_A/$, $WE_B/$ en $WE_C/$. Deze signalen zullen we in de volgende paragraaf nog tegenkomen.

2.3.6 Uitbreiding van de Register File

De Am2903 voorziet in een Register File van 16 woorden ter breedte van 4 bits. Dit aantal van 16 registers werd te laag bevonden en besloten werd dit uit te breiden tot 64 registers. De toe te voegen hardware dient aan een aantal eisen te voldoen om te passen bij de Am2903.

- Het RAM gedeelte dient als dual-port RAM te zijn uitgevoerd. Schrijven in het RAM gebeurt altijd op de geheugenlocatie geadresseerd door een van beide adresvectoren.
- Op beide output-ports van het RAM dienen latches geplaatst te zijn.
- In de Am2903 is de vertragingstijd van "B-adres stabiel" tot "DB-output stabiel" 49 ns. De toe te voegen registers dienen een toegangstijd te hebben, die in dezelfde grootte orde ligt.

De Am29705 is een bouwsteen die aan al deze eisen voldoet



Am 29705: Dual-Port RAM

figuur 2.16

(o.a. toegangstijd is 53 ns). In figuur 2.16 ziet men het blokschema van dit circuit, dat gerealiseerd is met behulp van de Low-Power Schottky technologie. Zie lit. (AM29). De Am29705 bevat een RAM van 16 woorden ter breedte van 4 bits. Om een totaal van 64 registers te verkrijgen, dienen we dus per 2903 drie van deze circuits toe te voegen.

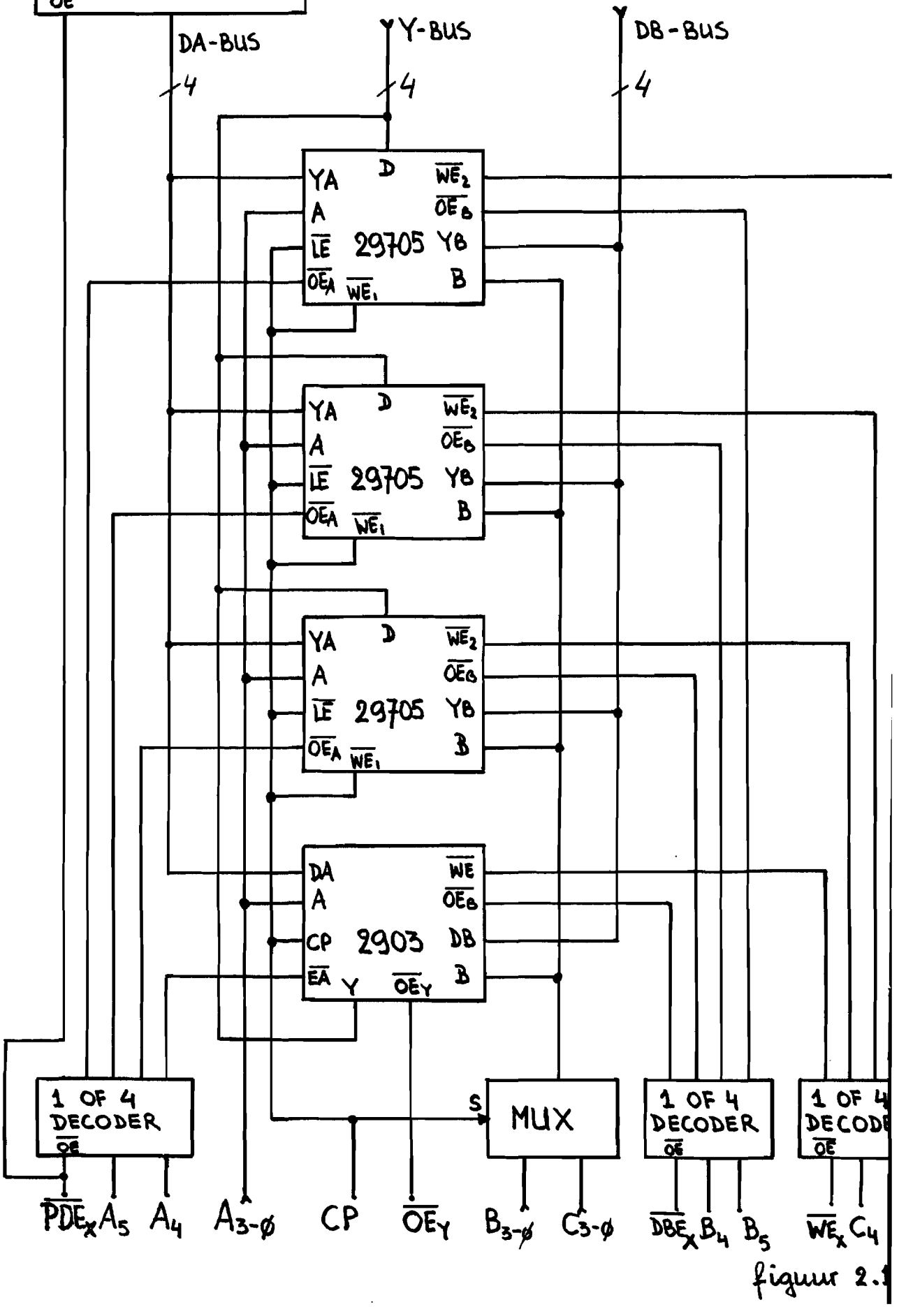
Door de LE (Latch Enable) en de WE_1/ (Write Enable) inputs van 29705 met het kloksignaal CP te verbinden, heeft men een RAM gerealiseerd, dat zich hetzelfde gedraagt als de interne RAM van de 2903. In figuur 2.17 zien we welke extra hardware er nodig is om het gewenste resultaat te verkrijgen.

Met behulp van het A-adres selecteert men een operand voor de R-input van de ALU. De adreslijnen A_3-A_0 selecteren een van de 16 woorden op de chip. A_5 en A_4 worden gedecodeerd om de OE_A/ lijnen van de RAM IC's te besturen. Met de besturingslijn PDE_X/ (Pipeline Data Enable) selecteert men immediate data, afkomstig uit het Pipeline Register. Deze data wordt op de DA-bus van de 2903's aangeboden.

De adreslijnen B_3-B_0 worden gebruikt om op elke chip het "source register" te adresseren. De gedecodeerde adreslijnen B_5 en B_4 besturen de OE_B/ inputs van de verschillende IC's. Door de outputs van de B_5,B_4 decoder "hoog" te houden (met behulp van het stuursignaal DBE_X/, DB-bus Enable) staat men toe dat de "source data" van buiten de Register File afkomstig is.

De adreslijnen C_3-C_0 worden gebruikt om het "destination register" op elk IC te adresseren. D.m.v. C_5 en C_4 worden de WE/ inputs van de IC's op de juiste wijze bestuurd. Door de outputs van de C_5,C_4 decoder "hoog" te houden - met het WE_X/ signaal - verbiedt men schrijven in alle registers van deze Register File kolom. Het WE_X/ signaal is een van de drie signalen WE_A/, WE_B/ of WE_C/ zoals die in figuur 2.15 zijn

UITBREIDING VAN DE REGISTER F



figuur 2.1

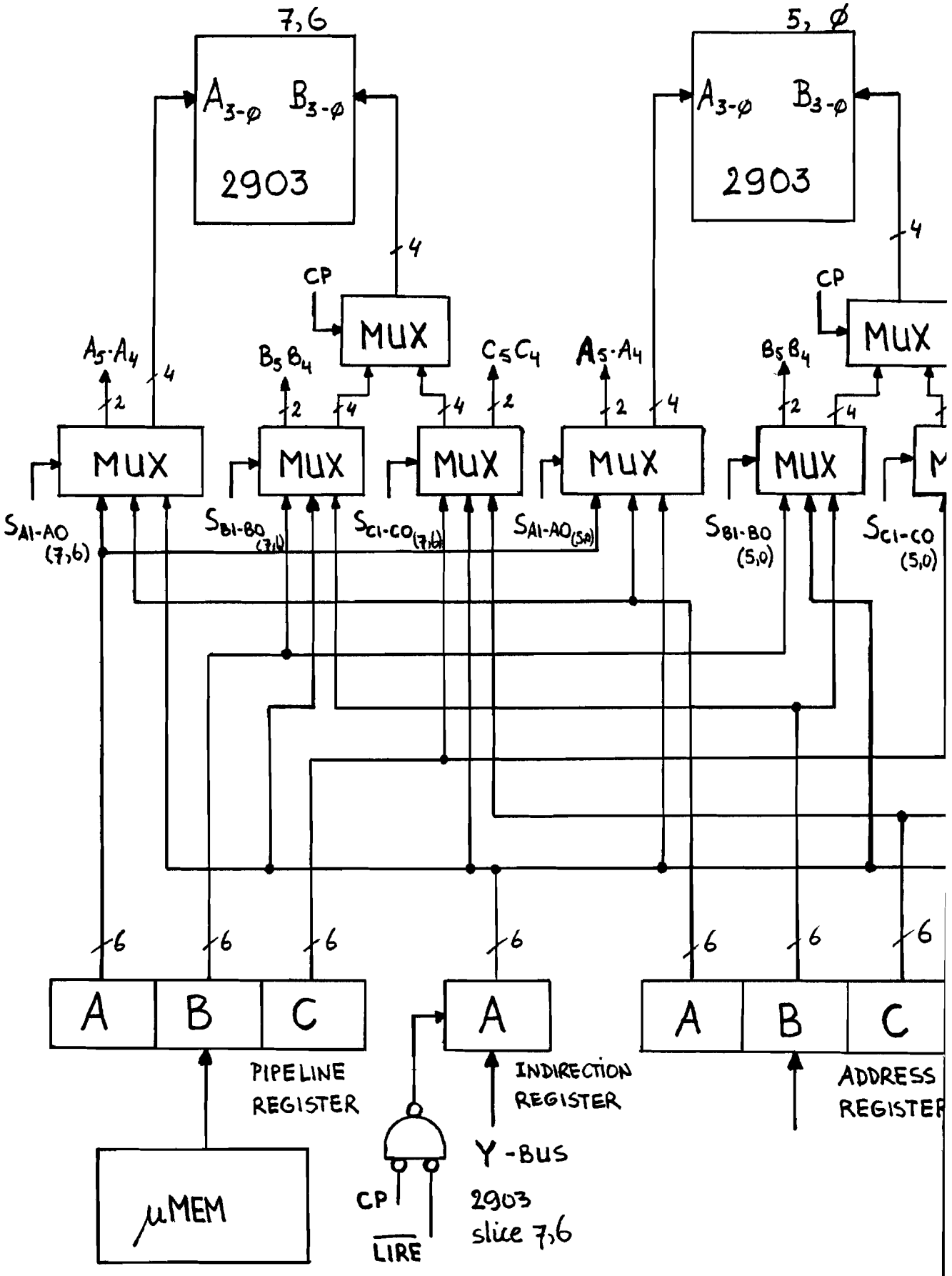
aangegeven. In de hele ALU komen dus drie C_5,C_4 decoders voor. Een voor 2903's 7 en 6 met WE_A/; een voor 2903's 5 en 4 met WE_B/ en een voor de 2903's 3 t/m 0 met WE_C/.
De andere voorzieningen ,A_5,A_4 decoder; B,C multiplexer en B_5 B_4 decoder, zijn in de hele ALU dubbel opgenomen. Een maal ten behoeve van de Exponent ALU en een maal ten behoeve van de Mantisse ALU. De enable-lijnen van de A_5,A_4 en B_5,B_4 decoders hebben respectievelijk de namen DBE_A/, DBE_{BC}/ en PDE_A/, PDE_{BC}/ meegekregen.

2.3.7 Adressering van de ALU

Als men de FP-ALU bedrijft is het handig om een mechanisme te hebben dat ervoor zorgt dat men de Exponent ALU en de Mantisse ALU ieder een verschillend adres kan aanbieden. Dit geldt voor zowel het A-, B- als C-adres. In het algemeen is het adres voor de Register File afkomstig van de Local Bus en bevindt zich in het Address Register. Er zijn zodanige voorzieningen getroffen, dat men kan kiezen uit een adres afkomstig uit het Pipeline Register, het Address Register of het Indirection Register. Zie figuur 2.18. De selectie van een adres geschiedt door middel van "4-line to 1-line data selectors". Deze selectors worden elk bestuurd d.m.v. twee selectielijnen: S_{X1} en S_{X0}. Hierbij geldt dat X is A, B of C. De toewijzing is als volgt:

S_{X1}	S_{X0}	Bron van het adres
0	0	Address Register
0	1	Pipeline Register
1	1	Indirection Register
1	0	Pipeline Register

ADRESSERING VAN DE ALU



figuur 2.1

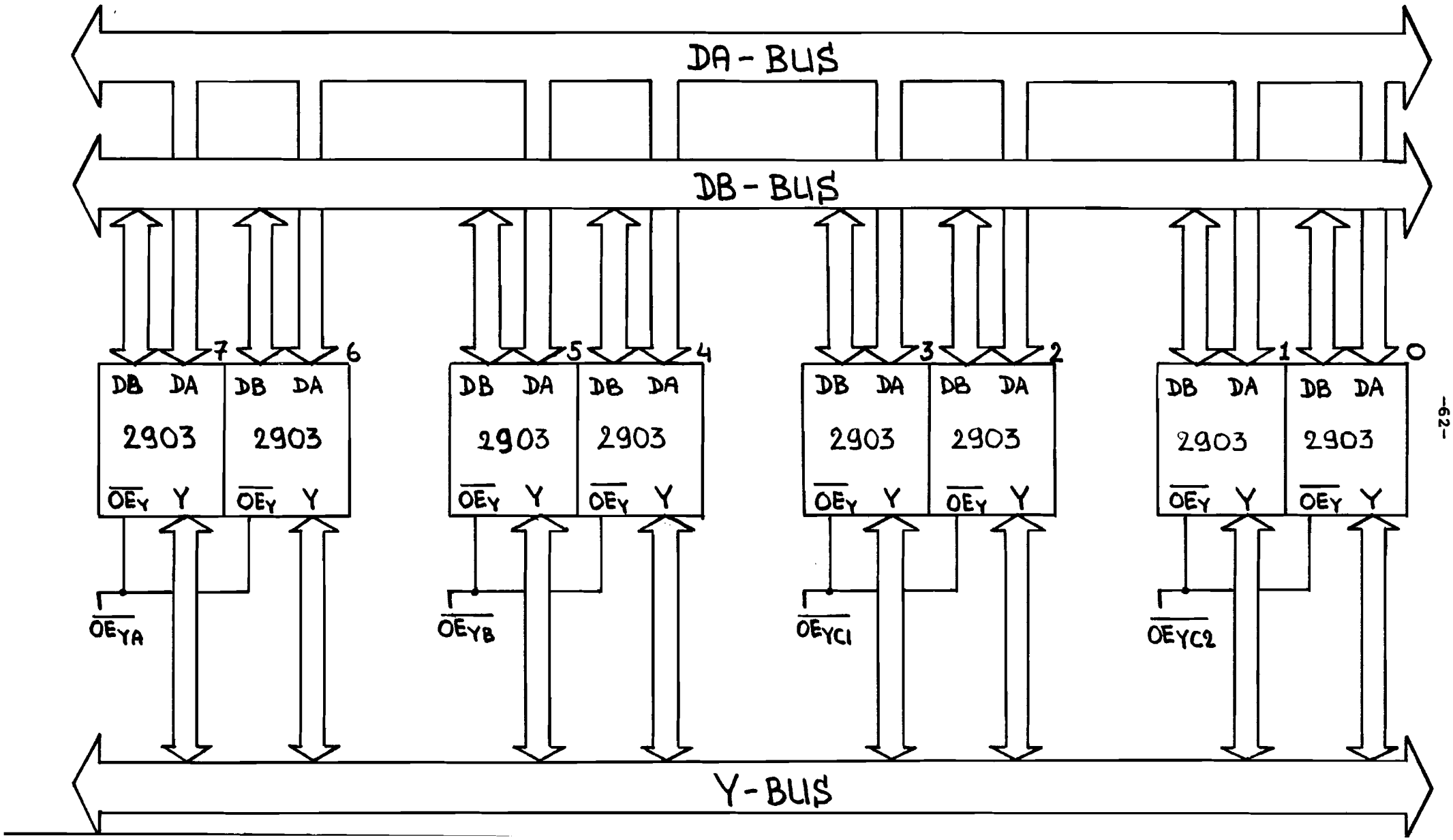
Het Pipeline Register is wegens software-technische redenen op twee manieren te adresseren. Verscheidene combinaties zijn mogelijk zoals bijvoorbeeld A-adres uit het Pipeline Register en B,C adres uit het Address Register.

Met behulp van het Indirection Register is het mogelijk indirecte adressering van de Register File te implementeren. Hierbij wordt de inhoud van een register uit de Byte ALU in het Indirection Register geplaatst. Dit register nu, kan op zijn beurt weer leverancier zijn van een A-, B- of C-adres voor de ALU. In het Pipeline Register dient hiertoe een bit gereserveerd te zijn: het LIRE/ bit (Load Indirection Register Enable). In die microcyclus waarin dit LIRE/ bit "laag" is wordt het Indirection Register geladen met het adres dat op de Y-bus posities 7 en 6 verschijnt. Met behulp van dit mechanisme is het mogelijk een hardware stack te implementeren op de Register File. Een microprogramma dient dan de boekhouding ten behoeve van de stack te verzorgen. (o.a. Increment & Decrement van Stack Pointer; Stack Full & Stack Empty indicatie.)

2.3.8 De Datapaden

In figuur 2.19 is weergegeven hoe de 2903's op de drie databussen (DA-bus, DB-bus en Y-bus) zijn aangesloten. Deze bussen zijn allen 32 bits breed, evenals het databus gedeelte van de Local Bus. De DA-, DB- en Y-busgedeelten die met 2903-7 zijn verbonden, noemen we respectievelijk DA-7, DB-7 en Y-7. Deze busgedeelten omvatten achtereenvolgens de buslijnen DA31 t/m DA28, DB31 t/m DB28 en Y31 t/m Y28. Een soortgelijke benaming geldt voor de overige busgedeelten.

De OE_Y/ lijnen van de 2903's zijn steeds per twee slices


DA-BUS
DB-BUS
7
6
5
4
3
2
1
0
DB DA
2903
OE_Y Y
OE_YA
OE_YB
OE_YC1
OE_YC2
Y-BUS

gekoppeld tot een stuurlijn. Dit is gedaan om per byte te kunnen beslissen of de ALU output F al dan niet op de Y-bus dient te verschijnen.

Om een data uitwisseling tussen de diverse ALU's mogelijk te maken zijn een aantal transceivers in het ontwerp opgenomen. Men zou hetzelfde kunnen bereiken door te schuiven via de ALU Shifters, maar dit zou erg traag zijn. De transceivers, in totaal 6 stuks ter breedte van 8 bits elk, kunnen allen een bidirectionele verbinding maken tussen de DB- en de Y-bus. In figuur 2.20 is weergegeven welke verbindingen er door de diverse transceivers gerealiseerd worden.

Transceiver Nr.	DB-bus Nr.	Y-bus Nr.
T1	DB-7,DB-6	Y-3,Y-2
T2	DB-7,DB-6	Y-1,Y-0
T3	DB-5,DB-4	Y-1,Y-0
T4	DB-3,DB-2	Y-7,Y-6
T5	DB-1,DB-0	Y-7,Y-6
T6	DB-1,DB-0	Y-5,Y-4

Verbindingen, gelegd m.b.v. transceivers T1 t/m T6.

figuur 2.20

Door de OE_Y/ lijnen van de 2903's en output-enable lijnen van de transceivers op de juiste wijze te besturen, kan men op verschillende plaatsen bytes "swappen" of "copieren". Men dient de ALU hiertoe een instructievector op te drukken die de S-operand ongewijzigd doorlaat naar de F outputs. Ook de ALU Shifter dient de data ongewijzigd door te laten.

Indien men een "byte swap" wenst te doen tussen registers op posities 7,6 en 1,0 dient men de OE/ stuursignalen van de transceivers T2 en T5 te activeren en de OE_Y/ lijnen van de 2903's als volgt te sturen: OE_{YA}/=1; OE_{YB}/=0; OE_{YC1}/=0 en OE_{YC2}/=1. Door de output-enable lijnen op deze wijze te besturen, blijft de inhoud van de Register File op posities 5 t/m 2 ongewijzigd. De gelezen data op deze posities wordt via de Y-bus weer teruggeschreven op de oorspronkelijke plaats.

Een "byte copy" is mogelijk door slechts een transceiver te activeren en drie OE_Y/ lijnen "laag" te maken. Een byte copy van een register op posities 7,6 naar een register op positie 1,0 kan als volgt gerealiseerd worden. Activeer transceiver T2 en stuur de OE_Y/ lijnen als volgt: OE_{YA}/=0; OE_{YB}/=0; OE_{YC1}/=0 en OE_{YC2}/=1. Hierdoor blijft de inhoud van het register op positie 7,6 behouden en wordt tevens gecopieerd naar een register op positie 1,0.

Het is mogelijk de Byte-ALU en de 3-Bytes ALU elk verschillende B- en C-adressen op te drukken. Hierdoor wordt een bytetransport tussen registers op ongelijke hoogte (d.w.z. met verschillende adressen) mogelijk. Dit transport vindt binnen een microcyclus plaats. Ook een 2-bytes (word) swap of copy is mogelijk tussen slicepositities 7 t/m 4 en 3 t/m 0, echter alleen tussen registers op gelijke hoogte.

Een andere vrijheid die de microprogrammeur nog heeft, is het selecteren van de richting waarin de transceivers de data doorlaten. Een voorbeeld waar men dit kan gebruiken is het volgende: Stel men wil twee bytes (byte-a en byte-b) bij elkaar optellen. Stel vervolgens dat byte-a zich in de Byte-ALU bevindt en byte-b in de 2-Bytes ALU (op slicesposities 1 en 0). Optellen van twee bytes dient altijd in de Byte-ALU te geschieden. Byte-b moet dus naar de Byte-ALU getransporteerd worden. De 2-Bytes ALU krijgt nu een zodanige instructievector

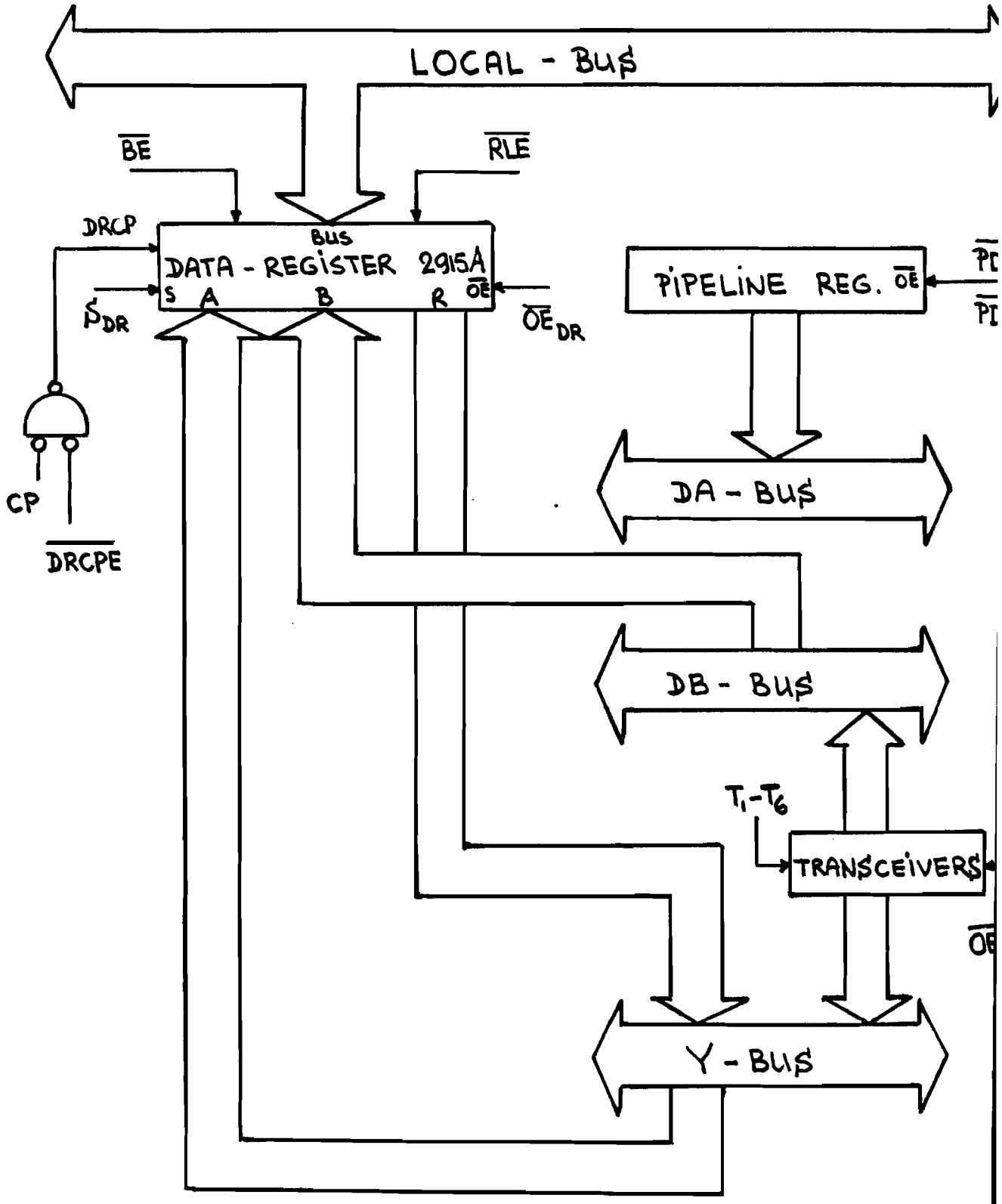
opgedrukt, dat byte-b ongewijzigd op de Y-outputs verschijnt. Door transceiver T2 in de richting Y-bus naar DB-bus te schakelen, komt byte-b via de DB-bus op de S-input van de Byte-ALU beschikbaar. Het stuursignaal DBE_A / dient dan "hoog" te zijn. Men kan de optelling nu meteen uitvoeren, indien men er voor zorgt dat byte-a tegelijkertijd op R-input van de Byte ALU verschijnt. Men heeft aldus gerealiseerd dat de optelling van beide bytes binnen een microcyclus uitgevoerd kan worden.

Uitwisseling van data tussen de Local Bus en de ALU verloopt via het Data Register. Het Data Register verzorgt een bidirectioneel datapad en is opgebouwd met behulp van Am2915A circuits. De Am2915A bevat twee registers (het Transmit Register en het Receive Register) met wat randlogica. Het Transmit Register bestaat uit vier D-flipflops, die geladen kunnen worden vanaf twee bronnen: A en B. Met behulp van selectielijn S kan men selecteren tussen A en B. Zie figuur 2.21. Data wordt in het Transmit Register geladen m.b.v. het DRCP-signaal (Driver Clock). Achter elke flipflop bevindt zich een "three-state busdriver". Door middel van het signaal BE/ (Bus Enable) kan de informatie in het Transmit Register op de Local Bus worden geplaatst. Aan dezelfde bus luisteren een aantal ontvangers, die op hun beurt weer vier D-flipflops aansturen (het Receive Register). Data wordt in het Receive Register geladen d.m.v. het RLE/-signaal (Receiver Latch Enable). Ook het Receive Register kent "three-state output drivers", die met het OE/-signaal (Output Enable) bestuurd worden. Voor uitgebreidere informatie betreffende de Am2915A zie literatuur (AM29).

Voor het ontwerp zijn acht van deze circuits nodig, daar het databus gedeelte van de Local Bus 32 bits breed is. De A en B inputs van de 2915A zijn verbonden met respectievelijk de Y-bus en de DB-bus van de 2903's. Ook de output R is verbonden met de Y-bus. Zie figuur 2.21.

De consequentie van deze keuze is, dat men de data afkomstig van de Local Bus altijd in de Register File moet

DE DATA PADEN



figuur 2.2

plaatsen, alvorens er een ALU bewerking mee te kunnen uitvoeren. Men kan het Receive Register dus niet rechtstreeks gebruiken als leverancier van immediate data. Als men dit wel had willen doen, dan zou men de R-outputs van de 2915A's met de DA- of de DB-bus van de 2903's moeten verbinden. Dit zou tot gevolg hebben, dat schrijven in de Register File vanaf de Local Bus altijd via de ALU (dus extra vertraging) zou moeten verlopen. Dit zou men kunnen versnellen door vier extra transceivers in het ontwerp op te nemen. Deze tranceivers zouden een rechtstreekse verbinding tussen de Y-bus en de DA- of de DB-bus tot stand moeten kunnen brengen. Hiervoor is niet gekozen, omdat de geringe tijdwinst niet opwoog tegen de extra hardware.

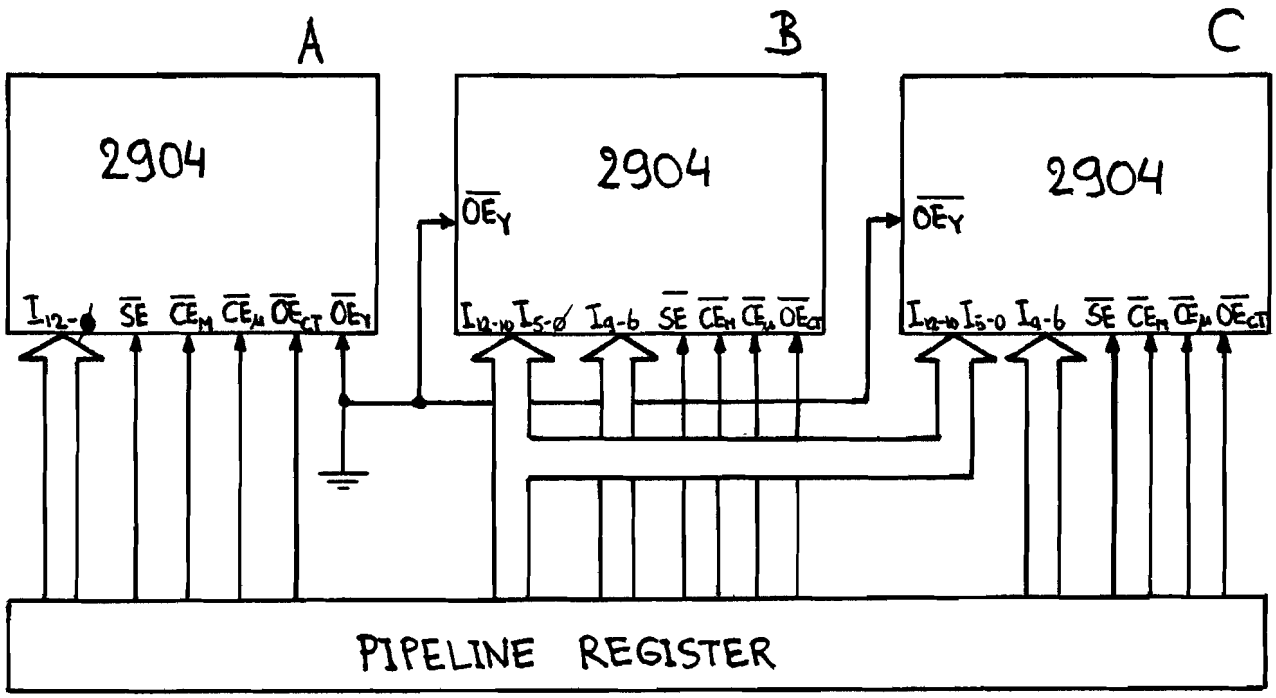
2.3.9 Besturing v.d. 2903's en 2904's

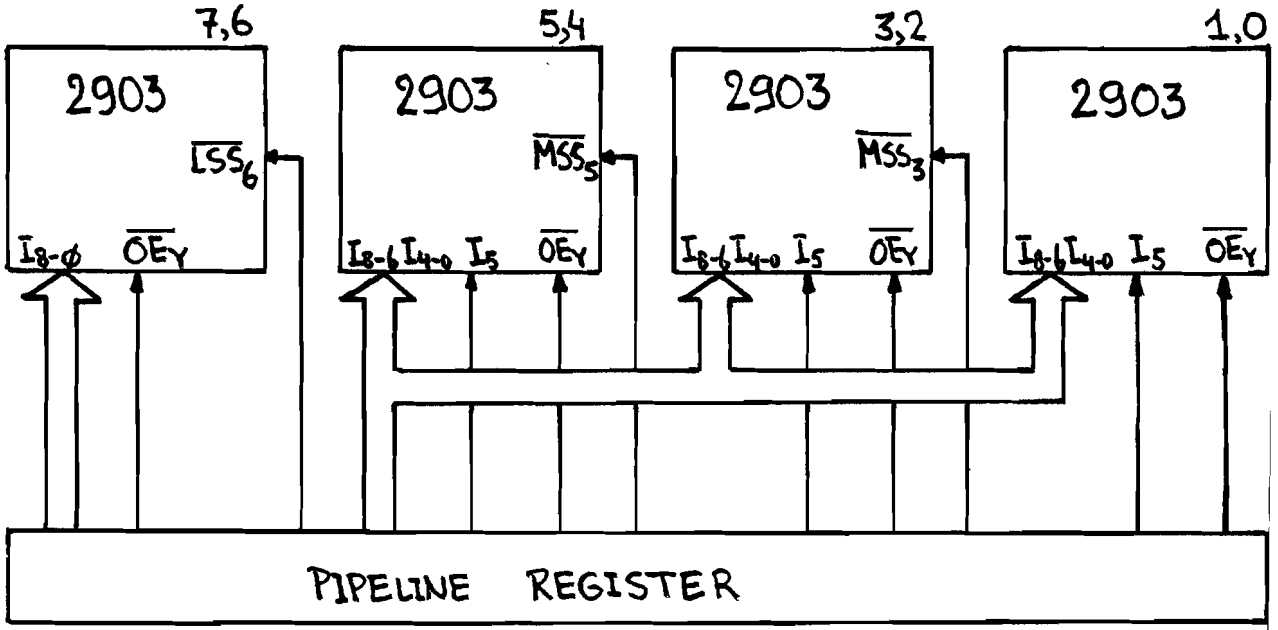
De 2903's en 2904's worden vanuit het Pipeline Register bestuurd d.m.v. de instructielijnen en de enable-lijnen. Zie figuur 2.22.

2904's.

Met behulp van de enable-lijnen kan men alle geheugen functies van de 2904 bevriezen en ook alle outputs in "three-state" zetten (behalve de C_o output). De C_o outputs van 2904's A, B en C worden via een multiplexer tot een signaal gevormd (zie paragraaf 2.3.2) en beinvloeden elkaar dus niet.

De 2-Bytes ALU en de 3-Bytes ALU kunnen nooit tegelijkertijd bedreven worden. Daarom worden 2904-B en C ook nooit tegelijkertijd gebruikt. In geval van de 3-Bytes ALU





BESTURING VAN 2903's EN 2904's.

figuur 2.22

dient 2904-C transparant te zijn, wat betreft de SIO- en QIO-verbindingen tussen 2903-3 en 2903-4. Hiertoe dient men de instrucielijnen I_9 t/m I_6 de waarde "A" te geven. Met behulp van I_{10} geeft men aan in welke richting de 2904 de SIO- en QIO-signalen doorgeeft. Als I_{10}=1, dan verschijnt de QIO_n-input op de QIO_0-output en de SIO_n-input op de SIO_0-output van de 2904 (naar links schuiven). Als echter I_{10}=0, dan verschijnt de QIO_0-input op de QIO_n-output en de SIO_0-input op de SIO_n-output (naar rechts schuiven).
In geval van een 4-Bytes ALU dient 2904-B eveneens transparant te zijn en dient dan dus dezelfde instructievector opgedrukt te krijgen als 2904-C.

Men kan een aantal plaatsen in het Pipeline Register besparen door 2904-B en C dezelfde instructielijnen I_{12} t/m I_{10} en I_5 t/m I_0 aan te bieden.

2903's.

De instructielijnen voor de 2903's kunnen gescheiden worden in twee groepen. Een instructievector I voor 2903's 7 en 6 en een voor 2903's 5 t/m 0. Dit geldt met uitzondering van instructielijn I_5. Per groep van twee 2903's is in een aparte I_5 voorzien. Dit is gedaan om een "Sign Extend" operatie mogelijk te maken.

Indien men bij een woord (2 bytes) een byte wil optellen in de 2-Bytes ALU, dan dient men eerst het byte te expanderen tot 2 bytes. Hierbij dient de numerieke waarde behouden te blijven. Stel dat het byte zich op sliceposisties 1 en 0 bevindt. We gaan er vanuit dat alle slices de geadresseerde operand ongewijzigd naar de ALU output F doorlaten. We kunnen het byte nu expanderen door 2903's 1 en 0 de instructie "F" (hexadecimale weergave I_8 t/m I_5) en de 2903's 3 en 2 de

instructie "E" op te drukken. In slice 1 en 0 verandert niets want "F" is een "Shifter Pass" instructie. In slices 3 en 2 echter zorgt de instructie "E" (Sign Extend) ervoor dat de ALU Shifter alle outputs Y gelijk maakt aan de input SIO_0 van slice 2. Op deze input staat in het algemeen het tekenbit van het byte. Via de Y-bus wordt het geexpandeerde byte teruggeschreven in de Register File. Deze sign extend operatie is alleen zinvol voor getallen in de 2's complement representatie.

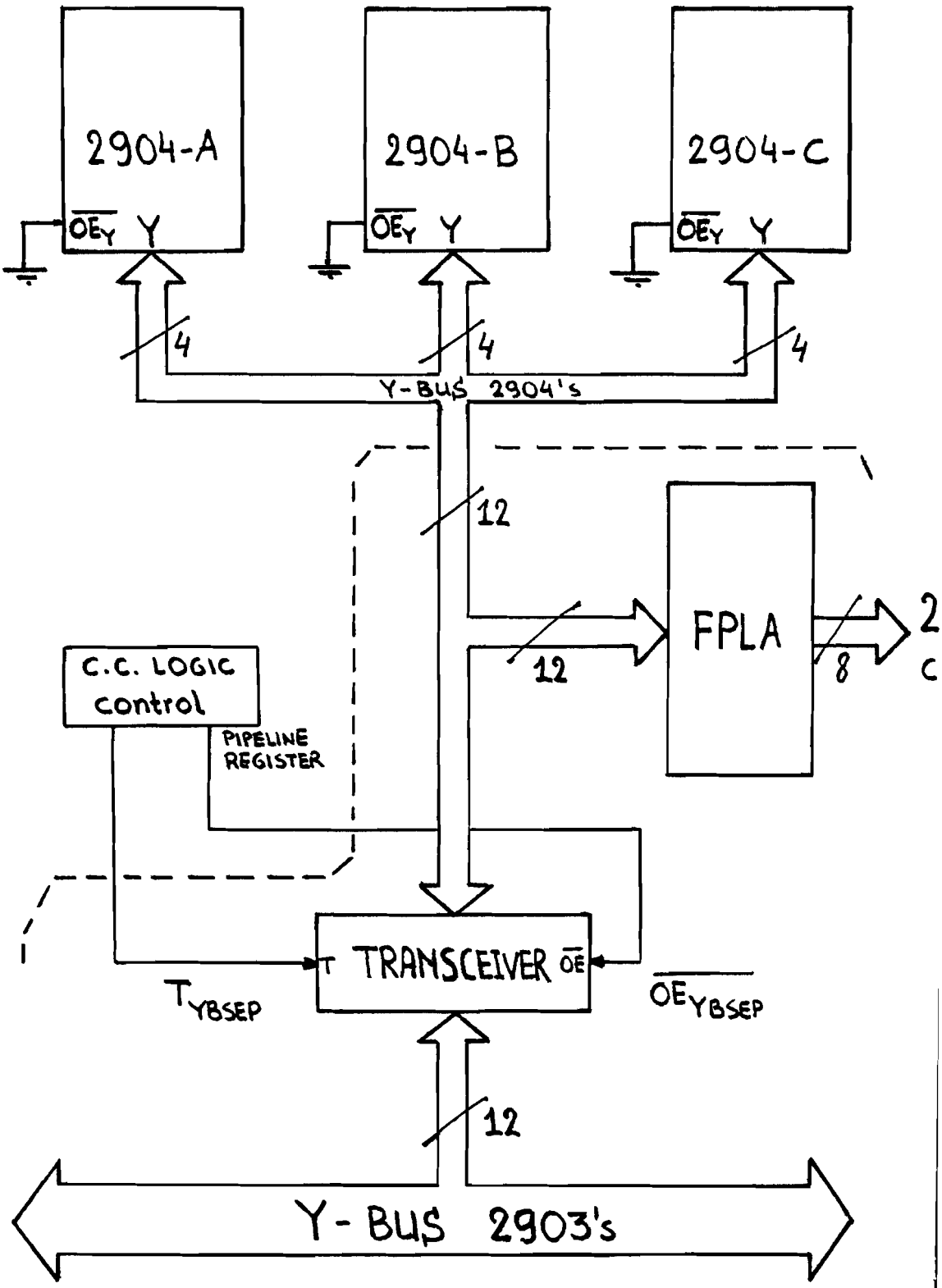
De hele sign extend operatie is binnen een microcyclus afgewerkt, ook als men over meer 2903's een sign extend wenst te doen. Door de instructielijn I_5 per groep van twee 2903's afzonderlijk te besturen is het mogelijk geworden om een sign extend te doen over een veelvoud van 8 bits. Zie figuur 2.22.

2.3.10 Conditie Code Logica

Zoals reeds eerder werd vermeld in paragraaf 2.2.2 kan men met behulp van de instructiebits I_5 t/m I_0 van de Am2904 op de CT-output een testbit selecteren, dat gevormd wordt door een combinatie van bits uit het MSR, het uSR of de directe status inputs I. Indien men wenst te testen op een combinatie waarin de 2904 niet voorziet, dan kan men dat toch doen door hiervoor een microprogramma te schrijven. Om op een combinatie van conditie code bits te testen dienen dan meerdere micro-instructies verwerkt te worden. Een snellere manier is om extra logica op te nemen, waarmee de gewenste combinaties in hardware gerealiseerd worden.

In figuur 2.23 zien we hoe dit door middel van een FPLA gerealiseerd kan worden. De ingang van het FPLA is verbonden met de Y-bus van de 2904's. Via de Y-bus van de 2904's kan men van elke 2904 het MSR, het uSR of de directe status inputs I ter beschikking krijgen. Door het FPLA op de juiste wijze te programmeren kan men elke gewenste conditie code genereren. Het is zelfs mogelijk een conditie code bit te genereren dat gevormd wordt door een combinatie van statusbits afkomstig uit verschillende 2904's. Dit is met name bij de Floating Point ALU van belang. Hier wil men testen op een combinatie van statusbits van de Exponent ALU en de Mantisse ALU. In de hoofdstukken die de software beschrijven wordt aan de functies, die in het FPLA geprogrammeerd dienen te worden, gerefereerd door middel van de namen CC_1 t/m CC_5.

Op de Y-bus van een 2904 kan men tegelijkertijd slechts een van de drie groepen statusbits ter beschikking hebben. Door middel van de instructielijnen I_5 en I_4 maakt men een keuze: het MSR, het uSR of de directe status inputs I. Het is dus met een FPLA niet mogelijk een conditie code bit te genereren, dat gevormd wordt door een combinatie van bits uit verschillende groepen statusbits van eenzelfde 2904.



CONDITION CODE LOGIC

figuur 2.23

De uitgangen van het FPLA zijn verbonden met de ingangen van een Am2922 conditie code multiplexer, die op zijn beurt het geselecteerde conditie code bit doorgeeft aan de CC/ ingang van de Am2910.

De statusbits in het MSR en het uSR van de 2904's vormen een onderdeel van het PSW (Program Status Word). De transceiver tussen de Y-bussen van de 2903's en de 2904's is opgenomen om het mogelijk te maken dat het PSW gered en teruggeschreven kan worden. De transceiver wordt bestuurd d.m.v. een tweetal besturingssignalen T_{YBSEP}/ (Transmit Y-bus Seperator) en OE_{YBSEP}/ (Output Enable Y-bus Seperator), die uit het Pipeline Register afkomstig zijn.

Daar de Y-bussen van de 2903's en 2904's gescheiden zijn, is het toegestaan dat de OE_Y/ besturingssignalen van de 2904's altijd "laag" zijn. Ook bij het terugschrijven van het PSW in de 2904's want bij deze "Load Y_Z, Y_C, Y_N, Y_{OVR} to MSR" instructie is het OE_Y/ signaal een "don't care". Bij alle andere instructies t.a.v. de 2904 is Y een output.

2.3.11 Voorspelling van het CC/ bit

De verwerkingstijd van een micro-instructie is sterk afhankelijk van de micro-operaties die in een micro-instructie zijn opgenomen. De verwerkingstijd van een micro-instructie is gelijk aan de langste verwerkingstijd van de afzonderlijke micro-operaties. Voor een systeem met een vaste klokfrequentie kan men nu de minimale klokperiode bepalen door het maximum te nemen van de verwerkingstijden van alle in het systeem voorkomende micro-instructies. Voor de meeste micro-instructies is nu teveel tijd gereserveerd.

Indien men een microprogramma zo snel mogelijk wil kunnen verwerken, dan dient men over te gaan op een systeem met een

variabele klokfrequentie. Per micro-instructie dient men de verwerkingstijd te berekenen en op te nemen in een veld van de betreffende micro-instructie. Met behulp van deze informatie kan men de klokgenerator besturen.

In paragraaf 1.3 hadden we gezien dat in negen van de zestien mogelijke 2910-instructies getest wordt op de CC/ input. De CC/ input van de 2910 kan uit de volgende bronnen voortkomen:

- Externe conditie code
- Conditie code logica
- Carry-out Register
- CT-outputs van de 2904's

De conditie code bits uit de tweede en vierde categorie kunnen worden afgeleid uit de directe status inputs van de 2904's. Deze inputs zijn pas stabiel nadat alle status outputs van de 2903's stabiel zijn geworden; in feite dus nadat de 2903's hun instructie verwerkt hebben. Hierdoor kan de adresberekening en het ophalen van het volgende uit te voeren microwoord pas gebeuren nadat de instructie van de 2903's verwerkt is. Het gevolg daarvan is dat we ons in een soort non-pipelined situatie bevinden, d.w.z. de micro-operaties worden sequentieel uitgevoerd en niet parallel. Dit resulteert in een ongewenst lange klokcyclus.

Een manier om dit te voorkomen is het niet toestaan van dergelijke instructies. Het is mogelijk zo te microprogrammeren dat het geselecteerde CC/ bit nooit van de directe status inputs I afhankelijk is. Hierdoor wordt de programmeur een beperking opgelegd. Er is echter een betere manier.

Vaak is de microprogrammeur in staat een goede voorspelling voor het CC/ bit te doen. Als we nu in het Pipeline Register voor deze voorspelling een bit reserveren (Prediction Bit : PR) dan kunnen we m.b.v. dit bit reeds bij de start van

een klokcyclus beginnen met de adresberekening en het ophalen van de volgende uit te voeren micro-instructie. Men dient wel te controleren of de voorspelling juist is. Bij een foute voorspelling moeten er maatregelen genomen worden om toch het juiste microwoord te selecteren en de klokcyclus eventueel te verlengen.

We zullen nu eerst een aantal tijden definieren om de Prediction logica te kunnen beschrijven. De meeste tijden worden berekend vanaf het tijdstip waarop een nieuwe instructie in het Pipeline Register wordt geplaatst, dus vanaf de opgaande flank van het System Clock signaal (CP).

ALU-tijd

De verwerkingstijd van een instructie in de 2903's is afhankelijk van de opgedrukte instructie. Niet elke instructie heeft dezelfde verwerkingstijd. Deze tijd noemen we de ALU-tijd.

CCVAL-tijd

De tijd die nodig is om een bepaald conditie code bit te selecteren en te laten stabiliseren noemen we de CCVAL-tijd (Condition Code Valid). Nadat deze tijd verstreken is zijn we er zeker van dat het geselecteerde CC/ bit stabiel is; dit is dus het tijdstip waarop we het CC/ bit mogen vergelijken met het Predition Bit.

PRuIF-tijd

De tijd die nodig is om aan de hand van het Prediction Bit het adres van een microwoord te bepalen en dit microwoord klaar te zetten om in het Pipeline Register te worden ingeklokt noemen we de PRuIF-tijd (Predicted Micro Instruc-

tion Fetch).

NuIF-tijd

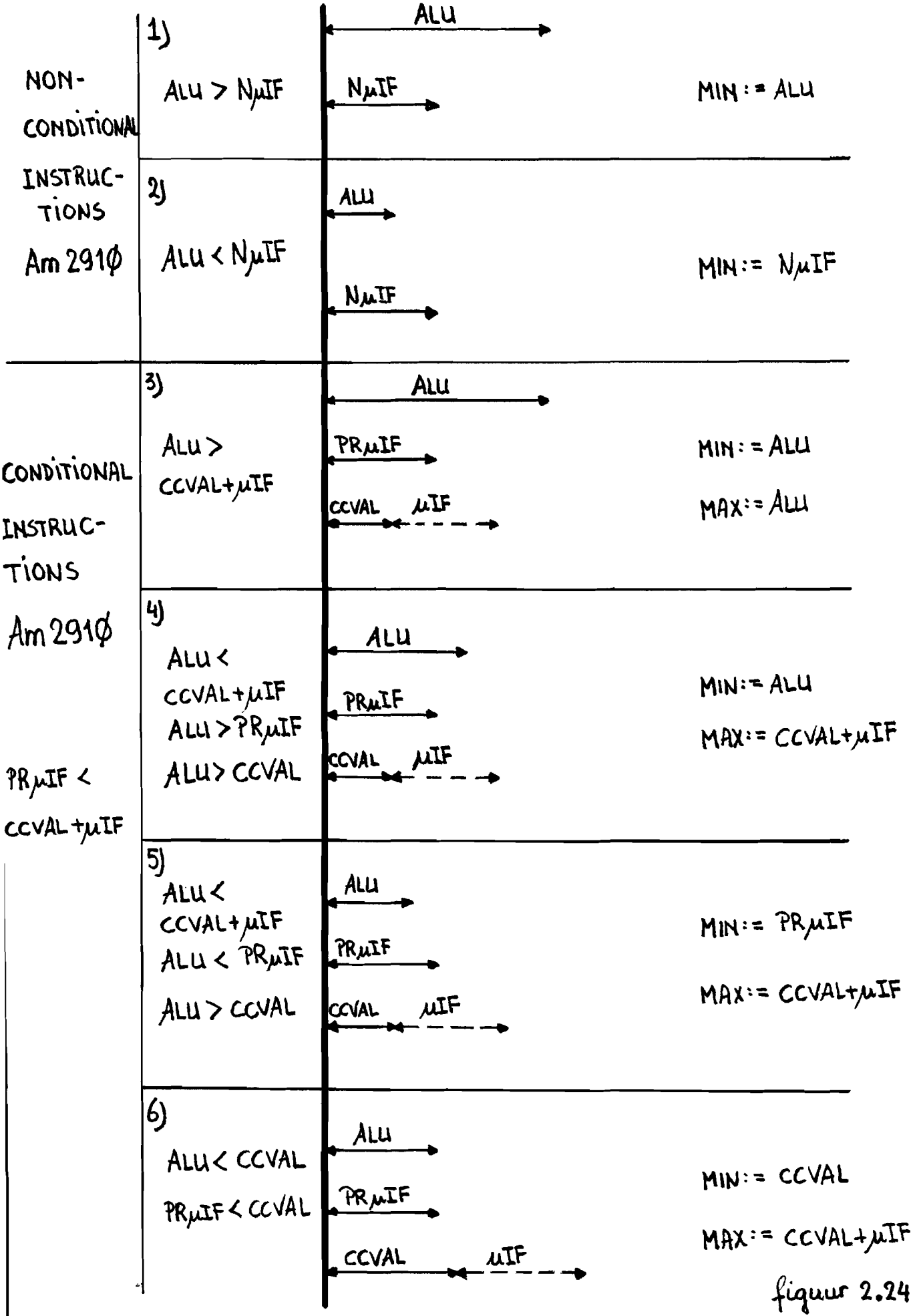
De NuIF-tijd (Non-conditional Micro-Instruction Fetch) valt in dezelfde categorie tijden als de PRuIF-tijd met het verschil dat hier een niet-conditionele instructie aan de 2910 wordt opgedrukt.

uIF-tijd

De uIF-tijd (Micro-Instruction Fetch) valt eveneens in dezelfde categorie tijden als de PRuIF-tijd, nu echter niet berekend vanaf het begin van een klokcyclus maar vanaf het tijdstip waarop tijdens de klokcyclus de polariteit van het aan de 2910 aangeboden CC/ bit omkeert (Prediction false).

Per micro-instructie willen we een minimale klokcyclus lengte bepalen. We kunnen nu een aantal gevallen onderscheiden. De eerste onderverdeling is in conditionele en niet-conditionele instructies. Vervolgens kunnen we de verwerkingstijden van de verschillende micro-operaties ten opzichte van elkaar laten varieren. Hierbij geldt de voorwaarde dat de PRuIF-tijd altijd korter is dan de CCVAL-tijd plus de uIF-tijd. Dit komt doordat de PRuIF- en de uIF-tijd ongeveer even lang zijn. In figuur 2.24 zijn verschillende combinaties weergegeven. Bij de conditionele instructies zijn de CCVAL- en de uIF-tijd achter elkaar getekend daar de bijbehorende micro-operaties sequentieel uitgevoerd worden. Indien de voorspelling van het CC/ bit juist is, dan hoeft de micro-operatie uIF niet meer uitgevoerd te worden. We kunnen nu twee belangrijke tijden definieren: MIN en MAX.

VERWERKINGSTIJDEN VAN VERSCHILLENDE MICRO-INSTRUCTIES



figuur 2.24

MIN-tijd

MIN is de verwerkingstijd van een microwoord als de voorspelling van het CC/ bit goed is.

MAX-tijd

MAX is de verwerkingstijd van een microwoord als de voorspelling van het CC/ bit fout is.

Bij een niet-conditionele instructie ligt de lengte van de klokcyclus vast en hebben we slechts een tijdsindicatie nodig. Hier wordt MIN gedefinieerd als:

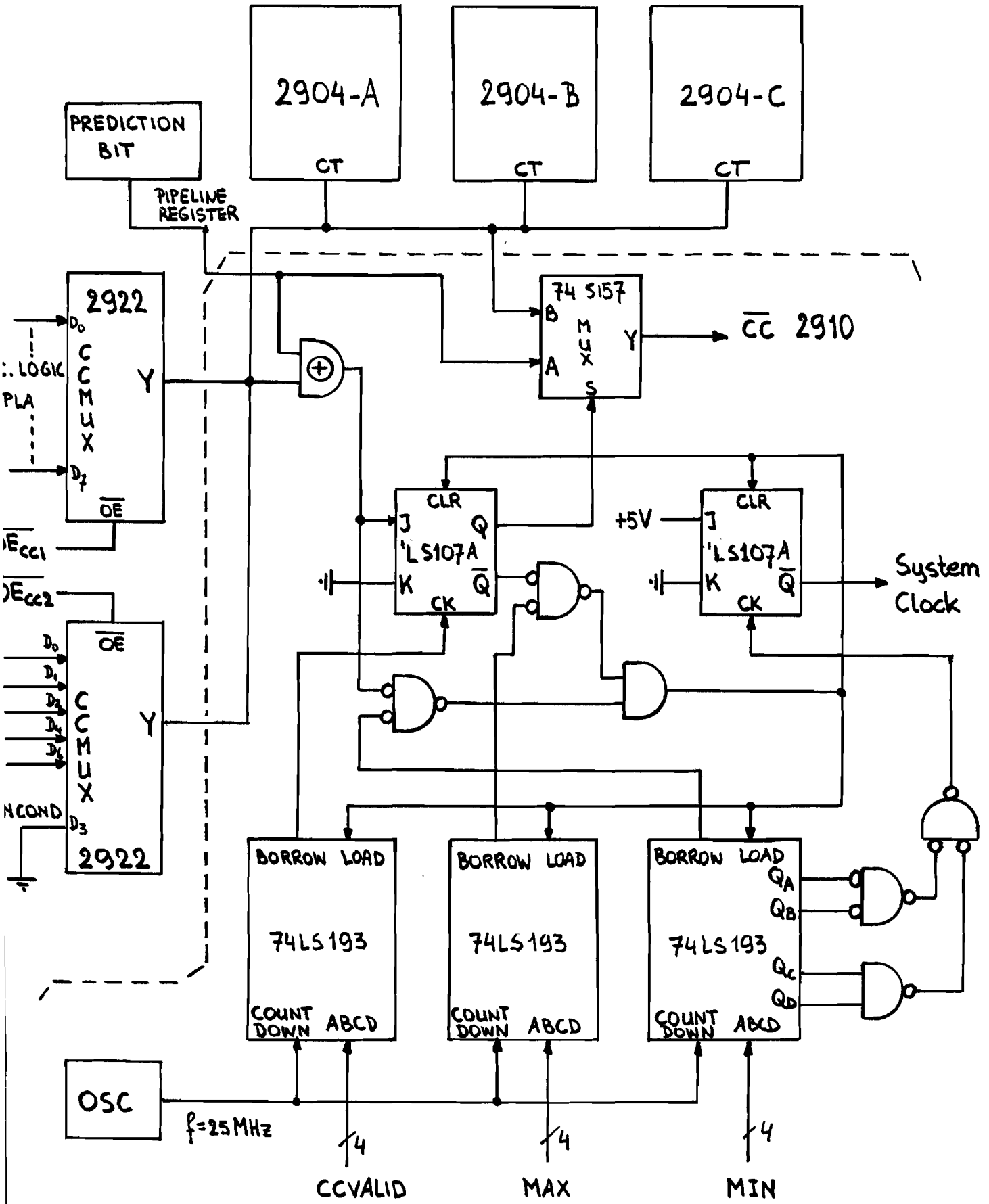
MIN := maximum (ALU,NuIF)

Bij een conditionele instructie zijn de waarden van MIN en MAX als volgt vastgelegd:

MIN := maximum (ALU,CCVAL,PRuIF)

MAX := maximum (ALU,CCVAL + uIF)

In figuur 2.24 zijn bij de verschillende combinaties ook de waarden van MIN en MAX weergegeven. Figuur 2.25 geeft aan hoe we dit in hardware kunnen realiseren. In elk microwoord zijn een drietal velden gereserveerd: CCVAL, MIN en MAX. De tijden staan in deze velden weergegeven als een getal tussen 0 en 15. Elk getal geeft een aantal klokperioden van de grondfrequentie aan (f_0 = 25 MHz). Het veld MIN moet in elke micro-instructie ingevuld zijn daar elke instructie een bepaalde verwerkingstijd heeft. De velden CCVAL en MAX krijgen een "default" waarde van 15 (de maximale tellerstand).



PREDICTION & CLOCK GENERATION LOGIC

figuur 2.25

In het ontwerp zijn drie 74LS193's opgenomen. Dit zijn synchrone, binaire 4 bits up/down counters, die hier als down counter worden toegepast. Als zo'n counter naar 0 is geteld dan wordt de Borrow-output enige tijd "laag". In het begin van elke klokcyclus worden de waarden van CCVAL, MIN en MAX elk in zo'n teller geklokt. Daar deze tellers een geheugenfunctie bezitten laten we ze als onderdeel van de Pipeline fungeren. In het begin van elke klokcyclus worden ook beide JK-flipflops gereset, zie fig. 2.25. De multiplexer (74S157) wordt hierdoor in toestand A geplaatst; d.w.z. het Prediction Bit wordt geselecteerd en aan de 2910 CC/ ingang aangeboden.

Zodra de CCVAL-teller naar nul geteld is en dus een Borrow-puls heeft afgegeven, weten we dat het geselecteerde conditie code bit geldig is. Dit bit wordt d.m.v. een EXOR vergeleken met het Prediction Bit. Het resultaat van deze test wordt in een JK-flipflop opgeslagen. Was de voorspelling juist dan staat er achter de EXOR een "0". Deze "0" zorgt ervoor dat de JK-flipflop in de begintoestand blijft en dat de Borrow-puls van de MIN-teller door een OR-poort gelaten wordt en aldus de klokcyclus kan beeindigen. De klokcyclus wordt beeindigd door de System Clock flipflop te resetten waardoor het System Clock signaal "hoog" gemaakt wordt.

Was de voorspelling fout dan staat er achter de EXOR een "1". Deze "1" zorgt ervoor dat de JK-flipflop omklapt en dat de OR-poort de Borrow-puls van de MIN-teller niet doorlaat. Doordat de JK-flipflop omklapt wordt de Q-output "1". Hierdoor klapt vervolgens ook de multiplexer om en wordt het correcte CC/ bit aan de 2910 aangeboden. De Q/-ouput van de JK-flipflop zorgt ervoor dat een Borrow-puls afkomstig van de MAX- teller doorgelaten wordt om de klokcyclus te beeindigen.

Bij een niet-conditionele instructie willen we dat de Borrow-puls van de MIN-teller de klokcyclus beeindigt. Hiervoor moet er achter de EXOR een "0" staan. Dit kunnen we bereiken door in het microprogramma het te selecteren CC/ bit en het Prediction Bit dezelfde "default" waarden toe te kennen.

De tweede JK-flipflop dient voor de System Clock generatie. Enige componenten in de Am2900 familie eisen namelijk een minimale "hoog"- en "laag"-tijd van hun kloksignaal. We komen hierop in de paragraaf over de timing (2.3.13) nog uitvoerig terug.

Het was ook mogelijk geweest om de variabele klokcyclus te realiseren met de Am2925. De Am2925, Clock Generator and Microcycle Length Controller, kan 8 verschillende cycluslengten genereren varierend van 3 tot 10 maal de lengte van de basisklokperiode. Er zijn tegelijkertijd vier System Clock outputs van dezelfde lengte maar met verschillende "duty-cycle" beschikbaar. Verder zijn er nog voorzieningen zoals RUN/HALT control, Single Step control en Wait control. De voornaamste argumenten om de Am2925 niet toe te passen in dit ontwerp zijn:

- De Am2925 kent geen Reset. Als een klokcyclus eenmaal gestart is dan moet deze helemaal worden afgemaakt, ook als er tussentijds enkele wait-states zijn ingelast.

- Als een wait-state wordt aangevraagd, dan duurt het minimaal een basisklokperiode alvorens de aanvraag gehonoreerd wordt. Het weer verlaten van de wait-state gaat eveneens met een vertraging van een basisklokperiode gepaard.

- In het huidige ontwerp kunnen we 16 verschillende cycluslengten programmeren tegen 8 in de Am2925.

- In de huidige opzet hebben we drie telfuncties nodig: CCVAL, MIN en MAX. De Am2925 bezit slechts een telfunctie. Indien men de Am2925 toch wil toepassen, dan zou men twee extra tellers nodig hebben en dus geen materiaalbesparing t.o.v. het huidige ontwerp verkrijgen.

2.3.12 Schematisch overzicht van de ALU hardware

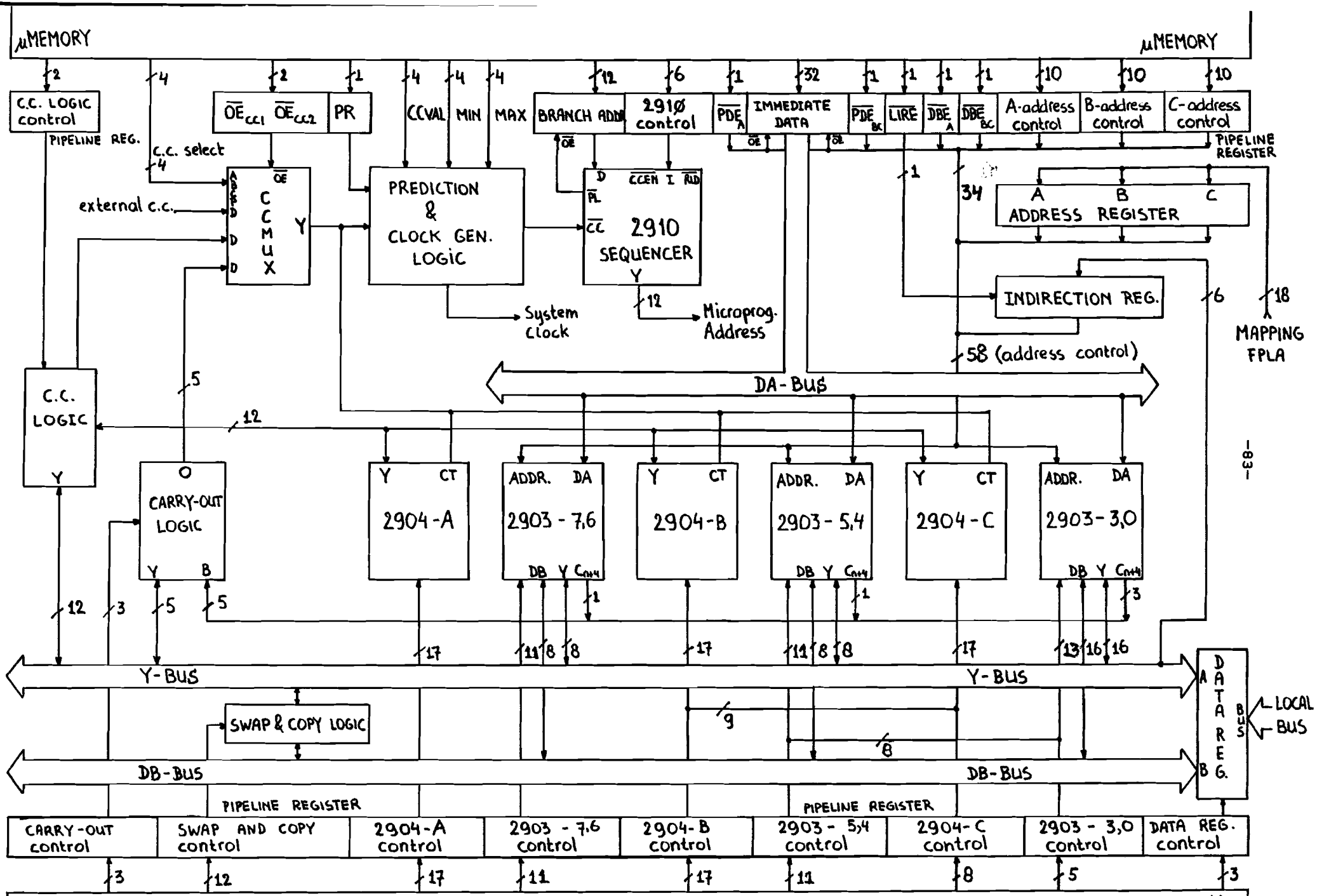
In figuur 2.26 is een schematisch overzicht van de hardware van de Arithmetic and Logic Unit weergegeven. De blokken, die in dit schema de 2903-nummers dragen, bevatten naast de Am2903 processoren ook de uitgebreide Register File, de Carry lookahead hardware, de Carry-in multiplexers, de schakelaars ten behoeve van de Z-lijn en de Schuif Verbindingen, de besturingslogica ten behoeve van de 2903's en de adresseringslogica van de Register File.

In het Pipeline Register zijn alle besturingsvelden van de diverse bouwstenen opgenomen. In de tabel van figuur 3.1 (zie hoofdstuk 3) wordt de samenstelling van elk besturingsveld vermeld. Als Pipeline Register kan de Am2954 gebruikt worden. Dit is een 8 bits register dat is opgebouwd uit D-flipflops met een gemeenschappelijke klok. Data wordt ingeklokt op de positieve flank van het kloksignaal.

Niet opgenomen in het schema zijn de drie ALU configuratie besturingslijnen 4-Bytes/, 3-Bytes/ en 2-Bytes/ met hun bijbehorende besturingslogica. Ook niet het "Operation Complete" signaal dat aan de CCU meedeelt dat een instructie afgewerkt is; hiervoor is nog een extra Pipeline bit nodig. De interface tussen de ALU en de Local Bus is nog niet verder uitgewerkt omdat de Local Bus is nog niet volledig gedefinieerd is.

Het door de Clock Generation Logic gegenereerde signaal System Clock dient als kloksignaal voor de hele ALU. In de diverse schema's wordt dit sigaal ook wel als CP (Clock Pulse) aangeduid.

De velden CC Select, CCVAL, MIN en MAX hebben geen apart Pipeline Register daar de bouwstenen die ze besturen intern een geheugenfunctie bezitten. Hiermee verzorgen deze bouwstenen zelf de Pipeline functie.



SCHEMATISCH OVERZICHT VAN DE ALU HARDWARE

figuur 2.26

De breedte van een microwoord bedraagt 196 bits. Om het microprogrammageheugen op te bouwen zijn, indien we uitgaan van byte-wide PROM's zoals de 3636 van Intel, minimaal 25 circuits nodig. Bij volledige bezetting (4K microwoorden) worden dit er zelfs 50. Het aantal benodigde Pipeline Register circuits (Am2954) bedraagt 23. De firmware die voor deze ALU geschreven werd (zie hoofdstuk 3) omvat ongeveer 100 microwoorden. Het is onwaarschijnlijk dat men in de toekomst de volledige bezetting (4K microwoorden) nodig heeft. Indien men het aantal circuits wil beperken dan kan men ook PROM's toepassen die intern al een data register bevatten zoals bijvoorbeeld de Am27S27 (512 x 8 bits).

2.3.13 Tijdberekeningen

Een belangrijk aspect bij het ontwerp van een microprogrammeerbaar systeem met variabele klokperiode is het berekenen van de optimale klokcycluslengte bij elke micro-instructie. Zoals we in paragraaf 2.3.11 gezien hebben moeten we een aantal belangrijke tijden berekenen; deze worden nu achtereenvolgens behandeld. Bij alle berekeningen wordt gerekend met de maximale vertragingstijden van de componenten om tot een "worst case" tijd te komen.

ALU-tijd

Bij veel ALU-instructies wordt er eerst data gelezen uit de Register File. Op de neergaande flank van het kloksignaal worden de output-latches van de Register File gesloten. We willen in verdere tijdberekeningen geen onderscheid maken of een register zich in een 2903 of in een 29705 bevindt. Alle 64 registers willen we gelijkwaardig behandelen. We gebruiken daarom de langste tijd van beide gevallen. De minimale "hoog"-tijd van het kloksignaal CP wordt bepaald door de tijd

die verloopt tussen de opgaande flank van CP en het stabiel zijn van de geselecteerde data in de output-latches van de Register File. In de figuren 2.17 en 2.18 zien we welke bouwstenen voor de vertragingen verantwoordelijk zijn. Figuur 2.27 geeft een tabel met de betreffende vertragingstijden weer. De eerste kolom bevat de tijden voor een register in een 2903, de tweede kolom voor een register in een 29705. Alle tijden worden in nanoseconden weergegeven. De minimale "hoog"-tijd van CP bedraagt dus 87,5 ns.

Device Nr.	Device Path	Path 1	Path 2
2954	CP naar Y	17	17
74S153	S naar Y	18	18
74S157	A naar Y	7,5	7,5
2903	B set up	27	-
29705	B set up	-	45
		+	+
Total	ns	69,5	87,5

Berekening van de minimale "hoog"-tijd van CP.

figuur 2.27

Evenals een minimale "hoog"-tijd kennen we ook een minimale "laag"-tijd van het kloksignaal CP. Als we willen schrijven in de Register File dan dient het IEN/ signaal "laag" te zijn. IEN/ wordt "laag" nadat het kloksignaal CP "laag" geworden is. We berekenen de tijd die verloopt tussen het "laag" worden van IEN/ en het weer "hoog" worden van het WE/ signaal van de Register File. Door figuur 2.15 en 2.17 te combineren zien we welke componenten er voor een vertraging zorgen. In de tabel van figuur 2.28 zijn de tijden weergegeven; ook hier weer een kolom voor de 2903 en een voor de 29705. De berekeningen zijn gemaakt t.a.v. 2903-6.

Device Nr.	Device Path	Path 1	Path 2
74S32	A,B naar Y	7	7
2903	IEN/ naar MSS/	22	22
74S157	B naar Y	6,5	6,5
74S32	A,B naar Y	7	7
74S139	G naar D	10	10
2903	WE/ set up	30	-
29705	WE/ set up	-	25
		———	——— +
Total	ns	82,5	77,5

Berekening van de minimale "laag"-tijd van CP.

figuur 2.28

De minimale "laag"-tijd van CP bedraagt 82,5 ns. De minimale klokcyclustijd bedraagt 82,5 + 87,5 = 170 ns. Om het kloksignaal te laten voldoen aan de minimale "laag"-tijd laten we de System Clock flipflop omklappen als de MIN-teller een bepaalde tellerstand bereikt heeft. In de volgende alinea's zullen we deze tellerstand X berekenen.

Op zijn vroegst wordt de System Clock flipflop gereset door de Borrow-puls van de MIN-teller. We gaan nu de gerealiseerde "laag"-tijd van CP uitrekenen. Hiertoe definieren we de tijden T1 en T2. Beide worden ze berekend vanaf de positieve flank van het Count Down signaal.

- T1 is de tijd die verloopt vanaf de basisklokperiode waarin de MIN-tellerstand X wordt totdat CP "laag" wordt.
- T2 is de tijd die verloopt vanaf de basisklokperiode waarin de MIN-tellerstand 0 wordt totdat CP weer "hoog" wordt.

Figuur 2.29 en 2.30 bevatten de berekeningen van respectievelijk T1 en T2.

Device Nr.	Device Path	Path
74LS193	CD naar Q	47
74S32	A,B naar Y	7
74S32	A,B naar Y	7
74LS107A	CK naar Q/	20
Total	ns	81

Berekening van de tijd T1.

figuur 2.29

Device Nr.	Device Path	Path
74LS193	CD naar Bor	24
74S32	A,B naar Y	7
74S08	A,B naar Y	7,5
74LS107A	CLR naar Q/	20
Total	ns	58,5

Berekening van de tijd T2.

figuur 2.30

De "laag"-tijd van CP bedraagt X maal de basisklokperiode min T1 plus T2. Er moet voldaan worden aan de eis:

$$X * 40 - 81 + 58,5 > 82,5$$

Hieruit volgt dat met X=3 aan deze voorwaarde voldaan wordt. In figuur 2.25 zien we hoe dit gerealiseerd is. Als de voorspelling van het CC/ bit fout is, dan blijft het kloksignaal nog langer "laag". De "hoog"-tijd van CP wordt gerealiseerd door de

MIN-teller een voldoend hoge startwaarde toe te kennen.

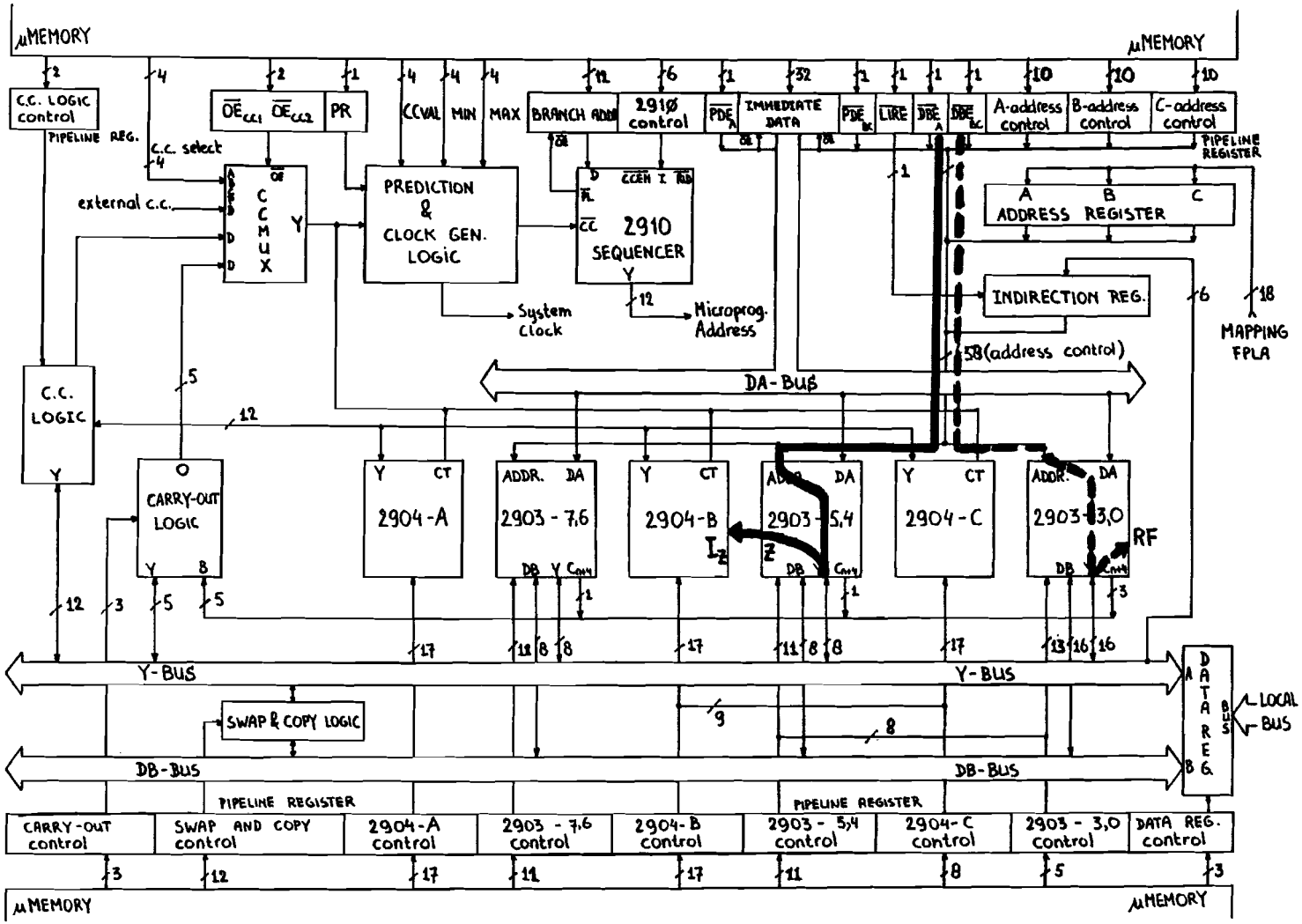
In de literatuur (AM29) zijn de combinatorische vertragingen in de 2903 als volgt gespecificeerd: een tabel voor de standaard ALU functies en een tabel per speciale functie. Voor de speciale functies van de 2903's zijn ook speciale verbindingspaden tussen de 2903's nodig. Deze paden kunnen met behulp van de 2904's gerealiseerd worden. Bij alle berekeningen wordt verondersteld dat de 2903's en de 2904's overeenkomstige instructies ontvangen.

In de figuren 2.31 t/m 2.34 zien we een aantal voorbeelden van berekeningen van ALU-tijden voor verschillende instructies. Deze voorbeelden betreffen allen de Mantisse ALU. In de tabel van figuur 2.35 zijn de ALU-tijden voor de verschillende instructies bij elkaar gezet. Ze zijn voor zowel de Exponent als de Mantisse ALU opgenomen. Indien men de 4-Bytes of de 2-Bytes ALU wenst te bedrijven dan dient men voor deze ALU's eerst nog soortgelijke berekeningen uit te voeren. Men kan de ALU-tijden nog onderverdelen in twee soorten. De eerste ALU-tijd is die tijd die nodig is om het resultaat van een bewerking weg te schrijven in de Register File (R set up). De tweede ALU-tijd is de tijd die nodig is om de statussignalen ten gevolge van een bewerking te laten stabiliseren (S set up). De tweede tijd is in het algemeen iets langer dan de eerste. Beide tijden staan vermeld in de tabel.

CCVAL-tijd

De CCVAL-tijd is de tijd die verstrijkt tussen de opgaande flank van het kloksignaal CP en het stabiel worden van de uitgang van de EXOR die het CC/ bit vergelijkt met het Prediction Bit. Deze tijd is nog afhankelijk van de geselecteerde bron. We kunnen de volgende indeling maken:

1) Het CC/ bit staat klaar voor de CC multiplexer.
2) Het CC/ bit wordt gevormd door de CC logica.



DEVICE NO.	DEVICE PATH	PATH 1	PATH 2
2954	CP→Y	17	17
74S153	S→Y	18	18
29705	A→YA	53	53
2903's	DA→Y	64	64
2903/29705	Y set up	20	-
2903's	Y→Z	-	24
2904-B	I set up	-	17
Totaal	ns	172	193

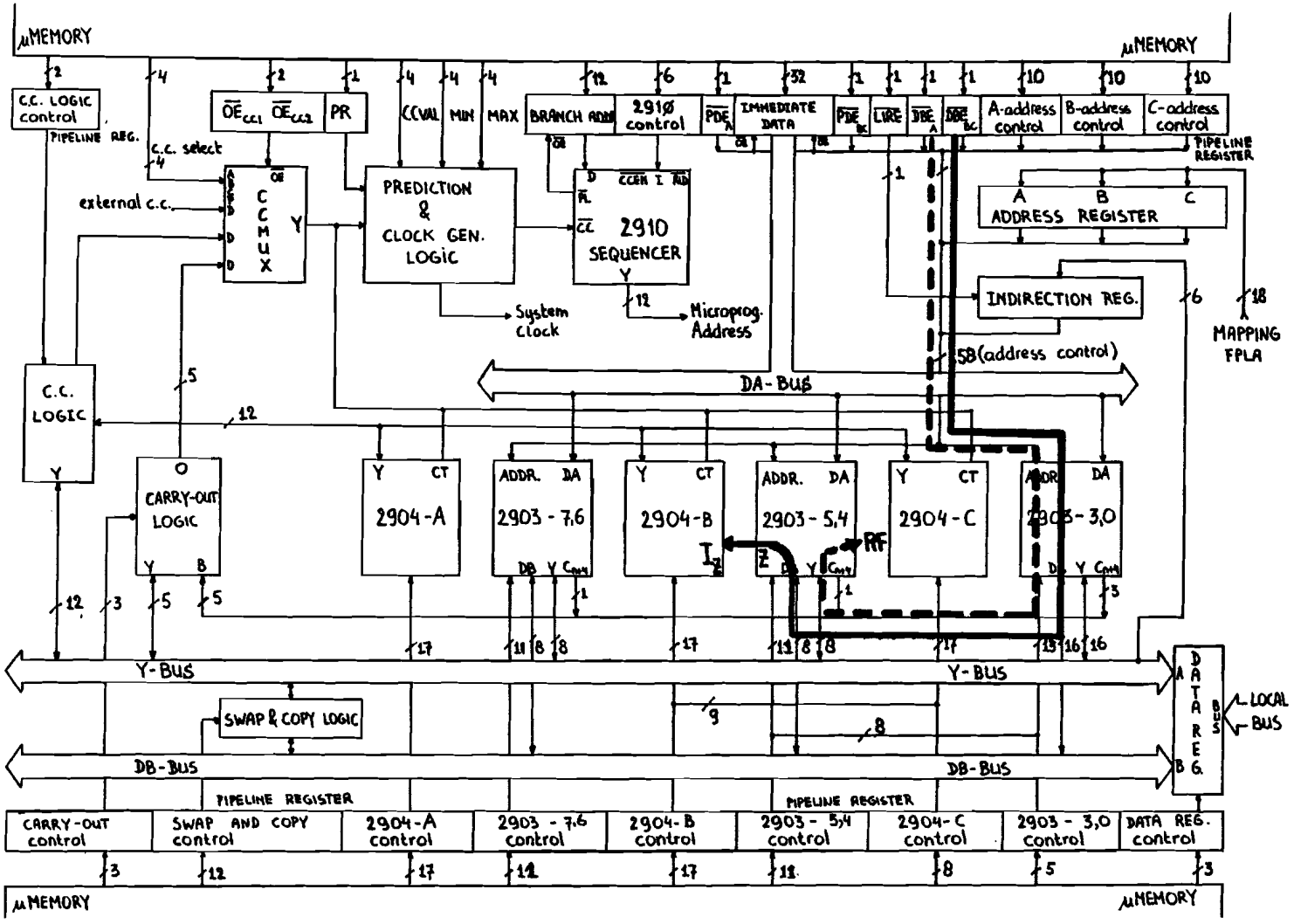
Path 1

Path 2

168

LOGIC OPERATION - NO SHIFT

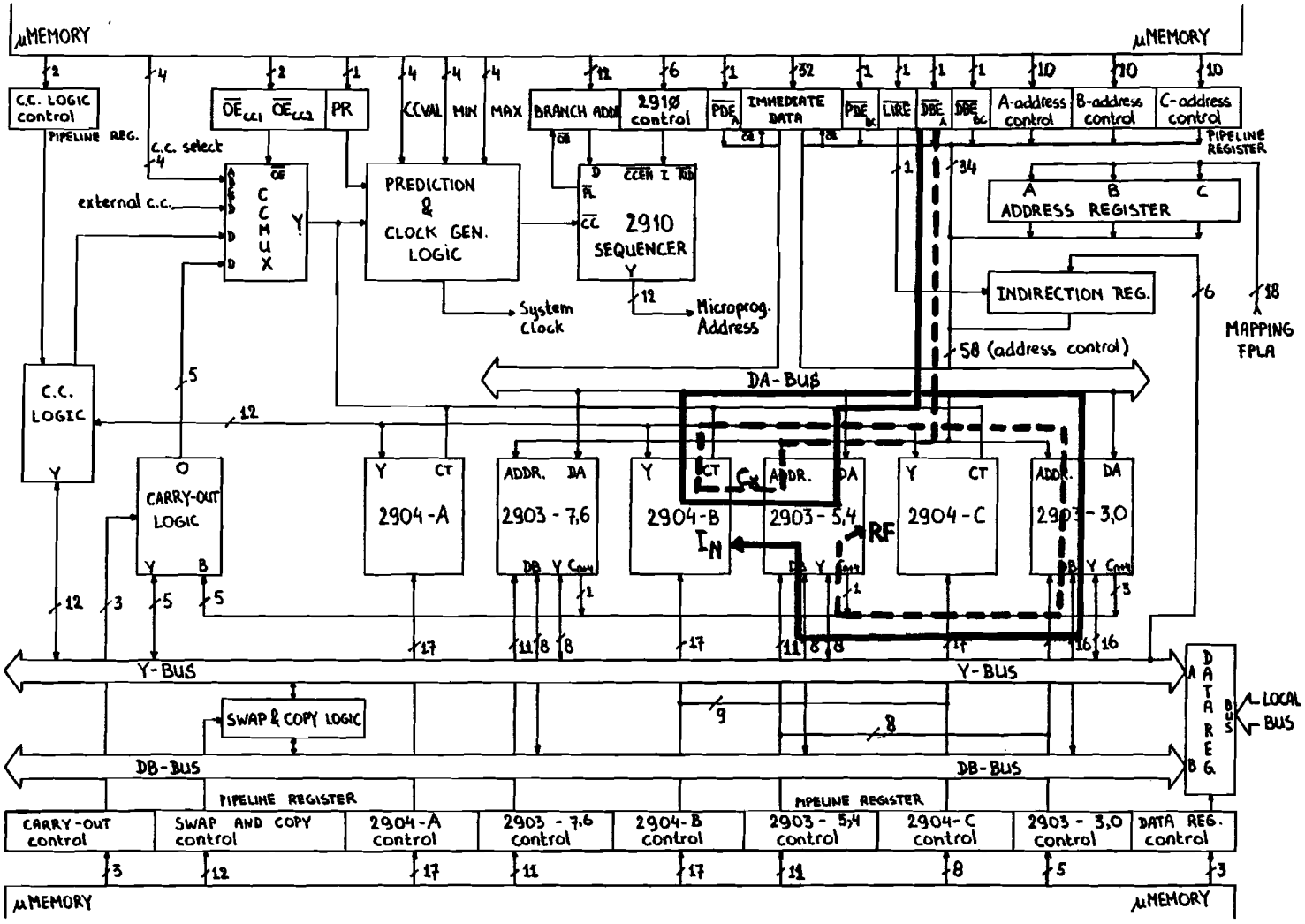
figuur 2.31



DEVICE No.	DEVICE PATH	PATH 1	PATH 2
2954	CP → Y	17	17
74S153	S → Y	18	18
74S157	D → Y	7,5	7,5
29705	B → YB	53	53
2903's	DB → $\overline{P}, \overline{G}$	47	47
2902A-1	$\overline{P_i}, \overline{G_i} \rightarrow \overline{P}, \overline{G}$	12	12
2902A-0	$\overline{P_0}, \overline{G_0} \rightarrow C_{n+x}$	9	9
2902A-2	$C_n \rightarrow C_{n+x}$	14	14
2903-5	$C_n \rightarrow Y$	40	-
2903-5	$C_n \rightarrow Z$	-	64
2903/29705	Y set up	20	-
2904-B	I set up	-	17
Totaal	ns	237,5	258,5

Path 1

Path 2



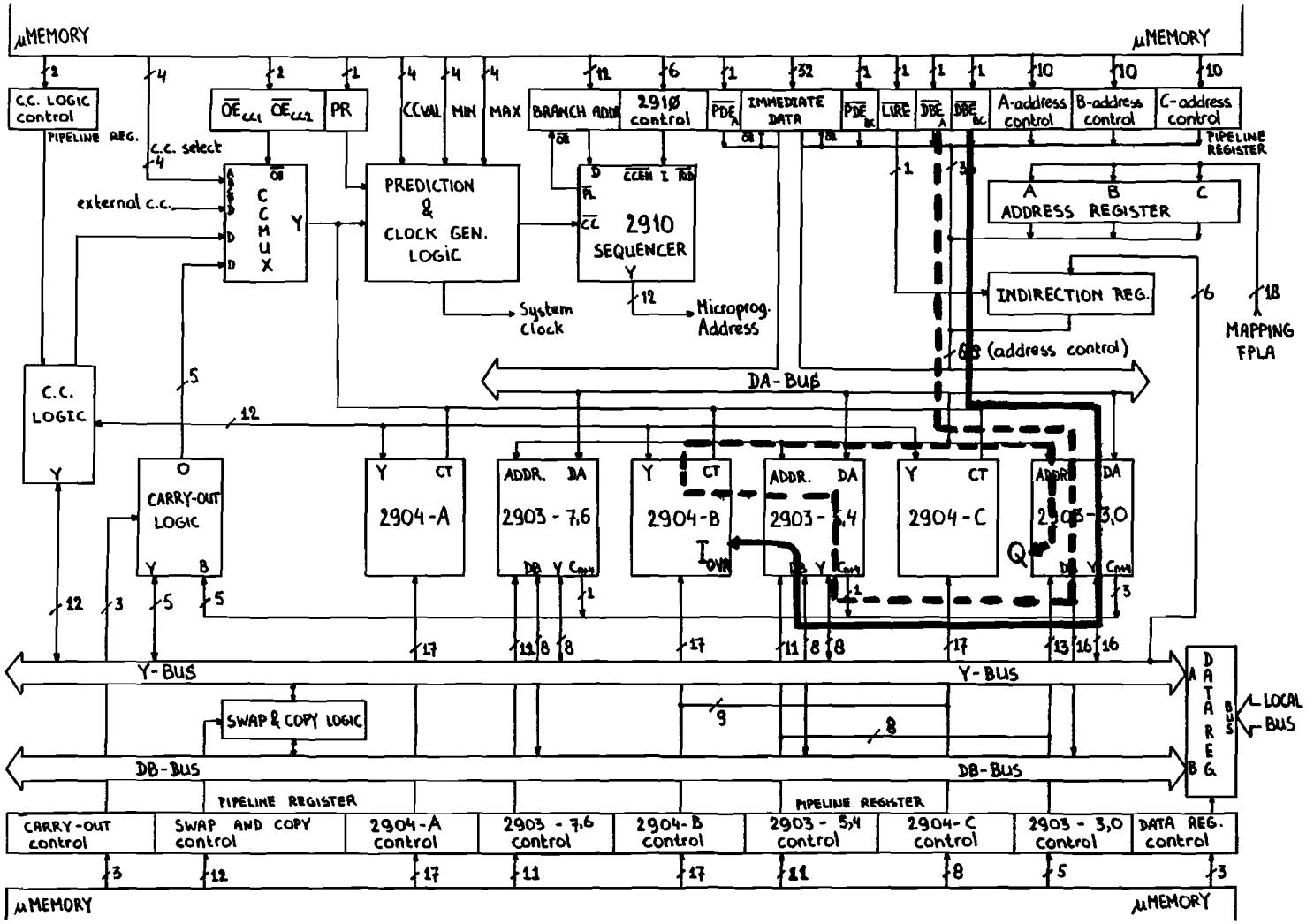
DEVICE NO.	DEVICE PATH	PATH 1	PATH 2
2954	CP→Y	17	17
74S153	S→Y	18	18
74S157	D→Y	7,5	7,5
29705	B→YB	53	53
2903-5	DB→Z	40	40
2904-B	C_x→C_o	20	20
74S153	D→Y	9	9
2902A-0	C_n→C_{n+x}	14	14
2902A-2	C_n→C_{n+x}	14	14
2903-5	C_n→Y	79	-
2903-5	C_n→N	-	79
2903/29705	Y set up	20	-
2904-B	I set up	-	17
Totaal	ns	291,5	288,5

→ Path 2

⇢ Path 1

SIGN/MAGNITUDE - TWO'S COMPLEMENT CONVERSION

figuur 2.33



DEVICE NO.	DEVICE PATH	PATH 1	PATH 2
2954	CP → Y	17	17
74S153	S → Y	18	18
74S157	D → Y	7,5	7,5
29705	B → YB	53	53
2903's	DB → $\overline{P}$,$\overline{G}$	49	49
2902A-1	$\overline{P}_i$,$\overline{G}_i$ → $\overline{P}$,$\overline{G}$	12	12
2902A-0	$\overline{P}_o$,$\overline{G}_o$ → C_{n+x}	9	9
2902A-2	C_n → C_{n+x}	14	14
2903-5	C_n → SIO_3	64	-
2903-5	C_n → OVR	-	58
2904-B	SIO_3 → QIO_0	26	-
CD4066	I/O → I/O	25	-
2903-0	Q set up	20	-
2904-B	I set up	-	17
Totaal	ns	314,5	254,5

Path 1

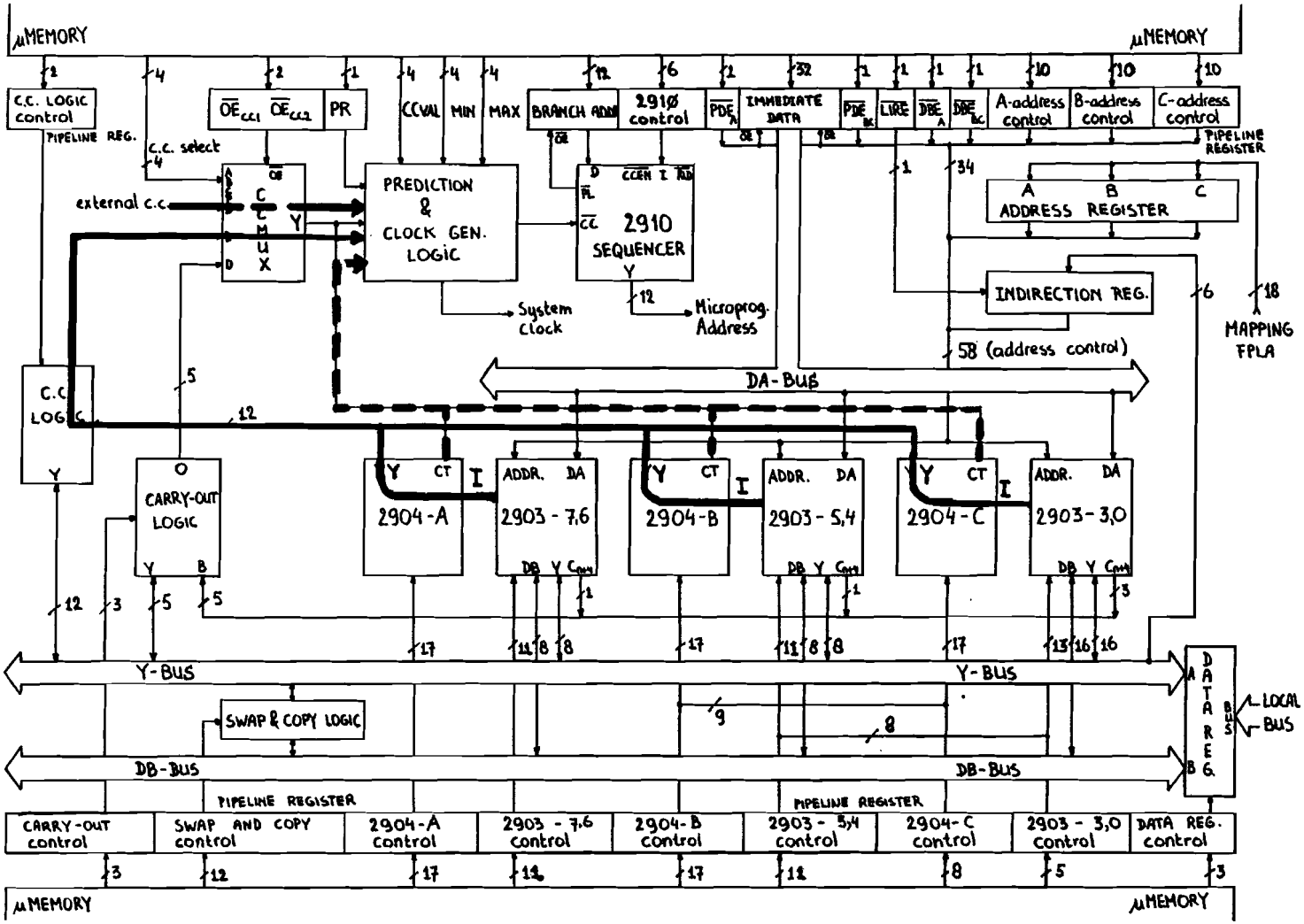
Path 2

TWO'S COMPLEMENT DIVIDE OPERATION

2903-Function	Mantissa ALU		Exponent ALU	
	R set up	S set up	R set up	S set up
Standard Functions				
Logic: No Shift	172	193	172	193
Rotate up	256	277	256	277
Rotate down	244	265	244	265
Arith: No Shift	237,5	258,5	222	243
Shift up	293,5	314,5	222	243
Shift down:				
2's Complement	313,5	334,5	298	319
Sign/Magnitude	296,5	317,5	270	291
Special Functions				
Unsigned multiply	298,5	254,5	228	238
2's Complement multiply	298,5	254,5	241	238
2's Compl mul/Last cycle	263,5	254,5	247	238
Increment by 1 or 2	260,5	239,5	223	244
S/M 2's Compl conversion	291,5	288,5	299,5	296,5
Single length normalize	239,5	254,5	223	238
Double length normalize	295,5	250,5	223	234
First divide operation	318,5	250,5	302	234
2's Complement divide	314,5	254,5	298	238
2's Compl divide C & R	239,5	254,5	200	238

ALU-tijden voor de verschillende 2903-instructies.

figuur 2.35



DEVICE No	DEVICE P.	P1	P2a	P2b	P3a	P3b
2922	CP→Y	40	-	-	-	-
2954	CP→Y	-	17	-	17	-
2904's	I_5I_4→Y	-	43	-	-	-
2904's	I input	-	-	S	-	S
2904's	I→Y	-	-	38	-	-
2904's	I→CT	-	-	-	-	48
FPLA	I→O	-	40	40	-	-
2922	D→Y	-	22	22	-	-
2904's	I_5-I_0→CT	-	-	-	50	-
74586	A,B→Y	10,5	10,5	10,5	10,5	10,5
Totaal	ns	50,5	132,5	S+110,5	77,5	S+58,5

Path 1
Path 2
Path 3

CCVAL - TIJDEN

3) Het CC/ bit is afkomstig van een 2904 CT-output.

ad 1) De te testen grootheid kan een extern conditie code bit zijn of een bit uit het Carry-out Register.

ad 2,3) De laatste twee categorien kunnen we elk nog in twee gevallen onderverdelen. Deze onderverdeling geeft aan of de statusbits, waarop getest wordt, afkomstig zijn uit de vorige of uit de momentane micro-instructie.

a) Het CC/ bit wordt afgeleid uit de inhoud van een of meerdere status registers.
b) Het CC/ bit wordt afgeleid uit de directe status inputs of uit een combinatie van bits afkomstig uit de status registers en de directe status inputs.

In figuur 2.36 zijn de vertragingspaden aangegeven en in de tabel staat de berekening van de verschillende tijden. De benaming van de kolommen komt overeen met de bovenstaande categorien: 1, 2a, 2b, 3a, en 3b.

NuIF-, PRuIF- en uIF-tijden

De NuIF- en de PRuIF-tijden zijn afhankelijk van de instructies die aan de Am2910 worden opgedrukt. We kunnen de 16 verschillende instructies van de 2910 opdelen in drie klassen:

1) I_3-I_0 = 8,9 of 15. Dit zijn de instructies waar het interne register als counter gebruikt wordt.
2) I_3-I_0 = 1,2,3,5,6,7 of 11. In deze instructies kan de input D als leverancier van het adres voor de volgende uit te voeren micro-instructie geselecteerd worden.
3) I_3-I_0 = 0,4,10,12,13 of 14. Overige instructies.

In de tabel van figuur 2.37 zijn de berekeningen van de NuIF- en de PRuIF-tijden in deze drie klassen weergegeven.

De uIF-tijd is niet afhankelijk van de opgedrukte instructie. Deze tijd wordt berekend vanaf het tijdstip dat een foute voorspelling geconstateerd is, dus vanaf het tijdstip dat de CCVAL-teller een Borrow-puls afgeeft die ervoor zorgt dat de JK-flipflop omklapt. Zie figuur 2.25.
Alle inputs van de 2910 zijn dan al enige tijd stabiel en alleen de CC/ input verandert nog. De uIF-tijd is dus een vaste tijd. In de tabel van figuur 2.37 is ook de berekening van de uIF-tijd opgenomen.

Device Nr.	Device Path	NuIF en PRuIF			uIF
		Kl.1	Kl.2	Kl.3	
2954	CP naar Y	-	17	17	-
2910	CP naar Y	125	-	-	-
2910	I naar PL/	-	51	-	-
2954	OE/ naar Y	-	18	-	-
2910	D naar Y	-	20	-	-
2910	I naar Y	-	-	70	-
74LS107A	CK naar Q/	-	-	-	20
74S157	S naar Y	-	-	-	12,5
2910	CC/ naar Y	-	-	-	43
3636	A naar O	80	80	80	80
2954	D set up	5	5	5	5
Total	ns	210	191	172	160,5

Berekening van NuIF-, PRuIF- en uIF-tijden.

figuur 2.37

Met behulp van de formules voor MIN en MAX zoals die in paragraaf 2.3.11 zijn afgeleid en de tijden die in deze paragraaf zijn berekend kan men de CCVAL-, MIN- en MAX-tijden berekenen. Als men nu deze tijden door 40 deelt en de uitkomsten

naar boven afrondt, dan vindt men de waarde die in de overeenkomstige velden van het microwoord ingevuld dienen te worden.

3 Ontwerp Software voor Floating Point ALU

3.1 Inleiding

Om op de machine, beschreven in hoofdstuk 2, een rekenorgaan te definieren is firmware (microprogrammatuur in ROM) benodigd. Om deze firmware op een behoorlijke manier te ontwikkelen is een symbolische vertaler (b.v. een micro-assembler) welkom. Daar een dergelijke vertaler niet beschikbaar was, hebben we zelf een assembler geconstrueerd, zie par. 3.2. De geschreven software bevat Floating Point routines voor zowel een register machine (par. 3.3) als stack machine (par. 3.4). Slechts enkele routines zijn daadwerkelijk in een microprogramma omgezet.

3.2 De constructie van een assembler.

3.2.1 Inleiding.

Om de software behorende bij de microprogrammeerbare machine te ontwikkelen en te documenteren hebben we gebruik gemaakt van een zelf ontworpen assembler. Deze assembler is geconstrueerd m.b.v. de meta-asembler AMDASM 29, een programma dat draait onder het AMDOS 29 Operating System (Extended CP/M, zie lit. (AMDS)).

In dit hoofdstuk komen verder de volgende onderwerpen aan bod:

par. 3.2.2: AMDASM 29 meta-assembler.
Hierin worden de voornaamste eigenschappen van deze assembler belicht.
par. 3.2.3: De assembler.
In deze paragraaf wordt de bouw van de assembler besproken en wordt tevens de definitieve indeling van het microwoord gepresenteerd.

3.2.2 AMDASM 29 meta-assembler

Een meta-assembler verschilt van een "gewone" assembler in het feit dat de meeste symbolen door de gebruiker zelf worden gedefinieerd voordat het eigenlijke assembleer-proces plaats vindt. Hoewel bij gewone assemblers de mogelijkheid bestaat symbolen voor bepaalde datawoorden te definieren geldt niet dat b.v. de instructies, inclusief lengte en formaat, door de gebruiker vastgelegd kunnen worden. Een dergelijke assembler is dan ook ontworpen om een verzameling vastgestelde formaten te converteren naar machinetaal voor een bepaalde machine.

Een micro-assembler daarentegen is flexibeler dan een traditionele assembler, omdat hij bruikbaar moet zijn in vele, van elkaar afwijkende, hardware configuraties.

Punten waarin een micro-assembler verschilt van een "gewone" assembler zijn:

1) Definitie van instructie-formaten.
2) Grotere woordbreedte.
3) Meerdere instructie-formaten benodigd voor het samenstellen van een micro-instruktie.

Vanwege deze punten wordt voor een micro-assembler meestal een meta-assembler gekozen.

De meta-assembler AMDASM 29 kent 2 fasen:

fase 1) Definition Phase.

fase 2) Assembly Phase.

De Assembly Phase lijkt veel op een gewone assembler. De source tekst wordt gelezen, labeling, het setten van de address counter en andere pseudo-opdrachten zijn mogelijk. Deze fase produceert een object code, verscheidene listings en cross-reference tables.

De Definition Phase wordt het eerst uitgevoerd. Het doel is tabellen te vullen waarin de symbolen van de gebruiker geassocieerd kunnen worden met de overeenkomstige bitpatronen. De Definition Phase stelt de gebruiker in staat om symbolen voor de verschillende formaten (format names) en symbolen voor constanten (constant names) te definieren. Verder kan ook de breedte van het microwoord worden gespecificeerd. Een micro-woord mag een breedte bezitten varierende van 1 tot 128 bitposities.

Elk van de door de gebruiker gedefinieerde velden is geassocieerd met een specifiek bitpatroon. Een format name wordt gebruikt om een micro-instruktie of een gedeelte daarvan te definieren. Een format definitie kan worden opgebouwd met de volgende velden:

- Numerieke velden. Deze definieren een specifiek bitpatroon binnen het micro-woord.
- Variabelen. Deze worden pas ingevuld indien het betreffende formaat tijdens de Assembly Phase wordt gespecificeerd.
- "Don't Care" states.

De "don't care" states worden zolang mogelijk in stand gehouden. Pas na een derde fase (Postprocessing Phase) worden deze toegewezen. De assembler noteert dan ook in de binary output listing een "X" voor elk niet gedefinieerd bit. Men kan dit gebruiken om, na assemblage, de breedte van het microwoord

te minimaliseren door b.v. invoering van "shared fields".

Resumerend kunnen we stellen dat:

1) De micro-assembler wordt gedefinieerd door de Definition Phase van AMDASM 29.
2) De microprogrammatuur kan daarna m.b.v. deze assembler worden vertaald.

3.2.3 De assembler

Om de software, geschreven voor de microprogrammeerbare machine, te ontwikkelen hebben we een assembler ontworpen. Vanwege het feit dat we het aantal mnemonics tot een aanvaardbaar minimum wilden beperken waren we genoodzaakt van de triviale indeling af te wijken. Onder de triviale indeling verstaan we die indeling van velden waarvan de bits gekoppeld zijn aan IC's i.p.v. gekoppeld aan functies. De triviale indeling wordt gegeven in de tabel van fig. 3.1.

Voor het indelen van de velden binnen een microwoord, zodanig dat deze velden gekoppeld zijn aan een specifieke functie, kunnen we de volgende regels opstellen:

1) Een veld dat wordt gevuld d.m.v. een mnemonic dient eerder te worden gedefinieerd dan de veld(en) die worden gevuld door de argumenten van deze mnemonic. Dit vanwege het feit dat de assembler een vaste substitutie volgorde kent van links naar rechts.
2) Twee velden van ongelijke lengte, waarvan de indelingen echter vrij goed met elkaar overeenkomen, kunnen worden voorzien van een en dezelfde mnemonic. Men definieert hiertoe de mnemonic d.m.v. het grootste veld en wel zodanig dat het gemeenschappelijk bitpatroon zich rechts in dat veld bevindt. Gebruikt men deze mnemonic voor het vullen van het

naam van het veld	indeling van het veld
2903-(7,6) control	$I_{8A}...I_{0A}$,IEN_A/,OE_A/
2903-(5,6) control	$I_{8BC}...I_{6BC}$,I_{5B}, $I_{4BC}...I_{0BC}$,OE_{YB}/,IEN_B/
2903-(3,0) control	I_{5C1},I_{5C2},OE_{YC1}/,OE_{YC2}/, IEN_C/
2904-A control	$I_{12A}...I_{0A}$,SE_A/,CE_{UA}/, CE_{MA}/,OE_{CTA}/
2904-B control	$I_{12B}...I_{0B}$,SE_B/,CE_{UB}/, CE_{MB}/,OE_{CTB}/
2904-C control	$I_{9C}...I_{6C}$,SE_C/,CE_{UC}/, CE_{MC}/,OE_{CTC}/
Swap & Copy control	(T_1,OE_1/)...(T_6,OE_6/)
Immediate data	$D_{31}...D_0$
Data enable control	PDE_A/,PDE_{BC}/,DBE_A/, DBE_{BC}/,LIRE/
A-address control	$A_5...A_0$,(S_{A1},S_{A0}-(7,6)), (S_{A1},S_{A0}-(5,0))
B-address control	$B_5...B_0$,(S_{B1},S_{B0}-(7,6)), (S_{B1},S_{B0}-(5,0))
C-address control	$C_5...C_0$,(S_{C1},S_{C0}-(7,6)), (S_{C1},S_{C0}-(5,0))
CC-Multiplex control	A,B,C,POL,OE_{CC2}/,OE_{CC1}/
2910 control	$I_3...I_0$,CCEN/,RLD/, $BRCNT_{11}...BRCNT_0$
Carry Out control	COLE/,COOE/,S_{CO}
CC Logic Control	T_{YBSEP},OE_{YBSEP}/
Data Register control	DCRPE/,S_{DR},OE_{DR}/
Prediction and Timing control	PR,$CCVAL_3...CCVAL_0$, $MAX_3...MAX_0$,$MIN_3...MIN_0$
ALU configuration control	4-BYTES/,3-BYTES/,2-BYTES/

figuur 3.1

het het kleinere veld dan kan m.b.v. een "left truncation"-operatie het bitpatroon passend gemaakt worden.

3) Zijn er echter enkele bits die wel in het kleinere maar niet in het grotere veld voorkomen dan kan toch een enkele mnemonic voor beide velden gekozen worden. De mnemonic wordt wederom gedefinieerd door het bitpatroon van het grootste veld met uitzondering van die bits die niet gemeenschappelijk zijn. Deze laatste bits kunnen worden gedefinieerd d.m.v. argumenten die op het bitpatroon, in samenwerking met de gegeven mnemonic, de gewenste afwijking kunnen aanbrengen.

4) AMDASM 29 staat slechts een breedte voor het microwoord van maximaal 128 bit toe. Daar de breedte van het microwoord 196 bit bedraagt, dient de assemblage in minimaal 2 slagen te geschieden.

Indien we nu deze regels toepassen bij een herindeling van het microwoord naar velden die functie georienteerd zijn, dan kunnen we de indeling gegeven door tabellen in fig. 3.2 en 3.3 verkrijgen. Niet alle instructies van de Am2900-bouwstenen zijn in de assembler geimplementeerd daar sommigen geen zinvol (in FP-mode) effekt hadden.

Het feit dat we hier te maken hebben met 2 tabellen vloeit voort uit regel 4); de scheidslijn is zodanig gekozen dat er een minimale koppeling tussen beide helften bestaat.

De nu volgende bespreking van de tabellen in fig. 3.2 en 3.3. is alleen van belang voor lezers die deze assembler willen gaan gebruiken. Voor het volgen van het verdere verhaal is de rest van deze paragraaf niet van essentieel belang.

functie	indeling	afstand	afwijking	samenhang
ALU configuration	4-BYTES/,3-BYTES/,2-BYTES/	-,109	-	A
Exponent ALU function	$I_{4A}...I_{1A}$,IEN_A/	3,104	1)	B
Exponent ALU source and destination control	OE_{YA}/,$I_{8A}...I_{5A}$	8,99	2)	B
Exponent ALU external source and destination control	DBE_A/,PDE_A/,$D_{31}...D_{24}$	13,89	3)	B
Mantissa ALU function	$I_{4BC}...I_{1BC}$,IEN_B/,IEN_C/	23,83	4)	C
Mantissa ALU source and destination control	I_{5C1},I_{5C2},OE_{YB}/,OE_{YC1}/, OE_{YC2}/,$I_{8BC}...I_{6BC}$,I_{5B}	29,74	5)	C
Mantissa ALU external source and destination control	DBE_B/,PDE_B/,$D_{23}...D_0$	38,48	6)	C
Floating Point ALU external sources and destination control	S_{DR},$DCRPE_{DR}$/,OE_{DR}/,LIRE/	64,44	7)	BCD
Pipeline Register Address	$A_5...A_0$,$B_5...B_0$,$C_5...C_0$	68,26	-	BC
Register Address control for Exponent ALU	S_{A1},S_{A0},I_{0A},S_{B1},S_{B0}, S_{C1},S_{C0}	86,19	8)	B
Register Address control for Mantissa ALU	S_{A1},S_{A0},I_{0A},S_{B1},S_{B0}, S_{C1},S_{C0}	93,12	8)	C
Swap and Copy control	(T_1,OE_1/)...(T_6,OE_6/)	100,-	-	E

figuur 3.2

functie	indeling	afstand	afwijking	samenhang
Exponent ALU shift control	$I_{10A}...I_{6A}, SE_A/$	-,78	9)	F
Exponent ALU status control	$OE_{CTA}/, I_{3A}...I_{1A}, I_{5A}, I_{4A}, I_{0A}, CE_{UA}/, CE_{MA}/$	6,69	10)	GJM
Exponent ALU carry-in control	I_{12A}, I_{11A}	15,67	-	H
Mantissa ALU shift control	$I_{10BC}, I_{9B}...I_{6B}, I_{9C}...I_{6C}, SE_B/, SE_C/$	17,56	11)	I
Mantissa ALU status control	$OE_{CTB}/, I_{3BC}...I_{1BC}, I_{5BC}, I_{4BC}, I_{0BC}, CE_{UB}/, CE_{MB}/, CE_{UC}/, CE_{MC}/, OE_{CTC}/$	28,44	12)	GJM
Mantissa ALU carry-in control	I_{12BC}, I_{11BC}	40,42	-	K
Floating Point ALU status transport control	$COLE/, S_{CO}, COOE/, T_{YBSEP}, OE_{YBSEP}/$	42,37	-	L
Condition Code multiplexer control	$A, B, C, OECC_1/, OECC_2/, POL$	47,31	13)	GJM
Sequencer control	$I_3...I_0, CCEN/, RLD/, BRCNT_{11}...BRCNT_0$	53,13	-	N
Prediction and Timing control	$PR, CCVAL_3...CCVAL_0, MAX_3...MAX_0, MIN_3...MIN_0$	71,-	-	O

figuur 3.3

In deze tabellen hebben de kolommen de volgende kopnamen:

1) "functie".
   Deze kolom bevat een naam die de functie aangeeft van dit veld.
2) "indeling".
   In deze kolom wordt de samenstelling van het veld in de afzonderlijke bits weergegeven.
3) "afstand".
   Om de positie van dit veld aan te geven (in Definition Phase) kunnen we gebruik maken van een getallenpaar. Van deze 2 getallen geeft het eerste de afstand aan van het begin van het microwoord tot het begin van het betreffende veld. Het tweede getal geeft de afstand van het einde van dit veld tot het einde van het microwoord weer.
4) "afwijkingen".
   De reden waarom is afgeweken van de triviale indeling wordt uit de doeken gedaan in de hierna volgende tekst, aangegeven door het getal in deze kolom.
5) "samenhang".
   Deze kolom geeft een aanduiding omtrent de samenhang van de verschillende velden. Het format dat de functie definieert, vult -geheel of gedeeltelijk- ook formats geassocieerd met de letters op dezelfde regel in deze kolom. Komen er meerdere letters op een regel voor dan wordt ter onderscheiding de letter, waarmee het format op deze regel is geassocieerd, onderstreept. Deze samenhang is gecreeerd, om op basis van functionele groepen, het aantal mnemonics tot een minimum te beperken. Ook deze kolom wordt uitvoerig behandeld.

Bespreking tabel in figuur 3.2 "afwijkingen"

ad 1) De ALU functie wordt gestuurd door $I_5 ... I_0$, zie (AM29) blz. 2-32 Table 2. De I_0 is, om redenen die in punt 8) vermeld worden, verplaatst naar de Register Address control velden.

ad 2) $I_8 ... I_5$ (en $OE_Y/$) besturen de ALU Destination of de Special Functions, zie (AM29) blz. 2-34 Table 3.

ad 3) De bits $DBE_A/$ en $PDE_A/$ vormen samen met de eerste 8 bit van de Immediate Data een veld dat de externe bron of bestemming van de Exponent ALU bestuurt.

ad 4) Ook hier is om dezelfde reden als in punt 1) de I_0 verplaatst.

ad 5) De gemeenschappelijke bits (Exponent en Mantissa ALU function) zijn rechts in het veld geplaatst, zie regel 2).

ad 6) De zelfde volgorde als in punt 3); nu echter met de 24 resterende data bits.

ad 7) LIRE/ is een enable-signaal dat het laden van het Indirection Register vanuit ALU slices 7,6 toestaat. Hoewel het physisch gezien een besturingssignaal voor de Exponent ALU is, verdient het toch de voorkeur om dit signaal in een ander veld onder te brengen. Logisch gezien bestaat er nl. geen koppeling tussen Exponent ALU en het Indirection Register, wel tussen de Floating Point ALU als geheel en dit register.

ad 8) Het I_0 instructie bit van de Am2903 heeft meerdere functies. Op de eerste plaats is het een ALU functie bit, met name die functies waarvan geen S-operand aangeboden hoeft te worden (uitgezonderd Special Functions, zie (AM29) blz. 2-32 Table 2).
Dan heeft I_0 ook nog invloed op de keuze van ALU source operands (zie (AM29) blz. 2-32 Table 1). I_0 stuurt de S-operand multiplexer en schakelt tussen Q Register en B operand.

De toewijzing van don't cares aan I_0 als ALU functie bit is zodanig dat geen "overlay conflict" hoeft op te treden met I_0 als S-operand selectie bit. Een dergelijk conflict treedt op indien twee mnemonics het zelfde veld proberen te vullen. Door nu I_0 links in het B register selectie veld op te nemen kan het in de meeste gevallen door een S-operand selectie mnemonic gevuld worden, regel 2): "left truncation". Bij die ALU functies waarin I_0 geen don't care is wordt dit bit gevuld door een ALU functie mnemonic, de resterende 2 bits van dit veld hoeven niet toegewezen te worden.
In het geval van een Special Function sluit het I_0 functie bit en het S-operand keuze veld elkaar niet meer uit. Met het invoeren van een nieuwe mnemonic, waarin het I_0 bit van het S-operand keuze veld is afgesplitst, wordt de conflict situatie omzeild.

Bespreking tabel in figuur 3.2 "samenhang" (en presentatie mnemonics)

Opmerkingen:

- Zie, voor de exacte definitie van de mnemonics, FPALU1.DEF bijlage I 1/8.
- De constructie "{ ... }" geeft aan dat er een keuze gemaakt moet worden uit de kolom die door beide haken wordt omgeven. Onderstreepte elementen geven default waarden weer. De tekens "△" en "," fungeren als delimiters tussen format name en argumenten enerzijds en argumenten onderling anderzijds.
  Er kunnen verder ook numerieke gegevens via de argumenten doorgespeeld worden:
  1) ‹byte›, 8 bit constante
  2) ‹word›, 16 bit constante
- De volgorde van behandeling is van boven naar beneden; functies die sterk samenhangen worden

gezamelijk besproken.

- De registers R_{16} ...R_{63} zijn niet voorzien van mnemonics, daar deze nog niet gebruikt worden.

ad A) velden: 1) ALU configuration

samenhang: Geen samenhang met andere velden.

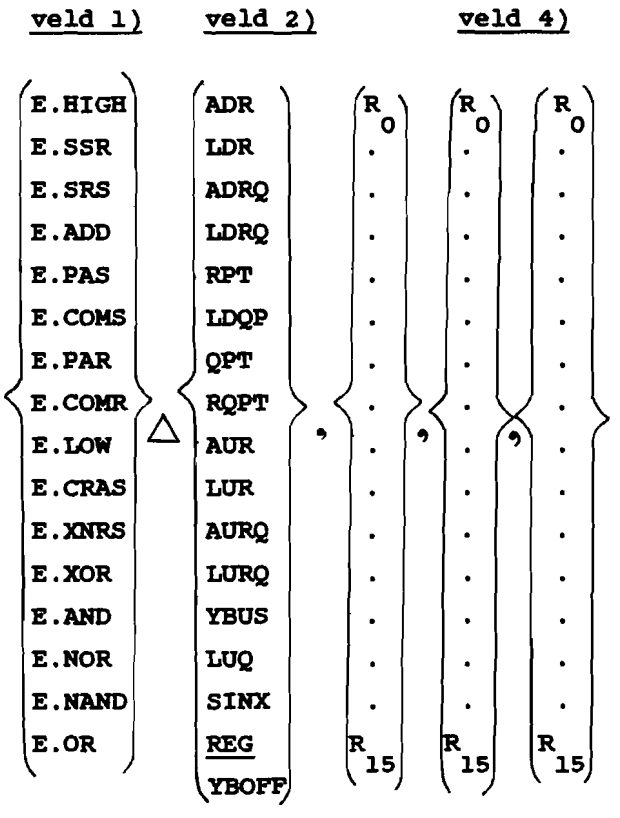
mnemonics: { DINT, FLOAT, SINT }

ad B) velden: 1) Exponent ALU function
2) Exponent ALU source and destination control
3) Exponent ALU external source and destination control
4) Pipeline Register Address

samenhang: Veld 1) is het hoofdveld (format name), velden 2), 3) en 4) fungeren als argument-velden. Hoe de velden moeten worden gespecificeerd is hieronder weergegeven.

mnemonics: veld 1) veld 2) veld 4)

{ E.SPP, E.SPFNW } Δ { UMUL, TCM, INC, SMTC, SLN, DLN, TCFD, TCD, TCDC } , { R_0 ... R_{15} } , { R_0 ... R_{15} } , { R_0 ... R_{15} }



dummy veld 3)

E.READ △ ⟨PL⟩, ⟨<byte>⟩

dummy veld 3)

E.WRITE △ ⟨DB⟩

velden 1),2) en 3)

E.ALUOFF

ad C) velden: 1) Mantissa ALU function
2) Mantissa ALU source and destination control
3) Mantissa ALU external source and destination control
4) Pipeline Register Address

samenhang: Veld 1) is wederom het hoofdveld (format name), velden 2), 3) en 4) fungeren als argumentvelden. Vanwege dezelfde indeling kunnen alle mnemonics met enkele wijzigingen worden overgenomen.

mnemonics: veld 1) veld 2) veld 4)

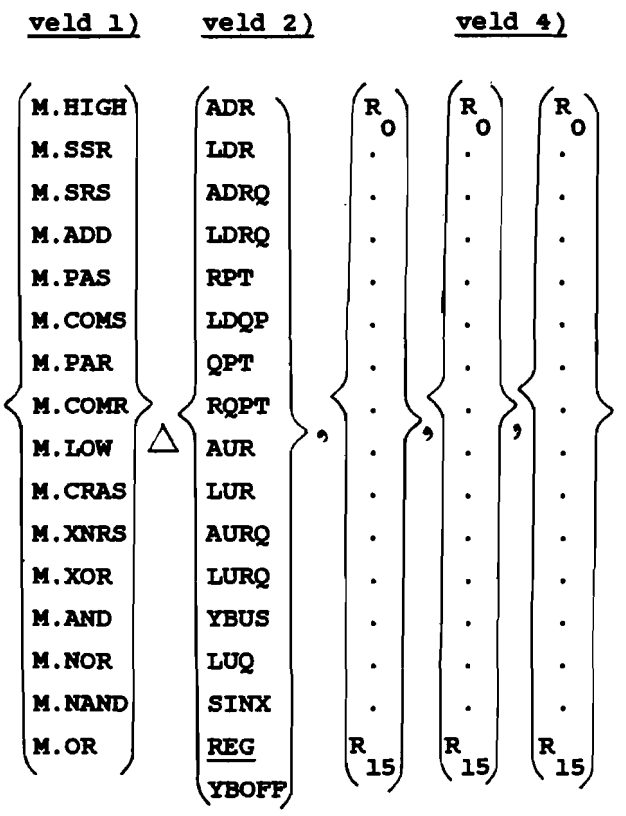
$$\{\text{M.SPF}, \text{M.SPFNW}\} \triangle \{\text{UMUL}, \text{TCM}, \text{INC}, \text{SMTC}, \text{SLN}, \text{DLN}, \text{TCFD}, \text{TCD}, \text{TCDC}\}, \{R_0, R_1, \ldots, R_{15}\}, \{R_0, R_1, \ldots, R_{15}\}, \{R_0, R_1, \ldots, R_{15}\}$$

dummy veld 3)

$$\text{M.READ} \triangle \{\underline{\text{PL}}\}, \{\langle\text{word}\rangle\}, \{\langle\text{byte}\rangle\}$$

dummy veld 3)

$$\text{M.WRITE} \triangle \{\underline{\text{DB}}\}$$



velden 1),2) en 3)

M.ALUOFF

ad BCD) velden: 1) Floating Point ALU external source and destination control
2) Exponent ALU external source and destination control
3) Mantissa ALU external source and destination control

samenhang: Een gedeelte van 2) en 3) zijn hoofdvelden. Veld 1) fungeert als argument. De gedeelten die ingevuld worden in velden 2) en 3)

hebben betrekking op het fixeren van de externe bronnen voor lezen en schrijven op Y-bus resp. DB-bus.

mnemonics: veld 2),3)

FP.READ

veld 2),3) veld 1)

FP.WRITE $\triangle \left\{\begin{matrix}\text{XDOFF}\\ \text{DB}\end{matrix}\right\}, \left\{\begin{matrix}\text{XDOFF}\\ \text{DB}\end{matrix}\right\}, \left\{\begin{matrix}\text{YB}\\ \underline{\text{DB}}\end{matrix}\right\}, \left\{\underline{\text{IR}}\right\}$

veld 2),3)

FP.XDOFF

ad BC) Reeds behandeld in ad B),C).

ad B) velden: 1) Register Address control for Exponent ALU

samenhang: Veld 1) is het argumentveld. Het hoofdveld vult geen bits; het verricht enkel een label functie.

mnemonics: dummy veld 1)

E.RAC $\triangle \left\{\begin{matrix}\text{QR}\\ \text{PL}\\ \text{AR}\\ \text{IR}\end{matrix}\right\}, \left\{\begin{matrix}\text{QR}\\ \text{PL}\\ \text{AR}\\ \text{IR}\end{matrix}\right\}, \left\{\begin{matrix}\text{QR}\\ \text{PL}\\ \text{AR}\\ \text{IR}\end{matrix}\right\}$

dummy veld 1)

$$E.SPFRAC \triangle \left\{\begin{matrix}PL\\AR\\IR\end{matrix}\right\} , \left\{\begin{matrix}PL\\AR\\IR\end{matrix}\right\} , \left\{\begin{matrix}PL\\AR\\IR\end{matrix}\right\}$$

ad C) velden: 1) Register Address control for Mantissa ALU

samenhang: Veld 1) is het argumentveld. Het hoofdveld vult geen bits; het verricht enkel een label functie.

mnemonics: dummy veld 1)

$$M.RAC \triangle \left\{\begin{matrix}QR\\PL\\AR\\IR\end{matrix}\right\} , \left\{\begin{matrix}QR\\PL\\AR\\IR\end{matrix}\right\} , \left\{\begin{matrix}QR\\PL\\AR\\IR\end{matrix}\right\}$$

dummy veld 1)

$$M.SPFRAC \triangle \left\{\begin{matrix}PL\\AR\\IR\end{matrix}\right\} , \left\{\begin{matrix}PL\\AR\\IR\end{matrix}\right\} , \left\{\begin{matrix}PL\\AR\\IR\end{matrix}\right\}$$

ad E) velden: 1) Swap and Copy control

samenhang: Veld 1) is het hoofdveld. Er zijn geen argumentvelden, daar in dit laatste veld verboden combinaties kunnen worden geprogrammeerd. Om een bus-conflict te

vermijden zijn alle toegestane combinaties van een mnemonic (zonder argumenten) voorzien.

mnemonics: veld 1)

{ COPY.B1.B3
COPY.B3.B1
SWAP.B3.B1
COPY.B0.B1
COPY.B1.B0
SWAP.B1.B0
COPY.B0.B2
COPY.B2.B0
SWAP.B2.B0
SWAP.OFF
COPY.OFF }

Om de hoeveelheid tekst, benodigd voor het programmeren van een regel micro-code, te beperken zijn vaak voorkomende bitpatronen gecombineerd tot een mnemonic. Deze is als volgt gedefinieerd:

velden:
1) ALU configuration
2) Exponent ALU external source and destination control
3) Mantissa ALU external source and destination control
4) Floating Point ALU external source and destination control
5) Swap and Copy control

samenhang: Velden 1),4) en 5) zijn hoofdvelden en worden door format name FP.MODE0 gevuld. De gedeelten van veld 2) en 3) die gevuld

worden hebben betrekking op het afzetten van de externe bronnen.

mnemonics: velden 1),4,5) velden 2),3)

FP.MODEO △ {E} , {M}

Bespreking tabel in figuur 3.3 "afwijkingen".

ad 9) De Exponent ALU shift control wordt gestuurd door bit $I_{10}...I_{6}$ en bit SE_A/, zie (AM29) blz. 2-90 Table 7.

ad 10) De bits $I_5...I_0$ besturen zowel Status Register als Condition Code Output control. Vergelijking van de Tables 1,2 en 4 blz. 2-86/88 (AM29) laat ons zien dat er in de Condition Code Output instruction code de meeste structuur zit. We laten ons dan ook door deze structuur leiden om d.m.v. een permutatie op $I_5...I_0$ gunstige mnemonics te bepalen. Indien we deze structuur nader bekijken dan kunnen we 3 onderdelen onderscheiden:

1) Output functie.
   $I_3...I_1$ bepaalt de aard van de functie die gegenereerd wordt. Dit kan varieren van het simpelweg doorschakelen van een van de 4 status bits (afkomstig van een Am2903) tot meer ingewikkelde combinaties (b.v. de 2's compl. kleiner dan: "<"). In het totaal kan uit 8 functies worden gekozen.
2) Conditie code bron.
   De bits I_5 en I_4 selecteren de bron waar uit de conditie code wordt samengesteld. Deze bron kan ofwel een van beide Status Registers (MSR,uSR) of wel de Direct Input (I-lijnen) zijn.
3) Inversie.
   De door de vorige 2 onderdelen 1) en 2) geselecteerde functie kan nog worden geinverteerd m.b.v. het I_0 bit.

Voor elk van de 3 onderdelen (velden) dient een apart symbool of mnemonic te worden gekozen (zie bespreking "samenhang"). De volgorde waarin deze 3 velden gevuld dienen te worden hebben we bepaald op 2),1) en dan 3).

ad 11) Zoals bij de Exponent ALU het geval was wordt ook hier het schuiven en roteren ondermeer door I_{10B} .. I_{6B}, SE / B bepaald. Besturingsbits I_{9C}...I_{6C} hebben geen aparte betekenis (ALU in FP-mode). Wel dient er, vanwege de plaatsing van de Am2904-C in de Am2903-array, steeds een vaste opdracht te worden aangeboden. Dit laatste heeft het effekt dat deze bouwsteen in de transparant mode wordt gehouden (zie par. 2.3.9).

ad 12) De herindeling van dit veld is om dezelfde reden als vermeld in punt 10) uitgevoerd.

ad 13) We zijn van de triviale indeling afgeweken daar we het inversie bit, gebruikt in de conditie code selectie, in de regel rechts in het veld plaatsen (zie ad 10)).

<u>Bespreking tabel in figuur 3.3 "samenhang" (en presentatie mnemonics).</u>

Opmerkingen:

- Zie, voor de exacte definitie van de mnemonics, FPALU2.DEF bijlage I 9/18.
- De constructies "$\{\cdots\}$" en "$\triangle$" hebben hier dezelfde functie als bij de bespreking van samenhang in figuur 3.2. Verder kunnen -behalve ‹byte› en ‹word›- ook de volgende numerieke gegevens via de argumenten worden doorgespeeld:
  1) ‹4 bit›, 4 bit constante
  2) ‹12 bit›, 12 bit constante
  3) ‹label›, een 12 bit address (absoluut of symbolisch)

ad F) velden: 1) Exponent ALU shift control

samenhang: Veld 1) is het hoofdveld (format name). Er zijn geen argumentvelden.

mnemonics: veld 1)

- E.SDL
- E.SDH
- E.SUL
- E.SUH
- E.SDDL
- E.SDDH
- E.SDUL
- E.SDUH
- E.RSD
- E.RSU
- E.SSXO
- E.RDD
- E.RDU
- E.SDMS
- E.SMS
- E.SDDCL
- E.SDUCL

ad GJM) velden: 1) Exponent ALU status control
2) Mantissa ALU status control
3) Condition Code multiplexer control

samenhang: Veld 1) treedt in bijna alle gevallen als argumentveld op. De velden 1),2) en 3) treden in geval van conditie code selectie (format names E.CT of E.REGCT) ook op als hoofdveld.

mnemonics: dummy veld 1)

E.REG △ { PAS.IL / SET / RESET / LOAD / LOAD.CI } , { M / U / UM }

dummy veld 1)

E.REG △ { INVERT / MOV.Y / MOV.U / LOAD.SO } , M

dummy veld 1)

E.REG △ { MOV.M / LOAD.OR } , U

dummy veld 1)

E.REG △ SWAP , UM

dummy veld 1)

E.REG △ { PAS.I / PAS.M / PAS.U }

veld 1)

E.REGOFF

velden 1),2) en 3) veld 1)

$$E.CT \triangle \left\{ \begin{matrix} OVR \\ C \\ N \\ Z \\ LEQ \\ LSS \\ C.OR.Z \\ NC.OR.Z \end{matrix} \right\} , \left\langle \begin{matrix} I \\ U \\ M \end{matrix} \right\rangle , \left\langle \begin{matrix} \underline{T} \\ F \end{matrix} \right\rangle$$

velden 1),2) en 3) veld 1)

$$E.REGCT \triangle \left\{ \begin{matrix} OVR \\ C \\ N \\ Z \\ LEQ \\ LSS \\ C.OR.Z \\ NC.OR.Z \end{matrix} \right\} , \left\langle \begin{matrix} I \\ U \\ M \end{matrix} \right\rangle , \left\langle \begin{matrix} \underline{T} \\ F \end{matrix} \right\rangle , \left\langle \begin{matrix} U \\ M \end{matrix} \right\rangle$$

ad H) velden: 1) Exponent ALU carry-in control

samenhang: Veld 1) is een argumentveld.

mnemonic: dummy veld 1)

$$E.CIN \triangle \left\langle \begin{matrix} ONE \\ ZERO \\ CX \end{matrix} \right\rangle$$

ad I) velden: 1) Exponent ALU shift control

samenhang: Veld 1) is het hoofdveld (format name). Er zijn geen argumentvelden.

mnemonics: veld 1)

- M.SDL
- M.SDH
- M.SUL
- M.SUH
- M.SDDL
- M.SDDH
- M.SDUL
- M.SDUH
- M.RSD
- M.RSU
- M.SSXO
- M.RDD
- M.RDU
- M.SDMS
- M.SMS
- M.SDDCL
- M.SDUCL

ad GJM) velden: 1) Exponent ALU status control
2) Mantissa ALU status control
3) Condition Code multiplexer control

samenhang: Veld 1) treedt in bijna alle gevallen als argumentveld op. De velden 1),2) en 3) treden in geval van conditie code selectie (format names M.CT of M.REGCT) ook op als hoofdveld.

mnemonics: dummy veld 1)

M.REG △ { PAS.IL, SET, RESET, LOAD, LOAD.CI } , { M, U, UM }

dummy veld 1)

M.REG △ { INVERT, MOV.Y, MOV.U, LOAD.SO } , M

dummy veld 1)

M.REG △ { MOV.M, LOAD.OR } , U

dummy veld 1)

M.REG △ SWAP , UM

dummy veld 1)

M.REG △ { PAS.I, PAS.M, PAS.U }

veld 1)

M.REGOFF

velden 1),2) en 3) veld 1)

$$\text{M.CT} \triangle \left\{\begin{array}{l}\text{OVR}\\ \text{C}\\ \text{N}\\ \text{Z}\\ \text{LEQ}\\ \text{LSS}\\ \text{C.OR.Z}\\ \text{NC.OR.Z}\end{array}\right\} , \left\{\begin{array}{l}\text{I}\\ \text{U}\\ \text{M}\end{array}\right\} , \left\langle\begin{array}{l}\underline{\text{T}}\\ \text{F}\end{array}\right\rangle$$

velden 1),2) en 3) veld 1)

$$\text{M.REGCT} \triangle \left\{\begin{array}{l}\text{OVR}\\ \text{C}\\ \text{N}\\ \text{Z}\\ \text{LEQ}\\ \text{LSS}\\ \text{C.OR.Z}\\ \text{NC.OR.Z}\end{array}\right\} , \left\{\begin{array}{l}\text{I}\\ \text{U}\\ \text{M}\end{array}\right\} , \left\langle\begin{array}{l}\underline{\text{T}}\\ \text{F}\end{array}\right\rangle , \left\langle\begin{array}{l}\text{U}\\ \text{M}\end{array}\right\rangle$$

ad K) velden: 1) Mantissa ALU carry-in control

samenhang: Veld 1) is een argumentveld.

mnemonic: dummy veld 1)

$$\text{M.CIN} \triangle \left\langle\begin{array}{l}\text{ONE}\\ \text{ZERO}\\ \text{CX}\end{array}\right\rangle$$

ad L) velden: 1) Floating Point ALU status transport control

samenhang: Veld 1) is het hoofdveld (format name)

mnemonic: veld 1)

{ ST.SAVE
ST.RESTR
CO.LOAD
ST.OFF }

ad GJM) velden: 1) Exponent ALU status control
2) Mantissa ALU status control
3) Condition Code multiplexer control

samenhang: Veld 1) treedt in bijna alle gevallen als argumentveld op. De velden 1),2) en 3) treden in geval van conditie code selectie ook op als hoofdveld.

mnemonics: dummy veld 3)

FP.CT △ { CC_1 CC_2 CC_3 CC_4 CC_5 CO_0 CO_1 CO_2 CO_4 CO_6 } , { T F }

veld 1),2,3)

FP.CTOFF

ad N) velden: 1) Sequencer control

samenhang: Veld 1) fungeert gedeeltelijk als hoofd- en als argumentveld.

mnemonics:

veld 1)	veld 1)
JZ, CJS, JMS, JMAP, CJP, JMP, PUSH, JSRP, CJV, JRP, RFCT, RPCT, CRTN, RTN, CJPP, LDCT, LOOP, CONT, TWB	△ UNCOND, LDCTPAR

dummy	veld 1)
GOTO, COUNT	△ <label>, <12 bit>

ad 0) velden: 1) Prediction and Timing control

samenhang: Veld 1) is het hoofdveld.

mnemonics: dummy veld 1)

PREDICT △ {T / F}

dummy veld 1)

TIMERS △ <4 bit>,<4 bit>,<4 bit>

Om ook hier de hoeveelheid tekst, benodigd voor het programmeren van een regel micro-code, te beperken zijn vaak voorkomende bitpatronen gecombineerd tot een mnemonic. De velden in kwestie zijn:

velden:
1) Exponent ALU shift control
2) Mantissa ALU shift control
3) Exponent ALU status control
4) Mantissa ALU status control
5) Floating Point ALU status transport control

samenhang: Alle velden zijn hoofdvelden.

mnemonics: velden 1),2,5)

FP.MODE0

velden 1),2)

FP.SHIFTOFF

velden 3),4)

FP.REGOFF

3.3 Implementatie Floating Point Routines op Register Machine

3.3.1 Inleiding

Naast de integer- en BCD-getalrepresentatie bestaat nog een derde manier om getallen te coderen: De Floating Point weergave. In tegenstelling tot de eerste twee beeldt deze laatste reele getallen af en wel met een relatief groot bereik. Dit hoofdstuk is verder onderverdeeld in de volgende paragrafen:

3.3.2 Afspraken
3.3.3 Vermenigvuldigen
3.3.4 Delen
3.3.5 Optellen/Aftrekken

Al deze routines zijn geimplementeerd op een register machine d.w.z. een emulatie van een register machine op de microprogrammeerbare machine. Met behulp van deze machine is het mogelijk enkelvoudige FP-bewerkingen (b.v. vermenigvuldigen) te verrichten. Dit vanwege het feit dat de adressering van operanden wordt verzorgd door het Address Register. Dit register is nl. niet vanuit de ALU te laden. De operanden zelf

bevinden zich in de Register File. Ook het resultaat wordt in een register geplaatst dat aangewezen wordt door het Address Register.

3.3.2 Afspraken

De Floating Point weergave van een getal bestaat uit drie onderdelen:

1) Mantisse (mant.)
2) Exponent (exp.)
3) Basis

Deze worden op de volgende wijze gecombineerd:

$$\text{FP-getal} = \text{mant.} * \text{Basis}^{\text{exp.}}$$

en kunnen in principe elk getal in $\mathbb{R}$ kunnen weergeven. Meestal worden voor deze onderdelen de volgende afspraken gemaakt:

Mantisse	-	binaire fractie	(expliciet).
Exponent	-	integer	(expliciet).
Basis	-	macht van 2, bv. 2 of 16	(impliciet).

Door beperkingen zoals registerbreedte etc. kunnen niet alle reele getallen worden weergegeven. Beperking op de breedte van de mantisse introduceert namelijk een relatieve onnauwkeurigheid in het FP-getal (orde grootte: 2^{-N}).
Beperking op de exponentbreedte (stel M bits) betekent een begrenzing van het bereik der weergegeven getallen. De exponent kan dan waarden aannemen tussen -2^{M-1} en $2^{M-1}-1$; het bereik hangt dan nog af van de keuze voor de basis.
Er zijn in hoofdzaak 2 alternatieven voor codering van FP-getallen, zie (FLOR):

1) Decimal FP
2) Binary FP

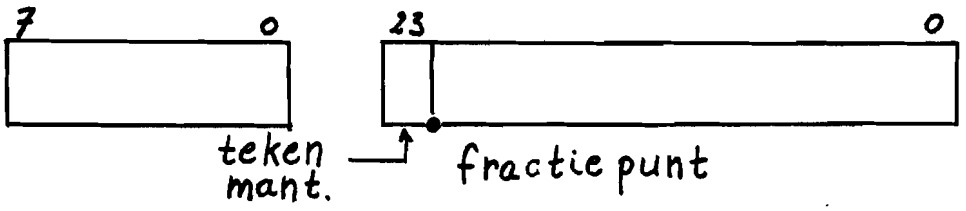
Decimal FP wordt door de Am2903 niet ondersteund (geen BCD arithmetic) en verdient derhalve niet onze aandacht.

In binary FP zijn de 2 hoofdvarianten mogelijk:
1) Sign Magnitude mantisse (SM)
2) Two's Complement mantisse (2's Compl.),
beiden met exponent basis 2 of 16.

ad 1) De mantisse wordt gecodeerd als een absoluut binair getal vergezeld door een tekenbit dat al naar gelang in het mantisse- of in het exponent-register geplaatst wordt. Er zijn dus 2 hoofdvarianten:
- Teken van het FP-getal in bit 23 van mantisse (fig. 3.4). De exponent kan dan waarden aannemen tussen -128 en +127.
- Teken van het FP-getal in bit 7 van de exponent (fig. 3.5). De exponent kan dan waarden aannemen tussen -64 en +63. Aangezien het bereik van het getal bij basis 2 niet zo groot is, kiest men deze wel eens 16 (bereik: $10^{\pm 18}$ tegen $10^{\pm 75}$). Verder wordt de exponent in een excess-64 code (met offset 64) opgeslagen. Dit levert voordelen op bij het vergelijken van twee 7 bit exponenten in een 8 bit "omgeving".

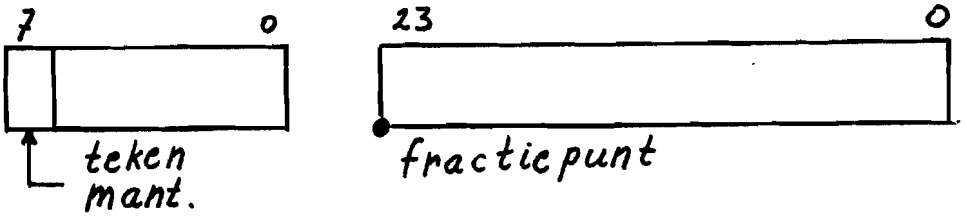
Deze representatie vorm wordt veel gebruikt in systemen die geen 2's compl. multiply/divide instructies bezitten, zie lit. (COON). Alle basisbewerkingen behalve Unsigned Divide worden door de Am2903 ondersteund.

ad 2) De mantisse wordt nu voorgesteld door een signed binair getal, zie fig. 3.6. Deze notatievorm biedt de volgende voordelen:
1) Gemakkelijke optel- en aftrek-bewerkingen.



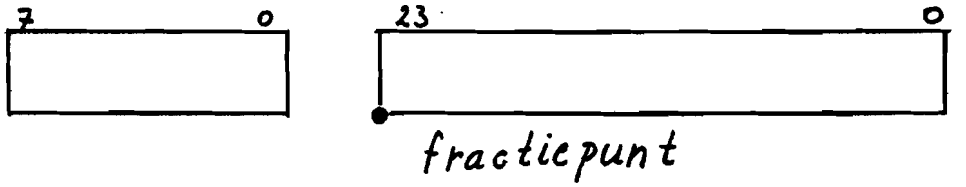
8 bit exponent 23 bit mantisse (SM)

fig. 3.4



7 bit exponent in excess-64 24 bit mantisse (SM)

fig. 3.5



8 bit exponent 24 bit mantisse (2's comp.)

fig. 3.6

2) Geen excess-code nodig voor 7 bit exponent.

Alle basis bewerkingen (signed!) +,-,/,* kunnen m.b.v. de Am2903 worden verricht.

Wegens de genoemde voordelen hebben we voor de laatste notatievorm gekozen, temeer door het feit dat Unsigned Divide niet tot de mogelijkheden van de Am2903 behoort.

Wat betreft de keuze van de basis voor de exponent komt natuurlijk alleen een 2 macht in aanmerking. Kandidaten zijn dan 2 of 16; tussenliggende waarden worden door de Am2903 niet ondersteund. In dit ontwerp is voor basis 2 gekozen daar dit resulteert in de meest eenvoudige hardware/firmware combinatie.

Resumerend:

1) Mantisse in 2's complement, $1/4 \leq$ mant. $< 1/2$
   $-1/2 \leq$ mant. $< -1/4$
2) Exponent in 2's complement, $-128 \leq$ exp. ≤ 127
3) Basis is 2

Door nu deze regels te projecteren op een dubbele registerbreedte kan een Double Precision-formaat worden verkregen. Een dergelijk formaat wordt gebruikt om tussenresultaten van sommige FP-bewerkingen op te slaan.

Nu is komen vast te staan op welke wijze de onderdelen gecodeerd zullen worden, kunnen we enkele voorbeelden geven.

Voorbeelden:	Decimaal	Binair	
		Exp.	Mant.
1)	+1	02H	40 00 00H
2)	-0.375	00H	A0 00 00H
3)	-1	01H	80 00 00H

4)	0	00H	00 00 00H
5)	0	80H	00 00 00H

Deze notatievorm blijkt 2 problemen op te leveren, zie (FLOR):

1) "Dirty zero" of "0 coding"
2) "-1 coding",

Beiden zijn in het voorbeeld opgenomen, zie 4),5) en 3).

ad 1) Een "dirty zero" treedt op indien na een FP-bewerking de mantisse gelijk is aan: 00 00 00H en de bijbehorende exponent > -128. Waarom een "dirty zero" problemen kan opleveren wordt uit de doeken gedaan in paragraaf 3.3.5: Optellen/aftrekken.

ad 2) In feite is "-1 coding" het probleem dat $-(-2^{N-1})$ niet te coderen is in N bits (2's complement). Dit kan moeilijkheden opleveren bij o.a. inverteren en normaliseren.

Om aan gebruikers van dit primitieve FP-pakket mededelingen te doen over het (de) eindresulta(a)t(en) moeten we een returncode invoeren. We definieren 3 returncode bits (zie (MICK)):

1) OVR -- Overflow (exp. > +127)
2) N -- Negative (mant. < 0)
3) Z -- Zero (mant. = 0)

ad 1) Het OVR-bit wordt bijgehouden in het Machine Status Register (MSR) van de 2904-A. Indien na een FP-routine dit bit geset is dan houdt dit in dat het resultaat te groot is om in he formaat te passen. De mantisse en exponent bevatten dan geen zinnige informatie.

ad 2,3) Het N- en Z-bit worden bijgehouden in het MSR van de 2904-B. De bits worden dan geset indien het resultaat negatief respectievelijk nul is.

Opmerking:

Los van deze returncode bestaat het begrip statusbits (in MSR) van de 2904-A en -B. Om aan te duiden welke bits afkomstig zijn uit de Byte-ALU en welke uit de Mantisse-ALU, gebruiken we vaak de naam van het statusbit met als index "exp." resp. "mant.". Voorbeeld:

Overflow-bit in Byte-ALU <—> OVR_{exp}.

3.3.3 Vermenigvuldigen

Basis idee

Een van de gemakkelijkste FP-bewerkingen is wel de FP-vermenigvuldiging. We gaan uit van de getallen $(exp_1, mant_1)$ en $(exp_2, mant_2)$. Het resultaat laat zich als volgt schrijven:

$$product = (mant_1 * mant_2) . 2^{(expl + exp2)}$$

$$= mant_3 . 2^{exp3}$$

M.a.w. vermenigvuldigen van mantissen en optellen van exponenten. De mantissen en exponenten voldoen aan het formaat zoals beschreven in paragraaf 3.3.2.

Problemen, die zich bij het vermenigvuldigen voor kunnen doen, zijn (zie (FLOR) e.a.):

1) $mant_1$ en/of $mant_2 = 0$
2) $mant_1 = mant_2 = -1/2$
3) exp_3 underflow
4) exp_3 overflow

ad 1) Detectie van het product = 0 is noodzakelijk vanwege de dirty zero- en normalisatie-problemen.

ad 2) Na het product -1/2 * -1/2 is geen normalisatieslag benodigd; bij alle anderen echter wel. Het microprogramma moet hierop anticiperen en passende maatregelen nemen.

ad 3) In feite kan het exp_3 underflow probleem worden opgelost door het product gelijk aan 0 te maken.

ad 4) Exponent overflow (na $exp_3 := exp_1 + exp_2$) hoeft niet direct te betekenen dat het product niet in het voorgeschreven formaat past. De normalisatieslag na een FP-vermenigvuldiging zou nl. een "geringe" overloop van exp_3 kunnen herstellen.

Op welke wijze de FP-vermenigvuldiging kan worden geimplementeerd op de Am2903 wordt behandeld in "Implementatie".

Implementatie.

De Am2903 kent o.a. een Signed Multiply- en een Two's Complement Add-instructie (t.b.v. mantisse resp. exponent). Beide operaties werken echter op integer grootheden. De stelling dat het vermenigvuldigen van 2's complement fracties kan worden vertaald in integer bewerkingen, wordt verantwoord door het volgende.

Definities:

1) $M_{i,s}$ - 24 bit 2's compl. integer (single prec.)
2) $M_{i,d}$ - 48 bit 2's compl. integer (double prec.)
3) $m_{i,s}$ - 24 bit 2's compl. mantisse, volgens

formaat par. 3.3.2 (single prec.)

4) $m_{i,d}$ - 48 bit 2's compl. mantisse, volgens formaat par. 3.3.2 (double prec.)

5) e_i - 8 bit exponent in 2's compl.

De index i, voor onderscheiding van operanden, kan de waarden 1,2 of 3 aannemen. De volgende relaties gelden:

$$M_{i,s} = m_{i,s} \cdot 2^{24}$$

$$M_{i,d} = m_{i,d} \cdot 2^{48}$$

De Am2903 rekent in integers:

$$M_{3,d} := M_{1,s} * M_{2,s}$$

Het gewenste product is echter:

$$\begin{aligned} \text{product} &:= (m_{1,s} * m_{2,s}) \cdot 2^{(e1 + e2)} \\ &= M_{1,s} * M_{2,s} \cdot 2^{-48} \cdot 2^{(e1 + e2)} \\ &= M_{3,d} \cdot 2^{-48} \cdot 2^{(e1 + e2)} \\ &= m_{3,d} \cdot 2^{(e1 + e2)} \end{aligned}$$

En na normalisatie (stel k schuifslagen):

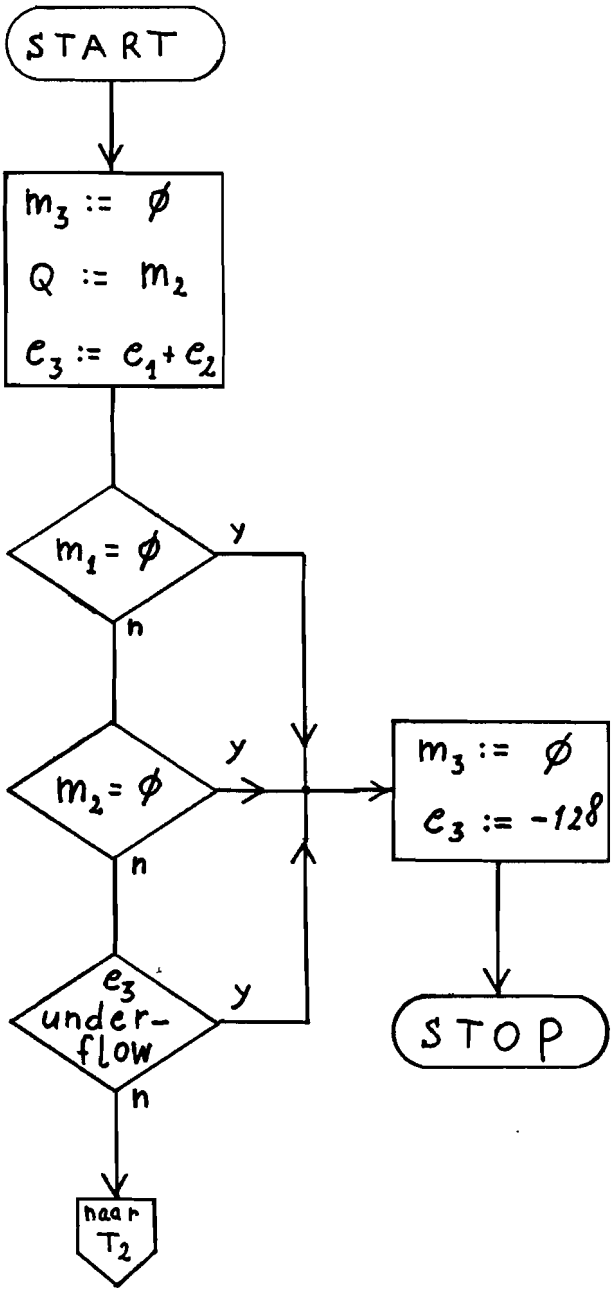
$$\begin{aligned} \text{product} &:= m_{3,d} * 2^{k} \cdot 2^{(e1 + e2 - k)} \\ &= m_{3,s} \cdot 2^{(e1 + e2 - k)}, \end{aligned}$$

waarin $m_{3,s}$ de 24 meest significante bits van het reeds genormaliseerde getal in $m_{3,d}$ bevat.

De FP-vermenigvuldiging kan opgesplitst worden in 3 taken te weten:

T1) Exponenten optellen en op de 0-uitkomst anticiperen (i.v.m. "dirty zero"-probleem).
T2) Vermenigvuldigen van mantissen.
T3) Normaliseren en corrigeren van een "geringe" overloop van de exponent.

ad T1) **stroomdiagram** **toelichting**



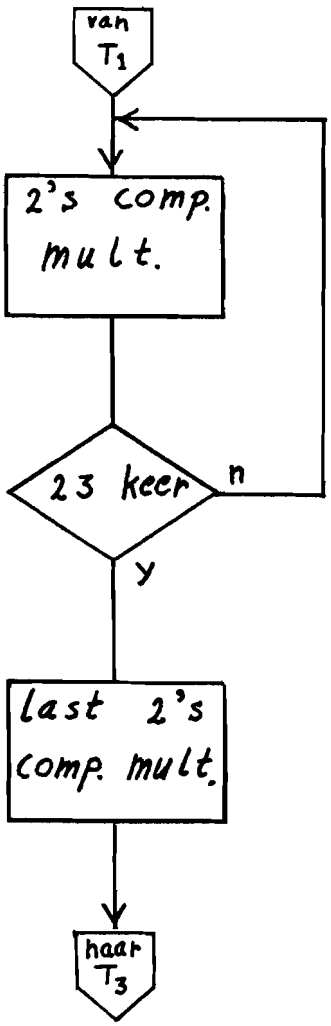
Init. partieel product.

Vul Q-register met multiplicand.

Tel exponenten op.

Test op product=0 i.v.m. het dirty zero probleem.

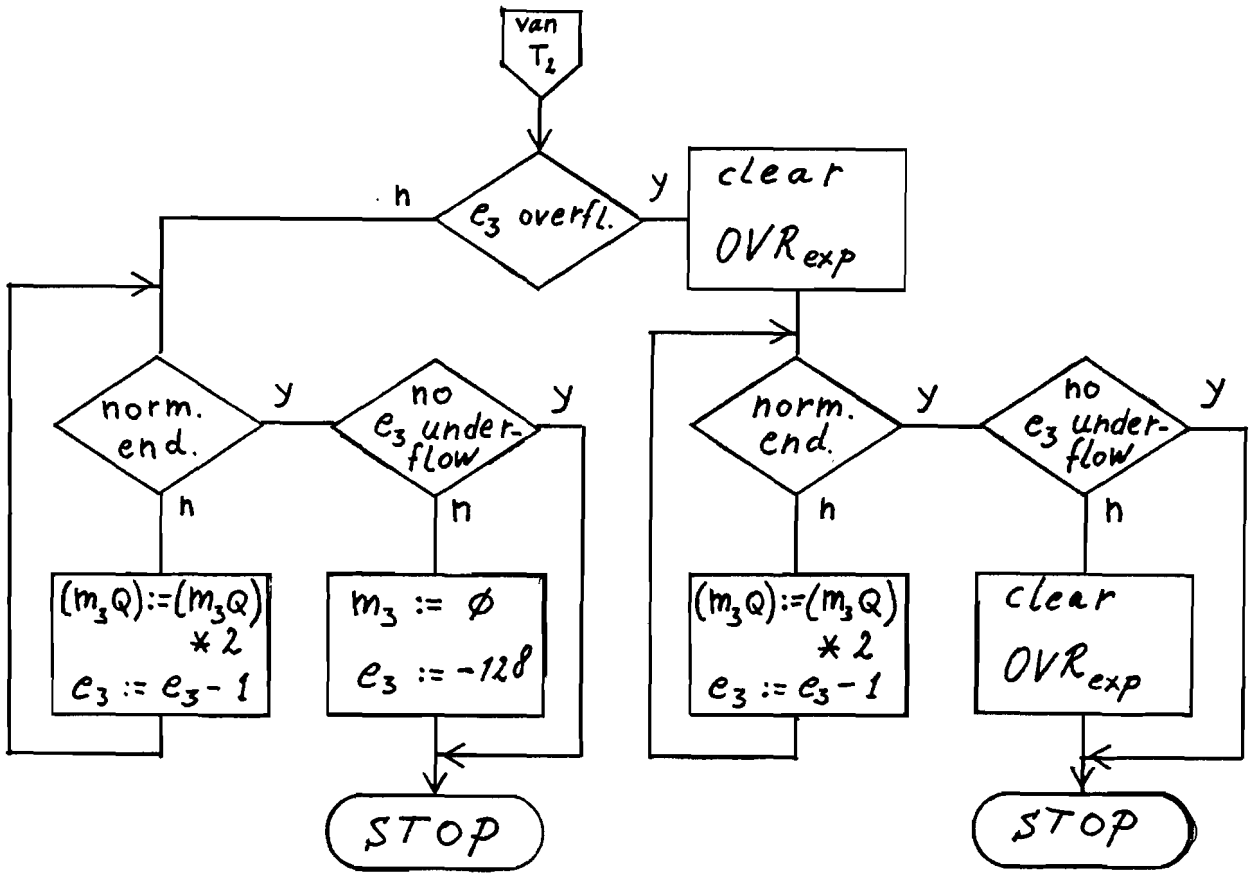
ad T2) **stroomdiagram** **toelichting**



2's Compl. Multiply in 23 slagen. Partieel product in (m_3,Q).

Corrigeer resultaat d.m.v. een Last 2's Compl. Multiply cycle. Resultaat in (m_3,Q).

ad T3) **stroomdiagram**



toelichting

Test e_3 op overflow. Is dit niet het geval dan worden schuifslagen uitgevoerd totdat de mantisse genormeerd is, of totdat de exponent kleiner wordt dan -128. In dit laatste geval wordt het product $0 \cdot 2^{-128}$.

Treedt er wel een overflow op dan tracht deze te herstellen via een normeringsslag.

Opmerkingen:

- (m_3,Q) is een concatenatie van register m_3 in de Register File en het Q Register.

- "Norm. end." (Normalization ended) is gedefinieerd als:

  $((m_3,Q)$ normalized or e_3 underflow).

- OVR is het Overflow-bit, evenals N en Z een van de statusbits van de Am2903. Er gelden de volgende definities:

  $OVR := (C_{n+4} \underline{exor} C_{n+3})_{MSS}$
  $Overflow := OVR \cdot N$
  $Underflow := OVR \cdot N/$

  Aangezien de Boolse functie "Norm. end." en de laatste 2 functies niet door de Am2904 Condition Code output gegenereerd kunnen worden zijn ze extern gerealiseerd, zie par. 2.3.10 "Conditie Code Logica".

  De 3 functies zijn:

  $CC_1 := (Y_{OVR} \cdot Y_N/)_{exp}$

$$CC_2 := (Y_{OVR} \,.\, Y_N)_{exp}$$
$$CC_3 := (Y_{OVR}/)_{mant} \,.\, (Y_{OVR} \,.\, Y_N/)_{exp}$$

Om programma-technische redenen is het wenselijk ook een variant van CC_3 mee te nemen:

$$CC_4 := (Y_{OVR}/)_{mant} \,.\, (Y_{OVR}/ + Y_N)_{exp}$$

De functie CC := true die ook benodigd is, kan worden gerealiseerd m.b.v. een stuursignaal van de Sequencer (CCEN/ van de Am2910).

Het bovenstaande stroomdiagram is omgezet in een microprogramma. Er is naar gestreefd om de gemiddelde tijdsduur van een vermenigvuldiging zo laag mogelijk te houden. Om dit te bereiken zijn de volgende regels gehanteerd:

1) Tracht de lengte van het microprogramma te verkorten door opeenvolgende microwoorden te combineren tot 1 woord.
2) Tracht ingeval van conditionele sprongopdrachten het Prediction bit (zie par. 2.3.11) een zodanige waarde te geven dat het meest frequent gebruikte pad door het programma het kortst duurt.
   Voorbeeld:
   Maak de tijdsduur van programmalussen -waarvan de loopcount van tevoren bekend is- korter. Dit kan geschieden door het prediction pad in het terugspring pad te kiezen.

Opmerkingen:

- Voor het FMUL microprogramma zie bijlage II 1/6. Daar de tijdberekeningen met de hand i.p.v. met de assembler uitgevoerd moesten worden, is vanwege tijdsgebrek alleen het FMUL programma voorzien van timerwaarden.

- De geschatte verwerkingstijd van deze routine bedraagt 11 microseconden.
- Verder is de signed integer multiply m.b.v. Am2903 bouwstenen gesimuleerd in APLZ80, bijlage II 7/10. Zie voor meer informatie (AM29) blz. 2-49.

3.2.4 Delen

Basis idee

Alhoewel FP-deling en -vermenigvuldiging verwante bewerkingen zijn kunnen bij de FP-deling enkele specifieke problemen optreden. Afgezien van deze problemen laat het resultaat zich schrijven als:

$$\text{quotient} := (\text{mant}_1 / \text{mant}_2) \, 2^{(\text{exp1} - \text{exp2})}$$

$$= \text{mant}_3 \, 2^{\text{exp3}}$$

M.a.w. delen van mantissen en aftrekken van exponenten.

Problemen, die zich bij het FP-delen voor kunnen doen, zijn (zie (FLOR) e.a.):

1) $\text{mant}_1 = 0$
2) $\text{mant}_2 = 0$
3) mant_3 ligt niet binnen het interval $[-1/2,-1/4)$ of $[1/4,1/2)$
4) exp_3 underflow
5) exp_4 overflow

ad 1) Detectie van quotient=0 is noodzakelijk i.v.m. de

"dirty zero"- en normalisatie-problemen.

ad 2) Delen door 0! De returncode OVR wordt geset.

ad 3) $Mant_3$ moet naar het gewenste formaat worden gebracht. Dit kan geschieden door schuifslagen in de mantisse en een correctie in de exponent uit te voeren.

ad 4,5) Evenals bij de FP-vermenigvuldiging kan ook hier een "geringe" overloop herstelt worden door een normalisatie-slag. Bij een FP-deling kan echter ook een "geringe" underflow hersteld worden. Het bestaan van deze correctie mogelijkheid berust echter op implementatie-gronden en wordt derhalve behandeld in "Implementatie".

Op welke wijze de FP-deling kan worden geimplementeerd op de Am2903 wordt behandeld in de volgende paragraaf.

Implementatie

De Am2903 kent een Signed Divide (t.b.v. de mantissen) en een Two's Complement Subtract (t.b.v. de exponenten). Beide operaties werken echter op integer grootheden. Bovendien moet er bij de Am2903 rekening worden gehouden met:

1) $|deler| > |deeltal|$; eventueel scaling (mantisse een positie naar rechts schuiven) van het deeltal noodzakelijk.
2) Het quotient moet nog gecorrigeerd worden (factor 2 te klein, zie ook verwijzing naar simulatie op het eind van deze paragraaf!).

Het vertalen van een FP-deling in de integer bewerkingen wordt verantwoord in het volgende. Voor definities van de gebruikte symbolen zie par. 3.3.3. Verder worden gebruikt:

se: Scratchpad register in de exponent Register File. In dit register is een getal opgeslagen

dat wordt gebruikt als scaling factor. Dit getal geeft aan hoeveel bitposities het deeltal naar rechts is verschoven om zo te voldoen aan punt 1).

$sm_{1,2}$: Hulpregisters waarin de absolute waarde van $m_{1,2}$ wordt bijgehouden.

De volgende relatie geldt voor een deling:

$$M_{3,s} := M_{1,d} / M_{2,s}$$

Inclusief correctie voor scaling (exp := exp + se) en quotient (exp := exp + 1) levert:

$$\begin{aligned} \text{quotient} &:= (m_{1,d} / m_{2,s}) \, 2^{(e1 - e2)} \\ &= M_{1,d} / M_{2,s} \, 2^{-24} \, 2^{(e1 - e2)} \\ &= M_{3,s} \, 2^{-24} \, 2^{(e1 - e2 + se + 1)} \\ &= m_{3,s} \, 2^{(e1 - e2 + se + 1)}. \end{aligned}$$

En na normalisatie (k schuifslagen):

$$\text{quotient} := m_{3,s} \, 2^{k} \, 2^{(e1 - e2 + (se + 1 - k))}$$

Bovenstaande formule toont ons tevens het mechanisme voor het corrigeren van een geringe exponent overflow/underflow. De twee effekten die hierin een rol spelen zijn:

- Het normaliseren. Dit reduceert de exponent met een bedrag k.
- Het "scalen" en de quotient-correctie. Beiden verhogen de exponent met een bedrag $se + 1$.

Krijgt een van beiden de overhand dan biedt dit de mogelijkheid om een correctieslag uit te voeren. We zullen nu de voorwaarde afleiden waarvoor een dergelijke herstelprocedure

succes heeft. Het bepalen van de exponent geschiedt in 2 stappen:

1) $e_3 := e_1 - e_2$ met returncode bit OVR_1
2) $e_3 := e_3 + se + 1$ met returncode bit OVR_2

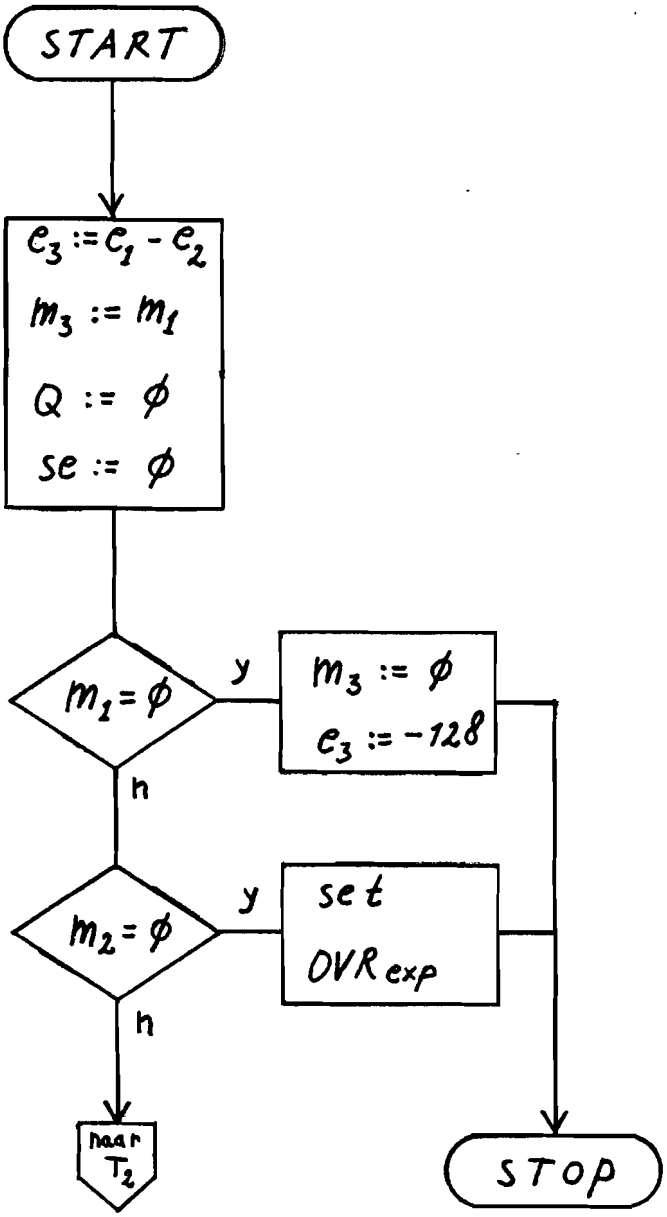
Veroorzaakt bewerking 1) een overflow en bewerking 2) een underflow dan is de procedure succesvol. Bewerking 2) kan geen overflow opleveren daar $|se| < 2$! Dit resulteert in de deelvoorwaarde: $overflow_1$ and OVR_2, daar OVR=1 wordt veroorzaakt door overflow of underflow.
Een dergelijk verhaal geldt ook voor de underflow. Combineren we beide beschouwingen dan:

"Correctie succes" := OVR_1 and OVR_2.

De FP-deling kan opgesplitst worden in 4 taken te weten:

T1) Exponenten aftrekken en testen op zowel deeltal = 0 ("dirty zero" probleem) of deler = 0 (exponent overflow).
T2) Testen of de relatie deler > deeltal geldt; e.v.t. correctie uitvoeren d.m.v. scaling.
T3) Het delen der mantissen.
T4) Normaliseren en corrigeren van een "geringe" exponent-overflow of -underflow.

ad T1) stroomdiagram



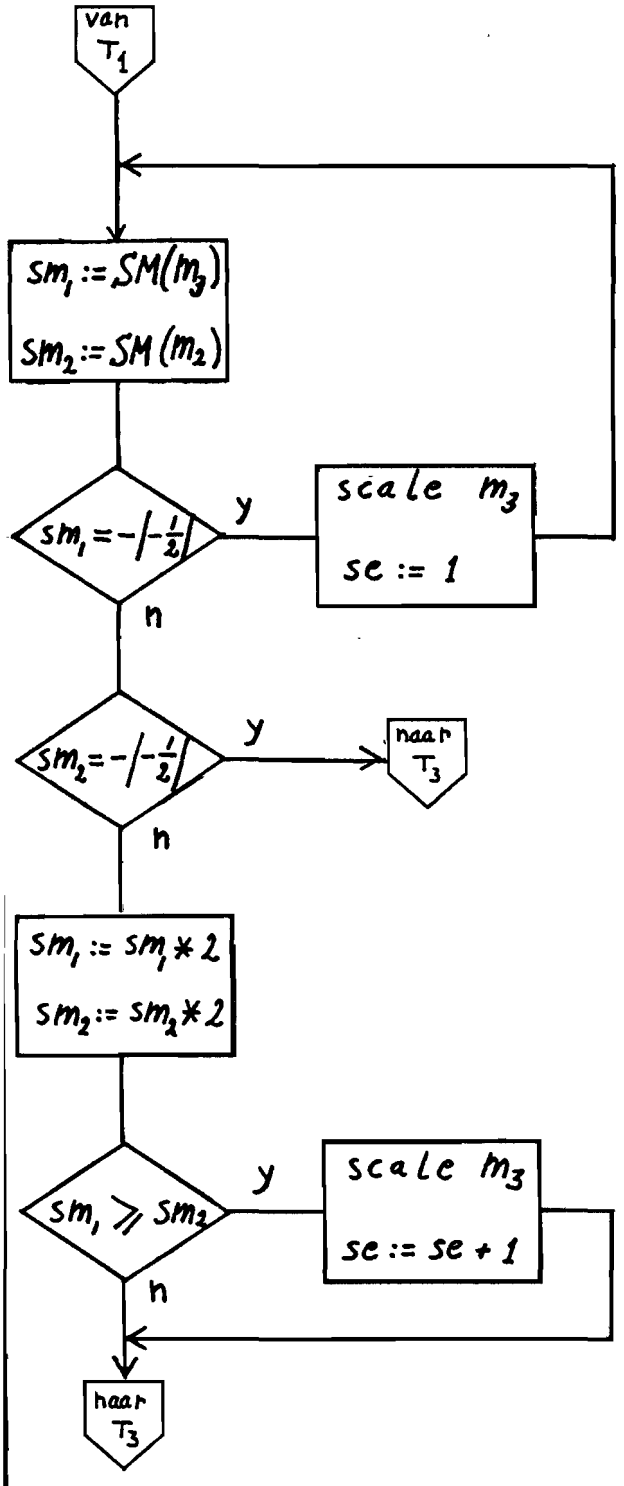
toelichting

Trek exponenten af.
Init. teller voor scale factor
Vul het Q-Register met 0; dit zijn nl. de 24 LSB van het double precision (48 bit) deeltal.
Init. "partieel quotient".

Test op deeltal = 0 i.v.m. het "dirty zero" probleem.

Test op deler=0.

ad T2) stroomdiagram



toelichting

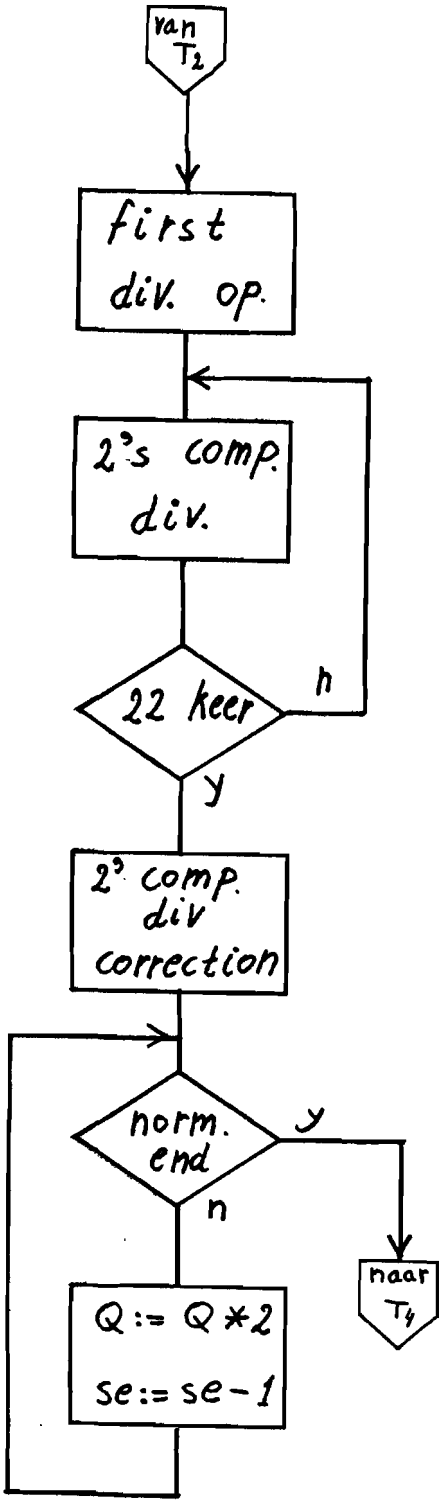
Bepaal de absolute waarde van $m_3 = m_1$ en m_2 via een 2's compl. naar sign magnitude conversie.

Test of deeltal m_1 de maximale waarde (=-1/2) heeft. Is dit het geval dan dient men m_3 ($=m_1$) te "scalen" om te voldoen aan $|\text{deler}| > |\text{deeltal}|$

Indien nu deler m_2 de maximale waarde heeft, dan is voldaan aan deze laatste eis en kan worden begonnen met het delen der mantissen.

Verwijder eerst het teken van sm_1 en sm_2 voordat getest kan worden of $|m_1| < |m_2|$ en voer dan eventueel een scaling uit.

ad T3) <u>stroomdiagram</u>



<u>toelichting</u>

Deling der mantissen in 3 fasen te weten:

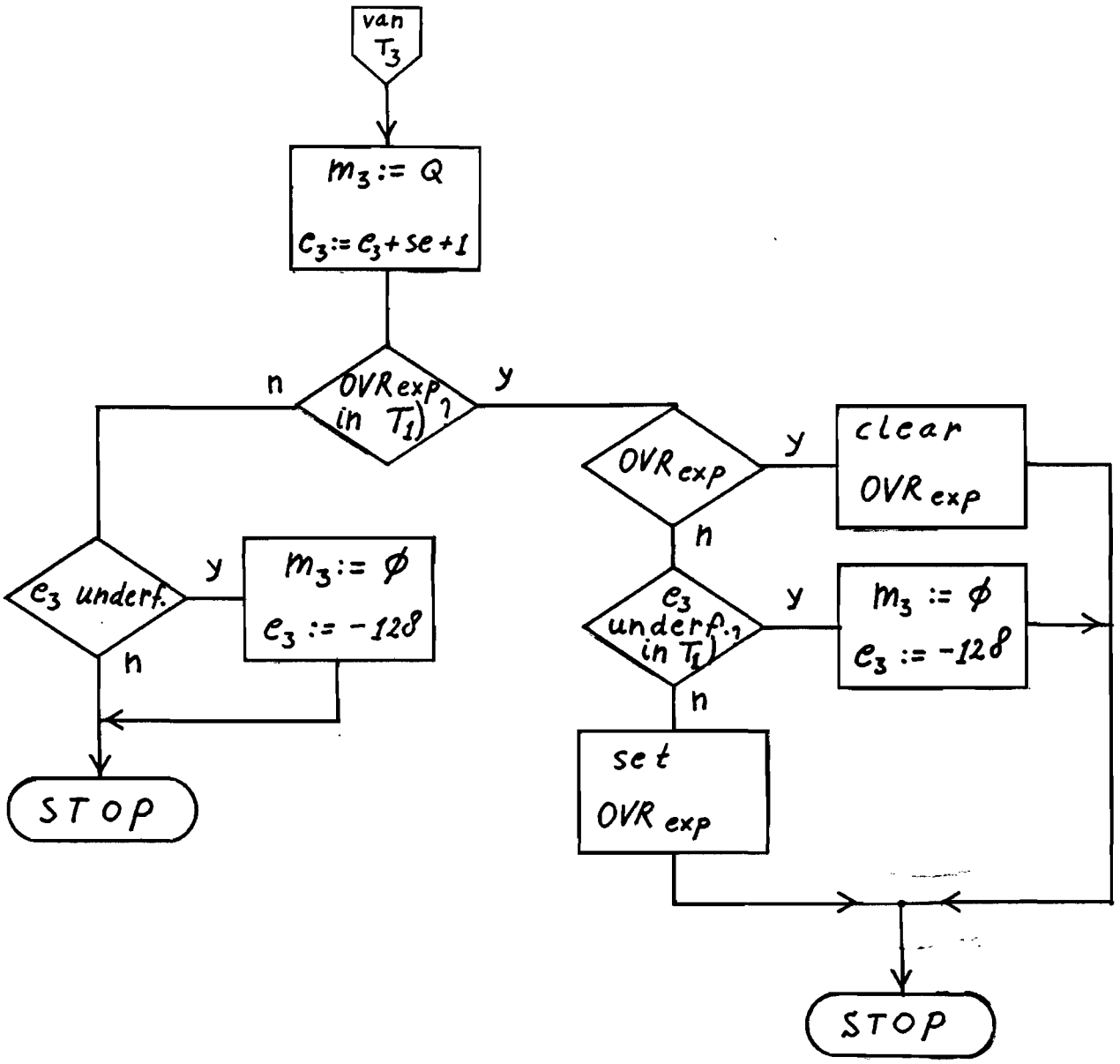
- 1 maal een First Divide Operation,
- 22 maal 2's Compl. Divide,
- 1 maal een correctie slag.

Het Q Register bevat het resultaat

De deling wordt gevolgd door de normalisatie procedure.

Opmerking:
Norm. end := Q Register normalized.

ad T4) <u>stroomdiagram</u>



toelichting

Red het quotient in een register.

Corrigeer de exponent van het resultaat met scaling factor en quotient-correctie.

Een "geringe" overflow/underflow kan worden gecorrigeerd indien in zowel T1) als in T4) de bewerking in de exponent ALU een OVR_{exp}=1 opleverde.

Leverde een van beiden - T1) of T4) - slechts een OVR_{exp}=1 op en was dit resultaat een gevolg van een underflow dan is het resultaat te klein: m_3:=0, e_3:=-128

In alle andere gevallen is OVR_{exp}=1 daar het resultaat te groot is om in het voorgeschreven formaat te passen.

Opmerkingen:

- Evenals bij FMUL het geval was dient ook hier een Boolse functie extern te worden gerealiseerd. Deze is: $CC_1 := (Y_{OVR} . Y_N)_{exp}$ en reeds gedefinieerd in FMUL.
- Het bovenstaand stroomdiagram is omgezet in een microprogramma. Voor het FDIV microprogramma zie bijlage III 1/6.
- De gemiddelde verwerkingstijd bedraagt 12 micro-seconden.
- Ook voor de signed integer divide bewerking m.b.v. Am2903 bouwstenen is simulatie programmatuur geschreven. Zie bijlage II 7/10 voor resultaten. Opgemerkt dient te worden dat het resultaat, bestaande uit remainder gevolgd door quotient, een faktor 2 te klein is, zie 2^e voorbeeld in deze bijlage. Tevens vindt er een afronding plaats in het minst significante bit van het deeltal, zie laatste voorbeeld. Voor meer informatie omtrent de deling zie (AM29) blz. 2-51.

3.2.5 Optellen / aftrekken

Basis idee

FP-optellen (of -aftrekken) is een vrij gecompliceerde FP-bewerking. Het resultaat van FP-optelling laat zich schrijven als:

som, verschil

$$\begin{cases} mant_3 := mant_1 \pm mant_2 \cdot 2^{-(\,expl - exp2\,)} \\ exp_3 := exp_1 \qquad \underline{indien}\ exp_1 \geq exp_2 \\ \\ mant_3 := mant_2 \pm mant_1 \cdot 2^{-(\,exp2 - expl\,)} \\ exp_3 := exp_2 \qquad \underline{indien}\ exp_2 > exp_1 \end{cases}$$

Problemen die zich bij het FP-optellen/-aftrekken kunnen voordoen zijn:

1) $exp_1 >> exp_2$ of $exp_2 >> exp_1$
2) $mant_3$ ligt niet in het interval $[-1/2,-1/4)$ of $[1/4,1/2)$
3) exp_3 underflow
4) exp_3 overflow

ad 1) Indien de beide exponenten zover uit elkaar liggen dat de te schuiven mantisse al zijn significante cijfers dreigt te verliezen dan hoeft deze laatste geen rol te spelen in deze FP-bewerking.

ad 2) Drie verschillende oorzaken zijn:
   a) Er ontstaan "leading zero's". Het gewenste formaat kan worden verkregen d.m.v. normaliseren mits de mantisse ongelijk aan 0 is!
   b) Is het resultaat echter gelijk aan 0 dan moet dit op een verstandige manier gecodeerd worden:

$mant_3 := 0$, $exp_3 := -128$. Omdat nl. bij optellen c.q. aftrekken de mantisse van het absoluut kleinste getal (lees: kleinste exponent) naar rechts wordt geschoven kan een "dirty zero" b.v. $0 \cdot 2^5$ er voor zorgen dat van het tweede getal $m_2 \cdot 2^{-2}$ nu $5-(-2)=7$ significante cijfers ontbreken in het eindresultaat! Dit kan worden voorkomen door de nul van de kleinst mogelijke exponentwaarde te voorzien.

c) Er treedt mantisse overflow/underflow op. Het juiste resultaat kan dan worden verkregen door de mantisse 1 positie naar rechts te schuiven met QIO_3 := (OVR exor N) van de MSS. Corrigeer het resultaat door de exponent met 1 te verhogen.

ad 3) Een exp_3 underflow resulteert in:

$mant_3 := 0$

$exp_3 := -128$

ad 4) Een exp_3 overflow resulteert in:

returncode $OVR_{exp} := 1$

Op welke wijze de FP-optelling kan worden geimplementeerd op de Am2903 wordt behandeld in de komende paragraaf.

Implementatie.

De Am2903 kent een Two's Complement Add en Subtract. Optellen c.q. aftrekken der mantissen kan direkt worden vertaald in integer bewerkingen. Voor berekeningen in de exponent ALU worden de volgende scratchpad registers gebruikt:

Register s_1) $s_1 := exp_1 - exp_2$.

s_1 geeft aan hoe groot het verschil is tussen beide exponenten.

Register s_2) Al naar gelang het teken van s_1 voldoet dit aan $s_2 := s_1 + 24$ of $s_2 := s_1 - 24$.
s_2 wordt gebruikt om te detecteren of beide FP-getallen een factor 2^{24} of meer uit elkaar liggen.

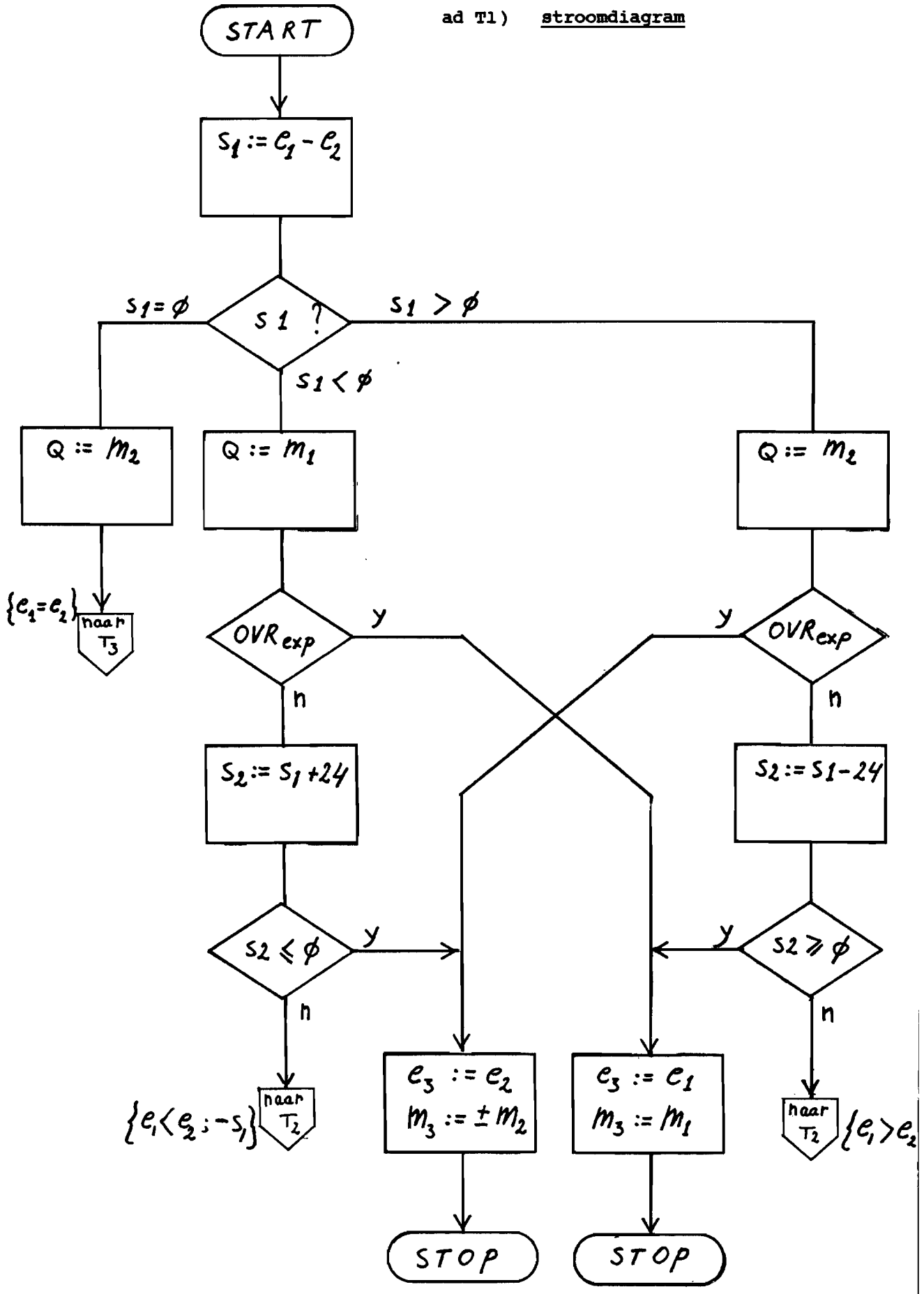
Opmerkingen:

- In de stroomdiagram treft u de notatie $\{.....\}$ aan bij de connector symbolen. Deze geven de voorwaarde aan waaronder de overgang is gemaakt.
- Verder komt in de flowchart de bewerking "$\pm$" voor. In geval van FP-optellen dient u de "+" en voor aftrekken de "-" te lezen.

De FP-optelling/-aftrekking kan worden opgesplitst in 4 taken te weten:

T1) Vergelijk de exponenten.
T2) Schuif een der mantissen totdat de exponenten gelijk zijn.
T3) Tel de mantissen nu op en stel de waarde van de exponent vast.
T4) Normaliseer de mantisse van het resultaat.

ad T1) **stroomdiagram**



toelichting

Stel d.m.v. een vergelijking vast of de exponenten:

1) Gelijk zijn.
2) Hun verschil kleiner is dan 24.
3) Hun verschil meer dan 24 bedraagt.

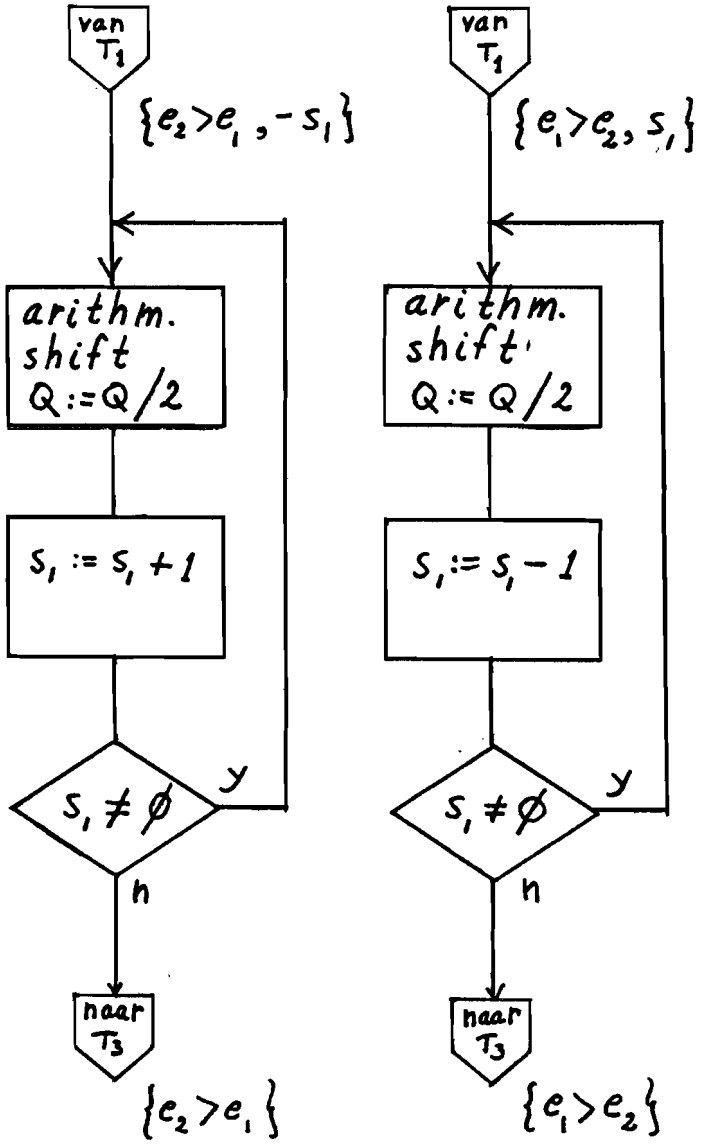
De volgende acties worden hierop ondernomen:

1) Initialiseer Q en sla T2) over.

2) Vul Q met de mantisse van het in absolute waarde kleinste FP-getal. Controleer of het verschil van de exponenten > -24 of < +24 is, zo ja ga dan naar T2).

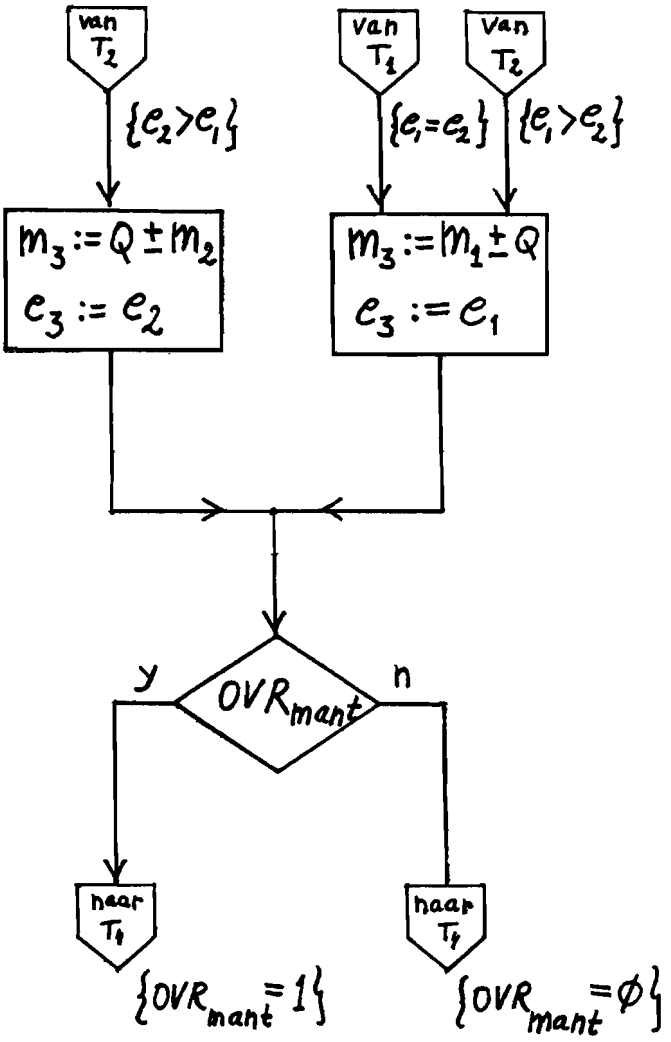
3) Is dit verschil in absolute waarde groter dan 24, of is er in de berekening van s_1 een $OVR_{exp} = 1$ geconstateerd, dan speelt het kleinste FP-getal geen rol in het resultaat.

ad T2) stroomdiagram toelichting



Schuif de mantisse -met behoud van het teken- naar rechts over een afstand aangegeven door $|s_1|$.

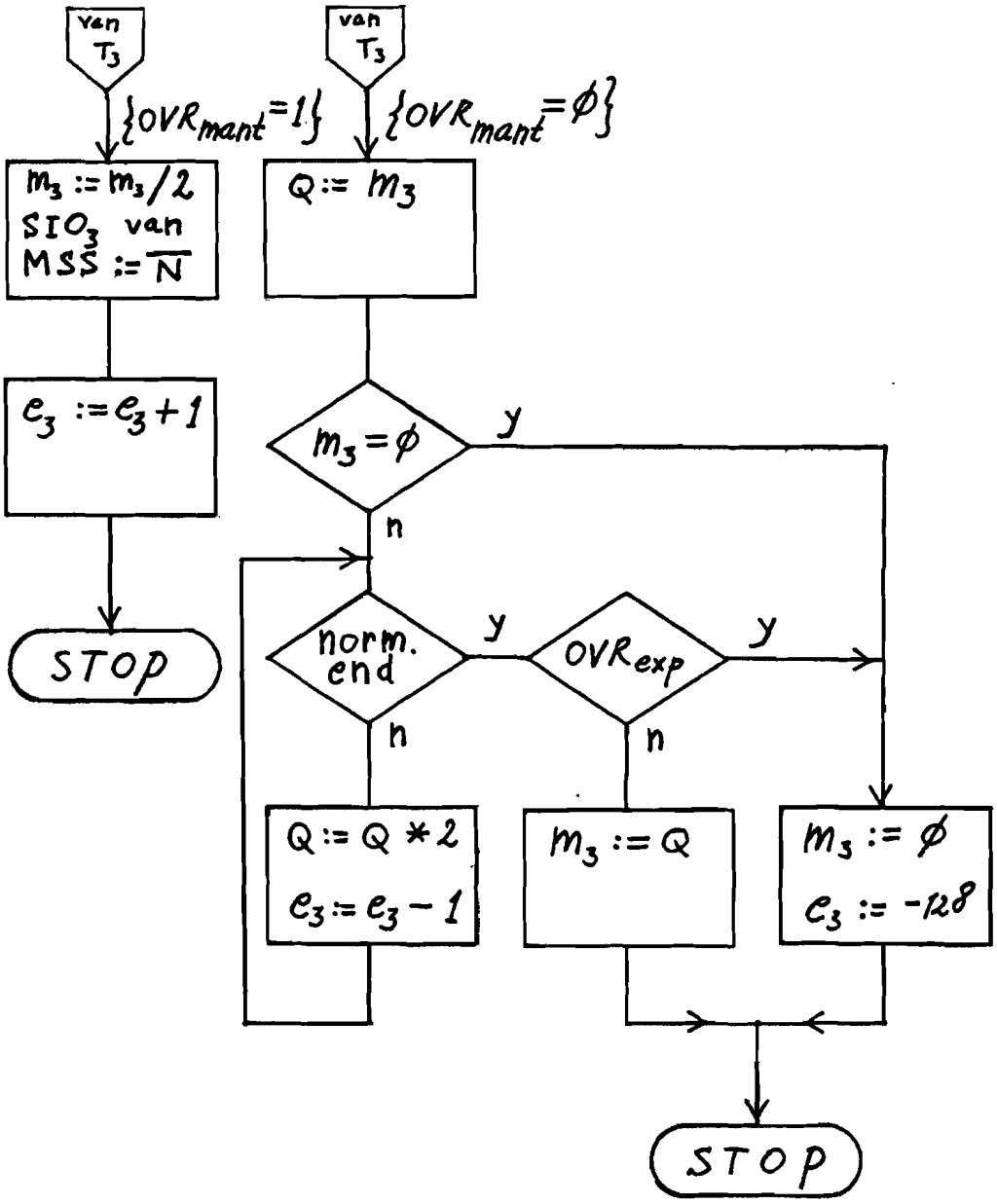
ad T3) **stroomdiagram** **toelichting**



Tel de mantissen op en bepaal de exponent van het resultaat.

Onthoud een eventuele OVR_{mant}=1.

ad T4) <u>stroomdiagram</u>



toelichting

Indien er tijdens de optelling van beide mantissen een OVR_{mant}=1 optrad dan kan deze worden gecorrigeerd door een schuifoperatie in de mantisse en een aanpassing in de exponent.

In het andere geval wordt eerst het Q Register gevuld (t.b.v. de Single Length Normalize instructie) en getest of de som niet 0 was (t.b.v. het "dirty zero"-probleem).

De normalisatiecyclus kan op 2 manieren worden verlaten:

1) Het getal in het Q Register is genormaliseerd.
2) Er trad een OVR_{exp}=1 op. (In dit geval een underflow).

Om te bepalen welk van deze 2 dan de oorzaak was dient een tweede vraag. Oorzaak:

1) resulteert in het overnemen van de genormaliseerde mantisse in het resultaat register m_3.
2) resulteert in m_3:=0 en e_3:=-128.

Opmerkingen:

- Norm. end := (Q normalized or OVR_{exp})
  Daar deze Boolse functie niet door een Am2904 kan worden samengesteld moet deze extern worden gerealiseerd. Dit resulteert in:
  CC_5 := (Y_{OVR}/)$_{mant}$. (Y_{OVR}/)$_{exp}$
- Van het bovenstaand stroomdiagram zijn de FP-optelling en -aftrekking omgezet in een microprogramma. Voor de FADD en FSUB microprogramma's zie bijlagen IV 1/6 en V 1/6.
- De verwerkingstijden van FADD zowel als FSUB hangen in sterke mate af van de operanden en varieert tussen de 3 en de 8 microseconden.

3.4 Implementatie Floating Point routines op Stack machine

3.4.1 Inleiding

Om expressies op een handige manier te verwerken kan men gebruik maken van een stack machine, zie (GEUR) blz. 86/90. Alle bewerkingen geschieden tussen operanden die in het stapelgeheugen zijn opgeslagen. Op welke wijze dit geheugen op de Register File kan worden afgebeeld wordt in par. 3.4.2 "Afspraken" behandeld. In de drie daarop volgende paragrafen worden voorbeelden van routines gegeven die op de stack machine kunnen worden verwerkt. Deze zijn:

- Vermenigvuldigen (par. 3.4.3)
- Sinus (par.3.4.4)
- Normaliseren (par. 3.4.5)

3.4.2 Afspraken

Ook hier gelden dezelfde afspraken m.b.t. de codering van het FP-getal. Aan de returncode bits OVR, N en Z wordt echter een 4e bit toegevoegd. Dit bit, het "Illegal argument"-bit, wordt door de FP-routine geset indien de functie die moet worden berekend geen zinvolle betekenis heeft voor het (de) aangeboden argument(en). Zie b.v. de natuurlijke logarithme in geval van een negatief argument. Dit returncode bit in het MSR van de 2904-B geplaatst onder het bit OVR.

Een ander onderwerp waarover een afspraak moet worden gemaakt is de implementatie van een stack-mechanisme. Het is

handig de LIFO-stack in de Register File te plaatsen. Met behulp van het Indirection Register (IR), dat als stackpointer dienst doet, kan de stack gevuld resp. geledigd worden (zie fig. 3.7). Omdat de hiervoor benodigde decrement- en increment-operaties niet in het IR kunnen plaats vinden, wordt een "schaduw"-wijzer SP' bijgehouden in de Register File. In geval van een PUSH wordt pointer SP' opgehoogd d.m.v. een Am2903 increment-instructie. In dezelfde microstap wordt het resultaat teruggeschreven in zowel SP' als stackpointer SP. In geval van een POP dient een decrement-operatie te worden uitgevoerd. Deze wordt uitgevoerd m.b.v. een optelling met -1 (inhoud hulpregister #1) daar de Am2903 geen decrement-instructie kent.

3.4.3 Vermenigvuldigen

Basis idee

Daar deze machine van origine een register machine is, kunnen alle resultaten van tussentijdse bewerkingen in registers worden opgeslagen. Alleen het ophalen van de operanden en het terugschrijven van het eindresultaat verschilt met de register machine. Het ligt dan ook voor de hand om de vermenigvuldiging beschreven in par. 3.3.3 als basis te gebruiken.

Implementatie

Het programma is uit de volgende 5 onderdelen opgebouwd:

1) POP multiplicand
2) DECREMENT stackpointer
3) POP multiplier
4) FMUL

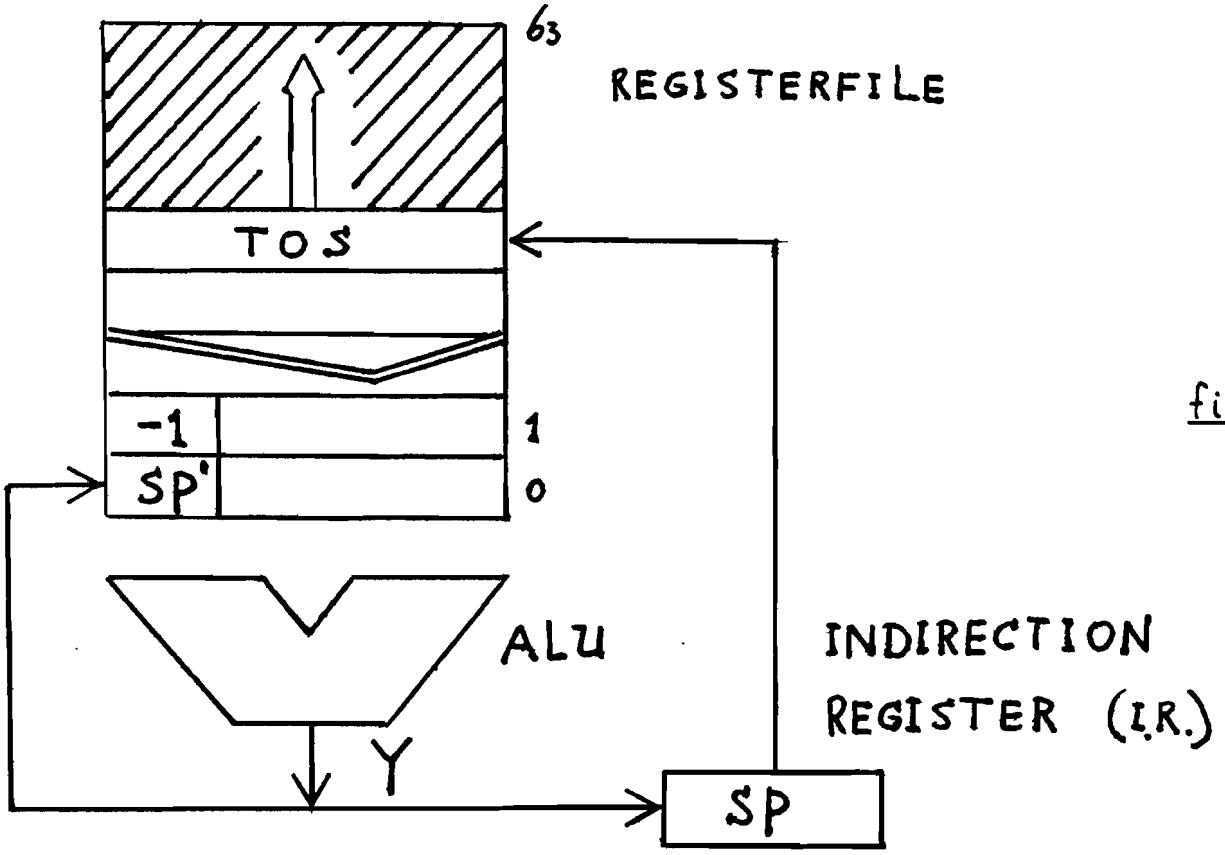

63
REGISTERFILE
TOS
-1
1
SP'
0
ALU
Y
INDIRECTION
REGISTER (I.R.)
SP

fig. 3

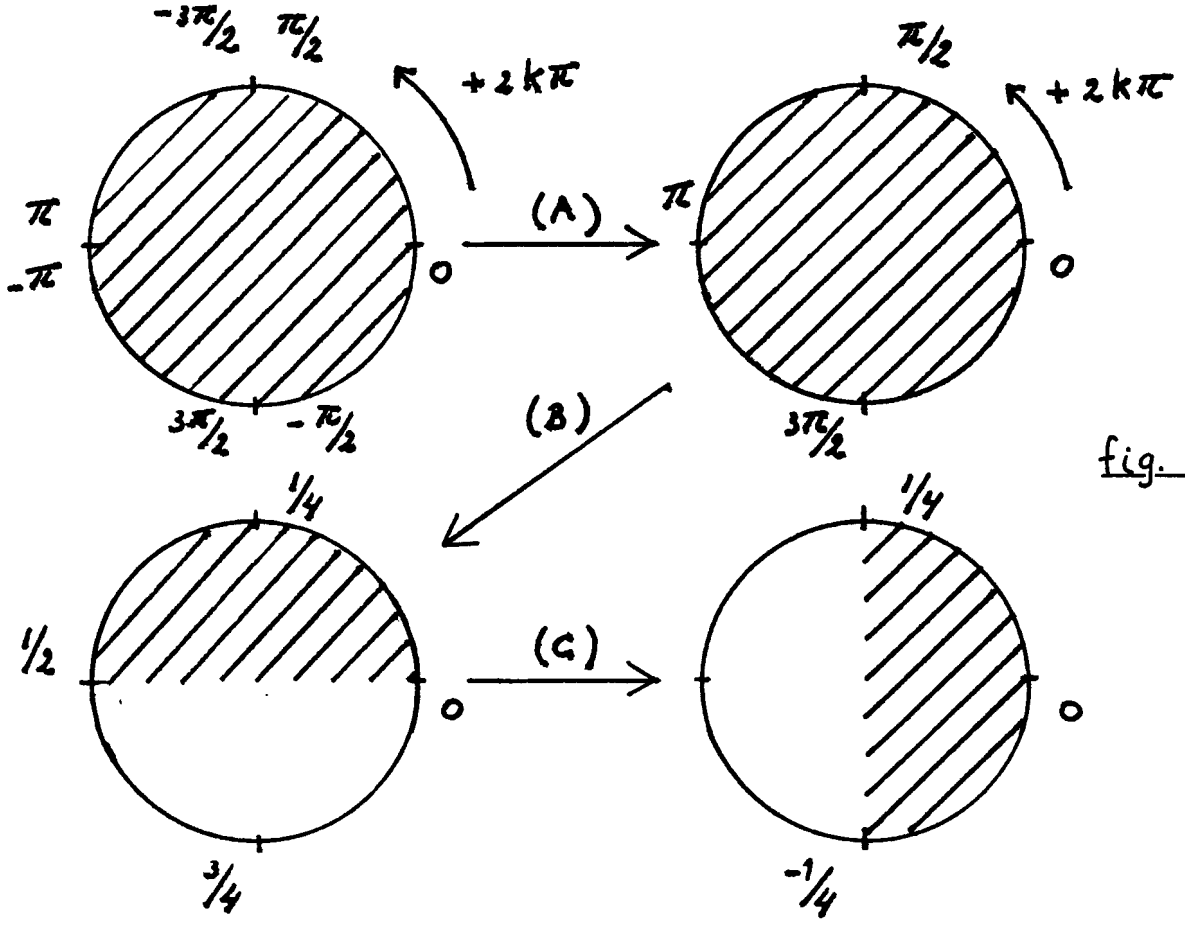

-3π/2
π/2
+2kπ
π
-π
0
(A)
3π/2
-π/2
(B)
1/4
1/2
3/4
(C)
-1/4

fig. 3

5) PUSH product

ad 1) Doordat het Indirection Register IR naar Top Of Stack TOS wijst kan direct het vermenigvuldigtal gelezen en in hulpregister #2 geplaatst worden.

ad 2) Door nu de "schaduw"-wijzer SP' met 1 te verlagen en het resultaat tevens naar IR te schrijven wordt het juiste TOS-element aangewezen.

ad 3) De vermenigvuldiger wordt in hulpregister #3 geplaatst.

ad 4) De routine FMUL (FP-vermenigvuldiging voor register machine, par. 3.2.3.) vermenigvuldigt de inhouden van hulpregisters #2 en #3 en plaatst het resultaat in register #3.

Opmerkingen:

- De gemiddelde tijdsduur van dit programma wordt geschat op 12 microseconden.
- Deze routine kent geen betekenis toe aan het "Illegal Argument"-bit.
- Zie voor het microprogramma STKFMUL bijlage VI 1/6.

3.4.4 Sinus

Basis idee

Met behulp van een polynoom kan de sinus snel en nauwkeurig worden bepaald. Bruikbaar voor ons doel is de polynoom gegeven in lit. (PDP8) blz. 8-31:

$$FSIN(Y.\pi/2) := A_1.Y + A_3.Y^3 + A_5.Y^5 + A_7.Y^7,$$

met $-1<Y\leq 1$.

De waarden van de coefficienten zijn:

$$A_1 = 1.5707949$$
$$A_3 = -0.64592098$$
$$A_5 = 0.07948766$$
$$A_7 = -0.003462476$$

Alhoewel deze routine is geschreven voor een pakket dat FP-getallen in een Sign Magnitude notatie opslaat kunnen we deze toch gebruiken. De absolute relatieve onnauwkeurigheid bedraagt als in ons geval 2^{-23}; goed voor 6 decimale cijfers.

Het argument van sin(X) dient eerst te worden teruggebracht tot het interval:

$$-\pi/2<X\leq \pi/2$$

Dit kan worden verzorgd door achtereenvolgens 3 afbeeldingen op X toe te passen, zie fig. 3.8

Afbeelding A).

Voor het bepalen van een fractie (benodigd in afbeelding B)) is het handig te werken met uitsluitend positieve argumenten. We gebruiken de gelijkheid:

$$\sin(-X) = -\sin(X), \quad \text{met } X>0.$$

Het teken dat bewaard dient te blijven kan evt. in een van de status registers (MSR of μSR) worden bewaard.

Afbeelding B).

Om de $2k\pi$-offset in het argument te elimineren kan gebruik worden gemaakt van een nieuwe maat voor de hoekgrootte: fracties i.p.v. radialen!
De afbeelding nu, bestaat uit 2 onderdelen:

1) beeld := fractie(origineel/(2π)),
met $2k\pi$ < origineel < $(2k+1)\pi$ en 0 ≤ beeld < 1.
Het bereik van dit beeld dient nog een factor 4 (afkomstig van $2\pi/(\pi/2)$) te worden verkleind.

2) Door zonodig gebruik te maken van de gelijkheid: $\sin(X) = -\sin(X-\pi)$ kan het bereik worden gehalveerd.

Afbeelding C).

Omdat het argument gereduceerd dient te worden tot het interval $-\pi/2 < X \leq \pi/2$ is nog een laatste afbeelding noodzakelijk. Indien het origineel zich binnen (1/4,1/2) bevindt kan het m.b.v. de gelijkheid $\sin(X) = \sin(\pi/2-(X-\pi/2))$ worden teruggebracht tot binnen het interval (0,1/4). Deze laatste afbeelding, inclusief het onthouden teken in MSR, levert het gevraagde interval (-1/4,1/4) op. Dit laatste interval is equivalent met $X \in (-\pi/2, \pi/2)$ en met $Y \in (-1,1)$

De berekening wordt op de stack machine uitgevoerd; we noteren dit in "Poolse notatie".

Y_s := Y Y *

FSIN := A_7 Y_s * A_5 + Y_s * A_3 + Y_s * A_1 + Y *

Opmerkingen:

- De gemiddelde verwerkingsduur bedraagt:
  V_{sinus} + 5 M_s + 3 A_s ≈ 10 + 5*11 + 3*5 = 80

microseconden. In deze formule betekenen de symbolen V_{sinus}, M_s, A_s achtereenvolgens : geschatte voorbereidingstijd i.v.m. de afbeeldingen en de tijden betreffende de stack-vermenigvuldiging en stack-optelling.

- De nauwkeurigheid bedraagt (lit. (PDP8):
  6 cijfers indien $|X| < \pi/2$ en daarbuiten 5 cijfers voor $|X| < 200$.
- Naarmate het argument groter wordt neemt ook de fout, veroorzaakt in de bepaling van de fractie, toe (zie vorige opmerking). Om te signaleren wanneer het argument een bepaalde drempel overschrijdt kan van de "Illegal argument" returncode gebruik worden gemaakt.

Implementatie

Wegens tijdgebrek kon deze FP-routine niet d.m.v. een microprogramma op de stack machine worden geimplementeerd.

3.4.5 Normaliseren

Basis idee

Om te garanderen dat na FP-bewerkingen het resultaat in het gewenste formaat is gecodeerd kan de FP-normalize routine worden gebruikt. De reeds besproken FP-routines bezitten allen een -op hun FP-bewerking afgestemd- microcode gedeelte voor het normaliseren van het resultaat. Omdat het normaliseren in principe na elke FP-bewerking uitgevoerd dient te worden en rekening houdend met toekomstige uitbreidingen van het FP-pakket (zie hoofdstuk 5) hebben we besloten ook deze routine in het pakket op te nemen.

Het algorithme voor deze FP-normalize instructie laat zich schrijven als:

```
while mant3 not normalized
      do  mant3 := mant3 * 2;
          exp3  := exp3 - 1
      od;
```

De volgende problemen kunnen optreden:

1) $Mant_3 = 0$.
2) Exp_3 underflow.

ad 1,2) In feite kunnen beide problemen worden opgelost door $mant_3 := 0$ en $exp_3 := -128$ te maken.

Op welke wijze de FP-normalize kan worden geimplementeerd op de Am2903 wordt behandeld in "Implementatie".

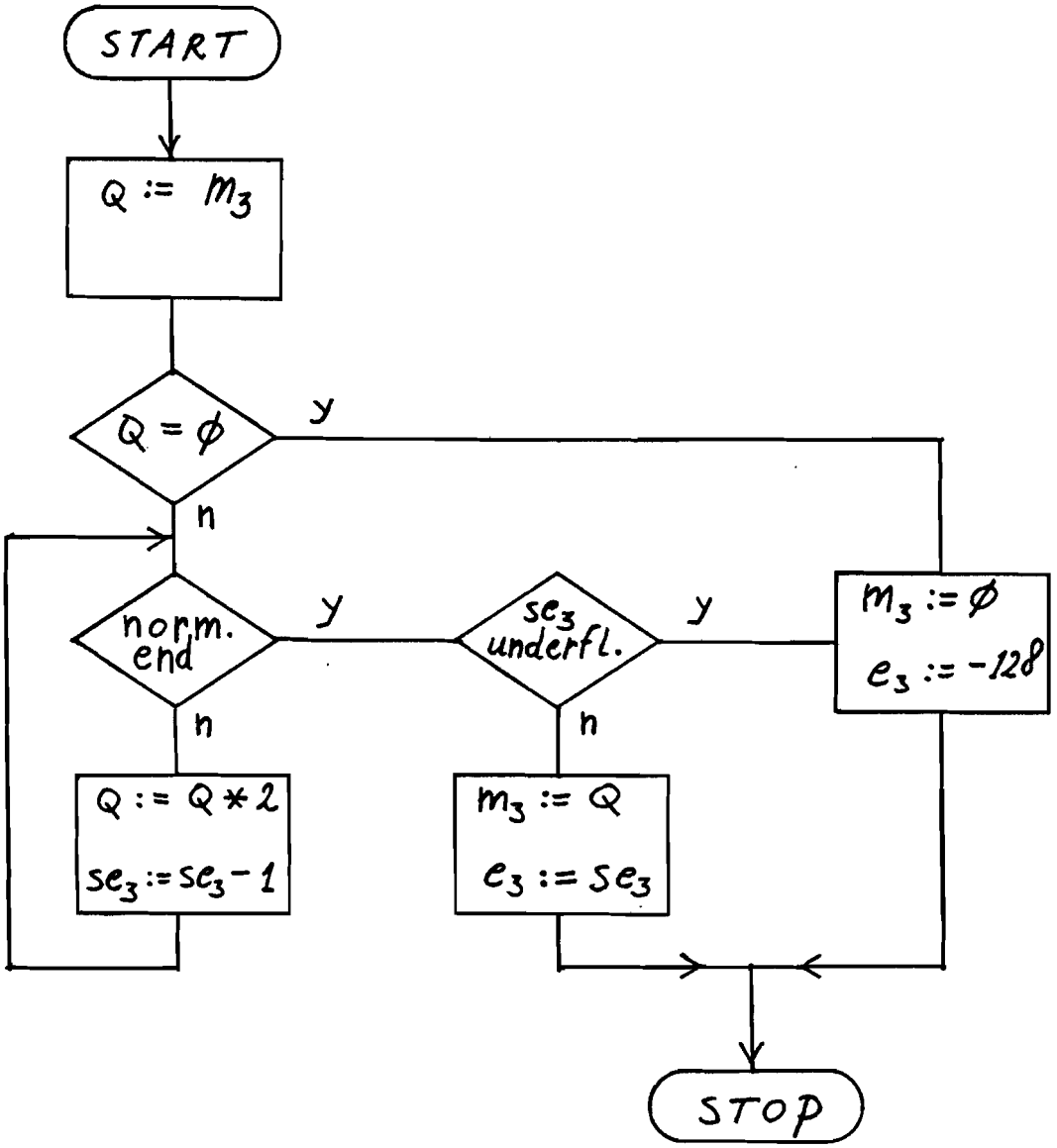
Implementatie

De Am2903 kent een Single Length Normalize operatie die wordt uitgevoerd m.b.v. het Q register. Dit register wordt geladen met het top element van de stack (TOS) om na normalisatie weer naar TOS terug te worden geschreven.

FNORM is voorts transparant m.b.t. de Negative- (N) en de Zero-flag (Z). Dit houdt in dat het deze flags van het aanroepende programma niet aantast. In die FP-bewerkingen waarin beide OVR-bits een rol spelen, kan een toegevoegde test dit probleem opvangen.

De FP-normalize kan als volgt worden gedefinieerd (zie stroomdiagram):

stroomdiagram



toelichting

Vul Q register met mantisse-deel van TOS (Top Of Stack); is deze 0 dan TOS := 0 . 2^{-128} .

Schuif de mantisse naar links totdat hij genormaliseerd is of totdat er een exponent underflow is opgetreden. In dit laatste geval wordt het resultaat 0! Zo niet dan bevat TOS het genormaliseerde resultaat.

Opmerkingen:

- Norm. end. := (Q normalized or OVR_{exp})
  Ook deze Boolse functie dient extern te worden gerealiseerd (Condition Code CC5).
- De gemiddelde verwerkingstijd hangt af van het aangeboden FP-getal en kan varieren van 2 tot 8 microseconden.
- Dit stroomdiagram is omgezet in het microprogramma FNORM, zie bijlage VII 1/4.

4 De geheugenkaart

4.1 Inleiding

Daar het de bedoeling is een kompleet systeem te bouwen, is behalve een ALU ook een geheugenkaart ontworpen. De voornaamste ontwerpeisen waren:

1) Geheugengrootte: 128K bytes, zowel byte- als woordadresseerbaar.
2) Error Detecting mogelijkheden.
3) Multibus compatible.

In par. 4.2 wordt op de bouw en de werking nader ingegaan.

4.2 Opbouw en werking

Het geheugen is gebaseerd op een 16K*1 bit dynamisch geheugen IC: de 2117-2. De timing voor lees-, schrijf- of refresh-acties wordt verzorgd door een 8202 Dynamic RAM Controller. Voor het schema van deze kaart zie fig. 4.1.

Voor ons doel is de Multibus met de volgende signalen uitgebreid (zie tabel in fig. 4.2):

- Adreslijnen ADR_{14}...ADR_{17}, direct adresseerbaar: 16Mbyte.
- Pariteitlijnen P_1...P_5, voor foutendetectie ingeval van Adres- en Data-transport.
- "Ready"-lijn RDY, geeft aan wanneer de master synchroniseert met het geheugen.
- "Refresh Request"-lijn RFRQ/, externe refresh opdracht aan dynamisch geheugen.

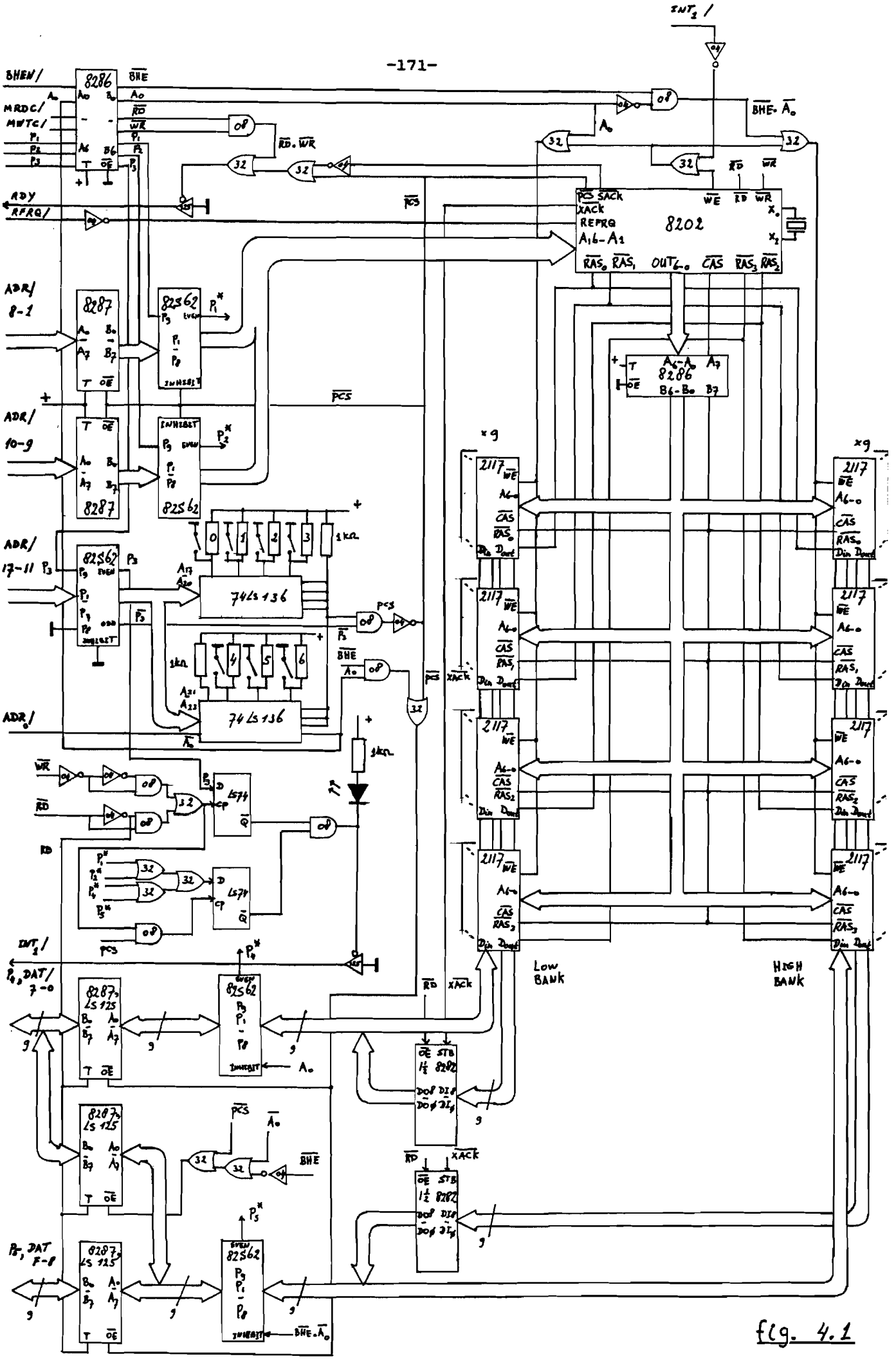

8286
8287
82562
8202
74LS136
LS74
LS125
2117
8282
Low BANK
HIGH BANK

fig. 4.1

Interface beschrijving

Voedingslijnen	12V, 5V, -5V, GND
Adres-lijnen	ADR_{17}/...ADR_{0}/ met pariteitbits P_1, P_2, P_3.
Byte High Enable	BHEN/
Data-lijnen	DAT_{F}/...DAT_{0}/ met pariteitbits P_4, P_5.
Memory Read	MRDC/
Memory Write	MWTC/
External Refresh Request	RFRQ/
Ready-lijn	RDY
Interrupt	INT_1/

figuur 4.2

Timing

t_{AS}	Address Set-Up Time	197	nanoseconden
t_{WI}	Write Command to Interrupt Time	221	—
t_{WRR}	Write Command to Ready Time	171	—
t_{RA}	Read Command Access Time	285	—

figuur 4.3

De 8202 kan maximaal 64K-byte dynamisch geheugen aansturen. Het geheugen is echter verdeeld in 2 banken: elk 9 bit breed (byte inclusief pariteitbit). De bank kan worden geselecteerd m.b.v. het Multibus signaal $ADR_0/$.

De kaartselect (PCS/) reageert op een bepaalde combinatie van $ADR_{17}/$...$ADR_{11}/$ die d.m.v. schakelaars 0 t/m 6 ingesteld kan worden. De data- zowel als adres-lijnen zijn, conform Multibus specificaties lit. (BART), "laag" actief. Om nu de juiste polariteit te herstellen en om de Multibus te ontlasten zijn deze signalen gebufferd door middel van transceivers (8287 of 8286 noninverting).

Indien de pariteit van een data- of adres-byte, inclusief het pariteitbit, even is dan wordt door de kaart een interrupt gegenereerd. Dit gebeurt op de neergaande flank van het WR/-signaal of op de opgaande flank van het RD/-signaal, zie fig. 4.4. Tevens brandt, voor de duur van $INT_1/$, een LED op de kaart.
Deze melding wordt pas gereset tijdens de volgend optredende kaartselect. Zie voor de timing de tabel in fig. 4.3.

Opmerkingen:

- De frequentie van het gekozen kristal voor de 8202 bedraagt 25 MHz.

- De "Ready"-lijn dient aan de processorkaart via een pull-up weerstand met de 5 V te zijn verbonden.

- Ten behoeve van het pariteitbit zijn de transceivers voor DAT_F-DAT_0 met 1 bit verbreed, zie fig. 4.5.

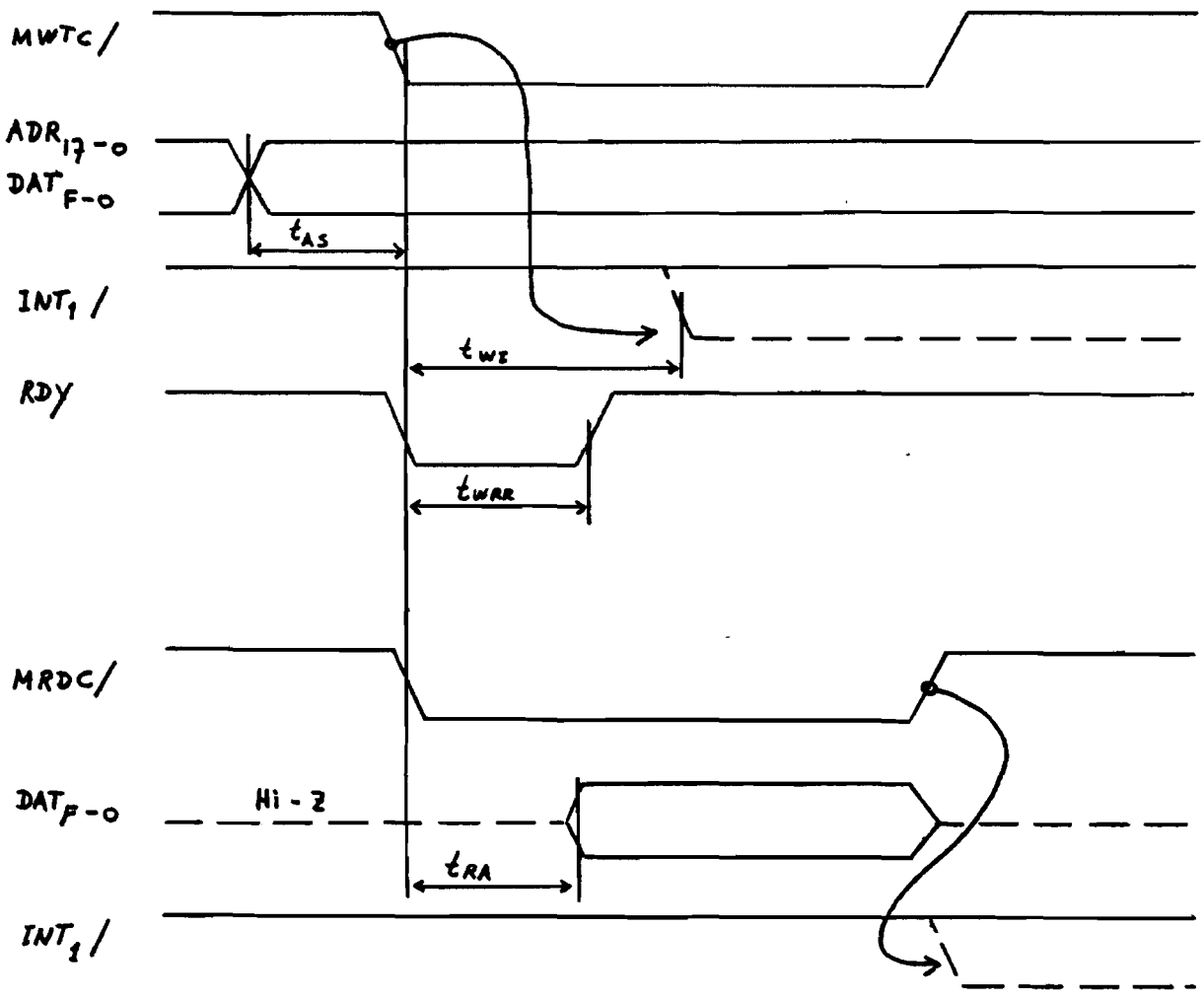

MWTC/
ADR17-0
DATF-0
tAS
INT1/
tWI
RDY
tWRR
MRDC/
DATF-0
Hi-Z
tRA
INT1/

fig. 4.4

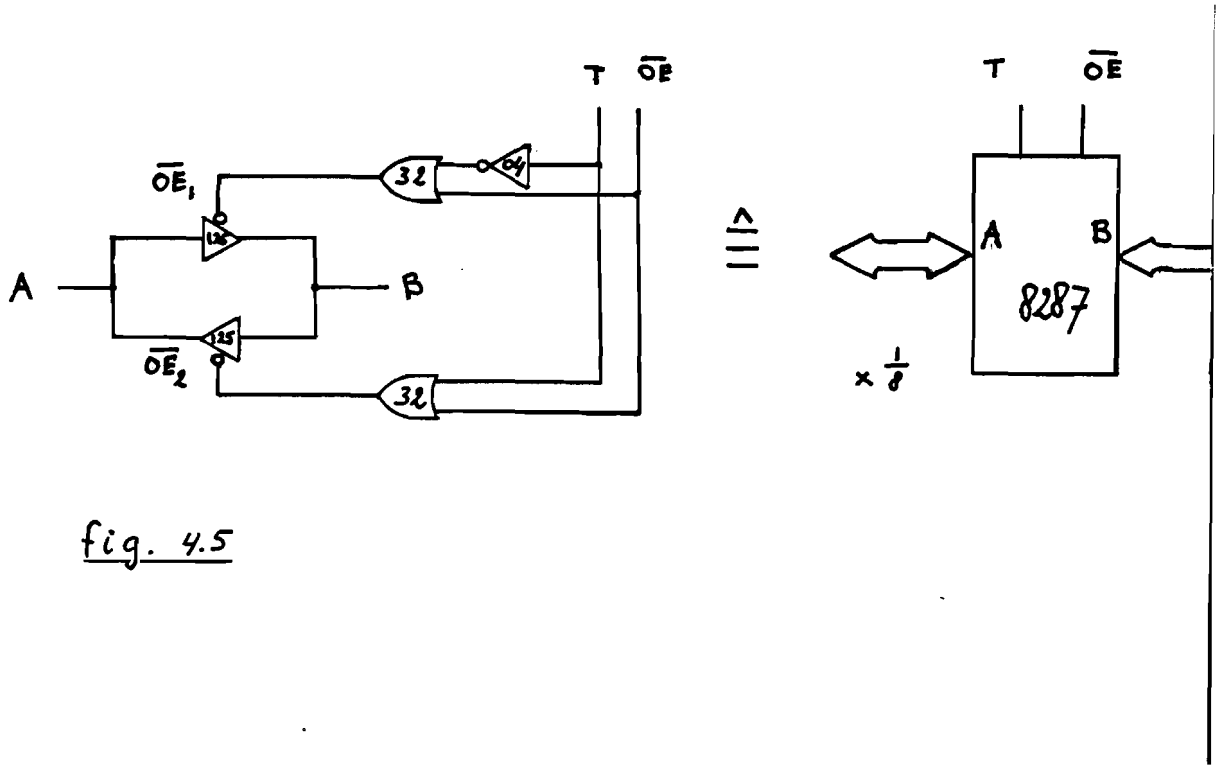

T
OE
OE1
32
04
A
126
B
125
OE2
32
≙
T
OE
A
B
8287
× 1/8

fig. 4.5

5 Conclusies en suggesties

Voordat dit project succesvol afgesloten kan worden, dient men nog veel werk te verzetten. Er dienen nog ontwerpen gemaakt te worden ten aanzien van de BIU, de CCU en een universele I/O kaart. Het systeem moet vervolgens gebouwd en getest worden. Verder is er nog software nodig om het systeem toegankelijk te maken voor toekomstige gebruikers.

Men doet er verstandig aan eerst de navolgende punten in overweging te nemen, alvorens over te gaan tot de bouw van de ALU zoals die in dit rapport beschreven is.

- In het huidige ontwerp zijn alle ALU configuraties op dezelfde groep 2903's geprojecteerd. Een andere mogelijkheid is dat elke ALU configuratie zijn eigen 2903's krijgt toegewezen. Het is aan te bevelen om eerst een prijs/prestatie onderzoek te verrichten voordat men een keuze maakt.

- Van de aangekondigde Am29203 zijn inmiddels (augustus '81) al meer gegevens bekend. Men dient te onderzoeken of het niet verstandig is de 2903's te vervangen door 29203's en het ontwerp hierop aan te passen. Met name het Carry-out Register wordt dan overbodig.

- In het huidige ontwerp is weinig moeite gedaan om de breedte van het microwoord te beperken. Men zou o.a. bits kunnen besparen door velden te coderen en meer gemeenschappelijke velden te gebruiken. Decoderen echter kost extra tijd en hardware. Ook hier dient een prijs/prestatie onderzoek verricht te worden.

- De geheugenkaart is voorzien van "error-detecting" mogelijkheden door toepassing van pariteitsbits. Het is echter ook mogelijk een ontwerp te maken waarbij bepaalde fouten

gecorrigeerd worden. Men zou hier bijvoorbeeld de Am2960 (Cascadable 16-bit Error Detection and Correction Unit) en de Am2961/Am2962 (4-bit Error Correction Multiple Bus Buffers) kunnen toepassen.

Bij het ontwerp van de micro-assembler en het schrijven van de Floating Point routines kwamen enkele nadelen van de AMDASM 29 meta-assembler aan het licht. Deze zijn:

- De substitutie volgorde van argumenten in format names is star. Dit betekent bijvoorbeeld dat een argument van een volgend in te vullen veld geen invloed meer kan hebben op de huidige substitutie. Een eenmaal vastgelegd veld is namelijk niet meer te wijzigen.

- Er zijn slechts beperkte bewerkingen m.b.t. de argumenten mogelijk. Dit betekent o.a. dat de waarden van de velden die t.a.v. de timing ingevuld moeten worden niet m.b.v. deze assembler uitgerekend kunnen worden.

- De breedte van de in te vullen microwoorden is in deze assembler begrensd op 128 bits. We hebben in dit ontwerp echter een microwoord ter breedte van 196 bits.

Deze nadelen kunnen worden ondervangen bij gebruik van een macro-assembler, die dan echter andere bezwaren met zich mee brengt. De tijdsduur van een assemblage bijvoorbeeld, kan dan sterk toenemen.

Een vergelijking van verwerkingstijden van onze FP-ALU met andere arithmetische processoren zoals de Am9511A en de 8087 levert de resultaten op zoals die in de tabel van figuur 5.1 zijn weergegeven. (zie verder lit.(ISBC,AMZ8)). Deze tijden hebben allen betrekking op een 32 bit FP-formaat. Bij de FP-ALU gaat het hierom geschatte tijden.

Routine	FP-ALU	8087	Am9511A
FMUL	11 µs	19 µs	48 µs
FDIV	12 µs	39 µs	51 µs
FADD	3-8 µs	14-18 µs	19-115 µs
FSIN	80 µs	200 µs	1280 µs

figuur 5.1

Uit de tabel blijkt dat, voor zover deze voorbeelden representatief zijn voor het totaal gedrag, onze machine gemiddeld een factor 2 sneller is dan zijn naaste concurrent, de 8087. De Am9511A staat geheel buitenspel.

Het geschreven FP-pakket is verre van volledig. Uitbreidingen zijn in verscheidene richtingen mogelijk:

1) Extended Arithmetic Functions
2) Floating Point Utilities
3) Dedicated Routines

ad 1) Dit omvat de goniometrische functies zoals cosinus, tangens, arcustangens. Verder de vierkantswortel, exponent, logarithme, random number generator etc.. Al deze functies kunnen, op enkele na, d.m.v. polynomen of kettingbreuken met de vereiste nauwkeurigheid worden benaderd (b.v. sinus, zie par. 3.4.4; verder lit. (IBM1,PDP8)).

ad 2) Onder de noemer "Floating Point Utilities" bevinden zich al die routines die niet direkt betrokken zijn bij rekenkundige bewerkingen, maar toch noodzakelijk zijn voor een soepele afhandeling hiervan. Enkelen zijn:
- Vergelijkings routines tussen verschillende formaten, b.v.:

FP-formaat	met	BCD-formaat
FP-formaat	met	integer-formaat

- Conversie routines, b.v.:

integer	<==>	BCD
integer	<==>	Floating Point
single precision	<==>	double precision

ad 3) Bij gerichte toepassingen zoals bij Digitale Signaal Bewerking of Netwerk Simulatie kan er behoefte bestaan aan apparatuur die, op een groot aantal FP-argumenten, snel een FP-bewerking kan verrichten, b.v.:

Fast Fourier Transform
Auto- en Kruis-correlatie
Matrixbewerkingen

Literatuurlijst

(MICK) Mick J. and Brick J., Bit-slice Microprocessor Design, McGraw-Hill Book Company, 1980.

(SALI) Salisbury A.B., Microprogrammable Computer Architectures, American Elsevier Publishing Co., 1976.

(HUSS) Husson S.S.,Microprogramming Principles and Practices, Prentice-Hall, 1970.

(FLOR) Flores I., The Logic of Computer Arithmetic, Prentice-Hall, 1963.

(LYUS) Lyusternik L.A. et al, Handbook for Computing Elementary Functions, Pergamon Press, 1965.

(AMDS) Advanced Microprogramming Development System, System 29/05 Manual, 1978.

(GEUR) Geurts A.G.M., Dictaat Besturingsprogrammatuur I, TH Eindhoven afdeling E, concept augustus 1979.

(COON) Coonen, A implementation guide to a proposed standard for floating point arithmetic, computer jan 1980, pages: 86-71.

(PDP8) PDP8 Handbook Series, Introduction To Programming, Digital, 1973.

(IBM1) IBM Series/1, Mathematical and Functional Subroutine Library: User's Guide, 1979.

(JACO) Jacobs J.W.M., Microprogrammeringsmodel, Stageverslag TH Eindhoven afd. E vakgr. EB, juli 1980.

(AM29) AM2900 Family Data Book, Advanced Micro Devices, 1979.

(TEXA) The TTL Databook for Design Engineers, Texas instruments, 1979.

(BART) Barthmayer J., Intel Multibus Interfacing, Intel Application Note AP-28A, jan. 1979.

(AMDD) Advanced Micro Devices Data Book, juli 1974.

(INTC) Component Data Catalog, Intel, 1979.

(INTP) Peripheral Design Handbook, Intel feb. 1979.

(NATI) National Semiconductor CMOS data book, 1977.

(AMZ8) AmZ8000 Family Data Book, 1980.

(ISBC) iSBC 337 Multimodule Numeric Data Processor, Datasheet 143087, Intel Corporation, 1980.

```
;       ************************************************************
;       *                                                          *
;       *       DEFINITION FILE FOR PART 1 OF THE CONTROL-         *
;       *       VECTOR FOR THE FLOATING POINT ALU.                 *
;       *                                                          *
;       *                                                          *
;       *       JULY 1981,              P. RAEDTS                  *
;       *                               J. JACOBS                  *
;       *                                                          *
;       ************************************************************

        WORD    112

;       FIELD SPECIFICATION OF THE FIRST 111 BITS OF THE MICRO-
;       WORD:

;             BIT   0-  2     FP ALU CONFIGURATION
;             BIT   3- 22     EXPONENT ALU CONTROL VECTOR
;             BIT  23- 63     MANTISSA ALU CONTROL VECTOR
;             BIT  64- 98     FP ALU ADDRESSING CONTROL
;             BIT  99-111     COPY & SWAP CONTROL
```

```
;       ****************************************************************
;       *                                                              *
;       *       BASIC EQUATES                                          *
;       *                                                              *
;       ****************************************************************



;       REGISTERS ( NO MNEMONICS FOR R16 THRU R63 )

R0:     EQU     H#0
R1:     EQU     H#1
R2:     EQU     H#2
R3:     EQU     H#3
R4:     EQU     H#4
R5:     EQU     H#5
R6:     EQU     H#6
R7:     EQU     H#7
R8:     EQU     H#8
R9:     EQU     H#9
R10:    EQU     H#A
R11:    EQU     H#B
R12:    EQU     H#C
R13:    EQU     H#D
R14:    EQU     H#E
R15:    EQU     H#F



;       2903 CONTROL & FUNCTIONS

SPF:    EQU     H#0             ;SPECIAL FUNCTIONS
HIGH:   EQU     H#0             ;ALU OUTPUT F FORCED HIGH
SSR:    EQU     H#1             ;S MINUS R
SRS:    EQU     H#2             ;R MINUS S
ADD:    EQU     H#3             ;R ADD S
PAS:    EQU     H#4             ;PASS S
COMS:   EQU     H#5             ;2'S COMPLEMENT S
PAR:    EQU     H#6             ;PASS R
COMR:   EQU     H#7             ;2'S COMPLEMENT R
LOW:    EQU     H#8             ;ALU OUTPUT FORCED LOW
CRAS:   EQU     H#9             ;COMPLEMENT R, AND WITH S
XNRS:   EQU     H#A             ;R EXCLUSIVE NOR WITH S
XOR:    EQU     H#B             ;R EXOR S
AND:    EQU     H#C             ;R AND S
NOR:    EQU     H#D             ;R NOR S
NAND:   EQU     H#E             ;R NAND S
OR:     EQU     H#F             ;R OR S
```

DEFINITION FILE FOR FLOATING POINT ALU (FIRST HALF)

```
;       ************************************************************
;       *                                                          *
;       *       MACHINE DEPENDENT EQUATES                          *
;       *                                                          *
;       ************************************************************



;       ALU SPECIAL FUNCTIONS

UMUL:   EQU     9H#000;         ;UNSIGNED MULTIPLY
TCM:    EQU     9H#002;         ;2'S COMPLEMENT MULTIPLY
INC:    EQU     9H#004;         ;INCREMENT BY ONE OR TWO
SMTC:   EQU     9H#185;         ;SIGN MAGNITUDE - 2'S COMPLEMENT
TCMC:   EQU     9H#006;         ;2'S COMPLEMENT MULTIPLY LAST STEP
SLN:    EQU     9H#008;         ;SINGLE LENGTH NORMALIZE
DLN:    EQU     9H#00A;         ;DOUBLE LENGTH NORMALIZE
TCFD:   EQU     9H#00A;         ;2'S COMPLEMENT FIRST DIVIDE OPERATION
TCD:    EQU     9H#00E;         ;2'S COMPLEMENT DIVISION
TCDC:   EQU     9H#00E;         ;2'S COMPLEMENT DIVISION CORRECTION



;       ALU DESTINATION MODIFIERS

ADR:    EQU     9H#000;         ;ARITH SHIFT DOWN, RESULT INTO RAM
LDR:    EQU     9H#181;         ;LOGICAL SHIFT DOWN, RESULT INTO RAM
ADRQ:   EQU     9H#002;         ;ARITH SHIFT DOWN, RESULT INTO RAM AND Q
LDRQ:   EQU     9H#183;         ;LOGICAL SHIFT DOWN, RESULT INTO RAM AND Q
RPT:    EQU     9H#004;         ;RESULT INTO RAM, GENERATE PARITY
LDQP:   EQU     9H#185;         ;LOGICAL SHIFT DOWN Q, GENERATE PARITY
QPT:    EQU     9H#006;         ;RESULT INTO Q, GENERATE PARITY
RQPT:   EQU     9H#187;         ;RESULT INTO RAM AND Q, GENERATE PARITY
AUR:    EQU     9H#008;         ;ARITH SHIFT UP, RESULT INTO RAM
LUR:    EQU     9H#189;         ;LOGICAL SHIFT UP, RESULT INTO RAM
AURQ:   EQU     9H#00A;         ;ARITH SHIFT UP, RESULT INTO RAM AND Q
LURQ:   EQU     9H#18B;         ;LOGICAL SHIFT UP, RESULT INTO RAM AND Q
YBUS:   EQU     9H#00C;         ;RESULT TO Y BUS ONLY
LUQ:    EQU     9H#18D;         ;LOGICAL SHIFT UP Q
SINX:   EQU     9H#00E;         ;SIGN EXTEND
REG:    EQU     9H#18F;         ;RESULT TO RAM
YBOFF:  EQU     9H#1FF;         ;YBUS THREE STATE OUTPUT



;       ALU CONFIGURATION

CFLOAT: EQU     B#101           ;SELECT FLOATING POINT MODE
CDINT:  EQU     B#011           ;SELECT DOUBLE INTEGER MODE
CSINT:  EQU     B#110           ;SELECT SINGLE INTEGER MODE
```

```
;       ALU SOURCES & DESTINATIONS

QR:             EQU     B#101   ;Q REGISTER
PL:             EQU     B#10    ;PIPELINE REGISTER
DB:             EQU     B#01    ;DATA INPUT ON DB 2903
IR:             EQU     B#11    ;INDIRECTION REGISTER
AR:             EQU     B#00    ;ADDRESS REGISTER
YB:             EQU     B#0     ;YBUS 2903


;       OTHER CONTROL VECTORS

E:              EQU     B#11    ;EXPONENT ALU EXTERNAL DATA: OFF
M:              EQU     B#11    ;MANTISSA ALU EXTERNAL DATA: OFF
YON:            EQU     B#0     ;OEY/ 2903
YOFF:           EQU     YON*
ALUON:          EQU     B#0     ;IEN/ 2903
ALUOFF:         EQU     ALUON*
IOSET:          EQU     B#1     ;SET IO INSTRUCTION BIT 2903
IOCLEAR:        EQU     IOSET*
XCEIVOFF:       EQU     N#FFF   ;TRANSCEIVERS FOR COPY CONTROL: OFF
XDOFF:          EQU     B#11    ;ALU EXTERNAL DATA: OFF
```

```
;       ****************************************************************
;       *                                                              *
;       *       SUBFORMAT DEFINITIONS                                  *
;       *                                                              *
;       ****************************************************************



;       EXPONENT ALU

SUB0:   SUB     ALUON,5V:H#0F:,55X,6V%X,6V%X,6V%X,2X,IOCLEAR,23X
SUB0A:  SUB     5V:H#0F:,55X,6V%X,6V%X,6V%X,2X,IOCLEAR,23X
SUB1:   SUB     ALUON,5V:H#0F:,55X,6V%X,6V%X,6V%X,2X,IOSET,23X
SUB2:   SUB     ALUON,5V:H#0F:,55X,6V%X,6V%X,6V%X,26X



;       MANTISSA ALU

SUB4:   SUB     ALUON,ALUON,9VH#18F:,30X,6V%X,6V%X,6V%X,
/               9X,IOCLEAR,16X
SUB4A:  SUB     9VH#18F:,30X,6V%X,6V%X,6V%X,
/               9X,IOCLEAR,16X
SUB5:   SUB     ALUON,ALUON,9VH#18F:,30X,6V%X,6V%X,6V%X,
/               9X,IOSET,16X
SUB6:   SUB     ALUON,ALUON,9VH#18F:,30X,6V%X,6V%X,6V%X,26X



;       REGISTER ADDRESS CONTROL

SUB8:   SUB     4X,6V%X,6V%X,6V%X,26X
SUB9:   SUB     2VX,3V%X,2VX
SUB10:  SUB     2VX,1X,2V:X,2VX



;       SWAP & COPY CONTROL

SUB11:  SUB     3X,PAS,ALUON
SUB12:  SUB     4X,DB,8X,PAS,ALUON,ALUON
SUB13:  SUB     6X,DB,25X,XDOFF,B#1,6V%X,6V%X,6V%X,PL,B#0,PL,PL,PL,B#0,PL,PL
```

```
;       ****************************************************************
;       *                                                              *
;       *       FORMAT DEFINITIONS                                     *
;       *                                                              *
;       ****************************************************************



;       ALU CONFIGURATION

FLOAT:          DEF     CFLOAT,109X
DINT:           DEF     CDINT,109X
SINT:           DEF     CSINT,109X



;       EXPONENT ALU CONTROL & FUNCTIONS

;##     ATTENTION I:
;##             DUE TO THE FACT THAT IO IS A MULTIFUNCTION PIN THERE
;##             IS A RESTRICTION IN SPECIFYING THE FORMAT NAMES:
;##                     1)  SPF OR SPFNW
;##                     2)  HIGH
;##                     3)  PAR
;##                     4)  COMR
;##                     5)  LOW
;##
;##             THESE INSTRUCTION CANNOT BE USED TOGETHER WITH
;##             M. OR E.RAC'S SECOND PARAMETER SPECIFIED!
;##             (SEE ALSO ATTENTION AT "REGISTER ADDRESS CONTROL" ON PAGE 7)
;##
;##     ATTENTION II:
;##             BECAUSE THE REGISTERS SPECIFIED IN THE FOLLOWING INSTRUCTIONS
;##             -EXPONENT AS WELL AS MANTISSA- OVERLAY IN THE SAME FIELD,
;##             THEY MAY ONLY BE SPECIFIED ONCE!

E.SPF:          DEF     3X,SPF,SUB0
E.SPFNW:        DEF     3X,SPF,ALUOFF,SUB0A
E.HIGH:         DEF     3X,HIGH,SUB1
E.SSR:          DEF     3X,SSR,SUB2
E.SRS:          DEF     3X,SRS,SUB2
E.ADD:          DEF     3X,ADD,SUB2
E.PAS:          DEF     3X,PAS,SUB2
E.COMS:         DEF     3X,COMS,SUB2
E.PAR:          DEF     3X,PAR,SUB1
E.COMR:         DEF     3X,COMR,SUB1
E.LOW:          DEF     3X,LOW,SUB1
E.CRAS:         DEF     3X,CRAS,SUB2
E.XNRS:         DEF     3X,XNRS,SUB2
```

```
DEFINITION FILE FOR FLOATING POINT ALU ( FIRST HALF )

E.OR:           DEF     3X,OR,SUB2
E.ALUOFF:       DEF     7X,ALUOFF,YOFF,4X,XDOFF,97X
E.READ:         DEF     13X,2YB#10,8Y%X,89X
E.WRITE:        DEF     13X,2YB#01,49X,SUB8



;       MANTISSA ALU CONTROL & FUNCTIONS

M.SPF:          DEF     23X,SPF,SUB4
M.SPFNW:        DEF     23X,SPF,ALUOFF,ALUOFF,SUB4A
M.HIGH:         DEF     23X,HIGH,SUB5
M.SSR:          DEF     23X,SSR,SUB6
M.SRS:          DEF     23X,SRS,SUB6
M.ADD:          DEF     23X,ADD,SUB6
M.PAS:          DEF     23X,PAS,SUB6
M.COMS:         DEF     23X,COMS,SUB6
M.PAR:          DEF     23X,PAR,SUB5
M.COMR:         DEF     23X,COMR,SUB5
M.LOW:          DEF     23X,LOW,SUB5
M.CRAS:         DEF     23X,CRAS,SUB6
M.XNRS:         DEF     23X,XNRS,SUB6
M.XOR:          DEF     23X,XOR,SUB6
M.AND:          DEF     23X,AND,SUB6
M.NOR:          DEF     23X,NOR,SUB6
M.NAND:         DEF     23X,NAND,SUB6
M.OR:           DEF     23X,OR,SUB6
M.ALUOFF:       DEF     27X,ALUOFF,ALUOFF,2X,YOFF,YOFF,YOFF,4X,XDOFF,72X
M.READ:         DEF     38X,2YB#10,8Y%X,16Y%X,48X
M.WRITE:        DEF     38X,2YB#01,24X,SUB8



;       FP ALU ( =COMBINED E. & M.ALU ) MNEMONICS

;##     ATTENTION:
;##             WHEN READING FROM THE 2903-YBUS INTO THE REGISTERFILE
;##             DESTINATION MODIFIER "YBOFF" SHOULD BE USED IN ORDER
;##             TO AVOID YBUS CONTENTION.

FP.MODEO:       DEF     CFLOAT,10X,2YX,23X,2YX,25X,XDOFF,B#1,32X,XCEIVOFF
FP.READ:        DEF     13X,XDOFF,23X,XDOFF,25X,B#101,44X
FP.WRITE:       DEF     13X,2YX,23X,2YX,24X,1Y!B#01!,B#01,1Y!*B#11!*,44X
FP.XDOFF:       DEF     65X,XDOFF,45X
FP.XCEIVOFF:    DEF     100X,XCEIVOFF



;       REGISTER ADDRESS CONTROL

;##     ATTENTION:
;##             BECAUSE IO IS SET TO 0 IF SPF IS SELECTED AND IS ALSO USED
;##             OF THE  REGISTER ADDRESS CONTROL FIELD, ".SPFRAC" SHOULD BE
;##             USED INSTEAD OF ".RAC" WHEN A "B"-REGISTER IS SPECIFIED.
```

```
E.RAC:          DEF     86X,SUB9,19X
M.RAC:          DEF     93X,SUB9,12X
E.SPFRAC:       DEF     86X,SUB10,19X
M.SPFRAC:       DEF     93X,SUB10,12X



;       COPY & SWAP CONTROL

;##     ATTENTION:
;##             NOT ALL POSSIBLE COMBINATIONS ARE CODED.
;##             USE SUB11,SUB12 AND SUB13 TO CONSTRUCT OTHERS.

SWAP.OFF:       DEF     100X,XCEIVOFF
COPY.OFF:       DEF     100X,XCEIVOFF
COPY.3.1:       DEF     SUB11,YON,SUB12,YON,YOFF,YON,SUB13,H#8FF
SWAP.2.0:       DEF     SUB11,YON,SUB12,YOFF,YON,YOFF,SUB13,H#FBE
;
        END

TOTAL PHASE 1 ERRORS =   0
```

```
;       ************************************************************
;       *                                                          *
;       *       DEFINITION FILE FOR PART 2 OF THE CONTROL-         *
;       *       VECTOR FOR THE FLOATING POINT ALU                  *
;       *                                                          *
;       *                                                          *
;       *       JULY 1981, BY           P. RAEDTS                  *
;       *                               J. JACOBS                  *
;       *                                                          *
;       ************************************************************

        WORD    84

;       FIELD SPECIFICATION OF THE FIRST 84 BITS OF THE MICRO-
;       WORD:

;               BIT    0- 16    STATUS & SHIFT CONTROL FOR EXPONENT ALU
;               BIT   17- 41    STATUS & SHIFT CONTROL FOR MANTISSA ALU
;               BIT   42- 70    SEQUENCE CONTROL
;               BIT   71- 83    PREDICTION & TIMING CONTROL
```

```
;       ****************************************************************
;       *                                                              *
;       *       BASIC EQUATES                                          *
;       *                                                              *
;       ****************************************************************



;       2904 SHIFT CONTROL

SDL:            EQU     5H#0;           ;SHIFT DOWN LOW
SDH:            EQU     5H#1;           ;SHIFT DOWN HIGH
SUL:            EQU     5H#12;          ;SHIFT UP LOW
SUH:            EQU     5H#13;          ;SHIFT UP HIGH
SDDH:           EQU     5H#3;           ;SHIFT DOUBLE DOWN HIGH
SDDL:           EQU     5H#6;           ;SHIFT DOUBLE DOWN LOW
SDUL:           EQU     5H#16;          ;SHIFT DOUBLE UP LOW
SDUH:           EQU     5H#17;          ;SHIFT DOUBLE UP HIGH
RSD:            EQU     5H#A;           ;ROTATE SINGLE DOWN
RSU:            EQU     5H#1A;          ;ROTATE SINGLE UP
SSXO:           EQU     5H#E;           ;SHIFT DOWN SIGN EXCLUSIVE OR OVERFLOW
RDD:            EQU     5H#F;           ;ROTATE DOUBLE DOWN
RDU:            EQU     5H#1F;          ;ROTATE DOUBLE UP
SDMS:           EQU     5H#5;           ;SHIFT DOWN SIGN OF MSR
SMS:            EQU     5H#2;           ;SHIFT DOWN BOTH:
/                               }               RAM (IN: 0, OUT: CARRY MSR)
/                               }             Q (IN: SIGN OF MCS, OUT: LOST)
SDDCL:          EQU     5H#7;           ;SHIFT DOUBLE DOWN INTO CARRY LOW
SDUCL:          EQU     5N#14;          ;SHIFT DOUBLE UP INTO CARRY LOW



;       2904 CONDITION CODE OUTPUT

OVR:            EQU     Q#3             ;OVERFLOW
C:              EQU     Q#5             ;CARRY
N:              EQU     Q#7             ;NEGATIVE (SIGN)
Z:              EQU     Q#2             ;ZERO
LEQ:            EQU     Q#0             ;LESS OR EQUAL FOR 2'S COMPL. NUMBERS
LSS:            EQU     Q#1             ;LESS          FOR 2'S COMPL. NUMBERS
C.OR.Z:         EQU     Q#6             ;CARRY OR ZERO
NC.OR.Z:        EQU     Q#4             ;NOT CARRY OR ZERO ( LESS OR EQUAL FOR
/                                       }                   UNSIGNED NUMBERS )



;       2904 CARRY IN CONTROL
```

```
;       ****************************************************************
;       *                                                              *
;       *       MACHINE DEPENDENT EQUATES                              *
;       *                                                              *
;       ****************************************************************



;       SHIFT CONTROL

SHIFTENB:       EQU     B#0             ;SE/
SHIFTDIS:       EQU     SHIFTENB*
TRANSPARANT:    EQU     H#A             ;INSTRUCTION VECTOR FOR 2904-C


;       STATUS REGISTER CONTROL

;##     ATTENTION:
;##             BECAUSE I5 I4 ALSO CONTROL THE Y OUTPUT OF THE 2904,
;##             CARE SHOULD BE TAKEN IN CHOOSING THE RIGHT BITSTRING
;##             FOR A PARTICULAR FUNCTION (OR MNEMONIC).
;##             THESE SIDE EFFECTS ARE DENOTED BY:  [ ..==>.. ]

SET:            EQU     Q#01            ;SET MSR OR USR [U==>Y]
RESET:          EQU     Q#11            ;CLEAR MSR OR USR [U==>Y]
INVERT:         EQU     Q#21            ;INVERT ONLY MSR [U==>Y]
SWAP:           EQU     Q#10            ;SWAP MSR AND USR [U==>Y]
MOV.M:          EQU     Q#10            ;MOVE MSR TO USR [U==>Y]
MOV.Y:          EQU     Q#00            ;MOVE YBUS 2904 TO MSR (LOAD FROM YBUS)
MOV U:          EQU     Q#00            ;MOV USR TO MSR [U==>Y]
PAS.I:          EQU     Q#42            ;PASS IMM. INPUT TO YBUS
PAS.M:          EQU     Q#04            ;PASS MSR TO YBUS (USE LOAD)
PAS.U:          EQU     Q#00            ;PASS USR TO YBUS (USE MOV.M)
/                                       ;AND DON'T SPECIFY SECOND PARAMETER
/                                       ;IN THE ".REG"-FORMAT
PAS.IL:         EQU     Q#77            ;PASS IMM. INPUT TO YBUS
/                                       ;AND LOAD INTO MSR.
LOAD:           EQU     Q#04            ;LOAD FROM IMM. INPUT INTO MSR OR USR,
/                                       ;[M==>Y]
LOAD.CI:        EQU     Q#42            ;LOAD MSR OR USR WITH CARRY
/                                       ;INVERT [U==>Y]
LOAD.OR:        EQU     Q#30            ;LOAD USR WITH OVERFLOW RETAIN [U==>Y]
LOAD.SO:        EQU     Q#20            ;LOAD FOR SHIFT THROUGH OVERFLOW,
/                                       ;[U==>Y]
/                                       ;OPERATION INTO MSR
```

```
U:              EQU     B#01            ;SELECT CT FROM USR
M:              EQU     B#10            ;SELECT CT FROM MSR
T:              EQU     B#0             ;SELECT TRUE OUTPUT (NON INVERT)
F:              EQU     T*
CTON:           EQU     B#0             ;PUT CT OUTPUT IN 3RD-STATE
CTOFF:          EQU     CTON*



;       STATUS REGISTER CONTROL

YON:            EQU     B#0             ;OEY/
YOFF:           EQU     YON*
CEON:           EQU     B#0             ;CE/
CEOFF:          EQU     CEON*
UM:             EQU     B#00            ;SELECT BOTH USR AND MSR
Y:              EQU     B#10            ;SELECT YBUS



;       CONDITION CODE MULTIPLEXER

CC1:            EQU     B#00110         ;THE FIRST 3 BITS CONTROL THE
CC2:            EQU     B#01010         ;ABC-FIELD OFF AM2922.
CC3:            EQU     B#01110         ;THE LAST 2 BITS CONTROL THE
CC4:            EQU     B#10010         ;SELECTION OF ONE OF THE AM2922'S
CC5:            EQU     B#10110         ;FOR A EXACT DEFINITION OF CC<I> OR
C00:            EQU     B#00001         ;CO<J> SEE REPORT.
C01:            EQU     B#00101
C02:            EQU     B#01001
C04:            EQU     B#10001
C06:            EQU     B#11001
NONCOND:        EQU     B#01101         ;USED IN NON-CONDITIONAL STATEMENTS



;       SEQUENCE CONTROL

UNCOND:         EQU     B#01            ;CONTROLS THE CCEN/
LDCTPAR:        EQU     B#10            ;AND THE RLD/ INPUT) BOTH ON AM2910
M.WIDTH:        EQU     24-1            ;WIDTH OF MANTISSA, 0 COUNTS!
M.WMIN2:        EQU     M.WIDTH-2



;       PREDICTION AND TIMING CONTROL

OFF:            EQU     B#1             ;PREDICTION OFF
```

```
;       ****************************************************************
;       *                                                              *
;       *       SUBFORMAT DEFINITIONS                                  *
;       *                                                              *
;       ****************************************************************


;       EXPONENT & MANTISSA SHIFT CONTROL

SUB0:   SUB     SHIFTENB,78X
SUB1:   SUB     TRANSPARANT,SHIFTENB,SHIFTENB,56X


;       EXPONENT & MANTISSA STATUS CONTROL

SUB2:   SUB     CTON,3VQ#2,2VB#10,1V*B#0*,2X
SUB3:   SUB     1X,6VX,2VB#11
SUB4:   SUB     13X,CTOFF,CTOFF,32X
SUB5:   SUB     CEOFF,CEOFF,CTOFF,10X,CTOFF,CTOFF,32X
```

```
;       ************************************************************
;       *                                                          *
;       *       FORMAT DEFINITIONS                                 *
;       *                                                          *
;       ************************************************************



;       EXPONENT SHIFT CONTROL

E.SDL:          DEF     SDL,SUB0
E.SDH:          DEF     SDH,SUB0
E.SUL:          DEF     SUL,SUB0
E.SUH:          DEF     SUH,SUB0
E.SDDH:         DEF     SDDH,SUB0
E.SDDL:         DEF     SDDL,SUB0
E.SDUL:         DEF     SDUL,SUB0
E.SDUH:         DEF     SDUH,SUB0
E.RSD:          DEF     RSD,SUB0
E.RSU:          DEF     RSU,SUB0
E.SSXO:         DEF     SSXO,SUB0
E.RDD:          DEF     RDD,SUB0
E.RDU:          DEF     RDU,SUB0
E.SDMS:         DEF     SDMS,SUB0
E.SMS:          DEF     SMS,SUB0
E.SDDCL:        DEF     SDDCL,SUB0
E.SDUCL:        DEF     SDUCL,SUB0
E.SHIFTOFF:     DEF     5X,SHIFTDIS,78X



;       MANTISSA SHIFT CONTROL

M.SDL:          DEF     17X,SDL,SUB1
M.SDH:          DEF     17X,SDH,SUB1
M.SUL:          DEF     17X,SUL,SUB1
M.SUH:          DEF     17X,SUH,SUB1
M.SDDH:         DEF     17X,SDDH,SUB1
M.SDDL:         DEF     17X,SDDL,SUB1
M.SDUL:         DEF     17X,SDUL,SUB1
M.SDUH:         DEF     17X,SDUH,SUB1
M.RSD:          DEF     17X,RSD,SUB1
M.RSU:          DEF     17X,RSU,SUB1
M.SSXO:         DEF     17X,SSXO,SUB1
M.RDD:          DEF     17X,RDD,SUB1
M.RDU:          DEF     17X,RDU,SUB1
M.SDMS:         DEF     17X,SDMS,SUB1
M.SMS:          DEF     17X,SMS,SUB1
M.SDDCL:        DEF     17X,SDDCL,SUB1
M.SDUCL:        DEF     17X,SDUCL,SUB1
M.SHIFTOFF:     DEF     26X,SHIFTDIS,SHIFTDIS,56X
```

```
;       EXPONENT STATUS CONTROL

;##     ATTENTION I:
;##             DUE TO THE FACT THAT I5-I0 ARE MULTIFUNCTION BITS THERE
;##             IS A RESTRICTION IN OVERLAYING THE FIELDS:
;##                     1)  CONDITION CODE OUTPUT
;##                     2)  STATUS REGISTER CONTROL
;##
;##             THE FOLLOWING PAIRS ARE ALLOWED:
;##                     1)  E.CT , M.REG
;##                     2)  E.REG, M.REG
;##                     3)  E.REG, M.CT
;##
;##             ONE COULD BYPASS THIS PROBLEM BY DEFINING A NEW
;##             FORMAT NAME WITH THE BITS I5-I0 SET ACCORDING TO
;##             THE WANTED COMBINATIONS OF FUNCTIONS. SEE FOR EXAMPLE:
;##                     E.REGCT  <E.CT PARAMETERS>,<E.REG 2ND PARAMETER>
;##                     M.REGCT  <M.CT PARAMETERS>,<M.REG 2ND PARAMETER>

;##     ATTENTION II:
;##             THE THREE SOURCES FOR A CONDITION CODE ARE TIED
;##             TOGETHER VIA 3-STATE OUTPUTS. SO USE ONLY ONE OF:
;##                     1)  AM2904-A,
;##                     2)  AM2904-B,
;##                     3)  AM2922,     AT A TIME.

E.CT:           DEF     6X,SUB2,13X,CTOFF,21X,CTOFF,CTOFF,32X
E.REG:          DEF     6X,SUB3,69X
E.REGCT:        DEF     6X,CTON,3V@#2,2V@#10,1V*@#0*,2V@#10,13X,CTOFF,8X,SUB4
E.CIN:          DEF     15X,2V@#00,67X
E.REGOFF:       DEF     13X,CEOFF,CEOFF,69X


;       MANTISSA STATUS CONTROL

M.CT:           DEF     28X,SUB2,CEOFF,CEOFF,CTOFF,10X,CTOFF,CTOFF,32X
M.REG:          DEF     28X,SUB3,CEOFF,CEOFF,CTOFF,44X
M.REGCT:        DEF     6X,CTOFF,21X,CTON,3V@#2,2V@#10,1V*@#0*,2V@#10,SUB5
M.CIN:          DEF     40X,2V@#00,42X
M.REGOFF:       DEF     35X,CEOFF,CEOFF,47X


;       STATUS TRANSPORT CONTROL
```

```
;       CONDITION CODE MUX

FP.CT:          DEF     6X,CTOFF,21X,CTOFF,18X,5VX,1VB#1,31X


;       FP CONTROL ( =COMBINED E. & M. STATUS CONTROL )

FP.MODEO:       DEF     5X,SHIFTDIS,20X,SHIFTDIS,SHIFTDIS,14X,B#11111,37X
FP.REGOFF:      DEF     13X,CEOFF,CEOFF,20X,CEOFF,CEOFF,CEOFF,CEOFF,45X
FP.CTOFF:       DEF     6X,CTOFF,21X,CTOFF,18X,NONCOND,32X
/                                               ;ALL CT OUTPUTS OFF
FP.SHIFTOFF:    DEF     5X,SHIFTDIS,20X,SHIFTDIS,SHIFTDIS,56X


;       SEQUENCE CONTROL

JZ:     DEF     53X,H#0,2VB#11,25X      ;JUMP ZERO (RESET)
CJS:    DEF     53X,H#1,2VB#11,25X      ;CONDITIONAL JUMP SUBROUTINE PIPELINE
JMS:    DEF     53X,H#1,B#1,1V:B#1,25X  ;UNCONDITIONAL CJS
JMAP:   DEF     53X,H#2,2VB#11,25X      ;JUMP MAP
CJP:    DEF     53X,H#3,2VB#11,25X      ;CONDITIONAL JUMP PIPELINE
JMP:    DEF     53X,H#3,B#1,1V:B#1,25X  ;UNCONDITIONAL CJP
PUSH:   DEF     53X,H#4,2VB#11,25X      ;PUSH/CONDITIONAL LOAD COUNTER
JSRP:   DEF     53X,H#5,2VB#11,25X      ;CONDITIONAL JUMP SUBROUTINE R OR
                                        ;PIPELINE
CJV:    DEF     53X,H#6,2VB#11,25X      ;CONDITIONAL JUMP VECTOR
JRP:    DEF     53X,H#7,2VB#11,25X      ;CONDITIONAL JUMP R OR PIPELINE
RFCT:   DEF     53X,H#8,2VB#11,25X      ;REPEAT LOOP, COUNTER NOT ZERO
RPCT:   DEF     53X,H#9,2VB#11,25X      ;REPEAT PIPELINE, COUNTER NOT ZERO
CRTN:   DEF     53X,H#A,2VB#11,25X      ;CONDITIONAL RETURN
RTN:    DEF     53X,H#A,B#1,1V:B#1,25X  ;UNCONDITIONAL CRTN
CJPP:   DEF     53X,H#B,2VB#11,25X      ;CONDITIONAL JUMP PIPELINE AND POP
LDCT:   DEF     53X,H#C,2VB#11,25X      ;LOAD COUNTER AND CONTINUE
LOOP:   DEF     53X,H#D,2VB#11,25X      ;TEST END LOOP
CONT:   DEF     53X,H#E,2VB#11,25X      ;CONTINUE
TWB:    DEF     53X,H#F,2VB#11,25X      ;THREE-WAY BRANCH
                                        ;THREE-WAY DEFINITION
                                        ;PASS TEST - CONTINUE PC AND POP
                                        ;FAIL TEST - REPEAT LOOP IF COUNTER NOT
                                        ;            ZERO
                                        ;FAIL TEST - JUMP PIPELINE AND POP IF
                                        ;            COUNTER ZERO
COUNT:  DEF     59X,12V%,13X            ;DEFINE COUNT
GOTO:   DEF     59X,12V:X,13X           ;DEFINE ADDRESS


;       PREDICTION AND TIMING CONTROL

PREDICT:        DEF     71X,1VB#1,12X
```

```
        END

 TOTAL PHASE 1 ERRORS =    0
```

```
                ;               FMUL ON ADDRESS MACHINE
                ;               OPERANDS IN ADDRESS REGISTER
                ;               CALL AS SUBROUTINE

0000            ORG     H#0

        FMUL:
0000 /          FP.MODE0 E,M & E.ADD ,,,R3 & E.RAC AR,AR,PL
     /          & M.PAS QPT & M.RAC ,AR
     /                  ;ADD EXPONENTS AND LOAD Q
0001            FP.MODE0 ,M & E.ALUOFF & M.PAR ,,,R3
     /          & M.RAC AR,,PL
     /                  ;TEST MULTIPLICAND (WRITE TO DUMMY DESTINATION)
0002            FP.MODE0 & E.ALUOFF & M.READ PL,H#00,H#0000
     /          & M.PAR ,,,R3 & M.RAC ,,PL
     /                  ;CLEAR PARTIAL PRODUCT REGISTER R3
0003            FP.MODE0 ,M & E.ALUOFF & M.SPF TCM,,R3,R3
     /          & M.SPFRAC AR,PL,PL
     /                  ;2'S COMPLEMENT MULTIPLY ( 1 TIME )
0004            FP.MODE0 ,M & E.ALUOFF & M.SPF TCM,,R3,R3
     /          & M.SPFRAC AR,PL,PL
     /                  ;2'S COMPLEMENT MULTIPLY ( 22 TIMES )
0005            FP.MODE0 ,M & E.ALUOFF & M.SPF TCMC,,R3,R3
     /          & M.SPFRAC AR,PL,PL
     /                  ;2'S COMPLEMENT MULTIPLY CORRECTION ( 1 TIME )
0006            FP.MODE0 E,M & E.PAS ,,R3 & E.RAC ,PL,AR
     /          & M.PAS ,, & M.RAC ,PL,AR
     /                  ;COPY RESULT INTO C-REG OF ADDRESS REGISTER
0007            FP.MODE0 ,M & E.ALUOFF & M.SPFNW DLN,,R3,
     /          & M.SPFRAC ,PL,
     /                  ;TEST FOR ALREADY NORMALIZED RESULT AND
     /                  ;DON'T WRITE!
0008            FP.MODE0 ,M & E.READ PL,H#FF & E.ADD ,,R3,R3
     /          & E.RAC ,PL,PL & M.SPF DLN,,, & M.SPFRAC ,PL,PL
     /                  ;UPDATE SHIFT COUNT
0009            FP.MODE0 E,M & E.PAS ,,R3 & E.RAC ,PL,AR
     /          & M.PAS ,, & M.RAC ,PL,AR
     /                  ;COPY RESULT IN C-REG OF ADDRESS REGISTER
        FZERO:
000A /          FP.MODE0 & E.READ PL,H#80 & E.PAR
     /          & E.RAC ,,AR & M.READ PL,H#00,H#0000 & M.PAR
     /          & M.RAC ,,AR
     /                  ;LOAD IMMEDIATE A "CLEAN ZERO"
        EXPOVERF:
000B /          FP.MODE0 ,M & E.ALUOFF & M.SPFNW DLN,,R3,
     /          & M.SPFRAC ,PL,
     /                  ;EXPONENT OVERFLOW OCCURED! TRY TO CORRECT IT
     /                  ;BY NORMALIZING
000C            FP.MODE0 ,M & E.READ PL,H#FF & E.ADD ,,R3,R3
     /          & E.RAC ,PL,PL & M.SPF DLN,,, & M.SPFRAC ,PL,PL
     /                  ;NORMALIZE AND UPDATE SHIFT COUNT
000D            FP.MODE0 E & E.PAS ,,R3 & E.RAC ,PL,AR
     /          & M.ALUOFF
        SETOVR:
000E /          FP.MODE0 E & E.PAS ,,R3 & E.RAC ,PL,AR
     /          & M.ALUOFF
```

```
                END
```

FLOATING POINT MULTIPLY (FIRST HALF)

```
0000 101001100111111X XXXXXXX010000000 000110111XXXXXXXX XXXXXXXXXXXXXXXX
     X111XXXXXXXXXXX 000011000001 0XX0 00XX111111111111
0001 101XXXX11XXXX11X XXXXXXX011000110 001111111XXXXXXXX XXXXXXXXXXXXXXXX
     X111XXXXXXXXXXX 000011XXXXXXX001 XX10111111111111
0002 101XXXX11XXXX11X XXXXXXX011000110 001111100000000000 0000000000000000
     X111XXXXXXXXXXX 000011XXXXXXXX1 XX10111111111111
0003 101XXXX11XXXX11X XXXXXXX000000000 000010111XXXXXXXX XXXXXXXXXXXXXXXX
     X111XXXXX000011 000011XXXXXXX000 1010111111111111
0004 101XXXX11XXXX11X XXXXXXX000000000 000010111XXXXXXXX XXXXXXXXXXXXXXXX
     X111XXXXX000011 000011XXXXXXX000 1010111111111111
0005 101XXXX11XXXX11X XXXXXXX000000000 000110111XXXXXXXX XXXXXXXXXXXXXXXX
     X111XXXXX000011 000011XXXXXXX000 1010111111111111
0006 101010000111111X XXXXXXX010000110 001111111XXXXXXXX XXXXXXXXXXXXXXXX
     X111XXXXX000011 XXXXXXXX01000XX0 1000111111111111
0007 101XXXX11XXXX11X XXXXXXX000011000 001010111XXXXXXXX XXXXXXXXXXXXXXXX
     X111XXXXX000011 XXXXXXXXXXXXXXX0 10XX111111111111
0008 101001100111101 111111100000000 001010111XXXXXXXX XXXXXXXXXXXXXXXX
     X111XXXXX000011 000011XX01010XX0 1010111111111111
0009 101010000111111X XXXXXXX010000110 001111111XXXXXXXX XXXXXXXXXXXXXXXX
     X111XXXXX000011 XXXXXXXX01000XX0 1000111111111111
000A 101011000111101 0000000011000110 001111100000000000 0000000000000000
     X111XXXXXXXXXXX XXXXXXXX1XX00XX1 XX00111111111111
000B 101XXXX11XXXX11X XXXXXXX000011000 001010111XXXXXXXX XXXXXXXXXXXXXXXX
     X111XXXXX000011 XXXXXXXXXXXXXXX0 10XX111111111111
000C 101001100111101 111111100000000 001010111XXXXXXXX XXXXXXXXXXXXXXXX
     X111XXXXX000011 000011XX01010XX0 1010111111111111
000D 101010C00111111X XXXXXXXXXXX11XX1 11XXXX11XXXXXXXX XXXXXXXXXXXXXXXX
     X111XXXXX000011 XXXXXXXX01000XXX XXXX1111111111111
000E 101010000111111X XXXXXXXXXXX11XX1 11XXXX11XXXXXXXX XXXXXXXXXXXXXXXX
     X111XXXXX000011 XXXXXXXX01000XXX XXXX1111111111111

TOTAL PHASE 2 ERRORS =  0
```

```
        ;       FMUL ON ADDRESS MACHINE
        ;       OPERANDS IN ADDRESS REGISTER
        ;       CALL AS SUBROUTINE

 0000           ORG     H#0

      FMUL:
 0000 /         FP.MODEO & E.REG LOAD,M & E.CIN ZERO
      /         & M.CT Z,I & M.CIN ZERO & CJP
      /         & GOTO FZERO & PREDICT F & TIMERS 8,D#13,8
      /                 ;JUMP ON MULTIPLIER EQUALS TO ZERO
 0001           FP.MODEO & E.REG PAS.M & M.REG LOAD,M
      /         & M.CIN ZERO & FP.CT CC1 & CJP
      /         & GOTO FZERO & PREDICT F & TIMERS 4,9,7
      /                 ;JUMP ON EXPONENT UNDERFLOW
 0002           FP.MODEO & E.REG PAS.M & M.CT Z,M
      /         & M.CIN ZERO & CJP & GOTO FZERO
      /         & PREDICT F & TIMERS 3,8,7
      /                 ;JUMP ON MULTIPLICAND ZERO
 0003           E SHIFTOFF & E.REGOFF & M.SSXO
      /         & M.REGOFF & FP.CTOFF & LDCT
      /         & COUNT M.WMIN2 & ST.OFF & PREDICT OFF
      /         & TIMERS ,,8
      /                 ;LOAD COUNTER AND DO FIRST MULTIPLY OPERATION
 0004           E.SHIFTOFF & E.REGOFF & M.SSXO
      /         & M.REGOFF & FP.CTOFF & RPCT
      /         & GOTO $ & ST.OFF & PREDICT OFF
      /         & TIMERS ,,8
      /                 ;2'S COMPLEMENT MULTIPLY ( 22 TIMES )
 0005           E.SHIFTOFF & E.REGOFF & M.SSXO
      /         & M.REGOFF & M.CIN CX & FP.CT CC2
      /         & ST.OFF & CJP & GOTO EXPOVERF
      /         & PREDICT F & TIMERS 4,9,7
      /                 ;2'S COMPLEMENT MULTIPLY CORRECTION
      /                 ;TEST SUM OF EXPONENTS.
 0006           FP.MODEO & E.REGOFF & E.CIN ZERO
      /         & M.REGOFF & M.CIN ZERO & FP.CTOFF
      /         & CONT & PREDICT OFF & TIMERS ,,7
      /                 ;COPY RESULT INTO C-REG OF ADDRESS REGISTER
 0007           E.SHIFTOFF & E.REGOFF & M.SDUL
      /         & M.CT C,I & M.CIN ZERO & M.REGOFF
      /         & ST.OFF & CRTN & PREDICT F
      /         & TIMERS 8,D#12,8
      /                 ;TEST FOR NORMALIZED RESULT
 0008           E.SHIFTOFF & E.REG PAS.IL,M & E.CIN ZERO
      /         & M SDUL & M.REG PAS.IL,M &  M.CIN ZERO
      /         & FP.CT CC3 & ST.OFF & CJP
      /         & GOTO $ & PREDICT T & TIMERS D#10,D#14,D#10
      /                 ;REPEAT SHIFTING UNTIL NORMALIZED OR EXPONENT
      /                 ;UNDERFLOW
 0009           FP.MODEO & E.REG PAS.M & E.CIN ZERO
      /         & M.REG LOAD,M & FP.CT CC1,F & CRTN
      /         & PREDICT T & TIMERS 4,6,6
```

```
FLOATING POINT MULTIPLY (SECOND PART)

      /       & RTN & PREDICT OFF & TIMERS ,,6
      /               ;RESULT IS A FLOATING ZERO!
      EXPOVERF:
 000B /       FP.MODE0 & E.REG RESET,M & M.REGOFF
      /       & M.CIN ZERO & M.CT C,I & CJP
      /       & GOTO SETOVR & PREDICT F & TIMERS 8,D#12,8
      /               ;CLEAR OVR BIT IN MSR, TEST IF ALREADY NORMALIZED.
 000C         E.SHIFTOFF & E.REG PAS.IL,M & E.CIN ZERO
      /       & M.SDUL & M.REG PAS.IL,M & M.CIN ZERO
      /       & FP.CT CC4 & ST.OFF & CJP
      /       & GOTO $ & PREDICT T & TIMERS D#10,D#14,D#10
      /               ;REPEAT UNTIL NORMALIZED OR EXPONENT UNDERFLOW
 000D         FP.MODE0 & E.REG RESET,M & E.CIN ZERO
      /       & M.REGOFF & FP.CT CC1 & CRTN
      /       & PREDICT F & TIMERS 4,9,4
      /               ;IF EXPONENT UNDERFLOW THEN CORRECTION WAS A SUCCESS
      SETOVR:
 000E /       FP.MODE0 & E.REG SET,M & E.CIN ZERO
      /       & M.REGOFF & FP.CTOFF & RTN
      /       & PREDICT OFF & TIMERS ,,5
      /               ;IF CORRECTION NOT POSSIBLE THEN EXIT WITH
      /               ;OVR BIT IN MSR SET

              END
```

FLOATING POINT MULTIPLY (SECOND PART)

```
 0000 XXXXX1X000100100 0XXXXXXXXX110010 111XX1110011111X XX11X00111100000
      0001010110001101 1000
 0001 XXXXX1100010011X XXXXXXXXXX111000 1001011100111110 0110100111100000
      0001010101001001 0111
 0002 XXXXX1X000100111X XXXXXXXXXX110010 101XX1110011111X XX11X00111100000
      0001010100111000 0111
 0003 XXXXX11XXXXXX11X X0111010100001XXX XXX11XXXXX111110 1101X11001100000
      0010101111111111 1000
 0004 XXXXX11XXXXXX11X X0111010100001XXX XXX11XXXXX111110 1101X10011100000
      0000100111111111 1000
 0005 XXXXX11XXXXXX11X X0111010100001XXX XXX11XXX10111110 1010100111100000
      0001011101001001 0111
 0006 XXXXX11XXXXXX110 0XXXXXXXXX111XXX XXX11XXX00111110 1101X111011XXXXX
      XXXXXXX111111111 0111
 0007 XXXXX1XXXXXX11X X1011010100000101 11111111001111IX XX11X101011XXXXX
      XXXXXXX110001100 1000
 0008 XXXXX1111111100 0101101010001111 1111011100111110 1110100111100000
      0001000010101110 1010
 0009 XXXXX11000100110 0XXXXXXXXX111000 1001011IXX111110 0110110101IXXXXX
      XXXXXXX001000110 0110
 000A XXXXX11000100100 0XXXXXXXXX111000 1001011100111110 1101X101011XXXXX
      XXXXXXX111111111 0110
 000B XXXXX1X001001101X XXXXXXXXXX110101 11111111001111IX XX11X00111100000
      0001110110001100 1000
 000C XXXXX1111111100 0101101010001111 1111011100111111 0010100111100000
      0001100010101110 1010
 000D XXXXX11001001100 0XXXXXXXXX111XXX XXX11XXXXX111110 0110110101IXXXXX
      XXXXXXX101001001 0100
 000E XXXXX11000001100 0XXXXXXXXX111XXX XXX11XXXXX111110 1101X101011XXXXX
      XXXXXXX111111111 0101

 TOTAL PHASE 2 ERRORS =   0
```

```
      ∇RESULT←AREG BINADD BREG;COUNT;POINTER
[10]  ⍝
[20]  ⍝ SIMULATION OF AM2903'S  F←R+S
[30]  ⍝
[40]  RESULT←(⍴AREG)⍴0
[50]  POINTER←COUNT←⍴AREG
[60]  CARRYIN←~Z←1
[70]  ⍝ ENTER LOOP
[80]  LOOP:RESULT[POINTER]←AREG[POINTER]BITADD BREG[POINTER]
[90]  Z←Z∧~RESULT[POINTER]
[100] CARRYIN←CARRYOUT
[110] →LOOP IF 1≠POINTER←COUNT←COUNT-1
[120] ⍝ LEAVE LOOP, AND PRESENT STATUS BITS.
[130] RESULT[POINTER]←AREG[POINTER]BITADD BREG[POINTER]
[140] OVR←CARRYOUT≠CARRYIN
[150] N←RESULT[1]
      ∇
```

```
      ∇RESULT←AREG BINSUB BREG;COUNT;POINTER
[10]  ⍝
[20]  ⍝ SIMULATION OF AM2903'S  F←R-S
[30]  ⍝
[40]  RESULT←(⍴AREG)⍴0
[50]  POINTER←COUNT←⍴AREG
[60]  CARRYIN←Z←1
[70]  ⍝ ENTER LOOP
[80]  LOOP:RESULT[POINTER]←AREG[POINTER]BITADD~BREG[POINTER]
[90]  Z←Z∧~RESULT[POINTER]
[100] CARRYIN←CARRYOUT
[110] →LOOP IF 1≠POINTER←COUNT←COUNT-1
[120] ⍝ LEAVE LOOP, AND PRESENT RESULT AND STATUS BITS.
[130] RESULT[POINTER]←AREG[POINTER]BITADD~BREG[POINTER]
[140] OVR←CARRYOUT≠CARRYIN
[150] N←RESULT[1]
      ∇
```

```
      ∇SUM←BITR BITADD BITS
[10]  ⍝
[20]  ⍝ FULL ADDER
[30]  ⍝
[40]  SUM←CARRYIN≠BITR≠BITS
[50]  CARRYOUT←(BITR∧BITS)∨CARRYIN∧BITR≠BITS
      ∇
```

```
      ∇WHERETO←LABEL IF BOOLEAN
[10]  ⍝
[20]  ⍝ IF FUNCTION
[30]  ⍝
[40]  WHERETO←LABEL×⍳BOOLEAN
      ∇
```

```
      ∇RESULT←AREG BINMUL QREG;BREG;SAVE;COUNT;SPFZ
[10]  ⍝
[20]  ⍝ SIMULATION OF AM2903'S SIGNED MULTIPLICATION
[30]  ⍝
[40]  BREG←(⍴AREG)⍴0
[50]  COUNT←¯1+⍴AREG
[60]  ⍝ ENTER LOOP
[70]  LOOP:SPFZ←¯1↑QREG
[80]  SAVE←BREG BINADD(⍴AREG)⍴0
[90]  →SKIPADD IF SPFZ=0
[100] SAVE←BREG BINADD AREG
[110] SKIPADD:SAVE←¯1↓(N≠OVR),SAVE,QREG
[120] BREG←(⍴AREG)↑SAVE
[130] QREG←(-⍴AREG)↑SAVE
[140] →LOOP IF 0≠COUNT←COUNT-1
[150] ⍝ LEAVE LOOP AND DO LAST CYCLE
[160] SPFZ←¯1↑QREG
[170] SAVE←BREG BINADD(⍴AREG)⍴0
[180] →SKIPSUB IF SPFZ=0
[190] SAVE←BREG BINSUB AREG
[200] SKIPSUB:RESULT←¯1↓(N≠OVR),SAVE,QREG
      ∇
```

```
      ∇RESULT←DIVIDEND BINDIV DIVISOR;QREG;AREG;BREG;COUNT;SAVE;SPFZ
[10] ⍝
[20] ⍝  SIMULATION OF AM2903'S SIGNED DIVISION
[30] ⍝
[40] COUNT←¯2+⍴DIVISOR
[50] QREG←(-⍴DIVISOR)↑DIVIDEND
[60] BREG←(⍴DIVISOR)↑DIVIDEND
[70] AREG←DIVISOR
[80] ⍝ DO FIRST CYCLE
[90] FIRSTCYCLE:SAVE←BREG BINADD(⍴BREG)⍴0
[100] SPFZ←AREG[1]=SAVE[1]
[110] SAVE←1↓SAVE,QREG,AREG[1]≠SAVE[1]
[120] BREG←(⍴DIVISOR)↑SAVE
[130] QREG←(-⍴DIVISOR)↑SAVE
[140] →LASTCYCLE IF COUNT=0
[150] ⍝ ENTER LOOP
[160] LOOP:SAVE←BREG BINADD AREG
[170] →SKIPSUB1 IF SPFZ=0
[180] SAVE←BREG BINSUB AREG
[190] SKIPSUB1:SPFZ←AREG[1]=SAVE[1]
[200] SAVE←1↓SAVE,QREG,AREG[1]=SAVE[1]
[210] BREG←(⍴DIVISOR)↑SAVE
[220] QREG←(-⍴DIVISOR)↑SAVE
[230] →LOOP IF 0≠COUNT←COUNT-1
[240] ⍝ LEAVE LOOP AND DO LAST CYCLE
[250] LASTCYCLE:SAVE←BREG BINADD AREG
[260] →SKIPSUB2 IF SPFZ=0
[270] SAVE←BREG BINSUB AREG
[280] SKIPSUB2:BREG←SAVE
[290] QREG←1↓QREG,1
[300] RESULT←BREG,QREG
      ∇
```

1 1 0 0 BINMUL 0 1 0 1

1 1 1 0 1 1 0 0

1 1 1 0 1 1 0 0 BINDIV 1 1 0 0

0 0 1 0 | 0 0 1 1

remainder quotiënt

1 1 1 0 1 1 0 1 BINDIV 1 1 0 0

0 0 1 0 | 0 0 1 1

remainder quotiënt

```
FLOATING POINT DIVIDE (FIRST PART)


        ;       FDIV ON ADDRESS MACHINE
        ;       OPERANDS IN ADDRESS REGISTER
        ;       CALL AS SUBROUTINE

0010            ORG     H#10

        FDIV:
0010 /          FP.MODE0 E,M & E.SRS ,,,R4 & E.RAC AR,AR,PL
     /          & M.PAS & M.RAC ,AR,PL
     /                  ;DIFFERENCE OF EXPONENTS IN R4
     /                  ;READ DIVISOR (M2) FOR TESTING
0011            FP.MODE0 ,M & E.READ PL,H#00 & E.PAR ,,,R3
     /          & E.RAC ,,PL & M.PAR ,,, & M.RAC AR,,PL
     /                  ;COPY DIVIDEND (M1) INTO R3
0012            FP.MODE0 ,M & E.ALUOFF & M.XOR QPT,R1,R1,
     /          & M.RAC PL,PL,
     /                  ;CLEAR QR
        AGAIN:
0013 /          FP.MODE0 ,M & E.ALUOFF & M.SPF SMTC,,R3,R1
     /          & M.SPFRAC ,PL,PL
     /                  ;2'S COMPLEMENT R3 ==> SIGN MAGNITUDE R1
0014            FP.MODE0 ,M & E.ALUOFF & M.SPF SMTC,,R4,R2
     /          & M.SPFRAC ,PL,PL
     /                  ;2'S COMPLEMENT R4 ==> SIGN MAGNITUDE R2
0015            FP.MODE0 ,M & E.ALUOFF & M.PAS LUR,,R1,R1
     /          & M.RAC ,PL,PL
     /                  ;SHIFT SIGN BIT OUT OF M1
0016            FP.MODE0 ,M & E.ALUOFF & M.PAS LUR,,R2,R2
     /          & M.RAC ,PL,PL
     /                  ;SHIFT SIGN BIT OUT OF M2
0017            FP.MODE0 ,M & E.ALUOFF & M.SRS ,R1,R2,
     /          & M.RAC PL,PL,
     /                  ;COMPARE M1 TO M2 (USE SUBTRACTION)
0018            FP.MODE0 E,M & E.PAS ,,R3,R3 & E.RAC ,PL,PL
     /          & M.PAS ADR,,, & M.RAC ,PL,PL
     /                  ;SCALE R3 1 POSITION (SHIFT RIGHT) AND UPDATE EXPONENT
        DIVID:
0019 /          FP.MODE0 ,M & E.ALUOFF & M.SPF TCFD,R4,R3,R3
     /          & M.SPFRAC ,PL,PL
     /                  ;FIRST DIVIDE OPERATION
001A            FP.MODE0 ,M & E.ALUOFF & M.SPF TCD,R4,R3,R3
     /          & M.SPFRAC ,PL,PL
     /                  ;2'S COMPLEMENT DIVIDE (22 TIMES)
001B            FP.MODE0 ,M & E.ALUOFF & M.SPF TCDC,R4,R3,R3
     /          & M.SPFRAC ,PL,PL
     /                  ;2'S COMPLEMENT DIVIDE CORRECTION
001C            FP.MODE0 ,M & E.ALUOFF & M.SPFNW SLN,,R3,R3
     /          & M.SPFRAC ,PL,PL
     /                  ;TEST IF NORMALIZATION IS NEEDED; R3 IS DUMMY SHIFTCOUNTER
001D            FP.MODE0 ,M & E.READ PL,H#FF & E.ADD ,,R3,R3
     /          & E.RAC ,PL,PL & M.SPF SLN,,, & M.SPFRAC ,PL,PL
     /                  ;PERFORM SINGLE LENGTH NORMALIZATION AS LONG AS NEEDED
        NORM:
001E /          FP.MODE0 E,M & E.ADD ,R4,R3 & E.RAC AR,PL,PL
     /          & M.PAS ,,, & M.RAC ,QR,PL
     /                  ;ADJUST EXPONENT AND COPY Q REGISTER INTO REGISTERFILE
001F            FP.MODE0 ,M & E.ALUOFF & M.PAS ,,R3,
```

```
        /       & M.RAC ,PL,AR
        /               ;COPY QUOTIENT INTO RESULTREGISTER,
        /               ;POINTED TO BY AR.
        FZERO:
 0020   /       FP.MODE0 & E.READ PL,H#80 & E.PAR ,,,
        /       & E.RAC ,,AR & M.READ PL,H#00,H#0000 & M.PAR ,,,
        /       & M.RAC ,,AR
        /               ;QUOTIENT IS FORCED TO FLOATING ZERO
        EXPOVERF:
 0021   /       FP.MODE0 & E.ALUOFF & M.ALUOFF
        /               ;ALU NO OPERATION
 0022           FP.MODE0 & E.ALUOFF & M.ALUOFF
        /               ;ALU NO OPERATION
        DVSZ:
 0023   /       FP.MODE0 & E.ALUOFF & M.ALUOFF
        /               ;ALU NO OPERATION
        SCALE:
 0024   /       FP.MODE0 E,M & E.PAR ,R3,,R3 & E.RAC PL,,PL
        /       & M.PAS ADR,,R3, & M.RAC ,PL,PL
        /               ;SCALE DIVIDEND, UPDATE EXPONENT
        CORROVER:
 0025   /       FP.MODE0 & E.ALUOFF & M.ALUOFF
        /               ;ALU NO OPERATION

                END
```

FLOATING POINT DIVIDE (FIRST PART)

```
0010 1010010001111111X XXXXXXX010000110 0011111XXXXXXXX XXXXXXXXXXXXXXX
     X111XXXXXXXXXXX 0001000000010XX0 0010111111111111
0011 1010110001111100 0000000011000110 0011111XXXXXXXX XXXXXXXXXXXXXXX
     X111XXXXXXXXXXX 000011XX1XX10001 XX10111111111111
0012 101XXXX11XXXX11X XXXXXXX101100000 0001101IXXXXXXX XXXXXXXXXXXXXXX
     X111000001000001 XXXXXXXXXXXXX100 10XX111111111111
0013 101XXXX11XXXX11X XXXXXXX000000110 0001011IXXXXXXX XXXXXXXXXXXXXXX
     X111XXXXXX000011 00000IXXXXXXXXX0 1010111111111111
0014 101XXXX11XXXX11X XXXXXXX000000110 0001011IXXXXXXX XXXXXXXXXXXXXXX
     X111XXXXXX000100 00001OXXXXXXXX0 1010111111111111
0015 101XXXX11XXXX11X XXXXXXX010000110 0010011IXXXXXXX XXXXXXXXXXXXXXX
     X111XXXXXX000001 00000IXXXXXXXX0 1010111111111111
0016 101XXXX11XXXX11X XXXXXXX010000110 0010011IXXXXXXX XXXXXXXXXXXXXXX
     X111XXXXXX000010 00001OXXXXXXXX0 1010111111111111
0017 101XXXX11XXXX11X XXXXXXX001000110 0011111IXXXXXXX XXXXXXXXXXXXXXX
     X111000001000010 XXXXXXXXXXXX100 10XX111111111111
0018 1010100001111111X XXXXXXX010000000 0000011IXXXXXXX XXXXXXXXXXXXXXX
     X111XXXXXX000011 00001IXX01010XX0 1010111111111111
0019 101XXXX11XXXX11X XXXXXXX000000000 0010101IXXXXXXX XXXXXXXXXXXXXXX
     X111000100000011 00001IXXXXXXXX0 1010111111111111
001A 101XXXX11XXXX11X XXXXXXX000000000 0011101IXXXXXXX XXXXXXXXXXXXXXX
     X111000100000011 00001IXXXXXXXX0 1010111111111111
0018 101XXXX11XXXX11X XXXXXXX000000000 0011101IXXXXXXX XXXXXXXXXXXXXXX
     X111000100000011 00001IXXXXXXXX0 1010111111111111
001C 101XXXX11XXXX11X XXXXXXX000011000 0010001IXXXXXXX XXXXXXXXXXXXXXX
     X111XXXXXX000011 00001IXXXXXXXX0 1010111111111111
001D 1010011001111101 1111111000000000 0010001IXXXXXXX XXXXXXXXXXXXXXX
     X111XXXXXX000011 00001IXX01010XX0 1010111111111111
001E 1010011001111111X XXXXXXX010000110 0011111IXXXXXXX XXXXXXXXXXXXXXX
     X111000100000011 XXXXXX0001010XX1 0110111111111111
001F 101XXXX11XXXX11X XXXXXXX010000110 0011111IXXXXXXX XXXXXXXXXXXXXXX
     X111XXXXXX000011 XXXXXXXXXXXXXX0 1000111111111111
0020 1010110001111101 0000000011000110 0011111000000000 0000000000000000
     X111XXXXXXXXXXX XXXXXXXX1XX00XX1 XX00111111111111
0021 101XXXX11XXXX11X XXXXXXXXXXX11XX1 11XXXX11XXXXXXX XXXXXXXXXXXXXXX
     X111XXXXXXXXXXX XXXXXXXXXXXXXXX XXXX111111111111
0022 101XXXX11XXXX11X XXXXXXXXXXX11XX1 11XXXX11XXXXXXX XXXXXXXXXXXXXXX
     X111XXXXXXXXXXX XXXXXXXXXXXXXXX XXXX111111111111
0023 101XXXX11XXXX11X XXXXXXXXXXX11XX1 11XXXX11XXXXXXX XXXXXXXXXXXXXXX
     X111XXXXXXXXXXX XXXXXXXXXXXXXXX XXXX111111111111
0024 1010110001111111X XXXXXXX010000000 0000011IXXXXXXX XXXXXXXXXXXXXXX
     X111000011000011 00001110IXX10XX0 1010111111111111
0025 101XXXX11XXXX11X XXXXXXXXXXX11XX1 11XXXX11XXXXXXX XXXXXXXXXXXXXXX
     X111XXXXXXXXXXX XXXXXXXXXXXXXXX XXXX111111111111
```

TOTAL PHASE 2 ERRORS = 0

```
        ;       FDIV ON ADDRESS MACHINE
        ;       OPERANDS IN ADDRESS REGISTER
        ;       CALL AS SUBROUTINE

 0010           ORG     H#10

      FDIV:
 0010 /         FP.MODE0 & E.REG LOAD,M & E.CIN ONE
      /         & M.REGCT Z,I,,M & M.CIN ZERO & CJP
      /         & GOTO DVSZ & PREDICT F
      /                 ;CHECK DIVISOR (M2) = 0, IF TRUE THEN SET OVERFLOW BIT!
 0011           FP.MODE0 & E.REG PAS.M & E.CIN ZERO
      /         & M.REG LOAD,M & M.CIN ZERO & FP.CTOFF
      /         & CONT & PREDICT OFF
      /                 ;
 0012           FP.MODE0 & E.REGOFF & M.REGOFF
      /         & M.CT Z,M & CJP & GOTO FZERO
      /         & PREDICT F
      /                 ;CHECK IF DIVIDEND (M1) = 0
      AGAIN:
 0013 /         FP.MODE0 & E.REGOFF & M.REG LOAD,U
      /         & M.CIN CX & FP.CTOFF & CONT
      /         & PREDICT OFF
      /                 ;
 0014           FP.MODE0 & E.REGOFF & M.REGCT OVR,U,,M
      /         & M.CIN CX & CJP & GOTO SCALE
      /         & PREDICT F
      /                 ;SCALE DIVIDEND BECAUSE !-1/2! < !DIVISOR! IS NOT TRUE!
 0015           E.SHIFTOFF & E.REGOFF & M.SUL
      /         & M.REGCT OVR,M,,M & M.CIN ZERO & ST.OFF
      /         & CJP & GOTO DIVID & PREDICT F
      /                 ;SHIFT SIGN OUT OF M1 AND PERFORM DIVISION
      /                 ;IF !M1! > !M2!
 0016           E.SHIFTOFF & E.REGOFF & M.SUL
      /         & M.REG LOAD,M & M.CIN ZERO & ST.OFF
      /         & FP.CTOFF & CONT & PREDICT OFF
      /                 ;SHIFT SIGN OUT OF M2
 0017           FP.MODE0 & E.REGOFF & M.REGCT N,I,,M
      /         & M.CIN ONE & CJP & GOTO DIVID
      /         & PREDICT F
      /                 ;COMPARE R1 TO R2 AND PROCEED WITH DIVISION
      /                 ;IF !M2! > !M1!
 0018           E.SHIFTOFF & E.REGOFF & E.CIN ONE
      /         & M.SSXO & M.REGOFF & M.CIN ZERO
      /         & ST.OFF & FP.CTOFF & CONT
      /         & PREDICT OFF
      /                 ;RESTORE DIVIDEND (AND SIGN!)
      DIVID:
 0019 /         E.SHIFTOFF & E.REGOFF & M.RDU
      /         & M.REGOFF & M.CIN ZERO & ST.OFF
      /         & FP.CTOFF & PUSH UNCOND & COUNT M.WMIN2
      /         & PREDICT OFF
      /                 ;FIRST DIVIDE OPERATION, SET UP LOOP
```

FLOATING POINT DIVIDE (SECOND PART)

```
001B          E.SHIFTOFF & E.REGOFF & M.RDU
      /       & M.REGOFF & M.CIN ZERO & ST.OFF
      /       & FP.CTOFF & CONT & PREDICT OFF
      /               ;CORRECT THE RESULT
001C          E.SHIFTOFF & E.REGOFF & M.SDUL
      /       & M.REGOFF & M.CT C,I & M.CIN ZERO
      /       & ST.OFF & CJP & GOTO NORM
      /       & PREDICT T
      /               ;CHECK QUOTIENT ALREADY NORMALIZED!
001D          E.SHIFTOFF & E.REGOFF & E.CIN ZERO
      /       & M.SDUL & M.REGOFF & M.CT OVR,I
      /       & M.CIN ZERO & ST.OFF & CJP
      /       & GOTO $ & PREDICT F
      /               ;SHIFT UNTIL NORMALIZED
      NORM:
001E  /       FP.MODEO & E.REGCT OVR,M,,U & E.CIN ONE
      /       & M.REG LOAD,M & M.CIN ZERO & CJP
      /       & GOTO EXPOVERF & PREDICT F
      /               ;ADJUST EXPONENT AND JUMP ON OVERFLOW IN MSR. THIS CON-
      /               ;DITION IS CREATED IN THE FIRST STEP OF THIS PROGRAM.
001F          FP.MODEO & E.REG MOV,U,M & M.REG LOAD,M
      /       & M.CIN ZERO & FP.CT CC1,F & CRTN
      /       & PREDICT T
      /               ;NO UNDERFLOW OCCURED ==> RESULT NOT TO SMALL!
      /               )                         LEAVE FDIV!
      FZERO:
0020  /       FP.MODEO & E.REG RESET,M & E.CIN ZERO
      /       & M.REG LOAD,M & M.CIN ZERO & FP.CTOFF
      /       & RTN & PREDICT OFF
      /               ;QUOTIENT IS TO SMALL! EXP < -128 OR QUOTIENT = 0
      EXPOVERF:
0021  /       FP.MODEO & E.REGOFF & E.CT OVR,U
      /       & M.REGOFF & CJP & GOTO CORROVR
      /       & PREDICT F
      /               ;IF POSSIBLE CORRECT SMALL OVERFLOW
0022          FP.MODEO & E.REG PAS,M & M.REGOFF
      /       & FP.CT CC1 & CJP & GOTO FZERO
      /       & PREDICT F
      /               ;IF EXPONENT UNDERFLOW THEN RESULT BECOMES FLOATING ZERO
      DVSZ:
0023  /       FP.MODEO & E.REG SET,M & M.REGOFF
      /       & FP.CTOFF & RTN & PREDICT OFF
      /               ;EXPONENT OVERFLOW!
      SCALE:
0024  /       E.SHIFTOFF & E.REGOFF & E.CIN ONE
      /       & M.SSXO & M.REGOFF & M.CIN ZERO
      /       & ST.OFF & FP.CTOFF & JMP
      /       & GOTO AGAIN & PREDICT OFF
      /               ;SCALE DIVIDEND, THEN TRY AGAIN
      CORROVR:
0025  /       FP.MODEO & E.REG RESET,M & M.REGOFF
      /       & FP.CTOFF & RTN & PREDICT OFF
      /               ;OVERFLOW CORRECTED

              END
```

```
 0010 XXXXX11000100100 1XXXXXXXXX110010 111101110011111X XX11X00111100000
      01000111XXXXXXXX XXXX
 0011 XXXXX11000100110 0XXXXXXXXX111000 1001011100111110 1101X111011XXXXX
      XXXXXXX1XXXXXXXX XXXX
 0012 XXXXX1XXXXXXX11X XXXXXXXXXX110010 1011111XX11111X XX11X00111100000
      01000001XXXXXXXX XXXX
 0013 XXXXX11XXXXXX11X XXXXXXXXXX111000 1000111110111110 1101X111011XXXXX
      XXXXXXX1XXXXXXXX XXXX
 0014 XXXXX11XXXXXX11X XXXXXXXXXX110011 0111011110111111X XX11X00111100000
      01001001XXXXXXXX XXXX
 0015 XXXXX11XXXXXX11X X100101010000011 1011011100111111X XX11X00111100000
      00110011XXXXXXXX XXXX
 0016 XXXXX11XXXXXX11X X100101010001000 1001011100111110 1101X111011XXXXX
      XXXXXXX1XXXXXXXX XXXX
 0017 XXXXX11XXXXXX11X XXXXXXXXXX110111 1111011101111111X XX11X00111100000
      00110011XXXXXXXX XXXX
 0018 XXXXX11XXXXXX110 1011101010001XXX XXX11XXX00111110 1101X111011XXXXX
      XXXXXXX1XXXXXXXX XXXX
 0019 XXXXX11XXXXXX11X X1111110100O1XXX XXX11XXX00111110 1101X01000100000
      00101011XXXXXXXX XXXX
 001A XXXXX11XXXXXX11X X1111110100O1XXX XXX11XXX00111110 1101X100011XXXXX
      XXXXXXX1XXXXXXXX XXXX
 001B XXXXX11XXXXXX11X X1111110100O1XXX XXX11XXX00111110 1101X111011XXXXX
      XXXXXXX1XXXXXXXX XXXX
 001C XXXXX1XXXXXXX11X X101101010000101 1111111100111111X XX11X00111100000
      00111100XXXXXXXX XXXX
 001D XXXXX1XXXXXXX110 0101101010000011 1111111100111111X XX11X00111100000
      00111011XXXXXXXX XXXX
 001E XXXXX10011101010 1XXXXXXXXX111000 1001011100111111X XX11X00111100000
      01000011XXXXXXXX XXXX
 001F XXXXX11000000010 XXXXXXXXXX111000 1001011100111110 0110110101 1XXXXX
      XXXXXXX0XXXXXXXX XXXX
 0020 XXXXX11001001100 0XXXXXXXXX111000 1001011100111110 1101X101011XXXXX
      XXXXXXX1XXXXXXXX XXXX
 0021 XXXXX10011011111X XXXXXXXXXX111XXX XXX11XXXXX11111X XX11X00111100000
      01001011XXXXXXXX XXXX
 0022 XXXXX11000100111X XXXXXXXXXX111XXX XXX11XXXXX111110 0110100111100000
      01000001XXXXXXXX XXXX
 0023 XXXXX11000001110X XXXXXXXXXX111XXX XXX11XXXXX111110 1101X101011XXXXX
      XXXXXXX1XXXXXXXX XXXX
 0024 XXXXX11XXXXXX110 1011101010001XXX XXX11XXX00111110 1101X00111100000
      00100111XXXXXXXX XXXX
 0025 XXXXX11001001110X XXXXXXXXXX111XXX XXX11XXXXX111110 1101X101011XXXXX
      XXXXXXX1XXXXXXXX XXXX

 TOTAL PHASE 2 ERRORS =   0
```

```
        ;       FADD ON ADDRESS MACHINE
        ;       OPERANDS IN ADDRESS REGISTER
        ;       CALL AS SUBROUTINE

0030            ORG     H#30

        FADD:
0030    /       FP.MODE0 E,M & E.SRS ,,,R1 & E.RAC AR,AR,PL
        /       & M.PAS ,,, & M.RAC ,AR,PL
        /               ;DETERMINE LARGEST EXPONENT
0031            FP.MODE0 & E.ALUOFF & M.ALUOFF
        /               ;ALU NO OPERATION
0032            FP.MODE0 ,M & E.READ PL,H#18 & E.ADD ,,R1,R2
        /       & E.RAC ,PL,PL & M.PAR QPT,,, & M.RAC AR,,
        /               ;ADD 24 (MANTISSA WIDTH) TO CHECK IF !NUMBER2!>>
        /               ;!NUMBER1! AND PREPARE Q REGISTER FOR ALLIGNING
0033            FP.MODE0 E,M & E.PAS ,,R1,R1 & E.RAC ,PL,PL
        /       & M.PAS LDQP,,, & M.RAC ,,
        /               ;ALLIGN MANTISSA M1, UPDATE EXP1
0034            FP.MODE0 E,M & E.PAS ,,,R3 & E.RAC ,AR,PL
        /       & M.ADD ,R1,, & M.RAC PL,QR,PL
        /               ;COPY EXPONENT FROM NUMBER2 (LARGEST) AND ADD M2 TO
        /               ;ALLIGNED M1 (IN Q REGISTER)
0035            FP.MODE0 E,M & E.PAS ,,R3, & E.RAC ,PL,AR
        /       & M.PAS LDR,,, & M.RAC ,PL,AR
        /               ;SUM EXCEEDED 24 BIT, CORRECT IT BY ADJUSTING THE EXPONENT
        /               ;AND SHIFTING THE MANTISSA 1 POSITION TO THE RIGHT
        RESNB1:
0036    /       FP.MODE0 E,M & E.PAR ,,, & E.RAC AR,,AR
        /       & M.PAR ,,, & M.RAC AR,,AR
        /               ;!NUMBER1!>>!NUMBER2! ==> RESULT:=NUMBER1
        NB1LARGER:
0037    /       FP.MODE0 & E.ALUOFF & M.ALUOFF
        /               ;ALU NO OPERATION
0038            FP.MODE0 ,M & E.ALUOFF & M.PAS QPT,,,
        /       & M.RAC ,AR,
        /               ;EXPONENTS ARE EQUAL, PREPARE Q REGISTER FOR ADDITION
        EXPNOTEQ:
0039    /       FP.MODE0 & E.READ PL,H#18 & E.SSR ,,R1,R2
        /       & E.RAC ,PL,PL & M.ALUOFF
        /               ;SUBTRACT 24 (MANTISSA WIDTH) TO CHECK IF !NUMBER2!<<
        /               ;!NUMBER1!
003A            FP.MODE0 ,M & E.ALUOFF & M.PAS QPT,,,
        /       & M.RAC ,AR,
        /               ;PREPARE Q REGISTER FOR ALLIGNING
003B            FP.MODE0 ,M & E.READ PL,H#FF & E.ADD ,,R1,R1
        /       & E.RAC ,PL,PL & M.PAS LDQP,,, & M.RAC ,,
        /               ;ALLIGN MANTISSA M2, UPDATE EXP2
        ADDM1Q:
003C    /       FP.MODE0 E,M & E.PAR ,,,R3 & E.RAC AR,,PL
        /       & M.ADD ,,, & M.RAC AR,QR,PL
        /               ;COPY EXPONENT FROM NUMBER1 (LARGEST) AND ADD
        /               ;M1 TO ALLIGNED M2 (IN Q REGISTER)
003D            FP.MODE0 E,M & E.PAS ,,R3, & E.RAC ,PL,AR
        /       & M.PAS LDR,,, & M.RAC ,PL,AR
        /               ;SUM EXCEEDED 24 BIT (MANTISSA WIDTH), SO CORRECT IT
        RESNB2:
```

```
 003E /       FP.MODE0 E,M & E.PAS ,,, & E.RAC ,AR,AR
      /       & M.PAS ,,, & M.RAC ,AR,AR
      /               ;(NUMBER2)>>(NUMBER1) ==> RESULT:=NUMBER2
      NORMALIZE:
 003F /       FP.MODE0 ,M & E.ALUOFF & M.PAS QPT,,R3,
      /       & M.RAC ,PL,
      /               ;LOST SIGNIFICANT BITS? PREPARE Q REGISTER FOR NORMALIZING
 0040         FP.MODE0 ,M & E.ALUOFF & M.SPFNW SLN,,,
      /       & M.SPFRAC ,,,
      /               ;TEST IF NUMBER ALREADY NORMALIZED
 0041         FP.MODE0 ,M & E.READ PL,H#FF & E.ADD ,,R3,R3
      /       & E.RAC ,PL,PL & M.SPF SLN,,, & M.SPFRAC ,,,
      /               ;NORMALIZE MANTISSA AND UPDATE EXPONENT
 0042         FP.MODE0 & E.ALUOFF & M.ALUOFF
      /               ;ALU NO OPERATION
      FZERO:
 0043 /       FP.MODE0 & E.READ PL,H#80 & E.PAR ,,,
      /       & E.RAC ,,AR & M.READ PL,H#00,H#0000 & M.PAR ,,,
      /       & M.RAC ,,AR
      /               ;LOAD IMMEDIATE A FLOATING ZERO
      STOREQ:
 0044 /       FP.MODE0 E,M & E.PAS ,,R3, & E.RAC ,PL,AR
      /       & M.PAS ,,, & M.RAC ,QR,AR
      /               ;MOVE RESULT FROM Q REGISTER INTO THE REGISTERFILE
      /               ;(REGISTER POINTED TO BY "C-REG" OF ADDRESS REGISTER)

              END
```

```
 0030 10100100011111IX XXXXXXX010000110 00111111XXXXXXXX XXXXXXXXXXXXXXXX
      X111XXXXXXXXXXX 0000010000010XX0 0010111111111111
 0031 101XXXX11XXXX11X XXXXXXXXXXX11XX1 11XXXX11XXXXXXXX XXXXXXXXXXXXXXXX
      X111XXXXXXXXXXX XXXXXXXXXXXXXXXX XXXX1111111111111
 0032 10100110011111100 0011000011000000 00011011XXXXXXXX XXXXXXXXXXXXXXXX
      X111XXXXXX000001 000010XX01010001 XXXX1111111111111
 0033 1G10100001111111X XXXXXXX010000110 00010111XXXXXXXX XXXXXXXXXXXXXXXX
      X111XXXXXX000001 000001XX01010XXX XXXX1111111111111
 0034 10101000011111IX XXXXXXX001100110 00111111XXXXXXXX XXXXXXXXXXXXXXXX
      X1110000011XXXXX 000011XX00010101 0110111111111111
 0035 10101000011111IX XXXXXXX010000110 00000111XXXXXXXX XXXXXXXXXXXXXXXX
      X111XXXXXX000011 XXXXXXXX01000XX0 1000111111111111
 0036 10101100011111IX XXXXXXX011000110 00111111XXXXXXXX XXXXXXXXXXXXXXXX
      X111XXXXXXXXXXX XXXXXX001XX00001 XX00111111111111
 0037 101XXXX11XXXX11X XXXXXXXXXXX11XX1 11XXXX11XXXXXXXX XXXXXXXXXXXXXXXX
      X111XXXXXXXXXXX XXXXXXXXXXXXXXXX XXXX1111111111111
 0038 101XXXX11XXXX11X XXXXXXX010000000 00011011XXXXXXXX XXXXXXXXXXXXXXXX
      X111XXXXXXXXXXX XXXXXXXXXXXXXXX0 00XX1111111111111
 0039 10100010011111100 0011000XXXX11XX1 11XXXX11XXXXXXXX XXXXXXXXXXXXXXXX
      X111XXXXXX000001 000010XX01010XXX XXXX1111111111111
 003A 101XXXX11XXXX11X XXXXXXX010000000 00011011XXXXXXXX XXXXXXXXXXXXXXXX
      X111XXXXXXXXXXX XXXXXXXXXXXXXXX0 00XX1111111111111
 003B 10100110011111101 1111111010000110 00010111XXXXXXXX XXXXXXXXXXXXXXXX
      X111XXXXXX000001 000001XX01010XXX XXXX1111111111111
 003C 10101100011111IX XXXXXXX001100110 00111111XXXXXXXX XXXXXXXXXXXXXXXX
      X111XXXXXXXXXXX 0000110011XX10001 0110111111111111
 003D 10101000011111IX XXXXXXX010000110 00000111XXXXXXXX XXXXXXXXXXXXXXXX
      X111XXXXXX000011 XXXXXXXX01000XX0 1000111111111111
 003E 10101000011111IX XXXXXXX010000110 00111111XXXXXXXX XXXXXXXXXXXXXXXX
      X111XXXXXXXXXXX XXXXXXXX00000XX0 0000111111111111
 003F 101XXXX11XXXX11X XXXXXXX010000000 00011011XXXXXXXX XXXXXXXXXXXXXXXX
      X111XXXXXX000011 XXXXXXXXXXXXXXX0 10XX1111111111111
 0040 101XXXX11XXXX11X XXXXXXX000011000 00100011XXXXXXXX XXXXXXXXXXXXXXXX
      X111XXXXXXXXXXX XXXXXXXXXXXXXXX0 XXXX1111111111111
 0041 10100110011111101 1111111000000000 00100011XXXXXXXX XXXXXXXXXXXXXXXX
      X111XXXXXX000011 000011XX01010XX0 XXXX1111111111111
 0042 101XXXX11XXXX11X XXXXXXXXXXX11XX1 11XXXX11XXXXXXXX XXXXXXXXXXXXXXXX
      X111XXXXXXXXXXX XXXXXXXXXXXXXXXX XXXX1111111111111
 0043 10101100011111101 0000000011000110 00111110000000000 0000000000000000
      X111XXXXXXXXXXX XXXXXXXX1XX00XX1 XX00111111111111
 0044 10101000011111IX XXXXXXX010000110 00111111XXXXXXXX XXXXXXXXXXXXXXXX
      X111XXXXXX000011 XXXXXXXX01000XX1 0100111111111111

 TOTAL PHASE 2 ERRORS =   0
```

```
        ;       FADD ON ADDRESS MACHINE
        ;       OPERANDS IN ADDRESS REGISTER
        ;       CALL AS SUBROUTINE

0030            ORG     H#30

        FADD:
0030 /          FP.MODE0 & E.REGCT M,I,F,I & E.CIN ONE
     /          & M.REGOFF & M.CIN ZERO & CJP
     /          & GOTO NB1LARGER & PREDICT F
     /                  ;JUMP ON |NUMBER1|>=|NUMBER2|
0031            FP.MODE0 & E.REGOFF & E.CT OVR,M
     /          & M.REGOFF & CJP & GOTO RESNB1
     /          & PREDICT F
     /                  ;JUMP ON |NUMBER1|>>|NUMBER2|
0032            FP.MODE0 & E.REGCT NC.OR.Z,I,,I & E.CIN ZERO
     /          & M.REG LOAD,M & CJP & GOTO RESNB2
     /          & PREDICT F
     /                  ;DETECT R1+24<=0 OR |NUMBER2|>>|NUMBER1|
0033            E.SHIFTOFF & E.REGCT Z,I,F,I & E.CIN ONE
     /          & M.SDMS & M.REGOFF & M.CIN ZERO
     /          & CJP & GOTO $ & PREDICT T
     /                  ;ALLIGN MANTISSA IN Q. SHIFT COUNT IS GIVEN BY (-R1)
0034            FP.MODE0 & E.REGOFF & E.CIN ZERO
     /          & M.REGCT OVR,I,F,I & M.CIN ZERO & CJP
     /          & GOTO NORMALIZE & PREDICT T
     /                  ;ADD MANTISSAS AND ON NON OVERFLOW GOTO NORMALIZE
0035            E.SHIFTOFF & E.REG LOAD,M & E.CIN ONE
     /          & M.SSXO & M.REG LOAD,M & M.CIN ZERO
     /          & FP.CTOFF & RTN & PREDICT OFF
     /                  ;OVERFLOW OCCURED ==> INVERT SIGN BIT, SHIFT MANTISSA
     /                  ;RIGHT 1 POSITION AND UPDATE EXPONENT
        RESNB1:
0036 /          FP.MODE0 & E.REG LOAD,M & E.CIN ZERO
     /          & M.REGOFF & M.CIN ZERO & FP.CTOFF
     /          & RTN & PREDICT OFF
     /                  ;|NUMBER1|>>|NUMBER2| SO RESULT IS NUMBER1
        NB1LARGER:
0037 /          FP.MODE0 & E.REGOFF & E.CT Z,I,F
     /          & M.REGOFF & CJP & GOTO EXPNOTEQ
     /          & PREDICT T
     /                  ;TEST EXPONENTS ARE UNEQUAL (WE'RE IN THE >= BRANCH)
0038            FP.MODE0 & E.REGOFF & M.REGOFF
     /          & M.CIN ZERO & FP.CTOFF & JMP
     /          & GOTO ADDN1Q & PREDICT OFF
     /                  ;EXPONENTS ARE EQUAL, SO GOTO ADD THEM!
        EXPNOTEQ:
0039 /          FP.MODE0 & E.REGCT OVR,M,,I & E.CIN ONE
     /          & M.REGOFF & CJP & GOTO RESNB2
     /          & PREDICT F
     /                  ;DETECT R1-24>=0 AND FREEZE CONDITION IN EXPONENT MSR.
     /                  ;SELECT OVR CONDITION CODE OF STEP 1 (SYMBOLIC ADDRESS:
     /                  ;FADD) TO ENSURE |NUMBER1|>|NUMBER2|.
```

```
FLOATING POINT ADD (SECOND PART)

      /                 ;R1-24)=0 OR |NUMBER1|>>|NUMBER2|
 003B           E.SHIFTOFF & E.REGCT Z,I,F,I & E.CIN ZERO
      /         & M.SDMS & M.REGOFF & M.CIN ZERO
      /         & CJP & GOTO $ & PREDICT T
      /                 ;ALLIGN MANTISSA M2 IN Q REGISTER AND UPDATE EXPONENT.
      ADDM1Q:
 003C /         FP.MODEO & E.REG LOAD,M & E.CIN ZERO
      /         & M.REGCT OVR,I,F,I & M.CIN ZERO & CJP
      /         & GOTO NORNALIZE & PREDICT T
      /                 ;ADD M1 TO ALLIGNED M2 IN Q AND COPY EXP1 INTO
      /                 ;EXPONENT RESULT.
 003D           E.SHIFTOFF & E.REG LOAD,M & E.CIN ONE
      /         & M.SSXO & M.REG LOAD,M & M.CIN ZERO
      /         & FP.CTOFF & RTN & PREDICT OFF
      /                 ;SUM EXCEEDED 24 BIT, CORRECT IT AND LEAVE FADD.
      RESNB2:
 003E /         FP.MODEO & E.REG LOAD,M & E.CIN ZERO
      /         & M.REGOFF & M.CIN ZERO & FP.CTOFF
      /         & RTN & PREDICT OFF
      /                 ;|NUMBER2|>>|NUMBER1| SO RESULT!=NUMBER2
      NORMALIZE:
 003F /         FP.MODEO & E.REGOFF & M.REGCT Z,I,T,I
      /         & M.CIN ZERO & CJP & GOTO FZERO
      /         & PREDICT F
      /                 ;IF MANTISSA=0 THEN RESULT!=FLOATING ZERO
 0040           E SHIFTOFF & E.REGOFF & M.SDUL
      /         & M.REGOFF & M.CT C,I & M.CIN ZERO
      /         & CJP & GOTO STOREQ & PREDICT F
      /                 ;CHECK MANTISSA IN Q REGISTER ON NORMALIZE-CONDITION
 0041           E.SHIFTOFF & E.REG LOAD,M & E.CIN ZERO
      /         & M.SDUL & M.REG LOAD,M & FP.CT CC5,T
      /         & CJP & GOTO $ & PREDICT T
      /                 ;REPEAT SHIFTING Q UNTIL ITS CONTENTS IS NORMALIZED
      /                 ;OR AN EXPONENT UNDERFLOW OCCURED.
 0042           FP.MODEO & E.REGOFF & E.CT OVR,M,F
      /         & M.REGOFF & CJP & GOTO STOREQ
      /         & PREDICT T
      /                 ;WAS THE TERMINATING OF THE LAST STEP CAUSED BY AN
      /                 ;EXPONENT UNDERFLOW?
      FZERO:
 0043 /         FP.MODEO & E.REG LOAD,M & E.CIN ZERO
      /         & M.REG LOAD,M & M.CIN ZERO & FP.CTOFF
      /         & RTN & PREDICT OFF
      /                 ;YES, IT WAS AN UNDERFLOW! RESULT!=FLOATING ZERO
      STOREQ:
 0044 /         FP.MODEO & E.REG LOAD,M & E.CIN ZERO
      /         & M.REG LOAD,M & M.CIN ZERO & FP.CTOFF
      /         & RTN & PREDICT OFF
      /                 ;Q REGISTER CONTAINS THE RESULT, SO STORE IT IN THE
      /                 ;REGISTERFILE.

                END
```

```
 0030 XXXXX10111110110 1XXXXXXXXX111XXX XXX11XXX0011111X XX11X00111100000
      01101111XXXXXXXX XXXX
 0031 XXXXX10011101111 XXXXXXXXXX111XXX XXX11XXXXX11111X XX11X00111100000
      01101101XXXXXXXX XXXX
 0032 XXXXX1C100111110 0XXXXXXXXX111000 10010111XX11111X XX11X00111100000
      01111101XXXXXXXX XXXX
 0033 XXXXX10010110110 1001011010001XXX XXX11XXX00XXXXXX XX11X00111100000
      01100110XXXXXXXX XXXX
 0034 XXXXX11XXXXXX110 0XXXXXXXXX110011 11011111001111X XX11X00111100000
      01111110XXXXXXXX XXXX
 0035 XXXXX11000100100 1011101010001000 10010111000XXXX0 1101X101011XXXXX
      XXXXXXX1XXXXXXXX XXXX
 0036 XXXXX11000100100 0XXXXXXXXX111XXX XXX11XXX00111110 1101X101011XXXXX
      XXXXXXX1XXXXXXXX XXXX
 0037 XXXXX10010110111 XXXXXXXXXX111XXX XXX11XXXXX11111X XX11X00111100000
      01110010XXXXXXXX XXXX
 0038 XXXXX11XXXXXX11X XXXXXXXXXX111XXX XXX11XXX00111110 1101X00111100000
      01111001XXXXXXXX XXXX
 0039 XXXXX10011101110 1XXXXXXXXX111XXX XXX11XXXXX11111X XX11X00111100000
      01111101XXXXXXXX XXXX
 003A XXXXX10001100111 XXXXXXXXXX111XXX XXX11XXX00111111X XX11X00111100000
      01101101XXXXXXXX XXXX
 003B XXXXX10010110110 0001011010001XXX XXX11XXX00XXXXXX XX11X00111100000
      01110110XXXXXXXX XXXX
 003C XXXXX11000100100 0XXXXXXXXX110011 11011111001111X XX11X00111100000
      01111110XXXXXXXX XXXX
 003D XXXXX11000100100 1011101010001000 10010111000XXXX0 1101X101011XXXXX
      XXXXXXX1XXXXXXXX XXXX
 003E XXXXX11000100100 0XXXXXXXXX111XXX XXX11XXX00111110 1101X101011XXXXX
      XXXXXXX1XXXXXXXX XXXX
 003F XXXXX11XXXXXX11X XXXXXXXXXX110010 11111111001111X XX11X00111100000
      10000111XXXXXXXX XXXX
 0040 XXXXX1XXXXXXX11X X101101010000101 1111111100XXXXXX XX11X00111100000
      10001001XXXXXXXX XXXX
 0041 XXXXX11000100100 0101101010001000 10010111XXXXXXX1 0110000111100000
      10000010XXXXXXXX XXXX
 0042 XXXXX10011110011X XXXXXXXXXX111XXX XXX11XXXXX11111X XX11X00111100000
      10001000XXXXXXXX XXXX
 0043 XXXXX11000100100 0XXXXXXXXX111000 10010111001111I0 1101X101011XXXXX
      XXXXXXX1XXXXXXXX XXXX
 0044 XXXXX11000100100 0XXXXXXXXX111000 10010111001111I0 1101X101011XXXXX
      XXXXXXX1XXXXXXXX XXXX

 TOTAL PHASE 2 ERRORS =   0
```

```
FLOATING POINT SUBTRACT (FIRST PART)

        ;       FSUB ON ADDRESS MACHINE
        ;       OPERANDS IN ADDRESS REGISTER
        ;       CALL AS SUBROUTINE

0050            ORG     H#50

        FSUB:
0050    /       FP.MODE0 E,M & E.SRS ,,,R1 & E.RAC AR,AR,PL
        /       & M.PAS ,,, & M.RAC ,AR,PL
        /               ;DETERMINE LARGEST EXPONENT
0051            FP.MODE0 & E.ALUOFF & M.ALUOFF
        /               ;ALU NO OPERATION
0052            FP.MODE0 ,M & E.READ PL,H#18 & E.ADD ,,R1,R2
        /       & E.RAC ,PL,PL & M.PAR QPT,,, & M.RAC AR,,
        /               ;ADD 24 (MANTISSA WIDTH) TO CHECK IF !NUMBER2!>>
        /               ;!NUMBER1! AND PREPARE Q REGISTER FOR ALLIGNING
0053            FP.MODE0 E,M & E.PAS ,,R1,R1 & E.RAC ,PL,PL
        /       & M.PAS LDQP,,, & M.RAC ,,
        /               ;ALLIGN MANTISSA M1, UPDATE EXP1
0054            FP.MODE0 E,M & E.PAS ,,,R3 & E.RAC ,AR,PL
        /       & M.SSR ,R1,, & M.RAC PL,QR,PL
        /               ;COPY EXPONENT FROM NUMBER2 (LARGEST) AND SUBTRACT
        /               ;ALLIGNED M1 (IN Q REGISTER) FROM M2
0055            FP.MODE0 E,M & E.PAS ,,R3, & E.RAC ,PL,AR
        /       & M.PAS LDR,,, & M.RAC ,PL,AR
        /               ;DIFFERENCE EXCEEDED 24 BIT, CORRECT IT BY ADJUSTING THE
        /               ;EXPONENT AND BY SHIFTING THE MANTISSA 1 POSITION TO
        /               ;THE RIGHT
        RESNB1:
0056    /       FP.MODE0 E,M & E.PAR ,,, & E.RAC AR,,AR
        /       & M.PAR ,,, & M.RAC AR,,AR
        /               ;!NUMBER1!>>!NUMBER2! ==> RESULT:=NUMBER1
        NB1LARGER:
0057    /       FP.MODE0 & E.ALUOFF & M.ALUOFF
        /               ;ALU NO OPERATION
0058            FP.MODE0 ,M & E.ALUOFF & M.PAS QPT,,,
        /       & M.RAC ,AR,
        /               ;EXPONENTS ARE EQUAL, PREPARE Q REGISTER FOR SUBTRACTION
        EXPNOTEQ:
0059    /       FP.MODE0 & E.READ PL,H#18 & E.SSR ,,R1,R2
        /       & E.RAC ,PL,PL & M.ALUOFF
        /               ;SUBTRACT 24 (MANTISSA WIDTH) TO CHECK IF !NUMBER2!<<
        /               ;!NUMBER1!
005A            FP.MODE0 ,M & E.ALUOFF & M.PAS QPT,,,
        /       & M.RAC ,AR,
        /               ;PREPARE Q REGISTER FOR ALLIGNING
005B            FP.MODE0 ,M & E.READ PL,H#FF & E.ADD ,,R1,R1
        /       & E.RAC ,PL,PL & M.PAS LDQP,,, & M.RAC ,,
        /               ;ALLIGN MANTISSA M2, UPDATE EXP2
        ADDM1Q:
005C    /       FP.MODE0 E,M & E.PAR ,,,R3 & E.RAC AR,,PL
        /       & M.SRS ,,, & M.RAC AR,QR,PL
        /               ;COPY EXPONENT FROM NUMBER1 (LARGEST) AND SUBTRACT
        /               ;ALLIGNED M2 (IN Q REGISTER) FROM M1
005D            FP.MODE0 E,M & E.PAS ,,R3, & E.RAC ,PL,AR
        /       & M.PAS LDR,,, & M.RAC ,PL,AR
        /               ;DIFFERENCE EXCEEDED 24 BIT (MANTISSA WIDTH)! CORRECT IT!
```

```
      RESNB2:
 005E /       FP.MODE0 E,M & E.PAS ,,, & E.RAC ,AR,AR
      /       & M.COMS ,,, & M.RAC ,AR,AR
      /                ;(NUMBER2)>>(NUMBER1) ==> RESULT:=NUMBER2
      NORMALIZE:
 005F /       FP.MODE0 ,M & E.ALUOFF & M.PAS QPT,,R3,
      /       & M.RAC ,PL,
      /                ;LOST SIGNIFICANT BITS? PREPARE Q REGISTER FOR NORMALIZING
 0060         FP.MODE0 ,M & E.ALUOFF & M.SPFNW SLN,,,
      /       & M.SPFRAC ,,,
      /                ;TEST IF NUMBER ALREADY NORMALIZED
 0061         FP.MODE0 ,M & E.READ PL,H#FF & E.ADD ,,R3,R3
      /       & E.RAC ,PL,PL & M.SPF SLN,,, & M.SPFRAC ,,,
      /                ;NORMALIZE MANTISSA AND UPDATE EXPONENT
 0062         FP.MODE0 & E.ALUOFF & M.ALUOFF
      /                ;ALU NO OPERATION
      FZERO:
 0063 /       FP.MODE0 & E.READ PL,H#80 & E.PAR ,,,
      /       & E.RAC ,,AR & M.READ PL,H#00,H#0000 & M.PAR ,,,
      /       & M.RAC ,,AR
      /                ;LOAD IMMEDIATE A FLOATING ZERO
      STOREQ:
 0064 /       FP.MODE0 E,M & E.PAS ,,R3, & E.RAC ,PL,AR
      /       & M.PAS ,,, & M.RAC ,QR,AR
      /                ;MOVE RESULT FROM Q REGISTER INTO THE REGISTERFILE
      /                ;(REGISTER POINTED TO BY "C-REG" OF ADDRESS REGISTER)

               END
```

```
0050 1010010001111111 XXXXXXX010000110 0011111XXXXXXXXX XXXXXXXXXXXXXXX
     X111XXXXXXXXXXX 0000010000010XX0 0010111111111111
0051 101XXXX11XXXX11X XXXXXXXXXXX11XX1 11XXXX11XXXXXXXX XXXXXXXXXXXXXXX
     X111XXXXXXXXXXX XXXXXXXXXXXXXXX XXXX111111111111
0052 1010011001111100 0011000011000000 0001101IXXXXXXXX XXXXXXXXXXXXXXX
     X111XXXXX000001 000010XX01010001 XXXX111111111111
0053 1010100001111111 XXXXXXX010000110 0001011IXXXXXXXX XXXXXXXXXXXXXXX
     X111XXXXX000001 000001XX01010XXX XXXX111111111111
0054 1010100001111111 XXXXXXX0001001I0 0011111IXXXXXXXX XXXXXXXXXXXXXXX
     X111000001XXXXX 000011XX00010101 0110111111111111
0055 1010100001111111 XXXXXXX010000110 0000011IXXXXXXXX XXXXXXXXXXXXXXX
     X111XXXXX000011 XXXXXXXX01000XX0 1000111111111111
0056 1010110001111111 XXXXXXX011000110 0011111IXXXXXXXX XXXXXXXXXXXXXXX
     X111XXXXXXXXXXX XXXXXX001XX00001 XX00111111111111
0057 101XXXX11XXXX11X XXXXXXXXXXX11XX1 11XXXX11XXXXXXXX XXXXXXXXXXXXXXX
     X111XXXXXXXXXXX XXXXXXXXXXXXXXX XXXX111111111111
0058 101XXXX11XXXX11X XXXXXXX010000000 0001101IXXXXXXXX XXXXXXXXXXXXXXX
     X111XXXXXXXXXXX XXXXXXXXXXXXXXX0 00XX111111111111
0059 1010001001111100 0011000XXXX11XX1 11XXXX11XXXXXXXX XXXXXXXXXXXXXXX
     X111XXXXX000001 000010XX01010XXX XXXX111111111111
005A 101XXXX11XXXX11X XXXXXXX010000000 0001101IXXXXXXXX XXXXXXXXXXXXXXX
     X111XXXXXXXXXXX XXXXXXXXXXXXXXX0 00XX11F111111111
005B 1010011001111101 1111111010000110 0001011IXXXXXXXX XXXXXXXXXXXXXXX
     X111XXXXX000001 000001XX01010XXX XXXX111111111111
005C 1010110001111111 XXXXXXX001000110 0011111IXXXXXXXX XXXXXXXXXXXXXXX
     X111XXXXXXXXXXX 000011001XX10001 0110111111111111
005D 1010100001111111 XXXXXXX010000110 0000011IXXXXXXXX XXXXXXXXXXXXXXX
     X111XXXXX000011 XXXXXXXX01000XX0 1000111111111111
005E 1010100001111111 XXXXXXX010100110 0011111IXXXXXXXX XXXXXXXXXXXXXXX
     X111XXXXXXXXXXX XXXXXXXX00000XX0 0000111111111111
005F 101XXXX11XXXX11X XXXXXXX010000000 0001101IXXXXXXXX XXXXXXXXXXXXXXX
     X111XXXXX000011 XXXXXXXXXXXXXXX0 10XX111111111111
0060 101XXXX11XXXX11X XXXXXXX0000110I0 0010001IXXXXXXXX XXXXXXXXXXXXXXX
     X111XXXXXXXXXXX XXXXXXXXXXXXXXX0 XXXX111111111111
0061 1010011001111101 1111111000000000 0010001IXXXXXXXX XXXXXXXXXXXXXXX
     X111XXXXX000011 000011XX01010XX0 XXXX111111111111
0062 101XXXX11XXXX11X XXXXXXXXXXX11XX1 11XXXX11XXXXXXXX XXXXXXXXXXXXXXX
     X111XXXXXXXXXXX XXXXXXXXXXXXXXX XXXX111111111111
0063 1010110001111101 0000000011000110 0011111000000000 0000000000000000
     X111XXXXXXXXXXX XXXXXXXX1XX00XX1 XX00111111111111
0064 1010100001111111 XXXXXXX010000110 0011111IXXXXXXXX XXXXXXXXXXXXXXX
     X111XXXXXX000011 XXXXXXXX01000XX1 0100111111111111

TOTAL PHASE 2 ERRORS =  0
```

```
        ;       FSUB ON ADDRESS MACHINE
        ;       OPERANDS IN ADDRESS REGISTER
        ;       CALL AS SUBROUTINE

0050            ORG     H#50

        FSUB:
0050    /       FP.MODE0 & E.REGCT N,I,F,I & E.CIN ONE
        /       & M.REGOFF & M.CIN ZERO & CJP
        /       & GOTO NB1LARGER & PREDICT F
        /               ;JUMP ON |NUMBER1|>=|NUMBER2|
0051            FP.MODE0 & E.REGOFF & E.CT OVR,M
        /       & M.REGOFF & CJP & GOTO RESNB1
        /       & PREDICT F
        /               ;JUMP ON |NUMBER1|>|NUMBER2|
0052            FP.MODE0 & E.REGCT NC.OR.Z,I,,I & E.CIN ZERO
        /       & M.REG LOAD,M & CJP & GOTO RESNB2
        /       & PREDICT F
        /               ;DETECT R1+24(=0 OR |NUMBER2|>|NUMBER1|
0053            E.SHIFTOFF & E.REGCT Z,I,F,I & E.CIN ONE
        /       & M.SDMS & M.REGOFF & M.CIN ZERO
        /       & CJP & GOTO $ & PREDICT T
        /               ;ALLIGN MANTISSA IN Q, SHIFT COUNT IS GIVEN BY (-R1)
0054            FP.MODE0 & E.REGOFF & E.CIN ZERO
        /       & M.REGCT OVR,I,F,I & M.CIN ZERO & CJP
        /       & GOTO NORMALIZE & PREDICT T
        /               ;SUBTRACT MANTISSAS AND ON NON OVERFLOW GOTO NORMALIZE
0055            E.SHIFTOFF & E.REG LOAD,M & E.CIN ONE
        /       & M.SSXO & M.REG LOAD,M & M.CIN ZERO
        /       & FP.CTOFF & RTN & PREDICT OFF
        /               ;OVERFLOW OCCURED ==> INVERT SIGN BIT, SHIFT MANTISSA
        /               ;RIGHT 1 POSITION AND UPDATE EXPONENT
        RESNB1:
0056    /       FP.MODE0 & E.REG LOAD,M & E.CIN ZERO
        /       & M.REGOFF & M.CIN ZERO & FP.CTOFF
        /       & RTN & PREDICT OFF
        /               ;|NUMBER1|>|NUMBER2| SO RESULT IS NUMBER1
        NB1LARGER:
0057    /       FP.MODE0 & E.REGOFF & E.CT Z,I,F
        /       & M.REGOFF & CJP & GOTO EXPNOTEQ
        /       & PREDICT T
        /               ;TEST EXPONENTS ARE UNEQUAL (WE'RE IN THE >= BRANCH)
0058            FP.MODE0 & E.REGOFF & M.REGOFF
        /       & M.CIN ZERO & FP.CTOFF & JMP
        /       & GOTO ADDM1Q & PREDICT OFF
        /               ;EXPONENTS ARE EQUAL, SO GOTO SUBTRACT THEM!
        EXPNOTEQ:
0059    /       FP.MODE0 & E.REGCT OVR,M,,I & E.CIN ONE
        /       & M.REGOFF & CJP & GOTO RESNB2
        /       & PREDICT F
        /               ;DETECT R1-24)=0 AND FREEZE CONDITION IN EXPONENT MSR.
        /               ;SELECT OVR CONDITION CODE OF STEP 1 (SYMBOLIC ADDRESS!
        /               ;FSUB) TO ENSURE |NUMBER1|>|NUMBER2|.
```

```
      /                 ;R1-24)=0 OR |NUMBER1|>>|NUMBER2|
005B          E.SHIFTOFF & E.REGCT Z,I,F,I & E.CIN ZERO
      /       & M.SDMS & M.REGOFF & M.CIN ZERO
      /       & CJP & GOTO $ & PREDICT T
      /                 ;ALLIGN MANTISSA M2 IN Q REGISTER AND UPDATE EXPONENT.
      ADDM1Q:
005C  /       FP.MODEO & E.REG LOAD,M & E.CIN ZERO
      /       & M.REGCT OVR,I,F,I & M.CIN ZERO & CJP
      /       & GOTO NORMALIZE & PREDICT T
      /                 ;ADD M1 TO ALLIGNED M2 IN Q AND COPY EXP1 INTO
      /                 ;EXPONENT RESULT.
005D          E.SHIFTOFF & E.REG LOAD,M & E.CIN ONE
      /       & M.SSXO & M.REG LOAD,M & M.CIN ZERO
      /       & FP.CTOFF & RTN & PREDICT OFF
      /                 ;DIFFERENCE EXCEEDED 24 BIT, CORRECT IT AND LEAVE FSUB.
      RESNB2:
005E  /       FP.MODEO & E.REG LOAD,M & E.CIN ZERO
      /       & M.REGOFF & M.CIN ONE & FP.CTOFF
      /       & RTN & PREDICT OFF
      /                 ;|NUMBER2|>>|NUMBER1| SO RESULT:=NUMBER2
      NORMALIZE:
005F  /       FP.MODEO & E.REGOFF & M.REGCT Z,I,T,I
      /       & M.CIN ZERO & CJP & GOTO FZERO
      /       & PREDICT F
      /                 ;IF MANTISSA=0 THEN RESULT:=FLOATING ZERO
0060          E.SHIFTOFF & E.REGOFF & M.SDUL
      /       & M.REGOFF & M.CT C,I & M.CIN ZERO
      /       & CJP & GOTO STOREQ & PREDICT F
      /                 ;CHECK MANTISSA IN Q REGISTER ON NORMALIZE-CONDITION
0061          E.SHIFTOFF & E.REG LOAD,M & E.CIN ZERO
      /       & M.SDUL & M.REG LOAD,M & FP.CT CC5,T
      /       & CJP & GOTO $ & PREDICT T
      /                 ;REPEAT SHIFTING Q UNTIL ITS CONTENTS IS NORMALIZED
      /                 ;OR AN EXPONENT UNDERFLOW OCCURED.
0062          FP.MODEO & E.REGOFF & E.CT OVR,M,F
      /       & M.REGOFF & CJP & GOTO STOREQ
      /       & PREDICT T
      /                 ;WAS THE TERMINATING OF THE LAST STEP CAUSED BY AN
      /                 ;EXPONENT UNDERFLOW?
      FZERO:
0063  /       FP.MODEO & E.REG LOAD,M & E.CIN ZERO
      /       & M.REG LOAD,M & M.CIN ZERO & FP.CTOFF
      /       & RTN & PREDICT OFF
      /                 ;YES, IT WAS AN UNDERFLOW! RESULT:=FLOATING ZERO
      STOREQ:
0064  /       FP.MODEO & E.REG LOAD,M & E.CIN ZERO
      /       & M.REG LOAD,M & M.CIN ZERO & FP.CTOFF
      /       & RTN & PREDICT OFF
      /                 ;Q REGISTER CONTAINS THE RESULT, SO STORE IT IN THE
      /                 ;REGISTERFILE.

                END
```

```
 0050 XXXXX10111110110 1XXXXXXXXX111XXX XXX11XXX0011111X XX11X00111100000
      10101111XXXXXXXX XXXX
 0051 XXXXX10011101111 XXXXXXXXXX111XXX XXX11XXXX11111X XX11X00111100000
      10101101XXXXXXXX XXXX
 0052 XXXXX10100111110 0XXXXXXXXX111000 10010111XX11111X XX11X00111100000
      10111101XXXXXXXX XXXX
 0053 XXXXX10010110110 1001011010001XXX XXX11XXX00XXXXXX XX11X00111100000
      10100110XXXXXXXX XXXX
 0054 XXXXX11XXXXXX110 0XXXXXXXXX110011 11011111001111X XX11X00111100000
      10111110XXXXXXXX XXXX
 0055 XXXXX11000100100 1011101010001000 10010111100XXXX0 1101X101011XXXXX
      XXXXXXX1XXXXXXXX XXXX
 0056 XXXXX11000100100 0XXXXXXXXX111XXX XXX11XXX00111110 1101X101011XXXXX
      XXXXXXX1XXXXXXXX XXXX
 0057 XXXXX10010110111 XXXXXXXXXX111XXX XXX11XXXX11111X XX11X00111100000
      10110010XXXXXXXX XXXX
 0058 XXXXX11XXXXXX11X XXXXXXXXXX111XXX XXX11XXX00111110 1101X00111100000
      10111001XXXXXXXX XXXX
 0059 XXXXX10011101110 1XXXXXXXXX111XXX XXX11XXXX11111X XX11X00111100000
      10111101XXXXXXXX XXXX
 005A XXXXX10001100111 XXXXXXXXXX111XXX XXX11XXX00111111X XX11X00111100000
      10101101XXXXXXXX XXXX
 005B XXXXX10010110110 0001011010001XXX XXX11XXX00XXXXXX XX11X00111100000
      10110110XXXXXXXX XXXX
 005C XXXXX11000100100 0XXXXXXXXX110011 11011111001111X XX11X00111100000
      10111110XXXXXXXX XXXX
 005D XXXXX11000100100 1011101010001000 10010111100XXXX0 1101X101011XXXXX
      XXXXXXX1XXXXXXXX XXXX
 005E XXXXX11000100100 0XXXXXXXXX111XXX XXX11XXX01111110 1101X101011XXXXX
      XXXXXXX1XXXXXXXX XXXX
 005F XXXXX11XXXXXX11X XXXXXXXXXX110010 11111111001111X XX11X00111100000
      11000111XXXXXXXX XXXX
 0060 XXXXX1XXXXXXX11X X101101010000101 11111111100XXXXX XX11X00111100000
      11001001XXXXXXXX XXXX
 0061 XXXXX11000100100 0101101010001000 10010111XXXXXXX1 0110000111100000
      11000010XXXXXXXX XXXX
 0062 XXXXX10011100111 XXXXXXXXXX111XXX XXX11XXXXX11111X XX11X00111100000
      11001000XXXXXXXX XXXX
 0063 XXXXX11000100100 0XXXXXXXXX111000 10010111001111110 1101X101011XXXXX
      XXXXXXX1XXXXXXXX XXXX
 0064 XXXXX11000100100 0XXXXXXXXX111000 10010111001111110 1101X101011XXXXX
      XXXXXXX1XXXXXXXX XXXX

 TOTAL PHASE 2 ERRORS =   0
```

```
                ;               FMUL ON STACK MACHINE
                ;               OPERANDS ON STACK
                ;                       POP MULTIPLICAND FROM STACK
                ;                       POP MULTIPLIER FROM STACK
                ;                       PUSH PRODUCT BACK ON STACK
                ;               CALL AS SUBROUTINE

0100            ORG     H#100

        FMUL:
0100    /       FP.MODE0 E,M & E.PAR ,,,R2 & E.RAC IR,,PL
        /       & M.PAR ,,, & M.RAC IR,,PL
        /               ;POP MULTIPLICAND FROM STACK AND MOVE TO R2
0101            FLOAT & E.ADD ,R1,R0,R0 & E.RAC PL,PL,PL
        /       & M.ALUOFF & FP.WRITE XDOFF,,,IR & FP.XCEIVOFF
        /               ;DECREMENT STACK POINTER (R1 CONTAINS: -1)
        /               ;MANTISSA PERFORMS DUMMY FUNCTION
0102            FP.MODE0 E,M & E.PAR ,,,R3 & E.RAC IR,,PL
        /       & M.PAR ,,, & M.RAC IR,,PL
        /               ;POP MULTIPLIER FROM STACK AND MOVE TO R3
0103            FP.MODE0 E,M & E.ADD ,R2,R3,R4 & E.RAC PL,PL,PL
        /       & M.PAS QPT & M.RAC ,PL
        /               ;ADD EXPONENTS AND LOAD Q
0104            FP.MODE0 ,M & E.ALUOFF & M.PAR ,R2,,R2
        /       & M.RAC PL,,PL
        /               ;TEST MULTIPLICAND (WRITE TO DUMMY DESTINATION)
0105            FP.MODE0 & E.ALUOFF & M.READ PL,H#00,H#0000
        /       & M.PAR ,,,R4 & M.RAC ,,PL
        /               ;CLEAR PARTIAL PRODUCT REGISTER R4
0106            FP.MODE0 ,M & E.ALUOFF & M.SPF TCM,R2,R4,R4
        /       & M.SPFRAC PL,PL,PL
        /               ;2'S COMPLEMENT MULTIPLY ( 1 TIME )
0107            FP.MODE0 ,M & E.ALUOFF & M.SPF TCM,R2,R4,R4
        /       & M.SPFRAC PL,PL,PL
        /               ;2'S COMPLEMENT MULTIPLY ( 22 TIMES )
0108            FP.MODE0 ,M & E.ALUOFF & M.SPF TCMC,R2,R4,R4
        /       & M.SPFRAC PL,PL,PL
        /               ;2'S COMPLEMENT MULTIPLY CORRECTION ( 1 TIME )
0109            FP.MODE0 ,M & E.ALUOFF & M.SPFNW DLN,,R4,
        /       & M.SPFRAC ,PL,
        /               ;TEST FOR ALREADY NORMALIZED RESULT AND
        /               ;DON'T WRITE!
010A            FP.MODE0 E,M  & E.ADD ,,R4,R4
        /       & E.RAC PL,PL,PL & M.SPF DLN,,, & M.SPFRAC ,PL,PL
        /               ;UPDATE SHIFT COUNT
010B            FP.MODE0 E,M & E.PAS ,,R4,R4 & E.RAC ,PL,PL
        /       & M.PAS ,,, & M.RAC ,PL,PL
        /               ;READ R4 TO PRODUCE STATUSBIT FOR CC LOGIC
        FZERO:
010C    /       FP.MODE0 & E.READ PL,H#80 & E.PAR ,,,R4
        /       & E.RAC ,,PL & M.READ PL,H#00,H#0000 & M.PAR ,,,
        /       & M.RAC ,,PL
        /               ;LOAD IMMEDIATE A "CLEAN ZERO"
        EXPOVERF:
010D    /       FP.MODE0 ,M & E.ALUOFF & M.SPFNW DLN,,R4,
        /       & M.SPFRAC ,PL,
        /               ;EXPONENT OVERFLOW OCCURED: TRY TO CORRECT IT
```

```
      /                 )BY NORMALIZING
 010E                FP.MODEO E,M & E.ADD ,R1,R4,R4 & E.RAC PL,PL,PL
      /                & M.SPF DLN,,, & M.SPFRAC ,PL,PL
      /                 )NORMALIZE AND UPDATE SHIFT COUNT
 010F                FP.MODEO & E.ALUOFF & M.ALUOFF
      /                 )ALU NO OPERATION
      SETOVR:
 0110 /               FP.MODEO & E.ALUOFF & M.ALUOFF
      /                 )ALU NO OPERATION
      PUSHRESULT:
 0111 /               FP.MODEO E,M & E.PAR ,R4,, & E.RAC PL,,IR
      /               & M.PAR ,,, & M.RAC PL,,IR
      /                 )PUSH PRODUCT ON STACK

                      END
```

```
0100 10101100011111X XXXXXXX011000110 00111111XXXXXXXX XXXXXXXXXXXXXXX
     X111XXXXXXXXXXX 000010111XX10111 XX10111111111111
0101 10100110011111X XXXXXXXXXXX11XX1 11XXXX1XXXXXXXX XXXXXXXXXXXXXXX
     10100000010000O0 000000100101OXXX XXXX111111111111
0102 10101100011111X XXXXXXX011000110 00111111XXXXXXXX XXXXXXXXXXXXXXX
     X111XXXXXXXXXXX 000011111XX10111 XX10111111111111
0103 10100110011111X XXXXXXX010000000 00011011XXXXXXXX XXXXXXXXXXXXXXX
     X11100001000001l 000100100101OXX0 10XX111111111111
0104 101XXXX11XXXX11X XXXXXXX011000110 00111111XXXXXXXX XXXXXXXXXXXXXXX
     X11100001OXXXXXX 000010XXXXXXX101 XX10111111111111
0105 101XXXX11XXXX11X XXXXXXX011000110 00111110000000O0 0000000000000000
     X111XXXXXXXXXXX 000100XXXXXXXXX1 XX10111111111111
0106 101XXXX11XXXX11X XXXXXXX000000000 00001011XXXXXXXX XXXXXXXXXXXXXXX
     X11100001000010O 000100XXXXXXX100 10101111111111111
0107 101XXXX11XXXX11X XXXXXXX000000000 00001011XXXXXXXX XXXXXXXXXXXXXXX
     X11100001000010O 000100XXXXXXX100 10101111111111111
0108 101XXXX11XXXX11X XXXXXXX000000000 00011011XXXXXXXX XXXXXXXXXXXXXXX
     X11100001000010O 000100XXXXXXX100 10101111111111111
0109 101XXXX11XXXX11X XXXXXXX000011000 00101011XXXXXXXX XXXXXXXXXXXXXXX
     X111XXXXX000100 XXXXXXXXXXXXXXX0 10XX111111111111
010A 10100110011111X XXXXXXX000000000 00101011XXXXXXXX XXXXXXXXXXXXXXX
     X111XXXXX000100 000100100101OXX0 10101111111111111
010B 10101000011111X XXXXXXX010000110 00111111XXXXXXXX XXXXXXXXXXXXXXX
     X111XXXXX000100 000100XX01010XX0 10101111111111111
010C 10101100011111O1 0000000011000110 00111110000000O0 0000000000000000
     X111XXXXXXXXXXX 000100XX1XX10XX1 XX10111111111111
010D 101XXXX11XXXX11X XXXXXXX000011000 00101011XXXXXXXX XXXXXXXXXXXXXXX
     X111XXXXX000100 XXXXXXXXXXXXXX0 10XX111111111111
010E 10100110011111X XXXXXXX000000000 00101011XXXXXXXX XXXXXXXXXXXXXXX
     X11100000100010O 000100100101OXX0 10101111111111111
010F 101XXXX11XXXX11X XXXXXXXXXXX11XX1 11XXXX1XXXXXXXX XXXXXXXXXXXXXXX
     X111XXXXXXXXXXX XXXXXXXXXXXXXXX XXXX111111111111
0110 101XXXX11XXXX11X XXXXXXXXXXX11XX1 11XXXX1XXXXXXXX XXXXXXXXXXXXXXX
     X111XXXXXXXXXXX XXXXXXXXXXXXXXX XXXX111111111111
0111 10101100011111X XXXXXXX011000110 00111111XXXXXXXX XXXXXXXXXXXXXXX
     X11100010OXXXXXX XXXXXX101XX11101 XX11111111111111
```

TOTAL PHASE 2 ERRORS = 0

```
        ;       FMUL ON STACK MACHINE
        ;       OPERANDS ON STACK
        ;               POP MULTIPLICAND FROM STACK
        ;               POP MULTIPLIER FROM STACK
        ;               PUSH PRODUCT ON STACK
        ;       CALL AS SUBROUTINE

 0100           ORG     H#100

      FMUL:
 0100 /         FP.MODE0 & E.REGOFF & E.CIN ZERO
      /         & M.REGOFF & M.CIN ZERO & CONT
      /         & PREDICT OFF
      /                 ;POP MULTIPLICAND
 0101           FP.MODE0 & E.REGOFF & E.CIN ZERO
      /         & M.REGOFF & CONT & PREDICT OFF
      /                 ;DECREMENT STACK POINTER.
      /                 ;A TEST ON STACK OVERFLOW/UNDERFLOW IS POSSIBLE
      /                 ;WITHOUT SIGNIFICANT LOSS OF EXECUTION TIME.
 0102           FP.MODE0 & E.REGOFF & E.CIN ZERO
      /         & M.REGOFF & M.CIN ZERO & CONT
      /         & PREDICT OFF
      /                 ;POP MULTIPLIER
 0103           FP.MODE0 & E.REG LOAD,M & E.CIN ZERO
      /         & M.CT Z,I & M.CIN ZERO & CJP
      /         & GOTO FZERO & PREDICT F
      /                 ;JUMP ON MULTIPLIER EQUALS TO ZERO
 0104           FP.MODE0 & E.REG PAS.M & M.REG LOAD,M
      /         & M.CIN ZERO & FP.CT CC1 & CJP
      /         & GOTO FZERO & PREDICT F
      /                 ;JUMP ON EXPONENT UNDERFLOW
 0105           FP.MODE0 & E.REG PAS.M & M.CT Z,M
      /         & M.CIN ZERO & CJP & GOTO FZERO
      /         & PREDICT F
      /                 ;JUMP ON MULTIPLICAND ZERO
 0106           E.SHIFTOFF & E.REGOFF & M.SSX0
      /         & M.REGOFF & FP.CTOFF & LDCT
      /         & COUNT M.WMIN2 & ST.OFF & PREDICT OFF
      /                 ;LOAD COUNTER AND DO FIRST MULTIPLY OPERATION
 0107           E.SHIFTOFF & E.REGOFF & M.SSX0
      /         & M.REGOFF & FP.CTOFF & RPCT
      /         & GOTO $ & ST.OFF & PREDICT OFF
      /                 ;2'S COMPLEMENT MULTIPLY ( 22 TIMES )
 0108           E.SHIFTOFF & E.REGOFF & M.SSX0
      /         & M.REGOFF & M.CIN CX & FP.CT CC2
      /         & ST OFF & CJP & GOTO EXPOVERF
      /         & PREDICT F
      /                 ;2'S COMPLEMENT MULTIPLY CORRECTION
 0109           E.SHIFTOFF & E.REGOFF & M.SDUL
      /         & M.CT C,I & M.CIN ZERO & M.REGOFF
      /         & ST.OFF & CJP & GOTO PUSHRESULT
      /         & PREDICT F
      /                 ;TEST FOR NORMALIZED RESULT
```

```
        /               ;REPEAT SHIFTING UNTIL NORMALIZED OR EXPONENT
        /               ;UNDERFLOW
 0108           FP.MODE0 & E.REG PAS.M & E.CIN ZERO
        /       & M.REG LOAD,N & FP.CT CC1,F & CJP
        /       & GOTO PUSHRESULT & PREDICT T
        /               ;LEAVE FMUL IF NOT EXPONENT UNDERFLOW
        FZERO:
 010C /         FP.MODE0 & E.REG LOAD,M & E.CIN ZERO
        /       & M.REG LOAD,M & M.CIN ZERO & FP.CTOFF
        /       & JMP & GOTO PUSHRESULT & PREDICT OFF
        /               ;RESULT IS A FLOATING ZERO!
        EXPOVERF:
 010D /         FP.MODE0 & E.REG RESET,M & M.REGOFF
        /       & M CIN ZERO & M.CT C,I & CJP
        /       & GOTO SETOVR & PREDICT F
        /               ;CLEAR OVR BIT IN MSR, TEST IF ALREADY NORMALIZED.
 010E           E.SHIFTOFF & E.REG LOAD,UM & E.CIN ZERO
        /       & M.SDUL & M.REG LOAD,M & M.CIN ZERO
        /       & FP.CT CC4 & ST.OFF & CJP
        /       & GOTO $ & PREDICT T
        /               ;COPY I ALSO INTO USR, BECAUSE RESULT OF OPERATION IN
        /               ;FMUL1.SRC, SAME LINE, IS PASSED TO AM2922 CC4 VIA
        /               ;A USR==>YBUS TRANSPORT (SIDE EFFECT OF "RESET,M")
        /               ;REPEAT UNTIL NORMALIZED OR EXPONENT UNDERFLOW
 010F           FP.MODE0 & E.REG RESET,M & E.CIN ZERO
        /       & M.REGOFF & FP.CT CC1 & CJP
        /       & GOTO PUSHRESULT & PREDICT F
        /               ;IF EXPONENT UNDERFLOW THEN CORRECTION WAS A SUCCESS
        SETOVR:
 0110 /         FP.MODE0 & E.REG SET,M & E.CIN ZERO
        /       & M.REGOFF & FP.CTOFF & RTN
        /       & PREDICT OFF
        /               ;IF CORRECTION NOT POSSIBLE THEN EXIT WITH
        /               ;OVR BIT IN MSR SET, NO NEED FOR PUSHING A RESULT!
        PUSHRESULT:
 0111 /         FP.MODE0 & E.REGOFF & E.CIN ZERO
        /       & M.REGOFF & M.CIN ZERO & RTN
        /       & PREDICT OFF
        /               ;PUSH PRODUCT IN R4 ON STACK.

                END
```

FLOATING POINT MULTIPLY (SECOND PART)

```
 0100 XXXXX1XXXXXXX110 0XXXXXXXXX11XXXX XXX11XXX0011111X XXXXX111011XXXXX
      XXXXXXX1XXXXXXXX XXXX
 0101 XXXXX1XXXXXXX110 0XXXXXXXXX11XXXX XXX11XXXX11111X XXXXX111011XXXXX
      XXXXXXX1XXXXXXXX XXXX
 0102 XXXXX1XXXXXXX110 0XXXXXXXXX11XXXX XXX11XXX0011111X XXXXX111011XXXXX
      XXXXXXX1XXXXXXXX XXXX
 0103 XXXXX1X000100100 0XXXXXXXXX110010 111XX1110011111X XX11X00111100010
      00011001XXXXXXXX XXXX
 0104 XXXXX11000100111 XXXXXXXXXX111000 1001011100111110 0110100111100010
      00011001XXXXXXXX XXXX
 0105 XXXXX1X00010011X XXXXXXXXXX110010 101XX1110011111X XX11X00111100010
      00011001XXXXXXXX XXXX
 0106 XXXXX11XXXXXX11X X0111010100O1XXX XXX11XXXXX111110 1101X11001100000
      00101011XXXXXXXX XXXX
 0107 XXXXX11XXXXXX11X X0111010100O1XXX XXX11XXXXX111110 1101X10011100010
      00001111XXXXXXXX XXXX
 0108 XXXXX11XXXXXX11X X0111010100O1XXX XXX11XXX10111110 1010100111100010
      00011011XXXXXXXX XXXX
 0109 XXXXX1XXXXXXX11X X101101010000101 111111110011111X XX11X00111100010
      00100011XXXXXXXX XXXX
 010A XXXXX11000100100 0101101010001XXX XXX11XXX00111110 1110100111100010
      00010100XXXXXXXX XXXX
 010B XXXXX11000100110 0XXXXXXXXX111000 1001011XX111110 0110100111100010
      00100010XXXXXXXX XXXX
 010C XXXXX11000100100 0XXXXXXXXX111000 1001011100111110 1101X00111100010
      00100011XXXXXXXX XXXX
 010D XXXXX1X00100110X XXXXXXXXXX110101 111111110011111X XX11X00111100010
      00100001XXXXXXXX XXXX
 010E XXXXX11000100000 0101101010001000 1001011100111111 0010100111100010
      00011100XXXXXXXX XXXX
 010F XXXXX11001001100 0XXXXXXXXX111XXX XXX11XXXXX111110 0110100111100010
      00100011XXXXXXXX XXXX
 0110 XXXXX11000001100 0XXXXXXXXX111XXX XXX11XXXXX111110 1101X101011XXXXX
      XXXXXXX1XXXXXXXX XXXX
 0111 XXXXX1XXXXXXX110 0XXXXXXXXX11XXXX XXX11XXX0011111X XXXXX101011XXXXX
      XXXXXXX1XXXXXXXX XXXX

 TOTAL PHASE 2 ERRORS =   0
```

```
                ;               FMUL ON STACK MACHINE
                ;               OPERANDS ON STACK
                ;                       EXCHANGE TOP OF STACK WITH ITS
                ;                       NORMALIZED EQUIVALENT
                ;               CALL AS SUBROUTINE

0120            ORG     H#120

        FNORM:
0120    /       FP.MODEO E,M & E.PAS ,,,R2 & E.RAC ,IR,PL
        /       & M.PAS QPT,,, & M.RAC ,IR,
        /               ;POP STACK AND MOVE EXPOHENT PART TO R2
        /               ;AND MANTISSA TO Q REGISTER.
        /               ;INDIRECTION REGISTER (IR) ACTS AS STACKPOINTER.
0121            FP.MODEO ,M & E.ALUOFF & M.SPFNW SLN,,R2,R2
        /       & M.SPFRAC ,PL,PL
        /               ;TEST IF NORMALIZATION NEEDED.
        /               ;R2 IS DUMMY SHIFT COUNT.
0122            FP.MODEO E,M & E.ADD ,R1,R2,R2 & E.RAC PL,PL,PL
        /       & M.SPF SLN,,, & M.SPFRAC PL,PL,PL
        /               ;PERFORM SINGLE LENGTH NORMALIZE AS LONG AS NEEDED.
0123            FP.MODEO E,M & E.PAS ,,R2, & E.RAC ,PL,IR
        /       & M.PAS ,,, & M.RAC ,QR,IR
        /               ;COPY RESULT TO TOS
        FZERO:
0124    /       FP.MODEO & E.READ PL,H#80 & E.PAR ,,,
        /       & E.RAC ,,IR & M.READ PL,H#00,H#0000 & M.PAR ,,,
        /       & M.RAC ,,IR
        /               ;TOS != "FLOATING ZERO"

                END
```

```
 0120 10101000011111IX XXXXXXX010000000 00011011XXXXXXXX XXXXXXXXXXXXXXXX
      X111XXXXXXXXXXXX 000010XX01110XX0 11XX111111111111
 0121 101XXXX11XXXX11X XXXXXXX000011000 00100011XXXXXXXX XXXXXXXXXXXXXXXX
      X111XXXXXX000010 000010XXXXXXXXX0 10101111111111111
 0122 10100110011111IX XXXXXXX000000000 00100011XXXXXXXX XXXXXXXXXXXXXXXX
      X1110000010000I0 0000101001010100 10101111111111111
 0123 10101000011111IX XXXXXXX010000110 00111111XXXXXXXX XXXXXXXXXXXXXXXX
      X111XXXXXX000010 XXXXXXXX01011XX1 01111111111111111
 0124 10101100011111101 000000001100011O 00111110000000000 0000000000000000
      X111XXXXXXXXXXXX XXXXXXXX1XX11XX1 XX11111111111111
```

TOTAL PHASE 2 ERRORS = 0

```
        ;               FNORM ON STACK MACHINE
        ;               OPERAND ON STACK
        ;                       EXCHANGE TOP OF STACK WITH ITS
        ;                       NORMALIZED EQUIVALENT
        ;               CALL AS SUBROUTINE

0120            ORG     H#120

        FNORM:
0120    /       FP.MODE0 & E.REGOFF & E.CIN ZERO
        /       & M.REGCT Z,I,,M & M.CIN ZERO & CJP
        /       & GOTO FZERO & PREDICT F
        /               ;TEST IF MANTISSA OF TOS WAS 0!
0121            E.SHIFTOFF & E.REGOFF & M.SDUL
        /       & M.REGCT C,I,,M & M.CIN ZERO & ST.OFF
        /       & CRTN & PREDICT F
        /               ;QUIT FNORM IF RESULT WAS ALREADY NORMALIZED!
0122            E.SHIFTOFF & E.REGOFF & E.CIN ZERO
        /       & M.SDUL & M.REG PAS.I & M.CIN ZERO
        /       & ST.OFF & FP.CT CC5 & CJP
        /       & GOTO $ & PREDICT T
        /               ;SHIFT UNTIL NORMALIZED
0123            FP.MODE0 & E.REGOFF & E.CIN ZERO
        /       & E.CT OVR,I,F & M.REG LOAD,M & M.CIN ZERO
        /       & CRTN & PREDICT T
        /               ;LEAVE FNORM IF NO EXPONENT UNDERFLOW OCCURED
        FZERO:
0124    /       FP.MODE0 & E.REGOFF & E.CIN ZERO
        /       & M.REG LOAD,M & M.CIN ZERO & FP.CTOFF
        /       & RTN & PREDICT OFF
        /               ;TOS := "FLOATING ZERO"

                END
```

```
 0120 XXXXX11XXXXXX110 OXXXXXXXXX110010 111101110011111X XX11X00111100010
      01001001XXXXXXXX XXXX
 0121 XXXXX11XXXXXX11X X101101010000101 111101110011111X XX11X101011XXXXX
      XXXXXXX1XXXXXXXX XXXX
 0122 XXXXX11XXXXXX110 0101101010001100 010111110011111 0110100111100010
      01000100XXXXXXXX XXXX
 0123 XXXXX10011110110 OXXXXXXXXX111000 100101110011111X XX11X101011XXXXX
      XXXXXXXOXXXXXXXX XXXX
 0124 XXXXX11XXXXXX110 OXXXXXXXXX111000 100101110011110 1101X101011XXXXX
      XXXXXXX1XXXXXXXX XXXX

 TOTAL PHASE 2 ERRORS =   0
```