Processor units

Verhofstadt, P.W.J.

Award date:
1963

TECHNISCHE HOGESCHOOL EINDHOVEN

Afdeling Elektrotechniek

Groep EC-B

"Processor Units"

door

P.W.J. Verhofstadt

Het verslag van een afstudeeronderzoek verricht van september 1962 tot juni 1963.

INHOUD

0. Overzicht

Dit verslag handelt over het afstudeerwerk dat in de tijd van september 1962 tot en met mei 1963 onder leiding van prof.ir.A. Heetman in diens groep aan de Technische Hogeschool te Eindhoven verricht werd.

De opdracht behelsde het onderzoeken van recente ontwikkelingen en mogelijkheden op het gebied van "arithmetic units" voor digitale computers en het eventueel toepassen daarvan in een ontwerp.

Gezien deze opdracht is het verslag in twee duidelijk onderscheiden delen te splitsen.
Het eerste gedeelte A bevat de resultaten van het literatuuronderzoek en het tweede gedeelte B de beschrijving van een gedeeltelijk ontwerp van een "processor-unit".
Daarbij is hoofdzakelijk gebruik gemaakt van methodes die in gedeelte A beschreven zijn. De betreffende methodes zijn in A dan ook wat uitgebreider behandeld dan de meeste andere.
Tot slot is ook een uitgebreide geselecteerde literatuurlijst aan het verslag toegevoegd.

1. Inleiding

1.0. Algemeen

De moderne digitale computers zijn meestal te onderschieden in twee klassen. De grote, snelle machines zoals de IBM 7030; IBM 7090; RCA 604; UNIVAC LARC; Burroughs B5000; enz. Anderzijds de computers voor minder grote taken, waarbij bovendien meestal geen grote snelheid vereist is, zoals: LGP 30; IBM 1620; CDC 160; Bendix B15; enz.
Het doel van het hier beschreven onderzoek is nu de moderne ontwikkelingen op computer gebied eens te verzamelen en te bestuderen. Waarna dan, gebruik makende van moderne technieken, een zeer snelle, flexibele en universele rekenkundige eenheid ontworpen zal worden. Deze zal in staat moeten zijn ingewikkelde berekeningen uit te voeren en zal bovendien op vergaande autonome basis logische operaties moeten kunnen uitvoeren.
Bij dit alles zal in ruime mate gebruik gemaakt worden van nieuwe ontwikkelingen op materiaalgebied. Het accent zal echter voornamelijk liggen op het invoeren van logische verbeteringen waarbij ook minder bekende technieken toegepast zullen worden.
Naast de intussen reeds alom bekende methodes zoals "indirect addressing", "relative addressing", "field addressing", etc. zijn er een aantal technieken en procedé's ontwikkeld, die hoofdzakelijk verband houden met de rekenkundige eenheid. (L 1.1.; L 1.2.; L 1.3.; L 1.4.; L 1.5.; L 1.6.;). Het lijkt zinvol, zij het niet alle, hier een aantal van deze mogelijkheden kort aan te geven.

1.1. "Polish Notation"

Hierbij wordt afgeweken van de voor ons normale volgorde van notatie. Een voorbeeld:
y = (p + q + t) (m - n) / z wordt in deze notatie:
ypq + t + mn - . z / =
Men heeft hier dus de vergelijking y = x en schrijft deze yx =. In het gegeven geval geldt: x = a/z en men schrijft az /.
Verder is a = b.c. wat wordt tot bc. met b = d + t wat wordt dt + met d = p + q of pq + en c = m - n of mn -
Resumerend: ypq + t + mn - . z / =.

De operaties en de operanden hebben hier dus andere plaatsen dan in de ons vertrouwde notatie. Op deze wijze wordt een effectieve organisatie van de informatie verwerking bereikt (L 1.7.).

1.2. "Time-Sharing" en "Multiprogramming"

De ontwikkeling van de informatieverwerkende apparatuur is niet voor alle delen hiervan even snel gegaan. Sommige aspecten hebben minder aandacht gekregen dan andere. Ook zijn sommige processen reeds van nature trager dan andere. Dit alles heeft ertoe geleid dat b.v. in de huidige rekenmachines het rekenen veel sneller geschiedt dan de toevoer vanuit het geheugen. En dit laatste is vaak nog weer sneller dan de externe in- en uitvoer. Het ligt nu voor de hand om te trachten de snelle rekeneenheid te laten benutten door meer "gebruikers" tegelijkertijd. Hierbij verdeelt de rekeneenheid zijn aandacht en tijd over diverse problemen. Ook kan men invoer en uitvoer parallel laten werken, meerdere units tegelijkertijd laten werken en ook meerdere geheugens "in concurrency" gebruiken. Dat bij dit alles een straffe regelende hand noodzakelijk is zal als stringente eis iedereen duidelijk zijn. De problemen, die hiermede samenhangen zijn dan ook nog lang niet alle overwonnen. De mogelijkheid om in bepaalde gevallen prioriteit in te voeren voor bepaalde operaties boven andere opent weer nieuwe mogelijkheden en leidt tot zeer flexibele bewerkingen. (L 1.8.; L 1.9.; L 1.10.; L 1.11.; L 1.12.; L 1.13.; L 1.14.; L 1.15.).

1.3. "The Polymorphic Principle" en "Modularity"

Een computer is te verdelen in een aantal m.b.t. functie en plaats duidelijk te onderscheiden units. Het samenspel tussen deze units is van essentieel belang. In de meest gangbare situaties zijn aan de bekende groepering van de units een aantal nadelen verbonden. Deze kunnen b.v. zijn,het niet met kleine stappen uitbreidbare arsenaal van units, de onmogelijkheid van het verder werken bij storing van een van de units, enz. Ten einde hieraan tegemoet te komen, zijn een aantal

systemen voorgesteld, waarbij een grotere flexibiliteit in de opbouw van het systeem is verkregen. Het samenspel van een, binnen zekere grenzen willekeurig uitbreidbaar, aantal units wordt hierbij geregeld via een meer passieve schakelunit, die de gewenste configuratie van units tot stand brengt. Ook meerdere in elkaar geweven systemen zijn hierbij mogelijk. Vooral voor toepassingen waarbij aanpasbaarheid en universaliteit gewenst zijn biedt deze organisatie van geheugens, invoer, uitvoer, rekeneenheden etc. geheel nieuwe perspectieven . (L 1.16.; L 1.17.; L. 1.18).

1.4. "Hybrid" methoden

Het antwoord op de vraag: Een digitale of een analoge machine, is lang niet altijd direct te geven. Beide hebben hun specifieke toepassingsgebieden, maar daarnaast is er nog een overgangsgebied, waar niet éénduidig vastligt, welk systeem de voorkeur verdient. Een recente ontwikkeling die hier in een gedeelte van de grensgevallen een oplossing kan brengen is de zgn. "hybrid computer". Hiermede wordt die apparatuur bedoeld, die zowel van analoge als van digitale signalen gebruik maakt om de informatie te verwerken.
Men maakt hierbij van twee verschillende technieken gebruik:

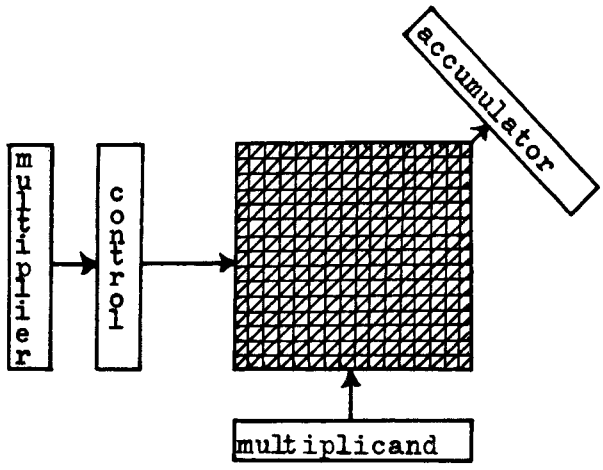
a. Een gedeelte van het signaal (meestal het meest significante deel) wordt in binaire vorm gerepresenteerd. Het minst significante deel daarnaast in analoge vorm. Het signaal bestaat dus uit twee signalen, waarvoor ook twee informatiekanalen aanwezig zijn. Tussen deze twee kanalen moet echter ook een zekere wisselwerking mogelijk zijn.
   Er kan nl. analoog-digitaal conversie nodig zijn (overflow, etc.) en ook omgekeerd. (L 1.19.).

b. De analoge signalen worden gestuurd door de digitale signalen (multiplexing). Het aanwezige digitale signaal bepaalt dus de weg, die door het analoge signaal gevolgd wordt. De digitale informatie bedient dus schakelaars die de analoge informatie al of niet doorlaten. (L 1.20.; L 1.21.).
   Door het gebruik van deze technieken kan men tegen een naar verhouding lage prijs een redelijk snelle en voldoende nauwkeurige rekenapparatuur verkrijgen. Waarbij tevens een aantal gevallen de meer aangepaste oplossing krijgen, wat bij het

gebruik van zuiver digitale of zuiver analoge informatie-verwerking moeilijk realiseerbaar zou zijn.

1.5. "Shiftrix" logica

Langs velerlei wegen wordt getracht vooral de tijd benodigd voor het vermenigvuldigen te verkorten. Een van de mogelijkheden hiertoe is het toepassen van een zgn. shiftrix (fig. 1). Hierbij maakt men gebruik van een schakelmatrix. De horizontale input wordt gevormd door de multiplicand bits, terwijl de verticale input via een control-unit gestuurd wordt door de multiplier bits. De organisatie is verder zo, dat de "diagonale" signalen in de accumulator gevoerd worden. De verbindingen zijn hierbij zodanig uitgevoerd, dat een activeren van de matrix (waarin de multiplicand) in de bovenste rij (door de meest significante multiplier-bit) ten gevolge heeft dat ook de juiste informatie in de accumulator wordt geschreven. Ditzelfde geldt ook voor alle andere rijen. Het grote voordeel is nu, dat het stuursignaal, dat boven begint en zo de rijen afloopt naar beneden, eenvoudig die rijen overslaat, waarvan het overeenkomstige multiplier-bit een nul is. Er gaat hier dus geen tijd verloren met het schuiven, enz. Hierdoor kan een aanzienlijke tijdwinst verkregen worden. Ook voor de overige rekenkundige operaties biedt het systeem gunstige mogelijkheden. Een nadeel is echter het grote aantal componenten dat nodig is voor de matrix (L 1.22.).



Figuur 1.

1.6. Microprogrammering

Als we een probleem op een digitale rekenmachine willen behandelen schrijven we een daartoe geschikt programma.
Dit programma bestaat dan uit een aantal opeenvolgende instructies, die de machine achtereenvolgens uitvoert.

Elk van deze instructies echter bestaat uit een aantal onderscheiden handelingen, die verricht moeten worden. Elke instructie vereist dus een duidelijk bepaalde reeks van operaties. In de computer nu zorgt de zgn. "sequencing unit" ervoor dat dit op de juiste manier gebeurt. Elke instructie bestaat dus als het ware uit een subprogramma (het microprogramma) van micro-operaties.
We zien dus het samenspel: computer, programma, macroinstructies, op verkleinde schaal terugkeren in:"sequencing unit", microprogramma, micro-operaties. Evenals in het macroprogramma kunnen in het micro-programma sommige operaties conditioneel zijn. Men kan nu i.p.v. een "sequencing unit" te gebruiken ook anders te werk gaan. Met een "sequencing unit" zit men immers gebonden aan een star ingebouwd microprogramma, dat voor alle mogelijke optredende situaties een oplossing moet weten. Het gevolg is dat het verloop van het microprogramma verre van optimaal en niet flexibel is.
Een meer flexibele en universele oplossing verkrijgt men als men de "processor unit" zodanig inricht, dat deze op bevel van buiten af een aantal zeer elementaire operaties kan uitvoeren in een volgorde die van buitenaf bepaald wordt. Ten behoeve van conditionele stappen e.d. dient de processor dan ook terugmeldingen te verrichten aan het apparaat dat hem stuurt.

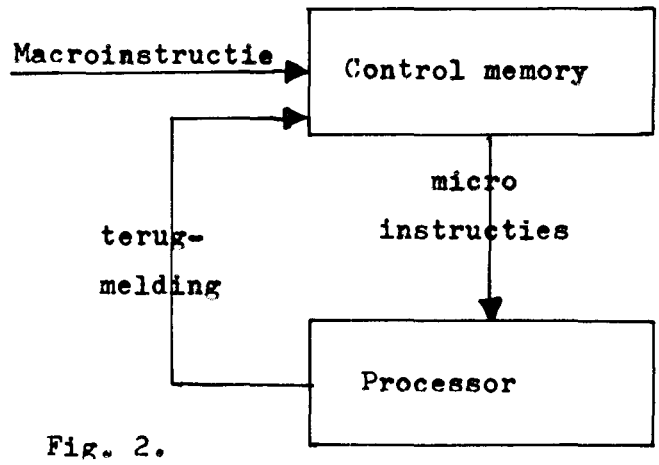


Fig. 2.

Het microprogramma is hierbij dus opgebouwd uit zeer elementaire micro-operaties, zoals: shift, transfer, clear, load register, load flip-flop, enz.
De gang van zaken is nu als volgt.
Een macro instructie wordt naar het "control memory" (zie fig.2) geleid. Hier wordt deze gedecodeerd en de instructie opgesplitst in de microinstructie. Vervolgens wordt de eerste microinstructie aan de processor toegevoerd en de eerste micro-operatie uitgevoerd.

Hierbij ontstaat eventueel een conditionele terugmelding. Hierna wordt de tweede microinstructie toegevoerd en door de processor uitgevoerd. Is het gehele microprogramma afgewerkt, dan wordt een signaal afgegeven en kan de volgende macroinstructie aan het "control memory" worden toegevoerd. Men kan hierbij dus gebruik maken van de mogelijkheid om veranderingen aan te brengen in de translatie d.m.v. het "control memory". Op deze wijze kan dus een betere aanpassing van het operatie arsenaal aan het specifieke probleem worden verkregen. Dit leidt tot grote flexibiliteit en meestal ook tot besparing aan "hardware". Een nadeel is echter dat de "over-all" snelheid meestal iets zal dalen t.g.v. de communicatie tussen processor en "control memory", die meer tijd kost dan in de starre uitvoering. Anderzijds zal echter ook meestal t.g.v. de flexibiliteitswinst weer een verhoging van snelheid kunnen optreden, zodat een éénduidige uitspraak moeilijk te doen zal zijn.
Het "control memory" kan op velerlei manieren georganiseerd zijn. Er is echter een onderscheid te maken tussen twee hoofdtypen:

a. Een kerngeheugen of een matrix die uitwisselbaar is. Hier kan men dus, analoog aan de situatie bij bijv. een ponskaarteninstallatie, de programma's verwisselen.

b. Een voor de operator of programmeur toegankelijk geheugen. De functie van het "control memory" wordt hier dus echt geprogrammeerd. De programmeur schrijft dus telkens ook de betreffende microprogramma instructies in. Dit is dus uiterst flexibel.
Soms wordt onder „microprogramming" ook alleen deze laatste aanpak van zaken verstaan.
In alle gevallen doet het er echter voor de processor niet zo zeer toe op welke wijze het "control memory" georganiseerd is. De opbouw van de processor kan dus geschieden vanuit het standpunt dat er van buitenuit steeds de nodige microinstructies zullen binnenkomen. Wel moeten natuurlijk ook de nodige faciliteiten voor terugmelding aanwezig zijn.
(L 1.23.; L 1.24.; L 1.25.; L 1.26.; L 1.27.; L 1.28.; L 1.29.).

1.7. "Built-in Logic"

In de meeste digitale machines worden de elementaire rekenkundige bewerkingen uitgevoerd door een samenspel van adders, subtractor, register, etc.
Men kan echter ook gebruik maken van zgn. "Table lookup" methodes. Hierbij zijn a.h.w. opteltabellen en de tafels van vermenigvuldiging in het geheugen opgeborgen. Het optellen of vermenigvuldigen van bijv. p en q heeft dan het oproepen van een bepaalde geheugenplaats ten gevolge.
Op deze plaats staat dan het resultaat p + q of p x q opgeborgen. Machines die van dit soort technieken gebruik maken zijn bijv. IBM 1620 en IBM Harvest (L 1.13.; L 1.30.).
Een andere ontwikkeling is het inbouwen van subroutines voor functieberekeningen. Vooral bij sommige "special purpose" computers kan dit voordelen brengen. Hierbij staan dus naast de gangbare instructietypes zoals add, subtract, transfer, enz. ook instructies ter beschikking als sine, cosine, square root, enz.
Dit versnelt natuurlijk het rekenen aanzienlijk, ook zal het programmeren eenvoudiger worden. Hiertegenover staat uiteraard het nadeel van de veel grotere hoeveelheid materiaal die nodig is. (L 1.31.).

1.8. Andere logische technieken

De meeste bestaande installaties hebben logica opgebouwd uit AND, OR en NOT-schakelingen. Recente ontwikkelingen maken echter met vrucht gebruik van NOR ($\bar{A}.\bar{B}$), NAND ($\bar{A} + \bar{B}$), SHEFFER-STROKE ($A.B + \bar{A}.\bar{B}$), enz. -schakelingen.
Vooral in de zgn. autosynchrone circuits (schakelingen zonder vaste klokfrequentie) wordt deze logica veelvuldig toegepast. Hiermede wordt bereikt dat de weg (wat het aantal logische elementen achter elkaar betreft), die de informatie moet afleggen minimaal is. Dit is, vooral i.v.m. het feit, dat de voortplantingssnelheid langzamerhand een limiet gaat vormen, van groot belang. De snelheidslimiet kan op deze manier weer aanzienlijk verschoven worden. Een bijkomend voordeel is dat de veelsoortigheid van benodigde circuits teruggebracht kan worden tot kleinere aantallen.

Als nadeel dient echter vermeld te worden, dat de totale aantallen benodigde circuits in het algemeen groter zullen zijn. (L 1.32.; L 1.33.; L 1.34.; L 0.6.).

2. Codes

2.0. Algemeen

Als iemand U zegt: "Er lopen twaalf koeien in de wei ginds achter de heuvel. Wilt U voor elk van hun een touw neenemen? ", dan weet U precies hoeveel touwen U moet meenemen. Twaalf namelijk. U weet echter niet alleen dat U twaalf stuks moet nemen, U weet ook dat er met het woord twaalf bedoeld wordt, dat wat we symbolisch schrijven als 12 en U weet dat U twaalf touwen hebt als U voor elk van de volgende nullen één touw meeneemt 0 0 0 0 0 0 0 0 0 0 0 0 (vooropgesteld dat U weet wat één is !). Wanneer U echter op een fictieve planeet in een ander zonnestelsel bent geland en U zegt: "Ik wil graag 12 bewoners van Uw planeet meenemen", hebt U echter kans dat U er tot Uw verbazing maar vijf meekrijgt. Als U dan even de U op de expeditie vergezellende wiskundige assistent raadpleegt, dan zal deze U waarschijnlijk vertellen, dat men klaarblijkelijk op de vreemde planeet het drietallig stelsel hanteert. Hieruit blijkt dus wel, dat men,om over hoeveelheden te kunnen praten, eerst moet weten in welk "stelsel" men zich uitdrukt !
Sinds vele eeuwen reeds gebruiken wij het tientallig of decimaal stelsel (verband houdende met onze tien vingers !?), alhoewel sommige mathematen beweren dat een twaalftallig stelsel efficienter zou zijn.
We zijn dus vrij om, als we iets nieuws gaan doen zoals bijv. rekenmachines bouwen, in een ander stelsel te gaan werken. We zullen dit echter in het algemeen alleen maar doen als we denken door deze nieuwigheid voordelen te kunnen behalen. In de loop der jaren zijn voor digitale informatieverwerkende systemen diverse stelsels of codes ontwikkeld en gebruikt. We zullen een aantal van de meest bekende hier kort de revue laten passeren.

2.1. Zuivere Codes

Bij deze codes stelt men een getal voor door een reeks symbolen: $A_n\ A_{n-1} \ldots\ldots\ldots A_2\ A_1\ A_0$, waarbij het getal de waarde: $W = A_n g^n + A_{n-1} \cdot g^{n-1} + \ldots\ldots + A_2 g^2 + A_1 g^1 + A_0 \cdot g^0$ heeft.

In deze representatie kunnen de symbolen A_i de waarden 0, 1, ... t/m g-1 aannemen. We noemen g het grondgetal van het stelsel. Tot dit soort codes behoort ook ons dagelijks decimale stelsel. De talstelsels die van belang zijn voor rekenapparatuur en die tot deze groep behoren zijn: het decimale stelsel, het binaire stelsel en soms het octonale stelsel. Hierbij heeft g respectivelijk de waarden 10, 2 en 8.

a. Decimale codes. Dit is het stelsel dat ons dagelijkse leven beheerst. Een voordeel van het gebruik van deze code in rekenapparatuur zou dus zijn, dat ze direct aansluit aan het algemeen gebruik. Een meestal zwaarwegend nadeel is echter het feit dat in de natuur slechts zeer weinig elementen aanwijsbaar zijn die op een tientallige basis werken. Met uitzondering van het gebruik in telbuizen voor geigertellers e.d., vindt het zuivere tientallige stelsel in de moderne rekenapparatuur dan ook praktisch geen toepassing.

b. Octonale code. Aan dit stelsel kleven dezelfde nadelen als aan het decimale stelsel. Bovendien vervalt hier nog het voordeel van het aangepast zijn aan het dagelijks gebruik. Een voordeel is echter dat ze gemakkelijk converteerbaar is in een binaire code omdat 8 een macht is van 2. Tegenover de zuiver binaire code heeft ze ook het voordeel dat minder lange reeksen van symbolen nodig zijn. Dit bevordert het gemak van de communicatie. Toch wordt de code slechts zeer weinig gebruikt. (L 2.1.).

c. Binaire code. Hier heeft g de waarde van 2 en de mogelijke symbolen zijn 0 en 1. Het voordeel is hier dus, dat de code aangepast is aan de techniek. Immers ringkernen, flip-flop, schakelaars, etc. hebben alle twee mogelijke toestanden, waaraan de waarden 0 en 1 toegekend kunnen worden. Een nadeel is uiteraard de moeite die het de "normale" gebruiker kost om binaire resultaten te interpreteren. Daarom zal in bijna alle gevallen conversie onvermijdelijk zijn.

2.2. Gemengde codes

Bij deze codes worden twee of meer talstelsels in een code verwerkt. Het getal zeven en dertig is in decimale code 37, in binaire code 100101.

Men kan nu echter ook een code formeren die elk decimaal cijfer apart in een binaire code weergeeft en deze reeks van binaire getallen dan op decimale basis achter elkaar zetten. Ons voorbeeld: 3 wordt 0011 en 7 0111. Een dan dus 37 is 0011 0111.
Men noemt dit een binair gecodeerde decimale getal representatie. Hetzelfde kan men uiteraard ook voor octonale getallen doen.
Daar het maximale decimale getal, dat men binair moet weergeven 9 is, heeft men per decimaal cijfer dus minimaal 4 binaire tekens of bits nodig. Vaak zijn het er echter meer, omdat men teken-bits, check-bits, e.d. wil weergeven. Ook wil men soms alphamerieke gegevens binair coderen. Dit heeft tot gevolg gehad dat een grote scala van binair gecodeerde decimale codes is ontstaan.
Een aantal van de meest voorkomende zijn:

Excess-three code 5-4-2-1 code Gray code (binair gereflecteerde code) 8-4-2-1- code 2-4-2-1- code (Aiken code)	4-bit numeriek
teken -8-4-2-1 code check -8-4-2-1 code	5-bit numeriek
check-teken -8-4-2-1 code komma-check -8-4-2-1 code	6-bit numeriek

Biquinaire codes (5, 6 of 7 bits)

7-bit alphamerieke codes (inclusief parity-bit)

(zie ook L 2.2. en L 2.3.). Bovenstaande zijn allemaal binair gecodeerde decimale codes, die elk voor zich gunstig of ongunstig zijn wat betreft: materiaalhoeveelheid, check mogelijkheden, carry problemen, inversie mogelijkheden, al of niet gespiegeld zijn, "constant weight positions", enz. enz.
Ook voor achttallige posities kan men binaire coderingen invoeren. Daar acht echter een macht van twee is, kan men er hier met meer vrucht gebruik van maken.

Er zijn nu bijv. maar 3 plaatsen per octonaal cijfer nodig. Deze 3 plaatsen worden nu ook ten volle gebruikt. Immers we hebben 2^3 mogelijke combinaties, die alle gebruikt worden. Bij de decimaal-binaire codes hebben we 4 plaatsen, wat $2^4 = 16$ mogelijkheden oplevert, waarvan er slechts 10 gebruikt worden. Dit is een van de redenen waarom men bijv. in de IBM 7090 deze code gebruikt (L 2.4.).

2.3. Speciale codes

In een aantal gevallen worden speciale coderingen toegepast. Vooral in de transmissie en gedurende de laatste tijd ook in speciale rekenschakelingen en processystemen.

a. Transmissie codes.

Een tweetal veel gebruikte codes zijn de 2 uit 5 code en de 3 uit 7 code. De check mogelijkheden zijn hierbij vrij gunstig. Het is echter erg omslachtig om met getallen in deze code rekenbewerkingen uit te voeren. Ze worden dan ook alleen als codes voor het overbrengen van informatie gebruikt (L 2.5.).

b. Codes met negatieve basis.

In de tot hier aangegeven zuivere codes is g altijd positief genomen. Het is echter ook mogelijk $g < 0$ te nemen. We hadden: $w = a_n g^n + a_{n-1} \cdot g^{n-1} + \ldots\ldots + a_1 \cdot g + a_0 \cdot g^0$. Nemen we nu $g = -2$ en bekijken 101101 (dus $a_0 = 1$; $a_1 = 0$; $a_2 = 1$; $a_3 = 1$; $a_4 = 0$; $a_5 = 1$). Dit heeft nu de waarde $w = 1.(-2)^5 + 0.(-2)^4 + 1.(-2)^3 + 1.(-2)^2 + 0.(-2)^1 + 1.(-2)^0 = -32 + 0 - 8 + 4 + 0 + 1 = -35$. Op dezelfde manier vinden we: 10011 = + 15. We zien dus dat we, zonder dat we het teken apart aangeven, zowel positieve als negatieve getallen kunnen weergeven. Het teken kan direct bepaald worden uit het aantal cijfers. Positieve getallen hebben een oneven aantal cijfers, negatieve een even aantal. (Immers het meest significante bit bepaalt het teken !). Voor dit soort getallen kunnen ook rekenregels opgesteld worden. Er is nog geen direkte toepassing van codes met een negatieve basis bekend, alhoewel voor speciale doeleinden zeer zeker gebruik gemaakt zal kunnen worden van de specifieke eigenschappen (L 2.6.).

c. "Signed-Digit Numbers"

Ook bij deze codes is een aparte tekeninformatie overbodig. Ieder cijfer bevat zijn eigen tekeninformatie. Vandaar ook de naam van de code. Het grote voordeel van deze code is echter het geheel weg blijven van kettingoverdracht bij het optellen. Er kan dus zeer snel mee gerekend worden in parallel schakelingen. Er is echter een ruime mate van "redundancy" aanwezig, zodat de ruimtebenutting niet erg efficiënt is. Het zijn verder talstelsels die positioneel zijn, d.w.z. aan een bepaalde plaats in het getal kan steeds een bepaalde waarde worden toegekend. Het criterium van de code zit in het feit dat bij een grondtal g de cijfers de waarde a kunnen aannemen zodanig dat $g + 2 \leqslant a \leqslant 2g - 1$, terwijl in de gebruikelijke zuivere codes a de waarden aan kan nemen $0 \leqslant a \leqslant g - 1$. De beperking bij "signed digit codes" is wel dat $g \geqslant 3$. Alhoewel er reeds apparatuur bestaat waarin de bovenstaande code wordt toegepast, is ze toch alleen voor speciale toepassingen bestemd (L 2.7.; L 2.8.).

d. "Residue-Number System"

Het residu van een getal met betrekking tot een modulus m is de rest die overblijft nadat van het betreffende getal het maximale veelvoud van m is afgetrokken zodanig dat het resultaat positief is.

Dus het residu van 10 m.b.t. de modulus 3 is 1.

Immers $10 - 3 \times 3 = 1$

We kunnen nu een getal aangeven door het residu en de modulus aan te geven. Hiermede is het getal echter slechts op een veelvoud van de modulus na bepaald. Om een éénduidige weergave te bereiken representeren we het getal dubbel. Dit wil zeggen, we geven voor het getal vier kenmerken. En wel residu r_1 en modulus m_1 en verder residu r_2 en modulus m_2.

Hierbij dragen we er zorg voor dat m_1 en m_2 priemgetallen zijn. In de praktijk nemen we dan meestal nog m_1 en m_2 vast zodat, om een getal aan te geven, twee cijfers (r_1 en r_2) voldoende zijn. Als voorbeeld bij $m_1 = 2$ en $m_2 = 5$ geldt:

$7 = 1$ Mod 2 en $7 = 2$ Mod 5. Of 7 = 1 2. Binnen het gebied dat bepaald wordt door het produkt $m_1 \times m_2$ zijn alle getallen nu éénduidig vastgelegd.

Dit gebied noemen we het bereik van de isomorphe betrekking. We kunnen nu ook met getallen in deze representatie gaan rekenen. Het voordeel dat hierbij in het oog springt is het feit dat voor een vermenigvuldiging slechts dezelfde tijd nodig is als voor een optelling (we behoeven slechts de residuen te vermenigvuldigen !). Voor een nadere bestudering zij gewezen naar de literatuur. Opgemerkt dient echter nog te worden, dat de grote problemen liggen op het gebied van de carry detectie en het deelproces. Ook geldt dit in zekere mate voor het aftrekken (L 2.9.; L 2.10.).

e. Gemodificeerde Gray Codes.

Het grote voordeel van de Gray codes is het feit dat bij de overgang van het ene getal op het naastbij hogere of lagere er slechts een bit verandert. Deze eigenschap wordt met succes toegepast in de analoog-digitaal conversie. Hierdoor vermijdt men immers het probleem van het precies gelijktijdig laten overgaan van de bits. Hier verandert er slechts een bit en het mechanische probleem van precies gelijke overgangen vervalt. Een nadeel is echter het feit dat men de Gray-code slechts zeer moeilijk rekenkundige operaties kan uitvoeren. Een conversie van de Gray-code in een andere is dus noodzakelijk. Dit kan vooral bij serie verwerking bezwaarlijk zijn. Men kan nu een modificatie aanbrengen door een extra bit aan het einde van het codewoord toe te voegen. Dit bit is in wezen een paritybit. Dus bijv. 4 = 0110 (Gray) wordt 4 = 01100.
Hierdoor wordt zowel het rekenen gemakkelijker als ook de detectie van enkele fouten mogelijk.
Het rekenen is echter nog steeds zodanig ingewikkeld, dat een aanzienlijke hoeveelheid materiaal meer nodig is dan bij het gebruik van een binaire code. Tech zal in bepaalde eenvoudige apparatuur de toepassing van deze code voordelen kunnen bieden (L 2.11.; L 2.12.; L 2.13.; L 2.14.).

f. Er zijn nog diverse aspecten aan deze coderingszaak.
Voor verdere bestudering van enkele verdere gezichtspunten zij verwezen naar de volgende literatuur (L 2.15.; L 2.16.; L 2.17.).

2.4. Enkele opmerkingen

Niettegenstaande het grote aantal codes dat boven is aangestipt komen in het algemeen voor grotere "general purpose computers" slechts de binaire of de binair gecodeerde decimale code in aanmerking. Waarbij opgemerkt dient te worden, dat onder decimale representatie in de literatuur vaak de binair gecodeerde representatie wordt verstaan. We zullen ook hier over binair enerzijds en "decimaal" anderzijds spreken.
In het hier te beschrijven ontwerp zal van de binaire representatie gebruik gemaakt worden. Tegenover de decimale voorstelling heeft dit verschillende voordelen, waarvan we hier de meest belangrijke zullen aangeven.

Voordelen: a. Minder materiaal nodig. Omdat de reken- en bewerkingsmethoden meer direct zijn, kan een aanzienlijke hoeveelheid materiaal bespaard worden.

b. Sneller. De bewerkingen zijn directer en eenvoudiger.

c. Kortere woordlengte. In de decimale representatie zijn per decimaal cijfer minimaal 4 bits en meestal 5 bits nodig. Voor een getal van 10 decimale plaatsen is dit 10 x 5 = 50 bits. Om 10^{10} in binaire vorm te representeren heeft men echter slechts 34 + teken bit + check bit = 36 bits nodig. Dit betekent een besparing van 35 % in ruimte. Dit betekent minder kostbare geheugenruimte, minder materiaal voor registers en dergelijke, minder tijd nodig voor de carry propagatie, enz.

Het belangrijkste nadeel, dat aan de binaire code kleeft, is de moeite die het de gebruiker kost om de resulaten direct te interpreteren. Het gevolg is dan vaak een meer ingewikkelde conversie dan bij het omzetten van zuiver decimale in binair gecodeerde decimale getallen.
Niettegenstaande dit werd besloten de zuivere binaire code te gebruiken. Dit vooral gezien de vereiste snelheid en flexibiliteit (L 2.18.).

3. Carry Propagatie

3.1. Algemeen

In de meeste computers kunnen alle rekenkundige bewerkingen teruggebracht worden tot optellen. Delen en vermenigvuldigen bestaan dan meestal uit series optellingen of aftrekkingen, waarbij een aftrekking vaak teruggebracht wordt tot een complementaire optelling. De snelheid van een rekenmachine zal dus als een van de belangrijkste grenzen de snelheid van een optelling hebben. Het is dus zaak deze opteltijd te minimaliseren.
Nu is (zoals bekend verondersteld wordt) bij parallelmachines de overdracht die uit een optelling van twee bits kan ontstaan in eerste instantie de snelheidsbeperkende factor.
Immers uit het optellen van twee bits kan niet eerder een correct resultaat komen voordat ook de uit de optelling van de vorige (minder significante) bits eventueel ontstane overdracht (carry) in rekening is gebracht.
Het probleem is nu, dat bij de optelling in de tweede trap t.g.v. de eerste carry weer een nieuwe carry kan ontstaan, enz.
De carry plant zich dus als het ware voort langs de trappen totdat er ergens twee bits zijn, die beide nul zijn. Hier eindigt dan de voorplantingsketting. Het ongunstigste geval, dat hierbij kan optreden, is propagatie over de gehele woordlengte.
Dit is bijv. het geval als we moeten optellen: 111111 + 000001 = = 1000000. De tijd die voor deze maximale propagatie nodig is zal dus in eerste instantie de snelheidsgrens aangeven. Methodes om de propagatietijd te bekorten zijn dan ook gedurende de laatste tien jaren in groten getale ontwikkeld. We zullen hier een kort overzicht geven van een aantal methodes en een vergelijking proberen te maken. Hierbij is in hoofdzaak gebruik gemaakt van het onderzoek verricht door Lehman (L 3.1.). Verdere algemene literatuur is: (L 3.2.; L 3.3.).
We zullen voornamelijk gebruik maken van de Engelse benamingen van de technieken omdat deze algemeen verbreid zijn en het minste aanleiding geven tot verwarring.

3.2. "Carry Storage"

Bij deze aanpak van zaken wordt bij de optelling van elke twee bits zoals gebruikelijk een som-bit en een carry-bit gevormd. Nu worden echter som-bits en carry-bits elk in een apart register opgeborgen.De som-bits in het gebruikelijke accumulatorregister en de carry-bits in een extra register, het zgn. carrystorage register. Na afloop van de optelling (die hier dus slechts zo lang duurt als het optellen van twee bits) hebben we dus twee getallen die samen het resultaat vormen.
We hebben dan een zgn. redundante getalrepresentatie. Deze representatie nemen we dus gedurende de gehele bewerkingstijd mee. Aan het einde kunnen we dan som- en carry-bits weer bij elkaar optellen. (assimilatie)
Dit levert uiteraard voordeel op bij herhaald (accumulatief) optellen dus ook bij vermenigvuldigen. De nadelen van het systeem zijn, dat men als men het teken van het resultaat wil bepalen of het optreden van overflow wil constateren, men de carry-bits en de som-bits moet optellen. De nieuwere "carry storage" technieken hebben echter veelal weer een oplossing voor deze problemen gevonden, zoals "radix four" optelling, speciale carry generator, enz. (L 3.1.; L 3.4.).

3.3. "Carry Detection"

In de meeste carry propagatie circuits moet men de toegemeten tijd zodanig kiezen dat ook in het ongunstigste geval de goede werking van het systeem nog verzekerd is. Dit betekent dat in veel gevallen er meer tijd beschikbaar is dan er nodig is. Wat onnodig tijdverlies impliceert. Dit zou men kunnen voorkomen door een signaal te geven als de optelling klaar is. Dit betekent dat men van trap tot trap over de gehele woordlengte een signaal moet doorgeven dat de betreffende trap "klaar" is. Men doet dit door van trap tot trap een carry signaal (als er een carry is) of een non-carry signaal (als er geen is) door te geven. Zo gauw nu aan de signaal uitgang van de meest significante trap een dergelijk signaal verschijnt, weet men dat de optelling beëindigd is. De volgende bewerking kan dan beginnen. De opteltijd is hierbij dus niet vast, maar afhankelijk van de sommanden. Deze techniek is dus bij uitstek geschikt voor asynchrone computers.

Ook hier treedt weer als nadeel op het feit dat er een aanzienlijke hoeveelheid extra materiaal vereist is. Bovendien zal in vele gevallen aan synchrone technieken de voorkeur gegeven worden (L 3.1.; L 3.5.; L 3.6.; L 3.7.).

3.4. "Simultaneous Carry"

De carry van elke trap is een functie van de twee sommand-bits van die trap en de carry van de vorige trap. Algemeen geldt:
$C_{n+1} = a_n \cdot b_n + (a_n + b_n) \cdot C_n$.
Waarbij: $C_n = a_{n-1} \cdot b_{n-1} + (a_{n-1} + b_{n-1}) \cdot C_{n-1}$, en C_{n-1} = ... enz. Uiteindelijk dan: $C_1 = a_0 \cdot b_0 + (a_0 + b_0) \cdot C_0$ en $C_0 = 0$, dus: $C_1 = a_0 \cdot b_0$. Op deze manier kunnen we dus de C_{n+1} uitdrukken in: a_0 ..... a_n en b_0 ..... b_n.
Als we nu een logisch circuit bouwen dat samengesteld is volgens: $C_{n+1} = F(a_0, ..., a_n , b_0 , ... , b_n)$ dan behoeven we dus niet te wachten op de carry van de andere trappen, maar kunnen zo gauw de a- en b- bits aanwezig zijn de C_{n+1} vormen door de logica (z.g. volledige logica). Dit geeft uiteraard een zeer grote versnelling van het carry propagatie proces. Een onoverkomelijk nadeel is dat bij een woordlengte van enige betekenis de benodigde hoeveelheid materiaal enorm zou zijn. Daarom wordt de techniek in deze vorm dan ook nooit toegepast. Een tussen oplossing werd gevonden door de woordlengte op te splitsen in een aantal delen van 5 à 10 bits en voor elk van de delen de volledige-logica techniek toepassen.
Voor de schakel tussen de woorddelen kan dan van een andere techniek gebruik gemaakt worden.
Dit geeft een aanzienlijke besparing op de materiaalkosten en zal toch nog een redelijke tijdwinst opleveren. Deze laatste methode noemt men wel complete logica per groep.
(L 3.1.; L 3.8.; L 3.9.; L 3.10.).

3.5. "Pyramid Carry"

Bij deze methode vormen we van de twee sommanden weer een pseudosom en een register met de carry-bits. Deze twee worden vervolgens weer opgesteld; echter zodanig, dat bij het samenvoegen slechts elk tweede nog bestaande carry-bit wordt voortgeplant.

De ontstane vertraging is zo minimaal. Natuurlijk zijn soms wel een groot aantal korte stappen nodig. Na enige tijd zijn dan alle carry-bits ingevoegd. De totale tijd is bij deze methode evenredig met log n, waarbij n het aantal bits per woord is.
De methode is i.v.m. de mogelijkheid van het optreden van lange kettingoverdrachten niet erg aantrekkelijk en wordt alleen soms in een enigszins gemodificeerde vorm gebruikt. (L 3.1.; L 3.8.)

3.6. "Carry-Skip" technieken

Wanneer een ontstane carry (C_n) naar de volgende trap overgaat wordt daar uit a_n, b_n en C_n weer een S_n en een C_{n+1} gevormd. We moeten dus met de C_n weer de gehele optelprocedure doorlopen om te bepalen of er al dan niet een C_{n+1} zal ontstaan. Hierbij moet de C_n door een vrij groot aantal poorten, en soms via een flip-flop, worden voortgeplant. Dit levert een aanzienlijke tijdsvertraging op. Het proces verloopt hierbij dus zoals in figuur 3 is aangegeven.

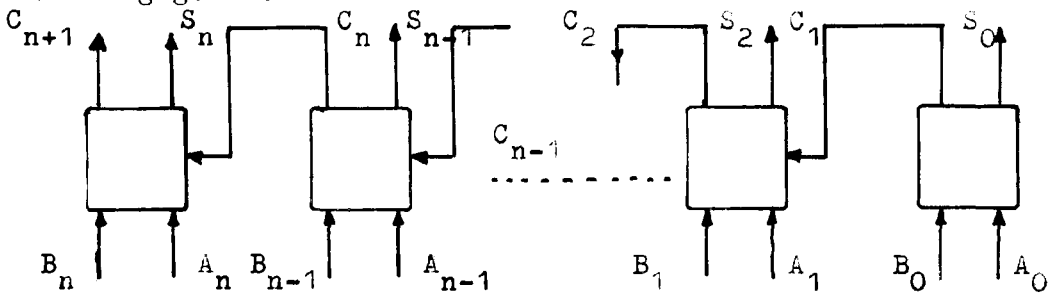


Fig.3

Een vrij grote snelheidswinst wordt verkregen, wanneer de carry de trappen, waaruit toch weer een nieuwe carry zal ontstaan, zou kunnen overslaan (skip). De aankomende C_n moet dan kunnen anticiperen of er al of niet een C_{n+1} zal ontstaan. Uiteraard is dit afhankelijk van de a_n en b_n. We kunnen nu een logisch circuit construeren, zodanig dat de carry slechts door maximaal een of twee poorten per trap behoeft als hij voortgeplant wordt. Deze poorten worden dan gestuurd door a_n en de b_n.
Uiteraard moet aan het sombepalingscircuit dan wel gemeld worden dat een carry gepasseerd is. Dit om in het geval: $a_n = 0$; $b_n = 1$; $C_n = 1$ niet alleen ervoor te zorgen dat $C_{n+1} = 1$, maar ook dat $S_n = 0$. Het informatie verloop wordt dan zoals in figuur 4 is aangegeven. Deze techniek werd in haar analoge vorm reeds door Babbage aangegeven.

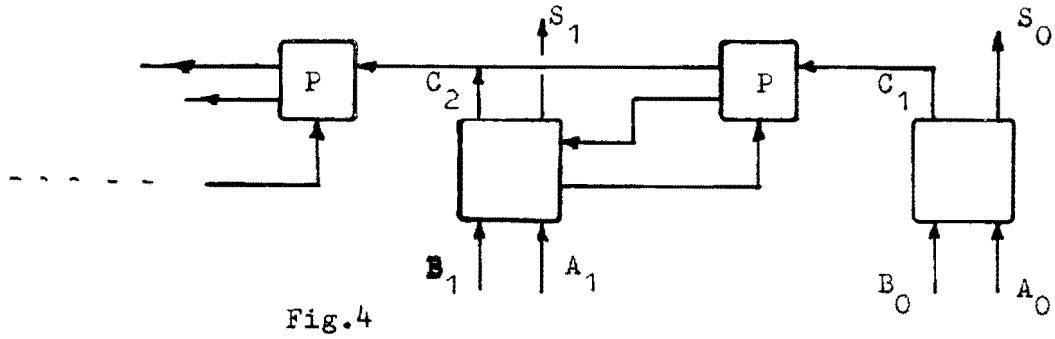


Fig.4

Voor enkele praktische circuits zij naar (L 3.11.) en voor een uitgebreidere toepassing naar (L 3.12.) verwezen.
Men kan natuurlijk nog verder gaan en stellen als a_i, b_i, a_{i+1} en b_{i+1} aan bepaalde voorwaarden voldoen, dan kan de carry twee trappen tegelijk overslaan. Men kan dan ook hiervoor weer de logica ontwerpen enz. Men kan zo steeds verder gaan en als maar groepjes, subgroepjes, andere combinaties, etc. vormen.
In het extreme geval bouwt men alle mogelijke combinaties in en komt dan op de volledige logica, zoals die in 3.4. is aangegeven, terecht. Meestal zal men echter een tussenoplossing kiezen, waarbij dan bepaalde technieken bestaan om optimale aanpassingen te verkrijgen (L.3.13). In het algemeen kan met de "skip-techniques" een vrij flexibele aanpassing aan de gestelde eisen worden verkregen. Speciaal wanneer gecombineerd met andere technieken (L 3.1.; L 3.2.).

3.7. "Exclusive Or" technieken

In 3.6. werd de carry via een door de betreffende trap gestuurde poort al of niet voortgeplant. In samenwerking met exclusive-or schakelingen zijn echter nog gunstiger resultaten te bereiken. Hierbij wordt een pre-set transistor al of niet geschakeld en de carry loopt al of niet door. We vormen nu a b en laten de output hiervan de schakelaar S (zie figuur 5) sturen.

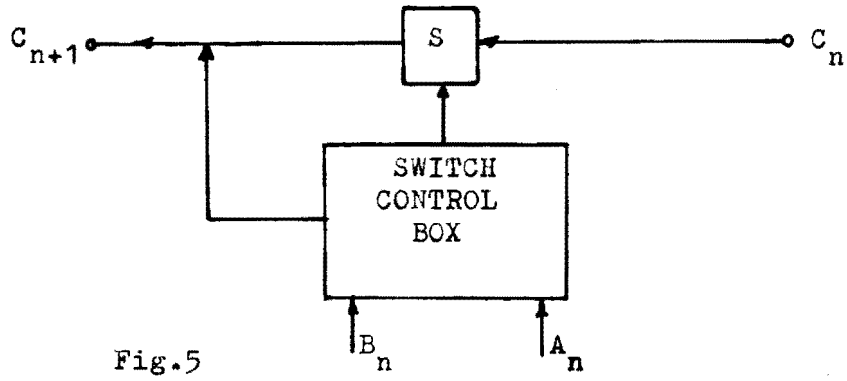


Fig.5

De carry loopt nu dus door de keten van schakelaars. Zaak is nu de schakelaars snel en met zo weinig mogelijk signaal vertraging te maken.

Voor dit doel gebruiken we daarom met succes transistoren. We komen dan tot datgene wat schematisch in figuur 6 is weergegeven.

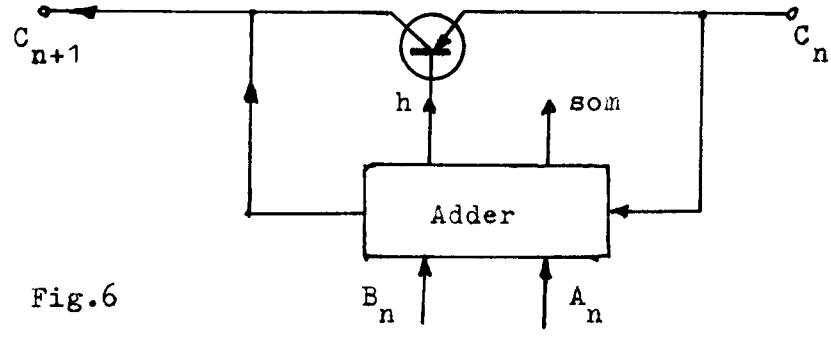


Fig.6

De carry C_n loopt hierbij door als de transistor open is, dus als de "Adder" een signaal levert. Dit wordt bepaald door: $h = a \quad b$. We kunnen op deze manier een zeer snelle propagatie krijgen. Een praktische uitvoering eist natuurlijk wel nadere aandacht voor belastingen, spanningsvallen en stabilitiet. (L 3.1.; L 3.14.; L 3.15.; L 3.16.; L 3.17.).

3.8. "Conditional Sum Adders"

Bij deze methode, die buitengewoon snel is, gaat men op een, op het eerste gezicht, nogal omslachtige manier te werk. En wel als volgt: We vormen van elke a_i en b_i de som en de bijbehorende carry en wel onder twee verschillende aannamen. In het eerste geval nemen we aan, dat de carry die in elke trap inkomt een 0 is, in het tweede geval dat deze een 1 is. Dus: eerste geval: a_i, b_i, $C_i = 0$; tweede geval a_i, b_i, $C_i = 1$. En vormen in beide gevallen S_i en C_{i+1}.

Vervolgens gaan we de trappen in paren bekijken, dus a_i (b_i) met a_{i+1} (b_{i+1}) etc. We vormen voor deze paren nu weer de som en de carry onder de aanname (conditie !) dat de inkomende carry in de rechtertrap van het paar resp. 0 en 1 is. Ditzelfde herhalen we voor elke tetrade, voor elke 8 bits, enz. Uiteindelijk hebben we dan de som en de carry voor het gehele woord. Deze laatste zijn dan ook niet langer conditioneel maar werkelijk. Dit lijkt een lange weg.Men dient echter te bedenken, dat in werkelijkheid niet al deze stappen gedaan worden. Men construeert hierbij nl. in componenten een logisch circuit, zodanig dat het eindresultaat in een keer bereikt wordt. Er bestaat nl. een logisch verband tussen de resultaten van elke boven aangegeven stap. Dit verband wordt dan in logica vastgelegd. Uiteraard kost dit dan een enorme hoeveelheid materiaal.

Dit is dan ook een van de zeer grote bezwaren van de methode. Een verder voordeel is echter dat de methode ook bruikbaar is voor andere talstelsels zoals bijv. het decimale stelsel. Voor verdere details wordt verwezen naar de aangegeven literatuur. (L 3.1.; L 3.2.; L 3.18).

3.9. Enkele andere methodes

Een methode die enige overeenkomst met bovengenoemde methode vertoont is de zgn. "Carry-Select Adder". Hierbij is de woordlengte van de beide sommanden opgesplitst in een aantal secties. Van elke sectie wordt nu de som bepaald op twee manieren. Bij de eerste manier wordt de som bepaald terwijl er een geforceerde carry in de minst significante trap wordt ingevoerd. Bij de tweede manier wordt er opgeteld zonder dat dit gebeurt. Welke van beide resultaten uiteindelijk gebruikt wordt, wordt bepaald door het al of niet optreden van een carry in de vorige sectie.
De grote van de secties zal hierbij ook een rol spelen. Men kan weer een grote snelheidswinst halen zoals gebruikelijk tegen de prijs van veel materiaal (L 3.19.).

3.10. De vergelijking

Wetende, dat elke vergelijking in zich een zeker risico met zich meebrengt, komt het de schrijver als gewenst voor hier te vermelden dat geen enkele vergelijking volkomen objectief is. Immers de een zal geinteresseerd zijn in het een, de ander in iets anders. De veel gehoorde klacht is dan ook dat men met behulp van een vergelijking kan bewijzen wat men wil, mits men het maar op een handige wijze aanpakt.
Zonder hier nader op in te gaan zou de schrijver zich op het standpunt willen stellen dat aan dit alles toch wel enigszins ontkomen kan worden, mits van tevoren nauwkeurig is aangegeven op welke basis de vergelijking is opgezet.
Dit zullen we hier dan ook eerst doen.
De volgende aannamen zijn gemaakt:

a. Transistoren en diodes zijn als logische schakelelementen gebruikt.
b. De prijs van een diode is als eenheid aangenomen. De transistorprijs is minimaal 3 tot maximaal 5 maal de diodeprijs genomen.

c. Een AND-OR combinatie wordt altijd gevolgd door een emitter-volger of een invertor (een twee transistor schakeling).
d. "Fan-in" en "fan-out" zijn beperkt tot 5.
e. Niveau herstellers (1 transistor) zijn aangenomen na elke 5 poorten.
f. De tijdsvertraging door een AND-OR-E_m volger combinatie is genomen als de tijdseenheid (t.e.).
g. De vertraging in een exclusieve-or, een niveau hersteller, een invertor en een emitter volger is op 1/4 tot 1/2 t.e. gesteld.
h. Per flip-flop zijn 4 transistors en 4 diodes aangenomen.
i. De "set-time" voor een flip-flop is als 1/4 tot 1/2 t.e. aangenomen.
j. De propagatie-tijd van "pre-set transistor switches" d.m.v. exclusive-ors is gesteld op 1/10 tot 1/2 t.e.
k. Waar van belang is een veiligheidsfactor van $1\frac{1}{2}$ ingevoerd wat de tijd betreft.
l. De woordlengte is in alle gevallen op 50 bits gesteld.

De op deze basis opgezette vergelijking is getabelleerd op de volgende pagina (Tabel 1).

Methode	Nr.	Verwezenlijking (evt. auteurs)	Gemiddelde hoeveelheid onderdelen		Onderdelen kosten per bit		Genormaliseerde snelheid per 50 bits	
			diodes	transist.	Min.	Max.	Min.	Max.
zonder versnelling	1		6	2,2	12,6	17	52,5	82,5
Carry Storage	2	Simpele logica	23	7	44	58	17,3	26,3
	3	"Radix four" met carry generatie	28,5	6,8	48,9	62,5	3,5	6,1
	4.	"Leading assimilation" in Nr. 2 toegepast	24	7,5	46,5	61,5	2,7	4,4
	5	"Leading assimilation" in Nr. 3 toegepast	29,5	7,3	51,4	66	1,6	3,1
	6	"Stored Skip"	25	7	40	50	2,5	3
Carry Detection	7	Gilchrist e.a.	16	5	31	51	orde 7	
Simultaneous Carry	8	Weinberger e.a.	31	1,8	36,4	40	4	6
Pyramid Carry	9	Nadler	15	6,5	34,5	47,5	6	8
Carry Skip	10	Enkelvoudige sprong	7	2,6	14,8	20	16	24
	11	Groepen en subgroepen	7,4	2	13,4	17,4	12	18
	12	Volledige logica	15	3,7	26,1	33,5	4	6
Exclusive-or	13	Kilburn e.a.	4	6,2	22,6	35	7,5	26,3
	14	Salter	6	6	24	35	7,5	26,3
	15	"Radix four exclusive-or"	10	4,6	19,8	27	3,8	13
Conditional	16	Sklansky	44	13,7	85,4	112,5	6	9

Tabel I.

Alhoewel deze vergelijking voor de ontwerper zeer zeker bruikbaar zal zijn, zal ze toch niet meer dan een eerste ruwe indicatie in de te volgen richting kunnen zijn.
Immers overwegingen op het gebied van betrouwbaarheid, moeilijkheden bij het realiseren van de circuits, enz. zullen uiteraard van essentieel belang zijn. Deze echter zijn in de onderhavige vergelijking in het geheel niet verdiskonteerd.
(L 3.1.; L 3.2.).

4. Het vermenigvuldigen

4.1. Algemeen

We zullen hier in het kader van onze studie m.b.t. de bestaande technieken een aantal methoden van vermenigvuldiging beschrijven. Voor een aantal veelgebruikte methoden zal de beschrijving iets uitvoeriger zijn, terwijl andere processen alleen kort aangegeven zullen worden. Op serie-technieken zal niet ingegaan worden (L 4.1.).

4.2. De "Normale" Parallel Methode

Binaire vermenigvuldiging doen we in feite als volgt, we tellen de multiplicand op de juiste plaats in een register op als het multiplier-bit een één is en wanneer dit laatste een nul is schuiven we de multiplicand alleen maar één plaats naar links. Op deze wijze krijgen we een aantal partiële produkten, die we op moeten tellen om tot het uiteindelijke produkt te komen. Een computer is echter weinig geschikt om meer dan twee getallen tegelijk op te tellen. Daarom tellen we na de vorming van elk partieel produkt dit direct op bij de som van de reeds gevormde partiële produkten.
Een zeer veel gebruikte methode is hierbij de volgende (zie figuur 7):

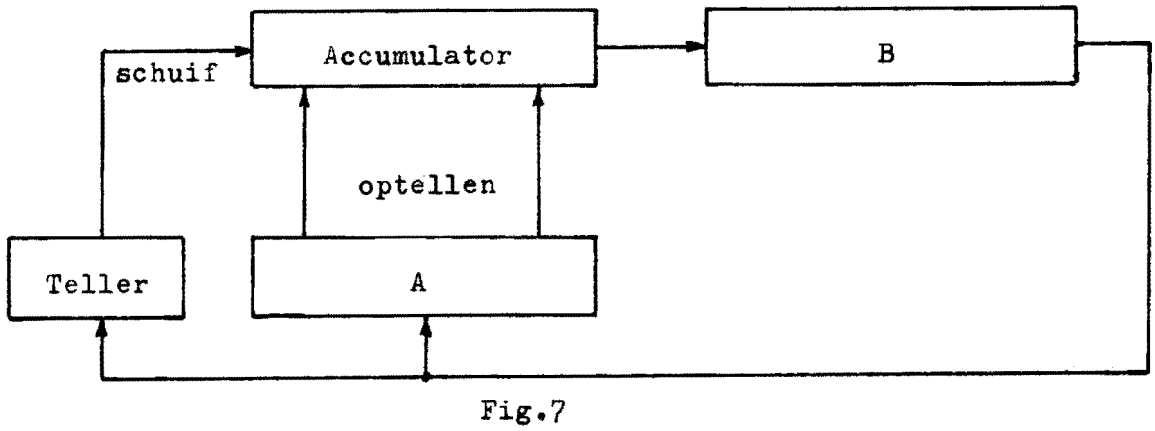


Fig.7

We kunnen de, bekend veronderstelde, methode als volgt kort aangeven. Is het minst significante bit van B een 1, dan tellen we A op bij de accumulatorinhoud (nog nul !) en schuiven hierna zowel de inhoud van de accumulator als van het B register één plaats naar rechts.

Was het minst significante bit B een 0, dan schuiven we alleen maar. Ook verlagen we in beide gevallen de teller met 1. Vervolgens herhalen we de procedure voor het tweede bit van B (dat nu achteraan staat !). Is de teller leeg, dan is de vermenigvuldiging klaar. Het meest significante deel van het produkt staat dan in de accumulator en het minst significante deel in het B-register.

4.3. Simultaan Vermenigvuldiging

Als we bijv. twee getallen van 4 bit X en Y hebben en we willen X x Y hebben, dan ontstaan hierbij de partiële produkten A, B, C en D. Bij een simultane vermenigvuldiger hebben we nu een optelcircuit, dat P = A + B, Q = C + D en R = P + Q = X x Y vormt. De vermenigvuldiging komt tot stand door de signalen van multiplicand en multiplier gedurende enige tijd aan een circuit aan te sluiten. Nadat de overgangsverschijnselen in het circuit afgelopen zijn, is dan het produkt aan de uitgang beschikbaar. En wel zo lang als de ingangssignalen aan de ingang blijven. Het circuit is dus eigenlijk opgebouwd volgens de directe logische regels voor het vermenigvuldigen. Het is een vrij snelle methode, maar de hoeveelheid vereist materiaal is aanzienlijk; vooral bij grotere woordlengte. De methode wordt zeer weinig gebruikt en we zullen er verder ook niet op ingaan (L 4.1.).

4.4. Asynchrone Vermenigvuldiging

De techniek beschreven in 4.2. kan ook worden toegepast in asynchrone circuits. De partiële som wordt hierbij in een asynchrone logische schakeling gevormd en het resultaat wordt ook direkt één plaats naar rechts verschoven in de accumulator geplaatst. Men heeft hierbij dus geen aparte schuifprocedure nodig. In het circuit ontstaat dus a.h.w. een uitsterven van signalen tot de operatie voltooid is. Ook dit is een snelle methode met weer als nadeel de grote hoeveelheid materiaal. Zodat ook deze methode niet veel gebruikt wordt (L 4.1.).

4.5. Serie-Parallel Vermenigvuldiging

Deze methode is een zeer snelle met (uiteraard !) de consequentie, dat er ook weer vrij veel materiaal vereist is. Toch wordt ze wel toegepast, vooral wanneer snelheid prioriteit bezit.

Bij deze methode (figuur 8) worden beide factoren opgesplitst in groepen van 3 bits, deze groepen worden dan na elkaar behandeld. We vormen bij deze manier van vermenigvuldigen d.m.v. een zgn. produkt generator de veelvouden M,2M, ..., 7M, waarbij M de in behandeling zijndegroep van 3 bits van de multiplicand is. Met behulp van een produkt selector, die gestuurd wordt door de vermenigvuldiger R, wordt dan het juiste veelvoud gekozen. In de blokken 3F wordt dan de som gevormd.

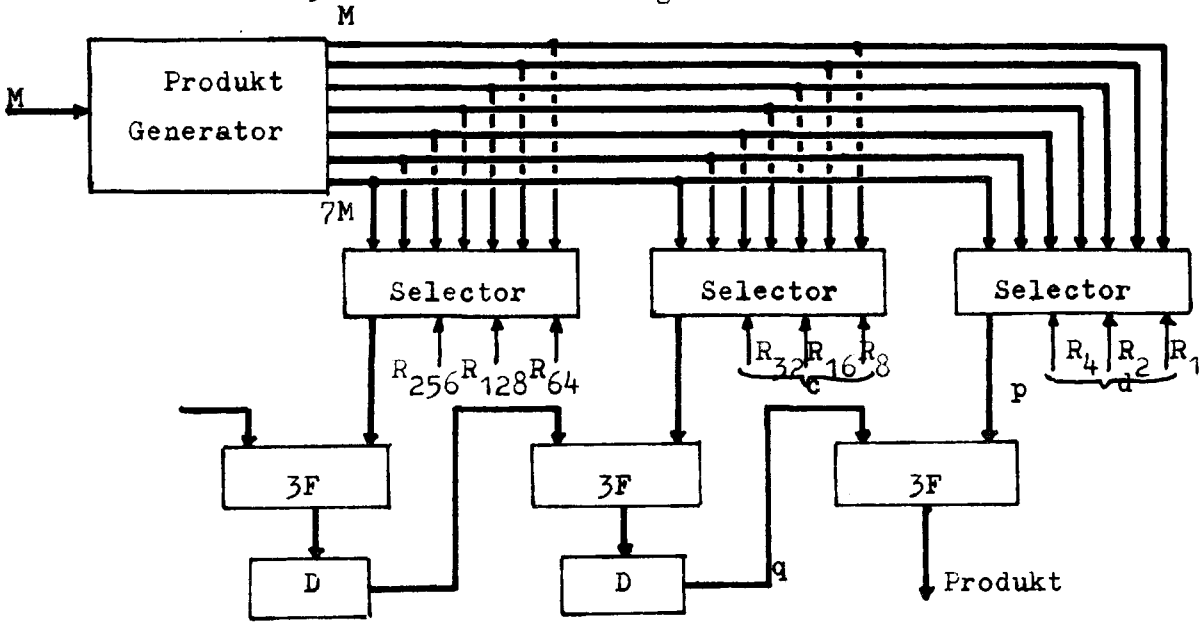


Fig.8

(De dikke lijnen stellen 3-voudige verbindingen voor.)

Dit blok bestaat in wezen uit 3 full-adders. De methode zal kort met een voorbeeld aangegeven worden.

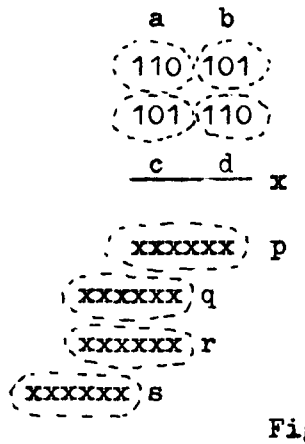


Fig.9

Er zijn dus twee getallen, die we willen vermenigvuldigen. We vermenigvuldigen daartoe bij deze methode eerst d met b en c met b. Daarbij ontstaan de resultaten p en q. Waarbij q t.o.v. p over 3 plaatsen verschoven is.
Vervolgens vormen we d x a en c x a. Dit levert weer r en s met s verschoven t.o.v. r.

Eerst fungeert nu b als M voor de produktgenerator en later a. De veelvouden b t/m 7 b worden dus eerst gevormd. Waarbij d en c de bits zijn die resp. aan de rechter en de middelste produkt selector

in figuur 8 worden toegevoerd. Dus $R_1 = 0$, $R_2 = 1$, $R_4 = 1$, $R_8 = 1$, $R_{16} = 0$ en $R_{32} = 1$.
In de rechter selector wordt dus 6M (= 6b) gekozen en aan 3F toegevoerd; in de middelste selector 5M (= 5b) (immers d en c zijn resp. 6 en 5). Dit zijn dus p en q. Het produkt komt nu aan de uitgang beschikbaar. Om echter de juiste verschuiving van q t.o.v. p te bereiken is een vertragingsblok D tussengeschakeld. Daarom komen dus eerst de drie rechtse bits van p aan de uitgang en vervolgens de som van de linkse drie bits van p en de rechtse 3 van q. Intussen is nu M = a aan de produktgenerator toegevoerd zodat de procedure hiermee herhaald wordt.
Opgemerkt zij nog, dat de produkten M t/m 7 M gegenereerd worden d.m.v. verdubbeling (vertraging) en eventueel optellen.

4.6. Het Verkort Vermenigvuldigen

In de "normale" methode, zoals die onder 4.2. aangegeven is, zijn even veel optel- en -schuif cycli vereist als er bits in de vermenigvuldiger (= multiplier) zijn. Nu komen echter in willekeurige binaire getallen steeds kortere of langere ketens van nullen of enen voor. Het is nu mogelijk om tijd te besparen door deze ketens in één keer te overbruggen. Immers als er nullen staan kunnen we zonder meer dit stuk overslaan mits we ervoor zorgen dat we wel het juiste aantal plaatsen opschuiven voordat we de volgende optelling doen.
Ook voor een keten van enen kan een aanzienlijke tijdsbesparing verkregen worden. Hebben we bijv. een reeks van n opeenvolgende enen in de vermenigvuldiger, dan is de numerieke waarde van dit gedeelte $2^{t+n-1} + 2^{t+n-2} + \ldots\ldots + ^{t+1} + 2^t$.
Waarbij aangenomen is, dat het meest rechtse bit van de keten het t-de bit van rechts in het gehele getal is.

Het is echter zonder meer duidelijk dat dit ook geschreven kan worden als $2^{t+n} - 2^t$. In plaats van n optellingen voor elk van de enen is het proces dus gereduceerd tot een aftrekking voor de meest rechtse 1 en een optelling voor de 0 direkt links van de reeks van enen. Het feit nu dat het mogelijk is om een produkt te bepalen door cycli over te slaan die overeenkomen met die in series nullen of enen is de basis van de onderhavige methode voor verkort vermenigvuldigen.

Teneinde voor elke optelling of aftrekking de juiste plaats te hebben is het beslist noodzakelijk om voor elke cyclus die overgeslagen wordt het resultaat over een plaats te verschuiven. Om ten volle profijt te trekken van het overslaan van cycli is het nu gewenst over meer dan een plaats tegelijk te kunnen schuiven. Ideaal is het hierbij om de verschuiving over de bits die overgeslagen kunnen worden te combineren met de verschuiving behorende bij de laatste werkelijke uitgevoerde cyclus (optelling of aftrekking). Als bijv. 3 cycli overgeslagen kunnen worden, moet dus eigenlijk 1 verschuiving van 4 plaatsen uitgevoerd kunnen worden i.p.v. 4 verschuivingen over 1 plaats. In theorie kan het voorkomen, dat bij een woordlengte n, n-1 cycli overgeslagen kunnen worden. In de praktijk echter is het ondoenlijk om voorzieningen in te bouwen die het mogelijk maken om te schuiven over naar verkiezing 1, 2, ..., n-1 plaatsen.
Er zal met minder volstaan moeten worden. In voorkomende gevallen zal men dan meerdere schuifcycli moeten toelaten.
Hoe groter het maximale aantal plaatsen is, dat overgeslagen kan worden des te groter is de efficiency. In figuur 10 is het resultaat van een aantal numerieke berekeningen m.b.t. de efficiency van de methode weergegeven. De berekeningen zijn uitgevoerd door Smith en Weinberger van het NBS in Washington (L 4.4.).
Men ziet dat het weinig winstgevend is om het maximale aantal cycli dat overgeslagen kan worden groter dan 3 of 4 te kiezen. De winst die men krijgt bij toevals-series is dan een factor drie. De krommen zijn berekend voor resp. de ongunstigste reeksen van de vorm 101010101010, reeksen die m.b.v. toevalscijfers samengesteld zijn en reeksen die de gunstigste configuratie vertonen.
Dit laatste is het geval wanneer de kettingen van nullen en enen een veelvoud zijn van n + 1, wanneer n het maximale aantal bits is dat kan worden overgeslagen.
Ten einde, uitgaande van het bovenbeschreven principe, tot een machinaal te verwezenlijken operatie te komen, moeten een aantal regels vastgelegd worden die de logische structuur van het circuit vastleggen. Deze regels bestaan uit twee wezenlijk verschillende delen. Ten eerste de regels, die bepalen of er opgeteld of afgetrokken moet worden of eventueel geen van beide. Immers soms kan het voorkomen dat het aantal plaatsen dat overgeslagen kan worden in een bepaald geval groter is dan het maximaal mogelijke aantal plaatsen n waarvoor uit praktische overwegingen voorzieningen zijn getroffen.

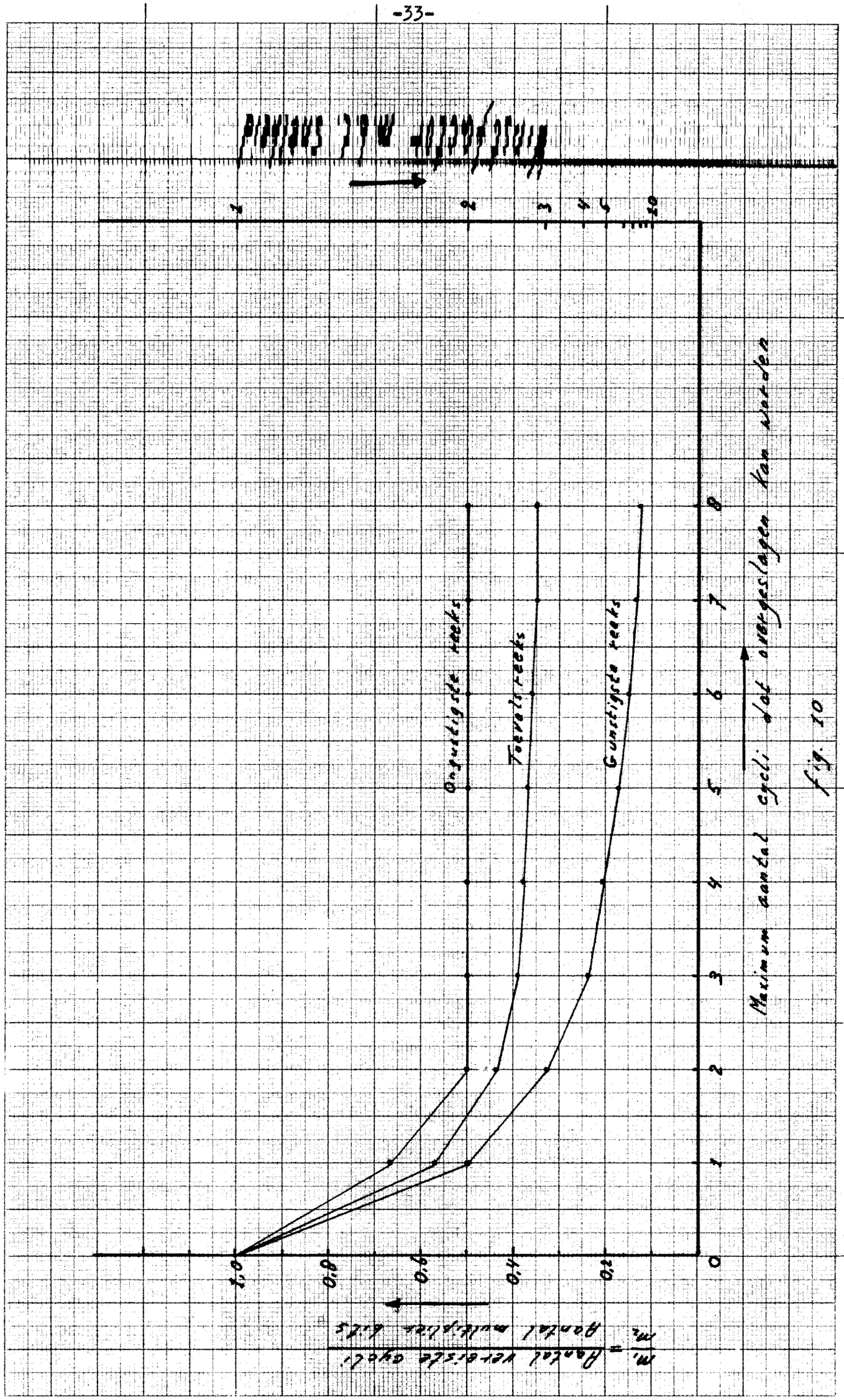

Maximum aantal cycli dat aangegeven kan worden
0 1 2 3 4 5 6 7 8
Ongunstigste reeks
Toevalsreeks
Gunstigste reeks
1,0 0,8 0,6 0,4 0,2
$\frac{m_1}{m_2} = \frac{\text{Aantal verbrijzelde cycli}}{\text{Aantal meetpunten}}$
1 2 3 4 5 10
Maximale factor m.b.t. toelaatbaar

fig. 10

Ten tweede de regels die bepalen over hoeveel plaatsen er in de accumulator en het vermenigvuldigerregister geschoven kan worden. Dit aantal loopt dus van 1 tot n + 1. Alle regels hangen nu af van het bit dat op het betreffende moment beschouwd wordt (dus het meest rechtse nog niet behandelde bit van de vermenigvuldiger) en van het teken van de tot dan toe verkregen som van partiële produkten; m.a.w. van het feit of de laatste bewerking een optelling of een aftrekking was.
De regels die op deze wijze verkregen worden zijn hieronder in tabelvorm aangegeven.

Bewerking	Vereiste condities
Tel multiplicand op	01/+ of 00/-
trek multiplicand af	11/+ of 10/.

Tabel 2. De regels voor optellen of aftrekken.

Bewerking	Vereiste condities
Schuif 1 plaats	10/+ of 01/-
Schuif 2 plaatsen	101/+ of 010/- of 100/± of 011/±
Schuif 3 plaatsen	1001/+ of 0110/- of 1000/± of 0111/±
Schuif 4 plaatsen	0001/+ of 1110/- of 0000/± of 1111/±

Tabel 3. De regels voor het schuiven

De symbolen in de rechterkolommen van bovenstaande tabellen hebben de volgende betekenis:
+ betekent dat de som der partiële produkten > 0 is;
- betekent dat de som der partiële produkten < 0 is.

de nullen en enen geven de waarden van de momentane meest rechtse bits van de vermenigvuldiger aan.
Voorbeeld: Is de vermenigvuldiger 10011011100101 en zijn we gevorderd tot en met het vierde bit van rechts en is de accumulatorinhoud positief, dan tellen we dus noch op noch trekken af maar schuiven direkt een plaats verder (aan de eerste voorwaarde uit de bovenste rij van tabel 3 is voldaan).
Het blokschema van het circuit dat deze methode realiseert is in figuur 11 aangegeven.

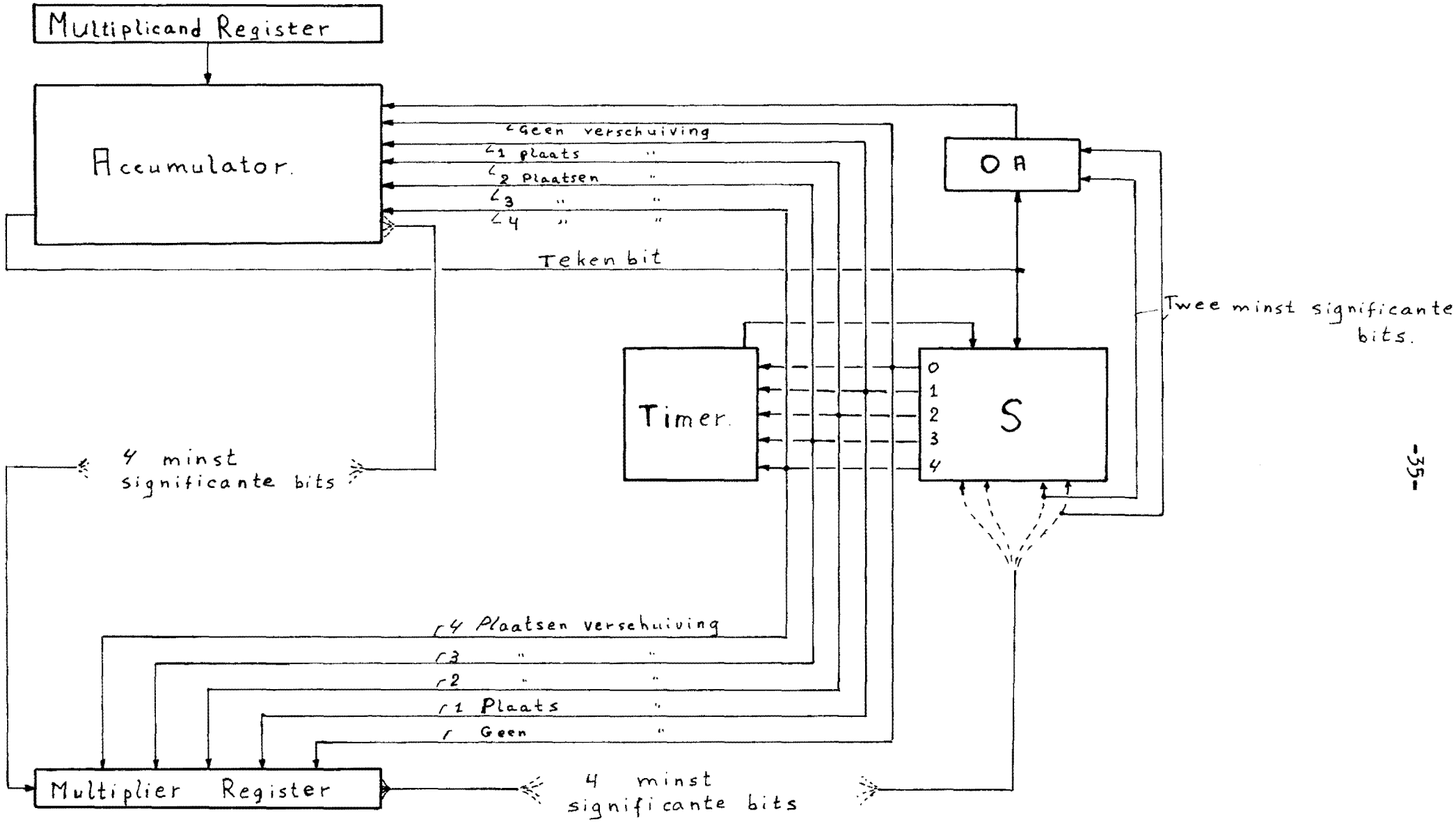

Multiplicand Register
Accumulator
Geen verschuiving
1 plaats "
2 Plaatsen "
3 " "
4 " "
Teken bit
O A
Twee minst significante bits.
Timer
S
0
1
2
3
4
4 minst significante bits
4 Plaatsen verschuiving
3 "
2 "
1 Plaats "
Geen "
Multiplier Register
4 minst significante bits

Fig.11

In de figuur is OA de schakeling die volgens tabel 2 bepaalt of de multiplicand al of niet bij de accumulatorinhoud opgeteld of er afgetrokken moet worden. Verder is S de schakeling die aan de hand van tabel 3 bepaalt over hoeveel plaatsen er geschoven moet worden. Als er geschoven wordt dient dit zowel in het accumulatorregister als in het vermenigvuldigerregister te geschieden. De minder significante bits worden dan uit de accumulator in het vermenigvuldigerregister geschoven. Dit laatste bevat dus uiteindelijk weer de minor van het produkt. De "timer" tenslotte is een schakeling die voor het in de juiste volgorde aflopen van de operaties zorg draagt. In het blokschema is te zien, dat de hoeveelheid extra materiaal nogal meevalt. Het belangrijkste probleem in het geheel is het scheppen van faciliteiten voor het schuiven over meerdere plaatsen.
(L 4.2.; L 4.3.; L 4.4.; L 4.5.; L 4.6.; L 4.7.).

5. Het Delen

5.1. Algemeen

Het deelproces bestaat in principe uit een aantal aftrekkingen. We kunnen het dus zeer wel vergelijken met het vermenigvuldigen. Ook hier zijn drie getallen bij betrokken. Het verschil is uiteraard dat we bij het delen het produkt-getal reeds weten en vragen naar de tweede ontbrekende produktfactor. We hebben dus de dividend en de divisor en vragen naar het quotient.
We kunnen de divisor dan vergelijken met de multiplicand bij het vermenigvuldigen. Het quotient komt dan overeen met de multiplier. Niettegenstaande het feit dat er veel overeenkomstige punten zijn in beide is het delen toch wezenlijk gecompliceerder dan het vermenigvuldigen. De belangrijkste problemen zijn daarbij de volgende:

a. Het starten van het deelproces;
b. Wat te doen als er bij een aftrekking een negatief resultaat ontstaat ?
c. Wat te doen met de overblijvende rest bij een deling die niet opgaat ?

Deze en andere problemen hebben er toe geleid, dat lange tijd de computer geen directe deelinstructie kenden (dit geldt ook nu nog vaak !). Het delen werd dan teruggebracht tot een vermenigvuldiging. Immers $a : b = a \times \frac{1}{b}$. Het inverteren van b nu kan geschieden m.b.v. een numeriek proces dat weer uit optellen en vermenigvuldigen bestaat. Op deze wijze is het rechtstreekse delen dus omzeild en vervangen door een subroutine. Het gehele proces is dan echter nogal omslachtig en tijdrovend. Gedurende de laatste jaren is dan ook gezocht naar een andere oplossing van het probleem. Men heeft hierbij o.a. redundante codes ontwikkeld: delingsprocessen ontwikkeld die gebruik maken van hogere machten van twee, etc.
Bijzonder veel aandacht is hierbij besteed aan het optreden van een negatief resultaat bij een aftrekking. Men heeft methodes ontwikkeld (zgn. "non-restoring division methods"), waarbij met het negatieve resultaat verder gewerkt kan worden. Dit in tegenstelling met de zgn. "restoring division". Tot algemene toepassingen heeft dit echter nog niet geleid (L 5.1.; L 5.2.; L 5.3.).

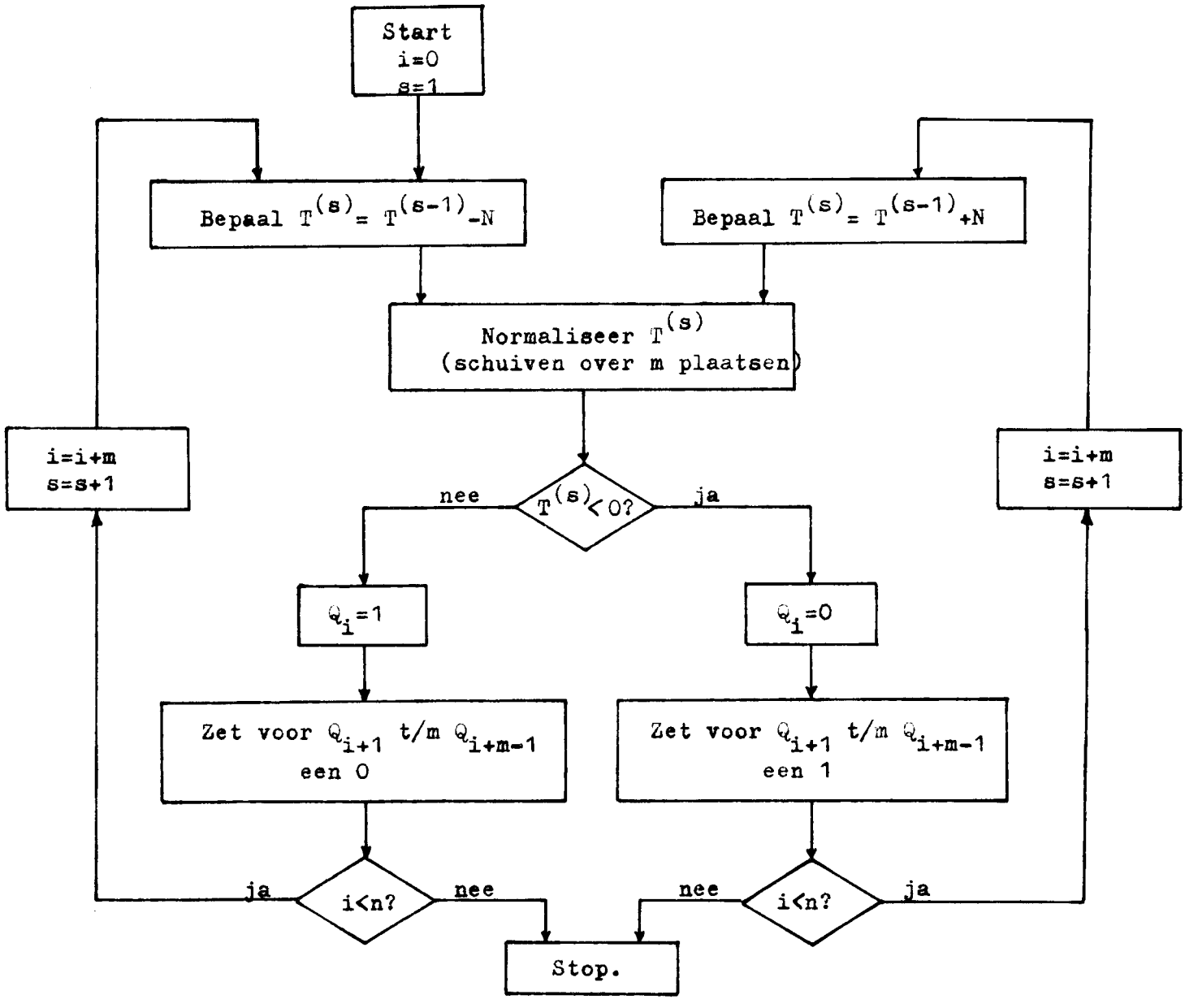
5.2. Het Verkorte Delen

Zoals we reeds zeiden bestaat het deelproces uit een aantal aftrekkingen. De tijd, benodigd voor het delen, zal dus bij benadering evenredig zijn met het aantal aftrekkingen dat verricht moet worden. Er is dus een mogelijkheid tot aanzienlijke tijdswinst, wanneer dit aantal gereduceerd kan worden. Bij het vermenigvuldigen hadden we een analoog probleem. We hebben daar ook een oplossing gezien. Eenzelfde soort oplossing is nu ook mogelijk voor het delen. Ten einde dit te bereiken zijn diverse methodes ontwikkeld geworden. (L 5.4.; L 5.5.; L 5.6.; L 5.7.; L 5.8.)
We zullen hier niet alle methodes beschrijven, maar ons beperken tot het kort aangeven van het algemene principe dat aan alle oplossingen ten grondslag ligt. De basis is, evenals bij het verkort vermenigvuldigen, het feit dat een binair getal samengesteld gedacht kan worden uit:

a. kettingen van nullen;
b. kettingen van enen;
c. kettingen van nullen met een geisoleerde een;
d. kettingen van enen met een geisoleerde nul.

Dit betekent dat een getal op kortere wijze is aan te geven. Een voorbeeld: 0, 1111 000 1101 0010 is als volgt opgebouwd te denken $2^0 - 2^{-4} + 2^{-7} - 2^{-10} - 2^{-11} + 2^{-14}$.
Dit betekent dat in een deling bijv. een quotient dat bovenstaande binaire vorm zou hebben ook met minder bewerkingen bepaald zou kunnen worden. Zonder diep op de theoretische achtergronden in te gaan, kan de methode, die daarbij gevolgd dient te worden, als volgt omschreven worden (zie ook figuur 12).

We willen dus $Q = \frac{T}{N}$ bepalen. We gaan er nu van uit dat zowel T als N genormaliseerd zijn. D.w.z. het eerste bit rechts van de komma is een 1. Verder staan zowel T als N zodanig in de registers dat de komma uiterst links staat. We beginnen met i = 0 en stoppen wanneer i = n, waarbij n het aantal bits is, dat we in het quotient wensen. Dus de nauwkeurigheid. We vormen $T^1 = T - N$. Laten we aannemen, dat $T < N$ is; dan is $T^1 < 0$. Dit leidt tot de rechter lus in figuur 12. Stel, er zijn m nullen rechts van de komma. T^1 wordt nu weer genormaliseerd. Ook plaatsen we een 0 in de meest rechtse positie van het Q-register.



(Opm.: m is het aantal plaatsen dat overgeslagen kan worden.)

Fig.12

Voor op één na elke stap die geschoven wordt bij het normaliseren plaatsen we tevens een 1 in de meest rechtse plaats van het Q-register en schuiven de inhoud van het Q-register tegelijkertijd een plaats naar links. Vervolgens hogen we de i-teller en de s-teller op. Nu wordt weer $T^{11} = T^{1} + N$ bepaald. Is T^{11} negatief, dan is de nieuwe Q_i (eigenlijk Q_m) = 0 en de volgende bits waarover genormaliseerd wordt zijn enen. Is T^{11} positief dan is $Q_i = 1$ en de volgende bits zijn nullen. Nu hogen we i en s weer op en bepalen T^{111}, enz. Zolang tot i = n, waarna we de bewerking stoppen.

Samengevat zijn er dus de volgende regels:

1. Als het deeltal in de normale vorm staat, schuif dan zoveel stappen als er nullen zijn tot aan de eerste één. Waarbij voor elk van de nullen, behalve de laatste, een nul van de rechterkant in het quotientregister geschoven wordt. Tel vervolgens het complement van de deler bij het deeltal op.
   a. In het resulaat in normale vorm, plaats dan een 1 rechts in het quotientregister en schuif zoveel stappen als er nullen zijn.
   b. In het resultaat in complementaire vorm, plaats dan een nul in het quotientregister en schuif zoveel stappen als er enen zijn.
2. Als het deeltal in complementaire vorm staat, schuif dan zoveel plaatsen als er enen voorop staan, voor elke één, behalve de laatste, een één in het quotientregister schuivend. Tel vervolgens de deler in normale vorm bij het deeltal op.
   a. In het resultaat in normale vorm, plaats dan een één in het quotientregister. Schuif vervolgens zoveel plaatsen als er nullen zijn.
   b. Is het resultaat in complementaire vorm, plaats dan een nul in het quotientregister en schuif zoveel plaatsen als er enen zijn.

Hieronder is ter illustratie een voorbeeld gegeven.

T	, 1001111100001100	Quotient
-N	-, 1101	↓
T^{1}	- 00,11000011110100	0,1....
+N	+,1101	↓
T^{11}	0000,1100001100	0,11000
-N	-,1101	↓
T^{111}	- 0000,110100	0,110000111
+N	+,1101	↓
T^{1111}	000000	0,110000111100

Het is duidelijk dat, teneinde efficient te werken, er over meer dan een plaats tegelijk geschoven moet kunnen worden.
Deze methode zal een aanzienlijke tijdsbesparing geven terwijl de investering in de vorm van materiaal, afgezien van het schuiven over meer plaatsen, minimaal is.

Het blijkt echter dat het systeem nog niet de optimale resultaten geeft. Het komt nl. voor dat men, de bovengenoemde regels volgend, een plaats te veel of te weinig opschuift om de optimale snelheid te halen. Dit is een gevolg van het feit dat men een binair getal niet op slechts één manier eenvoudiger kan schrijven, maar op meerdere. Ten einde optimale resultaten te bereiken, dient men ook de juiste schrijfwijze te hebben. Bij het vermenigvuldigen leverde dit weinig problemen op, daar men immers de vermenigvuldiger kent. Bij het delen kent men echter het quotient van tevoren niet.
Ten einde toch de beste resultaten te bereiken worden dan soms verschillende aannamen gemaakt en de gevolgen bij elk van de aannamen bepaald. Ten einde dit efficient te doen heeft men echter twee of meer "adders" nodig en dit weegt vaak niet op tegen de extra winst in tijd. Voor een uitgebreide beschouwing zij verwezen naar L 5.8. (voor een vergelijkende studie zie L 5.9.).
Door deze gedurende de laatste jaren, ontwikkelde methoden is het delen enigszins op de voorgrond gehaald.

6. De Hoofdlijnen van het Ontwerp

6.1. Algemeen

In de voorgaande hoofdstukken zijn een groot aantal mogelijkheden en moderne ontwikkelingen aangegeven en samengevat. De studie omvatte bijna alle gebieden die van belang zijn bij het ontwerp van een "processor" voor een moderne computer. In dit hoofdstuk nu zullen we de hoofdlijnen vastleggen waarlangs het ontwerp zich zal gaan voltrekken. Aangenomen wordt dat de verantwoording voor de keuze van bepaalde methoden en technieken in voldoende mate in de voorgaande hoofdstukken is gegeven. Er zal dan ook volstaan worden met het aangeven van de gekozen oplossingen.
De volgende specificaties zullen de gedaante van het ontwerp bepalen:

a. De informatieverwerking zal volledig parallel geschieden. Dit in verband met de vereiste snelheid.
b. Er zal inwendig een zuivere binaire code gebruikt worden. De conversie wordt hierbij aan de peripherie van de computer gedacht.
c. De besturing van de processor zal in micro-geprogrammeerde vorm geschieden. Hierbij zal in eerste instantie niet ingegaan worden op de vraag: "Waar komen de microbevelen vandaan?"
d. Er zal een schuifmechanisme ingebouwd worden, zodanig dat naar keuze over een of over meerdere plaatsen tegelijk geschoven zal kunnen worden.
e. Bij het optellen en aftrekken zal het probleem van de carry-propagatie aangepakt worden m.b.v. "exclusive-or" schakelingen waarin transistor schakelaars (zie paragraaf 3.7.).
f. Als vermenigvuldigings-methode wordt het verkort vermenigvuldigen volgens Smith en Weinberger gekozen (zie hoofdstuk 4).
g. Ook voor het delen wordt de verkorte methode gebruikt en wel de eenvoudige, niet helemaal optimale, methode met één "adder".

h. Alhoewel niet van direkt belang in het ontwerp, wordt gedacht aan een woordlengte van ongeveer 40 bits.

Binnen dit kader dienen dus de details van het ontwerp verder uitgewerkt te worden.

6.2. Het Instructie Repertoire

Zoals reeds in de aanhef van dit rapport gezegd is, is uitgegaan van het standpunt dat de processor een krachtig arsenaal van aritmetische en logische bewerkingsmogelijkheden dient te bezitten. Ten einde een zeer gevarieerd spectrum van opgaven aan te kunnen is het volgende instructie-programma samengesteld:

01	optellen met vaste komma	
02	aftrekken met vaste komma	
03	vergelijken met vaste komma	
04	vermenigvuldigen met vaste komma	
05	delen met vaste komma	
06	optellen met drijvende komma	
07	aftrekken met drijvende komma	
08	vergelijken met drijvende komma	
09	vermenigvuldigen met drijvende komma	
10	delen met drijvende komma	
11	bepaal $A \cdot B$	logische bewerkingen
12	bepaal $A \cdot \overline{B}$	
13	bepaal $\overline{A} \cdot B$	
14	bepaal $\overline{A} \cdot \overline{B}$	
15	bepaal $A + B$	
16	bepaal $A + \overline{B}$	
17	bepaal $\overline{A} + B$	
18	bepaal $\overline{A} + \overline{B}$	
19	bepaal $A \oplus B$	
20	bepaal $A \odot B$	
21	normaal indien 1, anders 0	extractie
22	complement indien 1, anders 0	
23	normaal indien 1, anders 1	
24	complement indien 1, anders 1	
25	complement indien 1, anders normaal	
26	complement indien 0, anders normaal	

Behalve deze instructies zullen ook nog diverse "transfer"- en "branch"-bevelen nodig zijn. Deze kunnen echter beter uitgewerkt worden nadat de andere details verder ontwikkeld zijn.

7. Optellen, Aftrekken en vergelijken met Vaste Komma

7.1. Het Overall - Schema

De beschrijving van het verloop van de bewerkingen geschiedt aan de hand van figuur 13. We nemen verder aan dat de woordlengte n bits is.
Begonnen wordt met de getallen A en B van buitenaf in de n-bits registers A_1 en B_1 te zetten. Het teken A en B komt daarbij in respectievelijk T_A en T_B. Tegelijkertijd worden de n-bit registers A_2 en B_2 op nul gezet. In het logisch netwerk N_3 wordt nu aan de hand van T_A, T_B en het feit of we moeten optellen of aftrekken bepaald of B voor de optelling al ($\bar{b}$) dan niet (b) geinverteerd moet worden. De uitgang van N_3 stuurt het netwerk N_2 dat òf de normale òf de complementaire uitgang van B_1 met het verdere circuit verbindt. N_2 is zodanig dat, zo gauw het netwerk op m geactiveerd wordt, de inhoud van B_1, al dan niet geinverteerd, in B_2 loopt. N_1 is simpeler, want hier behoeft de informatie in elk geval niet geinverteerd te worden. Het circuit bestaat dus uit n en-poorten. Zo gauw een signaal op de m-ingangen van N_1 en N_2 verschijnt lopen de getallen A en B in de eigenlijke optelschakeling, aangegeven met "Adder". Nadat de carry-propagatie geëindigd is staat dus aan de uitgang van de "Adder" het resultaat van de optelling klaar. Dit resultaat kan echter nog enige correctie behoeven. Daartoe wordt nu in de flip-flop Ov. een eventueel uit de optelling ontstane overflow geregistreerd. In samenwerking met de informatie uit N_3 (al of niet inverteren voor de bewerking) wordt nu in N_4 de te volgen procedure bepaald.
De informatie loopt dan via N_5 en N_6 als resultaat R met teken T_R naar de "Selector". Deze stuurt de informatie, afhankelijk van de besturing m, naar één van de zestien adressen S_1 tot en met S_{16}.

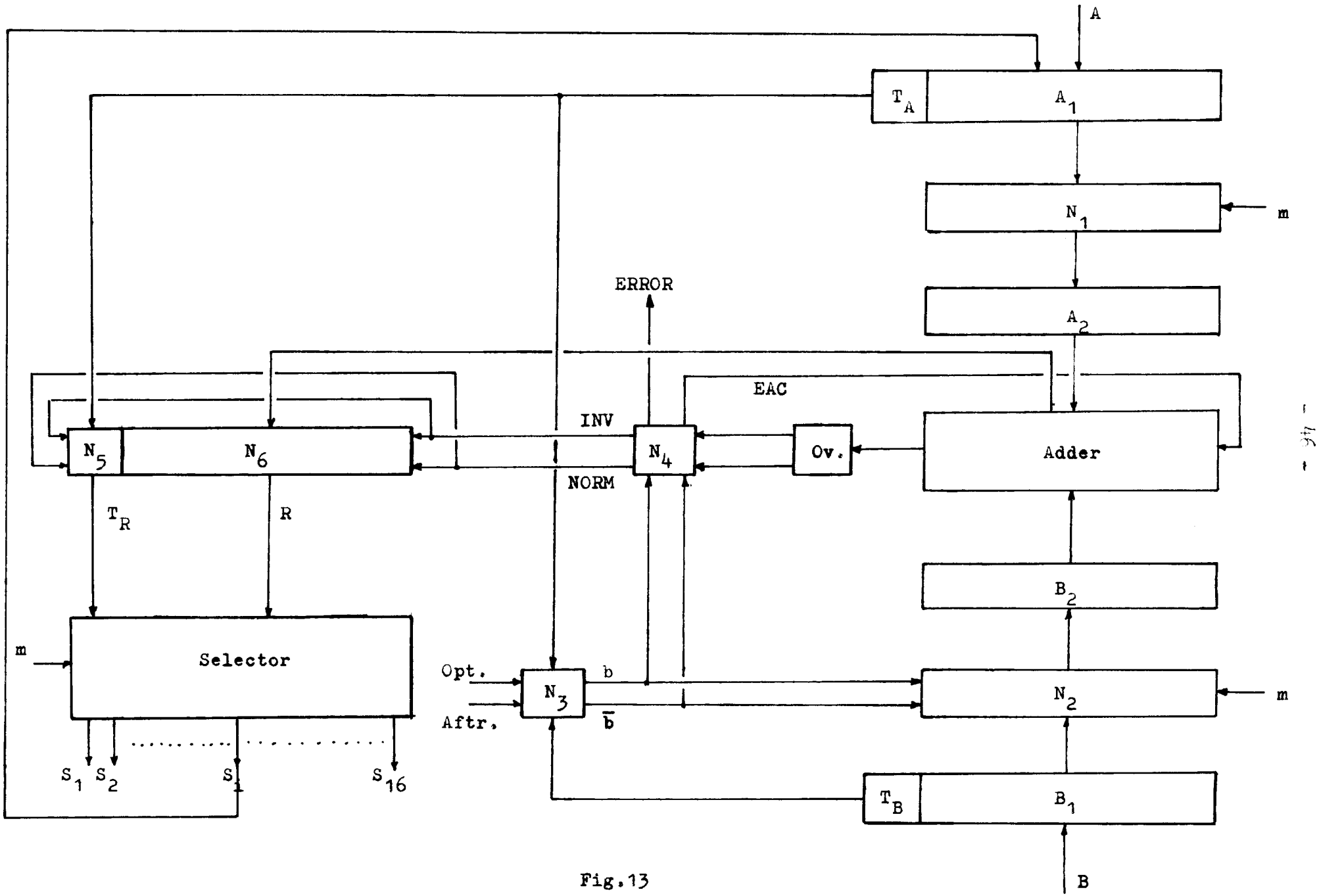

A
T_A
A_1
N_1
m
ERROR
A_2
EAC
INV
N_5
N_6
N_4
Ov.
Adder
NORM
T_R
R
B_2
m
Selector
Opt.
N_3
b
N_2
m
Aftr.
$\overline{b}$
S_1 S_2 S_i S_{16}
T_B
B_1
B

Fig.13

Teneinde het verloop van de informatiestroom bij de optel-procedure te laten zien is in fig, 14 m.b.v. een "flow-chart" deze stroom aangegeven.

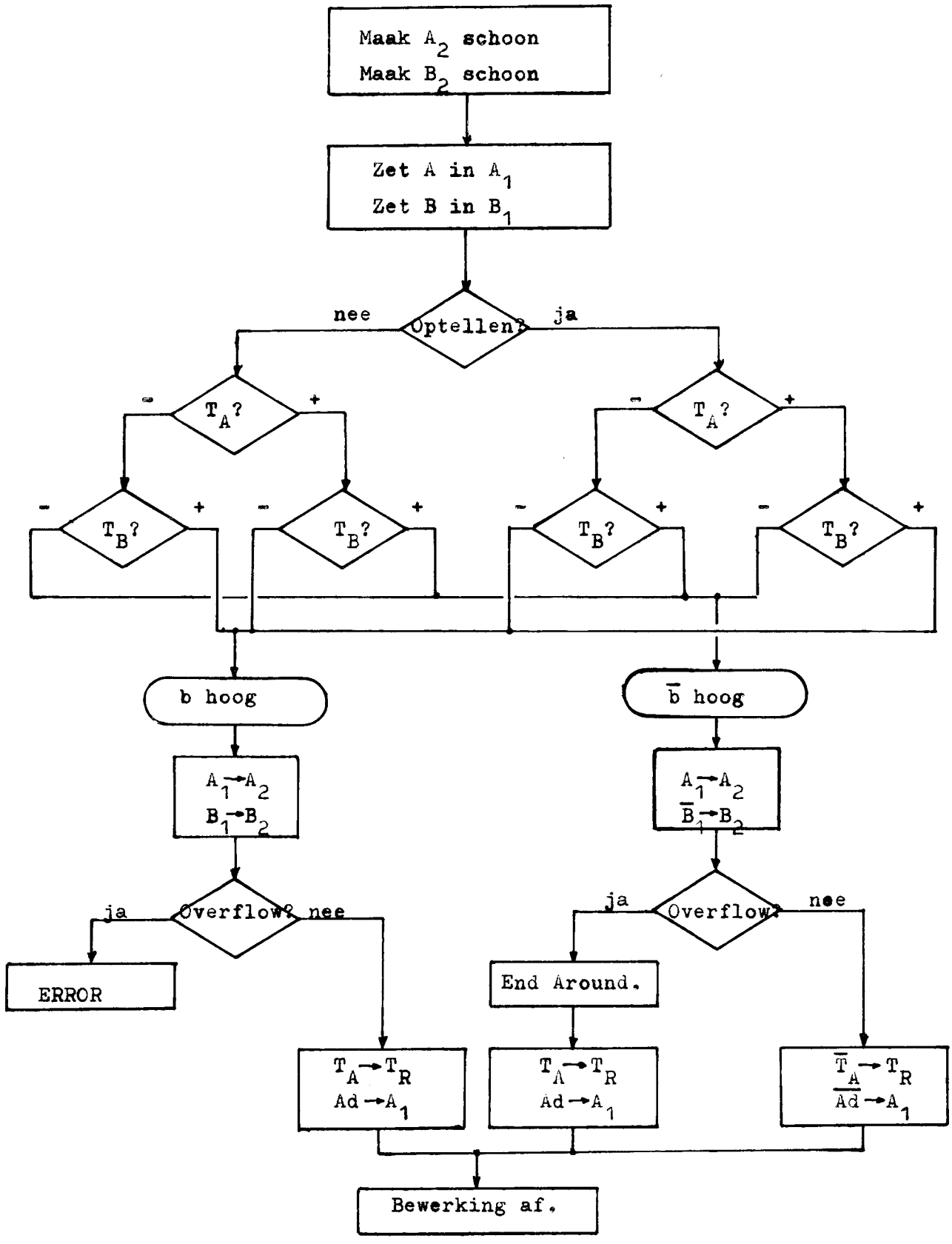


Fig.14

Allereerst worden dus de A_2 en B_2 registers schoongemaakt. Vervolgens de getallen met de tekens in de A_1 en B_1 registers geplaatst. Om nu te kunnen vaststellen of we B voor de bewerking moeten inverteren doorlopen we een vragenschema. Afhankelijk van de tekens van A en B en de soort bewerking is dus of op de b- of op de $\overline{b}$-lijn een signaal aanwezig.
Nemen we eerst het geval dat de b-lijn signaal voert.
We openen nu de poorten in N_1 en N_2 en de getallen A en B lopen van respectievelijk A_1 en B_1 naar respectievelijk A_2 en B_2. Ze worden dan meteen in de "Adder" opgeteld en het resultaat staat klaar. We vragen vervolgens of er een overflow is. Zo ja, dan wordt in N_4 (gestuurd door N_3 en Ov.) een error-signaal gegenereerd. Een overflow in deze situatie betekend immers dat het getal te groot is voor de machine-capaciteit. Treedt er geen overflow op, dan zorgt N_5 er voor dat T_R gelijk wordt aan T_A en de output van de "Adder" wordt via N_6 als resultaat R rechtstreeks met de Selector verbonden. Deze zorgt er voor dat het resultaat weer in A_1 terug geplaatst wordt.
Een van de adressen S_1 tot en met S_{16} is dus het A_1-register.
Ten slotte wordt dan aan de besturingseenheid (de micro-programmeringseenheid) gemeld dat de bewerking af is.
Nu het geval dat $\overline{b}$ hoog is. Dan wordt N_2 zo gestuurd, dat zodra een puls op m komt, niet B maar $\overline{B}$ van B_1 naar B_2 loopt.
In de "Adder" komt nu dus A + $\overline{B}$ als resultaat. De vraag die we weer stellen is: Treedt er een overflow op of niet. Zo ja, dan passen we een "end-around carry" toe, d.w.z. we forceren een carry in de nulde trap van de "Adder". Er ontstaat dus a.h.w. weer een nieuwe optelling en het nieuwe resultaat is het juiste. (Aangetoond kan worden, dat de carry-propagatie in beide optellingen samen nooit meer is dan de maximale carry-lengte die in één optelling op kan treden; d.w.z. n-1 plaatsen).
N_5 en N_6 worden nu weer zo bestuurd dat $T_R = T_A$ wordt en dat de output van de "Adder" als R aan de Selector verschijnt. Deze zendt het resultaat dan door naar A_1 en een "Bewerking af" signaal wordt gegeven.
Is er geen overflow opgetreden, dan moeten we na de bewerking nogmaals inverteren. De output van de "Adder" wordt nu via inverters in N_6 aan de Selector toegevoerd. Voor het teken geldt $T_R = \overline{T}_A$. Dit wordt in N_5 bewerkstelligd.

De Selector voert het resultaat aan A_1 toe en de bewerking is af.
Ter illustratie is hieronder van elke soort een voorbeeld gegeven (n = 4 aangenomen).

A	+	1001		-	1010		+	1010		+	1001
B	+	1010	+	-	0011	+	-	0011	+	-	1010	+
	+	1 0011		-	1101			⇓			⇓
		↓ ERROR			↓		+	1010		+	1001
				$T_R=T_A$	R=Add		-	1100		-	0101
							+	10110		+	1110
								carry			⇓
							+	0111		-	0001
								↓			↓
							$T_R=T_A$	R=Add		$T_R=\overline{T_A}$	$R=\overline{Add}$

Op deze wijze kan dus opgeteld en afgetrokken worden. Vergelijken berust is wezen ook op aftrekken. We willen daarbij echter meestal dat het resultaat niet in de registers terecht komt, want we zijn alleen geïnteresseerd in het teken van het resultaat. In de beschreven schakeling kan dit vrij eenvoudig bereikt worden door aan de Selector geen klokpuls toe te voeren. Het resultaat R vernietigt dan ook nergens de informatie in een register terwijl het zelf bij de volgende bewerking weer verdwijnt.
Een andere bewerking die in dit verband nader beschouwd kan worden is het schuiven. We hadden daarbij als eis gesteld, dat in één stap over meerdere plaatsen geschoven zou moeten kunnen worden. Wel nu, dit kan op elegante wijze m.b.v. de Selector bereikt worden.
Stel, we willen een getal over twee plaatsen naar rechts verschuiven. We plaatsen (na A_2 en B_2 schoongemaakt te hebben) het getal nu in het A_1-register terwijl in het B_1-register nullen ingevoerd worden. We laten $A + B \equiv A + 0$ nu in de "Adder" lopen en het resultaat A verschijnt aan de Selector.

D.m.v. de besturing m kiezen we nu als uitgang van de Selector S_3, waarbij aangenomen is dat S_3 verbonden is met A_1, met de restrictie dat elke bit-aansluiting over twee plaatsen naar rechts verschoven is. Het gevolg is, dat na de rondgang A weer in A_1 staat en wel over het gewenste aantal posities verschoven. Daar de Selector 16 posities heeft, kunnen we de informatie dus naar willekeur over een aantal plaatsen verschuiven. We hebben de posities S_1 tot en met S_{16} nu als volgt gebruikt:
7 posities waarbij R in A_1 geplaatst wordt en wel over respectievelijk -2, -1, 0, 1, 2, 3 en 4 plaatsen naar rechts verschoven,
7 posities om hetzelfde voor B_1 te doen.
2 posities voor andere adressen, zoals bijv. geheugenregisters e.d.
Zoals in het vorige hoofdstuk gezegd, worden de operaties gestuurd door een microprogramma. De voor de hier beschreven bewerkingen benodigde micro-operaties zijn in de volgende lijst samengevat.

1. Cl (A_2)
2. Cl (B_2)
3. Ld (A_1)
4. Ld (B_1)
5. ERROR
6. COMPL.
7. $(A_1) \longrightarrow (A_2)$
8. $(B_1) \longrightarrow (B_2)$ of $(\overline{B}_1) \longrightarrow (B_2)$
9. $(R) \longrightarrow (A_1)_{-2}$
10. $(R) \longrightarrow (A_1)_{-1}$
11. $(R) \longrightarrow (A_1)_{0}$
12. $(R) \longrightarrow (A_1)_{+1}$
13. $(R) \longrightarrow (A_1)_{+2}$
14. $(R) \longrightarrow (A_1)_{+3}$
15. $(R) \longrightarrow (A_2)_{+3}$
16. $(R) \longrightarrow (B_1)_{-2}$
17. $(R) \longrightarrow (B_1)_{-1}$
18. $(R) \longrightarrow (B_1)_{0}$
19. $(R) \longrightarrow (B_1)_{+1}$

20. $(R) \longrightarrow (B_1)_{+2}$

21. $(R) \longrightarrow (B_1)_{+3}$

22. $(R) \longrightarrow (B_1)_{+4}$

23. $(R) \longrightarrow (M)_1$

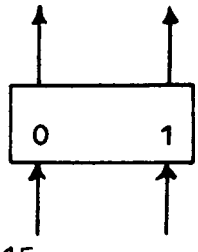
24. $(R) \longrightarrow (M)_2$

Hierbij betekent (X) de informatie-inhoud van X;
De eerste twee zijn de schoonmaak operaties, de volgende twee de operaties om de informatie in de registers te plaatsen. Daarna de error en de "Einde bewerking" operatie. De laatste 18 zijn alle transporteer operaties waarvan 9 tot en met 24 betrekking hebben op de Selector.
De indices $(\ )_i$ betekenen daarbij dat de informatie over i plaatsen naar rechts verschoven is. Operatie 23 en 24 betekent het transporteren van de informatie R naar twee nog nader te kiezen adressen.

7.2. De Logische Netwerken

We zullen nu de blokken, die in figuur 13 als geheel gegeven zijn nader uitwerken.

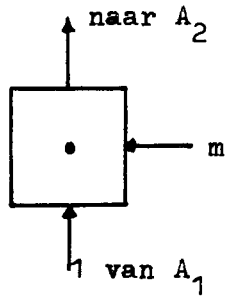
a. De registers A_1, B_1, A_2 en B_2 bestaan elk uit n RS-flip-flops, terwijl T_R en T_B elk uit één RS-flip-flop bestaan.



Figuur 15.

Symbolisch kunnen we ons de registers dus uit figuur 15 opgebouwd denken.

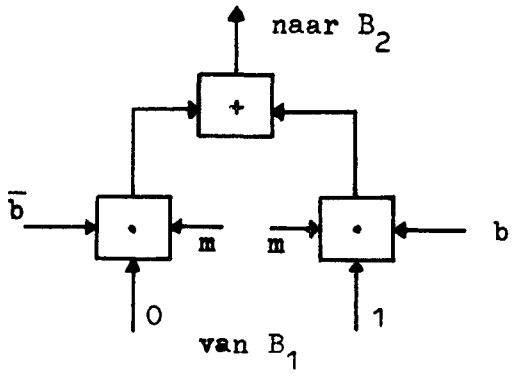
b. Het logisch netwerk N_1 bestaat uit n puls-en-poorten.



Figuur 16.

De informatie van A_1 wordt dus door de micro-operatie-puls m doorgegeven naar A_2. In figuur 16 is dit symbolisch aangegeven. Alleen aan de 1-uitgang van A_1 zit een poort !

c. N_2 is zoals in figuur 17 aangegeven. Hierbij wordt dus, afhankelijk van het signaal b of $\overline{b}$, òf B of $\overline{B}$ in B_2 geplaatst. Beide uitgangen van de B_1-flip-flop worden dus gebruikt. In totaal bestaat N_2 dus uit n circuits zoals in figuur 17 aangegeven.



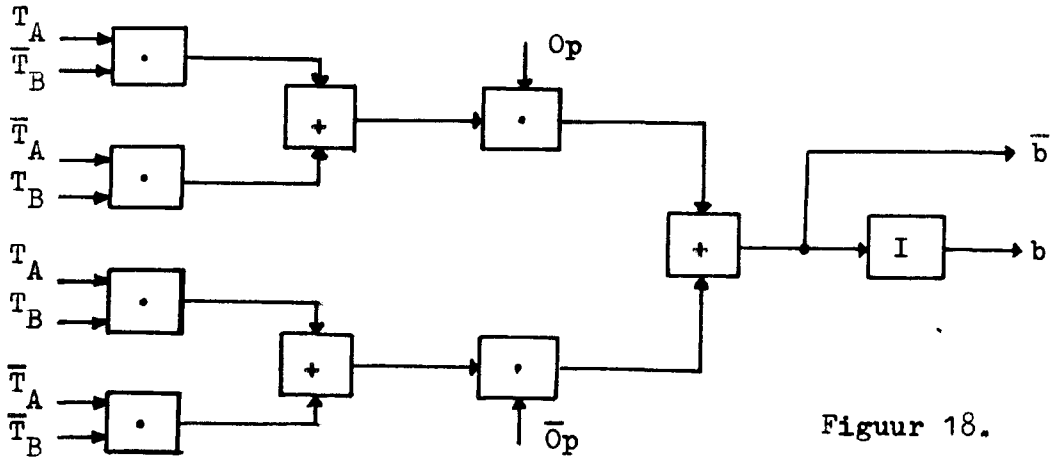
Figuur 17.

d. Het netwerk N_3. Geven we opteller aan door Op. en aftrekken door $\overline{Op}$. dan kunnen voor b en $\overline{b}$ de volgende vergelijkingen afgeleid worden (L 9.1.).

$$b = Op.\ (T_A \cdot T_B + \overline{T}_A \cdot \overline{T}_B) + \overline{Op}.\ (T_A \cdot \overline{T}_B + \overline{T}_A \cdot T_B)$$

$$\overline{b} = Op.\ (T_A \cdot \overline{T}_B + \overline{T}_A \cdot T_B) + \overline{Op}.\ (T_A \cdot T_B + \overline{T}_A \cdot \overline{T}_B)$$

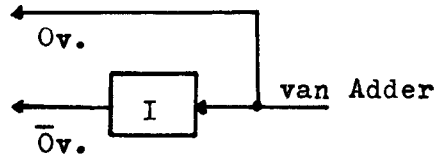
Waarbij T betekent dat het teken positief is en $\overline{T}$ dat het negatief is.



Figuur 18.

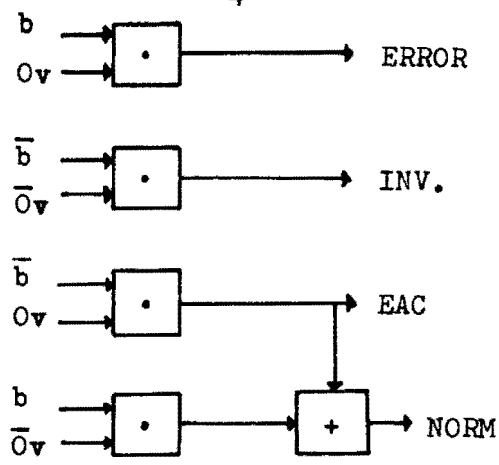
Het schema dat dit realiseert is in figuur 18 weergegeven. Hierbij is gebruik gemaakt van een invertor om uit b te komen tot $\overline{b}$. Dit netwerk is dus eenmaal benodigd.

e. Het blok Ov. bestaat uit een enkele inverter. Op deze wijze (zie figuur 19) wordt dus bereikt dat dus of een "overflow-signaal" of een "geen overflow-signaal" aan de uitgang beschikbaar is.



Figuur 20.

f. Het circuit N_4. We gaan uit van figuur 14 en kunnen dan de volgende logische vergelijkingen opstellen.

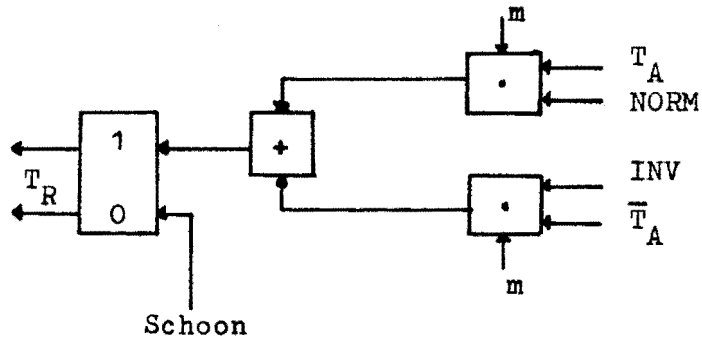


Figuur 20.

ERROR = b. Ov.
INV = $\overline{b}$. $\overline{Ov}$.
EAC = $\overline{b}$. Ov.
NORM = $\overline{b}$. Ov. + b. $\overline{Ov}$.

Hierbij betekent INV dat we het inverse van de "Adder output" aan de Selector moeten toevoeren en dat bovendien $T_R = \overline{T}_A$. Verder is EAC de afkorting van "end-around-carry" en NORM wil zeggen dat we de output van de "Adder" rechtstreeks met de Selector verbinden en dat ook $T_R = T_A$. Het schema hiervoor is in figuur 20 aangegeven.

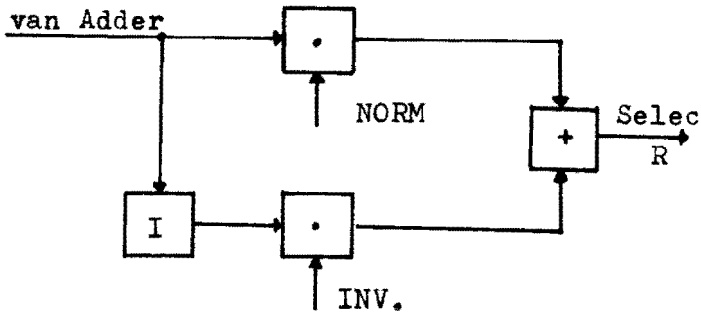
g. Vervolgens het netwerk N_5.



Figuur 21.

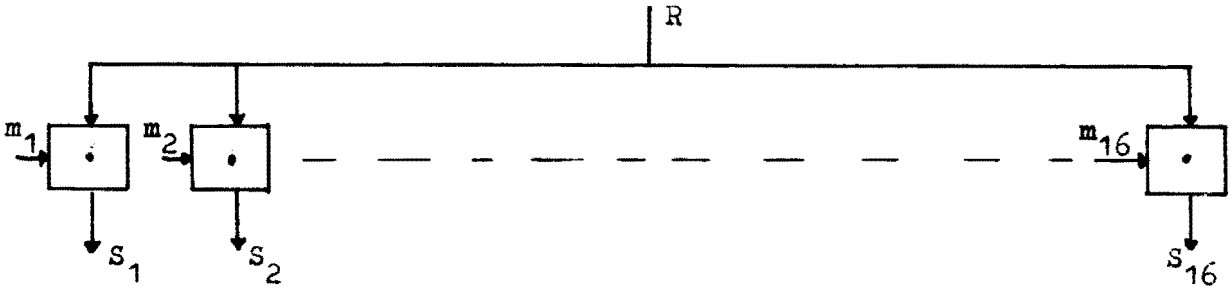
In figuur 20 is de opbouw van het circuit getekend. Tegelijkertijd met het schoonmaken van de A_2 en B_2 registers zetten we ook de flip-flop T_R op nul. Wanneer we nu de micro-programma-puls m toevoeren zijn er twee condities die in staat zijn om T_R op 1 te zetten. Deze zijn als T_A = 1 en we behoeven niet te inverteren en ook als T_A = 0 en we moeten wel inverteren.

h. De schakeling N_6. Deze bestaat uit n maal de schakeling zoals in figuur 22 aangegeven. Afhankelijk van het aanwezig zijn van een NORM- of een INV-signaal wordt dus of het output-signaal van de "Adder" of het inverse ervan met de ingang van de Selector verbonden.



Figuur 22.

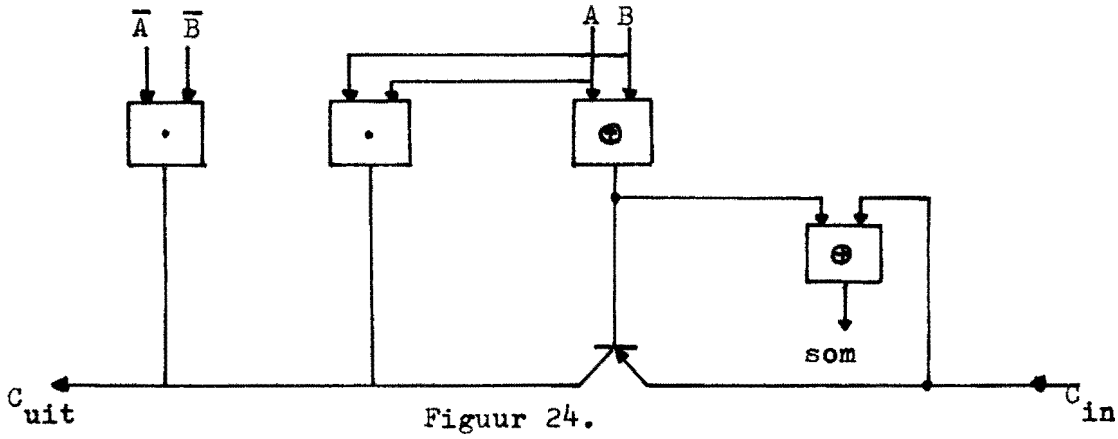
i. De Selector. Zoals reeds gezegd bestaat de Selector uit n + 1 zestien-standen schakelaar.



Figuur 23.

Eén schakelaar is in figuur 23 getekend. De informatie R stroomt afhankelijk van het stuursignaal m_i (afkomstig van het microprogramma) naar een van de adressen S_1 tot en met S_{16}.

j. De "Adder". Deze schakeling neemt de centrale plaats in bij de gehele bewerking.



Figuur 24.

Elk van de n trappen is opgebouwd volgens figuur 24.

Via twee "exclusive-or" schakelingen wordt de som bepaald. Terwijl A + B de transistor die voor de carry propagatie moet zorgen stuurt. De linkse n-poort zorgt ervoor dat de emitter van de carry-transistor van de volgende trap negatief wordt wanneer A en B beide nul zijn. De andere en-poort geeft een eventuele gegenereerde carry verder aan de volgende trap. De carry propagatietijd per trap kan hiermee tot ongeveer 3 n sec teruggebracht worden.

8. Enkele Schakelingen

8.1. Algemeen

Alhoewel het ongetwijfeld zinvol is om een logisch apparaat eerst op papier geheel in details uit te werken, is deze weg hier toch niet gevolgd. Dit hangt samen met het feit, dat er in het geheel geen ervaring beschikbaar was met schakelingen van een dergelijk hoge snelheid als die waarnaar gestreefd werd. Uit de studie kwam n.l. naar voren dat een klokfrequentie van 10 Mc als doel gesteld dient te worden. Betrouwbare circuits voor deze frequentie waren niet beschikbaar. Om nu de invloed na te gaan die het gebruik van deze zeer snelle technieken ongetwijfeld ook op het logische ontwerp heeft werd besloten enkele elementaire schakelingen te ontwerpen, te bouwen en te beproeven.
Hiervoor werden een flip-flop en de standaard schakeling voor de "Adder" gekozen.

8.2. De Flip-Flop

De schakeling zal dus met een frequentie van 10 Mc moeten kunnen werken. Dit omdat de bereikbare snelheid van de carry-propagatie methode in de grootte orde van 0,1 à 0,2 μsec. voor de gehele woordlengte ligt.
Als literatuur voor het ontwerpen van transistorschakelingen moge L 8.1.; L 8.2.; L 8.3. en L 8.4. aangevoerd worden.
Als meest geschikte techniek werd de RCTL-techniek gekozen, waarbij in eerste instantie uitgegaan zal worden van de Amerikaanse 2N1500 transistoren. Indien mocht blijken dat het gebruik van een ander type gewenst zou zijn, zal ongetwijfeld daarbij van de met de 2N1500 opgedane ervaring gebruik gemaakt kunnen worden (L 8.5.; L 8.6.; L 8.7.).

8.2.1. De Berekening

Het gelijkstroom circuit van een flip-flop heeft de volgende gedaante (figuur 25).

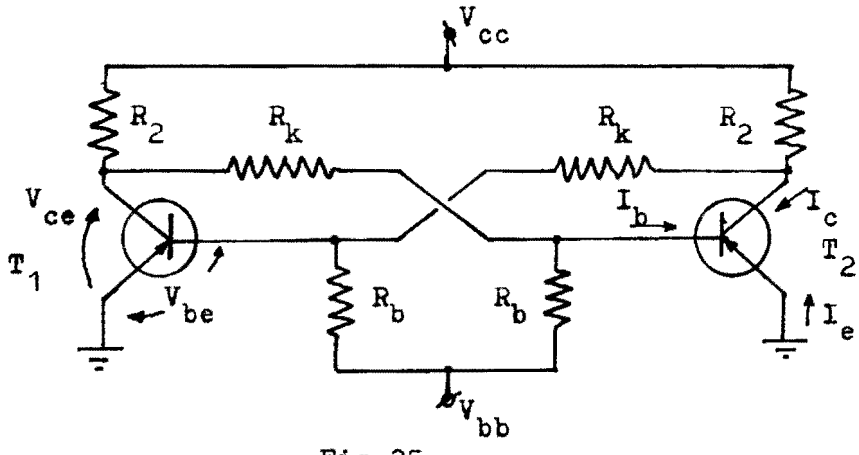


Fig.25

Als we uitgaan van een transistor waarvan we de gegevens kennen, hebben we 5 onbekenden nl. V_{cc}, V_{bb}, R_2, R_k en R_b.

We nemen nu V_{bb} en V_{cc} aan, zodanig, dat ze passen in het ontwerp als geheel en geen belemmering vormen voor het betrouwbaar werken van de flip-flop. (dus niet extreem hoog of extreem laag).
Bovendien is het gewenst $|V_{cc}| > |V_{bb}|$ te kiezen.
In ons geval kiezen we V_{bb} = + 4,5 Volt en V_{cc} = - 6 Volt.
Ook de maximale collectorstroom nemen we (met redelijkheid !) aan. We stellen $I_{c(max)}$ = 10 mA. Hieruit kunnen we dan R_2 berekenen.
En wel: $R_2 = \frac{V_{cc}}{I_{c(max)}} = \frac{6}{10^{-2}} = 600\,\Omega$. We nemen nu R_2 = 560 Ω, daar dit een gangbare waarde is.
We moeten nu nog R_k en R_b bepalen. Dit dient zodanig te gebeuren, dat de schakeling werkt als flip-flop. We gaan daartoe nu een synthese opzetten. Uitgegaan zal worden van een "on"-voorwaarde en een "off"- voorwaarde, waarbij aan beide tegelijk zal moeten worden voldaan.
Voor de "on"- voorwaarde kunnen we het volgende circuit opzetten (figuur 26).

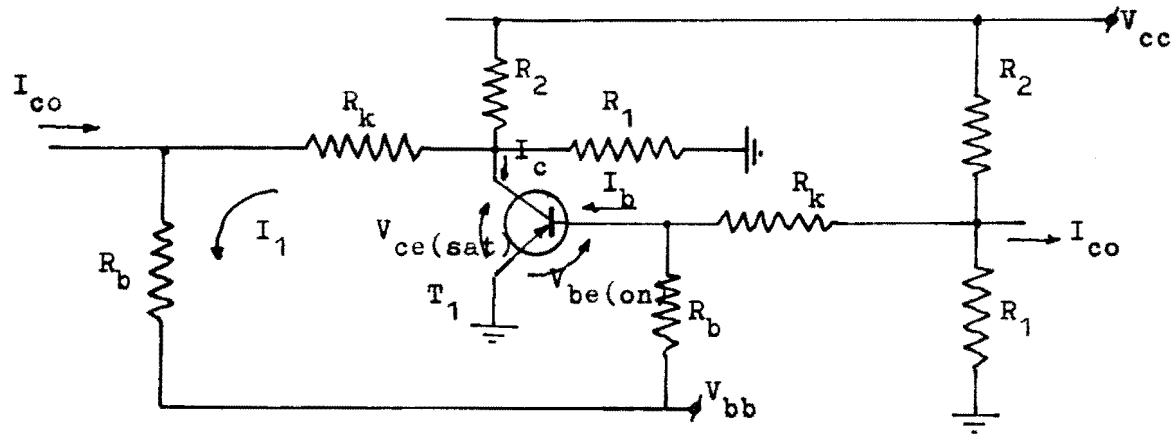


Fig.26

Hierbij stelt R_1 de uitwendige belasting voor.

Hiervoor geldt nu: $$\frac{V_{be(on)} - V_{bb}}{R_b} + I_b = \frac{\frac{V_{cc} \cdot R_1}{R_1 + R_2} - V_{be(on)}}{\frac{R_1 \cdot R_1}{R_1 + R_2} + R_k}$$

Substitueren we nu: $$I_b = \frac{I_c}{h_{fe(min)}}$$

$$I_c = \frac{\frac{V_{cc} \cdot R_1}{R_1 + R_2} - V_{ce(sat)}}{\frac{R_2 \cdot R_1}{R_1 + R_2}} - I_1$$

$$I_1 = \frac{V_{ce(sat)} - V_{bb}}{R_k + R_b}$$

Dan:

$$\frac{V_{be(on)} - V_{bb}}{R_b} + \frac{\frac{V_{cc} \cdot R_1}{R_1 + R_2} - V_{ce(sat)}}{\frac{R_1 \cdot R_2}{R_1 + R_2} \cdot h_{fe(min)}} + \frac{V_{bb} - V_{ce(sat)}}{(R_k + R_b) \cdot h_{fe(min)}} =$$

$$\frac{\frac{V_{cc} \cdot R_1}{R_1 + R_2} - V_{be(on)}}{\frac{R_2 \cdot R_1}{R_1 + R_2} + R_k}$$

De oplossing van R_b en R_k betekent in dit geval het oplossen van een kwadratische vergelijking. Ten einde dit te vermijden verwaarlozen we de derde term van het linkerlid. Daartoe dienen we ervoor te zorgen dat deze inderdaad verwaarloosbaar is. We nemen hier aan, dat hij daartoe $\leqslant$ 3 % van de tweede term dient te zijn, m.a.w.

$$R_b + R_k \geq \left| \frac{\frac{R_1 \cdot R_2}{R_1 + R_2} (V_{ce(sat)} - V_{bb})}{0{,}03 \left(\frac{V_{cc} \cdot R_1}{R_1 + R_2} - V_{ce(sat)}\right)} \right|$$

Is hier aan voldaan dan wordt de "on"-vergelijking:

$$\frac{V_{be(on)} - \overline{V_{bb}} + \dfrac{\dfrac{V_{cc} \cdot R_1}{R_1 + R_2} - V_{ce(sat)}}{\dfrac{R_1 \cdot R_2}{R_1 + R_2} \cdot h_{fe(min)}}}{R_b} = \frac{\dfrac{V_{cc} \cdot R_1}{R_1 + R_2} - V_{be(on)}}{\dfrac{R_2 \cdot R_1}{R_1 + R_2} + R_k}$$

Hierin geldt: $\overline{x} = 1{,}1 \cdot X_{nom.}$ en $\underline{x} = 0{,}9 \cdot X_{nom.}$

Dit betekent, dat we ten einde een "worst case design" te hebben, een tolerantie van 10 % in de voedingsspanningen en de weerstanden incalculeren.

Voor de gegeven transistor 2 N 1500 gelden nu de volgende specificaties:

$I_{co} = -5$ A

$V_{be(on)} = -0{,}4$ Volt

$V_{ce(sat)} = -0{,}2$ Volt

$h_{fe(min)} = 20.$

Dit ingevuld levert dan:

$$\frac{-5{,}95}{R_b} + \frac{123{,}2 - 4{,}67 . R_1}{11088 . R_1} = \frac{246 - 4{,}5 . R_1}{553{,}4 . R_1 + 0{,}99 . R_1 . R_k + 677{,}6 . R_k}$$

Vervolgens de "off"-voorwaarde.
Het daarbijbehorende circuit is (figuur 27):

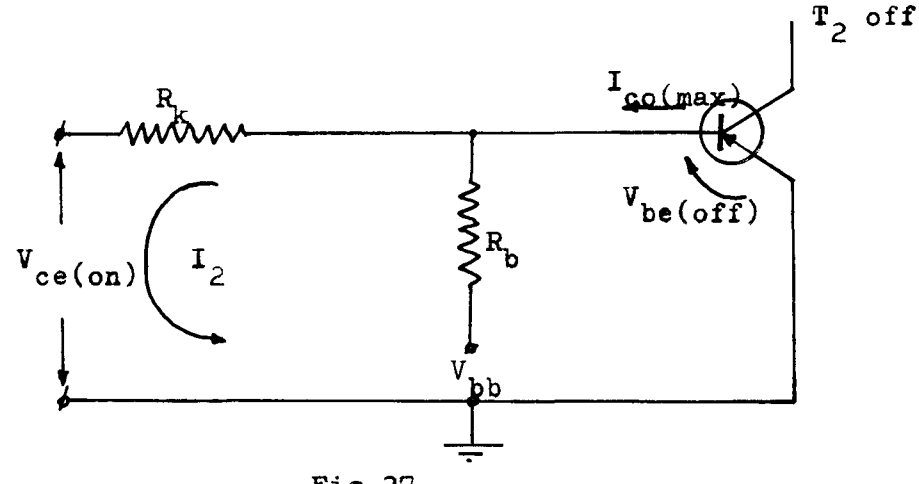


Fig.27

Er geldt:
$$\begin{cases} I_2 = \dfrac{(V_{bb} + I_{co(max)} \cdot R_b) - V_{ce(sat)}}{R_b + R_k} \\ V_{be(off)} = V_{bb} - (I_2 - I_{co(max)}) \cdot R_b \end{cases}$$

of:
$$V_{be(off)} = \underline{V_{bb}} - \frac{\underline{V_{bb}} - V_{ce(sat)} - I_{co(max)} \cdot \underline{R_k}}{\overline{R_b} + \underline{R_k}} \cdot \overline{R_b}$$

Ook hier is weer rekening gehouden met de toleranties.
In ons geval is: $V_{be(off)} = + 0{,}2$ Volt

Dit leidt tot: $R_b + 1125.10^{-8} \cdot R_k \cdot R_b - 7{,}875 \cdot R_k = 0$

In de grafiek figuur 28, zijn de betrekkingen tussen R_b en R_k uitgezet voor de "on"- en de "off"-conditie. Hierbij is de waarde van R_1 op 10 kΩ gesteld.

Opdat de juiste werking van de flip-flop verzekerd zal zijn, dient het gekozen werkpunt in het gearceerde gedeelte te liggen.
Ten einde niet te grote stromen te krijgen is het gewenst de R-waarden niet te klein te kiezen. Als werkpunt werd daarom in eerste instantie R_k = 8 k2 en R_b = 27 k gekozen.

De amplitude van het uitgangssignaal van een flip-flop is:
$$V_{outp.} = V_{cc} - I_{co(max)} \cdot R_2 - \left[\frac{V_{cc} - I_{co(max)} \cdot R_2 - V_{be(on)}}{R_2 + R_k}\right] \cdot R_2$$
$$- V_{ce(sat)}.$$

In het onderhavige geval leidt dit tot: $V_{outp.} = 5{,}4$ Volt.
De uitgangsspanning loopt dus van -0,6 Volt tot - 6 Volt.
Ten einde voor het schakelen een snellere responsie te verkrijgen, gaan we de koppelweerstanden R_k overbruggen door een "speed-up" condensator C_k.

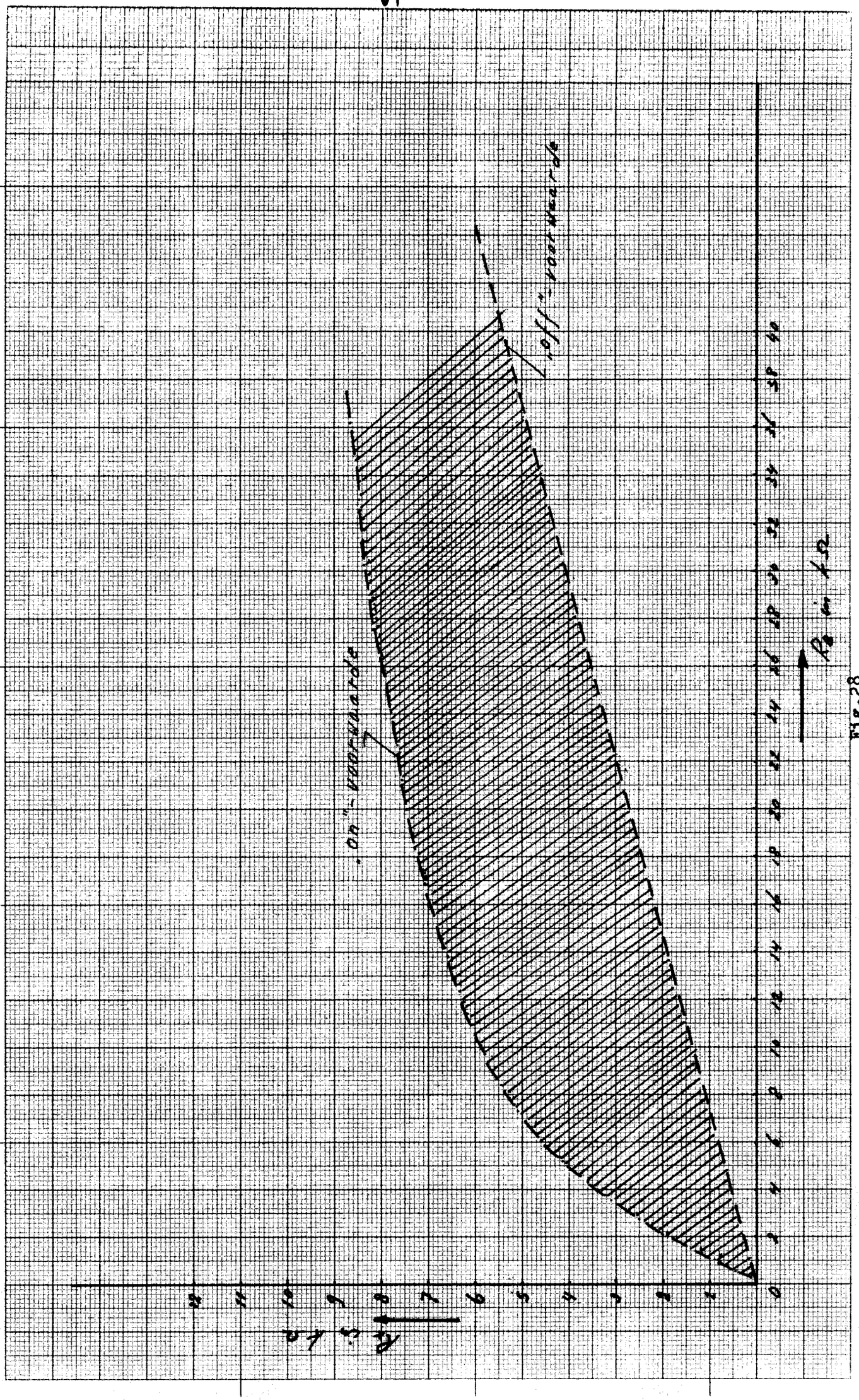


Fig.28

We moeten de waarde van C_k echter zo kiezen, dat de tijdconstante die de C_k met de weerstanden in de schakeling vormt niet te groot is. Verwaarlozen we de inwendige weerstanden van de transistoren, dan volgt voor C_k de voorwaarde:

$$C_k \approx \frac{R_k + R_b}{R_k \cdot R_b} \cdot \frac{1}{\text{max. frequentie}}$$

We willen een schakeling, die geschikt is voor 10 Mc/s. Dit levert ons: $C_k \approx 17$ pF.
We nemen nu $C_k = 22$ pF.
Het complete schema is nu dus (figuur 29).

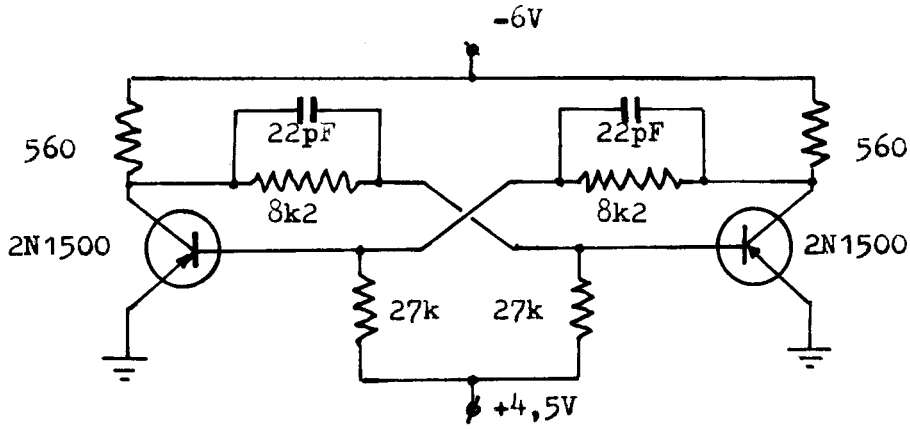


Fig.29

8.2.2. De Triggering

Ten einde een enkele flip-flop te kunnen testen is een T-schakeling ontworpen. De uiteindelijke flip-flop wordt dan zoals in figuur 30.

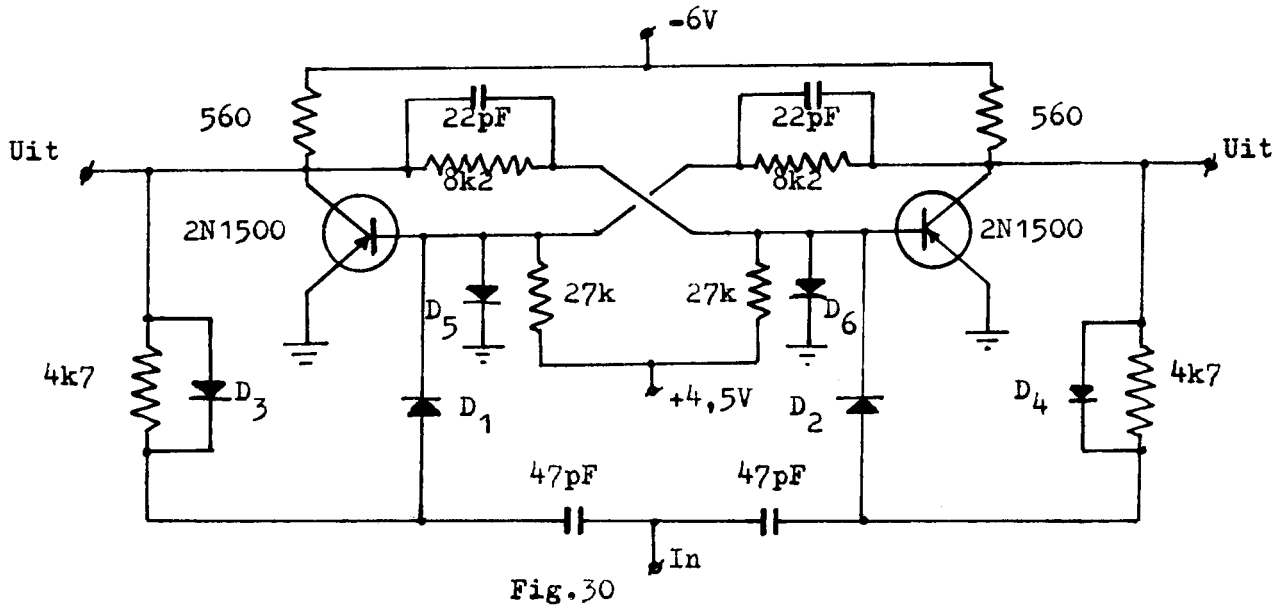


Fig.30

De ingangspuls wordt dus m.b.v. het netwerk bestaande uit de condensator (47 pF) en de weerstand (4 k7) gedifferentieerd en via respectievelijk D_1 en D_2 aan de bases toegevoerd. Hierbij is ervoor zorg gedragen, dat het produkt 47 pF x 4 k7 beneden $\frac{1}{f_t}$ ligt, met f_t is de maximale triggerfrequentie (hier dus 10 Mc).
Ten einde de triggering nog verder te versnellen en ook betrouwbaarder te maken zijn de "speed-up diodes" D_3 en D_4 aangebracht. Ook D_5 en D_6 dienen hetzelfde doel, daar ze de basislading snel afvoeren.
Voor D_1 en D_2 zijn diodes van het tyre S 555 G gebruikt voor D_3 tot en met D_6 van het type HG 1005.

8.2.3. De Resultaten

In figuur 31 is het fotografisch opgenomen beeld van in- en uitgangssignaal weergegeven.

Voor de verticale as geldt als schaal 5 V/cm en voor de horizontale 0,05 sec./cm.
Deze resultaten zijn gemeten aan een opstelling zoals in figuur 32 weergegeven.

Fig.31

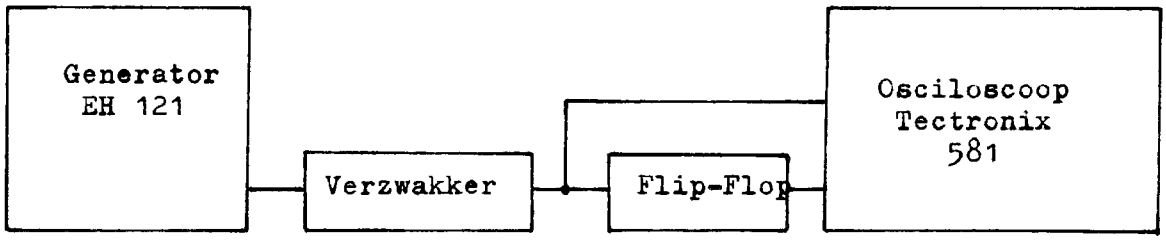


Fig.32

De ingangspuls is dus ongeveer 2 V hoog en 20 nsec. breed.
De pulsherhalingsfrequentie is 10 Mc.
Voor het outputsignaal geldt: almplitude ruim 5 V; delay-time 10n sec.; rise-time 5 n sec.; storage-time niet meetbaar; fall-time 30n sec.
Niettegenstaande door omstandigheden nog geen uitgebreide tests konden plaatshebben kan uit de reeds verkregen resultaten worden afgeleid, dat de schakeling zeer goed aan de gestelde eisen voldoet. Er werden ook spanningsvariatie-tests uitgevoerd; transistoren met hoge en met lage h_{FE} geprobeerd, kleine temperatuurtesten gedaan, etc. Bij dit alles werd, voor zover de onnauwkeurigheid van de tests een oordeel toelaat, bevonden dat aan alle eisen bleef voldaan. Ook variaties in de ingangspulsen hadden, binnen zekere grenzen m.b.t. steilheid, geen ongunstige invloed. Een weerstandsbelasting van 1 K had wel een afname van de outputspanning tot gevolg maar geen invloed op de stabilitiet.
Alhoewel uitgebreidere tests absoluut noodzakelijk zullen zijn, kan afsluitend gezegd worden dat de schakeling zeer goed aan de gestelde eisen voldeed.

8.3. De "Adder"-schakeling

Onder 7.2. ad. j., is de logische "lay-out" van de schakeling, het eerst gebracht door Salter (L 8.8.), gegeven. Het blokschema is zoals figuur 24 aangeeft. Het elektrische schema wordt dan zoals in figuur 33.

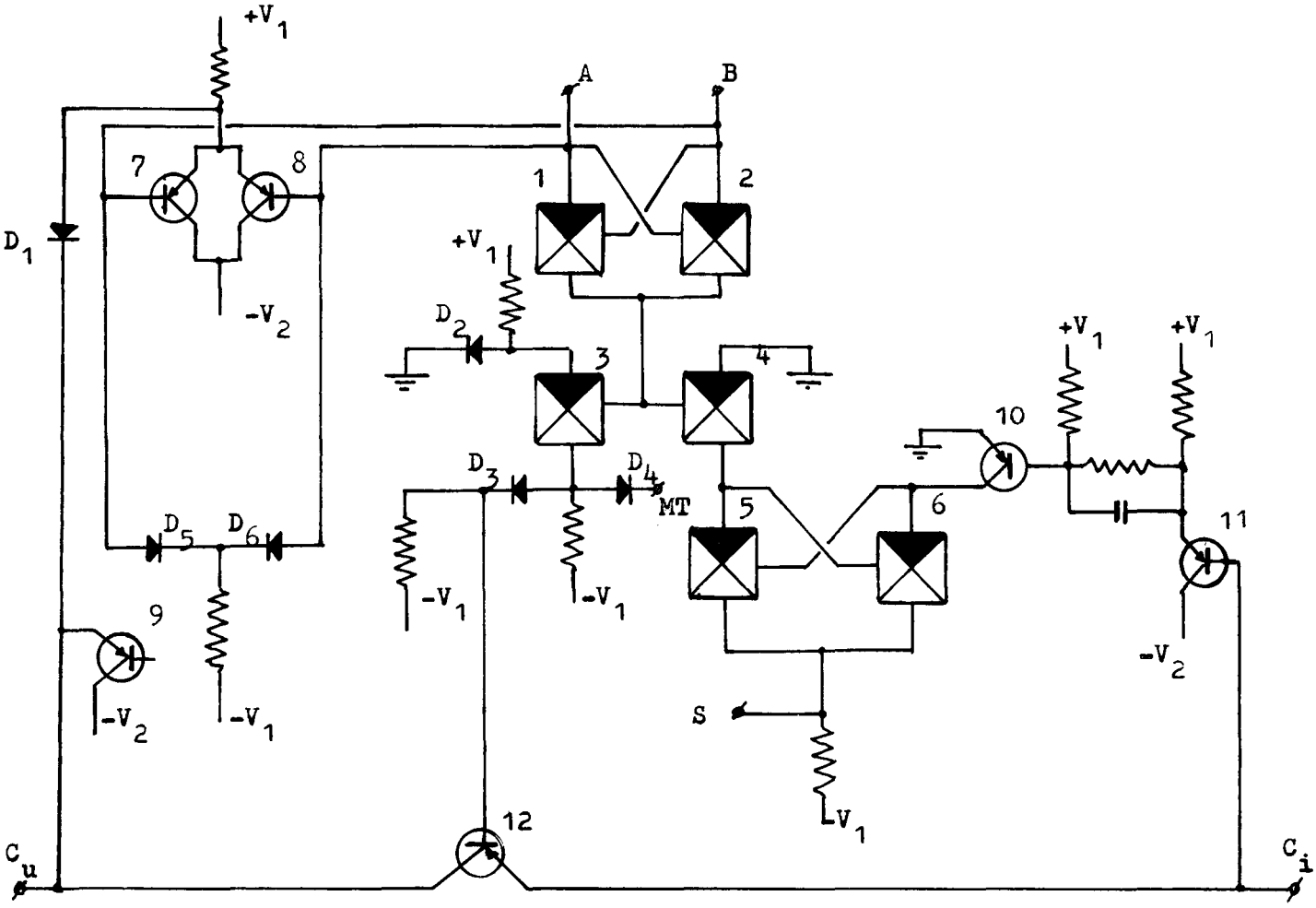


Fig.33

Hierbij dient men ook figuur 34 in gedachten te hebben.

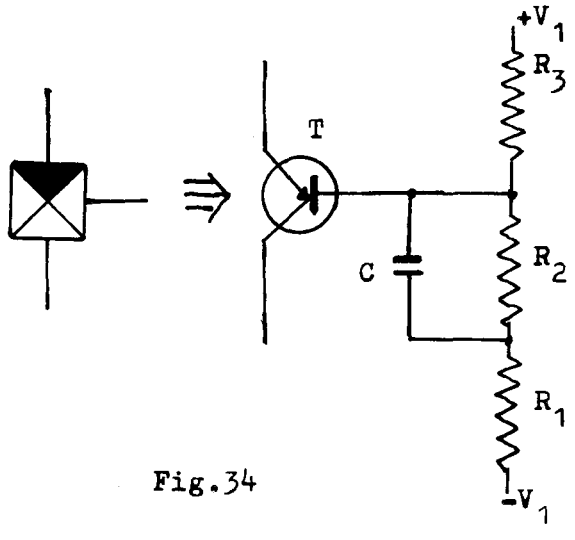


Fig.34

Het circuit bestaat dus uit 12 transistoren en 6 diodes met bijbehorende weerstanden en condensatoren. Links boven in figuur 33 wordt m.b.v. de transistoren 7 en 8 een gegenereerde carry via D_1 aan de "carry-line" doorgegeven.

Dit uiteraard alleen als A en B beide 1 zijn, want dan zijn 7 en 8 gesperd.
Zijn A en B beide nul, dan zijn D_5 en D_6 beide gesperd en de emitter-volger 9 open. Daardoor wordt de "carry-line" negatief en er wordt een C_u doorgegeven. Dit waren dus de getallen, dat A en B gelijk waren. Is dit niet zo, dan levert het "exclusive-or" circuit bestaande uit de blokken 1 en 2 een output. Via het blok 3 en D_3 wordt dan de transistor 12 geopend. Een eventueel inkomende carry C_i kan dus doorlopen naar de volgende trap.
Eveneens wordt via 4 het "exclusive-or" circuit bestaande uit 5 en 6 gestuurd. De andere input van dit circuit is de inkomende carry.
De emitter-volger 11 voorkomt daarbij een te grote belasting van de "carry-line". Transistor 10 is een herstelschakeling voor het carry-signaal. De som S wordt dus gevormd door de "exclusive-or" schakeling bestaande uit 5 en 6. Dus $S = (A \oplus B) \oplus C_{in}$.
Op de ingang van D_6 kan een signaal gezet worden zodanig dat alle transistoren 12 gesperd worden. Het ontstane resultaat is dan gevormd volgens de regels van de modulus-twee optelling. Ook vormen alle n stuks D_6 een en-poort die de carry-line bestuurt als alle A_i en B_i verschillend zijn. Dit omdat er minstens één signaal moet zijn dat de carry-line bestuurt.

8.3.1. De Berekening van de "Standaard Schakeling"

Om reden van het onder 8.1. aangegevene zal nu een schakeling volgens figuur 34 berekend worden. Deze schakeling komt in elke trap van de "Adder" zes maal voor en is daarom standaard schakeling genoemd. Ook hier zullen we weer RCTL gebruiken en als transistor weer de 2 N 1500 nemen. Het gelijkstroomcircuit is in figuur 35 aangegeven.
Ook hier nemen we weer een tolerantie van 10 % in de voedingsspanningen en de weerstandswaarden aan.
Dit in aanmerking nemende kunnen de volgende vergelijkingen opgezet worden.

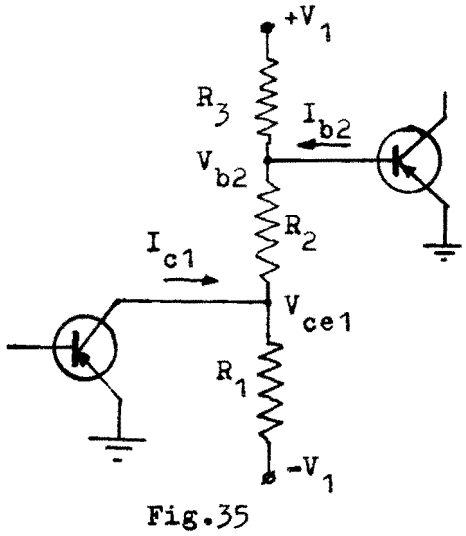


Fig.35

$$\begin{cases} I_{B_2}^{on} = \dfrac{V_{B_2}^{on} + V_1}{R_1 + R_2} + \dfrac{V_{B_2}^{on} - V_1}{R_3} \\ V_{B_2}^{off} = \dfrac{V_1 \cdot R_2 + V_{CE_1}^{on} \cdot R_3}{R_3 + R_2} \\ I_{C_1}^{on} = \dfrac{V_{CE_1}^{on} + V_1}{R_1} + \dfrac{V_{CE_1}^{on} - V_1}{R_3 + R_2} \end{cases}$$

De volgende waarden worden nu gesubstitueerd:

$V_1 = 6$ V

$V_B^{on} = -0{,}4$ V

$V_B^{off} = +0{,}2$ V

$V_{CE}^{on} = -0{,}2$ V

Het resulaat is dan:

$$\begin{cases} I_{B_2}^{on} = \dfrac{4{,}5}{R_1+R_2} - \dfrac{7{,}8}{R_3} \quad (\text{"on" - vergelijking}) & (1) \\ I_{C_1}^{on} = \dfrac{7{,}1}{R_1} - \dfrac{5{,}1}{R_2+R_3} & (2) \\ R_3 = 10{,}7 \cdot R_2 \quad (\text{"off" - vergelijking}) & (3) \end{cases}$$

Nemen we nu een $I_{B_2}^{on} = 0{,}5$ mA aan, dan zijn de "on"- en "off" - vergelijkingen respectievelijk als volgt te schrijven:

$$\begin{cases} R_3 = \dfrac{R_1 + R_2}{0{,}58 - 0{,}64.10^{-4}(R_1 + R_2)} \\ R_3 = 10{,}7 \cdot R_2 \end{cases}$$

In figuur 36 zijn deze beide vergelijkingen voor $R_1 = 1$ K uitgezet. We kiezen als werkpunt het punt: $R_2 = 1$ K5 en $R_3 = 12$ K.

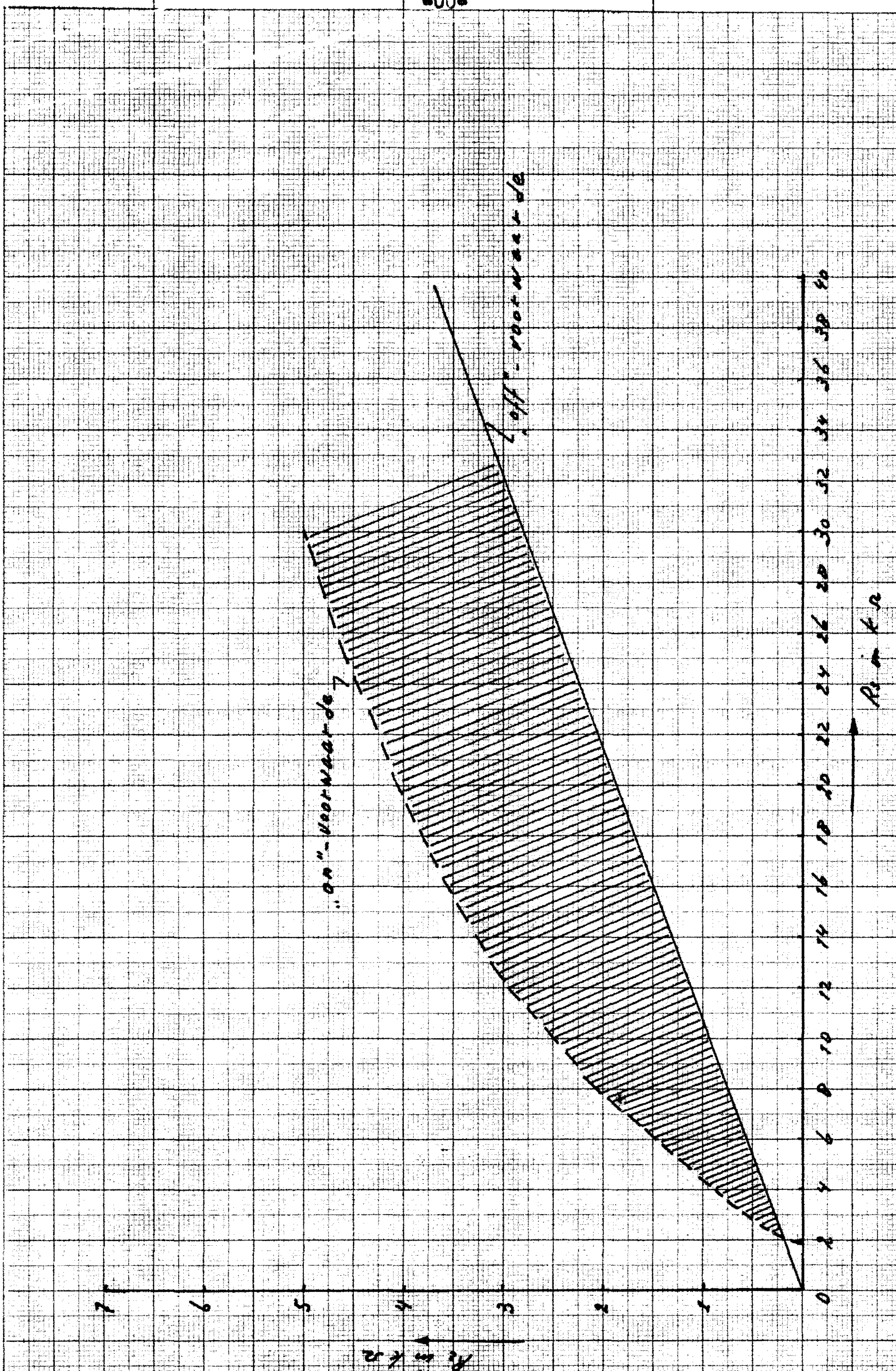


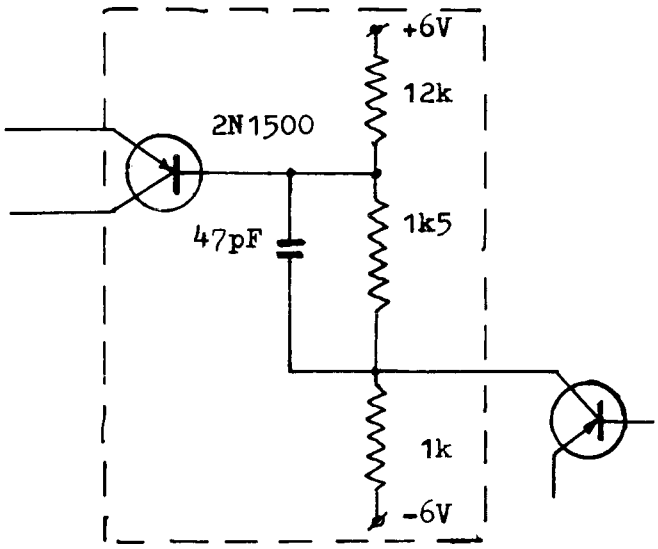
Fig. 36

Voor $I_{C_1}^{on}$ volgt dan: $I_{C_1}^{on} = 6{,}7$ mA. Dit is met een $h_{FE}(min)$ van 20 zeker te bereiken.
Immers $I_B^{on} = 0{,}5$ mA. En dus $I_B^{on} \cdot h_{FE} = 10$ mA.

Voor de condensator over R_2 geldt weer:

$$C_k \leqslant \frac{R_2 + R_3}{R_2 \cdot R_3} \cdot \frac{1}{t_{max}} \qquad \text{Hieruit} \quad C_k \leqslant 48 \text{ pF.}$$

We nemen $C_k = 47$ pF.



Het uiteindelijke circuit wordt dan zoals in figuur 37 is aangegeven.
Het gedeelte binnen de streepjeslijnen is hierbij de eigenlijke standaard schakeling.

Fig.37

8.3.2. De Resultaten

In figuur 38 zijn de in- en uitgangssignalen weer fotografisch weergegeven. Horizontaal 50n sec/cm en verticaal 5 V/cm.

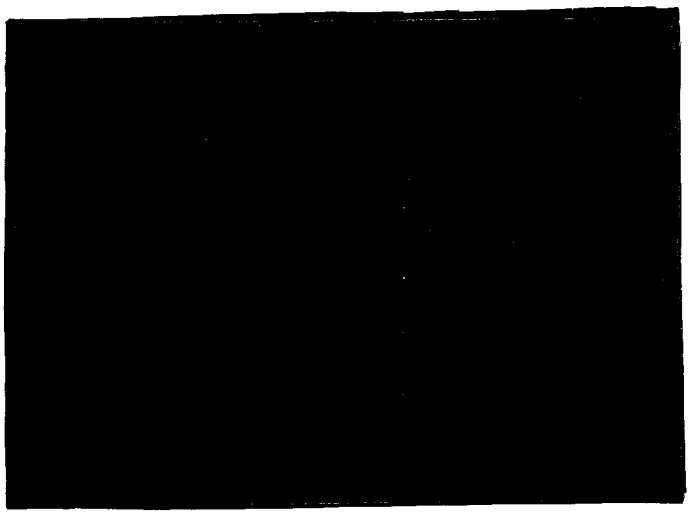


Fig.38

Bij een ingangssignaal van ongeveer 2 Volt ontstaat dus een uitgangssignaal van 5 V.
De flanksteilheid van de uitgangspuls is zeer goed te noemen (ongeveer 10n sec.).
De afvaltijd is t.g.v. de storage wat langer. Ook de "delay-time" is in de orde van 5n sec.

De gebruikte pulsherhalingsfrequentie was ook hier weer 10 Mc. Het gedrag van de schakeling bij deze frequentie was zeer bevredigend te noemen.
Ten slotte zijn ook nog enkele proefjes gedaan, waarbij de standaardschakeling de flip-flop stuurde en omgekeerd. Ook hierbij bleek een betrouwbaar gedrag op te treden. Effecten die hierbij wel zeer duidelijk aan de dag traden waren stralingseffecten t.g.v. de hoge frequentie. Ten gevolge hiervan traden allerlei secundaire vervormingen op. De werking bleef echter verzekerd maar de signalen waren minder fraai van vorm. Het is dus wel zaak in de toekomst zeer korte verbindingen e.d. te gebruiken om dit soort verschijnselen tegen te gaan.
In verband met deze voorwaarde en ook als gevolg van de eis naar steeds verder gaande miniaturisatie zal getracht worden de schakelingen zeer compact samen te bouwen.
Gestreefd wordt naar enkele cm^3 per schakelelement.

9. Besluit

De schrijver stelt het op prijs zijn erkentelijkheid te betuigen voor de vele hulp en medewerking die hij mocht ontvangen.
Op de eerste plaats wel van Prof. Heetman, die met zijn stimulerende en enthousiaste ideeën er steeds weer toe bijdroeg dat een oplossing voor de problemen gevonden werd.

Verder mijn collega's, de heren Stenbit en Engel, die door hun enerzijds stimulerende en anderzijds critiserende opmerkingen er steeds weer voor zorgden dat de ideeën niet een te fantastische omvang kregen. De discussies met hen hebben zeer zeker hun invloed op het beschreven werk gehad.

Ook de heer Weijland is de schrijver veel dank verschuldigd. De wijze waarop hij de als een plaag op hem neerdalende vragen naar materiaal en instrumenten wist op te vangen, heeft een effectieve uitwerking van de plannen en concepten sterk bevorderd.

Als laatste maar zeker niet als minste dient mej. Donkers vermeld te worden. De onvermoeibare wijze waarop zij het typewerk en de verdere bewerking van het onderhavige rapport verzorgd heeft getuigt van veel geduld en begrip.
Zonder haar hulp zouden vele problemen niet opgelost zijn.

10. Literatuur

a. Algemeen

L 0.1. Digital Computer and Control Engineering.
R.S. Ledley, McGraw-Hill, New York, 1960.

L 0.2. Logical Design of Digital Computers.
M. Phister, Jr., Wiley and Sons, New York, 1958.

L 0.3. Switching Circuits with Computer Applications.
W.S. Humphrey, Jr., McGraw-Hill, New York, 1958.

L 0.4. Planning a Computer System.
W. Buchholz, McGraw-Hill, New York, 1962.

L 0.5. A survey of Switching Circuit Theory.
E.J. McCluskey, Jr. and T.C. Bartee, McGraw-Hill, New York, 1962.

L 0.6. A New Approach to the Design of Switching Circuits.
H.A. Curtis, van Nostrand, Princeton, 1962.

L 0.7. Digital Computer design Fundamentals.
Y. Chu, McGraw-Hill, New York, 1962.

L 0.8. Taschenbuch der Nachrichtenverarbeitung.
K. Steinbuch, Springer, Berlin, 1962.

L 0.9. Digitale Rechenanlagen.
A.P. Speiser, Springer, Berlin, 1961.

b. Nieuwe ontwikkelingen

L 1.1. Trends in Design of Large Computer Systems.
C.W. Adams, Proceeding W.J.C.C. 1961, p.p. 362-364.

L 1.2. A Positive-Integer Arithmetic for Data Processing.
R.W. Murphy, IBM J. Res. and Dev., Vol. 1, p.p. 158-170.

L 1.3. An Algorithn for Coding Efficient Arithemetic Operations.
R.W. Floyd, Comm. A.C.M., vol. 4, p.p. 42-51.

L 1.4. Logical Design for Digital Computers.
R.V. Norton, Western Elec. Engr., Vol. 2, p.p. 28-36.

L 1.5. The Future of Automatic Digital Computers.
A.D. Booth, Comm. A.C.M., Vol. 3, p.p. 339-341.

L 1.6. Logical Design Methods.
R.K. Richards, Proc. W.J.C.C. 1958, p.p. 179-181.

L 1.7. B 5000 Manual.
Burroughs Corporation, Paoli Pa.

L 1.8. PILOT, The NBS Multicomputer System.
A.L. Leiner et al., Proc. E.J.C.C. 1958, p.p. 71-75.

L 1.9. Priority Line Selecting Circuitry.
J. Muir, IBM Techn. Discl. Bull., Vol. 5, Nr.3, p.p. 22-23.

L 1.10. Priority Circuit.
J.E. Elliott, IBM Techn. Discl. Bull., Vol. 5, Nr. 1, p. 40.

L 1.11. ATLAS, A New Concept in Large Computer Design.
Comm. A.C.M., Vol. 3, p.p. 367-368, June 1960.

L 1.12. Planning a Computer System.
W. Buchholz, McGraw-Hill, New York, 1962, Chs. 3,10,13,14,15.

L 1.13. The Harvest System.
P.S. Herwitz, J.H. Pomerene, Proc. W.J.C.C., 1960, p.p. 23-32.

L 1.14. Performance Advances in a Transistorized Computer System:
The Transac S-2000.
R.J. Segal, et al. Proc. E.J.C.C., 1958, p.p. 168-174.

L 1.15. Parallel Computing with Vertical Data.
W. Shooman, Proc. E.J.C.C., 1960, p.p. 111-115.

L 1.16. The Polymorphic Principle in Data Processing.
H.A. Keit, 1960 IRE Wescon. Com. Rec., Part 4, p.p. 24-28.

L 1.17. The Combi System, A Proposal for New Concept in Digital Data Processing.
H. Schwab, Comm. and Electr. Nr. 49, (Trans AIEE), Part 1, Vol. 7, p.p. 193-197, July 1960.

L 1.18. Communications Within a Polymorphic Intellectronic System.
G.P. West, R.J. Koerner, Proc. W.J.C.C., 1960, p.p. 225-230.

L 1.19. Combined Analog/Digital Computing Elements.
H. Schmid, Proc. E.J.C.C., 1960, p.p. 299-314.

L 1.20. Use of a Digital/Analog Arithmetic Unit Within a Digital Computer.
D. Wortzman, Proc. E.J.C.C., 1960, o.o. 269-282.

L 1.21. A High-Speed Analog-Digital Computer for Simulation.
R.C. Lee, F.B. Cox, IRE Trans. El. Comp., Vol.7, June 1959, p.p. 186-196.

L 1.22. The Shiftrix Machine Organisation for High-Speed Digital Computation.
G. Estrin, Proc. W.J.C.C. , 1958, p.p. 207-211.

L 1.23. Micro Programming.
M.V. Wilkes, Proc. E.J.C.C., 1958, p.p. 18-20.

L 1.24. The Design of a General Purpose Microprogram-Controlled Computer with Elementary Structure.
T.W. Kampe, IRE-Trans., Vol. EC-9, p.p. 208-213, 1960.

L 1.25. The Design of Program-Modifiable Microprogrammed Control Units.
A. Grasselli, IRE-Trans., Vol. EC-11, p.p. 336-339, 1962.

L 1.26. Digital Computer Design Fundamentals.
Y. Chu, McGraw-Hill, New York, 1962, p.p. 461-470.

L 1.27. A Note on Microprogramming.
H.T. Glantz, J. ACM., Vol 3, p.p. 78-84, April, 1956.

L 1.28. Micro-Programming.
R.J. Mercer, J. ACM, Vol. 4, p.p. 157-171, April, 1957.

L 1.29. Micro-Programming and Trickology.
W.L. v.d. Poel.
In: Digitale Informations Wandler. Walter Hoffmann.
Verlag Fr. Vieweg & Sohn, Braunschweig, 1962, s. 269-311.

L 1.30. A Decimal Adder Using a Stored Addition Table.
M.A. Maclean, D. Aspinall, Proc. IEE, Vol, 105 B, p.p. 129-135.

L 1.31. Methods of Speeding-up the Operation of Digital Computers.
I.Y. Akushsky, et al., Proc. Int. Conf. Inf. Proc. UNESCO, Paris, p.p. 382-389.

L 1.32. A New Approach to High-Speed Logic.
W.D. Rowe, Proc. W.J.C.C., 1959, p.p. 277-283.

L 1.33. Design Criteria for Autosynchronous Circuits.
J.C. Sims, Jr. and H.J. Gray, Proc. E.J.C.C., 1958, p.p. 94-99.

L 1.34. Digital Computer Design Fundamentals.
Y. Chu, McGraw-Hill, New York, 1962, chs. 3,5.

c. Codes

L 2.1. IBM Reference Manuel for the 7090-Data Processing System.

L 2.2. Nachrichtentheorie und Codierung.
E.R. Berger. Taschenbuch der Nachrichtenverarbeitung, von K. Steinbuch.
Springer Verlag, Berlin, 1962, s. 58-100.

L 2.3. Digital Computer and Control Engineering.
R.S. Ledley. McGraw-Hill, New York, 1960, p.p. 62-71.

L 2.4. Digital Computer Design Fundamentals.
Y. Chu, McGraw-Hill, New York, 1962, chps, 1 and 2.

L 2.5. Error-Correcting Codes.
W.W. Peterson, Wiley & Sons, New York, 1961, chp. 1.

L 2.6. Negative Base Number Systems.
L.B. Wadel, IRE-Trans., Vol. EC-6, June 1957, p. 123.

L 2.7. Signed-Digit Number Representations for Fast Parallel Arithmetic.
A. Avizienis, IRE-Trans., Vol. EC-10, Sept. 1961, p.p. 389-400.

L 2.8. On a Flexible Implementation of Digital Computer Arithmetic.
A. Avizienis, Proc. IFIP-Congress 62, North Holland Publishing Company, Amsterdam, 1962, p.p. 297-301.

L 2.9. The Residue Number System.
H.L. Garner, IRE-Trans., Vol. EC-8, June 1959, p.p. 140-147.

L 2.10. Digital Computer Design Fundamentals.
Y. Chu, McGraw-Hill, New York, 1962, p.p. 73-78.

L 2.11. Taschenbuch der Nachrichtenverarbeitung.
K. Steinbuch u.a., Springer Verlag, Berlin, 1962.
W. Krägeloh, A/D und D/A Umsetzer, s. 756-779.

L 2.12. A Gray Code Counter.
A.F. Fischmann, IRE-Trans., Vol. EC-6, June 1957, p. 120.

L 2.13. Gray Code Comparing Circuit.
M.P. Marcus, IBM Techn. Discl. Bull.

L 2.14. Arithmetic Operations for Digital Computers Using a Modified Reflected Binary Code.
H.M. Lucal, IRE-Trans., Vol. EC-8, Dec. 1959, p.p. 449-458.

L 2.15. A Ring Model for the Study of Multiplication for Complement Codes.
H.L. Garner, IRE-Trans., Vol. EC-8, March 1959, p.p. 25-30.

L 2.16. Generalized Parity Checking.
H.L. Garner, IRE-Trans., Vol. EC-7, Sept. 1958, p.p. 207-213.

L 2.17. Processing Data in Bits and Pieces.
F.B. Brooks, Jr.,G.A. Blaauw, W. Buchholz,
IRE-Trans., Vol. EC-8, June 1959, p.p. 118-124.

L 2.18. Fingers or Fists.
W. Buchholz, Commun. A.C.M., Vol. 2, Dec. 1959, p.p. 3-11.

d. Carry propagatie

L 3.1. A Comparative Study of Propagation Speed-up Circuits in Binary Arithmetic Units.
M. Lehman, Proc. IFIP Congress 62, (pre-print) North Holland Publishing Company, Amsterdam, 1962.

L 3.2. An Evaluation of Several Two-Summand Binary Adders.
J. Sklansky, IRE-Trans., Vol. EC-9, June 1960, p.p. 213-226.

L 3.3. Arithmetic Operations in Digital Computers.
R.K. Richards, van Nostrand, Princeton, 1955, Chp. 4.

L 3.4. Elimination of Carry Propagation in Digital Computers.
G. Metze, J.E. Robertson, Proc. Int. Conf. on Inf. Process.
UNESCO, Paris, p.p. 389-396.

L 3.5. The Determination of Carry Propagation Length for Binary Addition.
G.W. Reitweisner, IRE-Trans., Vol, EC-9, March 1960, p.p. 35-38.

L 3.6. Fast High-Accuracy Binary Parallel Addition.
H.C. Hendrickson, IRE-Trans., Vol. EC-9, Dec. 1960, p.p. 465-469.

L 3.7. Fast Carry Logic for Digital Computers.
B. Gilchrist, J.H. Pomerene, S.Y. Wong, IRE-Trans., Vol. EC-4,
Dec. 1955, p.p. 133-136.

L 3.8. High-Speed Arithmetic in Binary Computers.
O.L. MacSorley, Proc. IRE, Vol. 49, Nr. 1, Jan. 1961, p.p. 67-91.

L 3.9. A Logic for High-Speed Addition.
A. Weinberger, J.L. Smith, NBS-Circ. 591, 1958, p.p. 3-12.

L 3.10. A One-Microsecond Adder Using One-Megacycle Circuitry.
A. Weinberger, J.L. Smith, IRE-Trans., Vol EC-5, June 1956, p.p.65-73

L 3.11. Arithmetic Operations in Digital Computers.
R.K. Richards, van Nostrand, Princeton, 1955, p.p. 98-111.

L 3.12. Een digitale binaire accumulator.
P.W.J. Verhofstadt, Intern rapport T.H.E., afdeling E, 1962.

L 3.13. Skip Techniques for High-Speed Carry-Propagation in Binary Arithmetic Units.
M. Lehman, N. Burla, IRE-Trans., Vol. EC-10, December 1961, p.p. 691-698.

L 3.14. High-Speed Binary Parallel Adder.
G.L. Reijns, Tijdschrift NRG, Deel 27, Nr. 1, 1962, p.p. 1-15.

L 3.15. Parallel Addition in Digital Computers: A New Fast "Carry" Circuit.
T. Kilburn, D.B.G. Edwards, D. Aspinall, Proceedings I.E.E., 1959, Vol. 106 B, p.p. 464-466.

L 3.16. A Parallel Arithmetic Unit Using a Saturated-Transistor Fast-Carry Circuit.
T. Kilburn, D.B.G. Edwards, D. Aspinall, Proceedings I.E.E., 1960, Vol. 107 B, p.p. 573-584.

L 3.17. High-Speed Transistorized Adder for a Digital Computer.
F. Salter, IRE-Trans., Vol. EC-9, Dec. 1960, p.p. 461-464.

L 3.18. Conditional-Sum Addition Logic.
J. Sklansky, IRE-Trans., Vol. EC-9, June 1960, p.p. 226-231.

L 3.19. Carry-Select Adder.
O.J. Bedrij, IRE-Trans., Vol. EC-11, June 1962, p.p. 340-346.

L 3.20. Delay Line Adder.
D.W. Kean, IBM Techn. Disclos. Bulletin, Vol. 5, Nr. 3, Aug. 1962., p.p. 34-36.

L 3.21. A Fast Parallel Arithmetic Unit.
K.D. Focher, M. Lehman, Proc. I.E.E., Vol. 103 B, Suppl. 3, 1956, p.p. 520-527.

e. Vermenigvuldigen

L 4.1. Arithmetic Operations in Digital Computers.
R.K. Richards, van Nostrand, Princeton, 1955, p.p. 136-165.

L 4.2. A Note on High-Speed Digital Multiplication.
G. Estrene, B. Gilchrist, J.H. Pomerene,
I.R.E.-Trans., Vol. EC-5, September 1956, p.140.

L 4.3. Shortcut Multiplication and Division in Automatic Binary Digital Computers.
M. Lehman, Proc. IEE, vol. 105 B, September 1958, p.p. 496-504.

L 4.4. Methods for High-Speed Addition and Multiplication.
A. Weinberger, J.L. Smith, NBS-Circular 591, 1958.

L 4.5. High-Speed Digital Multiplication.
M. Lehman, IRE-Trans., Vol. EC-6, Sept. 1957, p.p. 204-205.

L 4.6. High-Speed Arithmetic in Binary Computers.
O.L. MacSorley, Proc. IRE, Vol. 49, Nr. 1, Jan. 1961, p.p. 71-80.

L 4.7. Digital Computer and Control Engineering.
R.S. Ledley, McGraw-Hill, New York, 1960, p.p. 528-533.

f. Delen

L 5.1. Arithmetic Operations in Digital Computers.
R.K. Richards, van Nostrand, Princeton, 1955, p.p. 165-176.

L 5.2. A New Class of Digital Division Methods.
J.E. Robertson, IRE-Trans., Vol. EC-7, Sept. 1958, p.p. 218-222.

L 5.3. Reducing Computing Time for Synchronous Binary Division.
R.G. Saltman, IRE-Trans., Vol. EC-10, June 1961, p.p. 169-174.

L 5.4. Digital Computer and Control Engineering.
R.S. Ledley, McGraw-Hill, New York, 1960, p.p. 533-537.

L 5.5. An Algorithm for Rapid Binary Division.
J.B. Wilson, R.S. Ledley, IRE-Trans., Vol. EC-10, December 1961, p.p. 662-670.

L 5.6. "Binary Arithmetic".
G.W. Reitweisner, in "Advances in Computers", Vol. 1, edition F.L. Alt, Academic Press, New York, 1960.

L 5.7. Techniques of Multiplication and Division for Automatic Binary Computers.
K.D. Focher, Quart. Journ. Mech. and Appl. Math., Vol. 11, August 1958, p.p. 364-384.

L 5.8. High-Speed Arithmetic in Binary Computers.
O.L. MacSorley, Proc. IRE, Vol. 49, Nr. 1, Jan, 1961, p.p. 80-91.

L 5.9. Statistical Analysis of Certain Binary Division Algorithms.
C.V. Freiman, Proc. IRE, Vol. 49, Nr. 1, Jan. 1961, p.p. 91-103.

g. Optellen met Vaste Komma

L 7.1. Een Digitale Binaire Accumulator.
P.W.J. Verhofstadt, Intern rapport T.H.E., afdeling E, 1962.

h. Enkele Schakelingen

L 8.1. Transistor Logic Circuits.
R.B. Hurley, Wiley ans Sons, New York, 1961.

L 8.2. Design of Transistorized Circuits for Digital Computers.
A.I. Pressman, Rider, New York, 1959.

L 8.3. Selected Semiconductor Circuits Handbook.
S. Schwartz, Wiley and Sons, New York, 1960.

L 8.4. Handbook of Semiconductor Electronics.
L.P. Hunter, McGraw-Hill, New York, 2 e.d., 1962.

L 8.5. Comparison of Saturated and Nonsaturated Switching Circuit Techniques.
G.H. Goldstick, IRE-Trans. Vol. EC-9, June 1960, p.161.

L 8.6. Transistor Current Switching and Routing Techniques.
D.B. Jarvis, L.P. Morgan, J.A. Weaver,
IRE-Trans. Vol. EC-9, Sept. 1960, p. 302.

L 8.7. Improvements to Current Switching.
F.K. Buelow, IRE-Trans., Vol. EC-9, Dec. 1960, p. 415.

L 8.8. High-Speed Transistorized Adder for a Digital Computer.
F. Salter, IRE-Trans., Vol. EC-9, Dec. 1960, p.p. 461-464.