1 Dataopslag

Gegevens in het werkgeheugen van een computersysteem zijn vluchtig. Daarmee bedoelen we dat deze gegevens verdwijnen als het systeem wordt uitgeschakeld of wanneer een programma in het geheugen met bijbehorende gegevens overschreven wordt door een ander programma. Om gegevens permanent op te slaan hebben we een dataopslagsysteem nodig. In praktijk zijn dit vaak schijven of disks maar voor grote archieven worden ook tapes toegepast.

1.1 RAID

De term RAID staat voor redundant array of independent disks. Hiermee wordt een techniek bedoeld die, gebruikmakend van meer disks, een hogere snelheid en grotere betrouwbaarheid kan opleveren.
Een RAID-systeem maakt vaak gebruik van SCSI-disks (er bestaan ook uitvoeringen op basis van IDE). De controller is zelf ook weer via SCSI op een computersysteem aangesloten. In de meeste gevallen wordt een RAID-systeem toegepast voor fileservers die over een betrouwbare en grote dataopslag moeten beschikken. We zullen een aantal mogelijkheden bespreken.

RAID 0
Door de data parallel op meer disks te schrijven is de datatransfersnelheid te verhogen. De datastroom wordt in stukjes gehakt die ieder op een eigen disk weggeschreven worden. Bij het lezen wordt er parallel van alle disks gelezen en worden de stukjes weer aaneengeregen tot de oorspronkelijke datastroom. In principe kan men zo met twee disks een verdubbeling van de datatransfersnelheid realiseren. De opslag wordt er niet betrouwbaarder door, want het uitvallen van één disk is fataal voor het systeem.

RAID 1
Bij RAID 1 worden gegevens ook parallel weggeschreven, maar nu krijgen beide disks (in het geval van twee disks) dezelfde data. De tweede disk is dus een kopie van de eerste. Dit heet *disk mirroring*. Het voordeel is een betrouwbaar disksubsysteem. Valt er een disk weg, dan kan de andere disk de data nog leveren. We hebben dus hier niet direct snelheidswinst, maar wel een grotere betrouwbaarheid van de dataopslag.

RAID 2
Databits worden parallel over alle disks van de array weggeschreven, waarbij elke disk een andere bit opslaat. Aan deze bits worden volgens de Hamming-code errorcorrectiebits toegevoegd, die ook weer op afzonderlijke disks worden opgeslagen. De verzameling disks werkt dus hier als een grote disk en eventuele problemen kunnen met error-correctie worden opgelost.

RAID 3
Deze methode lijkt veel op de vorige, maar nu worden de data in stukjes gehakt die over meer disks worden weggeschreven, terwijl het deel met de pariteitsbits (voor de error-correctie) op één extra disk wordt opgeslagen. Tot nu toe hebben alle genoemde RAID-technieken gemeen dat de disks synchroon werken. Er wordt voor elke disk op hetzelfde moment op dezelfde plek, gelezen of geschreven.

RAID 4
Bij RAID 4 heeft men deze synchronisatie laten varen en wordt de datastroom in blokken gesplitst die op afzonderlijke disks worden weggeschreven. Een pariteitsdisk zorgt hier weer voor de error-correctie. Een voordeel is dat lezen en schrijven van verschillende blokken parallel kan plaatsvinden. De pariteitsdisk blijkt echter in de praktijk de bottleneck te vormen.

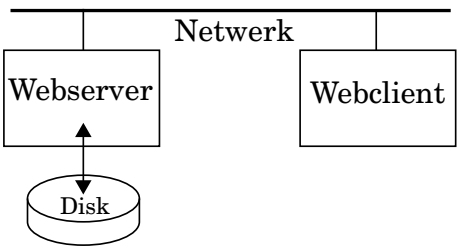
RAID 5
In tegenstelling tot RAID 4 worden de data en pariteitsbits over alle beschikbare disks verdeeld. Dit verlost de pariteitsdisk van RAID 4 van zijn intensieve taak.
De hier beschreven RAID-levels worden door leveranciers van RAID-disksubsystemen nog verder aangevuld en verfijnd om tot een zo efficiënt en betrouwbaar mogelijk systeem van dataopslag te komen. Het is ook mogelijk een computersysteem van meer disks te voorzien om zo met behulp van software een bepaald RAID-level te realiseren. Zo kent de serverversie van windows NT de RAID-levels 1 en 5. In NT-terminologie noemt men dit achtereenvolgens *disk mirroring* en *stripeset with parity*. Ook kunnen diverse Unix-systemen op softwarematige wijze een RAID-systeem implementeren. Wanneer speciale hardware wordt toegepast is de prestatie doorgaans het hoogst. Ook bieden hardware RAID systemen vaak de mogelijkheid om een defecte disk in een werkend systeem te vervangen. We noemen dit 'hot pluggable' RAID-systemen. Niet alle RAID-levels zijn eenvoudig met uitsluitend software op te zetten.

1.2 Dataopslag in computernetwerken

In computernetwerken kunnen computers via de netwerkverbindingen gegevens uitwisselen. Het is dus ook mogelijk om gegevens via het netwerk op een ander systeem op te slaan of de bij een ander systeem opgeslagen gegevens op te vragen. We lopen nu eigenlijk een beetje vooruit, aangezien computernetwerken nog uitvoerig besproken zullen gaan worden, maar gezien het onderwerp van dit hoofdstuk past de behandeling van de volgende onderwerpen het best op deze plaats.

1.2.1 DAS

Onder DAS verstaan we in deze context Direct Attached Storage. Als we spreken over een systeem dat webpagina's voor internet via een netwerkaansluiting beschikbaar stelt dan kunnen de webpagina's opgeslagen zijn op disksystemen die direct met deze webserver verbonden zijn. De gegevens staan zogezegd op de lokale disk. Figuur 1.1 geeft deze situatie weer.

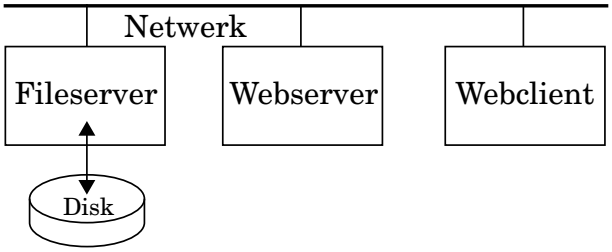


Figuur 1.1 DAS-gebaseerde webserver

Voordeel van deze aanpak is de eenvoud: één systeem voldoet. Deze aanpak is echter niet eenvoudig schaalbaar. Hiermee bedoelen we dat bij groei van de webtoegang er gewoon een grens ligt die niet eenvoudig overschreden kan worden.

1.2.2 NAS

Een tweede mogelijkheid is de volgende situatie. De webserver haalt zijn gegevens (webpagina's) niet van een lokale disk maar van een systeem dat via de netwerkverbinding bestanden beschikbaar stelt. Een dergelijk systeem heet een fileserver.

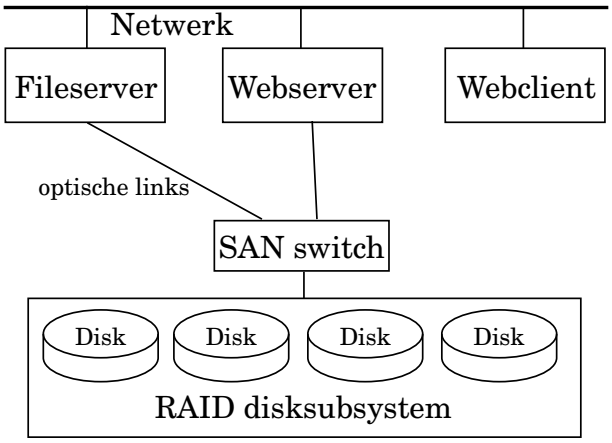


Figuur 1.2 NAS-systeem

Omdat we via het netwerk nu bestanden kunnen opvragen spreken we van Network Attached Storage ofwel NAS. Let wel, als de webserver een webpagina moet aanbieden aan het internet, vraagt de webserver eerst de gegevens van de fileserver. Deze gegevens komen bij de webserver aan die ze weer via het netwerk aan internet aanbiedt. Deze aanpak lijkt wat omslachtig maar heeft toch wel voordelen. Zo kunnen meer webservers gebruikmaken van de webpagina's op de fileserver, waardoor deze aanpak beter schaalbaar is. Natuurlijk hoeft NAS zich niet tot het opslaan van webpagina's te beperken. Allerlei bestanden kunnen op de fileserver hun plaats vinden en vervolgens weer via het net opgevraagd worden. We hebben nu een centrale dataopslag die ook eenvoudig op een back-upsysteem gezet kan worden.

1.2.3 SAN

De afkorting SAN staat voor Storage Area Network. Hierbij is de dataopslag een op zichzelf staand subsysteem geworden. De servers bereiken via een speciaal high-speed netwerk het SAN-systeem. Dit netwerk is vaak op glasvezel gebaseerd. Het SAN-systeem bestaat vrijwel altijd uit een verzameling RAID-systemen voorzien van een snelle netwerkinterface en een mogelijkheid het systeem als geheel te beheren.



Figuur 1.3 SAN-configuratie

1.3 Conclusie

Disksystemen worden fysiek kleiner, de opslagcapaciteit neemt toe evenals de betrouwbaarheid. De snelheid van disksystemen is sterk van invloed op de totale systeemprestatie. Een snelle computer met een trage disk is een onevenwichtig systeem. Naarmate het operating system complexer en krachtiger is, neemt ook de behoefte aan snelle disks toe. In netwerkomgevingen wordt de dataopslag vaak centraal geregeld, hiertoe zijn speciale technieken bedacht om de betrouwbaarheid, snelheid en schaalbaarheid groot te maken.

1.4 Opgaven

1. Wat verstaat men onder RAID? Beschrijf ook een aantal mogelijkheden van deze technologie.
2. Beschrijf het verschil tussen NAS en SAN.

2 Systeemprestatie

In dit hoofdstuk gaan we ons bezighouden met technieken die er enkel op gericht zijn een hogere systeemprestatie of systeem-performance te realiseren. We richten ons hierbij wel op de computer zelf en niet direct op de randapparatuur, waaronder harde schijven. Allereerst zullen we cachegeheugen bekijken. Vervolgens zullen we ons op de CPU-architectuur storten en zien wat op dat gebied mogelijk is om het uiterste uit de hardware te persen. Als voorlaatste onderwerp komen multiprocessorsystemen aan bod. We sluiten af met een overzicht van technieken die men bij de pc heeft toegepast om de systeemprestatie te verbeteren.

2.1 Cachetechnologie

Onder *caching* verstaat men de techniek van het beschikbaar houden van een kopie van eerder gebruikte gegevens op een plaats waar ze een volgende keer sneller te vinden zijn. Het aantal kopieën dat bewaard kan worden, is in de praktijk relatief gering, maar een goed doordacht cachingsysteem kan een aanzienlijke winst in snelheid opleveren. Data op een harde schijf zijn minder snel toegankelijk dan data in het werkgeheugen. Veel besturingssystemen houden daarom van diskblokken kopieën bij in het werkgeheugen. We noemen dit de buffercache of blockcache. Omdat een dergelijk cachesysteem helemaal in software is uit te voeren, spreken we van een softwarecache.
De toegangstijden van betaalbaar geheugen (in de praktijk dynamische RAM), zijn te lang om een snelle CPU op volle toeren te laten draaien. Een CPU verspilt in geval van traag geheugen tijd met zogenoemde 'wait-states'. Dit zijn ingelaste klokcycli om traag geheugen de tijd te geven de data (instructies of gegevens) te laten leveren. Het aanbrengen van een snel cachegeheugen levert winst omdat in de praktijk CPU's in programmalussen ronddraaien. Na één keer kunnen alle instructies uit zo'n lus in het cachegeheugen gekopieerd zijn en daarmee de volgende keer snel toegankelijk zijn voor de processor. Ook gegevens zijn vaak meer keren nodig. We spreken van een cachegeheugen of hardwarecache. Zowel de software- als de hardwarecache heeft de eigenschap dat de computergebruiker eigenlijk alleen de snelheidswinst merkt en verder geen rekening hoeft te houden met de aanwezigheid van een cache, we zeggen dat een cache 'transparant' is voor de eindgebruiker.
Het effect van een cache is uit te drukken in de 'hit rate'. Als een gevraagd item in de cache aanwezig is, spreken we van een cache hit. Is het gevraagde niet in de cache, dan hebben we een cache miss. De hit rate is de verhouding van cache hits ten opzichte van de totale hoeveelheid keren dat data worden opgevraagd (dit is de som van cache hits en cache misses). Vaak wordt de hit rate in een percentage weergegeven. Hit rates van 80% en hoger zijn niet ongebruikelijk bij een goed ontworpen cache.

De sleutelaar
Een persoon die graag aan motoren sleutelt, heeft een garage met een gereedschapsbord aan de muur. Tijdens het sleutelen wordt steeds gereedschap van het bord gehaald en vlakbij op de grond gelegd om snel bij de hand te zijn. Is gereedschap een tweede keer nodig en ligt het vlakbij dan hebben we een cache hit. Als het op de grond te vol raakt, wordt gereedschap dat niet meer nodig lijkt weer weggehangen. Worden er andere activiteiten ondernomen (context switch), dan wordt het

gereedschap op de grond helemaal opgeruimd. De grond in de buurt is de cache, het gereedschapsbord het geheugen. De toegangstijd tot het gereedschapsbord is veel groter dan tot de grond. Een context switch (zie operating systems) veroorzaakt hier een cache flush. Belangrijk verschil: een computer cache bevat een kopie van de bits uit het geheugen. Bij het sleutelen wordt het gereedschap van het gereedschapsbord niet gekopieerd.

Hardwarecache problemen
De volgende punten verdienen bij de realisatie van een cache bijzondere aandacht:

- Het wegschrijven van gegevens: hoe gaat dit in zijn werk, gaan de gegevens naar de cache, het geheugen of beide, en wanneer?
- DMA-activiteiten: een DMA-controller of een tweede CPU kan gegevens in het werkgeheugen, die ook in de cache zitten, veranderen. Volgt de cache die verandering of wordt die verandering in ieder geval opgemerkt?
- I/O-registers: I/O-registers horen niet 'gecached' te worden. Het veranderen van een bit in bijvoorbeeld het statusregister wordt dan wellicht niet door de CPU opgemerkt. Dit probleem is vergelijkbaar met het vorige.

2.1.1 Hardwarecache

Typen
Hardwareontwerpers kunnen kiezen voor de volgende drie mogelijkheden:

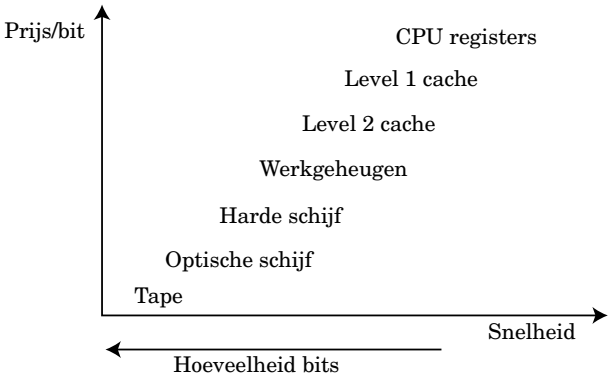
1. instructiecache, de machinecodes worden gecached;
2. datacache, de data worden gecached;
3. zowel instructies als data worden in dezelfde cache bijgehouden.

Er zijn ook CPU's die een gescheiden instructie- en datacache hebben. De CPU maakt dan gebruik van de eerste en tweede mogelijkheid.

Levels
Moderne CPU's beschikken veelal over een op de chip zelf aangebrachte kleine cache. Daarnaast kan ook een externe cache gebruikt worden. De eerste cache noemen we een level 1-cache, de externe cache heet een level 2-cache. Kijken we naar de toegangstijden voor data binnen een computersysteem dan krijgen we het volgende overzicht:

1. interne registers van de CPU;
2. level 1-cache;
3. level 2-cache;
4. werkgeheugen;
5. achtergrondgeheugen (disk);
6. back-up op tape of (schrijfbare) cd-rom.

Een eigenschap van dit rijtje is dat de hoeveelheid data die opgeslagen kunnen worden van boven naar beneden toeneemt, maar de toegangstijd neemt ook toe. Je zou ook kunnen stellen dat de prijs per bit van boven naar beneden gigantisch afneemt. In figuur 2.1 hebben we deze gegevens grafisch uitgezet.



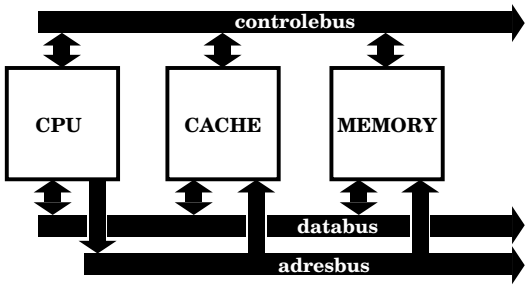
Figuur 2.1 Toegangssnelheid, prijs en hoeveelheid van bits

Cache-werking
Een hardwarecache moet, om goed te functioneren, twee zaken in korte tijd voor elkaar hebben.

1. Zijn de gevraagde data in het cachegeheugen aanwezig?

2. Zo ja, laat het cachegeheugen de data sneller leveren dan het werkgeheugen dit kan. Zo nee, maak een kopie van de door het werkgeheugen geleverde data en registreer ook het hierbij gebruikte adres, zodat de data een tweede keer door het cachegeheugen geleverd kunnen worden.

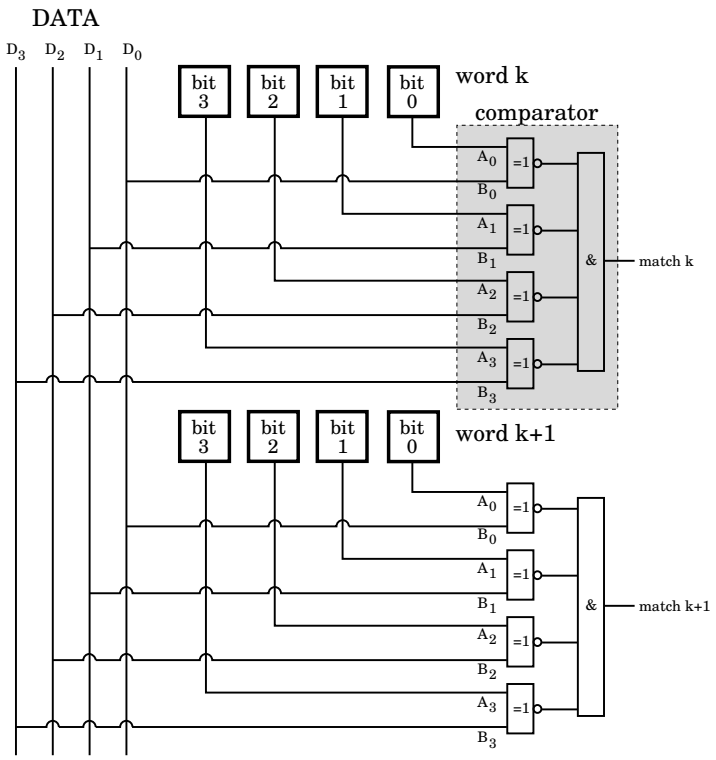
Figuur 2.2 geeft de plaats van het cachegeheugen in een computersysteem aan. De cache staat dus parallel aan het werkgeheugen. Bij een 'cache hit' levert het cachegeheugen data aan de CPU; bij een 'cache miss' worden de data die het werkgeheugen aan de CPU geeft in de cache opgeslagen. We hebben het hier over lees-acties van de CPU.



Figuur 2.2 Plaats van de cache in een systeem

Om te begrijpen hoe aan deze eisen kan worden voldaan, verdiepen we ons eerst in de werking van het zogenoemd associatief geheugen.

Associatief geheugen

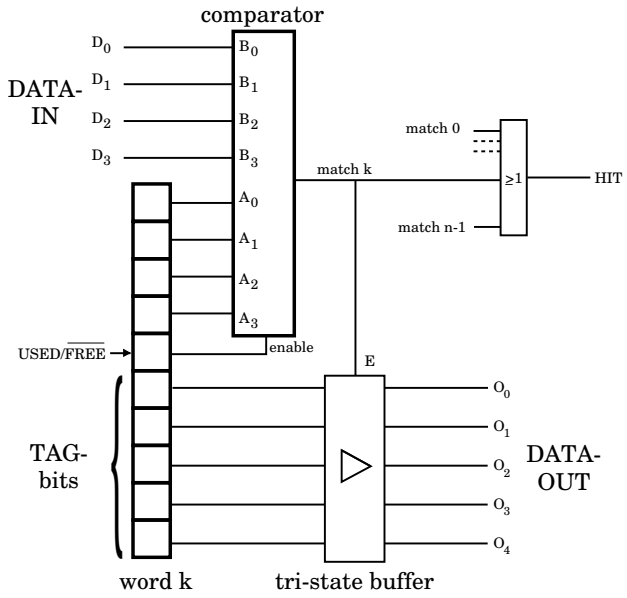


Figuur 2.3 Organisatie van het associatief geheugen

Bij de bespreking van RAM- en ROM-geheugen is het begrip geheugenadres naar voren gebracht. Elk geheugenwoord (van een vast aantal bits) is te adresseren met een binaire code. Deze binaire code geeft in het tweetallig stelsel het geheugenadres weer.

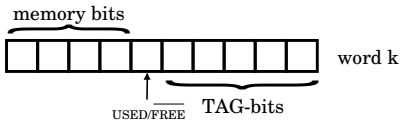
Associatief geheugen werkt anders. Om iets in het geheugen te vinden, wordt het te zoeken bitpatroon of datawoord aan het geheugen aangeboden. Het associatief geheugen geeft op de uitgang aan of het gevonden is en meestal komen er extra gegevens mee die aan het datawoord gekoppeld zijn, of anders gezegd daarmee geassocieerd zijn.

In figuur 2.3 is een mogelijke opbouw van het associatief geheugen weergegeven. Elke geheugenplaats is vier bits breed. Zoals uit de figuur blijkt zijn er heel veel comparators in de vorm van EXNOR-poorten aanwezig. Data in het geheugen worden gezocht met data. Data worden aangeboden en de comparators vergelijken voor elke geheugenplaats of de daar opgeslagen data gelijk zijn. Als de data gelijk zijn, geeft comparator x aan dat de data op positie x zitten. Dit systeem wordt zinvoller als we aan de gezochte data nog extra informatie koppelen in de vorm van TAG-bits.



Figuur 2.4 Eén geheugenlocatie met TAG-bits

Bekijken we figuur 2.4, dan zien we hier een meer uitgewerkt woord in het associatieve geheugen. De DATA-IN wordt met de opgeslagen bits in het geheugenwoord k vergeleken, een extra bit $USED/\overline{FREE}$ is hier nog gebruikt om aan te geven dat het om een wel of niet gebruikte geheugenlocatie gaat. Zijn de bits gelijk en is het een werkelijk gebuikte geheugenlocatie (USED is '1') dan wordt het match k-signaal actief. Als gevolg hiervan wordt het HIT-signaal actief en verschijnen ook de TAG-bits (hier 5 stuks) op de DATA-OUT-uitgangen. Let op: in figuur 2.4 is dus maar één geheugenlocatie van het hele geheugen (n-locaties) weergegeven.
Het HIT-signaal wordt actief als één van de locaties een match geeft met de aangeboden data op DATA-IN. We gaan ervan uit dat er in het hele geheugen hoogstens één match optreedt, anders zouden er conflicten met DATA-OUT ontstaan.



Figuur 2.5 Associatief geheugenwoord

Figuur 2.5 laat nog eens alle bits van een enkele geheugenlocatie zien. De TAG-bits zijn geassocieerd met de memory bits.

2.1.2 Cache-implementatie

Een hardwarecache is te maken met snel associatief geheugen, of met snel conventioneel geheugen (meestal statische RAM), of met een combinatie van deze twee. Hierbij geldt dat hoe meer associatief geheugen wordt toegepast hoe duurder het uiteindelijke ontwerp wordt. We zullen hier drie methoden van cache-implementatie

bespreken:

1. Associatieve mapping. Bij deze techniek wordt alleen associatief geheugen gebruikt.

2. Direct mapping. Een cache-implementatie die conventioneel geheugen gebruikt.

3. Set-associatieve mapping. Deze techniek combineert associatief en conventioneel geheugen.

Associatieve mapping
Associatief geheugen ligt ten grondslag aan deze cachetechniek. In het associatief geheugen worden op de positie van de geheugenbits de adressen opgeslagen. De TAG-bits bevatten de bijbehorende data. In dit cachegeheugen worden paren van geheugenadressen en bijbehorende data dus zó opgeslagen dat met behulp van het geheugenadres dit adres/data-paar weer te vinden is. Figuur 2.6 laat zien hoe een dergelijke cache georganiseerd is.

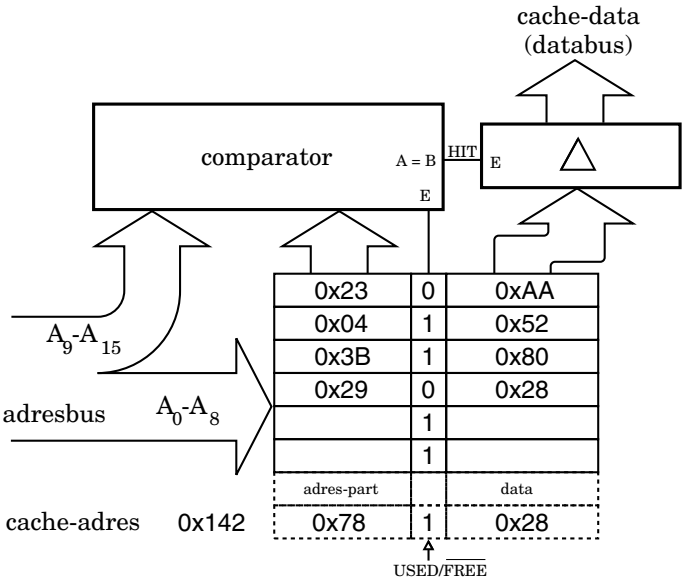
Wordt een adres een tweede keer door de CPU opgevraagd, dan zal het cachegeheugen de met het adres geassocieerde data leveren. Aan het USED/$\overline{\text{FREE}}$-bit is te zien of een geheugenplaats ook werkelijk in gebruik is.

Direct mapping
Een direct mapping cache gebruikt 'normaal' geheugen. Dit geheugen moet echter wel een korte toegangstijd hebben. De truc gaat als volgt en is het gemakkelijkst te volgen met een getallenvoorbeeld.

memory bits (adressen)	USED/$\overline{\text{FREE}}$	TAG-bits (cache-data)
0x7FFF	0	0x00AA
0x034A	1	0xB752
0x03BB	1	0x8000
0x0299	0	0x2708
0x1000	1	0x7151
0x3055	1	0x2222

Figuur 2.6 Associatieve cache

Stel, we willen een cache van 512 bytes, in een systeem dat werkt met adressen van 16-bits en een 8-bits databus (zeg maar de oude Z80). Het cachegeheugen zelf is dan (vanwege die 512 bytes) te adresseren met negen adreslijnen. De adreslijnen A_0 tot en met A_8 zitten direct aan deze adreslijnen van het cachegeheugen. Op elke geheugenplaats van het cachegeheugen wordt nu naast de data ook het resterende deel van het adres bewaard, dus A_9 tot en met A_{15}. Vraagt nu de CPU om de data van een zeker adres, dan komen data en nog een stuk geheugeninformatie met een noodgang uit het cachegeheugen. Dat stuk geheugeninformatie wordt met een comparator vergeleken met de overeenkomstige adreslijnwaarden van de CPU; komen ze overeen dan hebben we een cache hit.

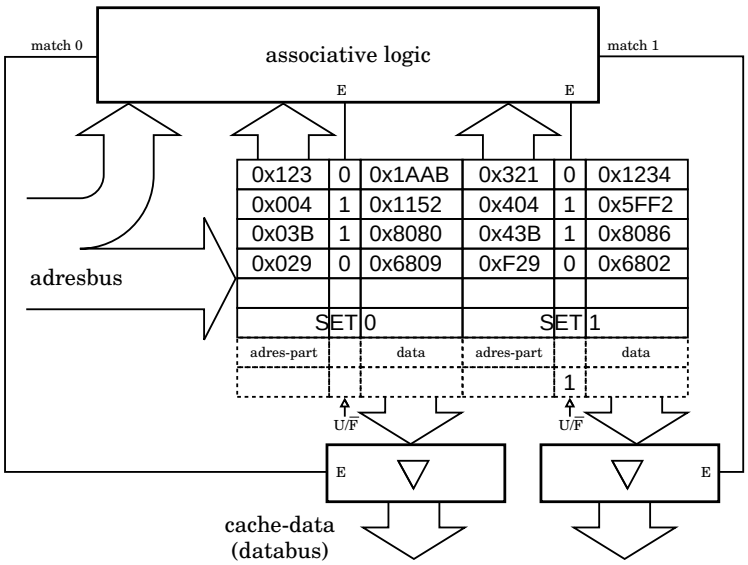


Figuur 2.7 Direct mapping cache

Voorbeeld
In ons systeem wordt de inhoud van adres 0xF142 gevraagd. Dit adres komt overeen met adres 0x142 (onderste negen bits) van het cachegeheugen. Op deze plaats zouden dan bij een cache hit de gevraagde data staan samen met het adresrestant 0x78 (bovenste zeven bits van 0xF142).

De kracht van direct mapping zit hem in de eenvoud van de hardware. De comparator is maar éénmaal nodig (en niet zoals bij associatief geheugen voor elke geheugenplek). De zwakte zit hem in het feit dat een hele verzameling adressen maar naar één cachelocatie gemapped wordt. Zo kunnen 0xF142 en 0x0142 en met hen nog vele andere adressen alleen van cachelocatie 0x142 gebruikmaken. De ene zal dus de andere in de cache overschrijven, terwijl er misschien nog genoeg vrije cacheruimte is.

Set-associatieve mapping
Deze techniek is een combinatie van bovenstaande technieken. Een zuiver associatieve cache zal zeker bij een grote opslagcapaciteit een complexe schakeling zijn vanwege de vele comparators en dergelijke. Een direct mapping cache is vanuit hardware-oogpunt aanzienlijk eenvoudiger, maar kan problemen geven met de caching van data die op dezelfde plek gemapped worden terwijl de cache nog lang niet vol is. Om dat laatste probleem te ondervangen, combineert een set-associatieve cache de direct mapping techniek met een beperkte associatieve schakeling. Per plek kan dan meer dan een datawoord bewaard worden; in de praktijk vaak twee of vier. Een associatief gedeelte zorgt weer voor het uitzoeken van het gezochte datawoord.



Figuur 2.8 Set-associatieve cache

Omdat het associatieve gedeelte maar twee of vier niveaus diep is, is deze hardware nog relatief overzichtelijk. Temeer daar deze associatieve hardware maar één keer uitgevoerd hoeft te worden (net als de comparator bij de direct mapping cache).
In het voorbeeld van figuur 2.8 is een cache met twee entries per set weergegeven. De associatieve logica zoekt uit welke van de twee een match oplevert (match 0 of match 1). De bijbehorende cachedata verschijnen dan via een tristate buffer op de databus.

Schrijven in cachegeheugen
Bij het schrijven van gegevens wordt meestal een van de volgende twee tactieken gevolgd:

1. *Write-through*. Zowel de cachedata als de data in het werkgeheugen worden tegelijk aangepast. Het gevolg is dat deze schrijfactie net zo traag is als het werkgeheugen zelf. Deze oplossing is echter wel eenvoudig.

2. *Write-back*. De data worden voorlopig alleen in het cachegeheugen geschreven, met een extra bit geeft men aan dat deze cache-entry nog naar het werkgeheugen gekopieerd moet worden (de plek in het cachegeheugen heet 'dirty'). Bij een cache flush of voordat deze cachelocatie overschreven wordt door nieuwe gegevens wegens het vollopen van de cache, worden data pas naar het werkgeheugen gekopieerd.

Replacement algoritme
Bij een volle cache moeten oude data plaatsmaken voor nieuwe. De vraag is vaak: welke data moeten eruit? Bij een direct mapping-cache zullen de nieuwe data maar op één plaats weggeschreven kunnen worden en hebben we geen keuze. Bij de associatieve cache en de set-associatieve cache moeten er wel keuzes gemaakt worden. Bij de set-associatieve cache is de keuze weliswaar beperkt, maar toch aanwezig. We zullen ons hier concentreren op de associatieve cache.
Een veel toegepast 'replacement algoritme' is het LRU-algoritme. LRU staat voor least recently used, of met andere woorden: we verwijderen de data in de cache die het minst recent gebruikt zijn. Dit lijkt een redelijke keuze en blijkt ook in

de praktijk goed te voldoen. Om dit in hardware te realiseren, voegen we aan elke cachelocatie een teller toe. Stel, we hebben 256 cachelocaties dan hebben we evenzoveel tellers (of counters) die allemaal een andere waarde hebben. Er is dus een locatie met een counter op 0, een andere counter staat op 255 en alle tussenliggende waarden zijn ook aanwezig. Bij het schrijven in de cache kiezen we steeds locatie 0 en zetten deze counter op 255. De andere counters verlagen we met één (dus de oude 255 wordt 254 en 1 wordt nu 0). Bij het lezen van een cachelocatie, ook wel cache-entry genoemd, zetten we de counter die bij deze entry hoort op 255. We verlagen alle counter-waarden die groter waren dan de oorspronkelijke counter-waarde met één. De locatie met de waarde 255 is dus het meest recent gebruikt en die met counter-waarde 0 is de minst recent gebruikte plek.
Om dit te maken hebben we voor elke locatie een binaire counter nodig. Ook moet elke counter op hetzelfde moment vergeleken kunnen worden met de waarde van de counter die hoort bij de locatie die net gelezen wordt. De counters met een hogere waarde worden dan met één verlaagd. Het zal duidelijk zijn dat dit in principe met voldoende magnitude-comparators wel haalbaar is, maar een behoorlijke hardware-investering vergt. Natuurlijk moet er ook een mechanisme zijn om de counters in eerste instantie (bijvoorbeeld bij het inschakelen van een systeem) allemaal een goede waarde te geven.
Een alternatief voor LRU is het random kiezen van een te verwijderen locatie. Deze aanpak lijkt wat grof, maar is hardwarematig eenvoudiger te implementeren dan LRU.

Cache flushing
Er doen zich in een computersysteem gebeurtenissen voor die aanleiding kunnen geven tot een cache flush. Bij deze actie worden alle data die gecached zijn ongeldig gemaakt. Als de adresvertaling door de MMU gewijzigd wordt bij een context switch kan het noodzakelijk zijn de cache te flushen. In hoofdstuk 13 over operating systems zullen we zien wat een MMU en een context switch zijn. Een flush is nodig als er het gevaar dreigt dat de data in de cache niet meer overeenkomen met de oorspronkelijke inhoud van het werkgeheugen. We zeggen dat de cache niet meer consistent is met het werkgeheugen.

Cache-consistentie
De inhoud van het cachegeheugen moet kloppen met het oorspronkelijke datageheugen. Zolang in een systeem slechts één CPU in het geheugen schrijft, geeft dit geen problemen. Deze komen er wel in een multiprocessorsysteem of een systeem dat DMA gebruikt. We noemen hier een drietal manieren om dit probleem op te lossen:

1. Flush de cache bij DMA. Zoals te verwachten is dit de meest eenvoudige maar ook de meest inefficiënte oplossing.

2. Laat de cache naar de systeembus luisteren (snooping). Komt daar een adres langs waar de inhoud van gecached is, maak dan deze inhoud ongeldig. We noemen dit een ‘snoopy-cache’.

3. Een meer geavanceerde aanpak is om met een snoopy-cache ook de cache-inhoud aan te passen aan de nieuwe data.

2.1.3 Softwarecachetechnieken

Veel besturingssystemen houden van diskblokken kopieën bij in het werkgeheugen. Deze verzameling diskblokken staat bekend als de buffercache. In feite gaat het hier om normaal computergeheugen. Dezelfde filosofie als die van de hardwarecache ligt hieraan ten grondslag: bewaar gegevens tijdelijk op een plaats waar ze een tweede

keer sneller te vinden zijn. Een buffercache is transparant voor de gebruiker, dit in tegenstelling tot een zogenoemde RAM-disk. In het geval van een RAM-disk wordt een compleet filesysteem in het RAM-geheugen opgebouwd. De gebruiker zal ook dit filesysteem als een echt filesysteem herkennen en er rekening mee moeten houden dat de gegevens verloren zullen gaan als de computer uitgezet wordt.

Replacement algoritme
Bij een buffercache is men behoorlijk flexibel in het kiezen van de algoritme die uitzoekt welk blok uit de buffercache mag verdwijnen, omdat het in principe de software is die dit bepaalt. Het eerdergenoemde LRU-vervangingsschema is ook hier een veelgebruikte aanpak. Daarnaast is het mogelijk enkele belangrijke diskblokken – ongeacht hoe vaak ze gebruikt worden – in de buffercache vast te zetten, zodat ze bij het inlezen van een groot bestand in de buffercache blijven bestaan.

Schrijven in de buffercache
Een blok waarvan de gegevens veranderen wordt aangemerkt als 'dirty', zodat bij vervanging van dit blok ook een schrijfactie naar de harde schijf wordt ondernomen. Sommige systemen zullen eens in de zoveel tijd de dirty blokken wegschrijven, andere systemen doen dat meteen, dus deze gebruiken het eerdergenoemde write-through mechanisme. Systemen waarbij de gebruiker op elk moment zonder verdere actie een diskette moet kunnen verwijderen passen dit laatste schema toe. Hetzelfde geldt voor systemen die zonder 'shutdown'-commando uitgezet mogen worden.

Grootte van de buffercache
De grootte kan bij sommige systemen ingesteld worden. Andere systemen maken zelf een keuze, afhankelijk van de hoeveelheid beschikbaar geheugen. Weer andere systemen werken met een vaste grootte.

2.2 RISC versus CISC

2.2.1 Moderne CPU-architectuur

Bekijken we een CPU intern, dan zien we een aantal functionele eenheden zoals de bus interface unit, de ALU, de registerset, enzovoort. In de eerste CPU's waren deze eenheden nauw gerelateerd. Bij latere CPU-architecturen kregen deze onderdelen een meer zelfstandige status. De bus interface unit hield zich bezig met het ophalen en wegschrijven van gegevens (zowel instructies als data). De ALU voert logische en rekenkundige bewerkingen uit. In principe is er niets op tegen deze twee units tegelijk aan het werk te zetten. Bij moderne CPU's treffen we dit ook aan. De functionele eenheden worden zoveel mogelijk benut en eventueel dubbel uitgevoerd om de 'CPU-fabriek' als geheel optimaal te laten functioneren, dat wil zeggen: per seconde zoveel mogelijk instructies uit te laten voeren. Deze aanpak heeft geleid tot de RISC-architectuur en de concepten van RISC zijn ook in de ontwikkeling van andere CPU's terug te vinden.

2.2.2 RISC-filosofie

De achterliggende gedachte bij RISC is: houd het simpel. Er is een tijd geweest dat nieuwere generaties CPU's een steeds uitgebreidere instructieset kregen. De enorme complexiteit die dit met zich mee bracht bleek uiteindelijk remmend te gaan werken. De performance nam niet evenredig toe en compilerbouwers (personen die 'vertalers' voor hoge programmeertalen maken) hadden moeite deze complexe instructieset optimaal toe te passen. CPU's met zo'n complexe instructieset worden

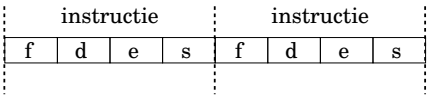
aangeduid met CISC (*complex instruction set CPU*). RISC staat voor *reduced instruction set CPU (of computer)*. Hiermee wordt bedoeld een CPU die een beperkt aantal instructies kent die op zeer hoge snelheid kunnen worden uitgevoerd. Het blijkt dat de prestatie van deze RISC-CPU's die van CPU's met een complexe instuctie set kan overtreffen. De RISC-processors zijn eenvoudiger van opbouw. De instructieset is met zorg gekozen, zodat instructies die zeer vaak voorkomen wel aanwezig zijn. Meer zeldzame instructies uit de CISC instructieset moeten dan met verscheidene eenvoudige instructies nagebootst worden. Omdat deze vaak toch complexe instructies relatief weinig voorkomen, is de RISC als geheel toch sneller. Zuivere RISC-architectuur heeft de volgende kenmerken:

- Alle instructies zijn even groot (in bytes).
- Er is geen uitgebreide keuze aan adresseer-modes.
- Er zijn veel interne registers beschikbaar.
- Er zijn geen instructies die direct bewerkingen op het geheugen uitvoeren. Communicatie met het geheugen gaat alleen met LOAD (geheugeninhoud ophalen en opslaan in een register) en STORE (register wegschrijven naar het extern geheugen). Men noemt dit ook wel *load and store architectuur*.

Het zal duidelijk zijn dat RISC-CPU's niet machinecode-compatible zijn met CISC-CPU's.

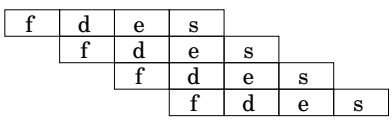
2.2.3 Pipelining

De snelheid van CPU's is een belangrijke factor in de totale performance van een computersysteem (maar zeker niet de enige). Het gebruik van caches zorgt ervoor dat een CPU niet opgehouden wordt door te traag geheugen. De snelheid van de CPU zelf is op te voeren door gebruik te maken van *pipelining*. Functionele eenheden binnen een CPU zijn dan tegelijk bezig met het afhandelen van een gedeelte van de instructie.



Figuur 2.9 Normale instructieafhandeling

In figuur 2.9 is de instuctieafhandeling van een 'normale' CPU schematisch weergegeven. Een instructie neemt vier basis cycles in beslag. De instructiedelen zijn aangegeven met f (fetch = ophalen), d (decode), e (execute) en s (store = resultaat opslaan).



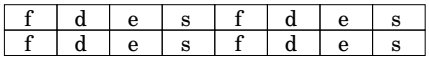
Figuur 2.10 Pipeline

Bij een pipeline-CPU krijgen we figuur 2.10 te zien. Een individuele instructie duurt nog even lang maar per cycle wordt een instructie afgerond. Op één moment worden dus meer (in de figuur: vier) instructies onder handen genomen. Een pipeline

is te vergelijken met een lopende band, de instructies worden door deeleenheden afgewerkt. Dit systeem werkt mooi, maar helaas zijn er wat vervelende problemen. Voorwaardelijke sprongopdrachten vormen een probleem. 'Hoe moet de pipeline gevuld blijven?' 'Wordt de sprong wel of niet genomen?' Speciale oplossingen die bekend staan als 'branch prediction' zijn hiervoor bedacht. Een ander probleem wordt veroorzaakt door het feit dat bepaalde operanden voor instructies nog niet klaar zijn, omdat een instructie wat verderop in de pipeline er nog mee bezig is. Een goed ontworpen compiler kan hier aardig helpen door de instructies in zo'n volgorde te zetten dat het pipelinemechanisme optimaal functioneert.

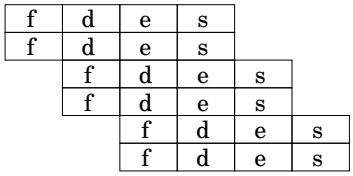
2.2.4 Superscalaire architectuur en superpipelining

Bij een superscalaire architectuur zijn functionele eenheden meervoudig uitgevoerd zodat ze parallel kunnen werken. Figuur 2.11 geeft dit schematisch weer. Evenals pipelining levert dit effectief snelheidswinst.



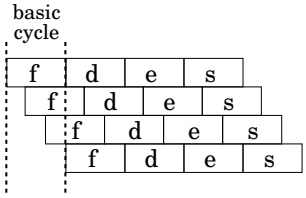
Figuur 2.11 Superscalaire instructieafhandeling

Een superscalaire architectuur is weer te combineren met pipelining met als gevolg een nog hogere prestatie (zie figuur 2.12).



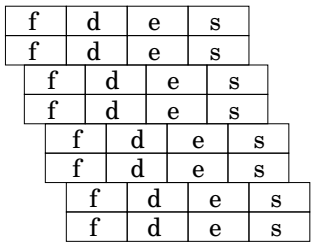
Figuur 2.12 Superscalaire architectuur plus pipelining

Rest ons nog één truc om de snelheid op te voeren: de superpipeline. Als een functionele unit gevoed kan worden met een nieuwe instructie voordat hij helemaal klaar is met de vorige krijgen we figuur 2.13. In één cycle behandelt een functionele eenheid drie instructies (waarvan er één helemaal voltooid wordt). We spreken nu van *superpipelining* van orde 3.



Figuur 2.13 Superpipeline

Natuurlijk is superpipelining weer te combineren met de superscalaire architectuur. Overigens blijven nu nog steeds de problemen van sprongen en operand completion bestaan, ze manifesteren zich zelfs nog sterker.



Figuur 2.14 Superscalaire architectuur met superpipelining

2.2.5 PowerPC, Pentium

De Pentium is een product van Intel die de 8088 serie voortzet. Om het uiterste uit de performance te halen, heeft de Pentium RISC-eigenschappen, maar blijft toch een CISC (al was het alleen al omdat er compatibiliteit moet zijn met de CISC-lijn 8086-286-386-486).
De PowerPC is een Motorola RISC-chip, speciaal bedoeld om als CPU te fungeren in de systemen van IBM en Apple. De volgende tabel geeft een overzicht van verschillen en overeenkomsten.

	PowerPC	**Pentium**
architectuur	RISC	RISC/CISC
maximum instructies per clock-cycle	3	2
cache	32 kB data/instructie	8 kB data 8 kB instructie
databus	64 bits	64 bits
adresbus	32 bits	32 bits
aantal registers	32 algemeen 32 floating point	8 algemeen floating point stack

Tabel 2.1 Vergelijking van de PowerPC met de Pentium

2.2.6 VLIW

VLIW betekent Very Long Instruction Word. Deze ontwikkeling beschrijft een CPU architectuur, waarbij op machinecode niveau zoveel mogelijk parallelisme wordt toegepast. Natuurlijk hebben ze hier ook weer een afkorting voor bedacht namelijk ILP (Instruction-Level Parallelism). Een instructie beslaat een groot aantal bits (bijvoorbeeld 128) en is opgebouwd uit een aantal primitieve RISC-achtige instructies die parallel kunnen worden uitgevoerd. De compiler, die de machinecode genereert, is verantwoordelijk voor het zo groeperen van de primitieve instructies dat dit parallelisme ook werkelijk mogelijk is. Het goed laten functioneren van een VLIW CPU wordt dus op software niveau opgelost.
Het probleem van voorwaardelijke sprongen wordt als volgt aangepakt: beide takken van de sprong worden parallel uitgevoerd tot duidelijk wordt welke tak overeenkomt met de sprongvoorwaarde. De 'verkeerde tak' wordt dan geannuleerd. De branch prediction die we bij RISC zagen wordt bij deze aanpak overbodig.
Voorbeelden van VLIW architecturen zijn de Crusoe van het bedrijf Transmeta en de 64-bits architectuur van Intel, aangeduid met IA64.

2.3 Multiprocessorsystemen

De prestatie van een computersysteem is verder op te voeren door specifieke vaak op de I/O betrekking hebbende taken uit te besteden aan een 'hulpje' in de vorm van een *coprocessor*. Bekend zijn de mathematische coprocessors, die gespecialiseerd zijn in rekenen met floating point getallen (niet-gehele getallen). Maar ook grafische bewerkingen als het verplaatsen van een bitmap in het videogeheugen (bitblock transfer of bitblt) kunnen meestal met succes door een *grafische coprocessor* op een videokaart worden uitgevoerd. Deze hulpprocessors nemen de hoofd-CPU allerlei routineklussen uit handen, waardoor de uitvoering van programma's sneller kan verlopen.
Bij een *multiprocessor* zijn er meer CPU's beschikbaar die kunnen bijdragen aan de uitvoering van een programma. Dit is dus duidelijk anders dan het inzetten van coprocessors die alleen gespecialiseerde deeltaken kunnen uitvoeren.

2.3.1 Classificatie van Flynn

Flynn heeft een classificatie opgesteld die computersystemen indeelt op basis van instructie- en datastromen. Een enkele CPU werkt met één stroom van instructies op één datastroom. Ook andere oplossingen zijn mogelijk, zoals het volgende overzicht laat zien:

- **SISD** (single instruction single data). De architectuur van één instructiestroom, werkend op één datastroom. Dit is de architectuur die we tot nu toe steeds gezien hebben als *de* computerarchitectuur.
- **SIMD** (single instruction multiple data). Deze klasse beschrijft systemen die bekend staan onder de naam vector-computers. Een enkele instructie werkt op een verzameling van datastromen. Bij berekeningen met matrices en vectoren komen de sterke kanten van deze architectuur naar voren. Parallelle rekeneenheden voeren dezelfde operatie uit op ieder een ander stuk van de data. De zogenoemde MMX-instructies van de derde generatie Pentium-processoren vallen in deze klasse.
- **MISD** (multiple instruction single data). Een datastroom gaat door een reeks processors die ieder een eigen instructiestroom hebben. Dit model lijkt een beetje op een pipeline.
- **MIMD** (multiple instruction multiple data). Deze architectuur ligt aan de meeste multiprocessorsystemen ten grondslag. Meer processors werken parallel aan ieder een eigen datastroom.

2.3.2 Architecturen

Parallelle computers van de MIMD-klasse zijn weer in twee hoofdgroepen te verdelen:

1. **Shared memory multiprocessors:** de systemen zijn sterk gekoppeld (vaak bevinden ze zich in dezelfde kast met gemeenschappelijke voeding). De CPU's delen het werkgeheugen en beschikken meestal over een eigen cache. De toegang tot het gemeenschappelijk geheugen is in deze aanpak namelijk een bottleneck aangezien processors vaak via een gemeenschappelijke bus bij dat geheugen moeten komen. De cache geeft in dit geval verlichting.
2. **Message-passing multicomputers:** dit model gaat uit van meer zelfstandige computers die op de een of andere manier gegevens in de vorm van berichten (messages) kunnen uitwisselen. In de praktijk komt dit vaak neer op een netwerk van computersystemen. Het netwerk geeft de mogelijkheid tot onderlinge communicatie.

In beide gevallen wordt het leven van de systeembouwer eenvoudiger als de computers of CPU's identiek zijn. In shared memory multiprocessorsystemen is dit vrijwel altijd het geval. Bij multicomputers is gelijkheid van processors wel handig, maar veel minder noodzakelijk.

Van belang is ook of alle CPU's in een multiprocessor dezelfde toegang tot I/O en geheugen hebben. Is dit het geval dan spreken we van een *symmetric multiprocessor*. Bij zo'n systeem zijn de CPU's gelijkwaardig. Bij het opstarten is er nog wel één CPU die de systeemsoftware (het operating system) van disk laadt, maar als het operating system actief wordt, nemen alle aanwezige CPU's op gelijke wijze deel aan het uitvoeren van de taken. Deze vorm van multiprocessing wordt ondermeer toegepast in Solaris, Windows-NT en Linux. Bij een *asymmetric multiprocessor* kan maar één of een beperkt aantal processors het operating system runnen en de I/O-operaties uitvoeren. De overblijvende processors heten dan attached processors (AP) en staan onder controle van één of meer master processors (MP). Bij SMP systemen ziet men architecturen waarin alle CPU's dezelfde toegangstijd tot het geheugen hebben en architecturen waar er verschil is in toegangstijd tot verschillende delen van het geheugen. In het laatste geval kan een CPU een bepaald stuk geheugen sneller bereiken dan een andere CPU, terwijl die andere wellicht weer sneller bij een ander stuk geheugen kan komen. De eerstgenoemde architectuur heet Uniform Memory Access ofwel UMA. Bij de tweede architectuur spreekt men van NUMA (Non Uniform Memory Access).

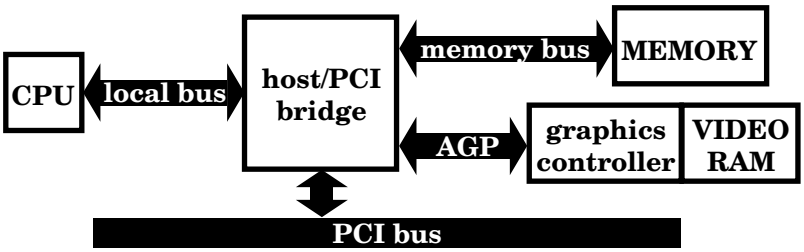
2.4 Pc performance

Na de algemene beschouwingen over systeem performance is het een goed moment om eens te inventariseren wat er zoal in de pc wereld is gebeurd om de prestatie van de systemen op te voeren.

RAM
Het RAM-geheugen is in de loop der tijden steeds sneller geworden. Bij SDRAM (Synchronous Dynamic RAM) heeft men de toegang tot het geheugen gesynchroniseerd met de systeemklok, waardoor de toegangstijd weer wat korter is dan bij Fast Page RAM en EDO RAM. De hoeveelheid cachegeheugen is met de tijd gegroeid met als gevolg een hogere hit-rate. Afhankelijk van het besturingssysteem en de toepassingen is ook de hoeveelheid geheugen een prestatie bepalende factor.

Systeem bus
De kloksnelheid van de systeembus is van een krappe 5MHz bij de eerste pc toegenomen tot 133 MHz. Daarvoor is bij 486 systemen een systeemklok van 33MHz een tijd lang de standaard geweest. De Pentium-systemen hebben relatief lang met 66MHz-systeembussen gefunctioneerd. De uitwendige systeemsnelheid voor processoren wordt ook wel de snelheid van de frontside bus genoemd.

PCI en AGP
Grafische boards worden sneller toegankelijk via de AGP-aansluiting. AGP staat voor Accelerated Graphics Port en is een extra connector naast het PCI-slot, waarmee de CPU buiten de PCI-bus om snel bij het videogeheugen kan komen. Omgekeerd heeft ook de graphics processor op het videoboard snelle toegang tot het systeemgeheugen. In figuur 2.15 is de plaats van de AGP-poort in een systeem aangegeven.



Figuur 2.15 AGP-poort in een pc

De AGP-poort kent de volgende maximale snelheden:

- AGP 1x: 266 MByte/s.
- AGP 2x: 533 MByte/s.
- AGP 4x: 1066 MByte/s.
- AGP 8x: 2,1 GByte/s.

De PCI-bus heeft inmiddels de trage ISA-bus verdrongen. De ISA-bus is alleen maar aanwezig om oude langzame uitbreidingskaarten te ondersteunen. Door AGP wordt de toch wel snelle PCI-bus ontlast van de veel van de bus eisende datatransfers van en naar het videogeheugen.

Harde schijf
De prestaties van de EIDE-schijven zijn toegenomen en bij de toegang heeft men DMA-technieken toegepast. Voor zeer hoge prestatie in serversystemen bestaat er ook de mogelijkheid Ultra-Wide SCSI schijven met een snelle SCSI-controller in te zetten. Met de inzet van hardware RAID-systemen kan men zowel de betrouwbaarheid als de snelheid van een serversysteem opvoeren.

CPU
Op het gebied van de CPU is er een hele ontwikkeling geweest. De eerste pc maakte gebruik van de intel 8088. Vervolgens is het AT-model gebaseerd op de 80286, via de 386 en 486 komen we bij de Pentium-processoren. We zullen een beknopt overzicht geven van de diverse Pentium-processoren:

Pentium	eerste generatie op 60 en 75 MHz
Pentium	tweede generatie met kloksnelheden tot 166 MHz
Pentium MMX	derde generatie met extra instructies gebaseerd op SIMD
Pentium Pro	geschikt voor multiprocessorsystemen (symmetric multiprocessor)
Pentium II	een combinatie van Pentium PRO en MMX. De Pentium II is ook in gestripte vorm uitgebracht (met kleinere onboard cache) voor de goedkopere systemen (Celeron)
Pentium III	uitgebreid met KNI wat Katmai New Instruction betekent (Katmai was de codenaam voor de opvolger van de Pentium II). Dit zijn op floating point gebaseerde SIMD-instructies, waarmee multimediatoepassingen sneller kunnen worden. Intel noemt deze instructies Streaming SIMD Extensions.
Pentium IV	een belangrijk verschil met zijn voorganger is het caching-systeem. Bij de Pentium IV worden instructies gedecodeerd en vertaald naar RISC-achtige micro-operaties, de micro-ops. Dit was bij de voorganger ook zo, maar aangezien het decoderen een complex proces is, hebben de ontwerpers besloten de instructie-cache voorbij de decodeereenheid te zetten. De micro-ops worden nu dus gecached. Dit maakt de Pentium IV voor een groot deel een RISC-architectuur met een CISC-huidje eromheen.

Naast Intel hebben ook AMD en Cyrix min of meer gelijkwaardige CPU's op de markt gebracht. De AMD K6-2 introduceerde de 3DNow instructies, die als voorloper van de Katmai instructies van de Pentium III de trend hebben gezet. 3D staat voor 'driedimensionaal' en richt zich vooral op het ruimtelijk weergeven van bewegende beelden. AMD en Cyrix richten zich met hun prijsstelling meer op de onderste laag van de pc-markt die uit single-CPU-systemen bestaat.

IA64

Vanaf de 386 tot en met de Pentium heeft men een 32-bits architectuur gevolgd. Deze CPU's vallen onder de zogeheten IA32 CPU's (Intel Architecture 32-bits) De volgende CPU-ontwikkeling van Intel is gebaseerd op een 64-bits architectuur ook wel aangeduid met IA64. Deze ontwikkeling is een samenwerkingsverband tussen Intel en Hewlett Packard. De CPU maakt gebruik van een aangepaste vorm van VLIW die door de ontwikkelaars Explicitly Parallel Instruction Computing of EPIC wordt genoemd. EPIC en VLIW hebben als basisprincipe dat het eenvoudig mogelijk is aan te geven welke instructies parallel uitgevoerd kunnen worden. De CPU, die onder de codenaam merced ontwikkeld is, heeft een 128-bits databus. De IA64 haalt instructies als 128-bits eenheden binnen. Zo'n groep bits heet in IA64 terminologie een bundle. De bundle bestaat uit drie instructieslots van elk 41 bits en een 5-bits template. De instructies worden hier syllables genoemd. De templatebits geven aan welke instructies parallel kunnen worden uitgevoerd en beperken zich niet tot een bundle. De processor kan meer bundles bekijken en vervolgens syllables parallel uitvoeren. Met IA64 heeft Intel de directe softwarecompatibiliteit met IA32 laten varen. Ook intern treffen we een andere organisatie aan. De processor beschikt over 128 64-bits registers voor het werken met gehele getalen (integers) en 128 82-bits registers voor floating point bewerkingen. De processor zal bij programmavertakkingen (conditionele sprongen) beide takken volgen totdat duidelijk wordt welke keuze de juiste is. Een complexe techniek als sprongvoorspelling wordt hiermee overbodig.

Zoals al bij de bespreking van VLIW is opgemerkt, zullen de verbeteringen van deze architectuur pas goed tot hun recht komen als de compilers, waarmee de programmatuur naar machinecode wordt omgezet, optimaal van de nieuwe eigenschappen

van deze CPU gebruikmaken.

3D versnelling
Versnelling van de grafische prestaties zijn niet alleen te bereiken met een daarvoor ontworpen instructieset van de CPU. Ook grafische coprocessoren kunnen een belangrijke verbetering geven. De 3Dfx-chipsets kunnen een enorme verbetering geven bij de weergave van 3D beelden. Na de introductie van deze chips is er door fabrikanten van grafische hardware een heel scala aan 3D grafische chips ontwikkeld.

Multiprocessorsystemen
Ook voor pc-systemen zijn moederboards ontworpen waarop meer CPU's tegelijk actief kunnen zijn. Een dergelijk multiprocessorsysteem is alleen zinvol als de software hier ook gebruik van kan maken. Met Windows-NT, Linux en BeOS heeft men naast een aantal Unix-varianten inmiddels al een ruime keuze.

2.5 Conclusie

De technieken die in het eerste deel van dit hoofdstuk aan bod zijn gekomen, helpen om een systeem als geheel beter te laten presteren. Daarbij moet niet uit het oog verloren worden dat het gaat om een geheel. Een snelle CPU zonder cache en met een traag geheugen presteert relatief weinig. Ook de prestatie van de disk (het disk-subsysteem in het algemeen) moet passen bij wat de CPU kan.
Zo is het heel verleidelijk een hoge kloksnelheid voor de CPU te kiezen. Als de overige systeemcomponenten daar in snelheid en capaciteit (denk bijvoorbeeld aan de hoeveelheid geheugen) sterk bij achterblijven, blijkt het systeem niet evenwichtig samengesteld met als gevolg een tegenvallende prestatie. In het algemeen kan men stellen dat de kloksnelheid van de CPU een overschat systeemgegeven is en dat de geheugengrootte een onderschat systeemgegeven is.

2.6 Opgaven

1. Geef een definitie van een cache.
2. Wat verstaat men onder een hardwarecache? Wat verstaat men onder een softwarecache?
3. Wat is associatief geheugen?
4. Welke mogelijke hardwarecache-realisaties zijn er? Geef van elk de voor- en nadelen.
5. Wat is een multiprocessor?
6. Wat is de classificatie van Flynn?

3 Clusters

3.1 Linux

3.1.1 Linux in clusters

Aangezien Linux ook voor clustertechnologie een veel voorkomende keuze is, lijkt dit de aangewezen plaats om eens te kijken wat zo'n cluster eigenlijk is.
Een cluster is een verzameling computers die met behulp van een netwerkverbinding en de juiste software samenwerken om een taak te realiseren. We zullen een algemene beschouwing geven over clustertechnologie en dit vervolgens toelichten aan de hand van drie praktische op Linux-gebaseerde voorbeelden.

Typen clusters
We onderscheiden twee typen clusters:

- Clusters om zware rekenproblemen of veel IO-processing te realiseren. Voorbeelden hiervan zijn beowulf clusters of clusters van werkstations, ook wel aangeduid met de naam CoW.
- Clusters voor hoge beschikbaarheid ofwel *high availability clusters*. Hierbij krijgt een applicatie of service met clustertechnologie een hoge beschikbaarheid, omdat er redundantie is. Hiermee voorkomt men zogenoemde 'single point of failure'. Bij de bouw van een dergelijk cluster moet er goed op gelet worden dat er toch niet ongemerkt een single point of failure kan zijn.

Clustertechnologie is niet systeemgebonden, maar vaak worden Unix of Unix-afgeleiden (FreeBSD of Linux) gebruikt als basis operating system voor het cluster. Hiervoor zijn de volgende redenen te noemen:

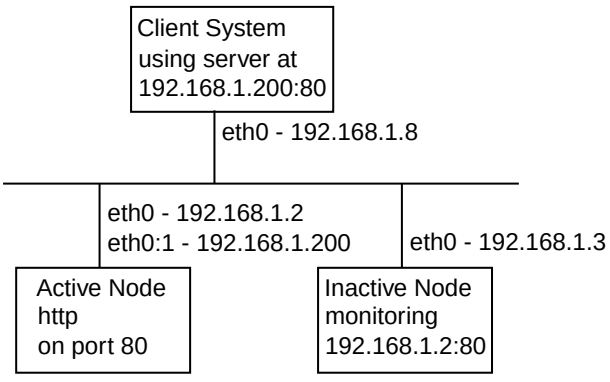
- Stabiliteit van Unix/Linux/FreeBSD.
- Configureerbaarheid van Unix/Linux/FreeBSD.
- Licentiekosten van met name Linux en FreeBSD. Voor commerciële Unix-en zijn vaak relatief voordelige licenties voor clusters beschikbaar.
- Het open source karakter van Linux en FreeBSD. Veel clustertechnologieën zijn nog in ontwikkeling en hierbij is het open sourcemodel in het voordeel.

Het probleem dat men Unix ervaart als een complex operating system speelt in de meeste clusteromgevingen geen rol, omdat daar vrijwel altijd voldoende technische kennis en expertise op het gebied van Unix al aanwezig is.
Bij veel clusterrealisaties kom je regelmatig de volgende twee begrippen tegen: *heartbeat* en *stonith*. Heartbeat is een signaal waarmee de verschillende nodes van een cluster aangeven dat ze nog functioneren. Voor het heartbeatsignaal wordt soms een afzonderlijk netwerk aangelegd. Naast heartbeat kan het in bepaalde situaties wenselijk zijn een niet goed functionerende node uit te zetten of te rebooten. Een systeem wat hiervoor gebruikt kan worden noemt men stonith, deze afkorting betekent 'shoot the other node in the head'.

Beowulf clusters
Beowulf clusters worden toegepast bij rekenproblemen, waarbij het mogelijk is de rekentaak te splitsen in afzonderlijke delen die parallel worden uitgevoerd. Hieronder vallen simulaties van complexe modellen op het gebied van meterologie, natuurkunde, sterrenkunde, scheikunde en biologie. Een andere rekenintensieve klus is rendering van filmbeelden. Hiervoor worden clusters belast met het doorrekenen van filmanimaties. Een dergelijke cluster noemt men ook wel renderfarm. Als laatste voorbeeld noemen we clusters die ingezet worden voor cryptografische toepassingen. Het cluster wordt meestal opgebouwd op basis van standaard-pc-hardware. In principe hebben de systemen geen eigen monitor of toetsenbord, maar communicaren alleen via een netwerk met de buitenwereld. Vaak treffen we een masternode aan van waaruit het cluster te beheren is en waar applicaties opgestart kunnen worden. Het upgraden van memory of het veranderen van hardware-instellingen is bij een cluster van bijvoorbeeld 256 pc's, die elk een eigen standaardkast hebben een behoorlijk tijdrovende klus. Daarom worden steeds vaker rack-mountable units gebruikt. Hierin zijn de componenten veel eenvoudiger te bereiken. Ook de koeling van de systemen is een punt van aandacht, om nog maar te zwijgen van de power consumptie.
Om clusters voor de eerdergenoemde toepassingen succesvol in te zetten is het van belang dat de software te paralleliseren is. Hiervoor geldt de regel dat een groot aantal onafhankelijke berekeningen in de software moet zitten. Het gevolg is dan dat de nodes van een cluster tegelijk deze berekeningen uitvoeren en de communicatie overhead relatief gering is. De meest toegepaste parallele programmeersystemen zijn MPI en PVM.

High availability clusters

Fail-over server (fos)
Een fail-over server is een cluster bestaande uit twee nodes. Een node is de actieve node en levert service (http, ftp etc.) terwijl de andere node, de inactieve node, de service van de actieve node in de gaten houdt en klaarstaat om in te vallen als de actieve node uitvalt (standby node). Bij de Linux implementatie van fos werkt een service meestal op een Virtual IP address. Dit is een tweede IP-nummer dat aan de netwerkinterface is toegekend. Dit nummer wordt door de backup node overgenomen op het moment dat de actieve node uitvalt. De service blijft zodoende op hetzelfde IP-nummer beschikbaar.

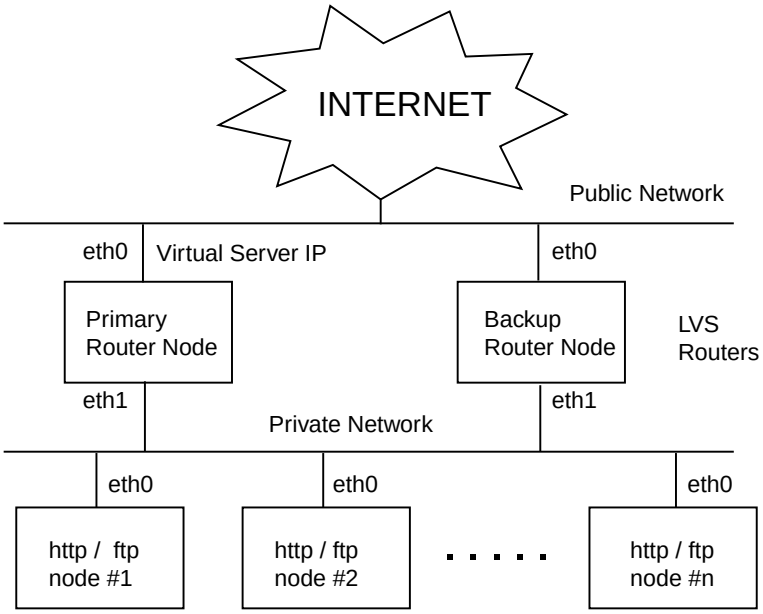


Figuur 3.1 Componenten van een fos

De inactieve node test de service. We noemen dit service monitoring. Dit komt meestal neer op het testen van de poort waarop de service beschikbaar is. Het is

mogelijk een teststring te sturen en een reply af te wachten. Daarnaast zenden beide nodes een periodiek signaal het netwerk op die de naam heartbeat message draagt. Elke node verwacht eens in de zoveel tijd een heartbeat van de andere node. Als de heartbeat uitblijft, wordt dat gezien als een falende node en treedt de failover in werking.

Linux virtual Server (lvs)
We bekijken eerst de opbouw van een lvs. Een Linux virtual server bestaat uit een verzameling servers, die via een fail-over systeem van routers verbonden is met het netwerk waarvoor ze service verlenen. De buitenwereld gebruikt het adres van de actieve router voor toegang tot de service. De router kiest een van de servers uit de serverfarm waar het echte werk op gedaan wordt.
Het adres waar de router de service op aanbiedt is weer een Virtual IP die door de backup router wordt overgenomen ingeval van problemen met de primary router. Het tweede netwerk kan een non routable netwerknummer hebben (192.168.x.x 10.x.x.x o.i.d.), maar ook 'normale' IP-nummers zijn mogelijk.



Figuur 3.2 Opbouw van een lvs

Om te voorkomen dat een node al het werk op zijn bordje krijgt hebben we een systeem nodig waarbij het werk goed verdeeld wordt. Dit noemen we load balancing. Bij lvs zijn vier load balancing methoden mogelijk:

1. Round robin: de jobs worden over de beschikbare servers verdeeld waarbij elke keer een volgende server aan de beurt komt.

2. Least-connections: De nieuwe jobs gaan naar servers met de minste actieve connecties.

3. Gewogen round robin: deze methode volgt de round robin verdeling, maar servers met een grotere capaciteit komen vaker aan de beurt. De beheerder van het cluster kent zelf de capaciteitsgewichten toe en deze worden bijgesteld met de dynamic load informatie.

4. Gewogen least-connections. Hierbij wordt naast het aantal actieve connecties

ook nog rekening gehouden met de capaciteit. Ook hier is de capaciteit een gegeven dat wordt bijgesteld met de dynamic load informatie.

Clusters op de langere termijn
Ondanks het feit dat CPU's en computersystemen op zichzelf steeds krachtiger worden, zullen clusters blijven bestaan, omdat ze per definitie krachtiger zijn dan een enkele computer van dezelfde architectuur. Voor zware rekenproblemen blijkt een cluster een economische oplossing te zijn. Voor high availability lijkt een cluster een relatief goedkope en voor de hand liggende oplossing.

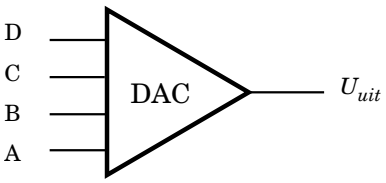
4 AD- en DA-conversie

Als we een analoog signaal digitaal willen opslaan en eventueel verder verwerken met digitale technieken, dan moet het analoge signaal in een digitale code worden omgezet. Men spreekt dan van *analoog-digitaalconversie* ofwel kortweg *AD-conversie*. Omgekeerd noemen we het proces, waarbij een digitale code in een analoog signaal wordt omgezet, *digitaal-analoogconversie* of *DA-conversie*. Door de komst van de microcomputer is de digitale registratie en verwerking van signalen sterk in belang toegenomen. Aangezien gegevens van een opnemer meestal analoge signalen zijn, is de AD-conversie hiervoor onontbeerlijk. Het is nu mogelijk om rond een (micro)computer een regelsysteem op te zetten waarbij men gebruikmaakt van AD- en DA-conversie. Een voordeel van deze aanpak is onder meer dat de regelactie geheel in software uitgevoerd is en daardoor eenvoudig aan nieuwe of complexe situaties aan te passen is. Een tweede voordeel is dat meetgegevens digitaal beschikbaar zijn en met digitale opslagmethoden bewaard kunnen worden.

De elektronische bouwsteen die de digitaal-analoog conversie verricht, noemen we een digitaal-analoogconverter ofwel kortweg *DAC*. De tegenhanger is de analoog-digitaalconverter afgekort met *ADC*. We zullen nu de eigenschappen van de ADC en DAC en de mogelijke realisatie bekijken. Omdat bij AD-conversie vaak een DAC als tegenkoppelelement wordt gebruikt, zullen we met DA-conversie beginnen.

4.1 DA-conversie

De DA-conversie is een proces waarbij een waarde, aangegeven in digitale code (bijvoorbeeld binair of BCD), wordt omgezet in een spanning of stroom die daarmee evenredig is. Een DA-convertor of DAC is een schakeling met een aantal ingangen waarop een digitale code wordt aangeboden en één uitgang waarop een analoge spanning of stroom beschikbaar is. Figuur 4.1 stelt een 4-bit DAC voor met een spanningsuitgang.



Figuur 4.1 4-bit DAC

Een 4-bit DAC met een evenredigheidsfactor van 1 geeft voor de binaire code 0001, 1 volt op de uitgang. Tabel 4.1 geeft de uitgangsspanning voor alle mogelijke ingangscodes (natuurlijk zijn ook andere evenredigheidsfactoren mogelijk).

DAC-tabel

INPUT				U_{uit}
D	C	B	A	[V]
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

Tabel 4.1 Uitgangsspanning van een DAC voor alle mogelijke ingangscodes

De hoogste uitgangsspanning noemt men de *full scale output* (fs-value). Elke ingang van de DAC heeft een andere bijdrage tot de analoge uitgangsspanning. Bekijken we tabel 4.2, dan zien we dat de bijdrage van elke digitale ingang gewogen wordt volgens de positie in het binaire getal. A heeft het gewicht 1, B 2, C 4 en D 8.

INPUT				U_{uit}
D	C	B	A	[V]
0	0	0	1	1
0	0	1	0	2
0	1	0	0	4
1	0	0	0	8

Tabel 4.2 Uitgangsspanning van een DAC per bit

A noemt men de minst significante bit (LSB = Least Significant Bit) en D noemt men de meest significante bit (MSB = Most Significant Bit).

4.1.1 Enkele DAC-parameters

Resolutie
Over de parameter 'resolutie' zijn verscheidene definities in omloop. De resolutie van een DAC is gedefinieerd als de kleinste verandering van de analoge output als gevolg van een verandering in de digitale input. De resolutie is altijd gelijk aan het gewicht van het LSB. De resolutie van de DAC van ons voorbeeld is dus 1 volt. Deze definitie van resolutie wordt ook wel *stapgrootte* genoemd. Vaak geeft men de resolutie aan als een percentage van de maximale uitgangsspanning. Dus in ons voorbeeld.

resolutie = stapgrootte/fs-value = $1/15 \times 100\% = 6.67\%$

Voorbeeld: We beschouwen een 10-bit DAC met stapgrootte van 20 mV. Hiervoor geldt:

fs-value = $(2^{10} - 1) \times 20$ mV $= 1023 \times 20$ mV $= 20.46$ V

resolutie = $((20 \times 10^{-3})/20.46) \times 100\% = 0.1\%$

Merk op dat de stapgroote er eigenlijk niet toe doet. We kunnen ook zeggen:

resolutie = $(\text{aantal stappen})^{-1} \times 100\%$

Het aantal bits bepaalt de resolutie.

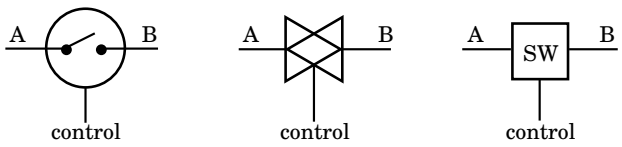
Insteltijd
De *insteltijd* (*settling time)* is de tijd die de DAC nodig heeft om een, bij een bepaalde input-code behorende, uitgangswaarde in te stellen. Deze tijd is meestal maximaal wanneer alle input-bits van 0 naar 1 gaan.

Uitgangsspanning
In ons voorbeeld produceert de DAC alleen positieve uitgangsspanningen, maar in de praktijk zijn ook DAC's mogelijk die met 2-complement-code aangestuurd kunnen worden en, al naar gelang de ingangscode, een positieve of negatieve uitgangsspanning produceren.

4.1.2 Principe-realisatie van DAC's

We zullen hier niet al te diep op ingaan, omdat DAC's veelal als een complete elektronische component geleverd worden. Maar om enig inzicht te krijgen in de werking van een DAC zullen we hier twee mogelijke realisaties bespreken. Voor het begrip van deze schakelingen is kennis van de werking van een operationele versterker, ook wel aangeduid als opamp, nodig. In het volgende stukje gaan we van deze kennis uit en deze stof is alleen bedoeld voor die lezers die het naadje van de kous willen weten.

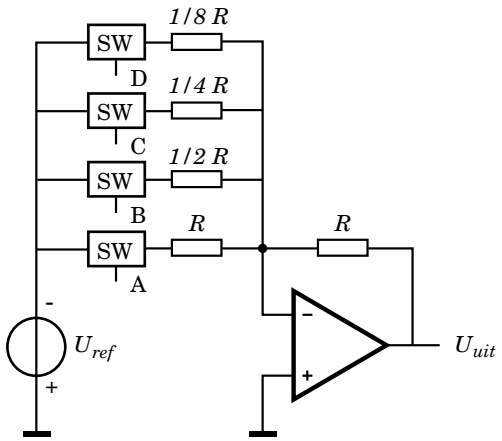
We zullen daartoe eerst het begrip analoge schakelaar of *analog switch* moeten toelichten. Een analog switch is een elektronische schakelaar (dat wil zeggen een transistor of FET treedt op als schakelelement), die geopend of gesloten kan worden door op de stuuringang (Control-Input) een logische '0' respectievelijk '1' aan te bieden. Bij een '0' is de schakelaar dus open, bij een '1' dicht (dat wil zeggen geleidend). Voor deze schakelaar zijn nogal wat namen en symbolen in de omloop, zoals *transmissiepoort*, *bilaterale schakelaar* en de overeenkomstige Engelse namen. Figuur 4.2 geeft een aantal gebruikte symbolen.



Figuur 4.2 Symbolen analog switch

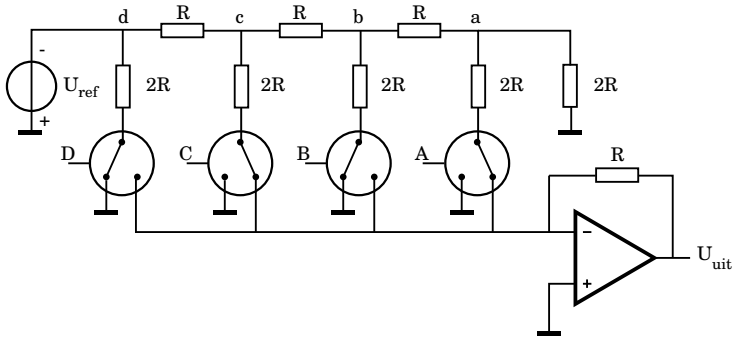
Ga zelf na of het mogelijk is om met twee switches en een invertor (digitale invertor) een ompool-schakelaar te maken.

Deze switches kunnen een analoge spanning die weliswaar niet buiten het bereik van de voedingsspanning van de switches mag vallen, in- of uitschakelen. Met behulp van deze switches is een 4-bit DAC als in figuur 4.3 te realiseren.



Figuur 4.3 Principe-realisatie van een DAC met een opamp

Bij het fabriceren van een geïntegreerde schakeling is het erg lastig om een set nauwkeurige weerstanden over een groot bereik te maken. Wel is het goed mogelijk om identieke weerstanden te maken. Daarom past men in een IC vaak een ander principe toe, namelijk een opamp met een *laddernetwerk*, zie figuur 4.4.



Figuur 4.4 Realisatie van een DAC met een opamp en een laddernetwerk

Deze schakeling is ook nu weer een som-versterker. Door het laddernetwerk staat op de punten d, c, b en a achtereenvolgens U_{ref}, $U_{ref}/2$, $U_{ref}/4$ en $U_{ref}/8$. Dit is onafhankelijk van de standen van de switches, omdat deze tussen aarde of virtuele aarde schakelen. De min-ingang van de opamp zal door de tegenkoppeling van de uitgang en de zeer hoge versterkingsfactor van de opamp vrijwel op dezelfde potentiaal zijn als de plus-ingang, die aan aarde ligt. Eigenlijk hebben we hier net als in figuur 4.3 ook weer een optelschakeling. Afhankelijk van de standen van de switches worden de spanningen op de punten d, c, b en a wel of niet bij de uitgangsspanning opgeteld. Let ook op de aansluiting van de referentiespanning U_{ref}. Aangezien we in beide voorbeelden een inverterende schakeling hebben, gebruiken we voor positieve uitgangsspanningen een negatieve referentiespanning.

4.2 Analoog-digitaalconversie

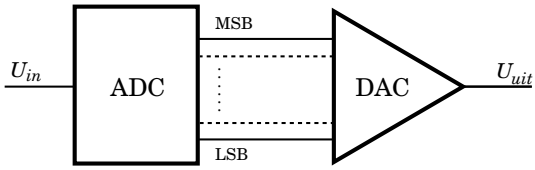
Een ADC produceert een digitale code die evenredig is met de analoge input. De digitale code kan binair, BCD of van een andere code zijn. ADC's die binaire code produceren, worden veelal toegepast in meet- en regelsystemen met digitale

signaalverwerking; de ADC met BCD-code-uitgangen vindt men meestal in digitale voltmeters en daarvan afgeleide meetapparatuur.

4.2.1 ADC-parameters

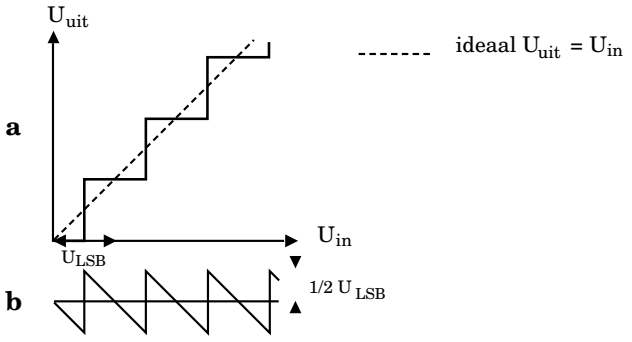
Resolutie
De resolutie van een ADC wordt meestal opgegeven als het aantal bits van de uitgangscode.

Kwantisatiefout
Bekijk de schakeling van figuur 4.5: een N-bit ADC gevolgd door een ideaal verondersteld N-bit DAC.



Figuur 4.5 Ideale DAC in serie met een ADC

Wanneer we voor deze schakeling U_{uit} tegen U_{in} uitzetten, dan vinden we voor een ideale ADC de grafiek van figuur 4.6. We zien in deze figuur een afwijking van de rechte lijn door de oorsprong met helling 1. Het verschil tussen de maxima van deze afwijking noemen we de kwantisatiefout; voor een ideale ADC is deze fout U_{LSB}. De kwantisatiefout ontstaat doordat de ADC-uitgang slechts discrete waarden kan aannemen en er dus een afronding moet plaatsvinden. De fout wordt kleiner naarmate het aantal bits voor dezelfde spanning toeneemt. Voor een niet-ideale ADC kunnen nog andere afwijkingen optreden. De trapjes uit de figuur kunnen bijvoorbeeld niet elke keer even hoog zijn of zelfs tijdelijk even naar beneden gaan in plaats van omhoog.



Figuur 4.6 **a)** $U_{uit}(U_{in})$ voor ADC-DAC-combinatie
b) Afwijking van de ideale lijn

Conversietijd
De tijd die nodig is om de analoge ingangsspanning om te zetten in digitale informatie. Deze tijd is sterk afhankelijk van het toegepaste conversieprincipe. De AD-conversie duurt meestal langer dan de DA-conversie. Men moet ervoor waken dat de ingangsspanning van de ADC niet sterk kan veranderen tijdens het conversieproces; indien dit wel het geval is kan een *sample-and-hold*schakeling uitkomst bieden. Deze schakeling bemonstert het te converteren signaal en bewaart

de momentane waarde totdat de ADC de conversie uitgevoerd heeft. Voor digitale voltmeters is het meestal voldoende om 2 à 4 conversies per seconde te doen. Voor spraakverwerking en videotoepassingen moet de conversie sneller gebeuren. Wordt bij de conversie een DAC toegepast, dan is de settling time van de DAC van invloed op de conversietijd.
We zullen nu een aantal methoden van AD-conversie bespreken, met hun voor- en nadelen. We wijzen erop dat ook ADC's meestal als compleet IC geleverd worden, zodat het zelf bouwen meestal niet de moeite loont.

4.2.2 AD-conversiemethoden

In al deze methoden komen we de opamp weer tegen. Nu wordt hij gebruikt als analoge comparator. Meestal gaat het hier om een comparator waarvan de uitgang een logisch niveau ('0' of '1') geeft. De uitgangsinformatie is dan direct door logische schakelingen te verwerken.

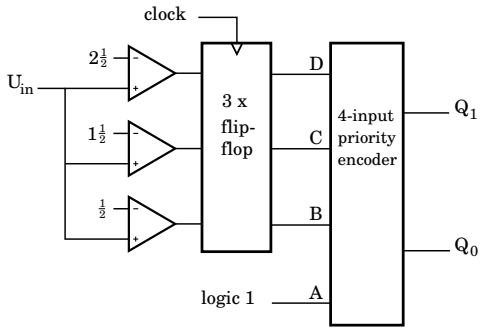
Parallelmethode
Met behulp van een aantal comparators vergelijkt men de ingangsspanning met een aantal referentiespanningen. Deze referentiespanningen zijn meestal met een spanningsdeler afgeleid uit één referentiespanning. Het resultaat wordt in een register vastgelegd; de uitgangen van deze flipflops worden toegevoerd aan een *priority encoder*. Zo'n priority encoder heeft de volgende waarheidstabel.

Waarheidstabel priority encoder

INPUT								U_{uit}		
A	B	C	D	E	G	H	I	Q_2	Q_1	Q_0
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0
x	1	0	0	0	0	0	0	0	0	1
x	x	1	0	0	0	0	0	0	1	0
x	x	x	1	0	0	0	0	0	1	1
x	x	x	x	1	0	0	0	1	0	0
x	x	x	x	x	1	0	0	1	0	1
x	x	x	x	x	x	1	0	1	1	0
x	x	x	x	x	x	x	1	1	1	1

Tabel 4.3 3-bits priority encoder

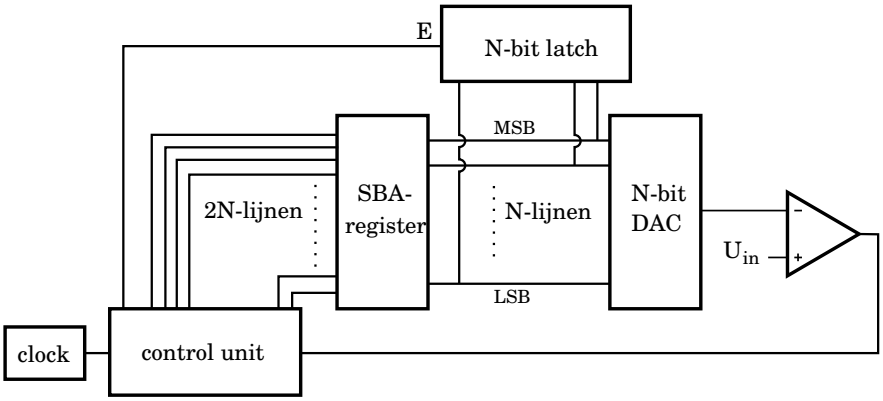
De priority encoder levert dus op de uitgangen het binaire nummer van de 'hoogste' ingang die '1' is; ingang A is nummer 0, ingang B is nummer 1, C is nummer 2 en I is nummer 7. Een priority encoder is eenvoudig uit elementaire poortschakelingen op te bouwen. Voor de parallelmethode vinden we voor twee bits de schakeling van figuur 4.7.



Figuur 4.7 2-bit ADC volgens de parallelmethode

Een *voordeel* van deze schakeling is dat hij zeer snel is. Elke klokpuls levert een nieuwe conversie.
Het *nadeel* is dat voor een toenemend aantal bits het aantal comparators, referentiespanningen en flipflops enorm toeneemt. Voor een N-bit ADC zijn immers $(2^N - 1)$ comparators, referentiespanningen en flipflops nodig.
Met de opkomst van integratietechnieken met hoge componentdichtheid is deze methode meer en meer in gebruik gekomen. Vooral voor digitale beeldverwerking is deze ADC zeer bruikbaar. Deze ADC wordt ook wel *flash converter* genoemd.

Weegmethode
Figuur 4.8 stelt een blokschema voor van een ADC waarin het principe van *successive bit approximation* (SBA) wordt toegepast. Vertalen we deze kreet, dan wordt dat 'opeenvolgende bitbenadering'. Dit is nou niet direct verhelderend. Wel is duidelijk dat elke keer één bit bepaald wordt. Laten we maar eens kijken wat er gebeurt.



Figuur 4.8 Blokschema ADC volgens SBA-methode

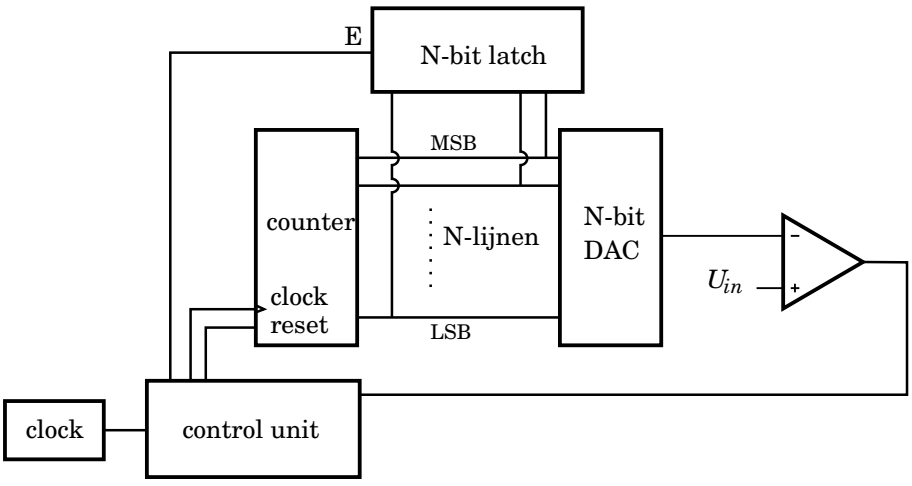
Het SBA-register bevat N flipflops, waarvan de uitgangen ieder verbonden zijn met een N-bit DAC. Veronderstel dat in eerste instantie alle flipflops '0' zijn. De controller set eerst één flipflop (maakt de uitgang '1') van het SBA-register. Het is de flipflop waarvan de uitgang met de MSB-ingang van de DAC verbonden is. De andere flipflops blijven '0'. De DAC levert nu de bij de ingangscode behorende analoge waarde. De comparator bekijkt of deze DAC-uitgangsspanning hoger of lager is dan U_{in}; afhankelijk hiervan zorgt de controller ervoor dat de flipflop, die met de MSB-input van de DAC is verbonden, geset blijft of weer '0' gemaakt wordt. Als namelijk de DAC-spanning hoger is dan U_{in} zal de flipflop weer '0' worden, was de spanning lager dan blijft de flipflop '1'. Hierna zal de controller de flipflop, die met het op één na meest significante bit van de DAC verbonden is, '1' maken en opnieuw kijkt de comparator of de DAC-spanning hoger of lager is dan U_{in}. Indien de spanning van de DAC hoger is dan U_{in}, zal deze flipflop gereset worden; was de DAC-spanning lager dan U_{in} dan zal de flipflop geset blijven. Deze procedure herhaalt zich tot en met de flipflop die met het LSB van de DAC verbonden is. In het SBA-register staat nu het binaire equivalent van U_{in}. Dit resultaat wordt in de N-bit data latch opgeslagen. De flipflops van het SBA-register worden gereset en de hele conversie kan opnieuw beginnen.
In principe speelt zich hier de elektronische vorm van een raadspelletje af. Er moet een getal geraden worden in een bepaald gebied. De snelste manier om het getal te raden, is om eerst de helft van het gebied te nemen en te vragen of het te raden

getal hoger of lager is. Vervolgens nemen we de helft van de helft en bakenen steeds meer het gebied af waarin het getal zit. We zullen met een voorbeeld de gang van zaken toelichten. We beschouwen een 4-bit ADC met de DAC uit het eerste voorbeeld (fs-value 15 volt, stapgrootte 1 volt, 4-bits). Het SBA-register bestaat uit vier flipflops. Stel, U_{in} is 10,2 volt.

- Begin: alle flipflops zijn '0'.
- Stap 1: code 1000 in het SBA-register; DAC levert 8 volt; comparator geeft '1', omdat 8 volt lager is dan 10,2 volt; resultaat: 1000 in het SBA-register.
- Stap 2: code 1100 in het SBA-register; DAC levert 12 volt; comparator geeft '0'; resultaat 1000 in het SBA-register.
- Stap 3: code 1010 in het SBA-register; DAC levert 10 volt; comparator geeft '1'; resultaat 1010 in het SBA-register.
- Stap 4: code 1011 in het SBA-register; DAC levert 11 volt; comparator geeft '0'; resultaat 1010 in het SBA-register.
- Stap 5: de inhoud van het SBA-register wordt in de latch geklokt; daarna worden de flipflops gereset en gaan we weer terug naar stap 1.

De latch bevat na conversie binair %1010, dit is 10 in het decimale stelsel en klopt dus met de ingangsspanning. Nauwkeuriger dan 1 volt kunnen we namelijk niet werken met deze voorbeeld-ADC.
Voordeel: deze methode is redelijk snel, per stap winnen we één bit. Dus 12 bits in 12 stappen. Voor een toenemend aantal bits wordt de schakeling niet gigantisch veel groter.
Nadeel: de controller is een redelijk complexe schakeling. De DAC moet nauwkeurig zijn en een korte settling time hebben om de conversie snel te kunnen doen.
Deze conversiemethode wordt toegepast wanneer een snelle conversie met hoge nauwkeurigheid nodig is.

Telmethode met DAC
Deze methode laat een binaire counter met een daaraan gekoppelde DAC tellen tot de gewenste code is bereikt. Het bereiken van de code wordt aangegeven door een comparator (zie figuur 4.9).



Figuur 4.9 Blokschema ADC met counter en DAC

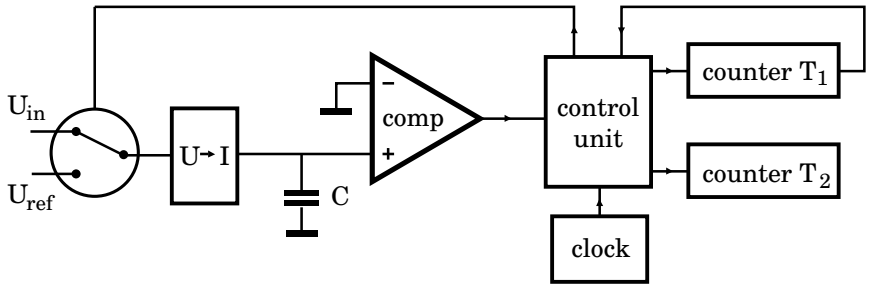
Voor elke nieuwe conversie begint de counter weer opnieuw te tellen. De gevonden waarde wordt in een register bewaard.

Voordeel: de controller-logica is heel eenvoudig (in tegenstelling tot SBA).

Nadeel: de methode is vrij traag omdat we voor twaalf bits gemiddeld 2^{11} klokpulsen kwijt zijn voordat de conversie compleet is. Ook is de conversietijd niet constant. Als men met een constante interval digitale waarden wil bepalen is dat een nadeel. De methode vereist een nauwkeurige DAC.

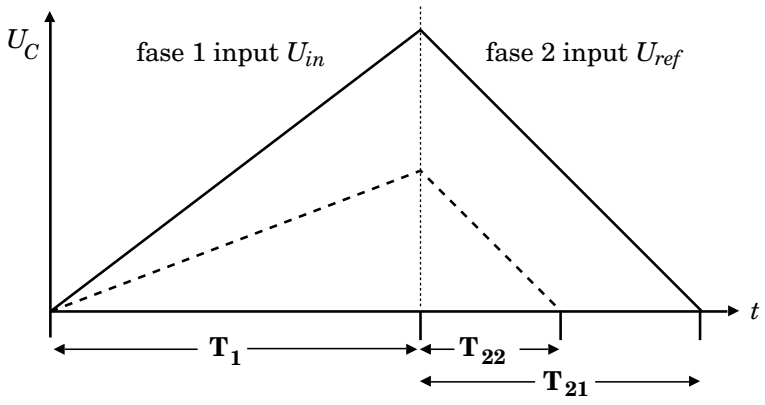
Door een zogenoemde up/$\overline{\text{down}}$ counter te gebruiken kan deze methode sneller zijn. De controller-logica stuurt dan de up/$\overline{\text{down}}$-ingang van de teller, die dan niet telkens vanaf nul hoeft te beginnen. Omdat de ADC het ingangssignaal nu 'volgt', heet deze configuratie een *tracking ADC*.

Telmethode met dual ramp integration
Dit is een methode die gebruikmaakt van twee hellingen (dual ramp) en een integrerende werking heeft. In tegenstelling tot de eerdergenoemde telmethode gebruiken we geen (meestal dure) DAC. Dit maakt de schakeling relatief goedkoop. De hellingen waar we hierover spreken zijn lineair stijgende of dalende spanningen. Deze lineair stijgende of dalende spanningen zijn weer het gevolg van het integreren (in de tijd) van een constante spanning. We geven eerst het blokschema.



Figuur 4.10 ADC met dual ramp integration

De condensator C is in eerste instantie ongeladen. Een spanningsstroomomzetter wordt door de controller op U_{in} geschakeld. De condensator wordt met een snelheid, die onafhankelijk is van de grootte van U_{in}, opgeladen. De oplaadtijd wordt bepaald door counter T_1 en is vast! Hierna wordt de ingang door de controller op een *negatieve* referentiespanning geschakeld en de condensator wordt weer ontladen. De ontlaadtijd wordt bijgehouden in counter T_2. Het tijdstip van ontladen wordt door de comparator aangegeven. Op het moment dat de condensator ontladen is, bevat T_2 een binaire waarde die evenredig is met U_{in}, waarmee de conversie voltooid is. We zullen dit verduidelijken aan de hand van figuur 4.11. In figuur 4.11 is de spanning over de condensator voor twee verschillende ingangsspanningen aangegeven. De ingangsspanning U_{in} produceert gedurende T_1 een met een bepaalde helling (ramp) stijgende spanning over C. Gedurende T_2 wordt deze spanning weer tot nul gereduceerd met een constante helling (de ingang is dan op U_{ref} geschakeld). Voor de tweede ingangsspanning, die maar half zo groot is als U_{in}, gelden de gestippelde lijnen. We zien dat T_{22} ook maar de helft van T_{21} is.

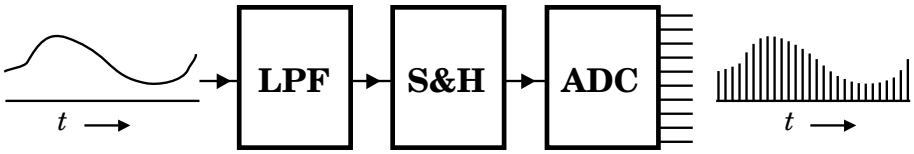


Figuur 4.11 Spanning over C als functie van de tijd voor twee verschillende ingangsspanningen

Voordeel: deze methode levert zonder dure DAC een grote resolutie. Door de integrerende werking worden eventuele storingen en ruis op de ingang weggefilterd. Deze methode leent zich ook goed voor het toepassen van BCD-counters, waardoor direct een display aangestuurd kan worden.
Nadeel: omdat dit een telmethode is, is hij ook langzaam. Gedurende een conversie moet de klokfrequentie stabiel zijn.
Toepassing: deze methode wordt toegepast in digitale voltmeters. Hier is de traagheid van de conversie meestal geen bezwaar, omdat men kan volstaan met 2 à 10 conversies per seconde.

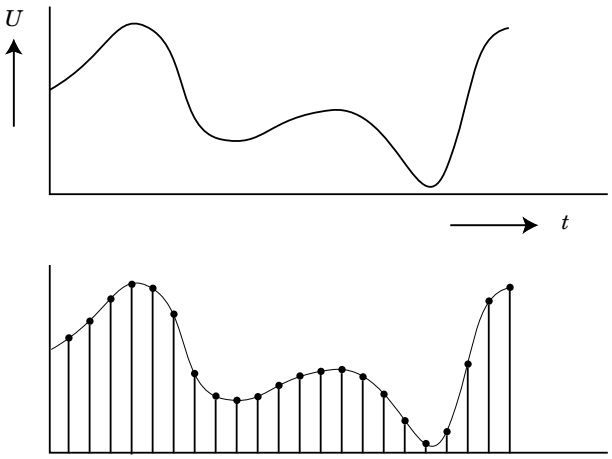
4.3 Digitaliseren van analoge signalen

En visuele voorstelling van het digitaliseren van analoge signalen is weergegeven in figuur 4.12. Links zien we een spanning die in de tijd varieert spanning. Dit signaal wordt eerst aangeboden aan een laagdoorlaatfilter (LPF, Low Pass Filter), waarover direct meer. Vervolgens wordt het signaal aangeboden aan een blokje met de naam 'sample-and-hold' (S&H). Pas daarna komt de ADC.



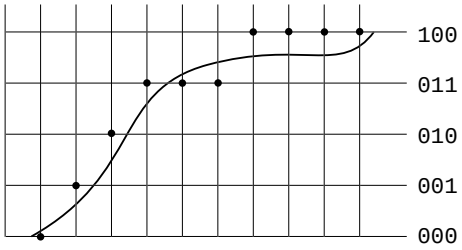
Figuur 4.12 A-D-conversieproces

Een ADC krijgt een elektrisch signaal aangeboden dat varieert in de tijd. Om de conversie goed te laten verlopen, moet de omzetter telkens een bemonstering van het in de tijd veranderende signaal nemen. Zo'n bemonstering, ook wel sample genoemd, wordt door een sample-and-holdschakeling gemaakt. Deze schakeling neemt de waarde van het signaal en levert de ADC een spanning die gedurende de conversie stabiel is.
Na de conversie wordt weer een nieuwe sample genomen en wordt opnieuw een conversie gedaan. Al deze conversies leveren steeds een (binair) getal op waarvan we de hoogte op een tijdas kunnen uitzetten, zoals in figuur 4.12 rechts is te zien.

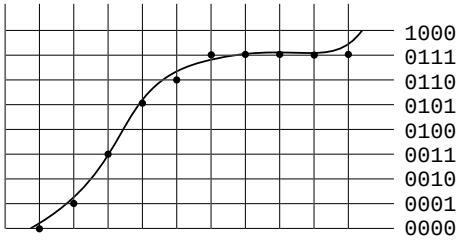


Figuur 4.13 Samplingproces

In figuur 4.13 is het ingangssignaal samen met de samples te zien. Bij de omzetting door de ADC wordt noodzakelijkerwijs een afronding gemaakt. Het maakt daarbij uit hoeveel bits we gebruiken om onze digitale waarde te vormen. Hoe meer bits, des te nauwkeuriger we het analoge signaal kunnen volgen. De kleine afwijking als gevolg van het afronden noemen we kwantisatieruis. De volgende twee plaatjes demonstreren dat een grotere resolutie een betere benadering geeft. Het lijntje geeft het analoge signaal weer. De zwarte stippen zijn de digitale samples.



Figuur 4.14 Samplefout bij een zekere resolutie



Figuur 4.15 Samplefout bij hogere resolutie

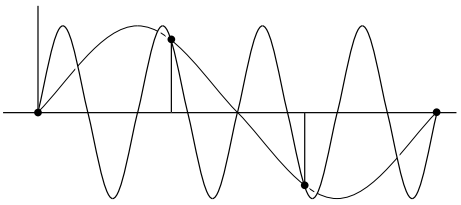
Bij de hogere resolutie liggen de zwarte stippen veel dichter bij het oorspronkelijke analoge signaal.

Gegeven de conversietijd van de ADC is er een maximum aan het aantal samples per seconde. Deze samplefrequentie brengt met zich mee dat er een bovengrens is aan

de frequentie van het ingangssignaal. Het sampling theorema zegt dat de maximale frequentie van het ingangssignaal de helft van de samplefrequentie is, dus:

$$f_{max} = \frac{f_{sample}}{2}$$

Bevat het te converteren signaal toch hogere frequenties, dan dienen deze eerst met een analoog laagdoorlaatfilter (LDF, LPF) weggefilterd te worden. Als dit niet gedaan wordt, treedt er een effect op dat de naam 'aliasing' draagt. Stel, we hebben een samplefrequentie van 1000 Hz. Wanneer we een sinussignaal van 1000 Hz met deze frequentie bemonsteren dan vinden we een constante waarde (we bemonsteren de sinus telkens op dezelfde plek). Een signaal met constante waarde is een gelijkspanning. De frequentie van een gelijkspanning is 0 Hz. We kunnen dus een signaal van 1000 Hz niet van een signaal van 0 Hz onderscheiden, vandaar de naam 'aliasing'. Zo is er bij deze samplefrequentie ook geen verschil tussen de samples van een signaal van 1 Hz en een signaal van 999 Hz. De hogere frequenties zien eruit als lage frequenties. Het lijkt of het hoogfrequente gedeelte van het spectrum gespiegeld in de lagere frequenties terugkomt.



Figuur 4.16 Sampletheorema

Een plaatje van het samplingproces met te lage samplefrequenties is in figuur 4.16 weergegeven. Te zien is dat de hoog-frequente sinus bij deze lage samplefrequentie niet meer te onderscheiden is van een sinus met veel lagere frequentie. Het ingangsfilter waarmee we de te hoge frequenties weghalen wordt ook wel anti-alias filter genoemd.

4.4 Digitale signaalbewerking

Een bemonsterd gedigitaliseerd signaal kan digitaal worden bewerkt. Het is ook mogelijk om digitaal te filteren. Ook transformaties van tijd- naar frequentiedomein en omgekeerd weer terug zijn mogelijk. De techniek om dit te doen, staat bekend als fast Fouriertransformatie (FFT). Eigenlijk is de FFT een uitgekiende versie van de discrete Fouriertransformatie (DFT). Met uitgekiend bedoelen we dat de berekening zeer efficiënt wordt uitgevoerd met als gevolg dat de transformatie snel is.
Digitale filtering en digitale signaalbewerking in het algemeen zijn in de jaren zestig in de belangstelling gekomen, omdat toen digitale systemen beschikbaar waren die snel genoeg de nodige berekeningen konden uitvoeren. Overigens moeten we onderscheid maken tussen bewerkingen op data reeksen achteraf en real-time bewerkingen die, zoals de naam al aangeeft, vrijwel direct resultaat moeten opleveren. De toepassing geeft meestal aan met welke van de twee situaties we te maken hebben.
Om de betekenis van digitale signaalbewerking wat duidelijker te maken, noemen we hier een aantal toepassingen zonder ook maar de pretentie te hebben hiermee een volledig overzicht te geven.

- Bewerkingen op geluidsdata. Muziek wordt tegenwoordig veelal digitaal geregistreerd en is ook digitaal te bewerken. Ongewenste geluiden kunnen weggefilterd

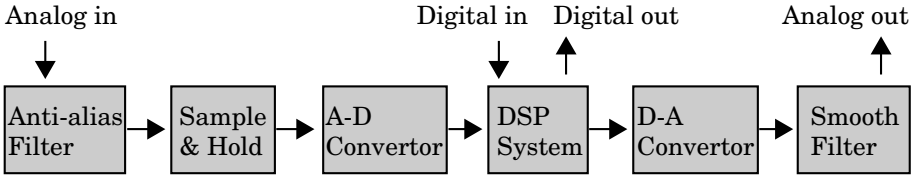
worden, verhulde fragmenten kunnen naar voren gehaald worden, enzovoort. Het analyseren van spraaksignalen en het genereren van kunstmatige spraak (spraakanalysen en synthese) zijn speciale vormen van bewerkingen op geluidsdata.

- Bewerkingen op tijdsamples van gegevens uit de meteorologie, economie, sterrenkunde en natuurkunde om cyclische patronen en trends te detecteren. Of om door sterke ruis verhulde details zichtbaar te maken.
- Bewerkingen op tweedimensionale samples van beelden van scanners. Beelden kunnen contrastrijker of juist vager gemaakt worden. Ook kunnen real-time bewerkingen toegepast worden.
- Analyse van spectra van sonarapparatuur, medische apparatuur en radar.
- Bij analoge en digitale communicatie kunnen we met succes digitale signaalbewerking toepassen om tot een meer betrouwbare of kwalitatief betere communicatie te komen. Ook GSM-toestellen maken gebruik van DSP.
- Bepaalde vormen van datacompressie kunnen met DSP-technieken gerealiseerd worden. Voorbeelden hiervan zijn jpeg en mpeg.

Voor digitale signaalbewerking bestaat een speciale klasse processors. Deze dragen de naam DSP (digital signal processor). De hardware van deze DSP's is geoptimaliseerd voor bewerkingen die bij digitale signaalbewerking een rol spelen.
Daarnaast is DSP in een heleboel toepassingen binnengeslopen. Waar vroeger analoge technieken werden ingezet worden nu steeds vaker digtale oplossingen gezocht op basis van DSP.

4.4.1 De plaats van DSP

De component waarmee DSP meestal wordt gerealiseerd heet digital signal processor, ook afgekort met DSP. De eigenschappen van deze component komt verderop in dit boek te sprake. Voorlopig richten we onze aandacht op digital signal processing. De gewenste eigenschappen van de componenten die we daarvoor nodig hebben komen dan vanzelf naar voren. Digital signal processing is het bewerken van gedigitaliseerde signalen. Een geluid is een analoog signaal. Met een microfoon is zo'n signaal om te zetten in een elektrisch signaal dat ook analoog is. Pas als we dit analoge signaal gedigitaliseerd hebben, kunnen we gebruikmaken van DSP-technieken. Meestal moet er dan aan het eind van de rit weer een omzettig plaatsvinden van digitaal naar analoog.



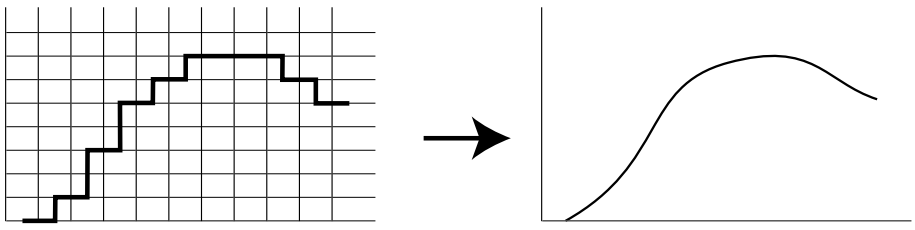
Figuur 4.17 Overzicht van DSP

In de figuur zijn de eerdergenoemde stappen weergegeven. Merk op dat digitale signaalverwerking in principe van digitale input gebruik kan maken en ook digitale output voor bijvoorbeeld opslag kan genereren. In principe lijkt het omslachtig om DSP toe te passen. We zouden namelijk ook direct van analoge technieken gebruikt kunnen maken om het gewenste resultaat te bereiken. Toch heeft de digitale benadering een aantal voordelen.

- Bij analoge technieken verouderen componenten van bijvoorbeeld een filter waardoor de werking in de tijd verandert. Bij digitale filtering speelt dit probleem niet.
- Gedigitaliseerde data zijn eenvoudig verliesvrij op te slaan om later bewerkt te worden. Bij het vastleggen van analoge signalen treedt er verlies op van informatie en zal een magnetische drager na verloop van tijd aan kwaliteit inleveren.
- Met DSP zijn bewerkingen mogelijk waar de analoge techniek te kort schiet of slechts met dure componenten te verwezenlijken is.
- Het transport van digitale informatie is verliesvrij.

We zullen de afzonderlijke blokken van de eerder getoonde figuur nu bespreken. Het eerste deel is al eerder aan de orde geweest. We passen een anti-alias filter toe vanwege de door het sampling theorema opgelegde maximale frequentie waarmee het systeem kan werken. De sample en hold bemonstert het signaal en houdt dit tijdens de conversie constant. Vervolgens komt de ADC aan de beurt. Van deze component zijn de snelheid en de resolutie belangrijke gegevens. De snelheid is van belang in verband met de maximale sample frequentie. Mocht de ADC sneller kunnen werken, dan bestaat de mogelijkheid een techniek toe te passen die bekend staat onder de naam oversampling. Oversampling is het nemen van meer samples dan strikt genomen nodig is. Dit is bijvoorbeeld handig als we een heel eenvoudig anti-alising filter gebruiken. We kunnen dan ook nog met digitale techieken filteren. De DSP kan de samples bewerken met bewerkingen die we nog zullen gaan bekijken. Na de DSP kunnen we met behulp van een DAC weer terug naar de analoge wereld. De uitgang van een DAC ziet eruit als een soort histogram waarbij de blokken tegen elkaar aan staan. Dit wijkt af van de gewenste vloeiende analoge spanning. Er bestaan twee technieken om de uitgang wat vloeiender te maken.

1. We kunnen met sample interpolatie meer samples naar de DAC sturen waarbij de nieuwe samples als een gemiddelde waarde geldt van de twee samples waar hij tussen komt. Dit effect komt beter tot zijn recht als de DAC een grotere resolutie heeft. De traptredes worden dan kleiner en fijner.
2. Een tweede mogelijkheid die vrijwel altijd wordt toegepast is een eenvoudig analoog laagdoorlaatfilter. Zo'n filter heeft een integrerende werking en zal de scherpe kantjes uit het signaal gladstrijken. In figuur 4.18 is dit te zien. Links staat de uitgangsspanning van de DAC en rechts het gladgestreken signaal uit het filter.



Figuur 4.18 Effect van laagdoorlaatfilter

4.4.2 DSP-bewerkingen

We zullen hier enkele veelvoorkomende signaalbewerkingen bespreken. Om een indruk van de bewerking te geven, maken we hier en daar gebruik van wiskunde.

Convolutie

Gegeven twee eindige rijen, $x(k)$ en $h(k)$ met lengte N_1 en N_2. De convolutie is gedefinieerd als:

$$y(n) = h(n) \otimes x(n) = \sum_{k=-\infty}^{\infty} h(k)x(n-k) = \sum_{k=0}^{M-1} h(k)x(n-k) \quad n = 0, 1, \ldots, M-1$$

Hierin is: $M = N_1 + N_2 - 1$
Convolutie zullen we als bewerking terugvinden bij digitale filters.

Correlatie

We zullen hier de wiskunde van de correlatie achterwege laten en ons beperken tot een beschrijving. Er bestaan twee vormen van correlatie, autocorrelatie en kruiscorrelatie.

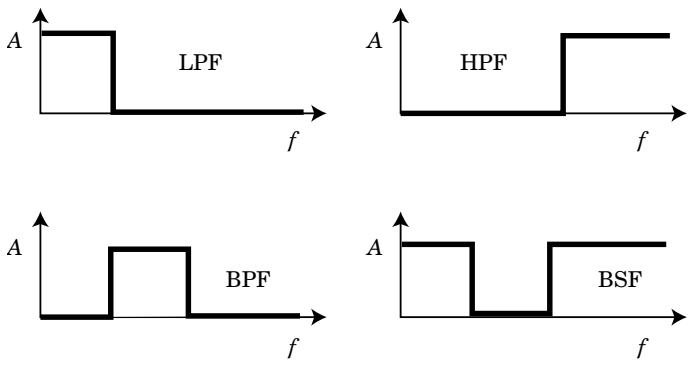
- De kruiscorrelatie is een maat voor de overeenkomsten tussen twee signalen. Zijn er geen overeenkomsten dan is de kruiscorrelatie nul. De kruiscorrelatie wordt gebruikt voor de detectie of het herstel van signalen die in ruis gedompeld zijn, zoals reflectiesignalen van radarsystemen of signalen die een lange weg afgelegd hebben en daarbij verzwakt zijn.
- De autocorrelatie heeft betrekking op maar één signaal en geeft informatie over de structuur van het signaal en het tijdsgedrag. Autocorrelatie is een speciale vorm van kruiscorrelatie en is bijvoorbeeld handig als men op zoek is naar verborgen periodiciteit.

Filtering

In het algemeen kunnen filters in vier typen verdeeld worden. Dit geldt voor zowel analoge als digitale filters:

- Low pass filter (LPF).
- High pass filter (HPF).
- Band pass filter (BPF).
- Band stop filter (BSF).

De volgende figuur geeft de filterwerking schematisch weer.

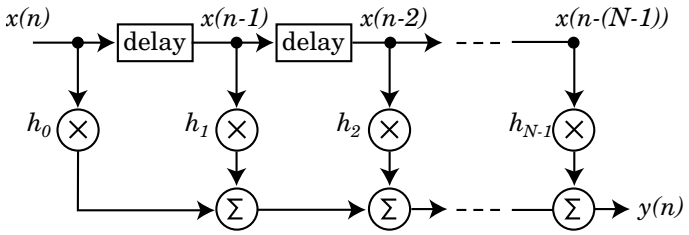


Figuur 4.19 Basisfilters

In de praktijk lukt het nooit om rechte filterflanken te krijgen. Een LPF zal bijvoorbeeld vanaf een zekere frequentie (cutoff frequency) een verzwakking geven die als een schuine lijn in de filterkarakteristiek is te zien (waarbij we de frequentieas logaritmisch uitzetten). Een eerste orde filter geeft een verzwakking van 6 dB per octaaf. Bij tweede orde filters is dit 12 dB en bij derde orde 18 dB per octaaf. Een filter kent zowel een amplitude als een fasekarakteristiek. Als de fasekarakteristiek een rechte lijn is, spreken we van een lineair fasefilter. Dergelijke filters laten de faseverhoudingen van de onderlinge frequentiecomponenten intact. Gevolg is dat het signaal, afgezien van het filtereffect, zijn oorspronkelijke vorm behoudt. Spitsen we ons toe op digitale filtering, dan vinden we als algemene vorm van een digitaal filter (van het type FIR, zie verderop) de volgende vergelijking:

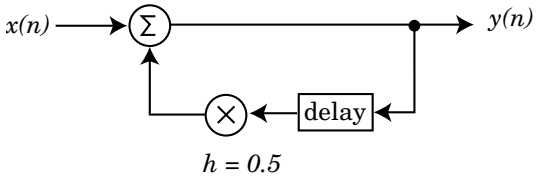
$$y(n) = \sum_{k=0}^{N-1} h(k)x(n-k)$$

Hierin zijn $x(n)$ en $y(n)$ de input en de output van het filter en $h(k)$ met $k = 0, 1, \ldots, N-1$ de filtercoëfficiënten. Bij digitale filters onderscheiden we twee typen: filters die wel of geen terugkoppeling hebben. Zonder terugkoppeling krijgen we filters van het type FIR (Finite Impulse Response). Een FIR heet ook wel een niet-recursief filter. De algemene opbouw van een FIR is te zien in de figuur 4.20.



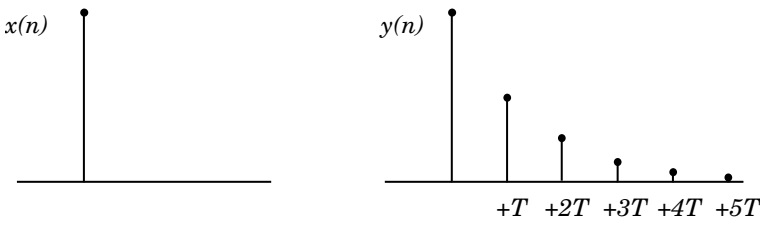
Figuur 4.20 Finite Impluse Response

Passen we terugkoppeling toe dan hebben we te maken met IIR-filters (Infinite Impulse Response). Dit type filter wordt ook wel een recursief filter genoemd. De uitgang van dit type filter is een gewogen gemiddelde van ingangswaarden en teruggekoppelde uitgangswaarden. Een simpel systeem waarin deze terugkoppeling is weergegeven is te zien in de figuur 4.21. We hebben slechts één delay element toegepast.



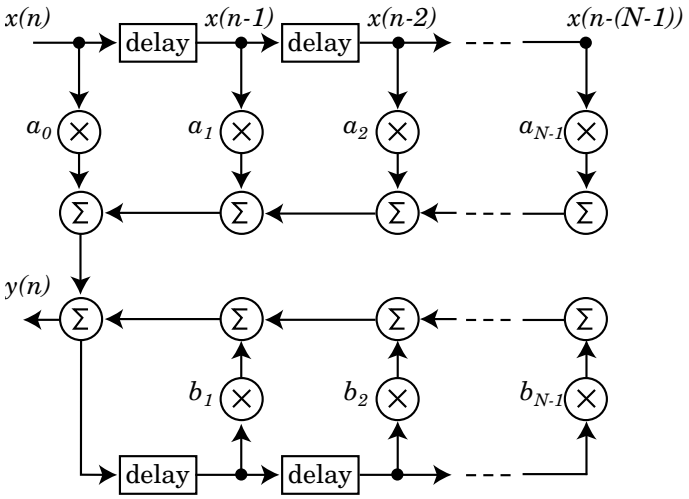
Figuur 4.21 Infinite Impulse Response

Dit filter geeft op de eenheidspuls een reeks afnemende uitgangspulsen, zie figuur 4.22.



Figuur 4.22 Voorbeeld van IIR filter

Een algemeen schema van een IIR-filter is weergegeven in figuur 4.23.



Figuur 4.23 Realisatie van een IIR-filter

Een FIR is onvoorwaardelijk stabiel en geeft lineaire fasedraaing. Dit laatste is noodzakelijk bij toepassingen waarin men fasevervorming wil vermijden zoals bij bewerking van audiosignalen. Een IIR is vaak eenvoudiger van uitvoering en ook sneller. Meestal hebben we geen lineaire fasedraaiing en is het filter niet onvoorwaardelijk stabiel.

Transformatie

We bekijken in dit onderdeel alleen de Fouriertransformatie. Overigens kent de verwante cosinustransformatie een interessante toepassing namelijk jpeg- en mpeg-beeldcompressie. Met de Fouriertransformatie maken we een overgang van het tijdsdomein naar het frequentiedomein. Als we het gedrag van een signaal in de tijd kennen, geeft de Fouriertransformatie een overzicht van de frequentiecomponenten waaruit het signaal opgebouwd is. Zo levert de Fouriertransformatie van een zuivere sinus in het tijdsdomein één enkele piek in het frequentiedomein. Dit is namelijk de frequentie van het sinussignaal. Complexere vormen van signalen kan men opgebouwd denken uit een samenstelsel van sinusvormige signalen met zekere amplitudes en frequenties. De Fouriertransformatie laat zien uit welke frequenties het signaal bestaat. Van deze frequenties geeft de Fouriertransformatie ook de amplitudes en onderlinge faseverhoudingen.
De Fouriertransformatie wordt gebruikt voor signaalanalyse, signaalfiltering en signaalbewerkingen die in het frequentiedomein eenvoudiger zijn uit te voeren dan in

het tijdsdomein. We voeren bij de laatste toepassing de transformatie uit, passen de bewerking toe en transformeren terug naar het tijdsdomein. Het terugtransformeren heet de inverse Fouriertransformatie.

DFT
Voor samplesets van signalen is een speciale vorm van de Fouriertransformatie van toepassing namelijk de discrete Fouriertransformatie. De discrete Fouriertransformatie ziet er in formulevorm als volgt uit:

$$F_m = \sum_{n=0}^{N-1} f_n e^{-i(2\pi mn/N)} \quad m = 0, 1, \ldots, N-1$$

De inverse tranformatie heeft de volgende vorm:

$$f_n = \frac{1}{N} \sum_{m=0}^{N-1} F_m e^{i(2\pi mn/N)} \quad n = 0, 1, \ldots, N-1$$

Feitelijk levert de Fourriertranformatie een transformatie van samples f_n in het tijdsdomein, naar een spectrum F_m in het frequentiedomein. We substitueren nu een eenvoudige schrijfwijze voor de complexe e-macht:

$$W_N = e^{-i\frac{2\pi}{N}}$$

zodat:

$$W_N^{mn} = e^{-i2\pi mn/N}$$

De DFT krijgt dan de vorm:

$$F_m = \sum_{n=0}^{N-1} f_n W_N^{mn} \quad m = 0, 1, \ldots, N-1$$

Het term voor term berekenen van F_m levert in totaal N^2 vermenigvuldigingen van complexe getallen op.

FFT
De fast Fouriertransformatie (FFT) is een uitgekiende methode voor het berekenen van de DFT. Hierin is een aantal optimalisaties doorgevoerd, waardoor het aantal vermenigvuldigingen is teruggebracht tot $\frac{N}{2}\log_2 N$. Dit is van de orde $N \log_2 N$. Als we dit voor een sampleset van 1024 doorrekenen dan is:

$$N^2 = 1048576 \quad en \quad N\log_2 N = 10240$$

De FFT levert dus een enorme besparing in rekentijd. Kijken we naar een voorbeeld met acht samples, dan vinden we voor de eerste drie frequentietermen de volgende vergelijkingen:

$$F_0 = f_0W_8^0 + f_1W_8^0 + f_2W_8^0 + f_3W_8^0 + f_4W_8^0 + f_5W_8^0 + f_6W_8^0 + f_7W_8^0$$

$$F_1 = f_0W_8^0 + f_1W_8^1 + f_2W_8^2 + f_3W_8^3 + f_4W_8^4 + f_5W_8^5 + f_6W_8^6 + f_7W_8^7$$

$$F_2 = f_0W_8^0 + f_1W_8^2 + f_2W_8^4 + f_3W_8^6 + f_4W_8^0 + f_5W_8^2 + f_6W_8^4 + f_7W_8^6$$

Hierbij hebben we gebruikgemaakt van het feit dat de complexe e-macht net als de sinus en de cosinus periodiek is. Voor een complexe e-macht geldt:

$$e^{ix} = \cos x + i \sin x$$

$$e^{\frac{1}{2}\pi i} = i \qquad e^{\pi i} = -1 \qquad e^{-\frac{1}{2}\pi i} = -i$$

Voor onze schrijfwijze kunnen we de volgende gelijkheden opstellen:

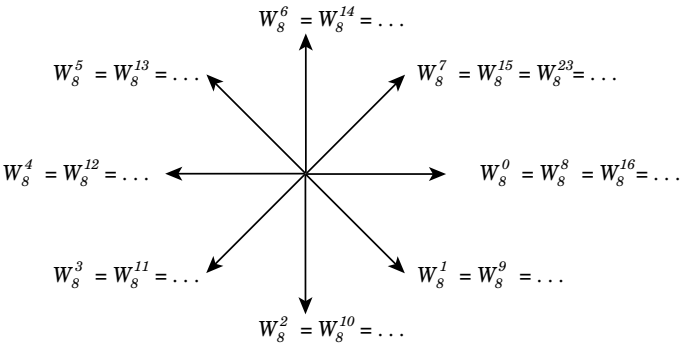
$$W_8^0 = 1 \quad W_8^0 = W_8^8 = W_8^{16} \ldots \qquad W_8^1 = W_8^9 = W_8^{17} \ldots$$

Wanneer we nu naar de berekeningen van F_m kijken, dan zien we nogal wat terugkerende producten. Daarnaast kunnen we gebruikmaken van het periodiek zijn van W_8^{mn}. Dankzij deze redundantie kunnen we de reductie van vermenigvuldigingen realiseren voor de FFT. Een extra actie die relatief eenvoudig van aard is, is dat we de volgorde van de te transformeren samplereeks moeten wijzigen. De samples worden gezet in de volgorde 0, 4, 2, 6, 1, 5, 3, 7. Schrijven we deze getallen in binaire vorm, dan krijgen we:

000 100 010 110 001 101 011 111

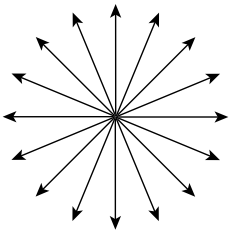
Lezen we de reeksjes van elk drie bits verkeerd om, dus met het LSB links in plaats van rechts, dan krijgen we weer de oorspronkelijke nummering terug. We noemen dit decompositie met bit reversal.

We zullen proberen om aan de hand van een aantal figuren het transformatieproces weer tegeven. Bekijken we FFT op een sampleset van 8, dan vinden we de volgende figuur voor de mogelijke W's.



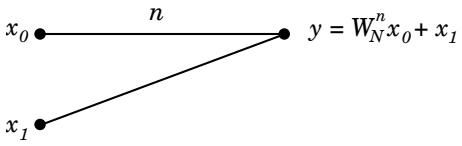
Figuur 4.24 Fasehoeken acht-punts FFT

Voor een sampleset van 16 krijgen we een vergelijkbaar plaatje.



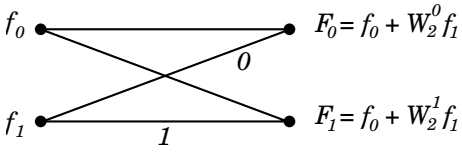
Figuur 4.25 Fasehoeken zestien-punt FFT

De vermenigvuldiging en optelling die we bij de DFT (en FFT) zien kunnen we grafisch weergeven in een flowdiagram, zie figuur 4.26.



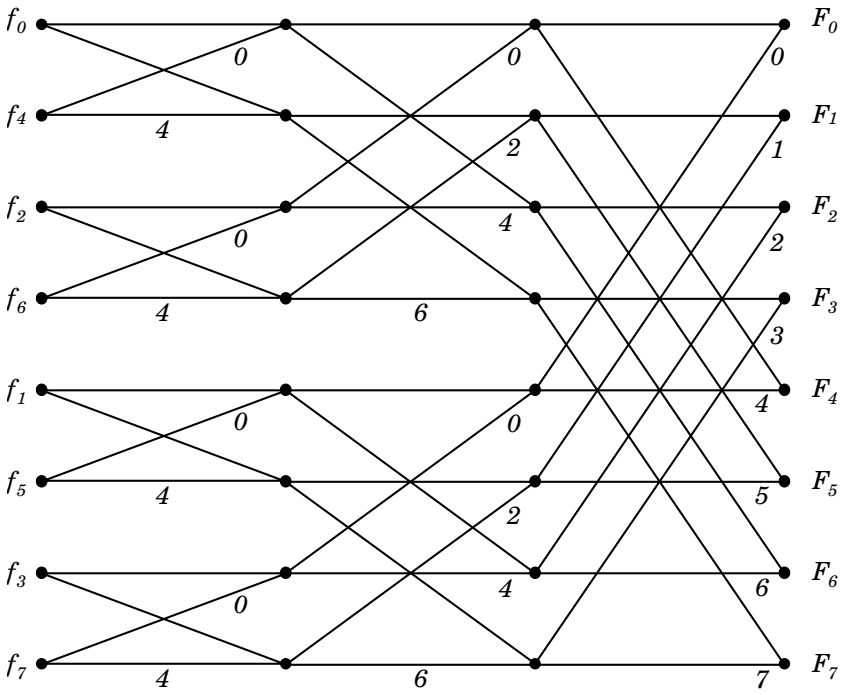
Figuur 4.26 Signal flowdiagram

Met een flowdiagram is een FFT voor twee samples in beeld te brengen. Een dergelijk plaatje noemt men een butterfly, zie figuur 4.27.



Figuur 4.27 Butterfly voor twee-punts FFT

Als we een FFT voor acht samples in beeld brengen, zien we een figuur met drie passes. We beginnen met butterflies voor buursamples (die overigens wel via input-bit reversal gesorteerd zijn). Na drie slagen van doorrekenen van een verzameling van steeds wijder wordende butterflies, komen we bij het resultaat. Elke slag noemen we een pass. Voor ons voorbeeld hebben we drie passes nodig. Algemeen geldt dat voor een sampleset van N samples $\log_2 N$ passes nodig zijn. Elke pass bevat $N/2$ butterflies.



Figuur 4.28 Acht punts FFT

Modulatie

Als laatste bewerking van DSP noemen we modulatie. Voor transport van signalen over grote afstanden is het vaak nodig het signaal aan de eigenschappen van het transportmedium aan te passen. Zo is een analoge telefoonlijn niet in staat direct digitale signalen te transporteren. We moeten modulatie toepassen om dit wel mogelijk te maken. Veel modems gebruiken DSP-technieken om dit te bewerkstelligen. De inzet van DSP is vooral belangrijk wanneer we met een gegeven bandbreedte van het communicatiekanaal zoveel mogelijk bits per seconde willen transporteren.

4.5 Opgaven

1. Een audio-cd bevat geluidsamples die met een frequentie van 44 kHz zijn vastgelegd. Wat is de maximaal haalbare frequentie van het geluid?
2. Hoe groot is de resolutie van een 14-bits DAC?
3. Wat is een flash convertor? Waar past men deze conversie toe?
4. Beschrijf de werking van een ADC die gebaseerd is op dual ramp integration.
5. Noem enkele toepassingen van digitale signaalbewerking.
6. Waartoe dient een sample-and-holdschakeling?
7. Wat is aliasing?

8. Waarom is het anti-aliasing filter altijd een analoog filter, met andere woorden: waarom kan dit niet achteraf digitaal gedaan worden?

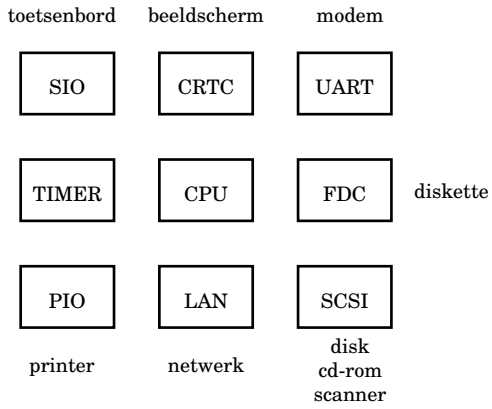
9. Noem enkele verschillen tussen een FIR- en een IIR-filter.

5 Input en output

In dit hoofdstuk gaan we wat nader in op input en output ofwel I/O.

5.1 Typen I/O-poorten

De verscheidenheid aan I/O-poorten is groot. In figuur 5.1 is de CPU weergegeven, omringd door allerlei I/O-chips die weer verschillende I/O-taken afhandelen.



Figuur 5.1 CPU omgeven door I/O-chips

Een groep I/O-poorten is toegespitst op de *seriële datacommunicatie*. Dit is een vorm van datatransport waarbij de bits na elkaar over een verbindingslijn getransporteerd worden. I/O-chips die deze communicatie ondersteunen, dragen namen als: UART (universal asynchronous receiver and transmitter), USART (universal asynchronous and synchronous receiver and transmitter), ACIA (asynchronous communication interface adapter) en SIO (serial input output).
Bij *parallelle I/O* worden meer bits tegelijk van de buitenwereld ontvangen, respectievelijk naar de buitenwereld verzonden. Afkortingen die hier gebruikt worden zijn: PPI (parallel periferal interface), PIA (parallel interface adapter) en PIO (parallel input and output).
Vrijwel elk computersysteem beschikt over één of meer timers. Deze timers kunnen de CPU om de zoveel tijd een interrupt zenden. Een speciale timer chip wordt ook wel *real-time clock* genoemd en bevat naast de tijd informatie over de datum. Deze chips hebben vaak een kleine batterij in de buurt om de werking te garanderen als de computer wordt uitgezet. Let op: de CPU-klok is een timing-signaal dat de werking van de CPU en daarmee van het hele systeem stuurt, dit is niet de timer chip waar we hier over spreken.
Naast deze algemene I/O zijn er chips die speciale I/O-taken mogelijk maken. We spreken van *dedicated I/O-chips*. Voorbeelden hiervan zijn:

- chips die de disk I/O verzorgen;
- LAN (local area network) controllers;

- SCSI-controllers (een speciale bus voor randapparatuur);
- CRTC (cathode ray tube controller): levert de signalen voor grafische output op een beeldscherm;
- GCC (graphic controller chips): deze leveren geavanceerde ondersteuning voor beeldschermaansturing;
- DMA-controllers: deze voeren *direct memory access*-taken uit ter ondersteuning van een andere I/O-chip.

5.2 Configureren van de I/O-poort

I/O-poorten worden tegenwoordig zelden meer uit discrete logicabouwstenen opgebouwd maar zijn veelal min of meer complexe geïntegreerde schakelingen, waarvan de functionaliteit met behulp van software nog nader te specificeren is. Een I/O-schakeling die voor seriële I/O wordt gebruikt kan met de juiste codes verteld worden hoe het datacommunicatieprotocol er precies uit zal gaan zien. Om een schakeling op de juiste manier te gebruiken, is een zogenoemde *initialisatie* van de I/O-chip nodig. Dit kan één keer gebeuren bij het opstarten, maar het is ook mogelijk dat er per applicatie een nieuwe initialisatie nodig is of dat in een applicatie de initialisatie veranderd moet worden.
De functionaliteit van een I/O-chip wordt vastgelegd door in één of meer controleregisters van die chip de juiste codes te schrijven. Als het om meer controleregisters gaat, is soms ook de volgorde van beschrijven van belang. We noemen dit het *configureren* of *programmeren* van de I/O-poort. We zullen eens op een rijtje zetten welke programmeermogelijkheden er zijn:

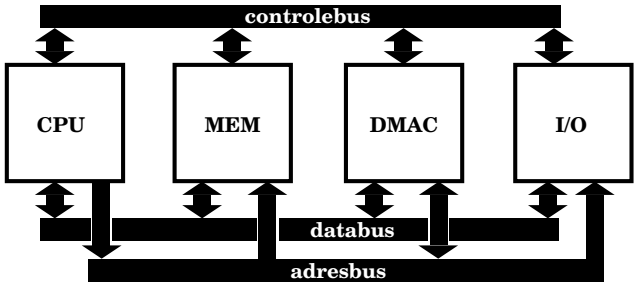
- Er is een controleregister dat door de CPU geadresseerd kan worden en gevuld met de juiste code (bijvoorbeeld de 6852-ACIA).
- Er zijn meer controleregisters die afzonderlijk door de CPU geadresseerd kunnen worden en gevuld met de juiste codes (voorbeeld: de Zilog-8553 serial I/O-chip).
- Er zijn veel controleregisters die door de CPU maar via één geheugenadres bereikt kunnen worden. Naast de controleregisters kan de CPU via een extra zogeheten pointer-register aangeven welk controleregister hij op het controleregisteradres verwacht (voorbeeld: Motorola-6845 CRT-controller).
- De I/O-chip is in staat zelf zijn informatie met DMA-technieken uit het RAM- of ROM-geheugen te halen. De CPU beschrijft in dat geval een adres- of pointerregister in de I/O-schakeling die aangeeft waar de I/O-chip de programmeercode moet zoeken. Vaak is er dan toch één direct adresseerbaar controleregister waarmee de chip geactiveerd kan worden. (voorbeeld: de Lance-ethernet-chip).

5.2.1 Interrupts

De meeste I/O-chips hebben de mogelijkheid een interrupt te genereren. De voorwaarden voor het geven van een interrupt worden dan bij de initialisatie vastgelegd. Aangezien het interrupt-mechanisme hardware-afhankelijk is (wel of geen interruptvectoren, priority daisy chains, enzovoort), zijn I/O-chips ofwel toegesneden op een bepaalde CPU of CPU-familie, ofwel op interrupt-hardware die configureerbaar is bij initialisatie. In het eerste geval zijn deze chips met wat extra hardware of in beperkte mate inzetbaar in 'vreemde' omgevingen. Datasheets of application notes van de betreffende I/O-chip geven vaak uitgewerkte voorbeelden van deze situaties.

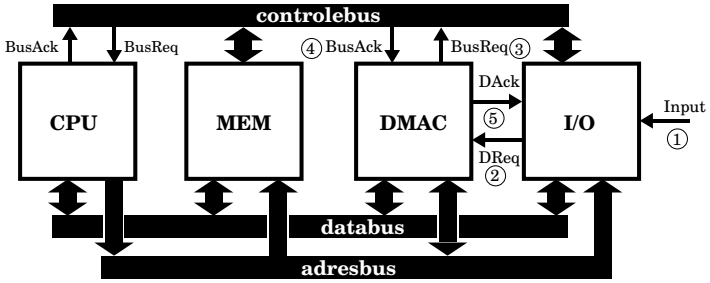
5.2.2 DMA-technieken

DMA staat voor *direct memory access*. Bij deze techniek wordt de CPU bij de I/O-afhandeling geholpen door een DMA-controller een zogeheten DMAC. Figuur 5.2 geeft het algemene blokschema van een computersysteem de plaats van een DMAC.



Figuur 5.2 Direct memory access (DMAC)

Stel, de CPU wil een blok met 512 bytes van een schijf lezen, dan zou de disk-controller bij elke byte een interrupt kunnen genereren. De CPU haalt dan in de interrupt-service-routine de byte op en plaatst die in een van tevoren gereserveerd stukje geheugen. Deze actie kost echter relatief veel CPU-tijd, omdat de CPU bij een interrupt nogal wat overhead nodig heeft om registers naar de stack te schrijven en ze er weer af te halen. Een DMAC maakt het leven van de CPU een stuk aangenamer.
Als er een byte van schijf beschikbaar komt, meldt de diskcontroller dit aan de DMAC. De DMAC heeft van tevoren van de CPU vernomen waar de diskdata in het geheugen moeten komen en om hoeveel bytes het gaat. De DMAC vraagt aan de CPU of eventueel een andere busmaster om de systeembus op het moment dat er data beschikbaar komen. Als de bus beschikbaar is, voert de DMAC de schrijfcyclus uit in samenwerking met de I/O-chip (in dit geval dus de diskcontroller).



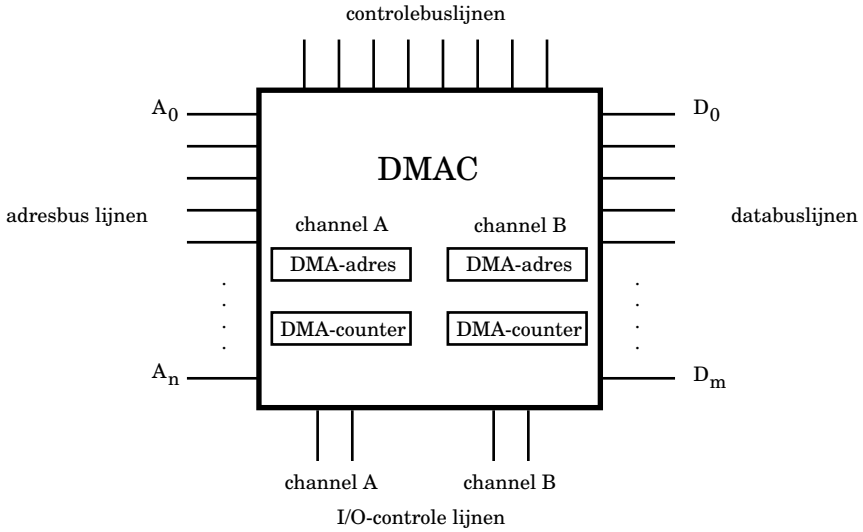
Figuur 5.3 Volgorde van gebeurtenissen bij DMA

De gang van zaken is weergegeven in figuur 5.3. Achtereenvolgens vinden de volgende gebeurtenissen plaats (in de figuur aangegeven als omcirkelde cijfertjes):

1. er is input;
2. de I/O geeft een data request-signaal aan de DMAC;
3. de DMAC vraagt met een bus request om de bus;
4. de CPU geeft een bus acknowledge ten teken dat de bus beschikbaar is;

5. de DMAC geeft aan het I/O-device een data acknowledge, zodat de data via de databus naar een adres in het geheugen geschreven kunnen worden. Het I/O-device zet de data op de databus, de DMAC levert het adres en de sturing van de controlebussignalen.

Deze manier van werken is aanzienlijk sneller dan het afhandelen van een interrupt-routine voor elke byte. Na afloop van het hele datablok krijgt de CPU pas een interrupt om aan te geven dat de hele leesactie voltooid is. De DMAC is een support-chip die dus dient ter ondersteuning van de I/O. Figuur 5.4 geeft een idee van de interne opbouw van een DMAC. Vaak zijn er ten behoeve van verschillende I/O-chips afzonderlijke kanalen beschikbaar. Elk kanaal heeft een adresregister, waarin de CPU aangeeft waar de data geplaatst moeten worden en een countregister waarin de hoeveelheid bytes aangegeven wordt.
In plaats van een DMAC kan ook een 'gewone CPU' aan een I/O-subsysteem toegevoegd worden om de DMA-functie uit te voeren. Deze CPU wordt dan alleen voor I/O- en DMA-activiteiten ingezet. De software voor deze CPU bevindt zich meestal in een ROM die aan het I/O-subsysteem is toegevoegd. Via een aantal registers die zowel voor de I/O-CPU als voor de systeem-CPU toegankelijk zijn, kan de systeem-CPU informatie aan de I/O-CPU doorgeven. Net als bij een DMAC gaat het hier om de plaats waar de data naar toe moeten (of vandaan moeten komen) en de hoeveelheid data.



Figuur 5.4 Opbouw van een 2-kanaals DMAC

5.3 Opgaven

1. Beschrijf de werking van DMA.
2. Wat verstaat men onder configureren van een I/O-poort?

6 Device drivers en OS

6.1 Inleiding

Wanneer een computersysteem wordt aangezet, zal het voor gebruikers niet toegankelijk zijn als het systeem niet met een programma in de juiste banen wordt geleid. Zo'n programma dat het computersysteem laat werken, noemt men een besturingssysteem of een operating system. In eerste instantie komt de CPU na een power-up of een reset in een programma terecht dat in ROM is opgeslagen. Dit programma is een hulpprogramma om het eigenlijke besturingsprogramma of operating system binnen te halen. De code in ROM noemt men bootstrapprogramma. Vaak haalt de ROM-bootstrap een wat meer uitgebreid bootstrapprogramma van disk. Deze tweede of secondary bootstrap haalt dan het operating system binnen. Het besturingsprogramma, dat op deze wijze met het bootstrapprogramma van de disk wordt gehaald, zal ervoor zorgen dat de computer toegankelijk wordt voor de gebruiker(s) en zich op een voorgeschreven manier zal gaan gedragen.

6.2 Taken van het operating system

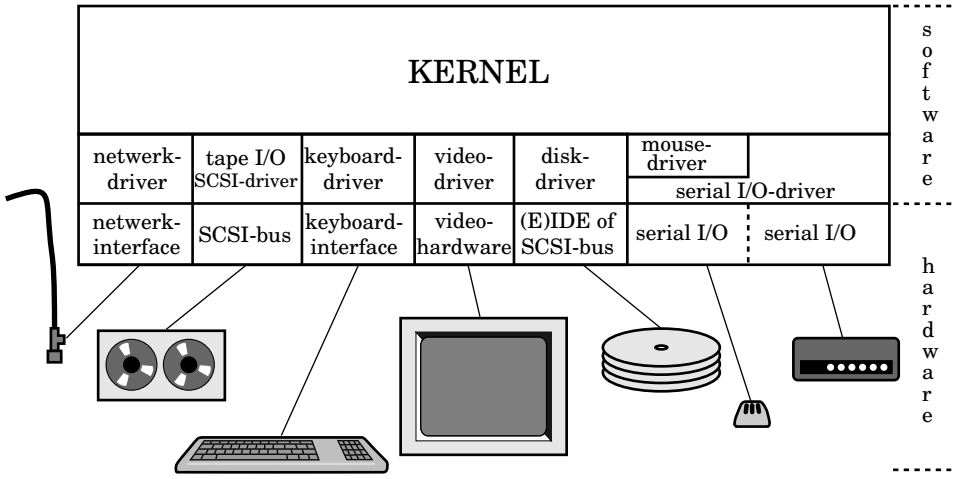
Een operating system of OS zal het functioneren van een computer regelen:

- Het regelt de communicatie met de randapparaten (disk, tape, terminal).
- Het verdeelt, regelt en organiseert de systeem-resources (CPU-tijd, geheugen, diskruimte).
- Het biedt de gebruiker de mogelijkheid op een hardwareonafhankelijke manier van het systeem gebruik te maken.

6.3 Kernel, device drivers

Het operating system bestaat uit een machineonafhankelijk deel en een machineafhankelijk deel. Het hardwareonafhankelijke deel noemt men de *kernel*. Omdat het operating system met randapparatuur moet communiceren, worden aan deze kernel stukken programma toegevoegd die op een voorgeschreven manier de communicatie met deze randapparaten of *devices* regelen. Deze programma's noemt men *device drivers*.

In figuur 6.1 is een operating system van een pc of werkstation schematisch weergegeven. De device drivers vormen de koppeling met de hardware waar de randapparatuur weer op aangesloten is.



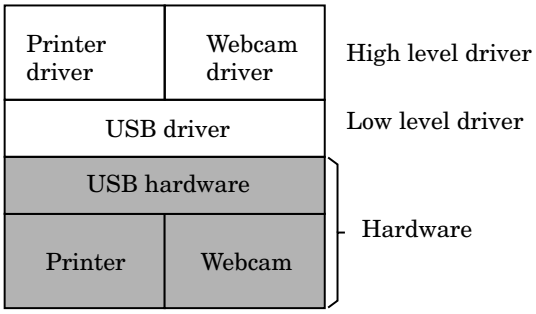
Figuur 6.1 Kernel met device drivers

6.4 Device drivers

Devicedrivers zijn softwarecomponenten waarmee we de I/O aansturen. Meestal vormen device drivers een onderdeel van een operating system. De meeste moderne operating systems bieden de mogelijkheid device drivers aan het operating system toe te voegen. Sommige systemen moeten dan opnieuw gestart worden, de wat beter doordachte systemen laten het toevoegen en eventueel verwijderen van device drivers zonder herstart (reboot) toe. Hardwarefabrikanten leveren via internet, cd-rom of diskette device drivers voor populaire operatings systems met de hardware mee.

6.4.1 Low- en highlevel drivers

De eigenlijke aansturing van de hardware wordt ook wel de low-level device driver genoemd. Bij bepaalde I/O-poorten is het met deze device driver mogelijk om met de aangesloten hardware te communiceren. Dit is dan niet het hele verhaal, want de hardware kan wel eens heel divers zijn. Kijk bijvoorbeeeld naar USB of SCSI dan is de lowlevel of generieke driver nog niet genoeg om van alle mogelijk aansluitbare hardware alle eigenschappen te benutten. En tweede stuk van een driver, die we high level driver noemen, kan dan in een applicatie of als extra toevoeging aan het operating system zorgdragen voor het aansturen van de specifieke hardware. Figuur 6.2 laat zien hoe dat er voor USB uitziet, als we daar een webcam en een printer op aangesloten hebben.



Figuur 6.2 Low level en high level driver

De highlevel driver maakt gebruik van de low level driver. De lowlevel driver bevindt zich vrijwel altijd in kernel space. De highlevel driver kan in user space zitten, dit is het deel waar ook de applicaties draaien.

6.4.2 Interrupts

Interrupt vormen het mechanisme om relatief trage I/O of I/O-events die op een willekeurig moment optreden met een systeem te combineren, waar de CPU zo efficiënt mogelijk wordt ingezet. Hiermee bedoelen we dat de CPU pas op de I/O reageert als dat ook werkelijk nodig is. We laten de CPU geen tijd verspillen met polling. Bij polling vraagt de CPU met enige regelmaat aan de I/O-devices of er nog wat te doen is. Interrupts worden als hardwaresignalen naar de CPU gestuurd en deze zal met een speciaal stukje software (de interrupt-afhandelingsroutine) daarop reageren. De interrupt-afhandelingsroutine wordt meestal gevonden met een tabel van adressen (pointers) in het geheugen op een speciaal voor een specifieke CPU gereserveerde plaats. Deze tabel heet de interrupt-vectortabel. Dus we krijgen het volgende scenario: de CPU krijgt een interrupt en zoekt in de interrupt-vector het adres van de software (machinecode) waarnaartoe gesprongen moet worden. Het adres van de interrupt-afhandelingroutine wordt zo gevonden en de routine wordt uitgevoerd.
De interrupt-afhandeling is een onderdeel van de device driver aangezien in de routine ook de hardware van het I/O-device moet worden aangestuurd om er bijvoorbeeld voor te zorgen dat het I/O-device ophoudt met het geven van een interruptsignaal. Hoe dat gaat is sterk afhankelijk van de toegepaste hardware.

6.4.3 Taken van device drivers

De software waar een lowlevel device driver uit bestaat kent een viertal specifieke taken:

1. Initialisatie. De meeste I/O-hardware zal bij het inschakelen van de voedingsspanning (power-up) of na een reset in een inactieve toestand verkeren en wachten op een actie van de CPU, waarbij interne registers met de juiste configuratiecodes gevuld worden. Als dit proces, dat we initialisatie noemen, heeft plaatsgevonden, zal het device gaan functioneren.

2. Herconfiguratie. Na initialisatie is de I/O-hardware klaar voor gebruik. Het kan echter voorkomen dat de oorspronkelijke initialisatie moet worden herzien. Neem bijvoorbeeld de initialisatie van een seriële poort, waarbij het aantal bits

in een frame bij initialisatie is vastgelegd. Een applicatie zoals een modem-aanstuurprogramma kan een ander frame formaat eisen en dan moeten we de initialisatie aanpassen. We noemen dit herconfiguratie.

3. Aansturing. Dit is de hoofdtaak van de devicedriver. Via system calls maakt een applicatie gebruik van de I/O. Het operating system zal de data doorsturen naar het I/O-device of eigenlijk de device driver. Het operating system dient er zorg voor te dragen dat de dataoverdracht van of naar het device op de juiste wijze in zijn werk gaat. Als er bijvoorbeeld twee processen iets willen printen, moet de uitvoer niet door elkaar lopen. Het device moet voor een proces tijdelijk geblokkeerd zijn zolang het andere proces daarmee bezig is. Dit is een zaak van het operating system en niet van de device driver. De driver zorgt voor de dataoverdracht. Meestal zorgt de aansturing voor het aanbieden van data aan de I/O voor verzending naar de buitenwereld, of voor het klaarzetten van de I/O om data van de buitenwereld te ontvangen. Het tweede deel van de I/O-tranfer vindt meestal plaats in de interupt-afhandelingsroutine. Tenminste als we met I/O op basis van interrupt werken, wat vrijwel altijd het geval is. Passen we geen interrupts toe dan moet de CPU de I/O met enige regelmaat vragen of de transfer al heeft plaatsgevonden. We noemen deze manier van werken 'pollen'. Pollen lijkt op het gebruik van een telefoontoestel zonder bel of ander signaal, waarbij de gebruiker telkens de hoorn opneemt om te horen of er toevallig iemand aan de lijn is.

4. Interrupt afhandeling. Een device driver levert ook het gedeelte van de interrupt-afhandelingsroutine voor het betreffende I/O-device. Hiervoor moet de interrupt-vectortabel zo aangepast worden dat daadwerkelijk de interrupt afhandeling bij de juiste machinecode van de device driver terechtkomt. Het kan ook zijn dat er een generieke afhandelingsroutine de echte interrupt-afhandelingsroutine (van de driver) aanroept. In de interrupt-afhandelingsroutine worden de taken uitgevoerd die als gevolg van de interrupt gedaan moeten worden, zoals bijvoorbeeld data van het I/O-device ophalen. Daarnaast zal een interrupt-afhandelingsroutine het I/O-device zodanig aanspreken dat het interruptsignaal, afkomstig van het device, niet meer actief blijft. Soms wordt dit bereikt met behulp van een hardwaresignaal. Soms wordt de interrupt uitgezet als het I/O-device door de CPU aangesproken wordt, soms moet de CPU via een bit in een controlregister van het I/O-device de interrupt uitzetten. Een derde actie die als gevolg van de interrupt uitgevoerd moet worden behoort tot het operating system. Er moet namelijk uitgezocht worden welk proces op de interrupt zat te wachten. Dit proces wordt dan weer in de rij van uitvoerbare processen gezet. Het proces wordt weer 'runnable'. Samengevat hebben we dus de volgende drie dingen:

   (a) datatransfer afronden;

   (b) interruptsignaal uitzetten;

   (c) juiste proces zoeken en wakker schudden.

6.5 Eenvoudige driver

6.5.1 Inleiding

Om te laten zien hoe een hardware level device driver in C geschreven kan worden, kiezen we een eenvoudig device dat voor seriële datacommunicatie in te zetten is.

6.5.2 ACIA

Een ACIA is een asynchronous communication interface adapter. Dit is inmiddels al een bejaarde chip, maar door zijn eenvoud goed te gebruiken voor ons voorbeeld. We bekijken eerst de functionaliteit, en vervolgens de registerset. Hardwarematig ziet de ACIA eruit als een chip met aansluitingen voor de systeembus en connectie met de buitenwereld. De systeembusaansluitingen zijn de datalijnen, de Interruptlijn en controlbuslijnen zoals R/W Voor de buitenwereld heeft de chip de TxD (transmit data), RxD (receive data) aansluitingen en enkele V24 signalen zoals CTS, RTS en DCD. Voor de CPU ziet de ACIA eruit als twee geheugenlocaties. Intern zijn er echter vier registers. Het hangt van R/W af welk registerpaar er gekozen wordt. Bij lezen hebben we Status Register en Receive Data Register. Voor schrijven hebben we achtereenvolgens het Control Register en het Transmit Data Register. Globaal is de werking als volgt: De CPU legt in het control register de werking van de chip vast. Via het transmit data register kan de CPU zenden, via het receive data register kan data ontvangen worden. In het statusregister is de status en werking van de ACIA in de gaten te houden. De programmeur moet zich realiseren dat data die naar het Control Register weggescherevn worden, niet van diezelfde lokatie terug te lezen zijn. Bij het lezen krijgen we namelijk de inhoud van het Status Register.

```
ACIA 6850
UART register usage
offset from startaddress
|
0  control/status
   w ......00  counter/1
   w ......01  counter/16
   w ......10  counter/64
   w ......11  master reset
   w ...000..  7 bit, even parity, 2 stopbit
   w ...001..  7 bit,  odd parity, 2 stopbit
   w ...010..  7 bit, even parity, 1 stopbit
   w ...011..  7 bit,  odd parity, 1 stopbit
   w ...100..  8 bit,   no parity, 2 stopbit
   w ...101..  8 bit,   no parity, 1 stopbit
   w ...110..  8 bit, even parity, 1 stopbit
   w ...111..  8 bit,  odd parity, 1 stopbit
   w .00.....   rts,  dis/en xmit interrupt
   w .01.....   rts,  en/dis xmit interrupt
   w .10.....  -rts,  dis/en xmit interrupt
   w .11.....   rts,  dis/en xmit intrpt, xmit brk-level
   w 1.......   en/-dis receive interrupt
   r .......1  rcve data reg full
   r ......1.  xmit data reg empty
   r .....1..  -data carrier detect (or has been down)
   r ....1...  -clear to send
   r ...1....  framing error
   r ..1.....  receiver overrun
   r .1......  parity error
   r 1.......  interrupt request
1  data
   r  receive
   w  transmit
```

Een tweede aspect waar we rekening mee moeten houden is dat de registers van de ACIA niet gecached mogen worden. Testen we namelijk op een bit in het Status Register dan kan dit na enige tijd veranderen. Als de CPU deze gegegevns in de cache heeft staan zijn ze dus niet meer consistent. (ook een snooping cache biedt hier geen uitkomst, bedenk zelf waarom). Een oplossing in C is om de bijbehorende variabelen door de modifier *volatile* vooraf te laten gaan.

```
struct acia{
char ctrlstat;
char data;
};

#define ACIAPTR 0xFF78E000

#define DIV1 0x00
#define DIV16 0x01
#define DIV64 0x02
#define RESET 0x03

#define RIE 0x80
#define RTSO_TIE 0x40
#define B8_NOPAR_STOP2 0x10

struct acia *aciaptr;

aciaptr = (struct acia*)ACIAPTR;
/* master reset */
aciaptr->ctrlstat = RESET;
/* initialise */
aciaptr->ctrlstat = RIE | RTSO_TIE | B8_NOPAR_STOP2 | DIV16;
```

Om te kijken hoe het met de adressen van de onderdelen van de struct zit gegeven we hier een klein testprogramma:

```
#include <stdio.h>

struct acia{
        char ctrlstat;
        char data;
};

int main(void) {
 struct acia ac;
 struct acia *aptr;
 aptr = &ac;
 printf("pointer to acia is %x\n", aptr);
 printf("pointer to ac.ctrlstat is %x\n", &ac.ctrlstat);
 printf("pointer to ac.data is %x\n", &ac.data);
}
```

Een alternatief voor de strct acia is een union, waarin we de leesbare en schrijfbare registers combineren.

```
union acia{
        struct {
```

```
            char ctrl;
            char data;
        }w;
        struct {
            char status;
            char data;
        }r;
};
```

Het volgende stukje C laat zien dat de aanduiding inderdaad klopt:

```
#include <stdio.h>

union acia{
        struct {
            char ctrl;
            char data;
        }w;
        struct {
            char status;
            char data;
        }r;
};

int main(void) {
 union acia ac;
 union acia *aptr;
 aptr = &ac;
 printf("pointer to acia is %x\n", aptr);
 printf("pointer to ac.w.ctrl is %x\n", &ac.w.ctrl);
 printf("pointer to ac.r.status is %x\n", &ac.r.status);
 printf("pointer to ac.w.data is %x\n", &ac.w.data);
 printf("pointer to ac.r.data is %x\n", &ac.r.data);
}
```

De output van dit stukje C is:

```
pointer to acia is bffff866
pointer to ac.w.ctrl is bffff866
pointer to ac.r.status is bffff866
pointer to ac.w.data is bffff867
pointer to ac.r.data is bffff867
```

Natuurlijk is het ook mogelijk een struct van unions te maken zoals in het volgende voorbeeld is te zien:

```
struct acia{
        union {
            char ctrl;
            char stat;
        }c;
        union {
            char data_out;
            char data_in;
        }d;
};
```

Een testprogramma laat zien dat we de juiste registers bereiken.

```
#include <stdio.h>

struct acia{
        union {
            char ctrl;
            char stat;
        }c;
        union {
            char data_out;
            char data_in;
        }d;
};

int main(void) {
 struct acia ac;
 struct acia *aptr;
 aptr = &ac;
 printf("pointer to acia is %x\n", aptr);
 printf("pointer to ac.c.ctrl is %x\n", &ac.c.ctrl);
 printf("pointer to ac.c.stat is %x\n", &ac.c.stat);
 printf("pointer to ac.d.data_out is %x\n", &ac.d.data_out);
 printf("pointer to ac.d.data_in is %x\n", &ac.d.data_in);
}
```

De output is:

```
pointer to acia is bffff866
pointer to ac.c.ctrl is bffff866
pointer to ac.c.stat is bffff866
pointer to ac.d.data_out is bffff867
pointer to ac.d.data_in is bffff867
```

Sommige p[rogrammeurs vinden de hexadecimale codes in de defines niet zo mooi en prefereren een andere mogelijkheid. Kijk maar naar het volgende stukje code:

```
#include <stdio.h>
#define bit(n) (1<<n)

#define VALUE bit(2)

int main(void){
 int val;
 val = bit(7);
 printf("zeven levert: %d en VALUE is %d\n", val, VALUE);
 val = bit(6) | bit(4) | bit(1);
 printf("samenstel van bits levert: %d\n", val);
 return 0;
}
```

De output van dit testprogramma waarin de bit-macro wordt gebruikt levert:

```
zeven levert: 128 en VALUE is 4
samenstel van bits levert: 82
```

Als laatste noemen we de mogelijkheid van bitfields. Hiermee kunnen we met variabelen van een of meer bits werken. Omdat compilers nogal slordig en onvoorspelbaar met bitfields omgaan, zou ik ze persoonlijk niet gebruiken. Bij het bestuderen van code voor devicedrivers kunnen ze echter voorkomen, vandaar dat ze hier genoemd worden.

```
struct ctrl_reg {
int low_3 : 3;
int m_1 : 1;
int high_4 : 4;
};

struct ctrl_reg myreg;

myreg.low = 6; /* write 110 to lower 3 bits */
myreg.high_4 = 0xa /* write 1010 to 4 bit bitfield */
```

Een probleem waar we rekening mee moeten houden is de zogenoemde allignment. In principe is een char 8 bits groot. Een struct van meer chars zal dan meestal goed afbeelden op een opeenvolging van bytes. Afhankelijk van de CPU en de compiler kunnen structs, die bijvoorbeeld een mengeling van chars, shorts en ints hebben, gaten bevatten omdat de compiler een short altijd op een even adres wil laten beginnen. De oplossing is om dan alleen met chars te werken en zo de short als higher en lower byte te beschrijven. Overigens speelt hier ook het little en big endian probleem mee. Als voorbeeld hier een struct met chars en een long:

```
#include <stdio.h>

struct acia{
        char ctrlstat;
        long bytecount;
        char data0;
        char data1;
};

int main(void) {
 struct acia ac;
 struct acia *aptr;
 aptr = &ac;
 printf("pointer to acia is %x\n", aptr);
 printf("pointer to acia.ctrlstat is %x\n", &ac.ctrlstat);
 printf("pointer to acia.bytecount is %x\n", &ac.bytecount);
 printf("pointer to acia.data0 is %x\n", &ac.data0);
 printf("pointer to acia.data1 is %x\n", &ac.data1);
}
```

Als output verschijnt op het scherm:

```
pointer to acia is bffff860
pointer to acia.ctrlstat is bffff860
pointer to acia.bytecount is bffff864
pointer to acia.data0 is bffff868
pointer to acia.data1 is bffff869
```

Aan deze output is te zien dat er tussen de eerste char en de long een gat van drie bytes is gevallen. Bij de twee chars data0 en data1 is er geen ruimte tussen de

twee struct members. Nu we de mapping van het device in memory space uitvoerig bekeken hebben, wordt het tijd om wat nader naar het aansturen van de chip te kijken. De initialisatie hebben we al gezien. Passen we programmed I/O toe dan moeten we in een lusje op invoer gaan wachten. Dit is in C als volgt te coderen:

```
#define FRAME_ERROR (1<<4)
#define OVERRUN_ERROR (1<<5)
#define PARITY_ERROR (1<<6)

while(!(aciaptr->status & RECEIVER_READY))
; /* wait for RECEIVER_READY bit to become TRUE
input = aciaptr->data;
error_status = (aciaptr->ctrlstatus & (FRAME_ERROR | PARITY_ERROR |
                OVERRUN_ERROR);
```

Let op: bij drivers gebruiken we meestal de bitwise AND of OR. Een driver op basis van interrupts gebruikt vrijwel dezelfe code. Alleen is nu de while lus niet nodig. Wel is het handig te testen of de acia ook echt verantwoordelijk is voor het interrupt signaal. Het ingelezen character kan in een inputbuffer geplaatst worden en het operating system kan een proces dat ingeslapen is op dit event wakker maken.

6.6 Opgaven

1. Beschrijf de taken van een operating system.
2. Wat zijn de vier taken van een device driver?
3. Wat verstaat men onder het endian probleem?
4. Geef een stukje C-code waarmee de waarde 0x56 naar geheugenplek 0x2F00 wordt geschreven.