Universiteit van Amsterdam

Samenvatting

Computer Organization and Design

20 september 2017

Studenten:
R. van den Berge
11889330

C. Bras
11917318

M. Höfelmann
11844299

R.A.J. Wacanno
11741163

Tutor:
W.R.J. Vermeulen

Practicumgroep:
A1 - ALGOL

Docent:
A. van Inge

Cursus:
Architectuur en Computerorganisatie

Vakcode:
5062ARCO6Y

Inhoudsopgave

1 Computerabstracties en -technologie

1.1 Principes Binnen de Computerarchitectuur

Bij het ontwerpen van computers is men in de loopt de jaren met bepaalde ideeën en principes gekomen welke van groot belang zijn geweest binnen het veld van de computerarchitectuur. Dit zijn veelal principes die bij het bestuderen van de theorie geregeld — het zij direct, het zij indirect — langs zullen komen. Hiervan zijn de volgende acht op dit moment het meest relevant.

De Wet van Moore

Deze 'wet', ooit geformuleerd door Gordon Moore, stelt dat iedere 18–24 maanden het aantal transistoren per IC (Integrated Circuit) verdubbelt.

Het Gebruik van Abstractie

Wanneer men praat over bepaalde onderdelen van een computer zijn niet alle details even relevant. Hierom wordt in deze context vaak abstractie toegepast.

De Versnelling van het Veel Voorkomende

Hoewel dit misschien voor de hand ligt is het ook bij computerontwerp van belang dat, wanneer een proces uit verschillende delen bestaat, het meeste rendement behaald kan worden op het gebied van prestaties wanneer de delen die met grotere frequentie voorkomen efficiënter worden gemaakt.

Parallelle Processen

In sommige gevallen kunnen processen die parallel draaien gebruikt worden om prestaties te verbeteren

Pijplijnen

Bij pijplijnen gaat het om een bijzondere vorm van parallelle processen. Wordt een proces zo geoptimaliseerd dat diens individuele onderdelen samen betere prestaties behalen dan wanneer ieder onderdeel op eigen houtje bezig is.

Voorspellen

Voor sommige processen is het, met het oog op prestaties, voordeliger bepaalde aannames te doen wanneer men weet dat deze dermate accuraat is en mits een mogelijke fout niet te zwaar zal wegen.

De Geheugenhiërarchie

Voor iedere computer is het van belang dat deze beschikt over mogelijkheid het een en ander op te slaan. Verschillende soorten opslag kunnen echter enorm uiteen lopen op het gebied van snelheid, capaciteit, maar ook kosten. Het is daarom dat binnen hardware een bepaalde hiërarchie bestaat met aan de bovenkant het snelste, maar tevens ook kleinste en duurste geheugen en onderaan de opslag met de grootste capaciteit en laagste kosten, maar ook de laagste snelheid.

Betrouwbaarheid d.m.v. Redundantie

Wanneer het van belang is dat een bepaalt systeem betrouwbaar is en dus onder vrijwel iedere omstandigheid blijft functioneren kunnen, an bepaalde onderdelen binnen het systeem, meerdere exemplaren worden geïnstalleerd om zo te garanderen dat het systeem bij een mogelijk defect naar behoren blijft werken.

1.2 Onderliggende Software

Om een programma succesvol op hardware te kunnen laten draaien is bepaalde software essentieel. Deze software vormt een tussenlaag tussen het programma en de hardware en maakt het op deze manier mogelijk dat deze twee correct met elkaar kunnen communiceren. Belangrijke voorbeelden van deze software zijn het besturingssysteem, de compiler en de assembler.

Het Besturingssysteem

Het besturingssysteem is onder andere verantwoordelijk voor: het omgaan met in- en uitvoer; het verdelen van opslag en geheugen onder de verschillende processen en mogelijk maken dat meerderen programma's tegelijkertijd en op gecontroleerde op een systeem draaien.

De Compiler

De compiler is onderdeel van een keten die high-level programmatuur omzet om uiteindelijk door de hardware gebruikt te kunnen worden. Om precies te zijn zet de compiler talen (als bijvoorbeeld C en Java) om in zogenaamde assembleertalen.

De Assembler

De assembler maakt de door de compiler gestarte keten af door de reeds genoemde assembleertalen om te zetten in een reeks bits (nullen en enen). Deze bits kunnen vervolgens voor de hardware worden ingelezen en verwerkt.

1.3 Onderliggende Hardware

Om het mogelijk te maken de uitvoer hiervoor genoemde software mogelijk te maken zijn bepaalde hardwarecomponenten binnen een systeem vereist.

In- en Uitvoer

Onder invoer verstaat men hetgeen gegevens in het geheugen schrijft en onder uitvoer hetgeen gegevens uit het geheugen leest.

De Processor

De processor wordt vaak gezien als het hart van een computer; het stelt de computer in staat gegevens van buiten te verwerken om zo nieuwe gegevens aan de gebruiker te presenteren. Deze gegevens haalt de processor uit het geheugen met behulp van **datapaden**. Deze datapaden stellen de processor tevens in staat gegevens
Binnen de processor zit ook een **aansturingseenheid**. Deze stuurt de het datapad, het geheugen en de in- en uitvoer aan op basis van programma-instructies.

Geheugen

Het geheugen onder het snellere deel van de geheugenhiërarchie en staat dichter bij de processor om deze zo van de voor het programma vereiste data te voorzien. Het geheugen kan nog weer worden opgedeeld in het **cache-** en het **werkgeheugen**. Het werkgeheugen bevat de data van het draaiende programma en het cachegeheugen fungeert als een buffer tussen de processor en het werkgeheugen.

Opslag

Opslag beslaat het onderste en dus langzaamste maar ook goedkoopste deel van de geheugenhiërarchie. Bij opslag kan men denken aan harde schijven maar ook het zogenaamde flash geheugen dat in usb-sticks en SSD's zit. Opslag is ideaal voor langetermijnsopslag en voor het bewaren van data die sporadisch door een programma wordt opgevraagd.

Instructieset Architectuur

De instructieset architectuur bepaald welke instructies een stuk assembleertaal kan bevatten wanneer het op een bepaalde processor draait.

1.4 Streven naar Prestaties

Bij het bouwen van computersystemen wordt altijd getracht om zo goed mogelijke prestaties te realiseren. Om dit te kunnen doen moet men in staat zijn om in te zien wat nou eigenlijk goede prestaties zijn; hoe prestaties gemeten kunnen worden en hoe prestaties daadwerkelijk kunnen worden verbeterd.

Prestaties Vaststellen

Bij een computer zijn er twee parameters die als graadmeter voor diens prestaties kunnen dienen met elk zijn eigen toepassing.
Uitvoeringstijd is vooral voor de individuele computergebruiker van belang en kan gedefinieerd worden als de hoeveelheid tijd die het kost on een programma in zijn volledigheid te laten draaien. Twee systemen kunnen op deze manier vergeleken worden als men stelt dat:

$$Prestaties_X = \frac{1}{Uitvoeringstijd_X}$$

Dan volgt het volgende:

$$Prestaties_X > Prestaties_Y$$

$$\frac{1}{Uitvoeringstijd_X} > \frac{1}{Uitvoeringstijd_Y}$$

$$Uitvoeringstijd_Y > Uitvoeringstijd_X$$

Bandbreedte is voor datacentrum administratoren van belang en heeft als betekenis de hoeveelheid die per tijdseenheid verwerkt kan worden.

Prestaties Meten

Voor het meten van prestaties kan de uitvoeringstijd van de processor worden gemeten; ofwel de tijd die de processor er over doet een bepaalde taak uit te voeren. Deze kan zowel in seconden als cycli van de processor worden uitgedrukt.

Prestaties van de Processor

De prestaties van een processor kunnen door een tal van factoren worden beïnvloed. Een simpele formule die dit weergeeft is de volgende:

$$Uitvoeringstijd = Processorcycli \cdot Tijd/cyclus$$

Omdat de tijd per cyclus de inverse van de klokfrequentie is kan men ook stellen dat:

$$Uitvoeringstijd = \frac{Processorcycli}{Klokfrequentie}$$

Prestaties m.b.t. Instructies

De prestaties hangen ook nauw samen met de aard van de instructies. Zo is het volgende waar:

$$Klokcycli = Instructies \cdot cycli/instructie$$

De term CPI (Cycles Per Instruction) wordt gedefinieerd als de gemiddelde hoeveelheid cycli die nodig is voor het uitvoeren van een instructie. Met de CPI kan ook de volgende vergelijking worden opgesteld:

$$Uitvoeringstijd = \frac{Instructies \cdot CPI}{Klokfrequentie}$$

Uiteindelijk kan met dit alles een algehele vergelijking worden opgesteld die er als volgt uit ziet:

$$Tijd = Seconden/Programma = \frac{Instructies}{Programma} \cdot \frac{Klokcycli}{Instructie} \cdot \frac{Seconden}{Klokcyclus}$$

De Onontkoombare Stroombarrière

Met het najagen van steeds betere prestaties is men gestuit op onontkoombare eigenschappen van elektriciteit en energie in het algemeen. Wanneer men de klokfrequentie van een processor tracht te verhogen zal deze ook steeds meet stroom verbruiken. Dit heeft tot gevolg dat de processor tijdens gebruik steeds warmer zal worden waardoor beter koeling vereist is. Teven dient, vooral in het geval van bijvoorbeeld draagbare apparaten de toevoer van stroom afdoende te zijn om dergelijke processoren te laten functioneren op hun gewenste snelheid.

De Vermeerdering van Processorkernen

In de meer recente geschiedenis heeft men voor het verbeteren van de prestaties van een systeem zijn heil gezocht in het gebruiken van meerdere processorkernen binnen een processor. Deze ontwikkeling stamt uit het paralleliseren van processen, daar meerdere processorkernen het mogelijk maken meer gegevens tegelijkertijd te verwerken.

1.5 Problematiek op het Gebied van Prestatieverbetering

De Wet van Amdahl

Zie **De Versnelling van het Veel Voorkomende** in combinatie met de onderstaande vergelijking.

$$Uitvoeringstijd_{na} = \frac{Uitvoeringstijd_{beïnvloed}}{Verbeteringshoeveelheid} + Uitvoeringstijd_{onbeïnvloed}$$

2

3

4 De Processor

4.1 Het Pijplijn-principe in een notendop

Het **pijplijnen** is een veelgebruikte manier om je architectuur instructies te laten uitvoeren.

Om te illustreren waarop dit pijplijnen berust kan men een viertal handelingen nemen: A, B, C en D welke na elkander en tezamen herhaaldelijk uitgevoerd moeten worden. Zonder er niet al teveel bij na te denken zou men de vier handelingen steeds na elkaar uit kunnen voeren (als ABCDABCD enz.). Dit is echter geen efficiënte strategie als de combinatie van handelingen vier of vijf keer uitgevoerd moet worden aangezien de tijd die dit zal kosten gelijk zal zijn aan $n \cdot (A + B + C + D)$; ofwel, de som van de tijd die iedere individuele handeling kost maal het aantal keer dat het herhaald moet worden.

Om deze herhaling van stappen in kortere tijd te laten verlopen kun je het principe van pijplijnen toepassen. Als de als voorbeeld genomen proces ABCD op deze manier wordt uitgevoerd zal dit er — horizontaal afgezet tegen de tijd — zo uit zien:

```
A  B  C  D
   A  B  C  D
      A  B  C  D
```

En zo zou deze serie handelingen er uitzien als het niet gepijplijnd was:

```
A  B  C  D  A  B  C  D  A  B  C  D
```

Als men deze twee diagrammen vergelijkt zijn er twee dingen die opvallen: wanneer de gepijplijnde aanpak benut wordt, is de totale uitvoeringsduur drastisch verminderd en bij het gepijplijnd uitvoeren lopen de individuele ABCD-processen parallel aan elkaar. In dit laatste schuilt dan ook de kracht van het pijplijnen. Door nadat een stap, zoals bijvoorbeeld A, voor de eerst set handelingen is uitgevoerd zal deze onmiddellijk herhaald worden voor de volgende en zo gebeurd dat ook met de andere stappen. Op deze manier wordt op ieder moment toch maar een van de stappen tegelijkertijd uitgevoerd, maar kan toch een significante versnelling gerealiseerd worden door het parallelliseren van de processen.

Onder perfecte omstandigheden zou de versnelling van de uitvoeringstijd van een collectie herhaalde processen als gevolg van pijplijnen gelijk zijn aan het aantal losse stappen waaruit ieder proces bestaat. In dat geval zou de volgende vergelijking kunnen worden opgesteld:

$$Tijd\ tussen\ instructies_{gepijplijnd} = \frac{Tijd\ tussen\ instructies_{niet\ gepijplijnd}}{Hoeveelheid\ pijplijn\text{-}stappen}$$

Met instructies worden hierboven processen als die van ABCD bedoeld

Er is echter vrijwel nooit sprake van perfecte omstandigheden. Het niet evenveel tijd kosten van individuele stappen en het feit dat er aan het begin en einde van iedere uitvoer minder instructies parallel kunnen lopen maakt dat de theoretisch beloofde versnelling niet haalbaar is. Een illustratie hiervan volgt alsmede de toepassing van pijplijnen binnen de MIPS-architectuur.

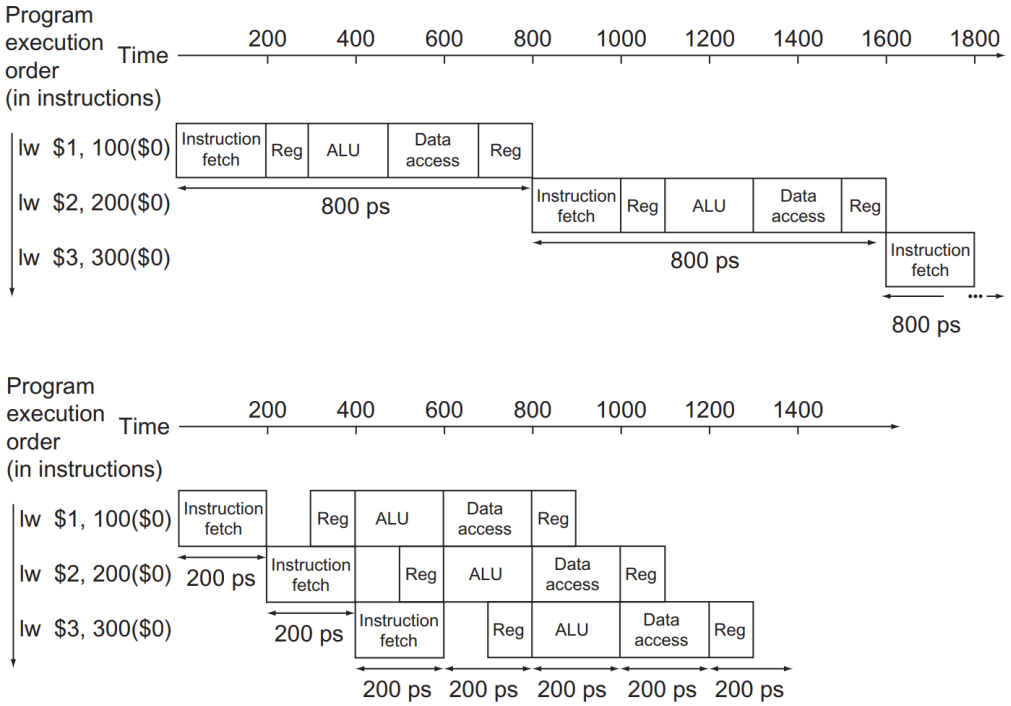
Zoals stappen A, B, C en D zijn er ook binnen een processor bepaalde handelingen die herhaaldelijk worden uitgevoerd. Dit zijn over het algemeen de volgende vijf stappen:

1. Het uit het geheugen halen van een instructie. *(Instruction fetch)*
2. Het lezen van registers terwijl de instructie gedecodeerd wordt; hetgeen bij MIPS-instructie van met de reguliere indeling gelijktijdig kan plaatsvinden. *(Register read)*
3. Het uitvoeren van de operatie of berekenen van een geheugenadres. *(ALU operation)*
4. Het benaderen van een operand in het data-geheugen. *(Data access)*
5. Het schrijven van de resultaten in een register. *(Register write)*

Als we nu ter illustratie de `lw`-instructie nemen heeft deze per stap de volgende hoeveelheid tijd nodig:

Instruction	*Instruction fetch*	*Register read*	*ALU operation*	*Data access*	*Register write*	*Total time*
Load word (`lw`)	200 ps	100 ps	200 ps	200 ps	100 ps	800 ps

In de volgende twee figuren zijn zowel de gepijplijnde als de niet gepijplijnde herhaalde uitvoer van `lw` uitgezet tegen de tijd:



Door middel van pijplijnen kan de totale uitvoeringstijd van het bovenstaande voorbeeld van 2400 ps naar 1400 ps worden teruggebracht. Hoewel dit natuurlijk een significante verbetering is, is het niet in overeenstemming met de eerdergenoemde aanname dat de reductie in tijd gelijk zou staan aan het aantal individuele stappen.

De discrepantie tussen de theoretische en daadwerkelijke tijd zijn toe te schrijven aan de eerdergenoemde problemen op het gebied van pijplijnen. Allereerst zijn niet alle stappen even lang in duur. Wegens de aard van het pijplijnen is het cruciaal dat er bij het uitvoeren van stappen een bepaalde ritmiek zodat zodat deze steeds op gelijke interval worden uitgevoerd. Indien er dus een stap is die eerder voltooid is dan anderen moet er een korte rust ingevoegd worden om zo de ritmiek te handhaven. Als tweede valt de tijdswinst tegen als gevolg van het geringe aantal herhalingen. Een groter aantal herhalingen zou de verspilling van tijd die plaatsvindt aan het begin en einde van de uitvoer namelijk teniet doen.

Hier is een vergelijking van versnelling tussen — links — de 3 reeds beken herhalingen en — rechts — een hypothetisch aantal herhalingen van 1.000.000 inclusief de eerdere 3:

$$\frac{2.400\ ps}{1.400\ ps} = 1.71 \qquad \frac{800.002.400\ ps}{200.001.400\ ps} = 4.00$$

4.1.1 Instructiesets voor Pijplijnen

In het geval van MIPS, een instructieset expliciet bedoeld voor pijplijnen, zijn op vier fronten keuzes gemaakt die het gebruik voor pijplijnen enorm bevorderen. Allereerst zijn instructies gelijk in lengte. Ten tweede is er een gelimiteerd aantal instructie-indelingen. Ten derde komen geheugen-operanden enkel voor in *load*- en *store*-instructies. Ten vierde speelt ook de uitlijning van operanden in het geheugen hierbij een rol.

4.2 Pijplijnen m.b.t. het Datapad en de Aansturingseenheid