# Inhoud

# Hoofdstuk 1: Wat is CSA?

*Een computer system architect stelt computers op maat samen aangezien niet alle out-of-the-box-computers[1] geschikt zijn (bvb: gamecomputers, bedrijfssystemen, …). In computersystemen maken we onderscheid tussen hardware (met een specifiek gebruikerspubliek) en software (met een bepaald gebruikerdoel).*

## Dingen waarop een CSA moet letten

- Rekenkracht (processor)
- Grafische weergave
- Batterijcapaciteit en elektriciteitsverbruik
- Afmetingen, gewicht, hitteproductie, koelvereisten

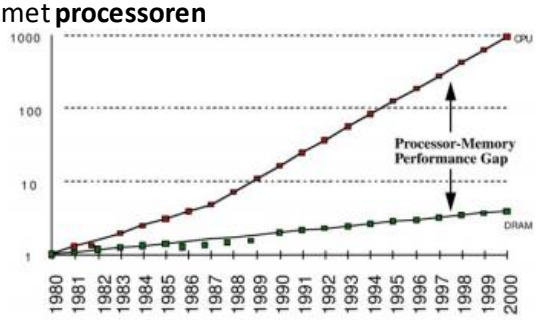
➔ Hangt af van de vereisten van de klant

## Wat gebruikt een CSA

- Nieuwe producten en evolutie (bvb: blogs lezen, reviews van nieuwe hardware lezen)
- Wetmatigheden

Moore's law:
- De **processorsnelheid verdubbelt** elke **twee** jaar (min of meer)
- De **geheugens verdubbelen** hun **capaciteit** ongeveer elke **twee** jaar

Memory wall: (hangt samen met Moore's law)
- **Geheugen latency** (tijd tussen begin en einde van een actie) stijgt maar met **1,1** keer per twee jaar
- **Onevenwicht** met **processoren**




➔ Oplossing: buffer[2]

[1] Out-of-the-box is een term voor producten die geen tot weinig installatie behoeven. Het product is direct gebruiksklaar zodra het uit de doos gehaald wordt.

2 Een buffer is een deel van het computergeheugen waarin gegevens geplaatst kunnen worden om later uitgelezen te worden. Een buffer vormt een wachtrij die asynchrone communicatie mogelijk maakt en wordt ingezet om het schrijven en lezen van gegevens niet tegelijk plaats te laten vinden.

## Snelheid versus energiegebruik

- Als **snelheid stijgt**, **daalt** de **kostprijs** van **oudere** chips (bruikbaar in embedded)
- Als **snelheid stijgt**, blijft het **stroomverbruik** toch **gelijk**
- Als **snelheid constant** blijft dan **halveert** het **stroomverbruik**

➔ CSA kiest voor snelheid of kostprijs

## Active power

Op een processor staat vaak het maximum stroomverbruik geschreven maar dit is niet interessant voor een CSA aangezien deze meer heeft aan het stroomverbruik terwijl de processor gewoon werkt (= active power).

$$P = 1/2 * C * V^2 * f * \alpha$$

| Symbool | Omschrijving |
|---|---|
| P | Active power |
| C | Capacitance of het vermogen tot opslaan van een elektrische lading (bvb: condensator) |
| V | Voltage |
| f | Frequentie |
| α | Activiteitsfactor (bvb: hoeveel werkt de processor) |

## Fabricagekost

Chips produceren kost geld en dit zal dus de prijs beïnvloeden. Zo bepaalt de grootte van chips mee de kostprijs (grote chips zijn duurder dan kleine chips).

- Silicium (onzuiverheden) en laagfouten zijn mogelijk
- Bij productieproces is er uitval (verhoogt eindprijs van de chips)
- Grootte chips bepaalt mee de kostprijs (grote chips zijn duurder door onzuiverheden en drukfouten op de schijf)

## Voorbeeldvragen

**Vraag 1.1**

*Een CSA vindt het wiel niet opnieuw uit. Hij maakt vaak een keuze uit bestaande hardware en software. Waarmee houdt hij rekening bij die keuze voor de hardware en de software? Verklaar in je eigen woorden.*

Een CSA houdt rekening met de eisen van de gebruiker en waarvoor het systeem moet worden gebruikt. Het doel van een CSA is een zo performant mogelijk systeem te maken voor een bepaald doel, binnen het budget van de klant.

**Vraag 1.2**

*Waarom zijn out-of-the-box-computersystemen niet geschikt voor de meeste computersysteemarchitecten? Leg uit aan de hand van een fictief voorbeeld.*

Een CSA maakt een computersysteem voor een bepaald doel. Een out-of-the-box-computersysteem is gemaakt voor een breed publiek en heeft dus bredere functies. Zo'n computersysteem gaat zich niet toespitsen op specifieke doelen. Daarnaast is het een voordeel om een systeem zelf te laten bouwen omdat je dan exact kan zeggen wat je nodig hebt en wat niet. Zo heb je als digital artist een heel goed scherm nodig met een goede kleurresolutie maar hoeft je CPU niet de beste CPU te zijn (een i5 volstaat).

**Vraag 1.3**

*Waarom vormt het elektriciteitsverbruik een belangrijk issue bij het ontwikkelen van een computersysteem? Verklaar aan de hand van een voorbeeld.*

De issue bij een vast toestel (dat je constant aan het elektriciteitsnet hangt) is dat dit kostmatig duurder kan zijn. Bij mobiele apparaten is de issue dat deze zo min mogelijk stroom mogen verbruiken, de batterij niet te groot mag zijn en dat de snelheid van opladen voldoende snel moet zijn.

**Vraag 1.4**

*Geef Moore's law zoals die vandaag toegepast wordt.*

Eerst stelde Moore's law dat de snelheid van processoren en de capaciteit van het geheugen elk jaar verdubbelt. Echter, dit is niet meer waar omdat we de transistoren niet meer zo veel kunnen verkleinen. Daardoor spreken we nu van een verdubbeling om de twee jaar.

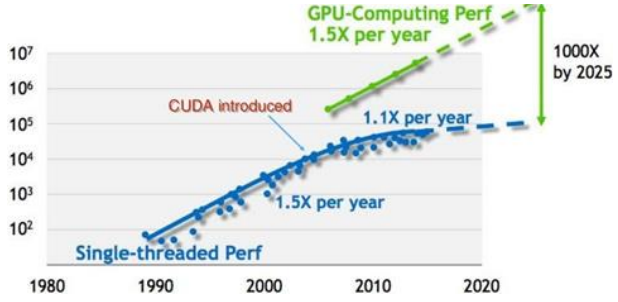
**Vraag 1.5**

*Leg in het eigen woorden het concept van de memory wall uit.*

Hoewel de processoren en de capaciteit van het geheugen elke twee jaar verdubbelen neemt de latency van het geheugen maar met 1,1 toe per twee jaar. Hierdoor ontstaat er een kloof tussen de snelheid van de processor en de snelheid van het geheugen. De CPU zal dus moeten wachten op het geheugen. Dit kan tijdelijk opgelost worden door cache en buffers.

**Vraag 1.6**

*Welk verschijnsel wordt in het onderstaande diagram aangetoond? Geef de naam van die wet en leg hem in eigen woorden uit.*




Dit is de memory wall. Zie vraag 1.5 voor de beschrijving van wat dit is.

**Vraag 1.7**

*Leg in eigen woorden het verband tussen snelheid van een microprocessor en het bijbehorende stroomverbruik uit.*

CPU's worden sneller gemaakt door meer transistoren te plaatsen en dit zou dus voor meer stroomverbruik zorgen maar doordat de transistoren kleiner worden, wordt dit uitgebalanceerd waardoor het stroomverbruik constant blijft over de verschillende generaties van CPU's.

**Vraag 1.8**

*Leg in eigen woorden uit hoe de CSA voordeel kan halen uit de snelheidstoename bij de ontwikkeling van processoren.*

Je kan op twee verschillende manieren daar voordeel uit halen:

- De snelheid verdubbelt elke twee jaar. Zo kan het systeem elk jaar sneller worden. Hierdoor blijft de prijs van het systeem constant.
- De prijs voor dezelfde performance neemt elk jaar af. Hierdoor kan het systeem elk jaar goedkoper worden voor een vaste performance.

Je moet kiezen uit deze twee want je kan ze niet beiden hanteren.

**Vraag 1.9**

*Hoe komt het dat grote chips duurder zijn dan kleinere chips? Verklaar aan de hand van een eenvoudige tekening en licht in eigen woorden toe.*

Elke chip wordt op een wafer (silicium schijf) geprint. De prijs voor één wafer blijft nagenoeg constant. Hoe kleiner de chips, hoe meer chips er op een wafer kunnen. In deze wafer zitten ook onzuiverheden. Hoe kleiner de chip, hoe kleiner de kans dat een chip een onzuiverheid bevat. Alle mislukte chips tellen ook mee voor de verkoopprijs en bij een grote chip zijn dit er nu eenmaal meer.

**Vraag 1.10**

*Noteer de formule waarmee je het actieve stroomverbruik van een CPU berekent.*

$P = 1/2 * C * V^2 * f * \alpha$

**Vraag 1.11**

*Wat is actief stroomverbruik van een CPU? Leg in je eigen woorden uit.*

Actief stroomverbruik is de hoeveelheid stroom die de CPU gebruikt wanneer deze effectief aan het werken is.

**Vraag 1.12**

*Welke twee grote kostenverhogende problemen kunnen er bij de productie van microchips ontstaan? Som ze op.*

1. Onzuiverheden in het silicium
2. Productiefouten bij het printen van transistoren op de wafer (= laagfouten)

**Vraag 1.13**

*Wat bereken je precies met onderstaande formule? Of met andere woorden, waarvoor staat P (exacte benaming)?*

*$P = ½ * C * V^2 * f * alfa$*

P is het actief vermogen. Dit is het vermogen wanneer de CPU in gebruik is.

**Vraag 1.14**

*Wanneer je in de specificaties van een CPU kijkt, staat er vaak een stroomverbruik vermeld. Over wat voor soort stroomverbruik gaat het dan? Geef de naam van dat stroomverbruik.*

Dit is het maximale stroomverbruik bij peak performance.

**Vraag 1.15**

*Som twee manieren op waarmee een CSA op de hoogte kan blijven van de nieuwste producten en evoluties.*

1. Het lezen van artikels op betrouwbare blogs
2. Het lezen van reviews over nieuwe hardware

# Hoofdstuk 2: Performance

*Performance meten is belangrijk omdat een architect moet weten of de hardware prestaties voldoende afgestemd zijn op de klantvereisten en omdat een CSA toekomstgericht moet werken. De twee belangrijkste begrippen rond performance zijn latency en throughput.*

## Latency

Latency is de tijd tussen het begin van een programmaproces en het einde ervan (niet over de I/O-routines). Synoniemen voor latency zijn: execution time, response time, CPU time. Latency is afhankelijk van assembly instructions en de frequentie en het gebruik ervan.

## Throughput

Dit is het aantal acties die de processor per seconde kan verwerken (in MIPS[3]). Synoniemen van throughput zijn performance level en bandwidth (in GB/s). Desondanks, een CPU met de hoogste throughput is niet noodzakelijk de beste. Dit is afhankelijk van de applicaties die de klant wilt gebruiken.

## Formule

$$throughput = \frac{1}{latency} \qquad CPU\ performance = \frac{1}{execution\ time}$$

MAAR: bijkomende factoren:

- Clock rate: de snelheid in Hz (1 Hz = 1 actie per seconde)
- Clock cycle: de tijd tussen twee pulsen van de oscillator (in s)

[3] Million Instructions Per Seconds

➔ Aangepaste formule:

$$clock\ cycle\ time\ = \frac{1}{clock\ rate}$$

## Speedup en benchmarking

Wanneer een CSA een prototype voor de klant samenstelt moet de performance gemeten worden. Zo ga je proberen te ontdekken wat de speedup[4] is. Er zijn drie manieren om de performance te meten:

- Gebruiker zelf laten testen (onbetrouwbaar)
- Zelf testen met data en app klant (onbetrouwbaar)
- Benchmarking (betrouwbaar)

Benchmarking is een reeks van programma's rond een bepaald thema en die gaan dan objectieve resultaten geven rond de performance van de computers. Om prototypes te vergelijken moet je dan verschillende prototypes met dezelfde benchmark testen. De benchmarks worden opgesteld door de industrie.

## Iron law of performance

Deze wet geeft aan waarvan de CPU-tijd afhankelijk is. Deze is afhankelijk van het **aantal gestelde instructies, het aantal cycli per instructie en het aantal seconden per cyclus.** De basiswet stelt dat de verwerking altijd op dezelfde manier gebeurt: er wordt evenveel tijd gespe ndeerd. De dag van vandaag zijn er ook processoren met ongelijke cycli.

De iron law voor gelijke instruction times:

$$CPU\ time\ = \frac{sec}{per\ program} \qquad CPU\ time\ = \frac{number\ of\ instructions}{per\ program} \times \frac{cycles}{per\ instruction} \times \frac{sec}{per\ cycle}$$

De iron law voor ongelijke instruction times:

$$CPU\ time\ = \sum_i instructions\ counted_i \times cycles\ per\ instruction_i\ \times \frac{sec}{per\ cycle}$$

$\sum_i$ = de som van de getelde instructies en i = de huidige instruction time

## Amdahl's law

Deze law heb je nodig wanneer je een speedup doet bij slechts delen van een programma en/ of slechts enkele instructies.

$$speedup\ = \frac{1}{(1-\ frac_{enh}) + \frac{frac_{enh}}{speedup_{enh}}}$$

[4] De speedup is hoeveel de ene configuratie sneller is dan de andere configuratie.

## Voorbeeldvragen

**Vraag 2.1**

*Waarom moet een systeemarchitect de performance van een computersysteem kunnen meten?*

Performance meten is belangrijk omdat een architect moet weten of de hardware prestaties voldoende afgestemd zijn op de klantvereisten en omdat een CSA toekomstgericht moet werken.

**Vraag 2.2**

*Leg in eigen woorden het begrip latency bij CPU's uit.*

Latency is de tijd tussen het begin van een programmaproces en het einde ervan. Synoniemen voor latency zijn: execution time, response time, CPU time.

**Vraag 2.3**

*Leg in eigen woorden het begrip throughput bij CPU's uit.*

Dit is het aantal acties die de processor per seconde kan verwerken. Synoniemen van throughput zijn performance level en bandwidth.

**Vraag 2.4**

*Leg aan de hand van een formule het verband tussen throughput en latency uit. In de formule zelf mag je de begrippen latency en throughput niet gebruiken, wel het begrip performance.*

$$CPU\ performance = \frac{1}{execution\ time}$$

Throughput is hetzelfde als CPU performance en latency is hetzelfde als de execution time.

**Vraag 2.5**

*Wat is het verschil tussen de clock cycle time en de clock rate? Verklaar in je eigen woorden.*

De clock cycle is de tijd tussen twee pulsen van de oscillator (in seconden). De clock rate is de snelheid waarmee de pulsen komen (aantal pulsen in één seconde in Hz).

**Vraag 2.6**

*Som twee foutieve manieren van het testen van prototypes op én één goede manier.*

Fout:

- Gebruiker laten testen
- Zelf testen met de data en app van de klant

Juist:

- Benchmarking met specifieke testprogramma's

**Vraag 2.7**

*Verklaar in eigen woorden het begrip speedup.*

De toename (of afname) van de performance van het ene systeem ten opzichte van een ander systeem.

- Speedup > 1: toename
- Speedup < 1: afname

**Vraag 2.8**

*Schets in eigen woorden wat de Iron law algemeen precies inhoudt.*

De Iron Law of performance geeft aan waarvan de CPU-tijd afhankelijk is. Deze is afhankelijk van **het aantal gestelde instructies, het aantal cycli per instructie en het aantal seconden per cyclus**. Een computer system architect moet hiermee rekening houden bij het maken van een systeem. Voor een bepaald systeem kan de focus naar één van deze gaan (of meerdere).

**Vraag 2.9**

*Leg in eigen woorden het verschil tussen de Iron law en Amdahl's law uit.*

De Iron law geef aan welke aspecten kunnen worden gewijzigd om de performance te veranderen. Amdahl's law geeft aan wat dan de speedup is van die verandering.

**Vraag 2.10**

*Met welke wiskundige bewerking moet je de speed up van een prototype correct berekenen? Geef de naam van die bewerking.*

Geometric mean = $\left(\prod_{i=1}^{n} x_i\right)^{\frac{1}{n}} = \sqrt[n]{x_1 x_2 \cdots x_n}$

**Vraag 2.11**

*Wanneer meet je throughput in MIPS en wanneer in GB/s?*

Wanneer je over het over het aantal instructies hebt, gebruik je MIPS. Als je gebruik maakt van het synoniem bandwidth, gebruik je GB/s.

**Vraag 2.12**

*Wanneer een computersysteem architect twee microprocessoren voor een project selecteert, dan is de processor met de hoogste throughput niet altijd de beste keuze. Waarom?*

Dit is afhankelijk van de applicaties die de klant wilt gebruiken.

**Vraag 2.13**

*Hoe kunnen assembly instructies en de gebruiksfrequentie van assembly instructions de latency beïnvloeden? Leg in je eigen woorden uit.*

Sommige instructies zorgen voor een stall in de processor. Hierdoor zal de latency toenemen.

**Vraag 2.14**

*Wat is een benchmark? Verklaar en licht de toepassing ervan toe.*

Benchmarking is een reeks van programma's rond een bepaald thema en die gaan dan objectieve resultaten geven rond de performance van de computers. Om prototypes te vergelijken moet je dan verschillende prototypes met dezelfde benchmark testen. De benchmarks worden opgesteld door de industrie.

**Vraag 2.15**

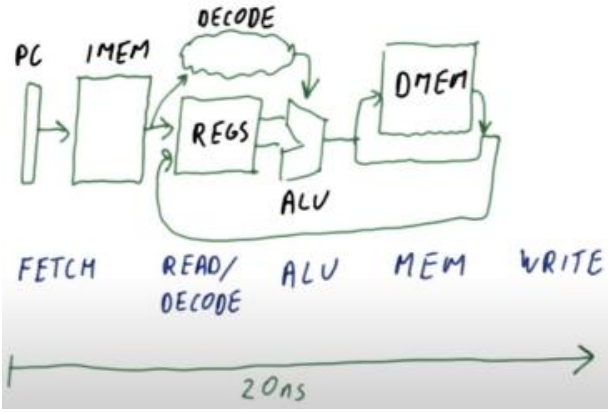
*Hoe ontstaan ijkpunten in benchmarking? Leg met je eigen woorden uit.*

De ijkpunten zijn de huidige standaards waarmee vergeleken wordt. Dit zijn benchmarks die door de producenten zelf worden uitgevoerd volgens een bepaald voorschrift.

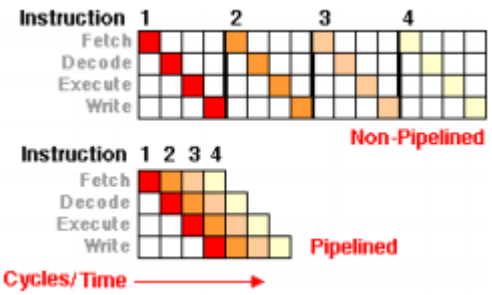
# Hoofdstuk 3: Pipelining

*Pipelining is het optimaal inzetten van CPU-hardware-componenten om de performance te verhogen. Bij pipelining voert de CPU al een deel van de volgende instructie uit terwijl de vorige instructie nog niet afgewerkt is.*

## Flow




## Pipelining




Op deze manier duurt het eventjes voor de pipeline gevuld is maar daarna komen de instructies er één na één uit.

## Speedup berekenen

Single cycle: bijvoorbeeld latency van 40 ns (nanoseconden)
➔ Eén stap = 40 / 5 = 8 ns
➔ Troughput = 1 instructie / 8 ns

## Snelheid pipeline

De snelheid van een pipeline wordt uitgedrukt in cycles per instruction (CPI). Als we niet zouden werken met pipelines dan hebben we standaard 1 CPI maar dankzij een pipeline ligt dat meestal hoger dan één. Een probleem dat zich voordoet zijn stalls (we raken niet meer aan 1 CPI). Dit zijn cyclussen in de pipeline zonder input. Stalls worden veroorzaakt door:

- Structural dependencies
- Control dependencies
- Data dependencies

### Structural dependencies

Dit zijn resource-conflicten in de pipeline. Een voorbeeld hiervan zijn twee verschillende instructies die tegelijkertijd iets op hetzelfde register proberen te doen (load en store).

Een oplossing hiervoor is het geheugen opsplitsen in twee onafhankelijke delen voor instructies (code memory) en storage (data memory)[5].

### Control dependencies

Hierbij maken we een sprong in het geheugen (BRANCH[6], CALL[7], JMP[8], …) maar het doeladres ontbreekt.

Een oplossing hiervoor is branch prediction (= op voorhand voorspellen wat de uitkomst gaat zijn).

### Data dependencies

Dit komt voor wanneer een instructie data gebruikt van een vorige instructie.

Een oplossing hiervoor is operand forwarding waarbij tussentijdse resultaten bijgehouden worden.

## Regels

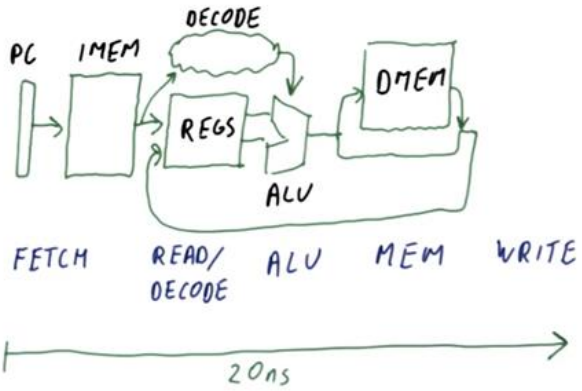
- Backward travelling instructies – in welke fase dan ook – zijn gevaarlijk
- Forward traveling instructies worden vergrendeld op elke fase
- Logica wordt in éénzelfde fase toegepast

[5] Dit is renaming.
[6] Condition
[7] Andere informatie nemen
[8] Naar een label gaan

## Voorbeeldvragen

### Vraag 3.1

*Teken een basisschets van een pipeline van een CPU. Voeg labels op je schets toe.*




### Vraag 3.2

*Verklaar onderstaande schets. Waarover gaat die schets in grote lijnen?*

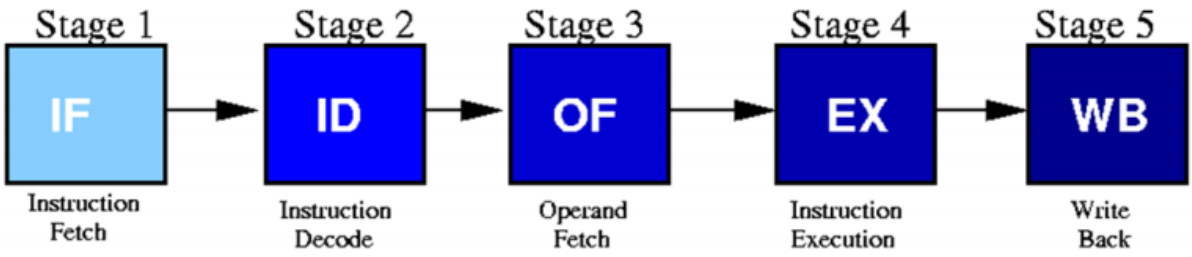
| Instr No. | Pipeline Stage | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | IF | ID | EX | MEM | WB | | |
| 2 | | IF | ID | EX | MEM | WB | |
| 3 | | | IF | ID | EX | MEM | WB |
| 4 | | | | IF | ID | EX | MEM |
| 5 | | | | | IF | ID | EX |
| Clock Cycle | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

Deze figuur geeft de pipeline van een CPU weer (IF, ID, EX, MEM, WB). De groene lijn is de huidige cyclus. Bij elke cyclus zal de groene strook naar rechts opschuiven.

Instructie 1 zit in de pipeline in de MEM stage. Instructie 2 bevindt zich iets minder ver in de pipeline en zal bezig zijn met de EX stage. Instructie 5 bevindt zich nog niet in de pipeline.

### Vraag 3.3

*Teken de flow van instructies in een CPU op een eenvoudige manier en plaats labels.*




### Vraag 3.4

*Som de vijf stadia op die instructies binnen de CPU doorlopen. Je hoeft de stadia daarbij niet uit te leggen.*

- Instructies halen (F)
- Registers uitlezen en decoderen (D)
- ALU gebruiken (A)
- Memory gebruiken (M)
- Wegschrijven naar registers (W)

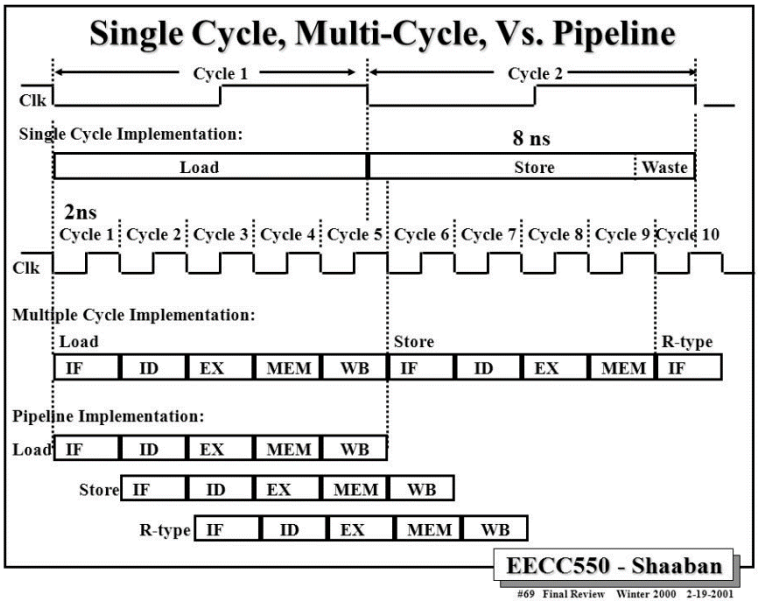
**Vraag 3.5**

*Hoe komt het dat instructieverwerking door een single cycle microprocessor trager is dan een microprocessor met pipelining, terwijl een microprocessor met pipelining steeds een trage start neemt en een single cycle microprocessor er direct invliegt?*

Omdat bij een single cycle microprocessor er maar één instructie wordt gedaan per klok cyclus. Hierdoor zal het een aantal cycli duren voordat de gehele actie is uigevoerd. Bij een microprocessor met pipeline zal het enkele cycli duren voordat de pipeline is gevuld, maar vanaf dan zal er elke cyclus een actie worden voltooid.

**Vraag 3.6**

*Bereken de throughput van een microprocessor single cycle die 40 ns per cycle nodig heeft, waarbij alle onderdelen dezelfde tijd nemen.*




40 ns (is tijd) / 5 (aantal fases in pipeline) = 8 ns (tijd voor 1 instructie)

**Vraag 3.7**

*Met welke eenheid druk je pipelining in een CPU uit? Schrijf zowel de afkorting als de benaming voluit.*

CPI: cycles per instruction

**Vraag 3.8**

*Verklaar in eigen woorden het begrip stall in pipelining van een CPU.*

De vertraging in uitvoering van een instructie zodat een “error” kan opgelost worden ( dit kan iets banaal zijn zoals wanneer de instructie voor u nog het geheugen aan het gebruiken is en je moet even wachten want het duurde langer dan gedacht)

**Vraag 3.9**

*Wat kan een stall in een CPU-pipeline veroorzaken? Licht in eigen woorden toe.*

- Structural dependencies

  Dit zijn resource-conflicten in de pipeline. Een voorbeeld hiervan zijn twee verschillende instructies die tegelijkertijd iets op hetzelfde register proberen te doen (load en store).

- Control dependencies

  Hierbij maken we een sprong in het geheugen (BRANCH, CALL, JMP, …) maar het doeladres ontbreekt.

- Data dependencies

  Dit komt voor wanneer een instructie data gebruikt van een vorige instructie.

**Vraag 3.10**

*Welke drie types van dependencies bestaan er in CPU-pipelining? Som ze op (zonder verklaring).*

- Structural dependency
- Control dependency
- Data dependency

**Vraag 3.11**

*Wat doet de assembly-instructie JMP? Verklaar in je eigen woorden.*

JMP kort voor jump, springt naar een andere plaats in de code (door middel van een pointer).

**Vraag 3.12**

*Hoe kan je het probleem van een structural dependency in CPU-pipelining oplossen? Benoem de techniek en verklaar die kort.*

Renaming: splitst het geheugen op in 2 onafhankelijke delen voor instructies en storage

**Vraag 3.13**

*Hoe kan je het probleem van een control dependency in CPU-pipelining oplossen? Benoem de techniek en verklaar die kort.*

Branch prediction: op voorhand voorspellen wat uitkomst gaat zijn

**Vraag 3.14**

*Hoe kan je het probleem van een data dependency in CPU-pipelining oplossen? Benoem de techniek en verklaar die kort.*

Operand forwarding: houdt tussentijdse resultaten apart bij

**Vraag 3.15**

*Som één belangrijke regel binnen de CPU-pipelining op.*

- Backward travelling instructies (in eender welke fase) zijn gevaarlijk
- Forward traveling instructies worden vergrendeld op elke fase
- Logica wordt in éénzelfde fase toegepast ( → in de ALU)

# Hoofdstuk 4: Branches

*Een branche is een instructie in assembly die de CPU mogelijk een andere instructie laat uitvoeren. Normaal gezien heb je een program counter die bijhoudt wat het adres van de volgende uitvoer-instructie is. Echter, een branche kan de program counter wijzigen.*

*Je hebt twee soorten branches: conditional branches en unconditional branches.*

**Conditional branche**

Deze branche zorgt niet noodzakelijk voor een verandering van het geheugen. Wanneer we naar een ander adres moeten springen, spreken we van een target address. Dit kan uit drie types bestaan:

- Direct (branche bevat het doeladres in de instructie)
- Indirect (branche weet waar het adres te vinden is)
- Relative (verschil tussen huidig en target adres)

**Unconditional branche**

Deze branche zorgt altijd voor een verandering van het geheugen. Dit komt bijvoorbeeld voor bij GOTO en JMP.

## Branche stalls

Een branche kan een stall veroorzaken in de pipeline. Zo kan je bij een conditional branche terugkeren naar een vorige fase. Bij een unconditional branche kan het zijn dat er geen geldig adres beschikbaar is.

Een oplossing hiervoor is branch prediction (of geen branches gebruiken).

**Branch prediction**

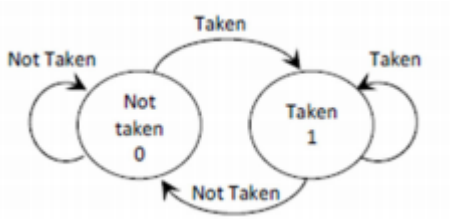
De uitvoering van instructies wordt versneld door de uitkomst van een conditionele branch te voorspellen en direct de juiste code uit te voeren. Dit doen we met een hardware integrated circuit (IC).

De branch prediction wordt gedaan door de IC die de resultaten van die conditionele instructie bijhoudt elke keer als ze uitgevoerd wordt. Dit heeft tot gevolg dat hoe vaker de instructie gebeurd is, hoe nauwkeuriger de voorspelling wordt.

```
if (voorwaarde) {
        //code uitvoeren indien taken
} else {
        //code uitvoeren indien not taken
}
```

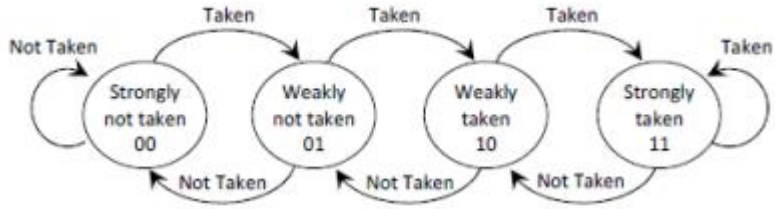
De eerste keer dat de prediction runt gaat hij gebruik maken van een statische prediction maar nadat de prediction al enkele keren is uitgevoerd gaat hij over naar een dynamische prediction.

1-bit prediction:




Voordeel: Je hebt minder geheugenruimte nodig omdat je 1 bit minder nodig hebt om het resultaat op te slaan.
Nadeel: Je maakt meer fouten.

2-bit prediction:




Voordeel: Je maakt minder fouten (2x minder).
Nadeel: Je hebt meer geheugenruimte nodig en de verwerking duurt langer.

## Voorbeeldvragen

**Vraag 4.1**

*Verklaar in eigen woorden wat een branche in assembly programming precies is.*

Een branche is een instructie in assembly die de CPU mogelijk een andere instructie laat uitvoeren. Normaal gezien heb je een program counter die bijhoudt wat het adres van de volgende uitvoer-instructie is. Echter, een branche kan de program counter wijzigen.

**Vraag 4.2**

*Geef een voorbeeld van een conditionele branch in een hogere programmeertaal (code).*

If-else, for loop

**Vraag 4.3**

*Kan je een if-instructie in assembly in één enkele instructie schrijven ? Antwoord met ja of neen.*

Neen:

// eerst een compare instructie die de waarde van 2 pointers vergelijkt

// dan een conditionele JMP instructie

// minimum 2 instructies dus

**Vraag 4.4**

*Wat is een target address bij een branche voor een CPU?*

Het adres wanneer we naar een ander adres moeten springen.

**Vraag 4.5**

*Waarom is er bij een conditional branch niet altijd sprake van branching?*

Omdat dit afhankelijk is van een bepaalde conditie. Soms gaat de huidige instructieset verder en soms wordt er naar een andere instructie gesprongen.

**Vraag 4.6**

*Geef een algemeen voorbeeld over hoe een conditional branch een stall in de CPU-pipeline kan veroorzaken.*

Bij een if-else conditie zijn er twee instructiesets die kunnen worden uitgevoerd. Eén voor wanneer het de if-statement is en één voor wanneer het de else-statement is. Vervolgens wordt een compare instructie in de pipeline geplaatst om de conditie toe te passen. Hierachter volgt direct ook één van de twee instructiesets zodat er geen gaten in de pipeline ontstaan. Nu kan het zijn dat uit de compare blijkt dat de verkeerde instructieset in de pipeline zit. Dan zal deze uit de pipeline moeten en de andere instructieset in de pipeline worden geplaatst. Dit zorgt voor een stall.

**Vraag 4.7**

*Geef een algemeen voorbeeld over hoe een unconditional branch een stall in de CPU-pipeline kan veroorzaken.*

Als het adres dat gegeven is niet een geldig adres is.

**Vraag 4.8**

*Leg in eigen woorden het concept van branch prediction voor een CPU uit.*

De uitvoering van instructies wordt versneld door de uitkomst van een conditionele branch te voorspellen en direct de juiste code uit te voeren. Dit doen we met een hardware integrated circuit (IC).

De branch prediction wordt gedaan door de IC die de resultaten van die conditionele instructie bijhoudt elke keer als ze uitgevoerd wordt. Dit heeft tot gevolg dat hoe vaker de instructie gebeurd is, hoe nauwkeuriger de voorspelling wordt.

**Vraag 4.9**

*Waarom verkies je best hardware prediction boven software prediction voor een CPU?*

Wanneer je een hardware prediction doet dan ga je een integrated circuit (IC) gebruiken die in de CPU ingebouwd is en die eigenlijk dedicated is (dat betekent dat hij zich enkel en alleen gaat bezighouden met prediction). Als je dit software-matig zou doen dan moeten al die voorspellingen ook via de CPU zelf passeren en dan moet de CPU zelf rekenkracht gaan gebruiken om ervoor te zorgen dat hij die prediction kan doen terwijl het net de bedoeling is van de prediction om het CPU-proces sneller te laten verlopen. Dit is waarom hardware prediction sneller is dan software prediction.

**Vraag 4.10**

*Hoe gebeurt branch prediction bij een CPU in de praktijk?*

Dit gebeurt met hardware.

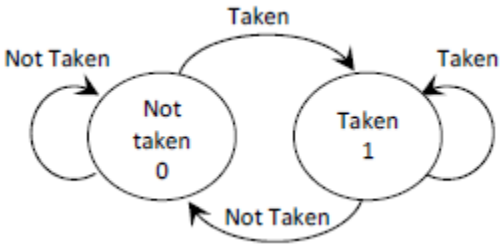
**Vraag 4.11**

*Wat betekent het Engelse woord "taken" bij CPU-branch prediction?*

Dit betekent dat de branch wordt genomen: de instructies zullen op een dezelfde plaats verdergaan, de PC zal worden veranderd.

**Vraag 4.12**

*Teken een schema met cirkels voor het concept 1-bit prediction. Voeg labels toe.*




**Vraag 4.13**

*Wat is het voordeel van 1-bit prediction ten opzichte van 2-bit prediction?*

Je hebt minder geheugenruimte nodig omdat je 1 bit minder nodig hebt om het resultaat op te slaan.

**Vraag 4.14**

*Wat is het voordeel van 2-bit prediction ten opzichte van 1-bit prediction?*

Je maakt minder fouten (2x minder).

**Vraag 4.15**

*Welke twee soorten branch prediction bestaan er? Som ze op.*

- Static prediction
- Dynamic prediction