Examenvragen 2023-2024

De bedoeling is om bij iedere vraag een kort, krachtig, volledig en juist antwoord te formuleren.
De quotering zal gebeuren op 2 (0 voor een fout antwoord, 1 voor een min of meer juist antwoord en 2 voor een volledig correct antwoord).
Dus de moeilijkheid is eigenlijk om in zo weinig woorden/zinnen de vraag de beantwoorden.
Let op, sommige details zijn weldegelijk belangrijk en dienen ook vermeld te worden. Zie daarom de lessen en eventueel uw eigen nota's.

Hoofdstuk 1

1. Wat zijn de voordelen van het gebruik van een compiler?

   Er worden eerst voorbewerkingen gemaakt vooraleer het uiteindelijke bestand gecompileerd wordt naar een andere taal (bv van C naar assembly). Dit heeft als voordeel dat onze compiler meerdere keren door dit bestand loopt en met als gevolg kan de compiler toekomstgericht werken. Onder andere: Hoeveel data is er nodig?, Welke gegevens hebben we nodig?, Hoeveel cpu kracht zal benodigd zijn?,... We werken dus niet blindelings, met als gevolg dat gecompileerde code zeer optimaal is.

2. Wat zijn de nadelen van het gebruik van een compiler?

   Een compiler heeft bestanden nodig om zijn functies/doeleinden uit te voeren en dit bestaat pas op het niveau van een operating system. Hierdoor kan een compiler dus ook niet op de lagere niveaus aanwezig zijn.

3. Wat zijn de voordelen van het gebruik van een interpreter?

   Alles kan in memory gebeuren en het concept van een bestand is dus niet nodig om een block code uit te voeren.

4. Wat zijn de nadelen van het gebruik van een interpreter?

   Stel we willen een bash file uitvoeren, de interpreter zal lijn per lijn dit bestand overlopen en uitvoeren. Dit heeft als

gevolg dat we in vergelijking tot een compiler niet in de toekomst kunnen kijken en dus wel kunnen vastlopen op onvoorziene omstandigheden. Hierdoor zijn meeste interpreters eerder aan de trage kant.

Hoofdstuk 2

5. Wat is de gedachte achter een CISC-architectuur?

(Complex instruction set computer) Men wil bij een CISC-architectuur zo complex mogelijke instructies sequentieel laten uitvoeren. Zo complex mogelijk zodat er optimaal gebruik gemaakt wordt van de hardware.

6. Wat is het nadeel van een CISC-architectuur?

Het werkt goed voor heel dure sterke hardware systemen maar dit is eerder een probleem voor goedkopere systemen die dat niet aankunnen. (dit omdat het goedkopere systeem niet genoeg hardware kan voorzien om de complexe instructie uit te voeren)

Hierdoor zullen de goedkopere systemen gebruik moeten maken van een microprogramma om de complexe instructieset te interpreteren, wat zal leiden tot enorm trage systemen.

7. Wat is de gedachte achter een RISC-architectuur?

(Reduced instruction set computer) Men wil zoveel mogelijk instructies tegelijkertijd te starten en niet te kijken naar de individuele uitvoeringstijd van een instructie.

8. Waarom is de instructieset beperkt bij RISC?

Omdat men werkt op een soort assemblageband en omdat het proces zo geautomatiseerd is, moet men de vrijheidsgraden consequent beperken.

Dus als gevolg van onze instructies op een gepipelined systeem te werken (/ zoveel mogelijk instructies tegelijkertijd uit te voeren) moeten onze instructies heel erg op elkaar lijken qua grootte, uitvoeringstijd en complexiteit.

9. Hoe wordt neerwaartse compatibiliteit bereikt tussen CISC en RISC?

Men gebruikt een CISC-achtige instructieset die omgezet wordt naar RISC micro-instructies die uitgevoerd worden op ons datapad. Waardoor we gebruik kunnen maken van de RISC eigenschappen en een pipeline/ alles wat daarbij komt.

10. Welke regels worden er aan RISC-instructies opgelegd?

- Alle instructies moeten rechtstreeks op de hardware uitgevoerd worden.
- Het aantal instructies dat wordt uitgevoerd per tijdseenheid moeten worden gemaximaliseerd. (kan dus zijn dat instructies out of order worden uitgevoerd)
- Instructies moeten makkelijk decodeerbaar zijn (snel en eenvoudig weten wat er nodig is bv -> vaste instructielengte, …
- Alleen load en store operaties mogen refereren naar memory, de rest van onze instructieset mag enkel gebruik maken van registers.
- Heel veel registers moeten voorzien worden.

11. Wat is een superscalaire architectuur? Hoe wordt de pipeline benut?

Een superscalaire architectuur is een architectuur met meerdere rekeneenheden. Dit om stap 4 (bv: ALU, LOAD, STORE, …) te ontlasten aangezien stap 3 meer instructies kan aanleveren dan stap 4 kan uitvoeren. Dit via paren van instructies die goed bij elkaar passen (benut pipline).

12. Op basis van wat werkt een GPU en hoe wordt dit bij een vectorprocessor gedaan?

GPU's werken volgens SIMD (Single Instruction, Multiple Data). -> aantal kernen afstemmen op het aantal data elementen (streven naar oneindig veel kernen)

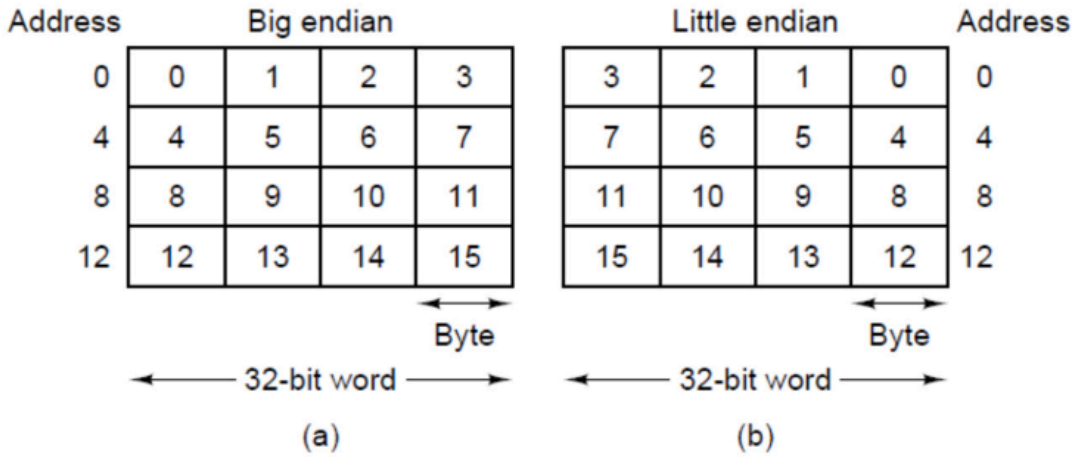
Vectorprocessors werkt ook adhv SIMD met aparte registers (SSE - Streaming SIMD Extensions). -> aantal vector

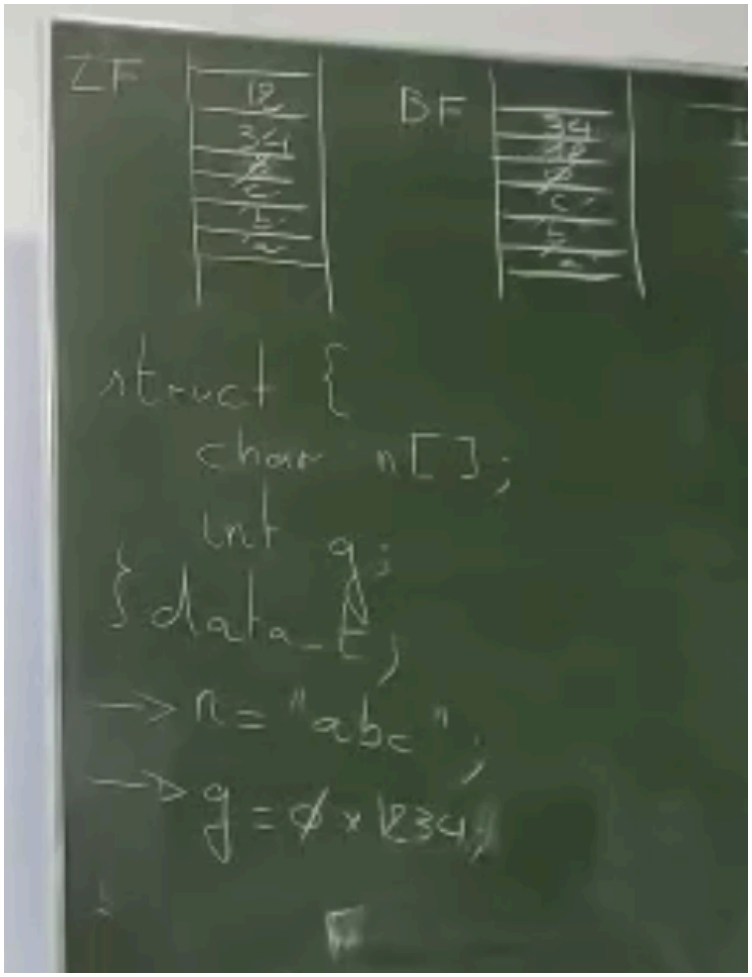
grootte afstemmen op het aantal data elementen (streven naar oneindig grote registers)

13. Maak een duidelijke schets van de voorstelling van variabele **p** in little endian en big endian en leg uit wat het probleem juist is bij de omzetting van de ene naar de andere voorstelling.

```
typedef struct {
        char str[4];
        int leeftijd;
} persoon;


Persoon p={“Wim”,50};
```





Het probleem bij het omzetten tussen big-endian en little-endian heeft te maken met de manier waarop byte gegevens, zoals integers, in het geheugen worden opgeslagen. In little-endian wordt het kleinere deel eerst

opgeslagen, terwijl in big-endian het grotere deel eerst komt. Als je probeert gegevens te lezen of schrijven met de verkeerde volgorde, kan dit leiden tot verwarring en onjuiste waarden bij het verplaatsen van gegevens tussen systemen met verschillende endian-formaten. (GPT)

14. Op basis van welk principe werken cachegeheugens en leg kort het principe uit? Geef enkele voorbeelden.

Het werkt op basis van lokaliteit. Dit betekent dat de cache op een gegeven tijd werkt in een bepaald geheugengebied, hiermee wordt bedoeld dat wanneer we een bepaald adres ophalen, dat meer dan waarschijnlijk de omliggende adressen ook nodig zijn.

Bv: een array, queue, stack, enz. (denk gwn opeenvolgende adressen)

15. Waarom werd RAID bedacht? Wat is het idee achter RAID?

Redundant Array of Inexpensive Disks.

Zodat we data kunnen recupereren in geval dat de schijf uitvalt.

Het idee erachter was eerder meerdere schijven te hebben waarop we onze data verspreiden omdat we dan meerdere IO opdrachten tegelijkertijd konden rennen. Het bijproduct hiervan is dat er veel grotere kans is dat 1 van de schijven uitvalt. Daarom zijn er RAIDs gekomen en worden tegenwoordig eerder independent disks in plaats van inexpensive disks genoemd.

16. Waarom is er bij RAID meer nood aan een redundante schijf?

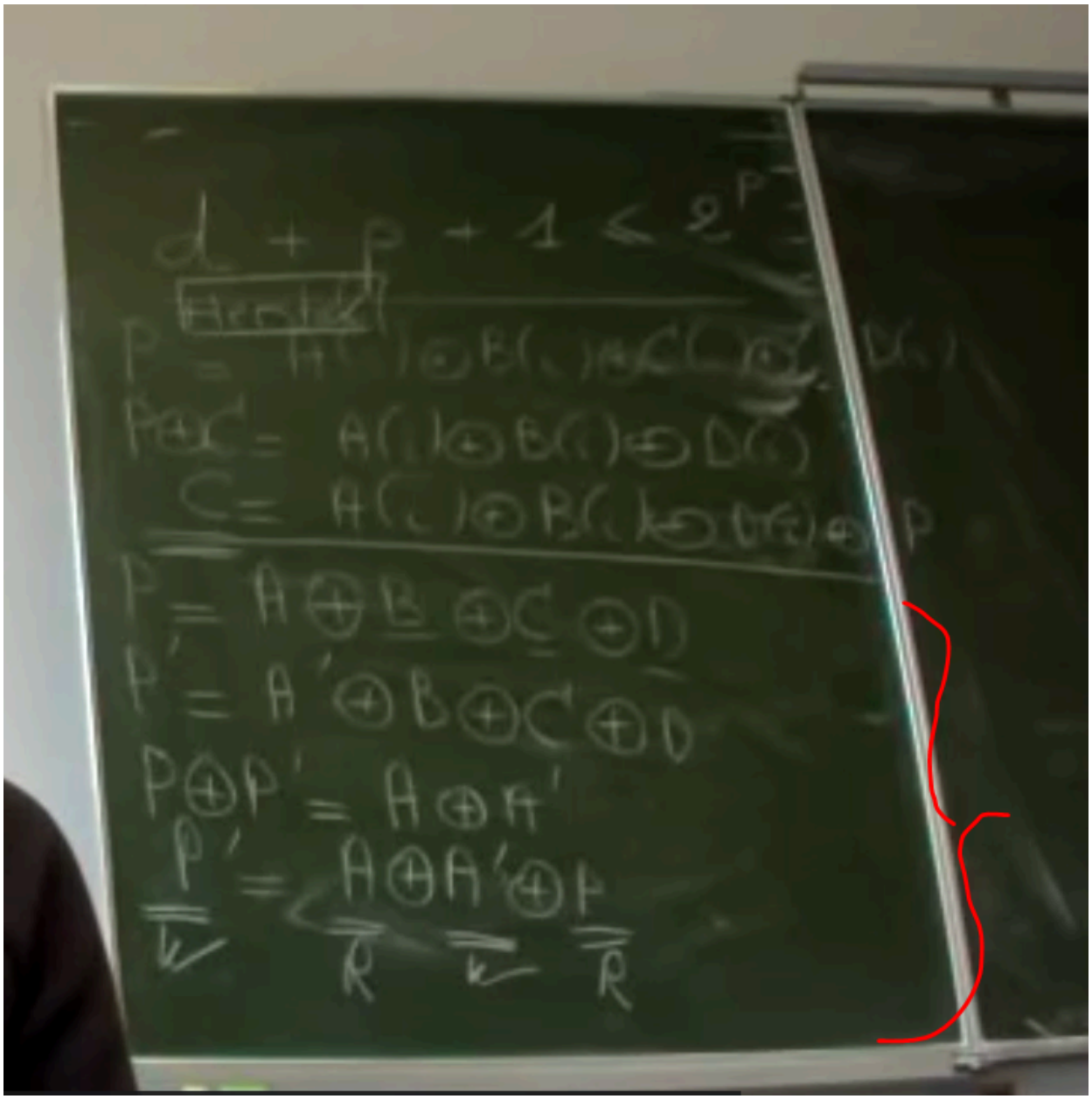
Omdat om het probleem van de beperkte io snelheid op te lossen ze meerdere schijven gebruikte wat als gevolg had dat er ook een grotere kans was dat een schijf uitvalde als 1 schijf 1/1000 kans heeft dan is dit met vier schijven 1/250 …

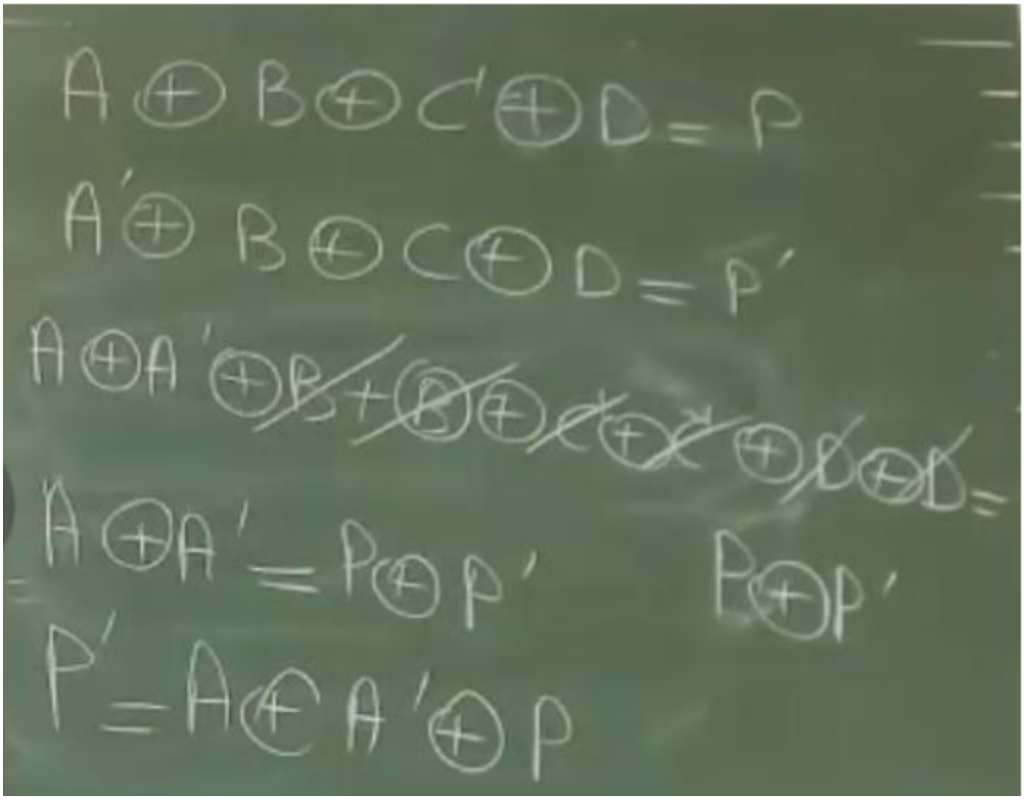
17. Wat is de impact van de strip-grootte op de prestaties?

RAID level 0 works best with large requests, the bigger the better. If a request is larger than the number of drives times the strip size, some drives will get multiple requests, so that when they finish the first request they start the second one. It is up to the controller to split the request up and feed the proper commands to the proper disks in the right sequence and then assemble the results in memory correctly. Performance is excellent and the implementation is straightforward.

18. Bij RAID-4 is er een schrijfstraf… Die kan herleid worden tot twee leesopdrachten en twee schrijfopdrachten.

Gegeven een RAID-systeem met 4 disks; geef dan het bewijs dat dit kan met slechts 4 I/O-opdrachten ongeacht het aantal schijven.


d + p + 1 ≤ 2^p
P = A ⊕ B ⊕ C ⊕ D
P ⊕ C = A ⊕ B ⊕ D
C = A ⊕ B ⊕ D
P = A ⊕ B ⊕ C ⊕ D
P' = A' ⊕ B ⊕ C ⊕ D
P ⊕ P' = A ⊕ A'
P' = A ⊕ A' ⊕ P
L
R
L
R



19. Wat bedoelt men bij RAID-2 met mechanisch gesynchroniseerd en waarom is RAID-2 betrouwbaarder dan RAID-3?

Mechanische sync -> gesynchroniseerd qua armpositie en rotationele positie vd schijf. Raid 2 is betrouwbaarder omdat RAID-3 alleen maar error detection heeft bij random errors, terwijl bij RAID-2 de Hamming code dit kan oplossen (error correction)

20. Hoe komt het dat solid-state drives een write amplification factor hebben en gewone klassieke magneetschijven niet?

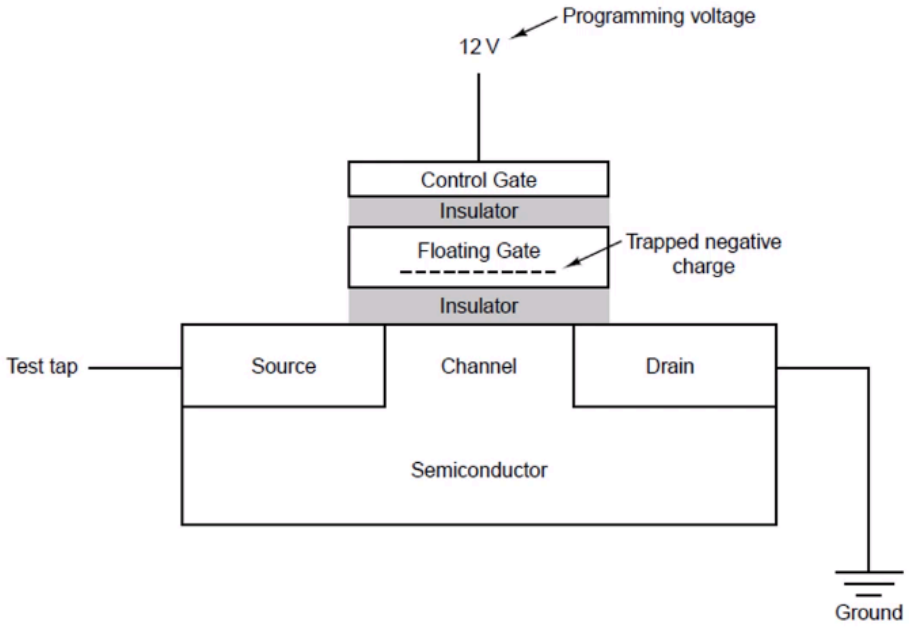
Write amplification -> als we 1 schrijfopdracht doen, dat leidt tot meerdere schrijfopdrachten zonder dat we het willen.

Bij een klassieke harde schijf zijn er 2 mogelijkheden, we kunnen lezen of we kunnen schrijven en als er al data op een locatie zit is dat geen probleem, het wordt gewoon overschreven.

Bij een systeem gebaseerd op een floating gate mosfet gaat dat niet zomaar (solid state is hierop gebaseerd). We

moeten eerst alle transistoren deblokkeren dit betekent dat wanneer we een schrijfopdracht wil doorvoeren we eerst de cellen moeten wissen en dan pas onze schrijfopdracht kunnen doen. Dit geeft als volgt een probleem dat de wis-operatie op een veel groter block is dan de schrijf-operatie (schrijf is 2-,8-,16-,... kb en een wis-operatie is 128-,256-,512kb). Dus als we 4kb willen schrijven moeten we bv 256 kb eerst in de controller gaan laden, de nodige updates uit te voeren en dan wegschrijven naar een andere locatie vooraleer de wissel-operatie plaats kan vinden. Dit wordt er bedoeld met write amplification.

21. Maak een schets van een floating gate MOSFET.



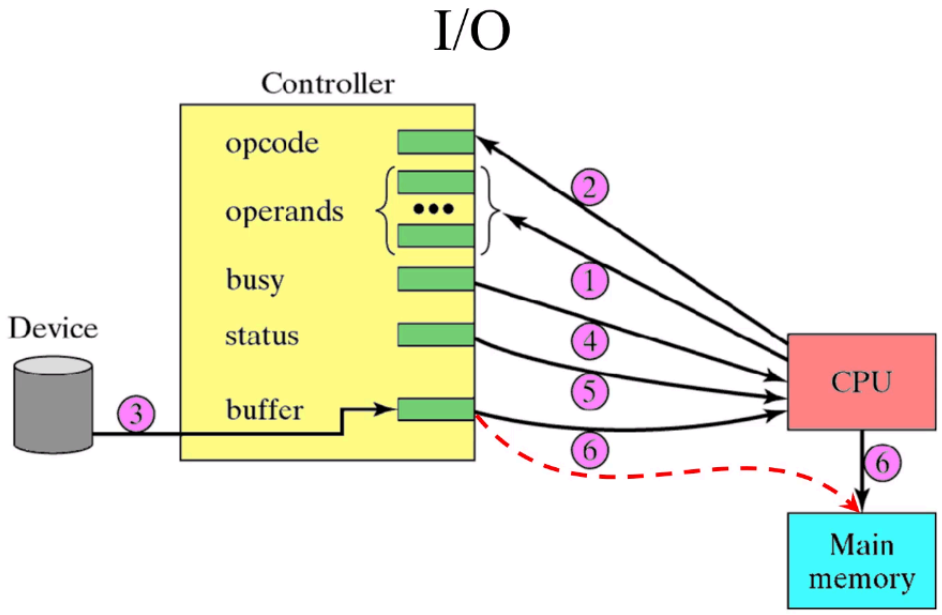
22. Wat is het probleem wanneer je DMA gebruikt in combinatie met cachegeheugens?

Wanneer men adhv DMA data overschrijft dat het cachegeheugen ingelezen heeft, zal het cachegeheugen met een “verouderde versie” zitten.

Dit gebeurde langs beide wegen want als we schrijf-operaties naar het cachegeheugen toelaten kan het zijn dat ons cachegeheugen een recentere versie bevat

dan ons geheugen en als onze DMA dan data van het geheugen naar de schijf wilt schrijven zal hij verouderde data nemen.

23. Geef stap voor stap aan hoe de CPU een I/O-opdracht geeft aan een I/O-device en hoe het I/O-device het antwoord terugstuurt.



1. Functie vastleggen in 1 van de registers van ons I/O-device
2. De argumenten meegeven van wat er exact moet gebeuren naar het I/O-device
3. Onze OS moet dan het proces dat deze opdracht aangevraagd heeft even stilleggen.
   Het proces in kwestie wordt tijdelijks uitgeswapt.
4. Wanneer de lees-operatie klaar is (melding adhv een interrupt bv) en wordt die data op de buffer weggeschreven
5. De status vlaggen worden aangepast
6. De OS zal het proces dat staat te wachten op deze IO bewerking deblokkeren en in de queue van gerede processen brengen
   De OS zal de data uit de buffer lezen en wegschrijven naar het geheugen

Stap 6 kan overgslagen worden dit gebeurd dan via de DMA (rode stippellijn)

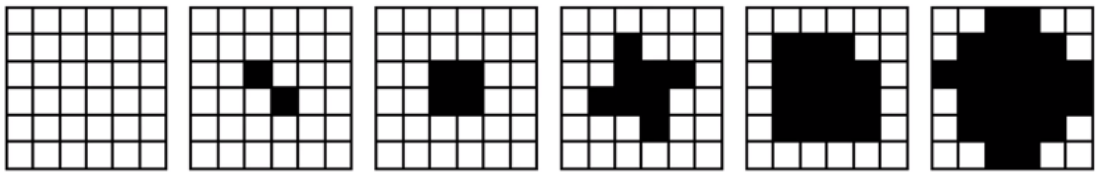
24. Bij een bus waar ook de CPU zich op bevindt, worden de prioriteiten onder de verschillende devices verdeeld. Welke prioriteit krijgt de CPU en waarom? Hoe noemt men dit fenomeen?

CPU krijgt de laagste prioriteit dit komt omdat we software interrupts (afkomstig van de cpu zelf) en hardware interrupts (afkomstig van I/O-devices) hebben en hardware interrupts zijn belangrijker dan software interrupts omdat deze mogelijks kunnen leiden tot dataverlies moesten we ze laten wachten. Daarom heeft een CPU de laagste prioriteit. Wanneer er geen I/O-devices gebruikmaken van een bus kan de CPU deze bus in beslag nemen, dit noemt cycle stealing.

25. Wat is de functie van de embedded CPU bij een laserprinter?

Om PostScripts of PDF's om te zetten naar afbeelding en tekst. We sturen dus geen bitmap meer naar de printer maar gewoon broncode. (bitmap kan rap enorm zwaar zijn, wat belastend is voor het netwerk)

26. Hoe worden grijstinten bekomen? Maak een schets en leg uit.



We maken gebruik van een techniek die we halftoning noemen.

We nemen 6x6 cellen elk met 37 mogelijkheden (allemaal wit tot allemaal zwart).

256/37 ~= 7 dus de grijswaarde van a is ongeveer 0-7, terwijl de grijswaarde van b ergens tussen 14-21 ligt,...

27. Waarom is het afdrukken van kleuren eigenlijk niet zo triviaal? Welke kleurmodellen worden er gebruikt?

Omdat schermen werken met het RGB (Red, Green, Blue) systeem terwijl printers gebruik maken van CMYK (Cyan, Magenta, Yellow, Key -> Black). Het omvormen van het ene kleur systeem naar het andere is allesbehalve gemakkelijk.

- Color monitors use transmitted light;
  - Color printers use reflected light.
- Monitors have 256 intensities per color;
  - Color printers must halftone.
- Monitors have a dark background;
  - Paper has a light background.
- The RGB gamut of a monitor and the CMYK gamut of a printer are different.

28. Wat zijn de beperkingen van ASCII en hoe heeft men ASCII eigenlijk in eerste instantie uitgebreid?

ASCII heeft 128 combinaties, enkel cijfers en letters zonder accenten of leestekens, adhv 7 bits per teken. Men breidde uit van een 7-bit systeem naar een 8 bit-systeem (om uit te breiden voor andere Europese landen).

29. Unicode werkt met 16-bit karakters. Welke problemen zijn hierbij voorgekomen en hoe heeft men die codering in eerste en enige instantie uitgebreid zonder het aantal bits te verhogen?

Omdat ons alfabet gebaseerd is op de letters die we hebben waarmee alles hiervan werkte maar in andere landen (de meer oosterse landen (Asia)) zijn hun alfabetten gebaseerd op klanken en uitspraken. Bij die talen komen er vaak nieuwe letters bij wat voor problemen zou zorgen in het systeem, we zouden te weinig combinaties hebben. Men wou uitbreiden door in 3 dimensies te werken (*codeplane*) dit betekent dat we een

extra parameter meegeven en dat dezelfde combinatie meerdere keren kan voorkomen maar op verschillende “planes”.

30. Wat zijn de eigenschappen van UTF-8. Maak een schets van een sequentie van 3 bytes. Hoe worden ASCII-tekens voorgesteld?

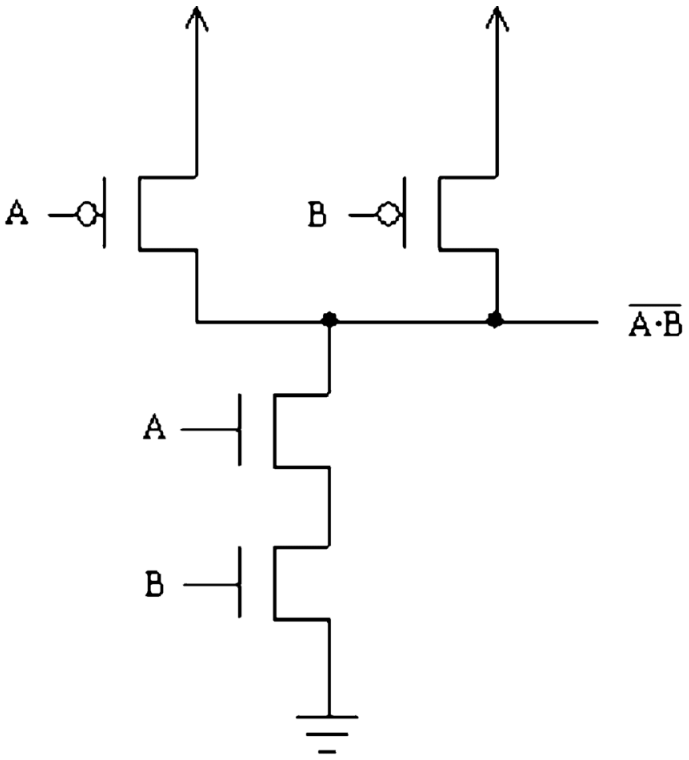
Antwoord met vb zelfsync:
- Variabele lengte, tekens worden gecodeerd met 1 tot 4 bytes
- Een van de belangrijke eigenschappen is zelfsynchronisatie. Dit betekent dat het mogelijk is om te bepalen waar een nieuwe UTF-8-sequentie begint, zelfs als je niet aan het begin van een reeks bent gestart. Elk teken begint met een bepaald patroon dat aangeeft hoeveel bytes het teken gebruikt. Bijvoorbeeld:
1 byte-teken: 0xxxxxxx
2 byte-teken: 110xxxxx 10xxxxxx
3 byte-teken: 1110xxxx 10xxxxxx 10xxxxxx
4 byte-teken: 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

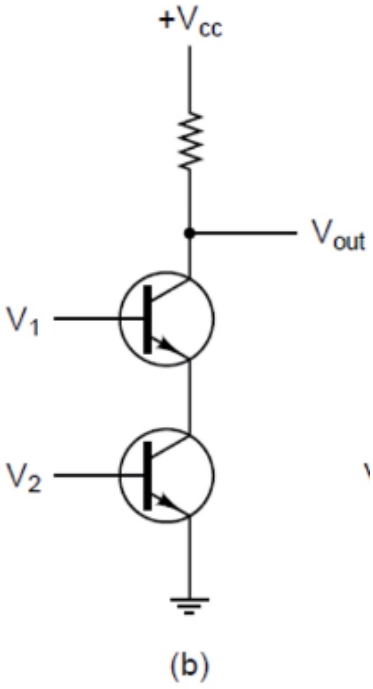
De bits "10" aan het begin van elke byte die geen start-byte is, geven aan dat dit een vervolg-byte is. Als een byte begint met "0", is het een ASCII-teken. Dit betekent dat je, als je ergens in een UTF-8-sequentie begint, kunt achterhalen waar het volgende karakter begint door naar de bits te kijken en te bepalen hoeveel bytes het karakter in beslag neemt.
- Schets sequentie 3 bytes: 1110xxxx 10xxxxxx 10xxxxxx
- ASCII-tekens: 1 byte en begint met 0

Hoofdstuk 3

31. Maak een schets van een NAND-poort met CMOS-technologie. Hoeveel transistoren heb je nodig om een AND-poort te maken?

NAND
AND (6 transistoren)





(b)

32. Maak een schets van een half-adder. Wat is het probleem? Maak vervolgens een schets van een full-adder.

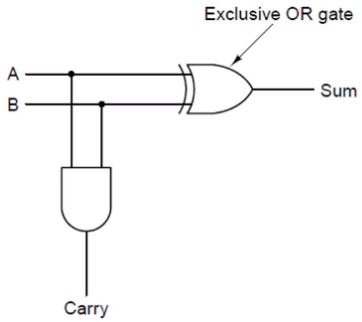
Half-adder

werkt goed bij minst beduidende bits maar de bits daar tussenin zijn er problemen want we moeten

rekening houden met die bits die er naast liggen, carry van bit die ervoor afging zal problemen geven.

Arithmetic Circuits (2)

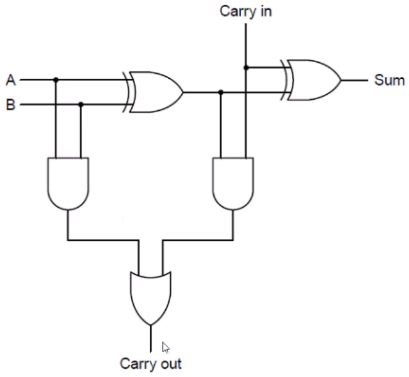
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Full-adder (gwn 2 half-adders na elkaar)

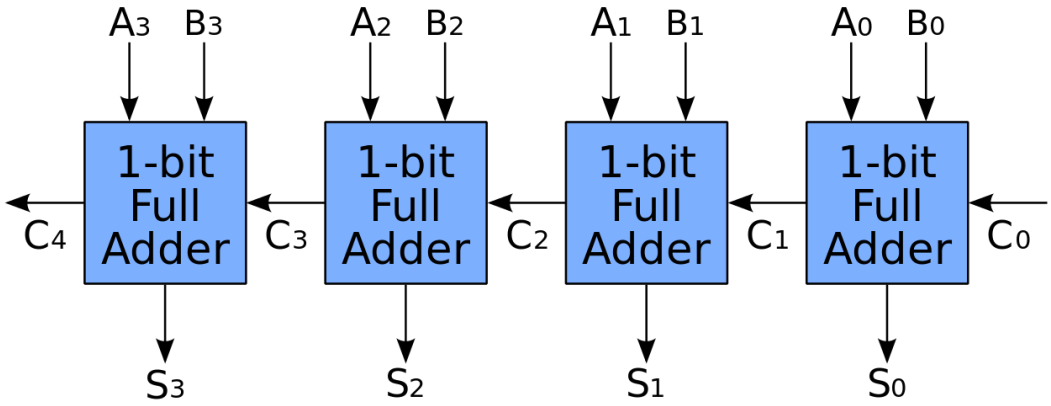
› Arithmetic Circuits (3)

A	B	Carry in	Sum	Carry out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



33. Wat is een ripple carry adder? De performantie kan je opdrijven door bv. een carry select adder te gebruiken. Hoe werkt dit?

Een ripple carry adder is een schakeling voor binair optellen die bestaat uit meerdere full-adders. Het voegt bits één voor één toe, afhankelijk van de vorige carry. Met carry select adder parallelle berekeningen doen om de snelheid te verhogen door vooraf mogelijke carry-bits te selecteren.



34. Wat is het voordeel van het gebruik van de signalen RAS en CAS bij een geheugenchip? Wat is dan weer het voordeel van het gebruik van verschillende geheugenbanken?

Dat we veel minder pinnen hebben door gebruik van RAS (Row Address Strobe) en CAS (Column Address Strobe) wat het dus ook goedkoper maakt alhoewel het hierdoor ook iets trager werkt.

Het voordeel van extra geheugenbanken is dat er mogelijk is tot overlap wat enige vorm van parallelliteit toelaat (ook minder pinnen).

35. Wat is meestal de breedte van de databus bij een CPU?

Meestal recht evenredig met de woordlengte van de CPU

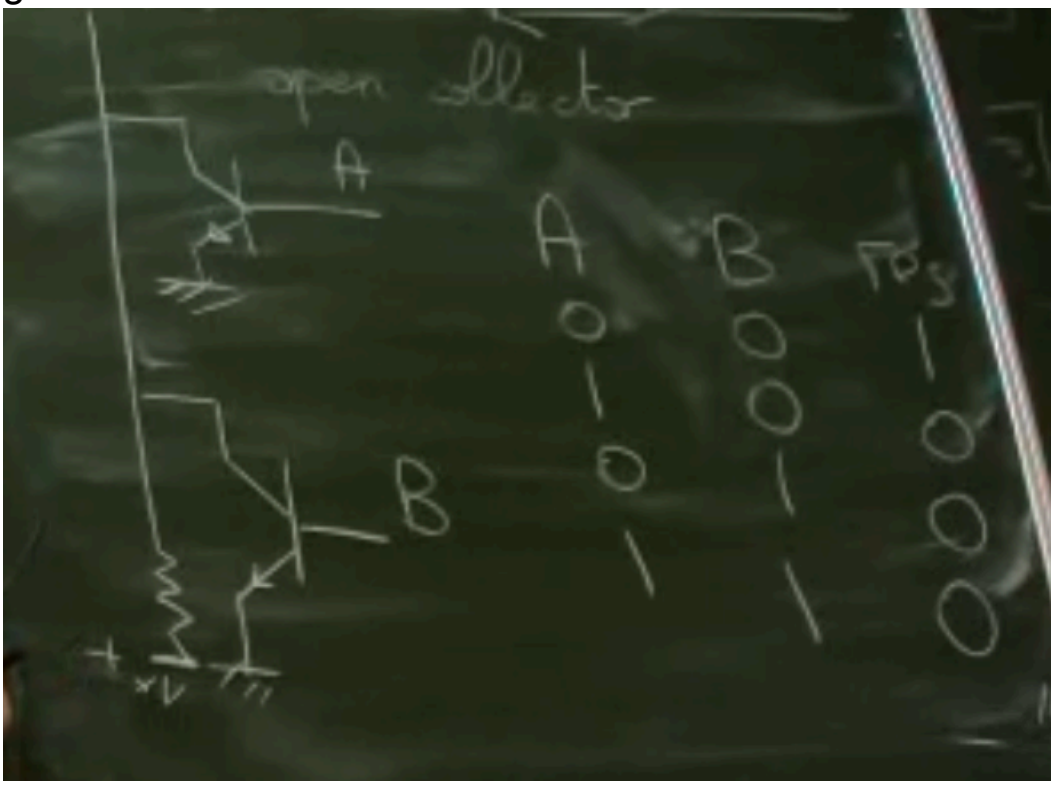
36. Waarom heb je een busdriver nodig voor het aansluiten van een I/O-controller op een bus? (twee redenen)

Omdat onze bus een bus protocol heeft en door deze busprotocol zijn onze spanningsniveaus niet echt compatibel met die van de microprocessor/-controller, deze moeten dus ook geregeld worden.

Hoe meer apparaten hoe meer impedantie dat we hebben dit moet ook geregeld worden zodat we zeker genoeg “power” hebben om onze lijn naar boven of omlaag te trekken.

37. Maak een schets van een wired-or schakeling met twee devices die gebruikmaken van een open-collector busdriver. Waarom is dit wired-or?

OF schakeling omdat men met negatieve logica werkt en wired-or omdat de manier is waarop we devices op een bus gaan gaan aansluiten.



38. Wat zijn de nadelen van een parallelle bus waardoor hedendaagse bussen seriële bussen zijn? Maak een schets om eventueel te verduidelijken.

/

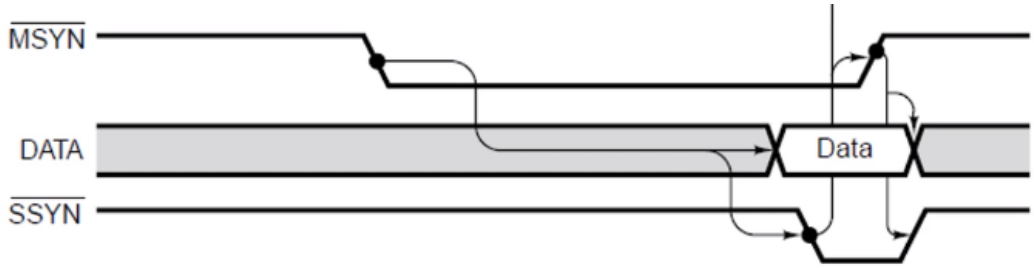
39. Wat wordt er bedoeld met een full-handshake? Geef een voorbeeld.

De up and down lijnen noemen we de handshake lijnen ->

Master vraagt voor data -> slave zet de transfer op en vraagt aan master om deze te lezen -> master leest de data in

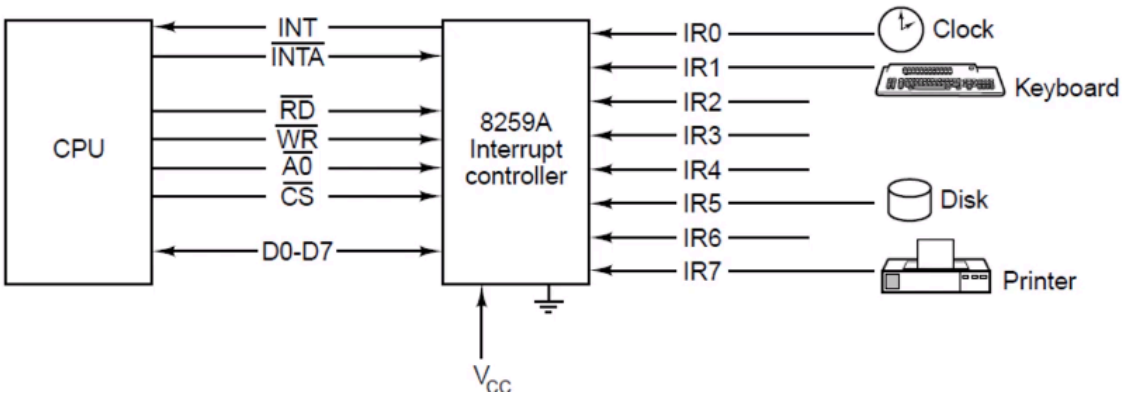
en vraagt aan slave om de volgende transfer op te zetten. Dit is een full handshake

Voorbeeld: /



40. Bespreek stap voor stap hoe een interrupt wordt afgehandeld bij gebruik van de 8259A prioriteitsinterruptcontroller.

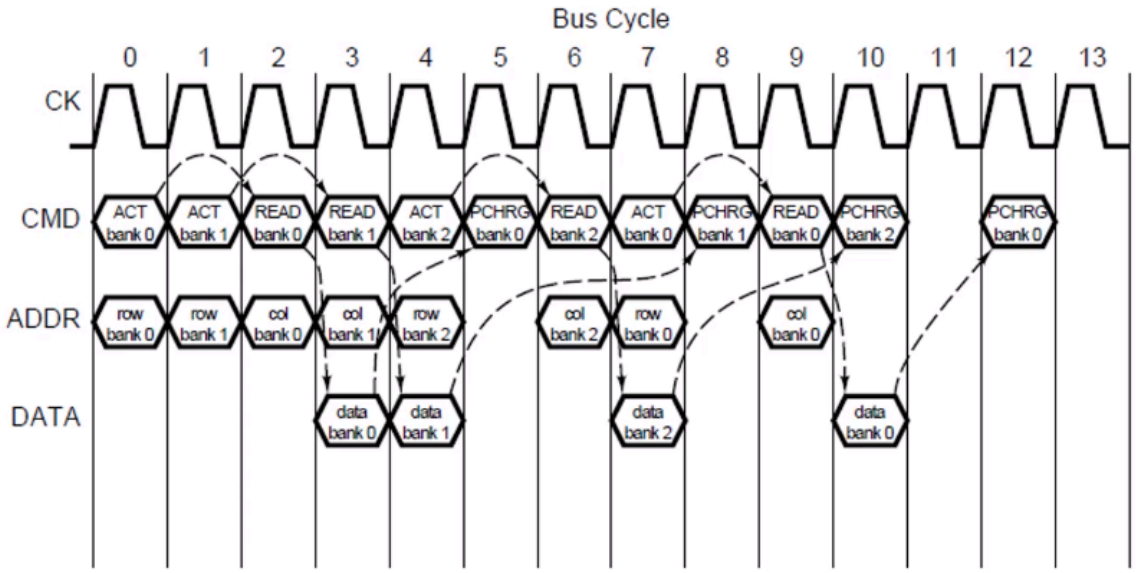
1. Een device geeft een interrupt (eender wat)
2. 8259A stuurt deze interrupt door naar de CPU (kan meestal niet direct antwoorden hierop)
3. CPU geeft interrupt ACK aan de 8259A IC -> INTA gebruikt om te tonen dat CPU klaar is
   om interrupt te verwerken en dat 8259A het interrupt nummer mag doorsturen
4. 8259A zal het interrupt nummer (index in IVT) doorsturen via de D0-D7 poort
5. Het interrupt adres wordt in de programmateller ingeladen (mogelijks onnodig)
6. Oude programmateller wordt op de stapel geplaatst + CPU context
   (mogelijks onnodig)
7. CPU schrijf dan bit weg naar 1 van de 2 registers van 8259A om aan te geven dat de interrupt afgehandeld is
8. Geen andere interrupts? Dan zal 8259A de INT lijn weghalen anders repeat cycle

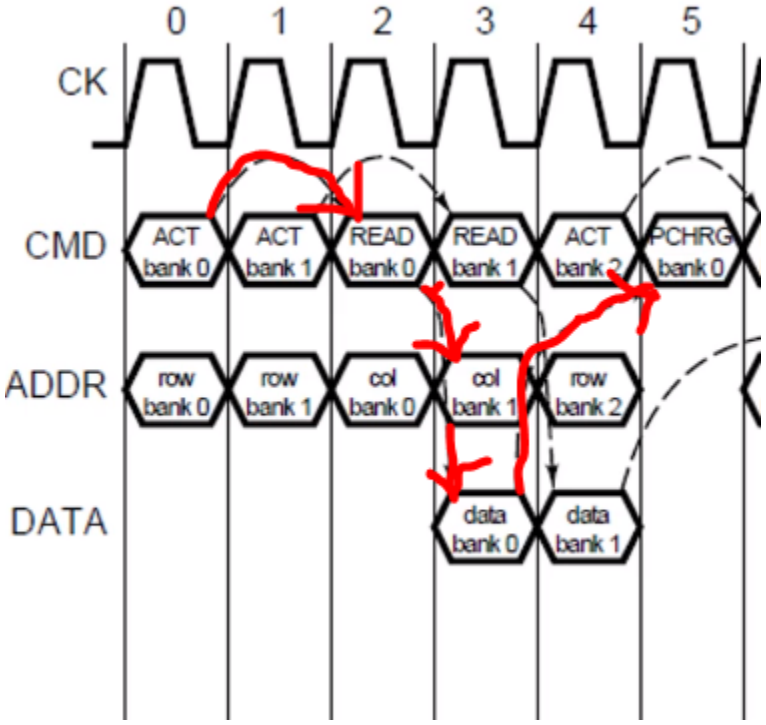


41. Hoe werkt pipelining voor DDR3/4/5 SDRAM?

Activate phase -(1 cycli ertussen)-> Read phase (-> col adres meegeven -> data ophalen) -(2 cycli ertussen)-> Precharge phase

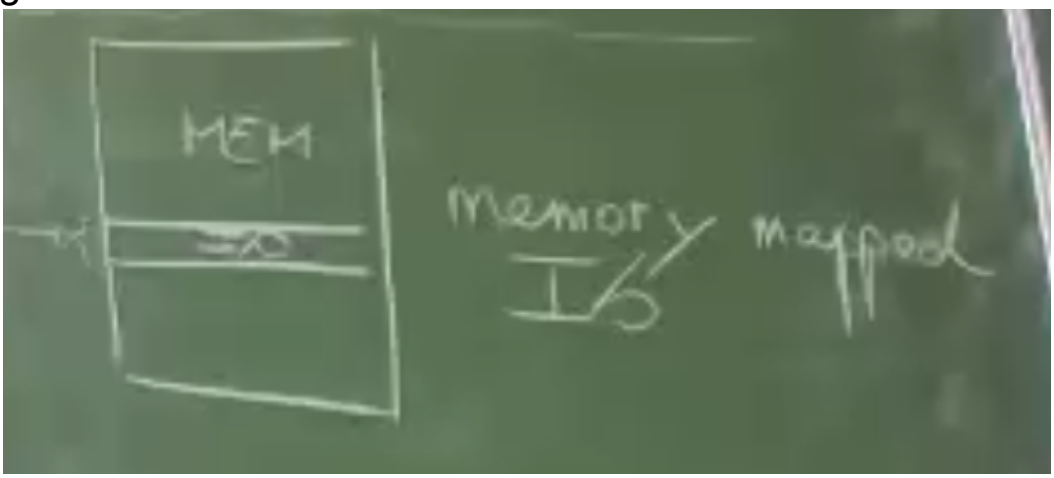
Pipelining


Bus Cycle
0 1 2 3 4 5 6 7 8 9 10 11 12 13
CK
CMD
ACT bank 0
ACT bank 1
READ bank 0
READ bank 1
ACT bank 2
PCHRG bank 0
READ bank 2
ACT bank 0
PCHRG bank 1
READ bank 0
PCHRG bank 2
PCHRG bank 0
ADDR
row bank 0
row bank 1
col bank 0
col bank 1
row bank 2
col bank 2
row bank 0
col bank 0
DATA
data bank 0
data bank 1
data bank 2
data bank 0


0 1 2 3 4 5
CK
CMD
ACT bank 0
ACT bank 1
READ bank 0
READ bank 1
ACT bank 2
PCHRG bank 0
ADDR
row bank 0
row bank 1
col bank 0
col bank 1
row bank 2
DATA
data bank 0
data bank 1

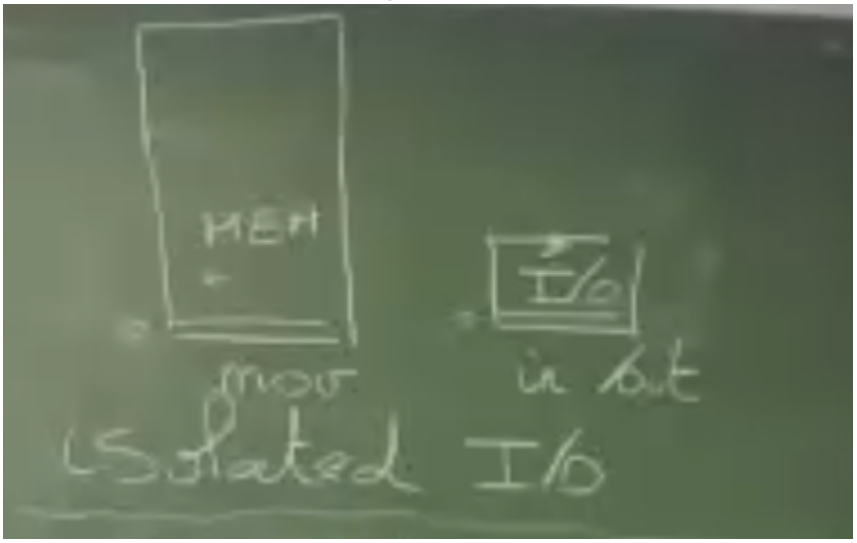
42. Bespreek memory mapped I/O en geef voor- en nadelen? Maak desnoods een schets.

+ Kunnen het schalen met een groter geheugen dat bv niet kan bij isolated.
- We kunnen een gedeelte van ons gwn geheugen niet gebruiken.



43. Wat is isolated I/O, hoe werkt dit en wat zijn de voor- en nadelen? Maak desnoods een schets.

+ We kunnen het volledige geheugen gebruiken.
- Moeten instructies toevoegen (in/out) en een adresseer-mode moeten toevoegen, hierdoor is administratie wat complexer.



44. Wat is partiële adres decodering en waarom is dit niet aangewezen?

Een simpelere vorm van de normale adres decodering waarbij we kijken voor bits als volgt:

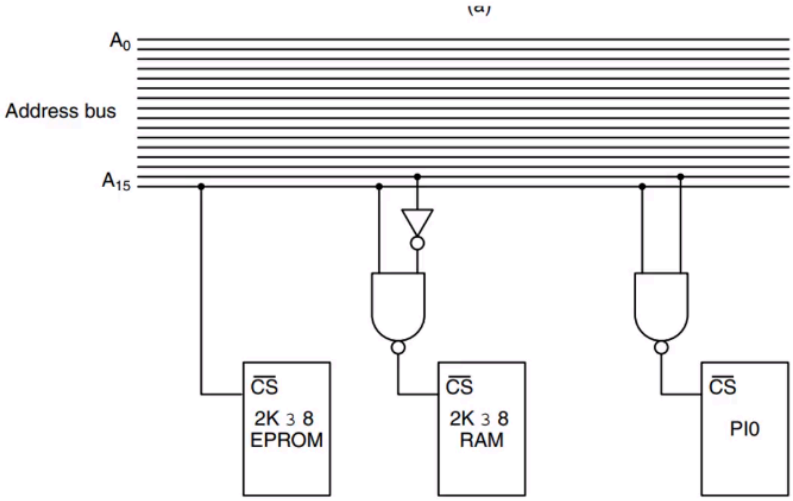
A15 = 0 -> EPROM

A15 = 1, A14 = 0 -> RAM

A15 = 1, A14 = 1 -> PI0

Het grote nadeel van dit systeem is dat alle andere adressen niet kunnen gebruikt worden voor iets anders. Dit geeft veelal fouten omdat er vaak toch meerdere delen nodig zijn etc, dus gebruikt men normale full adres decoding.

› Partial address decoding



Hoofdstuk 4

45. Wat is in zijn eenvoudigste vorm de definitie van een datapad?

Een hoop registers dat via input-bussen verbonden is met één of meer rekeneenheden en via een terugkeer-bus gekoppeld is aan diezelfde registers.

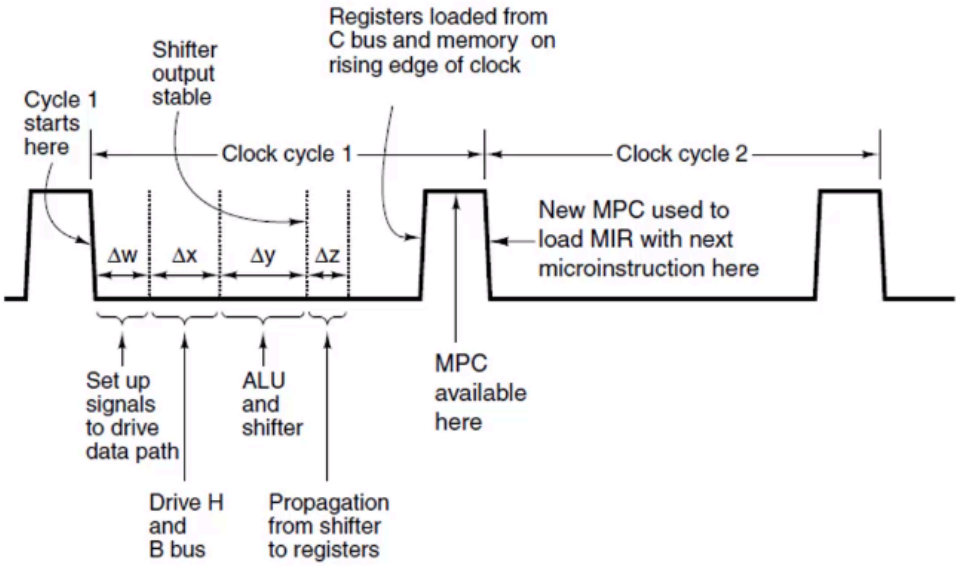
46. Wat zijn de mogelijkheden om een geheugenaanvraag te doen via de MIC-1? Wanneer wordt een geheugenaanvraag gesteld en wanneer wordt het antwoord verwacht?

indirect en directe adressering

- 2 Signals to indicate memory read/write via MAR/MDR.
- 1 Signal to indicate memory fetch via PC/MBR.

47. Wat is de datapadtiming van de MIC-1?

Data Path Timing (1)



48. Waarom is timing op een datapad heel belangrijk?

Enkel na de bepaalde timing van
delta w * delta x * delta y * delta z
zal hij een zinvolle waarde bevatten in de C bus.

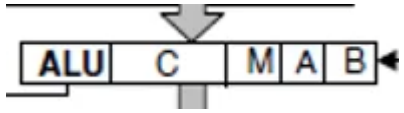
49. Waarom moet de puls van de klok kort en krachtig zijn op het einde van een datapadcyclus?

Ze moet kort, krachtig en edge triggered zijn omdat de schakeling constant onder spanning/stroom staat en de ALU zal dus altijd wel een waarde produceren op de C-bus. Dus zeker edge triggered in plaats van level triggered anders kan het zijn dat hij opnieuw een cyclus doet.

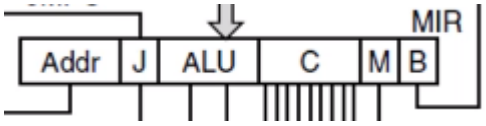
50. Uit welke velden bestaat een micro-instructie en een micro-operatie? Wat is het verschil tussen beide?

Het verschil is dat micro-operaties geen next address heeft omdat ze uit hun eigen al op volgorde staan, ook heeft het geen Jam bits: JAMZ, JAMN, JMPC.

Micro-operaties ->

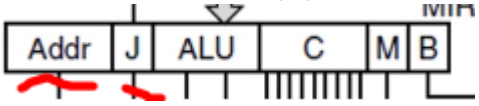


Micro-instructie ->

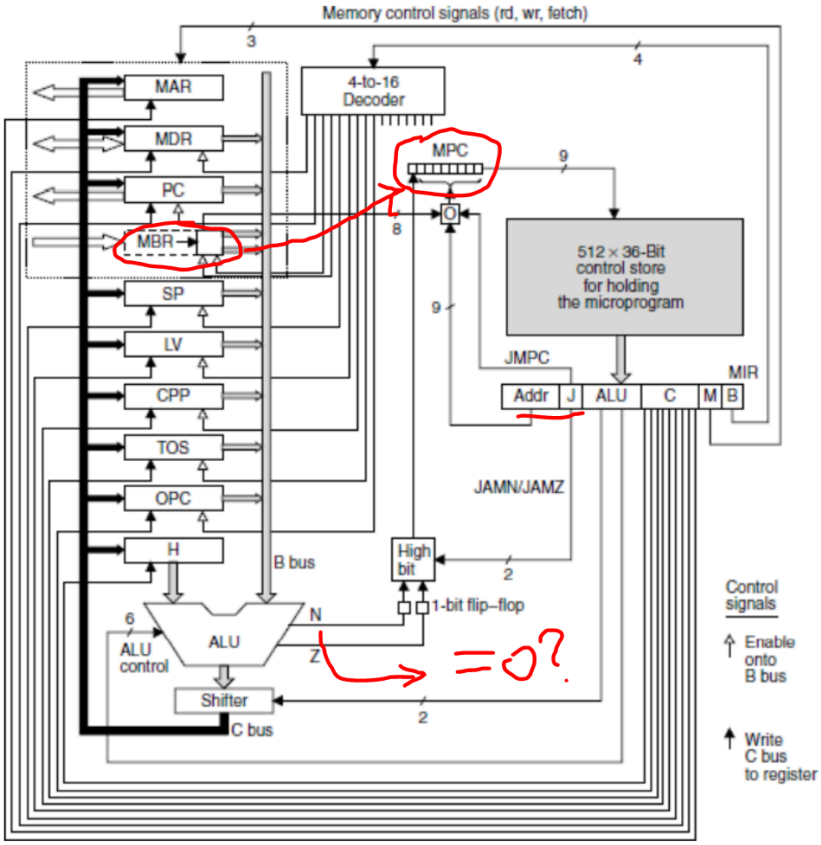


51. Parallel aan de aansturing van het datapad wordt het adres van de volgende micro-instructie bepaald. Hoe gebeurt dit precies?

Terwijl ons datapad een micro-instructie aan het uitvoeren is, zal hij parallel het adres van de volgende instructie bepalen. Hiervoor wordt het “next address” veld gebruikt en ook JAM bits (J).



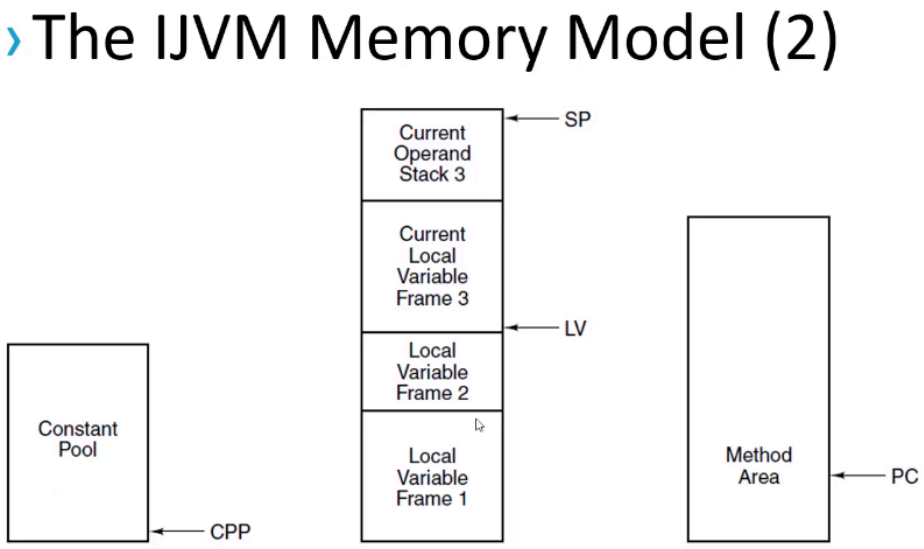
(kijken of het resultaat negatief is / 0 -> dan JMPC zegt je bent aan laatste micro-instructie voor afhandelen van ISA-level instructie -> neem MBR toestand in MPC laden)



52. Geef het geheugenmodel van de IJVM en bespreek wat de verschillende delen bevatten en hoe ze gebruikt worden tijdens de uitvoer van een programma.

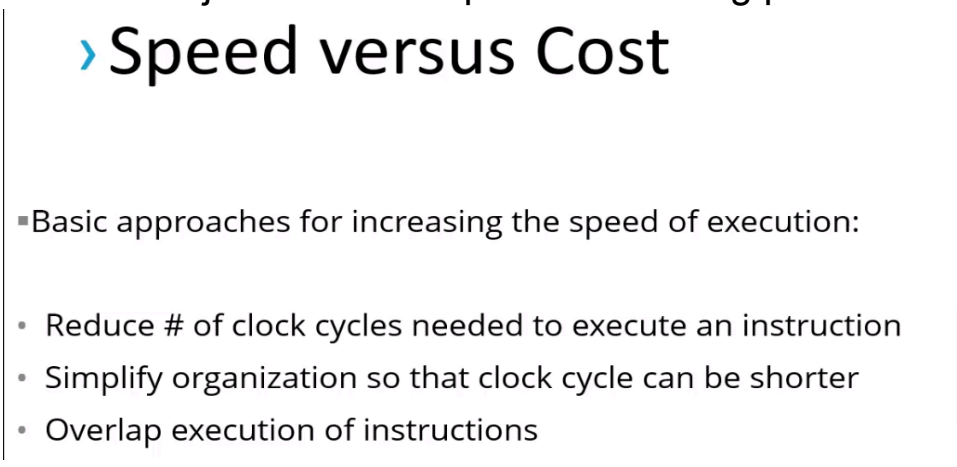
1. The constant pool -> Als we klasse of interface inladen zal de constant pool alle constante daarvan inladen (literaire-,letterlijke-,string-,... constanten), om hier iets uit op te halen gebruiken we de constant pool pointer (CPP)
2. The Local variable frame -> we hebben al een methode staan
   frame 1 -> Hoort bij deze methode
   frame 2 -> Binnen deze methode wordt een tweede methode aangeroepen die staat hier
   frame 3 -> Binnen de tweede methode werdt een derde methode aangeroepn die staat hier (is ook current LVF)
3. The operand stack -> Boven op de LVF doen we eventueele berekening die komen op de stapel (gebruikt stack pointer)

4. The method area -> Bevat onze code, gebruiken de PC om te bewegen naar de volgende uit te voeren ISA level instructie (byte geadresseerd)



53. Hoe kan de uitvoeringstijd van ISA-instructies worden verbeterd?

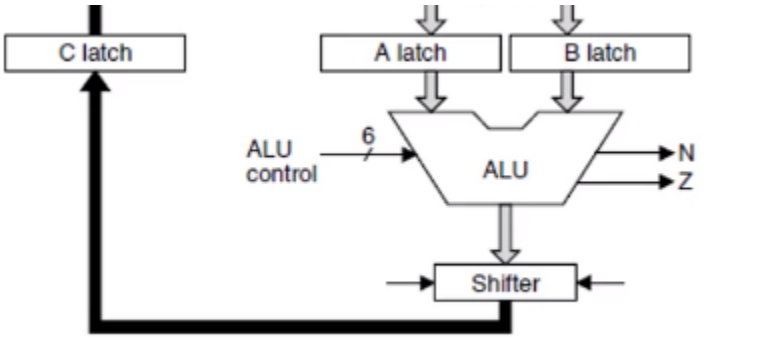
Instruction fetch unit: dit is een aparte unit om instructies op te halen; een eerste vorm van parallellisatie. Dit gebeurt onafhankelijk van het datapad verbetering performantie



54. Waarom is het belangrijk dat de C-bus constant in de gaten wordt gehouden?

55. Waarom zijn bij een gepipelined model van de MIC-2 drie latch registers toegevoegd?

Dienen om toestand te bewaren van vorige datapad cyclus. Op einde van datapad cyclus zal nu de toestand van A opgeslagen in A latch en toestand van B in B latch. Uitkomst van ALU en shifter in de C latch en gaat hetgeen dat in C bus staat vast zetten in de doel registers. Zo vormen we de MIC-3.



56. Gegeven onderstaande MAL-code:

MAR = SP-1; rd
MAR = SP
H = MDR; wr

Maak van deze drie micro-instructies een gepipelinde versie waarbij er 3 latch-registers (A, B en C) werden toegevoegd om de componenten onafhankelijk te doen werken. Welk probleem kom je hier tegen?

Pipelined Design: The Mic-3 (4)

Label	Operations	Comments
swap1	MAR = SP – 1; rd	Read 2nd word from stack; set MAR to SP
swap2	MAR = SP	Prepare to write new 2nd word
swap3	H = MDR; wr	Save new TOS; write 2nd word to stack
swap4	MDR = TOS	Copy old TOS to MDR
swap5	MAR = SP – 1; wr	Write old TOS to 2nd place on stack
swap6	TOS = H; goto (MBR1)	Update TOS

Pipelined Design: The Mic-3 (5)

	Swap1	Swap2	Swap3	Swap4	Swap5	Swap6
Cy	MAR=SP–1;rd	MAR=SP	H=MDR;wr	MDR=TOS	MAR=SP–1;wr	TOS=H;goto (MBR1)
1	B=SP					
2	C=B–1	B=SP				
3	MAR=C; rd	C=B				
4	MDR=Mem	MAR=C				
5			B=MDR			
6			C=B	B=TOS		
7			H=C; wr	C=B	B=SP	
8			Mem=MDR	MDR=C	C=B–1	B=H
9					MAR=C; wr	C=B
10					Mem=MDR	TOS=C
11						goto (MBR1)

We zitten met het probleem dat we in cyclus 3 willen beginnen maar dat gaat niet omdat MDR er nog niet staat en komt pas aan in cyclus 4. Zitten met het probleem van een reeks conditie waarbij we iets willen gebruiken dat nog moet aangevraagd moet worden. Dit is een Read After Write Dependency (andere versies bestaan) en kan niet “opgelost” worden.

57. Er bestaan twee soorten lokaliteit, dewelke en geef bij beide duidelijke en juiste voorbeelden.

Lokaliteit -> opeenvolgende adressen bevatten opeenvolgende elementen

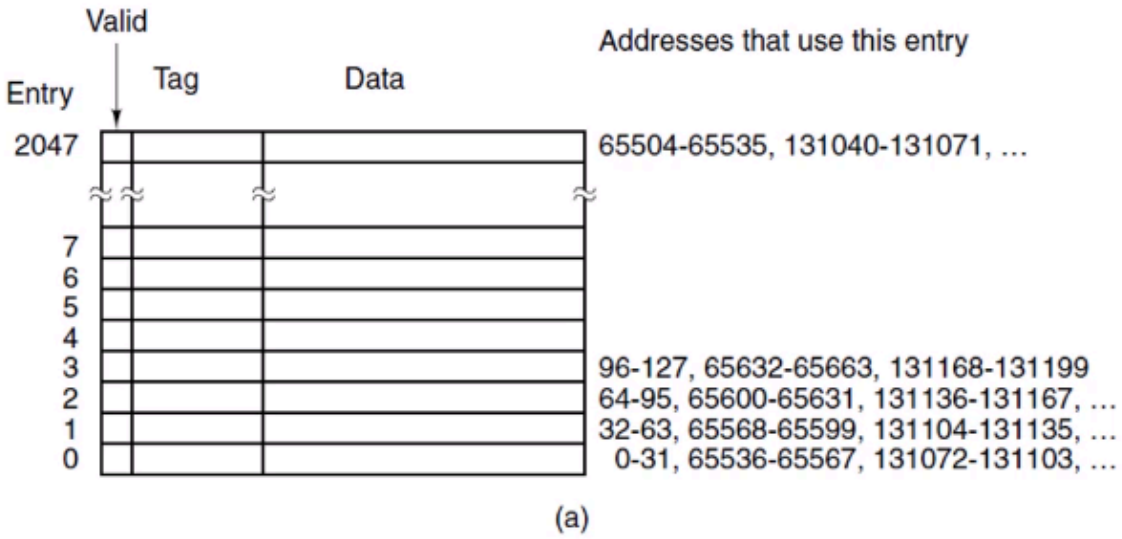
Ruimtelijke lokaliteit -> een programma, Array, Stack,....

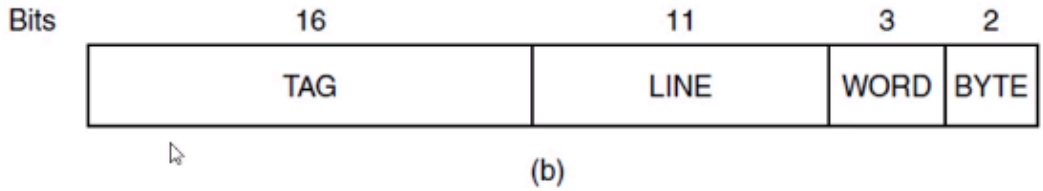
Tijdelijke lokaliteit -> In while/for lus repetitief dezelfde instructies uitvoeren

58. Veronderstel je hebt een direct mapped cache met 4096 rijen en cache lijnen van 128 bytes. Hoe kan je dan voor een 32-bit adres bepalen op welke rij in de cache je moet gaan zoeken? (de getallen kunnen worden aangepast). Wanneer zal je collisions krijgen (i.e. cache lijnen die naar dezelfde lijn verwijzen), geef de berekening van dat getal.

Cachelijn van 128 bytes: 128 = 2^7 => 7 LSB zijn offset
Totaal 4096 lijnen: 4096 = 2^12 => 12 Line bits
32-bit adres: 32-7-12 = 13 Tag bits

De 12 line bits bepalen dus op welke lijn in de cache je moet gaan zoeken. Er zal een collision optreden wanneer je bv gaat zoeken naar adres 0 en 524288(4096x128). Deze adressen hebben beide dezelfde line-bits (beide line 0), maar hun tags verschillen (resp. 0 en 1).
0000000000000 000000000000 0000000 en
0000000000001 000000000000 0000000 (=2^19=254288)





Collision -> Wanneer een adres ons naar cache lijn brengt waarvan de tags niet overeenkomen (verschillende cachelijnen kunnen leiden tot hetzelfde cache-adres).

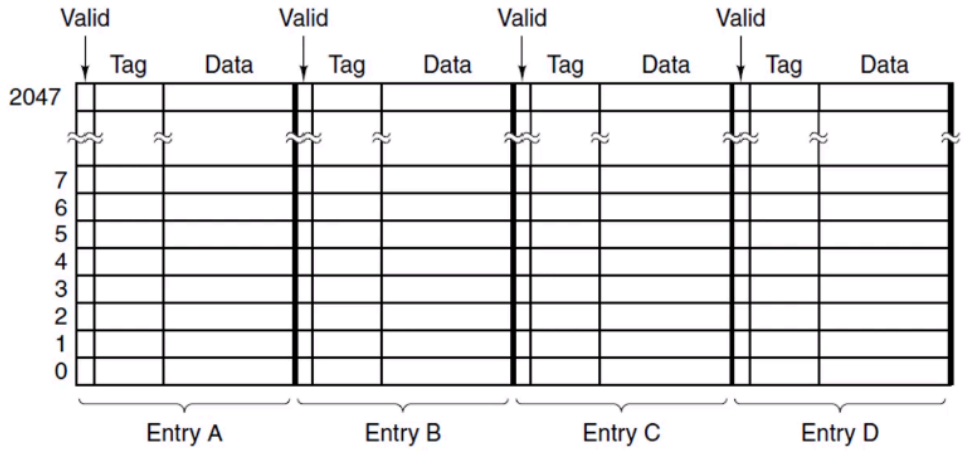
59. Veronderstel een 12-wegs set associatieve cache met 8192 rijen en cache lijnen van 64 bytes, hoe kan je dan voor een 32-bit adres gaan bepalen op welk adres je moet gaan zoeken? Wat is het voordeel van het gebruik van een 12-wegs set associatieve cache?

Pakken ons 32-bit adres kappen de 5 minst beduidende bits weg -> halen rijadres op met modulo 2^8192 -> kijken of de valid bit op 0 staat dan checken we tag (entry A) -> tag matcht niet naar volgende kolom of die valid is en of de tag matcht (entry B) -> …

Het voordeel van deze manier van werken is dat wanneer we een collision hebben, we hebben bv al iets in rij 2 van entry A staan maar de tags matchen niet. We kijken in

entry B maar daar staat valid bit op 1 -> nu kunnen we inplaats van een overwrite te doen op de vorige cache gewoon onze cache deponeren in het lege slot van entry B (kan dus ook meer cache houden).

› Set-Associative Caches



(Dit is een 4-ways)

60. Hoe kan je schrijf operaties naar cache lijnen gaan implementeren? Wat doe je voor louter schrijfopdrachten?

We kunnen dus inconsistenties creëren als we schrijfopdrachten doen wanneer we gebruikmaken van lokale kopieën.

Oplossingen:

Write Back -> Aanpassen in de cache en dan gewoon wachten tot wanneer de cache lijn verwijderd wordt en we dan een update doen van het hoofdgeheugen. Om dit te doen moeten we wel een extra bit voorzien om aan te tonen dat de originele data aangepast is (dirty flag).

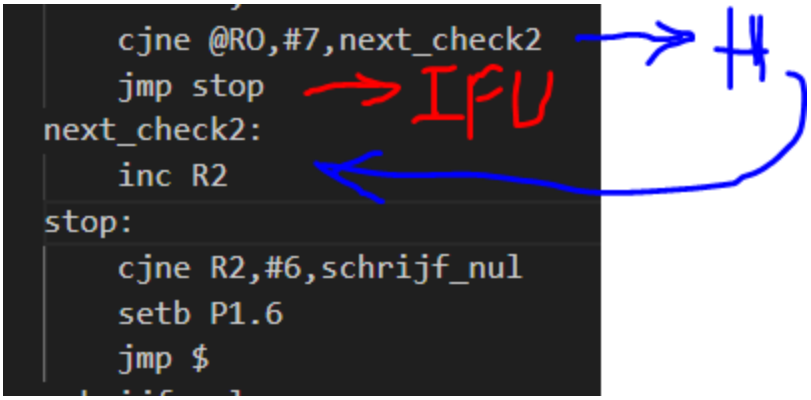
Write Through -> (eenvoudigst) bij schrijfopdracht passen de cache aan en onmiddellijk ook in het hoofdgeheugen.

Bij een louter schrijfopdracht voor data die zich niet in de cache bevindt is er geen 1 antwoord, het is een “topic for discussion”. Sommige zeggen het naar de cache te smijten andere zeggen dat dit niet de bedoeling is en so on.

61. Wat is het probleem bij de uitvoering van sprong-opdrachten? Hoe kan dat worden opgelost zonder gebruik te maken van sprong-voorspelling? Wat is de meest eenvoudige vorm van sprong-voorspelling?

Pipelining gaat ervan uit dat opeenvolgende instructies “mooi op elkaar volgen”, dat ze gewoon sequentieel in het geheugen komen. Instruction fetch units gaat volgende al ophalen terwijl de huidige nog gedecodeerd wordt en gaat er dus gewoon van uit dat volgende instructie op adressen komt die volgt op de huidige instructie maar zowel onvoorwaardelijke als voorwaardelijke sprong-opdrachten werken zo niet. Omdat we naar een ander adres richten dan het vorige adres, de ideale oplossing voor pipelining is één zonder spronginstructies (geen if, whiles, for,...). De manier om dit op te lossen zonder sprong-voorspelling is adhv van een delay sprong om 1 cyclus te wachten onmiddellijk naar een onvoorwaardelijke sprong. Dit gaat niet echt bij een voorwaardelijke sprong omdat we nog niets weten tot wanneer we aan het uitvoeren zijn (conditie waar of niet waar). De andere oplossing is de backward-forward regel -> backward branches voorspel je als te nemen en forward branches voorspel je als niet te nemen.

(Heel kort uitgelegd: ik maak sprong -> instruction fetch unit heeft al volgende instructie genomen maar ik spring naar ergens anders dus da ga niet correcte opeenvolgende instructie zijn -> problemen -> oplossen met stalling of prediction
Dus als we hier kijken zou IFU je “jmp stop” geven als volgende instructie maar we willen “inc R2” doen. (H is huidig, IFU is instruction fetch unit))

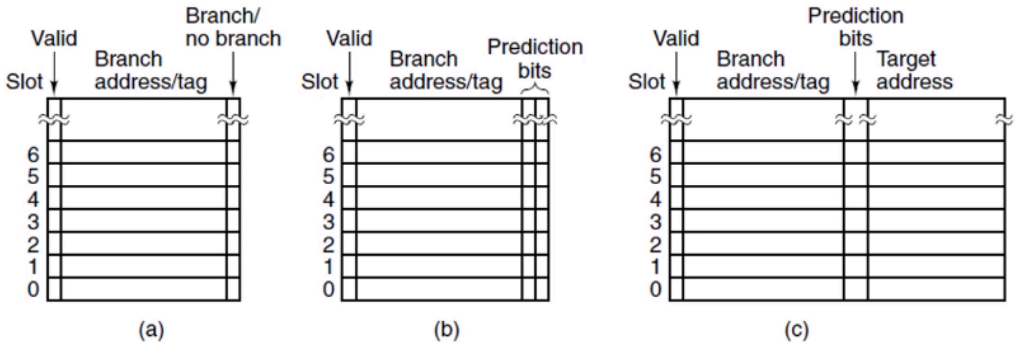


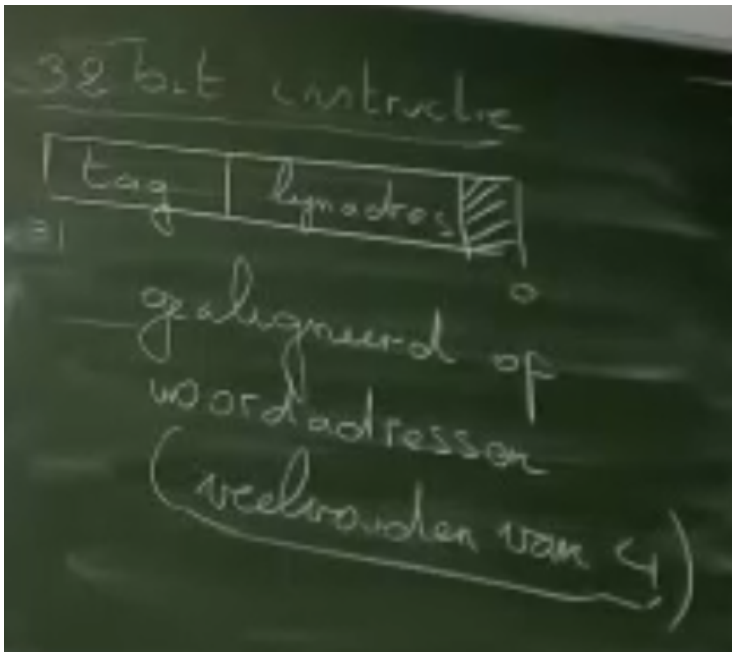
62. Veronderstel een geschiedenis tabel van 1024 lijnen en 64-bit instructie lengtes. Hoe kan je dan bepalen op welke rij in de tabel je moet gaan zoeken? (de getallen kunnen worden aangepast). Wanneer zal je collisions krijgen (i.e. instructies die naar dezelfde lijn verwijzen), geef de berekening van dat getal.

Twee minst beduidende bits wegkappen (shift right 2 posities) -> Modulo aantal rijen doen -> dan hebben we ons rijadres (en als we deling hebben, hebben we ook onze tag)

Bij een collision -> wordt gebruik gemaakt van associativiteit moest de tag mis zijn kunnen we is kijken naar de volgende entry

Dynamic Branch Prediction (1)

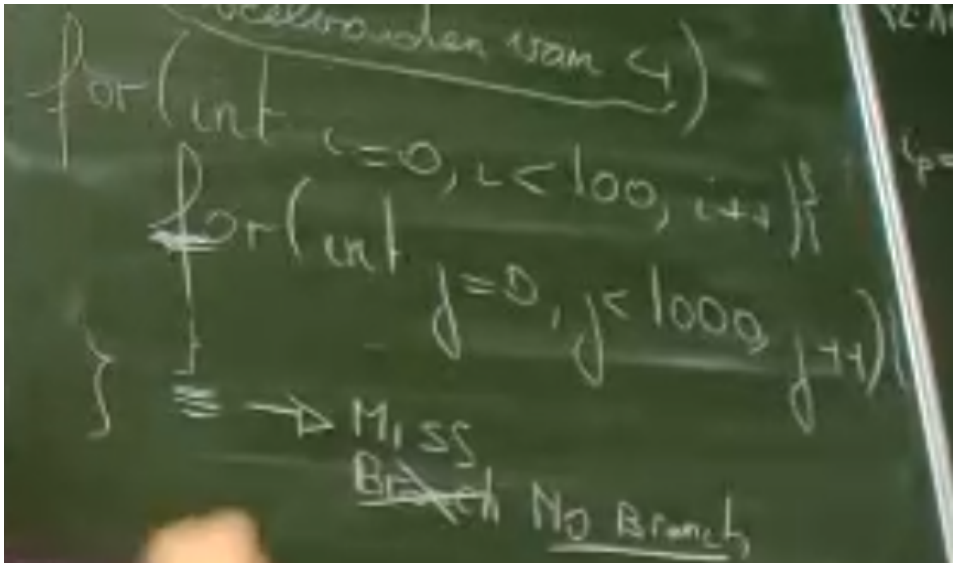




63. Waarom zijn de prestaties met twee geschiedenis bits bij dynamische sprong-voorspelling beter. Geef een concreet voorbeeld om dit aan te tonen.

Bij 1 bit -> Op de for branch en op de accolade no branch waardoor hij continu foute voorspellingen zal maken. (Hier mss niet zo erg maar pak for lus van bv: j <5)

Met 2 bits (1ste zegt wat we vorige keer gdn hebben, 2de zegt welke beslissing we nemen) -> Pas na 2 keer van gedacht veranderen. Geeft meestal beter resultaat (zeker in lussen)



64. Gegeven een superscalaire architectuur die 2 instructies tegelijk kan hebben in de execute-fase en gegeven de volgende instructies die in volgorde worden uitgevoerd:
R0=R1+R2
R1=R3*R7
R1=R2*R3
R4=R1*R6
Welke problemen kom je tegen en waar? Hoeveel cycli zal dit vergen om in volgorde uit te voeren?

We komen in de problemen met feit dat we R1 willen inlezen er naartoe willen schrijven in dezelfde cycli dit gaat niet. We moeten wachten tot wanneer onze eerste lijn klaar is vooraleer de tweede instructie lijn uitgevoerd kan worden hetzelfde geldt voor lijn 3.

Again in Cycli 2 problemen met het feit dat we zowel R1 willen lezen als naartoe schrijven en beiden gaan niet.

Ik heb geen idee hoe'k dees proper invul tbh ik snap het niet goed.

					Registers being Read								Registers being written							
Cy	#	Decoded	Iss	Ret	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7

1	1	R0=R1+R2	1			1	1						1							
	2	R1=R3*R7	-																	
2	3	R1=R2*R3	-																	
	4	R4=R1*R6	-																	
3																				
4																				

65. Gegeven een superscalaire architectuur die 2 instructies tegelijk hebben in de execute-fase en gegeven de volgende instructies die uit volgorde worden uitgevoerd:
R0=R1+R2
R1=R3*R7
R1=R2*R3
R4=R1*R6
Welke problemen kom je tegen en waar? Hoeveel cycli zal dit vergen om in volgorde uit te voeren?

We komen in de problemen met feit dat we R1 willen inlezen er naartoe willen schrijven in dezelfde cycli dit gaat niet. We moeten wachten tot wanneer onze eerste lijn klaar is vooraleer de tweede instructie lijn uitgevoerd kan worden hetzelfde geldt voor lijn 3.

Again in Cycli 2 problemen met het feit dat we zowel R1 willen lezen als naartoe schrijven en beiden gaan niet.

					Registers being Read								Registers being written							
Cy	#	Decoded	Iss	Ret	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
1	1	R0=R1+R2	1			1	1						1							
	2	R1=R3*R7	-			1	1	1				1	1							
2	3	R1=R2*R3	-			1	2	2				1	1							
	4	R4=R1*R6	-			2	2	2			1	1	1				1			

3							2	2	2			1	1	1				1			
4				1			1	1	2			1	1					1			

66. Waarom is het zo dat de meeste systemen toelaten dat instructies in volgorde moeten stoppen?

Dit is omdat als we een interrupt doen we precies weten waar we gekomen zijn in de uitvoering van een programma. (gaat over ISA-level instructies, noemt precise interrupts)

67. Gegeven onderstaand codefragment (niet noodzakelijk dezelfde code):

```
void verdubbel_indien_gevonden(register int *tab, register int n,
volatile int *getal){
        register int i=0;
        while (i<n && tab[i]!=getal)
                i++;
                if (i==n)
                        *getal*=2;
}
```

Waarom is het onverstandig om *getal*=2 naar voor te schuiven bij uitvoering? Welke problemen kan je tegenkomen en hoe kunnen ze opgelost worden?

rond 1:13:0 labo 24 november

68. Wanneer is het handig om poison bit bij een register te voegen? Geef een voorbeeld waar dit gebruikt wordt.

69. Bij de i7-pipeline wordt er gebruikgemaakt van store-to-load forwarding? Wat is dit en bespreek.

Store-to-load fowarding: Voorgangers van de micro-operatie die een schrijfoperatie wil doen naar de

level 1 datacache gaan gewoon lezen uit de level 1 datacache. Opvolgers van de micro-operaties die willen lezen halen hun data uit de buffer.

Hoofdstuk 5

70. Wat is geheugenalignering (geef een codevoorbeeld) en waarom is dit zo belangrijk?

```c
#include <stdio.h>

typedef struct {
        int getal;
        char c;
} test;

int main(){
        test t1,t2;
        t1.getal=7;
        t1.c='a';
        t2.getal=8;
        t2.c='b';
        return 0;
}
```

```
main:
        push    ebp
        mov     ebp, esp
        sub     esp, 16
        mov     DWORD PTR [ebp-8], 7
        mov     BYTE PTR [ebp-4], 97
        mov     DWORD PTR [ebp-16], 8
        mov     BYTE PTR [ebp-12], 98
        mov     eax, 0
        leave
        ret
        .size   main, .-main
        .ident  "GCC: (GNU) 10.3.1 20210422 (Red Hat 10.3.1-1)"
        .section        .note.GNU-stack,"",@progbits
```

Ook al neemt onze struct maar 5 bytes in, nemen we 8 om te vermijden dat we niet in veelvoud van 4 zouden zitten. T'zit hem in het feit dat we gebruik maken van een

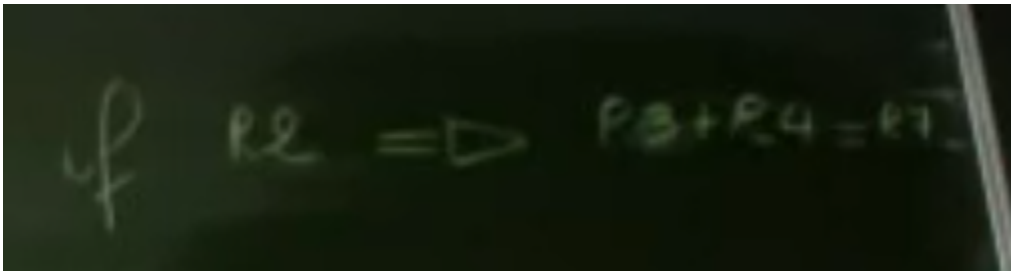
32-/64-bit databus en moesten we dit niet doen riskeren we altijd het feit dat we 2 lees-operaties zouden moeten doen (data kan dan zitten op bv 2 cachelijnen).

Hoofdstuk 8

71. Wat is het idee achter een VLIW-CPU? Waarom presteert dit toch ondermaats?

Leggen verantwoordelijkheid bij de compiler om bundels aan te maken en out of order execution te doen (in compile-time dus), terwijl dit bij andere processoren op runtime gebeurt (out of order execution. Hierdoor presteert het ondermaats omdat niet alles geweten is in compile-time en omdat het er niet altijd in slaagt om alle functionele eenheden van een waarde te voorzien.

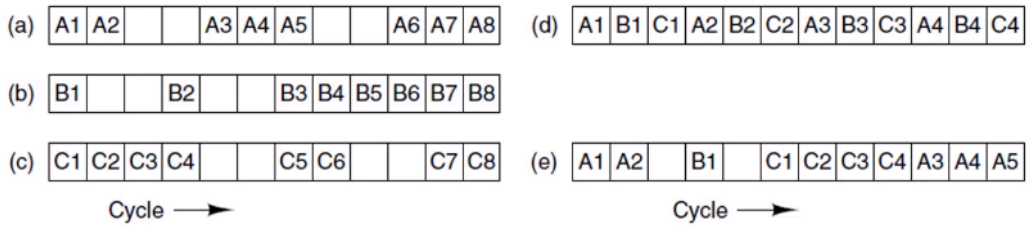
72. Wat is predicated execution en wat is het voordeel?



Altijd een voorwaarde laten voorafgaan, aka iedereen instructie begint met if-testje. Het voordeel hiervan is dat we veel minder jumps gaan moeten doen om bepaalde code te verwezenlijken.

73. Wat is fine-grained multithreading. Geef een voorbeeld met 4 threads en leg uit waarom dit niet altijd bruikbaar is?

› On-Chip Multithreading (1)



Fine-grained multithreading is niets meer dan gewoon round robin afwisselen. Dus elke keer wisselen we af van thread (eerste instructie van thread A dan eerste van thread B, …).
Er is wel een probleem hierbij: Pipeline met k stages heeft behoefte aan k threads het probleem hierbij is dat de kans klein is dat we bezitten over de benodigde hoeveelheid threads.

74. Wat is coarse-grained multithreading en leg uit hoe dit werkt met 4 threads. Hoe kan dit eventueel nog iets worden geoptimaliseerd waardoor een dode cyclus kan worden overgeslagen?

Coarse-grained multithreading: Voer uit tot wanneer je problemen tegenkomt en verwissel dan van hardware thread.
Optimalisatie: Zodra een instructie is opgehaald gaat hij al wisselen in het geval dat hij problemen verwacht van die instructie. Bv bij een Load operaties gaat hij niet riskeren dat hij bv CacheMiss zal hebben en zal gwn al wisselen maar zal dit dan waar niet doen bij bv een maal opdracht.

75. Welke bronnen moeten bij multithreading sowieso worden gedupliceerd?

Programmateller, Geheugenmap (voor het regelen naar welke scratch register je mag schrijven en ophalen) (/register bestand) , Interrupt controle en interrupt afhandeling.

76. Wat is full-resource sharing, partitioned sharing en threshold sharing? Geef telkens voor- en nadelen.

Full-resource: First come first served ⟶ De eerste die resources nodig heeft pakt ze ook. Nadeel hiervan is natuurlijk dat als we een trage thread hebben en een snelle thread.

De snelle zal continu alle resources pakken en hierdoor zal de instructie queue vollopen met allemaal instructies van de traagste thread. Hij wordt dus ook bottleneck van het systeem.

Partitioned: Resources meestal delen in twee. Bv: de helft mag gebruikt worden door thread A, andere helft door thread B. Nadeel hiervan is dat een partitie een vaste lijn is, dus indien een thread meer resources nodig heeft kan dat niet.

Het kan dus goed zijn dat thread A nog resources over heeft en thread B meer nodig heeft dan hij ter beschikking heeft maar dat is nu eenmaal het nadeel van het systeem.

Threshold: Dynamische grens (meet in the middle tussen partitie en full-resource)

Hoofdstuk 6

77. Aan welke vier randvoorwaarden moet ieder geheugenbeheersysteem voldoen?

78. Bespreek de werking van paginering (zonder virtueel geheugen). Wat is het verschil tussen paginering en vaste partitionering? Hoe gebeurt de adresvertaling bij paginering (maak een schets)?

79. Bespreek de werking van segmentatie (zonder virtueel geheugen)? Wat is het verschil tussen dynamische partitionering en segmentatie? Waarom wordt dit model voor de gebruiker bewust zichtbaar wordt gehouden. Geef een voorbeeld waar je handig gebruik kan maken van segmenten.

    Hoe gebeurt de adresvertaling bij segmentatie (maak een schets)?

80. Bespreek de stappen die moeten ondernomen worden wanneer bij adresvertaling wordt vastgesteld dat een deel van het proces zich niet in het geheugen bevindt?

81. Wat zijn de drie voordelen van het gebruik van virtueel geheugen? Wat is het nadeel van het gebruik van virtueel geheugen?

82. Welke twee parameters moet het besturingssysteem in de gaten houden om te zien of er bij het gebruik van virtueel geheugen te veel dan wel te weinig paginafouten optreden? Wat wordt er bedoeld met “thrashing”? Hoe kan het besturingssysteem oordeelkundig inschatten welke pagina’s in de toekomst nodig zullen zijn en welke niet?

83. Wat is de ideale grootte van een pagina? Maak de afweging tussen de voor- en nadelen van kleine en grootte pagina’s en waarom men eigenlijk opteert voor paginagroottes van enkele kilobytes.

84. Wat is het verschil tussen een pagina-ingang bij een systeem met paginering zonder en met virtueel geheugen.

85. Wat is het probleem waarom men opteert voor systemen met dubbele paginering en een geïnverteerde paginatabel?

86. Waarom heeft de MMU een TLB? Hoe werk dit?

87. Wat is de werking van een systeem met een geïnverteerde paginatabel?

88. Wat is de reden waarom systemen met virtueel geheugen niet meer uitsluitend gebruik maken van segmentatie? Men lost dit op door gebruik te maken van systemen die gebruikmaken van paginering en segmentatie. Maak een schets hoe de adresvertaling hier verloopt.