COMPUTERSYSTEMEN 1

Axel Hamelrijck

EXAMEN JANUARI 2019 UCLL TOEGEPASTE INFORMATICA

Beste medestudenten,

Jullie mogen meegenieten van deze samenvatting die ik heb gemaakt,
maar ik neem geen verantwoordelijkheden op voor slechte punten of voor onvolledigheid.

Vergeet natuurlijk jullie oefeningen niet te hermaken op Edublend en de oefenexamens te bekijken.
Probeer de oefeningen van de examens (zie Toledo) ook eens te maken op papier, zo wordt het op het examen gevraagd.

Ik leg de oefeningen uit op de manier hoe ik het begrijp.

Ik wens jullie verder nog veel succes met de examens.

Donaties zijn altijd welkom voor mijn harde werk:

https://paypal.me/axelele

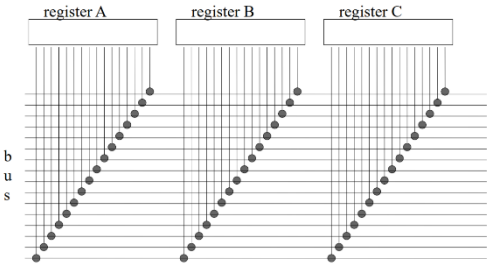
Hoofdstuk 1: Computer en gegevens

Hardware

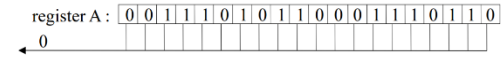
- Bits → **0 of 1**
- Registers → stukjes **geheugen in processor** van bepaalde **lengte (macht van 2) --> aantal bits**

Registers

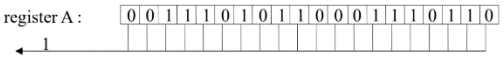
- transport tussen registers
  - Directe verbinding → moeilijk → busverbinding
  - Parallelwerking: **(duurder & sneller)**
    - Voor iedere bit is er een andere geleider



  - Seriewerking: **(goedkoper & trager)**
    - Eén geleider → elke bit 1 per 1



Laatste bit;



Voorlaatste bit;

Binaire voorstelling van integers

- Decimaal
  - 10 cijfers → 0,1,2…9
- Binair
  - 0,1
  - > 1 → meerdere bits achter elkaar
  - → 10011 = **00010011** = 1x2 4 + 0x2 3 + 0x2 2 + 1x2 1 + 1x2 0 = (19)d
    - Leidende nullen tellen ook

Omzetting binair naar decimaal

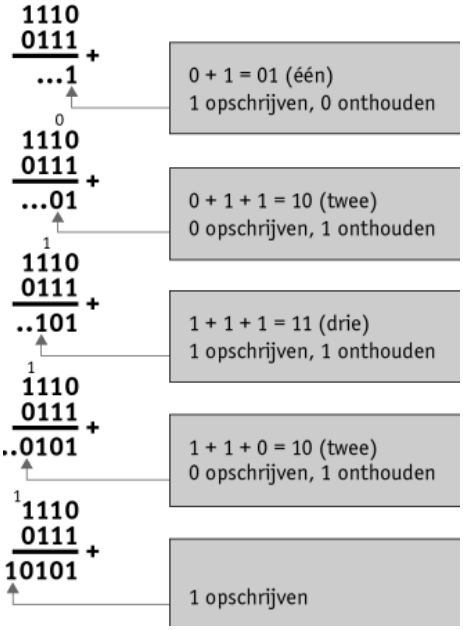
10011 → 1x2 4 + 0x2 3 + 0x2 2 + 1x2 1 + 1x2 0 = (19)d

Omzetting decimaal naar binair

- Omgekeerd → 131 =

128	64	32	16	8	4	2	1
1	**0**	**0**	**0**	**0**	**0**	**1**	**1**

Binaire optelling



Binaire vermenigvuldiging met 2

2x **10111** ➔ 2x (1x2 4+0x2 3+1x2 2+1x2 1+1x2 0) ➔ 1x2 5+0x2 4+1x2 3+1x2 2+1x2 1+0x2 0 ➔ **101110**

- ⇨ x2 ➔ achteraan 0 toevoegen

Hexadecimaal stelsel

16 cijfers ➔ 0 – 1 – 2 – 3 – 4 – 5 – 6 – 7 – 8 – 9 – A – B – C – D – E – F

Omzetten hexadecimaal naar binair

Eerst omzetten naar binair ➔ dan naar decimaal

Omzetten binair naar hexadecimaal

Elke 4 bits (vanachter beginnen)→ 1 hexadecimaal getal →

d.	h.	b.	d.	h.	b.
0	0	0000	8	8	1000
1	1	0001	9	9	1001
2	2	0010	10	A	1010
3	3	0011	11	B	1011
4	4	0100	12	C	1100
5	5	0101	13	D	1101
6	6	0110	14	E	1110
7	7	0111	15	F	1111

Hexadecimale optelling

- ⇨ **1 optellen** bij het getal
  - Zoals bij **binair** (onthouden)

Negatieve getallen (binair en hexadecimaal)

- In decimaal: + en –
- **In binair: 0 en 1**
  - **- → 1**
  - **+ → 0**
    - **op de eerste bit**
- **Context!** → context is belangrijk om te weten of het getal negatief zou zijn of niet
- Met negatief → **2-complementsnotatie**
  - Hogere programmeertaal laat je kiezen welke notatie (uint vs **int** bv)

*Omzetten negatief decimaal naar binair (**vb -9**)*

1. Binaire omstelling van absolute waarde (**001001**)
2. Complement nemen **(1 in 0 veranderen en vice versa)** (**110110**)
3. 1 bijtellen aan dat complement (**110111**)

*Omzetten negatief binair naar decimaal (**101110**)*

1. Complement nemen (**010001**)
2. 1 bijtellen aan dat complement (**010010**)
3. Omzetten naar binair (**18**)
4. – voor zetten (**-18**)

Binaire voorstelling van tekst

- **ASCII** → American Stantard code for information interchange
- Per karakter → 8 bits (1 byte) gebruikt
  - A ➔ 01000001
  - A ➔ 01100001
  - …

Betekenis inhoud van byte (vb C3 ➔ 1100 0011)

1. Niet kunnen weten
2. 1100 0011 ➔ 195 (zonder negatief)
3. 1100 0011 ➔ -61 (met negatief)
4. ➔ aan de **inhoud van een byte** kan met **NIET zien** wat de **betekenis** is

- Teveel verschillende tekens (â, é, è, ü,…)➔ ondersteuning nodig
- **Unicode** ➔ ondersteuning voor meer karakters
  - UTF-16 ➔ 2 bytes per karakter
  - UTF-32 ➔ 4 bytes per karakter
  - UTF-8 ➔ variable aantal bytes per karakter

- Opeevolging van karakters ➔ **string**
- "Het Spaanse graan heeft de orkaan ...“ ➔ 48 65 74 20 53 70 61 61 6E 73 65 20 ...
- Komt als **ASCII** in geheugen ➔ kan **NIET** mee worden gerekend

- Een getal intypen ➔ ASCII voorstelling van getal komt in geheugen ➔ NIET de binaire voorstelling

Hoofdstuk 2: Werkgeheugen, centrale verwerkingseenheid, programma's

Werkgeheugen

- Elke byte ➔ uniek adres (altijd positief – **van 00000000 tot FFFFFFFF**)
- Maximale grootte werkgeheugen ➔ bepaald door grootte van het adres
- Groeperen
  - 1 byte ➔ 256 verschillende getallen
  - 2 bytes ➔ 65.536 verschillende getallen ➔ **woord**
  - **4 bytes** ➔ 4.294.967.296 verschillende getallen ➔ **dubbelwoord**
    - int in Java → dubbelwoord → -2.147.483.648 tot 2.147.483.647
- 2 manieren om adressen aan te duiden
  - **Adres** van (**dubbel**)**woord** is adres van **eerste byte** van dat **(dubbel)woord**
    - ➔ **absolute adres** ➔ **adres van byte / dubbelwoord**
  - Combinatie van basis en verplaatsing
    - Adres ➔ basis + verplaatsing
      - Notatie: (basis:verplaatsing)
      - Vb: 012A3C4D ➔ (012A3C00:0000004D)

Centrale verwerkingseenheid (CPU): bevelen

- Doet dingen ➔ bevelen die een achter een worden uitgevoerd ➔ met data (EAX, EBX, ECX, EDX,… ➔ registers)
  - Uitvoering ➔ data in registers / werkgeheugen verandert
- Bevel bestaat uit 2 delen
  - Kopieer naar EAX, tel op bij EAX, kopieer inhoud EAX naar, …
    - **Functiecode**
  - De inhoud van een dubbelwoord in het werkgeheugen of een register
    - **Operand**
- **Memotechnische functiecodes** ➔
  - **mov** eax, [200h]
  - **add** eax, [240h]
  - **sub** [300h], eax
    - h = **hexadecimaal**
    - eax = EAX
    - Adres tussen [en]
    - **Linkse parameter** krijgt **resultaat**
- Inv (getal opvragen) & uit (getal weergeven) ➔ **geen** echte bevelen – gemaakt voor UCLL

Vertaalprogramma vs assembleerprogramma

- Programma in machinetaal wordt **geassembleerd** door het assembleerprogramma (E.: **assembler**)
  - **1 bevel** voor de programmeur is **1 bevel na vertaling**
- Programma in hogere programmeertaal wordt **gecompileerd** door een compilator (E.: **compiler**)
  - **1 bevel** voor de programmeur is 1, 2, ... **100 bevelen na vertaling**

Programma's schrijven

- Begint altijd met
  - %include "gt.asm"
    covar
    inleiding
- Eindigt met:
  - slot
- plaats van het adres een naam schrijven ➔ symbolisch adres
  - x: resd 1
    - resd 1 ➔ 1 dubbelwoord reserveren voor dat adres
    - resw 1 ➔ woord
    - resb 1 ➔ byte
    - dd 1 ➔ constant
- ➔ alleen maar symbolische adressen tijdens oefeningen

Constanten

- "een: dd 1"
  - Symbolisch adres → "een"
  - Inhoud dubbelwoord is 1
- Plaats in het geheugen met een **voorgedefinieerde** waarde
  - **Wel inhoud** tijdens de vertaling

Veranderlijke

- "hulpd: resd 1"
  - Symbolisch adres → "hulpd"
- Plaats in het geheugen voor een **tussenresultaat**
  - **Geen inhoud** tijdens de vertaling

Vermenigvuldigingen

- getal1: dd 7
  getal2: dd 5
- mov eax, [getal1]
  imul dword [getal2]
  - → inhoud van **eax** wordt **vermenigvuldigd** met de inhoud van [getal2]
  - eax → 35
    - alleen de **inhoud** van **eax** kan vermenigvuldigd worden
  - Resultaat van het product komt in het **registerpaar** (**edx**, **eax**)
- Resultaat → **64 bits**
  - Eerste helft in **edx** → vaak leeg (00000000)
- Imul **dword** → (32 bits) * (32 bits)
  - Imul **byte** → (8 bits) * (8 bits)
  - Imul **word** → (16 bits) * (16 bits)

Deling

- nul: dd 0
  deeltal: dd 85
  deler: dd 3
- mov edx, [nul]
  mov eax, [deeltal]
  idiv dword, [deler]
  - **inhoud** van **registerpaar (edx en eax)** wordt als één getal **gedeeld**
    - **edx** → moet altijd 0 zijn of het neemt die inhoud mee als getal
  - **quotient** → eax
  - **rest** → edx

Deling negatief getal

- In EDX, EAX moet **correcte voorstelling** van deeltal staan
  - 00 00 00 00 → deeltal positief
  - FF FF FF FF → deeltal negatief
    - → eerst **vermenigvuldigen** met 1
- een: dd 1
  deeltal: dd -20
  deler: dd 3
- mov eax, [deeltal]
  imul dword [een]
  idiv dword [deler]
  - door de **imul dword** wordt het **hele negatief getal** in (**EDX**, **EAX**) geplaatst
  - **zelfde** als **gewone** deling

Sprongbevelen

- Kracht van computer → **herhaling**
- **haha**: mov eax, [x]
  add eax, [een]
- mov [hulpd], eax
  jmp haha
  - **haha** → symbolisch adres van een bevel

Voorwaardelijke sprong

- Assembly →
  - **Vergelijken** (is x > y?)
  - Dan: **spring** als "**het**" **groter** is
  - Geen **else** stuk (in vergelijking met Java (if… else…)
- **Vlaggen** worden ingesteld
  - **Bit in de CPU** →
    - Nul vlag (ZF)
    - Teken vlag (SF)
    - Overloop vlag (OF)
  - **Stelt toestand** na een bepaalde instructie **voor**
  - **Add, sub** → resultaat wordt vergeleken met nul
    - **Resultaat = 0 → ZF = 1**
    - **Resultaat < 0 → SF = 1**
    - **Overloop → OF = 1**
  - Twee getallen vergelijken → **cmp**
    - Getal1: resd 1
      getal2: resd 1
    - Mov eax, [getal1]
      cmp eax, [getal2]
      - Achter de **schermen** wordt **getal2** van **getal 1 afgetrokken**
      - **Resultaat** → bepaalt de **vlaggen**
      - Getal1 = getal2 → **ZF = 1**
      - Getal1 < getal2 → **ZF = 0 en SF =/= OF**
      - Getal1 > getal2 → Z**F = 0 en SF = OF**
    - Wij programmeren zo niet

Instructies: jg (jump if greater)

- Springt indien getal1 > getal2 (het eerste t.o.v. het tweede)
  - → ZF = 0 en SF = OF
- **je … jne (equal … not equal)**
- **jl … jnl (less … not less)**
- **jg … jng (greater … not greater)**
- **jnl = jge (greater or equal)**
- **jng = jle (less or equal)**
- **jo … jno (overflow … not overflow)**
  - **Eerst vlaggen instellen** → add, sub, cmp
    - **Dan voorwaardelijke** sprong naar symbolisch adres

Uitvoering van programma's

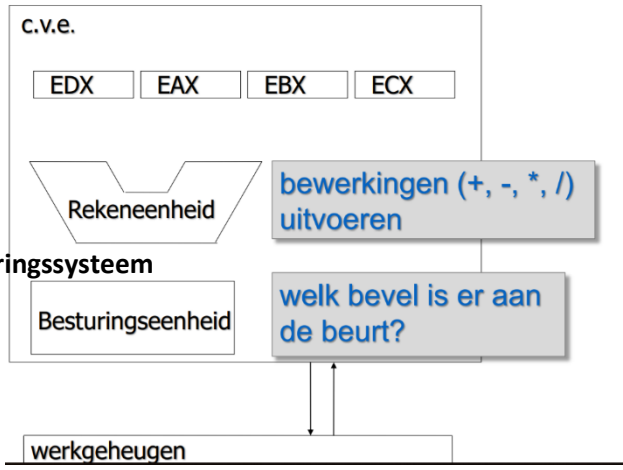
- A1 9C 01 00 00 ➔ mov eax, [alfa]
- F7 2D A8 01 00 00 ➔ imul dword [mu]
- 03 05 A0 01 00 00 ➔ add eax, [beta]

→ worden omgezet door **vertaalprogramma**

  - IDE (assembly) → ASCII → exe (hexadecimal)

Assembler

- Vervangt symbolische adressen
  - [alfa] → 00 00 01 9C
- Vertaalt **ASCII voorstelling** van **bevelen**
  - 6D 6F 76 20 65 61 78 2C 5B 61 6C 66 61 5D 0D 0A
    - ➔ wordt A1 9C 01 00 00
- Vertaalt **constanten** en **veranderlijken**
  - 62 65 74 61 3A 20 64 64 20 2D 35
    - ➔ wordt FF FF FF FB
- Schrijft een .**obj bestand** weg
- Programma (.exe) wordt gemaakt door de **linker**
  - **Combinatie van verschillende object-bestanden**
- Programma wordt in **werkgeheugen** geladen door **besturingssysteem**
  - **Constanten** en **verandelijken** op een locatie
  - **Programmacode** op andere locatie
    - → OS kiest deze locaties



Besturingseenheid

Bevat twee registers →

- **Bevelregister**
  - Bevat bytes van het **bevel** dat **uitgevoerd** moet worden
- **Bevelenteller**
  - Bevat adres van **volgende bevel** (**instruction pointer of** EIP**)**

- **Haalcyclus**
  - **Volgend bevel** ophalen
  - **Bevelenteller aanpassen** (lengte van opgehaald bevel bij optellen)
- **Uitvoercyclus**
  - **Bevel analyseren** (wat moet er gebeuren)
  - **Ervoor** zorgen dat het **bevel** gebeurt

Werking bevelenteller en bevelenregister zonder sprongbevelen (oefeningen)

- Bevelenteller optellen met aantal bytes van het bevelenregister
- Verder tellen van het einde van vorige bevelenregister → nieuwe bevelenregister
- Zo opnieuw
    - Dit deel is duidelijker via de oefeningen op Edublend (Hoofdstuk 2 → Sprongbevelen (werking) en met de dia's (dia 77 – 86)

Werking bevelenteller en bevelenregistern met sprongbevelen (oefeningen)

- Zelfde werking! → gewoon met of zonder sprong erbij zetten
    - Zonder sprong → bevel na de jump
    - Met sprong → bevelenteller wordt destinatie van de jump
- Zie oefeningen Edublend

Men kan maar 1 byte springen ➔ van -128 tot 127

Bevelenwachtrij

Tussen werkgeheugen en CPU ➔ verbinding

- Wordt gebruikt om data te kopiëren
    - Van register naar uitkomst
        - mov [uitkomst], eax
    - van werkgeheugen naar register
        - mov eax, [uitkomst]
    - één van de operanden bij berekening
        - imul dword [factor]
    - **de bevelen zelf**
- Verbinding → traag ➔ nooit 2 dingen tegelijk
- Bv. 32 bits breed (kan+ of – zijn)

Traag ➔ het mov-bevel wordt opgehaald tijdens de uitvoering van de optelling (verbinding is dan leeg)

Rekeneenheid

- Arithmetic and Logical Unit (ALU)
- Uitvoering van bewerkingen (+, -, *, /, AND, OR,…)
- 2 ingangen, 1 uitgang
- Krijgt bevel van besturingseenheid

Klok

Produceert met perfecte regelmaat spanningspulsen (een puls = een tik)

- ♣Bvb. 3.700.000.000 cycli per seconde, d.i. een klok van 3.7GHz

Uitvoering van bevel → in stappen, volgens het "tikken" van de klok

1. eax, edx kopiëren op (verschillende) bus
2. Getal van elke bus kopiëren naar één ingang van A.L.U.
3. + bevel naar A.L.U.
4. uitgang A.L.U. kopiëren naar bus
5. bus kopiëren naar eax

Klok → dirigent

Voor elk bevel → bepaald aantal klokcycli nodig

- sprong-bevel: 1 cyclus
- optelling/aftrekking: 3 cycli
- vermenigvuldiging: 18 cycli

Snellere klok → snellere uitvoering van bevelen

Hoofdstuk 3

Hoeveel bits?

- **Werkgeheugen** →
  - opgedeeld in **geheugencellen** (bv. 8 bits, kan meer of minder zijn)
  - **Grootte** van cel: **resolutie** van geheugen
- **Geheugencel** → **uniek adres** en **inhoud**
  - **Lengte** van adres → **bepaalt** aantal **cellen**

Basis en verplaatsing

- Programma's → **relatieve adressen** = **basis** + **verplaatsing**
- Machinebevelen → enkel verplaatsingen
- Tijdens uitvoering → absolute adres bepaald

Segmenten

- Aantal **opeenvolgende bytes** die logisch **achter elkaar** hoort → **segment**
- Segment adres → basis van segment (adres van eerste byte)
- **Codesegment** (bevelen), **datasegment** (data), …
- **Veel programmas** in **geheugen**

Totale grootte van **programmas** kan **>** dan **werkgeheugen** → **inactieve programma's** naar **harde schijf**

- Elk segment → **descriptor**
  - Eigenschappen van segment
    - Grootte
    - Staat in geheugen
    - Basisadres
  - 8 bytes groot
  - Worden opgeslagen in **descriptorentabel** (in cache van CPU)
    - plaats voor **8192 descriptoren**

- Elk descriptor → **segmentselector** → uniek nummer van (0000)h tot (1FFF)h
  - Bewaard in **segmentregisters** in CPU
    - **Intel** → 6 segmentregisters van 2 bytes groot
      - CS, DS, SS, ES, FS, GS

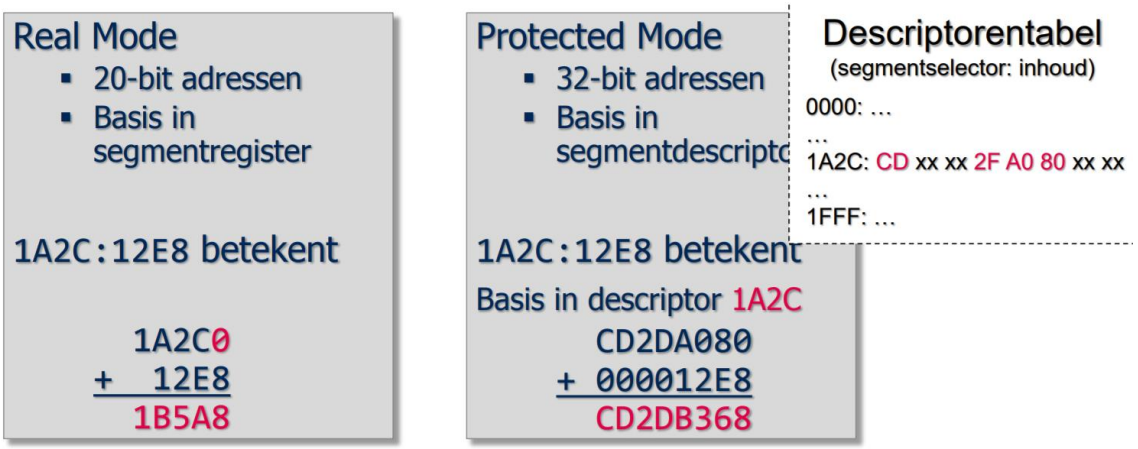
Segment met verplaatsing 00002DA4 en segmentselector 02AC → 02AC:00002DA4

Basisadres → **descriptor** → **descriptorentabel** → locatie aangegeven door **segmentselector**

- **Intel 8086 CPU** → werkgeheugen van 1MB
  - **Adressen → 20 bits**
  - **Verplaatsingen → 16 bits**
- **Basis → opgeslagen** in **segmentregisters**
  - **CS** voor bevelen
  - **DS** voor data
  - **Geen** segmentselectors / descriptoren / descriptorentabel

- Basis van 20 bits → 16 bit register?
  - Elk basisadres deelbaar door 16
    - **Laatste 4 bits 0**
    - Enkel 16 eerste bits opslaan in segmentregister
- Voorbeeld:
  - Inhoud DS → 3A2C
  - Verplaatsing → 12E8
  - Uiteindelijk → 3A2C0
  - \+ 12E8
  - 3B5A8

➔ **Real mode** ➔ wordt nog ondersteund bij opstarten → later overstappen naar **protected mode**



Adreswijziging

- Adressen → gewijzig worden met **indexregisters**
  - Bv. **ESI en EDI**
- Inv [getal + edi]
  - Berekend adres ➔ getal + (inhoud EDI]
  - Programmeur bepaalt inhoud **indexregisters**
- **Voorbeeld programma:**
- sub eax, eax
  mov ecx, 10
  sub edi, edi
  hoger: **cmp ecx, 0**
  jle verder
  inv [getal + **edi**]
  add eax, [getal + **edi**]
  add edi, 4
  sub ecx, 1
  jmp hoger
  verder: …
- ➔ **oefeningen op edublend**

Rijen

- **Belangrijkste gegevensstructuur**
  - Gemiddelde temperatuur voor elke dag
  - Uitslagen van een student
  - Uitslagen punten op een vak
- In hoge programmeertaal ➔ **array**
- **Rij =/= verzameling**
  - **Orde** → elk element heeft opvolger buiten het laatste

Loop-bevel

Werking van oefeningen adreswijzigingen

➔ Inhoud ecx verminderen en vergelijken met 0

- move ecx, 10
  lus: …
  ..
  loop lus

Karakterstrings

Stringdefinities en bevelen

Constanten (DEFINE byte)

- vb. antwrd: db ‘ja’
- geheugen → **ascii-waarde** v.d. **constante** → 4A 41
- vertaalprogramma kent ‘antwrd’

Variabelen (REServation)

- vb. gegeven: RESB 20
- vertaalprogramma kent ‘gegeven’
- in geheugen → ?? ?? ?? … ?? ?? ?? → (20 bytes)

Stringbevelen

- x86 →
- movsb → move string element byte
- rep movsb → rep = repeat
- cld en std → clear direction / set direction
- stosb → store string element byte
- rep stosb → rep = repeat
- lodsb → load string element byte

movsb

- 1 byte kopiëren
  - van verplaatsing in **ESI** (S = **source**)
  - naar verplaatsing in **EDI** (D = **destination)**
- Verschil mov?
  - 4 bytes kopïeren
  - Van **register** naar **dubbelwoord** –of– van **dubbelwoord** naar **register** –of– van **register** naar **register**

ESI en EDI

- Mov esi, b
  - → verplaatsing van b wordt in esi geplaatst
- Mov edi, a
  - → verplaatsing van a wordt in edi geplaatst
- **Nooit 2 adressen in 1 bevel**
  - Bij mov altijd register
  - Movsb wel **2 geheugenadressen** nodig (verplaatsingen die werden gezet in **edi** en **esi**)

Van links naar rechts of van rechts naar links kopiëren?
➔ richtingsvlag zetten
→ DF=0 (na cld) → van voor naar achter
→ DF=1 (na std) → van achter naar voor

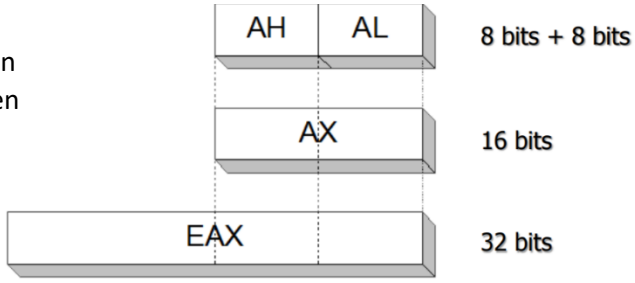
Veel bytes kopiëren

```
cld
mov esi, b
mov edi, a
mov ecx, 6
rep movsb
```

➔ rep movsb repeat het aantal dat in ecx staat

Stosb

- 1 byte kopiëren
  - Van inhoud 'al'
  - Naar EDI
- Inhoud EDI aanpassen
  - Indien DF=0: met 1 vermeerderen
  - • Indien DF=1: met 1 verminderen



Rep stosb

Zelfde als rep movsb (ecx aantal)

Registers

AL → deel van EAX

Deel van string bewerken

```
string: resb 10
…
mov edi, string + 3
mov al, '*'
mov ecx, 5
rep stosb
```

Adres: **string**

?? ?? ?? 2A 2A 2A 2A 2A ?? ??

Lodsb

Van geheugen naar AL (omgekeerde van stosb)

Strings afdrukken

Niet via uit, via **schrijf**

Keuzes:

- Uitvoer naar **1 bestand**
- Eenheid: 1 lijn
- Aantal tekens **ligt vast 70** (+2)
- Andres ligt ook vast: **outarea**

➔ Eerst bestand creëren

- **openuit** → schrijf
- Welke string?
  - Vanaf adres outarea
- Hoeveel?
  - 70 + 0D 0A → 72 bytes
  - **Outarea** moet gedefinieerd zijn (uitvoerzone)

Begin van programma →

- covar
  outarea: **resb** 70
  db 0Dh, **0Ah**
  vb1: db 'Voorbeeld'
- inleiding
  openuit

70 bytes vanaf **outarea** → waar de **strings** komen → **worden gekopieerd** naar bestand
Bytes vanaf **outarea** → **uitvoerzone**

Besluit

outarea: **resb** 70
db 0Dh, **0Ah**

Wat je wil tonen in uitvoerzone (outarea) → met movsb en stosb

Openuit en schrijf oproepen

Van integer naar string

Getallen → **meerdere voorstellingen** (binair, hexa, ascii, decimaal,…) ➔ **ASCII** om af te **drukken**

➔ **omzetten!**

```
mov edi, outarea + ??
std
mov ebx, 10
lus: mov edx, 0
idiv ebx
add dl, 30h
xchg al, dl
stosb
xchg al, dl
cmp eax, 0
jne lus
```

(gewoon van buiten kennen)

Invoerbestanden

Bestand bestaat uit **records** → iets dat logisch samen hoort → alle records even lang (max 70 tekens)

Bestand lezen

- in **programma**: **invoerzone** definiëren
  - **inarea: resb 70**
- Bestand openen met **openin**
- Een lijn lezen met **lees**
- OS houdt **bestandswijzer** bij
  - ➔ aantal bytes dat al gekopieerd is
  - Begint bij 0

```
Inarea: resb 70
…
lees
```

- Kopieert bytes invoerbestand → werkgeheugen
  - Naar **inarea**
  - Vanaf **bestandswijzer** tot vòòr **0D0A**
    - Eventueel aangevult met spaties
- Bestandswijzer **verhogen** met
  - Aantal **gekopieerde bytes**
  - **+2** (omwille van 0D0A)

Bestand verwerken

```
        inarea: resb 70
        …
hoger:  lees
        … ;verwerking van record
        jmp hoger
        …
        slot
```

Probleem: *wat gebeurt er als het bestand op is? Wanneer stopt dit programma?*

- Als het **bestand** op is → **eax wordt 0**
- Programmeur moet dus **na** het **lees**-bevel inhoud van **eax testen**

Van string naar integer

```
hulp: resd 1
…
hoger: …
lees
mov esi, inarea
mov ecx, 4
tekstbin
mov [hulp], eax
mov esi, inarea + 4
mov ecx, 4
tekstbin
add eax, [hulp]
```

Tekstbin: tekst naar binair

- Vooraf: **verplaatsing 1e** byte **in ESI**
- Aantal **bytes** in **ECX**
- Dan **tekstbin**
- ➔ **binaire** voorstelling komt in **EAX**
- Enkel **ASCII waardes** (30 t.e.m. 39) mogen voorkomen
    - Spaties wel
- **Resultaat** moet in **EAX** kunnen

Hoofdstuk 4: programmatuur

Assembleerprogramma

Machine en assembleerbevelen

Programma → **machinebevelen (add, mov, sub,…)** && **assembleerbevelen (extern, section,…)**

Assembleerbevelen → aanwijzingen aan **vertaalprogramma** → gebruikt deze tijdens vertalen = **DIRECTIEF**

Uit de UCLL bevelen

- extern ExitProcess
  …
  call ExitProcess
  - terug naar besturingssysteem; ExitProcess → symbolisch adres
- [section .data]
  …
  [section .code]
  - Hier begint data- en code segment
- start:
  - hier begint de uitvoering

Taak van vertaalprogramma

Wij gebruiken:

- Mnemotechnische functiecodes
- Symbolische adressen
- Directe operanden
- …
  - Processor verwacht deze machinebevelen

Vertaalde versie → alles moet hierin staan om het bevel uit te voeren

Wat is alles?

- Functiecode
  - Wat je wilt doen
- Operand
  - Met wat je wil werken
    - Directe operand
    - Verplaatsing
- Welk register van de C.V.E.
  - EAX, EBX, …, EDI, ESI

Soms is vertaling 1 byte → movsb

Byteteller →

- Per segment andere teller
- Begint bij 0
- Waarde byteteller is aantal bytes dat al gebruikt is

Symbolische adressen

Naar een symbolisch adres wordt verwezen → kan eerst komen; voor de definitie

Niet elk bevel kan bij de eerste lezing vertaald worden → vertaling in 2 fases

- 1e fase
  - Elke lijn van broncode lezen en bytes tellen
    - Aparte teller voor elk segment (gegevens & bevelen)
  - Bij definitie van symbolisch adres → info bijhouden
    - Symbolisch adres & waarde byteteller opslaan in tabel
    - ➔ symbolentabel maken
- 2e fase
  - Programma leest iedere lijn
    - Vertaalt alles
  - Anders voor data- en codesegment
    - Constanten en veranderlijken ➔ verplaatsing (waarde byteteller)
    - Sprongbevelen → niet adres van bevel; → "spring bytes verder"

Vertalen van sprongbevelen

Vertaalprogramma weet hoe sprongen uitgevoerd worden:
haalcyclus en **uitvoeringscyclus**

017 7C ?? jl nega
019 …

Sprong → bevelenteller aanpassen

- Bevelenteller → 00000019
- nega == 00000020
- jl nega → 7C 07 ➔ 00000019 + 00000007 = 00000020 (hexa)

[fuck vertalen van sprongbevelen, ik sla dit deel over]

Uitvoering programma

Vertaalde versie op schijf → 0000002D 00000064 FFFFFFF1 00000041 0000004B FFFFFFEA FFFFFFEE FFFFFFF9 00000014 0000000A ????????...

Programma gestart, gaat het besturingssysteem:

- het programma kopieren naar werkgeheugen
- basis (data- en codesegment) instellen
- springen naar begin van programma

Programma kan om het even waar in het geheugen staan (in programma → verplaatsingen, geen adressen)

Andere plaatsen:

- Basis heeft andere waarde
- Verplaatsingen blijven hetzelfde

Subroutines, linker, macro's

Waarom?

- Grote programma's **verdelen** in **deelprogramma's** die **samenwerken**
  - Verdeel en heers
  - Meerdere programmeurs
  - Correcties
  - Herbruikbaar
- 2 soorten: **macro's** en **subroutines**

Register EIP

- CPU heeft adres van volgend uit-te-voeren bevel nodig
- Adres = basis + verplaatsing

Intel processor:

- Verplaatsing volgend uit te voeren bevel
  - = inhoud bevelenteller
  - = inhoud EIP

Stapelbevelen

Stapen → datastructuur in werkgeheugen

> Data toevoegen aan top van stapel met push
>
> Data afhalen van de top van stapel met pop

Stapel → deel van werkgeheugen

- Apart segment
- Aangegeven met stack pointer (= stapelwijzer)
- Intel: bewaard in ESP
  - ESP ➔ verplaatsing van bovenste byte van stapel

Data op de stapel →

- Inhoud van register: push ecx
- Dubbelwoord: push dword [getal]
- Verplaatsing: push dword getal

Push → iets op stapel zetten

- Eerst ESP verminderen (4,2,??)
- Dan kopieren (registerinhoud)

Pop → omgekeerd

ESP → verplaatsing van bovenste byte van de stapel

Niets op stapel → ruimte gereserveerd voor de stapel; ESP → verplaatsing 1^e byte van stapel

Stapel → snelste manier om registerinhoud te bewaren

Call-, ret bevelen

Call:

- Onthoudt waar je bent in code
- Spring naar (zoals jump)
- Terugkeeradres → inhoud EIP → adres van volgend bevel → wordt bewaard op stapel

Ret:

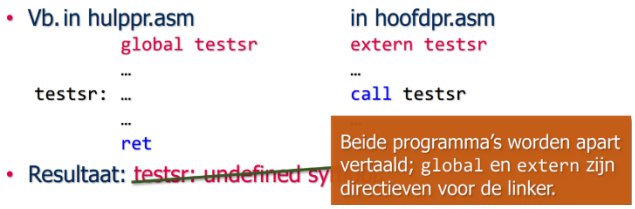
- Ga terug van waar je komt (call bevel)

Subroutines: nevenwerkingen & parameters → zie boek p 122-127 en dia's 34-74

Externe subroutines

Subroutines van ander programma →

Assembler en linker:



- Assembler → converteert .asm bestand naar .obj bestand
  - Asm bestand kan bestaan uit code en datasegment
- Linker → verschillende objectbestanden samen en genereert executable

Systeemroutines

Toegang tot randapparatuur via OS → subroutines → in Windows: kernel32.dll

Linker linkt programma met kernel32.dll

Systeemroutines oproepen →

1. Routine als extern declareren
2. Nodige parameters op stapel zetten
3. call …

```
extern ExitProcess
…
call ExitProcess
```

Macro's

Macro → afkorting aantal programmalijnen

Oproepen → naam macro in code zetten (bv wissen)

```
%macro wissen 0 ; macro-prototype
    cld
    mov edi, outarea
    mov ecx, 70
    mov al, ' '
    rep stosb
%endmacro
```

Hogere vs lagere programmeertalen

Inwendige machinetaal → hexadecimale / binaire instructies; geen symbolische adressen

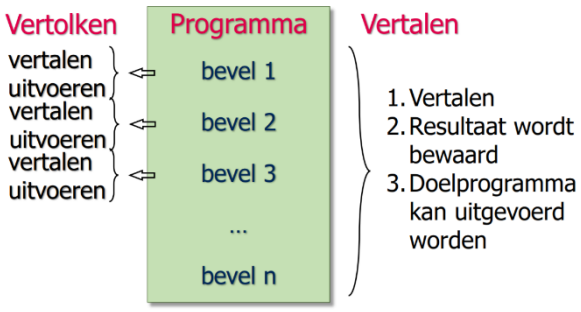
Uitwendige machinetaal → mnemotechnische functiecodes; geen symbolische adressen

Lagere programmeertaal:

- Mnemotechnische functiecode
- 1 bevel = 1 processorinstructie
- Wordt geassembleerd door assembleerprogramma (assembler)
- Ook: **assembleertaal**

Hogere programmeertaal

- 1 bevel = 1 of meerdere instructies
- Wordt cecompileerd door compilator (compiler)



Vertalen vs vertolken

Vertolken → minder geheugen

- telkens opnieuw vertalen bij opstart van programma
- bij lussen wordt elk bevel v.d. lus opnieuw vertaald

Bevel → kan worden opgedeeld in meer elementaire bevelen

- add, sub, …: ophalen van een dubbelwoord uit het werkgeheugen
- add, sub, cmp: aanpassen van de vlaggen
- lodsb, stosb, movsb: aanpassen van esi en/of edi

→ voor ieder bevel is er een lijstje met uit te voeren microbevelen → **microprogramma**

- machinebevelen uitvoeren → microprogramma uitvoeren
- microprogramma's → bewaard in geheugen

- 1 bevel uit hogere programmeertaal → vervangen door meerdere machinebevelen
  - Gebeurt op voorhand door compiler
- 1 machine bevel → vervangen door meerdere microbevelen
  - Gebeurt tijdens uitvoering programma
  - CPU interpreteert machinebevelen

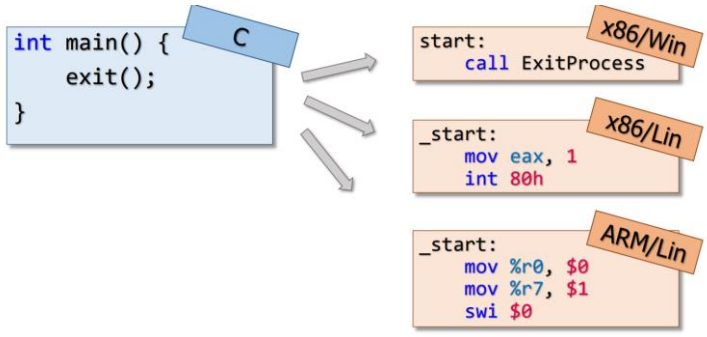
CISC: Complex Instruction Set Computer

- Veel verschillende en krachtige machinebevelen
- Bv. x86 intel processoren

RISC: Reduced Instruction set Computer

- Beperkte set machinebevelen
- Bevelen zijn minder krachtig
- Geen microbevelen
- Meer registers; minder verbruik
- "20% van bevelen deden 80% van al het computerwerk"

Java Virtuele Machine (JVM)



Java → andere aanpak

JVM → bestaat niet echt; eigen taal → **bytecode**

Java-code → compileerd naar bytecode → hetzelfde voor elk apparaat

Bytecode uitvoeren → vertolker nodig

- Elk bytecode-bevel omzetten naar machinebevelen van het apparaat
- Deze machinebevelen uitvoeren

Vertolker → (1 per CPU / OS) → **JVM**

- Bevelenteller & bevelenregister
- Geen andere registers
- Gebruik van stapel voor 'alles' bv optelling 2 getallen
    - Zet 1e getal op de stapel
    - Zet 2e getal op de stapel
    - Tel op (hierdoor worden de 2 bovenste getallen van de stapel gehaald en opgeteld)
    - Resultaat (de som) komt op de stapel

JIT-Compilatie → Just in Time

- Na interpretatie wordt de vertaalde versie bewaard voor een volgende uitvoering
- Efficiëntere uitvoering van o.a. lussen

Hoofdstuk 5

Moederbord, chipset, bussen

Moederbord → elektronische componenten, conectoren, socketten, verbindingen

Chipsets

Onderdelen van **computer** sturen **bits naar elkaar**

- CVE ←→werkgeheugen
- Werkgeheugen ←→ harde schijf
- Netwerkkaart ←→ geheugen
- … ←→ …

Lang geleden → **busverbinding**

- 1 gemeenschappelijke verbinding
- Gebruik om beurten

Beter → brug-ic's → **chipsets**

- Noordbrug voor snelle apparaten
  - Processor
  - Geheugen
  - Grafische kaart
  - Tegenwoordig geïntegreerd in CPU
- Zuidbrug voor tragere apparaten

Bussen

Eenheden direct verbinden → snel maar duur

Bus → meer dan 2 eenheden gebruiken hetzelfde transportmiddel → trager maar goedkoper

- Vroeger → alles in pc via 1 bus

Buseigenschappen

Breedte

- Aantal bits dat tegelijkertijd **gestuurd** kunnen **worden**
  - 32,64,128,…
  - **NIET** hetzelfde als brandbreedte

Cyclus

- Buslijn moet zelfde toestand blijven
  - 0 of 1 plaatsen → duurt even
- **Klok** regelt timing bussignalen
  - "één 1 op buslijn" → gedurende 1 klokcyclus 5V spanning op de lijn
- Bus → cyclus → tijd vereist om één bit te kopiëren

Frequentie

- Omgekeerde van cyclus
  - Frequentie → bv 33MHz
  - Cyclus is dan → 1/33000000 sec
    - 30 ns (nanoseconden)
- Buscyclus van bv. 30 ns → langer dan C.V.E.-cyclus
  - Bus is flessenhals (bottleneck) → buscyclus kan veel langer dan CVE cyclus zijn
    - Oplossing → bloktransfer
  - Indien alles via zelfde bus → nog trager

MT/S

- Megatransfer per seconde
- Frequentie (in MHz)
- Aantal keer dat iets op 1 buslijn kan worden gekopieerd

Debiet & bandbreedte

- **Debiet** → aantal (Giga)byte dat per seconde kan worden gekopieerd
  - → **bandbreedte** → bvb 8GB/s
- Debiet → MT/s x busbreedte

PCI → Peripheral Component Interconnect → verbonden met zuidbrug

Verschillende PCI versies, bv PCI-X 533

- frequentie : 533 MHz (PCI-X 533)
- breedte : 64 bit (8 bytes per cyclus)
- debiet : 4,3 Gigabyte/seconde

Parallel vs serieel

Parallel

- PCI-X 533 stuurt 64 bits per keer **parallel door**
  - Parallelle communicatie → traag
  - 64 bits sturen → wachten om zeker te zijn dat alles is aangekomen

Seriel

- Slechts één lijn gebruiken → hogere snelheid

PCIe → PCI express → bus met **bidirectionele seriële** kanalen

- Per kanaal kan aparte datastroom verstuurd worden
- PCIe 3.0 → 985 MB/s per kanaal per richting

PCIe x16 (of x4 of x8) → 16 lanes → debiet ➔ 16 x 985 MB/s x 2 richtingen → ~32 GB/s

Processor

Functie → **machinebevelen uitvoeren**

CPU → IC (geïntergeerd Unit)

- Bestaat uit transistoren
- Verbonden met de rest van apparatuur → pinnen
    - Vroeger → 40 pinnen
    - Nu → honderden pinnen

Wet van Moore → aantal transistoren verdubbelt om de 18-24 maanden

Gevolg ➔

- Grotere woordlengte
- Meer en krachtigere bevelen
- Grotere cache op C.V.E.

IC → matrix structuur

- Spoorbreedte → breedte van rij/kolom
- Uitgedrukt in nanometer (45nm, 32 nm,…)
- Verwachting → 5 nm tegen 2021

Transistoren → enkele atomen

Woordlengte → aantal bits van getallen waar processor mee werkt

- 8,16,32,64,…
- Grootte van de →
    - Registers
    - Bussen in (c.v.e.) en tussen c.v.e. en werkgeheugen
    - ALU (arithmetic logic unit)

Rekenen met getallen van 32 bits beter dan getallen van 16 bits

- Type processor → bevelenset
    - Machinebevelen die processor kan uitvoeren
    - Hoe meer bevelen, hoe meer transistoren

Nut meer bevelen in bevelenset →

- Rekenen met kommagetallen
    - Subroutines en rekenen en integers
    - Coprocessor
    - Vanaf i486 → processor kan floating point bewerkingen

Krachtigheid processor

- Bevelen → krachtiger
- Klok → sneller
- Prestaties meten → **Mips**
    - Aantal miljoen machinebevelen per seconde
- Megaflops → miljoenen floating point operations

Bevelenwachtrij

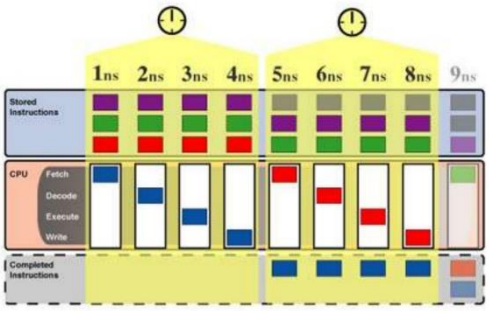
- Tijdens uitvoering van machinebevel → ophalen volgend bevel
- Snelheidswinst

Pijplijn

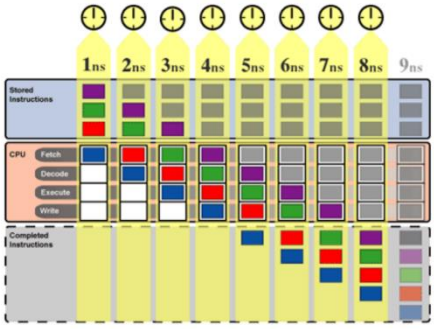
Pijplijn → stap verder

- Uitvoering van bevel in meerdere stappen

Zonder pijplijn →



Met pijplijn →



MIPS pijlpijn →

1. Instruction fetch → instructie ophalen uit geheugen
2. Instruction Decode – instructie interpreteren
3. Execute – instructie uitvoeren
4. Memory – geheugentoegang (indien nodig)
5. Writeback – resultaat wegschrijven

20 bevelen met pijplijn → 1ste bevel afgewerkt na 6 cycli; bevel 2 na 6 cycli en 20 bevelen na 24 cycli

Bij voorwaardelijke sprong (je, jgl,…) → **Branch prediction**

Fout → **roll back** uitgevoerd

Bevelen die elkaar inhalen in pijplijn → **out-of-order execution**

Besluit →

Vereiste voor pijpijn → alle fases 'gelijkaardig' → worden binnen 1 cyclus afgewerkt

Lengte pijplijn → varieert per processor

Multiprocessing

Hedendaagse server → meer dan één processor op moederbord

- Vroeger → verbonden via Front Side Bus (FSB
- Nu → andere technologiën → bv. Intel QuickPath

Multiprocessing → *simultaan uitvoeren van twee of meerdere programma's op 1 computer met meer dan 1 CPU*

1 CPU → multiprogrammering

- OS laat processor wisselen van programma (bv 100x per seconde)
- Meerdere programma's → kunnen stapsgewijs uitgevoerd worden

Nadeel → wisselen programma's zorgt voor **overhead**

- Registers moeten eerst weggeborgen worden in geheugen & hersteld voor terugkeer
    - Op x64 → 40+ registers
- Oplossingen → **hyperthreading** en **multicore**

Hyperthreading

- Processor kan status van meerdere processen bijhouden
- Meerdere sets van registers

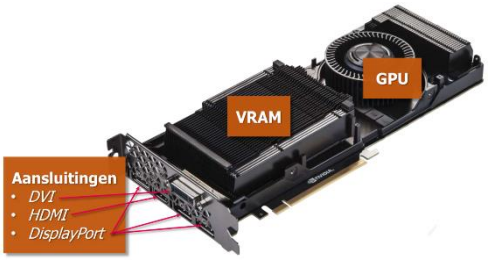
Multicore

- Een stap verder dan hyperthreading
- Meerdere sets van registers
- Meerdere ALU's

Grafische Verwerkingseenheid

Scherm → pixels → puntjes met bepaalde kleur

- Per pixel → 3 bytes → rood, groen, blauw
- Tegenwoordig 4e byte → opaciteit



Aantal bytes in breedte en hoogte → **resolutie**

Beeldscherm voorzien van data → taak van **grafische verwerkingseenheid**

Veel data →

- Bewegende beelden minimum 25× per seconde verversen, maar typisch 60×
- 4K-resolutie: 4096 × 2160 × 4 bytes × 60/s = 2GB/s (!)
- Via PCIe ×16 verbonden met de noordbrug

GPU → doet veel meer

- → zeer veel werk
- Krachtige gpu nodig

Bv nVidia GTX Titan X

- 3072 processor cores
- 12GB VRAM
- 336,5 GB/s geheugen bandbreedte

Werkgeheugen

Gekopieerd van CPU naar werkgeheugen en omgekeerd:

- Bevelen opgehaald uit werkgeheugen
- Resultaten weggeborgen in werkgeheugen
- Gegevens in werkgeheugen

CPU → **leest** / **schrijft** het werkgeheugen

Werkgeheugen → gecontroleerd door **geheugenbesturingseenheid** (**GBE** of **memory controller**)

- Vroeger → chip tussen bus en werkgeheugen
- Nu → geïntegreerd in **noordbrug**
- Noordbrug → **geïntegreerd** in **CPU**

GBE → 2 registers

- **Geheugenadresregister** (**GAR**)
- **Geheugenbufferregister** (**GBR**)

Communicatie CPU & Werkgeheugen

1. CPU berekent adres
    a. Bevel → inhoud EIP + basis
    b. Data → basis + verplaatsing in bevel (+ indexregister)
2. CPU plaatst adres in GAR
    a. Lees →
        i. CPU geeft lees bevel
        ii. GBE plaatst data in GBR
        iii. CPU kopieert naar eax, ebx, ALU,…
    b. Schrijf →
        i. CPU plaatst data in GBR
        ii. CPU geeft schrijft-bevel
        iii. GBE slaat data op

Per lees- of schrijfbewerking worden er meerdere bytes gekopieerd (woord)

Lees- of schrijfbewerking is klaar wanneer →

- Asynchroon → werkgeheugen geeft signaal indien klaar
- Synchroon → na aantal klokcycli (wordt nu gebruikt)

Eigenschappen Werkgeheugen

- Toegangstijd → tijd tussen lees-signaal en beschikbaar zijn van data in GBR
- Cyclustijd → duur tussen 2 bewerkingen in geheugen (bv. 2 lees bewerkingen)
- Frequentie → aantal klokcycli per seconde
    - Bv. 200 MHz → 1 cyclus = 5 ns

SDRAM / DIMM

S → Synchroon

DRAM → Dynamic Random Access Memory

Werkgeheugen → aantal IC's → bestuurd door G.B.E.

- CPU levert adres aan GBE en lees-signaal / of levert adres, data en schrijftsignaal aan GBE
- GBE doet de rest

SDRAM → modules (DIMM → Dual In Line Memory Module → dubbele rij contactpunten)

DIMM → bestaat uit geheugen IC's

IC's → bestaat uit transistoren

Heeft typisch 8 of 16 IC's

DIMM → 168 tot 288 contactpunten

- Aantal voor data overdracht
- Aantal voor adres
- Aantal voor besturingssignalen

Data in DIMM → deel van adres nodig om DIMM te selecteren

- 1,2,… bits
- Chip-select → DIMM select

Vb →

- Module van 1 GB
  - 8 SDRAM-IC's van 128MB
  - 1 cyclus → per IC 1 byte gelezen / geschreven
    - Woordlengte → 64 bits
  - Elke IC → zelfde signalen van GBE

IC's

→ opgebouwd uit **banks**

- Bank → matrix (rijen en kolommen)
- Kruispunt van 1 rij en 1 kolom → bevat een bit
- Vb →
  - 128MB kan opgeslagen worden in 8 matrices van 16 Mbyte

Matrix van 16MB

- Bevat invoerlijnen
  - Besturingsdata
  - Besturingslijnen
- Bevat 8 I/O lijnen
  - Bidirectioneel (afhankelijk van lees / schrijf)
  - Per I/O lijn: 1 bit → 8 bits / cyclus
- Bevat 2^{27} bits
  - 16384 (214) rijen en 8192 (213) kolommen
- 1 rij → 8192 of 2^{13} bits → 1024 of 2^{10} bytes

Adressering

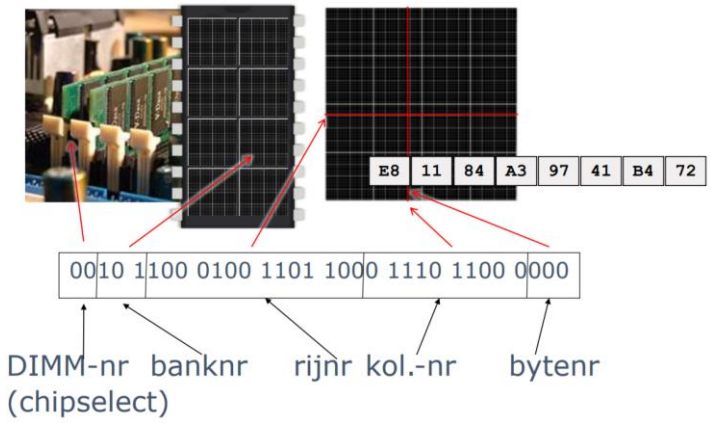
(1) DIMM selecteren → 2 bits

IC van 1 DIMM → zelfde bewerking, 1 byte lezen / schrijven

Welke byte selecteren? 3 bits → 2^3 matrices

- **(2)** Welke matrix (bank)
- Welke byte in matrix
  - (3) Welke rij? 14 bits want 2^{14} rijen
  - (4) Welke kolom? 10 bits van 2^{10} groepen van 8 bits

Vb:

- CVE vraagt 4 bytes vanaf adres 2C4D8EC0
- GBE levert (altijd) 8 bytes vanaf adres (binair)
  - 00101100010011011000111011000000
- Adres → opgedeeld
- 

Besturingslijnen

- WE → Write Enable (0 = lezen, 1 = schrijven)
- Rij en kolomnummer → zelfde lijnen na elkaar
  - /RAS (row address strobe) → rij nummer
  - /CAS (column address strobe) → kolom (groep) nummer
- In 3 fases

1. Fase 1 → rij selecteren
   a. GBE stuurt bank- en rijnummer
   b. GBE geeft RAS-signaal
   c. **Elke IC v.d. DIMM selecteert rij**
2. Fase 2 → kolom selecteren
   a. GBE deactiveert WE (= geen schrijf)
   b. GBE stuurt kolomnummer
   c. GBE geeft CAS-signaal
   d. **Elke IC levert byte op I/O lijnen**
3. Fase 3 → precharge
   a. GBE deactiveert RAS en CAS
   b. IC maakt zich klaar voor volgende lees / schrijf operatie

Wachttijden

GBE → stuurt rijnummer → … → IC levert byte op I/O lijnen

RAS to CAS delay → RC-**wachttijd**

CAS-signaal → … → IC levert byte op I/O lijnen → **CAS-latency**

GBE mag niet direct weer rijnummer sturen (volgend lees / schrijf) → **precharge latency**

Vermeld als → CAS – RC – precharge (- som)

Bv. →3-2-2 (of 3-2-2-7)

Signalen → komen volgens timing van bus

Performantie

Vb.: SDRAM-module met wachttijden van 3-2-2

Vb →

```
mov eax, [2C4D8EC0h]
mov ebx, [2C4D8EC4h]
mov ecx, [2C4D8EC8h]
mov edx, [2C4D8ECCh]
```

- GBE →
  - Geef 8 bytes vanaf 2C4D8EC0
  - Geef 8 bytes vanaf 2C4D8EC8
- Addressen → zelfde DIMM-, bank-, en rijnummer
  - 1^e lees → 7 cycli
  - 2^e lees → 3 cycli (alleen CAS-latency)

Bloktransfer gaat verder

- Uitlezen 64 bits woorden met opeenvolgende adressen → CAS-latency valt weg
- → 1 64 bit woord / cyclus

Waar? → in **CVE** (in cache)

Uitbreiding mogelijk

Wet van Moore → meer transistoren per IC

- Matrices met meer rijen en / of kolommen
- Meer matrices
- ECC (error correcting code) → extra schakelingen om fouten op te sporen & verbeteren

DDR SDRAM

Gewone RAM → SDR SDRAM → Single Data Rate SDRAM → 1 bits / memory cyclus

Geen bits afgegeven bij dalen van spanning

DDR SDRAM → Double Data Rate SDRAM → 2 bits / memory cyclus

Bij stijgende en dalende spanning → bits afgegeven

Ook:

- DDR2 → 4 bits / memory cyclus
- DDR3 → 8 bits / memory cyclus
- …

Snelheid

Debiet bij bloktransfer aan 100 MHz →

- SDR → elke klokpuls 1 bit → 100 Megatransfers per sec per IC
  - Debiet → 800 MB/s'
- DDR → 2 bits / klokpuls → 200 MT/s
  - Debiet → 1,6GB/s
- DDR2 → 400 MT/s (Debiet → 3.2GB/s)
- DDR3 → 800MT/s (debiet → 6,4GB/s)

Soorten transistorgeheugens & cache

Soorten transistorgeheugens

Meerdere soorten transistoren

- Bipolair
- MOS → MOSFET → metal oxide…
- NMOS, PMOS / combinatie → CMOS (complementary MOS)

Un → vooral CMOS

Verschillende soorten geheugen → DRAM, SRAM, ROM,PROM,…

DRAM → Dynamic RAM

- Per bit → 1 MOS-transistor, 1 condensator
  - Klein, bus veel bits per mm^2
- MOS → dynamisch → informatie lekt weg → voortdurend opfrissen
- DRAM → herschrijven na lezen

SRAM → Static RAM

- Static → geen opfrissing nodig
- Per bit → 6 transistoren, 1 flipflop
  - **Sneller** dan DRAM
  - Groter, dus **minder bits** per mm^2

Andere eigenschappen

- Energieverbruik, snelheid, aantal bits/mm^2, prijs/bit,…
- CMOS → laag energieverbruik
- SRAM → snelst, maar duur
- DRAM → trager, maar goedkoper
- Verstandigste toepassing SRAM → cache

ROM → Read Only Memory

Tijdens fabricage → opslaan van informatie (niet verandelijk)

→ software op computer te zetten

- PROM → programmable ROM → 1 keer info opslaan
- EPROM → erasable PROM → toch te veranderen (bestalen met UV om te wissen)
- EEPROM → electrically EPROM → toch te veranderen (wissen gebeurt byte per byte)
- Flash geheugen → wissen per blok

Cache

Informatie → liever bijhouden in register dan werkgeheugen

Niet veel registers (eax, ebx, ecx, edx,…)

Oplossing → snel klein geheugen dat kopie bevat van meest gebruikte bytes

→ **voorgeheugen** of **cache**

In cache hardware → algoritme om te beslissen wat in cache komt

→ bytes die CVE gebruikt staan meestal in elkaars buurt (**principe van lokaliteit)** → voorspelbaar

- CVE geeft →
  - Adres + lees bevel
  - Adres + data + schrijfbevel
- GBE doet de rest →
  - Info uit echt werkgeheugen OF
  - Info uit de cache

Cache → GEEN extra oplsagcapaciteit → enkel kopiën

Ophalen woord

Vb → woord met adres 22DC2A0C

- Nagaan of het in cache staat
  a. Ja → leveren
  b. Nee →
     i. Ophalen uit werkgeheugen (trager)
     ii. Alle woorden met adres 22DC2A**xx** (64x4bytes, bloktransfer)
     iii. Waarom? → **later nodig voor lokaliteit**

Opslaan woord

Vb → kopieer inhoud van EAX naar 12ABC020

- Opslaan in cache en verder gaan met uitvoering volgend bevel
  - Probleem → **cache inconsistentie**
    - Oplossing → onmiddellijk kopiëren naar werkgeheugen
    - Wachten tot inhoud cache gewist is

Hit → GBE levert info snel

Miss → GBE levert info later

Na miss → ook opslaan in cache

Later heeft CVE misschien zelfde informatie nodig

Opslaan van woord → komt altijd in cache

Soms iets weglaten uit cache →

- Item dat het langst niet meer is gebruikt
- **Trefverhouding** tot meer dan 90%

Staat adres in cache?

- CPU vraagt woord met adres 07600ADC
- Staat het in cache of niet?
  - Alle items nagaan → alle tijdwinst weg
  - Computergeheugen is als magazijn (weten waar alles staat)
- **→ associatief geheugen**

Associatief geheugen

Vb. 1912, kust New Foundland, film, ijsberg? → Titanic

Computer werkt alleen met adressen → programma nodig om associatief te denken

```
0000BB: 00 00 2A C0 FF FF 0A 10 C0 …
22DC2A: CC CC CC CC AB 00 FE 23 AB …
```

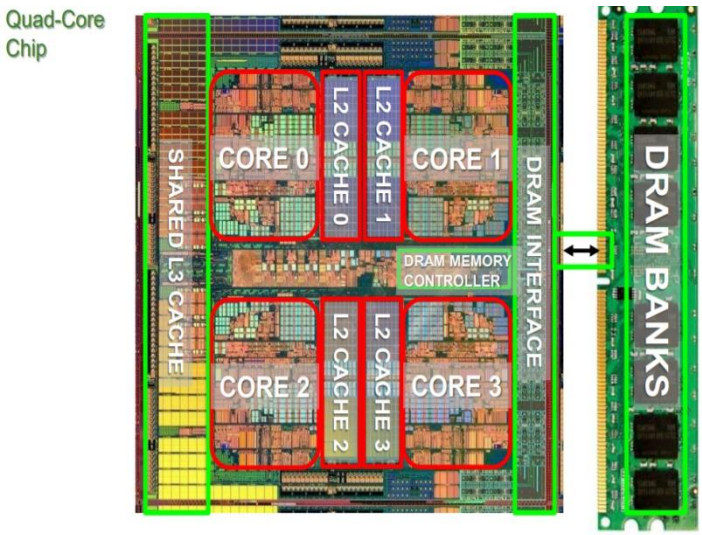
1. CVE vraagt woord met adress 22DC2A04
   a. 22DC2A04 wordt vergeleken met 0000BB, 22DC2A, … **terzelfdertijd**
   b. **2e lijn → hit** → AB 00 FE 23 → gevraagde word

- Werkgeheugen → om info op te vragen moet je het adres leveren
- Associatief geheugen → deel van inhoud opgeven, bv. 22DC2A

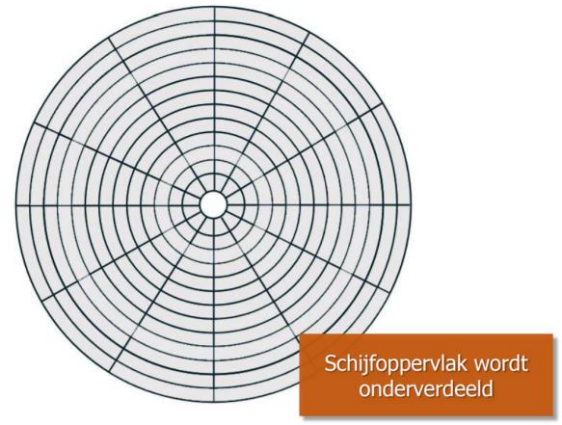
Associatief geheugen → ook **content addressable memory** (**CAM**) genoemd

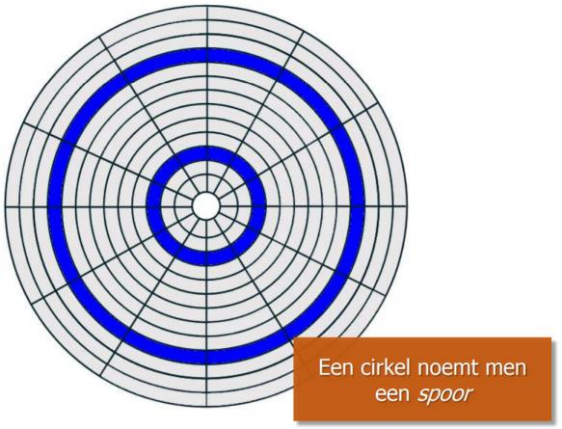
Cache →

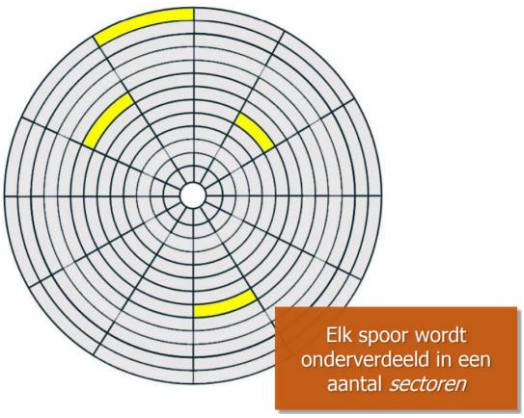
- Vroeger: extra op moederbord
  - SRAM-chip;
  - eigen zgn. cache-controller;
- Later:
  - één cache op moederbord: L2
  - één cache op CPU: L1
- Nu:
  - Geen cache meer op moederbord
  - Alle caches in CPU: L1, L2, L3
  - Eventueel elke core haar eigen L1-cache

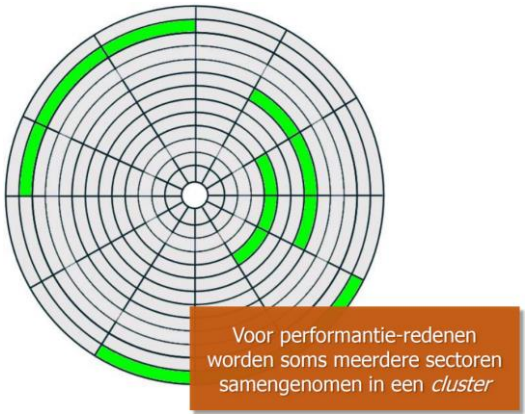


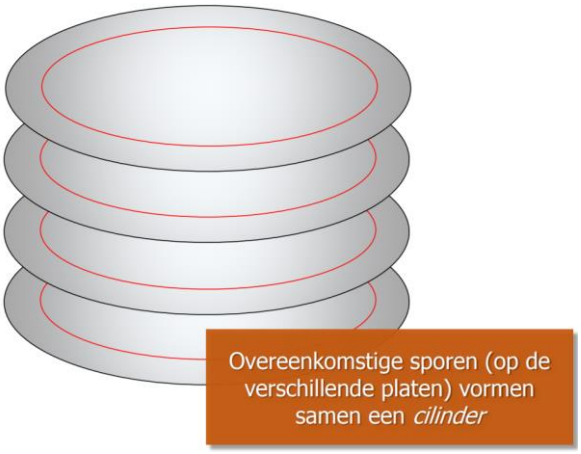
Magneetschijven (harde schijven)



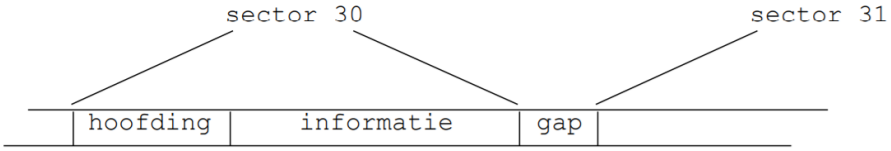








Indeling spoor



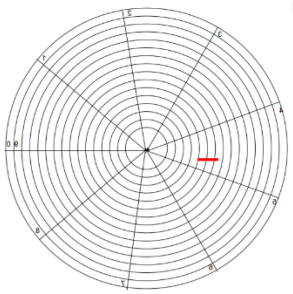
Naast data → ook foutcorrigerende code → pariteitsbits

Krijgen waarde 0 of 1 al naar gelang het aantal enen in de byte even of oneven is

0001 0101 → 1
0110 0110 → 0

Om meerdere fouten te herkennen → extra pariteitsbyte toegevoegd aan groep bytes

Capaciteit



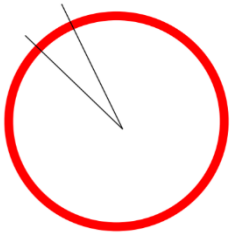
→ aantal bytes dat beschikbaar is

Netto capaciteit → aantal bytes dat voor data kan gebruikt worden

Bruto capaciteit → netto capaciteit + hoofdingen + controle bits + gaps

Radiale eenheid → volgens een straal → aantal sporen per lengte eenheid

Lineaire dichtheid → volgens een spoor



- Vroeger → aantal bits per spoor constant
  - Lineaire dichtheid in buitenste sporen is lager dan in binnenste sporen
- Nu → lineaire dichteid constant
  - Meer sectoren in buitenste sporen
  - **Zone Bit Recording**

Toegang →

- Gebruiker wil bestand lezen
  - OS houdt bij in welk sector bestand zit
- OS stuurt lees operaties voor betreffende sectoren naar **schijfbesturingseenheid (SBE)**
- Altijd hele sector in 1 keer gelezen / geschreven

Welke sector?

Vroeger → doorgegeven aan SBE van cilinder, kop en sectornummer

Nu → elke sector heeft uniek nummer (→**LBA-nummer**)

1. Arm (met de kop) verplaatst zich naar goede spoor → **zoektijd** (**seektime**)
    a. Zoektijd → variabel
    b. Startpositie → niet gekend
    c. Afstand → niet gekend
    d. Gemiddelde nemen van vele toegangen
        i. 4ms tot 15 ms
2. Wachten tot juiste sector onder kop bevindt → **rotationele wachttijd (latency)**
    a. Willekeurig tussen 0 en 1 van 1 toer
    b. Gemiddeld → helft van tijd voor 1 toer

Gemiddelde rotationele wachttijd voor schijf 7200 tmp?

- 7200 toeren per minuut
- 120 toeren per sec
- 1 toer in 1/120 sec → 0,008333 sec
- 1 toer in 8,33 ms

Gemiddelde rotationele wachttijd → 8,333 ms / 2 = 4,17 ms

Toegangstijd → **zoektijd + gemiddelde rotationele wachttijd**

Typisch 5ms – 20 s

Gebruiker schrijft veel data op meer dan 1 spoor →

- Zelfde spoornummer op ander oppervlak
- Zo weinig mogelijk van cilinder veranderen
    - Cilinder vol → cilinder ernaast

Rest hangt af van snelheid en gegevensoverdracht

- **Intern** → van schijf naar buffer
- **extern** → van buffer naar werkgeheugen
- eventueel herlezen indien fout (pariteit)

Lokaliteit → Bestanden die elkaar nodig hebben gaan dichter bij elkaar staan op de schijf (bv. een game, een programma,…)

Schijfbesturingseenheid

Schijf communiceert met rest van apparatuur via de SBE

SBE →

- arm bewegen op juiste spoor (**seek**)
- van alle sectoren de hoofding lezen, tot juiste sector passeert (**search**)
- Schrijven:
    - Gegevens (bv 4kbyte) aannemen van CPU en opslaan in buffer (SBE)
    - **Wegschrijven naar schijf**
        - Elektrische informatie (0V / 5V) → magnetische informatie (noord / zuid)
- Lezen:
    - Gegevens lezen van schrijf en opslaan in buffer (SBE)
        - Magnetische informatie (noord / zuid)→ elektrische informatie (0V / 5V)
        - Gegevens (4kbyte) naar CVE sturen'
- **Pariteitsbits berekenen** en toevoegen (bij schrijven)
- **Pariteitsbits controleren** (bij lezen), eventueel opnieuw lezen
- Clusters niet lezen in de volgorde van vraag → wachttijden zo klein mogelijk maken → **native command queuing (NCQ)**

Soorten SBE

Verschillende standaarden

- **SCSI**
    - Duurder dan ATA
    - Vooral bedoeld voor servers
    - Ook Serial attached SCSI → SAS
- **S-ATA**
    - Serieel ATA
    - 2 draden voor data; 2 draden voor besturing
    - Zeer populair voor hedendaagse low-end tot high-end schijven
- **M.2**
    - Nieuwe standaard voor zéér snelle schijven
    - Klein-form factor

Sectornummering & Zone Bit Recording

Sectornummering

Vroeger →

OS Sector opvragen → rechtstreeks cilinder, hoofd en sector nummer geven

Nadeel →

- OS moet geometrie kennen van schijf
- Hedendaagse SSD schijven niet magnetisch, cilinders & hoofden niet van toepassing

Nu →

Sector opvragen via **LBA-nummer**

- Eerst → 28 bit getal → 2^{28} sectoren van 512bytes → max schijfgrootte van 128GB
- Tegenwoordig → 48 bits → 2^{48} sectoren van 512 bytes → max grootte van 128PB (PetaByte)

Zone bit Recording

Buitenste spoor → meer bits → omdat sporen langer zijn

> ZBR → aantal sectoren per spoor is groter aan buitenkant

Vb. **zonder ZBR** →

- Schijf met 10000 cilinders
- 50 sectoren per cilinder
- Per oppervlak: 10000 × 50 sectoren, → 500000 sectoren

Vb. **met ZBR** →

- Buitenste sporen bevatten 99 sectoren
- Binnenste sporen bevatten 50 sectoren
- Per 200 sporen, 1 sector verschil
  - Sp 0 t/m 199: 99 sectoren
  - Sp 200 t/m 399: 98 sectoren …
  - Sp 9800 t/m 9999: 50 sectoren
- Dus (50+51+…+99) × 200 = 745000 sectoren
  - +49% dan **zonder ZBR**

RAID

Schijf → moet snel zijn

> Harde schijf is trager dan CPU / RAM

Schijf → betrouwbaar zijn

> Harde schijven crashen soms

Oplossing? → **Redundant Array Of Independent Disks (RAID)**

- SBE heeft meerdere schijven ter beschikking, **CPU ziet alleen SBE**
- **SBE** noemen we dan **RAID-Controller**

RAID-0 (ook disk striping genoemd)

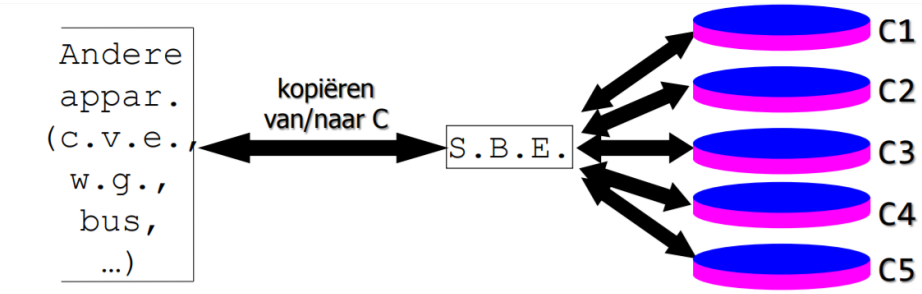
- SBE verdeelt blokken data over bvb. 3 harde schijven
- Resultaat →
  - Lezen / schrijven is sneller
  - Foutbestendigheid is kleiner

RAID-1 (ook disk mirroring genoemd)

- SBE kopieert data op 2 harde schijven
- Resultaat →
  - Beide schijven zijn identieke kopieën
  - Extra backup
  - Schrijven is even snel
  - Lezen gaat sneller

RAID-5

- SBE kopieert blokken data naar bv. 5 harde schijven
  - Data verdeeld in groepen van 4 blokken
  - Per 4 blokken wordt een 5de **pariteitsblok** berekend
- Resultaat →
  - Lezen / schrijven is sneller
  - Schijf defect → geen info verloren



- Pariteitsblok niet altijd op dezelfde schijf

1e blok	2e blok	3e blok	4e blok	P(1,2,3,4)
5e blok	6e blok	7e blok	P(5,6,7,8)	8e blok
9e blok	10e blok	P(9,10,11,12)	11e blok	12e blok
...	...	...	...	...

Formules voor oefeningen

Debiet → (breedte / 8) * snelheid

Breedte → (debiet / snelheid) * 8

Snelheid → debiet * (8/breedte)

Rotationele wachttijd → ((1/toeren per seconde) / 2)*1000

Sector leestijd → (Rationele wachttijd *2)/sector per spoor

Debiet → MT/s x busbreedte

Debiet →

1. snelheid/60 000
2. uitkomst * sectorPERspoor
3. uitkomst * sectorGroote
4. uitkomst / 1024
5. uitkomst / 1024
6. uitkomst * 1000