

Toegepaste Informatica

MBI07a

2022-2023



**UCLL**  
HOGESCHOOL

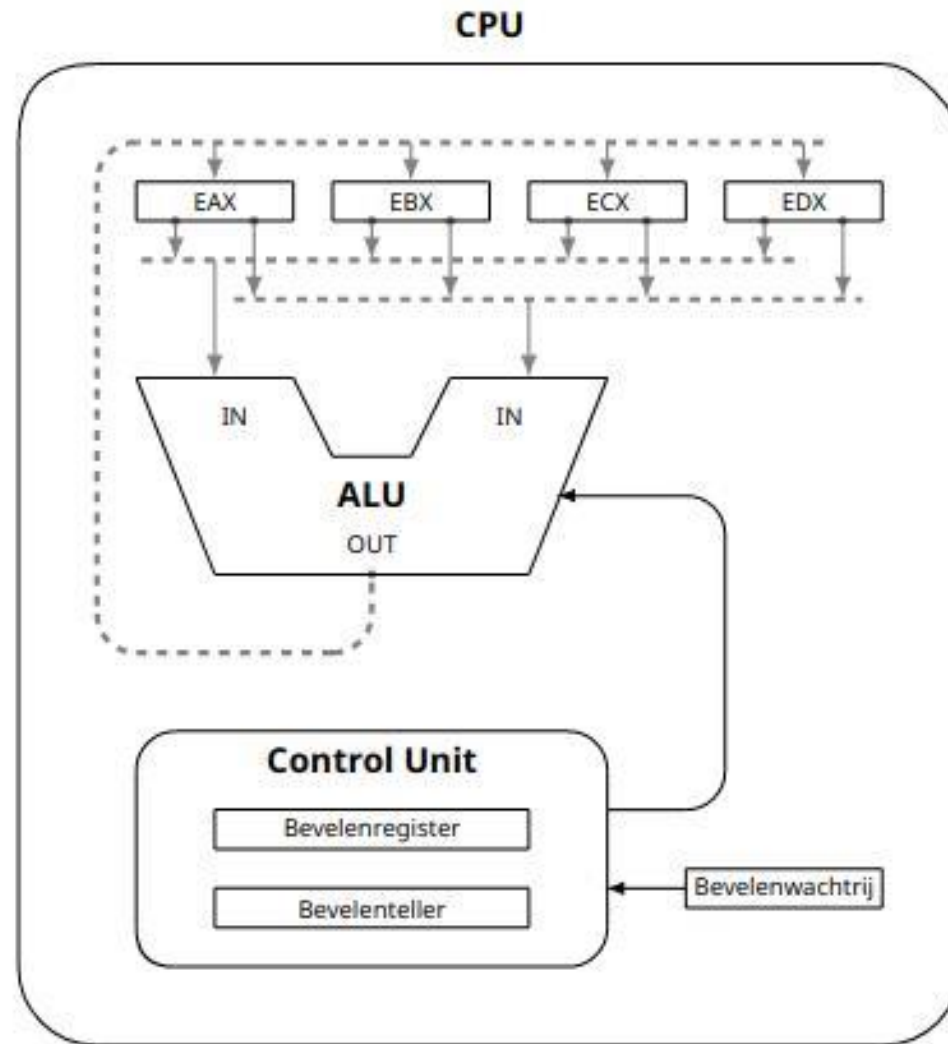


Computer  
Systems

CPU & Assembly

Jeroen Jean, Rudi Swennen, Tiebe Van  
Nieuwenhove, Frédéric Vogels

# CPU



# Registers

- Kleine stukje geheugen voor CPU
  - Tijdelijk oplan van waarden
  - Input en output van instructies
- Eigenschappen:
  - Vaste lengte
  - Inhoud kan gekopiëerd worden
  - Nieuwe inhoud overschrijft oude.

Zet h'01020304' in EAX:

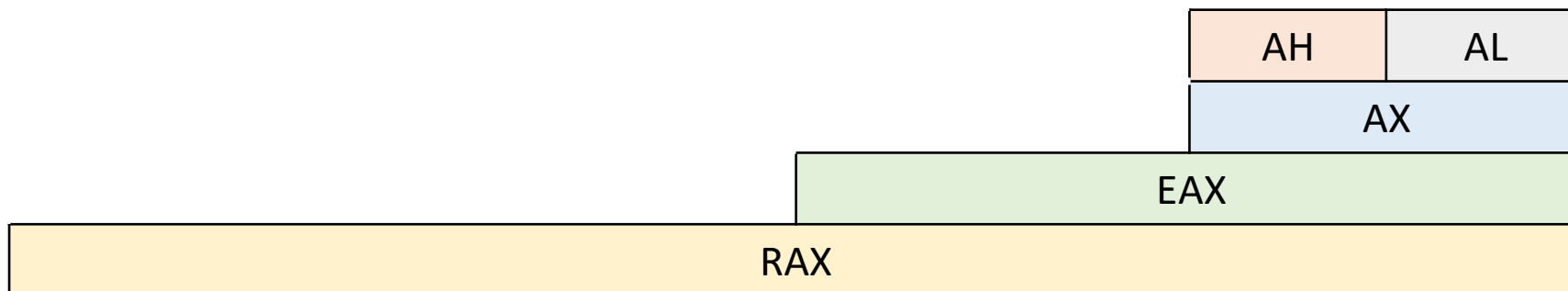
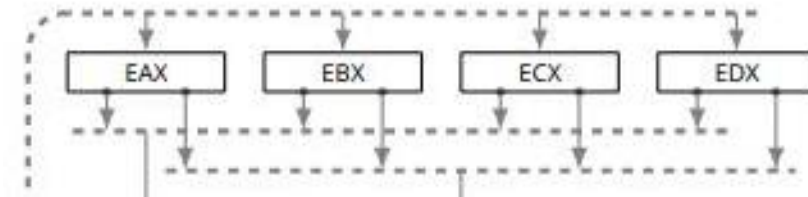
EAX			
01	02	03	04

Zet h'00000001' in EAX:

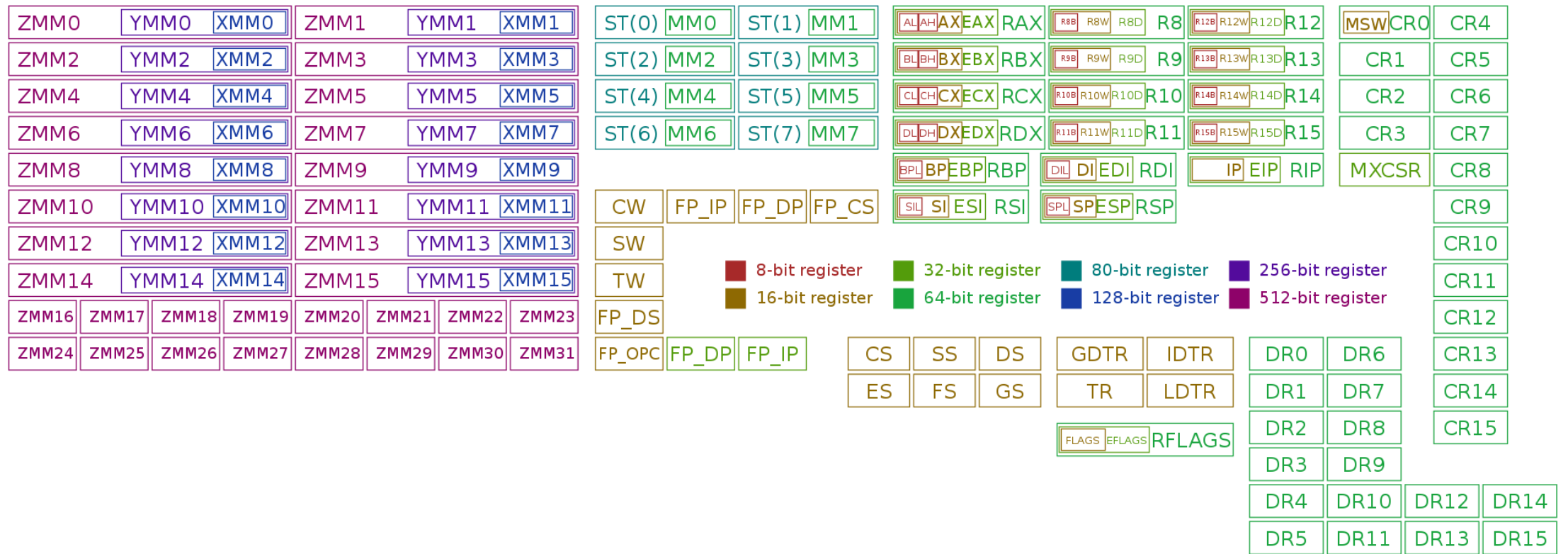
EAX			
00	00	00	01

# Register

- Meer dan 500 registers
- Intel: 16 general purpose registers
  - 4 voor berekeningen
    - EAX, EBX, ECX, EDX --> 32bit
    - RAX, RBX, RCX, RDX --> 64bit
- Elk deel kan apart aangesproken worden:

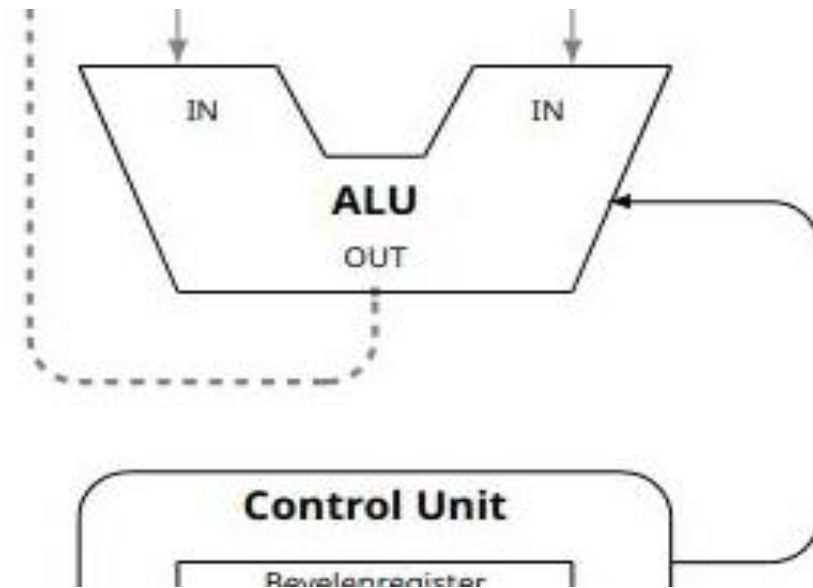


# x86-64 registers



# ALU

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



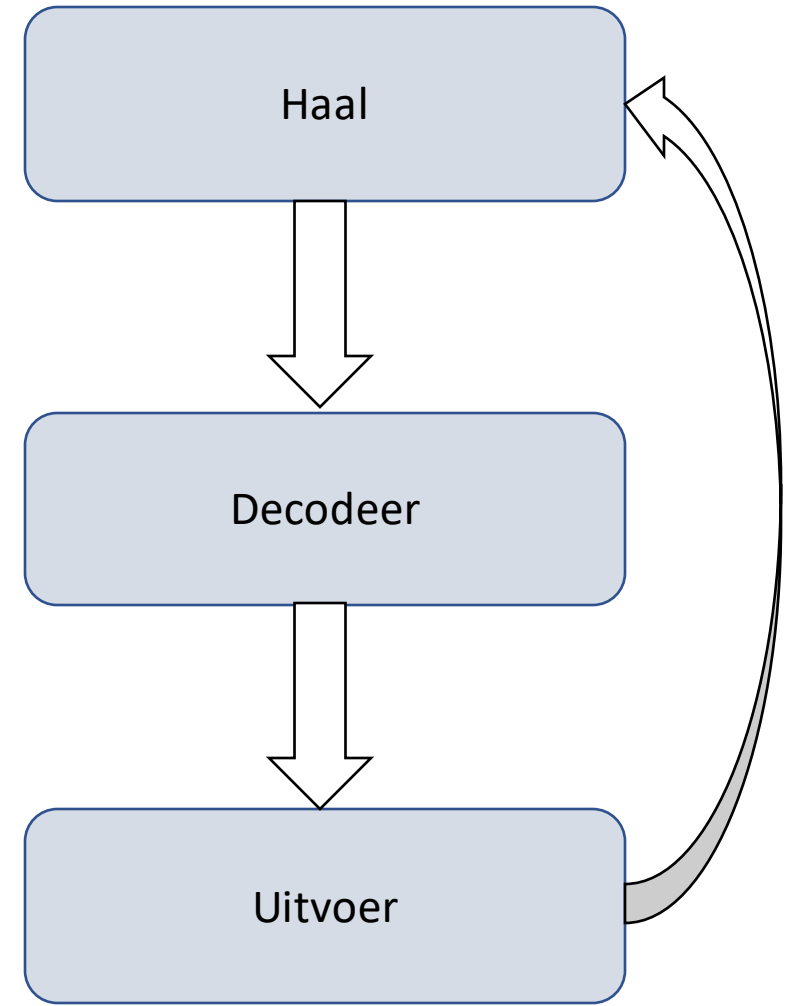
# Control Unit

- Zorgt voor het uitvoeren van code.
- Bevat twee specifieke registers:
  - Het **bevelregister** (instruction register, IR) bevat de bytes van het bevel dat uitgevoerd wordt
  - De **bevelenteller** (instruction pointer, EIP) bevat het adres van het volgende bevel



# Control unit

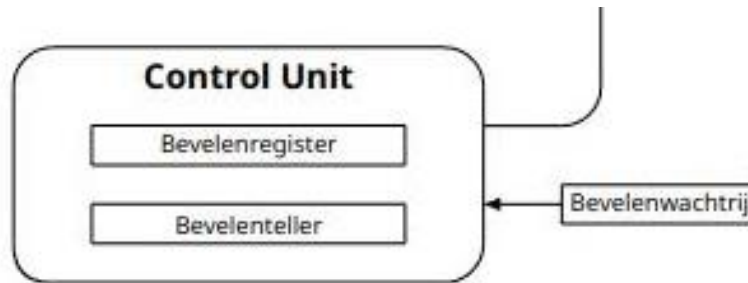
- Programma uitvoeren = vaste volgorde:
  - Haalcyclus:
    - Bevel in bevelenregister plaatsen waarvan adres in bevelenteller staat.
    - Bevelenteller aanpassen naar volgende op te halen bevel.
  - Decodeercyclus:
    - Bevel interpreteren door decoder.
  - Uitvoercyclus:
    - Bevel in bevelenregister uitvoeren.





# Bevelenwachtrij

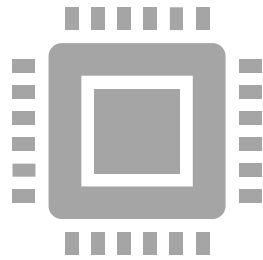
- Tussen werkgeheugen en CPU is er een verbinding
  - Wordt gebruikt om data te kopiëren
  - Ze is bvb. 32 bits breed (kan ook meer of minder zijn)



# Bevelenwachtrij

- Wat wordt er gekopieerd?
  - van register naar werkgeheugen
  - van werkgeheugen naar register
  - één van de operanden bij een berekening
  - de bevelen zelf
- De verbinding is relatief **traag**, er kunnen nooit twee dingen tegelijkertijd gekopieerd worden.
  - ==> Tijdens wachten (*op bv uitkomst van optelling*): volgende bevel ophalen en in bevelwachtrij plaatsen.

# Intel CPU modes

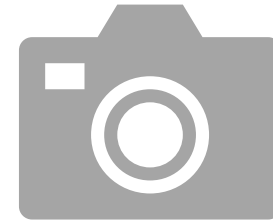


**Intel CPU's werken in 3 modes:**

Real

Protected

SMM (System Management Mode)



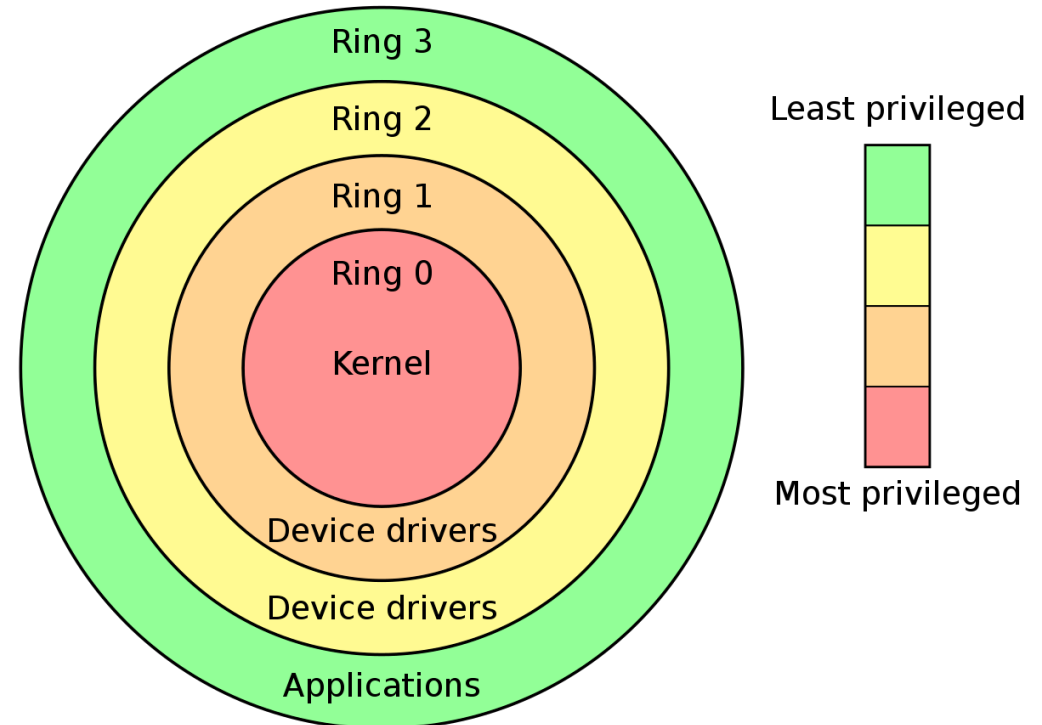
**Opstarten:**

Real mode (1 MB mogelijk om te adresseren)

Protected mode zodra OS begint op te starten

# Rings of protection

- Waar draait USER en KERNEL code?
  - Volgens privilege level !
- Level bepaald tot wat je toegang hebt.
- Tegenwoordig:
  - Kernel: Ring 0
  - User code: Ring 3



# Rings of protection: security level


- Zowel ring 0 als ring 3 in protected mode
- 2 Bits in **codes segment register (CSR)**
  - **Current Privilege Level (CPL)**
    - 00 = Kernel code
    - 03 = User code
- CSR: verwijst naar deel van geheugen waar uitgevoerde code staat.
- OS en CPU bepalen samen security level via CPL
  - Welke instructies mogen uitgevoerd worden?
  - Tot welk deel van geheugen heeft code toegang?

# RISC en CISC

- **CISC:** Complex Instruction Set Computer

- Veel verschillende en krachtige machinebevelen
- vb.: Intel x86 processoren

- **RISC:** Reduced Instruction Set Computer

- Beperkte set van machinebevelen
- Bevelen zijn minder krachtig
- Geen microbevelen 
- Meer registers, minder verbruik
- “John Cocke of IBM Research originated the RISC concept in 1974 by proving that about 20% of the instructions in a computer did 80% of the work.”

# RISC: ARM

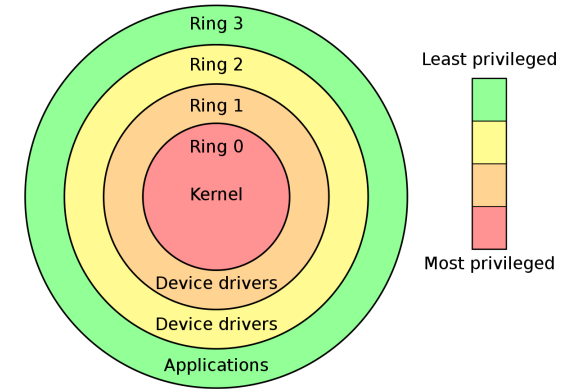
- Advanced RISC Machine
- SoC (System on a chip):
  - Belangrijkste componenten op 1 chip: CPU, GPU, Memory, Input/output interfaces, ...
- Meest gekende ARM chip fabrikant: Qualcomm
  - Bijna alle smartphone's hebben een ARM chip van Qualcomm
- Meest baanbrekende ARM chip: M1 / M2
  - Apple design
  - ARM delivers Instruction set

# RISC: ARM

- 31 GP registers

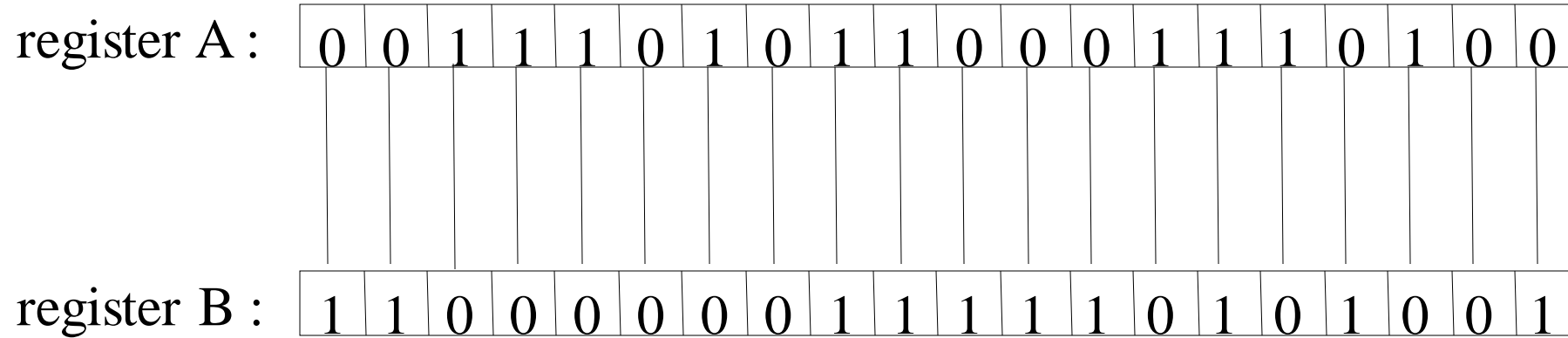
- 7 Modes of operation:

- Bepaald tot welke registers er toegang is en welke instructies mogelijk zijn.
  - User : unprivileged mode under which most tasks run
  - FIQ : entered when a high priority (fast) interrupt is raised
  - IRQ : entered when a low priority (normal) interrupt is raised
  - Supervisor : entered on reset and when a Software Interrupt instruction is executed
  - Abort : used to handle memory access violations
  - Undef : used to handle undefined instructions
  - System : privileged mode using the same registers as user mode





# Klok



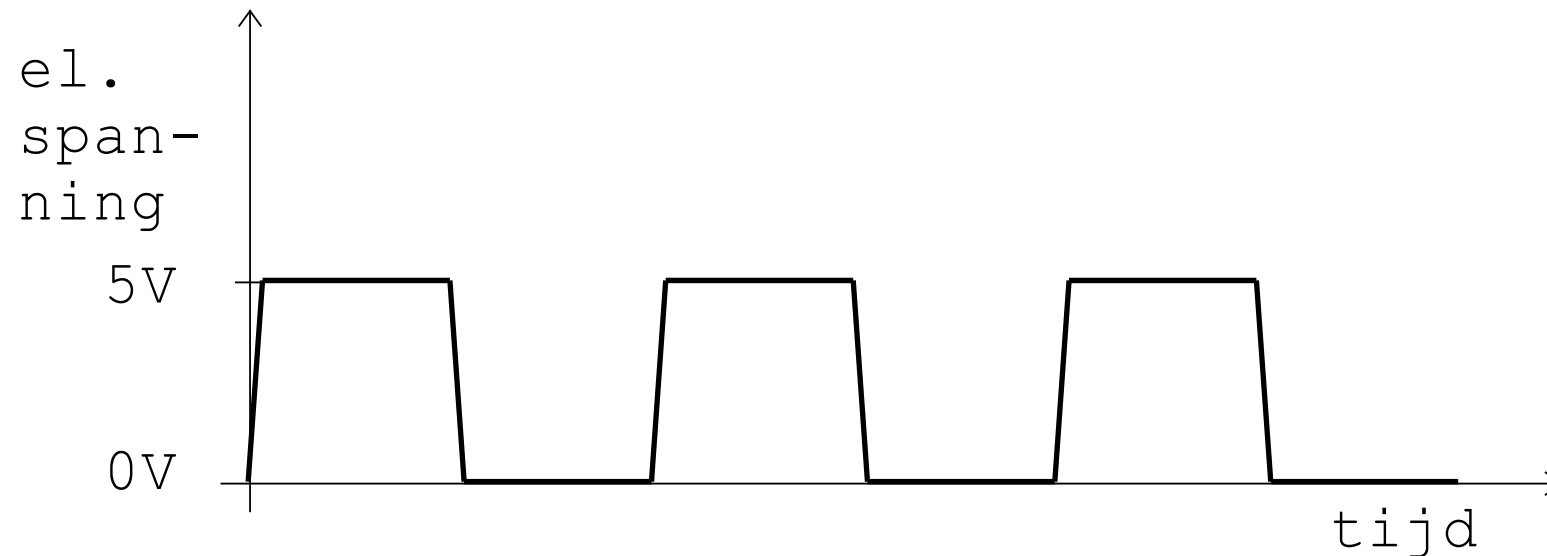
**Vraag:** wanneer is de berekening van een som klaar?

**Vraag:** hoe lang moet een verbinding gesloten zijn om te rekenen?

**Vraag:** wanneer is het ophalen van data uit het werkgeheugen klaar?

# Klok

- De 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



# Klok

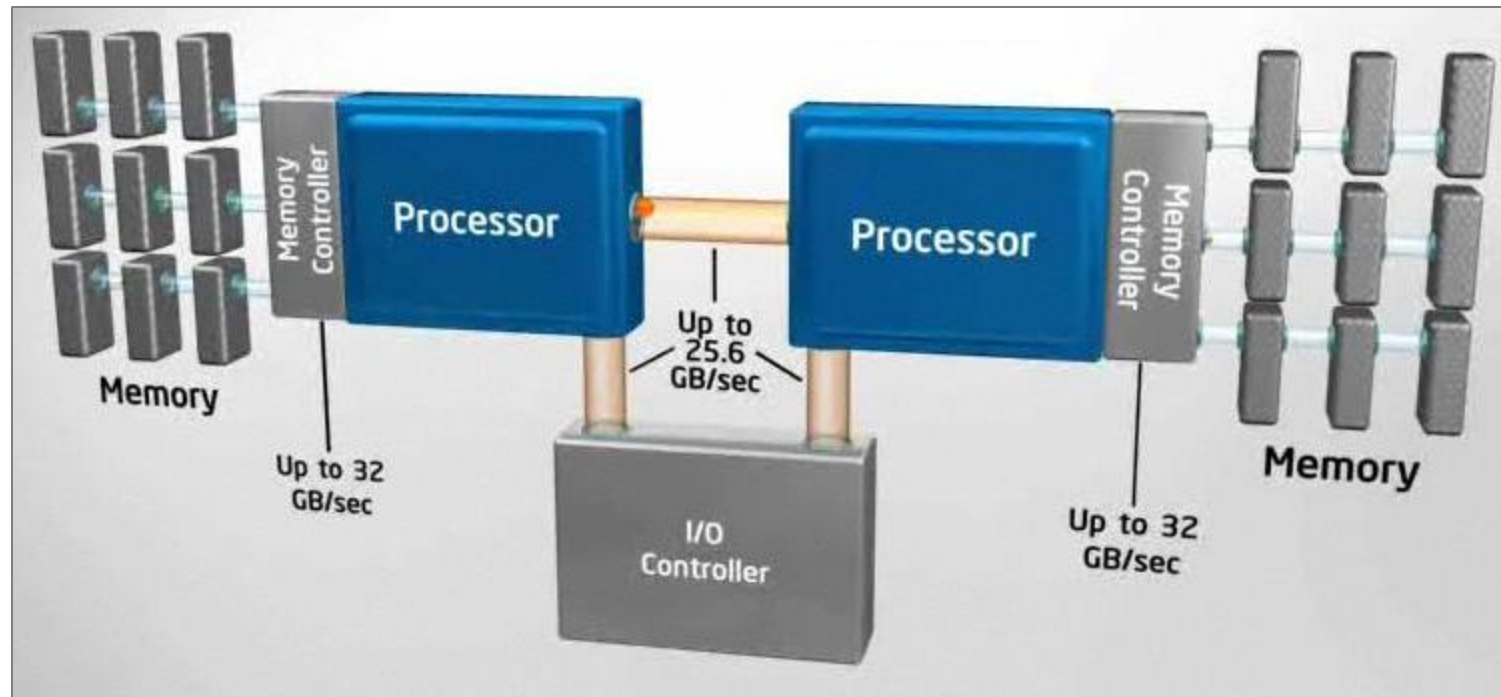
- Uitvoering van een bevel: in **stappen**, volgens het "tikken" van de klok, bvb.: `add eax, edx`
  1. `eax, edx` kopiëren op (verschillende) bus
  2. Getal van elke bus kopiëren naar één ingang van A.L.U.
  3. + bevel naar A.L.U.
  4. uitgang A.L.U. kopiëren naar bus
  5. bus kopiëren naar `eax`
- Klok als **dirigent**

# Klok

- Voor elk bevel: bepaald aantal klokcycli nodig
  - sprong-bevel: 1 cyclus
  - optelling/aftrekking: 3 cycli
  - vermenigvuldiging: 18 cycli
- Als de klok sneller tikt (en als de apparatuur meekan): snellere uitvoering van de bevelen.
  - Als in een fabriek de lopende band sneller loopt, wordt er dan meer geproduceerd?

# Meer dan één processor

- Hedendaagse server: meer dan één processor op het moederbord
  - Tegenwoordig: andere technologieën, bijv. Intel QuickPath



# Multiprocessing

- **Multiprocessing:** *“het simultaan uitvoeren van twee of meerdere programma’s op een computer met meer dan één CPU.”*
- Wat als we maar één CPU hebben? **Multiprogrammering.**
  - Het besturingssysteem laat de processor wisselen van programma (bijv. 100× per seconde)
  - Meerdere programma’s kunnen dan stapsgewijs uitgevoerd worden

# Nadeel Multiprogrammering

- Wisselen van programma's zorgt voor **overhead**
  - Registers e.d. moeten eerst weggeborgen worden in het geheugen, en hersteld worden vóór terugkeer
    - Op x64: 40+ registers
- Er zijn oplossingen voor: **Hyperthreading** en **multicore**

# Hyperthreading en Multicore

- **Hyperthreading**
  - Processor kan status van meerdere processen bijhouden
  - Er zijn meerdere sets van registers
- **Multicore**
  - Een stap verder dan hyperthreading
  - Er zijn meerdere sets van registers
  - Er zijn ook meerdere ALU's



# CPU

- Kan dingen doen...
  - Bevelen die een voor een uitgevoerd worden
  - Staan achter elkaar in het geheugen, bvb. A1 00 02 00 00 03 05 40 02 00 00  
A3 00 03 00 00 2B 05 00 02 00 00 03 05 20 03 00 00
- ... met data
  - In registers (EAX, EBX, ECX, EDX, ...) of het werkgeheugen
- Door de uitvoering van bevelen verandert de data in de registers en/of het werkgeheugen

# Assembly

- **Lagere** programmeertaal:
  - Mnemotechnische functiecode
  - Mnemotechnische geheugen adressen (= variabelen)
  - **1 bevel = 1 machine instructie**
    - Kan onmiddellijk uitgevoerd worden
- **Hogere** programmertaal:
  - Mnemotechnische functiecode
  - Mnemotechnische geheugen adressen (= variabelen)
  - **1 bevel = meerdere machine instructies**
    - Moet nog 'vertaald' worden door compiler

# Code Blocks

- Assembly programma = vaste structuur
- 3 code blocks:
  - data sectie (niet verplicht)
  - bss sectie (niet verplicht)
  - text sectie (verplicht)

# Vb

```
%include "io.inc"
```

```
section .data
```

```
vier: dd 1
```

```
miljard: dd 1000000000
```

```
viern: dd 2000000000
```

```
section .bss
```

```
help: resd 1
```

```
section .text
```

```
global _start
```

```
_start:
```

```
    mov ebp, esp
```

```
    xor eax, eax
```

```
    mov eax, [viern]
```

```
    imul dword [vier]
```

```
    mov eax, [viern]
```

```
    imul dword [vier]
```

```
    mov [help], edx
```

```
    PRINT_UDEC 4, [help]
```

```
    ret
```

# Data sectie

Directive = hulp voor assembler,  
geen echte code

- Wordt aangegeven door: 'section .data'
- Bevat alle geïnitieerde variabelen
  - = stukje geheugen reserveren en reeds een startwaarde geven

Directive	Doel	Gebruikt geheugen
DB	Definiëer Byte	alloceerd 1 byte
DW	Definiëer Word	alloceerd 2 bytes
DD	Definiëer Dubbelwoord	alloceerd 4 bytes
DQ	Definiëer Quadwoord	alloceerd 8 bytes
DT	Definiëer Tien Bytes	alloceerd 10 bytes

```
section .data
choice: DB 'y'
number: DW 12345
neg_number: DW -12345
number2: DD 23456463
```

# BSS sectie

- Wordt aangegeven door: 'section .bss'
- Bevat alle niet geïnitieerde data
  - = stukje geheugen reserveren en **GEEN** startwaarde geven

Directive	Doel
RESB	Reserveer 1 byte
RESW	Reserveer 1 woord
RESD	Reserveer 1 Dubbelwoord
RESQ	Reserveer 1 Quadwoord
REST	Reserveer Tien Bytes

```
section .bss
    sign: RESB 1
    number: RESW 1
    big_number: RESD 1
```

# Text sectie

- Bevat de eigenlijk code
- Wordt aangegeven door: 'section .text'
- Moet steeds volgende code bevatten

```
global _start
```

```
_start:
```

```
....
```

```
....
```

```
....
```

```
mov EAX, 1 ;System call number (sys_exit)
```

```
int 0x80 ;Perform System Call
```

# Constanten

- Constanten in assembly worden gedefinieerd door **EQU**
- Staan in 'section .data'
- EQU:
  - Neemt geen geheugenplaats in tijdens uitvoer
  - Waarde wordt letterlijk vervangen in code tijdens assembleren

```
section .data
    star: EQU '*'
    number: EQU 5128
    big_number: EQU -2423523453
```



# MOV

- CPU kan alleen werken met registers.

==> Waarden kopiëren vanuit geheugen naar register en omgekeerd.

mov <reg>,<reg>

mov <reg>,<mem>

mov <mem>,<reg>

mov <reg>,<const>

**Werking:** Kopieert inhoud van tweede operand naar locatie van eerste operand

# ADD

- Werking: Telt de waarde van de tweede operand bij de waarde van de eerste operand en plaatst het resultaat in de eerste operand

add <reg>,<reg>

add <reg>,<mem>

add <reg>,<con>

# SUB

- Werking: trekt de waarde van de tweede operand af van de waarde in de eerste operand. Het resultaat wordt in de eerste operand geplaatst.

sub <reg>,<reg>

sub <reg>,<mem>

sub <reg>,<con>

# Vermenigvuldigen

$$\begin{array}{r} 82 \\ \times 45 \\ \hline 3690 \end{array}$$

$$\begin{array}{r} 32\text{-bit} \\ \times 32\text{-bit} \\ \hline 64\text{-bit} \end{array}$$

Voorbeeld:

```
...  
getal1: dd 7  
getal2: dd 5  
...  
mov eax,[getal1]  
imul dword [getal2]  
...
```

Past niet in een register

# Effect

Na vertaling :

getal1: dd 7

getal2: dd 5

getal1  
00 00 00 07

getal2  
00 00 00 05

Na uitvoering van:

mov eax, [getal1]

imul dword [getal2]

EAX :

00 00 00 07

EDX :

00 00 00 00

EAX :

00 00 00 23

# imul dword [adres]

- **adres:** adres van een dubbelwoord
- inhoud van **EAX** wordt vermenigvuldigd met de inhoud van het dubbelwoord
  - Alleen de inhoud van EAX kan vermenigvuldigd worden
- het product (resultaat) komt in het REGISTERPAAR **(EDX,EAX)**

**Opmerking:** *er zijn varianten  
maar die doen we niet!*

# imul dword [adres]

- Resultaat is altijd **64 bits** (=16 hex. Cijfers of 8 bytes)
  - De eerste helft hebben we meestal niet nodig; toch is hij er.
- Dword-aanduiding geeft aan dat [adres] een 32-bit getal bevat
  - Lang geleden: (8 bits)\*(8bits) → **imul byte** [adres]
  - Minder lang geleden: (16 bits)\*(16 bits) → **imul word** [adres]
  - Kan nog. Daarom: **dword**

# Nog een voorbeeld

neg: dd -3

pos: dd 5

mov eax,[neg]

imul dword [pos]

neg  
FF FF FF FD

pos  
00 00 00 05

EAX :  
FF FF FF FD

EDX :      EAX :  
FF FF FF FF    FF FF FF F1



# Vermenigvuldiging

- Eerst één van de factoren in EAX
- Dan: `imul dword [adres]`  
(adres van de 2<sup>o</sup> factor)
- $$\begin{array}{r} \text{EAX} \\ * \text{dub.w.} \\ \hline (\text{EDX}, \text{EAX}) \end{array}$$

**Merk op:** *als het resultaat ligt tussen -2,1...miljard en +2,14... miljard, dan kunnen we verder werken met de inhoud van EAX.*

# Deling

- Waarom zeggen we: 10 gedeeld door 2 is 5? Omdat  $5 * 2 = 10$ .

$$\begin{array}{r} \text{EAX} \\ * \text{ dub.w.} \\ \hline (\text{EDX}, \text{EAX}) \end{array}$$

$$\begin{array}{r|l} (\text{EDX}, \text{EAX}) & \text{dub.w.} \\ \hline & \text{EAX} \\ \hline & \text{EDX} \end{array}$$

# Deling

```
nul: dd 0
```

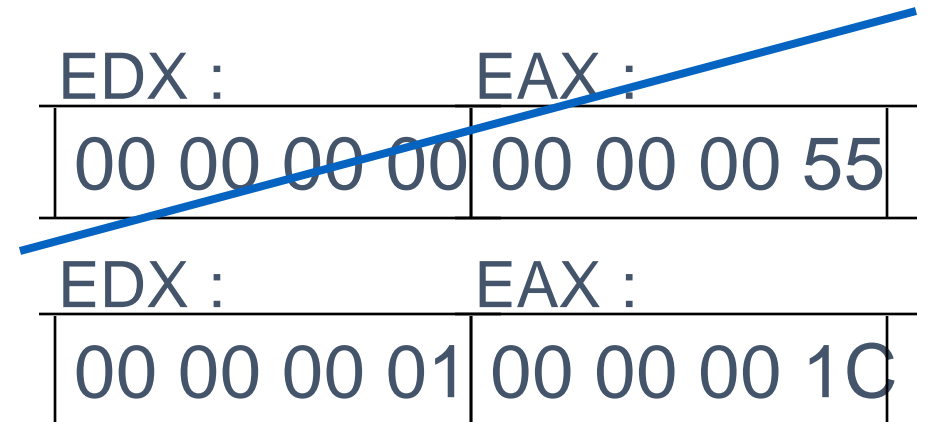
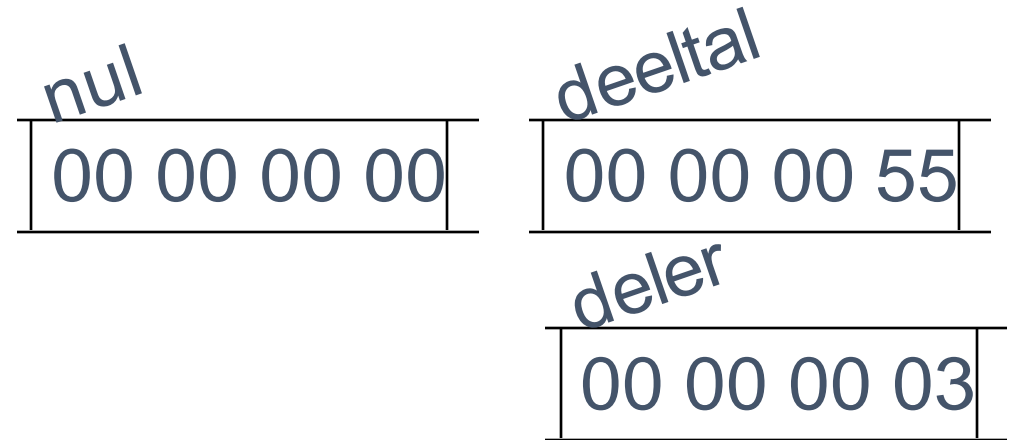
```
deeltal: dd 85
```

```
deler: dd 3
```

```
mov edx, [nul]
```

```
mov eax, [deeltal]
```

```
idiv dword [deler]
```



# idiv dword [adres]

- inhoud van REGISTERPAAR (EDX,EAX) wordt als één getal, gedeeld
- **adres**: adres van de deler (een dubbelwoord)
- het quotiënt komt in **EAX**
- de rest komt in **EDX**

**Opgelet:** *wanneer het resultaat niet in EAX  
geraakt, crasht het programma!*

# Deling Negatief Getal

```
nul: dd 0  
deeltal: dd -20  
deler: dd 3
```

```
mov edx, [nul]  
mov eax, [deeltal]  
idiv dword [deler]
```

**Merk op:** 5555554E is  
een positief getal!

nul	00 00 00 00	deeltal	FF FF FF EC
deler	00 00 00 03	EDX :	00 00 00 00
		EAX :	FF FF FF EC
		EDX :	55 55 55 4E

# Deling Negatief Getal

- In (EDX,EAX) moet de correcte voorstelling van het deeltal staan.  
Welk?
  - 00 00 00 00 als het deeltal niet negatief is
  - FF FF FF FF als het deeltal wel negatief is
- Hoe doen we dit? Vermenigvuldig met 1.

# Deling Negatief Getal

een: dd 1

deeltal: dd -20

deler: dd 3

mov eax, [deeltal]

imul dword [een]

idiv dword [deler]



# Deling

- Deling van positief of negatief getal
  - deeltal in **EAX**
  - **imul dword** [adres\_getal1]
  - **idiv dword** [adres\_deler]
- Ben je zeker dat het deeltal  $\geq 0$  is?
  - 0 naar EDX is ook goed