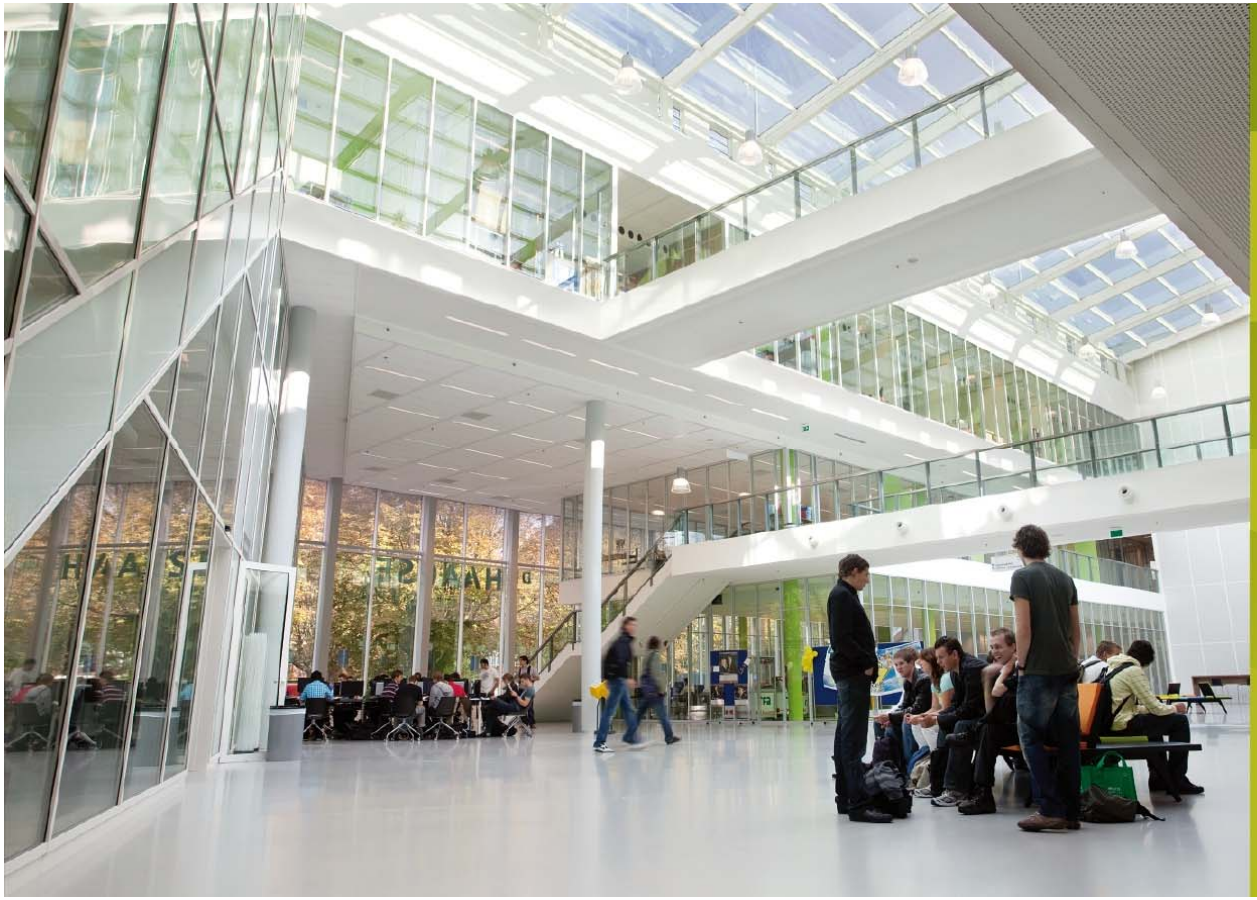


Academie voor Technology, Innovation & Society Delft
Academie voor ICT & Media

Inleiding microcontrollers

Week 2 – Introductie microcontroller
Jesse op den Brouw (met dank aan Ben Kuiper)
INLMIC/2018-2019

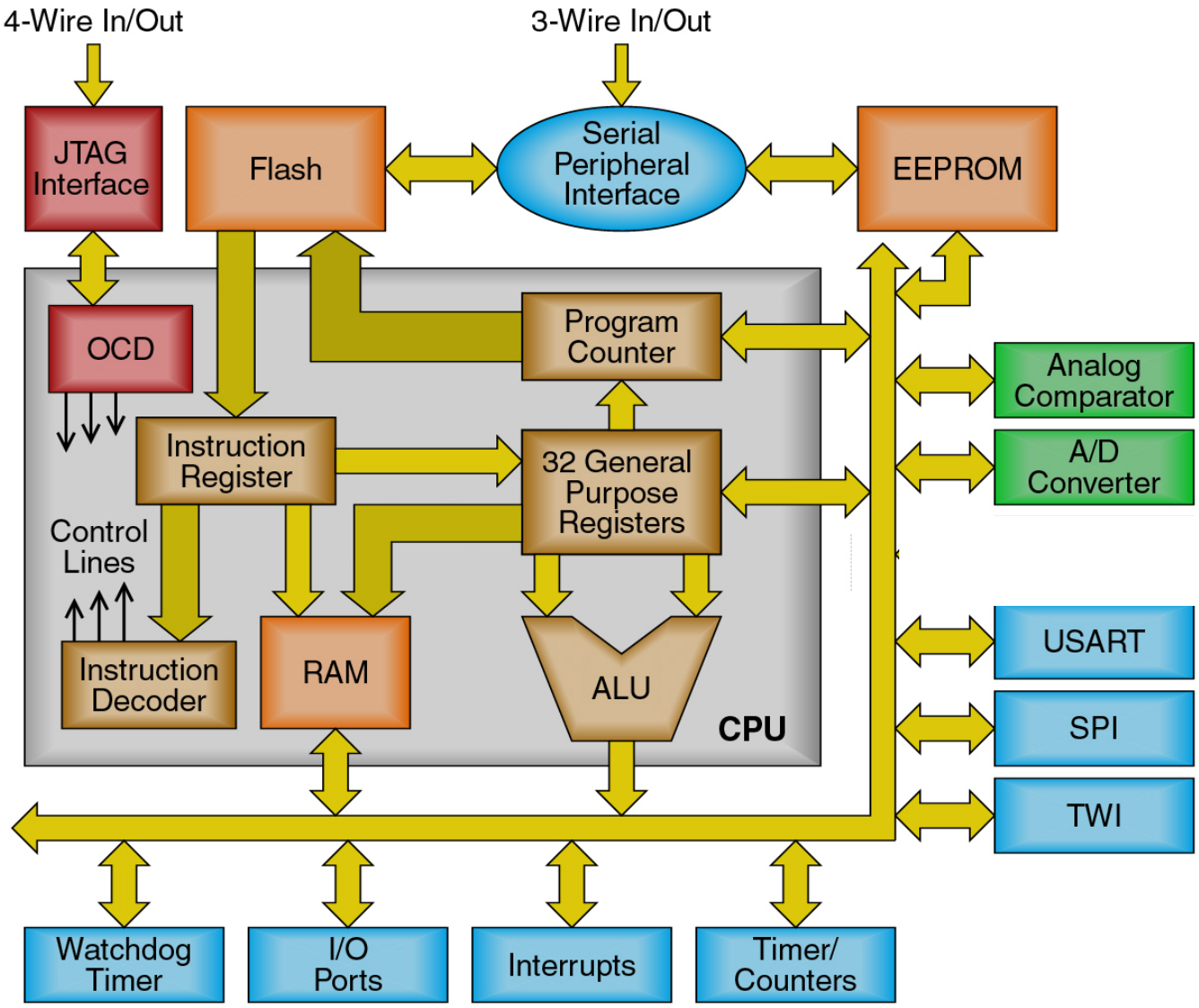
HAAGSE HOGESCHOOL

Week 2

- Datapad AVR-CPU
- Registers
- ALU
- Programmageheugen
- Fetch-decode-execute
- Instructies
- Assembler basics

DE HAAGSE HOGESCHOOL

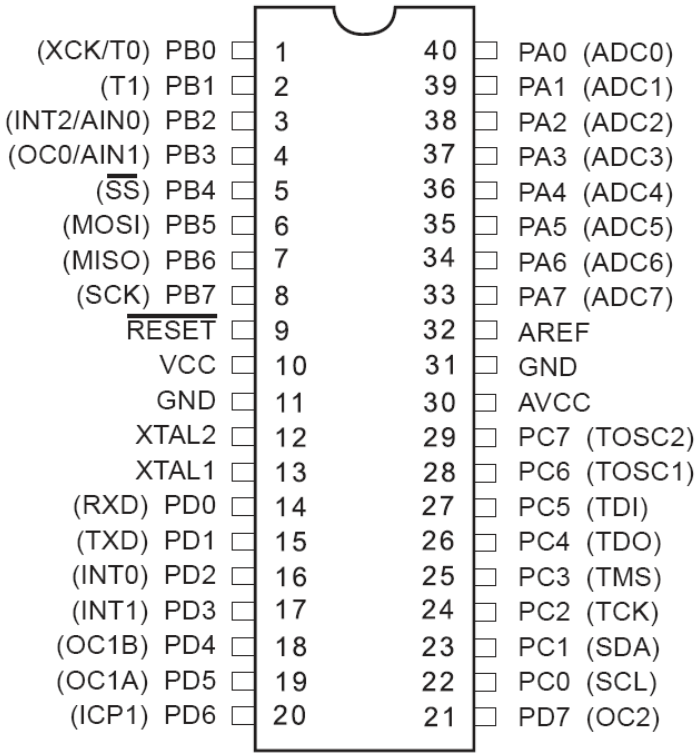
Blokdiagram ATmega32



ATmega32 Features

- 131 instructies
- 32 KB Flash ROM programmageheugen
- 2 KB Intern SRAM
- 1024 Bytes EEPROM

- Twee 8-bit timers
- Een 16-bit timer
- Two Wire Interface = I^2C
- 8x1 10 bits ADC
- USART
- SPI
- 32 I/O lijnen

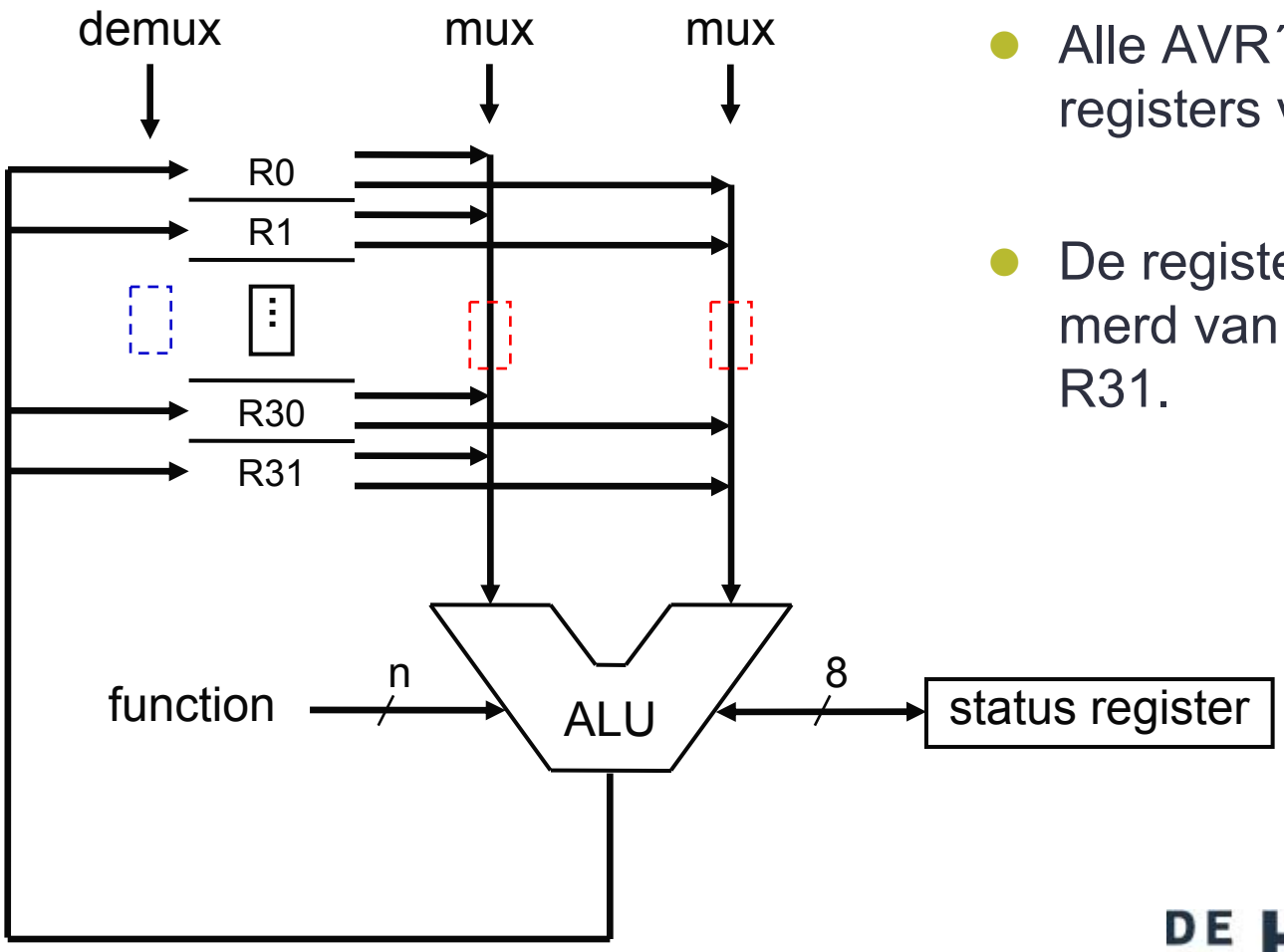


DE HAAGSE HOGESCHOOL

Datapad AVR-CPU

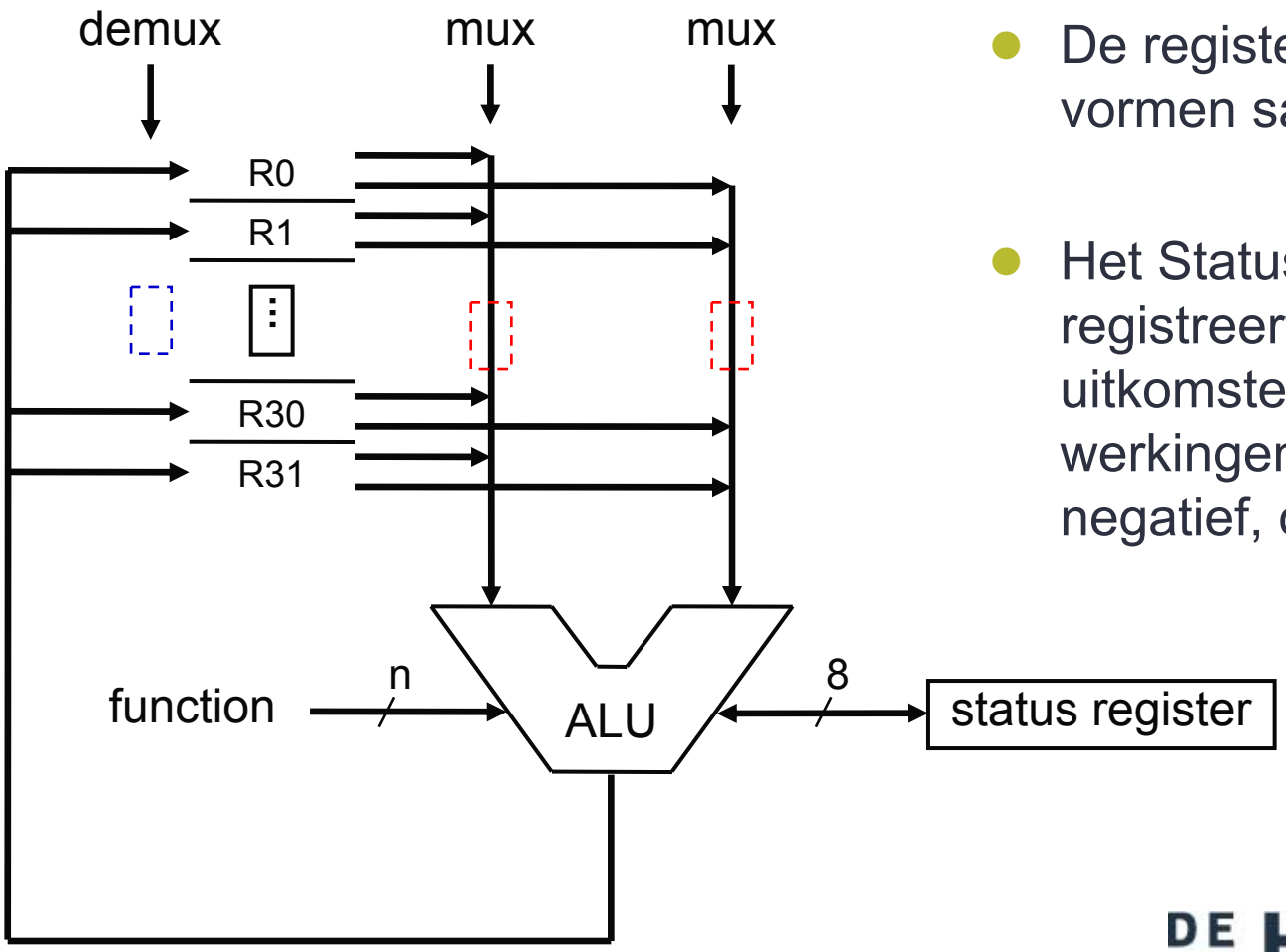
- Een microcontroller CPU verwerkt data door middel van instructies.
- Een microcontroller heeft dus (data)opslag.
- De data wordt verwerkt in een eenheden van 8 bits.
- Verwerken kan zijn: rekenkundige operaties (optellen, aftrekken) en logische operaties (and, or, not, schuiven).
- Dit verwerken wordt gerealiseerd door de ALU: Arithmetic and Logic Unit

Datapad AVR-CPU



- Alle AVR´s hebben 32 registers van 8 bits breed.
- De registers zijn genummerd van R0 tot en met R31.

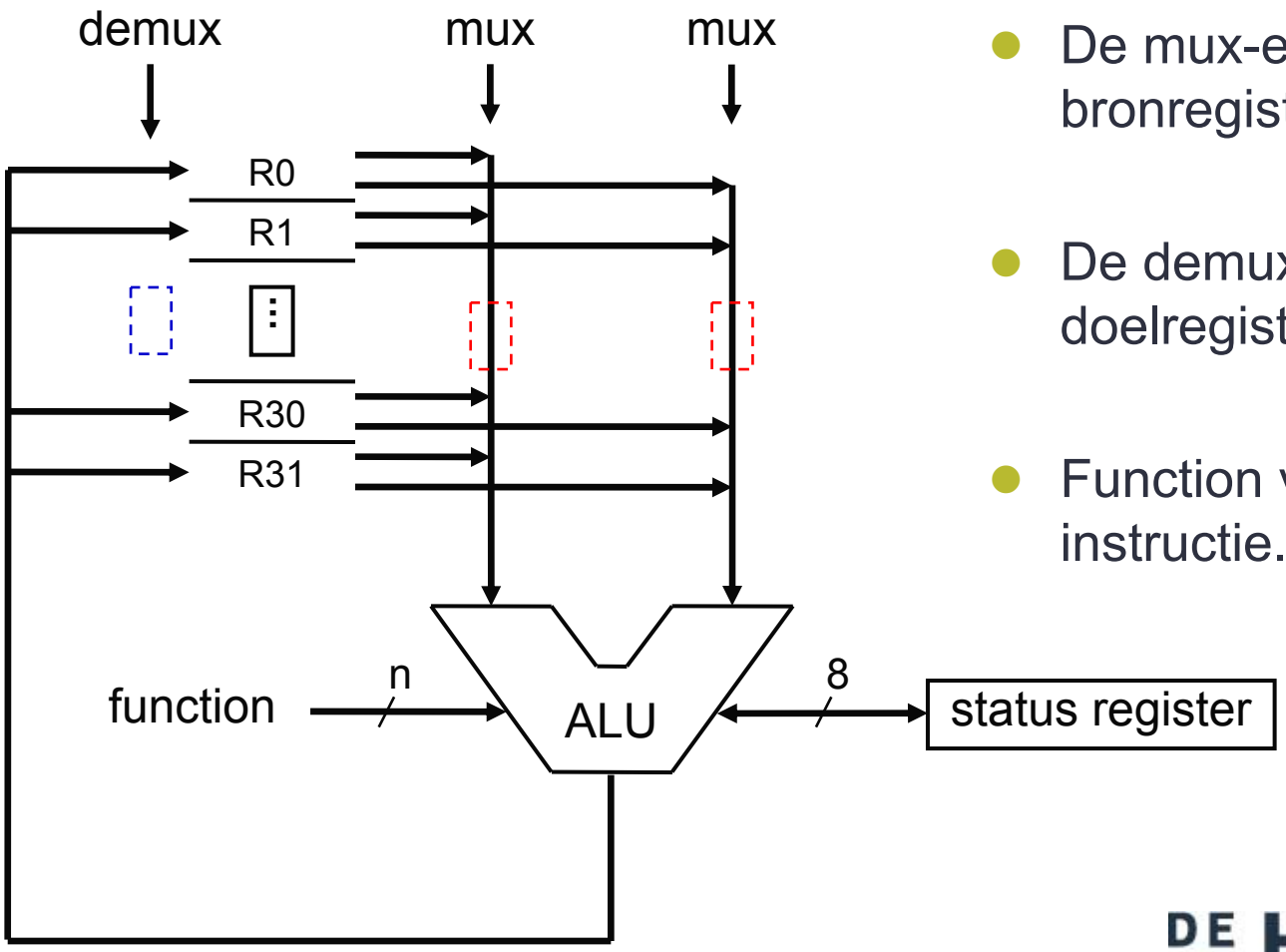
Datapad AVR-CPU



- De registers en de ALU vormen samen het datapad.
- Het Status Register registreert belangrijke uitkomsten van de bewerkingen zoals carry, negatief, overflow en nul.

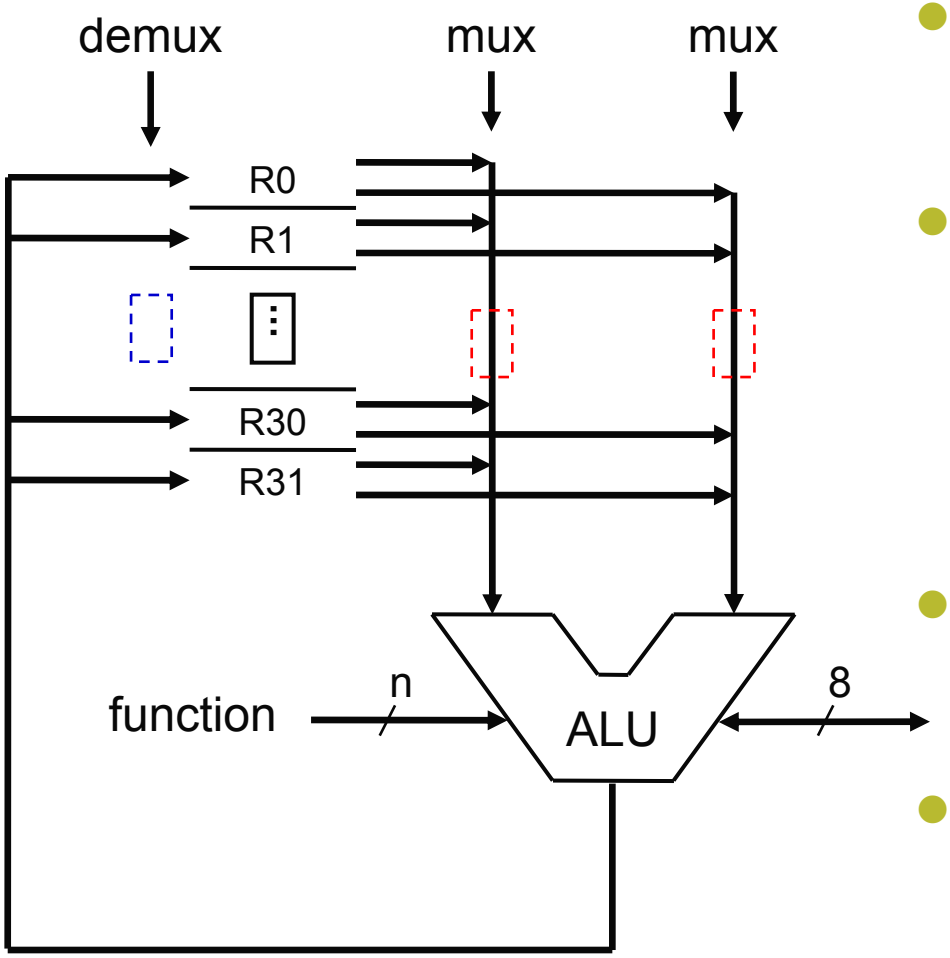
DE HAAGSE HOGESCHOOL

Datapad AVR-CPU

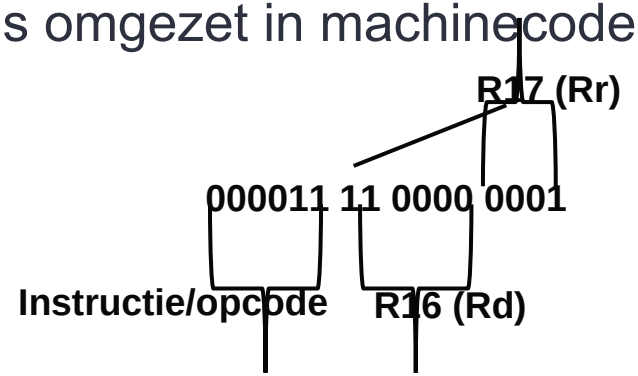


- De mux-en selecteren de bronregisters.
- De demux selecteert het doelregister.
- Function volgt uit de instructie.

Datapad AVR-CPU

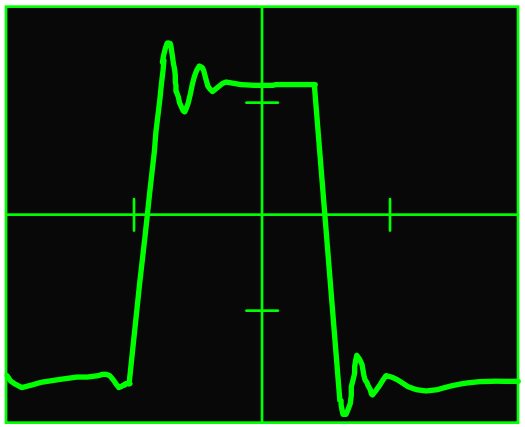


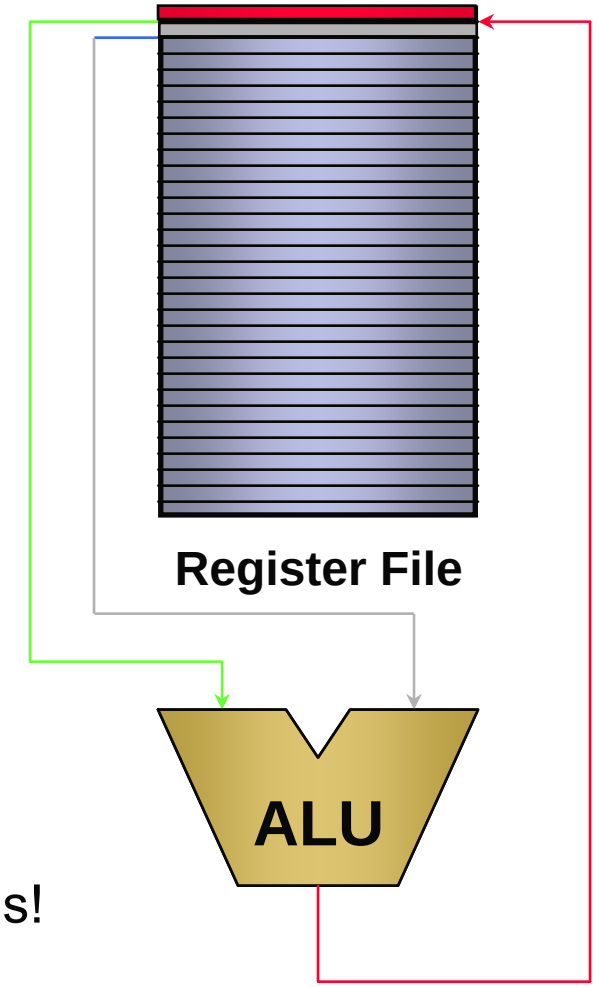
- Voorbeeld instructie:
  ADD R16,R17
  (R16=R16+R17)
- Is omgezet in machinecode:



- De opcode bestuurt de controlesignalen van de ALU (indirect, later meer hierover)
- De registervelden besturen de multiplexers

Single Cycle Instruction Execution

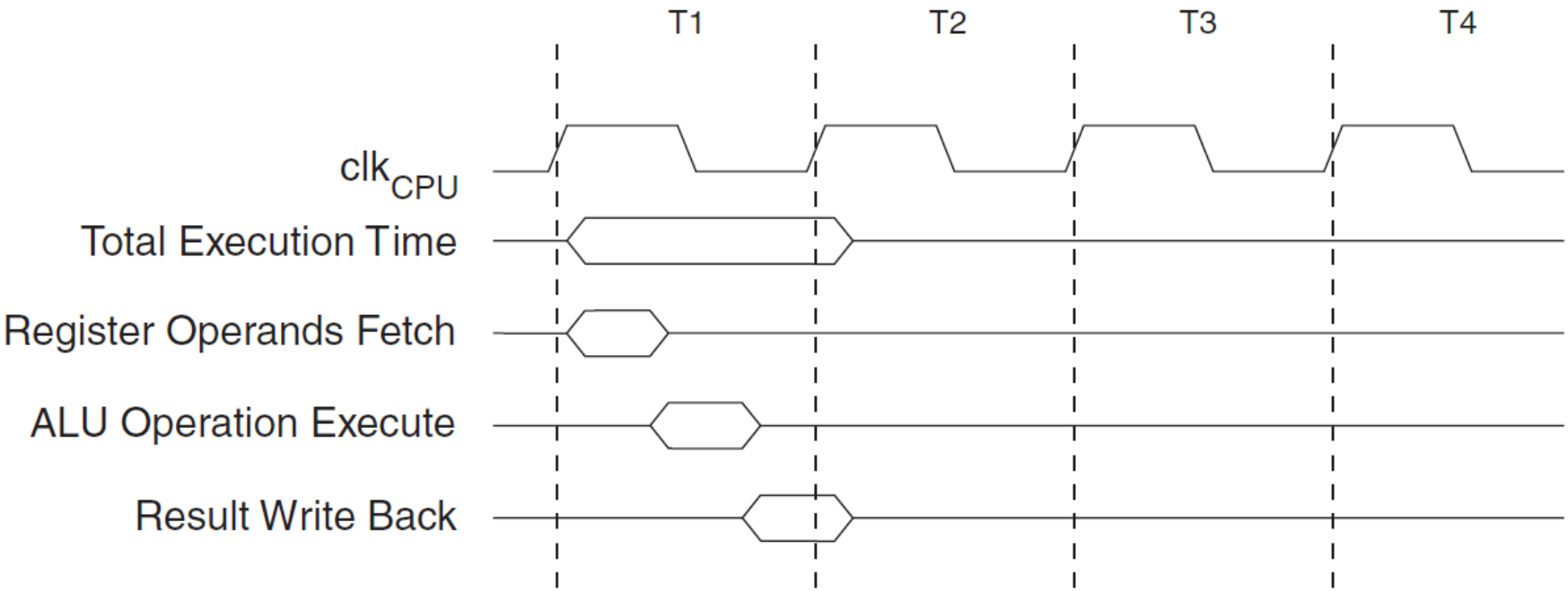




Registeroperaties duren precies één klokpuls!

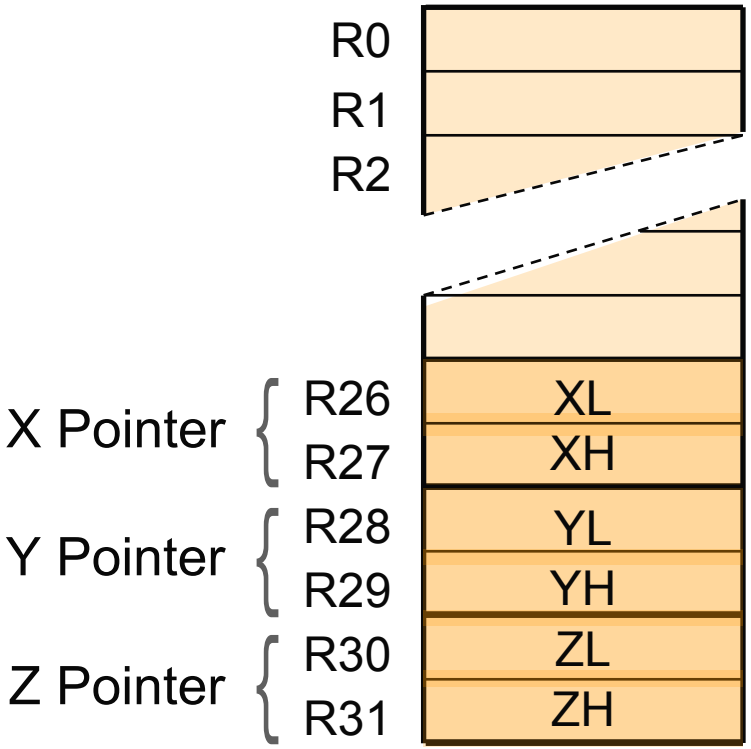
Single Cycle Instruction Execution

- Registeroperaties duren precies één klokpuls.



AVR Register File

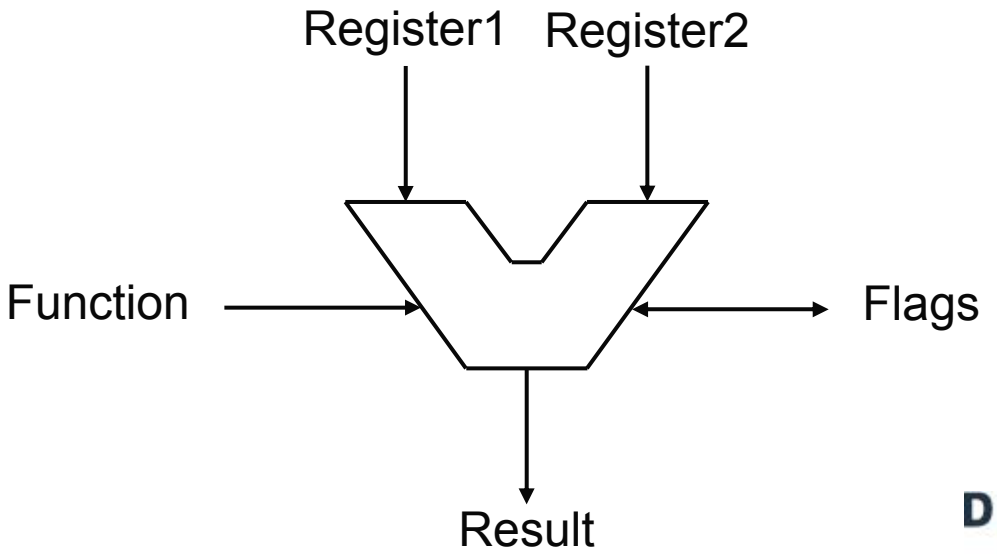
- 32 8-bit General Purpose Registers
  - Alle hebben direct toegang tot de ALU
- 3 registerparen werken als 16-bit pointers naar SRAM-geheugen.
- Een aantal instructies kan alleen werken met R16 t/m R31.
- Een aantal instructies werkt met R0 en R1.



AVR ALU

- De ALU is een combinatorische schakeling die een aantal rekenkundige en logische bewerkingen kan uitvoeren.

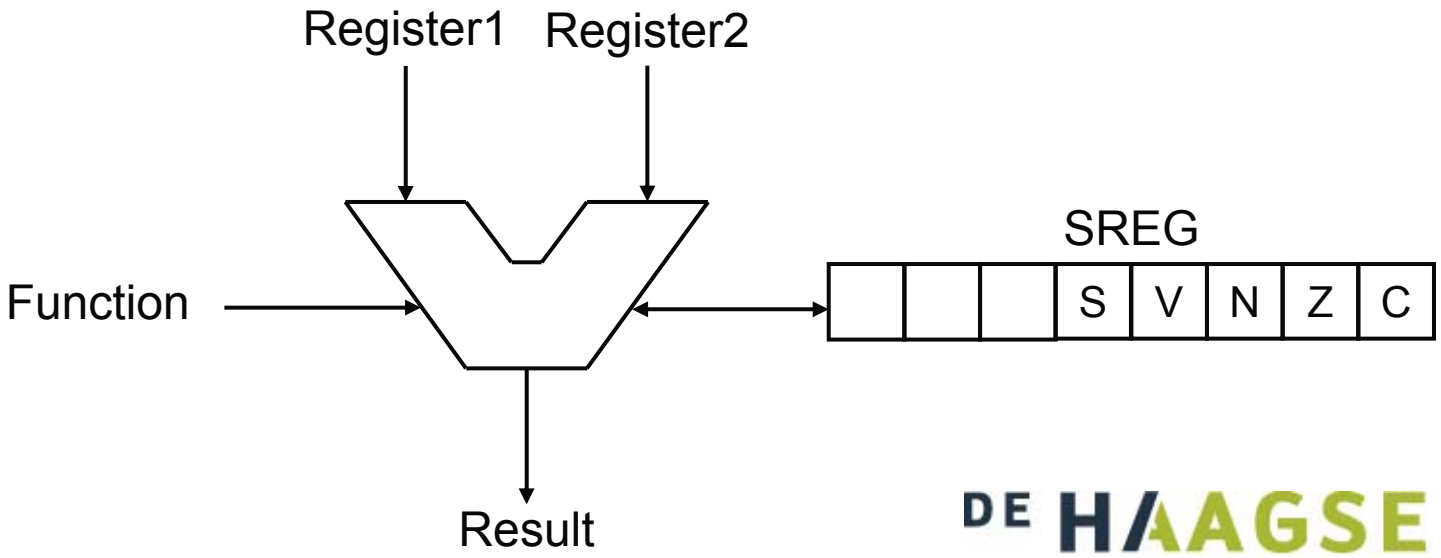
- Bewerkingen: optellen, aftrekken, vermenigvuldigen, delen, bitwise AND/OR/XOR, inverse, schuiven, roteren, *pass thru*.



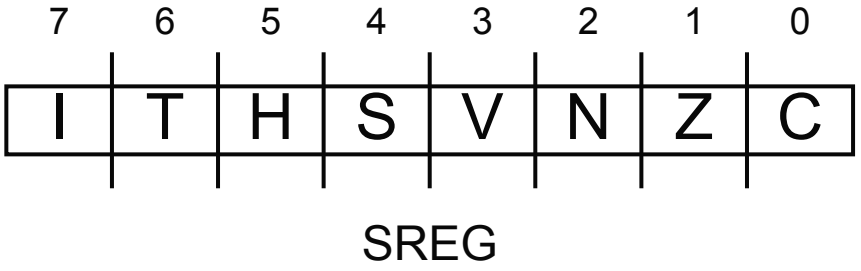
AVR ALU

- Interessante informatie over de berekeningen worden weergegeven door de *flags*.

- De flags worden gecombineerd tot een register. Deze wordt Status Register genoemd.



AVR status register



- Het Status Register is een 8-bit register en is opgenomen in het datapad.

- Status Register Flags: Interrupt Enable, T bit storage, Half Carry, Sign, oVerflow, Negative, Zero, Carry.

AVR status register

- De C-flag wordt gebruikt in multi-byte bewerkingen en geeft overflow bij unsigned bewerkingen aan.
- De Z-flag wordt gezet wanneer het resultaat van een bewerking nul is, anders wordt het gewist.
- De N-flag wordt gezet wanneer het resultaat van een bewerking negatief is (msb =1), anders wordt het gewist.
- De V-flag wordt gezet als een 2's complement bewerking het tekenbit foutief heeft gewijzigd.

AVR status register

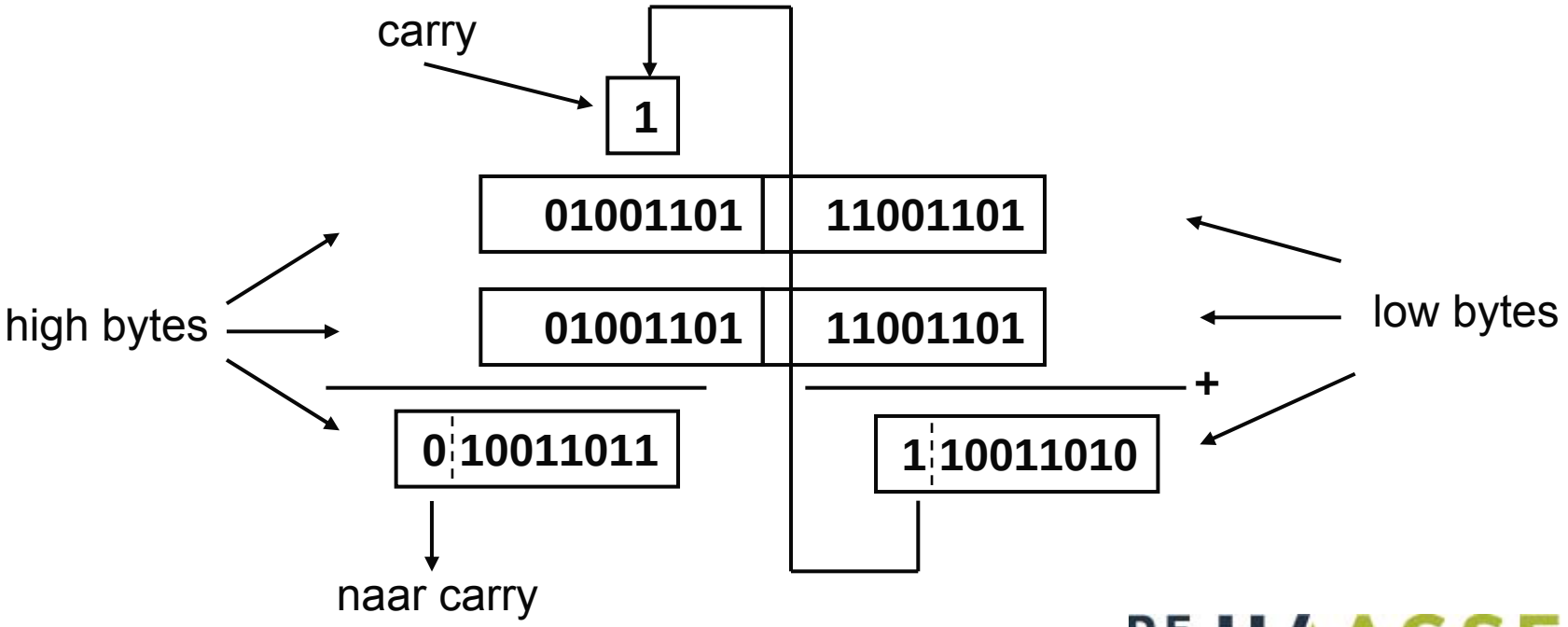
- De S-flag geeft het teken aan van het resultaat van een optel/aftrekbewerking (exor van de N-flag en de V-flag).
- De H-flag wordt gebruikt bij BCD-rekenkunde en geeft aan of het lage nibble een overflow heeft gehad.
- De T-flag is een vlag die door de gebruiker gebruikt kan worden d.m.v. assemblerinstructies.
- De I-flag geeft aan of interrupts worden gehonoreerd (zie week 5).

AVR ALU

- Met behulp van de flags kunnen beslissingen genomen worden.
- Voorbeeld bij instructie SUB R16, R17 (R16=R16-R17):
- Als Register16 = Register17, dan wordt Z = 1
- Als Register16 < Register17, dan wordt N = 1
- Als Register16 > Register17, dan wordt Z = 0 en N = 0
- …

Multibyte optelling

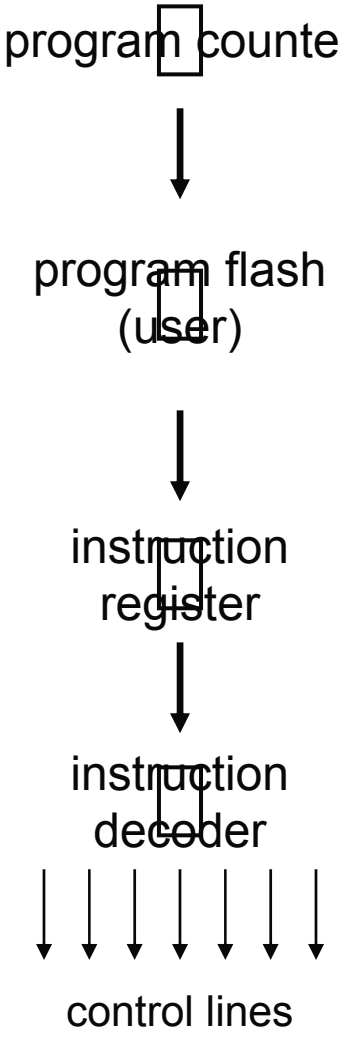
- De Carry flag wordt gebruikt bij multi-byte bewerkingen. Zo kunnen 16- en 32-bit optellingen worden gedaan met een 8-bit processor.



AVR programmageheugen

- Een microcontroller is een kleine computer met gespecialiseerde I/O.
- Een microcontroller bevat dus een programma (lijst van instructies).
- Een microcontroller heeft dus programmageheugen waar deze instructies zijn opgeslagen.
- Het programmageheugen is uitgevoerd als Flash en is niet vluchtig (non volatile).

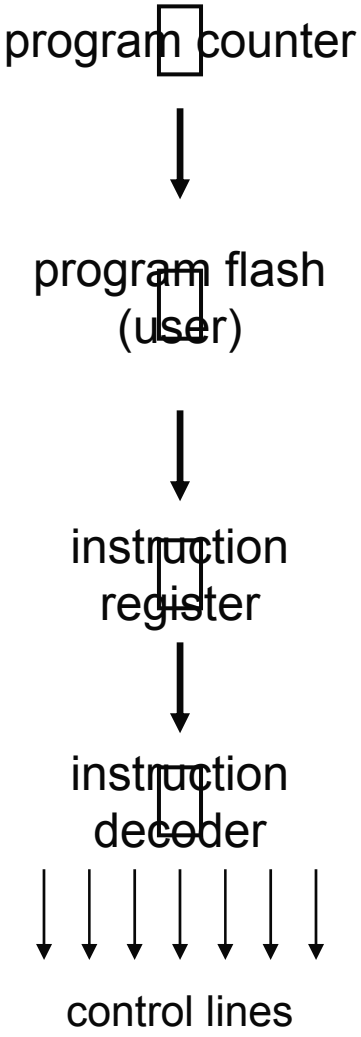
AVR programmageheugen



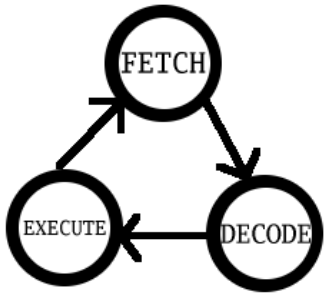
- De programmateller (program counter) houdt bij waar de microcontroller met het uitvoeren van de instructies is.
- De uit te voeren instructie wordt geladen in het instructieregister (instruction register). Deze bevat dus de huidige instructie die wordt uitgevoerd.
- De instructiedecoder (instruction decoder) decodeert de bitpatronen in het instructie-register en stuurt de logica verder aan (control lines).

DE HAAGSE HOGESCHOOL

AVR programmageheugen



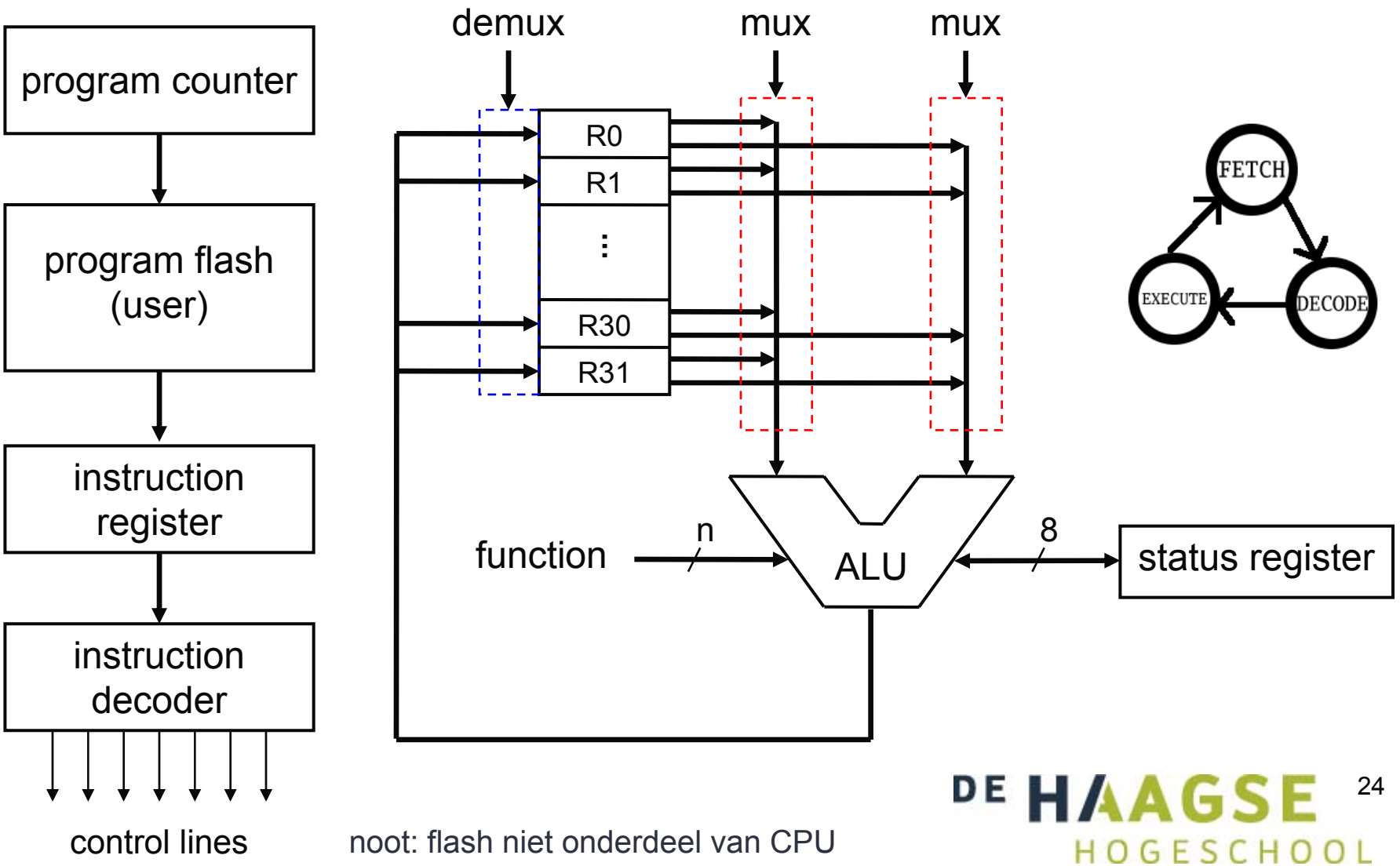
- Het uitvoeren van de instructies gaat d.m.v de Von Neumann-cyclus:
- Ophalen instructie (fetch)
- Decoderen instructie (decode)
- Uitvoeren instructie (execute)



Programmaverwerking

- Programmateller start bij adres 0.
- De instructie op adres 0 wordt opgehaald.
- De instructie wordt uitgevoerd.
- De programma teller wordt met 1 opgehoogd.
- De volgende instructie wordt opgehaald.
- enz.

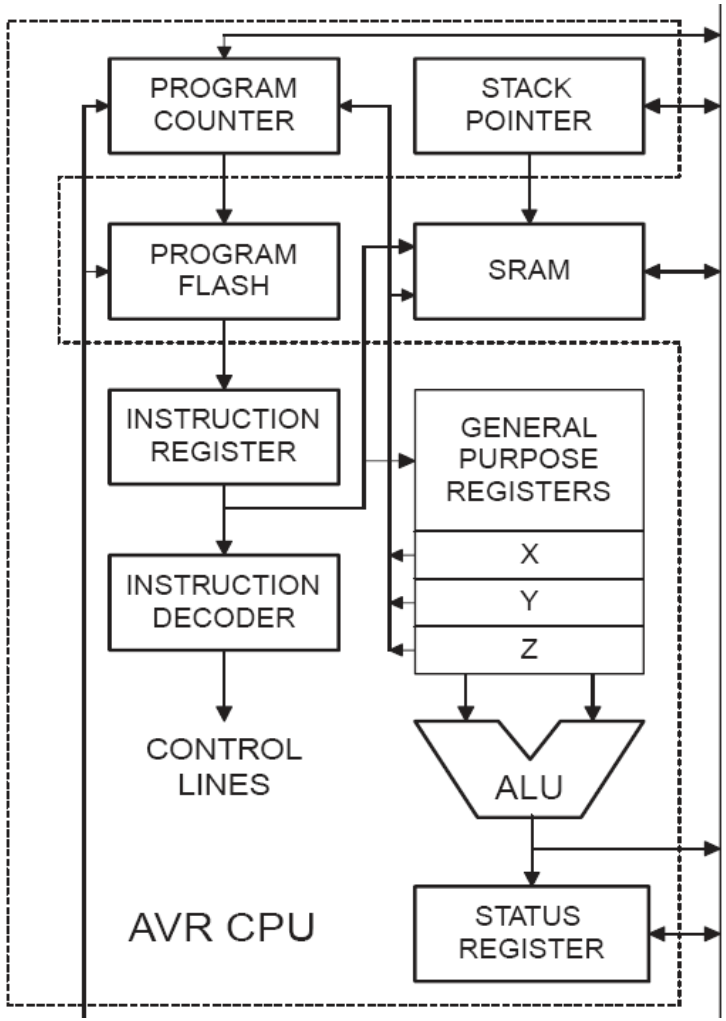
AVR CPU



noot: flash niet onderdeel van CPU

DE HAAGSE HOGESCHOOL

AVR CPU



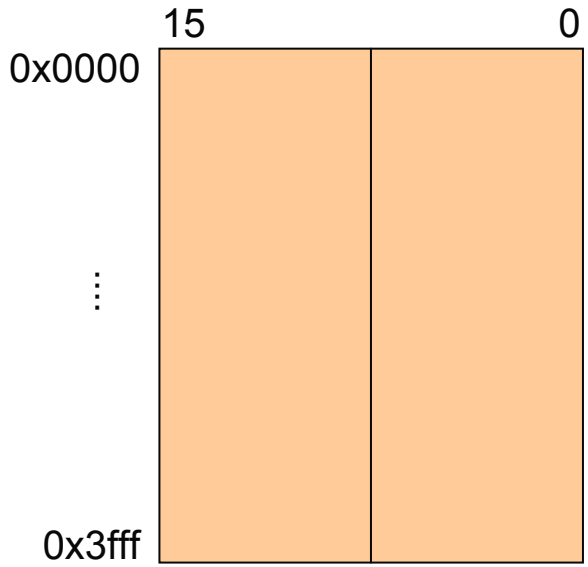
AVR CPU bestaat uit:

- registerfile, ALU en Status Register
- Program Counter, Instruction Register en Instruction Decoder
- Stack Pointer (week 4)
- Program Flash en SRAM behoren niet tot de CPU

DE HAAGSE HOGESCHOOL

Flash ROM

- De ATmega32 heeft 32 kbytes Flash ROM.
  - PC is 22 bits. ATmega32 gebruikt de onderste 14 bits.
- Instructies zijn georganiseerd in 16-bit words of 32-bit double words.
- Er zijn dus 16 k words aan instructies mogelijk.



adressen in words!

Plaatsing bytes (8-bits) in woorden (16-bits)

- De machinecode in de Flash ROM is 16 bits.
- Een byte is 8 bits ➔ 2 manieren van opslaan.
- Little Endian en Big Endian*).
- Bijv. machinecode voor het zenden van de inhoud van R16 naar PORTB is (16 bits):

  1011 1011 0000 1000 = 0xBB08

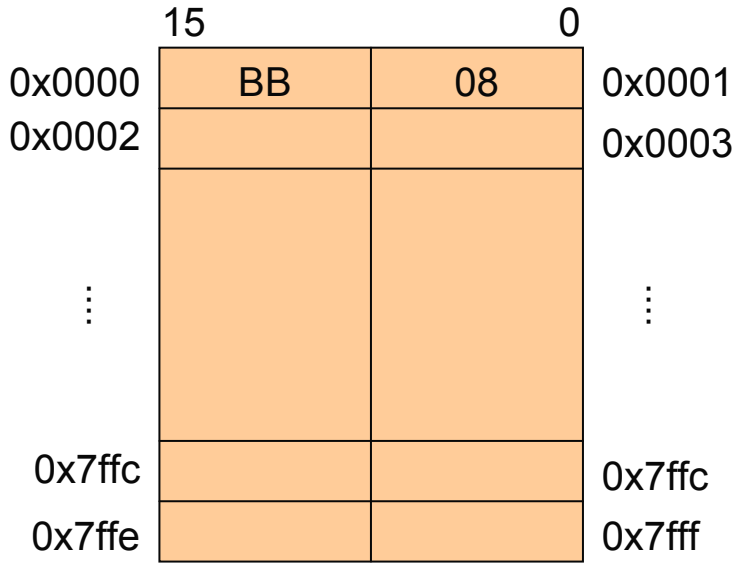
- MSB byte (0xBB) gaat in het laagste adres.
- LSB byte (0x08) het daarop volgende hogere adres.
- Dit noemen we Big Endian.

*) http://en.wikipedia.org/wiki/Endianness

DE HAAGSE HOGESCHOOL

Plaatsing bytes (8-bits) in woorden (16-bits)

- Hieronder een plaatje betreffende de *memory map*.
- Let hier op wanneer je gebruik maakt van constanten in Flash-ROM.



adressen in bytes!

SRAM

- De ATmega32 heeft 2048 bytes statisch RAM.
  - 16 bits adres.
- Inhoud is vluchtig (*volatile*), verdwijnt na spanningverlies.
- Data-opslag:
  - Gegevens tijdens draaien applicatie.
- Manipulatie via registers.

I/O

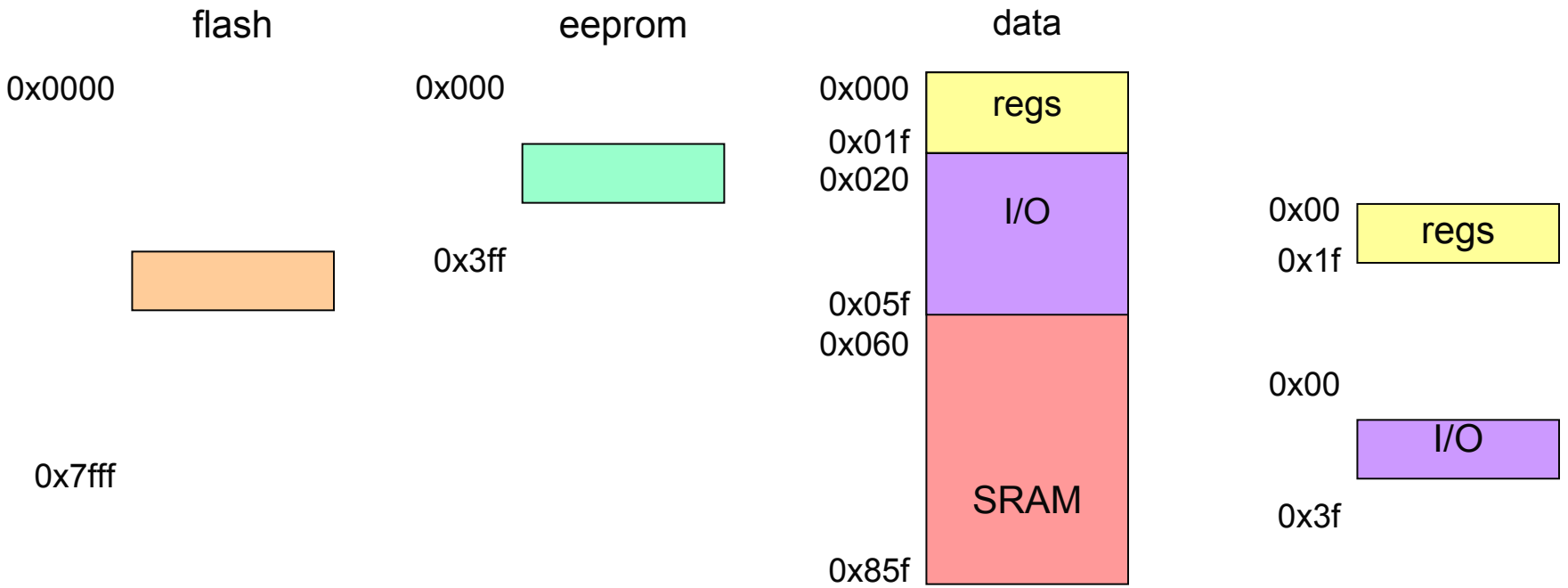
- De ATmega32 heeft 64 I/O-registers.
- Hiermee kan de hardware worden gemanipuleerd.
  - I/O ports, Timer/Counter, Usart, ADC, ...
- Inhoud is vluchtig, beginwaarden na power-on/reset.
- Speciale instructies (in, out).
- Wordt behandeld in week 3.

EEPROM

- De ATmega32 heeft 1024 bytes EEPROM.
  - 16 bits adres, ATmega32 gebruikt onderste 10 bits.
- Inhoud blijft bewaard na spanningverlies.
- Data-opslag:
  - Constanten.
  - Gebruikersparameters voor applicatie.
- Manipulatie via I/O registers.

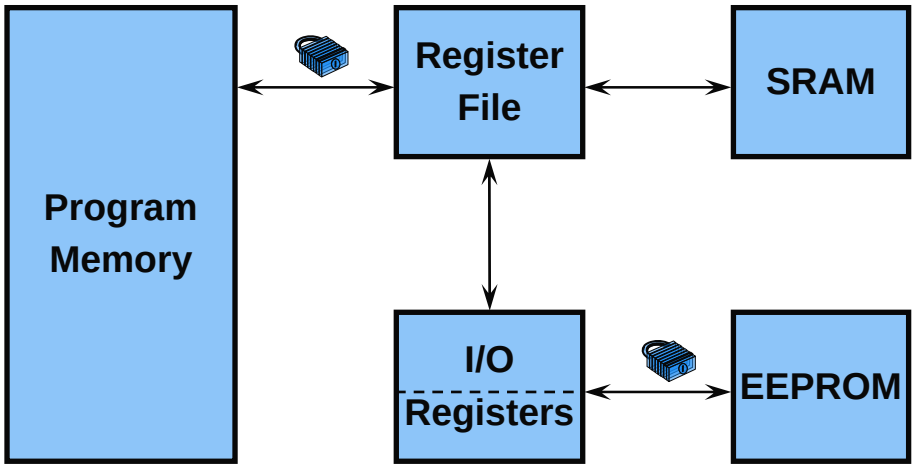
Address- en memory map

- De AVR heeft drie geheugens: Flash, eeprom en data.
- Registers en I/O zijn gemapped in de data-ruimte!



Geheugenoverzicht en verbindingen

- General purpose registers regelen alle dataverplaatsingen
- Direct Program memory access
- Direct SRAM access
- EEPROM access via IO-registers
  - EEAR,EEDR,EECR
- Secure Program memory en EEPROM access
  - Timed sequence



Programmeren

- Voor het programmeren van de AVR-microcontroller zijn drie mogelijkheden:

- Programmeren van de juiste bitpatronen.
  - Deze bitpatronen worden machinecode genoemd. Doen we niet.

- Programmeren in assembly.
  - Dit gaan we doen.

- Programmeren in een hogere programmeertaal zoals C.
  - Dit wordt uitgewerkt in MICPRG.

Machinecode

- Het gebruik van machinecode bij het opstellen van een programma voor een microcontroller is niet handig.
- Het is lastig alle codes uit het hoofd te kennen en het is foutgevoelig.
- Daarnaast kost het veel tijd om de codes samen te stellen.
- Voorbeeld:

0000.1100.0101.0001	0c51	tel R1 bij R5 op en plaats in R5

Assembly

- Het is beter om een eenvoudige programmmeertaal op te stellen die de programmeur verlost van het samenstellen van de machinecode.
- Deze taal moet eenvoudig te leren zijn en goed de instructies uitdrukken in voor de mens te begrijpen woorden.
- Zo'n programmeertaal wordt *assembly* genoemd.
- Het vertalen van de assembly-code naar machinecode wordt gedaan door een *assembler* (to assemble = samenstellen).

Opbouw machinecode en assemblycode

- Een instructie in machinecode bestaat uit een *opcode* en *operands*.
- Opcode staat voor Operation Code en geeft aan wat er gaat gebeuren.
- Voorbeeld: 0000.1111.0000.0001 000011 → add
- In assembler wordt deze opcode aangegeven door een zgn. Mnemonic*
- Voorbeeld: ADD R16, R17

* = Een mnemonic kan meerdere opcodes vertegenwoordigen, dit is dus niet helemaal waar.

Operands

- Een instructie moet vaak met data werken. Het aangeven/aanduiden van deze data noemt men een operand

- Voorbeeld:

```
jmp     0x03f    – Spring naar adres 0x03f (hexadecimal)
ldi     r16,23   – load register R16 with 23 decimal
add     r5,r1    – add contents of R1 to R5 and put in R5
```

Operands

- Er zijn ook instructies die geen operands hebben:

```
nop     - no operation
ret     - return from subroutine
wdr     - reset watchdog timer
```

- Er zijn geen instructies die drie of meer operands hebben.*

- *) behalve directives.

Labels

- jmp betekent dat de microcontroller door deze instructie een sprong maakt naar een nieuw adres. Dat adres moet natuurlijk opgegeven worden:

```
    jmp    0x03f
```

- Het gebruik van adressen in de vorm van een getal is niet handig. Als er nieuwe instructies worden ingevoegd, of de instructies worden verplaatst, verandert ook het sprongadres (doeladres).

- Een handig hulpmiddel zijn *labels* (etiket).
- De assembler berekent tijdens assembleren het doeladres.

- Voorbeeld:

```
        jmp    loop
        ...
loop:   ldi    r16,23
```

Directives

- De assembler kan meer dan het vertalen van assemblercode naar machinecode.
- Sommige mnemonics/instructies genereren bitpatronen, andere dienen voor overzichtelijke programma's.
- Deze mnemonics worden *directives* genoemd.
- Directives kunnen operands hebben.

Directives

- .org zet het startpunt voor te genereren code

```
        .org   0x020
start:  ldi    r16,23
```

- De ldi-instructie wordt gecodeerd vanaf adres 0x020.
- Label start krijgt de waarde 0x020.

Directives

- .equ ken een waarde toe aan een label

```
.equ   max_count   = 23
.equ   buffer_size = max_count+1
.org   0x020
start: ldi   r16,max_count
```

- De label max_count krijgt de waarde 23.
- De label buffer_size krijgt de waarde 24.

Directives

- .def ken een naam aan een register toe (aliasing)

```
.def   loop_count  = r16
.def   max_loops   = r17
...
       ldi    loop_count,23
again: ...
       cp     loop_count,max_loops  ;compares r16 & r17
       brne   again
```

- De naam loop_count is een alias voor register 16

Directives

- .byte reserveert bytes in SRAM-geheugen.
- .dseg geeft start van *datasegment* aan.

```
.dseg
.org   0x0060
array: .byte  10      ; loopt van 0x0060 t/m 0x0069
i:     .byte  1       ; 0x006a
```

- Datasegment begint op 0x0060 en reserveert een array van tien bytes. De label array krijgt de waarde 0x0060, i wordt 0x006a.

- Met .byte kan het gebruik van variabelen worden nagebootst.

Directives

- .db definieert byte-constanten in programmageheugen.
- .cseg geeft start van *codesegment* aan.

```
        .cseg
        .org   0x100
        table: .db    23,45,67,87
```

- Codesegment begint op 0x100 en definieert een tabel van vier bytes. De label table krijgt de waarde 0x100.

- Er zijn instructies om deze data over te hevelen naar registers en RAM.

Directives

- .dw definieert word-constanten in programmageheugen.
- .cseg geeft start van *codesegment* aan.

```
.cseg
.org   0x100
varlist:  .dw 0, 0xffff, 0b1001110001010101, -32768
```

- Codesegment begint op 0x100 en definieert een tabel van vier words (acht bytes). De label varlist krijgt de waarde 0x100.

Directives

- .include voegt een file in op een bepaald punt.

```
.include "m32def.inc"

.org   0x020
ldi    r23,0xff
out    DDRA,r23
```

- Leest de file m32def.inc in. DDRA is gedefinieerd in deze file.

- Noot: nooit file includen in zichzelf!

Assembler format

- Hieronder het format van de assembly:

  [; comment]

  [label:]

  [label:] mnemonic [operand[,operand]] [; comment]

  [label:] directive [operand[,operand[,…]]] [; comment]

- [] = optioneel.

Typen instructies

- We onderscheiden de volgende typen instructies:

- Data transfer:
  - laden van registers uit het geheugen
  - opslaan van registers in het geheugen
  - stack manipulatie

- Rekenkundige bewerkingen:
  - optellen, aftrekken,
  - verhogen/verlagen met 1,
  - vermenigvuldigen, delen, schuiven.

Typen instructies

- Logische bewerkingen:
  - AND, OR, EXOR, NOT
  - testen van bits
  - bitmanipulatie
- Spronginstructies
  - absolute en relatieve sprongen
  - voorwaardelijke en onvoorwaardelijke sprongen
- Subroutine aanroep en terugkeer van subroutine
- Speciale instructies:
  - NOP, WDR, SLEEP, BREAK

Belangrijke gegevens voor instructie

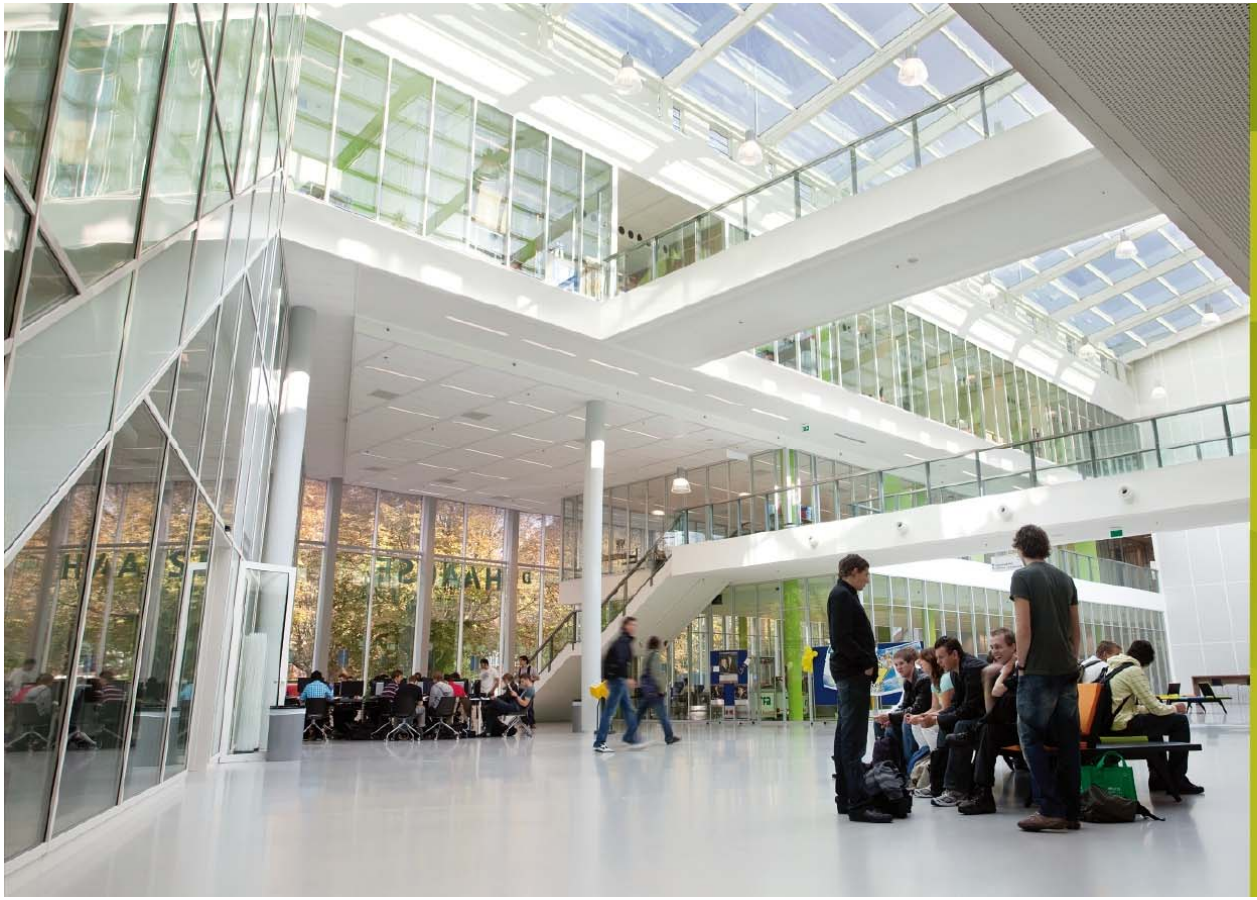
- Hoe worden de status flags beinvloed?
- Hoe vindt de CPU de data waarop de instructie moet werken (adresseermethode)?
- Hoeveel geheugenadressen beslaat de instructie?
- Hoeveel klokcycli neemt de instructie minimaal in beslag?
- Kijk voor deze zaken in de datasheet! (Atmel AVR instructieset die op blackboard staat)

Voorbeeldprogramma

```
.include "m32def.inc"

.def    temp        = r16

.equ    AllInputs   = 0x00
.equ    AllOutputs  = 0xff

.org 0x000

        ldi     temp,AllInputs          ; Port A is input
        out     DDRA,temp               ; uses r16 for data movement
        ldi     temp,AllOutputs         ; Port B is output
        out     DDRB,temp

loop:   in      temp,PINA               ; read Port A (switches)
        out     PORTB,temp              ; output to Port B (leds)
        rjmp    loop                    ; and again
```

Literatuur/leeswerk

- H2S1, H2S2, H3S1, H3S2, H3S4, H3S8
- Amerikaanse editie: H1S1, H1S2, H2S1, H2S2, H2S4, H2S8



Academie voor Technology, Innovation & Society Delft

Academie voor ICT & Media

De Haagse Hogeschool, Delft
015-2606311
J.E.J.opdenBrouw@hhs.nl
www.dehaagsehogeschool.nl

DE HAAGSE HOGESCHOOL