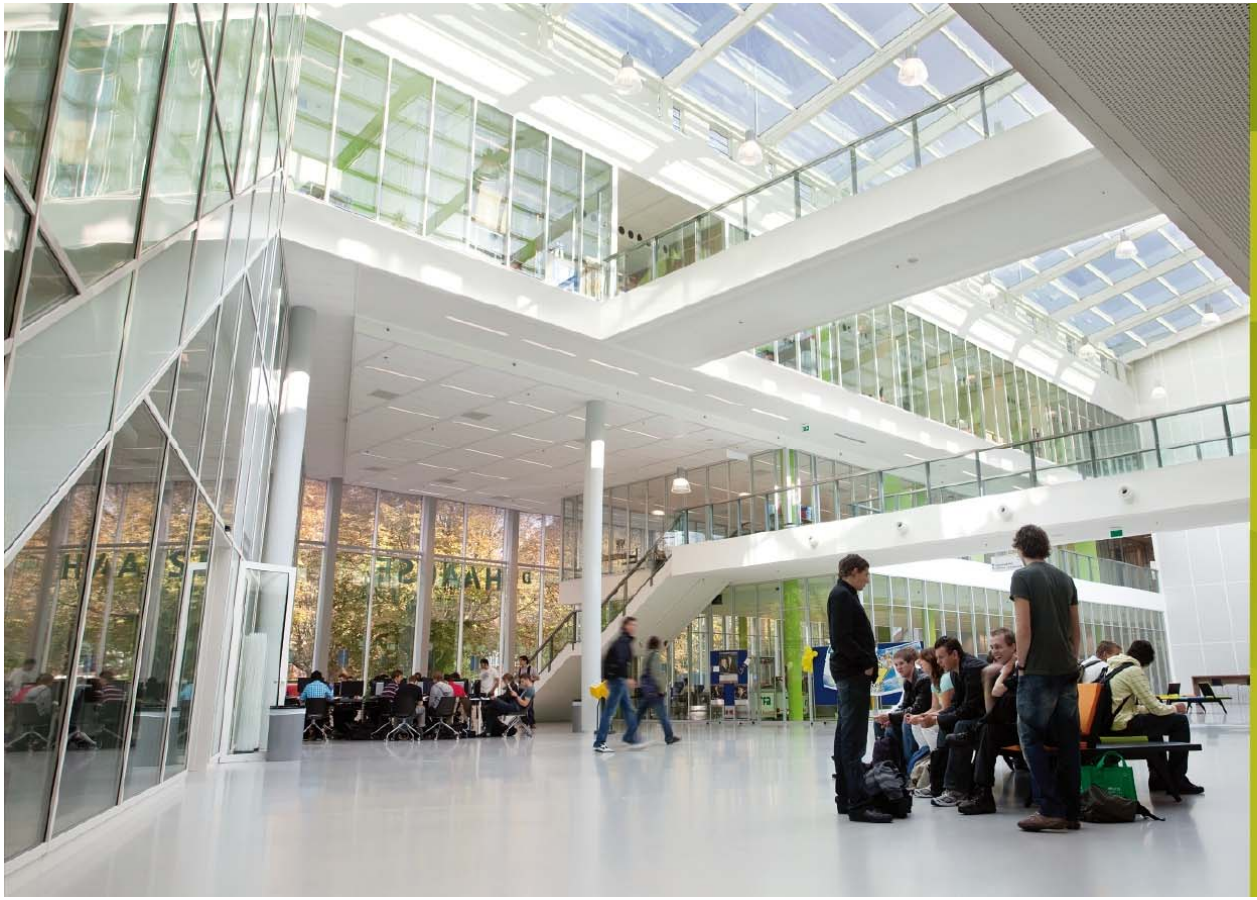


Academie voor Technology, Innovation & Society Delft
Academie voor ICT & Media

Microcontrollers

Week 4 – Introductie microcontroller

Jesse op den Brouw (met dank aan Ben Kuiper)

INLMIC/2018-2019

DE HAAGSE HOGESCHOOL

Week 3

- Lussen
- Ontdenderen
- Stack
- Parameteroverdracht

Wachtlussen

- In een programma willen we het volgende realiseren:
  - Doe iets
  - Wacht halve seconde
  - Doe iets
- Wachten kan gerealiseerd worden door zinloze instructies (nop) uit te voeren

- Probleem: hoeveel nop-s moet je uitvoeren om een halve seconde te wachten?
- 1 nop-instructie kost 1 klokpuls
- Aantal klokpulsen wachten: fcpu * wachttijd
- fcpu = 3686400, wachttijd = 0,5 seconden → 1843200 klokpulsen!

- En dus 1843200 nop-instructies
  → past niet in Flash ROM!

DE HAAGSE HOGESCHOOL

Wachtlussen

- Eenvoudig beginnen: wacht 100 klokpulsen.
  - Nu 100x nop.

- Handiger is om 100 keer één nop te tellen → lus.

```
        ldi   r16,100       ; 100x *)
loop:   subi  r16,1         ; verlaag R16 met 1
        nop                 ; de nop
        brne  loop          ; spring als Z=0
```

- Z-vlag wordt 1 als R16 van 0x01 naar 0x00 gaat.

- Noot: natuurlijk kost dit totaal meer dan 100 klokpulsen.

DE HAAGSE HOGESCHOOL

Wachtlussen

- Maar `subi` en `brne` kosten ook klokpulsen.
  - `subi` kost 1 klokpuls, `brne` kost 2 klokpulsen (indien wordt gesprongen)

- Nu is `nop` niet meer nodig, `subi` en `brne` kosten 3 pulsen per lus → 33 lussen, `ldi` kost 1 klokpuls.

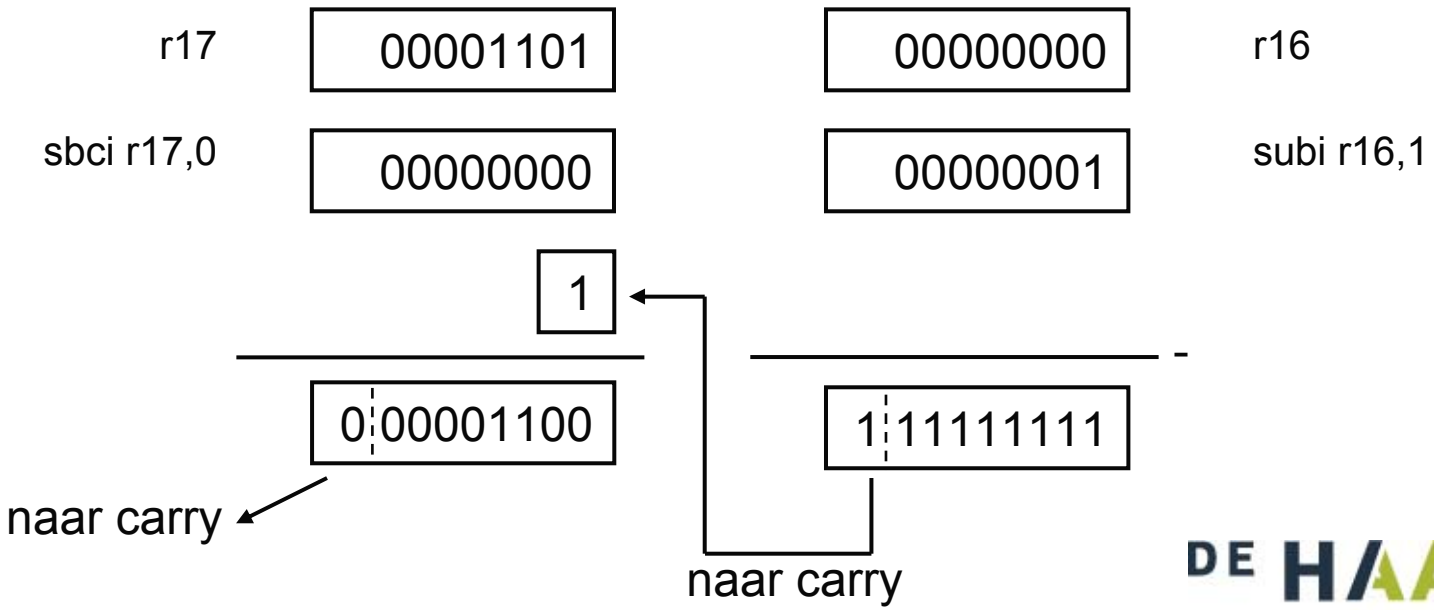
- Z-vlag wordt 1 als r16 van 0x01 naar 0x00 gaat.

```
        ldi   r16,33        ; 33x
loop:   subi  r16,1         ; verlaag R16 met 1
        brne  loop          ; spring als Z=0 (R16 ≠ 0)
        nop                 ; precies 100 kp
```

Wachtlussen

- Stel we willen meer dan 255 lussen wachten. Dit kan niet met één register.

- Oplossing: twee registers gebruiken, extra subtract-instructie.



Wachtlussen

- Lus wordt nu één instructie langer en kost één extra klokpuls.
- Om 3000 lussen te wachten moet dit geschreven worden in hexadecimale notatie: 0x0bb8.
- In R17 wordt 0x0b geplaatst, 0xb8 wordt in R16 geplaatst.

```
        ldi   r16,0xb8  ; 0xb8  (3000 = 0x0bb8)
        ldi   r17,0x0b  ; 0x0b
loop:   subi  r16,1     ; verlaag R16 met 1
        sbci  r17,0     ; verlaag R17 met 1 indien C
        brne  loop      ; spring als Z=0 (R17:R16 ≠ 0)
```

Wachtlussen

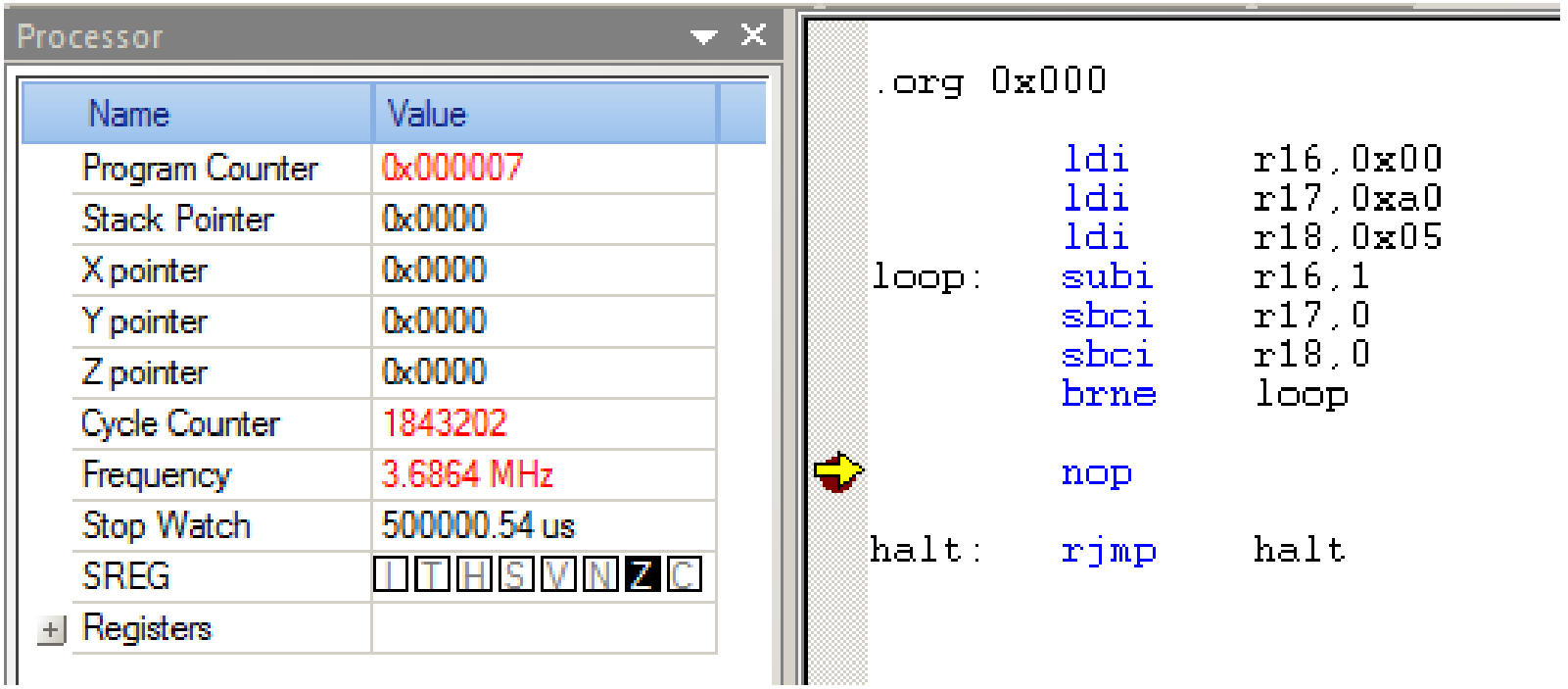
- We moeten 1843200 klokpulsen tellen om een halve seconde te wachten. Dit moet met drie registers, waardoor elke lus 5 klokpulsen kost, dus 368640 lussen.

```
        ldi   r16,0x00      ; 0x05a000 = 368640
        ldi   r17,0xa0      ;
        ldi   r18,0x05      ;
loop:   subi  r16,1         ; verlaag R16 met 1
        sbci  r17,0         ; verlaag R17 met 1 als C
        sbci  r18,0         ; verlaag R18 met 1 als C
        brne  loop          ; spring als Z=0
```

DE HAAGSE HOGESCHOOL

Wachtlussen

- Hieronder een simulatie van deze wachtlus.



Wachtlussen

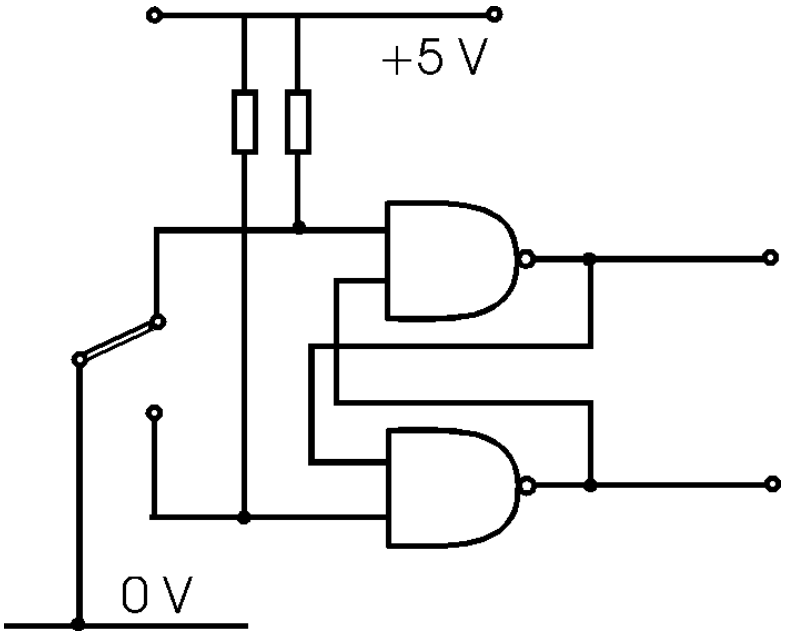
- Algemene formule: $$aantal\_lussen = \frac{fcpu \cdot wachttijd}{klokpulsen\_per\_lus}$$

- Een register: 3 klokpulsen per lus, 768 klokpulsen
- Twee registers: 4 klokpulsen per lus, 262144 klokpulsen
- Drie registers: 5 klokpulsen per lus, 83886080 klokpulsen
- Vier registers: 6 klokpulsen per lus, 25769803776 klokpulsen

- Opmerking: vier register wachtlus op 3,6864 MHz levert een maximale wachttijd van 1,94 uur.

Schakelaar ontdenderen

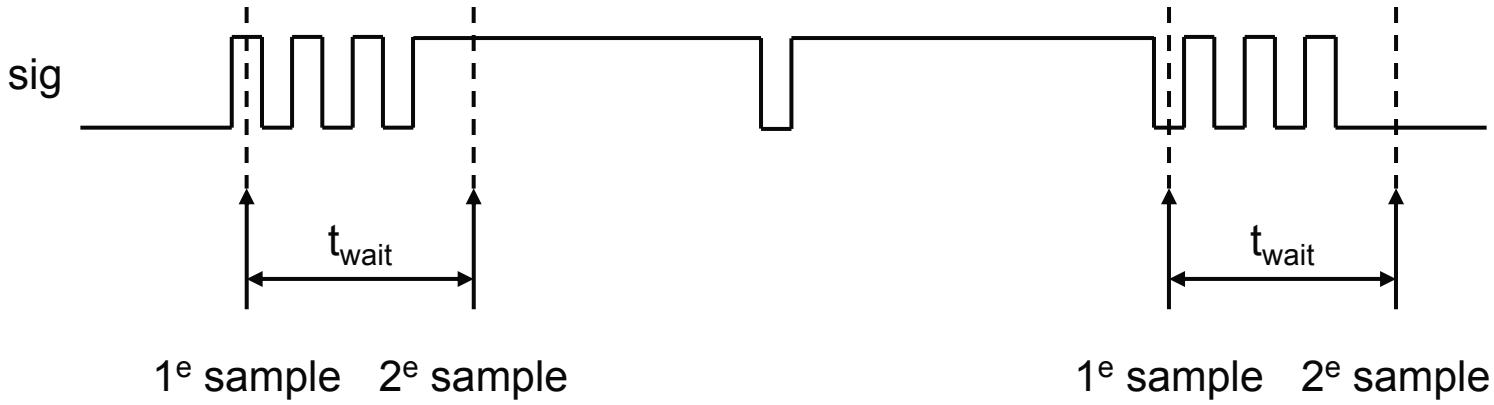
- Mechanische schakelaars en drukknoppen hebben last van “denderen”.
- Na het omklappen van de metalen geleider, stuitert (dendert) deze heen en weer tussen de twee kontaktpunten.
- Ontdenderen kan met een eenvoudige digitale schakeling.



Schakelaar ontdenderen & ontstoren

- In een digitale component zoals een FPGA of een microcontroller is deze schakeling niet aanwezig.

- Een alternatief is om een schakelaar twee keer in te lezen met een korte pauze ertussen.



Schakelaar ontdenderen & ontstoren

- In pseudocode:

  lees schakelaar (wacht op schakelaar ingedrukt)
  wacht korte tijd
  lees schakelaar
  waardes verschillend? dan opnieuw

- Wachtijd van 10 ms tot 50 ms.

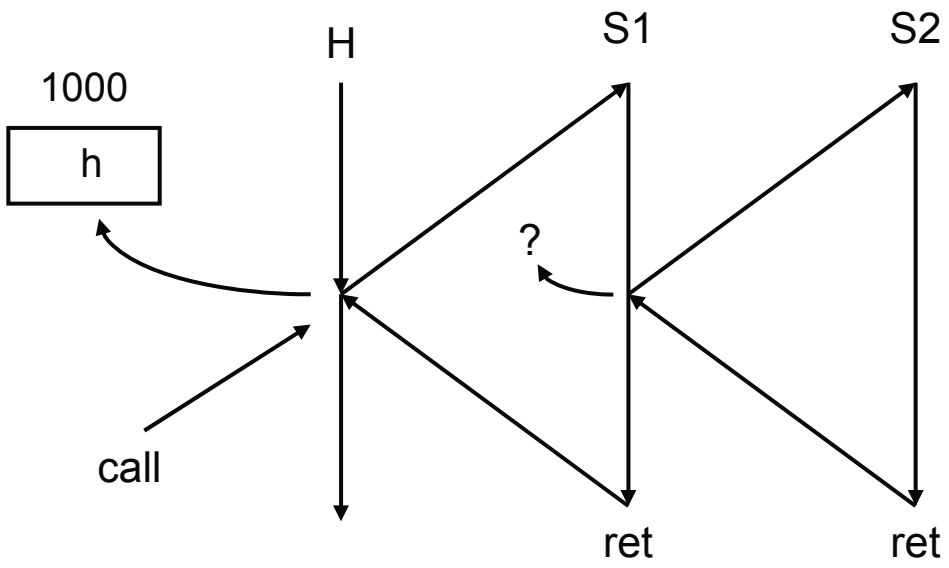
Stack

- Veel gebruikte code, zoals het afdrukken van tekens of getallen, worden in zogenaamde *subroutines* ondergebracht.
- Deze kunnen worden aangeroepen door het hoofdprogramma.
- Natuurlijk moet na afloop van de subroutine het hoofdprogramma verder gaan waar het gebleven was.
- Er moet dus een terugkeeradres (*return address*) bekend zijn.

Stack

- Dit terugkeeradres kan in een geheugenplaats worden opgeslagen, bijvoorbeeld adres 1000.

- Maar wat nu als een subroutine zelf weer een subroutine aanroept (zie figuur)?



H = hoofdprogramma

S1 = subroutine 1

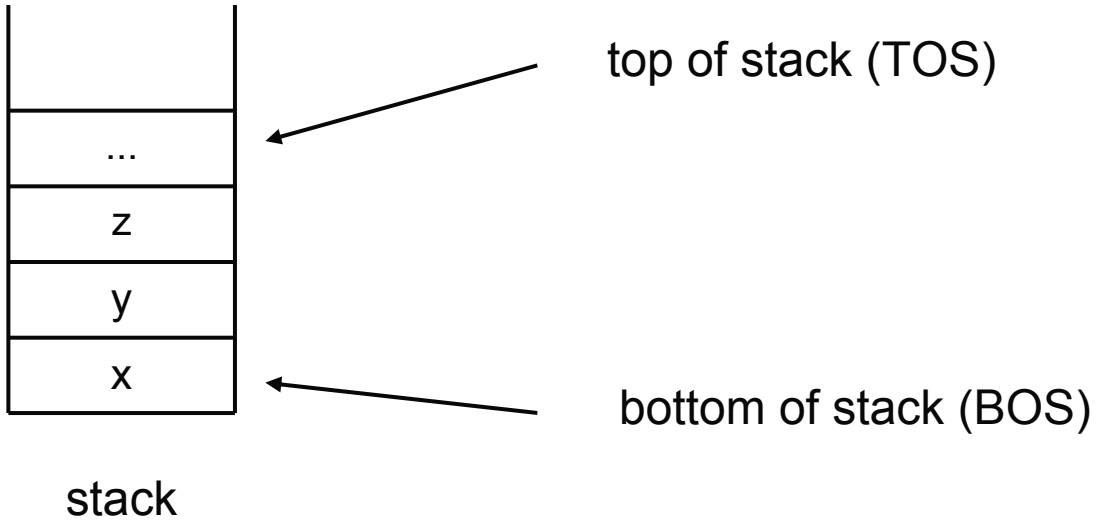
S2 = subroutine 2

call = aanroep subroutine

ret = return

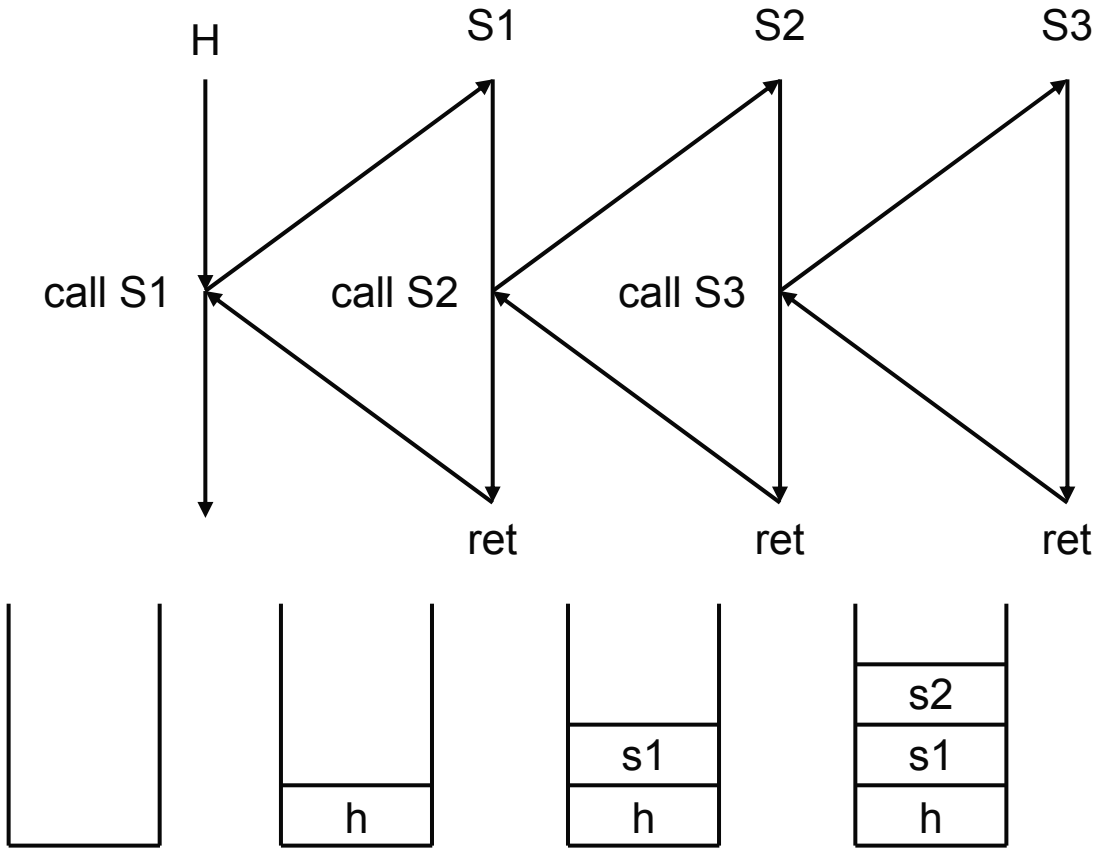
Stack

- Beter is om een *stack* (stapel) te maken waar de terugkeer-adressen opgestapeld worden.



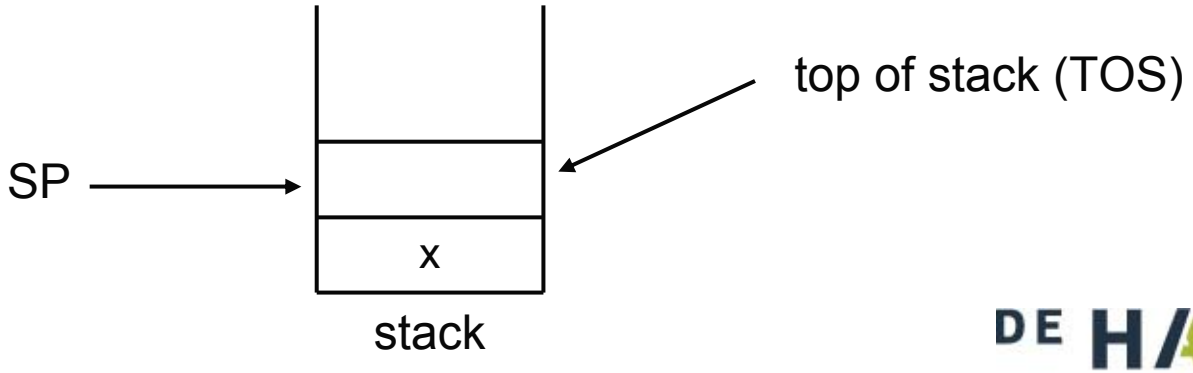
Stack

- Het adres boven op de stapel is dan het terugkeeradres dat het eerste aan de beurt is.



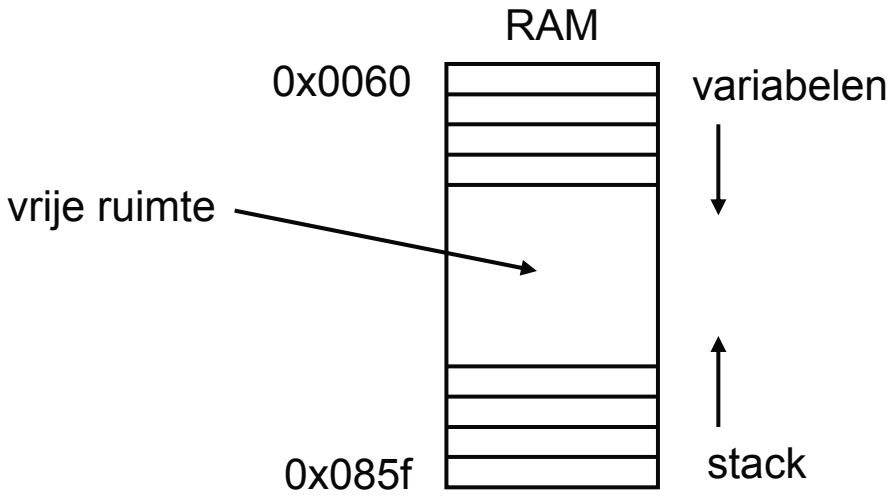
Implementatie stack

- Een stack is te implementeren met behulp van een stukje RAM en een wijzer (*stack pointer*).
- Deze stack pointer, afgekort tot SP wijst altijd naar de bovenkant van de stack (*top of stack, TOS*).
- Dit is de eerstvolgende vrije plaats.



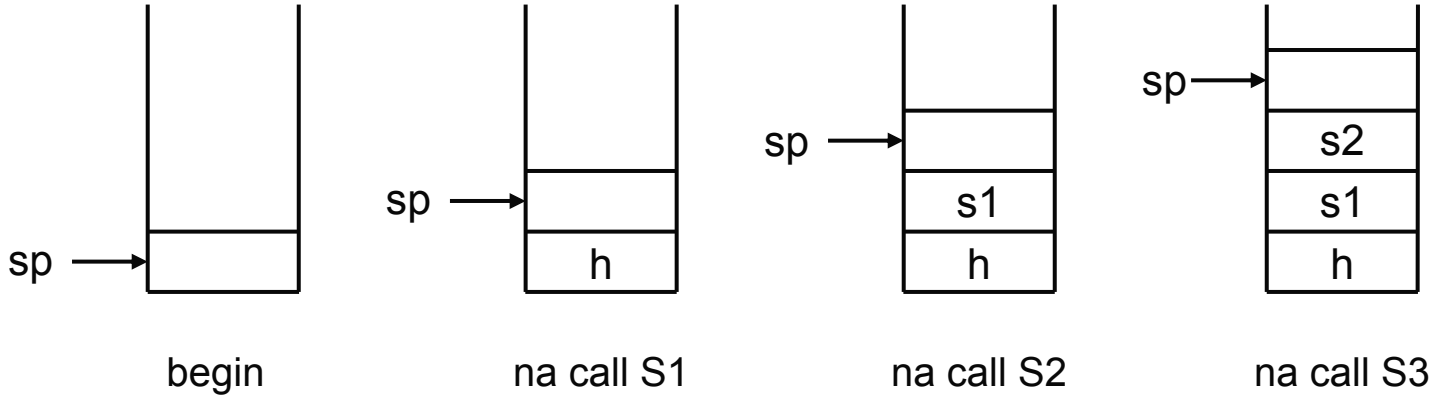
Implementatie

- Verder is gekozen om de stack te laten groeien van hoge adressen naar lage adressen (van hoog naar laag).
- De reden is dat variabelen van laag naar hoog groeien.
- Het gebruik van RAM wordt dan optimaal benut, en de kans op botsing is dan het kleinst.



Implementatie

- De keuze om de stack pointer naar de eerstvolgende vrije plaats te laten wijzen heeft tot gevolg dat *na* het plaatsen van het terugkeeradres, de stackpointer moet worden *verlaagd*.
- Dit wordt een *post decrement stack* genoemd.
- Hieruit volgt dat bij het ophalen van het terugkeeradres de stack pointer *eerst* moet worden verhoogd.

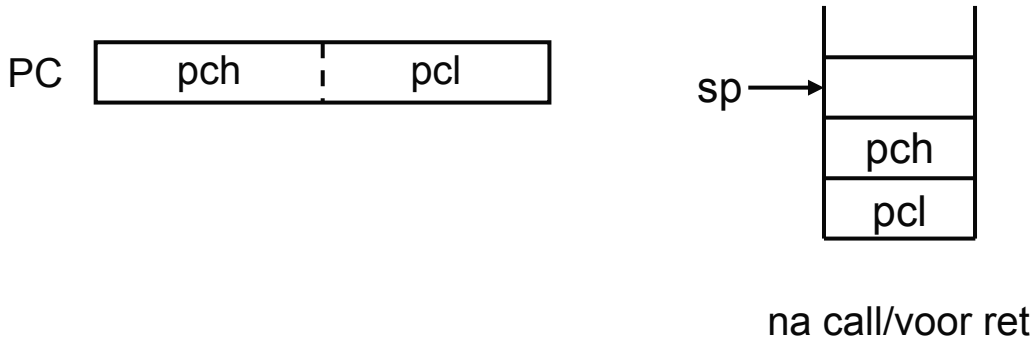


Terugkeeradres

- Het terugkeeradres is het adres direct na de aanroepinstructie `CALL` (na het ophalen van de `CALL`-instructie staat dit adres in de PC).
- Bij de ATmega32 is dit een 2-byte adres zodat in totaal 64 k words kan worden geadresseerd.
- De stack pointer wordt dus met twee verlaagd.
- Bij AVR's met meer Flash-ROM worden drie bytes op de stack gezet.
- De `call`-instructie past de vlaggen niet aan.

Terugkeeradres op stack

- Omdat het terugkeeradres twee bytes beslaat, moet de controller dit in twee slagen op de stack zetten.
- Eerst wordt de laagste 8 bits op de stack gezet (LSB).
- Daarna worden de hoogste 8 bits op de stack gezet (MSB).



na call/voor ret

Terugkeren

- Er wordt teruggekeerd naar het aanroepende programma (*caller*) na een `RET`-instructie (return from subroutine).
- Er worden twee of drie bytes van de stack gehaald (afhankelijk van het type) en geplaatst in de Program Counter.
- `RET` past de vlaggen niet aan.

Push en pop

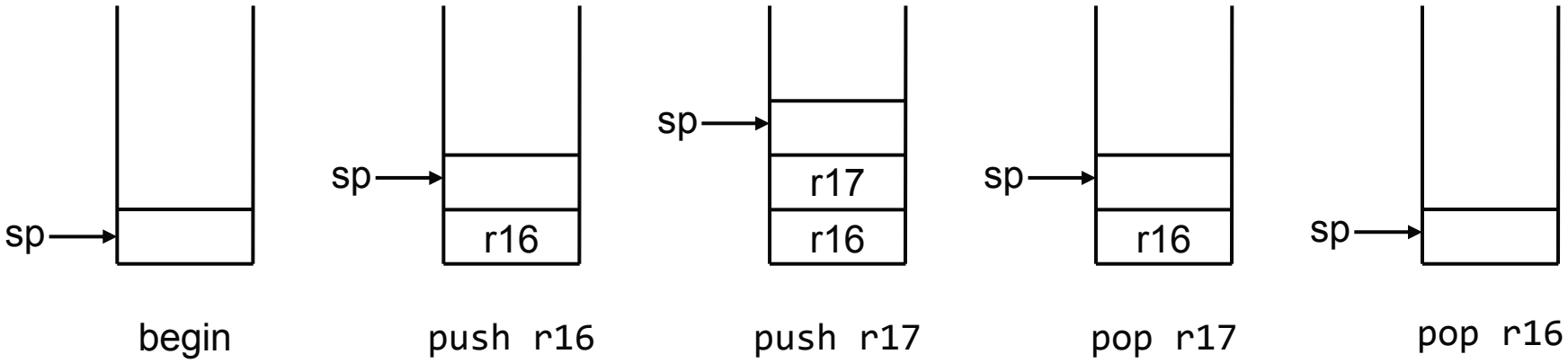
- Tijdens het uitvoeren van de subroutine-instructies worden registers gebruikt. De inhoud kan dan gewijzigd worden. Als deze registers ook in het hoofdprogramma gebruikt worden, is het maar de vraag of na terugkeer deze gegevens nog intact zijn.

- Met de instructie `PUSH` wordt de inhoud van een register op de stack gezet, en de stack pointer met één verlaagd.

- Met de instructie `POP` wordt de stack pointer met één verhoogd en een register geladen met de inhoud van de *top of stack*.

Push en pop

- Let op de volgorde van push en pop.



Push en pop

- Je kan hier leuke dingen mee doen:

```
; alternatief voor mov
push   r16
pop    r17

; verwisselen van twee registers
push   r16
push   r17
pop    r16
pop    r17
```

Voorbeeld wachtlus

```
        ...
        call   delay          ; aanroep wachtlus
        ...
delay:  push   r24            ; opslaan registers
        push   r25
        ldi    r24,0xb8       ; laden van aantal lussen
        ldi    r25,0x0b       ; 3000x
loop:   subi   r24,1          ; de wachtlus
        sbci   r25,0
        brne   loop
        pop    r25            ; ophalen registers
        pop    r24
        ret                   ; terugkeren
```

Initialisatie stack pointer

- Voordat de stack kan worden gebruikt, moet de stack pointer geladen worden met het adres van het hoogste vrije RAM-adres. Bij de ATmega32 is dat 0x085f.

- De stack pointer zit verstopt in de I/O-registers en moet met in en out-instructies benaderd worden. De stack pointer is 16-bits, de I/O registers zijn 8-bits. De stack pointer bestaat dus uit twee registers.

```
ldi     r16,0x5f        ; low byte adres
out     0x3d,r16        ; in low byte SP
ldi     r16,0x08        ; high byte adres
out     0x3e,r16        ; in high byte SP
```

Initialisatie stack pointer

- Voor elke ATmega is het hoogste RAM-adres of de plaats van de stack pointer verschillend.

- In de .include file zijn hiervoor equates en defines gemaakt.

```
.include     “m32def.inc”

ldi    r16,low(RAMEND)     ; low = low byte of ...
out    SPL,r16             ; SPL = SP low
ldi    r16,high(RAMEND)    ; high = high byte of ...
out    SPH,r16             ; SPH = SP high
```

Stack under/overflow

- Als er meer data op de stack wordt geplaatst (*pushen, gepushd*), heb je een *stack overflow*.
- Als er teveel data van de stack wordt gehaald (*poppen, gepopd*), heb je een *stack underflow*.
- Er is geen hardwarevoorziening die dan ingrijpt. De programmeur moet dit zelf in de gaten houden.

Parameteroverdracht

- Een functie/routine is een stukje code dat door het hoofdprogramma kan worden aangeroepen.
- Voorbeeld: `delay` uit vorige slides.
- `delay` heeft echter een vaste wachttijd. Handiger is het om de wachttijd variabel te maken.
- Het hoofdprogramma moet deze tijd op één of andere manier aan de functie overdragen.

Parameteroverdracht

- De functie wordt nu voorzien van parameters.
- Er zijn twee manieren van overdracht mogelijk.
- Via registers.
- Via stack.
- Alleen via registers wordt besproken.

Parameteroverdracht

- Registers kunnen worden gebruikt voor het overdragen van parameters.
- Bij de wachtlus wordt de ‘tijd’ door middel van twee registers overgedragen. Het is dus een 2-byte getal.
- Hiervoor worden R24 en R25 gebruikt. Dit moet goed gedocumenteerd worden!
- R24 bevat het lage byte, R25 bevat de hoge byte.

Parameteroverdracht

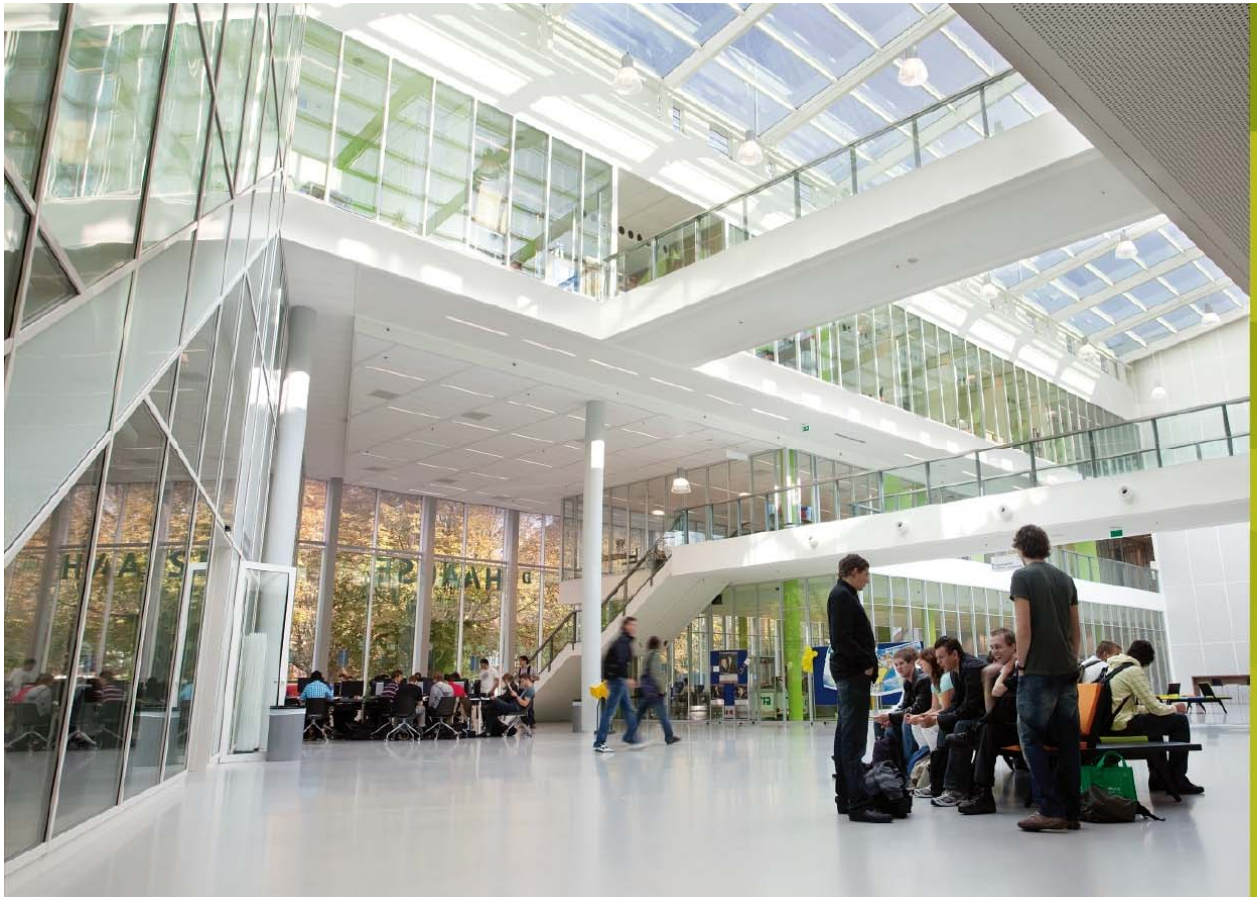
```
        ...
        ldi    r24,0xb8      ; laden van aantal lussen
        ldi    r25,0x0b      ; 3000x
        call   delay         ; aanroep wachtlus
        ...
delay:
loop:   subi   r24,1         ; de wachtlus
        sbci   r25,0
        brne   loop
        ret                  ; terugkeren
```

NB: na terugkeer zijn de inhouden van R24 en R25 veranderd!

DE HAAGSE HOGESCHOOL

Literatuur/leeswerk

- H4S2, H5S1, H5S2
- sheets



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