A&C week 1

Remi Reuvekamp - Timo Strating - ITV2E

2 oktober 2018, Groningen

Inhoudsopgave

1. *Opdracht 1*

1.1 A

```
SET Rd    ; Set bits in Register
CBR Rd, K  ; Clear Bit Register
```

1.2 B

```
NEG Rd    ; Vervangt de waarde in Rd met de twos complements versie ervan.
```

1.3 C

```
BR__    ; Alle Branch commando's beginnen met BR
        ; Bij conditionele sprongen staat er bij de Operation if...
        ;  in pseudo C-achtige code en de assembly begint altijd met BR.
```

1.4 D

```
IN p     ; Deze instructies dienen voor het instellen van
OUT p    ;   een poort als input of output
```

1.5 E

Hoelang duurt de CALL instructie(bij de ATmega32) ? Waarom duurt deze instructie zo lang?

We kunnen tegelijkertijd fetchen, decoden en uitvoeren. Deze extra stappen betekenen alleen ook dat we extra moeten wachten in het geval dat het leeg moet. Het duurt daarom 4 cycles.

1.6 F

```
RJMP        ; is relatief vanaf het huidige adres waardoor het lukt binnen
            ;  16 bits. Als nadeel heeft dit alleen dat je niet door het
            ;  volledige geheugen kan springen
JMP         ; is relatief vanaf het begin van het geheugen hiervoor is wel
```

```
;  een volledig 16 bits adres nodig waardoor het commando 32 bits
;  nodig heeft.
```

1.7 G

```
ADC      ; Add is met carry flag
ADD      ; Add
```

1.8 H

Stel dat je 2 registers bij elkaar op wil tellen die allemaal 1-nen bevatten, is dan n carry bit wel voldoende?

ja met 2 getallen die binair alleen maar bestaat uit 1-nen kan ja als je ze plus elkaar doet niet meer getallen nodig hebben dan je kwijt kan met 1 extra carry

2. *Opdracht 2*

2.1 Assembly

0000 1100 01010101. Wat betekend deze opt code? Kun je uitleggen waarom er in dit geval twee antwoorden mogelijk zijn ?

```
LSL R..          ; 0000 11dd dddd dddd
ADD R.. R..      ; 0000 11rd dddd rrrr
; Logical Shift left is een add met zich zelf
```

De optcode voor een Logical shift left en Add zijn uiteindelijk het zelfde omdat ze exact het zelfde doen.

3. *Opdracht 3*

3.1 A

Waar staat dit bestand in je file systeem ? In de huidige working directory

3.2 B

Met welke adressen corresponderen de symbolen PORTB, DDRB en PINB ?(Zoek in het tekstbestand m328Pdef.inc)

```
PORTB: 0x05
DDRB: 0x04
PINB: 0x03
```

3.3 C

Waarvoor staan de symbolen XH en XL ?

XH: de eerste register van het X (X bestaat uit 2 registers). Hierin staan de meest significante bits.

XL: het tweede register van X. Hierin staan de minst significante bits.

3.4 D

Wat is het startadres voor het RAM ? Wat is de grootte van het RAM ? Wat is het hoogste RAM adres ?(Data Memory Declarations)
Startadres: 0x0100

Grootte: 2048

4. *Opdracht 4*

Geef aan wat het binaire resultaat is na de volgende stukjes code.

4.1 A

```
ldi r16,~0xf0    ; r16 = 0b 00001111
```

4.2 B

```
.equ a = 5           ; a = 0b 0000 0101
.equ b = 0xff16      ; b = 0b 1111 1111 0001 0110
ldi r18,low(a|b)     ; r18 = 0001 0111
```

4.3 C

```
.equ a = 5           ; a = 0b 0000 0101
.equ b = 0xff        ; b = 0b 1111 1111
ldi r18, a^b         ; r18 = 0b 1111 1010
```

4.4 D

```
ldi r16, (1<<5)|(1<<7)  ; r16 = 1010 0000
out PORTB, r16          ; PORTB = r16
                        ; PORTB wordt: 1010 0000
```

4.5 E

```
ldi r16, ~(1<<PB3)      ; r16 = 1111 0111
```

4.6 F

```
ldi r16,low(RAMEND)     ; r16 = low( 0x08ff ) = 1111 1111
out SPL,r16             ; SPL   0x3d = r16
ldi r16,high(RAMEND)    ; r16 = high( 0x08ff ) = 0000 1000
out SPH,r16             ; SPH   0x3e = r16
```

```
RAMEND: 0x08ff: 2303

0000 1000 1111 1111
Naar SPL wordt 0000 1000 geschreven
Naar SPH wordt 1111 1111 geschreven
```

5. *Opdracht 5*

```
ldi r16, 0xAA           ; r16 = 0b 10101010
ldi r17, 0x12           ; r17 = 0b 00010010
```

5.1 A

Wat is na elke instructie de waarden van 16 en r17 ? NB. voor elke instructie worden r16 en r17 weer op bovenstaande waarden gezet.

```
neg r16         ; Change to two's complement   r16 = 0b 01010110
swap r17        ; Swap Nibbels  r17 = 0b 0010 0001 = 0x21
sbr r16, 3      ; Set Bits in Register r16 = 0b 10101011
dec r17         ; Decrement  r17 -= 1  r17 = 0b 00010001
ori r16, 0xF0   ; Logical Or with immediate  wat betekent
                ;  OR with constante in het register r16
```

5.2 B

Wat is de waarden van r0 en r1 na : mul r16, r17?

```
mul r16, r17    ; Multiply heeft mogelijk meerdere carry's nodig dus
                ;  daarom komt het resultaat in registers r0 en r1
```

6. *Opdracht 9*

6.1 A

Leg uit wat de instructie 'lpm' doet.

LPM R.., Z; Laad Program Memory. ¡¡¡ TODO ¿¿¿

6.2 B

```
.org 0  ; Dit vertelt de assembler dat hier adres 0 begint
  ldi zh, high(ASCII_TABLE << 1) ;define Z-pointer ; REGISTER, Inmediate
  ; Laad het getal 0x00 in Register R31
  ldi zl, low(ASCII_TABLE << 1) ; REGISTER, Inmediate
  ; Laad het getal 0x28 in Register R30
  ldi r16, 0x00 ; REGISTER, Inmediate
  ; Laad het getal 0x0 in Register R16.
  ; Hiermee gaan we de input/output status van DDRC zetten.
  out DDRC, r16 ;PORTC input ; I/O, Direct
  ; Zet alle pins van PORTC als output
  ldi r16, 0xff ; REGISTER, Inmediate
  ; Laad het getal 0xff in Register R16. Zie 2 naar boven.
  out DDRD, r16 ;PORTD output ; I/O, Direct
  ; Zet alle pins van PORTD als output

BEGIN:
  in r16, PINC ; I/O, Direct
  ; Zet alle de waarde van de pins van PinC naar register R16
  andi r16, 0b00000111 ; mask upper 5 bits ; REGISTER, Inmediate
  ; Doe een AND op binair niveau met het getal 5. Wat als
  ; gevolg heeft dat 0111 als een masker werkt en alleen de
  ; laatste 3 eenen over neemt van R16
  add zl, r16  ; REGISTER, Direct
  ; Voeg R16 toe aan R30
  lpm r17, z ; SRAM, Indirect
  ; Het laad R31 in R17 als de waarde van R30 deelbaar is door
  ; 2 zonder rest anders laad het R30 in R17
  ; out PORTD, r17 ; I/O,  Direct
  ; Het resultaat wordt daarna naar de output geschreven
  rjmp BEGIN ; SRAM
  ; Spring weer terug naar het label BEGIN:


.org 20 ; SRAM
```

```
; Het zegt tegen de assembler dat wat hierna komt bij adress 20 begint

ASCII_TABLE: .DB '0','1','2','3','4','5','6','7' ; SRAM
; De ASCII tekens 0 1 2 3 4 5 6 7 worden in het code segment geladen
```

6.3 C

Zie de code.

Het leest een binair getal van PORTC, converteerd dat naar de waarde daarvan in ASCII, en output dat op PORTD.

7. Opdracht 10

7.1 A

```
.dseg
    .org SRAM_START

dest:
    ; Reserveer 20 bytes voor Hello world
    .byte 20

.cseg
    ; De eerste instructie is om naar het label start te springen
    rjmp start

src:
    ; De string die we gaan kopieren van het code segment naar het data segment
    .db "hello world !"
    ; De lengte van de sting "hello world !"
    .equ length=13
    ; Dit is het tijdelijke register waar de data komt te staan tijdens het kopieren
    .def temp = r0
    ; De teller die bij houd hoeveel tekens wij van de string al hebben gekopieerd
    .def counter = r17

start:
    ; Z-pointer wijst naar de source maar gaat x2 of to wel << 1
    ; omdat lpm naar de aller eerste bit kijkt om te behalen of
    ; hij het hoge of het lage gedeelte van de Z pointer
    ; moet hebben
  ldi zh,high(2*src)
  ldi zl,low(2*src)
  ; X-pointer wijst naar de desination
  ldi xh,high(dest)
  ldi xl,low(dest)
  clr counter

loop:
    ; Laad waar de Z pointer naar point
  lpm
  ; Verhoog de Z pointer met 1
  inc zl
  ; Sla de Z pointer op
  st x+,temp
  ; Verhoog De counter met 1 om aan het volgende char te beginnen
```

```
  inc counter
  ; Vergelijk of we al op het eind van de sting zijn
  cpi counter, length
  ; Zolang als er nog meer chars zijn die we kunnnen kopieren
  ; dan beginnen we weer opnieuw met het kopieren van een char
  brlt loop

end:
    ; infinite loop
  rjmp end
```

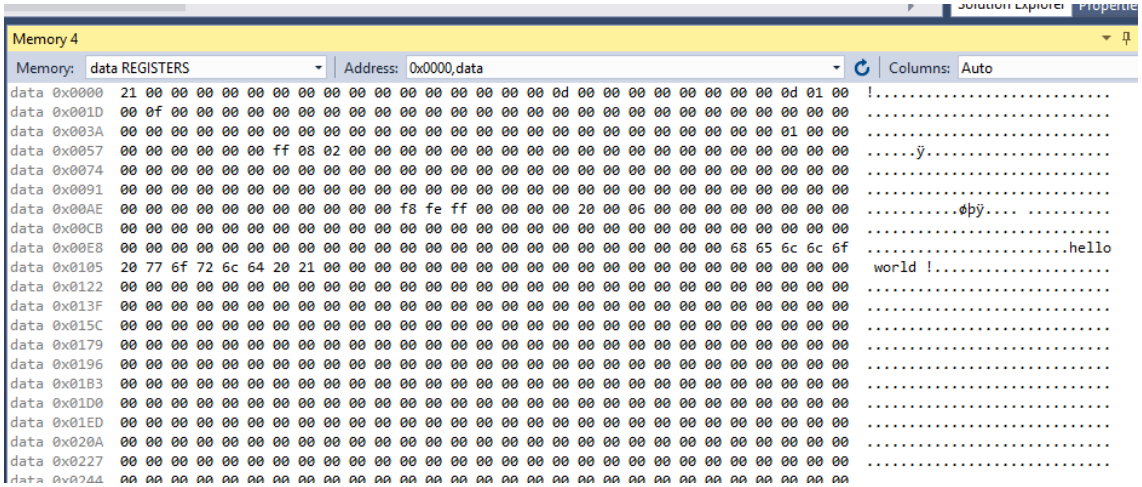
7.2 B

Het geen wat in het code segment staat wordt naar het datasegment overgezet. In dit geval wordt "Hello World" vanuit het data segment overgezet naar het code segment.

7.3 C

LMP pakt de Z pointer en plaatst het in R0 (of Rd, als die specified is). De Z pointer is alleen al 2 registers lang dus 16 bits wat LMP daarom doet is dat het kijkt naar de minst significante bit. Deze bit bepaald of het naar Zl (het rigister van Z waar de minst waardevolle bits in staan) of Zh (het andere register van Z) gaan kijken.

7.4 D



7.5 E

```
.dseg
.db "hello world !"
```

Bovenstaande regel zou niet werken omdat de datasegment tijdens het resetten leeggehaald zou worden. De assembler zou dus de string opslplitsen in de verschillende geheugencellen maar tijdens de reset wordt daarna alles weer overschreven