Assembly en C
Opdracht 1
2018-09-27

Timo Strating
Remi Reuvekamp

ITV2E

1.

A.
ser rX
clr rX

B.
Vervangt de waarde in Rd met de two’s complements versie ervan.
Klopt dit? Idk?

C.
Bij conditionele sprongen staat bij de Operation “if...” in pseudo C-achtige code.

D.
OUT: Schrijf de contents van het gegeven register naar de gegeven Input/Output ding (ports, timers, “configuration registers”, whatever dat is)
IN: Lees de contents van een Input/Output register en lees het in het gegeven register.

E.
CALL duurt 4 of 5 ticks. Want subroutine voortbereiden (storen waar de aanroep is gemaakt) kost tijd.

F.
RJMP is 16bits
JMP is 32bits (command, niet-relatief adres).

G.
ADC is met carry flag.
ADD is zonder carry flag.

H.
Ja?

2.
ADD
LSL
Logical Shift Left is een wrapper helper ding wat basically gewoon een ADD is met zichzelf.
Waarschijnlijk maakt de assembler er ook gewoon een ADD van.

3.

A.
In de huidige working directory?

B.
PORTB: 0x05
DDRB: 0x04
PINB: 0x03

C.
XH: de eerste register van het X (X bestaat uit 2 registers)
XL: het tweede register van X.

D.
Startadres: 0x0100
Grootte: 2048

4.

A.
```
ldi r16,~0xf0
```
0x0F

B.
```
.equ a = 5
.equ b = 0xff16
ldi r18, low(a|b)
```
0000 0000 0000 0101
1111 1111 0001 1010
1111 1111 0001 1111

```
        low: 0001 1111

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

D.
.include "m32def.inc"
ldi r16, (1<<5)|(1<<7)
out PORTB, r16
        PORTB wordt: 0, want de or van bitshift


E.
.include "m32def.inc"
ldi r16, ~(1<<PB3)
        PB3 is 0, dus 1<<0 is 1.
        ~0000 0001 = 1111 1110
        r16 wordt ingeladen met 254

F.
.include "m32def.inc"
ldi r16,low(RAMEND)
out SPL,r16
ldi r16,high(RAMEND)
out SPH,r16

RAMEND: 0x08ff: 2303

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

5.

r16: 0xAA (1010 1010)
r17: 0x12 (0001 0010)

NEG: Wat?
SWAP r17 (Swap nibbles):
        r17 wordt 0010 0001
SBR r16, 3:
        r16 wordt 1010 1011 want bit 0 en 1 worden op 1 gezet
DEC r17:
        r17 wordt 0001 0001 (want er wordt met 1 gedecrement)
ORI r16, 0xF0:
        1010 1010
        1111 0000
        r16 wordt:
        1111 1010

8.
A
.DSEG
a: .BYTE 2

B
.DSEG
c: .BYTE 1

C
.DSEG
c: .equ 9032

D
```

```
.CSEG
b: .db “COMP9032”
```

9

A.

Loads one byte pointed to by the Z-register into the destination register Rd.

B.

```
.org 0
; Het zegd tegen de assembler dat wat hierna komt bij adress 0 begind
        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, 0x0
; REGISTER                          ; Inmediate                          ; Laad het getal 0x0 in Register
R16. We moeten dit doen omdat we een register alleen maar kunnen gbruiken om een port te zetten
        out DDRC, r16 ;PORTC input                                                                ; I/
O                                 ; Direct                            ; Zet alle pins van PORTC als output
        ldi r16, 0xff
; REGISTER                          ; Inmediate                          ; Laad het getal 0x0 in Register
R16. Zie 2 naar boven.
        out DDRD, r16 ;PORTD output                                                                ;
I/O                                ; Direct                            ; Zet alle pins van PORTD als
output
BEGIN:
; Dit is een Label
        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 zegd tegen de
assembler dat wat hierna komt bij adress 20 begind
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
```

C.

Convert een byte (0-9) wat op port C staat naar de ASCII waarde ervan. Deze output wordt daarna naar port D geschreven.

10

A

```
.dseg
.org SRAM_START
dest: .byte 20

.cseg
        rjmp start

src: .db "hello world !"
```

```
.equ length = 13
.def temp = r0 ; label voor registers, andere naam voor de compiler
.def counter = r17

start:
        ldi zh, high(2*src); Z-pointer wijst naar adres src*2
        ldi zl, low(2*src)
        ldi xh, high(dest); X-pointer wijst naar begin
        ldi xl, low(dest)
        clr counter ; clear counter
loop:
        lpm ; Loads one byte pointed to by the Z-register into the destination register Rd.
        inc zl ; increment z-low
        st x+,temp
        inc counter ; increment counter
        cpi counter, length ; vergelijk
        brlt loop ; als false, ga verder met de loop.
end:
        rjmp end
```

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.

C

Om te zorgen dat de least significant bit ‘0’ is, daar kijkt LPM na om te bepalen welk deel die moet bekijken.

D

https://imgur.com/oXkW8yZ

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