

Assembly

Özgun M

January 31, 2016

Laborationsrapport för kursen Datorgrafik 2016 VT

Kjell Mårdensjö

Örebro Universitet

Contents

1	Laboration 1: Lysdioder & Strömställare	3
1.1	Uppgift 1 - Programmet Hello World	4
1.2	Uppgift 2 - Assemblerprogram för att tända och släcka en lysdiod med en strömbrytare	4
1.3	Uppgift 3 - Assemblerprogram för att tända och släcka en lysdiod med en switch	5
1.4	Uppgift 4 - Ringräknare	7
1.5	Uppgift 5 - Johnsonräknare	9
1.6	Uppgift 6 - Lysdiodsmönster	11

1 Laboration 1: Lysdioder & Strömställare

I kommande koduppvisning kommer läsaren att märka främst två typer av funktioner, `init_func` och `loop_func`, som är generella namn på två funktioner som används i assembly-koden. Notera att dessa namn får och kan ändras till ens eget tycke men i denna laboration var funktionsnamnen oförändrade. C-koden ser ut som på nedanstående sätt och står för grunden av assembly-programmeringen. Istället för att skriva funktionerna i vanlig C ska dessa skrivas i assembly-kod som står med i slutet av respektive uppgift.

```
#include <avr/io.h>

int main(void)
{
    init_func();

    while (1
    {
        loop_func();
        wait_milliseconds(300);
    }
}
```

I varje laboration kommer dessutom att fördefinierade värden för portarna och ingångarna att finnas:

```
;;;--- I/O-adresses for Port D ---
#define PIN_D    0x10
#define DDRD     0x11
#define PORTD    0x12

;;;--- I/O-adresses for Port C ---
#define PIN_C    0x13
#define DDRC     0x14
#define PORTC    0x15

;;;--- I/O-adresses for Port B ---
#define PIN_B    0x16
#define DDRB     0x17
#define PORTB    0x18

;;;--- I/O-adresses for Port A ---
#define PIN_A    0x19
#define DDRA     0x1A
#define PORTA    0x1B
```

1.1 Uppgift 1 - Programmet Hello World

Denna uppgift var främst för att prova lite lätt assembly och därför finns ingen redovisning.

1.2 Uppgift 2 - Assemblerprogram för att tända och släcka en lysdiod med en strömbrytare

Denna uppgift gick ut på att kunna tända en diod genom att trycka in respektive diod-knapp.

Algorithm 1 Tända/släcka en lysdiod med strömbrytare

```
1      .data
2
3  lamps: .byte 0    ;; unsigned char lamps = 0;
4
5      .text
6      .global init_func
7
8  init_func:
9
10     ;; DDRA = 0x00
11     LDI     R20, 0x00
12     OUT     DDRA, R20
13
14     ;; DDRB = 0xFF;
15     LDI     R20, 0xFF
16     OUT     DDRB, R20
17
18     RET
19
20
21     .text
22     .global loop_func
23
24  loop_func:
25
26     ;; PORTB = PINA
27     IN      R20, PINA
28     OUT     PORTB, R20
29
30     RET
```

1.3 Uppgift 3 - Assemblerprogram för att tända och släcka en lysdiod med en switch

Denna uppgift är har samma förutgrunder som föregående däremot är skillnaden att en switch används för att tända en lysdiod på ATmega32. När tredje switchen aktiveras lyser den första biten i registret som styr lysdioderna.

Algorithm 2 Tända/släcka en lysdiod med switch

```
1  lamps:  .byte 0          ;;  unsigned char lamps = 0;
2
3          .text
4          .global init_func
5
6  init_func:
7          ;; DDRA = 0x00
8          LDI      R20, 0x00
9          OUT      DDRA, R20
10
11         ;; DDRB = 0xFF;
12         LDI      R20, 0xFF
13         OUT      DDRB, R20
14
15         ;; PORTB = 0xFF;
16         OUT      PORTB, R20
17
18         RET
19
20         .text
21         .global loop_func
22
23  loop_func:
24         ;; R20 = PINA & 0x08
25         IN R20, PINA
26         COM R20
27         ANDI R20, 0b00001000
28
29         ;; R20 = R20 >> 3
30         LSR R20
31         LSR R20
32         LSR R20
33
34         ;; R21 = PORTB & 0xFE
35         IN R21, PORTB
36         COM R21
37         ANDI R21, 0xFE
38         OR R20, R21      ; R20 = R20 | R21
39
40         COM R20
41         OUT PORTB, R20 ; PORTB = R20
42
43         RET
```

1.4 Uppgift 4 - Ringräknare

Denna uppgift gick ut på att implementera en ringräknare. Processen för att skapa programmet följdes genom en figur från kurskompendiet som visar stegen för en lösning.

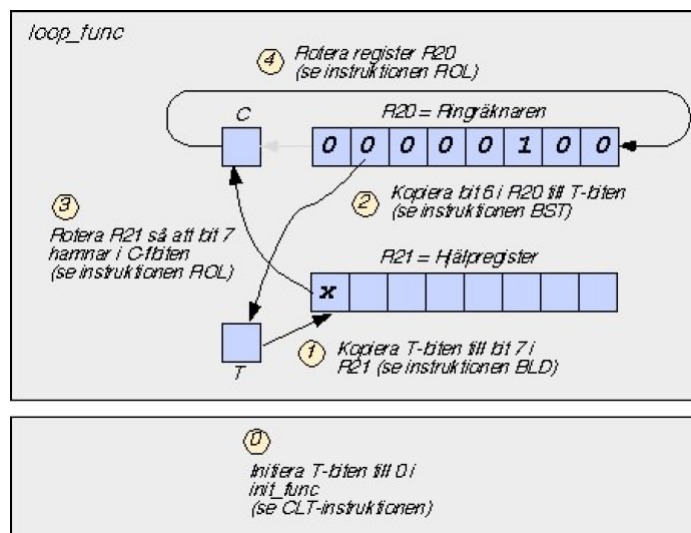


Figure 1: Ringräknare lösning genom register-användning

Som det syns i ovanstående figur finns det olika steg man kan följa. Notera att det finns alternativa lösningar till denna uppgift men denna valdes för att lättare kunna designa koden för ringräknaren. I denna lösning används processorns register medan de andra representerar lösningar som involverar en mönstertabell eller en if-else-sats. För enkelhetens skull gjordes denna uppgift med en lösning som använder processorns register genom T-bit utlisering.

Stegen som följdes demonstreras i nedanstående steg med tillhörande assembly-kod i algoritm-figuren.

1. I första steget initierades T-biten till 0 genom att använda CLT-instruktionen.
2. Andra steget inkluderade att kopiera T-biten till bit 7 i processorns R21-register som är användes som ett hjälpregister genom att använda instruktionen BLD.
3. Samma sak utfördes på registret R20, men i detta fall kopierades bit 6 till T-biten. R20 är ringräknaren, alltså registret som aktiverar lysdioderna.
4. Eftersom R21 fick bit 7 så roterades den så att den hamnar i C-biten som i nästa steg roterades in i ringräknaren-registret R20, båda använde instruktionen ROL.

Algorithm 3 Ringräknare genom registeranvändning

```
1      .data
2
3      .text
4      .global init_func
5
6 init_func:
7
8      LDI R20, 0xFF
9      OUT DDRB, R20
10     CLT
11
12     RET
13
14     .text
15     .global loop_func
16
17 loop_func:
18
19     BLD R21, 7
20     BST R20, 6
21     ROL R21
22     ROL R20
23     OUT PORTB, R20
24
25     RET
```

1.5 Uppgift 5 - Johnsonräknare

Koden som beskrivs nedan är i princip identisk mot ringräknar koden. Den enda skillnaden är att ett register inverteras i johnsonräknaren så att det fortsätter lysa även efter dioden har flyttat sig. I ringräknaren behövdes inte det eftersom det bara var en diod som skulle förflytta sig hela tiden.

Algorithm 4 Johnsonräknare genom registeranvändning

```
1      .data
2
3      .text
4      .global init_func
5
6  init_func:
7
8      LDI R20, 0xFF
9      OUT DDRB, R20
10     CLT
11
12     RET
13
14     .text
15     .global loop_func
16
17  loop_func:
18
19     BLD      R21, 7
20     BST      R20, 6
21     COM      R21      ;; Detta steg gör ringräknaren
                       till en johnsonräknare
22     ROL      R21
23     ROL      R20
24     OUT PORTB, R20
25
26     RET
```

Algorithm 5 Johnsonräknare genom if-else-användning

```
1  #define  vjohn R20
2  #define  one R21
3  #define  vjohn_and R22
4  vjohn_temp: .byte 0
5
6          .text
7          .global init_func
8
9  init_func:
10         LDI    vjohn, 0xFF
11         OUT    DDRB, vjohn
12         LDI    vjohn, vjohn_temp ;; R20 = vjohn <=> R20 = 0
13         LDI    one, 1    ;; R21 = 1
14         RET
15
16         .text
17         .global loop_func
18
19  loop_func:
20         LDS    vjohn, vjohn_temp ;; R20=vjohn
21         MOV    vjohn_and, vjohn ;; R22 = R20
22         ANDI   vjohn_and, 0x80    ;; R22 & 0x80
23         BREQ   if_equal
24         RJMP   else_f
25
26  if_equal:
27         LSL    vjohn ;; R20 << 1
28         ADD    vjohn, one    ;; R20 += R21
29         RJMP   end_loop
30
31  else_f:
32         LSL    vjohn ;; R20 << 1
33         RJMP   end_loop
34
35  end_loop:
36         STS    vjohn_temp, vjohn
37         COM    vjohn ;; Inverterar för lysdioderna
38         OUT    PORTB, vjohn
39         RET
```

1.6 Uppgift 6 - Lysdiodsmönster

I denna uppgift skulle ett mönster skapas. Mönstret i fråga ser ut som en johnsonräknare som i cykler räknar inåt och sedan utåt. En representation kan fås genom nedanstående figur:

```
00000000
10000001
11000011
11100111
11111111
11100111
11000011
10000001
```

Figure 2: Lysdiodsmönster

För uppgiften användes en lista för att hålla reda på vilka lysdioder som ska lysa. Loop-funktionen gick igenom en for-loop liknande sats som aktiverade respektive position i listan.

Algorithm 6 Lysdiodsmönster genom tabell

```
1      .data
2      .global LEDS
3
4  LEDS: .byte 0x00, 0x81, 0xC3, 0xE7, 0xFF, 0xE7, 0xC3,
        0x81
5
6      .text
7      .global init_func
8
9  init_func:
10     ;; i = 0
11     LDI R28, 0x00
12     LDI R29, 0x00
13
14     ;; DDRB = 0xFF
15     LDI R24, 0xFF
16     OUT DDRB, R24
17
18     RET
19
20
21     .text
22     .global loop_func
23
24  loop_func:
25     ;; PORTB = LEDS[i]
26     LDI R30, lo8(LEDS)
27     LDI R31, hi8(LEDS)
28
29     ;; LEDS + i - R31:R30 + R29:R28
30     ADD R30, R28
31     ADC R31, R29
32
33     LD R24, Z      ;; Z = R31:R30
34     /*COM R24*/    ;;ignore
35     OUT PORTB, R24
36     ADIW R28, 0x01 ;; i += 1
37
38     ;; i = i & 0x07
39     ANDI R28, 0x07
40     ANDI R29, 0x00
41
42     RET
```
