# ASSEMBLERPROGRAMMERING DEL 2

ÖZGUN MIRTCHEV

## Innehåll



*Date*: 2016-01-09 15:37:48.

**Anvisningar för svar i XCEL-dokument**

. Följande bör du tänka på då svaren skall anges i ett XCEL-dokument:

(1) Ändra ALDRIG format (.xls) när du sparar XCEL-dokumentet till något annat (.xlsx, etc.).
(2) Problem med inmatning av svar till vissa svarsceller där tecken tas bort av kalkylprogrammet.
   (a) Formatera svarscellerna genom att markera dessa och ändra formatet till TEXT.
   (b) Ett enkelt och generellt råd är att markera alla celler i svarskolumnen och ändra deras format till TEXT.
   (c) I OpenOffice använder du höger musknapp för att via kontextmenyn (Fomat Cells) formatera svarscellerna till formatet TEXT.

## 1. Negativt tal med ordlängden 8 bitar

*avr_2k_a.aw2*

Med vilket positivt decimalt tal kodas -84 i 8 bitars 2-komplementkod? Svara med tre decimala siffror.

## 2. Laddning av register R20 med en konstant

*avr_instruction_ldi_a.aw2*

Vilket binärvärde, 8 binära siffror, erhålls i register R20 i följande 3 fall:

```
LDI R20,244 ; Fall 1
LDI R20,0xF4 ;Fall 2
LDI R20,-12 ;Fall 3
```

## 3. Addition av två 8-bitars tal

*avr_add_b.aw2*

Skriv ett assemblerprogram som adderar två 8-bitars tal lagrade på minnesplatserna vU och vV. Resultatet skall lagras som ett 8-bitarstal på minnesplatsen vZ. Antag att följande är deklarerat:

```
        .data
        .global vU,vV,vZ
vU:     .byte   0
vV:     .byte   0
vZ:     .byte   0
```

Vilket av följande program är korrekt.

*a*) `ADD vZ,vU+vV`

*b*)
```
LDS R22,vU
LDS R23,vV
ADD R22,R23
STS vZ,R22
```
*c*)
```
LDI R22,vU
LDI R23,vV
ADD R22,R23
STS vZ,R22
```
*d*)
```
LDS R22,vU
LDS R23,vV
ADD R22,R23
STS vZ,R23
```

## 4. Subtraktion av tal

*avr_sub_a.aw2*

Skriv de instruktioner som behövs för att utföra följande tilldelningssats i C:

```
int va,vb;

va = vb-1;
```

Antag att variablerna va och vb är globala och skall deklareras i asssembler. I register R1 finns 0x00.

*a*)
```
            .data
            .global va,vb
va:       .word 0
vb:       .word 0

            .text
...

           LDS   R22,vb+1
           LDS   R23,vb
           SUBI R22,1
           SBC   R23,R1
           STS   va+1,R22
           STS   va, R23
```
*b*)
```
            .data
            .global va,vb
va:       .word 0
vb:       .word 0
```

```
                .text
...

            LDS   R22,vb
            LDS   R23,vb+1
            SUBI R22,1
            SBC   R23,R1
            STS   va,R22
            STS   va+1, R23
```

*c*)
```
             .data
             .global va,vb
va:       .word 0
vb:       .word 0

                .text
...

            LDS   R22,vb
            SUBI R22,1
            STS   va,R22
```

## 5. LSL,LSR,ASR och ASL

*avr_instruction_shift_a.aw2*

*a*) Antag att register R20 har värdet 11001101. Vilket värde har R20 efter att instruktionen LSL R20 har utförts?
*b*) Antag att register R20 har värdet 11011100. Vilket värde har R20 efter att instruktionen LSR R20 har utförts?
*c*) Antag att register R20 har värdet 11111001. Vilket värde har R20 efter att instruktionen ASR R20 har utförts?
*d*) Antag att register R20 har värdet 11000101. Vilket värde har R20 efter att instruktionen ASL R20 har utförts?

## 6. Vad innebär ett högerskift aritmetiskt?

*avr_instruction_shift_b.aw2*

Högerskift innebär aritmetiskt?

*a*) Multiplikation med 2
*b*) Division med 2

## 7. Handexekvering

*avr_if_else_a.aw2*

Vilket hexadecimalt värde har register R20 efter att nedanstående kod har exekverats?

```
        LDI R20, 105
        LDI R21, 18
        LDI R22, 18
        SUB R21, R22
        BRNE stop
        SUBI R20, 0xFF
stop:
        RJMP stop
```

Ange resultatet som två hexadecimala siffror.

## 8. BREQ

*avr_instruction_breq_a.aw2*

Vilken flagga i statusregistret och på vilket värde av denna utförs det villkorliga hoppet med branch-instruktionen BREQ?

*a*) C=1
*b*) C=0
*c*) Z=0
*d*) N=1
*e*) Z=1
*f*) N=0

## 9. BRSH eller BRGE

*avr_instruction_brge_brsh_a.aw2*

När två heltal av datatypen char, teckenbit, jämförs, skall ett hopp utföras då:

```
tal1 >= tal2
```

vilken hoppinstruktion skall användas?

*a*) BRGE
*b*) BRSH

## 10. SUB och CP

*avr_instruction_sub_cp_a.aw2*

Vilken skillnad föreligger mellan SUB- och CP-instruktionerna? Vad är rätt av följande påståenden?

*a*) Instruktionen CP finns ej, det är ett skällsord.
*b*) CP påverkar enbart flaggregistret SREG.
*c*) SUB påverkar enbart flaggregistret SREG.
*d*) Ingen.

## 11. IF-SATS

*avr_if_else_c.aw2*

Koda följande if-sats i assembler:

```
if ( uc >= 0 )
    uc=0x01;
```

där variabeln uc är en global variabel, deklarerad på följande sätt i C

```
unsigned char uc=0;
```

vilket i assemblern blir

```
        .data
        .global uc
uc: .byte 0
```

*a*)

```
        ;; if ( uc >= 0 ) //uc-0 >=0 -> then
L_TEST:
        LDS R20, uc
        LDI R21. 0
        CP R20,R21 //uc-0
        BRLT L_THEN
        JMP L_ELSE
        ;; uc=0x01;
L_THEN:
        LDI R20,0x01
        STS uc, R20
        JMP L_END ;Onödigt, då
        ;else-delen
        ;är tom
        ;;
L_ELSE:
L_END:
```

*b)*
```
        ;; if ( uc >= 0 ) //uc-0 >=0 -> then
L_TEST:
        LDS R20, uc
        LDI R21. 0
        CP R20,R21 //uc-0
        BRGE L_THEN
        JMP L_ELSE
        ;; uc=0x01;
L_THEN:
        LDI R20,0x01
        STS uc, R20
        JMP L_END ;Onödigt, då
        ;else-delen
        ;är tom
        ;;
L_ELSE:
L_END:
```

*c)*
```
        ;; if ( uc >= 0 ) //uc-0 >=0 -> then
L_TEST:
        LDS R20, uc
        LDI R21. 0
        CP R20,R21 //uc-0
        BRSH L_THEN
        JMP L_ELSE
        ;; uc=0x01;
L_THEN:
        LDI R20,0x01
        STS uc, R20
        JMP L_END ;Onödigt, då
        ;else-delen
        ;är tom
        ;;
L_ELSE:
L_END:
```

*d)*
```
        ;; if ( uc >= 0 ) //uc-0 >=0 -> then
L_TEST:
        LDS R20, uc
        LDI R21. 0
        CP R20,R21 //uc-0
        BRGE L_ELSE
        JMP L_THEN
        ;; uc=0x01;
```

```
   L_THEN:
           LDI R20,0x01
           STS uc, R20
           JMP L_END ;Onödigt, då
           ;else-delen
           ;är tom
           ;;
   L_ELSE:
   L_END:
```

## 12. IF-SATS

*avr_if_else_b.aw2*

Koda följande if-sats i assembler:

```
if ( uc >= 0 )
    uc=0x01;
```

där variabeln uc är en global variabel, deklarerad på följande sätt i C

```
signed char uc=0;
```

vilket i assemblern blir

```
        .data
        .global uc
uc: .byte 0
```

*a)*

```
           ;; if ( uc >= 0 ) //uc-0 >=0 -> then
   L_TEST:
           LDS R20, uc
           LDI R21. 0
           CP R20,R21 //uc-0
           BRLT L_THEN
           JMP L_ELSE
           ;; uc=0x01;
   L_THEN:
           LDI R20,0x01
           STS uc, R20
           JMP L_END ;Onödigt, då
           ;else-delen
           ;är tom
           ;;
   L_ELSE:
   L_END:
```

*b)*
```
        ;; if ( uc >= 0 ) //uc-0 >=0 -> then
L_TEST:
        LDS R20, uc
        LDI R21. 0
        CP R20,R21 //uc-0
        BRGE L_THEN
        JMP L_ELSE
        ;; uc=0x01;
L_THEN:
        LDI R20,0x01
        STS uc, R20
        JMP L_END ;Onödigt, då
        ;else-delen
        ;är tom
        ;;
L_ELSE:
L_END:
```

*c)*
```
        ;; if ( uc >= 0 ) //uc-0 >=0 -> then
L_TEST:
        LDS R20, uc
        LDI R21. 0
        CP R20,R21 //uc-0
        BSH L_THEN
        JMP L_ELSE
        ;; uc=0x01;
L_THEN:
        LDI R20,0x01
        STS uc, R20
        JMP L_END ;Onödigt, då
        ;else-delen
        ;är tom
        ;;
L_ELSE:
L_END:
```

*d)*
```
        ;; if ( uc >= 0 ) //uc-0 >=0 -> then
L_TEST:
        LDS R20, uc
        LDI R21. 0
        CP R20,R21 //uc-0
        BRGE L_ELSE
        JMP L_THEN
        ;; uc=0x01;
```

```
L_THEN:
        LDI R20,0x01
        STS uc, R20
        JMP L_END ;Onödigt, då
        ;else-delen
        ;är tom
        ;;
L_ELSE:
L_END:
```

## 13. Stackpekarens värde

*avr_subroutine_stack_b.aw2*

Följande programkod gäller i upgifterna:

```
0066 E070     LDI R23,0x00
0067 E184     LDI R24,0x14
0068 E090     LDI R25,0x00
0069 940E0070 CALL 0x0070
006B 93900061 STS 0x0061,R25
```

Antag att stackpekarregistret SP har värdet 0x0409. När programräknaren PC har ordadressen 0x0069 skall ett subrutinanrop göras med CALL-instruktionen. Vad händer med stackpekaren då CALL utförs? Ange stackpekarens nya värde som ett hexadecimalt tal.

## 14. CALL/RCALL

*avr_subroutine_call_rcall_a.aw2*

Vad är det som lagras på stacken då instruktionen CALL eller RCALL används?

*a*) Inget
*b*) Återhoppsadressen, adressen till nästföljande instruktion efter CALL/RCALL
*c*) Återhoppsadressen, adressen till positionen för den aktuella CALL/RCALL-instruktionen

## 15. Parametrar till e subrutin

*avr_subroutine_parameters_a.aw2*

Olika sätt finns att överföra parametrar till en subrutin, vilka av följande är fel?.

*a*) Via något av processorns register.
*b*) Via att använda globala variabler i dataminnet.
*c*) Via datastacken genom att lägga parameterdata på denna med PUSH-instruktionen
*d*) Via datastacken genom att lägga parameterdata på denna med POP-instruktionen.

## 16. Rätt och fel om subrutiner

*avr_subroutine_true_false_a.aw2*

Vad är fel om subrutiner?

*a*) En subrutin måste avslutas med ett RET för att kunna hitta tillbaka till anropsstället.
*b*) Om man hoppar till en subrutin med till exempel JMP spårar programmet ur totalt då RET-instruktionen exekveras.
*c*) När CALL/RCALL används lagras en återhoppsadress på stacken.
*d*) En stack är ej nödvändigt för att implementera subrutiner.

## 17. Återhoppsadressen och stacken

*avr_subroutine_stack_a.aw2*

Följande programkod gäller i upgifterna:

```
0066 E070     LDI R23,0x00
0067 E184     LDI R24,0x14
0068 E090     LDI R25,0x00
0069 940E0070 CALL 0x0070
006B 93900061 STS 0x0061,R25
```

Antag att stackpekarregistret SP har värdet 0x045F. När programräknaren har ordadressen 0x0069 skall ett subrutinanrop göras med CALL-instruktionen.
Hur ser stacken ut efter detta? Observera att Big-endian lagring görs av återhoppsadresser

*a*)
```
045F: ??
0460: 6B
0461: 00 <-SP
```
*b*)
```
045D: 00 <- SP
045E: 6B
045F: ??
```
*c*)
```
045F: 6B
0460: 00
0461: ?? <-SP
```
*d*)
```
045D: ?? <- SP
045E: 00
045F: 6B
```

## 18. Vilken instruktion används vid anrop av en subrutin?

*avr_jump_a.aw2*

En subrutin med namnet subAdd skall anropas, vilket är det korrekta sättet?

*a*) JMP subAdd
*b*) RJMP subAdd
*c*) BREQ subAdd
*d*) CALL subAdd

## 19. Tolkning av maskinkod

*avr_address_a.aw2*

Ett litet problemmed GNU-kompilatorn är att den arbetar med programminnet som byteorganiserat. AVR-processorn har ett ordadresserat programmminne. Där ett ord är två bytes. Följande objektkodsdump råder med byteadressering:

```
010C: 0E 94 89 04 CALL Func
```

På vilken byteadress i programminnet ligger subrutinen Func som skall anropas? Observera att i maskinkoden (0E 94 89 04) används ordadressen och ej byteadressen. Om du känner dig osäker på byteordning, studera då main.lss-filen. som finns i projektets default-katalog.
Ange byte adressen som ett fyra siffrors hexadecimalt tal .

## Bilaga A. Svarstyper

Olika typer av frågor förekommer såsom multipelt val, numeriskt värde, etc., varje typ av fråga har sin typ av svarsalternativ.

- Ordagrann jämförelse
  - Jämförelse görs bokstav för bokstav.
- Multipelt val (multiple choice)
  - Svarsalternativen är en bokstav A, B, C, etc. Rättningen är okänslig för om svaret anges med VERSALER eller gemener.
- Sant/falskt, rätt/fel, och ja/nej
  - Frågor av typen sant/falskt, rätt/fel och a/nej. anges med svarsvärden för det som är RÄTT/SANT/TRUE/YES med bokstäverna R, S, T, och Y och för det som är FEL/FALSE/NO med bokstäverna F och N. Svarsalternativet fungerar både med VERSALER och gemener. För sant kan även siffran 1 användas och för falskt siffran 0.
- Kommaseparerad lista av ord
- Numeriskt värde med storhet
  - Numeriska svarsvärden är heltal eller reella tal och anges med DECIMALPUNKT istället för decimalkomma.
  - Exempel 1: 125.3
- Numeriskt värde med storhet och enhet
  - Enheter som understödjs:
    - Alla enheter som representeras med 1 tecken understödjs, t.ex. $s$ för sekunder, $m$ för meter, etc.
    - ohm - anges som ohm.
    - Ett blanktecken skall finnas mellan det numeriska värdet och enheten.
  - Prefix som understödjs t.ex. n för ns (nanosekunder):
    - m - milli
    - u - mikro
    - n - nano
    - p - pico
    - k - kilo
    - M - Mega
    - G - Giga
    - T - Terra
- Enkelt matematiskt uttryck
  - Exempel 1: y=x**2+3*x+4
  - Exempel 2: 24*12+3
- Hexadecimal sträng
  - Exempel 1: bb45
  - Exempel 2: 0x00A6

– Exempel 3: 0XFFE3

Svaren är ej känsliga för extra vita-tecken (blanktecken, etc.) då t.ex tre blanktecken efter varann ersätts med ett blanktecken.

A.1. **Decimalpunkt.** I alla svar som kräver ett reellt tal som svar används decimalpunkt istället för decimalkomma.

A.2. **Enkla matematiskt uttryck.** Ett enkelt matematiskt uttryck har en syntax som motsvaras av det man finner i programmeringsspråk som C, MatLab, Ruby, etc. Då små detaljer kan skilja olika programmeringsspråk åt.
Observera att skillnad föreligger mellan HELTALSDIVISION och division av reella all så kallad FLYTTALSDIVISION. Ex 1: 10 / 6 ger som resultat 1, däremot så ger 10.0 / 6 resultat 1.666667. Den enkla regeln är att om en av operanderna vid division är ett reellt tal fås en flyttalsdivsion, dvs en helt normalt resultat med decimaldelar. Om båda operanderna är ett heltal erhålls alltid ett heltalsresultat av division dvs decimaldelarna kastas.
I exemplena som följer i tabellen har variablerna a och b heltalsvärdena 10 respektive 20 och c och d de reella värdena 10.0 respektive 20.0.

| Operator | Beskrivning | Exempel |
|---|---|---|
| + | Additionsoperatorn | a+b ger som resultat 30 |
| - | Subtraktionsoperatorn | a-b ger som resultat -10 |
| * | Multiplikationsoperatorn | a*b ger som resultat 200 |
| / | Divisionsoperatorn | a/b ger som resultat 0 (heltalsdivision)<br>c/d ger som resultat 0.5 (flyttalsdivision, reell division) |
| % | Modulusoperatorn | |
| ** | Exponent | a ** b ger a upphöjt i b dvs 10 upphöjt i 20<br>b ** 2 ger som resultat 400 |

Följande matematiska funktioner kan användas:

| Funktion | Beskrivning | Exempel |
|---|---|---|
| acos | Arcuscos | |
| asin | Arcussinus | |
| atan | Arcustangens | |
| cos | Cosinus | |
| exp | Exponentialfunktionen | |
| log | e-logaritmen (ln) | |
| log10 | 10-logaritmen | log10(100) ger värdet 2 |
| log2 | 2-logaritmen | |
| sin | Sinus | sin(PI/2) ger värdet 1.0 |
| sqrt | Kvadratroten | sqrt(4) ger värdet 2 |
| tan | Tangens | |

A.3. **Enkla differentialekvationsuttryck.** En differentialekvation som t. ex. $\frac{d^2y}{dt^2} + 3\frac{dy}{dt} + 2y = 3\frac{du}{dt} + 2u$

anges som ett enkelt differentialekvationsuttryck anges på följande form:
y”+3*y’+2*y = 3*u’ + 2*u
Den enkla regeln är att varje derivering anges med en apostrof, dvs för andraderivatan blir det två apostrofer efter varann.