# Datorteknik
# Övningsuppgifter

Stefan Gustafsson, Michael Josefsson

ver 0.4 2017-10-11

OBS! I uppgifterna får du själv mata in relevanta data för programmen. Vid simulering nollställs samtliga register och minnen. Använd exempelvis instruktionerna `ldi`, `sts` för detta.

Ladda ned AtmelStudio från `atmel.com`. På youtube ligger en liten introduktion till AtmelStudio på `https://youtu.be/3oljaclTg6M` som du kan ha nytta av för att hitta runt i miljön. Det är samma miljö som används vid laborationerna så du måste kunna den.

## 1.1 Assemblerprogrammering

1. Flytta innehållet i minnescell `$110` till minnescell `$112`.
2. Addera innehållet i minnescellerna `$110` och `$111`. Placera summan i minnescell `$112`.
3. Skifta innehållet i minnescell `$110` en bitposition åt vänster och placera resultatet i minnescell `$111`. Den minst signifikanta biten (*Least Siginificant Bit, LSB*, bit 0) skall vara nollställd i resultatet.
4. Flytta de fyra mest signifikanta bitarna (*Most Significant Bit, MSB*) i minnescell `$110` till de fyra minst signifikanta bitarna i minnescell `$111`. De fyra mest signifikanta bitarna i minnescell `$111` skall vara nollställda efter instruktionen.
5. Dela innehållet i minnescell `$110` i två fyrabitarsdelar (s.k. *nibbles* eller *nybbles*). Lagra de mest signifikanta bitarna i bit 3–0 i minnescell `$111` och de minsta signifikanta bitarna i bit 3–0 i minnescell `$112`.
6. Minnescellerna `$110` och `$111` innehåller två heltal i binär representation utan tecken. Placera det största av talen i minnescell `$112`.
7. Beräkna kvadraten på ett fyrabitars tal $X$ i binär representation utan tecken. Talet finns lagrat i minnescellen `$110`. Placera resultatet i minnescell `$112`.
8. Gör uppgift 7 men med hjälp av denna tabell utplacerad i FLASH-minnet:

| Adress | Innehåll | Kommentar |
|---|---|---|
| BAS+0 | $00 | $0^2 = 0$ |
| BAS+1 | $01 | $1^2 = 1$ |
| BAS+2 | $04 | $2^2 = 4$ |
| BAS+3 | $09 | $3^2 = 9$ |
| BAS+4 | $10 | $4^2 = 16$ |
| BAS+5 | $19 | $5^2 = 25$ |
| BAS+6 | $24 | $6^2 = 36$ |
| BAS+7 | $31 | $7^2 = 49$ |
| BAS+8 | $40 | $8^2 = 64$ |
| BAS+9 | $51 | $9^2 = 81$ |
| BAS+A | $64 | $10^2 = 100$ |
| BAS+B | $79 | $11^2 = 121$ |
| BAS+C | $90 | $12^2 = 144$ |
| BAS+D | $A9 | $13^2 = 169$ |
| BAS+E | $C4 | $14^2 = 196$ |
| BAS+F | $E1 | $15^2 = 225$ |

9. Gör uppgift 8 igen men kopiera först över tabellen till SRAM och använd denna tabell istället.

10. Bestäm kvadraten på talet $X$ i föregående uppgift med hjälp av instruktionen `MUL` utan att använda någon tabell. Vilka för- och nackdelar finns med de olika metoderna?

11. Konvertera ett tal $X$ ($X \in [0, 15]$) till ASCII-koden för motsvarande hexadecimala siffra. Talet $X$ finns i minnescell `$110`. Lagra resultatet i minnescell `$112`.

    Lös gärna uppgiften både med och utan tabellslagning. Du får skapa eget tabellinnehåll.

12. Översätt följande programsnutt i högnivåspråk till assembler. Talen `START` och `SLUT` är heltal som representeras med 8 bitar i tvåkomplementsform och ligger lagrade i adresserna `$110` och `$114`. Variabeln `index` lagras lämpligen i ett dataregister:

```
for(index = START; index <= SLUT; index++){
...
}
```

13. Uttryck `for`-loopen

```
for(expr1; expr2; expr3){
...
}
```

    som en `while`-loop. Går den att skriva som en `do`-loop?

14. Översätt följande programsnutt i högnivåspråk till assembler. Variabeln `x` representeras med 8 bitar i binär kod utan tecken och ligger lagrad i register `r16`.

```
while (x > 10){
    x = x -5;
}
```

15. Skriv ett program som väljer mellan tre olika programvägar beroende på innehållet i minnescell `$101`. Om minnescellen innehåller `$01` skall hopp ske till rutinen `ETT`, för `$02` skall hopp ske till rutinen `TVA` och för `$03` skall hopp ske till `TRE`. Använd instruktionen `breq` för hoppen. Vad händer i ditt program om minnescellen innehåller ett annat tal?

16. Skiftinstruktioner förekommer ofta.

    a) Vad är skillnaden mellan aritmetiskt och logiskt högerskift?

    b) Vad är skillnaden mellan aritmetiskt och logiskt vänsterskift?

17. I en centralenhet (*Central Processing Unit*, CPU) finns ofta ett statusregister med så kallade flaggor, där man vanligen finner bitarna `C`, `V`, `Z` och `N`. Förklara hur dessa fyra flaggor påverkas av olika instruktioner.

18. En assemblerinstruktion består av två delar. Den ena är operationskoden, som anger vilken slags operation som skall utföras. Vilken är den andra delen? Måste den alltid finnas?

19. Vad är skillnaden mellan absolut och relativ adressering? Vilka fördelar har man av att använda relativ adressering där det är möjligt?

20. Vid ett visst tillfälle innehåller de interna registren följande:

| Register | Innehåll |
|---|---|
| r20 | $61 |
| r21 | $F5 |
| X | $1F0 |

SRAM innehåller samtidigt följande:

| Cell | Innehåll |
|---|---|
| $100 | $19 |

Beräkna vilka register och minnesceller som påverkas av följande instruktioner, och ange det nya innehållet efter det att instruktionen utförts. Instruktionerna antas utföras med de angivna värden var och en för sig från samma utgångstillstånd.

a) `mov r16,r20` b) `lds r20,$100` c) `clr r21`
d) `inc r21` e) `sts $100,r19` f) `ldi r20,$F5`
g) `ld r17,X` h) `ld r17,X+` i) `mul r20,r21`

21. a) Hur sker anrop av en subrutin?
    b) Hur sker återhopp till huvudprogrammet?
    c) Varför kan man inte använda en vanlig hoppinstruktion för återhoppet?
    d) Kan en subrutin anropa en subrutin?
    e) Kan en subrutin anropa sig själv?
    f) Finns det någon begränsning för antalet möjliga nivåer av subrutinanrop?
    g) Vad bör man tänka på när man använder interna register i en subrutin?
22. Parametrarna till en omfattande subrutin får sällan plats i de interna registren. I sådana fall lägger man oftast parametrarna på stacken innan subrutinen anropas.
    a) Antag att en 16-bitars adress lagts på stacken varefter en subrutin anropas. Hur når man argumentet? Hämta talet till `r30:r31`.
    b) Vilken av `r30:r31` är minst signifikant byte?
    c) Vad måste man tänka på efter återhoppet från subrutinen? Vad händer annars?
23. Använd logiska instruktioner (`andi`, `ori`, `eor` och `com`) för att göra följande:
    a) Nollställ register `r16`.
    b) Ettställ de fyra mest signifikanta bitarna i `r16`.
    c) Nollställ de tre minst signifikanta bitarna i `r16`.
    d) Utför bitvis `NOR` på register `r16` och `r17`. Lägg resultatet i `r18`.
    e) Invertera bitarna 2 till 6 i `r16`.
    f) Flytta de fyra minst signifikanta bitarna i `r16` till de fyra minst signifikanta bitarna i `r17`. Övriga bitar i `r17` skall inte ändras. Innehållet i `r16` får förstöras.
24. Skriv en subrutin som beräknar en udda paritetsbit till en 7-bitars ASCII-kod som finns i de sju minst signifikanta bitarna i `r20`. Utdata från subrutinen skall vara ASCII-koden kompletterad med paritetsbiten i bit 7. Lagra även resultatet i `D1`.

    Vilka felfall kan inträffa? Vad händer om paritetsbiten redan är satt?
25. En väluppfostrad subrutin bör inte ändra innehållet i de register som inte uttryckligen används för överföring av parametrar. Komplettera subrutinen från föregående uppgift så att det utåt sett inte ändrar något av registren utöver `r20`.

26. Beräkna medelvärdet av ett antal 8-bitars positiva tal lagrade i minnet med start på en adress som anges av innehållet i `X`. Antalet tal är högst 256. Det sista talet är ett negativt tal som markerar slutet på serien. Lagra medelvärdet i `r16`, och lagra antalet tal i `r17`.

    Vilka felfall kan inträffa? Vad händer i ditt program om antalet tal är noll? Hur kan man gardera sig mot fel i detta fall?

27. Tillverka en *odometer*, en BCD-räknare med fyra siffror som räknar från `0000` till `9999`. Gör komplett lösning med strukturdiagram, assemblerkod och avslutande simulering i Atmelstudio. Använd registren `r20`–`r23` för de olika siffrorna.

    Variant:

    1) Kläm ihop två BCD-siffror i en byte för att spara plats (viktigt i en mikrocontroller).
    2) Lägg siffrorna i SRAM istället. Blev det bättre? Sämre? Lika bra?
    3) Vilka ändringar behövs för att räkna till något annat maxvärde än `9999`? (`2359` verkar lämpligt för ett digitalur till exempel.)

28. Skriv en subrutin som med ett argument `n` plockar ut den `n`:te byten ur en lista i FLASH-minnet (programminnet). Listan är högst 64 bytes lång. Använd assemblerinstruktionen `lpm`.

    Variant:

    1) Hindra uppslagning utanför listan.
    2) Se till att inga register onödigtvis påverkas.

29. Skriv ett assemblerprogram som avgör vilka operationer $op \in \{+, -, *\}$ som ger störst resultat av uttrycket

    $$A\, op\, B\, op\, C\, op\, D$$

    Där talen $A$–$D$ är givna BCD-kodade siffror.

    Utvärderingen ska ske i den ordning siffrorna kommer dvs prioriteringsregler kan bortses från. Till exempel utvärderas $A = B = C = D = 5$ dvs $5 + 5 * 5 - 5$ stegvis till $5 + 5 = 10, 10 * 5 = 50, 50 - 5 = 45$ med slutresultatet $45$.

    1) Finn en bra datastruktur
    2) Skriv strukturdiagram
    3) Programmera och simulera i assembler

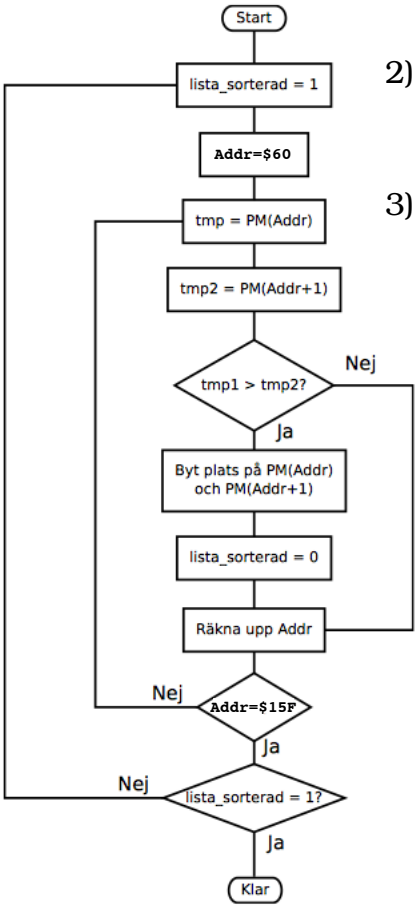
30. Använd instruktionerna `sts/lds`, `st/ld` respektive `std/ldd` för att komma åt variabler i SRAM.

    Variant:

1) Använd dem för att speciellt komma åt dataregistren `r0` osv.

31. Skriv den subrutin som korrekt hämtar sin argumentbyte från stacken om den anropande funktionen placerat argumentet där innan subrutinhoppet. Rita karta över stackinnehållet.
32. Man kan använda pekarna `X`, `Y` och `Z` i assemblerarkitekturen. Men alla kan inte peka på allt. Vilka begränsningar finns?
33. Skriv ett program som sorterar minnesinnehållet i SRAM-adresserna `$60`–`$15F` (255 bytes) i storleksordning. Se till att sorteringen fungerar även om flera värden i minnet är lika.

    Algoritmen *bubbelsort* är relativt enkel att beskriva i ett flödesschema (PM betyder här *primärminne* dvs vårt SRAM):




Variant:

1) Översätt flödesschemat till strukturdiagram.
2) Optimera koden så antalet klockcykler blir så få som möjligt
3) Hur ska listan vara beskaffad för att algoritmen ska ta så många cykler som möjligt? Så få som möjligt?

34. Skriv en subrutin som beräknar kvadratroten ur ett sextonbitars teckenlöst tal $N$ som befinner sig i `r25:r24`. `r24` innehåller den lägre halvan av talet och det i denna byte resultatet skall lagras. `r25` är noll vid återhoppet. Tips, om du behöver, finns i fotnoten[1].

## 1.2 In- och utmatning

35. Antag att I/O-registren är laddade med följande data:

| Register | Innehåll |
|---|---|
| DDRA | $F0 |
| DDRB | $0F |

    Vilka av dataledningarna på portarna `PORTA` och `PORTB` är in- respektive utgångar?

    Vilka instruktioner används för att läsa in respektive skriva ut data på dessa portar?

36. Skriv kod för att använda `PORTA` bitar 0, 2, 4 och 6 och `PORTB` bitar 1, 3, 5 och 7 som utgångar och övriga dataledningar som ingångar.

37. I databladen nämns *active pull-up* på vissa I/O-pinnar. Vad betyder det?

## 1.3 Avbrott

38. Ge exempel på några orsaker som kan resultera i avbrott?

39. Förklara skillanden mellan instruktionerna `ret` och `reti`. Vad händer om man blandar ihop dem?

40. På vilka adresser ligger avbrottsvektorn för A/D-omvandling?

41. Ge exempel på när det kan vara lämpligt att använda externt avbrott. Vad är alternativet?

42. Vad bör man tänka på när man använder interna register i en avbrottsrutin?

[1] Newton-Raphsons metod är i allmänhet snabb och har kvadratisk konvergens. Tyvärr innehåller den division vilket är mödosamt för vår processor. Den har dock en `MUL`-instruktion så... Prova bit för bit med början i `$80`, kvadrera och jämför, om kvadraten är större än $N$ kan man fortsätta prova med `$40` osv. Är kvadraten mindre än $N$ fyller man på "provroten" med en bit `$60`, provar igen osv. Metoden kan göras överraskande kompakt på AVR-arkitekturen: Under 100 klockcykler för en rotdragning!

## 1.4 Aritmetik

43. Omvandla följande decimala tal till binär representation:

    a) $5$ b) $15$ c) $25$ d) $1000$ e) $2047$ f) $65$ g) $17$

44. Omvandla talen i uppgiften ovan till hexadecimal representation.

45. Hur kan 8875 + 3455 vara 13441?

46. Omvandla följande tal till binär form:

    a) $F_{16}$ b) $67_{16}$ c) $FE_{16}$ d) $2C_{16}$ e) $6A_{16}$ f) $0A_{16}$ g) $ABCD_{16}$

47. Utför följande binära additioner. Kontrollera resultatet genom att omvandla talen till decimal representation och addera dem i decimal form också.

    a) $10110_2 + 111_2$
    b) $100_2 + 110_2$
    c) $111111_2 + 1 + 1 + 1$
    d) $1011011_2 + 111001_2 + 1100_2$
    e) $111111_2 + 111111_2 + 111111_2$

48. Utför uppgiften ovan men antag att talen är sexbitars tvåkomplements tal. Notera speciellt eventuellt spill.

49. Vilken talbas ges med symbolerna ”0, 1, 2, ..., A, B, ... Z” (dvs 0 till Z ingår)? Vilket decimalt tal är då XYZ?

50. Beräkna den åttabitars binära tvåkomplementsrepresentationen av följande decimala tal:

    a) $-5$ b) $25$ c) $-25$ d) $31$ e) $-100$ f) $-1$ g) $-65$

51. Hur ser en talbas ut om den uttrycks i den egna basen?

52. Utför följande binära subtraktioner. Betrakta talen som åttabitars tvåkomplements tal. Kontrollera resultatet genom att omvandla talen till decimal representation.

    a) $00010111_2 - 00000011_2$ b) $01011101_2 - 00111110_2$ c) $11010_2 - 1110001_2$

53. På samma sätt som man har tvåkomplement i basen två borde man kunna ha tiokomplement i decimalbasen? Hur skulle man göra då? Vilka tiokomplementkodade tal kan man representera med enbart två siffror?

54. Det binära bitmönstret $11010000$ kan tolkas på två sätt. Ange det decimala tal som bitmönstret representerar i följande fall:

    a) Talet representeras som ett tal utan tecken.

    b) Talet representeras på 2-komplementformat.

55. Översätt textsträngen ”MIKROdator” till ASCII-kod. Hur kan man veta att strängen är slut?

56. Ange en enkel metod att översätta siffrorna $0$ – $9$ till ASCII-kod? Antag att siffrorna finns i sin binära representation $0000_2$ – $1001_2$ i lägre delen av en byte.

## 1.5 Övrigt

Här kommer lite att fundera på för den extra intresserade studenten.

### 1.5.1 Högnivåspråk

På den låga hårdvarunära nivå vi hittills programmerat har det framgått att det är ett mycket tunt lager fernissa som skiljer oss från digitalteknikens brutala verklighet. Man kan förstå att våra processorinstruktioner är ”väl” valda för att motsvara det en programmerare normalt vill kunna utföra utan att se för mycket av den underliggande hårdvaran.

Ett ytterligare steg upp i abstraktionsnivå kan vi få genom att programmera i ett högnivåspråk. I utvecklingsmiljön Atmel Studio kan man även skapa projekt som programmeras i språket C. C-kompilatorn översätter sedan C-programkoden till assembler innan en avslutande kompilering till Dalia-kortet sker.

Skriv om någon laboration i C. Studera den resulterande assemblerkoden.

Frågeställningar:

a) Hur implementeras till exempel if-, for-, och switch-satser i assembler? Kan du göra en bättre implementation? Lek runt med olika variabeltyper (int, char, long, float?) vad händer med assemblerkoden?

b) I ett C-projekt kan graden av kompilatoroptimering göras. -O0 är optimeringsfritt medan -O2 är en vanlig, rätt aggressiv, optimering. Känner du igen din kod efter kompilatorns optimering?

c) Industriellt är i praktiken C allenarådande vid denna typ av lågnivåprogrammering. Varför? Varför tillåts i så fall överhuvudtaget assemblerprogrammering?

### 1.5.2 Arkitekturer

I ett berömt paper från 1945 beskriver matematikern John von Neumann det som senare skulle kallas för von Neumann-arkitekturen i datorkonstruktionskretsar. Ungefär samtidigt snickrade man på Harvard

fram en alternativ arkitektur som passande nog kom att kallas Harvard-arkitekturen.

Denna vinjett handlar om dessa två arkitekturer. Det är ett enormt fält varför vi bara kan nosa lite översiktligt på det. Förhoppningen är att ändå få en uppfriskande inblick vad som är väsentligt för data- och programflöden i processorer/kontrollers.

Lämpliga frågeställningar äro:

a) Vad skiljer i grova drag de olika arkitekturerna åt?

b) Vilken typ är vår mikrokontroller i kursen?

c) Hur har man dragit fördel av detta i vår mikrokontroller?

d) Hitta andra mikrokontrollerfamiljer. Fråga b) osv.