Datorteknik
Övningsuppgifter

ver 0.92 2020-11-05

Innehåll

1. Assemblerprogrammering

Operationer mellan register och minne är grundläggande.

Det är en bra idé att provköra uppgifterna i AtmelStudio. Det är samma miljö som används vid laborationerna så du måste kunna den. Vid simulering nollställs samtliga register och minnen, i verkligheten garanteras enbart att I/O-registren nollställs.

1. Ladda den decimala konstanten 198 i `r16` och den hexadecimala konstanten `64` (=100 decimalt) i `r17`, samt den binära konstanten `10010011` i `r18`
2. Kopiera innehållet i `r16` till `r18`
3. Kopiera innehållet i minnescell `$110` till minnescell `$112`.
4. Addera innehållet i minnescellerna `$110` och `$111`. Placera summan i minnescell `$112`.
5. Skifta innehållet i minnescell `$110` en bitposition åt vänster och placera resultatet i minnescell `$111`.
6. Nollställ de fyra mest signifikanta bitarna i `r16` (de fyra minst signifikanta bitarna ska vara opåverkade)
7. Ettställ de tre mest signifikanta bitarna i `r16` (övriga bitar ska vara opåverkade)
8. Innehållet i minnescellen `$110` kan ses som två fyrabitarsdelar (s.k. *nibbles* eller *nybbles*). Lagra de mest signifikanta bitarna (7–4) i bit 3–0 i minnescell `$111` och de minsta signifikanta bitarna (3–0) i bit 3–0 i minnescell `$112`.
9. Minnescellerna `$110` och `$111` innehåller två heltal i binär representation utan tecken. Placera det största av talen i minnescell `$112`.
10. Beräkna $10 \cdot X$ på ett fyrabitars tal X i binär representation utan tecken. Talet X finns lagrat i minnescellen `$110`. Placera resultatet i minnescell `$112`.
11. Addera talet `7` till register `r16`
12. Innehållet i registerparet `r31:r30` innehåller tillsammans ett 16-bitars tal. Öka på 16-bitars-talet med 3.

13. Innehållet i registerparet `r17:r16` innehåller tillsammans ett 16-bitars tal. Öka på 16-bitars-talet med 3.

14. Använd instruktionerna `ld/st` respektive `ldd/std` för att komma åt data i SRAM. Dvs, du ska inte använda instruktionerna `lds/sts`, utan `ld/st` samt `ldd/std` i kombination med en pekare (`X`, `Y` eller `Z`).

    a) Kopiera innehållet i minnescell `$110` till minnescell `$111`

    b) Addera innehållet i minnescell `$110` med innehållet i minnescell `$111`, spara resultatet i minnescell `$112`. (Tips: Använd Y-pekaren, och ladda den bara en gång)

    c) På adresserna `$110` till `$119` finns parvisa tal som ska adderas två och två, och resultaten ska sparas med början på adress `$120`. Dvs summan av innehållet i adress `$110` och adress `$111` ska sparas på adress `$120`, och summan av innehållet i adress `$112` och adress `$113` ska sparas på adress `$121`, osv. Lösningen bör lämpligen innehålla en loop. Går det att lösa med bara en pekare?

    d) Gör ett program som tilldelar `r0` till `r15` med `0` till `15`. Alltså `r0=0`, `r1=1`, `r2=2` osv. Programmet får bara använda en pekare samt register `r16`.

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

Register	Innehåll
r20	$61
r21	$F5
X	$100

    SRAM innehåller samtidigt följande:

Cell	Innehåll
$100	$19

    Avgör 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 ursprungsvärden var och en för sig från samma utgångstillstånd.

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

16. Använd logiska instruktioner (som `and`, `andi`, `or`, `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.

17. Översätt ASCII-textsträngen ”MIKROdator” till hexadecimala tal. Hur läggs den in i FLASH-minnet. Hur kan man veta att strängen är slut?

18. Ett program skall skriva ASCII-tecken i `r16` på en display. Men något är fel, strängen ’`ABCDEFG`’ skrivs som ’ `$4DTdt`’. Varför? Vad är fel? Hur kan man åtgärda felet?

19. Om BCD-räknare. Skriv kod som upprepat, från noll,

    a) räknar upp `r16`:s lägre halva bcd-kodat.

    b) räknar upp `r16`:s övre halva bcd-kodat. (Här kan man tänka sig två sätt.) Registrets andra halva får inte påverkas.

20. Skriv en subrutin som med ett argument `n` i `r16` 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.

21. Beräkna kvadraten på ett tal i X i `r16`:s lägre halva med hjälp av en 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	$15^2 = 225$

22. 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?
23. Konvertera en siffra i `r16`$\in [0, 9]$ till sin ASCII-kod.
24. 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?
25. 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 `r25:r24`.
    b) Vad måste man tänka på efter återhoppet från subrutinen? Vad händer annars?
26. I `r16` finns, i binär form, de två sista siffrorna av ett årtal. Året 1942 ger `r16=42` osv.

    Ett årtal är ett skottår om talet är jämnt delbart med 4 eller 400.
    a) Skriv en subrutin som avgör om talet i `r16` är ett skottår eller inte.
    b) Antag istället att `r24:r25` innehåller hela årtalet (0000..9999). Skriv en subrutin som avgör om året är ett skottår eller inte.

    Uppgifterna nedan kan utföras på papper eller med hjälp av AtmelStudio.
27. Omvandla följande decimala tal till binär representation:
    a) 5 b) 15 c) 25 d) 1000 e) 2047 f) 65 g) 17
28. Omvandla talen i uppgiften ovan till hexadecimal representation.
29. 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) $ABCD_{16}$
30. 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) $111111_2 + 111111_2 + 111111_2$
31. Beräkna den åttabitars binära tvåkomplementsrepresentationen av följande decimala tal:
    a) -5 b) 25 c) -25 d) 31 e) -100 f) -1 g) -65

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

33. 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) $11010_2 - 1110001_2$

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

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

36. 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.

37. 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?

38. 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.

39. Det finns några intressanta bit-manipulerande knep som man kanske inte genomskådar omedelbart. Här visas några att analysera. Frågeställningen är ”vad gör koden”? Ansätt några indata och ta reda på det!

a)
```
eor     r16,r17
ret
```

b)
```
mov     r17,r16
andi    r17,$AA
lsr     r17
andi    r16,$55
lsl     r16
or      r16,r17
mov     r17,r16
andi    r17,$CC
lsr     r17
lsr     r17
andi    r16,$33
lsl     r16
lsl     r16
or      r16,r17
mov     r17,r16
andi    r17,$F0
lsr     r17
lsr     r17
lsr     r17
lsr     r17
andi    r16,$0F
lsr     r16
lsr     r16
lsr     r16
lsr     r16
or      r16,r17
ret
```

c)
```
mov     r17,r16
andi    r17,$AA
lsr     r17
andi    r16,$55
add     r16,r17
mov     r17,r16
andi    r17,$CC
lsr     r17
lsr     r17
andi    r16,$33
add     r16,r17
mov     r17,r16
swap    r17
add     r16,r17
andi    r16,$0F
ret
```

1.1. Quiz-frågor

Här följer lite blandade frågor som inte kräver programmering.

40. 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.

41. Skiftinstruktioner förekommer ofta.

    a) Vad är skillnaden mellan aritmetiskt och logiskt högerskift?

    b) Vad är skillnaden mellan aritmetiskt och logiskt vänsterskift?

42. 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?

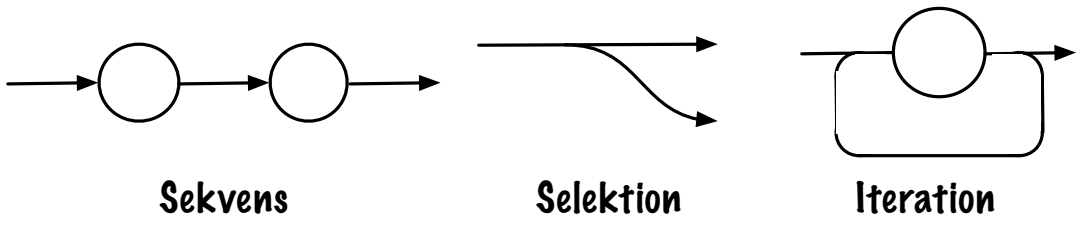
43. Vad är skillnaden mellan absolut och relativ adressering? Vilka fördelar har man av att använda relativ adressering där det är möjligt?
44. I databladen nämns *active pull-up* på vissa I/O-pinnar. Vad betyder det?
45. Om subrutiner
    a) Hur sker anrop av en subrutin?
    b) Hur sker återhopp till 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?
46. Ge exempel på några orsaker som kan resultera i avbrott?
47. Förklara skillanden mellan instruktionerna `ret` och `reti`. Vad händer om man blandar ihop dem?
48. Var återfinns avbrottsvektorn för A/D-omvandling?
49. Hur utformar man en avbrottrutin som använder interna register?

2. Strukturerad programmering

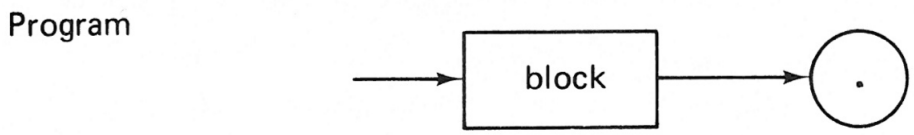
För att öva på konstruktion av JSP-diagram kommer några *syntaxdiagram* att tolka strukturerat. Syntaxdiagram kanske kan vara bra att ha sett någon gång men här finns de med enbart som underlag till JSP-övningarna.

Ett syntaxdiagram är en beskrivning av vilka komponenter som bygger upp ett programmeringsspråk. Diagrammet kan vara ett steg i processen att skriva en kompilator för språket. Här ska vi använda syntaxdiagrammen för att konstruera motsvarande strukturdiagram.

Man läser ett syntaxdiagram från vänster till höger och strukturkomponenterna *sekvens*, *iteration* och *selektion* återfinns i diagrammet fast inte i JSP-form.

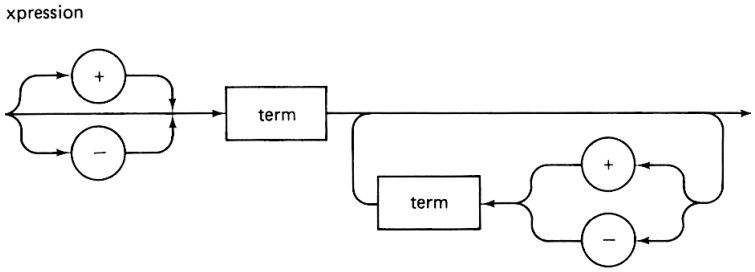


Nedan visas några syntaxdiagram att översätta till JSP-notation. Diagrammen definierar exempelspråket PL/0.[1]

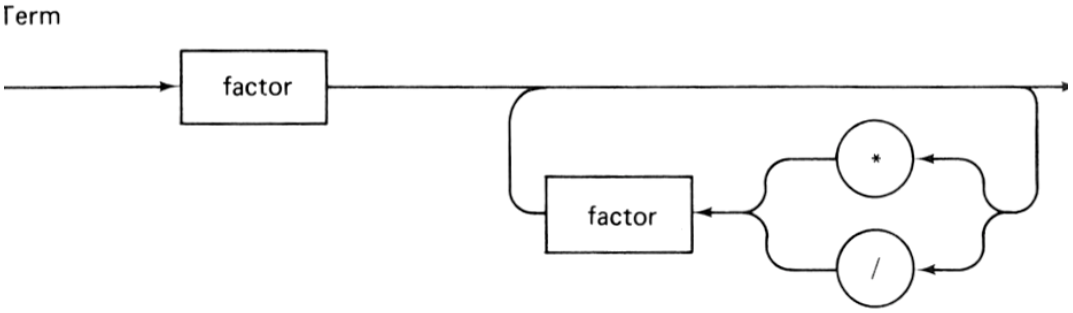


Ett PL/0-program består av ett **block** *följt av en punkt.*

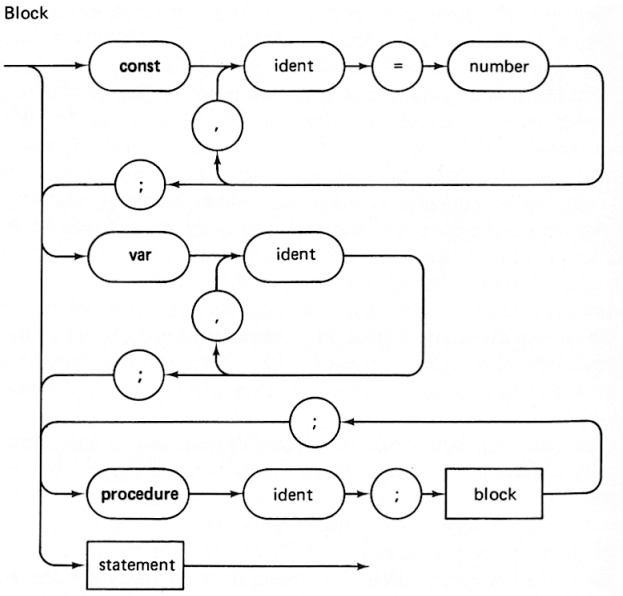
[1] Wirth, *Algorithms + Datastructures = Programs*



*Ett **expression** inleds med **+**, **-** eller en **term** och kan följas av **+** eller **-** och en term.*

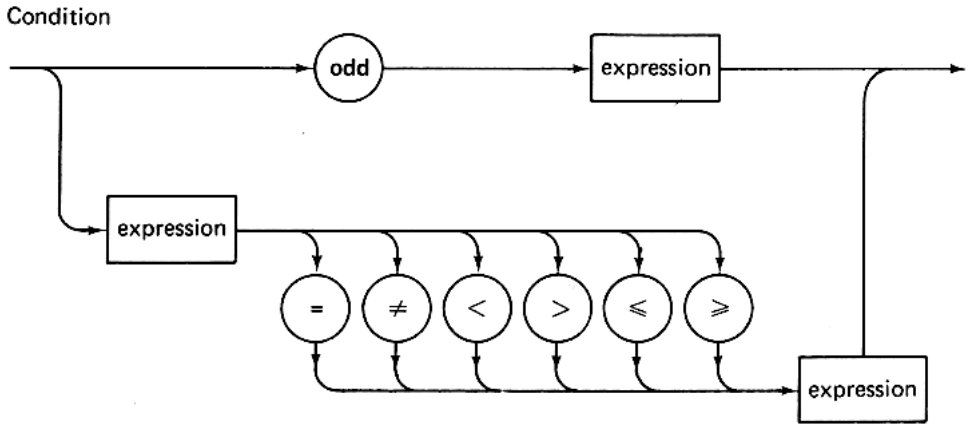


*Syntaxdiagram för en **term**.*

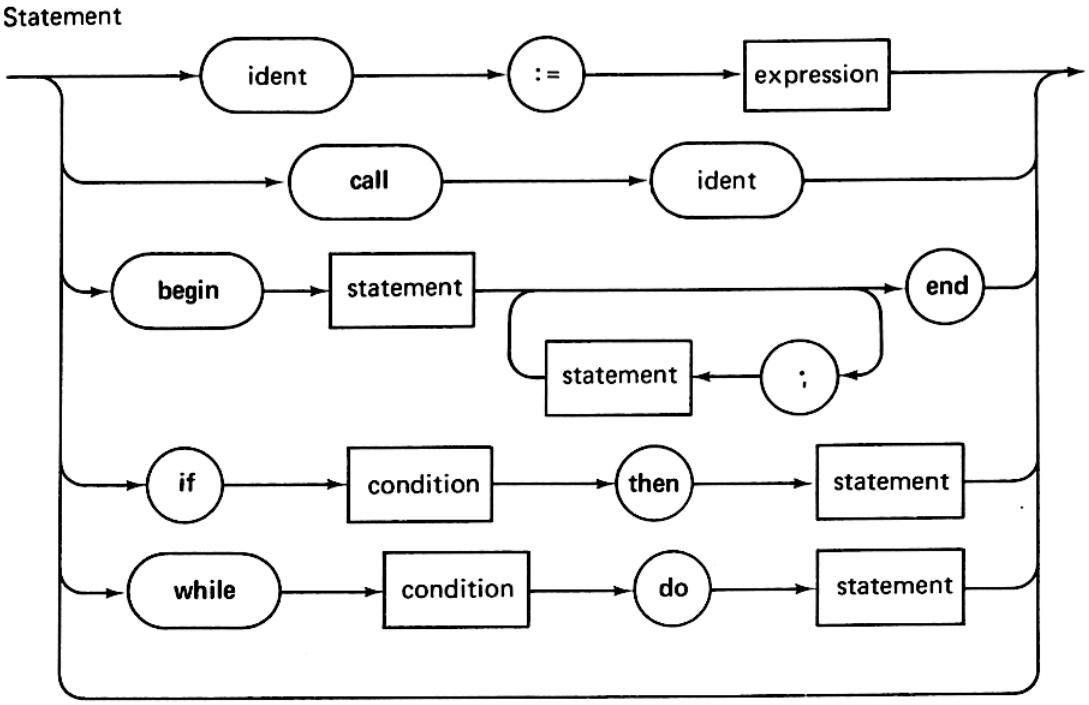


*Ett **block** börjar antingen med en konstantlista efter **const**, variabellista efter **var**, procedur **procedure** eller ett **statement**. Variabellistan utgörs av kommaseparerad lista av identifierare **ident** och raden avslutas med ett*

semikolon ;. Övriga språkelement konstrueras analogt.



Syntaxdiagram för ett ***condition****.*



Syntaxdiagram för ett ***statement****. Här känner vi igen språkelement tillhörande ett högnivåspråk.*

50. Hyfsa följande rutin.

```
RACKET1_UP:
    lds r16,RACKET1
    lsl r16
    cpi r16,192
    breq RACK1_LIM_UP
    sts RACKET1,r16
    ret
RACK1_LIM_UP:
    ori r16,224
    sts RACKET1,r16
    ret
```

51. Studera koden i föreläsningshäftet för att summera talen 1 till 255:

    Beräkna summan

    $$\sum_{1}^{255} = 1 + 2 + 3 + \ldots + 255$$

    och lagra resultatet i `$2D2`.

    a) Modifiera så den tar ett argument i `r16` istället för att ha hårdkodat värdet `255`.

    b) Använd `push` och `pop` för att minimera rutinens kringeffekter.

    c) Modifiera så att både argument och resultatadress anges på stacken, dvs anropet skall kunna vara

```
ldi     r16,HIGH($02D2) ; adr H
push    r16
ldi     r16,LOW($02D2)  ; adr L
push    16
ldi     r16,255         ; arg
push    r16
call    PARITET
pop     r0              ; dummy pop
pop     r0              ; dummy pop
pop     r0              ; dummy pop
```

52. Koden nedan kommer ur en projektdokumentation. Kommentarer är borttagna, här är bara programstrukturen viktig. Snygga till den!

```
TWI_INTERRUPT:
    push r22
    in  r22, SREG
    push r22
    in r22, TWSR
    andi r22, $F8
    cpi r22,$60
    breq SET_TWINT
    cpi r22,$80
    breq READ_DATA
    cpi r22,$A2
    breq SEND_DATA
    rjmp SET_TWINT
TWI_FINISHED:
    pop r22
    out SREG, r22
    pop r22
    reti
SET_TWINT:
    ldi r22, (1<<TWINT)|(1<<TWEN)|(1<<TWEA)|(1<<TWIE)
    out TWCR, r22
    rjmp TWI_FINISHED
READ_DATA:
    in r22, TWDR
    mov r17, r22
    andi r17, 0b00000111
    andi r22, 0b00011000
    lsr r22
    lsr r22
    lsr r22
    sts SELECTED_COLUMN, r17
    sts PLAYER, r22
    call ADD_DOT
    rjmp SET_TWINT
SEND_DATA:
    lds r22, TWI_DATA
    out TWDR, r22
    ldi r22, (1 << TWINT) | (1 << TWEN)
    out TWCR, r22
    rjmp SET_TWINT
```

53. Följande kod är ur en projektdokumentation. Skriv den bättre. Hur mycket mindre blir den?

```
// Name: movCOUNT (subroutine)
// Purpose: Store to SRAM
// Uses: r16, r17
movCOUNT:
   cpi   r17,1
   breq  STORE_ONE
   cpi   r17,2
   breq  STORE_TWO
   cpi   r17,3
   breq  STORE_THR
   cpi   r17,4
   breq  STORE_FOU
   cpi   r17,5
   breq  STORE_FIV
   cpi   r17,6
   breq  STORE_SIX
   cpi   r17,7
   breq  STORE_SEV
   cpi   r17,8
   breq  STORE_EIG
STORE_ONE:
   sts   MINNE,r16
   jmp   END_STORE
STORE_TWO:
   sts   MINNE+1,r16
   jmp   END_STORE
STORE_THR:
   sts   MINNE+2,r16
   jmp   END_STORE
STORE_FOU:
   sts   MINNE+3,r16
   jmp   END_STORE
STORE_FIV:
   sts   MINNE+4,r16
   jmp   END_STORE
STORE_SIX:
   sts   MINNE+5,r16
   jmp   END_STORE
STORE_SEV:
   sts   MINNE+6,r16
   jmp   END_STORE
STORE_EIG:
   sts   MINNE+7,r16
END_STORE:
   ret
```

```
// Name: readRAM (subroutine)
// Purpose: Get a byte from SRAM
// Uses: r17, r18
readRAM:
   cpi   r17,0
   breq  FIRST
   cpi   r17,1
   breq  SECOND
   cpi   r17,2
   breq  THIRD
   cpi   r17,3
   breq  FOURTH
   cpi   r17,4
   breq  FIFTH
   cpi   r17,5
   breq  SIXTH
   cpi   r17,6
   breq  SEVENTH
   cpi   r17,7
   breq  EIGHTH
FIRST:
   lds   r18,MINNE
   jmp   ENDZ
SECOND:
   lds   r18,MINNE+1
   jmp   ENDZ
THIRD:
   lds   r18,MINNE+2
   jmp   ENDZ
FOURTH:
   lds   r18,MINNE+3
   jmp   ENDZ
FIFTH:
   lds   r18,MINNE+4
   jmp   ENDZ
SIXTH:
   lds   r18,MINNE+5
   jmp   ENDZ
SEVENTH:
   lds   r18,MINNE+6
   jmp   ENDZ
EIGHTH:
   lds   r18,MINNE+7
ENDZ:
   ret
```

54. Koden på nästa sida är tagen ur en projektdokumentation. Den kan skrivas bättre. Gör det! Hur mycket mindre än nuvarande knappt 400 bytes kan du få koden?

    Subrutinen `FEL_KNAPP` är belägen någon annanstans. Macrot `TC_MACRO` är definierat som

```
   in    r16,TCNT0
   andi  r16,$07
```

```
LAMPOR_LYS:
   cpi   r16,$00
   breq  NOLL
   cpi   r16,$01
   breq  ETT
   cpi   r16,$02
   breq  TVA
   cpi   r16,$03
   breq  TRE
   cpi   r16,$04
   breq  FYRA
   cpi   r16,$05
   breq  FEM
   cpi   r16,$06
   breq  SEX
   cpi   r16,$07
   breq  SJU
NOLL:
   ldi   r16,$01
   out   PORTB,r16
   jmp   SIGNAL
ETT:
   ldi   r16,$02
   out   PORTB,r16
   jmp   SIGNAL
TVA:
   ldi   r16,$04
   out   PORTB,r16
   jmp   SIGNAL
TRE:
   ldi   r16,$08
   out   PORTB,r16
   jmp   SIGNAL
FYRA:
   ldi   r16,$10
   out   PORTB,r16
   jmp   SIGNAL
FEM:
   ldi   r16,$20
   out   PORTB,r16
   jmp   SIGNAL
SEX:
   ldi   r16,$40
   out   PORTB,r16
   jmp   SIGNAL
SJU:
   ldi   r16,$80
   out   PORTB,r16
   jmp   SIGNAL
SIGNAL:
   sbi   PORTD,1
KNAPP0:
   sbis  PINA,0
   rjmp  KNAPP1
   rjmp  CHECK0
KNAPP1:
   sbis  PINA,1
   rjmp  KNAPP2
   rjmp  CHECK1
KNAPP2:
   sbis  PINA,2
   rjmp  KNAPP3
   rjmp  CHECK2
KNAPP3:
   sbis  PINA,3
   rjmp  KNAPP4
   rjmp  CHECK3
KNAPP4:
   sbis  PINA,4
   rjmp  KNAPP5
   rjmp  CHECK4
KNAPP5:
   sbis  PINA,5
   rjmp  KNAPP6
   rjmp  CHECK5
KNAPP6:
   sbis  PINA,6
   rjmp  KNAPP7
   rjmp  CHECK6
KNAPP7:
   sbis  PINA,7
   rjmp  KNAPP0
   rjmp  CHECK7
CHECK0:
   sbic  PINA,0
   rjmp  CHECK0
   cpi   r16,$01
   brne  FEL
   andi  r16,$FE
   out   PORTB,r16
   call  LJUD
KNAPP_0:
   sbic  PINA,0
   jmp   KNAPP_0
   TC_MACRO
   jmp   LOOP
CHECK1:
   sbic  PINA,1
   rjmp  CHECK1
   cpi   r16,$02
   brne  FEL
   andi  r16,$FD
   out   PORTB,r16
   call  LJUD
KNAPP_1:
   sbic  PINA,1
   jmp   KNAPP_1
   TC_MACRO
   jmp   LOOP
CHECK2:
   sbic  PINA,2
   rjmp  CHECK2
   cpi   r16,$04
   brne  FEL
   andi  r16,$FB
   out   PORTB,r16
   call  LJUD
KNAPP_2:
   sbic  PINA,2
   jmp   KNAPP_2
   TC_MACRO
   jmp   LOOP
CHECK3:
   sbic  PINA,3
   rjmp  CHECK3
   cpi   r16,$08
   brne  FEL
   andi  r16,$F7
   out   PORTB,r16
   call  LJUD
KNAPP_3:
   sbic  PINA,3
   jmp   KNAPP_3
   TC_MACRO
   jmp   LOOP
FEL:
   jmp   FEL_KNAPP
CHECK4:
   sbic  PINA,4
   rjmp  CHECK4
   cpi   r16,$10
   brne  FEL
   andi  r16,$EF
   out   PORTB,r16
   call  LJUD
KNAPP_4:
   sbic  PINA,4
   jmp   KNAPP_4
   TC_MACRO
   jmp   LOOP
CHECK5:
   sbic  PINA,5
   rjmp  CHECK5
   cpi   r16,$20
   brne  FEL
   andi  r16,$DF
   out   PORTB,r16
   call  LJUD
KNAPP_5:
   sbic  PINA,5
   jmp   KNAPP_5
   TC_MACRO
   jmp   LOOP
CHECK6:
   sbic  PINA,6
   rjmp  CHECK6
   cpi   r16,$40
   brne  FEL
   andi  r16,$BF
   out   PORTB,r16
   call  LJUD
KNAPP_6:
   sbic  PINA,6
   jmp   KNAPP_6
   TC_MACRO
   jmp   LOOP
CHECK7:
   sbic  PINA,7
   rjmp  CHECK7
   cpi   r16,$80
   brne  FEL
   andi  r16,$7F
   out   PORTB,r16
   call  LJUD
KNAPP_7:
   sbic  PINA,7
   jmp   KNAPP_0
   TC_MACRO
   jmp   LOOP
LJUD:
   ; code for sound
   ret
LOOP:
   ; some more code
```

55. Även koden nedan kommer från en projektdokumentation. Hur bör den utföras? Vilken blir vinsten i kodstorlek?

```
GET_KEY:
   push  r18
   in    r17,PIND
   andi  r17,$0F
   mov   r18,r17
   clr   r17
   cpi   r18,$07
   breq  zero
   cpi   r18,$08
   breq  one
   cpi   r18,$09
   breq  two
   cpi   r18,$0C
   breq  three
   cpi   r18,$04
   breq  four
   cpi   r18,$05
   breq  five
   cpi   r18,$06
   breq  six
   cpi   r18,$0D
   breq  seven
   cpi   r18,$01
   breq  eight
   cpi   r18,$02
   breq  nine
   cpi   r18,$03
   breq  ten
   cpi   r18,$0E
   breq  eleven
   cpi   r18,$0A
   breq  twelve
   cpi   r18,$00
   breq  thirteen
   cpi   r18,$0B
   breq  fourteen
   cpi   r18,$0F
   breq  fifteen
zero:
   ldi   r17,$00
   rjmp  skiprest
one:
   ldi   r17,$01
   rjmp  skiprest
two:
   ldi   r17,$02
   rjmp  skiprest
three:
   ldi   r17,$03
   rjmp  skiprest
four:
   ldi   r17,$04
   rjmp  skiprest
five:
   ldi   r17,$05
   rjmp  skiprest
six:
   ldi   r17,$06
   rjmp  skiprest
seven:
   ldi   r17,$07
   rjmp  skiprest
eight:
   ldi   r17,$08
   rjmp  skiprest
nine:
   ldi   r17,$09
   rjmp  skiprest
ten:
   ldi   r17,$0A
   rjmp  skiprest
eleven:
   ldi   r17,$0B
   rjmp  skiprest
twelve:
   ldi   r17,$0C
   rjmp  skiprest
thirteen:
   ldi   r17,$0D
   rjmp  skiprest
fourteen:
   ldi   r17,$0E
   rjmp  skiprest
fifteen:
   ldi   r17,$0F
   rjmp  skiprest
skiprest:
   pop   r18
   ret
```

3. C-kod

56. Rutinen `s2h` översätter en ASCII-sträng till sin hexadecimala mostvarighet. Starta ett C-projekt i AtmelStudio och provkör.

```
void s2h(char *str, char *xstr)
{
    unsigned char tkn,l,i,k;

    l = strlen(str);
    for(i=0, k=0; i<l; i++)
    {
        tkn = str[i];
        if(tkn >= 0 && tkn <= 0x0f)
        {
            xstr[k] = 0x30;
            if(tkn >= 0 && tkn <= 9)
                xstr[k+1] = 0x30 + tkn;
            else
                xstr[k+1] = 0x37 + tkn;
        }
        else  if(tkn >= 0x10 && tkn <= 0x1f)
        {
            xstr[k] = 0x31; tkn -= 0x10;
            if(tkn >= 0 && tkn <= 9)
                xstr[k+1] = 0x30 + tkn;
            else
                xstr[k+1] = 0x37 + tkn;
        }
        else  if(tkn >= 0x20 && tkn <= 0x2f)
        {
            xstr[k] = 0x32; tkn -= 0x20;
            if(tkn >= 0 && tkn <= 9)
                xstr[k+1] = 0x30 + tkn;
            else
                xstr[k+1] = 0x37 + tkn;
        }
        else  if(tkn >= 0x30 && tkn <= 0x3f)
        {
            xstr[k] = 0x33; tkn -= 0x30;
            if(tkn >= 0 && tkn <= 9)
                xstr[k+1] = 0x30 + tkn;
            else
                xstr[k+1] = 0x37 + tkn;
        }
        else  if(tkn >= 0x40 && tkn <= 0x4f)
        {
            xstr[k] = 0x34; tkn -= 0x40;
            if(tkn >= 0 && tkn <= 9)
                xstr[k+1] = 0x30 + tkn;
            else
                xstr[k+1] = 0x37 + tkn;
        }
        else  if(tkn >= 0x50 && tkn <= 0x5f)
        {
            xstr[k] = 0x35; tkn -= 0x50;
            if(tkn >= 0 && tkn <= 9)
                xstr[k+1] = 0x30 + tkn;
            else
                xstr[k+1] = 0x37 + tkn;
        }
        else  if(tkn >= 0x60 && tkn <= 0x6f)
        {
            xstr[k] = 0x36; tkn -= 0x60;
            if(tkn >= 0 && tkn <= 9)
                xstr[k+1] = 0x30 + tkn;
            else
                xstr[k+1] = 0x37 + tkn;
        }
        else  if(tkn >= 0x70 && tkn <= 0x7f)
        {
            xstr[k] = 0x37; tkn -= 0x70;
            if(tkn >= 0 && tkn <= 9)
                xstr[k+1] = 0x30 + tkn;
            else
                xstr[k+1] = 0x37 + tkn;
        }
        k = k + 2;
    }
    xstr[k] = 0x00;
}
```

a) Undersök vad rutinen gör för något. Det kan vara bra att ha ASCII-tabellen nära. Förslag på testprogram är

```
int main(int argc, const char *argv[]) {
    char *str="AZaz.!9";
    char *hex="               ";

    s2h(str,hex);
    printf("Input: %s \t Output: %s \n", str, hex);
    return 0;
}
```

b) Skriv motsvarande rutin i assembler — som den rimligen skulle gjorts från början.

c) Med ledning av insikterna från assemblerrutinen: Skriv om den i bättre C.

d) Hur stora blev de olika lösningarna? Jämför med originalet.

e) Hur påverkar C-kompilatorns olika optimeringsflaggor (till exempel `-O1`, `-O2`, `-Os`)?

f) Studera hur anropet sker från C till assembler (via menyalternativet "Debug/Windows/Disassembly"). Anpassa din egen assemblerrutin och använd denna istället för C-kod.

g) Vilka begränsningar finns i originalrutinen? Kan dessa byggas bort i din version? Drag slutsatsen att man inte kan lita på "internet-kod" utan att undersöka den mycket noggrant själv!

57. Följande rutin översätter ett BCD-värde i `bcd` till dess representation för att visas på en sjusegmentsdisplay. Argumentet `cc` är 1 om dislayen är av typen *common cathode* och noll annars.

```
char bcdto7seg(char bcd, char cc)
{
    char retval;

    switch(bcd)
    {
      case 0: if(cc==1) retval = 0b00111111;
              else
              if(cc==0) retval = 0b11000000;
              break;
      case 1: if(cc==1) retval = 0b00000110;
              else
              if(cc==0) retval = 0b11111001;
              break;
      case 2: if(cc==1) retval = 0b01011011;
              else
              if(cc==0) retval = 0b10100100;
              break;
      case 3: if(cc==1) retval = 0b01001111;
              else
              if(cc==0) retval = 0b10110000;
              break;
      case 4: if(cc==1) retval = 0b01100110;
              else
              if(cc==0) retval = 0b10011001;
              break;
      case 5: if(cc==1) retval = 0b01101101;
              else
              if(cc==0) retval = 0b10010010;
              break;
      case 6: if(cc==1) retval = 0b01111101;
              else
              if(cc==0) retval = 0b10000010;
              break;
      case 7: if(cc==1) retval = 0b00000111;
              else
              if(cc==0) retval = 0b11111000;
              break;
      case 8: if(cc==1) retval = 0b01111111;
              else
              if(cc==0) retval = 0b10000000;
              break;
      case 9: if(cc==1) retval = 0b01101111;
              else
              if(cc==0) retval = 0b10010000;
    }

    return retval;
}

.
```

Din uppgift är att snygga upp koden och minimera programminnesåtgången. Lämpliga steg *kan* vara att studera koden och sedan

a) identifiera och eliminera onödig kod

b) beskriva i ord vad rutinen *egentligen* gör

c) skriva en bättre (här: kortare) kod i C

d) skriva en bättre kod i assembler.

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

```
for(index = START; index != END; index++){
...
}
```

59. Uttryck `for`-loopen

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

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

60. 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;
}
```

4. Programmeringsuppgifter

Här följer några lite större uppgifter. Tänk på[1]:

> You don't understand a problem
> until you can simplify it.
>
> — L. Brodie

61. Beräkna summan 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, som skall ingå i summan, är angivet med omvänt tecken. Lagra summan i `r17:r16`, och lagra antalet tal i `r18`.

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

62. Tillverka en *odometer*, en BCD-räknare med fyra siffror som räknar från `0000` till `9999`. Simulera din kod i Atmelstudio. Använd registren `r20`–`r23` för de olika siffrorna.

    Variant:

    1) Kläm ihop två BCD-siffror i en byte för att spara minnesplats (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.)

63. Skriv ett assemblerprogram som avgör vilka operationer $op \in \{+, -, \star\}$ som löser ekvationen

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

    Där talen A–D är givna BCD-kodade siffror och N valfritt, till exempel $A = B = C = D = 4$ och $N = 20$.

[1] Ur *Leo Brodie, Thinking Forth*

Utvärderingen ska ske genom uttömmande sökning[2] 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.

Använd instruktionen `mul` för multiplikation.

Programmet måste således uppräkna alla operationer:

`+++`, `++-`, `++*`, `+-+`, `+-`, `+-*`, osv

1) Rita strukturdiagram och pseudokod.
2) Programmera och simulera i assembler
3) Fundera på hur du skulle avgöra om ekvationen alltid har en lösning. Ändras din metod om enbart siffror ≤ 5 får användas?

64. Skriv den subrutin som korrekt hämtar sin argumentbyte från stacken om den anropande funktionen placerat argumentet där innan subrutinhoppet. Rita karta över stackinnehållet.

65. Man kan använda pekarna `X`, `Y` och `Z` i assemblerarkitekturen. Men alla kan inte peka på allt. Vilka begränsningar finns?

[2] Så kallad *brute forcing*

5. Koppling till 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 AtmelStudio 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?

A. Lösningsförslag

Förslagen är just förslag, det finns ofta fler än ett sätt att koda en lösning. Ibland anges flera förslag.

```
1:
   ldi   r16,198
   ldi   r17,$64
   ldi   r18,0b10010011

2:
   mov   r18,r16

3:
   lds   r16,$110
   sts   $112,r16

4:
   lds   r16,$110
   lds   r17,$111
   add   r16,r17
   sts   $112,r16

5:
   lds   r16,$110
   lsl   r16
   sts   $111,r16

6:
   andi  r16,$0F
     eller
   andi  r16,0b00001111
     eller
   ldi   r17,$0F
   and   r16,r17

7: ori   r16,$E0
     eller
   ori   r16,0b11100000
     eller
   ldi   r17,$E0
   or    r16,r17

8:
   lds   r16,$110
   mov   r17,r16
   andi  r16,$F0
   lsr   r16
   lsr   r16
   lsr   r16
   sts   $111,r16
   andi  r17,$0F
   sts   $112,r17
     eller
   lds   r16,$110
   mov   r17,r16
   swap  r16
   andi  r16,$0F
   sts   $111,r16
   andi  r17,$0F
   sts   $112,r17

9:
   lds   r16,$110
   lds   r17,$111
   mov   r18,r17
   cp    r16,r17
   brlo  SKIP
   mov   r18,r16
SKIP:
   sts   $112,r18

10:
   lds   r16,$110
   ldi   r17,10
   mul   r16,r17
   sts   $112,r0
     eller
   lds   r16,$110
   lsl   r16
   mov   r17,r16
   lsl   r16
   lsl   r16
   add   r16,r17
   sts   $112,r16

11:
   subi  r16,-7
     eller
   inc   r16
   inc   r16
   inc   r16
   inc   r16
   inc   r16
   inc   r16
   inc   r16
     eller
   ldi   r17,7
```

```
   add   r16,r17

12:
   adiw  r30,3

13:
   movw  r30,r16
   adiw  r30,3
   movw  r16,r30
     eller
   subi  r16,-3
   brcc  DONE
   inc      r17
DONE:

14a:
   ldi   ZH,HIGH($110)
   ldi   ZL,LOW($110)
   ld    r16,Z
   ldi   ZH,HIGH($111)
   ldi   ZL,LOW($111)
   st    Z,r16
     eller
   ldi   ZH,HIGH($110)
   ldi   ZL,LOW($110)
   ld    r16,Z+
   st    Z,r16
     eller
   ldi   YH,HIGH($110)
   ldi   YL,LOW($110)
   ld    r16,Y
   std   Y+1,r16

14b:
   ldi   YH,HIGH($110)
   ldi   YL,LOW($110)
   ld    r16,Y
   ldd   r17,Y+1
   add   r16,r17
   std   Y+2,r16

14c:
   ldi   YH,HIGH($110)
   ldi   YL,LOW($110)
   ldi   XH,HIGH($120)
   ldi   XL,LOW($120)
   ldi   r18,5
LOOP:
   ld    r16,Y+
   ld    r17,Y+
   add   r16,r17
   st    X+,r16
   dec   r18
   brne  LOOP
```

```
14d:
   ldi   YH,HIGH(16)
   ldi   YL,LOW(16)
   ldi   r16,15
LOOP:
   st    -Y,r16
   dec   r16
   brne  LOOP

15:
   a) r16 = $61
   b) r20 = $19
   c) r21 = 0
   d) r21 = $F6
   e) adress $100 = $F5
   f) r20 = $F5
   g) r17 = $19, X = $100
   h) r17 = $19, X = $101
   i) r1:r0 = $FBD5

16:
   a) andi r16,$00

   b) ;76543210=11110000=$F0
      ori   r16,$F0
   c)
      ;76543210=11111000=$F0
      andi  r16,$F8
   d) or    r16,r17
      com   r16
      mov   r18,r16
   e) ;76543210=01111100=$7C
      ldi   r17,$7C
      eor   r16,r17
   f) andi  r16,$0F
      andi  r17,$F0
      or    r17,r16

17: .db $4D,$49,$4B,$52,$4F,
       $64,$61,$74,$6F,$72
      eller
   .db "MIKROdator"
     Avsluta med $00: ...$6F,$72,$00
        eller
     length byte innan: $0A,$4D,$49,...

18:  swap r16

19:
BCD_LOWER_NIBBLE:
   andi  r16,$F0
BCD_LOOP:
   inc   r16
   cpi   r16,10   ; tio
   brne  BCD_LOOP
   andi  r16,$F0
   ; here is 0..9
   ; in lower nibble
   jmp   BCD_LOOP
```

```
BCD_UPPER_NIBBLE:
   andi  r16,$0F
BCD_LOOP:
   subi  r16,-$10
   cpi   r16,$A0     ; $A << 4
   brne  BCD_LOOP
   andi  r16,$0F
   ; here is 0..9 in
   ; upper nibble
   jmp   BCD_LOOP


20:
LOOKUP:
   ldi      ZH,HIGH(LIST_START<<1)
   ldi      ZL,LOW(LIST_START<<1)
   add      ZL,r16
   clr      r17
   adc      ZH,r17
   lpm      r16,Z
   ret

LOOKUP1:
   call     BRACKET
   brne     NOT_IN_RANGE
   call     LOOKUP
NOT_IN_RANGE:
   ret

BRACKET:
   cpi     r16,0
   brcs    BRACKET_END
   cpi     r16,63
   brcc    BRACKET_END
   sez
BRACKET_END:
   ret

LOOKUP2:
   push     r17
   ldi      ZH,HIGH(LIST_START<<1)
   ldi      ZL,LOW(LIST_START<<1)
   add      ZL,r16
   clr      r17
   adc      ZH,r17
   lpm      r16,Z
   pop      r17
   ret

21:
X2:
   andi  r16,$0F
   ldi      ZH,HIGH(X2_TAB<<1)
   ldi      ZL,LOW(X2_TAB<<1)
   add      ZL,r16
   lpm      r16,Z
   ret
   .org $XX00
X2_TAB:
   .db   0,1,4,9,16,...,225

22:
   andi     r16,$0F
   muls     r16,r16
   mov      r16,r0
   ret

23:
   ori      r16,$30
   ret

24:
   lds      r16,$101
   dec      r16
   breq     ETT
   dec      r16
   breq     TVA
   dec      r16
   breq     TRE
   ; >4

25:
ANROP:
   ldi      r16,HIGH(ADRESS)
   push     r16
   ldi      r16,LOW(ADRESS)
   push     r16
   call     RUTIN24
   pop      r16 ; b) ta bort argument
   pop      r16 ; b) ta bort argument
   ret      ; b) annars katastrof!

RUTIN: a)
   mov      ZH,SPH
   mov      ZL,SPL
   ldd      r24,Z+3
   ldd      r25,Z+4
   ret

26:
   Metodtips (i python):
   if(r16 % 400)==0 return True
   if(r16 % 100)==0 return False
   if(r16 % 4) == 0 return True
   Med upprepad subtraktion som '%'

27:
   a) 101  b) 1111  c) 11001
   d) 1111101000  e) 11111111111
   f) 1000001 g) 10001

28:
   a) 101=$5 b) 1111=$F c) 11001=$19
   d) 1111101000=$3E8
   e) 11111111111=$7FF f) 1000001=$41
   g) 10001=$11
```

29:
a) 1111 b) 1100111 c) 11111110
d) 101100 e) 1101010
f) 00001010 g) 1010101111001101

30:

31:
a) 11111011 b) 00011001 c) 11100111
d) 00011111 e) 10011100
f) 11111111 g) 10111111

32:
10

33:
Tips X-Y=X+(-Y) osv

34:
a)

0	1	2	3	4	5	6	7	8	9	
+0	+1	+2	+3	+4	-5	-4	-3	-2	-1	<- 10's compl

b)
-50 --> +49

35:
Basen 10+26= 36.
XYZ = 44027

36:
208 resp -48 (=208-256)

37:
A7-A4 ut, A3-A0 in
B7-B4 in, B3-B0 ut

```
in  r16,PINA        in  r16,PINB
out PORTA,r16       out PORTB,r16
```

38:
```
PORT_CONFIG:
   ldi   r16,$55
   out   DDRA,r16
   ldi   r16,$AA  ; eller 'lsl r16'
   out   DDRB,r16
```

39:

40:
Se kurslitteraturen.

41:
a) MSB kopieras eller inte
b) Ingen

42:
Instruktion = Op-kod [+ operand[er]]
2 operander: mov r16,r18
1 operand: com r16
0 operand: nop

43:
Se kurslitteraturen.
Relativ - kortare, snabbare

44:
Att pinnarna blir svaga 'ett'-or. (5V)

45:
a) med call eller rcall
b) med ret
c) jmp konsumerar inte returadressen
d) javisst
e) javisst
f) ja, stackens maximala storlek
g) rutinen pushar/poppar sina egna interna register

46:
Extern signal, intern timer. Dessutom EEPROM-skrivning, serie-kommunikation m fl

47:
reti aktiverar avbrott igen

48:
ADCCaddr. (ATmega16A $1C, 328p $15)

49:
```
push  r16
in    r16,SREG
push  r16
push  interna reg
:
pop   interna reg
pop   r16
out   SREG,r16
pop   r16
reti
```

49:

Rutinen skall enbart ha en `ret`. Instruktionen `sts` förekommer på två ställen, bara ett behövs. Sammantaget blir den hyfsade rutinen alltså:

```
RACKET1_UP:
   lds   r16,RACKET1
   lsl   r16
   cpi   r16,$C0
   brne  EXIT
   ori   r16,$E0
EXIT:
   sts   RACKET1,r16
   ret

   eller utan lsl

RACKET1_UP:
   lds   r16,RACKET1
   cpi   r16,$60
   brne  RACK1_LIM_UP
   ori   r16,$70
RACK1_LIM_UP:
   sts   RACKET1,r16
   ret
```

51:

```
   simulera
```

.

52: Notera att, som koden är skriven, *kan* r22 bara innehålla $60, $80 eller $A2. Uteslutning gör att man i så fall inte behöver jämföra alla tre! Det här verkar suspekt, men så är koden skriven. Enligt JSP skall ett *default*-alternativ alltid finnas.

```
TWI_INTERRUPT:
   push  r22
   in    r22,SREG
   push  r22

   in    r22,TWSR
   andi  r22,$F8



   cpi   r22,$60
   breq  SET_TWINT

   cpi   r22,$80
   brne  SEND_DATA

READ_DATA:
   in    r22,TWDR
   mov   r17,r22
   andi  r17,$07
   andi  r22,$18
   lsr   r22
   lsr   r22
   lsr   r22
   sts   SELECTED_COLUMN,r17
   sts   PLAYER,r22
   call  ADD_DOT
   rjmp  SET_TWINT


SEND_DATA:
   lds   r22,TWI_DATA
   out   TWDR,r22
   ldi   r22,(1<<TWINT)|
             (1<<TWEN)
   out   TWCR,r22

SET_TWINT:
   ldi   r22,(1<<TWINT)|
             (1<<TWEN) |
             (1<<TWEA) |
             (1<<TWIE)
   out   TWCR,r22

TWI_FINISHED:
   pop   r22
   out   SREG,r22
   pop   r22
   reti
```

Notera att den är en avbrottsrutin: Spara statusregistret och skapa ”lokala variabler”.

Läs av TWSR, maska, och betrakta innehållet $60, $80 eller $A2 om det inte var de två förra. Brancha ut till respektive kodstycke:

Om $60: Hoppa

Om **inte** $80 (dvs $A2), hoppa till SEND_DATA i annat fall fortsätt i READ_DATA:

:

Binära konstanter ersatta med hexadecimala.

:

:

:

:

:

Hoppa ur via SET_TWINT.

:

Gör SEND_DATA.

:

:

:

:

Gemensam *exit* genom SET_TWINT.

Återställ innan återhopp från avbrottet. Notera en enda *exit*-punkt. Klar!

Det kan underlätta att rita programflödet med linjer och pilar för att avslöja en alternativ och effektivare programstruktur.

Med koden i detta skick kan det nu vara värt att studera själva funktionen och intruktionerna igen. Det är inte ovanligt att optimeringsmöjligheter öppnat sig när man ser koden med nya ögon efter omstrukturering.

`TWI_INTERRUPT`: Omordnad och optimerad

```
TWI_INTERRUPT:
   push  r22
   in    r22,SREG
   push  r22
   in    r22,TWSR
   andi  r22,$F8
   cpi   r22,$80
   breq  READ_DATA
   cpi   r22,$A2
   breq  SEND_DATA
   rjmp  SET_TWINT
READ_DATA:
   in    r22,TWDR
   mov   r17,r22
   andi  r17,$07
   andi  r22,$18
   lsr   r22
   lsr   r22
   lsr   r22
   sts   SELECTED_COLUMN,r17
   sts   PLAYER,r22
   rcall ADD_DOT
   rjmp  SET_TWINT
SEND_DATA:
   lds   r22,TWI_DATA
   out   TWDR,r22
   ldi   r22,(1<<TWINT)|(1<<TWEN)
   out   TWCR,r22
SET_TWINT:
   ldi   r22,(1<<TWINT)|...|(1<<TWIE)
   out   TWCR,r22
TWI_FINISHED:
   pop   r22
   out   SREG, r22
   pop   r22
   reti
```

`53-55`:

Skriv om i editor

`63`: En möjlig JSP-struktur är: