

Datorteknik TSEA82 + TSEA57

Fö2

Instruktioner

Datorteknik Fö1 : Agenda

- Repetition
- Instruktioner
- Labb0
- Övningsuppgifter
- Tid för frågor

Repetition

Instruktioner

I databladet för mikrokontrollern finns drygt hundralet instruktioner. Dessa kan delas in i fem huvudgrupper:

- Grupp 1. Instruktioner som flyttar data (`ldi`, `mov`, ...)
- Grupp 2. Aritmetiska instruktioner (`add`, `sub`, `subi`, ...)
- Grupp 3. Logiska instruktioner (`asl`, `ror`, ...)
- Grupp 4. Hoppinstruktioner (`jmp`, `brxx`, `call`, ...)
- Grupp 5. I/O-instruktioner (`out`, `in`)

Instruktioner : Grupp 1. Flytta data¹

Till, och mellan, dataregister (generella register) flyttar man data med `ldi` och `mov`.

Exempel: `ldi`

Flytta konstanten 23 till `r16`, "ladda `r16` med 23".

The diagram shows the assembly instruction `ldi r16, 23 ; Load immediate` with labels and arrows pointing to its components:

- `ldi` is labeled "instruktion" with an arrow pointing up.
- `r16` is labeled "destinationsregister" with an arrow pointing up.
- `23` is labeled "konstant" with an arrow pointing up.
- `; Load immediate` is labeled "kommentar" with an arrow pointing up.

Instruktioner : Grupp 1. Flytta data

16-bitars register, kan inneha tal mellan 0-65535

För att peka ut en adress i arbetsminnet (\$0060-\$045F) krävs mer än 8 bitar, då kan man använda ett 16-bitars pekarregister, X, Y eller Z, som motsvaras av registerparen r27:r26, r29:r28 respektive r31:r30.

Exempel: Invertera talet som finns på adress \$0102

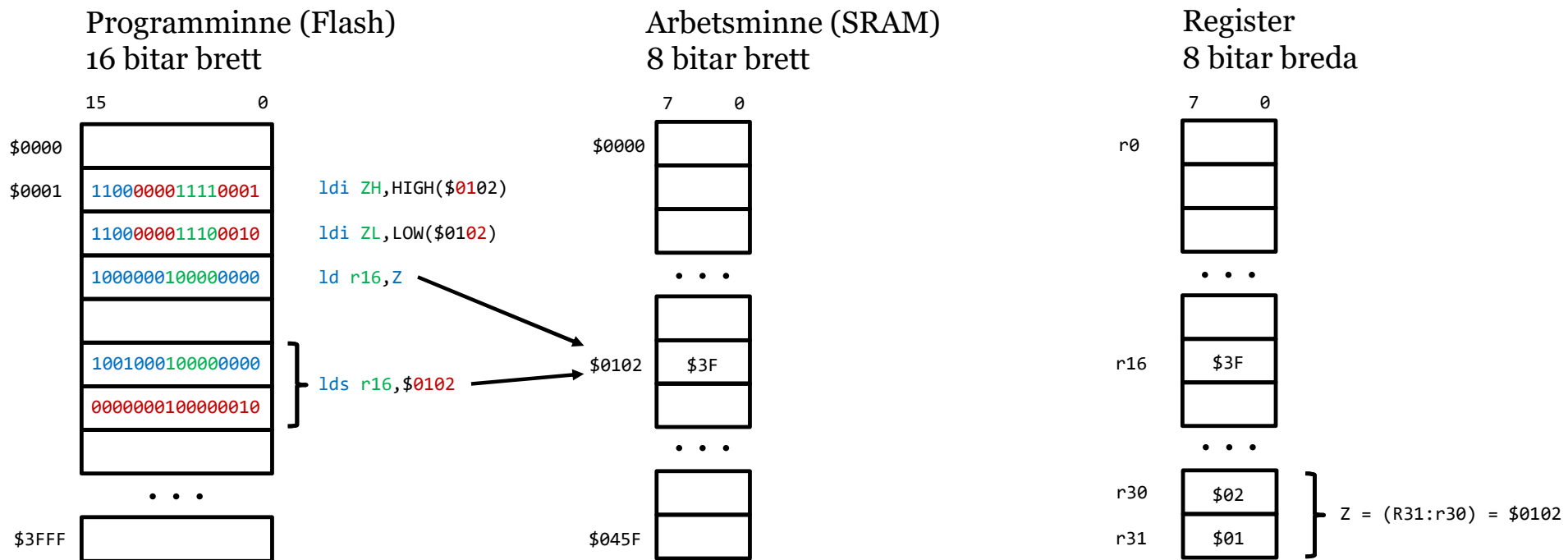
Med pekare:

```
ldi    ZH,HIGH($0102) ; ZH(r31)=01
ldi    ZL,LOW($0102)  ; ZL(r30)=02
ld     r16,Z           ; r16=Mem(Z)
com    r16             ; complement r16
st     Z,r16           ; Mem(Z)=r16
```

Utan pekare:

```
lds    r16,$0102      ; r16=Mem($0102)
com    r16             ; complement r16
sts    $0102,r16      ; Mem($0102)=r16
```

Instruktioner : Grupp 1. Flytta data

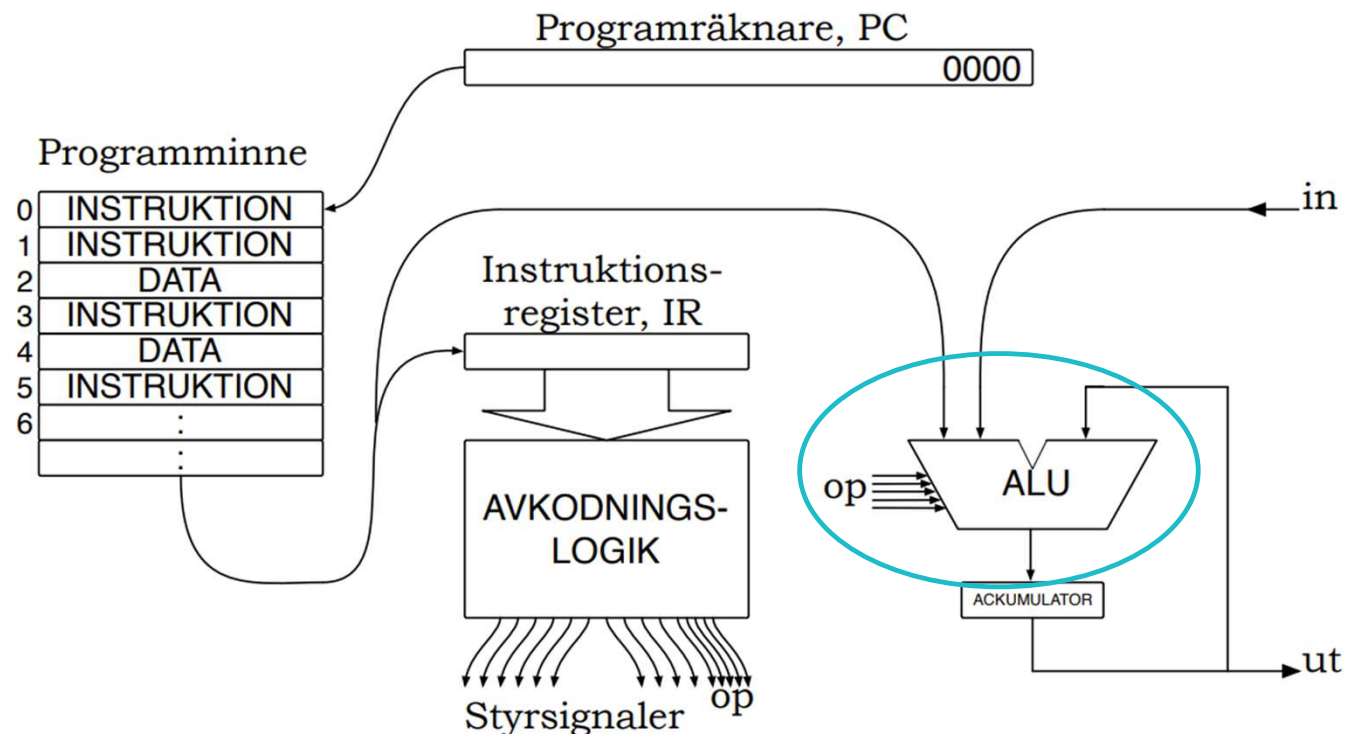


Instruktioner, resten ...

Instruktioner : Grupp 2. Aritmetiska operationer

Aritmetiska instruktioner använder ALU:n via olika operationer såsom t ex addition eller subtraktion.

Resultatet av operationen sparas i något register och status (positivt, negativt ...) av operationen sparas i statusregistret.



Instruktioner : Grupp 2. Aritmetiska operationer

Aritmetiska instruktioner är instruktioner som gör någon form av beräkning, såsom add (addition), sub (subtraktion), mul (multiplikation) eller fmul (fixpunktsmultiplikation).

Dessa (add, sub, mul, fmul) förekommer i lite olika varianter, beroende på hur man vill utföra själva operationen, t ex mellan register, med en konstant eller med eller utan tecken (mul, fmul).


Dock har inte ALU:n i en enkel processor (som ATmega16) stöd för division. Det kostar helt enkelt för mycket (i form av chip-yta) att bygga en divisionsinstruktion.

Instruktioner : Grupp 2. Aritmetiska operationer

Exempel: add, adc

Addera de två 16-bitarstalen som finns i r21:r20 och r17:r16 till r17:r16.

add r16,r20 ; ingen ingående carry
 adc r17,r21 ; med carry (om den finns)


11 11 1

r21:r20	00000000:11101000	=232 (0x00E8)
r17:r16	00000001:00101100	=300 (0x012C)

r17:r16	00000010:00010100	=532 (0x0214)

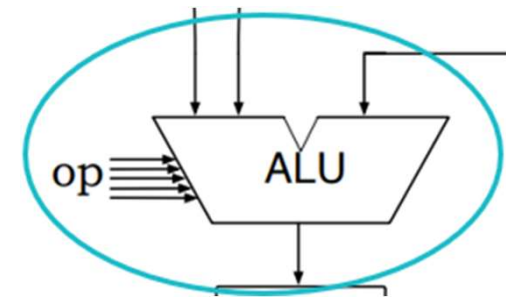
Instruktioner : Grupp 3a. Logiska operationer

Exempel: `andi`

Använd `andi` för att maska fram de tre minst signifikanta bitarna ur byten på adress `$0120`:

```
lds    r16,$0120    ; r16=Mem($0120)
andi   r16,$07       ; $07=0000 0111
```

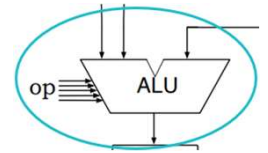
```
r16    10110101
$07    00000111    andi
-----
r16    00000101
```



Instruktioner : Grupp 3b. Skiftinstruktioner

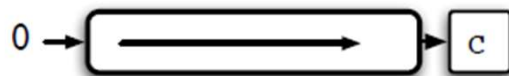
Exempel: lsr, asr

lsr och asr skiftar ett steg åt höger, logiskt respektive aritmetiskt



```
ldi r16, 0b10100101
```

```
lsr r16
```



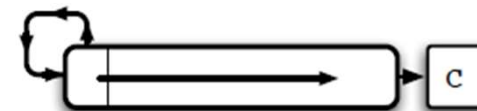
```
C:r16  0:10100101
```

```
lsr
```

```
C:r16  1:01010010
```

```
ldi r16, 0b10100101
```

```
asr r16
```



```
C:r16  0:10100101
```

```
asr
```

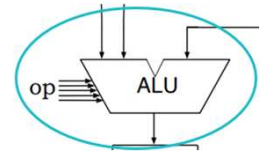
```
C:r16  1:11010010
```

Högerskift motsvarar division med 2

Instruktioner : Grupp 3b. Skiftinstruktioner

Exempel: `lsl, (asl)`

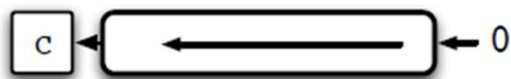
`lsl` och `asl` skiftar ett steg åt vänster, logiskt respektive aritmetiskt



`ldi r16, 0b10100101`

`lsl r16`

`asl` finns inte som instruktion, då den fungerar precis som `lsl`, dvs använd `lsl`



C:r16 0:**1**0100101 `lsl`

C:r16 **1**:0100101**0**

Vänsterskift motsvarar multiplikation med 2

Instruktioner : Grupp 4. Hoppinstruktioner

Hoppinstruktioner används för att styra programflödet, dvs för att ta sig till en annan del av programmet. Det finns två sorters hopp:

- Villkorliga hopp
- Ovillkorliga hopp

Villkorliga hopp utförs om ett visst villkor är uppfyllt. Villkoret baseras på en (eller flera) av statusregistrets flaggor (... , Z, N, C, V).

Ovillkorliga hopp utförs alltid, dvs de är oberoende av några villkor.

Utöver detta finns *anrop* av subrutiner, vilket inte ska förväxlas med hopp.

Vid *anrop* återkommer programmet till instruktionen efter anropet, vilket inte sker med hopp.

Instruktioner : Grupp 4. Hoppinstruktioner

Exempel: Ovillkorligt hopp, `jmp`

```
A:           ; A är en symbolisk adress (label)
  instr
  instr
  jmp      B
  ...

B:           ; B är en symbolisk adress (label)
  instr      ; hit gick hoppet
  ...
```


Instruktioner : Grupp 4. Hoppinstruktioner

Ovillkorliga hopp kan vara absoluta eller relativa.

Exempel: Hoppa från \$1000 till \$1200 dels med jmp, dels med rjmp:

Adress	Absolut hopp	Relativt hopp
\$1000 A:	jmp B ; 'jmp \$1200'	rjmp B ; 'jmp +\$200'

\$1200 B:		

Med absoluta hopp, jmp, når man hela programminnet (men instruktionen kräver då dubbel lagringsplats, 32 bitar, i programminnet).

Med relativa hopp, rjmp, kan man bara hoppa en viss längd från där man befinner sig, -2048 ... +2047 steg (instruktionen behöver då bara 16 bitar i programminnet).

Instruktioner : Grupp 4. Hoppinstruktioner

Exempel: Villkorligt hopp, brmi

I adresserna \$101 och \$102 finns två *teckenlösa tal*, placera det största talet i adress \$103

```
lds    r16,$101    ; r16=Mem($101)
lds    r17,$102    ; r17=Mem($102)
cp      r16,r17     ; (r16-r17), update flags (Z,N,C,V)
brmi   R17BIG      ; branch on minus (N==1), r17 biggest
mov     r17,r16     ; r16 was biggest, so r17=r16
```

R17BIG:

```
sts     $103,r17    ; Mem($103)=r17
```

Instruktioner : Grupp 4. Hoppinstruktioner

Det finns ett antal olika branch-instruktioner för olika villkor, t ex:

brne hopp om $Z=0$, dvs om resultat $\neq 0$
breq hopp om $Z=1$, dvs om resultat $= 0$
brpl hopp om $N=0$, dvs om resultat ≥ 0
brmi hopp om $N=1$, dvs om resultat < 0
brcc hopp om $C=0$, dvs om carry inte har inträffat
brcs hopp om $C=1$, dvs om carry har inträffat
...

Instruktioner : Grupp 4. Hoppinstruktioner

Exempel: brcs, brcc

Skriv kod som sätter Z om ett tal i r16 är i intervallet [1..8]
Teckenlösa tal förutsätts

Processorn vet inte om ett tal är med eller utan tecken. Det bestämmer programmeraren.

BRACKET:

```
    cpi    r16,1          ; r16 - 1
    brcs   BRACKET_DONE  ; C=1 Z=0 om r16 < 1
                                ; C=0 Z=1 om r16 = 1
                                ; C=0 Z=0 om r16 > 1
```

; was at least 1

```
    cpi    r16,9          ; r16 - 9
    brcc   BRACKET_DONE  ; C=1 Z=0 om r16 < 9
                                ; C=0 Z=1 om r16 = 9
                                ; C=0 Z=0 om r16 > 9
```

; and at most 8

```
    sez                                ; set Z flag
```

BRACKET_DONE:

```
    ret
```

Instruktioner : Grupp 4. Hoppinstruktioner

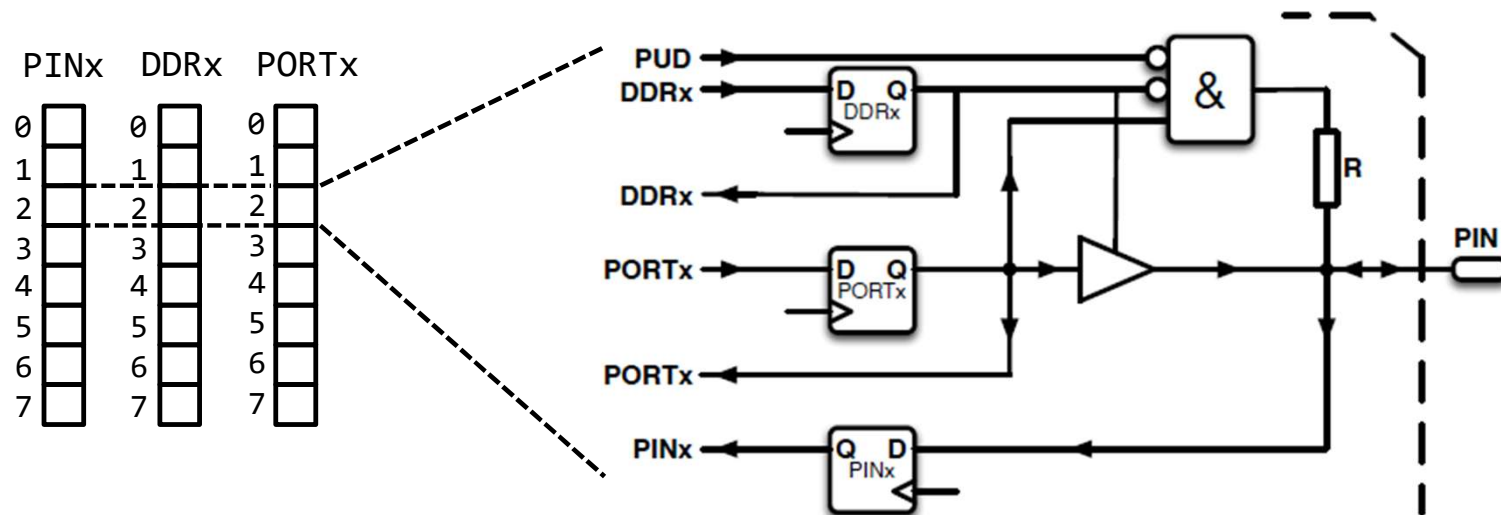
Exempel: brcc, brne

Beräkna $\sum_{i=1}^{255} i = 1 + 2 + \dots + 255$
och lagra resultatet i \$02D2

Summan kommer inte rymmas
inom ett register, därför används
r22:r21 för att lagra summan.

```
clr    r22        ; sum = 0
clr    r21
ldi    r20,255    ; index
AGAIN:
add     r21,r20    ; sum += index
brcc    NOCARRY
inc     r22
NOCARRY:
dec     r20        ; index--
brne    AGAIN      ; index≠0?
sts     $02D2,r21   ; store sum
sts     $02D3,r22
```

I/O-portarna i processorn är 8 bitar breda. Varje enskild bit i en port kan individuellt fungera som ingång eller utgång. Det styrs via ett datarikttningsregister DDRx. Utvärdet på en port bestäms av portregistret PORTx, och invärdet läses från det sk pinregistret PINx (x=A, B, C el. D).



Instruktioner : Grupp 5. I/O-instruktioner

Exempel: Konfigurera PORTB så att bit 0-3 är ingångar och bit 4-7 utgångar.

Läs sedan portens fyra låga bitar och skriv ut dem på porten fyra höga bitar.

```
ldi    r16,$F0    ; $F0=11110000
out    DDRB,r16   ;      ooooiiii (o=out,i=in)
in     r16,PINB   ; read from PINB
andi   r16,$0F    ; mask bits
lsl    r16        ; shift left 1 step
lsl    r16        ; shift left 1 step
lsl    r16        ; shift left 1 step
lsl    r16        ; shift left 1 step
out    PORTB,r16  ; write to PORTB
```

Anm1. Använd instruktionen swap istället för 4 st lsl

Anm2. Observera att man skriver till PORT och läser från PIN

Anm3. Behövs egentligen andi r16,\$0F ?

Instruktioner : Grupp 5. I/O-instruktioner

Instruktionen out skriver alltså till samtliga 8 bitar i ett I/O-register, t ex en port. Man kan också manipulera enstaka bitar i ett I/O-register utan att påverka övriga bitar, med instruktionerna sbi och cbi:

out PORTB, r16 ;

7	6	5	4	3	2	1	0
0	1	1	1	1	0	1	0

 r16=\$7A

└───┬───┘└───┬───┘
7A

sbi PORTB, 2 ;

7	6	5	4	3	2	1	0
0	1	1	1	1	1	1	0

└───┬───┘└───┬───┘
7E

cbi PORTB, 5 ;

7	6	5	4	3	2	1	0
0	1	0	1	1	0	1	0

└───┬───┘└───┬───┘
6E

Instruktioner : Grupp 5. I/O-instruktioner

Man kan jämföra hela innehållet i ett generellt register (r0-r31) med ett värde med compare-instruktionen `cpi`, det går *inte* att göra med ett I/O-register utan att först kopiera I/O-registret till ett generellt register.

Däremot kan man testa (och hoppa) enskilda bitar i ett register (generellt eller I/O-register) med skip-instruktionen:

```
sbrc  rX,b  ; hoppa om bit b i register rX nollställd
sbrs  rX,b  ; hoppa om bit b i register rX ettställd
sbic  A,b   ; hoppa om bit b i I/O-register A nollställd
sbis  A,b   ; hoppa om bit b i I/O-register A ettställd
```

Instruktioner : Grupp 5. I/O-instruktioner

Exempel: En tryckknapp är ansluten till port B, bit 2. Använd `sbic` för att ge ett sant ($\neq 0$) värde i `r16` om knappen är nedtryckt, annats falskt ($=0$).

```
    ; --- GET_KEY. Bit 2 = 1 if key pressed
GET_KEY:
    clr    r16        ; r16=0
    sbic   PINB,2     ; skip next instruction if not pressed
    ser    r16        ; r16=$FF
    ...
```

Labb0 ...

Övningsuppgifter ...

Tid för Frågor

Anders Nilsson

www.liu.se