Datorteknik TSEA82 + TSEA57
Fö1

Introduktion till kursen

LINKÖPINGS UNIVERSITET

Datorteknik Fö1 : Agenda

- Informationskanaler
- Introduktion till kursen Datorteknik
- Fö1 : Datormodellen, Programmerarmodellen, Instruktioner
- Nödvändiga programvaror
- Tid för frågor

Informationskanaler

Informationskanaler

- Lisam : Används endast för labbanmälan
- Microsoft Teams : Används till hjälp med laborationer utanför labbtid
- Mail + Teams : Används för övrig kommunikation
- Websidan : Används för kursmateriel och dylik information
  -http://www.isy.liu.se/edu/kurs/TSEA82/

Introduktion till kursen Datorteknik

Vad är datorteknik, i den här kursen?

Assemblerinstruktioner

Subrutiner

Binär aritmetik

Assemblerprogrammering

Adresseringsmoder

Digitalteknik

ATmega16

Stacken

A/D- D/A-omvandlare

Avbrott

Samläsning D & I

D : TSEA82, 4hp 4 labbar (+labb 0)

I : TSEA57, 6hp 5 labbar (+labb 0)

Labbar + LAX = Godkänd kurs

LAX är Lab-eXamination!

Godkända
Labb : ca 95%
Lax : ca 80%

Vad händer när?

V13	V14	V15	V16	V17	V18	V19	V20	V21
Fö1	Fö4	Fö6	Fö7	Fö8	Fö9	Le		TPVT2
Fö2	La0	La1	La2		La3	La4	La5	LAX!
Fö3	Fö5							

Vad händer sen?

TSEA57 → TSEA56 : Elektronik kandidatprojekt (vt2023)

TSEA82 → TSEA83 : Datorkonstruktion (vt2023)

→ TSEA29 : Konstruktion med mikrodator (ht2023)

Resultat från föregående års utvärdering

Resultat 2021

Hur många läste kursen TSEA82/TSEA57 : 100 / 11
Hur många svarade på utvärderingen : 13 / 2
Helhetsbetyg : 4,77 / 5,00
Förändringar inför årets kurs:
-Återgång från distansläge
-Labbar med Dalia

Websida + kursmateriel

...

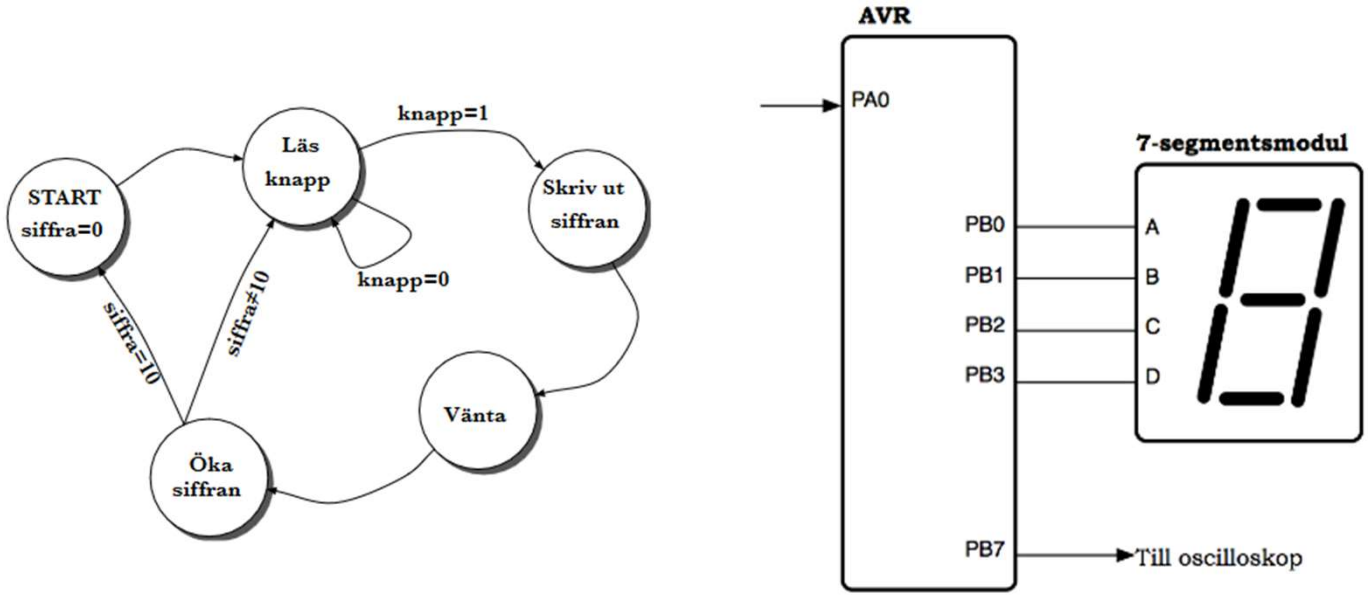
Laborationer

Laborationer

- Alla laborationer görs parvis el individuellt
- Labb0 är till för att bekanta sig med labmiljön
- Labb1->Labb4 är ordinarie labbar
- Labb5 är endast för I-programmet
- Anmälan till alla tillfällen görs i Lisam

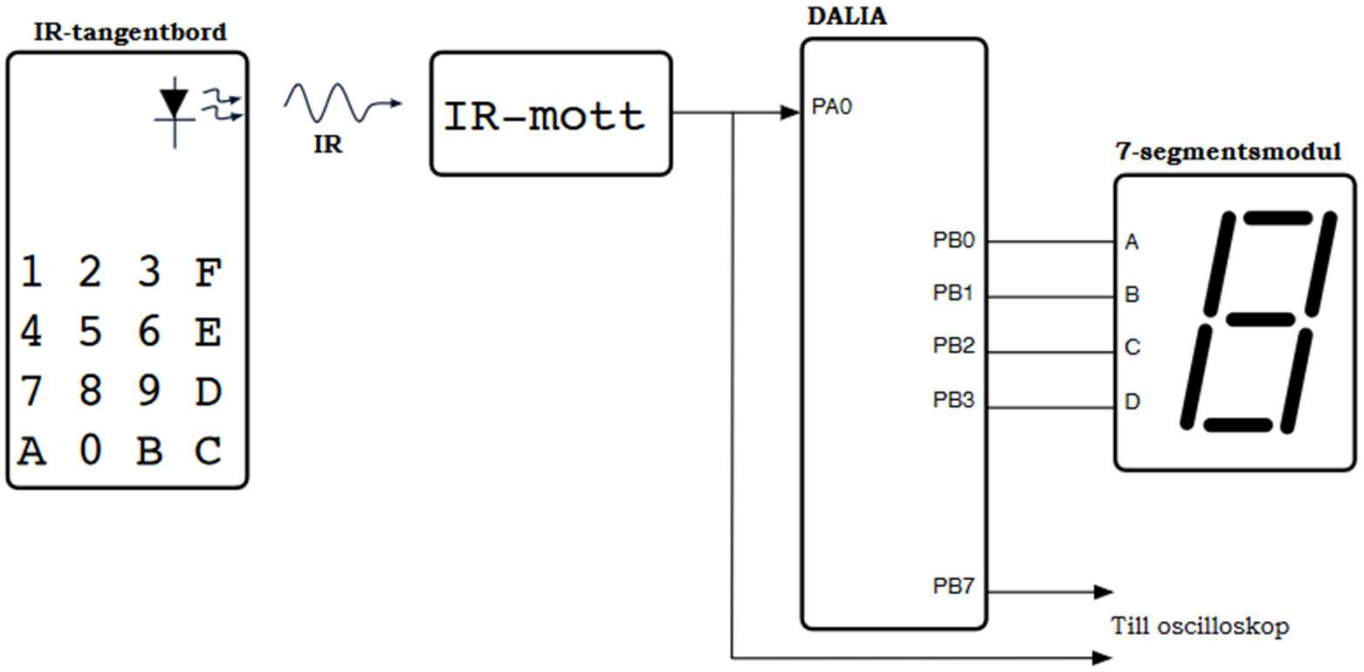
Labb0

- Atmel Studio + Miniprojekt



Labb1

- IR-länk

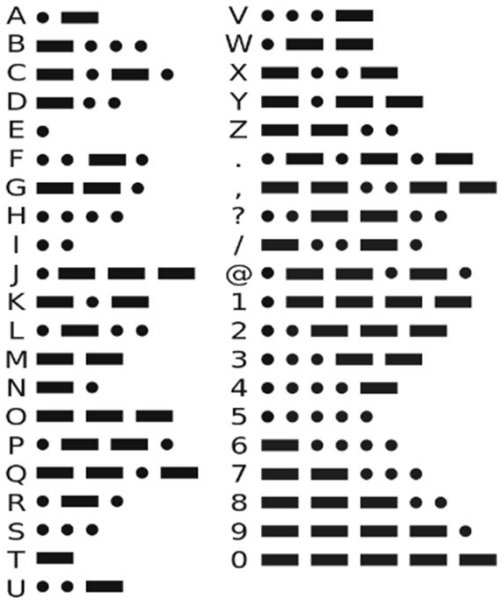


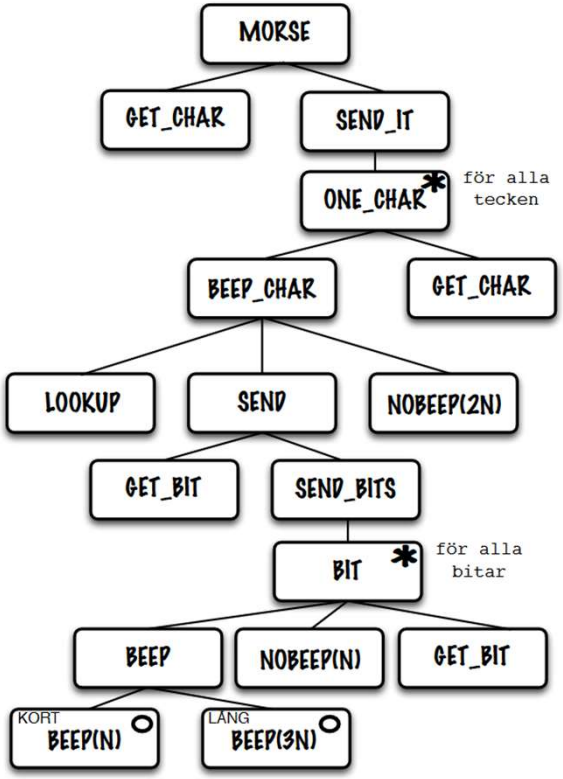
Moment:

- Enklare assembler-programmering
- Timing

Labb2

- Signalering av morse-kod





Moment:
- Subrutiner
- Timing

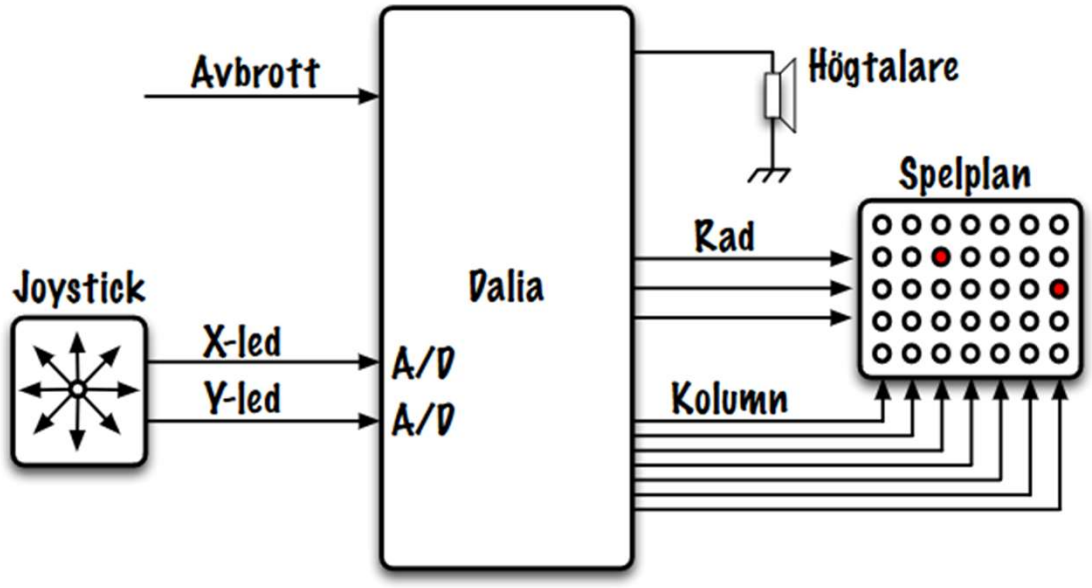
Labb3

- Digital-ur

Moment:
- Avbrott
- Multiplexning
- Subrutiner
- Timing

Labb4

- Spel



Moment:
- A/D-omvandling
- Avbrott
- Multiplexning
- Subrutiner
- Timing

Labb5, endast I-programmet

- C-programmering (digitalur + skenbar parallellism)

Moment:
- C-programmering
- Timers
- Parallellism
- Avbrott
- Multiplexning

LAX

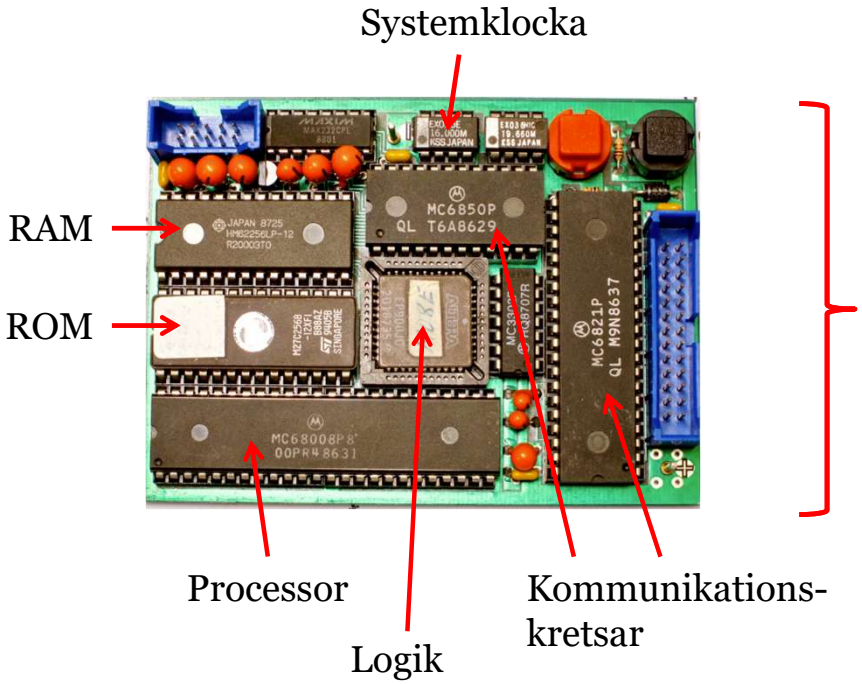
LAX (Lab-eXamination)

- LAX:en görs individuellt
- Uppgiften delas ut vid en viss tidpunkt, och ska redovisas inom 90 minuter
- En enklare mindre uppgift

Datormodellen + Programmerarmodell

Utveckling

Tutor (1985-2015)



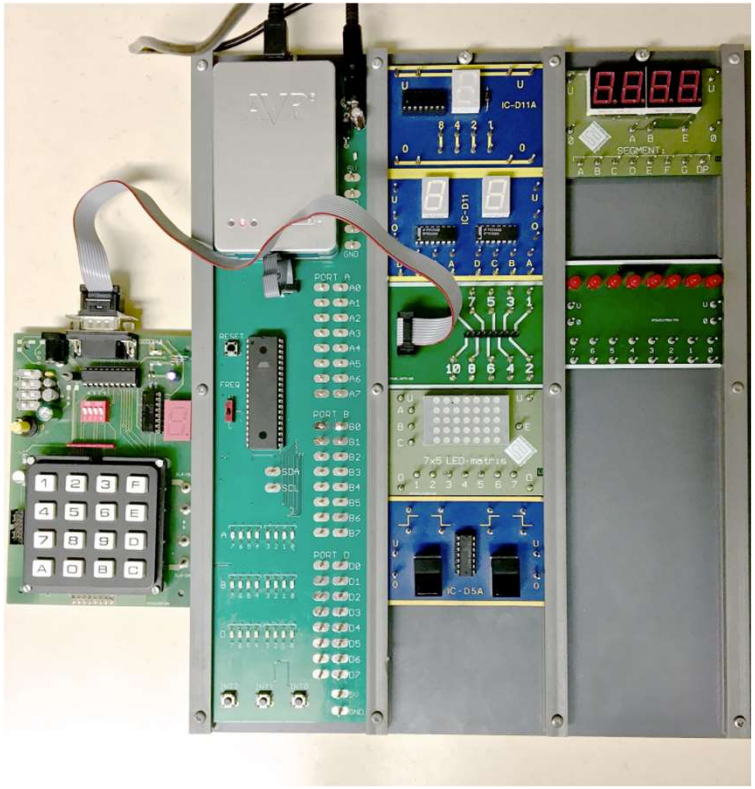
ATmega16 mikrokontroller

Allt på ett chip!
(och lite till)

Arduino UNO (ATmega328)

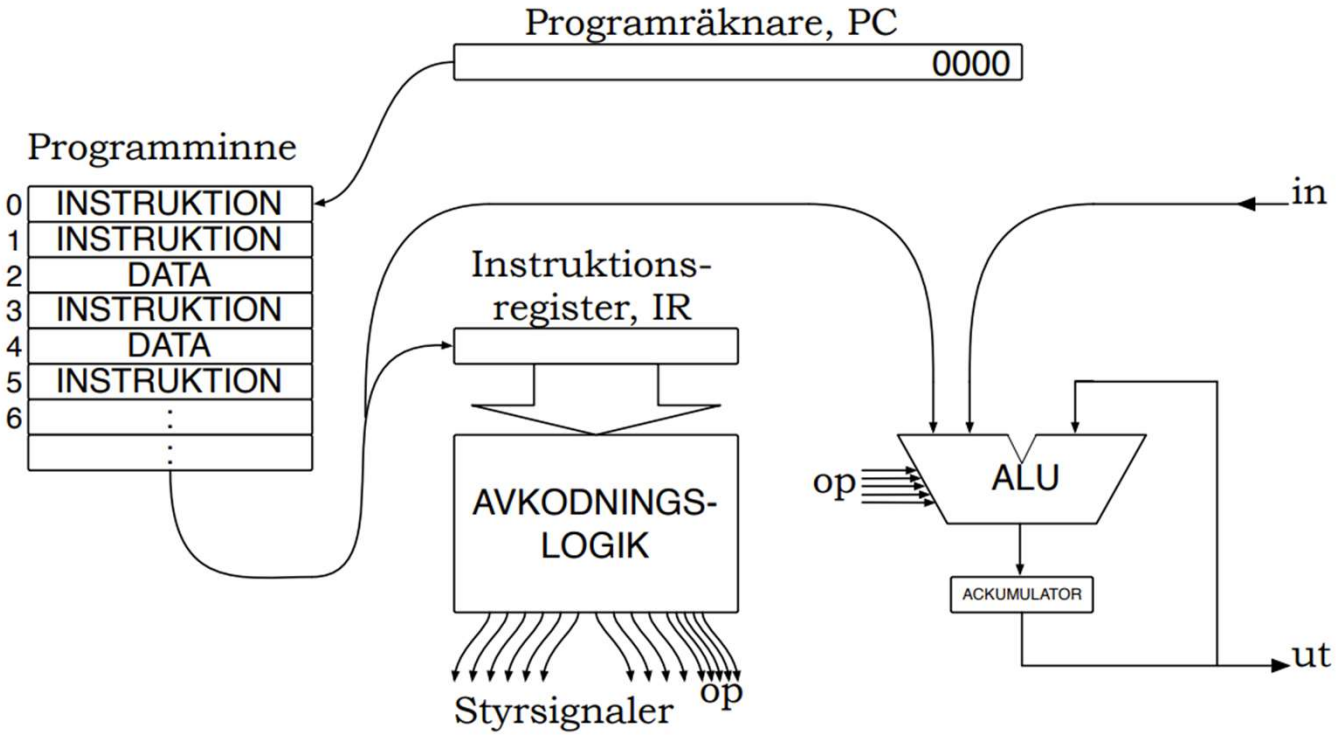
Ett av ***många*** användningsområden!

Dalia



Vi använder Dalia och processorn ATmega16.

Datormodell (grundläggande)



Nödvändiga delar

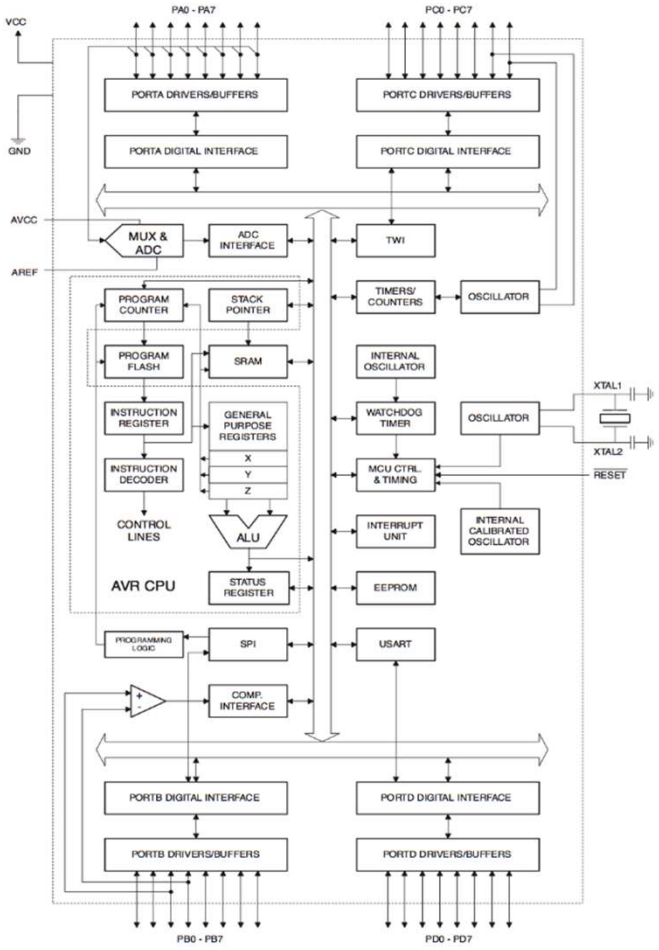
- Programräknare
- Programminne
- Instruktionsregister
- Avkodning+styrsignaler
- ALU
- In/Ut-enheter

Datormodell (ATmega16)

Utöver CPU:ns nödvändiga delar

- EEPROM
- Flera timers
- USART
- SPI
- TWI
- Komparator
- Debug-logik
- Watchdog
- m m ...

Används inte
i den här kursen,
MEN kommer till
användning i senare
kurser.



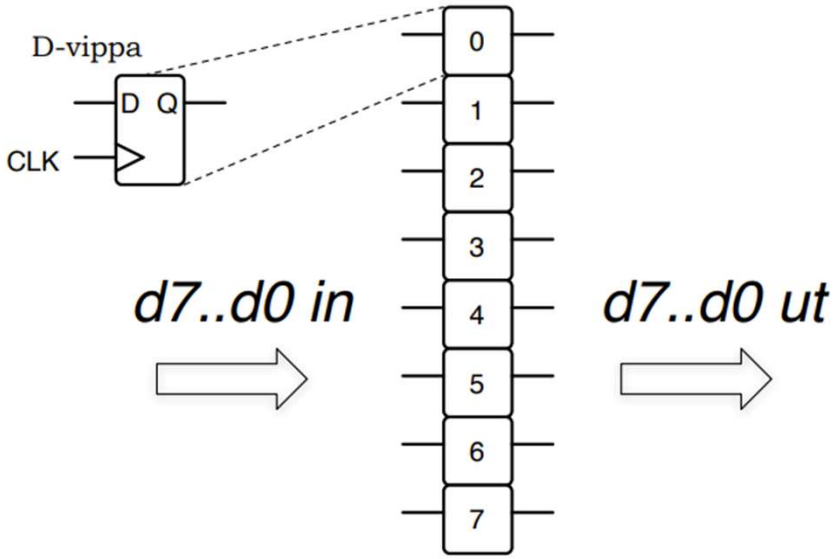
Programmerarmodell

Processorns grundläggande delar, registeruppsättning, programräknare och statusregister brukar kallas processorns *programmerarmodell*.

För ATmega16 blir det:

- Arbetsminnet, SRAM (för temporär lagring)
- Generella register, `r0-r31` (för generell användning)
- IO-register (register med specifik användning)
- Programräknare (adress till nuvarande instruktion)
- Statusregister (information om det senaste resultatet från ALU:n)

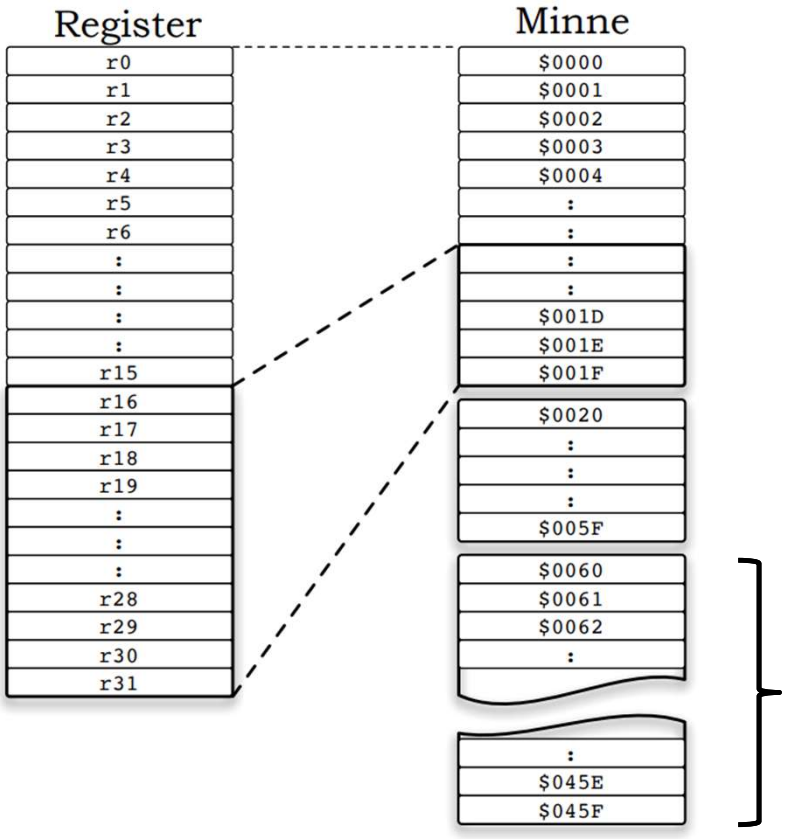
Programmerarmodell : Generella register (r0-r31)



- Samtliga generella register är `8` bitar breda
- Generella register `r0-r31` kan användas för godtyckligt syfte. MEN ”bara” `r16-r31` kan användas för alla processorns aritmetiska och logiska operationer.[1] Bara vissa operationer fungerar för `r0-r15`.
- Vissa generella register, `r24-r31`, kan kombineras parvis två och två, för att bilda `16` bitar breda register. `[r25:r24]`, `[r27:r26]`, `[r29:r28]`, `[r31:r30]`

[1]Instruktionen `and` kan utföras mellan godtyckliga register (`r0-r31`), men `andi` med en konstant kan bara utföras med `r16-r31`.

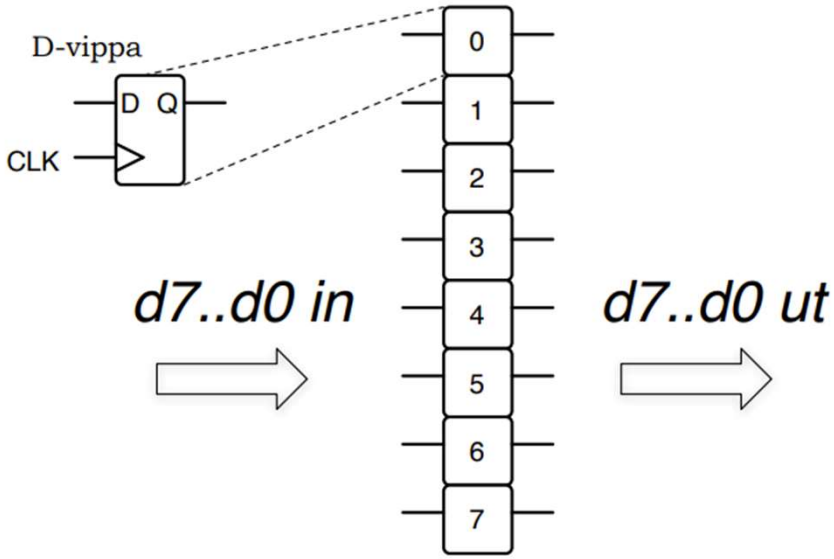
Programmerarmodell : Generella register (minnesmappning)



- Samtliga register har en motsvarande minnesadress
- Skriver man något till t ex register `r7`, så finns det tillgängligt att läsa på adress `$0007`.
- P s s, skriver man något till t ex adress `$0002`, så finns det tillgängligt att läsa i register `r2`.

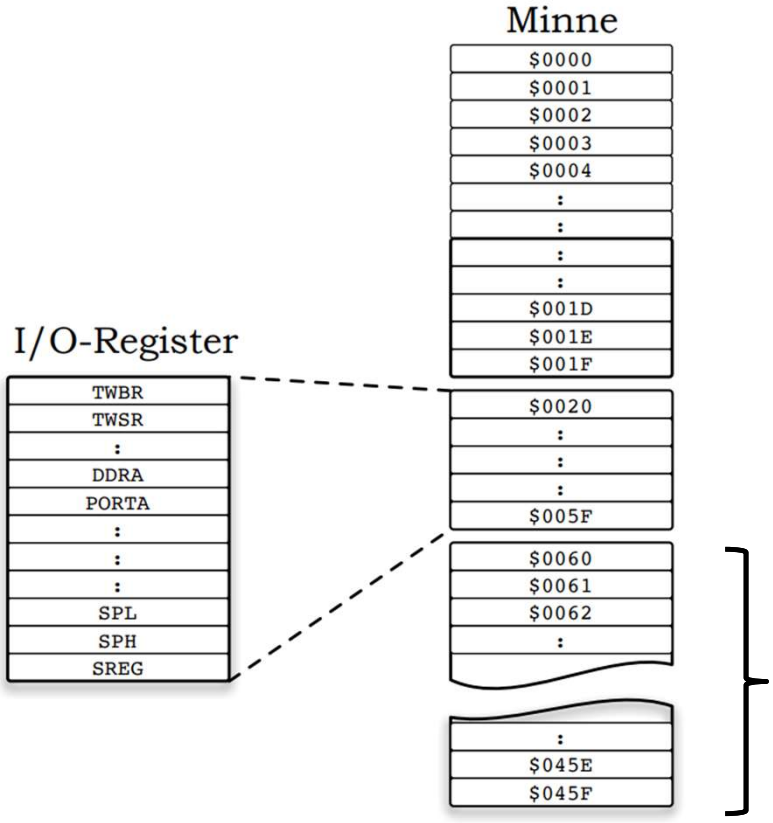
SRAM (1kB)

Programmerarmodell : I/O-register



- Samtliga I/O-register är 8 bitar breda
- I/O-register har ett specifikt syfte. T ex att styra datariktningen på en viss port (`DDRB`), eller att sätta portens utvärde (`PORTB`), eller för att läsa portens invärde (`PINB`).
- Alla register som inte är generella register, räknas som I/O-register, även om de inte har med in- eller ut-data att göra.
- Vissa I/O-register, t ex stackpekarens `SPH` och `SPL`, kombineras parvis två och två, för att bilda 16 bitar breda register. `SP=SPH:SPL`

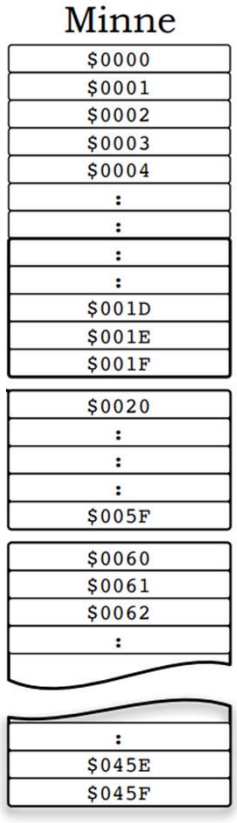
Programmerarmodell : I/O-register (minnesmappning)



- Samtliga I/O-register har en motsvarande minnesadress
- Skriver man något till t ex I/O-register `PORTB`, så finns det tillgängligt på adress `$0025`.
- P s s, skriver man något till just adress `$0025`, så finns det tillgängligt på `PORTB`.

SRAM (1kB)

Programmerarmodell : Arbetsminne (SRAM)



- Adresserna `$0000-$005F` är bara en mappning (en koppling) till något register, generellt register eller I/O-register.
- Det egentliga arbetsminnet (SRAM) finns mellan adresserna `$0060-$045F` (gäller ATmega16)

Programmerarmodell : Statusregistret (SREG)

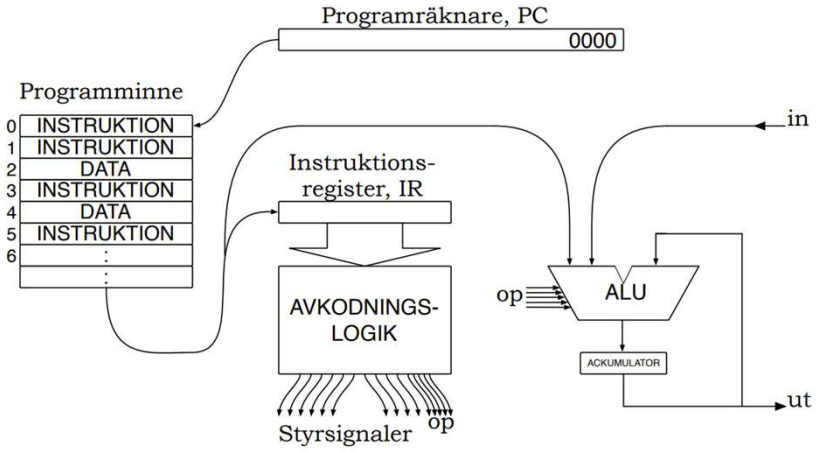
SREG

I	T	H	S	V	N	Z	C

Flagga	Innebörd
Z	1 om resultatet lika med noll
N	1 om resultatet mindre än noll, det vill säga mest signifikant bit satt
C	1 om *Carry*, det vill säga minnesbiten vid beräkningar vid teckenlösa tal satt
V	1 om *oVerflow*, om felaktigt resultat vid beräkningar av *2-komplementkodade* tal

- Statusregistret innehåller information, så kallade flaggor, om det senaste resultatet från t ex en aritmetisk eller logisk operation. Dvs, resultatet från den aritmetiska logiska enheten, ALU:n.
- Flaggorna Z, N, C och V är mest intressanta.

Programmerarmodell : Programräknaren (PC)

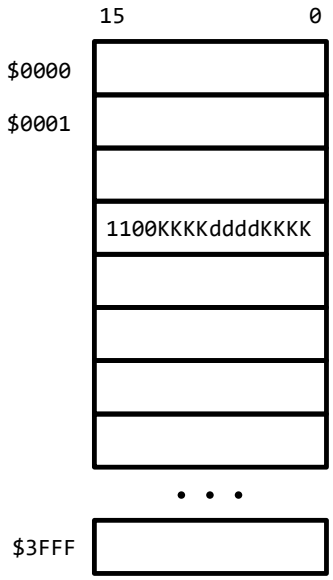


- Programräknaren innehåller adressen till den instruktion som utförs för tillfället.
- När programmet kör rad för rad kommer PC att räknas upp allteftersom.
- Om programmet gör ett hopp kommer PC att laddas med en ny adress, dvs dit man hoppar.

Instruktioner (början)

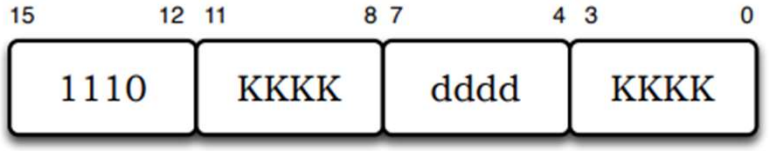
Instruktionsformat

Programminne (Flash)
16 bitar brett



- Instruktioner lagras i programminnet i ett instruktionsformat.
- Det finns flera instruktionsformat, dvs det varierar beroende på instruktion.
- Programminnet är 16 bitar brett
- Instruktionen `ldi r16,18` skulle t ex använda sig av nedanstående instruktionsformat och lagras som `1110 0001 0000 0010`:

`ldi Rd,K`



Instruktioner

I databladet för mikrokontrollern finns drygt hundratalet 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`)

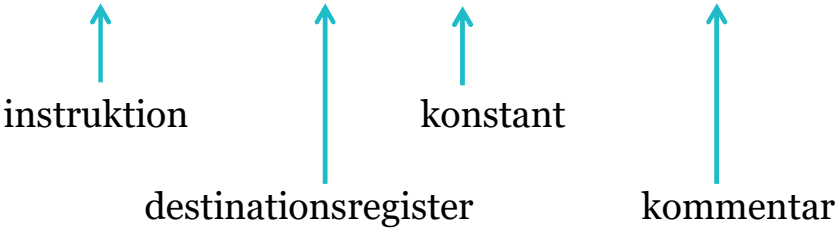
Instruktioner : Grupp 1. Flytta data[1]

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

Exempel: `ldi`

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

```
ldi     r16,23      ; Load immediate
```



[1]Egentligen flyttar man inte data, man kopierar data.

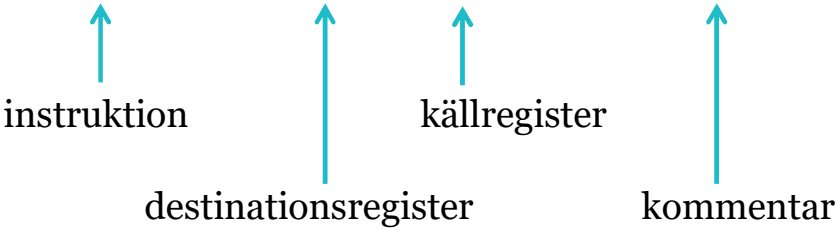
Instruktioner : Grupp 1. Flytta data[1]

Mellan generella register används instruktionen `mov`.

Exempel: `mov`

Flytta (kopiera) innehållet i register `r16` till `r13`

```
mov     r13,r16    ; Load immediate
```



[1]Egentligen flyttar man inte data, man kopierar data.
[Simulera i Atmel Studio]

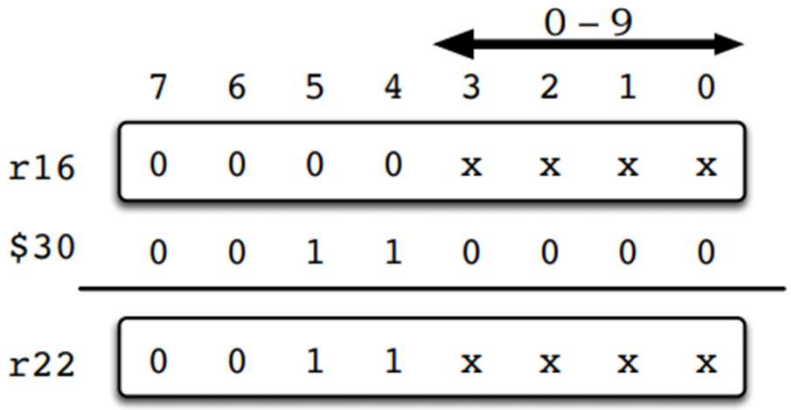
Instruktioner : Grupp 1. Flytta data

8-bitars register, kan inneha tal mellan `0-255` (`00000000-11111111`)

Exempel: Decimalsiffra till ASCII[1]

Register `r16` innehåller en siffra, `0-9`.

Översätt den till ASCII[1]




```
ldi     r22,$30
add     r22,r16
   alternativt
mov     r22,r16
ori     r22,$30
```


[1]American Standard Code for Information Interchange

Instruktioner : Grupp 1. Flytta data

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

Exempel: Öka på `r24:r25` med ett steg

`adiw` fungerar enbart för registerparen `r25:r24`, `r27:r26`, `r29:r28`, `r31:r30`.

```
adiw  r24,1    ; r25 underförstås
```

Exempel: Öka på `r17:r16` med ett steg

```
    inc   r16      ; affects Z but not C
    brne  DONE
    inc   r17
DONE:
```

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
```

Nödvändiga programvaror

Nödvändiga programvaror

- Atmel Studio, kursen använder version 7
- Om man inte har Windows:
  -Virtual Box (Mac/Linux)
- Instruktioner finns vid följande länk:
  http://www.isy.liu.se/edu/kurs/TSEA82/atmelarduino/

Tid för Frågor

Anders Nilsson

www.liu.se

LINKÖPINGS UNIVERSITET