Aktivera Kursens mål:

- Konstruera en dator mha grindar och programmera denna
- Studera en kommersiell processor - CPU12/HC12/MC12

Aktivera Förra veckans mål:

- Skriva program (för FLEX)
- Introduktion till CPU12

Veckans mål:

- Studera CPU12/HC12/MC12
- Utvecklingsmiljön Eterm
- Skriva assemblerprogram (för MC12)
- Lab4
- Adressavkodning
- Ansluta minnen och I/O-portar

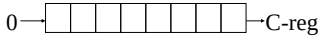
Dagens mål: Du ska kunna

- **Känna till olika instruktionstyper**
- Kontrollstrukturer
  - If then else; While, Do while
  - Använda tabell 5 (Branch'er) i instruktionslistan
  - "testa bitar"
- Skriva delayrutiner
- Skriva assemblerprogram för CPU12
- Laboration 4 och MC12

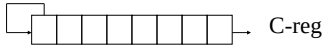
Nya instruktioner

DAA Decimal adjust A-reg

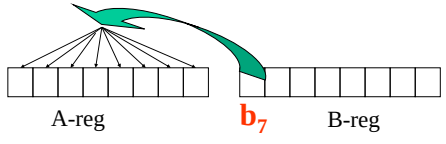
LSRA A/2 utan tecken



ASRA A/2 med tecken
(ett negativt tal blir inte mindre än -1!)



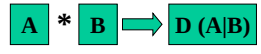
SEX Sign extend



Nya instruktioner

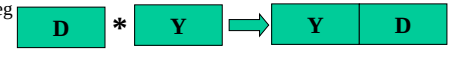
8-bit MUL (Unsign) **MUL**

A-reg*B-reg→D-Reg
(8 * 8 = 16 bit)



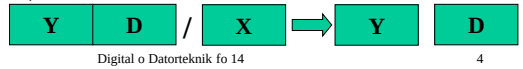
16-bit MUL ….. (Sign/Unsign) **EMUL, EMULS**

D-reg*Y-reg→Y:D-reg
(16 * 16 = 32 bit)



16-bit DIV….. (Sign/Unsign) **EDIV, EDIVS**

Y:D-reg / X-reg→Y (kvot) D-rest
(32 / 16 = 16 bit)



Nya instruktioner

Instruktioner för addition

Mnemonic	Funktion	Operation
ABA	Addera B till A	(A)+(B) → A
ABX	Addera B till X anm: Ekv. med `LEAX B,X`	(X)+(B) → X
ABY	Addera B till Y anm: Ekv. med `LEAY B,Y`	(Y)+(B) → Y
ADCA	Addition med carry till A	(A)+(M)+C → A
ADCB	Addition med carry till B	(B)+(M)+C → B
ADDA	Addition till A	(A)+(M) → A
ADDB	Addition till B	(B)+(M) → B
ADDD	Addition till D (A:B)	(D)+(M:M+1) →D

Nya instruktioner

Instruktioner för addition

Mnemonic	Funktion	Operation
INC	Incrementera i minnet	(M)+$01 → M
INCA	Inkrementera A	(A)+ $01 → A
INCB	Inkrementera B	(B)+ $01 → B
INS	Inkrementera SP anm: Ekv. med `LEAS 1,SP`	(SP)+ $0001 → SP
INX	Inkrementera X anm: Ekv. med `LEAX 1,X`	(X)+ $0001 → X
INY	Inkrementera Y anm: Ekv. med `LEAY 1,Y`	(Y)+ $0001 → Y

Nya instruktioner

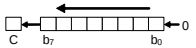
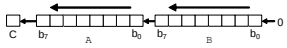
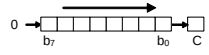
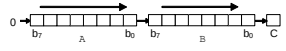
Instruktioner för subtraktion

Mnemonic	Funktion	Operation
SBA	Subtrahera B från A	(A)-(B) → A
SBCA	Subtrahera med borrow från A	(A)-(M)-C → A
SBCB	Subtrahera med borrow från B	(B)-(M)-C → B
SUBA	Subtrahera från A	(A)-(M) → A
SUBB	Subtrahera från B	(B)-(M) → B
SUBD	Subtrahera från D (A:B)	(D)-(M:M+1) →D

Mnemonic	Funktion	Operation
DEC	Dekrementera i minnet	(M)-$01 → M
DECA	Dekrementera A	(A)- $01 → A
DECB	Dekrementera B	(B)- $01 → B
DES	Dekrementera SP anm: Ekv. med `LEAS -1,SP`	(SP)- $0001 → SP
DEX	Dekrementera X anm: Ekv. med `LEAX -1,X`	(X)- $0001 → X
DEY	Dekrementera Y anm: Ekv. med `LEAY -1,Y`	(Y)- $0001 → Y

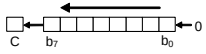
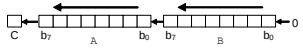
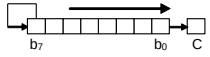
Nya instruktioner

Logiska skiftoperationer

Mnemonic	Funktion	Operation
LSL	Logiskt vänsterskift i minnet	
LSLA	Logiskt vänsterskift A	
LSLB	Logiskt vänsterskift B	
LSLD	Logiskt vänsterskift D	
LSR	Logiskt högerskift i minnet	
LSRA	Logiskt högerskift A	
LSRB	Logiskt högerskift B	
LSRD	Logiskt högerskift D	

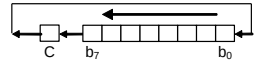
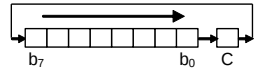
Nya instruktioner

Aritmetiska skiftoperationer

Mnemonic	Funktion	Operation
ASL	Aritmetiskt vänsterskift i minnet (ekv. med LSL)	
ASLA	Aritmetiskt vänsterskift A (ekv. med LSLA)	
ASLB	Aritmetiskt vänsterskift B (ekv. med LSLB)	
ASLD	Aritmetiskt vänsterskift D (ekv. med LSLD)	
ASR	Aritmetiskt högerskift i minnet	
ASRA	Aritmetiskt högerskift A	
ASRB	Aritmetiskt högerskift B	

Nya instruktioner

Instruktioner för rotation (carry-skift)

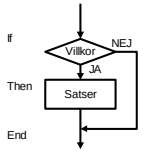
Mnemonic	Funktion	Operation
ROL	Rotation vänster via carry i minnet	
ROLA	Rotation vänster via carry A	
ROLB	Rotation vänster via carry B	
ROR	Rotation höger via carry i minnet	
RORA	Rotation höger via carry A	
RORB	Rotation höger via carry B	

LV6 Fo14

Dagens mål: Du ska kunna

- Känna till olika instruktionstyper
- **Kontrollstrukturer**
  - **If then else; While, Do while**
  - **Använda tabell 5 (Branch'er) i instruktionslistan**
  - **"testa bitar"**
- Skriva delayrutiner
- Skriva assemblerprogram för CPU12
- Laboration 4 och MC12

Kontrollstruktur if (...) {Satser}



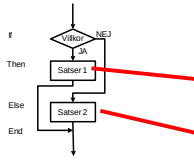

```
if (DipSwitch != 0)
   HexDisp = Dipswitch;
```

Bättre kodning...

```
        ...
        TST   DipSwitch
        BEQ   end

        LDAB  DipSwitch
        STAB  HexDisp
end     ...
        ...
```

BNE	"Hopp" om ICKE *zero*	Z=0
BEQ	"Hopp" om *zero*	Z=1

Kontrollstruktur **if (...) {Satser1} else {Satser2}**




```
if (DipSwitch == 0)
      HexDisp = 1;
else
      HexDisp = 2;
```

```
            ...
            LDAB  DipSwitch
            TSTB
            BNE   else

            LDAB  #1
            STAB  HexDisp
            BRA   end

else        LDAB  #2
            STAB  HexDisp
end         ...
            ...
```

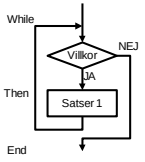
{Satser1}

{Satser2}

BNE	"Hopp" om ICKE *zero*	Z=0

Kontrollstruktur **while (...) {Satser}**

while (...) {...}




```
while (DipSwitch != 0)
      HexDisp = 1;
```

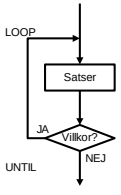
```
            ...
while
            LDAB  DipSwitch
            TSTB
            BEQ   end_while

            LDAB  #1
            STAB  HexDisp
            BRA   while
end_while
            ...
            ...
```

{Satser}

BEQ	"Hopp" om *zero*	Z=1

Kontrollstruktur **do {Satser} while (...)**

do {...} while (...)




```
do
{
      HexDisp = 0;
}while (DipSwitch >= 10);
```

```
            ...
do
            MOVB  #0,HexDisp

            LDAB  DipSwitch
            CMPB  #10
            BHS   do
            ...
```

{Satser}

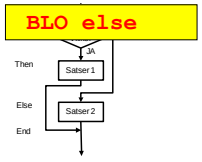
Test av tal utan tecken		
BHS	Villkor: R≥M	C=0

Villkorlig programflödeskontroll

Mnemonic	Funktion	Villkor
	Enkla flaggtest	
BCS	"Hopp" om *carry*	C=1
BCC	"Hopp" om ICKE *carry*	C=0
BEQ	"Hopp" om *zero*	Z=1
BNE	"Hopp" om ICKE *zero*	Z=0
BMI	"Hopp" om *negative*	N=1
BPL	"Hopp" om ICKE *negative*	N=0
BVS	"Hopp" om *overflow*	V=1
BVC	"Hopp" om ICKE *overflow*	V=0
	Test av tal utan tecken	
BHI	Villkor: R>M	C + Z = 0
BHS	Villkor: R≥M	C=0
BLO	Villkor: R<M	C=1
BLS	Villkor: R≤M	C + Z = 1
	Test av tal med tecken	
BGT	Villkor: R>M	Z + (N ⊕ V) = 0
BGE	Villkor: R≥M	N ⊕ V = 0
BLT	Villkor: R<M	N ⊕ V = 1
BLE	Villkor: R≤M	Z + (N ⊕ V) = 1

Inst s23, Tabell 5

Kontrollstruktur **if (...) {Satser1} else {Satser2}**



R >= M

```
if (DipSwitch >= 5)
      HexDisp = 1;
else
      HexDisp = 2;
```

```
            ...
        1)  LDAB  DipSwitch
        2)  CMPB  #5
        3)  Bxx   else

            LDAB  #1
            STAB  HexDisp
            BRA   end          {Satser1}

else        LDAB  #2
            STAB  HexDisp      {Satser2}

end         ...
            ...
```

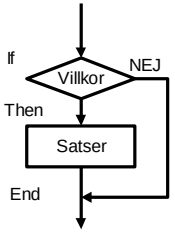
Test av tal utan tecken		
BLO	Villkor: R<M	C=1

Test av tal med tecken		
BLT	Villkor: R<M	N ⊕ V = 1

Inst s23, Tabell 5

Att testa bitar på inporten – *EN BIT*

If villkor then
Satser
end



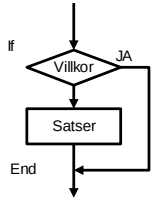
if b_3=1 then
Satser
end

```
        LDAA    Inport
        ANDA    #%00001000
        BEQ     end

        ---         Satser

end
        ---
```

```
        LDAA    Inport
        BITA    #%00001000
        BEQ     end

        ---         Satser

end
        ---
```


Att testa bitar på inporten – *FLERA BITAR*

if (b_3=1) && (b_0=1) then goto end
Satser
end




```
        LDAA    Inport
        ANDA    #%00001001
        CMPA    #%00001001
        BEQ     end

        ---         Satser 1

end
        ---
```

```
        BRSET   Inport,#%00001001,end

        ---         Satser 1

end
        ---
```


LV6 Fo14

Dagens mål: Du ska kunna

- Känna till olika instruktionstyper
- Kontrollstrukturer
  - If then else; While, Do while
  - Använda tabell 5 (Branch'er) i instruktionslistan
  - "testa bitar"
- **Skriva delayrutiner**
- Skriva assemblerprogram för CPU12
- Laboration 4 och MC12

Läs kurt! Lär dig mera!

Nya instruktioner

Mnemonic	Funktion	Villkor
DBEQ	Dekrementera innehåll i register. “Hoppa” om resultatet = 0. (register: A,B,D,X,Y,SP)	(register) – 1 ⇒ register om(register)=0; “hoppa”; annars: nästa instruktion
DBNE	Dekrementera innehåll i register. “Hoppa” om resultatet ≠ 0. (register: A,B,D,X,Y,SP)	(register) – 1 ⇒ register om(register)≠0; “hoppa”; annars: nästa instruktion
IBEQ	Inkrementera innehåll i register. “Hoppa” om resultatet = 0. (register: A,B,D,X,Y,SP)	(register) + 1 ⇒ register om(register)=0; “hoppa”; annars: nästa instruktion
IBNE	Inkrementera innehåll i register. “Hoppa” om resultatet ≠ 0. (register: A,B,D,X,Y,SP)	(register) + 1 ⇒ register om(register)≠0; “hoppa”; annars: nästa instruktion
TBEQ	Testa innehåll i register. “Hoppa” om resultatet = 0. (register: A,B,D,X,Y,SP)	om(register)=0; “hoppa”; annars: nästa instruktion
TBNE	Testa innehåll i register. “Hoppa” om resultatet ≠ 0. (register: A,B,D,X,Y,SP)	om(register)≠0; “hoppa”; annars: nästa instruktion

Delay-rutiner

```
Delay( unsigned int count )
{
  while (count > 0)
    count = count - 1;
}
```

Parameter ’count’ finns i register D vid anrop. Anm. count=0 är EJ TILLÅTET.

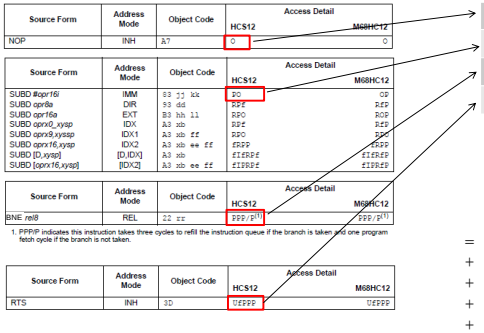
```
* Subrutin 'Delay'

Delay           NOP

Delay_loop      NOP
                NOP
                SUBD    #1
                BNE     Delay_loop

                RTS
```

instruktion	antal ggr.
NOP	1
NOP	count
NOP	count
SUBD #1	count
BNE	Count (”taken”)
BNE	1 (not taken)
RTS	1

```
= NOP     ( 1 + 2 count)
+ SUBD#1  ( count )
+ BNE_T   ( count-1 )
+ BNE_NT  ( 1 )
+ RTS     ( 1 )
=  ?
```


Delay-rutiner

(exekveringstider, dvs antal cykler, fås ur instruktionslistan...)



instruktion	# cykler
NOP	1
SUBD #1	2
BNE	3/1
RTS	5

```
= 1 ( 1 + 2 count)
+ 2 ( count )
+ 3 ( count-1 )
+ 1 ( 1 )
+ 5 ( 1 )
= 7 count + 4
```


Delay-rutiner

Exempel: Bestäm ’count’ för 10 ms fördröjning i ett MC12-system

MC12

Frekvens/ cykeltid	Min. (’count’ = 1) 11 cykler	Max. (’count’ = $FFFF) 458749 cykler
8 MHz/ 125 ns.	1,375 µs	57,34 ms

Lösning:

$$(7count + 4)125ns = 10ms$$

$$(7count + 4)125ns = 10000000ns$$

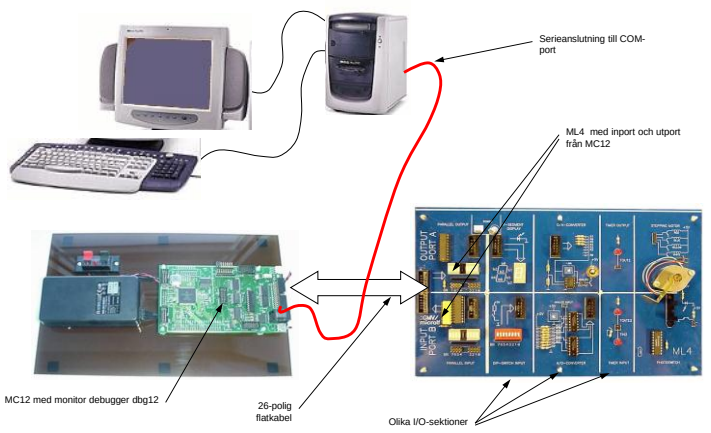
$$count = \frac{\frac{10000000}{125} - 4}{7} = 11428$$

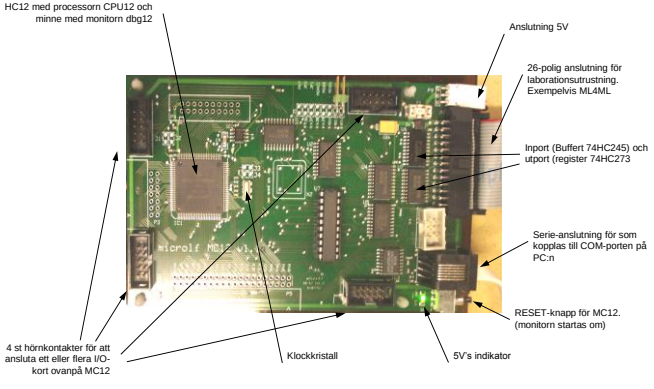
Uppskatta motsvarande fördröjning i simulatorn

... Tar c:a 14 sekunder

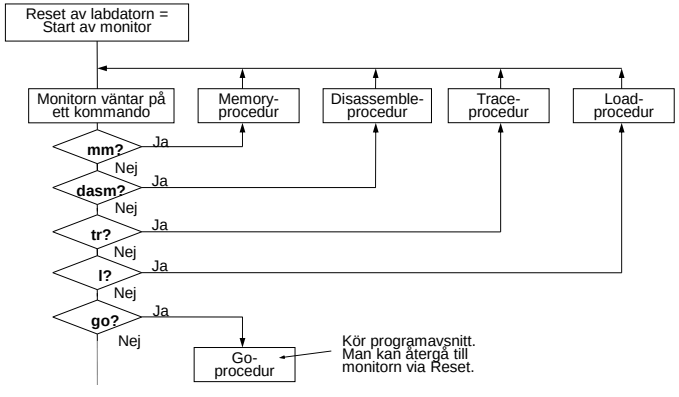
Dagens mål: Du ska kunna

- Känna till olika instruktionstyper
- Kontrollstrukturer
  - If then else; While, Do while
  - Använda tabell 5 (Branch'er) i instruktionslistan
  - "testa bitar"
- Skriva delayrutiner
- Skriva assemblerprogram för CPU12
- **Laboration 4 och MC12**

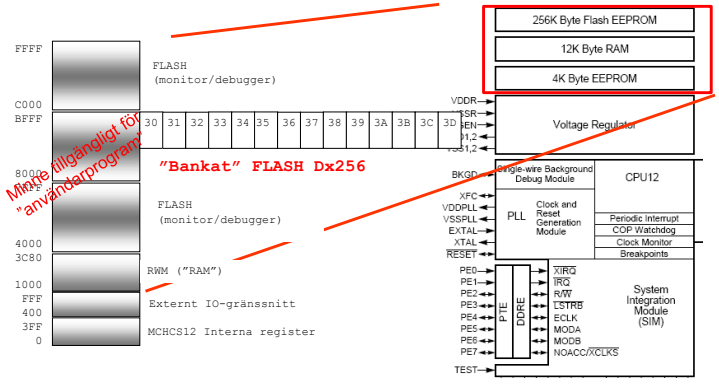




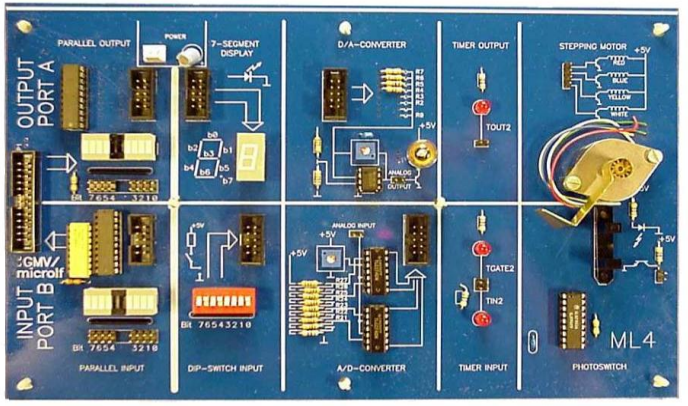
MC12 – dbg12



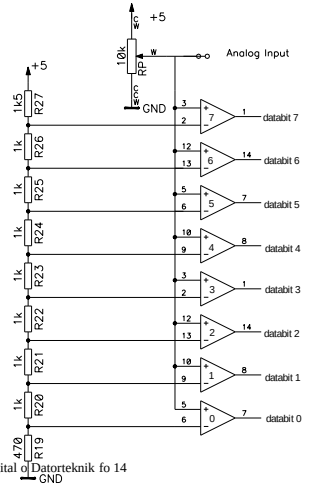
HCS12DG256, ”core”



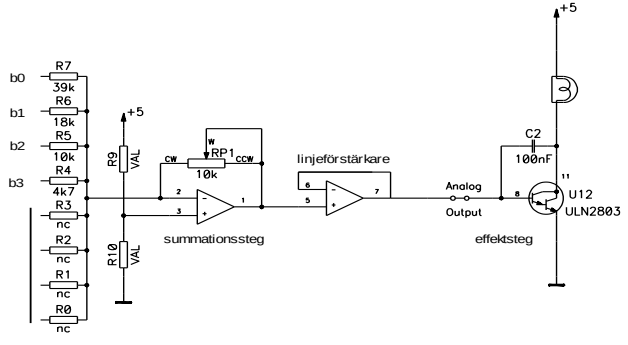
Mindre tillgängligt för ”användarprogram”



Analog-digital-omvandling



Digital- Analog-omvandling



Veckans mål: **LV6 Fo15**

- Studera CPU12/HC12/MC12
- Utvecklingsmiljön Eterm
- Skriva assemblerprogram (för MC12)
- Lab4
- Adressavkodning
- Ansluta minnen och I/O-portar

Dagens mål: Du ska kunna….

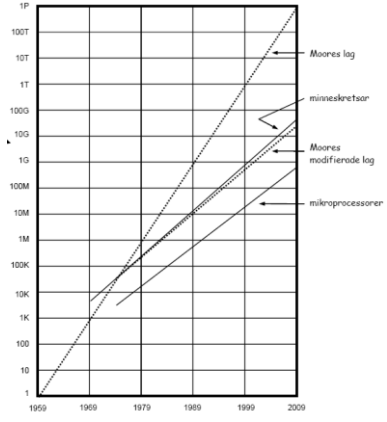
- Skilja på olika minnestyper
- Förklara principer för olika bussprotokoll
- Beskriva MC12's bussar (anslutningar)
- Förstå begreppen Timing och VM (Valid Memory Adress)
- Konstruera adressavkodningslogik ….
  - 1) Fullständig Adr Avk
  - 2) Ofullständig Adr Avk
  - Processorns adressrum

Lite om Minnen

•"RAM"	Random Access Memory
•RWM	Read Write Memory
•SRAM	Statiskt RAM
•DRAM	Dynamiskt
•ROM	Read Only Memory
•PROM	Programmable ROM
•EPROM	Erasable PROM
•EEPROM	Electrically EPROM
•FLASH	

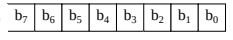
Mikroelektronikens utveckling

Antalet transistorer som ryms på en kiselbricka....



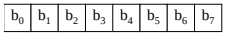
Bit ordning "bit endianess"

big-endian ("LSB 0")

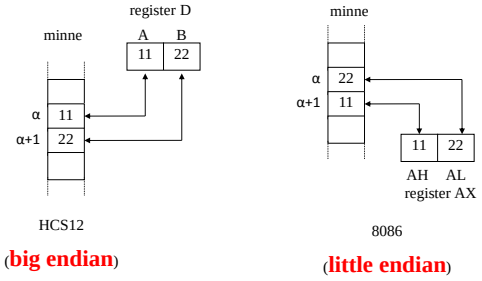


8-bitars ord där b_7 är den MEST signifikanta biten och b_0 den MINST signifikanta biten

little-endian ("MSB 0")

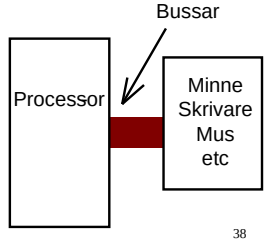


HCS12/Intel 8086 **byte ordning**

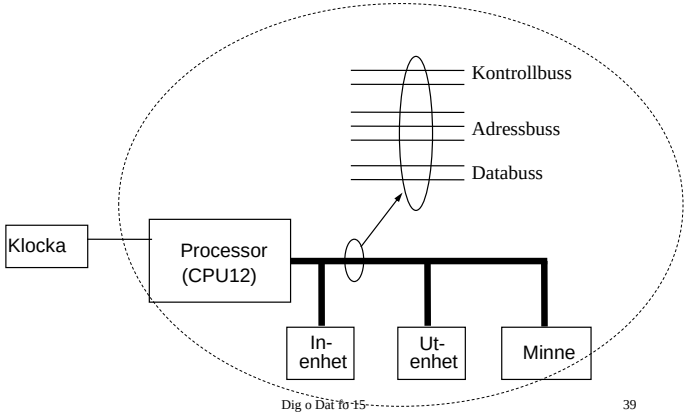


Dagens mål: Du ska kunna….

- Skilja på olika minnestyper
- **Förklara principer för olika bussprotokoll**
- Beskriva MC12's bussar (anslutningar)
- Förstå begreppen Timing och VM (Valid Memory Adress)
- Konstruera adressavkodningslogik ….
  - 1) Fullständig Adr Avk
  - 2) Ofullständig Adr Avk
  - Processorns adressrum

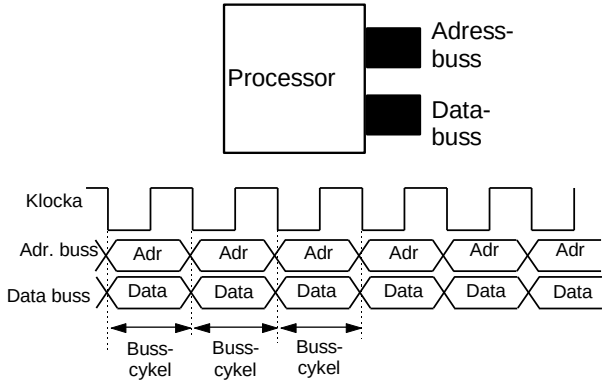


S



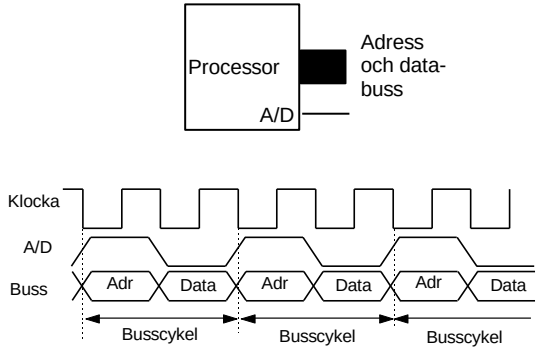
Icke Multiplexad Buss

S



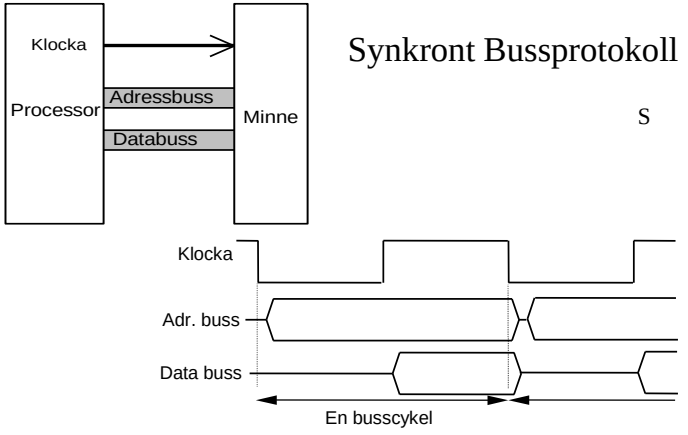
Multiplexad Buss

S



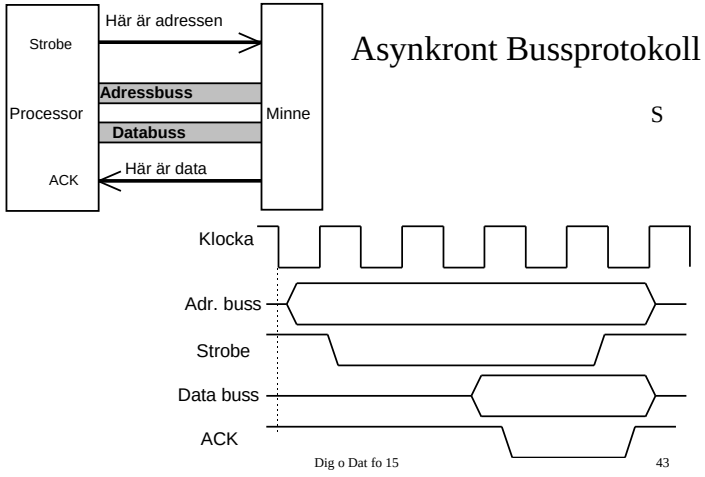
Synkront Bussprotokoll

S



Asynkront Bussprotokoll

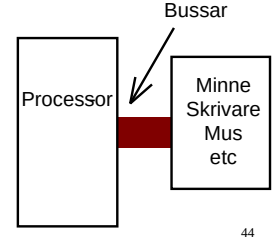
S



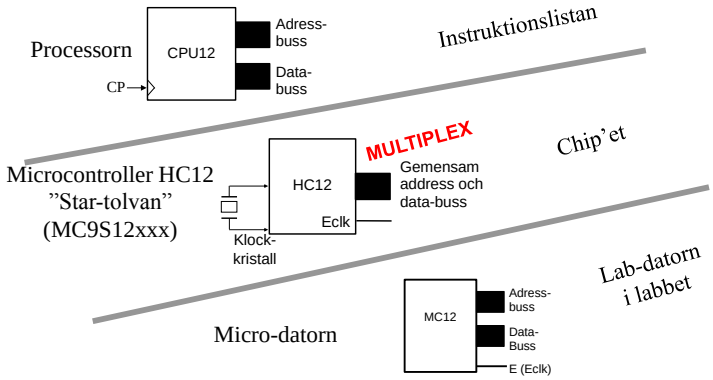
LV6 Fo15

Dagens mål: Du ska kunna….

- Skilja på olika minnestyper
- Förklara principer för olika bussprotokoll
- **Beskriva MC12's bussar (anslutningar)**
- Förstå begreppen Timing och VM (Valid Memory Adress)
- Konstruera adressavkodningslogik ….
  - 1) Fullständig Adr Avk
  - 2) Ofullständig Adr Avk
  - Processorns adressrum

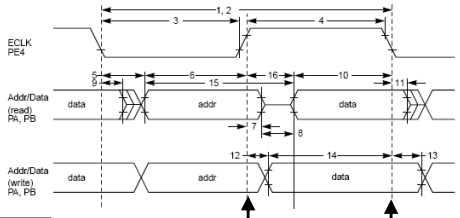


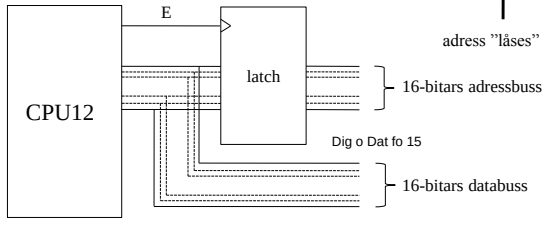
CPU12 / HC12 / MC12



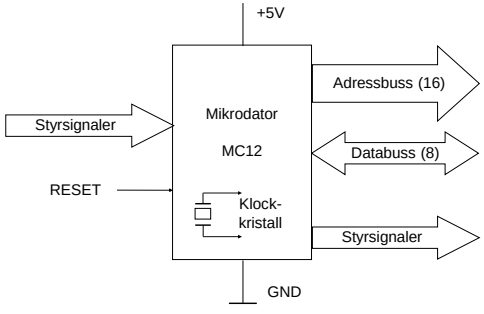
Synkron Multiplex-buss, HC12

Vid positiv flank hos E(CLK) "låses" adressen via latchen





MC12: anslutningar



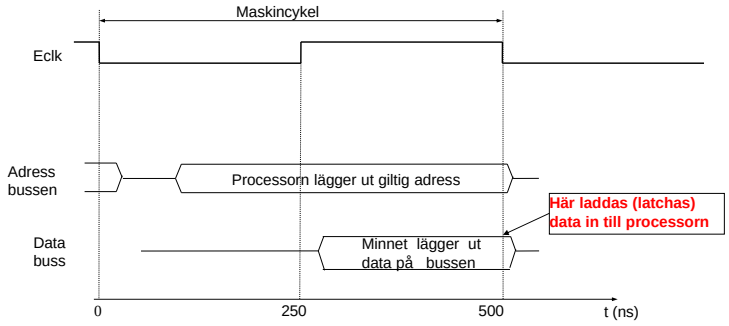
LV6 Fo15

Dagens mål: Du ska kunna….

- Skilja på olika minnestyper
- Förklara principer för olika bussprotokoll
- Beskriva MC12's bussar (anslutningar)
- **Förstå begreppen Timing och VM (Valid Memory Adress)**
- Konstruera adressavkodningslogik ….
  - 1) Fullständig Adr Avk
  - 2) Ofullständig Adr Avk
  - Processorns adressrum

MC12's-Timing

s

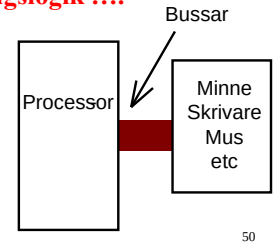


VALID MEMORY ADRESS; VMA

VALID ADRESS; VA

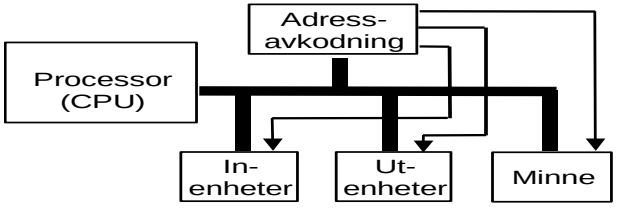
Dagens mål: Du ska kunna….

- Skilja på olika minnestyper
- Förklara principer för olika bussprotokoll
- Beskriva MC12's bussar (anslutningar)
- Förstå begreppen Timing och VM (Valid Memory Adress)
- **Konstruera adressavkodningslogik ….**
  - 1) Fullständig Adr Avk
  - 2) Ofullständig Adr Avk
  - Processorns adressrum



In och Utportar på FLEX

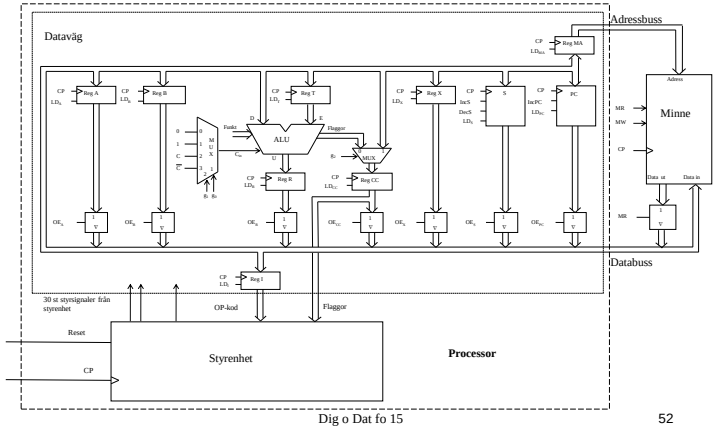
Arb s 167



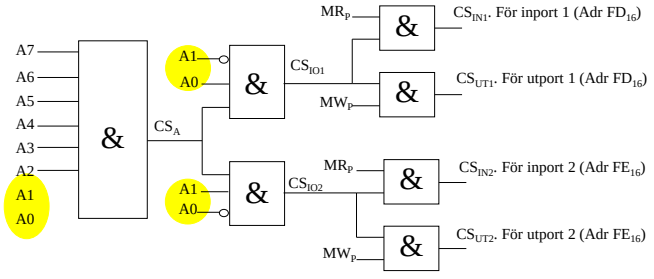
LDAA Adr
STAA Annan Adr

LDAA INPORT
STAA UTPORT

Arb s 133

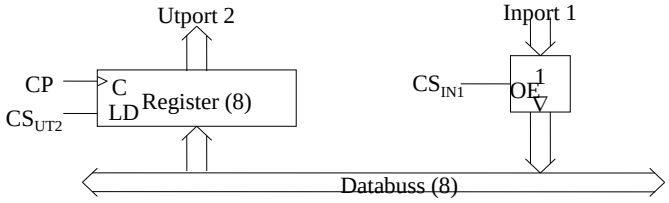


In och Utportar på FLEX Arb s 168



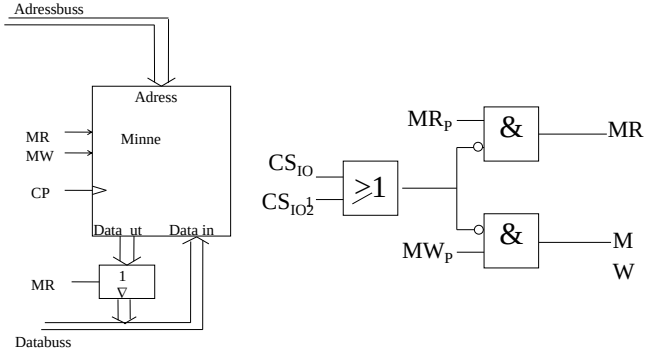

```
LDAA   $FD          LDAA   INPORT1
STAA   $FE          STAA   UTPORT2
```


In och Utportar på FLEX Arb s 168




```
LDAA   $FD          LDAA   INPORT1
STAA   $FE          STAA   UTPORT2
```

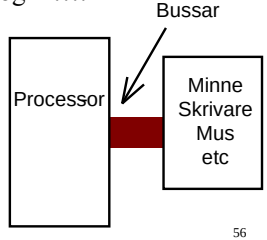

In och Utportar på FLEX Arb s 169



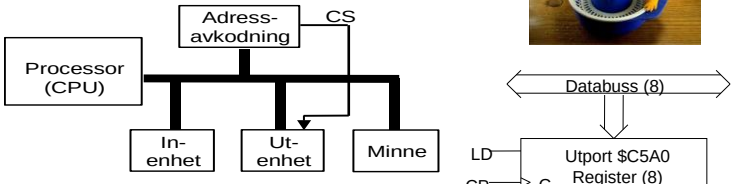
LV6 Fo15

Dagens mål: Du ska kunna….

- Skilja på olika minnestyper
- Förklara principer för olika bussprotokoll
- Beskriva MC12's bussar (anslutningar)
- Förstå begreppen Timing och VM (Valid Memory Adress)
- Konstruera adressavkodningslogik ….
  - **1) Fullständig Adr Avk**
  - 2) Ofullständig Adr Avk
  - Processorns adressrum



Adressavkodning

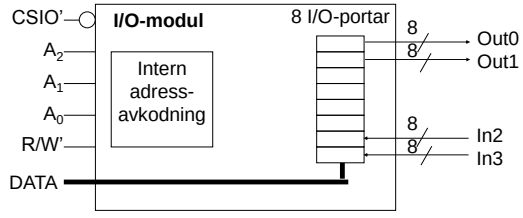



```
STAA   $C5A0
STAA   UTPORTADRESS
```

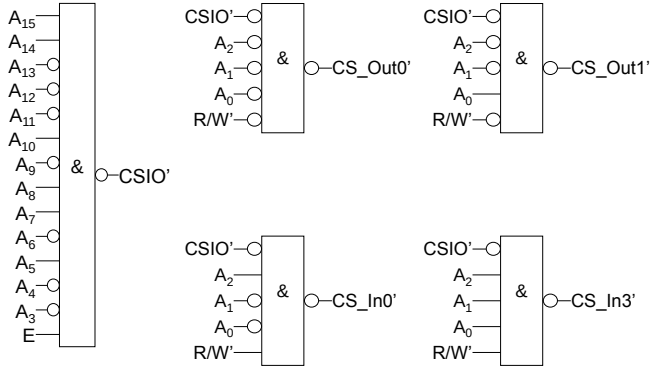

Ut0 på adress $C5A0
Ut1 på adress $C5A1
Ut2 på adress $C5A2
Ut3 på adress $C5A3
In0 på adress $C5A4
In1 på adress $C5A5
In2 på adress $C5A6
In3 på adress $C5A7

Adressavkodning

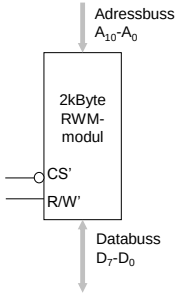
Om vi behöver flera I/O-portar ???



Adressavkodning



Att ansluta en 2-kbyte RWM- modul med startadress $4000

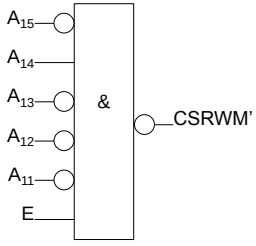


Arbetsgång:

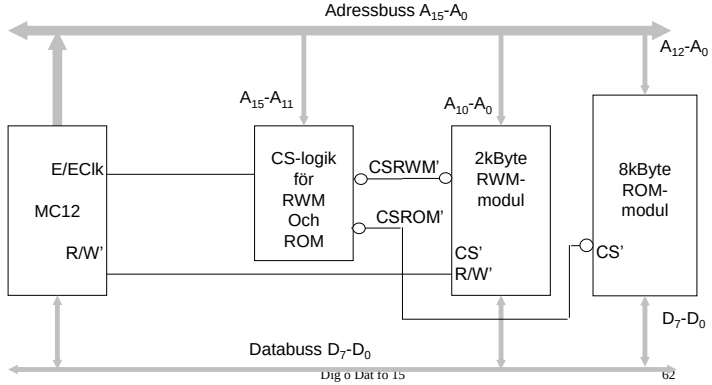
- ”Tolka” beskrivningen av minnesmodulen
- Rita tabell
- Ange modulens första adress
- Ange modulens sista adress
- Märk ut konstanta resp varierande adressledningar
- Rita adressavkodningslogiken

Att ansluta en 2-kbyte RWM-modul

S

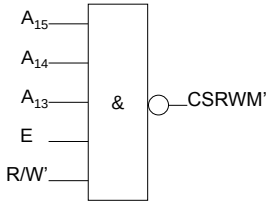


Att ansluta en 8-kbyte ROM- modul till ett befintligt system



Att ansluta en 8-kbyte ROM-modul

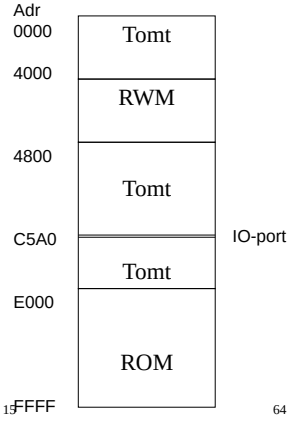
S



WRITE: R/W'= 0
READ: R/W'=1

Processorns adressrum

Beskrivande figur
över hur minnesmoduler
och IO-portar är placerade
i minnet



Uppgift

Konstruera adressavkodningen för följande:

4 kByte RWM från adress 0
8 kByte ROM på de högsta adresserna
En I/O-area på 256 Byte med start på adress $6000

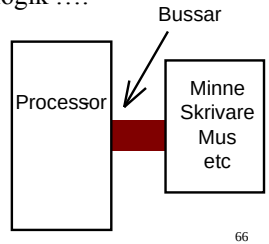
Du har tillgång till

8 kByte ROM-modul
4 kByte RWM-modul

Använd fullständig adressavkodning

Dagens mål: Du ska kunna….

- Skilja på olika minnestyper
- Förklara principer för olika bussprotokoll
- Beskriva MC12's bussar (anslutningar)
- Förstå begreppen Timing och VM (Valid Memory Adress)
- Konstruera adressavkodningslogik ….
  - 1) Fullständig Adr Avk
  - **2) Ofullständig Adr Avk**
  - Processorns adressrum



Uppgift

Som förra.... men använd ofullständig adressavkodning!

Konstruera adressavkodningen för följande:

4 kByte RWM från adress 0
8 kByte ROM på de högsta adresserna
En I/O-area på 256 Byte med start på adress $6000

Du har tillgång till

8 kByte ROM-modul
4 kByte RWM-modul

Uppgift

Konstruera adressavkodningen för följande:

En I/O-area på 512 Byte med start på adress $0000
RWM från I/O-areans slut till adress $1FFF
ROM från adress $2000-$FFFF

Du har tillgång till

64 kByte ROM-modul
8 kByte RWM-modul