Föreläsningsanteckningar
2. Mikroprogrammering I

Olle Seger 2012
Anders Nilsson 2016

Innehåll

1 Inledning **2**

2 En enkel dator **2**

3 Komponenter **3**
3.1 Register 3
3.2 Universalräknare 3
3.3 ALU = Arithmetic Logic Unit 4

4 Några styrenheter **4**

5 Programmerarmodell **6**
5.1 Begränsningar 6
5.2 Adresseringsmoder och instruktionslista. 6
5.3 Mikrogrammerarmodell 8

6 Mikroprogrammering **10**
6.1 Ett litet asm-program 10
6.2 Hämtfasen 10
6.3 Absolut 11
6.4 LDA 11
6.5 Omedelbar 11
6.6 ADD 12
6.7 STA 12
6.8 K1 och K2 12

7 Ytterligare mikroprogrammering **13**
7.1 Indirekt adressering 13
7.2 Underförstådd adressering 13

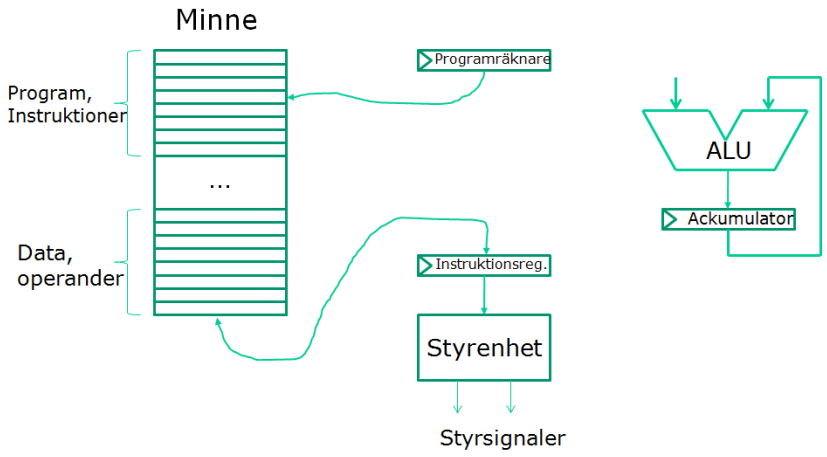
1 Inledning

Föreläsningen handlar om att bygga en enkel dator endast med kända komponenter från kursen i digitalteknik. Det visar sig att den mest komplexa delen är styrenheten. Vi väljer här att realisera styrenheten med hjälp av mikroprogrammering.

Källor för denna föreläsning är först och främst [1], men även [2, 3].

2 En enkel dator

Vi börjar med att dra oss till minnes, hur en enkel dator, se figur 1, är uppbyggd. Programräknaren pekar ut den instruktion som står i tur att exekveras. I minnet



Figur 1: En enkel dator.

finns också operander. Den utpekade instruktionen hämtas till instruktionsregistret. Innehållet i detta bestämmer en följd av styrsignaler, som genereras av styrenheten. Dessa styrsignaler styr alla delar av datorn.

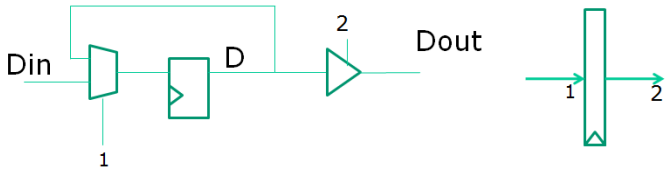
Viktigt att förstå är att datorn arbetar i två takter:

1. *hämta*. Instruktion hämtas till instruktionsregistret.
2. *utför*. Instruktionen utförs.

3 Komponenter

3.1 Register

Ett register är helt enkelt ett antal D-vippor med laddvillkor.



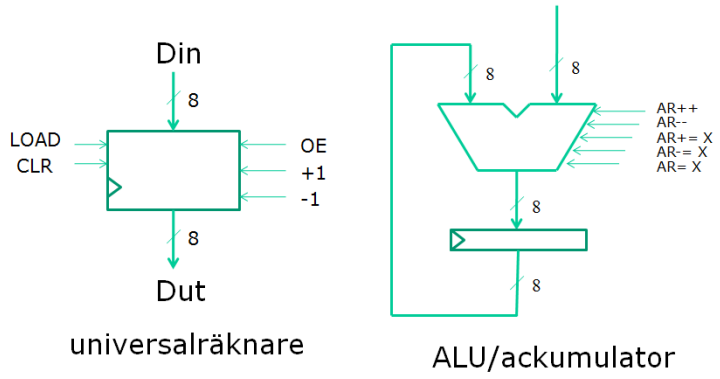
Registret fungerar enligt denna tabell:

1	2	Funktion
0	0	D = D, Dut = Z
1	0	D = Din, Dut = Z
0	1	D = D, Dut = D
1	1	D = Din, Dut = D

3.2 Universalräknare

Nedanstående universalräknare fungerar enligt denna tabell:

OE	CLR	LOAD	+1	-1	Funktion
0	-	-	-	-	Dut = Z
1	1	-	-	-	Dut = 0
1	0	1	-	-	Dut = Din
1	0	0	1	-	Dut = Dut+1
1	0	0	0	1	Dut = Dut+1

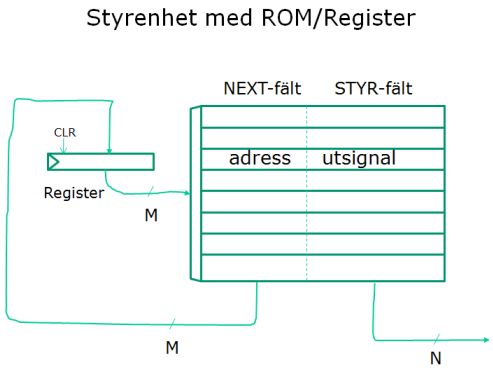


3.3 ALU = Arithmetic Logic Unit

Ovanstående figur (till höger) är en ALU med ackumulatorregister. Signalnamnen är självförklarande.

4 Några styrenheter

Nedanstående figur visar en enkel autonom styrenhet uppbyggd med ett ROM och ett register.

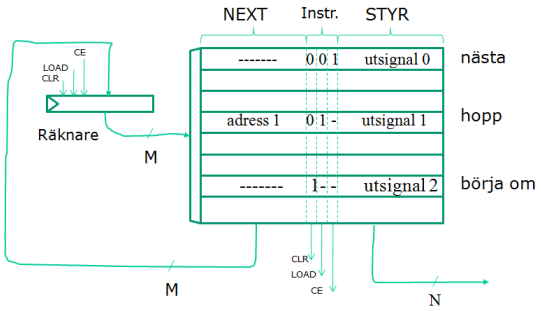


Innehållet i minnet är indelat i två fält

- *NEXT*-fält som anger nästa rad i minnet.
- *STYR*-fält som är styrsignalen.

Den uppmärksamme läsaren undrar måhända varför NEXT-fältet behövs. Det enda den här apparaten kan göra är ju att mata ut en förutbestämd följd av styrsignaler. Varför inte lägga styrsignalerna i ordning och byta ut registret mot en räknare?

Vi nappar delvis på detta resonemang och byter nu ut registret mot en universalräknare och får därmed några möjligheter till.

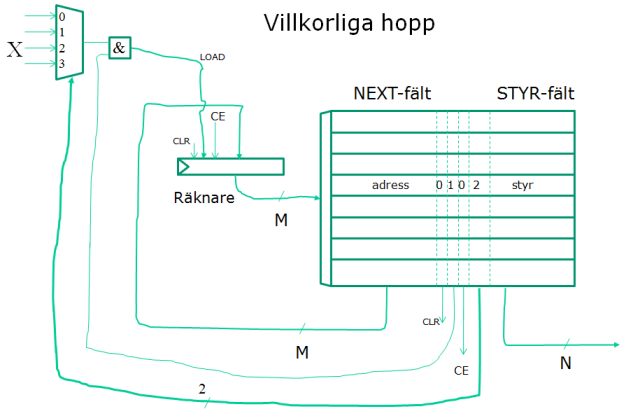


Ett fält har tillkommit

- *Instruktions*-fält som styr signalerna CLR, LOAD och CE på räknaren.

Innehållet i detta fält kan betraktas som en hoppinstruktion och innehållet i minnet kallas mikroprogram.

Nu kompletterar vi styrenheten med insignaler och villkorliga hopp.



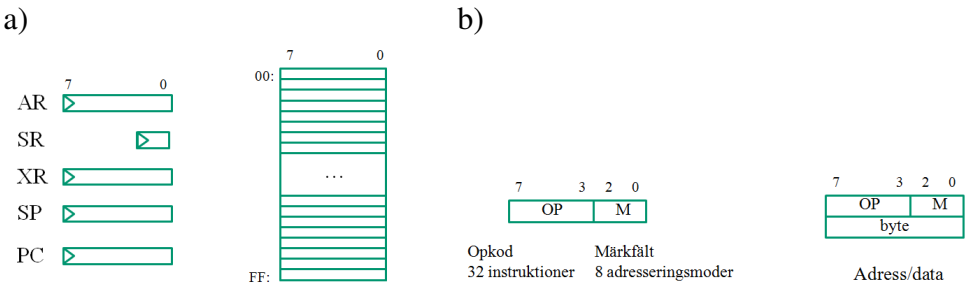
Instruktionen i mikrominnet har följande funktion: om insignalen $X = 1$ så fås ett hopp till `adress` annars står vi kvar på samma rad. Uttryckt med andra ord väntar mikroprogrammet på denna rad tills $X = 1$.

5 Programmerarmodell

5.1 Begränsningar

För att datorn inte ska bli för svår att bygga inför vi en del begränsningar se figur 2:

- endast 8-bitars register. Detta är ju förstås ganska orealistiskt för pekarregistren eftersom det medför att vi endast kan ha $2^8 = 256$ minnesceller.
- endast två instruktionsformat. En instruktion består av
  - en byte för OPkod (5 bitar) och adresseringsmod (3 bitar). Detta räcker alltså till 32 opkoder och 8 adresseringsmoder. Inte helt orealistiskt!
  - eventuellt en extra byte, som innehåller ett tal, en adress eller en förskjutning.



Figur 2: Begränsningar i den realiserade datorn. a) endast 8-bitars register i programmerarmodellen b) endast 2 instruktionsformat.

5.2 Adresseringsmoder och instruktionslista.

Vi kodar adresseringsmoderna på följande sätt:

M 3 bitar	Adresseringsmod	EA
0	absolut	A
1	indirekt	M(A)
2	indexerad	XR+A
3	relativ	PC+2+d
4	omedelbar	PC+1
5	underförstådd	

Vi väljer att ta med följande instruktioner:

OPkod 5 bitar	Mnemonics	Verkan	Adresseringsmod	flaggor ZN
0	LDA	AR=M(EA)	absolut, indirekt, index, omedelbar	ZN
1	STA	M(EA)=AR	absolut, indirekt, index	-
2	ADDA	AR=AR+M(EA)	absolut, indirekt, index, omedelbar	ZN
3	SUBA	AR=AR-M(EA)	absolut, indirekt, index, omedelbar	ZN
4	INCA	AR=AR+ 1	underförstådd	ZN
5	DECA	AR=AR-1	underförstådd	ZN
6	LDX	XR=M(EA)	absolut, indirekt, index, omedelbar	ZN
7	STX	M(EA)=XR	absolut, indirekt, index	–
8	INX	XR=XR+ 1	underförstådd	ZN
9	DEX	XR=XR-1	underförstådd	ZN
A	LDS	SP=M(EA)	absolut, indirekt, index, omedelbar	ZN
B	STS	M(EA)=SP	absolut, indirekt, index	–
C	INS	SP=SP+1	underförstådd	ZN
D	DES	SP=SP-1	underförstådd	ZN
E	PUSH	M(SP)=AR, SP=SP-1	underförstådd	–
F	PULL	SP=SP+ 1, AR=M(SP)	underförstådd	ZN
10	JMP	PC=PC+2+D	relativ	–
11	JMPN	Om N=1 PC=PC+2+D annars PC=PC+2	relativ	–
12	JMPZ	Om Z=1 PC=PC+2+D annars PC=PC+2	relativ	–
13	JSR	M(SP)=PC+2,SP=SP-1 PC=PC+2+D	relativ	–
14	RTS	SP=SP+ 1, PC=M(SP)	underförstådd	-

Vi har här valt att särbehandla hoppen. Hoppen använder endast relativ mod. Inga andra instruktioner använder denna mod. Detta är ju förstås en fix som kommer att förenkla våra mikroprogramerarmödor.

Instruktionen `LDA 8` kodas:

00000	000
0000 1000	

till skillnad från instruktionen `LDA #8` som kodas:

00000	100
0000 1000	

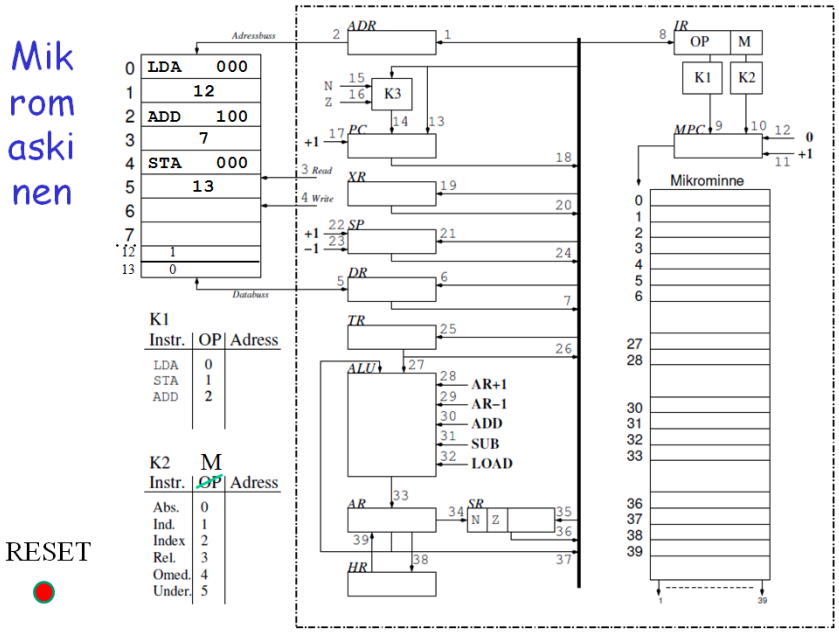
5.3 Mikrogrammerarmodell

Vi är nu redo att ge oss på mikroprogrammerarmodellen i fig. 3. Vi anser att den består av tre större byggblock:

- *primärminne* med adressregistret `ADR`.
- *databehandlingsenhet* med de registret, som ingår i (asm)-programmerarmodellen. Dessutom behövs hjälpregistret `HR` och det temporära registret `TR`
- *styrenhet* med mikroprogramräknaren `MPC`, instruktionsregistret `IR`, mikrominne och två look-up tables `K1` och `K2`.

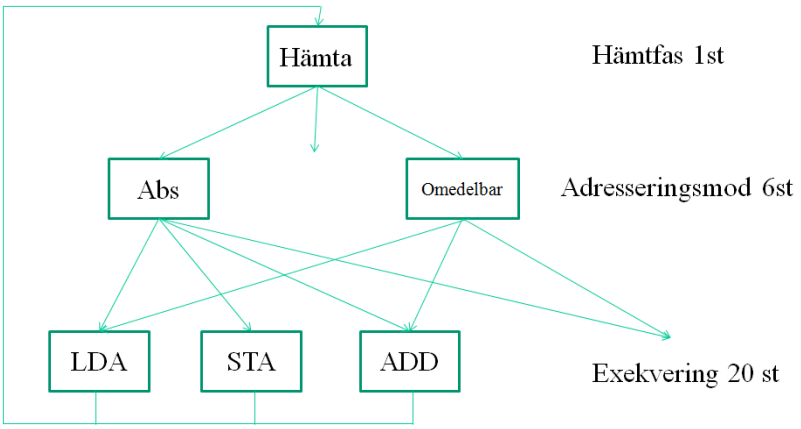
Innan vi ger oss på att skriva mikroprogram, konstaterar vi att vår dator arbetar i tretakt:

1. *hämtfas*. En instruktion hämtas till instruktionsregistret `IR`. Instruktionen pekas ut av `PC`. Innan vi lämnar denna är det alltså lämpligt att räkna upp `PC` och därefter hoppa till
2. *adressmodsfas*. Här kommer vi att behöva 6 olika mikroprogramsnuttar. Gemensamt för dessa är att operandens EA beräknas och placeras i `ADR`. Ett undantag finns dock, hoppadresser placeras i `TR`. Detta är möjligt eftersom hoppen (och endast hoppen) använder relativ adresseringsmod. Om vi har hämtat en byte från minnet räknar vi upp `PC` igen. Sedan sker hopp till
3. *exekveringsfas*. Här behövs lika många snuttar som instruktioner. Observera nu att dessa rutiner inte känner till vilken adresseringsmod, som använts i den föregående fasen. Undantaget är hoppen. Alltså, EA finns i `ADR`. Hämta operanden och gör något med den. Därefter hopp till rad 0 i mikroprogrammet.



Figur 3: Datormodell för mikroprogrammeraren.

Mikroprogrammet kommer alltså att vara organiserat så här:



6 Mikroprogrammering

6.1 Ett litet asm-program

Låt oss som ett exempel skriva mikrokod som kan exekvera detta enkla asm-program. Programmet befinner sig i primärminnet (M). Vi lägger märke till att instruktionerna `LDA` och `STA` använder absolut adresseringsmod och `ADD` omedelbar adresseringsmod.

```
 0: LDA 12      ; M(12) ->
 2: ADD #7      ; AR=AR+7
 4: STA 13      ; M(13)
    ...
12: 1
13: 0
```

När asm-programmet har exekverat bör alltså innehållet i minnescell vara 8, eftersom $M(12) + 7 = 1 + 7 = 8 -> M(13)$.

6.2 Hämtfasen

När vi har tryckt på reset är läget följande:

Hämtfasens uppgift är nu att hämta den instruktion, som pekas ut av PC, till IR. Vi kan sammanfatta detta så här:

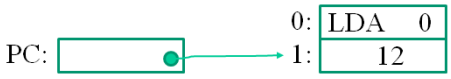
```
M(PC++) -> IR
```

Mikrokod för hämtfasen:

```
0: pc->adr, mpc++                    18,1,11
1: adr->PM,PM->dr,mpc++              2,3,5,11
2: dr->ir,mpc++                      7,8,11
3: PC++, K2->mpc                     17,10
```

6.3 Absolut

PC pekar nu på byten efter instruktionen.



Byten är adressen (EA) till operanden. Detta ska alltså göras:

```
M(PC++) -> ADR
```

Mikrokod:

```
 4: pc->adr,mpc++                    18,1,11
 5: PM->dr,mpc++                     2,3,5,11
 6: dr->adr,K1->mpc,PC++             7,1,9,17
```

6.4 LDA

ADR pekar ut operanden. Alltså:

```
M(ADR) -> AR
```

Mikrokod:

```
30: PM->dr,mpc++                     2,3,5,11
31: dr->tr,mpc++                     7,25,11
32: tr->ar,mpc++                     27,32,33,11
33: status, 0->mpc                   34,12
```

6.5 Omedelbar

Operanden pekas ut av PC!

Alltså:

```
PC++ -> ADR
```

Mikrokod:

```
27: pc->adr, mpc++                   18,1,11
28: pc++, K1->mpc                    17,9
```

6.6 ADD

ADR pekar ut operanden. Alltså:

```
M(ADR) + AR -> AR
```

Mikrokod:

```
36: PM->dr,mpc++          2,3,5,11
37: dr->tr, mpc++         7,25,11
38: ar+tr->ar,mpc++         27,33,30,11
39: status, 0->mpc         34,12
```

6.7 STA

ADR pekar ut operanden. Alltså:

```
AR -> M(ADR)
```

Mikrokod:

```
34: ar->dr,mpc++                        37,6,11
35: adr->PM,dr->PM,0->mpc               2,4,5,12
```

6.8 K1 och K2

Vi har skrivit mikrokod för 3 instruktioner. Innehållet i K1 blir följande:

instr	OP	adress
LDA	0	30
STA	1	34
ADD	2	36

Vi har också skrivit mikrokod för 2 adresseringsmoder. Innehållet i K2 blir följande:

a-mod	M	adress
absolut	0	4
...		
omedelbar	4	27

7 Ytterligare mikroprogrammering

7.1 Indirekt adressering

Instruktion `LDA (D)` fungerar så här:

```
M(M(D)) -> AR,
```

dvs byten efter opkoden pekar på ett ställe i minnet som innehåller adressen till operanden. Mikrokod för indirekt adresseringsmod:

```
 7: pc->adr, pc++, mpc++              18,1,17,11
 8: adr->PM->dr, mpc++                2,3,5,11
 9: dr->adr, mpc++                    7,1,11
10: adr->PM->dr, mpc++                2,3,5,11
11: dr->adr, k1->mpc                  7,1,9
```

7.2 Underförstådd adressering

Instruktionen `INCA` fungerar så här:

```
AR+1 -> AR,
```

dvs operanden är ackumulatorn själv. Här går det faktiskt att hoppa över adresseringsmoden genom att lägga exekveringsfasens adress direkt i K2.

Om vi gör på det tänkta sättet blir mikrokoden för underförstådd adresseringsmod:

```
29: k1->mpc                   9
```

Mikrokod för `INCA`:

```
44: AR+1->AR, mpc++           27,28,33,11
45: status, 0->mpc            34,12
```

Referenser

[1] Roos O. (1995): *Grundläggande datorteknik*, Studentlitteratur.

[2] Josefsson M. (2008): *Föreläsningsunderlag TSEA49 Datorteknik D del 2*, LiU-tryck.

[3] Danielsson P-E och Bengtsson L. (1996): *Digital teknik*, Studentlitteratur.