Föreläsningsanteckningar
3. Mikroprogrammering II

Olle Seger 2012
Anders Nilsson 2016

1 Inledning

Datorn, som vi byggde i förra föreläsningen, har en stor brist. Den saknar I/O. I denna föreläsning kompletterar vi datorn. Vi tittar också närmare på hur avbrott implementeras.

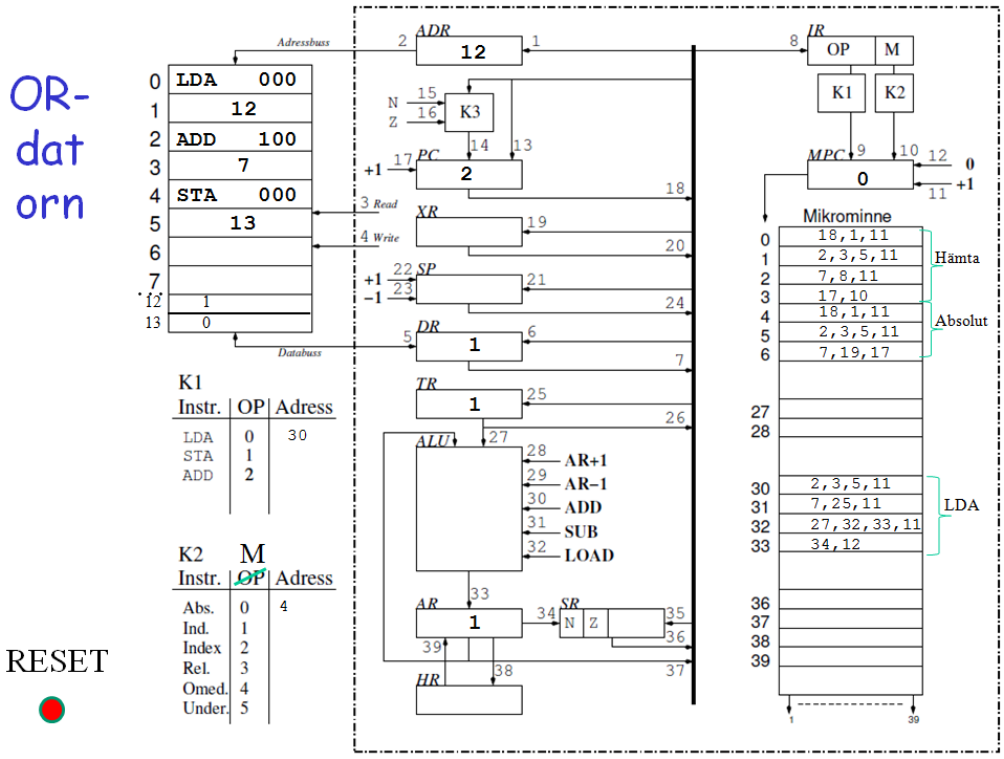
Källor till föreläsningen är [1, 2].

2 Repetition

Vi börjar med att åter ta oss en titt på Olle Roos-datorn. I mikrominnet finns kod för att exekvera instruktion `LDA 12`, nämligen

- *hämtfas*, som hämtar instruktionen.
- *absolut*, som beräknar EA för absolut adresseringsmod.
- *LDA*, som exekverar *LDA*.

Uppslagstabellerna K1 och K2 är ifyllda med startadresserna för *LDA* och *absolut*, respektive.

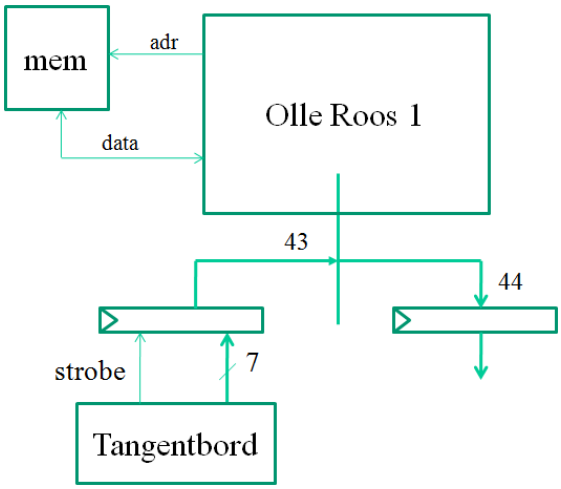


Figuren visar tillståndet efter att den första instruktion har exekverats. Vi ser att allt verkar stämma:

- *PC=2*, nästa instruktion pekas ut.
- *AR=1*, `AR` har laddats med innehållet i cell 12.
- *ADR=12*, dvs den senaste minnesaccessen.
- *MPC=0*, dvs en hämtfas startar nu.

3 I/O

Det enklaste sättet att installera I/O på vår dator är att koppla in 2 register på den centrala bussen, enligt nedanstående figur.



Vi behöver nu två nya instruktioner:

- *IN*, som kopierar IN-registret till ackumulatorn, IN->AR.
- *OUT*, som kopierar ackumulatorn till OUT-registret, AR->OUT.

Mikrokoden för dessa instruktioner visas här:

```
IN
    50: in->tr, mpc++     43,25,11
    51: tr->ar, mpc++     27,32,33,11
    52: status, 0->mpc    34,12

OUT
    53: ar->out, 0->mpc     37,44,12
```

Båda instruktioner använder underförstådd adresseringsmod (saknar adresseringsmod):

```
UNDERFÖRSTÅDD
  29: K2->mpc          9
```

Vi kan nu skriva ett asm-program, som läser in tecken från tangentbordet. Tecknen placeras i en buffert i minnet. Lägg märke till att tangentbordet ger ifrån sig 7 bitar och en strobe, som signalerar att en knapp är nedtryckt. Vi är listiga och ansluter stroben till MSB. Vi använder några instruktioner, som vi inte har mikrokod för ännu.

```
READ:
      IN          ; hämta tecken+strobe
      BPL  READ   ; om strobe=0, hopp till READ
      STA  0(X)   ; lägg i buffert
      INX         ; X++
      ...
```

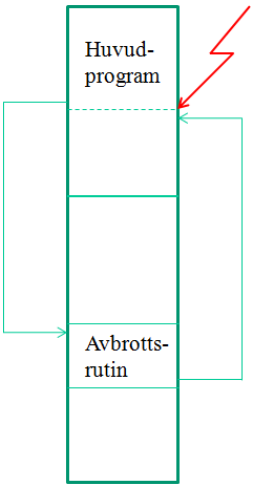
Koden är förstås ett exempel på polling. Programmet kommer att loopa tills någon trycker på tangentbordet. Det kan väl vara OK om datorn inte har något annat för sig.

Man brukar säga att det finns tre olika sätt att arrangera I/O:

- *polling*. Programmet väntar på att stroben ska bli hög, läser tecknet och placerar i minnet. Kallas också programstyrd I/O, busy waiting.
- *Avbrott*. Programmet behöver inte alls vänta på stroben. När stroben går hög startar en avbrottsrutin, som läser in tecknet och placerar i bufferten. Bättre!
- *DMA*. I/O kretsen skriver själv (genom att ta över lämpliga bussar) i minnet. DMA = direkt minnesaccess. Programmet behöver bara uppmärksammas när return har kommit in (avbrott!). DMA ske genom att koppla bort CPUn från bussarna eller genom att utnyttja lediga minnescykler. Bäst!

4 Avbrott

Avbrott innebär i korthet att en yttre signal avbryter ett huvudprogram och istället körs en avbrottsrutin. När den kört klart (och den ska vara kort) återupptas huvudprogrammet:



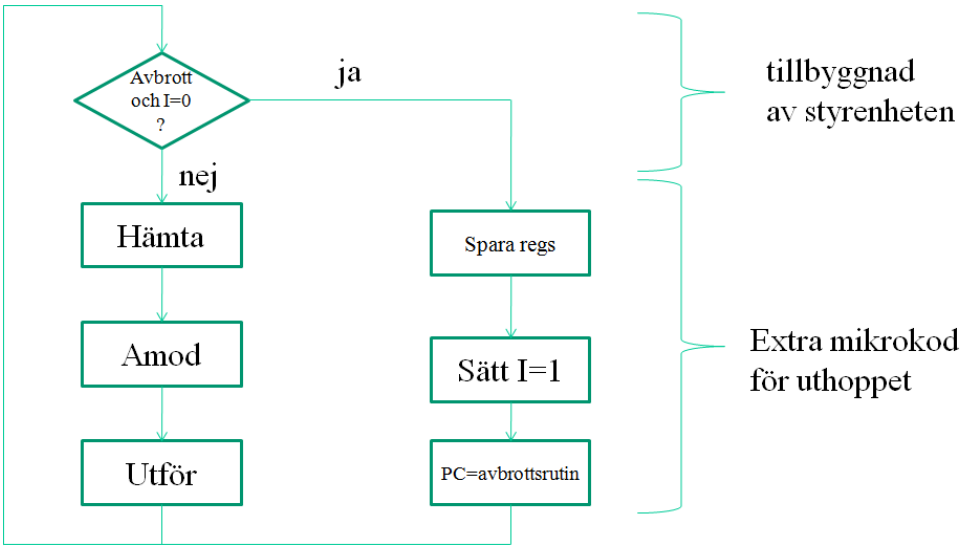
Vi vill att det ska fungera så här:

1. Gör klart pågående instruktion
2. Spara undan processorns tillstånd på stacken, dvs PC, SR, XR, AR
3. Förhindra fler avbrott genom att sätta en spärrvippa I.
4. Hoppa till avbrottsrutinen
5. Sist i avbrottsrutinen, hoppa tillbaka till huvudprogrammet.

Vår mikroprogrammerade dator måste nu kompletteras på olika ställen:

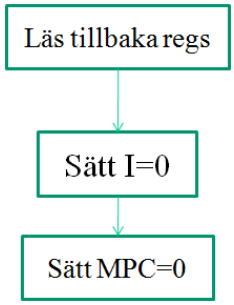
- *hårdvara*. Mikroprogramräknarens (MPC) styrning (nästa adress) måste kompletteras.
- *mikroprogram*. Uthoppet till avbrottsrutinen. Observera att detta inte är en instruktion.
- *hårdvara*. Spärrvippan I, alltså avbrott på/av
- *hårdvara*. Programräknarens (PC) styrning (nästa adress) måste kompletteras.
- *mikroprogram*. Tre nya instruktioner, EI (enable interrupt), DI (disable interrupt) och RTI (return from interrupt).

Figuren visar flödet för mikroprogrammet efter kompletteringen:

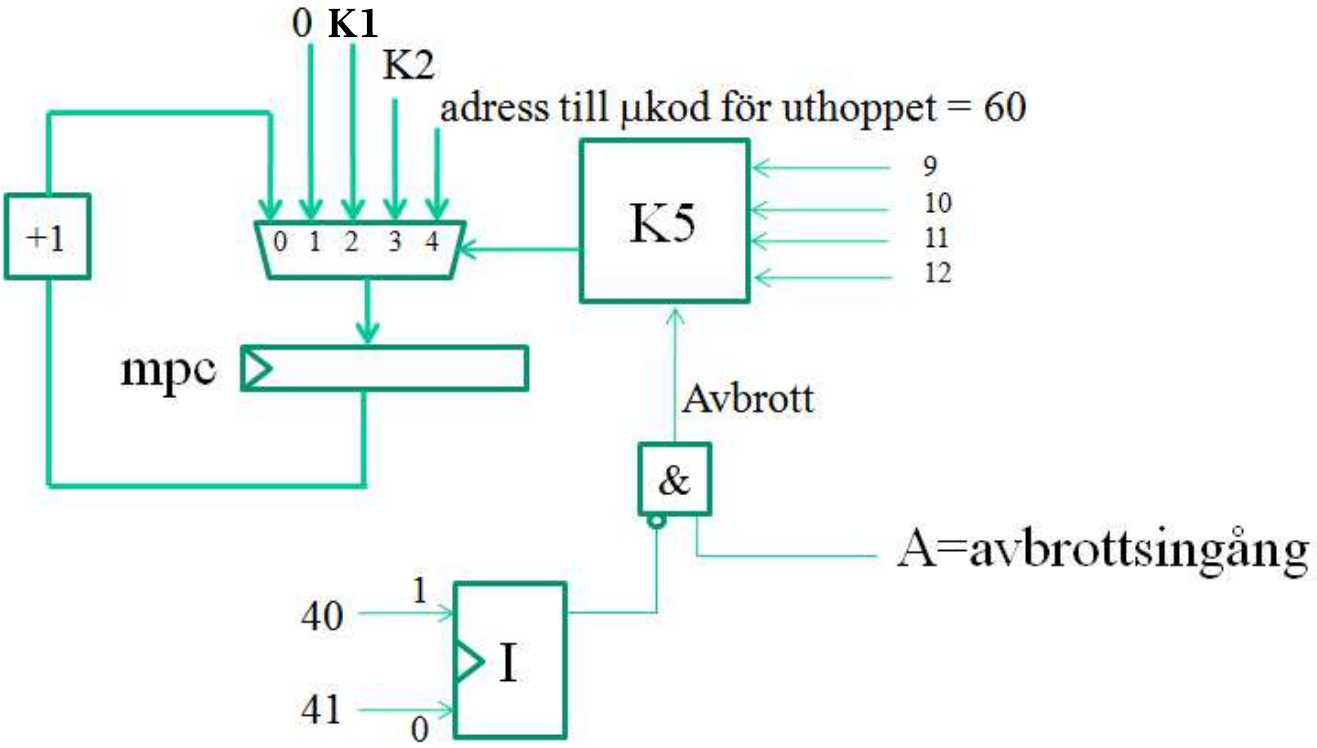


Boxarna är mikroprogram och valet är multiplexerstyrning vid MPC.

En ny instruktion `RTI` behövs. Den gör motsatsen till uthoppet, nämligen



Nästa figur figur visar ombyggnationerna runt MPC:



Det centrala i figuren är multiplexern, som väljer nästa adress för MPC. Här har tillkommit adressen för uthoppsmikroprogrammet. Signalerna 9,10,11 och 12 kommer från mikrominnet. Vi kommer ihåg att signalen 12 aktiveras på sista raden i exekveringsfasen, dvs vi är mellan två instruktioner.

Vi vill nu att avbrott ska ske om signalen 12 är hög, avbrottssignalen A är hög och spärrvippan I är låg. I övrigt ska det fungera som förut.

Det kombinatoriska nätet K5 ska alltså innehålla

9	10	11	12	avbr	hopp
1	0	0	0	-	2
0	1	0	0	-	3
0	0	1	0	-	0
0	0	0	1	0	1
0	0	0	1	1	4

Mikroprogrammet för uthoppet. Här läggs i tur och ordning PC, XR, AR och SR på stacken.

```
60: sp->adr, mpc++              24,1,11
61: pc->dr, mpc++               18,6,11
62: skriv, sp--,mpc++           2,4,5,23,11

63: sp->adr, mpc++              24,1,11
64: xr->dr, mpc++               20,6,11
65: skriv, sp--,mpc++           2,4,5,23,11

66: sp->adr, mpc++              24,1,11
67: ar->dr, mpc++               37,6,11
68: skriv, sp--,mpc++           2,4,5,23,11

69: sp->adr, mpc++              24,1,11
70: sr->dr, mpc++               36,6,11
71: skriv, sp--,mpc++           2,4,5,23,11

72: I=1,mpc++                   40,11
73: avbrvek->PC, mpc=0          42,12
```

Ett ganska långt mikroprogram som synes. Det består av fyra nästan lika avsnitt och lite krafs på slutet. Som mikroprogrammerare bör man nu ställa sig frågan om detta går att göra på färre rader, vilket ju förstås är samma sak som färre klockcykler. Ett sätt att tänka är att försöka göra mer på varje rad och att lägga allting så tidigt som möjligt.

Om vi tittar på början av programmet ser vi att `sp--` kan flyttas upp från rad 62 till 60. Det medför att hela rad 63 kan bakas in i rad 62. Så här alltså:

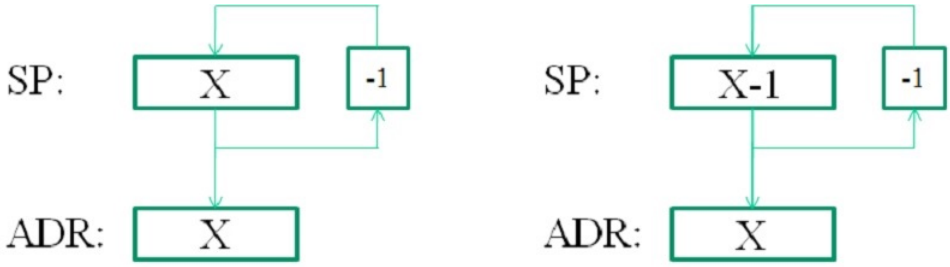
```
60: sp->adr, mpc++              60: sp->adr, sp--, mpc++
61: pc->dr, mpc++               61: pc->dr, mpc++
```

```
62: skriv, sp--, mpc++          62: skriv, sp->adr, mpc++

63: sp->adr, mpc++
...
```

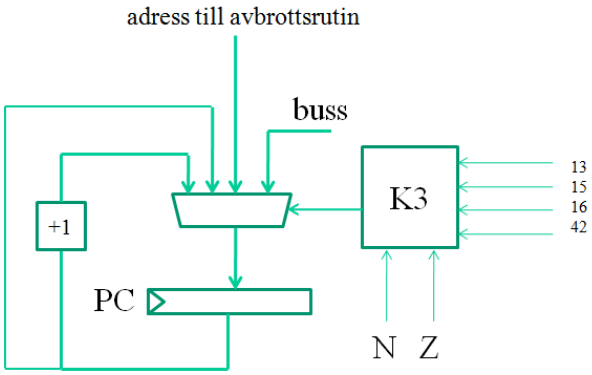
Genom att överlappa på detta sätt sparar vi 3 klockcykler. I slutet av programmet ser vi att rad 71,72 0ch 73 kan slås ihop. Det kanske finns mer att göra?

Observera att vår nya rad 60 innebär detta



som är en vanligt sätt att spara en cykel.

Till slut förtydligar vi hur hårvaran run PC är uppbyggd:



Adress till avbrottsrutin avser här en adress i primärminnet, där asm-programmet finns.

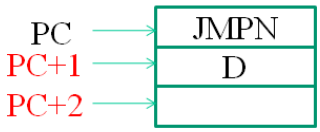
Nätet K3 har följande sanningstabell:

13	15	16	42	N	Z	PC
0	0	0	1	-	-	vektor
1	0	0	0	-	-	BUSS
0	1	0	0	0	-	PC
0	1	0	0	1	-	BUSS
...						

5 Ett villkorligt hopp

Det villkorliga hoppet `JMPN D` fungerar enligt
om N=1 så PC+2+D->PC annars PC+2->PC.

Hoppadressen räknas alltså relativt adressen för nästa instruktion.



Här behövs kod för självrelativ a-mod (K2(3)=12):

```
19: PC->ADR, PC++, MPC++
20: PC->TR, minne->DR, MPC++
21: DR->TR, TR->AR, AR->HR, MPC++
22: AR+TR->AR, MPC++
23: HR->AR, AR->TR, K1->MPC
```

och för själva hoppet (K1(17)=78)

```
78: TR->PC(N), 0->MPC          26,15,12
```

6 Indexerad adressering

En mycket matnyttig adresseringsmod är indexering. Hos RISC-processorer, där man skalar bort onödigt jox, brukar den vara den enda a-moden!

Instruktion `STA D(X)` fungerar så här:
AR -> M(XR+D).

Här behövs kod för indexerad a-mod (K2(3)=19): Vi hämtar denna kodsnutt från [1]:

```
12: XR->TR, AR->HR, mpc++
13: PC->ADR, TR->AR, mpc++
14: M->DR, mpc++
15: DR->TR, mpc++
16: AR+TR->AR, mpc++
17: AR->ADR, mpc++
18: PC++, HR->AR, K1->MPC
```

alltså 7 klockcykler. Vi kan inte låta bli att förbättra lite:

```
12: PC->ADR, PC++, mpc++
13: M->DR, XR->TR, mpc++
14: DR->TR, TR->AR, AR->HR, mpc++
15: AR+TR->AR, mpc++
16: HR->AR, AR->ADR, K1->mpc
```

Observera noga vad rad 14 gör.

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.