LABORATION

Datorteknik Y

Mikroprogrammering

Version 3.8
2013 (AE)
2024 (KP)

1 Inledning

Syftet med laborationen är att skapa en känsla för vad som händer i en enkel dator då en maskinkodsinstruktion (även kallad assemblerinstruktion) exekveras. Syftet är också att exemplifiera några av de byggblock som en processor kan vara uppbyggd av. Efter genomförd laboration ska du ha en grundförståelse för hur en enkel dator fungerar och hur mikrokod och maskinkod tillsammans kan användas för att skapa en (relativt) hårdvaruoberoende programmeringsmodell för en assemblerprogrammerare.

Den dator som används i laborationen är helt mikroprogrammerbar och finns beskriven i häftet "Beskrivning av MIA-systemet". I det här laborationerna kommer du att få skriva all mikrokod som behövs för att kunna exekvera ett antal maskinkodsinstruktioner. Du kommer även att få skriva några maskinspråksprogram för att testa din konstruktion.

2 Praktiska råd

Precis som i all annan mjukvaruutveckling är det bra om du kan utveckla ditt program i små delar och testa dessa delar var för sig. Det är givetvis också lämpligt att spara undan gamla versioner så att du kan gå tillbaka till dessa om du av misstag skulle radera dina fungerande program.

Tänk också på att laborationen är ett examinationsmoment, det är alltså viktigt att båda personerna i en laborationsgrupp har förstått alla moment i laborationen.

Det är ett **krav** att mikroprogrammet är dokumenterat (det räcker inte att bara skriva in värden i lmia). Varje rad som används i mikrokodsminnet ska ha en kort beskrivning av vad raden gör. Denna dokumentation kan göras i excel, på papper, i word eller på annat lämpligt sätt.

3 Laborationen

Innan första labtillfället ska detta lab-PM vara noggrannt genomläst. Likaså bör de kapitel i kurslitteraturen som berör mikroprogrammering vara genomlästa. Till hjälp vid mikroprogrammeringen finns speciella blanketter sist i detta PM. På dessa skriver man styrordets utseende i binär form. Om så önskas kan man också översätta talet till hexadecimal form, eftersom det går fortare att mata in mikrokodsprogram hexadecimalt i LMIA-systemet. Motsvarande blankett finns också som excelfil.

Uppgift 1

Skriv mikroprogrammet för följande maskininstruktioner:

Instruktion	Betydelse	Adresserings-moder	Påverkar flaggor
`LOAD GRx,M,ADR`	GRx := PM(A)	00,01,10,11	-
`STORE GRx,M,ADR`	PM(A) := GRx	00,10,11	-
`ADD GRx,M,ADR`	GRx :=GRx+PM(A)	00,01,10,11	Z,N,O,C
`SUB GRx,M,ADR`	GRx :=GRx-PM(A)	00,01,10,11	Z,N,O,C
`AND GRx,M,ADR`	GRx :=GRx and PM(A)	00,01,10,11	Z,N
`LSR GRx,M,Y`	GRx skiftas logiskt höger Y (ADR-fältet) steg	- (ange 00)	Z,N,C utskiftad bit
`BRA ADR`	PC :=PC+1+ADR	- (ange 00)	-
`BNE ADR`	PC := PC+1+ADR om Z=0, annars PC:= PC+1	- (ange 00)	-
`HALT`	avbryt exekv.	- (ange 00)	-

Kommentarer:
PM(A) är minnesplatsen i PM som bestäms av M och ADR i instruktionen.
För LSR är lagras Y i ADR-fältet i instruktionen. Y=0 behöver inte fungera.
PC+1 i hoppinstruktionerna avser den uppräkning som görs av PC i hämtfasen.

Hämtfas

Börja med att skriva mikroprogrammet för instruktionshämtningen. Det ska börja i mikroadress 00 eftersom exekveringen av en maskininstruktions mikroprogram alltid avslutas med att mikroprogramräknaren nollställs. Hämtrutinen ska göra följande:

```
uM adress       utför
00              ASR:= PC
01              IR:= PM, PC:= PC+1
```

Effektivadressberäkning

Efter instruktionshämtningen ska instruktionens effektivadress beräknas och läggas i ASR. Detta görs för alla instruktioner trots att det inte behövs för t ex BRA, som alltid är relativadresserande, eller LSR som inte berör minnet. Det behövs en mikroprogramsekvens för var och en av de fyra olika adresseringssätten. Vilken sekvens som ska utföras bestäms av M-fältet via nätet K2.

Man får alltså:

```
uM adress       utför
02              uPC:= K2(M-fältet)
```

För de fyra adresseringslägen som finns i datorn behövs följande mikrokod:

```
uM adress       utför
03              ASR:=IR, uPC:= K1(OP-fältet) ; Direktadressering
```

```
04              ASR:=PC, PC:= PC+1, uPC:= K1(OP-fältet) ; Immediate

05              ASR:= IR                        ; Indirekt adressering
06              ASR:= PM, uPC:= K1(OP-fältet)
                ; Se kommentar i Uppgift 3 om indirekt adressering!

07              AR:= IR                         ; Indexerad adressering
08              AR:= GR3+AR
09              ASR:= AR, uPC:= K1(OP-fältet)
```

Observera att K2 också måste programmeras på följande sätt:

Adress 0	anger startadress för direktadresseringsprogrammet
Adress 1	dito för omedelbar operand
Adress 2	dito för indirektadressering
Adress 3	dito för indexerad adressering

Minnesinnehållet ges av de olika sekvensernas adresser enligt ovan. Exekveringssekvenserna för instruktionerna kan läggas på godtycklig plats i mikrominnet då hoppadressen till respektive sekvens anges i K1. Det är dock lämpligt att placera dessa sekvenser omedelbart efter adressberäkningen.

Tips: M-fältet i instruktionen kommer alltid att vara satt till 11 när indexerad adressering används. För att få fram GR3 kan du alltså använda S-biten.

Förberedelseuppgift: Det finns två olika additionsinstruktioner i ALU:n. Vilken av dessa ska användas på adress 8 i mikrominnet för att utföra AR:=GR3+AR och varför? Skriv kommentar i excelbladet vid redovisningen.

Uppgift 2

På adress 0xFE i minnet finns fyra stycken fyra bitar breda tal lagrade enligt nedan:

A	B	C	D

Skriv ett assemblerprogram som använder de instruktioner du implementerade i uppgift 1 som räknar ut värdet A + B + C + D och lagrar detta på position 0xFF i minnet.

Exempel: Om värdet 0x53AF finns på på adress 0xFE så ska 0x0021 skrivas in på adress 0xFF.

Uppgift 3

Skriv ett program som sorterar en lista med tvåkomplementstal. Listan börjar på adress 0xE0 och slutar på adress 0xFF. När programmet körts klart ska listan vara sorterad och det minsta värdet ska finnas på adress 0xE0. Tabell 1 visar ett exempel på hur ditt program ska bete sig. Denna tabell kan skapas i lmia med hjälp av kommandot "load sort data" i redigera-menyn. Glöm inte att tabellen också kan innehålla flera likadana värden.

Prestandakrav: Din lösningen ska kräva maximalt 100 000 klockcykler för att sortera exempellistan.

Du får implementera sorteringen på valfritt sätt, men det lättaste sättet att göra detta är antagligen genom att använda *bubble sort* enligt följande algoritm[1]

1. För varje element i listan:
    - Jämför detta element med nästa element.
    - Om nästa element är mindre (om uppgiften hade krävt sorterad lista i fallande ordning hade testen istället varit större än) än det nuvarande elementet så byter du plats på dessa.
2. Gå igenom listan om och om igen tills du inte längre behöver byta plats på några element. Då vet du att listan är sorterad.

Tips: Det blir antagligen lättare att skriva ditt sorteringsprogram om du implementerar en eller flera av följande instruktioner:

- `CMP` (samma som `SUB` förutom att GRx inte uppdateras)
- `BGE` (hoppa om `GRx` var större än eller lika med `PM(A)` i `CMP`-instruktionen) (båda talen använder tvåkomplement!)
- `BEQ` (hoppa om Z-flaggan är satt)

Du får dock, om du vill, implementera helt andra instruktioner om det passar ditt sorteringsprogram bättre. (Du kan exempelvis tänkas ha nytta av stöd för en stack genom PUSH/POP eller JSR/RTS, eller skapa en instruktion som både testar och byter plats på minnesvärden om det behövs.) Du får även, om du vill, byta ut adresseringsläget indirekt adressering mot något annat om du tycker att det skulle passa din algoritm bättre! Det ska dock minst finnas en instruktion i PM som hämtas och avkodas. Se även Appendix 1 för ett lämpligt flödesschema. Notera att jämförelsen där är större än (inte större än eller lika med).

[1]Det är värt att notera att Bubble-sort är en tämligen usel sorteringsalgoritm. Det finns dock några ganska enkla sätt att snabba upp den, men för de som är ute efter att skriva en riktigt snabb sorteringsalgoritm så rekommenderar jag er att antingen läsa i en lärobok om datastrukturer och algoritmer, alternativt att ta en titt på Wikipedia-sidan om *"Sorting algorithm"*. (Däremot så är det inte säkert att de mest avancerade algoritmer går att anpassa till LMIA så att de är bättre än enklare algoritmer i de fall då enbart 32 element ska sorteras.)

Tävling

Det prestandakrav som finns i denna laboration är väldigt lätt att nå, men för att uppmuntra er till att försöka skriva snabba program så kommer en highscore-lista publiceras över de labgrupper som lyckas skriva den snabbaste sorteringsalgoritmerna. Om ni vill vara med i denna tävling så ska ni maila in er lösning (som ni sparat ifrån LMIA) till examinator. Se kurshemsidan för information om deadlines för tävlingen. Skriv Tävling och mikrokod i ärenderaden.

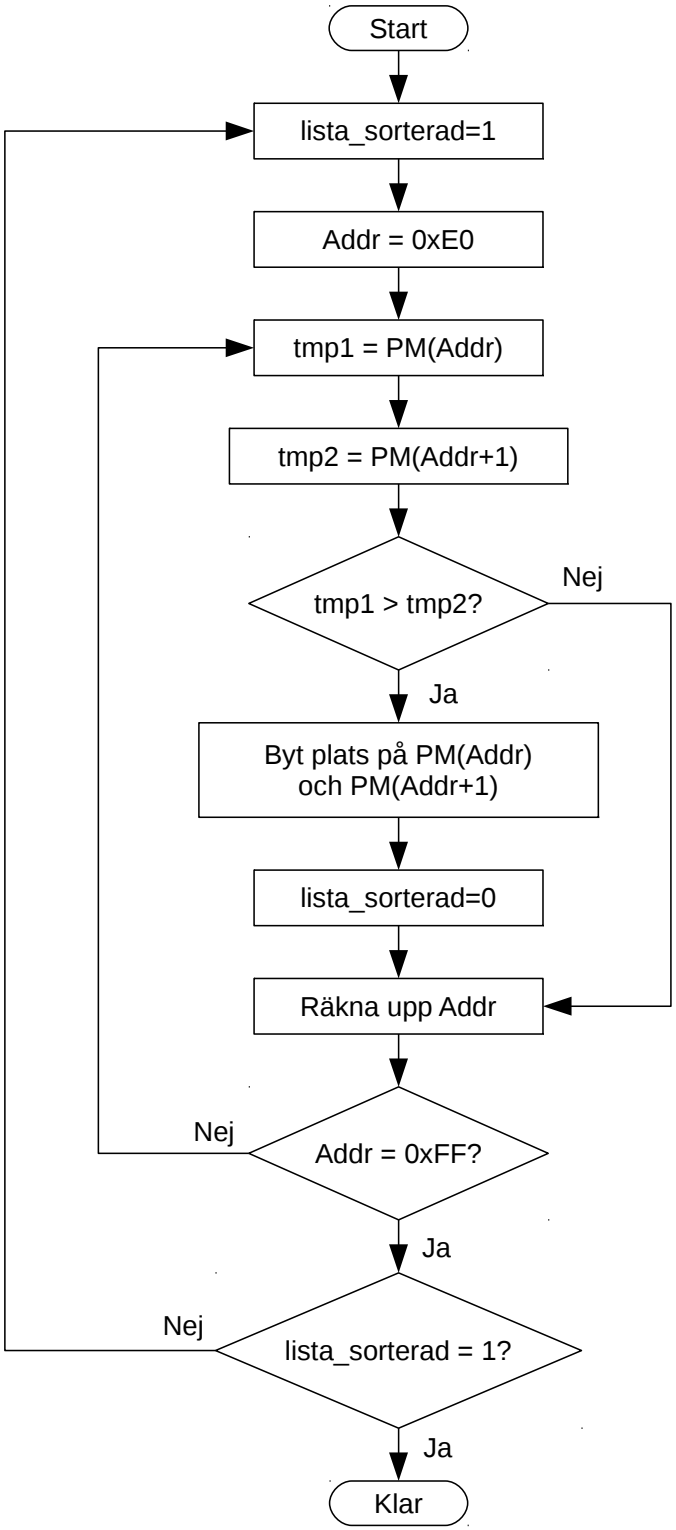
Notering: Vi kommer att använda fem slumpmässigt genererade listor[2] för att testa prestandan på era lösningar. Ni tjänar alltså inte på att optimera ert program för just den lista som anges i Tabell 1.

[2]För de som är intresserade så kommer vi att använda $5 \cdot 32 \cdot 16$ bitar tagna ifrån `/dev/random`.

Adress	Före	Efter
E0	92f1	8034
E1	8034	835f
E2	971b	8832
E3	99fb	90e8
E4	7ef1	92f1
E5	90e8	959f
E6	5ee7	969c
E7	3de3	971b
E8	7351	99fb
E9	53ed	9f74
EA	56a2	9fc4
EB	dea5	b11c
EC	6c5a	bd89
ED	835f	bfb0
EE	7c67	dea5
EF	ec86	ec86
F0	bd89	3de3
F1	969c	4c67
F2	5f63	53ed
F3	72d7	56a2
F4	959f	5ee7
F5	6081	5f63
F6	4c67	6081
F7	7e12	623d
F8	9fc4	6c5a
F9	b11c	7044
FA	623d	72d7
FB	8832	7351
FC	78ea	78ea
FD	9f74	7c67
FE	7044	7e12
FF	bfb0	7ef1

Tabell 1: Ett sorteringsexempel

Appendix 1: Flödesschema för bubblesort



Adress	ALU	TB	FB	S	P	LC	SEQ	myADR	(Hexadecimalt)	Kommentar

Lathund till fälten (se labhäfte för mer information)

ALU:
0000 NOP
0001 Buss
0010 Buss'
0011 0
0100 AR+Buss
0101 AR-Buss
0110 AR & Buss
0111 AR | Buss

ALU:
1000 AR+Buss (inga flaggor)
1001 AR << 1
1010 ARHR << 1
1011 AR >> 1 (aritmetiskt)
1100 ARHR >> 1 (aritmetiskt)
1101 AR >> 1 (logiskt)
1110 Rotera AR åt vänster
1111 Rotera ARHR åt vänster

TB	FB
000 NOP	000 NOP
001 IR	001 IR
010 PM	010 PM
011 PC	011 PC
100 AR	100 -
101 HR	101 HR
110 GRx	110 GRx
111 Styrord	111 ASR

P
0 NOP
1 PC++

LC
00 NOP
01 LC--
10 LC=Buss
11 LC=uADR

S
0 GRx-mux styrs av GRx-fältet
1 GRx-mux styrs av M-fältet

SEQ
0000 uPC++
0001 uPC=K1
0010 uPC=K2
0011 uPC=0
0101 uPC=uADR
0110 uJSR uADR
0111 uRTS
1000 Hopp om Z=1
1001 Hopp om N=1
1010 Hopp om C=1
1011 Hopp om O=1
1100 Hopp om L=1
1111 HALT

Adress	ALU	TB	FB	S	P	LC	SEQ	myADR	(Hexadecimalt)	Kommentar

Lathund till fälten (se labhäfte för mer information)

```
ALU:
0000 NOP
0001 Buss
0010 Buss'
0011 0
0100 AR+Buss
0101 AR-Buss
0110 AR & Buss
0111 AR | Buss

ALU:
1000 AR+Buss (inga flaggor)
1001 AR << 1
1010 ARHR << 1
1011 AR >> 1   (aritmetiskt)
1100 ARHR >> 1 (aritmetiskt)
1101 AR >> 1   (logiskt)
1110 Rotera AR åt vänster
1111 Rotera ARHR  åt vänster

TB            FB
000 NOP       000 NOP
001 IR        001 IR
010 PM        010 PM
011 PC        011 PC
100 AR        100 -
101 HR        101 HR
110 GRx       110 GRx
111 Styrord   111 ASR

P
0 NOP
1 PC++

LC
00 NOP
01 LC--
10 LC=Buss
11 LC=uADR

S
0 GRx-mux styrs av GRx-fältet
1 GRx-mux styrs av M-fältet

SEQ
0000 uPC++
0001 uPC=K1
0010 uPC=K2
0011 uPC=0
0101 uPC=uADR
0110 uJSR uADR
0111 uRTS
1000 Hopp om Z=1
1001 Hopp om N=1
1010 Hopp om C=1
1011 Hopp om O=1
1100 Hopp om L=1
1111 HALT
```

Adress	ALU	TB	FB	S	P	LC	SEQ	myADR	(Hexadecimalt)	Kommentar

Lathund till fälten (se labhäfte för mer information)

ALU:
0000 NOP
0001 Buss
0010 Buss'
0011 0
0100 AR+Buss
0101 AR-Buss
0110 AR & Buss
0111 AR | Buss

ALU:
1000 AR+Buss (inga flaggor)
1001 AR << 1
1010 ARHR << 1
1011 AR >> 1 (aritmetiskt)
1100 ARHR >> 1 (aritmetiskt)
1101 AR >> 1 (logiskt)
1110 Rotera AR åt vänster
1111 Rotera ARHR åt vänster

TB		FB	
000	NOP	000	NOP
001	IR	001	IR
010	PM	010	PM
011	PC	011	PC
100	AR	100	-
101	HR	101	HR
110	GRx	110	GRx
111	Styrord	111	ASR

P
0 NOP
1 PC++

LC
00 NOP
01 LC--
10 LC=Buss
11 LC=uADR

S
0 GRx-mux styrs av GRx-fältet
1 GRx-mux styrs av M-fältet

SEQ
0000 uPC++
0001 uPC=K1
0010 uPC=K2
0011 uPC=0
0101 uPC=uADR
0110 uJSR uADR
0111 uRTS
1000 Hopp om Z=1
1001 Hopp om N=1
1010 Hopp om C=1
1011 Hopp om O=1
1100 Hopp om L=1
1111 HALT

Adress	ALU	TB	FB	S	P	LC	SEQ	myADR	(Hexadecimalt)	Kommentar

Lathund till fälten (se labhäfte för mer information)

ALU:
0000 NOP
0001 Buss
0010 Buss'
0011 0
0100 AR+Buss
0101 AR-Buss
0110 AR & Buss
0111 AR | Buss

ALU:
1000 AR+Buss (inga flaggor)
1001 AR << 1
1010 ARHR << 1
1011 AR >> 1 (aritmetiskt)
1100 ARHR >> 1 (aritmetiskt)
1101 AR >> 1 (logiskt)
1110 Rotera AR åt vänster
1111 Rotera ARHR åt vänster

TB	FB
000 NOP	000 NOP
001 IR	001 IR
010 PM	010 PM
011 PC	011 PC
100 AR	100 -
101 HR	101 HR
110 GRx	110 GRx
111 Styrord	111 ASR

P
0 NOP
1 PC++

LC
00 NOP
01 LC--
10 LC=Buss
11 LC=uADR

S
0 GRx-mux styrs av GRx-fältet
1 GRx-mux styrs av M-fältet

SEQ
0000 uPC++
0001 uPC=K1
0010 uPC=K2
0011 uPC=0
0101 uPC=uADR
0110 uJSR uADR
0111 uRTS
1000 Hopp om Z=1
1001 Hopp om N=1
1010 Hopp om C=1
1011 Hopp om O=1
1100 Hopp om L=1
1111 HALT