TSEA28 Datorteknik Y (och U)

Föreläsning 10

Kent Palmkvist, ISY

LINKÖPINGS UNIVERSITET

Dagens föreläsning

- Addresseringsmoder
- Introduktion till laboration 4 (mikroprogrammering)
- Avbrott i mikroprogrammerad dator

Praktiska kommentarer

- Labutrustning i MUX1 borttagen
  - Labbet används i andra kurser
  - Distansutrustning kommer finnas kvar i MUX2
- Anmälningslista till lab 4 och 5 släpps på imorgon Onsdag 27/3 kl 12.45
  - 2 + 2 h per labb precis som tidigare, anmälan till tider via lisam anmälan
  - Lab 4 enbart simulering, kan köras via thinlinc
    - Redovisning måste göras på plats i labbet
  - Ingen distansutrustning för lab 5, måste köras på plats

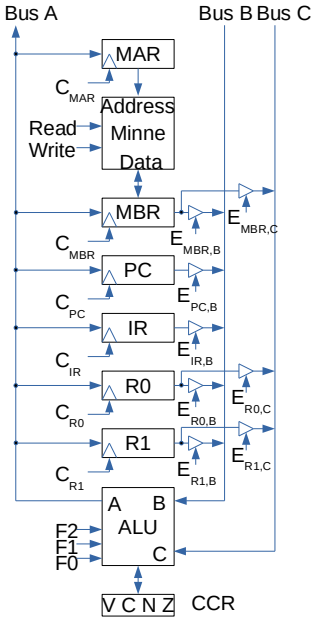
Maskininstruktioner

- Exempel på maskininstruktioner som ska kunna utföras

Op-code	Namn		Operation
0 0 0	LOAD	R0,M	[R0] <- [M]
0 0 1	LOAD	R1,M	[R1] <- [M]
0 1 0	STORE	M,R0	[M] <- [R0]
0 1 1	STORE	M,R1	[M] <- [R1]
1 0 0	ADD	R1,R0	[R1] <- [R1] + [R0]
1 0 1	SUB	R1,R0	[R1] <- [R1] – [R0]
1 1 0	BRA	T	[PC] <- T
1 1 1	BEQ	T	Om [Z] = 1 då [PC] <- T

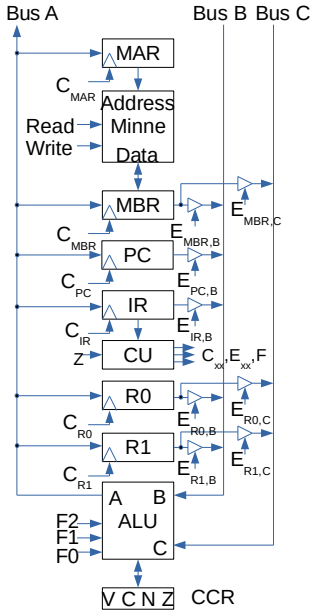
Op-code |	M eller T

- Mikroprogrammet beskriver för varje instruktion alla styrsignaler och steg för implementering av instruktionen
  - Hämta instruktion (fetch)
  - Utför instruktion (beror på vilken instruktion som hämtats)



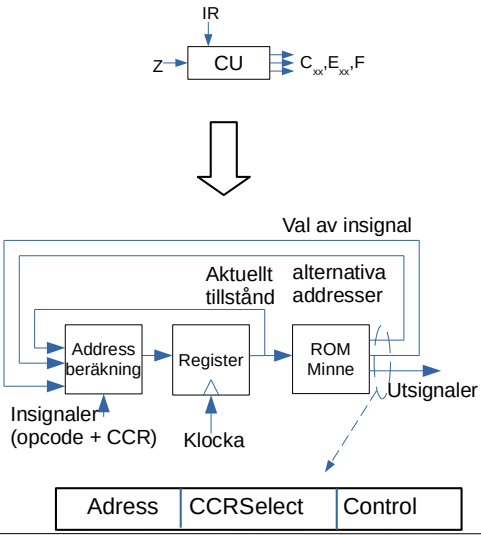
Processorarkitektur inkl CU

- Inte alla bitar i IR behöver kopplas till CU
  - Räcker med opcode-bitarna (3 stycken)
  - Endast Z inkopplad i exemplet då bara instruktionen BEQ använder Z-flaggan
- CU är mikroprogrammerad
  - Effektivt sätt realisera sekvenser av styrsignaler



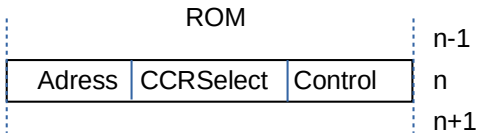
Generell mikroprogrammeringsstruktur

- ROM-minnet innehåller mikroprogrammet
  - En address för varje steg
  - Tre delar i ROM utdata
    - Styrsignaler (En bit för varje styrsignal)
    - En (eller flera) styrsignaler för val av CCR bitar
    - En (eller flera) nästa address (steg, tillstånd) beroende på resultat av opcode + CCR-bitar



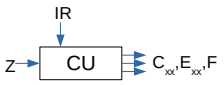
Mikrokodsinstruktionens delar

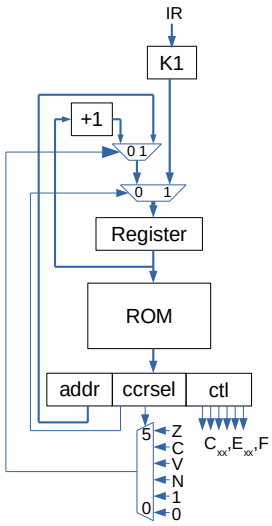
- CCRSelect väljer vilken nästa instruktion blir
  - n+1
  - Hopp till Adress
  - Villkorligt hopp till Adress
    - CCRSelect väljer vilken flagga som bestämmer om hoppet ska tas
  - Implementera med mux styrd av CCRSelect,
- Adressfältet används inte alltid
- För val av adress för respektive maskinkod behövs motsvarighet till case-sats
  - Används av alla instruktioner, kan vara värt extra hårdvara



Mer komplett kontrollenhet

- Data i varje address i ROM (mikroprogramminnet)
  - Styr enskilda kontrollsignaler i arkitekturen (Cxx,E_{XX})
  - Välj vad som kan ge nästa address (+1, addr, K1)
  - Alternativ nästa address till mikroprogrammet
  - Mikroprogramminne inte samma som programminne!
- IR består av operationskod och argument
  - Operationskod (Add, Move, etc.) går via K1

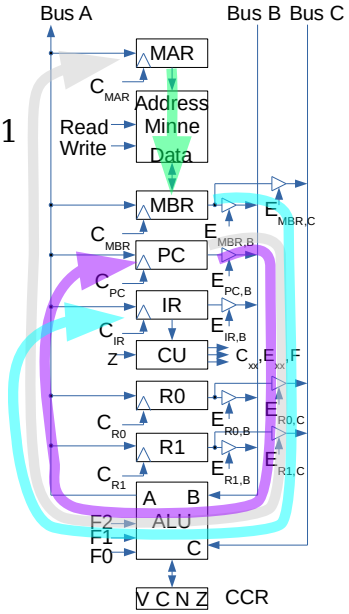




Exekvering av instruktion, del 1

- Fetch: Läs instruktion från minne till IR, öka PC med 1
  - Kopiera PC till MAR
  - Öka PC med 1 (peka på instruktion efter)
  - Läs minne till MBR
  - Kopiera MBR till IR
  - Välj rätt mikroprogramaddress baserat på opcode
  - Motsvarande styrsignaler (en av flera möjliga lösningar)

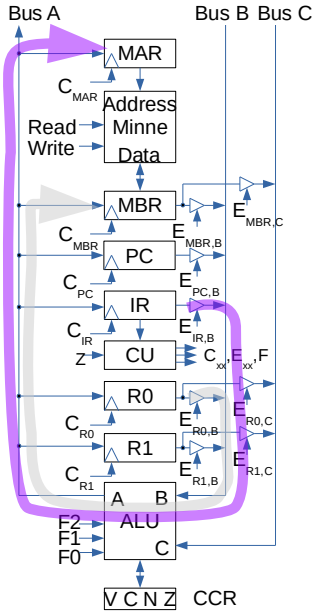
	uaddr	ccrsel	kontrollsignal	Addr
○	0	+1	$E_{PC,B}$ = 1, F2F1F0 = 0,0,0, C_{MAR}	-
●	1	+1	$E_{PC,B}$ = 1, F2F1F0 = 0,1,0, C_{PC}	-
●	2	+1	Read=1,C_{MBR}	-
●	3	+1	$E_{MBR,B}$ = 1, F2F1F0 = 0,0,0, C_{IR}	-
	4	K1	-	-



Exekvering av instruktion, del 2

- Utför instruktion 010 (STORE M,R0)
  - Kopiera R0 till MBR
  - Kopiera IR till MAR (får bara med adressdelen av instruktionen)
  - Skriv minne från MBR, starta om mikroprogrammet
  - Motsvarande styrsignaler (en av flera möjliga lösningar)

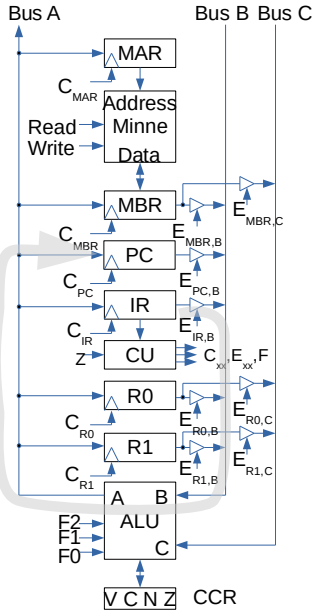
	uaddr	ccrsel	kontrollsignal	Addr
○	k	+1	$E_{R0,B}$ = 1, F2F1F0 = 0,0,0, C_{MBR}	-
●	k+1	+1	$E_{IR,B}$ = 1, F2F1F0 = 0,0,0, C_{MAR}	-
●	k+2	1	Write=1	0



Exekvering av instruktion, del 2

- Utför instruktion 111 (BEQ T)
  - Hoppa över nästa mikroinstruktion om Z = 1
  - Starta om mikroprogrammet
  - Kopiera IR till PC, starta om mikroprogrammet
  - Motsvarande styrsignaler (en av flera möjliga lösningar)

uaddr	Addrsel	kontrollsignal	Addr
l	Z	-	l+2
l+1	1	-	0
l+2	1	$E_{IR,B}$ = 1, F2F1F0 = 0,0,0, C_{PC}	0

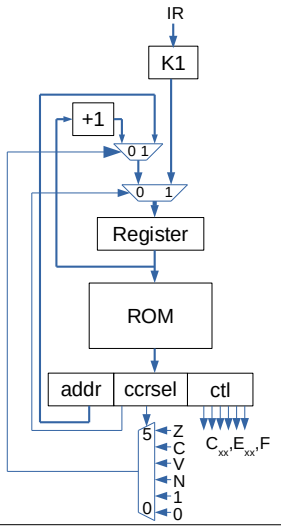


Mikroprogram för exemplet

uaddr	Addrsel	kontrollsignal	Addr
0	+1	$E_{PC,B}$ = 1, F2F1F0 = 0,0,0, C_{MAR}	-
1	+1	$E_{PC,B}$ = 1, F2F1F0 = 0,1,0, C_{PC}	-
2	+1	Read=1,C_{MBR}	-
3	+1	$E_{MBR,B}$ = 1, F2F1F0 = 0,0,0, C_{IR}	-
4	K1	-	-
k (5)	+1	$E_{R0,B}$ = 1, F2F1F0 = 0,0,0, C_{MBR}	-
k+1 (6)	+1	$E_{IR,B}$ = 1, F2F1F0 = 0,0,0, C_{MAR}	-
k+2 (7)	1	Write=1	0
l (8)	Z	-	l+2
l+1 (9)	1	-	0
l+2 (10)	1	$E_{IR,B}$ = 1, F2F1F0 = 0,0,0, C_{PC}	0

```
 0  00000 0000 000001 01000000 000 00
 1  00000 0000 000100 01000000 010 00
 2  00000 0000 000000 00000000 000 10
 3  00000 0000 001000 10000000 000 00
 4  00000 1000 000000 00000000 000 00
 5  00000 0000 000010 00010000 000 00
 6  00000 0000 000001 00100000 000 00
 7  00000 0001 000000 00000000 000 01
 8  01010 0101 000000 00000000 000 00
 9  00000 0001 000000 00000000 000 00
10  00000 0001 000100 00100000 000 00
```

ROM
5 bit addr
4 bit ccrsel
6 bit Cxx
8 bit Exx
3 bit F2F1F0
1 bit Read,1 bit Write

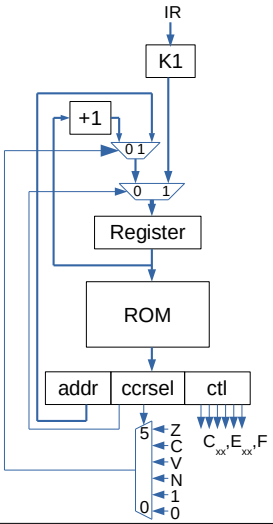


Mikrokod, minnet K1

- K1 anger vilken adress i ROM en viss maskininstruktion startar på
  - Uppslagstabell (litet minne)

Addr	Data	
000	.....	mikrokodsaddress för LOAD R0,M
:		
010	00101	mikrokodsaddress för STORE M,R0
:		
111	01000	mikrokodsaddress för BEQ T

```
0  00000 0000 000001 01000000 000 00  ; Fetch
1  00000 0000 000100 01000000 010 00
2  00000 0000 000000 00000000 000 10
3  00000 0000 001000 10000000 000 00
4  00000 1000 000000 00000000 000 00
5  00000 0000 000010 00010000 000 00  ; STORE
6  00000 0000 000001 00100000 000 00
7  00000 0001 000000 00000000 000 01
8  01010 0101 000000 00000000 000 00  ; BEQ
9  00000 0001 000000 00000000 000 00
10 00000 0001 000100 00100000 000 00
```

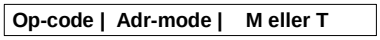


Viktiga kommenterar

- Resultat från flytt inte tillgängligt förrän nästa klockcykel (steg) i registren
  - Ex: ladda IR måste slutföras innan K1 kan väljas
  - Ex: ALU måste slutföra operation innan flaggor kan användas i villkorliga hopp
- Bara en källa får slås på till en buss per mikrokodssteg
  - Går att förstöra (bränna upp) krets om flera slås på samtidigt

Adresseringsmoder

- Samma grundoperation kan användas med olika adresserinsmoder
  - Omedelbar (t ex add R0,#0x10)
  - Absolut (t ex add R0,0x100)
  - Indirekt (t ex add R0,[R1])
  - Offset etc. (t ex add R0,[R1,#4])
- Endast hämtning av 1:a argumentet skiljer
  - Samma mikrokod i övrigt
- Utöka maskinkodsinstruktionen med bitar för adresseringsmode
  - Antar allt får plats på en adress i minnet



Jämför Thumb kod

- Maskininstruktionen kan vara olika lång beroende på adresseringsmode

Typ	Maskinkod	Assembly	; kommentar
Register	0x4608	MOV r0,r1	; 16-bit instruktion
Omedelbar	0x2034f241	MOV r0,#0x1234	; 32-bit instruktion
Absolut	Finns bara en begränsad version i thumb/ARM där endast adresser nära början respektive slutet kan användas.		
Indiret	0x6808	LDR r0,[r1]	; 16-bit instruktion
Relativ	0x0022F8D1	LDR r0,[r1,#34]	; 32-bit instruktion

- Inte så tydligt vilka bitar i maskininstruktionen som skulle vara adresseringsmod

Adresseringsmoder, ej ARM

- Många processorer tillåter andra instruktioner förutom LDR och STR att läsa/skriva i minnet
  - Exempel MC68000: ADD 0x1234567,D0
    - Addera värdet i minnescell 0x1234567 till värdet i registret D0, svaret placeras i D0.
  - Exempel MC68000: AND 4(A0),D1
    - Beräkna adress genom att addera 4 till värdet i A0, hämta värdet på den beräknade adressen, beräkna logisk and mellan hämtade värdet i A0 och D1, spara resultatet i D1
- De flesta instruktioner använder då minnesadresser
  - Addressberäkning behöver göras för alla instruktioner
- Precis som för avkodning av OPCODE-bitarna
- Utöka maskinkodsinstruktionen med bitar för adresseringsmode
  - Antar allt får plats på en adress i minnet

Op-code |	Adr-mode |	M eller T

Jämför M68000 (16-bit brett minne)

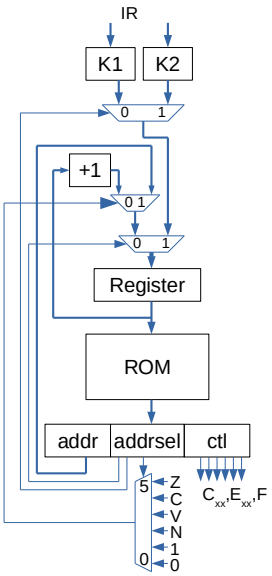
- Maskininstruktionen kan vara olika lång beroende på adresseringsmode (D0, D1 och A0 är register)
  - Register 0xC081 AND D1,D0 ; D0 = D0 AND D1
  - Omedelbar 0x0280 AND #$1234567,D0 ; D0 = D0 AND 0x01234567
    0x0123
    0x4567
  - Absolut 0xC0B9 AND $1234567,D0 ; D0 = D0 AND Minne[0x01234567]
    0x0123
    0x4567
  - Indirekt 0xC090 AND (A0),D0 ; D0 = D0 AND Minne[A0]
  - Relativ 0xC0A8 AND 4(A0),D0 ; D0 = D0 AND Minne[A0+4]
    0x0004
- Vissa adresseringsmoder kräver fler läsningar av programminnet.
  - Parametrar och konstanter i efterföljande programminnesadresser
  - Om flera adresser behöver PC påverkas beroende på adresseringsmode

Adresseringsmoder

- Adresseringsmode kräver avkodning
  - Många möjliga val, används samma sätt som för avkodning av opcode

Op-code |	Adr-mode |	M eller T

- Lägg till K2 som väljer olika mikrokodsrutiner beroende på addresseringsmode
  - Beräkna slutlig minnesadress, spara i t ex MAR
  - Mikrokoden kan då förutsätta att MAR redan pekar på rätt minnesadress



Exekvering av maskininstruktion, med adresseringsmode

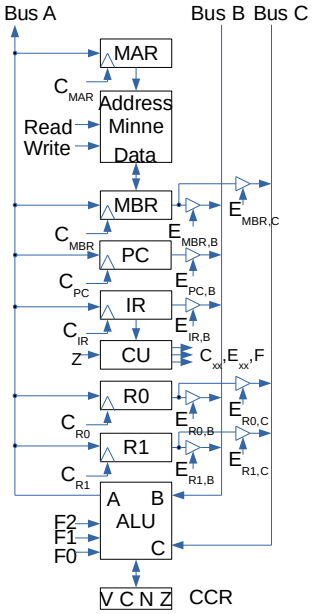
- Total sekvens består av tre delar
  - Del 1a Hämta instruktion till IR, räkna upp PC
  - Del 1b Beräkna adress för argument baserat på adresseringsmode, placeras i MAR
  - Del 2 Utför operation baserat på opcode. Minnesadress till argument finns i MAR
- MAR pekar på adress till värdet (efter del 1b slutförts)
  - Register: Värdet inte lagrat i minnet, måste hanteras speciellt (använder inte MAR)
  - Omedelbar: nästa adress i programminnet (PC), öka även PC! Gäller ifall omedelbar adresseringsmode använder extra ord i programminnet för konstanten
  - Direkt: argument i maskininstruktion
  - Indirekt: registervärde
  - Offset: registervärde + argument

Adresseringsmoder, implementation efter del 1a

- Sekvens i mikroprogrammet efter del 1a (uprogaddr = mikroprogramadress)
  - IR innehåller aktuell maskininstruktion, PC pekar på nästa

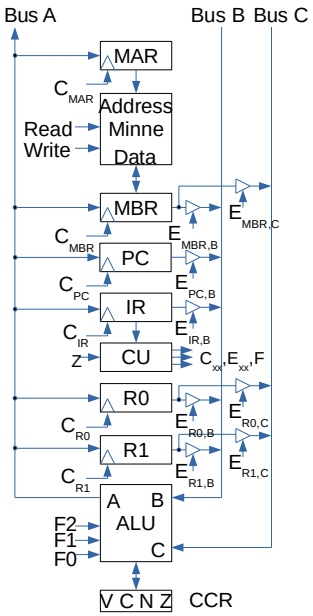
```
5   IR-K2 -> uprogaddr  ; del 1b, avkoda Adr-mode
23  PC -> MAR, PC=PC+1, IR-K1 -> uprogaddr  ; Omedelbar
24  IR -> MAR, IR-K1-> uprogaddr      ; Direkt, argument är
                                      ; adress
25  R1 -> MAR, IR-K1 -> uprogaddr     ; Indirekt
26  R1 + IR -> MAR, IR-K1 -> uprogaddr ;  offset register
27  Read   ; Exempel ADD (dvs K1 för ADD opcode = 27)
28  MBR ADD R0 -> R0, 0 -> uprogramaddress
```

- Inte angivit explicit E, C och F2-F0.
  - Add-version ovan enbart för omedelbar, direkt, indirekt och offset register adresss
  - ADD => Skicka ut R0 på bus C, MBR på bus B



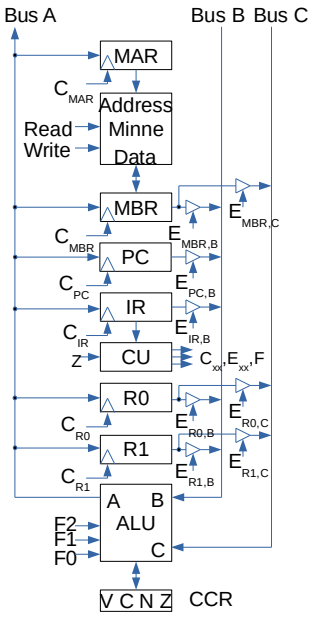
Alternativa lösningar till arkitekturen

- Kan kanske minska antal bussar (ta bort C)
  - Kräver något sätt att ge två olika indata vid addition och subtraktion
    - Använd ett extra mellanvärdesregister istället
    - Kostar mer tid (flytta till mellanregister innan operation)
- Kan öka antal bussar (A1 och A2)
  - Behöver fler vägar från utsignaler (Exx) till bussar
    - Lägg multiplexer parallellt med ALU
  - Möjliggör snabbare exekvering
    - Flytta värde samtidigt med beräkning i ALU
  - Mer hårdvara krävs
    - Inklusive längre mikrokodsinstruktionsord



Alternativa lösningar till arkitekturen, forts.

- Komplexare register
  - PC som kan räkna nästa adress utan att använda ALU
    - Laddbar räknare
  - SP med automatisk upp/nedräkning
- Minimera antal kontrollbitar
  - Avkodare för generering av EXX,B
    - 3 bitar räcker (5 olika källor)
    - Sparar då 2 bitar
  - Samma kan göras även för CXXX
    - Begränsar då möjligheten att skriva samma värde till flera register samtidigt (ovanligt?)
  - Vertikal mikroprogrammering iställlet för horisontell

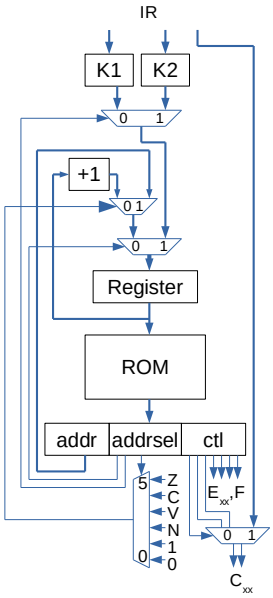


Stöd för fler register

- Fler register => fler dubbleringar av mikrokod med små ändringar för respektive register
  - Ineffektivt
- Låt register anges som några bitar i maskinkoden (assemblerade instruktionen)
  - Lägg till multiplexer för att kunna välja registerval
  - Kräver ytterligare bit i mikrokodsordet
  - Minskar total mängd mikrokod (en rutin för alla register)
  - Samma sak kan göras som för buss-enable signaler (dvs vertikal mikrokodning av E_{xx}) för att använda få bitar i maskininstruktionen.

maskininstruktion

operation	Adrmode	Reg-src	Reg-dest



Viktiga begränsningar för mikroprogram

- Ordbredd på mikroprogram ökar om max antal rader i mikroprogram ökar
  - Fler bitar behövs för adressfältet
- Om varje maskininstruktion utför mer nyttigt arbete, desto färre antal instruktioner behövs i applikationen
  - Ger bättre prestanda då hämtning av maskininstruktion inte utför något nyttigt arbete, bara tar tid
  - Kräver mindre programminne för applikationen
  - Allt svårare bygga kompilatorer som kan utnyttja komplicerade instruktioner
  - Begränsat möjligt antal olika instruktioner för given ordlängd

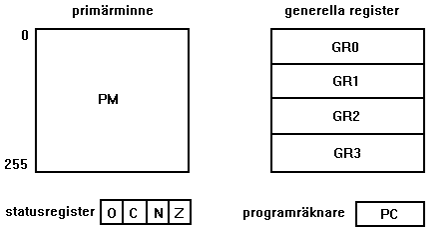
Historisk trend

- Antal instruktioner i mikroprocessorer ökar fortfarande
  - Specialinstruktioner för grafik, kodning, flyttal etc.
- Begränsning i antal möjliga instruktioner löstes genom prefix-instruktioner respektive större ordlängd/längre instruktioner
  - Ex: Speciell maskininstruktion som gör att efterföljande maskininstruktion använder en alternativ mikrokod för att tolkas
    - Ex: Prefix i x86-datorer (PC)
  - Ex: Speciell maskininstruktion som gör att registren i efterföljande instruktion byts mot annan uppsättning
    - Ex: IX och IY-register i Z80
  - Ex: Speciell maskininstruktion som byter mode (ersätter i princip mikrokodsminnet med ett annat) för alla instruktioner som följer
    - Ex: ARM har multipla instruktionsuppsättningar: ARM, THUMB etc.
  - Ordlängd och prefix påverkar både storlek på applikationen och prestanda
    - Prefix undviks ofta numera, fix instruktionslängd vanligare

Lab 4 mikroprogrammering

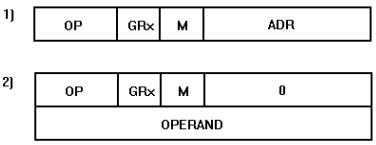
Introduktion laboration 4 (MIA)

- Simuleringsmodell för övning på mikroprogrammering
  - Fördefinierad datorarkitektur med en databuss
- Maskinspråksmodell
  - Primärminne (256 ord a 16 bitar)
  - 4 generella register
  - PC
  - 4 flaggor (OCNZ)
  - 16-bitars databuss, 8-bitars addressbuss



MIA maskininstruktionsformat

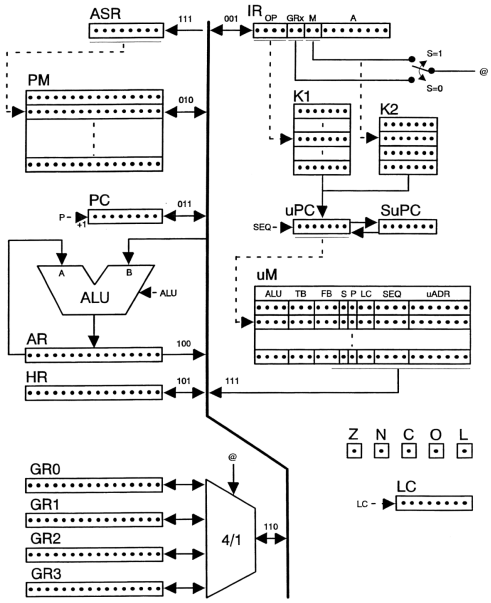
- 2 Instruktionsformat
  - Ett 16-bitars ord
  - Ett 16-bitars ord samt en 16 bitars operand
  - Inte samma som i exemplen tidigare!



- OP anger operation
- GRx angera vilket generellt register ska användas
- M anger addresseringsmode
- ADR minnesaddress eller offset vid hopp

Laboration intern arkitektur mikroprogrammering (MIA)

- 1 delad databuss
- 1 sändare och 1 mottagare per överföring på bussen
  - Vertikal mikroprogrammering
- Inget MBR register
- 1 hjälpregister (HR)
- Mikroprogram kan använda 1 nivå av subrutinanrop
  - SuPC register kan lagra återhoppsaddress i mikrominnet



Mikroinstruktionsformat

- ALU styr ALU:ns funktion
- TB anger källan till bussens värde
- FB anger vart bussens värde sparas
- S Välj M eller GRx fält i maskininstruktionen
- P Räkna upp PC
- LC Loopräknare i mikrokoden
- SEQ styr eventuella hopp i mikrokoden
- uADR måladdress vid vissa hopp i mikrokoden

Laborationens syfte

- Träna och förstå hur mikroprogrammering fungerar
  - Tentor kan innehålla en liknande programmeringsuppgift…
- Implementera några få instruktioner
- Kör ett enkelt summeringsprogram med instruktionerna
- Bygg ett maskinkodsprogram för sortering mha era implementerade instruktioner
  - Extra instruktioner behövs och egna specialinstruktioner tillåtna
- Tävling…
  - Se labbanvisningen
  - Vinnande labbgrupp vinnner: ???

Jämförelse maskinkod och mikrokod

- Programmeringsmässigt ganska lika
  - Kod tolkas steg för steg, hopp kan göras i sekvensen, allt har en adress
- Mikrokod mer parallell till sin natur
  - Kan få flera saker att utföras i samma klockcykel
- Maskinkod mer generell
  - Arkitekturen internt i processorn kan ändras utan att maskinkoden skrivs om
  - Kan återanvända all tidigare skriven och kompilerad programvara
  - Färre detaljer att hålla ordning på

Avbrott och mikrokod

- Avbrottsingång blir ytterligare en insignal till styrenheten
  - Påverkar sekvensen av steg
  - Testas lämpligen först i hämtfasen (som en flagga)
    - Avbrott bryter inte instruktionen mitt i
- Alternativ sekvens om avbrott aktivt när nästa instruktion ska hämtas
  - Spara PC på stack
  - Spara statusregister på stack
  - Hämta/sätt adress till avbrottsrutin
  - Spärra så avbrott inte sker igen på nästa instruktion
- Behövs då också återhopp från avbrott (återställ PC, flaggor mm)