TSEA28 Datorteknik Y (och U)

Föreläsning 9

Kent Palmkvist, ISY

LINKÖPINGS UNIVERSITET

Dagens föreläsning

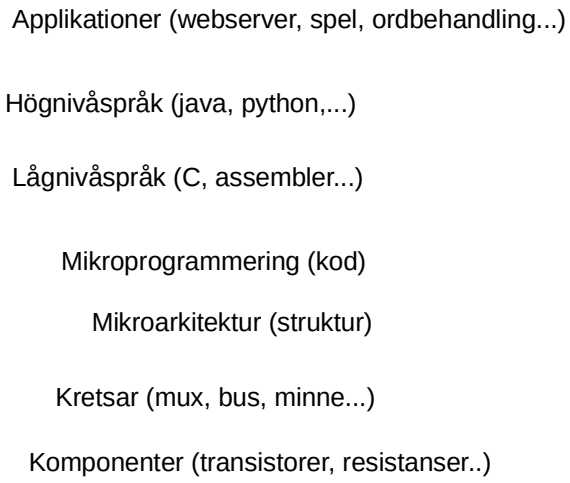
- Byggblocken i en processor
  - Hur de fungerar
- Grundläggande intern arkitektur
  - Vidareutveckling av vad som visats tidigare
- Styrning med mikrokod
  - Skapa styrsekvenser med minne och register

Praktiska kommentarer

- Labutrustning i MUX1 har tagits bort
  - Labbet används i andra kurser
  - Distansutrustning kommer finnas kvar i MUX2
- Anmälningslista till lab 4 och 5 släpps Onsdag 27/3 kl 12.45
  - Lab 4 enbart simulering, kan köras via thinlink
    - Redovisning på plats i lab
  - Ingen distansutrustning för lab 5, måste köras på plats

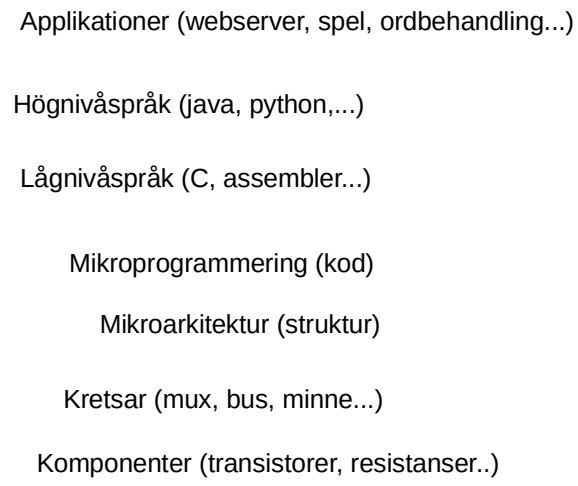
Abstraktionshierarki

- Datorns uppbyggnad och funktion kan beskrivas på olika nivåer
  - Lägre abstraktionsnivå => färre personer involverade i design
  - I denna kurs fokuserar vi på mikroarkitektur upp till lågnivåspråk



Abstraktionshierarki, forts.

- Lågnivåspråk kräver kunskap om datorns uppbyggnad
  - T ex assemblerspråk specifikt för olika klasser av datorer
    - ARM (t ex tfn) skiljer sig från intel/AMD x86_64 (t ex laptop)
- Mikroprogrammering kräver kunskap om datorn interna struktur
  - T ex hur olika register i processorn är ihopkopplade
  - Finns inte i alla datorer (och oftast inte tillgängligt)



Motivering

- Prestanda på en dator (operationer per sekund) bestäms av
  - Hur ofta en ny instruktion kan startas
  - Hur snabbt minnet kan läsas/skrivas
  - Vilka typer av instruktioner som finns
- Ska under denna del av kursen titta närmare på hur prestanda kan påverkas
  - Kräver kunskap om hur instruktioner implementeras i datorn
- Maskinkod är en abstraktion
  - Flera olika strukturer kan implementera samma maskinkod med olika intern struktur och prestanda (80386 vs i7 vs Ryzen)

En närmare titt på datorn

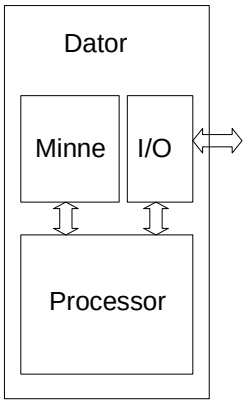
- Varje enskild maskinkodsinstruktion (assemblerinstruktion) motsvarar oftast en sekvens av aktiviteter
  - Hämta instruktion från minnet
  - Avkoda instruktion
  - Hämta argument från minne eller register
  - Beräkna resultat
  - Skriv resultat till minne eller register

Exempel: ldr r1,[r2, r3]
Läs instruktionen från minnet
Beräkna adress r2+r3
Läs från minnescell på adress (dvs r2+r3)
Skriv värdet i register r1

- Dessa delsteg bestäms av den interna uppbyggnaden av processorn
  - Register, instruktionsuppsättning, vägar att skicka data, etc.

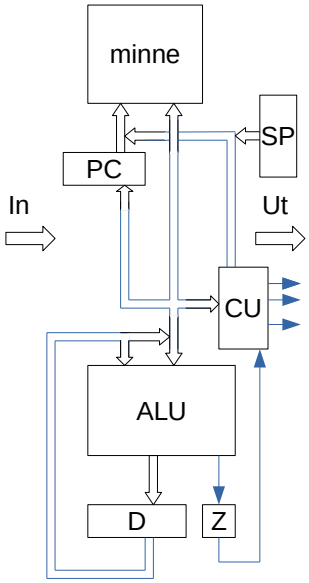
Dator vs processor etc.

- Dator
  - Komplett system med minnen och I/O (input/output)
- Processor (CPU – Central Processing Unit)
  - Beräkningsenheten i datorn. Allt utom minnen och I/O
- Mikroprocessor
  - Speciell version av processor där hela processorn sitter på ett chip
- Mikrokontroller
  - Enklare processor med vissa delar av I/O och ofta minne på ett chip
- SOC
  - System on Chip: Komplett dator med minne och I/O-enheter. Oftast högre prestanda (jämfört med mikrokontroller)



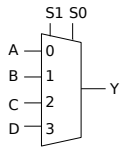
Datorns byggstenar

- Funktionsblock
  - Beräkna eller lagra data
  - T ex minne, ALU
- Kommunikation
  - Koppla ihop register, minnen mm
  - Bussar
- Kontroll (CU)
  - Bestäm vilket data skickas vart
  - Avkodning, styrsignaler
- Kommer utgå från boken (Figur 7.1)
  - Finns även bilder på bokens hemsida!



Byggblock

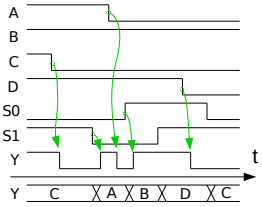
Mer om byggblock: Multiplexer



- För dom som läst digitalteknik tidigare borde detta vara repetition (Se även avsnitt 2.9, 2.10 i boken)
- Multiplexer
  - Välj att skicka ut värde på Y från en av ingångarna
  - Numrering motsvarar binära talet på styringång (S)

S1	S0	Y
0	0	A
0	1	B
1	0	C
1	1	D

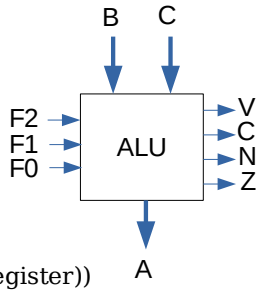
- Funktion beskriven med sanningstabell
  - Insignal och motsvarande utsignal
  - Komplett tabell (med bara 0 och 1 som insignalvärden) ger 64 kombinationer (6 ingångar => 2^6=64 olika indata)
- Funktion tidsmässigt
  - Ändring på ingång ger (nästan) direkt påverkan på utgång
  - Gäller både ingång A-D och S1 S0



Aritmetisk logisk enhet (ALU)

- Beräkna alla funktioner, välj sedan rätt med mux
  - Funktion väljs med F2 F1 F0:

    B -> A
    C -> A
    B+1 -> A
    C+1 -> A
    B-1 -> A
    C-1 -> A
    B+C -> A
    C-B -> A

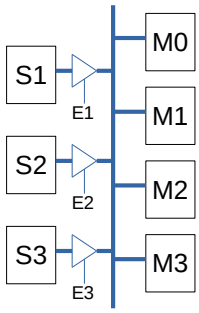


  - Beräkning uppdaterar flaggvärden (ofta kallad CCR (Condition Code Register))

    V: overflow vid 2-komplementberäkning
    C: Minnessiffra
    N: Negativt resultat (kopia av MSB)
    Z: Resultat = 0
- Alla utgångar påverkas direkt av ändring på ingång
  - Inget minne, ingen klocka

Buss, tristate-grind

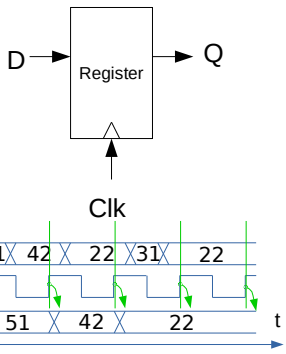
- Ofta många bitar parallellt, ofta en 2-potens
  - T ex 8 bitars databuss, 16 bitars addressbuss, eller likn.
- Flera sändare (Sx) kopplas till flera mottagare (Mx)
- Måste välja vem som får sända
  - Om flera försöker samtidigt förstörs sändningen
    - Jfr med att prata i mun på varandra
  - Sköts elektriskt med en s k tristategrind
    - Skickar ut 0, 1 eller Z
    - Z ut betyder urkopplad sändare
- En eller flera mottagare kan ta emot
  - Utpekade mottagare sparar värde som sändes

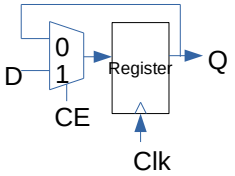


E	Y
0	Z, urkopplad
1	A

Register

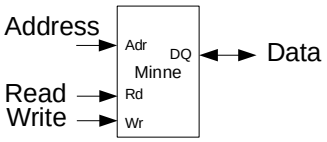
- Register
  - Laddar nytt värde varje gång klockan (CLK) går från 0 till 1 (positiv flank)
  - Om ingen flank behålls gamla värdet
- Ibland finns även CE (Clock Enable)
  - Ladda registret bara om CE = 1

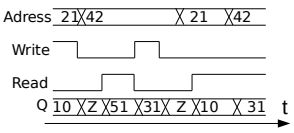




Minne

- Address pekar ut vilken minnescell som ska läsas/ändras
- Data innehåller värde till eller från minnescell
- Två olika funktioner
  - Läs: Read = 1, Kopiera värde i minnescell som address pekar på till DQ
  - Skriv: Write = 1, minnescell som address pekar på sätts till värde på DQ
  - Om varken skrivning eller läsning kopplas DQ ifrån (värde Z)
- Större minnen ofta långsammare än processorn





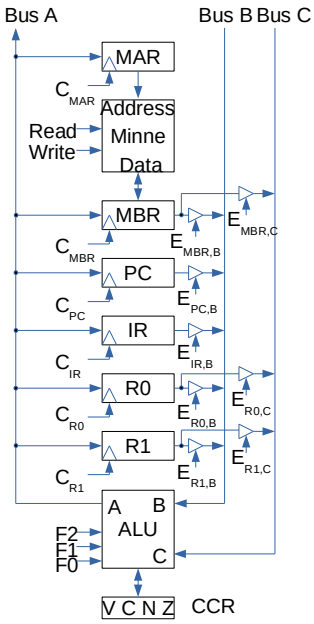
Skriv 10 till adress 21,
läs address 42 (innehåller värde 51)
skriv värde 31 till address 42
läs address 21 (10 pga skrivning tidigare)
läs address 42 (31 pga skrivning tidigare)

En enkel datoraktitektur

- En av många möljliga

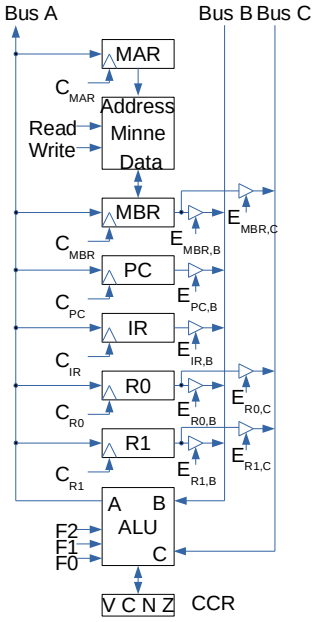
Enkel processorarkitektur

- En av många möjliga arkitekturer
  - Figur 7.1 i boken
- Enkel processor med få generella register
  - R0, R1 Generella dataregister
  - Övriga register inte direkt tillgängliga
    - PC Programräknare (peka på nästa instruktion)
    - CCR Statusflaggor (kopierar V,C,N,Z från ALU)
    - Instruktioner kan arbeta register-till-register och register-till/från-minne



Enkel processorarkitektur, forts.

- Register för att hålla address och data till/från minne
  - MAR Memory address register
  - MBR Memory buffer register
    - Ladda från minne om Read=1
    - Ladda från buss A om Read=0
- IR håller reda på aktuell instruktion som utförs
  - Inklusive adressargument
- Bus A drivs bara av ALU
- Bus B och C kan drivas av olika register (en i taget)
  - Bara en källa per gång för varje bus (via $E_{XX,B}$ resp. $E_{XX,C}$)

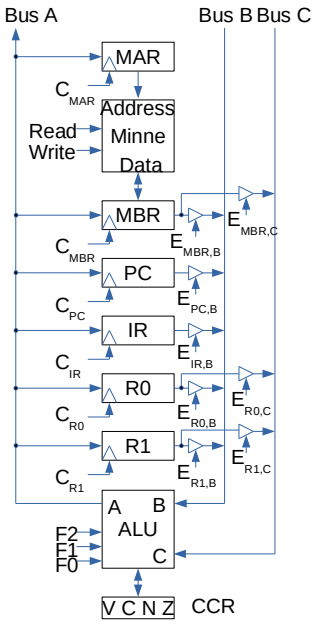


ALU operationer

- ALU utför beräkning eller kopiering
  - Välj funktion med F2, F1, F0

F2	F1	F0	Operation
0	0	0	A = B
0	0	1	A = C
0	1	0	A = B + 1
0	1	1	A = C + 1
1	0	0	A = B – 1
1	0	1	A = C – 1
1	1	0	A = B + C
1	1	1	A = C – B

- Uppdatera statusregistret CCR vid varje ALU-beräkning
  - V: aritmetiskt overflow (2-komplementsberäkning)
  - C: minnesbit vid operation (positiva heltal)
  - N: negativt resultat
  - Z: resultat noll

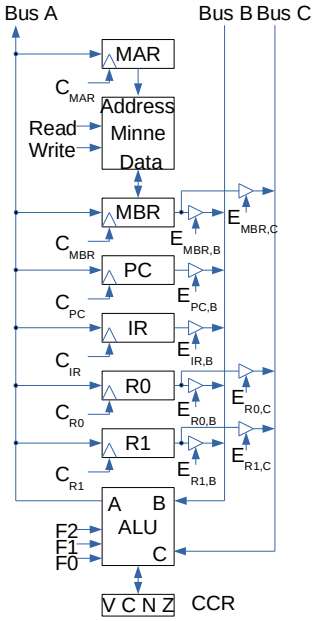


Styrsignaler

- Styrsignaler definierade i arkitekturen

Minne	Read	Läs från minne och placera värde i Data
Minne	Write	Skriv värde i Data till minnesaddress
Clock	C_{XXX}	Spara värde i register XXX (MBR, MAR, PC, IR, R0, R1)
Enable	$E_{XXX,B}$	Kopiera värde från XXX (MBR,PC,IR,R0,R1) till buss B
Enable	$E_{XXX,C}$	Kopiera värde från XXX till buss C (MBR,R0,R1)
ALU	F2F1F0	Välj funktion i ALU
CCR	CCR	Uppdatera flaggor

- Varje styrsignal ges värde 1 eller 0 i varje steg
  - Antag styrsignal = 0 om den inte anges



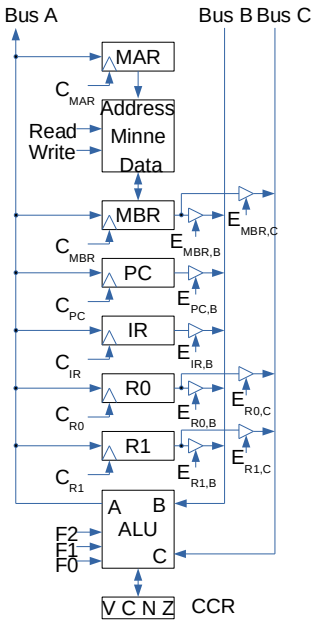
Maskininstruktioner

- 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

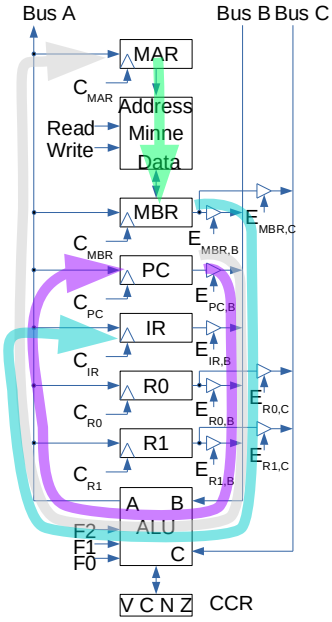
- Sekvens av styrsignaler (mikroprogram) beskriver för varje instruktion alla styrsignaler och steg för implementering av instruktionen
  - Hämta instruktion (fetch), görs för alla instruktioner
  - Utför instruktion (styrsignaler beror på vilken instruktion som hämtats)



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
  - Motsvarande styrsignaler (en av flera möjliga lösningar, går att använda 3 steg istället...)

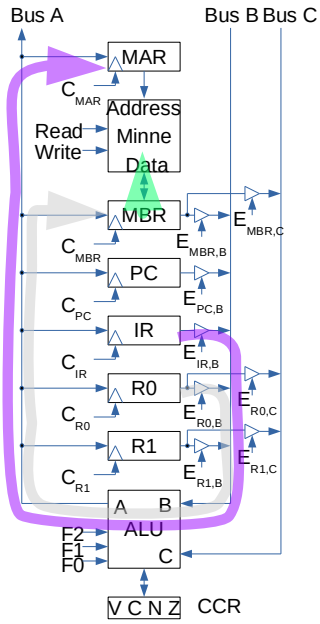
Steg	
T0	$E_{PC,B}$ = 1, F2F1F0 = 0,0,0, C_{MAR}
T1	$E_{PC,B}$ = 1, F2F1F0 = 0,1,0, C_{PC}
T2	Read=1,C_{MBR}
T3	$E_{MBR,B}$ = 1, F2F1F0 = 0,0,0, C_{IR}



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
  - Motsvarande styrsignaler (en av flera möjliga lösningar)

    Steg
    - T0 $E_{R0,B} = 1$, F2F1F0 = 0,0,0, C_{MBR}
    - T1 $E_{IR,B} = 1$, F2F1F0 = 0,0,0, C_{MAR}
    - T2 Write=1
  - Totalt 7 steg för 1 instruktion (fetch + utför)

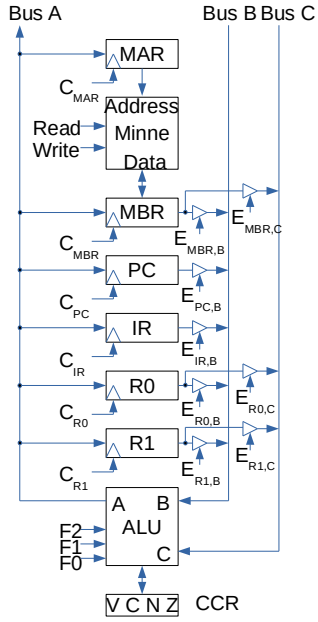


Ytterligare exempel, del 2

- Utför instruktion 101 (SUB R1,R0)
  - R1 = R1 - R0
  - Skicka ut R1 på C-bussen, R0 på B-bussen, beräkna R1-R0 och spara resultatet på A bussen i R1 och uppdatera CCR
  - Motsvarande styrsignaler

    Steg

    T0 $E_{R0,B} = 1$, $E_{R1,C} = 1$, F2F1F0 = 1,1,1, CCR, C_{R1}
  - Denna instruktion tar totalt 5 steg
    - 4 i fetch och 1 i execute

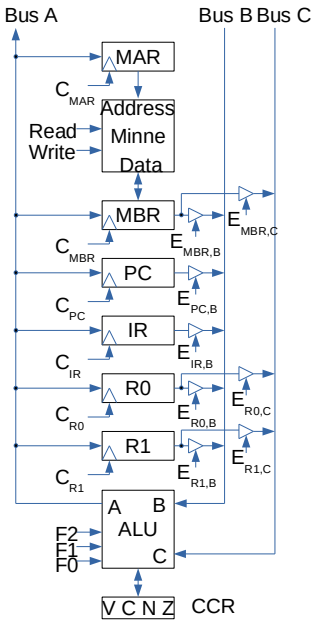


Ytterligare extra exempel, del 2

- Utför instruktion 110 (BRA T)
  - Ett absolut hopp (argument T är adressen nästa instruktion finns på)
  - Skicka ut IR på B-bussen och spara resultatet på A-bussen i PC
  - Motsvarande styrsignaler

    Steg

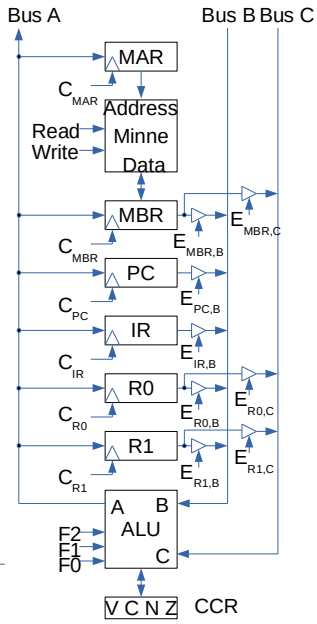
    T0 $E_{IR,B}$ = 1, F2F1F0 = 0,0,0, C_{PC}
  - Denna instruktion tar totalt 5 steg
    - 4 i fetch och 1 i execute



Styrenhet

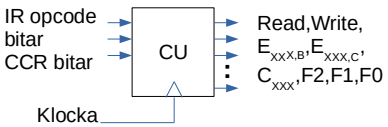
Hantering av tid och ordning

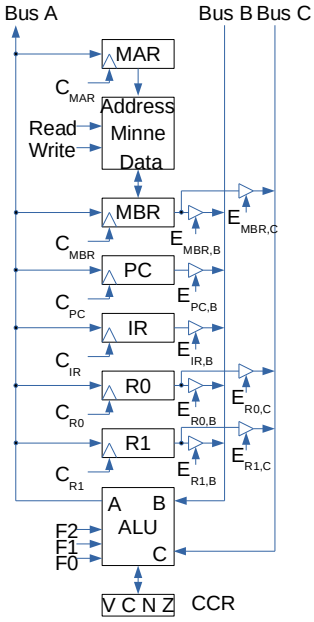
- En sak i taget
  - I varje klockcykel gör varje enhet (bus, ALU etc.) bara en operation (olika för respektive enhet)
    T ex minnet kan inte först läsa och sedan skriva i samma klockcykel
  - Olika saker händer på olika ställen i datorn samtidigt (parallellt)
- Allt styrs av klockan
  - Jfr ”styrdans” eller militär marsch
    - Om inte alla tar ett steg vid samma tidpunkt börjar man trampa varandra på tårna
- Synkron timingmodell
  - Alla enheter får klocksignalen samtidigt



Alla steg måste styras av kontrollenheten (CU)

- Kontrollenheten genererar en sekvens av styrsignaler till datorn
  - Måste hålla ordning på i vilket steg den befinner sig
  - Alla styrsignaler ändras vid varje klockflank
  - Olika sekvenser beroende på instruktion och flaggornas värde



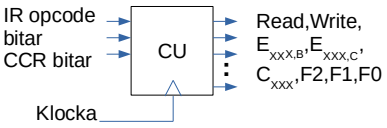


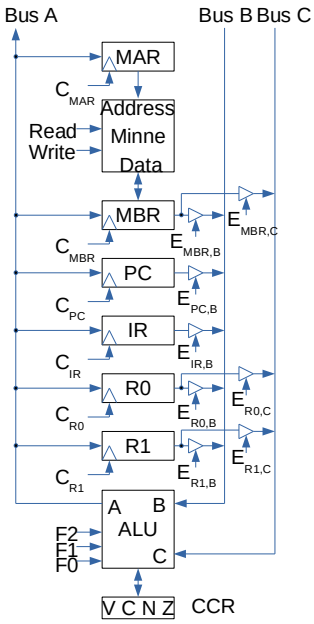
Alla steg måste styras av kontrolleneheten (CU), cont.

- Varje opcode har egen instruktionssekvens

Op-code |	M eller T

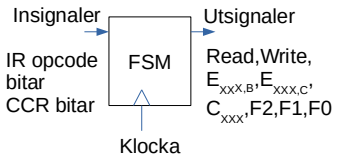
  - Fetchdelen gemensam (IR innehåller instruktionen efteråt)
  - Efter fetch måste rätt sekvens väljas
    - Använd 3 vänstra bitarna i IR-registret

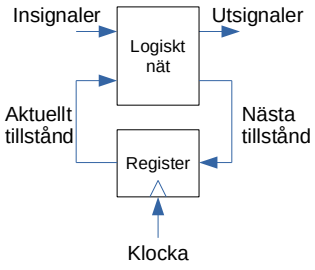




Tillståndsmaskin (Finite State Machine)

- Sekvens av utsignaler genereras med fast frekvens styrd av klocka
  - Utsignal bestäms av aktuellt steg (tillstånd) samt eventuellt av värden på ingång
- Lösning enligt digitalteknik
  - Register för aktuellt tillstånd
    - Klockas med klocka
  - Logiskt nät översätter kombination insignaler och aktuellt tillstånd till utsignaler och nytt tillstånd
  - Logiskt nät kan bli väldigt stort och komplext
    - Antal ingångar till nätet
    - Behöver designas på nytt om något ska ändras
  - ROM (minne som bara läses) kan användas
    - Stort om många insignaler

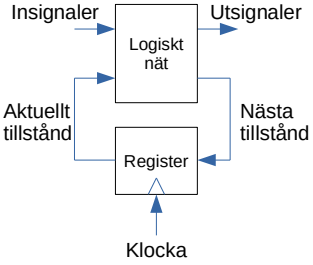


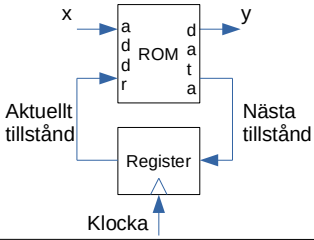


Tillståndsmaskin, exempel

- x=0 ska ge sekvensen y=3141531415314... och x=1 ska ge sekvensen y=2718271827...
- Läsbart minne (ROM) istället för logiskt nät
  - Stor uppslagstabell
  - Adress till ROM: x plus 3 bitar (=8x+aktuellt tillstånd)
  - Data från ROM:y, nästa tillstånd

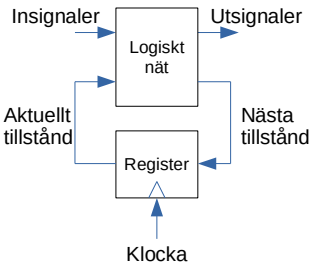
Address (8x+aktuellt)	Data (y, nästa)
0	3 1
1	1 2
2	4 3
3	1 4
4	5 0
5-7	3 1
8	2 9
9	7 10
10	1 11
11-15	8 8

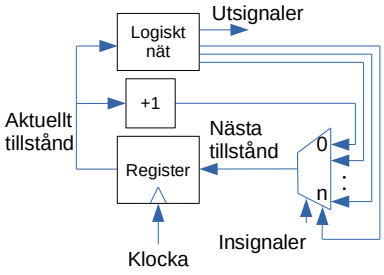




Alternativ implementation av tillståndsmaskin

- Alternativ för att skapa förbestämd sekvens
  - Räknare med minne som översätter till styrsignaler
    - Mycket mindre hårdvara
    - Logiskt nät minst halverat (beror på antal insignaler)
  - Bygg med register och en +1 operation
    - Återanvänd funktion
  - Måste lägga till styrsignal som startar om sekvens (nollställning) alternativt laddar startsignal beroende på insignal

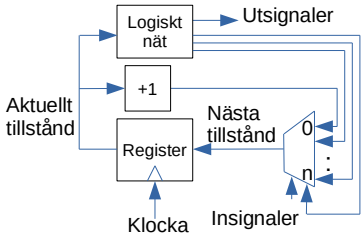


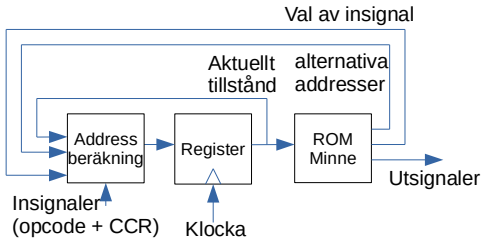


Generell mikroprogrammeringsstruktur

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

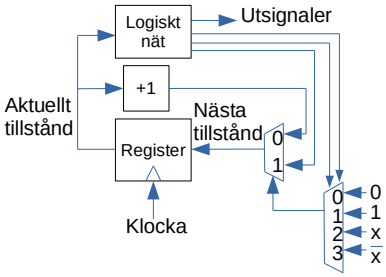
Adress	CCRSelect	Control





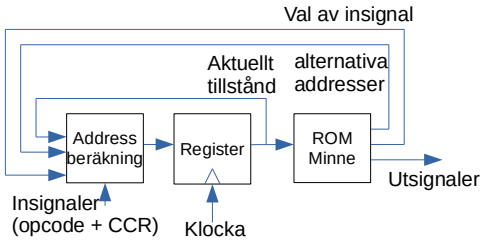
Sekvensexemplet, igen

- Värde ut är kontrollsignaler (3, 1 etc.), insignal (x), 1 adress, 2 styrsignaler väljer inget hopp (0), alltid hopp (1) eller hopp om x=1 respektive hopp om x=0
- Varje steg kan gå till nästa adress om x är samma som förra klockcykeln (fortfarande i samma sekvens)
- Problem vid omstart av sekvens, behöver hoppa till annan adress än +1. Kan inte utföras samtidigt som andra sekvensen startas (har bara en annan adress än +1).
  - Tillstånd 11 => nästa tillstånd måste bli 8 (men om x=0 borde det vara 0 istället!)



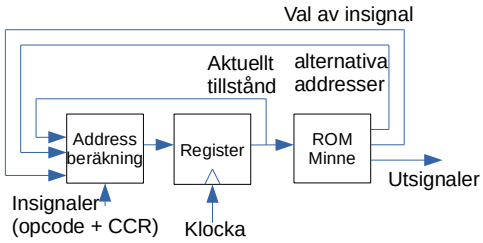
Fördelar med mikroprogrammerad struktur

- Enkel att ändra och bygga ut
  - Öka antal adresser i ROM eller öka antal bitar per adress kan ge nya instruktioner
  - Byt aktiv styrsignal genom att ändra enskilda bitar i minnet
    - Eventuellt möjligt ”patcha” processorinstruktioner!
- Enkel att återanvända konstruktion
  - Ändra bara minnesinnehåll
- Effektiv för stora tillståndsmaskiner



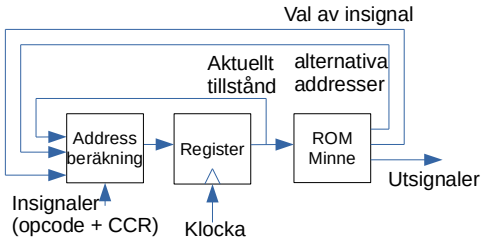
Begränsningar hos mikroprogrammerad struktur

- Begränsat antal olika förgreningar från varje tillstånd
  - Problem med avkodning av opcode? Går att fixa.
- Begränsat antal samtidiga villkorliga CCR flaggor testas per klockcykel
- Utsignaler beror bara på indata från föregående klockcykel
  - Digitalteknik: Mealy FSM reagerar inom samma klockcykel



Generell mikroprogrammeringsstruktur, forts.

- Avvägning mellan storlek på ROM och möjlig funktion
  - Fler möjliga insignalskombinationer/nästa tillstånd kräver fler alternativa adresser och insignalval => större ROM
  - Få alternativa adresser => avkodning av t ex opcode tar längre tid
    - Kräver sekvens av jämförelser, bit för bit av opcode
  - Effektivare lösning: uppslagstabell för nästa tillstånd baserat på opcode-bitarna

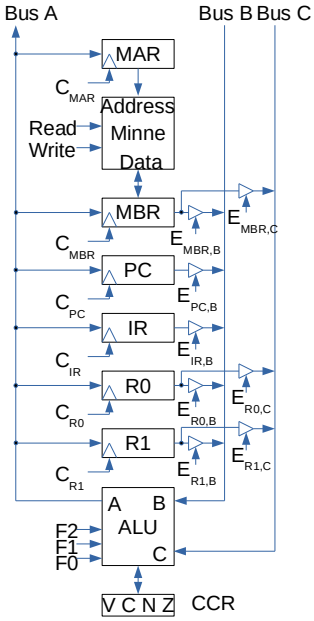


Maskininstruktioner

- Exempel på maskininstruktioner som ska kunna utföras

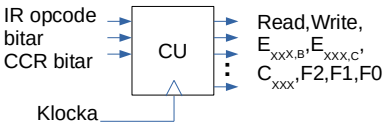
Op-code	Namn		Operation
0 0 0	LOAD	R0,M	[R0] <- [M]
0 0 1	LOAD	R1,M	[R1] <- [M]
0 1 0	STORE	M,R0	[M] <- [R0]
0 1 1	STORE	M,R1	[M] <- [R1]
1 0 0	ADD	R1,R0	[R1] <- [R1] + [R0]
1 0 1	SUB	R1,R0	[R1] <- [R1] – [R0]
1 1 0	BRA	T	[PC] <- T
1 1 1	BEQ	T	Om [Z] = 1 då [PC] <- T

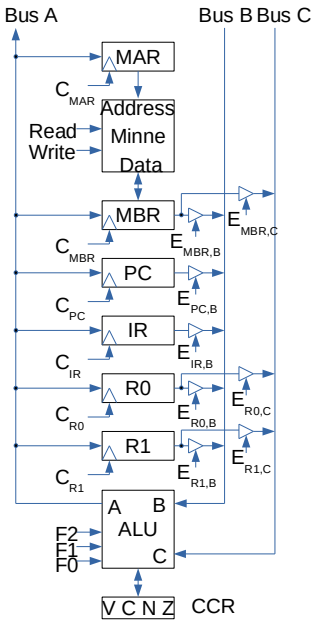
- Mikroprogrammet beskriver för varje instruktion alla styrsignaler och steg för implementering av instruktionen
  - Hämta instruktion (fetch)
  - Utför instruktion (beror på vilken instruktion som hämtats)



Processorarkitektur

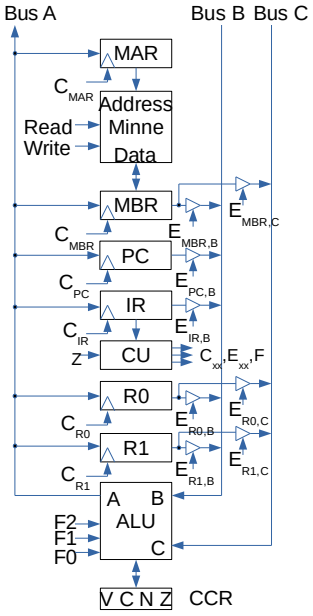
- Exempel baserad på kapitel 7
  - Bussar, register, minne, ALU
- Bilden saknar en styrenhet (CU)
  - Måste generare alla styrsignal
  - Varje styrsignal har ett värde per klockcykel





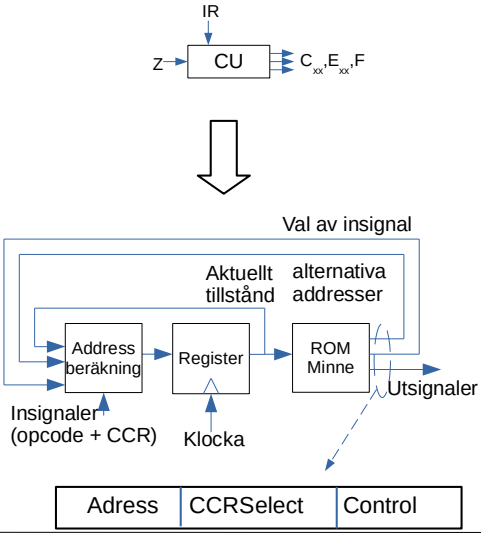
Processorarkitektur inkl CU

- 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



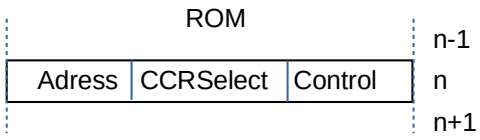
Generell mikroprogrammeringsstruktur

- 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



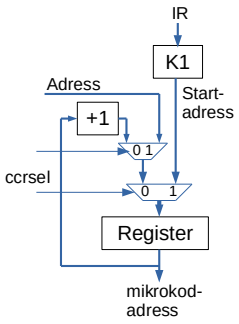
Mikrokodsinstruktionens delar

- CCRSelect (ccrsel) väljer vilken nästa instruktion blir
  - Två att välja bland:
    - n+1
    - Hopp till Adress
  - Implementera med mux styrd av bit i CCRSelect (0 => +1, 1 => hopp till Adress)
  - Villkorligt hopp till Adress
    - CCRSelect väljer vilken flagga som bestämmer om hoppet ska tas
- 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



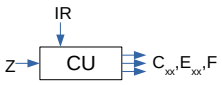
Implementering av opcodeavkodning

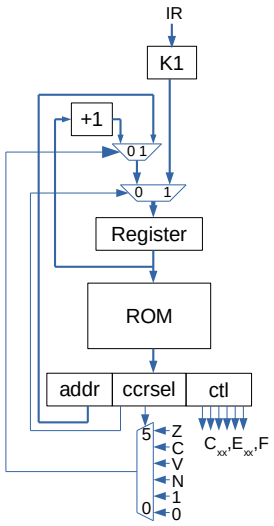
- Antag instruktionen hämtats till IR-registret
- Nästa steg ska bero av alla 3 opcode-bitarna
  - 8 olika möjliga startadresser
  - Använd en tabell (dvs ett litet ROM-minne) som lagrar dessa adresser
    - Slå upp rätt adress mha opcodebitarna
  - Använd den uppslagna adressen på motsvarande sätt som för vanliga hopp
    - Implementera med en extra mux som väljer tabelladress eller vanliga hoppfältet från mikrokoden



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 till K1 som svarar med startadress i ROM för kod specifik för en viss instruktion

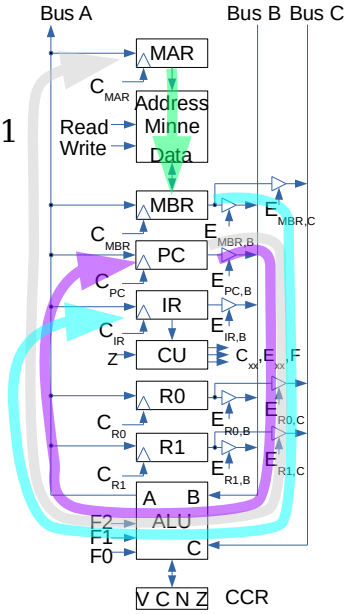




Exekvering av instruktion, del 1

- 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)

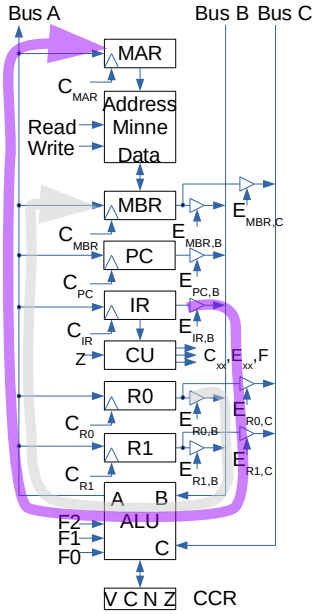
	uaddr	ccrsel	kontrollsignal	Addr
○	0	+1	$E_{PC,B}$ = 1, F2F1F0 = 0,0,0, C_{MAR}	-
●	1	+1	$E_{PC,B}$ = 1, F2F1F0 = 0,1,0, C_{PC}	-
●	2	+1	Read=1,C_{MBR}	-
●	3	+1	$E_{MBR,B}$ = 1, F2F1F0 = 0,0,0, C_{IR}	-
	4	K1	-	-



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)

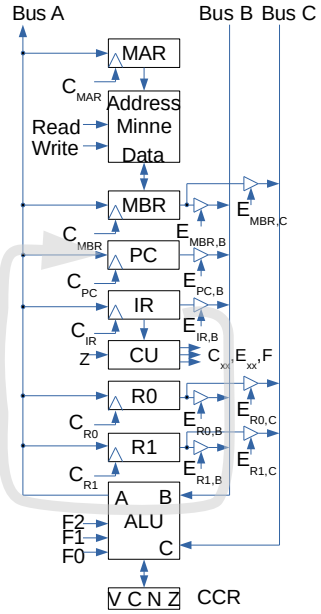
	uaddr	ccrsel	kontrollsignal	Addr
○	k	+1	$E_{R0,B}$ = 1, F2F1F0 = 0,0,0, C_{MBR}	-
●	k+1	+1	$E_{IR,B}$ = 1, F2F1F0 = 0,0,0, C_{MAR}	-
●	k+2	1	Write=1	0



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

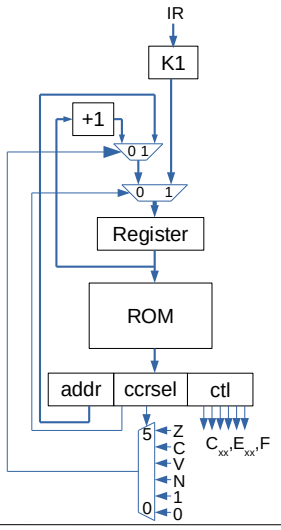


Mikroprogram för exemplet

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

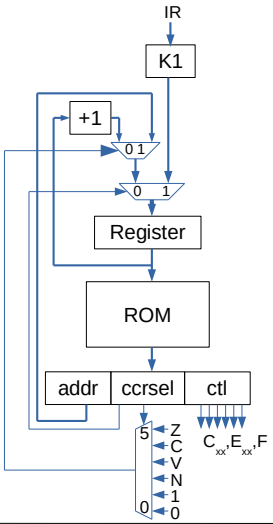


Mikrokod, minnet K1

- 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