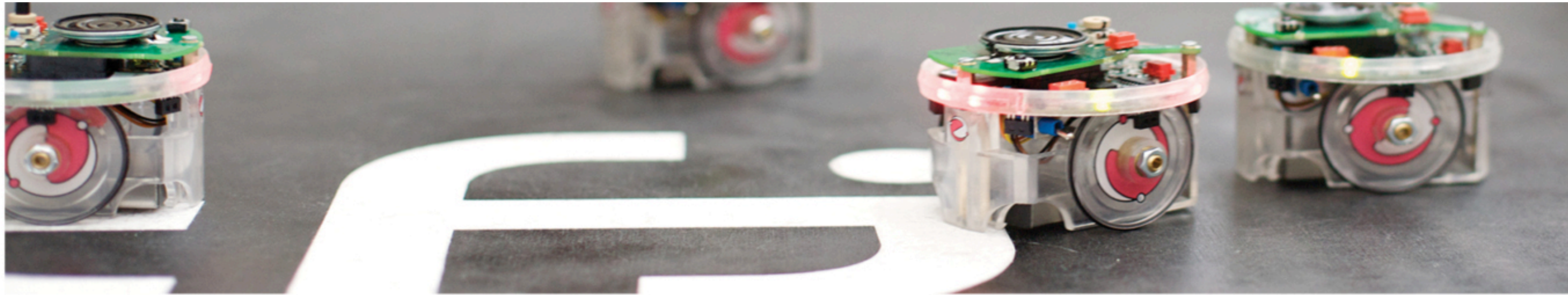


UiO : Institutt for informatikk
Det matematisk-naturvitenskapelige fakultet

INF2270

Datamaskin Arkitektur

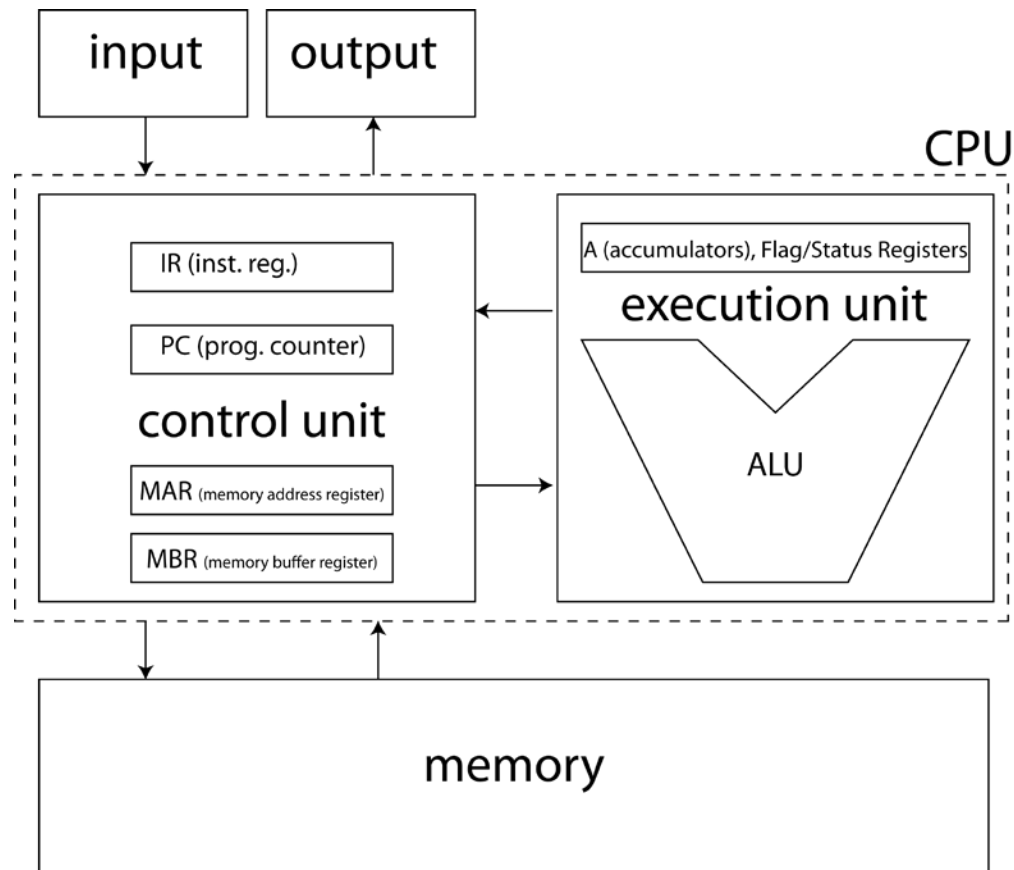
Hovedpunkter

- Von Neumann Arkitektur
- ALU
- Minne
  - SRAM
  - DRAM
  - RAM Terminologi
  - RAM Signaler
- Register
  - Register overføringsspråk

Von Neumann Arkitektur

John von Neumann publiserte i 1945 en model for datamaskin arkitektur som brukes fortsatt den dag i dag. Hovedbidraget hans og unikheten i denne modellen er å bruke et enkelt minneelement som både skal brukes for program og data.

Von Neumann Arkitektur Block Diagram



- IR: (instruksjonsregister)

  Her lagres maskinkoden av instruksjonen som skal eksekvereres

- PC: (program counter / IP (instruksjon pointer))

  Her lagres minneadressen for den neste maskinkode instruksjonen.

- MAR: (memory address register)

  Halvparten av registeret er dedikert til å lagre minneadressen som det skal leses fra eller skrives til fra CPU

- MBR: (memory buffer register)

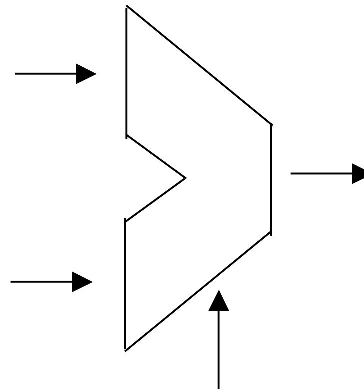
  Den andre halvpart av registeren som lagrer selve dataen som er lest fra minne eller skal skrive inn på minne fra CPU.

Data- og instruksjonsbus

- Bus er kommunikasjonskanalen mellom registere, funksjonelle enheter (ALU), minne og I/O enheter.
- Bus kan deles mellom flere enheter, men kun en som kan sende av gangen.
- I en von Neumann arkitektur er det en bus mellom minne og CPU som skal både overføre instruksjon og data, dette vil da være flaskehalsen.
- Internt i en CPU er den en eller flere bus(er) som overfører data mellom interne registere.

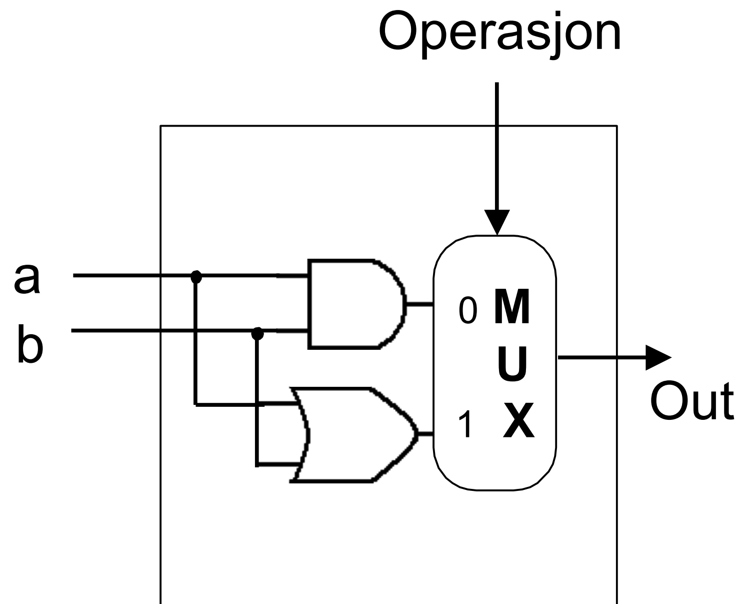
Aritmetisk logisk enhet (ALU)

- Den delen av CPU hvor logiske og aritmetiske beregninger utføre, f.eks addisjon, subtraksjon, AND, OR osv…

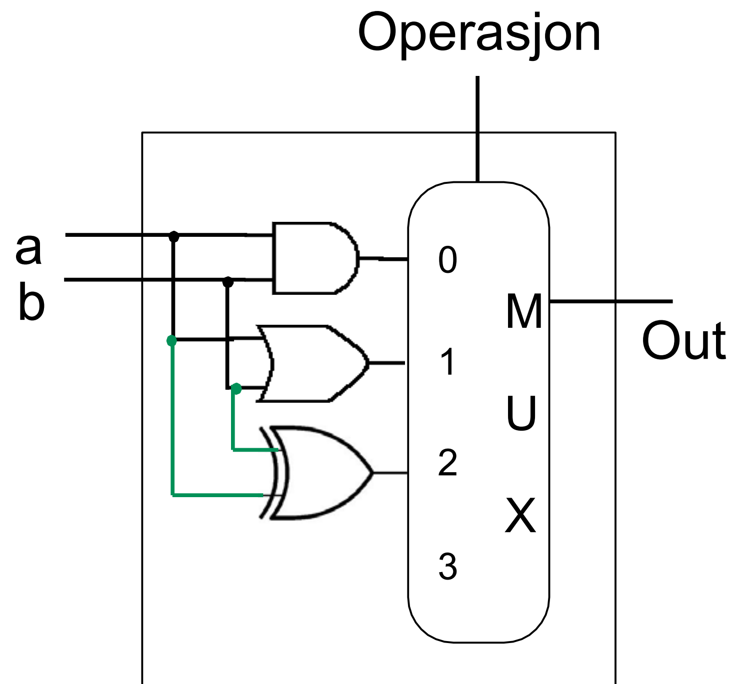


- Operasjoner som ALU skal utføre: (eksempelvis)
  - AND
  - OR
  - XOR
  - Addisjon
  - Subtraksjon

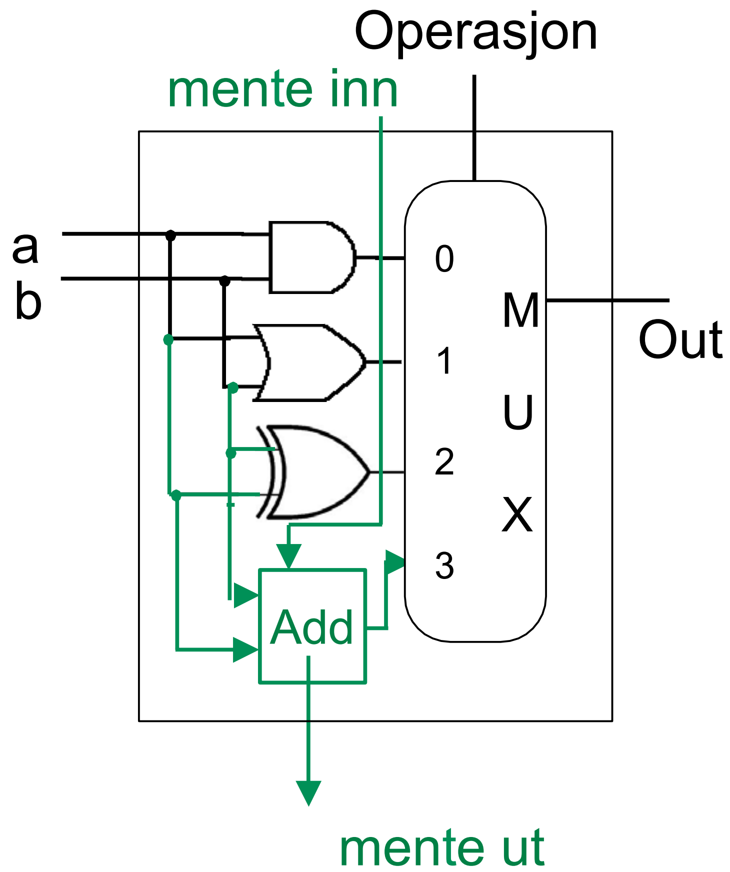
- Følgende byggeblokker trenger vi:
  - Fulladder
  - Multiplekser
  - AND,OR og NOT



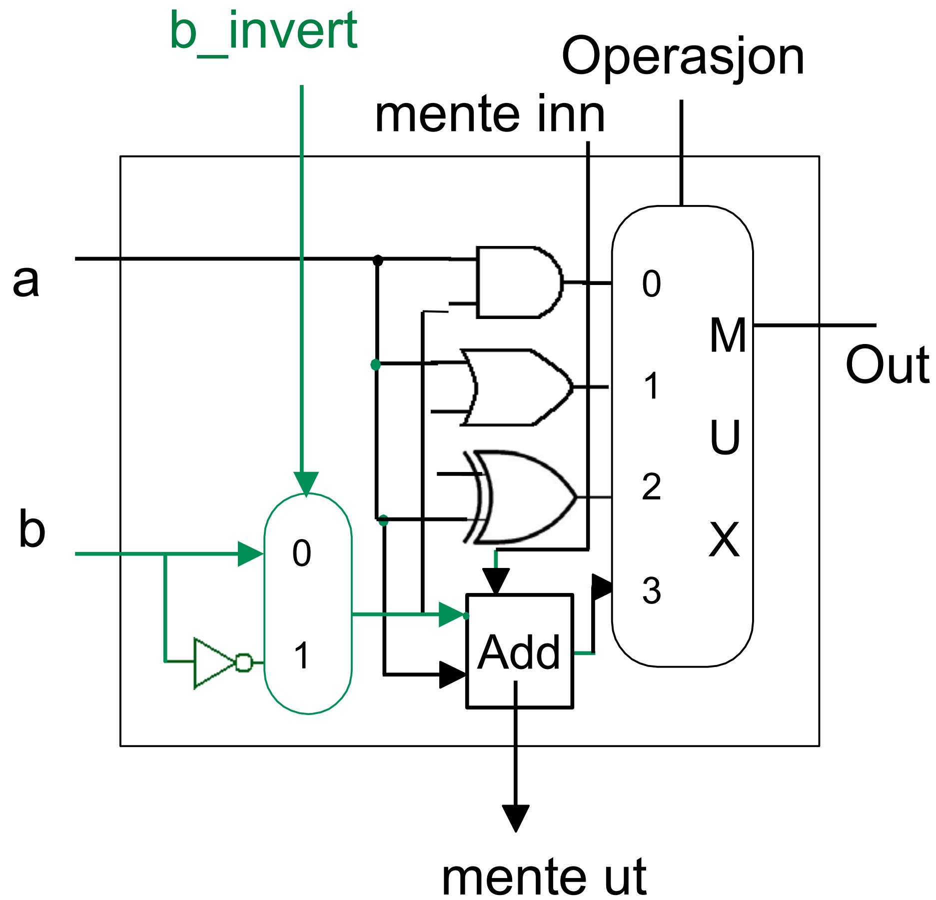
Operasjon	Out
0	AND (ab)
1	OR (a+b)



Operasjon		Out
0	0	AND
0	1	OR
1	0	XOR
1	1	?



Operasjon		Out
0	0	AND
0	1	OR
1	0	XOR
1	1	SUM

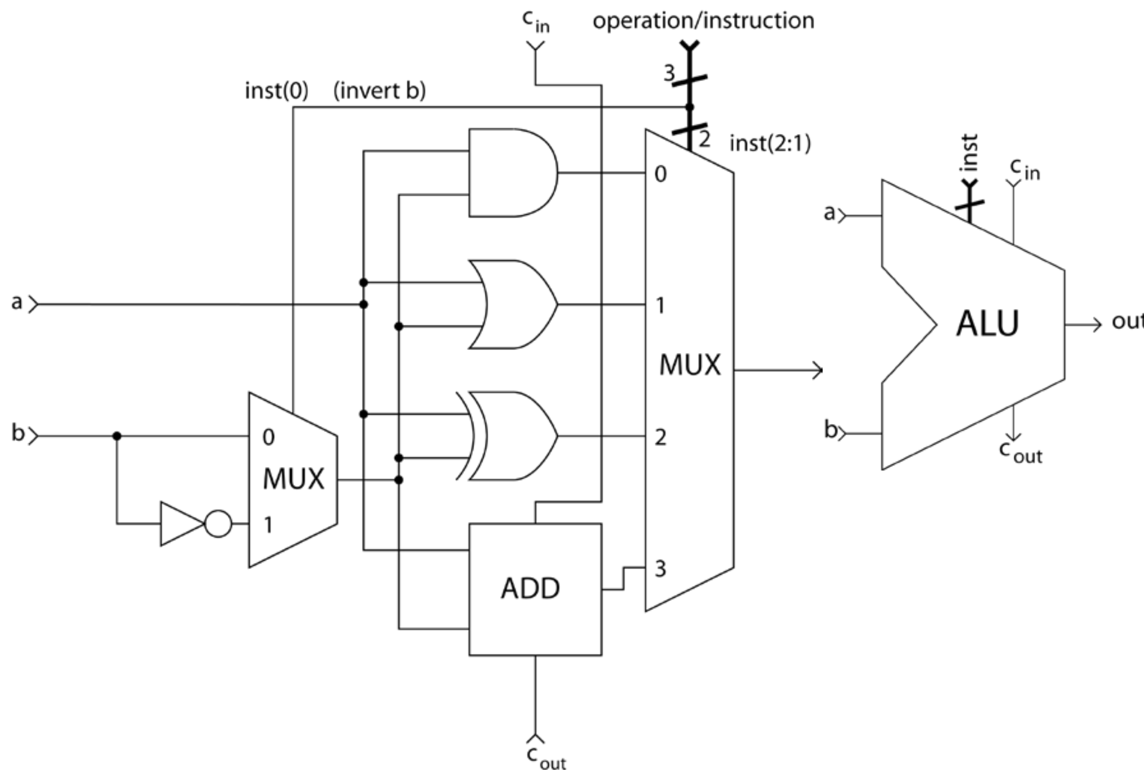


Operasjon = a2a1a0
a2 = b_invert
a1 = MSB MUX
a0 = LSB MUX

Operasjon	Out
000	ab
001	a+b
010	a XOR b
011	a SUM b
100	ab´
101	a+b´
110	a XOR b´
111	a SUB b

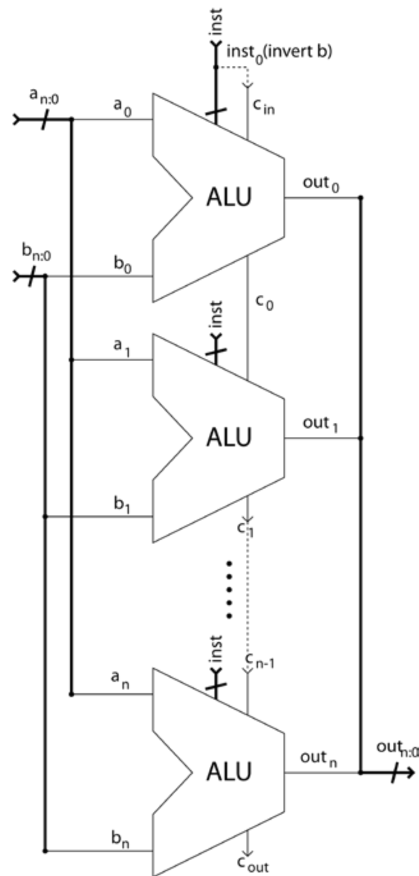
1-bit ALU

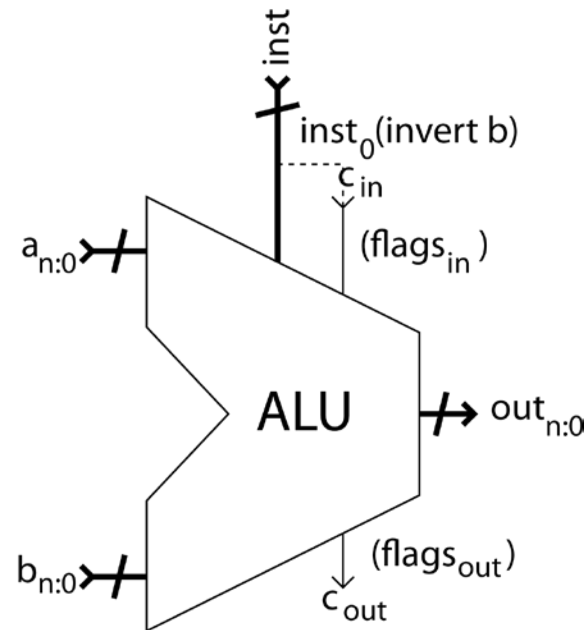
inst = a2a1a0
Inst(2) = a2 invert b
Inst(1) = a1 MSB MUX
Inst(0) = a0 LSB MUX



inst	computation
000	a · b
001	a · b´
010	a + b
011	a + b´
100	a ⊕ b
101	a ⊕ b´
110	a + b
111	a - b

N-bit ALU

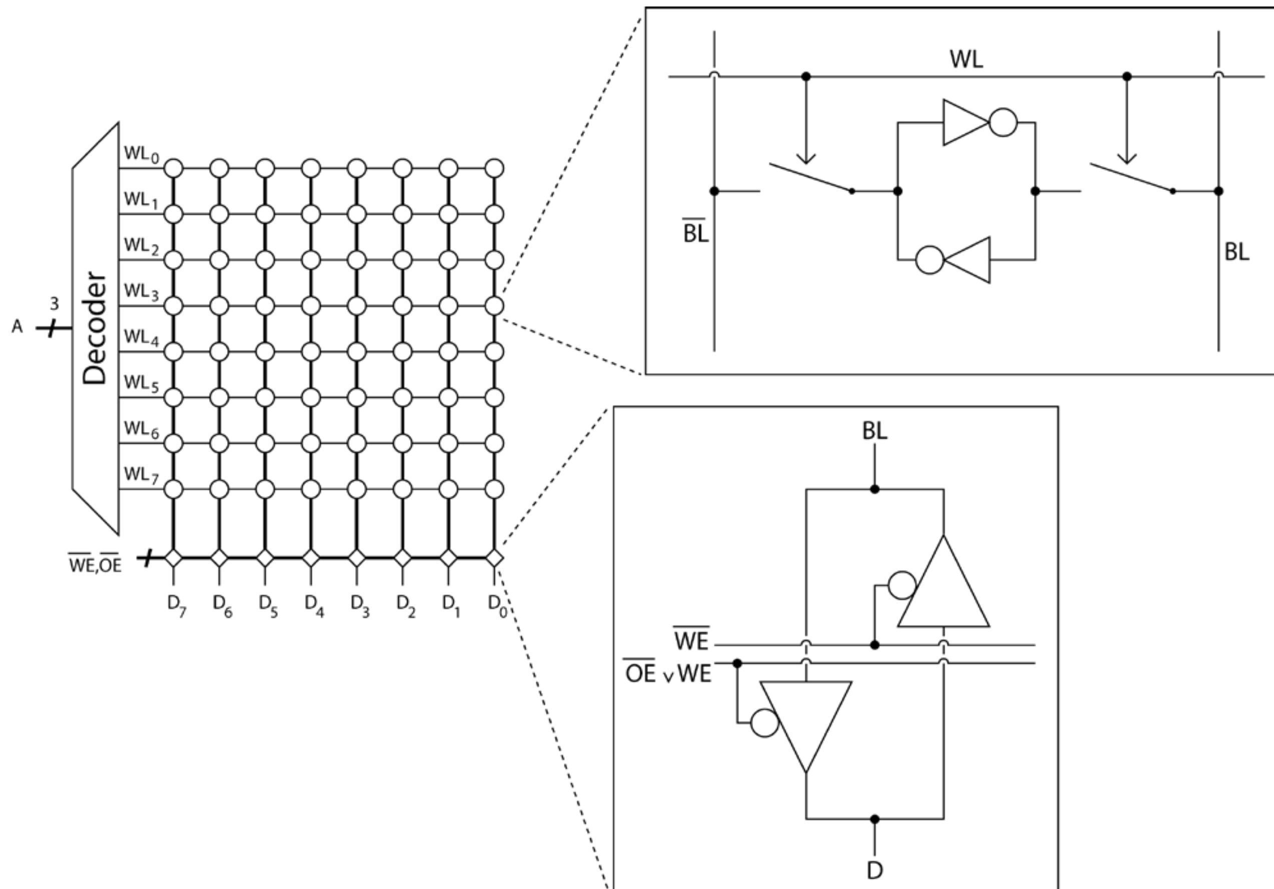




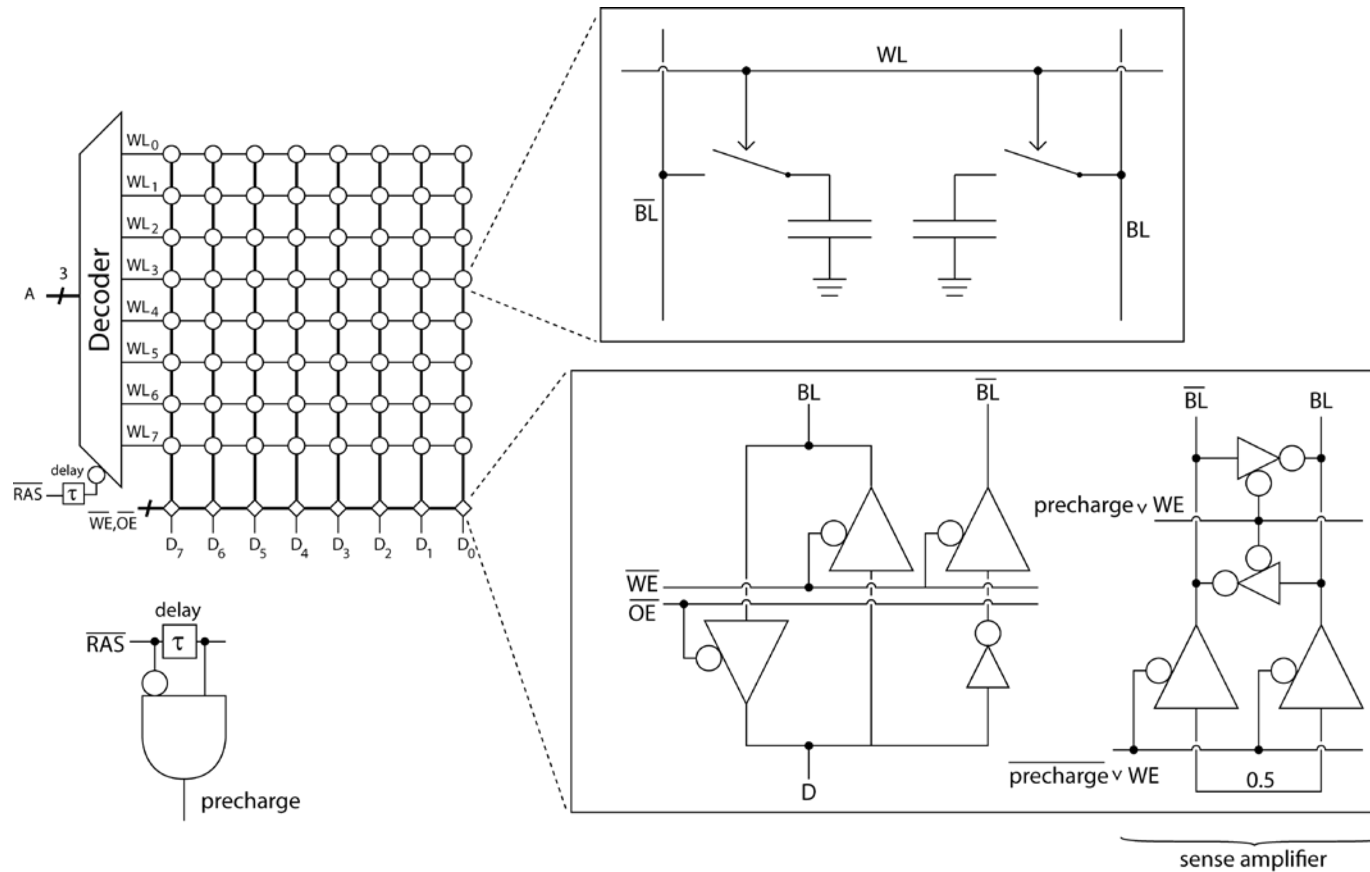
ALUer i CPU

- Moderne CPU kan inneholde flere ALUer
- ALUer kan designes til å håndtere flere funksjoner per trinn og mer komplekse.
- Alternativt kan man bruke software aktivt til å designe slik at en ALU kan utføre komplekse operasjoner over flere trinn.
- TRADE-OFF: hvor skal man plassere kompleksiteten, i hardware eller software
- Ulemper med å sette kompleksitet i hardware er høyere kostnader i fbm strømforbruk, areal og produksjonskost.
- ALU designet spiller en stor rolle med hensyn på CPU sin ytelse, da det mest komplekse operasjonen setter maksimal klokkefrekvens.

Static Random Access Memory SRAM



Dynamic Random Access Memory - DRAM



RAM terminologi

Address space:

Størrelsen på adresserbare minne enheter, oppgis enten i word eller bytes.

Word length:

Antall bit som kan bli lest/skrevet i en operasjon

Memory size:

Størrelsen er gitt av address space * word length.

RAM Signaler

$\overline{\text{WE}}$, write enable:

Som oftest aktiv lav. Brukes til å skille mellom read og write aksessering.

$\overline{\text{RAS}}$/$\overline{\text{CAS}}$, row/column access strobe:

Brukes primært i DRAM til å styre decodere for henholdsvis rad og kolonne

$\overline{\text{OE}}$, output enable:

Brukes til å kunne regulerer tilgangen til bus(en), da bus(en) kan være drevet av andre enheter.

$\overline{\text{CS}}$, chip select:

En styresignal som tillater å bruke flere RAM(er) istedenfor bare en på samme adresse bus.

DRAM refresh!

- Kapasitiv (Capacitive) lagring er ikke selvbærende slik som flip-floper. Grunnet de elektriske egenskapene til en kapasitans gjør at den “lekker” og derav vil verdien gradvis forsvinne med tid.
- For å kunne opprettholde riktig verdi er man nødt til å “friske” (refresh) opp verdien.

Static vs. Dynamic RAM

	SRAM	DRAM
Aksessering hastighet	+	-
Minne tetthet	-	+
Behov for "refresh"	+	-
Kompleksitet mht interne styresignaler	+	-
Pris pr bit	-	+

Register Transfer Language (RTL)

Symbol	Forklaring
X	Registrer X
[X]	Innholdet i X
$\leftarrow$	Eksekverer
M(	Minne M
[M([X])]	Innholdet i minne gitt av adressen *[X]*

RTL Eksempel

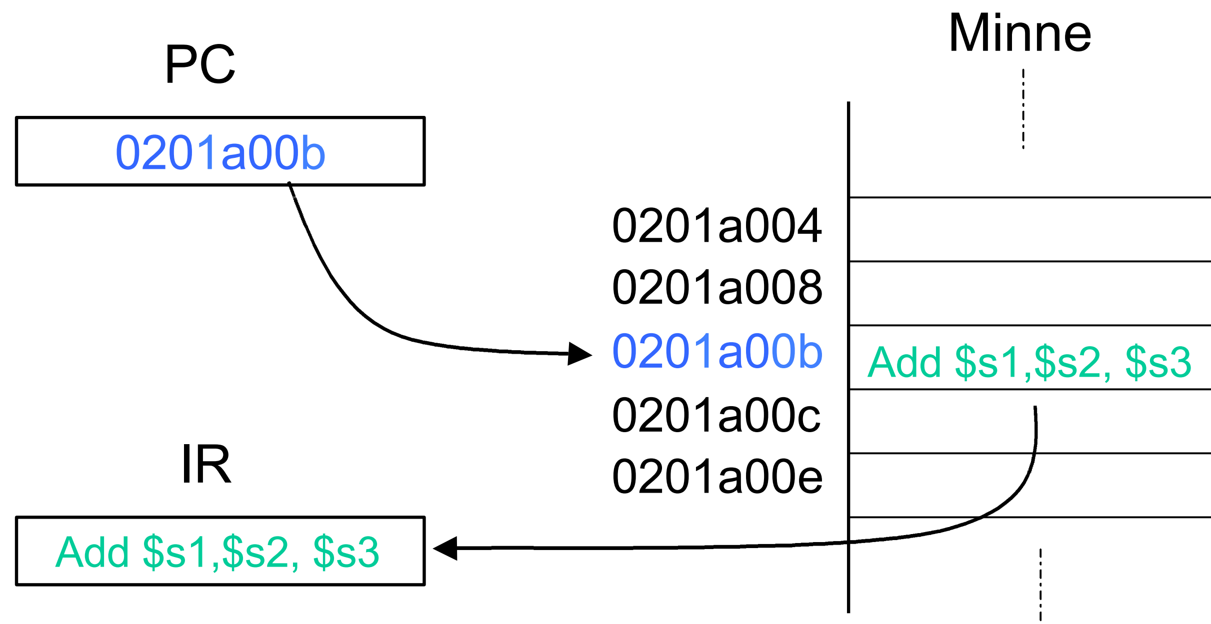
$[IR] \leftarrow [MBR]$ Innholdet gitt av MBR blir overflyttet/overskrevet innholdet gitt av IR

Eksekvering av instruksjoner

- Hver instruksjon består av minst tre oppgaver:
  1. Hent instruksjonen (“Fetch)
  2. Dekod instruksjonen (“Decode)
  3. Utfør instruksjonen (“Execute)

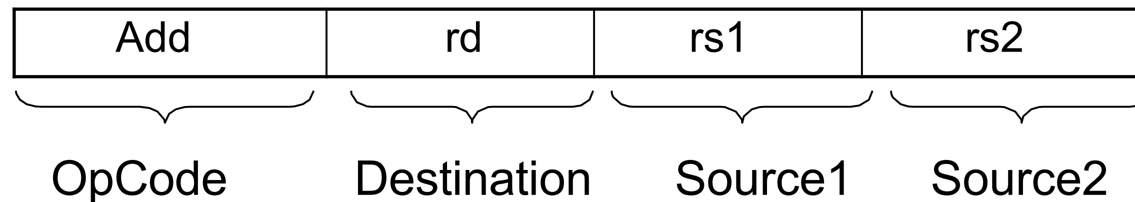
Fetch

- Henter instruksjonen fra minnet og legger den i insruksjonsregisteret IR



En ny instruksjon hentes fra minne for hver start på en instruksjonssykel. En tilstandsmaskin (FSM) i kontrollenheten (CU) genererer de riktige styresignalene.

Decode

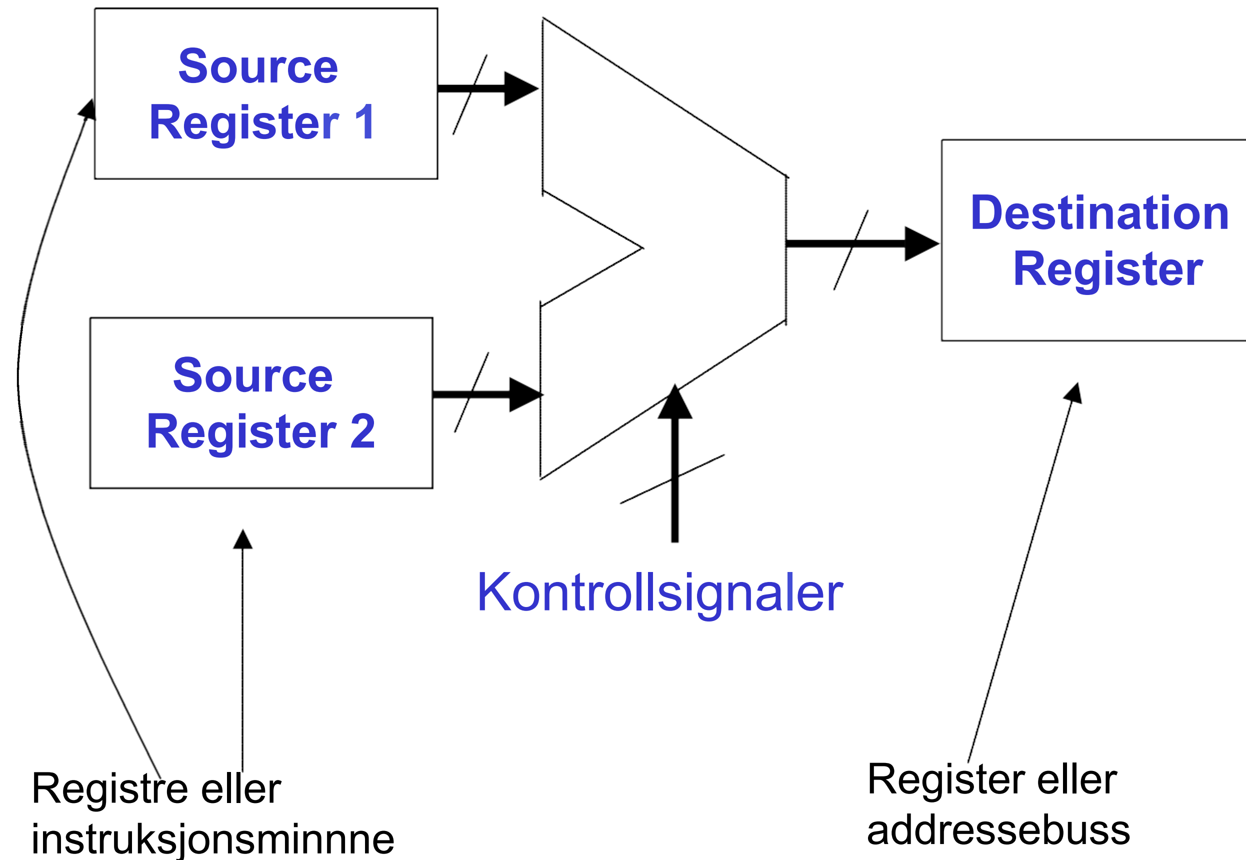


OpCode:	Instruksjonstypen
Destination:	Resultatet plasseres her
Source 1:	1. operand til instruksjonen
Source 2:	2. operand til instruksjonen

Ikke alle instruksjoner har alle feltene, og noen har flere felter i tillegg.

I tillegg må kontrollsignaler til bla. ALU settes slik instruksjonen krever.

Execute



OBS! Sourceregisterne kan ikke trigges av samme klokkeflanke som destinasjonsregisteret.

Eksempel på eksekvering av Maskinkode

Ved oppstart av en CPU vil program telleren (PC) peke til / være initiert til en spesifikk minneadresse.

Her i dette eksemplet er dette 0.

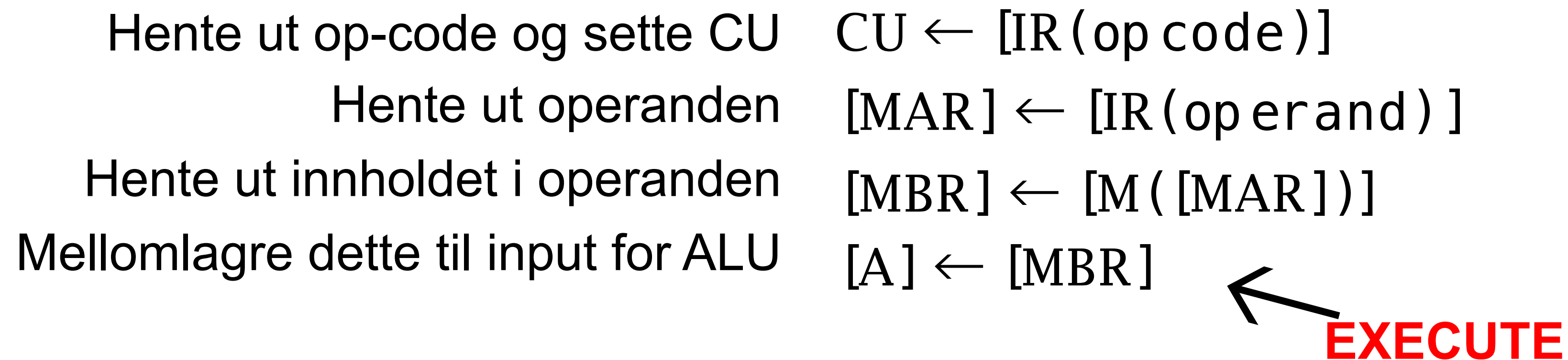
Mem adr	Innhold
0	move 4
1	add 5
2	store 6
3	stop
4	1
5	2
6	0
.	.

Fetch

Henter minneadressen	$[MAR] \leftarrow [PC]$ (nå er $[PC]=0$)
Øker programtelleren	$[PC] \leftarrow [PC] + 1$
Henter ut innholdet i minne	$[MBR] \leftarrow M([MAR])$
Overskriver innholdet i IR med [MBR]	$[IR] \leftarrow [MBR]$

Decode

Instruksjonen i IR er nå “move 5”



Nå må vi hente ut neste verdi for ALU inngangen

Decode forts.

Fetch og decode som før

[MAR] ← [PC] (nå er [PC]=1)

[PC] ← [PC] + 1

[MBR] ← M([MAR])

[IR] ← [MBR]

Instruksjonen i IR er nå “add 5”.

CU ← [IR(opcode)]

Decode forts.

$[MAR] \leftarrow [IR(operand)]$
$[MBR] \leftarrow [M([MAR])]$

Nå har vi i [A] som er verdien i adresse 4 som er tallet 1
$ALU \leftarrow [A]; ALU \leftarrow [MBR]$
$[B] \leftarrow ALU$

EXECUTE

Nå må vi hente ut neste instruksjon som forteller hvor dette skal lagres

Decode forts.

Fetch og decode som tidligere vist

....

....

....

....

....

[MBR] ← [B]
[MAR] ← [IR(operand)]
[M([MAR])] ← [MBR]

Resultat

- Nå er resultatet skrevet i minneadresse 6.

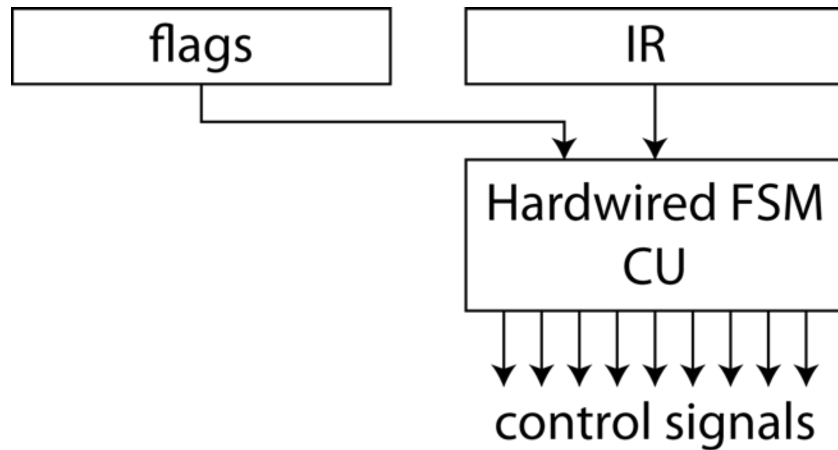
Mem adr	Innhold
0	move 4
1	add 5
2	store 6
3	stop
4	1
5	2
6	3
.	.

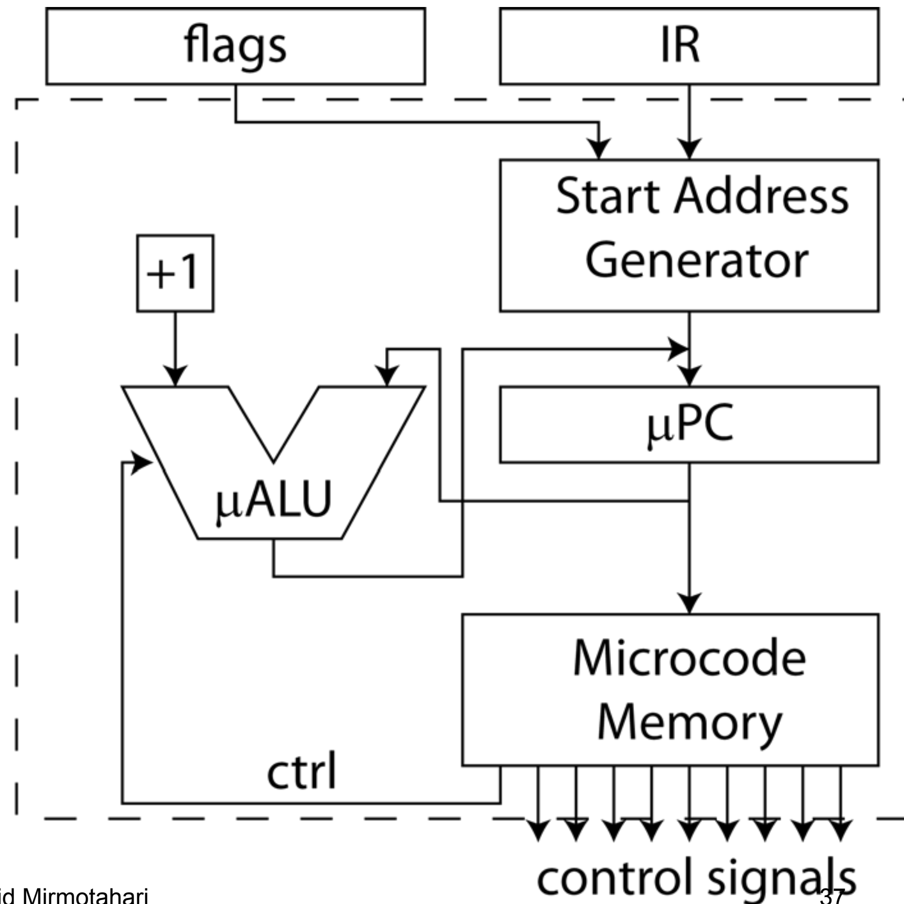
RISC vs CISC

Hittil har vi sett på **hardwired CU** arkitektur, hvor det er en **hardwired FSM** som gir de styresignalene decodet ut fra IR.

En mer fleksibel alternativ er å bruke **microcode** og en enkel prosessor som genererer styresignalene gjennom **microinstruction** i **microprogram memory** (typisk ROM)

Hardwired vs. Microprogrammed CU





Fordeler og ulemper

	Microarchitecture	Hardwired
Occurrence	CISC	RISC
Flexibility	+	-
Design Cycle	+	-
Speed	-	+
Compactness	-	+
Power	-	+

Men det skal sies at nå i moderne CPU så er avstanden mellom RISC og CISC blitt mye mindre