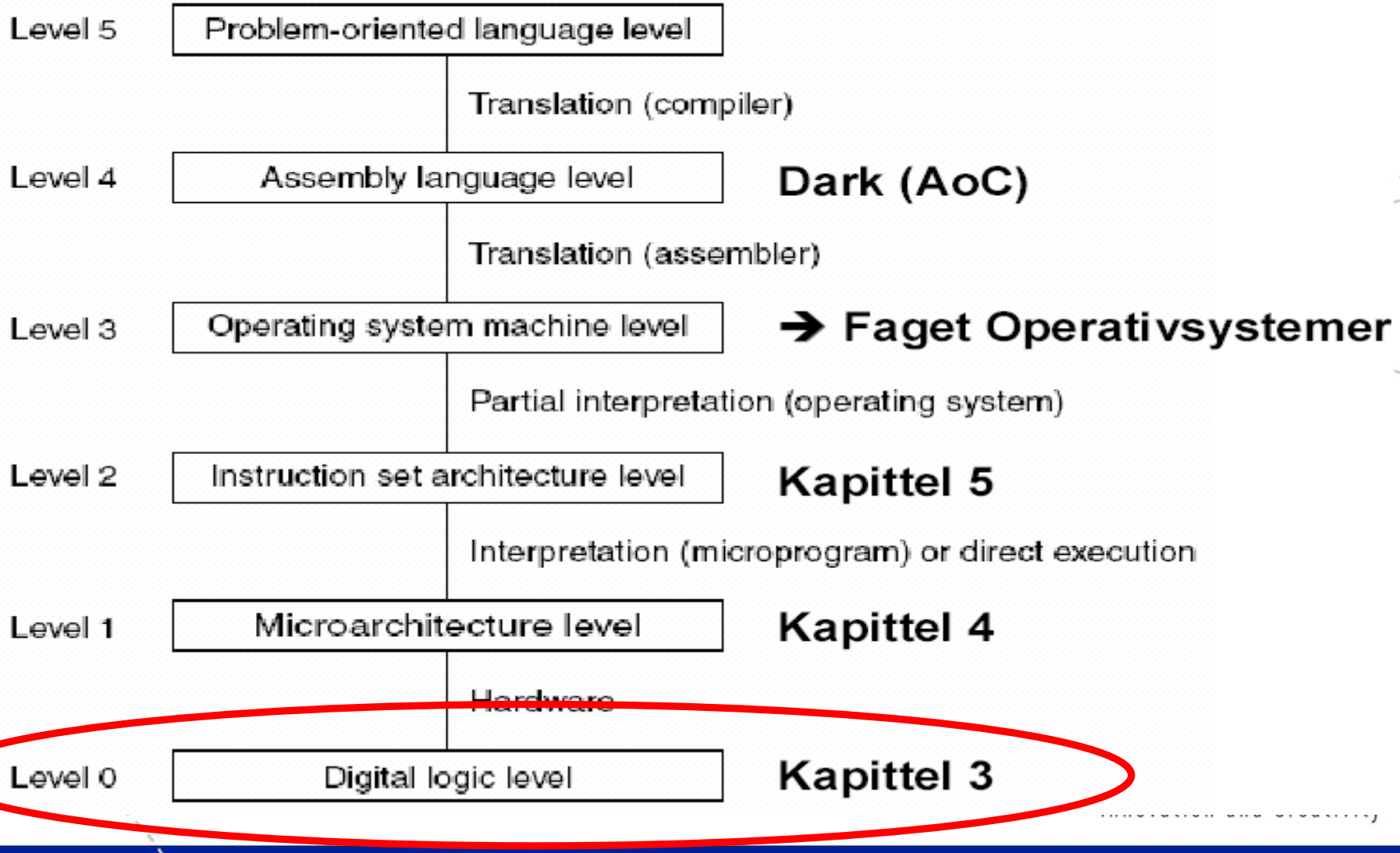
Spørjetime 2008

NTNU
Innovation and Creativity

Spørsmål (botn opp)

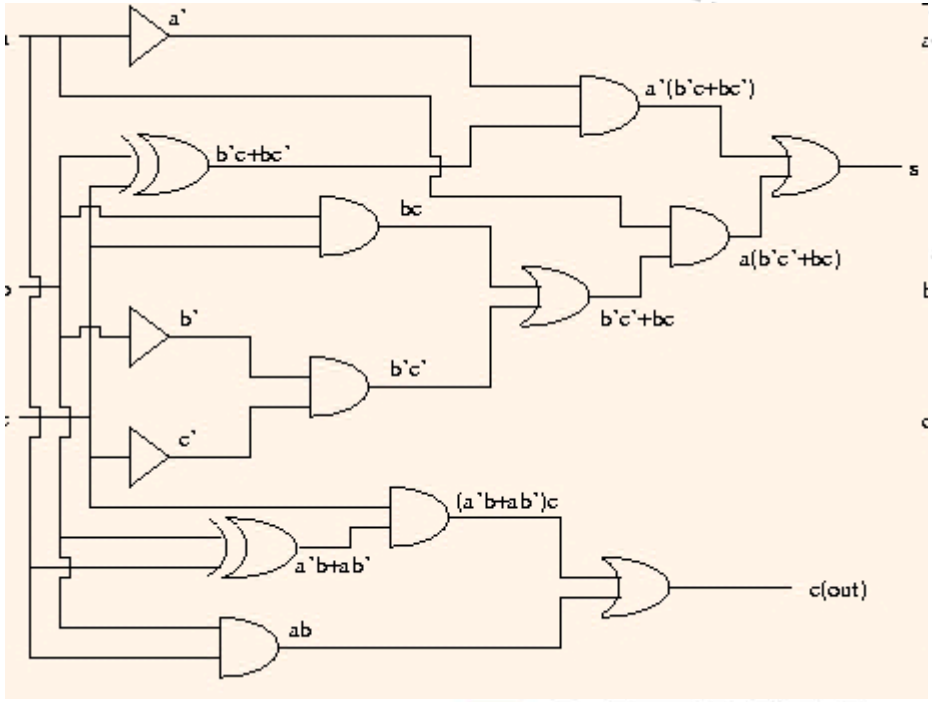
- Fulladder, halvadder etc
- Avbrudahandtering 1
- Adressedekoding
- Generelle I/O-berekningar (rekne klokke antal bit)
- DMA
- Arbitrering
- IJVM (litt om alt)
- IJVM-mikroprogramm (som mange har forstått, eksamensrelevant)
- IJVM-implementasjonar
- Samleband
- Avbrudahandtering 2
- CISC-RISC
- Addresseringsmodi
- Superscalareprosessorar
- Minne i SIMD og MIMD
- Big endian/little endian

Kapittel 3: Digital logic level



Nivå 0: Digtalekretsar

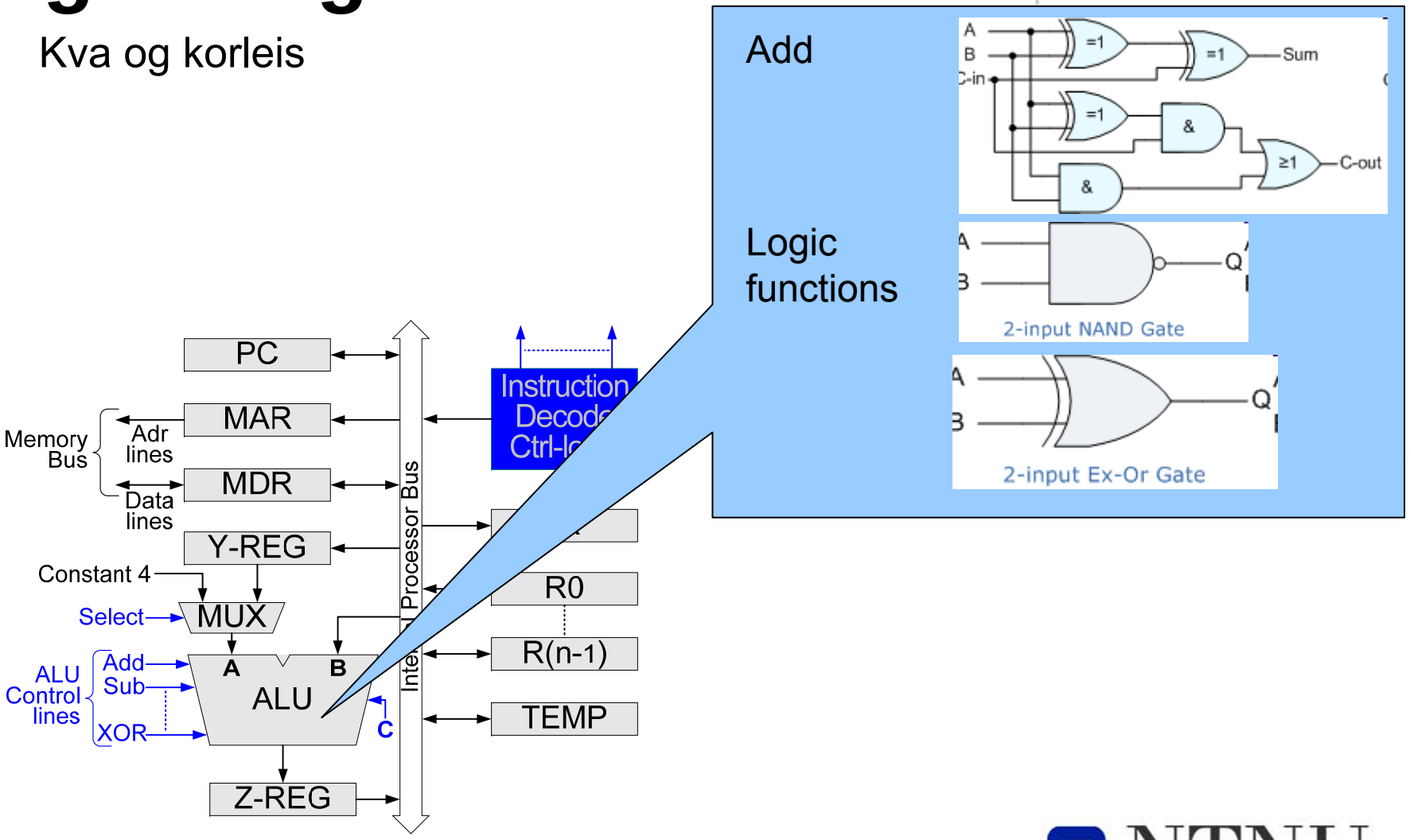
- Fundamentale komponentar
  - AND, OR, NOT,NAND, NOR XOR porter
  - D-vipper for lagring av eit bit
- Samansette komponentar
  - Aritmetiske kretsar –
    - adderere, skiftere, …
  - Dekodere
  - Multiplekser
  - Registre
    - 8, 16, 32, 64 vipper



NTNU
Innovation and Creativity

Digital logic level

- Kva og korleis

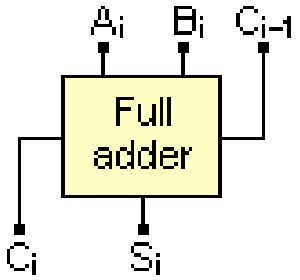


Kort repetisjon 3.0 -> 3.3.4

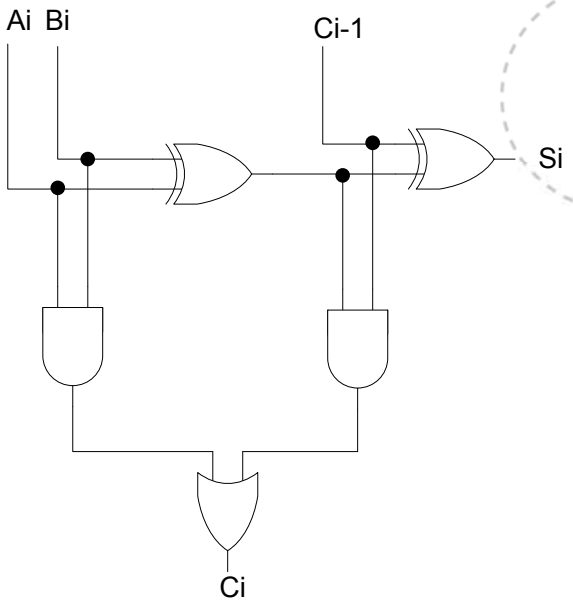
- Kan lage kombinatoriske kretsar
  - Kretsar uten klokke eller sekvensielle eigenskapar
    - Decoders
    - Encoders
    - Multiplexers
    - Demultiplexers
    - **Binary Adders**
    - Binary Subtractors

- Eksempel kretsar: Full adder

$$\begin{array}{cccc} 0 & 0 & 1 & 1 \\ +\,0 & +\,1 & +\,0 & +\,1 \\ \hline 0 & 1 & 1 & 10 \end{array}$$



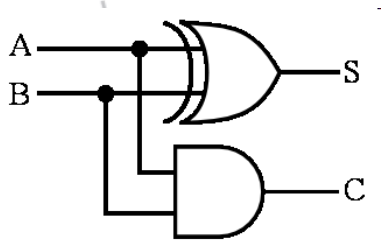
A_i	B_i	C_{i-1}	S_i	C_i
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1



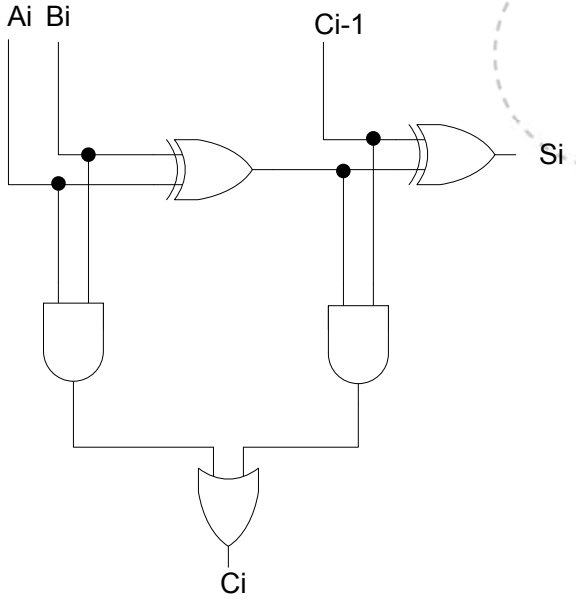
NTNU
Innovation and Creativity

- Halvadder
  - Adderar 1 bit nummer A og B, svar: S og mente
    - A + B = S
    - 0 + 0 = 0
    - 0 + 1 = 1
    - 1 + 0 = 1
    - 1 + 1 = 0
  - Carry mente bit (C)
    - 1 + 1 = 1
    - 0 + DC = 0
    - DC + 0 = 0
- Fulladder
  - Adderar 1 bit nummer A og B med mente Ci, svar: S og mente ut Co
    - A +B+ Ci = S og Co
    - 0 + 0 + 0 = 0 og 0
    - 0 + 0 + 1 = 1 og 0
    - 0 + 1 + 0 = 1 og 0
    - 0 + 1 + 1 = 0 og 1
    - 1 + 0 + 0 = 1 og 0
    - 1 + 0 + 1 = 0 og 1
    - 1 + 1 + 0 = 0 og 1
    - 1 + 1 + 1 = 1 og 1

- Eksempel kretsar: Halvadder adder



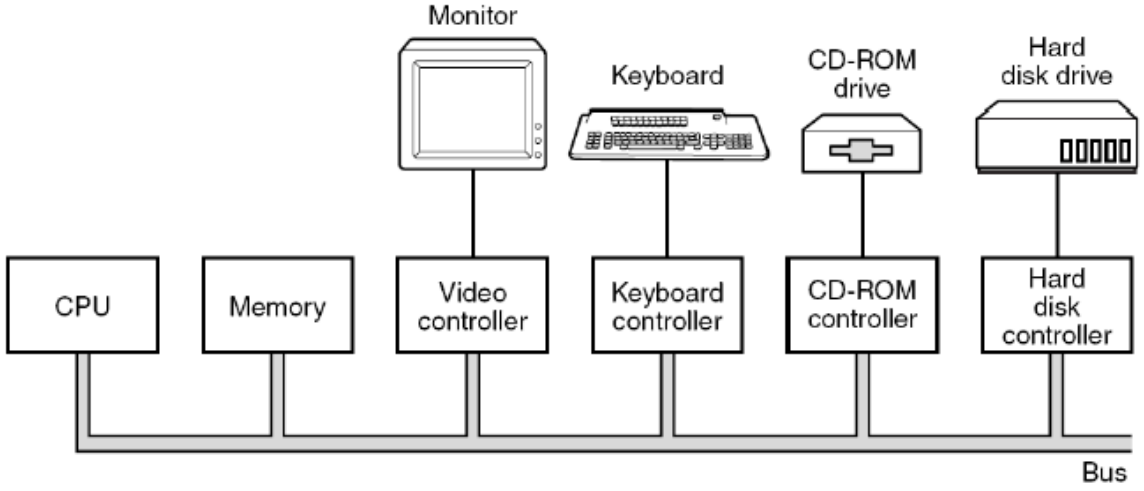
- Eksempel kretsar: Fulladder



Polling, avbrudd og DMA

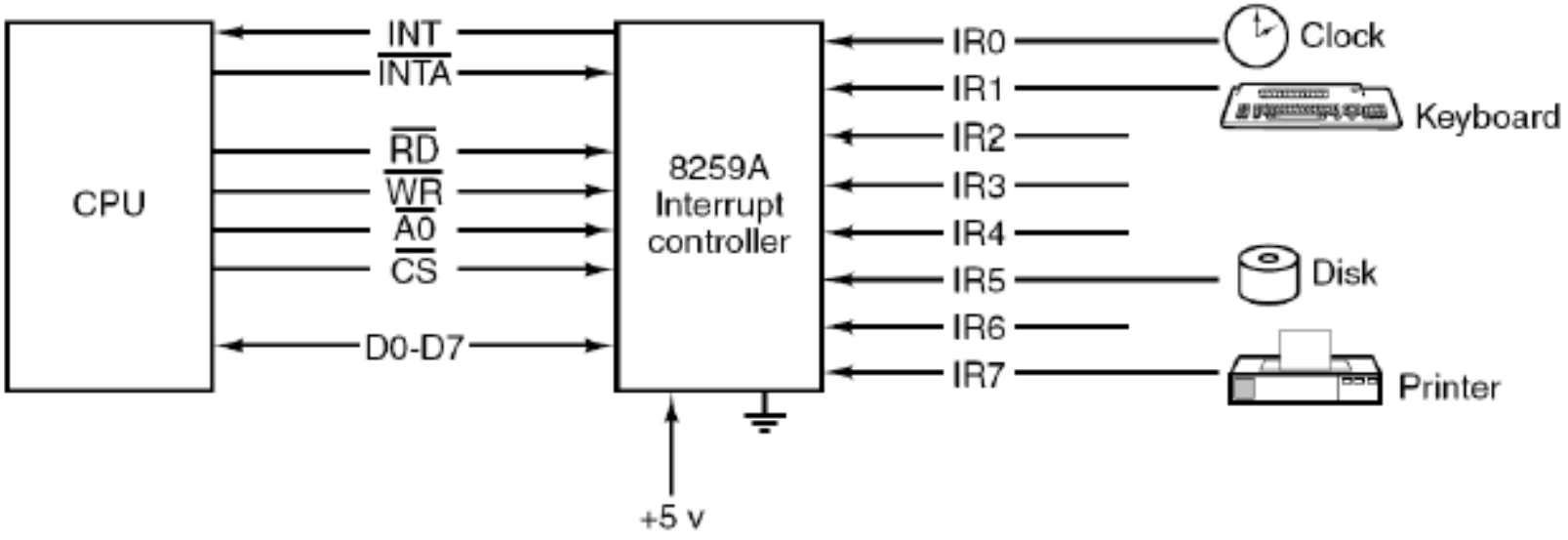
- Polling (Programstyrt I/O)
  - Prosessor sjekkar om externeinheit har data klar (*er det trykt på ein tast?*)
- Interrupt (Avbrudd)
  - Externeinheit seier frå når data er klar (*det er trykt på ein tast*)

- Direct Memory Acess
  - Kontroller overfører direkte til/fra hovedlager
  - Prosessor tar seg kun av oppstart av overføring
  - Prosessor kan dermed gjøre noko anna imens



Busser og avbrudd

- Avbruddskontroller velger hvis flere avbrudd kommer samtidig og aktiverer INT
- Når INTA blir aktivert, overfører nummer på enhet
- Prosessor bruker nummer til å finne adresse til avbruddsrutine (via ”interrupt vector”)

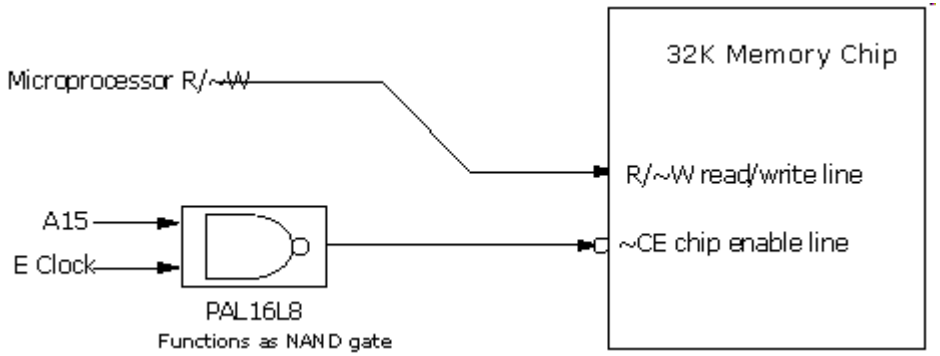


Adresse dekoding

- Minnekart
  - Kva ligg på kva minne adresse
  - Kan då aksesera einheitar ved å lese/skrive til minne adresser

Adresse dekoding

- Dekoding
  - Gi ein einheit eller eit minne område ei adresse
  - Brukar adresselinjer og dekodar for ei bestemt adresse eller adresse område

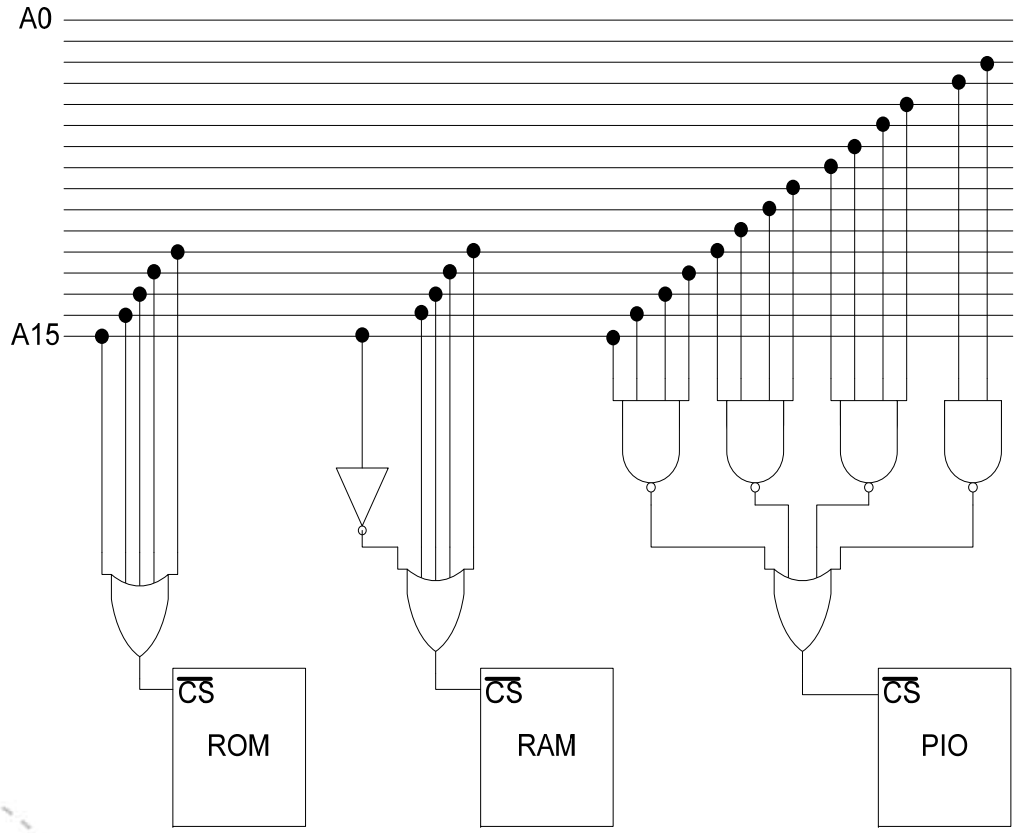


- Når:
  - A15 = ”1”
  - E Clock = ”1”
  - ~CE er då = ”0” og aktiv Minnebrikken er valgt
  - R/W bestemer om det skal lesast eller skrivast

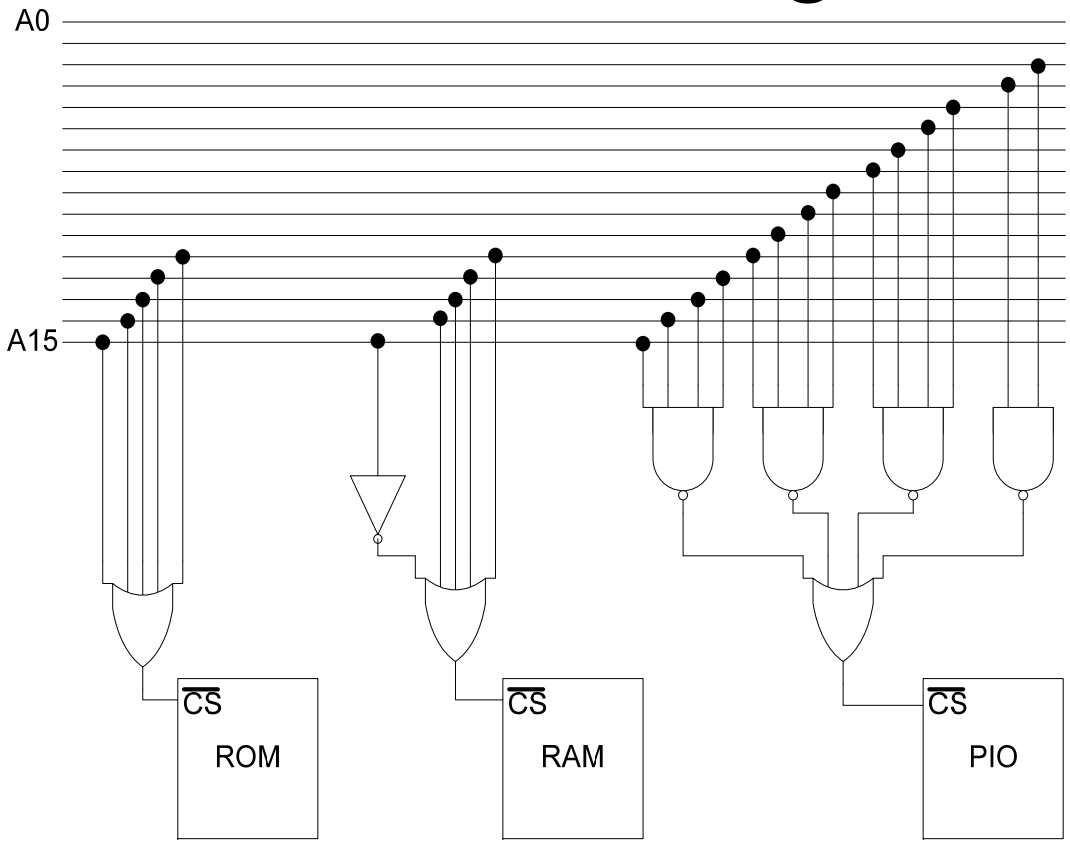
NTNU
Innovation and Creativity

Adresse dekoding

- Dekoding
  - 3 einheitar ROM, RAM og PIO skal adresse mappast
  - Kvar har eit adresseområde



Adresse dekoding finne adr. område



ROM:

Dekod: 0000 0XXX XXXX XXXX

Høg: 0000 0111 1111 1111

0 7 F F

Låg: 0000 0000 0000 0000

0 0 0 0

RAM:

Dekod: 1000 0XXX XXXX XXXX

Høg: 8 7 F F

Låg: 8 0 0 0

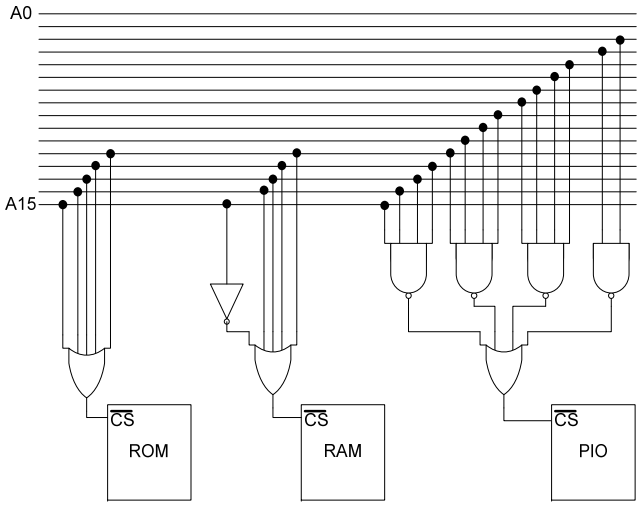
PIO:

Dekod: 1111 1111 1111 11XX

Høg: F F F F

Låg: F F F C

Adresse dekoding lage adr. kart



PIO:

Dekod: 1111 1111 1111 11XX

Høg: F F F F

Låg: F F F C

RAM:

Dekod: 1000 0XXX XXXX XXXX

Høg: 8 7 F F

Låg: 8 0 0 0

ROM:

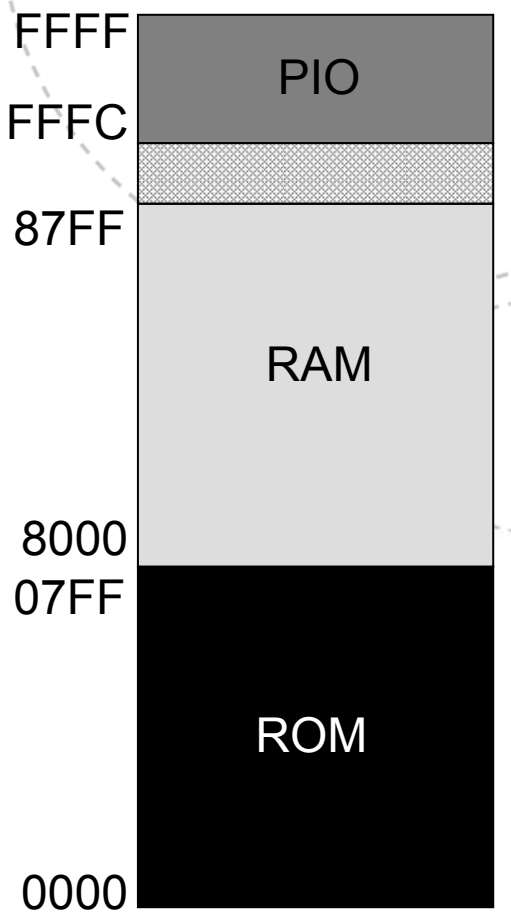
Dekod: 0000 0XXX XXXX XXXX

Høg: 0000 0111 1111 1111

0 7 F F

Låg: 0000 0000 0000 0000

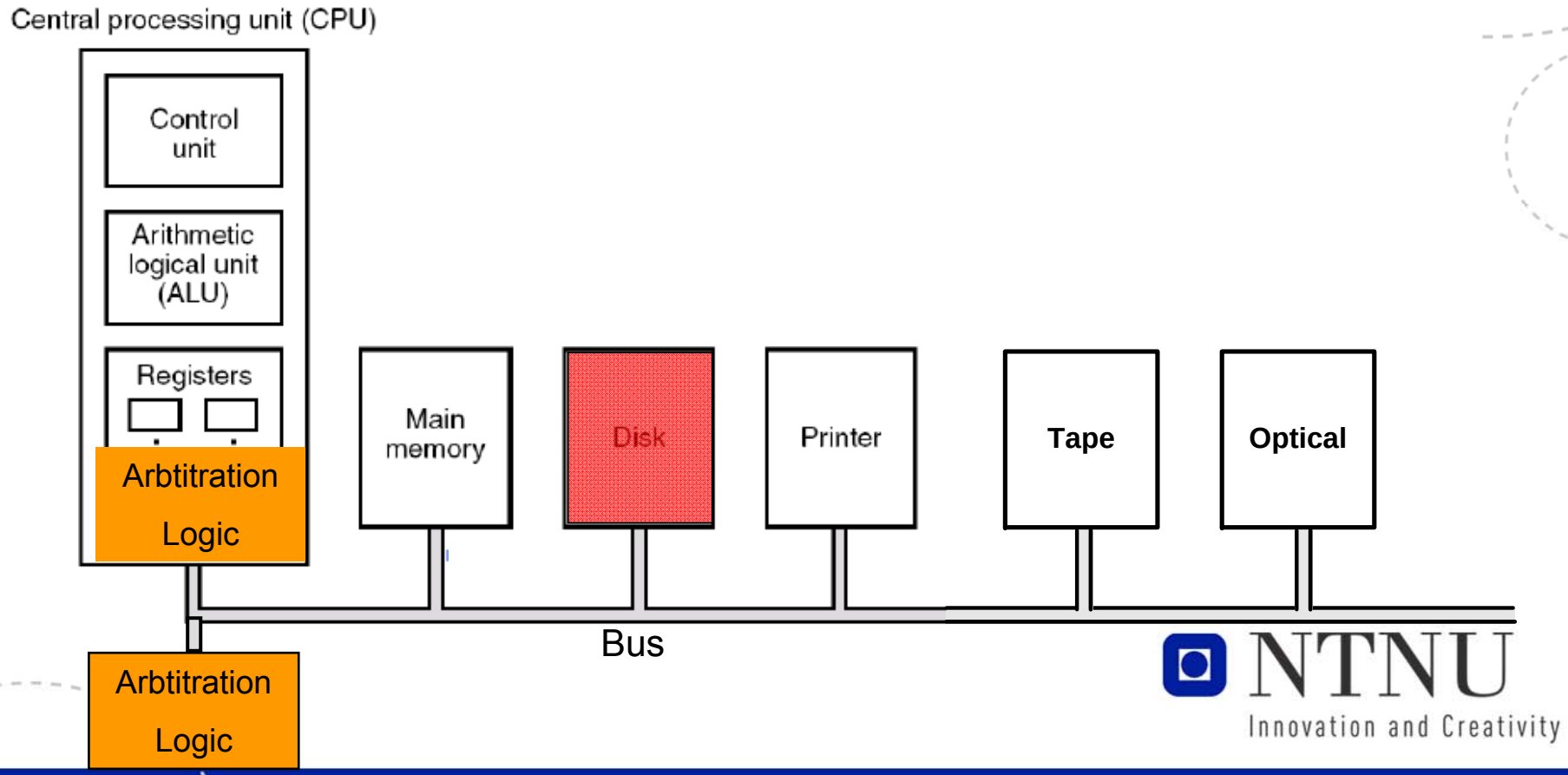
0 0 0 0



NTNU
Innovation and Creativity

Arbitrering: Kven kontrollerar bussen

- Må ha logikk for buss arbitrering
  - Internt i CPU
  - Ekstern arbitreringseinheit

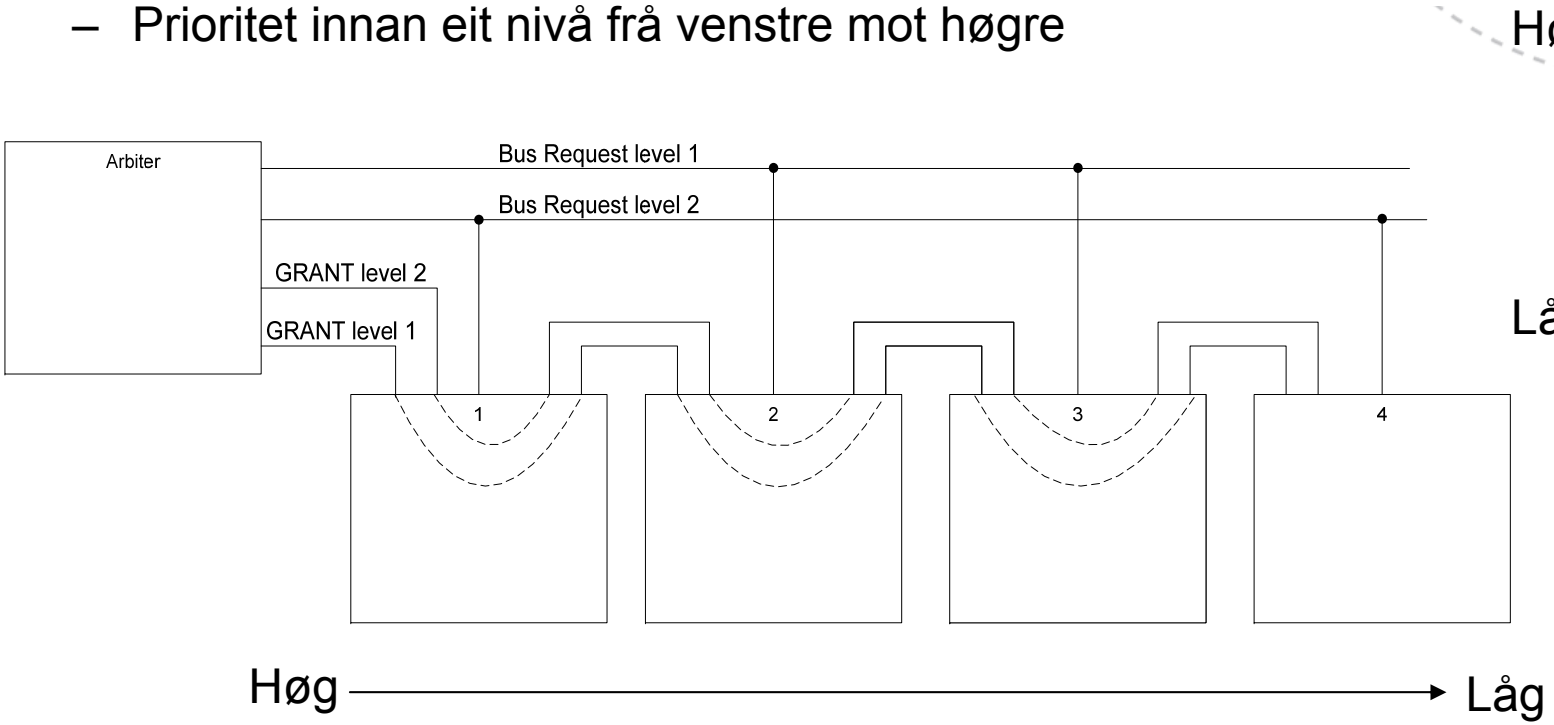


Arbitrering: To typar

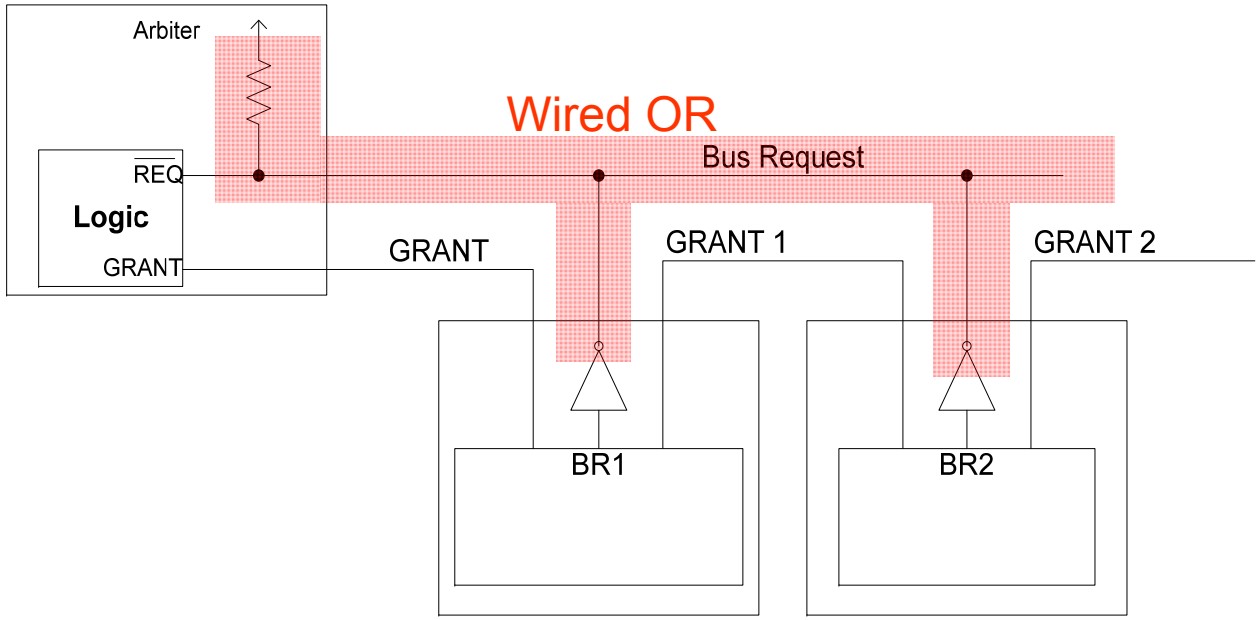
- Sentralisert arbitrering
  - Eigen sentral arbitreringseining
  - Bestemt sentralt kven som får bussen ved samtidig ”request”
- Desentralisert arbitrering
  - Einingane ”forhandlar” om bussen
  - Ofte fleire busslinjer
  - Ingen sentral arbitreringslogikk

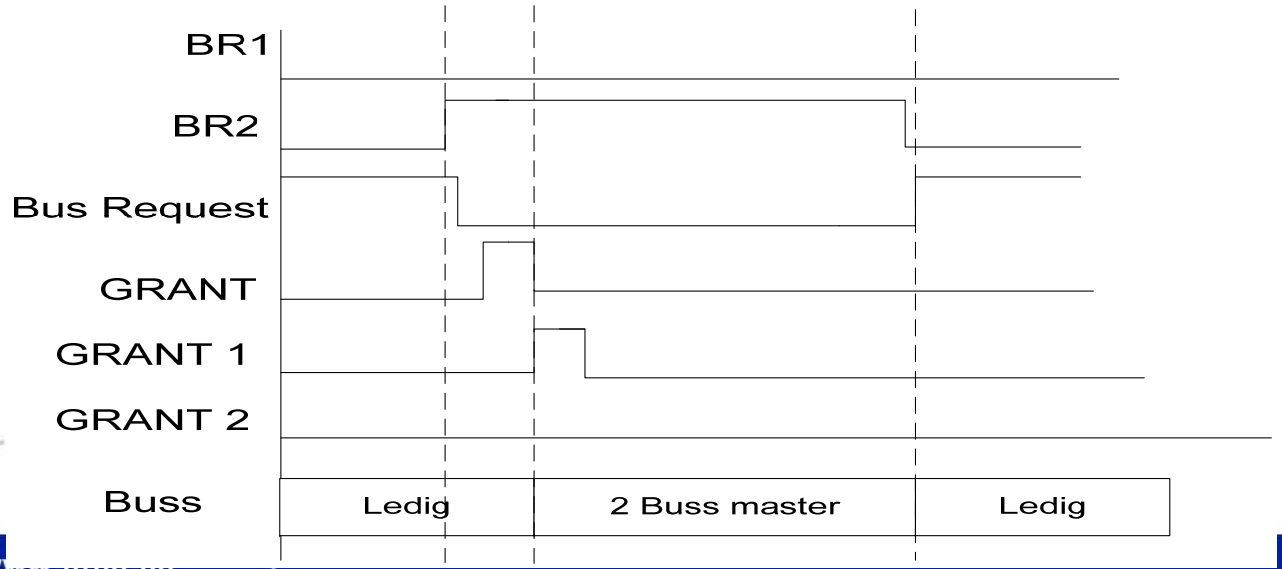
Sentralisert arbitrering: Detaljar

- Fleire nivå
  - To prioritetsnivå level 1 og level 2
  - Level 1 høgare enn level 2
  - Prioritet innan eit nivå frå venstre mot høgre



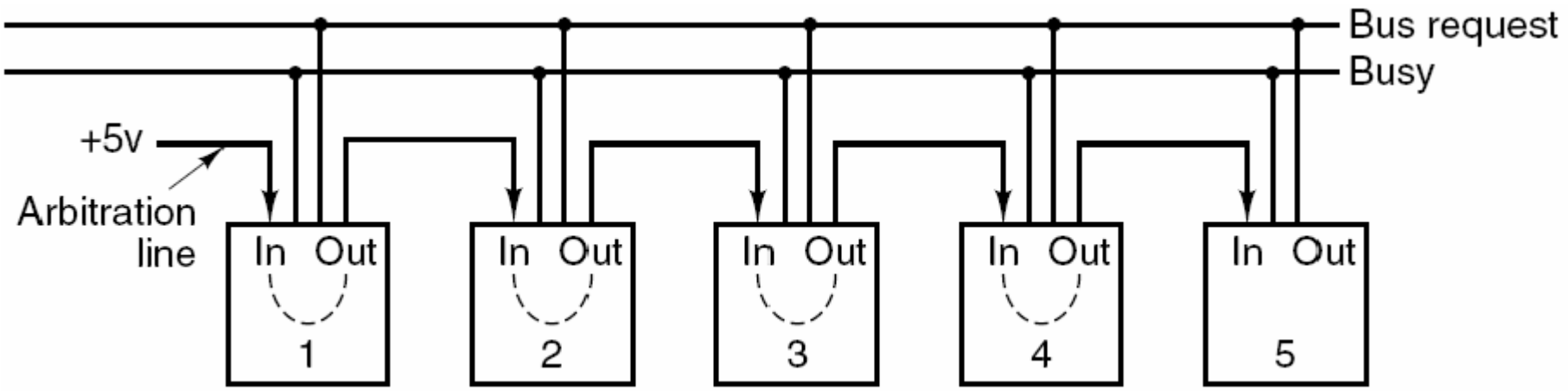
Sentralisert arbitrering: Detaljar





NTNU
Innovation and Creativity

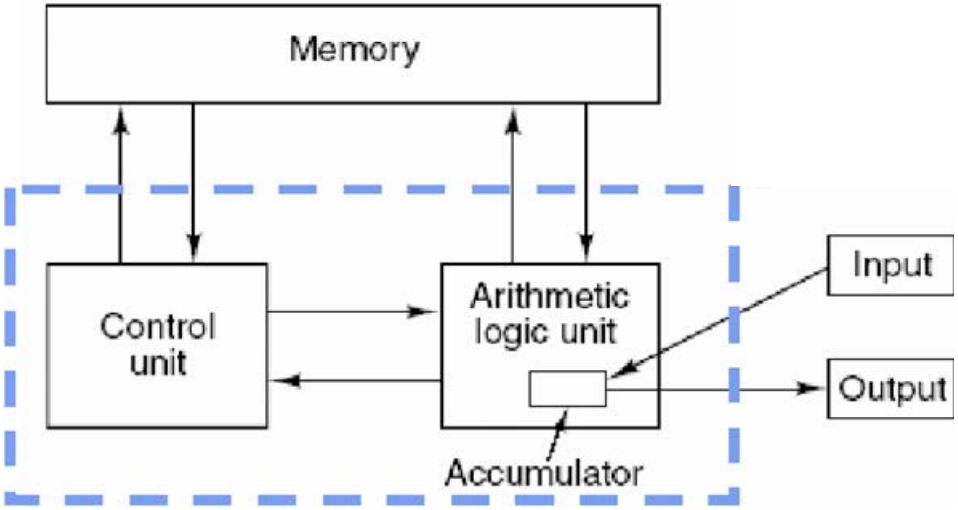
Desentralisert arbitrering

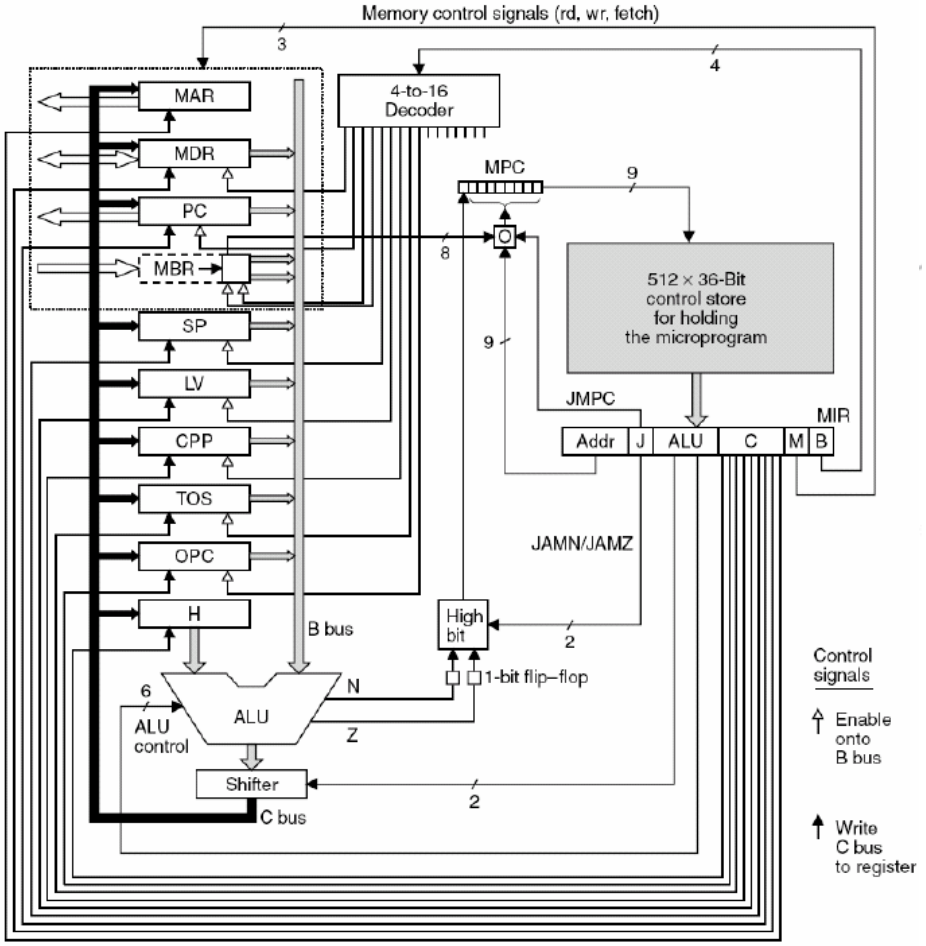


- Ingen sentral arbiter
  - Tilgang bestemt av ”Busy” og ”Arbitration line”
  - Ikkje naudsynt med sentral logikk for arbitrering
- Signal
  - Bus request: wiered_OR
  - Busy: Styrt av den som er buss ”master”
  - Arbitration line: Daisy chained

IJVM: Generell datamaskin

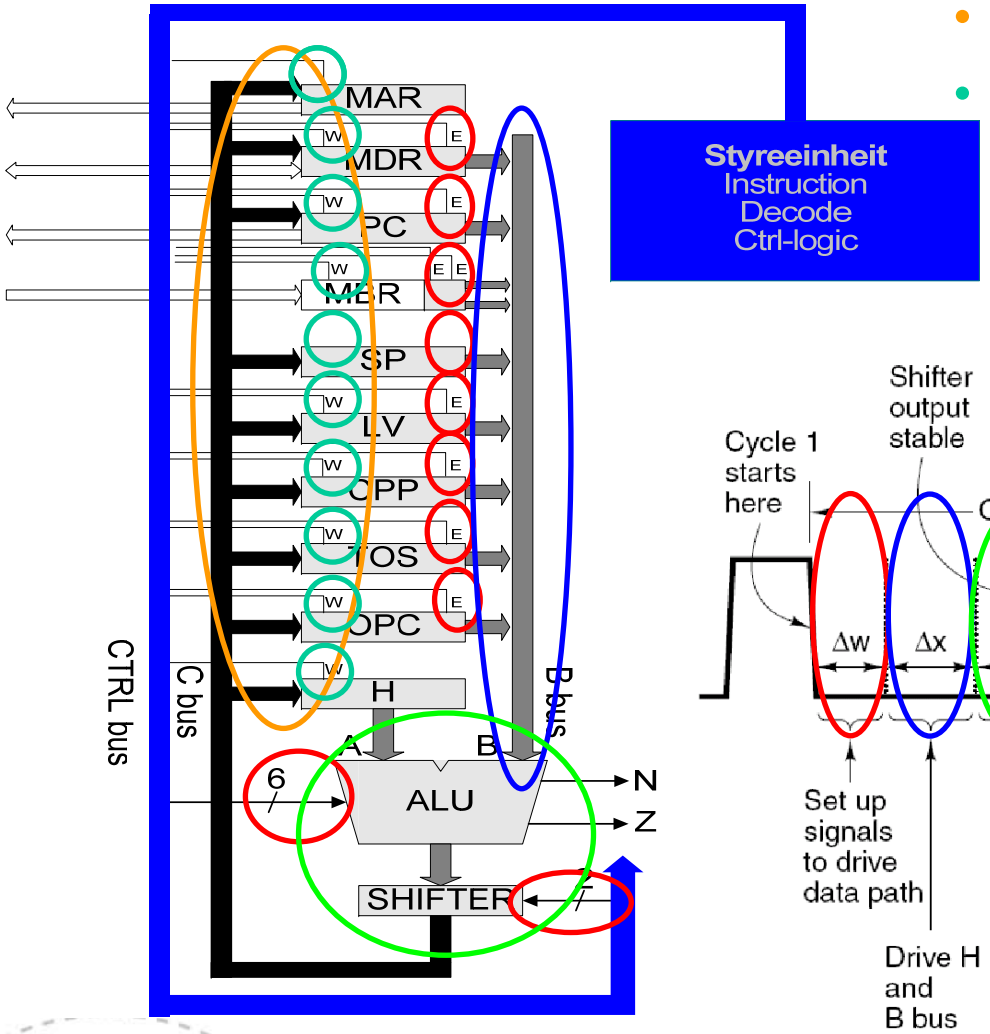
Von Neumann-arkitektur

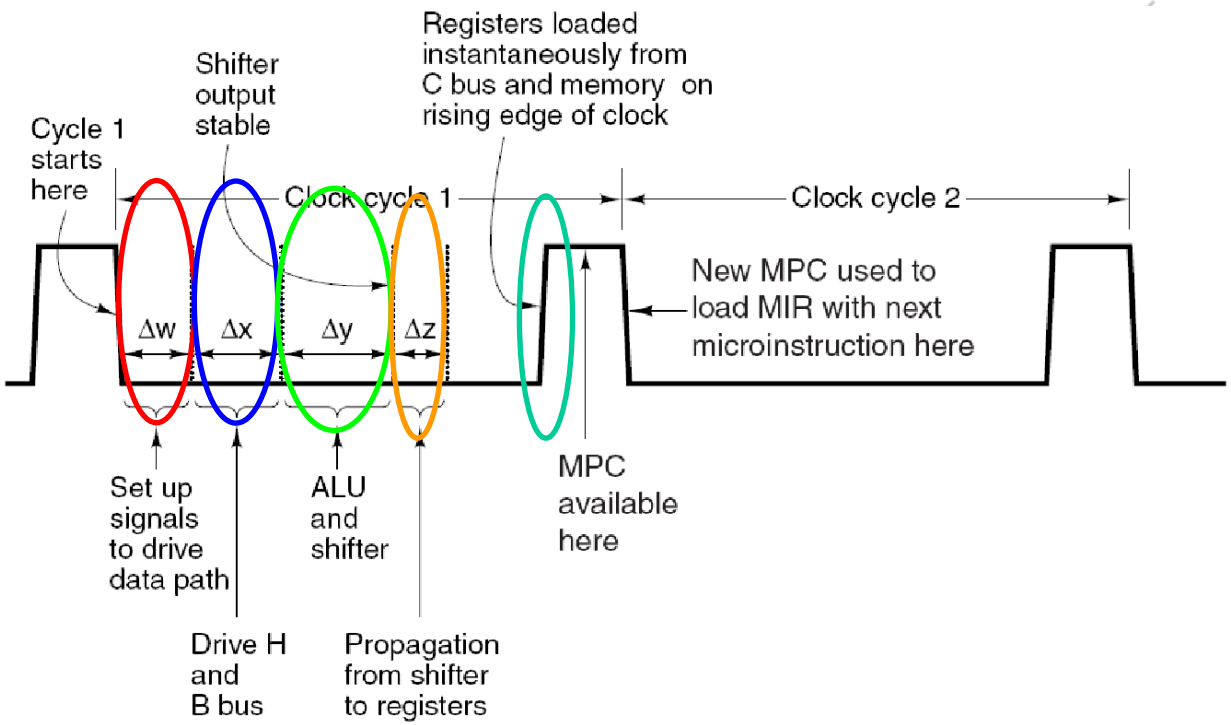




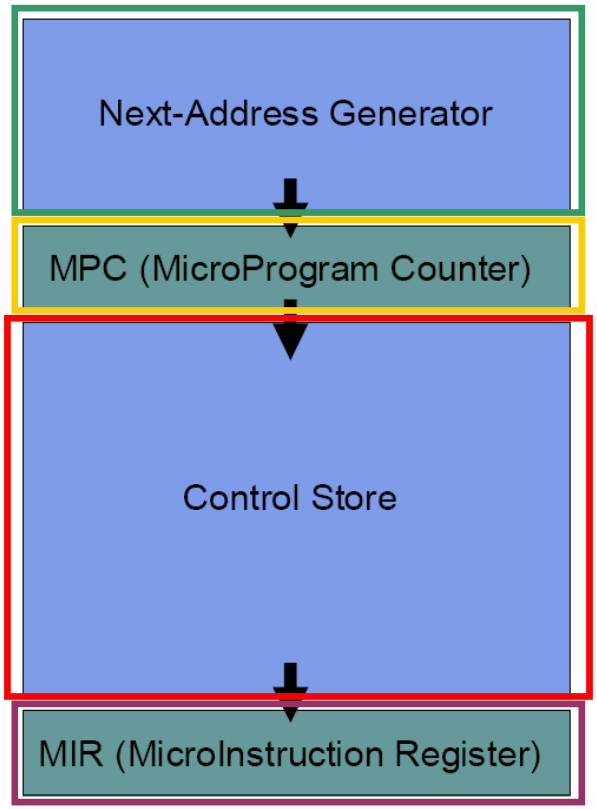
Utføre instruksjon

- Set opp kontrollsignal (Δw)
- Registerverdi til B bus (Δx)
- ALU og Shifter forsinkelse (Δy)
- Forsinkelse på C bus til register (Δz)
- Last register

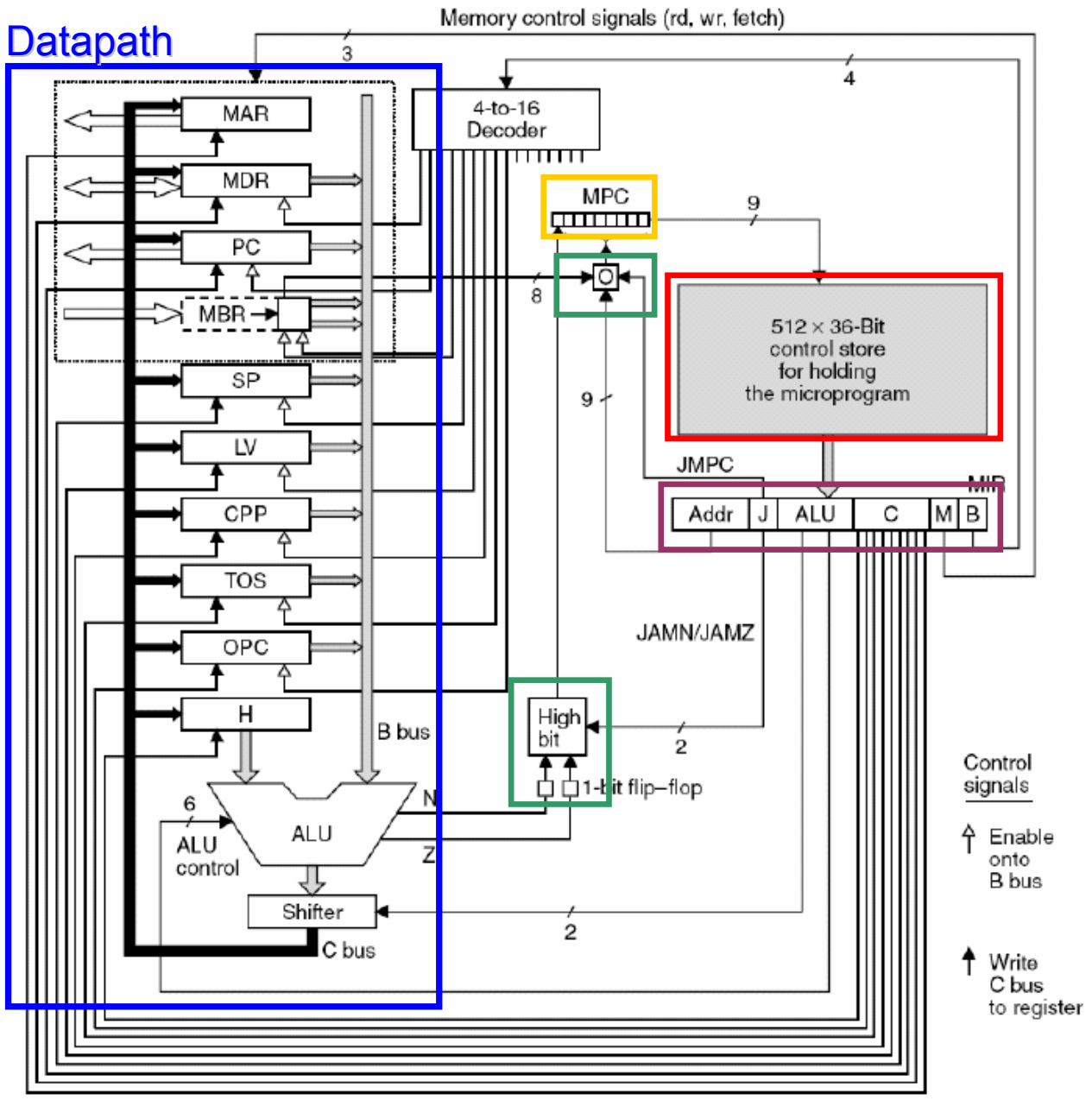




IJVM



Datapath

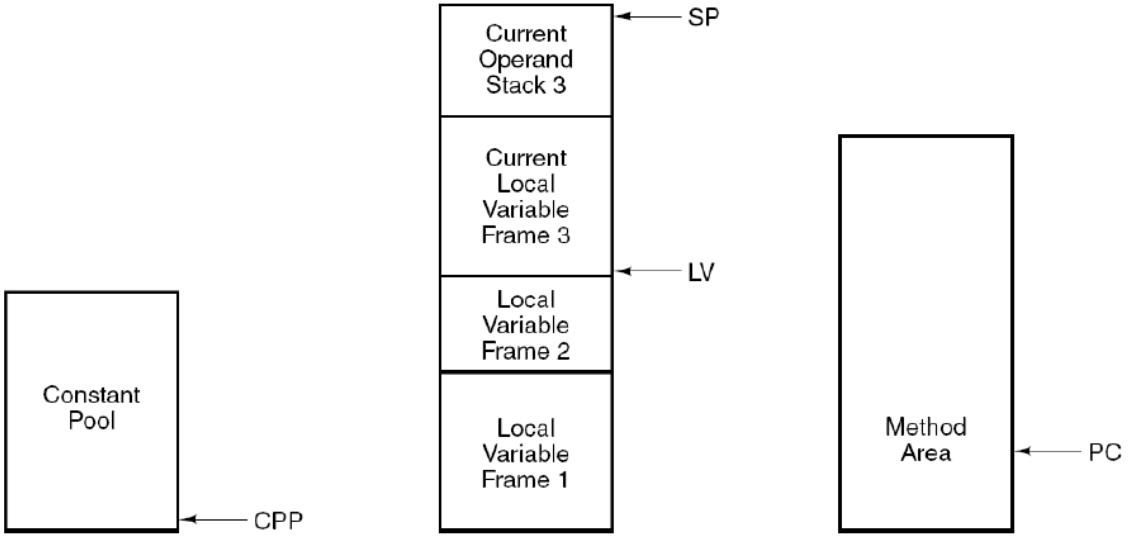


IJVM: MicrArch vs Instruction Set Architecture

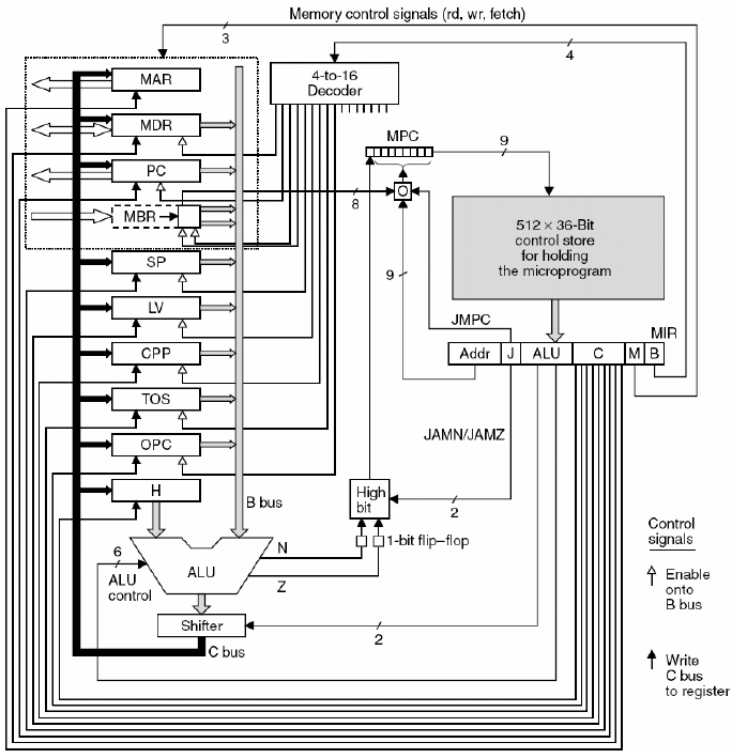
- Instruksjonsset

Hex	Mnemonic	Meaning
0x10	BIPUSH *byte*	Push byte onto stack
0x15	ILOAD *varnum*	Push local variable onto stack
0x13	LDC_W *index*	Push constant from constant pool onto stack
0x57	POP	Delete word on top of stack
0x36	ISTORE *varnum*	Pop word from stack and store in local var.
0x59	DUP	Copy top word on stack and push onto stack
0x5F	SWAP	Swap the two top words on the stack

- Minnemodell

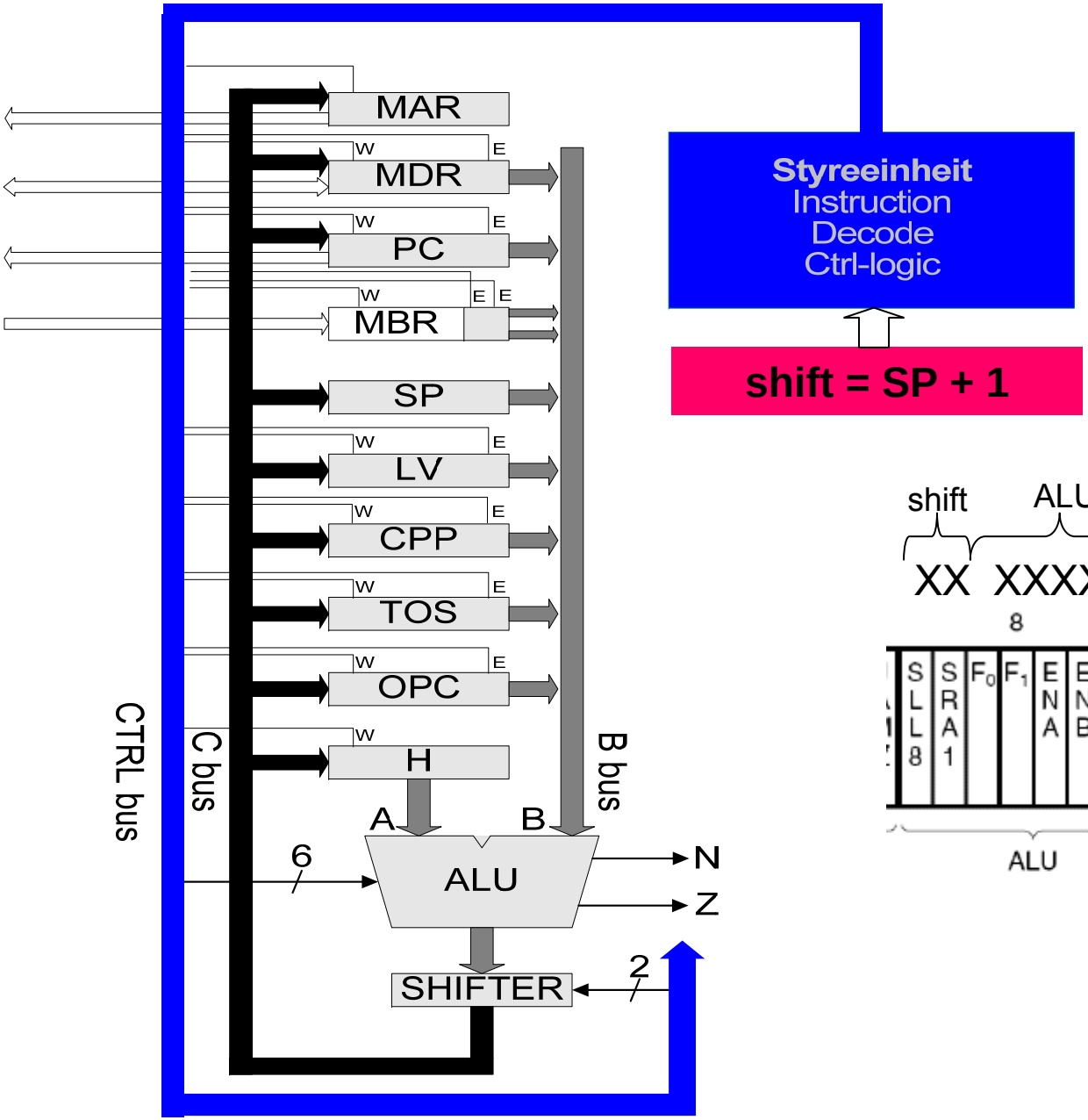


- Micro Arkitektur

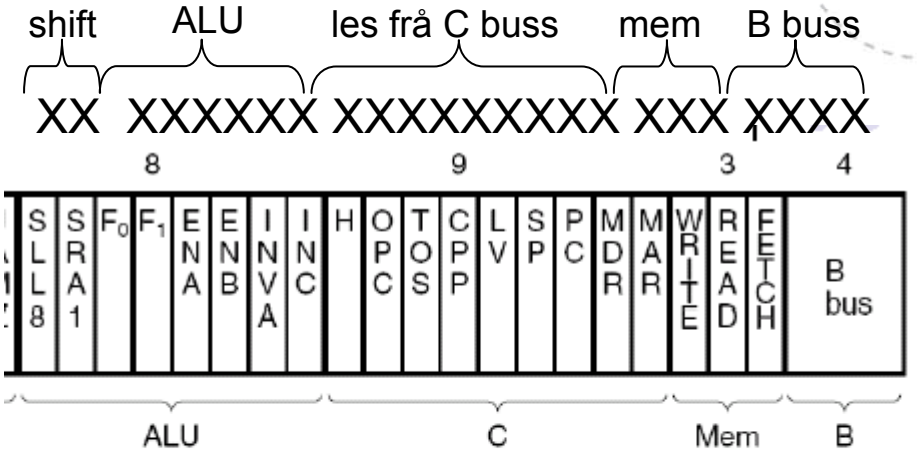


NTNU
Innovation and Creativity

Lita oppgåve 1:



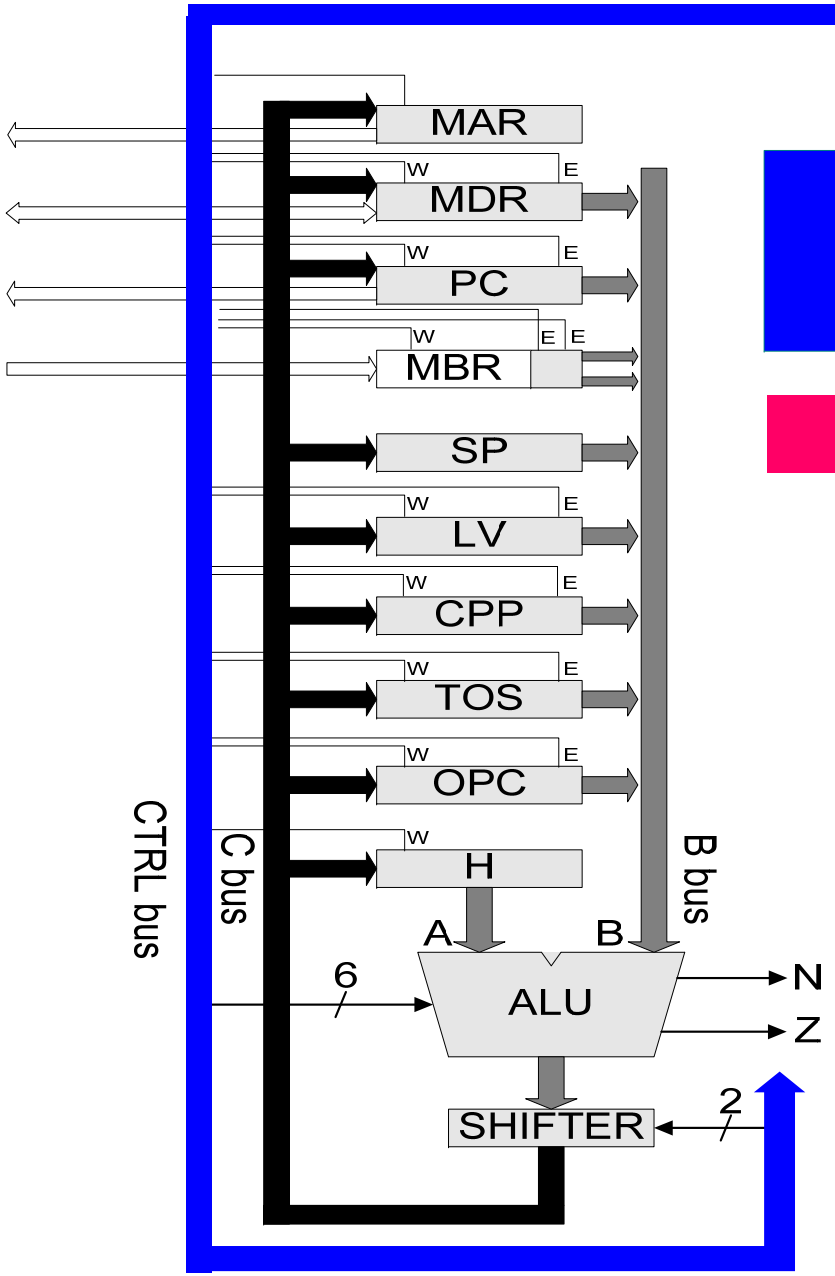
F_0	F_1	ENA	ENB	INVA	INC	Function
0	1	1	0	0	0	A
0	1	0	1	0	0	B
0	1	1	0	1	0	$\overline{A}$
1	0	1	1	0	0	$\overline{B}$
1	1	1	1	0	0	A + B
1	1	1	1	0	1	A + B + 1
1	1	1	0	0	1	A + 1
1	1	0	1	0	1	B + 1
1	1	1	1	1	1	B – A
1	1	0	1	1	0	B – 1
1	1	1	0	1	1	–A
0	0	1	1	0	0	A AND B
0	1	1	1	0	0	A OR B
0	1	0	0	0	0	0
1	1	0	0	0	1	1
1	1	0	0	1	0	–1



B bus registers

0 = MDR	5 = LV
1 = PC	6 = CPP
2 = MBR	7 = TOS
3 = MBRU	8 = OPC
4 = SP	9-15 none

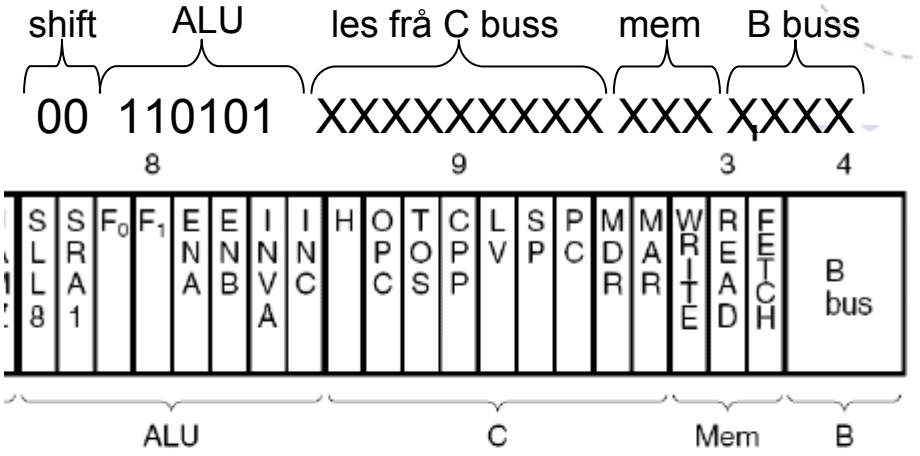
Svar lita oppgåve



Styreeinheit
Instruction
Decode
Ctrl-logic

shift = SP + 1

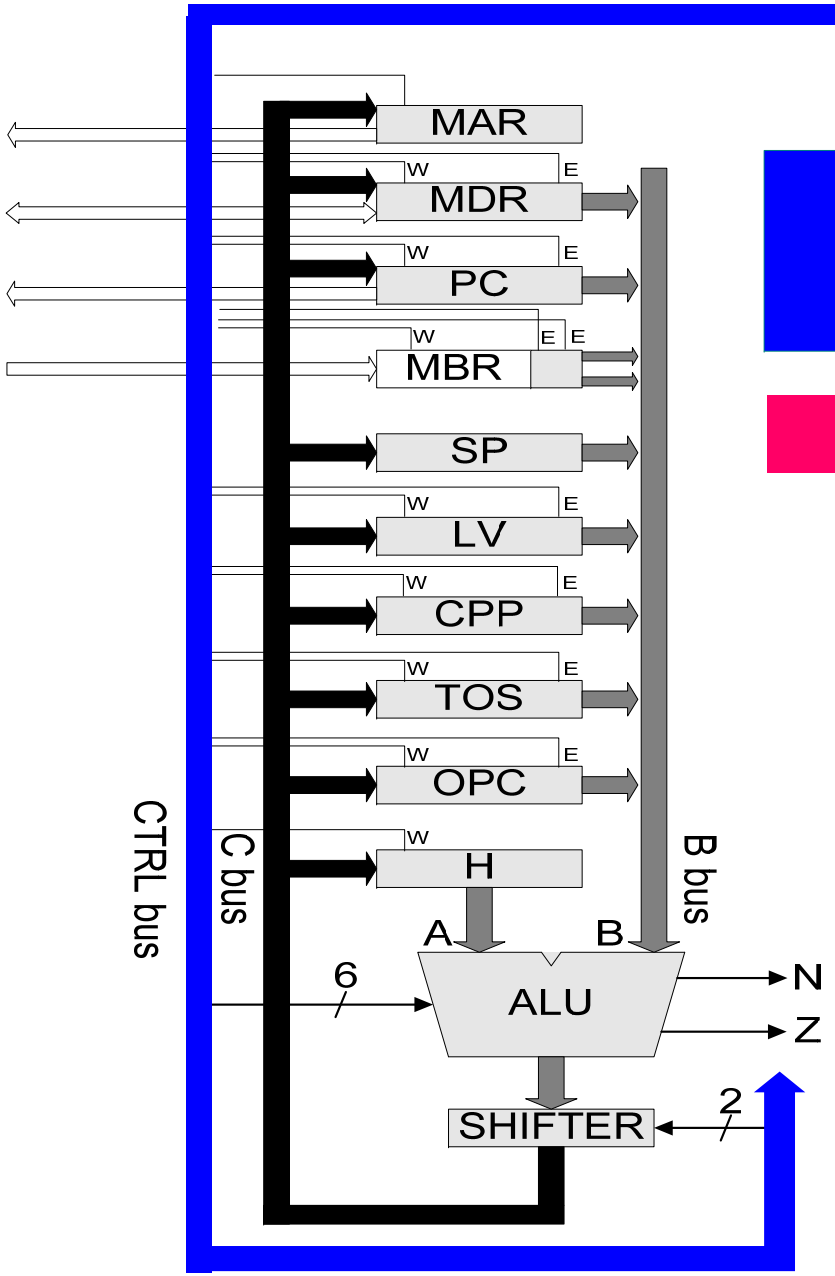
F_0	F_1	ENA	ENB	INVA	INC	Function
0	1	1	0	0	0	A
0	1	0	1	0	0	B
0	1	1	0	1	0	$\overline{A}$
1	0	1	1	0	0	$\overline{B}$
1	1	1	1	0	0	A + B
1	1	1	1	0	1	A + B + 1
1	1	1	0	0	1	A + 1
1	1	0	1	0	1	B + 1
1	1	1	1	1	1	B – A
1	1	0	1	1	0	B – 1
1	1	1	0	1	1	–A
0	0	1	1	0	0	A AND B
0	1	1	1	0	0	A OR B
0	1	0	0	0	0	0
1	1	0	0	0	1	1
1	1	0	0	1	0	–1



B bus registers

0 = MDR	5 = LV
1 = PC	6 = CPP
2 = MBR	7 = TOS
3 = MBRU	8 = OPC
4 = SP	9-15 none

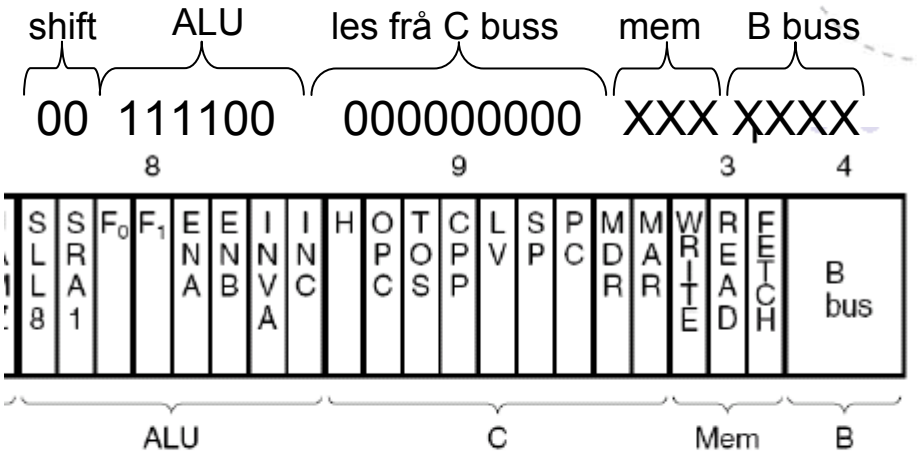
Svar lita oppgåve



Styreeinheit
Instruction
Decode
Ctrl-logic

shift = SP + 1

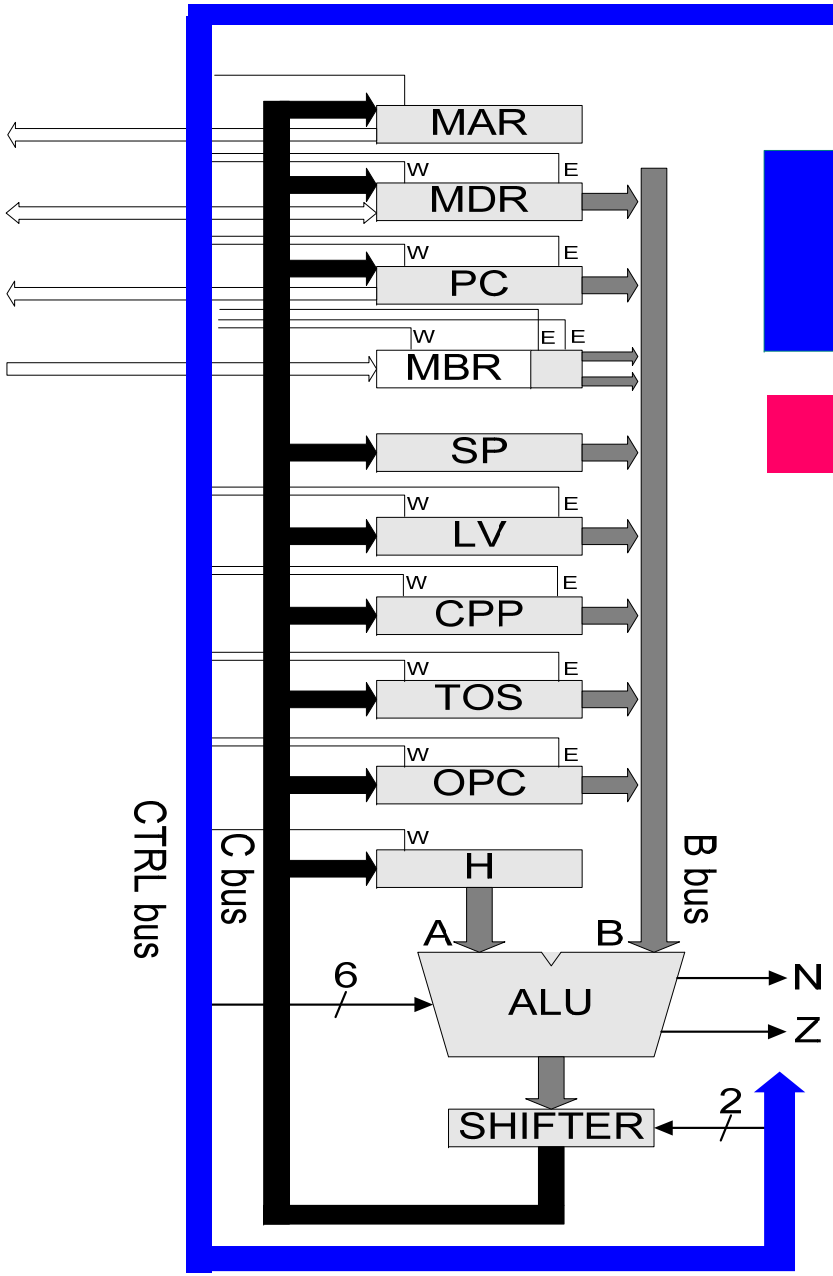
F_0	F_1	ENA	ENB	INVA	INC	Function
0	1	1	0	0	0	A
0	1	0	1	0	0	B
0	1	1	0	1	0	$\overline{A}$
1	0	1	1	0	0	$\overline{B}$
1	1	1	1	0	0	A + B
1	1	1	1	0	1	A + B + 1
1	1	1	0	0	1	A + 1
1	1	0	1	0	1	B + 1
1	1	1	1	1	1	B – A
1	1	0	1	1	0	B – 1
1	1	1	0	1	1	–A
0	0	1	1	0	0	A AND B
0	1	1	1	0	0	A OR B
0	1	0	0	0	0	0
1	1	0	0	0	1	1
1	1	0	0	1	0	–1



B bus registers

0 = MDR	5 = LV
1 = PC	6 = CPP
2 = MBR	7 = TOS
3 = MBRU	8 = OPC
4 = SP	9-15 none

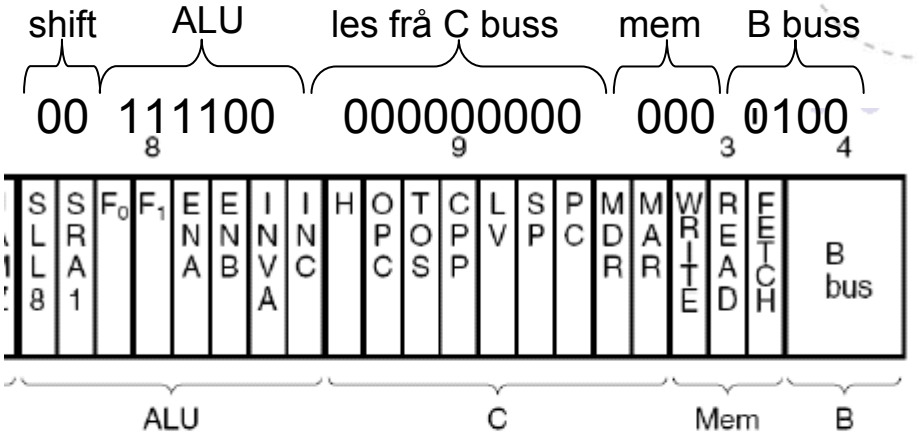
Svar lita oppgåve



Styreeinheit
Instruction
Decode
Ctrl-logic

shift = SP + 1

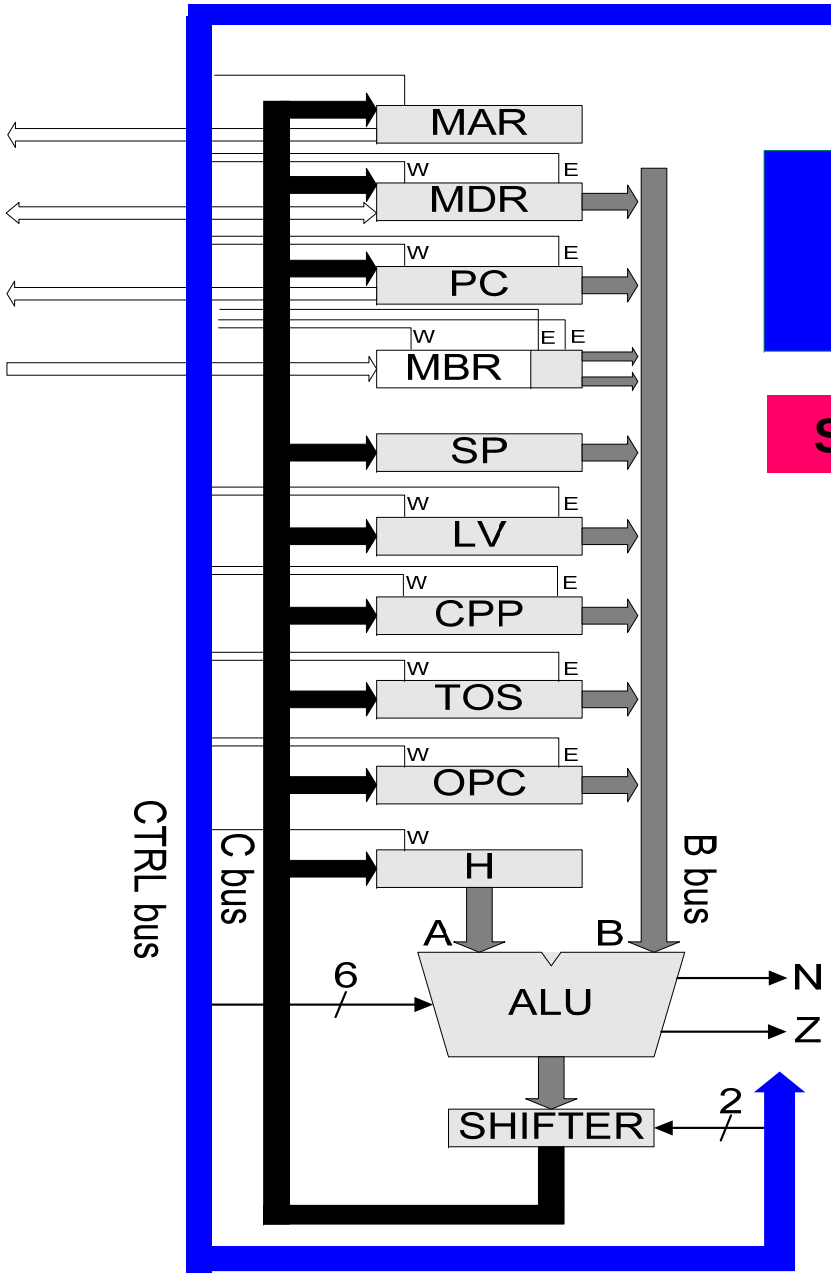
F_0	F_1	ENA	ENB	INVA	INC	Function
0	1	1	0	0	0	A
0	1	0	1	0	0	B
0	1	1	0	1	0	$\overline{A}$
1	0	1	1	0	0	$\overline{B}$
1	1	1	1	0	0	A + B
1	1	1	1	0	1	A + B + 1
1	1	1	0	0	1	A + 1
1	1	0	1	0	1	B + 1
1	1	1	1	1	1	B – A
1	1	0	1	1	0	B – 1
1	1	1	0	1	1	–A
0	0	1	1	0	0	A AND B
0	1	1	1	0	0	A OR B
0	1	0	0	0	0	0
1	1	0	0	0	1	1
1	1	0	0	1	0	–1



B bus registers

0 = MDR	5 = LV
1 = PC	6 = CPP
2 = MBR	7 = TOS
3 = MBRU	8 = OPC
4 = SP	9-15 none

Eksempel: 2 brukar B og C buss



Styreeinheit
Instruction
Decode
Ctrl-logic

SP = TOS + OPC

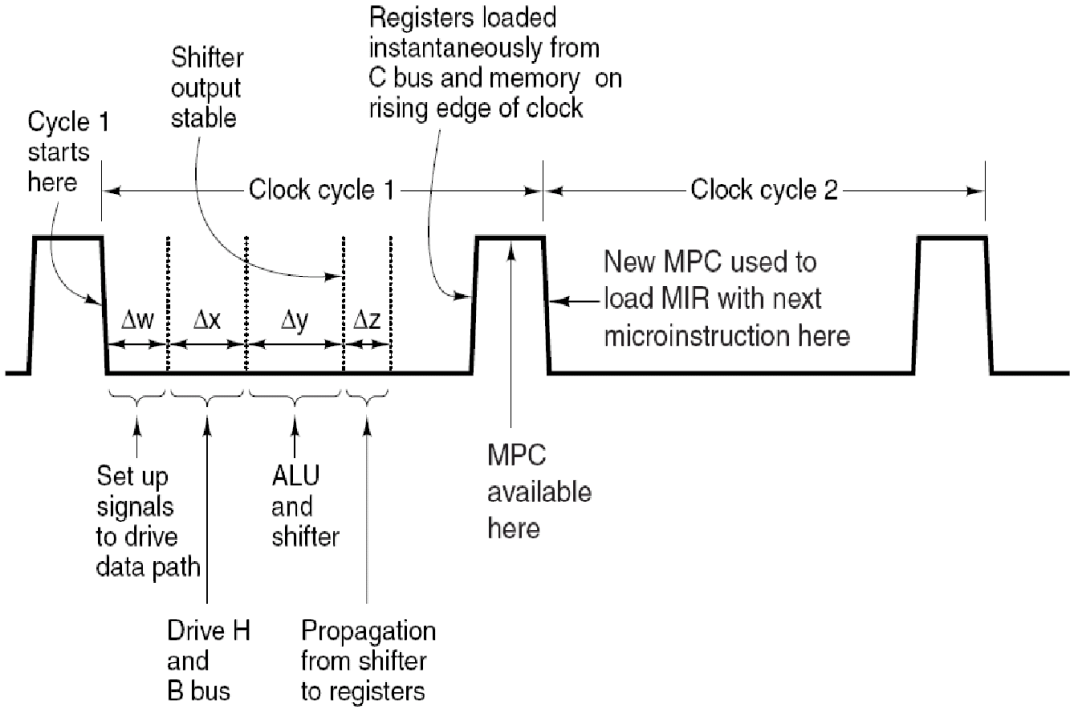
TOS eller OPC må fyrst i H-register: (velger TOS)

Instruksjon må då:

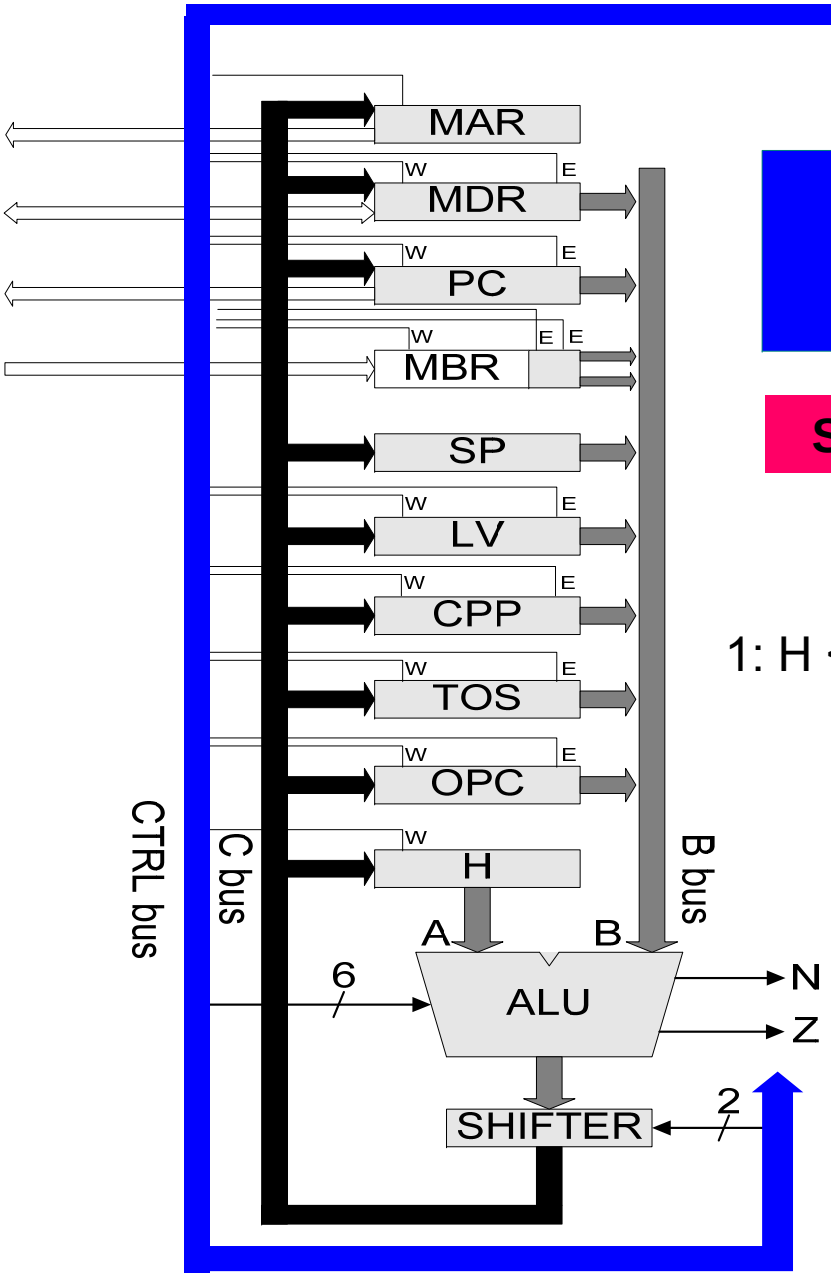
1: H <- TOS

2: H + OPC, SP <- SHIFT

To microinstruksjonar



Eksempel: 2

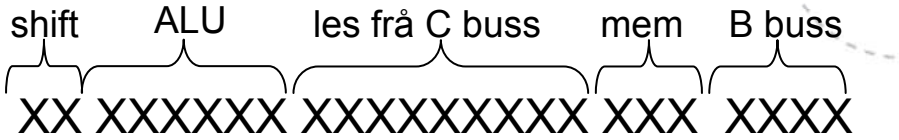


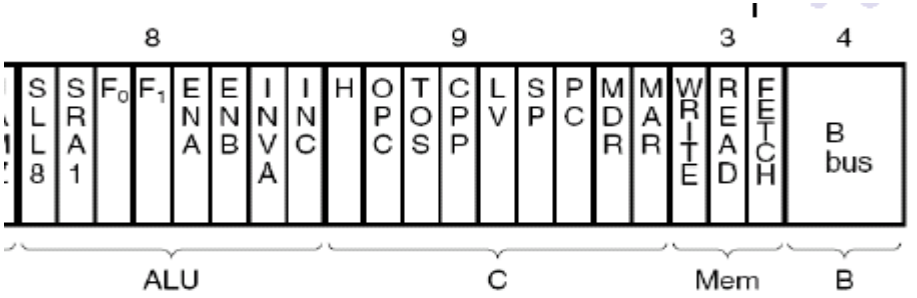
Styreeinheit
Instruction
Decode
Ctrl-logic

SP = TOS + OPC

F_0	F_1	ENA	ENB	INVA	INC	Function
0	1	1	0	0	0	A
0	1	0	1	0	0	B
0	1	1	0	1	0	$\bar{A}$
1	0	1	1	0	0	$\bar{B}$
1	1	1	1	0	0	A + B
1	1	1	1	0	1	A + B + 1
1	1	1	0	0	1	A + 1
1	1	0	1	0	1	B + 1
1	1	1	1	1	1	B – A
1	1	0	1	1	0	B – 1
1	1	1	0	1	1	–A
0	0	1	1	0	0	A AND B
0	1	1	1	0	0	A OR B
0	1	0	0	0	0	0
1	1	0	0	0	1	1
1	1	0	0	1	0	–1

1: H <- TOS:

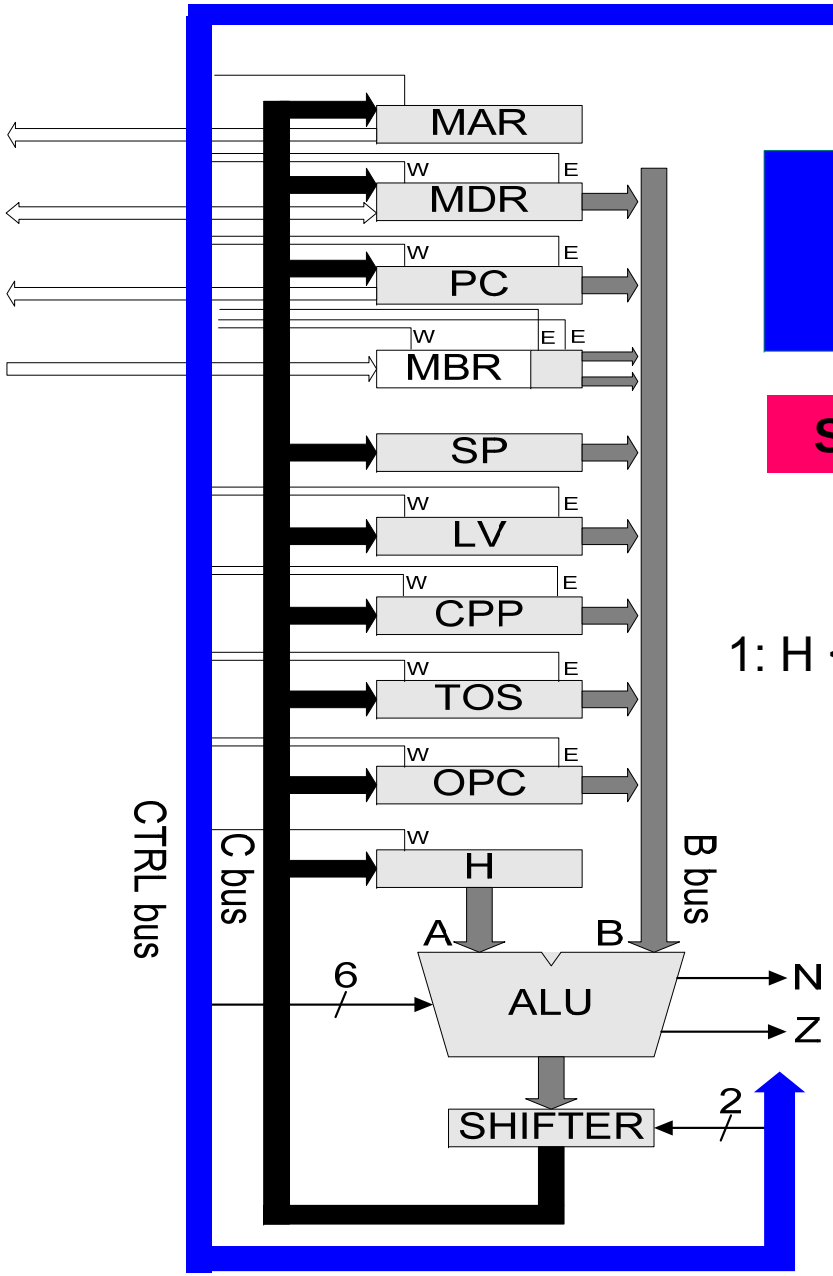




B bus registers

0 = MDR	5 = LV
1 = PC	6 = CPP
2 = MBR	7 = TOS
3 = MBRU	8 = OPC
4 = SP	9-15 none

Eksempel: 2



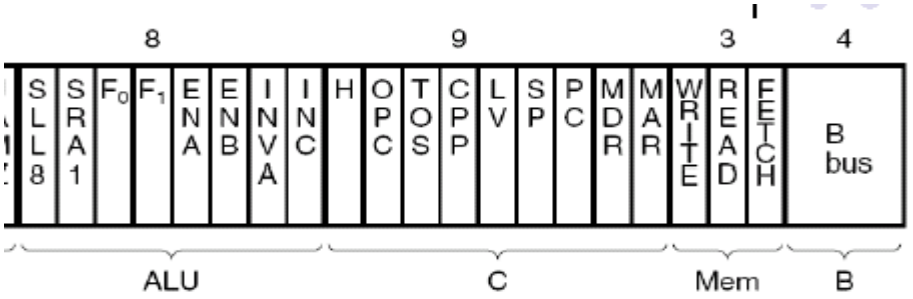
Styreeinheit
Instruction
Decode
Ctrl-logic

SP = TOS + OPC

F_0	F_1	ENA	ENB	INVA	INC	Function
0	1	1	0	0	0	A
0	1	0	1	0	0	B
0	1	1	0	1	0	$\bar{A}$
1	0	1	1	0	0	$\bar{B}$
1	1	1	1	0	0	A + B
1	1	1	1	0	1	A + B + 1
1	1	1	0	0	1	A + 1
1	1	0	1	0	1	B + 1
1	1	1	1	1	1	B – A
1	1	0	1	1	0	B – 1
1	1	1	0	1	1	–A
0	0	1	1	0	0	A AND B
0	1	1	1	0	0	A OR B
0	1	0	0	0	0	0
1	1	0	0	0	1	1
1	1	0	0	1	0	–1

1: H <- TOS:

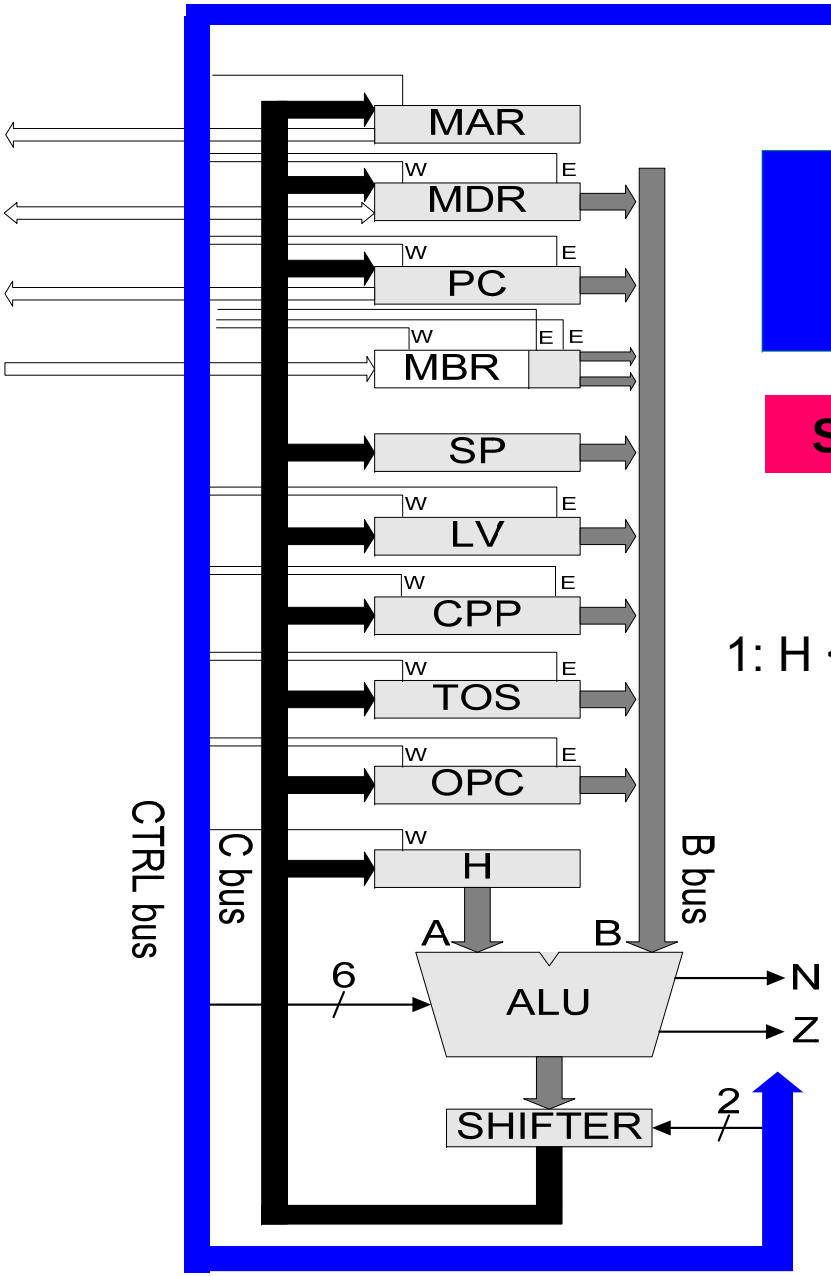
shift	ALU	les frå C buss	mem	B buss
00	010100	XXXXXXXXX	XXX	XXXX



B bus registers

0 = MDR	5 = LV
1 = PC	6 = CPP
2 = MBR	7 = TOS
3 = MBRU	8 = OPC
4 = SP	9-15 none

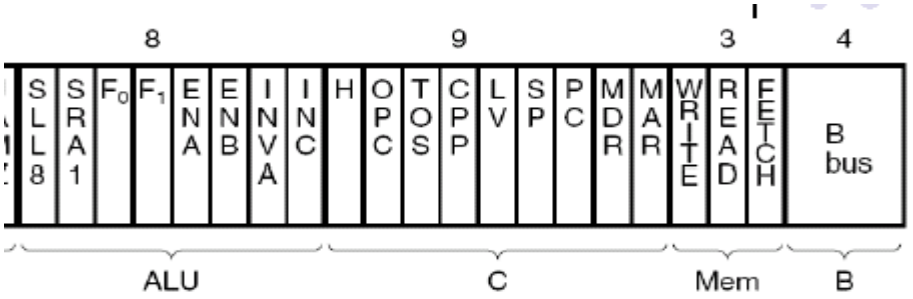
Eksempel: 2



F_0	F_1	ENA	ENB	INVA	INC	Function
0	1	1	0	0	0	A
0	1	0	1	0	0	B
0	1	1	0	1	0	$\bar{A}$
1	0	1	1	0	0	$\bar{B}$
1	1	1	1	0	0	A + B
1	1	1	1	0	1	A + B + 1
1	1	1	0	0	1	A + 1
1	1	0	1	0	1	B + 1
1	1	1	1	1	1	B – A
1	1	0	1	1	0	B – 1
1	1	1	0	1	1	–A
0	0	1	1	0	0	A AND B
0	1	1	1	0	0	A OR B
0	1	0	0	0	0	0
1	1	0	0	0	1	1
1	1	0	0	1	0	–1

1: H <- TOS:

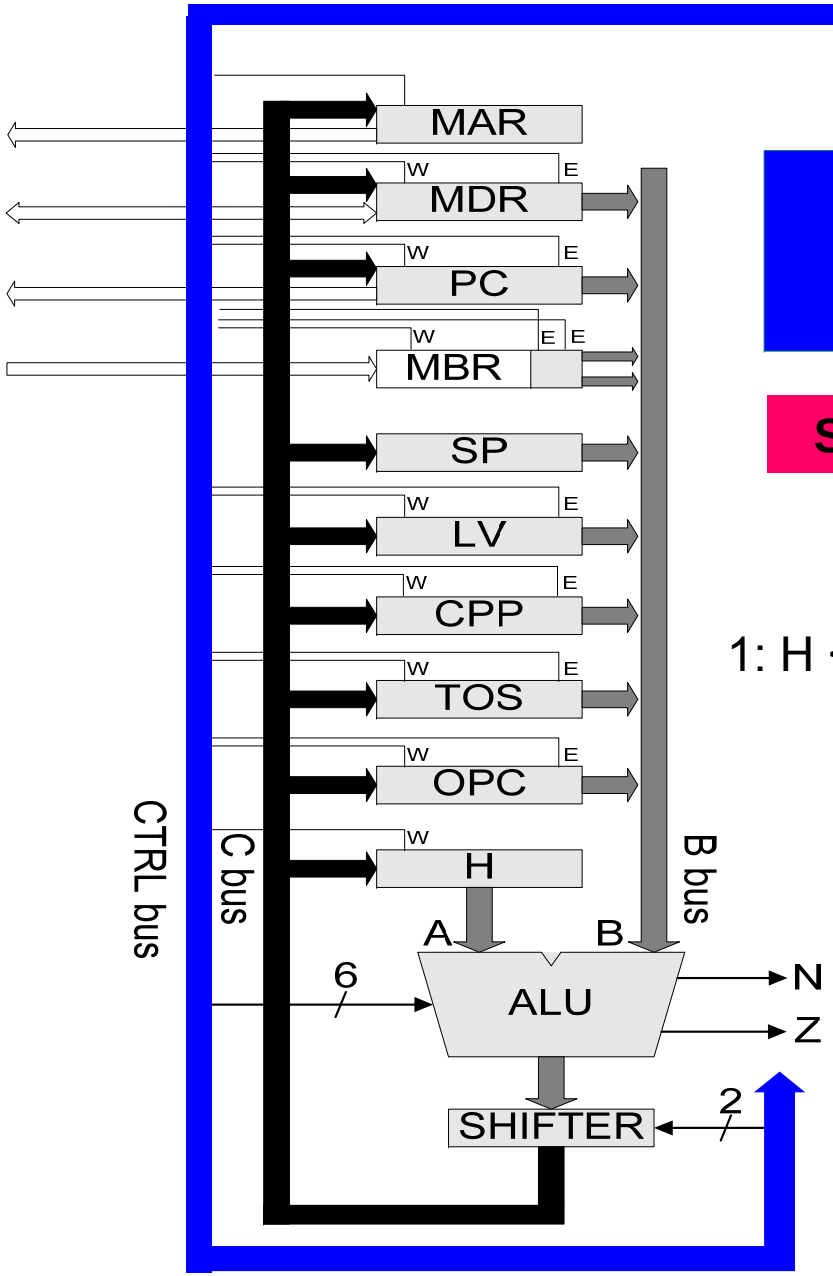
shift	ALU	les frå C buss	mem	B buss
00	010100	1000000000	XXX	XXXX



B bus registers

0 = MDR	5 = LV
1 = PC	6 = CPP
2 = MBR	7 = TOS
3 = MBRU	8 = OPC
4 = SP	9-15 none

Eksempel: 2



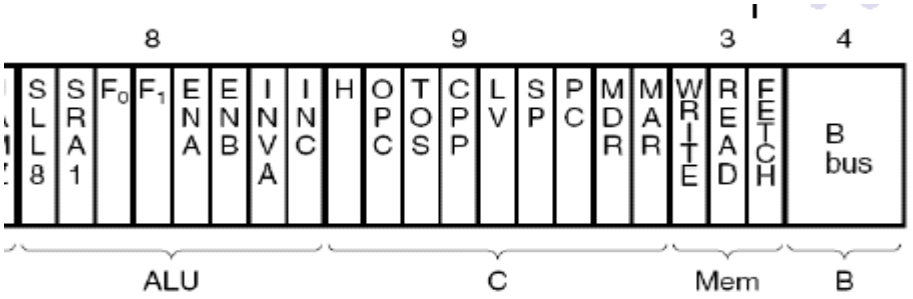
Styreeinheit
Instruction
Decode
Ctrl-logic

SP = TOS + OPC

F_0	F_1	ENA	ENB	INVA	INC	Function
0	1	1	0	0	0	A
0	1	0	1	0	0	B
0	1	1	0	1	0	$\bar{A}$
1	0	1	1	0	0	$\bar{B}$
1	1	1	1	0	0	A + B
1	1	1	1	0	1	A + B + 1
1	1	1	0	0	1	A + 1
1	1	0	1	0	1	B + 1
1	1	1	1	1	1	B – A
1	1	0	1	1	0	B – 1
1	1	1	0	1	1	–A
0	0	1	1	0	0	A AND B
0	1	1	1	0	0	A OR B
0	1	0	0	0	0	0
1	1	0	0	0	1	1
1	1	0	0	1	0	–1

1: H <- TOS:

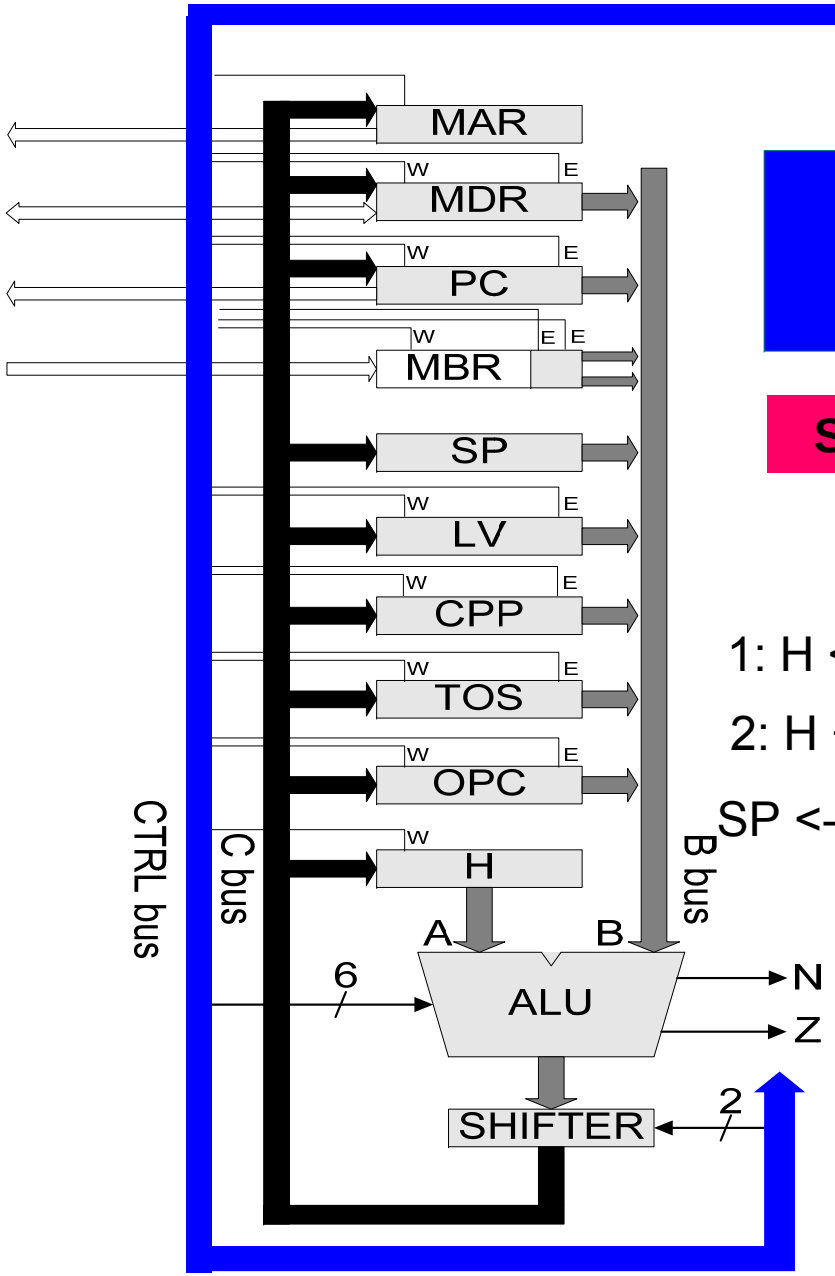
shift	ALU	les frå C buss	mem	B buss
00	010100	1000000000	000	0111



B bus registers

0 = MDR	5 = LV
1 = PC	6 = CPP
2 = MBR	7 = TOS
3 = MBRU	8 = OPC
4 = SP	9-15 none

Eksempel: 2

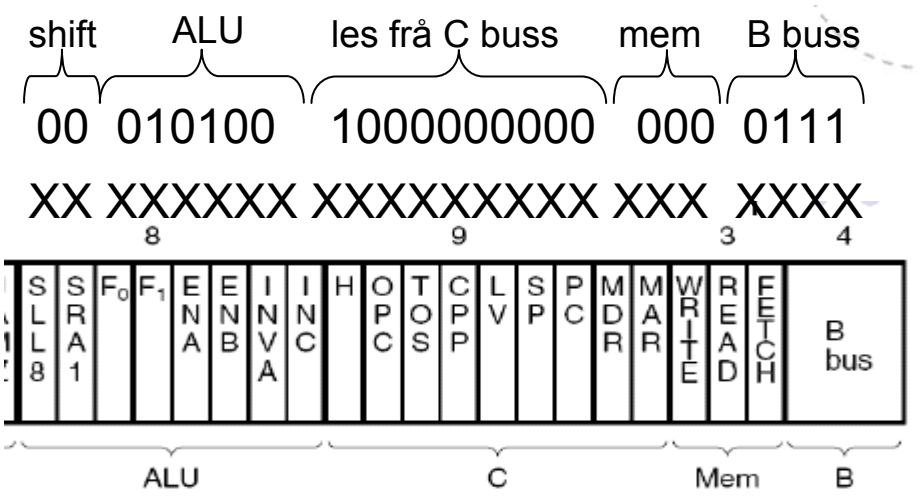


Styreeinheit
Instruction
Decode
Ctrl-logic

SP = TOS + OPC

F_0	F_1	ENA	ENB	INVA	INC	Function
0	1	1	0	0	0	A
0	1	0	1	0	0	B
0	1	1	0	1	0	$\bar{A}$
1	0	1	1	0	0	$\bar{B}$
1	1	1	1	0	0	A + B
1	1	1	1	0	1	A + B + 1
1	1	1	0	0	1	A + 1
1	1	0	1	0	1	B + 1
1	1	1	1	1	1	B – A
1	1	0	1	1	0	B – 1
1	1	1	0	1	1	–A
0	0	1	1	0	0	A AND B
0	1	1	1	0	0	A OR B
0	1	0	0	0	0	0
1	1	0	0	0	1	1
1	1	0	0	1	0	–1

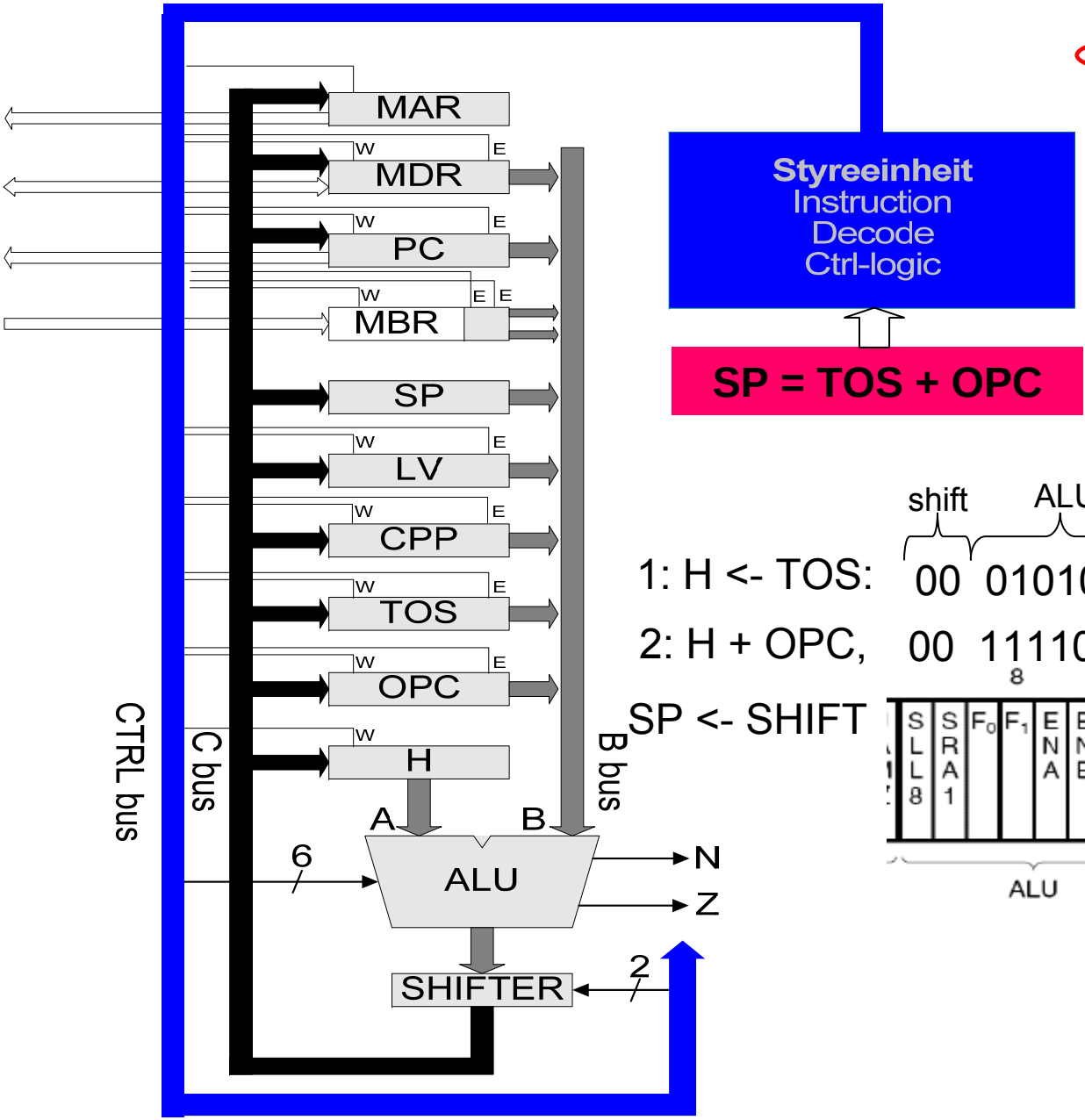
1: H <- TOS:
2: H + OPC,
SP <- SHIFT



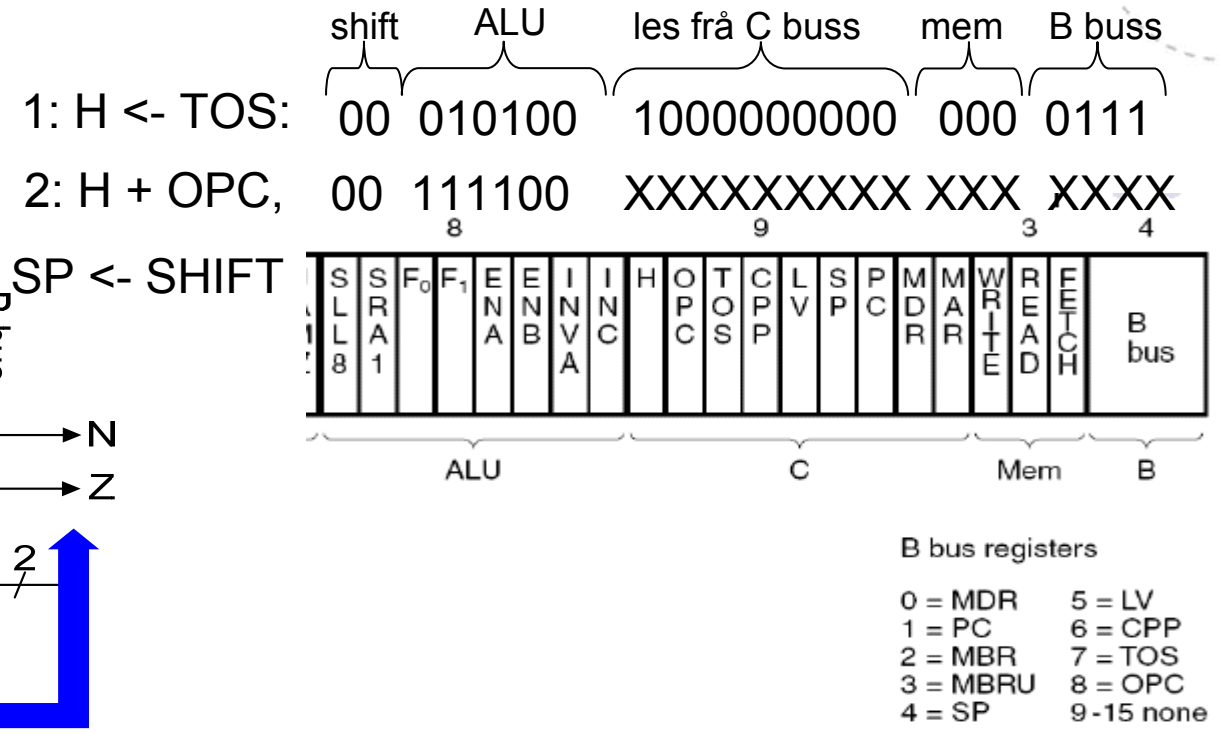
B bus registers

0 = MDR	5 = LV
1 = PC	6 = CPP
2 = MBR	7 = TOS
3 = MBRU	8 = OPC
4 = SP	9-15 none

Eksempel: 2



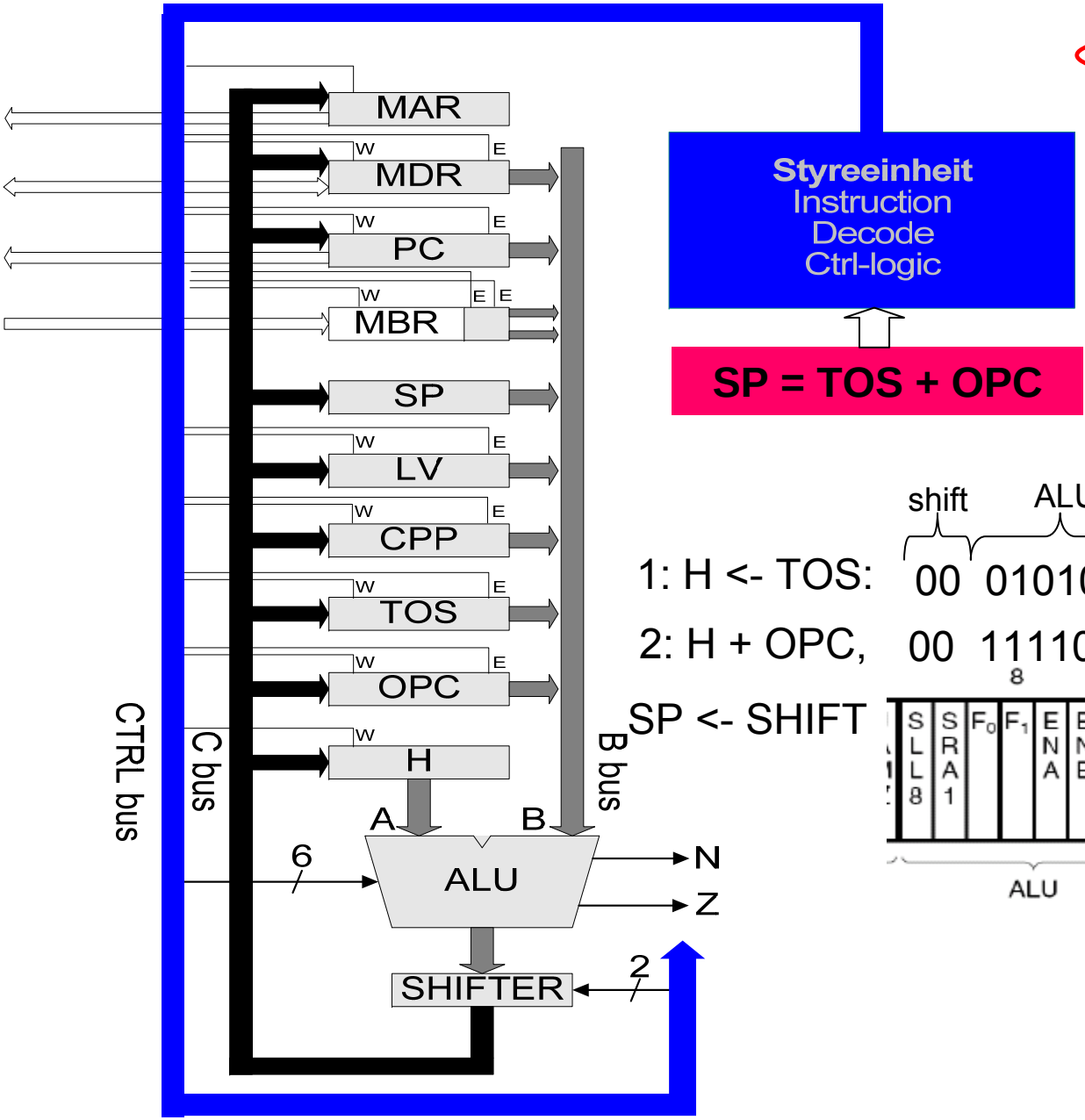
F_0	F_1	ENA	ENB	INVA	INC	Function
0	1	1	0	0	0	A
0	1	0	1	0	0	B
0	1	1	0	1	0	$\bar{A}$
1	0	1	1	0	0	$\bar{B}$
1	1	1	1	0	0	A + B
1	1	1	1	0	1	A + B + 1
1	1	1	0	0	1	A + 1
1	1	0	1	0	1	B + 1
1	1	1	1	1	1	B – A
1	1	0	1	1	0	B – 1
1	1	1	0	1	1	–A
0	0	1	1	0	0	A AND B
0	1	1	1	0	0	A OR B
0	1	0	0	0	0	0
1	1	0	0	0	1	1
1	1	0	0	1	0	–1



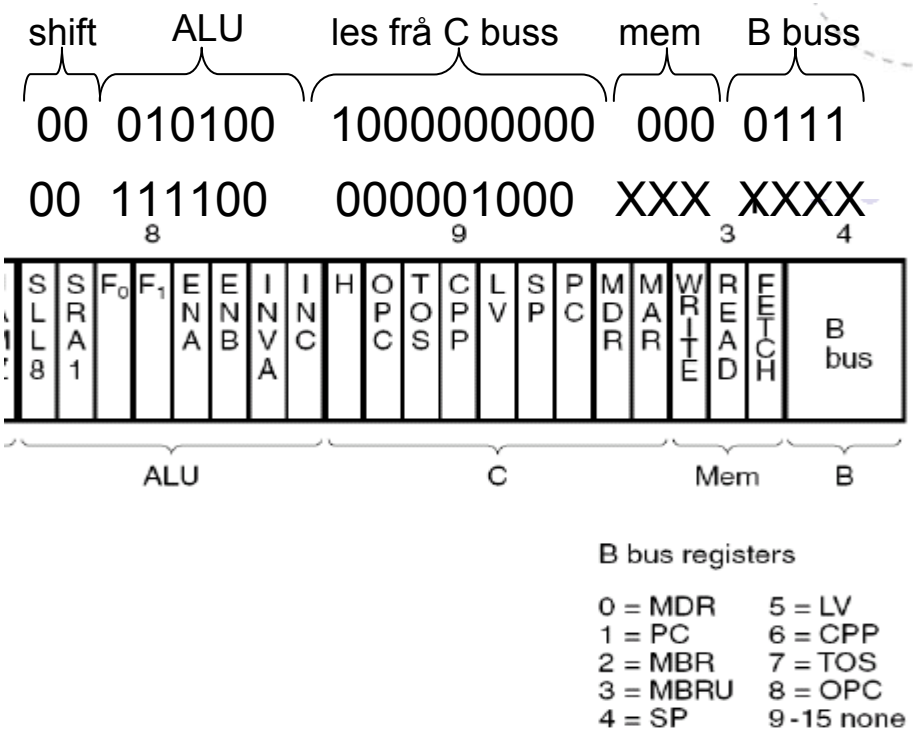
B bus registers

0 = MDR	5 = LV
1 = PC	6 = CPP
2 = MBR	7 = TOS
3 = MBRU	8 = OPC
4 = SP	9-15 none

Eksempel: 2



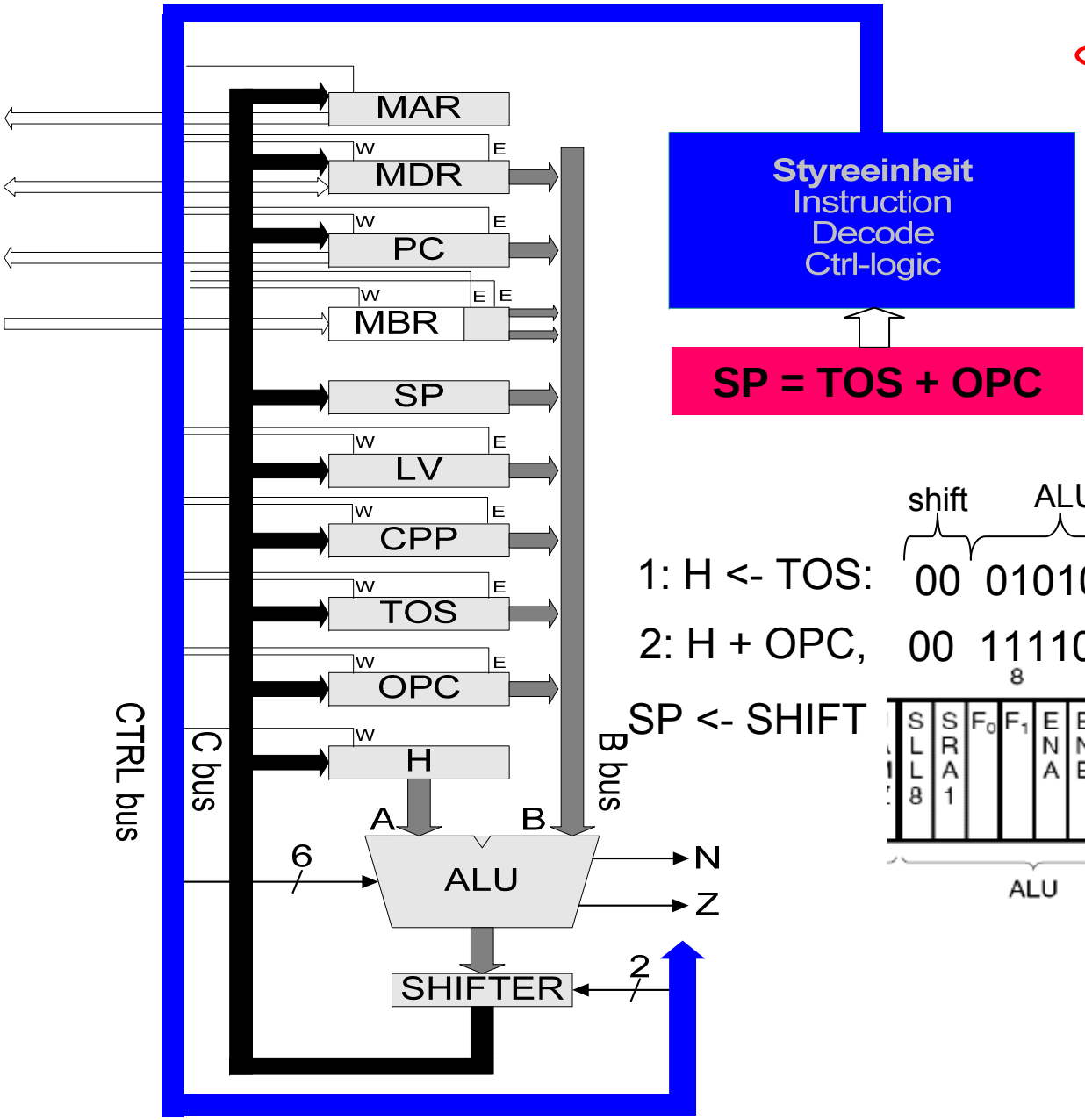
F_0	F_1	ENA	ENB	INVA	INC	Function
0	1	1	0	0	0	A
0	1	0	1	0	0	B
0	1	1	0	1	0	$\bar{A}$
1	0	1	1	0	0	$\bar{B}$
1	1	1	1	0	0	A + B
1	1	1	1	0	1	A + B + 1
1	1	1	0	0	1	A + 1
1	1	0	1	0	1	B + 1
1	1	1	1	1	1	B – A
1	1	0	1	1	0	B – 1
1	1	1	0	1	1	–A
0	0	1	1	0	0	A AND B
0	1	1	1	0	0	A OR B
0	1	0	0	0	0	0
1	1	0	0	0	1	1
1	1	0	0	1	0	–1



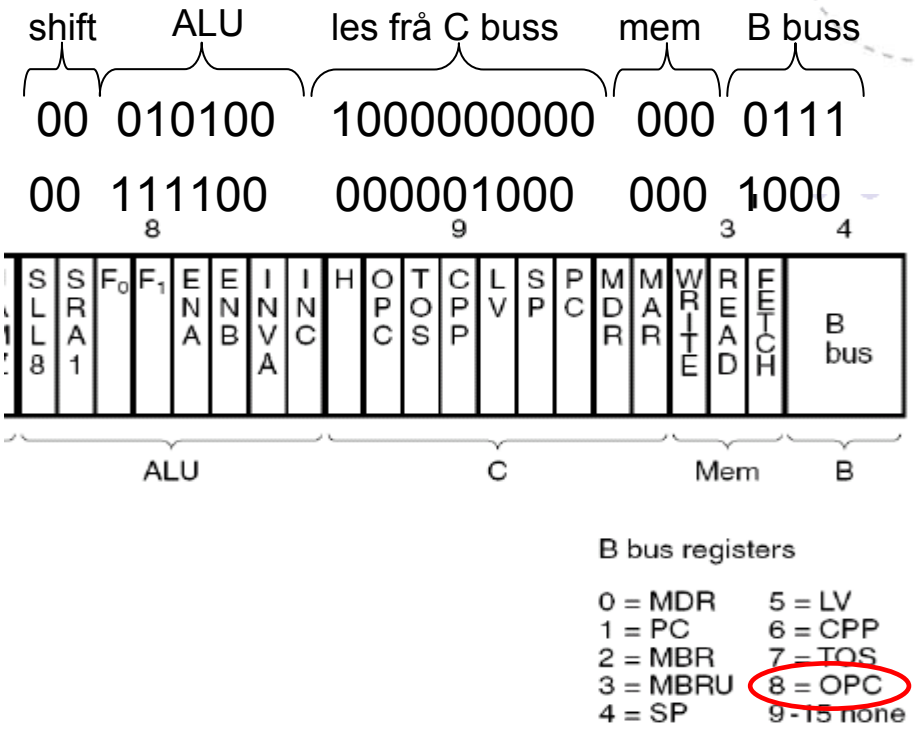
B bus registers

0 = MDR	5 = LV
1 = PC	6 = CPP
2 = MBR	7 = TOS
3 = MBRU	8 = OPC
4 = SP	9-15 none

Eksempel: 2



F_0	F_1	ENA	ENB	INVA	INC	Function
0	1	1	0	0	0	A
0	1	0	1	0	0	B
0	1	1	0	1	0	$\bar{A}$
1	0	1	1	0	0	$\bar{B}$
1	1	1	1	0	0	A + B
1	1	1	1	0	1	A + B + 1
1	1	1	0	0	1	A + 1
1	1	0	1	0	1	B + 1
1	1	1	1	1	1	B – A
1	1	0	1	1	0	B – 1
1	1	1	0	1	1	–A
0	0	1	1	0	0	A AND B
0	1	1	1	0	0	A OR B
0	1	0	0	0	0	0
1	1	0	0	0	1	1
1	1	0	0	1	0	–1

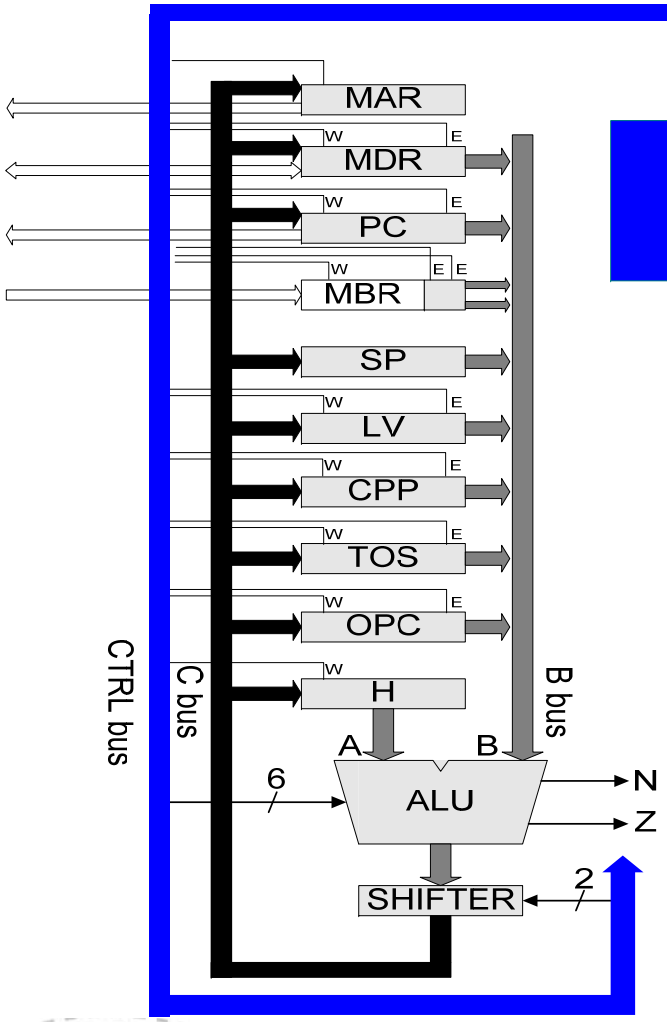


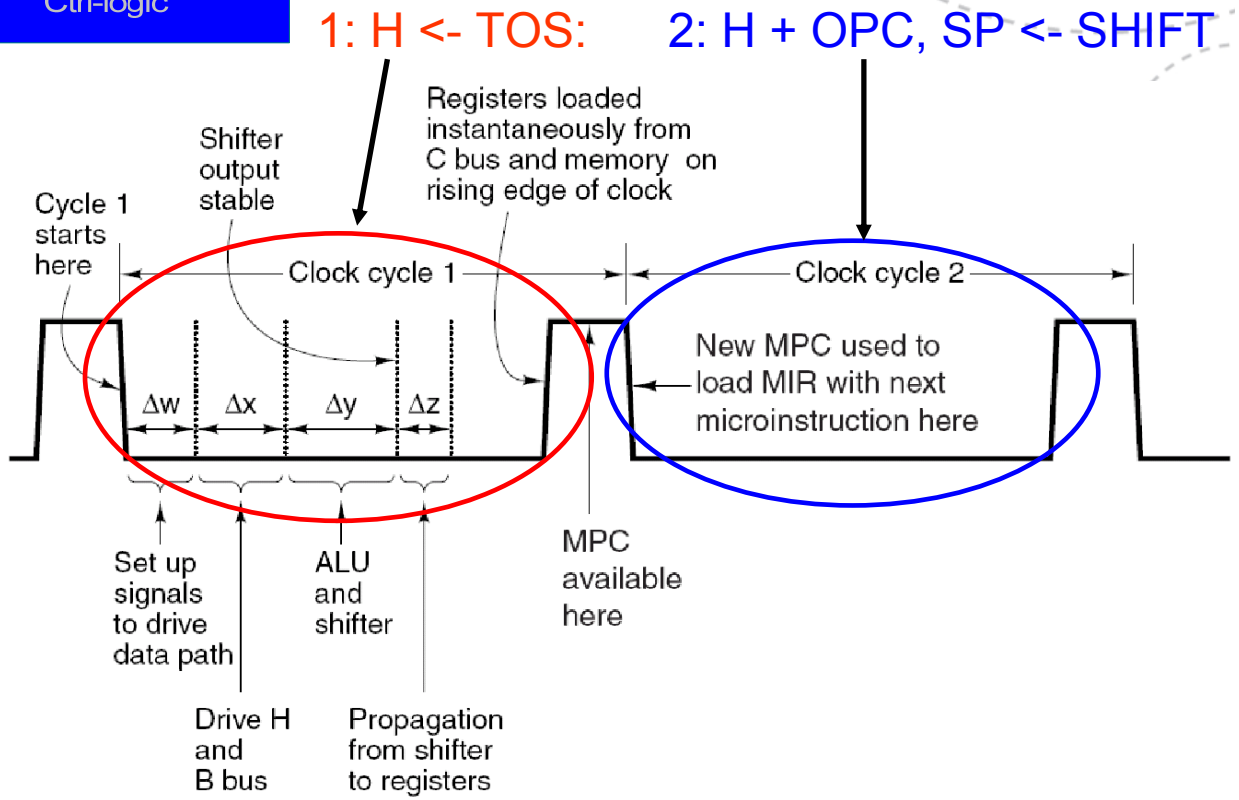
B bus registers

0 = MDR	5 = LV
1 = PC	6 = CPP
2 = MBR	7 = TOS
3 = MBRU	8 = OPC
4 = SP	9-15 none

Utføre instruksjon

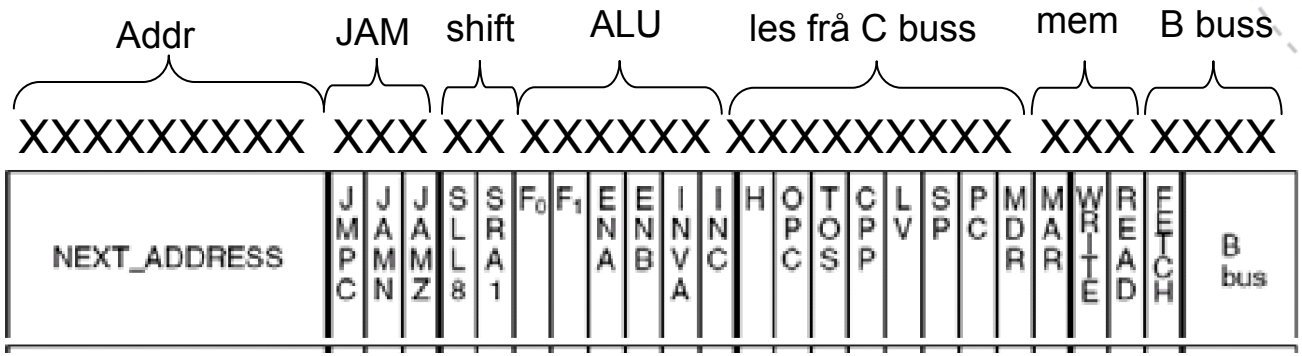
	shift	ALU	les frå C buss	mem	B buss
1:	00	010100	1000000000	000	0111
2:	00	111100	000001000	000	1000





Microinstruksjon Addr feltet

- **Addr gir addressa til neste microinstruksjon**



- Addr peikar på addressa til neste microinstruksjon i instruksjonen
  - Adr 0: Instruksjon 1
    - 1. microinstruksjon ligg på Addr 0, microinstruksjon Addr peikar på Addr 1
    - 2. microinstruksjon ligg på Addr 1, microinstruksjon Addr peikar på 2
    - 3. og siste microinstruksjon ligg på Addr 3, Addr peikar på start ny instruksjon

NTNU
Innovation and Creativity

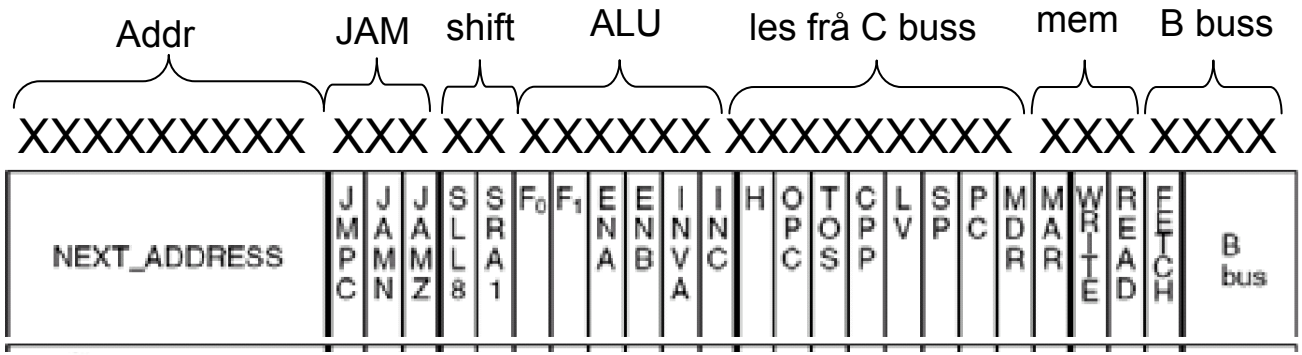
Microinstruksjon Addr feltet

- **Addr gir addressa til neste microinstruksjon**
- **Instruksjon: SP = TOS + OPC (forrige forelesing)**

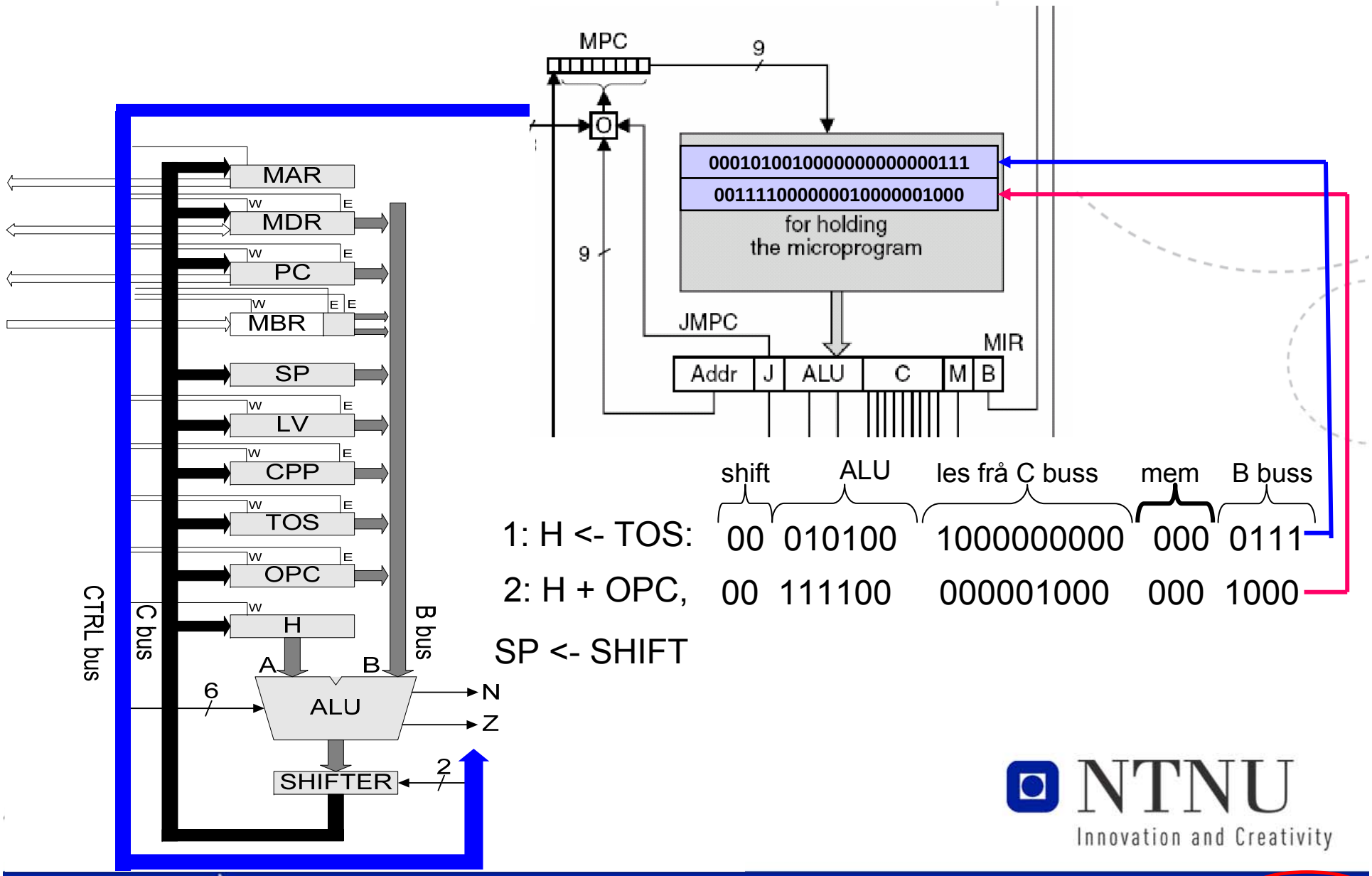
	shift	ALU	les frå C buss	mem	B buss
1: H <- TOS:	00	010100	1000000000	000	0111
2: H + OPC, SP <- SHIFT:	00	111100	000001000	000	1000

- Addr kan då sjå slik ut (H <- TOS ligg på Addr 0):

	Addr	shift	ALU	les frå C buss	mem	B buss
1: H <- TOS:	000000001 xxx	00	01010	1000000000	000	0111
2: H + OPC, SP <- SHIFT:	000000002 xxx	00	111100	000001000	000	1000



Fleire microinstr. for instr.



	shift	ALU	les frå C buss	mem	B buss
1: H <- TOS:	00	010100	1000000000	000	0111
2: H + OPC,	00	111100	000001000	000	1000

SP <- SHIFT

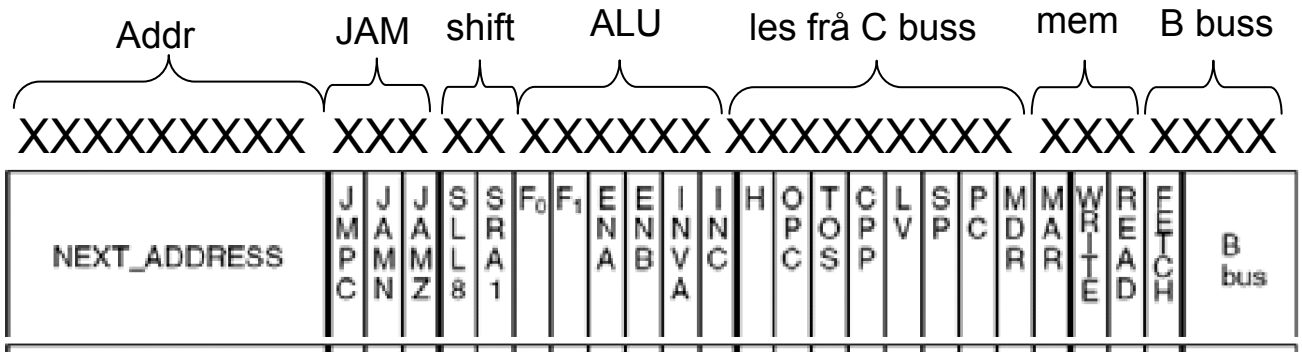
NTNU
Innovation and Creativity

Instruksjonstypar I/O avbrudd

- Programstyrt I/O
  - Prosessor sjekker om data er klart
- Avbruddsinitiert I/O
  - I/O-enhet sier ifra når data er klart
  - Unngår å kaste bort prosessortid
- Adresse til avbruddsrutine
  - Ikke-vektorisert: Fast, en for alle avbrudd
  - Vektorisert: Avbruddskilde oppgir adresse
- Mer om avbrudd senere

Må ha hopp inn i microinstruksjon

- **Utvidar med JAM**
  - JAMZ: Sjekkar Z-flagget (zero) til ALU
  - JAMN: Sjekkar N-flagget (negativ) til ALU
  - JMPC: Ekstra hoppting kjem seinare
- No mogleg å detektera N og Z for å kunne bestemme hopp
  - Kan sjekke resultat frå beregning utført av ALU
  - Kan bruke resultatet til betingahopp

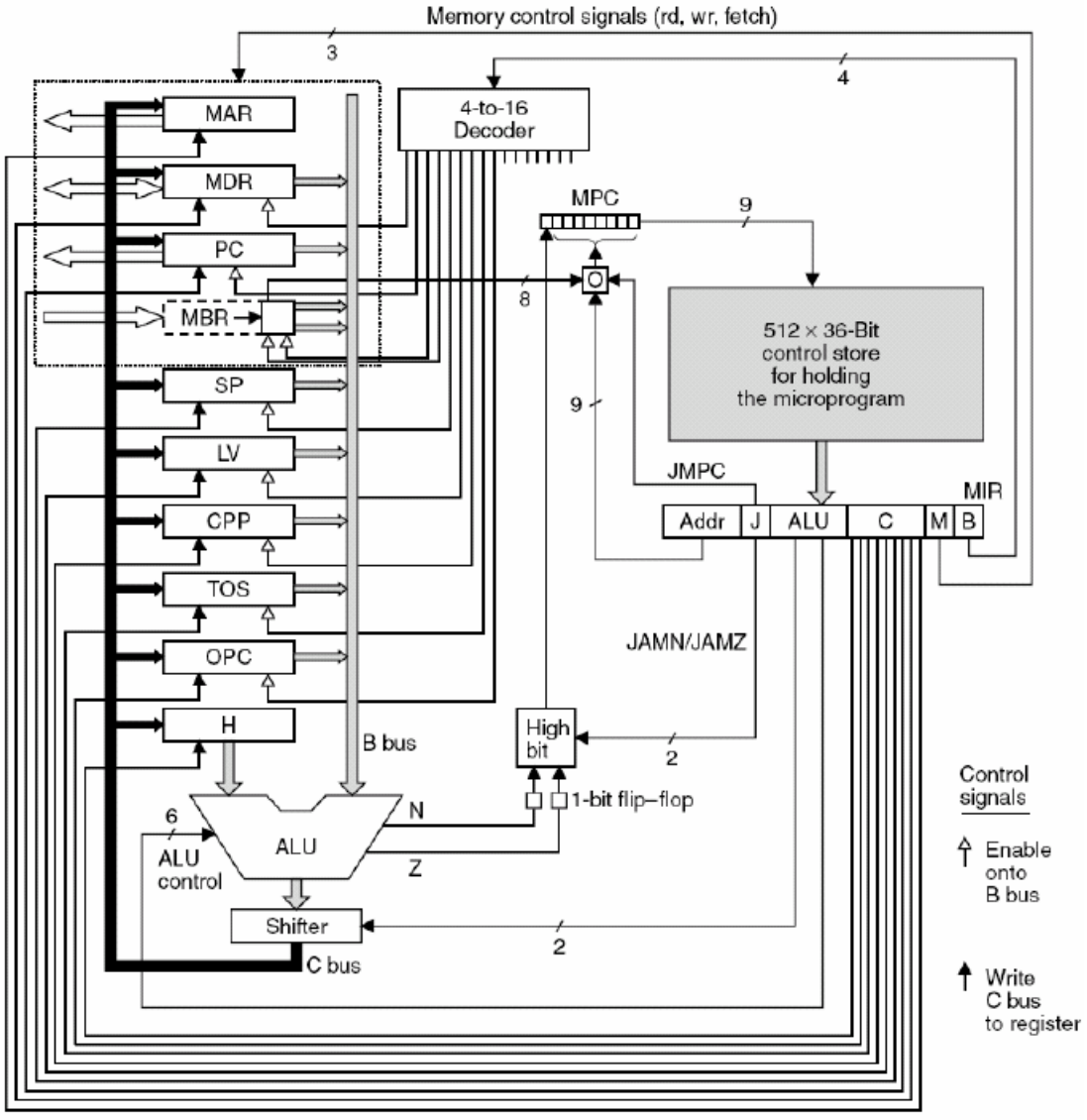


IJVM-implementasjonar

- Kva kan gjerast for å auke ytinga
  - Auke mengda logikk og **endring av arkitektur**
    - Instruction Fetch Unit
    - Samleband
    - Instruksjonskø

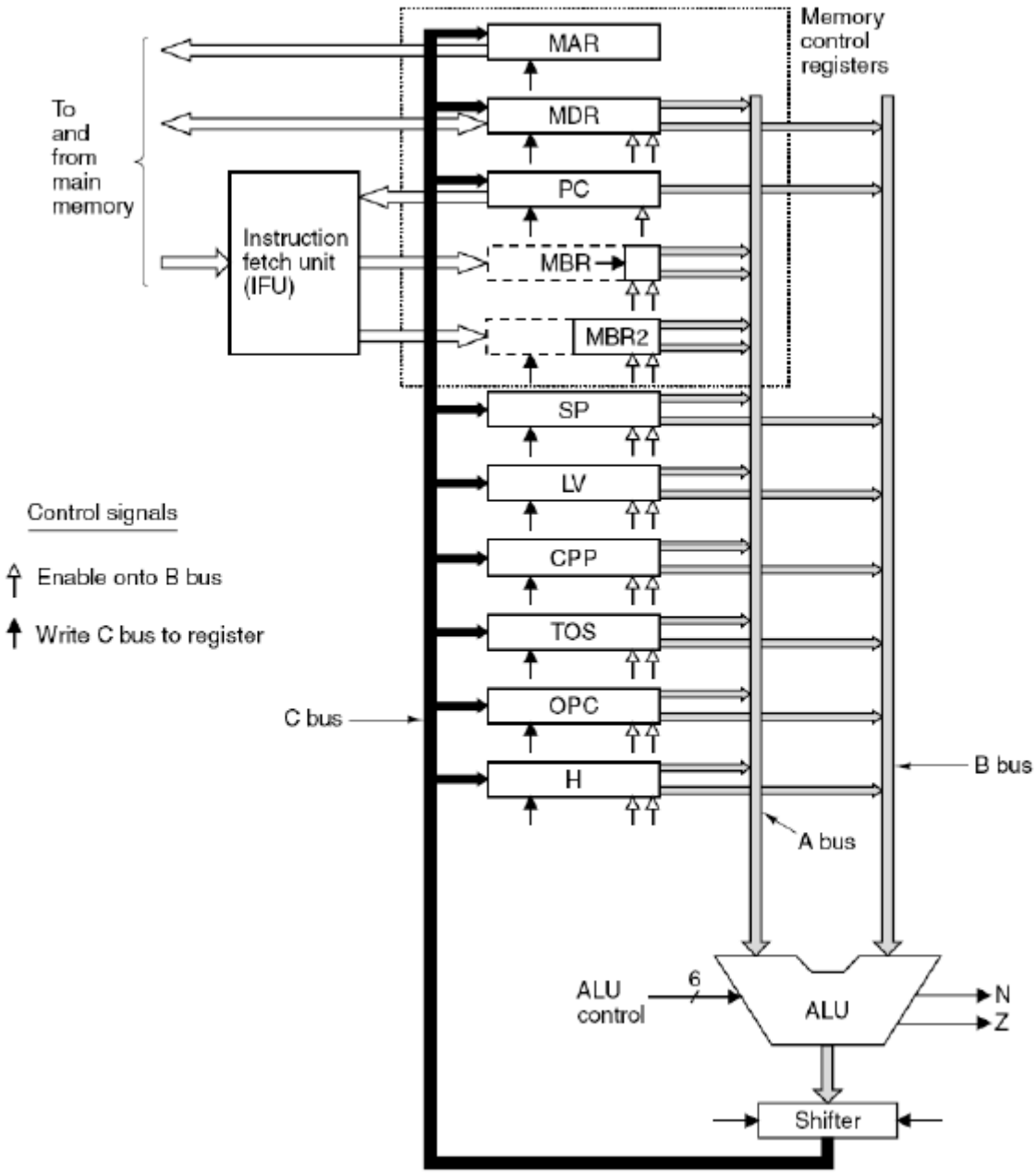
IJVM Innfører A-buss

- Endrar ”data path”
  - Samankopling av komponentar

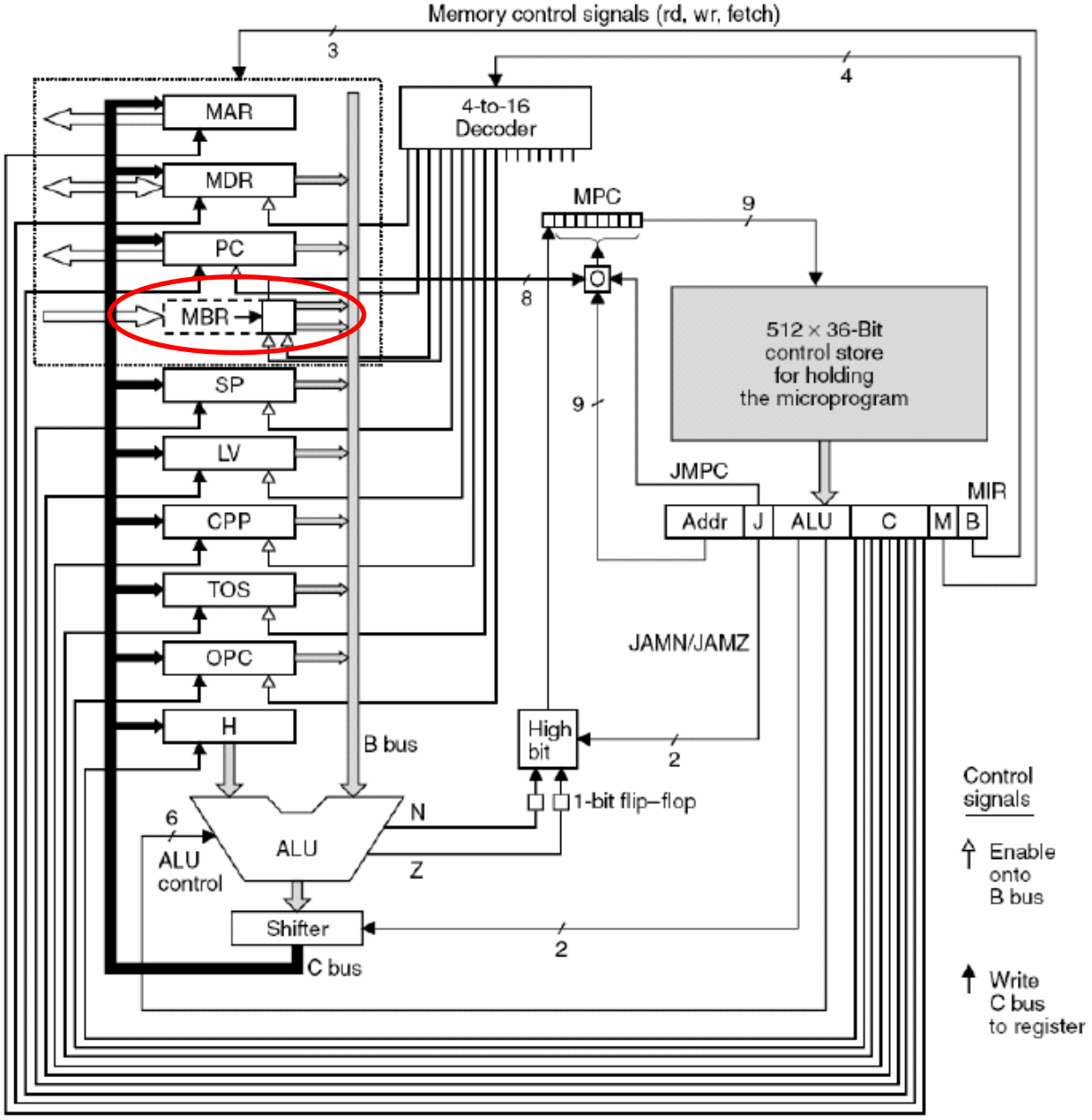


IJVM Innfører A-buss

- Endrar ”data path”
  - Samankopling av komponentar
- ALU operasjonar:
  - To tlifeldige register
  - Sparar ein microinstruksjor
    - H-reg som mellomlagring



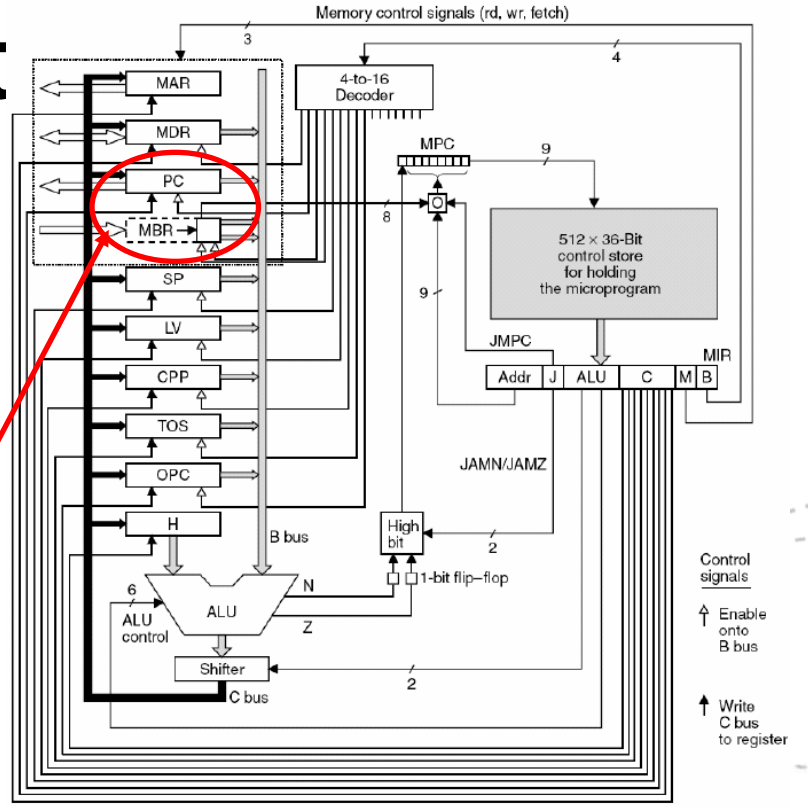
Instruction Fetch Unit

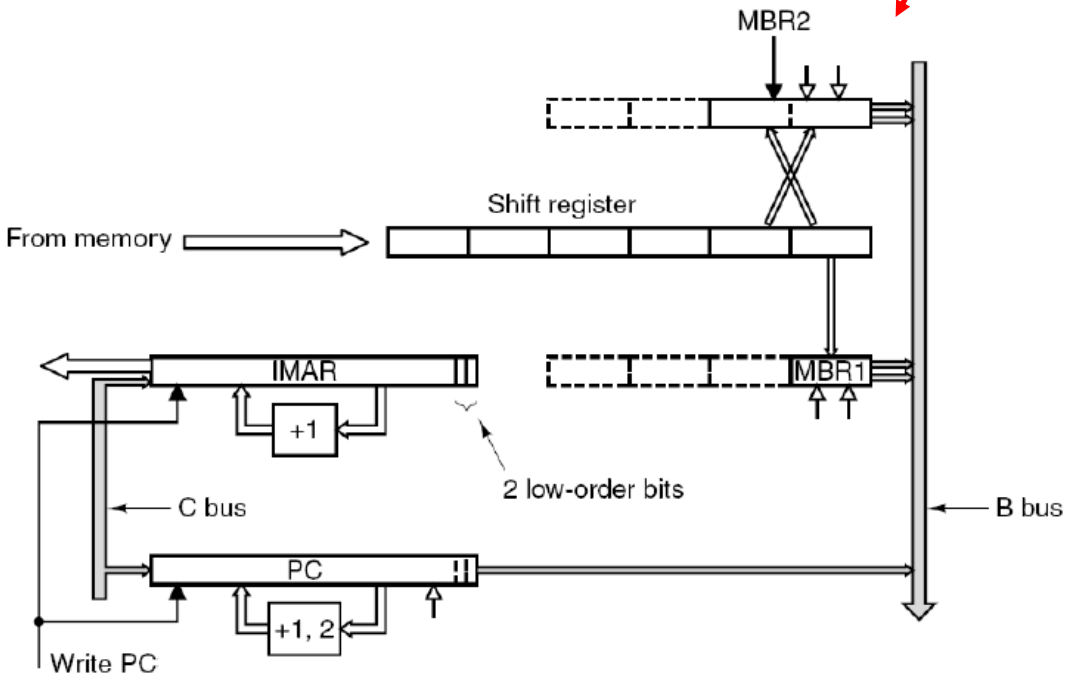


NTNU
Innovation and Creativity

Instruction Fetch Unit

- Instruksjon
  - 8 bit (1 byte)
- Minne grensesnitt
  - 32 bit (4 Byte)
- Brukar
  - 32 bit buss
  - Hentar 8 bit
  - 24 bit ubrukt

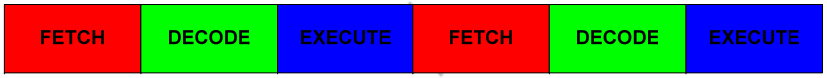


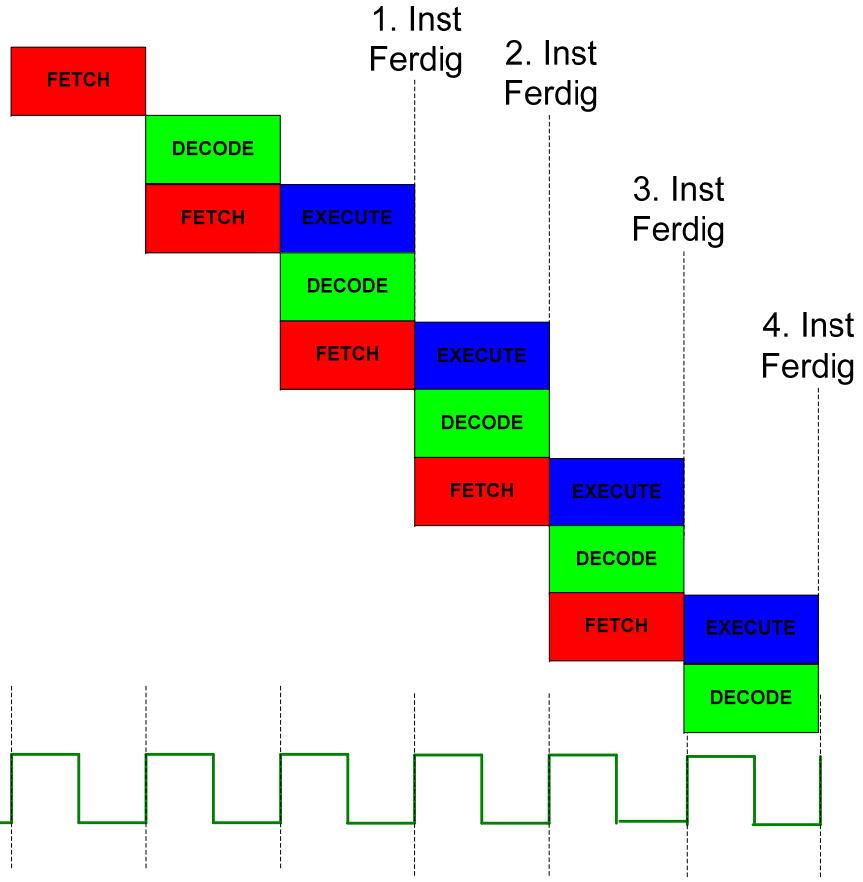


NTNU
Innovation and Creativity

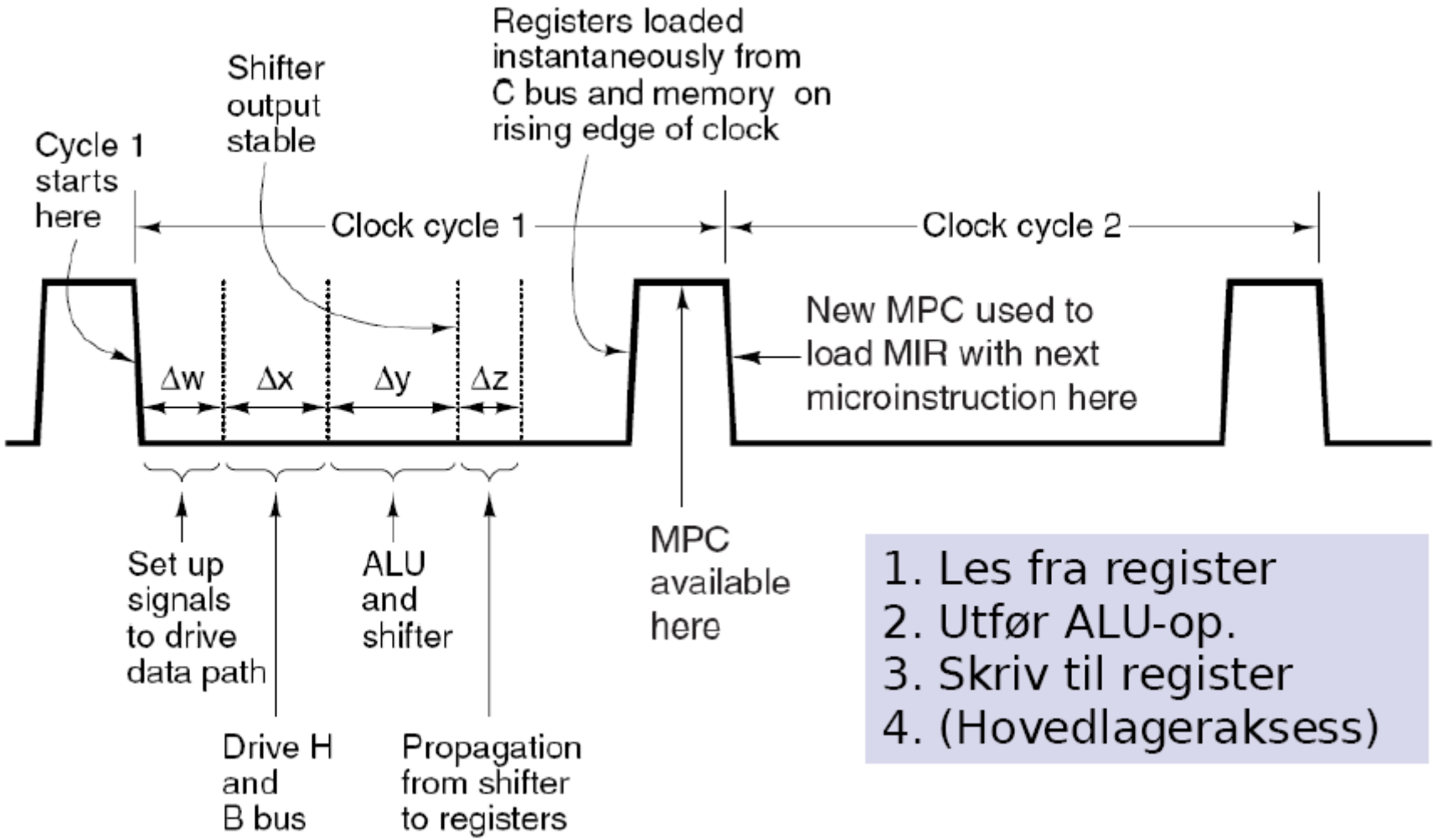
Samleband (Pipelining)

- **Fetch Decode Execute**
- Dei tre prinsipielle stega i instruksjonutføring
- Lagar oss ein tretrinns styreeining
  - 1: Fetch
  - 2: Decode
  - 3: Execute
  - Kan startast uavhengig av kvarandre
- Raskare
  - Enkle trinn
  - Kortare tid





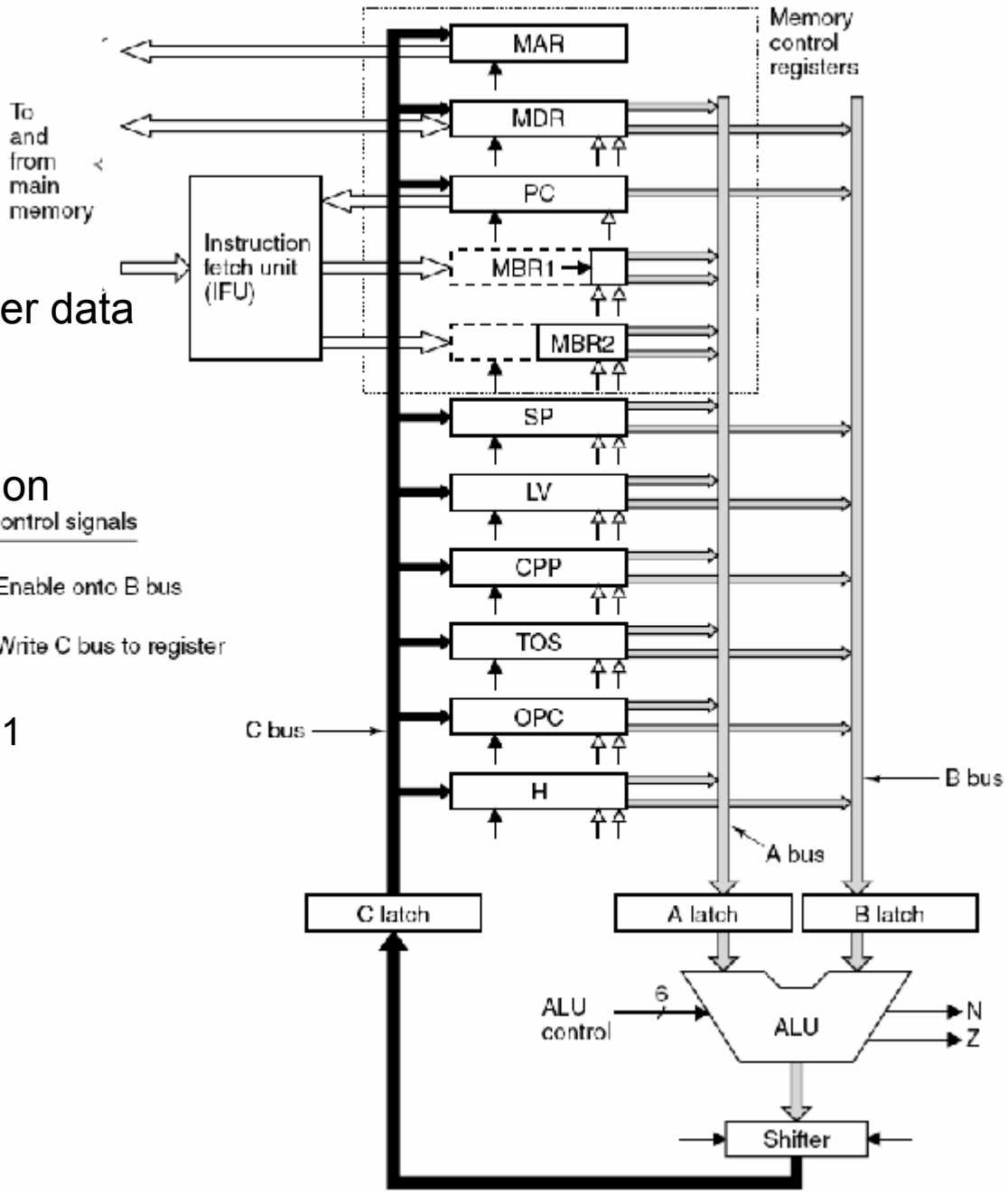
Samleband (Pipelining)

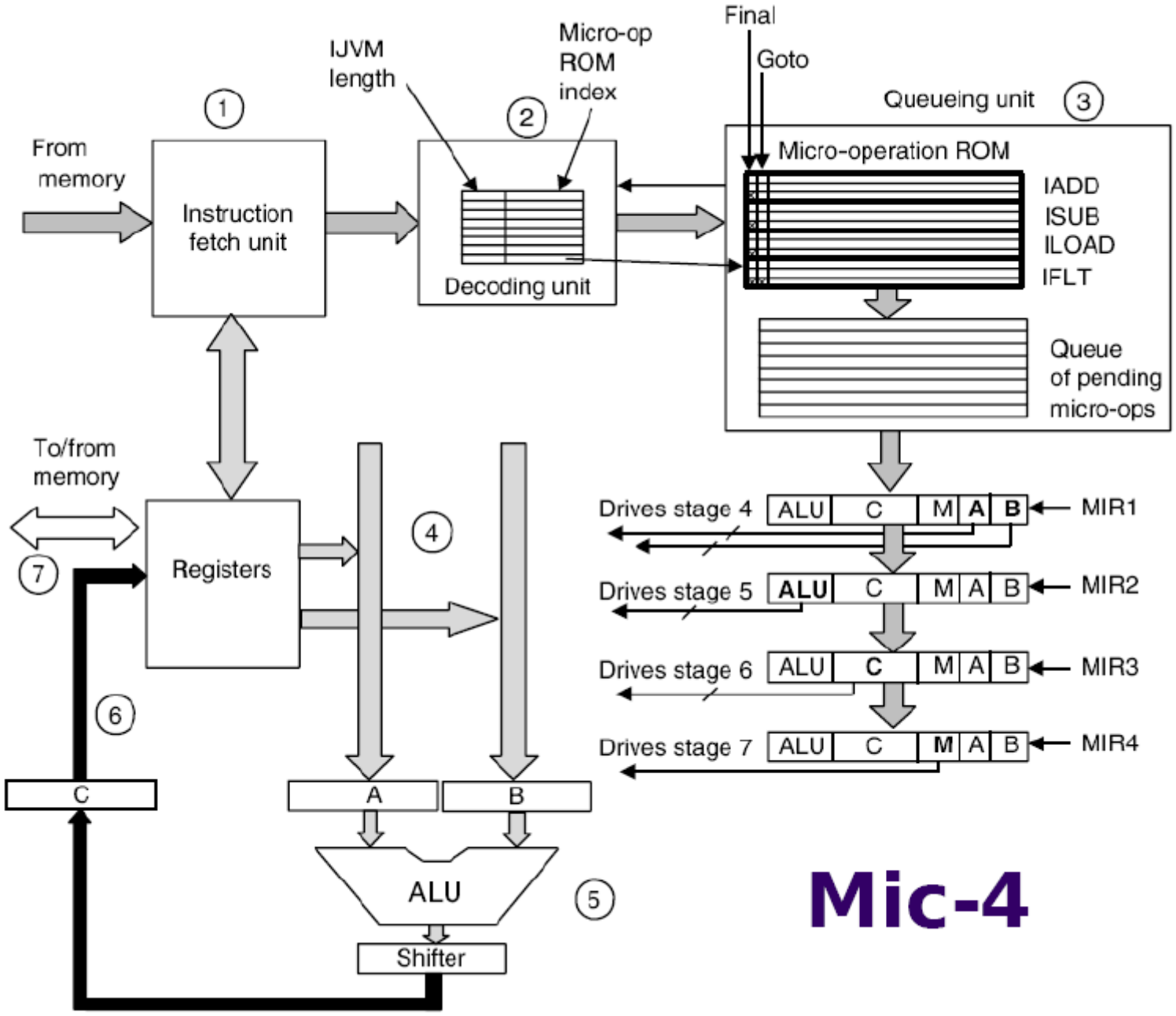


1. Les fra register
2. Utfør ALU-op.
3. Skriv til register
4. (Hovedlageraksess)

Samleband

- A-B-C-register
  - ”Sluse” mellom utføringssteg
  - Lastast i slutten av clk-syklus
  - To Instruksjonar skriv ikkje over data for kvarandre
  - Her 4 klokkeperiodar instruksjon
  - Klokkeperiode kortare
    - Kortare tid på å utføre 4 enn 1



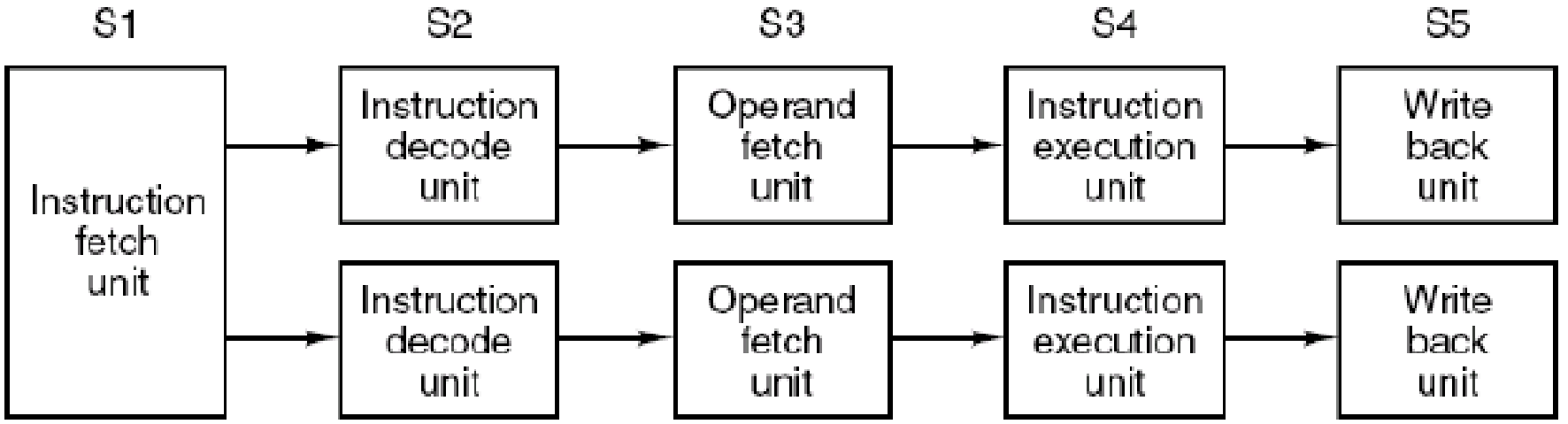

Final
Goto
IJVM
length
Micro-op
ROM
index
Queueing unit
3
1
2
From
memory
Instruction
fetch unit
Decoding unit
Micro-operation ROM
IADD
ISUB
ILOAD
IFLT
Queue
of pending
micro-ops
To/from
memory
Registers
4
7
6
5
C
A
B
ALU
Shifter
Drives stage 4
Drives stage 5
Drives stage 6
Drives stage 7
ALU
C
M
A
B
MIR1
MIR2
MIR3
MIR4

Mic-4

Parallellitet (Superscalare SIMD og MIMD)

- Essensielt for å auke ytinga
- To typar:
  - 1) Instruksjonsnivåparallellitet
    - Fleire instruksjonar utføres samtidig (1 prosessor)
    - Samleband
      - Eks:
        - Unødvendig at ALU er ledig mens CPU leser instruksjon
        - Kan bruke ALU mens ein kopierar data mellom register
    - Superskalaritet: Duplisering av CPU-komponentar
  - 2) Prosessornivåparallellitet
    - En prosessor er bra, fleire er betre?

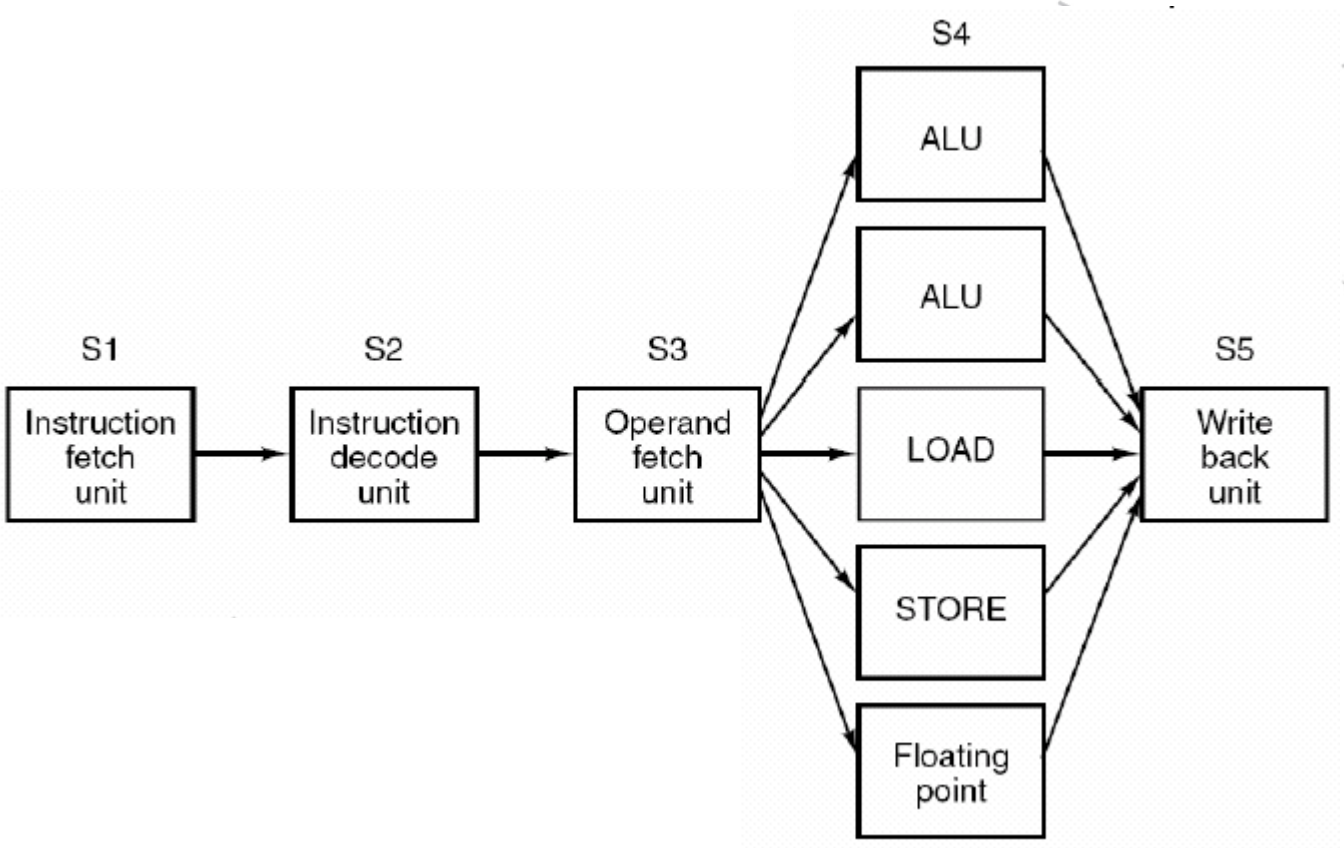
Superskalaritet



- Hentar to instruksjonar om gangen
- Dupliserer steg 2-5
- Har dermed opptil 10 instruksjonar under utføring
- Figuren tilsvarer ca. den første Pentium prosessoren

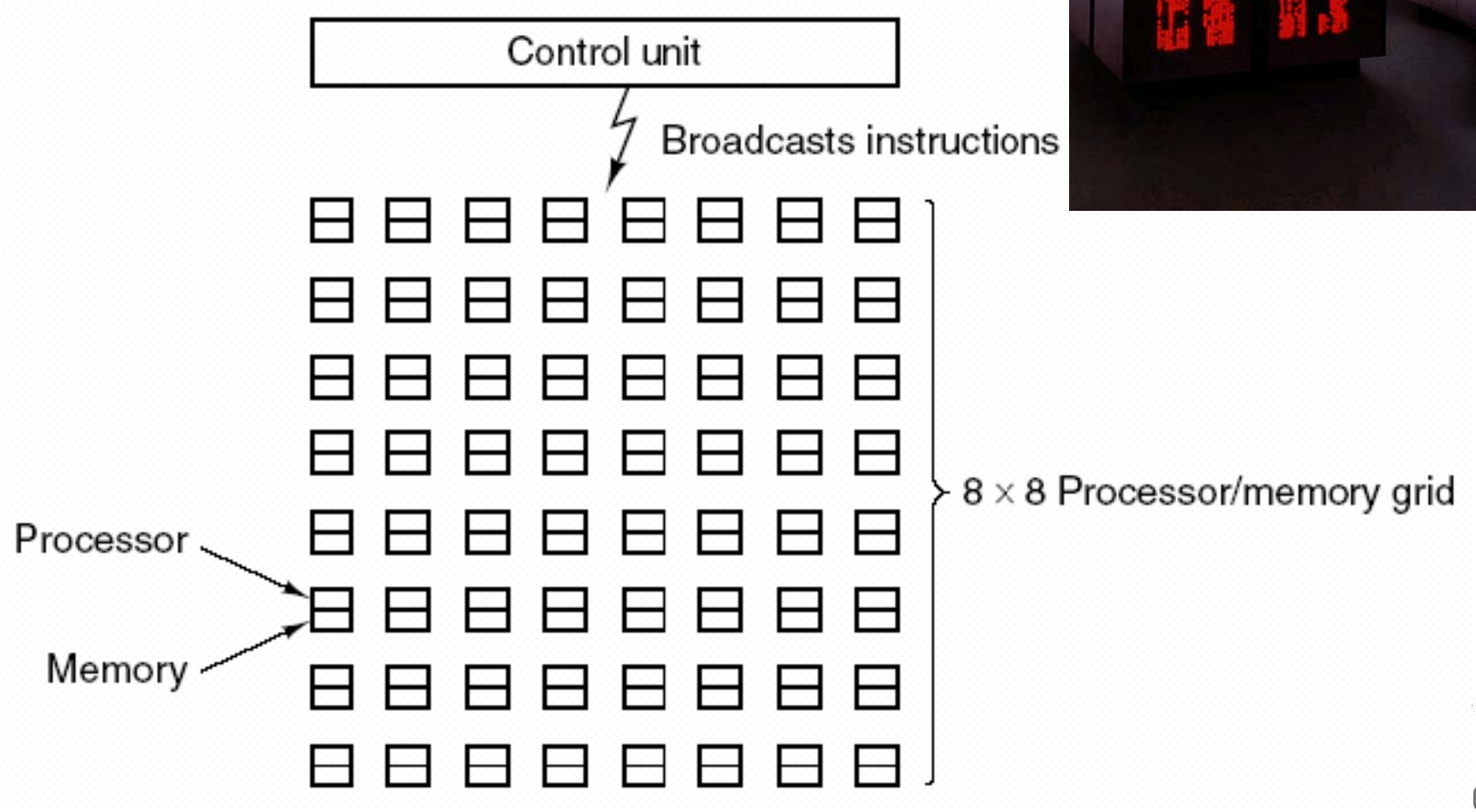
Superskalaritet

- Dyrt å duplisere alt 4x (Areal (mengd transistorar))
- Utføring tar mest tid
- Figur tilsvarer Pentium II
- Dupliserar mest tidkrevjande steg



Meir parallellitet 1

Array computer (SIMD)



Creativity

Thinking Machines Corporation

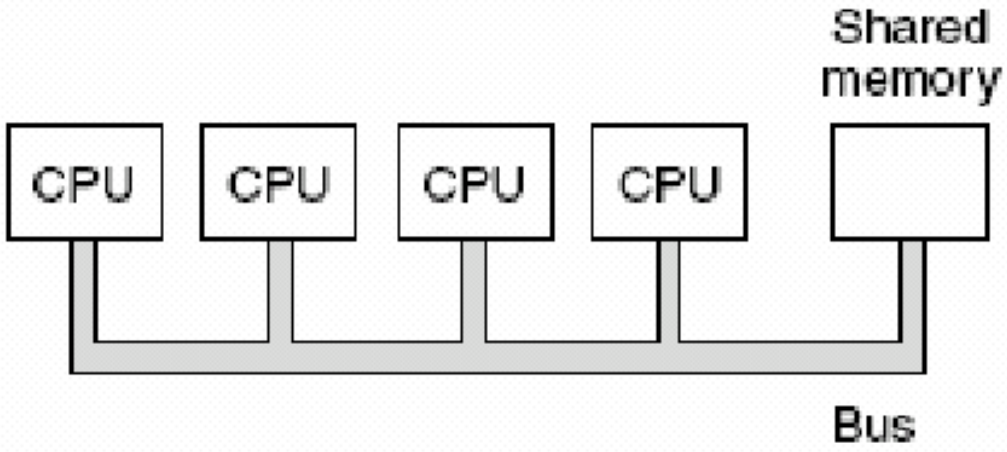
- Connection Machine models CM-1 og CM-2, 65 536 serie prosessorar
- Spesial kompilator overset kode for parallell SIMD program
- Store rundt 1990
  - DARPA redd for å gå glipp av noko
  - Ein av dei mest lovande firma
  - Maskiner stort sett kunn brukande innan forsking
- Konkurs 1994

Connection Machine's cameo in Jurassic Park

NTNU
Innovation and Creativity

Meir parallellitet 2

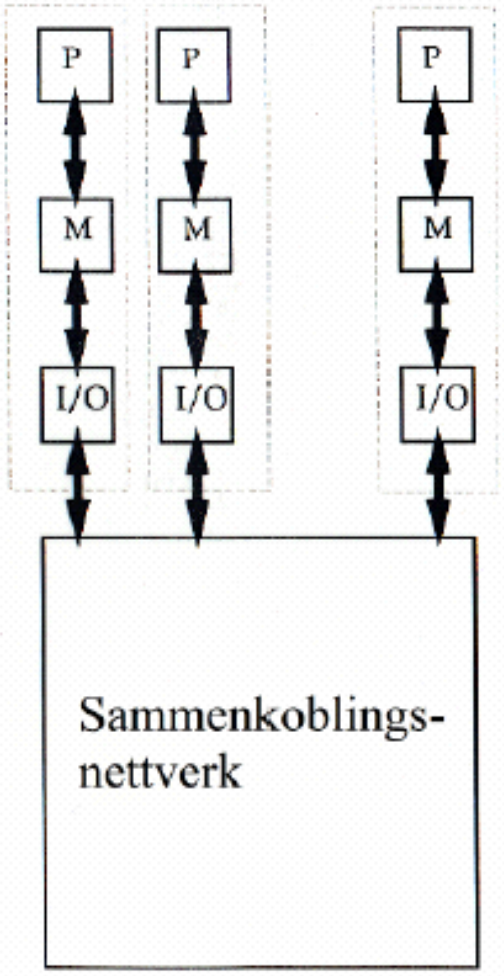
Multiprosessor (MIMD)



- Uavhengige prosessorer (forskjell fra array computer)
- Delt hovedlager
- Buss blir flaskehals etter hvert
  - Kan ha litt lokalt lager til hver prosessor

Meir parallellitet 3

Multidatamaskin (MIMD)

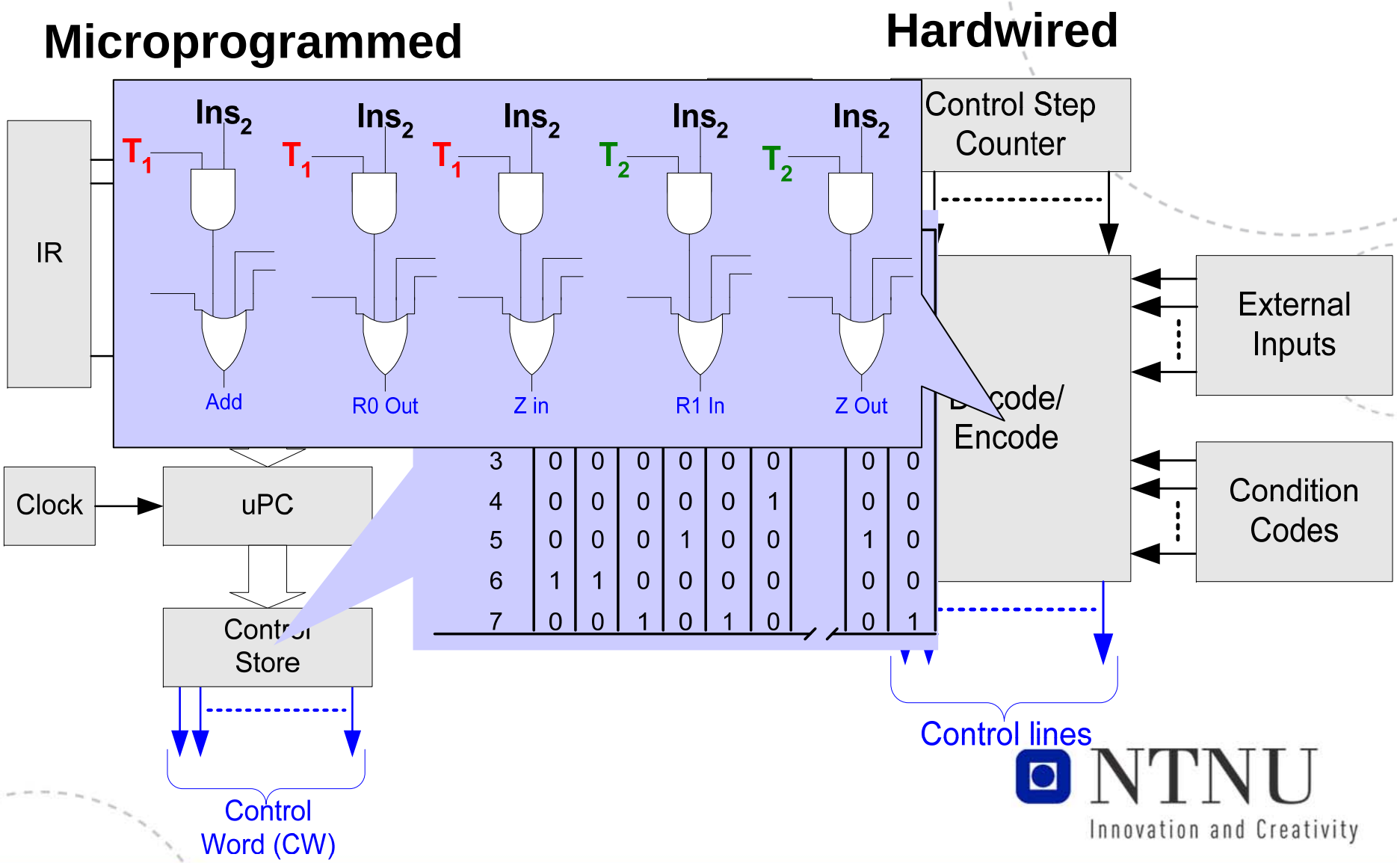


- Distribuert hovedlager
- "Maskiner" med nesten 10.000 prosessorer laget
- Lettere å lage enn multiprosessor
- Men vanskeligere å programmere
  - Ikke delt lager

Instruksjonset og styreeinheit 1

- To hovud typar styreeinheit
- Microprogram
  - Instruksjonar er bygd opp av mikroinstruksjonar
  - Maskin i styreeinheiten som utfører instruksjonane stegvis
  - Fleksibel (kan endre på instruksjonssete)
  - Mange typar instruksjonar og adresseringsmåtar
- ”Hardwired”
  - Kunn maskinvare
  - Ei maskinvare statemaskin som styreeinheit (eller helst berre logic)
  - Kan ikkje endrast (oppdaterast)
  - Rask

Instruksjonset og styreeinheit 2



Instruksjonsett type CISC/RISC

- CISC (Complex Instruction Set Computer)
  - Avanserte instruksjonar (kan utføre mykje)
    - Bruke fleire einheitar
    - Nærare høgnivå språk

ADD MinneAdrX, MinneAdrY, MinneAdrSvar

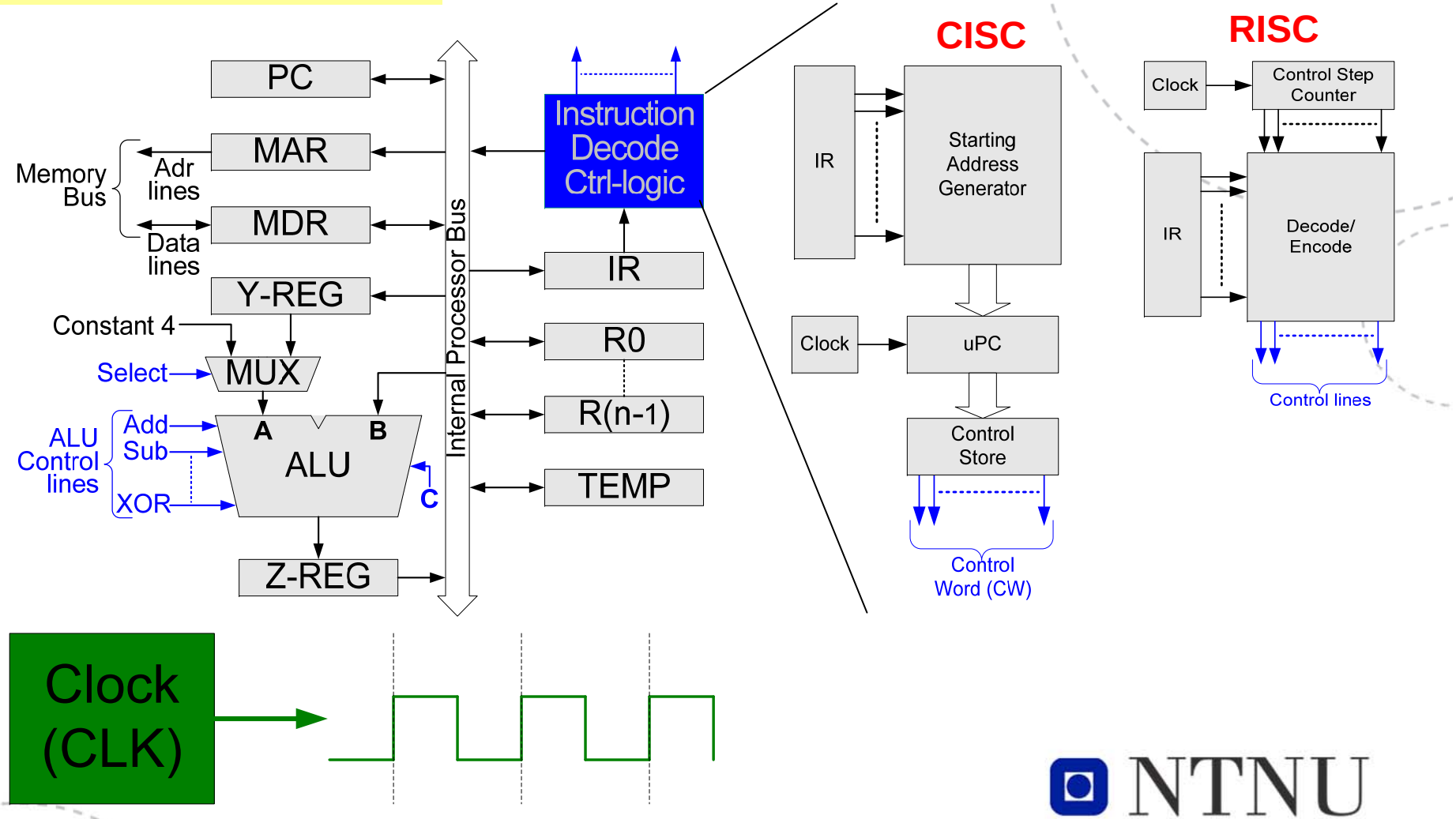
  - Fleksibelt instruksjonsett (Viss mikroprogram kan oppdaterast/endrast)
  - Variabel lengde på instruksjonar
  - Variabel tid på å utføre instruksjonar
  - Mikroprogram (ikkje alltid)
  - Mange typar instruksjonar og adresseringsmåtar
- RISC (Reduced Instruction Set Computer)
  - Enkle instruksjonar
    - Register til register instruksjonar
    - Bruke mange enkle instruksjonar på å utføre oppgåver

ADD R0, R1

  - Fast instruksjonsett (Hardwired)
  - Fast lengde på instruksjonar
  - Fast tid på å utføre instruksjonar
  - Ei maskinvare statemaskin som styreeinheit (eller helst berre logic)
  - Kunn ein type adresering LOAD/STORE
  - Rask

CISC: ADD MinneAdrX, MinneAdrY, MinneAdrSvar

RISC: ADD R0, R1



CISC/RISC Prosessorar

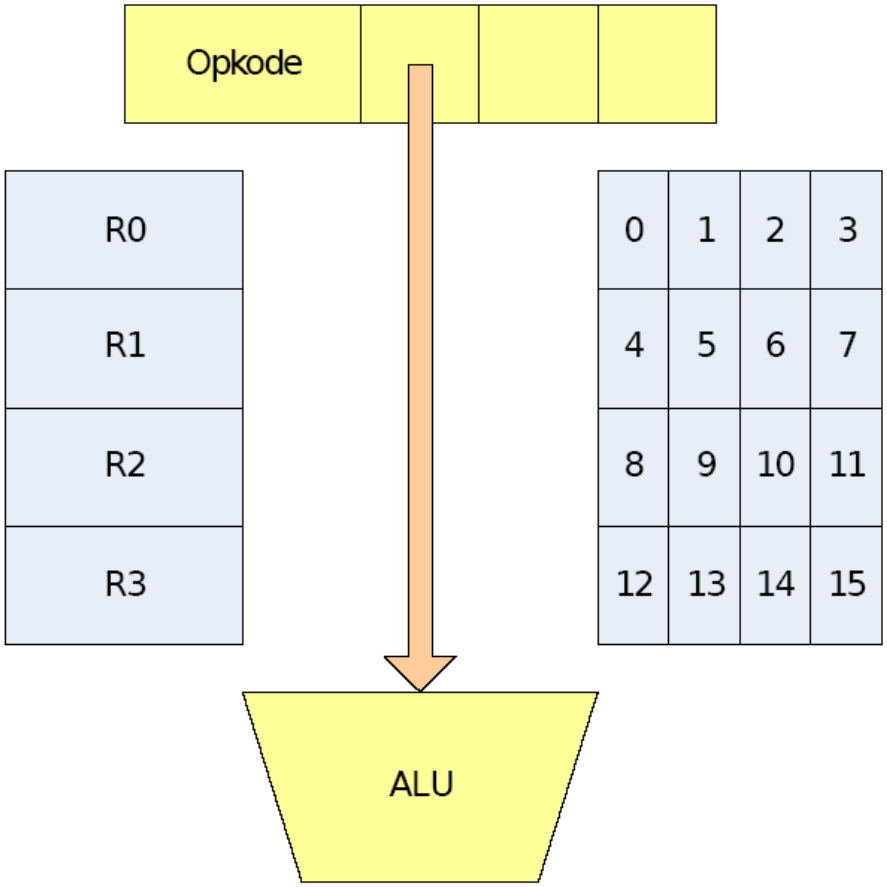
	CISC			RISC	
Prosessor	IBM 370/168	VAX 11/780	Intel 80486	SPARC	MIPS R4000
Årstall	1973	1978	1989	1987	1991
# instruksjoner	208	303	235	69	94
Instr. størrelse (B)	2-6	2-57	1-11	4	4
# adr.modi	4	22	11	1	1
# gen. registre	16	16	8	40-520	32
Mikroprogram (KB)	420	480	246	-	-

Adresseringsmodi

- Hvordan skal vi oppgi en operand?
- Adresseringsmodus
  - Regel for tolking av adressefelt
  - Mål: Effektiv adresse ➔ Peker på operand
  - Spesifiseres av opkode eller eget modus-felt i instruksjonen
  - Forskjellige operander i samme instruksjon kan ha forskjellig adresseringsmodus

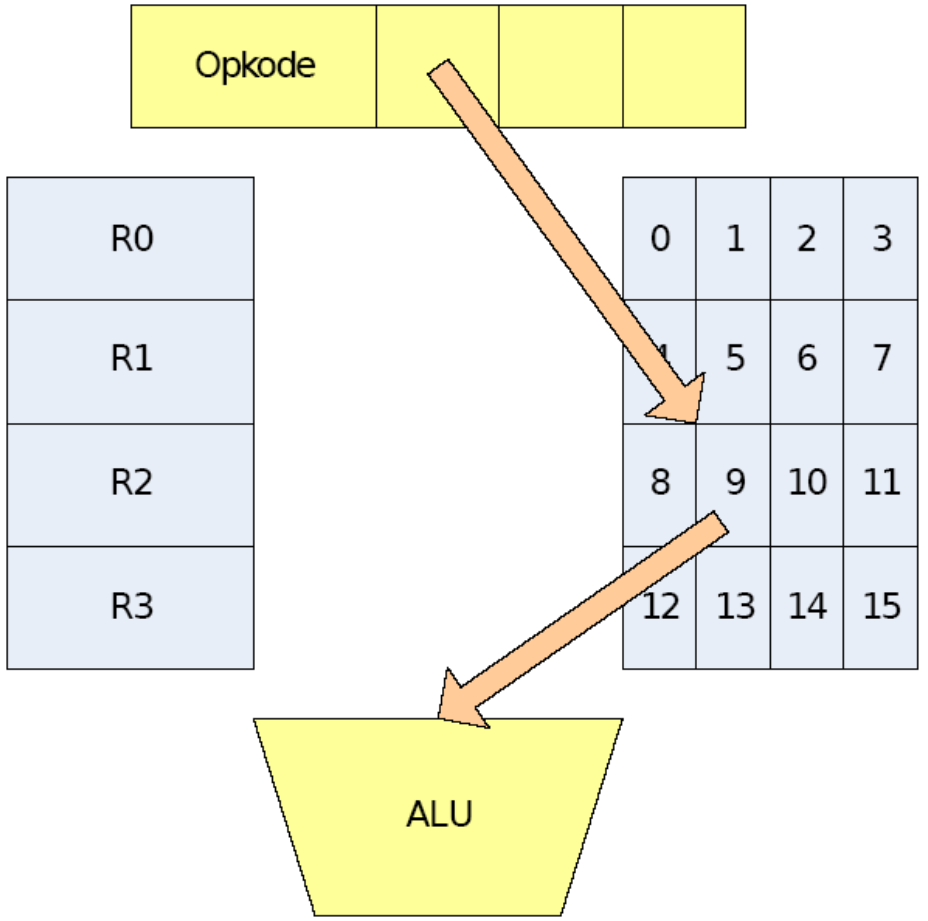
”Immediate” adressering

- Operanden ligger direkte i instruksjonen
- Er dermed tilgjengelig uten videre
- Størrelse på operand begrenset av feltlengde
- Kan bare brukes til konstanter – verdi bestemmes ved kompilering



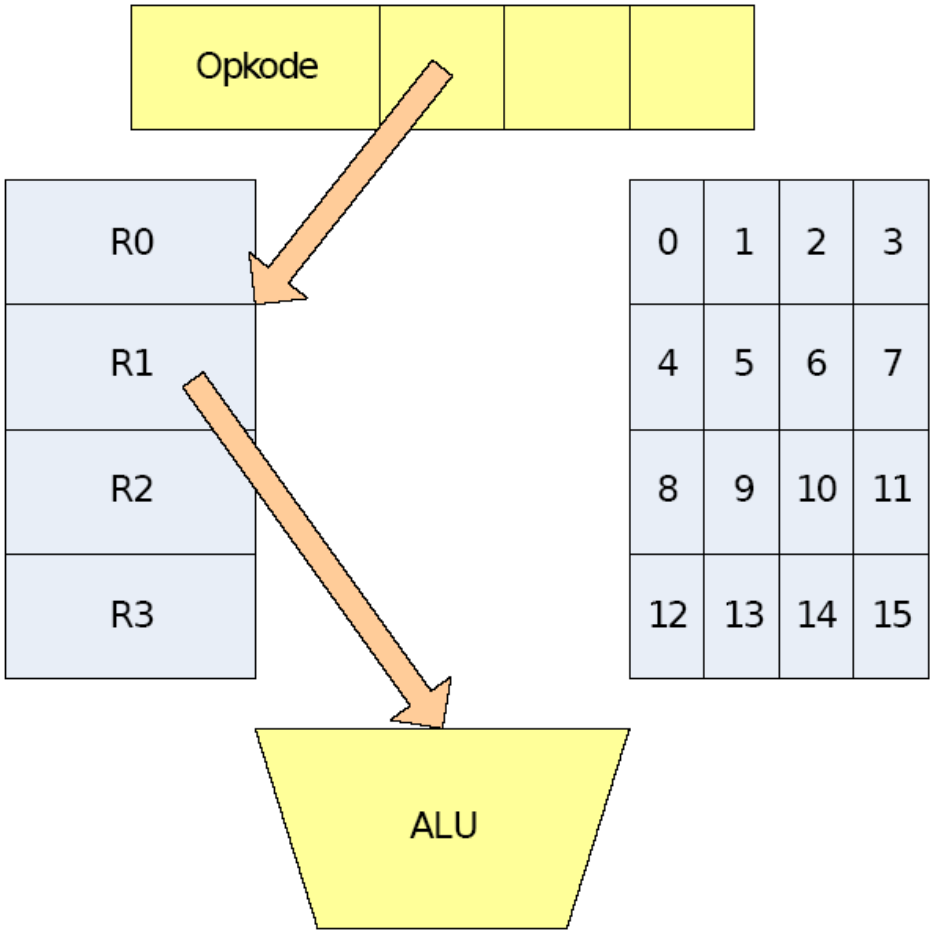
Direkte adressering

- Instruksjonsfelt inneholder hovedlageradresse
- Feltlengde begrenser adresseområde
- Adresse bestemt ved kompilering – lite fleksibelt
- Kan f.eks. bli brukt for globale variable



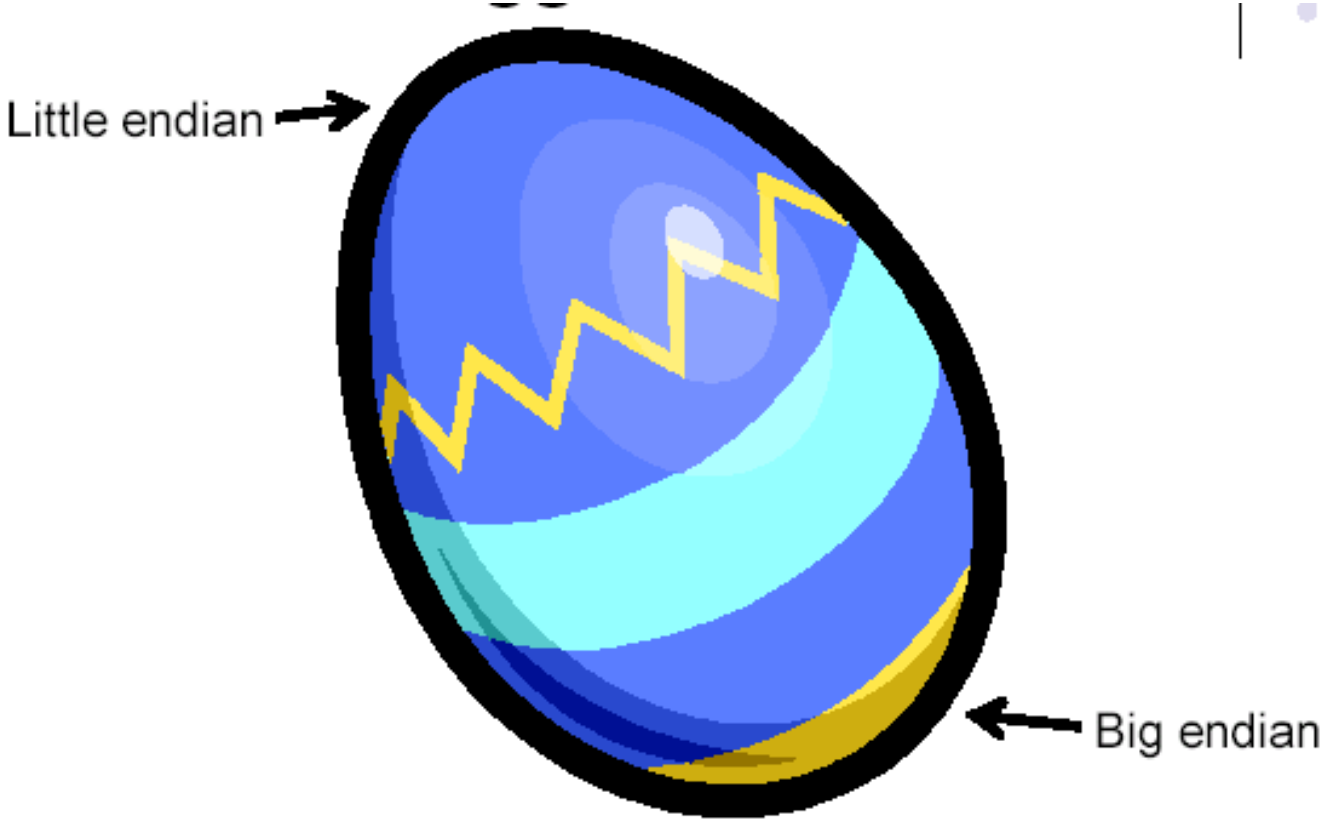
Registeradressering

- Instruksjonsfelt inneholder registernummer
- Veldig mye brukt
- RISC-ark. bruker nesten bare registeradressering
- Feltlengde begrenser antall registre



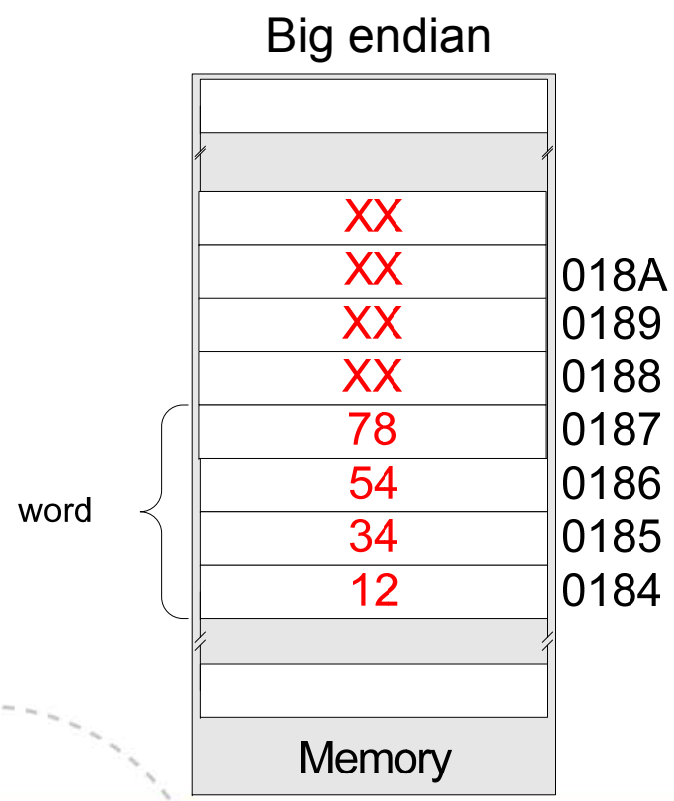
Dataorganisering i minne

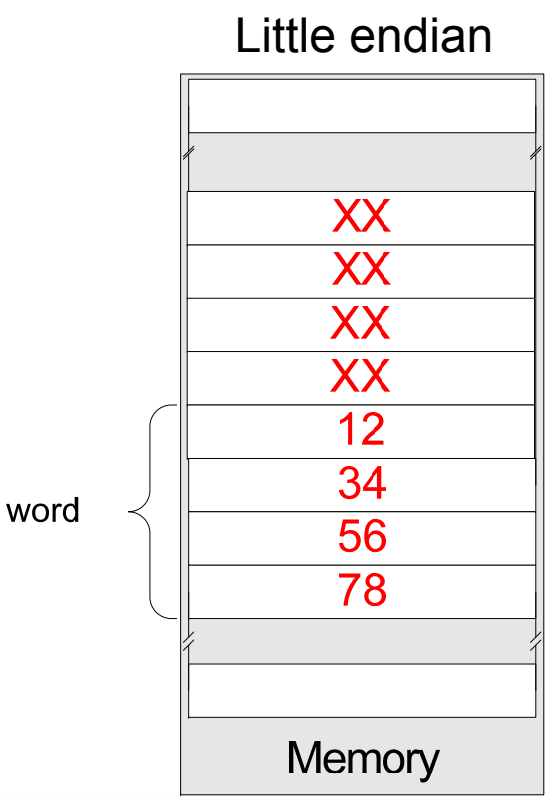
- Kvar skal kuttar ein egget

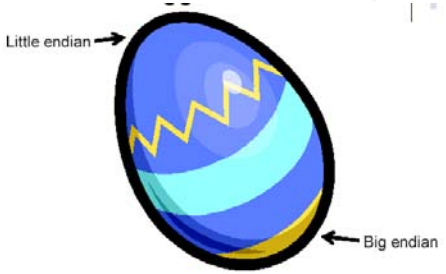


Dataorganisering: Byte-rekkefølge

- Ord på meir enn en byte kan lagrast på to måtar...
- Eksempel: **0x12 34 56 78** (32 bits/4 bytes hex)
- Viktig: Adressa til *ordet* er uendra!



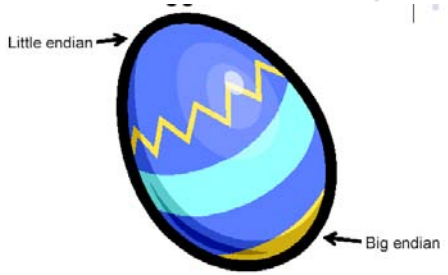




NTNU
Innovation and Creativity

Dataorganisering: Byte-rekkefølge

Address	Big endian				Little endian				Address
0	0	1	2	3	3	2	1	0	0
4	4	5	6	7	7	6	5	4	4
8	8	9	10	11	11	10	9	8	8
12	12	13	14	15	15	14	13	12	12
				Byte				Byte	
				32-bit word				32-bit word	



	Big endian				Little endian				
0	J	I	M			M	I	J	0
4	S	M	I	T	T	I	M	S	4
8	H	0	0	0	0	0	0	H	8
12	0	0	0	21	0	0	0	21	12
16	0	0	1	4	0	0	1	4	16

NTNU
Innovation and Creativity