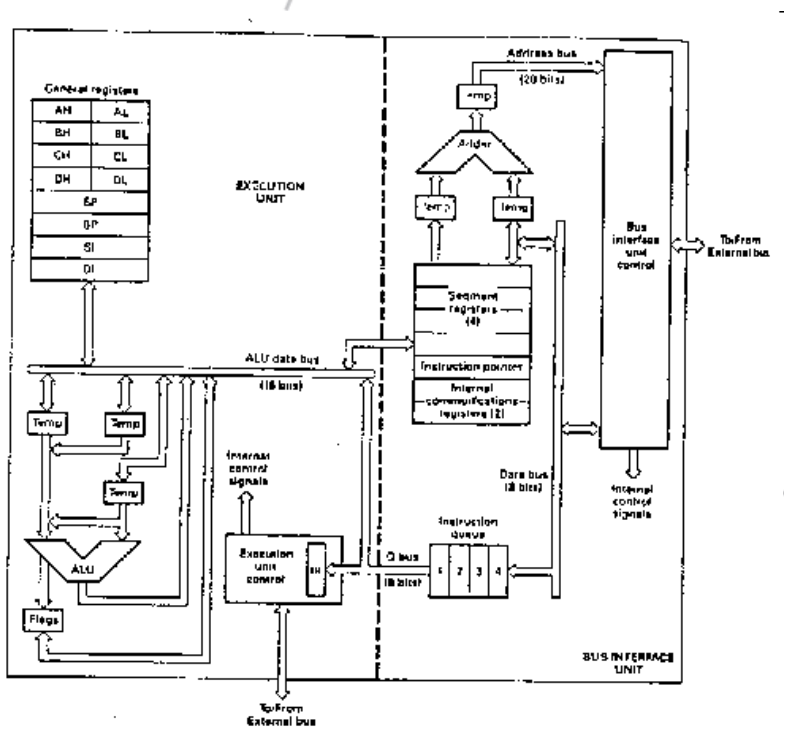
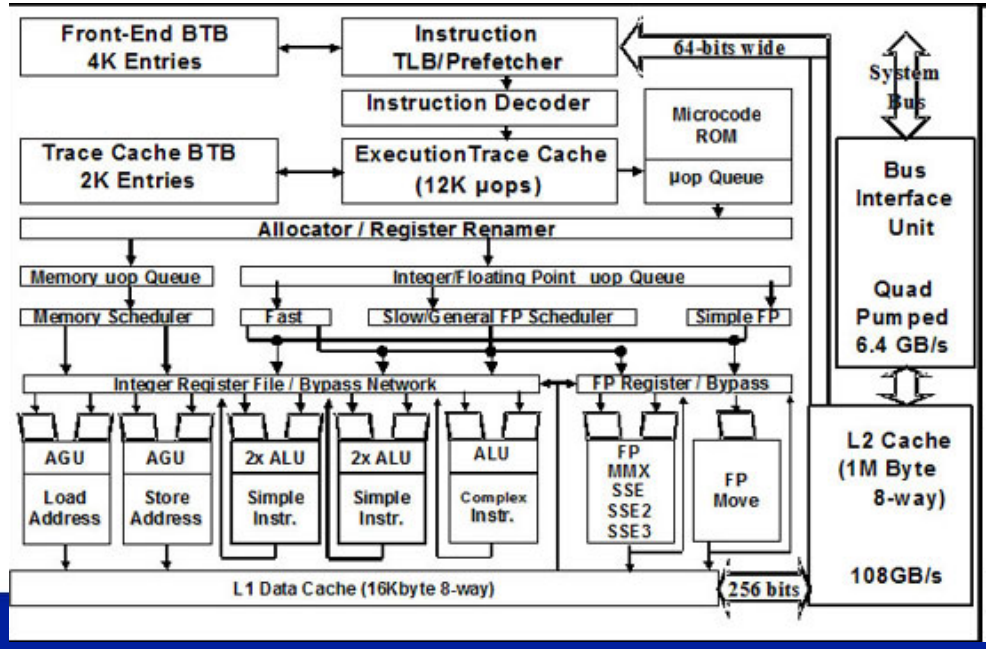
Fortsetelse Microarchitecture level

IJVM

- Implementasjon
  - Detaljar for å utføre instruksjonssettet
  - Ein gitt implementasjon har ein gitt yting
  - Endre ytinga
    - Teknologi (prosess)
    - Transistor implementasjon
    - Digital design av komponentar
    - Auke mengda logikk
    - Endring Av arkitektur
- Kva kan gjerast for å auke ytinga
  - Auke mengda logikk og endring av arkitektur





IJVM som eksempel

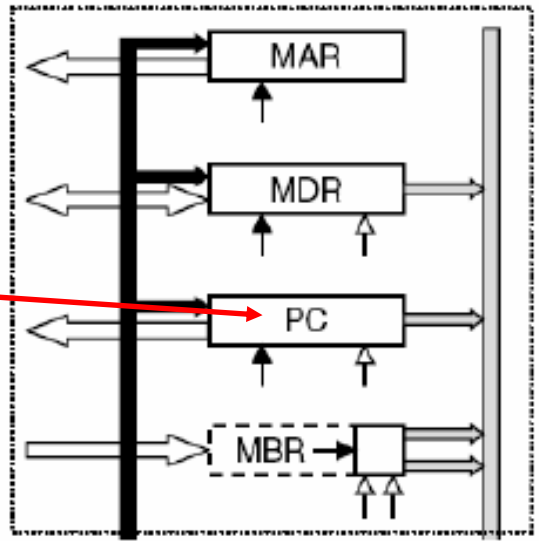
- Kva kan gjerast for å auke ytinga
  - Auke mengda logikk og endring av arkitektur
    - Instruction Fetch Unit
    - Samleband
    - Instruksjonskø
    - Hurtigbuffer

Flytte funksjonalitet til maskinvare:

- 1 ILOAD
- 2 PC = PC + 1
- 3 IADD
- 4 PC = PC + 1
- 5 ISTORE
- 6 PC = PC + 1
- 7 ILOAD
- 8 PC = PC + 1
- 9 BIPUSH
- 10 PC = PC + 1
- 11 IF_ICMPEQ (betinga hopp) **L1**
- 12 PC = Opdater PC
- 13 ILOAD
- 14 PC = PC + 1
- 15 BIPUSH
- 16 PC = PC + 1
- 17 ISUB
- 18 PC = PC + 1
- 19 ISTORE
- 20 PC = PC + 1
- 21 GOTO **L2**
- 22 PC = **L2**
- 23 BIPUSH **L1**
- 24 PC = PC + 1
- 25 ISORE
- 26 PC = PC + 1
- 27 XXXXX **L2**

- PC = PC +1
  - **ALU**
  - **B-Buss**
  - **C-Buss**
- Okuperar ressursar
  - Legg til logikk
    - PC = PC + 1

•PC = PC +1
•Hopp

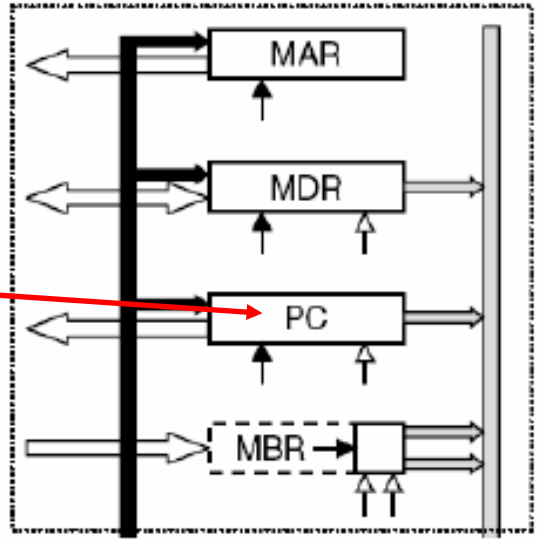


Flytte funksjonalitet til maskinvare:

- 1 ILOAD
- 2 IADD
- 3 ISTORE
- 4 ILOAD
- 5 BIPUSH
- 6 IF_ICMPEQ (betinga hopp) **L1**
- 7 PC = Opdater PC
- 8 ILOAD
- 9 BIPUSH
- 10 ISUB
- 11 ISTORE
- 12 GOTO **L2**
- 13 PC = **L2**
- 14 BIPUSH **L1**
- 15 ISORE
- 16 XXXXX **L2**

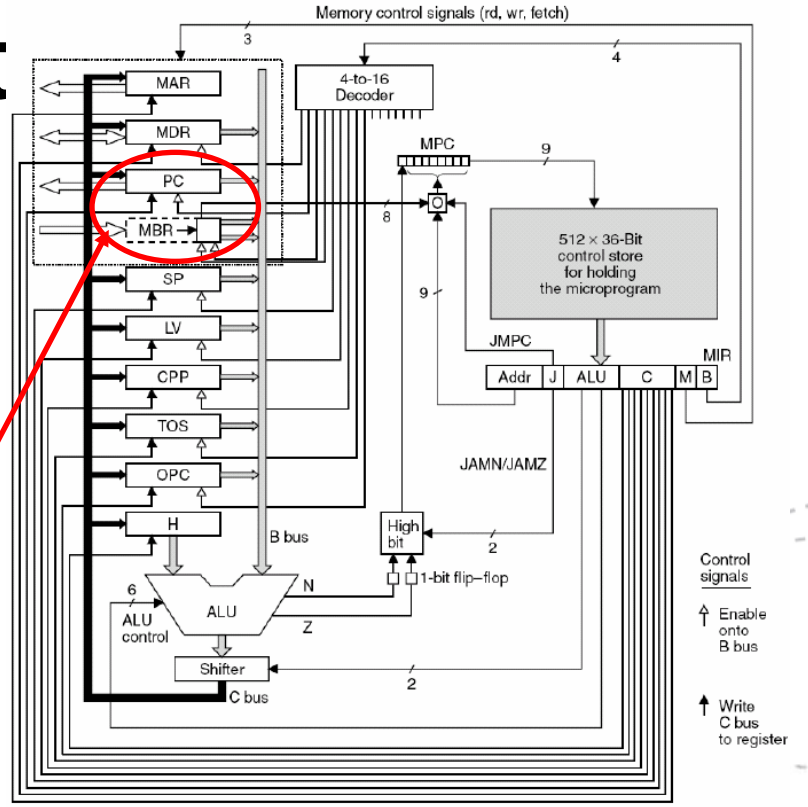
- PC = PC +1
  - **ALU**
  - **B-Buss**
  - **C-Buss**
- Okuperar ressursar
  - Legg til logikk
    - PC = PC + 1
  - Redusert med 11 gonger bruk av
    - ALU
    - B-Buss
    - C-Buss

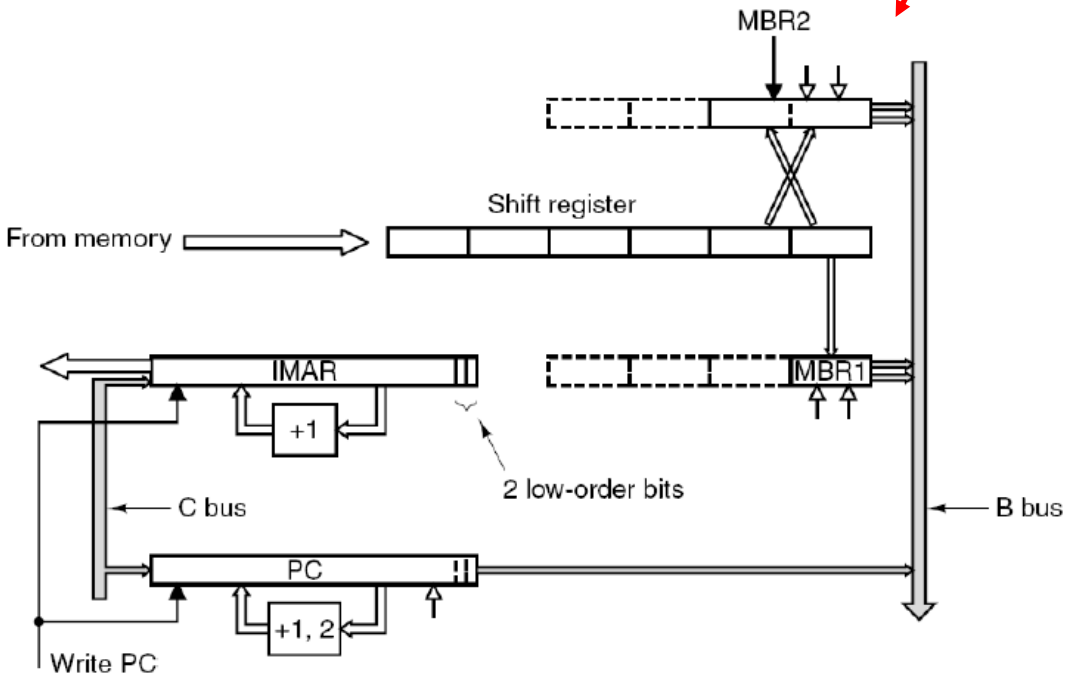
•PC = PC +1
•Hopp



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

Instruction Fetch Unit (IFU)

- For kvar instruksjon
  - PC increment (ALU)
  - PC peikar på neste instruksjon
  - Operandar lest frå minne
  - Operandar skrives til minne
  - ALU operasjon, resultat lagra (ein eller anna plass

- Viss:
  - Instruksjonar med ekstra operandar
    - Kvar operand må hentast (1 Byte viss i programminne) Brukar PC dvs ALU

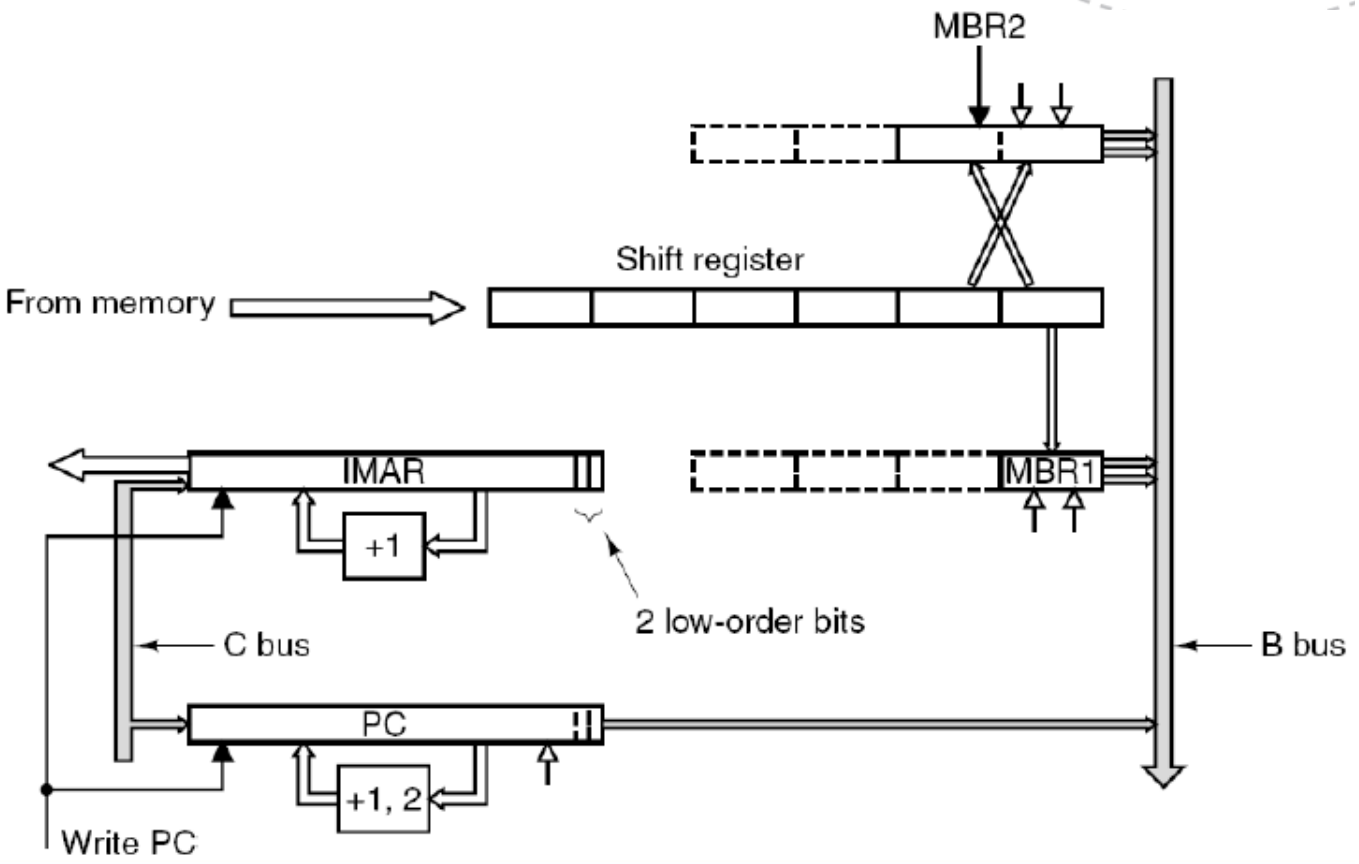
Instruction Fetch Unit (IFU)

- To moglege alternativ:
  - 1:
  - IFU tolkar opCode
  - IFU avgjer antal ekstra operandar (ut frå opCode)
  - Hentar operandar (fetch)
  - Skriv operandar til register
  - PC peikar på neste instruksjon
  - Operandar lest frå minne
  - Operandar skrives til minne
  - ALU operasjon, resultat lagra (ein eller anna plass

  - 2:
  - Gjere instruksjonar kontinuerleg tilgjengelege for styreeining
  - Kø med instruksjonar som skal utførast (8 og 16 bit)

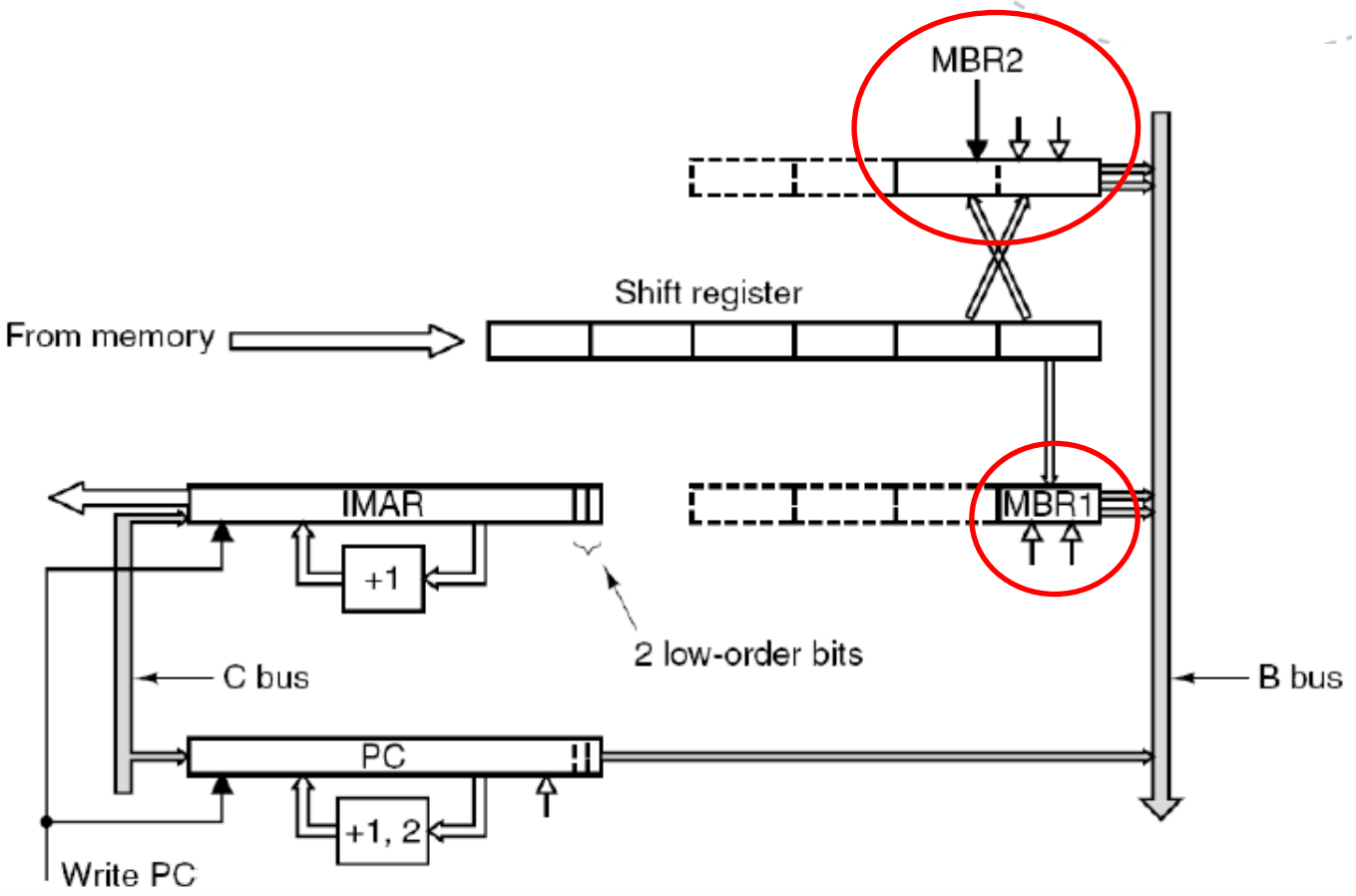
Instruction Fetch Unit

- 2:
  - Gjere instruksjonar kontinuerleg tilgjengelege for styreeining
  - Kø med instruksjonar som skal utførast (8 og 16 bit)

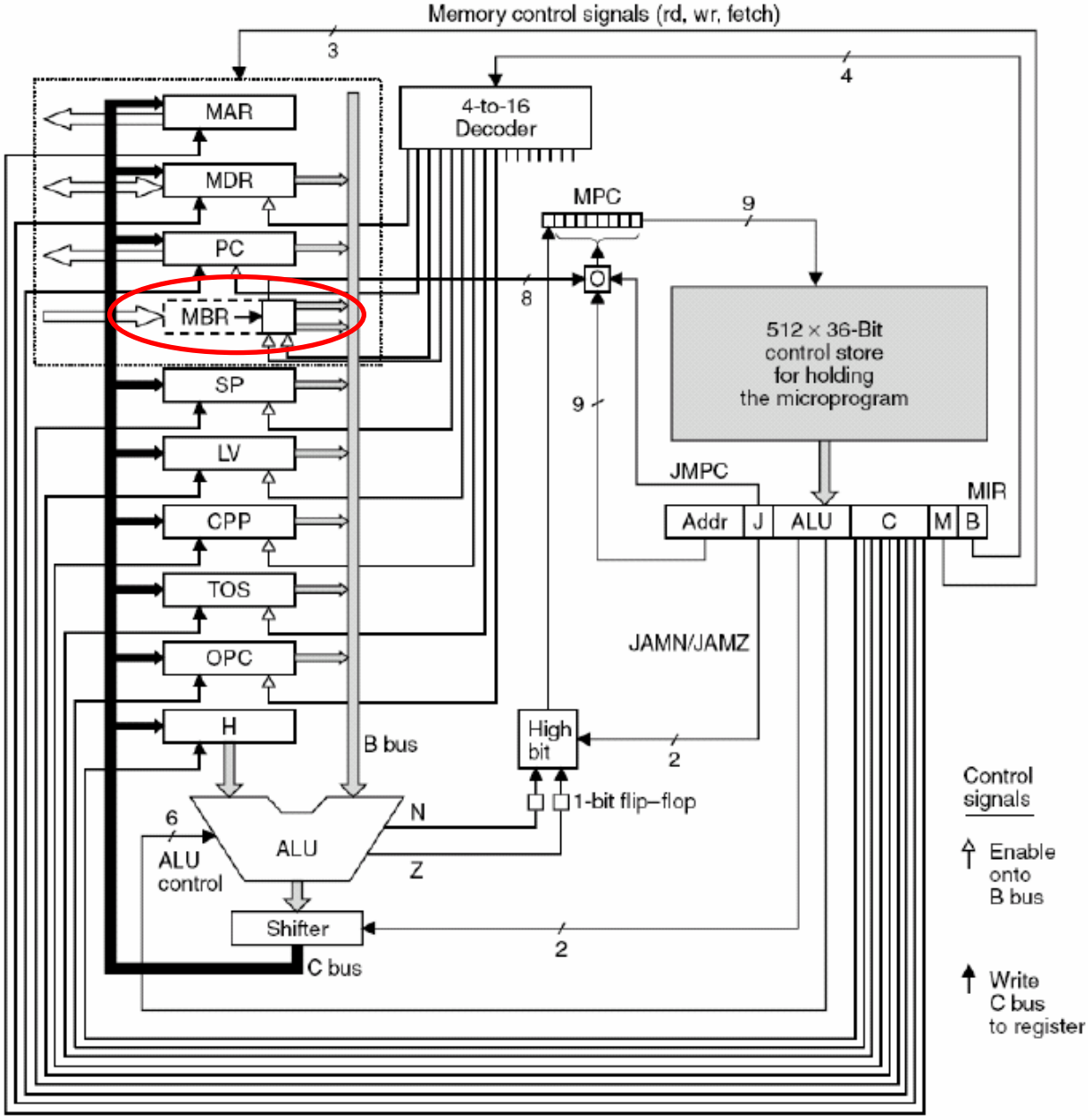


Instruction Fetch Unit

- Innfører MBR2
  - MBR1: 8 bit
  - MBR2: 16 bit



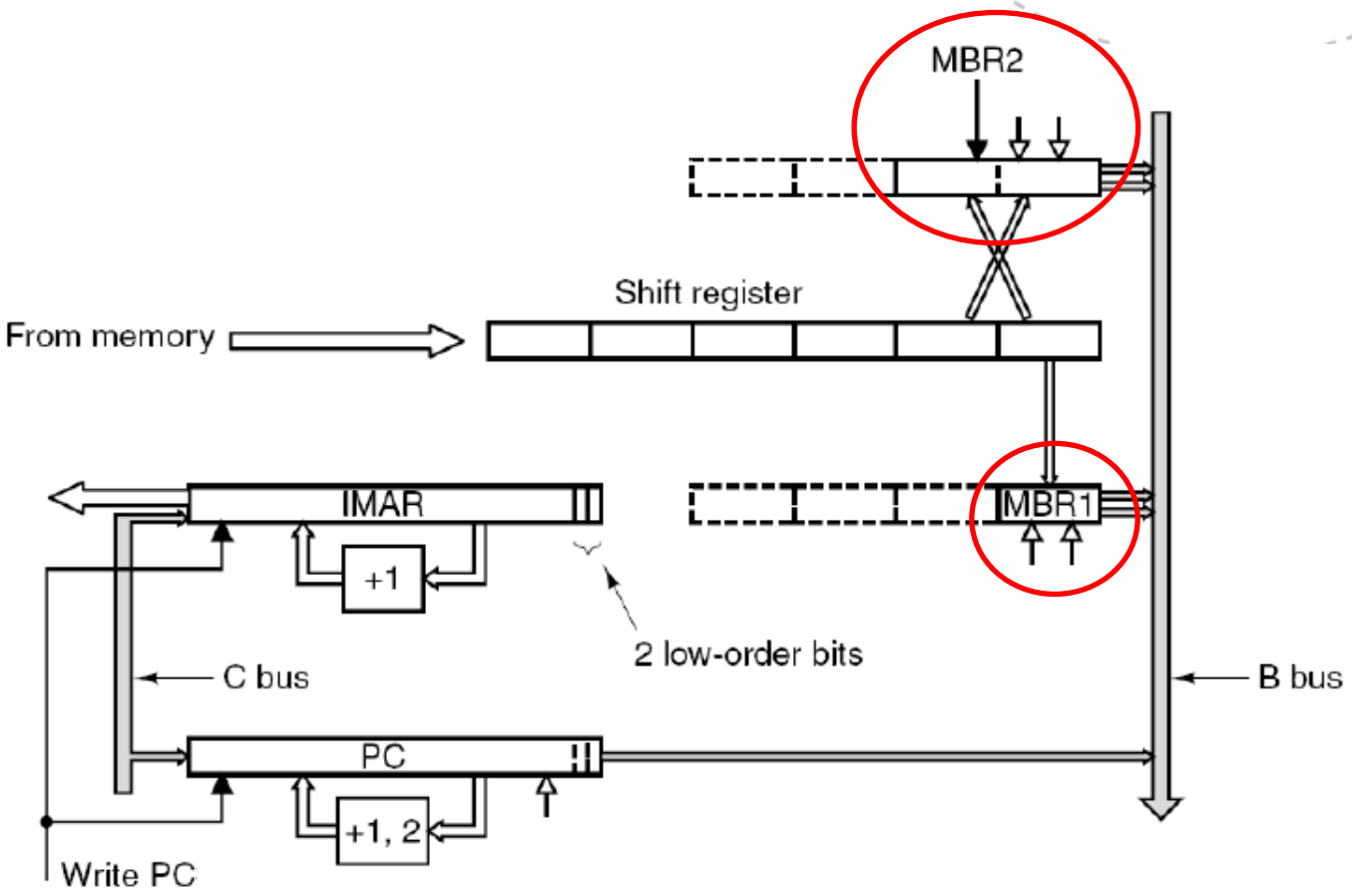
Instruction Fetch Unit



NTNU
Innovation and Creativity

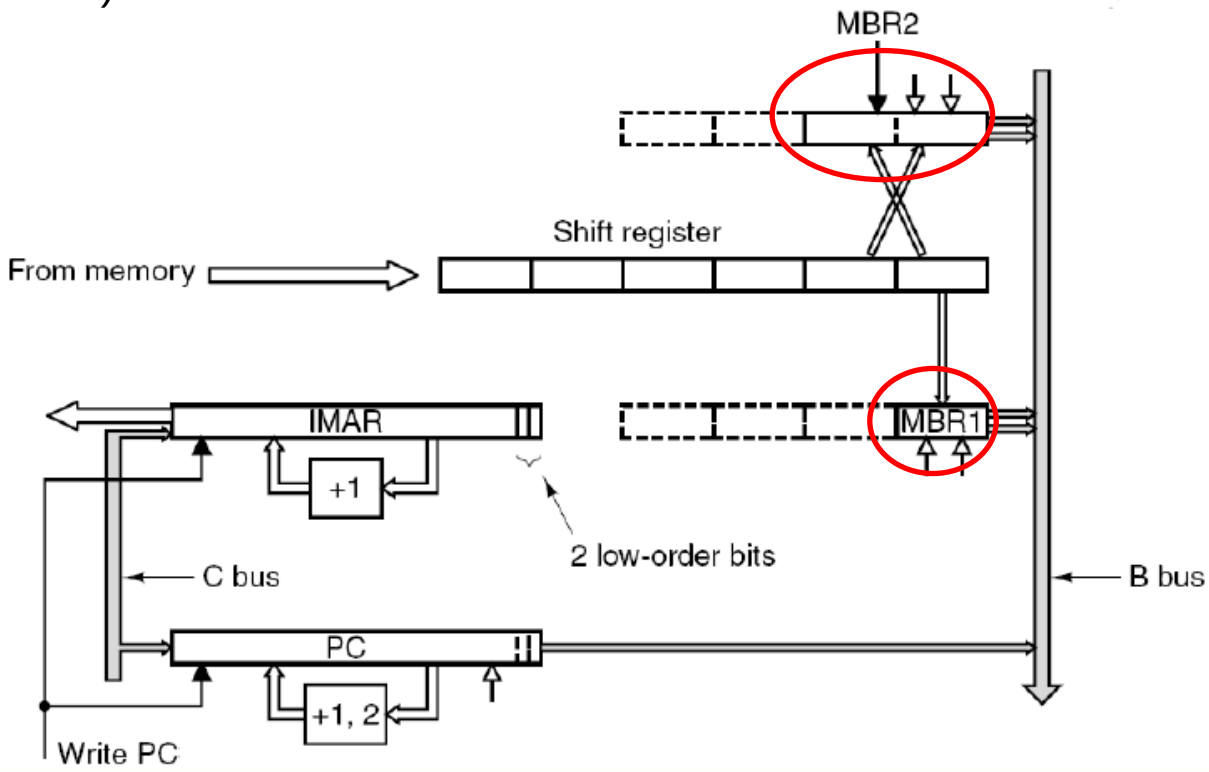
Instruction Fetch Unit

- Innfører MBR2
  - MBR1: 8 bit
  - MBR2: 16 bit



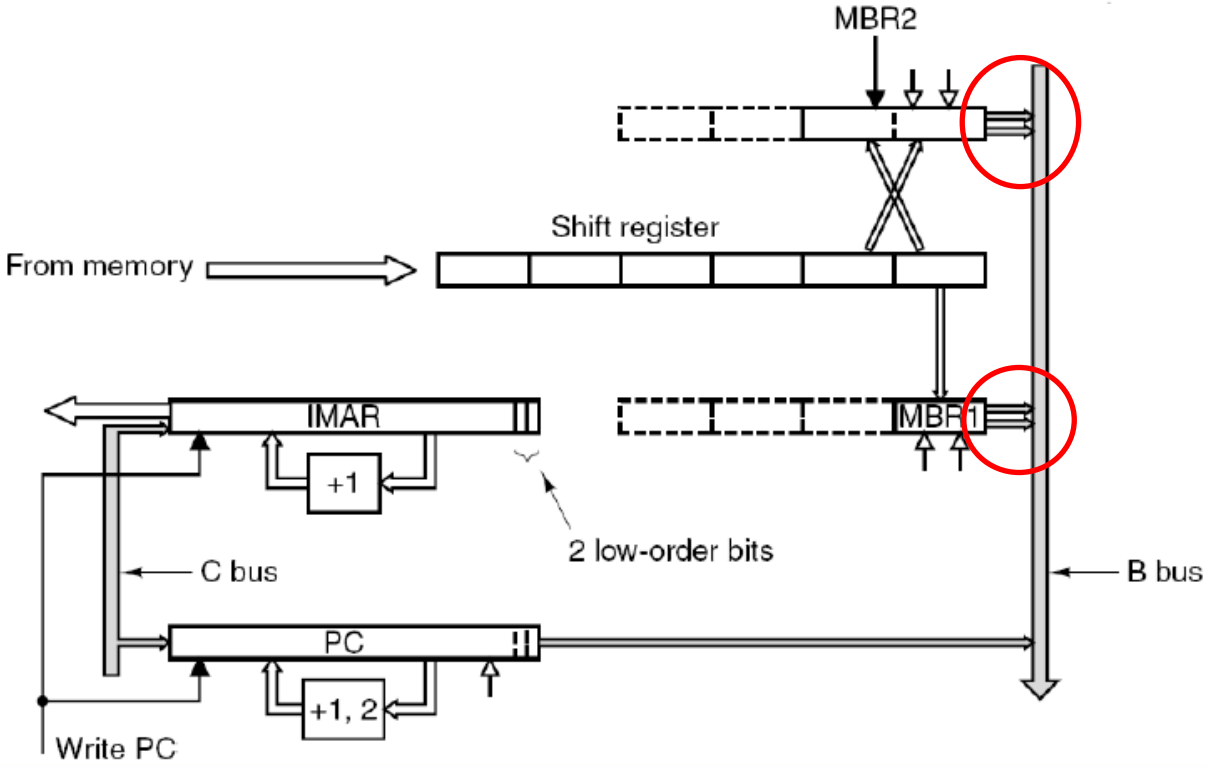
Instruction Fetch Unit

- Shift register
  - Kø av instruksjonar
  - MBR1: 1 byte (først lest)
  - Når MBR1 lest, last MBR1 med neste instruksjon
  - MBR2: 2 byte (først leste)
  - Neste to instruksjonar (operandar)



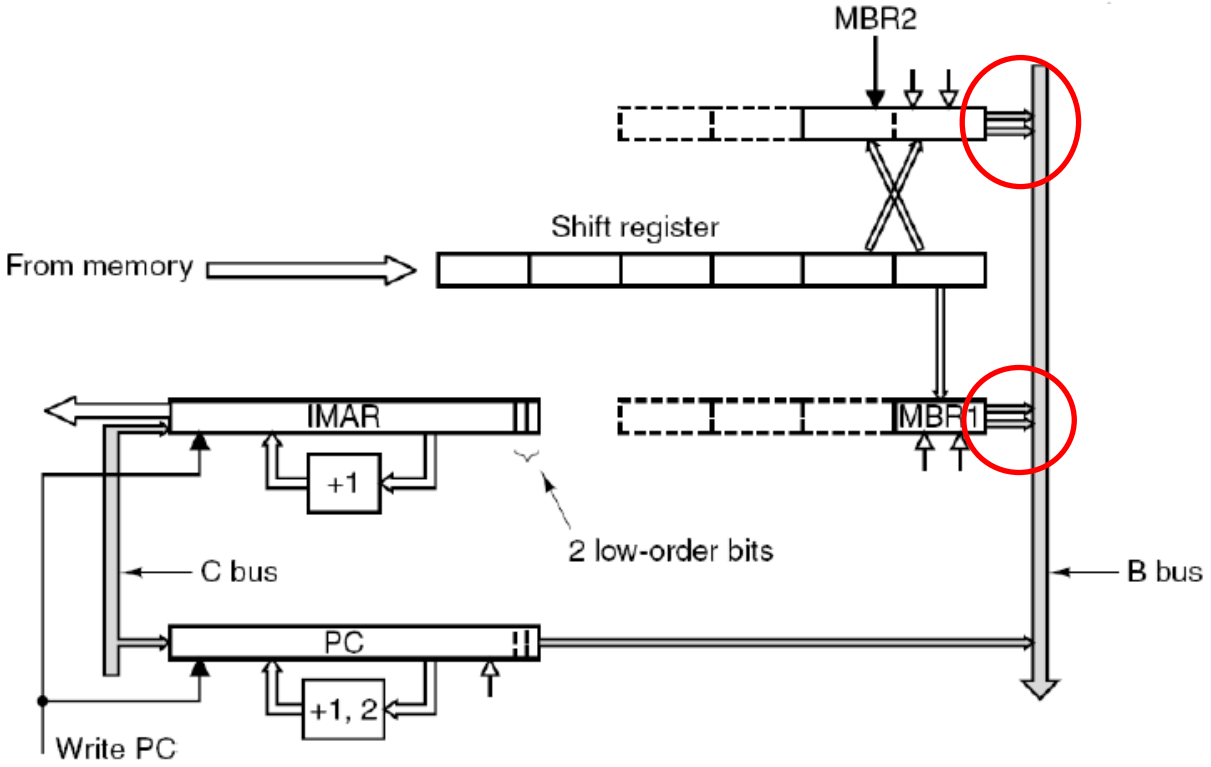
Instruction Fetch Unit

- Shift register
  - Kø av instruksjonar
  - MBR1: 1 byte (først lest)
  - Når MBR1 lest, last MBR1 med neste instruksjon
  - MBR2: 2 byte (først leste)
  - Neste to instruksjonar (operandar)
  - Sign extention (16 bit verdiar)
    - Legge innhald i reg ut på B-buss (MSD eller LSD)



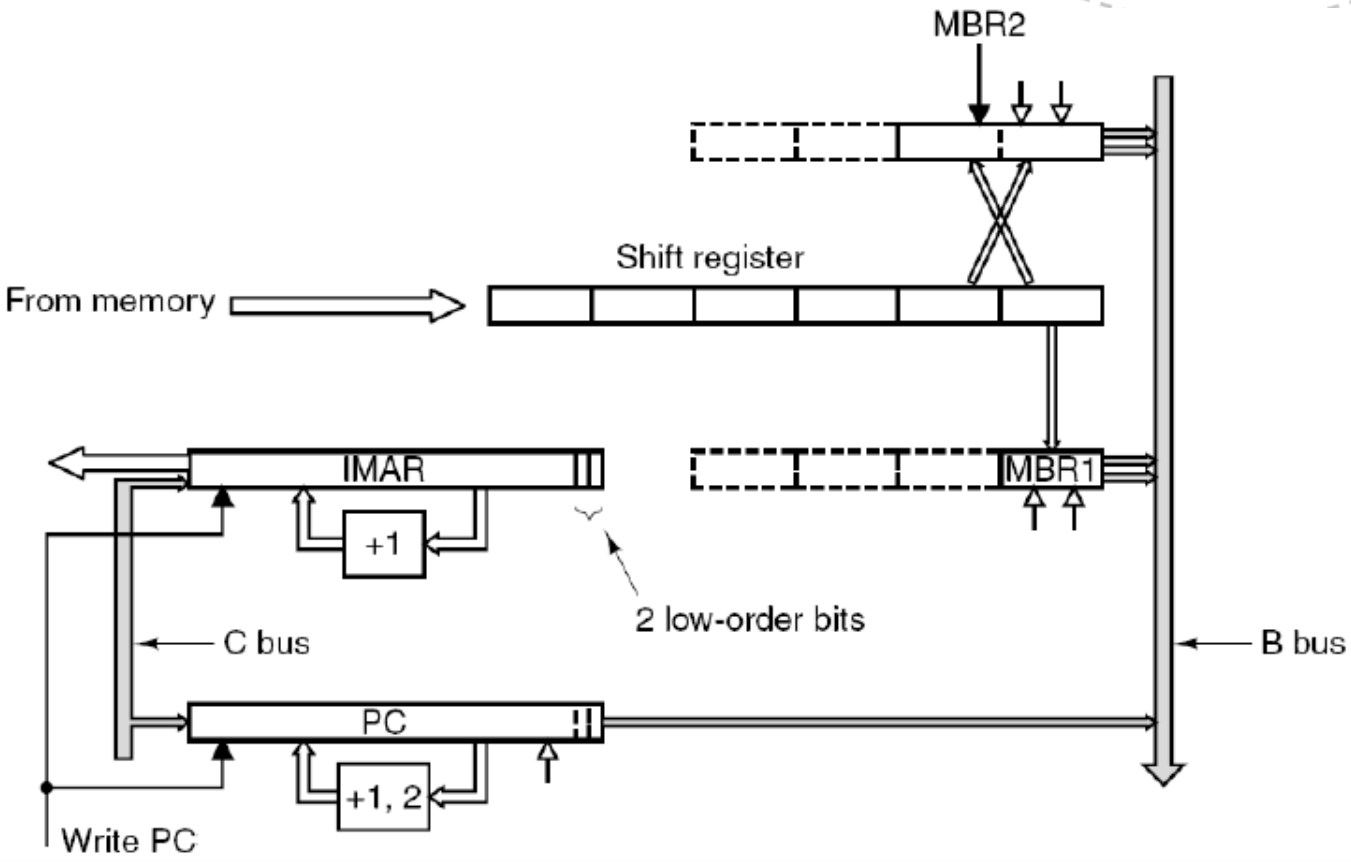
Instruction Fetch Unit

- Shift register
  - Kø av instruksjonar
  - MBR1: 1 byte (først lest)
  - Når MBR1 lest, last MBR1 med neste instruksjon
  - MBR2: 2 byte (først leste)
  - Neste to instruksjonar (operandar)
  - Sign extention (16 bit verdiar)
    - Legge innhald i reg ut på B-buss (MSD eller LSD)



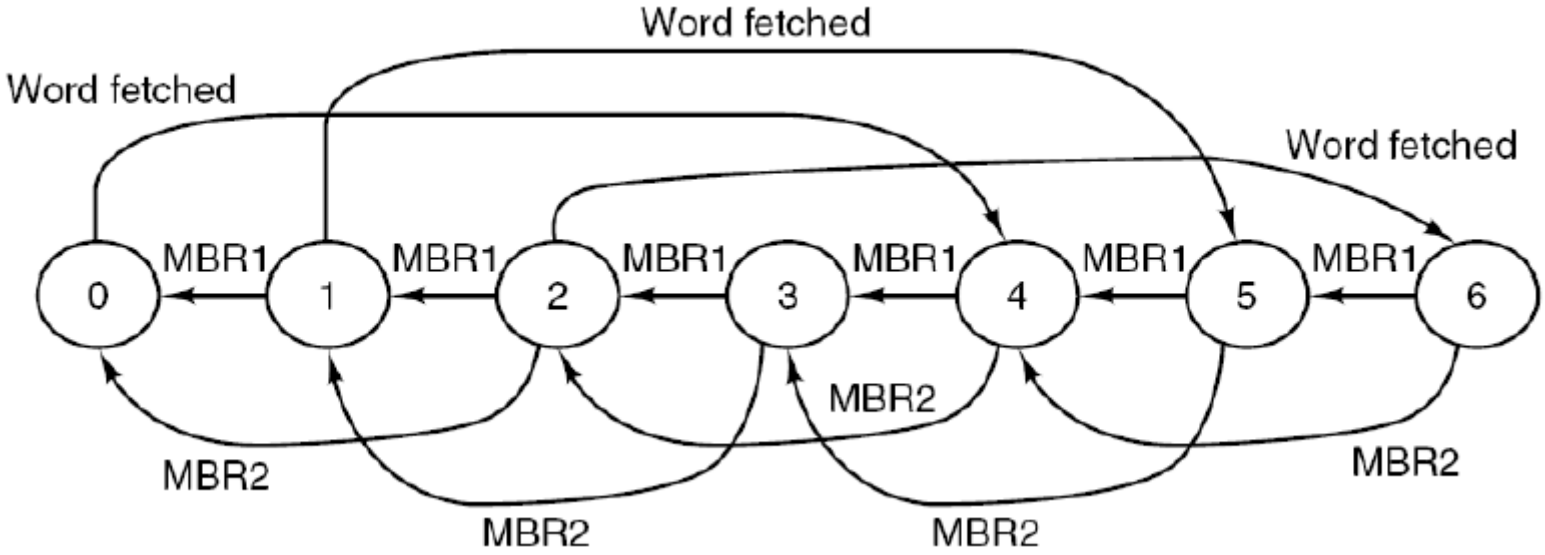
Instruction Fetch Unit

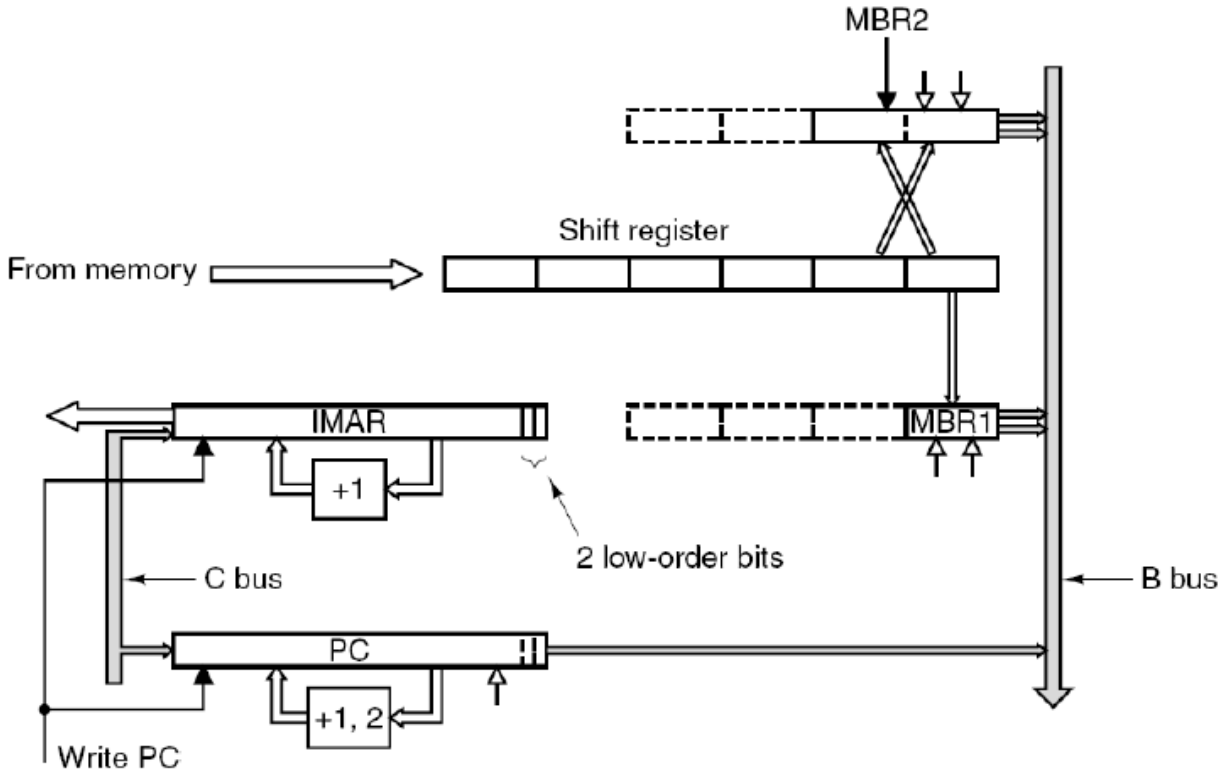
- IMAR
  - Innført eige register for adressering av programminne
    - MAR ikkje okkupert til adressering av instruksjonar
    - Har lokal increment for sekvensiell utførelse
    - Lastast frå C-buss ved hopp
    - Kopi av PC

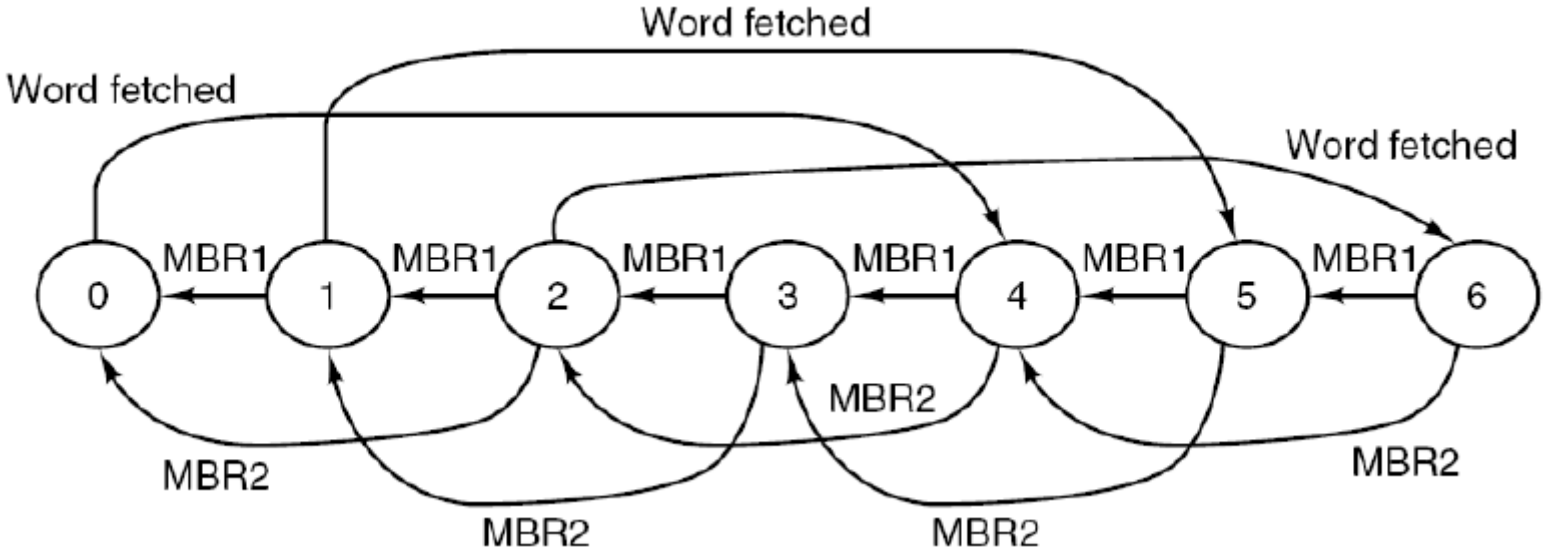


Instruction Fetch Unit FSM

- MBR1: når MBR1 er lest
- MBR2 når MBR2 er lest
- Word fetched når 4 byte er lest frå minne til shift registeret
- Tilstand namn gir antal byte i shift registeret




MBR2
Shift register
From memory
IMAR
MBR1
+1
2 low-order bits
C bus
B bus
PC
+1, 2
Write PC


Word fetched
Word fetched
Word fetched
MBR1
MBR1
MBR1
MBR1
MBR1
MBR1
0
1
2
3
4
5
6
MBR2
MBR2
MBR2
MBR2
MBR2

Instruction Fetch Unit (IFU)

- PC = PC + 1 i maskinvare
  - IFU gjer PC increment
- To Byte operandar tilgjengeleg
- IMAR register mot minne

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

Utdrag Gunnars Uopdaterte plan

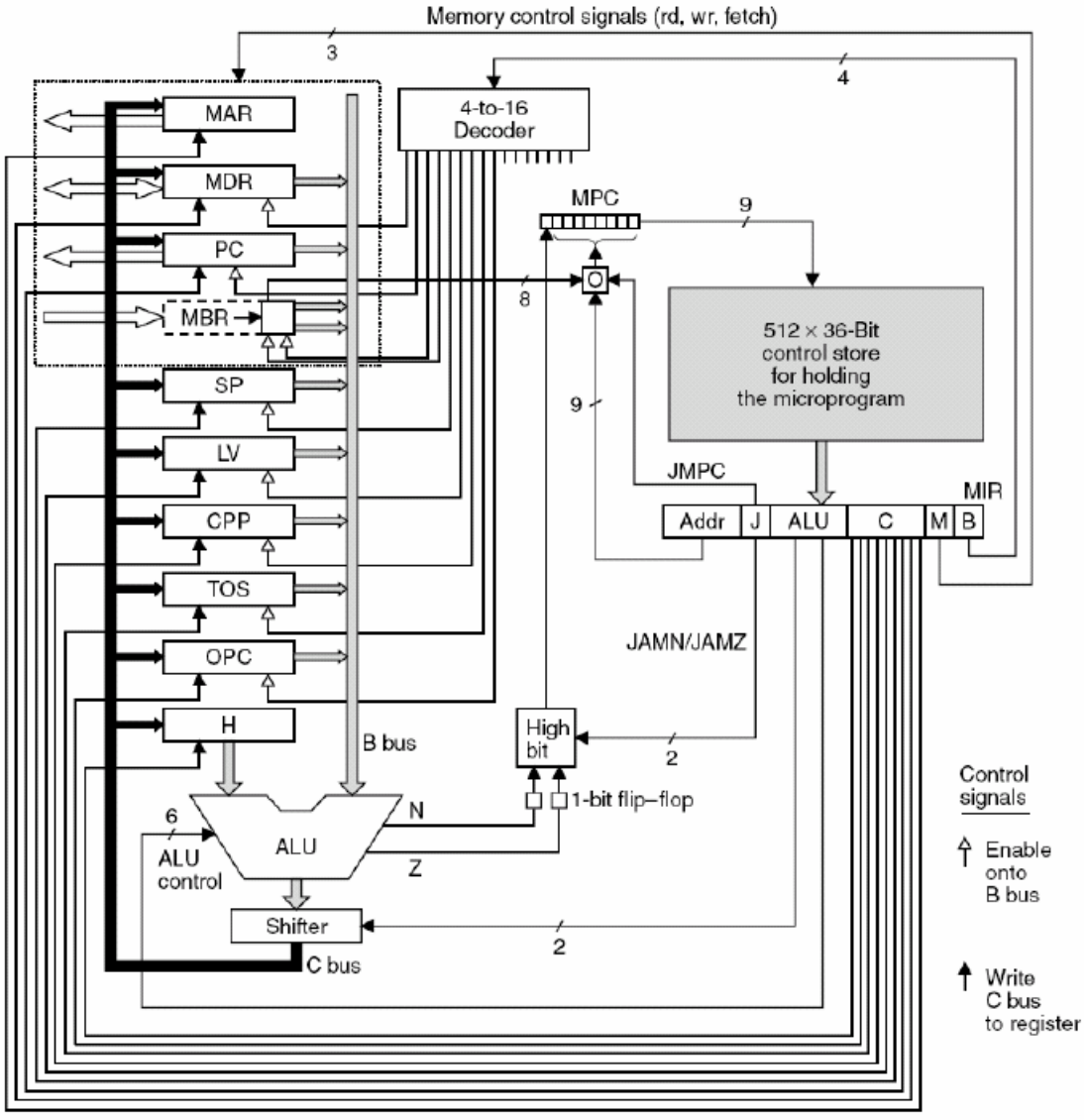
- 19/11 Spørje/lure på, der studentane (dykk) må finne ut om det er noko som er uklart. Eg må ha melding om kva fredag 14/11
- 20/11 Oppsummering

IJVM som eksempel

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

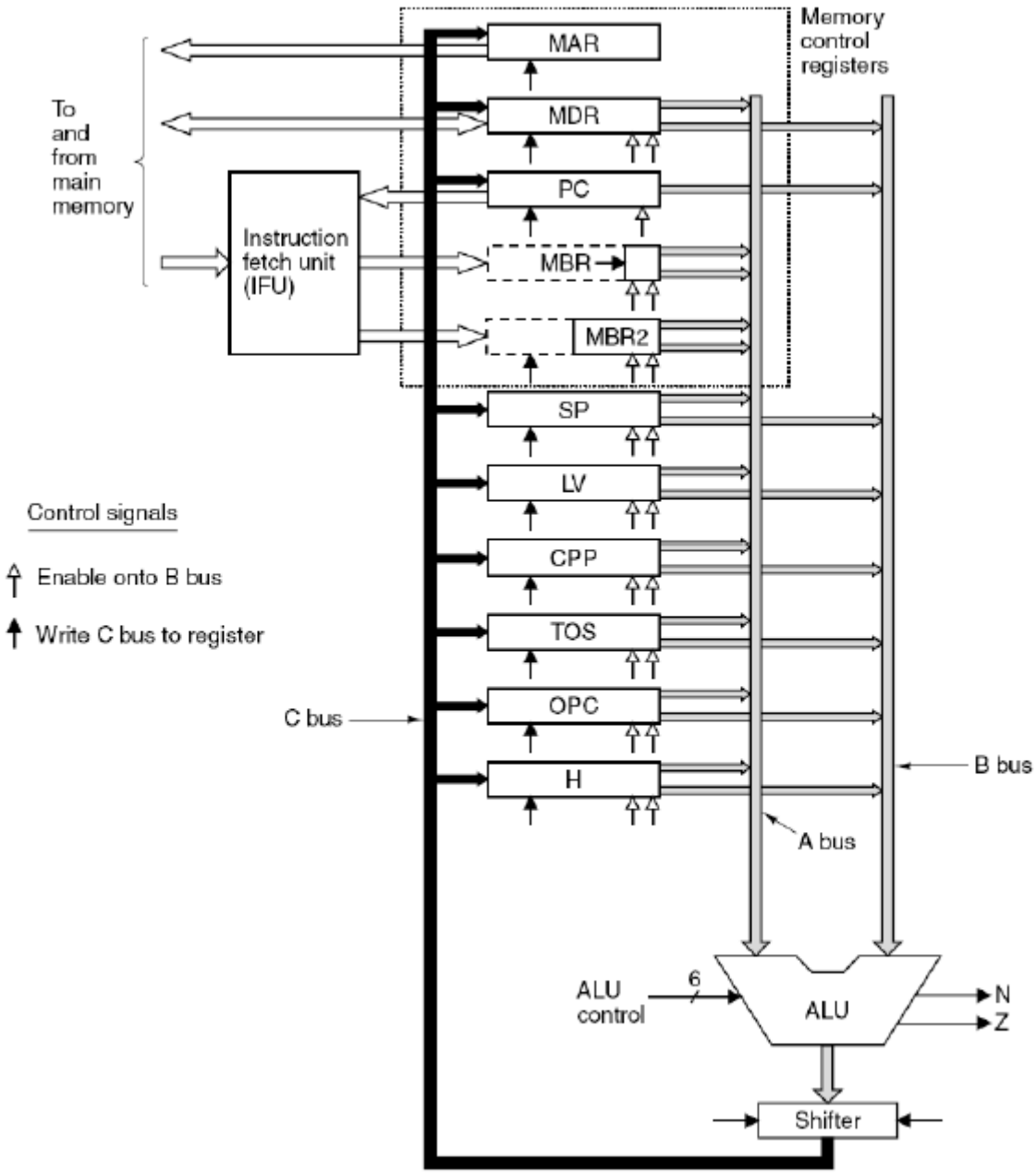
IJVM Innfører A-buss

- 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



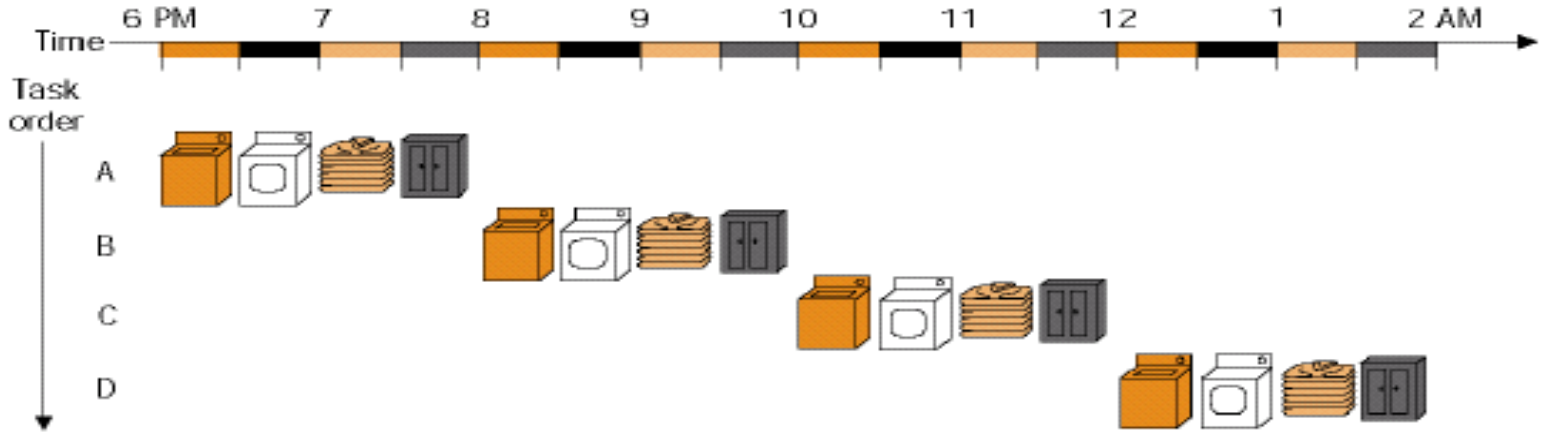
IJVM som eksempel

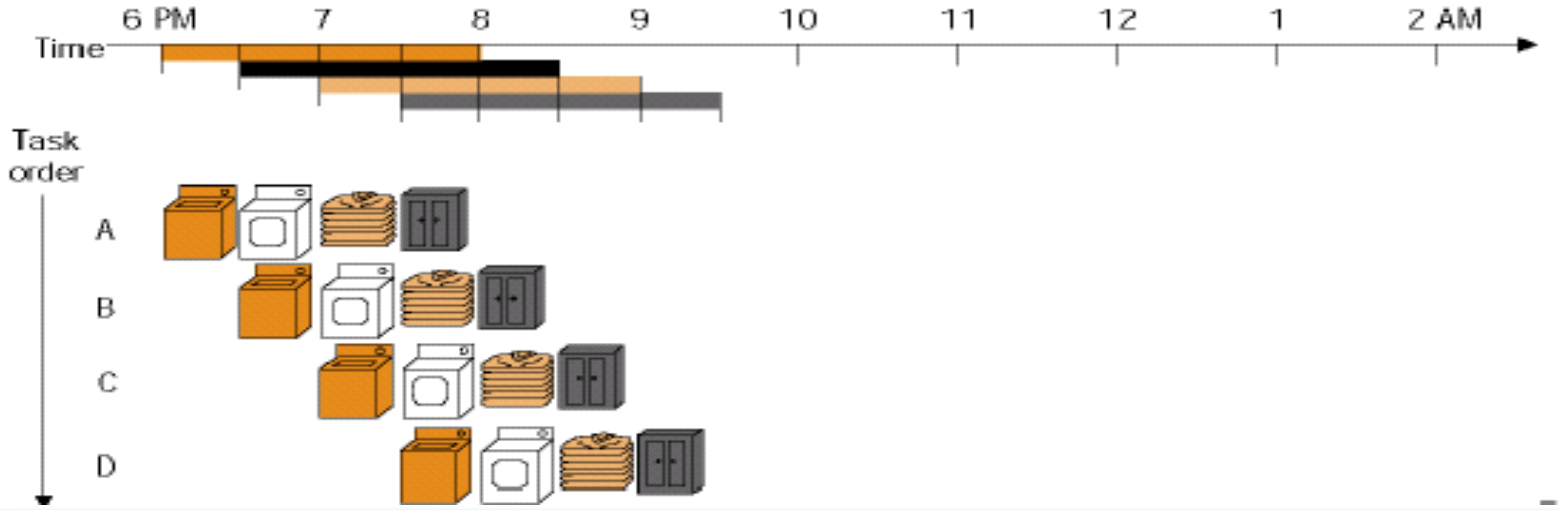
- Kva kan gjerast for å auke ytinga
  - Auke mengda logikk og endring av arkitektur
    - Instruction Fetch Unit
    - **Samleband**
    - Instruksjonskø
    - Hurtigbuffer

Parallellitet

- Essensielt for å øke ytelse
- To typer:
  - 1) Instruksjonsnivåparallellitet
    - Fleire instruksjonar utføres samtidig (1 prosessor)
    - Samleband
      - Eks: Unødvendig at ALU er ledig mens CPU leser instruksjon
    - Superskalaritet: Duplisering av CPU-komponenter
  - 2) Prosessornivåparallellitet
    - Ein prosessor er bra, fleire er betre?

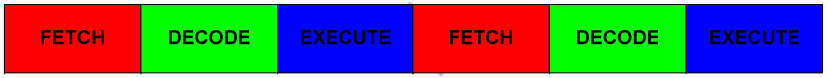
Samleband (Pipelining)

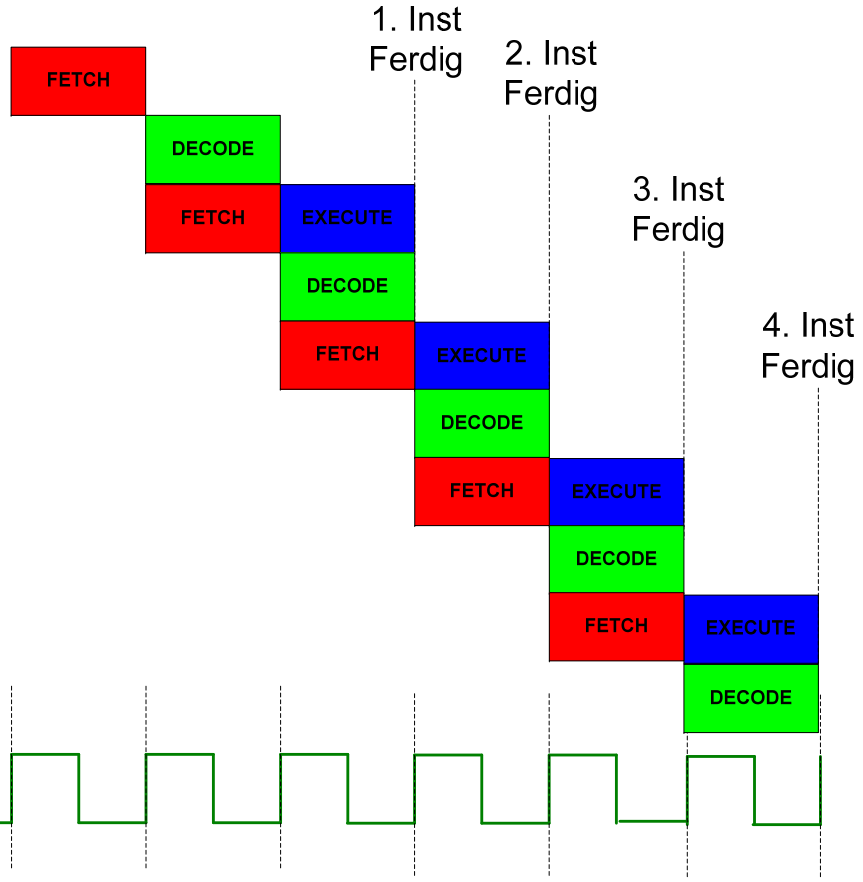




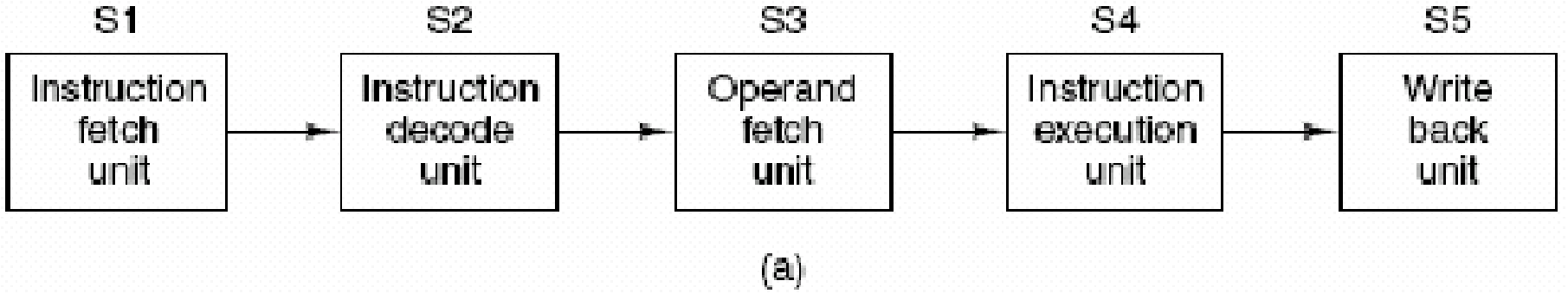
Samleband (Pipelining)

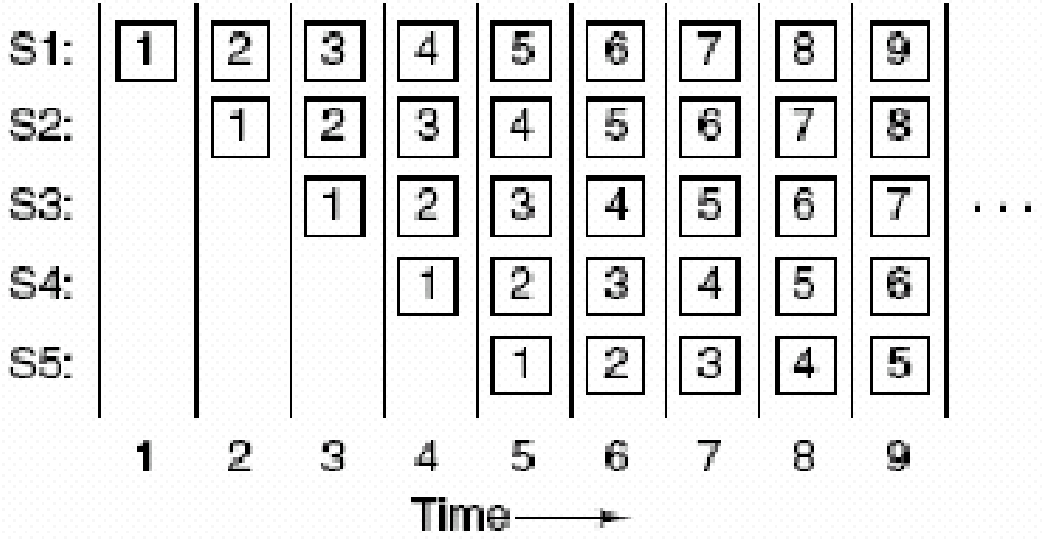
- **Fetch Decode Execute**
- Dei tre prinsipielle stega i instruksjonutføring
- Lagar oss ein tretrinns styreeining
  - 1: Fetch
  - 2: Decode
  - 3: Execute
  - Kan startast uavhengig av kvarandre
- Raskare
  - Enkle trinn
  - Kortare tid



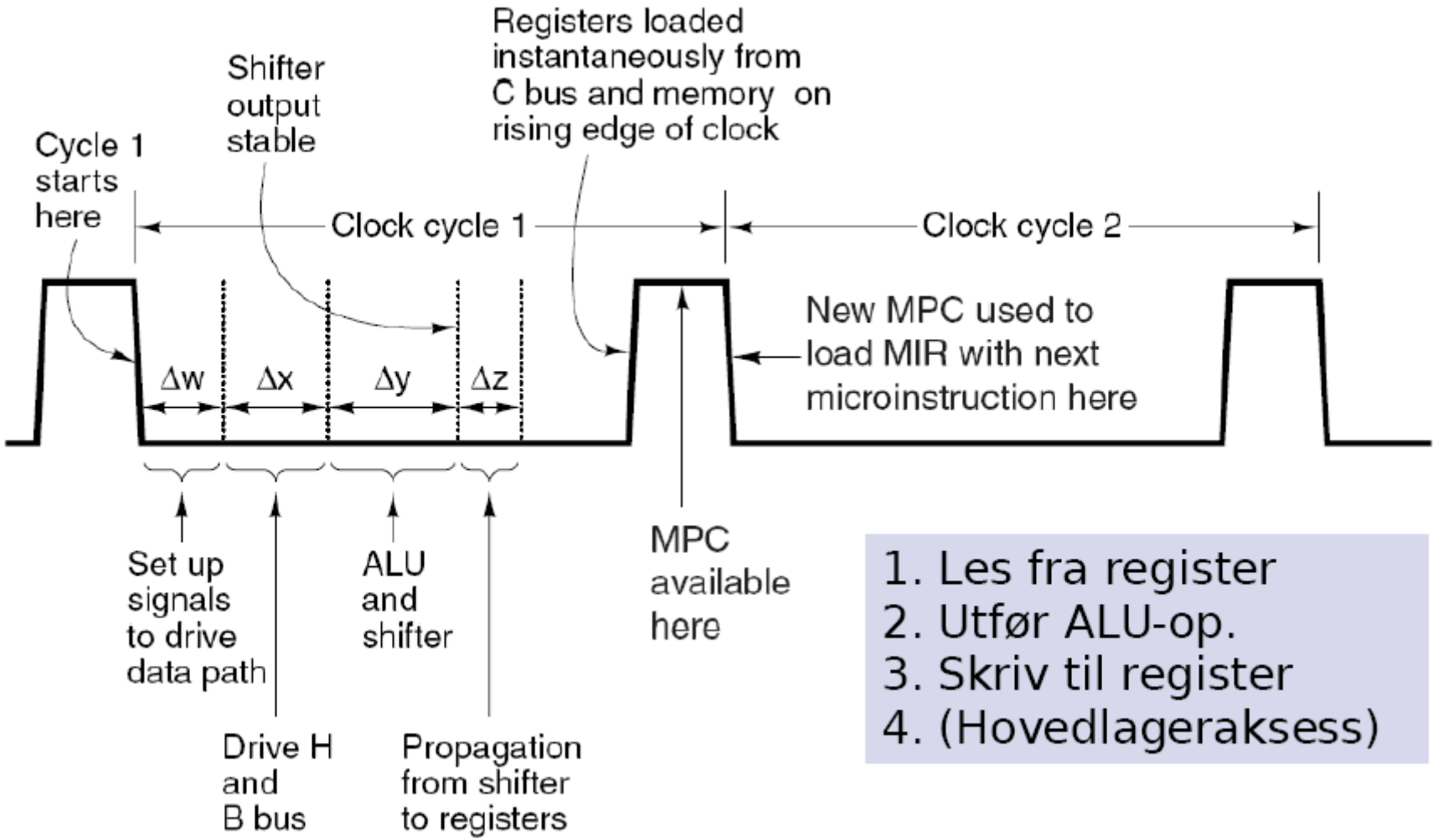


Samleband (Pipelining)



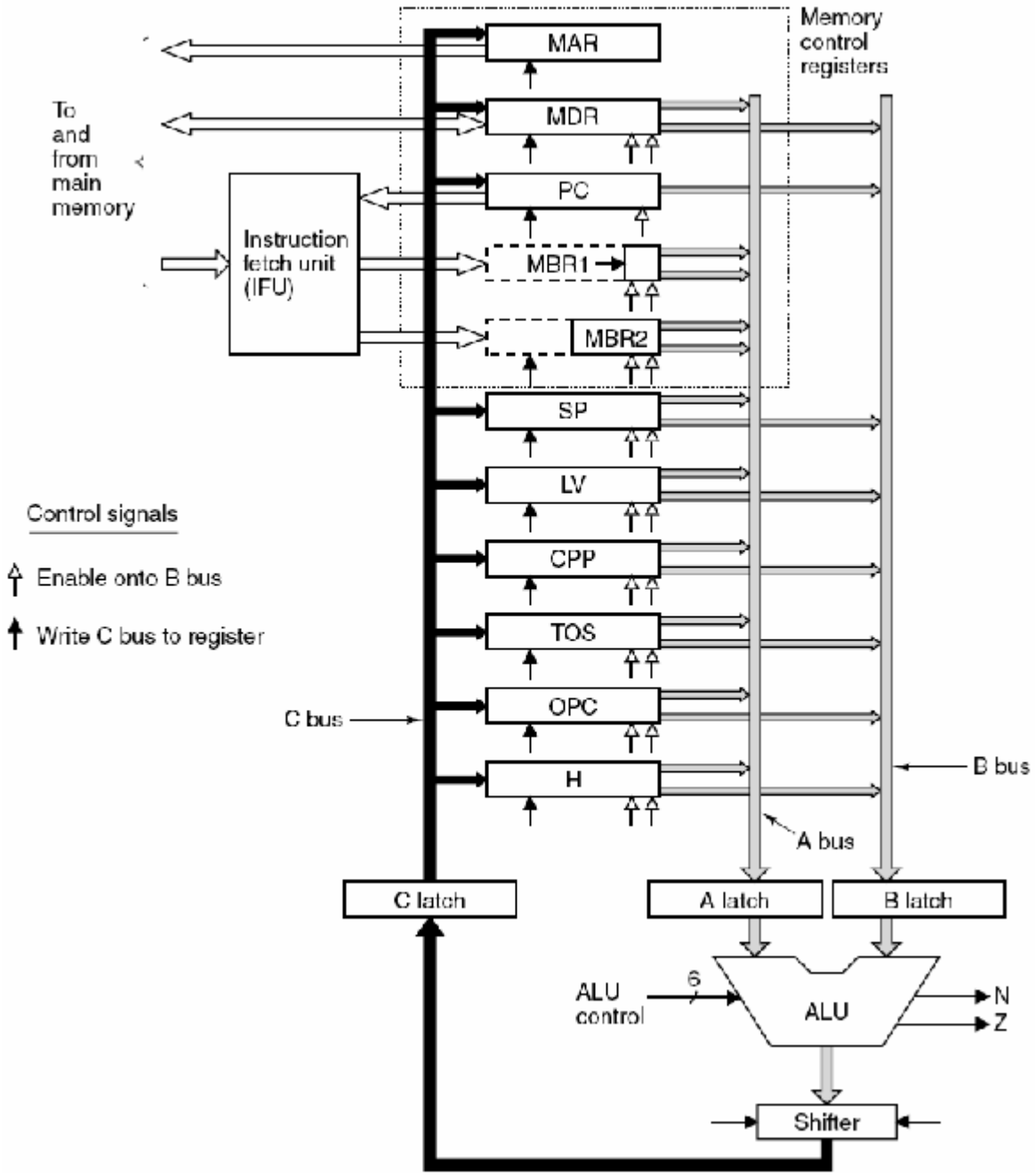


Samleband (Pipelining)

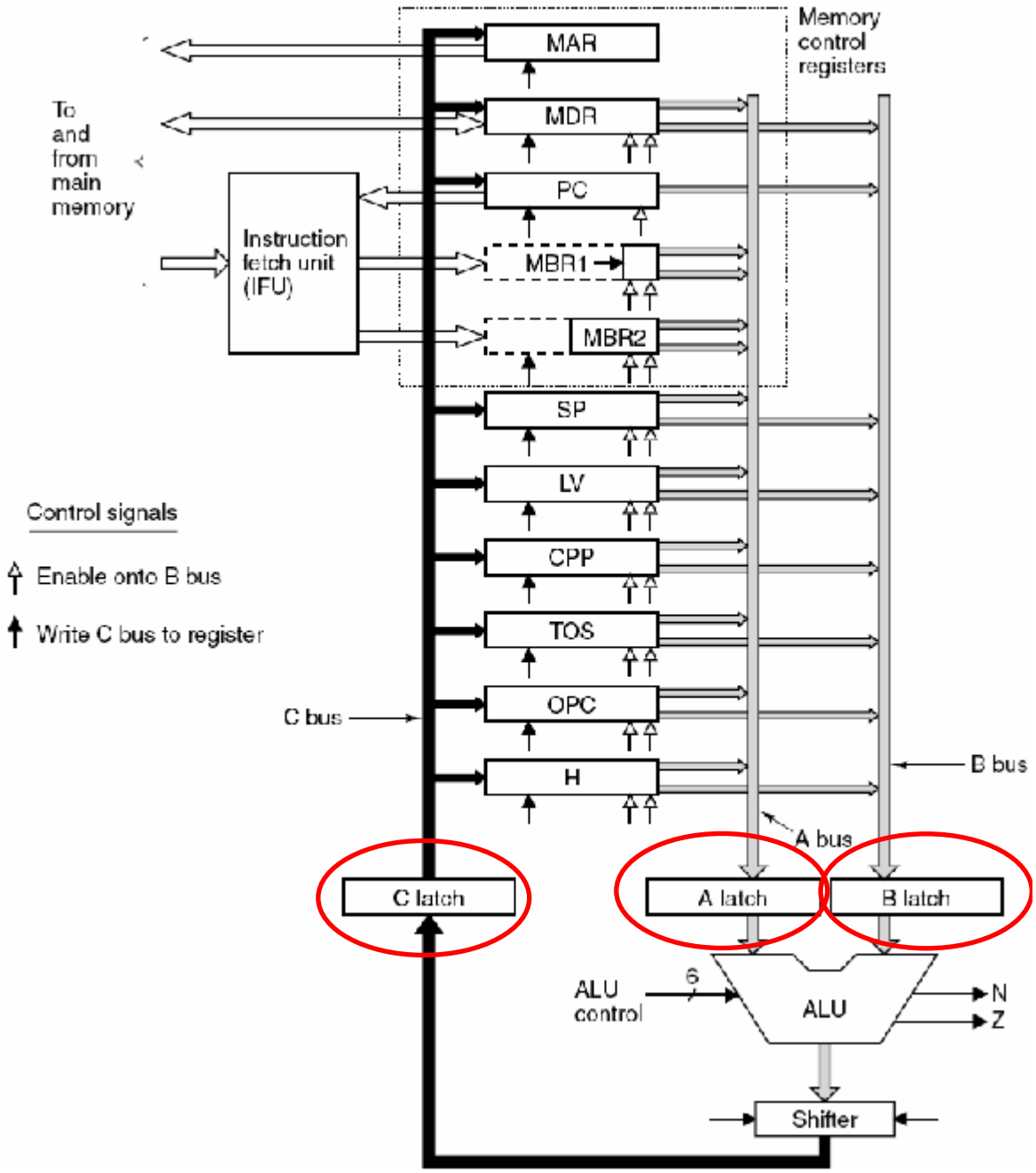


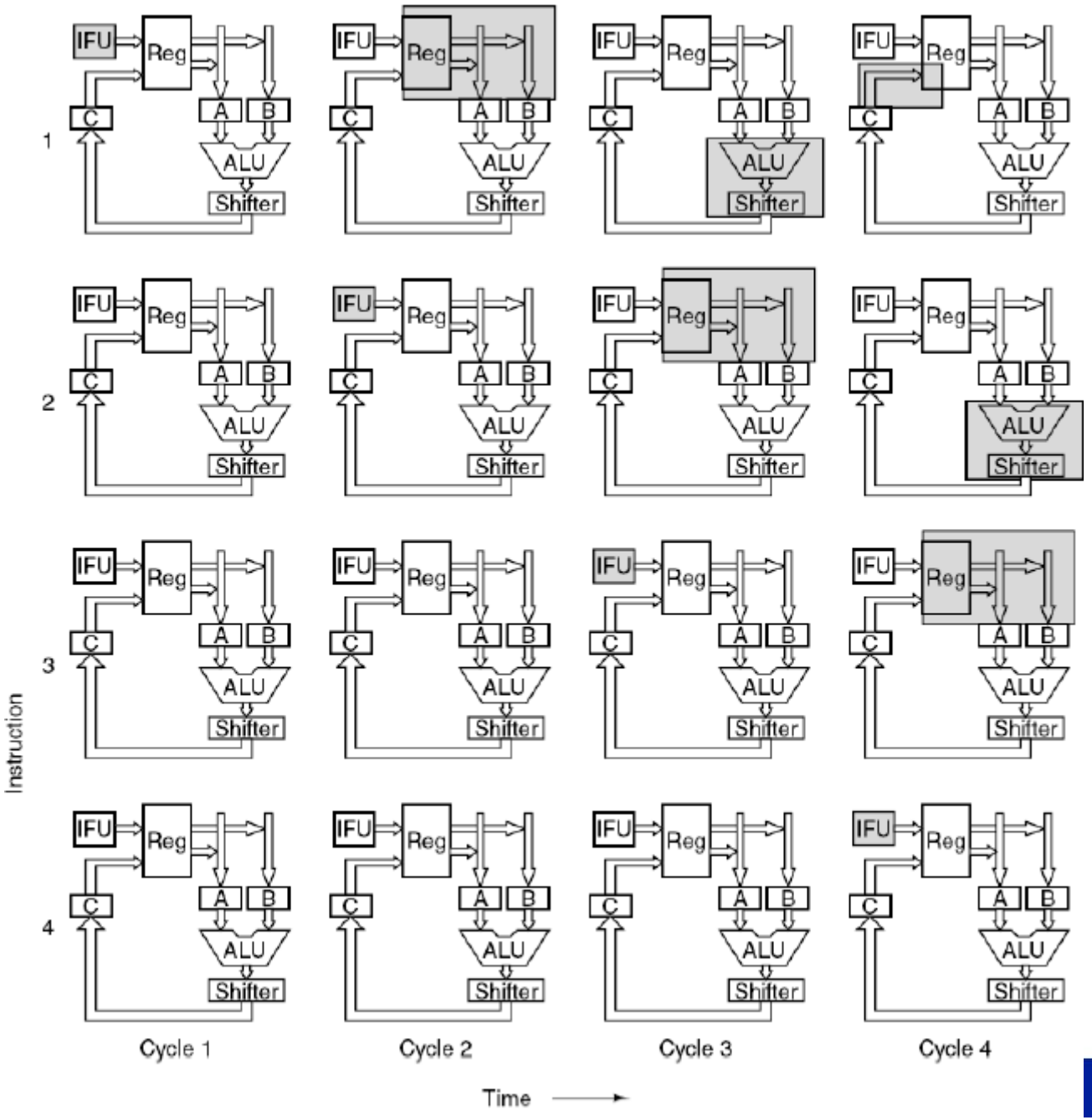
1. Les fra register
2. Utfør ALU-op.
3. Skriv til register
4. (Hovedlageraksess)

Samleband


Memory control registers
MAR
MDR
PC
MBR1
MBR2
SP
LV
CPP
TOS
OPC
H
To and from main memory
Instruction fetch unit (IFU)
Control signals
Enable onto B bus
Write C bus to register
C bus
B bus
A bus
C latch
A latch
B latch
ALU control
6
ALU
N
Z
Shifter

Samleband

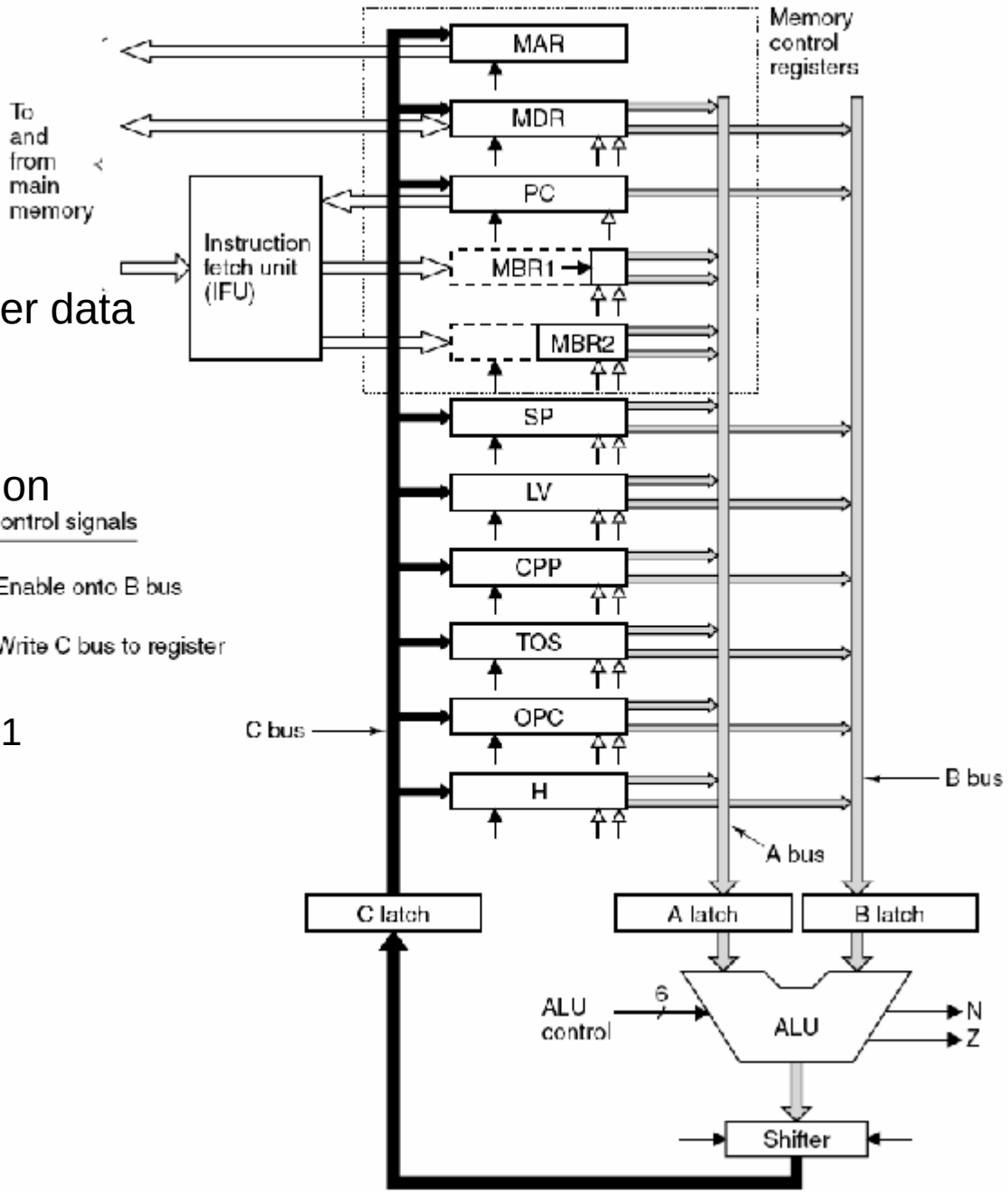

Memory control registers
MAR
MDR
PC
MBR1
MBR2
SP
LV
CPP
TOS
OPC
H
To and from main memory
Instruction fetch unit (IFU)
Control signals
Enable onto B bus
Write C bus to register
C bus
B bus
A bus
C latch
A latch
B latch
ALU control
6
ALU
N
Z
Shifter



NTNU
Innovation and Creativity

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



IJVM som eksempel

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

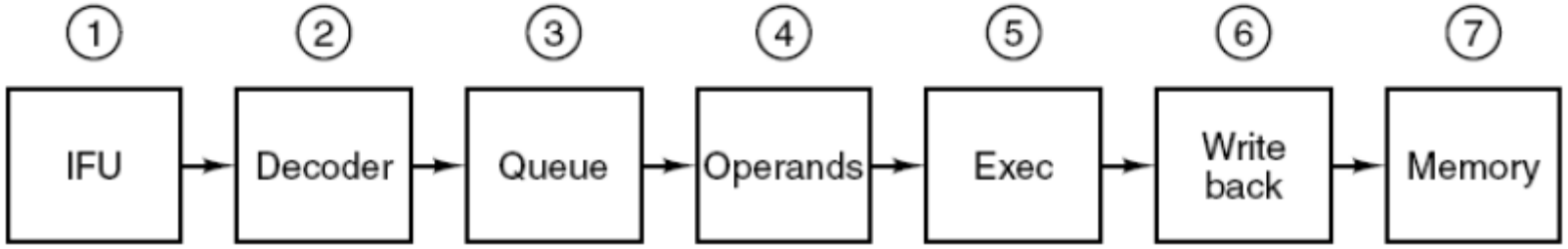
Samleband Hopp og prefetch

- Forutsetter at man vet adressene til µInstruksjonene som skal utføres ”snart”
- Greit ved sekvensiell utføring
- **Ikke greit** ved hoppinstruksjoner
  - Eks: goto (MBR)
- Mulig strategi
  - Sjekk om siste µInstr. er en hoppinstruksjon
  - Hvis nei, hent videre
  - Hvis ja, stopp inntil etterfølgende µInstr. er klar

NTNU
Innovation and Creativity

Samleband 7 steg

1. Hent instruksjoner (som før)
2. Finn type og lengde på instruksjon
3. Finn mikroprogram og legg mikroinstruksjonene i dette mikroprogrammet i en kø
4. Hent operander (som før)
5. Utfør ALU-operasjon (som før)
6. Skriv til register (som før)
7. Hovedlageraksess (som før)

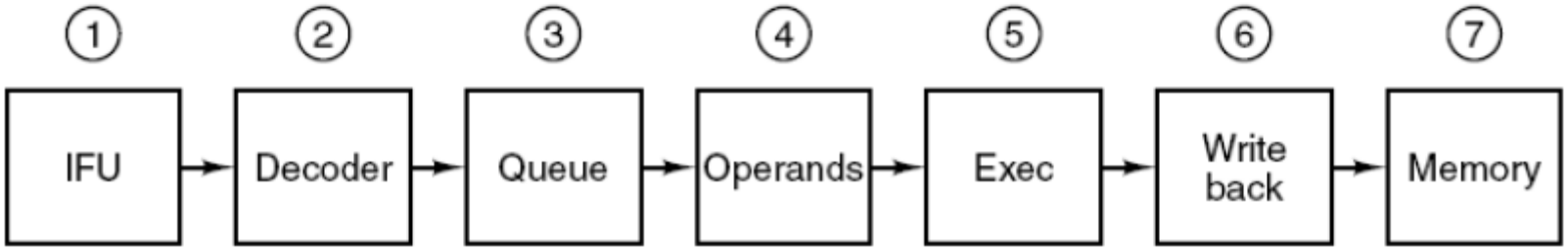


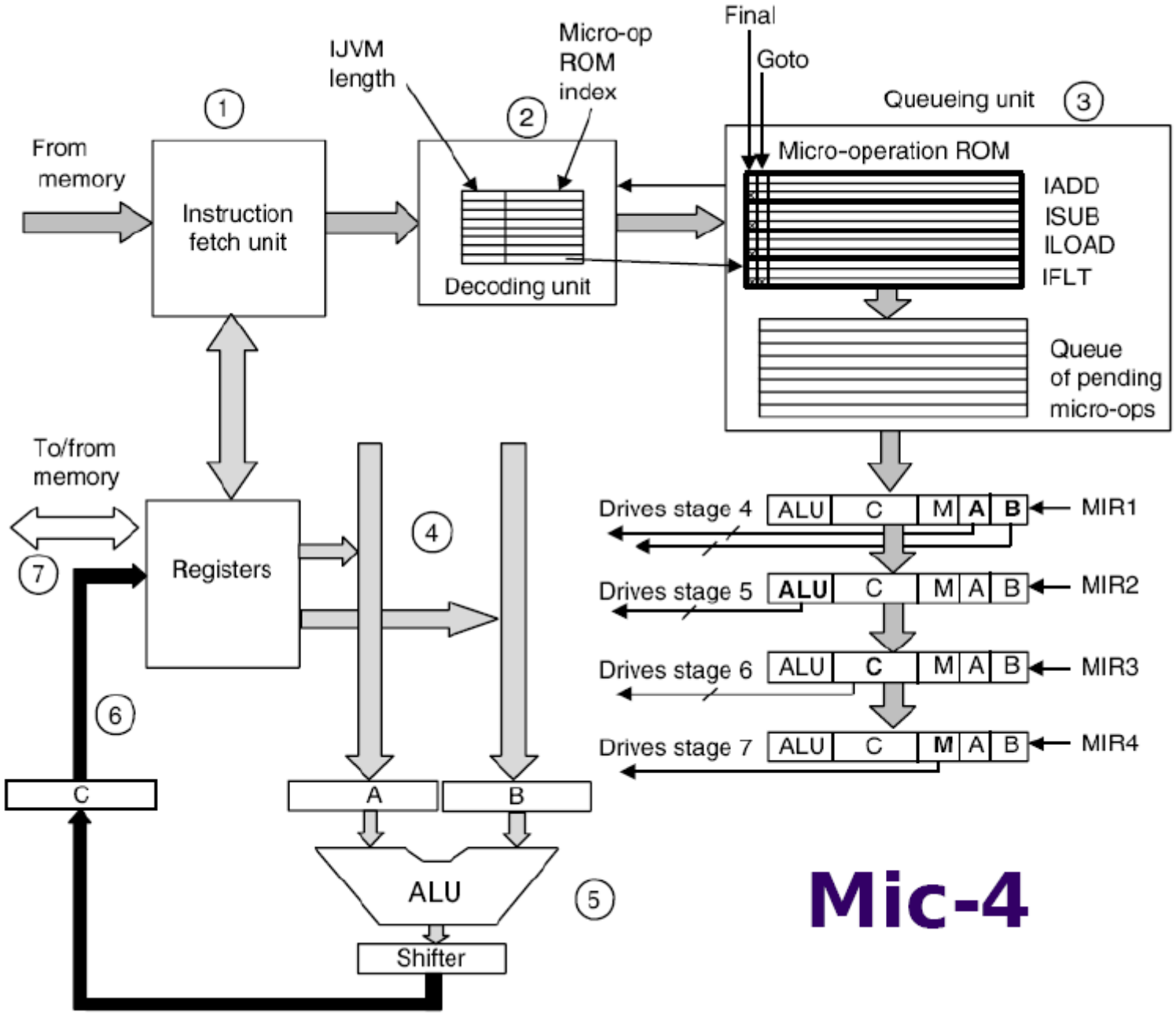
Samleband dekoding instr. kø

- Dekoding
  - Slå opp på Opcode (første instr.byte)
  - Finn instr.lengde og adresse til mikroprogram
  - Trenger instr.lengde for å skille opcode fra operander
- Instruksjonskø
  - Inneholder styrelager og kø
  - Kopierer mikroprogram til kø
  - Hvis finner hoppinstr., stopp til adresse er kjent
  - Hvis ikke hoppinstr., få neste fra dekoder

Samleband 7 steg

1. Hent instruksjoner (som før)
2. Finn type og lengde på instruksjon
3. Finn mikroprogram og legg mikroinstruksjonene i dette mikroprogrammet i en kø
4. Hent operander (som før)
5. Utfør ALU-operasjon (som før)
6. Skriv til register (som før)
7. Hovedlageraksess (som før)





Mic-4

NTNU

novation and Creativity

IJVM som eksempel

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