Norwegian University of Science and Technology
Engineering
The Department of Computer and Information Science

TDT4160
DATAMASKINER GRUNNKURS
EKSAMEN

17. AUGUST, 2013, 09:00–13:00

Kontakt under eksamen:
Gunnar Tufte 73590356/97402478

Tillatte hjelpemidler:
D.
Ingen trykte eller håndskrevne hjelpemidler tillatt.
Bestemt, enkel kalkulator tillatt.

Målform:
Bokmål

OPPGAVE 1: OPPSTART, LITT AV HVERT (25 %)

a. Minnesystemet i en datamaskin har ett nivå av hurtigbuffer (cache). Hva er gjennomsnittlig minneaksesstid (memory access time) for minnesystemet hvis hovedminnet har en aksesstid på $10\mu s$ og hurtigbuffer har en aksesstid på $1\mu s$ med et trefforholdstall (hit ratio) på 80 %?

b. Hva er den funksjonelle forskjellen mellom en halvadderer (half adder) og en fulladderer (full adder)?

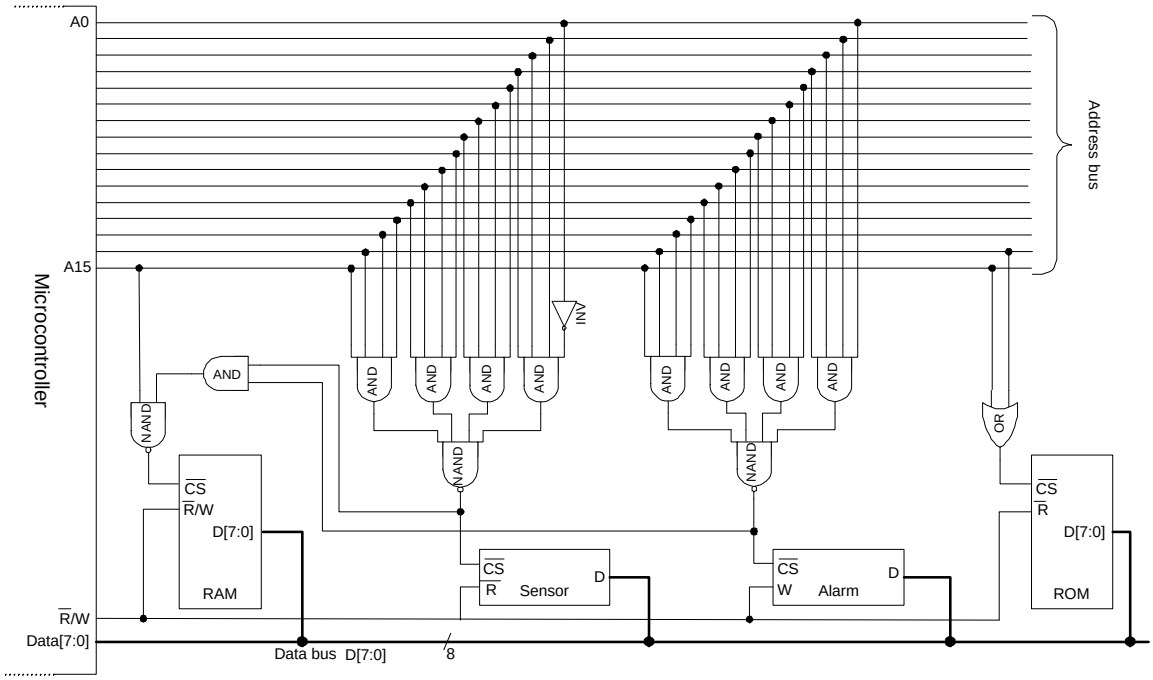
c. Hva karakteriserer en "array processor"?

d. Hva er RAW-avhengighet?

e. I minnesystemsammenheng, forklar lokalitetsprinsippet (the locality principle).

OPPGAVE 2: DIGITALT LOGISK NIVÅ (25 % (15 % PÅ A, 5 % PÅ B OG C))

I figur 1 er det eksterne bussgrensesnittet for en mikrokontroller vist. Det er brukt en RAM-brikke på 64kB for data og en ROM-brikke på 32kB for program. En sensor som overvåker strålingsnivå og en alarm er I/O-enhetene i systemet. Alle enhetene har et aktivt lavt (logisk ”0”) CS (Chip Select)-signal.



Figur 1: Address decoding.

a. Etter en programvareoppdatering er programstørrelsen økt fra 12 104 til 28 104 bytes. Det nødvendige dataområdet for å kjøre programvaren er økt fra 29 324 til 32 078 bytes.

   i) Kan programminnet lagre den nye programvaren? Forklar.

   ii) Kan dataminnet støtte den nye programvaren? Forklar.

b. Er full adressedekoding (address decoding) brukt på noen av enhetene som er koblet til bussen? Forklar.

c. Svar på følgende spørsmål ut i fra den informasjonen du har tilgjengelig:

   i) Er minneenehetene synkrone eller asynkrone? Forklar.

   ii) Er "handshaking” brukt? Forklar.

OPPGAVE 3: MIKROARKITEKTUR OG MIKROINSTRUKSJONER (25 % (5 % PÅ A, 10 % PÅ B OG C))

Bruk vedlagte diagram i figur 3, figur 4, figur 5 og figur 6 for IJVM til å løse oppgavene.

a. Er innholdet i registeret MIR tilgjengelig for programmereren? Forklar.

b. Lag mikroinstruksjon(er) for følgende IJVM-operasjon: *OPC = TOS + LV + CPP*.

   Du trenger ikke ta hensyn til Addr- og J-feltene. Oppgi korrekt bit-verdi for ALU-, C-, Mem- og B-feltene. – Se figur 4.

c. IJVM-registrene i figur 3 er satt til følgende verdier:

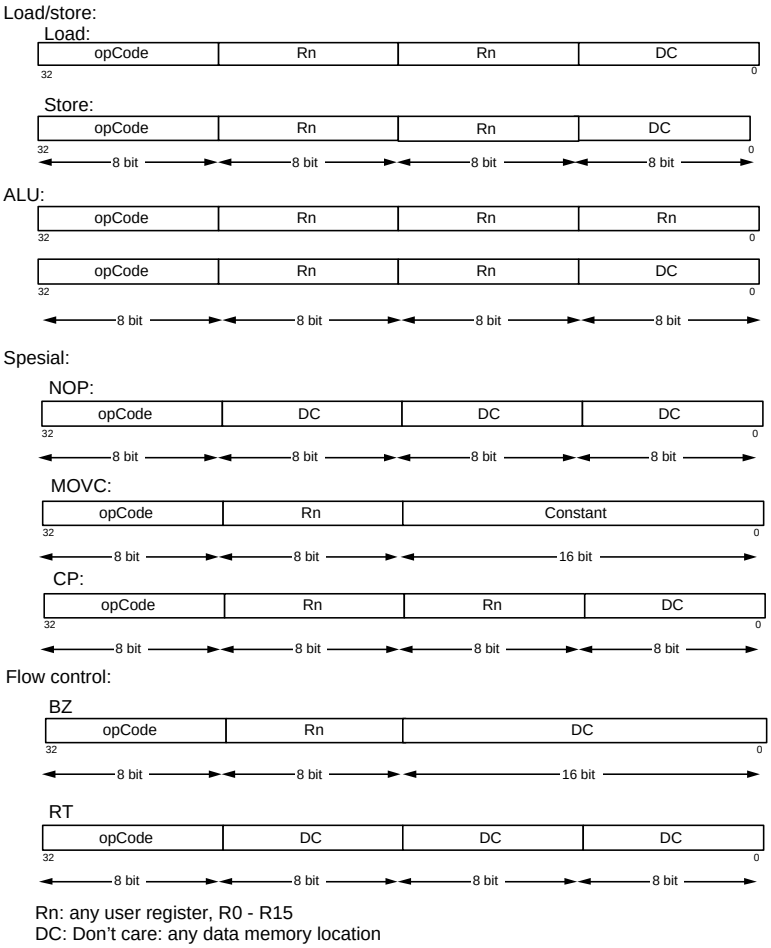
   ”SP”: hex(0001),
   ”LV”: hex(0505),
   ”CPP”: hex(0003),
   ”TOS”: hex(0004),
   ”OPC”: hex(0005),
   ”H”: hex(FF0A).

   Hva er TOS-registerets innhold etter at de to følgende mikroinstruksjonene har blitt utført? Oppgi svaret i hex-format.

   1: ALU: 010100, C: 100000000, Mem: 000 og B: 0101
   2: ALU: 111101, C: 001000000, Mem: 000 og B: 0101

OPPGAVE 4: INSTRUKSJONSSETT ARKITEKTUR (ISA)(25 %)

En svært enkel prosessor har en laste- og lagre-instruksjon (load and store instruction), 8 ALU-instruksjoner, noen spesielle instruksjoner som inkluderer NOP-instruksjonen og to flytkontroll-instruksjoner (flow control instructions). Instruksjonsformatet er vist i figur 2. Alle registre og busser er 32-bit. Prosessoren har en Harvard-arkitektur.



Figur 2: Instruction formats.

Instructions set:

LOAD: Load data from memory.

load Rn, Rn Load register Rn from memory location in Rn.

STORE: Store data in memory.

store Rn, Rn Store register Rn in memory location in Rn.

ALU: Data manipulation, register–register operations.

ADD Rn, Rn, Rn ADD, $Rn = Rn + Rn$. Set Z-flag if result =0.

NAND Rn, Rn, Rn Bitwise NAND, $Rn = \overline{Rn \cdot Rn}$. Set Z-flag if result =0.

OR Rn, Rn, Rn Bitwice OR, $Rn = Rn + Rn$. Set Z-flag if result =0.

INV Rn, Rn Bitwice invert, $Rn = \overline{Rn}$. Set Z-flag if result =0.

INC Rn, Rn Increment, $Rn = Rn + 1$. Set Z-flag if result =0.

DEC Rn, Rn Decrement, $Rn = Rn - 1$. Set Z-flag if result =0.

MUL Rn, Rn, Rn Multiplication, $Rn = Rn * Rn$. Set Z-flag if result =0.

CMP, Rn, Rn Compare, Set Z-flag if $Rn = Rn$

Special: Misc.

CP Rn, Rn Copy, $Rn < -Rn$

NOP Waste of time, 1 clk cycle.

MOVC Rn, constant Put a constant in register $Rn = C$.

Flow control: Branch.

BZ, Rn Conditional branch on zero, $PC = Rn$.

RT Return, return from branch.

Rn: Any user register.
DC: Don't care.

a. Instruksjonen BZ lagrer innholdet av PC og statusregisteret før hoppet (the branch) utføres.

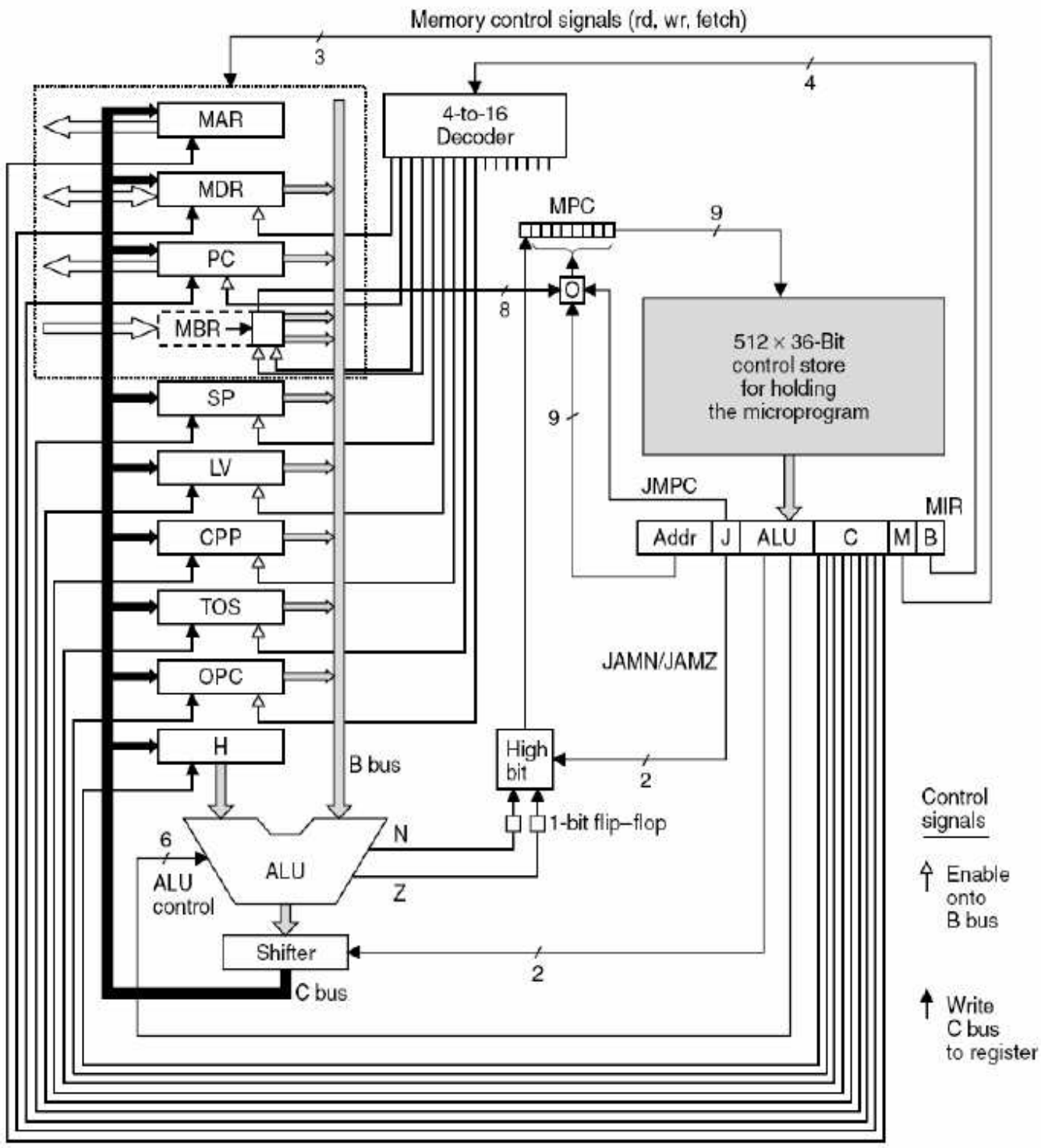
   i) Hvorfor?

   ii) Hvor lagres slik informasjon vanligvis?

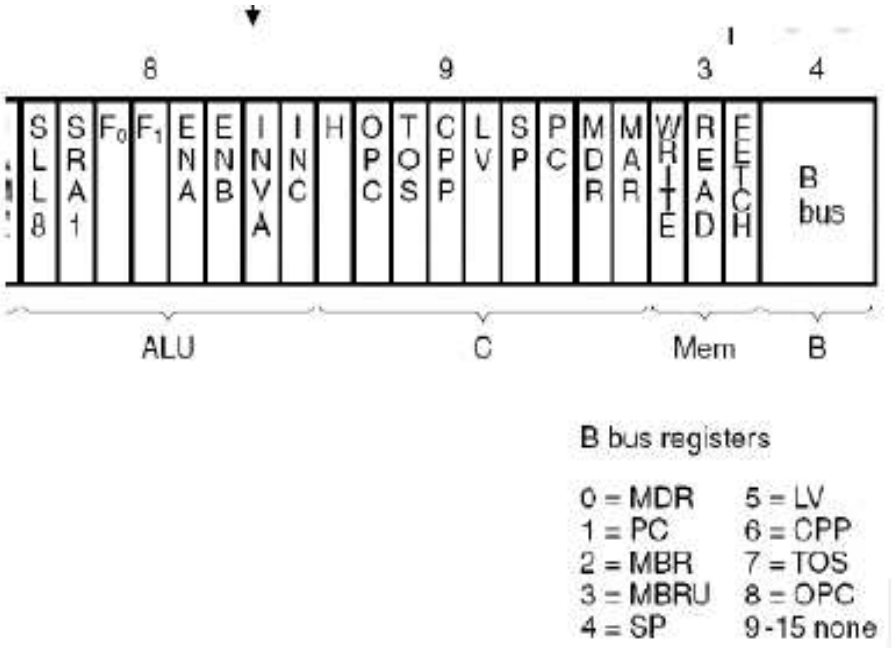
b. Denne prosessoren er sterkt influert av RISC designfilosofi. Hvilke deler av den tilgjengelige informasjonen støtter dette utsagnet?

c. Hvilke typer adresseringsmodi (addressing modes) er brukt for de ulike instruksjonene i denne prosessoren?

IJVM appendix



Figur 3: Block diagram (IJVM).



Figur 4: Microinstruction format (IJVM).

Answer Key for The Exam

OPPGAVE 1: OPPSTART, LITT AV HVERT (25 %)

a. Minnesystemet i en datamaskin har ett nivå av hurtigbuffer (cache). Hva er gjennomsnittlig minneaksesstid (memory access time) for minnesystemet hvis hovedminnet har en aksesstid på $10\mu s$ og hurtigbuffer har en aksesstid på $1\mu s$ med et trefforholdstall (hit ratio) på 80 %?

Answer: $2s$; mean access time = $c + (1 - h) * m)$; $1 + (1 - 0.8) * 10) = 2$

b. Hva er den funksjonelle forskjellen mellom en halvadderer (half adder) og en fulladderer (full adder)?

Answer: A full adder calculate $a + b + carry$ and outputs the sum and a carry bit. A half adder calculates $a + b$ and outputs the sum and a carry bit.

c. Hva karakteriserer en "array processor"?

Answer: large number of identical processors executing the same program on different data sets.

d. Hva er RAW-avhengighet?

Answer: A micro instruction (or pipeline stage try to read a register before it is written to by the previous (micro) instruction.

e. I minnesystemsammenheng, forklar lokalitetsprinsippet (the locality principle).

Answer: Locality in space and time. If a memory location has been accessed, probability of a access a neighbouring address. If a data has been accessed, probability of re-accessing. The cache and memory hierarchies build on these principles.

OPPGAVE 2: DIGITALT LOGISK NIVÅ (25 % (15 % PÅ A, 5 % PÅ B OG C))

I figur 1 er det eksterne bussgrensesnittet for en mikrokontroller vist. Det er brukt en RAM-brikke på 64kB for data og en ROM-brikke på 32kB for program. En sensor som overvåker strålingsnivå og en alarm er I/O-enhetene i systemet. Alle enhetene har et aktivt lavt (logisk ”0”) CS (Chip Select)-signal.

a. Etter en programvareoppdatering er programstørrelsen økt fra 12 104 til 28 104 bytes. Det nødvendige dataområdet for å kjøre programvaren er økt fra 29 324 til 32 078 bytes.

   i) Kan programminnet lagre den nye programvaren? Forklar.

   ii) Kan dataminnet støtte den nye programvaren? Forklar.

Answer: i) no its only 2 14 of program memory
ii) yes, the data memory is of size $2^15 - 2$.

b. Er full adressedekoding (address decoding) brukt på noen av enhetene som er koblet til bussen? Forklar.

   Answer: Yes on the sensor and alarm. All addresses fully specified.

c. Svar på følgende spørsmål ut i fra den informasjonen du har tilgjengelig:

   i) Er minneenehetene synkrone eller asynkrone? Forklar.

   ii) Er "handshaking” brukt? Forklar.

   Answer: Asyncron, no clock.
   No handshaking, no req ack signals.

OPPGAVE 3: MIKROARKITEKTUR OG MIKROINSTRUKSJONER (25 % (5 % PÅ A, 10 % PÅ B OG C))

Bruk vedlagte diagram i figur 3, figur 4, figur 5 og figur 6 for IJVM til å løse oppgavene.

a. Er innholdet i registeret MIR tilgjengelig for programmereren? Forklar.

Answer: Not visible. No way to address MIR directly.

b. Lag mikroinstruksjon(er) for følgende IJVM-operasjon: *OPC = TOS + LV + CPP*.

Du trenger ikke ta hensyn til Addr- og J-feltene. Oppgi korrekt bit-verdi for ALU-, C-, Mem- og B-feltene. – Se figur 4.

Answer: Can use: 1: ALU: B-function, B-bus: TOS, C-bus: H
2: ALU: A + B, B-Bus: LV, C-bus: H
3: ALU: A + B, B-bus: CPP, C-bus: OPC

Or any other combination solving the spesification.

c. IJVM-registrene i figur 3 er satt til følgende verdier:

"SP": hex(0001),
"LV": hex(0505),
"CPP": hex(0003),
"TOS": hex(0004),
"OPC": hex(0005),
"H": hex(FF0A).

Hva er TOS-registerets innhold etter at de to følgende mikroinstruksjonene har blitt utført? Oppgi svaret i hex-format.

1: ALU: 010100, C: 100000000, Mem: 000 og B: 0101
2: ALU: 111101, C: 001000000, Mem: 000 og B: 0101

Answer: 1:
ALU: 010100 (B) C: 100000000 (H) Mem: 000 (ingen mem opprasjon) B: 0101 (5 LV)

2
ALU: 111101 (A+B +1) C: 001000000 (TOS) Mem: 000 (ingen mem opprasjon) B: 0101 (5 LV)
(TOS = 0A0B)

OPPGAVE 4: INSTRUKSJONSSETT ARKITEKTUR (ISA)(25 %)

En svært enkel prosessor har en laste- og lagre-instruksjon (load and store instruction), 8 ALU-instruksjoner, noen spesielle instruksjoner som inkluderer NOP-instruksjonen og to flytkontroll-instruksjoner (flow control instructions). Instruksjonsformatet er vist i figur 2. Alle registre og busser er 32-bit. Prosessoren har en Harvard-arkitektur.

Instructions set:

LOAD: Load data from memory.

load Rn, Rn Load register Rn from memory location in Rn.

STORE: Store data in memory.

store Rn, Rn Store register Rn in memory location in Rn.

ALU: Data manipulation, register–register operations.

ADD Rn, Rn, Rn ADD, $Rn = Rn + Rn$. Set Z-flag if result =0.

NAND Rn, Rn, Rn Bitwise NAND, $Rn = \overline{Rn \cdot Rn}$. Set Z-flag if result =0.

OR Rn, Rn, Rn Bitwice OR, $Rn = Rn + Rn$. Set Z-flag if result =0.

INV Rn, Rn Bitwice invert, $Rn = \overline{Rn}$. Set Z-flag if result =0.

INC Rn, Rn Increment, $Rn = Rn + 1$. Set Z-flag if result =0.

DEC Rn, Rn Decrement, $Rn = Rn - 1$. Set Z-flag if result =0.

MUL Rn, Rn, Rn Multiplication, $Rn = Rn * Rn$. Set Z-flag if result =0.

CMP, Rn, Rn Compare, Set Z-flag if $Rn = Rn$

Special: Misc.

CP Rn, Rn Copy, $Rn < -Rn$

NOP Waste of time, 1 clk cycle.

MOVC Rn, constant Put a constant in register $Rn = C$.

Flow control: Branch.

BZ, Rn Conditional branch on zero, $PC = Rn$.

RT Return, return from branch.

Rn: Any user register.
DC: Don't care.

a. Instruksjonen BZ lagrer innholdet av PC og statusregisteret før hoppet (the branch) utføres.

   i) Hvorfor?

   ii) Hvor lagres slik informasjon vanligvis?

   Answer: i) PC: to be able to return from a subrutine
   ststus register: to restore the status register to be able to resume the main program.
   ii) On a stack.

b. Denne prosessoren er sterkt influert av RISC designfilosofi. Hvilke deler av den tilgjengelige informasjonen støtter dette utsagnet?

   Answer: 1) All instructions equal length.
   2) Load/store architecture.
   3) General registers.

c. Hvilke typer adresseringsmodi (addressing modes) er brukt for de ulike instruksjonene i denne prosessoren?

   Answer: Register, e.g. ALU instructions
   Register Indirect: Load/store, BZ
   Imidiate: MOVC

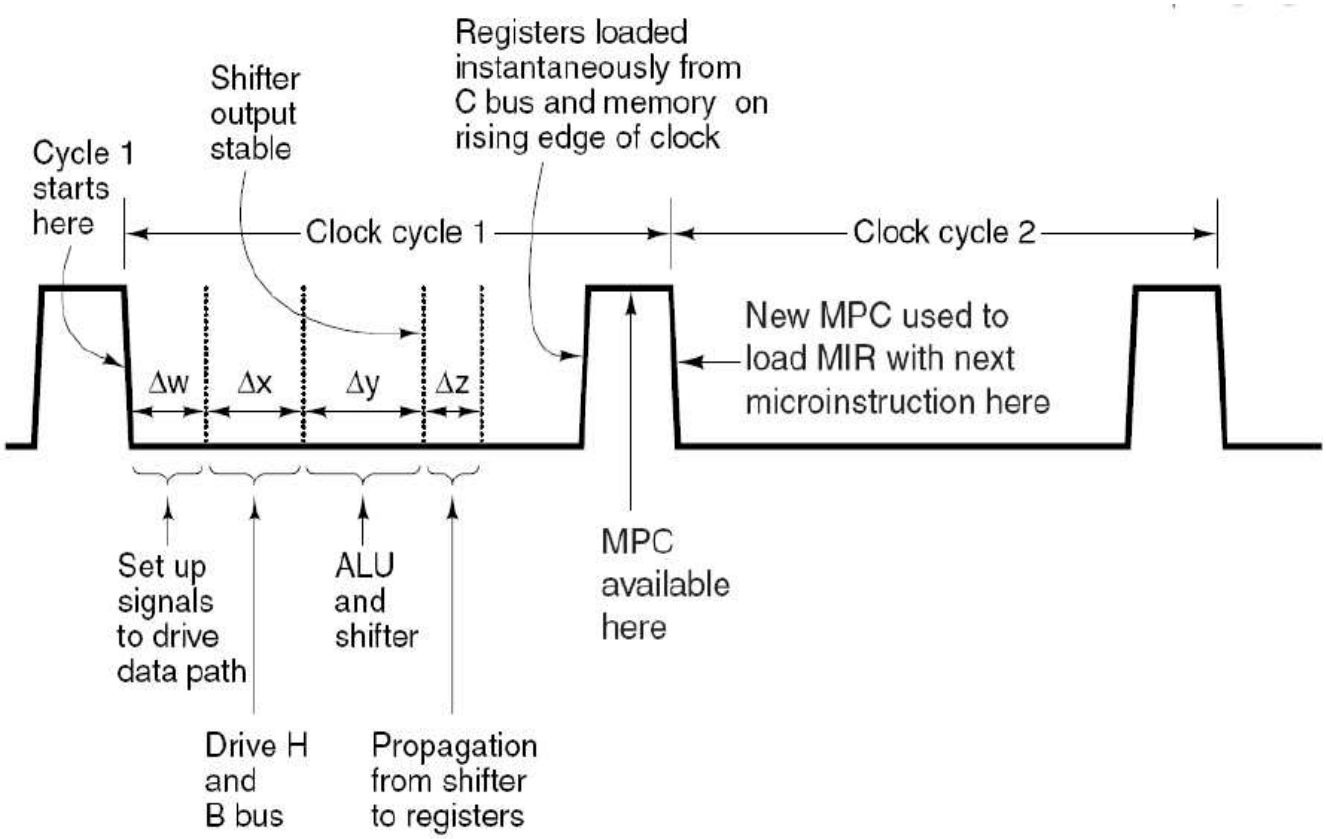
   A bit of slack here on names as long as explained or if they mess with instruction types (zero adr, 2 adr etc).

IJVM appendix

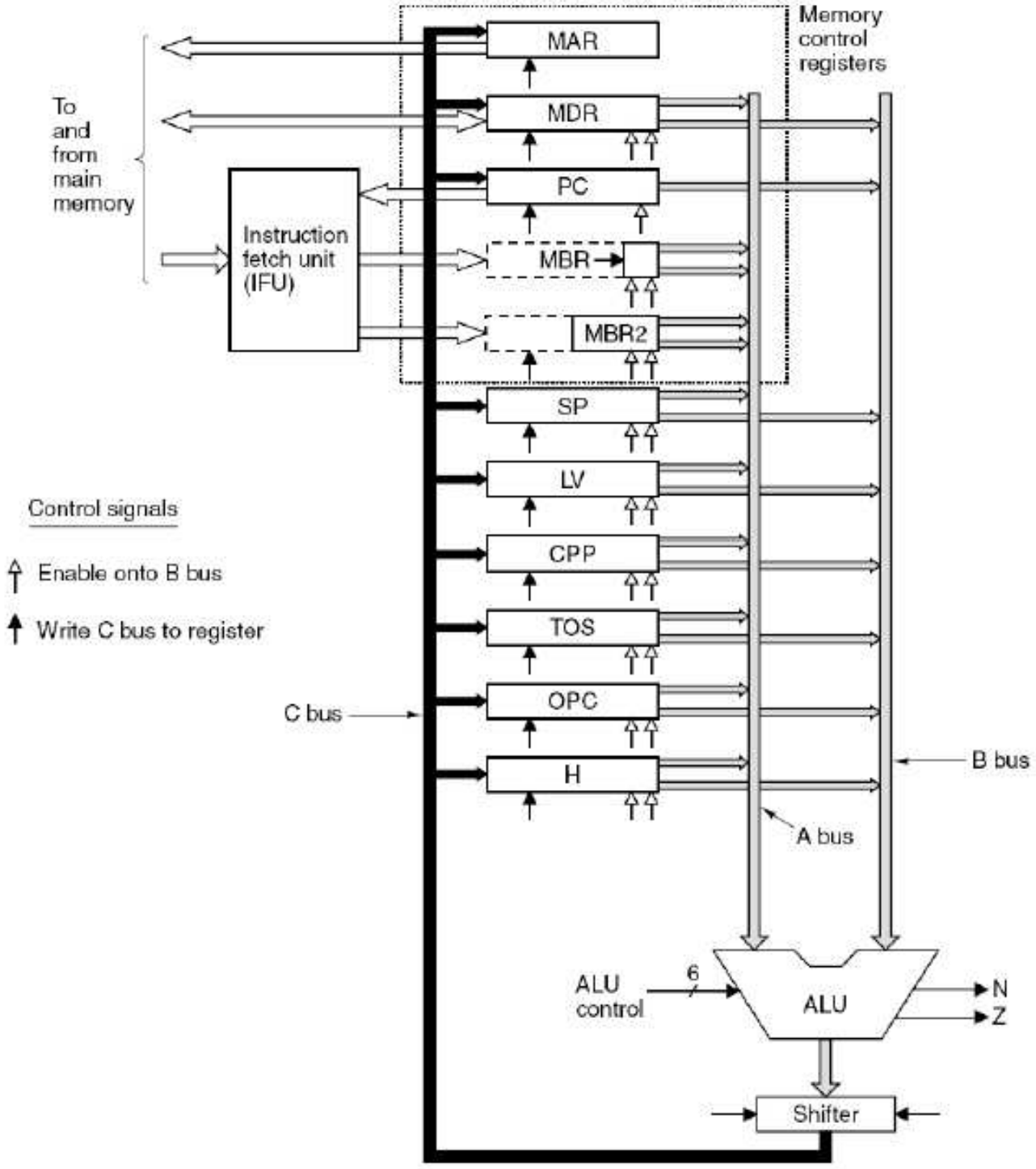
F_0	F_1	ENA	ENB	INVA	INC	Function
0	1	1	0	0	0	A
0	1	0	1	0	0	B
0	1	1	0	1	0	$\overline{A}$
1	0	1	1	0	0	$\overline{B}$
1	1	1	1	0	0	A + B
1	1	1	1	0	1	A + B + 1
1	1	1	0	0	1	A + 1
1	1	0	1	0	1	B + 1
1	1	1	1	1	1	B – A
1	1	0	1	1	0	B – 1
1	1	1	0	1	1	–A
0	0	1	1	0	0	A AND B
0	1	1	1	0	0	A OR B
0	1	0	0	0	0	0
1	1	0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

SLR1	SLL8	Function
0	0	No shift
0	1	Shift 8 bit left
1	0	Shift 1 bit right

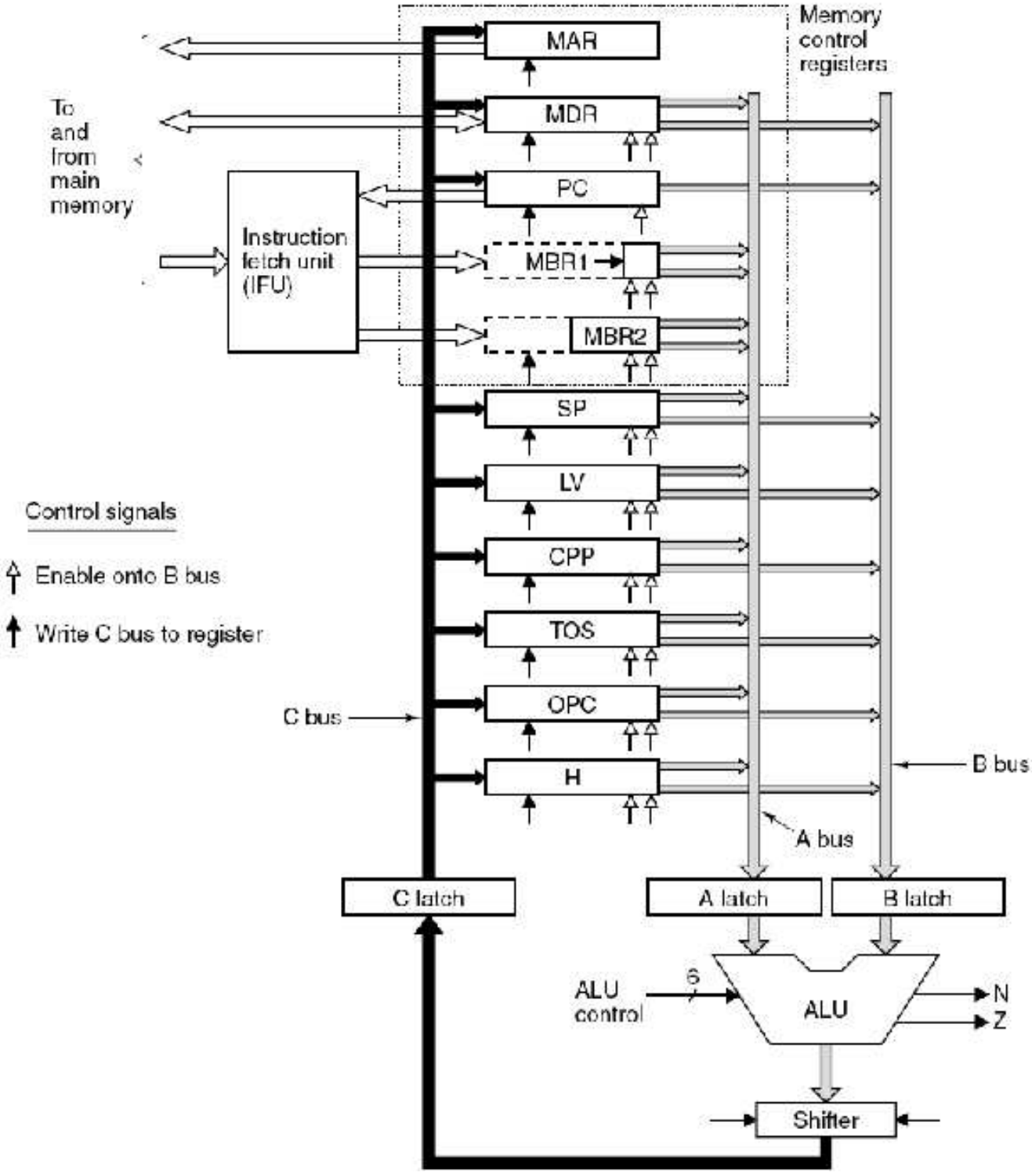
Figur 5: ALU functions (IJVM).



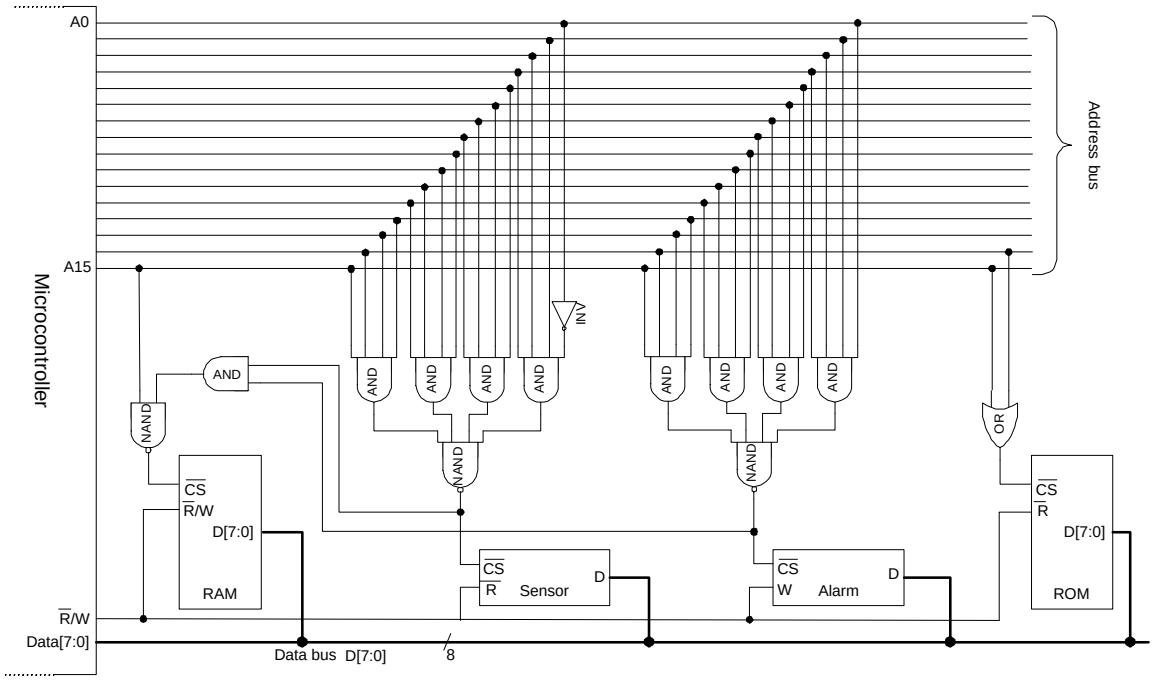
Figur 6: Timing diagram (IJVM).



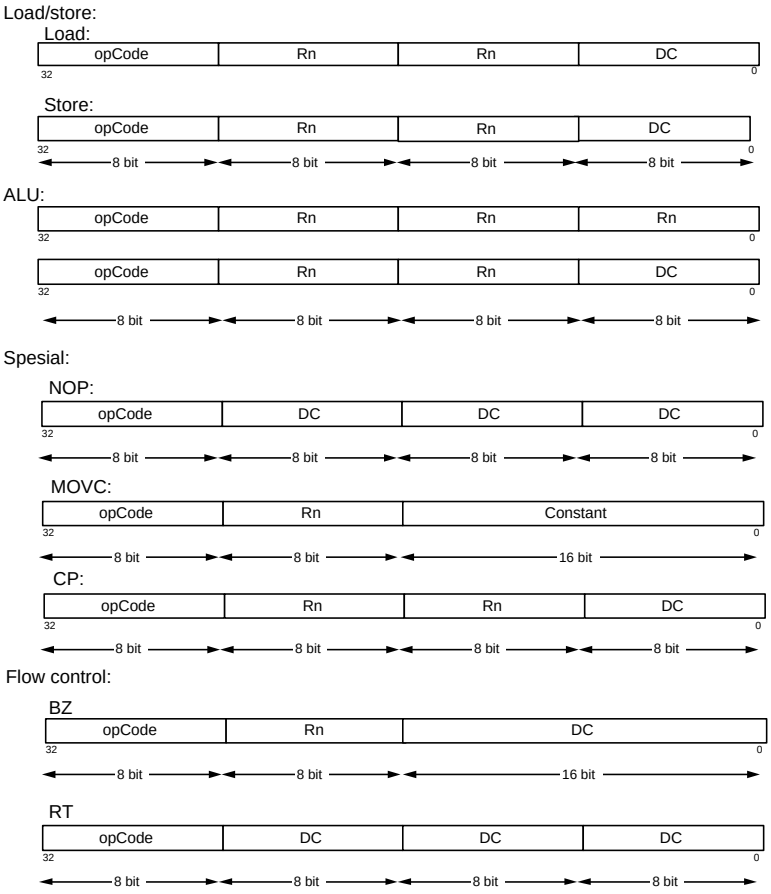
Figur 7: Alternative microarchitecture I.



Figur 8: Alternative microarchitecture II.

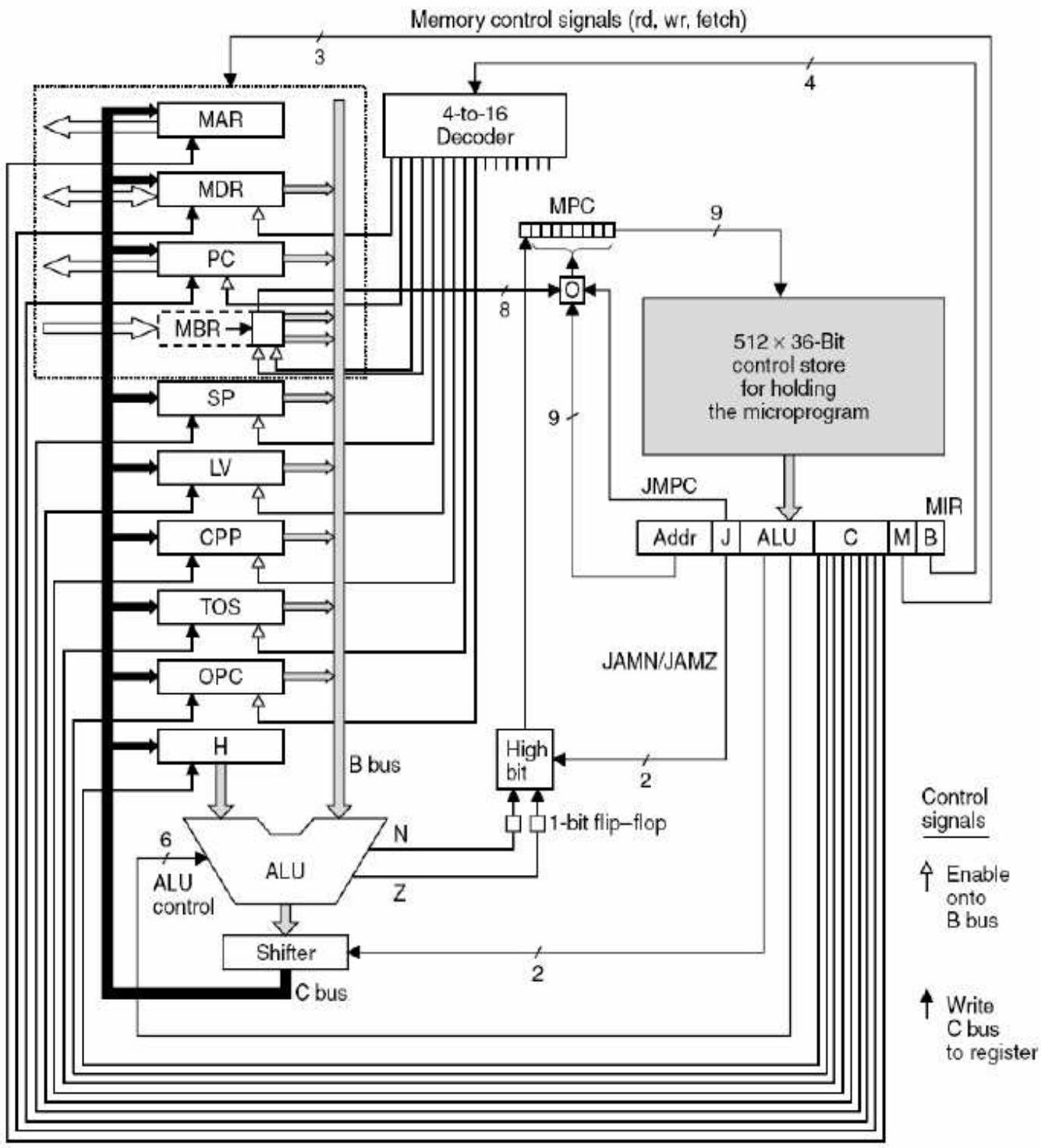


Figur 9: Address decoding.

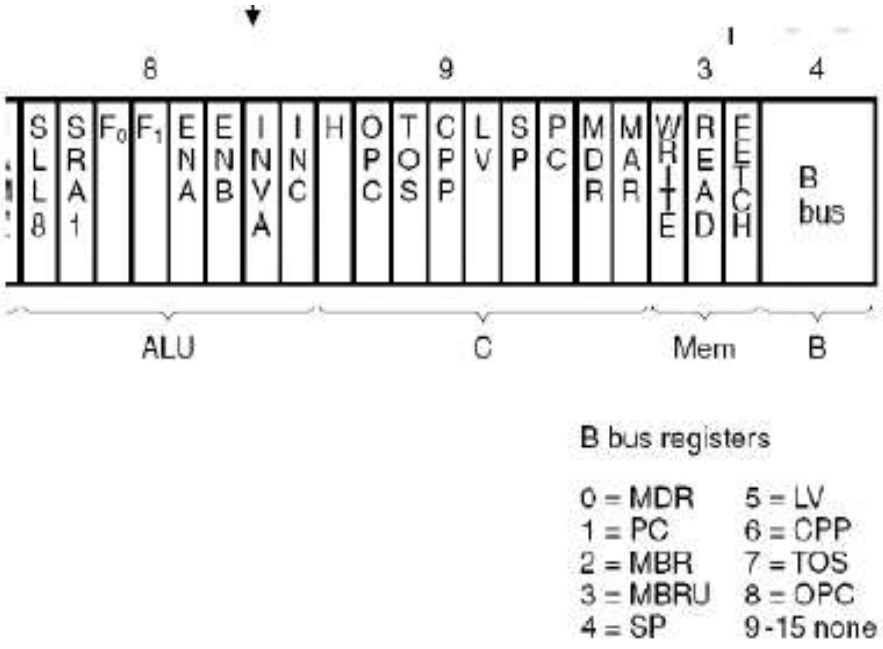


Rn: any user register, R0 - R15
DC: Don't care: any data memory location

Figur 10: Instruction formats.



Figur 11: Block diagram (IJVM).

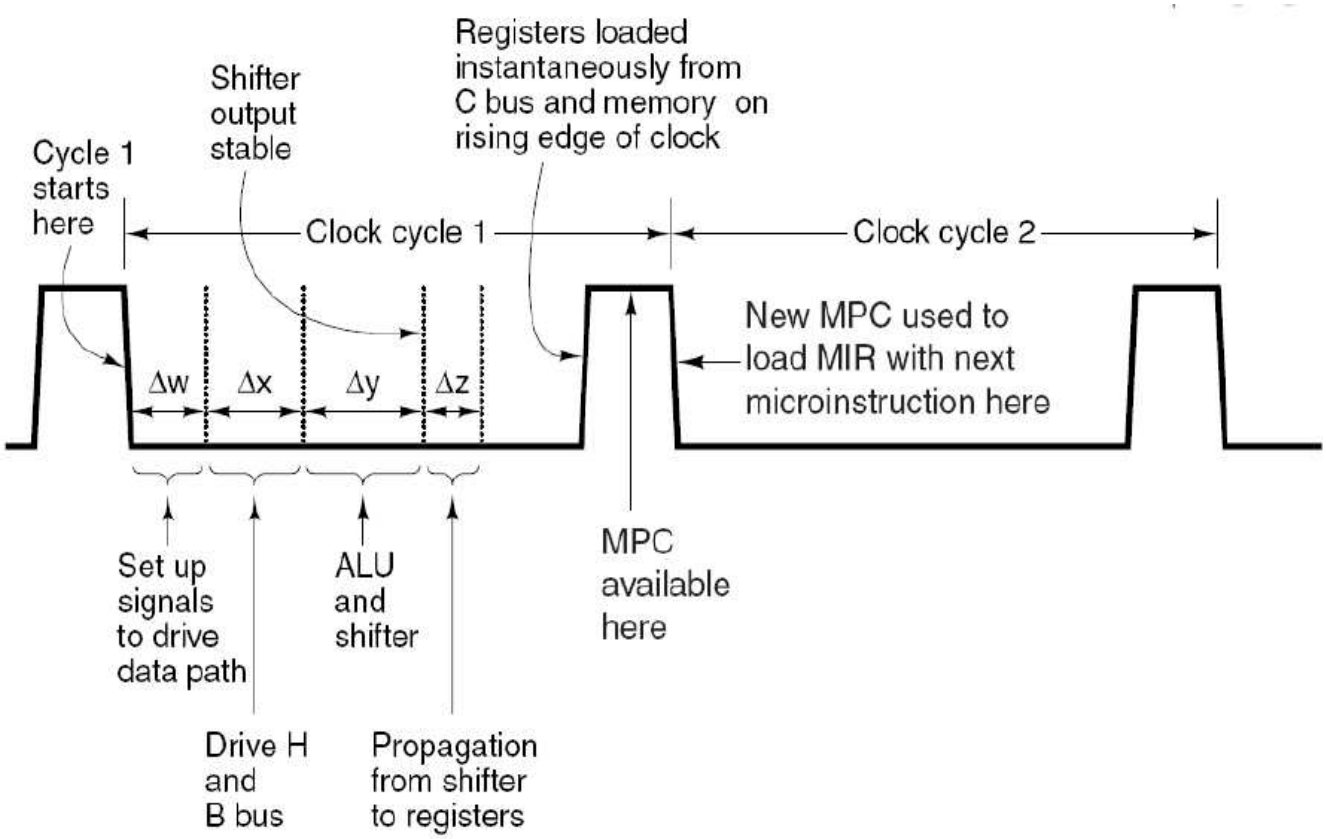


Figur 12: Microinstruction format (IJVM).

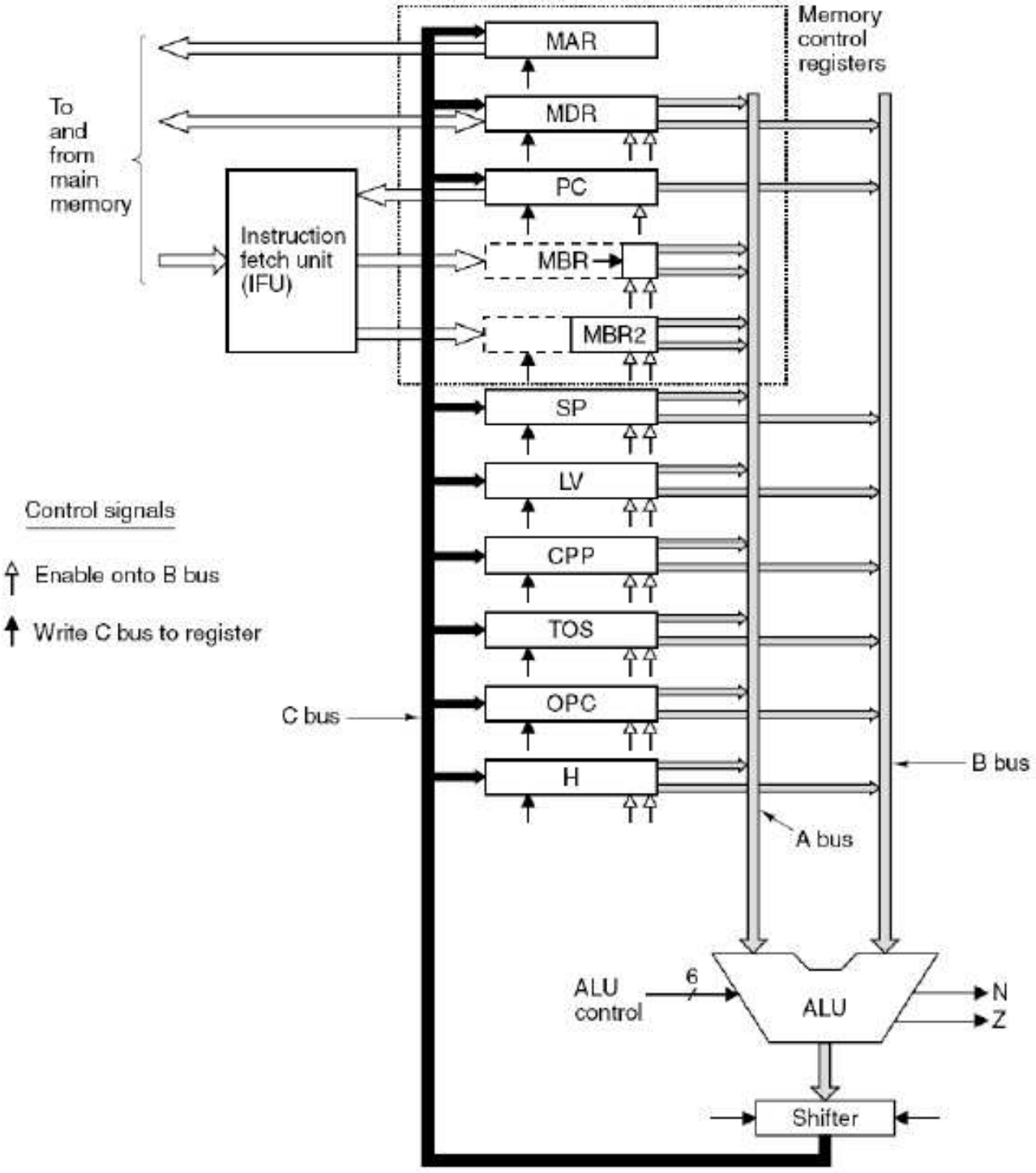
F_0	F_1	ENA	ENB	INVA	INC	Function
0	1	1	0	0	0	A
0	1	0	1	0	0	B
0	1	1	0	1	0	$\overline{A}$
1	0	1	1	0	0	$\overline{B}$
1	1	1	1	0	0	A + B
1	1	1	1	0	1	A + B + 1
1	1	1	0	0	1	A + 1
1	1	0	1	0	1	B + 1
1	1	1	1	1	1	B – A
1	1	0	1	1	0	B – 1
1	1	1	0	1	1	–A
0	0	1	1	0	0	A AND B
0	1	1	1	0	0	A OR B
0	1	0	0	0	0	0
1	1	0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

SLR1	SLL8	Function
0	0	No shift
0	1	Shift 8 bit left
1	0	Shift 1 bit right

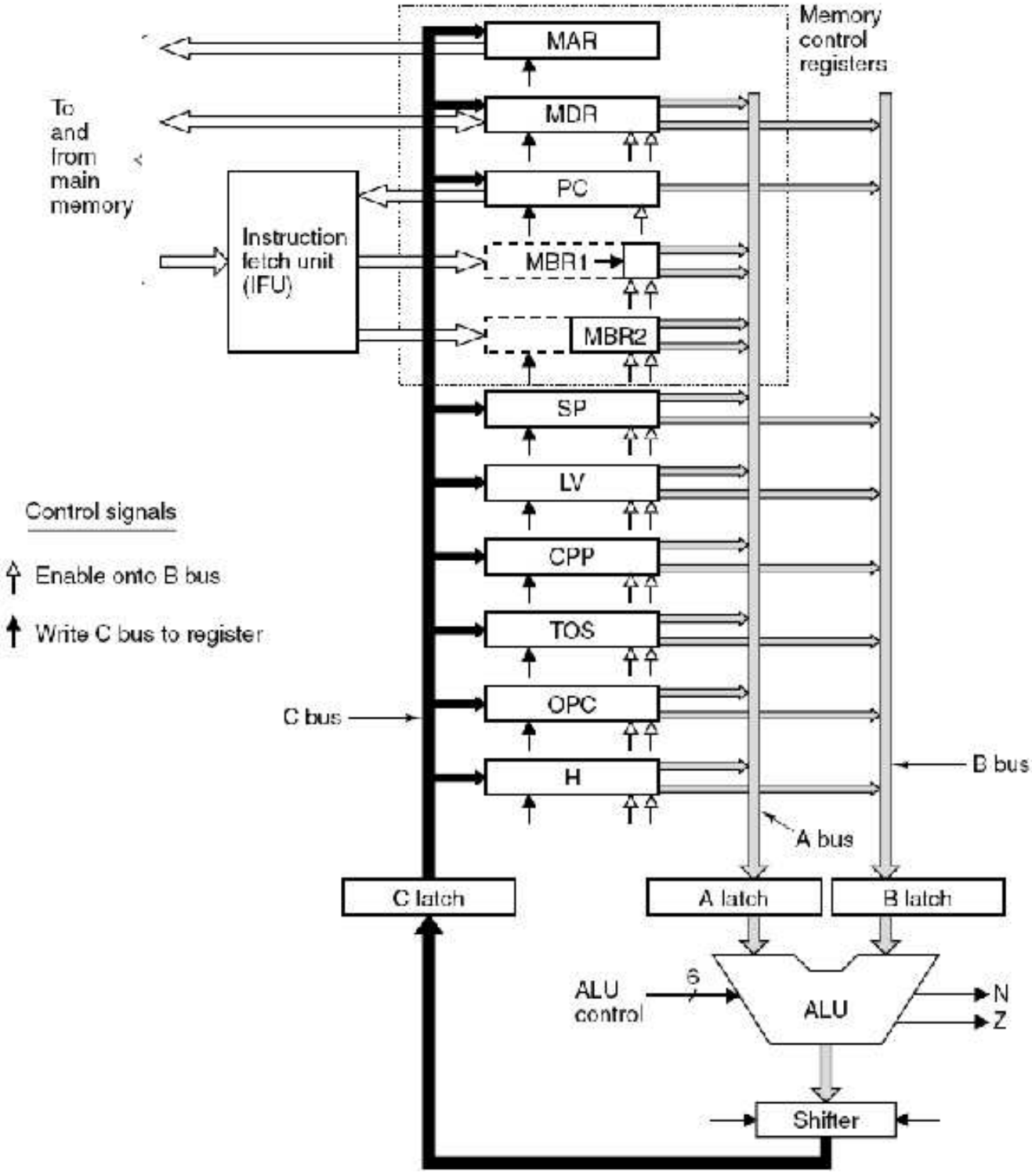
Figur 13: ALU functions (IJVM).



Figur 14: Timing diagram (IJVM).



Figur 15: Alternative microarchitecture I.



Figur 16: Alternative microarchitecture II.