NTNU
Kunnskap for ei betre verd

Institutt for datateknikk og informasjonsvitenskap

Eksamensoppgåve i TDT4160 datamaskiner og digitalteknikk

Fagleg kontakt under eksamen: Johannes H. Jensen

Tlf.: 454 71 010

Eksamensdato: 7. august 2018

Eksamenstid (frå-til): 9:00 – 13:00

Hjelpemiddelkode/Tillatne hjelpemiddel: D: Ingen prenta eller handskrivne hjelpemiddel tillatne. Bestemt, enkel kalkulator tillaten.

Annan informasjon:

Målform/språk: nynorsk

Sidetal (utan framside): 8

Sidetal vedlegg: 3

Informasjon om trykking av eksamensoppgåve

Originalen er:

1-sidig x **2-sidig ☐**

svart/kvit x **fargar ☐**

Skjema for fleire val? ☐

Kontrollert av:

_____

Dato Sign

Merk! Studentane finn sensur i Studentweb. Har du spørsmål om sensuren må du kontakte instituttet ditt. Eksamenskontoret vil ikkje kunne svare på slike spørsmål.

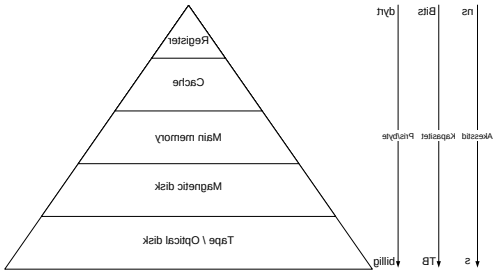
Oppgåve 1 Oppstart, litt av kvart (25 %)

a)

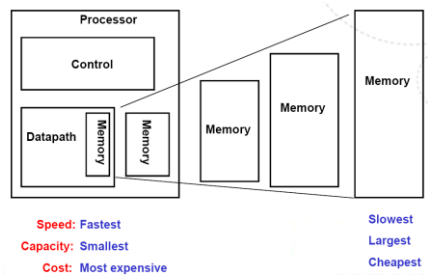
Skisser minnehierarkiet for eit typisk datamaskinsystem. Indiker aksesstid og kostnad (pris/bit).

Svar:

Fleire mulige, men må ha med korleis akesstid og kostnad endrar seg i minnehierarkiet. Skisse kan f. eks. Være:

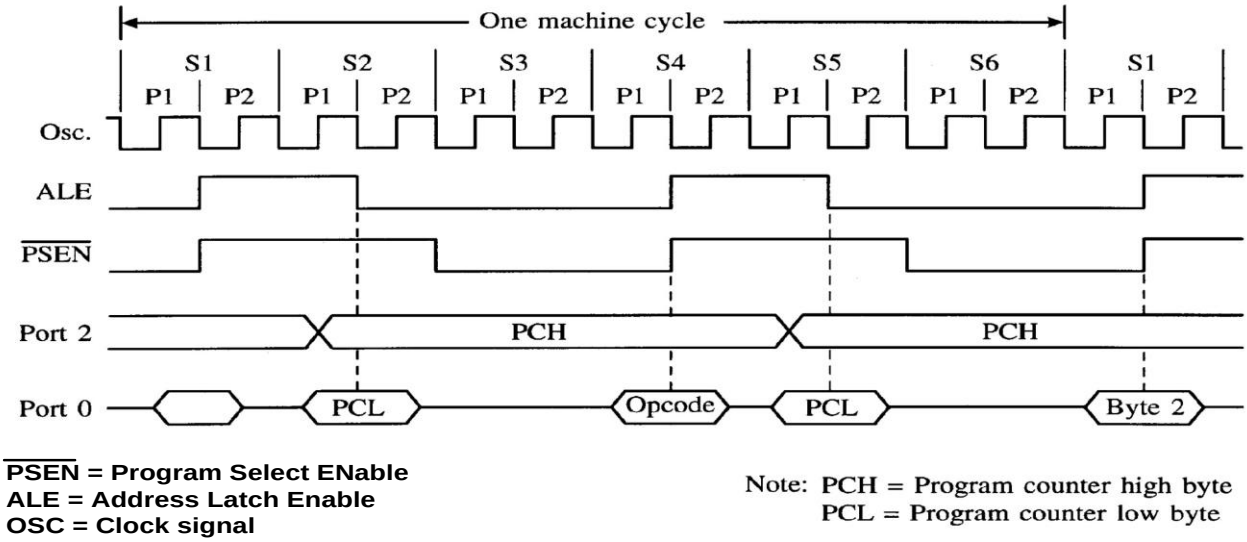


eller



b)

Figur 1 viser ein Mikrokontroller sitt grensesnitt mot eksternt programminne. Bussignala i figuren angir lesing av programminne. Port 0 og Port 2 er 8-bit portar. Prosessoren har 16-bit adressering og 8-bit bus for opcode og operandar. Kva blir metoden som nyttast til å få til dette med kunn to 8-bit portar kalla? Forklar kort kva som skjer når ein instruksjon adresserast og hentast (instruction fetch).



Figur 1 Bussgrensesnitt.

Svar: I figuren vises ein bussoverføring som nyttar **multiplexa adresse og data buss.** Over føringa skjer som følgjer:

1: Prosessoren legjer ut 16 bit adresse til OpCode Lav (8 bit, PCL) på port 0, høg 8 bit på port 1 (PCH).

2: ALE signalet gir signal til latch (som held PCL) latchar låg byte av adresse til Pminne. Pminne adressert med 16 bit PCL og PCH PCH direkte frå port 2, PCL frå latch. VED ALE låg, port 0 overfører data (OpCode eller Operand)

3: ~PSEN går høg, Opcode leses.

4: Viss operandar trengs for OpCode (som her) PCL og PCH opdatert til adressa til operand. etc

c)

Forklar kort:

i) Samanheng mellom samleband (pipeline) djupn (mengd steg) og ILP.

ii) Eventuelle ulemper med djupe samleband.

Svar:

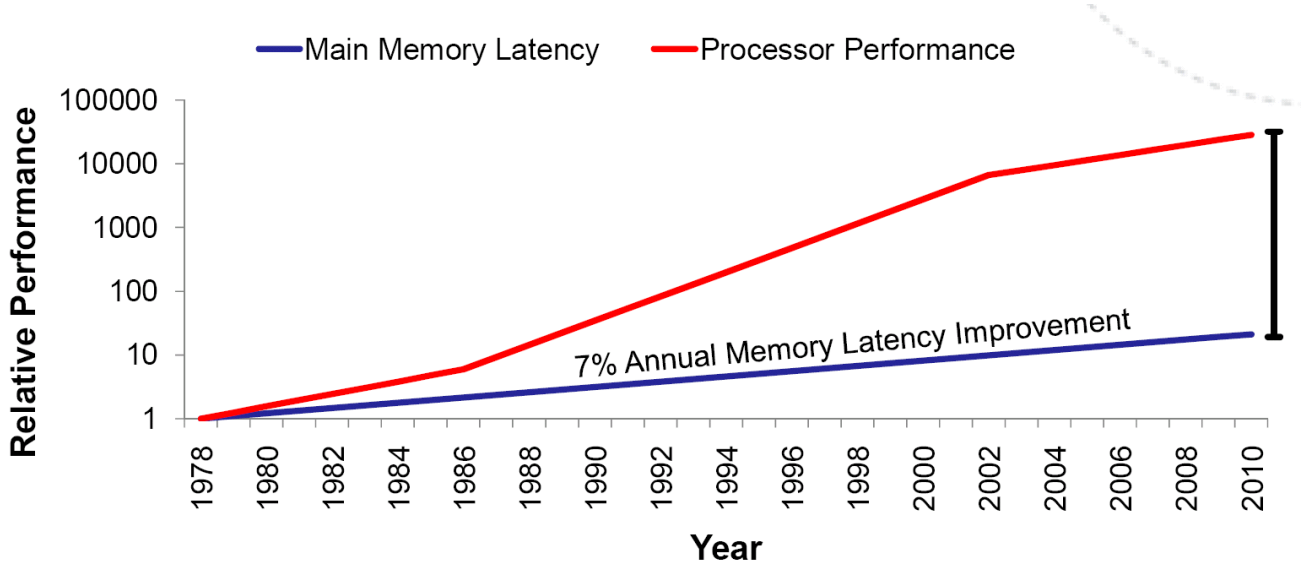
i) Antal steg i smlebåndet aukar antal instruksjonar som ligg underbehandling i samlebandet. Idielt kan ein ha ein instruksjon for kvart steg.

ii) Ved branch der ein ikkje treff må samleband tømmast og så fyllast, dette gir klokke periodar der samlebandet ikkje utfører «nyttigt arbeid»

d)

«Memory wall» (minnevegg) kan mellom anna skape problem innan skalering. Kva utfordring for prosessorar og minne gjer memory wall?

Svar:



Sidan prosessorytelse aukar raskare enn minne (akesstid) vil ein ende opp med at minne ikkje klarar å «fore» prosessoren med data/instruksjonar fort nok. Prosessor skalering (ytelse) var lenge primert auking i klokke frekvens. I dag auking av antal kjerner. Begge desse skaleringsparametera møter «minnevegg» sidan minne vil bli flaskehals.

e)

Samanlikn desse eigenskapane for statisk RAM (SRAM) og dynamisk RAM (DRAM):

i) Areal. **Svar:** (SRAM «stort» (6 trans), DRAM «mindre» (2 trans)

ii) Energibruk. **Svar:** (SRAM «stort» (6 trans), DRAM «mindre» (2 trans)

iii) Aksesstid **Svar:** (SRAM «kort», DRAM «lengre»

Oppgåve 2 Digitalt Logisk Nivå (25 % (a: 5 %, b: 10 %, c: 10 %))

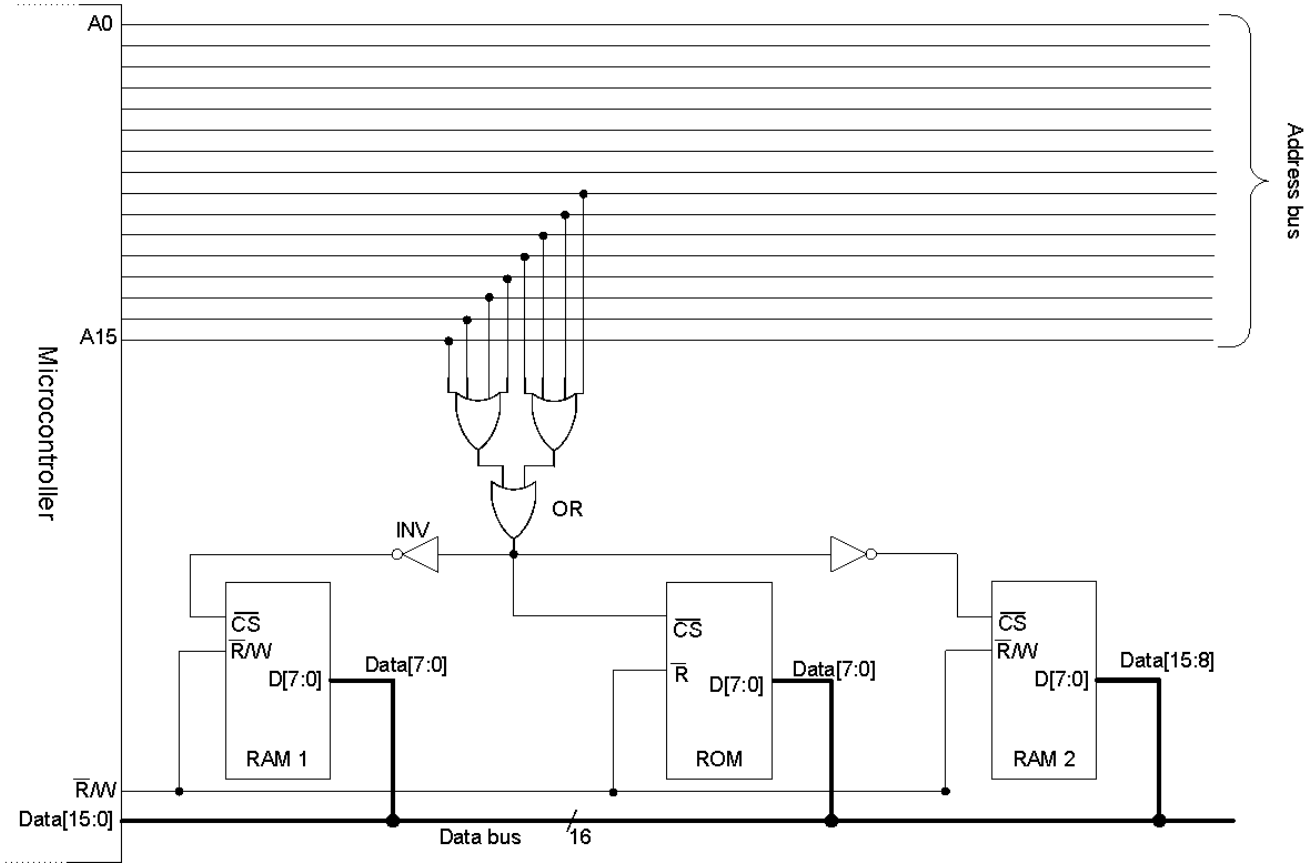
a)

Forklar kort skilnaden på asynkron og synkron bussoverføring?

Svar: asynkron handshake, ikkje klokke som master, synkron klokke master,

b)

I eit innvevd system (embedded system) nyttast det ein mikrokontroller med 8-bit instruksjonar og 16-bit data. Figur 2 viser det eksterne bussgrensesnittet med adressedekodingslogikk for mikrokontrollaren. Det er ein ROM-brikke for program og to RAM-brikkar for data. Alle einingane nyttar eit aktivt lågt (logisk "0") CS (Chip Select)-signal.



Figur 2 Address decoding.

i) Finn adresseområda for RAM og ROM. Eventuelt ledig minneområde og overlapp.

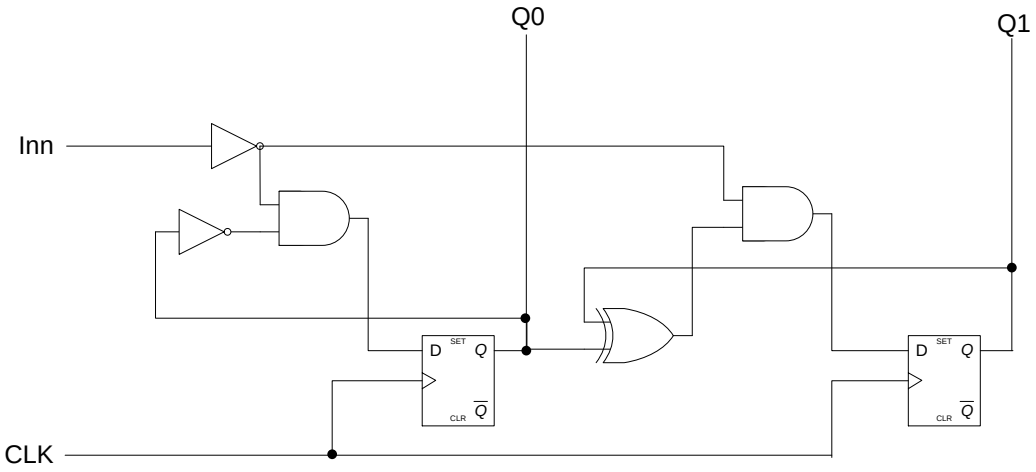
ii) Angi tal på byte som kan adresserast i programminne (ROM) og dataminne (RAM).

Svar: RAM: 0100 - FFFF
ROM: 0000 – 00FF

ROM er 8 bit og det kan då adreserast 256 byte (0000 – 00FF)
RAM (RAM1 og RAM2 gir håg og lav av eit 16 bit ord) det kan då adreserast 65280 byte i kvar brikke, totalt 2 x 65280 byte (130560 byte) sidan kvar adresselokasjon er på 2 byte

c)

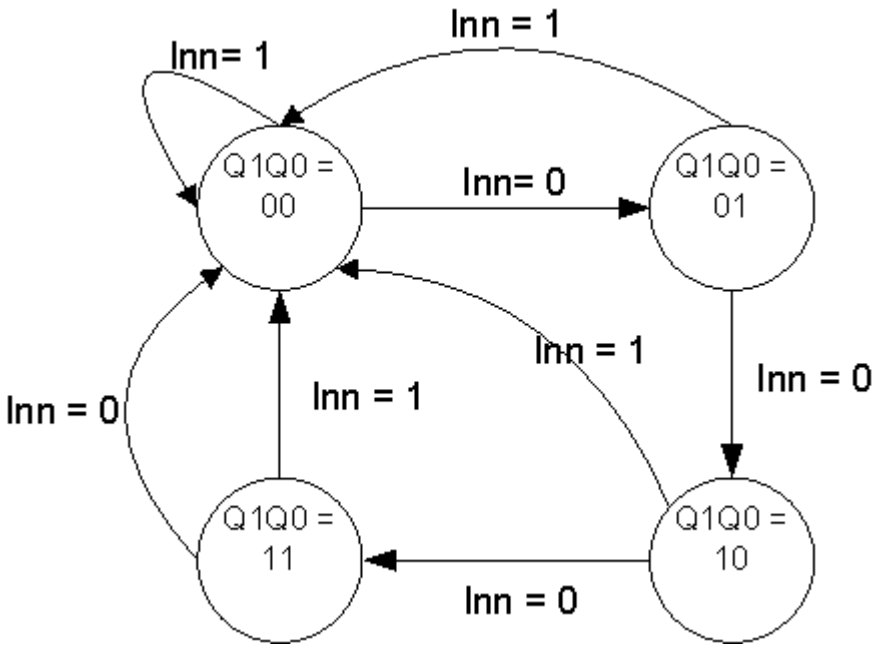
Figur 3 viser ei FSM (Finite State Machine). Datavippe med D0 og Q0 til venstre. Datavippe med D1 og Q1 til høgre.



Figur 3 Finite State Machine logikk.

i) Finn dei logiske uttrykka for D0 og D1 (excitation equation).

ii) Figuren under viser eit forsøk på å teikne tilstandsdiagram for tilstandsmaskina. Er diagrammet korrekt?



Figur 4 Finite State Machine tilstandsdiagram.

Svar:

i) D0 = (~inn and Q0), D1 = ~inn and (Q1 xor Q2)

ii) Tilstandsdiagramet er korrekt. Transisjonar er rett, og det er ingen udefinerte tilstandar eller input kombinasjonar.

Oppgåve 3 Mikroarkitektur og mikroinstruksjoner (20 % (a: 5 %, b: 5 % og 10 % på c))

Bruk vedlagte diagram i figur 7, figur 8, figur 9, figur 10 og figur 11 for IJVM til å løyse oppgåva.

a)
Kva type instruksjonar nyttar JMPC-signalet (figur 7)?
Svar: Branch instruksjonar (conditional branch) JMPC signalet blir brukt for å kunne gi to forskjelige mulige programstiar i microprogrammet for branch instruksjonar. (betingelsar for branch er gitt av N og Z flagga).

b)
Utifrå tilgjengeleg informasjon. Er IJVM mest lik ein RISC- eller CISC-type prosessor? Forklar kort.
Svar: CISC, standar argument, spesialregister framfor mange generelle, microprogramert kompliserte instruksjonar (multi cycle),m ikkje load/store arkitektur, etc alle gode argument.
c)
OpCode i IJVM er 8-bit. Kvifor nyttar då IJVM 9-bit til adressering av control store?

Svar: 9. bit blir brukt i branch microinstruksjonar. Viss branch neste microinstruksjon i 1xx, ved ikkje branch, neste microinstruksjon i 0xx i control store. 9. bit gir då muligheit for to forskjellige programstiar i microprogram for branch instruksjonar.

d)
I control store for IJVM vist i figur 6 ligg følgjande mikroinstruksjonar for den tenkte instruksjonen ADDaLotSUB:

MI 1: H = OPC
MI 2: H = TOS + H
MI 3: H = CPP + H
MI 4: H = LV + H
MI 5: OPC = SP - H

i) Oppgje mikroinstruksjonar for ADDaLotSUB. Sjå vekk frå Addr- og J-felta i mikroinstruksjonsformatet. Angi korrekte bit for ALU, C, Mem og B gitt i figur 8.
Svar:
M1: ALU = B, C = H, B= OPC
M2: ALU = A + B, C = TOS, B = H
M3: ALU = A + B, C = CPP, B = H
M4: ALU = A + B, C = LV, B = H
M5: ALU = B – A, C = SP, B = OPC

ii) Forklar korleis ADDaLotSUB kan endrast for å få ein meir effektiv instruksjon når mikroarkitekturen i figur 6 endrast til mikroarkitekturen i figur 11.
Svar: **Eksempel**
M1: H = TOS + OPC
M2: H = CPP + H
M3: H = LV + H
M4: OPC = SP - H

iii) ADDaLotSUB utføres med følgjande registerverdiar H = 0x00 00 00 FF, OPC = 0x00 00 00 01, TOS = 0x00 00 00 02, CPP = 0x00 00 00 03, LV = 0x00 00 00 04 og SP = 0x00 00 00 0A. Vil utføringa påverke Z-flagget eller N-flagget? Forklar kort.

Svar: OPC = SP – H: 0x00 00 00 0A - 0x00 00 00 0A gir 0 som vil gi Z(zero) = 1, N(negativ) = 0,

Oppgåve 4 Instruksjonssett arkitektur (ISA) (25 % (a: 5 %, b: 5 %, c: 10 og 5 % på d))

SKYNETi42 er en svært enkel prosessor. SKYNETi42 har ein «load», ein «store», åtte ALU-instruksjonar og nokre spesialinstruksjonar, inkludert NOP-instruksjonen og tre flytkontrollinstruksjonar (flow control instructions). Instruksjonsformatet for instruksjonane er vist i figur 5. Figur 6 visar instruksjonssettet. Alle register og bussar er 32-bit. Det er 32 generelle register tilgjengeleg. Prosessoren har en Harvard-arkitektur. Bruk figur 5 og figur 6 til å løyse oppgåva.

a) SKYNETi42 har ingen subtraksjonsinstruksjon, men det er mulig å utføre subtraksjon på SKYNETi42. Kort forklaring for korleis (metode) og kvifor SKYNETi42 med sitt instruksjonssett kan utføre subtraksjon.
Svar: ToarKomplement. A – B, B, (A + ~b + 1), gir A – B. Prosessoren har ADD, INV, INC instruksjonar, eventuelt kan ein laste 1 som konstant med MOVC, har då alt som trengs for sub. Med toarkomplement.

b) Følgande psaudokode for en SKYNETi42 versjon med samleband kjøres:

```
        :
ADD R1, R1; R2
ADD R1, R1; R3
ADD R1, R1; R4
        :
```

Er det avhengigheitar i koden som kan bli problematisk når koden køyrer på ein SKYNETi42 mikroarkitektur som nyttar samleband? Angi eventuelle avhengigheitar.
Svar: RAW, WAR for R1 for dei to siste instruksjonane (kva som skjer før I koden er ukjent).

c)
R0 har følgande verdi: 0xFFFF 0000, R16 har følgande verdi: 0x0001 0009, R17 har følgande Verdi: 0xFFFF 0010, R18 har følgande verdi: 0x0001 0003 og R19 har følgande Verdi: 0xFFFF EFFF. I dataminnet ligger følgande data frå adresse 0xFFFF 0000:

Adresse Data
0xFFFF 0000: 0xFF FF F0 00
0xFFFF 0001: 0x00 00 00 02
0xFFFF 0002: 0x00 00 00 03
0xFFFF 0003: 0x00 00 00 04
0xFFFF 0004: 0x00 00 00 00
0xFFFF 0005: 0x00 00 00 05
0xFFFF 0007: 0x00 00 00 06
0xFFFF 0008: 0x00 00 00 07
0xFFFF 0009: 0x00 00 00 08
:
0xFFFF 0010: 0x00 00 00 01
0xFFFF 0011: 0x00 00 00 02
0xFFFF 0012: 0x00 00 00 03
0xFFFF 0013: 0x00 00 00 04
0xFFFF 0014: 0x00 00 00 42
:
0xFFFF EFFF: 0x00 00 00 42
0xFFFF F000: 0x00 00 00 03

Følgande psaudokode er en del av et større program. Kodesnutten starter på adresse 0000 FFFD i programminnet. Svar på spørsmåla ut frå tilgjengelig informasjon.

1 0x0000 FFFD: MOVC R10, 0x0000;
2 0x0000 FFFE: MOVC R9, 0x0000;
3 0x0000 FFFF: LOAD R2, R0;
4 0x0001 0000: LOAD R1, R2;
5 0x0001 0001: CMP R9, R1;
6 0x0001 0002: BZ; R16
7 0x0001 0003: LOAD R8, R17;
8 0x0001 0004: ADD R9, R8; R9;
9 0x0001 0005: DEC R1, R1
10 0x0001 0006: INC R17, R17;
11 0x0001 0007: CMP R1, R10;
12 0x0001 0008: BNZ R18;
13 0x0001 0009: STORE R9, R19;

Forklar kva som skjer i koden. Kva verdi vil være lagra i minneadresse 0xFFFF EFFF etter at koden har kjørt?

Svar: Koden utfører følgjande med desse verdiane som resultat:

Inst nummer	Resulta
1	R10 = 0
2	R9 = 0
3	R2 = FF FF F0 00
4	R1 = 03
5	R9(0) cmp R1(03) Z = 0
6	BZ (Z = 0)
7	R8 = 01
8	R9 = 1 + 0, R9 = 1
9	R1 = 2
10	R17 = FF FF 00 11
11	R1 cmp R10, Z = 0
12	BNZ, Z = 0, Hopp (R18 inst 7)
7	R8 = 2
8	R9 = 3
9	R1 = 1
10	R17 = FF FF 00 12
11	R1 cmp R10, Z = 0
12	BNZ, Z = 0, Hopp (R18 inst 7)
7	R8 = 3
8	R9 = 6
9	R1 = 0
10	R17 = FF FF 00 13
11	R1 cmp R10, Z = 1
12	BNZ, Z = 1, ikkje hopp Pc = PC +1
13	R9 = 6, R19 = FF FF EF FF, 6 lagra I adrlokasjon FF FF EF FF

d)

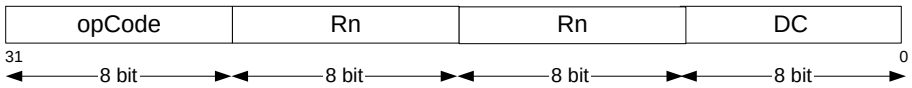
I ei anna køyring blir alle verdiar (register og minne) sett tilbake til gitt utgangspunkt med unntak av minneadresse 0xFFFF F000 som no blir gitt verdien: 0xFFFF 0004 (0xFFFF F000: 0x00 00 00 04). Kva verdi vil R9 ha viss programmet no blir køyrt på ny?

Svar: No vil løkka (inst 7 – 12) utført ein gong meir (inst 4: R1 = 4), R9 vil då få verdien 0A (6 + 4 .

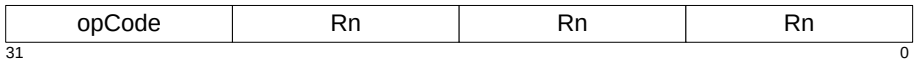
Load/store:

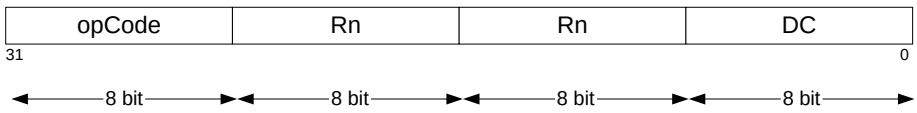
Load:

Store:



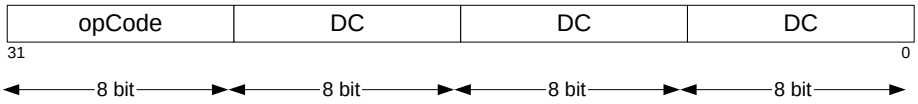
ALU:



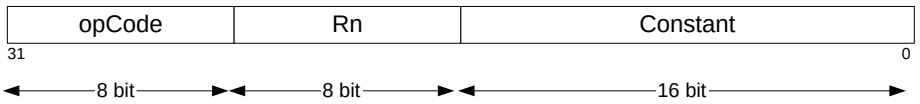


Spesial:

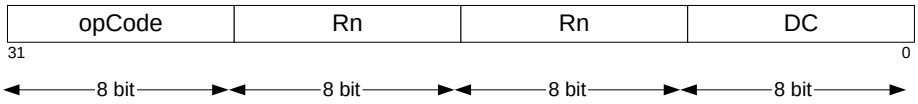
NOP:



MOVC:

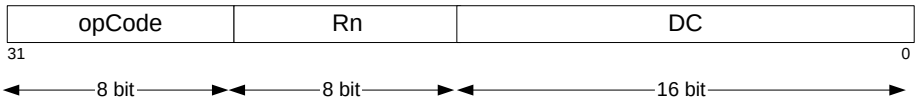


CP:

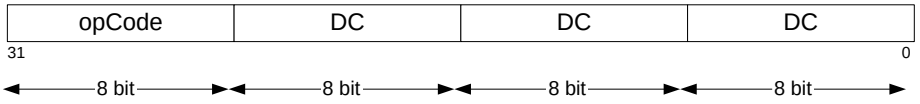


Flow control:

BZ/BNZ



RT



Rn: any user register, R0 - R31
DC: Don’t care: any data memory location

Figur 5 Instruction format SKYNETi42

Instructions set:

LOAD: Load data from memory.

load Ri, Rj Load register Ri from memory location in Rj.

STORE: Store data in memory.

store Ri, Rj Store register Ri in memory location in Rj.

ALU: Data manipulation, register–register operations.

ADD Ri, Rj, Rk ADD, $Ri = Rj + Rk$. Set Z-flag if result =0.
NAND Ri, Rj, Rk Bitwise NAND, $Ri = \overline{Rj \cdot Rk}$. Set Z-flag if result =0.
OR Ri, Rj, Rk Bitwice OR, $Ri = Rj + Rk$. Set Z-flag if result =0.
INV Ri, Rj Bitwice invert, $Ri = \overline{Rj}$. Set Z-flag if result =0.
INC Ri, Rj Increment, $Ri = Rj + 1$. Set Z-flag if result =0.
DEC Ri, Rj Decrement, $Ri = Rj - 1$. Set Z-flag if result =0.
MUL Ri, Rj, Rk Multiplication, $Ri = Rj * Rk$. Set Z-flag if result =0.
CMP, Ri, Rj Compare, Set Z-flag if $Ri = Rj$

Special: Misc.

CP Ri, Rj Copy, $Ri < -Rj$ (copy Rj into Ri)
NOP Waste of time, 1 clk cycle.
MOVC Ri, constant Put a constant in register $Ri = C$.

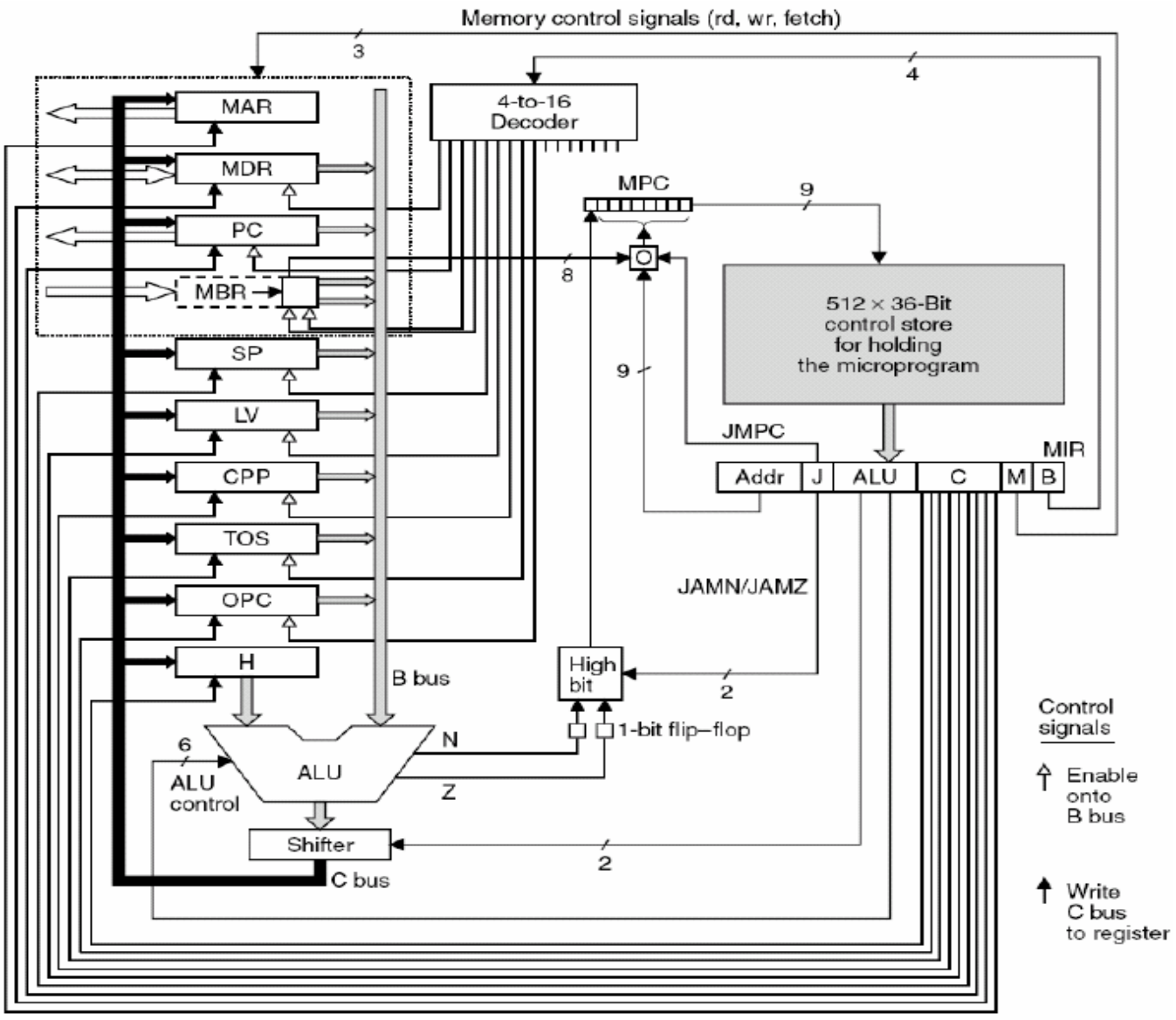
Flow control: Branch.

BZ, Ri Conditional branch on zero (Z-flag = 1) , $PC = Ri$.
BNZ, Ri Conditional branch on non zero (Z-flag = 0), $PC = Ri$.
RT Return, return from branch.

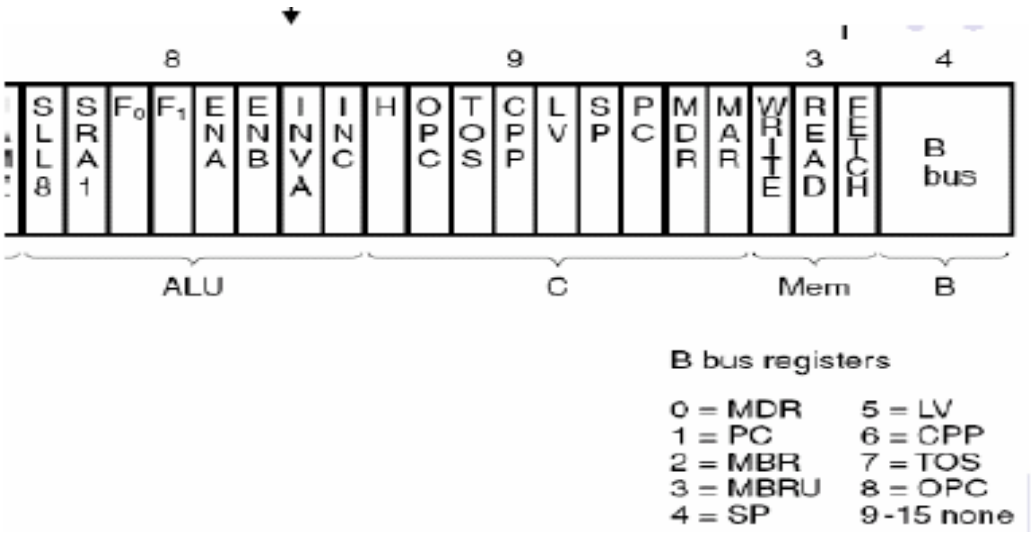
Ri, Rj and Rk: Any user register.
DC: Don't care.

Figur 6 Instruction set SKYNETi42.

Vedlegg IJVM



Figur 7 IJVM

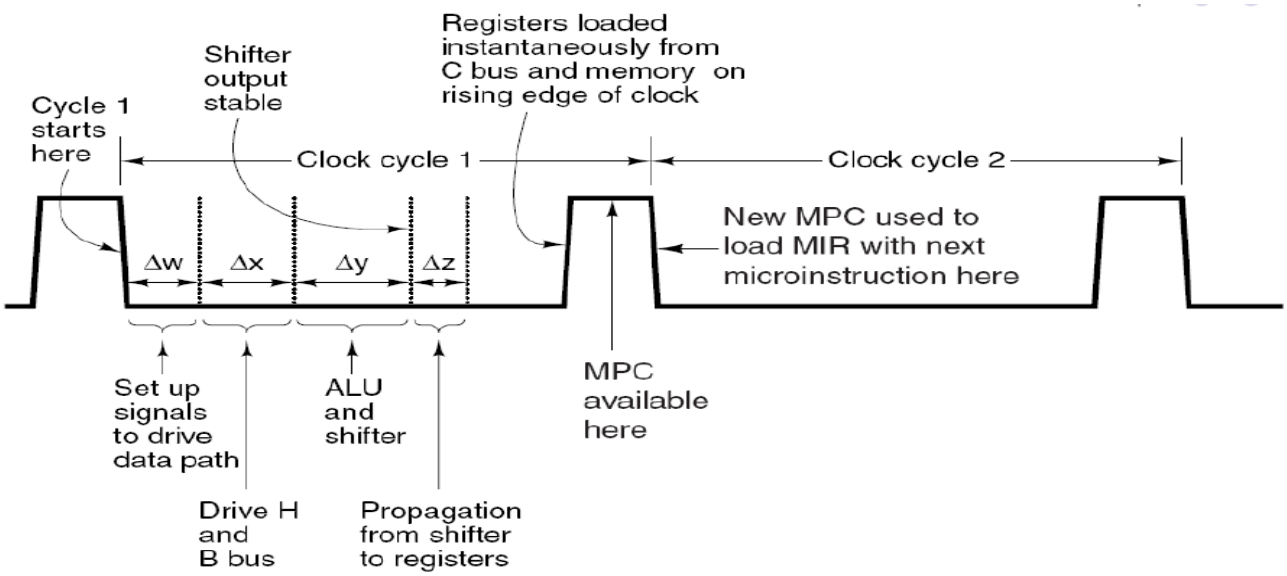


Figur 8 Microinstruction format.

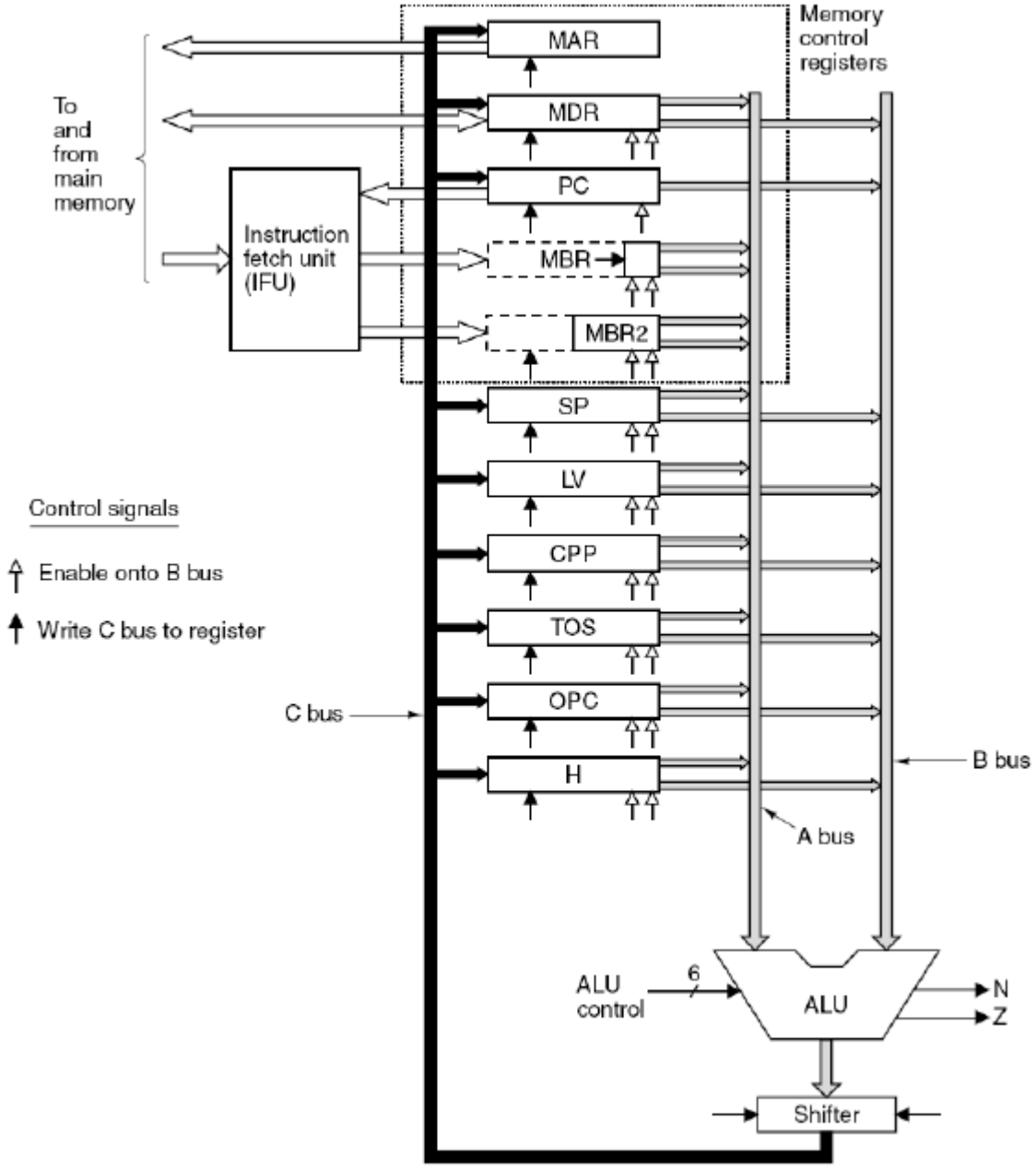
F_0	F_1	ENA	ENB	INVA	INC	Function
0	1	1	0	0	0	A
0	1	0	1	0	0	B
0	1	1	0	1	0	$\overline{A}$
1	0	1	1	0	0	$\overline{B}$
1	1	1	1	0	0	A + B
1	1	1	1	0	1	A + B + 1
1	1	1	0	0	1	A + 1
1	1	0	1	0	1	B + 1
1	1	1	1	1	1	B – A
1	1	0	1	1	0	B – 1
1	1	1	0	1	1	–A
0	0	1	1	0	0	A AND B
0	1	1	1	0	0	A OR B
0	1	0	0	0	0	0
1	1	0	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 9 ALU functions.



Figur 10 Timing diagram.



Figur 11 MIC 2.