NTNU
Kunnskap for ei betre verd

Institutt for datateknikk og informasjonsvitenskap

Eksamensoppgåve i TDT4160 datamaskiner og digitalteknikk

Fagleg kontakt under eksamen: Gunnar Tufte

Tlf.: 97402478

Eksamensdato: 11. desember 2017

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): X

Sidetal vedlegg: X

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 (20 % (a: 2,5 %, b: 7,5 %, c:.5 %, d: 2,5 og e: 2,5 %)

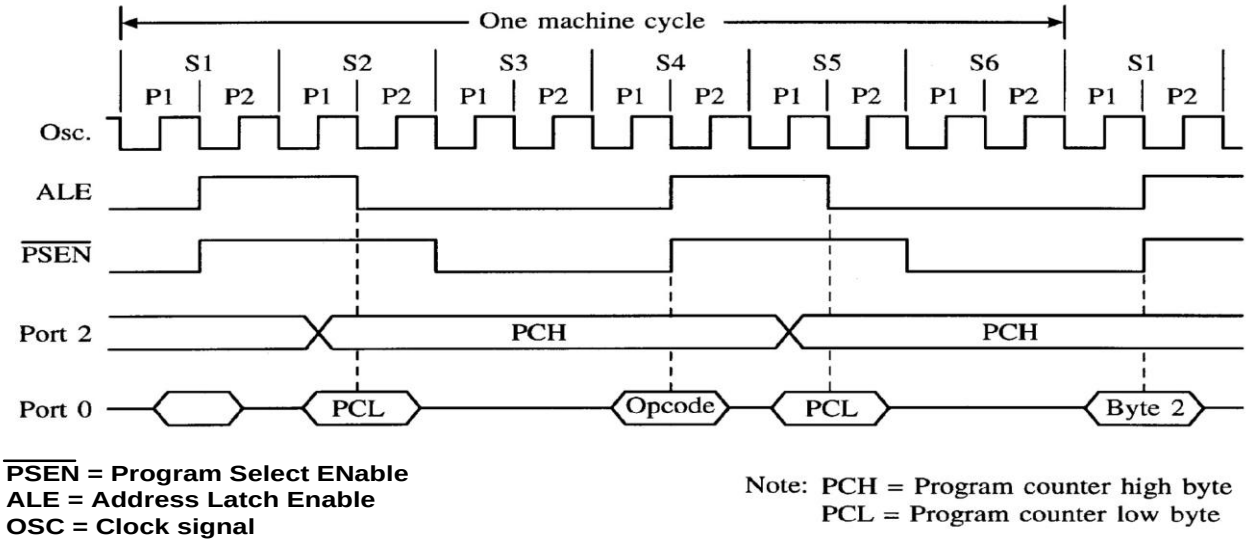
a)
Omgrepet «stored program computer» vart innført med von Neumann-arkitekturen. Kva ligg i dette omgrepet?
Svar: Program og data lagra i minne. Samme minne der det er maskina som held orden på om aksess er data eller instruksjonar. Maskina hentar instruksjonar til kontrolleining, data overføres mellom minne og ALU (utførandeeining). Sjå kapitel 1 figur 1.5.

b)
Figur 1 viser ein Mikrokontroller sitt grensesnitt mot eksternt minne. Busssignala i figuren angir lesing av programminne. Port 0 og Port 2 er 8-bit portar. Svar på følgjande spørsmål ut frå tilgjengeleg informasjon:

i) Er dette ein synkron eller asynkron bussoverføring? Forklar kort.
Svar: Synkron, signal synkron med klokke

ii) Kva er storleiken (maks adresserom) på programminnet? Forklar kort.
Svar: 2^16, 2 8-bit portar gir 16 bit adr buss. Det er multiplexa bussoverføring.

iii) Kva er bitbredden på programminne? Forklar kort.
Svar: 8-bit, opCode på port 0 (8-bit).



Figur 1 Bussgrensesnitt.

c)

i) Kva vil det seie at ein prosessor er «superscalar»?
Svar: Replisering av utførande eining(ar) i samlebånd.

ii) Kan ein innføre superscalaritet utan å påvirke ISA?
Svar: Ja, men krever ekstra logikk for å handtere instruksjonsutføring.

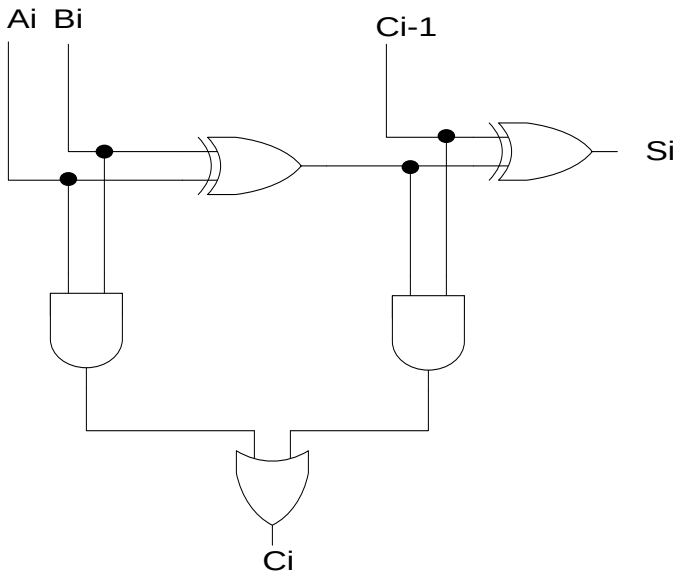
d) Kva kjenneteiknar en prosessor med Harvard-arkitektur? Forklar kort.
Svar: Separat minneområde for data og program, seperat buss (adressbuss for databuss og adressebuss for prog buss).

e) «Latency hiding» er eit viktig konsept i moderne datamaskinarkitektur. Er cache ein del av dette konseptet? Forklar kort kvifor, eventuelt kvifor ikkje.
Svar: Cache er ein del av konseptet, ved å redusere gjennomsnitleg aksesstid «skjuler ein» latency.

Oppgåve 2 Digitalt Logisk Nivå (20 % (a: 4 %, b: 7 %, c: 7 % og 2 % på d))

a)

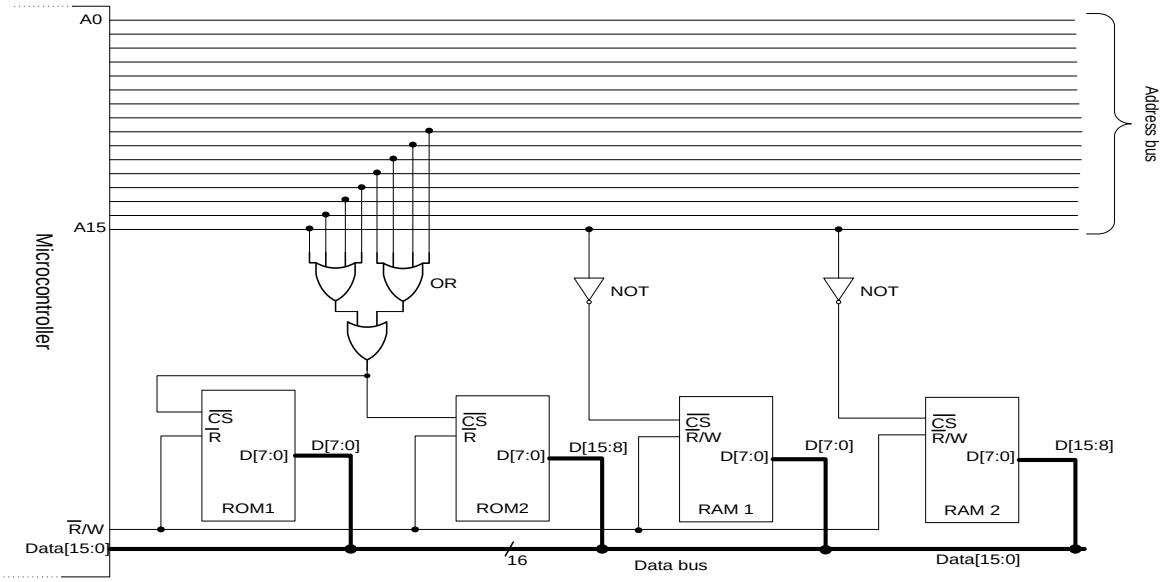
Figur 2 viser logikken til ein av funksjonane i ein 1 bit ALU. Kva funksjon?



Figur 2 Logikk for 1-bit ALU-funksjon.

Svar: Full adder

b)
I eit innvevd system (embedded system) nyttast det ein 16-bit mikrokontroller. Figur 3 viser det eksterne bussgrensesnittet med adressedekodingslogikk for mikrokontrolleren. Det er to ROM-brikkar for program og to RAM-brikkar. Alle einingane nyttar eit aktivt lågt (logisk "0") CS (Chip Select)-signal.



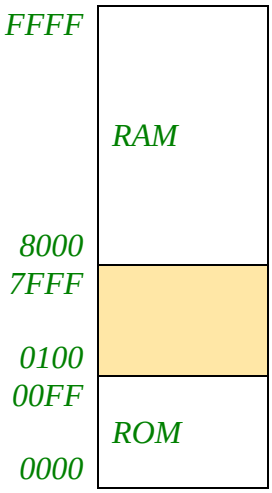
Figur 3 Address decoding.

i) Finn adresseområde for RAM og ROM. Eventuellt ledig minneområde og overlapp.

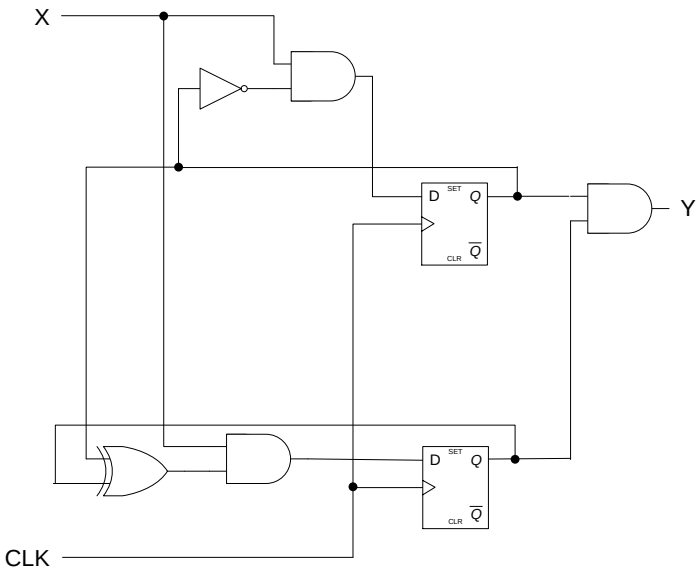
Svar: ROM 0000 – 00FF, RAM 8000 – FFFF, det er 16 bit databuss så ingen kollisjon

ii) Teikn minnekart utfrå adresseområde

Svar: Teikn som funnet, pass på minne har 16-bit bredde.



c)
Figur 4 viser ei FSM (Finite State Machine). Datavippe med D0 og Q0 er øverst. Datavippe med D1 og Q1 nederst. .



Figur 4 Finite State Machine (FSM) 1.

Finn dei logiske uttrykka for D0 og D1 (excitation equation). Angi neste tilstands uttrykka (next state equations) og lag transisjonstabell (next-state-table) for kretsen som viser oppførselen til denne FSM-en og utgangssignalet Y.

Svar:

Logikk uttrykk:

D0 = $\overline{Q0}$ and X

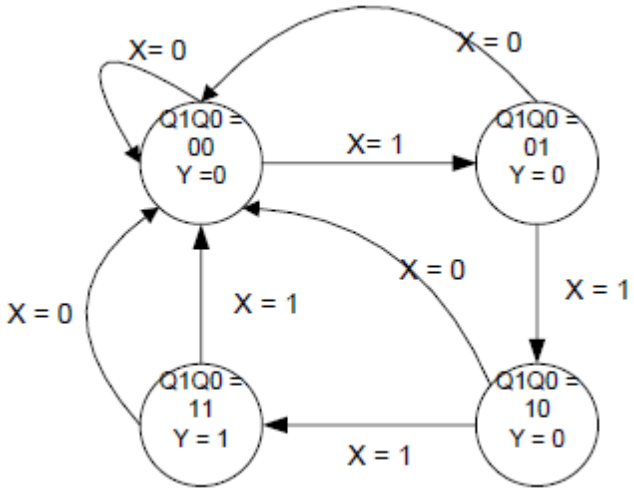
D1 = (Q0 xor Q1) and X

Q0 (nxt) = D0 = Q0(nxt) = D1

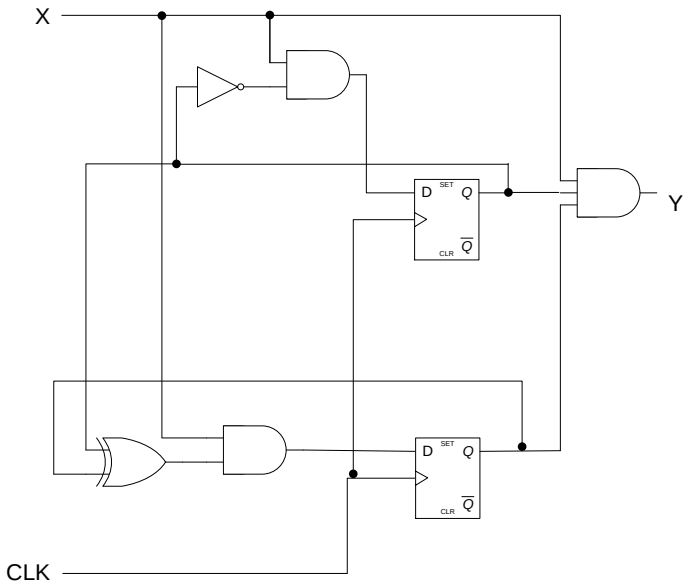
Y = Q0 and Q1

Tabell	X = 1			X = 0		
Q1Q0	Q1(nxt)	Q0 (nxt)	Y	Q1(nxt)	Q0 (nxt)	Y
0 0	0	1	0	0	0	0
0 1	1	0	0	0	0	0
1 0	1	1	0	0	0	0
1 1	0	0	1	0	0	0

Tilstandsdiagram (om ønskelig, ikke spurt om i oppgaven):



d)
Tilstandsmaskina i figur 4 blir endra til vist FSM i figur 5. Korleis påverkar endringen oppførselen til utgongssignalet Y?



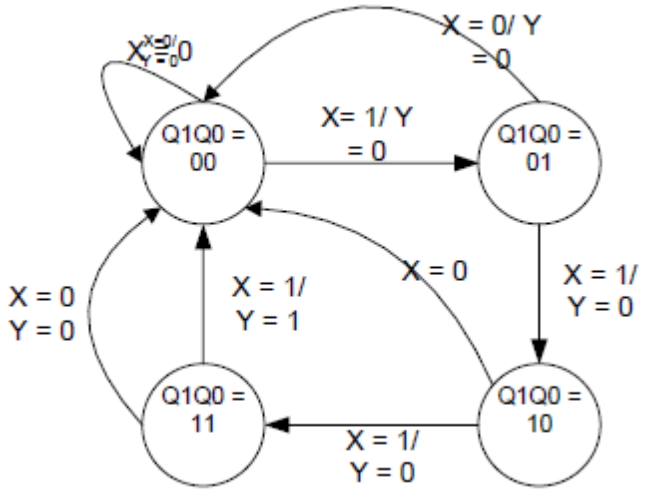
Figur 5 Finite State Machine (FSM) 2.

Svar: Har no ei Mealy maskin, Y er ikkje lenger synkron med klokka.
Tilstandsmaskinen er ein teller, Y er 1 når talet 11 er nådd. X er eit reset-signal (aktivt lågt) som kan restarte teljinga. Huks utgangen Y skiftar her med ein gong når Y går frå 1 til 0 (asynkron reset).
Dette er ei Mealy maskin. State register og inngang blir brukt til dekoding av otgangssignal.

Ny transisjonstabell og tilstandsdiagram: (treng berre forklaring på kva som skjer)

D0 = $\overline{Q0}$ and X
D1 = (Q0 xor Q1) and X
Q0 (nxt) = D0 = Q0(nxt) = D1
Y = (Q0 and Q1 and X)

Tabell	X = 1			X = 0		
Q1Q0	Q1(nxt)	Q0 (nxt)/	Y	Q1(nxt)	Q0 (nxt)	/Y
0 0	0	1	0	0	0	0
0 1	1	0	0	0	0	0
1 0	1	1	0	0	0	0
1 1	0	0	1	0	0	0



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

Bruk vedlagte diagram i figur 9, figur 10, figur 11 og figur12 for IJVM til å løse oppgavene.

a)

Komponenten 4-to-16 Decoder i figur 9 er brukt til å kontrollere kva som skal liggje på B-buss. For C-Bussen er separate bit brukt til å kontrollere kvart register. Kvifor?

Svar: For B-buss dekoderen reduserar antal bit i microinstruksjon dette kan gjerast sidan kunn eit register kan leggjast på B-bus omgongen. For C-buss seperate bit gjer at fleire register kan oppdaterast samstundes.

b)

For mikroarkitekturen i Figur 9. Lag mikroinstruksjon(ar) som utfører logisk NOR. Data ligg i TOS-registeret og LV-registeret (*TOS NOR LV*). Resultatet skal lagrast i H-registeret og MDR-registeret.

Sjå vekk frå Addr- og J-felte i mikroinstruksjonsformatet. Angi korrekte bit for ALU, C, Mem og B gitt i figur 10.

Svar:

H = TOS	*ALU: 010100 (B),*	*C: 100000000 (H),*	*B: 0111 (TOS)*
H = H or LV	*ALU: 011100 (A OR B),*	*C: 100000000 (H),*	*B: 0101 (LV)*
H = MDR = not H	*ALU: 011010 (NOT A),*	*C: 100000010 (H og MDR),*	*B: 0*

Andre kombinasjonar også mogleg.

c)

For mikroarkitekturen i Figur 9. Under utføring av ein instruksjon ligg verdien 0xA3 i MPC (MPC peikar på adresse 0xA3 i control store).

Innhaldet i MDR-registeret er 0xAAAA AAAA.
Innhaldet i H-registeret er 0x5555 5555.
Innhaldet i LV-registeret er 0xA5A5 0000.

Tabellen under viser to utsnitt av innhaldet i control store.
Kva skjer under resten av instruksjonsutføringa? Forklar kort kva mikroinstruksjon gjer.
Kva verdi vil MDR-registeret ha når MPC får verdien 0x00?

Control store Address	Next_Adr	JMPC	JAMN	JAMZ	SLL8	SRA1	F0	F1	ENA	ENB	INVA	INC	H	OPC	TOS	CPP	LV	SP	MDR	MAR	Write	Read	FETCH	B Buss			
0A3	010100100	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0A4	010100101	0	0	1	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	1
0A5	010100110	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
0A6	000000000	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
													:														
1A5	110100110	0	0	0	0	0	1	1	0	1	0	1	0	0	0	0	1	0	0	0	0	0	0	0	1	0	1
1A6	110100111	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	1
1A7	000000000	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
1A8	110101001	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	1
													:														

Svar:
0A3 J = 0, ALU = B, C = LV, B = MDR → LV = MDR = 0xAAAA AAAA → MPC = 0xA4
0A4 JZ = 1, ALU = A and B, C = LV, B = LV → LV = H and LV = 0 (Z = 1) → MPC = 0x1A5
1A5 J = 0, ALU = B + 1, C = LV, B = LV → LV = LV + 1 = 1 → MPC = 1A6
1A6 J = 0, ALU = B, C = MDR, B = LV → MDR = LV = 1 → MPC = 1A7
1A7 J = 0, ALU = 0, C = 0, B = none → ***MPC = 0x00*** *og* ***MDR = 1****.*

Oppgåveteksten var litt uklar om kva for mikroinstruksjonar som skulle forklarast: alle i tabellen eller berre dei som utførast. Meininga var det siste, men begge løysningar gir full score.

Oppgåve 4 Instruksjonssett arkitektur (ISA) (20 % (a: 2,5 %, b: 2,5 %, c: 10 og 5 % på d))

RaM 007 er en svært enkel prosessor. RaM 007 har én «load», én «store», åtte ALU-instruksjoner og nokre spesialinstruksjoner, inkludert NOP-instruksjonen og tre flytkontrollinstruksjoner (flow control instructions). Instruksjonsformatet for instruksjonene er vist i figur 6, figur 7 viser instruksjonssettet. Alle register og busser er 32-bit. Det er 32 generelle register tilgjengelig. Prosessoren har en Harvard-arkitektur. Bruk figur 6 og figur 7 til å løse oppgava.

a)
Nyttar RaM 007 fast instruksjonslengde?
Svar: Ja, alle instruksjonar som hentast frå P-minne er 32 bit.
b)
Gi eit eksempel på ein RaM 007 instruksjon som nyttar «immediate addressing».
Svar: MOVC
c)
R0 har følgjane verdi: 0x0000 FFFF, R8 har følgjane verdi: 0xFFFF 0000, R11 har følgjane verdi: 0x0000 0000, R12 har følgjane verdi: 0x0000 0003 og R13 har følgjane verdi 0x0001 0005. I dataminnet ligger følgende data fra adresse 0xFFFF 0000:

Adresse	Data
0xFFFF 0000:	0x00 00 00 01
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

Følgende psaudokode er en del av et større program. Kodesnutten starter på addresse 0000 FFFF i programminnet. Svar på spørsmåla ut fra tilgjengelig informasjon.

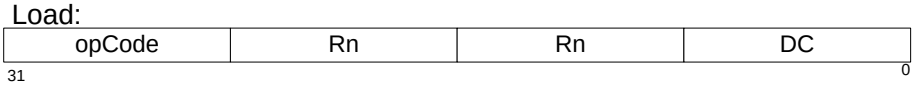
	Svar:
0x0000 FFFF: LOAD R1, R8;	*R1 = [R8] = 0x00 00 00 01 (R8 = 0xFFFF 0000)*
0x0001 0000 ADD R28, R1, R1;	*R28 = R1 + R1 (ved R1 = 0, Z = 1)*
0x0001 0001: BZ R13;	*Viss Z = 1: Hopp til adr R13: 0x0001 0005*
0x0001 0002: ADD R11, R1, R11;	*R11 = R1 + R11: R11 = 1 … + 2 … + 3 … + 4 … + 0*
0x0001 0003: INC R8, R8;	*R8 = R8 + 1: R8 = 0xFFFF 0001 … 0xFFFF 0002 …*
0x0001 0004: BNZ R0;	*Viss Z = 0 (R8 ikkje 0 i forrige inst) hopp til 0x0000 FFFF*
0x0001 0005: STORE R11, R8;	*Lagre R11 på adresse gitt av R8. (0xFFFF 0005)*

Forklar hva som skjer i koden. Hvilken verdi vil R11 ha etter at koden har kjørt?
Svar: Snutten vil køyre i loop 5 gonger, 5. gong lesest adr: 0xFFFF 0004 (som inneheld 0) og inst BZ R13 vil hoppe ut av loopen. R11 vil då ha fått verdien ***0d10 = 0xA****.*

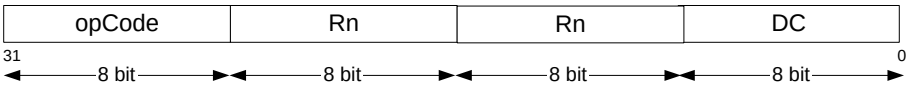
e)
I ein anna køyring blir alle verdiar (register og minne) sett tilbake til gitt utgangspunkt med untak av R8 som no blir gitt verdien: 0xFFFF 0004. Kva verdi vil R11 ha viss programet no blir køyrt på ny?
Svar: 1. instruksjon vil laste 0x0 inni R1, R1 + R1 = 0, Z =1 og BZ til 0x0001 0005, med uendra R11 (R11 = 0x0), 0x0 vil bli lagra på adr: 0xFFFF 0000, snutt ferdig.

Load/store:

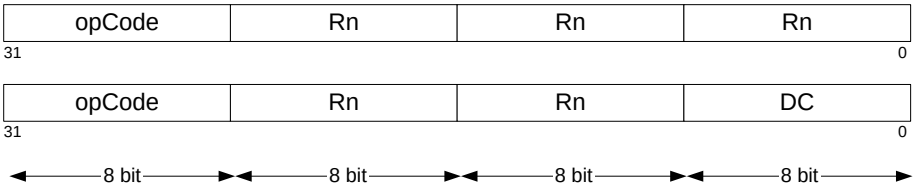
Load:



Store:

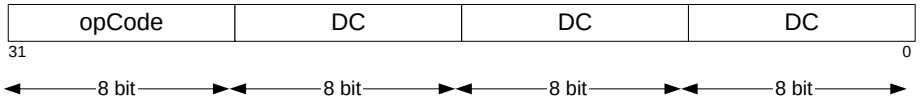


ALU:

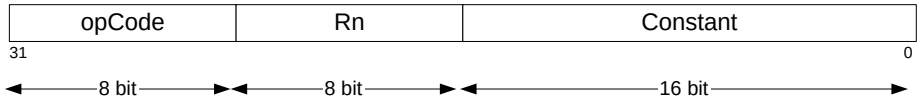


Spesial:

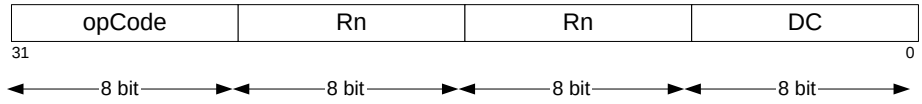
NOP:



MOVC:

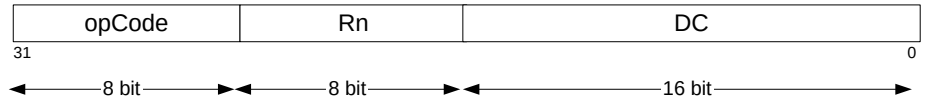


CP:

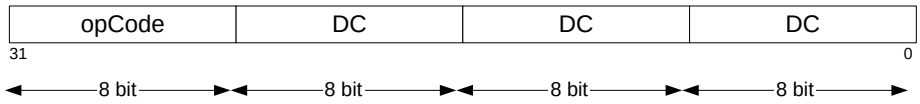


Flow control:

BZ/BNZ



RT



Rn: any user register, R0 - R31

DC: Don’t care: any data memory location

Figur 6 Instruction format Ram 007

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.

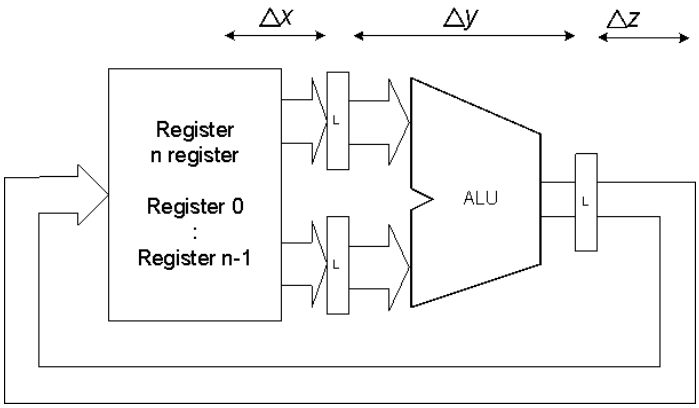
BZ, Ri Conditional branch on zero (Z-flag = 1) , $PC = Ri$.
BNZ, Ri Conditional branch on non zero (Z-flag = 0), $PC = Ri$.
RT Return, return from branch.

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

Figur 7 Instruction set RaM 007.

Oppgåve 5 Yting (20 % (a: 5 %, b: 10 %, 2,5 % på c og 2,5 på d))

a)
Figur 8 viser ein data path som inkluderar ein registerbank, ein ALU og tre latch-ar. $\Delta x = 10ns$, $\Delta y = 25ns$ og $\Delta Z = 12ns$. Utfrå tilgjengeleg informasjon, kva parameter begrensar klokkefrekvensen, og kva er maksimum klokkefrekvens? Forklar svaret kort.
Svar: Lengste trinn, her kan ikkje klokkeperioden være kortare enn Δy.
Maks klokkefrekvens blir 1/25ns = 40Mhz.



Figur 8 Data path.

b)
1) Figur 9 viser den orginale IJVM. Figur 13 viser ein versjon av IJVM der det er innført ein ekstra buss (A bus) og ein Instruction Fetch unit for å auke ytelsen.

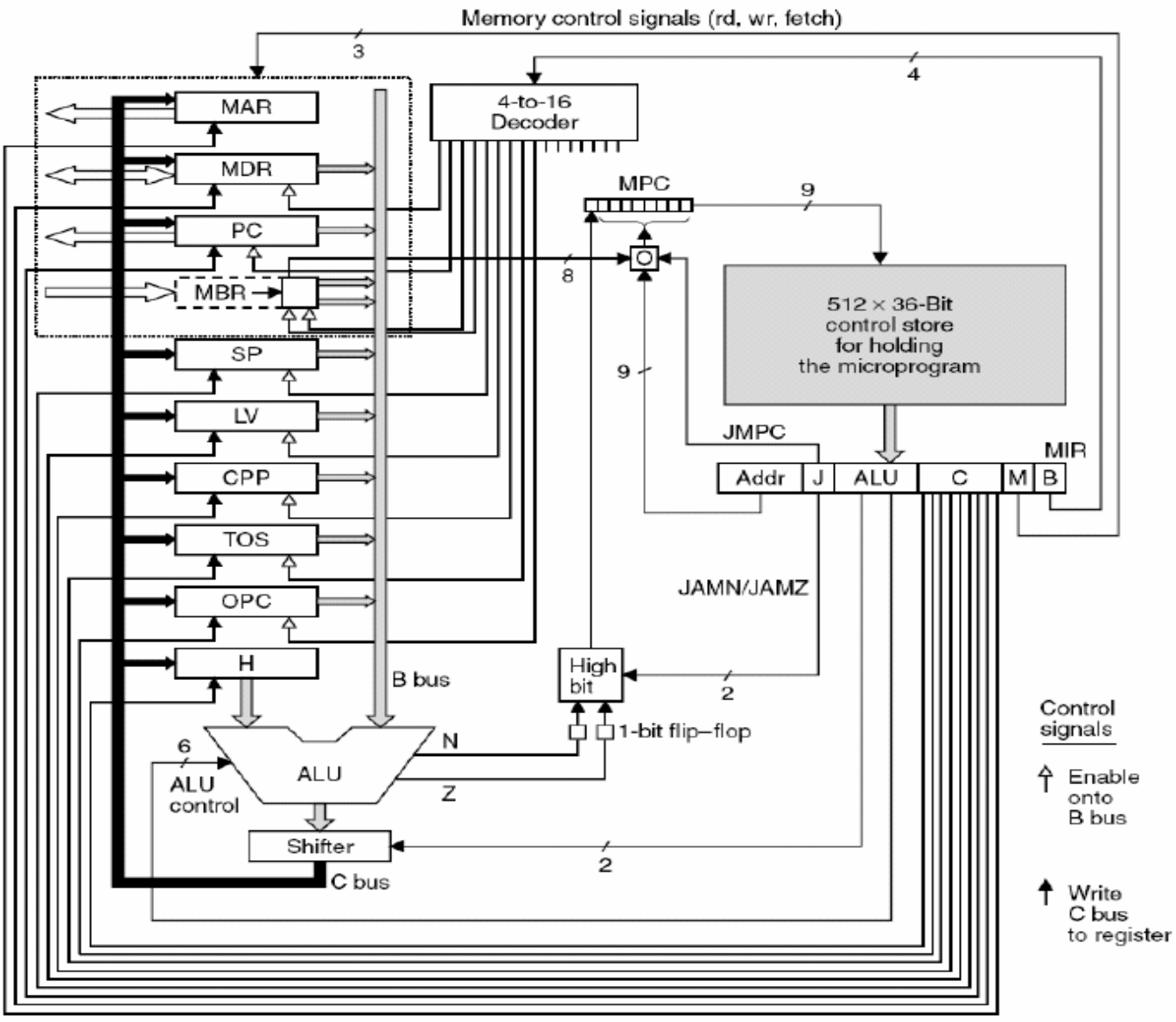
i) Korleis påverkar innføringa av den ekstra bussen (A bus) instruksjonsutføringa? Forklar kort.
Svar: ALU har tilgong til to vilkårlege register, ikkje nødvendig med «mellomlagring» i H-registeret. Ferre microinstruksjonar i instruksjon, raskare instruksjonsutførelse.

ii) Må mikroprogramma for instruksjonane i control store endrast for å kunne nytte den ekstra bussen? Forklar kort.
Svar: Ja, må legge til felt for A-Buss og skrive om instruksjonar frå å nytte H-registeret til å nytte begge bussane.

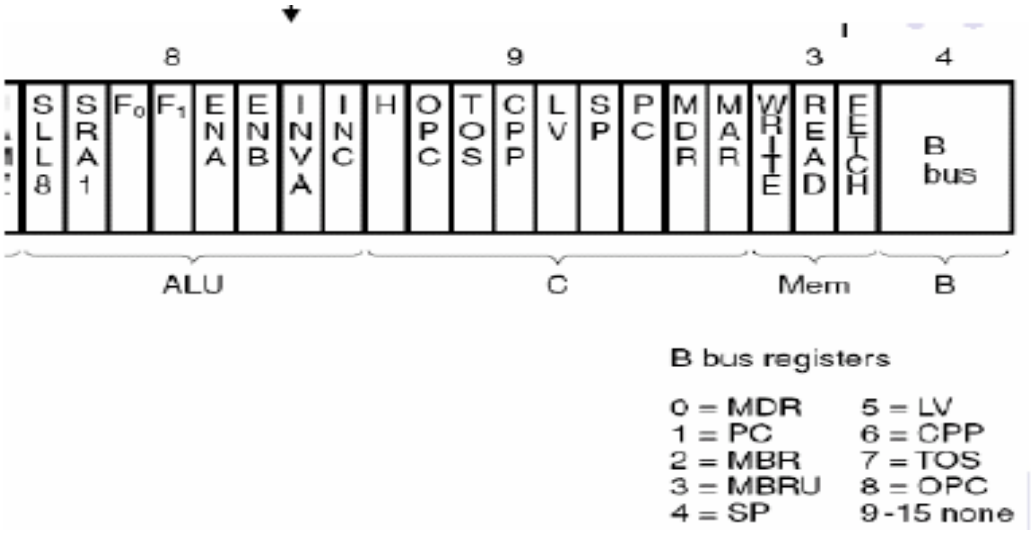
c)
Kva utvikling skisserer «Moore's law»?
Svar: Kort, eksponensiell vekst av antal gates på ein brikke (transistorar og OK).
Frå 1970 talet dobbling kvart år, i nyare tid dobbling ca kvar 18 månad.
Ekstra info: Moore's law baserar seg på kva som er lønnsomt å produsere. Altså det er mulig å ha eksponensiell vekst sidan prosessteknologien er økonomisk lønsom. (I dag er me nær at det er for dyrt å gå ned i dimensjon, neste prosess er ikkje økonomisk lønsom. Samt me nærmar oss også dei fysiske grensene for kva som er mulig)

d)
Kva parameter kunne ein redusere for å få ned energiforbruket ved å gå frå prosessorar med ein kjerne med djupe samlebånd (pipelines) til CMP-baserte prosessorar?
Klokkefrekvensen kunne senkast (eller være uendra, liten vekst) sidan ein går frå ILP til prosessornivå parallellitet.

Vedlegg IJVM



Figur 9 IJVM

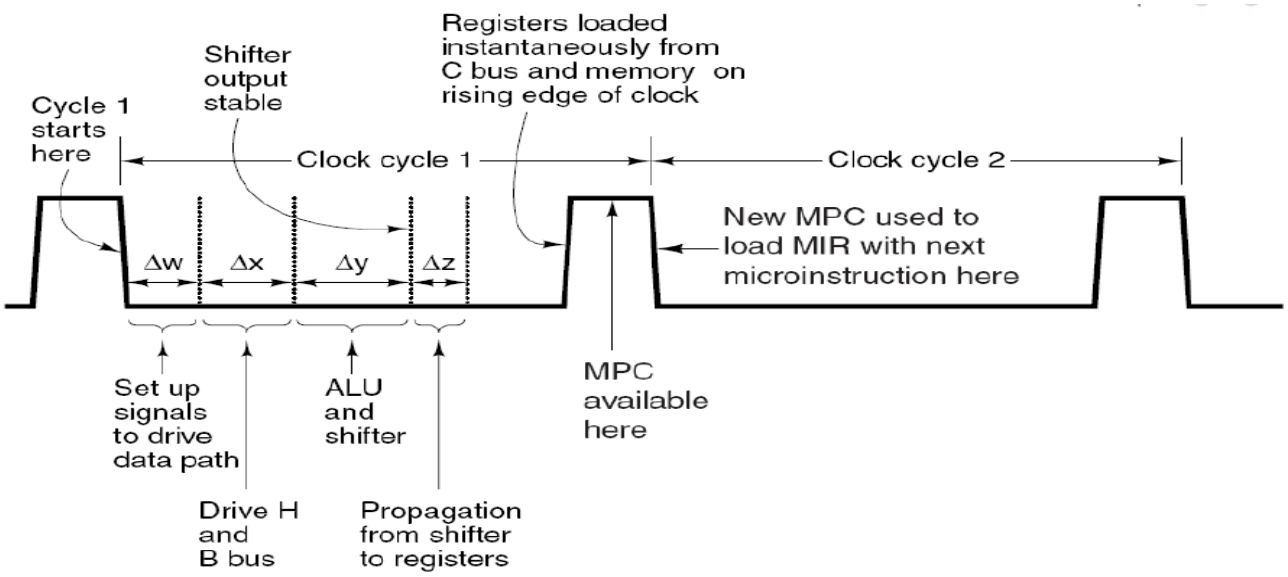


Figur 10 Microinstruction format.

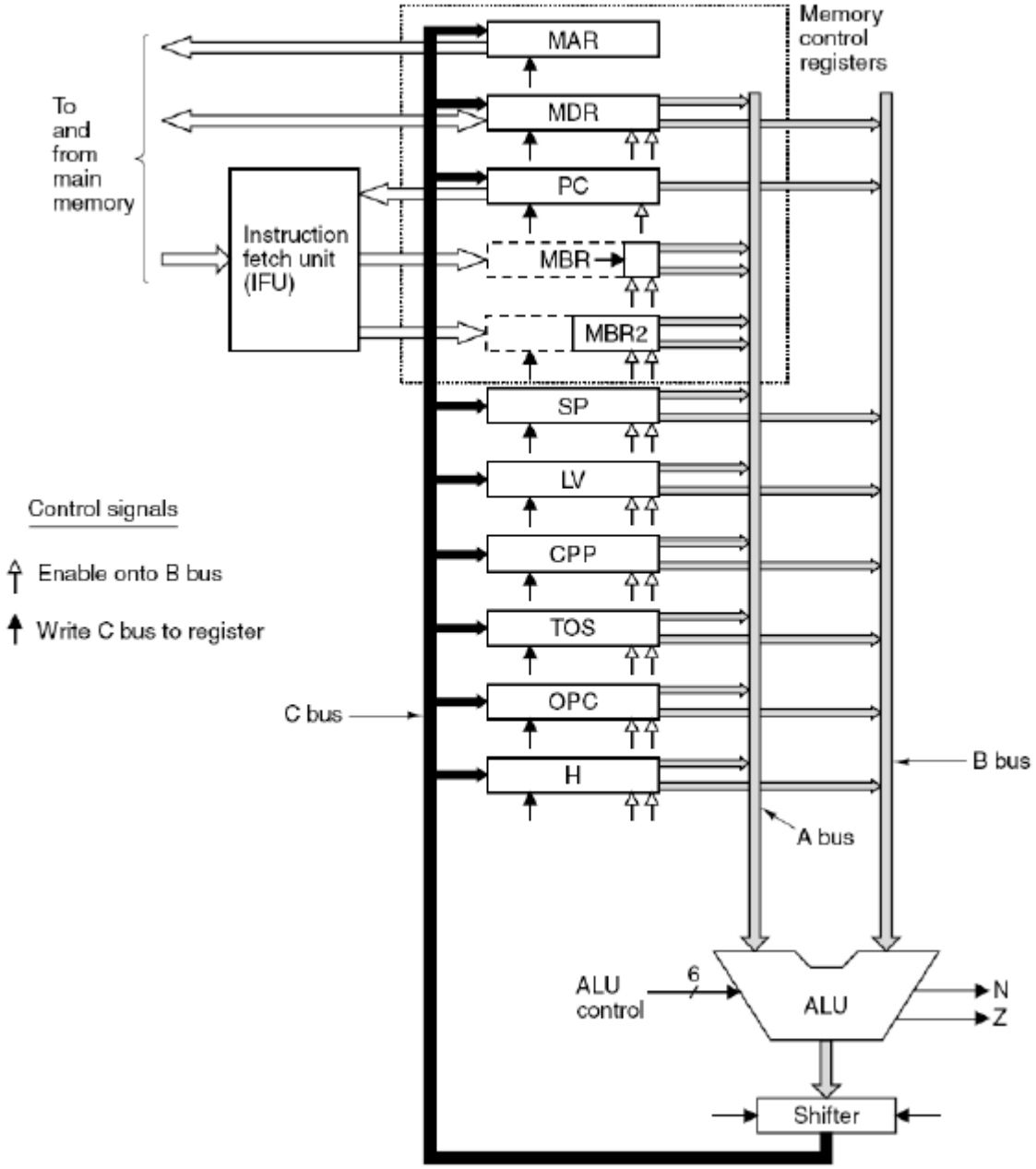
F_0	F_1	ENA	ENB	INVA	INC	Function
0	1	1	0	0	0	A
0	1	0	1	0	0	B
0	1	1	0	1	0	$\overline{A}$
1	0	1	1	0	0	$\overline{B}$
1	1	1	1	0	0	A + B
1	1	1	1	0	1	A + B + 1
1	1	1	0	0	1	A + 1
1	1	0	1	0	1	B + 1
1	1	1	1	1	1	B – A
1	1	0	1	1	0	B – 1
1	1	1	0	1	1	–A
0	0	1	1	0	0	A AND B
0	1	1	1	0	0	A OR B
0	1	0	0	0	0	0
1	1	0	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 11 ALU functions.



Figur 12 Timing diagram.



Figur 13 MIC 2.