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: 28 november 2016

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): 9

Sidetal vedlegg: 2

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)

Figur 1 viser ei prinsippskisse av von Neumann-datamaskiner. Relater enhetene i lista under (i – iii) til minne (memory), kontroll (control unit) eller ALU (arithmetic logic unit) i Figur 1.

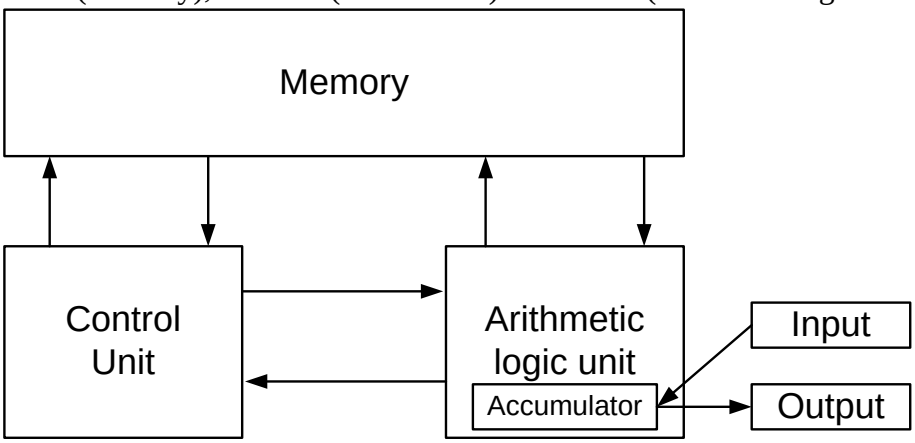
i) Register *Svar: ALU*

ii) mikroinstruksjonsminne (microprogram control store) *Svar: Control unit.*

iii) hurtigbuffer (cache) *Svar: Minne*

Svar:Fordeling bygger på von-Neumann arkitektur, sjå Lysark og bok.

til rett eining (minne (memory), kontroll (control unit) eller ALU (arithmetic logic unit)) i Figur 1.



Figur 1 Prinsippskisse av von Neumann-maskin.

b)

Forklar kort hva som ligg i omgrepa:

i) Instruksjonsnivå paralelitet (Instruction-Level Parallelism (ILP))

ii) Prosessornivå parallellitet (Processor-level Parallelism)

*Svar: **ILP**: fleire **instruksjonar i parallel** (pipeline) på **ein kjerne** (side 65-69).*
***Prosessor-level**) **fleire kjernar** som jobbar i paralell, **multiple eller single instruction stream**. Sjå også CMP-artike (bok side 65-69l.*

c)

Forklar kort forskjellen på programert I/O med travel venting (programmed I/O with busy waiting) og avbruddsdreven I/O (interrupt-driven I/O).

*Svar: **Busy waiting**: prosessor må lese status (f.eks. i ein loop (opptatt med å loope)) til I/O eining.Prosessor vil då være opptatt med å vente, får ikkje kjørt anna kode enn loop ved venting.*
***IRQ:** I/O eining kan gi IRQ-signal til prosessor for å få utført I/O oppgåva. Prosessor treng ikkje overvåke i SW (IRQ code for eining utførast når IRQ signal gis, prosessor kan utføre anna kode når I/O handtering ikkje er nødvendigt.. Sjå slides og boka f.eks 109 og 414)*

d)

Forklar kort kva som ligg i omgrepet lokalitet (locality)) i samband med minnesystem, f. eks. hurtigbuffer og virtueltminne (paging).

***Lokalitet i tid og rom (**Temporal and spatial locality) :*
***Tid:** sansynlig at data/inst brukt vil bli brukt igjen.*
***Rom:** sansynlig at data/inst aksesert har nærligande data/inst som kjem til å bli aksessert.brukes til å få ein effektiv (låg aksesstid) avbilding av minne i f.eks cache. Sjå boka side 83 og 4.5.1 + slides*

e)

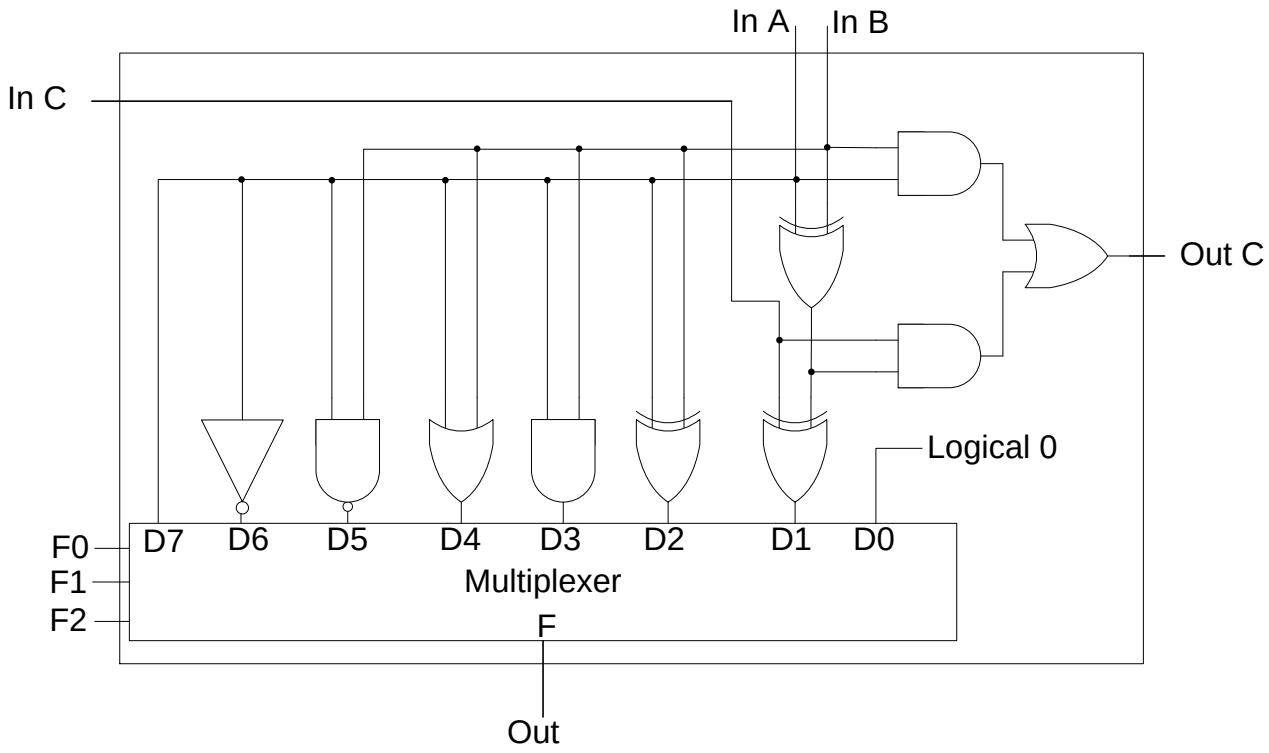
Forklar kort forskjellen på homogene (homogeneous) og hetrogene (heterogeneous) Multi Processorar (CMP).

Svar: homogene like kjerner, hetrogene ulike. Sjå f.eks CMP artikel for forklaring.

Oppgåve 2 Digitalt Logisk Nivå (20% (a: 6%, b: 6%, c: 6% og 2% på d))

a)

Figur 2 viser ein enkel 1-bit ALU. Det er 2 data inngangar (in A og in B), ein data utgang (out), 1 C-inngang og 1 C-utgang. ALUen har 3 kontroll inngangar for å bestemme funksjon (F0, F1 og F2). Finn kva aretmetisk eller logisk funksjon ALUen gjer for dei mulige kombinasjonane av kontroll inngangane (000 - 111). Bruk gjerne ein tabell som for IJVM sine ALU-funksjonar.



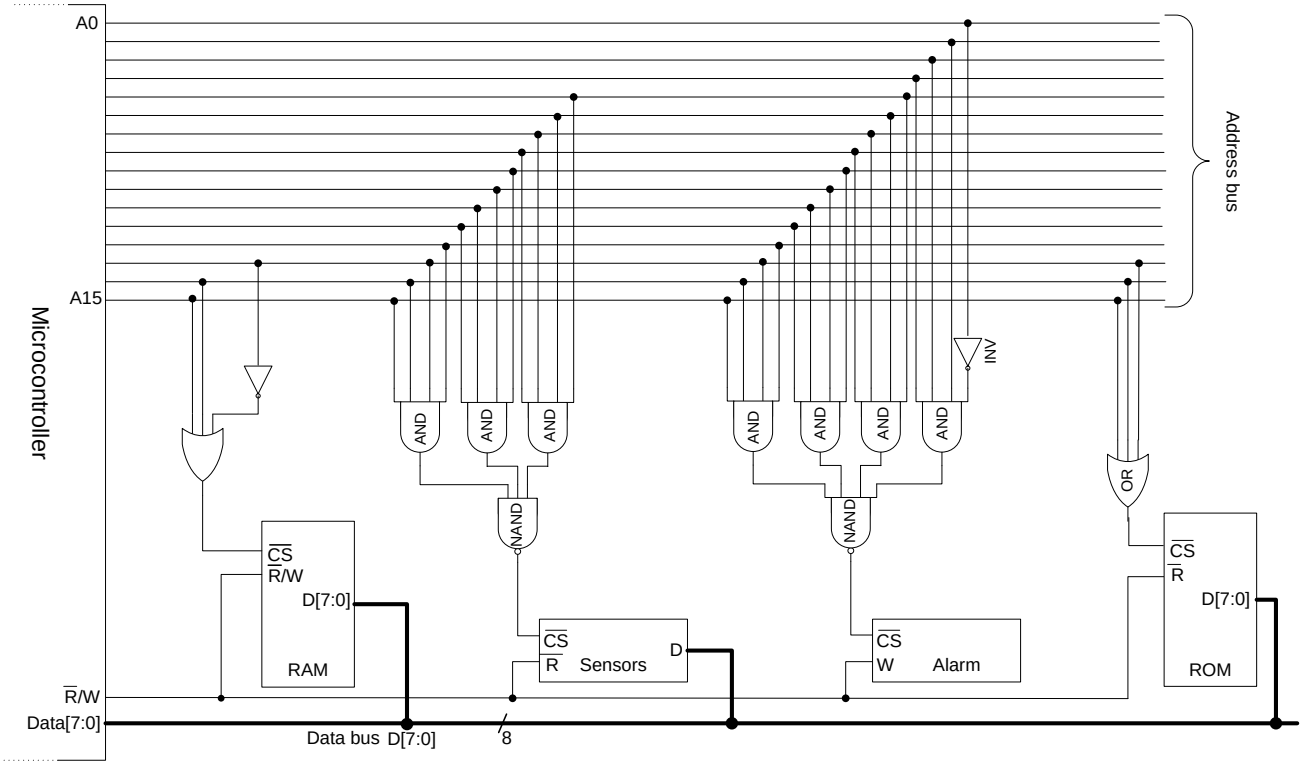
Figur 2 Enkel 1-bit ALU.

*Svar: Enkel 1 bit ALU med 2 data inngangar og ein carry inngang. F2 – F0 velger ALU funksjo, resultat av valg tilgjengeleg på mux-utgang .IJVM har 6 funksjonsinngangar (fig 11). Kan då f.eks. forklare virkemåte(logiske/*aretmetisk funksjonar) *på samme måte som fig. 11:*

F2-F0	*Out*	*Komentar*
000:	*0*	*(satt til logisk 0)*
001	*Full adder ADD A+B*	*(Add med carry, fullader C er mente bit (carry inn og ut))*
010	*A XOR B*	
011	*A AND B*	
100	*A OR B*	
101	*A NAND B*	
110	*NOT A*	
111	*A*	

Her er alle funksjonar aktive. Utganges funksjon er kunn gitt av kva F-inngangane er. Carry logikken er såleis aktiv for alle funksjonane (Out C = (A and B) + ((A xor B) and in C).

b)
I eit innvevd system (embedded system) for miljø overvaking er det ein sensor eining med 16 sensorar tilkopla. Det er ein alarm som viser viss det blir målt verdiar over ein terskel for ein eller fleire av sensorane.I systemet er det nytta ein mikrokontroller. Figur 3 viser det eksterne bussgrensesnittet med adressedekodingslogikk for mikrokontrolleren. Det er ein ROM-brikke for program, ein RAM-brikke, ein senormodul og ein alarmmodul. Alle einingane nyttar eit aktivt lågt (logisk "0") CS (Chip Select)-signal. Alle einingane nyttar 8 bit data.



Figur 3 Address decoding.

i) Finn adresseområde for RAM, ROM, Sensors og Alarm.
ii) Teikn minnekart utfrå adresseområde
iii) Er overlapp i adresseringa. I så fall er dette problematisk? Forklar kort.

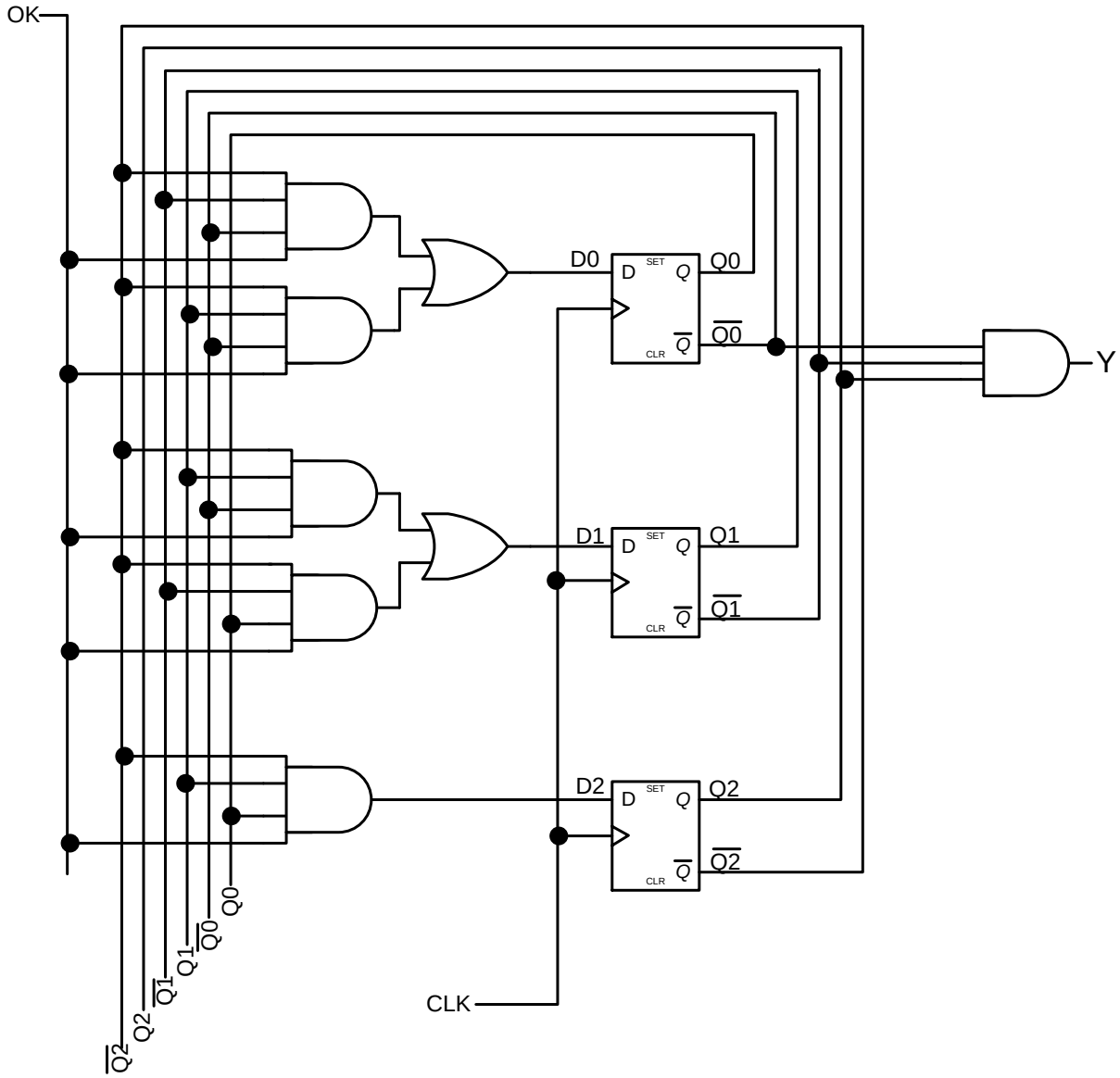
Svar:
i)

ROM: 0000 – 1FFF
RAM: 2000 – 3FFF
Sensor: FFF0- FFFF
Alarm: FFFE

ii)
Minnekart må teiknast for utteljing (utteljing viss område funnet i i) og minnekart stemmer)

iii)
Det er overlapp mellom sensor og alarm på adr: FFFE, men ikkje nødvendigvis problematisk sidan sensor kun brukar read og alarm kunn write, det vil aldri bli konflikt i minne (eining) aksess.Her ogsåutteljing viss begrunna, og relater til i)

c)
Figur 4 viser ei FSM som sjekkar at eit inngangssignal er stabilt over ei viss tid målt i klokke pulsar. FSM-en vil gi klarsignal om at alt er OK ved å legge Y høg. Logikk for reset og set for vippene er ikkje tegna inn.



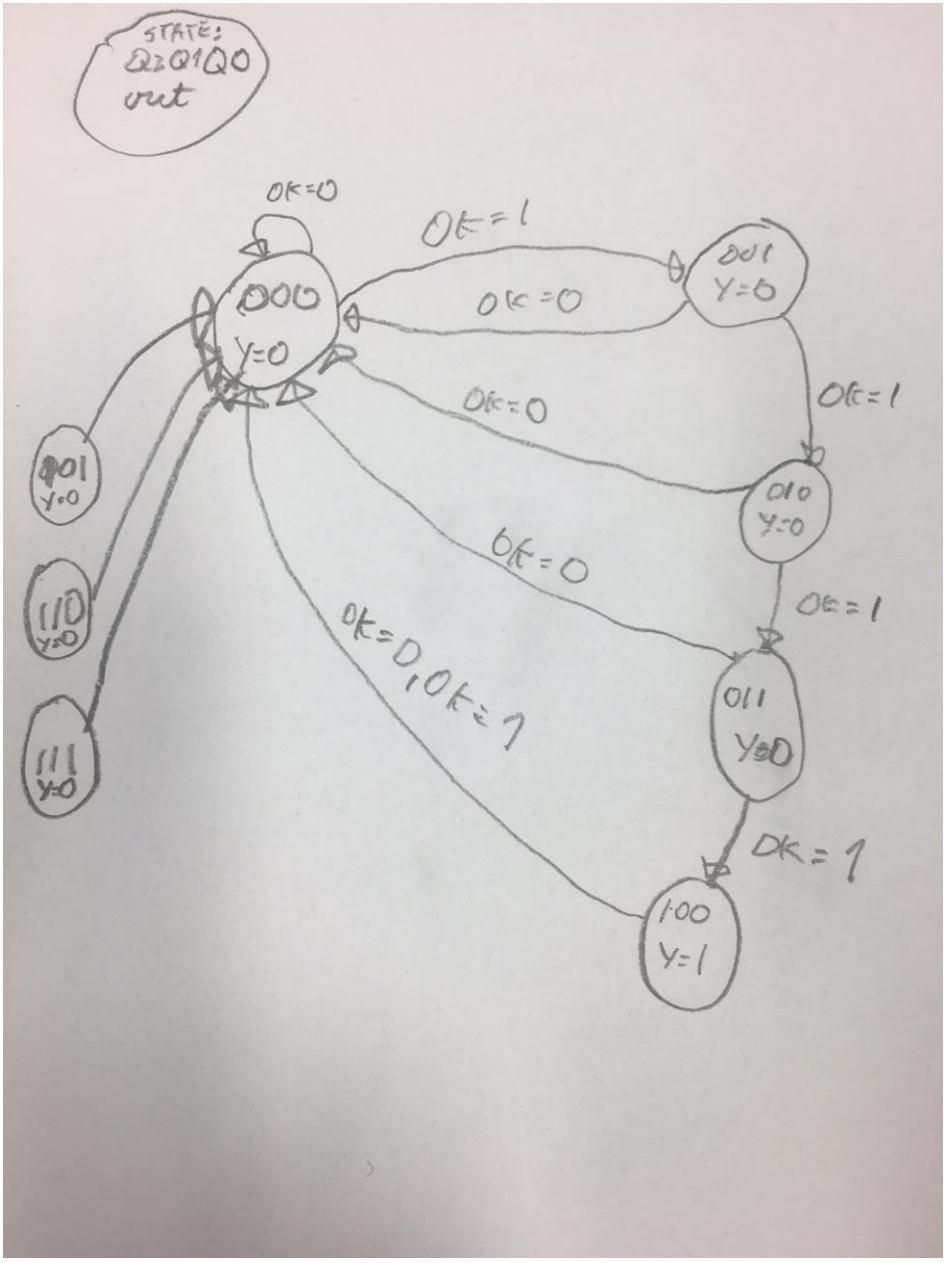
Figur 4 Finite State Machine (FSM)

Finn dei logiske uttrykka for D0, D1 og D2 (exciation 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:
D0 = (~Q2~Q1~Q0 OK) + (~Q2Q1~Q0 OK)
D1 = (~Q2 Q1 ~Q0 OK) + (~Q2~Q1Q0 OK)
D2 = (~Q2 Q1 Q0 OK)
Q0= D0, Q1 = D1, Q2 = D2.
Y = Q2 ~Q1~Q0. Teikn tabel.f. eks:

CurStat Q2Q1Q0	*Nxt State OK = 1*	*Nxt State OK = 0*	*Y*
000	*001*	*000*	*0*
001	*010*	*000*	*0*
010	*011*	*000*	*0*
011	*100*	*000*	*0*
100	*000*	*000*	*1*
101	*000*	*000*	*0*
110	*000*	*000*	*0*
111	*000*	*000*	*0*

Og state diagram kan då teiknast slik (viss ein ønskjer, ikkje spørsmål om å teilne):



d)

Er FSM-en i Figur 4 av type Mealy eller Moore?

Svar: Kunn State driven, i.e. Moore. Utgang Y kunn avhengig av current state til state registeret.

Oppgåve 3 Mikroarkitektur og mikroinstruksjoner (20% (a: 4%, b: 5%, c: 5%, d: 4% og 2% på e))

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

a)

IJVM (og i mange addre arkitekturar) har fire register som grensesnitte mot eksternt minne. I IJVM er det følgjande register: MAR, MDR, PC og MBR. Forklar kort korleis desse registera brukast og kva som er lagra i dei.

*Svar: **MAR**: MemoryAdrReg; peikar til dataminne adr.*
***MDR**: MemoryDataReg; data til/frå dataminne.*
***PC:** peikar til instruksjonsminne (opcode eventuelt operands).*
***MBR:** MemoryBufferReg; Opcode frå instruksjons (eller operand). Reg inneheld Opcode git av peikaren i PC.*
Sjå bok/lysark for forklaring.

b)

For mikroarkitekturen i Figur 9. Kva er minimum mengd mikroinstruksjonar ein må bruke for å kopiere verien i H-registeret til alle desse registera: OPC, TOS, CPP. Angi korleis dette kan gjerast effektivt ved å oppgi mikroinstruksjon(ar).

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

Svar:ALU: A (H-reg), C: OPC, TOS, CPP (bit = 1 for alle desse (skrives samtidig) .B-buss (DC, ikkje i bruk). Då mulig å gjere på ein microinstruksjon.Bit i mikroinstruksjon utfrå vedlegg

c)

For mikroarkitekturen i Figur 9. Lag mikroinstruksjonar for ein funksjon som kan teste om innhalde i registera MDR og LV er likt. Angi korleis ein slik likskap kan bli detektert.

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

Svar: Brukar inv og AND. 1: flytter LV invert til H: ALU: ~B, C: H, B:LV. Har då LV inv i H. 2: ALU: A AND B, C: 0 (lagrar ikkje resultat (kan velge å putte svar i f.eks H), B: MDR. Viss like vil ~LV AND MDR gi 0 og sette Z flagget . Andre løysingar mulig, f.eks. sub i andre mikro inst.
Bit i mikroinstruksjon utfrå vedlegg

d)

OpCode i IJVM er på 8-bit (f.eks. 0x60 IADD og 0x99 IFEQ). MPC er på 9 bit. Kva funksjon har det ekstra bite i MPC? Når er det i bruk?

*Svar: MSD(bit 8) i MPC har som funksjon å opne for programflyt i **microprogram**. MSD kan manipulereast av N og Z flagg for å velje ein av to adr. i control store, e.g. for branch zero instr.; viss Z = 0, MPC = 0xxxxxxxx. Viss Z = 1; MPC = 1xxxxxx. På denne måten kan ein få to alternative programstiar i microprog for betinga hopp instruksjonar.Gitt av J felta i mikroinstruksjon.Sjå kapittel 4, lysark.*

e)

For IJVM arkitekturen i figur 9. Kva er lengden (i bit) på ein mikroinstruksjon og kva er maksimalt mengd mikroinstruksjonar for denne versjonen av IJVM.

Svar: i figur 9 er control store definert til å inneholde ord på 36 bit. Det er 512 lokasjonar. Så då er ein micro instruksjon 36 bit og det er plass til max 512 micro instruksjonar.

Oppgåve 4 Instruksjonssett arkitektur (ISA) (20 (20% (a: 3%, b: 4%, c: 3% og 10% på d))

BarabassX42 er ein særs enkel prosessor. BarabassX42 har ein "load", ein "store", 8 ALU-instruksjonar og nokre spesialinstruksjonar, inkludert NOP-instruksjonen og to flytkontrollinstruksjonar (flow control instructions). Instruksjonsformatet for instruksjonane er vist i figur 5, figur 6 viser instruksjonssettet. Alle register og bussar er 32-bit. Det er 32 generelle register tilgjengeleg. Prosessoren har ein Harvard architecture. Bruk figur 5 og figur 6 til å løyse oppgåva.

a)
Denne prosessoren har mange eigenskapar frå RISC-prosessorar. Utfrå tilgjengelig informasjon nevn minst tre av eigenskapane til BarabassX42 som er RISC-inspirert.
Svar: F.eks:

Load/store architectur
Like lengde på alle instruksjonar (instuksjons format)
Få addreseringsmodes
Enkle instruksjonar (dei fleste untatt muligens eksternminne aksess) kan gjerast på 1 clk)'
Register-Register basert instruksjonar
Mange generelle register.
(sjå bok lysark for meir)

b)
Gi eksempel på ein BarabassX42 instruksjon som nyttar adresseringsmodi (addressing mode) imidiate adressing.
Svar: MOVC. Operand gitt som verdi i instruksjon.

Gi eksempel på ein BarabassX42 instruksjon med instruksjonsformatet null adresseinstruksjon (zero-address instruction).
Svar: NOP og RT. Begge har ikkje operandar. Det er kunn opCode.
c)
Kva er det maksimale adresserommet BarabassX42 kan adressere?
Svar: 2^32 (oxFFFFFFFF + 1). Det er 32 bit adressebuss og register som innheld adr. Peikar er på 32 bit.
d)
R8 har følgjande verdi: 0xFFFF 0000, R9 har følgjande verdi: 0xFFFF 0002. I data minne ligg følgjande data frå adresse 0xFFFF 0000:

Adresse	Data
0xFFFF 0000:	0x00 00 00 55
0xFFFF 0001:	0x00 AA 00 00
0xFFFF 0002:	0x00 00 00 AA
0xFFFF 0003:	0x00 00 55 00
0xFFFF 0004:	0x00 00 00 00

Følgjande psaudokode er ein del av eit større program. Svar på spørsmåla utfrå tilgjengeleg innformasjon.

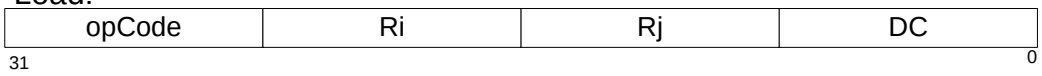
LOAD R1, R8;	*R1 = 00 00 00 55, Load frå ekternt minne (R8 adr, operand til R1)*
LOAD R2, R9;	*R2 = 00 00 00 AA, Load frå ekternt minne (R9 adr, operand til R2)*
STORE R1, R2;	*00 00 00 AA -> 00 00 00 55; Store ekternt mine (R1 lagra på adr gitt av R2)*
ADD R1, R1, R8;	*R1 = 55 + FFFF 0000= FFFF 0055, ADD, R1 ny verdi*
STORE R1, R8;	*FFFF 0000 <- FFFF 0055, Store resultat frå ADD i ektern adr gitt av R8*
LOAD R1, R2;	*R1 = 55, LOAD R1 med data frå ekternt minne adr gitt av R2 (0000AA)*

MOVC R16, 0x0055; *R16 = 0x55, Immidiate load av constant.*
ADD R1, R1, R16; *R1 = 55 + 55 = AA, ADD, ny verdi i R1*
CMP R1, R2; *AA = AA, Samanlignar to like verdiar Z-flagg = 1.*
BZ R1; *Z = 1, Hopp til instruksjonsminne adr gitt av R1 = PC = 00 00 00 AA*

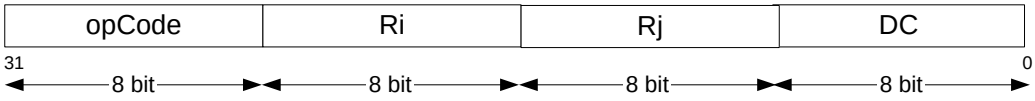
Forklar kva som skjer i koden. Kva verdi vil R1 ha etter at koden har køyrt?
Svar: R1 = 00 00 00 AA. Forklaring over som komentar til kodegitt over.

Load/store:

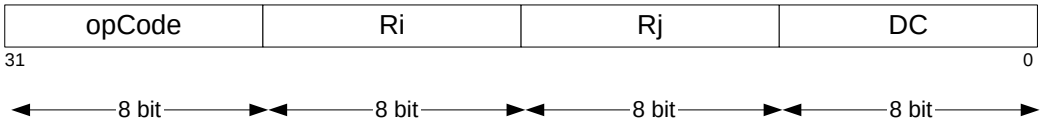
Load:



Store:

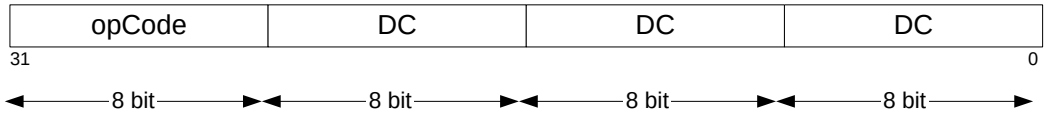


ALU:

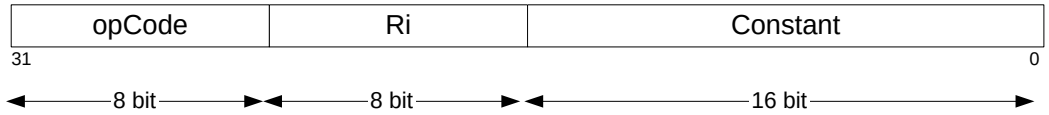


Spesial:

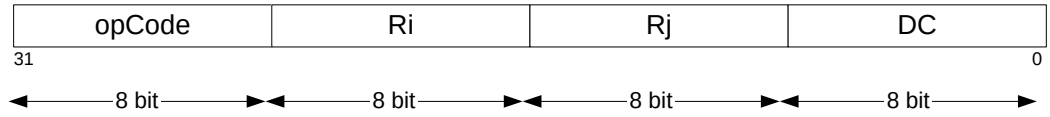
NOP:



MOVC:

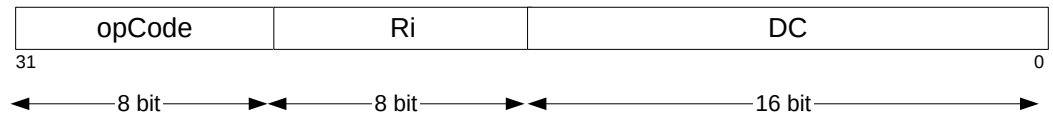


CP:

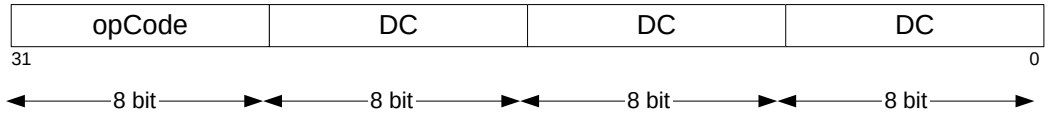


Flow control:

BZ



RT



Ri, Rj and Rk: any user register, R0 - R31
DC: Don’t care: any data memory location

Figur 5 Instruction format BarabassX42

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, $PC = Ri$.
RT Return, return from branch.

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

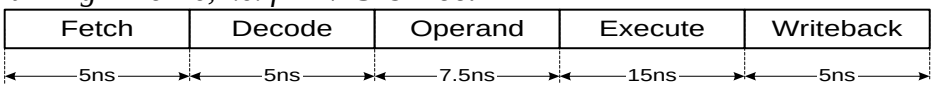
Figur 6 Instruction set BarabassX42.

Oppgåve 5 Ytelse (20 (20% (a: 5%, b: 10%, og 5% på c))

a)
Ein prosessor har eit minnesystem med RAM og eit nivå med hurtigbuffer (cache). Minnesystemet har følgjande eigenskapar:

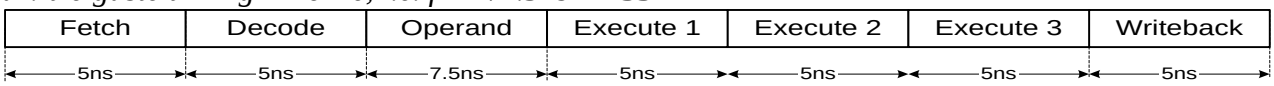
RAM: aksesstid: 100ns
Cache: aksesstid 10 ns
Trefforholdstal (Hit ratio) på 80%

Kva er gjennomsnittleg aksesstid for dette minnesystemet?
*Svar akessTime = c + (1 – h) * m = 10 + (1 – 0.8) * 100 = 30ns*
b)
1) Figur 7 viser samlebåndet (pipeline) for ein prosessor. Det er 5 steg. Estimer maksimal klokkefrekvens for denne prosessoren?
Svar: treigaste trinn gir klokke, i.e. f = 1/15ns ***= 66.7 Mhz***



Figur 7 5 stage pipeline.

2) For å auke ytinga blir execute steget delt i tre som vist i figur 8. Kva er no maksimal klokkefrekvens for prosessoren?
Svar: treigaste trinn gir klokke, i.e. f = 1/7.5ns = ***133 Mhz***

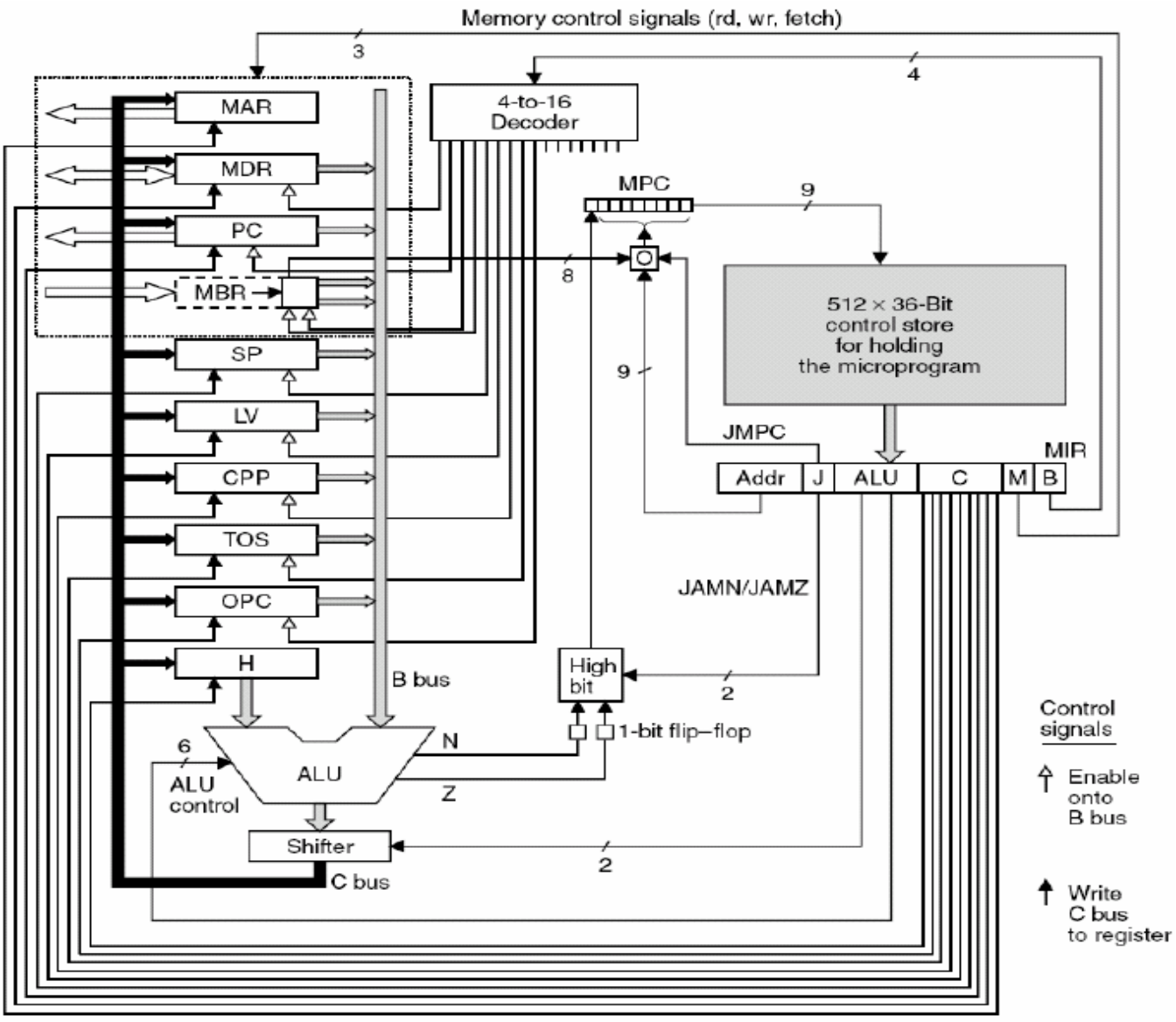


Figur 8 7 stage pipeline.

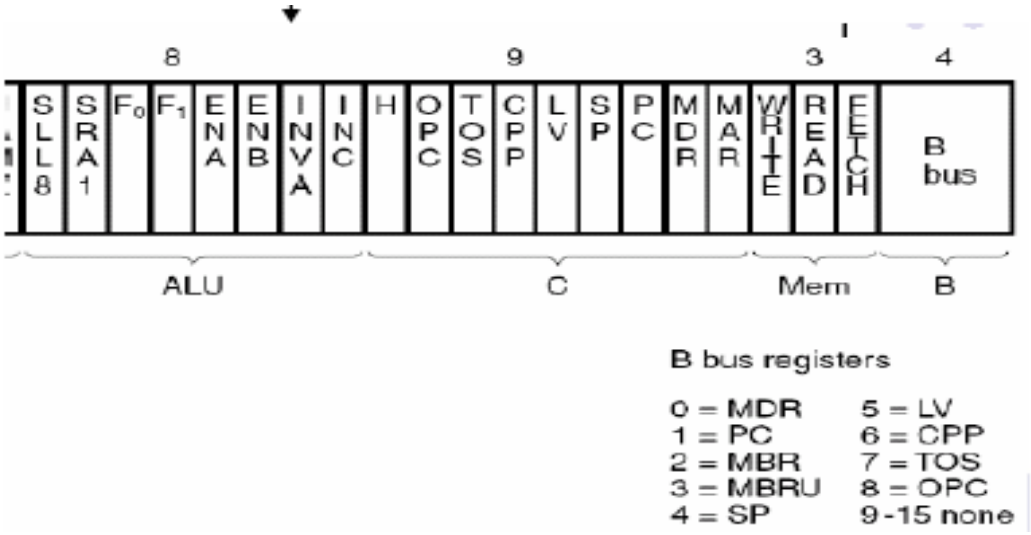
3) Det eksisterar ein variant av prosessoren utan samlebånd. Estimer kva klokkefrekvensen på denne varianten er?
Svar:Då må alle enkelt operasjonane i instruksjon utførast på ein clk periode. Må då sette klokke frekvensen til summen av enkelt operasjonar, i.e. f = 1/37.5ns = ***26.667 Mhz***

4) Kva ulemper medfører det å ha djupe (mange steg) i samlebånd? Forklar kort.
Svar: Djupe piupelines er problematisk ved ***programflyt*** *operasjonar, f.eks conditional branch. Då må pipeline tømmast viss ein hoppar (eller viss branch predict bommar og fyller pipeline med feil programsti instruksjonar/operands).* ***Avhengigheiter*** *der resultat av ein instrukson ikkje er f.eks ferdigt før det skal brukast av neste instruksjon, får då «stal» i pipelina*
c)
Figure 9 viser mikroarkitekturen til IJVM. Foreslå to tiltak som aukar ytelsen.
Svar: Frå bok forelesing:
-Innføre A-buss, kan då gi ALU tilgong på 2 vilkårlege register (sparar mikroinstruksjonar (raskare instruksjonsutføring))
-Innføre Instruction Fetch Unit (IFU) som hentar instruksjonar automagisk. Slepp å bruke datapath til å oppdatere PC og kan ha kø av instruksjonar og operandar (slepp å vente på minne aksess (pre-fetch).
-Innføre pipeline, mange muligheiter, f.eks. fetch, decode, operand fetch (kø), tre trinn i datapath, writeback.
-Cache, for å minke aksesstid for minne aksess.

Vedlegg IJVM



Figur 9 IJVM

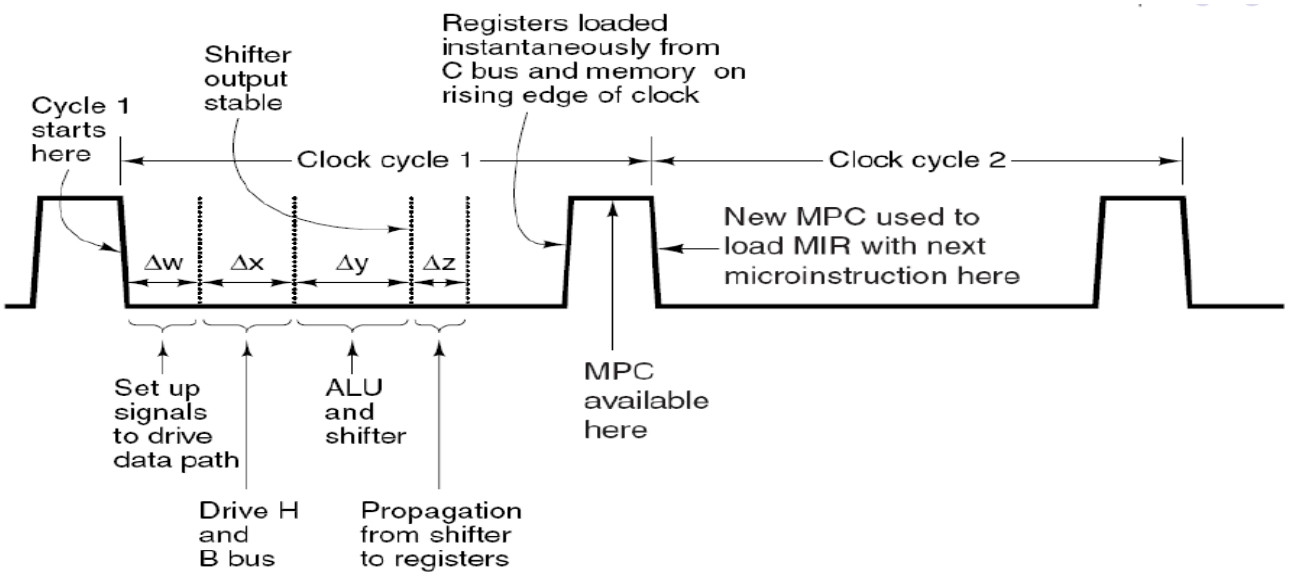


Figur 10 Microinstruction format.

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 iming diagram.