

# i Informasjon

## **Skriftlig eksamen IN2060 - Digitalteknikk og datamaskinarkitektur Høst 2023**

Tid for eksamen: 12. desember kl. 09:00 til 12. desember kl. 13:00

Varighet: 4 timer

Hjelpemidler: Ingen

N.B.: Inspera kalkulator tilgjengelig ved behov

**Det er viktig at du leser denne forsiden nøye før du starter.**

### **Om oppgavene**

Oppgavesettet består av forskjellige typer oppgaver; både oppgaver der du skal taste inn tall og forskjellige typer flervalgsoppgaver. Noen oppgaver kan ha vedlegg som er vesentlige for å løse oppgaven.

Pass derfor på å sjekke at du har lest og besvart hele oppgaven for hver oppgave, og benytt gjerne rullefeltene (scrollbarene) til henholdsvis vedlegg og oppgave for å kontrollere at du har fått med deg alt. Vedleggene kan forstørres fra topplinjen.

Flervalgsoppgaver med radioknapper kan endres, men ikke skrues av når du har valgt alternativ.

Oppgaver med mer enn ett riktig svar har avkrysningsbokser der man kan fylle inn inntil det antall svar som er riktig. Det er ikke mulig å krysse av flere svar enn det som er riktig.

### **Om poeng i dette eksamenssettet**

I dette oppgavesettet er det mulig å oppnå 100 poeng totalt. Poengene for hver oppgave er oppgitt på oversiktssiden for å angi vektingen av hver oppgave slik at du kan disponere tiden. Det blir ikke gitt trekk for feil avkryssning.

Lykke til!

## 1 Digital representation 1

### Digital representasjon

Gjør om desimaltallet  $(37)_{10}$  til et 8 bits binærtall.

#### Velg ett alternativ

- 00100010
- 00110011
- 00100111
- Ingen av alternativene er korrekte.
- 00100101



---

Maks poeng: 2

## 2 Digital representasjon 2

### Digital representasjon

Gjør om det oktale tallet  $(33)_8$  til et 8 bits binærtall.

#### Velg ett alternativ

- 00101101
- 00011011
- Ingen av alternativene er korrekte.
- 00010111
- 00101111



---

Maks poeng: 2

### 3 Bitopløsning

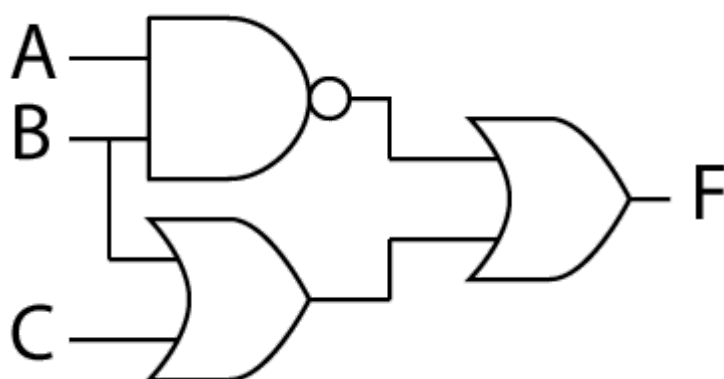
Hva er det største positive tallet du kan utrykke med et binært 8-bits tall på 2'er komplement form?

:  (127) .

Maks poeng: 2

### 4 Boolske porter

Hvilket funksjonsuttrykk gjenspeiler portimplementasjonen under?



Velg ett alternativ

- $F = A'B' + (B+C)'$
- $F = A'B'(B+C)$
- $F = (AB)' + B + C$
- Ingen av alternativene er korrekte.
- $F = AB' + (B+C)$



Maks poeng: 2

## 5 Boolsk algebra

Forenkle følgende uttrykk maksimalt

$$F = (A+B)(A+B')(A'+C)$$

Velg ett alternativ

- $F = AC$
- Ingen av alternativene er korrekte
- $F = AB+AC$
- $F = A'B$
- $F = AB+BC+AC'$



---

Maks poeng: 3

## 6 Kombinatorisk og sekvensiell logikk

Hvilke to utsagn om kombinatorisk og sekvensiell logikk er **riktige**?

**Velg de to alternativene som er riktige.**

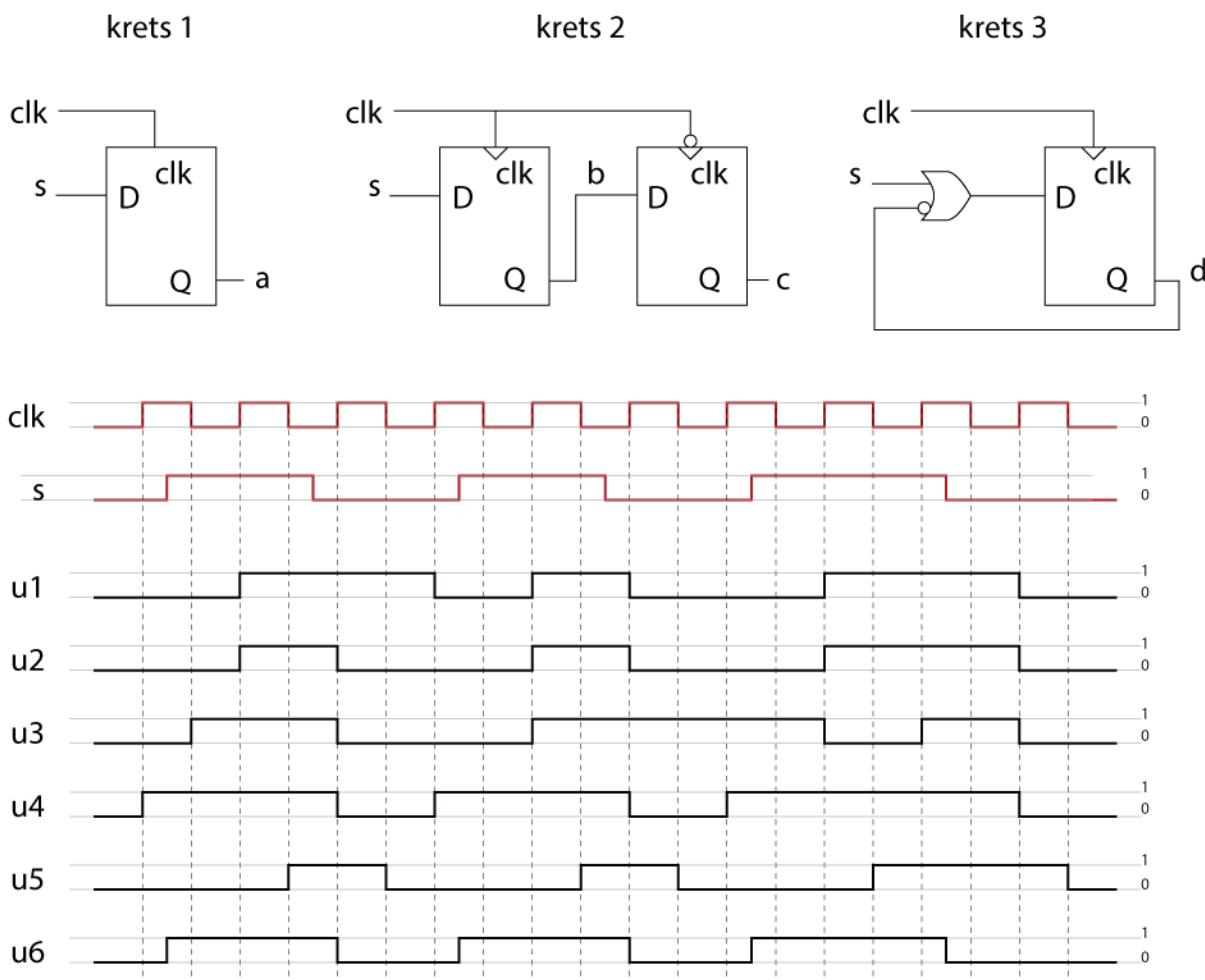
- Kombinatoriske kretser er synkrone og opererer med et klokkesignal.
- En teller er et eksempel på en sekvensiell krets. ✓
- Utgangen av en sekvensiell krets avhenger både av nåværende innganger og forrige tilstand i systemet. ✓
- En flip-flop er et eksempel på en kombinatorisk krets
- En klokke er nødvendig for å synkronisere kombinatoriske kretser.
- En halvadder er et eksempel på en sekvensiell digital krets

---

Maks poeng: 2

## 7 Sekvensielle kretser

Hver av de tre kretsene under tar inn klokkesignalet **clk** og inngangssignalet **s**. Anta at utgangene **a**, **b**, **c** og **d** har startverdien **0**. Hvilke tidsforløp under (u1 til u6) hører til de forskjellige utgangene? Legg merke til at det er gitt to tidsforløp for mye. Du skal ikke ta hensyn til portforsinkelse. **Studér kretsene nøye og merk forskjellen på latcher og flippflopfer i illustrasjonen.**



- Utgang **a** hører til  (u1, u2, u3, u4, u5, **u6**)
- Utgang **b** hører til  (u1, **u2**, u3, u4, u5, u6)
- Utgang **c** hører til  (u1, u2, u3, u4, **u5**, u6)
- Utgang **d** hører til  (u1, u2, u3, **u4**, u5, u6)



## 8 VHDL og kritisk sti

**Merk:** I denne oppgaven forventes det eksakte tallsvar i samsvar med angitt prefiks og størrelse.

I en ASIC er propageringsdelayet til en to-inputs port (and, or, xor) 50 ps.

Vi har en fulladder som angitt med VHDL koden på side 1 i dokumentet til oppgaven.

Hva blir propageringsdelayet til én fulladder hvis den implementeres i ASICen?

Sett inn et tall:  (150) ps.

Vi lager en krets ved å koble sammen flere fulladdere som vist i multibit-entiteten på side 2 i dokumentet. Hva kalles en slik krets?

**Velg ett alternativ**

- Ripple-carry adder ✓
- multibit-carrier
- multiply-carry chain
- Carry lookahead adder
- Prefix adder

Hva er den lengste stien (critical path) i multibit-kretsen?

**Velg ett alternativ**

- Fra a(0) eller b(0) til sum(3) ✓
- Fra cin til carry(4)
- Fra cin til b(3)
- Fra a(0) til b(3)
- Fra a(3) til sum(0)

Hva blir det maksimale propageringsdelayet i multibit-kretsen, gitt samme propageringsdelay per port som angitt i første deloppgave?

Sett inn et tall:  (350) ps.

I en FPGA blir hver fulladder implementert i en egen oppslagstabell (LUT). Hver oppslagstabell har et propageringsdelay på 200 ps for alle signaler.

Hva blir det totale propageringsdelayet om multibit-kretsen implementeres i en FPGA?



Sett inn et tall:  (600) ps.

I FPGA implementasjonen bruker vi et register til å lagre dataene før neste steg i en lengre pipeline.

Lagring i register gir et tillegg i kritisk sti (critical path) på 200 ps.

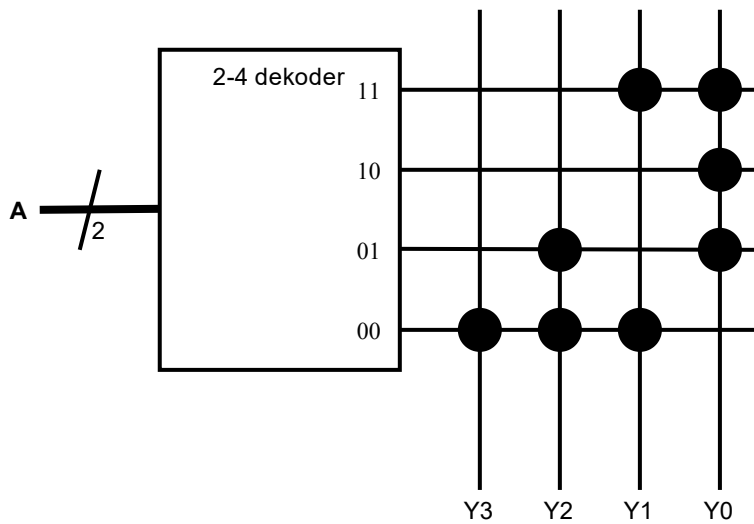
Hva blir den maksimale klokkefrekvensen vi kan ha på FPGA-implementeringen?

Sett inn et tall:  (1250) MHz.

---

Maks poeng: 12

## 9 Oppslagstabell



Figuren viser en implementering av en oppslagstabell (LUT). Hvilke logiske uttrykk implementerer de ulike utgangene til LUTen?

Plasser de riktige alternativene i sine respektive bokser

[Hjelp](#)

A1 xnor A0

A1 or A0

A1 nand A0

not A1

not (A1 or not A0)

not A0

A0 not A1

not (A1 or A0)

A1 and A0

(A1 or A0) and not A0

Y3=  ✓

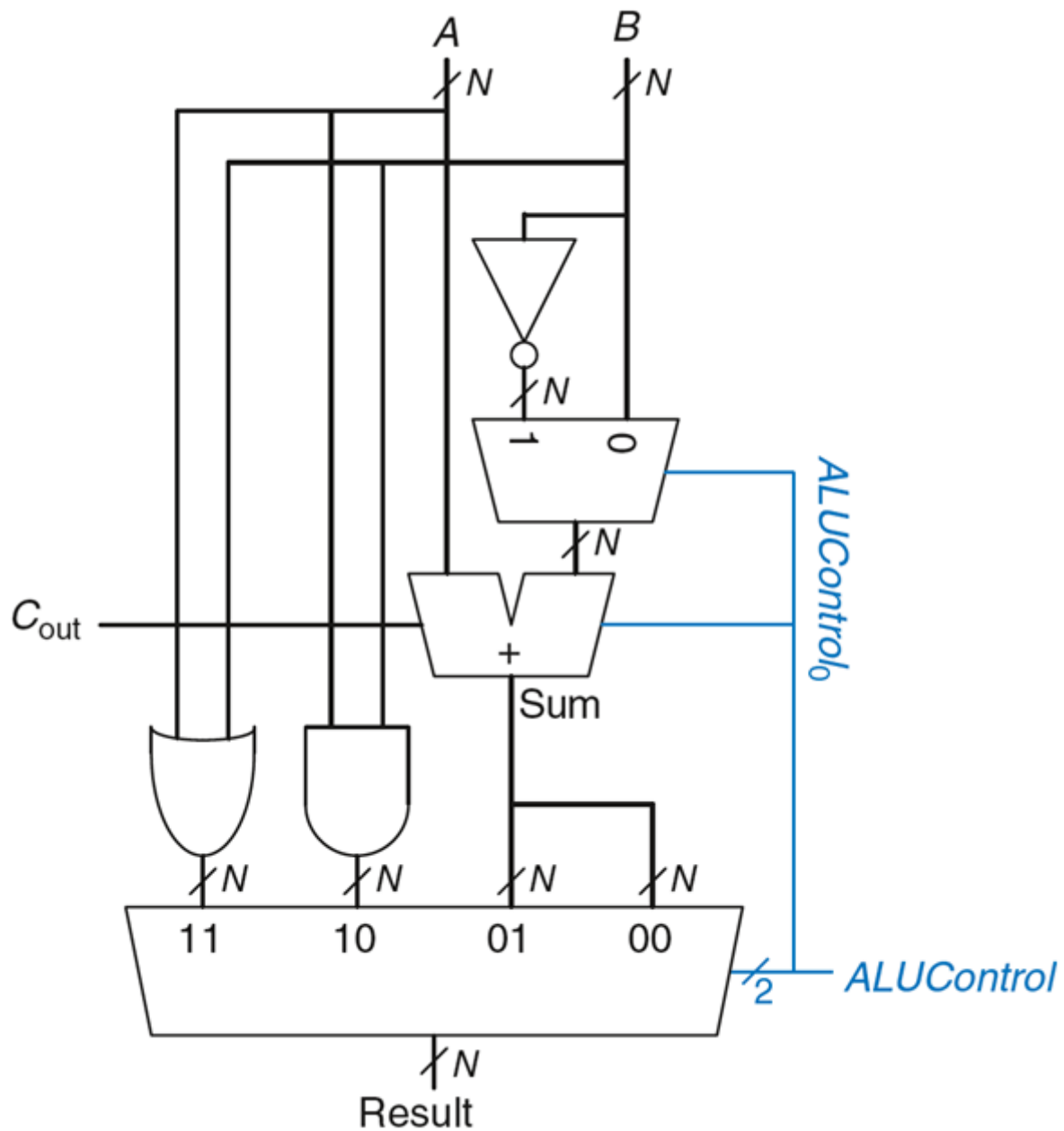
Y2=  ✓

Y1=  ✓

Y0=  ✓

Maks poeng: 4

## 10 ALU



Vi har en ALU som vist på figuren over,  $N=16$ .

Vi setter  $A = 0x0065$  og  $B = 0xFFDD$

Hva blir resultatet («Result») når vi setter ALUcontrol(1 downto 0) til...

a) "10"

Velg ett alternativ

0x0000

0x0045

0xFFFF

0x0042

0x0088

b) "01"

## Velg ett alternativ

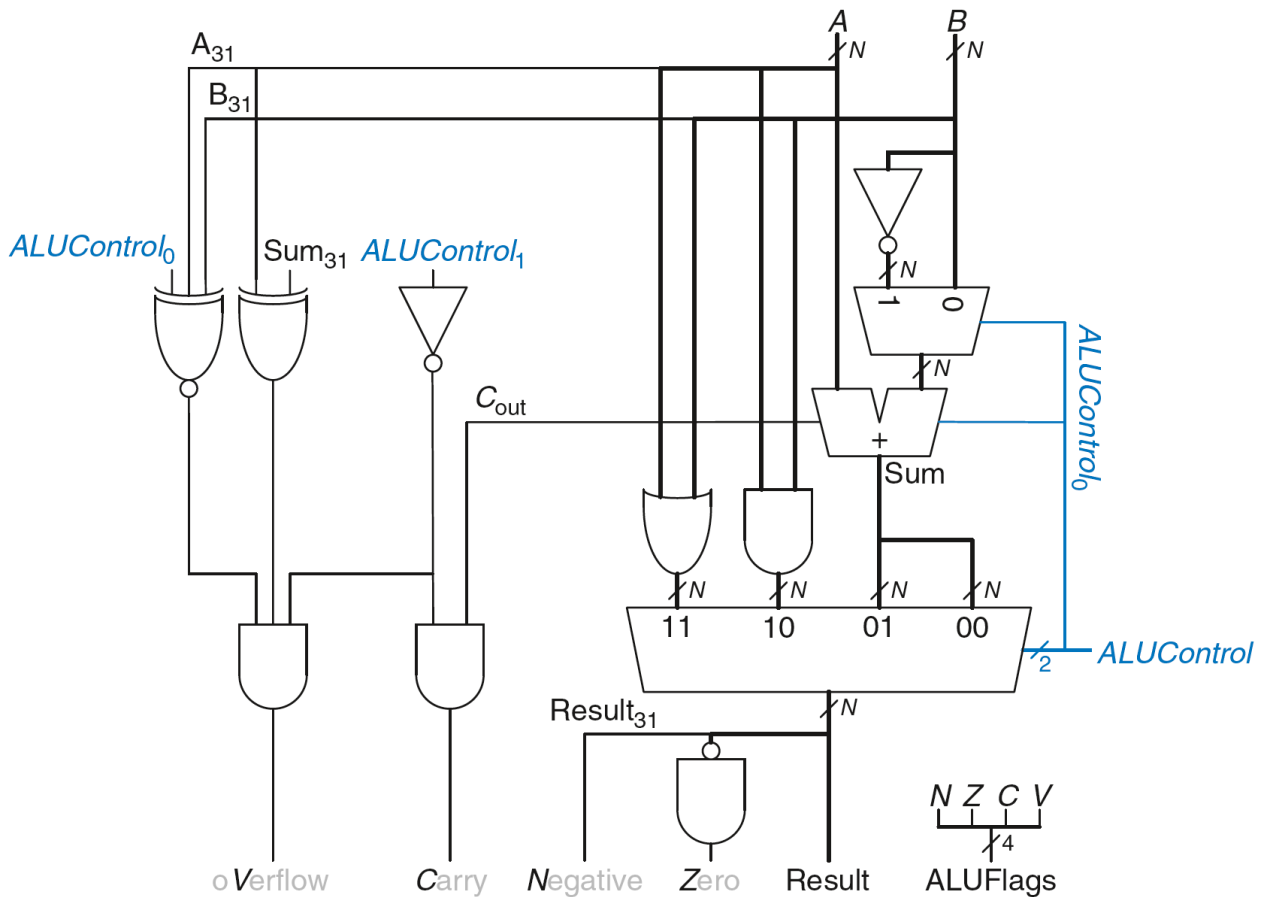
0x0021

0x0000

0xFFFF

0x0088

0xFFBE



Figuren over viser en ALU med kontrollflagg

c) Vi gjør samme operasjon som i a). Hvilket flagg blir satt:

## Velg ett alternativ

V

Z

C

alle fire

N

ingen

Maks poeng: 6

## 11 Maskinkode

Velg det alternativet som gir en nøyaktig beskrivelse av maskinkode.

**Velg ett alternativ:**

- Maskinkode er en form for bytecode som blir tolket av operativsystemet
- Maskinkode består av heksadesimale representasjoner av programinstruksjoner.
- Maskinkode er en serie instruksjoner skrevet i assemblerspråk
- Maskinkode består av binære instruksjoner som representerer operasjonene som utføres direkte av mikroarkitekturen.
- Maskinkode er en serie instruksjoner skrevet i et høynivå programmeringsspråk.

---

Maks poeng: 2

## 12 Branch Target Address

Gitt utsnittet av ARM assemblerkoden under, hvilken tallverdi må *imm24* feltet til *maskinkoden* til Branch instruksjonen (BL) ha?

```

0x8000      SUB R0, R0, R1
0x8004      BL LABEL
0x8008      ADD R1, R0, #9
0x800C      SUB R0, R0, R1
0x8010      ORR R2, R1, R3
0x8014      ADD R0, R1, R2
0x8018 Label ADD R3, R3, #23
0x801C      SUB R3, R3, #23

```

Velg ett alternativ:

- 5
- ingen av verdiene er riktige
- 3 ✓
- 20
- 4

---

Maks poeng: 2

## 13 Tolking av maskinkode

Dekod følgende ARM instruksjon (maskinkode) slik det er beskrevet i læreboken, og velg de alternativene som danner tilsvarende assembler-instruksjon.

**0xE0910002**

Velg alternativ (ADD, ANDEQ, ADDLT, **ADDS**) Velg alternativ (R2, **R0**, R6, R4)

Velg alternativ (R2, R0, **R1**, R4) Velg alternativ (**R2**, #2, #1, R1)

---

Maks poeng: 6

## 14 Oversette til assemblerkode

C funksjonen nedenfor gjør modulo operasjonen  $a\%b$  for positive heltall. Oversett C funksjonen nedenfor til assembler ved å fylle inn de manglende instruksjonene. Husk at a er lagt i r0 og b i r1 før funksjonen kalles.

```
int Modulo(int a, int b){
    while(a >= b){
        a = a - b;
    }
    return a;
}
```

MODULO: CMP r0, r1

Velg alternativ (BGE END, BNE END, **BLT END**, BX LR, BLE END)

Velg alternativ (**SUB r0, r0, r1**, ADD r0, r0, r1, SUB r1, r1, r0, LSR r0, r0, r1, CMP r0, r1)

Velg alternativ (BEQ MODULO, **B MODULO**, LSL r0, r0, #2, ADD r0, r1)

END: Velg alternativ (PUSH LR, POP LR, POP PC, MOV r0, PC, **MOV PC, LR**)

---

Maks poeng: 6

## 15 Funksjonskallkonvensjonen

Gitt koden nedenfor, hvilke registre må F2 huske å lagre på stacken i følge funksjonskallkonvensjonen?

```
F2: MOV R2, #2
     MOV R3, #5
     ADD R4, R0, R2
     ADD R5, R0, R3
     MUL R0, R4, R5
     MOV PC, LR
```

Velg ett alternativ:

- R1-R4, LR
- R4, R5
- R5, PC
- PC, LR
- R1-R3
- R5



---

Maks poeng: 2



## 16 Oversett til C

Hvilken av de følgende C funksjonene gjør det samme som denne assemblerfunksjonen?

```
F2:    CMP R0, R1
      BEQ CASE2
CASE1: ADD R0, R1, #5
      B END
CASE2: ADD R0, R1, #3
END:   MOV PC, LR
```

Velg ett alternativ:

```
int F2(int a, int b){
    if(a == b){
        return b + 3;
    }
    else{
        return b + 5;
    }
}
```



```
int F2(int a, int b){
    if(a == b){
        return b + 3;
    }
    else{
        return b + 3;
    }
}
```

```
int F2(int a, int b){
    if(a == b){
        return a + 5;
    }
    else{
        return a + 3;
    }
}
```

```
int F2(int a, int b){
    if(a != b){
        return b + 3;
    }
    else{
        return b + 5;
    }
}
```

---

Maks poeng: 3

## 17 Mikroarkitektur og ytelse

Vurder tre mikroprosessorarkitekturer: *Single-Cycle*, *Multi-Cycle* og *Pipeline* som beskrevet i læreboken. Anta at **klokkefrekvensen er den samme for alle tre arkitekturerne**, og at de utfører det samme instruksjonssettet. Hvilken av følgende påstander er generelt sant angående deres ytelse gitt antagelsen over?

**Velg ett alternativ:**

- Pipeline er alltid raskere enn Single-Cycle, og Single-Cycle er alltid raskere enn Multi-Cycle.
- Alle tre arkitekturerne vil ha identiske ytelseskarakteristikker.
- Pipeline er alltid raskere enn Multi-Cycle, og Multi-Cycle er alltid raskere enn Single-Cycle.
- Single-Cycle vil være raskere enn Pipeline og Pipeline vil typisk være raskere enn Multi-Cycle. ✓
- Ingen av utsagnene er korrekte.

---

Maks poeng: 2

## 18 Mikroarkitektur antall klokkesykler

Gitt følgende assemblerkode

```
MOV R0, #1
ADD R0, R1, R2
SUB R1, R1, R0
CMP R0, R1
ADDEQ R2, R1, R2
ADD R3, R1, R2
```

Hvor mange klokkesykler vil følgende mikroarkitektur-design bruke på kjøre dette programmet?

- Et *Single-cycle* design som beskrevet i boka.:  (6) .
- Et *Multicycle* design der du kan anta en fast CPI på 4 for alle instruksjonstyper  (24) .
- Et 5 steps *Pipeline* design med hazard enhet som beskrevet i boka  (10) .

---

Maks poeng: 3

## 19 Branch prediction

I en pipeline mikroarkitektur, hva er formålet med *branch prediction*?

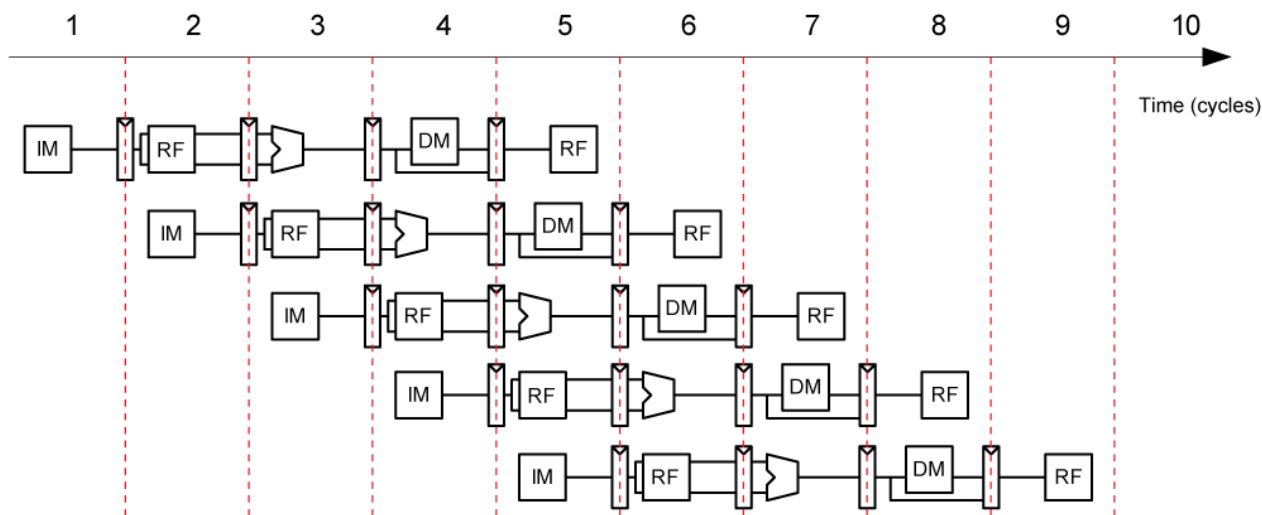
Velg ett alternativ:

- Å kunne forutsi utfallet av branch-instruksjoner og redusere pipeline stalls. ✓
- Å synkronisere parallelle prosesseringsenheter for å oppnå en effektiv instruksjonsutførelse
- Å forhindre at instruksjoner hentes fra feil minneadresse
- Å minimere data-avhengigheter i pipeline'en.
- Ingen av utsagnene er korrekte.

---

Maks poeng: 2

## 20 Pipeline



Gitt følgende ARM-assemblerprogram:

```

ADD R1, R1, R2
LDR R3, [R2, #20]
STR R7, [R1, #24]
SUB R2, R7, R1
ADD R4, R7, R2

```

Hva slags pipelineforløp vil koden over gi? Anta en 5-steps pipelinet prosessor som illustrert over (tilsvarende som i boka), men uten noen form for hazard-håndtering. Her skriver vi til registerfilen i første halvdel av klokkeperioden, og leser av i andre halvdel av klokkeperioden. Velg de alternativene under som er korrekte for pipelineforløpet.

Hva slags register aktivitet har vi i følgende klokkesykler?

- I sykel 3  (leser fra og skriver til registeret, **leser fra registeret**, ingen register aktivitet, skriver til registeret)
- I sykel 4  (**leser fra registeret**, skriver til registeret, leser fra og skriver til registeret, ingen register aktivitet)
- I sykel 5  (**leser fra og skriver til registeret**, leser fra registeret, skriver til registeret, ingen register aktivitet)
- I sykel 6  (leser fra registeret, ingen register aktivitet, skriver til registeret, **leser fra og skriver til registeret**)

Hvilken hazard har vi i følgende klokkesykler?

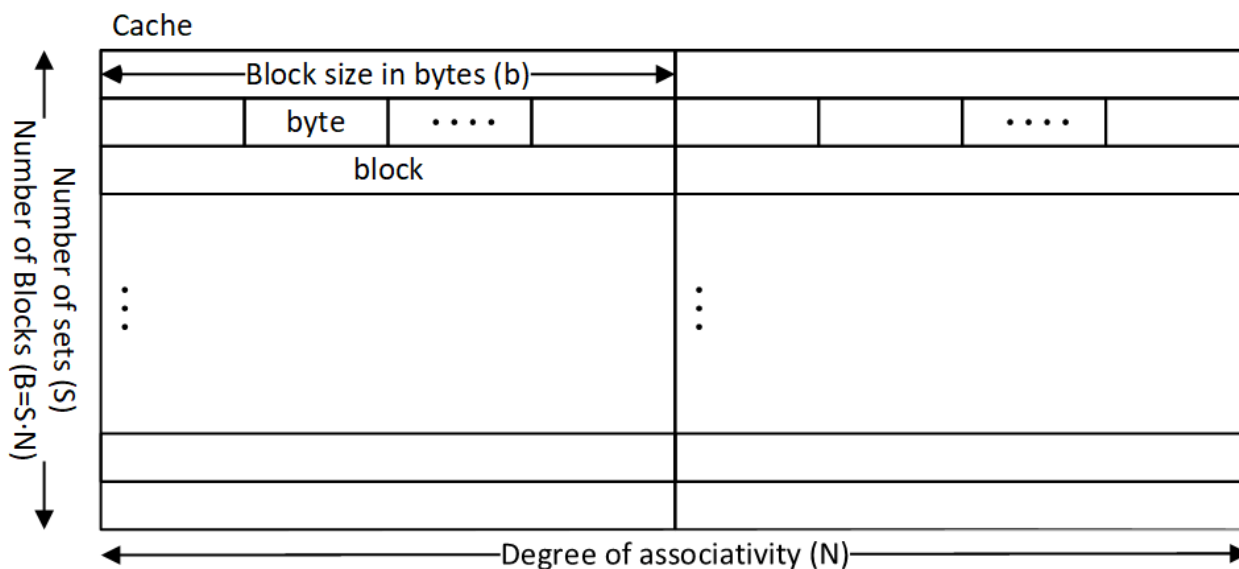
- I sykel 3  (ingen hazard, control hazard, data hazard)
- I sykel 4  (ingen hazard, data hazard, control hazard)
- I sykel 5  (data hazard, control hazard, ingen hazard)
- I sykel 6  (ingen hazard, data hazard, control hazard)

---

Maks poeng: 6

## 21 Direkte-mappet cache

I denne oppgaven skal du fylle inn heltall.



Figuren over viser en generisk cache.

Til et fysisk minne på 256 MB bruker vi en direkte-mappet cache ( $N=1$ ) med blokkstørrelse på to ord, fire byte per ord, og kapasitet på 32 kB. Hver byte i minnet har en egen adresse.

Hvor mange bit må en adresse ha for å kunne adressere hele det fysiske minnet?

 (28)

Hvor mange set vil cachen ha?

 (4000 - 4100)

Hvor mange bit må vi ha i adresse-tagene til cachen?

 (13)

Maks poeng: 6

## 22 Minneoppslag

I denne oppgaven skal du oppgi tallsvar. Tallsvarene kontrolleres med en presisjon på inntil 2 desimaler. Det er greit å benytte flere desimaler, men ikke nødvendig ved korrekt avrunding.

Vi har en direkte-mappet cache med kapasitet på 32 ord og en blokkstørrelse på 2 ord og en ordstørrelse på 4 byte.

Et program gjør minneaksess fra følgende adresser i sekvens:

0x41, 0x45, 0x49, 0x84, 0x60, 0x65, 0x40, 0x44, 0x48, 0x84, 0x60, 0x64

Hvor mange av minneaksessene vil resultere i bom (miss) ved oppslag i cache?

(4)

Hva er treffraten for disse minneaksessene?

(0,66 - 0,67) (andel av én)

Vi sammenligner med en to-veis set-assosiativ cache med samme kapasitet og blokkstørrelse på fire ord. Den nye cachen skifter ut blokken det er lengst siden ble brukt (LRU).

Hva blir missraten om vi kjører den samme sekvensen med minneoppslag i den nye cachen?

(0,25) (andel av én)

Hva konvergerer treffraten mot dersom sekvensen gjentas uendelig mange ganger?

(1) (andel av én)

---

Maks poeng: 10

## 23 Virtuelt minne

Hvilket utsagn er riktig om sideoppslag (paging, page table)

**Velg ett alternativ:**

- Sideoppslagstabellen lagres i nivå-2 cache
- Sideoppslag kan ikke caches
- Sideoppslagstabellen bruker ALU til beregning av fysiske adresser
- Sideoppslagstabellen lagres i hovedminnet eller harddisk/SSD ✓
- Sideoppslagstabellen har typisk 16 oppslag

Hvilket utsagn er riktig om sideoppslagscachen (TLB)?

**Velg ett alternativ**

- TLB er vanligvis fullt assosiativ ✓
- TLB står for Transmission logic block
- Hit-raten til TLB er ekstremt lav fordi det virtuelle minnet er kjempestort
- Oppslag i TLB tar alltid flere klokkesykler
- TLB er direkte-mappet

Hvis vi har like stort virtuelt minne som fysisk minne (f.eks 4GB), hvilken fordel har vi da av å bruke virtuelt minne i en datamaskin?

**Velg ett alternativ**

- Virtuelt minne gir oss mer lagringsplass uansett
- Virtuelt minne sørger for at ulike programmer kan dele minne
- Det er aldri noen fordel å bruke virtuelt minne uten harddisk eller SSD
- Virtuelt minne kan huske lengre tilbake enn konvensjonelt minne
- Virtuelt minne gjør at ulike programmer kan benytte samme adresser uten å komme i konflikt ✓



---

Maks poeng: 3

**Question 12**  
Attached



## Data-processing instructions

Name	Description	Operation
ADD Rd, Rn, Src2	Add (+)	$Rd = Rn + Src2$
SUB Rd, Rn, Src2	Subtract (-)	$Rd = Rn - Src2$
AND Rd, Rn, Src2	Bitwise AND (&)	$Rd = Rn \& Src2$
ORR Rd, Rn, Src2	Bitwise OR ( )	$Rd = Rn   Src2$
EOR Rd, Rn, Src2	Bitwise Exclusive OR (^)	$Rd = Rn \wedge Src2$
BIC Rd, Rn, Src2	Bitwise Clear	$Rd = Rn \& \sim Src2$
MVN Rd, Rn, Src2	Bitwise NOT (~)	$Rd = \sim Rn$
LSL Rd, Rn, Src2	Logical Shift Left (<<)	$Rd = Rn \ll Src2$
LSR Rd, Rn, Src2	Logical Shift Right (>>)	$Rd = Rn \gg Src2$
MOV Rd, Src2	Move (=)	$Rd = Src2$
CMP Rd, Src2	Compare	Set flags (see below) based on $Rd - Src2$

Remember that we can also set condition flags by appending an *S* to the end of our Data-processing instructions.

Name	Description
ADDS Rd, Rn, Src2	Add (as above) <b>and</b> set condition flags
SUBS Rd, Rn, Src2	Subtract (as above) <b>and</b> set condition flags
ANDS Rd, Rn, Src2	Bitwise AND (as above) <b>and</b> set condition flags

## Multiply instructions

Name	Description	Operation
MUL Rd, Rn, Rm	Multiply (*)	$Rd = Rn * Rm$
MULS Rd, Rn, Rm	Multiply (*) <b>and</b> set condition flags	$Rd = Rn * Rm$
MLA Rd, Rn, Rm, Ra	Multiply and Accumulate	$Rd = (Rn * Rm) + Ra$

## Memory instructions

Name	Description	Operation
STR Rd, [Rn, ± Src2]	Store Register	$Mem[Adr] = Rd$
LDR Rd, [Rn, ± Src2]	Load Register	$Rd = Mem[Adr]$

## Branch instructions

Name	Description	Operation
B label	Branch	$PC = (PC + 8) + imm24 \ll 2$
BL label	Branch and Link	$LR = (PC + 8) - 4;$ $PC = (PC + 8) + imm24 \ll 2$
BX Rd	Branch and eXchange	Branch to address pointed to in Rd (used for return)

## Condition flags

Flag	Name	Description
N	Negative	Instruction result is negative
Z	Zero	Instruction result is zero
C	Carry	Instruction caused a carry out
V	oVerflow	Instruction caused an overflow

## Condition mnemonics

Mnemonic	Name	CondEx
EQ	Equal	Z
NE	Not Equal	!Z
CS/HS	Carry set / unsigned higher or same	C
CC/LO	Carry clear / unsigned lower	!C
MI	Minus / negative	N
PL	Plus / Positive <i>or</i> zero	!N
VS	Overflow	V
VC	No overflow	!V
HI	Unsigned higher	!Z AND C
LS	Unsigned lower or same	Z OR !C
GE	Signed greater than or equal	!N XOR !V
LT	Signed less than	N XOR V
GT	Signed greater than	!Z AND (!N XOR !V)
LE	Signed less than or equal	Z OR (N XOR V)

**Question 13**  
Attached



# Maskinkodevedlegg

## Betingetkjøring mnemonics

Kode	Mnemonic	Navn
0000	EQ	Likhet
0001	NE	Ulikhet
0010	CS/HS	Set Carry
0011	CC/LO	Fjern Carry
0100	MI	Minus / negativt tall
0101	PL	Plus / positivt eller null
0110	VS	Overflyt / set overflyt (Overflow)
0111	VC	Ikke overflyt / fjern overflyt (Overflow)
1000	HI	Høyere - positive heltall (Unsigned higher)
1001	LS	Lavere - positive heltall (Unsigned lower)
1010	GE	Større eller lik - heltall (Signed greater than or equal)
1011	LT	Mindre - heltall (Signed less than)
1100	GT	Større - heltall (Signed greater than)
1101	LE	Mindre eller lik - heltall (Signed less than or equal)
1110	AL	Ubetinget - alltid utfør

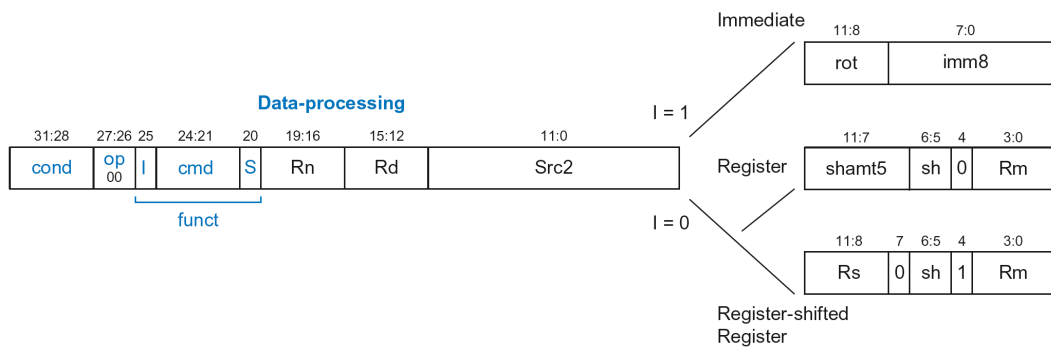


Figure 1: Data processing instruction format

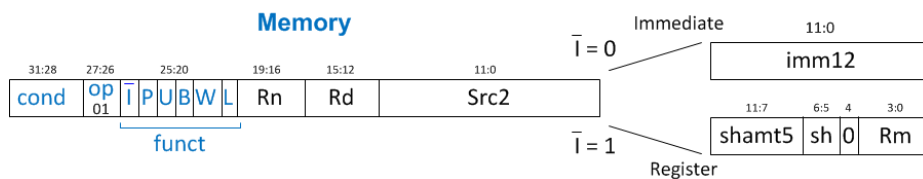


Figure 2: Memory processing instruction format

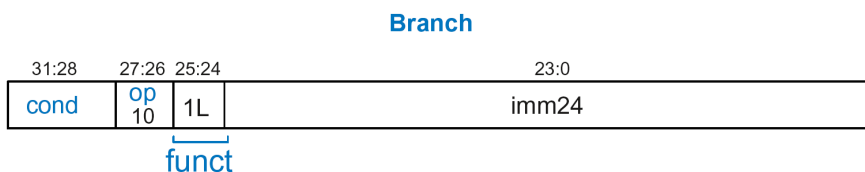


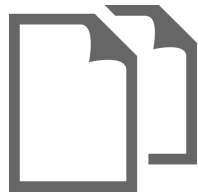
Figure 3: Branch instruction format

**Table B.1 Data-processing instructions**

cmd	Name	Description	Operation
0000	AND Rd, Rn, Src2	Bitwise AND	$Rd \leftarrow Rn \& Src2$
0001	EOR Rd, Rn, Src2	Bitwise XOR	$Rd \leftarrow Rn \wedge Src2$
0010	SUB Rd, Rn, Src2	Subtract	$Rd \leftarrow Rn - Src2$
0011	RSB Rd, Rn, Src2	Reverse Subtract	$Rd \leftarrow Src2 - Rn$
0100	ADD Rd, Rn, Src2	Add	$Rd \leftarrow Rn + Src2$
0101	ADC Rd, Rn, Src2	Add with Carry	$Rd \leftarrow Rn + Src2 + C$
0110	SBC Rd, Rn, Src2	Subtract with Carry	$Rd \leftarrow Rn - Src2 - \bar{C}$
0111	RSC Rd, Rn, Src2	Reverse Sub w/ Carry	$Rd \leftarrow Src2 - Rn - \bar{C}$
1000 ( $S = 1$ )	TST Rd, Rn, Src2	Test	Set flags based on Rn & Src2
1001 ( $S = 1$ )	TEQ Rd, Rn, Src2	Test Equivalence	Set flags based on Rn ^ Src2
1010 ( $S = 1$ )	CMP Rn, Src2	Compare	Set flags based on Rn - Src2
1011 ( $S = 1$ )	CMN Rn, Src2	Compare Negative	Set flags based on Rn + Src2
1100	ORR Rd, Rn, Src2	Bitwise OR	$Rd \leftarrow Rn   Src2$
1101	<b>Shifts:</b>		
$I = 1$ OR ( $instr_{11:4} = 0$ )	MOV Rd, Src2	Move	$Rd \leftarrow Src2$
$I = 0$ AND ( $sb = 00$ ; $instr_{11:4} \neq 0$ )	LSL Rd, Rm, Rs/shamt5	Logical Shift Left	$Rd \leftarrow Rm \ll Src2$
$I = 0$ AND ( $sb = 01$ )	LSR Rd, Rm, Rs/shamt5	Logical Shift Right	$Rd \leftarrow Rm \gg Src2$
$I = 0$ AND ( $sb = 10$ )	ASR Rd, Rm, Rs/shamt5	Arithmetic Shift Right	$Rd \leftarrow Rm \ggg Src2$
$I = 0$ AND ( $sb = 11$ ; $instr_{11:7, 4} = 0$ )	RRX Rd, Rm, Rs/shamt5	Rotate Right Extend	$\{Rd, C\} \leftarrow \{C, Rd\}$
$I = 0$ AND ( $sb = 11$ ; $instr_{11:7} \neq 0$ )	ROR Rd, Rm, Rs/shamt5	Rotate Right	$Rd \leftarrow Rn \text{ ror } Src2$
1110	BIC Rd, Rn, Src2	Bitwise Clear	$Rd \leftarrow Rn \& \sim Src2$
1111	MVN Rd, Rn, Src2	Bitwise NOT	$Rd \leftarrow \sim Rn$

NOP (no operation) is typically encoded as 0xE1A000, which is equivalent to MOV R0, R0.

**Question 14**  
Attached





## Data-processing instructions

Name	Description	Operation
ADD Rd, Rn, Src2	Add (+)	$Rd = Rn + Src2$
SUB Rd, Rn, Src2	Subtract (-)	$Rd = Rn - Src2$
AND Rd, Rn, Src2	Bitwise AND (&)	$Rd = Rn \& Src2$
ORR Rd, Rn, Src2	Bitwise OR ( )	$Rd = Rn   Src2$
EOR Rd, Rn, Src2	Bitwise Exclusive OR (^)	$Rd = Rn \wedge Src2$
BIC Rd, Rn, Src2	Bitwise Clear	$Rd = Rn \& \sim Src2$
MVN Rd, Rn, Src2	Bitwise NOT (~)	$Rd = \sim Rn$
LSL Rd, Rn, Src2	Logical Shift Left (<<)	$Rd = Rn \ll Src2$
LSR Rd, Rn, Src2	Logical Shift Right (>>)	$Rd = Rn \gg Src2$
MOV Rd, Src2	Move (=)	$Rd = Src2$
CMP Rd, Src2	Compare	Set flags (see below) based on $Rd - Src2$

Remember that we can also set condition flags by appending an *S* to the end of our Data-processing instructions.

Name	Description
ADDS Rd, Rn, Src2	Add (as above) <b>and</b> set condition flags
SUBS Rd, Rn, Src2	Subtract (as above) <b>and</b> set condition flags
ANDS Rd, Rn, Src2	Bitwise AND (as above) <b>and</b> set condition flags

## Multiply instructions

Name	Description	Operation
MUL Rd, Rn, Rm	Multiply (*)	$Rd = Rn * Rm$
MULS Rd, Rn, Rm	Multiply (*) <b>and</b> set condition flags	$Rd = Rn * Rm$
MLA Rd, Rn, Rm, Ra	Multiply and Accumulate	$Rd = (Rn * Rm) + Ra$

## Memory instructions

Name	Description	Operation
STR Rd, [Rn, ± Src2]	Store Register	$Mem[Adr] = Rd$
LDR Rd, [Rn, ± Src2]	Load Register	$Rd = Mem[Adr]$

## Branch instructions

Name	Description	Operation
B label	Branch	$PC = (PC + 8) + imm24 \ll 2$
BL label	Branch and Link	$LR = (PC + 8) - 4;$ $PC = (PC + 8) + imm24 \ll 2$
BX Rd	Branch and eXchange	Branch to address pointed to in Rd (used for return)

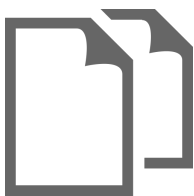
## Condition flags

Flag	Name	Description
N	Negative	Instruction result is negative
Z	Zero	Instruction result is zero
C	Carry	Instruction caused a carry out
V	oVerflow	Instruction caused an overflow

## Condition mnemonics

Mnemonic	Name	CondEx
EQ	Equal	Z
NE	Not Equal	!Z
CS/HS	Carry set / unsigned higher or same	C
CC/LO	Carry clear / unsigned lower	!C
MI	Minus / negative	N
PL	Plus / Positive <i>or</i> zero	!N
VS	Overflow	V
VC	No overflow	!V
HI	Unsigned higher	!Z AND C
LS	Unsigned lower or same	Z OR !C
GE	Signed greater than or equal	!N XOR !V
LT	Signed less than	N XOR V
GT	Signed greater than	!Z AND (!N XOR !V)
LE	Signed less than or equal	Z OR (N XOR V)

**Question 5**  
Attached



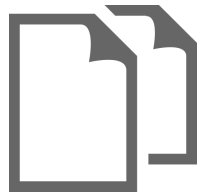
# Theorems

Number	Theorem	Dual	Name
T1	$B \cdot 1 = B$	$B + 0 = B$	Identity
T2	$B \cdot 0 = 0$	$B + 1 = 1$	Null Element
T3	$B \cdot B = B$	$B + B = B$	Idempotency
T4	$(B')' = B$		Involution
T5	$B \cdot B' = 0$	$B + B' = 1$	Complements

#	Theorem	Dual	Name
T6	$B \cdot C = C \cdot B$	$B+C = C+B$	Commutativity
T7	$(B \cdot C) \cdot D = B \cdot (C \cdot D)$	$(B + C) + D = B + (C + D)$	Associativity
T8	$B \cdot (C + D) = (B \cdot C) + (B \cdot D)$	$B + (C \cdot D) = (B+C) (B+D)$	<u>Distributivity</u>
T9	$B \cdot (B+C) = B$	$B + (B \cdot C) = B$	Covering
T10	$(B \cdot C) + (B \cdot \bar{C}) = B$	$(B+C) \cdot (B+\bar{C}) = B$	Combining
T11	$(B \cdot C) + (\bar{B} \cdot D) + (C \cdot D) = (B \cdot C) + (\bar{B} \cdot D)$	$(B+C) \cdot (\bar{B}+D) \cdot (C+D) = (B+C) \cdot (\bar{B}+D)$	Consensus

#	Theorem	Dual	Name
T12	$\overline{B_0 \cdot B_1 \cdot B_2 \dots} = \overline{B_0} + \overline{B_1} + \overline{B_2} \dots$	$\overline{B_0 + B_1 + B_2 \dots} = \overline{B_0} \cdot \overline{B_1} \cdot \overline{B_2} \dots$	DeMorgan's Theorem

**Question 8**  
Attached



```
library IEEE;
  use IEEE.STD_LOGIC_1164.all;

entity fulladder is
  port(
    a, b, cin : in  std_logic;
    sum, cout : out std_logic
  );
end entity fulladder;

architecture dataflow of fulladder is
  signal m : std_logic;
begin
  m    <= a xor b;
  sum  <= m xor cin;
  cout <= (a and b) or (m and cin);
end architecture dataflow;
```

```

library IEEE;
    use IEEE.STD_LOGIC_1164.all;

entity multibit is
    port(
        a, b: in  std_logic_vector(2 downto 0);
        cin : in  std_logic;
        sum : out std_logic_vector(3 downto 0)
    );
end entity multibit;

architecture structural of multibit is
    component fulladder is
        port(
            a, b, cin  : in  std_logic;
            sum, cout  : out std_logic
        );
        signal carry: std_logic_vector(3 downto 0);
begin
    carry(0) <= cin;
    sum(3) <= carry(3);

    S1: fulladder
        port map(
            a => a(0),
            b => b(0),
            cin => carry(0),
            cout => carry(1),
            sum => sum(0));

    S2: fulladder
        port map(
            a => a(1),
            b => b(1),
            cin => carry(1),
            cout => carry(2),
            sum => sum(1));

    S3: fulladder
        port map(
            a => a(2),
            b => b(2),
            cin => carry(2),
            cout => carry(3)
            sum => sum(2));
end architecture structural;

```