

Informasjon

Oppgave	Tittel	Maks poeng	Oppgavetype
---------	--------	------------	-------------

i	Informasjon		Dokument
---	-------------	--	----------

Digital representasjon

Oppgave	Tittel	Maks poeng	Oppgavetype
---------	--------	------------	-------------

1	Digital representasjon 1	2	Flervalg
---	--------------------------	---	----------

2	Digital representasjon 2	3	Flervalg
---	--------------------------	---	----------

Kombinatorisk logikk

Oppgave	Tittel	Maks poeng	Oppgavetype
---------	--------	------------	-------------

3	Fra porter til funksjonsuttrykk	2	Flervalg
---	---------------------------------	---	----------

4	Boolsk algebra 4	3	Flervalg
---	------------------	---	----------

Sekvensiell logikk

Oppgave	Tittel	Maks poeng	Oppgavetype
---------	--------	------------	-------------

5	Kombinatorisk og sekvensiell logikk	2	Flervalg (flere svar)
---	-------------------------------------	---	-----------------------

6	Sekvensielle kretser	12	Paring
---	----------------------	----	--------

Digital design

Oppgave	Tittel	Maks poeng	Oppgavetype
---------	--------	------------	-------------

7	HDL-tilstandsmaskin	10	Flervalg
---	---------------------	----	----------

Digitale byggeblokker

Oppgave	Tittel	Maks poeng	Oppgavetype
---------	--------	------------	-------------

8	Adder	2	Flervalg
---	-------	---	----------

9	Minne	2	Fyll inn tall
---	-------	---	---------------

10	Programmerbar logikk	4	Flervalg
----	----------------------	---	----------

11	Aritmetisk logisk enhet (ALU)	6	Flervalg
----	-------------------------------	---	----------

Oppgave	Tittel	Maks poeng	Oppgavetype
12	Funksjonsskallkonvensjon	2	Flervalg
13	Assemblerprogrammering	10	Sammensatt
14	Assemblerprogrammering 2	2	Fyll inn tall
15	Maskinkode	4	Flervalg

Mikroarkitektur

Oppgave	Tittel	Maks poeng	Oppgavetype
16	Prosessorytelse	2	Flervalg
17	Mikroarkitektur flervalg	2	Flervalg
18	Hvor får vi hazard?	6	Flervalg (flere svar)
19	CPI og pipeline	2	Flervalg
20	CPI til forskjellige mikroarkitektur design	6	Nedtrekk

Minnesystemer

Oppgave	Tittel	Maks poeng	Oppgavetype
21	Cache oppbygning	6	Flervalg
22	Cache miss	8	Flervalg
23	Virtuelt minne	2	Flervalg

i Informasjon

Eksamen IN2060 - Digitalteknikk og datamaskinarkitektur

Eksamensdag: 26. november kl 14:30

Tid for eksamen: 4 timer

Hjelpemidler: Ingen

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 vekten av hver oppgave slik at du kan disponere tiden. Det blir ikke gitt trekk for feil avkryssning.

Lykke til!

1 Digital representasjon 1

Digital representasjon

Gjør om desimaltallet $(54)_{10}$ til et 8 bits binærtall.

Velg ett alternativ

- 00101110
- 00110110
- Ingen av alternativene er korrekte.
- 00111010
- 00100100



Maks poeng: 2

2 Digital representasjon 2

Digital representasjon

Gjør om desimaltallet $(-21)_{10}$ til et 8 bits binærtall på 2'ers komplement form.

Velg ett alternativ

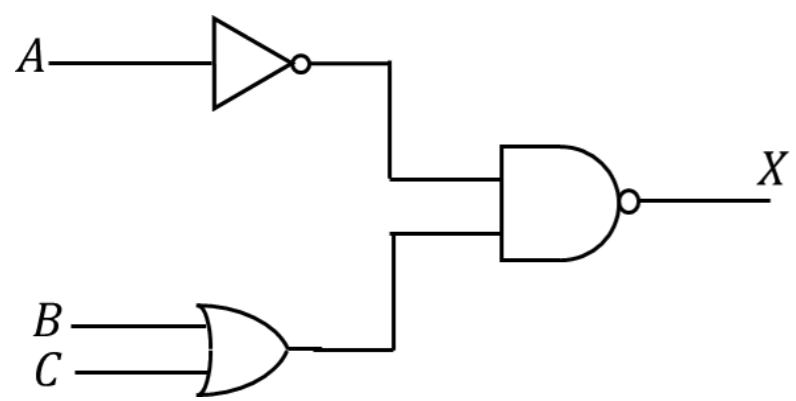
- Ingen av alternativene er korrekte.
- 00011011
- 11011001
- 01011001
- 11101011



Maks poeng: 3

3 Fra porter til funksjonsuttrykk

Hvilket funksjonsuttrykk gjenspeiler portimplementasjonen under?



Velg ett alternativ

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



Maks poeng: 2

4 Boolsk algebra 4

Forenkle følgende uttrykk maksimalt

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

Velg ett alternativ

- $F = (A+B)C$
- $F = A(B+C)$
- $F = A + BC$
- $F = B+C$
- $F = AC+B$



Maks poeng: 3

5 Kombinatorisk og sekvensiell logikk

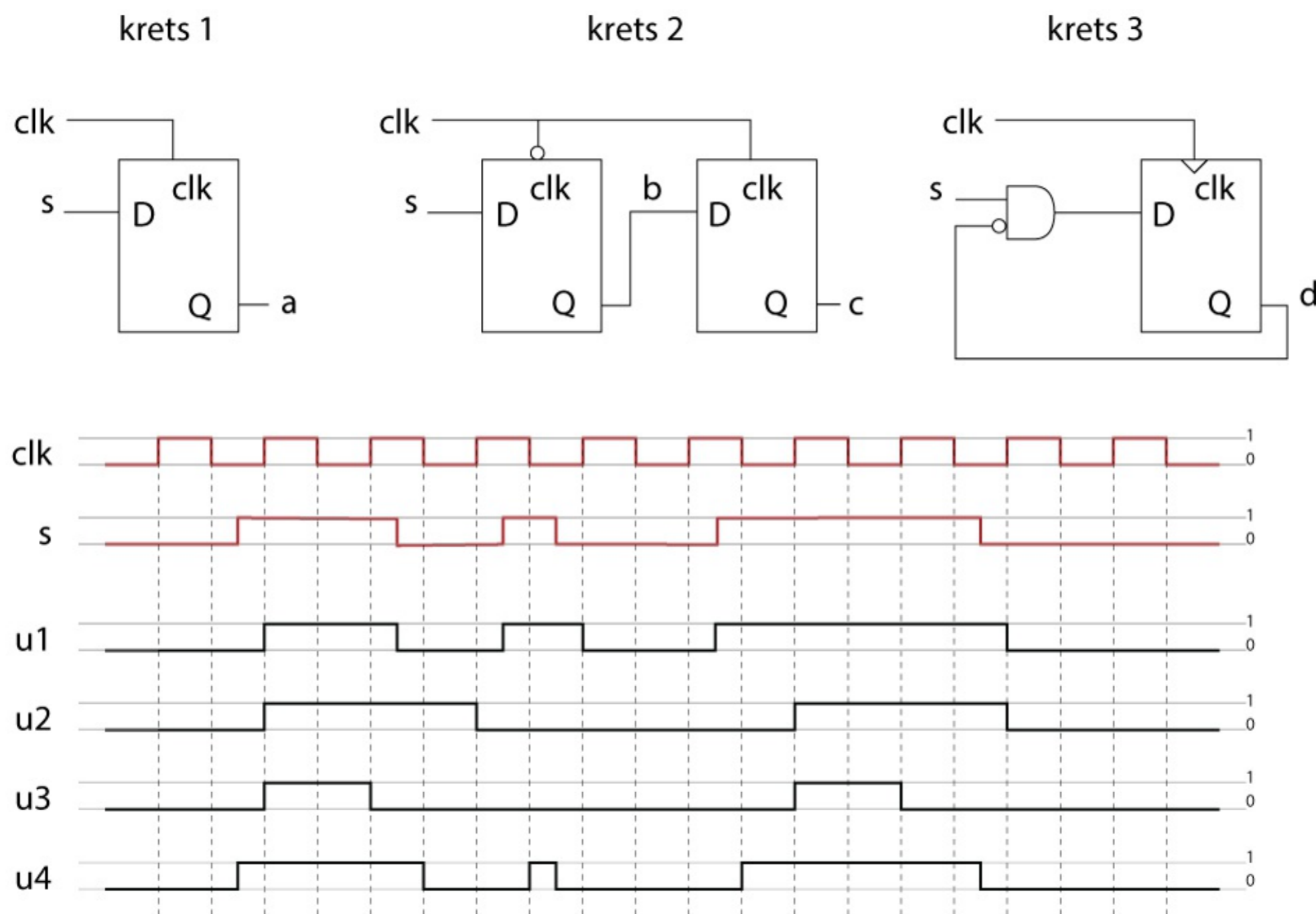
Hvilke to utsagn om kombinatorisk og sekvensiell logikk er **feil**?

Velg de to alternativene som er gale.

- Sekvensiell logikk husker tidligere verdier.
- Sekvensiell logikk har tilbakekoblinger.
- Utgangen til kombinatorisk logikk kan kun forandre verdi på positiv eller negativ klokkeflanke. ✓
- Utgangen til kombinatorisk logikk er kun et resultat av nåværende innganger.
- Utgangen til sekvensiell logikk er et resultat av nåværende og tidligere innganger.
- Kombinatorisk logikk kan inneholde minnelementer. ✓

Maks poeng: 2

6 Sekvensielle kretser



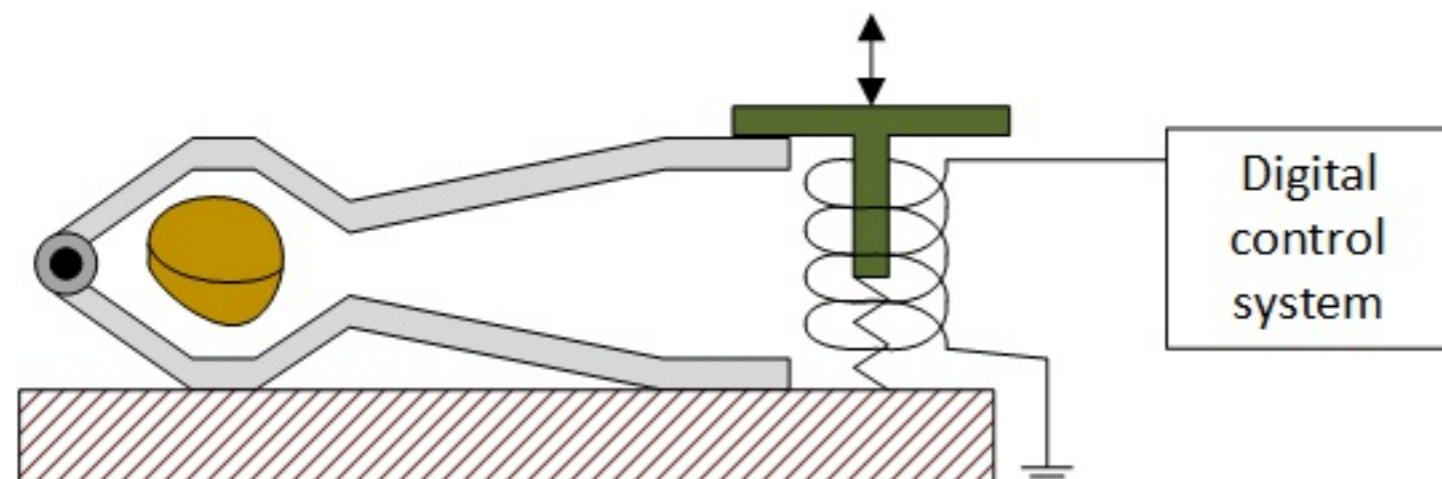
Hver av de tre kretsene over tar inn klokkesignalet clk og inngangsignalet s, som vist i figuren. Anta at a, b, c og d har startverdien 0. Finn ut hvilke av tidsforløpene u1, u2, u3 og u4, som tilsvarer tidsforløpene til signalene a, b, c og d. Du skal ikke ta hensyn til portforsinkelse. **Hint: Merk forskjellen på latcher og flippflopper i illustrasjonen.**

	a	b	c	d
u1	<input type="radio"/> ✓	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
u2	<input type="radio"/>	<input type="radio"/>	<input type="radio"/> ✓	<input type="radio"/>
u3	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/> ✓
u4	<input type="radio"/>	<input type="radio"/> ✓	<input type="radio"/>	<input type="radio"/>

Maks poeng: 12

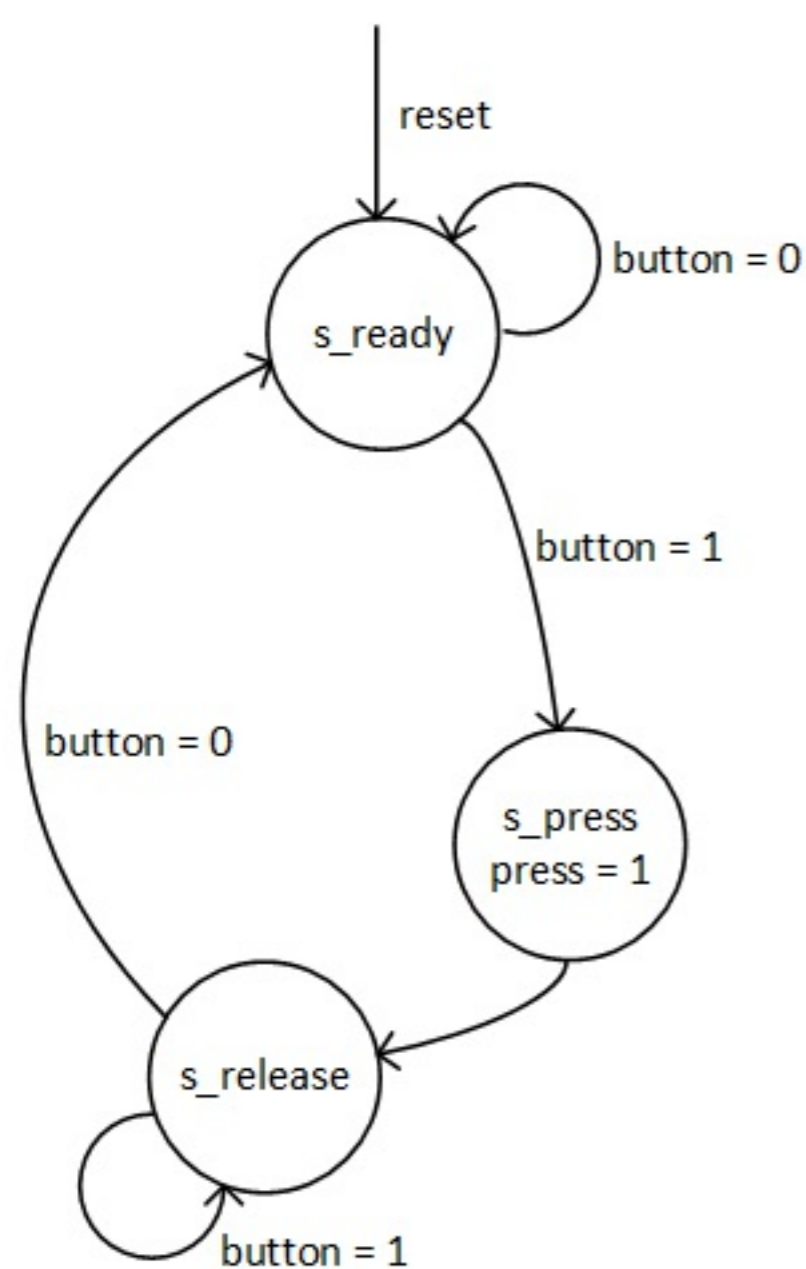
7 HDL-tilstandsmaskin

Du har fått til oppdrag å programmere HDL til en digitalt kontrollert nøtteknekker. Rent fysisk styrer du en magnetisk aktuator med ett signal som når det er aktivt (digital '1') vil trykke nøtteknekkeren sammen slik at nøtten knekkes.



Figur 1: Nøtteknekker system

Vi skal bare gi ett kort signal for å aktivere nøtteknekkeren når man trykker knappen. Den skal ikke holdes nede og trekke mer strøm enn nødvendig. Derfor skal kontrollsystemet implementeres med en tilstandsmaskin.



Figur 2: Diagram tilstandsmaskin.

Den digitale kretsen har et klokkesignal (clk) og et resetsignal (reset) i tillegg knappesignalet (button), og styringssignalet "press".

I denne oppgaven skal du velge alternativer slik at du til sammen får en kode som implementerer tilstandsmaskinen i Figur 2.

Oppgavealternativene med følgekode er i pdf-dokumentet som er lagt ved oppgaven. For hvert spørsmål er det 5 alternativer (A1- A5), hvorav kun ett hører til der det står **<alternativ>**. De alternativene som ikke er riktige har en eller annen form for feil, enten ved at de ikke passer til designet, eller dysfunksjonell kode (*ikke enkelttegn*).

Vær obs på at rekkefølgen alternativene er stilt opp i er tilfeldig!

Entitet

Velg ett alternativ

A1 A2 A4 A3 A5

Klokket prosess**Velg ett alternativ**

A1 A5 A4 A3 A2

Tilstandsmaskin**Velg ett alternativ**

A4 A1 A2 A5 A3

Kombinatorisk output**Velg ett alternativ**

A4 A3 A2 A5 A1

Antall flippfloppe

Tilstandsmaskiner kan kodes på forskjellige måter ut i fra hva de skal brukes til. Ønsker man feilsjekking av tilstander kan det være praktisk å ha én flippflopp for hver tilstand, såkalt "one-hot encoding", ellers kan man benytte gray-kode der bare ett register skifter ved gyldig overgang mellom to tilstander. Hvis man ønsker å bruke færrest mulig lagringselementer kan man ha binærkoding av tilstandene til flippfloppene.

Hva er det minste antall flippfloppe som kan brukes for å implementere tilstandsmaskinen i denne oppgaven?

Velg ett alternativ

1 2 3 4 5

Maks poeng: 10

8 Adder

Under følger fem påstander om fulladdere. Vi forutsetter at antall bit er 32 bit eller mer. Fire av utsagnene stemmer, ett stemmer ikke.

Hvilket utsagn er *stemmer ikke*?

Velg det alternativet som er galt

For en prefiksadder er propageringstiden på formen:

- $K1 + K2 \log_2(\text{antall bit})$

(Der $K1$ og $K2$ er konstanter)

- En carry-lookahead adder vil normalt være raskere enn en ripple-carry-adder

- Ripple-carry adder trenger færre porter enn carry-lookahead og prefiks-adder

For en carry-lookahead er propageringstiden på formen:

- $K1 + K2 \log_2(\text{antall bit})$

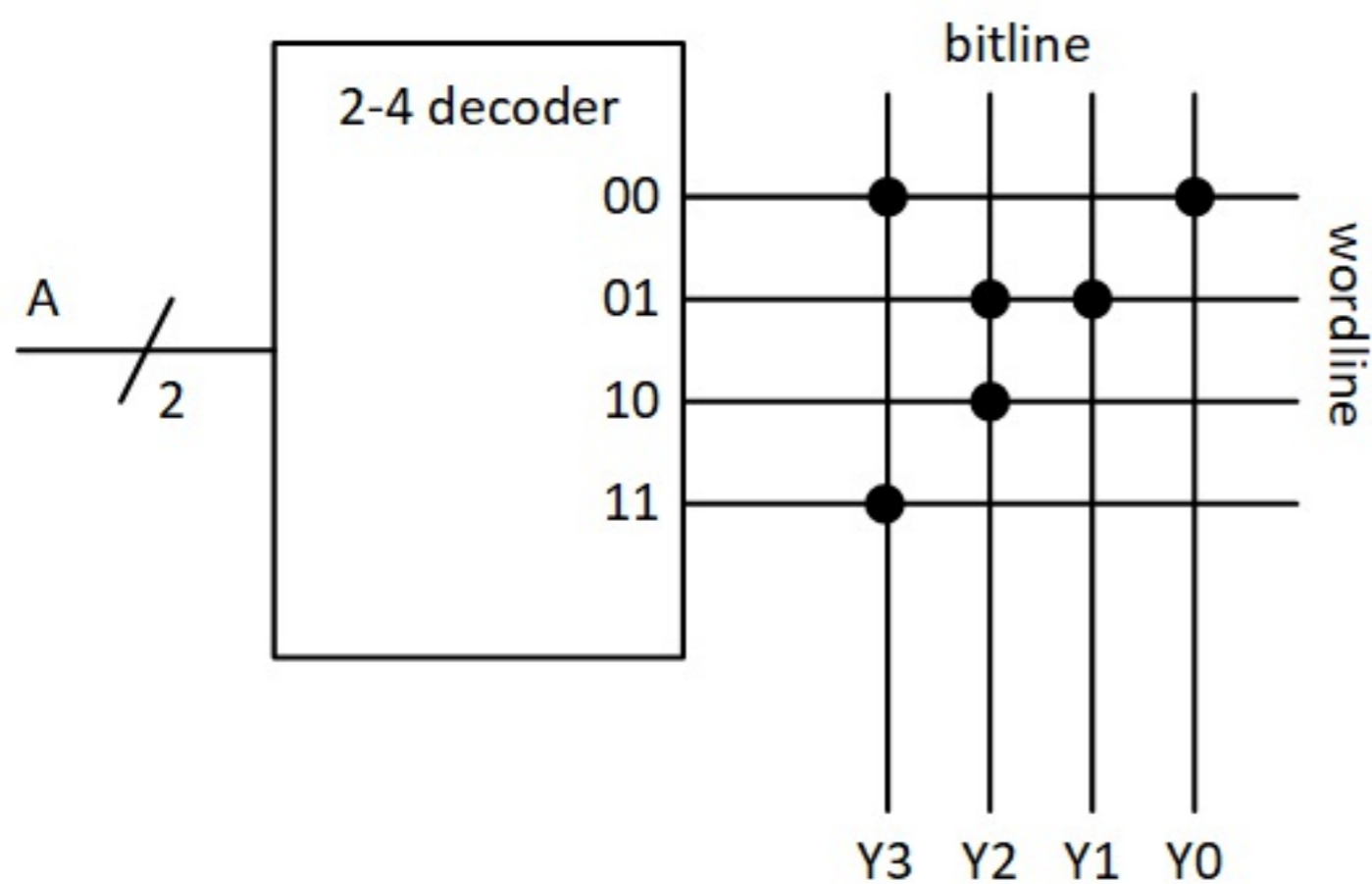


(Der $K1$ og $K2$ er konstanter)

- En prefiksadder vil normalt være raskere enn en ripple-carry-adder

Maks poeng: 2

9 Minne



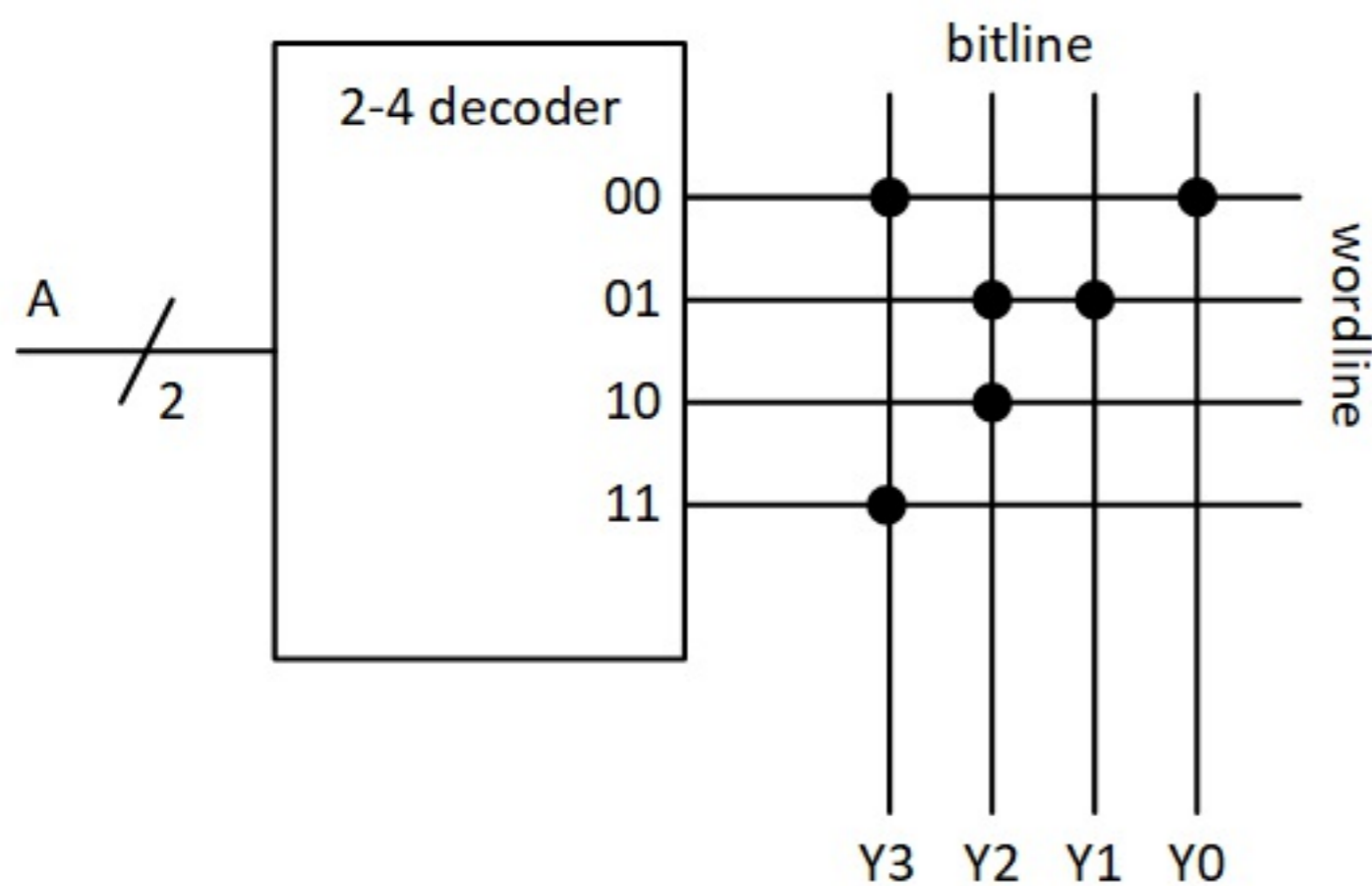
Vi har et ROM minne lik den som er gitt på figuren over. Hver dott representerer logisk '1'.

Hva blir verdien på Y om adressen (a) er 1?

Skriv tallet på vanlig form (**heltall, base 10**): (6).

Maks poeng: 2

10 Programmerbar logikk



En ROM eller oppslagstabell (LUT - look up table) kan brukes både som minne og til å lage logiske funksjoner.

Hvilken logisk funksjon gir Y0?

Velg ett alternativ

- A0 and A1
- (A0 or A1) and ('A0 or A1) and (A1 or 'A1) and ('A0 or 'A1)
- '(A0 and A1)
- (A0 and A1) or ('A0 and A1) or (A0 and 'A1) or ('A0 and 'A1)
- ingen av de øvrige alternativene

✓

Hvilket utsagn er logisk korrekt:

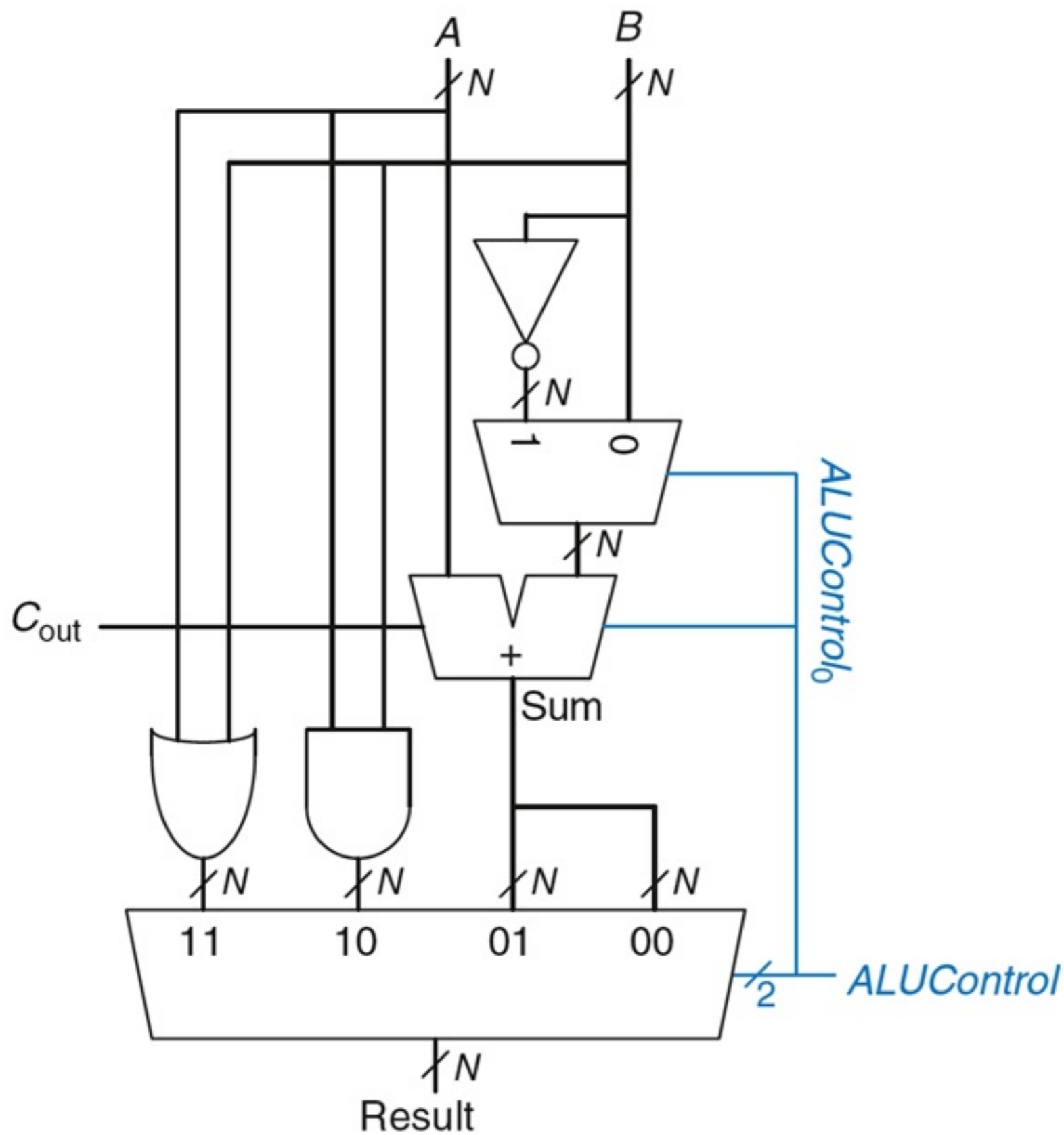
Velg ett alternativ

- Y0 or Y1 = A0 or A1
- Y3 = 'Y2
- Y3 = A0 xor A1
- Y3 and Y2 = 1
- Y3 and Y0 = Y1 or Y2

✓

Maks poeng: 4

11 Aritmetisk logisk enhet (ALU)



Vi har en ALU som vist på figuren over.

Vi setter $A = 0x0A55$ og $B = 0x05AA$

Hva blir Resultatet («Result») når vi setter $ALUcontrol(1 \text{ downto } 0)$ til...

a) "10"

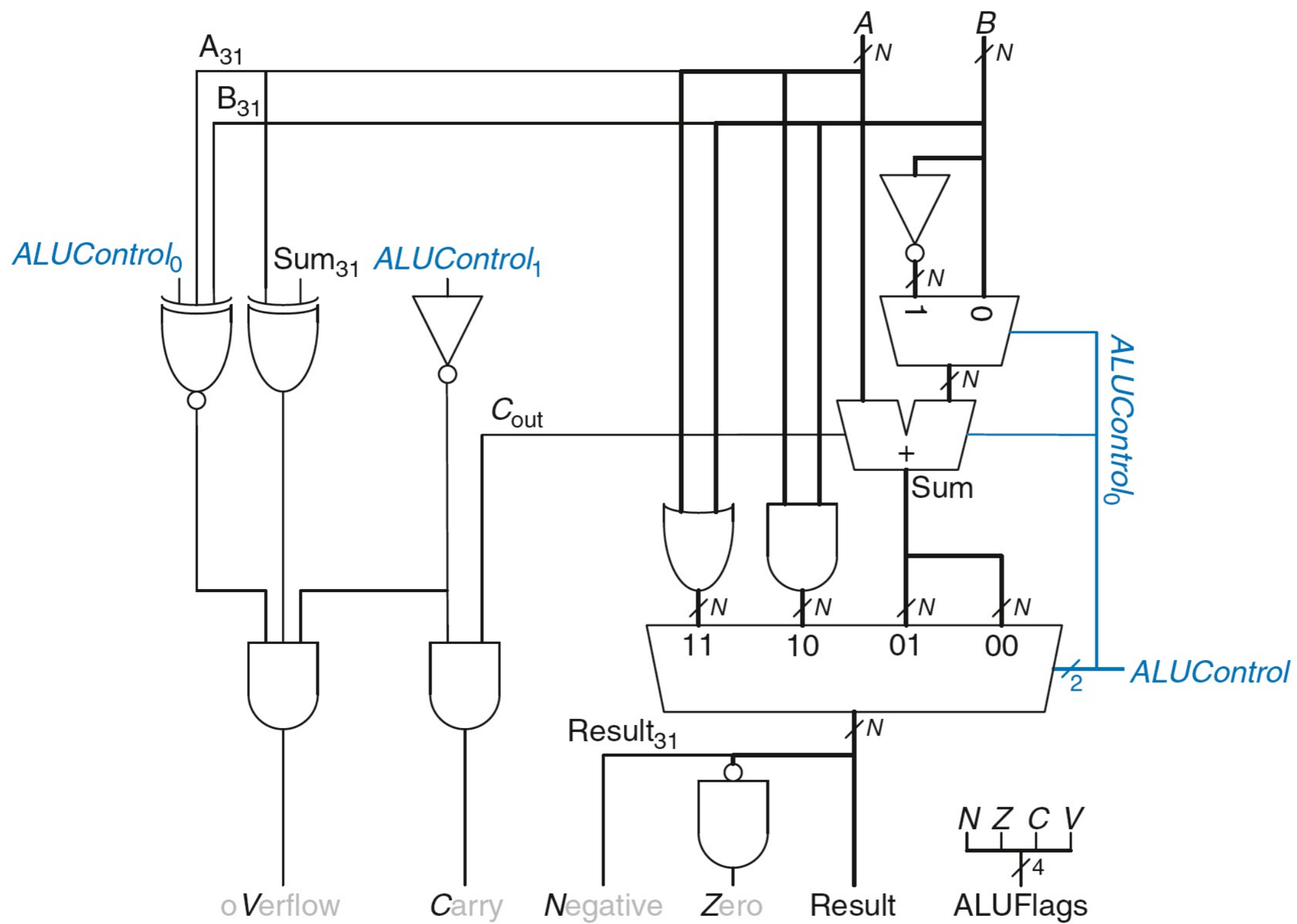
Velg ett alternativ

- 0x0B55
 0xFB55
 0x04AB
 0x0FFF
 0x0000

b) "01"

Velg ett alternativ

- 0x0FFF
 0x0B55
 0xFB55
 0x04AB
 0x0000



Figuren over viser en ALU med kontrollflagg

c) Vi gjør samme operasjon som i a). Hvilke(t) flagg blir satt:

Velg ett alternativ

- | | | | | | |
|-----------------------|-----------------------|----------------------------------|-----------------------|-----------------------|-----------------------|
| V | N | Z | C | alle fire | ingen |
| <input type="radio"/> | <input type="radio"/> | <input checked="" type="radio"/> | <input type="radio"/> | <input type="radio"/> | <input type="radio"/> |

Maks poeng: 6

12 Funksjonskallkonvensjon

Anta at vi, før et funksjonskall på en ARMv4/ARMv7 prosessor, ønsker å beholde verdiene som ligger i R0-R3, hvem er det som har ansvar for å lagre disse verdiene slik at de ikke blir overskrevet?

Velg ett alternativ

- Den som blir kalt ('Callee')
- Operativsystemet gjøre det for oss
- Det er ikke nødvendig
- Den som kaller ('Caller')
- Prosessoren gjør det for oss



Maks poeng: 2

13 Assemblerprogrammering

Vi ønsker å oversette følgende program til ARM assembler. Du kan anta at 'g' ligger i 'R0' og 'h' ligger i 'R1'. Vi antar videre at det eksisterer 'labels' vi kan bruke i hoppinstruksjoner.

```
if(g < 2) {
    h = h + 1;
} else {
    h = h * 2;
}
```

Velg fra nedtrekslistene under for å oversette programmet:

BEGIN:

(CMP R0, R1, ADD R1, R1, #1, BLT END, **CMP R0, #1**)

(BEQ ELSE, BLT BEGIN, ADD R1, R1, #1, **BGT ELSE**)

(**ADD R1, R1, #1**, SUB R0, R1, #2, LSL R1, R1, #1, MUL R1, R1, #2)

(ADD R1, R1, #1, LSL R1, R1, #1, BEQ END, **B END**)

ELSE:

(ADD R1, R1, #2, B BEGIN, SUB R1, R1, #1, **LSL R1, R1, #1**)

END:

Maks poeng: 10

14 Assemblerprogrammering 2

Vi ønsker å oversette følgende program til ARM assembler. Du kan anta at 'g' ligger i 'R0' og 'h' ligger i 'R1'. Vi antar videre at det eksisterer 'labels' vi kan bruke i hoppinstruksjoner.

```
if(g < 2) {  
    h = h + 1;  
} else {  
    h = h * 2;  
}
```

Fyll inn tallet under for å besvare spørsmålet:

Hvis vi bruker betingetkjøring ('Conditional Execution'), hva er minimum antall instruksjoner vi trenger for å oversette programmet til assembler: (3)

Maks poeng: 2

15 Maskinkode

Dekod følgende instruksjon og velg det alternativet som er riktig.

`0xE2821010`

Velg ett alternativ

- Instruksjonen er 'Data Processing' ✓
- Instruksjonen er ugyldig
- Instruksjonen vil sette 'Conditional Flags'
- Instruksjonen er betinget på likhet (f.eks. 'BEQ')
- Instruksjonen bruker **ikke** en immediate

Maks poeng: 4

16 **Prosessorytelse**

Hvilken av alternativene er den beste måten å vurdere ytelsen til en prosessor?

Velg ett alternativ

- Ved å sammenligne klokkehastigheten med strømforbruket.
- Beregnes ved å gange den gjennomsnittlige CPI'en med klokkehastigheten.
- Ytelsen bestemmes kun av klokkehastigheten.
- Vi kan benytte benchmarks som måler visse aspekter av ytelsen. ✓
- Ved å beregne hvor mange instruksjoner prosessoren klarer å utføre iløpet av en bestemt tid.

Maks poeng: 2

17 Mikroarkitektur flervalg

Hvorfor er en pipeline prosessor raskere enn en multicycle prosessor?

Velg ett alternativ

- Multicycle vil typisk kjøre saktere grunnet *stall* og *hazards*.
- Vi får utført flere instruksjoner på samme tid. ✓
- Vi slipper å bruke tilstandmaskiner for å få riktig kontrollsignaler.
- Vi kan benytte oss av *data forwarding* for å slippe å vente på ALU operasjoner.
- Vi kan benytte oss av en raskere klokkehastighet.

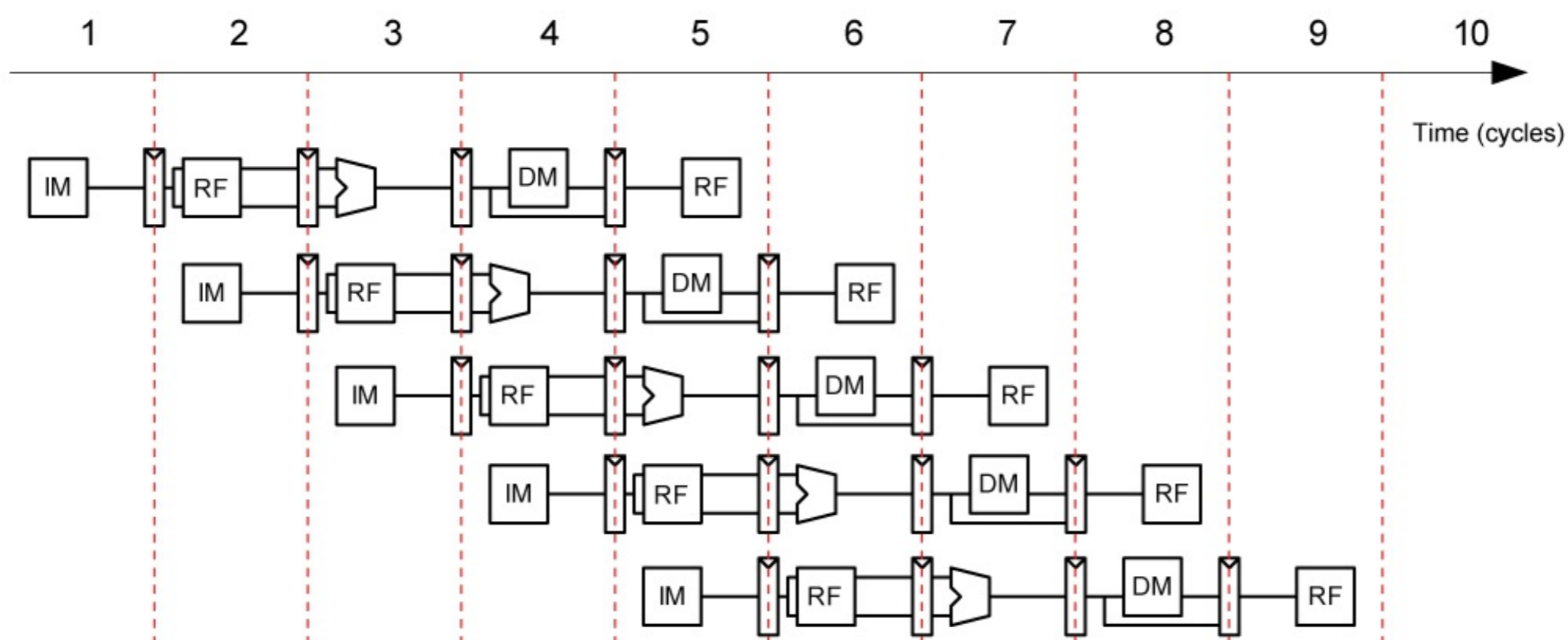
Maks poeng: 2

18 **Hvor får vi hazard?**

Gitt følgende ARM-assemblerprogram:

```
ADD R0, R4, R5
ADD R1, R6, R5
SUB R7, R0, R5
AND R6, R6, R0
ORR R4, R6, R5
```

Hvor oppstår det *data hazard* under kjøring av dette programmet? Anta en 5-steps pipelinet prosessor som illustrert under (tilsvarende som i boka), men uten noen form for hazard-håndtering. Du skal ikke foreta noen *stall*, men kun lokalisere hvor hazard vil oppstå. Her skriver vi til registerfilen i første halvdel av klokkeperioden, og leser av i andre halvdel av klokkeperioden.



Angi de to stedene vi får data hazard.

- Data hazard i sykel 3.
- Data hazard i sykel 4. ✓
- Data hazard i sykel 5.
- Data hazard i sykel 6. ✓
- Data hazard i sykel 7.
- Data hazard i sykel 8.

Maks poeng: 6

19 CPI og pipeline

Hvorfor får vi ikke en CPI på 1 når vi benytter oss av en standard pipeline prosessor? (CPI = Cycles Per Instructions)

Velg ett alternativ

- Siden instruksjonene er delt opp i mindre deler, får vi bare utført en liten del i løpet av en klokkeperiode.
- Vi trenger mer funksjonalitet og kontroll-logikk som øker CPI'en.
- Grunnet data forwarding.
- Grunnet pipeline/sequencing overhead.
- Grunnet data hazards og control hazards. ✓

Maks poeng: 2

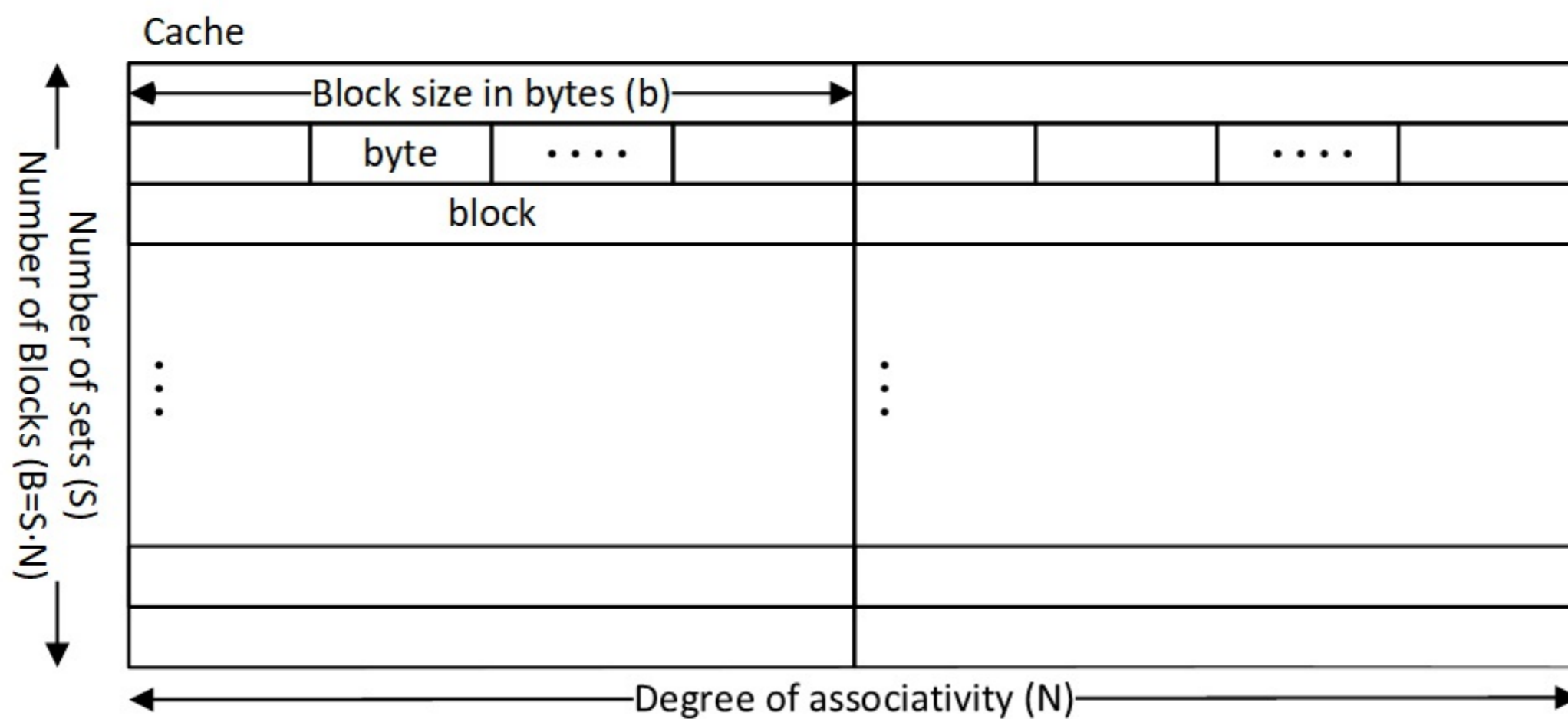
20 CPI til forskjellige mikroarkitektur design

Hva er fornuftige CPI beregninger for følgende mikroarkitekturer?

- Singel-Cycle mikroarkitektur (1, 0.5, 2, Ikke mulig å beregne.)
- En typisk multicycle mikroarkitektur med 5 steg (som beskrevet i boka) (1.12, 4.24, 1, 5)
- Pipeline mikroarkitektur med 6 steg (1.32, 6, 1, 1/6)

Maks poeng: 6

21 Cache oppbygning



Figuren over viser en generisk cache.

Til et fysisk minne på 16MB bruker vi en direkte-mappet cache (N=1) med blokkstørrelse på fire ord, fire byte per ord, og kapasitet på 4kB. Hver byte i minnet har en egen adresse.

a) Hvor mange bit trenger adresser i det fysiske minnet?

Velg ett alternativ

- 4 16 20 10 24
-

b) Hvor mange set vil cachen ha?

Velg ett alternativ

- 2^{24} 2^{20} 2^{12} 2^4 2^8
-

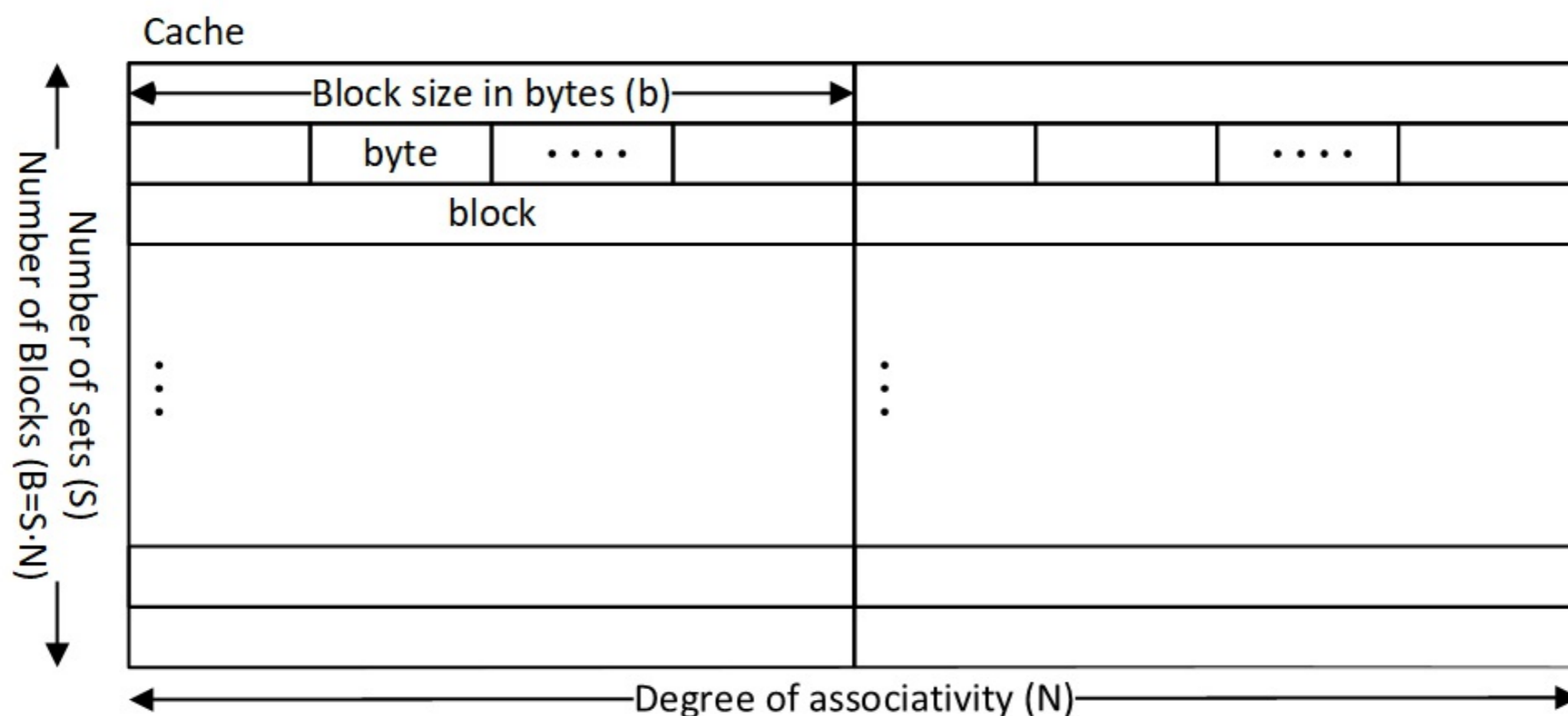
c) Hvor mange bit trenger adresse-tagene til cachen?

Velg ett alternativ

- 24 16 12 14 10
-

Maks poeng: 6

22 Cache miss



Figuren over viser en generisk cache. Til et fysisk minne på 16MB bruker vi en direkte-mappet cache (N=1) med blokkstørrelse på fire ord, fire byte per ord, og kapasitet på 4kB. Hver byte i minnet har en egen adresse.

Vi kjører assembly-koden som vist under:

```

MOV R0, #5
MOV R1, #0x00000000

LOOP  CMP R0, #0
      BEQ DONE
      LDR R4, [R1, #8]
      LDR R3, [R1, #16]
      LDR R5, [R1, #20]
      SUB R0, R0, #1
      B  LOOP

DONE

```

a) Hvor mange tvungne miss («compulsory misses») blir det med denne koden?

Velg ett alternativ

- 15 1 2 5 3
-

b) Hva blir miss raten når vi kjører denne koden?

Velg ett alternativ

- 20% 2/3 1/15 2/15 100%
-

Vi endrer til ny kode, som vist under:

```
MOV R0, #5
MOV R1, #0x00000000
MOV R2, #0x00002000
LOOP  CMP R0, #0
      BEQ DONE
      LDR R4, [R1, #8]
      LDR R3, [R1, #16]
      LDR R5, [R2, #20]
      SUB R0, R0, #1
      B  LOOP
DONE
```

c) Hva blir miss raten med den nye koden?

Velg ett alternativ

- 11/15 1/15 4/15 2/15 20%
-

Vi bytter cachen til en (N=2) 2-way set associative cache med blokkstørrelse på 4 ord og samme totale kapasitet.

d) Hva blir miss raten til den siste koden med det nye cache-systemet?

Velg ett alternativ

- 33,3% 4/15 11/15 2/15 3/15
-

Maks poeng: 8

23 Virtuelt minne

Hvilket av følgende utsagn er korrekt om virtuelt minne?

Velg ett alternativ

- Virtuelle adresser gir programmer alltid tilgang til samme fysiske adresse
- Kompilatorer og assembler-programmerere trenger ikke å holde styr på virtuelle adresser
- Den som programmerer må alltid kjenne den fysiske minneadressen
- Virtuelle adresser kan brukes til å hindre at to programmer får tilgang til samme fysiske adresse ✓
- Cacher lagrer bare virtuelle adresser

Maks poeng: 2

Question 13
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) and set condition flags
SUBS Rd, Rn, Src2	Subtract (as above) and set condition flags
ANDS Rd, Rn, Src2	Bitwise AND (as above) and set condition flags

Multiply instructions

Name	Description	Operation
MUL Rd, Rn, Rm	Multiply (*)	$Rd = Rn * Rm$
MULS Rd, Rn, Rm	Multiply (*) and 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 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) and set condition flags
SUBS Rd, Rn, Src2	Subtract (as above) and set condition flags
ANDS Rd, Rn, Src2	Bitwise AND (as above) and set condition flags

Multiply instructions

Name	Description	Operation
MUL Rd, Rn, Rm	Multiply (*)	$Rd = Rn * Rm$
MULS Rd, Rn, Rm	Multiply (*) and 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 15
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

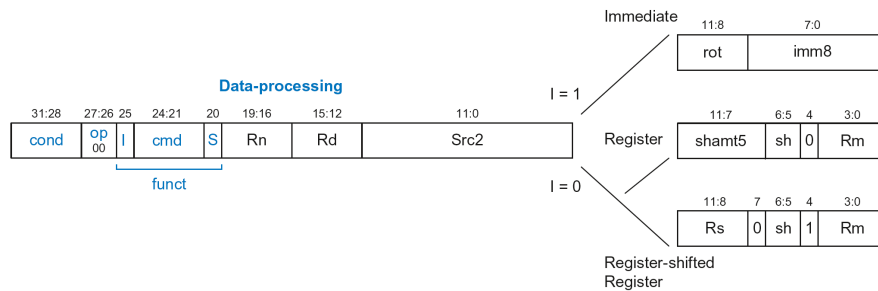


Figure 1: Data processing instruction format

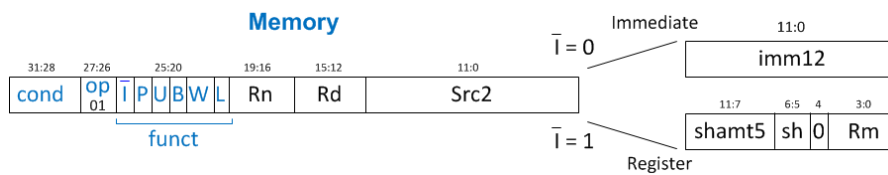


Figure 2: Memory processing instruction format

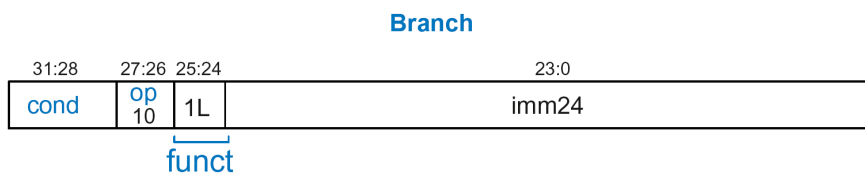


Figure 3: Branch instruction format

Question 4
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 7
Attached



```
library IEEE;  
use IEEE.std_logic_1164.all;
```

```
-- Entity  
entity nut_cracker is  
<alternativ>  
end entity nut_cracker;
```

```
A1:  
port(  
    button      : in  std_logic;  
    press, release : out std_logic  
);  
  
A2:  
port(  
    clk      : signal;  
    reset    : signal;  
    button   : signal;  
    press    : signal  
);  
  
A3:  
port(  
    clk, reset : out std_logic;  
    press, button : in  std_logic  
);  
  
A4:  
port(  
    clk      : in  std_logic;  
    reset    : in  std_logic;  
    button   : in  std_logic;  
    press    : out std_logic  
);  
  
A5:  
port(  
    clk, reset      : in  signal;  
    reset button, release : out signal  
);
```

```
-- Declarations
architecture rtl of nut_cracker is
    type state_type is (s_ready, s_press, s_release);
    signal current_state, next_state : state_type;

Begin
-- Clocked process
<alternativ>
```

```
A1:
process (clk, reset) is
begin
    if reset = '1' then
        current_state <= s_ready;
    elsif rising_edge(clk) then
        current_state <= next_state;
    end if;
end process;
```

```
A2:
process (clk) is
begin
    if clk = 1 then
        current_state <= next_state;
    end if;
end process;
```

```
A3:
process (current_state) is
begin
    if reset = '1' then
        current_state <= s_ready;
    elsif rising_edge(clk) then
        current_state <= next_state;
    end if;
end process;
```

```
A4:
process (clk, reset) is
begin
    if reset = '1' then
        s_ready <= current_state;
        s_press <= next_state;
    elsif rising_edge(clk) then
        s_press <= current_state;
        s_release <= next_state;
    end if;
end process;
```

```
A5:
process (clk, reset) is
begin
    if reset = '1' then
        current_state <= s_ready;
    elsif clk(1) then
        next_state <= s_press;
    elsif clk(2) then
        next_state <= s_release;
    elsif clk(3) then
        next_state <= s_ready;
    end if;
end process;
```

```
-- State machine process
```

```
process(all) is  
begin  
  <alternativ>  
end process;
```

```
A1:
```

```
case s_ready is  
  when current_state =>  
    if button then s_press;  
    else s_ready;  
    end if;  
  when next_state =>  
    if button then s_release;  
    else s_ready;  
  when others => s_ready;  
case s_press is  
  when current_state =>  
    if button then s_press;  
    else s_release;  
    end if;  
  when next_state =>  
    if button then s_ready;  
    else s_release;  
  when others => s_ready;  
end case;
```

```
A2:
```

```
case current_state is  
  when s_ready =>  
    if button then  
      next_state => s_press;  
    else  
      next_state => s_ready;  
    end if;  
  when s_press =>  
    next_state => s_release;  
  when s_release =>  
    if button then  
      next_state => s_release;  
    else  
      next_state => s_ready;  
    end if;  
  when others => next_state <= s_ready;  
end case;
```

```
A3:
```

```
current_state <= next_state;
```

```
A4:
```

```
case current_state is  
  when s_ready =>  
    release <= '1';  
    if button then  
      next_state => s_press;  
    else  
      next_state => s_ready;  
    end if;  
  when s_press =>  
    press <= '1';  
    next_state => s_release;  
  when others => next_state <= s_ready;  
end case;
```

```
A5:
```

```
case state_type is  
  when s_ready =>  
    if button then s_press;  
    else s_ready;  
    end if;  
  when s_press => s_release;  
  when s_release =>  
    if button then s_release;  
    else s_ready;  
    end if;  
  when others => s_ready;  
end case;
```

```
-- Combinatorial output
<alternativ>
end architecture rtl;
```

```
A1:
  press when s_press;
  release when s_release;

A2:
  if button then press <= '1';
  else press <= '0';

A3:
  press <= '1' when (current_state = s_press);
  release <= '1' when (current_state = s_release);

A4:
  current_state <= next_state;

A5:
  press <= '1' when current_state = s_press else '0';
```