Information

Oppgave	Tittel	Poeng	Oppgavetype
i	Informasjon		Informasjon eller ressurser

Digital representation

Oppgave	Tittel	Poeng	Oppgavetype
1	Digital representation 1	2	Flervalg
2	Digital representasjon 2	2	Fyll inn tall
3	2'er komplement	2	Fyll inn tall

Combinational logic

Oppgave	Tittel	Poeng	Oppgavetype
4	Boolske porter	2	Flervalg
5	Boolsk algebra	3	Flervalg

Sequential logic

Oppgave	Tittel	Poeng	Oppgavetype
6	Kombinatorisk og sekvensiell logikk	2	Flervalg (flere svar)
7	Sekvensielle kretser	12	Nedtrekk

Digitale byggeblokker 1

Oppgave	Tittel	Poeng	Oppgavetype
8	LUT1-1	2	Plasser i tekst

9	LUT1-2	1	Nedtrekk
10	LUT1-3	1	Nedtrekk

Digitale byggeblokker 2

Oppgave	Tittel	Poeng	Oppgavetype
11	LUT2-1	2	Plasser i tekst
12	LUT2-2	1	Nedtrekk
13	LUT2-3	1	Nedtrekk

ALU

Oppgave	Tittel	Poeng	Oppgavetype
14	ALU	6	Flervalg

HDL

Oppgave	Tittel	Poeng	Oppgavetype
15	Kritisk sti multiplekser	1	Fyll inn tall
16	Kritisk sti multiplekser 2	3	Fyll inn tall
17	Kritisk sti fulladder	1	Fyll inn tall
18	Kritisk sti ALU-1	1	Flervalg
19	Kritisk sti, ALU-2	3	Fyll inn tall

Computer Architecture

Oppgave	Tittel	Poeng	Oppgavetype
20	ARM-maskinkode	2	Flervalg

21	Branch Target Address	2	Fyll inn tall
22	Tolking av maskinkode	6	Nedtrekk
23	Oversette til assemblerkode	6	Nedtrekk
24	Tolke assemblerkode	3	Flervalg
25	Arm funksjonskallkonvensjonen	2	Flervalg

Mikroarkitektur

Oppgave	Tittel	Poeng	Oppgavetype
26	Mikroarkitektur og ytelse	2	Flervalg
27	Mikroarkitektur og ytelse	3	Fyll inn tall
28	Data-forwarding	2	Flervalg
29	Pipeline	6	Nedtrekk

Minnesystemer

Oppgave	Tittel	Poeng	Oppgavetype
30	Cache-1	6	Fyll inn tall
31	Cache-2	6	Fyll inn tall
32	VM	6	Fyll inn tall

Informasjon

Skriftlig eksamen
IN2060 - Digitalteknikk og datamaskinarkitektur
Høst 2024

Tid for eksamen: 9. desember kl. 09:00 til 9. 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 skrus 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 avkrysning.

Lykke til!

1 Digital representation 1

Digital representasjon

Gjør om desimaltallet $(25)_{10}$ til et 8 bits binærtall.

Velg ett alternativ

- Ingen av alternativene er korrekte.
- 00011010
- 00111011
- 00011001
- 00101001

Maks poeng: 2

2 Digital representasjon 2

Digital representasjon

Gjør om det **hexadesimale** tallet $(D4)_{16}$ til desimaltall. [ ] .

Maks poeng: 2

3 2'er komplement

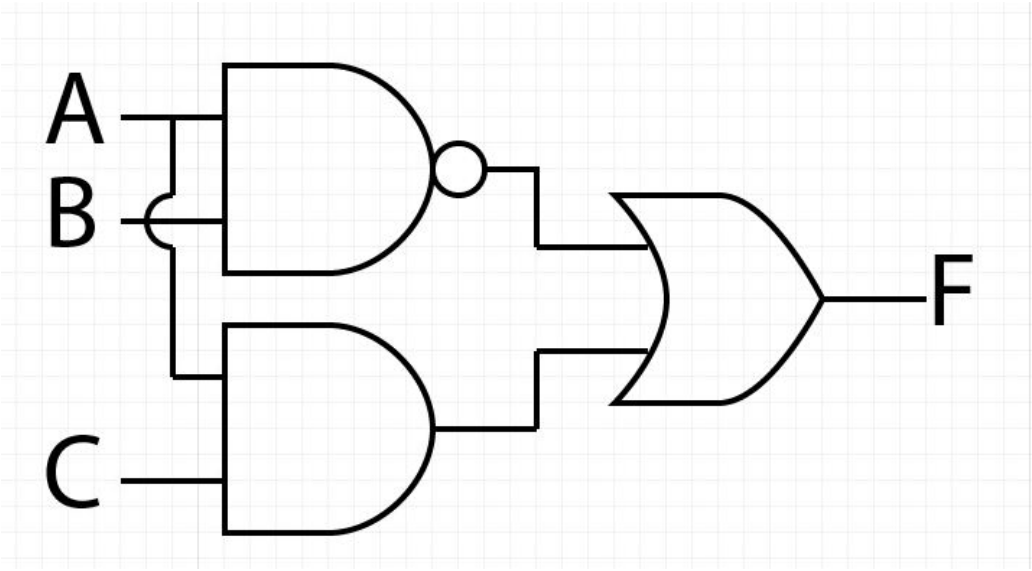
Digital representasjon

Gjør om **toerkomplement**-tallet $(11101101)_2$ til desimaltall : [ ] .

Maks poeng: 2

4 Boolske porter

Hvilket funksjonsuttrykk gjenspeiler portimplementasjonen under?



Velg ett alternativ

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

Maks poeng: 2

5 Boolsk algebra

Forenkle følgende utrykk maksimalt

F = (A+B′)·(A+B)
Velg ett alternativ

- F = A+B'
- F = A
- F = B
- Ingen av alternativene er korrekte
- F = B′

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.

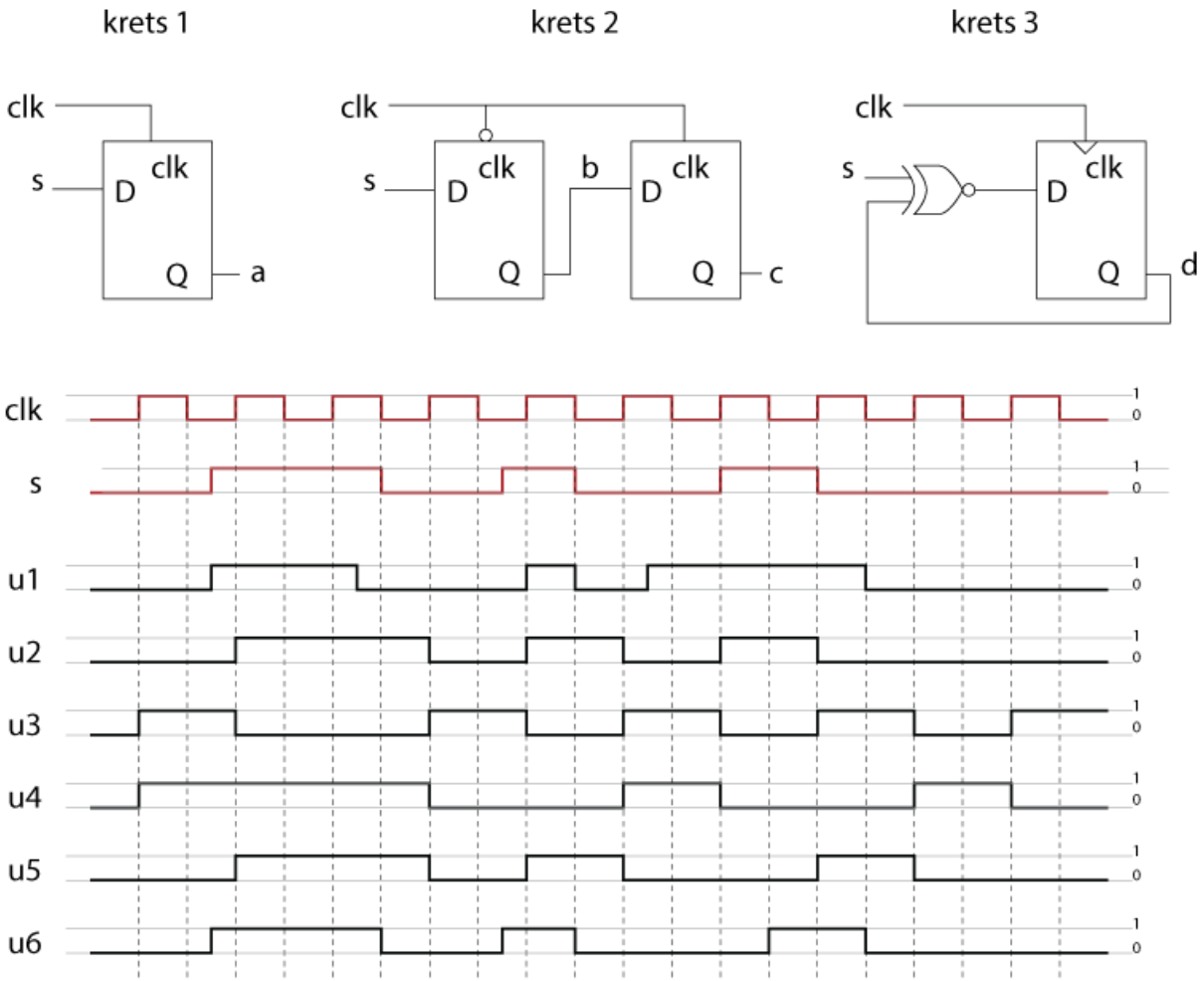
- [ ] En halv-adder og en full-adder er begge eksempler på sekvensielle kretser.
- [ ] Vi kan konstruere sekvensielle kretser med logiske porter, uten noen form for minneelementer.
- [ ] Flip-flop'er og latch'er er vanlige å bruke i kombinatoriske kretser.
- [ ] I kombinatorisk logikk avhenger utgangen kun av de nåværende inngangene og ikke av tidligere innganger.
- [ ] Kombinatoriske kretser må alltid inkludere tilbakekoblinger for å fungere korrekt.
- [ ] Utgangen til en sekvensiell logisk krets kan endre seg basert på tidligere innganger og klokkepulsen

---

Maks poeng: 2

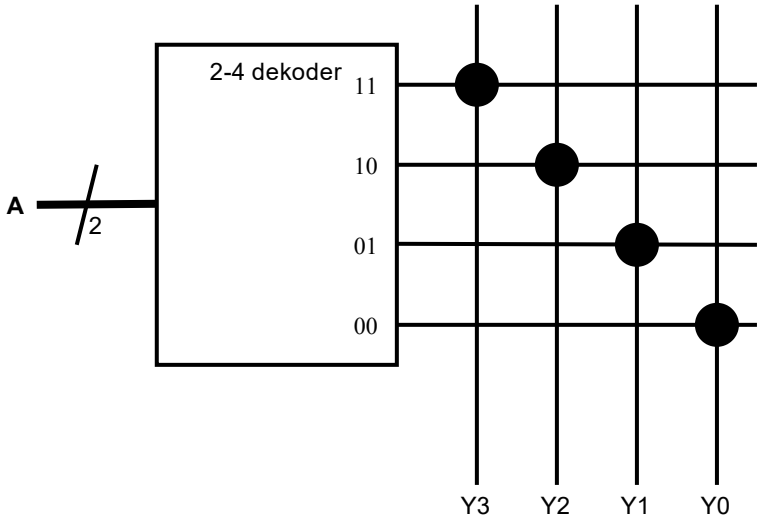
7 Sekvensielle kretser

Hver av de tre kretsene under tar inn klokkesignalet **clk** og inngangsignalet **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. **Studer kretsene nøye og merk forskjellen på latcher og flippflopper i illustrasjonen.**



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

Maks poeng: 12



8 LUT1-1

Hva blir formelen for hver av utgangene til oppslagstabellen?

Alle felt skal fylles. Der to felt følger etter hverandre skal eventuelle blanke brukes lengst til høyre.
Eks: Y_{eks} = **NOT** A_1 **XOR** A_0 :

$Y_{eks} =$ [NOT] A_1 [XOR] [ ] A_0

Sett inn riktig alternativ.

Hjelp

[NOT] [XOR] [ ] [AND] [OR]

$Y_3 =$ [ ] A_1 [ ] [ ] A_0

$Y_2 =$ [ ] A_1 [ ] [ ] A_0

$Y_1 =$ [ ] A_1 [ ] [ ] A_0

$Y_0 =$ [ ] A_1 [ ] [ ] A_0

Maks poeng: 2

9 LUT1-2

Hva blir utgangen med følgende innganger..?

A = 0x0 gir **Y** = Velg alternativ (ingen av alternativene, 0x0, 0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7, 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF).

A = 0x3 gir **Y** = Velg alternativ (ingen av alternativene, 0x0, 0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7, 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF).

Maks poeng: 1

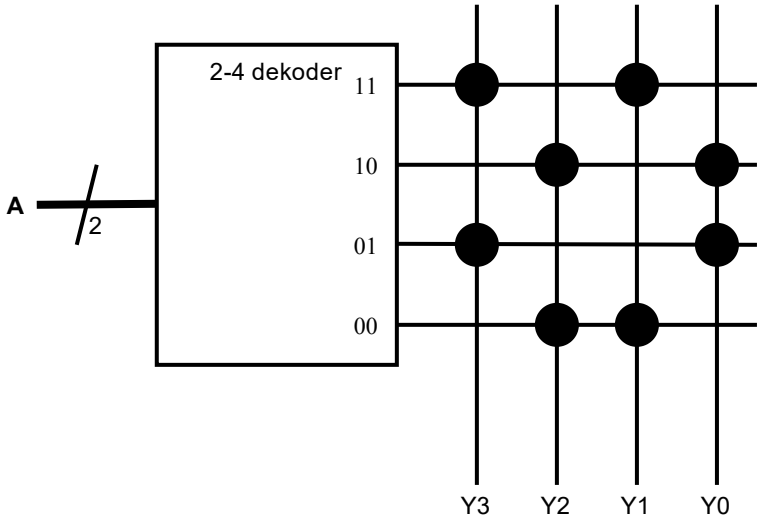
10 LUT1-3

Hva må inngangen være, dersom utgangen er som følger?

Y = 0x4 gir **A** = Velg alternativ (ingen løsning, 0x0, 0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7, 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF).

Y = 0x6 gir **A** = Velg alternativ (ingen løsning, 0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7, 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF).

Maks poeng: 1



11 LUT2-1

Hva blir formelen for hver av utgangene til oppslagstabellen?

Sett inn riktig alternativ. Bruk det tomme alternativet dersom det ikke skal stå noe i et felt.
Eks: Y_{eks} = **<A_1> <XOR>** A_0

Hjelp

OR | A1 | XOR | XNOT | XNOR | AND | A0 | NOT

Y_3 = [ ]

Y_2 = [ ] [ ]

$Y_1 = A_1$ [ ] A_0

$Y_0 = A_1$ [ ] A_0

Maks poeng: 2

12 LUT2-2

Hva blir utgangen med følgende innganger..?

A = 0x0 gir **Y** = Velg alternativ (ingen av alternativene, 0x0, 0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7, 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF)

A = 0x3 gir **Y** = Velg alternativ (ingen av alternativene, 0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7, 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF)

Maks poeng: 1

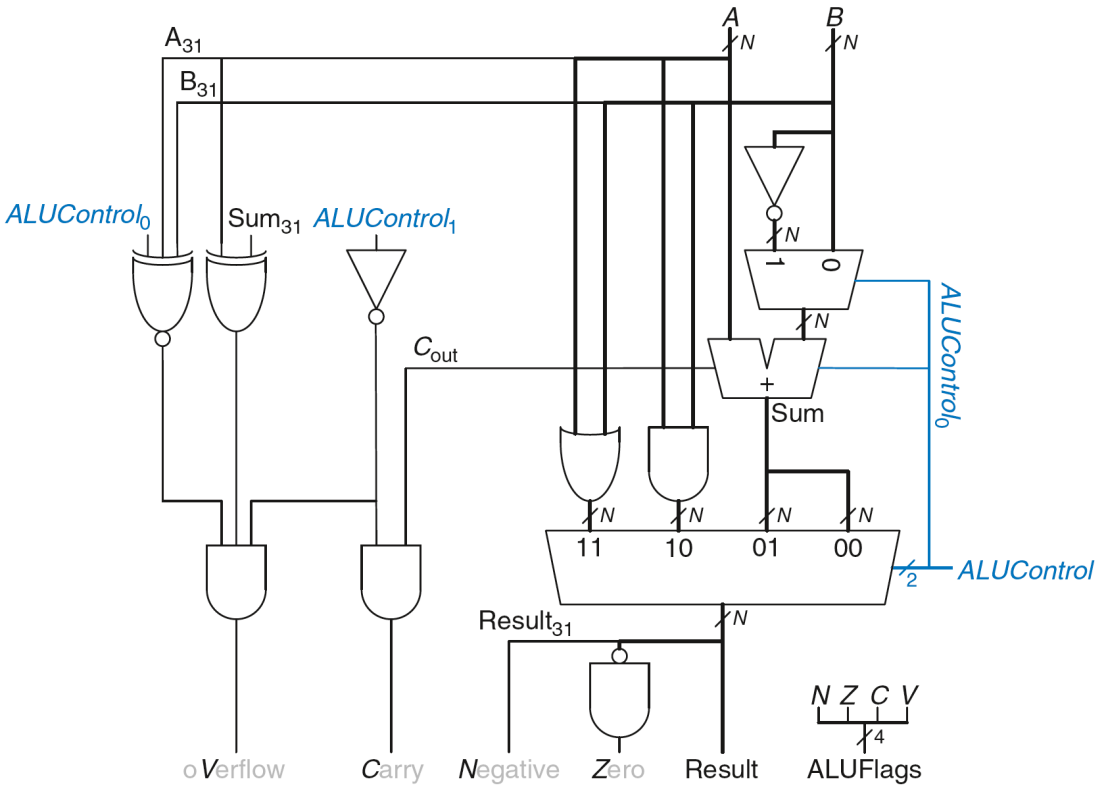
13 LUT2-3

Hva må inngangen være, dersom utgangen er som følger?

Y = 0x4 gir **A** = Velg alternativ (ingen løsning, 0x0, 0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7, 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF).

Y = 0x6 gir **A** = Velg alternativ (ingen løsning, 0x0, 0x1, 0x2, 0x3, 0x4, 0x5, 0x6, 0x7, 0x8, 0x9, 0xA, 0xB, 0xC, 0xD, 0xE, 0xF).

Maks poeng: 1


A
B
N
A31
B31
ALUControl0
Sum31
ALUControl1
1
0
Cout
+
Sum
ALUControl0
11
10
01
00
2
ALUControl
Result31
N Z C V
4
oVerflow
Carry
Negative
Zero
Result
ALUFlags

14 ALU

Vi har en ALU som vist på figuren (til venstre)
Vi setter A = 0x4A50 og B = 0x05A2

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

a) "10"

Velg ett alternativ

0xFFFF 0x4FF2 0x0000 0x04AB 0x2004

b) "11
Velg ett alternativ

0x2004 0x4FF2 0x04AB 0x44AE 0xFFFF

c) "00"
Velg ett alternativ

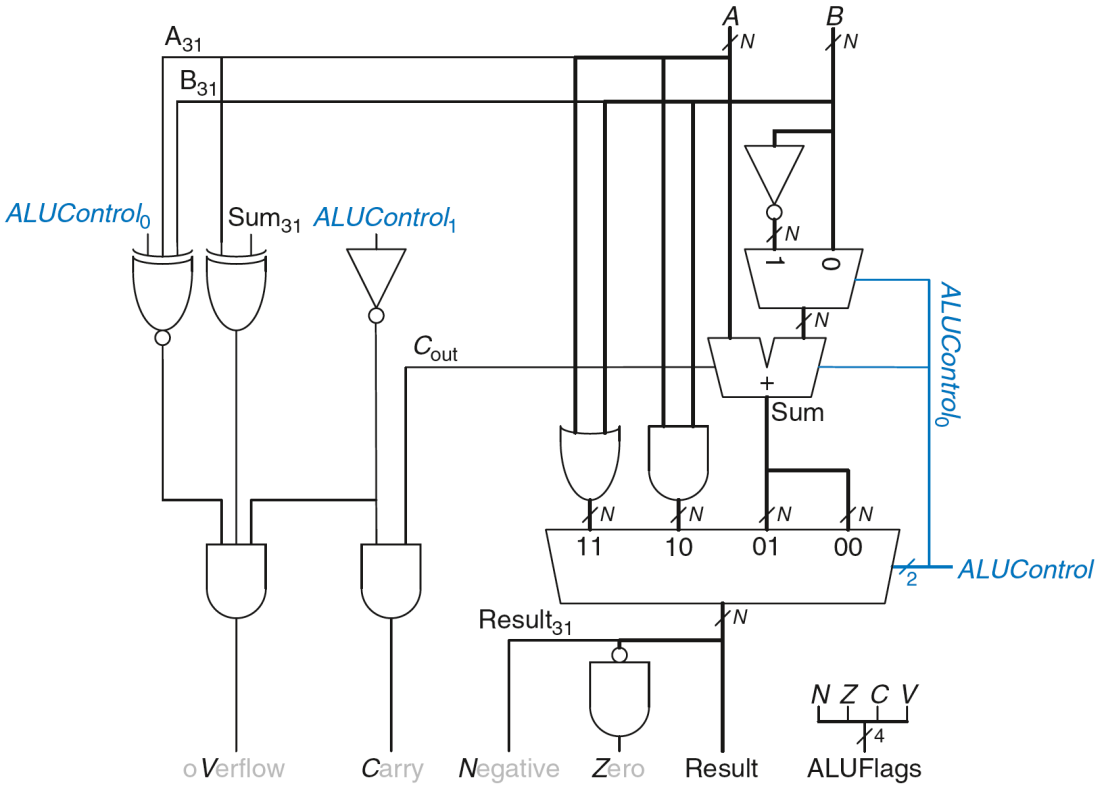
0x2004 0x0000 0xFFFF 0x4FF2 0x44AE

Vi bruker samme data, A = 0x4A50 og B = 0x05A2, og setter ALUkontroll til "01"

d) Hvilke(t) flagg blir satt
Velg ett alternativ

alle fire N V ingen Z C

Maks poeng: 6



Figuren viser en ALU med kontrollsignal. (Brukes i to av oppgavene om kritisk sti)

For oppgavene om kritisk sti gjelder følgende:
Propageringsforsinkelsen til hver inverter er på 10ps.
Propageringsforsinkelsen til hver to-input port er på 50 ps.
All logikk bygges med to-input porter.
For eksempel vil tre-inputs AND-porten som brukes til å beregne overflow lages med to to-input AND-porter.
Rekkefølgen flerinput-porter settes opp skal gi kortest mulig kritisk sti.
Mentepropageringen til addisjonskretsen er av typen "*ripple-carry*".

15 Kritisk sti multiplekser

En multiplekser som velger mellom A og B er bygd opp med følgende VHDL kode:

```
Z <= (A and S) or (B and not S);
```

Hva blir propageringsforsinkelsen til denne multiplekseren?

[ ] ps.

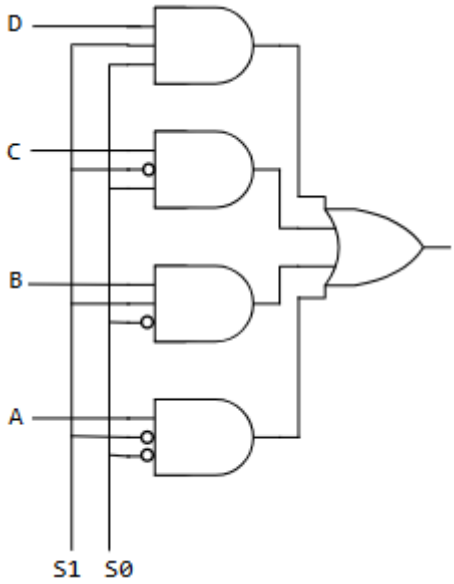
Maks poeng: 1

16 Kritisk sti multiplekser 2

En 4-input multiplekser som velger mellom en av inngangene (A-D) kan skrives med følgende VHDL kode:

```
with S select Z <=
  D when "11",
  C when "10",
  B when "01",
  A when others;
```

Med flerinputsporter kan den implementeres slik:



Hvis vi skal bruke to-input porter, så blir det mer sammensatt.

Hvor mange AND-porter får vi dersom vi bare bruker to-input porter?

[ ] .

Hvor mange OR-porter blir det dersom vi bare bruker to-input porter?

[ ] .

Hva blir portforsinkelsen til kritisk sti gjennom denne multiplekseren?
Husk å ta med invertere i beregningen)

[ ] ps.

Maks poeng: 3

17 Kritisk sti fulladder

Hver fulladder i kretsen er bygget opp med følgende VHDL kode:

```
sum <= A xor B xor Cin;
Cout <= ((A xor B) and Cin) or (A and B);
```

Hva blir den maksimale propageringsforsinkelsen til én fulladder?

[ ] ps.

Maks poeng: 1

18 Kritisk sti ALU-1

Hvor starter og stopper den kritiske stien i ALUen?
Velg ett alternativ:

- Fra A eller B til N
- Fra A eller B til Result
- Fra B til Z
- Fra ALUcontrol til C
- Fra ALUcontrol til Sum31

Maks poeng: 1

19 Kritisk sti, ALU-2

Hva blir maksimal portforsinkelse gjennom addisjonskretsen fra **A til Sum**?

ps.

Tips: *Husk at "**Sum**" er utgangen til addisjonskretsen, ikke hele ALUen*

Hva blir portforsinkelsen til kritisk sti gjennom **hele ALUen**?

ps.

Tips: *Husk at kritisk sti for ALUen ikke nødvendigvis tar samme vei som for hver enkelt krets i de øvrige oppgavene.*

Maks poeng: 3

20 ARM-maskinkode

Hva kjennetegner ARM-maskinkode sammenlignet med andre maskinkoder, som x86?
Velg ett alternativ:

- ARM-maskinkode bruker utelukkende 64-bit instruksjoner, mens x86 bruker 32-bit instruksjoner.
- ARM-maskinkode kan kjøre på alle typer prosessorer uten tilpasninger, uavhengig av produsent.
- ARM-maskinkode krever betydelig mer strøm enn x86-maskinkode, og er derfor upraktisk for mobile enheter.
- ARM-maskinkode er basert på en RISC (Reduced Instruction Set Computer) arkitektur, som bruker færre og enklere instruksjoner enn x86.
- ARM-maskinkode kan bare brukes på mobile enheter, mens x86 brukes på stasjonære datamaskiner.

Maks poeng: 2

21 Branch Target Address

Gitt utsnittet av ARM assemblerkoden under, hvilken tallverdi må *imm24* feltet til *maskinkoden* til Branch instruksjonen (B) ha?: [ ] .

```
0x1000          MOV R0, #10
0x1004          MOV R1, #5
0x1008          ADD R2, R0, R1
0x100C          B label1
0x1010          SUB R3, R2, #1
0x1014  label1  MOV R4, #7
```

Maks poeng: 2

22 Tolking av maskinkode

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

$0x00804002$

[Velg alternativ] (ADDLT, ADDS, ~~ANDEQ~~ ADDEQ, ADD) [Velg alternativ] (R0, R6, R2, R4)

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

Maks poeng: 6

23 Oversette til assemblerkode

C-funksjonen nedenfor finner den største verdien av **a** og **b** og returnerer denne. Oversett C-funksjonen til ARM-assembler ved å fylle inn de manglende instruksjonene. Anta at **a** er lagt i **R0** og **b** i **R1** før funksjonen kalles.

```
int max(int a, int b) {
   if (a > b) {
      return a;
   } else {
      return b;
   }
}
```

MAX: Velg alternativ (adds R0, R1, BGE END, CMP R0, R1, BNE END, BX LR)

Velg alternativ (ADD R0, R0, R1, CMP R0, R1, BGT MAX, MOVLE R0, R0, MOVGT R0, R0)

Velg alternativ (LSL R0, R0, #2, MOVLE R0, R1, MOVGT R0, R1, ADD R0, R1, #1, BLE MAX)

END: Velg alternativ (BX PC, PUSH LR, MOV R0, PC, MOV PC, LR, POP LR)

Maks poeng: 6

24 Tolke assemblerkode

Under er et utsnitt av ARM-assemblerkode gjengitt. Anta at verdien av **n** er lagret i register **R0**. Hva gjør denne koden?

```
        MOV R1, #0
        MOV R2, #1
label:  ADD R1, R1, R2
        ADD R2, R2, #1
        CMP R2, R0
        BLE label
```

Velg ett alternativ:

- Den teller antall ganger verdien i R2 kan adderes til R1 før R1 blir lik eller større enn n.
- Den beregner differansen mellom n og summen av tallene fra 1 til n.
- Den beregner summen av tallene fra 1 til n.
- Den beregner summen av partallene fra 1 til n.
- Den dobler verdien av n.

Maks poeng: 3

25 Arm funksjonskallkonvensjonen

I ARM-arkitekturen har vi **leaf-funksjoner** og **non-leaf-funksjoner**. Disse har ulike krav i forhold til funksjonskallkonvensjonen.

Hvilket av følgende utsagn er korrekt?
Velg ett alternativ:

- Non-leaf-funksjoner lagrer alltid R0 på stacken, siden dette er registeret brukt for returadressen.
- En non-leaf-funksjon trenger aldri å lagre LR, siden ARM sørger for at returadressen alltid lagres automatisk på stacken.
- En leaf-funksjon trenger ikke å lagre LR på stacken, men må unngå å overskrive LR for å kunne returnere korrekt.
- En leaf-funksjon må lagre LR og alle andre registre på stacken for å sikre at returadressen og alle data bevares.
- Både leaf- og non-leaf-funksjoner må alltid lagre LR og R0 til R3 på stacken, uansett om de kaller andre funksjoner eller ikke.

Maks poeng: 2

26 Mikroarkitektur og ytelse

Under er det noen påstander om mikroprosessorarkitekturene *Single-Cycl*e, *Multi-Cycl*e og *Pipeline*. Hvilken av påstandene korrekt?

Velg ett alternativ:

- Single-Cycle arkitekturen har høyere gjennomstrømning enn Pipeline arkitekturen fordi alle instruksjoner fullføres i én syklus.
- I en Pipeline arkitektur økes ytelsen ved å redusere antall klokkesykluser per instruksjon (CPI) til én, uavhengig av hvor komplekse instruksjonene er.
- Pipeline arkitekturen forbedrer ytelsen ved å tillate overlappende utførelse av instruksjoner, noe som øker gjennomstrømningen sammenlignet med både Single-Cycle og Multi-Cycle arkitekturer.
- En Multi-Cycle arkitektur har alltid lavere CPI enn Pipeline arkitektur, fordi alle instruksjoner fullføres i en fast mengde sykluser.
- Multi-Cycle arkitekturen er tregere enn Single-Cycle arkitekturen fordi alle instruksjoner alltid må fullføres i det samme antall sykluser.

Maks poeng: 2

27 Mikroarkitektur og ytelse

Gitt følgende assemblerkode. Her har vi nummerert instruksjonene fra 1 til 6

#1 MOV R1, #5
#2 ADD R2, R1, R3
#3 SUB R1, R2, R4
#4 MUL R3, R1, R2
#5 ADD R4, R3, R1
#6 ADD R5, R4, R1

- Anta at vi skal kjøre disse instruksjonene med en 5 stegs *Pipeline* design med hazard enhet som beskrevet i boka. Hvor mange klokkesykluser vil pipeline-prosessoren bruke på å utføre assemblerkoden over? [ ] .
- Anta at du har en multicycle-prosessor som har en CPI på 4 for denne typen instruksjoner. Hvor mange klokkesykluser vil denne bruke på koden over? [ ] .
- Anta at pipeline-prosessoren kjører med en klokkehastighet på 2 GHz. Hvilken klokkehastighet (i GHz) må en single-cycle-prosessor ha for å bli ferdig samtidig som pipeline-prosessoren? [ ] Ghz.

Maks poeng: 3

28 Data-forwarding

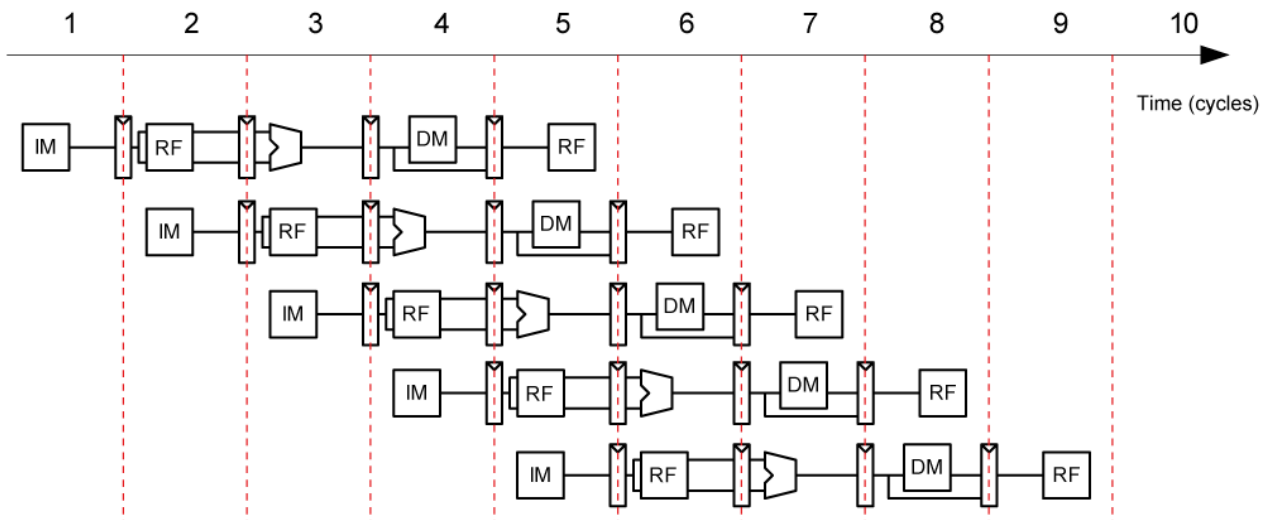
I en pipeline mikroarkitektur, hva er formålet med *data-forwarding*?

Velg ett alternativ:

- Å redusere antall instruksjoner som kjøres i pipeline-en samtidig for å unngå strukturelle hazards.
- Å redusere klokkesyklusene per instruksjon ved å utføre flere operasjoner samtidig.
- Å forhindre dataavhengigheter ved å sende instruksjoner tilbake til fetch-stadiet.
- Å gi pipeline-en muligheten til å hente nye instruksjoner raskere fra minnet ved å bruke cache.
- Å sende resultatene av en instruksjon direkte til en etterfølgende instruksjon uten å vente på at de blir skrevet til registerfilen.

Maks poeng: 2

29 Pipeline



Gitt følgende ARM-assemblerprogram:

LDR R4, [R1, #8]
ADD R1, R1, R2
STR R4, [R2, #12]
SUB R2, R2, R1
ADD R3, R3, R1

Hva slags pipelineforløp vil koden over gi? Anta en 5-stegs 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 4 Velg alternativ (leser fra registeret, skriver til registeret, ingen register aktivitet, leser fra og skriver til registeret)
- I sykel 5 Velg alternativ (leser fra registeret, skriver til registeret, leser fra og skriver til registeret, ingen register aktivitet)
- I sykel 6 Velg alternativ (ingen register aktivitet, leser fra og skriver til registeret, leser fra registeret, skriver til registeret)
- I sykel 7 Velg alternativ (skriver til registeret, leser fra og skriver til registeret, ingen register aktivitet, leser fra registeret)

Hvilken hazard har vi i følgende klokkesykler?

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

Maks poeng: 6

30 Cache-1

Vi sammenligner to prosessorsystemer med foreskjellig cache-oppsett. Utenom cachen, er prosessorene like og yter det samme.

System A har en gjennomsnittlig treff-rate på 85% når det leser fra cache, mens system B har et gjennomsnitt på 93% når det leser fra cachen.

Lesing fra cache tar én klokkesyklus, mens lesing fra hovedminnet tar 100 klokkesykler for begge systemene. Vi antar 100% treff ved lesing fra hovedminnet.

Inntil tre siffers presisjon kan være påkrevd for svarene i denne oppgaven.

a) Hva er gjennomsnittlig aksess-tid (i klokkesykler) for system A?

b) Hva er gjennomsnittlig aksess-tid (i klokkesykler) for system B?

For en gitt oppgave, bruker system A 90% av tiden på minneaksess- resten blir brukt til utregning.

c) Hvor stor andel tid vil system B bruke sammenlignet med system A for den samme oppgaven?

Oppgi svaret i prosent: %

Maks poeng: 6

31 Cache-2

Vi har en direkte-mappet cache med 2KB datakapasitet. Hvert ord har 4 byte og blokkstørrelsen er på 4 ord. Hver byte kan adresseres.

Merk: I a) og b) er svaret heltall, mens i c) og d) er inntill 3 siffers presisjon påkrevd.

a) Hvor mange sett er det i denne cachen?

Vi leser følgende sekvens med adresser én gang:
0x010, 0x014, 0x01C, 0x020, 0x820, 0x424, 0x42C, 0x82C

b) Hvor mange av disse åtte leseoperasjonene resulterer i en cache-miss?

Vi omstarter systemet (cachen tømmes) og så leser vi samme sekvens av adresser 10 ganger.

c) Hva blir treffraten (hit-rate) for denne operasjonen?

(*oppgi svar som andel*)

Vi bytter til et oppsett med en to-veis assosiativ cache med samme (2KB) datakapasitet, men med en blokkstørrelse på 2 ord.

d) Hva blir treffraten med den nye cachen dersom vi gjør den samme leseoperasjonen som i c)?

(*oppgi svar som andel*)

Maks poeng: 6

32 VM

Vi har et system med virtuelt minne som kan adressere 2^{32} bytes.
Systemet har ubegrenset harddiskstørrelse, men kun 256MB fysisk minne.
Vi antar både virtuelt og fysisk minne har 16KB sidestørrelse (16KB page size).

Alle svar i denne oppgaven er heltall.

Hvor mange bit er det i den fysiske adressen?

Hvor mange bit har det virtuelle sideantallet?

Hvor mange bit har det fysiske sideantallet?

Maks poeng: 6

Question 21
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 ^ Src2`
`BIC Rd, Rn, Src2`	Bitwise Clear	`Rd = Rn & ~Src2`
`MVN Rd, Rn, Src2`	Bitwise NOT (`~`)	`Rd = ~Rn`
`LSL Rd, Rn, Src2`	Logical Shift Left (`<<`)	`Rd = Rn << Src2`
`LSR Rd, Rn, Src2`	Logical Shift Right (`>>`)	`Rd = Rn >> 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 << 2`
`BL label`	Branch and Link	`LR = (PC + 8) - 4; PC = (PC + 8) + im24 << 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 *or* 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 22
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

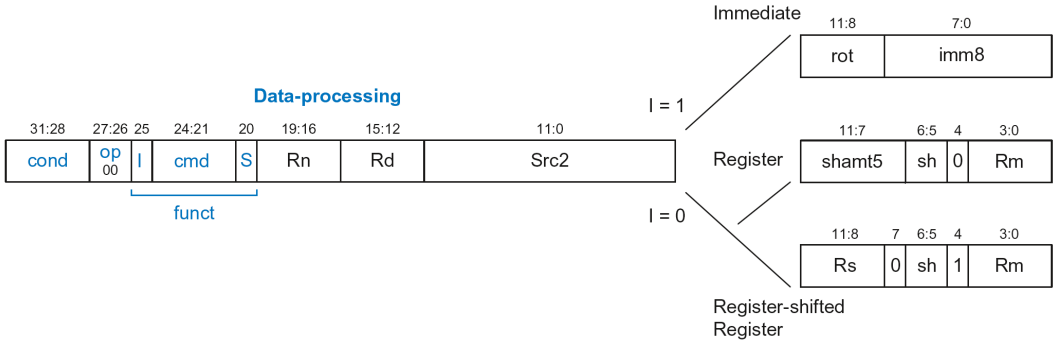


Figure 1: Data processing instruction format

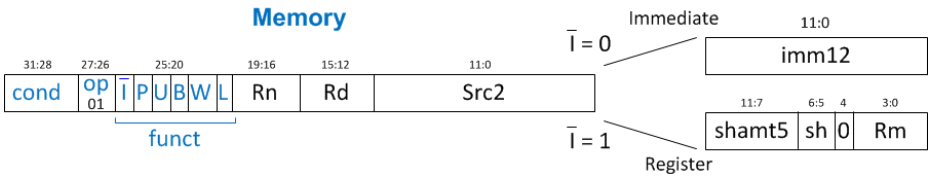


Figure 2: Memory processing instruction format

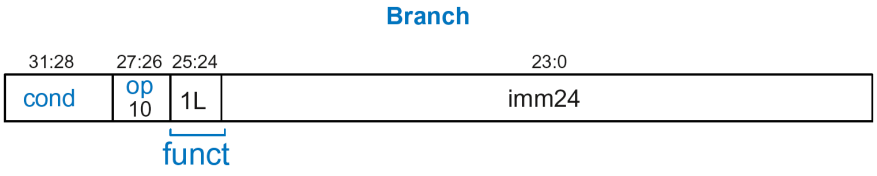


Figure 3: Branch instruction format

Table B.1 Data-processing instructions

cmd	Name	Description	Operation
0000	`AND Rd, Rn, Src2`	Bitwise AND	`Rd ← Rn & Src2`
0001	`EOR Rd, Rn, Src2`	Bitwise XOR	`Rd ← Rn ^ Src2`
0010	`SUB Rd, Rn, Src2`	Subtract	`Rd ← Rn - Src2`
0011	`RSB Rd, Rn, Src2`	Reverse Subtract	`Rd ← Src2 - Rn`
0100	`ADD Rd, Rn, Src2`	Add	`Rd ← Rn+Src2`
0101	`ADC Rd, Rn, Src2`	Add with Carry	`Rd ← Rn+Src2+C`
0110	`SBC Rd, Rn, Src2`	Subtract with Carry	`Rd ← Rn - Src2 -` $\overline{C}$
0111	`RSC Rd, Rn, Src2`	Reverse Sub w/ Carry	`Rd ← Src2 - Rn -` $\overline{C}$
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 ← Rn | Src2`
1101	Shifts:		
$I = 1$ OR ($\text{instr}_{11:4} = 0$)	`MOV Rd, Src2`	Move	`Rd ← Src2`
$I = 0$ AND ($sh = 00$; $\text{instr}_{11:4} \neq 0$)	`LSL Rd, Rm, Rs/shamt5`	Logical Shift Left	`Rd ← Rm << Src2`
$I = 0$ AND ($sh = 01$)	`LSR Rd, Rm, Rs/shamt5`	Logical Shift Right	`Rd ← Rm >> Src2`
$I = 0$ AND ($sh = 10$)	`ASR Rd, Rm, Rs/shamt5`	Arithmetic Shift Right	`Rd ← Rm>>>Src2`
$I = 0$ AND ($sh = 11$; $\text{instr}_{11:7,\ 4} = 0$)	`RRX Rd, Rm, Rs/shamt5`	Rotate Right Extend	`{Rd, C} ← {C, Rd}`
$I = 0$ AND ($sh = 11$; $\text{instr}_{11:7} \neq 0$)	`ROR Rd, Rm, Rs/shamt5`	Rotate Right	`Rd ← Rn ror Src2`
1110	`BIC Rd, Rn, Src2`	Bitwise Clear	`Rd ← Rn & ~Src2`
1111	`MVN Rd, Rn, Src2`	Bitwise NOT	`Rd ← ~Rn`

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

Question 23
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 ^ Src2`
`BIC Rd, Rn, Src2`	Bitwise Clear	`Rd = Rn & ~Src2`
`MVN Rd, Rn, Src2`	Bitwise NOT (`~`)	`Rd = ~Rn`
`LSL Rd, Rn, Src2`	Logical Shift Left (`<<`)	`Rd = Rn << Src2`
`LSR Rd, Rn, Src2`	Logical Shift Right (`>>`)	`Rd = Rn >> 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 << 2`
`BL label`	Branch and Link	`LR = (PC + 8) - 4; PC = (PC + 8) + im24 << 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 *or* 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 24
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 ^ Src2`
`BIC Rd, Rn, Src2`	Bitwise Clear	`Rd = Rn & ~Src2`
`MVN Rd, Rn, Src2`	Bitwise NOT (`~`)	`Rd = ~Rn`
`LSL Rd, Rn, Src2`	Logical Shift Left (`<<`)	`Rd = Rn << Src2`
`LSR Rd, Rn, Src2`	Logical Shift Right (`>>`)	`Rd = Rn >> 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 << 2`
`BL label`	Branch and Link	`LR = (PC + 8) - 4; PC = (PC + 8) + im24 << 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 *or* 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 • 1 = B	B + 0 = B	Identity
T2	B • 0 = 0	B + 1 = 1	Null Element
T3	B • B = B	B + B = B	Idempotency
T4	(B')' = B		Involution
T5	B • B' = 0	B + B' = 1	Complements

#	Theorem	Dual	Name
T6	B•C = C•B	B+C = C+B	Commutativity
T7	(B•C) • D = B • (C•D)	(B + C) + D = B + (C + D)	Associativity
T8	B • (C + D) = (B•C) + (B•D)	B + (C•D) = (B+C) (B+D)	Distributivity
T9	B • (B+C) = B	B + (B•C) = B	Covering
T10	(B•C) + (B•$\overline{C}$) = B	(B+C) • (B+$\overline{C}$) = B	Combining
T11	(B•C) + ($\overline{B}$•D) + (C•D) = (B•C) + ($\overline{B}$•D)	(B+C) • ($\overline{B}$+D) • (C+D) = (B+C) • ($\overline{B}$+D)	Consensus

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