Information

Oppgåve	Tittel	Poeng	Oppgåvetype
i	Informasjon		Informasjon eller ressurser

Digital representation

Oppgåve	Tittel	Poeng	Oppgåvetype
1	Digital representation 1	2	Fleirval
2	Digital representasjon 2	2	Fyll inn tal
3	2'er komplement	2	Fyll inn tal

Combinational logic

Oppgåve	Tittel	Poeng	Oppgåvetype
4	Boolske porter	2	Fleirval
5	Boolsk algebra	3	Fleirval

Sequential logic

Oppgåve	Tittel	Poeng	Oppgåvetype
6	Kombinatorisk og sekvensiell logikk	2	Fleirval (fleire svar)
7	Sekvensielle kretser	12	Nedtrekk

Digitale byggeblokker 1

Oppgåve	Tittel	Poeng	Oppgåvetype
8	LUT1-1	2	Plasser i tekst

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

Digitale byggeblokker 2

Oppgåve	Tittel	Poeng	Oppgåvetype
11	LUT2-1	2	Plasser i tekst
12	LUT2-2	1	Nedtrekk
13	LUT2-3	1	Nedtrekk

ALU

Oppgåve	Tittel	Poeng	Oppgåvetype
14	ALU	6	Fleirval

HDL

Oppgåve	Tittel	Poeng	Oppgåvetype
15	Kritisk sti multiplekser	1	Fyll inn tal
16	Kritisk sti multiplekser 2	3	Fyll inn tal
17	Kritisk sti fulladder	1	Fyll inn tal
18	Kritisk sti ALU-1	1	Fleirval
19	Kritisk sti, ALU-2	3	Fyll inn tal

Computer Architecture

Oppgåve	Tittel	Poeng	Oppgåvetype
20	ARM-maskinkode	2	Fleirval

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

Mikroarkitektur

Oppgåve	Tittel	Poeng	Oppgåvetype
26	Mikroarkitektur og ytelse	2	Fleirval
27	Mikroarkitektur og ytelse	3	Fyll inn tal
28	Data-forwarding	2	Fleirval
29	Pipeline	6	Nedtrekk

Minnesystemer

Oppgåve	Tittel	Poeng	Oppgåvetype
30	Cache-1	6	Fyll inn tal
31	Cache-2	6	Fyll inn tal
32	VM	6	Fyll inn tal

1 Digital representation 1

Digital representasjon

Gjer om desimaltalet $(25)_{10}$ til eit 8 bits binærtal.

Vel eitt alternativ

- 00111011
- 00011001 ✔
- Ingen av alternativa er korrekte.
- 00101001
- 00011010

Maks poeng: 2

2 Digital representasjon 2

Digital representasjon

Gjer om det heksadesimale talet $(D4)_{16}$ til desimaltal. [ ] (212) .

Maks poeng: 2

3 2'er komplement

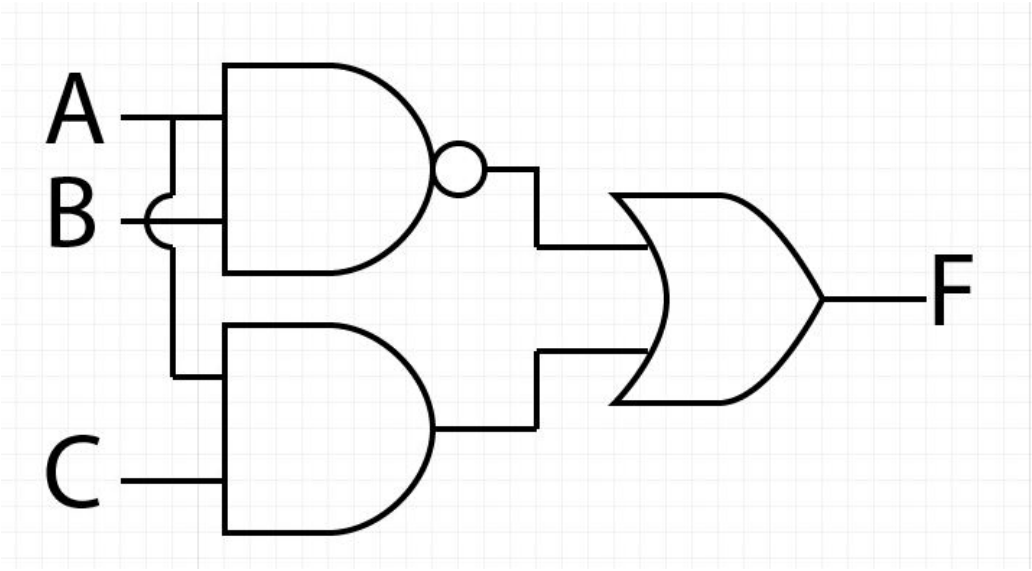
Digital representasjon

Gjer om **toerkomplement**-talet $(11101101)_2$ til desimaltal. : [ ] (-19) .

Maks poeng: 2

4 Boolske porter

Kva funksjonsuttrykk speglar portimplementasjonen under?



Vel eitt alternativ

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

Maks poeng: 2

5 Boolsk algebra

Forenkla følgjande uttrykk maksimalt.

F = (A+B′)·(A+B)
Vel eitt alternativ

- F = A ✔
- F = B′
- Ingen av alternativa er korrekte.
- F = A+B'
- F = B

Maks poeng: 3

6 Kombinatorisk og sekvensiell logikk

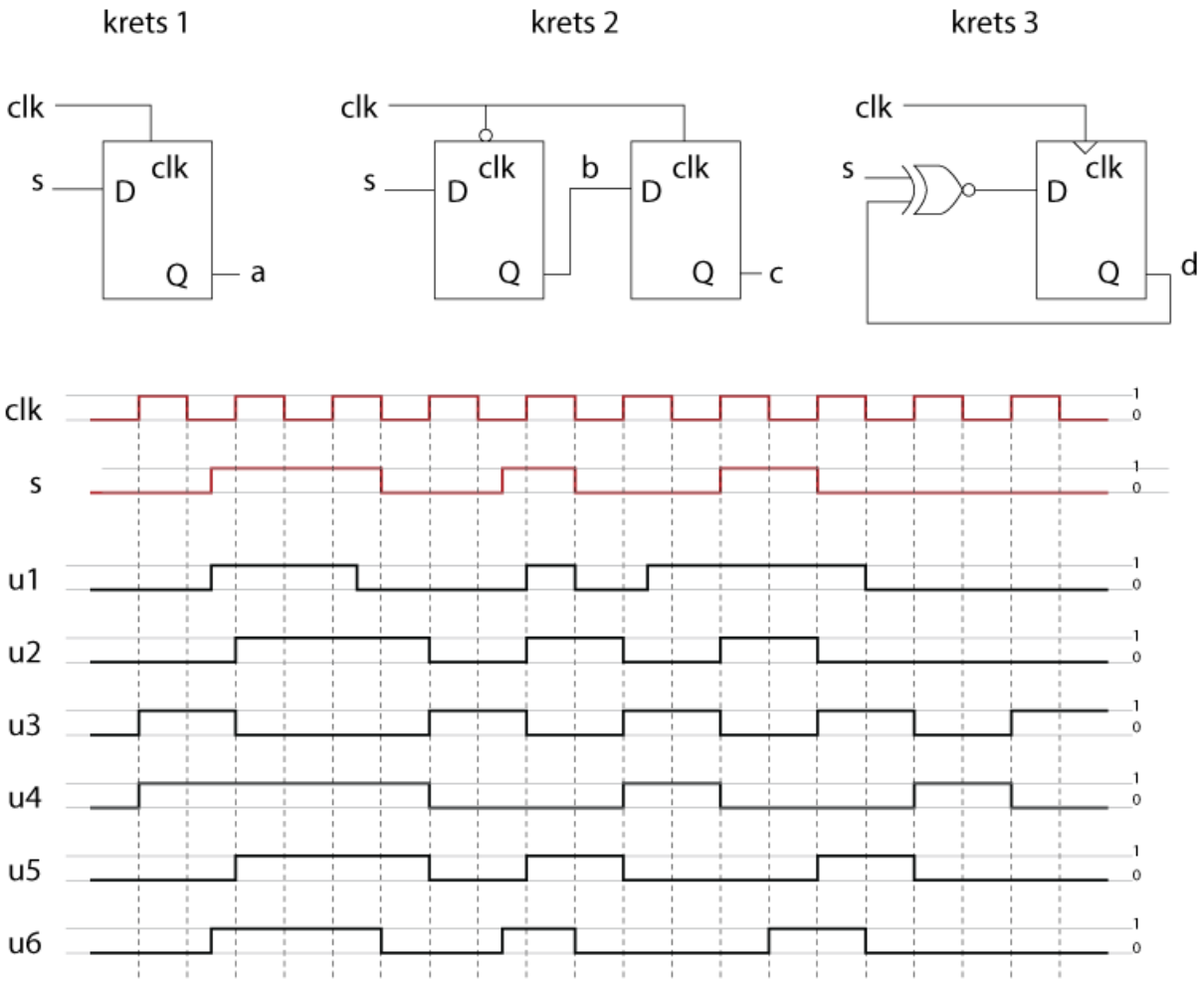
Kva to utsegner om kombinatorisk og sekvensiell logikk er riktige?
Vel dei to alternativa som er rette.

- [ ] Ein halv-adder og ein full-adder er begge døme på sekvensielle kretsar.
- [ ] Vi kan konstruere sekvensielle kretsar med logiske portar utan nokon form for minneelement.
- [ ] Utgangen til ein sekvensiell logisk krets kan endre seg basert på tidlegare inngangar ✔ klokkepulsen.
- [ ] Kombinatoriske kretsar må alltid inkludere tilbakekoplingar for å fungere korrekt.
- [ ] I kombinatorisk logikk er utgangen berre avhengig av dei noverande inngangane og ikkje av ✔ tidlegare inngangar.
- [ ] Flip-flop'ar og latch'ar er vanlege å bruke i kombinatoriske kretsar.

Maks poeng: 2

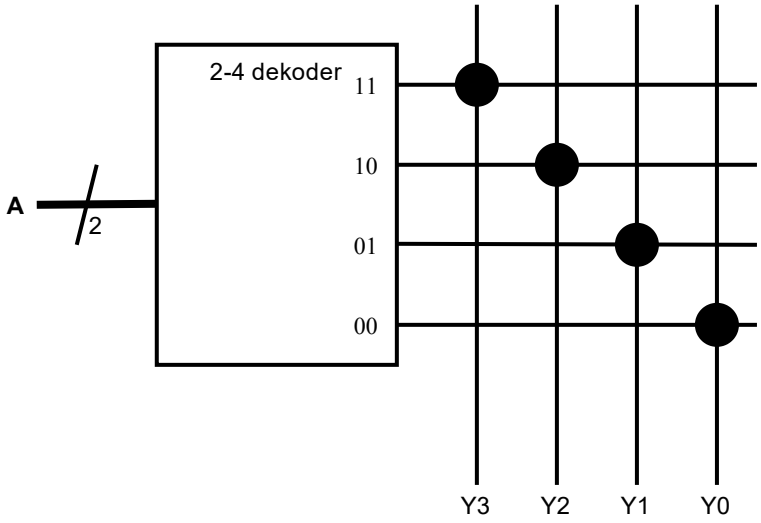
7 Sekvensielle kretser

Hver av dei tre kretsane under tek inn klokkesignalet **clk** og inngangssignalet **s**. Gå ut frå at utgangane **a**, **b**, **c** og **d** har startverdien **0**. Kva tidsforløp under (u1 til u6) høyrer til dei ulike utgangane? Merk at det er gitt to tidsforløp for mykje. Du skal ikkje ta omsyn til portforsinking. **Studer kretsane nøye og merk forskjellen mellom latches og flip-flops i illustrasjonen.**



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

Maks poeng: 12



8 LUT1-1

Kva blir formelen for kvar av utgangane til oppslagstabellen?

Alle felt skal fyllast. Der to felt følgjer etter kvarandre skal eventuelle blanke brukast lengst til høgre.

Eks: Y_{eks} = **NOT** A_1 **XOR** A_0 :

Y_{eks} = NOT A_1 XOR [ ] A_0

Sett inn riktig alternativ.

Hjelp

AND | OR | XOR | NOT | [ ]

Y_3 = [ ] A_1 [AN] A_0

Y_2 = [ ] A_1 [AN] [NC] A_0

Y_1 = [NC] A_1 [AN] [ ] A_0

Y_0 = [NC] A_1 [AN] [NC] A_0

Maks poeng: 2

9 LUT1-2

Kva blir utgangen med følgjande inngangar..?

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

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

Maks poeng: 1

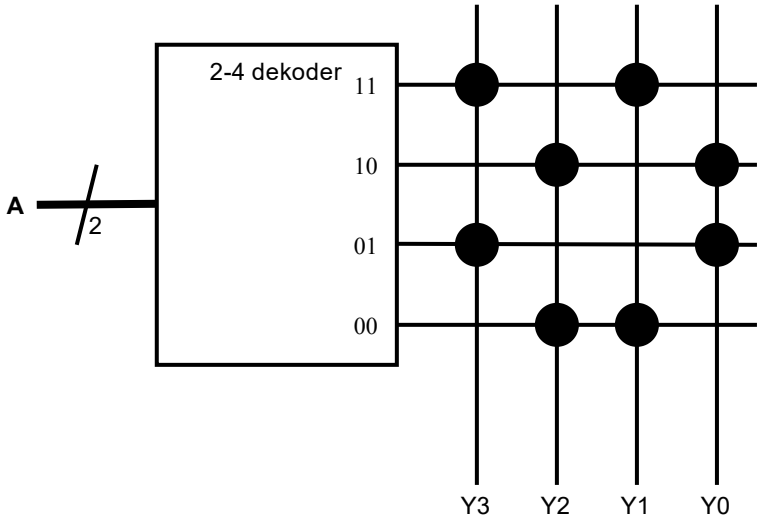
10 LUT1-3

Kva må inngangen vere, dersom utgangen er som følgjer?

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

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

Maks poeng: 1



11 LUT2-1

Kva blir formelen for kvar av utgangane til oppslagstabellen?

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

Hjelp

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

Y_3 = A0 ✔

Y_2 = NOT ✔ A0 ✔

$Y_1 = A_1$ XNC ✔ A_0

$Y_0 = A_1$ XOR ✔ A_0

Maks poeng: 2

12 LUT2-2

Kva blir utgangen med følgjande inngangar..?

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

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

Maks poeng: 1

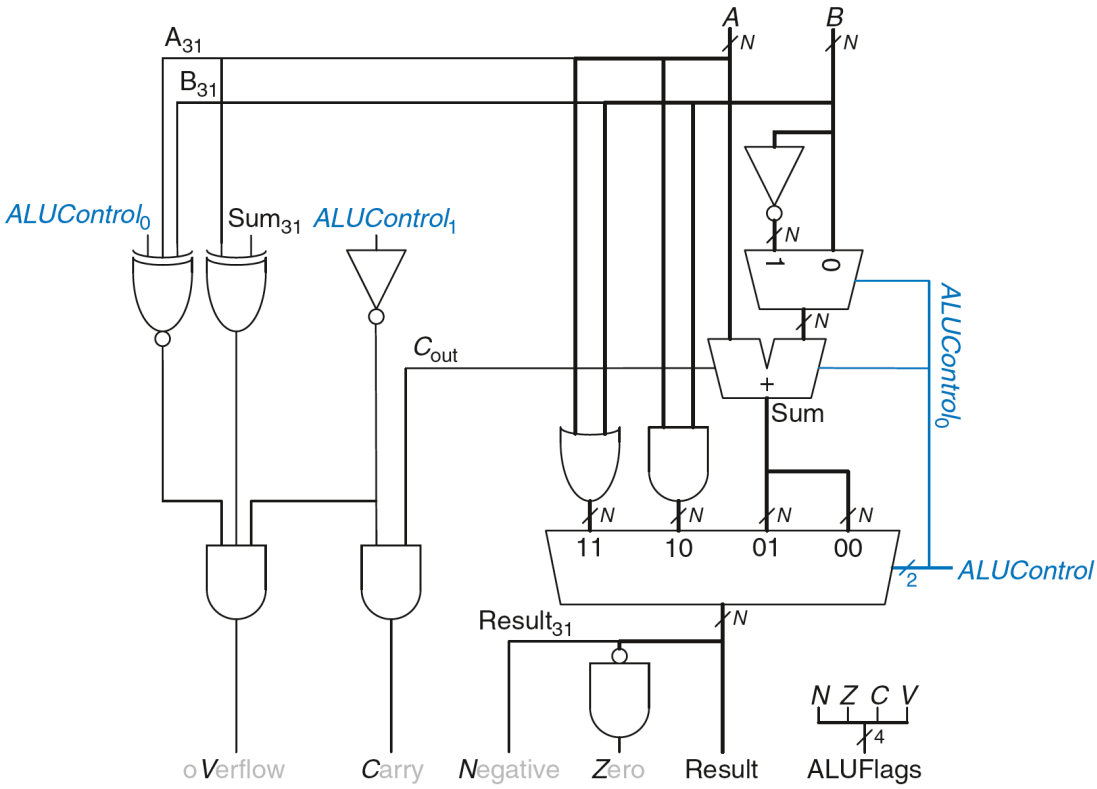
13 LUT2-3

Kva må inngangen vere, dersom utgangen er som følgjer?

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

Y = 0x6 gir **A** = Vel alternativ (ingen løysing, 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
Cout
1
0
+
Sum
ALUControl0
11
10
01
00
2
ALUControl
Result31
N Z C V
4
oVerflow
Carry
Negative
Zero
Result
ALUFlags

14 ALU

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

Kva blir Resultatet («Result) når vi set ALUcontrol(1 downto 0) til...

a) "10"

Vel eitt alternativ

0x0000	0x04AB	0x2004	0x4FF2	0xFFFF
✔				

b) "11
Vel eitt alternativ

0x4FF2	0x04AB	0x2004	0x44AE	0xFFFF
✔				

c) "00"
Vel eitt alternativ

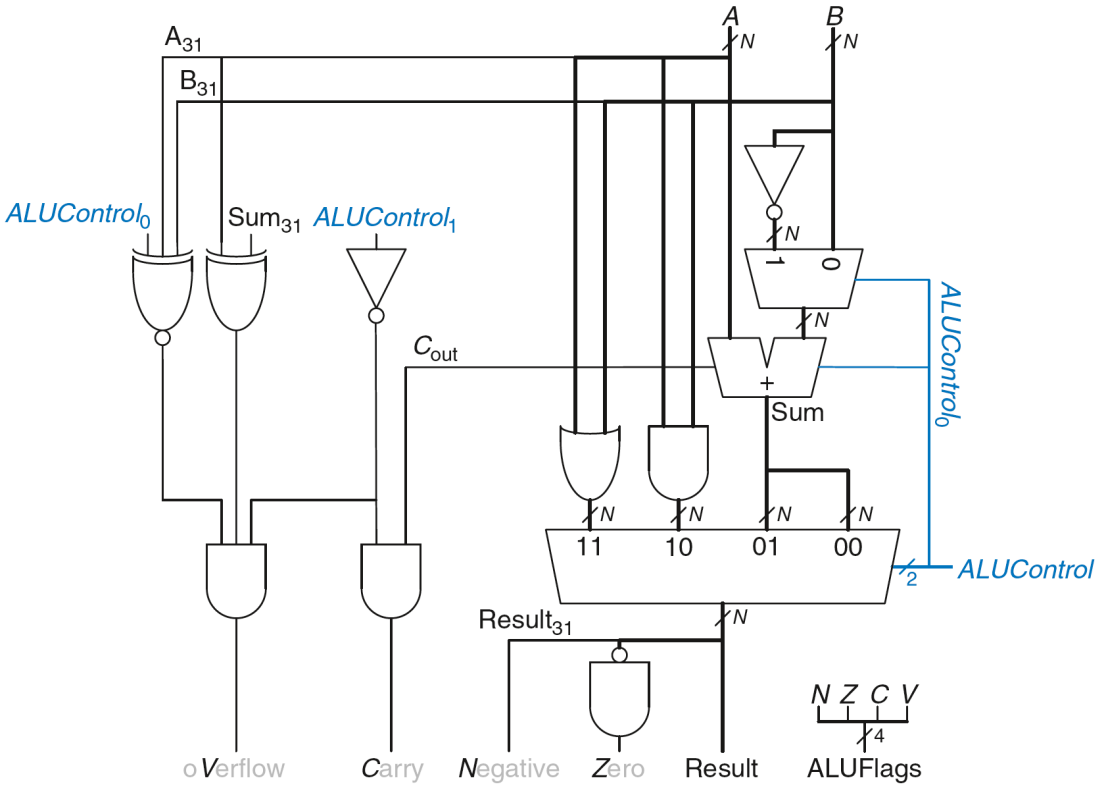
0x4FF2	0x0000	0xFFFF	0x2004	0x44AE
✔				

Vi brukar same data, A = 0x4A50 og B = 0x05A2, og set ALUkontroll til "01"

d) Kva flagg blir sett
Vel eitt alternativ

V	alle fire	ingen	Z	N	C
					✔

Maks poeng: 6



Figuren viser ei ALU med kontrollsignal. (Brukast i to av oppgåvene om kritisk sti)

For oppgåvene om kritisk sti gjeld følgjande:
Propageringsforsinkinga til kvar inverter er på 10ps.
Propageringsforsinkinga til kvar to-input port er på 50 ps.
All logikk blir bygd med to-input portar.
For eksempel vil tre-inputs AND-porten som blir brukt til å berekne overflow lagast med to to-input AND-portar.
Rekkjefølga fleireinput-portar blir sett opp i skal gi kortast mogleg kritisk sti.
Mentepropageringa til addisjonskretsen er av typen "ripple-carry".

15 Kritisk sti multiplekser

Ein multiplekser som vel mellom A og B er bygd opp med følgjande VHDL-kode:

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

Kva blir propageringsforsinkinga til denne multiplekseren?

[ ] (110) ps.

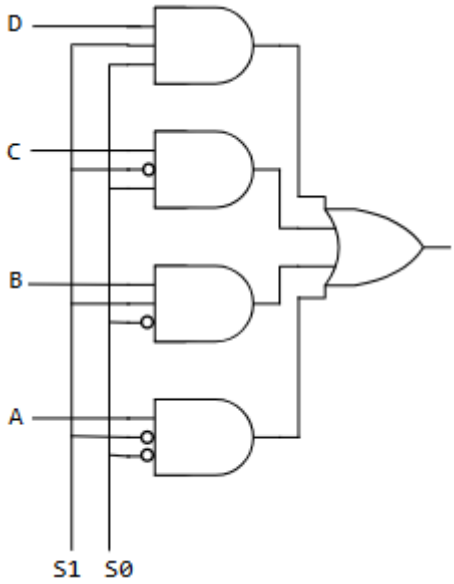
Maks poeng: 1

16 Kritisk sti multiplekser 2

Ein 4-input multiplekser som vel mellom ein av inngangane (A-D) kan skrivast med følgjande VHDL-kode:

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

Med fleireinput-portar kan den implementerast slik:



Dersom vi skal bruke to-input portar, så blir det meir samansett.

Kor mange AND-portar får vi dersom vi berre brukar to-input portar?

[ ] (8)

Kor mange OR-portar blir det dersom vi berre brukar to-input portar?

[ ] (3)

Kva blir portforsinkinga til kritisk sti gjennom denne multiplekseren?
(Hugs å ta med inverterar i berekninga)

[ ] (210) ps.

Maks poeng: 3

17 Kritisk sti fulladder

Kvar fulladder i kretsen er bygd opp med følgjande VHDL-kode:

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

Kva blir den maksimale propageringsforsinkinga til éin fulladder?

[ ] (150) ps.

Maks poeng: 1

18 Kritisk sti ALU-1

Kvar startar og stoppar den kritiske stien i ALUen?
Vel eitt alternativ

- ○ Frå ALUcontrol til Sum31
- ○ Frå B til Z ✔
- ○ Frå ALUcontrol til C
- ○ Frå A eller B til N
- ○ Frå A eller B til Result

Maks poeng: 1

19 Kritisk sti, ALU-2

Kva blir maksimal portforsinking gjennom addisjonskretsen frå A til Sum?

[ ] (3200) ps.

Tips: *Hugs at "Sum" er utgangen til addisjonskretsen, ikkje heile ALUen*

Kva blir portforsinkinga til kritisk sti gjennom heile ALUen?

[ ] 3720-3770 (~~3770~~) ps.

Tips: *Hugs at kritisk sti for ALUen ikkje nødvendigvis tek same veg som for kvar enkelt krets i dei øvrige oppgåvene.*

Maks poeng: 3

20 ARM-maskinkode

Kva kjenneteiknar ARM-maskinkode samanlikna med andre maskinkoder, som x86?

Vel eitt alternativ

- ARM-maskinkode kan køyrast på alle typar prosessorar utan tilpassingar, uavhengig av produsent.
- ARM-maskinkode er basert på ein RISC (Reduced Instruction Set Computer) arkitektur, som bruker færre og enklare instruksjonar enn x86. ✔
- ARM-maskinkode krev betydelig meir straum enn x86-maskinkode, og er derfor upraktisk for mobile einingar.
- ARM-maskinkode kan berre brukast på mobile einingar, medan x86 blir brukt på stasjonære datamaskiner.
- ARM-maskinkode bruker utelukkande 64-bit instruksjonar, medan x86 bruker 32-bit instruksjonar.

Maks poeng: 2

21 Branch Target Address

Gitt utsnittet av ARM assemblerkoden under, kva tallverdi må *imm24*-feltet til maskinkoden til Branch-instruksjonen (B) ha?: [ ] (0) .

```
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 følgjande ARM instruksjon (maskinkode) slik det er beskriven i læreboka, og vel dei alternativa som dannar tilsvarande assembler-instruksjon.

$0x00804002$

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

Vel alternativ (R0, R1, R2, R4) Vel alternativ (#1, R2, #2, R1)

Maks poeng: 6

23 Oversette til assemblerkode

C-funksjonen nedanfor finn den største verdien av **a** og **b** og returnerer denne. Omset C-funksjonen til ARM-assembler ved å fylle ut dei manglande instruksjonane. Anta at **a** er lagt i **R0** og **b** i **R1** før funksjonen vert kalla.

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

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

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

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

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

Maks poeng: 6

24 Tolke assemblerkode

Under er eit utdrag av ARM-assemblerkode gitt. Anta at verdien av **n** er lagra i register **R0**. Kva gjer denne koden?

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

Vel eitt alternativ

- Den reknar ut differansen mellom n og summen av tala frå 1 til n.
- Den reknar ut summen av partala frå 1 til n.
- Den doblar verdien av n.
- Den tel kor mange gongar verdien i R2 kan leggjast til R1 før R1 blir lik eller større enn n.
- Den reknar ut summen av tala frå 1 til n. ✔

Maks poeng: 3

25 Arm funksjonskallkonvensjonen

I ARM-arkitekturen har vi **leaf-funksjonar** og **non-leaf-funksjonar**. Desse har ulike krav i høve til funksjonskallkonvensjonen.

Kva av følgjande utsagn er korrekt?
Vel eitt alternativ

- Ein non-leaf-funksjon treng aldri å lagre LR, sidan ARM sørgjer for at returadressa alltid vert lagra automatisk på stacken.
- Non-leaf-funksjonar lagrar alltid berre R0 på stacken, sidan dette er registeret brukt for returadressa.
- Ein leaf-funksjon må lagre LR og alle andre register på stacken for å sikre at returadressa og alle data vert bevara.
- Ein leaf-funksjon treng ikkje å lagre LR på stacken, men må unngå å overskrive LR for å kunne returnere korrekt. ✔
- Både leaf- og non-leaf-funksjonar må alltid lagre LR og R0 til R3 på stacken, uavhengig av om dei kallar andre funksjonar eller ikkje.

Maks poeng: 2

26 Mikroarkitektur og ytelse

Nedanfor er det nokre påstandar om mikroprosessorarkitekturane Single-Cycle, Multi-Cycle og Pipeline. Kva for ein av påstandane er korrekt

Vel eitt alternativ

- Single-Cycle-arkitekturen har høgare gjennomstrøyming enn Pipeline-arkitekturen fordi alle instruksjonar blir fullførte i éin syklus.
- Multi-Cycle-arkitekturen er treigare enn Single-Cycle-arkitekturen fordi alle instruksjonar alltid må fullførast i det same talet syklusar.
- Ein Multi-Cycle-arkitektur har alltid lågare CPI enn Pipeline-arkitekturen, fordi alle instruksjonar blir fullførte i eit fast tal syklusar.
- Pipeline-arkitekturen forbetrar ytinga ved å tillate overlappande utføring av instruksjonar, noko som aukar gjennomstrøyminga samanlikna med både Single-Cycle- og Multi ✔ cle-arkitekturane.
- I ein Pipeline-arkitektur blir ytinga auka ved å redusere talet på klokkesyklusar per instruksjon (CPI) til éin, uavhengig av kor komplekse instruksjonane er.

Maks poeng: 2

27 Mikroarkitektur og ytelse

Gitt følgjande assemblerkode. Her har vi nummerert instruksjonane frå 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 køyre desse instruksjonane med ein 5-stegs pipeline-design med hazard-eining som skildra i boka. Kor mange klokkesyklusar vil pipeline-prosessoren bruke på å utføre assemblerkoden ovanfor? [ ] (10)
- Anta at du har ein multicycle-prosessor som har ein CPI på 4 for denne typen instruksjonar. Kor mange klokkesyklusar vil denne bruke på koden ovanfor? [ ] (24)
- Anta at pipeline-prosessoren køyrer med ein klokkehastigheit på 2 GHz. Kva klokkehastigheit (i GHz) må ein single-cycle-prosessor ha for å bli ferdig samtidig som pipeline-prosessoren? [ ] (1.2) GHz.

Maks poeng: 3

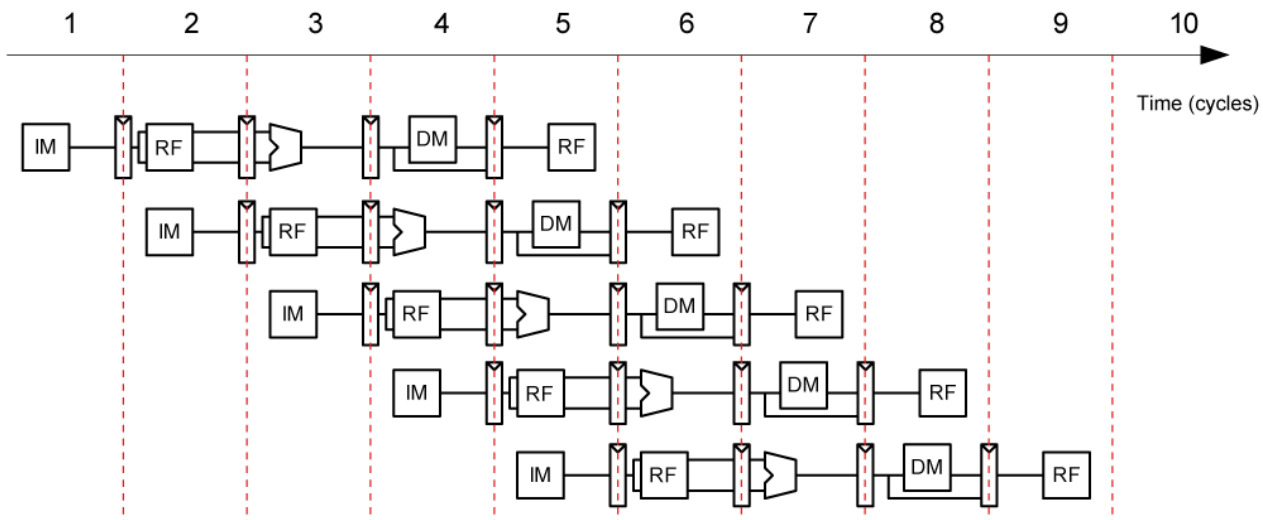
28 Data-forwarding

I ein *pipeline* mikroarkitektur, kva er formålet med *data-forwarding*?
Vel eitt alternativ

- Å gi pipeline-en moglegheit til å hente nye instruksjonar raskare frå minnet ved å bruke cache.
- Å hindre dataavhengnadar ved å sende instruksjonar tilbake til fetch-stadiet.
- Å redusere klokkesyklane per instruksjon ved å utføre fleire operasjonar samtidig.
- Å redusere talet på instruksjonar som køyrer i pipeline-en samtidig for å unngå strukturelle hazards.
- Å sende resultata frå ein instruksjon direkte til ein etterfølgjande instruksjon utan å vente på at dei blir skrivne til registerfila. ✔

Maks poeng: 2

29 Pipeline



Gitt følgjande ARM-assemblerprogram:

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

Kva slags pipelineforløp vil koden over gi? Gå ut frå ein 5-stegs pipeline-prosessor som illustrert over, men utan nokon form for hazard-håndtering. Her skriv vi til registerfila i første halvdel av klokkeperioden, og les av i andre halvdel av klokkeperioden. Velg dei alternativa under som er korrekte for pipelineforløpet.

Kva slags register aktivitet har me i følgjande klokkesyklar?

- I sykel 4 Vel alternativ (les frå og skriv til registeret, les frå registeret, ingen register aktivitet, skriv til registeret)
- I sykel 5 Vel alternativ (les frå registeret, skriv til registeret, les frå og skriv til registeret, ingen register aktivitet)
- I sykel 6 Vel alternativ (les frå og skriv til registeret, ingen register aktivitet, skriv til registeret, les frå registeret)
- I sykel 7 Vel alternativ (les frå og skriv til registeret, ingen register aktivitet, skriv til registeret, les frå registeret)

Kva hazard har me i følgjande klokkesyklar?

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

Maks poeng: 6

30 Cache-1

Vi samanliknar to prosessorsystem med forskjellig cache-oppsett. Utanom cachen, er prosessorane like og yter det same.

System A har ein gjennomsnittleg treff-rate på 85% når det les frå cache, medan system B har eit gjennomsnitt på 93% når det les frå cachen.

Lesing frå cache tar éin klokkesyklus, medan lesing frå hovudminnet tar 100 klokkesyklar for begge systema. Vi antar 100% treff ved lesing frå hovudminnet.

Inntil tre sifers presisjon kan vere påkravd for svara i denne oppgåva.

a) Kva er gjennomsnittleg aksess-tid (i klokkesyklusar) for system A? (15,8 - 16)

b) Kva er gjennomsnittleg aksess-tid (i klokkesyklusar) for system B? (7,9 - 8)

For ei gitt oppgåve, brukar system A 90% av tida på minneaksess - resten blir brukt til utrekning.

c) Kor stor del tid vil system B bruke samanlikna med system A for den same oppgåva?

Oppgi svaret i prosent: (53 - 56) %

Maks poeng: 6

31 Cache-2

Vi har ein direkte-mappa cache med 2KB datakapasitet. Kvart ord har 4 byte og blokkstorleiken er på 4 ord. Kvar byte kan adresserast.

Merk: I a) og b) er svaret heiltal, medan i c) og d) er inntil 3 sifers presisjon påkravd.

a) Kor mange sett er det i denne cachen?

[ ] (128)

Vi les følgjande sekvens med adresser éin gong:
0x010, 0x014, 0x01C, 0x020, 0x820, 0x424, 0x42C, 0x82C

b) Kor mange av desse åtte leseoperasjonane resulterer i ein cache-miss?

[ ] (4)

Vi omstartar systemet (cachen vert tømd) og så les vi same sekvens av adresser 10 gonger.

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

[ ] (0.720 - 0,730) (oppgje svar som andel)

Vi byter til eit oppsett med ein to-veg assosiativ cache med same (2KB) datakapasitet, men med ein blokkstorleik på 2 ord.

d)Kva blir treffraten med den nye cachen dersom vi gjer den same leseoperasjonen som i c)?

[ ] (0,57 - 0,58) (oppgje svar som andel)

Maks poeng: 6

32 VM

Vi har eit system med virtuelt minne som kan adressere 2^{32} bytes.
Systemet har uavgrensa harddiskstorleik, men berre 256MB fysisk minne.
Vi antar både virtuelt og fysisk minne har 16KB sidestorleik (16KB page size).

Alle svar i denne oppgåva er heiltal.

Kor mange bit er det i den fysiske adressa? [ ] (28)

Kor mange bit har det virtuelle sideantallet? [ ] (18)

Kor mange bit har det fysiske sideantallet? [ ] (14)

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

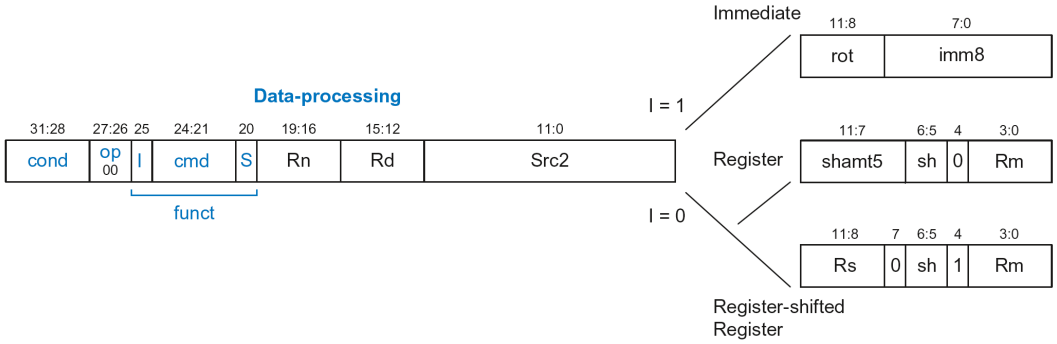


Figure 1: Data processing instruction format

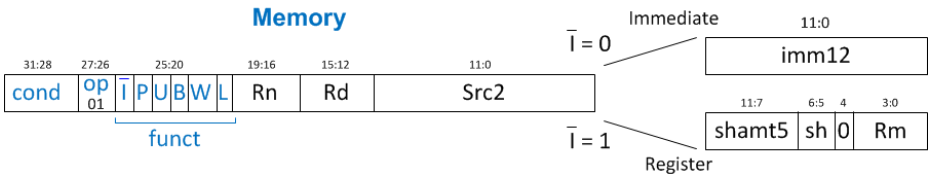


Figure 2: Memory processing instruction format

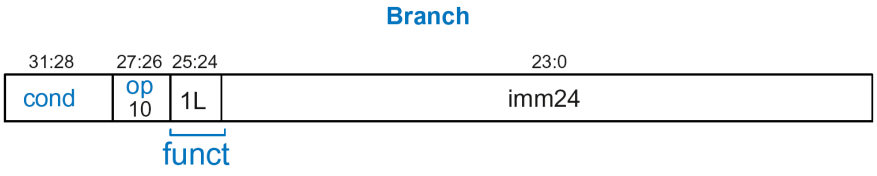


Figure 3: Branch instruction format

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 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