### Information

| Oppgave | Tittel | Maks poeng | Oppgavetype |
|---|---|---|---|
| i | Informasjon | | Informasjon eller ressurser |

### Digital representation

| Oppgave | Tittel | Maks poeng | Oppgavetype |
|---|---|---|---|
| 1 | Digital representation 1 | 2 | Flervalg |
| 2 | Digital representasjon 2 | 2 | Flervalg |
| 3 | Bitoppløsning | 2 | Fyll inn tall |

### Combinational logic

| Oppgave | Tittel | Maks poeng | Oppgavetype |
|---|---|---|---|
| 4 | Boolske porter | 2 | Flervalg |
| 5 | Boolsk algebra | 3 | Flervalg |

### Sequential logic

| Oppgave | Tittel | Maks poeng | Oppgavetype |
|---|---|---|---|
| 6 | Kombinatorisk og sekvensiell logikk | 2 | Flervalg (flere svar) |
| 7 | Sekvensielle kretser | 12 | Nedtrekk |

### HDL

| Oppgave | Tittel | Maks poeng | Oppgavetype |
|---|---|---|---|
| 8 | Hardwarespråk | 10 | Flervalg (flere svar) |

**Digital building blocks**

| Oppgave | Tittel | Maks poeng | Oppgavetype |
|---|---|---|---|
| 9 | Oppslagstabell | 5 | Plasser i tekst |
| 10 | Dekoder | 2 | Flervalg (flere svar) |

**Computer Architecture**

| Oppgave | Tittel | Maks poeng | Oppgavetype |
|---|---|---|---|
| 11 | Maskinkode kompatibilitet | 2 | Flervalg |
| 12 | Funksjonskall ARM assembler | 6 | Nedtrekk |
| 13 | Branch Target Adress | 2 | Flervalg |
| 14 | Oversette til assembler | 6 | Nedtrekk |
| 15 | Maskinkode | 6 | Nedtrekk |

**Mikroarkitektur**

| Oppgave | Tittel | Maks poeng | Oppgavetype |
|---|---|---|---|
| 16 | Mikroarkitektur og ytelse | 2 | Flervalg |
| 17 | Kjøretid til forskjellige mikroarkitekturer | 4 | Fyll inn tall |
| 18 | Pipeline | 8 | Nedtrekk |
| 19 | Control Hazards | 2 | Flervalg |

**Minnesystemer**

| Oppgave | Tittel | Maks poeng | Oppgavetype |
|---|---|---|---|
| 20 | Aksesstid | 4 | Fyll inn tall |

| | | | |
|---|---|---|---|
| 21 | Cache | 6 | Fyll inn tall |
| 22 | Minneoppslag | 10 | Fyll inn tall |

# 1 Digital representation 1

**Digital representasjon**

Gjør om desimaltallet $(42)_{10}$ til et 8 bits binærtall.

**Velg ett alternativ**

- 00100010
- 00101010 ✔
- Ingen av alternativene er korrekte.
- 00110011
- 00110110

Maks poeng: 2

# 2 Digital representasjon 2

**Digital representasjon**

Gjør om det heksadesimale tallet $(4E)_{16}$ til et 8 bits binærtall.

**Velg ett alternativ**

- 01101111
- 01101010
- Ingen av alternativene er korrekte.
- 01001110 ✔
- 01001011

Maks poeng: 2

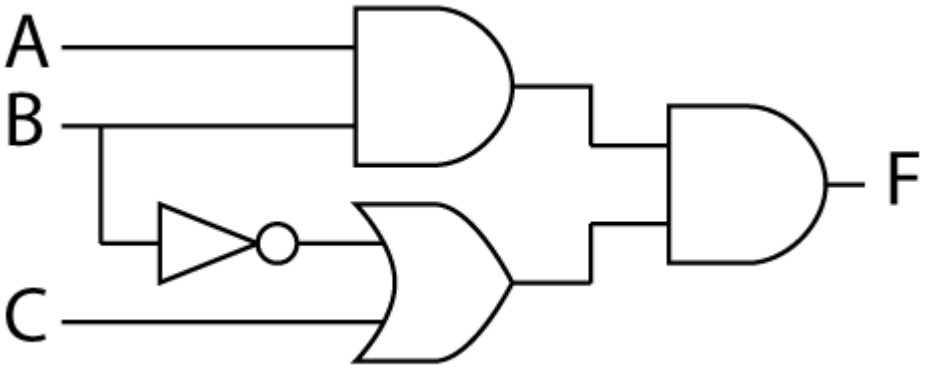
## 3 Bitoppløsning

Hvor mange forskjellige verdier kan man utrykke med et binært 12-bits tall? : (4096) .

Maks poeng: 2

# 4 Boolske porter

Hvilket funksjonsuttrykk gjenspeiler portimplementasjonen under?




**Velg ett alternativ**

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

Maks poeng: 2

# 5 Boolsk algebra

Forenkle følgende utrykk maksimalt

F = AB+A’C+BC

**Velg ett alternativ**

- F = AC+B
- F = AB + C
- F = A + BC
- F = B
- F = AB+A'C ✔

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.**

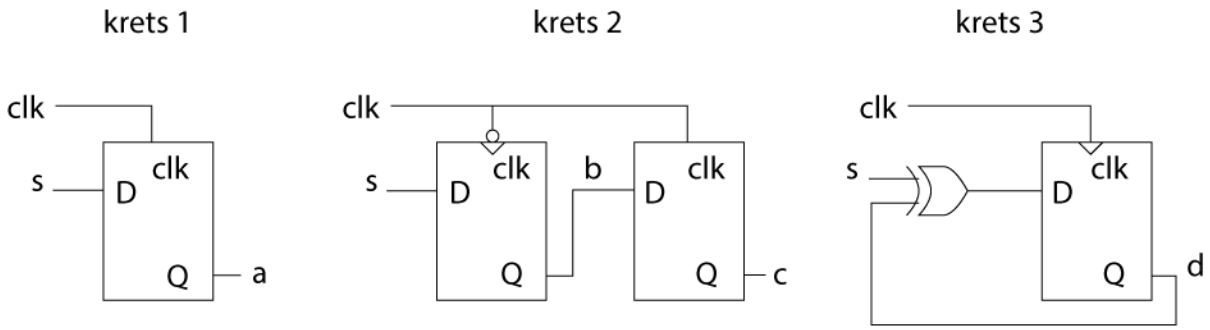
- [ ] Kombinatorisk logikk kan inneholde tilbakekoblinger
- [ ] Kombinatorisk logikk er basert på minneceller.
- [ ] Utgangen til synkrone sekvensielle systemer kan kun forandre verdi når klokken er høy.
- [ ] En tilstandsmaskin er basert på asynkron logikk.
- [ ] Man kan summere sammen tall med kombinatorisk logikk alene. ✔
- [ ] Funksjonen til synkrone sekvensielle systemer kan beskrives med tabeller. ✔

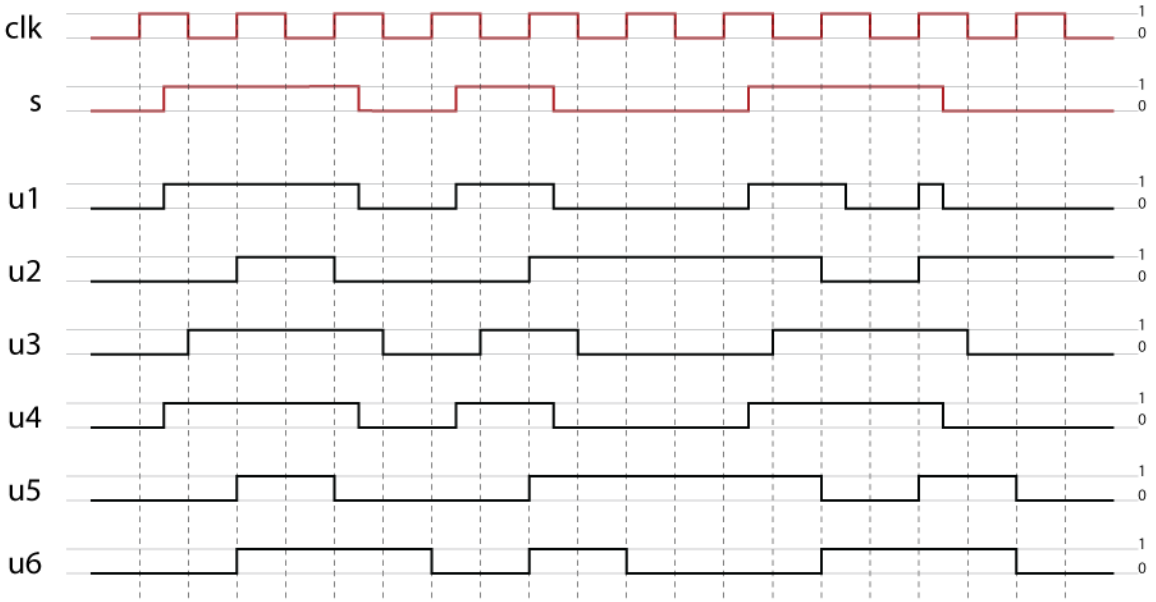
---

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 Hardwarespråk

Hvilke utsagn er sanne?

**Velg ett alternativ**

- [ ] VHDL og ARM-Assembler er eksempler på hardwarespråk
- [ ] Et hardwarespråk definerer den logiske funksjonen til en elektrisk krets ✔
- [ ] Syntese er når et hardwarespråk oversettes til maskinkode
- [ ] En implementasjon i VHDL lagres i minnet til prosessoren
- [ ] Et hardwarespråk definerer maskinkode direkte

**Velg ett alternativ**

- [ ] En port i VHDL kan være av typen in, out eller inout ✔
- [ ] En utgang i VHDL må ikke settes i tristate (høy impedanse)
- [ ] En testebenk for digitale design er en benk der vi spenner fast kretsen når vi tester den
- [ ] Strukturell kode i VHDL kan bare brukes i simulering
- [ ] En testbenk brukes til psykisk testing

**Velg ett alternativ**

- [ ] En prosess i VHDL kan bare gi verdi til ett signal
- [ ] Ingen av de andre alternativene er riktige
- [ ] Prosesser i VHDL kan bare brukes i simulering
- [ ] En prosess i VHDL kan brukes til å gi verdi til mange signaler ✔
- [ ] Prosesser i VHDL kan ikke brukes i simulering

**Velg ett alternativ**

- [ ] Simuleringskode kan ikke inneholde dataflytsbeskrivelser
- [ ] Registeroverføringsnivået (RTL) er det laveste nivået vi kan beskrive hardware på
- [ ] Strukturell kode beskriver hvordan moduler er koblet sammen ✔
- [ ] Dataflytkode beskriver registertilordninger
- [ ] En modul i VHDL må ha stor forbokstav i navnet

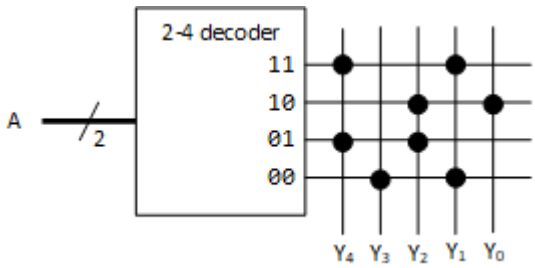
**Velg ett alternativ**

- [ ] Hvis ett signal av typen std_logic drives både '0' og '0' samtidig, så vil resultatet i simulering bli 'X'
- [ ] Hvis ett signal av typen std_logic drives både 'H' og '0' samtidig, så vil resultatet i simulering bli '0' ✔
- [ ] Hvis ett signal av typen std_logic drives både 'L' og '1' samtidig, så vil resultatet i simulering bli '0'
- [ ] Hvis ett signal av typen std_logic drives både '1' og '0' samtidig, så vil resultatet i simulering bli '0'
- [ ] Hvis ett signal av typen std_logic drives både 'H' og '1' samtidig, så vil resultatet i simulering bli '0'

---

Maks poeng: 10

# 9 Oppslagstabell




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

**Plasser de riktige alternativene i sine respektive bokser**

Hjelp

| | | | |
|---|---|---|---|
| A0 nand A1 | A0 xor A1 | A0 | A0 and A1 |
| not A1 | not A0 and A1 | not A0 | A0 xnor A1 |
| not A0 and not A1 | A0 not A1 | | |

Y4= A0 ✔

Y3= not A0 and not . ✔

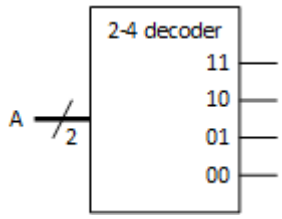
Y2= A0 xor A1 ✔

Y1= A0 xnor A1 ✔

Y0= not A0 and A1 ✔

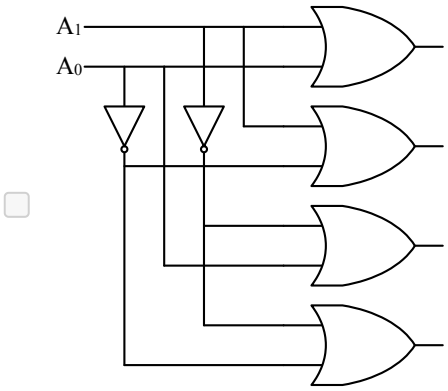
Maks poeng: 5

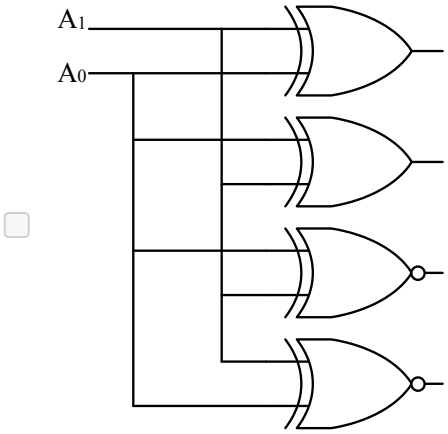
# 10 Dekoder

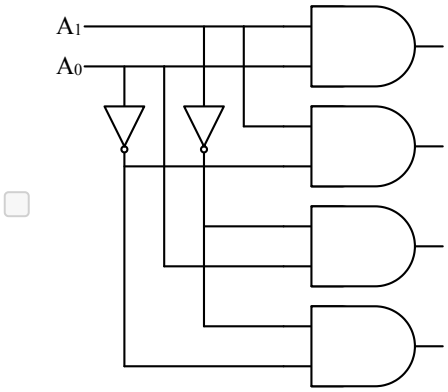


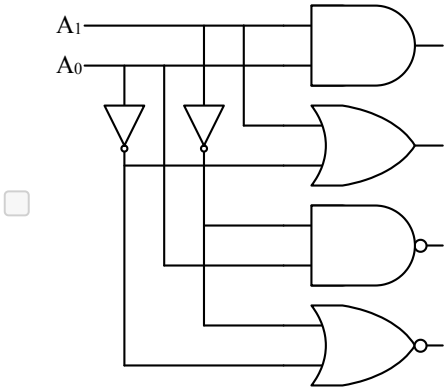

På figuren over har vi en generisk 2-4 dekoder. Hvilket av skjemaene under beskriver en 2-4 dekoder?

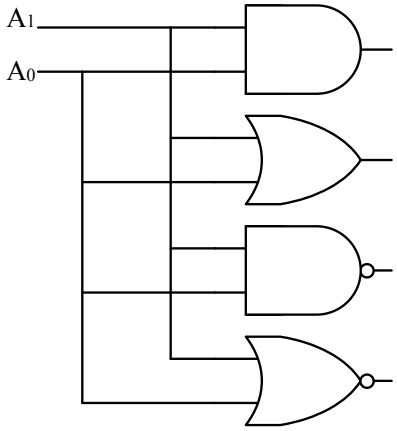
**Velg ett alternativ:**






Maks poeng: 2

## 11 Maskinkode kompatibilitet

Du har kompilert maskinkode for en spesifikk ARM mikroarkitektur "A". Hvilke andre mikroarkitekturer vil denne koden være kompatibel med?

Hvilket utsagn er korrekt under.

**Velg ett alternativ:**

- Kun mikroarkitekturer som baserer seg på samme arkitektur som "A". ✔
- Alle ARM mikroarkitekturer.
- Kun kompatibel med mikroarkitekturen som maskinkoden er kompilert for.
- Ingen av utsagnene er korrekte.
- Alle mikroarkitetkurer som har like mange pipeline steg som "A".

---

Maks poeng: 2

# 12 Funksjonskall ARM assembler

Under er det utsnitt av et enkelt eksempel på funksjonskall i Arm assembler

```
0x00000200 MAIN    BL FUNC
0x00000204         ADD R4, R5, R6
...
0x00401020 FUNC    ADD ...
....
0x004010XX         MOV PC, LR
```

Fyll inn riktig informasjon under. Her skal du følge ARM sin funksjonskallkonvensjon.

- BL instruksjonen setter LR til Velg alternativ (0x00000208, PC + 4, 0x00401020, PC + 8) .
- Funksjonen FUNC må legge verdien den ønsker å returnere til MAIN i registerverdien Velg alternativ (R0, R4, SP, LR).
- Funksjonen FUNC må sørge for å ikke overskrive Velg alternativ (R4-R11, R0-R3, stack innhold under SP, R12).

---

Maks poeng: 6

# 13 Branch Target Adress

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

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

**Velg ett alternativ:**

- -16
- -14
- -6 ✔
- ingen av verdiene er riktige
- 4

Maks poeng: 2

# 14 Oversette til assembler

Vi ønsker å oversette følgende *for løkke* til ARM assembler.

```
for (i = 6; i != 0; i = i-1)
   sum = sum+i;
```

I assemblerkoden under skal du anta at '*i*' ligger i '*R0*' og '*sum*' ligger i '*R1*'. Velg de riktige instruksjonene under slik at assemblerkoden blir samsvarende med *for løkken* over.

*### Start av assemblerkode ###*

```
      MOV R0, #6    ; i
      MOV R1, #0    ; sum
FOR
```

Velg alternativ (BLT FOR, ADDS R0, R0, R1, CMP R0, #1, ADD R1, R1, R0)

Velg alternativ (ADD R0, R0, R1, ADDS R1, R1, R0, SUBEQ R0, R0, #1, SUBS R0, R0, #1)

Velg alternativ (BLT FOR, BNE FOR, B FOR, BMI FOR)

---

Maks poeng: 6

## 15 Maskinkode

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

$0xE2454003$

Velg alternativ (SUB, ANDEQ, ADDLT, SUBS) Velg alternativ (R5, R3, R6, R4)
Velg alternativ (R5, R4, R0, R3) Velg alternativ (R3, #3, #1, R1)

---

Maks poeng: 6

## 16 Mikroarkitektur og ytelse

Hva er korrekt om mikroarkitektur og ytelse?

**Velg ett alternativ:**

- Vi vil ikke få problemer med *hazards* i en multi-cycle mikroarkitektur. ✔
- Vi vil typisk få en mer strømgjerrig mikroarkitektur dersom vi benytter oss av mange pipeline steg.
- Det er fordelaktig med en høy CPI.
- En pipeline mikroarkitektur vil typisk ha en effektiv CPI på 1.
- I en multicycle mikroarkitektur vil alle instruksjonene typisk ta like lang tid å utføre.

---

Maks poeng: 2

# 17 Kjøretid til forskjellige mikroarkitekturer

Gitt følgende assemblerkode. Her har vi nummerert instruksjonene fra fra 1 til 5

**#1 MOV R0, #1**
**#2 ADD R0, R1, R2**
**#3 SUB R1, R1, R0**
**#4 ORR R0, R1, R2**
**#5 ADD R2, R1, R2**

Anta mikroarktekturene som vi har gått igjennom i kurset (som er beskrevet i læreboka). Du skal videre anta at Single-cycle mikroarkitekturen har en klokkefrekvens på 1 GHz, som gir en klokkeperiode på 1 ns. Multi-cycle og Pipeline mikroarkitekturene har begge en klokkefrekvens på 2 Ghz. Du skal anta at multicycle har en fast CPI på 4. Pipeline mikroarkitekturen har 5 steg og har en hazard enhet med data-forwarding (som beskrevet i læreboka).

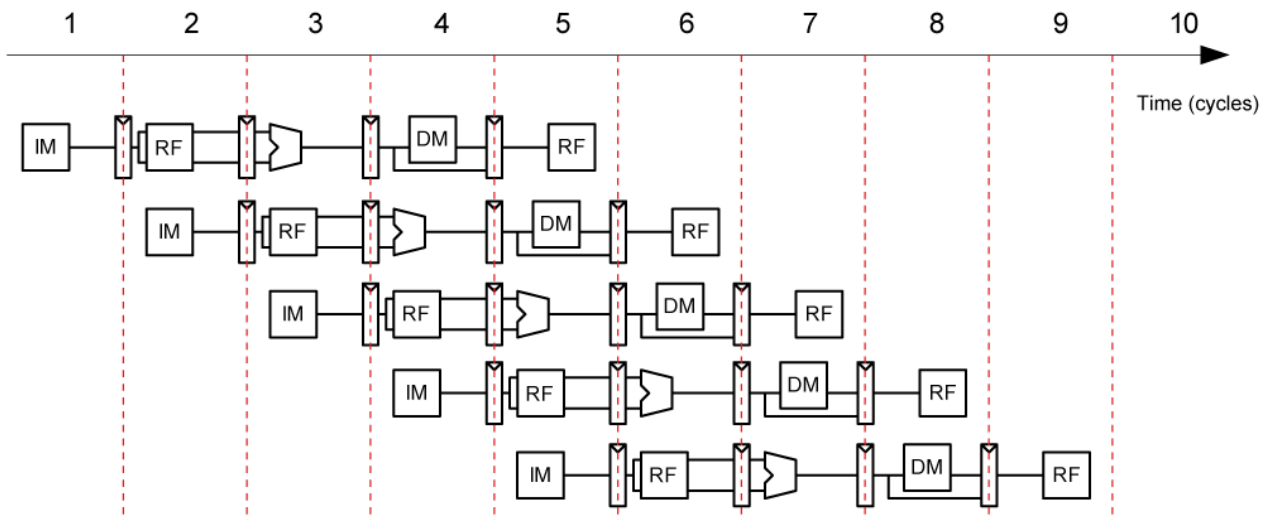
Angi korrekte tall i utsagnene under.

- Single-cycle prossesoren blir ferdig med koden iløpet av [ ] (5) ns.
- Pipeline prosessoren vil bli ferdig iløpet av [ ] (4,5 - 4.5) ns.
- Pipeline prosessoren vil være opptatt med å utføre opptil [ ] (5) instruksjoner samtidig.
- Når Pipeline processoren blir ferdig med programkoden vil multi-cycle prosessoren være opptatt med å utføre instruksjon nummer # [ ] (3) .

---

Maks poeng: 4

# 18 Pipeline




Gitt følgende ARM-assemblerprogram:

**STR R7, [R1, #24]**
**LDR R0, [R2, #20]**
**AND R8, R1, R7**
**SUB R1, R7, R0**
**ADD R4, R7, R5**

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

Hvilken hazard har vi i følgende klokkesykler?

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

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

---

Maks poeng: 8

# 19 Control Hazards

Hva er korrekt om *Control Hazards*. Her skal du ta utgangspunktet i den forenklede pipeline mikroarkitekturen som vi har gjennomgått i kurset/læreboken.

**Velg ett alternativ:**

- Ingen av utsagnene er korrekte.
- Det vil ofte lønne seg å vente til en branch er avklart før vi leser inn nye instruksjoner.
- Miss prediction penalty er antall instruksjoner vi må flushe når en branch prediksjo ✔ ller.
- Ubetingete branch instruksjoner vil aldri gi control hazards.
- Vi kan *flushe* instruksjoner for å øke pipeline ytelsen.

---

Maks poeng: 2

# 20 Aksesstid

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

Vi har et prosessorsystem med to nivå med cache og et fysisk minne.
Oppslag i det fysiske minnet tar 100 klokkesykler. Oppslag i nivå 2 cache tar 20 klokkesykler. Oppslag i cachen nærmest prosessoren (nivå 1) tar en klokkesykel.

Vi har en algoritme med mange minneaksesser. Oppslagene i cachen nærmest prosessoren har en treffrate på 80%. Oppslagene i nivå 2 cache har en treffrate på 90%. Oppslag i hovedminne gir alltid treff.

Hva blir gjennomsnittlig aksesstid for ett vilkårlig minneoppslag:

(6,35 - 6,45) klokkesykler

Oppgaven åpner for ulike tolkninger av antall klokkesykler det tar å aksessere underliggende nivåer
(20 vs 21 og 100 vs 121 )
Ved bruk av 21 og 121, så blir riktig svar 7,0 klokkesykler

Vi kan kjøre algoritmen på en nyere prosessor med kun ett nivå cache. Ettnivås cachen har en høyere treffrate. Aksesstid for cache er en klokkesykel som i den forrige arkitekturen, og hovedminnet bruker også 100 klokkesykler. Klokkefrekvensen er den samme for begge arkitekturene.

Hvor stor må treffraten for cachen i det nye prosessorsystemet være for at det skal lønne seg å benytte dette fremfor det første?

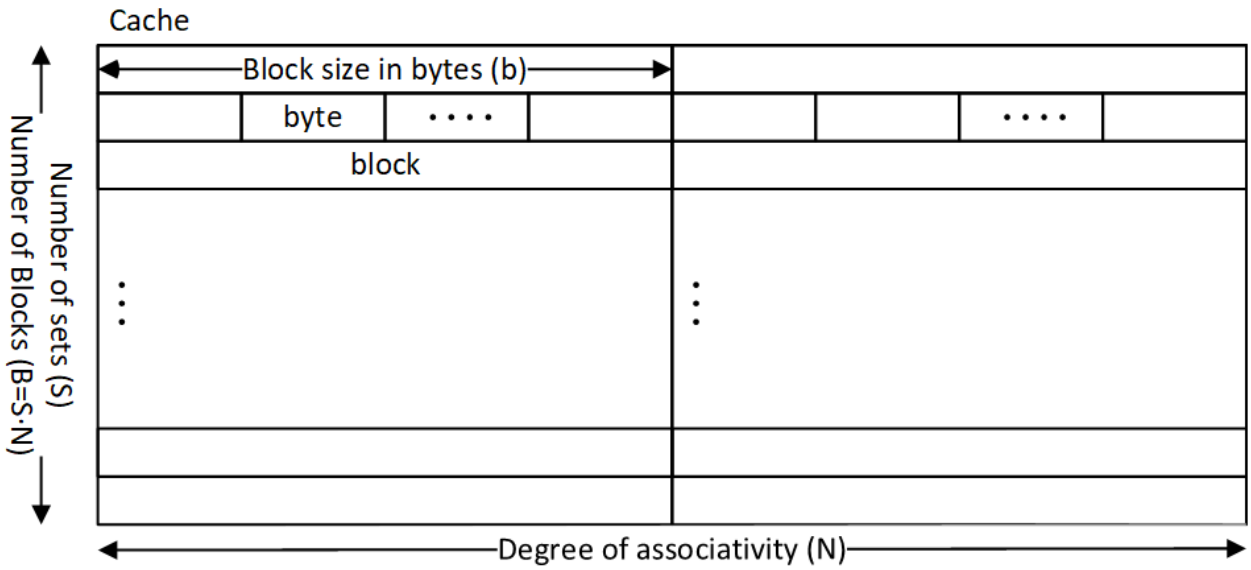
(94 - 95) %

Alternativt 94,0%

Maks poeng: 4

# 21 Cache

**I denne oppgaven skal du fylle inn heltall.**




Figuren over viser en generisk cache.

Til et fysisk minne på 512 MB bruker vi en direkte-mappet cache (N=1) med blokkstørrelse på fire ord, fire byte per ord, og kapasitet på 32 kB. Hver byte i minnet har en egen adresse.

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

(29)

Hvor mange set vil cachen ha?

(2000 - 2050)

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

(14)

Maks poeng: 6

# 22 Minneoppslag

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

Vi har en direkte-mappet cache med kapasitet på 16 ord og en blokkstørrelse på 4 ord og en ordstørrelse på 4 byte.

Et program gjør minneaksess fra følgende adresser i sekvens:

0x40, 0x44, 0x48, 0x84, 0x60, 0x64, 0x40, 0x44, 0x48, 0x84, 0x60, 0x64

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

(5)

Hva er treffraten for disse minneaksessene?

(0,58 - 0,59)

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

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

(0,745 - 0,755)

Hva konvergerer miss-raten mot dersom sekvensen gjentas uendelig mange ganger i når programmet kjøres med to-veis cachen?

(0,495 - 0,505)

Maks poeng: 10

**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 ^ 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 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 ^ 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 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 |

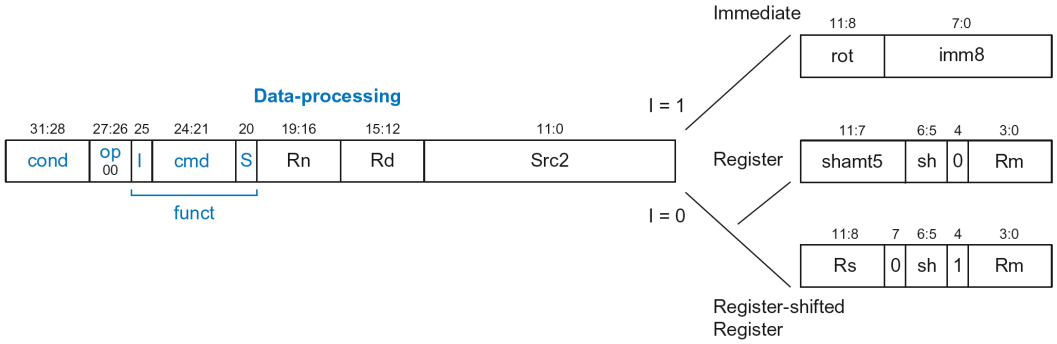



Figure 1: Data processing instruction format

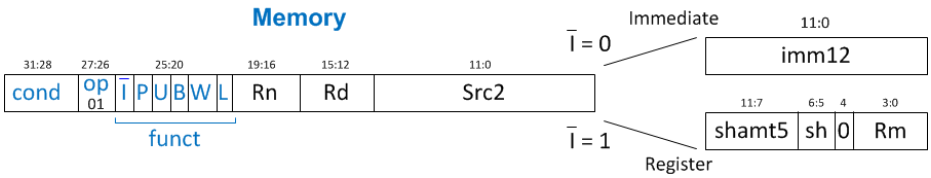



Figure 2: Memory processing instruction format

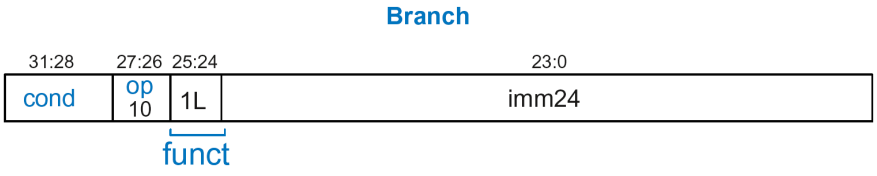



Figure 3: Branch instruction format

**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 17**
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 |