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 representation 2	3	Flervalg
3	Bit resolution	1	Fyll inn tall

Combinational logic

Oppgave	Tittel	Maks poeng	Oppgavetype
4	Boolean circuits to functions	2	Flervalg
5	Boolean algebra	3	Flervalg

Sequential logic

Oppgave	Tittel	Maks poeng	Oppgavetype
6	Combinational and sequential logic	2	Flervalg (flere svar)
7	Sekvensielle kretser	12	Nedtrekk

HDL

Oppgave	Tittel	Maks poeng	Oppgavetype
8	HDL	14	Nedtrekk

Digital building blocks

Oppgave	Tittel	Maks poeng	Oppgavetype
9	Shifter	2	Flervalg
10	Look up table	2	Flervalg

Computer Architecture

Oppgave	Tittel	Maks poeng	Oppgavetype
11	Computer architecture	2	Flervalg
12	Procedure Call Standard convention	4	Flervalg (flere svar)
13	Translate to Assembler	6	Nedtrekk
14	Branch Target Adress	2	Flervalg
15	Machine code	6	Nedtrekk

Mikroarkitektur

Oppgave	Tittel	Maks poeng	Oppgavetype
16	The difference between architecture and microarchitecture	2	Flervalg
17	Microarchitecture performance	2	Flervalg
18	Microarchitecture amount of clock cycles	3	Fyll inn tall
19	Pipeline	8	Nedtrekk
20	Pipeline control signals	2	Flervalg

Minnesystemer

Oppgave	Tittel	Maks poeng	Oppgavetype

21	Cache 1	6	Fyll inn tall
22	Cache 2	8	Fyll inn tall
23	Virtual memory	6	Fyll inn tall

Informasjon

Skriftlig eksamen
IN2060 - Digitalteknikk og datamaskinarkitektur
Høst 2021
Varighet: 3. desember kl. 15:00 til 3. desember kl. 19:00
Tid for eksamen: 4 timer
Hjelpemidler: Ingen

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

Generell informasjon:

- Besvarelsen din skal reflektere ditt eget, selvstendige arbeid og skal være et resultat av din egen læring og arbeidsinnsats.
- Om du vil trekke deg fra eksamen, trykk på hamburgermeny oppe til høyre i Inspera og velg "Jeg vil trekke meg".

Samarbeid under eksamen:
Det er ikke tillatt å samarbeide eller kommunisere med andre under eksamen. Samarbeid og kommunikasjon vil bli betraktet som forsøk på fusk.

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 $(36)_{10}$ til et 8 bits binærtall.
Velg ett alternativ

- 00100110
- 00110110
- 00110101
- Ingen av alternativene er korrekte.
- 00100100

Maks poeng: 2

2 Digital representation 2

Digital representasjon
Gjør om desimaltallet $(-26)_{10}$ til et 8 bits binærtall på 2'ers kompliment form.
Velg ett alternativ

- 11100110
- Ingen av alternativene er korrekte.
- 01100111
- 11101010
- 00100110

Maks poeng: 3

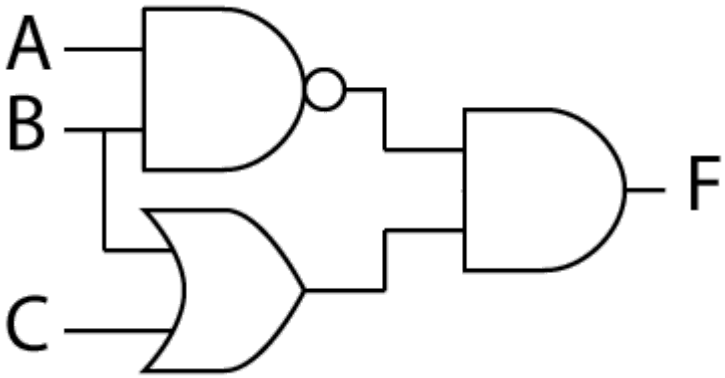
3 Bit resolution

Hva er minimum antall bit man trenger for å kunne utrykke 500 forskjellige fargetoner i en variabel? : .

Maks poeng: 1

4 Boolean circuits to functions

Hvilket funksjonsuttrykk gjenspeiler portimplementasjonen under?



Velg ett alternativ

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

Maks poeng: 2

5 Boolean algebra

Forenkle følgende utrykk maksimalt

F = AB + B(A'+ AC)
Velg ett alternativ

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

Maks poeng: 3

6 Combinational and sequential logic

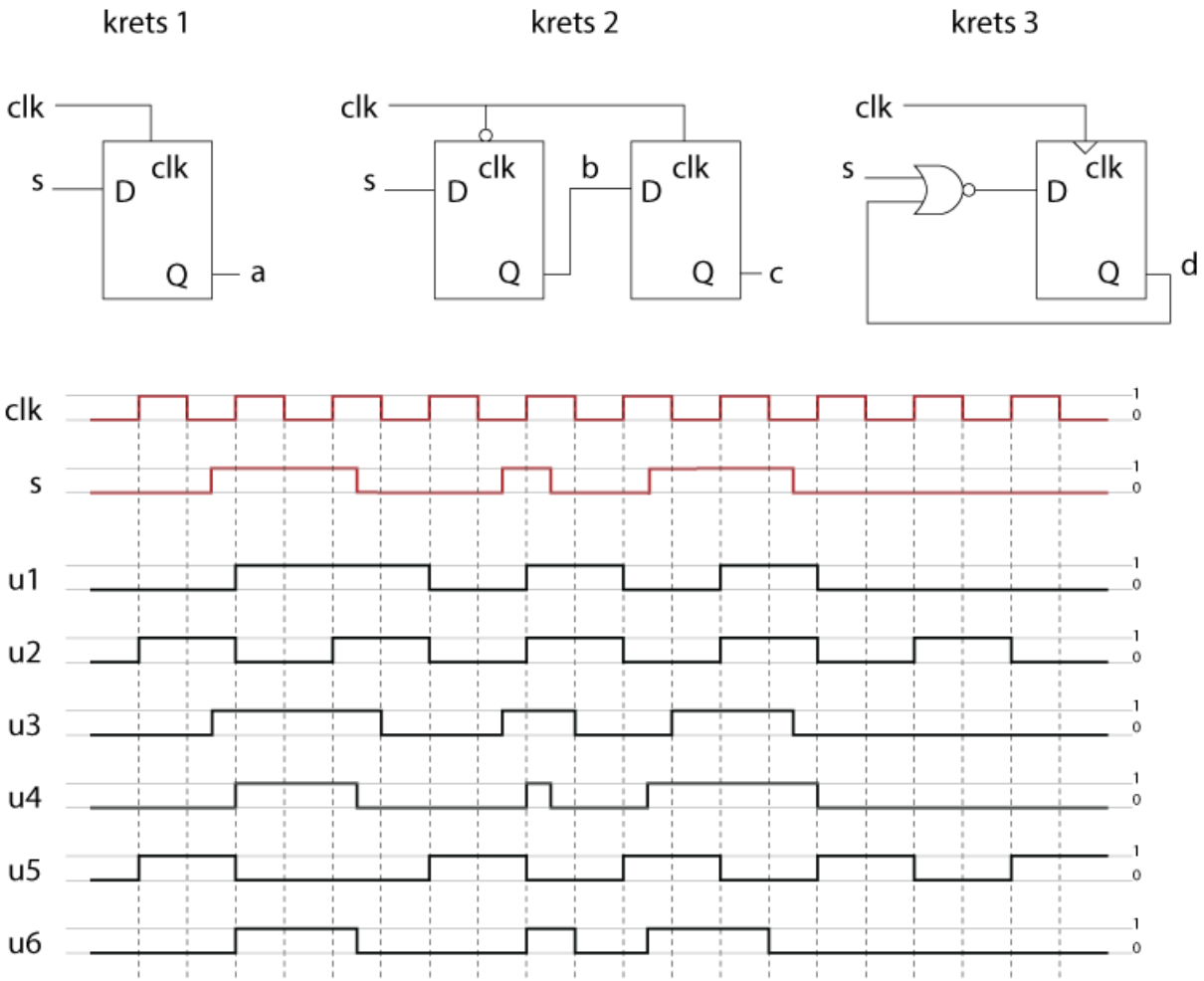
Hvilke to utsagn om kombinatorisk og sekvensiell logikk er **riktige**?
Velg de to alternativene som er riktige.

- Sekvensiell logikk kan inneholde kombinatorisk logikk.
- Det er ikke mulig å gi en unik beskrivelse av en kombinatorisk krets med en tabell.
- Synkron sekvensiell logikk er ikke avhengig av et klokkesignal.
- Kombinatorisk logikk kan huske tidligere utgangsverdier.
- Sekvensiell logikk inneholder låsekretser.
- Utgangen til sekvensiell logikk er kun et resultat av nåværende innganger.

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 HDL

I pdf'en (til venstre) er det fem forskjellige VHDL moduler (en per side).
I denne oppgaven skal du fullføre setningene slik at påstandene blir gyldige.
(*I denne oppgaven er det engelske ordlyd på alternativene, siden enkelte av begrepene kun har vært benyttet med engelsk ordlyd i kurset*).
Hint:
"Combinational" kan oversettes til "kombinatorisk" på Norsk.
RTL betyr "Register transfer level", og beskriver bruk av registre.

Krets 1 beskriver en Velg alternativ (prefiksadder, half adder, full adder, peltier adder, ripple carry adder, testbench, carry lookahead adder, flow adder, simulation module) som er fullstendig Velg alternativ (sequential, orthogonal, combinational, direct, recursive, indirect) og er skrevet med Velg alternativ (RTL, pinned, fluent, structural, gated, dataflow, behavioral)kodestil.

Krets 2 beskriver en Velg alternativ (sequential, orthogonal, recursive, combinational, direct, indirect) krets som er skrevet med Velg alternativ (dataflow, structural, gated, behavioral, RTL, fluent, pinned) kodestil.

I krets 3 implementerer komponenten «fulladder» en fulladder med a og b som input, c for mente inn og y som mente ut.
Krets 3 beskriver en Velg alternativ (ripple carry adder, carry lookahead adder, peltier adder, full adder, testbench, prefix adder, flow adder, half adder, simulation module) krets skrevet med Velg alternativ (behavioral, pinned, fluent, dataflow, RTL (Register transfer level), gated, structural) kodestil.

Krets 4 beskriver en Velg alternativ (simulation module, ripple carry adder, testbench, prefix adder, carry lookahead adder) krets skrevet ned Velg alternativ (RTL, fluent, gated, structural, behavioral, pinned) kodestil.

Krets 5 beskriver en Velg alternativ (harvesting module, simulation module, testbench, prefix adder, ripple carry adder, carry lookahead adder, digestion module) som er skrevet med Velg alternativ (structural, fluent, pinned, RTL, behavioral, gated) kodestil.

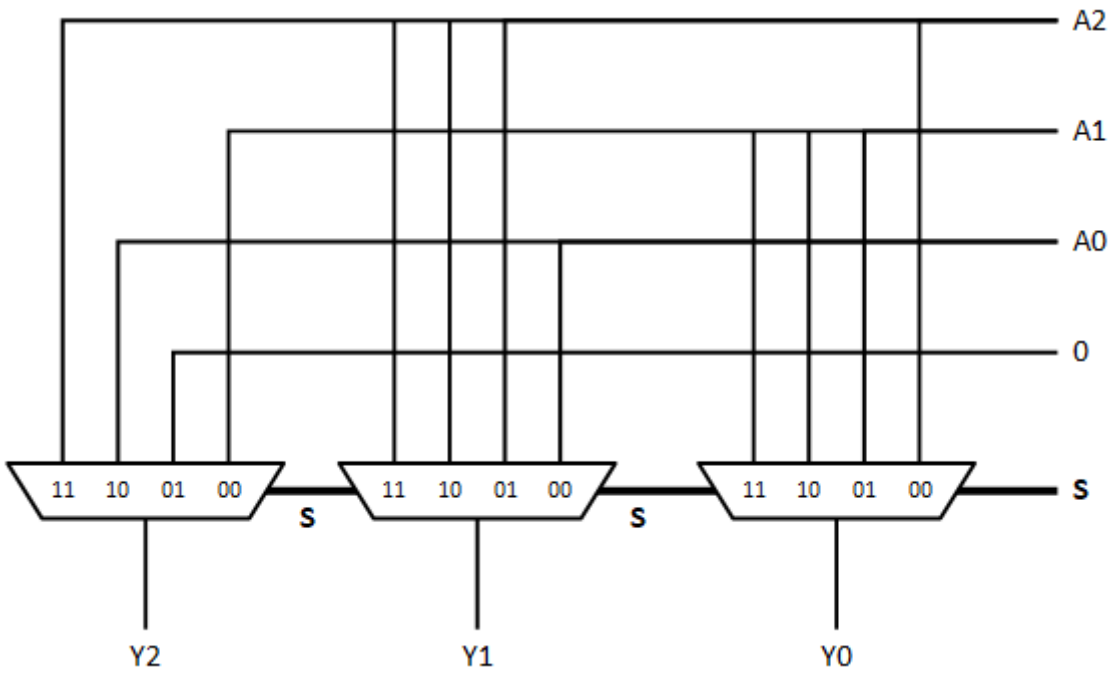
Det er tre navngitte komponenter, "fulladder_3", "component_5A" og "compenent_5B", som kan være implementasjoner av de andre modulene (kun navneendring må til) Hvilke av modulene kan fungere som de forskjellige komponentene?

Fulladder 3 kan være Velg alternativ (circuit 1, circuit 2, circuit 3, circuit 4, circuit 5, none of the above).
Component 5A kan være Velg alternativ (circuit 1, circuit 2, circuit 3, circuit 4, circuit 5, none of the above).

Component 5B kan være Velg alternativ (circuit 1, circuit 2, circuit 3, circuit 4, circuit 5, none of the above).

Maks poeng: 14

9 Shifter



Kretsen over shifter eller roterer et 3-bits signal 1 steg avhengig av select-signalet (S).
Hvilken VHDL funksjon er implementert for de forskjellige S-verdiene?
Hint: L = Logisk og A = aritmetisk for shift-operasjoner. S = Shift, RO = Rotate, L = Left, R= Right

S = 00 gir
Velg ett alternativ:

SRA ○ SLL ○ ROR ○ ROL ○ SLA ○ SRL ○

S = 01 gir
Velg ett alternativ

ROR ○ SRA ○ SRL ○ ROL ○ SLL ○ SLA ○

S= 10 gir
Velg ett alternativ

ROL ○ SLL ○ SRL ○ SRA ○ ROR ○ SLA ○

S= 11 gir

Velg ett alternativ

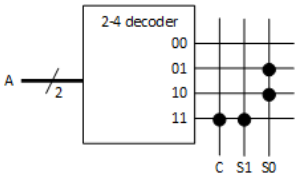
SRA	SLL	ROR	SLA	ROL	SRL
○	○	○	○	○	○

Maks poeng: 2

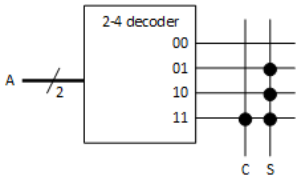
10 Look up table

Hvilken av disse oppslagstabellene virker som en halvadder?

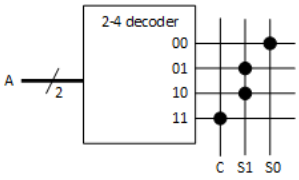
Velg ett alternativ:



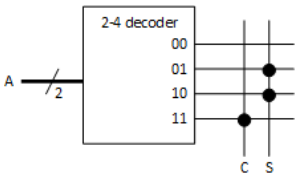
○



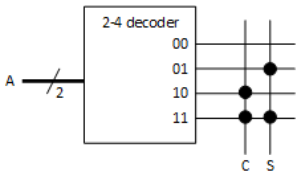
○



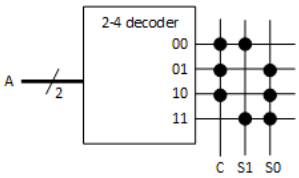
○



○



○



○

Maks poeng: 2

11 Computer architecture

Hva er korrekt om datamaskinarkitektur under?
Velg ett alternativ:

- ARM er av prosessortypen CISC.
- Arkitekturen definerer alle instruksjonene som en prosessor skal støtte.
- Godt designet maskinkode kan generelt kjøre på flere arkitekturer.
- Ingen av utsagnene er korrekte.
- ARM støtter svært mange kompliserte instruksjoner.

Maks poeng: 2

12 Procedure Call Standard convention

Basert på *funksjonskallkonvensjonen* til ARM og assemblerkoden under, hvilke registerverdier må funksjonen *F1* huske å mellomlagre på stacken?

```
F1:
   PUSH { ??? }
   ADD R3, R0, R1
   ADD R4, R2, R3
   SUB R5, R2, R3
   ORR R0, R5, R4
   POP  { ??? }
   MOV PC, LR
```

Velg de to riktige alternativene.

- [ ] LR
- [ ] R3
- [ ] PC
- [ ] R5
- [ ] R2
- [ ] R4
- [ ] R0
- [ ] R1

Maks poeng: 4

13 Translate to Assembler

Vi ønsker å oversette følgende program til ARM assembler. Du kan anta at '*a*' ligger i '*R0*' og '*i*' ligger i '*R1*'. Velg riktige instruksjoner under.

```
if (a < i)
   a = a + i;
else
   a = a – i;
```

Velg alternativ (CMP R0, R1, BLT L1, CMP R0, #1, ADD R0, R0, R1)

Velg alternativ (LSL R0, R1, R0, ADDMI R0, R0, R1, SUB R0, R1, R0, ADDVS R0, R1, R0)

Velg alternativ (SUBPL R0, R0, R1, SUBNE R0, R0, R1, SUBGE R0, R0, #1, SUBLT RO, RO, #1)

Maks poeng: 6

14 Branch Target Adress

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

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

Velg ett alternativ:

- 18
- 3
- ingen av verdiene er riktige
- 6
- 2

Maks poeng: 2

15 Machine code

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

$0xE2901002$

Velg alternativ (ANDEQ, ADD, AND, ADDS) Velg alternativ (R2, R1, R3, R0)
Velg alternativ (R0, R3, R2, R1) Velg alternativ (#2, R1, R0, R2)

Maks poeng: 6

16 The difference between architecture and microarchitecture

Hva er riktig om arkitektur og mikroarkitektur?
Velg ett alternativ:

- Mikroarkitekturen kan velge hvilken instruksjoner den ønsker å støtte.
- Det er hovedsaklig arkitekturen som bestemmer hvorvidt en CPU får høy ytelse eller ikke.
- Man står fritt til å velge egen mikroarkitektur løsning når man skal implementere en arkitektur.
- Arkitekturen beskriver minimum antall pipeline steg som prosessoren må ha.
- Ingen av utsagnene er korrekte.

Maks poeng: 2

17 Microarchitecture performance

Hva er korrekt om mikroarkitektur og ytelse?
Velg ett alternativ:

- Pipeline mikroarkitekturer vil typisk ha en høyere MIPS ytelse enn Single-cycle.
- CPI'en til en pipeline arkitektur er entydig definert av antall pipeline steg.
- Single-cycle design gir typisk løsninger med effektiv utnyttelse av hardware.
- Single-cycle design vil typisk gi mikroarkitekturer med høy klokkehastighet.
- Multicycle design gir typisk mikroarkitekturer med svært høy IPC.

Maks poeng: 2

18 Microarchitecture amount of clock cycles

Gitt følgende assemblerkode

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

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

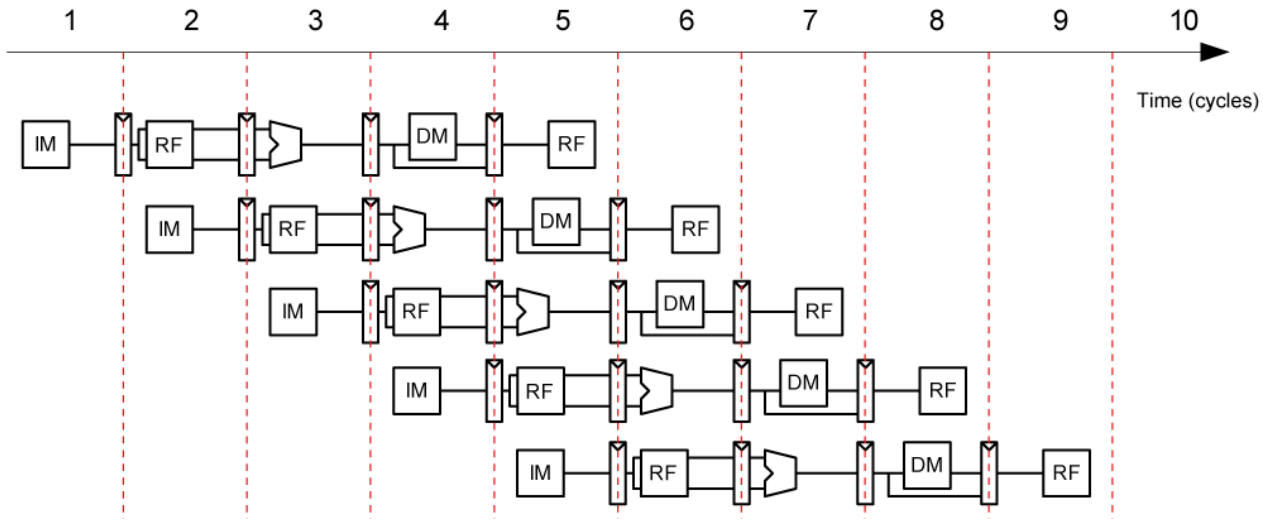
- Et *Single-cycle* design som beskrevet i boka.: [ ].
- Et *Multicycle* design der du kan anta en fast CPI på 4 for alle instruksjonstyper [ ].
- Et 5 stegs *Pipeline* design med hazard enhet som beskrevet i boka [ ].

Maks poeng: 3

19 Pipeline

Gitt følgende ARM-assemblerprogram:

ORR R0, R1, R2
STR R6, [R1, #20]
ADD R7, R1, R2
AND R6, R6, R0
SUB 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, skriver til registeret, leser fra registeret, leser fra og skriver til 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 (skriver til registeret, leser fra og skriver til registeret, leser fra registeret, ingen register aktivitet)
- I sykel 6 Velg alternativ (skriver til registeret, leser fra og skriver til registeret, leser fra registeret, ingen register aktivitet)

Hvilken hazard har vi i følgende klokkesykler?

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

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

Maks poeng: 8

20 Pipeline control signals

Hvordan sørger vi for at kontrollsignalene til en Pipeline mikroarkitektur blir korrekte?
Velg ett alternativ:

- Vi benytter oss av en tilstandsmaskin som sørger for at hvert steg av instruksjonene får riktige kontrollsignaler.
- Vi benytter av oss pipeline buffrene til å forsinke kontrollsignalene, slik at de følger instruksjonen.
- Vi benytter oss av en extend modul som sørger for kontrollsignalene blir utvidet på riktig måte.
- Vi benytter MUX'er som sørger for at kontrollsignalene blir rutet til riktig del av instruksjonen.
- Vi benytter oss av *Data Forward* som sørger for at dataflyten følger kontrollsignalene.

Maks poeng: 2

21 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

22 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, 0x08C, 0x424, 0x42C, 0xC2C

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?

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)?

Maks poeng: 8

23 Virtual memory

Vi har et system med virtuelt minne som kan adressere 2^{32} bytes.
Systemet har ubegrenset harddiskstørrelse, men kun 32MB fysisk minne. Vi antar både virtuelt og fysisk minne har 4KB sider (4KB 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 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

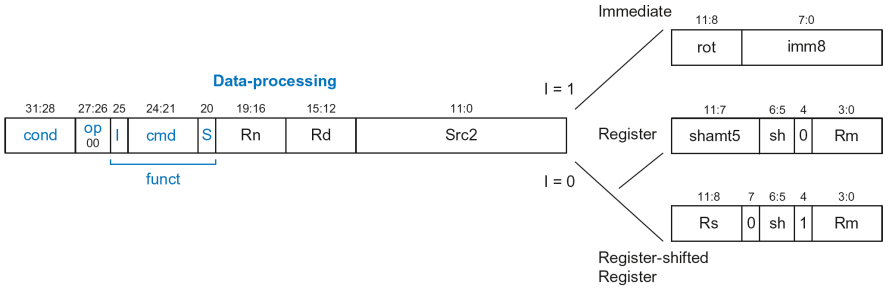


Figure 1: Data processing instruction format

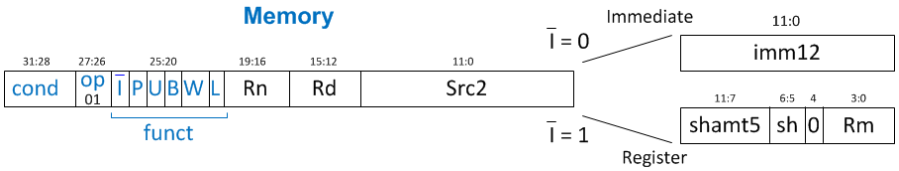


Figure 2: Memory processing instruction format

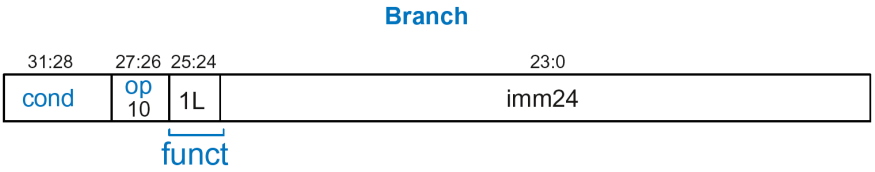


Figure 3: Branch instruction format

Question 18
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

Question 8
Attached

```vhdl
entity circuit_1 is
  port(
    a : in std_logic;
    b : in std_logic;
    c : in std_logic;
    x : out std_logic;
    y : out std_logic);
end entity;

architecture style_1 of circuit_1 is
begin
  x <=
    (a and b and c) or
    (a and not (b or c)) or
    (b and not (a or c)) or
    (c and not (a or b));

  y <=
    (a and b) or
    (a and c) or
    (b and c);
end architecture;
```

```vhdl
entity circuit_2 is
  port(
    reset : in std_logic;
    clk   : in std_logic;
    a     : in std_logic;
    b     : in std_logic;
    c     : in std_logic;
    x     : out std_logic;
    y     : out std_logic);
end entity;

architecture style_2 of circuit_2 is

begin
  process(clk) is
  begin
    if rising_edge(clk) then
      if reset then
        x <= '0';
        y <= '0';
      else
        x <=
          '1' when
            (a xor b xor c) = '1' else
          '0';
        y <=
          '1' when
            ((a and b) = '1') or
            ((a and c) = '1') or
            ((b and c) = '1') else
          '0';
      end if;
    end if;
  end process;
end architecture;
```

```vhdl
entity circuit_3 is
  port(
    a  : in std_logic_vector(7 downto 0);
    b  : in std_logic_vector(7 downto 0);
    c  : in std_logic;
    x  : out std_logic_vector(7 downto 0);
    y  : out std_logic
  );
end entity;

architecture style_3 of circuit_3 is
  component fulladder_3 is
    port(
      a     : in std_logic;
      b     : in std_logic;
      c     : in std_logic;
      x     : out std_logic;
      y     : out std_logic
    );
  end component;

  signal c_sig : std_logic_vector(7 downto 0);
  signal y_sig : std_logic_vector(7 downto 0);
begin
  INSTANTIATION: for i in 0 to 7 generate
    I_COMP: fulladder_3
    port map(
      a => a(i),
      b => b(i),
      c => c_sig(i),
      x => x(i),
      y => y_sig(i)
    );
  end generate;

  y <= y_sig(7);
  c_sig <= y_sig(6 downto 0) & c;

end architecture;
```

```vhdl
entity circuit_4 is
  port(
    a  : in integer;
    b  : in integer;
    c  : in std_logic;
    x  : out std_logic_vector(7 downto 0);
    y  : out std_logic
  );
end entity;

architecture style_4 of circuit_4 is
begin
  process(all) is
    variable v: integer;
  begin
    v := (a + b + 1) when c else (a + b);
    x <= std_logic_vector(to_unsigned(v, 8));
    y <= '1' when v > 255 else '0';
  end process;
end architecture;
```

```
entity circuit_5 is
end entity;

architecture style_5 of circuit_5 is
  component component_5A is
    port(
      a  : in integer;
      b  : in integer;
      c  : in std_logic;
      x  : out std_logic_vector(7 downto 0);
      y  : out std_logic
    );
  end component;

  component component_5B is
    port(
      a  : in std_logic_vector(7 downto 0);
      b  : in std_logic_vector(7 downto 0);
      c  : in std_logic;
      x  : out std_logic_vector(7 downto 0);
      y  : out std_logic
    );
  end component;

  signal a, b : integer range 0 to 255 := 0;
  signal c : std_logic := '0';
  signal xA, xB : std_logic_vector(7 downto 0);
  signal yA, yB : std_logic;

begin
  SIM: component_5A
  port map(
    a => a,
    b => b,
    c => c,
    x => xA,
    y => yA
  );

  DUT: component_5B
  port map(
    a => std_logic_vector(to_unsigned(a,8)),
    b => std_logic_vector(to_unsigned(b,8)),
    c => c,
    x => xB,
    y => yB
  );

  STIMULI: process is
  begin
    wait for 20 ns;
    for i in 0 to 255 loop
      for j in 0 to 255 loop
        for k in 0 to 1 loop
          a <= i;
          b <= j;
          c <= '1' when k = 1 else '0';
          wait for 5 ns;
          assert (xA = xB) report ("Calculation error") severity failure;
          assert (yA = yB) report ("Carry error") severity failure;
          wait for 5 ns;
        end loop;
      end loop;
    end loop;
    report ("Finished OK!");
    std.env.stop;
  end process;
end architecture;
```