Information

Question	Question title	Marks	Question type
i	Informasjon		Information or resources

Digital representation

Question	Question title	Marks	Question type
1	Digital representation 1	2	Multiple Choice
2	Digital representation 2	3	Multiple Choice
3	Bit resolution	1	Numeric Entry

Combinational logic

Question	Question title	Marks	Question type
4	Boolean circuits to functions	2	Multiple Choice
5	Boolean algebra	3	Multiple Choice

Sequential logic

Question	Question title	Marks	Question type
6	Combinational and sequential logic	2	Multiple Response
7	Sekvensielle kretser	12	Inline Choice

HDL

Question	Question title	Marks	Question type
8	HDL	14	Inline Choice

Digital building blocks

Question	Question title	Marks	Question type
9	Shifter	2	Multiple Choice
10	Look up table	2	Multiple Choice

Computer Architecture

Question	Question title	Marks	Question type
11	Computer architecture	2	Multiple Choice
12	Procedure Call Standard convention	4	Multiple Response
13	Translate to Assembler	6	Inline Choice
14	Branch Target Adress	2	Multiple Choice
15	Machine code	6	Inline Choice

Mikroarkitektur

Question	Question title	Marks	Question type
16	The difference between architecture and microarchitecture	2	Multiple Choice
17	Microarchitecture performance	2	Multiple Choice
18	Microarchitecture amount of clock cycles	3	Numeric Entry
19	Pipeline	8	Inline Choice
20	Pipeline control signals	2	Multiple Choice

Minnesystemer

Question	Question title	Marks	Question type

21	Cache 1	6	Numeric Entry
22	Cache 2	8	Numeric Entry
23	Virtual memory	6	Numeric Entry

ℹ Informasjon

Written examination
IN2060 - Digital Design and Computer Architecture
Autumn 2021

Duration: 4 hours; December 3. 15:00 to December 3. 19:00

Permitted aids: None

It is important that you read this front page before you start.

General information:

- Your answer should reflect your own independent work and should be a result of your own learning and work effort.
- If you want to withdraw from the exam, press the hamburger menu at the top right of Inspera and select "Withdraw".

Collaboration during the exam:

It is not allowed to collaborate or communicate with others during the exam. Cooperation and communication will be considered as attempted cheating.

About the exercises

The exam consist of different types of exercises; some in which numbers shall be entered and different types of multiple choice exercises. Some exercises may have attachments necessary for solving each task.

Make sure you have read and answered all parts of each exercise, and use the scrollbars to check both tasks and information in the attachments. Attachments can be enlarged using the attachment menu line.

Multiple choice exercises using radio buttons can be changed but not turned off once an alternative is chosen. Exercises having more than one correct answer will allow as many checked boxes as there are correct answers. It is not possible to check more boxes than there are correct answers.

About score in this exam

It is possible to achieve a total of 100 points. The points obtainable for each exercise is listed in the overview page to allow each student to manage their time usage. There is no deduction of points for wrong answers.

Good luck!

1 Digital representation 1

Digital representation
Convert the decimal number $(36)_{10}$ into a 8 bit binary number.
Select one alternative:

- None of the alternatives are correct.
- 00100110
- 00100100 ✔
- 00110110
- 00110101

Maximum marks: 2

2 Digital representation 2

Digital representation
Convert the decimal number $(-26)_{10}$ into an 8 bit binary number on 2's complement form.
Select one alternative:

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

Maximum marks: 3

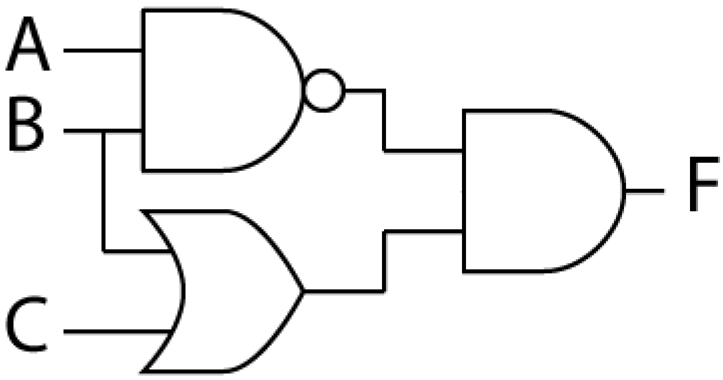
3 Bit resolution

What is the minimum number of bits needed to be able to express 500 different colors/hues in one variable? : [ ] (9).

Maximum marks: 1

4 Boolean circuits to functions

Which logical function F reflects the port implementation below?



Select one alternative:

- F = (AB)'(B+C) ✔
- None of the alternatives are correct.
- F = A'B'(B+C)
- F = AB+(B'+C')
- F = AB'+(B+C)

Maximum marks: 2

5 Boolean algebra

Find the minimal expression for F.

F = AB + B(A'+ AC)

Select one alternative:

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

Maximum marks: 3

6 Combinational and sequential logic

Which **two** statements about combinational and sequential logic are **correct**?

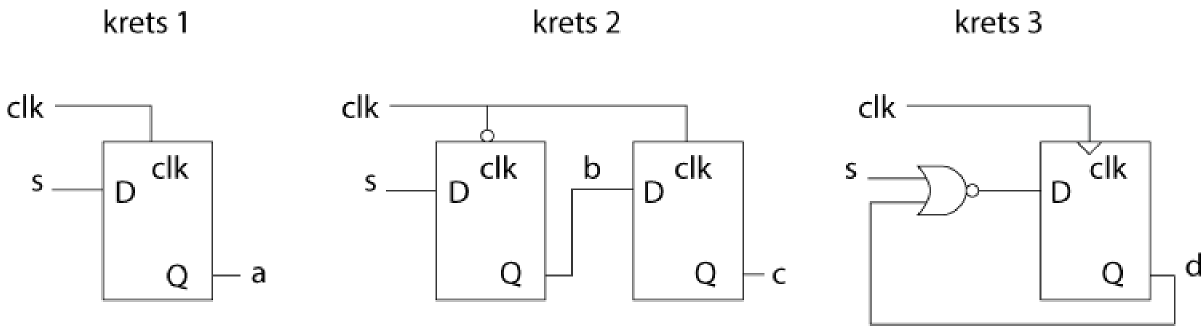
Select two alternatives:

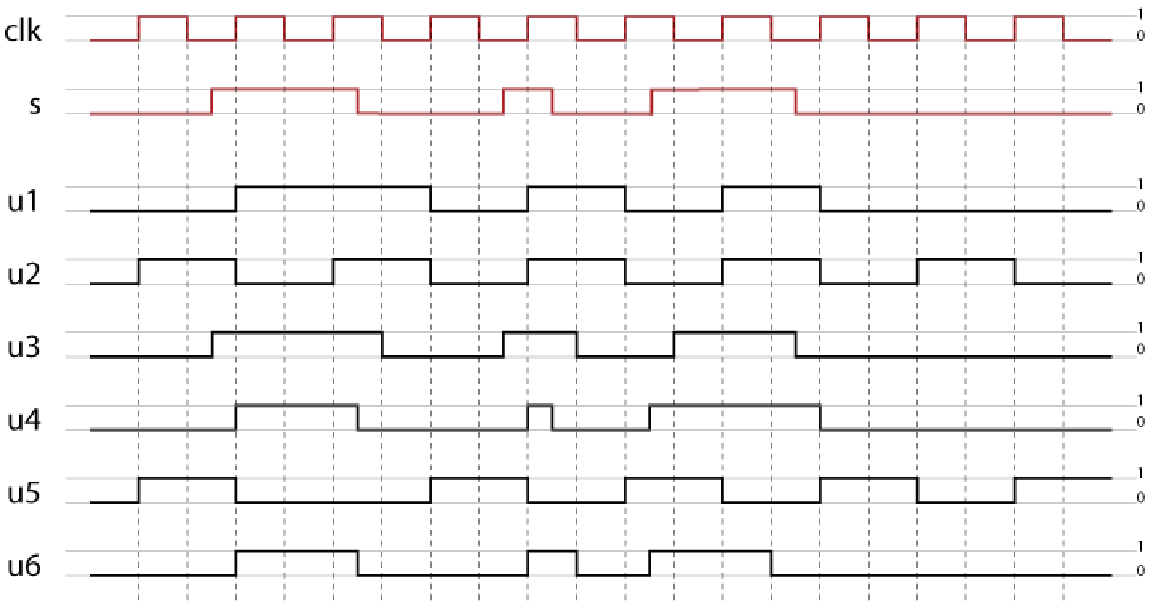
- ☐ The output of sequential logic is only a result of current inputs.
- ☐ Sequential logic contains bistable elements. ✔
- ☐ Sequential logic can contain combinational logic. ✔
- ☐ Synchronous sequential logic does not depend on a clock signal.
- ☐ Combinational logic can remember previous output values.
- ☐ It is not possible to give a unique description of a combinational circuit with a table

Maximum marks: 2

7 Sekvensielle kretser

Each of the three circuits below are fed the clock signal clk and the input signal s. Assume that the outputs a, b, c and d have the start value 0. Which of the signals below (**u1** to **u6**) belong to the different outputs? Note that two extra signals have been given. You do not need to pay attention to gate delay. **Study the circuits carefully and notice the difference between latches and flip flops in the illustration.**





- Output **a** corresponds to Select alternative (u1, u2, u3, u4, u5, u6)
- Output **b** corresponds to Select alternative (u1, u2, u3, u4, u5, u6)
- Output **c** corresponds to Select alternative (u1, u2, u3, u4, u5, u6)
- Output **d** corresponds to Select alternative (u1, u2, u3, u4, u5, u6)

Maximum marks: 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.

Hints:
"Combinational" translates to "kombinatorisk" in Norwegian.
RTL code style means "Register transfer level", and does describe register usage.

Circuit 1 describes a/an [Select alternative] (full adder, simulation module, prefix adder, half adder, carry lookahead adder, ripple carry adder, peltier adder, test_bench, flow adder) which is purely [Select alternative] (direct, sequential, indirect, recursive, combinational , orthogonal) and it is written using [Select alternative] (RTL, fluent, structural, dataflow, pinned, gated, behavioral)code style.

Circuit 2 describes a/an [Select alternative] (recursive, orthogonal, direct, sequential , indirect, combinational) circuit and it is written using [Select alternative] (fluent, structural, dataflow, gated, pinned, behavioral, RTL) code style.

In circuit 3, the component "fulladder" does implement a fulladder, using a and b as input, c for carry in and y for carry out.

Circuit 3 describes a/an [Select alternative] (testbench, simulation module, peltier adder, full adder, half adder, prefix adder, carry lookahead adder, flow adder , ripple carry adder) circuit which is written using [Select alternative] (pinned, gated, RTL (Register transfer level), behavioral, fluent, structural, dataflow) code style.

Circuit 4 describes a/an [Select alternative] (carry lookahead adder, ripple carry adder, prefix adder, simulation module, testbench) circuit which is written using [Select alternative] (gated, RTL, structural, behavioral, pinned, fluent) code style.

Circuit 5 describes a/an [Select alternative] (simulation module, digestion module, prefix adder, testbench, harvesting module, ripple carry adder, carry lookahead adder) which is written using [Select alternative] (gated, structural, fluent, behavioral, pinned, RTL) code style.

There are three named components, "fulladder_3", "component_5A" and "compenent_5B", which can be among the other modules (*renaming required*). Which modules may correspond to the components?

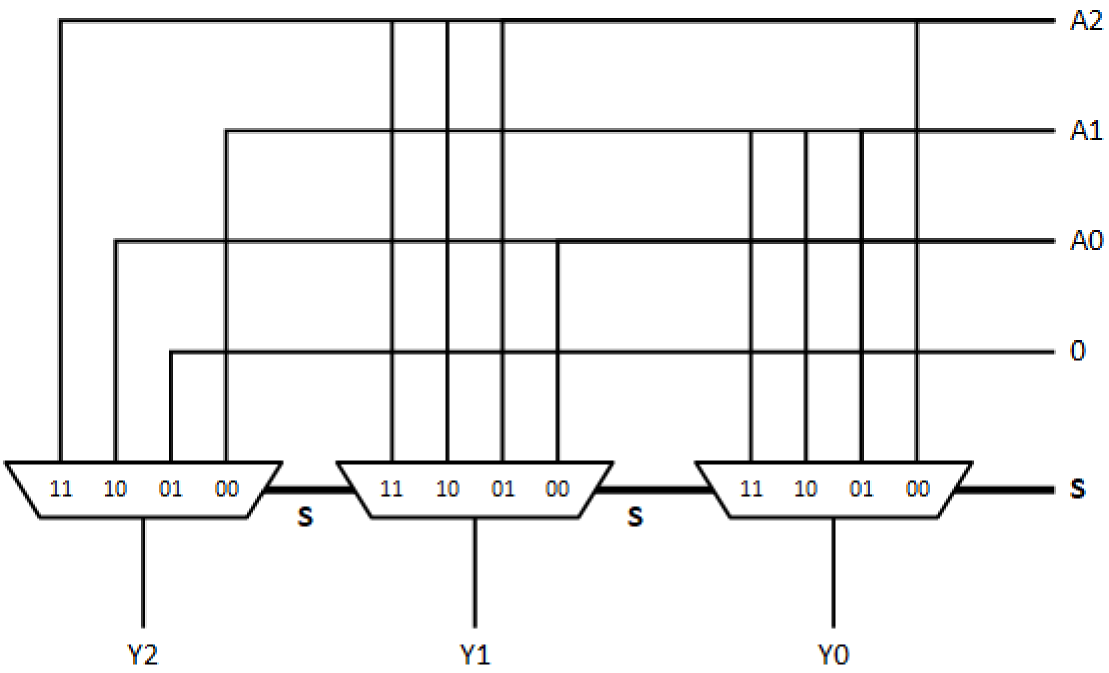
Fulladder 3 should be [Select alternative] (circuit 1, circuit 2, circuit 3, circuit 4, circuit 5, none of the above).

Component 5A should be [Select alternative] (circuit 1, circuit 2, circuit 3, circuit 4, circuit 5, none of the above).

Component 5B should be Select alternative (circuit 1, circuit 2, circuit 3, circuit 4, circuit 5, none of the above).

Maximum marks: 14

9 Shifter



The circuit above shifts or rotates a 3 bit signal 1 step depending on the select signal (S).
Which VHDL-function is implemented for the different S inputs?
Hint: *L = logical, A = Arithmetic for shift operations. S = Shift, RO = Rotate, L = Left, R= Right*

S = 00 selects
Select one alternative:

- SRA
- SLA
- ROL ✔
- ROR
- SLL
- SRL

S = 01 selects
Select one alternative:

- SRL ✔
- SLA
- ROR
- SRA
- ROL
- SLL

S= 10 selects
Select one alternative:

- ROR ✔
- SRL
- ROL
- SRA
- SLA
- SLL

S= 11 selects

Select one alternative:

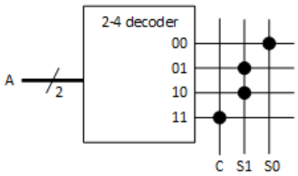
- SRL
- ROR
- SRA
- ROL
- SLA
- SLL

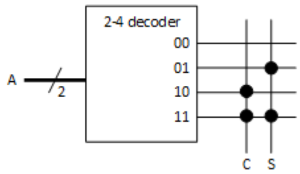
Maximum marks: 2

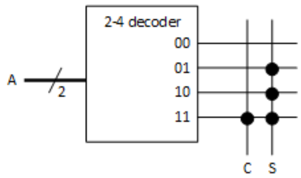
10 Look up table

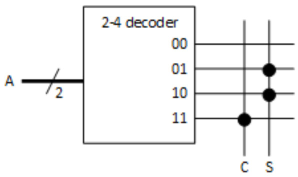
Which of these lookup tables represent a half adder?

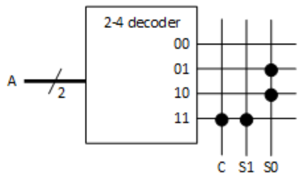
Select one alternative:

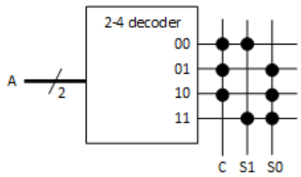












Maximum marks: 2

11 Computer architecture

What is the correct statement about computer architecture below?
Select one alternative:

- ◯ The architecture defines all the instructions that a processor should support. ✔
- ◯ None of the statements are correct.
- ◯ ARM is a processor type of CISC.
- ◯ Well-designed machine code can generally run on multiple architectures.
- ◯ ARM supports many complex instructions.

Maximum marks: 2

12 Procedure Call Standard convention

Based on the *Procedure Call Standard* convention for ARM, and the assembler code below, which register values must the function F1 remember to temporarily save on the stack?

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

Select two alternatives:

- ☐ PC
- ☐ R2
- ☐ R5 ✔
- ☐ R4 ✔
- ☐ R0
- ☐ LR
- ☐ R3
- ☐ R1

Maximum marks: 4

13 Translate to Assembler

We want to translate the following program to ARM assembler. You can assume that '*a*' is in '*R0*' and '*i*' is in '*R1*'. Select the correct instructions below.

if (a < i)
a = a + i;
else
a = a – i;

Select alternative (ADD R0, R0, R1 , CMP R0, #1 , CMP R0, R1 , BLT L1)

Select alternative (LSL R0, R1, R0, ADDVS R0, R1, R0, SUB R0, R1, R0, ADDMI R0, R0, R1)

Select alternative (SUBNE R0, R0, R1, SUBPL R0, R0, R1, SUBGE R0, R0, #1 , SUBLT RO, RO, #1)

Maximum marks: 6

14 Branch Target Adress

Given the section of the ARM assembler code below, what numerical value must the *imm24* field of the machine code of the Branch instruction (BLT) have?

```
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
```

Select one alternative:

- 18
- None of the values are correct
- 3 ✔
- 2
- 6

Maximum marks: 2

15 Machine code

Decode the following ARM instruction (machine code) as described in the course book, and select the options that form the corresponding assembler instruction.

$0xE2901002$

Select alternative (AND, ADD, ADDS, ANDEQ) Select alternative (R0, R3, R1, R2)
Select alternative (R1, R3, R2, R0) Select alternative (R2, #2, R1, R0)

Maximum marks: 6

16 The difference between architecture and microarchitecture

Which statement is correct about architecture and microarchitecture?
Select one alternative:

- ○ It is mainly the architecture that determines whether a CPU gets high performance or not.
- ○ None of the statements are correct.
- ○ The architecture describes the minimum number of pipeline stages that the processor must have.
- ○ The microarchitecture can choose which instructions it wants to support.
- ○ You are free to choose your own microarchitecture solution when implementing an architecture. ✓

Maximum marks: 2

17 Microarchitecture performance

Which statement is correct about microarchitecture and performance?
Select one alternative:

- ○ Pipeline microarchitectures will typically have a higher MIPS performance than Single-cycle. ✓
- ○ Single-cycle design typically provides microarchitectures with high clock speeds.
- ○ Multicycle design typically provides microarchitectures with very high IPC.
- ○ Single-cycle design typically provides solutions with efficient utilization of hardware.
- ○ The CPI of a pipeline architecture is uniquely defined by the number of pipeline steps.

Maximum marks: 2

18 Microarchitecture amount of clock cycles

Given the following assembler code

```
MOV R1, #1
MOV R2, #3
ADD R0, R1, R2
SUB R1, R1, R2
CMP R0, R1
```

How many clock cycles will the following microarchitecture designs use to run the above code?

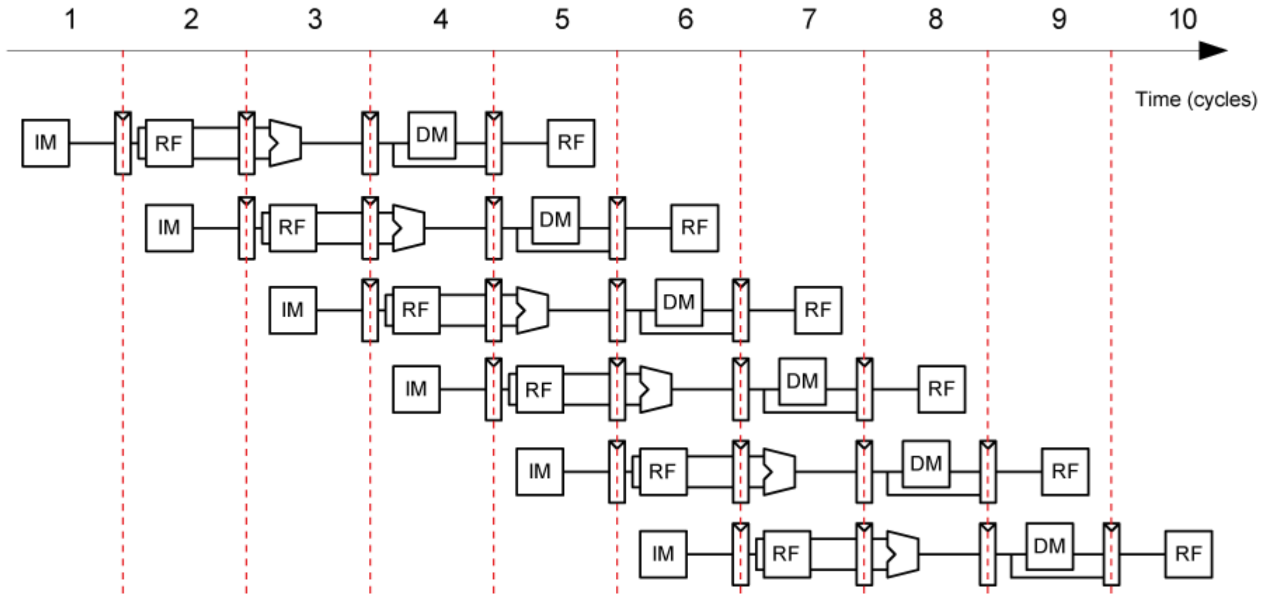
- A *Single-cycle* design as described in the course book: ☐ (5).
- A Multicycle design where you can assume a fixed CPI of 4 for all instruction types ☐ (20).
- A 5-stage Pipeline design with a hazard unit as described in the course book ☐ (9).

Maximum marks: 3

19 Pipeline

Given the following ARM assembler program:

ORR R0, R1, R2
STR R6, [R1, #20]
ADD R7, R1, R2
AND R6, R6, R0
SUB R4, R7, R5



What kind of pipeline sequence will the code above give? Assume a 5-step pipeline processor as illustrated above (similar to the course book), but without any kind of hazard handling. Here we write to the register file in the first half of the clock period, and read in the second half of the clock period. Select the options below that are correct for the pipeline sequence.

What kind of register activity do we have in the following clock cycles?

- In cycle 3 Select alternative (reads from and writes to the register, writes to the register, reads from the register, no register activity)
- In cycle 4 Select alternative (reads from the register, writes to the register, reads from and writes to the register, no register activity)
- In cycle 5 Select alternative (reads from and writes to the register, no register activity, reads from the register, writes to the register)
- In cycle 6 Select alternative (reads from and writes to the register, no register activity, reads from the register, writes to the register)

What kind of hazard do we have in the following clock cycles?

- In cycle 3 Select alternative (control hazard, no hazard, data hazard)
- In cycle 4 Select alternative (data hazard, control hazard, no hazard)

- In cycle 5 Select alternative (control hazard, data hazard, no hazard)
- In cycle 6 Select alternative (data hazard, control hazard, no hazard)

Maximum marks: 8

20 Pipeline control signals

How do we ensure that the control signals of a Pipeline microarchitecture are correct?

Select one alternative:

- We use the pipeline buffers to delay the control signals so that they follow the instru ✔ ons.
- We use *Data Forward* which ensures that the data flow follows the control signals.
- We use MUXs that ensure that the control signals are routed to the correct part of the instruction.
- We use a state machine that ensures that each step of the instructions receives the correct control signals.
- We use an extend module that ensures that the control signals are extended correctly.

Maximum marks: 2

21 Cache 1

We compare two processors systems with different cache setups. Other than the cache, the systems perform equally.

System A has an average hit rate of 85% when reading from the cache, while system B has an average hit rate of 93% when reading from cache.

For both systems, a cache hit result in an access time of 1 clock cycle, while reading from main memory has an average access time of 100 clock cycles. We assume a 100% hit rate in main memory.

For all answers in this exercise, up to three digits precision may be required.

a) What is the average access time (in clock cycles) for system A? [ ] (15,8 - 16)

b) What is the average access time (in clock cycles) for system B? [ ] (7,9 - 8)

For a given task, system A uses 90% of the time on memory access, the remaining time is used on calculations.

c) What is the proportion of time usage for B compared to A for the given task?

State the answer as a percentage: [ ] (54 - 56)%

Maximum marks: 6

22 Cache 2

Consider a direct mapped cache with a capacity of 2KB data. Each word is 4 byte and the block size is 4 words. Each byte is addressable.

Note: in a) and b) the answer is an integer value. For c) and d), up to three digit precision may be required.

a) How many sets are there in this cache? ☐ (128)

We read the following sequence of addresses once:
0x010, 0x014, 0x01C, 0x020, 0x08C, 0x424, 0x42C, 0xC2C

b) How many of these eight read operations will result in a cache-miss? ☐ (5)

We reboot the system (cache cleared), then read the same address-sequence as in b) exactly 10 times.

c) What will be the hit rate for this operation? ☐ (0,710 - 0,720)

We change the cache to a two-way set associative cache with the same (2kB) data capacity, but with a block size of 2 words.

d) What will be the new hit rate if we do the same read operation as in c), using the new cache?

☐ (0,91 - 0,92)

Maximum marks: 8

23 Virtual memory

Consider a virtual memory system that can address a total of 2^{32} bytes.
You have unlimited hard drive space, but are limited to 32 MB of physical memory. Assume that virtual and physical pages are 4KB in size.

All answers in this exercise are integer values.

How many bits is the physical address? ☐ (25)

How many bits are the virtual page numbers? ☐ (20)

How many bits are the physical page numbers? ☐ (13)

Maximum marks: 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

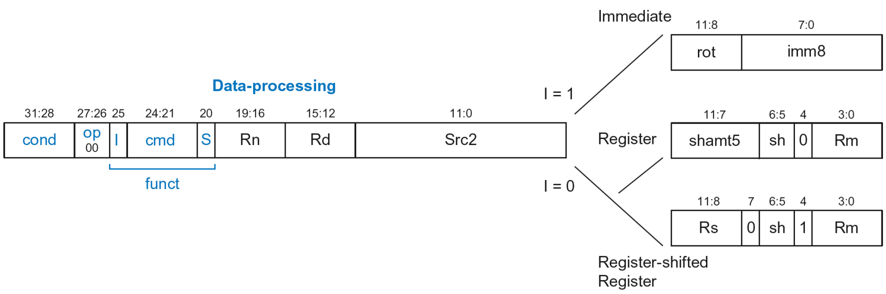


Figure 1: Data processing instruction format

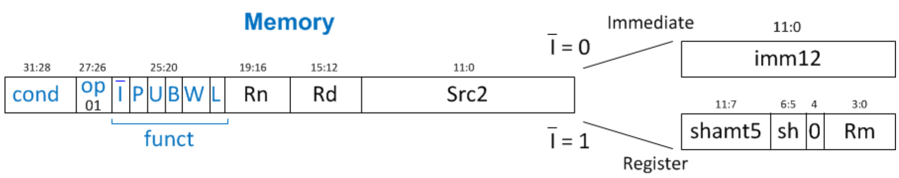


Figure 2: Memory processing instruction format

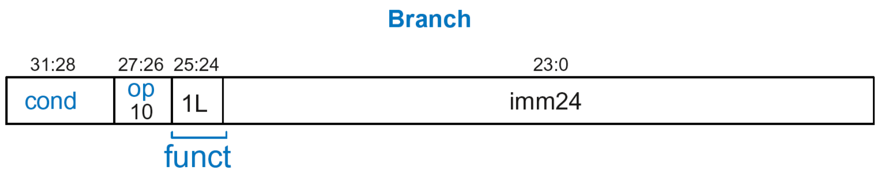


Figure 3: Branch instruction format

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	$(B \bullet C) + (B \bullet \overline{C}) = B$	$(B+C) \bullet (B+\overline{C}) = B$	Combining
T11	$(B \bullet C) + (\overline{B} \bullet D) + (C \bullet D) =$ $(B \bullet C) + (\overline{B} \bullet D)$	$(B+C) \bullet (\overline{B}+D) \bullet (C+D) =$ $(B+C) \bullet (\overline{B}+D)$	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;
```

```vhdl
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;
```