


Module	Computer Fundamentals / COMP1027 (CSF) / Semester 1
Module Convenor(s)	Tissa Chandesa

Assessment Name	Coursework 2	Weight	15%
Description and Deliverable(s)	<p>The coursework (details below) focuses on Combinatorial & Sequential Circuits of this module. It is an INDIVIDUAL WORK.</p> <p style="text-align: center;"><u>Preliminaries</u></p> <ul style="list-style-type: none"> In the previous coursework and lab sessions, we used combinatorial logic to construct many of the various core logic circuits that form the basis of a CPU. In this coursework, we continue some combinatorial circuits, but we are also going to start producing some of the various sequential circuits used in a computer (both inside the CPU, and outside to form memory). As you should recall from the lecture, sequential logic circuits are those that can store information and whose output is based not just on some combination of the inputs but also on the previous state of the circuit. The nand2tetris Hardware Simulator lets us construct this kind of circuit by using the supplied DFF chip (which mimics a D flip-flop), and the simulated clock. This clock can either be driven by a test script, or by pressing the 'clock' toolbar button (highlighted below):  <ul style="list-style-type: none"> The clock button enables you to simulate the oscillations of the clock signal within a computer. The DFF chip is setup to update its output pin out on the clock rising to high phase, storing whatever is currently at its input pin in. <p style="text-align: center;"><u>Important Notes</u></p> <ol style="list-style-type: none"> Do not worry if you didn't manage to complete the previous coursework, or your implementation doesn't work correctly. The nand2tetris project provides built-in implementations (implemented in Java) of all the gates required. To ensure you use these built-in implementations, we recommend you keep the files for this exercise in its own separate folder. You are highly recommended to read section A.7 — Sequential Chips — of Appendix A of <i>The Elements of Computing Systems: Building a Modern Computer From First Principles</i> — the book which accompanies nand2tetris — when implementing these circuits. Relevant chapters are on their website at: http://nand2tetris.org/course. The sequential components you will build in this coursework are similar to those in the third nand2tetris project and you will need to implement the logic circuits using the DFF primitive chip, alongside the gates you learnt and practiced with in the previous coursework and lab sessions (You are allowed to use the built-in versions of the basic gates, or your own if you prefer). We suggest that you work on them in the order specified: Bit, Register, RAM8, RAM64, RAM512, RAM4K, and RAM16K, since you can then use the earlier/smaller chips to build the more complex ones (e.g., Register can be defined relatively simply in terms of Bit, rather than having to specify it all from scratch). Again, Chapter 3 of the nand2tetris book contains more detailed descriptions of each circuit. Hint: You will be required to create feedback loop to implement Bit, where the output of a DFF is connected back to its input (possibly via some combinatorial logic). This can lead to errors from the Hardware Simulator if you use the name of Bit's output pin (out) as an input to another gate. You can get around this 		



by specifying two pins connect to the out pin of the `DFF`, one that connects to output pin of the `Bit`, and another that you can use to connect to other parts of the chip, e.g.: `DFF(in=..., out=out, out=x);` where `x` can be used to connect to other logic gates.

Files to Download

The skeleton `.hdl`, some `.tst` and `.cmp` files, for some of the chips required in this coursework, are provided in the [CW2-Files.zip](#). Whenever they are not provided, then you need to create them.

1. Download the zip file, from the "Coursework 2" area on Moodle.
2. Extract the files.
3. For each task (below), use the corresponding `.hdl` skeleton file to start implementing the required chips. You'll find that helpful and provide some additional hints (on top of what is already provided in the coursework). It will also save you starting from scratch, and make sure that you submit the correct files.

PART 1: Combinatorial Chips

Task A: Mux and DMux

1. **Implement the Mux gate** utilising any of the following gates, ONLY:

Not (...);
And (...);
Or (...);

2. **Implement the DMux gate.** *Hint: You can use two of the above gate types.*

SUBMISSION: SUBMIT your answers, i.e., completed `Mux.hdl` & `DMux.hdl` files. (And all associated testing scripts & compare files).

Task B: Mux4Way16, Mux8Way16, DMux4Way, DMux8Way

1. **Implement the Mux4Way16.** *Hint: Use 3 Mux16 chips. Each Mux16 takes two 16-bit inputs. Two Mux16s' 16-bit outputs are fed as the two 16-bit inputs of the third Mux16 {Remember the binary-tree-like construction. Drawing the tree would help in visualising the implementation}.* The skeleton code would look as below (you need to fill in "??" yourself):

Mux?? (a=a, b=b, sel=??, out=??);
Mux?? (a=c, b=d, sel=??, out=??);
Mux?? (a=??, b=??, sel=??, out=out);

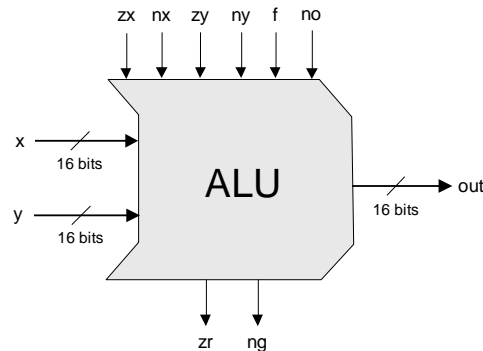
2. **Implement the Mux8Way16.** *Hint: Again, think in terms of binary tree construction of Mux16 gates.*
3. **DMux4Way, DMux8Way.** *Hint: Again, think in terms of binary tree construction of gates (nodes).*

SUBMISSION: SUBMIT your answers, i.e., completed `Mux4Way16.hdl`, `Mux8Way16.hdl`, `DMux4Way.hdl` & `DMux8Way.hdl` files. (And all associated testing scripts & compare files).

Task C: ALU

1. Make sure you do the first two tasks, first. Then, read Chapter 2, pages 35-38. Finally, revise the lecture slides and the **Oct 20, 2022**, lab session.

2. Implement an ALU chip (**n2tALU**) that computes 18 functions (see illustrative in Figure 1).



`out(x, y, control bits) =`
`x+y, x-y, y-x,`
`0, 1, -1,`
`x, y, -x, -y,`
`x!, y!,`
`x+1, y+1, x-1, y-1,`

Figure 1: ALU

Hint: If you understood and successfully implemented an ALU during our Oct 20, 2022, Lab Session, then task C will be very easy for you.

SUBMISSION: SUBMIT your answers, i.e., completed **n2tALU.hdl** files. (And all associated testing scripts & compare files).

PART 2: Sequential Chips

Task D: Bit, Register and RAM

1. Implement the Bit.

This has one input, `in`, and one output, `out`, and is designed to store a single bit of information. A further input controls whether the output changes to match the input or freezes at its current value. If `load` is true (1), then the output should be updated to match the input.

(i.e., $out(t) = in(t-1)$), otherwise, if `load` is false, the output should remain unchanged (i.e., $out(t) = out(t-1)$)

*Hint: The answer – chip implementation diagram – is already given in the lecture and lab. So, you simply need to write the corresponding hdl code for its implementation to run in the simulator. You will be using the **built-in DFF as your building block** in this task, and all the sequential circuits.*

2. Implement the 8-bit Register.

This has one 8-bit input bus, `in`, and one output bus, `out`, and is designed to store a single `byte` of information. As with Bit, a further input `load` controls whether the output should be updated to reflect the new input value (as Bit).

Hint: Again, 4-bit register is demonstrated in the lecture (also look at Chapter 3). 8-bit register is an array of 8 Bit chips. Pay close attention that the use of variable ($t+1$, t) in the descriptions refer to time, not the location of the “array” as you would consider in standard higher-level programming.

3. Implement RAM8.

This has one input bus, `in`, and one output bus, `out`. It defines an array of 8 “8-bit” registers. A second input bus, `address`, indexes which register is to be accessed. The `out` bus should contain the value stored in the register specified by `address`. A further pin, `load`, is used to decide whether the register specified by `address` should be updated by the value on the bus, `in`, or not.

Hint: The “memory (RAM8)” is essentially an array of 8 “8-bit” Registers, each (Register) of which must be addressable – obviously this means selectable (i.e., Multiplex-able). Since `address` is an integer of 0, 1, 2, ..., or 7 in binary representation (you should now know why `address[3]` is used). Revise lecture slides about the RAM and how to implement “READ” and “WRITE”.

- You will need to implement a **Mux8Way8** gate (i.e., `Mux8Way8.hdl`) and submit it within your folder to allow the RAM to work. (.tst and .cmp files as well as a skeleton .hdl are provided in the zip file). Any other chips you may implement, to enable the implementation of the RAM, will need to be included in your submission (within the same folder of the task). **You should NOT use the Mux8Way16 in this task.**

4. Implement RAM16.

As RAM8 but this time defining 64 memory locations (i.e., 64 registers) index-able by `address`.

Hint: The overall design is like RAM8. You would now use an array of 8 RAM8 as your memory banks. Obviously, you use `address[6]`. This then needs to be partitioned into two-halves to select between the 8 RAM8s and then between the 8 Registers. You may use `address[0..2]` and `address[3..5]` but you must choose the right one for the RAM8s and Registers.

SUBMISSION: SUBMIT your answers, i.e., completed `Bit.hdl`, `Register.hdl`, `RAM8.hdl`, `RAM64.hdl` and the `Mux8Way8.hdl` files. (And all associated testing scripts & compare files).

Task E: (16-bit) RAM 512, RAM4K, Program Counter (PC)

1. Implement RAM512 and RAM4K (16-bit RAMs).

RAM512: defining 512 memory locations (i.e., 512 registers) index-able by `address`.

RAM4K: defining 4096 memory locations (i.e., 4096 registers) index-able by `address`.

{All are 16-bits, allowing you to use the Built-in chips}

Hint: By now you should have an idea about the hierarchical structure of larger memory banks. RAM512 would need an array of 8 RAM64 and so forth. You should now be able to resolve the addressing issue. If you are still stuck, read Section 3.5 in Chapter 3, page 54.

2. Implement a PC (a 16-bit Program Counter), which has *load*, *increment*, and *reset* controls. You can use any required Built-in components to implement your PC.

SUBMISSION: SUBMIT your answers, i.e., completed **RAM512.hdl**, **RAM4K.hdl**, and **PC.hdl** files. (And all associated testing scripts & compare files).

Task F: MyALU

1. Implement an ALU chip (MyALU), of your own design, that computes 19 functions (the 18 functions from Task C, in addition to the $X \text{ xor } Y$).
2. Table 1 shows the functions (Output), as per the required order and the 5 control bits (**C4**, **C3**, **C2**, **C1**, **C0**) with some samples of their values (you are required to complete them according to the Decimal values left column).
3. The .hdl skeleton is provided, but you will need to create your .tst and .cmp files for this chip.

Table 1: MyALU Functions (Output)

Decimal of Cs	C4	C3	C2	C1	C0	Output
0	0	0	0	0	0	0
1						1
2						-1
3	0	0	0	1	1	X
4						Y
5	0	0	1	0	1	X'
6						Y'
7						-X
8						-Y
9						X+1
10						Y+1
11						X-1
12						Y-1
13						X+Y
14						X-Y
15						Y-X
16						X AND Y
17						X OR Y
18						X xor Y

SUBMISSION: SUBMIT your answers, i.e., completed **MyALU.hdl** files. (And all associated testing scripts & compare files).

Final Submission Instructions

For ALL tasks, students are allowed to initially discuss (in small groups) and seek advice from peers. However, at the end, the work is still **individual** and should be **submitted separately** and **independently**.

For ALL tasks, each student should submit his/her own files (as listed below). **We reserve the right to ask all/some students to explain their submitted work at any time. Failure to explain it properly could affect YOUR mark.**

All students **MUST** submit on Moodle a **ZIP** archive file. **Any other archive file formats WILL result in a penalty of a 5% deduction of your overall mark.** Within your ZIP archive, it should contain all your individual worked files. Name that ZIP file as **CSF-CW2-XXXXXXXXX.zip**, where the "XXXXXXXXX" is the 8-digits of your student ID.

All the files should have the (**EXACT**) file names, and arranged under the (**EXACT**) subfolder name/structure as detailed on page 6:

	<p>SubFolder Name - Task A:</p> <p>Mux.hdl DMux.hdl (And all associated testing scripts & compare files)</p> <p>SubFolder Name - Task B:</p> <p>Mux4Way16.hdl Mux8Way16.hdl DMux4Way.hdl DMux8Way.hdl (And all associated testing scripts & compare files)</p> <p>SubFolder Name - Task C:</p> <p>n2tALU.hdl (And all associated testing scripts & compare files)</p> <p>SubFolder Name - Task D:</p> <p>Bit.hdl Register.hdl RAM8.hdl RAM64.hdl Mux8Way8.hdl Remember: include Mux8Way8.hdl, required for the RAM (And all associated testing scripts & compare files)</p> <p>SubFolder Name - Task E:</p> <p>RAM512.hdl RAM4K.hdl PC.hdl (And all associated testing scripts & compare files)</p> <p>SubFolder Name - Task F:</p> <p>MyALU.hdl (And all associated testing scripts & compare files)</p> <p>On Moodle, please click on the “Coursework 2” link within the “Coursework” section to perform your submission.</p>
Release Date	Friday, 28 th October 2022
Submission Date	Friday, 11 th November 2022, by 11:55pm
Late Policy (University of Nottingham default will apply, if blank)	Work submitted after the deadline will be subject to a penalty of 5 marks (the standard 5% absolute) for each late working day out of the total 100 marks.
Feedback Mechanism and Date	Marks and written individual feedback will be returned via Moodle within the week commencing 5 December 2022.
Assessment Criteria	<p><u>IMPORTANT NOTE:</u></p> <ol style="list-style-type: none"> You are advised to add relevant comments to the code (in the .hdl files) to indicate your understanding of the implementation. As the implementation become more and more complex, those comments become extremely useful for you (especially when we ask you). You are allowed to use Built-in chips, for chips/gates that are already done in the previous coursework, exercise and/or any auxiliary chips that you may need (except any of those that are required, to be developed within the same task). The exception could be in Task E, when implementing a larger RAM. In Task E, you are allowed (and encouraged for performance purposes) to use smaller RAM (including Built-in) as your building block.

3. Marking & checks will be done through scripts, which expects specific file name (case sensitive). Any deviation from that will result in error, therefore, **resulting in a loss of marks**.
4. All chips will be tested through automated testing scripts.
5. Missing any required file(s), using different formats, naming, ..., etc. will attract **penalty of 10% of your overall mark**.
6. Any submitted work that explicitly exhibit any cutting corners/plagiarism, ..., etc will result in the **entire coursework 2 being awarded 0%**.
7. **Chips that are unable to run will result in 0% being awarded for that task.**

Assessment Breakdown:

Task A (10%):

Mux – Correct chip structure & executes correctly:	5
DMux – Correct chip structure & executes correctly:	5

Task B (20%):

Mux4Way16 – Correct chip structure & executes correctly:	5
Mux8Way16 – Correct chip structure & executes correctly:	5
DMux4Way16 – Correct chip structure & executes correctly:	5
DMux8Way16 – Correct chip structure & executes correctly:	5

Task C (15%):

n2tALU – Correct chip structure & executes correctly:	15
--	-----------

Task D (25%):

Bit – Correct chip structure & executes correctly:	5
Register – Correct chip structure & executes correctly:	5
RAM8 – Correct chip structure & executes correctly:	5
RAM64 – Correct chip structure & executes correctly:	5
Mux8Way8 – Correct chip structure & executes correctly:	5

Task E (15%):

RAM512 – Correct chip structure & executes correctly:	5
RAM4K – Correct chip structure & executes correctly:	5
PC – Correct chip structure & executes correctly:	5

Task F (15%):

MyALU – Correct chip structure & executes correctly:	15
---	-----------

k