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

Assessment Name	Coursework 3	**Weight**	22.5%
Description and Deliverable(s)	The coursework (details below) focuses on **The Hack Computer** (*Bring everything together)* of this module. It is a **MINI GROUP WORK – 3 persons per group.** **Preliminaries** • This coursework has **THREE** parts, as detailed below. • In the previous coursework and lab sessions, we used *combinatorial* and *sequential* logic circuits to construct many of the various core logic circuits that form the basis of a CPU and Memory. • This coursework concludes all the effort by completing the construction of the HACK Computer, i.e., putting it all together and have the HACK Computer working & executing instructions. (i). **Firstly**, the *MEMORY* will be built according to the HACK architecture. (ii). **Secondly**, the *CPU* will combine the *ALU* and other circuits. But you'll need to develop the "*Control Unit*" that manages the data flow and execution of instructions. (iii). **Finally**, all are connected as one chip (the "*Computer*"). • You should go through the Lecture slides, Chapter 5, and Appendix A of the primary text *carefully*. • More lab reading resources are available at www.nan2tetris.org/05.php. In particular, "Chapter 5" and its relevant appendices. **Important Notes** 1. Note that for this coursework, **all tasks** are made as a mini group work. You are encouraged to divide the different tasks among yourselves, within the group. You should finish your tasks faster this way. However, you must maintain a healthy communication between yourselves to avoid conflict and delay in completing this coursework by the stipulated deadline. 2. Do not worry if you did not manage to complete any of the previous coursework or your implementation (from previous coursework) did not work correctly since nand2teris provides built-in implementations. We recommend you keep the files for this coursework away from the previous coursework (and you **MUST** maintain the provided folder structure). 3. All files for Parts 1, 2 and 3 will be under THREE folders. Please see and stick to the provided zip file and its folder structure and file names. **Files to Download** **CW3-Files.zip** provides all the skeleton .hdl and .asm (*as well as most of the related .tst and .cmp files*). 1. Download the zip file, from the "Coursework 3" area on Moodle. 2. Extract the files and fill in the required HDL and Assembly codes, as per the tasks in this assessment sheet (*both part 1 and part 2 in their respective folders*). 3. **MAINTAIN** the same folder structure, file name, and all test and compare files when you submit. You can add any additional files you may need to get your submissions working correctly.		

PART 1: Assembly Code

In this part of the coursework, you are required to provide two Assembly programs (for the HACK computer) to the following tasks:

a) An Assembly code that does **Multiplication**:

- Implement a multiplication Assembly code that multiplies **R1** (**RAM[1]**) and **R2** (**RAM[2]**) and stores the result in **R3** (**RAM[3]**).
  Hint: Obviously, there is no multiplication instruction/code to use (the underlying ALU can only add numbers). So, how do you perform a multiplication for two numbers A and B? Well, you can just add A for B – numbers of times. For example, if B is 3, then A x B is the same as A + A + A. Here, you would treat A as a stored constant while B is a variable that you keep track of.

- **If you make no progress**. Why don't you implement the "algorithm" in C-Language and use what you learn from the "Addition" example, demonstrated in detail in the lecture and lab, to come out with the corresponding machine code for this task.

b) An Assembly code that does **Factorial**:

- Implement an Assembly program to calculate the factorial of a given number, **n**, F(n). A factorial of a number is given by:

  F(n) = n*(n-1)*(n-2)*…*2*1

- The user should enter the value of the number **n** into **R0**, i.e., **RAM[0**].
- The factorial result **F(n)** should be saved in **RAM[1]**.

SUBMISSION: SUBMIT your answers, i.e., completed **Mult.asm** & **Factorial.asm** files. (And associated tst & cmp files).

PART 2: The HACK Computer

Task A: Memory

- **Implement the Memory Chip**.
  Hint: The specification for the memory chip is described in the lecture and Chapter 5. Note that you would have to use RAM16K, Screen and Keyboard "parts" (built-in, but you should refresh yourself with its interface specifications in Chapter 5, for the latter two).

- **If you make no progress.** You need to understand what makes a memory chip – there is this need for addressing/selecting "memory banks". How do you select 3 outputs (RAM16K, Screen, Keyboard)? (Mux4Way*? DMux4Way*?). You still need to interpret/decode a 15-bit address and pass them through to the right "memory banks" (RAM16K, Screen or Keyboard). The memory addresses are already given to you (e.g., lecture slides, and Chapter 5). The "logic" for interpretation/decoding would make more sense once you convert them to binary. You would note that only certain address bits are crucial for selection.

SUBMISSION: SUBMIT your answers, i.e., completed **Memory.hdl** file (And associated tst & cmp files).

Task B: CPU

- **Implement the CPU chip.**
  Hint: This may be the most challenging task – so, give it enough attention. The CPU implementation (framework) is given in the lecture and Chapter 5, Section

5.3.1. In general, the CPU as a complex logical gate would fetch and execute instructions in their corresponding A- and C-Instruction codes (16-bits long).

- **If you make no progress.** Look at the Chip diagram of CPU implementation in the lecture and in Chapter 5. Adopt a divide-and-conquer approach, that is, try to solve the problem by parts. Use this skeleton and compare with the Chip diagram:

```
// Instruction decode
// Use a combination of elementary logical gates to decode the instructions
// You should first decode between A and C-Instructions,
// then the computation and destination
:
And (a=cInst, b=instruction[4], out=destD);
:
:

// A register and input mux
Mux??(…);
?? (…);
ARegister (…);

// D register
DRegister(in=aluOut, load=destD, out=dReg);

// ALU and input mux
Mux16 (…);
ALU (…);

// PC with jump test
// Use a combination of elementary logical gates to implement the truth table for
// jump functions, given in lectures.
// For example, try to figure out why one implementation would make use of:
// Or (a=jle, b=jgt, out=jump);
:
:
PC (in=aReg, reset=reset, inc=true, load=jump, out[0..14]=pc);
```

Note: the above pseudo-code is for illustration purposes only. It may **NOT** be complete/accurate for the HDL similar. You only use it as a guide/hint.

SUBMISSION: SUBMIT your answers, i.e., completed **CPU.hdl** file. (And associated tst & cmp files). See **PART 3** for details of additional tasks.

Task C: Computer

- **Implement the Computer chip.**
  Hint: This is easy and as a bonus only 3-lines of code:

```
CPU (inM=??, instruction=??, reset=??, WriteM=??, outM=??, address=??, pc=??);
Memory (in=??, load=??, address=??, out=??);
ROM32K (address=??, out=??);
```

- The provided zip file contains a couple of test files (e.g**.,** *ComputerMax, ComputerAdd, ComputerRect*), to test your Computer chip. Make sure to utilise them and properly test your chip before submission.

SUBMISSION: SUBMIT your answers, i.e., completed **Computer.hdl** file. (And associated tst & cmp files).

PART 3: Circuitry Diagram and Justification

In **PART 2** – **task B,** you would have designed a circuitry diagram prior to implementing the CPU chip. As such, you are required to submit your full **circuit diagram** of your CPU implementation, along with a 300 words summary justifying

your circuitry design. Within folder PART 3 in the provided zip file contains a Word document (Justification.docx) for you to add your justification. Please make sure to fill your details (e.g., *name and studentID*) as well as your fellow group team members. Also, please indicate clearly which group are you from. **Note**: your justification should not be more than 300 words.

SUBMISSION: SUBMIT your answers, i.e., completed **Circuitry Diagram.pdf** and **Justification.pdf** files.

Final Submission Instructions

For ALL tasks, students are required to work in small groups.

Each group will have **ONE** submission **ONLY** (by *one nominated member of the group*).

The nominated member for each group will need to fill in the following Excel Spreadsheet: CSF-CW3-Group-Listings with all group members' name and their respective Student IDs prior to submission. **Please take note of the group number in column Group No.**

We reserve the right to ask all/some students to explain any/all of your submitted work at any time. Failure to explain it properly could affect YOUR mark.

Compress the whole folder structure/tree only (*see the provided zip file, for an example*). **Avoid** compressing each individual folder/file. Nested compression will prevent the marking process and could result in marking scripts failing. Should this occur to your submission, your entire coursework 3 mark **WILL** be awarded a 0%. So, please be careful!

The group submission must be submitted via Moodle as a **ZIP** archive file. **Any other archive file formats WILL result in a penalty of a 5% deduction of your group's overall mark.** Within your ZIP archive, it should contain all the requested files (same folder structure as per the downloaded files). Name that ZIP file as **CSF-CW3-XX.zip**, where the "XX" is your Group Number (*i.e.,* 01, 20, 54).

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

1_Assembly:

Factorial.asm
Factorial.cmp
Factorial.tst

Mult.asm
Mult.cmp
Mult.tst

(All associated tst & cmp files)

2_HACK:

Add.hack
Computer.hdl
ComputerAdd.cmp
ComputerAdd.tst
ComputerMax.cmp
ComputerMax.tst
ComputerRect.cmp
ComputerRect.tst
CPU.cmp
CPU.hdl
CPU.tst
Max.hack
Memory.cmp
Memory.hdl
Memory.tst

<table>
<tr><td></td><td>Rect.hack
(All associated tst & cmp files)

3_Circuit Diagram and Justification:
Circuit Diagram.PDF
Justification.PDF

On Moodle, please click on the “Coursework 3” link within the “Coursework” section to perform your group’s submission.</td></tr>
<tr><td>Release Date</td><td>Friday, 11th November 2022</td></tr>
<tr><td>Submission Date</td><td>Wednesday, 30th November 2022, by 11:55pm</td></tr>
<tr><td>Late Policy
(University of Nottingham default will apply, if blank)</td><td>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.</td></tr>
<tr><td>Feedback Mechanism and Date</td><td>Marks and written individual feedback will be returned via Moodle within the week commencing 19 December 2022.</td></tr>
<tr><td>Assessment Criteria</td><td>
<u>IMPORTANT NOTE:</u>

You are advised to <u>add relevant comments</u> 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 <u>allowed to use Built-in chips</u>, 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 in this coursework 3 itself).
Marking & checks will be done through scripts, which expects specific file name (case sensitive). Any deviation from that will results in error, therefore, resulting in a loss of marks.
All chips will be tested through automated testing scripts.
Missing any required file(s), using different formats, naming, …, etc. will attract penalty of 10% of your overall mark.
Any submitted work that explicitly exhibit any cutting corners/plagiarism, …, etc will result in the entire coursework 3 being awarded 0%.
Chips that are unable to run will result in 0% being awarded for that task.

<u>Assessment Breakdown:</u>

Part 1 – Assembly Code (15%):
<table>
<tr><td>Multiplication Program</td><td>5</td></tr>
<tr><td>Factorial Program</td><td>10</td></tr>
</table>
Part 2 – HACK Computer (35%):
<table>
<tr><td>Task A – Memory.hdl</td><td>5</td></tr>
<tr><td>Task B – CPU.hdl</td><td>15</td></tr>
<tr><td>Task C – Computer.hdl</td><td></td></tr>
<tr><td>Pass “Add” test</td><td>5</td></tr>
<tr><td>Pass “Max” test</td><td>5</td></tr>
<tr><td>Pass “Rect” test</td><td>5</td></tr>
</table>
Part 3 – Circuitry Diagram & Justification (50%):
<table>
<tr><td>Circuitry Design</td><td>20</td></tr>
<tr><td>Justification of Circuitry Design</td><td>30</td></tr>
</table>
</td></tr>
</table>