School of Computing: assessment brief

Module title	Computer Processors
Module code	XJCO1212
Assignment title	**August Resit**
Assignment type and description	In-course assessment. Requires design implementation and testing of code written in HDL and assembly language
Rationale	Provides an opportunity to write HDL and assembly code including understanding the implementation of branching and functions.
Word limit and guidance	This coursework should take less than 25 hours to complete.
Weighting	100%
Submission deadline	**3pm 9/8/24 UK time**
Submission method	Gradescope
Feedback provision	Feedback will be provided through Gradescope
Learning outcomes assessed	Describe the basic building blocks of a computer in detail and explain how they are composed to construct computing machinery. Apply appropriate tools to develop, simulate and test logic circuits (CAD). Explain how high level programming constructs, such as 'if' statements and 'for' loops, are implemented at a machine level
Module lead	Samson Fabiyi
Other Staff contact	Heng Lui

1. **Assignment guidance**

   There are two sections to this resit assessment. Section I requires implementation of HDL programs and Section II requires the implementation of assembly language.

2. **Assessment tasks**

SECTION I

Your task is to design and implement a circuit in hdl which takes two 2-bit numbers (A, B) and (C, D) as input and produces a 3-bit output (E, F, G).

The final circuit has 6 inputs in total (f_1, f_0, A, B, C, D) and 3 outputs (E, F, G).

The function of the circuit is determined by the two inputs f_1 and f_0.

The truth tables below define the operation of the circuit for each combination of f_1 and f_0.

A	B	C	D	F	G
0	0	0	0	1	1
0	0	0	1	1	0
0	0	1	0	0	1
0	0	1	1	0	0
0	1	0	0	1	1
0	1	0	1	1	0
0	1	1	0	0	1
0	1	1	1	0	0
1	0	0	0	1	1
1	0	0	1	1	0
1	0	1	0	0	1
1	0	1	1	0	0
1	1	0	0	1	1
1	1	0	1	1	0
1	1	1	0	0	1
1	1	1	1	0	0

Table 1: When $f_1, f_0 = (1, 1)$ FZero

A	B	C	D	F	G
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	0	0
0	0	1	1	0	0
0	1	0	0	0	0
0	1	0	1	0	1
0	1	1	0	0	0
0	1	1	1	0	1
1	0	0	0	0	0
1	0	0	1	0	0
1	0	1	0	1	0
1	0	1	1	1	0
1	1	0	0	0	0
1	1	0	1	0	1
1	1	1	0	1	0
1	1	1	1	1	1

Table 2: When $(f_1, f_0) = (1, 0)$ FOne

A	B	C	D	F	G
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	1	1
0	1	0	0	0	1
0	1	0	1	0	1
0	1	1	0	1	1
0	1	1	1	1	1
1	0	0	0	1	0
1	0	0	1	1	1
1	0	1	0	1	0
1	0	1	1	1	1
1	1	0	0	1	1
1	1	0	1	1	1
1	1	1	0	1	1
1	1	1	1	1	1

Table 3: When $(f_1, f_0) = (0, 1)$ FTwo

A	B	C	D	E	F	G
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	1	0
0	0	1	1	0	1	1
0	1	0	0	0	0	1
0	1	0	1	0	1	0
0	1	1	0	0	1	1
0	1	1	1	1	0	0
1	0	0	0	0	1	0
1	0	0	1	0	1	1
1	0	1	0	1	0	0
1	0	1	1	1	0	1
1	1	0	0	0	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	1
1	1	1	1	1	1	0

Table 4: When $(f_1, f_0) = (0, 0)$ FThree

(a) For each of the truth tables above, implement a logic circuit in HDL that will perform the function specified by the truth table. You should create one circuit for each truth table and test it produces the required output for that truth table. The circuit for each truth table should be named FZero, FOne, FTwo, FThree corresponding to the names of the truth tables given above and must have corresponding filenames FZero.hdl, FOne.hdl, FTwo.hdl, FThree.hdl.

Each circuit must have four inputs named A,B,C and D. Chips FZero, FOne and FTwo will have two outputs (F,G). Chip FThree will have three outputs named E,F and G.

You must only use the built-in AND, NAND, OR, NOR, NOT, Mux, XOR or DMux chips.

The test files provided (.tst and .cmp) can be used to test each output of a chip. For example FZero1.tst tests the F output of the chip FZero.hdl and FZero2.tst tests the G output of FZero.hdl.

[9 marks]

(b) Combine all four circuits into one circuit which takes all six inputs and three

outputs and test it to ensure it produces the correct output depending on the value of the inputs f_1 and f_0. Call the chip FALL. You can test this chip using FALL.tst but may wish to create further tests before submission. The value of output is undefined (can be either 0 or 1) unless $(f_1, f_0) = (0, 0)$

You must only use the built-in AND, NAND, OR, NOR, NOT, Mux, XOR or DMux chips.

[4 marks]

(c) Stretch Activity

When performing computational operations it is often useful to be able to execute a sequence of operations, each one using the output of the previous step as an input to the next step. For example to OR 3 values X OR Y OR Z you might first calculate X OR Y and then on the next step apply OR Z to the previous output (X OR Y).

For this task adapt the circuit FALL so that it can combine a sequence of operations defined by different values for f_1 and f_0 at each step, by enabling the outputs F_t and G_t of step t to be used (feedback) as the inputs for the next operation C_{t+1} and D_{t+1} for step $t + 1$. You should also add a further input (Load) to the chip which when Load = 1 will enable you to load new inputs to C_t and D_t and when set to 0 sets $C_{t+1} = F_t$ and $D_{t+1} = G_t$. The Load input will allow you to manually set the values of C and D at the start and during the sequence if required.

Call this chip FSEQ. You can test this chip using FSEQ.tst but may wish to create further tests before submission.

You must only use the built-in AND, NAND, OR, NOR, NOT, Mux, DMux, XOR or DFF chips.

[7 marks]

[Total for Section I 20 marks]

SECTION II

The Feistel cipher is a symmetric block cipher encryption framework which is the basis of many modern day encryption algorithms. In this coursework you will implement a Feistel cipher system as a software implementation in both a high level language and Hack Assembly.

In a Feistel cipher the plaintext, P, to be encrypted is split into two equal size parts L_0 and R_0 such that $P = L_0R_0$. A function F is applied to one half of the plaintext, combined with a key, and the result is XOR'd with the other half of the plaintext.

Feistel ciphers often employ multiple rounds of this scheme. In general the scheme works as follows, for all $i = 0, \ldots, n$,

$$L_{i+1} = R_i$$
$$R_{i+1} = L_i \oplus F(R_i, K_i)$$

To decrypt an encrypted message using this cipher we can apply the same procedure in reverse. For $i = n, n-1, \ldots, 0$,

$$R_i = L_{i+1}$$
$$L_i = R_{i+1} \oplus F(L_{i+1}, K_i)$$

For this coursework we are interested in the 16-bit Feistel cipher which uses 4 rounds. The function $F(A, B) = A \oplus \neg B$.

The keys are derived from a single 8-bit key K_0 such that,

$$K_0 = b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0$$
$$K_1 = b_6 b_5 b_4 b_3 b_2 b_1 b_0 b_7$$
$$K_2 = b_5 b_4 b_3 b_2 b_1 b_0 b_7 b_6$$
$$K_3 = b_4 b_3 b_2 b_1 b_0 b_7 b_6 b_5$$

(a) Write a program (XOR.asm) in HACK assembly that implements an XOR function between two 16-bit values stored in RAM[3] and RAM[4] and stores the result in RAM[5].

[6 marks]

(b) Write a program (Rotate.asm) in HACK assembly that implements an algorithm to rotate the bits of a 16-bit number left (Least Significant bit (LSb) to Most Significant bit (MSb)). The original number should be stored in RAM[3] the number of times to rotate the bits should be in RAM[4] and the result stored in RAM[5], i.e. 1010111100000000 rotated left 3 times would be 0111100000000101 where the MSb is used to replace the LSb.

[12 marks]

(c) Write a program (FeistelEncryption.asm) in HACK assembly, that implements the described Feistel encryption system. The initial key, K_0, will be stored in RAM[1], and the 16-bit plaintext will be stored in RAM[2]. The result of the encryption should be stored in RAM[0].

[12 marks]

[Total for Section II 30 marks]

3. **General guidance and study support**

   Tools required to simulate the hardware and CPU are provided on Minerva under Learning resources: Software.

   Please ensure the files you upload work with the test files provided and use the filenames provided in this sheet. **Do not alter the format of the lines of these test files in any way. The spacing in each line needs to be preserved** You are of course welcome to build your own test files in the same format or add to these files.

   Ensure the files you upload pass the submission tests provided on Gradescope. These are not necessarily the same tests as those that will be used to grade your submission.

4. **Assessment criteria and marking process**

   This coursework will be marked using Gradescope. Feedback will be provided through Gradescope and example solutions discussed in class.

   Marks are awarded for passing the automated tests on the submitted programs detailed below.

5. **Presentation and referencing**

   Submitted code should provide suitable comments where possible.

6. **Submission requirements**

   Links to submit your work can be found on Minerva under Assessment and feedback/Submit my work.

   For section I submit only your hdl files, uploaded individually. Ensure you use only the filenames provided in this specification sheet. **The names must match the specification exactly, including the use of upper and lower case characters** i.e. FZero.hdl is valid however, fzero.hdl or FZero.HDL are not valid.

   For section II submit only your asm files.

7. **Academic misconduct and plagiarism**

   Academic integrity means engaging in good academic practice. This involves essential academic skills, such as keeping track of where you find ideas and information and referencing these accurately in your work.

By submitting this assignment you are confirming that the work is a true expression of your own work and ideas and that you have given credit to others where their work has contributed to yours.

8. **Assessment/marking criteria grid**

Section I

No marks will be awarded for tests which fail or use of chips other than those listed.

- Part (a) There is one test to check the complete truth table for each output of the chips [9 marks].
- Part (b) There are four tests to check the complete truth table of the FALL chip [4 marks].
- Part (c) will be evaluated by testing it on three sequences of functions of various lengths [7 marks].

[Total for Section I 20 marks]

Section II

No marks will be awarded for tests which fail

- Part a) is graded using 3 tests, each worth 2 marks. [max 6 marks]
- Part b) is graded using 4 tests, each worth 3 marks. [max 12 marks]
- Part c) is graded using 4 tests, each worth 3 marks [max 12 marks]

[Total for Section II 30 marks]