UNIVERSITY of York

Department of Electronic Engineering

Assessments 2019/20

ELE00067M

Digital Design

This assessment (**Homework 1**) contributes **20%** of the assessment for this module.

Clearly indicate your **Exam Number** on every separate piece of work submitted.

Unless the assessment specifies a group submission, you should assume all submissions are individual and therefore should be your own work.

All assessment submissions are subject to the Department's policy on plagiarism and, wherever possible, will be checked by the Department using Turnitin software.

Submission is via VLE and is due by **12:00** on **25 November 2019 (Autumn Term, Week 9, Monday)**. Please try and submit early as any late submissions will be penalised.

Please remember that if this is your first year of study, you need to complete the mandatory Academic Integrity Tutorial http://www.york.ac.uk/integrity/

ELE00067M Digital Design - Homework Set 1

Try to answer all questions. This is an INDIVIDUAL effort. Access any material you deem useful (lecture notes, books, internet). You are allowed to discuss general approaches (for example, using the lecture slides as support) but not specific solutions, as this would be considered collusion.

IMPORTANT: This assessment should be submitted as a single PDF file through the VLE. Answer each question on a separate page. Write your exam number on every page. Failure to do so might result in your homework not being marked.

Question 1 *[24 marks]*

Consider the following simple program, which checks the parity of a word loaded from memory and then stores the parity value back to memory:

```
data ← DMEM[data_addr];            -- register indirect addr.
parity ← 0;                        -- initialize variables
i ← word_length;
mask ← 1;
while i ≠ 0 do                     -- for each bit in word
     temp ← data and mask;         -- check if it is one
     parity ← temp xor parity;     -- keep track of the parity
     data ← data >> 1;             -- go to next bit
     i ← i - 1;
end while;
DMEM[data_addr+2] <= parity;       -- base plus offset addr.
```

Assuming that you want to execute this program on **architecture B** (Lecture 6):

- Can you execute the program on this architecture? Obviously, the hardware is capable of it (the ALU is able to implement all the required operations), but the instruction set defined in lecture is not. What is the minimal set of additional instructions that need to be implemented to execute this program? *For each new instruction, define its operation and determine an opcode value and an instruction coding compatible with the existing instruction set. [4 marks]*
- Assume that the values `word_length` and `data_addr` are already stored in R1 and R7, respectively. Further, assume the following register mapping for the variables: R2=>data, R3=>parity, R4=>i, R5=>mask, R6=>temp. You can also assume that the program would continue with additional instructions beyond the ones shown. *Convert the program into assembler and then to machine language (in binary and hex notation), using the instruction set and coding of architecture B and the additional instructions you have defined. [10 marks]*
- *For every instruction within the program, define the values of all the necessary control signals for the execution of the program on architecture B. Identify "don't care" values with 'Φ'. [10 marks]*

Question 2 *[12 marks]*

Using the multi-cycle architecture C, define (in a table) the operations to be carried out and the values of all the control signals required for each cycle in the execution of the following instructions. Identify "don't care" values. [4 marks each]:

`shr R3, R1, x'b;` [shift the contents of R1 right by 11 bits and store the result in R3]
`loadi R5, x'af1f;` [load into R5 the contents of the memory at address AF1F]
`brneq R3, x'11A;` [if R3 is not equal to 0, then jump to PC+11A, else continue]

Use tables in the following format (leaving blank any unused step):

SHR R3,R1, 5	RA[2:0]	RB[2:0]	WA[2:0]	MA[15:0]	IMM[15:0]	OEN	S[1:4]	AL[2:0]	SH[5:0]	WEN
S1 - Fetch	ΦΦΦ	ΦΦΦ	ΦΦΦ	x'ΦΦΦΦ	x'ΦΦΦΦ	Φ	ΦΦΦΦ	ΦΦΦ	ΦΦΦΦΦΦ	Φ
S2 - RegRd	ΦΦΦ	ΦΦΦ	ΦΦΦ	x'ΦΦΦΦ	x'ΦΦΦΦ	Φ	ΦΦΦΦ	ΦΦΦ	ΦΦΦΦΦΦ	Φ
S3 - ALU	ΦΦΦ	ΦΦΦ	ΦΦΦ	x'ΦΦΦΦ	x'ΦΦΦΦ	Φ	ΦΦΦΦ	ΦΦΦ	ΦΦΦΦΦΦ	Φ
S4 - MemRW	ΦΦΦ	ΦΦΦ	ΦΦΦ	x'ΦΦΦΦ	x'ΦΦΦΦ	Φ	ΦΦΦΦ	ΦΦΦ	ΦΦΦΦΦΦ	Φ
S5 - RegWr	ΦΦΦ	ΦΦΦ	ΦΦΦ	x'ΦΦΦΦ	x'ΦΦΦΦ	Φ	ΦΦΦΦ	ΦΦΦ	ΦΦΦΦΦΦ	Φ

Question 3 *[14 marks]*

In your final project, you will eventually design a 16/32 bit microprocessor, that is, a processor with 32-bit instructions and 16-bit data. In this exercise, you are asked to define the encoding for the instruction set of a similar processor, using the criteria outlined in the lectures.
The processor specifications are:

- A dual-read, single-write bank of 32 registers of 16 bits each.
- For the computation of the (minimal) size of the offsets, you should assume that the offsets should be able to fully cover:
  - a word-addressable data address space of 512 16-bit words.
  - a word-addressable instruction address space of 256 instructions (i.e. an 8-bit PC).
- An ALU/shifter capable of implementing:
  - 1 identity operation (no operation on input A);
  - 4 logic operations (AND, OR, XOR, NOT);
  - 4 arithmetic operations (add, subtract, increment, decrement);
  - 4 (logical) shift by n bits operations (shift left, shift right, rotate left, rotate right);
  - all flags (7) required for the control instructions (no overflow).
- An instruction set containing the instructions specified below in Table 1.

Determine a coding for each of the instructions in the set. Assign OPCODE values (6 bits) and specify the position of every field within each kind of instruction. [14 marks]

Category	Instruction	Operation
No-operation	`nop`	None
Arithmetic	`add rt, ra, rb`	rt <= ra + rb
	`sub rt, ra, rb`	rt <= ra – rb
	`addi rt, ra, imm`	rt <= ra + immediate value
	`subi rt, ra, imm`	rt <= ra – immediate value
	`inc rt, ra`	rt <= ra + 1
	`dec rt, ra`	rt <= ra – 1
Logic	`not rt, ra`	rt <= NOT ra
	`and rt, ra, rb`	rt <= ra AND rb
	`or rt, ra, rb`	rt <= ra OR rb
	`xor rt, ra, rb`	rt <= ra XOR rb
	`andi rt, ra, imm`	rt <= ra AND immediate value
	`ori rt, ra, imm`	rt <= ra OR immediate value
	`xori rt, ra, imm`	rt <= ra XOR immediate value
	`shl rt, ra, n`	rt <= ra shifted left by n bits
	`shr rt, ra, n`	rt <= ra shifted right by n bits
	`rol rt, ra, n`	rt <= ra rotated left by n bits
	`ror rt, ra, n`	rt <= ra rotated right by n bits
Transfer	`move rt, ra`	rt <= ra
	`loadi rt, imm`	rt <= DMEM[imm] {direct addressing}
	`loadr rt, ra`	rt <= DMEM[ra] {register indirect addressing}
	`loado rt, ra, off`	rt <= DMEM[ra+off] {base plus offset addressing}
	`stori rb, imm`	DMEM[imm] <= rb {direct addressing}
	`storr rb, ra`	DMEM[ra] <= rb {register indirect addressing}
	`storo rb, ra, off`	DMEM[ra+off] <= rb {base plus offset addressing}
Control	`jmp off`	Jump to IMEM[PC+off]
	`brc ra, cond, off`	If condition is true, then jump to IMEM[PC+off], else continue Conditions: ra = 0 ; ra ≠ 0 ; ra = 1; ra < 0; ra > 0; ra ≤ 0; ra ≥ 0

Table 1

(note that NOP by convention should have opcode "000000" and is normally grouped with arithmetic instructions)

Keep in mind:

- Similar opcodes for similar operations.
- Similar fields should be placed in the same bits within the instruction.