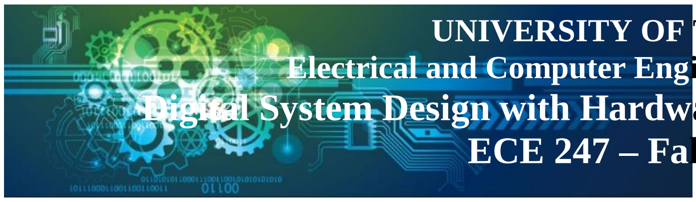

UNIVERSITY OF TEHRAN
Electrical and Computer Engineering Department

Digital System Design with Hardware Description Languages

ECE 247 – Fall 1396

Today's Date:

First Name: ....

Last Name: ....

Student Id Number: ....

Signature: ....

Grade:

Problem 1. ______	/34
Problem 2. ______	/33
Problem 3. ______	/32
Free. ____________	/01
Total: ______	/100

Hormuz Island, Near Qeshm, Persian Gulf, Iran; Day 1396

Regulations:

* DO NOT USE LAPTOPS
* EXTRA SHEETS WILL NOT BE ACCEPTED
* THIS IS AN OPEN BOOK OPEN NOTE EXAM
* YOU MUST SHOW COMPLETE WORK ON ALL PROBLEMS
* YOU HAVE EXACTLY 185 MINUTES FOR WORKING ON THIS TEST
* YOU ARE TRUSTED AND BY SIGNING HERE YOU ARE INDICATING THAT THIS EXAM IS YOUR OWN WORK ONLY: Signature:

1. **CPU, A Complete RTL Design.** In this problem you are to: **A)** Design a processor with the instruction set shown in Table 1. The processor has 6 registers R0, R1, R2, R3, R4, R5 (16-bit general purpose registers), of which R5 can also serve as a 4-bit integer Index register (4 LSB). Unsigned data stored in memory or registers are 16-bit fixed point with 4-bit integer part and 12-bit fractional part. There is a lookup table (LUT) that stores reciprocal of n, where n is 0<n<16. This table is addressed with the Index register (i.e., R5). There are separate Instruction (256 word) and Data (64 word) memories in this processor. Instructions of this processor are shown in Table 1. In this table, instruction opcodes are 4-bit, register specification are 3 bits and memory address is 6 bits. **A1)** Show datapath components, bussing structure, control signals, and data and address busses. **A2)** Show the controller state diagram including control signals as used in the datapath. Show the template of the VHDL description of the controller, using Huffman coding style. **B)** Suppose the coefficient numbers of Taylor expansion of exponential function are stored in the LUT. Write an assembly code (using the instruction set of the Processor in Part A) that calculates the exponential of a number that is stored in address 10100. Store the result at 10110.

Table 1: Instruction Set

Instruction format	**Description**	**Example**

`LD Address, Rdes`
`0000, Address, Rdes, xxx`
	`Rdes ← M[Address]`	`0000, 000100, 000, xxx`
		`R0 ← M[4]`

`ST Address, Rsrc`
`0001, Address, Rsrc, xxx`
	`M[Address] ← Rsrc`	`0001, 000100, 000, xxx`
		`M[4] ←R0`

`Add Rdes, Rsrc1, Rsrc2`
`0010, xxx, Rdes, Rsrc1, Rsrc2`
	`Rdes ←Rsrc1 + Rsrc2`	`0010, xxx, 100, 000, 001`
		`R4 ← R0 + R1`

`Sub Rdes, Rsrc1, Rsrc2`
`0011, xxx, Rdes, Rsrc1, Rsrc2`
	`Rdes ←Rsrc1 - Rsrc2`	`0011, xxx, 100, 000, 001`
		`R4 ← R0 - R1`

`*Mult Rdes, Rsrc1, Rsrc2`
`0100, xxx, Rdes, Rsrc1, Rsrc2`
	`Rdes ←Rsrc1[11:0] * Rsrc2 [11:0]`	`0100, xxx, 100, 000, 001`
		`R4 ← R0 * R1`

`Inc Rsrc`
`0101, xxxxxxxxx, Rsrc`
	`Rsrc ++`	`0101, xxxxxxxxx, 101`
		`R5 ←R5[15:12] +1 (only for R5)`
		`0101, xxxxxxxxx, 100`
		`R4← R4 +1`

`Dec Rsrc`
`0110, xxxxxxxxx, Rsrc`
	`Rsrc --`	`0110, xxxxxxxxx, 101`
		`R5 ←R5[15:12] -1 (only for R5)`
		`0110, xxxxxxxxx, 100`
		`R4 ←R4 -1`

`BL Address, Rsrc1, Rsrc2`
`0111, Address, Rsrc1, Rsrc2`
	`If Rsrc1 < Rsrc2 then`
	`PC ← 00&&Address`
		`0111, 010101, 000, 001`
		`If R0<R1 then PC ← M[21]`

`BE Address, Rsrc1, Rsrc2`
`1000, Address, Rsrc1, Rsrc2`
	`If Rsrc1 == Rsrc2 then`
	`PC ← 00&&Address`
		`1000, 010101, 000, 001`
		`If R0 == R1 then PC ← M[21]`

`BG Address, Rsrc1, Rsrc2`
`1001, Address, Rsrc1, Rsrc2`
	`If Rsrc1 > Rsrc2 then`
	`PC ← 00&&Address`
		`1001, 010101, 000, 001`
		`If R0>R1 then PC ← M[21]`

`BZ Address, Rsrc`
`1010, Address, xxx, Rsrc`
	`If Rsrc == 0 then`
	`PC ← 00&&Address`
		`1010, 010101, xxx, 000`
		`If R0 == 0 then PC← M[21]`

`REC Rdes`
`1011, xxxxxxxxx, Rdex`
	`Rdes ← LUT[R5]`	`1011, xxxxxxxxx, 011`
		`R3 ← LUT[R5]`

`Hlt`
`1100, 1101, 1110, 1111`
	`Halt the execution of program`
	`PC← 0`	

* the most 12- significant bits of multiplication results are stored in the Rdes. The four remaining bits are replaced with 0000.

2. **SystemC Channels.** Write a SystemC channel with interfaces for 8- and 32-bit put and get (put8, put32, get8, and get32). **Putting:** Data will be ready for getting only after complete 32 bits of data have been put in the channel. This can be achieved by four *put8*s or one *put32*. After a channel has collected 32 bits of data, no more data can be put in the channel until all four bytes are received, and the channel is empty. Putting into the channel is blocking. **Getting**: Data can be collected only when the channel has received 32 bits of data. Getting from the channel can be done by four *get8*s or one *get32*. Getting from the channel is blocking. **A)** Show the necessary interfaces. **B)** Implement the interfaces by developing a SystemC channel. **C)** Show how an initiator and a target can be connected using the channel of Part B.

3. **RTL Design, Appeciating Channels.** RT level design of an interface circuit is to be done. The initiator, target and the interface you are designing are driven by the same clock signal. The initiator bus is an 8-bit bus and that of the target is a 32-bit unidirectional bus. The initiator uses its *dataBus* to place data in the interface. A valid data on this bus is accompanied by a positive pulse on *dataValid* that is issued by the initiator. When four bytes of data have been collected by the interface, the data becomes available for the target to receive. The target has a choice of getting a 32-bit data by issuing *getData32* and getting the data on *targetBus*, or get an 8-bit data by issuing *getData8*. In the former case, all 32-bits will be used, and in the latter case, the received 8-bit data will be placed on eight right-most bits of *targetBus*. **A)** Show the datapath of this interface circuit. B) Show state machine for the controller of the interface circuit. **C)** Write SystemC description of the datapath circuit including the interconnections and busses. **D)** Write SystemC description of the interface circuit. **E)** Show datapath and control unit wiring forming the complete interface circuit.