NEW YORK UNIVERSITY

EE4313 Final Exam – Fall 2015
Due Date: See NYU Classes announcement

Final Exam:

8-Bit NYU Processor Design

An instruction set of 16 instructions (15 from us, and one from you)
Eight 8-bit registers
Capable of executing simple programs

NEW YORK UNIVERSITY ABU DHABI

Final Exam: Processor Design

Introduction

In this lab – your final exam, you will design an 8-bit processor, the NYU processor, which is capable of executing programs. This processor supports an instruction set of 16 instructions, 15 of which have already been specified for you. It also has eight 8-bit registers, R0 through R7.

Terminology Q&A

1. What is a processor?

A processor is a hardware component that is capable of executing programs.

2. What is a program?

A program is a sequence of instructions to accomplish a certain computational task. Example: A program that can take the average of 10 input numbers.

3. What is an instruction?

An instruction is an operation to be performed on a number of operands, with the result to be written into the destination also defined by the instruction.

Example: ADD R3, R1, R2 is an instruction that adds the content of two registers R1 and R2 (operands: R1 and R2), and writes the result into the destination register R3.

Example: ADDI R2, R5, 1 is an instruction that adds the content of the register R5 with 1 (operands: R5 and 1), and writes the result into the destination register R2.

Example: BNE R1, R2, -4 is an instruction that jumps to four instructions earlier only if the content of register R1 is not equal to the content of register R2 (operands: R1 and R2). BNE means Branch if Not Equal to.

4. How does the processor execute a program?

The binary code of the instructions in the program is loaded into the instruction memory, where the instructions are fetched (read and decoded) one at a time sequentially, with the exception of branch and jump instructions, which may change the sequence of execution. A register, Program Counter, keeps track of the address of the next instruction to execute from the instruction memory. Every time an instruction is executed, the content of the Program Counter must be updated to point to the next instruction to execute.

5. What is binary code?

The binary code of an instruction is the binary representation of the instruction. In our processor, every instruction has a corresponding 16-bit binary code. These 16 bits properly encode the type of the operation, the operands and the destination of the instruction.

Example: “ADDI R1, R0, 7” instruction has a 16-bit binary code of “0001001000000111”, which specifies that this is an addition operation with two operands R0 and 7 and a destination register of R1.

6. What happens when an instruction is executed by the processor?

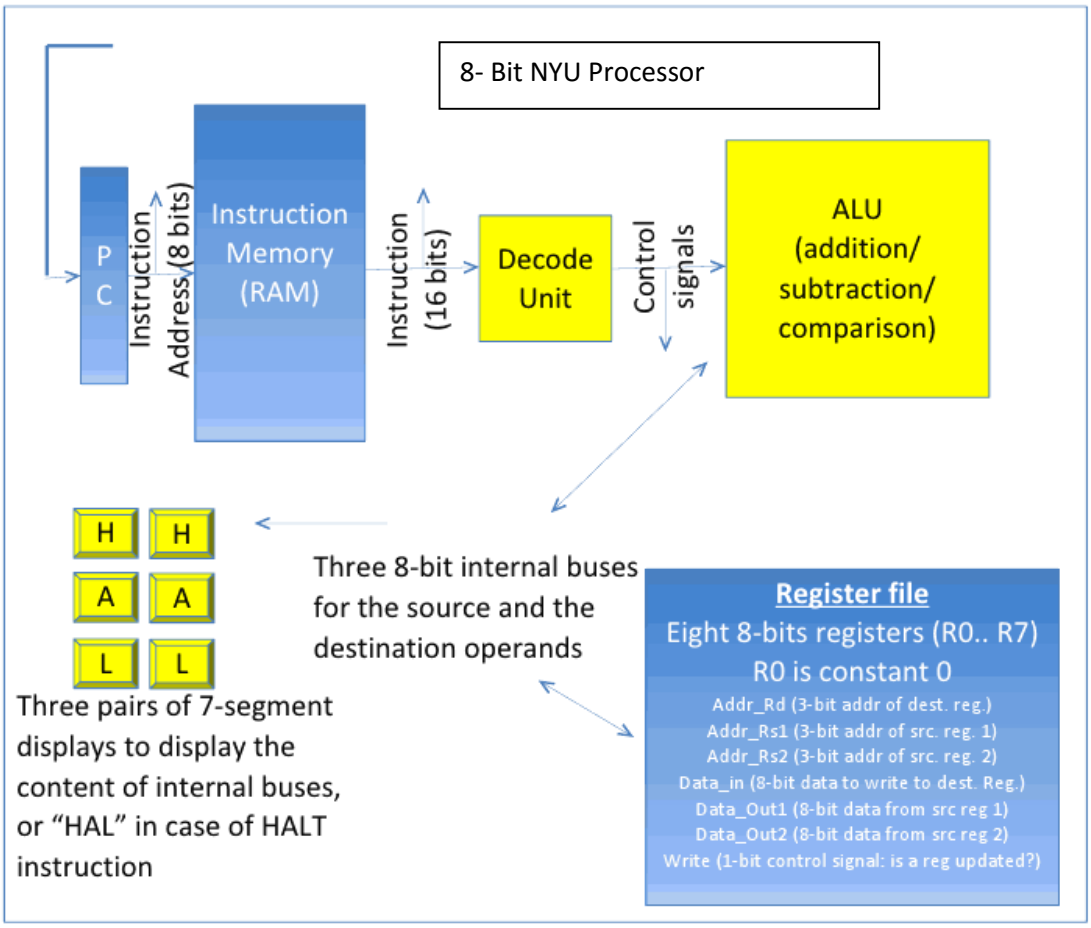
The operation specified by the instruction is carried out. The following are done concurrently (1) Instruction Fetch: The content of the instruction is read from the instruction memory and decoded. (2) PC Update: The content of the Program Counter is updated to point to the next instruction in the instruction memory.

(3) Execute: For arithmetic and logic operations (add, subtract, and, or, shift), the operation performed and the content of one of the registers is updated with the result. For other instructions (branch, jump and halt), the content of the Program Counter may possibly be updated. There are also other types of instructions - we’ll ignore them for now.

7. How does the execution of a program begin and end?

The binary code of the entire program is loaded into the instruction memory, with the first instruction

written into the first two bytes. The program counter is reset to 0 to point to the first instruction. With an active clock, the processor starts executing one instruction in every cycle. The last instruction is always the Halt instruction, which, upon execution, terminates the program by preventing the update of the Program Counter, after which the processor hangs.



Processor Components

• **Program counter (PC) register:** This is an eight -bit register that contains the address of the next instruction to be executed by the processor. Every time an instruction is executed, this register should be incremented by 1, so it points to the subsequent instruction. This register can be reset to all 0's, pointing to the first entry in the instruction RAM. This component will be provided to you.

• **Instruction Memory (RAM):** This 256 -entry block contains the instructions to be executed. The instruction whose address is specified by the PC register is executed within one clock cycle, during which the PC is incremented by 1 also - in the next clock cycle, the next instruction in the RAM is executed as a result. You will be able to load your code (sequence of instructions) to this block via GUI to be able to run your instructions by executing them one at a time. Obviously, the 8 -bit output of PC will be connected to the address inputs of this block, and its 16 -bit output will provide the content of the instruction. This component will be provided to you.

• **Decode Unit:** This block takes as input some or all of the 16 bits of the instruction, and computes the proper control signals to be utilized for other blocks. These signals are generated based on the type and the content of the instruction being executed. You will implement this block, which can be considered as the "brain" of the processor, as it orchestrates the flow of operations.

• **ALU:** This block can be considered as the "muscles" of the processor. It does all the hard work, such as

addition, subtraction, comparison, etc. It uses the control signals generated by the Decode Unit, as well as the data from the registers or from the instruction directly. It computes data that can be written into one of the registers (including PC). You will implement this block by referring to the instruction set.

• **Register File:** This block contains eight 8 -bit registers. The content of two of these registers, whose addresses need to be provided from the two 3 -bit input ports of the block, Addr_Rs1 and Addr_Rs2, can be placed on the two 8 -bit output ports of the block, data_out1 and data_out2. The content of one register, whose address is provided from the 3 -bit input port of the block, Addr_Rd, can be updated with the 8 -bit data provided from the Data_in port of the block, only if the write input of the block is active -1. Both read and write operations can be conducted (possibly on the same register) within the same clock cycle. This component will be provided to you.

• **Internal buses:** You will implement and use three 8 -bit internal buses to transport the two operands and the result of the instructions in between the register file and the ALU.

• **7(segment displays:** You will implement and use 7 -segment displays (along with their look -up tables) to display the content of the internal buses continuously. You have previously used these blocks in one of your earlier labs. These blocks will show "HALT", whenever the "HALT" instruction is executed.

Instructions

Every instruction has 16 bits that define the type of the instruction as well as the operands and the destination of the result. These instructions are stored in the instruction memory (RAM), from where they are fetched one at a time, and are executed through the ALU. The first instruction in the RAM, for instance, is stored in location 00000000. PC register originally consists of 00000000 to point to this instruction.
The instruction content is:

Opcode (4 bits)	Rd (3 bits)	Rs1 (3 bits)	Rs2 (3 bits)	Tail (3 bits)

Where Opcode determines the operation to be performed, Rd denotes the address of the destination register in which the result of the operation will be written, and Rs1 and Rs2 denote the addresses of the two registers that contain the two operands. The Tail field is used by some of the instructions.
The instruction set supported by the NYU processor is defined below. All operations are performed assuming 2's complement notation for the operands and the result, unless otherwise specified:

Opcode	Mnemonic	Description	Operation
0000	ADD	Add Registers	Rd= Rs1+Rs2
0001	ADDI	Add Immediate	Rd= Rs1+ Offset (6 bits from the Rs2 and tail fields, sign -extended into 8 bits to create the offset)
0010	SUB	Subtract Registers	Rd= Rs1 -Rs2
0011	SUBI	Subtract Immediate	Rd= Rs1 - Offset (6 bits from the Rs2 and tail fields, sign -extended into 8 bits to create the offset)
0100	AND	Register bitwise And	Rd= R1 & R2 (two 8 -bit numbers are bitwise ANDed)
0101	ANDI	And Immediate	Rd= R1 & Offset (6 bits from the Rs2 and tail fields, padded with 00 in the most significant bits, create the 8 -bit unsigned offset for the bitwise AND operation)
0110	OR	Register bitwise OR	Rd= R1|R2 (two 8 -bit numbers are bitwise ORed)
0111	ORI	OR Immediate	Rd= R1|Offset (6 bits from the Rs2 and tail fields, padded with 00 in the most significant bits, create the 8 -bit unsigned offset for the bitwise OR operation)

1000	SHL	Shift Left by offset bits	Rd = Rs1 << offset (3 -bit unsigned number defined by the tail field denotes the number of positions Rs1 is shifted to the left - the rightmost positions are filled with 0s and the leftmost positions are lost in the process)
1001	SHR	Shift Right by offset bits	Rd = Rs1 >> offset (3 -bit unsigned number defined by the tail field denotes the number of positions Rs1 is shifted to the right - the leftmost positions are filled with 0s and the rightmost positions are lost in the process)
1010	HAL	Halt	PC = PC (disabling the update of PC, and thus creating an infinite loop) - display "HAL"
1011	Unused		
1100	JMP	Jump	PC = PC + offset (6 bits from the Rs2 and tail fields, sign - extended into 8 bits to create the offset)
1101	BLT	Branch if less than	If (Rs1 < Rs2) then PC = PC + offset (6 bits from the Rd and tail fields, sign -extended into 8 bits to create the offset)
1110	BE	Branch if equal	If (Rs1 = =Rs2) then PC = PC + offset (6 bits from the Rd and tail fields, sign -extended into 8 bits to create the offset)
1111	BNE	Branch if not equal	If (Rs1= Rs2) then PC = PC + offset (6 bits from the Rd and tail fields, sign -extended into 8 bits to create the offset)

The jump and the branch instructions can be used to implement "loops". This is accomplished by updating PC by the instruction - unless a jump/branch instruction updates PC, it is always incremented by 1 as a part of executing any instruction.

As the register file has eight registers, 3 bits are used to specify each register in the instruction. Rd, Rs1 and Rs2 fields therefore consist of 3 bits each. The table below defines the straightforward encoding of registers:

Rd/Rs1/Rs2	Register Specified
000	R0=00000000
001	R1
010	R2
011	R3
100	R4
101	R5
110	R6
111	R7

R0 contains constant -0, and instructions are not allowed to write into this special register.

To Do List

- Implement the NYU processor by designing the missing blocks, and making the proper connections.
- Finish up your design
- Do a performance (what is the max speed of your processor) and area (number of gates you used from each type) analysis. Please explain your analysis well.
- Test your design by executing the following program (load the binary code to the memory, reset your PC, and kick it off) that adds 7 and 8:

  ADDI R1, R0, 7 // R1 = 7
  ADDI R2, R0, 8 // R2 = 8
  ADD R3, R1, R2 // R3 = R1 + R2
  HAL // HALT
  The 8 -byte binary code for the instruction sequence above is: 00010010 00000111

00010100 00001000
00000110 01010000
0100000 00000000

- Test your design by executing the following binary code. Describe what this code does by first figuring out the corresponding program, and then by explaining it in plain English: 00010010 00000101
00010100 00000000
00010100 10000010
00110010 01000001
11111110 01000110
10100000 00000000
- Describe a task/procedure (get as creative as you can) in words, write the program for it, turn it into binary code, and load it on your processor's instruction memory to execute the code. Check the results and show that the program does what it is supposed to do.
- Opcode 1011 is left unused intentionally. This is an opportunity for you to extend this processor. What instruction would you like to add to this processor? Add an instruction that would be quite handy to have. You may want to justify this by re-writing the same program above by using your new instruction. For this, you will probably end up changing your original design. So:
  - Create another Project.
  - Do a performance (what is the max speed of your processor) and area (number of gates you used from each type) analysis again and compare against your original design. Is this new instruction worth having? Please explain pros and cons.

  • Prepare the following by explaining all the above in addition to elaborating on how you carried out the work n a:
  - Report
  - Presentation (10 mins + 2 mins Q&A per group)

Deliverables

- Project Files (including vhdl files in zip folder): One for your original 15-instruction processor design, and one for the extended 16-instruction processor design.
- Report & presentation files: You can submit the report in .doc, or .pdf format, and the presentation in .pdf, or .ppt format.

IMPORTANT NOTE ON FINAL EXAM PHASES:

PHASE 1: Deadline NOVEMBER 30:

- Complete designing ALU.

PHASE 2: Deadline December 7:

- Complete designing Decoder Unit.
- Complete the processor design.
- Run the sample program.

PHASE 3: Deadline December 14:

- Complete the processor design with additional instruction.
- Run your own program on processor.

PHASE 4: Deadline December 21:

- Run the program given by us on your processor. (Program will be given on Dec -18th)

The final exam consists of multiple phases and deadlines.
KEEP CHECKING NYU CLASSES ANNOUNCEMENT FOR INFORMATION ON THE PHASES AND DEADLINE.

REMEMBER: Your codes will be checked for plagiarism. DO NOT COPY.