NYU-6463 Processor Design (Project Description)

Groups of 4. Final Due Date: December 16. 30 Points

For the final project, you will implement a 32-bit processor in VHDL, called NYU-6463 Processor, which is capable of executing programs. The processor supports the instruction set specified in the next section.

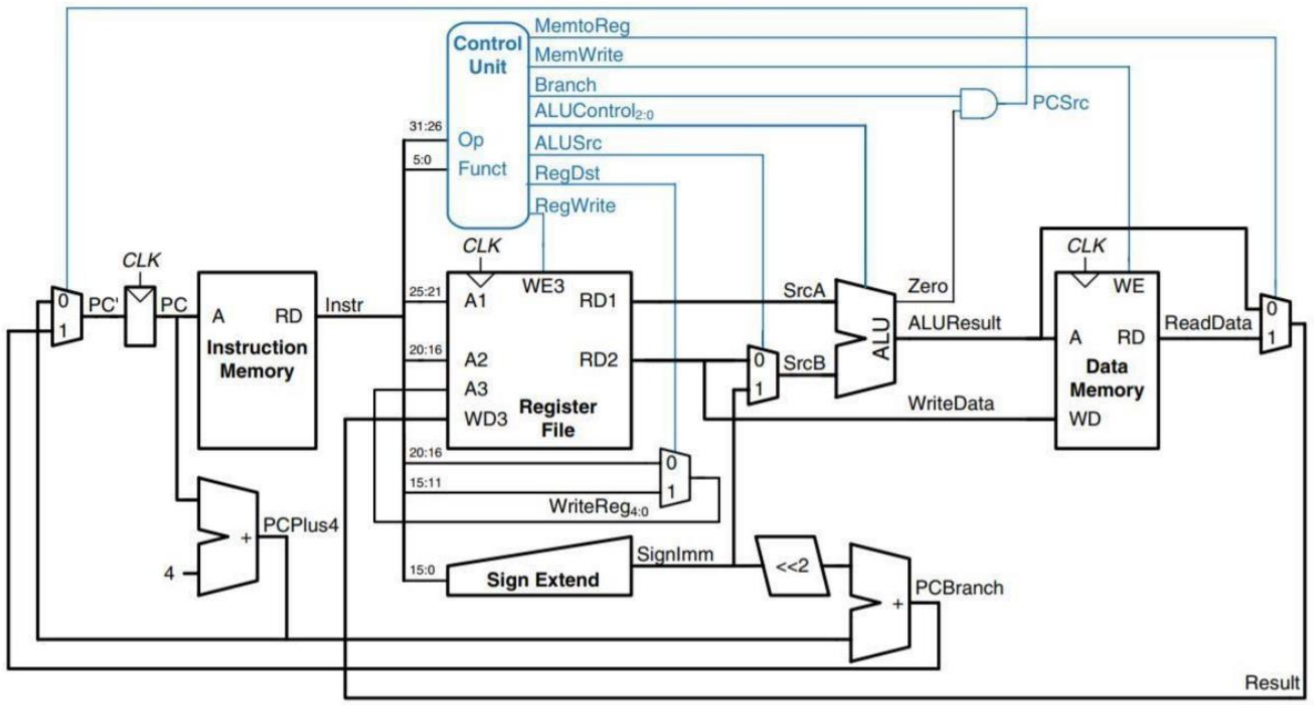
1. Design Specification

Every instruction has 32 bits that define the type of instruction as well as the operands and the destination of the result. The NYU-6463 Processor has three instruction types: (a) R-Type for arithmetic instructions, (b) I-Type for immediate value operations, load and store instructions, and (c) J-Type for jump instructions. The instruction formats and the instruction set are detailed in the **Supplementary Material**.

2. Processor Components

The processor comprises the following components:

- **Program counter (PC) register:** This is a 32 -bit register that contains the address of the next instruction to be executed by the processor.
- **Decode Unit:** This block takes as input some or all of the 32 bits of the instruction, and computes the proper control signals that are required for correctly coordinating the other blocks in your design. These signals are generated based on the type and the content of the instruction being executed.
- **Register File:** This block contains 32 32-bit registers. The register file supports two independent register reads and one register write in one clock cycle. 5 bits are used to address the register file. R0 value is always 0. It is read only register.
- **ALU:** This block performs operations 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.
- **Instruction and Data Memory:** The instruction memory is initialized to contain the program to be executed. Instruction memory width is 8-bits. The data memory stores the data and is accessed using `LW` (load word) and `SW` (store word) instructions. Data memory width is 16-bits. The address width for the data and instruction memory will be determined by the amount of memory available in your FPGA. Use up to as much memory as your FPGA allows.



NYU-6463 Processor (NB: control/status signals might be different)

3. Processor Operation

The NYU-6463 Processor performs the tasks of instruction fetch, instruction decode, execution, memory access and write-back all in one clock cycle.

- First, the PC value is used as an address to index the instruction memory which supplies a 32-bit value of the next instruction to be executed.
- This instruction is then divided into the different fields shown in Table 1. The instructions' opcode field bits [31-26] are sent to the decode unit to determine the type of instruction to execute.
- The type of instruction then determines which control signals are to be asserted and what function the ALU is to perform, therefore, decoding the instruction.
- The instruction register address fields Rs bits [25 - 21], Rt bits [20 - 16], and Rd bits [15-11] are used to address the register file. For rot_amt bits refer to the instruction format for the group that you have been allocated to.

The opcode provides the addresses used for the register file. Depending upon the instruction, the register values are read or written. These data values can then be operated on by the ALU. The operation is determined by the control unit to either compute a memory address (e.g. load or store), perform an arithmetic operation (e.g. and, or, xor and add, xor and sub), or compare (e.g. branch). If the instruction decoded is arithmetic, the ALU result must be written to a register. If the instruction decoded is a load or a store, the ALU result is then used to address the data memory. The final step writes the ALU result or memory value back to the register file.

4. Task Overview

- Implement the NYU-6463 Processor as per the specification. We encourage you to write your own programs and check your design for different cases. NOTE: You need to implement only the instructions described in Table 2. You cannot add additional instructions.

- Do a performance (max speed of your processor) and area (number of gates you used from each type) analysis of your design. Explain your analysis comprehensively. You may be expected to identify and explain the critical path of your design.

- Implement your design on FPGA. You should be able to show the result (register values R0-R31) of the execution of the program after every cycle. You can decide what interfaces to use to display your values (for example, you might use the LEDs/7-segment displays and switches, or UART)

    - Both single instruction stepping and complete program execution should be supported.

    - Your overall design should support changing the program while your processor is running on the FPGA.

- Write a program to implement the RC5 block cipher as well as round key generation using the instructions in Table 1. You should read the key and plaintext from the memory and write the ciphertext back to memory. Run your program on your designed processor and show that it works properly.

- Describe how many cycles are required to complete RC5 encryption and decryption on NYU-6463 Processor.

5. Deliverables:

1. Put your processor source code and assembly code in a zipped folder.
2. Your report in PDF format, including:
    a. design block diagram
    b. simulation screenshots
    c. performance and area analysis of design
    d. description of RC5 implementation in assembly
    e. description of processor interfaces (how do you provide inputs and display results)
    f. details about how you verified your overall design
3. Demo video of the processor executing RC5 algorithms on FPGA. (5 minutes)
4. An explanation of how one could optimize and improve your design in the future.
5. Poster presentation

6. Milestones:

Task	**Due Date**
T1 Develop all individual processor components and corresponding testbenches/simulation scripts	Nov 11
T2 Interconnect all individual processor components into the complete processor and develop corresponding testbench/simulation	Nov 18
T3 Develop the I/O infrastructure required to display the register contents, transfer new programs, and prototype your design on FPGA	Nov 30
T4 Complete all remaining documentation tasks include reports, a demo, a 5 minute video, and a poster presentation	Dec 16

November:

MON	TUE	WED	THU	FRI	SAT	SUN
				1	2	3
4	5	6	7	8	9	10
11 **T1 due**	12	13	14	15	16	17
18 **T2 due**	19	20	21	22	23	24
25	26	27	28	29	30	

December:

MON	TUE	WED	THU	FRI	SAT	SUN
						1
2 **T3 due**	3	4	5	6	7	8
9	10	11	12	13	14	15
16 **T4 Due**	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

Supplementary Material

Instruction format (R-type) for Group A:

Bit 31.. *..0*

Opcode (6-bits)	Rs (5-bits)	Rt (5-bits)	Rd (5-bits)	Rot_amt (4-bit)	Unused (1-bits)	Funct (6-bits)

(a) R-type Instruction Formats

Instruction format (I-type) for All groups:

Opcode (6-bits)	Rs (5-bits)	Rt (5-bits)	Address/Immediate (16-bits)

(b) I-type Instruction Format

Instruction format (J-type) for All groups:

Opcode (6-bits)	Address (26-bits)

(c) J-type Instruction Format

NYU-6463 Processor instruction types.

Table 1. NYU-6463 Processor instruction fields

Field	**Description**
Opcode	6-bit primary operation code
Rd	5-bit specifier for the destination register
Rs	5-bit specifier for the source register
Rt	5-bit specifier for the target (source/destination) register
Address/Immediate	16-bit signed immediate used for logical operands, arithmetic signed operands, load/store address byte offsets, and PC-relative branch signed instruction displacement
Address	26-bit index shifted left two bits to supply the low-order 28 bits of the jump target address
rot_amt	Rotate amount as specified by the instruction format (3 bits)
Funct	6-bit function field used to specify functions within the primary opcode

The instruction set supported by the NYU-6463 Processor is defined below. All operations are performed assuming 2’s complement notation for the operands and result.

Table 2. NYU-6463 Processor Instruction Set

Mnemonic	**Description**	**Type**	**Opcode (Hex)**	**Func (Hex)**	**Operation**
AND	Register bitwise And	R	00	12	Rd = Rs and Rt
ANDI	And Immediate	I	03		Rt = Rs and SignExt(Imm)
OR	Register bitwise OR	R	00	13	Rd = Rs or Rt
NOR	Register bitwise NOR	R	00	14	Rd = !(Rs or Rt)
ORI	OR Immediate	I	04		Rt = Rs or SignExt(Imm)
LW	Load Word	I	07		Rt ← Mem[SignExt(Imm) + Rs]
SW	Store Word	I	08		Mem[SignExt(Imm) + Rs] ← Rt
BLT	Branch if less than	I	09		If (Rs < Rt) then PC = PC + 4 + (Imm & “00”)
BEQ	Branch if equal	I	0A		If (Rs = = Rt) then PC = PC + 4 + (Imm & “00”)
BNE	Branch if not equal	I	0B		If (Rs /= Rt) then PC = PC + 4 + (Imm & “00”)
JMP	Jump	J	0C		PC = (PC + 4) + ([SignExt(address)](29 downto 0) & "00")
XRLR	Xor and Left rotate	R	00	10	Rd = (Rs XOR Rt) <<< rot_amt
RRXR	Right rotate and Xor	R	00	11	Rd = (Rs >>> rot_amt) Xor Rt
LRAD	Left rotate and Add	R	00	15	Rd = (Rs <<< rot_amt) + Rt
SBRR	Sub and Right rotate	R	00	16	Rd = (Rs-Rt) >>> rot_amt
~~XLA~~	~~Xor, Left rotate, Add~~	~~R~~	~~00~~	~~17~~	~~Rd = ((Rs XOR Rt) <<< rot_amt) + R30~~
~~SRX~~	~~Sub, Right rotate, Xor~~	~~R~~	~~00~~	~~18~~	~~Rd = ((Rs-R30) >>> rot_amt) Xor Rt~~
HAL	Halt	J	3F		*Processor stalls indefinitely*

Reminder: & in VHDL is concatenation

Please NOTE: Jump instruction is DIFFERENT from the MIPS standard