NYU-6463 Processor Design

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.

Design Specification

Instructions

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 content for these three types are shown in Figure 1. A description of each of the fields used in the three different instruction types is provided in Table 1.

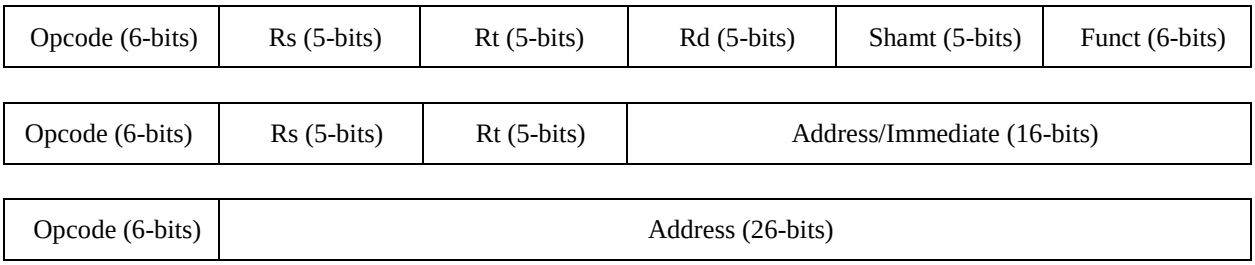

Figure 1. NYU-6463 Processor instruction types. (Please see Table 1 for description)

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
Shamt	5-bit shift amount
Funct	6-bit function field used to specify functions within the primary opcode

Table 1. NYU-6463 Processor instruction fields

The instruction set supported by the NYU-6463 Processor is defined in Table 2. All operations are performed assuming 2's complement notation for the operands and the result, unless otherwise specified.

Mnemonic	Description	Type	Opcode (Hex)	Func (Hex)	Operation
ADD	Add Registers	R	00	10	Rd = Rs + Rt
ADDI	Add Immediate	I	01		Rt = Rs + SignExt(Imm)
SUB	Subtract Registers	R	00	11	Rd = Rs - Rt
SUBI	Subtract Immediate	I	02		Rt = Rs - SignExt(Imm)
AND	Register bitwise And	R	00	12	Rd = Rs & Rt
ANDI	And Immediate	I	03		Rt = Rs & SignExt(Imm)
OR	Register bitwise OR	R	00	13	Rd = Rs | Rt
NOR	Register bitwise NOR	R	00	14	Rd = !(Rs | Rt)
ORI	OR Immediate	I	04		Rt = Rs | SignExt(Imm)
SHL	Shift Left by immediate bits	I	05		Rt = Rs << Imm
SHR	Shift Right by immediate bits	I	06		Rt = Rs >> 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 + 1 + Imm
BEQ	Branch if equal	I	0A		If (Rs = = Rt) then PC = PC + 1 + Imm
BNE	Branch if not equal	I	0B		If (Rs != Rt) then PC = PC + 1 + Imm
JMP	Jump	J	0C		PC = {(PC + 1)[31:26], address}
HAL	Halt	J	3F		

Table 2. NYU-6463 Processor Instruction Set

Processor 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 to be utilized for other blocks. 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.

• **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. The data memory stores the data and is accessed using load word and store word instructions.

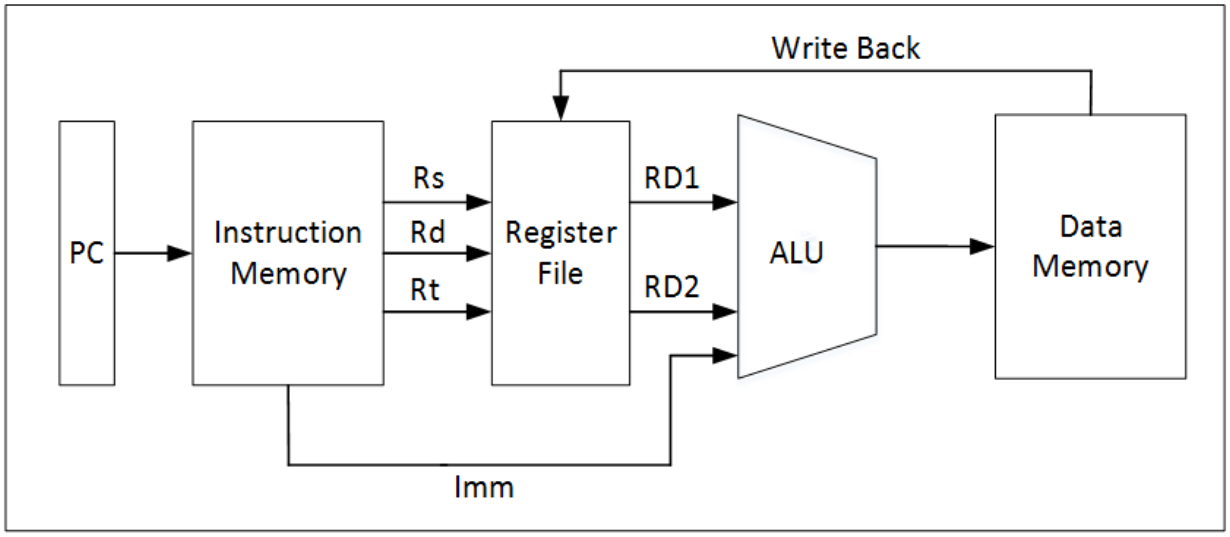

Figure 2. NYU-6463 Processor

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 (for NYU-6463 Processor, the shamt field is not used). 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. The register file reads in the requested addresses and outputs the data values contained in these registers. These data values can then be operated on by the ALU whose operation is determined by the control unit to either compute a memory address (e.g. load or store), compute an arithmetic result (e.g. add, and or sub), or perform a 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.

What you need to do

1. Implement the NYU-6463 Processor with the specification described above. 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.
2. Do a performance (max speed of your processor) and area (number of gates you used from each type) analysis. Explain your analysis comprehensively.
3. Write an assembly program to implement the RC5 block cipher (encryption and decryption) as well as round key generation using the instructions in Table 1. Submit your assembly codes alongside your processor implementation.
4. Convert the assembly codes into machine code and run them on your designed processor and show that it works properly. You should read the key and plaintext from the memory and write the ciphertext back to memory.
5. Implement your design on FPGA. You should be able to show the result of the execution of the program after every cycle. (Both single instruction stepping and complete program execution should be supported).
6. Describe how many cycles are required to complete RC5 encryption and decryption on NYU-6463 Processor.
7. Support updating the instruction memory (changing the program being executed) while your processor is running on the FPGA (optional).

Deliverables:

1. Share your github repository with achd.el6463@gmail.com.
2. Processor source code and assembly codes.
3. Your report in pdf format including:
   - Design block diagram,
   - Simulation screenshots for each of the checkpoints in next page.
   - High level description of how you implement and run 3 components of RC5 (encryption, decryption and key expansion) on your designed processor,
   - Description of processor interfaces (how you control the inputs and observe the outputs),
   - Performance and area analysis,
   - Details about how you verified your overall design.

Project Checkpoints:

1. Complete designing ALU.
2. Complete designing Decoder Unit.
3. Complete the processor design.
4. Run the sample programs given below & verify your CPU.
5. Complete RC5 Assembly code for Encryption and Decryption.
6. Complete the processor design with additional interfaces and single stepping (7-segment LEDs, Switches for input control).
7. Complete RC5 Assembly code (Key Expansion, Encryption and Decryption).
8. Optional: Complete the processor design with mechanism to update Instruction memory (I-Mem). (You can use just switches & push buttons to update I-Mem, or use other interfaces such as UART)

Date for presentation and demo will be announced soon.

Sample Programs:

We've provided two sample programs and their corresponding machine code to test your processor.

Sample Program 1:

```
ADDI R1, R0, 7          // R1 = 7
ADDI R2, R0, 8          // R2 = 8
ADD R3, R1, R2          // R3 = R1 + R2 =15
HAL                     // HALT
```

Sample Program 1 Machine Code:

```
00000100000000010000000000000111
00000100000000010000000000001000
00000000010000010001100000010000
11111100000000000000000000000000
```

Sample Program 2:
This program checks complete instruction set of 18 Instructions. Present the final result (register contents and data memory and program counter value) of this program in your report.

```
ADDI R1, R0, 2          // R1=2
ADDI R3, R0, 10         // R3=10D
ADDI R4, R0, 14         // R4=14D (D=Decimal)
ADDI R5, R0, 2          // R5=2
SW R4, 2(R3)            // 14D is stored in Data memory location 12D
SW R3, 1(R3)            // 10D is stored in Data memory location 11D
SUB R4, R4, R3
SUBI R4, R0, 1
AND R4, R2, R3
ANDI R4, R2, 10
OR R4, R2, R3
LW R2, 1(R3)            // R2=10D (Loaded back from memory)
ORI R4, R2, 10
NOR R4, R2, R3
SHL R4, R2, 10
SHR R4, R2, 10
BEQ R5, R0, -2
BLT R5, R4, -2
BNE R5, R4, 0
JMP 22
HAL
```

Sample Program 2 Machine Code:

```
00000100000000010000000000000010
00000100000000011000000000001010
00000100000000100000000000001110
00000100000000101000000000000010
00100000011000110000000000000001
00000000011001000010000000010001
00001000000001000000000000000001
00000000011000100010000000010010
00001100010001000000000000001010
00000000011000100010000000010011
00011100011000100000000000000001
00010000010001000000000000001010
00000000011000100010000000010100
00010100010001000000000000001010
00011000010001000000000000001010
00100000001001000000000000000010
00101000000001011111111111111110
00100100100001011111111111111110
00101100100001010000000000000000
00110000000000000000000000010110
11111100000000000000000000000000
```

Deadlines:

Please check announcements in NYU Classes.