NYU-6463 Processor Design

For the final project, you will implement a 32-bit processor in VHDL, called NYU-6463 Processor, which can execute programs. The processor should support 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.

Opcode (6-bits)	Rs (5-bits)	Rt (5-bits)	Rd (5-bits)	Shamt (5-bits)	Funct (6-bits)

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

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

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	1	Rd = Rs + Rt
ADDI	Add Immediate	I	01		Rt = Rs + SignExt(Imm)
SUB	Subtract Registers	R	00	3	Rd = Rs - Rt
SUBI	Subtract Immediate	I	02		Rt = Rs - SignExt(Imm)
AND	Register bitwise And	R	00	5	Rd = Rs & Rt
ANDI	And Immediate	I	03		Rt = Rs & SignExt(Imm)
OR	Register bitwise OR	R	00	7	Rd = Rs | Rt
NOR	Register bitwise NOR	R	00	9	Rd = !(Rs | Rt)
ORI	OR Immediate	I	04		Rt = Rs | SignExt(Imm)
SHR	Shift Right by immediate bits	I	05		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 + x + Imm
BEQ	Branch if equal	I	0A		If (Rs = = Rt) then PC = PC + x + Imm
BNE	Branch if not equal	I	0B		If (Rs != Rt) then PC = PC + x + Imm
JMP	Jump	J	0C		PC = target 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 (Word-addressable or Byte-addressable):** 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.

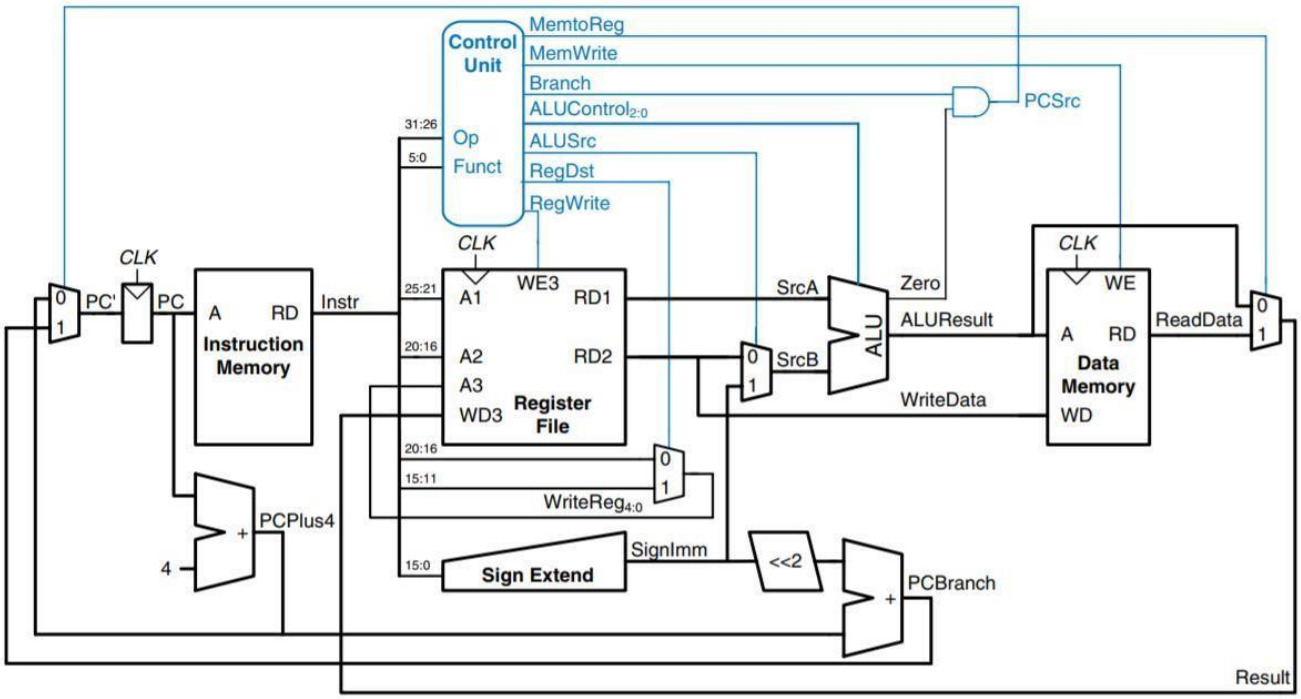

Figure 7.11 Complete single-cycle MIPS processor

Source-Digital Design and Computer Architecture by David Harris, Sarah L. Harris (page 375)

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. 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).
4. Write a program to implement the RC5 block cipher as well as round key generation using only the instructions from 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.
   a. Describe how many cycles are required to complete RC5 encryption and decryption on NYU-6463 Processor.
   b. Draw you program's flow chart
5. Support changing the program while your processor is running on the FPGA.

Deliverables:

1. Put your code in BitBucket repository and share it with the TAs. (mos283,wz1230, ka1809)
2. Put your processor source code and assembly code in a zipped folder.

3. Your report in pdf format including your
   a. design block diagram
   b. simulation screen shots
   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

4. demo video of the processor executing RC5 algorithms on FPGA.(5 to 10 minutes)

5. How you are going to optimate your design in future

Note:

No extra instruction allowed unless you get permission to use the following VHDL operators while designing the processor.

Deadlines:

Nov 19th - Implementation and demo of the processor on the fpga. We will provide small programs with assembly code to use. Each team is required to submit the **demo video**, **vhd code** and **test-benches**.

Nov 26th - Assembly coding of encryption - demo of its working on fpga. assuming the key is availiable. Each team is required to submit the **demo video**, **assembly code**, and **test-benches**.

Dec 3rd - Assembly coding of key-expansion and decryption. Each team is required to submit the **demo video**, **assembly code**, and **test-benches**.

Dec 10th - Final Demo. This will be done in person. Time and location will be announced.

Dec 17th - Final report due.