Cairo University
Faculty of Engineering
EECE Department

Spring 2020
Second Year
Measurements Course

N-BIT GENERAL PURPOSE INTEGER PROCESSOR

Objective:

To design a simple processor. The design should conform to the Instruction Set Architecture (ISA) specifications described in the following sections (or been uniquely created). The processor will be implemented on a PCB-mounted FPGA. The FPGA-implemented processor will then be used in a simple application using LCD and keypad.

Introduction:

The processor in this project has a RISC-like ISA. Instructions are fixed length instructions with N-bits width. There are general purpose registers; (R0, R1 … RX:X is number of registers) grouped in a single register file. The memory address space is $2^N * N$ bits and is N-bits addressable. The memory is to be divided into independent data and instruction memories. There is an Arithmetic, Logic and Shift Unit (ALSU) which is responsible of executing integer arithmetic, logic and shift instructions.

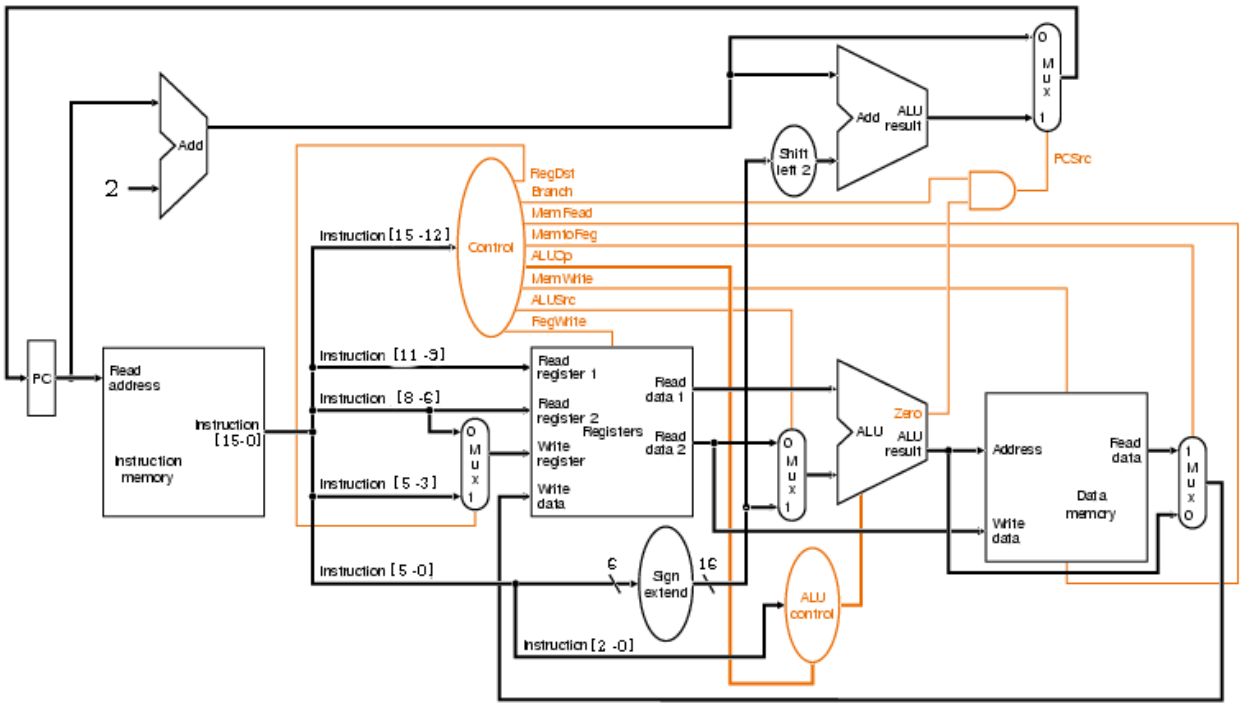
Project Flow:

Each team shall consist of 10 up to 15 team members.

All teams are required to:

- Write the VHDL (verilog is an alternative option) code of the processor.
- Synthesizing and Implementing the processor on Xilinx ISE program.
- Selecting a simple application for your processor interfacing with LCD and Keypad
- Designing a PCB that contains an FPGA and the selected application.
- Configuring the FPGA (mounted on the PCB) with your Design.

Architecture Overview (Top level view of the conventional 16 bits / 2 bytes MIPS processor):



Components:

1) ALSU

ALSU stands for Arithmetic, Logic, and Shifting Unit. It's the main executing unit where most of the instructions are actually executed. It's divided into three sub-units: Arithmetic unit, Shift unit, and Logic unit.

2) Register File

The used register file contains 8 registers; each of them is a 16-bit register. The register file has two read (output) ports and a single write (input) port. Each of these ports should have its own 3-bit address bus and a read/write signal. Details of designing the register file are up to each team.

3) Memory interfacing

The processor interfaces with a 2^{16} address space memory. Hence it has 16 address lines. The memory, data or instruction, should be designed to be able to transfer 16-bits (word=2 bytes) each transaction.

4) Instruction Register (IR)

The IR register is 16-bit register in the processor; it should hold the data of the fetched instruction while it's being decoded and executed. The opcodes brought to the IR using one memory transaction.

5) Address Calculating Unit

The address sent to the PC can have multiple sources. For data transfer instructions such as an immediate address inside the opcode, an indirect address from a register pointed to by the opcode. For branch (jump) or call instructions, the address is given as a PC-relative offset. Hence to find the actual physical address, we have to add this offset to the current PC value.

6) Control Unit

Control Unit is the actual brain that controls the whole processor. It reads the instruction opcode from the IR, decodes it and sends the appropriate control signals needed each clock to every block in the processor. These control signals are all the address lines, MUX selections, enables, read/write signals, etc. Control unit is the core of the processor, most of the work and optimization is done in the Control unit. It controls the decoder, which is very critical in determining the maximum clock frequency of the processor.

ISA Specifications

Instructions are categorized into three basic categories:

1. **Data transfer** — Data transfer instructions are responsible for moving data among memory locations, processor registers, or input ports.
2. **Arithmetic/Logical/Shift** — These instructions perform arithmetic, logical, and shift operations.
3. **Control Transfer Instructions** — Control-transfer instructions (**CTI**s) include PC-relative branches, function calls, and register-indirect jumps.

A) Registers

PC<0:15>	16-bit program counter
R[0:7] <0:15>	8 16-bit general purpose registers
OUT <0:15>	Output register

B) Processor Input and Outputs

IN_PORT<0:n>	n-bit data input port
OUT_PORT<0:15>	16 -bit data output port
RESET_IN<0>	Reset signal
CLK<0>	Clock signal

C) Signals Description:

If RESET.IN is asserted: clear all registers and set PC = 0

D) Instruction-format

OP<0:3>	4-bit operation code
RS<0:2>	3-bit source register specifier
RT<0:2>	3-bit (source/destination) register specifier
RD<0:2>	3-bit destination register specifier
IMM<0:5>	6-bit immediate value, branch displacement or address displacement
TARGET<0:11>	12-bit jump target address

I-Type (Immediate)

OP 4-bit	RS 3-bit	RT 3-bit	IMM 6-bit

I-type instructions have a 6-bit immediate field that codes an immediate operand, a branch target offset, or a displacement for a memory operand. For a branch target offset, the immediate field contains the signed difference between the address of the following instruction and the target label.

J-Type (Jump)

OP 4-bit	TARGET 12-bit

The only J-type instructions are the jump instructions j and jal. These instructions require a 12-bit coded address field to specify the target of the jump. The coded address is formed from the bits at positions 11 to 0 in the binary representation of the address.

When a J-type instruction is executed, a full 16-bit jump target address is formed by concatenating the high order four bits of the PC (the address of the instruction following the jump), and the 12 bits of the TARGET field.

R-type (Register) (Opcode 0x0)

OP 4-bit	RS 3-bit	RT 3-bit	RD 3-bit	FUNC 3-bit

Main processor instructions that do not require a target address, immediate value, or branch displacement use an R-type coding format. This format has fields for specifying of up to three registers. For instructions that do not use all of these fields, the unused fields are coded with all 0 bits.

Instruction	Meaning	Type	Opcode	FUNC
ADD	Add	R	0x0	0x0
SUB	Subtract	R	0x0	0x1
AND	Bitwise AND	R	0x0	0x2
OR	Bitwise OR	R	0x0	0x3
NOR	Bitwise NOR (NOT-OR)	R	0x0	0x4
XOR	Bitwise XOR (Exclusive-OR)	R	0x0	0x5
SLL	Logical Shift Left	R	0x0	0x6
SRL	Logical Shift Right	R	0x0	0x7
ADDI	Add Immediate	I	0x4	NA
ANDI	Bitwise AND Immediate	I	0x5	NA
ORI	Bitwise OR Immediate	I	0x6	NA
LW	Load Word	I	0x7	NA
SW	Store Word	I	0x8	NA
J	Jump to Address	J	0xb	NA
JAL	Jump and Link	J	0xc	NA
JR	Jump to Address in Register	R	0x3	0x0
BEQ	Branch if Equal	I	0x9	NA
BNE	Branch if Not Equal	I	0xa	NA
IN	Store the input value	R	0x1	0x0
OUT	Output a register value	R	0x2	0x0
NOP	No operation	-	0xe	NA
HLT	Halt	-	0xf	NA

E) Instruction Description

Assembly Format	Description
ADD R_d, R_s, R_t	R_d <-- [R_s] + [R_t]; PC <-- [PC] + 2
SUB R_d, R_s, R_t	R_d <-- [R_s] - [R_t]; PC <-- [PC] + 2
AND R_d, R_s, R_t	R_d <-- [R_s] AND [R_t]; PC <-- [PC] + 2
OR R_d, R_s, R_t	R_d <-- [R_s] OR [R_t]; PC <-- [PC] + 2
NOR R_d, R_s, R_t	R_d <-- [R_s] NOR [R_t]; PC <-- [PC] + 2
XOR R_d, R_s, R_t	R_d <-- [R_s] XOR [R_t]; PC <-- [PC] + 2
SLL R_d, R_s	R_d <-- [R_s] << 1; PC <-- [PC] + 2
SRL R_d, R_s	R_d <-- [R_s] >> 1; PC <-- [PC] + 2
ADDI R_t, R_s, immediate	R_t <-- [R_s] + ($[I_5]^{10}$ || [$I_{5..0}$]); PC <-- [PC] + 2
ANDI R_t, R_s, immediate	R_t <-- [R_s] AND (0^{10} || [$I_{5..0}$]); PC <-- [PC] + 2
ORI R_t, R_s, immediate	R_t <-- [R_s] OR (0^{10} || [$I_{5..0}$]); PC <-- [PC] + 2
LW R_t,offset(R_s)	R_t <-- M{[R_s] + $[I_5]^{10}$ || [$I_{5..0}$]}; PC <-- [PC] + 2
SW R_t,offset(R_s)	M{[R_s] + $[I_5]^{10}$ || [$I_{5..0}$]} <-- [R_t]; PC <-- [PC] + 2
J jump_target	PC <-- [$PC_{15..13}$] || [$I_{11..0}$] || 0
JAL jump_target	R_7 <-- [PC] + 2; PC <-- [$PC_{15..13}$] || [$I_{11..0}$] || 0
JR R_s	PC <-- [R_s]
BEQ R_s,R_t,offset	if [R_s] = [R_t] then PC <-- [PC] + 2 + ($[I_5]^9$ || [$I_{5..0}$] || 0) else PC <-- [PC] + 2
BNE R_s,R_t,offset	if [R_s] <> [R_t] then PC <-- [PC] + 2 + ($[I_5]^9$ || [$I_{5..0}$] || 0) else PC <-- [PC] + 2
IN R_d	R_d <-- IN_PORT; PC <-- [PC] + 2
OUT R_s	OUT_PORT <-- [R_s]; PC <-- [PC] + 2
NOP	PC <-- [PC] + 2
HLT	-

[$I_{y..x}$] denotes the contents of the Instruction register from bit of position x to bit of position y.

[$PC_{y..x}$] denotes the contents of the Program Counter register from bit of position x to bit of position y.

|| indicates concatenation of bit fields and **Superscripts** indicate repetition of a binary value.

Delivery Phases

Phase 1 (Deadline : 13-03-2020)

- Finishing the PCB design and order it from the manufacturer
- Selecting the needed electronic components and ordering it from the distributor
- Designing a **Full** schematic of the processor. In other words, you will be required to draw all the components (entities) of the processor as well as the input and outputs for each entity. All the control signals should be clearly shown in the schematic. The goal of this phase is to plan for the architecture of the processor; next phases are highly dependent on the first delivery. Remember don't show full details in the schematic.
- Selecting the application

Phase 2 (Deadline : 19-4-2020)

- Writing the VHDL codes for the schematic developed in the earlier phase. You will be required to finish the **Full** design of the processor.
- Synthesizing the written VHDL on ISE program with no errors.

Phase 3 (Deadline : 14-5-2020)

- Write the code of the application using the above instruction set
- Testing the PCB and running the application on the processor.

General Advice

- Compile and synthesize your design on regular bases (after each modification) so that you can figure out new errors early. Accumulated errors are harder to track.
- Use the engineering sense to back trace the error source.
- As much as you can, don't ignore warnings, just if you know that it's a non solved warnings.
- After each major step, and if you have a working processor, save the design before you modify it.
- Start early and give yourself enough time for testing.