Cairo University	CMP 301
Faculty of Engineering	Spring 2020
Computer Engineering Department	

Architecture Project

Objective

To design and implement a simple 5-stage pipelined processor, **Harvard Architecture**.
The design should conform to the ISA specification described in the following sections.

Introduction

The processor in this project has a RISC-like instruction set architecture. There are eight 4-byte general purpose registers; R_0, till R7. Another two general purpose registers, One works as program counter (PC). And the other, works as a stack pointer (SP); and hence; points to the top of the stack. The initial value of SP is (2^32-1). The memory address space is 4 GB of 16-bit width and is word addressable. (N.B. word = 2 bytes). [redacted] **update 28_March : The bus between memory and the processor is 16-bit widths for instruction memory and 32-bit widths for data memory**

When an interrupt occurs, the processor finishes the currently fetched instructions (instructions that have already entered the pipeline), then the address of the next instruction (in PC) is saved on top of the stack, and PC is loaded from address [2-3] of the memory (the address takes two words). To return from an interrupt, an RTI instruction loads PC from the top of stack, and the flow of the program resumes from the instruction after the interrupted instruction. **Take care of corner cases like Branching,Calling.**

ISA Specifications

A) Registers

R[0:7]<31:0> ; Eight 32-bit general purpose registers
PC<31:0> ; 32-bit program counter
SP<31:0>; 32-bit stack pointer
CCR<3:0> ; condition code register
Z<0>:=CCR<0> ; zero flag, change after arithmetic, logical, or shift operations
N<0>:=CCR<1> ; negative flag, change after arithmetic, logical, or shift operations
C<0>:=CCR<2> ; carry flag, change after arithmetic or shift operations.

B) Input-Output

IN.PORT<31:0> ; 32-bit data input port
OUT.PORT<31:0> ; 32-bit data output port
INTR.IN<0> ; a single, non-maskable interrupt
RESET.IN<0> ; reset signal

Rsrc1 ; 1st operand register
Rsrc2 ; 2nd operand register
Rdst : result register
EA ; Effective address (20 bit)
Imm ; Immediate Value (16 bit)

Take Care that Some instructions will Occupy more than one memory location

Mnemonic	Function	Grade
One Operand		
NOP	PC ← PC + 1	**3 Marks**

NOT Rdst	NOT value stored in register Rdst
	R[Rdst] ← 1's Complement(R[Rdst]);
	If (1's Complement(R[Rdst]) = 0): Z ←1; else: Z ←0;
	If (1's Complement(R[Rdst]) < 0): N ←1; else: N ←0	

INC Rdst	Increment value stored in Rdst
	R[Rdst] ←R[Rdst] + 1;
	If ((R[Rdst] + 1) = 0): Z ←1; else: Z ←0;
	If ((R[Rdst] + 1) < 0): N ←1; else: N ←0	

DEC Rdst	Decrement value stored in Rdst
	R[Rdst] ←R[Rdst] – 1;
	If ((R[Rdst] – 1) = 0): Z ←1; else: Z ←0;
	If ((R[Rdst] – 1) < 0): N ←1; else: N ←0	

OUT Rdst	OUT.PORT ← R[Rdst]	
IN Rdst	R[Rdst] ←IN.PORT	
Two Operands		
SWAP Rsrc, Rdst	Store the value of Rsrc 1 in Rdst and the value of Rdst in Rsc1 flag shouldn't change	**4 Marks**
ADD Rsrc1, Rsrc2, Rdst	Add the values stored in registers Rsrc1, Rsrc2 and store the result in Rdst If the result =0 then Z ←1; else: Z ←0; If the result <0 then N ←1; else: N ←0	
IADD Rsrc1,Rdst,Imm	Add the values stored in registers Rsrc1 to Immediate Value and store the result in Rdst If the result =0 then Z ←1; else: Z ←0; If the result <0 then N ←1; else: N ←0	
SUB Rsrc1, Rsrc2, Rdst	Subtract the values stored in registers Rsrc1, Rsrc2 and store the result in Rdst If the result =0 then Z ←1; else: Z ←0; If the result <0 then N ←1; else: N ←0	
AND Rsrc1, Rsrc2, Rdst	AND the values stored in registers Rsrc1, Rsrc2 and store the result in Rdst If the result =0 then Z ←1; else: Z ←0; If the result <0 then N ←1; else: N ←0	
OR Rsrc1, Rsrc2, Rdst	OR the values stored in registers Rsrc1, Rsrc2 and store the result in Rdst If the result =0 then Z ←1; else: Z ←0; If the result <0 then N ←1; else: N ←0	

SHL Rsrc, Imm	Shift left Rsrc by #Imm bits and store result in same register **Don't forget to update carry**	
SHR Rsrc, Imm	Shift right Rsrc by #Imm bits and store result in same register **Don't forget to update carry**	
Memory Operations		
PUSH Rdst	M[SP--] ← R[Rdst];	**2.5 Marks**
POP Rdst	R[Rdst] ← M[++SP];	
LDM Rdst, Imm	Load immediate value (16 bit) to register Rdst R[Rdst] ← {0,Imm<15:0>}	
LDD Rdst, EA	Load value from memory address EA to register Rdst R[Rdst] ← M[EA];	
STD Rsrc, EA	Store value in register Rsrc to memory location EA M[EA] ←R[Rsrc];	
Branch and Change of Control Operations		
JZ Rdst	Jump if zero If (Z=1): PC ←R[Rdst]; (Z=0)	**2.5 Marks**
JMP Rdst	Jump PC ←R[Rdst]	
CALL Rdst	(M[SP] ← PC + 1; sp-2; PC ← R[Rdst])	
RET	sp+2, PC ←M[SP]	
RTI	sp+2; PC ← M[SP]; Flags restored	

Input Signals		**Grade**
Reset	PC ←{M[1], M[0]} //memory location of zero and all registers get reseted	0.5 Mark
Interrupt	M[Sp]←PC; sp-2;PC ← {M[3],M[2]}; Flags preserved	2 Mark

Phase1 Requirement: Report Containing: (week#9)

- Instruction format of your design
  - Opcode of each instruction
  - Instruction bits details (i.e. which bits are reserved to Rsrc1,Rsrc2,..)
- Schematic diagram of the processor with data flow details.
  - ALU / Registers / Memory Blocks
  - Dataflow Interconnections between Blocks & its sizes (all wires should be drawn and named between blocks)
  - Control Unit detailed design (each instruction and the control signals they generate)
- Pipeline stages design
  - Pipeline registers details (Size, Input, Connection, …)
  - Pipeline hazards and your solution including
    i. Data Forwarding
    ii. Hazard Detection Unit
    iii. Dynamic Branch Prediction

Software Delivery: (week#11)

- A software to compile sample programs and generate the equivalent hex files to be loaded to the memory. (Assembler code that converts assembly program (Text File) into machine code according to your design (Memory File))

Phase2 Requirement: (week#13)

- Implement and integrate your architecture
  - **VHDL** Implementation of each component of the processor
  - **VHDL** file that integrates the different components in a single module
- Simulation Test code that reads a program file and executes it on the processor.
  - Setup the simulation wave
  - Load Memory File & Run the test program
- Report that contains any design changes after phase1
- Report that contains pipeline hazards considered and how your design solves it.

Project Testing

- You will be given different test programs. You are required to compile and load it onto the RAM and **reset** your processor to start executing from memory location 0000h. Each program would test some instructions (you should notify the TA if you haven't implemented or have logical errors concerning some of the instruction set).
- **You MUST prepare a waveform using do files with the main signals showing that your processor is working correctly (R0-R7, PC,SP,Flags,CLK,Reset,Interrupt, IN.port,Out.port).**

Evaluation Criteria

- Each project will be evaluated according to the number of instructions that are implemented, and Pipelining hazards handled in the design. Table 2 shows the evaluation criteria.
- Failing to implement a working processor will nullify your project grade. No credits will be given to individual modules or a non-working processor.
- Unnecessary latching or very poor understanding of underlying hardware will be penalized.

- Individual Members of the same team can have different grades, you can get a zero grade if you didn't work while the rest of the team can get fullmark, Make sure you balance your Work distribution.
- Failing to deliver software(Assembler) or prepare test cases on time will result in a penalty.
- **Cheating == Zero in All course work not just project.**

Table 2: Evaluation Criteria

Marks Distribution	Design	2.5
	Instructions	Stated above (13.5 marks)
	Handling Hazard	2 marks
	2-bit dynamic branch prediction with address calculation in fetch stage (take care of hazards and forwarding)	2

Team Members

- Each team shall consist of a maximum of four members

Phase 1 Due Date

- Delivery a softcopy on Elearning, **and** bring the hardcopy in the discussion.
- Week 9, Sunday 5th of April 2020, The Discussion will be during the regular section.
- Delivery should be a PDF named with your team Number and the cover page should contains your team members

Software Delivery: (week#11)

- Week 11, Tuesday, a zipped folder named with your team number with contains
  - your software,
  - a readme on how to run it
  - any necessary libraries.
  - an example program that contains all instructions.
- on elearning

Project Due Date

- Delivery a softcopy on Elearning, or CD at eng. Dina's MailBox
- Week 13, Sunday 3rd of May 2020. The demo will be during the regular lab session.

General Advice

1. Compile your design on regular bases (after each modification) so that you can figure out new errors early. Accumulated errors are harder to track.
2. Use the engineering sense to back trace the error source.
3. As much as you can, don't ignore warnings.
4. Read the transcript window messages in Modelsim carefully.
5. After each major step, and if you have a working processor, save the design before you modify it (use versioning tool if you can as git & svn).
6. Always save the ram files to easily export and import them.
7. Start early and give yourself enough time for testing.
8. Integrate your components incrementally (i.e: Integrate the RAM with the Registers, then integrate with them the ALU …).

9. Use coding convention to know each signal functionality easily.
10. Try to simulate your control signals sequence for an instruction (i.e: Add) to know if your timing design is correct.
11. There is no problem in changing the design after phase1, but justify your changes.
12. Always reset all components at the start of the simulation.
13. Don't leave any input signal float "U", set it with 0 or 1.
14. Remember that your VHDL code is a HW system (logic gates, Flipflops and wires).
15. Use Do files instead of re-forcing all inputs each time.