Cairo University	CMP N 301
Faculty of Engineering	Spring 2021
Computer Engineering Department	

Architecture Project

Objective

To design and implement a simple 5-stage pipelined processor, **von Neumann or Harvard**.
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 a 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 (2^20-2). The memory address space is 1 MB of 16-bit width and is word addressable. (N.B. word = 2 bytes). You are allowed to make the data bus 32 bits to access two consecutive words.

~~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 1 of the memory. To return from an interrupt, an RTI instruction loads the 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.**~~

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<2:0>	; 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

Rsrc ; 1st operand register
Rdst ; 2nd operand register and result register field
Offset ; Address offset (16 bit)
Imm ; Immediate Value 5 bits for shifting and 16 bits otherwise

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

Mnemonic	Function	Grade
One Operand		
NOP	PC ← PC + 1	**5 Marks**
SETC	C ←1	
CLRC	C ←0	
CLR Rdst	Sets Rdst to zeros Z ←1	

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	

NEG Rdst	Negate the value stored in register Rdst
	R[Rdst] ← 2's Complement(R[Rdst]);
	If (2's Complement(R[Rdst]) = 0): Z ←1; else: Z ←0;
	If (2's Complement(R[Rdst]) < 0): N ←1; else: N ←0	

OUT Rdst	OUT.PORT ← R[Rdst]	
IN Rdst	R[Rdst] ←IN.PORT	
Two Operands		
MOV Rsrc, Rdst	Move value from register Rsrc to register Rdst	**5 Marks**
ADD Rsrc, Rdst	Add the values stored in registers Rsrc, Rdst and store the result in Rdst and updates carry If the result =0 then Z ←1; else: Z ←0; If the result <0 then N ←1; else: N ←0	
SUB Rsrc, Rdst	Subtract the values stored in registers Rsrc, Rdst and store the result in Rdst and updates carry If the result =0 then Z ←1; else: Z ←0; If the result <0 then N ←1; else: N ←0	
AND Rsrc, Rdst	AND the values stored in registers Rsrc, Rdst 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 Rsrc, Rdst	OR the values stored in registers Rsrc, Rdst 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 Rdst,Imm	Add the values stored in registers Rdst to Immediate Value and store the result in Rdst and updates carry 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**	
RLC Rdst	Rotate left Rdst ; C ←R[Rdst]<31>; R[Rdst] ← R[Rdst]<30:0>&C	
RRC Rdst	Rotate right Rdst ; C ←R[Rdst]<0>; R[Rdst] ←C&R[Rdst]<31:1>;	
	Memory Operations	
PUSH Rdst	X[SP] ← R[Rdst]; SP-=2	
POP Rdst	SP+=2; R[Rdst] ← X[SP];	
LDM Rdst, Imm	Load immediate value (16 bit) to register Rdst R[Rdst] ← Imm<15:0>	**4 Marks**
LDD Rdst, offset(Rsrc)	Load value from memory address Rsrc + offset to register Rdst R[Rdst] ← M[Rsrc + offset];	
STD Rsrc1, offset(Rsrc2)	Store value that is in register Rsrc1 to memory location Rsrc2 + offset M[Rsrc2 + offset] ←R[Rsrc1];	
	Branch and Change of Control Operations	
JZ Rdst	Jump if zero If (Z=1): PC ←R[Rdst]; (Z=0)	
JN Rdst	Jump if negative If (N=1): PC ←R[Rdst]; (N=0)	
JC Rdst	Jump if negative If (C=1): PC ←R[Rdst]; (C=0)	**3 Marks**
JMP Rdst	Jump PC ←R[Rdst]	
CALL Rdst	(X[SP] ← PC + 1; sp-=2; PC ← R[Rdst])	
RET	sp+=2, PC ←X[SP]	
~~RTI~~	~~sp+=2; PC ← X[SP]; Flags restored~~	

Input Signals		Grade
Reset	PC ← M[0] //memory location of zero	0.5 Mark
~~Interrupt~~	~~X[Sp]←PC; sp-=2;PC ← M[1]; Flags preserved~~	~~3 Mark~~

Phase1 Requirement: Report Containing:

- Instruction format of your design
    - Opcode of each instruction
    - Instruction bits details
- Schematic diagram of the processor with data flow details. (Including Branching even if you are not planning to implement it).
    - ALU / Registers / Memory Blocks
    - Dataflow Interconnections between Blocks & its sizes

    - Control Unit detailed design
- Pipeline stages design
    - Pipeline registers details (Size, Input, Connection, …)
    - Pipeline hazards and your solution including
        1. Data Forwarding
        2. Static Branch Prediction

Phase2 Requirement

- 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
- Assembler code that converts assembly program (Text File) into machine code according to your design (Memory File)
- 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 in detail.
- 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.

Table 2: Evaluation Criteria

Marks Distribution	Instructions	Stated above (14.5 marks) that will be scaled to 21 marks
	Efficient Handling of Hazard	4 marks
Bonus Marks	Branching Instructions with control hazards	Bonus that can replace a quiz grade
	2-bit dynamic branch prediction with address calculation in fetch	2 mark bonus to credit work grade

Team Members

- Each team shall consist of a maximum of four members

Phase 1 Due Date

- Delivery a softcopy on blackboard.
- Week 9, Sunday 16th of May 2021 at 9 am, The discussion will be during the regular lab session.

Project Due Date

- Delivery a softcopy on blackboard.
- Week 12, Monday 14th of June 2021 at 9 am. 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. Start by finishing a working processor that does all one operands only. Integrating early will help you find a lot of errors. You can then add each type of instructions and integrate them into the working processor.
3. Use the engineering sense to back trace the error source.
4. As much as you can, don't ignore warnings.
5. Read the transcript window messages in Modelsim carefully.
6. After each major step, and if you have a working processor, save the design before you modify it (use a versioning tool if you can as git & svn).
7. Always save the ram files to easily export and import them.
8. Start early and give yourself enough time for testing.
9. Integrate your components incrementally (i.e: Integrate the RAM with the Registers, then integrate with them the ALU …).
10. Use coding conventions to know each signal functionality easily.
11. Try to simulate your control signals sequence for an instruction (i.e: Add) to know if your timing design is correct.
12. There is no problem in changing the design after phase1, but justify your changes.
13. Always reset all components at the start of the simulation.
14. Don't leave any input signal float "U", set it with 0 or 1.
15. Remember that your VHDL code is a HW system (logic gates, Flipflops and wires).
16. Use Do files instead of re-forcing all inputs each time.