



## Architecture Project

### Objective

To design and implement a simple 5-stage pipelined processor, Harvard (Program and data memories are separated).

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 2-byte general purpose registers; R<sub>0</sub>, till R<sub>7</sub>. Another three special purpose registers, One works as a program counter (PC). The second is the Exception program counter. And the last, works as a stack pointer (SP); and hence; points to the top of the stack. The initial value of SP is (2<sup>20</sup>-1). The memory address space is 1 MB of 16-bit width and is word addressable. ( **N.B. word = 2 bytes**). The data bus can either be 16 or 32 bits according to your design.

When an exception occurs, the address of the instruction causing the exception is saved in the exception program counter, and the correct exception handler is called. There are two types of exceptions that can occur: Empty Stack Exception and invalid memory address. Empty stack exception occurs when a pop is called while the stack is empty. The exception handler address for this exception is stored in M[2] and M[3]. Invalid memory address occurs when the address exceeds the range of 0xFF00. The exception handler address for this exception is stored in M[4] and M[5] in either data memory or instruction memory per your design.

### ISA Specifications

#### A) Registers

R[0:7]<15:0> ; Eight 16-bit general purpose registers

PC<31:0> ; 32-bit program counter

SP<31:0>; 32-bit stack pointer

EPC<31:0>; 32-bit exception program counter

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.

#### A) Input-Output

IN.PORT<16:0> ; 16-bit data input port

OUT.PORT<16:0> ; 16-bit data output port

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 16 bits

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

Mnemonic	Function	Grade
One Operand		
NOP	PC ← PC + 1	3.5 Marks
HLT	Freezes PC until a reset	
SETC	C ←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	
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	4 Marks
ADD Rdst, Rsrc1, Rsrc2	Add the values stored in registers Rsrc1, Rsrc2 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 Rdst, Rsrc1, Rsrc2	Subtract the values stored in registers Rsrc1, Rsrc2 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 Rdst, Rsrc1, Rsrc2	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	
IADD Rdst, Rsrc ,Imm	Add the values stored in registers Rsrc 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	
Memory Operations		
PUSH Rdst	X[SP] ← R[ Rdst ]; SP=1	4 Marks
POP Rdst	SP+=1; R[ Rdst ] ← X[SP];	
LDM Rdst, Imm	Load immediate value (16 bit) to register Rdst	

	$R[Rdst] \leftarrow Imm<15:0>$	
LDD Rdst, offset(Rsrc)	Load value from memory address Rsrc + offset to register Rdst $R[Rdst] \leftarrow M[R[Rsrc] + offset];$	
STD Rsrc1, offset(Rsrc2)	Store value that is in register Rsrc1 to memory location Rsrc2 + offset $M[R[Rsrc2] + offset] \leftarrow R[Rsrc1];$	
<b>Branch and Change of Control Operations</b>		
JZ Rdst	Jump if zero If (Z=1): $PC \leftarrow R[Rdst];$ (Z=0)	<b>4 Marks</b>
JN Rdst	Jump if negative If (N=1): $PC \leftarrow R[Rdst];$ (N=0)	
JC Rdst	Jump if negative If (C=1): $PC \leftarrow R[Rdst];$ (C=0)	
JMP Rdst	Jump $PC \leftarrow R[Rdst]$	
CALL Rdst	$(X[SP] \leftarrow PC + 1; sp -= 2; PC \leftarrow R[Rdst])$	
RET	$sp += 2; PC \leftarrow X[SP]$	
INT index	$X[SP] \leftarrow PC; sp -= 2; \text{Flags reserved};$ $PC \leftarrow M[index + 6] \& M[index + 7]$	
RTI	$sp += 2; PC \leftarrow X[SP]; \text{Flags restored}$	

<b>Input Signals</b>		<b>Grade</b>
Reset	$PC \leftarrow M[0] \& M[1]$ //memory location of zero	0.5 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.
  - 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
    - i. Data Forwarding
    - ii. 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 the memory location written in address 0000h (PC = M[0] & M[1] not 0). 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,EPC,,Flags,CLK,Reset,IN.port,Out.port). Show main signals and only main signals please, try to keep wave clean

### 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

<b>Marks Distribution</b>	Instructions	Stated above 14 (16 scaled to 14)
	Data Hazards	1.5 marks
	Control Hazards	1.5 marks
	Exception Handling	3 marks
<b>Bonus Marks</b>	Make the interrupt vector table address dynamic. It starts at an address X where $X = M[6] \& M[7]$	2 mark bonus

### Team Members

- Each team shall consist of a **maximum of four members**

### Phase 1 Due Date

- Delivery a softcopy on blackboard.
- Week 9, Sunday 5th of December 2021 at 9 am, The discussion will be during the regular lab session.

### Project Due Date

- Delivery a softcopy on blackboard.
- Week 13, Sunday 2nd of January 2022 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.