Cairo University
Faculty of Engineering
Computer Engineering Department

Computer Architecture
Fall 2024

Computer 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; R0, till R7. 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 (2^12-1). The data & instruction memory address spaces are 4Kx16.

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 IM[2] and IM[3]. Invalid memory address occurs when the address exceeds the range of 0x0FFF. The exception handler address for this exception is stored in IM[4] and IM[5].

ISA Specifications

A) Registers

R[0:7]<15:0>	; Eight 16-bit general purpose registers
PC<15:0> ; 16-bit program counter	
SP<15:0>; 16-bit stack pointer	
EPC<15:0>; 16-bit exception program counter	
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<15:0>	; 16-bit data input port
OUT.PORT<15:0>	; 16-bit data output port
RESET.IN<0>	; reset signal

- **Legend**

IM	; Instruction Memory
DM	; Data Memory
Rsrc1	; 1st operand register
Rsrc2	; 2nd operand register
Rdst	; Destination register
Offset	; Address offset (16 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	**3.5 Marks**
HLT	Freezes PC until a reset	
SETC	C ←1	
NOT Rdst, Rsrc1	NOT value in Rsrc1 and store it in Rdst R[Rdst] ← 1's Complement(R[Rsrc1]); Updates Zero flag Updates Negative flag	
INC Rdst, Rsrc1	Increment value in Rsrc1 and store it in Rdst R[Rdst] ←R[Rsrc1] + 1; Updates Zero flag Updates Negative flag Updates Carry flag	
OUT Rsrc1	OUT.PORT ← R[Rsrc1]	
IN Rdst	R[Rdst] ←IN.PORT	
Two Operands		
MOV Rdst, Rsrc1	Move value from register Rsrc1 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 Updates Zero flag Updates Negative flag	
SUB Rdst, Rsrc1, Rsrc2	Subtract the values stored in registers Rsrc1, Rsrc2 and store the result in Rdst and updates carry Updates Zero flag Updates Negative flag	
AND Rdst, Rsrc1, Rsrc2	AND the values stored in registers Rsrc1, Rsrc2 and store the result in Rdst Updates Zero flag Updates Negative flag	
IADD Rdst, Rsrc1, Imm	Add the values stored in registers Rsrc1 to Immediate Value and store the result in Rdst and updates carry Updates Zero flag Updates Negative flag	
Memory Operations		
PUSH Rsrc1	DM[SP] ← R[Rsrc1]; SP-=1	**4 Marks**
POP Rdst	SP+=1; R[Rdst] ← DM[SP];	
LDM Rdst, Imm	Load immediate value (16 bit) to register Rdst R[Rdst] ← Imm<15:0>	
LDD Rdst, offset(Rsrc1)	Load value from memory address Rsrc1 + offset to register Rdst R[Rdst] ← DM[R[Rsrc1] + offset];	

STD Rsrc1, offset(Rsrc2)	Store value that is in register Rsrc1 to memory location Rsrc2 + offset DM[R[Rsrc2] + offset] ←R[Rsrc1];	
Branch and Change of Control Operations		
JZ Rsrc1	Jump if zero If (Z=1): PC ←R[Rsrc1]; Resets Zero flag (Z=0)	**4 Marks**
JN Rsrc1	Jump if negative If (N=1): PC ←R[Rsrc1]; Resets Negative flag (N=0)	
JC Rsrc1	Jump if carry If (C=1): PC ←R[Rsrc1]; Resets Carry flag (C=0)	
JMP Rsrc1	Jump PC ←R[Rsrc1]	
CALL Rsrc1	(DM[SP] ← PC + 1; sp-=2; PC ← R[Rsrc1])	
RET	sp+=2, PC ←DM[SP]	

INT index	DM[SP] ← PC + 1; sp-=2;Flags reserved;
	PC ← IM[index + 6] & IM[index + 7]
	index is either 0 or 2.	

RTI	sp+=2; PC ← DM[SP]; Flags restored	

Input Signals		**Grade**
Reset	PC ← IM[0] & IM[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
- Pipeline stages design
  - Pipeline registers details (Size, Input, Connection, …)
  - Pipeline hazards and your solution including
    1. i. Data Forwarding
    2. 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)

- Implement the project on DE1-SoC board.

- Report that contains
  - Any design changes after phase1
  - Pipeline hazards considered, how your design solves it and the performance enhancement.
  - Comparison between the processor and the pipeline processor according to:
    1. FPGA resources
    2. Clock speed

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 0x0000 (PC = IM[0] & IM[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. <u>No credits will be given to individual modules or a non-working processor</u>.
- 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 (16 scaled to 14)
	Data Hazards	1.5 marks
	Control Hazards	1.5 marks
	Exception Handling	3 marks
Bonus Marks	Make the interrupt vector table address dynamic. It starts at an address X where X = IM[6] & IM[7]	2 mark bonus

Team Members

- Each team shall consist of a maximum of four members

Phase 1 Due Date

- Delivery a softcopy.
- Week 10

Project Due Date

- Delivery a softcopy.
- Week 13

General Advice

1. Compile your design on regular basis (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).
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 or testbenches instead of re-forcing all inputs each time.