Faculty of Engineering
Electrical Engineering Dep.
Instructor: Dr.Gihan Naguib

CEE210/CEE216
Computer Engineering
2nd year computer/Commination
Spring 2024

Pipelined MIPS Processor Implementation

Objectives:

- Designing and testing a Pipelined 16-bit processor

Instruction Set Architecture

In this project, you will design a simple 16-bit RISC processor with seven 16-bit general purpose registers: R1 through R7. R0 is hardwired to zero and cannot be written, we are left with seven registers. There is also one special-purpose 16-bit register, which is the program counter (PC). All instructions are only 16 bits. There are three instruction formats, R-type, I-type, and J-type as shown below:

R-type format

5-bit opcode (Op), 3-bit destination register Rd, and two 3-bit source registers Rs & Rt and 2-bit function field F

Op^5	**F^2**	**Rd^3**	**Rs^3**	**Rt^3**

I-type format

5-bit opcode (Op), 3-bit destination register Rd, 3-bit source register Rs, and 5-bit immediate

Op^5	**Imm^5**	**Rs^3**	**Rt^3**

J-type format :

5-bit opcode (Op) and11-bit Immediate

Op^5	**Imm^{11}**

Register Use

For R-type instructions, Rs and Rt specify the two source register numbers, and Rd specifies the destination register number. The function field F can specify at most four functions for a given opcode. We can reserve several opcodes for R-type instructions.

For I-type instructions, Rs specifies a source register number, and Rt can be a second source or a destination register number. The immediate constant is only 5 bits because of the fixed-size nature of the instruction. The size of the immediate constant is suitable for our uses. The 5-bit immediate constant can be signed or unsigned depending on the opcode. The immediate constant is signed (range is -16 to +15), except for shift and rotate instructions (range is 0 to 31).

The J-type format is used by J (jump), JAL (jump-and-link), and for LUI instructions. The 11-bit immediate is used for PC-relative addressing and constant formation.

Instruction Encoding

Nine R-type instructions, twelve I-type instructions, and three J-type instructions are defined. These instructions, their meaning, and their encoding are shown below:

	Instruction	*Meaning*	*Encoding*				
R-Type	*AND*	***Reg(Rd)= REg(Rs) & Reg(Rt)***	**OP=0**	**F=0**	**Rd**	**Rs**	**Rt**
	OR	***Reg(Rd)= REg(Rs) | Reg(Rt)***	**OP=0**	**F=1**	**Rd**	**Rs**	**Rt**
	XOR	***Reg(Rd)= REg(Rs) ^ Reg(Rt)***	**OP=0**	**F=2**	**Rd**	**Rs**	**Rt**
	Nor	***Reg(Rd)= ~ (REg(Rs) | Reg(Rt))***	**OP=0**	**F=3**	**Rd**	**Rs**	**Rt**
	Add	***Reg(Rd)= REg(Rs) + Reg(Rt)***	**OP=1**	**F=0**	**Rd**	**Rs**	**Rt**
	Sub	***Reg(Rd)= REg(Rs) - Reg(Rt)***	**OP=1**	**F=1**	**Rd**	**Rs**	**Rt**
	SLT	***Reg(Rd)= (REg(Rs)* <s *Reg(Rt))* ?1:0**	**OP=1**	**F=2**	**Rd**	**Rs**	**Rt**
	SLTU	***Reg(Rd)* = (*REg(Rs)* unsigned < *Reg(Rt))*? 1:0**	**OP=1**	**F=3**	**Rd**	**Rs**	**Rt**
	JR	***PC=Reg (Rs)***	***OP=2***	***F=0***	***0***	***Rs***	***0***
I-Type	*AndI*	***Reg(Rt)* = *REg(Rs)* & (Imm5)**	***4***	**Imm^5**	***Rs***	***Rd***	
	ORI	***Reg(Rt)= REg(Rs)* | (Imm5)**	***5***	**Imm^5**	***Rs***	***Rd***	
	XORI	***Reg(Rt)= REg(Rs)* ^ (Imm5)**	***6***	**Imm^5**	***Rs***	***Rd***	
	AddI	***Reg(Rt)= REg(Rs)* + signed (Imm5)**	***7***	**Imm^5**	***Rs***	***Rd***	
	SLL	***Reg(Rt)* = *REg(Rs)* << (Imm4)**	***8***	**Imm^5**	***Rs***	***Rd***	
	SRL	**Reg(Rt) = Reg(Rs) zero>> Imm4**	***9***	**Imm^5**	***Rs***	***Rd***	
	SRA	**Reg(Rt) = Reg(Rs) sign>> Imm4**	***10***	**Imm^5**	***Rs***	***Rd***	
	ROR	**Reg(Rt) = Reg(Rs) rot>> Imm4**	***11***	**Imm^5**	***Rs***	***Rd***	
	LW	***Reg(Rt)* = MEM[*REg(Rs)* + signed(Imm5)]**	***12***	**Imm^5**	***Rs***	***Rd***	
	SW	**MEM[*REg(Rs)* + signed(Imm5)]= *Reg(Rt)***	***13***	**Imm^5**	***Rs***	***Rd***	
	BEQ	**Branch if *REg(Rs)* == *Reg(Rt)***	***14***	**Imm^5**	***Rs***	***Rd***	
	BNE	**Branch if *REg(Rs)* != *Reg(Rt)***	***15***	**Imm^5**	***Rs***	***Rd***	
	BLT	**Branch if (Rs < Rt)**	***16***	**Imm^5**	***Rs***	***Rd***	
	BGE	**Branch if (Rs >= Rt)**	**17**	**Imm^5**	***Rs***	***Rd***	
J-Type	LUI	**R1 = Imm11 << 5**	***18***	***Imm^{11}***			
	J	**PC = PC + signed(Imm11)**	***30***	***Imm^{11}***			
	JAL	**R7 = PC + 1 , PC= PC + Signed(Imm11)**	***31***	***Imm^{11}***			

Instruction Description

.

Opcodes 0 and 1 are used for R-type instructions. Opcode 2 is used for the JR (jump register) instruction. Opcodes 4 through 17 are used for I-type instructions. The 5-bit immediate constant is zero-extended for ANDI, ORI, and XORI. It is sign-extended for the remaining instructions. There are three shift and one rotate instruction. To shift or rotate, use the lower 4 bits of Imm^5 as the shift/rotate amount. There is only one rotate left (ROL) instruction. To rotate right by n bits, you can rotate left by 16 – n bits, because registers are 16 bits. The Load Upper Immediate (LUI) is of the J-type to have an 11-bit immediate constant loaded into the upper 11 bits of register R1. The LUI can be combined with ORI to load any 16-bit constant into a register. Although the instruction set is reduced, it is still rich enough to write useful programs.

Memory

Your processor will have separate instruction and data memories with 2^{16} words each. Each word is 16 bits or 2 bytes. Memory is ***word addressable***. Only words (not bytes) can be read and written to memory, and each address is a word address. This will simplify the processor implementation. The PC contains a word address (not a byte address). Therefore, it is sufficient to increment the PC by 1 (rather than 2) to point to the next instruction in memory.
Also, the Load and Store instructions can only load and store words. There is no instruction to load or store a byte in memory

Register File

Implement a Register file containing seven 16-bit registers R1 to R7 with two read ports and one write port.R0 is hardwired to zero.

Arithmetic and Logical Unit (ALU)

Implement a 16-bit ALU to perform all the required operations (shown in tables above)

Addressing Modes

PC-relative addressing mode is used for branch and jump instructions.
For branching, the branch target address is computed as follows:

PC = PC + sign-extend (Imm^5),
by adding the contents of PC to sign-extended 5-bit Immediate.

For jumps (J and JAL):

PC = PC + sign-extend (Imm^{11}).

For LW and SW base-displacement addressing mode is used. The base address in register (Rs) is added to the sign-extended 5-bit immediate to compute the memory address.

Program Execution

.

The program will be loaded and will start at address 0 in the instruction memory. The data segment will be loaded and will start also at address 0 in the data memory. You may also have a stack segment if you want to support procedures. The stack segment can occupy the upper part of the data memory and can grow backwards towards lower addresses. The stack segment can be implemented completely in software. You can dedicate register R6 as the stack pointer. To terminate the execution of a program, ***the last instruction in the program can jump or branch to itself indefinitely*** (because there is no underlying operating system to terminate the program).

Phase 1: Building a Single cycle Processor

It is recommended that you start by building the datapath and control of a single-cycle processor and ensure its correctness. You should have sufficient test cases that ensure the correct execution of ALL instructions in the instruction set. You should also write test cases that show the correct execution of complete programs. To verify the correctness of your design, show the values of all registers (R1 to R7) at the top-level of your design. Provide output pins for registers R1 through R7, and make their values visible at the top level of your design to simplify testing and verification.

Phase 2: Building Pipeline processor

Once you have succeeded in building a single-cycle processor and verified its correctness, design and implement a pipelined version of your design. Make a copy of your single-cycle design, then convert it and implement a pipelined datapath and its control logic. Add pipeline registers between stages. Design the control logic to detect data dependencies among instructions and implement the forwarding logic. You should handle properly

- The data hazards when there are data dependencies between instructions and stall the pipeline when needed.
- **The control hazards** of the branch and jump instructions. For branch and jump instructions, reduce the delay to one cycle only. Stall the pipeline for one clock cycle after a jump or a taken branch instruction. If the branch is not taken, then there is no need to stall the pipeline.
- Also, stall the pipeline after a LW instruction, if it is followed by a dependent instruction.

Bonus:

By integrating a 2-bit dynamic branch predictor into your simulator, you can effectively minimize the frequency of pipeline flushes required for branch and jump instructions. This enhancement leads to a significant reduction in the delay, ultimately achieving a zero-cycle delay for such instructions.

Testing and verification

To demonstrate that your CPU is working, you should do the following:

- Test all components and sub-circuits independently to ensure their correctness. For example, test the correctness of the ALU, the register file, the control logic separately, before putting your components together.
- Test each instruction independently to ensure its correct execution.
- Test sequences of dependent instructions to ensure the correctness of the forwarding
- Write a sequence of instructions to verify the correctness of ALL instructions. Demonstrate the correctness of all ALU R-type and I-type instructions. Demonstrate the correctness of LW and SW instructions. Similarly, you should demonstrate the correctness of all branch and jump instructions
- Test sequences of dependent instructions to ensure the correctness of the forwarding logic. Also, test a LW (load word) followed by a dependent instruction to ensure stalling the pipeline correctly by one clock cycle.
- Test the behavior of taken and untaken branch instructions and their effect on stalling the pipeline.
- Make several copies and versions of your design before making changes, in case you need to go back to an older version.

Test Programs:

- Write a sample program that adds an array of integers. Two procedures are required. The main procedure initializes the array elements with some constant values. It then calls the second procedure after passing the array address and the number of elements as parameters in two registers. The second procedure uses the parameters to compute the sum of the array elements and returns the result in a register. Convert the program into machine instructions by hand and load it into the instruction memory starting at address 0. Also save the image of the instruction and data memories into files and reload them later for testing purposes. **Having two procedures, you will be able to test the JAL and JR instructions.**
- Write additional programs as necessary for further testing, translate them by hand, and save them into files. These files can be loaded into the instruction memory and executed. Their data can be saved as well in files and loaded into the data memory.
- Document all your test programs and files and include them in the report document.

Project Report

The report document must contain sections highlighting the following:

1. Design and Implementation

- Specify clearly the design giving detailed description of the datapath, its components, control, and the implementation details.
- Provide drawings of the component circuits and the overall datapath.
- Provide a description of the control logic and the control signals. **Provide a table giving the control signal values for each instruction. Provide the logic equations for each control signal.**
- Provide a description of the forwarding logic, the cases that were handled, and the cases that stall the pipeline, and the logic that you have implemented to stall the pipeline
- Carry out the design and implementation with the following aspects in mind:
  - Correctness of the individual components
  - Correctness of the overall design when wiring the components together
  - Completeness: all instructions were implemented properly, detecting dependences and forwarding was handled properly, and stalling the pipeline was handled properly for all cases.

2. Simulation and Testing

- Carry out the simulation of the processor developed using Logisim.
- Describe the test programs that you used to test your design with enough comments describing the program, its inputs, and its expected output. List all the instructions that were tested and work correctly. ***List all the instructions that do not run properly.***
- Document all your test programs and files and include them in the report document
- Describe all the cases that you handled involving dependences between instructions, forwarding cases, and cases that stall the pipeline
- Also provide snapshots of the Simulator window with your test program loaded and showing the simulation output results.

3. Teamwork

- Two or at most three students can form a group. Write the names of all the group members on the project report title page.
- Group members are required to coordinate their work among themselves, so that everyone is involved in design, implementation, simulation, and testing.
- Clearly show the work done by each group member using a chart and prepare an execution plan showing the time frame for completing the subtasks of the project. You can also mention how many meetings were conducted between the group members to discuss the design, implementation, and testing.

Submission Guidelines

- ❖ The single-cycle processor design should be completed at Saturday **20-4-2024**.
  - ➢ It should be fully operational. You should have sufficient test cases ready to prove that your CPU is fully functional.
- ❖ The pipelined processor design should be completed at **Saturday 4-5-2024** .
  - ➢ It should be fully operational. You should have sufficient test cases ready to prove that your pipelined CPU is fully functional.
- ❖ If your CPU is not fully operational then identify which instructions do not work properly, or which hazards are not handled properly to avoid the loss of many marks.

❖ All submission will be done through both emails:

- ➢ gaa11@fayoum.edu.eg
- ➢ gna00@Fayoum.edu.eg

❖ The project should be submitted **on the due date by midnight.**

❖ Attach one zip file containing:

- All the design circuits and sub-circuits,
- The test programs, their source code and binary instruction files that you have used to test your design, their test data.
- Project report document
- ***Video demonstrating your work in details*.**

Grading policy

The grade will be divided according to the following components:

- **Correctness: whether your implementation is working**
- **Completeness and testing: whether all instructions and cases have been implemented, handled, and tested properly**
- **Participation and contribution to the project**
- **Report document**
- **Project presentation and discussion**