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

CEE210/CEE216
Computer Engineering
2nd year computer/Commination
Spring 2022

Pipelined MIPS Processor Implementation

Project Objectives:

- Designing a Pipelined 16-bit MIPS-like processor
- Using Logisim simulator to model and test the processor
- Teamwork practice

Instruction Set Architecture

In this project, you will design a simple 16-bit MIPS-like processor with seven 16-bit general-purpose registers: R1 through R7. R0 is hardwired to zero and cannot be written. There is also one special-purpose 12-bit register, which is the program counter (PC). All instructions are 16 bits and there are three instruction formats: R-type, I-type, and J-type as shown below:

R-type format

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

Op^4	Rs^3	Rt^3	Rd^3	$Funct^3$

I-type format

4-bit opcode (Op), 3-bit destination register Rd, 3-bit source register Rs, and 6-bit immediate

Op^4	Rs^3	Rt^3	Imm^6

J-type format :

4-bit opcode (Op) and12-bit Immediate

Op^5	Imm^{12}

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 can specify at most eight functions for a given code. Opcodes 0 and 1 are reserved for R-type instructions.

For I-type instructions. The immediate constant is signed (range is -32 to +31), except for shift and rotate instructions (range is 0 to 61).

For J-type, a 12-bit immediate constant is used for J(Jump), JAL(jump and link) and LUI(Load upper Immediate) instructions.

Instruction Encoding

The instructions, their meaning, and encoding are shown in the following table:

Instruction	Meaning	Encoding				
R-Type						
AND	Reg(Rd) = Reg(Rs) & Reg(Rt)	Op = 0000	Rs	Rt	Rd	f = 000
OR	Reg(Rd) = Reg(Rs) | Reg(Rt)	Op = 0000	Rs	Rt	Rd	f = 001
NOR	Reg(Rd) = ~(Reg(Rs) | Reg(Rt))	Op = 0000	Rs	Rt	Rd	f = 010
XOR	Reg(Rd) = Reg(Rs) ^ Reg(Rt)	Op = 0000	Rs	Rt	Rd	f = 011
SLL	Reg(Rd) = Reg(Rs) << Reg(Rt)	Op = 0000	Rs	Rt	Rd	f = 100
SRL	Reg(Rd) = Reg(Rs) zero>> Reg(Rt)	Op = 0000	Rs	Rt	Rd	f = 101
SRA	Reg(Rd) = Reg(Rs) sign>> Reg(Rt)	Op = 0000	Rs	Rt	Rd	f = 110
ROL	Reg(Rd) = Reg(Rs) rotate<< Reg(Rt)	Op = 0000	Rs	Rt	Rd	f = 111
ADD	Reg(Rd) = Reg(Rs) + Reg(Rt)	Op = 0001	Rs	Rt	Rd	f = 000
SUB	Reg(Rd) = Reg(Rs) – Reg(Rt)	Op = 0001	Rs	Rt	Rd	f = 001
SLT	Reg(Rd) = Reg(Rs) signed< Reg(Rt)	Op = 0001	Rs	Rt	Rd	f = 010
SLTU	Reg(Rd) = Reg(Rs) unsigned<Reg(Rt)	Op = 0001	Rs	Rt	Rd	f = 011
SLTE	Reg(Rd) = Reg(Rs) signed≤ Reg(Rt)	Op = 0001	Rs	Rt	Rd	f = 100
SLTEU	Reg(Rd) = Reg(Rs) unsigned≤ Reg(Rt)	Op = 0001	Rs	Rt	Rd	f = 101
JR	PC = lower 12 bits of Reg(Rs)	Op = 0001	Rs	000	000	f = 110
JALR	Rd=PC+1, PC=Rs	Op = 0001	Rs	000	Rd	f = 111
I-Type						
LW	Reg(Rt) = Mem(Reg(Rs) + ext(im^{6}))	Op = 0010	Rs	Rt	Immediate^{6}	
SW	Mem(Reg(Rs) + ext(im^{6})) = Reg(Rt)	Op = 0011	Rs	Rt	Immediate^{6}	
ANDI	Reg(Rt) = Reg(Rs) & ext(im6)	Op = 0110	Rs	Rt	Immediate^{6}	
ORI	Reg(Rt) = Reg(Rs) | ext(im6)	Op = 0111	Rs	Rt	Immediate^{6}	
ADDI	Reg(Rt) = Reg(Rs) + ext(im^{6})	Op = 1000	Rs	Rt	Immediate^{6}	
BEQ	Branch if (Reg(Rs) == Reg(Rt))	Op = 0100	Rs	Rt	Immediate^{6}	
BNE	Branch if (Reg(Rs) != Reg(Rt))	Op = 0101	Rs	Rt	Immediate^{6}	
BLT	Branch if ((Reg(Rs) < Reg(Rt))	OP = 1101	Rs	Rt	Immediate^{6}	
BGT	Branch if ((Reg(Rs) > Reg(Rt))	OP=1111	Rs	Rt	Immediate^{6}	
BLTZ	Branch if (Reg(Rs) < 0)	Op = 1100	Rs	0	Immediate^{6}	
BLEZ	Branch if (Reg(Rs) ≤ 0)	Op = 1100	Rs	1	Immediate^{6}	
BGTZ	Branch if (Reg(Rs) > 0)	Op = 1110	Rs	0	Immediate^{6}	
BGEZ	Branch if (Reg(Rs) ≥ 0)	Op = 1110	Rs	1	Immediate^{6}	
J-Type						
J	PC = Immediate^{12}	Op = 1001	Immediate^{12}			
JAL	R7 = PC + 1, PC = Immediate^{12}	Op = 1011	Immediate^{12}			
LUI	R1 = Immediate^{12} << 4	Op = 1010	Immediate^{12}			

Instruction Description

There are three shift and one rotate instructions. For shift and rotate instructions, the least significant 4 bits of register Rt are used as the shift/rotate amount. The Load Upper Immediate (LUI) is of the J-type to have a 12-bit immediate constant loaded into the upper 12 bits of register R1. The LUI can be combined with ORI (or ADDI) to load any 16-bit constant into a register. Although the instruction set is reduced, it is still rich enough to write useful programs. We can have procedure calls and returns using the JAL, JALR and JR instructions.

Opcodes 2 and 3 define the load word (LW) and store word (SW) instructions. These two Instructions address 32-bit words in memory. Displacement addressing is used. The effective memory address = Reg(Rs)+ sign_extend(Imm6). Register Rt is a destination register for LW, but a source for SW. Loading/storing a byte or half word are not defined to simplify the project.

Memory

Your processor will have separate instruction and data memories with $2^{12} = 4096$ 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 (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.

Addressing Modes

- For branch instructions PC-relative addressing mode is used:

  PC = PC + sign-extend (immediate6).

- For jump instructions (J and JAL), direct addressing is used:
  PC = Immediate^{12}.
- For LW and SW instructions, base- addressing mode is used. The base address in register Rs contains the memory address.

Register File

Implement a Register file containing eight 16-bit registers R1 to R7 with two read ports and one write port. R0 is hardwired to zero.R0 cannot be written

Arithmetic and Logical Unit (ALU)

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

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

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 control hazards of the branch and jump instructions. Also, stall the pipeline after a LW instruction, if it is followed by a dependent instruction.

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.
- ❖ 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 short programs and loops to verify the correctness of a sequence of instructions. Make sure that your pipelined CPU can handle data hazards and control hazards properly.
- ❖ Write a procedure of your choice. Write a main function to call the procedure.
- ❖ Translate test programs into machine instructions. Load the instructions 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.
- ❖ 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*

- Provide drawings of the various components and the overall datapath.
- Provide a complete description of the control logic and the control signals. **Provide a table giving the control signal values for each instruction.**
- Provide a complete description of the forwarding logic, the cases that were handled, and the logic that you have implemented to handle the control hazards stall the pipeline

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*

- Three or at most four 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.
- Show the work done by each group member using a chart.

Submission Guidelines

❖ The single-cycle processor design should be completed at **26-4-2022**.

➢ It should be fully operational. You should have sufficient test cases ready to prove that your CPU is fully functional. **You will be evaluated by your TA.**

➢

❖ The pipelined processor design should be completed at **10-5-2021**.

➢ 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, as well as the 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**

■ **Report document**