KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
COLLEGE OF COMPUTER SCIENCES & ENGINEERING

COE 301 Computer Organization

Project -Term 182

Pipelined Processor Design
Due Dates:
Single Cycle – Week 13
Pipelined Processor – Week 15

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 register numbers (Rs, Rt, and Rd), 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 register numbers (Rs and Rt), and 6-bit immediate constant

Op^4	Rs^3	Rt^3	$Immediate^6$

J-type format

4-bit opcode (Op) and 12-bit immediate constant

Op^4	$Immediate^{12}$

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 opcode. Opcodes 0 and 1 are reserved 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 6 bits because of the fixedsize nature of the instruction. The 6-bit immediate constant is assumed to be sign-extended for all instructions.

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

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

Instr	Meaning	Encoding				
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
XNOR	Reg(Rd) = Reg(Rs) ~^ Reg(Rt)	Op = 0000	Rs	Rt	Rd	f = 100
SLL	Reg(Rd) = Reg(Rs) << Reg(Rt)	Op = 0000	Rs	Rt	Rd	f = 101
SRL	Reg(Rd) = Reg(Rs) zero>> Reg(Rt)	Op = 0000	Rs	Rt	Rd	f = 110
SRA	Reg(Rd) = Reg(Rs) sign>> 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
MOVZ	Reg(Rd) = Reg(Rs) if (Reg(Rt)==0)	Op = 0001	Rs	Rt	Rd	f = 100
MOVN	Reg(Rd) = Reg(Rs) if (Reg(Rt)!=0)	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
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 = 0100	Rs	Rt	Immediate^6	
ORI	Reg(Rt) = Reg(Rs) | ext(im6)	Op = 0101	Rs	Rt	Immediate^6	
ADDI	Reg(Rt) = Reg(Rs) + ext(im^6)	Op = 0110	Rs	Rt	Immediate^6	
BEQ	Branch if (Reg(Rs) == Reg(Rt))	Op = 0111	Rs	Rt	Immediate^6	
BNE	Branch if (Reg(Rs) != Reg(Rt))	Op = 1000	Rs	Rt	Immediate^6	
BLT	Branch if (Reg(Rs) signed < Reg(Rt))	Op = 1001	Rs	Rt	Immediate^6	
BLE	Branch if (Reg(Rs) signed ≤ Reg(Rt))	Op = 1010	Rs	Rt	Immediate^6	
BGT	Branch if (Reg(Rs) signed > Reg(Rt))	Op = 1011	Rs	Rt	Immediate^6	
BGE	Branch if (Reg(Rs) signed ≥ Reg(Rt))	Op = 1100	Rs	Rt	Immediate^6	
J	PC = Immediate^{12}	Op = 1101	Immediate^{12}			
JAL	R7 = PC + 1, PC = Immediate^{12}	Op = 1110	Immediate^{12}			
LUI	R1 = Immediate^{12} << 4	Op = 1111	Immediate^{12}			

There are three shift instructions. For shift instructions, the least significant 4 bits of register Rt are used as the shift 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.

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 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.

Addressing Modes

For branch instructions (BEQ, BNE, BLT, BLE, BGT and BGE), PC-relative addressing mode is used: PC = PC + sign-extend(immediate^6). 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.

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.

To terminate the execution of a program, the last instruction in the program can jump or branch to itself indefinitely.

Building a Pipelined Processor

Design 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, hazard detection and stall unit.

Design Alternatives

When designing the datapath and control unit, explore alternative design options and justify why a given design alternative is chosen. For example, when designing the control unit consider implementing it using a decoder and a set of OR/NOR gates vs. using a ROM to store the control signals vs. optimizing the equation of each control signal separately. When designing the ALU and the shifter unit, consider alternative designs and justify why a design alternative is chosen. The same should be applied for all design decisions in your CPU.

Testing

To test the correct functionality of your designed CPU, you need to implement the following programs:

1. A program the counts the number of 1's in a register,

2. The **bubble sort procedure** given in class. Use this procedure to sort an array of 8 words of your choice,
3. A program that tests each of the remaining untested instructions to demonstrate their correct operation.

WARNING

Although Logisim is stable, it might crash from time to time. Therefore, it is best to save your work often. Make several copies and versions of your design before making changes, in case you need to go back to an older version.

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 (highlighting the design choices you made and why, and any notable features that your processor might have.) Document clearly design alternatives explored and why a given design is selected.
- Provide drawings of the component circuits 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 the logic equations for each control signal.
- Provide a complete 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.
- Provide list of sources for any parts of your design that are not entirely yours (if any).
- 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.
- Test each of the components individually and demonstrated its correct operation including the ALU and register file.
- Describe all the features of the simulator used for simulating your design with a clear emphasis on its advantages and limitations (if any) for simulating the design, list the known bugs or missing features (if any).
- 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.
- Describe all the cases that you handled involving dependencies 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

- This project is a team work project with a maximum of four students. Make sure to write the names of all the group members on the project report title page.
- Each group should assign a group leader that leads the conduction of the project, divided the project tasks among the team members.
- Project tasks should be divided among the group members so that each group member contributes equally in the project and everyone is involved in all the following activities:
    - Design and Implementation
    - Simulation and Testing
    - Design and results reporting
- 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.
- Students who **help** other team members should mention that to earn credit for that.

Submission Guidelines

All submissions will be done through Blackboard.

Attach one zip file containing all the design circuits, the test programs source code and binary instruction files that you have used to test your design, their test data, as well as the report document.

Submit also a hard copy of the report to the lab instructor.

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. It should be noted that being in a group that implemented the project correctly does not qualify you to get full mark. Your mark will depend on your contribution in the project.
- Report organization and clarity.

Late policy:

The project should be submitted on the due date. Late projects are accepted, but will be penalized 5% for each late day and for a maximum of 5 late days (or 25%). Projects submitted after 5 late days will not be accepted.