COE 308 – Computer Architecture

Term 081 – Fall 2008

Project 2: Pipelined Processor Implementation

Objectives:

- Using the LogiSim simulator
- Designing and testing a Pipelined 16-bit processor
- Teamwork

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, so we are left with seven registers. There is also one special-purpose 16-bit register, which is the program counter (PC). All instructions are 16 bits. 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 number (Rs and Rt), and 6-bit signed 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. We will reserve opcode 0 for R-type instructions. It is also possible to reserve more opcodes, if more R-type instructions exist.

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 fixed-size nature of the instruction. The size of the immediate constant is suitable for our uses. The 6-bit immediate constant is signed (and sign-extended) for all I-type 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

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

Instr	Meaning	Encoding				
SLL	Reg(Rd) = Reg(Rs) << Reg(Rt)	Op = 0000	Rs	Rt	Rd	f = 000
ROL	Reg(Rd) = Reg(Rs) rotate<< Reg(Rt)	Op = 0000	Rs	Rt	Rd	f = 001
SRL	Reg(Rd) = Reg(Rs) zero>> Reg(Rt)	Op = 0000	Rs	Rt	Rd	f = 010
SRA	Reg(Rd) = Reg(Rs) sign>> Reg(Rt)	Op = 0000	Rs	Rt	Rd	f = 011
AND	Reg(Rd) = Reg(Rs) & Reg(Rt)	Op = 0000	Rs	Rt	Rd	f = 100
OR	Reg(Rd) = Reg(Rs) | Reg(Rt)	Op = 0000	Rs	Rt	Rd	f = 101
NOR	Reg(Rd) = ~(Reg(Rs) | Reg(Rt))	Op = 0000	Rs	Rt	Rd	f = 110
XOR	Reg(Rd) = Reg(Rs) ^ 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
JR	PC = lower 12 bits of Reg(Rs)	Op = 0001	Rs	000	000	f = 111
ANDI	Reg(Rt) = Reg(Rs) & ext(im_6)	Op = 0100	Rs	Rt	Immediate^6	
ORI	Reg(Rt) = Reg(Rs) | ext(im_6)	Op = 0101	Rs	Rt	Immediate^6	
ADDI	Reg(Rt) = Reg(Rs) + ext(im^6)	Op = 1000	Rs	Rt	Immediate^6	
SLTI	Reg(Rt) = Reg(Rs) signed< ext(im_6)	Op = 1010	Rs	Rt	Immediate^6	
LW	Reg(Rt) = Mem(Reg(Rs) + ext(im^6))	Op = 0110	Rs	Rt	Immediate^6	
SW	Mem(Reg(Rs) + ext(im^6)) = Reg(Rt)	Op = 0111	Rs	Rt	Immediate^6	
BEQ	Branch if (Reg(Rs) == Reg(Rt))	Op = 1001	Rs	Rt	Immediate^6	
BNE	Branch if (Reg(Rs) != Reg(Rt))	Op = 1011	Rs	Rt	Immediate^6	
J	PC = Immediate^{12}	Op = 1100	Immediate^{12}			
JAL	R7 = PC + 1, PC = Immediate^{12}	Op = 1101	Immediate^{12}			
LUI	R1 = Immediate^{12} << 4	Op = 1111	Immediate^{12}			

Opcodes 0 and 1 are used for R-type instructions. There are three shift and one rotate instruction. To shift or rotate, use the least significant 4 bits of register Rt 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 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 and JR instructions.

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:

SLL, ROL, SRL, SRA, AND, OR, NOR, XOR, ADD, SUB, SLT, SLTU

Addressing Modes

PC-relative addressing mode is used for branch and jump instructions.
For branching (BEQ, BNE), the branch target address is computed as follows:
PC = PC + sign-extend(Imm6), by adding the contents of PC to sign-extended 6-bit Immediate.

For jumps (J and JAL): PC = PC + sign-extend(Imm12).

For LW and SW base-displacement addressing mode is used. The base address in register Rs is added to the sign-extended 6-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.
To terminate the execution of a program, the last instruction in the program can jump or branch to itself indefinitely.

Getting Started with Logisim

You should first download Logisim from the COE 308 course website or Logisim website http://ozark.hendrix.edu/~burch/logisim/. Logisim is very easy to use. To get started, you can read the documentation available under the Logisim website/ Course WebCT.

Building a Pipelined Processor

Design and implement a pipelined-datapath and its control logic. A five-stage pipeline should be constructed similar to the pipeline presented in the class lectures. Add pipeline registers between stages. Design the control logic to detect data dependencies among instructions and implement the forwarding logic. 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.

Testing

- 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 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.
- 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. 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.

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. There are also some known bugs. For example, 3-input or higher-input XOR or XNOR gates do not function properly when more than 2 inputs are equal to logic '1', so do not use them. However, 2-input XOR and XNOR gates function properly. Make sure you test your sub-circuits with enough test cases before using them.

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.)
- 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.
- 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 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

- As in the first project, two or at most three students can form a group. It is best to continue with the same group. Make sure to write the names of all the group members on the project report title page.
- Group members are required to coordinate the work equally among themselves so that everyone is involved in all the following activities:
  - Design and 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.
- Students who **help** other team members should mention that to earn credit for that.

Submission Guidelines

All submissions will be done through WebCT.

Attach one zip file containing all the design circuits, the 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 during the class lecture.

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

Late policy

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