COE 308 – Computer Architecture
Term 061 – Fall 2006

Project 2: Single Cycle Processor Design
Due Sunday, December 10, 2006 by Midnight

Objectives:

- Using the Logisim simulator
- Designing and testing a Single-Cycle 16-bit processor
- Teamwork

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, 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 also 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 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 than eight 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:

Eight R-type instructions, six I-type instructions, and three J-type instructions are defined. These instructions, their meanings, and their encodings are shown below:

Instr	Meaning	Encoding				
OR	Reg(Rd) = Reg(Rs) | Reg(Rt)	Op = 0000	Rs	Rt	Rd	f = 000
AND	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
ADD	Reg(Rd) = Reg(Rs) + Reg(Rt)	Op = 0000	Rs	Rt	Rd	f = 100
SUB	Reg(Rd) = Reg(Rs) – Reg(Rt)	Op = 0000	Rs	Rt	Rd	f = 101
SLT	Reg(Rd) = Reg(Rs) < Reg(Rt)	Op = 0000	Rs	Rt	Rd	f = 110
JR	Jump register: PC = Reg(Rs)	Op = 0000	Rs	000	000	f = 111
ADDI	Reg(Rt) = Reg(Rs) + ext(im^6)	Op = 0100	Rs	Rt	Immediate^6	
SLTI	Reg(Rt) = Reg(Rs) < ext(im^6)	Op = 0110	Rs	Rt	Immediate^6	
LW	Reg(Rt) = Mem(Reg(Rs) + ext(im^6))	Op = 1000	Rs	Rt	Immediate^6	
SW	Mem(Reg(Rs) + ext(im^6)) = Reg(Rt)	Op = 1001	Rs	Rt	Immediate^6	
BEQ	Branch if (Reg(Rs) == Reg(Rt))	Op = 1010	Rs	Rt	Immediate^6	
BNE	Branch if (Reg(Rs) != Reg(Rt))	Op = 1011	Rs	Rt	Immediate^6	
J	PC = PC + 1 + ext(im^{12})	Op = 1100	Immediate^{12}			
JAL	R7 = PC + 1, PC = PC+1+ext(im^{12})	Op = 1101	Immediate^{12}			
LUI	R1 = Immediate^{12} << 4	Op = 1111	Immediate^{12}			

Although the instruction set is too much reduced, it is still rich enough to write some 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 $2^{12} = 4096$ words each (this is the maximum that can be supported under the current version of Logisim). 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 branches (BEQ and BNE) and jumps (J and JAL), PC-relative addressing mode is used. PC = PC + 1 + sign-extend(imm^6) for branches and PC = PC + 1 + sign-extend(imm^{12}) for jumps. For LW and SW base-displacement addressing mode is used. The base address in register Rs is added to the sign-extended immediate^6 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 my COE 308 course website. Logisim is very easy to use. To get started, you can read the documentation available under the Logisim website.

Building a Single-Cycle Processor

Build a single-cycle datapath and its control logic. You can test this part by loading a program into the instruction memory and loading its data into the data memory.

Testing:

To test the implementation, write a simple program that adds 5 array elements. Two procedures are required. The main procedure initializes the 5 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 in the program, you will be able to test the JAL and JR instructions. You can also write other programs and additional code as necessary to test all the instructions that you have implemented.

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.

Groups:

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 students involved in your group on the project report.

Submission Guidelines:

All submissions will be by email sent to:

mudawar@ccse.kfupm.edu.sa ; rfarooqi@ccse.kfupm.edu.sa

Subject: COE 308 Project 2

Attach one zip file containing the design circuits, the binary image files of the sample programs that you have used to test your design, as well as the report document.

Report:

The report should contain the names of all the group members and the collaboration efforts among the group members (how the group members collaborated and divided the work

among themselves). It should contain the circuit diagrams and a description of the circuit design. You should describe the sample code that was used to test your design in assembly language and in binary. Make sure to demonstrate the instructions that were implemented correctly and to identify the instructions that were not implemented or do not function properly.

Grading policy:

The grade will be divided according to the following components:

- Correctness: whether your implementation is working
- Completeness and testing: whether all instructions 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.