COMP 303 MIPS Processor Design
Project 3: Simple Execution Loop

Due date: November 20, 23:59

Overview:

In the first three projects for COMP 303, you will design and implement a subset of the MIPS32 architecture in Logisim, a software logic simulator. The goal of these projects is to move you from designing small special-purpose circuits, such as those you designed as homework exercises, to building complex, general-purpose CPUs. We will be implementing a basic 32-bit MIPS processor. We will ignore more advanced features such as the MIPS coprocessor instructions and traps/exceptions. Read this document entirely, paying special attention to the section “Deviation from the MIPS standard”.

We will be using Logisim, a free hardware design and circuit simulation tool. We have implemented a library of components to help you, and Logisim comes with libraries containing basic gates, memory chips, multiplexors and decoders, and other simple components. You may use any of these basic components in your designs, but ***you may not use the Logisim arithmetic library*** anywhere in your design. If you wish to download additional libraries from the web to use in your design, please check with the course staff for approval.

YOU HAVE TO USE THE ALU YOU HAVE IMPLEMENTED FOR PROJECT2. EACH GROUP SHOULD USE THEIR OWN ALU IN ORDER TO GET POINTS FROM THIS PROJECT. OTHERWISE THE COPY REGULATIONS WILL BE APPLIED.

Project 3: **Simple Execution Loop**

Here we will implement the basic execution loop for the processor, but only implementing two flow control instructions (unconditional jumps), and a few arithmetic and logic instructions. Your design should consist of a program counter, a MIPS Program ROM to store the compiled assembly code of the program, a 32x32 register file, your 32-bit ALU, and other components, along with the instruction decode logic and control circuits needed to connect them together.

The design will be non-pipelined, so that in each clock cycle the processor fetches and executes a complete instruction. For the arithmetic instructions, this includes: fetching the instruction to execute from the program ROM, decoding the instruction, selecting arguments from the register file, computing a result, storing the result in the register file, and incrementing the program counter by 4. For *jump* instructions, no value is written to the register file, and the program counter is set to a value selected from the register file or from the immediate field of the instruction.

Begin by connecting a register file, your ALU, and enough circuitry to implement a few arithmetic instructions. Once this is working, you can add logic to handle the jump instructions. For this project, your processor must implement the following MIPS instructions: **J, JR, ADDU, SUBU, AND, OR, XOR, NOR, ANDI, LUI.**

There is a strange quirk of the MIPS architecture. The PC-relative branch targets for control flow instructions do not refer to the address of the current instruction; instead they refer to the address of the instruction immediately following the current one. [You will see later on that this is because a high-performance implementation of the MIPS is “pipelined,” and the pipeline structure means that the PC has already been updated to point to the next instruction by the time the branch target is computed. You may ignore this comment for now and do not need to understand pipelining, as long as you accept that, by convention, the branch target address is computed from the address of the instruction following the current one. Also, the real MIPS has a “branch delay slot,” which you should ignore for this project. Your processor will not have a branch delay slot]

We have provided some very simple test programs, but they do not thoroughly test the processor. Write a test program that more fully tests all of the features you have implemented in this project. This is a critical step, and you should spend a fair amount of time developing a comprehensive set of test programs that exercise all of the different parts of your processor.

Components you will need: The cs316 library contains a triple-ported (dual read / single write) 32-bit wide by 32-registers deep register file, and a *Program* memory chip capable of loading a MIPS assembly language file directly into its memory. Logisim's *memory* library contains registers you can use for your program counter.

Hint: The instruction decode logic in this project should be extremely simple – do not waste time implementing an efficient decoder, because you will have to build a new decoder in project 3 anyway.

What to submit:

Submit a single Logisim project file containing your processor, and a text file containing the problems in your implementations. For instance, a sample text file would be: “In my implementation, xor and ori do not work but others have no problem”.

Help and Hints:

Ask the TAs for help. We expect to see most students in office hours during the course of the project.

Do a pencil and paper design for all components before implementing in Logisim.

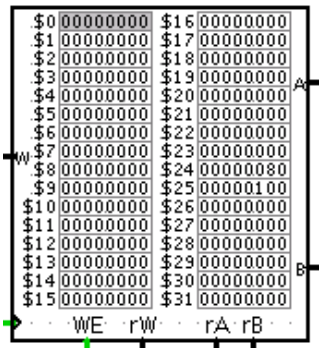
Deviation from the MIPS Standard:

You can ignore any MIPS instruction or feature not mentioned in this document, and can assume that your processor will never encounter anything but legal instructions from the list above. In addition:

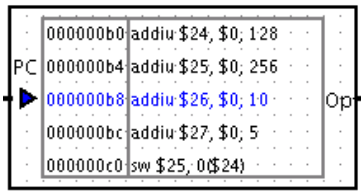
- Assume all jumps will be word aligned and to valid ROM addresses.
- Assume traps or exceptions will not occur (i.e., do not implement them).
- Branch and jumps take effect immediately, rather than after a one-instruction delay slot (although we still compute the destination *PC* exactly as specified in the MIPS standard).

Loading the cs316 library: Select *project→Load library→Jar Library.* Select *cs316.jar*, and enter *edu.cornell.cs316.Components* as the class name. This library contains the following circuits:

Register File: A 32-bit wide by 32-registers deep register file. Register $0 is hard-wired to zero at all times, and writes to $0 are ignored. Inputs *rA* and *rB* are used to select register values to output on the *A* and *B* outputs. On the rising edge of the clock, if *WE*is high the data value at input *D* is stored in register *rW*.



Program ROM: A 32-bit wide byte-addressed ROM, with built-in MIPS assembler. Use the attributes panel to load a MIPS assembly file into the ROM. The *PC* input specifies the address of the current instruction, and must be a multiple of 4. The output is the 32-bit instruction at the *PC* address, or zero if the *PC* has gone past the end of the ROM. Note that *zero* also happens to encode a no-op in MIPS. The assembly language is in the format described below.



MIPS (subset) assembly syntax:

The Program ROM component understands all of the instructions you will implement. The syntax is standard MIPS syntax. Labels are case sensitive, everything else ignores case. Anything following a '#' is a comment. In this project, you will only use a few of the instructions listed below.

Labels are used to mark places in the code. Jumps and branches can refer to these labels (for jumps, the address is inserted directly, for branches, a relative offset from the branch instruction is computed and inserted, accounting for the delay slot).

The instruction syntax is the same as given in the MIPS standard (and different from the output of gcc and many other tools). Registers are written as $0, $1, ..., $31, and the destination register goes on the left, followed by source registers and immediate values on the right. Absolute addresses (for J) are given in hex (i.e., 0x12ab), and all other integers can be in hex or signed decimal (i.e., 0x12ab or 1234 or -1234). The special constant *PC* can be used anywhere an integer is needed, and the assembler will replace it by the address of the instruction itself. The *PC* will normally fit in 15 bits or less

Some examples of instructions are:

Jumps	J 0x24 J my_label JR $31
Branches	BEQ $5, $6, -12 BEQ $5, $6, my_loop_top BLEZ $9, 16 BLEZ $9, my_loop_done
Memory Load/Store	LW $12, -4($30) SW $12, 0($30)
Immediate load	LUI $14, 0x123
Immediate arithmetic	ADDIU $12, $0, 0x1234
Register arithmetic	ADDU $13, $0, $20
Shifts	SLL $13, $13, 2 SLLV $15, $14, $3
Labels	my_loop_top: BNE $5, $6, 16 my_loop_done:

MIPS (subset) opcode summary (from the MIPS handbook):

Table A-2 MIPS32 Encoding of the Opcode Field

opcode		*bits 28..26*							
		0	1	2	3	4	5	6	7
bits 31..29		000	001	010	011	100	101	110	111
0	000	*SPECIAL* δ	*REGIMM* δ	J	JAL	BEQ	BNE	BLEZ	BGTZ
1	001	ADDI	ADDIU	SLTI	SLTIU	ANDI	ORI	XORI	LUI
2	010	*COP0* δ	*COP1* δ	*COP2* θδ	*COP3* θδ	BEQL ϕ	BNEL ϕ	BLEZL ϕ	BGTZL ϕ
3	011	β	β	β	β	*SPECIAL2* δ	JALX ε	ε	*
4	100	LB	LH	LWL	LW	LBU	LHU	LWR	β
5	101	SB	SH	SWL	SW	β	β	SWR	CACHE
6	110	LL	LWC1	LWC2 θ	PREF	β	LDC1	LDC2 θ	β
7	111	SC	SWC1	SWC2 θ	*	β	SDC1	SDC2 θ	β

Table A-3 MIPS32 *SPECIAL* Opcode Encoding of Function Field

function		*bits 2..0*							
		0	1	2	3	4	5	6	7
bits 5..3		000	001	010	011	100	101	110	111
0	000	SLL	*MOVCI* δ	SRL	SRA	SLLV	*	SRLV	SRAV
1	001	JR	JALR	MOVZ	MOVN	SYSCALL	BREAK	*	SYNC
2	010	MFHI	MTHI	MFLO	MTLO	β	*	β	β
3	011	MULT	MULTU	DIV	DIVU	β	β	β	β
4	100	ADD	ADDU	SUB	SUBU	AND	OR	XOR	NOR
5	101	*	*	SLT	SLTU	β	β	β	β
6	110	TGE	TGEU	TLT	TLTU	TEQ	*	TNE	*
7	111	β	*	β	β	β	*	β	β

Is optimal always best? It is not enough to simply build a working circuit. We want you to build a *good* working circuit. But we leave it up to you to define what

good is, and we expect you to document your criteria and how you fulfill them. For example, you might aim to make the fastest processor possible, at the expense of simplicity and number of gates. Or you might aim for a minimalist design using the fewest gates possible, even if it makes your processor slower. Eventually, in Project 3, you will be asked to document your goals and justify the choices you made. All of your designs should be clear and easy to follow, and should be annotated (with text labels) for any unusual or difficult parts of the circuit.

Academic Integrity. As one of the most widely studied architectures, MIPS has a wealth of information available on the web and in textbooks. You may consult any of the MIPS architecture documentation available to you in order to learn about the instruction set, what each instruction does, etc. But we expect your design to be entirely your own. If you are unsure if it is okay to borrow from some other source, just ask the TAs, and give credit in your final writeup. If you are unsure about asking the TAs, then it is probably not okay. Plagiarism in any form will not be tolerated.