COMP 303 MIPS Processor Design
Project 3: MIPS Processor

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.

Project 3: MIPS Processor

In this project you will complete the design of your MIPS processor by adding most of the remaining MIPS instructions. Your basic execution loop from the previous project should contain most of the major components, with the exception of RAM for the load/store instructions. For this project, we will use a split-memory design: The Program ROM will store a read-only copy of the instructions, and a separate Logisim RAM will be used to store data for the program's execution.

Begin by making a pencil-and-paper design showing all of the components you will need, and create a list of all control signals in the processor. You will not submit this paper design, but the TAs will demand to see it if you ever have a question about anything. Next, decide for each instruction what the value of each control signal must be, and use this information to design the instruction decode logic. Lastly, implement your decoder and the rest of the processor in Logisim, and test your circuit with test programs of your own making.

Your design must implementation the following subset of the MIPS architecture:

Jumps (without a delay slot)	J, JR
Branches (without a delay slot)	BEQ, BNE, BLEZ, BGTZ
Memory Load/Store	LW, SW
Immediate load	LUI
Immediate arithmetic	ADDIU, ANDI, ORI, XORI
Register arithmetic	ADDU, SUBU, AND, OR, XOR, NOR
Shifts (constant and variable)	SLL, SRL, SRA, SLLV, SRLV, SRAV

Finally, you must *document* your design in a short, 1 – 3 page paper. Include:
*   A short overview of your processor design, highlighting the design choices you made and why, and any notable features of your processor might have.
*   List of sources for any parts of your design that are not entirely yours, with the exception of built-in Logisim and cs316 library components (if any)
*   List of known bugs or missing features (if any)
*   An estimate of the number of gates used in your processor, with a breakdown of the number of gates for each major component (you can ignore the memory, program ROM, and register file).
*   A description of the *longest path* in your design (assume that the register file takes about 4 gate delays to perform a read, and that the MIPS program ROM and memory each take about 10 gate delays to do a read or write).

What to submit: A single Logisim project file containing your processor, a test program with comments describing the expected output, a text file containing your pencil-and-paper decode logic design (table giving the value of each control signal for every instruction, and formulas, diagrams, or descriptions of the logic for each), and a 1-3 page design document as described above.

For the Adventurous:

Note: These suggestions for an extra challenge will be examined (and commented on, if your project works well) but not graded. They will have no impact on the class grades. They are here to provide some direction to those who finish their assignments early and are looking for a way to impress friends and family.

- Implement the jump-and-link instructions JAL, JALR (barely challenging).
- Implement the delay slot for branches/jumps (moderately challenging).
- Implement multiply (MULT and/or MULTU) (moderately challenging).
    - *Note:* increasing the number of delay slots after MULT makes this almost easy.
- Implement pipelining (more challenging).

Help and Hints:

Ask the TAs for help. We expect to see most students in office hours during the course of the project. Extra hours will be scheduled as needed.

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 table above. In addition:
- Assume loads and stores will be word aligned and to valid RAM addresses.
- 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).
- Do not implement floating point, multiply/divide, or the HI and LO registers.
- Use a split-memory design (separate program and data memories).
- 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).

Logisim and Library Guide:

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

Loading the Memory library: Select *project→Load library→Built-in.* Select *Memory*. This library contains Logisim's built-in memory RAM chip, which is fully described in Logisim's help files. Below is a short description:

Logisim RAM: Configurable to 32-bit wide word-addressed RAM, with up to 12-bits of address (16KB storage). The Logisim help describes this chip. Since it is word-addressed, be sure to drop the lowest 2 bits of the address, in addition to any extra high order bits, and use a tri-state buffer for driving the bidirectional D pin. The following shows a typical circuit:
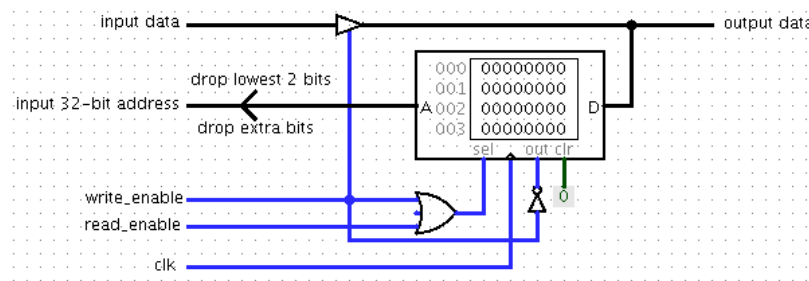
MIPS (subset) assembly syntax:

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.

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, PC
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

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.