Reduced Instruction Set Computer (RISC) Processor

6.111 - 2018

Bradley Jomard
Quinn Magendanz

Project Description

This project focuses on implementing a basic processor that can be used to run simple applications or a file system. We will be basing our design off of the MIT 6.004 Beta (or the RISC-V) since the implementation is open source and a large suit of test cases already exists for debugging and verification.

Implementation

Arithmetic Logic Unit (ALU)

The ALU performs basic logic computations on two 32-bit inputs. As a base, we will offer comparison, addition, subtraction, AND, OR, XOR, XNOR, left shift, right shift, and right shift with sign extension. These operations will be selected via a 6-bit selector input.

Control Logic

This memory unit stores the processor state corresponding to specific instructions. The control logic takes in a 6-bit opcode which represents an instruction, and maps it to a series of outputs that feed into the ALU, register unit, and other parts of the processor to perform the operation specified by the instruction.

Register File

The register file will control access to a set of thirty two 32-bit registers. Two inputs control register selection. A set of additional control inputs specify whether the current operation is a read or a write.

Instruction Memory

The instruction memory contains the set of instructions which define the program being run by the processor. It is organized into 32-bit lines representing one instruction. As an input, the instruction memory takes in a 32-bit instruction pointer which tracks the current line to be executed.

Program Counter

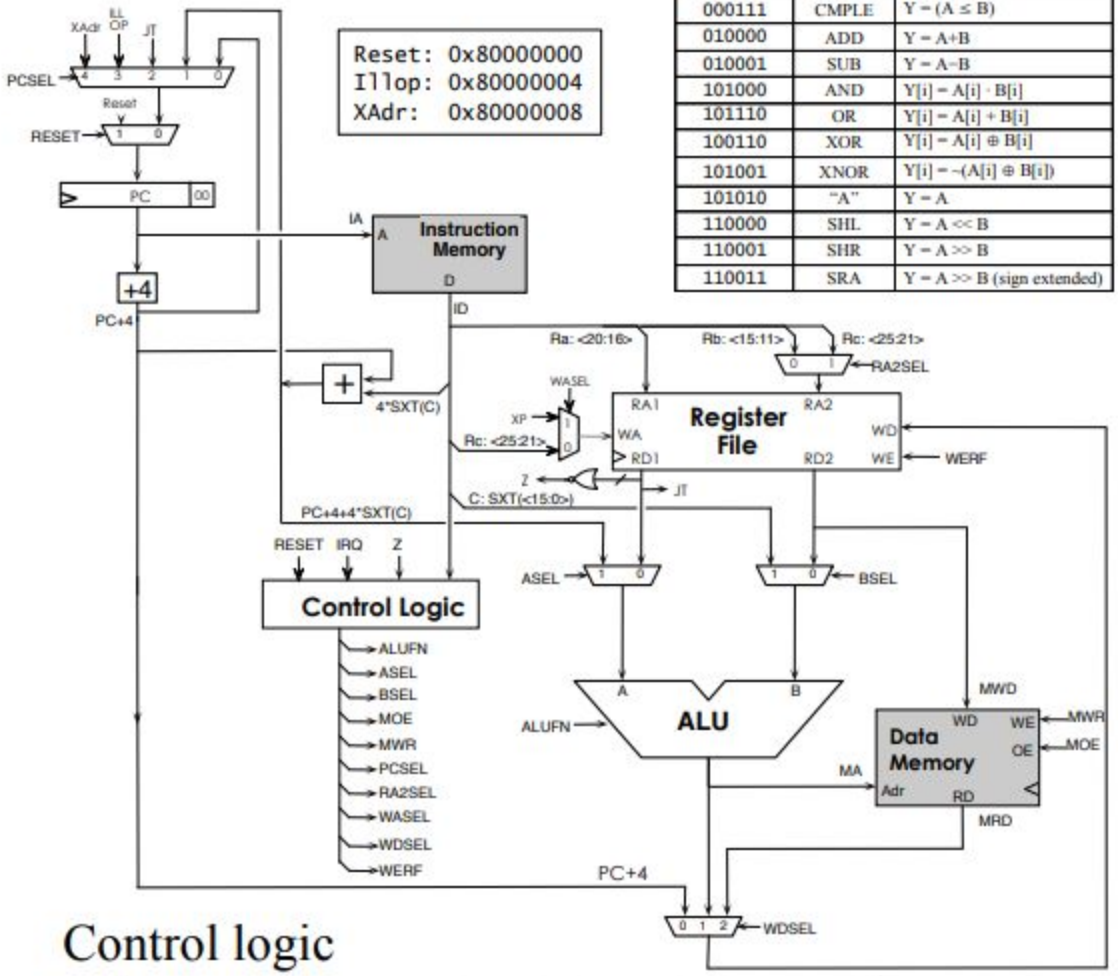
The program counter maintains the current state of execution of the current program. Usually, the program counter will read the current op-code and increment the counter to the next instruction. If the current instruction specifies a jump or addition/subtraction to the current instruction pointer, the program counter will move to the new instruction location.

Data Memory

Data memory will be a section of DRAM which the processor can read and write to by 32-bit chunks.

Unpipelined Beta

ALUFN[5:0]	Operation	Output value Y[31:0]
000011	CMPEQ	Y = (A == B)
000101	CMPLT	Y = (A < B)
000111	CMPLE	Y = (A ≤ B)
010000	ADD	Y = A+B
010001	SUB	Y = A−B
101000	AND	Y[i] = A[i] · B[i]
101110	OR	Y[i] = A[i] + B[i]
100110	XOR	Y[i] = A[i] ⊕ B[i]
101001	XNOR	Y[i] = ¬(A[i] ⊕ B[i])
101010	"A"	Y = A
110000	SHL	Y = A << B
110001	SHR	Y = A >> B
110011	SRA	Y = A >> B (sign extended)



Control logic

	RESET	IRQ	OP	OPC	LD	LDR	ST	JMP	BEQ	BNE	ILLOP
ALUFN[5:0]	--	--	F(op)	F(op)	"+"	"A"	"+"	--	--	--	--
ASEL	--	--	0	0	0	1	0	--	--	--	--
BSEL	--	--	0	1	1	--	1	--	--	--	--
MOE	--	--	--	--	1	1	0	--	--	--	--
MWR	0	0	0	0	0	0	1	0	0	0	0
PCSEL[2:0]	--	4	0	0	0	0	0	2	Z ? 1 : 0	Z ? 0 : 1	3
RA2SEL	--	--	0	--	--	--	1	--	--	--	--
WASEL	--	1	0	0	0	0	--	0	0	0	1
WDSEL[1:0]	--	0	1	1	2	2	--	0	0	0	0
WERF	--	1	1	1	1	1	0	1	1	1	1

Testing

To create our benchmark test cases, we will convert the testing framework provided to us by the 6.004 staff to valid verilog. This should provide per-module tests to verify successful implementation.

Deliverables

- Minimum
    - Successful implementation of each of the predefined modules.
- Success
    - Processor capable of executing a program loaded into instruction memory.
    - Pipelined implementation to improve performance
- Stretch
    - Add additional instructions to accommodate more complicated programs
    - Add complex instructions which are composites of multiple instructions, such as the mov (%eax), %ebx instruction.
    - Reduce clock cycle and improve performance.