Justin Kamal

I pledge my honor that I have abided by the Stevens Honors System.

Instruction Usage

The CPU includes four general-purpose registers named X0, X1, X2, and X3. These registers are used for general computation and memory operations. In assembly programs, they can be directly referred to by their names (e.g., X0, X1).

Keep in mind that these instructions are both case-sensitive and format-sensitive. Additionally, you may only use the general-purpose registers, X0-X31, and immediate values within the range of [-1024, 1023]. The CPU supports operations like loading from and storing into memory, adding register values and immediates, branching, and multiplying register values.

Labels can be defined by appending a colon (:) after the label name. However, duplicate labels in your program will result in an error. Similarly, attempting to branch to a non-existent label will also produce an error.

The opcode is represented using 11 bits, as this is the minimum required to accommodate all control signals. For instructions using immediates, 11 bits are also required to represent values in the range [-1024, 1023]. Registers are represented with 5 bits since this is sufficient to encode register numbers from 0 to 31. For instructions that do not involve immediates, 6 bits are used to encode a special immediate value (56), which requires at least 6 bits for representation.

The LED display is also added to the CPU to visually show the result of calculations performed by the Arithmetic Logic Unit (ALU). It provides a simple, readable interface for verifying the output of operations like addition, multiplication, or any other ALU computations.

Binary Encoding Overview

Each instruction in the CPU is represented using 24 bits, divided as follows:

[opcode (8 bits) | bit8 (1 bit) | rN (5 bits) | rM (5 bits) | rD (5 bits)]

Opcode Breakdown

The first 8 bits specify the operation code, determining the type of instruction and its control signals. It is further divided into smaller fields to toggle specific control signals required for CPU operations.

Structure of the Opcode:

- Read/Write (1 bit): Determines the mode of the CPU.

        - 1 indicates write mode, where values are overwritten.
        - 0 indicates read mode, where values are only read.
- Special ID (2 bits): Used to distinguish between special operations such as FLASH, FLUSH, LOAD, and SAVING instructions.
- Rn/Rd (1 bit): Determines which register will be used for the operation.
    - If 1 the value in rD is used.
    - If 0, the value in rN is used. This is particularly relevant for SAVING instructions.
- R/bit8 (1 bit): Specifies the use of immediate (bit8) values or register values:
    - 1: Uses an immediate value as the second argument for ALU operations.
    - 0: Uses the value from a register instead.
- ALU ID (3 bits): Specifies the type of arithmetic or logical operation to perform within the ALU.
    - Examples of ALU operations include addition, subtraction, multiplication, and logical comparisons.

Opcodes

Instruction	Mem Write	Mem Read	Reg Write	UBr	ALUSrc	ALUAction	Flag Set	Mem2Reg	CBr
mov	0	0	1	0	1	000	0	0	0
addr	0	0	1	0	1	000	0	0	0
addn	0	0	1	0	1	000	0	0	0
mul	0	0	1	0	1	010	0	0	0
str	1	0	0	0	0	000	0	0	0
ldr	0	1	1	0	0	000	0	1	0
b	0	0	0	1	0	000	0	0	0
cmp	0	0	0	0	1	001	1	0	0
b.eq	0	0	0	0	1	000	1	0	1

Instruction	Description	Usage	Binary Encoding
mov	Moves an immediate value into destination register	mov rd, imm	00101000000 (opcode) + imm + 00000 + rd
addr	Adds the value in two registers and stores the summation in destination register	addr rd, rn, rm	00101000000 (opcode) + 111000 (special imm value) + rm + rn + rd
addn	Adds the value in a register and the immediate value, stores result in destination register	addn rd, rn, imm	00101000000 (opcode) + imm + rn + rd
mul	Multiplies the value in two registers and stores the product in destination register	mul rd, rn, rm	00100100000 (opcode) + 111000 (special imm value) + rm + rn + rd
str	Stores the value of a register into memory address at given immediate offset	str rn, rd, imm	10001000000 (opcode) + imm + rn + rd
ldr	Loads the value of memory address into destination register at given immediate offset	ldr rn, rd, imm	01101000010 (opcode) + imm + rn + rd
b	Branches to a label	b label	00011000000 (opcode) + (label instruction address - current instruction address) + 00000 + 00000
cmp	Compares if two registers have the same values and sets condition codes	cmp rn, rm	0000001100 (opcode) + 111000 (special imm value) + rm + rn + 00000
b.eq	Branches to label if the two registers have the same value	b.eq label	00001000001 (opcode) + (label instruction address - current instruction address) + 00000 + 00000

To run

Start by ensuring that both the assembler and your program file are located in the same directory.

- Ensure Python is installed.
- The assembler does not require additional libraries.

If they are not, an error will occur. Next, open your terminal and use the following command:

```
python assembler.py [filename]
```

When running this command, make sure to include the file extension of your program file (.txt, .s, etc.)

*Ensure that the compiler has read, write and execute permissions.

**File names can only be 25 characters long

Executing this command the assembler generates two output files:

1. text_image: Contains the machine code for the instruction memory in hexadecimal format.
2. data_image: Contains the machine code for the data memory in hexadecimal format.

This newly created file serves as the executable, which you can load into the RAM of the instruction memory in Logisim to run your program.