Computer Organization and Architecture Lab

Project: Design a 32 - bit RISC Processor

Name: Harsh Singh Jadon

Roll Number: 19CS01061

1. **Instruction Encoding -**
    a. The Encoding format is of 32-bit size (which goes in accordance with the guidelines of RISC - 32 processor).
    b. The **opcode** for any instruction is given by the first five bits of the encoding. (More about opcodes in assembler table section).
    c. The next five bits after the opcode represents the **destination register (Rc)**, that is in which register you would like to write back your information.
    d. The next five bits after the destination register signifies the **source register - 1 (Ra)**, that is from which register of the Register file, you want to load your values into Register Ra.
    e. (Overlapping) Depending on the fact that, whether your instruction makes use of immediate value/ offset, the next 16 bits are decided.
        i. Case 1 (When no immediate value or offset is present) - In this case, the first five bits out of 16 bits, signifies the **source register - 2(Rb),** that is from which register of the Register File, you want to load your values into Register Rb. **The remaining bits are kept at zero.**
        ii. Case 2 (When immediate value or offset is present) - In this case, the next 16 bits, signifies the immediate value or the offset, needed for the operation.
    f. At last, we have one bit that signifies **Enable Immediate or Offset**. It means that if any instruction makes use of an offset value or an immediate value, then this bit is set to 1, else it is set to zero (0).

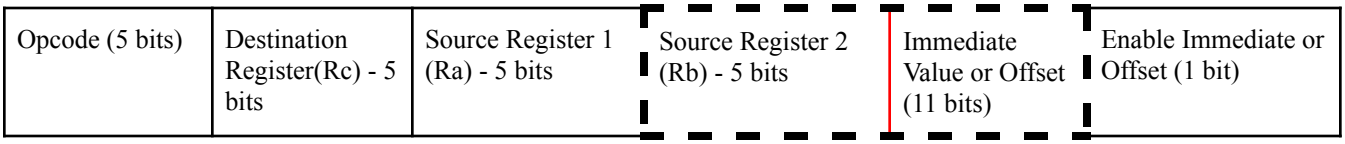

Fig 1: Instruction Encoding Format

Note: The dotted box symbolises that those 16 bits are overlapped and they take different formats depending on what the instruction is.

2. **Register File Encoding**
    a. The 32 - bit RISC processor, designed by me, comprises eight(8) General Purpose Registers, which are present inside the register file.
    b. Each register is represented from R0 - R7. (Ri means ith register).

Register	Encoding for Select Register (5 - bit)
R0	00000
R1	00001
R2	00010
R3	00011
R4	00100
R5	00101
R6	00110
R7	00111

Fig 2: Register Encoding

3. **Operation Encoding**
    a. The 32 - bit RISC processor designed by me, works in total for 13 assembly instructions.
    b. Each operation is identified by a 5-bit Opcode and 1 bit of Enable Immediate or Offset.

Operation (Full Name)	Operation (Short form)	Opcode (5 bits)	Enable Immediate or Offset (1 bit)
Halt	HLT	00000	0
Move	MOV	00001	0
Move Immediate	MVI	01000	1
Load	LOAD	00010	1
Store	STORE	00011	1
Addition	ADD	00100	0
Addition Immediate	ADI	00100	1
Subtract	SUB	00101	0
Subtract Immediate	SUI	00101	1
And	AND	00110	0
And Immediate	ANI	00110	1
Or	OR	00111	0
Or Immediate	ORI	00111	1

4. **Immediate Value Encoding**
    a. If the operation makes use of an immediate value or an offset, then the enable immediate value or offset bit is set to 1.
    b. The immediate value is represented using 16 bits, which are overlapped with source register - 2.
    c. For example, if X or offset is 7 (in decimal), then it is represented as -

0000 0000 0000 0111

5. Assembler Table

Note: a) The instruction should be written in the same order as given below, from left to right. Program Counter always starts from 0.
b) In the below example, for the **instruction column**, I am representing the offset or the immediate value in decimal format. It is converted into binary format in **immediate value** column.

I will now go through one example of each instruction, and tell how to represent it in my encoding format (32 - bit encoding).

Instruction (Numbers in decimal format)	Opcoe (5)	Destination Register (5)	Source Register - 1 (Ra) (5)	Source Register - 2 (Rb) (5)	Immediate Value or Offset (11)	Enable Immedi -ate (1)
HLT	00000	00000	00000	00000	00000000000	0
MOVE R0, R5	00001	00000	00101	00000	00000000000	0
MVI R7, 65535	01000	00111	00000	11111	11111111111	1
LOAD R5, 16(R0)	00010	00101	00000	00000	00000010000	1
STORE R5, 32(R0)	00011	00101	00000	00000	00000100000	1
ADD R7, R2, R3	00100	00111	00010	00011	00000000000	0
ADI R6, R2, 14	00100	00110	00010	00000	00000001110	1
SUB R4, R3, R2	00101	00100	00011	00010	00000000000	0
SUI R2, R4, 1	00101	00010	00100	00000	00000000001	1
AND R1, R2, R4	00110	00001	00010	00100	00000000000	0
ANI R3, R4, 65535	00110	00011	00100	11111	11111111111	1
OR R1, R2, R4	00111	00001	00010	00100	00000000000	0
ORI R3, R4, 65535	00111	00011	00100	11111	11111111111	1

Note: Bluenumbers denotes that these bits are don't care bits. It means that out of the valid instruction set as described above, you can take any value at that place and it won't effect the execution of a particular instruction. By default, I have taken it to be, a set of zeroes.

6. **How to store instructions in memory ?**
    a. We did convert our instructions into 32-bit binary format, but inside RAM(memory), the data is stored in hexadecimal format.
    b. I will go the conversion from binary to hexadecimal, for other examples below.
    c. The RAM/ Memory which I have used, has **address bit width of 16** and **data bit width of 32.**
    d. Size of RAM : 256 kB

Instruction (Numbers in decimal format)	Hexadecimal Encoding (For memory/ RAM)
HLT	0x00000000
MOVE R0, R5	0x080a0000
MVI R7, 65535	0x41c1ffff
LOAD R5, 16(R0)	0x11400021
STORE R5, 32(R0)	0x19400041
ADD R7, R2, R3	0x21c43000
ADI R6, R2, 14	0x2184001d
SUB R4, R3, R2	0x29062000
SUI R2, R4, 1	0x28880003
AND R1, R2, R4	0x30444000
ANI R3, R4, 65535	0x30c9ffff
OR R1, R2, R4	0x38444000
ORI R3, R4, 65535	0x38c9ffff