North South University

Department Of Electrical and Computer Engineering

Topic : **ISA Design Project proposal**
Course Name: Computer Organization and Architecture
Instructor: Dr. Tanzilur Rahman
Section: 02
Group : 04
Group members:

Name	ID
Mojahidul Islam	2013330642
Md Tahmid Ahmed Rakib	2021179642
Shahporan Hosen Shanto	2021192642

Introduction:

Here we are proposing an Instruction Set Architecture (ISA) for a 14 bit CPU. It will be able to perform operations on 14 bit data. For limitation of instructions length we have decided to keep 8 registers in our design 18 instructions and 3 formats.

Operands:

We have decided to keep 3 and 2 operands in our ISA model depending on the format. Mostly they are register based and there are also options for constants or immediate value.

Formats:

For our ISA design we kept 2 Formats for the instruction.

- R - Type
- I - Type
- J - Type

R - Type Format: R-Type format has 4 splits and is used for arithmetic and logical operation among variables.

- Bit 0 - 2 means destination register.
- Bit 3 - 5 means source 2 register.
- Bit 6 - 8 means source 1 register.
- Bit 9 - 13 means opcode.

Here ***Opcode*** will be used for the instructions. ***Rs1*** is source register 1. ***Rs2*** is source register 2. ***Rd*** is the destination register.

R - Type Format :

Opcode					Rs1			Rs2			Rd		
13	12	11	10	9	8	7	6	5	4	3	2	1	0

I - Type Format: I-Type format has 3 splits and is used for arithmetic and logical operation among variable and immediate value. Another use case of I type is for making jump instructions.

- Bit 0 - 5 means a binary of an actual value.
- Bit 6 - 8 means source and destination register.
- Bit 9 - 13 means opcode.

Here ***Opcode*** will be used for the instructions. ***Rsd*** will represent source as well as destination register. And ***Immediate*** will contain an immediate value.

I - Type Format :

Opcode					Rsd			Immediate					
13	12	11	10	9	8	7	6	5	4	3	2	1	0

J - Type Format: J type format will be used for conditional instructions. Like if R1 and R2 equal the goto 5 or jumping to a defined location.

- Bit 0 - 2 contains an immediate value.
- Bit 3 - 5 means source 2 register.
- Bit 6 - 8 means source 1 register.
- Bit 9 - 13 means opcode.

Here ***Opcode*** will be used for the instructions. ***Rs1*** is source register 1. ***Rs2*** is source register 2. And ***Immediate*** will contain an immediate value.

J - Type Format :

Opcode					Rs1			Rs2			Immediate		
13	12	11	10	9	8	7	6	5	4	3	2	1	0

Registers:

There are a total 8 registers numbered from R1 to R7 which are available for load data and make operation. The 1st register named ZERO always keeps value 0 for comparison purposes.

Serial	Name	Access	Comment
1	ZERO	Read only	Always 0
2	R1	R/W	variable
3	R2	R/W	variable
4	R3	R/W	variable
5	R4	R/W	variable
6	R5	R/W	variable
7	R6	R/W	variable
8	R7	R/W	variable

Register R1 to R7 can be used as variables for storing data for ALU operations and can be used for comparison. All these registers have read and write access so any data stored can be overridden. On the other side the 1st or ZERO register is always grounded and always carries value 0 for comparisons. I have only read access and do not have write access.

Instruction Table:

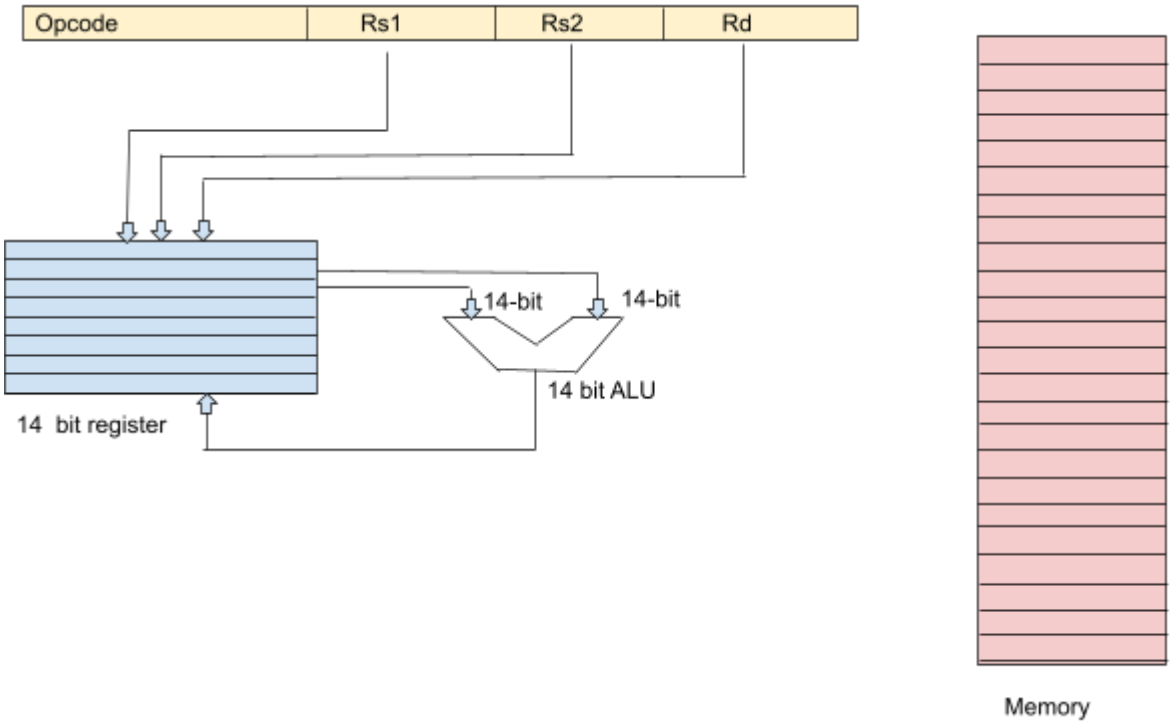
There are a total 18 operations in our ISA design. 4 arithmetic, 5 logical, 4 conditional, 1 unconditional, 2 data transfer and 2 input output operations. Their Name, uses, Type, Example, Meaning and opcodes are given in the table below.

No	***Type***	***NAME***	***Format***	***EXAMPLE***	***Meaning***	***Opcode***
01	Arithmetic	ADD	R	ADD R3, R1,R2	R3 = R2 + R2	00000
02	Arithmetic	ADDi	I	ADDi R1, 5	R1 = R1 + 5	00001
03	Arithmetic	SUB	R	SUB R3, R1,R2	R3 = R1 - R2	00010
04	Arithmetic	SUBi	I	SUBi R1, 5	R1 = R1 - 5	00011
05	Logical	AND	R	AND R3, R1,R2	R3 = R2 & R2	00100
06	Logical	OR	R	OR R3, R1,R2	R3 = R2 | R2	00101
07	Logical	XOR	R	XOR R3, R1,R2	R3 = R2 ^ R2	00110
08	Logical	SLL	I	SLL R1, 4	R1 = R1 << 4;	00111
09	Logical	SRL	I	SRL R1, 4	R1 = R1 >> 4;	01000
10	Conditional	COMP	R	COMP R3, R1, R2	R3 = (R1 < R2)	01001
11	Conditional	COMPi	J	COMPi R2, R1, 5	R2 = (R1 < 5)	01010
12	Conditional	EQL	J	EQL R1, R2, X	If (R1 == R2) branch to X	01011
13	Conditional	NEQL	J	NEQL R1, R2, X	If (R1 != R2) branch to X	01100
14	unconditional	JUMP	I	JUMP 5	Jump to 5	01101
15	Data Transfer	LOAD	J	LOAD R1, R5	R1 = Memory[R5]	01110
16	Data Transfer	STOR	J	STOR R1, R5	Memory[R5] = R5	01111
17	I/O	INPT	I	INPT R1	R1 = User Input	10000
18	I/O	OUT	I	OUT R1	Print R1 to Display	10001

Addressing Modes:

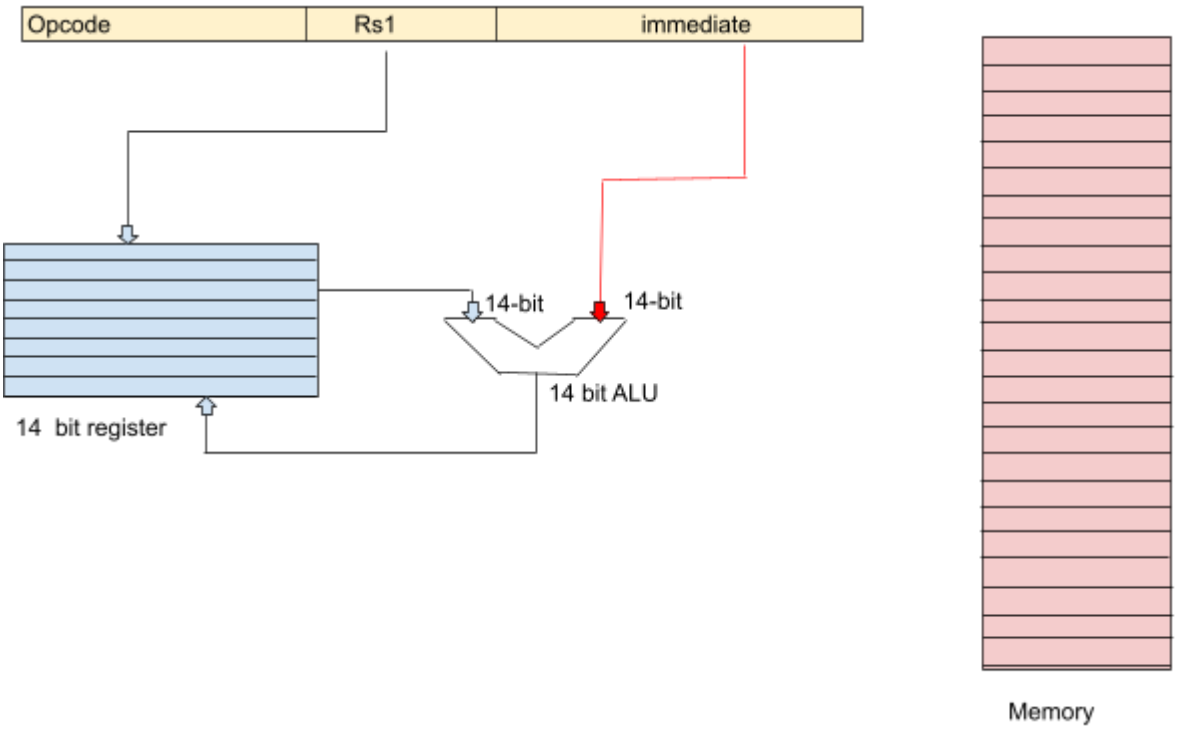
Registrar Addressing:

Here We will provide instruction code or opcode and three register addresses in the instruction. ALU will take two data from one or two registers and store the result after operation into another register according to the providing instruction. In this mode there is no interaction with the *Main Memory.* Here the diagram explains that clearly.



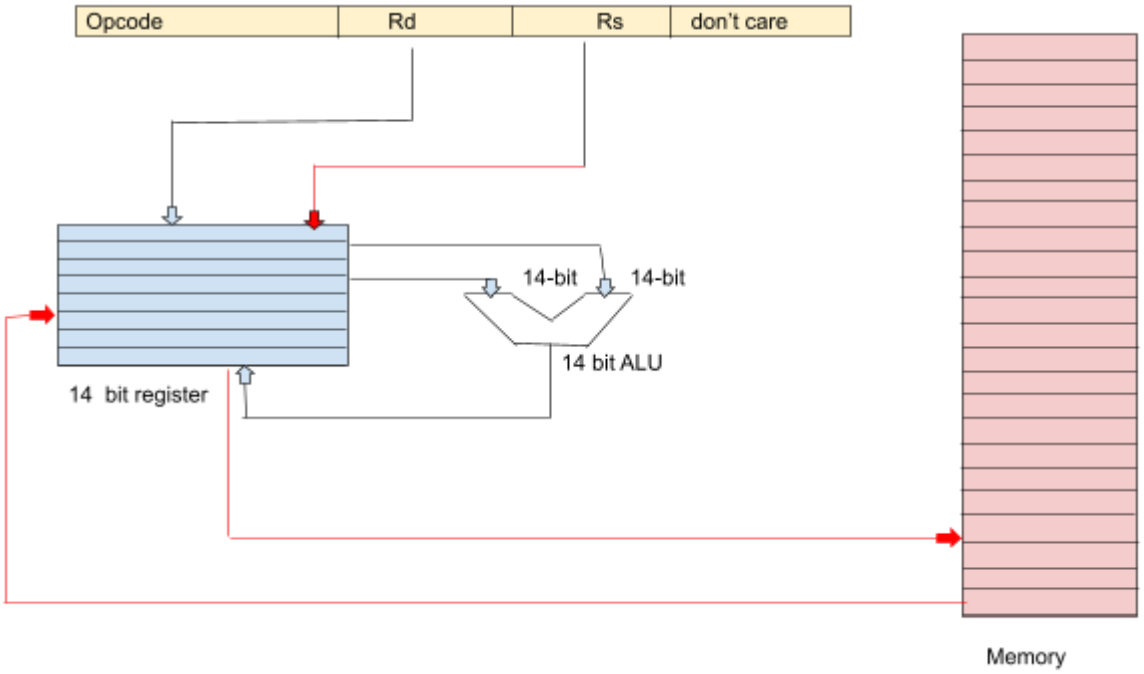
Immediate Addressing:

Here We will provide instruction code or opcode and one register address and an immediate value in the instruction. ALU will take one data from one register, one data from the instruction and store the result after operation into the same register. In this mode there is no interaction with the *Main Memory.* Here the diagram explains that clearly.



Memory (base) Addressing:

In memory Addressing os base addressing we provide two memory addresses in the instruction. One register as destination for the data to store and one register as source. This register will carry the calculated memory address of the array element.
So, **writing *LOAD R1, R2* will imply that, take data stored in R2 as a memory address, go to that address, bring the data from that address and store the data into R1.**



Examples For benchmarking

Simple Codes:

A = A + B; A = A - 5; A = A ^ B; A = A * 4; A = 0;

[Registers A = R1, B = R2]

Assembly Level translation

```
        ADD R1, R1, R2
        SUBi R1, 5
        XOR R1, R1, R2
        ADD R1,R1,R1
        ADD R1,R1,R1
        ADD R1,R1,R1
        AND R1, R1, ZERO
```

Conditional Code:

```
if(i < 2)
        sum += a;
else
        sum -= a;
```

[Registers i = R1, sum = R2, a = R3]

Assembly Level translation

```
                COMPi R4, R1, 2
                NEQL R4, ZERO, GO
                SUB R2, R2, R3;
                JUMP GO2
GO:             ADD R2, R2, R3;
GO2:
```

For Loop Execution:

```
for(i = 0; i <5; i++){
        sum += arr[i];
}
```

[Registers i = R1, sum = R2, arr(base) = R3]

Assembly Level translation

```
            AND R1, R1, ZERO
Start:      COMPi R4, R1, 5
            EQL R4, ZERO, Break
            SLL R4, R1, 2
            ADD R4, R3, R4
            LOAD R5, R4
            ADD R2, R2, R5
            ADDi R1, 1
            JUMP start
Break:
```

END OF ASSIGNMENT