Department of Electrical & Computer Engineering

North South University

PROJECT Title: ISA Design (Version:2)

Course: CSE 332 - Computer Organization & Design

Section – 02

Fall 2017

Submitted to: Dr. Tanzilur Rahman(TnR)
Assistant Professor,
Department of Electrical & Computer Engineering,
North South University

Submitted by: Group Members

	Name	ID
1.	Md. Sajid Ahmed	1610364042
2.	Md. Nazmul Islam	1511410042
3.	B. M. Raihanul Haque	1512756042

<u>Introduction:</u> Our task was to design a 8 bit RISC type ISA.

Objectives: Our objectives was to design a 8 Bit ISA which can solve a particular problems i. e. arithmetic addition, branching etc.

How many Operands?

- There are two operands, which we represented as **d** and **s**.

Type of Operands?

- Register based
- Memory based

How many Operations? Why?

We allocated 3 bits for the opcode, so the number of instructions can be executed is 2^3 or 8.

Types of Operations?

There will be in total four different types operations. The categories are:

- Arithmetic
- Logical
- Data transfer
- Conditional Branch
- Unconditional Branch

Catagory	**Operation**	**Name**	**Opcode**	**Type**	**Syntax**	**Comments**
Arithmetic	Addition	***add***	000	R	**add $rd, $rs**	Two register operands
Arithmetic	Addition as Subtraction	***sub***	001	R	**sub $rd, $rs**	Two register operands
Arithmetic	Addition immediate	***addi***	010	I	**addi $rd, const.**	Used to add constants
Data Transfer	load word	***lw***	011	I	**lw $rd, offset**	Load data from memory to register
Data Transfer	Store word	***sw***	100	I	**Sw offset($rd)**	Load data from register to memory
Conditional	Branch on equal	***beq***	101	I	**beq $rd, offset**	Check equality if else condition
Unconditional	Jump	***Jmp***	110	I	**Jmp offset**	Jump to Given location
Logical	Shift left logical	***sll***	111	R	**Sll $rd, offset**	Shift left by constant

How many Formats?

We would like to use 2 formats for our ISA

Register type – (R-Type)

Immediate Type – (I- Type)

(R-Type) ISA Format

opcode	**rd**	**rs**	**Shift Amount**
3 bits	**2 bits**	**2 bits**	**1 bits**

(I-Type) ISA Format

opcode	**rd**	**Immediate**
3 bits	**2 bits**	**3 bits**

List of registers

- As we have allocated 2 bits for the registers so we will have $2^2 = 4$ registers and all of them will be store type.

Register Table

Register name	Type	Reg. Number	Binary Value
$ac	Saves Values\ Accumulator	0	00
S_1$	Saves values	1	01
S_2$	Saves values	2	10
S_3$	Saves values	3	11

Detailed Instructions and Their Operations Instruction Description

Add: It adds two registers and stores the result in the first register.

ADD.

- Operation: **d = d + s**
- Syntax: **add $rd, $rs** **(***$rd = $rd +$rs* **)**

Sub

- Operation: **d = d – s**

- Syntax: **sub $rd, $rs (*$rd = $rd -$rs*)**

Addi : It adds a value from register with an integer value and stores the result in destination register.

- Operation: **d = s + constant**
- Syntax: **addi $rd, Constant (***$rd = $rd +const.***)**

lw: It loads required value from the memory to the register for calculation .

- Operation: d = M[s + offset]
- Syntax: **lw $rd, offset (*$rd =Mem[$rd+offset]*)**

sw: It stores specific value from register to memory.

- Operation: **M[d + offset] = s**
- Syntax: **sw $rs, offset (*Mem[$rd+offset] =$rd*)**

beq: It checks whether the values of two register s are same or not. If it is same it performs the following operation

- Operation: if (d == s) jump to offset

  else goto next line

- Syntax: **beq $rd, offset (*if $rd == $AC,* *goto* *offset* *location*)**

Sll : Shift left by constant we can use sll for multiplication

sll $rd,const. ($rd = $rd << const.)

Translating Some HLL codes using our Designed 8 Bit ISA

Assigned register for individual variable: ***$ac, $s1, $s2, $s3***

(i) g = A[i]

```
sll  $s2, 1            # here $s2 = i,   i = i*2
add  $s1, $s2          # here, $s1 = base address of A, $s2 = i
lw $s1, 0              # we have fetched the value that was stored in A[i]
sub $s3,s3             #initialize $s3 =0 for moving data into $s3
add $s3, $s2           # here we stored the value of A[i] in $s3
```

(ii) g = g + c

```
add $s1, $s2
```

(iii) g = g – c

```
sub $s1, $s2
```

(iv)

```
if ( i == j )             1. beq $s2,  La
        i = i+1;          2.addi $s2, -1
                          3. jmp Exit
   else                   La. addi $s2,  1
         i = i-1;         5. Jmp Exit
                          6. Exit:
```

(v) if (i <j)

```
            i = i+1;
    else
           i = i-1;
```

```
1      sub $s1,$s2
2.     sub $ac,$ac        [ac=(ac-ac) =0
3.     addi $s3, -4
4:     beq $s1, L2
La     addi $ac, -1
6:     beq  $s1, L1
7:     beq $s3 L2
10:    jmp La
L1:    addi $s1, 1
L2:    addi $s1, -1
```

For Loop :

for(int i = 0, i < a ; i++)   [**i = *$s1, a = $s2*   x = $s3**]
  x=x*8;

```
      sub $s1,$s1
      sub $ac,$ac          ac = 0
      add $ac,$s2          ac gets $s2
Loop: beq $s1,Exit
      sll  $s3,2           $s2*4
      sll $s3,1            $s2*8
      addi $s1,1
      jmp Loop

Exit:
```

Analysis and limitation:

We used 3 bit for opcode so we can use extra 1 bit for shift amount. 8 bit constrains limited our options so we were needed to be careful allocating space for opcode, source and destination. For this similar reason we allocated 3 bit for immediate .

Limitations we have:

1. We do not have ***slt*** instruction . We used an alternate method which is beq. It helps us comparing 2 numbers(by using a special purpose register name ***$ac***).
2. We have limited option in immediate format. For instance, we can not perform logical operations like AND,OR,NOT…. etc.
3. We cannot perform immediate operation whose size is larger than 2^3 =8 bits .