N O R T H   S O U T H   U N I V E R S I T Y

Centre of Excellence in Higher Education

School of Engineering and Physical Science

CSE332: Computer Organization and Architecture

ISA Design Project Milestone - 01

Course Code : CSE332
Course Title : Computer Organization and Architecture
Section : 9
Faculty : Dr. Tanzilur Rahman (Tnr)
Group Number : 08 (LAB group)
Group members:

Serial No.	Name	ID
1	Mojahidul Islam Rakib	2111077642
2	Sheikh Tanvir Hossain	2111647642
3	Syed Rakibul Hassain Apurbo	2111197642
4	Mohammad Zonayet Hassan Polok	2031257642
5	A K Faizul Haque Konok	1931939642

[This Part Done By Mojahidul Islam Rakib 2111077642]

SUBMITTED TO:

Dr. Tanzilur Rahman (Tnr)

Objective:

1. To design a new 10 bit single-cycle CPU that has separate Data and Instruction Memory.

2. To generate a machine code from a file containing assembly language. The assembler reads a program written in an assembly language, then translates it into binary code and generates output file containing machine code. The ISA should be general-purpose enough to be able to run provided general programs.

Design:

In our ISA design, we allocate 4bits for Opcode, 3bits for source register and another 3 bits for destination register.

Bit Size: 9-0 bits

9-6	5-3	2-0
Opcode	Destination	Source Operand
4bits	3bits	3bits

- ***Operands:*** As our ISA design, we have proposed 2 individual registers where each of the registers contain 2^3(8 bit memory).

- ***Types of Operands***: Because of using only 2 operands, our operands are register based on General Purpose Register where we can load and store values as temporary or fixed memory as well as assign immediate values and make paths with non-volatile memory.

- ***Operations***: Total bit reserved for our operation code is 4. By which, we can perform 2^4=16 operations. That means, we are allowed to set 16 different instructions to perform.

- ***Types of Operation*****:** Our design focusing on the following three categories of programs-

i. Simple Arithmetic & Logic Operations
ii. Programs That Require Checking Conditions
iii. Loop Type of Programs

That's Why We Need This 5 Types of Operations-

i. Arithmetic Operation
ii. Logical Operation
iii. Data Transfer Operation
iv. Conditional Operation
v. Unconditional Operation
vi. Shift

For completing these operations, we have to use some distinguish instructional operations. For Arithmetic (add, addi, sub,subi), for logical (nand,nor), for data transfer we must use (lw, sw), for condition check (beq, slt, slti), for unconditional (we also use -jump), for Shift (sll, srl) and for input and output(in,out).

Category/ Instruction Type	**Instruction/Operation**	**Format**	**Opcode**	**Example**	**Meaning**
Arithmetic	Add	R	0000	Add $S0, $S1	$S0 = $S0 + $S1
	Sub	R	0001	Sub $S0,$S1	$S0 = $S0 - $S1
	Addi	I	0010	Addi $S0, 20	$S0 = $S0 + 20
	Subi	I	0011	Subi $S0, 20	$S0 = $S0 - 20

Shift	SLL	I	0100	Sll $S0, 2	$S0=$T0 (Shift Left 2 Times)
	SRL	I	0101	Srl $S0, 2	$S0=$T0 (Right Left 2 Times)
Logic	NAND	R	0110	Nand $S0,$S1	$S0 = $S0 Nand $S1
	NOR	R	0111	Nor $S0,$S1	$S0 = $S0 Nor $S1
Data Transfer	LW	I	1000	Lw $S0, 2	$S0 = Mem[$T0 + Offset]
	SW	I	1001	Sw $S0, 2	Mem[$T0 + Offset] = $S0
Conditional	BEQ	I	1010	Beq $S0,4	If($S0 == $T0) Then Jump To 4
	SLT	I	1011	Slt $S0, $S1	If ($S0 < $S1) Then $T0 = 1 Else $T0 = 0
	SLTi	I	1100	Slti $S0, 4	If ($S0 < 4) Then $T0 = 1 Else $T0 = 0
Unconditional	J	J	1101	J L1	Jump To Level L1
In/out(put)	IN	I	1110	In $t1	Read a value from an input device and store it in $t1
	OUT	I	1111	Out $t2	Output the value in $t2 to an output device

Formats

In our design, we use 3 format of RISC architecture, they are-

- R- Type
- I -Type
- J –Type

R-Type MIPS Format for our ISA:

9-6	5-3	2-0
Opcode	Destination	Source Operand
4bits	3bits	3bits

I-Type MIPS Format for our ISA:

9-6	5-3	2-0
Opcode	Destination Register	Immediate
4bits	3bits	3bits

J-Type MIPS Format for our ISA:

9-7	6-0

Opcode	Target
4bits	7 bits

List of Register: As we have mentioned, our source operands' bit size are 3; so we have designed with $2^3 = 8$ registers for each operands.
We have selected registers from $zero, $s0-$s2 and $t0-$t3 and assigned 3bits for each of the register.

Register Name	**Binary Value**	**Commands**
$Zero	000	Ground to zero
$S0	001	Save
$S1	010	
$S2	011	
$t0	100	$t0 default checker Temporary data
$t1	101	
$t2	110	
$t3	111	

Addressing Modes:

- *Register Addressing* – Add $S1,$S2

> Operation - $S0 = $S0 + $S1

- *Immediate Addressing* – Addi $S0,11

> Operation - $S0 = $S0 + 11

- *Base Addressing* - Lw $S0, 2($S1)

> Operation - $S0 =Mem[$S1 + Offset]

Benchmark:

- *Simple arithmetic*
    - a = a + b , where a is in $S0, b is in $S1

      Syntax: Add $S0, $S1   Operation: $S0 = $S0 + $S1

  //Here it adds the content of the source register 1($s0) to the contents of the source register 2($s1) and saves it in the destination register($s0).

- *Logic Operation*
    - a = a AND b, where a is in $S0, b is in $S1

      Syntax: and $s0, $s1   Operation: $s0 = $s0 & $s1

//It does bit by bit logical AND operation between two source registers contents.

- *Programs that require checking conditions*
    - if (i == 10)   , where i is in $s2

      i = i + 4

  else

  i = i -5

```
sub $t0, $t0;       \\ $t0=0
addi $t0, 10        \\ $t0 = 10
beq $s2, L          \\ Here, compare $s2 = i with $t0 = 10. If equal then go to level L
subi $s1,5          \\ i = i – 4,   $s2 = $s2 - 5
J  Exit             \\ Jump to exit, end of program
L: addi $s2, 4      \\ i = i + 4, $s2 = $s2 + 4
Exit
```

- *Loop type of programs*
  - for (i = 0, i < 5, i++)   , where i is in $s2, a is in $s0

    a = a + 4

```
sub $s2, $s2       \\ $s2 = $s2 - $s2,   clear register and initialize i to zero
sub $t0, $t0       \\ clear temporary register
L1: slti $s2, 5     \\ if (s2 < 5) then t0 = 0 else t0 = 1
beq $zero, Exit     \\ compare t0 with zero if equal then exit and end program
addi $s0, 4          \\ $s0 = $s0 + 4
addi $s2, 1          \\ increment i by 1
J L1                 \\ jump to level and repeat loop
Exit
```

Throughout this phase (ISA project millstone one), we should test our CPU to use a variety of suggested Instruction that we have proposed.

To solve an issue, for instance a loop, we have to provide input to every instruction.

Then the instructions are loaded into the processor's instruction memory so that the instructions can be executed sequentially.

Yet we are unable to carry out those steps without instruction memory and ALU.

The End.