Instruction Set Architecture (ISA)

Objectives: Our objective was to design a 16 Bit ISA which can solve particular problems i. e. simple arithmetic, logical, branching, and data transfer operations.

Operands: We will be using three operands to carry out the operations.

Types of operands: For arithmetic instruction, we need register operands and for data transfer, we need memory operands. So, types of needed operands:

- Register based
- Memory based

Operations: The Op-Code consists of 4 bits, so the executable instructions number will be 2^4 or 16. However, currently only 11 operations will be executed in this ISA.

Types of operations: In our design there will be six different types of operation. The operations are:

- Arithmetic
- Logical
- Data Transfer
- Conditional
- Unconditional Jump
- Halt

Name	Op Code	Type	Operation	Syntax	Category	Meaning
nop	0000	I	Halts program	nop	Halt	Stops the program from running.
sll	0001	I	Shift left	sll $s1 $d2 2	Logical	Shift $s1 by 2 to left and write in $d2
sub	0010	R	Subtraction	sub $s1 $s2 $d3	Arithmetic	Subtract $s2 from $s1 and write in $d3
sw	0011	I	Store	sw $d1 $s2 3	Data Transfer	Store $s2 in Memory[$d1+3]
and	0100	R	Bit-by-bit AND	and $s1 $s2 $d3	Logical	And $s1 and $s2 and write in $d3
beq	0101	I	Equal check	beq $s1 $s2 11	Conditional	If $s1 = $s2, go to 11^{th} instruction in PC
jmp	0110	I	Jump to certain instruction in PC.	jmp 11	Unconditional jump	Jump to 11^{th} instruction in PC
slt	0111	R	Compare less than	slt $s1 $s2 $d3	Conditional	If $s1 < $s2, $d3=1. Else, $d3 = 0
add	1000	R	Addition	add $s1 $s2 $d3	Arithmetic	Add $s1 and $s2 and store in $d3
addi	1001	I	Addition with constant	addi $s1 $d2 5	Arithmetic	Add $s1 and 5 and write in $d2
lw	1010	I	Load	lw $s1 $d2 3	Data Transfer	Load from Memory[$s1+3] to $d2

Format: We have used two formats in this ISA and they are:

- Register type – R type
- Immediate type – I type

R-Type Format

OP	RS	RT	RD
4	4	4	4

I-Type Format

OP	RS	RT	IM
4	4	4	4

Note: As we are using the 'jmp' operation in I-type format, the immediate value has the size of 4 bits. So, the maximum number of instructions we can jump to is $1111_2 = 15_{10}$.

List of register: As we have allocated four bits register so the number of registers will be $2^4 = 16$.

Register Number	Conventional Name	Usage	Binary Value
0	$zero	General purpose	0000
1	$s1	General purpose	0001
2	$s2	General purpose	0010
3	$s3	General purpose	0011
4	$s4	General purpose	0100
5	$s5	General purpose	0101
6	$s6	General purpose	0110
7	$s7	General purpose	0111
8	$d1	General purpose	1000
9	$d2	General purpose	1001
10	$d3	General purpose	1010
11	$d4	General purpose	1011
12	$d5	General purpose	1100
13	$d6	General purpose	1101
14	$d7	General purpose	1110
15	$d8	General purpose	1111

Instruction Description:

nop: Stops the program from running.

- Syntax: nop

sll: Shifts a register value to the left.

- Syntax: sll $s1, $d2, 2
- Operation: $d2 = $s1<<2

sub: Subtraction between two registers value and stores the value in a register.

- Syntax: sub $s1, $s2, $d3
- Operation: $d3 = $s1 - $s2

sw: Stores the value of a register in the memory.

- Syntax: sw $d1, $s2, 3
- Operation: Mem[$d1+3] = $s2

and: Bit-by-bit AND between two registers value and stores the value in a register.

- Syntax: and $s1, $s2, $d3
- Operation: $d3 = $s1 && $s2

beq: Equal check between two registers. If they are equal, program counter jumps to a certain instruction.

- Syntax: beq $s1, $s2, 11
- Operation: if ($s1==$s2), go to 11^{th} instruction.

jmp: Jump to a certain instruction.

- Syntax: jmp 11
- Operation: Go to 11^{th} instruction.

slt: Compare less than between two register values. If true, value of the specified register changes.

- Syntax: slt $s1, $s2, $d3
- Operation: If $s1 < $s2, $d3=1. Else, $d3 = 0.

add: Addition between two registers value and stores the value in a register.

- Syntax: add $s1, $s2, $d3
- Operation: $d3 = $s1 + $s2

addi: Addition between immediate value and a register value. Stores the value in a register.

- Syntax: add $s1, $d2, 5
- Operation: $d2 = $s1 + 5

lw: Loads a value to the register file from memory.

- Syntax: lw $s1, $d2, 3
- Operation: $d2 = Mem[$s1+3]