

NORTH SOUTH UNIVERSITY

Department of Electrical & Computer Engineering

**Summer 2021**

**Course: CSE332**

Course Title: Computer Organization and Architecture

Project: ISA (Part-1)

Section: 2

**Group-1**

Group Members

|  |  |
| --- | --- |
| **NAME** | **ID** |
| Kazi Ramisa Rifa | 1911019642 |

**Introduction:**

Our purpose is to design a 14 bit single-cycle CPU that has separate Data and Instruction Memory. The ISA should be general purpose enough to be able to run provided general programs.

**Design:**

1. **Operands:** There are 2 operands where operands are 4 bits.
2. **Operand Type**: We used register type operands and memory based operands which means mixed operand.
3. **Operations:** In our design there are 16 operations because opcode is 4 bits [24=16 operations]. So opcode can be uniquely identified with 4 bit binary values.
4. **Operation Type:** In our design there are 5 types of operations.

* Arithmetic (3 categories): add, sub, addi
* Logical (5 categories): and, or , nor, sll, srl
* Conditional Branch (3 categories): beq, bne, slti
* Unconditional Jump (1 categories): j
* Data Transfer (4 categories): lw, sw, in, out

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Category** | **Instructions** | **Syntax** | **Meaning** | **Format** | **Opcode** |
| Arithmetic | Add | add $s0, $s1 | $s0 = $s1 + $s0 | R | 0000 |
| Subtract | sub $s0, $s1 | $s0 = $s1 - $s0 | R | 0001 |
| Add Immediate | addi $s0, offset | $s0 = $s0 + constant | I | 0010 |
| Logical | AND | and $s0, $s1 | $s0 = $s0 & $s1 | R | 0011 |
| OR | or $s0, $s1 | $s0 = $s0 | $s1 | R | 0100 |
| NOR | nor $s0, $s1 | $s0 = ~($s0 | $s1) | R | 0101 |
| Shift left logic | sll $s0, offset | $s0 = $s0<< constant | R | 0110 |
| Shift right logic | srl $s0, offset | $s0 = $s0>> constant | R | 0111 |
| Conditional  Branch | Branch on equal | beq $s0, target | If ($s0 == $s6)  Go to target address | I | 1000 |
| Branch on not equal | bne $s0, target | If ($s0!= $s6)  Go to target address | I | 1001 |
| Set on less than | slti $s0, offset | If ($s0 < constant) $s7=1;  Else $s7= 0; | I | 1010 |
| Unconditional  Jump | Jump | j target | Jump to target address | J | 1011 |
| Data Transfer | Load word | lw offset($s0) | $sp = Memory[$s0 + constant] | I | 1100 |
| Store word | sw offset($s0) | Memory[$s0 + constant] = $sp | I | 1101 |
| In | din $s0 | $s0 = User Input | R | 1110 |
| Out | dout $s0 | System Output = $s0 | R | 1111 |

1. **Formats:** We use 3 types of formats.

* Register Type (R-Type)
* Immediate Type (I-Type)
* Jump Type (J-Type)

R-Type List

|  |  |
| --- | --- |
| Operation | Opcode |
| add | 0000 |
| sub | 0001 |
| and | 0011 |
| or | 0100 |
| nor | 0101 |
| sll | 0110 |
| srl | 0111 |
| In | 1110 |
| Out | 1111 |

**R- Type ISA Format**

|  |  |  |  |
| --- | --- | --- | --- |
| **Opcode** | **rs** | **rd** | **sa** |
| 4 bits | 4 bits | 4 bits | 2 bits |

I-Type List

|  |  |
| --- | --- |
| Operation | Opcode |
| addi | 0010 |
| beq | 1000 |
| bne | 1001 |
| slti | 1010 |
| lw | 1100 |
| sw | 1101 |

**I-Type ISA Format**

|  |  |  |
| --- | --- | --- |
| **Opcode** | **rs** | **Immediate** |
| 4 bits | 4 bits | 6 bits |

J-Type List

|  |  |
| --- | --- |
| Operation | Opcode |
| J | 1011 |

**J-type ISA Format**

|  |  |
| --- | --- |
| **Opcode** | **Target** |
| 4 bits | 10 bits |

1. **Registers List:**

|  |  |  |
| --- | --- | --- |
| **Name of the Registers** | **Value Assigned (4 Bits**) | **Register Purpose** |
| $zero | 0000 | Hardwired to 0 |
| $sp | 0001 | Special purpose |
| $s0 | 0010 | General purpose |
| $s1 | 0011 | Do |
| $s2 | 0100 | Do |
| $s3 | 0101 | Do |
| $s4 | 0110 | Do |
| $s5 | 0111 | Do |
| $s6 | 1000 | Do |
| $s7 | 1001 | Do |
| $t0 | 1010 | Temporary |
| $t1 | 1011 | Do |
| $t2 | 1100 | Do |
| $t3 | 1101 | Do |
| $t4 | 1110 | Do |
| $t5 | 1111 | Do |

1. **Addressing Modes:**

* Register addressing modes.
* Immediate addressing modes.
* Base addressing modes.

**Instruction Description:**

add: It adds two registers and stores the result in destination register.

∙        Operation: $s0 = $s0 + $s1

∙        Syntax:  add $s0, $s1

sub: It subtracts two registers and stores the result in destination register.

∙        Operation: $s0 = $s0 - $s1

∙        Syntax: sub $s0, $s1

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

∙        Operation: $s0 = $s0 + offset

∙        Syntax: addi $s0, offset

and: It AND’s two register values and stores the result in destination register. Basically, it sets some bits to 0.

∙        Operation:  $s0 = $s0 && $s5

∙        Syntax: and $s0, $s5

or: It OR’s two register values and stores the result in destination register. Basically, it sets some bits to 1.

∙        Operation:  $s0 = $s0 || $s5

∙        Syntax: or $s0, $s5

nor: It NOR’s two register values and stores the result in destination register. Sometimes we use nor to get NOT of register value.

∙        Operation:  $s0 = $s0 nor $s1

∙        Syntax: nor $s0, $s1

sll: It shifts bits to the left and fill the empty bits with zeros. The shift amount is depended on the offset value.

∙        Operation: $s0= $s0 << offset

∙        Syntax: sll $s0, offset

srl: It shifts bits to the right and fill the empty bits with zeros. The shift amount is depended on the offset value.

∙        Operation: $s0= $s0 >> offset

∙        Syntax: srl $s0, offset

beq: It checks whether the values of two registers are same or not. If it’s same it performs the operation located in the address at offset value.

∙        Operation: if ($s0==$s6) jump to offset

else goto next line

∙        Syntax: beq $s0, offset

bne: It checks whether the values of two registers are same or not. If it’s not same it performs the operation located in the address at offset value.

∙        Operation: if ($s0!=$s6) jump to offset

else goto next line

∙        Syntax: bne $s0, offset

slti: If $s0 is less than offset, $sp is set to one. It gets zero otherwise.

∙        Operation:   if $s0 < offset $s7 = 1

else $s7 = 0

∙        Syntax: slti $s0, offset

lw: It loads required value from the memory and write it back into the register.

∙        Operation: $sp = MEM[$s0 + offset]

∙        Syntax:  lw offset($s0)

sw: It stores specific value from register to memory.

∙        Operation: MEM[$s0 + offset] = $sp

∙        Syntax:   sw offset($s0)

J: Jumps to the calculated address.

∙        Operation: PC = nPC

∙        Syntax: j target