­­

Project Documentation

**Submitted to: Shaikh Shawon Arefin Shimon**

**Project Prepared By:** Asif Mahmud (1511190642)

**Course:** CSE332

**Section:** 12  
**Semester:** Summer 2018

**01. How many operands?**

Answer: 3 operands [e.g.: z = x + y]

**02. Types of operands (register/memory/mixed)?**

Answer: Mixed

**03. How many Operations?**

Answer: 16

**04. Types of operations? (Arithmetic, logical, branch type?? How many from each category? List the opcodes and respective binary values)**

|  |  |  |
| --- | --- | --- |
| **Operation Types** | **Operations** | **Register Type** |
| 01. Arithmetic | ADD, SUB, Multiplication | R Type |
| 02. Logical | AND, OR, NOR, XOR | R Type |
| 03. Data Transfer | ADDi, ANDi, ORi, SLL, SRL, LW, SW | I-Type |
| 04. Conditional Branch | BEQ | I-Type |
| 05. Jump | Jump | J-Type |

**05. No. of format of instruction (how many different formats?)**

Answer: 3 Types of formats. R-Type, I-Type & J-Type.

**06. Describe each of the format (fields and field length)**

**R-Type Format: -**

* Each field is 4 bits in length
* OP is an operation code or opcode that selects a specific operation
* RS and RT are the first and second source registers
* RD is the destination register

|  |  |  |  |
| --- | --- | --- | --- |
| OP  4 bits | RD 4 bits | RS 4 bits | RT 4 bits |

**I-Type Format: -**

* Load work, store work, branch type, & immediate type are I-type
* Also used for shift left logical and shift right logical in this cpu.
* RS is a source register,   
  RT is an address for loads and stores, or an operand for branch and immediate arithmetic instructions. Also for shift amount.
* RD is a source register for branches, but a destination register for the other I-type instructions

|  |  |  |  |
| --- | --- | --- | --- |
| OP  4 bits | RD 4 bits | RS 4 bits | Address/Immediate/shift amount  4 bits |

**J-Type Format: -**

|  |  |
| --- | --- |
| OP  4 bits | (RS & RD & RT)  JUMP 12 bits |

**THE TRUTH TABLE**



**07. List of Registers:**

|  |  |  |
| --- | --- | --- |
| **Name of the Registers** | **Register Number** | **Value Assigned (4 Bits)** |
| $zero | 0 | 0000 |
| $t1 | 1 | 0001 |
| $t2 | 2 | 0010 |
| $t3 | 3 | 0011 |
| $t4 | 4 | 0100 |
| $t5 | 5 | 0101 |
| $t6 | 6 | 0110 |
| $t7 | 7 | 0111 |
| $s0 | 8 | 1000 |
| $s1 | 9 | 1001 |
| $s2 | 10 | 1010 |
| $s3 | 11 | 1011 |
| $s4 | 12 | 1100 |
| $s5 | 13 | 1101 |
| $s6 | 14 | 1110 |
| $s7 | 15 | 1111 |

**Instruction Description:**

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

Operation: $d = $s + $t

Syntax: add $d, $s, $t

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

Operation: $d = $s - $t

Syntax: sub $d, $s, $t

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

Operation: $d = $s && $t

Syntax: and $d, $s, $t

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

Operation: $d=$s || $t

Syntax: or $d, $s, $t

**XOR:** Exclusive ors two registers and stores the result in a register

Syntax: xor $d, $s, $t

Operation: XOR: Rd = Rs ^ Rt

**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: $d=$s nor $t

Syntax: nor $d, $s, $t

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

Operation: $d = $s + offset

Syntax: addi $d, $s, offset

**andi:** Bitwise AND’s a register and an immediate value and stores the result in a register.

Operation: $d = $s AND offset

Syntax: andi $d, $s, offset

**ori:** Bitwise OR’s a register and an immediate value and stores the result in a register.

Operation: $d = $s OR offset

Syntax: ori $d, $s, offset

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

Operation: $d = MEM[$s + offset]

Syntax: lw $d, offset($s)

**sw:** It stores specific value from register to memory.

Operation: MEM[$d + offset] = $s

Syntax: sw $s, offset($d)

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

Operation: if ($s==$t) jump to offset

else go to next line

Syntax: beq $s, $t, offset

**Jump** : Jump to the corresponding address.