­­

Project Assembler

**Submitted To:** Tanzilur Rahman (TnR)

**Project Prepared By:** Abdullah Mahmood (133-0639-042)

**Course:** CSE332  
**Semester:** Spring-2017

**North South University**

Department of Computer Science and Engineering

Homework 1, Spring-2017

Course No: CSE332, Sec 2, Faculty: TNR

Course Title: Computer Organization and Architecture

**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: 14 Opcodes

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

Answer: 4 types

|  |  |  |
| --- | --- | --- |
| **Operation Types** | **Operations** | **Register Type** |
| 01. Arithmetic | ADD, SUB | (R Type) |
| 02. Logical | AND, OR, NOR, XOR | (R Type) |
| 03. Data Transfer | ADDi, ANDi, ORi, XORi, LW, SW | (I-Type) |
| 04. Conditional Branch | BEQ, BNE | (I-Type) |

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

Answer: 2 Types of formats. R-Type, I-Type

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

Answer:

**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

For example: add $4, $3, $2

0000 0100 0011 0010

|  |  |  |  |
| --- | --- | --- | --- |
| 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
* RS is a source register,   
  an address for loads and stores, or an operand for branch and immediate arithmetic instructions
* RD is a source register for branches, but a destination register for the other I-type instructions

For Example: lw $5, 8($6)

1010 0101 0110 0100

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

**Format (fields and field length)**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Operations** | **Opcode (OP)**  **4 bit** | | | | **Destination (RD)**  **4 bit** | | | | **Source Reg (RS)**  **4 bit** | | | | **Target Reg (RT)**  **4 bit** | | | |
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| **ADD** | 0 | 0 | 0 | 0 | RD | | | | RS | | | | RT | | | |
| **SUB** | 0 | 0 | 0 | 1 | RD | | | | RS | | | | RT | | | |
| **NOR** | 0 | 0 | 1 | 0 | RD | | | | RS | | | | RT | | | |
| **OR** | 0 | 0 | 1 | 1 | RD | | | | RS | | | | RT | | | |
| **AND** | 0 | 1 | 0 | 0 | RD | | | | RS | | | | RT | | | |
| **XOR** | 0 | 1 | 0 | 1 | RD | | | | RS | | | | RT | | | |
|  | | | | | | | | |  | | | | | | | |
| **ADDi** | 1 | 0 | 0 | 0 | RD | | | | RS | | | | Immediate | | | |
| **lw** | 1 | 0 | 0 | 1 | RD | | | | RS | | | | Immediate | | | |
| **Sw** | 1 | 0 | 1 | 0 | RD | | | | RS | | | | Immediate | | | |
| **ORi** | 1 | 0 | 1 | 1 | RD | | | | RS | | | | Immediate | | | |
| **ANDi** | 1 | 1 | 0 | 0 | RD | | | | RS | | | | Immediate | | | |
| **XORi** | 1 | 1 | 0 | 1 | RD | | | | RS | | | | Immediate | | | |
| **BEQ** | 1 | 1 | 1 | 0 | RD | | | | RS | | | | Immediate | | | |
| **BNE** | 1 | 1 | 1 | 1 | RD | | | | RS | | | | Immediate | | | |

**07. List of Registers?**

Answer:

|  |  |  |
| --- | --- | --- |
| **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

Xori: Bitwise exclusive ors a register and an immediate value and stores the result in a register

Operation: $rt = $rs + imm

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

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

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

else goto next line

Syntax: bne $s, $t, offset

**Summary:**

* This is a 16-bit RISC Type CPU. As an ISA designer, I have chosen 3 mixed type operands, 15 opcodes, 5 types of operations, 3 types of instruction format, and finally I have given names and values of 16 different registers.
* This is a 16-bit CPU as a result there will be 16 registers in it. For the design, I believe simplicity favors regularity. So, I have assigned 4bits each to the opcodes, RD, RS, RT. 4 bits is fixed for the R-Type register and for the I-type, and J-Type the bits will vary in some cases because there is an immediate type.
* Smaller is faster. Though it is only a 16-bit CPU, it will handle very common operations. In this CPU, I tried to use as many as operations it can handle also keeping register’s values in my mind.