USER MANUAL FOR ASSEMBLER

Assembler operating user manual

BY-NABENDU BIKASH BINDA

Introduction:

Our task was to design an assembler which will convert our assembly code to machine language according to our ISA.

Objective:

Our main goal was to generate a machine code from a file containing assembly language. The assembler reads a program written in an assembly language, then translate it into binary code and generates output file containing machine code with hexadecimal format.

How to use:

In the input file the user has to give some instructions to convert into machine codes. The system will convert valid instructions into machine language and generate those codes into output file.

Input File:

The input file is located in a file named "input.txt". User will write down the code in this file.

List of Tables:

Register List-

We have selected registers from $t0, $t1, $t2 and $zero assigned 2 bits for each of the register. We selected 2 bits for register because of we design just 10 bits ISA.

Name of the Registers	Register Number	Value Assigned (2 Bits)
$t0	0	00
$t1	1	01
$t2	2	10
$zero	3	11

R-Type List-

We have selected following op codes and assigned functionality values (3 bits) for each of the op codes.

OP Code	Functionality
ADD	000
AND	001
OR	010

I-Type List-

We have selected following op codes and assigned functionality values (3 bits) for each of the op codes.

OP Code	Functionality
ADDi	011
LW	100
SW	101
BEQ	110

J-Type List-

We have selected following op codes and assigned functionality values (3 bits) for each of the op codes.

OP Code	Functionality
J	111

Instruction Description:

ADD: It adds two registers and stores the result in destination register.
Operation: $d = $s + $t
Syntax: ADD $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

ADDi: It ADD's a value from register with an integer value from immediate and stores the result in destination register.
Operation: $d = $s + immediate
Syntax: ADDi $d, $s, immediate

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 registers 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

J: It Jump to target address
Operation: JUMP label
Syntax: j labelNo

Limitation:

The user has to give spaces between instruction words in the input file. If user don't follow this format the system will show a valid code as invalid.