North South University

Department of Electrical & Computer Engineering

CSE332

Computer Organization and Architecture

Assembler Design

Submitted by:

Name: Md. Mashruf Ehsan

ID: 1831253642

Submitted to:

Tanjila Farah (TnF)

Introduction: Our task was to design an assembler which will convert the assembly code to machine language.

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.

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

Input File: The input file named "inputs". User will write down the MIPS code in this file.

List of Tables

Register List

We have selected registers from r0-r15 for general purpose. We assigned 4 bits for each of the register as we know in the instruction field in our ISA containing the register rs, rt and rd contains 4 bits each.

Conventional Name	Register Number	Binary Value
r0	0	0000
r1	1	0001
r2	2	0010
r3	3	0011
r4	4	0100
r5	5	0101
r6	6	0110
r7	7	0111
r8	8	1000

r9	9	1001
r10	10	1010
r11	11	1011
r12	12	1100
r13	13	1101
r15	14	1110
r15	15	1111

Op-Code List: We have selected following op codes and assigned op-code binary values (4 bits) for each of the op codes.

Name	Type	OpCode
nop		0000
sll	R	0001
and	R	0010
lw	I	0011
slt	R	0100
add	R	0101
addi	I	0110
sub	R	0111
sw	I	1000
jmp	J	1001
beq	I	1010

Instruction Description

nop: No operation.

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

- Operation: r3 = r1 << r2
- Syntax**:** sll r1 r2 r3

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

- Operation: r3 = r1 & r2
- Syntax: and r1 r2 r3

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

- Operation: r1 = Mem[r0+immediate]
- Syntax: lw r0 immediate

slt: If r1 is less than r2, r3 is set to one. It gets zero otherwise.

- Operation: if (r1 < r2)

  r3 = 1

  else

  r3 = 0
- Syntax: slt r1 r2 r3

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

- Operation: r1 = r1 + r2
- Syntax: add r1 r2 r1

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

- Operation: r1 = r2 + immediate
- Syntax: addi r2 r1 immediate

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

- Operation: r1 = r1 – r2
- Syntax: sub r1 r2 r1

sw: It stores specific value from register to memory.

- Operation: Mem[r0+immediate] = r1
- Syntax: sw r0 r1 immediate

jmp: Jumps to the calculated address.

- Operation: jum to target address
- Syntax: jmp target

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 (r1==r2) jump to immediate

else goto next line

- Syntax: beq r1 r2 immediate

Limitation:

The user has to give spaces between instruction words and nothing else like "," or "-" in between them in the "inputs" file. If user don't follow this format the system will show a valid code as invalid.

Manual:

To run the program, one needs to run the python file called "**assembler.py**" which is provided in the folder. If one wants to see the code then open the "**assembler.py**" file*, it is absolutely necessary that the folder which is containing the program, has called "**inputs**"* . This is the file from where the assembler reads the assembly codes. The program reads the code from **"inputs"** file and writes the corresponding binary

code in a file called **“outputs”.** We have already provided an input file with corresponding output file in the project folder. If one wants to try his/her own assembly code then, he/she needs to write the codes to the application through the input file. One important thing to notice is that, each line of the input file can only contain one instruction and words must be separated by spaces.