Department of Electrical & Computer Engineering

North South University

PROJECT Title: Assembler Design

Course: CSE 332 - Computer Organization & Design

Section – 02

Fall 2017

Submitted to: Dr. Tanzilur Rahman(TnR)
Assistant Professor,
Department of Electrical & Computer Engineering,
North South University

Submitted by: Group Members

	Name	ID
1.	Md. Sajid Ahmed	1610364042
2.	Md. Nazmul Islam	1511410042
3.	B. M. Raihanul Haque	1512756042

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 instructions into machine language and generate those codes into output file.

Input File

The input file is located in a folder named “File”. User will write down the assembly code in this file.

List of Tables

Register List

We have selected registers from $ac, $s1-$s3 and assigned 2 bits for each of the register as we know in the instruction field in our 8 bit system containing the register rs, and rd contain 2 bits.

Name of the Registers	Register Number	Value Assigned (2 Bits)
$ac	0	00
$s1	1	01
$s2	2	10
$s3	3	11

R-Type Op-Code List

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

Op-Code name	*Type*	*Op-Code binary*
add	(R-type)	000
sub	(R-type)	001
addi	(I-type)	010
lw	(I-type)	011
sw	(I-type)	100
beq	(I-type)	101
jmp	(I-type)	110
sll	(R-type)	111

Instruction Description

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

- Operation: **$rd = $rd + $rs**
- Syntax: **add $rd, $rs**

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

- Operation: **$rd = $rd - $rs**
- Syntax**:** **sub $rd, $rs**

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

- Operation: **$rd = $rd + offset**
- Syntax: **addi $rd, offset**

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

- Operation: **$rd = MEM[$rd + offset]**
- Syntax**:** **lw $rd, offset**

sw: It stores specific value from register to memory.

- Operation: **MEM[$rd + offset] = $rd**
- Syntax: **sw $rd, offset**

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

- Operation: **$rd= $rd << offset**
- Syntax**:** **sll $rd, 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 ($rs==$ac) jump to offset
  else goto next line
- Syntax**:** **beq $rs, offset**

jmp: Jumps to the calculated address.

- Operation: **PC = nPC**
- Syntax: **jmp target**
-

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.

This is a project for CSE332: Computer Organization and Architecture. This is a console based application with no GUI. The whole program has been written in C++ and no external library or frameworks were used. The entire program is written from scratch.

Technical details:

The application is built using multiple files. There are two header files, two source files and one driver file. The program uses object oriented programming techniques. There are two classes which have been named **R-type** and **I-type** according to their functionality. The supported instruction are:

R-type: add, sub, sll.

I-type: addi, lw, sw, beq, jmp.

The program supports wide range syntactic diversity. For example, if one doesn't put comma in the instructions or if one writes the instruction as a mixture of uppercase and lowercase characters, the program handles them. Also, writing invalid instruction or writing wrong register names are also handled by the program. But any instruction other than the mentioned ones will not be recognized by the assembler. Besides, only usable registers are **$s1 , $s2 , $s3, and $ac.**

Manual:

To run the program, one needs to open the application file called "Assembler1" which is provided in the folder. If one wants to see the code then open the project file from the provided folder*, it is absolutely necessary that the folder which is containing the program, has a text file **(.txt)** called "**input**"*. This is the file from where the assembler reads the assembly codes. The program reads the code from **"input.txt"** file and writes the corresponding binary code in a text file called **"output".** I 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.