North South University

Department Of Electrical and Computer Engineering

Topic : **Assembler Design**

Course Name: Computer Organization and Architecture

Instructor: Dr. Tanzilur Rahman

Section: 02

Group : 04

Group members:

Name	ID
Md Tahmid Ahmed Rakib	2021179642
Mojahidul Islam	2013330642
Shahporan Hosen Shanto	2021192642

Introduction:

In this assignment we designed an assembler for our designed ISA. We had to make a plan and design an assembler which would take assembly code input from a text file, Analyse it and save the output as hexadecimal into another text file. We have used C++ as a higher level language to build our assembler. Example with a high level C++ code

For Loop Execution:

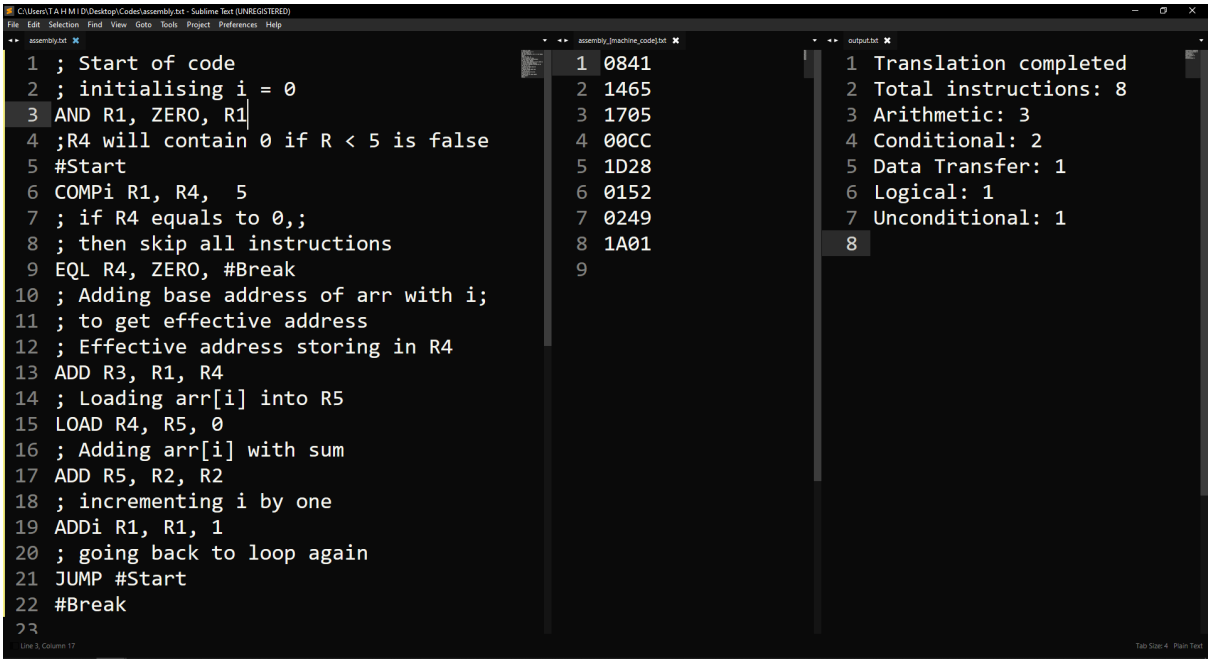
```
for(i = 0; i < 7; i++){
      sum += arr[i];
}
```

[Registers R1 = i, R2 = sum , R3 = arr(base)]

Assembly Level translation BLUE: Operation, Gray: Comment, Brown: Label, Black: operands or immediate

```
; Start of code.  initialising i = 0
AND R1, ZERO, R1
; R4 will contain 0 if condition R < 7 is false
#Start
COMPi R1, R4, 7
; if R4 equals to 0, then branch to #Break
EQL R4, ZERO, #Break
; Adding base address of arr with i to get effective address  Effective
; address of arr[i] saving in R4
ADD R3, R1, R4
; Loading arr[i] into R5.  R4 is effective address, so offset is zero
LOAD R4, R5, 0
; Adding arr[i] with sum
ADD R5, R2, R2
; incrementing i by one
ADDi R1, R1, 1
; going back to loop again
JUMP #Start
#Break
```

Input File	Output File	Console Output / stdout



How it works:

First the assembler looks for an input file. By default it is *assembly.txt* in the working directory. It reads the input file line by line and moves forward. The assembler converts every valid keyword in the instruction that is defined into our designed ISA into a defined Binary value. For example ADD becomes 00000. It takes all the arguments and operands, converts them into binary and generates a 14 bit valid binary instruction. After that it converts every 14 bit binary instruction into a Hexadecimal number as we are instructed. After conversation it creates an output text file (overwrites existing file) naming it inputFileName_[machine_code].txt and writes all instructions into the output file.

Assumptions:

We made some assumptions to make our work easy and bug free. One of the most difficult challenges we faced is to keep track of labels. So to make things easier we assumed that **every label will start with a hash (#) character**. For example **#LABEL1 is a label** but **LABEL is not** a label. Another hard task is to read a code that has been written before or by someone else. Comments become life saving in that situation. So we assumed that **a line that starts with a semicolon(;) is a comment** and will not be processed.

; this line is a comment

This line is not a comment

Additional Features:

Error Detection:

Our assembler has an error detection feature that can detect meaningless or invalid arguments and terminates the process indicating what causes the error.

Example1:

Input File	Output File	Console Output / stdout
		

Example 2:

Input File	Output File	Console Output / stdout
		

Example 3:

Input File	Output File	Console Output / stdout
		

Comments inside code:

Comments are very helpful for understanding codes later. So, we implemented a feature to write comments inside our assembly code and our assembler will skip that entire line. **Comment instruction starts with a semicolon ';'**

; This is a comment
This is not a comment

Example:

Input File	Output File	Console Output / stdout

```
; Adding value
ADD R1, R2, R3
; Subtracting value
SUB R1, R2, R4
```

```
0053
0454
```

```
Translation completed
Total instructions: 2
Arithmetic: 2
```

Limitations:

Due to our limited knowledge and time, we couldn't make everything cooked to perfection. Like our assembler thinks space (' ') and comma (',') are the same and treats them as spacing So if there are multiple comma or zero comma between two operands it will work on. It just checks for valid keywords and blindly translates them into machine code.

Example:

Input File	Output File	Console Output / stdout

```
ADD R1,, R2 R3
```

```
0053
```

```
Translation completed
Total instructions: 1
Arithmetic: 1
```

END OF REPORT