CO PROJECT-1

12 Bit Two-Pass Assembler

Himanshu Raj (2018038)
Ishaan Arora (2018041)

Introduction: This project is about assembler. We need to create a 12-bit two-pass assembler for converting assembly code to machine code. We are required to report errors (if any) encountered during the process.

Memory: As it is a 12-bit assembler and we have dedicated 4 bits to opcode so we have 8 bits for memory. So, we can use 0-255 to allot memory.

General format :

[label] Opcode Operand[operand…..] [Comment]

Square brackets indicate optional content

When more than one operand is to be specified then commas are used as separators.

Loading and Storing:

SAC (store accumulator contents into memory): It will store the current contents of the accumulator into the memory. Where the first byte in 12 bit is opcode and the other is the address.

LAC(load into accumulator from address): It loads the accumulator from the memory. (All the data will be copied to accumulator from memory.)

INP(Read from the terminal and put in address): This will lead to reading the contents from the terminal and then putting it in a specific address which is pre-decided.

STP(stop execution): It is the logical end of the main program.

Eg: 15 STP: This means at statement number 15 STP statement can be branched.

DSP (display value stored in an address on the terminal): We will be taking the value stored in a particular address and then would be displayed in the terminal.

CLA (clear accumulator): This instruction will clear our accumulator of all data and accumulator will be ready to next data.

Arithmetic Operations:

ADD: It will add the data from a given memory address and from the accumulator and will load into accumulator itself.

MUL: It will multiply the data from a given memory address and from the accumulator and will load into accumulator itself.

SUB: It will subtract the data from a given memory address and from the accumulator and will load into accumulator itself.

DIV: It will divide the data from a given memory address and from the accumulator and will load into accumulator itself.

Branching:

BRZ(Branch to address if accumulator value is
Zero): This will make the program counter branch to a particular address if the accumulator contains zero.

BRP(Branch to address if accumulator value is positive): This will make the program counter branch to a particular address if the accumulator contains a positive value.

BRN(Branch to address if accumulator value is negative): This will make the program counter branch to a particular address if the accumulator contains a negative value.

START: It should be the first instruction of our executable source code. Since it indicates the memory address from which we will be starting our location counter. It will decide where our control section will begin.

END: The assembler needs End directive to indicate that we are at the end of our source file. If END is reached during the first pass without getting any errors, then this means the second pass will begin. It will basically end the group declaration.

ERROR HANDLING

The following errors are handled by our assembler-

1. Memory Limit Error- The overflow caused due to more instructions and variables stored than the limit. The limit for our assembler is 256 (0-255).
2. No Instruction Provided- This error is due to no instruction provided in a line. Example- A B.
3. Multiple Instructions are given- Multiple instructions provided with no sense. Example- INP INP A
4. Not enough variable provided- This is due to non-availability of variables in an instruction where it is required. Example- INP
5. Label not provided when branching- This happens in branching type of instructions where we don’t provide a label for branching but it’s required. Example- BRP

6. Label Not Defined- When a label is not declared in the whole program. It will show an error when it'll check for "forward referencing" of the label.
7. Variable Not Defined- Using variable for arithmetic without declaring it with INP or SAC.
8. Label Already Declared- The same label cannot be declared twice in assembly code.
9. Label Already declared as Variable- A variable is stored with the same name of the label we are trying to create.
10. Variable Already declared as Label- A label is stored with the same name of the variable we are trying to create.
11. Branching to Variable- When a Branching instruction is provided with variable and not with a label. Example- BRP A where A is variable.
12. Providing Label instead of Variable- Example- INP L1 where L1 is Label.

How does our program works:

It will be working on Two-pass assembler. The first pass will be creating instructions and in second pass one, we will be setting our addresses.

First Pass: We will be reading the entire source program and will report an error (if any) encountered during the first pass. We will also be looking for labels and variables definition. Then symbols are collected and they are assigned an address. Then we make a symbol table and place the symbols (variable and label) in the table in the first pass, and at the end of the table will contain all the symbols that are used and defined in the program. For assigning the address to each label we will be maintaining a Location counter(LC). We will also check the label definition via forward referencing if it addresses is not allotted till then.

Second Pass: Instruction is read again and assembled by using a symbol table we made in the first pass. Following one line at a time, assembler goes through the program and will be generating machine code for that instruction. We made two files named MachineCode.txt and SymbolTable.txt where we write machine code and make symbol table respectively. Then after reading that line assembler goes to the next instruction and in this way whole program is converted to machine code. It will also process the assemble directives and write the object program and assembly listing.