**Name : Ahmed Ali Abd EL Megiud ID : 05**

**Name : Sherif Mohamed Mostafa ID : 20**

**Name : Hazem Morsy hassan ID : 16**

**Name : Bassam Rageh Ibrahim ID : 13**

system programming

phase 1

***Requirement Specifications:***

It is required to implement pass1 assembler :

* Pass1 is executed by entering the source file name.
* The parser is to handle the lines of the input file which may be instructions , comments lines , directives and storage declarations.
* The parser can handle all storage directives(ORG,BASE,BYTE,WORD,…….)
* The parser is capable of handling the following instructions:

a)2-byte with 1 or 2 symbolic registers.

b) RSUB (ignoring any operand or perhaps issuing a warning)

c) 3-byte PC-relative or (absolute) with symbolic or (non-symbolic) operand to include immediate, indirect, and indexed addressing.

d) 4-byte absolute with symbolic or non-symbolic operand to include immediate, indirect, and indexed addressing.

The output of this pass:

* Symbol table if this pass has been completed successfully without any errors.
* Index and address of each line.
* Error after each invalid line of the input file.

***Language :***

Program is implemented in c++ language.

***Design :***

The Main function of program is pass1 function of type void:

* The program skip any empty line and print the comment lines directly in file until reaching the first instruction .
* Then , it begins parsing the first instruction , if it contains start directive with valid line then the location counter is set to be the operand value , else it sets the location counter to zero and print this line in the output file with meaning full error message if it contains any errors.
* Now , it tries to read the next instruction line , if it is a valid line then we try to add its label to the symbol table, else it is printed directly in the output file with error message.
* After that , if the label is added to symbol table then we have to check the validity of the operation or the directive , else it is printed directly in the output file with error message.
* Then , if it (operation or directive ) is valid then we check the operand , else it is printed directly in the output file with error message.
* Finally if the operand is valid then we add the location counter by specific value according to its format if it is an operation or according to operand if it is a directive, else it is printed directly in the output file with error message.

***Main Data Structures and structs :***

**Structs:**

Data : it holds the data of each successfully parsed line

* Label.
* Operation.
* Operand.

operation: it holds the data of each operation

* Mnemonic : name of the operation .
* Operation code
* Format : it has three values

1. (2) for register to register operations (such as ADDR, SUBR,…..).
2. (3) for memory to register or register to memory or register to register operations (such as LDA,STA,RSUB,….)
3. (4) operations of format (3) but preceded by (+).

* Number of allowed operands.

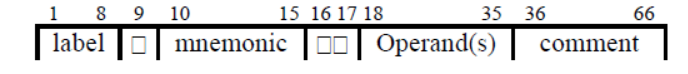
**Maps:**

Map<string , integer> called sym\_tab : it stores the successfully parsed labels(string) as the key and the address of this label (integer)as the value.

Map<string,operation> called op\_tab : it stores all the allowed operation’s mnemonic as the key and the operation struct of this operation as the value.

***Main Algorithms and Functions:***

**Parsing** : it is a Boolean function which parses the line has been read from the file and check its validity , if it is a valid line then the function returns true and divide the line and set the variables of struct data(label , operation, operand ) else it returns false.

in fixed format the line is considered valid if :

* It contains spaces at the locations 9,16,17.
* Any of these fields doesn’t contain space at first or mid of the field except for the operand of directive (byte).
* If operand field doesn’t contain strange characters except for the prefix (‘@’,’#’)
* Operation field may start with ‘+’.

**add to symtab:** it is a boolean function that checks the validity of label of every successfully parsed line. If the label is valid then it is inserted in the sym\_tab and the function returns true , else the function returns false.

label is considered valid if:

* it is not a name of register or a mnemonic of an operation stored in op\_tab.
* it is not previously declared (inserted in sym\_tab.

**Get\_operation:** it is a Boolean function that checks the validity of operation of every successfully parsed line. If the operation is valid then it assigns a global variable called current\_operation to this correct operation of the op\_tab and returns true , else it returns false.

operation is considered valid if:

* it is found in op\_tab which holds all operations supported in assembly.

**Validate\_operand:** it is a Boolean function that checks the validity of operand of every successfully parsed line. If the operand is valid then it returns true , else it returns false.

Operand is considered valid if:

* it has the same format and number of operand as that for the current operation.
* It doesn’t contain both prefix(‘@’,’#’) and indexing.
* For format (2) , all operands should be registers.

**Increment\_loc:** it is a function of type void that responsible for handling directives and setting their location counter.

* Directives (ORG,END,LTORG,BASE, NOBASE) should not have a label.
* Directives (NOBASE,LTORG) shouldn’t have an operand.
* Directives (BASE,EQU,ORG) should have an operand.
* Directive (EQU) should have a label and operand which may be numerical,’\*’ or backward refrenced.
* Directive (ORG) should have an operand which should be ‘\*’ or backward referenced label

if it is ‘\*’ then ,the location counter remains with no change else it takes the address of this backward referenced label that stored in sym\_tab

* Directive (RESW) : it’s operand should be numerical value and increase the location counter by triple of this value.
* Directive (RESB) : it’s operand should be numerical value and increase the location counter by of this value.
* Directive (BYTE) : it’s operand should be in one form of these:
  + - 1. for characters c’test string ’ (15 characters maximum) : location counter increases by number of characters.
      2. for Hexa X’A123F’ (14 characters maximum) : the location counter increases by (number of characters / 2)
* Directive (WORD) : it should contain numerical value and may be preceded by prefix(‘@’,’#’) and it increases the location counter by three.

***Errors handeled :***

* Limit exceeded for labels or operations
* Labels can’t start with a number.
* Misplaced label
* Invalid label.
* Missing operation
* Invalid operation
* Misplaced operand
* Invalid operand
* Label is declared more than once
* Label can’t have the name of register
* Label can’t have the name of operation
* Symbolic operands can’t start with a number
* It is not allowed to use indexed addressing with immediate or indirect addressing.
* Invalid number of operands
* all operands must be registers (for format 2)
* operand must be a label or a memory address (for format 4 and 3)
* Directives that must not have a label (LTORG, NOBASE)
* Directives that must have a label (EQU)
* Directives that must have an operand (EQU,BASE,ORG,….)
* Directives that must not have an operand (NOBASE,LTORG)
* Invalid operand format (for byte directive)
* Invalid hexa string (for byte directive)
* Odd number of hexa characters (for byte directive)
* Out of limit (for byte directive)
* Invalid characters in operand (for RESW and RESB directives)
* Invalid operand with org
* Invalid operand with equ
* Missing end statement or repeated end.

***Division of Work :***

Bassam : reading from and writing in files

Ahmed : Directives and pass1

Sherif : handling operations , operands and labels ( symbol table and operation table).

Hazem : parsing , pass1 and directives.

***Assumptions :***

* Instructions are in fixed format
* The output file which contains the instructions , errors and symbol table is called (listfile.txt)
* Literals and expressions are not supported.
* File directory must not contain any spaces.

***Sample runs :***

