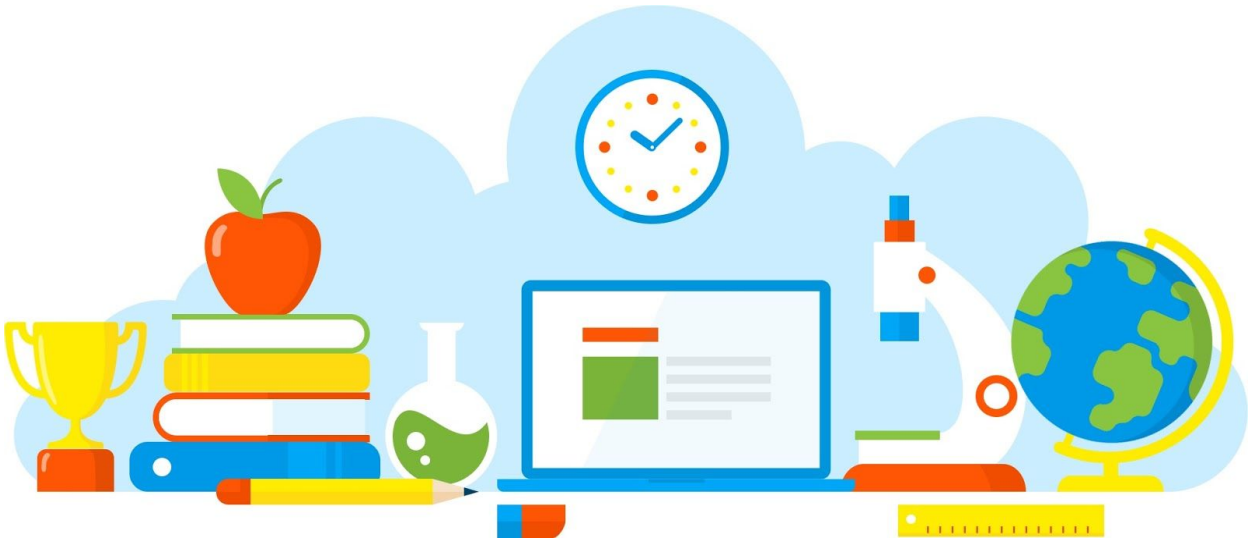

Assembler

Pass 2 Report

Rita samir 26
Ahmed hesham 10
Mohamed Kamal 59
Mohamed el-maghraby mohamed 55

Requirements specifications :

a) The assembler is to execute by entering assemble <source-file-name>.

b) The source file for the main program for this phase is to be named assemble.cpp.

c) The output of the assembler should include (at least):

1. Object-code file whose format is the same as the one described in the text book in section 2.1.1 and 2.3.5.

2. A report at the end of pass2. Pass1 and Pass2 errors should be included as part of the assembler report, exhibiting both the offending line of source code and the error.

d) The assembler should support:

1. EQU and ORG statements.

2. Simple expression evaluation. A simple expression includes simple (A <op> B) operand arithmetic, where <op> is one of +,-,*,/ and no spaces surround the operation, eg. A+B.

3. Literals (Including LTORG)

=C'<ASCII-TEXT>', =X'HEX-TEXT', =<DECIMAL-TEXT> forms.

Design :

The main design consist of 5 main modules parsing ,validation, executing the pass1 algorithm, executing pass 2 algorithm & printing the output

- The parsing module extract the components of each line.
- The validating module validate the components of every line.
- The pass2 module generate the object code for each line and print it in a file.
- Then executing the algorithm and printing the output.

Main data structures :

- Arrays & Lists & vectors : contain the components of every line sequenchelly.
- Maps : made one to one mapping between the operation code & their hexadecimal values & formats.
- Created data structures LIne & Rows : carry the values of the line components.

Algorithms description :

Algorithm Pass 2
(Figure 2.4(b), pp.54)

Pass 2:

```
begin
   read first input line {from intermediate file}
   if OPCODE = 'START' then
      begin
         write listing line
         read next input line
      end {if START}
   write Header record to object program
   initialize first Text record
   while OPCODE ≠ 'END' do
      begin
         if this is not a comment line then
            begin
               search OPTAB for OPCODE
               if found then
                  begin
                     if there is a symbol in OPERAND field then
                        begin
                           search SYMTAB for OPERAND
                           if found then
                              store symbol value as operand address
                           else
                              begin
                                 store 0 as operand address
                                 set error flag (undefined symbol)
                              end
                        end {if symbol}
                     else
                        store 0 as operand address
                     assemble the object code instruction
                  end {if opcode found}
               else if OPCODE = 'BYTE' or 'WORD' then
                  convert constant to object code
               if object code will not fit into the current Text record then
                  begin
                     write Text record to object program
                     initialize new Text record
                  end
               add object code to Text record
            end {if not comment}
         write listing line
         read next input line
      end {while not END}
   write last Text record to object program
   write End record to object program
   write last listing line
end {Pass 2}
```

Assumptions (if any) :

○ Add comments in free format preceded with “.”

Pass1 and Pass2 errors :

Pass 1 errors :

1. Make sure if the label already exists in the symTable or not.

```
bool found = checkIfSymbolDefinedBefore(row.getLabel());//return true if exist
if (found) {
    listFile.at(index).hasError = true;
    listFile.at(index).errorMessge = "The Label already exists";
```

2. Checking that equ doesn’t allow forward reference.

```
if (row.getop_code().compare("equ") == 0) {
    if(row.hasError){
        listFile.at(index).hasError=true;
        listFile.at(index).errorMessge=row.errorMessge;
        goto h;
    }
    string label = row.getOperand();
    int num = atoi(label.c_str());
    stringstream str;
    str << num;
    if (str.str().size() == label.size()) {
        symTab.insert(pair<string, string>(row.getLabel(), decimalToHex(num)));
        TypeTable.insert(pair<string, string>(row.getLabel(), "r"));

    } else if (symTab.count(label)) {
        symTab.insert(pair<string, string>(row.getLabel(), symTab.at(label)));
        TypeTable.insert(pair<string, string>(row.getLabel(), "r"));

    } else {
        listFile.at(index).hasError = true;
        listFile.at(index).errorMessge = "Not defined label, may be forward ref";
    }
```

3. In case of Byte directive ,The hex. Value must has an even numbers of digits.

```
case 'x': {
    if (((row.getOperand().size() - 3) % 2) == 0) {
        stringstream str;
        str << (row.getOperand().size() - 3) / 2;
        LOCCTR = addHex(LOCCTR, str.str());
    } else {
        listFile.at(index).hasError = true;
        listFile.at(index).errorMessge = "The hexidecimal value has an odd numbers of digits";
    }
}
    break;
```

4. If the op_code doesn't exist in the optable then set it as error

```
    }else {
        listFile.at(index).hasError = true;
        listFile.at(index).errorMessge = "Invalid op-code";
    }
```

5. In equ , it can’t be format four

```
if (row.getop_code().compare("equ") == 0) {
    if(row.format!=4){
        LOCCTR = addHex(LOCCTR, "-3");
        if(row.isExpression){
            row=calculateExpression(row);
            if(row.hasError){
                listFile.at(index).hasError=true;
                listFile.at(index).errorMessge=row.errorMessge;
            }                       stringstream str;
            str << hexToDecimal(row.getOperand());
            listFile.at(index).setOperand(str.str());
            row=listFile.at(index);
        }
        string label = row.getOperand();
        int num = atoi(label.c_str());
        stringstream str;
        str << num;
        if (str.str().size() == label.size()) {
            listFile.at(index).setAddress(decimalToHex(num));
        } else if (symTab.count(label)) {
            listFile.at(index).setAddress(symTab.at(label));
        } else {
            listFile.at(index).hasError = true;
            listFile.at(index).errorMessge = "Not defined label, may be forward ref";


        }
    }else {
        listFile.at(index).hasError=true;
        listFile.at(index).errorMessge="+ before equ";
        LOCCTR = addHex(LOCCTR, "-4");
    }
```

6. If the program doesn’t have an End directive.

```
if (listFile.at(listFile.size()-1).getop_code().compare("end")!=0) {
    Row r;
    r.hasError = true;
    r.errorMessge = "NO End Statement";
    listFile.push_back(r);
```

7. With equ directive in expressions, There isn't forward reference, All labels must be defined, also in pass 2 all labels must be in symTable.

```
if (symTab.find(partOfExpression) == symTab.end()) {
    bool flag = false;
    for(int j=0;j<partOfExpression.length();j++){
        if(isalpha(partOfExpression.at(j))){
            flag = true;
            break;
        }

    }
    if(flag){
        row.hasError = true;
        row.errorMessge = "expression has un defined label ";
        return row;
```

8. If numbers of terms in expressions less than three, It must be invalid one.

```
if(expressionList.size()<3){
    for(int ii=0;ii<expressionList.size();ii++){
        cout<< expressionList.at(ii)<<" ";
    }
    row.hasError = true;
    row.errorMessge = "expression  syntax error ";
    return row;
```

9. Relative expression can't be divided or multiplicated.

```
if(TypeTable.at(expressionList.at(i-1)) == "r" || TypeTable.at(expressionList.at(i+1)) == "r" ||
   TypeTable.at(expressionList.at(i-1)) == "Nr"||TypeTable.at(expressionList.at(i+1)) == "Nr")
{
    row.errorMessge = " Relative don't use * or /  operations";
    row.hasError = true;
    return row;


}else{
```

10. It is invalid expression, If it is addition of two relative terms.

```
if((TypeTable.at(expressionList.at(i-1)) == "r" && TypeTable.at(expressionList.at(i+1)) == "r") ||
   (TypeTable.at(expressionList.at(i-1)) == "Nr" && TypeTable.at(expressionList.at(i+1)) == "Nr")||
   (TypeTable.at(expressionList.at(i-1)) == "r" && TypeTable.at(expressionList.at(i+1)) == "Nr" ||
    (TypeTable.at(expressionList.at(i-1)) == "Nr" && TypeTable.at(expressionList.at(i+1)) == "r")
   )){

    row.hasError = true;
    row.errorMessge = "relative + relative is invalid expression";
    return row;
```

11. It is invalid expression, If it is subtraction of absolute and relative terms.

```
if(expressionList.at(i)=="-"){

   if((TypeTable.at(expressionList.at(i-1)) == "a" && TypeTable.at(expressionList.at(i+1)) == "r") ||
      (TypeTable.at(expressionList.at(i-1)) == "Na" && TypeTable.at(expressionList.at(i+1)) == "Nr")||
      (TypeTable.at(expressionList.at(i-1)) == "a" && TypeTable.at(expressionList.at(i+1)) == "Nr" ||
       (TypeTable.at(expressionList.at(i-1)) == "Na" && TypeTable.at(expressionList.at(i+1)) == "r")
      )){

       row.hasError = true;
       // cout<< "e7m" << TypeTable.at(expressionList.at(i-1)) <<" " << expressionList.at(i-1) ;
       row.errorMessge = "absolute - relative  is invalid expression";
       return row;
```

12. Literals can be only written by W or X or C .

```
if(listFile.at(index).getOperand().at(0)=='='){
    litLine line ;
    line.setLiteral(listFile.at(index).getOperand());
    //set Length and value of literal
    if(listFile.at(index).getOperand().at(1)=='c'||listFile.at(index).getOperand().at(1)=='C'){
        line.setLength(decimalToHex(listFile.at(index).getOperand().size()-4));
        string str=listFile.at(index).getOperand();
        string hexStr="";
        for(std::string::size_type i = 3; i < str.size()-1; ++i) {
            hexStr+=decimalToHex(int(str[i]));
        }
        line.setValue(hexStr);
    }else if (listFile.at(index).getOperand().at(1)=='x'||listFile.at(index).getOperand().at(1)=='X'){
        if(((listFile.at(index).getOperand().size() - 4) % 2) == 0) {
            line.setLength(decimalToHex((listFile.at(index).getOperand().size()-4)/2));
            string str = listFile.at(index).getOperand().substr(3,hexToDecimal(line.getLength())*2);
            line.setValue(str);
        } else{
            listFile.at(index).hasError = true;
            listFile.at(index).errorMessge = "The hexidecimal value has an odd numbers of digits";
        }
    }else if(listFile.at(index).getOperand().at(1)=='w'||listFile.at(index).getOperand().at(1)=='W'){
        line.setLength(decimalToHex((listFile.at(index).getOperand().size()-4)*3));
        string str = listFile.at(index).getOperand().substr(3,listFile.at(index).getOperand().size()-4);
        line.setValue(str);
    }else{
        listFile.at(index).hasError=true;
        listFile.at(index).errorMessge="The literal is not defined";
    }
```

Pass 2 errors :

1. If pc relative exceeds the bounds , We use base relative but the user must define the value of base register otherwise it sets to be zero.

```
int disp = addr - pc;
if (!(disp < 2024 && disp >= -2024) && base != -1) {
    //using base
    disp = addr - base;
    obj |= (1 << 2);
}else if(!(disp < 2024 && disp >= -2024)){
    disp = 0;
} else {
    obj |= (1 << 1);
}
```

Sample runs :

1.Test case:

```
STRCP2  START   1000
FIRST   LDT     #11
        LDX     #0
MOVECH  LDCH    STR1,X
        STCH    STR2,X
        TIXR    T
        JLT     MOVECH
STR1    BYTE    C'TEST STRING'
STR2    RESB    11
END     FIRST
```

Output:

```
Hstrcp2001000000027
T0010001c75000b05000053a00857a010b8503b2ff5746573742073747269 6E67
E001000
```

2.TestCase:

```
QUIZ2    START   0
FIRST    LDA     #3
         STX     THREE
         LDX     #0
        +LDS     THREE
         ADDR    A,X
        +STA     RESULT,X
RESULT   RESW    1
THREE    RESW    1
         END     FIRST
```

Output:

```
Hquiz2 000000000019
T000000130100031320100500006f10001690010f900013
M00000a05
M00001005
E000000
```

3.testCase:

Reference’s example at page 45

Output:

```
Hcopy  000000001077
T0000001d17202d6b202d4b1010360320262900003320074b10105d3f2fec032010
T00001d130f20160100030f200d4b10105d3e2003656F66
T0010361db410b400b44075101000e32019332ffadb2013a00433200857c003b850
T0010531d3b2fea1340004f0000f1b410774000e32011332ffa53c003df2008b850
T001070073b2fef4f000005
M00000705
M00001405
M00002705
E001000
```

Expressions examples:

1. General expression evaluation example =>

After calculate the expression we put the result (in hex) instead of it .

```
1     STRCP2  START   1000
2     FIRST   LDT     #11
3             LDX     #0
4     MOVECH  LDCH    STR1,X
5             TIXR    T
6     buffer  equ     first+1000
7     alpha   tixr    t
8     beta    org     1000+movech
9     gamma   lda     1000
10    test    equ     movech+1000+5*10
11            END     FIRST
```

```
line    Address     Label    Opcode  Operands   Comment
0       1000        strcp2   start   1000
1       1000        first    ldt     #11
2       1003                 ldx     #0
3       1006        movech   ldch    str1,x
4       1009                 tixr    t
5       13e8        buffer   equ     5096
6       100b        alpha    tixr    t
7                   beta     org     5102
8       13ee        gamma    lda     1000
9       1420        test     equ     5152
10      13f1                 end     first
```

```
---------------------------------
|          symbol Table         |
---------------------------------
|Label            |   Address   |
---------------------------------
|alpha            |      100b   |
---------------------------------
|beta             |      5102   |
---------------------------------
|buffer           |      13e8   |
---------------------------------
|first            |      1000   |
---------------------------------
|gamma            |      13ee   |
---------------------------------
|movech           |      1006   |
---------------------------------
|strcp2           |      1000   |
---------------------------------
|test             |      1420   |
---------------------------------
```

```
1   .234567890123456789
2   LAB2C    START   0000
3    .        LDA     ALPHA
4            LDB     #10
5            LDX     #0
6            ADDR    A,B
7       org     1000
8     .       STA     SAVEW,X
9
10     .      STA     SAVEW,X
11           LDA     =X'5f'
12           LDA     =X'df'
13           LDA     =C'8j5'
14
15           LDA     =w'1552'
16           LDA     =w'-152'
17           LDA     =W'124'
18           LDA     =W'-152'
19           LDA     =W'1624'
20           LDA     =W'4095'
21      ltorg
22      word     -1000
23      resw     1
24   .       resw     -1
25     .     resw     9999
26  .        resw     10000
27           BAse    alpha
28  .Format 4
29           +SUB    #12
30           LDX     #0
31           LDCH    HEXCHAR
32           STA     INPUT
33  LOOP     LDCH    STRING,X
34  .    lda     test
35  .    lda     none
36  v1   lda     4095
37  .v2       lda     4096
38  v3       lda     #4095
39  .v4       lda     #4096
40  v5       lda     @4095
41  .v6  lda     @4096
42  .v7  lda     -4095
43  ..v8       lda     -4096
```

```
42  .v7   lda     -4095
43  ..v8       lda     -4096
44  .v9       lda     #-4095
45  ..v10      lda     #-4096
46  .v11      lda     @-4095
47  .v12     lda     @-4096
48           COMP    INPUT
49       LDA     1000
50  sajed    EQU     loop-1000
51  b1       equ     4000
52  b11      equ     4000
53  b2       equ     1000
54  b3       equ     2000
55  b5       equ     500
56       BAse    1500
57       LDA     b1
58       nobase
59  .    lda     b11
60       LDA     b3
61           LDA     b2
62       resw    3500
63       BASE    4000
64       resw    3000
65  test     word    5
66      . LDA     b5
67  sajed1   EQU     loop+1000
68  sajed2   EQU     loop*1000
69  sajed3   EQU     loop/7
70  sajed4   EQU     loop-1000+952
71  sajed5   EQU     1000-loop-142
72  sajed6   EQU     loop-sajed1
73  sajed7   EQU     loop-1000
74           JEQ     FOUND
75           STCH    OUTPUT,X
76           TIX     #5
77       .    JLT     LOOP
78       nobaSe
79  FOUND    J       OUTPUT
80  ALPHA    WORD    2
81  SAVEW    RESW    2
82  HEXCHAR  BYTE    X'6145fd56'
83  INPUT    RESB    1
84  beso     BYTE    C'F4   '
85           BYTE    C'F5    '
```

```
71  sajed5   EQU     1000-loop-142
72  sajed6   EQU     loop-sajed1
73  sajed7   EQU     loop-1000
74           JEQ     FOUND
75           STCH    OUTPUT,X
76           TIX     #5
77       .    JLT     LOOP
78       nobaSe
79  FOUND    J       OUTPUT
80  ALPHA    WORD    2
81  SAVEW    RESW    2
82  HEXCHAR  BYTE    X'6145fd56'
83  INPUT    RESB    1
84  beso     BYTE    C'F4   '
85           BYTE    C'F5    '
86
87
88  STRING   RESW    1
89  OUTPUT   RESB    5
90
91           END
92
93
```

line	Address	Label	Opcode	Operands	Com
0	0000	lab2c	start	0000	
1	0000		ldb	#10	
2	3		ldx	#0	
3	6		addr	a,b	
4			org	1000	
5	3e8		lda	=X'5f'	
6	3eb		lda	=X'df'	
7	3ee		lda	=C'8j5'	
8	3f1		lda	=w'1552'	
9	3f4		lda	=w'-152'	
10	3f7		lda	=W'124'	
11	3fa		lda	=W'-152'	
12	3fd		lda	=W'1624'	
13	400		lda	=W'4095'	
14	403		ltorg		
15	403		=X'5f'	5f	
16	404		=X'df'	df	
17	405		=C'8j5'	386a35	
18	408		=w'1552'	1552	
19	414		=w'-152'	-152	
20	420		=W'124'	124	
21	429		=W'1624'	1624	
22	435		=W'4095'	4095	
23	441		word	-1000	
24	444		resw	1	
25	447		base	alpha	
26	447		sub	#12	
27	44b		ldx	#0	
28	44e		ldch	hexchar	
29	451		sta	input	
30	454	loop	ldch	string,x	
31	457	v1	lda	4095	
32	45a	v3	lda	#4095	
33	45d	v5	lda	@4095	
34	460		comp	input	
35	463		lda	1000	
36	6c	sajed	equ	108	
37	fa0	b1	equ	4000	

```
38      fa0         b11       equ     4000
39      3e8         b2        equ     1000
40      7d0         b3        equ     2000
41      1f4         b5        equ     500
42      466                   base    1500
43      466                   lda     b1
44      469
           ****Error: No op_code is exist
45      469                   lda     b3
46      46c                   lda     b2
47      46f                   resw    3500
48      2d73                  base    4000
49      2d73                  resw    3000
50      509b        test      word    5
51      83c         sajed1    equ     2108
52      0           sajed2    equ     0
           ****Error: Relative don't use * or /  operations
53      0           sajed3    equ     0
           ****Error: Relative don't use * or /  operations
54      424         sajed4    equ     1060
55      1000        sajed5    equ     4096
           ****Error:absolute - relative  is invalid expression
56      3e8         sajed6    equ     1000
57      6c          sajed7    equ     108
58      5095                  jeq     found
59      5098                  stch    output,x
60      509b                  tix     #5
61      509e
           ****Error: No op_code is exist
62      509e        found     j       output
63      50a1        alpha     word    2
64      50a4        savew     resw    2
65      50aa        hexchar   byte    X'6145fd56'
66      50aa        input     resb    1
67      50ab        beso      byte    C'F4    '
68      50ab                  byte    c'f5     '
69      50b1        string    resw    1
70      50b4        output    resb    5
71      50b9                  end
```

symbol Table	
Label	Address
alpha	50a1
b1	fa0
b11	fa0
b2	3e8
b3	7d0
b5	1f4
beso	50ab
found	509e
hexchar	50aa
input	50aa
lab2c	0000
loop	454
null	1000
output	50b4
sajed	6c
sajed1	83c
sajed4	424
sajed6	3e8
sajed7	6c
savew	50a4
string	50b1
test	509b
v1	457
v3	45a
v5	45d

```
Hlab2c 0000000050b9
T0000001d69000a05000090030323fb0323fc0323fd03240003240c032418032421
T0004001503242d5fdf386a351552-152124162440955fffffc18
T0004471c1d10000c0500005340090f400953c010034ff4010fff2b4009034f5f
T00046609034eff032364032f79
T00509b12000005332006 57a0192d00053f2013000002
T0050aa046145fd56
T0050ab0b4634202020663520202020
E30312020
```