國立嘉義大學資訊工程學系系統程式期末考考卷

學號：　　姓名：

1. (10%) Please compute the target addresses and values in register A of the following SIC/XE machine codes. And complete the two insreuctions.

I:003030	003600
J:003600	103000
K:006390	00C303
L:00C303	003030

(B) =006000

(PC)=003000

(X) =000090

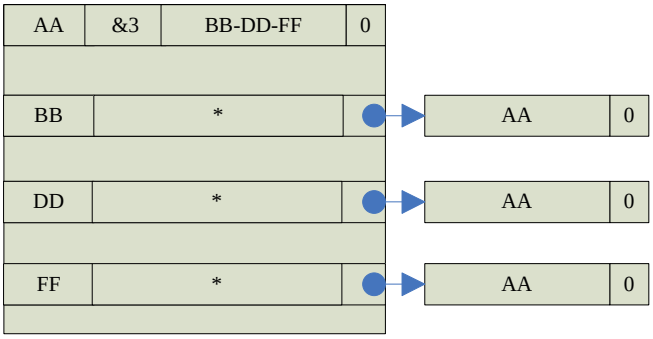
Assembler language	HEX	$(TA)_{16}$	(Value in A)$_{16}$
	036000	X	X
	022030	3030	103000
	032600	3600	103000
LDA @K	024390		003030
+LDA L	0310C303		003030

(hint: the opcode of LDA is 00.)

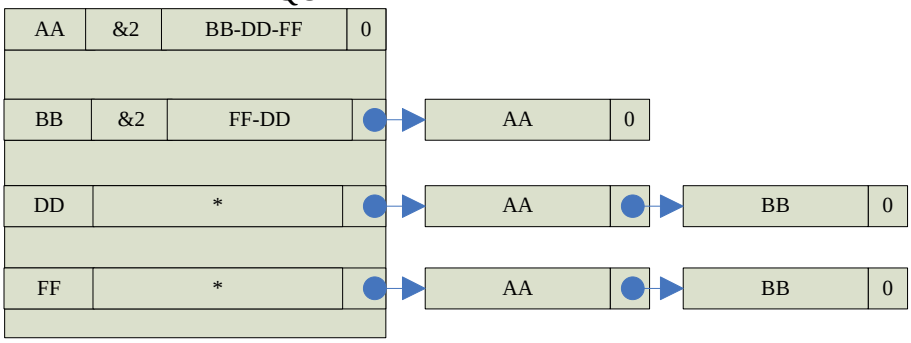
2. (8%)Please show the content of the symbol table when each instruction is read by Multi-Pass assemblers. (依字母由 A 到 D)?

```
AA      EQU     BB – DD - FF
BB      EQU     FF - DD
FF      EQU     4096
DD      EQU     256
```

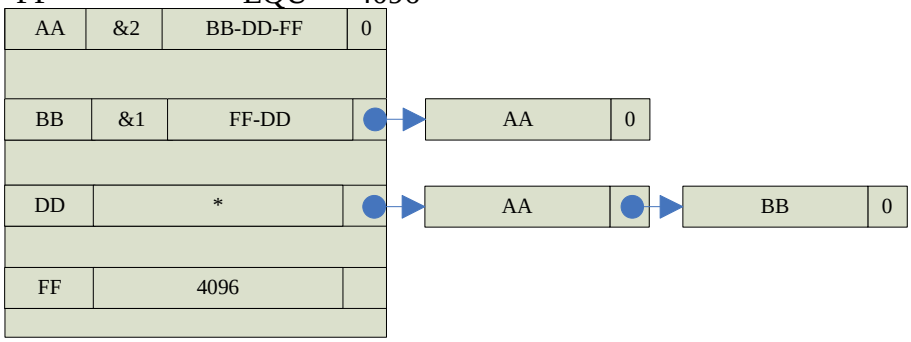
AA EQU BB – DD - FF



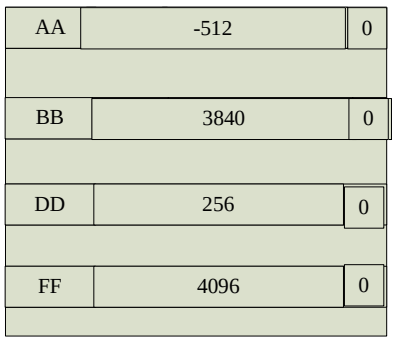
BB EQU FF - DD



FF EQU 4096



DD EQU 256

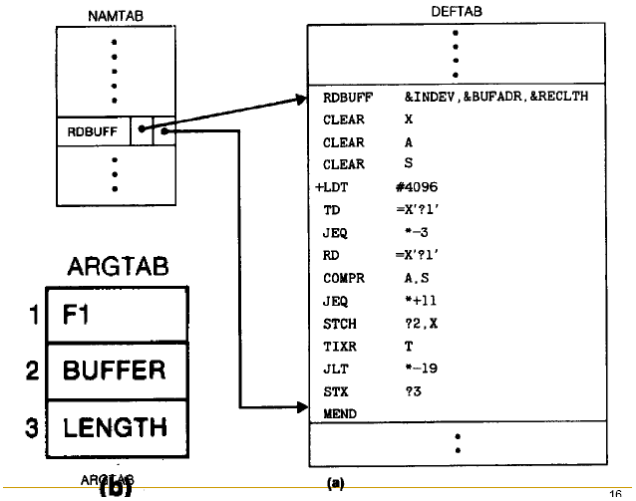


3. (15%)Please describe the behaviors of a two-pass macro processor in pass1 and pass2. What data structures do we need for such behaviors?

(a) All macro definitions are processed during the first pass.

(b) All macro invocation statements are expanded during the second pass.

- The macro definitions are stored in a definition table DFETAB, which contains the macro prototype and the statement that make up the macro body.
- The macro names are entered into NAMTAB, which serves as an index to DFETAB.
  - NAMTAB contains two pointers to the beginning and the end of the definition in DEFTAB
- The third data structure is an argument table ARGTAB, which is used during the expansion of macro invocations.
  - When a macro invocation statement is recognized, the arguments are stored in ARGTAB according to their position in the argument list.
  - As the macro is expanded, arguments from ARGTAB are substituted for the corresponding parameters in the macro body.



4. (6%) B[$l_1..u_1$, $l_2..u_2$] is a two-dimensional integer array. Note that the size of each element of B is w. Please give the relative address of B[s_1, s_2] under row-major order.

$w*[(s_1-l_1)*(u_2-l_2+1) + (s_2-l_2)]$

5. (10%) Please use the macro definition to expand the following macro invocation statement：

RDBUFF F1, BUFFER, LENGTH, (04, 12)

```
25   RDBUFF    MACRO    &INDEV,&BUFADR,&RECLTH,&EOR
27   &EORCT    SET      %NITEMS(&EOR)
30             CLEAR    X              CLEAR LOOP COUNTER
35             CLEAR    A
45            +LDT      #4096          SET MAX LENGTH = 4096
50   $LOOP     TD       =X'&INDEV'     TEST INPUT DEVICE
55             JEQ      $LOOP          LOOP UNTIL READY
60             RD       =X'&INDEV'     READ CHARACTER INTO REG A
63   &CTR      SET      1
64             WHILE    (&CTR LE &EORCT)
65             COMP     =X'0000&EOR[&CTR]'
70             JEQ      $EXIT
71   &CTR      SET      &CTR+1
73             ENDW
75             STCH     &BUFADR,X      STORE CHARACTER IN BUFFER
80             TIXR     T              LOOP UNLESS MAXIMUM LENGTH
85             JLT      $LOOP            HAS BEEN REACHED
90   $EXIT     STX      &RECLTH        SAVE RECORD LENGTH
100            MEND
```

(a)

(a) RDBUFF F1, BUFFER, LENGTH, (04, 12)

```
                CLEAR X
                CLEAR A
                +LDT #4096
$AALOOP         TD =X'F1'
                JEQ $AALOOP
                RD =X'F1'
                COMPR =X'000004'
                JEQ $AAEXIT
                COMPR =X'000012'
                JEQ $AAEXIT
                STCH BUFFER, X
                TIXR T
                JLT $AALOOP
$AAEXIT         STX LENGTH
```

6. (12%) Please describe the operation of the basic compiler functions.
   A. scanner
   B. parser
   C. code generator

- Lexical analysis - scanner
  - Scanning the source statement, recognizing and classifying the various tokens
- Syntactic analysis - parser
  - Recognizing the statement as some language construct.
  - Construct a parser tree (syntax tree)
- Code generation – code generator
  - Generate assembly language codes
  - Generate machine codes (Object codes)

7. (12%) Please draw the parse trees according to the following grammar rules.
   (a) A * 10 + B DIV C – D * E
   (b) A + 10 * B- C DIV D + E

```
1  <prog>         ::= PROGRAM <prog-name> VAR <dec-list> BEGIN <stmt-list> END.
2  <prog-name>    ::= id
3  <dec-list>     ::= <dec> | <dec-list> ; <dec>
4  <dec>          ::= <id-list> : <type>
5  <type>         ::= INTEGER
6  <id-list>      ::= id | <id-list> , id
7  <stmt-list>    ::= <stmt> | <stmt-list> ; <stmt>
8  <stmt>         ::= <assign> | <read> | <write> | <for>
9  <assign>       ::= id := <exp>
10 <exp>          ::= <term> | <exp> + <term> | <exp> - <term>
11 <term>         ::= <factor> | <term> * <factor> | <term> DIV <factor>
12 <factor>       ::= id | int | ( <exp> )
13 <read>         ::= READ ( <id-list> )
14 <write>        ::= WRITE ( <id-list> )
15 <for>          ::= FOR <index-exp> DO <body>
16 <index-exp>    ::= id := <exp> TO <exp>
17 <body>         ::= <stmt> | BEGIN <stmt-list> END
```

Figure 5.2 Simplified Pascal grammar.

8. (8%) Please explain why top-down parsers cannot be directly used with a grammar containing immediate left recursion. Give an example grammar with left recursion and the corresponding one without left recursion.

   An unending chain

   - Two grammar
     - <id-list> ::= id | <id-list>, id
     - <id-list> ::= id { , id }

9. (10%) Please write the object codes for the following statement according to the code generation procedures.

   SquAvr := Alpha * Beta + Delta Div Gamma

   LDA Alpha
   MUL Beta
   STA T1
   LDA Delta
   DTV Gamma
   STA T2
   LDA T1
   ADD T2
   STA SquAvr

10. (4%) Please illustrate two goals of an operating system.

(a) Run programs efficiently
(b) Make the computer easier to use
    i. Provide a user-friendly interface
(c) Improve the efficiency of hardware utilization
    i. Manage the resources of the computer

11. (11%) Please give three kinds of situations which can cause interrupts. And draw the sequence of event that occurs in response to an interrupt.
    (a) The completion of an I/O operation
    (b) The expiration of a preset time interval
    (c) An attempt to divide by zero

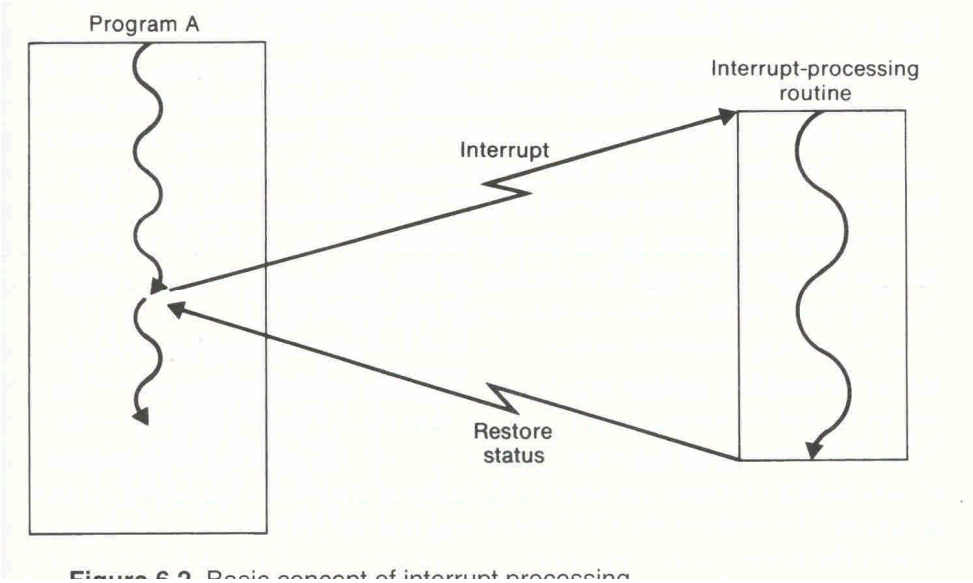


Figure 6.2 Basic concept of interrupt processing

```
<assign> ::= id := <exp>

             GETA ( <exp> )
             generate [STA    S(id)]
             REGA := null


<exp> ::= <term>

             S(<exp>) := S(<term>)
             if S(<exp>) = rA then
                REGA := <exp>


<exp>1 ::= <exp>2 + <term>

             if S(<exp>2) = rA then
                generate [ADD    S(<term>)]
             else if S(<term>) = rA then
                generate [ADD    S(<exp>2)]
             else
                begin
                   GETA (<exp>2)
                   generate [   ADD    S(<term>)]
                end
             S(<exp>1) := rA
             REGA := <exp>1
```

```
<exp>1 ::= <exp>2 - <term>

            if S(<exp>2) = rA then
               generate [SUB    S(<term>)]
            else
               begin
                  GETA (<exp>2)
                  generate [  SUB    S(<term>)]
               end
            S(<exp>1) := rA
            REGA := <exp>1


<term> ::= <factor>

            S(<term>) := S(<factor>)
            if S(<term>) = rA then
               REGA := <term>


<term>1 ::= <term>2 * <factor>

            if S(<term>2) = rA then
               generate [MUL    S(<factor>)]
            else if S(<factor>) = rA then
               generate [MUL    S(<term>2)]
            else
               begin
                  GETA (<term>2)
                  generate [MUL   S(<factor>)]
               end
            S(<term>1) := rA
            REGA := <term>1


<term>1 ::= <term>2 DIV <factor>

            if S(<term>2) = rA then
               generate [DIV    S(<factor>)]
            else
               begin
                  GETA (<term>2)
                  generate [DIV    S(<factor>)]
               end
            S(<term>1) := rA
            REGA := <term>1
```

```
<factor> ::= id

             S(<factor>) := S(id)


<factor> ::= int

             S(<factor>) := S(int)


<factor> ::= (  <exp>   )
             S(<factor>) := S(<exp>)
             if S(<factor>) = rA then
                REGA := <factor>
```

(b)

```
procedure GETA (NODE)
   begin
      if REGA = null then
         generate [LDA    S(NODE)]
      else if S(NODE) ≠ rA then
         begin
            create a new working variable Ti
            generate [STA    Ti]
            record forward reference to Ti
            S(REGA) := Ti
            generate [LDA    S(NODE)]
         end {if ≠ rA}
      S(NODE) := rA
      REGA := NODE
   end {GETA}
```