Example of SIC assembler language program

1 / 4

```
Line      source statement
5         COPY      START   1000        從輸入到輸出
10        FIRST     STL     RETADR      儲存並回傳位址
15        CLOOP     JSUB    RDREC       讀取輸入紀錄
20                  LDA     LENGTH      測試檔案是否到EOF
25                  COMP    ZERO        EOF = 0 ?
30                  JEQ     ENDFIL      EOF = 0則離開
35                  JSUB    WRREC       寫入 輸出紀錄(output record)
40                  J       CLOOP       跳回CLOOP
45        ENDFIL    LDA     EOF         插入 檔案終結符號
50                  STA     BUFFER
55                  LDA     THREE       把EOF長度設爲3
60                  STA     LENGTH
65                  JSUB    WRREC       寫入 EOF
70                  LDL     RETADR      取得回傳位址
75                  RSUB                回到 原呼叫程式(caller)
```

Example of SIC assembler language program

```
80   EOF      BYTE   C'EOF'
85   THREE    WORD   3
90   ZERO     WORD   0
95   RETADR   RESW   1
100  LENGTH   RESW   1
105  BUFFER   RESB   4096          4096 byte 的暫存區
110     .
115     .        SUBROUTINE TO READ RECORD INTO BUFFER
120     .        (呼叫副程式讀取紀錄到暫存區)
125  RDREC    LDX    ZERO          清除迴圈計數器
130           LDA    ZERO          把AX暫存器設爲0
```

Example of SIC assembler language program

```
135   RLOOP    TD     INPUT       測試輸入裝置
140            JEQ    RLOOP       執行迴圈直到輸入資料
145            RD     INPUT       讀取字元到AX暫存器
150            COMP   ZERO        是否爲紀錄結尾(EOR=0)
155            JEQ    EXIT        成立的話離開迴圈
160            STCH   BUFFER,X    儲存字元(X)到暫存區
165            TIX    MAXLEN      執行迴圈直到紀錄的
170            JTL    RLOOP       最大長度
175   EXIT     STX    LENGTH      儲存紀錄長度
180            RSUB               回到原呼叫程式
185   INPUT    BYTE   X'F1'       輸入 裝置的編碼
190   MAXLEN   WORD   4096
```

Example of SIC assembler language program

```
195   .
200   .        SUBROUTINE TO WRITE RECORD INTO BUFFER
205   .          (呼叫副程式寫入紀錄到暫存區)
210   WRREC   LDX    ZERO          清除迴圈計數器
215   WLOOP   TD     OUTPUT        測試輸入裝置
220           JEQ    WLOOP         執行迴圈直到輸入資料
225           LDCH   BUFFER,X      讀取暫存器(X)內容到暫存區
230           WD     OUTPUT        輸出字元
235           TIX    LENGTH        執行迴圈直到所有字元寫完
240           JLT    WLOOP
245           RSUB                 回到原呼叫程式
250  OUTPUT   BYTE   X'05'
255           END    FIRST         輸出 裝置的編碼
```

Object code of SIC assembler language program

Line	Loc	Source statement			object code
5	1000	COPY	START	1000	
10	1000	FIRST	STL	RETADR	141033
15	1003	CLOOP	JSUB	RDREC	482039
20	1006		LDA	LENGTH	001036
25	1009		COMP	ZERO	281030
30	100C		JEQ	ENDFIL	301015
35	100F		JSUB	WRREC	482061
40	1012		J	CLOOP	3C1003
45	1015	ENDFIL	LDA	EOF	00102A
50	1018		STA	BUFFER	0C1039
55	101B		LDA	THREE	00102D
60	101E		STA	LENGTH	0C1036
65	1021		JSUB	WRREC	482061
70	1024		LDL	RETADR	081033
75	1027		RSUB		4C0000

Object code of SIC assembler language program 2 / 4

```
80     102A   EOF      BYTE   C'EOF'     454F46
85     102D   THREE    WORD   3          000003
90     1030   ZERO     WORD   0          000000
95     1033   RETADR   RESW   1
100    1036   LENGTH   RESW   1
105    1039   BUFFER   RESB   4096
110    .
115    .      SUBROUTINE TO HEAD RECORD INTO BUFFER
120    .
125    2039   RECORD   LDX    ZERO       041030
130    203C            LDA    ZERO       001030
```

Object code of SIC assembler language program 3 / 4

```
135   203F   RLOOP    TD      INPUT       E0205D
140   2042            JEQ     RLOOP       30203F
145   2045            RD      INPUT       D8205D
150   2048            COMP    ZERO        281030
155   204B            JEQ     EXIT        302057
160   204E            STCH    BUFFER,X    549039
165   2051            TIX     MAXLEN      2C205E
170   2054            JTL     RLOOP       38203F
175   2057   EXIT     STX     LENGTH      101036
180   205A            RSUB                4C0000
185   205D   INPUT    BYTE    X’F1’       F1
190   205E   MAXLEN   WORD    4096        001000
```

Object code of SIC assembler language program

```
195         .
200         .      SUBROUTINE TO WRITE RECORD INTO BUFFER
205         .
210   2061  WRREC    LDX    ZERO           041030
215   2064  WLOOP    TD     OUTPUT         E02079
220   2067           JEQ    WLOOP          302064
225   206A           LDCH   BUFFER,X       509039
230   206D           WD     OUTPUT         DC2079
235   2070           TIX    LENGTH         2C1036
240   2073           JLT    WLOOP          382064
245   2076           RSUB                  4C0000
250   2079  OUTPUT   BYTE   X'05'          05
255                  END    FIRST
```

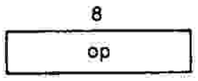
Mnemonic	Format	Opcode	Effect	Notes
ADD m	3/4	18	A ← (A) + (m..m+2)	
ADDF m	3/4	58	F ← (F) + (m..m+5)	X F
ADDR r1,r2	2	90	r2 ← (r2) + (r1)	X
AND m	3/4	40	A ← (A) & (m..m+2)	
CLEAR r1	2	B4	r1 ← 0	X
COMP m	3/4	28	(A) : (m..m+2)	C
COMPF m	3/4	88	(F) : (m..m+5)	X F C
COMPR r1,r2	2	A0	(r1) : (r2)	X C
DIV m	3/4	24	A ← (A) / (m..m+2)	
DIVF m	3/4	64	F ← (F) / (m..m+5)	X F
DIVR r1,r2	2	9C	r2 ← (r2) / (r1)	X
FIX	1	C4	A ← (F) [convert to integer]	X F
FLOAT	1	C0	F ← (A) [convert to floating]	X F
HIO	1	F4	Halt I/O channel number (A)	P X
J m	3/4	3C	PC ← m	
JEQ m	3/4	30	PC ← m if CC set to =	
JGT m	3/4	34	PC ← m if CC set to >	
JLT m	3/4	38	PC ← m if CC set to <	
JSUB m	3/4	48	L ← (PC); PC ← m	
LDA m	3/4	00	A ← (m..m+2)	
LDB m	3/4	68	B ← (m..m+2)	X
LDCH m	3/4	50	A [rightmost byte] ← (m)	
LDF m	3/4	70	F ← (m..m+5)	X F
LDL m	3/4	08	L ← (m..m+2)	
LDS m	3/4	6C	S ← (m..m+2)	X
LDT m	3/4	74	T ← (m..m+2)	X
LDX m	3/4	04	X ← (m..m+2)	
LPS m	3/4	D0	Load processor status from information beginning at address m (see Section 6.2.1)	P X
MUL m	3/4	20	A ← (A) * (m..m+2)	

Mnemonic	Format	Opcode	Effect	Notes
MULF m	3/4	60	F ← (F) * (m..m+5)	X F
MULR r1, r2	2	98	r2 ← (r2) * (r1)	X
NORM	1	C8	F ← (F) [normalized]	X F
OR m	3/4	44	A ← (A) | (m..m+2)	
RD m	3/4	D8	A [rightmost byte] ← data from device specified by (m)	P
RMO r1,r2	2	AC	r2 ← (r1)	X
RSUB	3/4	4C	PC ← (L)	
SHIFTL r1,n	2	A4	r1 ← (r1); left circular shift n bits. {In assembled instruction, r2 = n–1}	X
SHIFTR r1,n	2	A8	r1 ← (r1); right shift n bits, with vacated bit positions set equal to leftmost bit of (r1). {In assembled instruction, r2 = n-1}	X
SIO	1	F0	Start I/O channel number (A); address of channel program is given by (S)	P X
SSK m	3/4	EC	Protection key for address m ← (A) (see Section 6.2.4)	P X
STA m	3/4	0C	m..m+2 ← (A)	
STB m	3/4	78	m..m+2 ← (B)	X
STCH m	3/4	54	m ← (A) [rightmost byte]	
STF m	3/4	80	m..m+5 ← (F)	X F
STI m	3/4	D4	Interval timer value ← (m..m+2) (see Section 6.2.1)	P X
STL m	3/4	14	m..m+2 ← (L)	
STS m	3/4	7C	m..m+2 ← (S)	X
STSW m	3/4	E8	m..m+2 ← (SW)	P
STT m	3/4	84	m..m+2 ← (T)	X
STX m	3/4	10	m..m+2 ← (X)	
SUB m	3/4	1C	A ← (A) – (m..m+2)	
SUBF m	3/4	5C	F ← (F) – (m..m+5)	X F

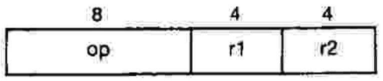
Mnemonic	Format	Opcode	Effect	Notes
SUBR r1,r2	2	94	r2 ← (r2) – (r1)	X
SVC n	2	B0	Generate SVC interrupt. {In assembled instruction, r1 = n}	X
TD m	3/4	E0	Test device specified by (m)	P C
TIO	1	F8	Test I/O channel number (A)	P X C
TIX m	3/4	2C	X ← (X) + 1; (X): (m..m+2)	C
TIXR r1	2	B8	X ← (X) + 1; (X): (r1)	X C
WD m	3/4	DC	Device specified by (m) ← (A) [rightmost byte]	P

Instruction Formats

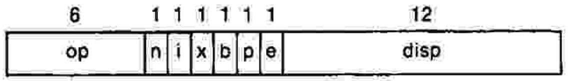
Format 1 (1 byte):



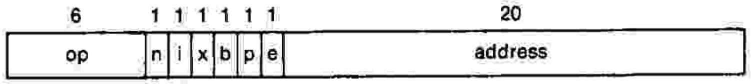
Format 2 (2 bytes):



Format 3 (3 bytes):



Format 4 (4 bytes):



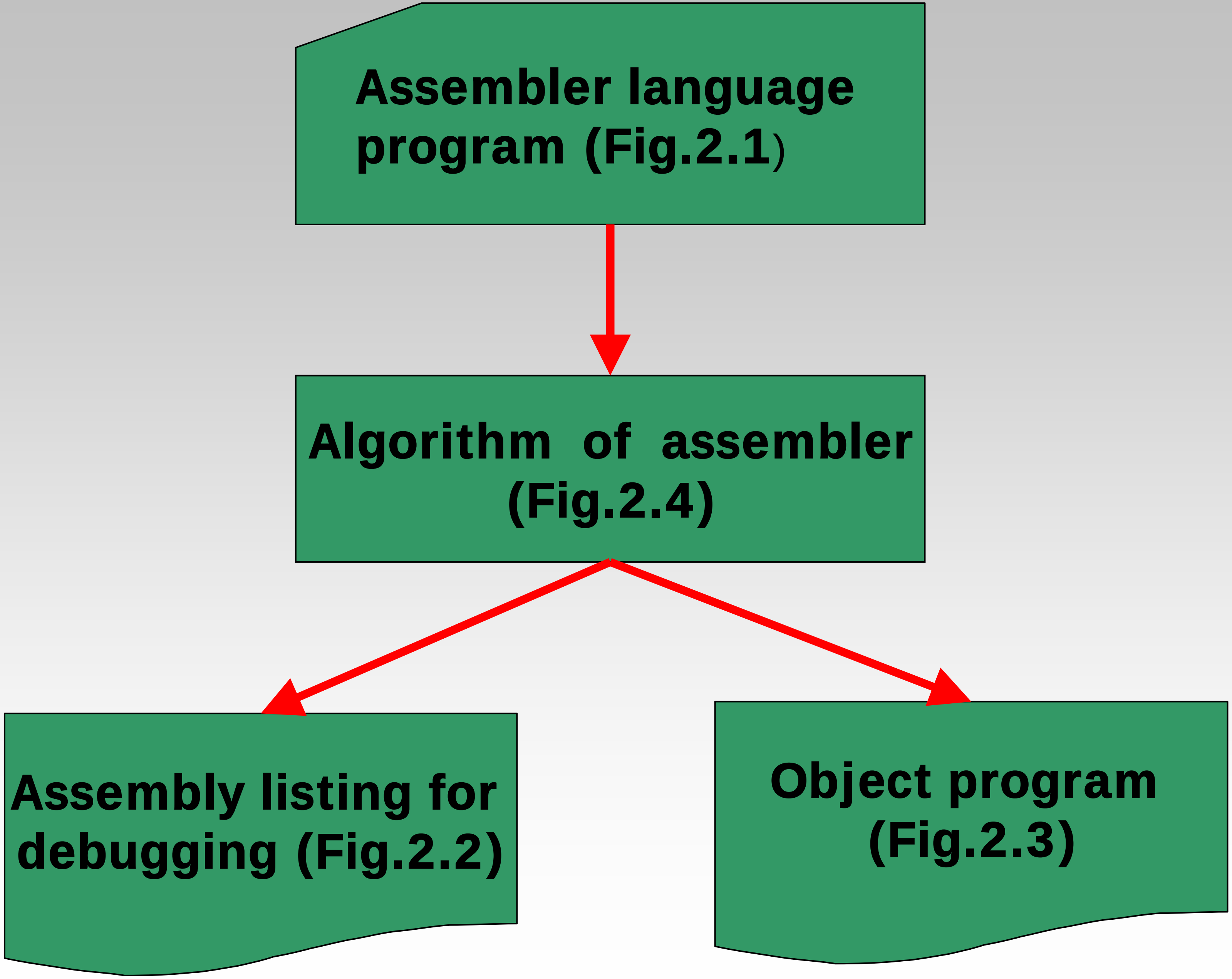
Addressing Modes

The following addressing modes apply to Format 3 and 4 instructions. Combinations of addressing bits not included in this table are treated as errors by the machine. In the description of assembler language notation, *c* indicates a constant between 0 and 4095 (or a memory address known to be in this range); *m* indicates a memory address or a constant value larger than 4095. Further information can be found in Section 1.3.2.

The letters in the Notes column have the following meanings:

4	Format 4 instruction
D	Direct-addressing instruction
A	Assembler selects either program-counter relative or base-relative mode
S	Compatible with instruction format for standard SIC machine. Operand value can be between 0 and 32,767 (see Section 1.3.2 for details).

Addressing type	Flag bits n i x b p e	Assembler language notation	Calculation of target address TA	Operand	Notes
Simple	1 1 0 0 0 0	op c	disp	(TA)	D
	1 1 0 0 0 1	+op m	addr	(TA)	4 D
	1 1 0 0 1 0	op m	(PC) + disp	(TA)	A
	1 1 0 1 0 0	op m	(B) + disp	(TA)	A
	1 1 1 0 0 0	op c,X	disp + (X)	(TA)	D
	1 1 1 0 0 1	+op m,X	addr + (X)	(TA)	4 D
	1 1 1 0 1 0	op m,X	(PC) + disp + (X)	(TA)	A
	1 1 1 1 0 0	op m,X	(B) + disp + (X)	(TA)	A
	0 0 0 - - -	op m	b/p/e/disp	(TA)	D S
	0 0 1 - - -	op m,X	b/p/e/disp + (X)	(TA)	D S
Indirect	1 0 0 0 0 0	op @c	disp	((TA))	D
	1 0 0 0 0 1	+op @m	addr	((TA))	4 D
	1 0 0 0 1 0	op @m	(PC) + disp	((TA))	A
	1 0 0 1 0 0	op @m	(B) + disp	((TA))	A
Immediate	0 1 0 0 0 0	op #c	disp	TA	D
	0 1 0 0 0 1	+op #m	addr	TA	4 D
	0 1 0 0 1 0	op #m	(PC) + disp	TA	A
	0 1 0 1 0 0	op #m	(B) + disp	TA	A


Assembler language program (Fig.2.1)
Algorithm of assembler (Fig.2.4)
Assembly listing for debugging (Fig.2.2)
Object program (Fig.2.3)

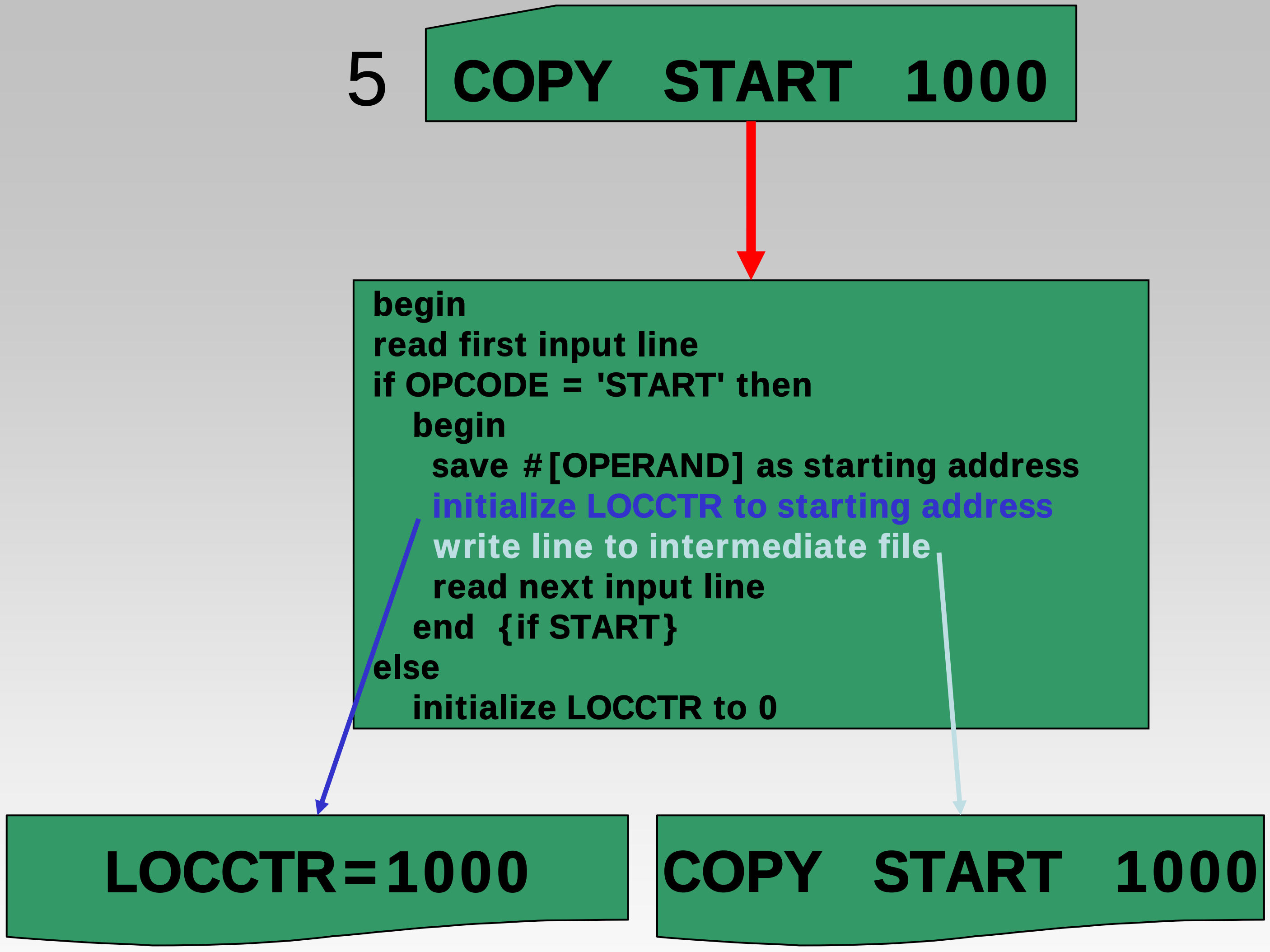
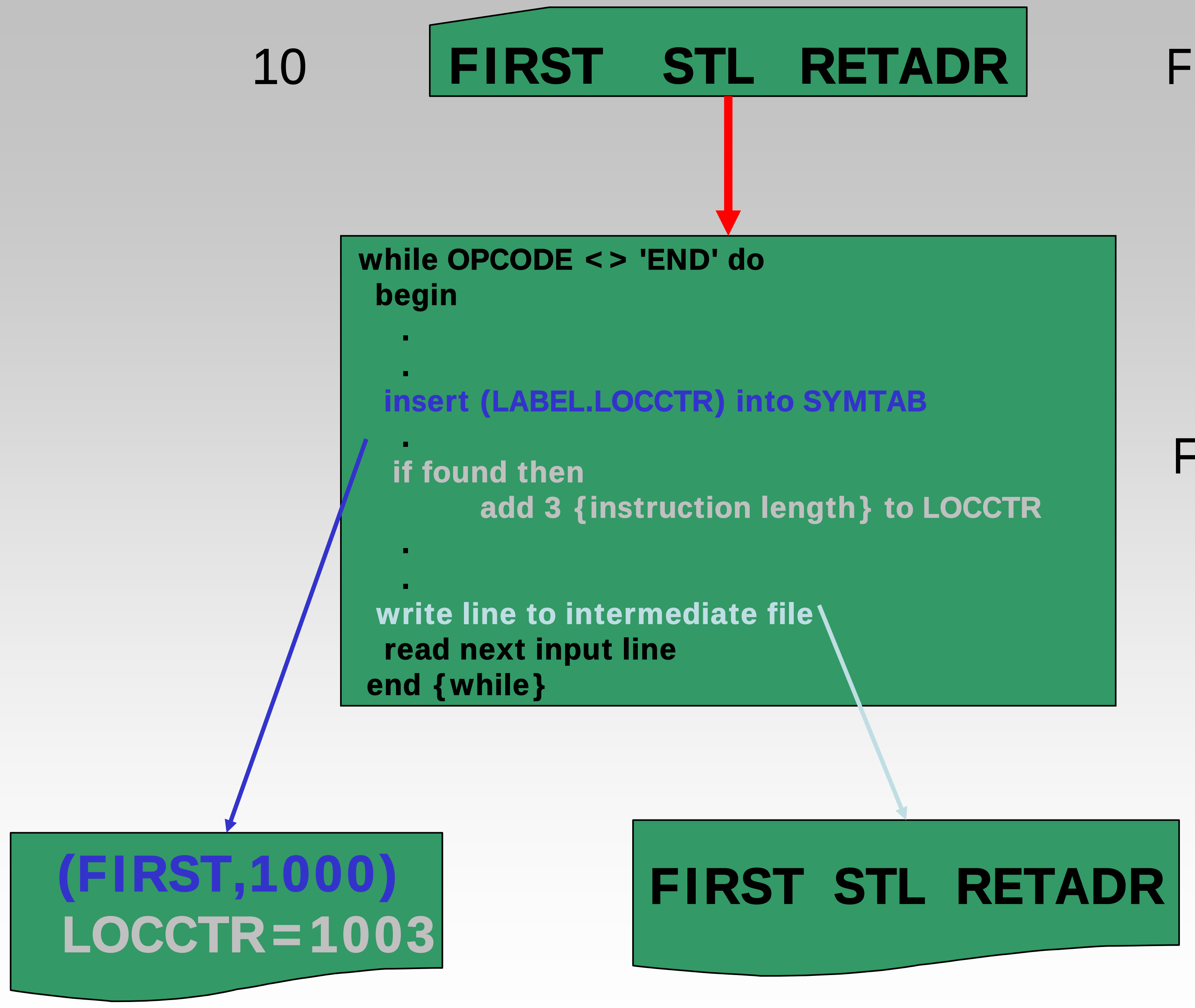


Fig.2.1

```
begin
read first input line
if OPCODE = 'START' then
    begin
      save #[OPERAND] as starting address
      initialize LOCCTR to starting address
      write line to intermediate file
      read next input line
    end  {if START}
else
    initialize LOCCTR to 0
```

Fig.2.4(a)

Intermediate file



10 FIRST STL RETADR

Fig.2.1

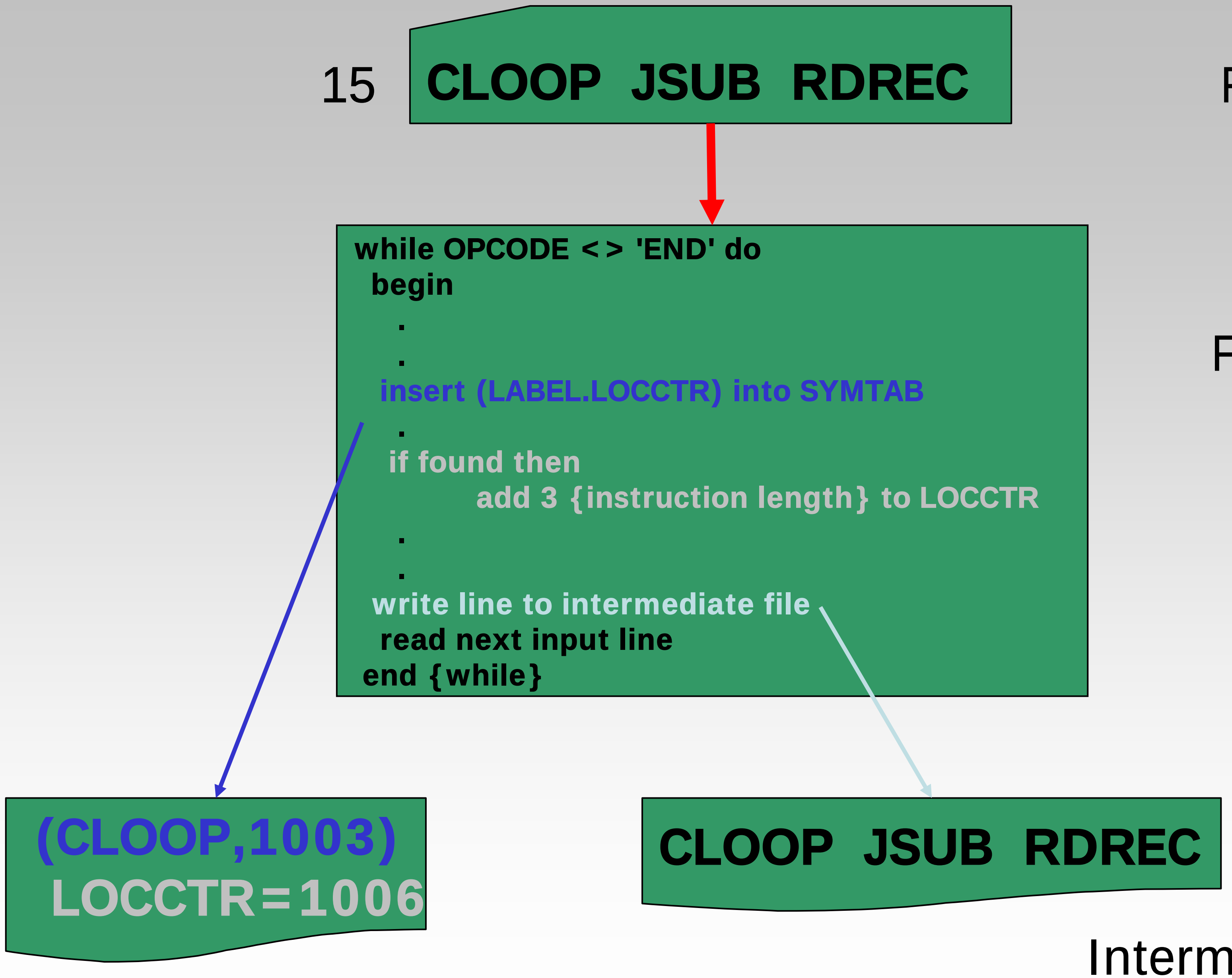
```
while OPCODE <> 'END' do
 begin
   .
   .
  insert (LABEL.LOCCTR) into SYMTAB
   .
   if found then
          add 3 {instruction length} to LOCCTR
   .
   .
  write line to intermediate file
  read next input line
 end {while}
```

Fig.2.4(a)

(FIRST,1000)
LOCCTR=1003

FIRST STL RETADR

Intermediate file

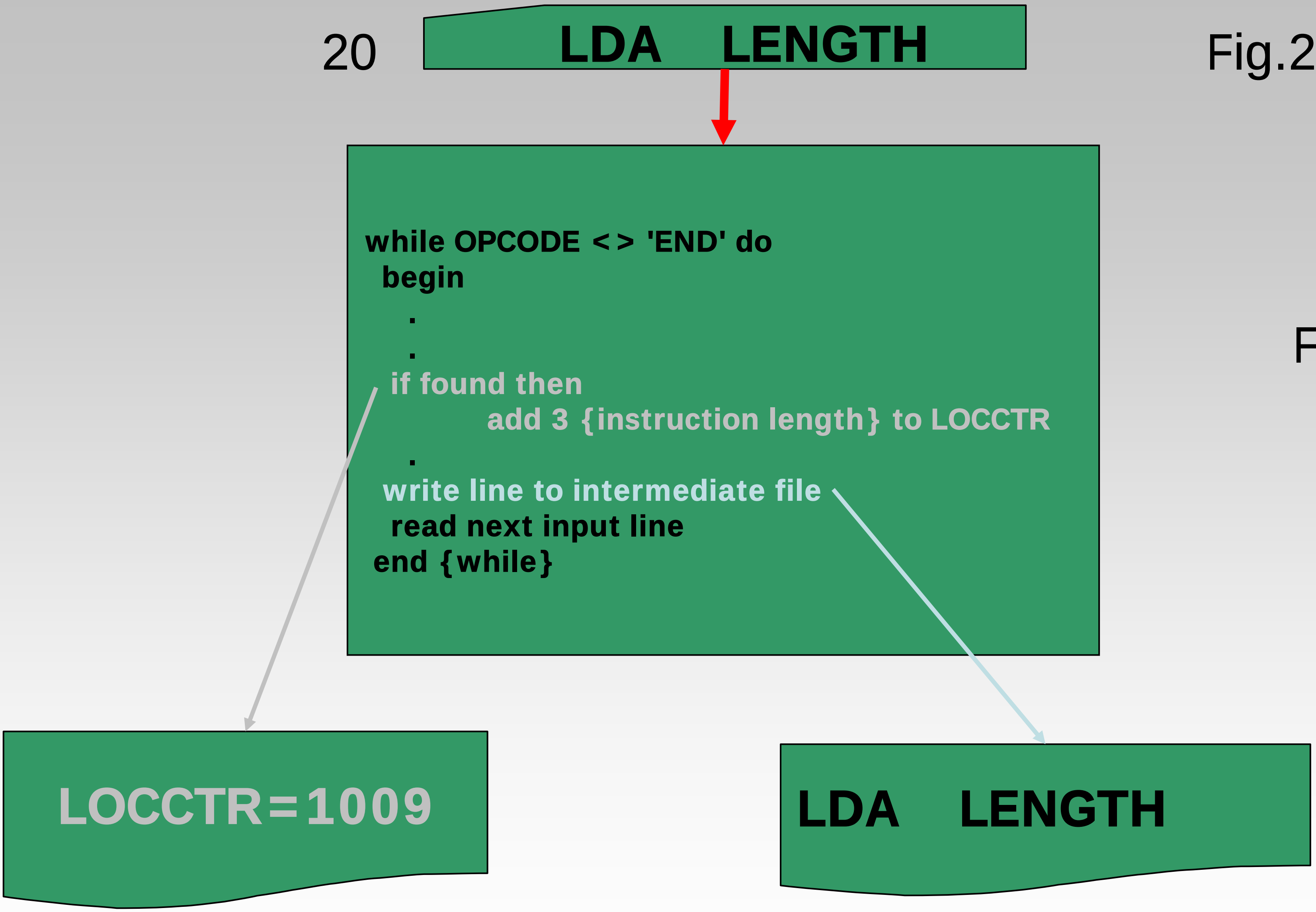


```
while OPCODE < > 'END' do
 begin
    .
    .
  insert (LABEL.LOCCTR) into SYMTAB
    .
   if found then
          add 3 {instruction length} to LOCCTR
    .
    .
  write line to intermediate file
  read next input line
 end {while}
```

Fig.2.1

Fig.2.4(a)

Intermediate file



```
while OPCODE <> 'END' do
  begin
     .
     .
   if found then
            add 3 {instruction length} to LOCCTR
     .
  write line to intermediate file
   read next input line
 end {while}
```

Fig.2.1

Fig.2.4(a)

Intermediate file

Fig.2.1

```
80   EOF     BYTE    C’EOF’
85   THREE   WORD    3
90   ZERO    WORD    0
```

Fig.2.4(a)

```
while OPCODE < > 'END' do
 begin
    .
    insert (LABEL.LOCCTR) into SYMTAB
    .
     else if OPCODE = 'WORD' then
        add 3 to LOCCTR
     else if OPCODE = 'BYTE' then
        begin
           find length of constant in bytes
           add length to LOCCTR
        end
    .
 write line to intermediate file
  read next input line
 end {while}
```

```
(EOF , 102A)
LOCCTR=102D
(THREE ,102D)
LOCCTR=1030
(ZERO , 1030)
LOCCTR=1033
```

```
EOF     BYTE    C’EOF’
THREE   WORD    3
ZERO    WORD    0
```

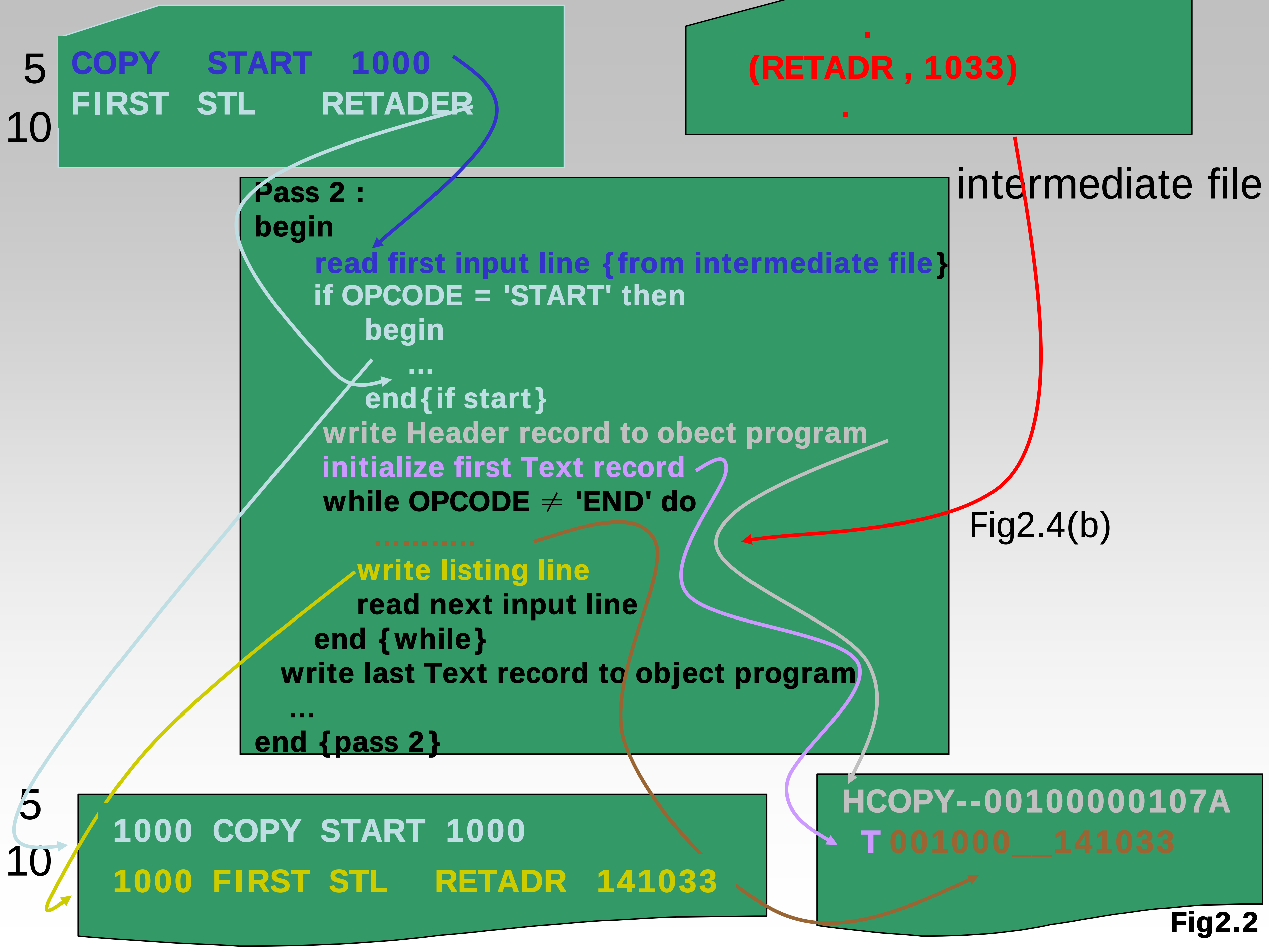
Intermediate file

Function of algorithm for pass_1 of assembler

(1)Assign address to all statements in the program

(2)Save the values (address) assigned to all labels

(3)Perform some processing of assembler directives



```
5   COPY    START   1000
10  FIRST   STL     RETADER
```

```
    .
(RETADR , 1033)
    .
```

intermediate file

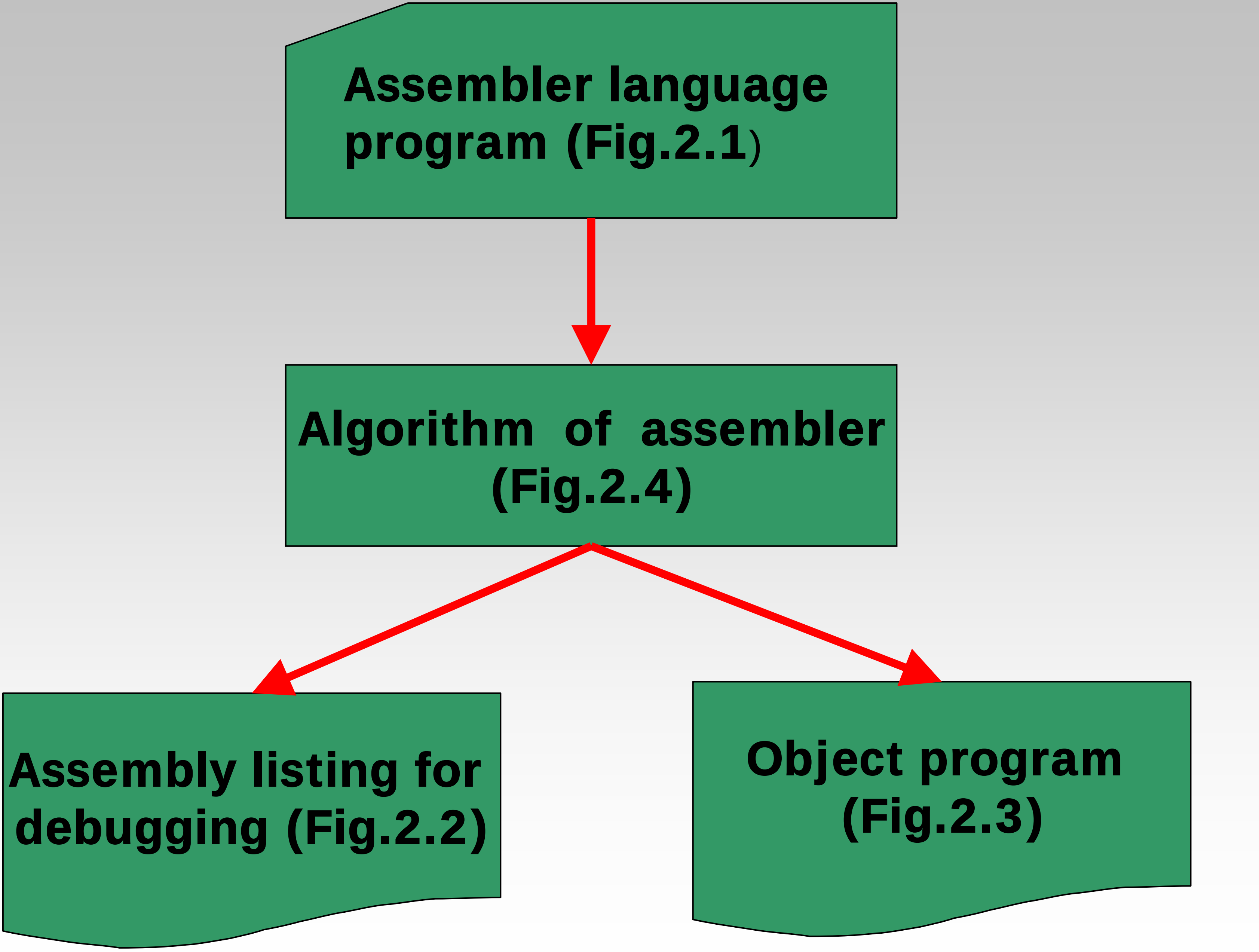
Fig2.4(b)

```
Pass 2 :
begin
    read first input line {from intermediate file}
    if OPCODE = 'START' then
        begin
            ...
        end{if start}
    write Header record to obect program
    initialize first Text record
    while OPCODE ≠ 'END' do
        ..........
        write listing line
        read next input line
    end {while}
   write last Text record to object program
   ...
end {pass 2}
```

```
5   1000  COPY  START  1000
10  1000  FIRST  STL    RETADR   141033
```

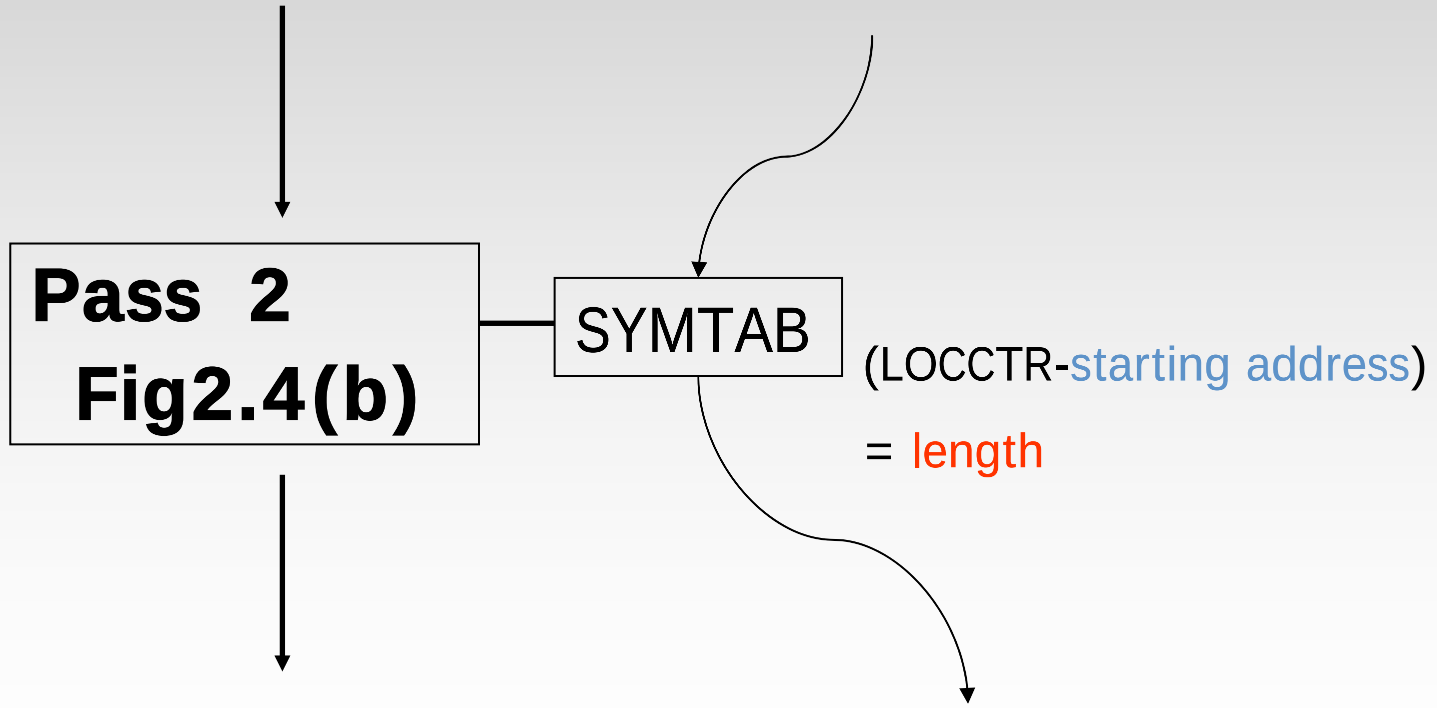
```
HCOPY--00100000107A
 T 001000__141033
```

Fig2.2


Assembler language program (Fig.2.1)
Algorithm of assembler (Fig.2.4)
Assembly listing for debugging (Fig.2.2)
Object program (Fig.2.3)

Object Program correspond to Fig 2.2 case1

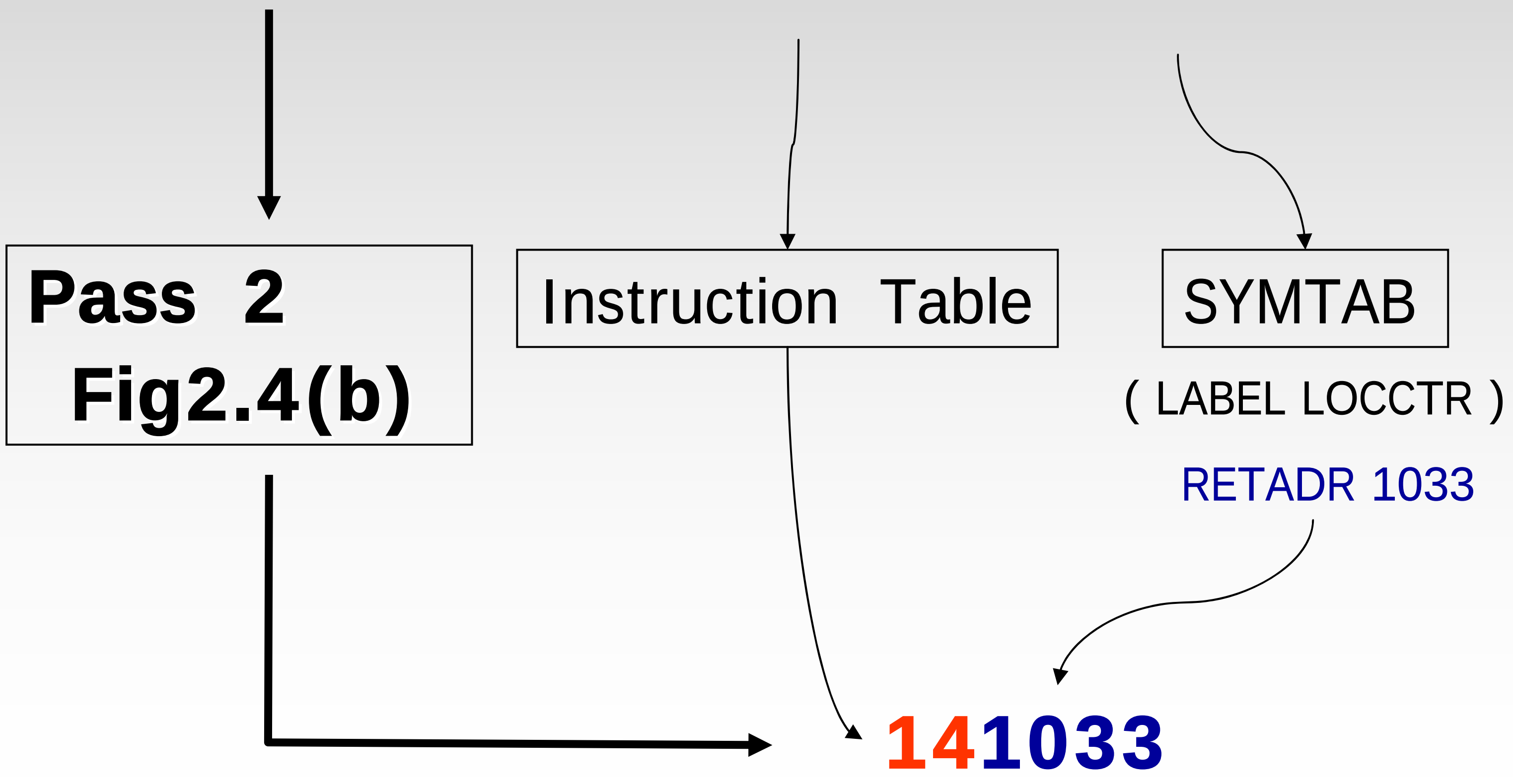
Line	Loc	Source statement		
5	1000	COPY	START	1000



H COPY 001000 00107A

Object Program correspond to Fig 2.2 case2

```
Line  Loc   Source statement
10    1000  FIRST     STL     RETADR
```



Object Program correspond to Fig 2.2 case3

Line	Loc	Source statement		
80	102A	EOF	BYTE	C'EOF'

character

Pass 2 Fig2.4(b)

```
else if OPCODE = 'BYTE' or 'WORD' then
                convert constant to object code
```

454F46

Object Program correspond to Fig 2.2 case4

Line	Loc	Source statement		
85	102D	THREE	WORD	3

```
Pass 2   Fig2.4(b)

else if OPCODE = 'BYTE' or 'WORD' then
                  convert constant to object code
```

$\longrightarrow$ 000003

Object Program correspond to Fig 2.2 case5

```
Line  Loc    Source statement
185 205D    INPUT     BYTE    X'F1'
```

hexadecimal

Pass 2 Fig2.4(b)

else if OPCODE = 'BYTE' or 'WORD' then
convert constant to object code

F1

Object Program correspond to Fig 2.2 case6

Line	Loc	Source statement		
190	205E	MAXLEN	WORD	4096

```
Pass  2   Fig2.4(b)

else if OPCODE = 'BYTE' or 'WORD' then
            convert constant to object code
```

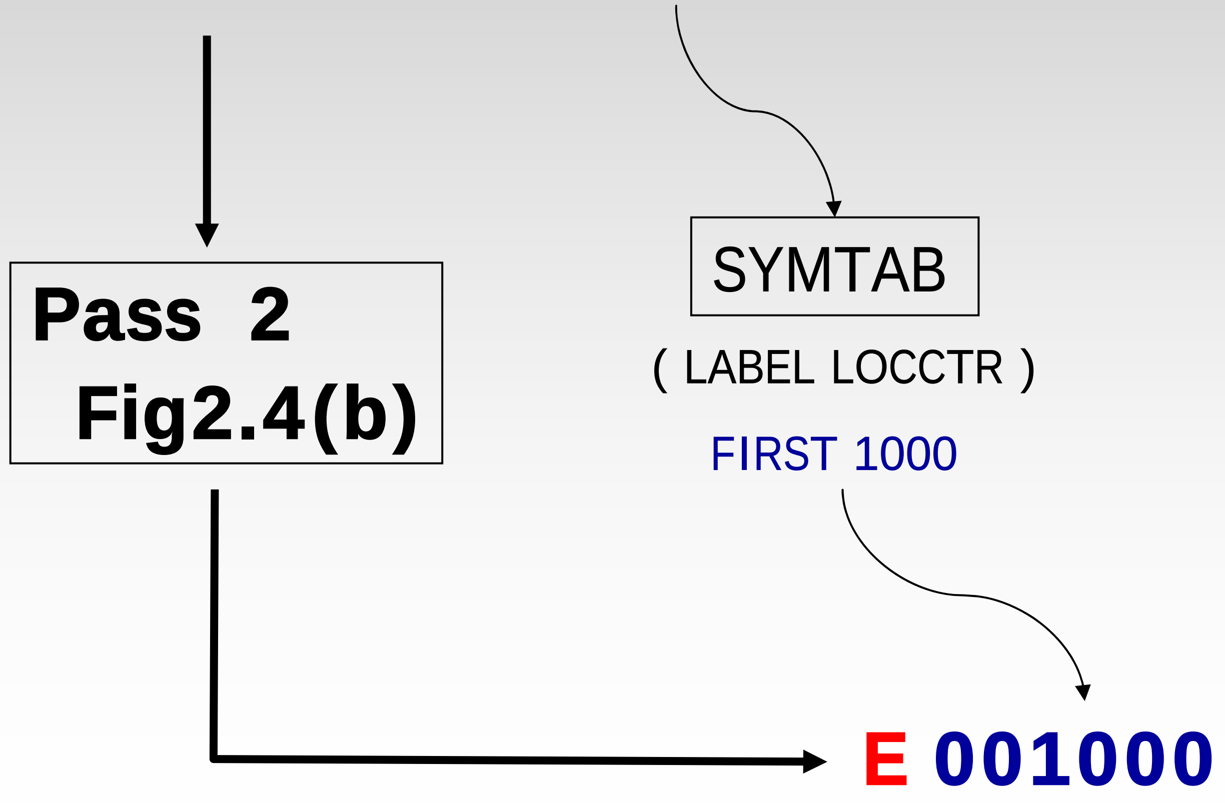
$4096 = 2^{12}$

12 8 4 1

001000

Object Program correspond to Fig 2.2 case7

Line	Loc	Source statement	
255		END	FIRST



Object Program correspond to Fig 2.2

```
HCOPY  00100000107A

T 001000 1E 141033 482039 001036 281030 301015 482061
    3C1003 00102A 0C1039 00102D

T 00101E 15 0C1036 482061 081033 4C0000 454F46 000003
    000000

T 002039 1E 041030 001030 E0205D 30203F D8205D 281030
    302057 549039 2C205E 38203F

T 002057 1C 101036 4C0000 F1  001000 041030 E02079 302064
    509039 DC2079 2C1036

T 002073 07 382064 4C0000 05

E 001000
```

Fig 2.3

2.2 Machine-dependent assembler

(1) Addressing mode symbols:

@ : indirect addressing mode

```
70              J       @RETADR

95   RETADR  RESW          1
```

# : immediate addressing mode

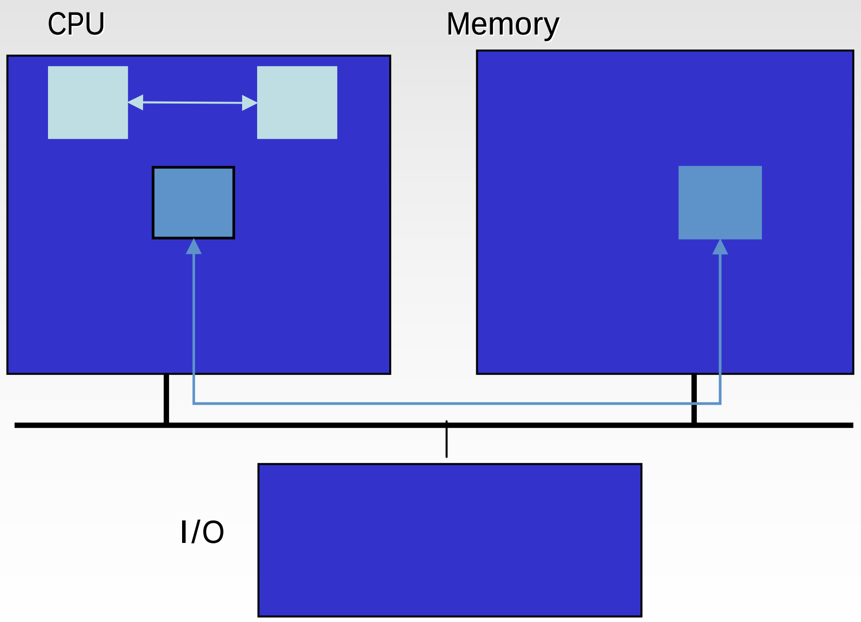
```
55              LDA     #3
```

+ : extended instruction format

```
15   CLOOP   +JSUB      RDREC
```

2.2 Machine-dependent assembler

(2) Use of register-register instructions instead of register memory instructions -> improve the exaction speed of the program.



2.2 Machine-dependent assembler

(3) If neither program-counter relative nor base relative addressing can be used, then the 4-byte extended Instruction format must be used.

```
    15    0006       CLOOP   +JSUB    RDREC

1036-0009                .
 =102D >1000
                         .
    125   1036       RDREC     CLEAR  X
```

2.2 Machine-dependent assembler

(4) Displacement calculation for program-counter relative and base addressing modes:

10 0000 FIRST STL RETADR

Since address (RETADR) =0030 and next address (FIRST) =0003, we obtain displacement=0030-0003=02D with pc relative addressing and neither indirect nor immediate addressing, the object code of this assembly instruction is
17202D

Opcode (STL) n i x b p e ..

000101 11 0 0 1 0 ..

1 7 2

2.2 Machine-dependent assembler

(5) The difference between pc relative addressing and base relative addressing is that the assembler knows what the contents of the program-counter will be at execution time but the base register is under the control of the programmer.

```
 20   000A            LDA    LENGTH
100   0033   LENGTH   RESW   1
175   1056   EXIT     STX    LENGTH
```

2.2 Machine-dependent assembler

(6) The displacement of pc relative mode is between -2048 and +2047 but the displacement of base relative mode is between 0 and 4095. For SIC/XE assembler, it attempt pc relative mode assembly first.

```
 20   000A           LDA    LENGTH
100   0033  LENGTH   RESW   1
175   1056  EXIT     STX    LENGTH
```

2.2 Machine-dependent assembler

(7) The kind of sharing of the common memory among programs is called multiprogramming. An object program that contains the information necessary to perform address modification is call a relocatable program.

Ex. 15 CLOOP +JSUB RDREC M 000007 05

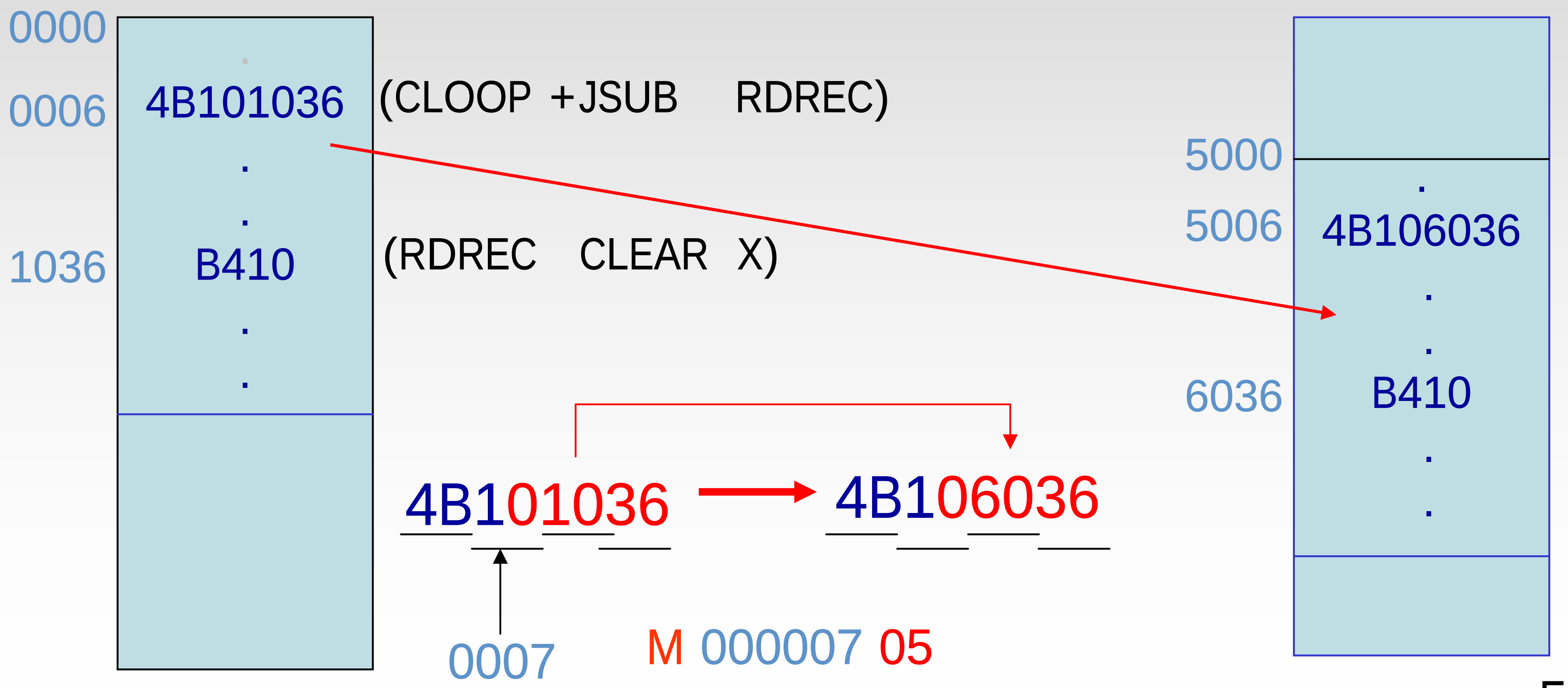


Fig 2.7

2.2 Machine-dependent assembler

(8) Modification record:

Col. 1 M

Col. 2-7 Starting location of the address field to be modified, relative to the beginning of the program.

Col. 8-9 Length of the address field to be modified in half-bytes.

```
15 CLOOP   +JSUB    RDREC
```

M 000007 05

(5*4=20 bits)

2.2 Machine-dependent assembler

(9) The instructions need not be modified:

* the instruction operand is not a memory address.

```
25          COMP   #0
```

* the operand is specified using pc relative or base relative addressing.

```
40    J      CLOOP
160   STCH   BUFFER,X
```

2.2 Machine-dependent assembler

(10) The only parts of the program that require modification at load time are those that specify direct address.

```
15  CLOOP   +JSUB   RDREC   M 000007 05
35          +JSUB   WRREC   M 000014 05
65          +JSUB   WRREC   M 000027 05
```

2.3 Machine-independent assembler features

2.3 Machine-independent assembler features

(1)Immediate addressing : the operand is assembled as part of the machine instruction.
Literal addressing : the operand value is specified as a constant at some other memory location.

2.3 Machine-independent assembler features

(2)LITTAB (literal table):

Pass 1: literal->LITTAB->LTORG->address

Pass 2: literal->LITTAB->address

2.3 Machine-independent assembler features

(3)Why use EQU?

*It is used for improved readability in place of numeric values.

*It is used for defining mnemonic names for registers.

*It is used to have the standard register mnemonic built into the assembler.

2.3 Machine-independent assembler features

(4)Why use ORG?

*It assigns values to symbols.

*It is used in label definition.

*Restriction: it must have been defined previously in the program.

2.3 Machine-independent assembler features

(5)Expressions are classified as either absolute expressions or relative expressions depending upon the type of value they produce.

*Absolute expressions: relative terms occur in pairs.

*Relative expressions: the remaining unpaired relative term must have a positive sign.

*Example:

RETADR(R),BUFFER(R),BUFEND(R),MAXLEN(A).

2.3 Machine-independent assembler features

(6)Program locks allow the generated machine instructions and data to appear in the object program in a different order from the corresponding source statements.

2.3 Machine-independent assembler features

(7)The assembler directive USE indicates which portions of the source program belong to the various blocks.

2.3 Machine-independent assembler features

(8)During pass 1, a separate location counter for each program block and each label in the program is assigned an address that in relative to the start of the block that contains it.

Block name	Block number	Address	Length
(default)	0	0000	0066
CDATA	1	0066	000B
CBLKS	2	0071	1000

Example:

```
20 0006 0   LDA LENGTH 032   ???
```

operand (LENGTH)=0003

start address of program block 1 (CDATA)=0066

->Target address=0003+0066=0069

->Since pc relative addressing, the required displacement=0069-0009=0060->???=060

2.3 Machine-independent assembler features

(9)The separation of the program into blocks has considerably reduced the addressing problems.

HCOPY…

T000000…

T00001E…

T000027…

T000044…

T00006C…

T00004D…

T00006D…

T000000…

2.3.5 Control sections and program linking

2.3.5 Control sections and program linking

(1)A control section is a part of the program that maintains its identity after assembly. When control section from logically related parts of a program, it is necessary to provide some means for linking them together. A major benefit of using control sections is the resulting flexibility.

2.3.5 Control sections and program linking

(2)The EXTDEF statement in a control section names symbols called external symbols, that are defined in this control sections and may be used by other sections.

2.3.5 Control sections and program linking

(3)The EXTREF statement names symbols that are used in this control sections and defined elsewhere.

2.3.5 Control sections and program linking

(4)Example:

(Fig 2.16)

```
15  0003  CLOOP   +JSUB RDREC 4B100000
```

2.3.5 Control sections and program linking

(5)Note the different between the handing of the expression on line 190 and the similar expression on line 107.

(Fig 2.16)

```
107 1000 MAXLEN  EQU     BUFEND-BUFFER

109 1000 MAXLEN  WORD  BUFEND-BUFFER
```

2.3.5 Control sections and program linking

(6)The assembler must include information in the object program that will cause the loader to insert the proper values where they are required. The required types of object code format to handle external defined or external referenced symbols are Define, Refer and revised Modification.

2.3.5 Control sections and program linking

(7)Example:

M00000405+RDREC

2.4
Assembler design options

2.4 Assembler design options 1/8

(1) Two pass assembler with overlay structure is designed to execute some of its segments overlaying others.

2.4 Assembler design options

(2)To reduce the size of the problem, many one-pass assemblers do prohibit forward references to data items.

2.4 Assembler design options

(3)There are two main types of one-pass assembler. One type produces object code directly in memory for immediate execution; the other type produces the usual kind of object program for later execution.

2.4 Assembler design options

(4) Load-and-go assembler: It scans source program → if operand is not defined, the operand address is omitted until the definition is encountered → if the value of some operand in SYMTAB is still marked with * after the completion of scanning source code, it indicate undefined symbol errors.

2.4 Assembler design options 5/8

(5) One-pass assemblers that produce object programs as output: The assembler generates another Text record with the correct operand address. When the program is loaded, this address will be inserted into the instruction by the action of the loader.

2.4 Assembler design options 6/8

(6) Multi-pass assembler can made as many passes as are needed to process the definitions of symbols.

2.4 Assembler design options 7/8

(7)The undefined symbol is stored in the SYMTAB in the defining expression is undefined while the expression might be pointed by the SYMTAB.
Symbol * identicates undefined operand. Associated with the entry of SYMTAB is a list of the symbols whose values depend on the symbols of this entry.

2.4 Assembler design options

(8) Operation of multi-pass assembler:

Defined symbol

→ SYMTAB (&n-1) or *

→ expression

→ recursive operation

→ in any symbols remained undefined

→ errors.