SWE4001 – System Programming
Module 3: Assembler
Lesson 8 of 9: Assembler Design Options

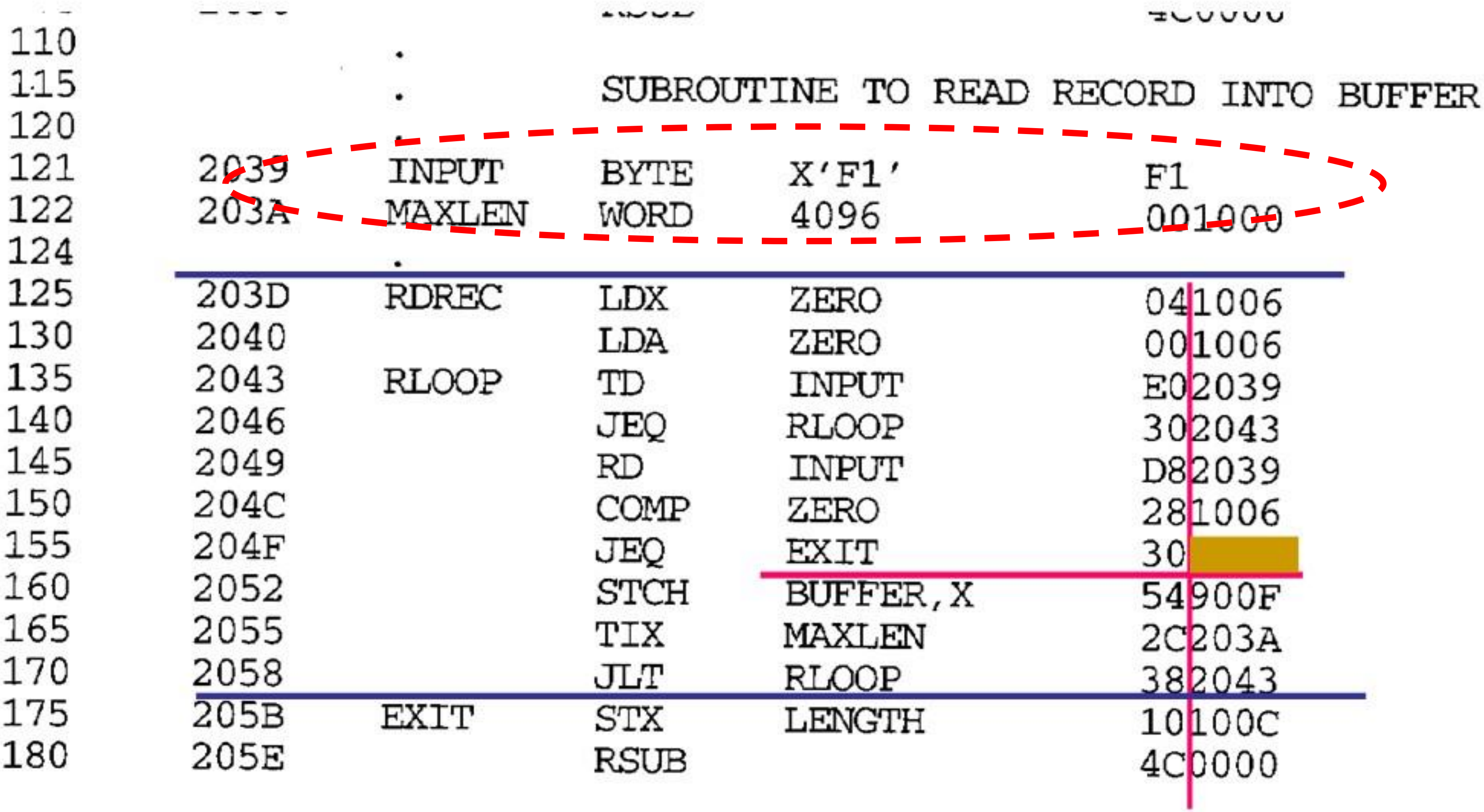
2.4 Assembler Design Options
2.4.1 Two-Pass Assembler

- Most assemblers
  - Processing the source program into two passes.
  - The internal tables and subroutines that are used only during Pass 1.
  - The SYMTAB, LITTAB, and OPTAB are used by both passes.
- The main problems to assemble a program in one pass involves forward references.

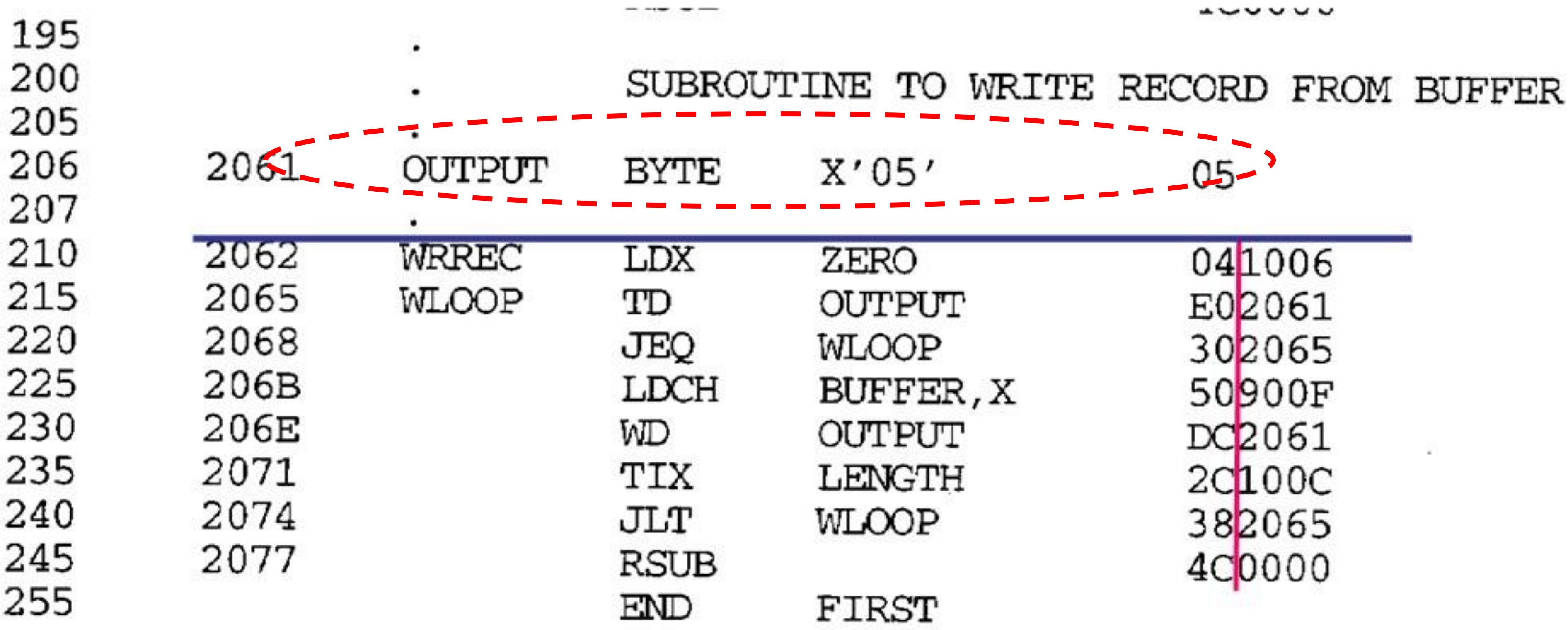
2.4.2 One-Pass Assemblers

- Eliminate forward references
  - Data items are defined before they are referenced.
  - But, forward references to labels on instructions cannot be eliminated as easily.
  - Prohibit forward references to labels.

Line	Loc	Source statement			Object code
0	1000	COPY	START	1000	
1	1000	EOF	BYTE	C'EOF'	454F46
2	1003	THREE	WORD	3	000003
3	1006	ZERO	WORD	0	000000
4	1009	RETADR	RESW	1	
5	100C	LENGTH	RESW	1	
6	100F	BUFFER	RESB	4096	
9		.			
10	200F	FIRST	STL	RETADR	141009
15	2012	CLOOP	JSUB	RDREC	48
20	2015		LDA	LENGTH	00100C
25	2018		COMP	ZERO	281006
30	201B		JEQ	ENDFIL	30
35	201E		JSUB	WRREC	48
40	2021		J	CLOOP	302012
45	2024	ENDFIL	LDA	EOF	001000
50	2027		STA	BUFFER	0C100F
55	202A		LDA	THREE	001003
60	202D		STA	LENGTH	0C100C
65	2030		JSUB	WRREC	48
70	2033		LDL	RETADR	081009
75	2036		RSUB		4C0000



```
105     2036               RSUB                        4C0000
110                 .
115                 .      SUBROUTINE TO READ RECORD INTO BUFFER
120                 .
121     2039    INPUT      BYTE      X'F1'             F1
122     203A    MAXLEN     WORD      4096              001000
124                 .
125     203D    RDREC      LDX       ZERO              041006
130     2040               LDA       ZERO              001006
135     2043    RLOOP      TD        INPUT             E02039
140     2046               JEQ       RLOOP             302043
145     2049               RD        INPUT             D82039
150     204C               COMP      ZERO              281006
155     204F               JEQ       EXIT              30
160     2052               STCH      BUFFER,X          54900F
165     2055               TIX       MAXLEN            2C203A
170     2058               JLT       RLOOP             382043
175     205B    EXIT       STX       LENGTH            10100C
180     205E               RSUB                        4C0000
```



```
         .
         .       SUBROUTINE TO WRITE RECORD FROM BUFFER
         .
2061     OUTPUT  BYTE    X’05’               05
         .
2062     WRREC   LDX     ZERO                041006
2065     WLOOP   TD      OUTPUT              E02061
2068             JEQ     WLOOP               302065
206B             LDCH    BUFFER,X            50900F
206E             WD      OUTPUT              DC2061
2071             TIX     LENGTH              2C100C
2074             JLT     WLOOP               382065
2077             RSUB                        4C0000
                 END     FIRST
```

Figure 2.18 Sample program for a one-pass assembler.

All variables are defined before they are used.

Two Types

- There are two types of one-pass assembler:
- **Produce object code directly in memory for immediate execution**
  - No loader is needed
  - Load-and-go for program development and testing
  - Good for computing center where most students reassemble their programs each time.
  - Avoids the overhead of writing the object program out and reading it back .
  - For a load-and-go assembler, the actual address must be known at assembly time, we can use an absolute program
- **Produce the usual kind of object program for later execution**

Internal Implementation

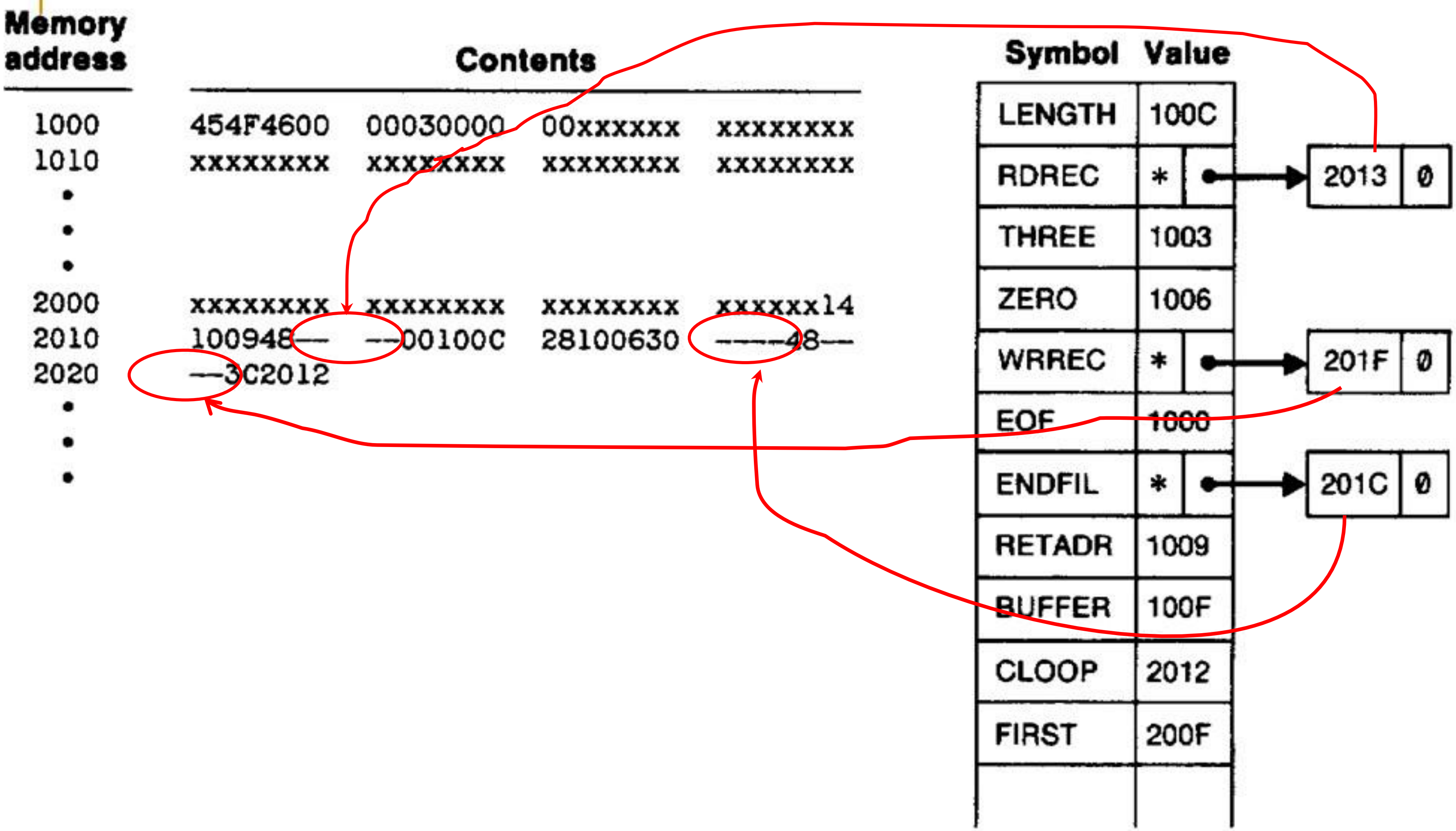
- The assembler generate object code instructions as it scans the source program.
- If an instruction operand is a symbol that has not yet been defined
  - The operand address is omitted when the instruction is assembled.
  - The symbol used as an operand is entered into the symbol table.
  - This entry is flagged to indicate that the symbol is undefined yet.

Internal Implementation

- The address of the operand field of the instruction that refers to the undefined symbol
  - Added to a list of forward references associated with the symbol table entry.
- When the definition of the symbol is encountered,
  - The forward reference list for that symbol is scanned,
  - The proper address is inserted into any instruction previously generated.

2.4.2 One-Pass Assemblers

- Load-and-go one-pass assembler
  - The assembler avoids the overhead of writing the object program out and reading it back in.
  - The object program is produced in memory, the handling of forward references becomes less difficult.
  - Figure 2.19(a), shows the SYMTAB after scanning line 40 of the program in Figure 2.18.
  - Since RDREC was not yet defined, the instruction was assembled with no value assigned as the operand address (denote by - - - -).

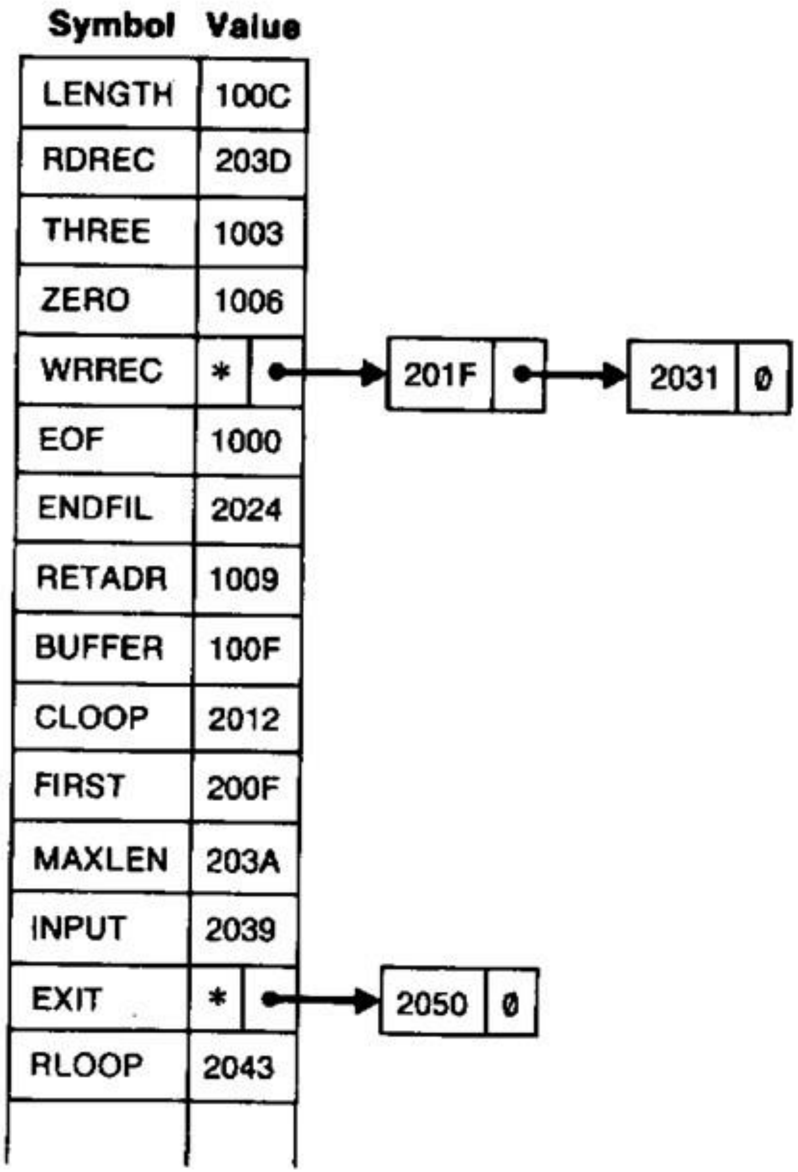


```
Memory
address                          Contents
1000      454F4600   00030000   00xxxxxx   xxxxxxxx
1010      xxxxxxxx   xxxxxxxx   xxxxxxxx   xxxxxxxx
  •
  •
  •
2000      xxxxxxxx   xxxxxxxx   xxxxxxxx   xxxxxx14
2010      100948--   --00100C   28100630   ----48--
2020      --3C2012
  •
  •
  •
```

Symbol	Value	
LENGTH	100C	
RDREC	*	→ 2013 | 0
THREE	1003	
ZERO	1006	
WRREC	*	→ 201F | 0
EOF	1000	
ENDFIL	*	→ 201C | 0
RETADR	1009	
BUFFER	100F	
CLOOP	2012	
FIRST	200F	

Figure 2.19(a) Object code in memory and symbol table entries for the program in Fig. 2.18 after scanning line 40.

```
Memory
address                          Contents
1000       454F4600  00030000  00xxxxxx  xxxxxxxx
1010       xxxxxxxx  xxxxxxxx  xxxxxxxx  xxxxxxxx
  •
  •
  •
2000       xxxxxxxx  xxxxxxxx  xxxxxxxx  xxxxxx14
2010       10094820  3D00100C  28100630  202448--
2020       --3C2012  0010000C  100F0010  030C100C
2030       48----08  10094C00  00F10010  00041006
2040       001006E0  20393020  43D82039  28100630
2050       ----5490  0F
  •
  •
  •
```



Symbol	Value
LENGTH	100C
RDREC	203D
THREE	1003
ZERO	1006
WRREC	* → 201F → 2031 ∅
EOF	1000
ENDFIL	2024
RETADR	1009
BUFFER	100F
CLOOP	2012
FIRST	200F
MAXLEN	203A
INPUT	2039
EXIT	* → 2050 ∅
RLOOP	2043

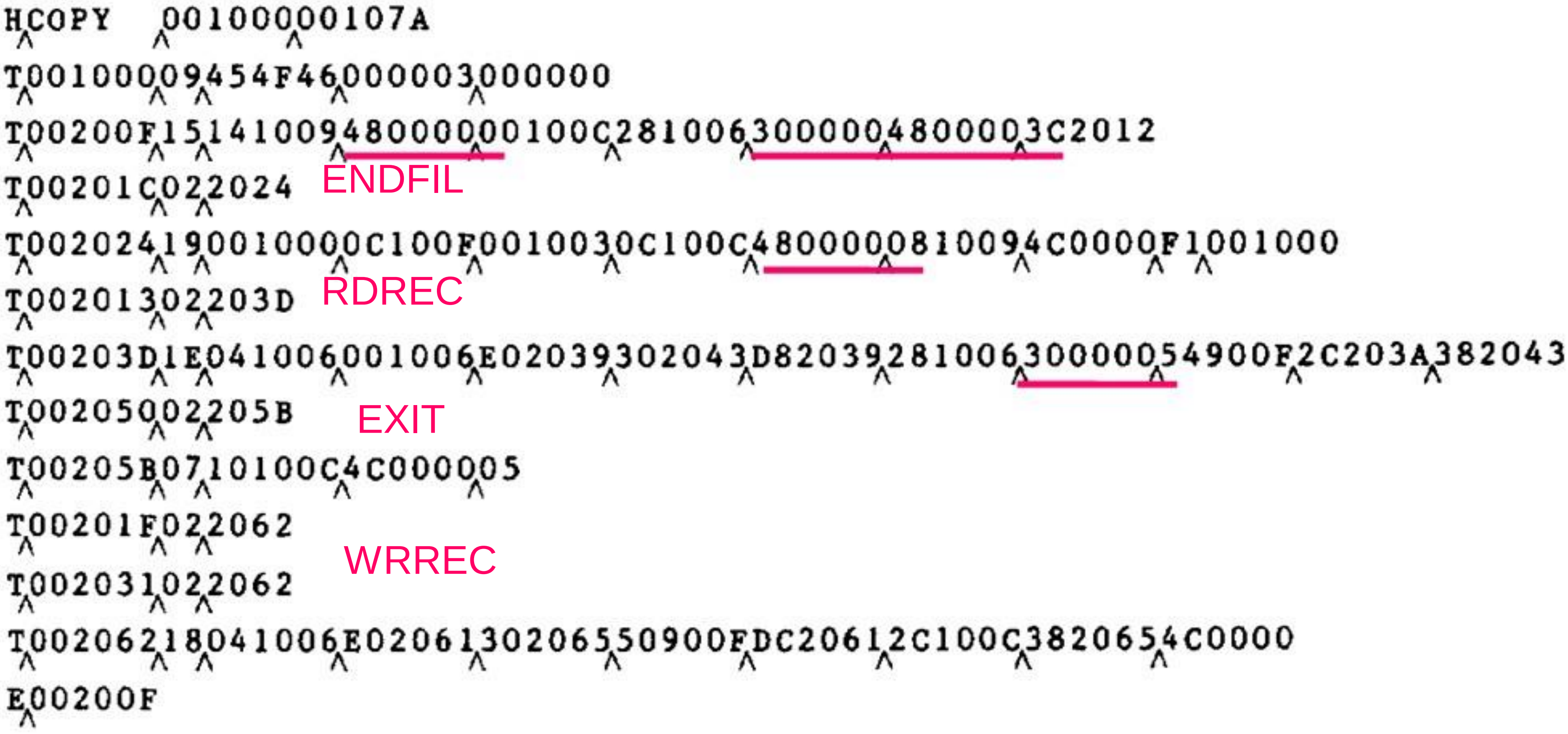
Figure 2.19(b) Object code in memory and symbol table entries for the program in Fig. 2.18 after scanning line 160.

2.4.2 One-Pass Assemblers

- Load-and-go one-pass assembler
  - RDREC was then entered into SYMTAB as an undefined symbol, the address of the operand field of the instruction (2013) was inserted.
  - Figure 2.19(b), when the symbol ENDFIL was defined (line 45), the assembler placed its value in the SYMTAB entry; it then inserted this value into the instruction operand field (201C).
  - At the end of the program, all symbols must be defined without any * in SYMTAB.
  - For a load-and-go assembler, the actual address must be known at assembly time.

2.4.2 One-Pass Assemblers

- Another one-pass assembler by generating OP
  - Generate another Text record with correct operand address.
  - When the program is loaded, this address will be inserted into the instruction by the action of the loader.
  - Figure 2.20, the operand addresses for the instructions on lines 15, 30, and 35 have been generated as 0000.
  - When the definition of ENDFIL is encountered on line 45, the third Text record is generated, the value 2024 is to be loaded at location 201C.
  - The loader completes forward references.



```
H^COPY  ^001000^00107A
T^001000^09^454F46^000003^000000
T^00200F^15^141009^480000^00100C^281006^300000^480000^3C2012
T^00201C^02^2024   ENDFIL
T^002024^19^001000^0C100F^001003^0C100C^480000^081009^4C0000^F1^001000
T^002013^02^203D   RDREC
T^00203D^1E^041006^001006^E02039^302043^D82039^281006^300000^54900F^2C203A^382043
T^002050^02^205B    EXIT
T^00205B^07^10100C^4C0000^05
T^00201F^02^2062
                    WRREC
T^002031^02^2062
T^002062^18^041006^E02061^302065^50900F^DC2061^2C100C^382065^4C0000
E^00200F
```

Figure 2.20 Object program from one-pass assembler for program in Fig. 2.18.

2.4.2 One-Pass Assemblers

- In this section, simple one-pass assemblers handled absolute programs (SIC example).

Multi-Pass Assemblers

- Restriction on EQU and ORG
  - No forward reference, since symbols’ value can’t be defined during the first pass
- Example:

```
ALPHA   EQU   BETA
BETA EQU   DELTA
DELTA   RESW 1
```

  - Assemblers with 2 passes cannot resolve

Multi-Pass Assembler

- The assembler directives that define symbol requires the R.H.S be defined previously in the source program
- If we use a two-pass assembler, the following symbol definition cannot be allowed.

```
ALPHA      EQU    BETA
BETA       EQU    DELTA
DELTA      RESW   1
```

- This is because ALPHA and BETA cannot be defined in pass 1.
  - we allow multi-pass processing,
  - DELTA is defined in pass 1,
  - BETA is defined in pass 2,
  - ALPHA is defined in pass 3, and the above definitions can
  - be allowed.
- This is the motivation for using a multi-pass assembler.

Multi-Pass Assemblers

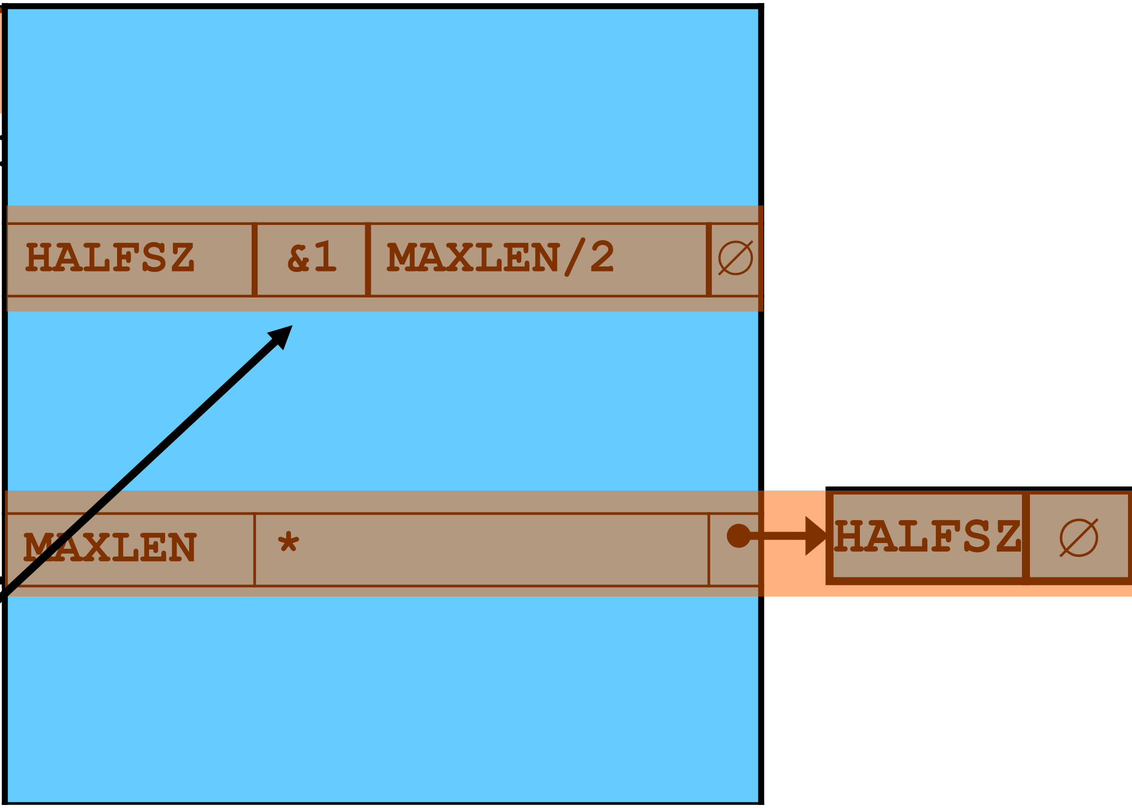
- Resolve forward references with as many passes as needed
  - Portions that involve forward references in symbol definition are saved during Pass 1
  - Additional passes through stored definitions
  - Finally a normal Pass 2
- Example implementation:
  - Use link lists to keep track of whose value depend on an undefined symbol

Multi-Pass Assembler Implementation

- Use a symbol table to store symbols that are not totally defined yet.
- For a undefined symbol, in its entry,
  - We store the names and the number of undefined symbols which contribute to the calculation of its value.
  - We also keep a list of symbols whose values depend on the defined value of this symbol.
- When a symbol becomes defined, we use its value to reevaluate the values of all of the symbols that are kept in this list.
- The above step is performed recursively.

Figure 2.21(a): After Pass 1

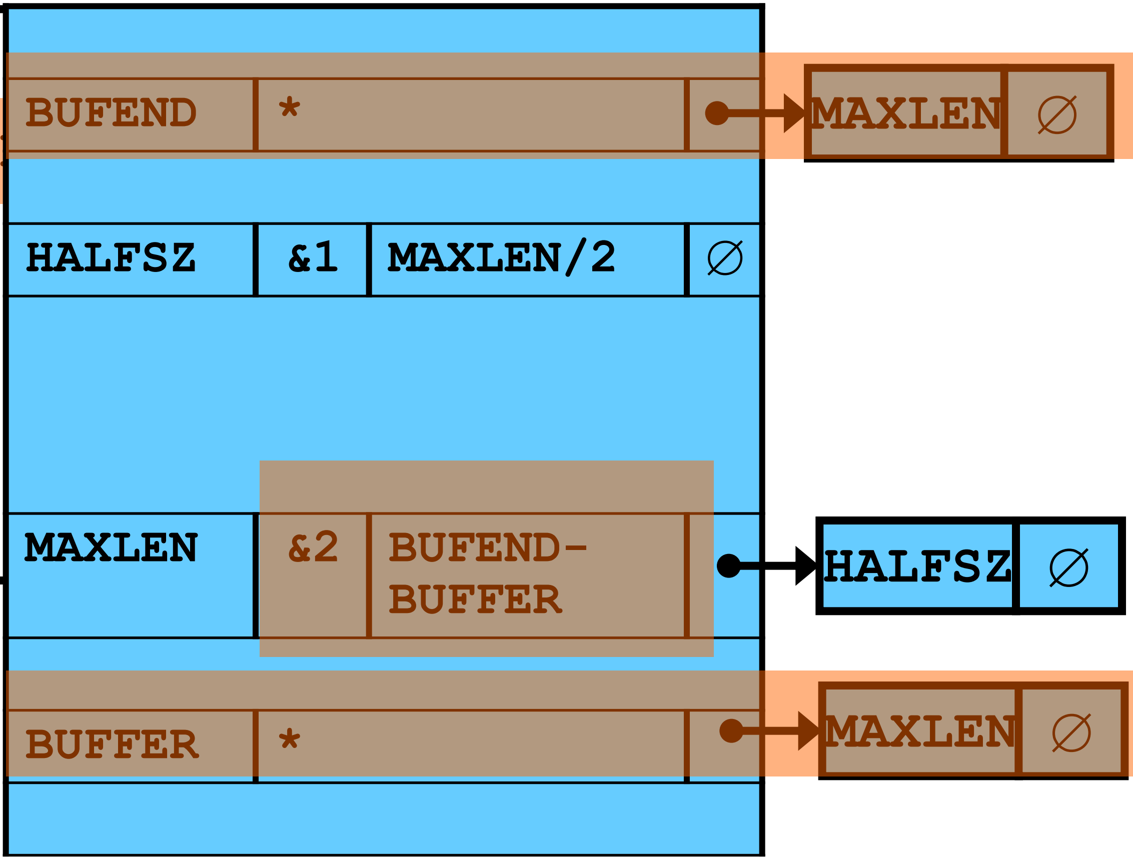
```
1 HALFSZ EQU   MAXLEN/2
2 MAXLEN EQU   BUFEND-BU
3 PREVBT EQU   BUFFER-1
    ...
4 BUFFER RESB  4096
5 BUFEND EQU   *
```



1 symbol undefined

Figure 2.21(c): MAXLEN Defined

```
1 HALFSZ  EQU   MAXLEN/2
2 MAXLEN  EQU   BUFEND-BU
3 PREVBT  EQU   BUFFER-1
    ...
4 BUFFER RESB 4096
5 BUFEND  EQU   *
```



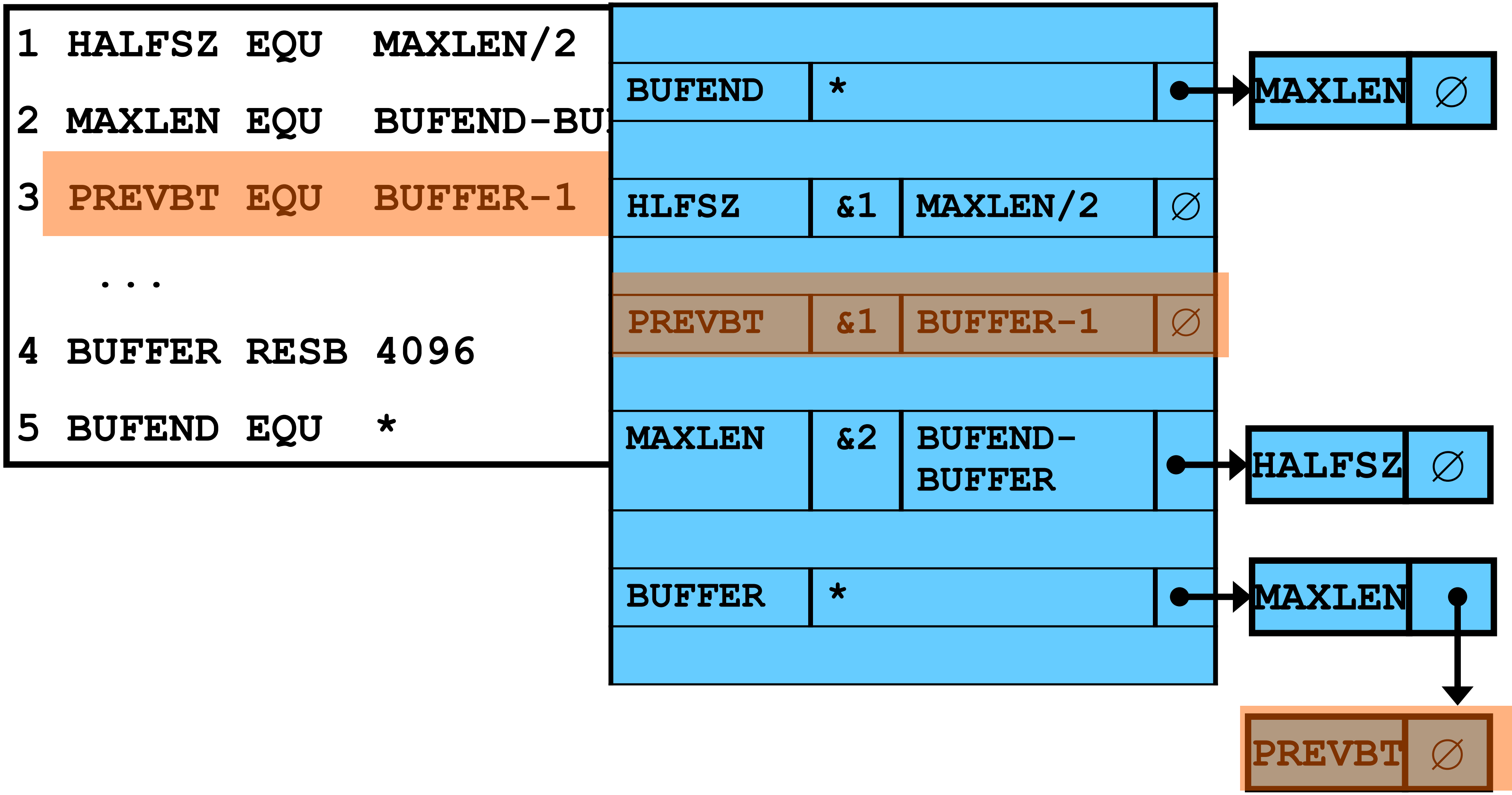
BUFEND * → MAXLEN ∅

HALFSZ &1 MAXLEN/2 ∅

MAXLEN &2 BUFEND-BUFFER → HALFSZ ∅

BUFFER * → MAXLEN ∅

Figure 2.21(d): PREVBT Defined

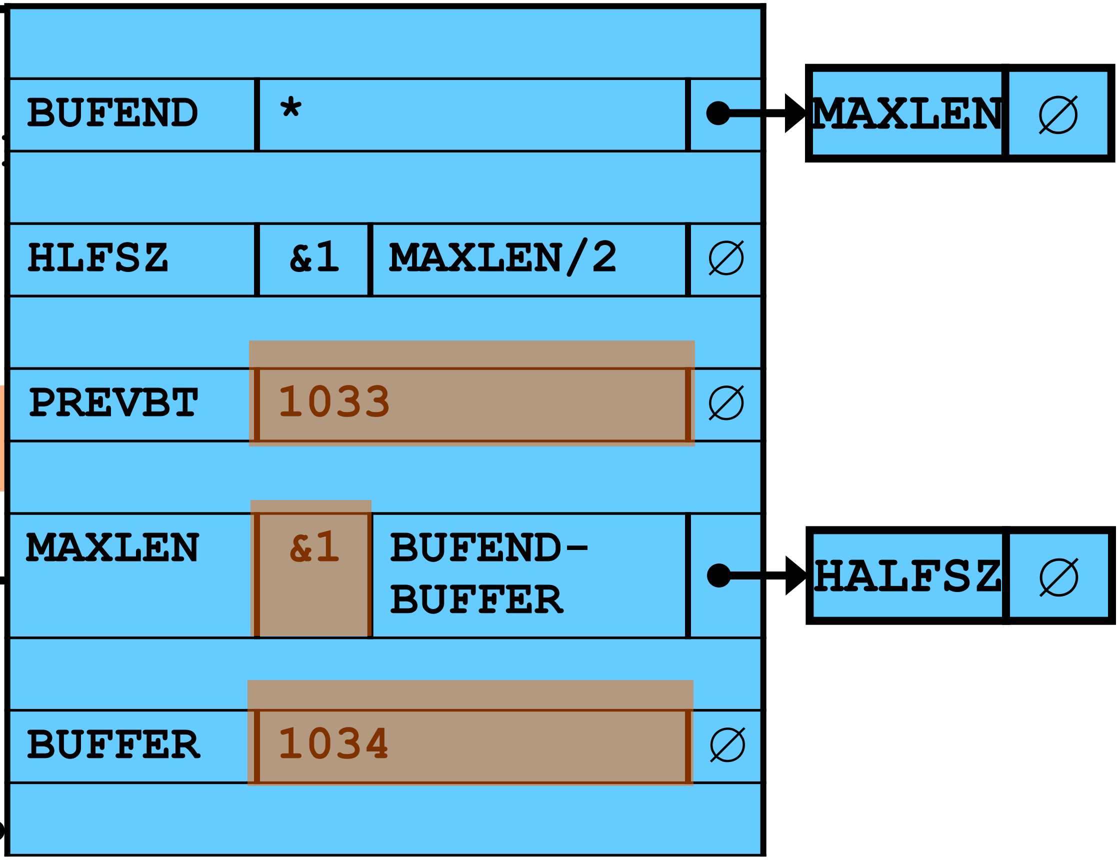


```
1  HALFSZ  EQU   MAXLEN/2
2  MAXLEN  EQU   BUFEND-BU
3  PREVBT  EQU   BUFFER-1
     ...
4  BUFFER RESB 4096
5  BUFEND  EQU   *
```

BUFEND	*		● → MAXLEN ∅
HLFSZ	&1	MAXLEN/2	∅
PREVBT	&1	BUFFER-1	∅
MAXLEN	&2	BUFEND-BUFFER	● → HALFSZ ∅
BUFFER	*		● → MAXLEN ● → PREVBT ∅

Figure 2.21(e): After Line 4

```
1 HALFSZ  EQU   MAXLEN/2
2 MAXLEN  EQU   BUFEND-BU
3 PREVBT  EQU   BUFFER-1
     ...
4 BUFFER RESB  4096
5 BUFEND  EQU   *
```



BUFEND		*	●
HLFSZ	&1	MAXLEN/2	∅
PREVBT		1033	∅
MAXLEN	&1	BUFEND- BUFFER	●
BUFFER		1034	∅

Assume
loc=1034

Figure 2.21(f): After Line 5

```
1 HALFSZ EQU   MAXLEN/2

2 MAXLEN EQU   BUFEND-BUFFER

3 PREVBT EQU   BUFFER-1

    . . .

4 BUFFER RESB 4096

5 BUFEND EQU   *
```

BUFEND	2034	∅
HLFSZ	800	∅
PREVBT	1033	∅
MAXLEN	1000	∅
BUFFER	1034	∅