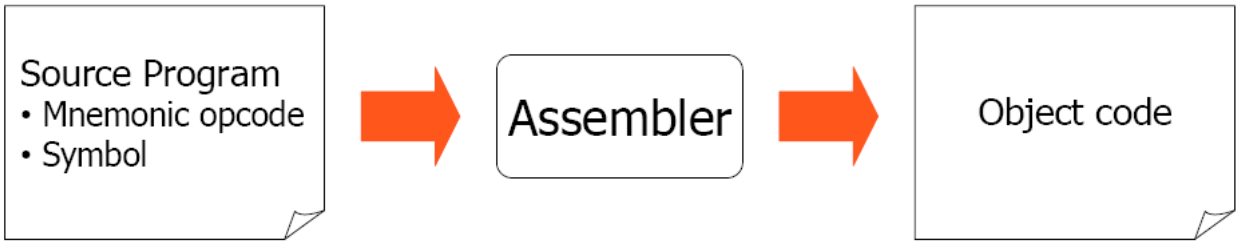
Assemblers

Dr. Monther Aldwairi

Fundamental Functions

- Generate machine language
  - Translate mnemonic operation codes to machine code
- Assign addresses to symbolic labels used by the programmer



Additional Functions

- Generate an image of what memory must look like for the program to be executed.
- Interpret assembler directives (Pseudo-Instructions)
  - They provide instructions to the assembler
  - They do not translate into machine code
  - They might affect the object code

Input / Output

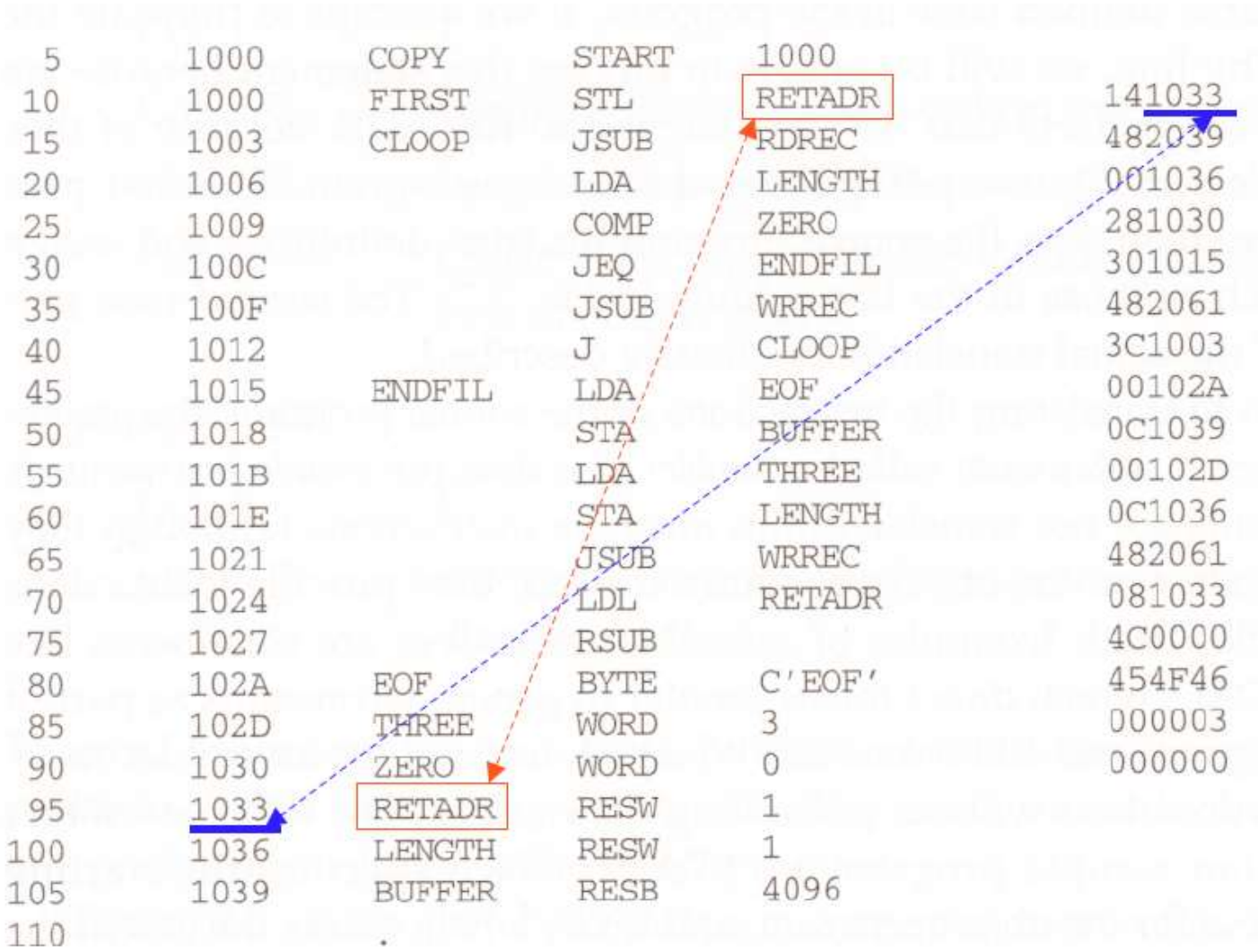
- Input Assembly Code
- Output
  - Assembly Listing
  - Object Code
- Intermediate files
  - Assembly Listing
  - LOCCTR
  - Instruction Length
  - Error Flags

Input

```
 5     COPY      START   1000        COPY FILE FROM INPUT TO OUTPUT
10     FIRST     STL     RETADR      SAVE RETURN ADDRESS
15     CLOOP     JSUB    RDREC       READ INPUT RECORD
20               LDA     LENGTH      TEST FOR EOF (LENGTH = 0)
25               COMP    ZERO
30               JEQ     ENDFIL      EXIT IF EOF FOUND
35               JSUB    WRREC       WRITE OUTPUT RECORD
40               J       CLOOP       LOOP
45     ENDFIL    LDA     EOF         INSERT END OF FILE MARKER
50               STA     BUFFER
55               LDA     THREE       SET LENGTH = 3
60               STA     LENGTH
65               JSUB    WRREC       WRITE EOF
70               LDL     RETADR      GET RETURN ADDRESS
75               RSUB                RETURN TO CALLER
80     EOF       BYTE    C'EOF'
85     THREE     WORD    3
90     ZERO      WORD    0
95     RETADR    RESW    1
100    LENGTH    RESW    1           LENGTH OF RECORD
105    BUFFER    RESB    4096        4096-BYTE BUFFER AREA
```

Forward reference

Output/ Assembly Listing



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
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		.			

Output

Record Type	Column	Content
1	1	Record Type = 1
	2-3	Byte Count
	4-7	The execution start address (Hex)
	8-9	Check Sum (CS)
2	1	Record Type = 2
	2-3	Byte Count
	4-7	The load address for the instruction/data (Hex)
	8-57	The code to be loaded in Hex
	58-59	Check Sum (CS)

Object Code

			ORG	4
0004	0011 0009		DC.W	17, 9
			ORG	$10
0010	5640 0004		LD.L	D0, 4
0014	5648 0006		LD.L	D1, 6
0018	1840		DIVU.L	D0, D1
001A	9408		BEQ	Zero
001C	5558 0008		LD.L	D3,#8
0020	63C3 0000		ST	D0,(D3)
0024	CC00	Zero	HLT	
0026			END	$10

Using this test program, your input to lab 1 should look like:
1030010EC
207000400110009DA
20F00105640000456480006184094O8BD
20D001C5558000863C30000CC003C

Design Approach

- One Pass: Line by Line
  - forward reference?
- Two Passes:
  - Pass 1
  - Pass2
- Intermediate files

One-Pass Assembler

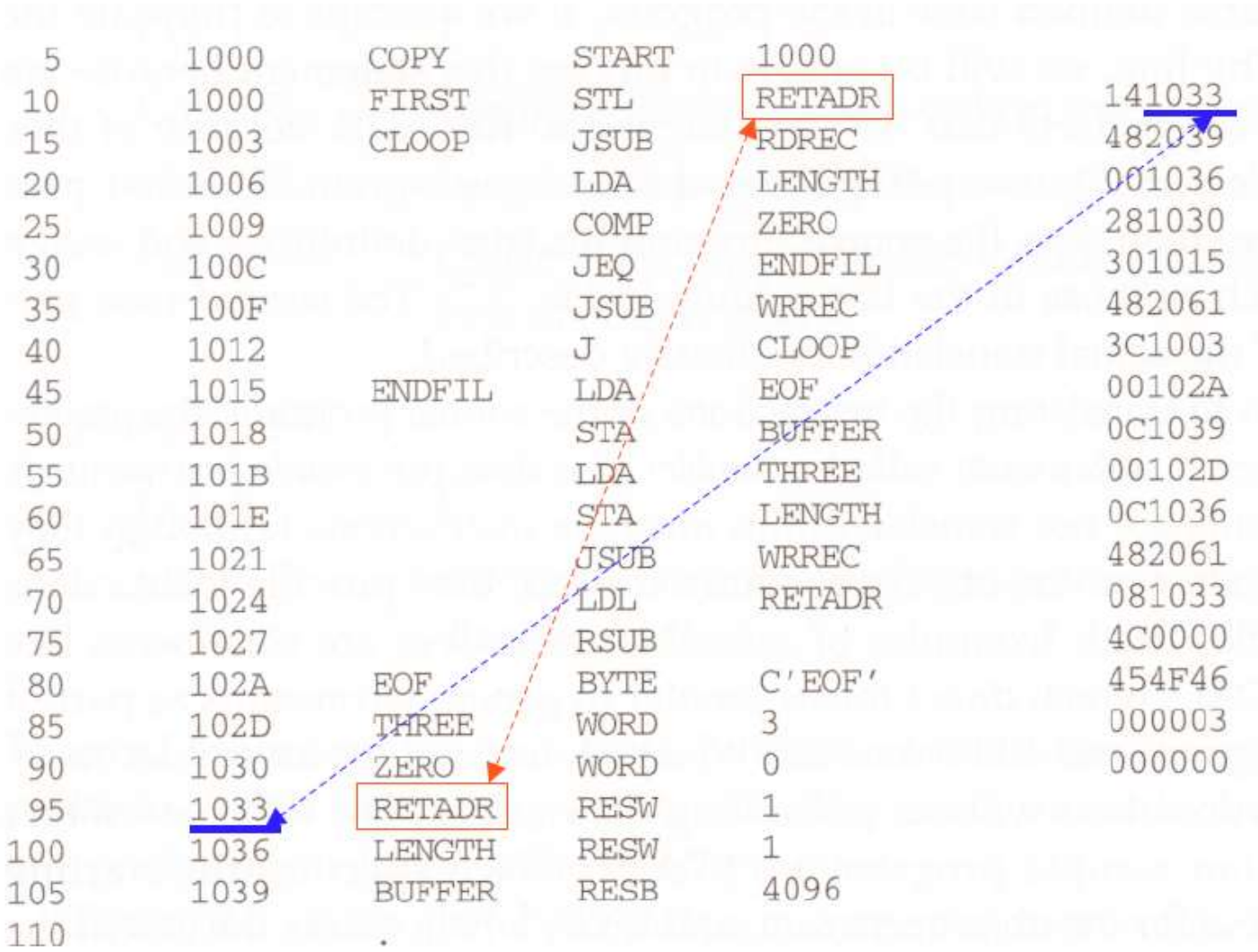
- The main problem is forward reference.
- Eliminating forward reference
  - Simply ask the programmer to define variables before using them.
- However, ?!
  - Backward jumps is too restrictive.
  - Forward jumps (Subroutine calls, Loops)

Input

```
 5     COPY      START   1000        COPY FILE FROM INPUT TO OUTPUT
10     FIRST     STL     RETADR      SAVE RETURN ADDRESS
15     CLOOP     JSUB    RDREC       READ INPUT RECORD
20               LDA     LENGTH      TEST FOR EOF (LENGTH = 0)
25               COMP    ZERO
30               JEQ     ENDFIL      EXIT IF EOF FOUND
35               JSUB    WRREC       WRITE OUTPUT RECORD
40               J       CLOOP       LOOP
45     ENDFIL    LDA     EOF         INSERT END OF FILE MARKER
50               STA     BUFFER
55               LDA     THREE       SET LENGTH = 3
60               STA     LENGTH
65               JSUB    WRREC       WRITE EOF
70               LDL     RETADR      GET RETURN ADDRESS
75               RSUB                RETURN TO CALLER
80     EOF       BYTE    C'EOF'
85     THREE     WORD    3
90     ZERO      WORD    0
95     RETADR    RESW    1
100    LENGTH    RESW    1           LENGTH OF RECORD
105    BUFFER    RESB    4096        4096-BYTE BUFFER AREA
```

Forward reference

Output/ Assembly Listing



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
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		.			

Output Object Code

Record Type	Column	Content
1	1	Record Type = 1
	2-3	Byte Count
	4-7	The execution start address (Hex)
	8-9	Check Sum (CS)
2	1	Record Type = 2
	2-3	Byte Count
	4-7	The load address for the instruction/data (Hex)
	8-57	The code to be loaded in Hex
	58-59	Check Sum (CS)

Example

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	48203D
20	2015		LDA	LENGTH	00100C
25	2018		COMP	ZERO	281006
30	201B		JEQ	ENDFIL	302024
35	201E		JSUB	WRREC	482062
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	482062
70	2033		LDL	RETADR	081009
75	2036		RSUB		4C0000
110					

Example

```
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            30205B
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
195
```

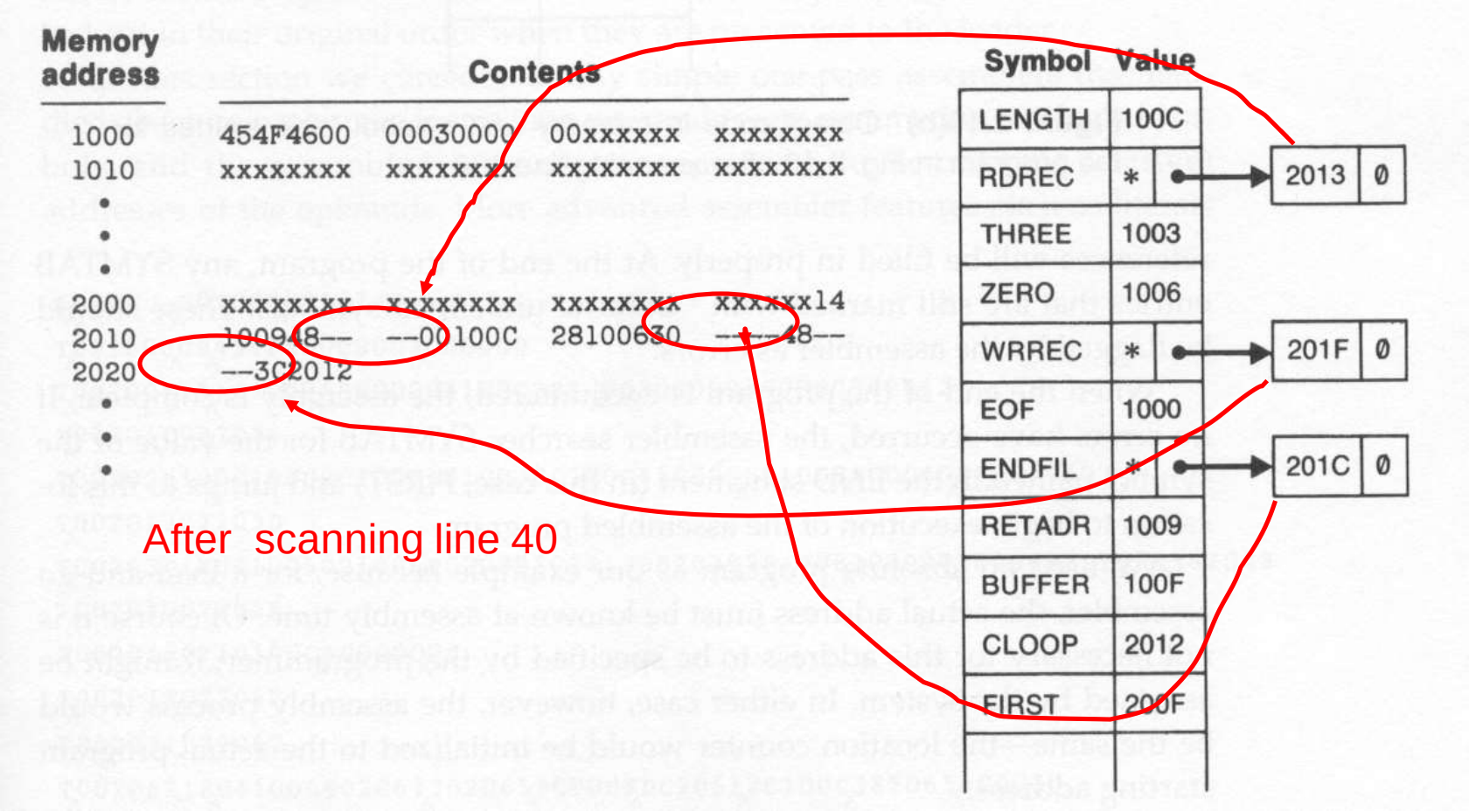
Forward Reference

- For any symbol that has not yet been defined
  1. omit the address translation
  2. insert the symbol into SYMTAB, and mark this symbol undefined
  3. the address that refers to the undefined symbol is added to a list of forward references associated with the symbol table entry
  4. when the definition for a symbol is encountered, the proper address for the symbol is then inserted into any instructions previous generated according to the forward reference list

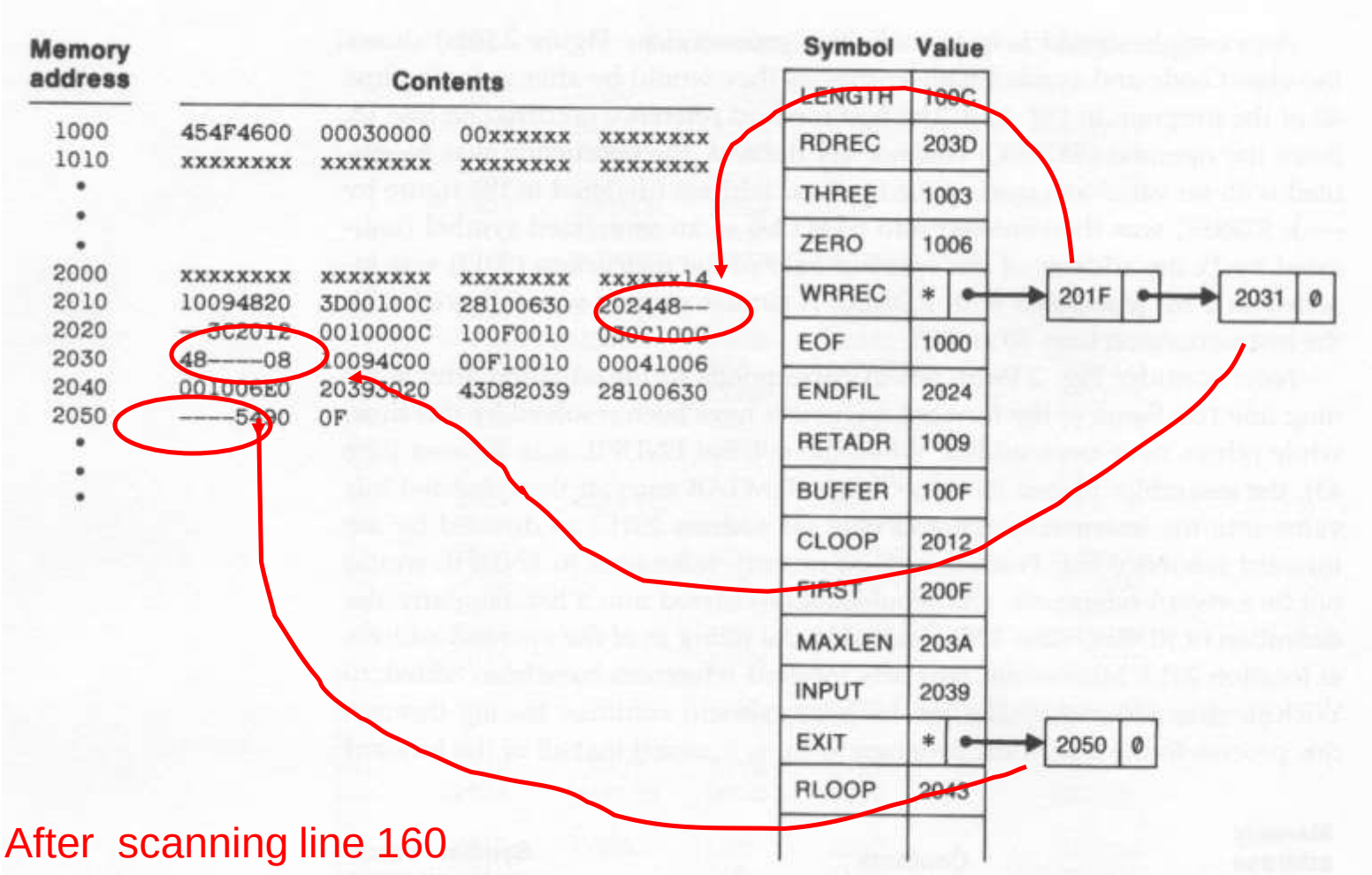
Load-and-go Assembler (Cont.)

- At the end of the program
  - any SYMTAB entries that are still marked with * indicate undefined symbols
  - search SYMTAB for the symbol named in the END statement and jump to this location to begin execution
- The actual starting address must be specified at assembly time

Processing Example



Processing Example



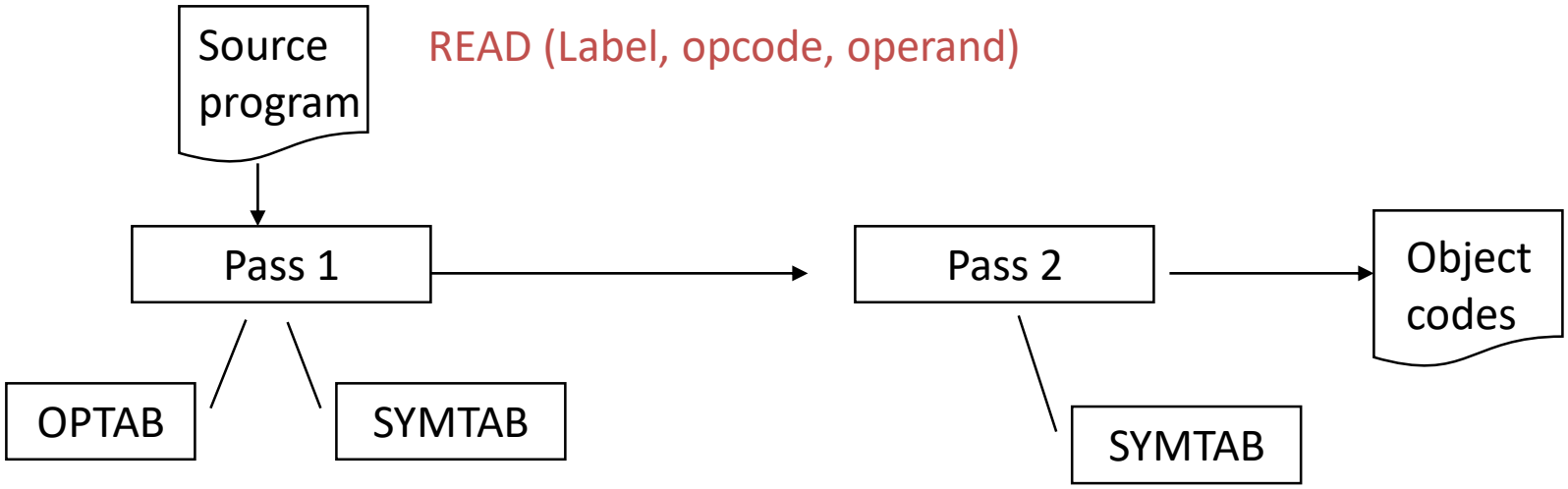
Two Passes

- Pass 1
  - Validate Opcodes
  - Assign addresses to all statements in the program
  - Scan the source for labels and save their values
  - Perform some processing of assembler directives
    - Determine the length of areas defined by DC, DS
- Pass 2
  - Translate/assemble the instructions
  - Generate Data Values defined by DC
  - Process the rest of the assembler directives
  - Write the Object Code and Assembly Listing

Data Structures

- Operation Code Table (OPTAB)
  - Opcode, Instruction format, and length
  - Pass 1: Validate opcodes
  - Pass2: Assemble instructions
- Symbol Table (SYMTAB)
  - Label name and value, error flags
  - Pass 1: Created!
  - Lookup symbols to insert in assembled instr.
- Location Counter
  - Initialed to the Org or End

A Simple Two Pass Assembler Implementation



Hash Tables

- OPTAB is static (access)
  - Retrieval efficiency
  - Key : Mnemonic operation
- SYMTAB (add, access)
  - Insertion and Retrieval efficiency
  - Key: Label Name
    - LOOP1, LOOP2, LOOP3…, A, X, Y, Z…

OPTAB (operation code table)

- Content
  - The mapping between mnemonic and machine code. Also include the instruction format, available addressing modes, and length information.
- Characteristic
  - Static table. The content will never change.
- Implementation
  - Array or hash table. Because the content will never change, we can optimize its search speed.
- In pass 1, OPTAB is used to look up and validate mnemonics in the source program.
- In pass 2, OPTAB is used to translate mnemonics to machine instructions.

Symbol Table (SYMTAB)

- Content
  - Include the label name and value (address) for each label in the source program.
  - Include type and length information (e.g., int64)
  - With flag to indicate errors (e.g., a symbol defined in two places)
- Characteristic
  - Dynamic table (I.e., symbols may be inserted, deleted, or searched in the table)
- Implementation
  - Hash table can be used to speed up search
  - Because variable names may be very similar (e.g., LOOP1, LOOP2), the selected hash function must perform well with such non-random keys.

Pass 1 Pseudo Code

Pass 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
   while OPCODE ≠ 'END' do
      begin
         if this is not a comment line then
            begin
               if there is a symbol in the LABEL field then
                  begin
```

Pass 1

```
            search SYMTAB for LABEL
            if found then
                set error flag (duplicate symbol)
            else
                insert (LABEL,LOCCTR) into SYMTAB
        end {if symbol}
    search OPTAB for OPCODE
    if found then
        add 3 {instruction length} to LOCCTR
    else if OPCODE = 'WORD' then
        add 3 to LOCCTR
    else if OPCODE = 'RESW' then
        add 3 * #[OPERAND] to LOCCTR
    else if OPCODE = 'RESB' then
        add #[OPERAND] to LOCCTR
```

Pass 1

```
            else if OPCODE  = 'BYTE' then
                begin
                    find length of constant in bytes
                    add length to LOCCTR
                end {if BYTE}
            else
                set error flag (invalid operation code)
        end {if not a comment}
    write line to intermediate file
    read next input line
end {while not END}
write last line to intermediate file
save (LOCCTR - starting address) as program length
end {Pass 1}
```

Pass 2 Pseudo Code

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
```

Pass 2

```
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
```

Pass 2

```
            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}
```