

Visualization of Assembly - Code Conversion into Machine Code

Pentti Lappalainen

University of Oulu, Electrical Engineering Department

pl1@ee.oulu.fi

Abstract

This paper describes the conversion of symbolic machine instructions provided by the user through a keyboard into machine code, which computer hardware is able to interpret. Key punches are first coded in ASCII-format and stored in computer memory. Then this information is processed by an assembler program using several tables and converted into binary object program for execution. This functionality is simulated and displayed on a screen to facilitate the comprehension to an undergraduate student. This program is available at URL <http://www.ee.oulu.fi/~pl1/kt1/>.

1. Introduction

Understanding the conversion of symbolic machine instructions into binary machine code is an essential part of undergraduate education in computer engineering. Conversion is made up of several phases and the whole process is fairly complicated and long story to be told in the class. It is quite an effort to the students to build a mental image of the process and comprehend it. This process can be facilitated by using computer simulation to emphasize key issues and guide the student in the thought process.

This paper describes the properties of computer software to simulate the conversion process of a two-pass assembler on computer display. It consists of three web pages, the first of which shows how the textual program input from a keyboard is converted into ASCII-code and stored in the mem-

ory in binary form. The second page shows the scanning of the source assembly code to create the address symbol table. This is necessary to provide numerical values to the user-defined variables. The final page shows the operation of several table lookup procedures to convert assembly language statement into binary machine code.

2. Conversion of text strings into ASCII-format

For easy access for humans characters are coded into positional format in a keyboard. Punching the keys humans describe a problem to be solved in a formal language (assembly language is used in this presentation). The description is made up of statements, which describe the algorithm to be executed. Statements consist of short text strings, items to represent actions to be taken and objects to be acted upon.

Text strings are converted into binary ASCII-format by a hardware converter, which recognizes the key punched by a user and provides respective binary representation of the character, Fig. 1.

The first screen of the simulator program shows how an individual key punch is converted by a table lookup process into binary code and then stored into computer memory to wait subsequent processing. It shows the structure of some assembly language statements and how meaningful information is coded into short character strings as well.

When the user has input the whole program to the computer memory, s/he can proceed starting the first phase of the assembler program.

```

ORG 100
LDA SUB
CMA
INC
ADD MIN
STA DIF
HLT
MIN DEC 83
SUB DEC -23
DIF HEX 0
END

```

punch the button D in the keyboard

Character D from the table

A	41	Q	51	6	36
B	41	R	52	7	37
C	43	S	53	8	38
D	44	T	54	9	39
E	45	U	55	space	20
F	46	V	56	(28
G	47	W	57)	29
H	48	X	58	*	2A
I	49	Y	59	+	2B
J	4A	Z	5A	,	2C
K	4B	0	30	-	2D
L	4C	1	31	.	2E
M	4D	2	32	/	2F
N	4E	3	33	=	3D
O	4F	4	34	CR	0D
P	50	5	35		

store 44 in memory

```

OR      4F 52
G 'space' 47 20
1 0      31 30
0 'CR'   30 0D
LD      4C 44
A 'space' 41 20
SU      53 55
B 'CR'   42 0D
CM      43 4D
A 'CR'   41 0D
IN      49 4E
C 'CR'   43 0D
AD      41 44
D 'space' 44 20
MI      4D 49
N 'CR'   4E 0D
ST      53 54
A 'space' 41 20
DI      44 59
F 'CR'   46 0D
HL      48 4C
T 'CR'   54 0D
MI      4D 49
N 'comma' 4E 2C
DE      44 45
C 'space' 43 20
8 3     38 33
'CR' S   0D 53
UB      55 42
'comma' D 2C 44
EC      45 43
'space' - 20 2D
2 3     32 33
DI      44 49
F,      46 2C
HE      48 45
X 0     58 30
'CR' E   0D 45
ND      4E 44

```

SP

01001111	01010010
01000111	00100000
00110001	00110000
00110000	00001101
01001100	01000100
01000001	00100000
01010011	01010101
01000010	00001101
01000011	01001101
01000001	00001101
01001001	01001110
01000011	00001101
01000001	01000100
01000100	00100000
01001101	01001001
01001110	00001101
01010011	01010100
01000001	00100000
01000100	01001001
01000110	00001101
01001000	01001100
01010100	00001101
01001101	01001101
01001110	00101100
01000100	01000101
01000011	00100000
00111000	00110011
00001101	01010011
01010101	01000010
00101100	01000100
01000101	01000011
00100000	00101101
00110010	00110011
01000100	01001001
01000110	00101100
01001000	01000101
01011000	00110000
00001101	01000101
01001100	01000100

Figure 1. Conversion and storage of characters from a keyboard.

3. First pass of the assembler program

Structure of the assembly language: instruction set, commands, addressing modes etc. are completely defined by the designer. The user is tied up to these decisions.

However, the user has complete freedom to give names to the variables s/he needs when describing the algorithm to be executed. During the first pass of the assembly process, a variable symbol table is created to store all the user variable names and associate a definite numerical address to be referenced during the second pass of the process, Fig. 2.

Assembler reads a statement in to a buffer and then scans it item by item to recognize user defined variables. They may appear at the beginning of a statement, where they are recognized from a separation mark (in this assembly language it is a comma). When the assembler finds an item it

writes the variable name into a symbol table and associates the value of a counter to this variable. The counter is used to count the position of the present statement in the program, which represents the address of the memory location, where it will be stored during execution.

Another position of a user defined variable is after a symbolic machine command. In this case no numerical address is associated with the variable, but it is stored in the address table and marked as a name which need to be associated with a numerical value in some phase of the process.

When the whole program is analyzed in this way and no errors have been found, second pass of the assembly process is run. Screen 2 of the simulator program shows the assembly language statement scanning and the buildup of the symbol address table.

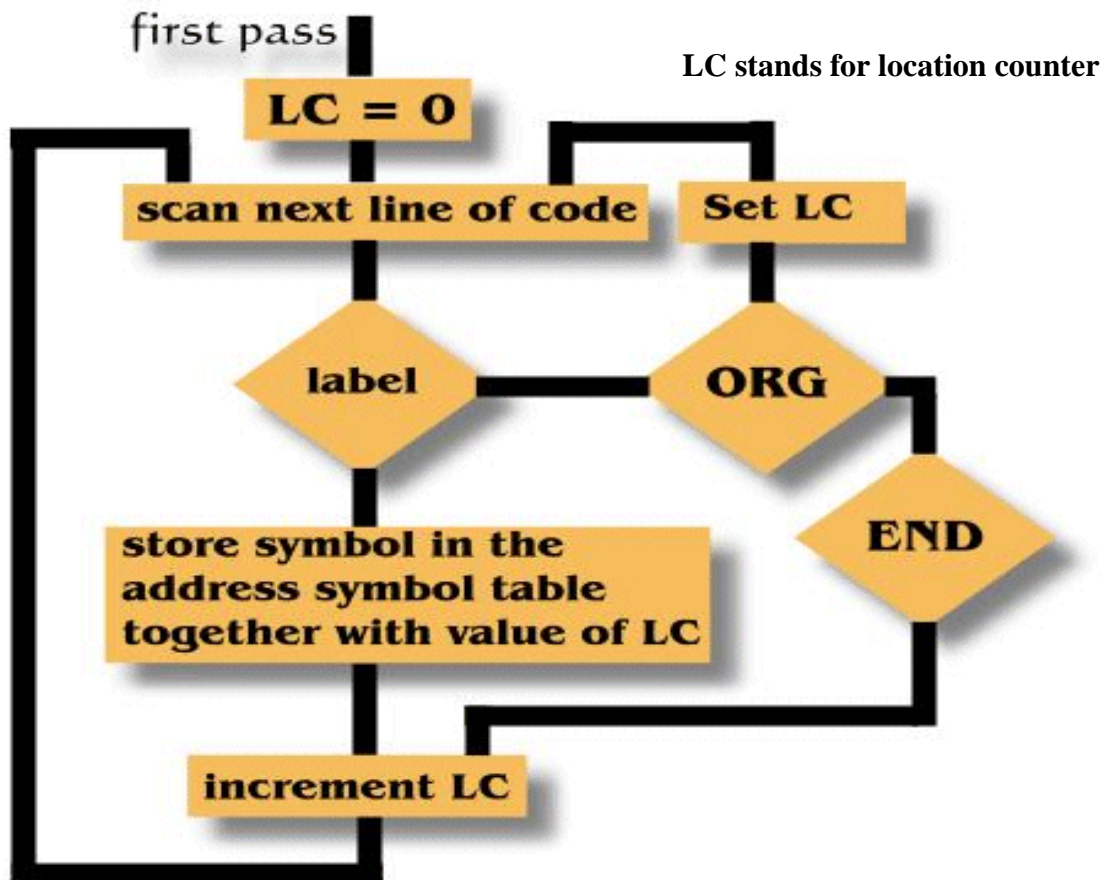


Figure 2. First pass of the assembler program.

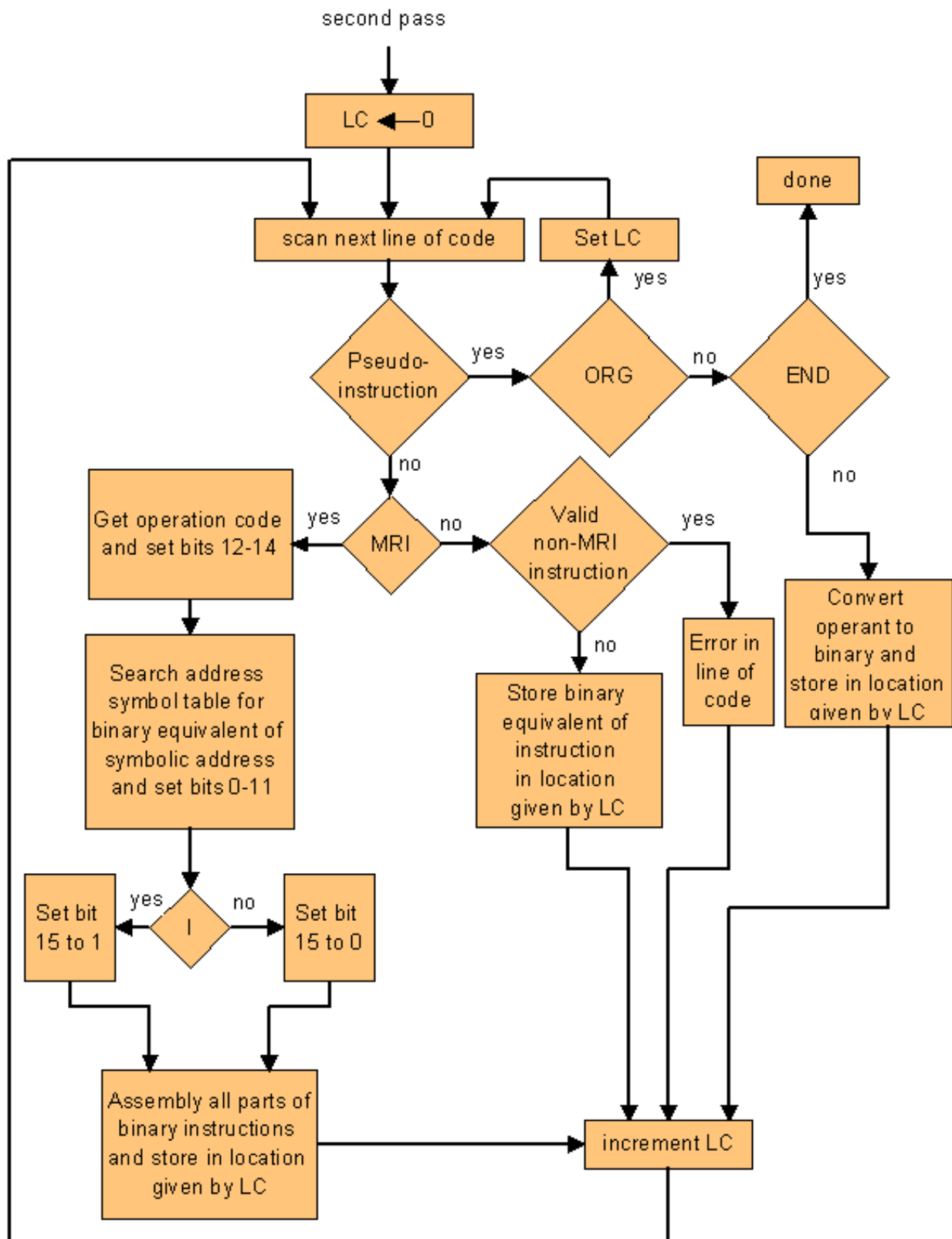


Figure 3. Second pass of the assembler program.

4. Second pass of the assembler program

During the second pass of the assembler program statements are scanned item by item and respective binary representation is searched from the tables included in the assembler: machine command, symbol address and pseudoinstruction tables.

If a statement starts with an item ending to a comma, it is skipped and the processing proceeds to next item. If it is a machine command, machine command table is searched and the corresponding binary equivalent extracted to a buffer to be used later in the assembly process.

After a machine command a variable name follows. Symbol address table is searched, binary address is extracted and assembled with the machine command code in the buffer. Then address mode information is processed if available and assembled to the final machine code of the statement.

In this way the whole program is processed statement by statement and assembled into binary object code. When the END pseudoinstruction is found, the physical end of the program is there and the user can proceed to execute the program.

Fig. 3 shows the process of converting an assembly language statement into a binary machine code using several tables and a table lookup procedure.

Screen 3 of the simulator shows how the program finds the required items in the tables provided and how this information is used to convert the symbolic assembly language statement into object code.

5. Conclusion

The simulator program described simulates the functionality of a simple two pass assembler program from keyboard entry of the program to the conversion of symbolic code into machine executable object code. The goal of the simulator design has been the facilitation of undergraduate student comprehension of the abstract assembly process.

6. References

- [1] M. Mano, *Computer System Architecture*. Prentice-Hall 1993.