Applied Computer Letters (ACL)

***DOI:** http://doi.org/10.7508/acl.01.2019.01.03*

ISSN: 2377-6242 (Print)
ISSN: 2377-8156 (Online)

ARTICLE

COMMON PROBLEMS IN THE LEARNING OF ASSEMBLY LANGUAGE PROGRAMMING

Thomas J. Liebler*

College of Engineering and Physical Sciences, University of New Hampshire, Durham NH 03824, United States
****Corresponding Author E-mail: Liebler@hotmail.com***

ARTICLE DETAILS

Article History:

Received 3 February 2019
Accepted 2 March 2019
Available online 26 March 2019

ABSTRACT

Language is one of the most closely related and direct programming languages with hardware and the highest efficiency in time and space. It is one of the compulsory courses of computer application technology in Colleges and universities. It plays an important role in training students to master programming technology and familiarize themselves with computer operation and program debugging technology. In the paper, several difficult problems encountered here are written in order to provide a clear understanding of the difficult problems in assembly language programming.

KEYWORDS

Storage space allocation, assembly language, language programming, address calculation.

1. INTRODUCTION

Assembly language is a programming language directly oriented to processors. The processor works under the control of instructions. Each instruction that the processor can recognize is called a machine instruction [1]. Each processor has its own set of instructions that can be recognized, called instruction set [2]. When the processor executes instructions, it takes different actions according to different instructions and completes different functions. It can not only change its internal working state, but also control the working state of other peripheral circuits [3].

Because assembly language has the characteristic of "machine dependence", when programmers write programs in assembly language, they can fully arrange various resources within the machine, so that they are always in the best use state. The program written in this way has short execution code and fast execution speed. Assembly, language is one of the most closely related and direct programming languages with hardware and the highest efficiency in time and space. It is one of the compulsory courses of computer application technology in Colleges and universities. It plays an important role in training students to master programming technology and familiarize themselves with computer operation and program debugging technology. This paper is the author's teaching of assembly language course. Several difficult problems encountered here are written for your reference.

2. ASSIGNMENT OF PROGRAM SPACE IN ASSEMBLER RUNTIME

Storage space allocation is needed when assembler runs, which is done by the operating system. When each assembler runs, it first allocates a segment prefix PSP, because DOS uses PSP to communicate with the loaded program.

PSP is 256 bytes, as shown in Figure 1. When the executable file is generated to a certain extent, the program is first transferred into memory when it is executed. At this time, the segment address of the program stored in memory is stored in DS. PSP occupies the first 256 bytes of DS:0000H segment. The contents are some instructions of the program, such as how much space the program occupies, etc. Then the real program address is the program address, and CS is pointed here, IP. Setting it to 0000, it is for this reason why CS is 10H larger than DS in general.

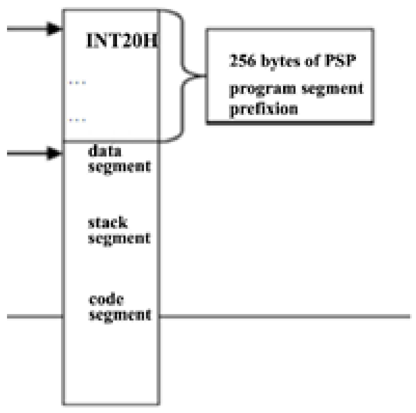
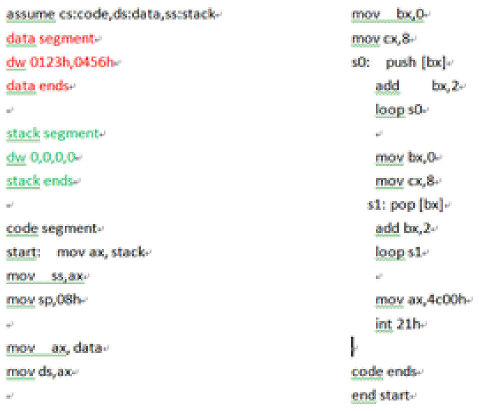


Figure 1: PSP.

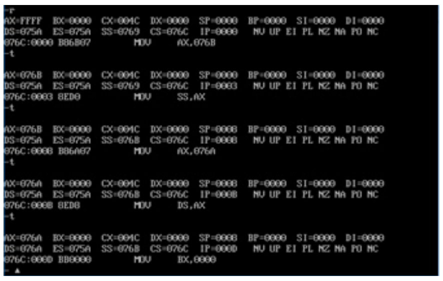
The following is a practical program to observe this effect. The program of Figure 2 runs in DEBUG step by step as shown in Figure 3 and Figure

4. From the execution process of Figure 3, it can be seen that the address space of the program starts from 075A:0000, followed by the program segment prefix PSP of 256 bytes, followed by the data segment. The address space starts from 076A:0000, the data segment size is 16 bytes, then the stack segment of 16 bytes, and the address space starts from 076B:0000. The address space starts at 076C:0000. Through the above experiments, it can be clearly explained to students how the address space of an assembly language program is actually allocated in memory. For example, the following information can be obtained from the above run result graph: when a program has just been loaded into memory and has not yet started running, the DS register stores the starting address of the program segment prefix of the program, and this is the beginning address of the program segment prefix. The address is assigned by the DOS operating system. If the definition order of data segment and stack segment in the program is changed, that is to say, the green part of the above code is put in front of the red code segment, and run as shown in Figure 4. Compared with Figure 3 and Figure 4, it can be seen that after changing the order, the program runs only to change the allocation order of the address of the data segment and the stack segment, but the address allocated by the data segment and the stack segment has not changed.



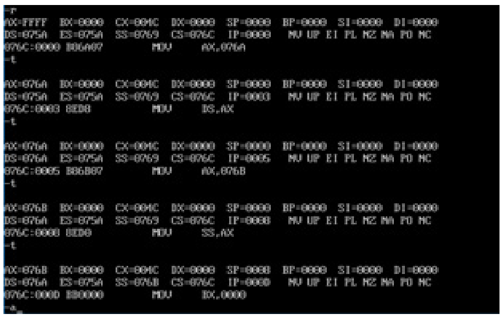
```
assume cs:code,ds:data,ss:stack
data segment
dw 0123h,0456h
data ends

stack segment
dw 0,0,0,0
stack ends

code segment
start:    mov ax, stack
mov    ss,ax
mov sp,08h

mov    ax, data
mov ds,ax

mov    bx,0
mov cx,8
s0:    push [bx]
      add       bx,2
      loop s0

      mov bx,0
      mov cx,8
    s1: pop [bx]
      add bx,2
      loop s1

      mov ax,4c00h
      int 21h

code ends
end start
```

Figure 2: Assembly programme 1.



```
-r
AX=FFFF  BX=0000  CX=004C  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=075A  ES=075A  SS=0769  CS=076C  IP=0000   NV UP EI PL NZ NA PO NC
076C:0000 B86B07        MOV     AX,076B
-t

AX=076B  BX=0000  CX=004C  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=075A  ES=075A  SS=0769  CS=076C  IP=0003   NV UP EI PL NZ NA PO NC
076C:0003 8ED0          MOV     SS,AX
-t

AX=076B  BX=0000  CX=004C  DX=0000  SP=0008  BP=0000  SI=0000  DI=0000
DS=075A  ES=075A  SS=076B  CS=076C  IP=0008   NV UP EI PL NZ NA PO NC
076C:0008 B86A07        MOV     AX,076A
-t

AX=076A  BX=0000  CX=004C  DX=0000  SP=0008  BP=0000  SI=0000  DI=0000
DS=075A  ES=075A  SS=076B  CS=076C  IP=000B   NV UP EI PL NZ NA PO NC
076C:000B 8ED8          MOV     DS,AX
-t

AX=076A  BX=0000  CX=004C  DX=0000  SP=0008  BP=0000  SI=0000  DI=0000
DS=076A  ES=075A  SS=076B  CS=076C  IP=000D   NV UP EI PL NZ NA PO NC
076C:000D BB0000        MOV     BX,0000
-
```

Figure 3: The running result 1 of assembly programme 1.

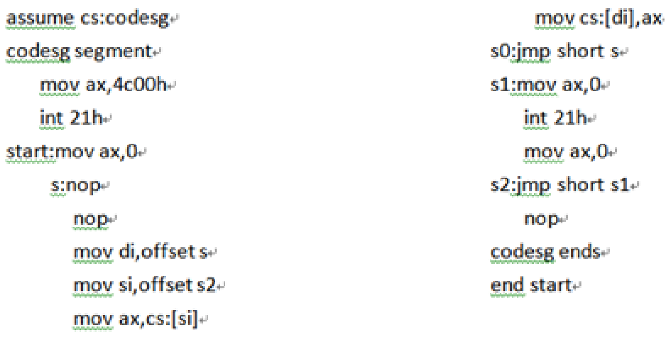


```
-r
AX=FFFF  BX=0000  CX=004C  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=075A  ES=075A  SS=0769  CS=076C  IP=0000   NV UP EI PL NZ NA PO NC
076C:0000 B86A07        MOV     AX,076A
-t

AX=076A  BX=0000  CX=004C  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=075A  ES=075A  SS=0769  CS=076C  IP=0003   NV UP EI PL NZ NA PO NC
076C:0003 8ED8          MOV     DS,AX
-t

AX=076A  BX=0000  CX=004C  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=076A  ES=075A  SS=0769  CS=076C  IP=0005   NV UP EI PL NZ NA PO NC
076C:0005 B86B07        MOV     AX,076B
-t

AX=076B  BX=0000  CX=004C  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=076A  ES=075A  SS=0769  CS=076C  IP=0008   NV UP EI PL NZ NA PO NC
076C:0008 8ED0          MOV     SS,AX
-t

AX=076B  BX=0000  CX=004C  DX=0000  SP=0008  BP=0000  SI=0000  DI=0000
DS=076A  ES=075A  SS=076B  CS=076C  IP=000D   NV UP EI PL NZ NA PO NC
076C:000D BB0000        MOV     BX,0000
-
```

Figure 4: The running result 2 of assembly programme 1.

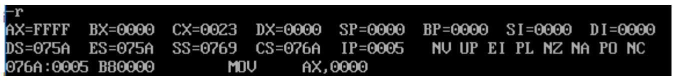
3. ADDRESS CALCULATION OF JUMP INSTRUCTIONS

The program is shown in Figure 5. The result of this program is shown in Figure 6. As can be seen from Figure 6, the instruction mov ax, 4c00h; the function of int 21h is simply to point the IP pointer to their next instruction: start: mov ax, 0.



```
assume cs:codesg
codesg segment
    mov ax,4c00h
    int 21h
start:mov ax,0
      s:nop
        nop
        mov di,offset s
        mov si,offset s2
        mov ax,cs:[si]
        mov cs:[di],ax
s0:jmp short s
s1:mov ax,0
    int 21h
    mov ax,0
s2:jmp short s1
    nop
codesg ends
end start
```

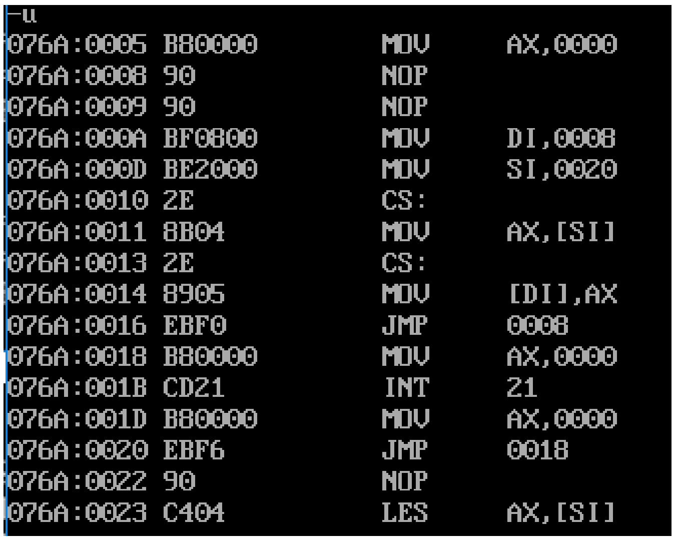
Figure 5: Assembly programme 2.



```
-r
AX=FFFF  BX=0000  CX=0023  DX=0000  SP=0000  BP=0000  SI=0000  DI=0000
DS=075A  ES=075A  SS=0769  CS=076A  IP=0005   NV UP EI PL NZ NA PO NC
076A:0005 B80000        MOV     AX,0000
```

Figure 6: The running result 1 of assembly programme 2.

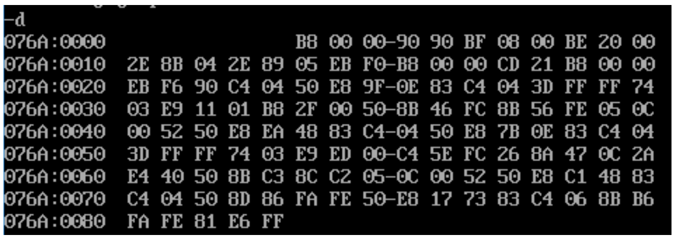
Using disassembly instructions, Figure 7 is obtained for the above programs. As can be seen from the figure, the physical address of label S1 is 076A:0018H, the physical address of label S2 is 076A: 0020H, and the instruction machine code of instruction s2: jmp short S1 is EBF6, in which the meaning of EB is jmp, and F6 represents the jump distance of the instruction. The distance value is calculated as follows: when the CPU reads in the machine code EBF6, the content of its IP pointer points to the next instruction nop. Its physical address is 076A:0022H, and the distance from the address to the physical address 076A:0018H of the label S1 is 10 (decimal system). Because it is a jump up, its value is negative. The complement of decimal system-10 is F6, and its address calculation is derived from 0022H + F6H = 0018H.



```
-u
076A:0005 B80000        MOV     AX,0000
076A:0008 90            NOP
076A:0009 90            NOP
076A:000A BF0800        MOV     DI,0008
076A:000D BE2000        MOV     SI,0020
076A:0010 2E            CS:
076A:0011 8B04          MOV     AX,[SI]
076A:0013 2E            CS:
076A:0014 8905          MOV     [DI],AX
076A:0016 EBF0          JMP     0008
076A:0018 B80000        MOV     AX,0000
076A:001B CD21          INT     21
076A:001D B80000        MOV     AX,0000
076A:0020 EBF6          JMP     0018
076A:0022 90            NOP
076A:0023 C404          LES     AX,[SI]
```

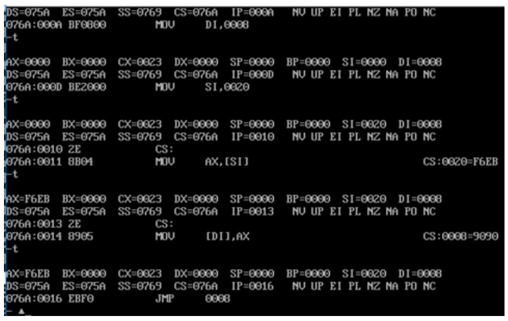
Figure 7: The running result 2 of assembly programme 2.

076A:0009 memory cells, and their machine codes are 90H, as shown in Figure 8. Then one-step running of the above program, when running to the instructions mov cs: [di], ax, as shown in Figure 9, the contents stored in the storage units 076A:0008 and 076A:0009 are programmed with EBF6, as shown in Figure 10. Continue to run instruction s0: jmp shorts, then CS and IP become 076A:0008H, then CPU reads machine code EBF6, then IP becomes 0010H. According to the above address jump calculation method, it is concluded that when machine code EBF6 is executed, IP will become 0000H, so the instruction mov ax, 4c00h, int 21h will be executed again, and when the instruction is executed again, the whole program will jump out of the node. Bundle, as shown in Figure 11. It can also be seen from the above process that when the instruction mov ax, 4c00h, int 21h is placed at the beginning of the program, its function is only to point the IP pointer to the next instruction address of this instruction, but when this instruction is executed during the execution of the program, it will cause the IP pointer to jump out of the whole program.



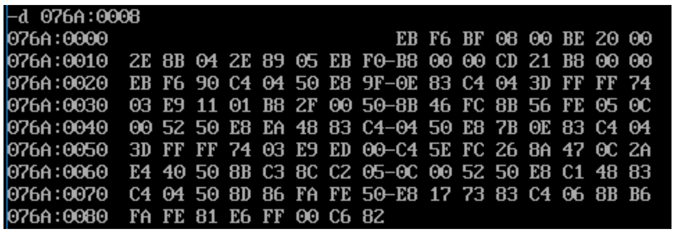
```
-d
076A:0000                         B8 00 00-90 90 BF 08 00 BE 20 00
076A:0010  2E 8B 04 2E 89 05 EB F0-B8 00 00 CD 21 B8 00 00
076A:0020  EB F6 90 C4 04 50 E8 9F-0E 83 C4 04 3D FF FF 74
076A:0030  03 E9 11 01 B8 2F 00 50-8B 46 FC 8B 56 FE 05 0C
076A:0040  00 52 50 E8 EA 48 83 C4-04 50 E8 7B 0E 83 C4 04
076A:0050  3D FF FF 74 03 E9 ED 00-C4 5E FC 26 8A 47 0C 2A
076A:0060  E4 40 50 8B C3 8C C2 05-0C 00 52 50 E8 C1 48 83
076A:0070  C4 04 50 8D 86 FA FE 50-E8 17 73 83 C4 06 8B B6
076A:0080  FA FE 81 E6 FF
```

Figure 8: The running result 3 of assembly programme 2.



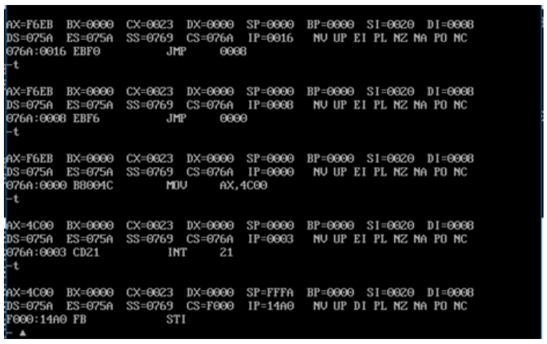
```
DS=075A  ES=075A  SS=0769  CS=076A  IP=000A   NV UP EI PL NZ NA PO NC
076A:000A BF0800        MOV     DI,0008
-t

AX=0000  BX=0000  CX=0023  DX=0000  SP=0000  BP=0000  SI=0000  DI=0008
DS=075A  ES=075A  SS=0769  CS=076A  IP=000D   NV UP EI PL NZ NA PO NC
076A:000D BE2000        MOV     SI,0020
-t

AX=0000  BX=0000  CX=0023  DX=0000  SP=0000  BP=0000  SI=0020  DI=0008
DS=075A  ES=075A  SS=0769  CS=076A  IP=0010   NV UP EI PL NZ NA PO NC
076A:0010 2E            CS:
076A:0011 8B04          MOV     AX,[SI]                        CS:0020=F6EB
-t

AX=F6EB  BX=0000  CX=0023  DX=0000  SP=0000  BP=0000  SI=0020  DI=0008
DS=075A  ES=075A  SS=0769  CS=076A  IP=0013   NV UP EI PL NZ NA PO NC
076A:0013 2E            CS:
076A:0014 8905          MOV     [DI],AX                        CS:0008=9090
-t

AX=F6EB  BX=0000  CX=0023  DX=0000  SP=0000  BP=0000  SI=0020  DI=0008
DS=075A  ES=075A  SS=0769  CS=076A  IP=0016   NV UP EI PL NZ NA PO NC
076A:0016 EBF0          JMP     0008
-▲
```

Figure 9: The running result 4 of assembly programme 2.



```
-d 076A:0008
076A:0000                                  EB F6 BF 08 00 BE 20 00
076A:0010  2E 8B 04 2E 89 05 EB F0-B8 00 00 CD 21 B8 00 00
076A:0020  EB F6 90 C4 04 50 E8 9F-0E 83 C4 04 3D FF FF 74
076A:0030  03 E9 11 01 B8 2F 00 50-8B 46 FC 8B 56 FE 05 0C
076A:0040  00 52 50 E8 EA 48 83 C4-04 50 E8 7B 0E 83 C4 04
076A:0050  3D FF FF 74 03 E9 ED 00-C4 5E FC 26 8A 47 0C 2A
076A:0060  E4 40 50 8B C3 8C C2 05-0C 00 52 50 E8 C1 48 83
076A:0070  C4 04 50 8D 86 FA FE 50-E8 17 73 83 C4 06 8B B6
076A:0080  FA FE 81 E6 FF 00 C6 82
```

Figure 10: The running result 5 of assembly programme 2.

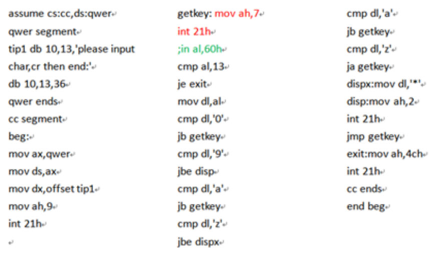


```
AX=F6EB  BX=0000  CX=0023  DX=0000  SP=0000  BP=0000  SI=0020  DI=0008
DS=075A  ES=075A  SS=0769  CS=076A  IP=0016   NV UP EI PL NZ NA PO NC
076A:0016 EBF0          JMP     0008
-t

AX=F6EB  BX=0000  CX=0023  DX=0000  SP=0000  BP=0000  SI=0020  DI=0008
DS=075A  ES=075A  SS=0769  CS=076A  IP=0008   NV UP EI PL NZ NA PO NC
076A:0008 EBF6          JMP     0000
-t

AX=F6EB  BX=0000  CX=0023  DX=0000  SP=0000  BP=0000  SI=0020  DI=0008
DS=075A  ES=075A  SS=0769  CS=076A  IP=0000   NV UP EI PL NZ NA PO NC
076A:0000 B8004C        MOV     AX,4C00
-t

AX=4C00  BX=0000  CX=0023  DX=0000  SP=0000  BP=0000  SI=0020  DI=0008
DS=075A  ES=075A  SS=0769  CS=076A  IP=0003   NV UP EI PL NZ NA PO NC
076A:0003 CD21          INT     21
-t

AX=4C00  BX=0000  CX=0023  DX=0000  SP=FFFA  BP=0000  SI=0020  DI=0008
DS=075A  ES=075A  SS=0769  CS=F000  IP=14A0   NV UP DI PL NZ NA PO NC
F000:14A0 FB            STI
-▲
```

Figure 11: The running result 6 of assembly programme 2.

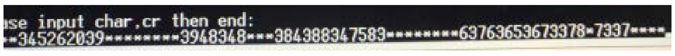
4. THE EXPLANATION OF HARD INTERRUPT AND SOFT INTERRUPT

Figure 12 is an assembler program that uses keyboard and screen for input and output. The result is that the number is displayed on the screen when the number is input from the keyboard, and the "*" number is displayed when other characters are input. The results are shown in Figure 13.



```
assume cs:cc,ds:qwer
qwer segment
tip1 db 10,13,'please input
char,cr then end:'
db 10,13,36
qwer ends
cc segment
beg:
mov ax,qwer
mov ds,ax
mov dx,offset tip1
mov ah,9
int 21h

getkey: mov ah,7
int 21h
;in al,60h
cmp al,13
je exit
mov dl,al
cmp dl,'0'
jb getkey
cmp dl,'9'
jbe disp
cmp dl,'a'
jb getkey
cmp dl,'z'
jbe dispx
cmp dl,'a'
jb getkey
cmp dl,'z'
ja getkey
dispx:mov dl,'*'
disp:mov ah,2
int 21h
jmp getkey
exit:mov ah,4ch
int 21h
cc ends
end beg
```

Figure 12: Assembly programme 3.



```
ise input char,cr then end:
**345262039********3948348***384388347583*********63763653673378*7337****
```

Figure 13: The running result of assembly programme 3.

Internal interruption means that the CPU does not follow the instructions that have just been executed down, but instead moves on to handle this particular information. External interruption refers to the CPU in the computer system, in addition to the ability to execute instructions and operations, but also should be able to control external equipment, receive their input and output to them. Statements in the program Mov ah, 7, int 21h belongs to function call No. 7 in DOS interrupt, which receives keyboard input information and belongs to soft interrupt. Its process is to find the entry address of interrupt program according to int 21h instruction. This interrupt program is used to read keyboard input characters, and the interrupt is triggered by int 21h instruction; instruction in al, 60H is the information read directly into port 60h, and the information of port 60H is the same as that of port 60h. Sample from the keyboard input, the reading process is still to use the same int 21h instruction pointed to the interrupt service program, but this call process is triggered by the keyboard keys caused by the changes in the keyboard internal circuit switch state, belongs to hard interrupt. The substitution of these three instructions well explains the difference and relationship between soft interrupt and hard interrupt.

5. CONCLUSIONS

For beginners, assembly language, because of its close combination with hardware, if their hardware knowledge is not enough, then the course will be more difficult to grasp; For teachers, some of the concepts are particularly difficult to explain clearly. This paper makes a preliminary discussion on some difficult problems in assembly language through examination, including assignment of program space, address calculation of jump instructions and explanation of hard interrupt and soft interrupt, hoping that through a few practical examples, it can be helpful to teachers and students engaged in this teaching work.

REFERENCES

[1] Qian, X. J. (2004). Recognition of Assembly Language. Teaching in China University, 47-49.

[2] Wang, S. (2013). Assembly Language (3rd ed.). Beijing: Tsinghua University Press.

[3] Wu, W., Wang, X., & Liu, X. Y. (2009). Teaching Reform of Assembly Language Programming. Journal of Southwest Normal University (Natural Science Edition), 34, 201-204.