8. Să se afișeze, pentru fiecare număr de la 32 la 126, valoarea numărului (în baza 8) și caracterul cu acel cod ASCII.

main.asm

Data Segment

```
pattern DB "  chr(%04X) = '%c'", 10, 13, 0 ; \l\n
```

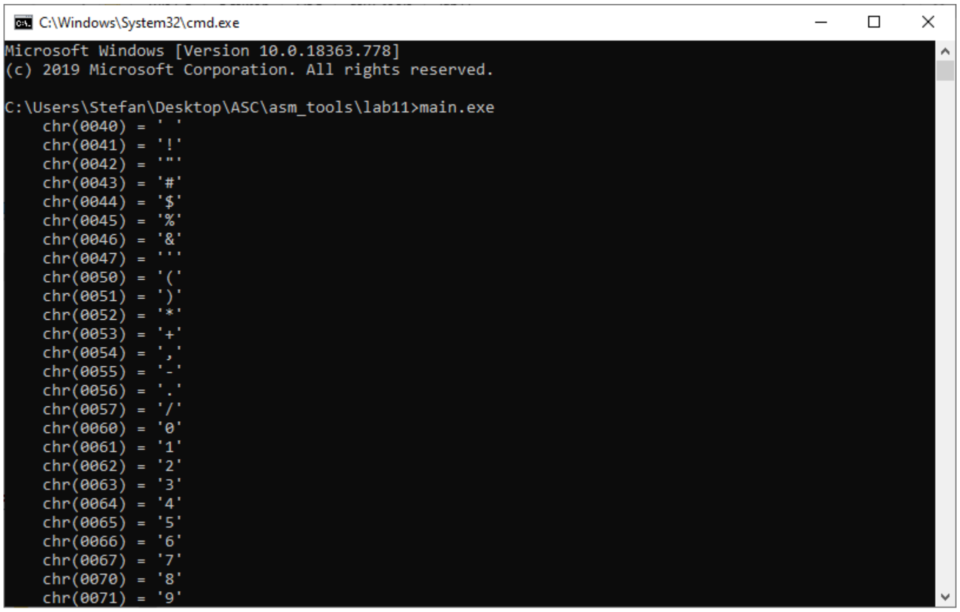
Code Segment

```
mov ECX, 127-32         ; ECX = count[32..126]
.iterate:
    pushad              ; backup reg state
    neg ECX             ; ECX = -ECX
    add ECX, 127        ; ECX += 127 := 32..126

    ; printf(pattern, literal_octal(ECX), ECX);
    ; param printf '%c', param literal_octal:
    push ECX
    call literal_octal  ; EAX = literal_octal(ECX)
    push EAX            ; param printf '%X'

    push pattern        ; param printf str
    call [printf]       ; print row
    add ESP, 4*3        ; clear stack

    popad               ; retrieve reg state
    loop .iterate       ; for(ECX=127-32;ECX--;)
```

b8.asm (modul)

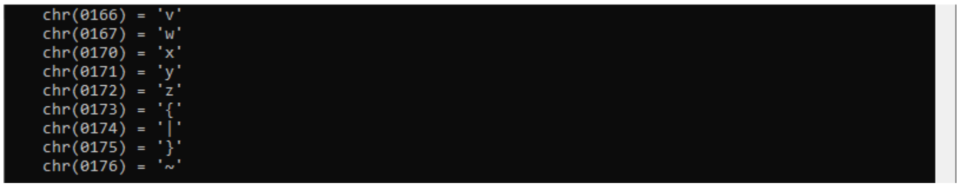
Code Segment

```
literal_octal:
    ; EAX = int(str(dec_to_oct(byte([ESP + 4]))))
    ; ex. [ESP+4]=9 => EAX = 11
    ; b7b6b5b4b3b2b1b0 => oooob7b6_oob5b4b3_oob2b1b0, oo:=0b
    ; ex. 162 = A2h = 1010_0010b => 0010_0100_0010b = 242h
    ; (<=> 242 oct)
    ; ex: printf("%X",literal_octal(val));

    xor EAX, EAX        ; EAX = 0
    mov AL, [ESP+4]     ; AL = byte([ESP+4])

    mov CL, 3           ; solve nibble 0
    call .insert_bit_0  ; EAX = b7b6_b5b4b3_oob2b1b0
    mov CL, 7           ; solve nibble 1
    ; fallthrough .insert_bit_0
    .insert_bit_0:
        ; CL < 31
        ; LEFT = bits EAX[31..CL+1]
        ; RIGHT = bits EAX[CL..0]
        ; EAX = {LEFT,RIGHT} := ((EAX xor RIGHT) << 1)|RIGHT
        mov EBX, 1    ; EBX = 1
        shl EBX, CL   ; EBX = 100...0 (cnt(0)=CL)
        dec EBX       ; EBX =  11...1 (RIGHT mask)
        and EBX, EAX  ; EBX = RIGHT
        xor EAX, EBX  ; EAX = {LEFT,00..0}
        add EAX, EAX  ; EAX <<= 1, avoid CL
        or  EAX, EBX  ; EAX = LEFT_0_RIGHT
        ret
```



```
Microsoft Windows [Version 10.0.18363.778]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Stefan\Desktop\ASC\asm_tools\lab11>main.exe
    chr(0040) = ' '
    chr(0041) = '!'
    chr(0042) = '"'
    chr(0043) = '#'
    chr(0044) = '$'
    chr(0045) = '%'
    chr(0046) = '&'
    chr(0047) = '''
    chr(0050) = '('
    chr(0051) = ')'
    chr(0052) = '*'
    chr(0053) = '+'
    chr(0054) = ','
    chr(0055) = '-'
    chr(0056) = '.'
    chr(0057) = '/'
    chr(0060) = '0'
    chr(0061) = '1'
    chr(0062) = '2'
    chr(0063) = '3'
    chr(0064) = '4'
    chr(0065) = '5'
    chr(0066) = '6'
    chr(0067) = '7'
    chr(0070) = '8'
    chr(0071) = '9'
```

...



```
    chr(0166) = 'v'
    chr(0167) = 'w'
    chr(0170) = 'x'
    chr(0171) = 'y'
    chr(0172) = 'z'
    chr(0173) = '{'
    chr(0174) = '|'
    chr(0175) = '}'
    chr(0176) = '~'
```

8. Să se afişeze, pentru fiecare număr de la 32 la 126, valoarea numărului (în baza 8) şi caracterul cu acel cod ASCII.

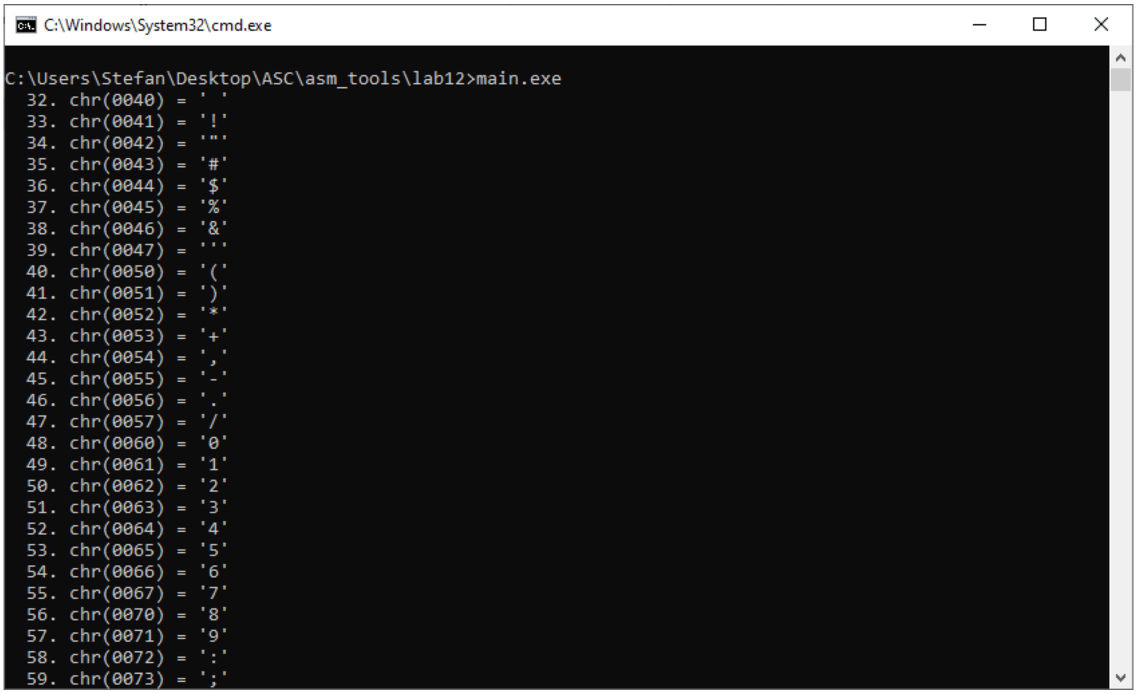
main.c

```c
#include <stdio.h>

int literal_octal(int n);

int main()
{
    for(char i=32; i <= 126; i++)
        printf(
            " %3i. chr(%04X) = '%c'\n",
            i,
            literal_octal(i),
            i
        );
    return 0;
}
```

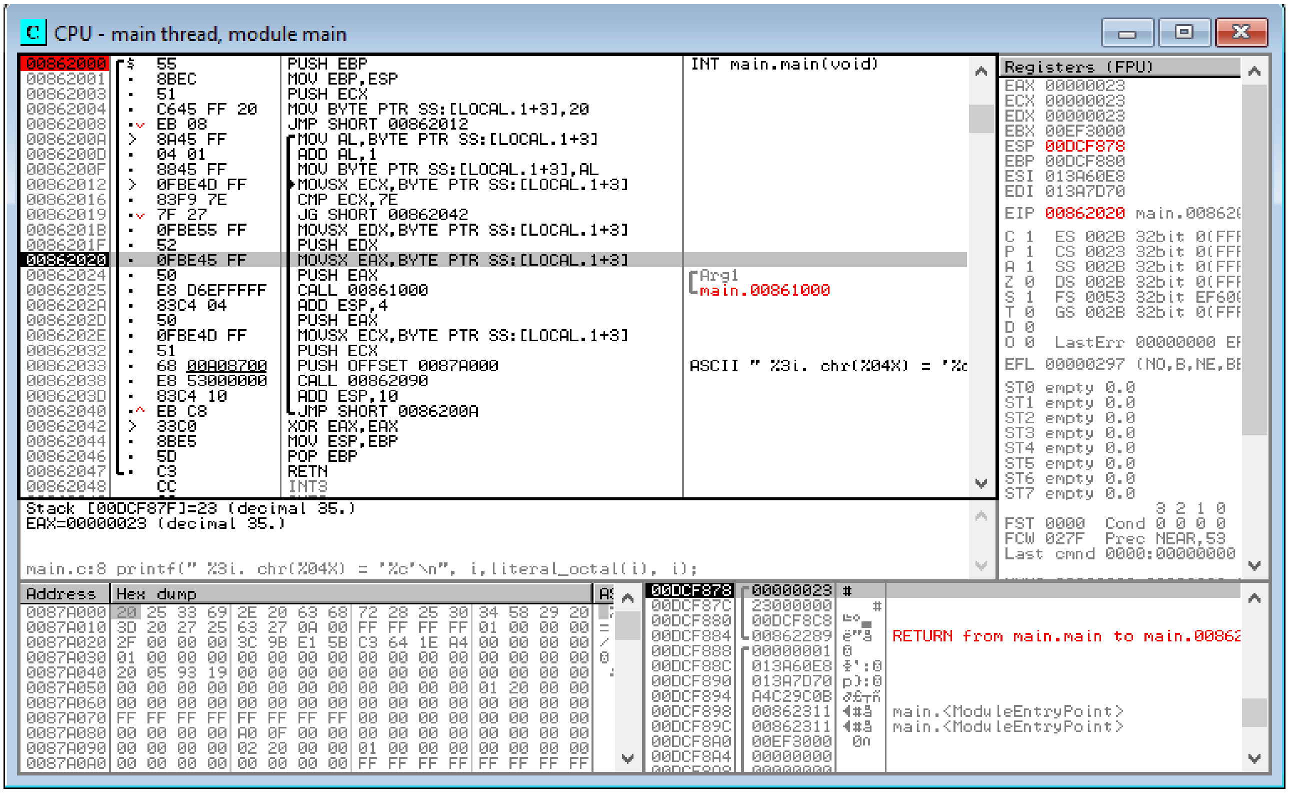
b8.asm (modul)

Code Segment

```asm
lbits 32
segment code use32 public code
global _literal_octal

_literal_octal:
    ; EAX = int(str(dec_to_oct(byte([EBP + 8]))))
    ; ex. [EBP+8]=9 => EAX = 11
    ; b7b6b5b4b3b2b1b0 => oooob7b6_oob5b4b3_oob2b1b0, oo:=0b
    ; ex. 162 = A2h = 1010_0010b => 0010_0100_0010b = 242h
    ; (<=> 242 oct)
    ; printf("%X",literal_octal(val));

    push   EBP         ; prepare stack
    mov    EBP, ESP

    push EBX           ; push only registers used by function
    push ECX

    xor EAX, EAX       ; EAX = 0
    mov AL, [EBP+8]    ; AL = byte([EBP+8])

    mov CL, 3          ; solve nibble 0
    call .insert_bit_0 ; EAX = b7b6_b5b4b3_oob2b1b0

    mov CL, 7          ; solve nibble 1
    call .insert_bit_0 ; EAX = b7b6_oob5b4b3_oob2b1b0

    pop ECX            ; retrieve registers from stack
    pop EBX

    mov   ESP, EBP     ; reset stack
    pop   EBP
    ret

    .insert_bit_0:
        ; CL < 31
        ; LEFT = bits EAX[31..CL+1]
        ; RIGHT = bits EAX[CL..0]
        ; EAX = {LEFT,RIGHT} := ((EAX xor RIGHT) << 1) | RIGHT
        mov EBX, 1   ; EBX = 1
        shl EBX, CL  ; EBX = 100...0 (cnt(0)=CL)
        dec EBX      ; EBX =  11...1 (RIGHT mask)
        and EBX, EAX ; EBX = RIGHT
        xor EAX, EBX ; EAX = {LEFT,00..0}
        add EAX, EAX ; EAX <<= 1, avoid CL
        or  EAX, EBX ; EAX = LEFT_0_RIGHT
        ret
```



```
C:\Users\Stefan\Desktop\ASC\asm_tools\lab12>main.exe
 32. chr(0040) = ' '
 33. chr(0041) = '!'
 34. chr(0042) = '"'
 35. chr(0043) = '#'
 36. chr(0044) = '$'
 37. chr(0045) = '%'
 38. chr(0046) = '&'
 39. chr(0047) = '''
 40. chr(0050) = '('
 41. chr(0051) = ')'
 42. chr(0052) = '*'
 43. chr(0053) = '+'
 44. chr(0054) = ','
 45. chr(0055) = '-'
 46. chr(0056) = '.'
 47. chr(0057) = '/'
 48. chr(0060) = '0'
 49. chr(0061) = '1'
 50. chr(0062) = '2'
 51. chr(0063) = '3'
 52. chr(0064) = '4'
 53. chr(0065) = '5'
 54. chr(0066) = '6'
 55. chr(0067) = '7'
 56. chr(0070) = '8'
 57. chr(0071) = '9'
 58. chr(0072) = ':'
 59. chr(0073) = ';'
```



```
00862000  $ 55             PUSH EBP                              INT main.main(void)
00862001  . 8BEC           MOV EBP,ESP
00862003  . 51             PUSH ECX
00862004  . C645 FF 20     MOV BYTE PTR SS:[LOCAL.1+3],20
00862008  .v EB 08         JMP SHORT 00862012
0086200A  > 8A45 FF        MOV AL,BYTE PTR SS:[LOCAL.1+3]
0086200D  . 04 01          ADD AL,1
0086200F  . 8845 FF        MOV BYTE PTR SS:[LOCAL.1+3],AL
00862012  > 0FBE4D FF      MOVSX ECX,BYTE PTR SS:[LOCAL.1+3]
00862016  . 83F9 7E        CMP ECX,7E
00862019  .v 7F 27         JG SHORT 00862042
0086201B  . 0FBE55 FF      MOVSX EDX,BYTE PTR SS:[LOCAL.1+3]
0086201F  . 52             PUSH EDX
00862020  . 0FBE45 FF      MOVSX EAX,BYTE PTR SS:[LOCAL.1+3]
00862024  . 50             PUSH EAX                              [Arg1
00862025  . E8 D6EFFFFF    CALL 00861000                         main.00861000
0086202A  . 83C4 04        ADD ESP,4
0086202D  . 50             PUSH EAX
0086202E  . 0FBE4D FF      MOVSX ECX,BYTE PTR SS:[LOCAL.1+3]
00862032  . 51             PUSH ECX
00862033  . 68 00A08700    PUSH OFFSET 0087A000                  ASCII " %3i. chr(%04X) = '%c
00862038  . E8 53000000    CALL 00862090
0086203D  . 83C4 10        ADD ESP,10
00862040  .^ EB C8         JMP SHORT 0086200A
00862042  > 33C0           XOR EAX,EAX
00862044  . 8BE5           MOV ESP,EBP
00862046  . 5D             POP EBP
00862047  . C3             RETN
00862048    CC             INT3
Stack [00DCF87F]=23 (decimal 35.)
EAX=00000023 (decimal 35.)

main.c:8 printf(" %3i. chr(%04X) = '%c'\n", i,literal_octal(i), i);
```

```
Registers (FPU)
EAX 00000023
ECX 00000023
EDX 00000023
EBX 00EF3000
ESP 00DCF878
EBP 00DCF880
ESI 013A60E8
EDI 013A7D70

EIP 00862020 main.00862

C 1  ES 002B 32bit 0(FFF
P 1  CS 0023 32bit 0(FFF
A 1  SS 002B 32bit 0(FFF
Z 0  DS 002B 32bit 0(FFF
S 1  FS 0053 32bit EF60
T 0  GS 002B 32bit 0(FFF
D 0
O 0  LastErr 00000000 EF

EFL 00000297 (NO,B,NE,BE

ST0 empty 0.0
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 0.0
ST7 empty 0.0
               3 2 1 0
FST 0000  Cond 0 0 0 0
FCW 027F  Prec NEAR,53
Last cmnd 0000:00000000
```

```
Address   Hex dump
0087A000  20 25 33 69 2E 20 63 68 72 28 25 30 34 58 29 20
0087A010  3D 20 27 25 63 27 0A 00 FF FF FF FF 01 00 00 00
0087A020  2F 00 00 00 3C 9B E1 5B C3 64 1E A4 00 00 00 00
0087A030  01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0087A040  20 05 93 19 00 00 00 00 00 00 00 00 00 00 00 00
0087A050  00 00 00 00 00 00 00 00 00 00 00 00 01 20 00 00
0087A060  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0087A070  FF FF FF FF FF FF FF FF 00 00 00 00 00 00 00 00
0087A080  00 00 00 00 A0 0F 00 00 00 00 00 00 00 00 00 00
0087A090  00 00 00 00 02 20 00 00 01 00 00 00 00 00 00 00
0087A0A0  00 00 00 00 00 00 00 00 FF FF FF FF FF FF FF FF
```

```
00DCF878  00000023  #
00DCF87C  23000000     #
00DCF880  00DCF8C8
00DCF884  00862289        RETURN from main.main to main.00862
00DCF888  00000001  0
00DCF88C  013A60E8
00DCF890  013A7D70  p}:0
00DCF894  A4C29C0B
00DCF898  00862311        main.<ModuleEntryPoint>
00DCF89C  00862311        main.<ModuleEntryPoint>
00DCF8A0  00EF3000   0n
00DCF8A4  00000000
```