Computer Architecture and Assembly Language

Practical Session 8

Position Independent Code - PIC

- **PIC has everything it needs internally**
- PIC can be placed somewhere in memory, **is executed properly regardless of its absolute address**
- PIC **can be added to any other program**, without fear that it might not work

Position Independent Code - requirements

No direct usage of labels

- No library functions ? ☹ → only system calls
- One section only
- ‘jump’ and ‘call’ functions ? ☺ → they are relative

No library functions

→ only system calls

We don’t know if and where the library functions are. Thus **there are no “printf” “gets” and so on**.

To perform I/O operation we have to **use the Linux system calls** because INT 0x80 isn’t a regular procedure - it is called via the **interrupt table** which **is static**.

One section only

We put all the code in a **single section** – .text (read-only) or .data (read-write).

Both .text and .data sections **may contain any valid assembly instruction**.

Usage of a single section gives us a possibility to **calculate a relative offset between a pair of code instruction addresses**, and thus **use offset instead of absolute address.**

Only relative jumps

```
 1
 2                                  section .data
 3 00000000 78563412                        numeric:    DD 0x12345678
 4 00000004 616263                          string:         DB 'abc'
 5 00000007 00000000                        answer:         DD      0
 6
 7                                  section .text
 8                                          global  _start
 9
10                                  _start:
11
12 00000000 60                             pushad
13 00000001 6A02                           push dword 2
14 00000003 6A01                           push dword 1
15 00000005 E815000000                     CALL myFunc
16                                  returnAddress:
17 0000000A A3[07000000]                   mov [answer], eax
18 0000000F 83C408                         add esp, 8
19 00000012 61                             popad
20 00000013 BB00000000                     mov ebx,0
21 00000018 B801000000                     mov eax,1
22 0000001D CD80                           int 0x80
23
24                                  myFunc:
25 0000001F 55                             push    ebp
26 00000020 89E5                           mov     ebp, esp
```

If all the code changes its position, **relative jumps are still valid**, because the address difference is preserved.

→ Address of myFunc label = 0x1F
→ Address of the next instruction after the call (i.e. 'mov [answer], eax') is 0xA
→ 0x1F-0xA=0x15, and we get exactly the binary code written here 'E815000000'

No direct usage of labels

```
 1
 2
 3                                        section .rodata
 4  00000000 48656C6C6F0A00                      name:   db "Hello",10,0
 5
 6                                               global main
 7                                               extern  printf
 8
 9                                        section .text
10                                        main:
11  00000000 68[00000000]                        push    name
12  00000005 E8(00000000)                        call    printf
13  0000000A 83EC04                              sub     esp, 4
14  0000000D B801000000                          mov     eax, 1
15  00000012 CD80                                int     80h
16
```

Labels are resolved at compile time to **absolute address**

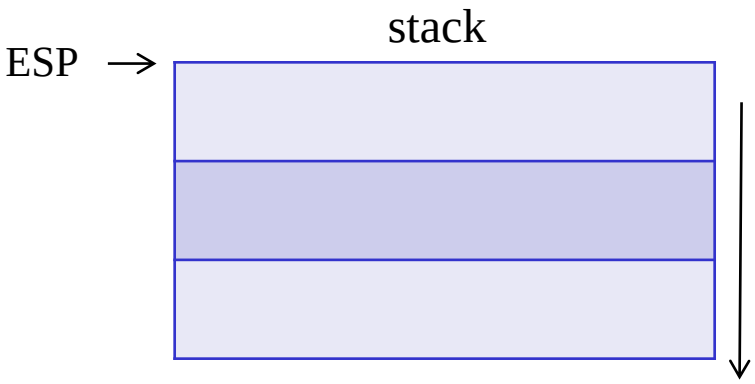
An absolute addresses of name would be resolved based on the offset of name in .rodata section. If the code is moved, the absolute address that was calculated before the moving **would not be correct any more**

Using labels – PIC example

```
section .text

name:       db          "Hello",10,0
nameLen     equ         $ - name

            global _start
get_my_loc:
            call        next_i
next_i:
            pop         edx
            ret
_start:
            call        get_my_loc
            sub         edx,  next_i – name
            mov         ecx, edx
            mov         edx, nameLen
            mov         eax, 4
            mov         ebx, 1
            int         80h
            mov         eax, 1
            int         80h
```

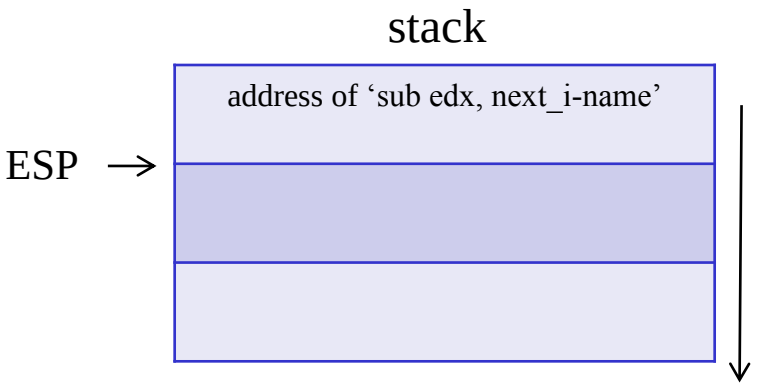


Using labels – PIC example

```
section .text

name:       db          "Hello",10,0
nameLen     equ         $ - name

            global _start
get_my_loc:
            call        next_i
next_i:
            pop         edx
            ret
_start:
            call        get_my_loc
            sub         edx,  next_i – name
            mov         ecx, edx
            mov         edx, nameLen
            mov         eax, 4
            mov         ebx, 1
            int         80h
            mov         eax, 1
            int         80h
```

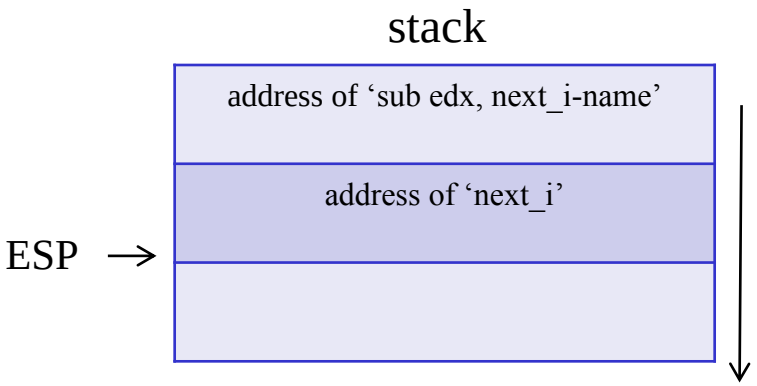


Using labels – PIC example

```
section .text

name:       db          "Hello",10,0
nameLen     equ         $ - name

            global _start
get_my_loc:
            call        next_i
next_i:
            pop         edx
            ret
_start:
            call        get_my_loc
            sub         edx,  next_i – name
            mov         ecx, edx
            mov         edx, nameLen
            mov         eax, 4
            mov         ebx, 1
            int         80h
            mov         eax, 1
            int         80h
```

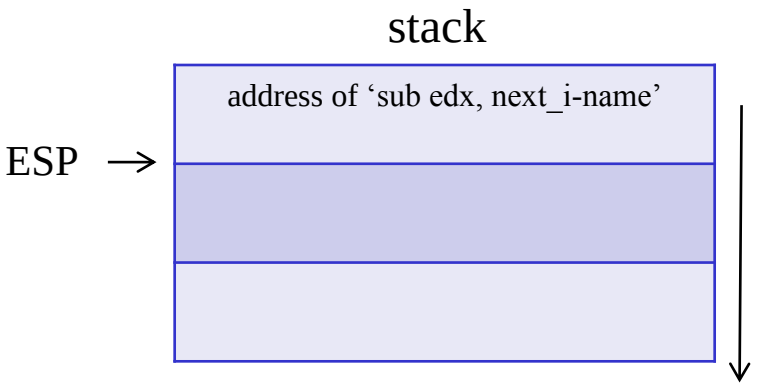


Using labels – PIC example

```
section .text

name:       db          "Hello",10,0
nameLen     equ         $ - name

            global _start
get_my_loc:
            call        next_i
next_i:
            pop         edx         ; edx gets address of 'next_i'
            ret
_start:
            call        get_my_loc
            sub         edx,  next_i – name
            mov         ecx, edx
            mov         edx, nameLen
            mov         eax, 4
            mov         ebx, 1
            int         80h
            mov         eax, 1
            int         80h
```

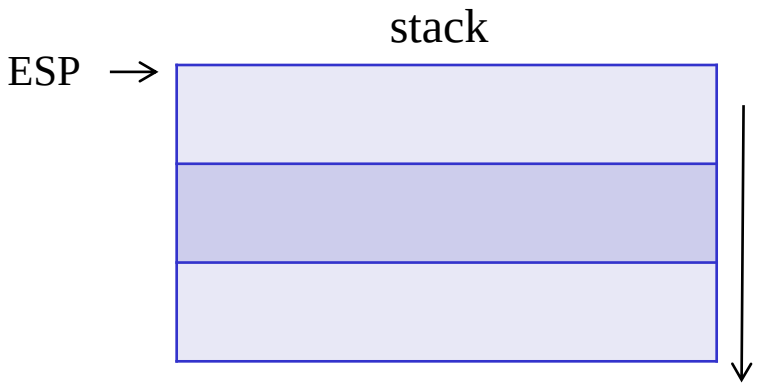


Using labels – PIC example

```
section .text

name:       db          "Hello",10,0
nameLen     equ         $ - name

            global _start
get_my_loc:
            call        next_i
next_i:
            pop         edx
            ret                     ; EIP gets address of 'sub edx, next_i-name'
_start:
            call        get_my_loc
            sub         edx,  next_i – name
            mov         ecx, edx
            mov         edx, nameLen
            mov         eax, 4
            mov         ebx, 1
            int         80h
            mov         eax, 1
            int         80h
```

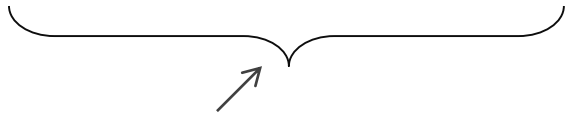


Using labels – PIC example

```
section .text

name:       db          "Hello",10,0
nameLen     equ         $ - name

            global _start
get_my_loc:
            call        next_i
next_i:
            pop         edx
            ret
_start:
            call        get_my_loc
            sub         edx,  next_i – name   ; edx = 'next_i' address – ('next_i' address – 'name' address)
            mov         ecx, edx
            mov         edx, nameLen
            mov         eax, 4
            mov         ebx, 1
            int         80h
            mov         eax, 1
            int         80h
```



the address difference between "next_i" and "name" is constant even if the code changes it's position

Using labels – PIC example

```
section .text

name:       db          "Hello",10,0
nameLen     equ         $ - name

            global _start
get_my_loc:
            call        next_i
next_i:
            pop         edx
            ret
_start:
            call        get_my_loc
            sub         edx,  next_i – name
            mov         ecx, edx
            mov         edx, nameLen
            mov         eax, 4
            mov         ebx, 1
            int         80h
            mov         eax, 1
            int         80h
```

←

why we may use ‘nameLen’ label directly ?

Using labels – PIC example

>nasm -f elf sample.s -l sample.lst

```
 1                                  section .text
 2
 3 00000000 48656C6C6F0A00          name:   db      "Hello",10,0
 4                                  nameLen         equ     $ - name
 5
 6                                                  global _start
 7                                  get_my_loc:
 8 00000007 E800000000                              call    next_i
 9                                  next_i:
10 0000000C 5A                                     pop     edx
11 0000000D C3                                     ret
12                                 _start:
13 0000000E E8F4FFFFFF                             call    get_my_loc
14 00000013 83EA0C                 <-- 0x0C = 'next_i' – 'name'
                                                   sub     edx, next_i - name
15 00000016 89D1                                   mov     ecx, edx
16 00000018 BA07000000                             mov     edx, nameLen
17 0000001D B804000000                             mov     eax, 4
18 00000022 BB01000000                             mov     ebx, 1
19 00000027 CD80                                   int     80h
20 00000029 B801000000                             mov     eax, 1
21 0000002E CD80                                   int     80h
```