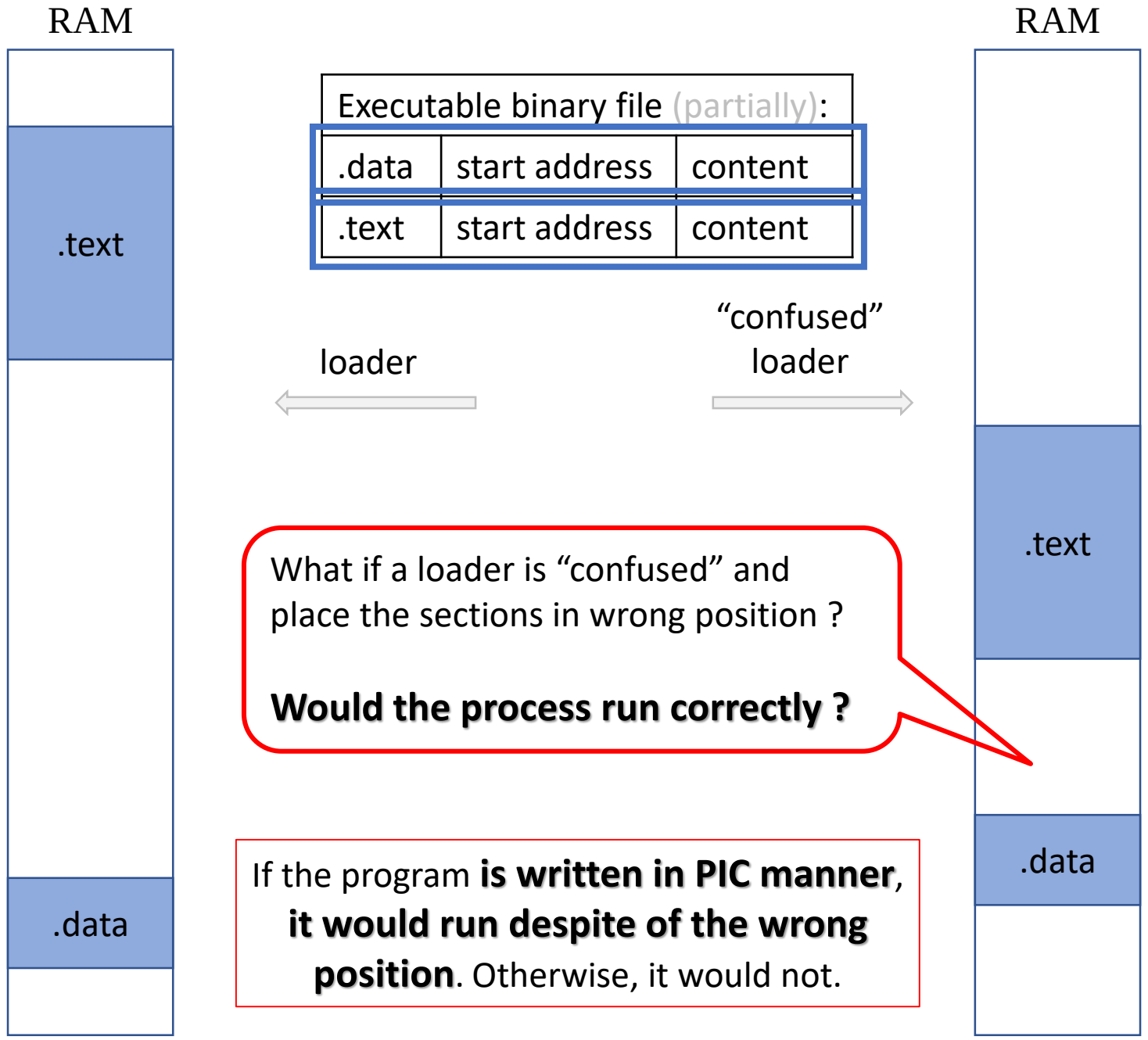
Computer Architecture and Assembly Language

Practical Session 9

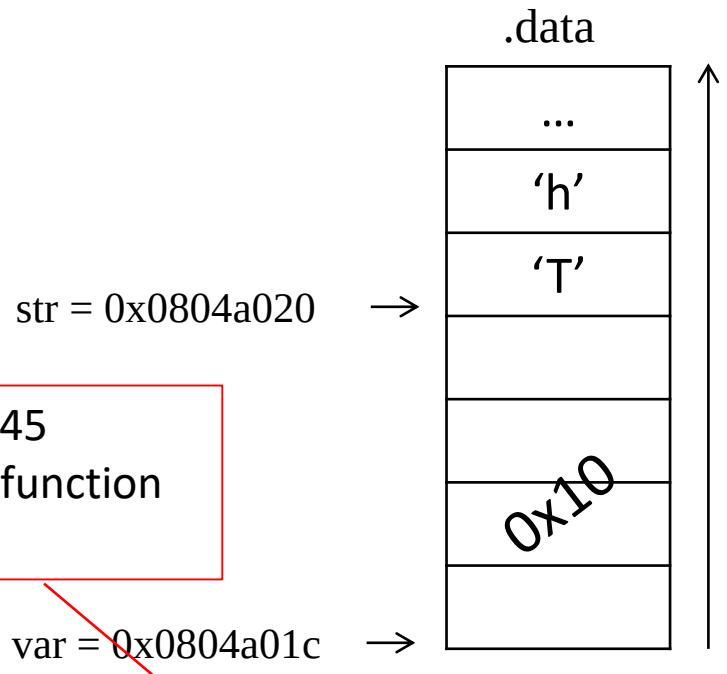
Position Independent Code (PIC)
can be executed at
any memory address
without modification

```
section .data
        var: dd 0x10
        str: db "The result is %d",10,0

        extern printf
        global main
section .text
main:
        call func
        push dword [var]
        push str
        call printf
        add esp, 8

        mov eax,1
        int 0x80


func:
        push ebp
        mov ebp, esp
        mov ecx, 3
        calcLoop:
                add dword [var],2
                loop calcLoop
        pop ebp
        ret
```



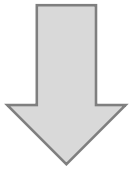
```
 1                                          section .data
 2 00000000 10000000                var:     dd 0x10
 3 00000004 5468652072657573756C-   str:     db "The result is %d",10,0
 4 0000000D 7420697320256400
 5
 6                                          extern printf
 7                                          global main
 8                                          section .text
 9                                  main:
10 00000000 E81A000000                     call func
11 00000005 FF35[00000000]                 push dword [var]
12 0000000B 68[04000000]                   push str
13 00000010 E8(00000000)                   call printf
14 00000015 83C408                         add esp, 8
15
16 00000018 B801000000                     mov eax,1
17 0000001D CD80                           int 0x80
18
19
20                                  func:
21 0000001F 55                             push ebp
22 00000020 89E5                           mov ebp, esp
23 00000022 B903000000                     mov ecx, 3
24                                  calcLoop:
25 00000027 8305[00000000]02               add dword [var],2
26 0000002E E2F7                           loop calcLoop
27 00000030 5D                             pop ebp
28 00000031 C3                             ret
29
```

0xfffffebb=-0x145
Offset of printf function
in section .text




```
Breakpoint 1, 0x08048410 in main ()
(gdb) x /1xg $eip
0x8048410 <main>:        0x1c35ff0000001ae8
(gdb)
0x8048418 <main+8>:      0x0804a02068 0804a0
(gdb)
0x8048420 <main+16>:     0x08c483ffffffebbe8
(gdb)
0x8048428 <main+24>:     0x5580cd00000001b8
(gdb)
0x8048430 <func+1>:      0x8300000003b9e589
(gdb)
0x8048438 <calcLoop+1>:  0xf7e2020804a01c05
(gdb)
0x8048440 <calcLoop+9>:  0x906690669066c35d
```

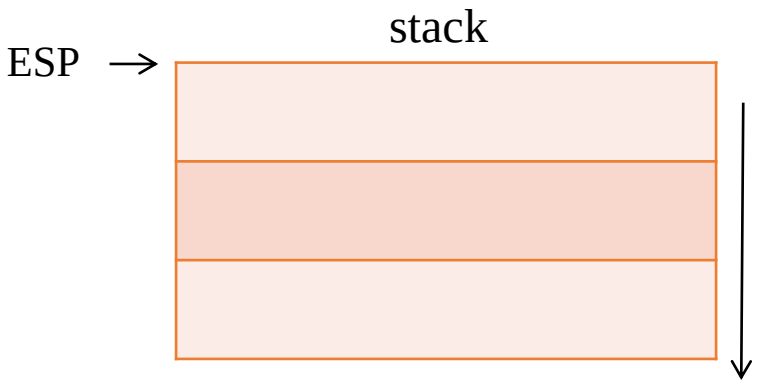
No direct usage of labels



- No library functions → only system calls ☹
  (their code is not necessarily PIC and thus not safe to use)
- Indirect usage of labels → jump, call, loop commands are PIC ☺
  (note that there are several types of jump and call commands)
- Variables address detection in run time → singe section code

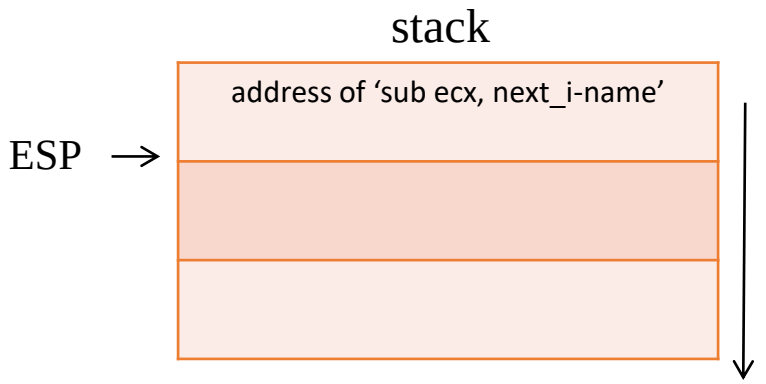
```
section .data
        var: dd 0x10

section .text

        global main
main:
        add dword [var],2

        mov eax,1
        int 0x80
```

```
section .data
        var: dd 0x10

get_my_loc:
        call next_i
next_i:
        pop ecx
        ret

        global main
main:
        call get_my_loc
        sub ecx,  next_i – var
        add dword [ecx],2

        mov eax,1
        int 0x80
```



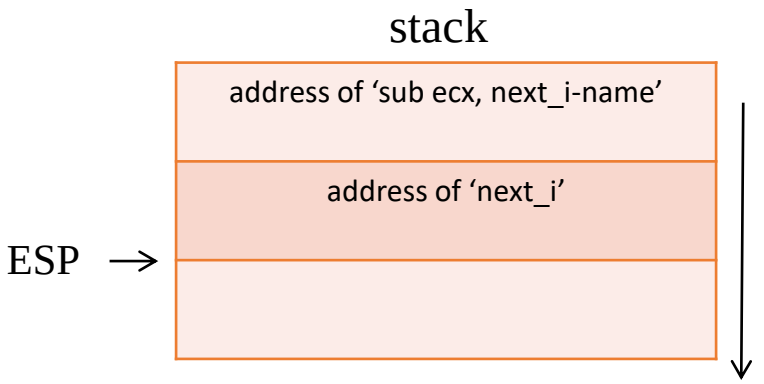
```
section .data
        var: dd 0x10

section .text

        global main
main:
        add dword [var],2

        mov eax,1
        int 0x80
```

```
section .data
        var: dd 0x10

get_my_loc:
        call next_i
next_i:
        pop ecx
        ret

        global main
main:
        call get_my_loc
        sub ecx,  next_i – var
        add dword [ecx],2

        mov eax,1
        int 0x80
```



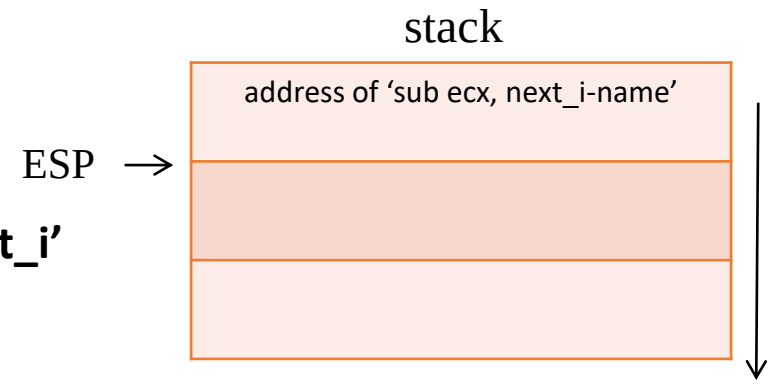
```
section .data
        var: dd 0x10

section .text

        global main
main:
        add dword [var],2

        mov eax,1
        int 0x80
```

```
section .data
        var: dd 0x10

get_my_loc:
        call next_i
next_i:
        pop ecx
        ret

        global main
main:
        call get_my_loc
        sub ecx,  next_i – var
        add dword [ecx],2

        mov eax,1
        int 0x80
```



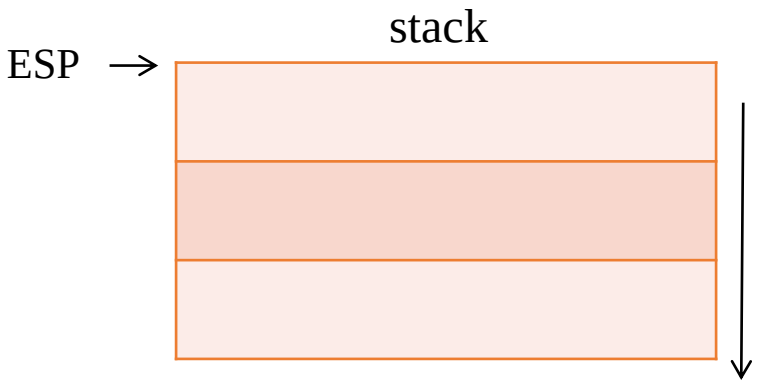
```
section .data
        var: dd 0x10

section .text

        global main
main:
        add dword [var],2

        mov eax,1
        int 0x80
```

```
section .data
        var: dd 0x10

get_my_loc:
        call next_i
next_i:
        pop ecx        ; ecx gets address of 'next_i'
        ret

        global main
main:
        call get_my_loc
        sub ecx,  next_i – var
        add dword [ecx],2

        mov eax,1
        int 0x80
```



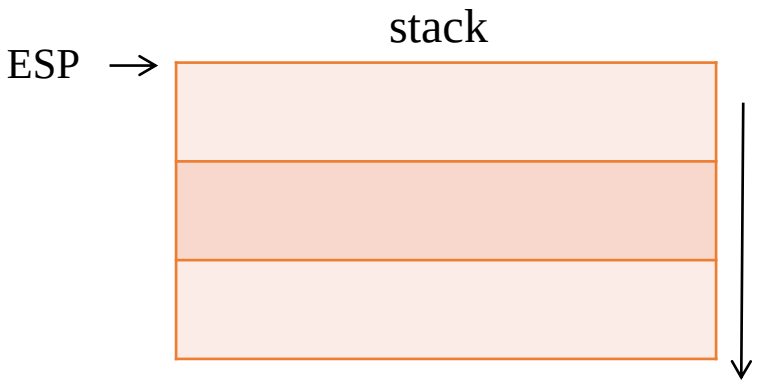
```
section .data
        var: dd 0x10

section .text

        global main
main:
        add dword [var],2

        mov eax,1
        int 0x80
```

```
section .data
        var: dd 0x10

get_my_loc:
        call next_i
next_i:
        pop ecx
        ret

        global main
main:
        call get_my_loc
        sub ecx,  next_i – var
        add dword [ecx],2

        mov eax,1
        int 0x80
```



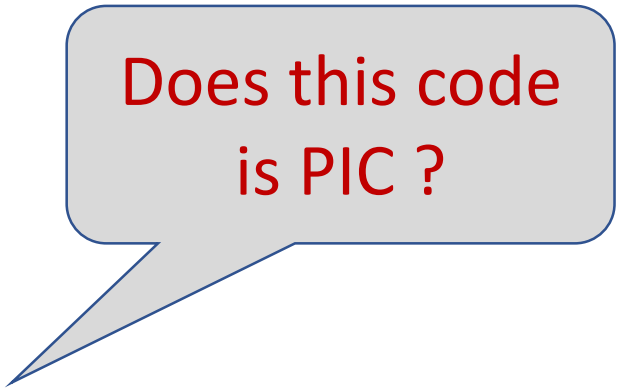
```
section .data
        var: dd 0x10

section .text

        global main
main:
        add dword [var],2

        mov eax,1
        int 0x80
```

```
section .data
        var: dd 0x10

get_my_loc:
        call next_i
next_i:
        pop ecx
        ret

        global main
main:
        call get_my_loc
        sub ecx,  next_i – var      ; ecx = 'next_i' address – ('next_i' address – 'var' address)
        add dword [ecx],2

        mov eax,1
        int 0x80
```

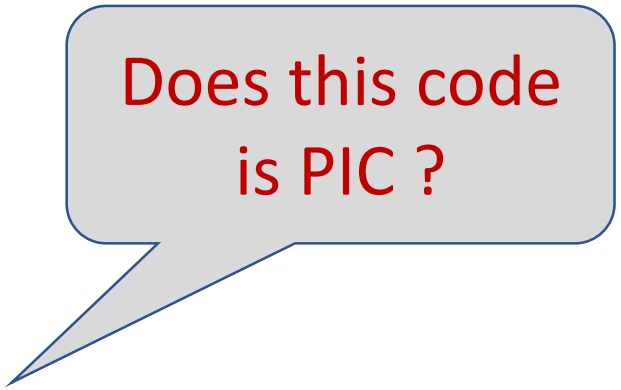


the address difference between "next_i" and "var" is constant even if the code changes it's position

```
section .text

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

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



```
section .text

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

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



may we use ‘nameLen’ label directly ?