زبان ماشین و اسمبلی

(۱۳-۱۱-۰۰۵)

دستورهای انشعاب

فراخوانی توابع C

دانشگاه شهید بهشتی
دانشکده‌ی مهندسی برق و کامپیوتر
بهار ۱۳۹۴
احمد محمودی ازناوه

http://faculties.sbu.ac.ir/~a_mahmoudi/

فهرست مطالب

- ساختارهای کنترلی
  - انشعاب غیرشرطی
  - فراخوانی تابع
- استفاده از توابع C

ساختارهای کنترلی

Memory

Address	Memory
	C3
0x0804807E	89
	00
	00
	00
	0A
0x08048079	BF
	00
	00
	00
	00
0x08048074	B8

Instruction Codes

B8 00 00 00 00
BF 0A 00 00 00
89 C3

Assembly Code

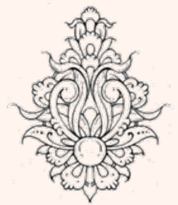
movl $0, %eax
movl $10, %edi
movl %eax, %ebx

next instruction

EIP

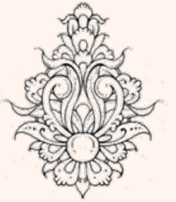
0x08048079

دستورات انشعاب مقدار موجود در EIP را تغییر می‌دهند



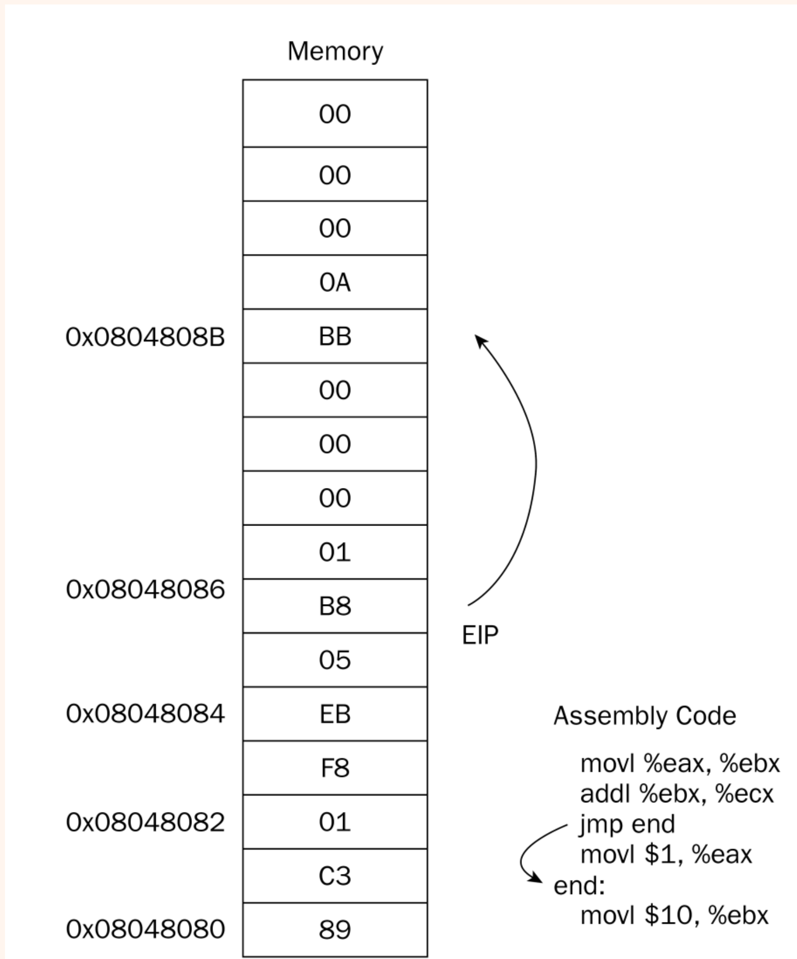
دستورات انشعاب

- دستورات انشعاب غیرشرطی
  - پرش (Jump)
  - فراخوانی تابع (calls)
  - فراخوانی وقفه (interrupts)
- دستورات انشعاب شرطی



پرش (jumps)

`jmp location`

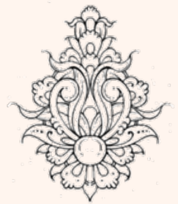


- معادل «GOTO» می‌باشد.
- در برنامه‌های اسمبلی استفاده از چنین دستوری نه تنها بد نیست بلکه در بسیاری موارد لازم است.
  - البته توجه داشته باشید که کارایی برنامه را کاهش می‌دهد.

`jmp address`

EIP ← address

در این عبارت جزییات آدرس‌دهی مشخص نشده است



پرش (jumps)

```
.section .text
.globl _start
_start:
    nop
    movl $1, %eax
    jmp overhere
    movl $10, %ebx
    int $0x80
overhere:
    movl $20, %ebx
    int $0x80
```

با استفاده از echo $? می‌توان وضعیت خروجی آخرین دستور را چک کرد

```
ahmad@ubuntu:~/Courses/Assembly/Sample$ ./jumptest
ahmad@ubuntu:~/Courses/Assembly/Sample$ echo $?
20
```

```
ahmad@ubuntu:~/Assembly/code/chap06$ objdump -D jumptest

jumptest:     file format elf32-i386


Disassembly of section .text:

08048054 <_start>:
 8048054:       90                      nop
 8048055:       b8 01 00 00 00          mov    $0x1,%eax
 804805a:       eb 07                   jmp    8048063 <overhere>
 804805c:       bb 0a 00 00 00          mov    $0xa,%ebx
 8048061:       cd 80                   int    $0x80

08048063 <overhere>:
 8048063:       bb 14 00 00 00          mov    $0x14,%ebx
 8048068:       cd 80                   int    $0x80
ahmad@ubuntu:~/Assembly/code/chap06$
```

مثال

```
ahmad@ubuntu:~/Courses/Assembly/Sample$ objdump -d example1_1.o
```

```
# example 1_1
.section .text
.globl _start
_start:
    movl $4, %eax
    movl $1, %ebx
    movl $output, %ecx
    movl $14, %edx
    int $0x80
    jmp label
output:
    .ascii "hello, world!\n"
label:
    movl $1, %eax
    movl $0, %ebx
    int $0x80
```

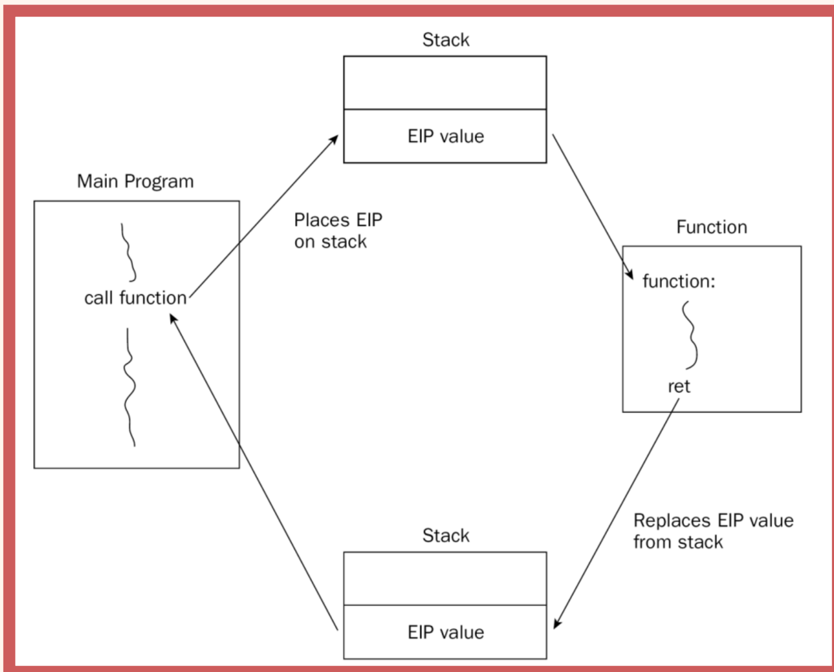
```
ahmad@ubuntu:~/MyData/courses/Asm/92_1/chapter2$ objdump -D example1_1

example1_1:     file format elf32-i386


Disassembly of section .text:

08048054 <_start>:
 8048054:       b8 04 00 00 00          mov    $0x4,%eax
 8048059:       bb 01 00 00 00          mov    $0x1,%ebx
 804805e:       b9 6c 80 04 08          mov    $0x804806c,%ecx
 8048063:       ba 0e 00 00 00          mov    $0xe,%edx
 8048068:       cd 80                   int    $0x80
 804806a:       eb 0e                   jmp    804807a <label>

0804806c <output>:
 804806c:       68 65 6c 6c 6f          push   $0x6f6c6c65
 8048071:       2c 20                   sub    $0x20,%al
 8048073:       77 6f                   ja     80480e4 <label+0x6a>
 8048075:       72 6c                   jb     80480e3 <label+0x69>
 8048077:       64 21 0a                and    %ecx,%fs:(%edx)
```

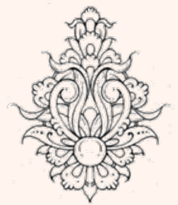
```
ahmad@ubuntu:~/Courses/Assembly/Sample$ ./example1_1
hello, world!
```

فراخوانی تابع (Calls)

```
call address
```

- شبیه دستور jump است؛ با این تفاوت که جایی که از آن جا آمده است را به خاطر می‌سپرد تا در صورت لزوم بتواند بازگردد.
- با کمک تابع می‌توان برنامه را به زیربخش‌های مناسب تقسیم نمود.
- در صورتی که از یک قطعه کد در دو قسمت از برنامه استفاده شود، نیازی به تکرار دوباره‌ی آن نیست.





فراخوانی تابع (ادامه...)

- برای فراخوانی تابع از دستور زیر استفاده می‌شود:

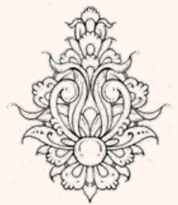
```
call address
```

- با فراخوانی دستور call ابتدا محتوای EIP به پشته منتقل شده و سپس این ثبات با آدرس تابع مورد نظر مقداردهی می‌شود. در پایان، دستور ret آدرس بازگشت را از پشته بر می‌دارد.

```
call address
```

```
ESP ← ESP-4
Mem[ESP] ← EIP
EIP ← Address
```

در حین اجرای دستور **call** ثبات **EIP** به دستور بعدی اشاره می‌کند



فراخوانی تابع (ادامه...)

- در بخش دوم، بازگشت به محل فراخوانی (آدرس دستوری که بعد از call قرار دارد) است:

```
ret
```

- این دستور هیچ عملوندی ندارد. آدرس بازگشت را از روی پشته بر می‌دارد.

```
ret
```

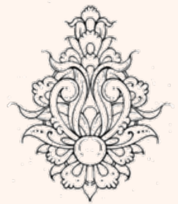
```
EIP ← Mem[ESP]
ESP ← ESP+4
```

فراخوانی تابع(ادامه...)

- به طور ساده می‌توان گفت یک تابع چنین ساختاری دارد:

```
function_label:
        <normal function code goes here>
        ret
```

برای فرستادن پارامترها به تابع چه راهی پیشنهاد می‌کنید؟



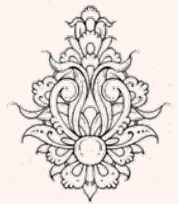
مثال

- تابعی بنویسید که عددی را دریافت و آن یک واحد افزایش دهد.

```
# first call
.section .text
.globl _start
_start:
    movl $0, %eax
    call increament

    movl $1, %eax
    movl $0, %ebx
    int $0x80

increament:
    addl $1, %eax
    ret
```

ورودی و خروجی هر دو در eax

پارامتر ارسالی

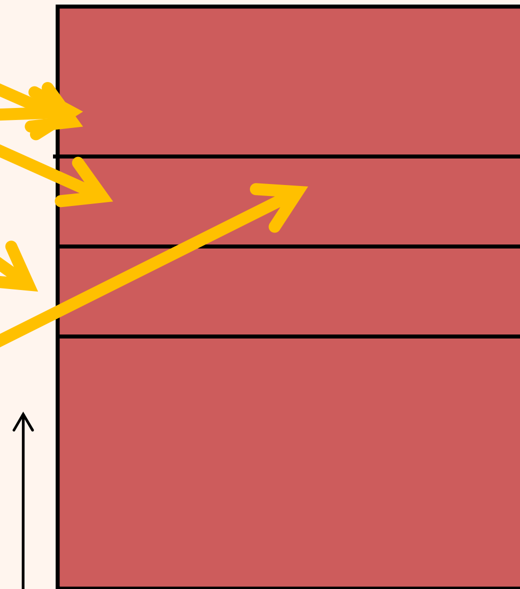


مثال(ادامه...)

ورودی از طریق پشته

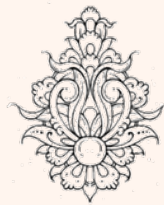
خروجی در eax

```
# second call
.section .text
.globl _start
_start:
    movl $0, %eax
    pushl %eax
    call increament
    addl $4,%esp

    movl $1, %eax
    movl $0, %ebx
    int $0x80

increament:
    movl 4(%esp),%eax
    addl $1, %eax
    ret
```



استفاده از توابع C در زبان اسمبلی

- برای نمایش خروجی برنامه از توابع سیستمی لینوکس استفاده کردیم.
- برای این کار می‌توان از توابع کتابخانه‌ای C نیز استفاده نمود.
- در ادامه خواهیم دید، چگونه می‌توان از توابع کتابخانه‌ای زبان C استفاده کرد.
- پارامترها در توابع C از طریق پشته منتقل می‌شوند، به ترتیب از راست به چپ باید روی پشته قرار بگیرند.



استفاده از توابع C در زبان اسمبلی

```
function(p1,p2,p3);
```

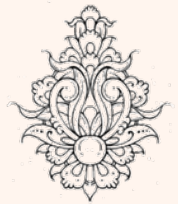
فرض: پارامترها ارسالی همگی چهاربایتی هستند

```
pushl p3
pushl p2
pushl p1
call function
addl $12,%esp
```

پارامترهای ارسالی

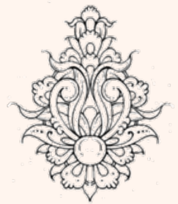
فراخوانی تابع

آزادسازی فضای پشته



استفاده از توابع C در زبان اسمبلی (ادامه...)

```
#cpuid.s Sample program to extract the processor Vendor ID
.section .data
output:
. ascii "The processor Vendor ID is 'xxxxxxxxxxxx'\n"

.section .text
.globl _start
_start:
  movl $0, %eax
  cpuid
  movl $output, %edi
  movl %ebx, 28(%edi)
  movl %edx, 32(%edi)
  movl %ecx, 36(%edi)
movl $4, %eax
movl $1, %ebx
movl $output, %ecx
movl $42, %edx
int $0x80
movl $1, %eax
movl $0, %ebx
int $0x80
```



استفاده از توابع C در زبان اسمبلی (ادامه...)

```
#cpuid.s Sample program to extract the processor Vendor ID
.section .data
output:
. asciz "The processor Vendor ID is '%s'\n"

.section .text
.globl _start
_start:
  movl $0, %eax
  cpuid
  movl $output, %edi
  movl %ebx, 28(%edi)
  movl %edx, 32(%edi)
  movl %ecx, 36(%edi)
pushl $buffer
pushl $output
call printf
addl $8, %esp
pushl $0
call exit
```

```
.section .bss
.lcomm buffer, 12
```

```
movl $buffer, %edi
```

استفاده از توابع C در زبان اسمبلی (ادامه...)

```
.section .data
output:
     .asciz "The processor Vendor ID is '%s'\n"
.section .bss
     .lcomm buffer, 12
.section .text
.globl _start
_start:
     movl $0, %eax
     cpuid
     movl $buffer, %edi
     movl %ebx, (%edi)
     movl %edx, 4(%edi)
     movl %ecx, 8(%edi)
     pushl $buffer
     pushl $output
     call printf
     addl $8, %esp
     pushl $0
     call exit
```

```
File  Edit  View  Terminal  Help
ahmad@ubuntu:~$ ld -o cpuid2 cpuid2.o
cpuid2.o: In function `_start':
(.text+0x1f): undefined reference to `printf'
cpuid2.o: In function `_start':
(.text+0x29): undefined reference to `exit'
ahmad@ubuntu:~$
```

در صورتی که مستقیما از gcc استفاده کنیم، چنین مشکلی پیش نمی آید

لینک کردن پویا

shared library file

• در سیستم‌عامل لینوکس کتابخانه‌های پویای زبان C در فایل libc.so.x قرار دارند. در این نام x بیانگر شماره‌ی ویرایش (version) کتابخانه است.

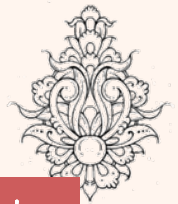
در سیستم من، این فایل /lib/libc.so.6 است.

• در هنگام لینک کردن با استفاده از پارامتر l– محل کتابخانه به لینکر اعلام می‌شود. بعد این پارامتر نام کتابخانه آورده می‌شود.

```
ahmad@ubuntu:~$ ld    -o cpuid2 -lc cpuid2.o
ahmad@ubuntu:~$
```

در این حالت لینکر فرض می‌کند که کتابخانه در فایل /lib/libx.so قرار دارد.

```
ahmad@ubuntu:~$ ld    -o cpuid2 -lc cpuid2.o
ahmad@ubuntu:~$ ./cpuid2
bash: ./cpuid2: No such file or directory
```



لینک کردن پویا

استفاده از توابع C در زبان اسمبلی (ادامه...)

- در حالت قبل لینکر، توانست فایل‌های مورد نظر خود را پیدا کند، اما باید به loader نحوه‌ی بارگذاری کتابخانه را اطلاع داد.
- از این رو باید به برنامه گفت که هنگام اجرا چگونه توابع مورد نیاز را بارگذاری کند. این کار با استفاده از پارامتر «-dynamic-linker» صورت می‌پذیرد. بدین ترتیب dynamic loader برای لینکر مشخص خواهد شد.

```
ahmad@ubuntu:~$ ld -dynamic-linker /lib/ld-linux.so.2  -o cpuid2 -lc cpuid2.o
ahmad@ubuntu:~$ ./cpuid2
The processor Vendor ID is 'GenuineIntel'
```