زبان ماشین و اسمبلی
(۱۳-۱۱-۰۰۵)

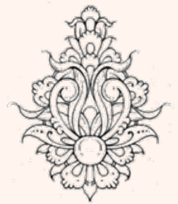
استفاده از زبان اسمبلی
در زبان‌های سطح بالا

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

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

فهرست مطالب

- نوشتن تابع در فایلی جداگانه
- استفاده کاربردی از زبان اسمبلی
  - فراخوانی تابع اسمبلی در C و C++
  - ایجاد کتابخانه به زبان اسمبلی
  - Inline asembly



نوشتن تابع در فایلی جداگانه

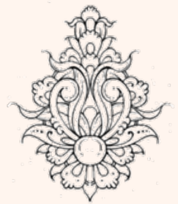
cube.s

```
.section .text
.type cube, @function
.globl cube
cube:
pushl %ebp
movl %esp, %ebp
movl 8(%ebp),%eax
imull 8(%ebp)
imull 8(%ebp)
movl %ebp, %esp
popl %ebp
ret
```

این راهنما نوع سمبل را مشخص می‌کند.

نام تابع باید به صورت سراسری اعلام شود تا سایر برنامه‌ها بتوانند به آن دسترسی داشته باشند.

```
as -o cube.o cube.s
```



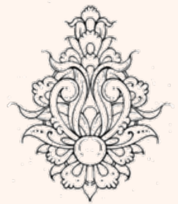
نوشتن تابع در فا

```
# exp12_1.s.section .data
num:
   .int 8
output:
   .asciz "cube(%d)=%d\n";
.section .text
.globl _start
_start:
    nop
    pushl num
    call cube
    addl $4,%esp
    pushl %eax
    pushl num
    pushl $output
    call printf
    addl $12,%esp
    movl $1, %eax
    movl $0, %ebx
    int $0x80
```

```
as -o exp12_1.o exp12_1.s -gstabs
```

```
ld -lc  exp12_1.o  cube.o -o exp12_1  -dynamic-linker /lib/ld-linux.so.2
```

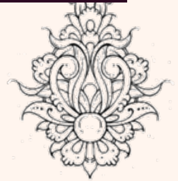
```
cube(8)=512
```



نوشتن تابع در فایلی جداگانه

```
.section .text
.type area, @function
.globl area
area:
pushl %ebp
movl %esp, %ebp
subl $4, %esp
fldpi
filds 8(%ebp)
fmul %st(0), %st(0)
fmulp %st(0), %st(1)
fstps -4(%ebp)
movl -4(%ebp), %eax
movl %ebp, %esp
popl %ebp
ret
```

```
as  functest4.s  -o functest4.o -g
as  area.s  -o area.o -g
```



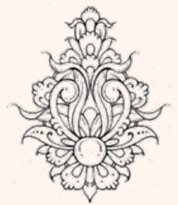
```
ld -dynamic-linker /lib/ld-linux.so.2 functest4.o area.o -lc -o functest4
```


این تابع مقدار خروجی را بر اساس استاندارد **cdecl** بر نمی‌گرداند

استفاده کاربردی(!) از زبان اسمبلی

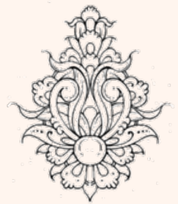
- تا اینجای کار با زبان اسمبلی آشنا شدیم، در ادامه با معلومات آموخته شده به صورت کاربردی‌تر برخورد خواهیم کرد.

- در مواردی می‌خواهیم از دستوراتی استفاده کنیم که کامپایلر قادر به استفاده از آن‌ها نیست.
  - برنامه را به زبان C نوشته و با استفاده از **S–** معادل اسمبلی آن را تولید کرده و دستورات مورد نظر را تغییر دهیم.
  - تابع اسمبلی را جداگانه نوشته و در برنامه‌ی C فراخوانی کنیم.
  - در تابع C به صورت مستقیم از زبان اسمبلی استفاده کنیم.



کتابخانه به زبان اسمبلی

- این بخش در مورد استفاده از توابع اسمبلی در زبان **C** (**C++**) است.
- با توجه به دشواری‌های که در نوشتن برنامه به زبان اسمبلی وجود دارد، یکی از شیوه‌های کاربردی تهیه‌ی کتابخانه‌ای است که توابع آن به صورت بهینه و به زبان اسمبلی نوشته شده باشد و در برنامه‌ی اصلی که به زبان سطح بالاست فراخوانی شود.
- به این مساله می‌توان نوشتن توابعی که امکان نوشتن آن‌ها در زبان سطح بالا وجود ندارد را نیز افزود.
- در این بخش، شیوه‌ی تهیه‌ی کتابخانه اسمبلی در linux به صورت مختصر بیان می‌شود.



فراخونی تابع

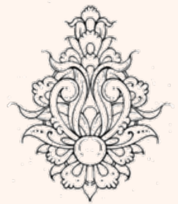
- طبیعتا این توابع باید به شیوه‌ی استاندارد مورد استفاده زبان سطح بالا نوشته شود، به عنوان مثال در زبان C به شیوه‌ی cdecl.
- در این شیوه هنگام شروع اجرای تابع، پشته به صورت زیر خواهد بود:

Program Stack

Program Stack	
	Indirect addressing
Function parameter 3	16(%ebp)
Function parameter 2	12(%ebp)
Function parameter 1	8(%ebp)
Return Address	4(%ebp)
Old EBP Value	(%ebp)

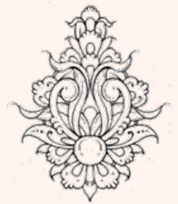
ESP → Old EBP Value

EBP → (%ebp)



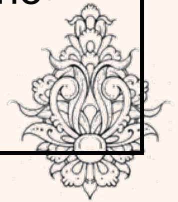
تغییر مقدار ثبات‌ها

- با توجه به این که تابع نوشته شده، توسط برنامه‌ی زبان سطح بالا فراخوانی می‌شود، باید تمام جزییات خواسته شده مورد توجه قرار گیرد.
- یکی از این مسائل، ثبات‌هایی هستند که مقدار آن‌ها پس از فراخوانی تابع باید حفظ شود.
- در جدول صفحه‌ی بعد این ثبات‌ها با رنگ قرمز مشخص شده‌اند.
  - تذکر: با برخی از ثبات‌ها یا کاربرد خاص آن‌ها در این درس آشنا نشده‌ایم، این مساله خللی در ادامه‌ی کار ایجاد نمی‌کند.



استفاده از ثبات‌ها

Register	Status
EAX	Used to hold the output value, but may be modified until the function returns
EBX	Used to point to the global offset table; must be preserved
ECX	Available for use within the function
EDX	Available for use within the function
EBP	Used as the base stack pointer by the C program; must be preserved
ESP	Used to point to the new stack location within the function; must be preserved
EDI	Used as a local register by the C program; must be preserved
ESI	Used as a local register by the C program; must be preserved
ST(0)	Used to hold a floating-point output value, but may be modified until the function returns
ST(1) - ST(7)	Available for use within the function



قالب کلی نوشتن توابع به زبان اسمبلی

```
.section .text
.type func, @function
.globl func
func:
pushl %ebp
movl %esp, %ebp
subl $12, %esp
pushl %edi
pushl %esi
pushl %ebx

<function code>

popl %ebx
popl %esi
popl %edi
movl %ebp, %esp
popl %ebp
ret
```

function prologue

در صورت نیاز به متغیر محلی

در صورت استفاده از این ثبات‌ها نوشتن این بخش لازم است.

function epilogue

لینک کردن

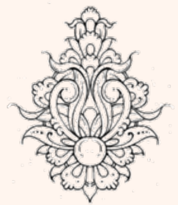
- برای کامپایل کردن با استفاده از gcc به راحتی می‌توان به صورت زیر عمل کرد:

```
gcc -o mainprog mainprog.c asmfunc1.s asmfunc2.s
```

- همچنین می‌توان از object file تابع استفاده کرد:

```
as -o asmfunc.o asmfunc.s
gcc -o mainprog mainprog.c asmfunc.o
```

- در هر دو حالت خروجی یک فایل اجرایی است که به صورت مستقل قابل اجرا خواهد بود.



مثال ۱

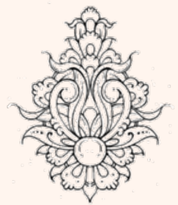
این تابع یک پیغام ثابت در خروجی چاپ می‌کند.
"This is a test message from the asm function\n"

```
.section .data
testdata:
.ascii "This is a test message from the asm function\n"
.section .text
.type asmfunc, @function
.globl asmfunc
asmfunc:
```

```
pushl %ebp
movl %esp, %ebp
pushl %ebx
```

```
movl  $4, %eax
movl  $1, %ebx
movl  $testdata, %ecx
movl  $45, %edx
int $0x80
```

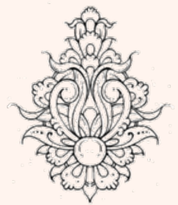
```
popl %ebx
movl %ebp, %esp
popl %ebp
ret
```



مثال ۱ (ادامه...)

```c
/* mainprog.c - An example of calling an assembly function */
#include <stdio.h>
void asmfunc();
int main(){
        printf("This is a test.\n");
        asmfunc();
        printf("Now for the second time.\n");
        asmfunc();
        printf("This completes the test.\n");
        return 0;
}
```

```
This is a test.
This is a test message from the asm function
Now for the second time.
This is a test message from the asm function
This completes the test.
```



مثال ۲

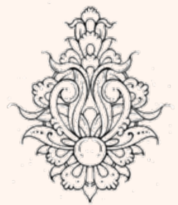
```
# square.s - An example of a function that returns an integer value
.type square, @function
.globl square
square:
pushl %ebp
movl %esp, %ebp
movl 8(%ebp), %eax
imull %eax, %eax
movl %ebp, %esp
popl %ebp
ret
```

این تابع یک مقدار صحیح دریافت و مربع آن را در خروجی چاپ می‌کند.

```
gcc -o inttest inttest.c square.s
```

```c
/* inttest.c - An example of returning an integer value */
#include <stdio.h>
int square(int);
int main(){
        int i = 2;
        int j = square(i);
        printf("The square of %d is %d\n", i, j);

        j = square(10);
        printf("The square of 10 is %d\n", j);
        return 0;
}
```

```
The square of 2 is 4
The square of 10 is 100
```



مثال ۳

می‌خواهیم تابعی بنویسیم که یک رشته برگرداند، به عنوان مثال شناسه‌ی پردازنده

Program Memory Space

main program code

0x8048460

call function

printf("%s\n", stringvalue);

C variable references data string memory location

function code

0x804848c

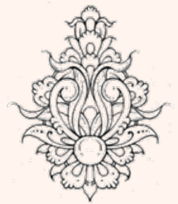
return with EAX=0x8049628

ret

data string

0x804628

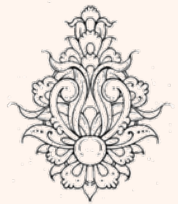
Memory Locations



مثال ۳

این تابع شناسه‌ی سازنده‌ی پردازنده را چاپ می‌کند:

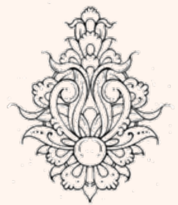
```
# cpuidfunc.s - An example of returning a string value
.section .bss
.comm output, 13
.section .text
.type cpuidfunc, @function
.globl cpuidfunc
cpuidfunc:
pushl %ebp
movl %esp, %ebp
pushl %ebx
pushl %edi
movl $0, %eax
cpuid
movl $output, %edi
movl %ebx, (%edi)
movl %edx, 4(%edi)
movl %ecx, 8(%edi)
movl $output, %eax
popl  %edi
popl %ebx
movl %ebp, %esp
popl %ebp
ret
```



مثال ۳ (ادامه...)

```
/* stringtest.c - An example of returning a string value */
#include <stdio.h>
char *cpuidfunc(void);
int main(){
        char *spValue;
        spValue = cpuidfunc();
        printf("The CPUID is: '%s'\n", spValue);
        return 0;
}
```

```
The CPUID is: 'GenuineIntel'
```

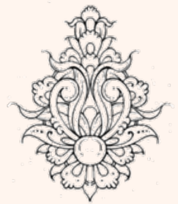


مثال ۴(استفاده از ممیز شناور)

```
.section .text
.type areafunc, @function
.globl areafunc
areafunc:
   pushl %ebp
   movl %esp, %ebp
   fldpi
   filds 8(%ebp)
   fmul %st(0), %st(0)
   fmul %st(1), %st(0)
   movl %ebp, %esp
   popl %ebp
   ret
```

```
The result is 314.159271
The result is 12.566371
```

```c
/* floattest.c - An example of using floating point return values */
#include <stdio.h>
float areafunc(int);
int main(){
        int radius = 10;
        float result;
        result = areafunc(radius);
        printf("The result is %f\n", result);

        result = areafunc(2);
        printf("The result is %f\n", result);
        return 0;
}
```

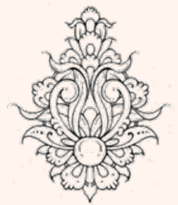


مثال ۵

```c
#include <stdio.h>
int greater(int, int);
int main(){
        int i = 10;
        int j = 20;
        int k = greater(i, j);
        printf("The larger value is %d\n", k);
        return 0;
}
```

```
.section .text
.globl greater
greater:
    pushl %ebp
    movl %esp, %ebp
    movl 8(%ebp), %eax
    movl 12(%ebp), %ecx
    cmpl %ecx, %eax
    jge end
    movl %ecx, %eax
end:
    movl %ebp, %esp
    popl %ebp
    ret
```

```
The larger value is 20
```



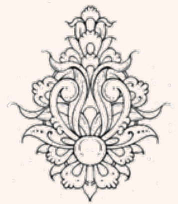
استفاده از توابع اسمبلی در C++

- شیوه‌ی فراخوانی تابع در C و C++ با وجود شباهت فراوان با هم تفاوت‌هایی دارد. برای استفاده از توابع C در C++ باید از واژه‌ی extern استفاده کرد:

```
#include <iostream>
using namespace std;
extern "C"{
   int square(int);
   float areafunc(int);
   char *cpuidfunc(void);
}
int main(){
   int radius = 10;
   int radsquare = square(radius);
   cout << "The radius squared is " << radsquare << endl;
   float result;
   result = areafunc(radius);
   cout << "The area is " << result << endl;
   cout << "The CPUID is " << cpuidfunc() << endl;
   return 0;
}
```

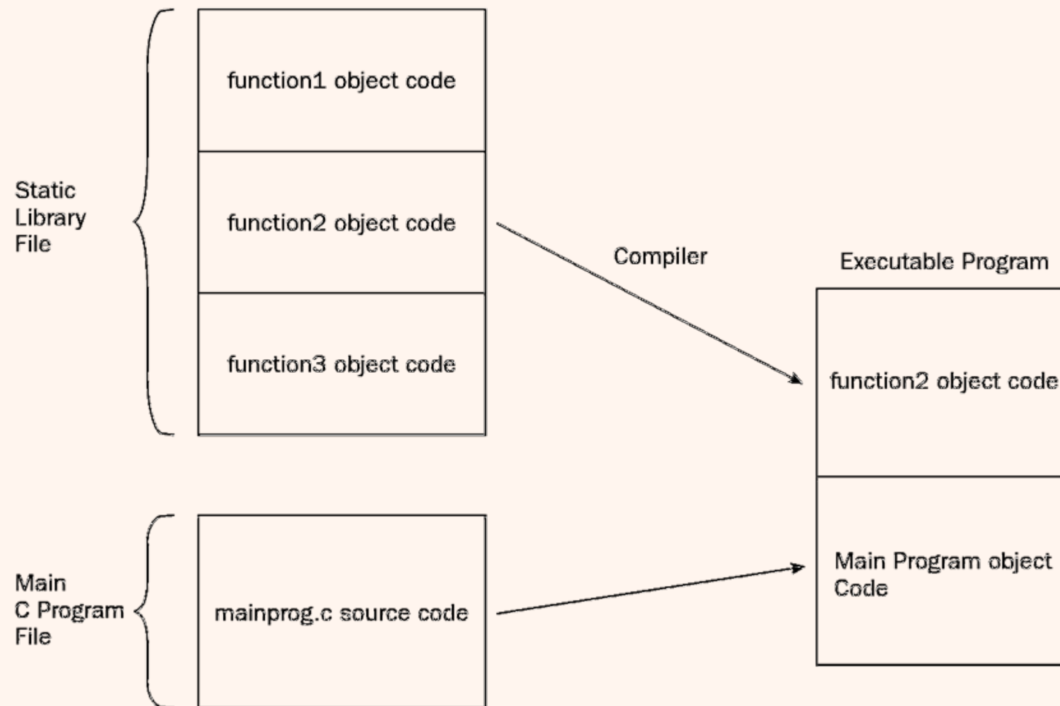
```
The radius squared is 100
The area is 314.159
The CPUID is GenuineIntel
```

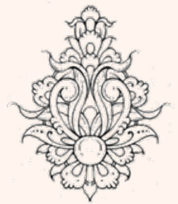
```
g++ -o externtest externtest.cpp square.o areafunc.o cpuidfunc.o
```



ایجاد کتابخانه‌ی ایستا

- استفاده از مجموعه‌ای از object fileها برای کامپایل کردن برنامه کار آسانی نیست، به ویژه آن که تعداد توابع زیاد باشد.
- با ایجاد یک کتابخانه از توابع می‌توان بر این مشکل غلبه کرد.
- در ادامه روش ایجاد کتابخانه‌ی ایستا به صورت مختصر مطرح می‌شود.



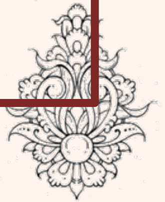


ایجاد کتابخانه‌ی ایستا

- در linux برای کتابخانه‌های ایستا از شیوه‌ی زیر استفاده می‌شود:
- libx.a
- x نام کتابخانه و پسوند a نوع کتابخانه را مشخص می‌کند.
- دستور ar برای ایجاد و اعمال تغییرات در کتابخانه‌ی ایستا مورد استفاده قرار می‌گیرند.
- برای ایجاد کتابخانه از پارامتر r استفاده می‌شود.

```
ar r libchap14.a square.o cpuidfunc.o
areafunc.o greater.o
```

```
ar: creating libchap14.a
```



```
gcc -o inttest inttest.c libchap14.a
```

کتابخانه‌ی ایستا

```
ar t libchap14.a
```

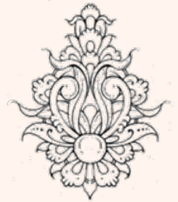
```
square.o
cpuidfunc.o
areafunc.o
greater.o
```

دستور **ar** سوییچ‌های مختلفی دارد، به عنوان مثال **t** جدولی از فایل‌ها را نمایش می‌دهد

تاثیر استفاده از کتابخانه‌ی ایستا بر حجم فایل اجرایی چیست؟

```
gcc -o inttest inttest.c square.o
```

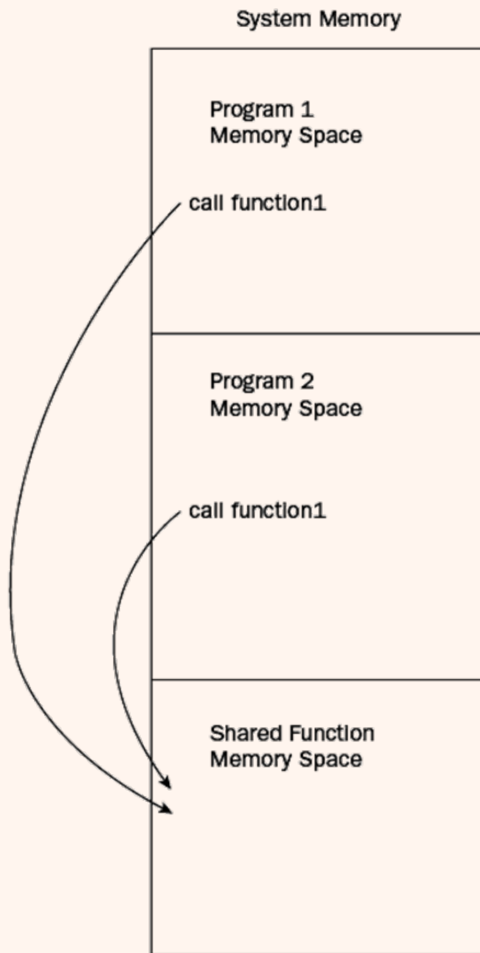
```
-rwxr-xr-x 1 ahmad ahmad 7202 2014-05-16 13:01 inttest
```



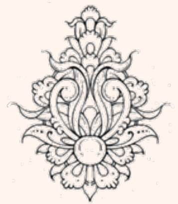
```
gcc -o inttest inttest.c libchap14.a
```

```
-rwxr-xr-x 1 ahmad ahmad 7202 2014-05-16 13:01 inttest
```

استفاده از کتابخانه‌های پویا



- پیش از این با مزایای کتابخانه‌های پویا آشنا شدیم.
- در ادامه به صورت مختصر با نحوه‌ی ساخت و استفاده از کتابخانه‌های پویا آشنا خواهیم شد.
- در linux برای نام‌گذاری کتابخانه‌های پویا از شیوه‌ی زیر استفاده می‌شود:
  - libx.so



ایجاد کتابخانه‌ی پویا

با استفاده از دستور زیر می‌توان یک کتابخانه‌ی پویا ساخت:

```
gcc -shared -o libchap14.so square.o
cpuidfunc.o areafunc.o greater.o
```

در عبارت زیر L– محل کتابخانه را مشخص می‌کند:

```
gcc -o inttest -L. -lchap14 inttest.c
```

```
./inttest: error while loading shared libraries: libchap14.so: cannot open shared object file: No
 such file or directory
```

```
ubuntu:~/MyData/courses/Asm/92_2/chap12$ ldd inttest
   linux-gate.so.1 =>  (0x00b4f000)
   libchap14.so => not found
   libc.so.6 => /lib/tls/i686/cmov/libc.so.6 (0x00d94000)
   /lib/ld-linux.so.2 (0x0039e000)
```

پیونده‌دهند پویا نیز باید از محل کتابخانه مطلع باشد، این کار به شیوه‌های مختلفی قابل انجام است. یکی از این راه‌ها استفاده از **متغیرهای محیطی** است:

```
export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:."
```

```
buntu:~/MyData/courses/Asm/92_2/chap12$ ldd inttest
 linux-gate.so.1 =>  (0x00259000)
 libchap14.so (0x00dfd000)
 libc.so.6 => /lib/tls/i686/cmov/libc.so.6 (0x00384000)
 /lib/ld-linux.so.2 (0x00367000)
```

```
ahmad@ubuntu:~/MyData/courses/Asm/92_1/chap14$ ./inttest
The square of 2 is 4
The square of 10 is 100
```

Inline Assembly

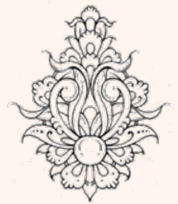
Inline Assembly

- با استفاده از دستور زیر می‌توان در یک برنامه به زبان C به صورت مستقیم از دستورات اسمبلی استفاده کرد:

```
asm ( "assembly code" );
```

- بدین ترتیب در بین " " دستورات اسمبلی قرار دارند.
  - بین دستورات از کاراکتر '\n' استفاده می‌شود.
  - در برخی کامپایلرها استفاده از کاراکتر tab قبل از هر دستور الزامی است.

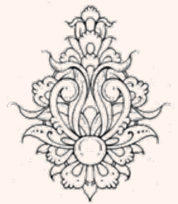
```
asm ( "movl $1, %eax\n\tmovl $0, %ebx\n\tint $0x80" ) ;
```



```
asm( "movl $1, %eax\n\t"
     "movl $0, %ebx\n\t"
     "int  $0x80" );
```

Inline Assembly (ادامه...)

```c
#include <stdio.h>

int main()
{
    int a = 10;
    int b = 20;
    int result;
    result = a * b;
    asm ( "nop" );
    printf("The result is %d\n",result);
    return 0;
}
```



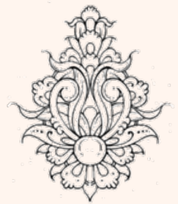
Inline Assembly (ادامه...)

```
main:
pushl %ebp
movl %esp, %ebp
andl $-16, %esp
subl $32, %esp
movl $10, 28(%esp)
movl $20, 24(%esp)
movl 28(%esp), %eax
imull 24(%esp), %eax
movl %eax, 20(%esp)
#APP
# 10 "asmtest.c" 1
nop
# 0 "" 2
#NO_APP
```

```
movl $.LC0, %eax
movl 20(%esp), %edx
movl %edx, 4(%esp)
movl %eax, (%esp)
call printf
movl $0, %eax
leave
ret
.size main, .-main
.ident "GCC: (Ubuntu
4.4.3-4ubuntu5) 4.4.3"
.section
.note.GNUstack,"",@prog
bits
```

بخشی که توسط کاربر تولید شده است

بخشی که توسط کامپایلر تولید شده است



استفاده از متغیرهای سراسری

- در برنامه‌های اسمبلی نوشته شده به زبان C می‌توان از «متغیرهای سراسری» تعریف شده در زبان نیز استفاده نمود:

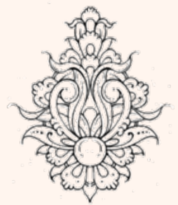
```c
#include <stdio.h>

int a = 10;
int b = 20;
int result;

int main(){
        asm ( "pusha\n\t"
              "movl a, %eax\n\t"
              "movl b, %ebx\n\t"
              "imull %ebx, %eax\n\t"
              "movl %eax, result\n\t"
              "popa");
        printf("the answer is %d\n", result);
        return 0;
}
```

با این شیوه می‌توان اطمینان داشت که تداخلی در استفاده از ثبات‌ها پیش نخواهد آمد.

استفاده از متغیر

```
.file "globaltest.c"
.globl a
.data
.align 4
.type a, @object
.size a, 4
a:
.long 10
.globl b
.align 4
.type b, @object
.size b, 4
b:
.long 20
.comm result,4,4
.section .rodata
.LC0:
.string "the answer is %d\n"
.text
.globl main
.type main, @function
```

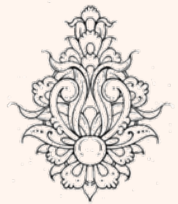
```
main:
pushl %ebp
movl %esp, %ebp
andl $-16, %esp
subl $16, %esp
#APP
# 9 "globaltest.c" 1
pusha
movl a, %eax
movl b, %ebx
imull %ebx, %eax
movl %eax, result
popa
# 0 "" 2
#NO_APP
movl result, %edx
movl $.LC0, %eax
movl %edx, 4(%esp)
movl %eax, (%esp)
call printf
movl $0, %eax
leave
ret
```



بهینه‌سازی توسط کامپایلر

- ممکن است کامپایلر، بهینه‌سازی‌هایی روی برنامه‌ی نوشته شده انجام دهد، حذف تابع‌هایی که هرگز فراخوانی نشده‌اند، اشتراک ثبات بین متغیرهایی که هم‌زمان مورد استفاده قرار نمی‌گیرند و حتی جابه‌جایی برخی دستورالعمل‌ها
- گاهی مواقع نوشتن برنامه این بهینه‌سازی‌ها بدتر کار را خراب می‌کند، در این شرایط است که باید گفت:

«ما را به خیر تو امید نیست!»



- در این شرایط کلمه‌ی «volatile» مورد استفاده قرار می‌گیرد.

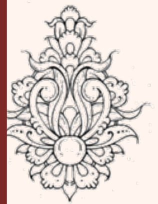
```
asm  volatile ( "assembly code" ) ;
```

سایر کامپایلرها

- در همه‌ی موارد و با همه‌ی کامپایلرها برای افزودن دستورالعمل‌های اسمبلی به یک شیوه برخورد نمی‌شود؛ در برخی کامپایلرها از «__asm__» استفاده می‌کنند.

```
__asm __ __volatile__ ( "pushla\n\t"
"movl a, %eax\n\t"
movl b, %ebx\n\t"
"imull %ebx, %eax\n\t"
"movl %eax, result\n\t"
"popa" ) ;
```

در visual studio از «__asm» استفاده می‌شود.

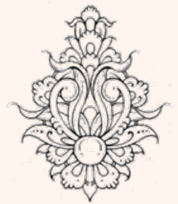


شیوه‌ی مبسوط

- اضافه کردن کدهای اسمبلی با این شیوه با محدودیت‌هایی همراه است، به عنوان مثال باید مراقب بود ثبات‌ها تغییر نکنند و یا تنها می‌توان به متغیرهای سراسری دسترسی داشت. از این رو شیوه‌ی دیگری مطرح می‌شود:

```
asm ( "assembly code" :output location : input operands : changed registers) ;
```

- در این شیوه چهار قسمت وجود دارد:
  - کد اسمبلی
  - لیست ثبات‌ها و حافظه‌های ورودی
  - لیست ثبات‌ها و حافظه‌های خروجی
  - لیست ثبات‌های تغییر یافته

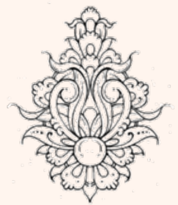


مشخص کردن ورودی‌ها و خروجی‌ها

- در این شیوه ورودی‌ها و خروجی‌ها هر دو می‌توانند از بین خانه‌های حافظه و یا ثبات‌ها انتخاب شوند:

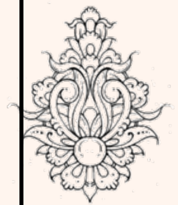
```
"constraint"(variable)
```

- در عبارت بالا variable یکی از متغیرهای برنامه به زبان C است.
- constraint مشخص می‌کند این متغیر باید در کجا قرار بگیرد (برای ورودی‌ها) و یا باید به کجا منتقل شود. (برای خروجی‌ها)
  - این بخش یک کاراکتر است.
  - برای خروجی پیش از این کاراکتر باید از **=** یا **+** استفاده شود. اولی برای حالتی که تنها برای ارسال خروجی مورد استفاده قرار گیرد و دومی برای حالتی که هم برای خواندن و هم برای نوشتن استفاده شود.



مشخص کردن ورودی‌ها و خروجی‌ها

Constraint	Description
a	Use the %eax, %ax, or %al registers.
b	Use the %ebx, %bx, or %bl registers.
c	Use the %ecx, %cx, or %cl registers.
d	Use the %edx, %dx, or $dl registers.
S	Use the %esi or %si registers.
D	Use the %edi or %di registers.
r	Use any available general-purpose register.
q	Use either the %eax, %ebx, %ecx, or %edx register.
A	Use the %eax and the %edx registers for a 64-bit value.
f	Use a floating-point register.
t	Use the first (top) floating-point register.
u	Use the second floating-point register.
m	Use the variable’s memory location.
i	Use an immediate integer value.
n	Use an immediate integer value with a known value.
g	Use any register or memory location available.



استفاده از ثبات‌ها

```c
/* regtest1.c - An example of using registers */
#include <stdio.h>
int main(){
   int data1 = 10;
   int data2 = 20;
   int result;

   asm ("imull %%edx, %%ecx\n\t"
        "movl %%ecx, %%eax"
        : "=a"(result)
        : "d"(data1), "c"(data2));

   printf("The result is %d\n", result);
   return 0;
}
```

```
main:
pushl %ebp
movl %esp, %ebp
andl $-16, %esp
subl $32, %esp
movl $10, 28(%esp)
movl $20, 24(%esp)
movl 28(%esp), %eax
movl 24(%esp), %ecx
movl %eax, %edx
#APP
# 10 "regtest1.c" 1
imull %edx, %ecx
movl %ecx, %eax
# 0 "" 2
#NO_APP
movl %eax, 20(%esp)
```

```
ahmad@ubuntu:~/Courses/Assembly/chapter9$ ./regtest1
The result is 200
```

استفاده از ثبات‌ها(ادامه...)

```c
/* regtest1.1.c - An example of
#include <stdio.h>
int main(){
    int data1 = 10;
    int data2 = 20;
    asm ("imull %%edx, %%eax\n\t"
         : "+a"(data1)
         : "d"(data2));

    printf("The datat1= is %d\n",
    return 0;
}
```

```
The datat1= is 200
```

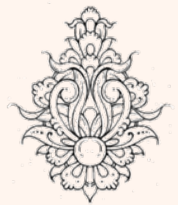
```
main:
        pushl  %ebp
        movl   %esp, %ebp
        andl   $-16, %esp
        subl   $32, %esp
        movl   $10, 28(%esp)
        movl   $20, 24(%esp)
        movl   24(%esp), %edx
        movl   28(%esp), %eax
#APP
# 6 "regtest1.1.c" 1
        imull %edx, %eax

# 0 "" 2
#NO_APP
        movl   %eax, 28(%esp)
        movl   $.LC0, %eax
        movl   28(%esp), %edx
        movl   %edx, 4(%esp)
        movl   %eax, (%esp)
```

Placeholders

در مثال قبل ورودی‌ها را در ثبات‌هایی خاص قرار دادیم و از آن استفاده کردیم، در حالاتی که بخواهیم ورودی‌ها توسط کامپایلر انتخاب شوند، به شیوه‌ی زیر عمل می‌کنیم:

```
asm ("assembly code"
: "=r"(result)
: "r"(data1), "r"(data2));
```

```
imull %1, %2
movl %2, %0
```



در ادامه به متغیر result با %0، به متغیر data1 با %1 و به متغیر data2 با %2 می‌توان دسترسی داشت.

بدین ترتیب می‌توان به خانه‌های حافظه هم دسترسی داشت، اما باید در انتخاب دقت لازم اعمال شود

مثال

```c
#include <stdio.h>

int main()
{
    int data1 = 10;
    int data2 = 20;
    int result;

    asm ("imull %1, %2\n\t"
         "movl %2, %0"
         : "=r"(result)
         : "r"(data1), "r"(data2));

    printf("The result is %d\n", resu
    return 0;
}
```

```
        movl    $10, -4(%ebp)
        movl    $20, -8(%ebp)
        movl    -4(%ebp), %edx
        movl    -8(%ebp), %eax
#APP
        imull %edx, %eax
        movl %eax, %eax
#NO_APP
        movl    %eax, -12(%ebp)
```

```
movl28(%esp), %eax
movl24(%esp), %edx
#APP
# 10 "regtest2.c" 1
imull %eax, %edx
movl %edx, %eax
# 0 "" 2
#NO_APP
movl%eax, 20(%esp)
```

```
ahmad@ubuntu:~/Courses/Assembly/chapter9$ ./regtest2
The result is 200
```

ورودی و خروجی یکسان

- می‌توان از ورودی و خروجی یکسان استفاده نمود:

```
#include <stdio.h>

int main()
{
    int data1 = 10;
    int data2 = 20;

    asm ("imull %1, %0"
         : "=r"(data2)
         : "r"(data1), "0"(data2));

    printf("The result is %d\n", data2);
    return 0;
}
```

```
        movl   28(%esp), %edx
        movl   24(%esp), %eax
#APP
# 9 "regtest3.c" 1
        imull %edx, %eax
# 0 "" 2
#NO_APP
        movl   %eax, 24(%esp)
```

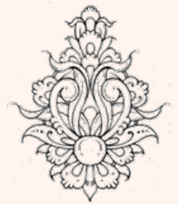
استفاده از نام دلخواه به جای شماره

```
%[name]"constraint"(variable)
```

```
        movl    28(%esp), %edx
        movl    24(%esp), %eax
#APP
# 9 "alttest.c" 1
        imull %edx, %eax
# 0 "" 2
#NO_APP
        movl    %eax, 24(%esp)
```

```
#include <stdio.h>

int main()
{
    int data1 = 10;
    int data2 = 20;

    asm ("imull %[value1], %[value2]"
        : [value2] "=r"(data2)
        : [value1] "r"(data1), "0"(data2));

    printf("The result is %d\n", data2);
    return 0;
}
```

```
The result is 200
```



فهرست ثبات‌های تغییر یافته

```c
#include <stdio.h>

int main()
{
   int data1 = 10;
   int result = 20;

   asm ("addl %1, %0"
        : "=d"(result)
        : "c"(data1), "0"(result)
        : "%ecx", "%edx");

   printf("The result is %d\n", result);
   return 0;
}
```

```
ahmad@ubuntu:~/Courses/Assembly/chapter9$ gcc badregtest.c -S
badregtest.c: In function ‘main’:
badregtest.c:9: error: can't find a register in class ‘DREG’ while reloading ‘asm’
badregtest.c:9: error: ‘asm’ operand has impossible constraints
```

در لیست ثبات‌های تغییریافته، ثبات‌های ورودی و خروجی نباید ذکر شود

فهرست ثبات‌های تغییر یافته(ادامه...)

```c
#include <stdio.h>

int main()
{
    int data1 = 10;
    int result = 20;

    asm ("movl %1, %%eax\n\t"
         "addl %%eax, %0"
         : "=r"(result)
         : "r"(data1), "0"(result)
         : "%eax");

    printf("The result is %d\n", result);
    return 0;
}
```

```
        movl    $10, 28(%esp)
        movl    $20, 24(%esp)
        movl    28(%esp), %ecx
        movl    24(%esp), %eax
        movl    %eax, %edx
#APP
# 9 "changedtest.c" 1
        movl %ecx, %eax
        addl %eax, %edx
# 0 "" 2
#NO_APP
        movl    %edx, 24(%esp)
```

```
The result is 30
```

استفاده از حافظه

```
#include <stdio.h>

int main()
{
    int dividend = 20;
    int divisor = 5;
    int result;

    asm("divb %2\n\t"
        "movl %%eax, %0"
        : "=m"(result)
        : "a"(dividend), "m"(divisor));

    printf("The result is %d\n", result);
    return 0;
}
```

```
main:
        pushl   %ebp
        movl    %esp, %ebp
        andl    $-16, %esp
        subl    $32, %esp
        movl    $20, 28(%esp)
        movl    $5, 24(%esp)
        movl    28(%esp), %eax
#APP
# 10 "memtest.c" 1
        divb 24(%esp)
        movl %eax, 20(%esp)
# 0 "" 2
#NO_APP
```

```
The result is 4
```

استفاده از اعداد ممیز شناور

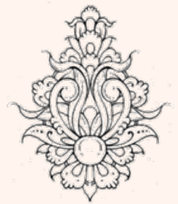
- نظر به این که اعداد ممیز شناور، در ثبات‌هایی ذخیره می‌شوند که ساختار پشته دارند، قضیه کمی متفاوت است.
  - در این حالت f به معنای هر ثبات ممیز شناور است
  - t برای استفاده از ثبات st(0) به کار می‌رود.
  - و u ثبات st(1) را نشان می‌دهد.
- هنگامی که بخواهیم خروجی بگیریم باید از t یا u استفاده کنیم.

مثال

```
The cosine is 0.000001, and the sine is 1.000000
```

```c
#include <stdio.h>

int main()
{
   float angle = 90;
   float radian, cosine, sine;

   radian = angle / 180 * 3.14159;

   asm("fsincos"
       :"=t"(cosine), "=u"(sine)
       :"0"(radian));

   printf("The cosine is %f, an
   return 0;
}
```

```
flds    44(%esp)
#APP
# 11 "sincostest.c" 1
        fsincos
# 0 "" 2
#NO_APP
        fstps   68(%esp)
        fstps   64(%esp)
        flds    64(%esp)
        flds    68(%esp)
        fxch    %st(1)
```

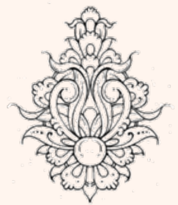


مثال

```
        movl    $10, 28(%esp)
#APP
# 9 "areatest.c" 1
        fild 28(%esp)
        fimul 28(%esp)
        fldpi
        fmul %st(1), %st(0)
# 0 "" 2
#NO_APP
        fstps  24(%esp)
        flds   24(%esp)
```

```c
#include <stdio.h>

int main()
{
    int radius = 10;
    float area;

    asm("fild %1\n\t"
        "fimul %1\n\t"
        "fldpi\n\t"
        "fmul %%st(1), %%st(0)"
        : "=t"(area)
        :"m"(radius)
        : "%st(1)");

    printf("The result is %f\n", area);
    return 0;
}
```

```
The result is 314.159271
```



استفاده از دستورهای پرش

```c
#include <stdio.h>

int main()
{
    int a = 10;
    int b = 20;
    int result;

    asm("cmp %1, %2\n\t"
        "jge greater\n\t"
        "movl %1, %0\n\t"
        "jmp end\n"
        "greater:\n\t"
        "movl %2, %0\n"
        "end:"
        :"=r"(result)
        :"r"(a), "r"(b));

    printf("The larger value is %d\n", result);
    return 0;
}
```

```
        movl    28(%esp), %eax
        movl    24(%esp), %edx
#APP
# 10 "jmptest.c" 1
        cmp %eax, %edx
        jge greater
        movl %eax, %eax
        jmp end
greater:
        movl %edx, %eax
end:
# 0 "" 2
#NO_APP
        movl    %eax, 20(%esp)
        movl    $.LC0, %eax
```

```
The larger value is 20
```

در صورتی که بخواهیم دو قسمت یکسان اسمبلی داشته باشیم این دو قسمت نباید از برچسب‌های یکسانی استفاده کنند.

استفاده از دستورهای پرش

```c
#include <stdio.h>

int main()
{
    int a = 10;
    int b = 20;
    int result;

    asm("cmp %1, %2\n\t"
        "jge 0f\n\t"
        "movl %1, %0\n\t"
        "jmp 1f\n"
        "0:\n\t"
        "movl %2, %0\n"
        "1:"
        :"=r"(result)
        :"r"(a), "r"(b));

    printf("The larger value is %d\n", result);
    return 0;
}
```

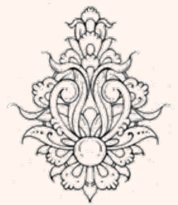
```
#APP
# 10 "jmptest2.c" 1
        cmp %eax, %edx
        jge 0f
        movl %eax, %eax
        jmp 1f
0:
        movl %edx, %eax
1:
# 0 "" 2
#NO_APP
        movl    %eax, 20(%esp)
        movl    $.LC0, %eax
```

مثال(چاپ شناسه‌ی پردازنده)

```c
/* cpuid.c */
#include <stdio.h>

int main(){
  char output[13];
  asm(  "mov $0, %%eax\n\t"
        "cpuid\n\t"
        "lea %[outstr], %%edi\n\t"
        "mov %%ebx, (%%edi)\n\t"
        "mov %%edx, 4(%%edi)\n\t"
        "mov %%ecx, 8(%%edi)\n\t"
        :[outstr]"=m"(output)
        :
        :"%eax","%ebx","%ecx","%edx","%edi");
        output[13]='\0';

   printf("%s\n", output);
   return 0;
}
```

```
#APP
# 6 "cpuid.c" 1
        mov $0, %eax
        cpuid
        lea 31(%esp), %edi
        mov %ebx, (%edi)
        mov %edx, 4(%edi)
        mov %ecx, 8(%edi)

# 0 "" 2
#NO_APP
```

```
GenuineIntel
```



استفاده از ماکرو

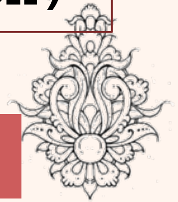
- در C و C++ می‌توان برای استفاده از یک ثابت و یا متی یک تابع پیچیده از ماکرو استفاده نمود.

```
#define NAME expression
```

```
#define MAX_VALUE 1024
```

```
#define NAME(input values, output value) (function)
```

```
#define SUM(a, b, result) ((result) = (a) + (b))
```



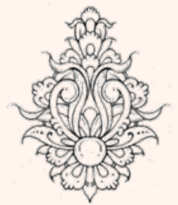
مثال

```c
#include <stdio.h>

#define SUM(a, b, r
        ((resul

int main()
{
    int data1 = 5, d
    int result;
    float fdata1 = 5
    float fresult;

    SUM(data1, data2
    printf("The resu
    SUM(1, 1, result
    printf("The resu
    SUM(fdata1, fdat
    printf("The floating result is %f\n", fresult);
    SUM(fdata1, fdata2, result);
    printf("The mixed result is %d\n", result);
    return 0;
}
```

```c
int main()
{
    int data1 = 5, data2 = 10;
    int result;
    float fdata1 = 5.0, fdata2 = 10.0;
    float fresult;

    ((result) = (data1) + (data2));
    printf("The result is %d\n", result);
    ((result) = (1) + (1));
    printf("The result is %d\n", result);
    ((fresult) = (fdata1) + (fdata2));
    printf("The floating result is %f\n", fresult);
    ((result) = (fdata1) + (fdata2));
    printf("The mixed result is %d\n", result);
    return 0;
}
```

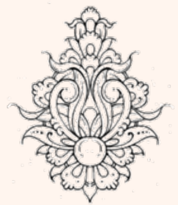
استفاده از ماکرو

```
#define GREATER(a, b, result) ({ \
        asm("cmp %1,%2\n\t" \
        "jge 0f\n\t" \
        "movl %1,%0\n\t" \
        "jmp 1f\n " \
        "0:\n\t" \
        "movl %2,%0\n " \
        "1:" \
        :"=r"(result) \
        :"r"(a), "r"(b)); })
```



استفاده از ماکرو

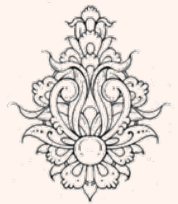
```
#include <stdio.h>

#define GREATER(a, b, result) ({ \
    asm("cmp %1, %2\n\t" \
        "jge 0f\n\t" \
        "movl %1, %0\n\t" \
        "jmp 1f\n\t" \
        "0:\n\t" \
        "movl %2, %0\n\t" \
        "1:" \
        :"=r"(result) \
        :"r"(a), "r"(b)); })

int main()
{
   int data1 = 10;
   int data2 = 20;
   int result;

   GREATER(data1, data2, result);
   printf("a = %d, b = %d    result: %d\n", data1, data2, result);

   data1 = 30;
   GREATER(data1, data2, result);
   printf("a = %d, b = %d    result: %d\n", data1, data2, result);
```



تکنیک‌های بهینه‌سازی

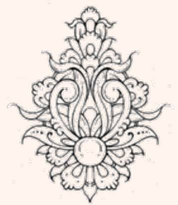
پیش‌گفتار

- بیشتر کامپایلرها افزون بر تبدیل زبان سطح بالا امکان بهینه‌سازی برنامه‌ی کامپایل شده را دارند.
- این کار معمولا به بهای افزایش زمان کامپایل انجام می‌شود.
- با توجه به تجربیاتی که در زبان اسمبلی کسب شده است، می‌توانیم با این تکنیک‌ها بیشتر آشنا شویم و در مواردی علاوه بر بهینه‌ساز کامپایلر به صورت مضاعف آن شیوه‌ها را اعمال کنیم.
- موضوع فصل پانزدهم کتاب آشنایی با چنین شیوه‌هایی است.



بهینه‌سازی

- در روزگار نخست برنامه‌نویسی به زبان C، دست‌کاری کد اسمبلی برای بهینه‌سازی معمول بوده است.
- در حال حاضر می‌توان برای چنین کاری از کامپایلر کمک گرفت.
- با استفاده از «-o» یک سطح ساده از بهینه‌سازی انجام می‌شود، با «-o2» و «-o3» سطوح پیشرفته‌تری از بهینه‌سازی انجام می‌شود.
- در صورتی که تکنیک خاصی مد نظر بود از «-f....» استفاده می‌شود.
- در واقع «-ox» ها مجموعه‌ای از روش‌های بهینه‌سازی را به صورت مجموع انجام می‌دهند.



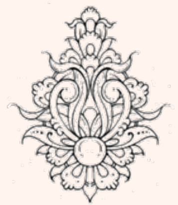
```
gcc -O2 -funroll-all-loops --save-temps unroll.c
```

بهینه‌سازی سطح یک

- -fdefer-pop:
- خالی کردن محتوای روی پشته(پارمترهای تابع) را به تعویق می‌اندازد.
- -fmerge-constant
- در این حالت کامپایلر سعی می‌کند از ثابت‌های یکسانی استفاده کند.
  – این شیوه مربوط به دستورهایی است که عملوند بلاواسطه ندارند.
- -fthread-jumps
- کامپایلر مسیر پرش متوالی را دنبال کرده و در صورت امکان مسیر پرش را کوتاه‌تر می‌کند.

بهینه‌سازی سطح یک(ادامه...)

- -floop-optimize:
- با انتقال برخی دستورها در حلقه و دستورهای پرش شرطی خاتمه دهنده به حلقه بهینه‌سازی می‌شوند.
- -fif-conversion
- با استفاده از جابجایی شرطی و تکنیک‌های محاسباتی عملکرد را بهبود می‌بخشد.
- -fif-conversion2
- از روش‌های محاسباتی پیچیده‌تر استفاده می‌کند.



بهینه‌سازی سطح یک(ادامه...)

- -fguess-branch-probability
- کامپایلر با حدس احتمال رخداد پرش‌های شرطی دستورها را جابجا می‌کند.
  - با f-nofg-guess-branch-probability می‌توان این بهینه‌سازی را غیرفعال کرد.
- -fcprop-registers
- برای برخی متغیرها که بین بخش‌های مختلف یکسان مورد استفاده قرار می‌گیرند، یک ثبات خاص در نظر گرفته می‌شود.