زبان ماشین و اسمبلی
(۱۳-۱۱-۰۰۵)

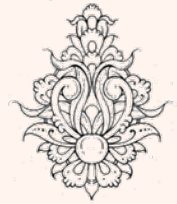
مراحل ساختن فایل اجرایی

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

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

فهرست مطالب

- مراحل ساخت فایل اجرایی
  - پیش‌پردازش
  - کامپایل
  - اسمبل
    - بخش‌های مختلف object file
  - لینک کردن
  - لینک کردن پویا در برابر لینک کردن ایستا



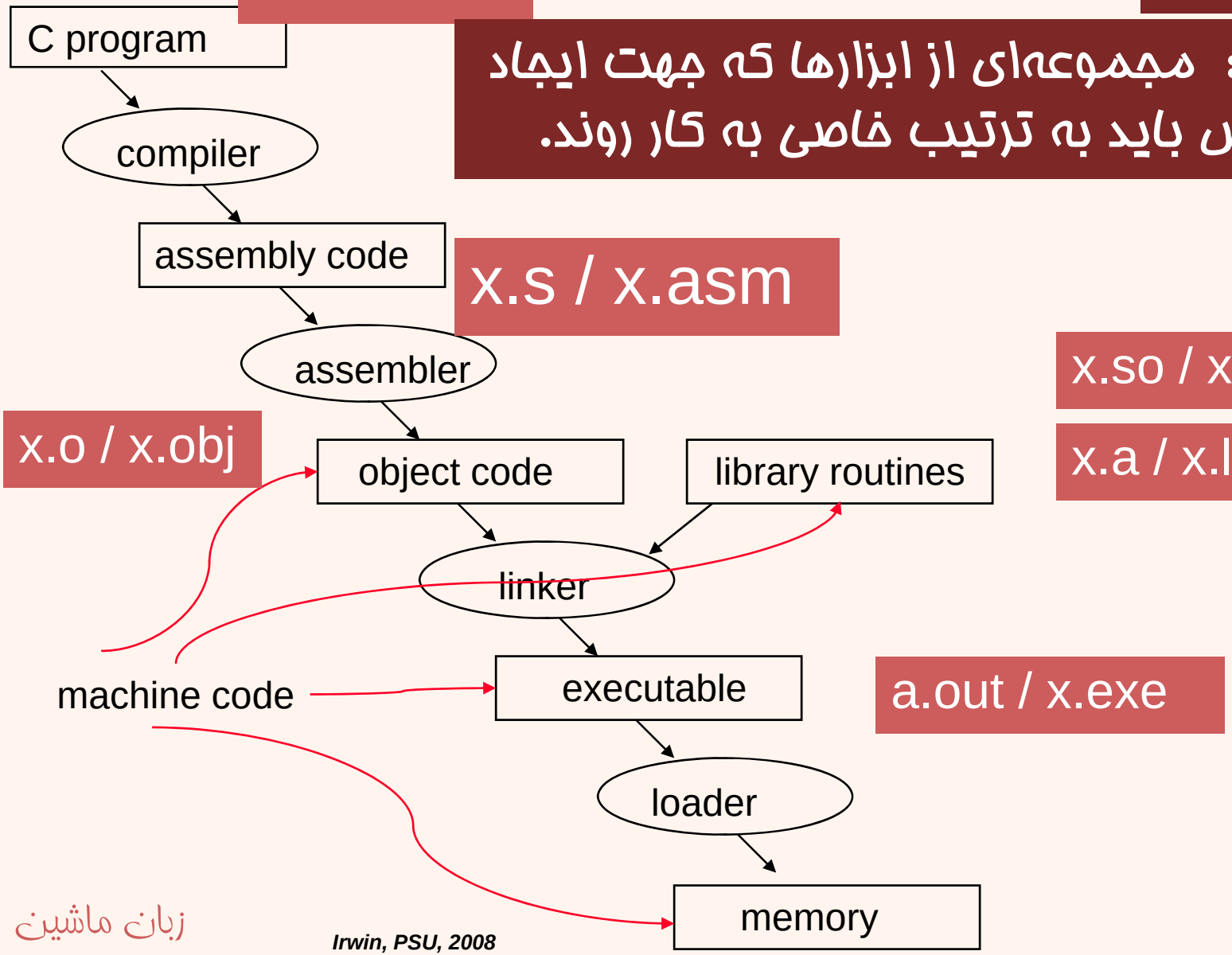
Linux/Windows(DOS)

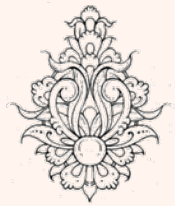
ترجمه‌ی یک برنامه

x.c / x.c

toolchain

زنجیره‌ی ابزار: مجموعه‌ای از ابزارها که جهت ایجاد محصولی خاص باید به ترتیب خاصی به کار روند.





مراحل تبدیل برنامه به فایل قابل اجرا

C PreProcess

- پیش‌پردازش

```
cpp hello.c > hello.i
```

- کامپایل

```
gcc -S hello.i
```

  - تبدیل کد C به اسمبلی
  - ممکن است کد اسمبلی شامل شبه‌دستور هم باشد.

- اسمبل کردن

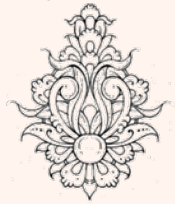
```
as hello.s -o hello.o
```

  - تبدیل کد اسمبلی به زبان ماشین

- لینک کردن

```
gcc hello.o –o hello
```

  - فایل اجرایی نهایی را فراهم می‌کند.



مثال(Hello world)

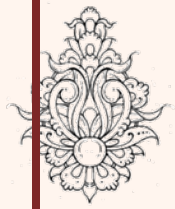
hello.c

```c
#include <stdio.h>
int main(){
   printf("Hello, World!\n");
   return 0;
}
```

cpp hello.c > hello.i

hello.i

```
extern int printf (__const char *__restrict __format, ...);

        .
        .
        .

# 2 "hello.c" 2
int main(){
   printf("Hello, World!\n");
   return 0;
}
```

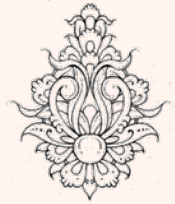


مثال

```
#include <stdio.h>
#define LIN
int main(){
#ifdef LIN
  system("clear");
  printf("Hi!\n");
#else
  system("cls");
  printf("Hello!\n");
#endif
return 0;
}
```

gcc –E filename.c

```
.
.
.
extern int ftrylockfile (FILE *__stream)
__attribute__ ((__nothrow__)) ;
extern void funlockfile (FILE *__stream)
__attribute__ ((__nothrow__));
# 916 "/usr/include/stdio.h" 3 4

# 2 "test_cpp.c" 2

int main(){

  system("clear");
  printf("Hi!\n");



return 0;
}
```

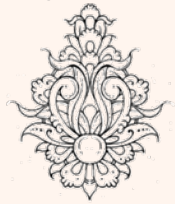


ادامه‌ی مثال

gcc -S hello.i

```
            .file       "hello.c"
            .section    .rodata
.LC0:
            .string     "Hello, World!"
            .text
.globl main
            .type       main, @function
main:
            pushl       %ebp
            movl        %esp, %ebp
            andl        $-16, %esp
            subl        $16, %esp
            movl        $.LC0, (%esp)
            call        puts
            movl        $0, %eax
            leave
            ret
            .size       main, .-main
            .ident      "GCC: (Ubuntu 4.4.3-4ubuntu5.1) 4.4.3"
            .section    .note.GNU-stack,"",@progbits
```

hello.s



- بسته به نوع کامپایلر، کد اسمبلی تولید شده ممکن است تفاوت‌هایی داشته باشد (این فرایند یک به یک نیست).
- در این فایل فراخوانی توابع موجود در فایل‌های دیگر نیز دیده می‌شود.

ادامه‌ی مثال

as hello.s -o hello.o

```
ahmad@ubuntu:~/MyData/courses/Asm/92_1/chaintools$ cat hello.o
ELF4(
           U Hello, World!GCC: (Ubuntu 4.4.3-4ubuntu5.1) 4.4.
3.symtab.strtab.shstrtab.rel.text.data.bss.rodata.comment.note.GNU-stack
                ^&AQ
                          ello.cmainputs
                                                   ahmad@ubuntu:~/MyData/co
```

```
ahmad@ubuntu:~/MyData/courses/Asm/92_1/chaintools$ objdump -d hello.o

hello.o:     file format elf32-i386


Disassembly of section .text:

00000000 <main>:
   0:   55                      push   %ebp
   1:   89 e5                   mov    %esp,%ebp
   3:   83 e4 f0                and    $0xfffffff0,%esp
   6:   83 ec 10                sub    $0x10,%esp
   9:   c7 04 24 00 00 00 00    movl   $0x0,(%esp)
  10:   e8 fc ff ff ff          call   11 <main+0x11>
  15:   b8 00 00 00 00          mov    $0x0,%eax
  1a:   c9                      leave
  1b:   c3                      ret
```

• با استفاده از objdump می‌توان محتوای object file را مشاهد نمود. این برنامه با پارامترهای مختلف قابل استفاده است.

ادامه‌ی مثال

ld -dynamic-linker /lib/ld-linux.so.2 -lc hello.o -e main ....

gcc hello.o

- در صورتی که از gcc استفاده کنیم، همه‌ی این مراحل انجام می‌شود.

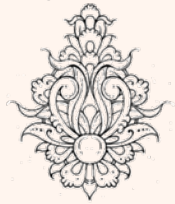
```
Disassembly of section .text:

080481a4 <main>:
 80481a4:       55                      push   %ebp
 80481a5:       89 e5                   mov    %esp,%ebp
 80481a7:       83 e4 f0                and    $0xfffffff0,%esp
 80481aa:       83 ec 10                sub    $0x10,%esp
 80481ad:       c7 04 24 c0 81 04 08    movl   $0x80481c0,(%esp)
 80481b4:       e8 db ff ff ff          call   8048194 <puts@plt>
 80481b9:       b8 00 00 00 00          mov    $0x0,%eax
 80481be:       c9                      leave
 80481bf:       c3                      ret
```

- در صورتی که از پارامتر –save-temps استفاده کنیم، نتیجه‌ی تمام مراحل نیز در فایل‌هایی جداگانه ذخیره می‌شود.

gcc hello.c -save-temps

اسمبلر

- از راهنماها(directives) برای تبدیل به کد ماشین استفاده می‌کند.(راهنماها به دستور تبدیل نمی‌شوند)
- اسمبلر شبه‌دستورها را با دستورهای زبان جایگزین می‌کند.
- نهایتا دستورها را به زبان ماشین تبدیل می‌کند.
- نماد دستورها به سادگی قابل تبدیل به کد ماشین هستند.
- در مورد عملوند دستورها چه می‌توان گفت؟ (به عنوان مثال: دستورهای پرش)

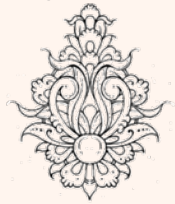


اسمبلر و عملوندها

- دستورهای که از نشانی‌دهی pc-relative استفاده می‌کنند به سادگی تبدیل می‌شوند (آدرس به محل قرارگیری برچسب بستگی ندارد، چرا که از فاصله‌ی نسبی استفاده می‌شود).

```
Loop: sll  $t1, $s3, 2
      add  $t1, $t1, $s6
      lw   $t0, 0($t1)
      bne  $t0, $s5, Exit
      addi $s3, $s3, 1
      j    Loop
Exit: …
```

مثال MIPS

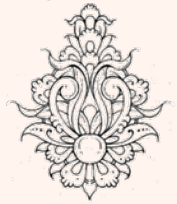


اسمبلر و عملوندها

- بعضی از برچسب‌ها روبه جلو(forward reference) هستند، اسمبلر قبل از دیدن آن‌ها باید آدرس آن‌ها را مشخص کند.
- برای این منظور اسمبل کردن در دو گام صورت می‌گیرد:

– ابتدا موقعیت همه برچسب‌ها خوانده می‌شود و یک جدول از نمادها (symbol table) تشکیل می‌شود.

– از اطلاعات جمع‌آوری شده برای تبدیل به کد ماشین استفاده می‌شود.

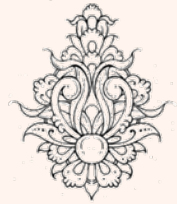


اسمبلر و عملوندها(ادامه...)

- برای عملوندهایی که از آدرس‌دهی مستقیم استفاده می‌کنند باید محل قرارگیری نهایی در حافظه مشخص شود.
  - به عنوان مثال برچسب مشخص کننده داده یا برچسب دستورهای پرش با آدرس‌دهی مستقیم
- این اطلاعات تنها در زمان لینک کردن قابل دسترسی خواهد بود.

Realocation table

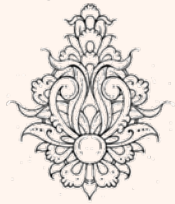
- از این جهت اطلاعات این برچسب‌ها در جدولی به نام realocation table همراه با object file قرار می‌گیرد.



Object file

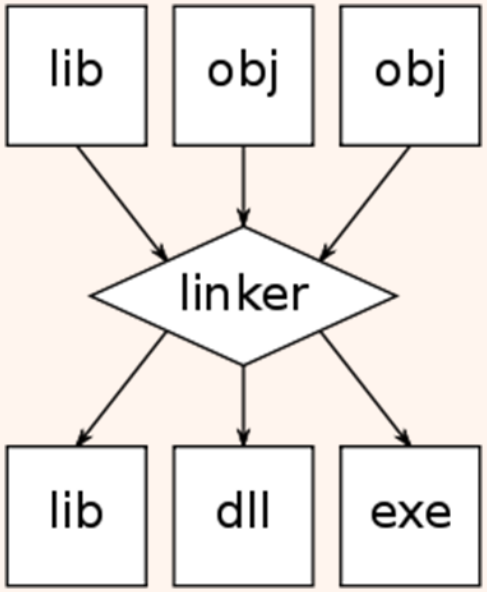
- به طور خلاصه object file شامل بخش‌های زیر است:
  - یک سرآیند(header): شامل اندازه و موقعیت سایر بخش‌ها
  - بخش دستورها: کد دستورالعمل‌ها(برخی عملوندها کامل نشده است)
  - بخش داده‌ها: نمایش دودویی داده‌ها مورد استفاده
  - Realocation table: شامل اطلاعاتی در مورد عملوندهایی که به آدرس مستقیم احتیاج دارند.
  - جدول نمادها (symbol table) با دستور nm می‌توان لیست نمادها را نمایش داد.

```
ahmad@ubuntu:~/MyData/courses/Asm/92_1/chaintools$ nm hello.o
00000000 T main
         U puts
```



لینکر

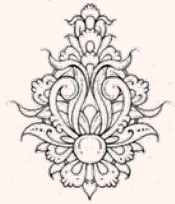
- پیونده‌ده یک برنامه‌ی سیستمی است که برنامه‌هایی که به زبان ماشین تبدیل شده‌اند را با هم ترکیب کرده، آدرس برچسب‌ها را مشخص می‌کند و در نهایت یک فایل اجرایی تولید می‌کند، این برنامه سرآیند فایل اجرایی شامل طول بخش‌های مختلف، محل قرار گیری در حافظه، کتابخانه‌های پویای مورد استفاده و ... را تکمیل می‌کند.
- بدین‌ترتیب می‌توان برنامه‌ها را جداگانه کامپایل و اسمبل کرد.



مثال (windows)

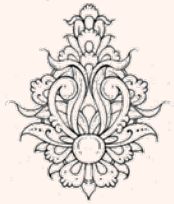
لینک کردن پویا در برابر ایستا

- برای این که مشکل پیش آمده برطرف شود، باید کتابخانه‌های توابع مورد استفاده را در اختیار لینکر قرار داد.
- این کار به دو شیوه صورت می‌پذیرد:
  - ایستا (static linking)
    با این کار توابع به برنامه افزوده می‌شوند، بدین‌ترتیب حجم فایل اجرایی به شدت افزایش می‌یابد.
  - پویا (dynamic linking)
    توابع مورد استفاده در زمان اجرا فراخوانی می‌شوند.



لینک کردن پویا در برابر ایستا

- در کتابخانه‌های ایستا، در صورت تغییر کتابخانه، باید فرآیند لینک دوباره انجام شود، اما در استفاده از کتابخانه‌های پویا در صورت تغییر توابع کتابخانه‌ای نیازی به تغییر در فایل‌های اجرایی نیست.
- در صورت استفاده کتابخانه‌های ایستا، فایل اجرایی به تنهای قابل استفاده خواهد بود.



لینک کردن پویا در برابر ایستا

- gcc به صورت پیش‌فرض از کتابخانه پویا استفاده می‌کند، در صورتی که نیاز به لینک ایستا باشد می‌توان از پارامتر –static استفاده کرد.

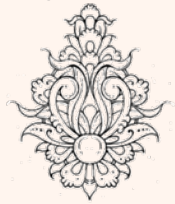
```
ahmad@ubuntu:~/MyData/courses/Asm/92_2/chaintools$ as hello_asm.s -o hello_asm.o
ahmad@ubuntu:~/MyData/courses/Asm/92_2/chaintools$ ld   hello_asm.o -o  hello_asm
ahmad@ubuntu:~/MyData/courses/Asm/92_2/chaintools$ gcc hello.c -o sfile -static
ahmad@ubuntu:~/MyData/courses/Asm/92_2/chaintools$ gcc hello.c -o dfile
ahmad@ubuntu:~/MyData/courses/Asm/92_2/chaintools$ ls -l
total 592
-rwxr-xr-x 1 ahmad ahmad   7177 2014-04-18 11:01 dfile
-rwxr-xr-x 1 ahmad ahmad    615 2014-04-18 11:00 hello_asm
-rw-r--r-- 1 ahmad ahmad    592 2014-04-18 11:00 hello_asm.o
-rw-r--r-- 1 ahmad ahmad    237 2014-04-18 10:48 hello_asm.s
-rw-r--r-- 1 ahmad ahmad     81 2014-04-18 01:02 hello.c
-rwxr-xr-x 1 ahmad ahmad 577982 2014-04-18 11:00 sfile
```

- با دستور ldd می‌توان کتابخانه‌های پویا مورد بررسی را نمایش داد.

```
ahmad@ubuntu:~/MyData/courses/Asm/92_2/chaintools$ ldd sfile
        not a dynamic executable
ahmad@ubuntu:~/MyData/courses/Asm/92_2/chaintools$ ldd dfile
        linux-gate.so.1 =>  (0x00a88000)
        libc.so.6 => /lib/tls/i686/cmov/libc.so.6 (0x00b3e000)
        /lib/ld-linux.so.2 (0x00afb000)
ahmad@ubuntu:~/MyData/courses/Asm/92_2/chaintools$
```

فایل اجرایی

- فایل اجرایی شامل سه بخش است:
  - یک سرآیند که شامل اطلاعات دقیقی از برنامه شامل اندازه‌ی سایر بخش‌ها، جدول نمادها و اطلاعات اشکال‌زدایی و ... است.
  - بخش دستورالعمل
  - بخش داده
- در صورت استفاده از s– (در gcc یا ld) جدول نمادها فایل اجرایی ضمیمه نمی‌شود.

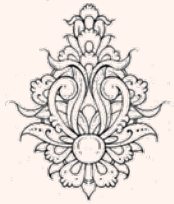


loader

بارکننده

- بخشی از سیستم‌عامل است.

– خواندن سرآیند برای مشخص کردن حجم برنامه، آماده کردن فضای حافظه‌ی برای داده‌ها و دستورها، منتقل کردن محتویات برنامه و داده‌ها به حافظه، آماده‌سازی پشته(مقداردهی اشاره‌گر پشته) و انتقال آرگومان‌ها، آماده‌سازی ثبات‌ها و پرش به آدرس شروع برنامه از وظایف آن است.



مثال

```
#include <stdio.h>
int a;
int b=2;
int main(){
   printf("a=%d\n",a);
   printf("b=%d\n",b);
   return 0;
```

بخش bss

بخش data

```
.globl main
        .type   main, @function
main:
        pushl   %ebp
        movl    %esp, %ebp
        andl    $-16, %esp
        subl    $16, %esp
        movl    a, %edx
        movl    $.LC0, %eax
        movl    %edx, 4(%esp)
        movl    %eax, (%esp)
        call    printf
        movl    b, %edx
        . . .
```

```
        .file   "exm.c"
        .comm   a,4,4
.globl b
        .data
        .align 4
        .type   b, @object
        .size   b, 4
b:
        .long   2
        .section .rodata
.LC0:
        .string "a=%d\n"
.LC1:
        .string "b=%d\n"
```

نشانی‌دهی مستقیم

ادامه‌ی مثال

Object file

```
00000000 <main>:
   0:   55                      push   %ebp
   1:   89 e5                   mov    %esp,%ebp
   3:   83 e4 f0                and    $0xfffffff0,%esp
   6:   83 ec 10                sub    $0x10,%esp
   9:   8b 15 00 00 00 00       mov    0x0,%edx
   f:   b8 00 00 00 00          mov    $0x0,%eax
  14:   89 54 24 04             mov    %edx,0x4(%esp)
  18:   89 04 24                mov    %eax,(%esp)
  1b:   e8 fc ff ff ff          call   1c <main+0x1c>
  20:   8b 15 00 00 00 00       mov    0x0,%edx
  26:   b8 06 00 00 00          mov    $0x6,%eax
  2b:   89 54 24 04             mov    %edx,0x4(%esp)
  2f:   89 04 24                mov    %eax,(%esp)
  32:   e8 fc ff ff ff          call   33 <main+0x33>
  37:   b8 00 00 00 00          mov    $0x0,%eax
  3c:   c9                      leave
  3d:   c3                      ret
```

- آدرس‌های مستقیم بعد از لینک کردن مشخص می‌شوند.

فایل اجرایی

```
080481a4 <main>:
 80481a4:       55                      push   %ebp
 80481a5:       89 e5                   mov    %esp,%ebp
 80481a7:       83 e4 f0                and    $0xfffffff0,%esp
 80481aa:       83 ec 10                sub    $0x10,%esp
 80481ad:       8b 15 a4 92 04 08       mov    0x80492a4,%edx
 80481b3:       b8 e2 81 04 08          mov    $0x80481e2,%eax
 80481b8:       89 54 24 04             mov    %edx,0x4(%esp)
 80481bc:       89 04 24                mov    %eax,(%esp)
 80481bf:       e8 d0 ff ff ff          call   8048194 <printf@plt>
 80481c4:       8b 15 a0 92 04 08       mov    0x80492a0,%edx
 80481ca:       b8 e8 81 04 08          mov    $0x80481e8,%eax
 80481cf:       89 54 24 04             mov    %edx,0x4(%esp)
 80481d3:       89 04 24                mov    %eax,(%esp)
 80481d6:       e8 b9 ff ff ff          call   8048194 <printf@plt>
 80481db:       b8 00 00 00 00          mov    $0x0,%eax
 80481e0:       c9                      leave
 80481e1:       c3                      ret
```