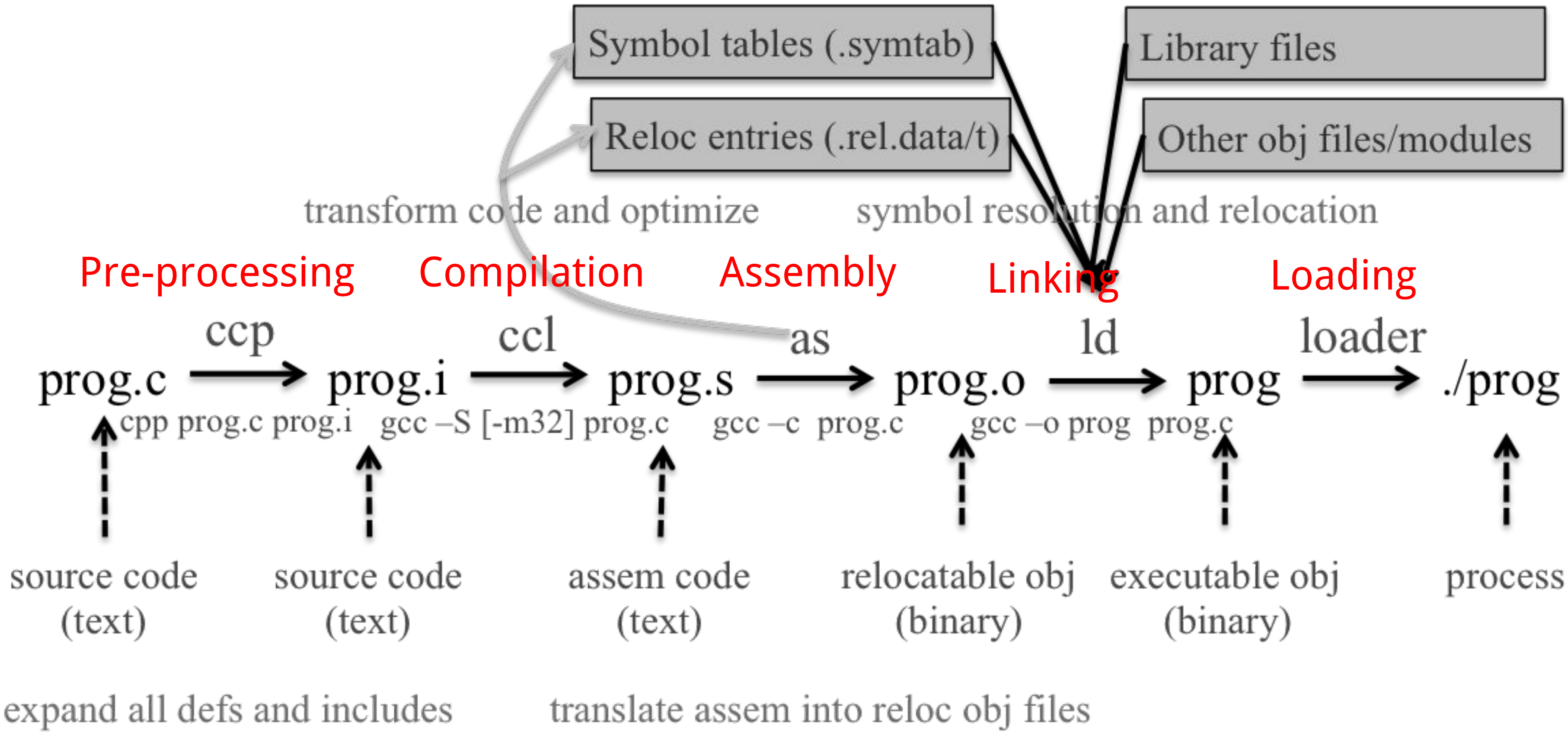
CSE 410/565: Computer Security

Instructor: Dr. Ziming Zhao

Background Knowledge: Compiler, linker, and loader

From a C program to a process



Loading and Executing a Binary Program on Linux

Validation (permissions, memory requirements etc.)

Operating system starts by setting up a new process for the program to run in, including a virtual address space.

The operating system maps an interpreter into the process's virtual memory.

Interpreter, e.g., /lib/ld-linux.so in Linux

The interpreter loads the binary into its virtual address space (the same space in which the interpreter is loaded).

It then parses the binary to find out (among other things) which dynamic libraries the binary uses.

The interpreter maps these into the virtual address space (using *mmap* or an equivalent function) and then performs any necessary last-minute relocations in the binary's code sections to fill in the correct addresses for references to the dynamic libraries.

Compiling a C program behind the scene (add_32 add_64)

add.c

```
#include "add.h"

#define BASE 50

int add(int a, int b)
{ return a + b +
BASE;}
```

add.h

```
#ifndef ADD_H
#define ADD_H

int add(int, int);

#endif
```

main.c

```
/* This program has an integer overflow vulnerability. */
#include "add.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define USAGE "Add two integers with 50. Usage: add a b\n"

int main(int argc, char *argv[])
{
  int a = 0;
  int b = 0;

  if (argc != 3)
   {
     printf(USAGE);
     return 0;}

  a = atoi(argv[1]);
  b = atoi(argv[2]);
  printf("%d + %d + 50 = %d\n", a, b, add(a, b));
}
```

```
gcc -Wall -save-temps -P -m32 -O2 add.c main.c -o add_32
```

```
gcc -Wall -save-temps -P -O2 add.c main.c -o add_64
```

Background Knowledge:
x86 architecture

Data Types

There are 5 integer data types:

Byte – 8 bits.
Word – 16 bits.
Dword, Doubleword – 32 bits.
Quadword – 64 bits.
Double quadword – 128 bits.

Endianness

- Little Endian (Intel, ARM)

Least significant byte has lowest address
Dword address: 0x0
Value: 0x78563412

- Big Endian

Least significant byte has highest address
Dword address: 0x0
Value: 0x12345678

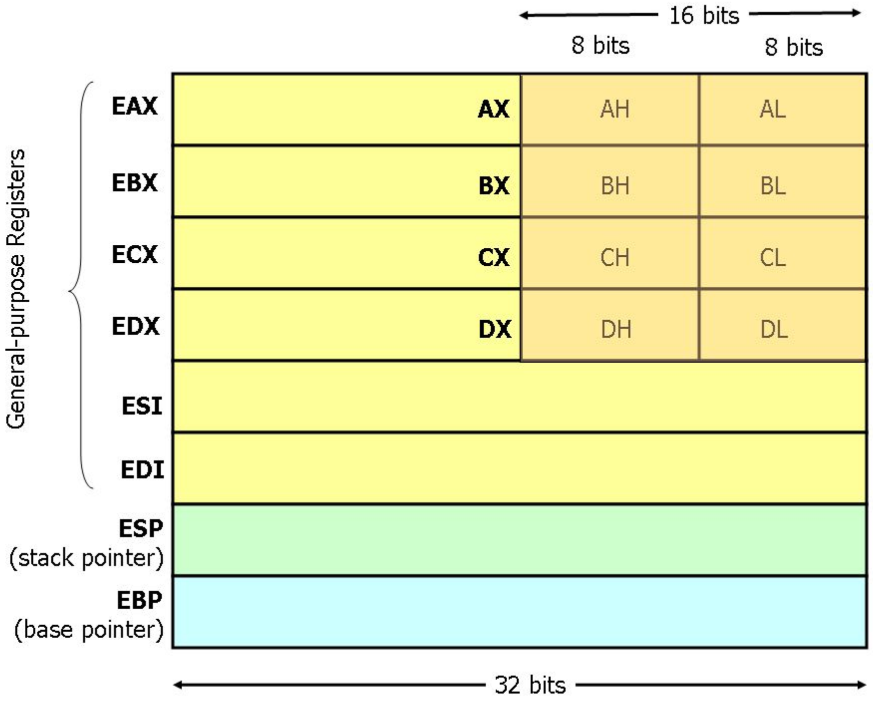
Address 0	0x12
Address 1	0x34
Address 2	0x56
Address 3	0x78

Base Registers

There are

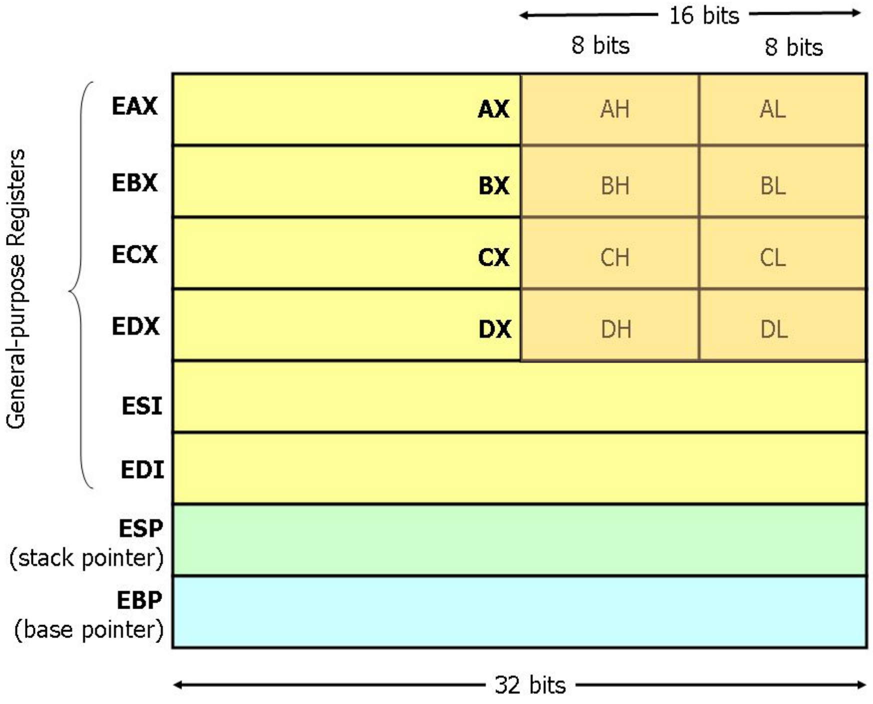
- Eight 32-bit “general-purpose” registers,
- One 32-bit EFLAGS register,
- One 32-bit instruction pointer register (eip), and
- Other special-purpose registers.

The General-Purpose Registers



- 8 general-purpose registers
- esp is the stack pointer
- ebp is the base pointer
- esi and edi are source and destination index registers for array and string operations

The General-Purpose Registers



- The registers eax, ebx, ecx, and edx may be accessed as 32-bit, 16-bit, or 8-bit registers.
- The other four registers can be accessed as 32-bit or 16-bit.

EFLAGS Register

The various bits of the 32-bit EFLAGS register are set (1) or reset/clear (0) according to the results of certain operations.

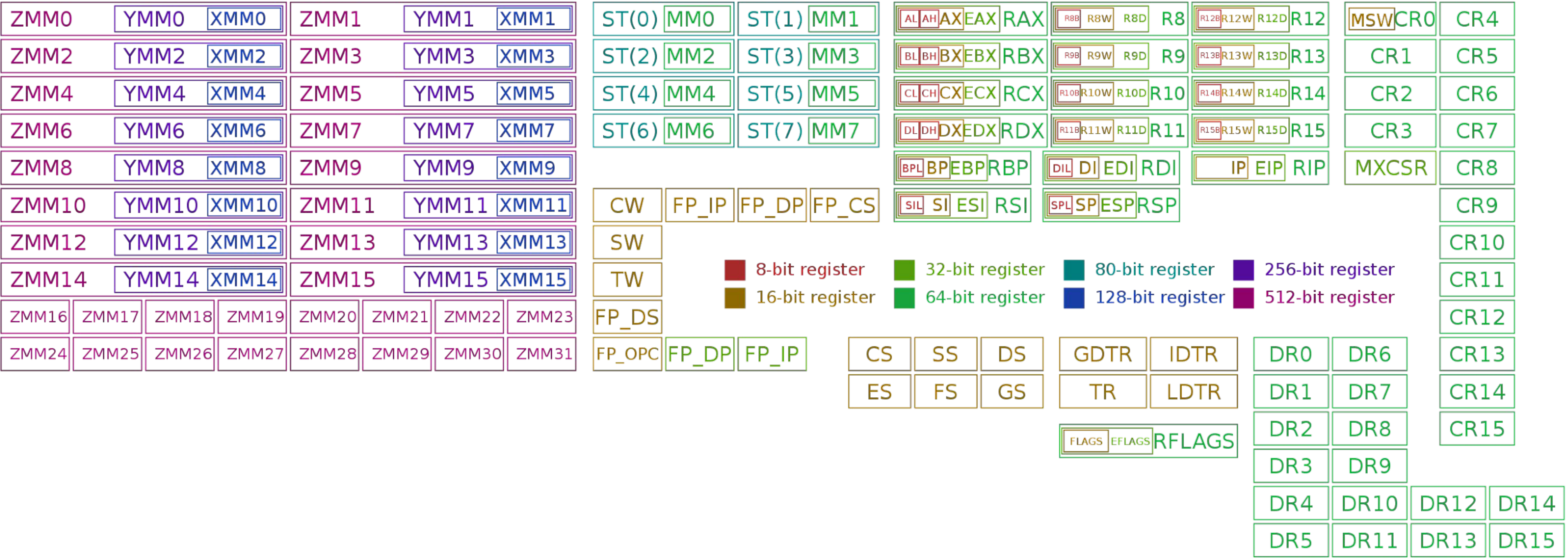
We will be interested in, at most, the bits

CF – carry flag
PF – parity flag
ZF – zero flag
SF – sign flag

Instruction Pointer (EIP)

Finally, there is the EIP register, which is the instruction pointer (program counter). Register EIP holds the address of the **next** instruction to be executed.

Registers on x86 and amd64



https://en.wikipedia.org/wiki/X86

Instructions

Each instruction is of the form

label: mnemonic operand1, operand2, operand3
The label is optional.

The number of operands is 0, 1, 2, or 3, depending on the mnemonic .

Each operand is either
- An immediate value,
- A register, or
- A memory address.

Source and Destination Operands

Each operand is either a source operand or a destination operand.

A source operand, in general, may be

- An immediate value,
- A register, or
- A memory address.

A destination operand, in general, may be

- A register, or
- A memory address.

Instructions

hlt – 0 operands
halts the central processing unit (CPU) until the next external interrupt is fired

inc – 1 operand; inc <reg>, inc <mem>

add – 2 operands; add <reg>,<reg>

imul – 1, 2, or 3 operands; imul <reg32>,<reg32>,<con>

Intel Syntax Assembly and Disassembly

Machine instructions generally fall into three categories: data movement, arithmetic/logic, and control-flow.

<reg32> Any 32-bit register (eax, ebx, ecx, edx, esi, edi, esp, or ebp)
<reg16> Any 16-bit register (ax, bx, cx, or dx)
<reg8> Any 8-bit register (ah, bh, ch, dh, al, bl, cl, or dl)
<reg> Any register
<mem> A memory address (e.g., [eax] or [eax + ebx*4]); [] square brackets
<con32> Any 32-bit immediate
<con16> Any 16-bit immediate
<con8> Any 8-bit immediate
<con> Any 8-, 16-, or 32-bit immediate

Addressing Memory

Move from source (operand 2) to destination (operand 1)

mov [eax], ebx (read as MOVE FROM x to y) Load 4 bytes from the memory address in EBX into EAX.

mov eax, [esi - 4] Move 4 bytes at memory address ESI - 4 into EAX. */

mov [esi + eax * 1], cl Move the contents of CL into the byte at address ESI+EAX*1.

mov edx, [esi + ebx*4] Move the 4 bytes of data at address ESI+4*EBX into EDX.

Addressing Memory

The size directives BYTE PTR, WORD PTR, and DWORD PTR serve this purpose, indicating sizes of 1, 2, and 4 bytes respectively.

mov [ebx], 2 isn't this ambiguous? We can have a default.

mov BYTE PTR [ebx], 2 Move 2 into the single byte at the address stored in EBX.

mov WORD PTR [ebx], 2 Move the 16-bit integer representation of 2 into the 2 bytes starting at the address in EBX.

mov DWORD PTR [ebx], 2 Move the 32-bit integer representation of 2 into the 4 bytes starting at the address in EBX.

Data Movement Instructions

mov — Move

Syntax
mov <reg>, <reg>
mov <reg>, <mem>
mov <mem>, <reg>
mov <reg>, <con>
mov <mem>, <con>

Examples
mov eax, ebx — copy the value in EBX into EAX
mov byte ptr [var], 5 — store the value 5 into the byte at location var

Data Movement Instructions

push — Push on stack; decrements ESP by 4, then places the operand at the location ESP points to.

Syntax
push <reg32>
push <mem>
push <con32>

Examples
push eax — push eax on the stack
push [var] — push the 4 bytes at address var onto the stack

Data Movement Instructions

pop — Pop from stack

Syntax
pop <reg32>
pop <mem>

Examples
pop edi — pop the top element of the stack into EDI.
pop [ebx] — pop the top element of the stack into memory at the four bytes starting at location EBX.

LEA Instructions

lea — Load effective address; used for quick calculation

Syntax
lea <reg32>, <mem>

Examples
Lea edi, [ebx+4*esi] — the quantity EBX+8*ESI is placed in EDI.

Arithmetic and Logic Instructions

add eax, 10 — EAX is set to EAX + 10
addb byte ptr [eax], 10 — add 10 to the single byte stored at memory address stored in EAX

sub al, ah — AL is set to AL - AH
sub eax, 216 — subtract 216 from the value stored in EAX

dec eax — subtract one from the contents of EAX

imul eax, [ebx] — multiply the contents of EAX by the 32-bit contents of the memory at location EBX. Store the result in EAX.

shr ebx, cl — Store in EBX the floor of result of dividing the value of EBX by 2n where n is the value in CL.

Control Flow Instructions

jmp — Jump

Transfers program control flow to the instruction at the memory location indicated by the operand.

Syntax
jmp <label> # direct jump
jmp <reg32> # indirect jump

Example
jmp begin — Jump to the instruction labeled begin.

Control Flow Instructions

jcondition — Conditional jump

Syntax
je <label> (jump when equal)
jne <label> (jump when not equal)
jz <label> (jump when last result was zero)
jg <label> (jump when greater than)
jge <label> (jump when greater than or equal to)
jl <label> (jump when less than)
jle <label> (jump when less than or equal to)

Example

```
cmp ebx, eax
jle done
```

Control Flow Instructions

cmp — Compare

Syntax
cmp <reg>, <reg>
cmp <mem>, <reg>
cmp <reg>, <mem>
cmp <con>, <reg>

Example
cmp byte ptr [ebx], 10
jeq loop

If the byte stored at the memory location in EBX is equal to the integer constant 10, jump to the location labeled loop.

Control Flow Instructions

call — Subroutine call

The call instruction first **pushes the current code location onto the hardware supported stack** in memory, and then performs **an unconditional jump to the code** location indicated by the label operand. *Unlike the simple jump instructions, the call instruction saves the location to return to when the subroutine completes*.

Syntax
call <label>
call <reg32>
Call <mem>

Control Flow Instructions

ret — Subroutine return

The ret instruction implements a subroutine return mechanism. This instruction pops a code location off the hardware supported in-memory stack to the program counter.

Syntax
ret

The Run-time Stack

The run-time stack supports procedure calls and the passing of parameters between procedures.

The stack is located in memory.

The stack grows towards **low memory**.

When we push a value, esp is decremented.

When we pop a value, esp is incremented.

Stack Instructions

enter — Create a function frame

Equivalent to:

```
push ebp
mov ebp, esp
sub esp, Imm
```

Stack Instructions

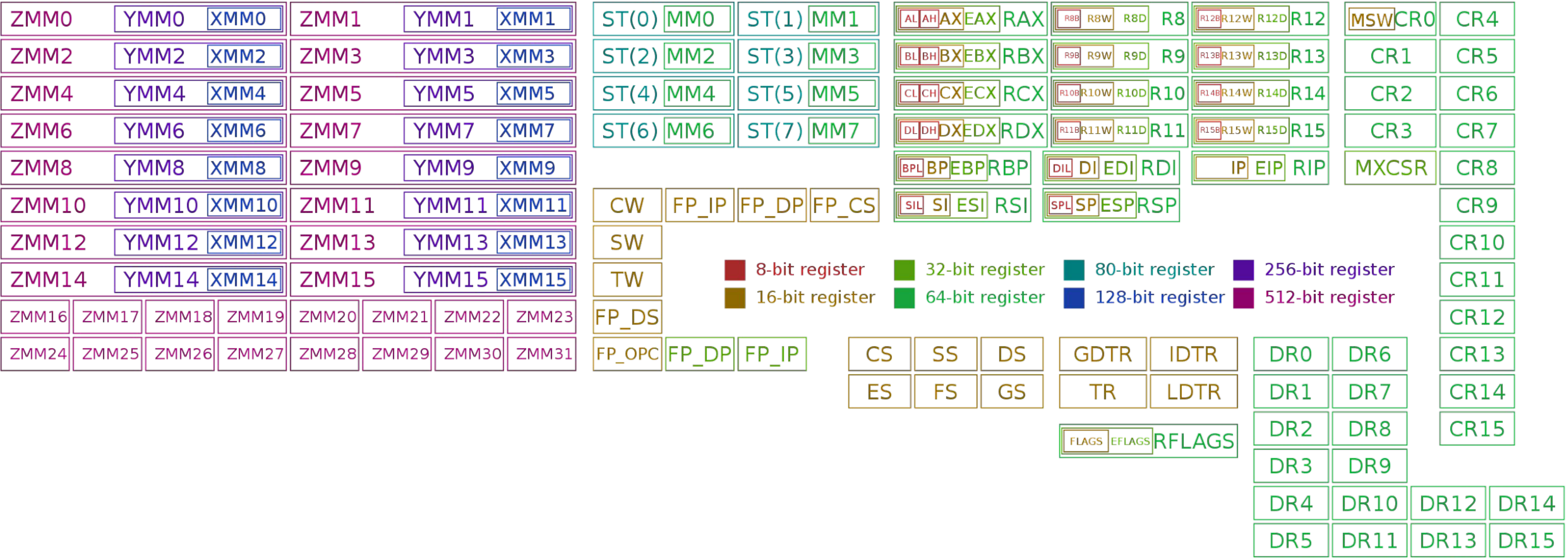
leave — Releases the function frame set up by an earlier ENTER instruction.

Equivalent to:

```
mov esp, ebp
pop ebp
```

Background Knowledge: amd64 architecture

Registers on x86 and x86-64



https://en.wikipedia.org/wiki/X86

x86 vs. x86-64 (code/ladd)

main.c

```
/*
This program has an integer overflow vulnerability.
 */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

long long ladd(long long *xp, long long y)
{
  long long t = *xp + y;
  return t;
}
```

```
int main(int argc, char *argv[])
{
  long long a = 0;
  long long b = 0;

  if (argc != 3)
    {
      printf("Usage: ladd a b\n");
      return 0;
    }

  printf("The sizeof(long long) is %d\n", sizeof(long long));

  a = atoll(argv[1]);
  b = atoll(argv[2]);

  printf("%lld + %lld = %lld\n", a, b, ladd(&a, b));
}
```

```
gcc -Wall -m32 -O2 main.c -o ladd
```

```
gcc -Wall -O2 main.c -o ladd64
```

x86 vs. x86-64 (code/ladd)

x86

```
000012c0 <ladd>:
    12c0:       f3 0f 1e fb             endbr32
    12c4:       8b 44 24 04             mov    eax,DWORD PTR [esp+0x4]
    12c8:       8b 50 04                mov    edx,DWORD PTR [eax+0x4]
    12cb:       8b 00                   mov    eax,DWORD PTR [eax]
    12cd:       03 44 24 08             add    eax,DWORD PTR [esp+0x8]
    12d1:       13 54 24 0c             adc    edx,DWORD PTR [esp+0xc]
    12d5:       c3                      ret
```

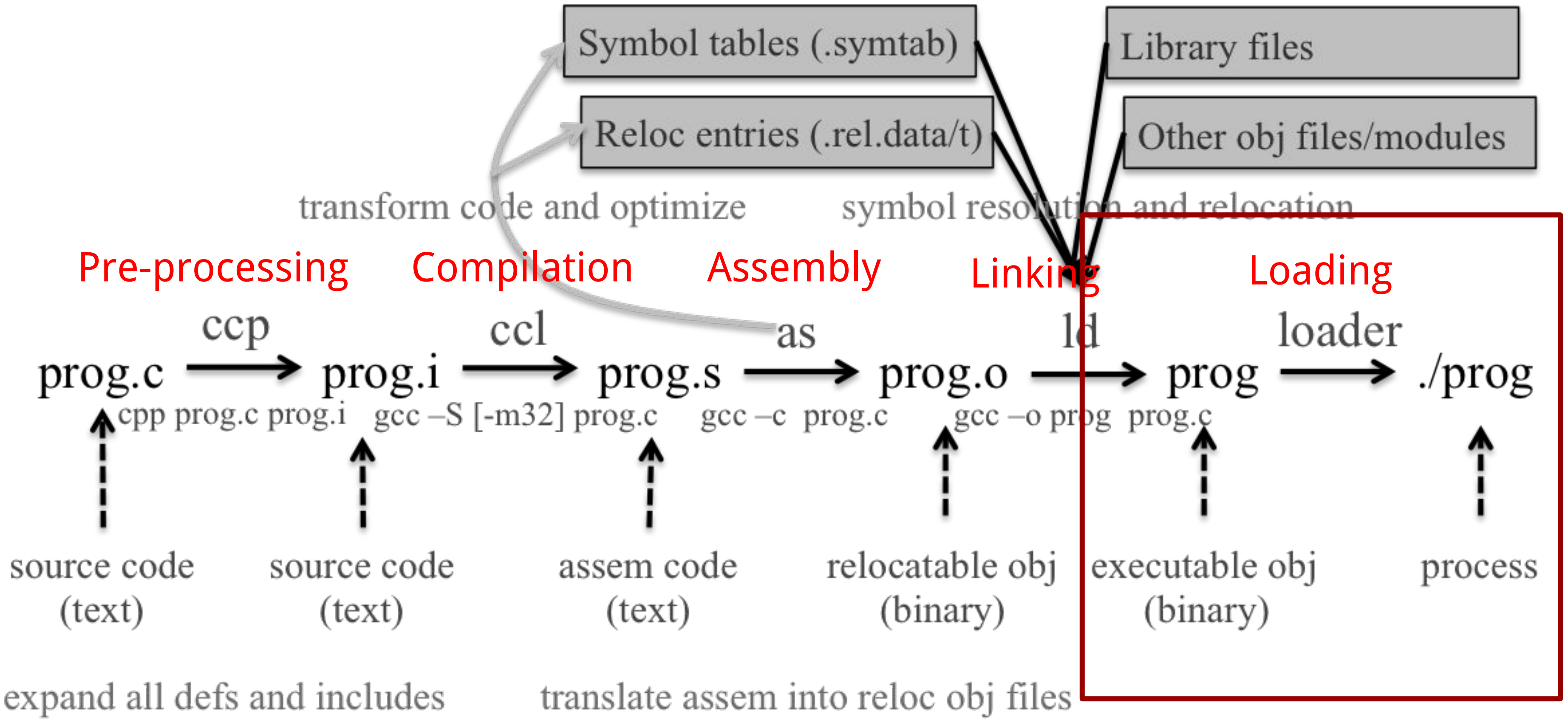
x86-64

```
0000000000001220 <ladd>:
    1220:       f3 0f 1e fa             endbr64
    1224:       48 8b 07                mov    rax,QWORD PTR [rdi]
    1227:       48 01 f0                add    rax,rsi
    122a:       c3                      ret
```

```
objdump -M intel -d ladd_32
objdump -M intel -d ladd_64
```

Background Knowledge:
Set-UID Programs

From a C program to a process



Real UID, Effective UID, and Saved UID

Each Linux/Unix **process** has 3 UIDs associated with it.

Real UID (RUID): This is the UID of the user/process that created THIS process. It can be changed only if the running process has EUID=0.

Effective UID (EUID): This UID is used to evaluate privileges of the process to perform a particular action. EUID can be changed either to RUID, or SUID if EUID!=0. If EUID=0, it can be changed to anything.

Saved UID (SUID): If the binary image file, that was launched has a Set-UID bit on, SUID will be the UID of the owner of the file. Otherwise, SUID will be the RUID.

Set-UID Program

The kernel makes the decision whether a process has the privilege by looking on the **EUID** of the process.

For non Set-UID programs, the effective uid and the real uid are the same. For Set-UID programs, **the effective uid is the owner of the program**, while the real uid is the user of the program.

What will happen is when a setuid binary executes, the process changes its Effective User ID (EUID) from the default RUID to the owner of this special binary executable file which in this case is - root.

```
ziming@ziming-ThinkPad:~$ ls -al /bin/
total 12676
drwxr-xr-x  2 root root    4096 May 26 00:14 .
drwxr-xr-x 26 root root    4096 May 18 09:57 ..
-rwxr-xr-x  1 root root 1113504 Jun  6  2019 bash
-rwxr-xr-x  1 root root  748968 Aug 29  2018 brltty
-rwxr-xr-x  3 root root   34888 Jul  4  2019 bunzip2
-rwxr-xr-x  1 root root 2062296 Mar  6  2019 busybox
-rwxr-xr-x  3 root root   34888 Jul  4  2019 bzcat
lrwxrwxrwx  1 root root       6 Jul  4  2019 bzcmp -> bzdiff
-rwxr-xr-x  1 root root    2140 Jul  4  2019 bzdiff
lrwxrwxrwx  1 root root       6 Jul  4  2019 bzegrep ->
-rwxr-xr-x  1 root root    4877 Jul  4  2019 bzexe
lrwxrwxrwx  1 root root       6 Jul  4  2019 bzfgrep ->
-rwxr-xr-x  1 root root    3642 Jul  4  2019 bzgrep
-rwxr-xr-x  3 root root   34888 Jul  4  2019 bzip2
-rwxr-xr-x  1 root root   14328 Jul  4  2019 bzip2recov
lrwxrwxrwx  1 root root       6 Jul  4  2019 bzless ->
-rwxr-xr-x  1 root root    1297 Jul  4  2019 bzmore
-rwxr-xr-x  1 root root   35064 Jan 18  2018 cat
-rwxr-xr-x  1 root root   14328 Apr 21  2017 chacl
-rwxr-xr-x  1 root root   63672 Jan 18  2018 chgrp
-rwxr-xr-x  1 root root   59608 Jan 18  2018 chmod
-rwxr-xr-x  1 root root   67768 Jan 18  2018 chown
-rwxr-xr-x  1 root root   10312 Jan 22  2018 chvt
-rwxr-xr-x  1 root root  141528 Jan 18  2018 cp
-rwxr-xr-x  1 root root  157224 Nov  5  2019 cpio
-rwxr-xr-x  1 root root  121432 Jan 25  2018 dash
-rwxr-xr-x  1 root root  100568 Jan 18  2018 date
-rwxr-xr-x  1 root root   76000 Jan 18  2018 dd
-rwxr-xr-x  1 root root   84776 Jan 18  2018 df
-rwxr-xr-x  1 root root  133792 Jan 18  2018 dir
-rwxr-xr-x  1 root root   72000 Mar  5 12:23 dmesg
```

```
-rwxr-xr-x  1 root root   39103 Apr 23  2019 setupcon
lrwxrwxrwx  1 root root       4 Aug 16  2018 sh -> dash
lrwxrwxrwx  1 root root       4 Aug 16  2018 sh.distrib -> dash
-rwxr-xr-x  1 root root   35000 Jan 18  2018 sleep
-rwxr-xr-x  1 root root  139904 May 11 10:40 ss
lrwxrwxrwx  1 root root       7 Mar  6  2019 static-sh -> busybox
-rwxr-xr-x  1 root root   75992 Jan 18  2018 stty
-rwsr-xr-x  1 root root   44664 Mar 22  2019 su
-rwxr-xr-x  1 root root   35000 Jan 18  2018 sync
-rwxr-xr-x  1 root root  182352 May  3 07:30 systemctl
lrwxrwxrwx  1 root root      20 May  3 07:30 systemd -> /lib/systemd/systemd
-rwxr-xr-x  1 root root   10320 May  3 07:30 systemd-ask-password
-rwxr-xr-x  1 root root   14400 May  3 07:30 systemd-escape
-rwxr-xr-x  1 root root   84328 May  3 07:30 systemd-hwdb
-rwxr-xr-x  1 root root   14416 May  3 07:30 systemd-inhibit
-rwxr-xr-x  1 root root   18496 May  3 07:30 systemd-machine-id-setup
-rwxr-xr-x  1 root root   14408 May  3 07:30 systemd-notify
-rwxr-xr-x  1 root root   43080 May  3 07:30 systemd-sysusers
-rwxr-xr-x  1 root root   71752 May  3 07:30 systemd-tmpfiles
-rwxr-xr-x  1 root root   26696 May  3 07:30 systemd-tty-ask-password-agent
-rwxr-xr-x  1 root root  423312 Jan 21  2019 tar
-rwxr-xr-x  1 root root   10104 Dec 30  2017 tempfile
-rwxr-xr-x  1 root root   88280 Jan 18  2018 touch
-rwxr-xr-x  1 root root   30904 Jan 18  2018 true
-rwxr-xr-x  1 root root  584072 May  3 07:30 udevadm
-rwxr-xr-x  1 root root   14328 Aug 11  2016 ulockmgr_server
-rwsr-xr-x  1 root root   26696 Mar  5 12:23 umount
-rwxr-xr-x  1 root root   35032 Jan 18  2018 uname
```

```
-rwxr-xr-x  1 root root   39103 Apr 23  2019 setupcon
lrwxrwxrwx  1 root root       4 Aug 16  2018 sh -> dash
lrwxrwxrwx  1 root root       4 Aug 16  2018 sh.distrib -> dash
-rwxr-xr-x  1 root root   35000 Jan 18  2018 sleep
-rwxr-xr-x  1 root root  139904 May 11 10:40 ss
lrwxrwxrwx  1 root root       7 Mar  6  2019 static-sh -> busybox
-rwxr-xr-x  1 root root   75992 Jan 18  2018 stty
-rwsr-xr-x  1 root root   44664 Mar 22  2019 su
-rwxr-xr-x  1 root root   35000 Jan 18  2018 sync
-rwxr-xr-x  1 root root  182352 May  3 07:30 systemctl
lrwxrwxrwx  1 root root      20 May  3 07:30 systemd -> /lib/systemd/systemd
-rwxr-xr-x  1 root root   10320 May  3 07:30 systemd-ask-password
-rwxr-xr-x  1 root root   14400 May  3 07:30 systemd-escape
-rwxr-xr-x  1 root root   84328 May  3 07:30 systemd-hwdb
-rwxr-xr-x  1 root root   14416 May  3 07:30 systemd-inhibit
-rwxr-xr-x  1 root root   18496 May  3 07:30 systemd-machine-id-setup
-rwxr-xr-x  1 root root   14408 May  3 07:30 systemd-notify
-rwxr-xr-x  1 root root   43080 May  3 07:30 systemd-sysusers
-rwxr-xr-x  1 root root   71752 May  3 07:30 systemd-tmpfiles
-rwxr-xr-x  1 root root   26696 May  3 07:30 systemd-tty-ask-password-agent
-rwxr-xr-x  1 root root  423312 Jan 21  2019 tar
-rwxr-xr-x  1 root root   10104 Dec 30  2017 tempfile
-rwxr-xr-x  1 root root   88280 Jan 18  2018 touch
-rwxr-xr-x  1 root root   30904 Jan 18  2018 true
-rwxr-xr-x  1 root root  584072 May  3 07:30 udevadm
-rwxr-xr-x  1 root root   14328 Aug 11  2016 ulockmgr_server
-rwsr-xr-x  1 root root   26696 Mar  5 12:23 umount
-rwxr-xr-x  1 root root   35032 Jan 18  2018 uname
```

Example: rdsecret

main.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <pwd.h>

int main(int argc, char *argv[])
{
   FILE *fp = NULL;
   char buffer[100] = {0};

   // get ruid and euid
   uid_t uid = getuid();
   struct passwd *pw = getpwuid(uid);
   if (pw)
   {
          printf("UID: %d, USER: %s.\n", uid, pw->pw_name);
   }

   uid_t euid = geteuid();
   pw = getpwuid(euid);
```

```
 if (pw)
   {
          printf("EUID: %d, EUSER: %s.\n", euid, pw->pw_name);
   }

   print_flag();

   return(0);
}


void print_flag()
{
          FILE *fp;
          char buff[MAX_FLAG_SIZE];
          fp = fopen("flag","r");
          fread(buff, MAX_FLAG_SIZE, 1, fp);
          printf("flag is : %s\n", buff);
          fclose(fp);
}
```

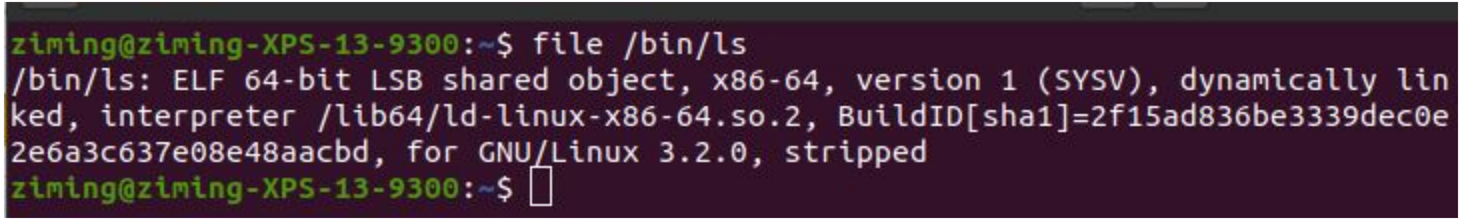
Background Knowledge:
ELF Binary Files

ELF Files

The **Executable** and **Linkable Format** (**ELF**) is a common standard file format for *executable files*, *object code*, *shared libraries*, and *core dumps*. Filename extension *none*, *.axf*, *.bin*, *.elf*, *.o*, *.prx*, *.puff*, *.ko*, *.mod* and *.so*

Contains the program and its data. Describes how the program should be loaded (program/segment headers). Contains metadata describing program components (section headers).

Command *file*




```
file /bin/ls
```

```
ziming@ziming-XPS-13-9300:~$ readelf -a /bin/ls
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              DYN (Shared object file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x67d0
  Start of program headers:          64 (bytes into file)
  Start of section headers:          140224 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         13
  Size of section headers:           64 (bytes)
  Number of section headers:         30
  Section header string table index: 29

Section Headers:
  [Nr] Name              Type             Address           Offset
       Size              EntSize          Flags  Link  Info  Align
  [ 0]                   NULL             0000000000000000  00000000
       0000000000000000  0000000000000000           0     0     0
  [ 1] .interp           PROGBITS         0000000000000318  00000318
       000000000000001c  0000000000000000   A       0     0     1
  [ 2] .note.gnu.propert NOTE             0000000000000338  00000338
       0000000000000020  0000000000000000   A       0     0     8
  [ 3] .note.gnu.build-i NOTE             0000000000000358  00000358
       0000000000000024  0000000000000000   A       0     0     4
  [ 4] .note.ABI-tag     NOTE             000000000000037c  0000037c
       0000000000000020  0000000000000000   A       0     0     4
  [ 5] .gnu.hash         GNU_HASH         00000000000003a0  000003a0
       00000000000000e4  0000000000000000   A       6     0     8
  [ 6] .dynsym           DYNSYM           0000000000000488  00000488
       0000000000000d08  0000000000000018   A       7     1     8
  [ 7] .dynstr           STRTAB           0000000000001190  00001190
       000000000000064c  0000000000000000   A       0     0     1
  [ 8] .gnu.version      VERSYM           00000000000017dc  000017dc
       0000000000000116  0000000000000002   A       6     0     2
  [ 9] .gnu.version_r    VERNEED          00000000000018f8  000018f8
       0000000000000070  0000000000000000   A       7     1     8
  [10] .rela.dyn         RELA             0000000000001968  00001968
       0000000000001350  0000000000000018   A       6     0     8
  [11] .rela.plt         RELA             0000000000002cb8  00002cb8
       00000000000009f0  0000000000000018  AI       6    25     8
  [12] .init             PROGBITS         0000000000004000  00004000
       000000000000001b  0000000000000000  AX       0     0     4
  [13] .plt              PROGBITS         0000000000004020  00004020
       00000000000006b0  0000000000000010  AX       0     0     16
```

INTERP: defines the library that should be used to load this ELF into memory.
LOAD: defines a part of the file that should be loaded into memory.

Sections:
.text: the executable code of your program.
.plt and **.got:** used to resolve and dispatch library calls.
.data: used for pre-initialized global writable data (such as global arrays with initial values)
.rodata: used for global read-only data (such as string constants)
.bss: used for uninitialized global writable data (such as global arrays without initial values)

Tools for ELF

gcc to make your ELF.
readelf to parse the ELF header.
objdump to parse the ELF header and disassemble the source code.
nm to view your ELF's symbols.
patchelf to change some ELF properties.
objcopy to swap out ELF sections.
strip to remove otherwise-helpful information (such as symbols).
kaitai struct (https://ide.kaitai.io/) to look through your ELF interactively.

Background Knowledge:
Memory Map of a Linux Process

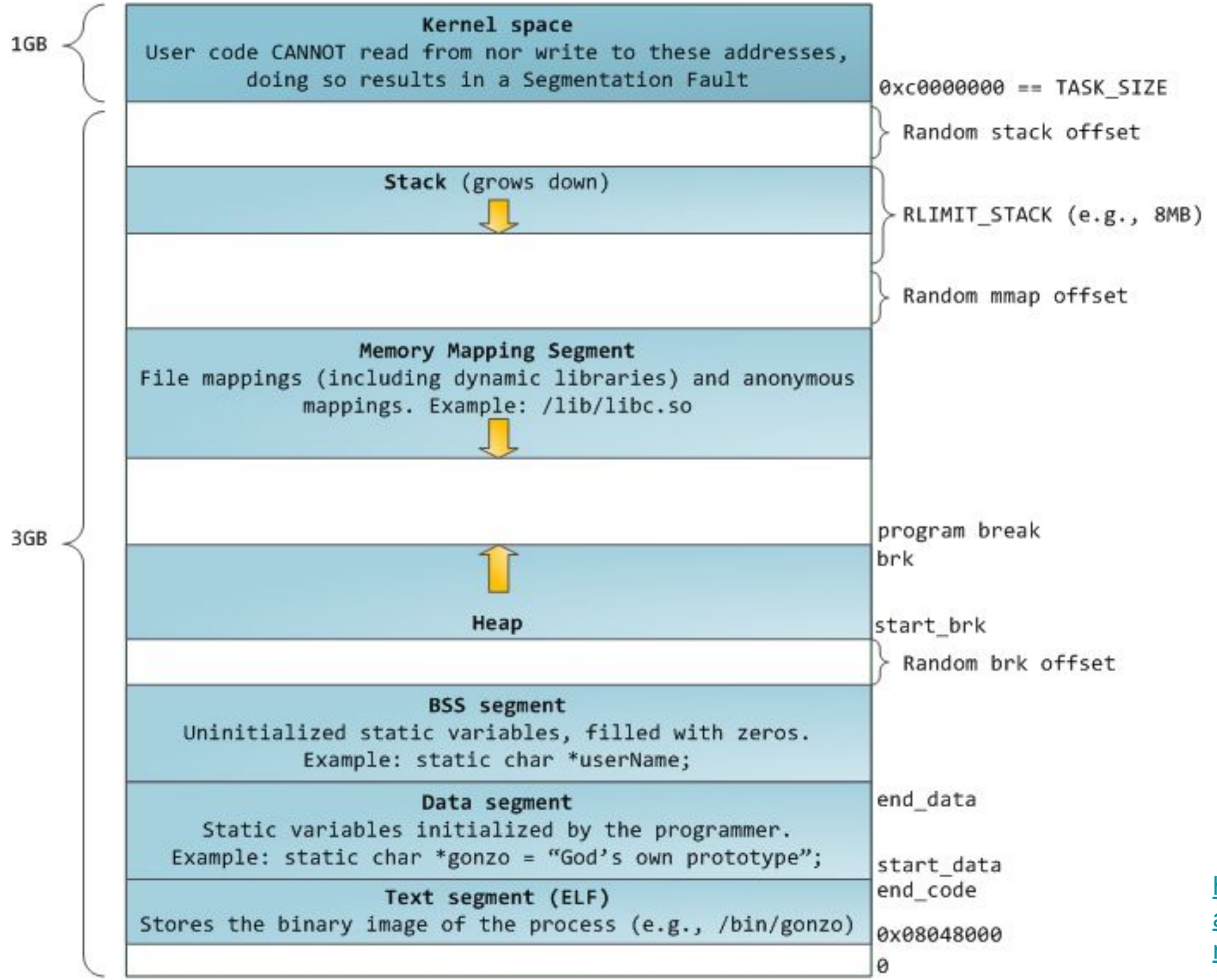
Memory Map of Linux Process (32 bit)

Each process in a multi-tasking OS runs in its own memory sandbox.

This sandbox is the **virtual address space**, which in 32-bit mode is **always a 4GB block of memory addresses**.

These virtual addresses are mapped to physical memory by **page tables**, which are maintained by the operating system kernel and consulted by the processor.

Memory Map of Linux Process (32 bit system)



https://manybutfinite.com/pos
anatomy-of-a-program-in-me
mory/

NULL Pointer in C/C++

```
int * pInt = NULL;
```

In possible definitions of NULL in C/C++:

```
#define NULL ((char *)0)
#define NULL 0

//since C++11
#define NULL nullptr
```

/proc/pid_of_process/maps

Example processmap.c

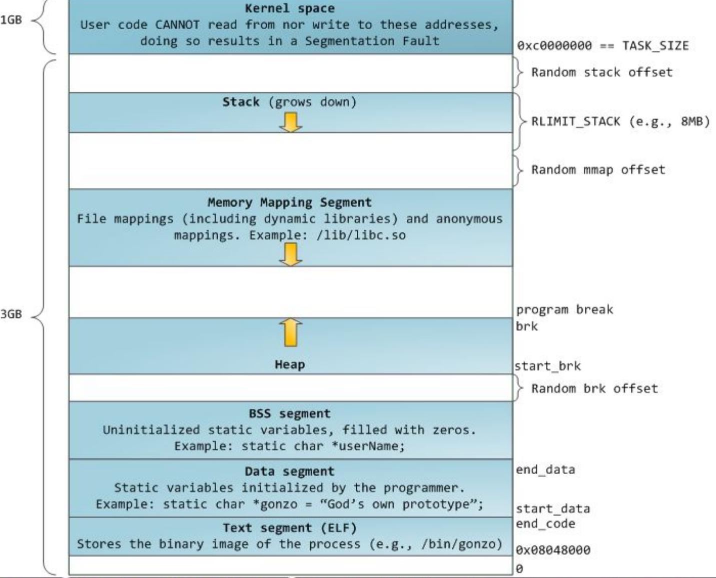
```
#include <stdio.h>
#include <stdlib.h>

int main()
{
		getchar();
		return 0;
}
```

cat /proc/pid/maps

pmap -X pid

pmap -X `pidof pm`




```
ziming@ziming-ThinkPad:~/Dropbox/myTeaching/System Security - Attack and Defense for Binaries UB 2020/code/processmap$ pmap -X 21732
21732:   ./pm
 Address Perm   Offset Device    Inode Size Rss Pss Referenced Anonymous LazyFree ShmemPmdMapped Shared_Hugetlb Private_Hugetlb Swap SwapPss Locked Mapping
56569000 r-xp 00000000 103:02 28575310    4   4   4          4         0        0              0              0               0    0       0      0 pm
5656a000 r--p 00000000 103:02 28575310    4   4   4          4         4        0              0              0               0    0       0      0 pm
5656b000 rw-p 00001000 103:02 28575310    4   4   4          4         4        0              0              0               0    0       0      0 pm
57cf2000 rw-p 00000000  00:00        0  136   4   4          4         4        0              0              0               0    0       0      0 [heap]
f7d73000 r-xp 00000000 103:02  2883591 1876 772 772        772         0        0              0              0               0    0       0      0 libc-2.27.so
f7f48000 ---p 001d5000 103:02  2883591    4   0   0          0         0        0              0              0               0    0       0      0 libc-2.27.so
f7f49000 r--p 001d5000 103:02  2883591    8   8   8          8         8        0              0              0               0    0       0      0 libc-2.27.so
f7f4b000 rw-p 001d7000 103:02  2883591    4   4   4          4         4        0              0              0               0    0       0      0 libc-2.27.so
f7f4c000 rw-p 00000000  00:00        0   12   8   8          8         8        0              0              0               0    0       0      0
f7f75000 rw-p 00000000  00:00        0    8   8   8          8         8        0              0              0               0    0       0      0
f7f77000 r--p 00000000  00:00        0   12   0   0          0         0        0              0              0               0    0       0      0 [vvar]
f7f7a000 r-xp 00000000  00:00        0    8   8   8          8         0        0              0              0               0    0       0      0 [vdso]
f7f7c000 r-xp 00000000 103:02  2883587  152 144 144        144         0        0              0              0               0    0       0      0 ld-2.27.so
f7fa2000 r--p 00025000 103:02  2883587    4   4   4          4         4        0              0              0               0    0       0      0 ld-2.27.so
f7fa3000 rw-p 00026000 103:02  2883587    4   4   4          4         4        0              0              0               0    0       0      0 ld-2.27.so
ffef3000 rw-p 00000000  00:00        0  132  12  12         12        12        0              0              0               0    0       0      0 [stack]
                                       ==== === === ========== ========= ======== ============== ============== =============== ==== ======= ======
                                       2372 988 988        988        60        0              0              0               0    0       0      0 KB
```

Memory Map of Linux Process (64 bit system)

```
ziming@ziming-ThinkPad:~/Dropbox/myTeaching/System Security - Attack and Defense for Binaries UB 2020/code/processmap$ pmap -X 22891
22891:   ./pm64
         Address Perm   Offset Device    Inode Size  Rss Pss Referenced Anonymous LazyFree ShmemPmdMapped Shared_Hugetlb Private_Hugetlb Swap SwapPss Locked Mapping
    55bf7ae37000 r-xp 00000000 103:02 28577490    4    4   4          4         0        0              0              0               0    0       0      0 pm64
    55bf7b037000 r--p 00000000 103:02 28577490    4    4   4          4         4        0              0              0               0    0       0      0 pm64
    55bf7b038000 rw-p 00001000 103:02 28577490    4    4   4          4         4        0              0              0               0    0       0      0 pm64
    55bf7cc0c000 rw-p 00000000  00:00        0  132    4   4          4         4        0              0              0               0    0       0      0 [heap]
    7fc7ebdb6000 r-xp 00000000 103:02   660090 1948  992   5        992         0        0              0              0               0    0       0      0 libc-2.27.so
    7fc7ebf9d000 ---p 001e7000 103:02   660090 2048    0   0          0         0        0              0              0               0    0       0      0 libc-2.27.so
    7fc7ec19d000 r--p 001e7000 103:02   660090   16   16  16         16        16        0              0              0               0    0       0      0 libc-2.27.so
    7fc7ec1a1000 rw-p 001eb000 103:02   660090    8    8   8          8         8        0              0              0               0    0       0      0 libc-2.27.so
    7fc7ec1a3000 rw-p 00000000  00:00        0   16   12  12         12        12        0              0              0               0    0       0      0
    7fc7ec1a7000 r-xp 00000000 103:02   660062  156  156   0        156         0        0              0              0               0    0       0      0 ld-2.27.so
    7fc7ec3a6000 rw-p 00000000  00:00        0    8    8   8          8         8        0              0              0               0    0       0      0
    7fc7ec3ce000 r--p 00027000 103:02   660062    4    4   4          4         4        0              0              0               0    0       0      0 ld-2.27.so
    7fc7ec3cf000 rw-p 00028000 103:02   660062    4    4   4          4         4        0              0              0               0    0       0      0 ld-2.27.so
    7fc7ec3d0000 rw-p 00000000  00:00        0    4    4   4          4         4        0              0              0               0    0       0      0
    7ffe05803000 rw-p 00000000  00:00        0  132   12  12         12        12        0              0              0               0    0       0      0 [stack]
    7ffe058b9000 r--p 00000000  00:00        0   12    0   0          0         0        0              0              0               0    0       0      0 [vvar]
    7ffe058bc000 r-xp 00000000  00:00        0    8    4   0          4         0        0              0              0               0    0       0      0 [vdso]
ffffffffff600000 r-xp 00000000  00:00        0    4    0   0          0         0        0              0              0               0    0       0      0 [vsyscall]
                                               ==== ==== === ========== ========= ======== ============== ============== =============== ==== ======= ======
                                               4512 1236  89       1236        80        0              0              0               0    0       0      0 KB
```