Machine Language
Goals of this Lecture

Help you to learn about:

• x86-64 machine language (in general)
• The assembly and linking processes

Why?

• Last stop on the “language levels” tour
• A power programmer knows the relationship between assembly and machine languages
• A systems programmer knows how an assembler translates assembly language code to machine language code
The Build Process

- **Preprocess**
  - `mypgm.c` → `mypgm.i`

- **Compile**
  - `mypgm.i` → `mypgm.s`

- **Assemble**
  - `mypgm.s` → `mypgm.o`

- **Link**
  - `mypgm.o` → `mypgm`
  - `libc.a` → `mypgm`

Covered in COS 320: Compiling Techniques

Covered here
CISC and RISC

x86-64 machine language instructions are complex

x86-64 is a
  • Complex Instruction Set Computer (CISC)

Alternative:
  • Reduced Instruction Set Computer (RISC)
## CISC and RISC Characteristics

<table>
<thead>
<tr>
<th>CISC</th>
<th>RISC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Many instructions</td>
<td>Few instructions</td>
</tr>
<tr>
<td>Many memory addressing modes (direct, indirect, base+displacement, indexed, scaled indexed)</td>
<td>Few memory addressing modes (typically only direct and indirect)</td>
</tr>
<tr>
<td>Hardware interpretation is complex</td>
<td>Hardware interpretation is simple</td>
</tr>
<tr>
<td>Need relatively few instructions to accomplish a given job (expressive)</td>
<td>Need relatively many instructions to accomplish a given job (not expressive)</td>
</tr>
<tr>
<td>Example: x86-64</td>
<td>Examples: MIPS, SPARC</td>
</tr>
</tbody>
</table>
CISC and RISC History

Stage 1: Programmers compose assembly language
- Important that assembly/machine language be expressive
- CISC dominated (esp. Intel)

Stage 2: Programmers compose high-level language
- Not important that assembly/machine language be expressive; the compiler generates it
- Important that compilers work well => assembly/machine language should be simple
- RISC took a foothold (but CISC, esp. Intel, persists)

Stage 3: Compilers get smarter
- Less important that assembly/machine language be simple
- Hardware is plentiful, enabling complex implementations
- Much motivation for RISC disappears
- CISC (esp. Intel) dominates the computing world
x86-64 Machine Language

x86-64 Machine Language after Assembly

x86-64 Machine Language after Linking
x86-64 Machine Language

x86-64 machine language
• Difficult to generalize about x86-64 instruction format
  • Many (most!) instructions are exceptions to the rules
  • Many instructions use this format…
# x86-64 Instruction Format

<table>
<thead>
<tr>
<th>Instruction prefixes</th>
<th>Opcode</th>
<th>ModR/M</th>
<th>SIB</th>
<th>Displacement</th>
<th>Immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Up to 4 prefixes of 1 byte each (optional)</td>
<td>1, 2, or 3 bytes</td>
<td>1 byte (if required)</td>
<td>1 byte (if required)</td>
<td>1, 2, or 4 bytes (if required)</td>
<td>1, 2, 4, or 8 bytes (if required)</td>
</tr>
</tbody>
</table>

### Mod
- 7
- 6
- 5

### Reg/Opcode
- 3
- 2
- 0

### R/M
- 7
- 6
- 5

### Scale
- 3
- 2

### Index
- 0

### Base

### Instruction prefix
- Sometimes a repeat count
- Rarely used; don’t be concerned
## x86-64 Instruction Format (cont.)

<table>
<thead>
<tr>
<th>Instruction prefixes</th>
<th>Opcode</th>
<th>ModR/M</th>
<th>SIB</th>
<th>Displacement</th>
<th>Immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Up to 4 prefixes of 1 byte each (optional)</td>
<td>1, 2, or 3 bytes</td>
<td>1 byte (if required)</td>
<td>1 byte (if required)</td>
<td>1, 2, or 4 bytes (if required)</td>
<td>1, 2, 4, or 8 bytes (if required)</td>
</tr>
</tbody>
</table>

### Opcode
- Specifies which operation should be performed
  - Add, move, call, etc.
  - Sometimes specifies additional (or less) information
## x86-64 Instruction Format (cont.)

<table>
<thead>
<tr>
<th>Instruction prefixes</th>
<th>Opcode</th>
<th>ModR/M</th>
<th>SIB</th>
<th>Displacement</th>
<th>Immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Up to 4 prefixes of 1 byte each (optional)</td>
<td>1, 2, or 3 bytes</td>
<td>1 byte (if required)</td>
<td>1 byte (if required)</td>
<td>1, 2, or 4 bytes (if required)</td>
<td>1, 2, 4, or 8 bytes (if required)</td>
</tr>
</tbody>
</table>

### ModR/M (register mode, register/opcode, register/memory)
- Specifies types of operands (immediate, register, memory)
- Specifies sizes of operands (byte, word, long)
- Sometimes contains an extension of the opcode
Sometimes 3 bits in ModR/M byte, along with extra bit in another field, specify a register

- For 8-byte registers:

<table>
<thead>
<tr>
<th>Extra</th>
<th>ModR/M</th>
<th>Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
<td>RAX</td>
</tr>
<tr>
<td>0</td>
<td>001</td>
<td>RCX</td>
</tr>
<tr>
<td>0</td>
<td>010</td>
<td>RDX</td>
</tr>
<tr>
<td>0</td>
<td>011</td>
<td>RBX</td>
</tr>
<tr>
<td>0</td>
<td>100</td>
<td>RSP</td>
</tr>
<tr>
<td>0</td>
<td>101</td>
<td>RBP</td>
</tr>
<tr>
<td>0</td>
<td>110</td>
<td>RSI</td>
</tr>
<tr>
<td>0</td>
<td>111</td>
<td>RDI</td>
</tr>
<tr>
<td>1</td>
<td>000</td>
<td>R8</td>
</tr>
<tr>
<td>1</td>
<td>001</td>
<td>R9</td>
</tr>
<tr>
<td>1</td>
<td>010</td>
<td>R10</td>
</tr>
<tr>
<td>1</td>
<td>011</td>
<td>R11</td>
</tr>
<tr>
<td>1</td>
<td>100</td>
<td>R12</td>
</tr>
<tr>
<td>1</td>
<td>101</td>
<td>R13</td>
</tr>
<tr>
<td>1</td>
<td>110</td>
<td>R14</td>
</tr>
<tr>
<td>1</td>
<td>111</td>
<td>R15</td>
</tr>
</tbody>
</table>

Similar mappings exist for 4-byte, 2-byte and 1-byte registers
**x86-64 Instruction Format (cont.)**

<table>
<thead>
<tr>
<th>Instruction prefixes</th>
<th>Opcode</th>
<th>ModR/M</th>
<th>SIB</th>
<th>Displacement</th>
<th>Immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Up to 4 prefixes of 1 byte each (optional)</td>
<td>1, 2, or 3 bytes</td>
<td>1 byte (if required)</td>
<td>1 byte (if required)</td>
<td>1, 2, or 4 bytes (if required)</td>
<td>1, 2, 4, or 8 bytes (if required)</td>
</tr>
</tbody>
</table>

- **Mod**
- **Reg/Opcode**
- **R/M**
- **Scale**
- **Index**
- **Base**

**SIB (scale, index, base)**

- Used when one of the operands is a memory operand that uses a scale, an index register, and/or a base register
x86-64 Instruction Format (cont.)

Displacement
- Part of memory operand, or…
- In jump and call instructions, indicates the displacement between the destination instruction and the jump/call instruction
  - More precisely, indicates:
    \[ \text{addr of destination instr} - \text{addr of instr following the jump/call} \]
- Uses little-endian byte order
x86-64 Instruction Format (cont.)

<table>
<thead>
<tr>
<th>Instruction prefixes</th>
<th>Opcode</th>
<th>ModR/M</th>
<th>SIB</th>
<th>Displacement</th>
<th>Immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Up to 4 prefixes of 1 byte each (optional)</td>
<td>1, 2, or 3 bytes</td>
<td>1 byte (if required)</td>
<td>1 byte (if required)</td>
<td>1, 2, or 4 bytes (if required)</td>
<td>1, 2, 4, or 8 bytes (if required)</td>
</tr>
</tbody>
</table>

- **Mod**
- **Reg/Opcode**
- **R/M**
- **Scale**
- **Index**
- **Base**

**Immediate**
- Specifies an immediate operand
- Uses little-endian byte order
Example 1

Assembly lang: \texttt{addq \%rax, \%rbx}

Machine lang: \texttt{4801c3}

Explanation:

\begin{verbatim}
01001000 00000001 11000011
\end{verbatim}

 Opcode: This is an \texttt{add} instruction whose src operand is an 8-byte register or memory operand and whose dest operand is a 8-byte register

 ModR/M: The M field of the ModR/M byte designates a register

 ModR/M: The src register is RAX

 ModR/M: The dest register is RBX

Observation: Sometimes opcode specifies operation (e.g. add) and format(s) of operand(s)
Example 2

Assembly lang:      movl $1, %ebx
Machine lang:      bb01000000

Explanation:

10111011 00000001 00000000 00000000 00000000

Opcode: This is a mov instruction whose src operand is a 4-byte immediate

Opcode: the destination operand is the EBX register

Immediate: The immediate operand is 1

Observation: Sometimes opcode specifies operation and operand(s)
Observation: Immediate operands are in little-endian byte order
Examples 3, 4

Assembly lang: pushq %rax

Machine lang: 50

Explanation:

01010000
Opcode: This is a pushq %rax instruction

Assembly lang: pushq %rcx

Machine lang: 51

Explanation:

01010001
Opcode: This is a pushq %rcx instruction

Observation: Sometimes opcode specifies operation and operand(s)
Observation: pushq is used often, so is optimized
Example 5

Assembly lang: \texttt{movl \text{-8(\%eax,\%ebx,4)}, \%edx}
Machine lang: \texttt{678b5498f8}

Explanation:

\begin{verbatim}
10100111 10001011 01010100 10011000 11111000
\end{verbatim}

Opcode: This is a \texttt{mov} instruction whose src operand is a 4-byte register or memory operand and whose dest operand is a 4-byte register

ModR/M: The src operand is a register, the dest operand is of the form \texttt{disp(base,index, scale)}, the base and index registers are 4-byte registers, and the disp is one-byte

ModR/M: The destination register is EDX
SIB: The scale is 4
SIB: The index register is EBX
SIB: The base reg is EAX

Displacement: The disp is \text{-8}

Observation: Two’s complement notation
Observation: Complicated!!!
Agenda

x86-64 Machine Language

x86-64 Machine Language after Assembly

x86-64 Machine Language after Linking
An Example Program

A simple (nonsensical) program:

```c
#include <stdio.h>
int main(void)
{
    printf("Type a char: ");
    if (getchar() == 'A')
        printf("Hi\n");
    return 0;
}
```

Let's consider the machine language equivalent after assembly...

```
.section ".rodata"
msg1:  .string "Type a char"
msg2:  .string "Hi\n"
.section ".text"
.globl main
main:
    movl $0, %eax
    movq $msg1, %rdi
    call printf
    call getchar
    cmpl $'A', %eax
    jne skip
    movl $0, %eax
    movq $msg2, %rdi
    call printf
skip:
    movl $0, %eax
    ret
```
Examining Machine Lang: RODATA

Assemble program; run objdump

$ gcc217 -c detecta.s
$ objdump --full-contents --section .rodata detecta.o

detecta.o: file format elf64-x86-64

Contents of section .rodata:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>54797065 20612063 6861723a 20004869</td>
</tr>
<tr>
<td>0010</td>
<td>0a00</td>
</tr>
</tbody>
</table>

- Assembler does not know addresses
- Assembler knows only offsets
- "Type a char" starts at offset 0
- "Hi\n" starts at offset 0e
Examining Machine Lang: TEXT

Assemble program; run objdump

Disassembly of section .text:

0000000000000000 <main>:

0: b8 00 00 00 00  
  mov $0x0,%eax

5: 48 c7 c7 00 00 00 00  
  mov $0x0,%rdi

8: R_X86_64_32S .rodata
  callq 11 <main+0x11>

c: e8 00 00 00 00  
  callq 16 <main+0x16>

11: e8 00 00 00 00 00  
  printf 0x4

16: 83 f8 41  
  cmp $0x41,%eax

19: 75 11  
  jne 2c <skip>

1b: b8 00 00 00 00 00  
  mov $0x0,%eax

20: 48 c7 c7 00 00 00 00  
  mov $0x0,%rdi

23: R_X86_64_32S .rodata+0xe
  callq 2c <skip>

27: e8 00 00 00 00 00  
  printf 0x4

000000000000002c <skip>:

2c: b8 00 00 00 00  
  mov $0x0,%eax

31: c3  
  retq

Let's examine one line at a time…

Machine language
Relocation records
Assembly language
Offsets
movl $0, %eax

$ gcc -c detecta.s
gcc -c detector.s
gcc --disassemble --reloc detector.o
detector.o: file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <main>:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Opcode</th>
<th>Mnemonic</th>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>b8 00 00 00 00</td>
<td>mov</td>
<td>$0x0,%eax</td>
</tr>
<tr>
<td>5</td>
<td>48 c7 c7 00 00 00 00</td>
<td>mov</td>
<td>$0x0,%rdi</td>
</tr>
<tr>
<td>8</td>
<td>R_X86_64_32S .rodata</td>
<td>callq</td>
<td>11 &lt;main+0x11&gt;</td>
</tr>
<tr>
<td>c</td>
<td>e8 00 00 00 00</td>
<td>callq</td>
<td>11 &lt;main+0x11&gt;</td>
</tr>
<tr>
<td>d</td>
<td>R_X86_64_PC32 printf-0x4</td>
<td>callq</td>
<td>16 &lt;main+0x16&gt;</td>
</tr>
<tr>
<td>11</td>
<td>e8 00 00 00 00</td>
<td>callq</td>
<td>16 &lt;main+0x16&gt;</td>
</tr>
<tr>
<td>12</td>
<td>R_X86_64_PC32 getchar-0x4</td>
<td>callq</td>
<td>2c &lt;skip&gt;</td>
</tr>
<tr>
<td>16</td>
<td>83 f8 41</td>
<td>cmp</td>
<td>$0x41,%eax</td>
</tr>
<tr>
<td>19</td>
<td>75 11</td>
<td>jne</td>
<td>2c &lt;skip&gt;</td>
</tr>
<tr>
<td>1b</td>
<td>b8 00 00 00 00</td>
<td>mov</td>
<td>$0x0,%eax</td>
</tr>
<tr>
<td>20</td>
<td>48 c7 c7 00 00 00 00</td>
<td>mov</td>
<td>$0x0,%rdi</td>
</tr>
<tr>
<td>23</td>
<td>R_X86_64_32S .rodata+0xe</td>
<td>callq</td>
<td>2c &lt;skip&gt;</td>
</tr>
<tr>
<td>27</td>
<td>e8 00 00 00 00</td>
<td>callq</td>
<td>2c &lt;skip&gt;</td>
</tr>
<tr>
<td>28</td>
<td>R_X86_64_PC32 printf-0x4</td>
<td>retq</td>
<td></td>
</tr>
</tbody>
</table>

0000000000000002c <skip>:

<table>
<thead>
<tr>
<th>Offset</th>
<th>Opcode</th>
<th>Mnemonic</th>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>2c</td>
<td>b8 00 00 00 00</td>
<td>mov</td>
<td>$0x0,%eax</td>
</tr>
<tr>
<td>31</td>
<td>c3</td>
<td>retq</td>
<td></td>
</tr>
</tbody>
</table>
movl $0, %eax

Assembly lang: movl $0, %eax
Machine lang: b800000000

Explanation:

10111000 00000000 00000000 00000000 00000000

Opcode: This is a mov instruction whose src operand is a 4-byte immediate
 Opcode: the destination operand is the EAX register
 Immediate: The immediate operand is 0
movq $msg1, %rdi

$ gcc217 -c detecta.s
$ objdump --disassemble --reloc detecta.o
detecta.o: file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <main>:
  0: b8 00 00 00 00       mov   $0x0,%eax
  5: 48 c7 c7 00 00 00 00       mov   $0x0,%rdi
     8: R_X86_64_32S  .rodata
c: e8 00 00 00 00       callq 11 <main+0x11>
   11: e8 00 00 00 00       callq 16 <main+0x16>
   16: 83 f8 41       cmp    $0x41,%eax
   19: 75 11       jne    2c <skip>
   1b: b8 00 00 00 00 00       mov   $0x0,%eax
   20: 48 c7 c7 00 00 00 00       mov   $0x0,%rdi
     23: R_X86_64_32S  .rodata+0xe
d: R_X86_64_PC32  printf-0x4
e8 00 00 00 00 00       callq 2c <skip>
   27: e8 00 00 00 00       callq 2c <skip>
   28: R_X86_64_PC32  printf-0x4

000000000000000002c <skip>:
  2c: b8 00 00 00 00       mov   $0x0,%eax
  31: c3       retq
movq $msg1, %rdi

Assembly lang: movq $msg1, %rdi
Machine lang: 48 C7 C7 00 00 00 00

Explanation:

01001000 11000111 11001011 00000000 00000000 00000000 00000000
 Opcode: This is a movq instruction with a 4-byte immediate source operand and a 8 byte register destination operand
 Opcode: The destination register is RDI
 Opcode: The destination register is RDI (cont.)
 Disp: The immediate(memory address) is 0

• movq must contain an address
• Assembler knew offset marked by msg1
  • msg1 marks offset 0 relative to beginning of RODATA section
• But assembler did not know address of RODATA section!
• So assembler didn’t know address marked by msg1
• So assembler couldn’t generate this instruction completely
Relocation Record 1

$ gcc217 -c detecta.s
$ objdump --disassemble --reloc detecta.o
detecta.o: file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <main>:
  0: b8 00 00 00 00       mov $0x0,%eax
  5: 48 c7 c7 00 00 00 00 mov $0x0,%rdi
     8: R_X86_64_32S .rodata
     c: e8 00 00 00 00     callq 11 <main+0x11>
     d: R_X86_64_PC32 printf-0x4
    11: e8 00 00 00 00     callq 16 <main+0x16>
     12: R_X86_64_PC32 getchar-0x4
    16: 83 f8 41       cmp $0x41,%eax
     19: 75 11       jne 2c <skip>
    1b: b8 00 00 00 00 mov $0x0,%eax
    20: 48 c7 c7 00 00 00 00 mov $0x0,%rdi
     23: R_X86_64_32S .rodata+0xe
    27: e8 00 00 00 00     callq 2c <skip>
     28: R_X86_64_PC32 printf-0x4

00000000000000002c <skip>:
  2c: b8 00 00 00 00       mov $0x0,%eax
  31: c3       retq
Dear Linker,

Please patch the TEXT section at offsets $08_H$ through $0B_H$. Do an “absolute” type of patch. When you determine the addr of the RODATA section, place that address in the TEXT section at the prescribed place.

Sincerely,
Assembler
call printf

$ gcc -c detecta.s
$ objdump --disassemble --reloc detecta.o

detecta.o:    file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <main>:
  0:  b8 00 00 00 00       mov    $0x0,%eax
  5:  48 c7 c7 00 00 00 00 mov    $0x0,%rdi
     8:  R_X86_64_32S   .rodata
  c:  e8 00 00 00 00       callq  11 <main+0x11>
     d:  R_X86_64_PC32    printf-0x4
  11:  e8 00 00 00 00       callq  16 <main+0x16>
     12:  R_X86_64_PC32    getchar-0x4
  16:  83 f8 41            cmp    $0x41,%eax
  19:  75 11               jne    2c <skip>
  1b:  b8 00 00 00 00 00   mov    $0x0,%eax
  20:  48 c7 c7 00 00 00 00 mov    $0x0,%rdi
     23:  R_X86_64_32S   .rodata+0xe
  27:  e8 00 00 00 00       callq  2c <skip>
     28:  R_X86_64_PC32    printf-0x4

00000000000000002c <skip>:
  2c:  b8 00 00 00 00 00   mov    $0x0,%eax
  31:  c3                  retq
call printf

Assembly lang: call printf
Machine lang: e8 00 00 00 00
Explanation:

11101000 00000000 00000000 00000000 00000000
Opcode: This is a call instruction with a 4-byte displacement
Disp: The displacement is 00000000_H (0)

- call must contain a displacement
- Assembler had to generate the displacement:
  [addr of printf] – [addr after call instr]
- But assembler didn’t know addr of printf
  • printf isn’t even present yet!
- So assembler couldn’t generate this instruction completely
Relocation Record 2

$ gcc217 -c detecta.s
$ objdump --disassemble --reloc detecta.o

detecta.o:    file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <main>:
  0:    b8 00 00 00 00          mov    $0x0,%eax
  5:    48 c7 c7 00 00 00 00 00 mov    $0x0,%rdi
  8: R_X86_64_32S      .rodata
     e8 00 00 00 00          callq  11 <main+0x11>
  c:  R_X86_64_32S      .rodata+0xe
     e8 00 00 00 00          callq  11 <main+0x11>
  11: e8 00 00 00 00          callq  16 <main+0x16>
     83 f8 41 mov    $0x41,%eax
  19:  75 11 jne    2c <skip>
  1b:    b8 00 00 00 00 00 mov    $0x0,%eax
  20:    48 c7 c7 00 00 00 00 mov    $0x0,%rdi
  23: R_X86_64_32S      .rodata+0xe
     e8 00 00 00 00          callq  2c <skip>
  27: e8 00 00 00 00          callq  2c <skip>
     83 f8 41 mov    $0x41,%eax
  2f:  75 11 jne    2c <skip>
  31:    c3 retq

000000000000002c <skip>:
Dear Linker,

Please patch the TEXT section at offsets $0d_{16}$ through $10_{16}$. Do a "relative" type of patch. When you determine the addr of printf, compute $[\text{addr of printf}] - [\text{addr after call}]$ and place the result at the prescribed place.

Sincerely,
Assembler
34

call getchar

$ gcc217 -c detecta.s
$ objdump --disassemble --reloc detecta.o
detecta.o: file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <main>:
  0:   b8 00 00 00 00         mov    $0x0,%eax
  5:   48 c7 c7 00 00 00 00  mov    $0x0,%rdi

  8: R_X86_64_32S .rodata
c:   e8 00 00 00 00         callq  11 <main+0x11>
d: R_X86_64_PC32 printf-0x4

11: e8 00 00 00 00         callq  16 <main+0x16>

  12: R_X86_64_PC32 getchar-0x4

  16: 83 f8 41         cmp    $0x41,%eax
  19: 75 11                jne   2c <skip>

  1b:   b8 00 00 00 00     mov    $0x0,%eax
  20:   48 c7 c7 00 00 00 00  mov    $0x0,%rdi

  23: R_X86_64_32S .rodata+0xe
c:   e8 00 00 00 00         callq  2c <skip>
d: R_X86_64_PC32 printf-0x4

00000000000000002c <skip>:
  2c:   b8 00 00 00 00     mov    $0x0,%eax
  31:   c3                retq
**call getchar**

**Assembly lang:**
```
call getchar
```

**Machine lang:**
```
e8 00 00 00 00
```

**Explanation:**

```
11101000 00000000 00000000 00000000 00000000
```

Opcode: This is a call instruction with a 4-byte displacement

Disp: The displacement is \(00000000_\text{H}(0)\)

- **call** must contain a **displacement**
- Assembler had to generate the displacement:
  - \([\text{addr of getchar}] - [\text{addr after call instr}]\)
- But assembler didn’t know addr of getchar
  - **getchar** isn’t even present yet!
- So assembler couldn’t generate this instruction completely
$ gcc217 -c detecta.s
$ objdump --disassemble --reloc detecta.o
detecta.o: file format elf64-x86-64

Disassembly of section .text:

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>b8 00 00 00 00</td>
<td>mov $0x0,%eax</td>
</tr>
<tr>
<td>5</td>
<td>48 c7 c7 00 00 00 00</td>
<td>mov $0x0,%rdi</td>
</tr>
<tr>
<td>8</td>
<td>R_X86_64_32S .rodata</td>
<td>callq 11 &lt;main+0x11&gt;</td>
</tr>
<tr>
<td>c</td>
<td>e8 00 00 00 00</td>
<td>callq 16 &lt;main+0x16&gt;</td>
</tr>
<tr>
<td>11</td>
<td>e8 00 00 00 00</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>R_X86_64_PC32 getchar-0x4</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>83 f8 41</td>
<td>cmp $0x41,%eax</td>
</tr>
<tr>
<td>19</td>
<td>75 11</td>
<td>jne 2c &lt;skip&gt;</td>
</tr>
<tr>
<td>1b</td>
<td>b8 00 00 00 00</td>
<td>mov $0x0,%eax</td>
</tr>
<tr>
<td>20</td>
<td>48 c7 c7 00 00 00 00</td>
<td>mov $0x0,%rdi</td>
</tr>
<tr>
<td>23</td>
<td>R_X86_64_32S .rodata+0xe</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>e8 00 00 00 00</td>
<td>callq 2c &lt;skip&gt;</td>
</tr>
<tr>
<td>28</td>
<td>R_X86_64_PC32 printf-0x4</td>
<td></td>
</tr>
<tr>
<td>000000000000000000c &lt;skip&gt;:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2c</td>
<td>b8 00 00 00 00</td>
<td>mov $0x0,%eax</td>
</tr>
<tr>
<td>31</td>
<td>c3</td>
<td>retq</td>
</tr>
</tbody>
</table>

Relocation Record 3
Dear Linker,

Please patch the TEXT section at offsets 12\text{H} through 15\text{H}. Do a “relative” type of patch. When you determine the addr of \texttt{getchar}, compute [offset of \texttt{getchar}] – [addr after call] and place the result at the prescribed place.

Sincerely,
Assembler
$ gcc217 -c detecta.s
$ objdump --disassemble --reloc detecta.o

detecta.o:   file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <main>:
   0:  b8 00 00 00 00         mov    $0x0,%eax
   5:  48 c7 c7 00 00 00 00   mov    $0x0,%rdi
   8:  R_X86_64_32S   .rodata
   c:  e8 00 00 00 00         callq  11 <main+0x11>
   d:  R_X86_64_PC32  printf-0x4
  11:  e8 00 00 00 00         callq  16 <main+0x16>
  12:  R_X86_64_PC32  getchar-0x4
  16:  83 f8 41              cmp    $0x41,%eax
  19:  75 11                 jne    2c <skip>
  1b:  b8 00 00 00 00 00     mov    $0x0,%eax
  20:  48 c7 c7 00 00 00 00   mov    $0x0,%rdi
  23:  R_X86_64_32S   .rodata+0xe
  27:  e8 00 00 00 00         callq  2c <skip>
  28:  R_X86_64_PC32  printf-0x4
  2c:  b8 00 00 00 00 00     mov    $0x0,%eax
  31:  c3                 retq
cmpl $'A', %eax

Assembly lang: cmpl $'A', %eax
Machine lang: 83 f8 41

Explanation:

10000011 11110000 01000001
Opcode: This is an instruction whose source operand is a one-byte immediate and whose destination operand is a register or memory

ModR/M: This is a cmpl instruction, and the last three bytes of the ModR/M field specify the destination register

ModR/M: The dest register is EAX
The immediate operand is $41H (A)
Disassembly of section .text:

0000000000000000  <main>:
  0:   b8 00 00 00 00          mov   $0x0,%eax
  5:   48 c7 c7 00 00 00 00 00 mov   $0x0,%rdi
  8: R_X86_64_32S  .rodata
     e8 00 00 00 00          callq 11 <main+0x11>
  c: d: R_X86_64_PC32  printf-0x4
     e8 00 00 00 00          callq 16 <main+0x16>
  11: 12: R_X86_64_PC32  getchar-0x4
    16: 83 f8 41            cmp    $0x41,%eax
    19: 75 11               jne    2c <skip>
    1b:  b8 00 00 00 00 00    mov   $0x0,%eax
    20:  48 c7 c7 00 00 00 00 00 mov   $0x0,%rdi
    23: R_X86_64_32S  .rodata+0xe
    27:  e8 00 00 00 00 00    callq 2c <skip>
    28: R_X86_64_PC32  printf-0x4

00000000000000002c  <skip>:
  2c:   b8 00 00 00 00 00 mov   $0x0,%eax
  31:   c3                  retq
jne skip

Assembly lang: jne skip
Machine lang: 75 11
Explanation:

01110101 00010001
 Opcode: This is a jne instruction with a one-byte displacement
     Disp: The displacement is 11_H (17_D)

- jne must contain a displacement
- Assembler had to generate the displacement:
  [addr of skip] – [addr after jne instr]
  Assembler did know addr of skip
- So assembler could generate this instruction completely
  2c_H – 1b_H = 11_H = 17_D
Is it clear why jump and call instructions contain displacements instead of addresses
movl $0, %eax

$ gcc217 -c detecta.s
$ objdump --disassemble --reloc detecta.o
detecta.o: file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <main>:
  0:    b8 00 00 00 00                      mov $0x0,%eax
  5:   48 c7 c7 00 00 00 00 00              mov $0x0,%rdi
     8: R_X86_64_32S .rodata
c:    e8 00 00 00 00 00                   callq 11 <main+0x11>
d:    R_X86_64_PC32 printf-0x4
11:    e8 00 00 00 00 00                   callq 16 <main+0x16>
16:    83 f8 41                            cmp $0x41,%eax
19:    75 11                               jne 2c <skip>
1b:    b8 00 00 00 00 00                   mov $0x0,%eax
20:   48 c7 c7 00 00 00 00 00              mov $0x0,%rdi
     23: R_X86_64_32S .rodata+0xe
27:    e8 00 00 00 00 00                   callq 2c <skip>
     28: R_X86_64_PC32 printf-0x4

0000000000000002c <skip>:
  2c:    b8 00 00 00 00 00                     mov $0x0,%eax
  31:    c3                                    retq
movl $0, %eax

Assembly lang: movl $0, %eax
Machine lang: b800000000

Explanation:

10111000 00000001 00000000 00000000 00000000

Opcode: This is a mov instruction whose src operand is a 4-byte immediate
 Opcode: the destination operand is the EAX register
 Immediate: The immediate operand is 0
movq $msg2, %rdi

$ gcc217 -c detecta.s
$ objdump --disassemble --reloc detecta.o
detecta.o:   file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <main>:
  0:   b8 00 00 00 00          mov    $0x0,%eax
  5:   48 c7 c7 00 00 00 00 00 mov    $0x0,%rdi
     8: R_X86_64_32S .rodata
     c:  e8 00 00 00 00          callq  11 <main+0x11>
     d: R_X86_64_PC32 printf-0x4
    11: e8 00 00 00 00          callq  16 <main+0x16>
    12: R_X86_64_PC32 getchar-0x4
    16: 83 f8 41          cmp    $0x41,%eax
    19: 75 11          jne    2c <skip>
    1b: b8 00 00 00 00          mov    $0x0,%eax
    20: 48 c7 c7 00 00 00 00 00 mov    $0x0,%rdi
    23: R_X86_64_32S .rodata+0xe
    27: e8 00 00 00 00          callq  2c <skip>
    28: R_X86_64_PC32 printf-0x4

000000000000002c <skip>:
  2c:  b8 00 00 00 00          mov    $0x0,%eax
  31:  c3          retq
movq $msg2, %rdi

Assembly lang:         movq $msg2, %rdi
Machine lang:         48 C7 C7 00 00 00 00

Explanation:

01001000 11000111 11001011 00000000 00000000 00000000 00000000
Opcode: This is a movq instruction with a 4-byte immediate source operand and a 8 byte register destination operand
Opcode: The destination register is RDI
Opcode: The destination register is RDI (cont.)
Disp: The immediate(memory address) is 0

• **movq** must contain an **address**
• Assembler knew **offset** marked by **msg2**
  • **msg2** marks offset 0eH relative to beginning of RODATA section
• But assembler did not know address of RODATA section!
• So assembler didn’t know **address** marked by **msg2**
• So assembler couldn’t generate this instruction completely
Relocation Record 4

$ gcc217 -c detecta.s
$ objdump --disassemble --reloc detecta.o
detecta.o: file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <main>:
  0:     b8 00 00 00 00          mov $0x0,%eax
  5:     48 c7 c7 00 00 00 00 00 mov $0x0,%rdi
        8: R_X86_64_32S .rodata
c:     e8 00 00 00 00 00          callq 11 <main+0x11>
d:     e8 00 00 00 00 00          callq 16 <main+0x16>
        11: R_X86_64_PC32 printf-0x4
        12: R_X86_64_PC32 getchar-0x4
  16:    83 f8 41              cmp $0x41,%eax
  19:    75 11              jne 2c <skip>
  1b:    b8 00 00 00 00 00          mov $0x0,%eax
  20:    48 c7 c7 00 00 00 00 00 mov $0x0,%rdi
        23: R_X86_64_32S .rodata+0xe
c:     e8 00 00 00 00 00          callq 2c <skip>
        28: R_X86_64_PC32 printf-0x4
  27:             b8 00 00 00 00 00 mov $0x0,%eax
  31:             c3          retq

0000000000000002c <skip>:
Dear Linker,

Please patch the TEXT section at offsets $23H$ through $26H$. Do an “absolute” type of patch. When you determine the addr of the RODATA section, add $0eH$ to that address, and place the result in the TEXT section at the prescribed place.

Sincerely,
Assembler
$ gcc -c detecta.s
$ objdump --disassemble --reloc detecta.o

detecta.o: file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <main>:
  0: b8 00 00 00 00  mov $0x0,%eax
  5: 48 c7 c7 00 00 00 00  mov $0x0,%rdi
  8: R_X86_64_32S .rodata
     e8 00 00 00 00  callq 11 <main+0x11>
     d: R_X86_64_PC32 printf-0x4
 11: e8 00 00 00 00  callq 16 <main+0x16>
 12: R_X86_64_PC32 getchar-0x4
 16: 83 f8 41   cmp $0x41,%eax
 19: 75 11  jne 2c <skip>
 1b: b8 00 00 00 00  mov $0x0,%eax
 20: 48 c7 c7 00 00 00 00  mov $0x0,%rdi
 23: R_X86_64_32S .rodata+0xe
 27: e8 00 00 00 00  callq 2c <skip>
 28: R_X86_64_PC32 printf-0x4
 30:  b8 00 00 00 00  mov $0x0,%eax
 31:  c3  retq

0000000000000002c <skip>:
call printf

Assembly lang: call printf
Machine lang: e8 00 00 00 00

Explanation:

11101000 00000000 00000000 00000000 00000000
Opcode: This is a call instruction with a 4-byte displacement
Disp: The displacement is 00000000H (0)

• call must contain a displacement
• Assembler must generate the displacement:
  [addr of printf] – [addr after call instr]
• But assembler didn’t know addr of printf
  • printf isn’t even present yet!
• So assembler couldn’t generate this instruction completely
Relocation Record 5

```
$ gcc217 -c detecta.s
$ objdump --disassemble --reloc detecta.o

detecta.o:     file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <main>:
  0:   b8 00 00 00 00       mov $0x0,%eax
  5:   48 c7 c7 00 00 00 00 mov $0x0,%rdi
  8: R_X86_64_32S .rodata
   c:   e8 00 00 00 00     callq 11 <main+0x11>
   d: R_X86_64_PC32 printf-0x4
  11: e8 00 00 00 00     callq 16 <main+0x16>
  12: R_X86_64_PC32 getchar-0x4
  16: 83 f8 41       cmp $0x41,%eax
  19: 75 11       jne 2c <skip>
  1b: b8 00 00 00 00 00 mov $0x0,%eax
  20: 48 c7 c7 00 00 00 00 mov $0x0,%rdi
  23: R_X86_64_32S .rodata+0xe
   27: e8 00 00 00 00     callq 2c <skip>
   28: R_X86_64_PC32 printf-0x4

000000000000002c <skip>:
  2c:   b8 00 00 00 00       mov $0x0,%eax
  31:   c3       retq
```
Dear Linker,

Please patch the TEXT section at offsets 28\textsubscript{H} through 2b\textsubscript{H}. Do a “relative” type of patch. When you determine the addr of printf, compute [addr of printf] – [addr after call] and place the result at the prescribed place.

Sincerely,
Assembler
movl $0, %eax

$ gcc217 -c detecta.s
$ objdump --disassemble --reloc detecta.o
detecta.o: file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <main>:
  0:   b8 00 00 00 00  mov   $0x0,%eax
  5:   48 c7 c7 00 00 00 00  mov   $0x0,%rdi
     8: R_X86_64_32S .rodata
c:   e8 00 00 00 00  callq 11 <main+0x11>
     d: R_X86_64_PC32 printf-0x4
 11:   e8 00 00 00 00  callq 16 <main+0x16>
    12: R_X86_64_PC32 getchar-0x4
 16:   83 f8 41      cmp   $0x41,%eax
 19:   75 11        jne   2c <skip>
 1b:   b8 00 00 00 00  mov   $0x0,%eax
 20:   48 c7 c7 00 00 00 00  mov   $0x0,%rdi
    23: R_X86_64_32S .rodata+0xe
c:   e8 00 00 00 00  callq 2c <skip>
    28: R_X86_64_PC32 printf-0x4

0000000000000002c <skip>:

 2c:   b8 00 00 00 00  mov   $0x0,%eax
 31:   c3           retq
movl $0, %eax

Assembly lang: movl $0, %eax
Machine lang: b8 00 00 00 00

Explanation:

10111000 00000000 00000000 00000000 00000000
Opcodes: This is a mov instruction whose source operand is a four-byte immediate and whose destination is EAX
The immediate operand is 0
$ gcc217 -c detecta.s
$ objdump --disassemble --reloc detecta.o
detecta.o: file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <main>:
  0:  b8 00 00 00 00      mov     $0x0,%eax
  5:  48 c7 c7 00 00 00 00 mov     $0x0,%rdi
      R_X86_64_32S    .rodata
c:  e8 00 00 00 00       callq   11 <main+0x11>
    d: R_X86_64_PC32    printf-0x4
  11: e8 00 00 00 00       callq   16 <main+0x16>
     R_X86_64_PC32    getchar-0x4
  16: 83 f8 41           cmp     $0x41,%eax
  19:  75 11             jne    2c <skip>
  1b: b8 00 00 00 00      mov     $0x0,%eax
  20: 48 c7 c7 00 00 00 00 mov     $0x0,%rdi
      R_X86_64_32S    .rodata+0xe
  23: R_X86_64_PC32    printf-0x4
  27: e8 00 00 00 00       callq   2c <skip>
      R_X86_64_PC32    printf-0x4
  2c: <skip>:
    2c: b8 00 00 00 00      mov     $0x0,%eax
     R_X86_64_PC32    printf-0x4
  31: c3                  retq
ret

Assembly lang: ret
Machine lang: c3
Explanation:

11000011
Opcode: This is a ret (alias retq) instruction
Agenda

x86-64 Machine Language
x86-64 Machine Language after Assembly
x86-64 Machine Language after Linking
From Assembler to Linker

Assembler writes its data structures to .o file

Linker:
- Reads .o file
- Writes executable binary file
- Works in two phases: resolution and relocation
Linker Resolution

Resolution

- Linker resolves references

For this program, linker:

- Notes that labels `getchar` and `printf` are unresolved
- Fetches machine language code defining `getchar` and `printf` from `libc.a`
- Adds that code to TEXT section
- Adds more code (e.g. definition of `_start`) to TEXT section too
- Adds code to other sections too
Relocation

- Linker patches (“relocates”) code
- Linker traverses relocation records, patching code as specified
Examining Machine Lang: RODATA

Link program; run objdump

$ gcc217 detecta.o -o detecta
$ objdump --full-contents --section .rodata detecta

detecta:  file format elf64-x86-64

Contents of section .rodata:

<table>
<thead>
<tr>
<th>Address</th>
<th>Bytes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>400638</td>
<td>01000200 00000000 00000000 00000000 .................</td>
<td></td>
</tr>
<tr>
<td>400648</td>
<td>54797065 20612063 6861723a 20004869 Type a char: .Hi</td>
<td></td>
</tr>
<tr>
<td>400658</td>
<td>0a00</td>
<td>..</td>
</tr>
</tbody>
</table>

(Partial) addresses, not offsets

RODATA is at ...00400638\textsubscript{H}
Starts with some header info
Real start of RODATA is at ...00400648\textsubscript{H}
"Type a char: " starts at ...00400648\textsubscript{H}
"Hi\n" starts at ...00400656\textsubscript{H}
Examining Machine Lang: TEXT

Link program; run objdump

No relocation records!

Addresses, not offsets

Let’s examine one line at a time…
```bash
$ gcc217 detecta.o -o detecta
$ objdump --disassemble --reloc detecta
detecta: file format elf64-x86-64
```

Disassembly of section .text:

```
0000000000400514 <main>:
  400514: b8 00 00 00 00          mov    $0x0,%eax
  400519: 48 c7 c7 48 06 40 00    mov    $0x400648,%rdi
  400520: e8 d3 fe ff ff          callq  4003f8 <printf@plt>
  400525: e8 ee fe ff ff          callq  400418 <getchar@plt>
  40052a: 83 f8 41                cmp    $0x41,%eax
  40052d: 75 11                   jne    400540 <skip>
  40052f: b8 00 00 00 00          mov    $0x0,%eax
  400534: 48 c7 c7 56 06 40 00    mov    $0x400656,%rdi
  400539: e8 b8 fe ff ff          callq  4003f8 <printf@plt>

0000000000400540 <skip>:
  400540: b8 00 00 00 00          mov    $0x0,%eax
  400545: c3                      retq
```
movq $msg1, %rdi

Recall: Real addr of RODATA = \textbf{\ldots}00400648_H

Linker replaced \textbf{\ldots}00000000_H with real addr of RODATA + 0
\hspace{1cm} = \textbf{\ldots}00400648_H + 0
\hspace{1cm} = \textbf{\ldots}00400648_H
\hspace{1cm} = addr denoted by \texttt{msg1}
$ gcc217 detecta.o -o detecta
$ objdump --disassemble --reloc detecta

detecta: file format elf64-x86-64

... 

Disassembly of section .text:

... 

0000000000400514 <main>:

400514: b8 00 00 00 00          mov $0x0,%eax
400519: 48 c7 c7 48 06 40 00    mov $0x400648,%rdi
400520: e8 d3 fe ff ff          callq 4003f8 <printf@plt>
400525: e8 ee fe ff ff          callq 400418 <getchar@plt>
40052a: 83 f8 41                cmp $0x41,%eax
40052d: 75 11                   jne 400540 <skip>
400532: b8 00 00 00 00          mov $0x0,%eax
400537: 48 c7 c7 56 06 40 00    mov $0x400656,%rdi
40053c: e8 b8 fe ff ff          callq 4003f8 <printf@plt>

0000000000400540 <skip>:
400540: b8 00 00 00 00          mov $0x0,%eax
400545: c3                      retq

... 

Addr of printf = ...004003f8_H

Linker replaced 00000000_H with [addr of printf] - [addr after call] = ...004003f8_H - ...00400525_H = ...ffffffed3_H = -301_D
$ gcc217 detecta.o -o detecta
$ objdump --disassemble --reloc detecta
detecta: file format elf64-x86-64
...
Disassembly of section .text:
...
0000000000400514 <main>:
  400514:     b8 00 00 00 00          mov    $0x0,%eax
  400519:     48 c7 c7 48 06 40 00    mov    $0x400648,%rdi
  400520:     e8 d3 fe ff ff          callq  4003f8 <printf@plt>
  400525:     e8 ee fe ff ff          callq  400418 <getchar@plt>
  40052a:     83 f8 41                cmp    $0x41,%eax
  40052d:     75 11                   jne    400540 <skip>
  40052f:     b8 00 00 00 00          mov    $0x0,%eax
  400534:     48 c7 c7 56 06 40 00    mov    $0x400656,%rdi
  40053b:     e8 b8 fe ff ff          callq  4003f8 <printf@plt>
0000000000400540 <skip>:
  400540:     b8 00 00 00 00          mov    $0x0,%eax
  400545:     c3                      retq
...
Addr of getchar = ...
Linker replaced 00000000_{H} with [addr of getchar] - [addr after call] = ...
= ...fffff{eee}_{H}
= \(-274\)_{D}
movq $msg2, %rdi

$ gcc217 detecta.o -o detecta
$ objdump --disassemble --reloc detecta
detecta: file format elf64-x86-64
...
Disassembly of section .text:
...
0000000000400514 <main>:
  400514:  b8 00 00 00 00            mov   $0x0,%eax
  400519:  48 c7 c7 48 06 40 00      mov   $0x400648,%rdi
  400520:  e8 d3 ff ff ff           callq 4003f8 <printf@plt>
  400525:  e8 ee ff ff ff           callq 400418 <getchar@plt>
  40052a:  83 f8 41                cmp    $0x41,%eax
  40052d:  75 11                   jne    400540 <skip>
  40052f:  b8 00 00 00 00 00       mov   $0x0,%eax
  400534:  48 c7 c7 56 06 40 00      mov   $0x400656,%rdi
  400539:  e8 b8 ff ff ff           callq 4003f8 <printf@plt>

0000000000400540 <skip>:
  400540:  b8 00 00 00 00 00       mov   $0x0,%eax
  400545:  c3                      retq
...

Recall: Real addr of RODATA = ...00400648_H

Linker replaced 00000000_H with real addr of RODATA + e_H
= ...00400648_H + e_H
= ...00400656_H
= addr denoted by msg2
call printf

$ gcc217 detecta.o -o detecta
$ objdump --disassemble --reloc detecta
detecta: file format elf64-x86-64
...
Disassembly of section .text:
...
0000000000400514 <main>:
  400514:       b8 00 00 00 00          mov    $0x0,%eax
  400519:       48 c7 c7 48 06 40 00    mov    $0x400648,%rdi
  400520:       e8 d3 fe ff ff          callq  4003f8 <printf@plt>
  400525:       e8 ee fe ff ff          callq  400418 <getchar@plt>
  40052a:       83 f8 41                cmp    $0x41,%eax
  40052d:       75 11                   jne    400540 <skip>
  40052f:       b8 00 00 00 00          mov    $0x0,%eax
  400534:       48 c7 c7 56 06 40 00    mov    $0x400656,%rdi
  40053b:       e8 b8 fe ff ff          callq  4003f8 <printf@plt>

0000000000400540 <skip>:
  400540:       b8 00 00 00 00          mov    $0x0,%eax
  400545:       c3                      retq
...

Addr of printf = \ldots004003f8_{H}

Linker replaced 00000000_{H} with [addr of printf] - [addr after call] = \ldots004003f8_{H} - \ldots00400540_{H} = \ldotsfffffffeb8_{H} = -328_{D}
Summary

**x86-64 Machine Language**
- CISC: many instructions, complex format
- Fields: prefix, opcode, modR/M, SIB, displacement, immediate

**Assembler**
- Reads assembly language file
- Generates TEXT, RODATA, DATA, BSS sections
  - Containing machine language code
- Generates **relocation records**
- Writes object (.o) file

**Linker**
- Reads object (.o) file(s)
- Does **resolution**: resolves references to make code complete
- Does **relocation**: traverses relocation records to patch code
- Writes executable binary file