Machine Language

Jennifer Rexford

1

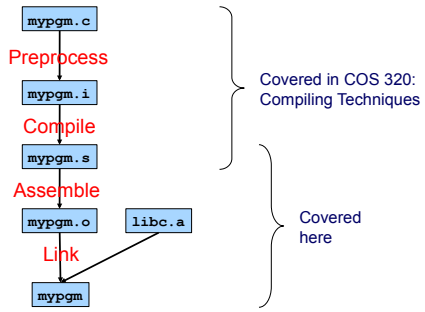
Goals of this Lecture

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

2

The Build Process



3

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)**

4

CISC and RISC Characteristics

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

5

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

6

Agenda

x86-64 Machine Language

x86-64 Machine Language after Assembly

x86-64 Machine Language after Linking

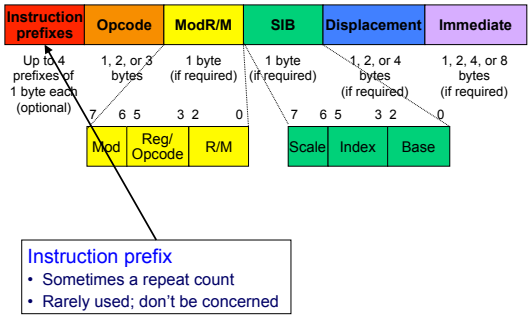
7

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…

8

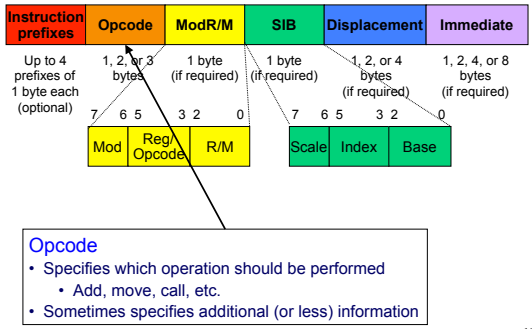
x86-64 Instruction Format



Instruction prefix
- Sometimes a repeat count
- Rarely used; don't be concerned

9

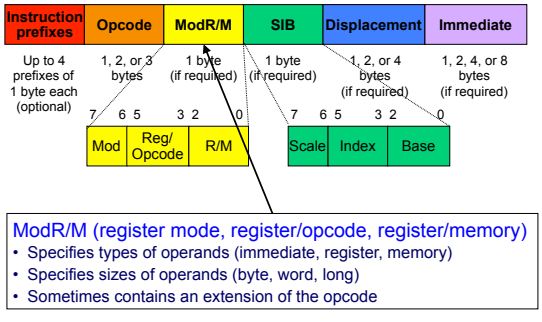
x86-64 Instruction Format (cont.)



Opcode
- Specifies which operation should be performed
  - Add, move, call, etc.
- Sometimes specifies additional (or less) information

10

x86-64 Instruction Format (cont.)



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

11

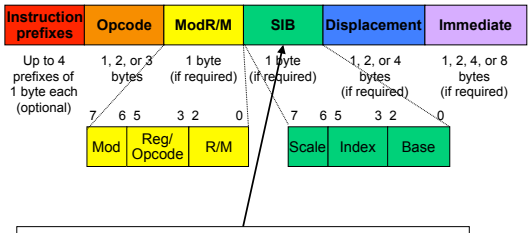
x86-64 Instruction Format (cont.)

Sometimes 3 bits in ModR/M byte, along with extra bit in another field, specify a register
- For 8-byte registers:

Extra	ModR/M	Register
0	000	RAX
0	001	RCX
0	010	RDX
0	011	RBX
0	100	RSP
0	101	RBP
0	110	RSI
0	111	RDI
1	000	R8
1	001	R9
1	010	R10
1	011	R11
1	100	R12
1	101	R13
1	110	R14
1	111	R15

Similar mappings exist for 4-byte, 2-byte and 1-byte registers

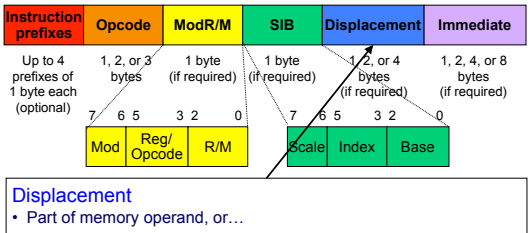
x86-64 Instruction Format (cont.)



SIB (scale, index, base)
- Used when one of the operands is a memory operand that uses a **s**cale, an **i**ndex register, and/or a **b**ase register

13

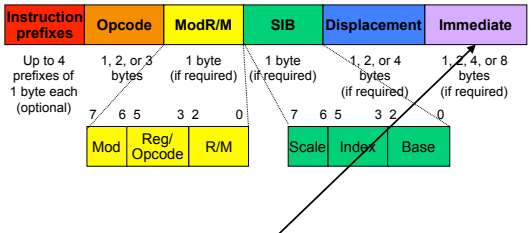
x86-64 Instruction Format (cont.)



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: [addr of destination instr] – [addr of instr following the jump/call]
- Uses little-endian byte order

14

x86-64 Instruction Format (cont.)



Immediate
- Specifies an immediate operand
- Uses little-endian byte order

15

Example 1

Assembly lang: `addq %rax, %rbx`
Machine lang: `4801c3`
Explanation:

```
01001000 00000001 11000011

Opcode: This is an 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)

16

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

17

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

18

Example 5

Assembly lang: `movl -8(%eax,%ebx,4), %edx`
Machine lang: `678b5498f8`
Explanation:

```
10100111 10001011 01010100 10011000 11111000

Opcode: This is a 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 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 -8
```

Observation: Two's complement notation
Observation: Complicated!!!

19

Agenda

x86-64 Machine Language

x86-64 Machine Language after Assembly

x86-64 Machine Language after Linking

20

An Example Program

A simple (nonsensical) program:

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

```
        .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
```

Let's consider the machine lang equivalent after assembly…

21

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:
 0000 54797065 20612063 6861723a 20004869  Type a char: .Hi
 0010 0a00                                 ..
```

Offsets | Contents

- Assembler does not know **addresses**
- Assembler knows only **offsets**
- `"Type a char"` starts at offset 0
- `"Hi\n"` starts at offset 0e

22

Examining Machine Lang: TEXT

Assemble program; run objdump

```
$ 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

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

Offsets
Machine language
Relocation records
Assembly language

Let's examine one line at a time…

23

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
  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
```

24

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>
                        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

000000000000002c <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 110010111 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

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


Relocation Record 1

```
8: R_X86_64_32S    .rodata
```

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

```
$ 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

000000000000002c <skip>:
  2c:    b8 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 00000000H (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

31

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        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

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

32

Relocation Record 2

`d: R_X86_64_PC32  printf-0x4`

Dear Linker,

Please patch the TEXT section at offsets $0d_H$ through 10_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

33

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
  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
```

34

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 00000000H (0)
```

- `call` must contain a **displacement**
- Assembler had to generate the displacement:
  [addr of `getchar`] – [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

35

Relocation Record 3

```
$ 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

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

36

Relocation Record 3

```
12: R_X86_64_PC32 getchar-0x4
```

Dear Linker,

Please patch the TEXT section at offsets 12_H through 15_H. Do a "relative" type of patch. When you determine the addr of `getchar`, compute [offset of `getchar`] – [addr after `call`] and place the result at the prescribed place.

Sincerely,
Assembler

37

cmpl $'A', %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
  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
```

38

cmpl $'A', %eax

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

```
10000011 11111000 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')
```

39

jne skip

```
$ 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

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

40

jne skip

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

```
01110101 00001101
Opcode: This is a jne instruction with a one-byte
displacement
        Disp: The displacement is 11H (17D)
```

- `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
  $2c_H - 1b_H = 11_H = 17_D$

41

jne skip

Is it clear why jump and call instructions contain displacements instead of addresses

42

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
  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
```


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        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

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 11010111 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 $0e_H$ 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        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

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


Relocation Record 4

```
23: R_X86_64_32S   .rodata+0xe
```

Dear Linker,

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

Sincerely,
Assembler

call printf

```
$ 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

000000000000002c <skip>:
  2c:    b8 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 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              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
```


Relocation Record 5

`28: R_X86_64_PC32 printf-0x4`

Dear Linker,

Please patch the TEXT section at offsets 28_H through $2b_H$. 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
  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
```


movl $0, %eax

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

`10111000 00000000 00000000 00000000 00000000`
`Opcode: This is a mov instruction whose source operand is a four-byte immediate and whose destination is EAX`
`The immediate operand is 0`

ret

```
$ 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

000000000000002c <skip>:
  2c:   b8 00 00 00 00              mov    $0x0,%eax
  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

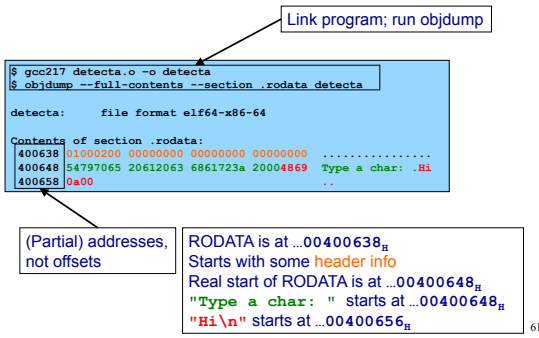
Resolution
- Linker resolves references

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

Linker Relocation

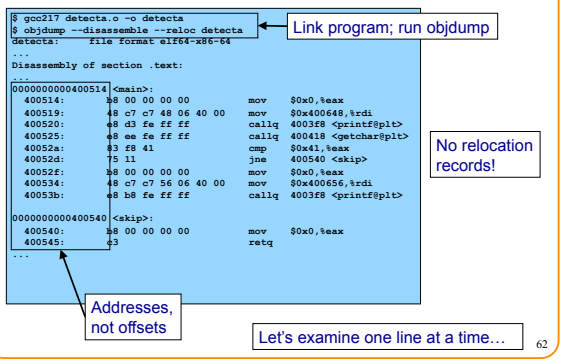
Relocation
- Linker patches ("relocates") code
- Linker traverses relocation records, patching code as specified

Examining Machine Lang: RODATA



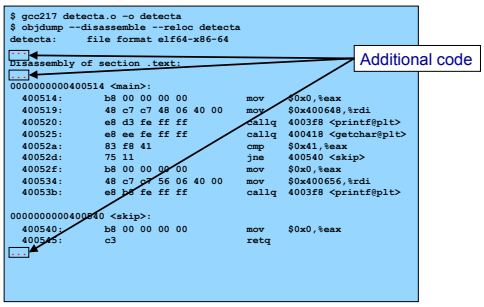
61

Examining Machine Lang: TEXT



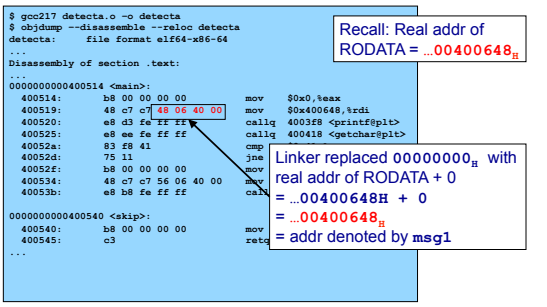
62

Additional Code



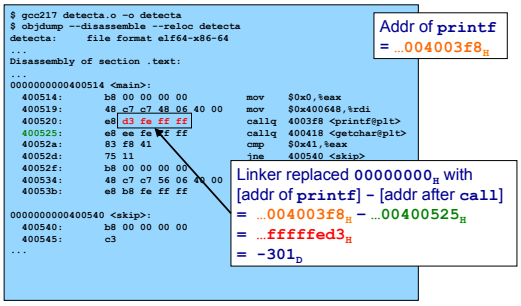
63

movq $msg1, %rdi



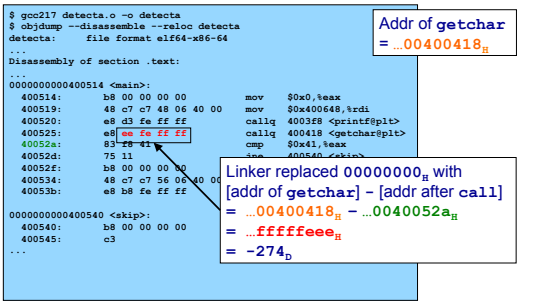
64

call printf



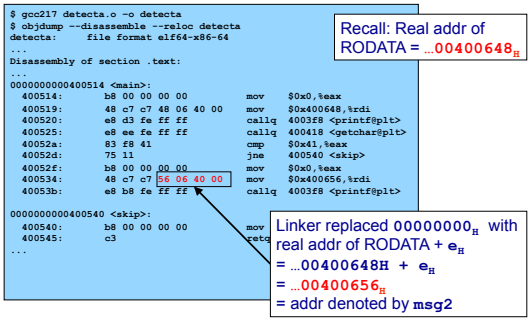
65

call getchar

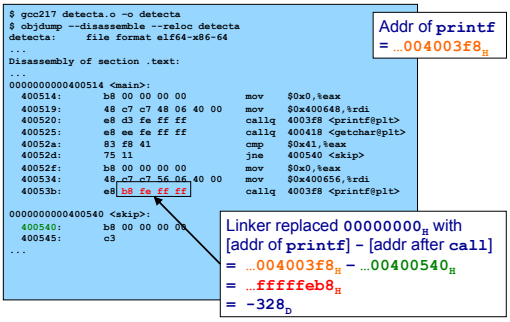


66

movq $msg2, %rdi



call printf



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