Overview

Young W. Lim

2020-02-11 Tue

Outline

Based on

1. "Self-service Linux: Mastering the Art of Problem Determination", Mark Wilding
1. "Computer Architecture: A Programmer's Perspective", Bryant & O'Hallaron

Compling 32-bit program on 64-bit gcc

- gcc -v
- gcc -m32 t.c
- sudo apt-get install gcc-multilib
- sudo apt-get install g++-multilib
- gcc-multilib
- g++-multilib
- gcc -m32
- objdump -m i386

Source codes (code.c)

- code.c

```
int accum = 0;

int sum(int x, int y)
{
  int t = x + y;
  accum += t;
  return t;
}
```

Source codes (main.c)

- main.c

```
int main()
{
  return sum(1, 3);
}
```

Linux Compiling and Linking Commands

- `gcc -m32 -O2 -S code.c`
  generating the assembly code `code.s`
- `gcc -m32 -O2 -c code.c`
  generating the object file `code.o`
- `gcc -m32 -O2 -o prog code.o main.c`
  generating the executable `prog`

Byte Examining Commands using gdb

- `gcc -m32 -O2 -g -c  code.c`
  generating the object file `code.o`
  with debugging information
- `gdb sum.o`
- `gdb) x/19xb sum`
  examining(`x`) 19 hex-formatted(`x`) bytes(`b`)

Byte Examining Commands using objdump

- `cc -m32 -O2 -c  code.c`
  generating the object file `code.o`
- `objdump -d code.o`
  generating the disassembly of `code.o`

- `gcc -m32 -O2 -o prog code.o main.c`
  generating the executable `prog`
- `objdump -d prog`
  generating the disassembly of `prog`

Assembly code code.s (1)

```
// code.s generated by
// gcc -m32 -O2 -S code.c
        .file   "code.c"
        .text
        .p2align 4,,15
        .globl  sum
        .type   sum, @function
sum:
.LFB0:
        .cfi_startproc
        call    __x86.get_pc_thunk.dx
        addl    $_GLOBAL_OFFSET_TABLE_, %edx
        movl    8(%esp), %eax
        addl    4(%esp), %eax
        addl    %eax, accum@GOTOFF(%edx)
        ret
        .cfi_endproc
```

Assembly code code.s (2)

```
.LFE0:
        .size   sum, .-sum
        .globl  accum
        .bss
        .align 4
        .type   accum, @object
        .size   accum, 4
accum:
        .zero   4
        .section        .text.__x86.get_pc_thunk.dx,"axG",@progbits,__x86.get_pc_t
        .globl  __x86.get_pc_thunk.dx
        .hidden __x86.get_pc_thunk.dx
        .type   __x86.get_pc_thunk.dx, @function
__x86.get_pc_thunk.dx:
```

Assembly code code.s (3)

```
.LFB1:
        .cfi_startproc
        movl    (%esp), %edx
        ret
        .cfi_endproc
.LFE1:
        .ident  "GCC: (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0"
        .section        .note.GNU-stack,"",@progbits
```

Assembly code main.s (1)

```
// code.s generated by
// gcc -m32 -O2 -S code.c
        .file   "main.c"
        .text
        .section        .text.startup,"ax",@progbits
        .p2align 4,,15
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        leal    4(%esp), %ecx
        .cfi_def_cfa 1, 0
        andl    $-16, %esp
        pushl   -4(%ecx)
        pushl   %ebp
        .cfi_escape 0x10,0x5,0x2,0x75,0
        movl    %esp, %ebp
        pushl   %ebx
        pushl   %ecx
```

Assembly code main.s (2)

```
.cfi_escape 0xf,0x3,0x75,0x78,0x6
.cfi_escape 0x10,0x3,0x2,0x75,0x7c
call    __x86.get_pc_thunk.bx
addl    $_GLOBAL_OFFSET_TABLE_, %ebx
subl    $8, %esp
pushl   $3
pushl   $1
call    sum@PLT
addl    $16, %esp
leal    -8(%ebp), %esp
popl    %ecx
.cfi_restore 1
.cfi_def_cfa 1, 0
popl    %ebx
.cfi_restore 3
popl    %ebp
.cfi_restore 5
leal    -4(%ecx), %esp
.cfi_def_cfa 4, 4
ret
.cfi_endproc
```

Assembly code main.s (3)

```
.LFE0:
        .size   main, .-main
        .section        .text.__x86.get_pc_thunk.bx,"axG",@progbits,__x86.get_pc_t
        .globl  __x86.get_pc_thunk.bx
        .hidden __x86.get_pc_thunk.bx
        .type   __x86.get_pc_thunk.bx, @function
__x86.get_pc_thunk.bx:
.LFB1:
        .cfi_startproc
        movl    (%esp), %ebx
        ret
        .cfi_endproc
.LFE1:
        .ident  "GCC: (Ubuntu 7.4.0-1ubuntu1~18.04.1) 7.4.0"
        .section        .note.GNU-stack,"",@progbits
```

Disassembly code of code.o

```
$ gcc -m32 -c -O2 code.c
$ objdump -d code.o
code.o:     file format elf32-i386


Disassembly of section .text:

00000000 <sum>:
   0:   e8 fc ff ff ff          call   1 <sum+0x1>
   5:   81 c2 02 00 00 00       add    $0x2,%edx
   b:   8b 44 24 08             mov    0x8(%esp),%eax
   f:   03 44 24 04             add    0x4(%esp),%eax
  13:   01 82 00 00 00 00       add    %eax,0x0(%edx)
  19:   c3                      ret

Disassembly of section .text.__x86.get_pc_thunk.dx:

00000000 <__x86.get_pc_thunk.dx>:
   0:   8b 14 24                mov    (%esp),%edx
   3:   c3                      ret
```

Disassembly code of main.o (1)

```
$ gcc -m32 -c -O2 main.c
$ objdump -d main.o

main.o:     file format elf32-i386


Disassembly of section .text.startup:

00000000 <main>:
   0:   8d 4c 24 04             lea    0x4(%esp),%ecx
   4:   83 e4 f0                and    $0xfffffff0,%esp
   7:   ff 71 fc                pushl  -0x4(%ecx)
   a:   55                      push   %ebp
   b:   89 e5                   mov    %esp,%ebp
   d:   53                      push   %ebx
   e:   51                      push   %ecx
   f:   e8 fc ff ff ff          call   10 <main+0x10>
  14:   81 c3 02 00 00 00       add    $0x2,%ebx
  1a:   83 ec 08                sub    $0x8,%esp
  1d:   6a 03                   push   $0x3
  1f:   6a 01                   push   $0x1
```

Disassembly code of main.o (2)

```
$ gcc -m32 -c -O2 main.c
$ objdump -d main.o

  21:   e8 fc ff ff ff          call   22 <main+0x22>
  26:   83 c4 10                add    $0x10,%esp
  29:   8d 65 f8                lea    -0x8(%ebp),%esp
  2c:   59                      pop    %ecx
  2d:   5b                      pop    %ebx
  2e:   5d                      pop    %ebp
  2f:   8d 61 fc                lea    -0x4(%ecx),%esp
  32:   c3                      ret

Disassembly of section .text.__x86.get_pc_thunk.bx:

00000000 <__x86.get_pc_thunk.bx>:
   0:   8b 1c 24                mov    (%esp),%ebx
   3:   c3                      ret
```

Disassembly code of prog (1)

```
$ gcc -m32 -c -O2 code.c
$ objdump -d code.o

Disassembly of section .text:

000003b0 <main>:
 3b0:   8d 4c 24 04             lea    0x4(%esp),%ecx
 3b4:   83 e4 f0                and    $0xfffffff0,%esp
 3b7:   ff 71 fc                pushl  -0x4(%ecx)
 3ba:   55                      push   %ebp
 3bb:   89 e5                   mov    %esp,%ebp
 3bd:   53                      push   %ebx
 3be:   51                      push   %ecx
 3bf:   e8 5c 00 00 00          call   420 <__x86.get_pc_thunk.bx>
 3c4:   81 c3 18 1c 00 00       add    $0x1c18,%ebx
 3ca:   83 ec 08                sub    $0x8,%esp
 3cd:   6a 03                   push   $0x3
 3cf:   6a 01                   push   $0x1
 3d1:   e8 4a 01 00 00          call   520 <sum>
```

Disassembly code of prog (2)

```
3d1:   e8 4a 01 00 00          call   520 <sum>
3d6:   83 c4 10                add    $0x10,%esp
3d9:   8d 65 f8                lea    -0x8(%ebp),%esp
3dc:   59                      pop    %ecx
3dd:   5b                      pop    %ebx
3de:   5d                      pop    %ebp
3df:   8d 61 fc                lea    -0x4(%ecx),%esp
3e2:   c3                      ret
```

Disassembly code of prog (3)

```
00000520 <sum>:
 520:   e8 f4 ff ff ff          call   519 <__x86.get_pc_thunk.dx>
 525:   81 c2 b7 1a 00 00       add    $0x1ab7,%edx
 52b:   8b 44 24 08             mov    0x8(%esp),%eax
 52f:   03 44 24 04             add    0x4(%esp),%eax
 533:   01 82 30 00 00 00       add    %eax,0x30(%edx)
 539:   c3                      ret
 53a:   66 90                   xchg   %ax,%ax
 53c:   66 90                   xchg   %ax,%ax
 53e:   66 90                   xchg   %ax,%ax
```

IA32 instructions (1)

- range in length from 1 to 15 bytes
- commonly used instructions have smaller number of bytes
- less commonly used have more operands
- from a given starting address, there is unique decoding of the bytes into machine instructions
  - for example, only pushl %ebp can start with byte 55

IA32 instructions (2)

- the disassembler determines the assembly code based only on the byte sequences in the object file depending neither on the source nor on assembly
- the disassembler uses a slightly different naming convention
  - for example, `l` is omitted in `pushl`

IA32 instructions (3)

- compared with the assembly code `code.s`, disassembly has `nop` at the end

IA32 sizes (1)

- byte (8-bit)
- word (16-bit)
- double word (32-bit)

IA32 sizes (2)

C	Intel Data	GAS	Size
`char`	Byte	`b`	1
`short`	Word	`w`	2
`int`	Double Word	`l`	4
`unsigned`	Double Word	`l`	4
`long int`	Double Word	`l`	4
`unsigned long`	Double Word	`l`	4
`char *`	Double Word	`l`	4
`float`	Single Precision	`s`	4
`double`	Double Precision	`l`	8
`long double`	Extended Precision	`t`	10/12