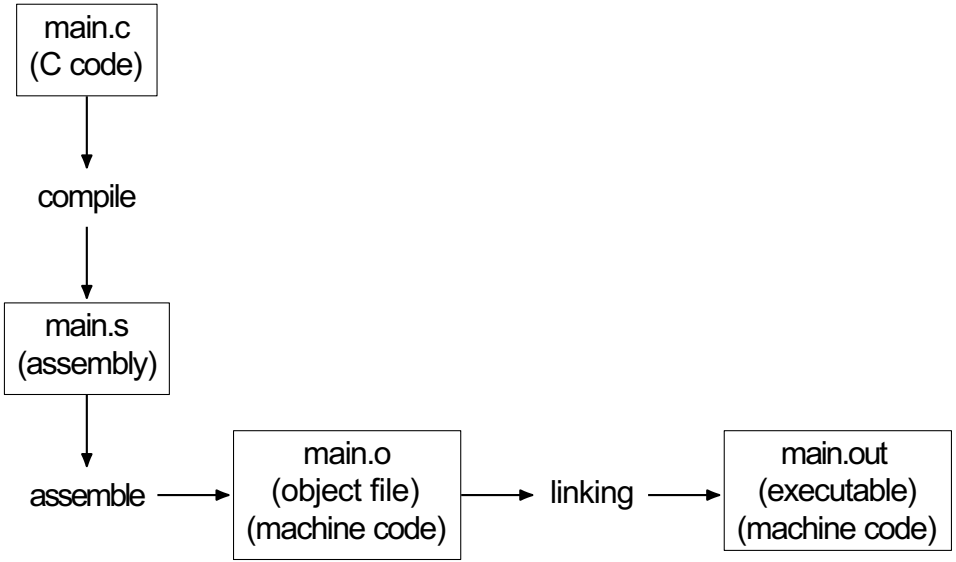
CSO-1

X86 Assembly

Daniel G. Graham PhD

1. Continue X86 Assembly Exploration
2. Big Picture View of Compilation Pipeline
3. Discussion of Computed Addresses
4. Extend the push pop example to use computed addresses

COMPILATION PIPELINE OVERVIEW



EXAMPLE C PROGRAM

hello.c

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

```
dgg6b@portal02:~/CS01/examples/compilationPipeline$ nano hello.c
```


GNU nano 6.3
hello.c
Modified
#include <stdio.h>
int main(void){
puts("Hello World");
return 0;
}
^G Help
^O Write Out
^W Where Is
^K Cut
^T Execute
^C Location
M-U Undo
^X Exit
^R Read File
^\ Replace
^U Paste
^J Justify
^/ Go To Line
M-E Redo

```
dgg6b@portal02:~/CS01/examples/compilationPipeline$ clang -S -O3 -Os hello.c -o hello.s
dgg6b@portal02:~/CS01/examples/compilationPipeline$ ls
hello.c  hello.s
dgg6b@portal02:~/CS01/examples/compilationPipeline$ nano hello.s
```

```
        .text
        .file   "hello.c"
        .globl  main                            # -- Begin function main
        .type   main,@function
main:                                   # @main
        .cfi_startproc
# %bb.0:
        pushq   %rax
        .cfi_def_cfa_offset 16
        movl    $.L.str, %edi
        callq   puts
        xorl    %eax, %eax
        popq    %rcx
        .cfi_def_cfa_offset 8
        retq
.Lfunc_end0:
        .size   main, .Lfunc_end0-main
        .cfi_endproc
                                        # -- End function
        .type   .L.str,@object                  # @.str
        .section        .rodata.str1.1,"aMS",@progbits,1
.L.str:
        .asciz  "Hello World"
        .size   .L.str, 12

        .ident  "clang version 14.0.6 (https://github.com/llvm/llvm-project.git f28c006a5895fc0e329fe15fead81e>
        .section        ".note.GNU-stack","",@progbits
```

[Read 28 lines]

EXAMPLE C PROGRAM

hello.c

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

hello.s

```
main:
        pushq   %rax
        movl    $.L.str, %edi
        callq   puts
        xorl    %eax, %eax
        popq    %rcx
        retq
--SNIP---
.L.str:
        .asciz  "Hello World"
```

```
dgg6b@portal02:~/CS01/examples/compilationPipeline$ clang -S -O3 -Os hello.c -o hello.s
dgg6b@portal02:~/CS01/examples/compilationPipeline$ ls
hello.c  hello.s
dgg6b@portal02:~/CS01/examples/compilationPipeline$ nano hello.s
dgg6b@portal02:~/CS01/examples/compilationPipeline$ clang -c hello.s -o hello.o
dgg6b@portal02:~/CS01/examples/compilationPipeline$ ls
hello.c  hello.o  hello.s
dgg6b@portal02:~/CS01/examples/compilationPipeline$ xxd hello.o
```

```
dgg6b@portal02:~/CS01/examples/compilationPipeline$ clang -c hello.s -o hello.o
dgg6b@portal02:~/CS01/examples/compilationPipeline$ ls
hello.c  hello.o  hello.s
dgg6b@portal02:~/CS01/examples/compilationPipeline$ xxd -g1 hello.o
00000000: 7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00  .ELF............
00000010: 01 00 3e 00 01 00 00 00 00 00 00 00 00 00 00 00  ..>.............
00000020: 00 00 00 00 00 00 00 00 48 02 00 00 00 00 00 00  ........H.......
00000030: 00 00 00 00 40 00 00 00 00 00 40 00 0b 00 01 00  ....@.....@.....
00000040: 50 bf 00 00 00 00 e8 00 00 00 00 31 c0 59 c3 48  P..........1.Y.H
00000050: 65 6c 6c 6f 20 57 6f 72 6c 64 00 00 63 6c 61 6e  ello World..clan
00000060: 67 20 76 65 72 73 69 6f 6e 20 31 34 2e 30 2e 36  g version 14.0.6
00000070: 20 28 68 74 74 70 73 3a 2f 2f 67 69 74 68 75 62   (https://github
00000080: 2e 63 6f 6d 2f 6c 6c 76 6d 2f 6c 6c 76 6d 2d 70  .com/llvm/llvm-p
00000090: 72 6f 6a 65 63 74 2e 67 69 74 20 66 32 38 63 30  roject.git f28c0
000000a0: 30 36 61 35 38 39 35 66 63 30 65 33 32 39 66 65  06a5895fc0e329fe
000000b0: 31 35 66 65 61 64 38 31 65 33 37 34 35 37 63 62  15fead81e37457cb
000000c0: 31 64 31 29 00 00 00 00 14 00 00 00 00 00 00 00  1d1)............
000000d0: 01 7a 52 00 01 78 10 01 1b 0c 07 08 90 01 00 00  .zR..x..........
000000e0: 14 00 00 00 1c 00 00 00 00 00 00 00 0f 00 00 00  ................
000000f0: 00 41 0e 10 4d 0e 08 00 00 00 00 00 00 00 00 00  .A..M...........
00000100: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000110: 4c 00 00 00 04 00 f1 ff 00 00 00 00 00 00 00 00  L...............
00000120: 00 00 00 00 00 00 00 00 00 00 00 00 03 00 02 00  ................
00000130: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000140: 00 00 00 00 03 00 04 00 00 00 00 00 00 00 00 00  ................
00000150: 00 00 00 00 00 00 00 00 1a 00 00 00 12 00 02 00  ................
00000160: 00 00 00 00 00 00 00 00 0f 00 00 00 00 00 00 00  ................
00000170: 15 00 00 00 10 00 00 00 00 00 00 00 00 00 00 00  ................
00000180: 00 00 00 00 00 00 00 00 02 00 00 00 00 00 00 00  ................
00000190: 0a 00 00 00 03 00 00 00 00 00 00 00 00 00 00 00  ................
000001a0: 07 00 00 00 00 00 00 00 04 00 00 00 05 00 00 00  ................
```

```
dgg6b@portal02:~/CS01/examples/compilationPipeline$ objdump -D hello.o

hello.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <main>:
   0:   50                      push   %rax
   1:   bf 00 00 00 00          mov    $0x0,%edi
   6:   e8 00 00 00 00          call   b <main+0xb>
   b:   31 c0                   xor    %eax,%eax
   d:   59                      pop    %rcx
   e:   c3                      ret

Disassembly of section .rodata.str1.1:

0000000000000000 <.rodata.str1.1>:
   0:   48                      rex.W
   1:   65 6c                   gs insb (%dx),%es:(%rdi)
   3:   6c                      insb   (%dx),%es:(%rdi)
   4:   6f                      outsl  %ds:(%rsi),(%dx)
   5:   20 57 6f                and    %dl,0x6f(%rdi)
   8:   72 6c                   jb     76 <main+0x76>
   a:   64                      fs
        ...

Disassembly of section .comment:

0000000000000000 <.comment>:
   0:   00 63 6c                add    %ah,0x6c(%rbx)
```

```
dgg6b@portal02:~/CS01/examples/compilationPipeline$ objdu
hello.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <main>:
   0:   50                      push   %rax
   1:   bf 00 00 00 00          mov    $0x0,%edi
   6:   e8 00 00 00 00          call   b <main+0xb>
   b:   31 c0                   xor    %eax,%eax
   d:   59                      pop    %rcx
   e:   c3                      ret

Disassembly of section .rodata.str1.1:

0000000000000000 <.rodata.str1.1>:
   0:   48                      rex.W
   1:   65 6c                   gs insb (%dx),%es:(%rdi)
   3:   6c                      insb   (%dx),%es:(%rdi)
   4:   6f                      outsl  %ds:(%rsi),(%dx)
   5:   20 57 6f                and    %dl,0x6f(%rdi)
   8:   72 6c                   jb     76 <main+0x76>
   a:   64                      fs
        ...

Disassembly of section .comment:

0000000000000000 <.comment>:
   0:   00 63 6c                add    %ah,0x6c(%rbx)
```

```
dgg6b@portal02:~/CS01/examples/compilationPipeline$ cla
dgg6b@portal02:~/CS01/examples/compilationPipeline$ ls
hello.c  hello.o  hello.s
dgg6b@portal02:~/CS01/examples/compilationPipeline$ xxd
00000000: 7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
00000010: 01 00 3e 00 01 00 00 00 00 00 00 00 00 00 00 00
00000020: 00 00 00 00 00 00 00 00 48 02 00 00 00 00 00 00
00000030: 00 00 00 00 40 00 00 00 00 00 40 00 0b 00 01 00
00000040: 50 bf 00 00 00 00 e8 00 00 00 00 31 c0 59 c3 4
00000050: 65 6c 6c 6f 20 57 6f 72 6c 64 00 00 63 6c 61 0
00000060: 67 20 76 65 72 73 69 6f 6e 20 31 34 2e 30 2e 3
00000070: 20 28 68 74 74 70 73 3a 2f 2f 67 69 74 68 75 0
00000080: 2e 63 6f 6d 2f 6c 6c 76 6d 2f 6c 6c 76 6d 2d 7
00000090: 72 6f 6a 65 63 74 2e 67 69 74 20 66 32 38 63 3
000000a0: 30 36 61 35 38 39 35 66 63 30 65 33 32 39 66 0
000000b0: 31 35 66 65 61 64 38 31 65 33 37 34 35 37 63 0
000000c0: 31 64 31 29 00 00 00 00 14 00 00 00 00 00 00 0
000000d0: 01 7a 52 00 01 78 10 01 1b 0c 07 08 90 01 00 0
000000e0: 14 00 00 00 1c 00 00 00 00 00 00 00 0f 00 00 0
000000f0: 00 41 0e 10 4d 0e 08 00 00 00 00 00 00 00 00 0
00000100: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0
00000110: 4c 00 00 00 04 00 f1 ff 00 00 00 00 00 00 00 0
00000120: 00 00 00 00 00 00 00 00 00 00 00 00 03 00 02 0
00000130: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 0
00000140: 00 00 00 00 03 00 04 00 00 00 00 00 00 00 00 0
00000150: 00 00 00 00 00 00 00 00 1a 00 00 00 12 00 02 0
00000160: 00 00 00 00 00 00 00 00 0f 00 00 00 00 00 00 0
00000170: 15 00 00 00 10 00 00 00 00 00 00 00 00 00 00 0
00000180: 00 00 00 00 00 00 00 00 02 00 00 00 00 00 00 0
00000190: 0a 00 00 00 03 00 00 00 00 00 00 00 00 00 00 0
000001a0: 07 00 00 00 00 00 00 00 04 00 00 00 05 00 00 0
```

SOMETHING IS MISSING

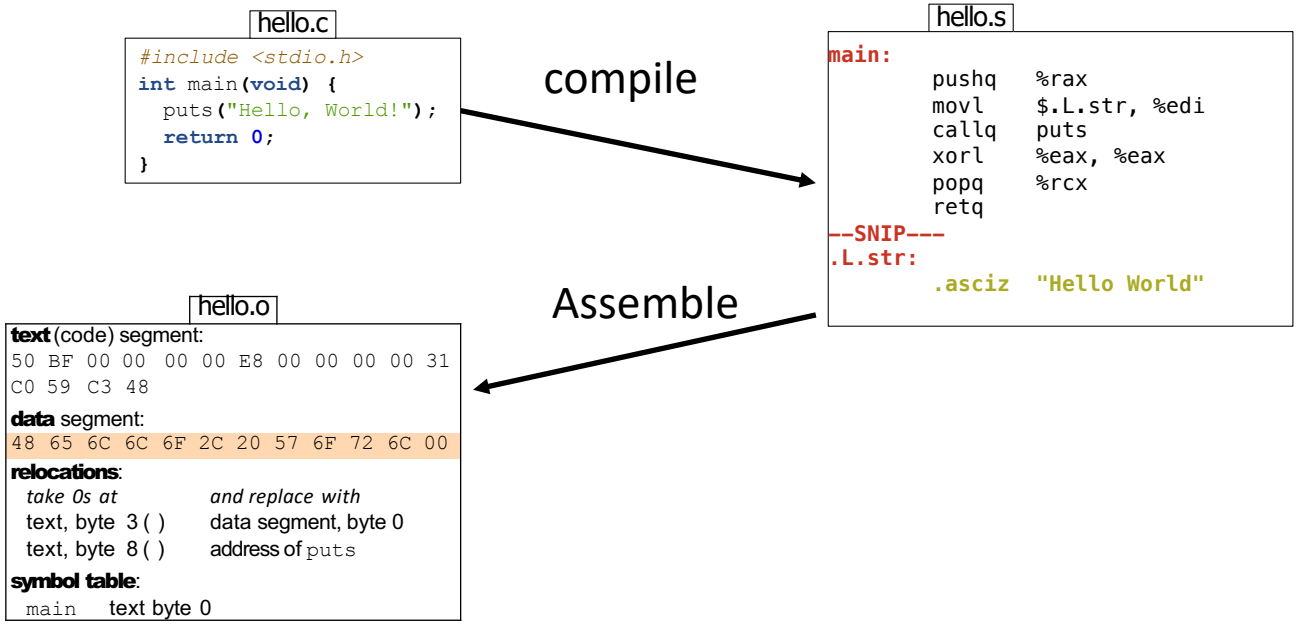
```
0000000000000000 <main>:
   0:   50                      push   %rax
   1:   bf 00 00 00 00          mov    $0x0,%edi
   6:   e8 00 00 00 00          call   b <main+0xb>
   b:   31 c0                   xor    %eax,%eax
   d:   59                      pop    %rcx
   e:   c3                      ret
```

```
main:
        .cfi_startproc
# %bb.0:
        pushq   %rax
        .cfi_def_cfa_offset 16
        movl    $.L.str, %edi
        callq   puts
        xorl    %eax, %eax
        popq    %rcx
        .cfi_def_cfa_offset 8
        retq
```

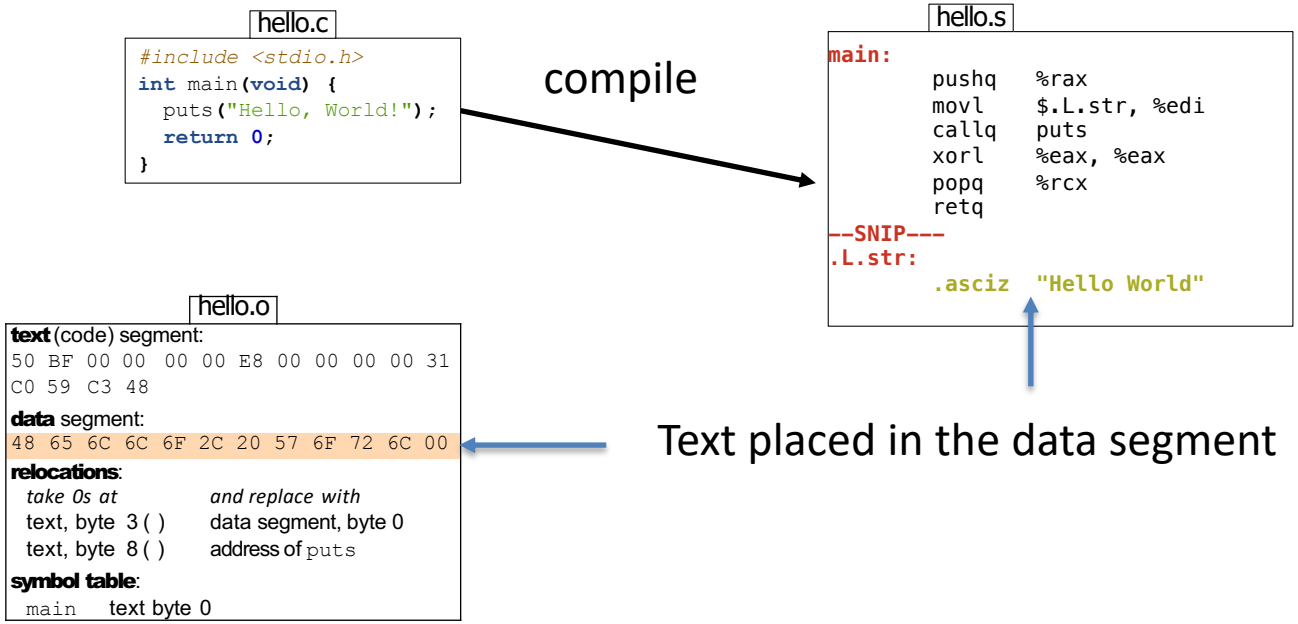
Notice the address of the function to call and the location of the string “Hello world is missing” is missing.

During the final step the linking step will add all these address

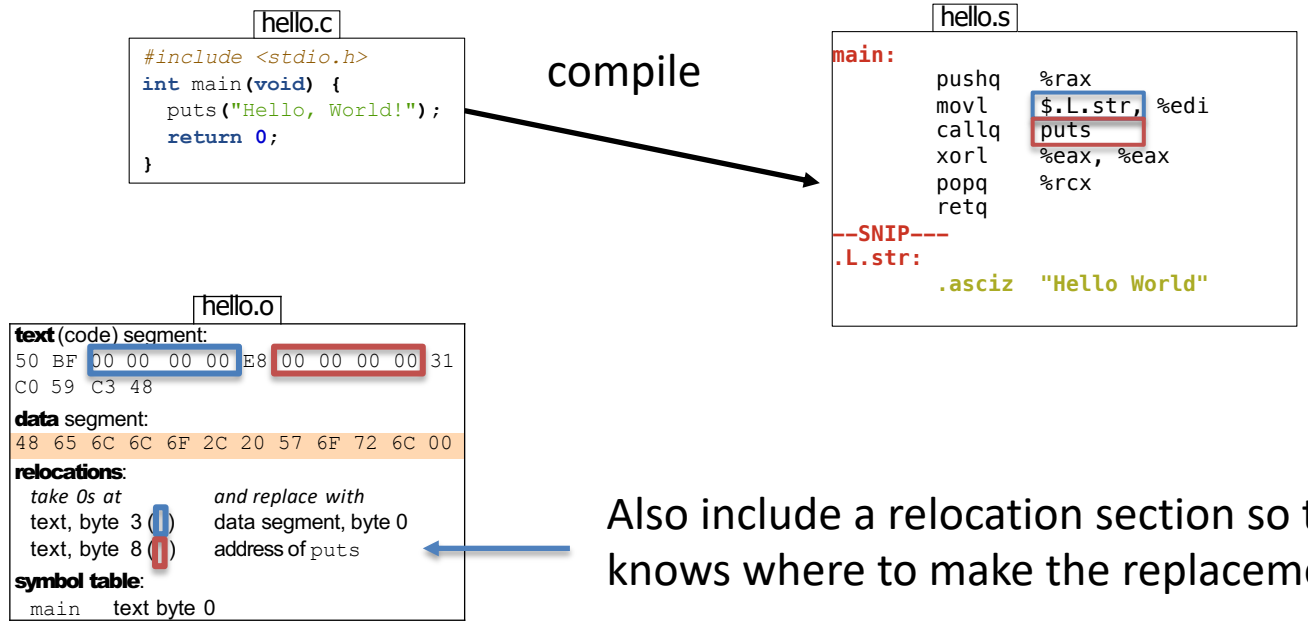
EXAMPLE C PROGRAM



EXAMPLE C PROGRAM

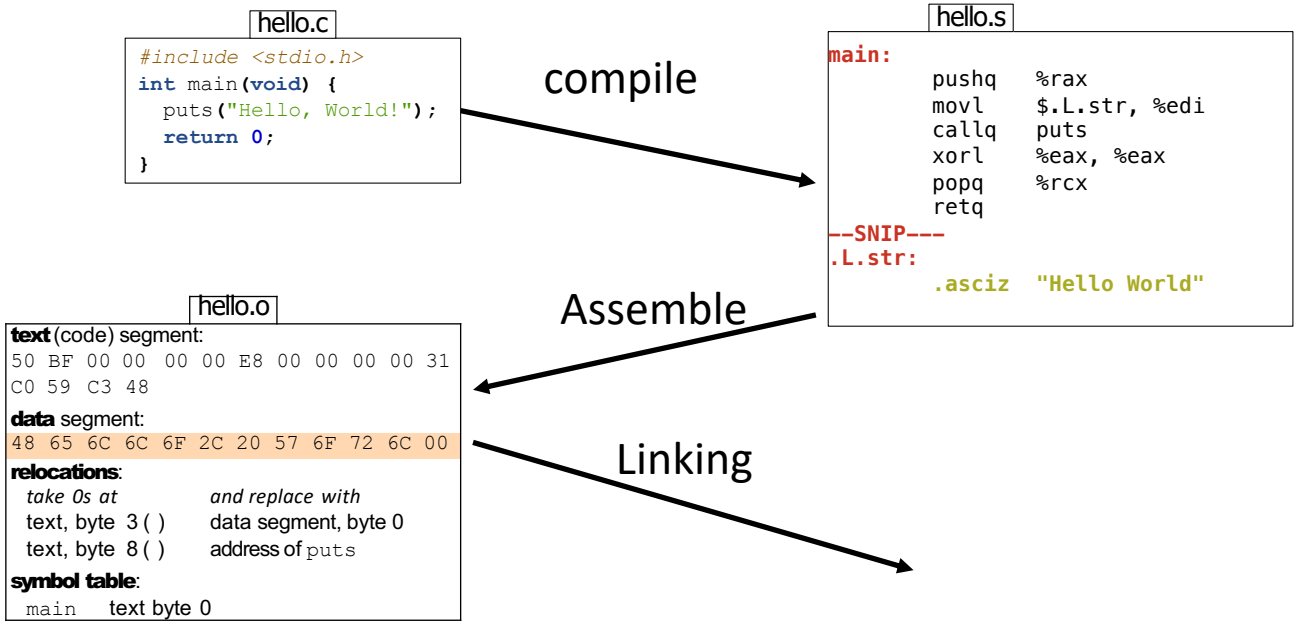


EXAMPLE C PROGRAM



Also include a relocation section so that linker knows where to make the replacements

EXAMPLE C PROGRAM



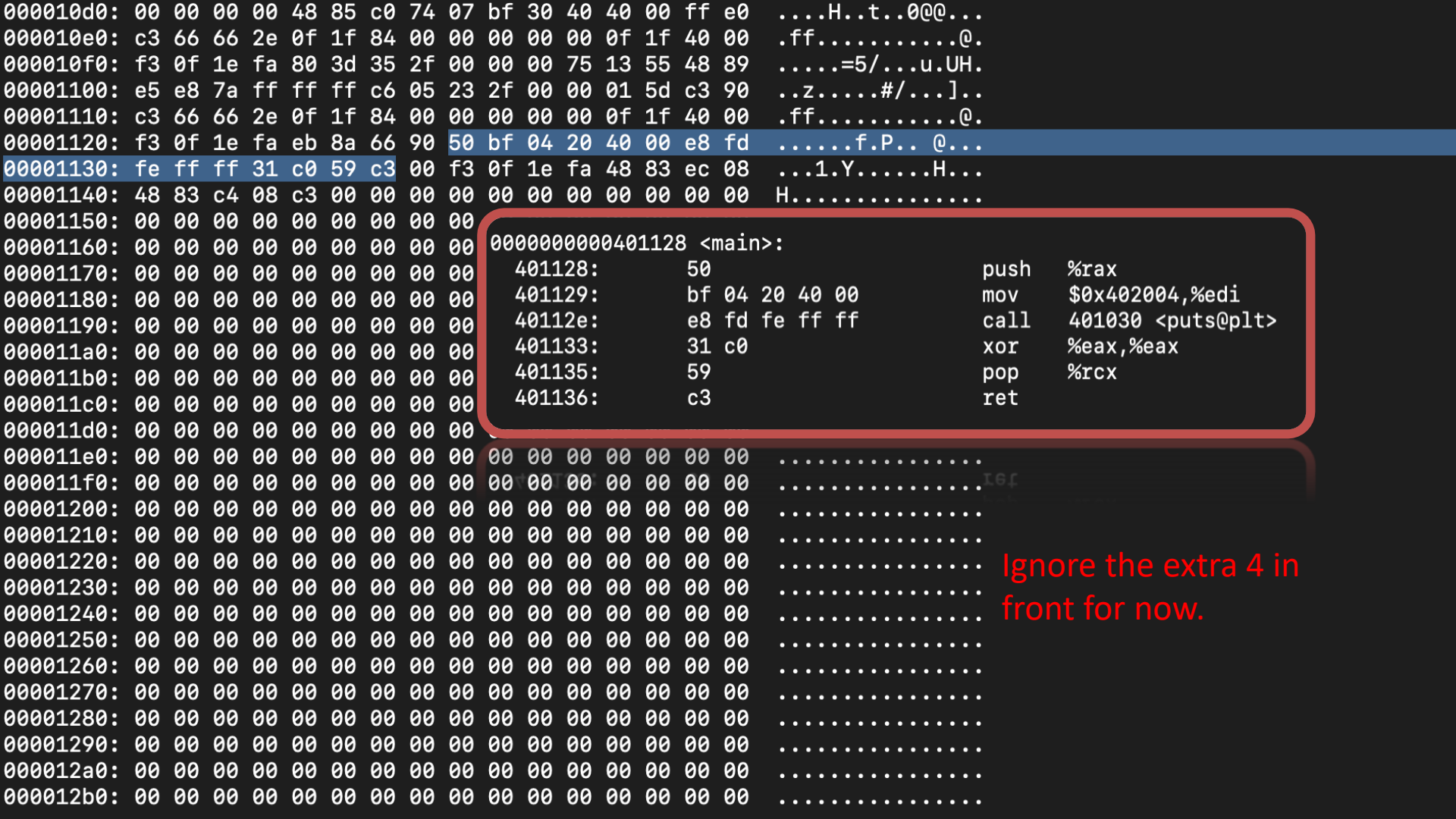
```
dgg6b@portal02:~/CS01/examples/compilationPipeline$ clang hello.s -o hello.out
dgg6b@portal02:~/CS01/examples/compilationPipeline$ ls
hello.c  hello.o  hello.out  hello.s
dgg6b@portal02:~/CS01/examples/compilationPipeline$
```

```
  401110:       c3                      ret
  401111:       66 66 2e 0f 1f 84 00    data16 cs nopw 0x0(%rax,%rax,1)
  401118:       00 00 00 00
  40111c:       0f 1f 40 00             nopl   0x0(%rax)

0000000000401120 <frame_dummy>:
  401120:       f3 0f 1e fa             endbr64
  401124:       eb 8a                   jmp    4010b0 <register_tm_clones>
  401126:       66 90                   xchg   %ax,%ax

0000000000401128 <main>:
  401128:       50                      push   %rax
  401129:       bf 04 20 40 00          mov    $0x402004,%edi
  40112e:       e8 fd fe ff ff          call   401030 <puts@plt>
  401133:       31 c0                   xor    %eax,%eax
  401135:       59                      pop    %rcx
  401136:       c3                      ret

Disassembly of section .fini:

0000000000401138 <_fini>:
  401138:       f3 0f 1e fa             endbr64
  40113c:       48 83 ec 08             sub    $0x8,%rsp
  401140:       48 83 c4 08             add    $0x8,%rsp
  401144:       c3                      ret

Disassembly of section .rodata:

0000000000402000 <_IO_stdin_used>:
  402000:       01 00                   add    %eax,(%rax)
  402002:       02 00                   add    (%rax),%al
```

Notice the addresses are filled in.

Is this in little endian or big endian?

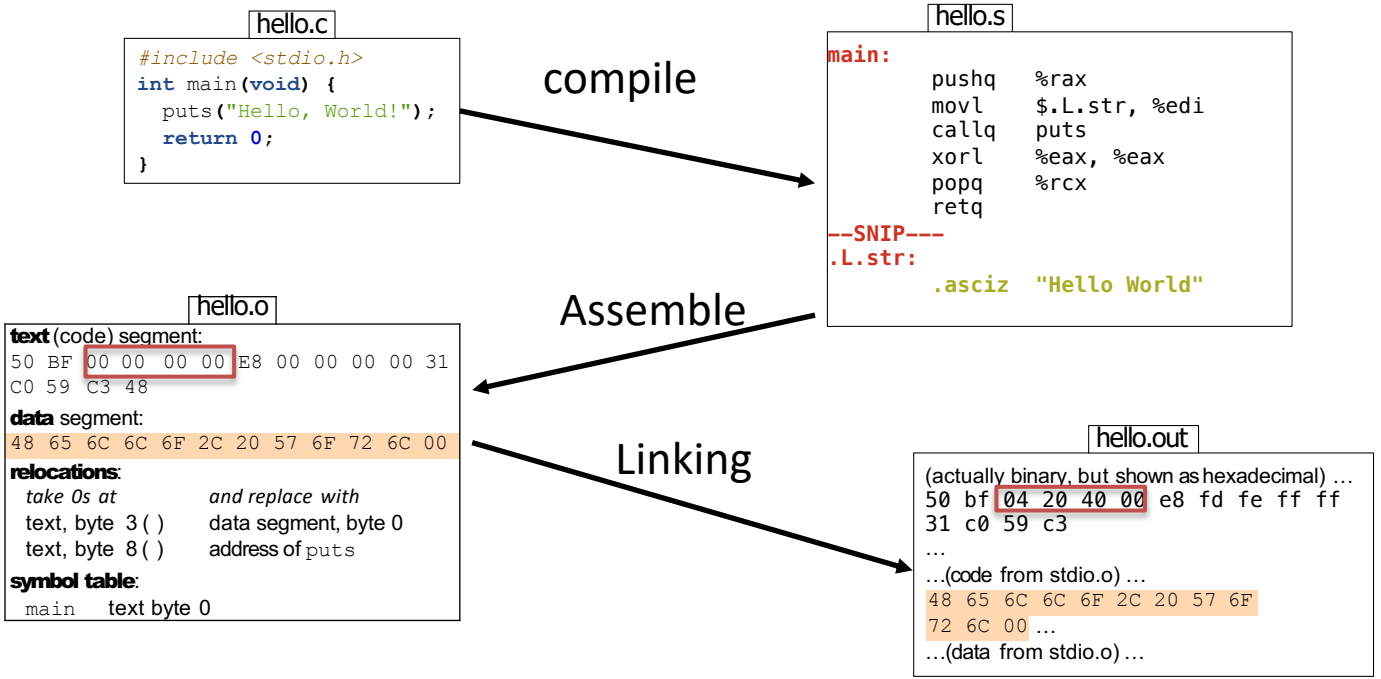
Let's go look at Locations:
401128
402004


000010d0: 00 00 00 00 48 85 c0 74 07 bf 30 40 40 00 ff e0 ....H..t..0@@...
000010e0: c3 66 66 2e 0f 1f 84 00 00 00 00 00 0f 1f 40 00 .ff.........@.
000010f0: f3 0f 1e fa 80 3d 35 2f 00 00 00 75 13 55 48 89 .....=5/...u.UH.
00001100: e5 e8 7a ff ff ff c6 05 23 2f 00 00 01 5d c3 90 ..z.....#/...]..
00001110: c3 66 66 2e 0f 1f 84 00 00 00 00 00 0f 1f 40 00 .ff.........@.
00001120: f3 0f 1e fa eb 8a 66 90 50 bf 04 20 40 00 e8 fd ......f.P.. @...
00001130: fe ff ff 31 c0 59 c3 00 f3 0f 1e fa 48 83 ec 08 ...1.Y......H...
00001140: 48 83 c4 08 c3 00 00 00 00 00 00 00 00 00 00 00 H...............
00001150: 00 00 00 00 00 00 00 00 00
00001160: 00 00 00 00 00 00 00 00 00
00001170: 00 00 00 00 00 00 00 00 00
00001180: 00 00 00 00 00 00 00 00 00
00001190: 00 00 00 00 00 00 00 00 00
000011a0: 00 00 00 00 00 00 00 00 00
000011b0: 00 00 00 00 00 00 00 00 00
000011c0: 00 00 00 00 00 00 00 00 00
000011d0: 00 00 00 00 00 00 00 00 00
000011e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
000011f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
00001200: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
00001210: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
00001220: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
00001230: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
00001240: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
00001250: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
00001260: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
00001270: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
00001280: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
00001290: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
000012a0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
000012b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ....
0000000000401128 <main>:
401128: 50 push %rax
401129: bf 04 20 40 00 mov $0x402004,%edi
40112e: e8 fd fe ff ff call 401030 <puts@plt>
401133: 31 c0 xor %eax,%eax
401135: 59 pop %rcx
401136: c3 ret
Ignore the extra 4 in front for now.

```
00001f30: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00001f40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00001f50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00001f60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00001f70: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00001f80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00001f90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00001fa0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00001fb0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00001fc0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00001fd0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00001fe0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00001ff0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00002000: 01 00 02 00 48 65 6c 6c 6f 20 57 6f 72 6c 64 00  ....Hello World.
00002010: 01 1b 03 3b 2c 00 00 00 04 00 00 00 10 f0 ff ff  ...;,...........
00002020: 70 00 00 00 30 f0 ff ff 48 00 00 00 60 f0 ff ff  p...0...H...`...
00002030: 5c 00 00 00 18 f1 ff ff 98 00 00 00 00 00 00 00  \...............
00002040: 14 00 00 00 00 00 00 00 01 7a 52 00 01 78 10 01  .........zR..x..
00002050: 1b 0c 07 08 90 01 00 00 10 00 00 00 1c 00 00 00  ................
00002060: e0 ef ff ff 26 00 00 00 00 44 07 10 10 00 00 00  ....&....D......
00002070: 30 00 00 00 fc ef ff ff 05 00 00 00 00 00 00 00  0...............
00002080: 24 00 00 00 44 00 00 00 98 ef ff ff 20 00 00 00  $...D....... ...
00002090: 00 0e 10 46 0e 18 4a 0f 0b 77 08 80 00 3f 1a 3b  ...F..J..w...?.;
000020a0: 2a 33 24 22 00 00 00 00 14 00 00 00 6c 00 00 00  *3$".........l...
000020b0: 78 f0 ff ff 0f 00 00 00 00 41 0e 10 4d 0e 08 00  x........A..M...
000020c0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000020d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000020e0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
000020f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00002100: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00002110: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

Great and there is our Hello World at X2004

FROM C TO AN EXECUTABLE



ADDRESSES

ADDRESSES

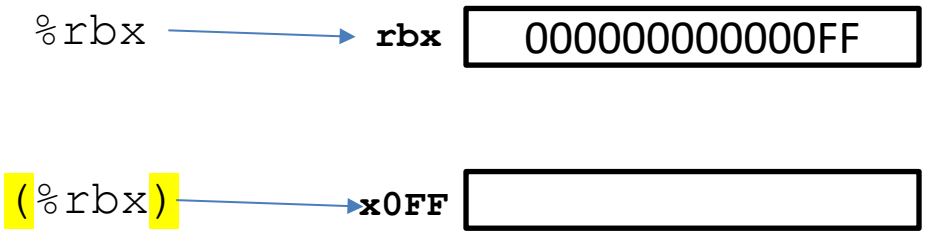
AT&T Syntax	Description
`addq 0x1000, %rax`	*rax ← rax + memory[0x1000]*
`addq $0x1000, %rax`	*rax ← rax + 0x1000*

no $ —> memory address

COMPUTED ADDRESS

NEXT TIME

() s represent value in memory



In the context of toy ISA
(%rbx) is equivalent to M[rbx]

MOVE (MOV) INSTRUCTIONS

Move instruction copies data from one location to another

mov <reg>,<reg>
mov <reg>,<mem>
mov <mem>,<reg>
mov <const>,<reg>
mov <const>,<mem>

AT&T SYNTAX VS INTEL SYNTAX

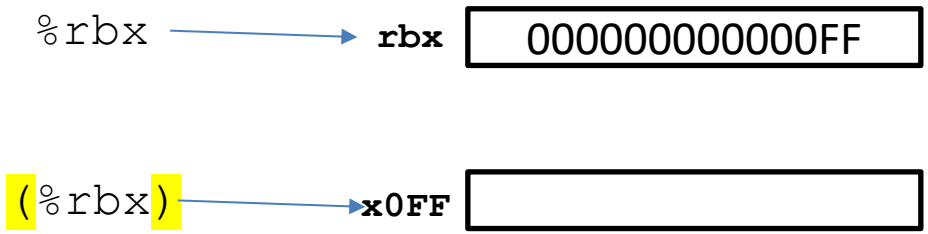
AT&T syntax	Intel Syntax
`movq $42, (%rbx)`	`mov QWORD PTR [rbx], 42`

We will be using AT&T syntax
Remember that the destination last

```
memory[rbx] <- 42
```

AN EXAMPLE WITH MOVE INSTRUCTION

```
movq $42, (%rbx)
// memory[rbx] ←    42
```



() s represent value in memory

COMPUTED ADDRESS

Base + (Index * Scale) + Displacement

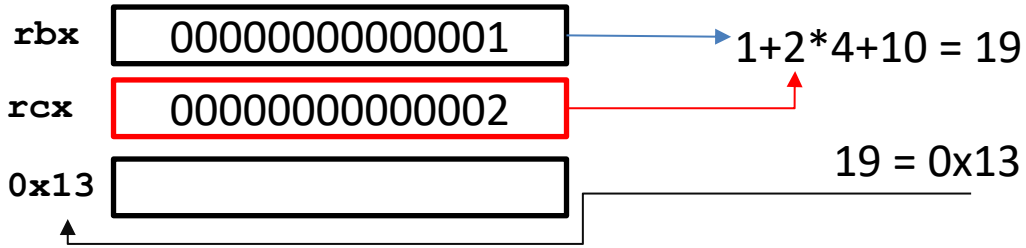
disp(base, index, scale)

AT&T syntax:

```
movq $42, 10(%rbx,%rcx,4)
```

$42 = 0x2A

rbx+(rcx*4)+10



Base + (Index * Scale) + Displacement

disp(base, index, scale)

SCALES ARE LIMITED

```
movq $42, 100(%rbx,%rcx,4)
```

Scales are limited to: 1, 2, 4, 8

disp(base, index, scale)

Base + (Index * Scale) + Displacement

disp(base, index, scale)

AT&T Syntax	Pseudo code
100(%rbx, %rcx, 4)	memory[rbx + rcx*4 + 100]
100(%rbx)	memory[rbx + 100]
100(%rbx,8)	memory[rbx * 8 + 100]
100(,%rbx,8)	memory[rbx * 8 + 100]
100(%rbx,%rcx)	memory[rbx+rcx+100]
100	memory[100]

NOW LET'S LOOK AT OUR PUSH POP EXAMPLE

BUT THIS TIME WITH MOVES

```
dgg6b@portal09:~/CS01/Assemble/lab$ nano stackExamplePart2.s
```

```
.globl main

main:

    movl    $0, -4(%rsp)        # Initialize a 4-byte integer variable at -4(%rsp) with the value 0.
    movl    $3, -8(%rsp)        # Initialize a 4-byte integer variable at -8(%rsp) with the value 3.
    movl    $7, -12(%rsp)       # Initialize a 4-byte integer variable at -12(%rsp) with the value 7.

    movl    -8(%rsp), %eax      # Load the value at -8(%rsp) (which is 3) into %eax register.
    imull   -12(%rsp), %eax     # Multiply %eax by the value at -12(%rsp) (which is 7) and store the result in %eax.

    movl    %eax, -16(%rsp)     # Store the result of the multiplication (in %eax) at -16(%rsp).

    movl    -16(%rsp), %eax     # Load the value at -16(%rsp) (which is the result of the multiplication) into %eax.
```

[Cancelled]

```
dgg6b@portal09:~/CS01/Assemble/lab$ clang stackExamplePart2.s -o stackExamplePart2.out
dgg6b@portal09:~/CS01/Assemble/lab$ lldb stackExamplePart2.out
(lldb) target create "stackExamplePart2.out"
Current executable set to '/u/dgg6b/CS01/Assemble/lab/stackExamplePart2.out' (x86_64).
(lldb) b main
Breakpoint 1: where = stackExamplePart2.out`main, address = 0x0000000000401108
(lldb) run
Process 1109193 launched: '/u/dgg6b/CS01/Assemble/lab/stackExamplePart2.out' (x86_64)
Process 1109193 stopped
* thread #1, name = 'stackExamplePar', stop reason = breakpoint 1.1
    frame #0: 0x0000000000401108 stackExamplePart2.out`main
stackExamplePart2.out`main:
->  0x401108 <+0>:  movl   $0x0, -0x4(%rsp)
    0x401110 <+8>:  movl   $0x3, -0x8(%rsp)
    0x401118 <+16>: movl   $0x7, -0xc(%rsp)
    0x401120 <+24>: movl   -0x8(%rsp), %eax
(lldb)
```

```
dgg6b@portal09:~/CS01/Assemble/lab$ lldb stackExamplePart2.out
(lldb) target create "stackExamplePart2.out"
Current executable set to '/u/dgg6b/CS01/Assemble/lab/stackExamplePart2.out' (x86_64).
(lldb) b main
Breakpoint 1: where = stackExamplePart2.out`main, address = 0x0000000000401108
(lldb) run
Process 1110828 launched: '/u/dgg6b/CS01/Assemble/lab/stackExamplePart2.out' (x86_64)
Process 1110828 stopped
* thread #1, name = 'stackExamplePar', stop reason = breakpoint 1.1
    frame #0: 0x0000000000401108 stackExamplePart2.out`main
stackExamplePart2.out`main:
->  0x401108 <+0>:  movl   $0x0, -0x4(%rsp)
    0x401110 <+8>:  movl   $0x3, -0x8(%rsp)
    0x401118 <+16>: movl   $0x7, -0xc(%rsp)
    0x401120 <+24>: movl   -0x8(%rsp), %eax
(lldb) stepi
Process 1110828 stopped
* thread #1, name = 'stackExamplePar', stop reason = instruction step into
    frame #0: 0x0000000000401110 stackExamplePart2.out`main + 8
stackExamplePart2.out`main:
->  0x401110 <+8>:  movl   $0x3, -0x8(%rsp)
    0x401118 <+16>: movl   $0x7, -0xc(%rsp)
    0x401120 <+24>: movl   -0x8(%rsp), %eax
    0x401124 <+28>: imull  -0xc(%rsp), %eax
(lldb)
```

```
(lldb) target create "stackExamplePart2.out"
Current executable set to '/u/dgg6b/CS01/Assemble/lab/stackExamplePart2.out' (x86_64).
(lldb) b main
Breakpoint 1: where = stackExamplePart2.out`main, address = 0x0000000000401108
(lldb) run
Process 1110828 launched: '/u/dgg6b/CS01/Assemble/lab/stackExamplePart2.out' (x86_64)
Process 1110828 stopped
* thread #1, name = 'stackExamplePar', stop reason = breakpoint 1.1
    frame #0: 0x0000000000401108 stackExamplePart2.out`main
stackExamplePart2.out`main:
->  0x401108 <+0>:  movl   $0x0, -0x4(%rsp)
    0x401110 <+8>:  movl   $0x3, -0x8(%rsp)
    0x401118 <+16>: movl   $0x7, -0xc(%rsp)
    0x401120 <+24>: movl   -0x8(%rsp), %eax
(lldb) stepi
Process 1110828 stopped
* thread #1, name = 'stackExamplePar', stop reason = instruction step into
    frame #0: 0x0000000000401110 stackExamplePart2.out`main + 8
stackExamplePart2.out`main:
->  0x401110 <+8>:  movl   $0x3, -0x8(%rsp)
    0x401118 <+16>: movl   $0x7, -0xc(%rsp)
    0x401120 <+24>: movl   -0x8(%rsp), %eax
    0x401124 <+28>: imull  -0xc(%rsp), %eax
(lldb) memory read -s4 -fA $rsp-4
0x7fffffffe4a4: 0x00000000
0x7fffffffe4a8: 0xf7da8d90
0x7fffffffe4ac: 0x00007fff
0x7fffffffe4b0: 0x00000000
0x7fffffffe4b4: 0x00000000
0x7fffffffe4b8: 0x00401108 stackExamplePart2.out`main
0x7fffffffe4bc: 0x00000000
0x7fffffffe4c0: 0x00000000
(lldb)
```

```
(lldb) target create "stackExamplePart2.out"
Current executable set to '/u/dgg6b/CS01/Assemble/lab/stackExamplePart2.out' (x86_64).
(lldb) b main
Breakpoint 1: where = stackExamplePart2.out`main, address = 0x0000000000401108
(lldb) run
Process 1110828 launched: '/u/dgg6b/CS01/Assemble/lab/stackExamplePart2.out' (x86_64)
Process 1110828 stopped
* thread #1, name = 'stackExamplePar', stop reason = breakpoint 1.1
    frame #0: 0x0000000000401108 stackExamplePart2.out`main
stackExamplePart2.out`main:
->  0x401108 <+0>:  movl   $0x0, -0x4(%rsp)
    0x401110 <+8>:  movl   $0x3, -0x8(%rsp)
    0x401118 <+16>: movl   $0x7, -0xc(%rsp)
    0x401120 <+24>: movl   -0x8(%rsp), %eax
(lldb) stepi
Process 1110828 stopped
* thread #1, name = 'stackExamplePar', stop reason = instruction step into
    frame #0: 0x0000000000401110 stackExamplePart2.out`main + 8
stackExamplePart2.out`main:
->  0x401110 <+8>:  movl   $0x3, -0x8(%rsp)
    0x401118 <+16>: movl   $0x7, -0xc(%rsp)
    0x401120 <+24>: movl   -0x8(%rsp), %eax
    0x401124 <+28>: imull  -0xc(%rsp), %eax
(lldb) memory read -s4 -fA $rsp-4
0x7fffffffe4a4: 0x00000000
0x7fffffffe4a8: 0xf7da8d90
0x7fffffffe4ac: 0x00007fff
0x7fffffffe4b0: 0x00000000
0x7fffffffe4b4: 0x00000000
0x7fffffffe4b8: 0x00401108 stackExamplePart2.out`main
0x7fffffffe4bc: 0x00000000
0x7fffffffe4c0: 0x00000000
(lldb)
```

Top of the stack have the value 0

```
stackExamplePart2.out`main:
->  0x401108 <+0>:  movl   $0x0, -0x4(%rsp)
    0x401110 <+8>:  movl   $0x3, -0x8(%rsp)
    0x401118 <+16>: movl   $0x7, -0xc(%rsp)
    0x401120 <+24>: movl   -0x8(%rsp), %eax
(lldb) stepi
Process 1110828 stopped
* thread #1, name = 'stackExamplePar', stop reason = instruction step into
    frame #0: 0x0000000000401110 stackExamplePart2.out`main + 8
stackExamplePart2.out`main:
->  0x401110 <+8>:  movl   $0x3, -0x8(%rsp)
    0x401118 <+16>: movl   $0x7, -0xc(%rsp)
    0x401120 <+24>: movl   -0x8(%rsp), %eax
    0x401124 <+28>: imull  -0xc(%rsp), %eax
(lldb) memory read -s4 -fA $rsp-4
0x7fffffffe4a4: 0x00000000
0x7fffffffe4a8: 0xf7da8d90
0x7fffffffe4ac: 0x00007fff
0x7fffffffe4b0: 0x00000000
0x7fffffffe4b4: 0x00000000
0x7fffffffe4b8: 0x00401108 stackExamplePart2.out`main
0x7fffffffe4bc: 0x00000000
0x7fffffffe4c0: 0x00000000
(lldb) stepi
Process 1110828 stopped
* thread #1, name = 'stackExamplePar', stop reason = instruction step into
    frame #0: 0x0000000000401118 stackExamplePart2.out`main + 16
stackExamplePart2.out`main:
->  0x401118 <+16>: movl   $0x7, -0xc(%rsp)
    0x401120 <+24>: movl   -0x8(%rsp), %eax
    0x401124 <+28>: imull  -0xc(%rsp), %eax
    0x401129 <+33>: movl   %eax, -0x10(%rsp)
(lldb)
```

```
stackExamplePart2.out`main:
->  0x401110 <+8>:  movl   $0x3, -0x8(%rsp)
    0x401118 <+16>: movl   $0x7, -0xc(%rsp)
    0x401120 <+24>: movl   -0x8(%rsp), %eax
    0x401124 <+28>: imull  -0xc(%rsp), %eax
(lldb) memory read -s4 -fA $rsp-4
0x7fffffffe4a4: 0x00000000
0x7fffffffe4a8: 0xf7da8d90
0x7fffffffe4ac: 0x00007fff
0x7fffffffe4b0: 0x00000000
0x7fffffffe4b4: 0x00000000
0x7fffffffe4b8: 0x00401108 stackExamplePart2.out`main
0x7fffffffe4bc: 0x00000000
0x7fffffffe4c0: 0x00000000
(lldb) stepi
Process 1111684 stopped
* thread #1, name = 'stackExamplePar', stop reason = instruction step into
    frame #0: 0x0000000000401118 stackExamplePart2.out`main + 16
stackExamplePart2.out`main:
->  0x401118 <+16>: movl   $0x7, -0xc(%rsp)
    0x401120 <+24>: movl   -0x8(%rsp), %eax
    0x401124 <+28>: imull  -0xc(%rsp), %eax
    0x401129 <+33>: movl   %eax, -0x10(%rsp)
(lldb) memory read -s4 -fA $rsp-8
0x7fffffffe4a0: 0x00000003
0x7fffffffe4a4: 0x00000000
0x7fffffffe4a8: 0xf7da8d90
0x7fffffffe4ac: 0x00007fff
0x7fffffffe4b0: 0x00000000
0x7fffffffe4b4: 0x00000000
0x7fffffffe4b8: 0x00401108 stackExamplePart2.out`main
0x7fffffffe4bc: 0x00000000
(lldb)
```

RSP hasn't changed
it still points to here

```
    0x401118 <+16>: movl   $0x7, -0xc(%rsp)
    0x401120 <+24>: movl   -0x8(%rsp), %eax
    0x401124 <+28>: imull  -0xc(%rsp), %eax
(lldb) memory read -s4 -fA $rsp-4
0x7fffffffe4a4: 0x00000000
0x7fffffffe4a8: 0xf7da8d90
0x7fffffffe4ac: 0x00007fff
0x7fffffffe4b0: 0x00000000
0x7fffffffe4b4: 0x00000000
0x7fffffffe4b8: 0x00401108 stackExamplePart2.out`main
0x7fffffffe4bc: 0x00000000
0x7fffffffe4c0: 0x00000000
(lldb) stepi
Process 1111684 stopped
* thread #1, name = 'stackExamplePar', stop reason = instruction step into
    frame #0: 0x0000000000401118 stackExamplePart2.out`main + 16
stackExamplePart2.out`main:
->  0x401118 <+16>: movl   $0x7, -0xc(%rsp)
    0x401120 <+24>: movl   -0x8(%rsp), %eax
    0x401124 <+28>: imull  -0xc(%rsp), %eax
    0x401129 <+33>: movl   %eax, -0x10(%rsp)
(lldb) memory read -s4 -fA $rsp-8
0x7fffffffe4a0: 0x00000003
0x7fffffffe4a4: 0x00000000
0x7fffffffe4a8: 0xf7da8d90
0x7fffffffe4ac: 0x00007fff
0x7fffffffe4b0: 0x00000000
0x7fffffffe4b4: 0x00000000
0x7fffffffe4b8: 0x00401108 stackExamplePart2.out`main
0x7fffffffe4bc: 0x00000000
(lldb) register read $rsp
     rsp = 0x00007fffffffe4a8
(lldb)
```

RSP still points here

```
0x7fffffffe4b8: 0x00401108 stackExamplePart2.out`main
0x7fffffffe4bc: 0x00000000
0x7fffffffe4c0: 0x00000000
(lldb) stepi
Process 1111684 stopped
* thread #1, name = 'stackExamplePar', stop reason = instruction step into
    frame #0: 0x0000000000401118 stackExamplePart2.out`main + 16
stackExamplePart2.out`main:
->  0x401118 <+16>: movl   $0x7, -0xc(%rsp)
    0x401120 <+24>: movl   -0x8(%rsp), %eax
    0x401124 <+28>: imull  -0xc(%rsp), %eax
    0x401129 <+33>: movl   %eax, -0x10(%rsp)
(lldb) memory read -s4 -fA $rsp-8
0x7fffffffe4a0: 0x00000003
0x7fffffffe4a4: 0x00000000
0x7fffffffe4a8: 0xf7da8d90
0x7fffffffe4ac: 0x00007fff
0x7fffffffe4b0: 0x00000000
0x7fffffffe4b4: 0x00000000
0x7fffffffe4b8: 0x00401108 stackExamplePart2.out`main
0x7fffffffe4bc: 0x00000000
(lldb) register read $rsp
     rsp = 0x00007fffffffe4a8
(lldb) stepi
Process 1111684 stopped
* thread #1, name = 'stackExamplePar', stop reason = instruction step into
    frame #0: 0x0000000000401120 stackExamplePart2.out`main + 24
stackExamplePart2.out`main:
->  0x401120 <+24>: movl   -0x8(%rsp), %eax
    0x401124 <+28>: imull  -0xc(%rsp), %eax
    0x401129 <+33>: movl   %eax, -0x10(%rsp)
    0x40112d <+37>: movl   -0x10(%rsp), %eax
(lldb)
```

```
    0x401120 <+24>: movl   -0x8(%rsp), %eax
    0x401124 <+28>: imull  -0xc(%rsp), %eax
    0x401129 <+33>: movl   %eax, -0x10(%rsp)
(lldb) memory read -s4 -fA $rsp-8
0x7fffffffe4a0: 0x00000003
0x7fffffffe4a4: 0x00000000
0x7fffffffe4a8: 0xf7da8d90
0x7fffffffe4ac: 0x00007fff
0x7fffffffe4b0: 0x00000000
0x7fffffffe4b4: 0x00000000
0x7fffffffe4b8: 0x00401108 stackExamplePart2.out`main
0x7fffffffe4bc: 0x00000000
(lldb) register read $rsp
     rsp = 0x00007fffffffe4a8
(lldb) stepi
Process 1111684 stopped
* thread #1, name = 'stackExamplePar', stop reason = instruction step into
    frame #0: 0x0000000000401120 stackExamplePart2.out`main + 24
stackExamplePart2.out`main:
->  0x401120 <+24>: movl   -0x8(%rsp), %eax
    0x401124 <+28>: imull  -0xc(%rsp), %eax
    0x401129 <+33>: movl   %eax, -0x10(%rsp)
    0x40112d <+37>: movl   -0x10(%rsp), %eax
(lldb) memory read -s4 -fA $rsp-12
0x7fffffffe49c: 0x00000007
0x7fffffffe4a0: 0x00000003
0x7fffffffe4a4: 0x00000000
0x7fffffffe4a8: 0xf7da8d90
0x7fffffffe4ac: 0x00007fff
0x7fffffffe4b0: 0x00000000
0x7fffffffe4b4: 0x00000000
0x7fffffffe4b8: 0x00401108 stackExamplePart2.out`main
(lldb)
```

7 is now at the top of the stack

```
    0x401129 <+33>: movl   %eax, -0x10(%rsp)
(lldb) memory read -s4 -fA $rsp-8
0x7fffffffe4a0: 0x00000003
0x7fffffffe4a4: 0x00000000
0x7fffffffe4a8: 0xf7da8d90
0x7fffffffe4ac: 0x00007fff
0x7fffffffe4b0: 0x00000000
0x7fffffffe4b4: 0x00000000
0x7fffffffe4b8: 0x00401108 stackExamplePart2.out`main
0x7fffffffe4bc: 0x00000000
(lldb) register read $rsp
     rsp = 0x00007fffffffe4a8
(lldb) stepi
Process 1111684 stopped
* thread #1, name = 'stackExamplePar', stop reason = instruction step into
    frame #0: 0x0000000000401120 stackExamplePart2.out`main + 24
stackExamplePart2.out`main:
->  0x401120 <+24>: movl   -0x8(%rsp), %eax
    0x401124 <+28>: imull  -0xc(%rsp), %eax
    0x401129 <+33>: movl   %eax, -0x10(%rsp)
    0x40112d <+37>: movl   -0x10(%rsp), %eax
(lldb) memory read -s4 -fA $rsp-12
0x7fffffffe49c: 0x00000007
0x7fffffffe4a0: 0x00000003
0x7fffffffe4a4: 0x00000000
0x7fffffffe4a8: 0xf7da8d90
0x7fffffffe4ac: 0x00007fff
0x7fffffffe4b0: 0x00000000
0x7fffffffe4b4: 0x00000000
0x7fffffffe4b8: 0x00401108 stackExamplePart2.out`main
(lldb) register read $rsp
     rsp = 0x00007fffffffe4a8
(lldb)
```

What value will get loaded in the %eax when we execute the next instruction

```
(lldb) stepi
Process 1111684 stopped
* thread #1, name = 'stackExamplePar', stop reason = instruction step into
    frame #0: 0x0000000000401120 stackExamplePart2.out`main + 24
stackExamplePart2.out`main:
->  0x401120 <+24>: movl   -0x8(%rsp), %eax
    0x401124 <+28>: imull  -0xc(%rsp), %eax
    0x401129 <+33>: movl   %eax, -0x10(%rsp)
    0x40112d <+37>: movl   -0x10(%rsp), %eax
(lldb) memory read -s4 -fA $rsp-12
0x7fffffffe49c: 0x00000007
0x7fffffffe4a0: 0x00000003
0x7fffffffe4a4: 0x00000000
0x7fffffffe4a8: 0xf7da8d90
0x7fffffffe4ac: 0x00007fff
0x7fffffffe4b0: 0x00000000
0x7fffffffe4b4: 0x00000000
0x7fffffffe4b8: 0x00401108 stackExamplePart2.out`main
(lldb) register read $rsp
     rsp = 0x00007fffffffe4a8
(lldb) stepi
Process 1111684 stopped
* thread #1, name = 'stackExamplePar', stop reason = instruction step into
    frame #0: 0x0000000000401124 stackExamplePart2.out`main + 28
stackExamplePart2.out`main:
->  0x401124 <+28>: imull  -0xc(%rsp), %eax
    0x401129 <+33>: movl   %eax, -0x10(%rsp)
    0x40112d <+37>: movl   -0x10(%rsp), %eax
    0x401131:       addb   %al, (%rax)
(lldb) register read $eax rax
     eax = 0x00000003
     rax = 0x0000000000000003
(lldb)
```

Remember this was a movl
So only 32 bits

```
(lldb) stepi
Process 1111684 stopped
* thread #1, name = 'stackExamplePar', stop reason = instruction step into
    frame #0: 0x0000000000401120 stackExamplePart2.out`main + 24
stackExamplePart2.out`main:
->  0x401120 <+24>: movl   -0x8(%rsp), %eax
    0x401124 <+28>: imull  -0xc(%rsp), %eax
    0x401129 <+33>: movl   %eax, -0x10(%rsp)
    0x40112d <+37>: movl   -0x10(%rsp), %eax
(lldb) memory read -s4 -fA $rsp-12
0x7fffffffe49c: 0x00000007
0x7fffffffe4a0: 0x00000003
0x7fffffffe4a4: 0x00000000
0x7fffffffe4a8: 0xf7da8d90
0x7fffffffe4ac: 0x00007fff
0x7fffffffe4b0: 0x00000000
0x7fffffffe4b4: 0x00000000
0x7fffffffe4b8: 0x00401108 stackExamplePart2.out`main
(lldb) register read $rsp
     rsp = 0x00007fffffffe4a8
(lldb) stepi
Process 1111684 stopped
* thread #1, name = 'stackExamplePar', stop reason = instruction step into
    frame #0: 0x0000000000401124 stackExamplePart2.out`main + 28
stackExamplePart2.out`main:
->  0x401124 <+28>: imull  -0xc(%rsp), %eax
    0x401129 <+33>: movl   %eax, -0x10(%rsp)
    0x40112d <+37>: movl   -0x10(%rsp), %eax
    0x401131:       addb   %al, (%rax)
(lldb) register read $eax rax
     eax = 0x00000003
     rax = 0x0000000000000003
(lldb)
```

What is the value of eax after the next instruction executes?

```
0x7fffffffe4a0: 0x00000003
0x7fffffffe4a4: 0x00000000
0x7fffffffe4a8: 0xf7da8d90
0x7fffffffe4ac: 0x00007fff
0x7fffffffe4b0: 0x00000000
0x7fffffffe4b4: 0x00000000
0x7fffffffe4b8: 0x00401108 stackExamplePart2.out`main
(lldb) register read $rsp
     rsp = 0x00007fffffffe4a8
(lldb) stepi
Process 1111684 stopped
* thread #1, name = 'stackExamplePar', stop reason = instruction step into
    frame #0: 0x0000000000401124 stackExamplePart2.out`main + 28
stackExamplePart2.out`main:
->  0x401124 <+28>: imull  -0xc(%rsp), %eax
    0x401129 <+33>: movl   %eax, -0x10(%rsp)
    0x40112d <+37>: movl   -0x10(%rsp), %eax
    0x401131:       addb   %al, (%rax)
(lldb) register read $eax rax
     eax = 0x00000003
     rax = 0x0000000000000003
(lldb) stepi
Process 1111684 stopped
* thread #1, name = 'stackExamplePar', stop reason = instruction step into
    frame #0: 0x0000000000401129 stackExamplePart2.out`main + 33
stackExamplePart2.out`main:
->  0x401129 <+33>: movl   %eax, -0x10(%rsp)
    0x40112d <+37>: movl   -0x10(%rsp), %eax
    0x401131:       addb   %al, (%rax)
    0x401133:       addb   %dh, %bl
(lldb) register read $eax
     eax = 0x00000015
(lldb)
```

7x3 = 21

21 = 0x15

```
    0x401131:       addb   %al, (%rax)
(lldb) register read $eax rax
     eax = 0x00000003
     rax = 0x0000000000000003
(lldb) stepi
Process 1111684 stopped
* thread #1, name = 'stackExamplePar', stop reason = instruction step into
    frame #0: 0x0000000000401129 stackExamplePart2.out`main + 33
stackExamplePart2.out`main:
->  0x401129 <+33>: movl   %eax, -0x10(%rsp)
    0x40112d <+37>: movl   -0x10(%rsp), %eax
    0x401131:       addb   %al, (%rax)
    0x401133:       addb   %dh, %bl
(lldb) register read $eax
     eax = 0x00000015
(lldb) stepi
Process 1111684 stopped
* thread #1, name = 'stackExamplePar', stop reason = instruction step into
    frame #0: 0x000000000040112d stackExamplePart2.out`main + 37
stackExamplePart2.out`main:
->  0x40112d <+37>: movl   -0x10(%rsp), %eax
    0x401131:       addb   %al, (%rax)
    0x401133:       addb   %dh, %bl
(lldb) memory read -s4 -fA $rsp-16
0x7fffffffe498: 0x00000015
0x7fffffffe49c: 0x00000007
0x7fffffffe4a0: 0x00000003
0x7fffffffe4a4: 0x00000000
0x7fffffffe4a8: 0xf7da8d90
0x7fffffffe4ac: 0x00007fff
0x7fffffffe4b0: 0x00000000
0x7fffffffe4b4: 0x00000000
(lldb)
```

Stores the result of EAX to the top of the stack

```
    0x401131:       addb   %al, (%rax)
(lldb) register read $eax rax
     eax = 0x00000003
     rax = 0x0000000000000003
(lldb) stepi
Process 1111684 stopped
* thread #1, name = 'stackExamplePar', stop reason = instruction step into
    frame #0: 0x0000000000401129 stackExamplePart2.out`main + 33
stackExamplePart2.out`main:
->  0x401129 <+33>: movl   %eax, -0x10(%rsp)
    0x40112d <+37>: movl   -0x10(%rsp), %eax
    0x401131:       addb   %al, (%rax)
    0x401133:       addb   %dh, %bl
(lldb) register read $eax
     eax = 0x00000015
(lldb) stepi
Process 1111684 stopped
* thread #1, name = 'stackExamplePar', stop reason = instruction step into
    frame #0: 0x000000000040112d stackExamplePart2.out`main + 37
stackExamplePart2.out`main:
->  0x40112d <+37>: movl   -0x10(%rsp), %eax
    0x401131:       addb   %al, (%rax)
    0x401133:       addb   %dh, %bl
(lldb) memory read -s4 -fA $rsp-16
0x7fffffffe498: 0x00000015
0x7fffffffe49c: 0x00000007
0x7fffffffe4a0: 0x00000003
0x7fffffffe4a4: 0x00000000
0x7fffffffe4a8: 0xf7da8d90
0x7fffffffe4ac: 0x00007fff
0x7fffffffe4b0: 0x00000000
0x7fffffffe4b4: 0x00000000
(lldb)
```

Read the value at the top the stack and stores it back into eax

NEXT TIME

Functions and x86 calling convention.

Swap Example with Mov instruction

Swap Example with lea (load effective address) instruction.

Later:

- jmp instruction and condition codes (Building loops)
- switch statements.