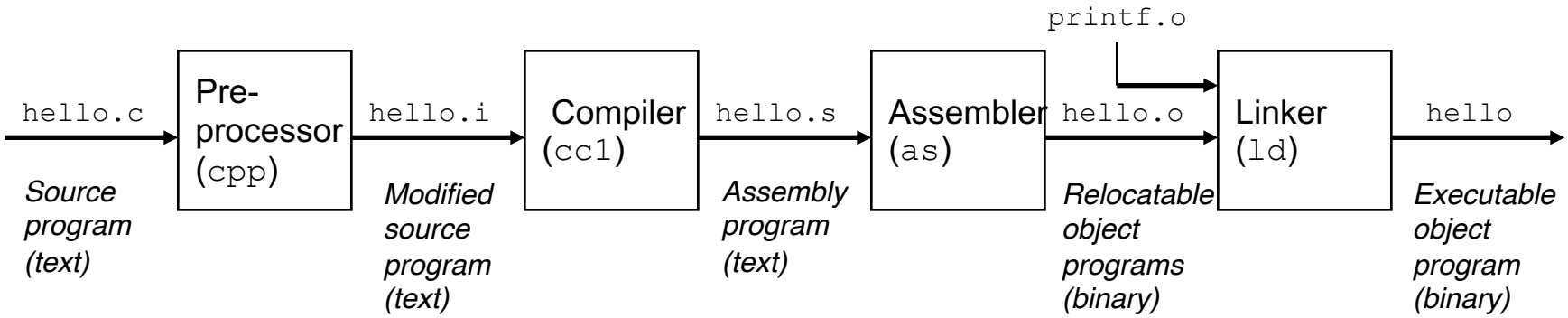
Lecture 4: Introduction to Assembly

CS 105

February 4, 2019

Compilation




```
#include<stdio.h>

int main(int argc,
         char ** argv){

  printf("Hello
          world!\n");
  return 0;
}
```

```
…
int printf(const char *
           restrict,
           ...)
__attribute__((__format_
_ (__printf__, 1, 2)));
…
int main(int argc,
         char ** argv){

  printf("Hello
          world!\n");
  return 0;
}
```

```
pushq   %rbp
movq    %rsp, %rbp
subq  $32, %rsp
leaq  L_.str(%rip), %rax
movl  $0, -4(%rbp)
movl  %edi, -8(%rbp)
movq  %rsi, -16(%rbp)
movq  %rax, %rdi
movb  $0, %al
callq _printf
xorl  %ecx, %ecx
movl  %eax, -20(%rbp)
movl  %ecx, %eax
addq  $32, %rsp
popq  %rbp
retq
```

```
55
48 89 e5
48 83 ec 20
48 8d 05 25 00 00 00
c7 45 fc 00 00 00 00
89 7d f8
48 89 75 f0
48 89 c7
b0 00
e8 00 00 00 00
31 c9
89 45 ec
89 c8
48 83 c4 20
5d
c3
```

gcc Option Summary

- Output options
  - Default is **a.out**
  - **-o <filename>**, output goes to the named file
  - **-c**, compile but do not link; output goes to **program.o**
  - **-S**, assemble only; output goes to **program.s**
  - **-E**, pre-process only; output goes to **program.i**
- Optimization options (uppercase “Oh,” not zero!)
  - **-O, -O1, -O2, -O3, -Og, -Os**
- Debugging option: **-g**, include symbolic debugging information
- Warning option example: **-Wall**
- Library option example: **-lm**, link with the math library

gcc, Typical Compilation

From the Data Lab:

```
$ gcc -O -Wall -lm -o btest bits.c btest.c decl.c tests.c
$ ./btest
```

Easier:

```
$ make
$ ./btest
```

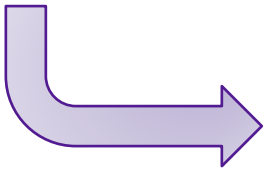
Managing Compilation with **make**

- **make** is a command that reads **Makefile**
- Example extracted from the Datalab **Makefile:**

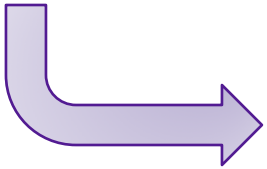
```
# Makefile that builds btest and other helper programs for the CS:APP data lab
#
CC = gcc
CFLAGS = -O -Wall
LIBS = -lm

all: btest fshow ishow

btest: btest.c bits.c decl.c tests.c btest.h bits.h
	$(CC) $(CFLAGS) $(LIBS) -o btest bits.c btest.c decl.c tests.c
```

- Actions taken *only* when sources are newer than target
- **all** is assumed when no target is given on the command line

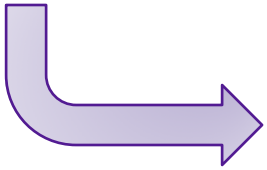
```
#include<stdio.h>

int main(int argc,
         char ** argv){

  printf("Hello
          world!\n");
  return 0;
}
```



```
…
int printf(const char *
           restrict,
           ...)
__attribute__((__format_
_ (__printf__, 1, 2)));
…
int main(int argc,
         char ** argv){

  printf("Hello
          world!\n");
  return 0;
}
```



```
pushq   %rbp
movq    %rsp, %rbp
subq  $32, %rsp
leaq  L_.str(%rip), %rax
movl  $0, -4(%rbp)
movl  %edi, -8(%rbp)
movq  %rsi, -16(%rbp)
movq  %rax, %rdi
movb  $0, %al
callq _printf
xorl  %ecx, %ecx
movl  %eax, -20(%rbp)
movl  %ecx, %eax
addq  $32, %rsp
popq  %rbp
retq
```

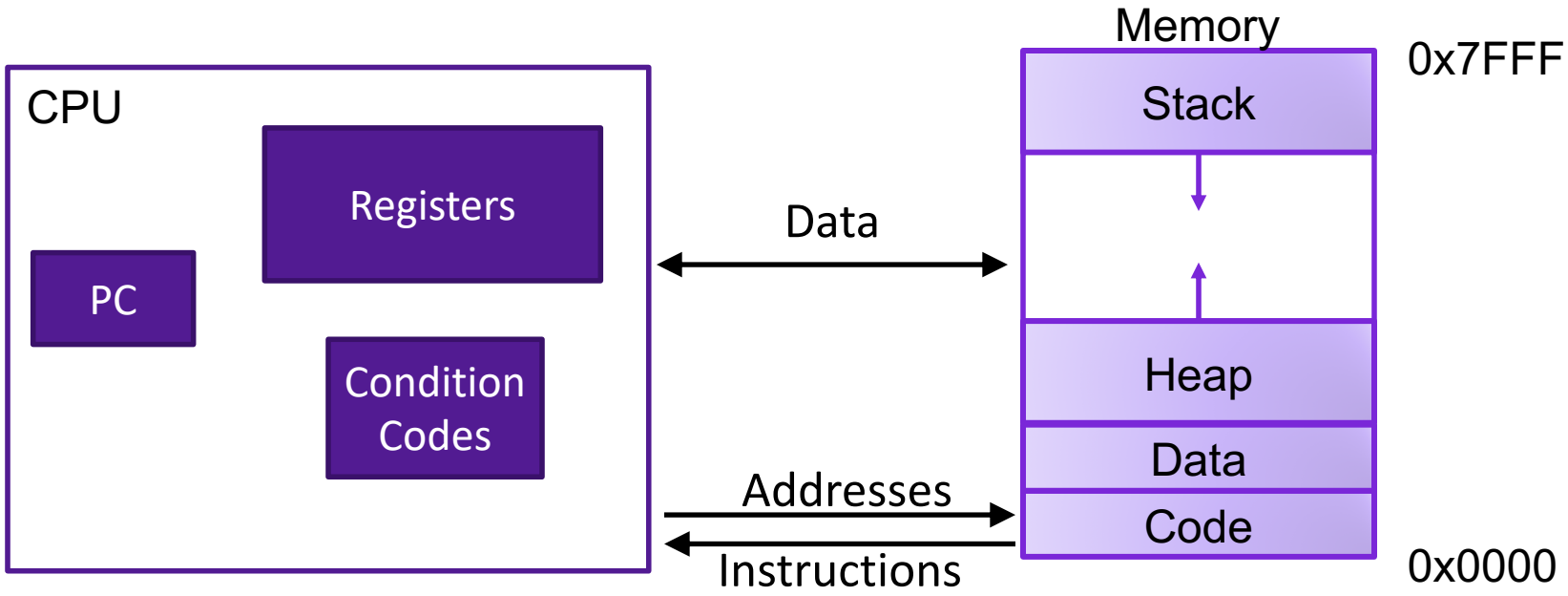


```
55
48 89 e5
48 83 ec 20
48 8d 05 25 00 00 00
c7 45 fc 00 00 00 00
89 7d f8
48 89 75 f0
48 89 c7
b0 00
e8 00 00 00 00
31 c9
89 45 ec
89 c8
48 83 c4 20
5d
c3
```

New Topic: x86-64 Assembly Language

- Evolutionary design, going back to 8086 in 1978
  - Basis for original IBM Personal Computer, 16-bits
- Intel Pentium 4E (2004): 64 bit instruction set
- Other languages are translated into x86 instructions and then executed on the CPU
  - Actual instructions are sequences of bytes
  - We give them mnemonic names

Assembly/Machine Code View



Programmer-Visible State

- PC: Program counter
- 16 Registers
- Condition codes

Memory

- Byte addressable array
- Code and user data
- Stack to support procedures

Program Counter

- Stores the address of the current instruction

- Repeat forever:
  - Fetch instruction at address in PC
  - Execute the instruction
  - Update PC

Assembly Characteristics: Operations

- Transfer data between memory and register
  - Load data from memory into register
  - Store register data into memory
- Perform arithmetic function on register or memory data
- Transfer control
  - Unconditional jumps to/from procedures
  - Conditional branches

X86-64 Integer Registers

%rax	%eax
%rbx	%ebx
%rcx	%ecx
%rdx	%edx
%rsi	%esi
%rdi	%edi
%rsp	%esp
%rbp	%ebp

%r8	%r8d
%r9	%r9d
%r10	%r10d
%r11	%r11d
%r12	%r12d
%r13	%r13d
%r14	%r14d
%r15	%r15d

X86-64 Register Usage Conventions

`%rax`, function result	`%r8`
`%rbx`	`%r9`
`%rcx`, fourth argument	`%r10`
`%rdx`, third argument	`%r11`
`%rsi`, second argument	`%r12`
`%rdi`, first argument	`%r13`
`%rsp`, stack pointer	`%r14`
`%rbp`, base pointer	`%r15`

Callee-saved registers are in yellow

Data Movement Instructions

- MOV source, dest Moves data source->dest

Sizes of C Data Types in x86-64

C declaration	Intel data type	Assembly suffix	Size (bytes)
char	Byte	b	1
short	Word	w	2
int	Double word	l	4
long	Quad word	q	8
char *	Quad word	q	8
float	Single precision	s	4
double	Double precision	l	8

Data Movement Instructions

- MOV source, dest — Move data source->dest
  - `movb` — Move byte
  - `movw` — Move word
  - `movl` — Move double word
  - `movq` — Move quad word

Exercise

Register	Value
%rax	0x100
%rcx	0x1
%rdx	0x3

Memory Address	Value
0x100	0xFF
0x104	0xAB
0x108	0x13

- What are the values of the following operands (assuming register and memory state shown above)?
  1. %rax
  2. 0x104
  3. $0x108
  4. (%rax)
  5. 4(%rax)

`movq` Operand Combinations

	Source	Dest	Src,Dest	C Analog
`movq`	*Imm*	*Reg*	`movq $0x4,%rax`	`temp = 0x4;`
		Mem	`movq $-147,(%rax)`	`*p = -147;`
	Reg	*Reg*	`movq %rax,%rdx`	`temp2 = temp1;`
		Mem	`movq %rax,(%rdx)`	`*p = temp;`
	Mem	*Reg*	`movq (%rax),%rdx`	`temp = *p;`

Cannot do memory-memory transfer with a single instruction

Exercise

- Write a C function void decode1(long *xp, long *yp, long *zp) that will do the same thing as the following assembly code:

```
decode1:
        movq (%rdi), %r8
        movq (%rsi), %rcx
        movq (%rdx), %rax
        movq %r8, (%rsi)
        movq %rcx, (%rdx)
        movq %rax, (%rdi)
        ret
```

C is close to Machine Language

```
*dest = t;
```

```
movq %rax, (%rbx)
```

```
0x40059e:  48 89 03
```

- C Code
  - Store value **t** where designated by **dest**
- Assembly
  - Move 8-byte value to memory
    - Quad words in x86-64 parlance
  - Operands:
    - **t:** Register **%rax**
    - **dest:** Register **%rbx**
    - ***dest:** Memory **M[%rbx]**
- Object Code
  - 3-byte instruction
  - at address **0x40059e**