x86-64 Assembly
CSE 351 Winter 2018

Instructor:
Mark Wyse

Teaching Assistants:
Kevin Bi
Parker DeWilde
Emily Furst
Sarah House
Waylon Huang
Vinny Palaniappan

http://xkcd.com/409/
Administrivia

- Lab 1 due today!
  - Submit `bits.c` and `pointer.c`
- Homework 2 due next Wednesday (1/24)
  - On Integers, Floating Point, and x86-64
Floating point topics

- Fractional binary numbers
- IEEE floating-point standard
- Floating-point operations and rounding
- Floating-point in C

- There are many more details that we won’t cover
  - It’s a 58-page standard...
Floating Point in C

❖ C offers two (well, 3) levels of precision

float 1.0f single precision (32-bit)
double 1.0 double precision (64-bit)
long double 1.0L (“double double” or quadruple) precision (64-128 bits)

❖ #include <math.h> to get INFINITY and NAN constants

❖ Equality (==) comparisons between floating point numbers are tricky, and often return unexpected results, so just avoid them!
Floating Point Conversions in C

- **Casting between int, float, and double changes the bit representation**
  - int → float
    - May be rounded (not enough bits in mantissa: 23)
    - Overflow impossible
  - int or float → double
    - Exact conversion (all 32-bit ints representable)
  - long → double
    - Depends on word size (32-bit is exact, 64-bit may be rounded)
  - double or float → int
    - Truncates fractional part (rounded toward zero)
    - “Not defined” when out of range or NaN: generally sets to Tmin (even if the value is a very big positive)
Number Representation Really Matters

- **1991**: Patriot missile targeting error
  - clock skew due to conversion from integer to floating point
- **1996**: Ariane 5 rocket exploded ($1 billion)
  - overflow converting 64-bit floating point to 16-bit integer
- **2000**: Y2K problem
  - limited (decimal) representation: overflow, wrap-around
- **2038**: Unix epoch rollover
  - Unix epoch = seconds since 12am, January 1, 1970
  - signed 32-bit integer representation rolls over to TMin in 2038
- **Other related bugs:**
  - 1982: Vancouver Stock Exchange (truncation instead of rounding)
  - 1994: Intel Pentium FDIV (floating point division) HW bug ($475 million)
  - 1997: USS Yorktown “smart” warship stranded: divide by zero
  - 1998: Mars Climate Orbiter crashed: unit mismatch ($193 million)
Roadmap

C:

```c

car *c = malloc(sizeof(car));
c->miles = 100;
c->gals = 17;
float mpg = get_mpg(c);
free(c);
```

Java:

```java

Car c = new Car();
c.setMiles(100);
c.setGals(17);
float mpg = c.getMPG();
```

Assembly language:

```
get_mpg:
    pushq  %rbp
    movq   %rsp, %rbp
    ...
    popq   %rbp
    ret
```

Machine code:

```
0111010000011000
100011010000010000000010
1000100111000010
110000001111101000001111
```

Computer system:

```
Windows 10
```

```
OS X Yosemite
```

Memory & data
Integers & floats
x86 assembly
Procedures & stacks
Executables
Arrays & structs
Processes
Virtual memory
Memory & caches
Java vs. C

OS:
Basics of Machine Programming & Architecture

❖ What is an ISA (Instruction Set Architecture)?
❖ A brief history of Intel processors and architectures
❖ Intro to Assembly and Registers
Translation

What makes programs run fast(er)?

- User program in C
  - C file

- Compile Time
  - C compiler
  - Assembler
  - .exe file

- Run Time
  - Hardware
HW Interface Affects Performance

Source code
Different applications or algorithms

Compiler
Perform optimizations, generate instructions

Architecture
Instruction set

Hardware
Different implementations

C Language

Program A

Program B

Your program

GCC

Clang

x86-64

ARMv8 (AArch64/A64)

Intel Pentium 4

Intel Core i7

AMD Ryzen

AMD Epyc

Intel Xeon

ARM Cortex-A53

Apple A7
Definitions

❖ **Architecture (ISA):** The parts of a processor design that one needs to understand to write assembly code
  ▪ “What is directly visible to software”

❖ **Microarchitecture:** Implementation of the architecture
  ▪ CSE/EE 469, 470

❖ Are the following part of the architecture?
  ▪ Number of registers?
  ▪ How about CPU frequency?
  ▪ Cache size? Memory size?
Instruction Set Architectures

❖ The ISA defines:
  ▪ The system’s state (e.g. registers, memory, program counter)
  ▪ The instructions the CPU can execute
  ▪ The effect that each of these instructions will have on the system state
Instruction Set Philosophies

❖ **Complex Instruction Set Computing (CISC):** Add more and more elaborate and specialized instructions as needed
  ▪ Lots of tools for programmers to use, but hardware must be able to handle all instructions
  ▪ x86-64 is CISC, but only a small subset of instructions encountered with Linux programs

❖ **Reduced Instruction Set Computing (RISC):** Keep instruction set small and regular
  ▪ Easier to build fast hardware
  ▪ Let software do the complicated operations by composing simpler ones
General ISA Design Decisions

❖ Instructions
  ▪ What instructions are available? What do they do?
  ▪ How are they encoded?

❖ Registers
  ▪ How many registers are there?
  ▪ How wide are they?

❖ Memory
  ▪ How do you specify a memory location?
# Mainstream ISAs

## x86
- **Designer**: Intel, AMD
- **Bits**: 16-bit, 32-bit and 64-bit
- **Introduced**: 1978 (16-bit), 1985 (32-bit), 2003 (64-bit)
- **Design**: CISC
- **Type**: Register-memory
- **Encoding**: Variable (1 to 15 bytes)
- **Endianness**: Little

## ARM
- **Design**: ARM Holdings
- **Bits**: 32-bit, 64-bit
- **Introduced**: 1985; 31 years ago
- **Design**: RISC
- **Type**: Register-Register
- **Encoding**: AArch64/A64 and AArch32/A32 use 32-bit instructions, T32 (Thumb-2) uses mixed 16- and 32-bit instructions. ARMv7 *user-space* compatibility\(^1\)
- **Endianness**: Bi (little as default)

## MIPS
- **Designer**: MIPS Technologies, Inc.
- **Bits**: 64-bit (32→64)
- **Introduced**: 1981; 35 years ago
- **Design**: RISC
- **Type**: Register-Register
- **Encoding**: Fixed
- **Endianness**: Bi

---

**Macbooks & PCs**
(Core i3, i5, i7, M)

**x86-64 Instruction Set**

**Smartphone-like devices**
(iPhone, iPad, Raspberry Pi)

**ARM Instruction Set**

**Digital home & networking equipment**
(Blu-ray, PlayStation 2)

**MIPS Instruction Set**

---

\(^1\) For user-space compatibility, see examples like T32 (Thumb-2) that use mixed 16- and 32-bit instructions.
# Intel/AMD x86 Evolution: Milestones

<table>
<thead>
<tr>
<th>Name</th>
<th>Date</th>
<th>Transistors</th>
<th>MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>8086</td>
<td>1978</td>
<td>29K</td>
<td>5-10</td>
</tr>
<tr>
<td>First 16-bit Intel processor. Basis for IBM PC &amp; DOS</td>
<td>1 MB address space</td>
<td></td>
<td></td>
</tr>
<tr>
<td>386</td>
<td>1985</td>
<td>275K</td>
<td>16-33</td>
</tr>
<tr>
<td>First 32-bit Intel processor, referred to as IA32</td>
<td>Added “flat addressing,” capable of running Unix</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pentium (P5)</td>
<td>1993</td>
<td>3.2M</td>
<td>60</td>
</tr>
<tr>
<td>First superscalar IA32</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Athlon (K7)</td>
<td>1999</td>
<td>22M</td>
<td>500-2333</td>
</tr>
<tr>
<td>First desktop processor with 1 GHz clock (at roughly same time as Pentium III)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Athlon 64 (K8)</td>
<td>2003</td>
<td>106M</td>
<td>1600-3200</td>
</tr>
<tr>
<td>First x86-64 processor architecture</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pentium 4E</td>
<td>2004</td>
<td>125M</td>
<td>2800-3800</td>
</tr>
<tr>
<td>First 64-bit Intel x86 processor</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Intel/AMD x86 Evolution: Milestones

<table>
<thead>
<tr>
<th>Name</th>
<th>Date</th>
<th>Transistors</th>
<th>MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core 2</td>
<td>2006</td>
<td>291M</td>
<td>1060-3500</td>
</tr>
<tr>
<td>First multi-core Intel Processor</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Core i7</td>
<td>2008</td>
<td>731M</td>
<td>1700-3900</td>
</tr>
<tr>
<td>Four cores</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AMD Phenom (K10)</td>
<td>2008</td>
<td>758M</td>
<td>1800-2600</td>
</tr>
<tr>
<td>First “true” quad core, with all cores on same silicon die</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Core i7 (Coffee Lake)</td>
<td>2017</td>
<td>?</td>
<td>2800-4700</td>
</tr>
<tr>
<td>Ryzen 7 (Zen)</td>
<td>2017</td>
<td>4.8B</td>
<td>3000-4200</td>
</tr>
</tbody>
</table>
Technology Scaling

Stuttering
- Transistors per chip, ‘000
- Clock speed (max), MHz
- Thermal design power*, w

Chip introduction dates, selected

Transistors bought per $, m

Breakdown!

Sources: Intel; press reports; Bob Colwell; Linley Group; IB Consulting; The Economist

*Maximum safe power consumption

http://www.economist.com/technology-quarterly/2016-03-12/after-moores-law
Transition to 64-bit

- Intel attempted radical shift from IA32 to IA64 (2001)
  - Completely new architecture (Itanium)
  - Execute IA32 code only as legacy
  - Performance disappointing

  - x86-64, evolutionary step from IA32

- Intel pursued IA64
  - Couldn’t admit its mistake with Itanium

- Intel announces “EM64T” extension to IA32 (2004)
  - Extended Memory 64-bit Technology
  - Nearly identical to AMD64!
Assembly Programmer’s View

- **Programmer-visible state**
  - **PC**: the Program Counter (%rip in x86-64)
    - Address of next instruction
  - Named registers
    - Together in “register file”
    - Heavily used program data
  - Condition codes
    - Store status information about most recent arithmetic operation
    - Used for conditional branching

- **Memory**
  - Byte-addressable array
  - Code and user data
  - Includes *the Stack* (for supporting procedures)
Three Basic Kinds of Instructions

1) Transfer data between memory and register
   - Load data from memory into register
     - \( %\text{reg} = \text{Mem}[\text{address}] \)
   - Store register data into memory
     - \( \text{Mem}[\text{address}] = %\text{reg} \)

2) Perform arithmetic operation on register or memory data
   - \( c = a + b; \quad z = x << y; \quad i = h \& g; \)

3) Control flow: what instruction to execute next
   - Unconditional jumps to/from procedures
   - Conditional branches

Remember: Memory is indexed just like an array of bytes!
x86-64 Assembly “Data Types”

- Integral data of 1, 2, 4, or 8 bytes
  - Data values
  - Addresses (untyped pointers)
- Floating point data of 4, 8, 10 or 2x8 or 4x4 or 8x2
  - Different registers for those (e.g. \%xmm1, \%ymm2)
  - Come from extensions to x86 (SSE, AVX, ...)
- No aggregate types such as arrays or structures
  - Just contiguously allocated bytes in memory
- Two common syntaxes
  - “AT&T”: used by our course, slides, textbook, gnu tools, ...
  - “Intel”: used by Intel documentation, Intel tools, ...
  - Must know which you’re reading

Not covered in 351
What is a Register?

❖ A location in the CPU that stores a small amount of data, which can be accessed very quickly (once every clock cycle)

❖ Registers have *names*, not *addresses*
  ▪ In assembly, they start with `%` (*e.g. %rsi*)

❖ Registers are at the heart of assembly programming
  ▪ They are a precious commodity in all architectures, but *especially* x86
### x86-64 Integer Registers – 64 bits wide

<table>
<thead>
<tr>
<th>%rax</th>
<th>%eax</th>
</tr>
</thead>
<tbody>
<tr>
<td>%rbx</td>
<td>%ebx</td>
</tr>
<tr>
<td>%rcx</td>
<td>%ecx</td>
</tr>
<tr>
<td>%rdx</td>
<td>%edx</td>
</tr>
<tr>
<td>%rsi</td>
<td>%esi</td>
</tr>
<tr>
<td>%rdi</td>
<td>%edi</td>
</tr>
<tr>
<td>%rsp</td>
<td>%esp</td>
</tr>
<tr>
<td>%rbp</td>
<td>%ebp</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>%r8</th>
<th>%r8d</th>
</tr>
</thead>
<tbody>
<tr>
<td>%r9</td>
<td>%r9d</td>
</tr>
<tr>
<td>%r10</td>
<td>%r10d</td>
</tr>
<tr>
<td>%r11</td>
<td>%r11d</td>
</tr>
<tr>
<td>%r12</td>
<td>%r12d</td>
</tr>
<tr>
<td>%r13</td>
<td>%r13d</td>
</tr>
<tr>
<td>%r14</td>
<td>%r14d</td>
</tr>
<tr>
<td>%r15</td>
<td>%r15d</td>
</tr>
</tbody>
</table>

- Can reference low-order 4 bytes (also low-order 2 & 1 bytes)
### Some History: IA32 Registers – 32 bits wide

<table>
<thead>
<tr>
<th>Register</th>
<th>Function</th>
<th>Name Origin</th>
<th>16-bit virtual registers (backwards compatibility)</th>
<th>Name Origin (mostly obsolete)</th>
</tr>
</thead>
<tbody>
<tr>
<td>%eax</td>
<td>accumulate</td>
<td>%ax</td>
<td>%ah</td>
<td>%al</td>
</tr>
<tr>
<td>%ecx</td>
<td>counter</td>
<td>%cx</td>
<td>%ch</td>
<td>%cl</td>
</tr>
<tr>
<td>%edx</td>
<td>data</td>
<td>%dx</td>
<td>%dh</td>
<td>%dl</td>
</tr>
<tr>
<td>%ebx</td>
<td>base</td>
<td>%bx</td>
<td>%bh</td>
<td>%bl</td>
</tr>
<tr>
<td>%esi</td>
<td>source index</td>
<td>%si</td>
<td></td>
<td></td>
</tr>
<tr>
<td>%edi</td>
<td>destination index</td>
<td>%di</td>
<td></td>
<td></td>
</tr>
<tr>
<td>%esp</td>
<td>stack pointer</td>
<td>%sp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>%ebp</td>
<td>base pointer</td>
<td>%bp</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### General Purpose
- **%eax**: 32-bit accumulator register.
- **%ecx**: 32-bit counter register.
- **%edx**: 32-bit data register.
- **%ebx**: 32-bit base register.
- **%esi**: 32-bit source index register.
- **%edi**: 32-bit destination index register.
- **%esp**: 32-bit stack pointer.
- **%ebp**: 32-bit base pointer.

16-bit virtual registers (backwards compatibility) and Name Origin (mostly obsolete) are shown in the diagram.
### Memory vs. Registers

<table>
<thead>
<tr>
<th>Memory</th>
<th>vs.</th>
<th>Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addresses</td>
<td>vs.</td>
<td>Names</td>
</tr>
<tr>
<td>0x7FFFD024C3DC</td>
<td>%rdi</td>
<td></td>
</tr>
<tr>
<td>Big</td>
<td>vs.</td>
<td>Small</td>
</tr>
<tr>
<td>~ 8 GB</td>
<td>(16 x 8 B) = 128 B</td>
<td></td>
</tr>
<tr>
<td>Slow</td>
<td>vs.</td>
<td>Fast</td>
</tr>
<tr>
<td>~50-100 ns</td>
<td>sub-nanosecond timescale</td>
<td></td>
</tr>
<tr>
<td>Dynamic</td>
<td>vs.</td>
<td>Static</td>
</tr>
<tr>
<td>Can “grow” as needed while program runs</td>
<td>fixed number in hardware</td>
<td></td>
</tr>
</tbody>
</table>
Operand types

❖ **Immediate**: Constant integer data
  - Examples: $0x400, −533
  - Like C literal, but prefixed with ‘$’
  - Encoded with 1, 2, 4, or 8 bytes depending on the instruction

❖ **Register**: 1 of 16 integer registers
  - Examples: %rax, %r13
  - But %rsp reserved for special use
  - Others have special uses for particular instructions

❖ **Memory**: Consecutive bytes of memory at a computed address
  - Simplest example: (%rax)
  - Various other “address modes”
Summary

❖ **x86-64** is a complex instruction set computing (CISC) architecture

❖ **Registers** are named locations in the CPU for holding and manipulating data
  - x86-64 uses 16 64-bit wide registers

❖ Assembly operands include immediates, registers, and data at specified memory locations
Floating Point Summary

❖ Floats also suffer from the fixed number of bits available to represent them
  ▪ Can get overflow/underflow
  ▪ “Gaps” produced in representable numbers means we can lose precision, unlike \texttt{ints}
    • Some “simple fractions” have no exact representation (\textit{e.g.} 0.2)
    • “Every operation gets a slightly wrong result”

❖ Floating point arithmetic not associative or distributive
  ▪ Mathematically equivalent ways of writing an expression may compute different results

❖ \textbf{Never} test floating point values for equality!

❖ \textbf{Careful} when converting between \texttt{ints} and \texttt{floats}!
Floating Point Summary

- Converting between integral and floating point data types *does* change the bits
  - Floating point rounding is a HUGE issue!
    - Limited mantissa bits cause inaccurate representations
    - Floating point arithmetic is NOT associative or distributive