Machine-level Representation of Programs

Jin-Soo Kim
(jinsoo.kim@snu.ac.kr)
Systems Software &
Architecture Lab.
Seoul National University

Spring 2019

Program? ≈ Recipe!

준비시간 :10분, 조리시간 :10분

재료
라면 1개, 스프 1봉지, 오징어 1/4마리, 호박 1/4개, 양파 1/2개, 양배추 1장, 당근 1/4개, 물 3컵(600cc)

Ingredients ≈ Data

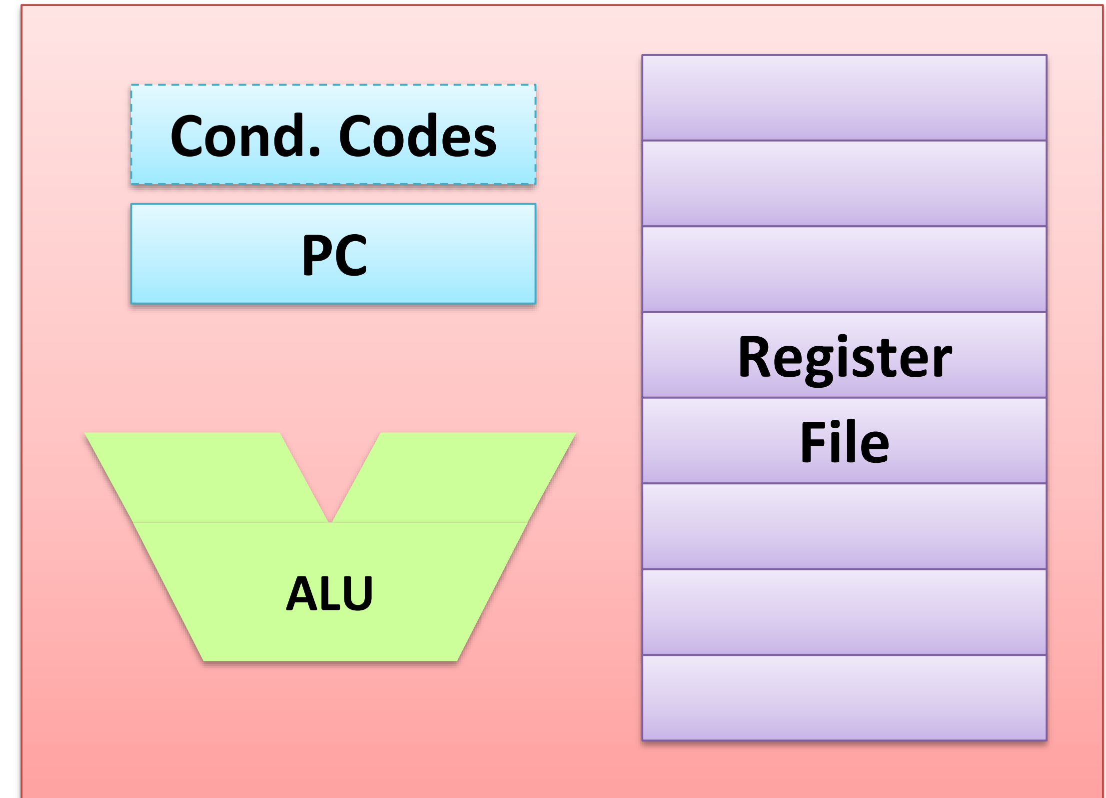
만드는 법
1.오징어는 껍질을 벗기고 깨끗하게 씻어 칼집으로 모양을 낸다.
2.호박, 양파, 양배추는 모두 채썬다.
3.냄비에 물 3컵을 붓고 끓인다.
4.물이 끓으면 스프를 넣고 오징어와 야채를 넣어 충분히 맛이 우러나도록 5분 정도 끓여준다.
5.끓으면 면을 넣어 익힌다.

Directions ≈ Instructions

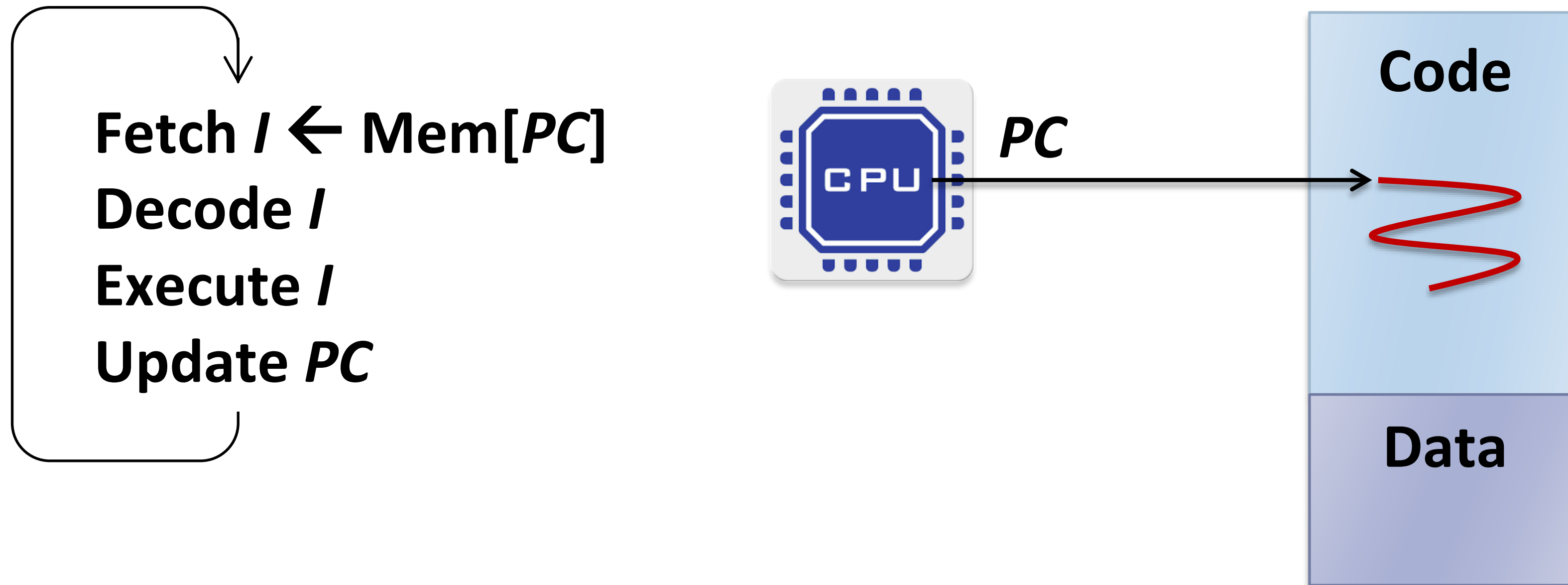
Source: http://user.chol.com/~yugenie/yo/jjambong.html

CPU

- Central Processing Unit
  - PC (Program Counter)
    - Address of next instruction
    - Called "EIP" (IA-32) or "RIP" (x86-64)
  - Register file
    - Heavily used program data
  - Condition codes
    - Store status information about most recent arithmetic operation
    - Used for conditional branching



The (Dumb) Life of CPU



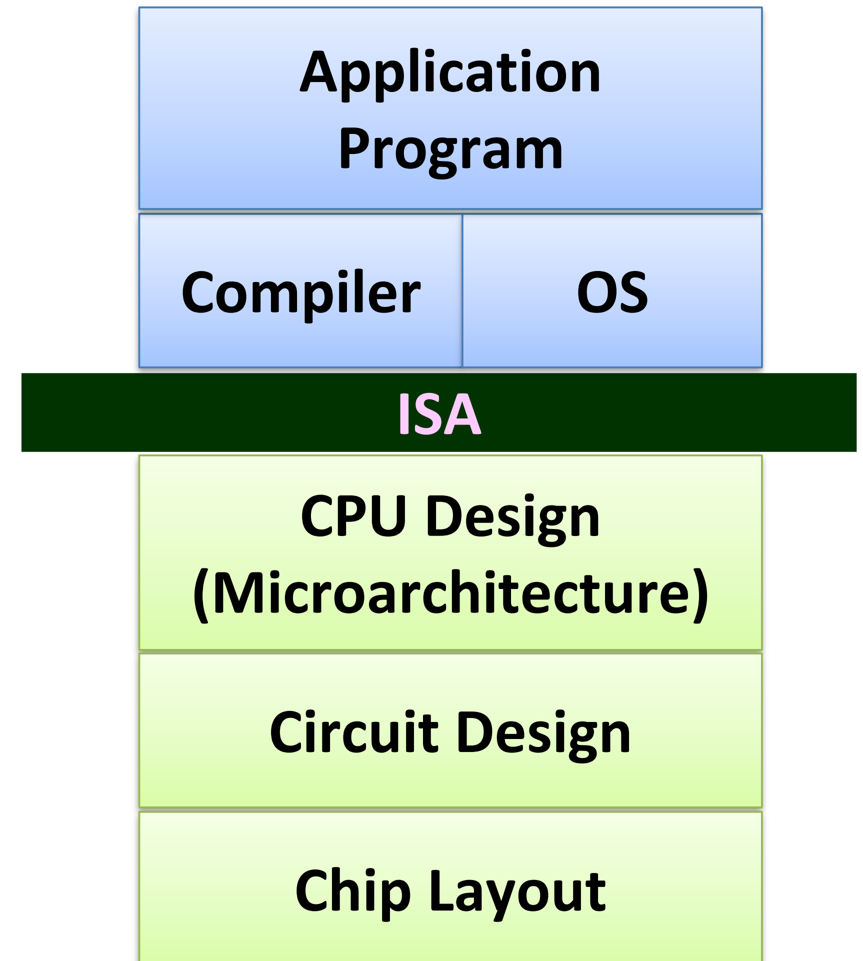
Architecture

> "**the attributes of a system as seen by the programmer, i.e., the conceptual structure and functional behavior, as distinct from the organization of the data flow and controls, the logical design, and the physical implementation**"
>
> *-- Amdahl, Blaauw, and Brooks, Architecture of the IBM System/360, IBM Journal of Research and Development, April 1964.*

- The visible interface between software and hardware
- What the user (OS, compiler, …) needs to know to reason about how the machine behaves
- Abstracted from the details of how it may accomplish its task

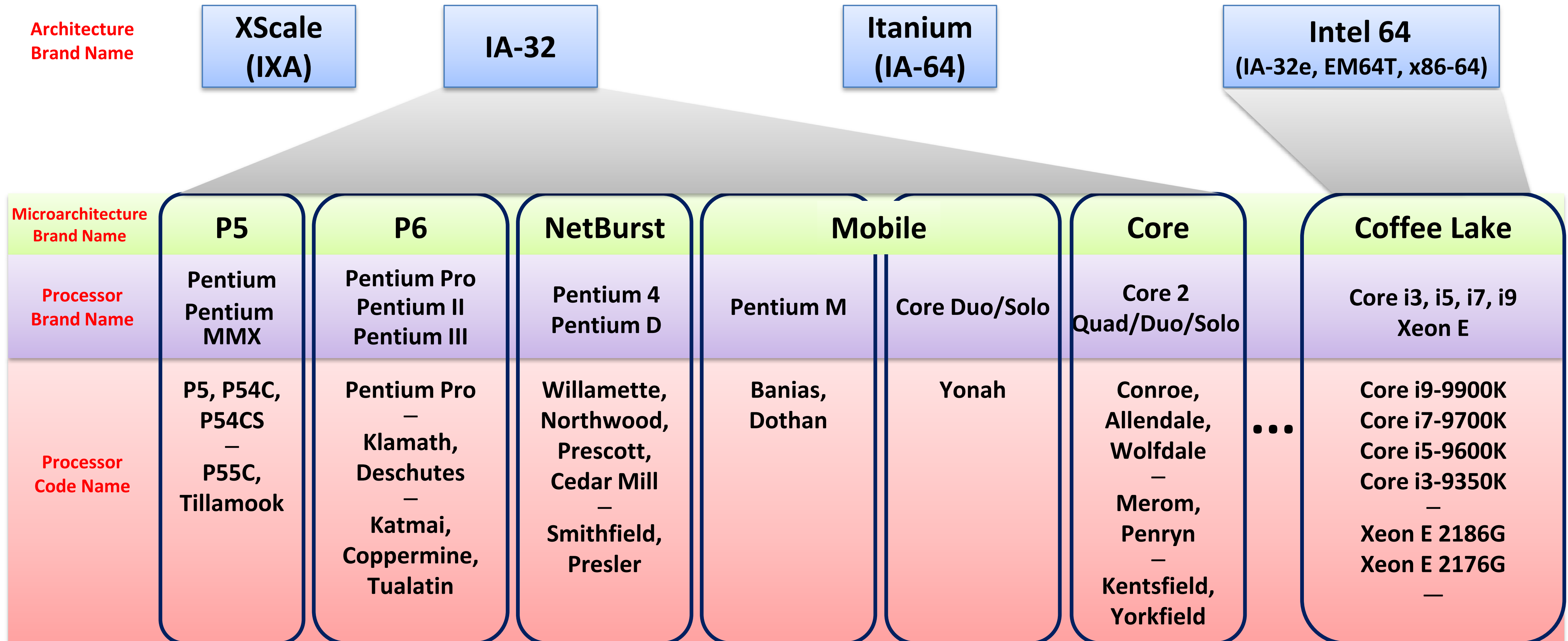
Instruction Set Architecture (ISA)

- Above: how to program machine
  - Processors execute instructions in sequence
- Below: what needs to be built
  - Use variety of tricks to make it run fast



- Instruction set
- Processor registers
- Memory addressing modes
- Data types and representations
- Byte ordering, …

Intel Architecture



Intel x86 Processors

- Evolutionary design
  - Starting in 1978 with 8086
  - Added more features as time goes on
  - Still support old features, although obsolete
  - Totally dominate laptop/desktop/server market

- Complex Instruction Set Computer (CISC)
  - Many different instructions with many different formats
  - Hard to match performance of Reduced Instruction Set Computer (RISC)
  - But Intel has done just that!
    - In terms of speed. Less so for low power

AMD: x86 Clones

- Historically
  - AMD has followed just behind Intel
  - A little bit slower, a lot cheaper
- Then
  - Recruited top circuit designers from Digital Equipment Corp. and other downward trending companies
  - Built Opteron: touch competitor to Pentium 4
  - Developed x86-64, their own extension to 64 bits
- Recent years,
  - Intel leads the world in semiconductor technology
  - AMD has fallen behind, but recently strikes back with Ryzen (2017)

Intel's 64-bit History

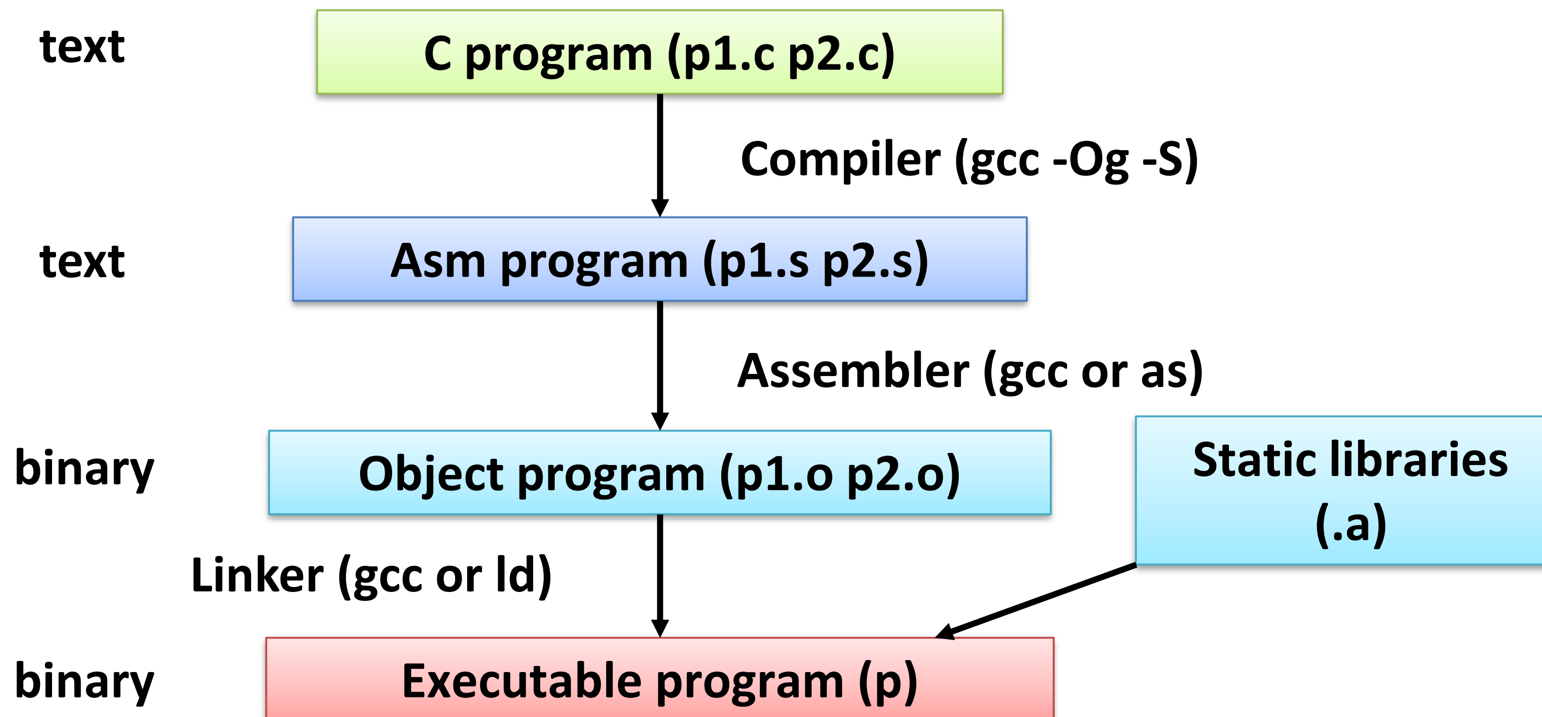
- 2001: Intel attempts radical shift from IA32 to IA64
  - Totally different architecture (Itanium)
  - Executes IA32 code only as legacy
  - Performance disappointing
- 2003: AMD steps in with evolutionary solution
  - x86-64 (now called "AMD64" or "Intel 64")
- 2004: Intel announces EM64T extension to IA32
  - Extended Memory 64-bit Technology
  - Almost identical to x86-64!
- All but low-end x86 processors support x86-64
  - But, lots of code still runs in 32-bit mode

2019 State of the Art

- Desktop: Core i9-9900K (Coffee Lake)
  - 9th generation Intel Core i9 processor with 14nm
  - 8 cores (16 threads) @ 3.6 – 5.0 GHz, 95W
  - Max 128 GB Memory (DDR4-2666), 16 MB L3 Cache
  - Integrated Graphics (UHD 630)
  - 16 PCIe 3.0 lanes
- Server: Xeon Platinum 8180M Processor (Skylake)
  - 6th generation Intel Xeon Scalable Processor with 14nm
  - 28 cores (56 threads) @ 2.5 – 3.8 GHz, 205W
  - Max 1.5 TB Memory (DDR4-2666), 38.5 MB L3 Cache
  - 48 PCIe 3.0 lanes

Turning C into Object Code

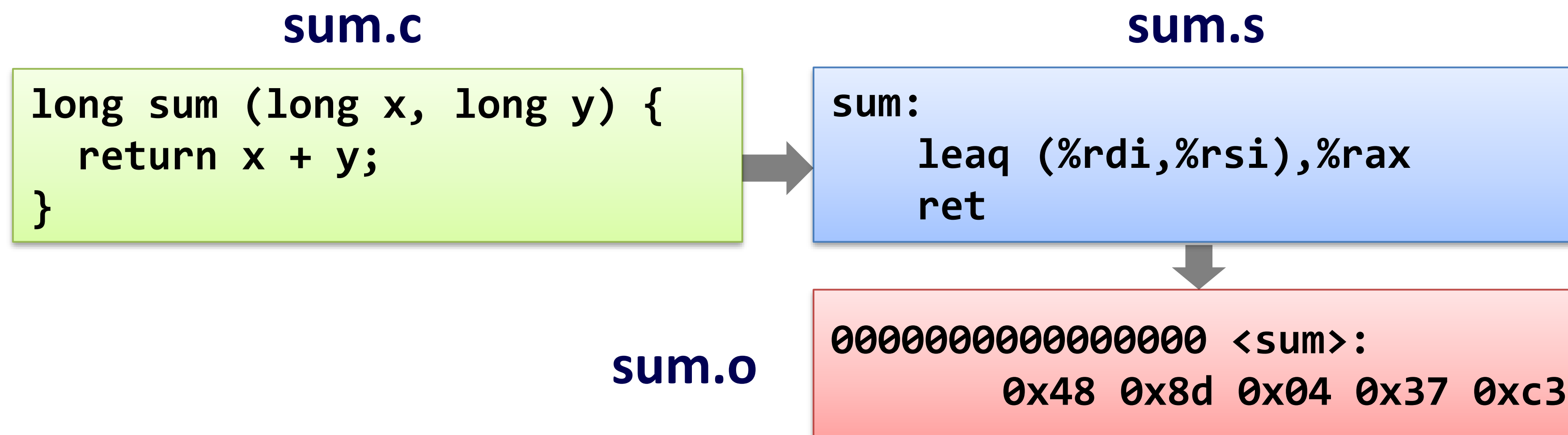
- `gcc -Og p1.c p2.c -o p`
  - Use basic optimizations (-Og)
  - Put resulting binary in file p



Compiling into Machine Code

- Machine code (or binary code)
  - The byte-level programs that a processor executes
- Assembly code
  - A text representation of machine code

- `gcc -Og -S sum.c`



Object Program

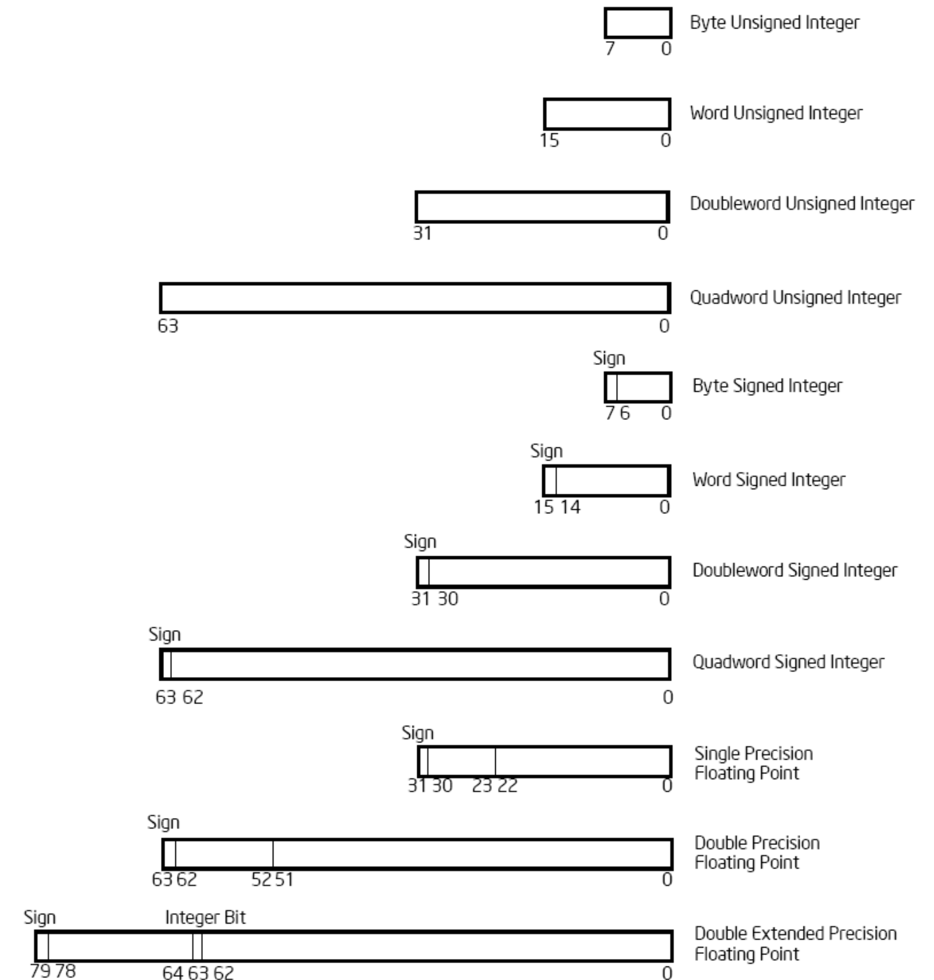
- **Assembler**
  - Translates `.s` into `.o`
  - Binary encoding of each instruction
  - Nearly-complete image of executable code
  - Missing linkages between code in different files
- **Linker**
  - Resolves references between files
  - Combines with static run-time libraries
    - e.g. code for `malloc()`, `printf()`, etc.
  - Some libraries are dynamically linked
    - Linking occurs when program begins execution

```
0x4004d6 <sum>:
0x48
0x8d
0x04
0x37
0xc3
```

- **Total of 5 bytes**
- **Each instruction 1 or 4 bytes**
- **Starts at address 0x4004d6**

Assembly: Data Types

- "integer" data of 1, 2, 4, or 8 bytes
  - Data values
  - Addresses (untyped pointers)
  - (cf.) In x86-64, a "word" means 16-bit data
- Floating point data of 4, 8, or 10 bytes
- No aggregated types such as arrays or structures
  - Just contiguously allocated bytes in memory



Assembly: Operations

- Perform an arithmetic or logical function on register or memory data
- Transfer data between memory and register
  - Load data from memory into register
  - Store register data into memory
- Transfer control
  - Unconditional jump
  - Conditional branch
  - Procedure call and return

Disassembling

- Disassembler: **`objdump -d sum.o`**
  - Useful tool for examining object code
  - Analyzes bit pattern of series of instructions
  - Produces approximate rendition of assembly code
  - Can be run on either `a.out` (complete executable) or `.o` (object code) file

```
sum.o:      file format elf64-x86-64

Disassembly of section .text:

0000000000000000 <sum>:
   0:  48 8d 04 37     lea    (%rdi,%rsi,1),%rax
   4:  c3              retq
```

Disassembling with gdb

- Disassemble procedure, `sum`

```
$ gcc -Og -g sum.c main.c
$ gdb a.out
(gdb) disassemble sum
Dump of assembler code for function sum:
0x00000000004004d6 <+0>:    lea   (%rdi,%rsi,1),%rax
0x00000000004004da <+4>:    retq
End of assembler dump.
```

- Examine 5 bytes starting at `sum`

```
(gdb) x/5 sum
0x4004d6 <sum>: 0x48    0x8d    0x04    0x37    0xc3
```

Whose Assembler?

- Intel/Microsoft differs from ATT/GAS (GNU Assembler)
  - Operands listed in opposite order

```
mov Dest, Src                              movq Src, Dest
```

  - Constants not preceded by '$', denote hex with 'h' at end

```
sub rsp,100h                               subq $0x100,%rsp
```

  - Operand size indicated by operands rather than operator suffix

```
cmp qword ptr [rbp-8],0                    cmpq $0,-8(%rbp)
```

  - Addressing format shows effective address computation

```
mov rax, qword ptr [rax*4+100h]            movq $0x100(,%rax,4),%rax
```

```
(gdb) set disassemble-flavor intel
(gdb) disassemble sum
0x00000000004004d6 <+0>:    lea  rax,[rdi+rsi*1]
0x00000000004004da <+4>:    ret
```

x86-64 Reference

- Intel 64 and IA-32 Architectures Software Developer's Manuals
  - Volume 1: Basic architecture
  - Volume 2: Instruction set reference, A-Z
  - Volume 3: System programming guide
  - Volume 4: Model-specific registers

- Available online:
  - http://software.intel.com/en-us/articles/intel-sdm

Byte Ordering

Memory Model

- Physical memory
  - DRAM chips can read/write 4, 8, 16 bits
  - DRAM modules can read/write 64 bits

- Programmer's view of memory
  - Conceptually, a very large array of bytes
  - Stored-program computers: keeps program codes and data in memory
  - Running programs share the physical memory
  - OS handles memory allocation and management

Machine Words

- Each computer has a "word size"
  - Nominal size of integer-valued data
    - Including addresses (= pointer size)
  - Until recently, most machines used 32-bit (4-byte) words
    - Limits addresses to 4 GB
    - Becoming too small for memory-intensive applications
  - Increasingly, machines have 64-bit (8-byte) word size
    - Potential address space $\simeq 18.4 \times 10^{18}$ bytes (18 EB)
    - x86-64 machines support 48-bit addresses: 256 TB
  - Machines support multiple data formats
    - Fractions or multiples of word size
    - Always integral number of bytes

Data Types in C

C Data Type	Typical 32-bit	Typical 64-bit	x86-64
`char`	1	1	1
`short`	2	2	2
`int`	4	4	4
`long`	4	8	8
`long long`	8	8	8
`float`	4	4	4
`double`	8	8	8
`long double`	-	-	10/16
`pointer`	4	8	8

Byte Ordering

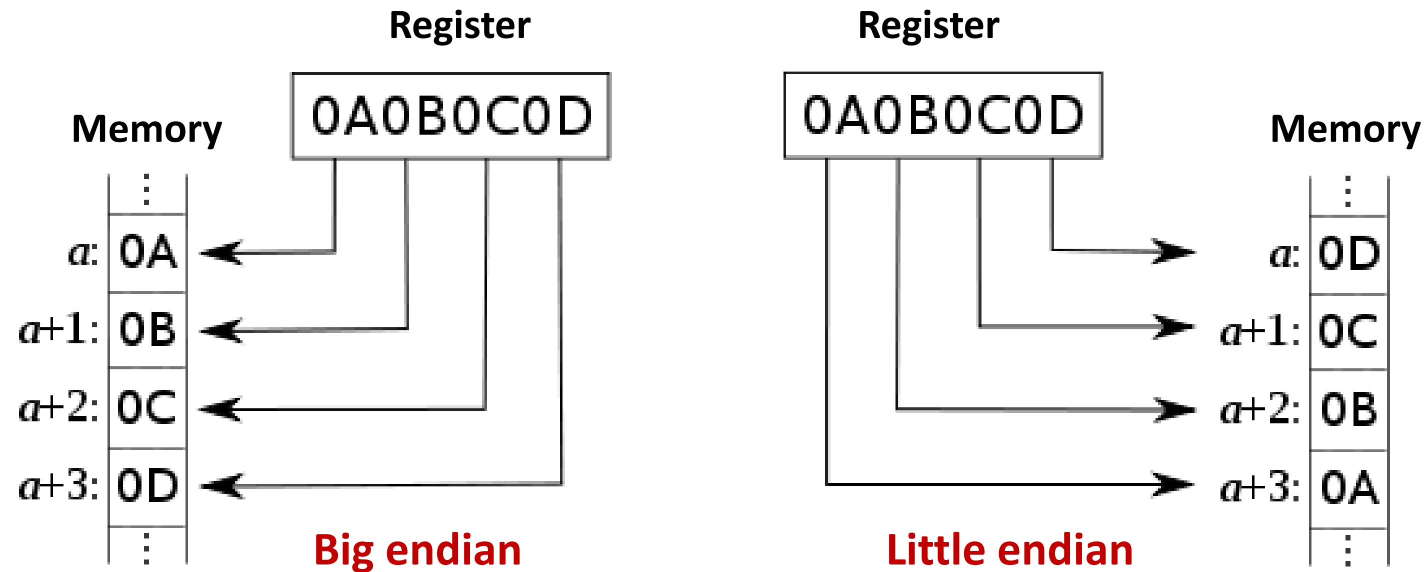
- How are the bytes within a multi-byte word ordered in memory?

- Conventions
  - Big endian: Sun, PowerPC Mac, Internet
  - Little endian: Intel x86, ARM running Android & iOS

- Note:
  - Alpha and PowerPC can run in either mode, with the byte ordering convention determined when the chip is powered up
  - Problem when the binary data is communicated over a network between different machines

Big vs. Little Endian

- Big endian
  - Least significant byte has highest address
- Little endian
  - Least significant byte has lowest address



Example 1

- Disassembly
  - Text representation of binary machine code
  - Generated by program that reads the machine code
- Example fragment

Address	Instruction Code	Assembly Rendition
8048365:	5b	pop %ebx
8048366:	81 c3 ab 12 00 00	add $0x12ab,%ebx
804836c:	83 bb 28 00 00 00 00	cmpl $0x0,0x28(%ebx)

- Deciphering numbers:

Value:	0x12ab
Pad to 32 bits:	0x000012ab
Split into bytes:	00 00 12 ab
Reverse:	ab 12 00 00

Example 2

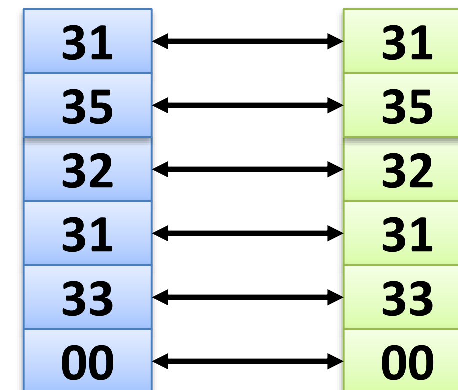
- What is the output of this program?
  - Solaris/SPARC: ?
  - Linux/x86-64: ?

```
#include   <stdio.h>

union {
    int             i;
    unsigned char   c[4];
} u;

int main ()  {
    u.i = 0x12345678;
    printf (“%x %x %x %x\n”,
            u.c[0], u.c[1], u.c[2], u.c[3]);
}
```

Representing Strings

- Strings in C
  - Represented by array of characters
  - Each character encoded in ASCII format
    - Standard 7-bit encoding of character set
    - Character '0' has code 0x30
    - Digit *i* has code 0x30 + *I*
  - String should be null-terminated
    - Final character = 0x00

char S[6] = "15213";



- Compatibility
  - Byte ordering not an issue

Summary

- It's all about bits & bytes
  - Numbers, programs, text, …
- Different machines follow different conventions
  - Word size
  - Byte ordering
  - Representations (integer, floating-point)
- When programming, be aware of
  - Type casting & mixed signed/unsigned expressions
  - Overflow
  - Error propagation
  - Byte ordering