Machine-Level Representation of Programs: Instruction set architectures

Yipeng Huang

Rutgers University

March 5, 2024

Table of contents

Quizzes and programming assignments

Short quiz 5

- Due Wednesday. All about floats.

Programming assignment 3

- Due Friday.

Reading assignments

CS:APP Chapters 3.1-3.4

Table of contents

The IEEE 754 number line

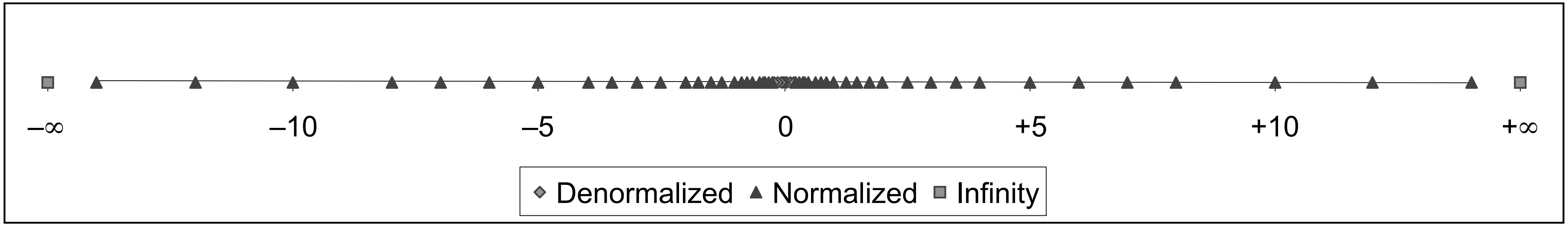


Figure: Full picture of number line for floating point values. Image credit CS:APP

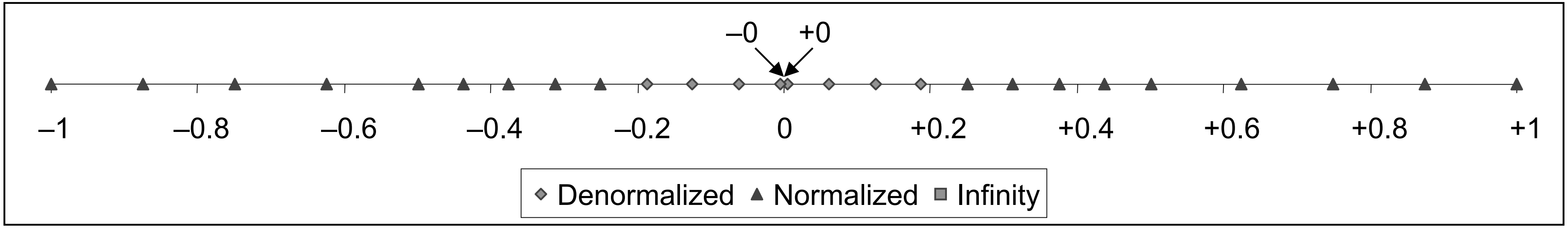


Figure: Zoomed in number line for floating point values. Image credit CS:APP

Floats: Summary

	normalized	denormalized
value of number	$(-1)^s \times M \times 2^E$	$(-1)^s \times M \times 2^E$
E	E = exp-bias	E = -bias + 1
bias	$2^{k-1} - 1$	$2^{k-1} - 1$
exp	$0 < exp < (2^k - 1)$	$exp = 0$
M	M = 1.frac	M = 0.frac
	M has implied leading 1	M has leading 0
	greater range	greater precision
	large magnitude numbers	small magnitude numbers
	denser near origin	evenly spaced

Table: Summary of normalized and denormalized numbers

Deep understanding 1: Why is exp field encoded using bias?

exp field needs to encode both positive and negative exponents.

Why not just use one of the signed integer formats? 2's complement, 1s' complement, signed magnitude?

Deep understanding 1: Why is exp field encoded using bias?

exp field needs to encode both positive and negative exponents.

Why not just use one of the signed integer formats? 2's complement, 1s' complement, signed magnitude?

Answer: allows easy comparison of magnitudes by simply comparing bits.

Deep understanding 1: Why is exp field encoded using bias?

exp field needs to encode both positive and negative exponents.

Why not just use one of the signed integer formats? 2's complement, 1s' complement, signed magnitude?

Answer: allows easy comparison of magnitudes by simply comparing bits.

Consider hypothetical 8-bit floating point format (from the textbook)

1-bit sign, $k = 4$-bit exp, 3-bit frac.

What is the decimal value of 0b1_0110_111?

$(-1)^1 \cdot (1.111_2) \cdot 2^{-1} = -1.875 \cdot 2^{-1}$

$E = exp - bias = 6 - (2^{k-1} - 1) = 6 - (2^3 - 1) = -1$

What is the decimal value of 0b1_0111_000?

$(-1)^1 \cdot (1.000) \cdot 2^0 = -1.000 = -2.000 \cdot 2^{-1}$

$E = exp - bias = 7 - 7 = 2^0$

Deep understanding 1: Why is exp field encoded using bias?

exp field needs to encode both positive and negative exponents.

Why not just use one of the signed integer formats? 2's complement, 1s' complement, signed magnitude?

Answer: allows easy comparison of magnitudes by simply comparing bits.

Consider hypothetical 8-bit floating point format (from the textbook)

1-bit sign, $k = 4$-bit exp, 3-bit frac.

What is the decimal value of 0b1_0110_111?

-1.875×2^{-1}

What is the decimal value of 0b1_0111_000?

-2.000×2^{-1}

Deep understanding 2: Why have denormalized numbers?

Why not just continue normalized number scheme down to smallest numbers around zero?

Answer: makes sure that smallest increments available are maintained around zero.

Suppose denormalized numbers NOT used.

What is the decimal value of 0b0_0000_001?

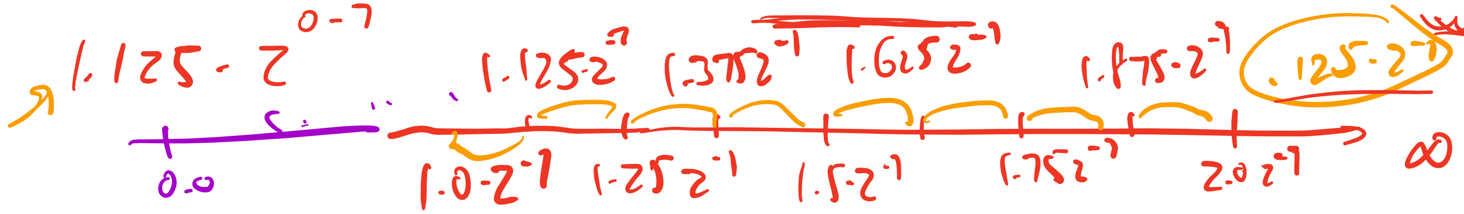
1.125×2^{-7}

What is the decimal value of 0b0_0000_111?

1.875×2^{-7}

What is the decimal value of 0b0_0001_000?

2.000×2^{-7}



Deep understanding 2: Why have denormalized numbers?

Why not just continue normalized number scheme down to smallest numbers around zero?

Answer: makes sure that smallest increments available are maintained around zero.

Suppose denormalized numbers ARE used.

What is the decimal value of 0b0_0000_001?

0.125×2^{-6}

What is the decimal value of 0b0_0000_111?

0.875×2^{-6}

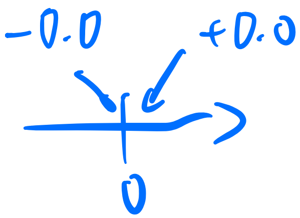
What is the decimal value of 0b0_0001_000?

1.000×2^{-6}

Floats: Special cases

number class	when it arises	exp field	frac field
+0 / -0		0	0
+infinity / -infinity	overflow or division by 0	$2^k - 1$	0
NaN not-a-number	illegal ops. such as $\sqrt{-1}$, inf-inf, inf*0	$2^k - 1$	non-0

Table: Summary of special cases



$\frac{1}{\frac{1}{\infty}} = \infty$ $\frac{1}{\frac{1}{-\infty}} = -\infty$

$\frac{1}{\infty} = +0.0$ $\frac{1}{-\infty} = -0.0$

Table of contents

How to multiply scientific notation?

Recall: $log(x \times y) = log(x) + log(y)$

Floating point multiplication

FP Multiplication

- $(-1)^{s1}\ M1\ \ 2^{E1}\ \ \times\ \ (-1)^{s2}\ M2\ \ 2^{E2}$
- Exact Result: $(-1)^{s}\ M\ \ 2^{E}$
  - Sign s: s1 ^ s2
  - Significand M: M1 x M2
  - Exponent E: E1 + E2
- Fixing
  - If M ≥ 2, shift M right, increment E
  - If E out of range, overflow
  - Round M to fit **`frac`** precision
- Implementation
  - Biggest chore is multiplying significands

Table of contents

Computer organization

Layer cake: remember the first day of class, we discussed what are parts of a computer?

- Society
- Human beings
- Applications
- Algorithms
- High-level programming languages ← Python, Java
- Interpreters
- Low-level programming languages ← C
- Compilers
- Architectures
- Microarchitectures
- Sequential/combinational logic
- Transistors
- Semiconductors

SW / HW

ISA instruction set architecture

abstraction

Stored program computer

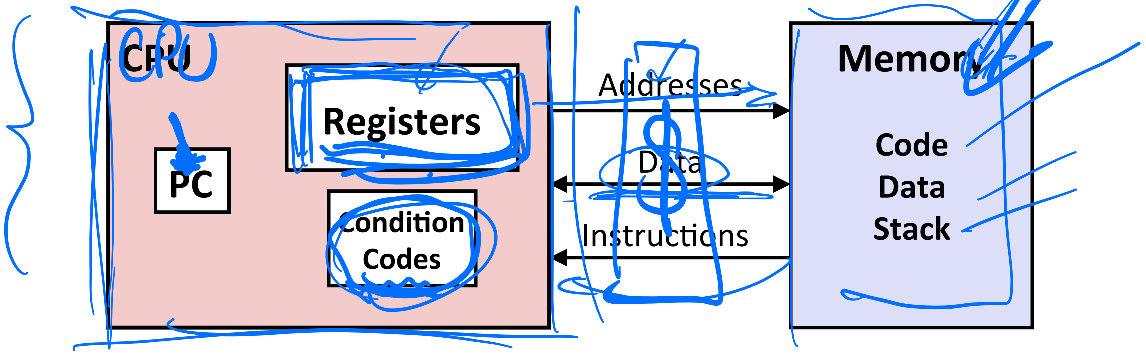
Stored program:

Instructions reside in memory, loaded as needed.

von Neumann architecture:

Data and instructions share same connection to memory.

Assembly/Machine Code View



Programmer-Visible State

- **PC: Program counter**
  - Address of next instruction
  - Called "RIP" (x86-64)
- **Register file**
  - Heavily used program data
- **Condition codes**
  - Store status information about most recent arithmetic or logical operation
  - Used for conditional branching

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

Memory hierarchy

	Capacity	Access speed
Internet		
Tape	250Pb	
Hard drives	16TB	2Mb/s
Solid state drives	4TB	2Gb/s
DRAM	8Gb - 1Tb+	8Gb/s
Last-level cache	64Mb	
Level-1 cache	1Mb	
Registers	1Kb	

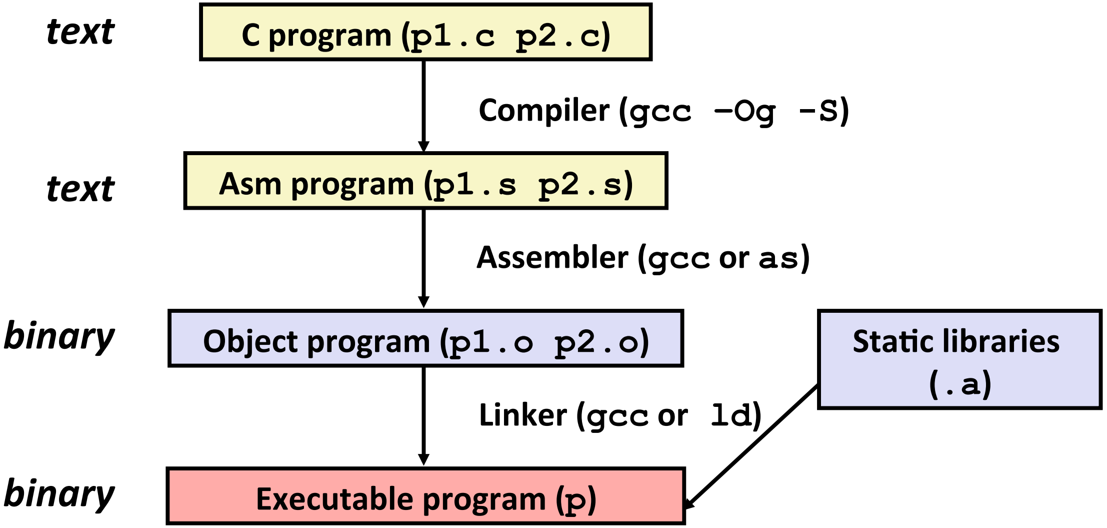
- Registers (.25ns; 4GHz => .25e-9s)

Table of contents

Unraveling the compilation chain

Turning C into Object Code

- Code in files `p1.c p2.c`
- Compile with command: `gcc –Og p1.c p2.c -o p`
  - Use basic optimizations (`–Og`) [New to recent versions of GCC]
  - Put resulting binary in file `p`



Assembly

Human readable machine code

- Very limited
- Not much control flow
- Any more complex functionality is built up
- for loops, while loops, turn into assembly sequence

Choice of what assembly to experiment with

- MIPS
- ARM
- x86 / x86-64 (not ideal for teaching, but it allows us to experiment on ilab)

Assembly instructions

Instructions for the microarchitecture

- Binary streams that tell an electronic circuit what to do
- Fetch, decode, execute, memory, writeback

A preview of microarchitecture

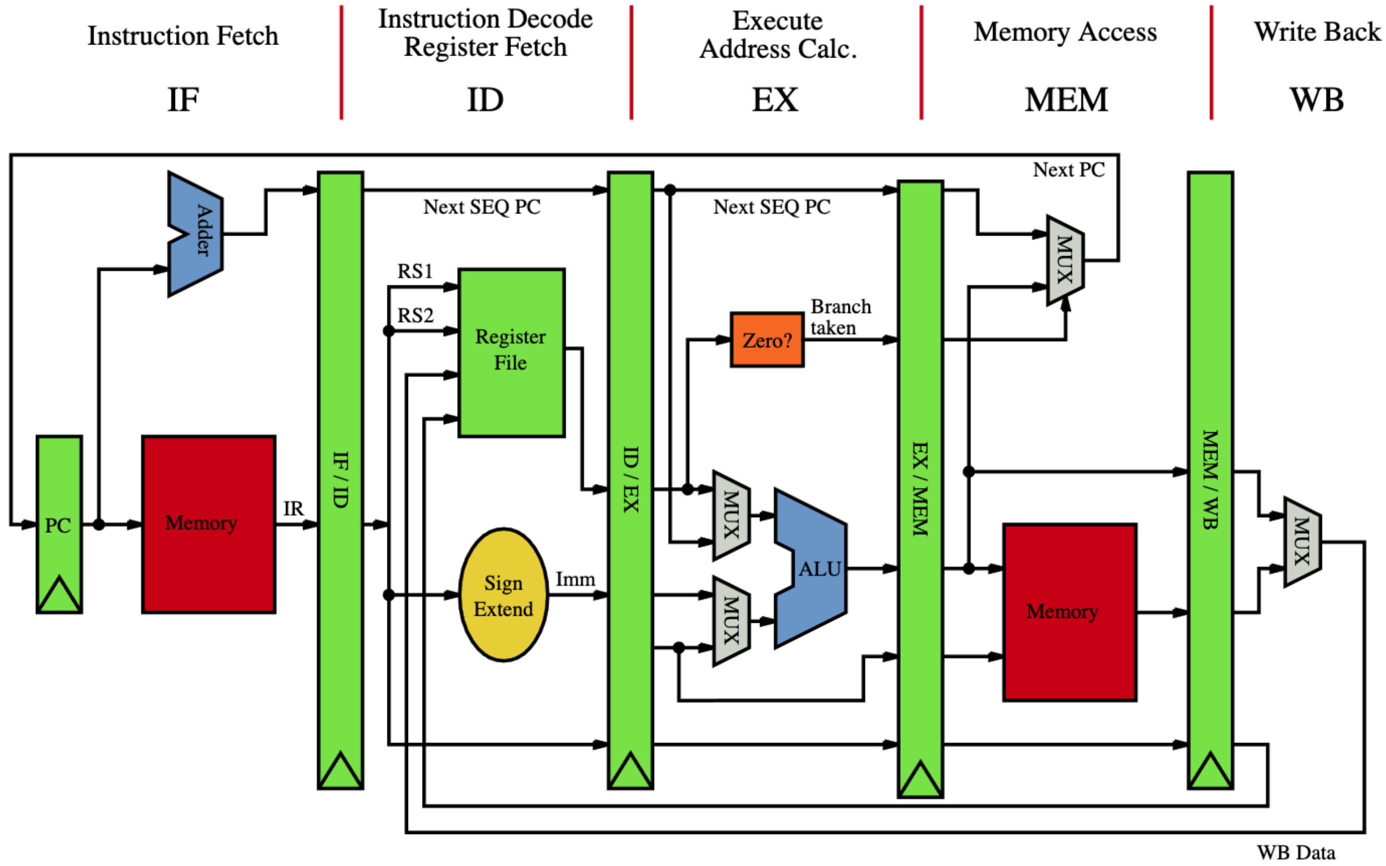


Figure: Stages of compilation. Image credit Wikimedia

Table of contents

Why are instruction set architectures important

Interface between computer science and electrical and computer engineering

- Software is varied, changes
- Hardware is standardized, static

Computer architect Fred Brooks and the IBM 360

- IBM was selling computers with different capacities,
- Compile once, and can run software on all IBM machines.
- Backward compatibility.
- An influential idea.

CISC vs. RISC

Complex instruction set computer

- Intel and AMD
- Have an extensive and complex set of instructions
- For example: x86's extensions: x87, IA-32, x86-64, MMX, 3DNow!, SSE, SSE2, SSE3, SSSE3, SSE4, SSE4.2, SSE5, AES-NI, CLMUL, RDRAND, SHA, MPX, SGX, XOP, F16C, ADX, BMI, FMA, AVX, AVX2, AVX512, VT-x, VT-d, AMD-V, AMD-Vi, TSX, ASF
- Can license Intel's compilers to extract performance
- Secret: inside the processor, they break it down to more elementary instructions

CISC vs. RISC

Reduced instruction set computer

- MIPS, ARM, RISC-V (can find Patterson and Hennessy Computer Organization and Design textbook in each of these versions), and PowerPC
- Have a relatively simple set of instructions
- For example: ARM's extensions: SVE;SVE2;TME; All mandatory: Thumb-2, Neon, VFPv4-D16, VFPv4 Obsolete: Jazelle
- ARM: smartphones, Apple ARM M1 Mac

Into the future: Post-ISA world

Post-ISA world

- Increasingly, the CPU is not the only character
- It orchestrates among many pieces of hardware
- Smartphone die shot
- GPU, TPU, FPGA, ASIC

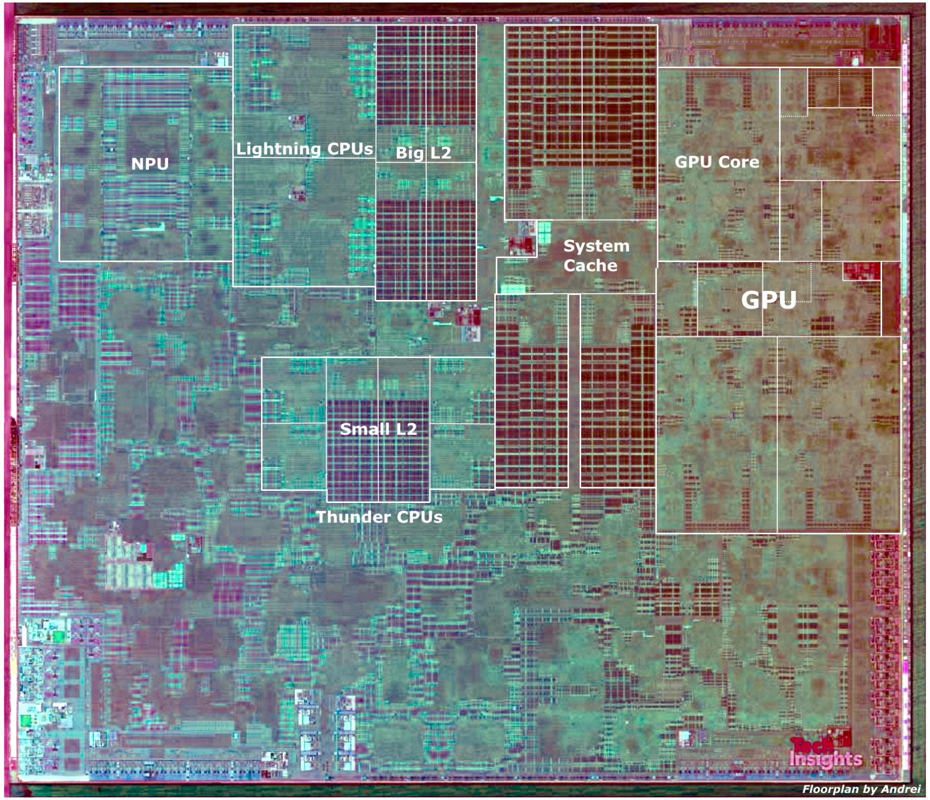
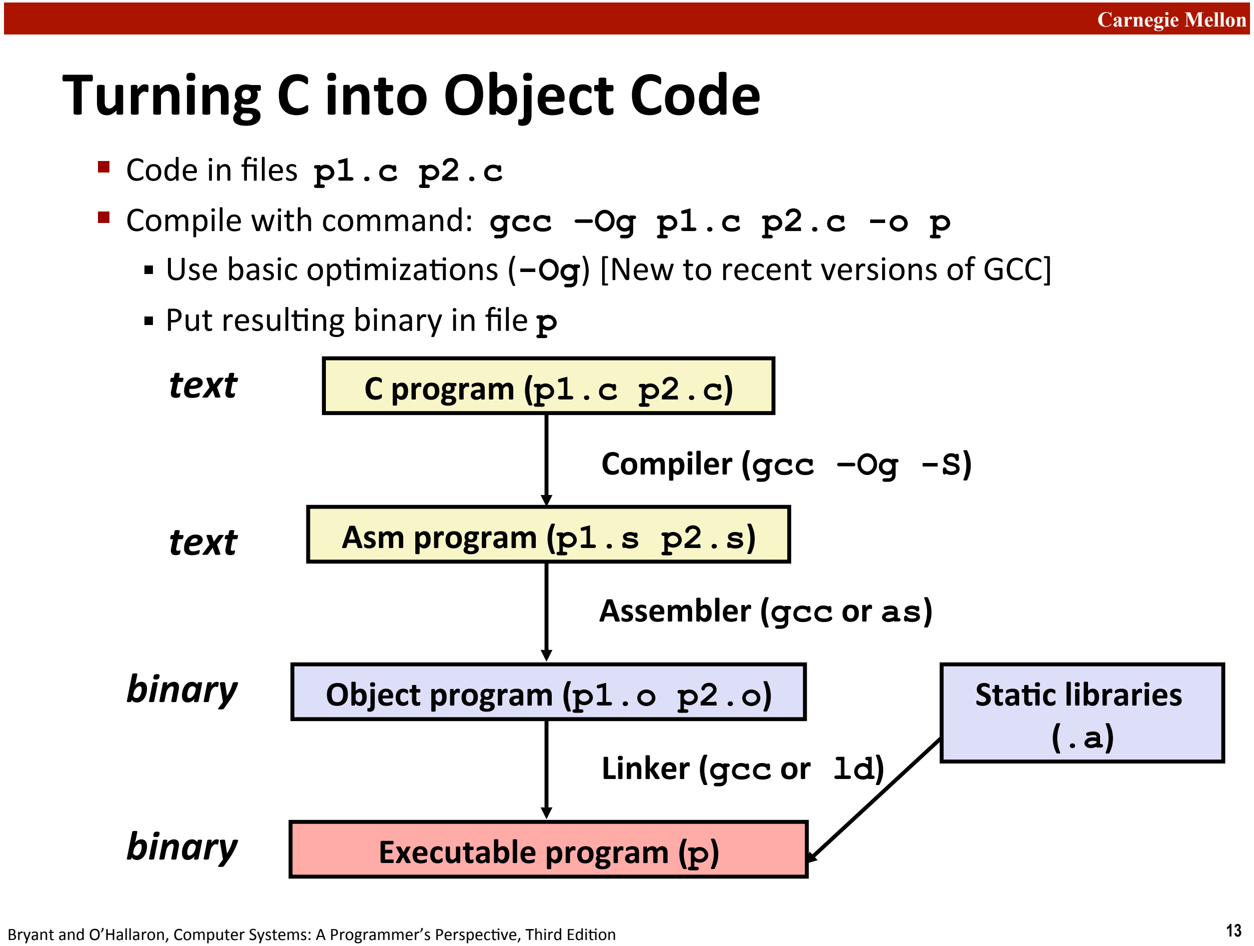


Figure: Apple A13 (2019 Apple iPhone 11 CPU). Image credit AnandTech

Table of contents

Unraveling the compilation chain



- gcc -0g -S swap.c
- objdump -d swap

Let's go to CS:APP textbook lecture slides (05-machine-basics.pdf) slide 28

Data movement instructions

Does unsigned / signed matter?

1. `void swap_uc ( unsigned char*a, unsigned char*b );`
2. `void swap_sc ( signed char*a, signed char*b );`

Swapping different data sizes

1. `void swap_c ( char*a, char*b );`
2. `void swap_s ( short*a, short*b );`
3. `void swap_i ( int*a, int*b );`
4. `void swap_l ( long*a, long*b );`

Table of contents

Data size and x86 / x86-64 registers

Assembly syntax

Instruction Source, Dest

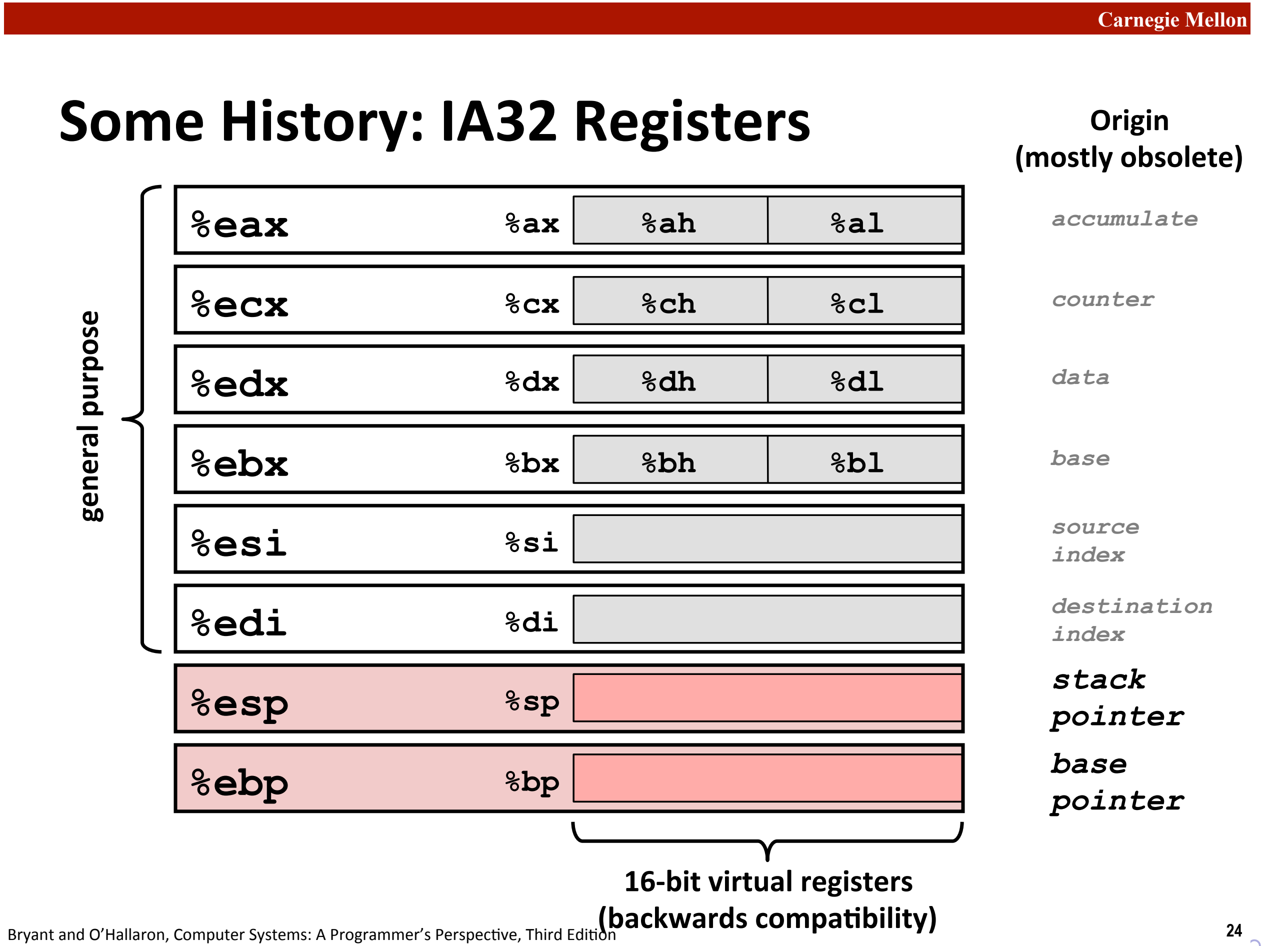
```
swap_l:
 movq (%rsi), %rax
 movq (%rdi), %rdx
 movq %rdx, (%rsi)
 movq %rax, (%rdi)
 ret
```

swap	data type	mov operation	registers
swap_uc	unsigned char	movb (move byte)	%al, %dl
swap_sc	signed char	movb (move byte)	%al, %dl
swap_c	char	movb (move byte)	%al, %dl
swap_s	short	movw (move word)	%ax, %dx
swap_i	int	movl	%eax, %edx
swap_l	long	movq	%rax, %rdx

Data size and IA32, x86, and x86-64 registers

data type	registers
char	%al, %dl
short	%ax, %dx
int	%eax, %edx
long	%rax, %rdx

Note the backward compatibility.



Data size and IA32, x86, and x86-64 registers

data type	registers
char	%al, %dl
short	%ax, %dx
int	%eax, %edx
long	%rax, %rdx

Note the backward compatibility.

x86-64 Integer Registers

%rax	%eax	%r8	%r8d
%rbx	%ebx	%r9	%r9d
%rcx	%ecx	%r10	%r10d
%rdx	%edx	%r11	%r11d
%rsi	%esi	%r12	%r12d
%rdi	%edi	%r13	%r13d
%rsp	%esp	%r14	%r14d
%rbp	%ebp	%r15	%r15d

- Can reference low-order 4 bytes (also low-order 1 & 2 bytes)

Data size and IA32, x86, and x86-64 registers

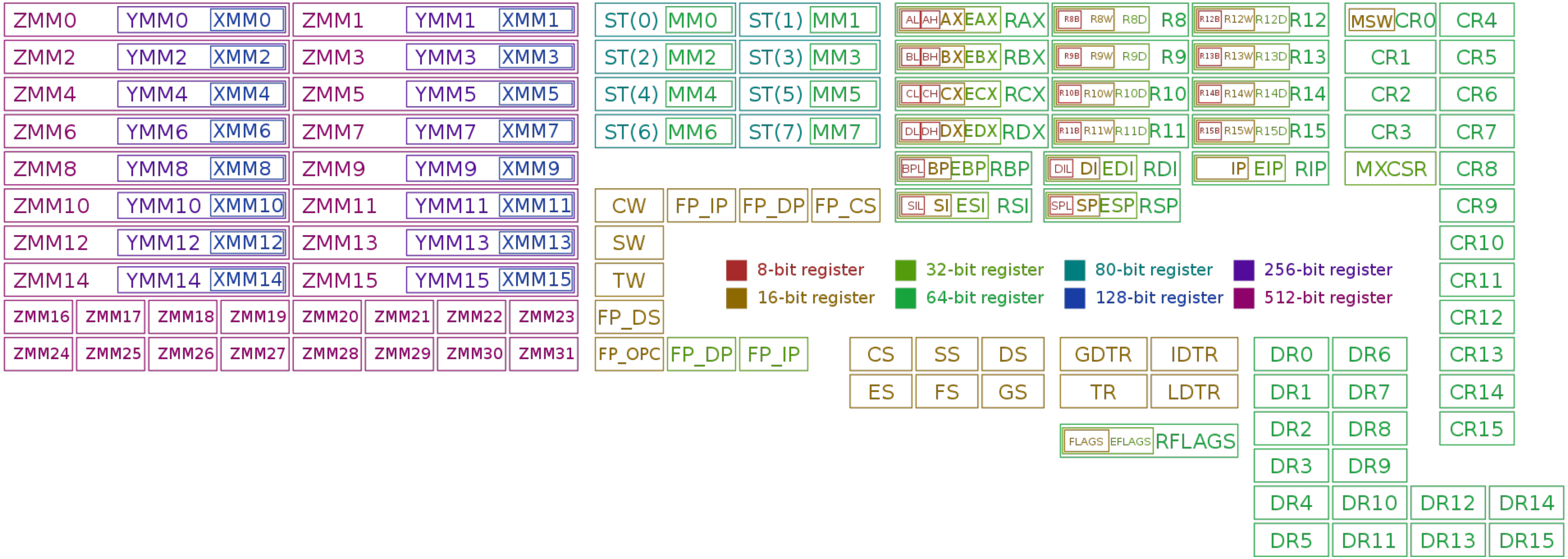


Figure: x86-64 with SIMD extensions registers. Image credit: `https://commons.wikimedia.org/wiki/File:Table_of_x86_Registers_svg.svg`