CISC vs. RISC

Jin-Soo Kim
(jinsoo.kim@snu.ac.kr)

Systems Software &
Architecture Lab.

Seoul National University

Fall 2019

MIPS

Chap. 2.16

MIPS ISA

- Stanford MIPS commercialized by MIPS Technologies (www.mips.com)
- Similar basic set of instructions
  - 32-bit instructions
  - 32 general purpose registers, register 0 is hardwired to 0
  - 32 floating-point registers
  - Memory accessed only by load/store instructions
  - Consistent use of addressing modes for all data sizes
- Different conditional branches (for <, <=, >, >=)
  - RISC-V: `blt`, `bge`, `bltu`, `bgeu`
  - MIPS: `slt`, `sltu` first, then use `beq` or `bne`
- MIPS is big-endian

```
slt  $t0, $s1, $s2
bne  $t0, $zero, L1
```

MIPS Registers

#	Name	Usage
0	zero	The constant value 0
1	at	Assembler temporary
2	v0	Values for results and expression evaluation
3	v1	
4	a0	Arguments
5	a1	
6	a2	
7	a3	
8	t0	Temporaries (Caller-save registers)
9	t1	
10	t2	
11	t3	
12	t4	
13	t5	
14	t6	
15	t7	

#	Name	Usage
16	s0	Saved temporaries (Callee-save registers)
17	s1	
18	s2	
19	s3	
20	s4	
21	s5	
22	s6	
23	s7	
24	t8	More temporaries (Caller-save registers)
25	t9	
26	k0	Reserved for OS kernel
27	k1	
28	gp	Global pointer
29	sp	Stack pointer
30	fp	Frame pointer
31	ra	Return address

MIPS Instruction Encoding

Register-register

	31 25	24 20	19 15	14 12	11 7	6 0
RISC-V	funct7(7)	rs2(5)	rs1(5)	funct3(3)	rd(5)	opcode(7)

	31 26	25 21	20 16	15 11	10 6	5 0
MIPS	Op(6)	Rs1(5)	Rs2(5)	Rd(5)	Const(5)	Opx(6)

Load

	31 20	19 15	14 12	11 7	6 0
RISC-V	immediate(12)	rs1(5)	funct3(3)	rd(5)	opcode(7)

	31 26	25 21	20 16	15 0
MIPS	Op(6)	Rs1(5)	Rs2(5)	Const(16)

Store

	31 25	24 20	19 15	14 12	11 7	6 0
RISC-V	immediate(7)	rs2(5)	rs1(5)	funct3(3)	immediate(5)	opcode(7)

	31 26	25 21	20 16	15 0
MIPS	Op(6)	Rs1(5)	Rs2(5)	Const(16)

Branch

	31 25	24 20	19 15	14 12	11 7	6 0
RISC-V	immediate(7)	rs2(5)	rs1(5)	funct3(3)	immediate(5)	opcode(7)

	31 26	25 21	20 16	15 0
MIPS	Op(6)	Rs1(5)	Opx/Rs2(5)	Const(16)

Intel x86

Chap. 2.17

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

The Intel x86 ISA

- Evolution with backward compatibility

Year	Name	TRs	Max Hz	Note
1974	8080	6K	2M	8-bit microprocessor (successor to 8008)
1978	8086	29K	8M	First 16-bit processor
1980	8087	45K	10M	Floating-point coprocessor (led to IEEE 754 standard)
1982	80286	134K	12.5M	24-bit addresses, MMU, segmented memory & protection
1985	80386	275K	20M	“IA-32”: first 32-bit processor, paging
1989	80486	1.2M	25M	Pipelined, on-chip caches and FPU
1993	Pentium	3.1M	60M	Superscalar, 64-bit data path, FDIV bug
1995	Pentium Pro	5.5M	200M	P6 microarchitecture, Out-of-order execution, PAE
1997	Pentium MMX	4.5M	200M	MMX
1999	Pentium III	8.2M	500M	SSE

The Intel x86 ISA (cont'd)

- Evolution with backward compatibility

Year	Name	TRs	Max Hz	Note
2000	Pentium 4	42M	1.5G	NetBurst microarchitecture, SSE2
2004	Pentium 4E	125M	2.8G	"Intel 64": first 64-bit processor, SSE3
2006	Core 2 Duo	291M	2.3G	Core microarchitecture, First multi-core processor, SSSE3
2008	Core 2 (Penryn)	820M	2.5G	SSE4.1
2008	Core i7	731M	2.9G	Nehalem microarchitecture, Quad-core, SSE4.2

- If Intel didn't extend with compatibility, its competitors would!
- Technical elegance ≠ market success

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)

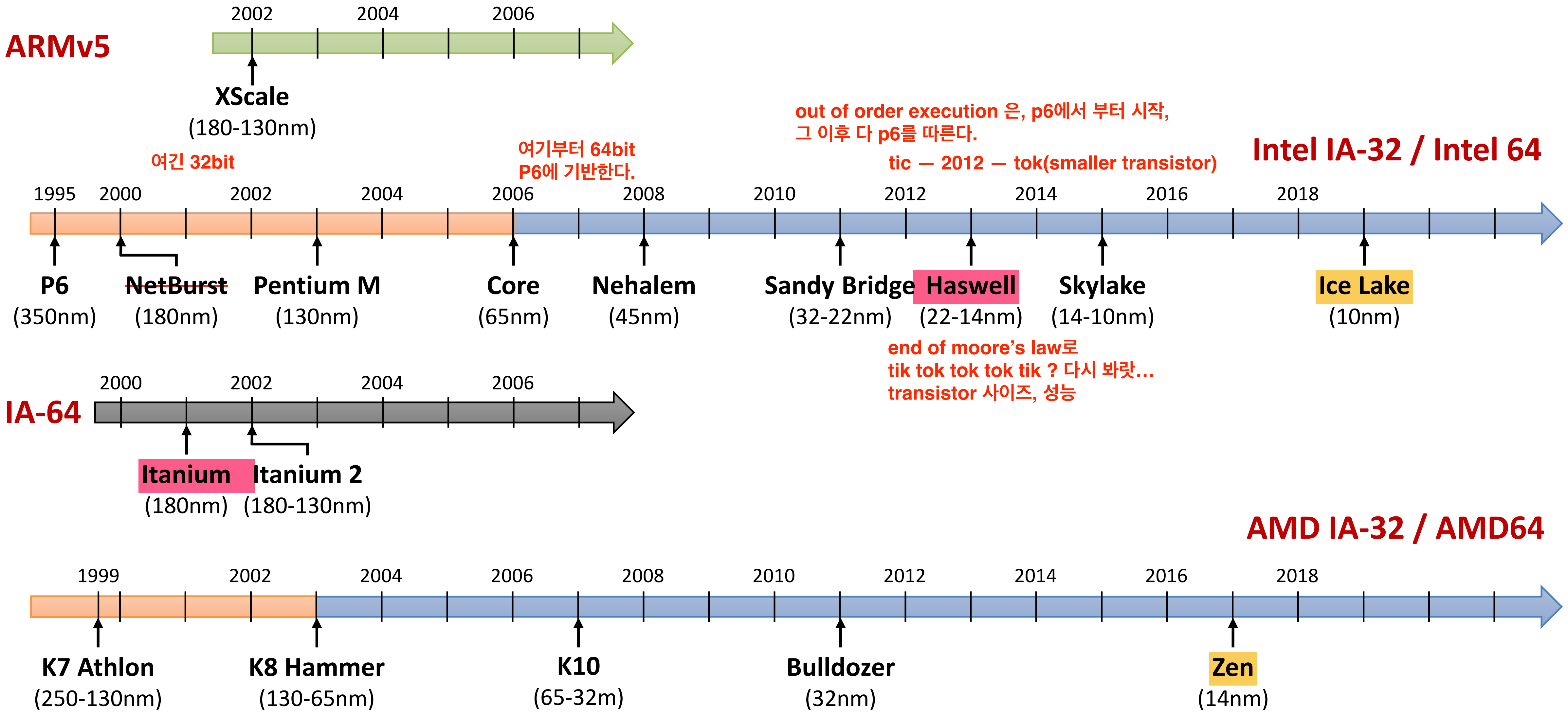
Intel's 64-bit History

- 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")

원래 AMD64에서 이름 변경?
여기서 inter 64된것, adopted

- 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

Intel / AMD Microarchitectures

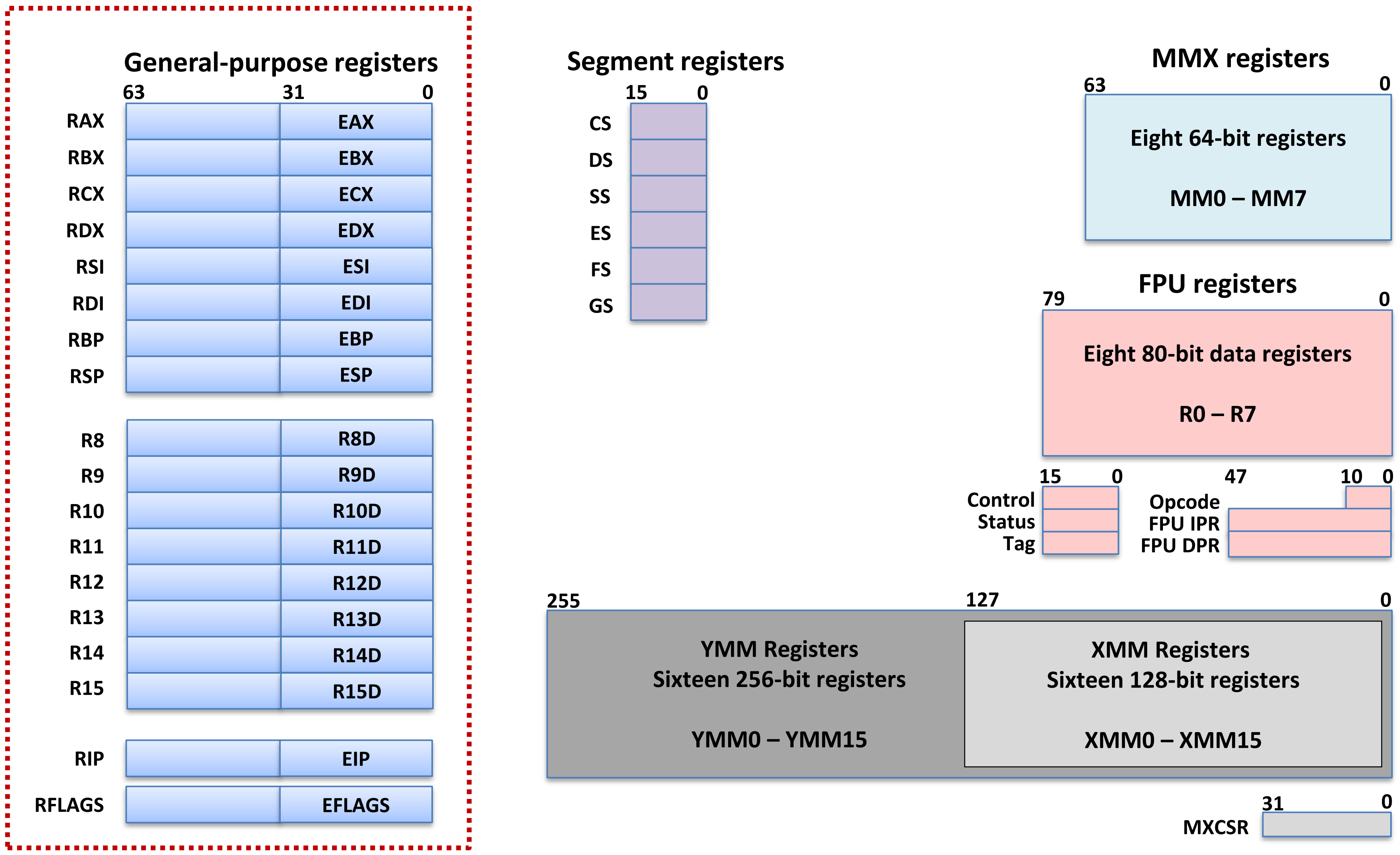


2019 State of the Art

- 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 9282 Processor (Cascade Lake)

  conservative -> but maximize performance(3 generation behind)
  - 6th generation Intel Xeon Scalable Processor with 14nm
  - 56 cores (112 threads) @ 2.6 – 3.8 GHz, 400W
  - Max 2 TB Memory (DDR4-2933), 77 MB L3 Cache
  - 40 PCIe 3.0 lanes

x86 Basic Execution Environment



x86 Operands

- Two operands per instruction
- The source can be an immediate, a register, or a memory location
- The destination can be a register or a memory location

Second source operand	Source/Dest operand	Example
Register	Register	addq %rcx, %rax
Immediate	Register	addq $4, %eax
Memory	Register	addq 0(%rcx), %rax
Register	Memory	addq %rax, 8(%rax)
Immediate	Memory	addq $4, 8(%rax)

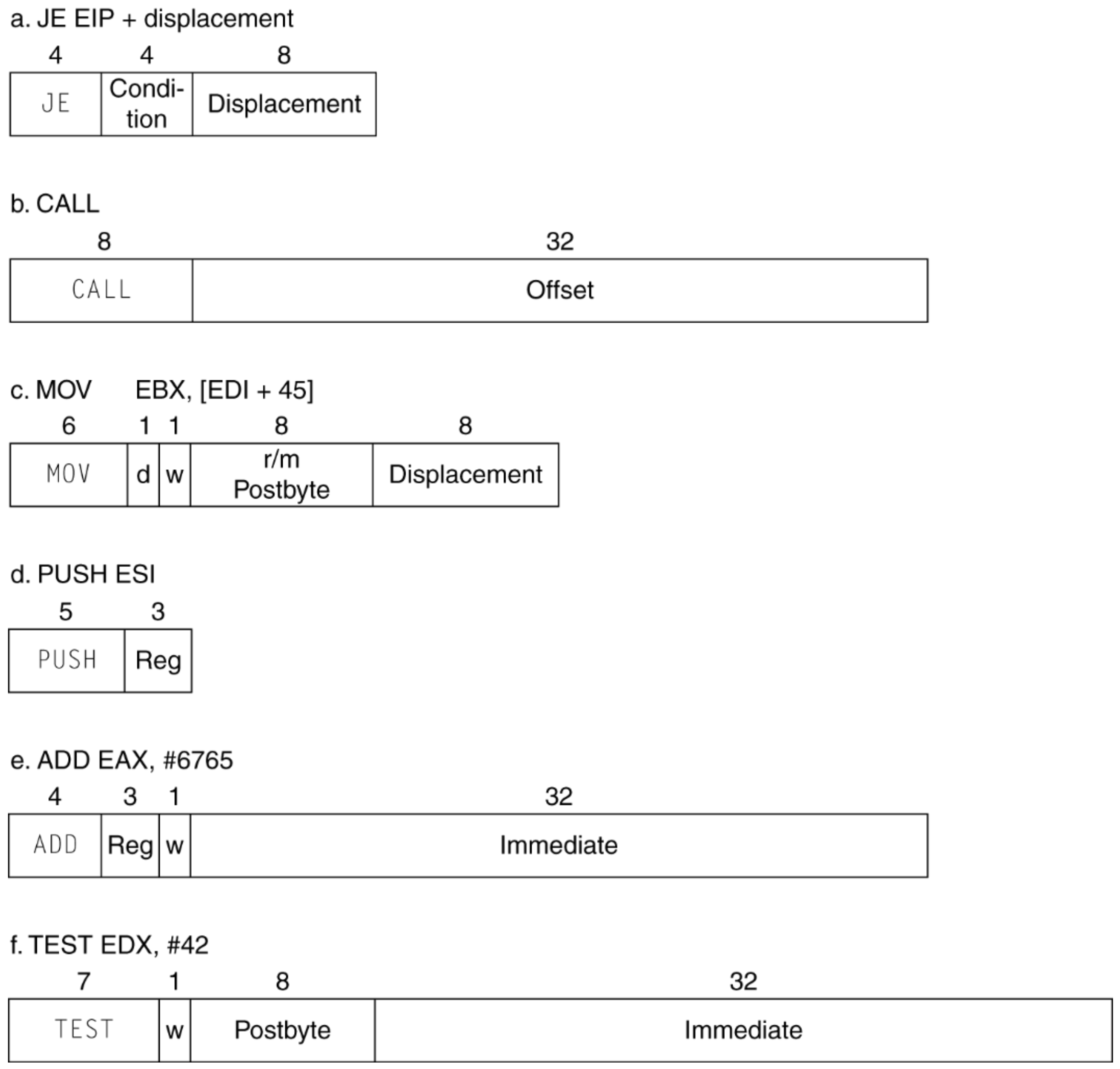
x86 Memory Addressing Modes

- `D(Rb,Ri,S)` Mem[Reg[Rb]+S*Reg[Ri]+D]
  - `D:` constant "displacement": 1, 2, or 4 bytes
  - `Rb`: Base register: any of 16 integer registers
  - `Ri`: Index register: any, except for `%rsp`
  - `S`: Scale: 1, 2, 4, or 8

Example	Effective address
`movq %rax, 8(%rdx)`	`%rdx + 8`
`movq %rax, 16(%rdx, %rcx)`	`%rdx + %rcx + 16`
`movq %rax, (,%rcx, 4)`	`%rcx * 4`
`movq %rax, 32(%rdx, %rcx, 8)`	`%rdx + %rcx * 8 + 32`

x86 Instruction Encoding

- Variable length encoding
  - Postfix bytes specify addressing mode
  - Prefix bytes modify operation
    - Operand length, repetition, locking, …
  - The length of an instruction can be up to 17 bytes
    - Up to 4 prefixes, each of which is 1 byte
    - 1 ~ 3 bytes for opcode
    - Up to 1 byte for simple addressing modes
    - Up to 1 byte for complex addressing modes
    - Up to 4 bytes for displacement
    - Up to 4 bytes for immediate data



CISC vs. RISC

Chap. 2.18 – 2.20

CISC (Complex Instruction Set Computer)

- Add instructions to perform "typical" programming tasks
  - DEC PDP-11 & VAX, IBM System/360, Motorola 68000, IA-32, Intel 64, …
- Stack-oriented instruction set
  - Use stack to pass arguments, save program counter, etc.
  - Explicit push and pop instructions
- Arithmetic instructions can access memory
  - Requires memory read and write during computation
  - Complex addressing modes
- Instructions have varying lengths
- Condition codes
  - Set as side effect of arithmetic and logical instructions

RISC (Reduced Instruction Set Computer)

- Philosophy: Fewer, simple instructions
  - Might take more to get given task done
  - Can execute them with small and fast hardware
  - Stanford MIPS, UCB RISC-V, Sun SPARC, IBM Power/PowerPC, ARM, SuprH, …

핸드폰, superH

- Register-oriented instruction set
  - Many more (typically 32+) registers
  - Use for arguments, return address, temporaries
- Only load and store instructions can access memory
- Each instruction has fixed size
- No condition codes

arm은 condition 있다. 따라 다른듯,

  - Test instructions return 0/1 in register

CISC vs. RISC

- Original debate
  - CISC proponents – easy for compiler, fewer code bytes
  - RISC proponents – better for optimizing compilers, can make run fast with simple chip design
- Current status
  - For desktop/server processors, choice of ISA not a technical issue
    - With enough hardware, can make anything run fast
    - Code compatibility more important
  - x86-64 adopted many RISC features
    - More registers, use them for argument passing
    - Hardware translates instructions to simpler μops
  - For embedded processors, RISC makes sense: smaller, cheaper, less power

문제, overhead 존재 바뀔 수없는 경우 존재한다.

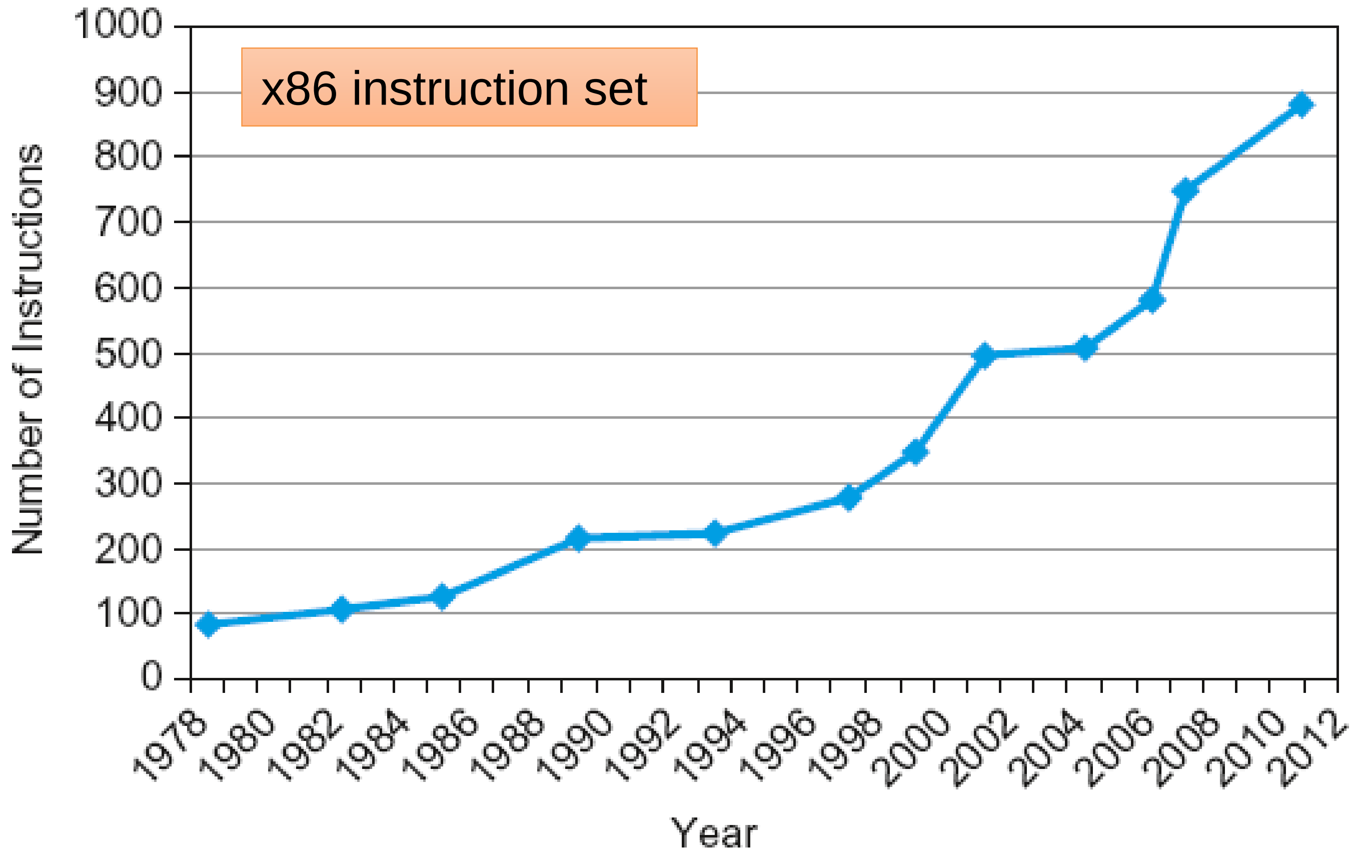
Fallacies

- Powerful instruction ⇒ higher performance
  - Fewer instructions required
  - But complex instructions are hard to implement
    - May slow down all instructions, including simple ones
  - Compilers are good at making fast code from simple instructions

- Use assembly code for high performance
  - But modern compilers are better at dealing with modern processors
  - More lines of code ⇒ more errors and less productivity

Fallacies (cont'd)

- Backward compatibility ⇒ instruction set doesn't change
  - But hey do accrete more instructions



Pitfalls

- Sequential words are not at sequential addresses
  - Increment by 4 (32-bit) or 8 (64-bit), not by 1!
- Keeping a pointer to an automatic variable after procedure returns
  - e.g., passing pointer back via an argument
  - Pointer becomes invalid when stack popped

Summary

- Design principles
  - Simplicity favors regularity
  - Smaller is faster
  - Good design demands good compromises
- Make the common case fast (and make the rare case correct)
- Layers of software/hardware
  - Compiler, assembler, linker, hardware
- RISC-V: typical of RISC ISAs