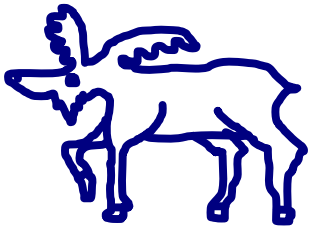
Lecture 5
History of ISA

Byung-gi Kim

School of Computer Science and Engineering

Soongsil University



2. Instruction: Language of the Computer (3rd edition)

2.1 Introduction

2.2 Operations of the Computer Hardware

2.3 Operands of the Computer Hardware

2.4 Representing Instructions in the Computer

2.5 Logical Operations

2.6 Instructions for Making Decisions

2.7 Supporting Procedures in Computer Hardware

2.8 Communicating with People

2.9 MIPS Addressing for 32-Bit Immediates and Addresses

2.16 Real Stuff: IA-32 Instructions

2.19 Historical Perspective and Further Reading

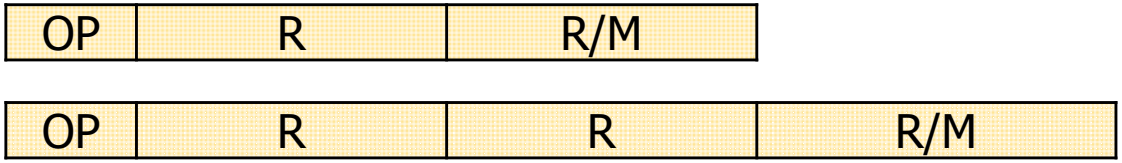
2.19 Historical Perspective and Further Reading

Accumulator Architectures

- **Single accumulator**
  - Registers expensive in early technologies (vacuum tubes)
  - Simple instruction decoding → Less logic → Faster cycle time
  - EDSAC, IBM 701, PDP-8
- **Extended accumulator**
  = dedicated register
  = special-purpose register
  - Intel 8086

General-Purpose Register Architectures

- **Register-memory architecture**
  - IBM 360, PDP-11, Motorola 68000



- **Load-store or register-register architecture**
  - CDC 6600, MIPS, SPARC

- **Memory-memory architecture**
  - VAX ?

Number of General-Purpose Registers

Machine	Number of general-purpose registers	Architectural style	Year
EDSAC	1	accumulator	1949
IBM 701	1	accumulator	1953
CDC 6600	8	load-store	1963
IBM 360	16	register-memory	1964
DEC PDP-8	1	accumulator	1965
DEC PDP-11	8	register-memory	1970
Intel 8008	1	accumulator	1972
Motorola 6800	2	accumulator	1974
DEC VAX	16	register-memory, memory-memory	1977
Intel 8086	1	extended accumulator	1978
Motorola 68000	16	register-memory	1980
Intel 80386	8	register-memory	1985
MIPS	32	load-store	1985
HP PA-RISC	32	load-store	1986
SPARC	32	load-store	1987
PowerPC	32	load-store	1992
DEC Alpha	32	load-store	1992
HP/Intel IA-64	128	load-store	2001
AMD64 (EMT64)	16	register-memory	2003

Figure 2.19.1

General-Purpose Register (GPR)

- **Any register can hold variable or pointer**
  - Simpler
  - More orthogonal
    - Opcode independent of register usage
  - More fast local storage
  - But, addresses and data must be same size
- **How many registers?**
  - More ⇒ fewer loads and stores
  - But, more instruction bits and longer cycle time

Compact Code and Stack Architectures

- **Instruction length**
  - MIPS R2000, SPARC are RISC architectures… 4 bytes
  - Intel IA-32 … 1 ~ 17 bytes
  - IBM S/360 … 2, 4, 6 bytes
  - VAX … 1 ~ 54 bytes
- **Stack computers**
  - Radical approach in the 1960s
    - In the belief that it was too hard for compilers to utilize registers effectively
    - Abandoned registers

      OP
  - HP3000, Burrough's B5000 , B5500, B6700, English Electric KDF9, Tandem Computers T/16, Inmos Transputers
  - Java bytecode
    - Compact instruction encoding is desirable.

x = b*c + d*e; y = b+c+d+e;

```
PUSH   b
PUSH   c
MUL
PUSH   d
PUSH   e
MUL
ADD
POP    x
PUSH   b
PUSH   c
ADD
PUSH   d
ADD
PUSH   e
ADD
POP    y
```

Stack machine
(0-address)

16 instructions
10 memory accesses

```
LOAD   R1, b
LOAD   R2, c
MUL    R1, R2
LOAD   R3, d
LOAD   R4, e
MUL    R3, R4
ADD    R1, R3
STORE  x, R1
LOAD   R1, b
ADD    R1, R2
LOAD   R3, d
ADD    R1, R3
ADD    R1, R4
STORE  y, R1
```

GPR machine
(2-address)

14 instructions
8 memory accesses
4 registers

```
LOAD   R1, b
MUL    R1, c
LOAD   R2, d
MUL    R2, e
ADD    R1, R2
STORE  x, R1
LOAD   R1, b
ADD    R1, c
ADD    R1, d
ADD    R1, e
STORE  y, R1
```

GPR machine
(2-address)

11 instructions
10 memory accesses
2 registers

```
LOAD   R1, b
LOAD   R2, c
MUL    R3, R1, R2
LOAD   R4, d
LOAD   R5, e
MUL    R6, R4, R5
ADD    R3, R3, R6
STORE  x, R3
ADD    R6, R1, R2
ADD    R6, R6, R4
ADD    R6, R6, R5
STORE  y, R6
```

RISC
(3-address)

12 instructions
6 memory accesses
6 registers

High-Level-Language Computer Architectures

- **Goal**
  - Making the hardware more like the programming language
  - HLL is executed by hardware without compilation/interpretation
- More efficient programming languages and compilers, plus expanding memory
  - ⇒ Commercially failed
- Burroughs B5000

RISC (Reduced Instruction Set Computer)

- Douglas W. Clark and Joel S. Emer at DEC, 1985
  - 20% of VAX instructions responsible for 60% of microcode
  - But only account for 0.2% of execution
- RISC defined by Colwell et. al. in "Computers, Complexity, and Controversy"
  1. Single-cycle operation
  2. Load/store design
  3. Hardwired control
  4. Few instructions/addressing modes
  5. Fixed instruction format
  6. More compile-time effort
- ARM, Hitachi SH, MIPS, IBM PowerPC, DEC Alpha, Sun Sparc, Hewlett Packard PA-RISC, Intel i860/i960, Motorola 88100/88110, AMD 29000 …

2.16 Real Stuff: IA-32 Instructions

- **Design alternative:**
  - provide more powerful operations
  - goal is to reduce number of instructions executed
  - danger is a slower cycle time and/or a higher CPI

> ***"The path toward operation complexity is thus fraught with peril. To avoid these problems, designers have moved toward simpler instructions"***

- **Let's look (briefly) at IA-32.**

Evolution with Backward Compatibility

- **8080 (1974): 8-bit microprocessor**
  - Accumulator, plus 3 index-register pairs
- **8086 (1978): 16-bit extension to 8080**
  - Complex instruction set (CISC)
- **8087 (1980): floating-point coprocessor**
  - Adds FP instructions and register stack
- **80286 (1982): 24-bit addresses, MMU**
  - Segmented memory mapping and protection
- **80386 (1985): 32-bit extension (now IA-32)**
  - Additional addressing modes and operations
  - Paged memory mapping as well as segments

Further Evolution…

- **i486 (1989): pipelined, on-chip caches and FPU**
  - Compatible competitors: AMD, Cyrix, …
- **Pentium (1993): superscalar, 64-bit datapath**
  - Later versions added MMX (Multi-Media eXtension) instructions
  - The infamous FDIV bug
- **Pentium Pro (1995), Pentium II (1997)**
  - New microarchitecture (see Colwell, *The Pentium Chronicles*)
- **Pentium III (1999)**
  - Added SSE (Streaming SIMD Extensions) and associated registers
- **Pentium 4 (2001)**
  - New microarchitecture
  - Added SSE2 instructions

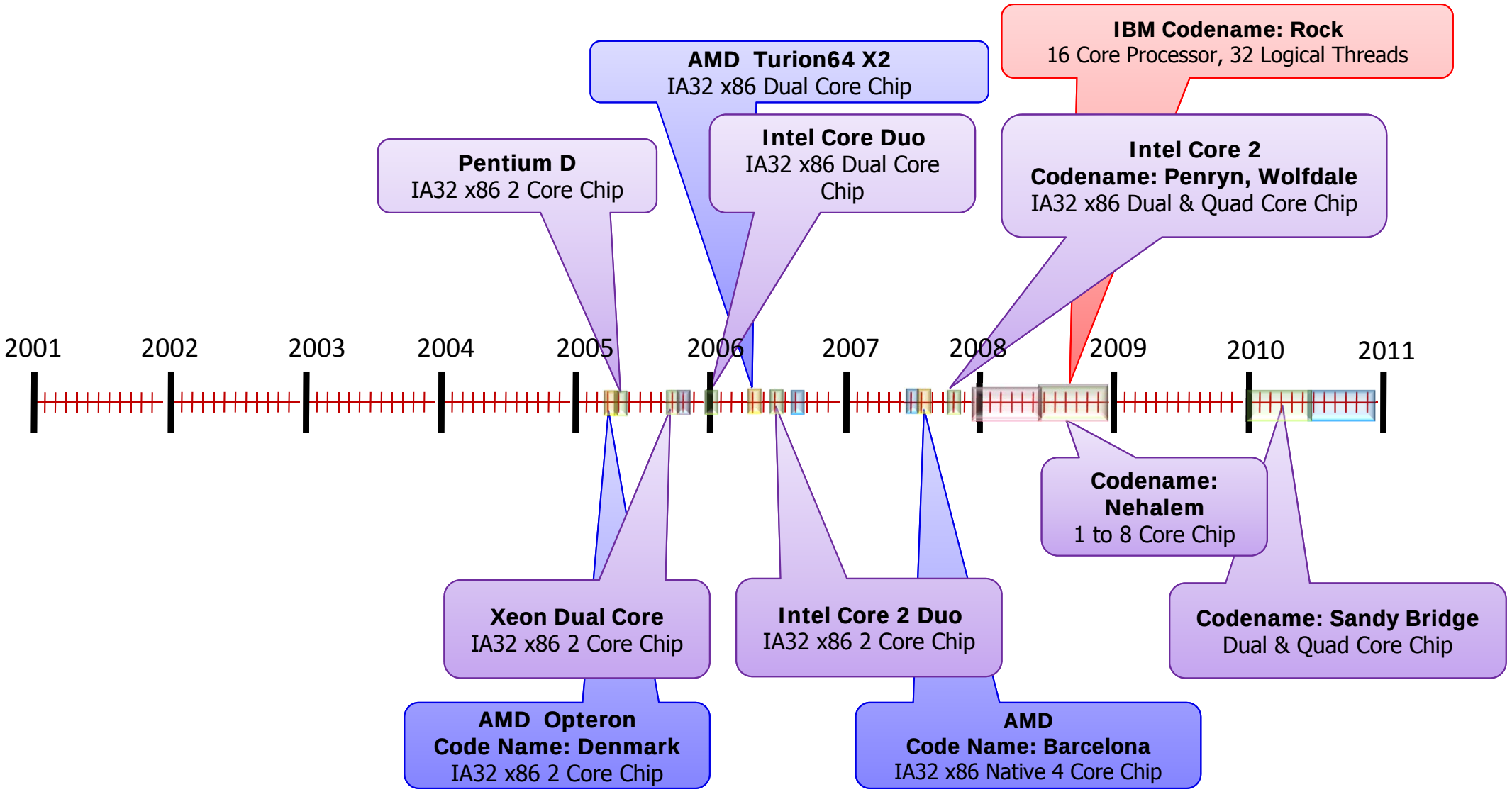
And Further…

- **AMD64 (2003): extended architecture to 64 bits**
- **EM64T – Extended Memory 64 Technology (2004)**
  - AMD64 adopted by Intel (with refinements)
  - Added SSE3 instructions
- **Intel Core (2006)**
  - Added SSE4 instructions, virtual machine support
- **AMD64 (announced 2007): SSE5 instructions**
  - Intel declined to follow, instead…
- **Advanced Vector Extension (announced 2008)**
  - Longer SSE registers, more instructions
- If Intel didn't extend with compatibility, its competitors would!
  - Technical elegance ≠ market success

Intel Multicores

- **Core Duo (Jan. 2006)**
- **Core 2 (Jul. 2006)**
  - Core 2 Duo
  - Core 2 Quad
  - Core 2 Extreme
- **Nehalem microarchitecture based (Jan. 2010)**
  - Core i3 … 2 cores
  - Core i5 … 2~4 cores
  - Core i7 … 2~6 cores
- **Sandy Bridge microarchitecture based (Jan. 2011)**
  - Core i3 … 2 cores
  - Core i5 … 2~4 cores
  - Core i7 … 2~4 cores

Multicore Trends



IA-32 Overview

- **Complexity:**
  - Instructions from 1 to 17 bytes long
  - one operand must act as both a source and destination
  - one operand can come from memory
  - complex addressing modes
    e.g., "base or scaled index with 8 or 32 bit displacement"
- **Saving grace:**
  - the most frequently used instructions are not too difficult to build
  - compilers avoid the portions of the architecture that are slow

"what the 80x86 lacks in style is made up in quantity, making it beautiful from the right perspective"

IA-32 Registers

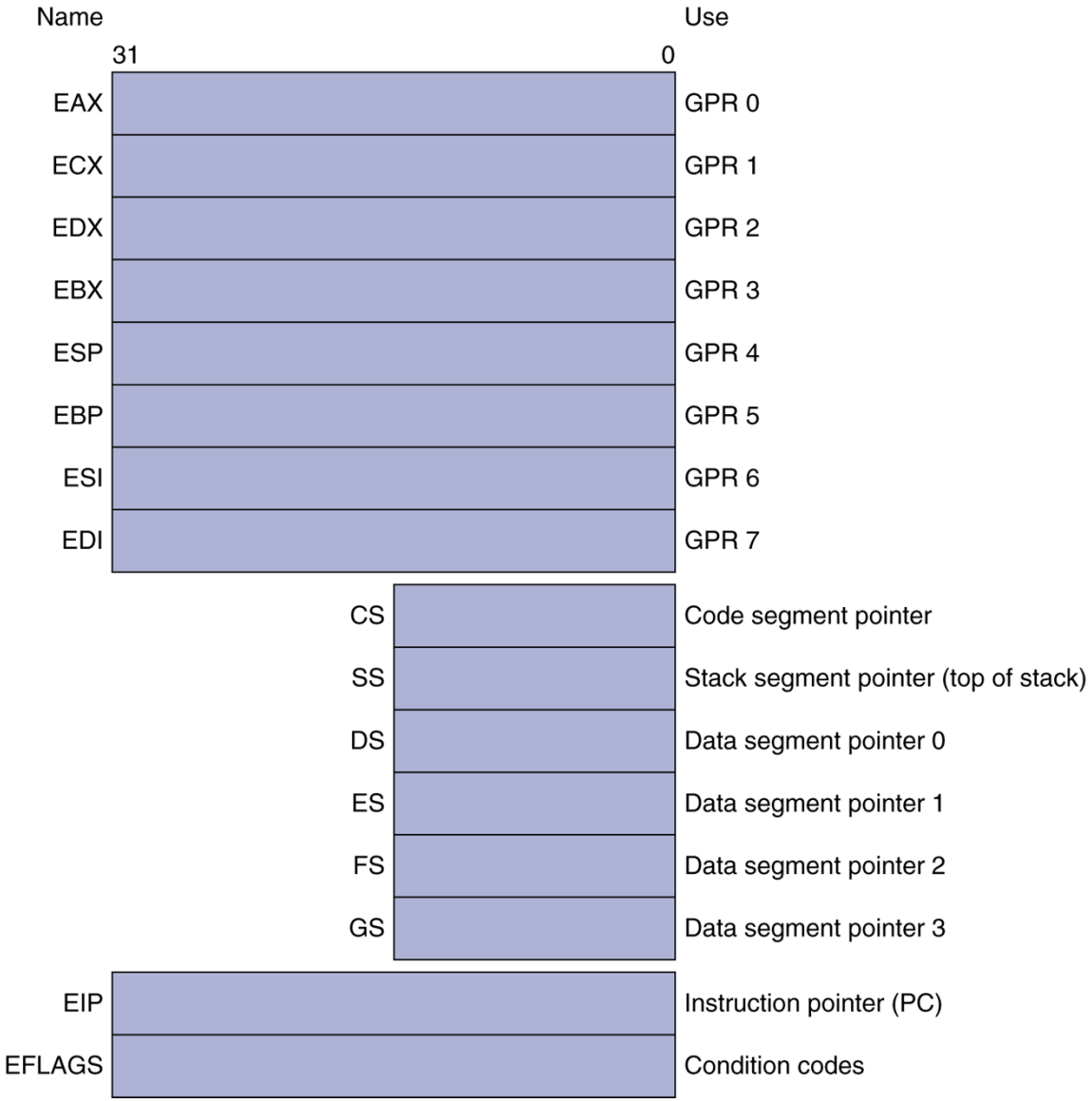


Figure 2.40

IA-32 Register Restrictions

- **Registers are not "general purpose"**
  - Note the restrictions below

Mode	Description	Register restrictions	MIPS equivalent
Register indirect	Address is in a register.	not ESP or EBP	`lw $s0,0($s1)`
Based mode with 8- or 32-bit displacement	Address is contents of base register plus displacement.	not ESP or EBP	`lw $s0,100($s1) # ≤16-bit # displacement`

Base plus scaled index	The address is Base + (2^{Scale} x Index) where Scale has the value 0, 1, 2, or 3.	Base: any GPR
		Index: not ESP
			`mul $t0,$s2,4`
			`add $t0,$t0,$s1`
			`lw $s0,0($t0)`

Base plus scaled index with 8- or 32-bit displacement	The address is Base + (2^{Scale} x Index) + displacement where Scale has the value 0, 1, 2, or 3.	Base: any GPR
		Index: not ESP
			`mul $t0,$s2,4`
			`add $t0,$t0,$s1`
			`lw $s0,100($t0) # ≤16-bit # displacement`

Figure 2.42

IA-32 instruction Formats

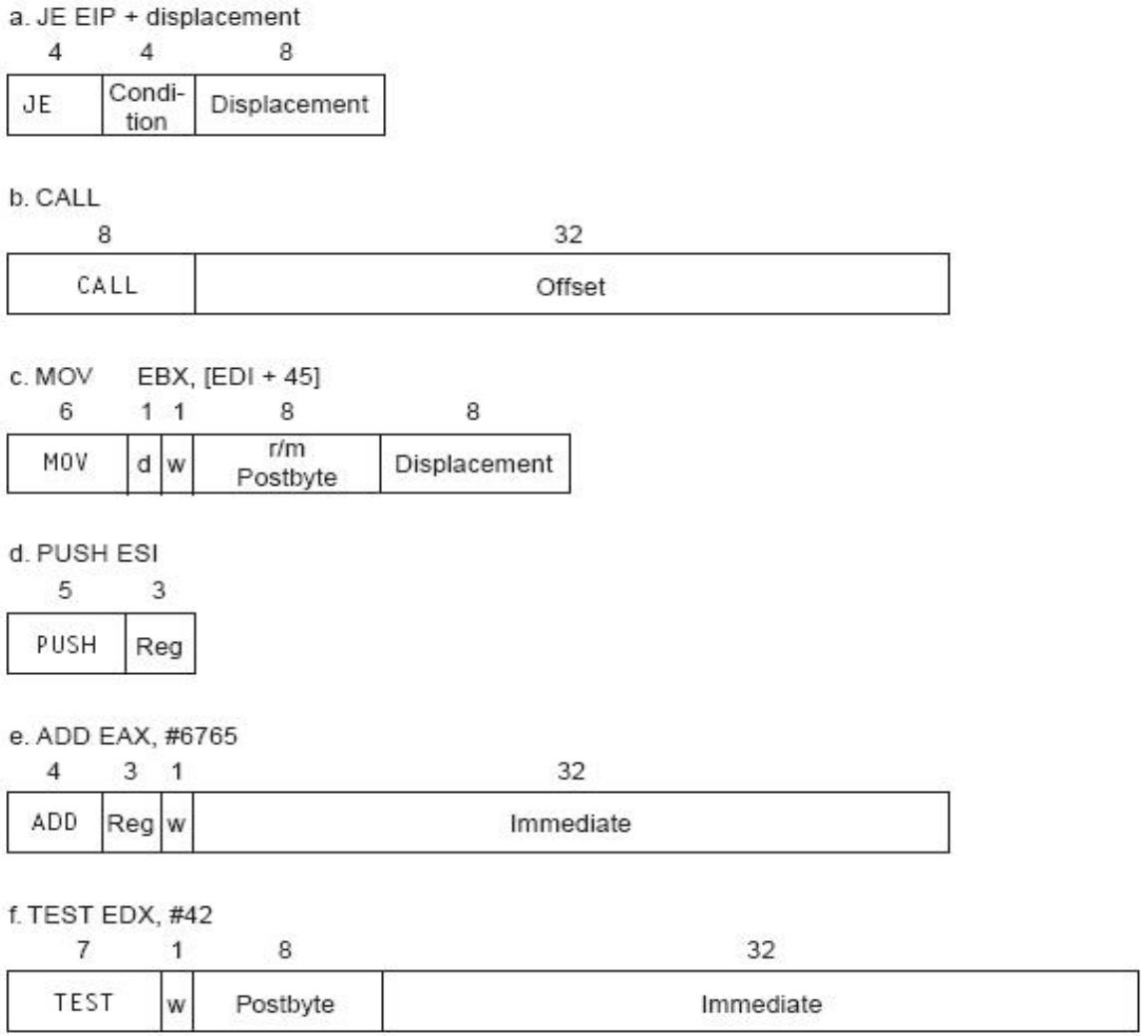


Figure 2.45

IA-32 Addressing Modes

- **Immediate**

```
MOV       EAX,10        ; EAX = 10
```

- **Direct**

```
MOV       EAX,I         ; EAX = Mem[&i]
I         DW    3
```

- **Register**

```
MOV       EAX,EBX       ; EAX = EBX
```

- **Register indirect**

```
MOV       EAX,[EBX]     ; EAX = Memory[EBX]
```

- **Based with 8- or 32-bit displacement**

```
MOV       EAX,[EBX+8]   ; EAX = Mem[EBX+8]
```

- **Based with scaled index (scale = 0 .. 3)**

```
MOV       EAX,ECX[EBX]  ; EAX = Mem[EBX + 2^scale * ECX]
```

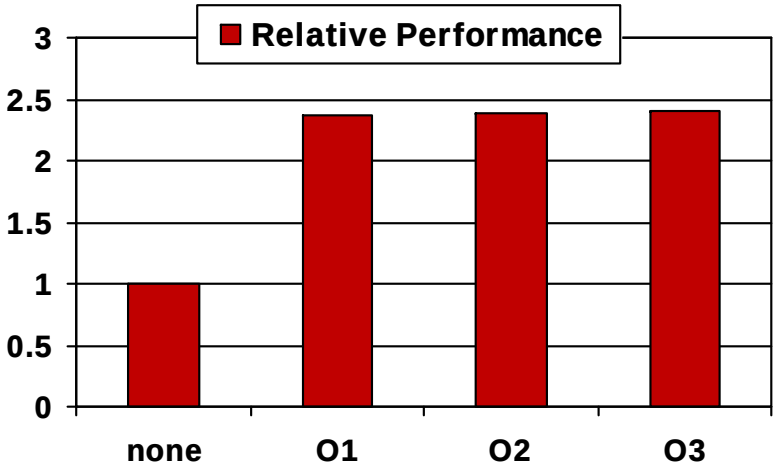
- **Based plus scaled index with 8- or 32-bit displacement**

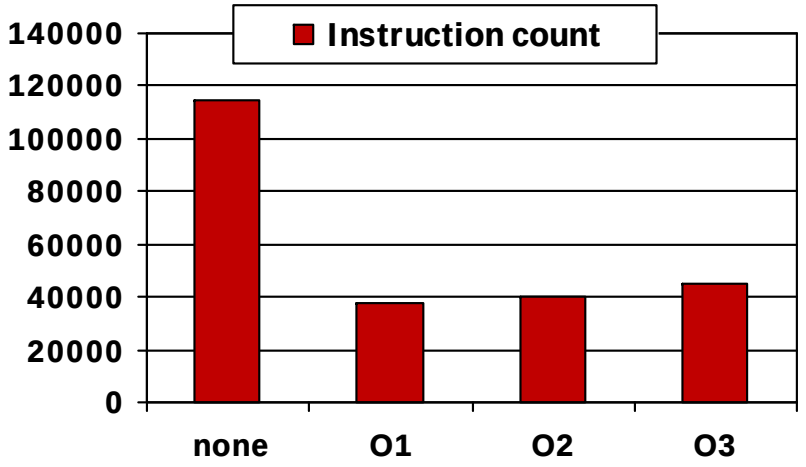
```
MOV       EAX,ECX[EBX+8]
```

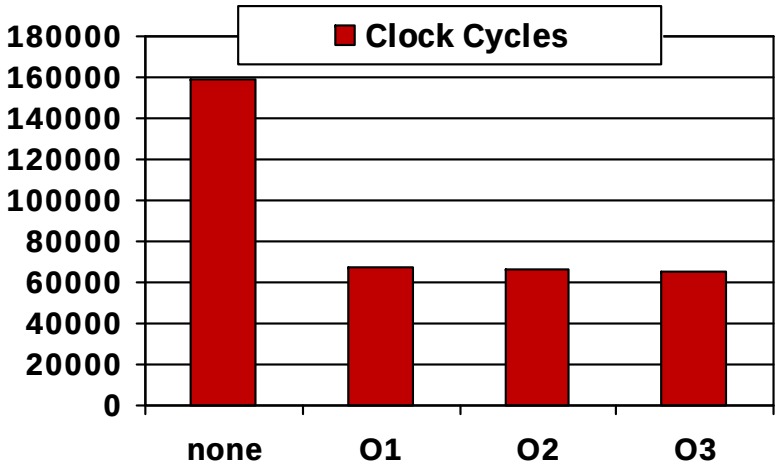
Supplement

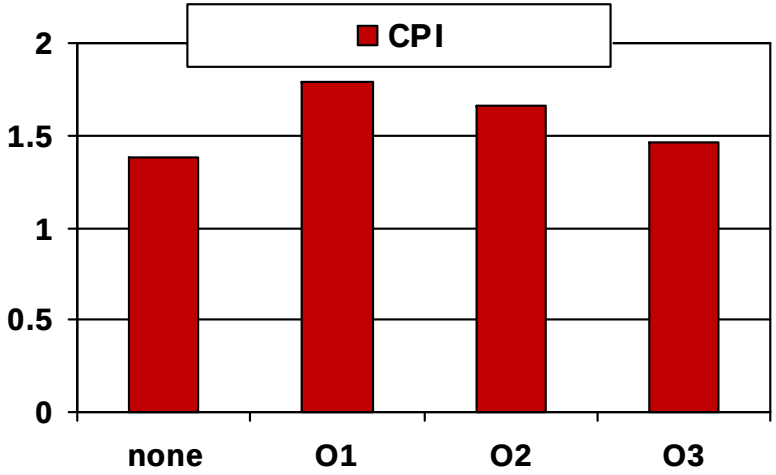
Effect of Compiler Optimization

Compiled with gcc for Pentium 4 under Linux

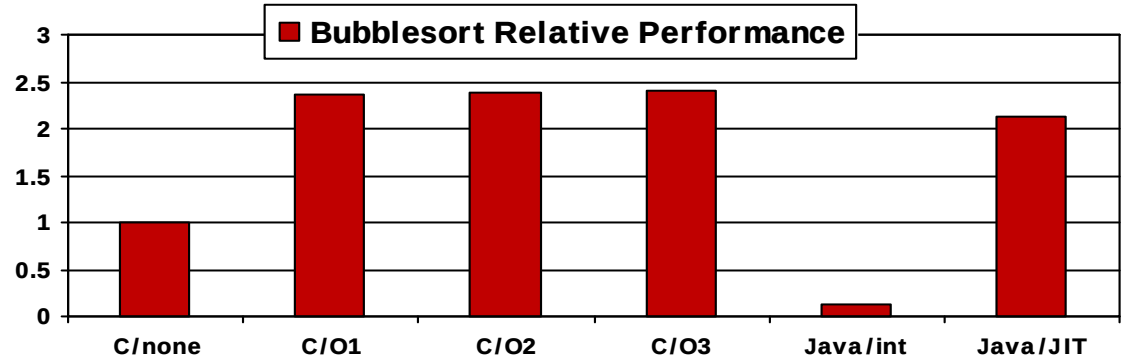


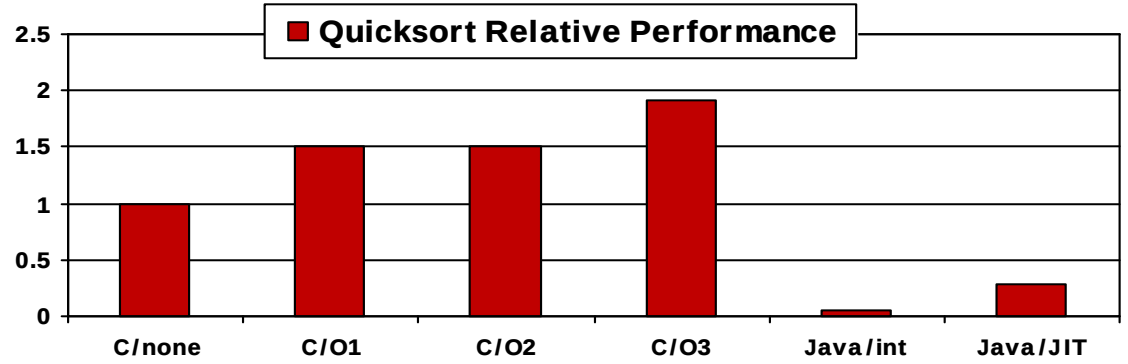


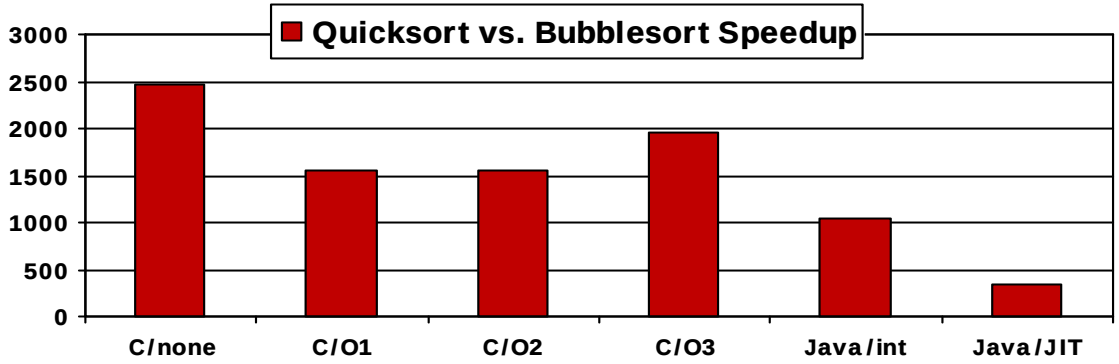




Effect of Language and Algorithm







Lessons Learnt

- Instruction count and CPI are not good performance indicators in isolation
- Compiler optimizations are sensitive to the algorithm
- Java/JIT compiled code is significantly faster than JVM interpreted
  - Comparable to optimized C in some cases
- Nothing can fix a dumb algorithm!