

Exam #1



- **March 31 (Thursday), 2011**
- **13:30 – 14:45**
- **#400102 (even student IDs)**
- **#400112 (odd student IDs)**

- **Scope**
 - Chap. 1 (except 1.4, 1.6 – 1.9)
 - Chap. 2 (except 2.11)
- **Closed-book exam**

ARM & IA-32

Jin-Soo Kim (jinsookim@skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
<http://csl.skku.edu>



ARM (1)

■ ARM & MIPS similarities

- ARM: the most popular embedded core
- Similar basic set of instructions to MIPS

	ARM	MIPS
Date announced	1985	1985
Instruction size	32 bits	32 bits
Address space	32-bit flat	32-bit flat
Data alignment	Aligned	Aligned
Data addressing modes	9	3
Registers	15 × 32-bit	31 × 32-bit
Input/output	Memory mapped	Memory mapped

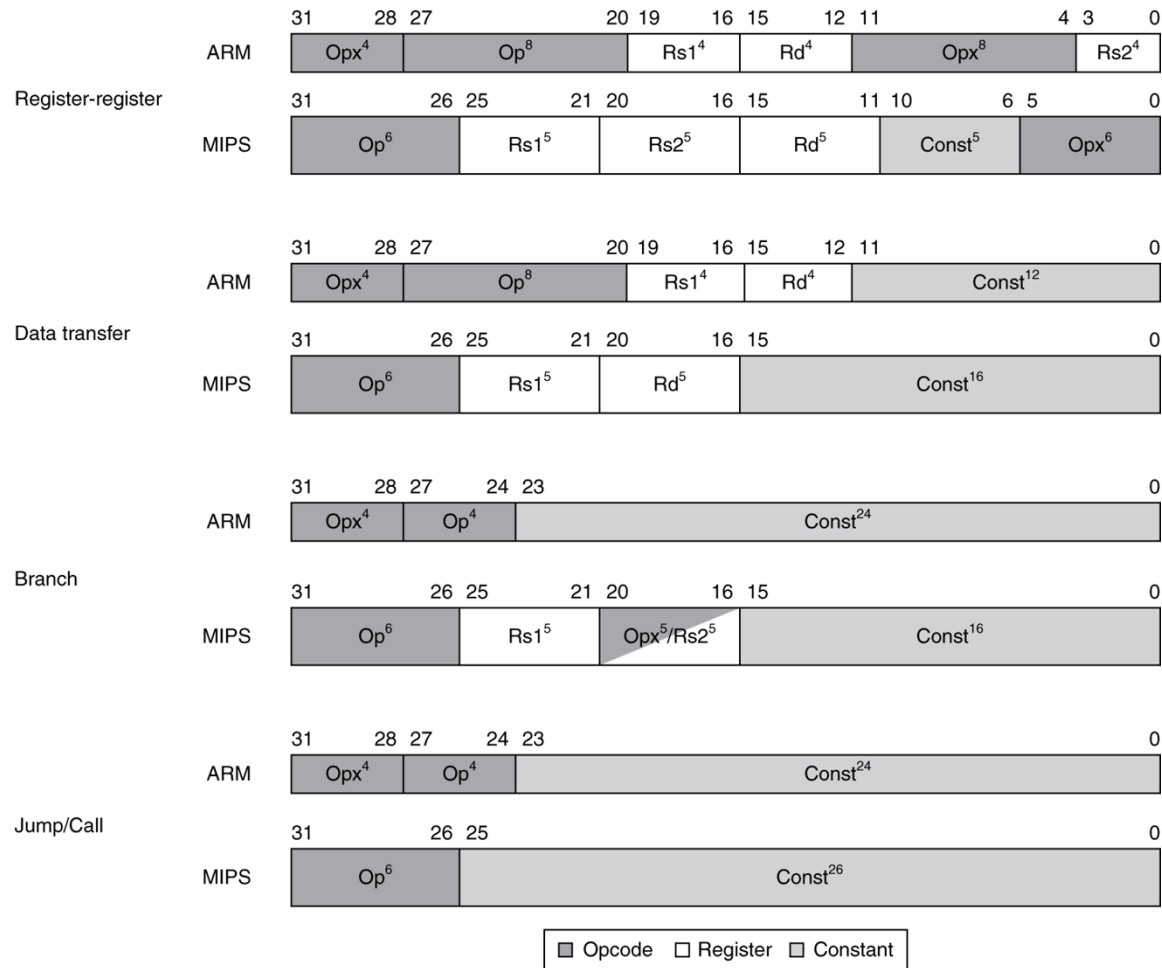
ARM (2)

▪ Compare and branch in ARM

- Uses condition codes for result of an arithmetic/logical instruction
 - Negative, zero, carry, overflow
 - Compare instructions to set condition codes without keeping the result
- Each instruction can be conditional
 - Top 4 bits of instruction word: condition value
 - Can avoid branches over a single instruction

ARM (3)

Instruction encoding



IA-32 (1)

▪ 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

IA-32 (2)

▪ Further evolution ...

- i486 (1989): pipelined, on-chip caches and FPU
 - Compatible competitors: AMD, Cyrix
- Pentium (1993): superscalar, 64-bit datapath
 - Later versions added MMX instructions
 - The infamous FDIV bug
- Pentium Pro (1995), Pentium II (1997)
 - New microarchitecture: P6
- Pentium III (1999)
 - Added SSE and associated registers
- Pentium 4 (2001)
 - New microarchitecture: NetBurst
 - Added SSE2 instructions

IA-32 (3)

■ 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

Technical elegance ≠ market success

IA-32 (4)

Basic x86 registers

Name	Use
EAX	GPR 0
ECX	GPR 1
EDX	GPR 2
EBX	GPR 3
ESP	GPR 4
EBP	GPR 5
ESI	GPR 6
EDI	GPR 7
CS	Code segment pointer
SS	Stack segment pointer (top of stack)
DS	Data segment pointer 0
ES	Data segment pointer 1
FS	Data segment pointer 2
GS	Data segment pointer 3
EIP	Instruction pointer (PC)
EFLAGS	Condition codes

IA-32 (5)

▪ Basic x86 addressing modes

- Two operands per instruction

Source/dest operand	Second source operand
Register	Register
Register	Immediate
Register	Memory
Memory	Register
Memory	Immediate

- Memory addressing modes

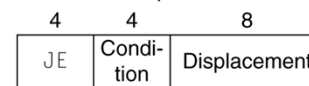
- Address in register
- Address = $R_{\text{base}} + \text{displacement}$
- Address = $R_{\text{base}} + 2^{\text{scale}} \times R_{\text{index}}$ (scale = 0, 1, 2, or 3)
- Address = $R_{\text{base}} + 2^{\text{scale}} \times R_{\text{index}} + \text{displacement}$

IA-32 (6)

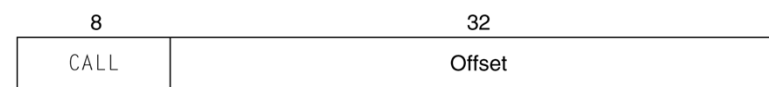
■ x86 instruction encoding

- Variable length encoding
- Postfix bytes specify addressing mode
- Prefix bytes modify operations
 - Operand length, repetition, locking, ...

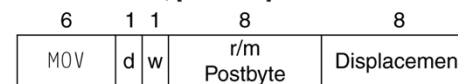
a. JE EIP + displacement



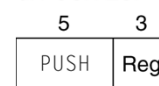
b. CALL



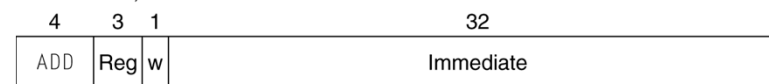
c. MOV EBX, [EDI + 45]



d. PUSH ESI



e. ADD EAX, #6765



f. TEST EDX, #42



IA-32 (7)

Instruction Prefixes	Opcode	ModR/M	SIB	Displacement	Immediate
----------------------	--------	--------	-----	--------------	-----------

Up to four prefixes of 1 byte each (optional)

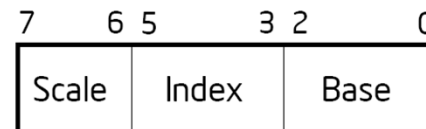
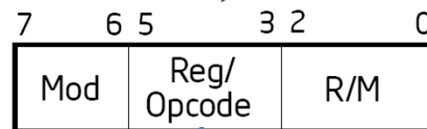
1-, 2-, or 3-byte opcode

1 byte (if required)

1 byte (if required)

Address displacement of 1, 2, or 4 bytes or none

Immediate data of 1, 2, or 4 bytes or none



Register or extended opcode

**Register or simple addressing modes:
[R], disp32, ([R]|X)+disp8/disp32, X**

**Lock and repeat
Segment override & Branch hints
Operand-size override
Address-size override**

**complex addressing modes:
 $X = [Rb + Ri * (1 | 2 | 4 | 8)]$**

IA-32 (8)

▪ Implementing IA-32

- Complex instruction set makes implementation difficult
 - Hardware translates instructions to simpler μ -operations
 - » Simple instructions: 1 – 1
 - » Complex instructions: 1 – many
 - Microengine similar to RISC
 - Market share makes this economically viable
- Comparable performance to RISC
 - Compilers avoid complex instructions

IA-32 (9)

▪ Peculiarities

- Segmented memory model
- 8 GPRs only
 - Can be partially accessed
 - Many memory accesses, short life time
 - Causes lots of anti- and output dependences
- One set of condition codes
 - Modified by most ALU operations
 - Various operations affect various flags
 - Short generate/use distance
- Explicit stack
- Little endian

IA-32 (10)

▪ CISC(Complex Instruction Set Computer)-style

- Huge number of assembly instructions
- Variable instruction lengths
 - Up to 17 bytes (~ 3 bytes on average)
- Instructions may reside in any byte address
- Arithmetic operations can read/write memory
- Multiple complex addressing modes
- Support for complex data types likes “strings”
 - The longer the string, the more cycles the instruction takes
- Registers associated with specific operations
- Implicit operands: `MUL EBX` → `EAX = EAX * EBX`

IA-32 (11)

▪ x86 overhead

- Major sources
 - Microcode ROMs: for decoding large, complex instructions
 - Prefetch logic: instructions are not a uniform size and hence can straddle cache lines
 - Segmented memory model: the decode logic has to check for and enforce code segment limits with its own dedicated address calculation hardware
- Transistor budget spent on x86 legacy support
 - ~ 30% for Pentium
 - ~ 40% for Pentium Pro
 - < 10% for Pentium 4
 - The percentage is even smaller for the very latest processors

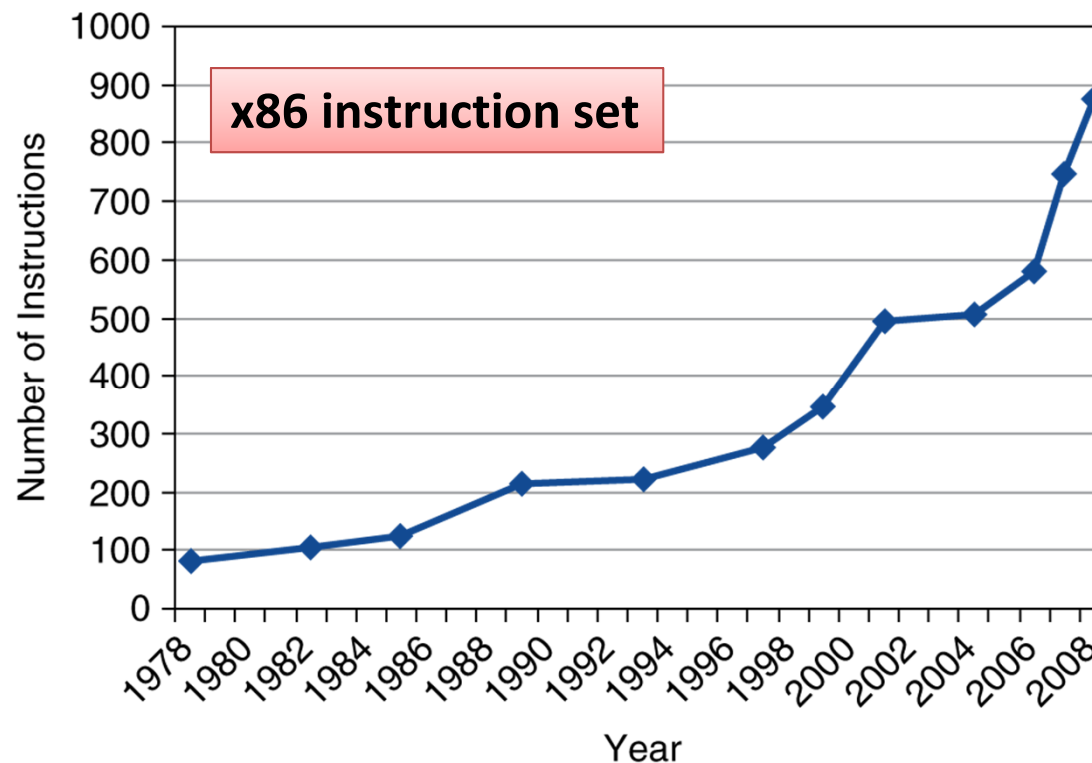
Fallacies (1)



- **Powerful instruction \Rightarrow 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 \Rightarrow more errors and less productivity

Fallacies (2)

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



Pitfalls



- **Sequential words are not at sequential addresses**
 - Increment by 4, 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

CISC Instruction Sets



- **Complex Instruction Set Computer**
 - Dominant style through mid-80's
 - Stack-oriented instruction set
 - Use stack to pass arguments, save program counter
 - Explicit push and pop instructions
 - Arithmetic instructions can access memory
 - Requires memory read and write
 - Complex address calculation
 - Condition codes
 - Set as side effect of arithmetic and logical instructions
 - Philosophy
 - Add instructions to perform “typical” programming tasks

RISC Instruction Sets

■ Reduced Instruction Set Computer

- Internal project at IBM, later popularized by Hennessy (Stanford) and Patterson (Berkeley)
- Fewer, simple instructions
 - Might take more instructions to get given task done
 - Can execute them with small and fast hardware
- Register-oriented instruction set
 - Many more (typically 32) registers
 - Use for arguments, return pointer, temporaries
- Only load and store instructions can access memory
- No condition codes
 - 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 processors, choice of ISA not a technical issue
 - With enough hardware, can make anything run fast
 - Code compatibility more important
- For embedded processors, RISC makes sense
 - Smaller, cheaper, less power

Concluding Remarks (1)



- **Design principles**
 - Simplicity favors regularity
 - Smaller is faster
 - Make the common case faster
 - Good design demands good compromises
- **Layers of software/hardware**
 - Compiler, assembler, hardware
- **MIPS: typical of RISC ISAs**
 - cf. Intel x86

Concluding Remarks (2)

- **Measure MIPS instruction executions in benchmark programs**
 - Consider making the common case fast
 - Consider compromises

Instruction class	MIPS examples	SPEC2006 Int	SPEC2006 FP
Arithmetic	add, sub, addi	16%	48%
Data transfer	lw, sw, lb, lbu, lh, lhu, sb, lui	35%	36%
Logical	and, or, nor, andi, ori, sll, srl	12%	4%
Cond. Branch	beq, bne, slt, slti, sltiu	34%	8%
Jump	j, jr, jal	2%	0%