CS152: Computer Systems Architecture
Storytime – x86 And Surrounding History

Sang-Woo Jun

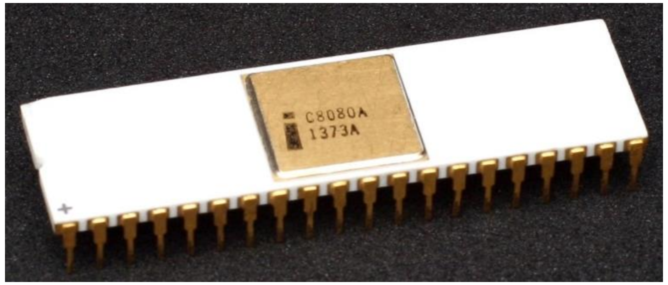
Winter 2021

UCI

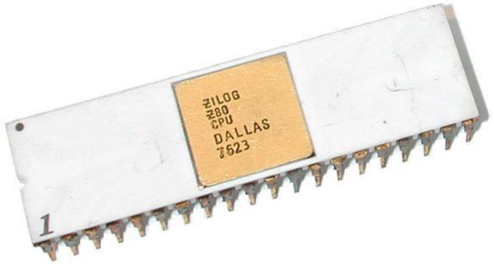
Large amount of material adapted from MIT 6.004, "Computation Structures",
Morgan Kaufmann "Computer Organization and Design: The Hardware/Software Interface: RISC-V Edition",
and CS 152 Slides by Isaac Scherson

x86: Evolution with backward compatibility

- 8080 (1974): 8-bit microprocessor
    - Accumulator, plus 3 index-register pairs
    - Widely successful, spawned many clones
    - Zilog Z80 still manufactured today!
- Intel iAPX 432 (1975): First 32-bit architecture
    - New ISA, not backwards compatible
    - High-level language features built in
        - Memory access control, garbage collection, etc in hardware
    - No explicit registers, stack-based
    - Bit-aligned variable-length ISA
    - Circuit too large! Spread across two chips
    - …Slow…



Intel 8080 (Photo Konstantin Lanzet)

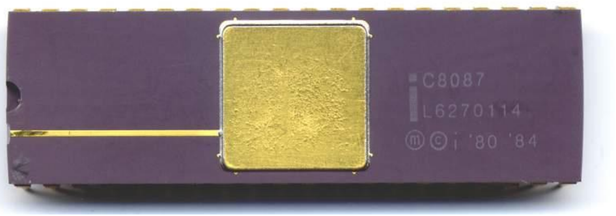


Zilog Z80 (Photo Gennadiy Shvets)

Toshiba Z84C00 (Photo Dhrm77, Wikipedia)

x86: Evolution with backward compatibility

- 8086 (1978): 16-bit extension to 8080
  - Intended temporary substitute until the delayed iAPX 432 became available
  - Backwards compatible with 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
  - Each segment was a 16-bit address space
    - Compatibility with legacy programs (CP/M, etc)

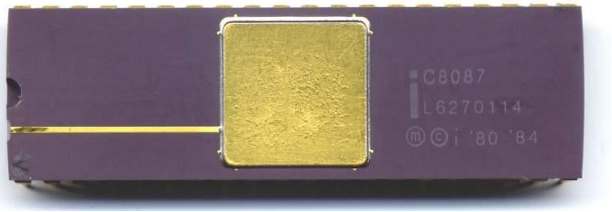


Intel 8087 (Photo Dirk Oppelt)

Intel 80286 (Photo Peter Binter)

x86: Evolution with backward compatibility

- 80386 (1985): 32-bit extension (now IA-32)
  - Additional addressing modes and operations
    - Paged memory mapping as well as segments
  - “Virtual 8086 mode”
    - Special operation mode for a task/process
    - Hardware virtualization support for legacy software (e.g., MS-DOS)
    - Multiple instances of DOS programs could run in parallel
  - OS can finally move beyond MS-DOS!
    - Previously stuck because DOS compatibility could not be ignored
    - DOS software expected exclusive hardware control…
    - Windows 3.1 built on this



Intel 8087 (Photo Dirk Oppelt)

Intel 80286 (Photo Peter Binter)

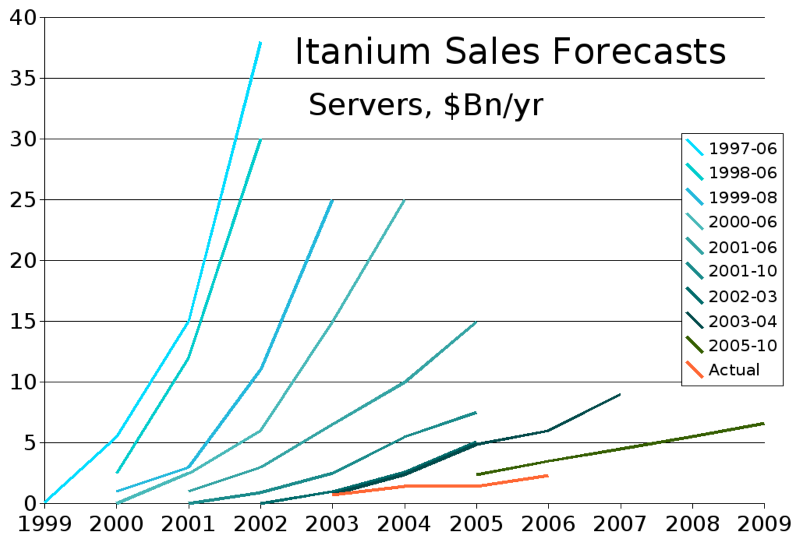
x86: Evolution with backward compatibility

- Further single-thread 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

Intel Pentium II (Photo Asimzb, Wikipedia)

x86: Evolution with backward compatibility

- Intel Itanium/EPIC (Explicitly Parallel Instruction Computing) – IA-64
  - Time to go beyond 32 bits and its 4 GB memory address limitation!
  - Time to go beyond 8080 backwards compatibility!
  - Time to go beyond CPI of 1!
  - “VLIW (Very Long Instruction Word)”
    - Each instruction (“bundle”) consists of three sub-instructions
    - Three instructions issued at once (CPI of 1/3 if lucky)
    - Lots of tricks to deal with data dependencies
    - Difficult design! Delay…
    - Some opinions: Writing compilers was hard…



ZDNet, “Mining Itanium”

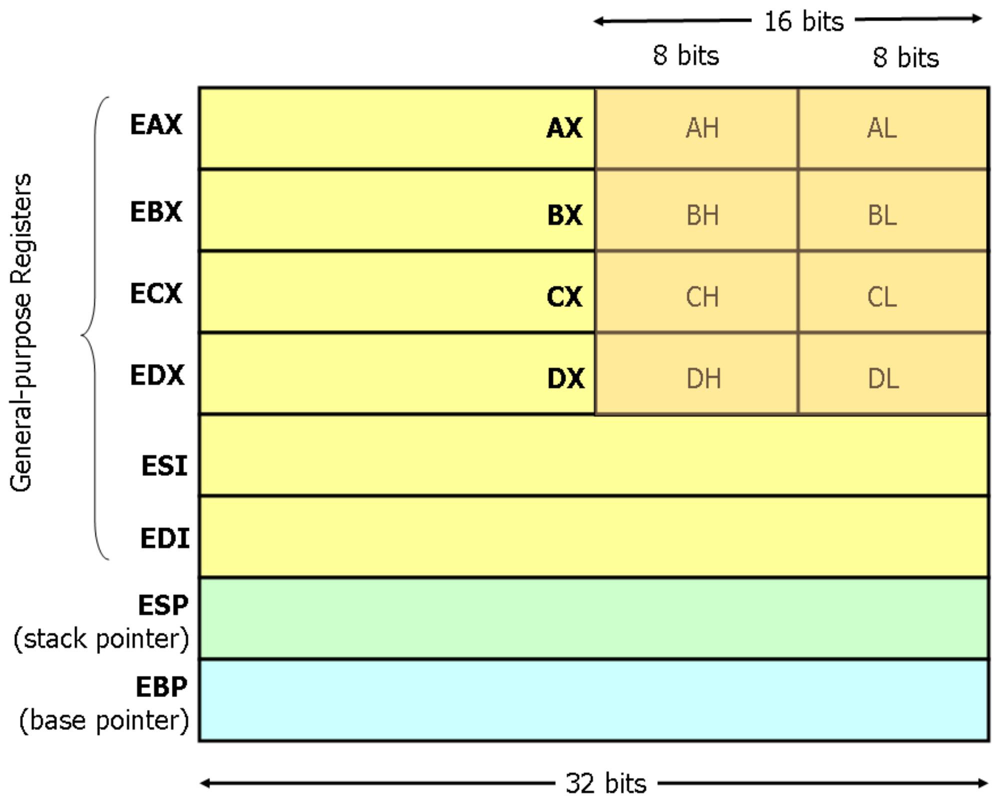
x86: Evolution with backward compatibility

- Meanwhile at AMD: AMD64, or x86-64
  - Backwards compatible architecture extension to 64 bits
  - Later also adopted by Intel
- Intel Core (2006) – Going dual-core
  - 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 x86 – Registers

- Much smaller number of registers compared to RISC-V
- Four ‘general purpose’ registers
  - Naming has historical reasons
  - Originally AX…DX, but ‘Extended’ to 32 bits
  - 64 bit extensions with ‘R’ prefix
- Aside: Now we know four is too little…
- Special registers for stack management
  - RISC-V has no special register (Except x0)



Aside: Intel x86 – Addressing modes

- Typical x86 assembly instructions have many addressing mode variants

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

- e.g., ‘add’ has two input operands, storing the add in the second

```
add <reg>, <reg>
add <mem>, <reg>
add <reg>, <mem>
add <imm>, <reg>
add <imm>, <mem>
```

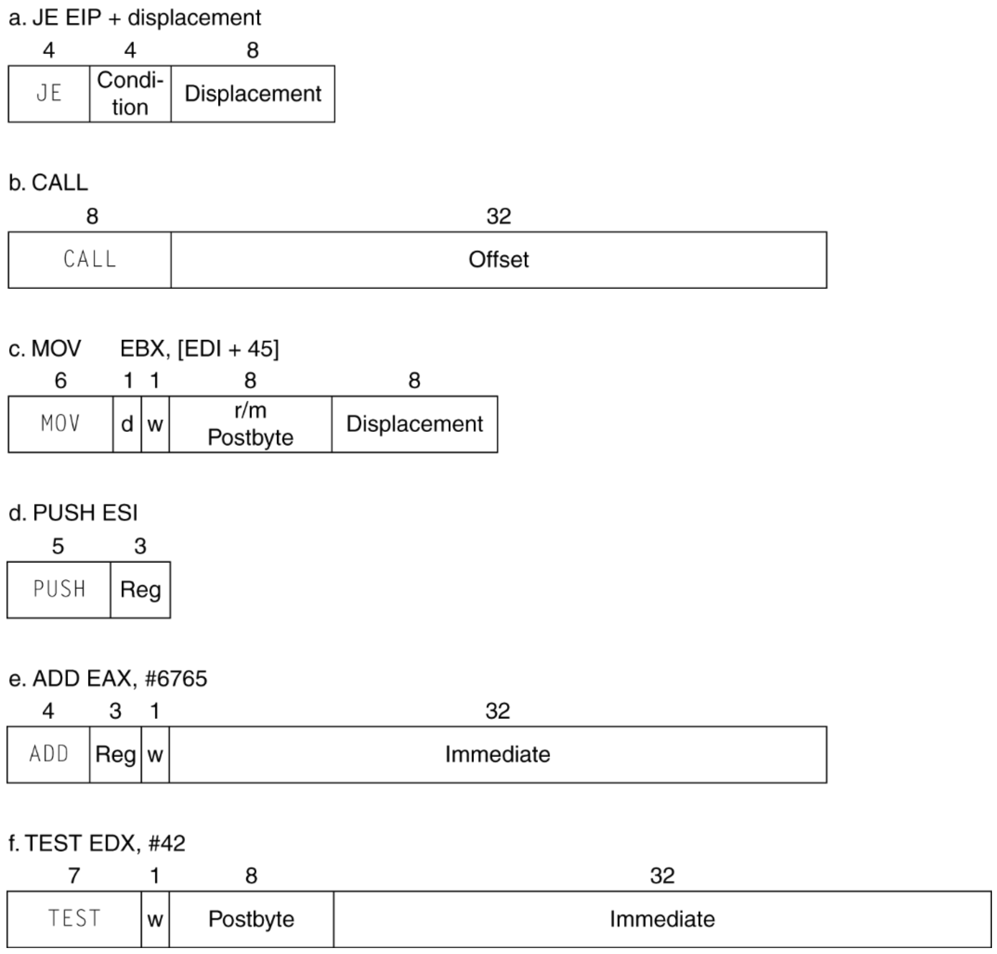
Examples
add $10, %eax — EAX is set to EAX + 10
addb $10, (%eax) — add 10 to the single byte stored at memory address stored in EAX

Example source: Guide to x86 Assembly - Yale FLINT Group

Aside: Intel x86 – Encoding

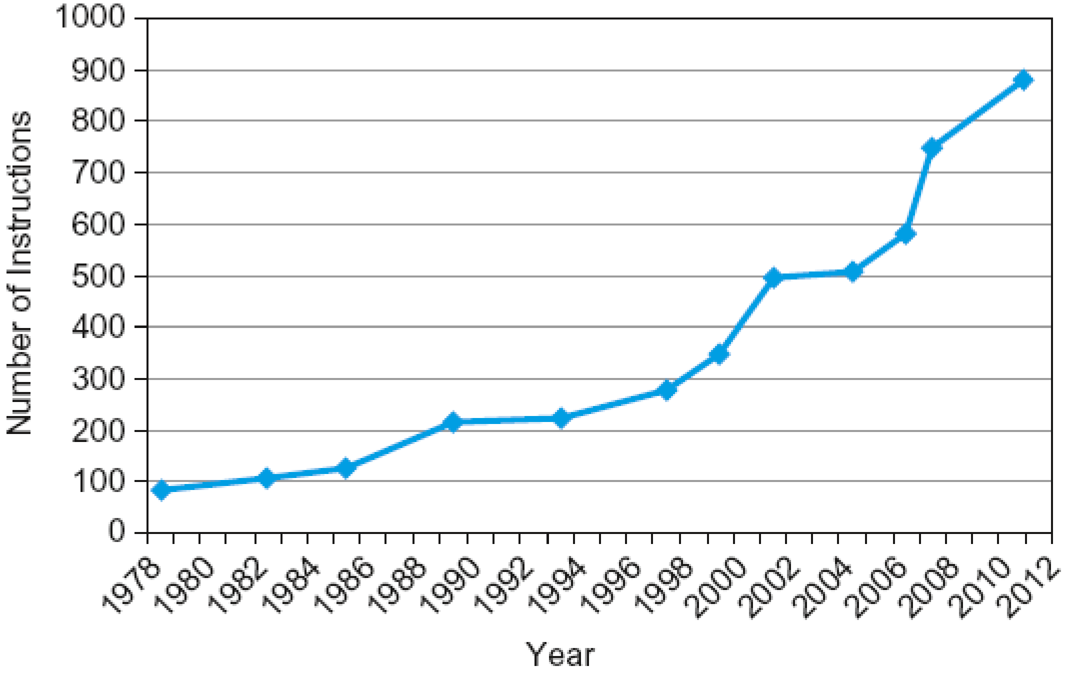
- Many many complex instructions
  - Fixed-size encoding will waste too much space
  - Variable-length encoding!
  - 1 byte – 15 bytes encoding
- Complex decoding logic in hardware
  - Hardware translates instructions to simpler microoperations
    - Simple instructions: 1–1
    - Complex instructions: 1–many
  - Microengine similar to RISC
  - Market share makes this economically viable

Comparable performance to RISC!
Compilers avoid complex instructions



Aside: x86 – Instruction accumulation

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



x86 instruction set

Wrapping up…

- Design principles
  1. Simplicity favors regularity
  2. Smaller is faster
  3. Good design demands good compromises
- Make the common case fast

- Powerful instruction $\nRightarrow$ 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