CS250P: Computer Systems Architecture
Some ISA Classifications

Sang-Woo Jun

Fall 2023

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

Course outline

- Part 1: The Hardware-Software Interface
  - What makes a ‘good’ processor?
  - Assembly language and programming conventions
- Part 2: Recap of digital design
  - Combinational and sequential circuits
  - How their restrictions influence processor design
- Part 3: Computer Architecture
  - Computer Arithmetic
  - Simple and pipelined processors
  - Caches and the memory hierarchy
- Part 4: Computer Systems
  - Operating systems, Virtual memory

Eight great ideas

- ❑ Design for Moore's Law
- ❑ Use abstraction to simplify design
- ❑ Make the common case fast
- ❑ Performance via parallelism
- ❑ Performance via pipelining
- ❑ Performance via prediction
- ❑ Hierarchy of memories
- ❑ Dependability via redundancy

today

The RISC/CISC Classification

- Reduced Instruction-Set Computer (RISC)
  - Precise definition is debated
  - Small number of more general instructions
    - RISC-V base instruction set has only dozens of instructions
    - Memory load/stores not mixed with computation operations
      (Different instructions for load from memory, perform computation in register)
    - Often fixed-width encoding (4 bytes for base RISC-V)
  - Complex operations implemented by composing general ones
    - Compilers try their best!
  - RISC-V, ARM (Advanced RISC Machines),
    MIPS (Microprocessor without Interlocked Pipelined Stages),
    SPARC, …

The RISC/CISC Classification

- Complex Instruction-Set Computer (CISC)
  - Precise definition is debated (Not RISC?)
  - Many, complex instructions
    - Various memory access modes per instruction (load from memory? register? etc)
    - Typically variable-length encoding per instruction
    - Modern x86 has thousands!
  - Intel x86,
    IBM z/Architecture,
  - …

The RISC/CISC Classification

- RISC paradigm is winning out
  - Simpler design allows faster clock
  - Simpler design allows efficient microarchitectural techniques
    - Superscalar, Out-of-order, …
  - Compilers very good at optimizing software
- Most modern CISC processors have RISC internals
  - CISC instructions translated on-the-fly to RISC by the front-end hardware
  - Added overhead from translation (silicon, power, performance, …)

CS250P: Computer Systems Architecture
RISC-V Introduction

Sang-Woo Jun

Fall 2022

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

Why learn assembly?

- We (typically) don't program with assembly any more

- BUT, important to understand architecture
  - Arithmetic in x86 has two operands (e.g., add eax ebx), while RISC-V has three (e.g., add x5 x6 x7)
  - x86 has six general-purpose registers, while RISC-V has 32

  - What drove these decisions? How does this impact processor design and performance?

We need a reference architecture

RISC-V Introduction

- We use RISC-V as a learning tool
- A free and open ISA from Berkeley
  - A clean-slate design using what was learned over decades
  - Uncluttered by backwards compatibility
  - Simplicity-oriented (Some say to a fault!)
- Many, many industry backers!
  - Google, Qualcomm, NVIDIA, IBM, Samsung, Huawei, …

RISC-V Introduction

- Composable, modular design
  - Consists of a base ISA -- RV32I (32 bit), RV64I (64 bit) We will use RV32I
  - And many composable extensions. Including:
    - ‘M’: Math extension. Multiply and divide
    - ‘F’, ‘D’: Floating point extensions, single and double precision
    - ‘A’: Atomic operations
    - ‘B’: Bit manipulation
    - ‘T’: Transactional memory
    - ‘P’: Packed SIMD (Single-Instruction Multiple Data)
    - ‘V’: Vector operators
    - Designer can choose to implement combinations: e.g., RV64IMFT
- Virtual memory (Sv32, Sv48) and privileged operations specified

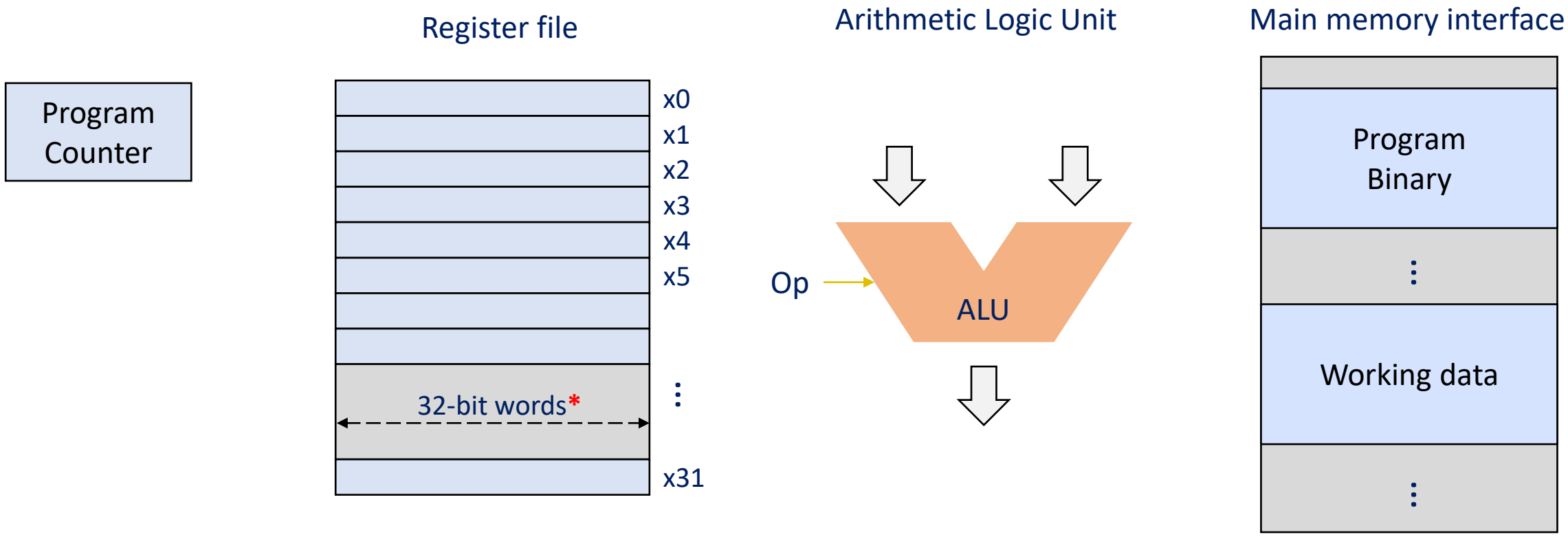
Structure of the ISA

- Small amount of fixed-size registers
  - For RV32I, 32 32-bit registers (32 64-bit registers for RV64)
  - A question: **Why isn't this number larger?** Why not 1024 registers?
  - Another question: Why not zero?
- Three types of instructions

Important!

1. Computational operation: from register file to register file
   - $x_d = Op(x_a, x_b)$, where $Op \in \{+, -, AND, OR, >, <, \ldots\}$
   - Op implemented in ALU
2. Load/Store: between memory and register file
3. Control flow: jump to different part of code

RISC-V base architecture components



- Current location in program execution
- 32 32-bit registers
- (64 bit words for RV64)
- Input: 2 values, Op
- Output: 1 value

Op ∈ {+, -, AND, OR, >, <, …}

- Actual memory outside CPU chip

Super simplified processor operation

inst = **mem**[**PC**]

next_PC = **PC** + 4

if (inst.**type** == **STORE**) **mem**[**rf**[inst.**arg1**]] = **rf**[inst.**arg2**]

if (inst.**type** == **LOAD**) **rf**[inst.**arg1**] = **mem**[**rf**[inst.**arg2**]]

if (inst.**type** == **ALU**) **rf**[inst.**arg1**] = **alu**(inst.**op**, **rf**[inst.**arg2**], **rf**[inst.**arg3**])

if (inst.**type** == **COND**) next_PC = **rf**[inst.**arg1**]

PC = next_PC

RISC-V never mixes memory and ALU operations!

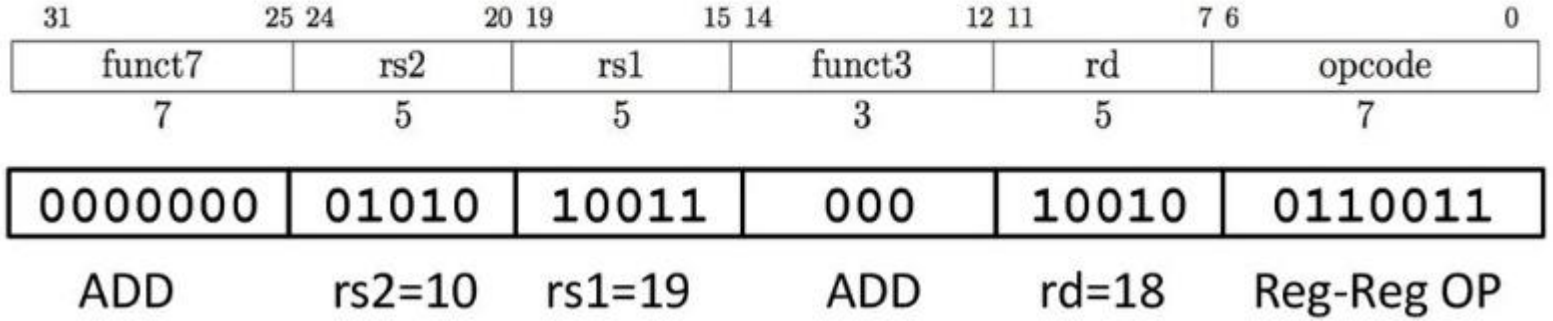
Why not?!

In the four bytes of the instruction,
type, **arg1**, **arg2**, **arg3**, **op**
needs to be encoded

A RISC-V Example (“00A9 8933”)

- This four-byte binary value will instruct a RISC-V CPU to perform
  - add values in registers x19 x10, and store it in x18
  - regardless of processor speed, internal implementation, or chip designer

```
add  x18,x19,x10
```



In the four bytes of the instruction,
type, **arg1**, **arg2**, **arg3**, **op**
needs to be encoded

Source: Yuanqing Cheng, “Great Ideas in Computer Architecture RISC-V Instruction Formats”

CS152: Computer Systems Architecture
Storytime – x86 And Surrounding History

Sang-Woo Jun

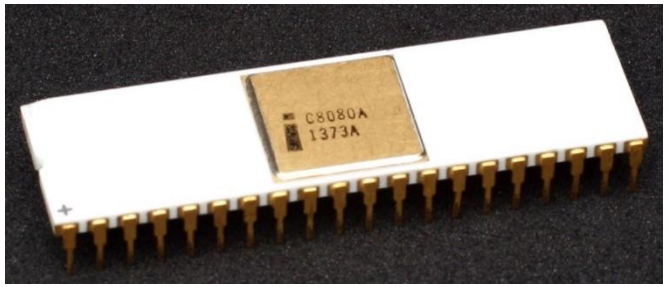
Fall 2023

UCI

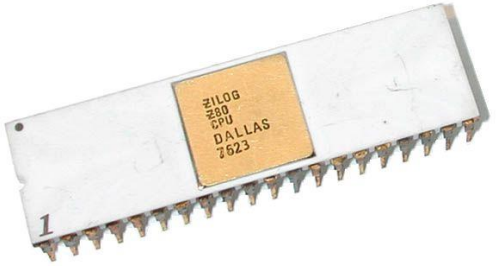
Large amount of material adapted from MIT 6.004, “Computation Structures”, Morgan Kaufmann “Computer Organization and Design: The Hardware/Software Interface: RISC-V Edition”, and CS 152 Slides by Isaac Scherson

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)

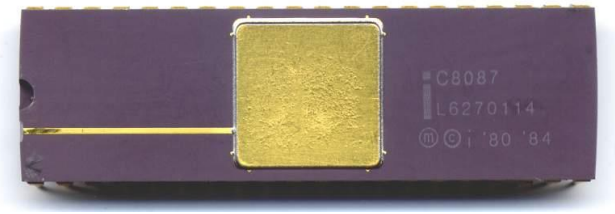


Zilog Z80 (Photo Gennadiy Shvets)

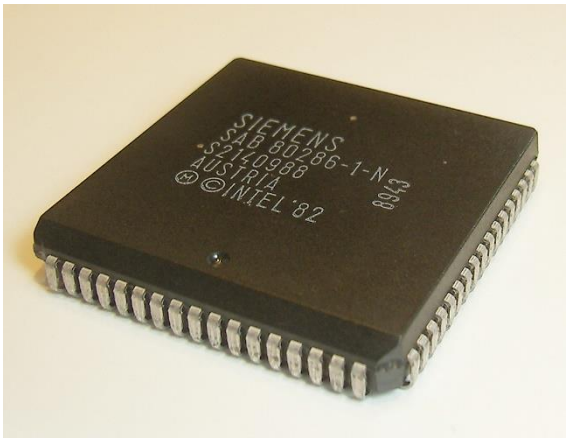
Toshiba Z84C00 (Photo Dhrm77, Wikipedia)

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
  - Widely popular, used in original IBM PC
  - 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)



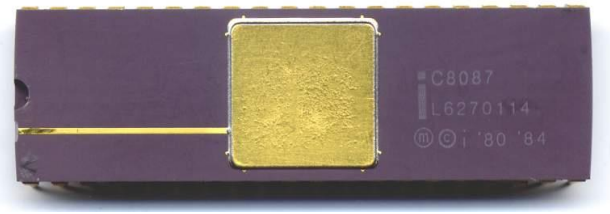
Intel 8087 (Photo Dirk Oppelt)



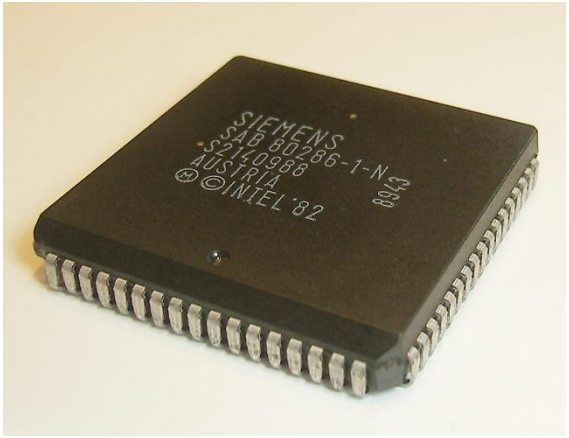
Intel 80286 (Photo Peter Binter)

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…
    - What made windows feasible! Windows 3.1 built on this



Intel 8087 (Photo Dirk Oppelt)



Intel 80286 (Photo Peter Binter)

x86: Evolution with backward compatibility

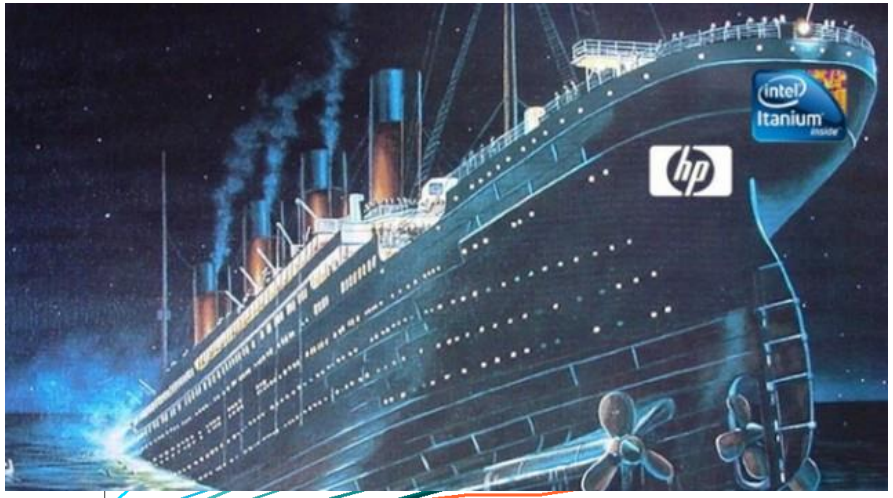
- Further single-thread evolution…
  - i486 (1989): pipelined, on-chip caches and FPU
    - Compatible competitors: AMD, Cyrix, …
  - Pentium (1993): superscalar, 64-bit datapath
    - Later versions added MMX (Multi-Media eXtension) instructions
    - The infamous FDIV bug
  - Pentium Pro (1995), Pentium 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 8080 backwards compatibility!
  - Time to go beyond transparent, single instruction stream!
  - "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…

ExtremeTech "Farewell, Godspeed, Itanic: Intel to Discontinue the Itanium Family"



ZDNet, "Mining Itanium"

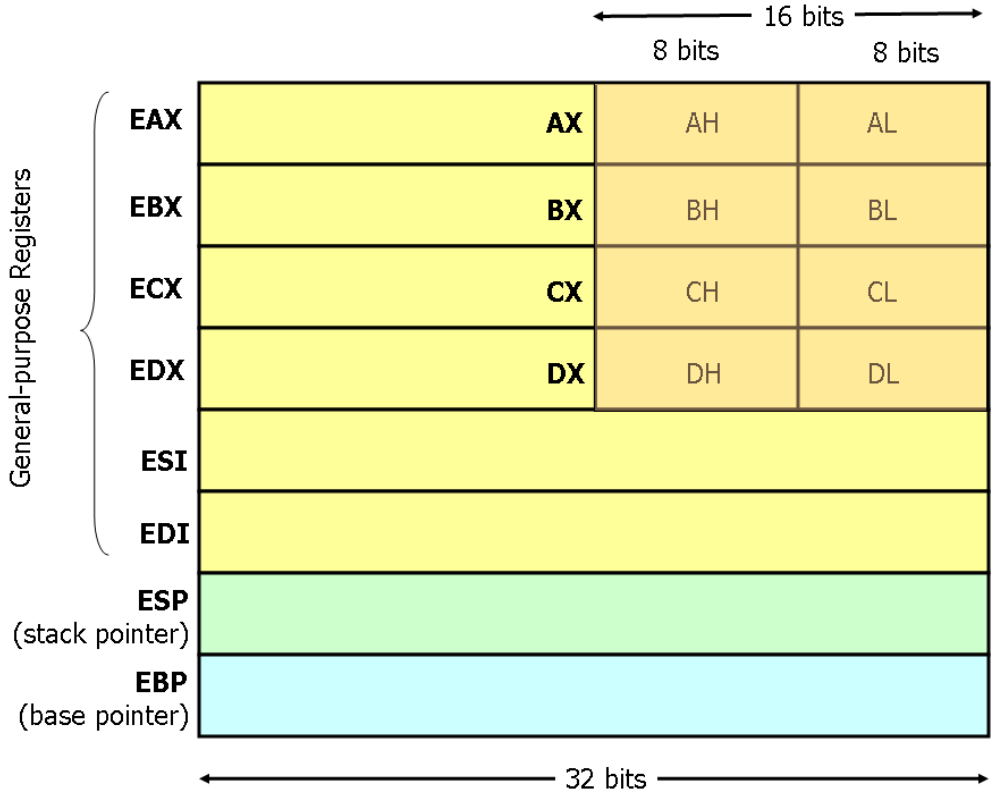
x86: Evolution with backward compatibility

- 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

CISC! But no "Memory -> Memory"

Example source: Guide to x86 Assembly - Yale FLINT Group

Aside: CISC and x86

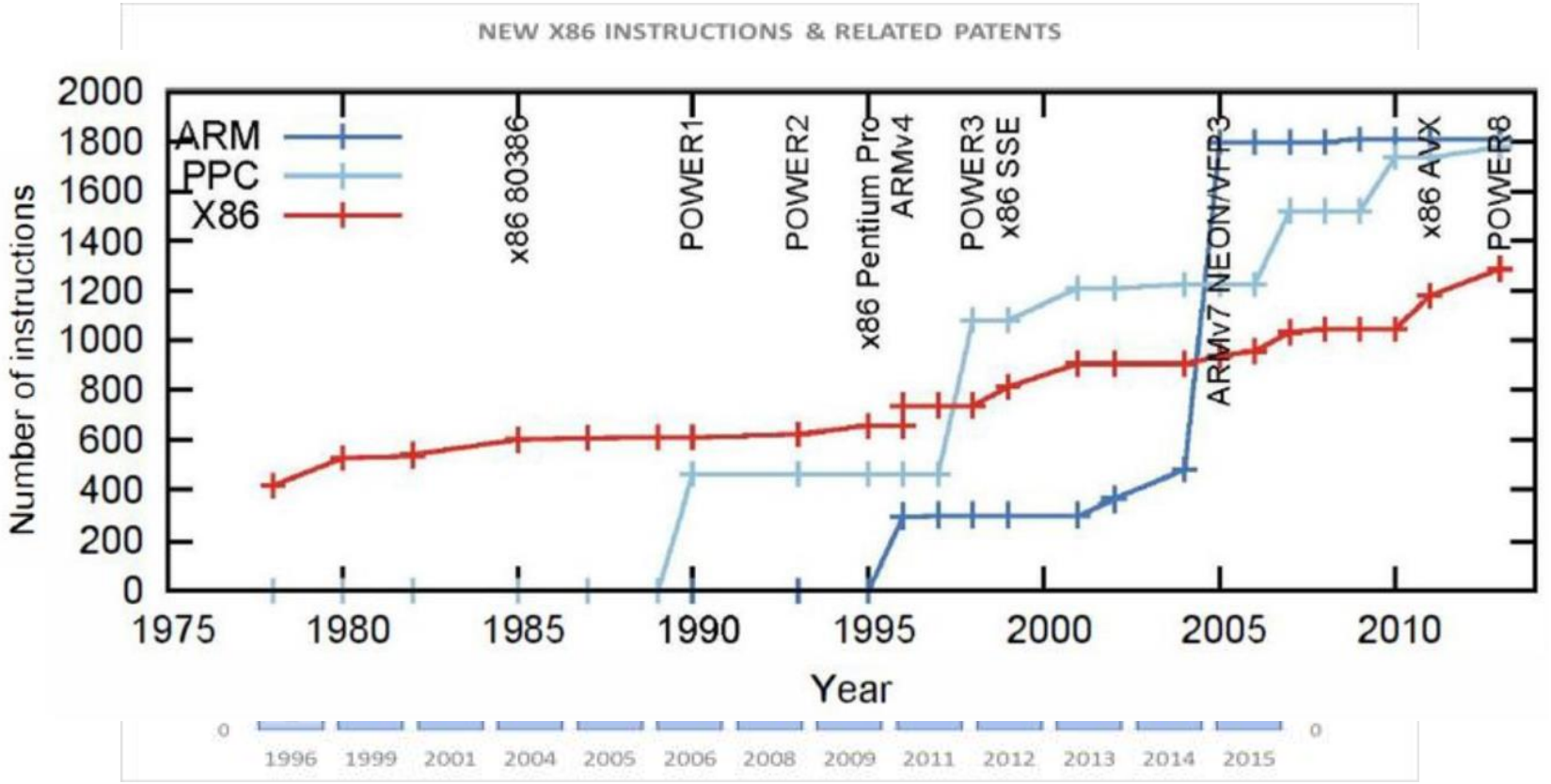
- x86 ISA is CISC ("Complex")

Hex	Mnemonics
C3	ret
48 b8 88 77 66 55 44 33 22 11	movabs rax,0x1122334455667788
64 ff 03	DWORD PTR fs:[ebx]
64 67 66 f0 ff 07	lock inc WORD PTR fs:[bx]
2e c4 e2 71 96 84 be 34 23 12 01	vfmaddsub132ps xmm0, xmm1, xmmword ptr cs: [esi + edi * 4 + 0x11223344]

Philipp Koppe et.al., "Reverse Engineering x86 Processor Microcode," USENIX security 2017

Aside: x86 – Instruction accumulation

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



Source: Intel Newsroom"X86: Approaching 40 and Still Going Strong"

Source: Dezső Sima "Client processors - 2 x86 ISA extensions"