CS252 Graduate Computer Architecture
Spring 2014
Lecture 3: CISC versus RISC

Krste Asanovic
krste@eecs.berkeley.edu
http://inst.eecs.berkeley.edu/~cs252/sp14
First 130 years of CompArch, from Babbage to IBM 360
- Move from calculators (no conditionals) to fully programmable machines
- Rapid change started in WWII (mid-1940s), move from electro-mechanical to pure electronic processors

Cost of software development becomes a large constraint on architecture (need compatibility)

IBM 360 introduces notion of “family of machines” running same ISA but very different implementations
- Six different machines released on same day (April 7, 1964)
- “Future-proofing” for subsequent generations of machine
Instruction Set Architecture (ISA)

- The contract between software and hardware
- Typically described by giving all the programmer-visible state (registers + memory) plus the semantics of the instructions that operate on that state
- IBM 360 was first line of machines to separate ISA from implementation (aka. microarchitecture)
- Many implementations possible for a given ISA
  - E.g., the Soviets build code-compatible clones of the IBM360, as did Amdahl after he left IBM.
  - E.g.2., today you can buy AMD or Intel processors that run the x86-64 ISA.
  - E.g.3: many cellphones use the ARM ISA with implementations from many different companies including TI, Qualcomm, Samsung, Marvell, etc.
- We use Berkeley RISC-V 2.0 as standard ISA in class
  - www.riscv.org
Control versus Datapath

- Processor designs can be split between *datapath*, where numbers are stored and arithmetic operations computed, and *control*, which sequences operations on datapath.
- Biggest challenge for early computer designers was getting control circuitry correct.

Maurice Wilkes invented the idea of microprogramming to design the control unit of a processor for EDSAC-II, 1958.
- Foreshadowed by Babbage’s “Barrel” and mechanisms in earlier programmable calculators.
Technology Influence

- When microcode appeared in 50s, different technologies for:
  - Logic: Vacuum Tubes
  - Main Memory: Magnetic cores
  - Read-Only Memory: Diode matrix, punched metal cards,

- Logic very expensive compared to ROM or RAM
- ROM cheaper than RAM
- ROM much faster than RAM
Microcoded CPU

Microcode ROM
(holds fixed µcode instructions)

Datapath

Main Memory
(holds user program written in macroinstructions, e.g., x86, RISC-V)
Microinstructions written as register transfers:

- **MA:=PC** means RegSel=PC; RegW=0; RegEn=1; MALd=1
- **B:=Reg[rs2]** means RegSel=rs2; RegW=0; RegEn=1; BLd=1
- **Reg[rd]:=A+B** means ALUop=Add; ALUEn=1; RegSel=rd; RegW=1
RISC-V Instruction Execution Phases

- Instruction Fetch
- Instruction Decode
- Register Fetch
- ALU Operations
- Optional Memory Operations
- Optional Register Writeback
- Calculate Next Instruction Address
Microcode Sketches (1)

Instruction Fetch:

\[
\begin{align*}
&MA, A := PC \\
&PC := A + 4 \\
&wait for memory \\
&IR := Mem \\
&dispatch on opcode
\end{align*}
\]

ALU:

\[
\begin{align*}
&A := Reg[rs1] \\
&B := Reg[rs2] \\
&Reg[rd] := ALUOp(A, B) \\
&goto instruction fetch
\end{align*}
\]

ALUI:

\[
\begin{align*}
&A := Reg[rs1] \\
&B := ImmI \quad // Sign-extend 12b immediate \\
&Reg[rd] := ALUOp(A, B) \\
&goto instruction fetch
\end{align*}
\]
Microcode Sketches (2)

LW:
A:=Reg[rs1]
B:=ImmI //Sign-extend 12b immediate
MA:=A+B
wait for memory
Reg[rd]:=Mem

goto instruction fetch

JAL:
Reg[rd]:=A // Store return address
A:=A-4 // Recover original PC
B:=ImmJ // Jump-style immediate
PC:=A+B

goto instruction fetch

Branch:
A:=Reg[rs1]
B:=Reg[rs2]
if (!ALUOp(A,B)) goto instruction fetch //Not taken
A:=PC //Microcode fall through if branch taken
A:=A-4
B:=ImmB
PC:=A+B

goto instruction fetch
Pure ROM Implementation

- How many address bits?
  \[ |\mu\text{address}| = |\mu\text{PC}| + |\text{opcode}| + 1 + 1 \]
- How many data bits?
  \[ |\text{data}| = |\mu\text{PC}| + |\text{control signals}| = |\mu\text{PC}| + 18 \]
- Total ROM size = \[2^{|\mu\text{address}|} \times |\text{data}|\]
## Pure ROM Contents

<table>
<thead>
<tr>
<th>Address</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>μPC</td>
<td>Opcode Cond? Busy?</td>
</tr>
<tr>
<td>fetch0</td>
<td>X X X</td>
</tr>
<tr>
<td>fetch1</td>
<td>X X 1</td>
</tr>
<tr>
<td>fetch1</td>
<td>X X 0</td>
</tr>
<tr>
<td>fetch2</td>
<td>ALU X X</td>
</tr>
<tr>
<td>fetch2</td>
<td>ALUI X X</td>
</tr>
<tr>
<td>fetch2</td>
<td>LW X X</td>
</tr>
</tbody>
</table>

ALU0 X X X X | A:=Reg[rs1] | ALU1
ALU1 X X X X | B:=Reg[rs2] | ALU2
ALU2 X X X X | Reg[rd]:=ALUOp(A,B) | fetch0
Single-Bus Microcode RISC-V ROM Size

- Instruction fetch sequence 3 common steps
- ~12 instruction groups
- Each group takes ~5 steps (1 for dispatch)
- Total steps $3 + 12 \times 5 = 63$, needs 6 bits for $\mu$PC

- Opcode is 5 bits, ~18 control signals

- Total size $= 2^{(6+5+2)} \times (6+18) = 2^{13} \times 24 = \sim 25$KB!
Reducing Control Store Size

- Reduce ROM height (#address bits)
  - Use external logic to combine input signals
  - Reduce #states by grouping opcodes

- Reduce ROM width (#data bits)
  - Restrict µPC encoding (next,dispatch,wait on memory,...)
  - Encode control signals (vertical µcoding, nanocoding)
Single-Bus RISC-V Microcode Engine

µPC jump = next | spin | fetch | dispatch | ftrue | ffalse
μPC Jump Types

- *next* increments μPC
- *spin* waits for memory
- *fetch* jumps to start of instruction fetch
- *dispatch* jumps to start of decoded opcode group
- *future/ffalse* jumps to fetch if Cond? true/false
## Encoded ROM Contents

<table>
<thead>
<tr>
<th>Address</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>μPC</td>
<td>Control Lines</td>
</tr>
<tr>
<td>fetch0</td>
<td>MA, A:= PC</td>
</tr>
<tr>
<td>fetch1</td>
<td>IR:= Mem</td>
</tr>
<tr>
<td>fetch2</td>
<td>PC:= A+4</td>
</tr>
<tr>
<td>ALU0</td>
<td>A:= Reg[rs1]</td>
</tr>
<tr>
<td>ALU1</td>
<td>B:= Reg[rs2]</td>
</tr>
<tr>
<td>ALU2</td>
<td>Reg[rd]:= ALUOp(A, B)</td>
</tr>
<tr>
<td>Branch0</td>
<td>A:= Reg[rs1]</td>
</tr>
<tr>
<td>Branch1</td>
<td>B:= Reg[rs2]</td>
</tr>
<tr>
<td>Branch2</td>
<td>A:= PC</td>
</tr>
<tr>
<td>Branch3</td>
<td>A:= A-4</td>
</tr>
<tr>
<td>Branch4</td>
<td>B:= ImmB</td>
</tr>
<tr>
<td>Branch5</td>
<td>PC:= A+B</td>
</tr>
</tbody>
</table>

Next μPC: next
Spin: spin
Dispatch: dispatch
False: ffalse
Fetch: fetch
## Implementing Complex Instructions


<table>
<thead>
<tr>
<th>Address</th>
<th>Data</th>
<th>Next µPC</th>
</tr>
</thead>
<tbody>
<tr>
<td>µPC</td>
<td>Control Lines</td>
<td></td>
</tr>
<tr>
<td>MMA0</td>
<td>MA:=Reg[rs1]</td>
<td>next</td>
</tr>
<tr>
<td>MMA1</td>
<td>A:=Mem</td>
<td>spin</td>
</tr>
<tr>
<td>MMA2</td>
<td>MA:=Reg[rs2]</td>
<td>next</td>
</tr>
<tr>
<td>MMA3</td>
<td>B:=Mem</td>
<td>spin</td>
</tr>
<tr>
<td>MMA4</td>
<td>MA:=Reg[rd]</td>
<td>next</td>
</tr>
<tr>
<td>MMA5</td>
<td>Mem:=ALUOp(A,B)</td>
<td>spin</td>
</tr>
<tr>
<td>MMA6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Complex instructions usually do not require datapath modifications, only extra space for control program.

Very difficult to implement these instructions using a hardwired controller without substantial datapath modifications.
Horizontal vs Vertical μCode

- **Horizontal μcode has wider µinstructions**
  - Multiple parallel operations per µinstruction
  - Fewer microcode steps per macroinstruction
  - Sparser encoding ⇒ more bits

- **Vertical μcode has narrower µinstructions**
  - Typically a single datapath operation per µinstruction
  - separate µinstruction for branches
  - More microcode steps per macroinstruction
  - More compact ⇒ less bits

- **Nanocoding**
  - Tries to combine best of horizontal and vertical μcode
Nanocoding

Exploits recurring control signal patterns in μcode, e.g.,

\[ \text{ALU}_0 \ A \leftarrow \text{Reg}[\text{rs1}] \]

... \[ \text{ALU}_{i_0} \ A \leftarrow \text{Reg}[\text{rs1}] \]

- MC68000 had 17-bit μcode containing either 10-bit μjump or 9-bit nanoinstruction pointer
  - Nanoinstructions were 68 bits wide, decoded to give 196 control signals
IBM 360: Initial Implementations

<table>
<thead>
<tr>
<th></th>
<th>Model 30</th>
<th>...</th>
<th>Model 70</th>
</tr>
</thead>
<tbody>
<tr>
<td>Storage</td>
<td>8K - 64 KB</td>
<td></td>
<td>256K - 512 KB</td>
</tr>
<tr>
<td>Datapath</td>
<td>8-bit</td>
<td></td>
<td>64-bit</td>
</tr>
<tr>
<td>Circuit Delay</td>
<td>30 nsec/level</td>
<td></td>
<td>5 nsec/level</td>
</tr>
<tr>
<td>Local Store</td>
<td>Main Store</td>
<td></td>
<td>Transistor Registers</td>
</tr>
<tr>
<td>Control Store</td>
<td>Read only 1µsec</td>
<td></td>
<td>Conventional circuits</td>
</tr>
</tbody>
</table>

IBM 360 instruction set architecture (ISA) completely hid the underlying technological differences between various models.

Milestone: The first true ISA designed as portable hardware-software interface!

With minor modifications it still survives today!
Microprogramming in IBM 360

<table>
<thead>
<tr>
<th></th>
<th>M30</th>
<th>M40</th>
<th>M50</th>
<th>M65</th>
</tr>
</thead>
<tbody>
<tr>
<td>Datapath width (bits)</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td>64</td>
</tr>
<tr>
<td>µinst width (bits)</td>
<td>50</td>
<td>52</td>
<td>85</td>
<td>87</td>
</tr>
<tr>
<td>µcode size (K µinsts)</td>
<td>4</td>
<td>4</td>
<td>2.75</td>
<td>2.75</td>
</tr>
<tr>
<td>µstore technology</td>
<td>CCROS</td>
<td>TCROS</td>
<td>BCROS</td>
<td>BCROS</td>
</tr>
<tr>
<td>µstore cycle (ns)</td>
<td>750</td>
<td>625</td>
<td>500</td>
<td>200</td>
</tr>
<tr>
<td>memory cycle (ns)</td>
<td>1500</td>
<td>2500</td>
<td>2000</td>
<td>750</td>
</tr>
<tr>
<td>Rental fee ($K/month)</td>
<td>4</td>
<td>7</td>
<td>15</td>
<td>35</td>
</tr>
</tbody>
</table>

- Only the fastest models (75 and 95) were hardwired
Microcode Emulation

- IBM initially miscalculated the importance of software compatibility with earlier models when introducing the 360 series
- Honeywell stole some IBM 1401 customers by offering translation software ("Liberator") for Honeywell H200 series machine
- IBM retaliated with optional additional microcode for 360 series that could emulate IBM 1401 ISA, later extended for IBM 7000 series
  - one popular program on 1401 was a 650 simulator, so some customers ran many 650 programs on emulated 1401s
  - (650 simulated on 1401 emulated on 360)
Microprogramming thrived in Seventies

- Significantly faster ROMs than DRAMs were available
- For complex instruction sets, datapath and controller were cheaper and simpler
- New instructions, e.g., floating point, could be supported without datapath modifications
- Fixing bugs in the controller was easier
- ISA compatibility across various models could be achieved easily and cheaply

Except for the cheapest and fastest machines, all computers were microprogrammed
“Iron Law” of Processor Performance

\[
\text{Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Time}}{\text{Cycle}}
\]

- Instructions per program depends on source code, compiler technology, and ISA
- Cycles per instructions (CPI) depends on ISA and \( \mu \)-architecture
- Time per cycle depends upon the \( \mu \)-architecture and base technology
CPI for Microcoded Machine

7 cycles 5 cycles 10 cycles
Inst 1 Inst 2 Inst 3

Total clock cycles = 7+5+10 = 22
Total instructions = 3
CPI = 22/3 = 7.33
CPI is always an average over a large number of instructions.
First Microprocessor
Intel 4004, 1971

- 4-bit accumulator architecture
- 8µm pMOS
- 2,300 transistors
- 3 x 4 mm²
- 750kHz clock
- 8-16 cycles/inst.

Made possible by new integrated circuit technology
Microprocessors in the Seventies

- Initial target was embedded control
  - First micro, 4-bit 4004 from Intel, designed for a desktop printing calculator
  - Constrained by what could fit on single chip
  - Accumulator architectures, similar to earliest computers
  - Hardwired state machine control

- 8-bit micros (8085, 6800, 6502) used in hobbyist personal computers
  - Micral, Altair, TRS-80, Apple-II
  - Usually had 16-bit address space (up to 64KB directly addressable)
  - Often came with simple BASIC language interpreter built into ROM or loaded from cassette tape.
VisiCalc – the first “killer” app for micros

• Microprocessors had little impact on conventional computer market until VisiCalc spreadsheet for Apple-II
• Apple-II used Mostek 6502 microprocessor running at 1MHz

Floppy disks were originally invented by IBM as a way of shipping IBM 360 microcode patches to customers!

[ Personal Computing Ad, 1979 ]
DRAM in the Seventies

- Dramatic progress in semiconductor memory technology
- 1970, Intel introduces first DRAM, 1Kbit 1103
- 1979, Fujitsu introduces 64Kbit DRAM

=> By mid-Seventies, obvious that PCs would soon have >64KBytes physical memory
Microprocessor Evolution

- Rapid progress in 70s, fueled by advances in MOSFET technology and expanding markets
- Intel i432
  - Most ambitious seventies’ micro; started in 1975 - released 1981
  - 32-bit capability-based object-oriented architecture
  - Instructions variable number of bits long
  - Severe performance, complexity, and usability problems
- Motorola 68000 (1979, 8MHz, 68,000 transistors)
  - Heavily microcoded (and nanocoded)
  - 32-bit general-purpose register architecture (24 address pins)
  - 8 address registers, 8 data registers
- Intel 8086 (1978, 8MHz, 29,000 transistors)
  - “Stopgap” 16-bit processor, architected in 10 weeks
  - Extended accumulator architecture, assembly-compatible with 8080
  - 20-bit addressing through segmented addressing scheme
IBM PC, 1981

- **Hardware**
  - Team from IBM building PC prototypes in 1979
  - Motorola 68000 chosen initially, but 68000 was late
  - IBM builds “stopgap” prototypes using 8088 boards from Display Writer word processor
  - 8088 is 8-bit bus version of 8086 => allows cheaper system
  - Estimated sales of 250,000
  - 100,000,000s sold

- **Software**
  - Microsoft negotiates to provide OS for IBM. Later buys and modifies QDOS from Seattle Computer Products.

- **Open System**
  - Standard processor, Intel 8088
  - Standard interfaces
  - Standard OS, MS-DOS
  - IBM permits cloning and third-party software
Presenting the IBM of Personal Computers.

IBM is proud to announce a product you may have a personal interest in. It's a tool that could soon be on your desk, in your home or in your child's schoolroom. It can make a surprising difference in the way you work, learn or otherwise approach the complexities (and some of the simple pleasures) of living.

It's the computer we're making for you.

In the past 30 years, the computer has become faster, smaller, less complicated and less expensive. And IBM has contributed heavily to that evolution.

Today, we've applied what we know to a new product we believe in: the IBM Personal Computer.

**IBM PERSONAL COMPUTER SPECIFICATIONS**

- **User Memory**
  - 16K - 256K bytes
- **Permanent Memory**
  - 8K-4096 bytes
- **Microprocessor**
  - High-speed, 8088
- **Auxiliary Memory**
  - 2 optional internal drives, 5\(\frac{1}{4}\)", 1300 bytes
- **Keys**
  - 10 keys, 8-bit, attach to a keyboard unit
- **Keyboard**
  - 10 keys, 8-bit, attach to a computer unit

**Display Screen**

- High-resolution
- 1728 x 2560 pixels
- 80 characters x 25 lines
- Upper and lower case
- Graphics, 5-line and 25-line

**Color/Graphics**

- 256 characters and symbols in IBM
- Graphics model
- Color menu
- 5200 x 2000

**Diagnostics**

- Built-in self-testing
- Error checking
- IBM BCD basic
- IBM BASIC, Pascal

**Communications**

- RS-232C interface
- Interface to printer and typewriter

**Printer**

- Up to 6000 lines per second

It's a computer that has reached a truly personal scale in size and in price: starting at less than $1,600 for a system that, with the addition of one simple device, hooks up to your home TV and uses your audio cassette recorder.

For flexibility, performance and ease of use, no other personal computer offers as many advanced features to please novice and expert alike (see the box). Features like high-resolution color graphics, ten user-defined function keys, the kind of expandability that lets you add a printer for word processing, or user memory up to 256KB. Or BASIC and Pascal languages that let you write your own programs. And a growing list of superior programs like VisiCalc, selected by IBM to match the quality and thoroughness of the system's total design.

This new system will be sold through channels which meet our professional criteria: the nationwide chain of 150 ComputerLand stores, and Sears Business Systems Centers. Of course, our own IBM Product Centers will sell and service the system. And the IBM Data Processing Division will serve those customers who want to purchase in quantity.

Experience the IBM Personal Computer. You'll be surprised how quickly you feel comfortable with it. And impressed with what it can do for you.

[ Personal Computing Ad, 11/81]
Microprogramming: early Eighties

- Evolution bred more complex micro-machines
  - Complex instruction sets led to need for subroutine and call stacks in μcode
  - Need for fixing bugs in control programs was in conflict with read-only nature of μROM
  - ➔ Writable Control Store (WCS) (B1700, QMachine, Intel i432, ...)

- With the advent of VLSI technology assumptions about ROM & RAM speed became invalid ➔ more complexity

- Better compilers made complex instructions less important.

- Use of numerous micro-architectural innovations, e.g., pipelining, caches and buffers, made multiple-cycle execution of reg-reg instructions unattractive
Writable Control Store (WCS)

- Implement control store in RAM not ROM
  - MOS SRAM memories now almost as fast as control store (core memories/DRAMs were 2-10x slower)
  - Bug-free microprograms difficult to write

- User-WCS provided as option on several minicomputers
  - Allowed users to change microcode for each processor

- User-WCS failed
  - Little or no programming tools support
  - Difficult to fit software into small space
  - Microcode control tailored to original ISA, less useful for others
  - Large WCS part of processor state - expensive context switches
  - Protection difficult if user can change microcode
  - Virtual memory required restartable microcode
Analyzing Microcoded Machines

- John Cocke and group at IBM
  - Working on a simple pipelined processor, 801, and advanced compilers inside IBM
  - Ported experimental PL.8 compiler to IBM 370, and only used simple register-register and load/store instructions similar to 801
  - Code ran faster than other existing compilers that used all 370 instructions! (up to 6MIPS whereas 2MIPS considered good before)

- Emer, Clark, at DEC
  - Measured VAX-11/780 using external hardware
  - Found it was actually a 0.5MIPS machine, although usually assumed to be a 1MIPS machine
  - Found 20% of VAX instructions responsible for 60% of microcode, but only account for 0.2% of execution time!

- VAX8800
  - Control Store: 16K*147b RAM, Unified Cache: 64K*8b RAM
  - 4.5x more microstore RAM than cache RAM!
IC Technology Changes Tradeoffs

- Logic, RAM, ROM all implemented using MOS transistors
- Semiconductor RAM ~ same speed as ROM
Exploits recurring control signal patterns in μcode, e.g.,

- \( \text{ALU}_0 \) \( A \leftarrow \text{Reg}[rs1] \)
- \( \ldots \)
- \( \text{ALU}_i \) \( A \leftarrow \text{Reg}[rs1] \)
- \( \ldots \)

- **MC68000** had 17-bit μcode containing either 10-bit μjump or 9-bit nanoinstruction pointer
  - Nanoinstructions were 68 bits wide, decoded to give 196 control signals
From CISC to RISC

- Use fast RAM to build fast instruction cache of user-visible instructions, not fixed hardware microroutines
  - Contents of fast instruction memory change to fit what application needs right now
- Use simple ISA to enable hardwired pipelined implementation
  - Most compiled code only used a few of the available CISC instructions
  - Simpler encoding allowed pipelined implementations
- Further benefit with integration
  - In early ‘80s, could finally fit 32-bit datapath + small caches on a single chip
  - No chip crossings in common case allows faster operation
Berkeley RISC Chips

RISC-I (1982) Contains 44,420 transistors, fabbed in 5 \( \mu m \) NMOS, with a die area of 77 mm\(^2\), ran at 1 MHz. This chip is probably the first VLSI RISC.

RISC-II (1983) contains 40,760 transistors, was fabbed in 3 \( \mu m \) NMOS, ran at 3 MHz, and the size is 60 mm\(^2\).

Stanford built some too…
Microprogramming is far from extinct

- Played a crucial role in micros of the Eighties
  - DEC uVAX, Motorola 68K series, Intel 286/386

- Plays an assisting role in most modern micros
  - e.g., AMD Bulldozer, Intel Ivy Bridge, Intel Atom, IBM PowerPC, ...
  - Most instructions executed directly, i.e., with hard-wired control
  - Infrequently-used and/or complicated instructions invoke microcode

- Patchable microcode common for post-fabrication bug fixes, e.g. Intel processors load µcode patches at bootup
Acknowledgements

- This course is partly inspired by previous MIT 6.823 and Berkeley CS252 computer architecture courses created by my collaborators and colleagues:
  - Arvind (MIT)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)
  - David Patterson (UCB)