CS 152 Computer Architecture and Engineering

Lecture 3 - From CISC to RISC

Dr. George Michelogiannakis
EECS, University of California at Berkeley
CRD, Lawrence Berkeley National Laboratory

http://inst.eecs.berkeley.edu/~cs152
ISA is the hardware/software interface
- Defines set of programmer visible state
- Defines instruction format (bit encoding) and instruction semantics
- Examples: IBM 360, MIPS, RISC-V, x86, JVM

Many possible implementations of one ISA
- 360 implementations: model 30 (c. 1964), z12 (c. 2012)
- x86 implementations: 8086 (c. 1978), 80186, 286, 386, 486, Pentium, Pentium Pro, Pentium-4 (c. 2000), Core 2 Duo, Nehalem, Sandy Bridge, Ivy Bridge, Atom, AMD Athlon, Transmeta Crusoe, SoftPC
- MIPS implementations: R2000, R4000, R10000, R18K, ...
- JVM: HotSpot, PicoJava, ARM Jazelle, ...

Microcoding: straightforward methodical way to implement machines with low logic gate count and complex instructions
Question of the Day

- Do you think a CISC or RISC single-cycle processor would be faster?
“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

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
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

But seventies brought advances in integrated circuit technology and semiconductor memory…
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

\[ \text{=> By mid-Seventies, obvious that PCs would soon have } \]
\[ >64K\text{Bytes physical memory} \]
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
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
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
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 µm NMOS, with a die area of 77 mm², ran at 1 MHz. This chip is probably the first VLSI RISC.

RISC-II (1983) contains 40,760 transistors, was fabbed in 3 µm NMOS, ran at 3 MHz, and the size is 60 mm².

Stanford built some too…
“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

<table>
<thead>
<tr>
<th>Microarchitecture</th>
<th>CPI</th>
<th>cycle time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microcoded</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Single-cycle unpipelined</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pipelined</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This lecture
“Iron Law” of Processor Performance

<table>
<thead>
<tr>
<th>Time</th>
<th>Instructions</th>
<th>Cycles</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program</td>
<td>Program</td>
<td>Instruction</td>
<td>Cycle</td>
</tr>
</tbody>
</table>

Instructions per program depends on source code, compiler technology, and ISA

Cycles per instructions (CPI) depends on ISA and μarchitecture

Time per cycle depends upon the μarchitecture and base technology

<table>
<thead>
<tr>
<th>Microarchitecture</th>
<th>CPI</th>
<th>cycle time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microcoded</td>
<td>&gt;1</td>
<td>short</td>
</tr>
<tr>
<td>Single-cycle unpipelined</td>
<td>1</td>
<td>long</td>
</tr>
<tr>
<td>Pipelined</td>
<td>1</td>
<td>short</td>
</tr>
</tbody>
</table>

This lecture
Hardware Elements

- **Combinational circuits**
  - Mux, Decoder, ALU, ...

- **Synchronous state elements**
  - Flipflop, Register, Register file, SRAM, DRAM

**Edge-triggered:** Data is sampled at the rising edge
Register Files

- Reads are combinational
Register File Implementation

- RISC-V integer instructions have at most 2 register source operands
A Simple Memory Model

Reads and writes are always completed in one cycle
- a Read can be done any time (i.e. combinational)
- a Write is performed at the rising clock edge if it is enabled

=> the write address and data must be stable at the clock edge

Later in the course we will present a more realistic model of memory
Implementing RISC-V
Without a bus

Single-cycle per instruction
datapath & control logic
(Should be review of CS61C)
Instruction Execution

Execution of an instruction involves

1. Instruction fetch
2. Decode and register fetch
3. ALU operation
4. Memory operation (optional)
5. Write back (optional)

and compute address of next instruction
Datapath: Reg-Reg ALU Instructions

RegWrite Timing?

rd ← (rs1) func (rs2)
Datapath: Reg-Imm ALU Instructions

```
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Opcode</th>
<th>Rs1</th>
<th>Func3</th>
<th>Rd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td>0x4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

rd ← (rs1) op immediate
Conflicts in Merging Datapath

Introduce muxes

rd ← (rs1) func (rs2)
rd ← (rs1) op immediate
Datapath for ALU Instructions

rd ← (rs1) func (rs2)
rd ← (rs1) op immediate

**Instruction Format:**
- **OpCode:** 5 bits
- **Func3:** 2 bits
- **Func7:** 3 bits
- **Rs2:** 3 bits
- **Rs1:** 3 bits
- **Immediate12:** 12 bits

**ALU Control:**
- **<6:0>**
- **<19:15>**
- **<24:20>**
- **<11:7>**

**RegWriteEn:**
- **we:** 1 bit
- **rd1:** 1 bit
- **rd2:** 1 bit

**ALU Inputs:**
- **rs1:** 3 bits
- **rs2:** 3 bits
- **wa:** 1 bit
- **wd:** 1 bit

**ALU Outputs:**
- **rd1:** 1 bit
- **rd2:** 1 bit

**ImmSel:**
- **<14:12>**

**ImmSelect:**
- **<24:20>**
- **<19:15>**

**Addr Generator:**
- **PC:** 1 bit
- **Inst:** 7 bits

**Mem/Inst. Memory:**
- **addr:** 8 bits
- **inst:** 4 bits

**Adder:**
- **0x4 Add**

**Op2Sel:**
- **Reg / Imm**

**CLC:**
- **clke:** 1 bit

**GPRs:**
- **rd1:** 1 bit
- **rd2:** 1 bit

**Controller:**
- **<6:0>**

**Reg Write:**
- **<6:0>**

**ImmSelect:**
- **<24:20>**
- **<19:15>**

**Imm:**
- **<31:20>**

**Instruction:**
- **<31:20>**

**Datapath:**
- **clk:** 1 bit
- **addr:** 8 bits
- **inst:** 4 bits
- **RegWriteEn:** 1 bit
- **rd1:** 1 bit
- **rd2:** 1 bit
- **we:** 1 bit
- **rd:** 1 bit
- **rs1:** 3 bits
- **rs2:** 3 bits
- **wa:** 1 bit
- **wd:** 1 bit
- **rd1:** 1 bit
- **rd2:** 1 bit
- **imm:** 3 bits
- **func7:** 3 bits
- **rs1:** 3 bits
- **func3:** 2 bits
- **rd:** 1 bit
- **opcode:** 3 bits
- **imm12:** 12 bits

**Diagram Components:**
- **PC:** 2 bits
- **Addr Generator:** 8 bits
- **Inst. Memory:** 7 bits
- **Adder:** 1 bit
- **RegWriteEn:** 1 bit
- **Op2Sel:** 1 bit
- **GPRs:** 1 bit
- **ImmSelect:** 3 bits
- **Imm:** 3 bits
- **func7:** 3 bits
- **rs2:** 3 bits
- **rs1:** 3 bits
- **func3:** 2 bits
- **rd:** 1 bit
- **opcode:** 3 bits
- **imm12:** 12 bits
- **clk:** 1 bit

**Datapath Flow:**
- **addr:** 8 bits
- **inst:** 4 bits
- **RegWriteEn:** 1 bit
- **rd1:** 1 bit
- **rd2:** 1 bit
- **we:** 1 bit
- **rd:** 1 bit
- **rs1:** 3 bits
- **rs2:** 3 bits
- **wa:** 1 bit
- **wd:** 1 bit
- **rd1:** 1 bit
- **rd2:** 1 bit
- **imm:** 3 bits
- **func7:** 3 bits
- **rs1:** 3 bits
- **func3:** 2 bits
- **rd:** 1 bit
- **opcode:** 3 bits
- **imm12:** 12 bits
- **clk:** 1 bit
Load/Store Instructions

rs1 is the base register
rd is the destination of a Load, rs2 is the data source for a Store
RISC-V Conditional Branches

- Compare two integer registers for equality (BEQ/BNE) or signed magnitude (BLT/BGE) or unsigned magnitude (BLTU/BGEU)

- 12-bit immediate encodes branch target address as a signed offset from PC, in units of 16-bits (i.e., shift left by 1 then add to PC).
Conditional Branches (BEQ/BNE/BLT/BGE/BLTU/BGEU)

PCSel  
\[
\text{br} 
\]

0x4  
\[
\text{Add} 
\]

\[\text{clk} \]

\[\text{WBSel} \]

\[\text{MemWrite} \]

\[\text{Add} \]

\[\text{RegWrEn} \]

Inst.

\[\text{addr} \]

\[\text{Inst. Memory} \]

\[\text{clk} \]

\[\text{PC} \]

\[\text{addr} \]

\[\text{inst} \]

\[\text{OpCode} \]

\[\text{ImmSel} \]

\[\text{Op2Sel} \]

\[\text{ALU Control} \]

\[\text{ALU} \]

\[\text{Br Logic - Bcomp?} \]

\[\text{we} \]

\[\text{rs1} \]

\[\text{rs2} \]

\[\text{rd1} \]

\[\text{wa} \]

\[\text{wd} \]

\[\text{rd2} \]

\[\text{GPRs} \]

\[\text{Imm Select} \]

\[\text{ALU} \]

\[\text{rddata} \]

\[\text{Memory} \]

\[\text{wdata} \]

\[\text{clk} \]

\[\text{OP2} \]

\[\text{rd1} \]

\[\text{rd2} \]
RISC-V Unconditional Jumps

- 20-bit immediate encodes jump target address as a signed offset from PC, in units of 16-bits (i.e., shift left by 1 then add to PC). (+/- 1MiB)
- JAL is a subroutine call that also saves return address (PC+4) in register rd
RISC-V Register Indirect Jumps

- Jumps to target address given by adding 12-bit offset (not shifted by 1 bit) to register rs1
- The return address (PC+4) is written to rd (can be x0 if value not needed)
Full RISC-V 1-Stage Datapath

Note: for simplicity, the CSR File (control and status registers) and associated datapath is not shown.
Hardwired Control is pure Combinational Logic

- op code
- Equal?

**Combinational Logic**

- ImmSel
- Op2Sel
- FuncSel
- MemWrite
- WBSel
- WASel
- RegWriteEn
- PCSel
ALU Control & Immediate Extension

Inst<14:12> (Func3)

Inst<6:0> (Opcode)

+ 0?

FuncSel (Func, Op, +, 0?)

ImmSel (IType_{12}, SType_{12}, UType_{20})

Decode Map

ALUop
# Hardwired Control Table

<table>
<thead>
<tr>
<th>Opcode</th>
<th>ImmSel</th>
<th>Op2Sel</th>
<th>FuncSel</th>
<th>MemWr</th>
<th>RFWen</th>
<th>WBSel</th>
<th>WASel</th>
<th>PCSel</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>*</td>
<td>Reg</td>
<td>Func</td>
<td>no</td>
<td>yes</td>
<td>ALU</td>
<td>rd</td>
<td>pc+4</td>
</tr>
<tr>
<td>ALUi</td>
<td>IType(_{12})</td>
<td>Imm</td>
<td>Op</td>
<td>no</td>
<td>yes</td>
<td>ALU</td>
<td>rd</td>
<td>pc+4</td>
</tr>
<tr>
<td>LW</td>
<td>IType(_{12})</td>
<td>Imm</td>
<td>+</td>
<td>no</td>
<td>yes</td>
<td>Mem</td>
<td>rd</td>
<td>pc+4</td>
</tr>
<tr>
<td>SW</td>
<td>SType(_{12})</td>
<td>Imm</td>
<td>+</td>
<td>yes</td>
<td>no</td>
<td>*</td>
<td>*</td>
<td>pc+4</td>
</tr>
<tr>
<td>BEQ(_{\text{true}})</td>
<td>SBTyple(_{12})</td>
<td>*</td>
<td>*</td>
<td>no</td>
<td>no</td>
<td>*</td>
<td>*</td>
<td>br</td>
</tr>
<tr>
<td>BEQ(_{\text{false}})</td>
<td>SBTyple(_{12})</td>
<td>*</td>
<td>*</td>
<td>no</td>
<td>no</td>
<td>*</td>
<td>*</td>
<td>pc+4</td>
</tr>
<tr>
<td>J</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>no</td>
<td>no</td>
<td>*</td>
<td>*</td>
<td>jabs</td>
</tr>
<tr>
<td>JAL</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>no</td>
<td>yes</td>
<td>PC</td>
<td>X1</td>
<td>jabs</td>
</tr>
<tr>
<td>JALR</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>no</td>
<td>yes</td>
<td>PC</td>
<td>rd</td>
<td>rind</td>
</tr>
</tbody>
</table>

Op2Sel = Reg / Imm
WASel = rd / X1
WBSel = ALU / Mem / PC
PCSel = pc+4 / br / rind / jabs
Single-Cycle Hardwired Control

We will assume clock period is sufficiently long for all of the following steps to be “completed”:

1. Instruction fetch
2. Decode and register fetch
3. ALU operation
4. Data fetch if required
5. Register write-back setup time

=> $t_C > t_{IFetch} + t_{RFetch} + t_{ALU} + t_{DMem} + t_{RWB}$

At the rising edge of the following clock, the PC, register file and memory are updated.
Question of the Day

- Do you think a CISC or RISC single-cycle processor would be faster?
Summary

- Microcoding became less attractive as gap between RAM and ROM speeds reduced, and logic implemented in same technology as memory
- Complex instruction sets difficult to pipeline, so difficult to increase performance as gate count grew
- Iron Law explains architecture design space
  - Trade instruction/program, cycles/instruction, and time/cycle
- Load-Store RISC ISAs designed for efficient pipelined implementations
  - Very similar to vertical microcode
  - Inspired by earlier Cray machines (CDC 6600/7600)
- RISC-V ISA will be used in lectures, problems, and labs
  - Berkeley RISC chips: RISC-I, RISC-II, SOAR (RISC-III), SPUR (RISC-IV)
Acknowledgements

- These slides contain material developed and copyright by:
  - Arvind (MIT)
  - Krste Asanovic (MIT/UCB)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)
  - David Patterson (UCB)

- MIT material derived from course 6.823
- UCB material derived from course CS252