Last Time in Lecture 2

• ISA is the hardware/software interface
  – Defines set of programmer visible state
  – Defines instruction format (bit encoding) and instruction semantics
  – Examples: MIPS, x86, IBM 360, JVM

• Many possible implementations of one ISA
  – 360 implementations: model 30 (c. 1964), z10 (c. 2008)
  – x86 implementations: 8086 (c. 1978), 80186, 286, 386, 486, Pentium, Pentium Pro, Pentium-4 (c. 2000), Core 2 Duo, Nehalem, AMD Athlon, Transmeta Crusoe, SoftPC
  – MIPS implementations: R2000, R4000, R10000, R18K, ...
  – JVM: HotSpot, PicoJava, ARM Jazelle, ...
Last Time in Lecture 2

• When microcode appeared, different technologies for:
  – Logic -> Vacuum Tubes
  – Main Memory -> Magnetic cores
  – Read-Only Memory -> Diode matrix, punched metal cards,+++ 
• Logic was expensive, and ROM much faster than RAM 
• Microcoding was a straightforward methodical way to implement machines with low logic gate count
• Microcode made it easy to add complex instructions
“Iron Law” of Processor Performance

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

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

<table>
<thead>
<tr>
<th></th>
<th>Inst 1</th>
<th>Inst 2</th>
<th>Inst 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cycles</td>
<td>7</td>
<td>5</td>
<td>10</td>
</tr>
</tbody>
</table>

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
When to add a new complex instruction?

• Does it improve performance?
• How much does it cost?
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
• Single 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 disk drives were originally invented by IBM as a way of shipping IBM 360 microcode patches to customers!
DRAM in the Seventies

Dramatic progress in MOSFET 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 size and speed through 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

- Advanced features for personal computers.
- High-resolution color graphics.
- Ten user-defined function keys.
- Diagnostics software and features.
- Back-up and protection software.
- Color printers and terminals.
- IBM personal computer monitors.
- IBM personal computer keyboards.
- IBM personal computer printers.
- IBM personal computer terminals.
- IBM personal computer disk drives.
- IBM personal computer tape drives.
- IBM personal computer hard drives.
- IBM personal computer peripheral equipment.

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.

The IBM Personal Computer and me.

For the IBM Personal Computer dealer nearest you, call (800) 447-0700. In Illinois, (800) 122-0600.

[Personal Computing Ad, 11/81]
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

• 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
Nanocoding

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

\[ ALU_0 \ A \leftarrow \text{Reg[rs]} \]
\[ \ldots \]
\[ ALU_i \ A \leftarrow \text{Reg[rs]} \]
\[ \ldots \]

- 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
  – Can change contents of fast instruction memory 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…
CS152 Administrivia

• PS1 available later today
• Lab 1 available before section on Thursday
  – Scott Beamer standing in for Andrew in Section on Thursday
    2-3:30pm, in 320 Soda
“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 upon the ISA and the microarchitecture
- Time per cycle depends upon the microarchitecture and the 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>
Hardware Elements

• Combinational circuits
  – Mux, Decoder, ALU, ...

  ![Mux Diagram]

  ![Decoder Diagram]

  ![ALU Diagram]

• Synchronous state elements
  – Flipflop, Register, Register file, SRAM, DRAM

  ![Flipflop Diagram]

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

- Reads are combinational
Register File Implementation

- Register files with a large number of ports are difficult to design
  - Almost all MIPS instructions have exactly 2 register source operands
  - *Intel’s Itanium, GPR File has 128 registers with 8 read ports and 4 write ports!!!*
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
  \[ \Rightarrow \text{the write address and data must be stable at the clock edge} \]

Later in the course we will present a more realistic model of memory
Implementing MIPS:

Single-cycle per instruction datapath & control logic
(Should be review of CS61C)
The MIPS ISA

Processor State
- 32 32-bit GPRs, R0 always contains a 0
- 32 single precision FPRs, may also be viewed as 16 double precision FPRs
- FP status register, used for FP compares & exceptions
- PC, the program counter
- some other special registers

Data types
- 8-bit byte, 16-bit half word
- 32-bit word for integers
- 32-bit word for single precision floating point
- 64-bit word for double precision floating point

Load/Store style instruction set
- data addressing modes- immediate & indexed
- branch addressing modes- PC relative & register indirect
- Byte addressable memory- big endian mode

All instructions are 32 bits
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

and the computation of the address of the next instruction
Datapath: Reg-Reg ALU Instructions

RegWrite Timing?

\[
\text{rd} \leftarrow (\text{rs}) \text{ func} (\text{rt})
\]
Datapath: Reg-Imm ALU Instructions

```
6 5 5 16
0x4 Add

31 26 25 21 20 16 15 0

rt ← (rs) op immediate
```
Conflicts in Merging Datapath

Introduce muxes

rd ← (rs) func (rt)
rt ← (rs) op immediate
Datapath for ALU Instructions

\[ (rs) \text{ op } \text{immediate} \]

\[ (rs) \text{ func } (rt) \]

\[ rt \Leftarrow (rs) \text{ op } \text{immediate} \]
Datapath for Memory Instructions

Should program and data memory be separate?

*Harvard style: separate* (Aiken and Mark 1 influence)
- read-only program memory
- read/write data memory

- Note:
  Somehow there must be a way to load the program memory

*Princeton style: the same* (von Neumann’s influence)
- single read/write memory for program and data

- Note:
  A Load or Store instruction requires accessing the memory more than once during its execution
Load/Store Instructions: *Harvard Datapath*

- **opcode**: 5 bits
- **rs**: 5 bits
- **rt**: 5 bits
- **displacement**: 16 bits

- **rs** is the base register
- **rt** is the destination of a Load or the source for a Store

Addressing mode: (rs) + displacement
MIPS Control Instructions

Conditional (on GPR) PC-relative branch

<table>
<thead>
<tr>
<th>opcode</th>
<th>rs</th>
<th>offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

BEQZ, BNEZ

Unconditional register-indirect jumps

<table>
<thead>
<tr>
<th>opcode</th>
<th>rs</th>
<th>offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

JR, JALR

Unconditional absolute jumps

<table>
<thead>
<tr>
<th>opcode</th>
<th>target</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>26</td>
</tr>
</tbody>
</table>

J, JAL

- PC-relative branches add offset×4 to PC+4 to calculate the target address (offset is in words): ±128 KB range
- Absolute jumps append target×4 to PC<31:28> to calculate the target address: 256 MB range
- jump-&-link stores PC+4 into the link register (R31)
- All Control Transfers are delayed by 1 instruction

we will worry about the branch delay slot later
Conditional Branches (BEQZ, BNEZ)

January 26, 2010
CS152, Spring 2010
Register-Indirect Jumps (JR)

PCSrc → br → rind → pc+4 → Add → 0x4

Register Write (RegWrite)

MemWrite → WBSrc

Instruction Memory (Inst. Memory)

GPRs

ALU

Control

Zero (zero?)

Data Memory (Data Mem)

We (we)

Ext (Ext)

Inst. (inst)

Addr (addr)

PC (PC)
Register-Indirect Jump-&-Link (JALR)
Absolute Jumps (J, JAL)
Harvard-Style Datapath for MIPS
Hardwired Control is pure Combinational Logic

op code

zero?

combinational logic

ExtSel
BSrc
OpSel
MemWrite
WBSrc
RegDst
RegWrite
PCSrc
ALU Control & Immediate Extension

Inst<31:26> (Opcode)
Inst<5:0> (Func)

ALUop

OpSel (Func, Op, +, 0?)

ExtSel (sExt_{16}, uExt_{16}, High_{16})

Decode Map
## Hardwired Control Table

<table>
<thead>
<tr>
<th>Opcode</th>
<th>ExtSel</th>
<th>BSrc</th>
<th>OpSel</th>
<th>MemW</th>
<th>RegW</th>
<th>WBSrc</th>
<th>RegDst</th>
<th>PCSrc</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>sExt\textsubscript{16}</td>
<td>Imm</td>
<td>Op</td>
<td>no</td>
<td>yes</td>
<td>ALU</td>
<td>rt</td>
<td>pc+4</td>
</tr>
<tr>
<td>ALUiu</td>
<td>uExt\textsubscript{16}</td>
<td>Imm</td>
<td>Op</td>
<td>no</td>
<td>yes</td>
<td>ALU</td>
<td>rt</td>
<td>pc+4</td>
</tr>
<tr>
<td>LW</td>
<td>sExt\textsubscript{16}</td>
<td>Imm</td>
<td>+</td>
<td>no</td>
<td>yes</td>
<td>Mem</td>
<td>rt</td>
<td>pc+4</td>
</tr>
<tr>
<td>SW</td>
<td>sExt\textsubscript{16}</td>
<td>Imm</td>
<td>+</td>
<td>yes</td>
<td>no</td>
<td>*</td>
<td>*</td>
<td>pc+4</td>
</tr>
<tr>
<td>BEQZ\textsubscript{z=0}</td>
<td>sExt\textsubscript{16}</td>
<td>*</td>
<td>0?</td>
<td>no</td>
<td>no</td>
<td>*</td>
<td>*</td>
<td>br</td>
</tr>
<tr>
<td>BEQZ\textsubscript{z=1}</td>
<td>sExt\textsubscript{16}</td>
<td>*</td>
<td>0?</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>R31</td>
<td>jabs</td>
</tr>
<tr>
<td>JR</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>no</td>
<td>no</td>
<td>*</td>
<td>*</td>
<td>rind</td>
</tr>
<tr>
<td>JALR</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>no</td>
<td>yes</td>
<td>PC</td>
<td>R31</td>
<td>rind</td>
</tr>
</tbody>
</table>

BSrc = Reg / Imm  
WBSrc = ALU / Mem / PC  
RegDst = rt / rd / R31  
PCSrc = pc+4 / br / rind / jabs
Single-Cycle Hardwired Control: 
Harvard architecture

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, the register file and the memory are updated
An Ideal Pipeline

• All objects go through the same stages
• No sharing of resources between any two stages
• Propagation delay through all pipeline stages is equal
• The scheduling of an object entering the pipeline is not affected by the objects in other stages

These conditions generally hold for industrial assembly lines. But can an instruction pipeline satisfy the last condition?
Summary

- Microcoding became less attractive as gap between RAM and ROM speeds reduced
- 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
- MIPS ISA will be used in class and problems, SPARC in lab (two very similar ISAs)
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