CS61CL Machine Structures

Lec 5 – Instruction Set Architecture

David Culler
Electrical Engineering and Computer Sciences
University of California, Berkeley
What is “Computer Architecture”?

- Coordination of many *levels of abstraction*
- Under a rapidly *changing set of forces*
- Design, Measurement, *and* Evaluation

**Applications**

- Operating System
- Compiler
- Firmware
- I/O system
- Datapath & Control
- Digital Design
- Circuit Design
- Layout & fab

**Semiconductor Materials**

- Die photo
- App photo
Forces on Computer Architecture

Technology

Programming Languages

Applications

Computer Architecture

Operating Systems

History

(A = F / M)
The Instruction Set: a Critical Interface

- **Properties of a good abstraction**
  - Lasts through many generations (portability)
  - Used in many different ways (generality)
  - Provides *convenient* functionality to higher levels
  - Permits an *efficient* implementation at lower levels
Instruction Set Architecture

... the attributes of a [computing] system as seen by the programmer, i.e. the conceptual structure and functional behavior, as distinct from the organization of the data flows and controls the logic design, and the physical implementation.

– Amdahl, Blaaw, and Brooks, 1964

-- Organization of Programmable Storage
-- Data Types & Data Structures: Encodings & Representations
-- Instruction Formats
-- Instruction (or Operation Code) Set
-- Modes of Addressing and Accessing Data Items and Instructions
-- Exceptional Conditions
Computer Organization

- Capabilities & Performance
  Characteristics of Principal Functional Units
    - (e.g., Registers, ALU, Shifters, Logic Units, ...)
- Ways in which these components are interconnected
- Information flows between components
- Logic and means by which such information flow is controlled.
- Choreography of FUs to realize the ISA
- Register Transfer Level (RTL) Description

Logic Designer's View

ISA Level

FUs & Interconnect
Fundamental Execution Cycle

- **Instruction Fetch**
  - Obtain instruction from program storage
- **Instruction Decode**
  - Determine required actions and instruction size
- **Operand Fetch**
  - Locate and obtain operand data
- **Execute**
  - Compute result value or status
- **Result Store**
  - Deposit results in storage for later use
- **Next Instruction**
  - Determine successor instruction

Diagram:
- **Processor**
  - regs
  - F.U.s
- **Memory**
  - program
  - Data
- von Neuman bottleneck
Elements of an ISA

• Set of machine-recognized data types
  – bytes, words, integers, floating point, strings, . . .

• Operations performed on those data types
  – Add, sub, mul, div, xor, move, ....

• Programmable storage
  – regs, PC, memory

• Methods of identifying and obtaining data referenced by instructions (addressing modes)
  – Literal, reg., absolute, relative, reg + offset, ...

• Format (encoding) of the instructions
  – Op code, operand fields, ...
Administrative Issues

- HW3 due before midnight
- HW4 will go out before morning, due next Wednesday
- Project 1 due midnight Friday 10/2
- Midterm 1 the following wed
  - alternate on Monday 6-7 pm
  - need to email instructor in advance
**Example: MIPS R3000**

<table>
<thead>
<tr>
<th>r0</th>
<th>r1</th>
<th>r31</th>
<th>PC</th>
<th>lo</th>
<th>hi</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Programmable storage**
- \(2^{32}\) x bytes
- 31 x 32-bit GPRs (R0=0)
- 32 x 32-bit FP regs (paired DP)
- HI, LO, PC

**Data types ?**
**Format ?**
**Addressing Modes ?**

**Arithmetic logical**
- Add, AddU, Sub, SubU, And, Or, Xor, Nor, SLT, SLTU, AddI, AddIU, SLTI, SLTIU, AndI, OrI, XorI, *LUI*
- SLL, SRL, SRA, SLLV, SRLV, SRAV

**Memory Access**
- LB, LBU, LH, LHU, LW, LWL, LWR
- SB, SH, SW, SWL, SWR

**Control**
- J, JAL, JR, JALR
- BEq, BNE, BLEZ, BGTZ, BLTZ, BGEZ, BLTZAL, BGEZAL

*32-bit instructions on word boundary*
Address Space

PC
regs
sp

reserved
0000000:
0040000:
code
<= instructions
1000000:
static data
<= externs
1008000:
heap
<= malloc
7FFFFFFF:
stack
<= Local variables
FFFFFFFFFFF:
stack
<= OS, etc.
FFFFFFFFFFF:
unused
MIPS Instruction Format

<table>
<thead>
<tr>
<th>R: Reg-Reg</th>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>shamt</th>
<th>funct</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>I: Reg-Imed</th>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>16</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>J: Jump</th>
<th>op</th>
<th>immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>6</td>
<td>26</td>
</tr>
</tbody>
</table>

- **Reg-Reg instructions (op == 0)**
  - add, sub, and, or, nor, xor, slt
  - sll, srl, sra

- **Reg-Immed (op != 0)**
  - addi, andi, ori, xori, lui
  - addiu, slti, sltiu
  - lw, lh, lhu, lb, lbu
  - sw, sh, sb

\[ R[rd] := R[rs] \text{ funct } R[rt]; \text{ pc:=pc+4} \]
\[ R[rd] := R[rt] \text{ shift shamt} \]
\[ R[rt] := R[rs] \text{ op Im16} \]
\[ R[rt] := \text{Mem}[R[rs] + \text{signEx}(\text{Im16})] \]
\[ \text{Mem}[R[rs] + \text{signEx}(\text{Im16})] := R[rt] \]
Addressing Modes

- How is the operand located?
  - Effective Address Calculation

- Immediate

- Register Direct

- Absolute

- Register Indirect

- Base + offset
Data Structure Access to Addressing Mode

- Pointer dereference: \(*p*\)
- Struct Field: \(S.\text{foo}\)
- Array Element \(A[i]\)
- Pointer deref to field \(p->\text{foo}\)
- Pointer to array \((*A)[i]\)
- Array of pointers \(*(A[i])\)
- Pointer to pointer \(**P**\)
- Array of ptrs to struct \(A[i]->\text{foo}\)
Computer Number Systems

• We all take positional notation for granted
  – \( D_{k-1} D_{k-2} \ldots D_0 \) represents \( D_{k-1}B^{k-1} + D_{k-2}B^{k-2} + \ldots + D_0 B^0 \)
    where \( B \in \{ 0, \ldots, B-1 \} \)

• We all understand how to compare, add, subtract these numbers
  – Add each position, write down the position bit and possibly carry to the next position

• Computers represent finite number systems
  – Generally radix 2
  – \( B_{k-1} B_{k-2} \ldots B_0 \) represents \( B_{k-1}2^{k-1} + B_{k-2}2^{k-2} + \ldots + B_0 2^0 \)
    where \( B \in \{ 0,1 \} \)
Unsigned Numbers - Addition

Example: $3 + 2 = 5$

Unsigned binary addition
Is just addition, base 2
Add the bits in each position and carry

```
  0 0 1 1
+   0 0 1 0
  1 0 1 1
```
Twos Complement number wheel

\[ B_{k-1} B_{k-2} \ldots B_0 \text{ represents } -B_{k-1}2^{k-1} + B_{k-2}2^{k-2} + \ldots + B_0 2^0 \]

Easy to determine sign, Only one representation for 0
Addition and subtraction just as in unsigned case
Simple comparison: \( A < B \) iff \( A - B < 0 \)
One more negative number than positive number
Sign Extension

Positive Number

-0*2^7 + B_6*2^6 + B_5*2^5 + B_4*2^4 + B_3*2^3 + B_2*2^2 + B_1*2^1 + B_0*2^0

Negative Number

-2^7 + B_6*2^6 + B_5*2^5 + B_4*2^4 + B_3*2^3 + B_2*2^2 + B_1*2^1 + B_0*2^0

-2^{15} + \ldots + 2^7 + B_6*2^6 + B_5*2^5 + B_4*2^4 + B_3*2^3 + B_2*2^2 + B_1*2^1 + B_0*2^0
MIPS Instruction Format

R: Reg-Reg

\[
\begin{array}{cccccc}
\text{op} & \text{rs} & \text{rt} & \text{rd} & \text{shamt} & \text{funct} \\
6 & 5 & 5 & 5 & 5 & 6 \\
\end{array}
\]

I: Reg-Immed

\[
\begin{array}{cccc}
\text{op} & \text{rs} & \text{rt} & \text{immediate} \\
6 & 5 & 5 & 16 \\
\end{array}
\]

J: Jump

\[
\begin{array}{cc}
\text{op} & \text{Addr} \\
6 & 26 \\
\end{array}
\]

- Reg-Reg instructions (op == 0)
- Reg-Immed (op != 0)
- Jumps
  - j
    \[
    \text{PC} := \text{PC}_{31..28} \|| \text{addr} \|| 00
    \]
  - jal
    \[
    \text{PC} := \text{PC}_{31..28} \|| \text{addr} \|| 00; \ R[31] := \text{PC} + 4
    \]
  - jr
    \[
    \text{PC} := \text{R}[\text{rs}]
    \]
MIPS Instruction Format

- Reg-Reg instructions (op == 0)
- Reg-Immed (op != 0)
- Jumps
- Branches
  - BEQ, BNE
    \[ PC := (R[rs] \neq R[rt]) ? PC + \text{signEx(im16)} : PC+4 \]
  - BLT, BGT, BLE, BGTE are pseudo ops
  - Move and LI are pseudo ops too
Evolution of Instruction Sets

- **Single Accumulator** (EDSAC 1950)
- Accumulator + Index Registers (Manchester Mark I, IBM 700 series 1953)

  - Separation of Programming Model from Implementation

  - High-level Language Based (Stack) (B5000 1963)
  - Concept of a Family (IBM 360 1964)

- General Purpose Register Machines

  - Complex Instruction Sets (Vax, Intel 432 1977-80)
  - Load/Store Architecture (CDC 6600, Cray 1 1963-76)

  - RISC

- iX86? (MIPS, Sparc, HP-PA, IBM RS6000, 1987)
Dramatic Technology Advance

• Prehistory: Generations
  – 1\textsuperscript{st} Tubes
  – 2\textsuperscript{nd} Transistors
  – 3\textsuperscript{rd} Integrated Circuits
  – 4\textsuperscript{th} VLSI....

• Discrete advances in each generation
  – Faster, smaller, more reliable, easier to utilize

• Modern computing: Moore’s Law
  – Continuous advance, fairly homogeneous technology
Moore’s Law

- “Cramming More Components onto Integrated Circuits”
  - Gordon Moore, Electronics, 1965
- # on transistors on cost-effective integrated circuit double every 18 months
Technology Trends: Microprocessor Capacity

Moore's Law

Itanium II: 241 million
Pentium 4: 55 million
Alpha 21264: 15 million
Pentium Pro: 5.5 million
PowerPC 620: 6.9 million
Alpha 21164: 9.3 million
Sparc Ultra: 5.2 million

CMOS improvements:
• Die size: 2X every 3 yrs
• Line width: halve / 7 yrs
Technology Trends

- Clock Rate: ~30% per year
- Transistor Density: ~35%
- Chip Area: ~15%
- Transistors per chip: ~55%
- Total Performance Capability: ~100%
- by the time you graduate...
  - 3x clock rate (>10 GHz)
  - 10x transistor count (100 Billion transistors)
  - 30x raw capability

- plus 16x dram density,
- 32x disk density (60% per year)
- Network bandwidth, …
Performance Trends

![Graph showing performance trends over time with different computer categories: Supercomputers, Mainframes, Minicomputers, and Microprocessors. MIPS R3000 is indicated as a point of interest.]
Processor Performance
(1.35X before, 1.55X now)
**Definition: Performance**

- Performance is in units of things per sec
  - bigger is better
- If we are primarily concerned with response time

\[
\text{performance}(x) = \frac{1}{\text{execution\_time}(x)}
\]

"**X is n times faster than Y**" means

\[
\begin{align*}
n &= \frac{\text{Performance}(X)}{\text{Performance}(Y)} \\
&= \frac{\text{Execution\_time}(Y)}{\text{Execution\_time}(Y)}
\end{align*}
\]
Metrics of Performance

- (millions) of Instructions per second: MIPS
- (millions) of (FP) operations per second: MFLOP/s
- Megabytes per second
- Cycles per second (clock rate)
- Answers per day/month

- Application
- Programming Language
- Compiler
- ISA
- Datapath
- Control
- Function Units
- Transistors
- Wires
- Pins