Instruction Set Architecture

Hung-Wei Tseng
Announcement

• Reading quizzes due this Thursday before class
• CSE141/CSE141L enrollment issues
Recap: computer architecture

• The art or science of building computers
  • Processors and memories are everywhere in computers
  • We care about performance, power, cost, and etc. when building computers
• Moore’s law still allows us to shrink the die size, but power consumption limits the single chip/core performance
• We are now in the era of multicore processors, but parallelizing a program to utilize multithreaded processors is not easy
Setup your i-clicker

- Register your i-clicker
  - Read here: https://csemoodle.ucsd.edu/mod/resource/view.php?id=12303

- Set your channel to “CA”
  - Press on/off button for 2 seconds
  - Press C and then press A
Outline

• How we talk to computers
• How to design an ISA (instruction set architecture)
• MIPS ISA
How we talk to computers
Which of the following is the language that processors speak?

- **A**
  - `lw $15, 0($2)`
  - `lw $16, 4($2)`
  - `sw $16, 0($2)`
  - `sw $15, 4($2)`

- **B**
  - Machine language!
  - One-to-one mapping

- **C**
  - Assembly language
  - `ALUOP[0:3] <= InstReg[9:11] & MASK`

- **D**
  - Control signal spec
  - `temp = v[k];`
  - `v[k] = v[k+1];`
  - `v[k+1] = temp;`

- **E**
  - High-level language
In the very old days...

- Physical configuration specified the computation a computer performed

The difference engine

ENIAC
The stored program computer

- The program is data
  - a series of bits
    - these bits are “instructions”!
  - lives in memory
- Program counter
  - points to the current instruction
  - processor “fetches” instructions from where PC points.
- advances/changes after instruction execution

Processor

PC

instruction memory

```
120007a30: 0f00bb27 ldah gp,15(t12)
120007a34: 509cbd23 lda gp,-25520(gp)
120007a38: 00005d24 ldah t1,0(gp)
120007a3c: 0000bd24 ldah t4,0(gp)
120007a40: 2ca422a0 ldl t0,-23508(t1)
120007a44: 130020e4 beq t0,120007a94
120007a48: 00003d24 ldah t0,0(gp)
120007a4c: 2ca4e2b3 stl zero,-23508(t1)
120007a50: 0004ff47 clr v0
120007a54: 28a4e5b3 stl zero,-23512(t4)
120007a58: 20a421a4 ldq t0,-23520(t0)
120007a5c: 0e0020e4 beq t0,120007a98
120007a60: 0204e147 mov t0,t1
120007a64: 0304ff47 clr t2
120007a68: 0500e0c3 br 120007a80
```
Which of the following is the language that processors speak?

- A: Assembly language
- B: Machine language!
- C: Instruction Set Architecture
- D: One-to-one mapping


\[
\begin{align*}
\text{lw} & \quad 00110001100100000000000000000000 \\
\text{lw} & \quad 00110011110010000000000000000100 \\
\text{sw} & \quad 00110001100100000000000000000000 \\
\text{sw} & \quad 00110001100100000000000000000100
\end{align*}
\]
Instruction Set Architecture (ISA)

- The contract between the hardware and software
- Defines the set of operations that a computer/processor can execute
- Programs are combinations of these instructions
  - Abstraction to programmers/compilers
- The hardware implements these instructions in any way it choose.
  - Directly in hardware circuit
  - Software virtual machine
  - Simulator
  - Trained monkey with pen and paper
How do you know about ISA?

- Which of the following is generally true about ISA?
  A. Many models of processors can support one ISA
  B. An ISA is unique to one model of processor
  C. Every processor can support multiple ISAs
  D. Each processor manufacturer has its own ISA
  E. None of the above
Example ISAs

- x86: intel Xeon, intel Core i7/i5/i3, intel atom, AMD Athlon/Opteron, AMD FX, AMD A-series
- MIPS: Sony/Toshiba Emotion Engine, MIPS R-4000(PSP)
- ARM: Apple A-Series, Qualcomm Snapdragon, TI OMAP, nVidia Tegra
- DEC Alpha: 21064, 21164, 21264
- PowerPC: Motorola PowerPC G4, Power 6
- IA-64: Itanium
- SPARC and many more ...
From C to Assembly

C program → compiler → Assembly

Assembly → assembler → Object

Object → linker → Executable

Executable → loader → Memory
ISA design
What ISA includes?

- Instructions: what programmers want processors to do?
  - Math: add, subtract, multiply, divide, bitwise operations
  - Control: if, jump, function call
  - Data access: load and store
- Architectural states: the current execution result of a program
  - Registers: a few named data storage that instructions can work on
  - Memory: a much larger data storage array that is available for storing data
  - PC: the number/address of the current instruction
What should an instruction look like?

- **Operations**
  - What operations?
  - How many operations?

- **Operands**
  - How many operand?
  - What type of operands?
    - Memory/register/label/number (immediate value)

- **Format**
  - Length
  - Formats?
How to encode an instruction into binary

- Assuming that the ISA has **36 operations and 32 registers**, how many of the following machine code/instruction formats are valid?

<table>
<thead>
<tr>
<th>Format</th>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>6</td>
<td>4 4 4 4</td>
</tr>
<tr>
<td>II</td>
<td>5</td>
<td>5 5 5 5</td>
</tr>
<tr>
<td>III</td>
<td>6</td>
<td>5 5 5</td>
</tr>
<tr>
<td>IV</td>
<td>7</td>
<td>5 5 5</td>
</tr>
<tr>
<td>V</td>
<td>6</td>
<td>6 6 6 6</td>
</tr>
</tbody>
</table>

A. 1
B. 2
C. 3
D. 4
E. 5
We will study two ISAs

- **MIPS**
  - Simple, elegant, easy to implement
    - That’s why we want to implement it in CSE141L
  - Designed with many-year ISA design experience
  - The prototype of a lot of modern ISAs
    - MIPS itself is not widely used, though

- **x86**
  - Ugly, messy, inelegant, hard to implement, ...
  - Designed for 1970s technology
  - The dominant ISA in modern computer systems

You should know how to write MIPS code after this class
You should know how to read x86 code after this class
MIPS
MIPS ISA

• All instructions are 32 bits
• 32 32-bit registers
  • All registers are the same
  • $zero is always 0
• 50 opcodes
• 3 instruction formats
  • R-type: all operands are registers
  • I-type: one of the operands is an immediate value
  • J-type: non-conditional, non-relative branches

<table>
<thead>
<tr>
<th>name</th>
<th>number</th>
<th>usage</th>
<th>saved?</th>
</tr>
</thead>
<tbody>
<tr>
<td>$zero</td>
<td>0</td>
<td>zero</td>
<td>N/A</td>
</tr>
<tr>
<td>$at</td>
<td>1</td>
<td>assembler temporary</td>
<td>no</td>
</tr>
<tr>
<td>$v0-$v1</td>
<td>2-3</td>
<td>return value</td>
<td>no</td>
</tr>
<tr>
<td>$a0-$a3</td>
<td>4-7</td>
<td>arguments</td>
<td>no</td>
</tr>
<tr>
<td>$t0-$t7</td>
<td>8-15</td>
<td>temporaries</td>
<td>no</td>
</tr>
<tr>
<td>$s0-$s7</td>
<td>16-23</td>
<td>saved</td>
<td>yes</td>
</tr>
<tr>
<td>$t8-$t9</td>
<td>24-25</td>
<td>temporaries</td>
<td>no</td>
</tr>
<tr>
<td>$gp</td>
<td>28</td>
<td>global pointer</td>
<td>yes</td>
</tr>
<tr>
<td>$sp</td>
<td>29</td>
<td>stack pointer</td>
<td>yes</td>
</tr>
<tr>
<td>$fp</td>
<td>30</td>
<td>frame pointer</td>
<td>yes</td>
</tr>
<tr>
<td>$ra</td>
<td>31</td>
<td>return address</td>
<td>yes</td>
</tr>
</tbody>
</table>
MIPS ISA (cont.)

- Only load and store instructions can access memory
- Memory is “byte addressable”
  - Most modern ISAs are byte addressable, too
  - byte, half words, words are aligned

<table>
<thead>
<tr>
<th>Address</th>
<th>Data</th>
<th>Address</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000</td>
<td>0xAA</td>
<td>0x0000</td>
<td>0xAA15</td>
</tr>
<tr>
<td>0x0001</td>
<td>0x15</td>
<td>0x0002</td>
<td>0x13FF</td>
</tr>
<tr>
<td>0x0002</td>
<td>0x13</td>
<td>0x0004</td>
<td>.</td>
</tr>
<tr>
<td>0x0003</td>
<td>0xFF</td>
<td>0x0004</td>
<td>.</td>
</tr>
<tr>
<td>0x0004</td>
<td>0x76</td>
<td>0x0005</td>
<td>.</td>
</tr>
<tr>
<td>...</td>
<td>.</td>
<td>...</td>
<td>.</td>
</tr>
<tr>
<td>0xFFFFE</td>
<td>.</td>
<td>0xFFFFE</td>
<td>.</td>
</tr>
<tr>
<td>0xFFFFF</td>
<td>.</td>
<td>0xFFFFF</td>
<td>.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Address</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000</td>
<td>0xAA1513FF</td>
</tr>
<tr>
<td>0x0004</td>
<td>.</td>
</tr>
<tr>
<td>0x0008</td>
<td>.</td>
</tr>
<tr>
<td>0x000C</td>
<td>.</td>
</tr>
<tr>
<td>...</td>
<td>.</td>
</tr>
<tr>
<td>0xFFFFC</td>
<td>.</td>
</tr>
</tbody>
</table>

Byte addresses

Half Word Addr

Word Addresses
R-type

- op $rd, $rs, $rt
  - 3 regs.: add, addu, and, nor, or, sltu, sub, subu
  - 2 regs.: sll, srl
  - 1 reg.: jr
- 1 arithmetic operation, 1 I-memory access
- Example:
    - opcode = 0x0, funct = 0x20
  - sll $t0, $t1, 8: $R[8] = $R[9] << 8
    - opcode = 0x0, shamt = 0x8, funct = 0x0
I-type

- `op $rt, $rs, immediate`
- `addi, addiu, andi, beq, bne, ori, slti, sltiu`
- `op $rt, offset($rs)`
- `lw, lbu, lhu, ll, lui, sw, sb, sc, sh`
- 1 arithmetic op, 1 I-memory and 1 D-memory access
- Example:
  - `lw $s0, 4($s2):`
  - `lw $s0, 0($s2)`
  - `add $s2, $s2, $s1`
  - `lw $s0, 0($s2)`

**Diagram:**

<table>
<thead>
<tr>
<th>opcode</th>
<th>rs</th>
<th>rt</th>
<th>immediate / offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>16 bits</td>
</tr>
</tbody>
</table>

**Comments:**
- only two addressing modes
- Example: `lw $s0, 4($s2)`, `lw $s0, 0($s2)`, `add $s2, $s2, $s1`
I-type (cont.)

- op $rt, $rs, immediate
  - addi, addiu, andi, beq, bne, ori, slti, sltiu
- op $rt, offset($rs)
  - lw, lbu, lhu, ll, lui, sw, sb, sc, sh
- 1 arithmetic op, 1 I-memory and 1 D-memory access
- Example:
  - beq $t0, $t1, -40
    
    if (R[8] == R[9]) PC = PC + 4 + 4*(-40)
J-type

- op immediate
  - j, jal
- 1 instruction memory access, 1 arithmetic op

Example:
- jal quicksort:
  R[31] = PC + 4
  PC = quicksort