Today

- Finish up the ISA design example
- Project description
- x86 assembly overview?

- Reminder: Lab hours this evening rescheduled to tomorrow. Also, extra office hours tomorrow: 11:30-1
Our ISA

- 35 bits
- 6 bit opcode
- 7 bit registers
- 3 operands (21 bits total)
- \text{add } r1, r2, r3 \Rightarrow R[r1] = R[r2] + R[r3]
- One instruction format \{opcode, r1, r2, r3\}
- Operations
  - int \{add, sub, mult, div, mod\}, fp \{add, sub, mult, divide\}, |, &, ^, ~, cmp\{lt, le, gt, ge, ne, eq\}, load, store, branch, jump
<table>
<thead>
<tr>
<th>Code</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>a = b + c;</code></td>
<td></td>
</tr>
<tr>
<td><code>a = b + c + d +... +z;</code></td>
<td></td>
</tr>
<tr>
<td><code>a = foo(b);</code></td>
<td></td>
</tr>
<tr>
<td><code>if (a) b = c;</code></td>
<td>Simplicity favors regularity</td>
</tr>
<tr>
<td><code>if (a == 4) b = c;</code></td>
<td>Smaller is faster</td>
</tr>
<tr>
<td><code>while (a != 0) a--;</code></td>
<td>Make the common case fast</td>
</tr>
<tr>
<td><code>a = 0xEADBEEF;</code></td>
<td>Good design demands good compromises</td>
</tr>
<tr>
<td><code>a = foo[4];</code></td>
<td></td>
</tr>
<tr>
<td><code>foo[4] = a;</code></td>
<td></td>
</tr>
<tr>
<td><code>a = foo.bar;</code></td>
<td></td>
</tr>
<tr>
<td><code>a = a + b + c + d +... +z;</code></td>
<td></td>
</tr>
<tr>
<td><code>a = foo(b);</code></td>
<td></td>
</tr>
</tbody>
</table>
Accessing Memory

• Load and store instructions should be the only instructions that access memory

• Loads in MIPS
  • lw r1, offset(r2) -> R[rt] = mem[R[rs] + imm]

• Stores in MIPS
  • sw r1, offset(r2) -> mem[R[rs] + imm] = R[rt]

• Does it makes sense that rt is an input to sw and an output of lw?
Large Constants

- Some constants are too big for the immediate field.
  - example: Create 0xDEADBEEF -- 32 bit values
- MIPS -- 16 bit immediate
  - add r1, zero, 0xDEAD
  - sll r1, r1, 16
  - ori r1, r1, 0xBEEF
- Alternative:
  - Assembly: LoadConst r1, 0xDEADBEEF
  - RTL: \( R[r1] = \text{mem}[\text{PC}+4]; \text{PC} = \text{PC} + 8. \)
Uniformity in MIPS

• 3 instruction formats: I, R, and J.
  • R-type: Register-register Arithmetic
  • I-type: immediate arithmetic; loads/stores
  • J-type: Non-conditional, non-relative branches
• opcodes are always in the same place
• rs and rt are always in the same place
• The immediate is always in the same place

• Similar amounts of work per instruction
  • 1 read from instruction memory
  • <= 1 arithmetic operations
  • <= 2 register reads
  • <= 1 register write
  • <= 1 data store/load

• Fixed instruction length
• Relatively large register file: 32
• Reasonably large immediate field: 16 bits
• Wise use of opcode space
  • 6 bits of opcode
  • I-type gets another 6 bits of “function”

R[r1] = mem[PC+4] breaks this uniformity
Supporting Function Calls

- Functions are an essential feature of modern languages
- What does a function need?
  - Arguments.
  - Storage for local variables.
  - To return control to the caller.
  - To execute regardless of who called it.
  - To call other functions (that call other functions...that call other functions)
- There are not instructions for this
  - It is a contract about how the function behaves
  - In particular, how it treats the resources that are shared between functions -- the registers and memory

```cpp
int Factorial(int x) {
    if (x == 0)
        return 1;
    else
        return x * Factorial(x - 1);
}
```
Register Discipline

- All registers are the same, but we assign them different uses.

<table>
<thead>
<tr>
<th>Name</th>
<th>number</th>
<th>use</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>Assemble Temp</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>$k0-$k1</td>
<td>26 - 27</td>
<td>Res. for OS</td>
<td>yes</td>
</tr>
<tr>
<td>$gp</td>
<td>28</td>
<td>global ptr</td>
<td>yes</td>
</tr>
<tr>
<td>$sp</td>
<td>29</td>
<td>stack ptr</td>
<td>yes</td>
</tr>
<tr>
<td>$fp</td>
<td>30</td>
<td>frame ptr</td>
<td>yes</td>
</tr>
<tr>
<td>$ra</td>
<td>31</td>
<td>return address</td>
<td>yes</td>
</tr>
</tbody>
</table>