INSTRUCTION SET ARCHITECTURE

Mahdi Nazm Bojnordi
Assistant Professor
School of Computing
University of Utah
Overview

- This lecture
  - Constant values
  - Immediate operands
  - Memory instructions
  - Instruction format
Constant Values

- Constant values are defined/used in code
  - Known to the programmer
  - Zero is commonly used

```c
int main() {
    int i, j;
    for (j = 0; j < 10; j++) {
        for (i = 0; i < mem_size >> 2; i += 16) {
            p[i] = 55;
        }
    }
    for (i = 0; i < mem_size >> 2; i += 16) {
        q[i] = 56;
    }
    return 0;
}
```

How to handle constants in the ISA?
Immediate Operand

- An instruction may require a constant as input

- An immediate instruction uses a constant number as one of the inputs (instead of a register operand)

- Putting a constant in a register requires addition to register $zero (a special register that always has zero in it) -- since every instruction requires at least one operand to be a register

- For example, putting the constant 1000 into a register:
  - addi $s0, $zero, 1000
The format of a load instruction:

```
lw $t0, 8($t3)
```

- **destination register**
- **source register**
- **any register**
- **a constant added to the register in brackets**
The format of a load instruction:

\[ \text{sw } \$t0, 8(\$t3) \]

- Source register
- Any register
- A constant added to the register in brackets
Example MIPS Translation

- int a, b, c, d[10]

- Task: bring a, b, c, d[0], and d[1] to $s1-$s5
Example MIPS Translation

- int a, b, c, d[10]

  - Task: bring a, b, c, d[0], and d[1] to $s1-$s5

    - addi $t0, $zero, 1000  # put base address 1000 in $t0;
      # $zero is a register that always equals zero
Example MIPS Translation

- **int a, b, c, d[10]**

- **Task: bring a, b, c, d[0], and d[1] to $s1-$s5**

```
addi $t0, $zero, 1000   # put base address 1000 in $t0;
                      # $zero is a register that always equals zero
lw  $s1, 0($t0)       # brings value of a into register $s1
lw  $s2, 4($t0)       # brings value of b into register $s2
lw  $s3, 8($t0)       # brings value of c into register $s3
lw  $s4, 12($t0)      # brings value of d[0] into register $s4
lw  $s5, 16($t0)      # brings value of d[1] into register $s5
```
Example MIPS Translation

Convert the following C code to assembly

Example MIPS Translation

- Convert the following C code to assembly

```
```

```
addi \$t0, \$zero, 1000   # put base address 1000 in \$t0;
# \$zero is a register that always equals zero
lw \$s0, 0($t0)            # a is brought into \$s0
lw \$s1, 20($t0)          # d[2] is brought into \$s1
add \$t1, \$s0, \$s1         # the sum is in \$t1
sw \$t1, 24($t0)           # \$t1 is stored into d[3]
```
Instructions are represented as 32-bit numbers
- Each instruction word has multiple fields

MIPS Instruction Types
- **R-type**
  - add $t0, $s1, $s2

```
   000000 10001 10010 01000 00000 100000

   00010010101…0010
   10001001001…1010
   11001001001…0001
   …
```

Memory

CPU
decoder
Instruction Formats

- Instructions are represented as 32-bit numbers
  - Each instruction word has multiple fields

MIPS Instruction Types

- **R-type**
  - `add $t0, $s1, $s2`

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>shamt</th>
<th>funct</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- CPU decoder
- Memory
Instruction Formats

- Instructions are represented as 32-bit numbers
  - Each instruction word has multiple fields

- MIPS Instruction Types
  - R-type
    - add $t0, $s1, $s2
  - I-type
    - lw $t0, 32($t1)

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>constant or address</th>
</tr>
</thead>
<tbody>
<tr>
<td>100011</td>
<td>01001</td>
<td>01000</td>
<td>0000000000100000</td>
</tr>
<tr>
<td>6 bits</td>
<td>5 bits</td>
<td>5 bits</td>
<td>16 bits</td>
</tr>
</tbody>
</table>

CPU decoder

Memory

00010010101...0010
10001001001...1010
11001001001...0001
...

Logical Operations

- Bitwise logical operations

<table>
<thead>
<tr>
<th>Logical operations</th>
<th>C operators</th>
<th>Java operators</th>
<th>MIPS instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shift left</td>
<td>&lt;&lt;</td>
<td>&lt;&lt;</td>
<td>sll</td>
</tr>
<tr>
<td>Shift right</td>
<td>&gt;&gt;</td>
<td>&gt;&gt;&gt;</td>
<td>srl</td>
</tr>
<tr>
<td>Bit-by-bit AND</td>
<td>&amp;</td>
<td>&amp;</td>
<td>and, andi</td>
</tr>
<tr>
<td>Bit-by-bit OR</td>
<td></td>
<td></td>
<td>or, ori</td>
</tr>
<tr>
<td>Bit-by-bit NOT</td>
<td>~</td>
<td>~</td>
<td>nor</td>
</tr>
</tbody>
</table>

- Shift
  - `sll $t2, $s0, 4`
  - `srl $t2, $s0, 4`
Logical Operations

- Bitwise logical operations

<table>
<thead>
<tr>
<th>Logical operations</th>
<th>C operators</th>
<th>Java operators</th>
<th>MIPS instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shift left</td>
<td>&lt;&lt;</td>
<td>&lt;&lt;</td>
<td>sll</td>
</tr>
<tr>
<td>Shift right</td>
<td>&gt;&gt;</td>
<td>&gt;&gt;&gt;</td>
<td>srl</td>
</tr>
<tr>
<td>Bit-by-bit AND</td>
<td>&amp;</td>
<td>&amp;</td>
<td>and, andi</td>
</tr>
<tr>
<td>Bit-by-bit OR</td>
<td></td>
<td></td>
<td>or, ori</td>
</tr>
<tr>
<td>Bit-by-bit NOT</td>
<td>~</td>
<td>~</td>
<td>nor</td>
</tr>
</tbody>
</table>

- Shift
- AND
  - `and $t0, $t1, $t2`
Logical Operations

- Bitwise logical operations

<table>
<thead>
<tr>
<th>Logical operations</th>
<th>C operators</th>
<th>Java operators</th>
<th>MIPS instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shift left</td>
<td>&lt;&lt;</td>
<td>&lt;&lt;</td>
<td>sll</td>
</tr>
<tr>
<td>Shift right</td>
<td>&gt;&gt;</td>
<td>&gt;&gt;&gt;</td>
<td>srl</td>
</tr>
<tr>
<td>Bit-by-bit AND</td>
<td>&amp;</td>
<td>&amp;</td>
<td>and, andi</td>
</tr>
<tr>
<td>Bit-by-bit OR</td>
<td></td>
<td></td>
<td>or, ori</td>
</tr>
<tr>
<td>Bit-by-bit NOT</td>
<td>~</td>
<td>~</td>
<td>nor</td>
</tr>
</tbody>
</table>

- Shift
- AND
- OR
- or $t0, $t1, $t2
Logical Operations

- Bitwise logical operations

<table>
<thead>
<tr>
<th>Logical operations</th>
<th>C operators</th>
<th>Java operators</th>
<th>MIPS instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shift left</td>
<td>&lt;&lt;</td>
<td>&lt;&lt;</td>
<td>sll</td>
</tr>
<tr>
<td>Shift right</td>
<td>&gt;&gt;</td>
<td>&gt;&gt;&gt;</td>
<td>srl</td>
</tr>
<tr>
<td>Bit-by-bit AND</td>
<td>&amp;</td>
<td>&amp;</td>
<td>and, andi</td>
</tr>
<tr>
<td>Bit-by-bit OR</td>
<td></td>
<td></td>
<td>or, ori</td>
</tr>
<tr>
<td>Bit-by-bit NOT</td>
<td>~</td>
<td>~</td>
<td>nor</td>
</tr>
</tbody>
</table>

- Shift
- AND
- OR
- NOT

- nor $t0, $t1, $t2