CS 4200/5200
Computer Architecture I

MIPS Instruction Set Architecture

Dr. Xiaobo Zhou
Department of Computer Science

Review: Organizational Trade-offs

\[
\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}
\]

Diagram:
- Application
- Programming Language
- Compiler
- ISA
- Datapath
- Control
- Function Units
- Transistors
- Wires
- Pins
- Instruction Mix
- CPI
- Cycle Time
**Review: Amdahl's Law**

ExTime after improvement = ExTime unaffected + Extime affected / amount of improvement

Speedup due to enhancement E:

\[
\text{Speedup}(E) = \frac{\text{ExTime w/o E}}{\text{ExTime w/ E}} = \frac{\text{Performance w/ E}}{\text{Performance w/o E}}
\]

Suppose that enhancement E accelerates a fraction F of the task by a factor S, and the remainder of the task is unaffected then,

\[
\text{ExTime(with E)} = ((1-F) + F/S) \times \text{ExTime(without E)}
\]

\[
\text{Speedup(with E)} = \frac{\text{ExTime(without E)}}{((1-F) + F/S) \times \text{ExTime(without E)}} = \frac{1}{(1-F) + F/S}
\]

---

**Instruction Set Design**

An instruction is a binary code, which specifies a basic operation (e.g. add, subtract, and, or) for the computer.

- **Operation Code**: defines the operation type
- **Operands**: operation source and destination
Levels of Representation

High Level Language
Program

Compiler

Assembly Language
Program

Assembler

Machine Language
Program

Machine Interpretation

MIPS R2000 / R3000 Registers

- MIPS (NEC, SGI, Sony, Nintendo), a typical of instructions after 1980s.

- 32-bit machine --> Programmable storage 2^32 x 4 bytes

- 31 x 32-bit GPRs (R0 = 0)

- 32 x 32-bit FP regs

- HI, LO, PC

- Big Endian

- Addressing modes:
  - register
  - immediate
  - displacement
  - PC-relative
  - pseudo-direct

- All instructions are 32-bit wide and must be aligned -- words must start at address that are multiple of 4.
### MIPS arithmetic instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Example</th>
<th>Meaning</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>add $1,$2,$3</td>
<td>$1 = $2 + $3</td>
<td>3 operands</td>
</tr>
<tr>
<td>subtract</td>
<td>sub $1,$2,$3</td>
<td>$1 = $2 – $3</td>
<td>3 operands</td>
</tr>
<tr>
<td>add immediate</td>
<td>addi $1,$2,100</td>
<td>$1 = $2 + 100</td>
<td>+ constant</td>
</tr>
<tr>
<td>sub immediate</td>
<td>subi $1,$2,100</td>
<td>$1 = $2 - 100</td>
<td>- constant</td>
</tr>
</tbody>
</table>

#### Example

E.g.  
\[ f = (g+h) - (i+j), \]
assuming \( f, g, h, i, j \) be assigned to \$1, \$2, \$3, \$4, \$5

- add $7, $2, $3 // register $7 contains \( g+h \)
- add $8, $4, $5 // register $8 contains \( i+j \)
- sub $1, $7, $8 // register $1 (f) gets the result
### MIPS logic instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Example</th>
<th>Meaning</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>and</td>
<td>and $1,$2,$3</td>
<td>$1 = $2 &amp; $3</td>
<td>Logical AND</td>
</tr>
<tr>
<td>or</td>
<td>or $1,$2,$3</td>
<td>$1 = $2</td>
<td>$3</td>
</tr>
<tr>
<td>xor</td>
<td>xor $1,$2,$3</td>
<td>$1 = $2 ⊕ $3</td>
<td>Logical XOR</td>
</tr>
<tr>
<td>nor</td>
<td>nor $1,$2,$3</td>
<td>$1 = ~(2</td>
<td>$3)</td>
</tr>
</tbody>
</table>

and imme. andi $1,$2, 100 $1 = $2 & 100 AND constant...

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>x and y</th>
<th>x or y</th>
<th>x xor y</th>
<th>x nor y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

### MIPS data transfer instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>LW $1, 30($2)</td>
<td>Load word</td>
</tr>
<tr>
<td>LH $1, 40($3)</td>
<td>Load half a word</td>
</tr>
<tr>
<td>LB $1, 40($3)</td>
<td>Load byte</td>
</tr>
<tr>
<td>SW $3, 500($4)</td>
<td>Store word</td>
</tr>
<tr>
<td>SH $3, 502($2)</td>
<td>Store half</td>
</tr>
<tr>
<td>SB $2, 41($3)</td>
<td>Store byte</td>
</tr>
</tbody>
</table>

In MIPS64: LD (load double word)

- LD $1, 30($2) load a double-word
- SD $3, 500($4) store a double-word
Example

Assume A is an array of 100 words, and compiler has associated the variables g and h with the register $1 and $2. Assume the base address of the array is in $3. Translate

\[ g = h + A[8] \]

\[
\begin{align*}
\text{lw} & \quad \text{4, 8($3)}; \quad \text{// 4 <-- A[8]} \\
\text{add} & \quad \text{1, 2, 4}; \quad \text{\downarrow} \\
\text{lw} & \quad \text{4, 32($3)}; \quad \text{// 4 <-- A[8]} \\
\text{add} & \quad \text{1, 2, 4} \\
A[12] = h + A[8] & \quad \text{SW} \quad \text{1, 48($3)}
\end{align*}
\]

Example

Assume A is an array of 100 words, and compiler has associated the variables g, h, and i with the register $1, $2, $5. Assume the base address of the array is in $3. Translate

\[ g = h + A[i] \]

\[
\begin{align*}
\text{add} & \quad \text{6, 5, 5}; \quad \text{// 6 = 2i} \\
\text{add} & \quad \text{6, 6, 6}; \quad \text{// 6 = 4i} \\
\text{add} & \quad \text{4, 3, 6}; \quad \text{// 4 <---absolute mem. address of} \\
& \quad \text{// A[i], used as new base} \\
\text{lw} & \quad \text{7, 0 ($4)}; \quad \text{// 7 = A[i]} \\
\text{add} & \quad \text{1, 2, 7}; \quad \text{// g = h + A[i]}
\end{align*}
\]
### MIPS branch, jump, compare instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Example</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>branch on equal</td>
<td>beq $1,$2,100</td>
<td>if ($1 == $2) go to PC+4+100</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Equal test; PC relative branch</td>
</tr>
<tr>
<td>branch on not eq</td>
<td>bne $1,$2,100</td>
<td>if ($1!= $2) go to PC+4+100</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Not equal test; PC relative</td>
</tr>
<tr>
<td>branch on eq. to 0</td>
<td>beqz $1,100</td>
<td>if ($1==0) go to PC+4+100</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero test; PC relative (pseudo-instruction; using $0)</td>
</tr>
<tr>
<td>jump</td>
<td>j 10000</td>
<td>go to 10000</td>
</tr>
<tr>
<td>jump register</td>
<td>jr $31</td>
<td>go to $31</td>
</tr>
<tr>
<td>jump and link</td>
<td>jal 10000</td>
<td>$31 = PC + 4; go to 10000</td>
</tr>
<tr>
<td>set on less than</td>
<td>slt $1,$2,$3</td>
<td>if ($2 &lt; $3) $1=1; else $1=0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Compare less than; 2's comp.</td>
</tr>
<tr>
<td>set less than imm.</td>
<td>slti $1,$2,100</td>
<td>if ($2 &lt; 100) $1=1; else $1=0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Compare &lt; constant; 2's comp.</td>
</tr>
</tbody>
</table>

### Example

```
if (i==j) go to L1;
  f = g + h;
L1:  f = f - i;
```

Assuming f, g, h, i, j ~ $1, $2, $3, $4, $5

```
beq $4, $5, L1
add $1, $2, $3
L1:  sub $1, $1, $4
```

```
  i == j ?
  yes
  no
f = g + h
L1:  f = f - i
```
Example

Loop: \( g = g + A[i]; \)
\( i = i + j; \)
if \( i \neq h \) go to Loop:

Assuming variables \( g, h, i, j \sim $1, $2, $3, $4 \) and base address of array is in \$5

Loop: \( \text{add $7, $3, $3} \)
\( \text{add $7, $7, $7} \)
\( \text{add $7, $7, $5} \)
\( \text{lw $6, 0($7)} \)
\( \text{add $1, $1, $6} // g = g + A[i] \)
\( \text{add $3, $3, $4} \)
\( \text{bne $3, $2, Loop;} \)

Example

while \( (A[i] == k) \)
\( i = i + j; \)
Assume \( i, j, \) and \( k \sim $17, $18, $19 \) and base of \( A \) is in \$3

\( A[i] == k ? \)
Yes
\( i = i + j \)
no
Exit
Example

while (A[i]==k)
    i = i+j;
Assume i, j, and k ~ $17, $18, $19 and base of A is in $3

Loop:   add $20, $17, $17
        add $20, $20, $20
        add $20, $20, $3

        lw $21,0($20)
        bne $21 $19 Exit       //exit loop if A[I] ! =k
        add $17, $17, $18
        j      Loop
Exit:

MIPS Fields

R-format:

       6   5   5   5   5       6
Register (direct)  op    rs   rt   rd  shamt  funct

*op: basic operation of the instruction, called opcode
*rs, the first register source operand
*rt: the second register source operand
*rd: the register destination operand
*shamt: shift amount
*func: function, select the variant of the op field.

I-format:

       6   5   5       16
Immediate op    rs   rt        immediate

J-format:

       6       26
jump op   addr.
MIPS Addressing Modes/Instruction Formats

R-format: Register (direct)

I-format: Immediate

Base+offset displacement

PC-relative

J-format: pseudodirect

Example

while (A[i] == k)
    i = i + j;
Assume i, j, and k ~ $17, $18, $19 and base of A is in $3

Assume the loop is placed starting at loc 8000

Loop:
1. add $20, $17, $17
2. add $20, $20, $20
3. add $20, $20, $3
4. lw $21,0($20)
5. bne $21, $19, Exit
6. add $17, $17, $18
   j Loop

Exit:

Loop: | 0 17 17 20 0 32 |
     | 0 20 20 20 0 32 |
     | 0 20 3 20 0 32 |
     | 35 20 21 0      |
     | 5 21 19 8 -> 2  |
     | 0 17 18 17 0 32 |
     | 2 8000 -> 2000  |
### MIPS arithmetic & logical instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Example</th>
<th>Meaning</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td>add $1,$2,$3</td>
<td>$1 = $2 + $3</td>
<td>3 operands;</td>
</tr>
<tr>
<td>subtract</td>
<td>sub $1,$2,$3</td>
<td>$1 = $2 – $3</td>
<td>3 operands;</td>
</tr>
<tr>
<td>add immediate</td>
<td>addi $1,$2,100</td>
<td>$1 = $2 + 100 + constant</td>
<td></td>
</tr>
<tr>
<td>add unsigned</td>
<td>addu $1,$2,$3</td>
<td>$1 = $2 + $3</td>
<td>3 operands;</td>
</tr>
<tr>
<td>subtract unsigned</td>
<td>subu $1,$2,$3</td>
<td>$1 = $2 – $3</td>
<td>3 operands;</td>
</tr>
<tr>
<td>add imm. unsign.</td>
<td>addiu $1,$2,100</td>
<td>$1 = $2 + 100 + constant</td>
<td></td>
</tr>
<tr>
<td>and</td>
<td>and $1,$2,$3</td>
<td>$1 = $2 &amp; $3</td>
<td>3 reg. operands;</td>
</tr>
<tr>
<td>or</td>
<td>or $1,$2,$3</td>
<td>$1 = $2</td>
<td>$3</td>
</tr>
<tr>
<td>xor</td>
<td>xor $1,$2,$3</td>
<td>$1 = $2 $3</td>
<td>3 reg. operands;</td>
</tr>
<tr>
<td>nor</td>
<td>nor $1,$2,$3</td>
<td>$1 = ~(S2</td>
<td>S3)</td>
</tr>
<tr>
<td>and immediate</td>
<td>andi $1,$2,10</td>
<td>$1 = $2 &amp; 10</td>
<td>Logical AND reg, constant</td>
</tr>
<tr>
<td>or immediate</td>
<td>ori $1,$2,10</td>
<td>$1 = $2</td>
<td>10</td>
</tr>
<tr>
<td>xor immediate</td>
<td>xori $1,$2,10</td>
<td>$1 = ~S2 &amp;~10</td>
<td>Logical XOR reg, constant</td>
</tr>
</tbody>
</table>

### MIPS data transfer instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>LW $1, 30($2)</td>
<td>Load word</td>
</tr>
<tr>
<td>LH $1, 40($3)</td>
<td>Load half a word</td>
</tr>
<tr>
<td>LB $1, 40($3)</td>
<td>Load byte</td>
</tr>
<tr>
<td>SW $3, 500($4)</td>
<td>Store word</td>
</tr>
<tr>
<td>SH $3, 502($2)</td>
<td>Store half</td>
</tr>
<tr>
<td>SB $2, 41($3)</td>
<td>Store byte</td>
</tr>
</tbody>
</table>

In MIPS64: LD (load double word)
- LD $1, 30($2) load a double-word
- SD $3, 500($4) store a double-word
Compare and Branch

° Compare and Branch
  • BEQ rs, rt, offset if R[rs] == R[rt] then PC-relative branch
  • BNE rs, rt, offset <>

° Compare to zero and branch
  • BLEZ rs, offset if R[rs] <= 0 then PC-relative branch
  • BGTZ rs, offset >
  • BLT <
  • BGEZ >=
  • BLTZAL rs, offset if R[rs] < 0 then branch and link (into R 31)
  • BGEZAL >=

Reading and Preview

• Reading:
  CO 4: Chapter 2 (MIPS)
  CA 5: Appendix A (ISA)