ECE4680
Computer Organization & Architecture

MIPS Instruction Set Architecture

Why is MIPS a good example?
Learn ISA further by this example.
How does IS fill up the gap between HLL and machine?

RISC Vs. CISC

RISC → CISC
- Determined by VLSI technology.
- Software cost goes up constantly. To be convenient for programmers.
- To shorten the semantic gap between HLL and architecture without advanced compilers.
- To reduce the program length because memory was expensive.
- VAX 11/780 reached the climax with >300 instructions and >20 addressing modes.

CISC → RISC
- Things changed: HLL, Advanced Compiler, Memory size, ...
- Finding: 25% instructions used in 95% time.
- Size: usually <100 instructions and <5 addressing modes.
- Other properties: fixed instruction format, register based, hardware control...
- Gains: CPI is smaller, Clock cycle shorter, Hardware simpler, Pipeline easier
- Loss: Program becomes longer, but memory becomes larger and larger, cheaper and cheaper. Programmability becomes poor, but people use HLL instead of IS.
- Result: although program is prolonged, the total gain is still a plus.
MIPS R2000 / R3000 Registers

- 32-bit machine --> Programmable storage \(2^{32}\) x bytes
- 31 x 32-bit GPRs (R0 = 0)
- 32 x 32-bit FP regs (f0 - f31, paired DP)
- Hi, LO, PC: SPRegisters
- Big Endian
- 2 nomenclatures(next slide)
- See Fig. A.18 at P. A-50 for more details

Addressing modes:
- immediate
- register
- displacement
- All instructions are 32-bit wide and must be aligned.

2 Nomenclatures of MIPS Registers (p.140, A-23)

<table>
<thead>
<tr>
<th>Name</th>
<th>number</th>
<th>Usage</th>
<th>Reserved on call?</th>
</tr>
</thead>
<tbody>
<tr>
<td>zero</td>
<td>0</td>
<td>constant value = 0</td>
<td>n.a.</td>
</tr>
<tr>
<td>at</td>
<td>1</td>
<td>reserved for assembler</td>
<td>n.a.</td>
</tr>
<tr>
<td>v0 – v1</td>
<td>2 – 3</td>
<td>values for results and expression evaluation</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>more temporaries</td>
<td>no</td>
</tr>
<tr>
<td>k0 – k1</td>
<td>26 – 27</td>
<td>Reserved for kernel</td>
<td>n.a.</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>

<table>
<thead>
<tr>
<th>Name</th>
<th>number</th>
<th>Usage</th>
<th>Reserved on call?</th>
</tr>
</thead>
<tbody>
<tr>
<td>zero</td>
<td>0</td>
<td>constant value = 0</td>
<td>n.a.</td>
</tr>
<tr>
<td>at</td>
<td>1</td>
<td>reserved for assembler</td>
<td>n.a.</td>
</tr>
<tr>
<td>v0 - v1</td>
<td>2 - 3</td>
<td>values for results and expression evaluation</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>more temporaries</td>
<td>no</td>
</tr>
<tr>
<td>k0 - k1</td>
<td>26 - 27</td>
<td>Reserved for kernel</td>
<td>n.a.</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>

ECE4680 Lec4 MIPS.4 February 6, 2002
### MIPS arithmetic and logic 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>$1 = 2 + 3$ $3$ operands; exception possible</td>
<td></td>
</tr>
<tr>
<td>subtract</td>
<td>sub $1$, $2$, $3$</td>
<td>$1 = 2 - 3$          $3$ operands; exception possible</td>
<td></td>
</tr>
<tr>
<td>add immediate</td>
<td>addi $1$, $2$, 100</td>
<td>$1 = 2 + 100$ + constant; exception possible</td>
<td></td>
</tr>
<tr>
<td>multiply</td>
<td>mult $2$, $3$</td>
<td>Hi, Lo = $2 \times 3$ 64-bit signed product</td>
<td></td>
</tr>
<tr>
<td>divide</td>
<td>div $2$, $3$</td>
<td>Lo = $2 \div 3$, Hi = remainder</td>
<td></td>
</tr>
<tr>
<td>Move from Hi</td>
<td>mfhi $1$</td>
<td>$1=Hi$ get a copy of Hi</td>
<td></td>
</tr>
<tr>
<td>Move from Lo</td>
<td>mflo $1$</td>
<td>$1=lo$</td>
<td></td>
</tr>
<tr>
<td>and</td>
<td>and $1$, $2$, $3$</td>
<td>$1 = 2 &amp; 3$ Logical AND</td>
<td></td>
</tr>
<tr>
<td>or</td>
<td>or $1$, $2$, $3$</td>
<td>$1 = 2 \mid 3$ Logical OR</td>
<td></td>
</tr>
<tr>
<td>xor</td>
<td>xor $1$, $2$, $3$</td>
<td>$1 = 2 \oplus 3$ Logical XOR</td>
<td></td>
</tr>
<tr>
<td>nor</td>
<td>nor $1$, $2$, $3$</td>
<td>$1 = \neg(2 \mid 3)$ Logical NOR</td>
<td></td>
</tr>
</tbody>
</table>

---

### Example (p110)

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

- add $7$, $2$, $3$
- add $8$, $4$, $5$
- sub $1$, $7$, $8$
### MIPS data transfer instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>SW 500($4), $3</td>
<td>Store word</td>
</tr>
<tr>
<td>SH 502($2), $3</td>
<td>Store half</td>
</tr>
<tr>
<td>SB 41($3), $2</td>
<td>Store byte</td>
</tr>
<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>
</tbody>
</table>

### Example (pp112-114)

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

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

\[
\begin{align*}
\text{lw} & \quad $4, 8($3); \quad 4 \leftarrow A[8] \\
\text{add} & \quad $1, $2, $4;
\end{align*}
\]

\[
\begin{align*}
\text{lw} & \quad $4, 32($3); \\
\text{add} & \quad $1, $2, $4
\end{align*}
\]

\[ A[12] = h + A[8] \quad \text{SW $1, 48($3)} \]
**Example (p114)**

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{addi} & \quad $6, 0, 4; \quad \text{$6 = 4$} \\
\text{mult} & \quad $5, $6; \quad \text{Hi,Lo = i*4} \\
\text{mflo} & \quad $6; \quad \text{$6 = i*4$, assuming i is small} \\
\text{add} & \quad $4, $3, $6; \quad \text{$4 \leftarrow \text{address of } A[i]$} \\
\text{add} & \quad $1, $2, $4
\end{align*}

---

**MIPS jump, branch, compare instructions**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Example</th>
<th>Meaning</th>
</tr>
</thead>
</table>
| branch on equal      | beq $1,$2,100 | if ($1 == $2) go to PC+4+100  
                             *Equal test; PC relative branch* |
| branch on not eq.    | bne $1,$2,100  | if ($1!= $2) go to PC+4+100  
                             *Not equal test; PC relative* |
| Pseudoinstruction    | slt, bge, bgt | not implemented by hardware,  
                             but synthesized by assembler |
| set less than        | slt $1,$2,$3  | if ($2 < $3) $1=1; else $1=0  
                             *Compare less than; 2’s comp.* |
| set less than imm.   | slli $1,$2,100 | if ($2 < 100) $1=1; else $1=0  
                             *Compare < constant; 2’s comp.* |
| jump                 | j $10000     | go to 10000  
                             *Jump to target address* |
| jump register        | jr $31       | go to $31  
                             *For switch, procedure return* |
| jump and link        | jal $10000   | $31 = PC + 4; go to 10000  
                             *For procedure call* |
Example (p123)

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  \text{ \textit{pseudoinstruction}}
add $1, $2, $3
L1: sub $1, $1, $4

\text{\textit{assembler}}

bqe $4, $5, L1
slt $1, $4, $5
beq $0, $1, L1

Example (p126)

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

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

Loop: add $7, $3, $3; i*2
      add $7, $7, $7; i*4
      add $7, $7, $5
      lw $6, 0($7); $6=A[i]
      add $1, $1, $6; g = g + A[i]
      add $3, $3, $4
      bne $3, $2, Loop;
**Example** (p127)

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  
   add $17, $17, $18  
   j Loop

Exit:

---

**MIPS Addressing Modes/Instruction Formats** (p118,148,152)

**R-format:**

Register (direct)

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>rd</th>
<th>smt</th>
<th>func</th>
</tr>
</thead>
</table>

**I-format:**

Immediate

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>immed</th>
</tr>
</thead>
</table>

**Base-index**

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>immed</th>
</tr>
</thead>
</table>

**PC-relative**

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>immed</th>
</tr>
</thead>
</table>

**J-format:**

<table>
<thead>
<tr>
<th>op</th>
<th>addr.</th>
</tr>
</thead>
</table>
Example: See machine code in memory (p149)

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:  add $20, $17, $17
        add $20, $20, $20
        add $20, $20, $3
        lw $21,0($20)
        bne $21, $19, Exit
        add $17, $17, $18
        j Loop

Exit:

Offset in branch is relative. Address in jump is absolute.
Address in Branch or Jump instruction is word address so that
they can go 4 times far as opposed to byte address. (p150)

Procedure Call and Stack

Stacking of Subroutine Calls & Returns and Environments:

A:
    CALL B
B:
    CALL C
C:
    RET

Some machines provide a memory stack(special hardware) as part of the
architecture (e.g. the VAX). Use special instructions, e.g. pop, push.

Sometimes stacks are implemented via software convention (e.g. MIPS).
Use same data transfer instructions, e.g., lw, sw.
Example in C: swap (pp163-165)

° Assume swap is called as a procedure
° Assume temp is register $15; arguments v and k ~ $16 and $17;
° Write MIPS code

```
swap(int v[], int k)
{
    int temp;
    temp = v[k];
    v[k] = v[k+1];
    v[k+1] = temp;
}
```

° Assume temp is register $15; arguments v and k ~ $16 and $17;
° Write MIPS code

```
sll $18, $17, 2  ; multiply k by 4
addu $18, $18, $16 ; address of v[k]
lw $15, 0($18) ; load v[k]
lw $19, 4($18) ; load v[k+1]
sw $19, 0($18) ; store v[k+1] into v[k]
sw $15, 4($18) ; store old v[k] into v[k+1]
```

 Registers $15, $16, $17, $18, $19 are occupied by caller ??
swap: MIPS

addi $sp,$sp, –24 ; Make room on stack for 6 registers
sw $31, 20($sp) ; Save return address
sw $15, 16($sp) ; Save registers on stack
sw $16, 12($sp)
sw $17, 8($sp)
sw $18, 4($sp)
sw $19, 0(sp)
....
lw $19, 0($sp) ; Restored registers from stack
lw $18, 4($sp)
lw $17, 8($sp)
lw $16, 12($sp)
lw $15, 16($sp)
lw $31, 20($sp) ; Restore return address
addi $sp,$sp, 24 ; restore top of stack
jr $31 ; return to place that called swap

Other ISAs

° Intel 8086/88 => 80286 => 80386 => 80486 => Pentium => P6
  • 8086 few transistors to implement 16-bit microprocessor
  • tried to be somewhat compatible with 8-bit microprocessor 8080
  • successors added features which were missing from 8086 over next 15 years
  • product of several different Intel engineers over 10 to 15 years
  • Announced 1978
° VAX simple compilers & small code size =>
  • efficient instruction encoding
  • powerful addressing modes
  • powerful instructions
  • few registers
  • product of a single talented architect
  • Announced 1977
## Machine Examples: Address & Registers

<table>
<thead>
<tr>
<th>Machine</th>
<th>Bytes</th>
<th>Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel 8086</td>
<td>$2^{20}$ x 8 bytes</td>
<td>AX, BX, CX, DX, SP, BP, SI, DI, CS, SS, DS, IP, Flags</td>
</tr>
<tr>
<td></td>
<td>acc, index, count, quot</td>
<td>stack, string</td>
</tr>
<tr>
<td></td>
<td>code, stack, data segment</td>
<td></td>
</tr>
<tr>
<td>VAX 11</td>
<td>$2^{32}$ x 8 bytes</td>
<td>r15-- program counter</td>
</tr>
<tr>
<td></td>
<td>16 x 32 bit GPRs</td>
<td>r14-- stack pointer</td>
</tr>
<tr>
<td></td>
<td>r13-- frame pointer</td>
<td>r12-- argument ptr</td>
</tr>
<tr>
<td>MC 68000</td>
<td>$2^{24}$ x 8 bytes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>8 x 32 bit GPRs</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7 x 32 bit addr reg</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 x 32 bit SP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 x 32 bit PC</td>
<td></td>
</tr>
<tr>
<td>MIPS</td>
<td>$2^{32}$ x 8 bytes</td>
<td>r15-- program counter</td>
</tr>
<tr>
<td></td>
<td>32 x 32 bit GPRs</td>
<td>r14-- stack pointer</td>
</tr>
<tr>
<td></td>
<td>32 x 32 bit FPRs</td>
<td>r13-- frame pointer</td>
</tr>
<tr>
<td></td>
<td>HI, LO, PC</td>
<td>r12-- argument ptr</td>
</tr>
</tbody>
</table>

---

## Homework 2, due Feb. 20, 2002

- Questions 3.2, 3.3, 3.5, 3.6, 3.7, 3.9, 3.11
- To answer question 3.7, please refer to Figure 3.13 (page 140) for register convention
- To answer 3.11, please refer to sort example in pages 166 for a skeleton of for loop