MIPS

MIPS is a “computer family”
- R2000/R3000 (32-bit)
- R4000/4400 (64-bit)
- R8000 (for scientific & graphics applications)
- R10000 & R12000 (64-bit & out-of-order execution)

MIPS originated as a Stanford research project
Microprocessor without Interlocked Pipe Stages

MIPS was bought by Silicon Graphics (SGI) & is now independent
- focuses on embedded processors for game machines (e.g., Nintendo)

MIPS is a RISC

MIPS Registers

Part of the state of a process

Thirty-two 32-bit general purpose registers (GPRs): $0, $1, ..., $31
- integer arithmetic
- address calculations
- temporary values

By convention software uses different registers for different purposes (next slide)

A 32-bit program counter (PC)

Two 32-bit registers HI and LO used specifically for multiply and divide
- HI & LO concatenated for the product
- LO for the quotient; HI for the remainder

Thirty-two 32-bit registers used for floating-point arithmetic: $f0, $f1, ..., $f31
- often used as 16 64-bit registers for double precision FP

Other special-purpose registers (later)
## MIPS Register Names and gcc Conventions

<table>
<thead>
<tr>
<th>Register</th>
<th>Name</th>
<th>Use</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0</td>
<td>zero</td>
<td>always 0</td>
<td>cannot be written</td>
</tr>
<tr>
<td>$1</td>
<td>$at</td>
<td>reserved for assembler</td>
<td>don’t use it!</td>
</tr>
<tr>
<td>$2, $3</td>
<td>$v0, $v1</td>
<td>function return</td>
<td></td>
</tr>
<tr>
<td>$4 - $7</td>
<td>$a0 - $a3</td>
<td>pass first 4 procedure/function arguments</td>
<td></td>
</tr>
<tr>
<td>$8 - $15</td>
<td>$t0 - $t7</td>
<td>temporaries</td>
<td>caller saved (callee uses them without saving them)</td>
</tr>
<tr>
<td>$16 - $23</td>
<td>$s0 - $s7</td>
<td>temporaries</td>
<td>callee saved (callee assumes they will be available on function return)</td>
</tr>
<tr>
<td>$24, $25</td>
<td>$t8, $t9</td>
<td>temporaries</td>
<td>caller saved</td>
</tr>
<tr>
<td>$26, $27</td>
<td>$k0, $k1</td>
<td>reserved for the OS</td>
<td>don’t use them!</td>
</tr>
<tr>
<td>$28</td>
<td>$gp</td>
<td>pointer to global static memory</td>
<td>points to the middle of a 64KB block in the static data segment (next slide)</td>
</tr>
<tr>
<td>$29</td>
<td>$sp</td>
<td>stack pointer</td>
<td>points to the last allocated stack location (next slide)</td>
</tr>
<tr>
<td>$30</td>
<td>$fp</td>
<td>frame pointer</td>
<td>points to the stack frame (later)</td>
</tr>
<tr>
<td>$31</td>
<td>$ra</td>
<td>procedure/function return address</td>
<td></td>
</tr>
</tbody>
</table>

## Memory Usage

A software convention

- **text segment**: the code
- **data segment**
  - **static data**: objects whose size is known to the compiler & whose lifetime is the whole program execution
  - **dynamic data**: objects allocated as the program executes (malloc, new)
- **stack segment**: LIFO process-local storage
MIPS Load-Store Architecture

Most instructions compute on operands stored in registers
- load data into a register from memory
- compute in registers
- the result is stored into memory

For example:
\[
\begin{align*}
a &= b + c \\
d &= a + b
\end{align*}
\]
is “compiled” into:
- load b into register $x$
- load c into register $y$
- $z \leftarrow x + y$
- store $z$ into a
- $z \leftarrow z + x$
- store $z$ into d

MIPS Information Units

Data types and sizes
- byte
- half-word (2 bytes)
- word (4 bytes)
- float (4 bytes using single-precision floating-point format)
- double (8 bytes using double-precision floating-point format)

Memory is byte-addressable

A data type must start on an address evenly divisible by its size in bytes
Every word starts at an address that is divisible by 4. Which byte in the word is byte 0? How is the data in \texttt{.byte a, b, c, d} stored?

**Big Endian**

<table>
<thead>
<tr>
<th>31</th>
<th>24</th>
<th>23</th>
<th>16</th>
<th>15</th>
<th>8</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>f</td>
<td>g</td>
<td>h</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0's</td>
<td>0's</td>
<td>0's</td>
<td>110\textsubscript{2}</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Most significant byte is the lowest byte address. Word is addressed by the byte address of the most significant byte.

**Little Endian**

<table>
<thead>
<tr>
<th>31</th>
<th>24</th>
<th>23</th>
<th>16</th>
<th>15</th>
<th>8</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>c</td>
<td>b</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>h</td>
<td>g</td>
<td>f</td>
<td>e</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0's</td>
<td>0's</td>
<td>0's</td>
<td>110\textsubscript{2}</td>
<td>12</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Least significant byte is the lowest byte address. Word is addressed by the byte address of the least significant byte.

Problems when transferring data structures that contain a mixture of integers & characters between big & little endian computers.

Transfer from big endian computer to little endian computer
MIPS Information Units

MIPS supports both big- and little-endian byte orders

SPIM uses the byte order of the machine it’s running on
- Intel: little-endian
- Alpha, SPARC, Mac: big-endian

Words in SPIM are listed from left to right, but byte addresses are little-endian within a word

byte 0x7fffebd2
word 0x7fffebd4
half-word 0x7fffebde

[0x7fffebd0] 0x00400018 0x00000001 0x00010aff 0x00000005 0x00010aff