C335 Computer Structures

MIPS Instructions (Part #2)

Dr. David R. Surma

Department of Computer and Information Sciences

Adapted from Morgan Kaufmann, Dr. L. Zhang, and others
The datapath **executes** the instructions as directed by control.

Memory stores both instructions and data.

**Processor**
- Control: 000000 00100 00010 0001000000100000
- Datapath: contents Reg #4 ADD contents Reg #2 results put in Reg #2

**Memory**

**Devices**
- Input
- Output
Processor Organization

- **Control** needs to have *circuitry* to:
  - Decide which is the next instruction and input it from memory
  - Decode the instruction
  - Issue signals that control the way information flows between datapath components
  - Control what operations the datapath’s functional units perform

- **Datapath** needs to have *circuitry* to
  - Execute instructions - functional units (e.g., adder) and storage locations (e.g., register file)
  - Interconnect the functional units so that the instructions can be executed as required
  - Load data from and store data to memory
Review and more: RISC vs. CISC

- **RISC philosophy**
  - fixed instruction lengths
  - load-store instruction sets
  - limited number of addressing modes
  - limited number of operations

- **MIPS, ARM, Sun SPARC, HP PA-RISC, IBM PowerPC …**

- Instruction sets are measured by how well compilers use them as opposed to how well assembly language programmers use them

- **CISC (C for complex), e.g., Intel x86**
Review and More: ISA

- The language of the machine
  - Want an ISA that makes it easy to build the hardware and the compiler while maximizing performance and minimizing cost

- Stored program (von Neumann) concept
  - Instructions are stored in memory (as is the data)

- Our target: the MIPS ISA
  - similar to other ISAs developed since the 1980's
  - used by Broadcom, Cisco, NEC, Nintendo, Sony, ...

Design goals: maximize performance, minimize cost, reduce design time (time-to-market), minimize memory space (embedded systems), minimize power consumption (mobile systems)
The Four Design Principles

1. _____ favors regularity.
2. _____ is faster.
3. Make the _____ _____ fast.
4. Good design demands good __________.
Review and More: MIPS Arithmetic Instruction

- MIPS assembly language arithmetic statement
  
  - `add $t0, $s1, $s2`
  
  - `sub $t0, $s1, $s2`

- Each arithmetic instruction performs only one operation

- Each arithmetic instruction specifies exactly three operands

  - Destination $\leftarrow$ Source1  \text{ op } Source2

  - Operand order is fixed (the destination is specified first)

- The operands are contained in the datapath’s register file ($t0, s1, s2$)
Operands of arithmetic instructions must be from a limited number of special locations contained in the datapath’s register file:

- Thirty-two 32-bit registers
  - Two read ports
  - One write port

Registers are:

- Fast
  - Smaller is faster & Make the common case fast
- Easy for a compiler to use
  - e.g., \((A*B) - (C*D) - (E*F)\) can do multiplies in any order
- Improves code density
  - Since registers are named with fewer bits than a memory location
### Naming Conventions for Registers

<table>
<thead>
<tr>
<th>Register</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$zero constant 0 (Hardware)</td>
</tr>
<tr>
<td>1</td>
<td>$at reserved for assembler</td>
</tr>
<tr>
<td>2</td>
<td>$v0 expression evaluation &amp;</td>
</tr>
<tr>
<td>3</td>
<td>$v1 function results</td>
</tr>
<tr>
<td>4</td>
<td>$a0 arguments</td>
</tr>
<tr>
<td>5</td>
<td>$a1</td>
</tr>
<tr>
<td>6</td>
<td>$a2</td>
</tr>
<tr>
<td>7</td>
<td>$a3</td>
</tr>
<tr>
<td>8</td>
<td>$t0 temporary: caller saves</td>
</tr>
<tr>
<td></td>
<td>(callee can clobber)</td>
</tr>
<tr>
<td>15</td>
<td>$t7</td>
</tr>
<tr>
<td>16</td>
<td>$s0 callee saves</td>
</tr>
<tr>
<td></td>
<td>(caller can clobber)</td>
</tr>
<tr>
<td>23</td>
<td>$s7</td>
</tr>
<tr>
<td>24</td>
<td>$t8 temporary (cont’d)</td>
</tr>
<tr>
<td>25</td>
<td>$t9</td>
</tr>
<tr>
<td>26</td>
<td>$k0 reserved for OS kernel</td>
</tr>
<tr>
<td>27</td>
<td>$k1</td>
</tr>
<tr>
<td>28</td>
<td>$gp pointer to global area</td>
</tr>
<tr>
<td>29</td>
<td>$sp stack pointer</td>
</tr>
<tr>
<td>30</td>
<td>$fp frame pointer</td>
</tr>
<tr>
<td>31</td>
<td>$ra return address (Hardware)</td>
</tr>
</tbody>
</table>
Registers vs. Memory

- Arithmetic instructions **operands** must be in registers
  - only thirty-two registers are provided

- Compiler associates variables with registers

- What about programs with *lots* of variables?
Memory is a large, single-dimensional array
An address acts as the index into the memory array

Processor – Memory Interconnections

- The data stored in the memory
- The address of the data

Processor

Memory

32 bits = 4 Bytes = 1 Word

How many Locations?

$2^{32}$ Bytes
(4 GB)
$\rightarrow 2^{30}$ Words
(1 GW)

32 locations

10
101
1
Compiling with Loads and Stores

Assuming variable \( b \) is stored in \( s_2 \) and that the base address of array \( A \) is in \( s_3 \), what is the MIPS assembly code for the C statement

\[
\]

\[
\begin{array}{c|c}
\text{A[0]} & s_3 \\
\hline
\text{A[1]} & s_3 + 4 \\
\text{A[2]} & s_3 + 8 \\
\text{A[3]} & s_3 + 12 \\
\hline
\end{array}
\]

\[
lw \quad t0, 8(s_3) \\
sub \quad t0, t0, s2 \\
sw \quad t0, 32(s_3)
\]
Assuming that the base address of array A is in register $s4$, and variables b, c, and i are in $s1$, $s2$, and $s3$, respectively, what is the MIPS assembly code for the C statement
\[ c = A[i] - b \]