MIPS History

• MIPS is a computer family
  – R2000/R3000 (32-bit); R4000/4400 (64-bit); R8000; R10000 (64-bit) etc.
• MIPS originated as a Stanford research project under the direction of John Hennessy
  – Microprocessor without Interlocked Pipe Stages
• MIPS Co. bought by SGI
• MIPS used in previous generations of DEC (then Compaq, now HP) workstations
• Now MIPS Technologies is in the embedded systems market
• **MIPS is a RISC**
MIPS is a RISC

- **RISC** = *Reduced Instruction Set Computer*
- R could also stand for “regular”
- All arithmetic-logical instructions are of the form
  \[
  R_a \leftarrow R_b \text{ op } R_c
  \]

- MIPS (as all RISC’s) is a *Load-Store* architecture
  - ALU operates only on operands that are in registers
  - The only instructions accessing memory are load and store
Registers

• Registers are the “bricks” of the CPU
• Registers are an essential part of the ISA
  – Visible to the hardware and to the programmer
• Registers are
  – Used for high speed storage for operands. For example, if $a, b, c$ are in registers 8, 9, 10 respectively
    
    \[
    \text{add } $8,$9,$10 \quad \# a = b + c
    \]
  – Easy to name (most computers have 32 (integer) registers visible to the programmer and their names are 0, 1, 2, \ldots , 31)
  – Used also for addressing memory
Registers (ct’d)

• Not all registers are “equal”
  – Some are special-purpose (e.g., register 0 in MIPS is wired to the value 0)
  – Some are used for integer and some for floating-point (e.g., 32 of each in MIPS)
  – Some have restricted use by convention (cf. App. A pp A22-23)
  – Why no more than 32 or 64 registers
    • Well, sometimes there is (SPARC, Itanium, Cray, Tera)
    • Smaller is faster
    • Instruction encoding (names have to be short)
    • There can be more registers but they are invisible to the ISA
      – this is called register renaming (see CSE 471)
Memory system

- Memory is a hierarchy of devices with faster and more expensive ones closer to CPU
  - Registers
  - Caches (hierarchy: on-chip, off-chip)
  - Main memory (DRAM)
  - Secondary memory (disks)
Information units

• Basic unit is the *bit* (has value 0 or 1)
• Bits are grouped together in information units:
  – *Byte* = 8 bits
  – *Word* = 4 bytes (= 32 bits: the length of a MIPS integer register)
  – Double word = 2 words
  – etc.
Memory addressing

- Memory is a single-dimensional array of information units
  - Each unit has the same size
  - Each unit has its own address
  - Address of an unit and contents of the unit at that address are different
Addressing

• In most of today’s computers, the basic I-unit that can be addressed is a byte
  – MIPS is *byte addressable*

• The *address space* is the set of all I-units that a program can reference
  – The address space is tied to the length of the registers
  – MIPS has 32-bit registers. Hence its address space is 4G bytes
  – Older micros (minis) had 16-bit registers, hence 64 KB address space (too small)
  – Some current (Alpha, Itanium, Sparc, Altheon, Pentium 4-EMT64) machines have 64-bit registers, hence an enormous address space
The CPU - Instruction Execution Cycle

• The CPU executes a program by repeatedly following this cycle
  1. Fetch the next instruction, say instruction $i$
  2. Execute instruction $i$
  3. Compute address of the next instruction, say $j$
  4. Go back to step 1

• Of course we’ll optimize this but it’s the basic concept
What’s in an instruction?

• An instruction tells the CPU
  – the operation to be performed via the **OPCODE**
  – where to find the operands (source and destination)

• For a given instruction, the ISA specifies
  – what the OPCODE means (semantics)
  – how many operands are required and their types, sizes etc. (syntax)

• **Operand is either**
  – register (integer, floating-point, PC)
  – a memory address
  – a constant
ISA MIPS Registers

- Thirty-two 32-bit registers $0, $1, … , $31 used for
  - integer arithmetic; address calculation; temporaries; special-purpose functions (stack pointer etc.)
- A 32-bit Program Counter (PC)
- Two 32-bit registers (HI, LO) used for mult. and division
- Thirty-two 32-bit registers $f0, $f1, … , $f31 used for floating-point arithmetic
  - Often used in pairs: 16 64-bit registers
- Registers are a major part of the "state" of a process
## MIPS Register names and conventions

<table>
<thead>
<tr>
<th>Register</th>
<th>Name</th>
<th>Function</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0</td>
<td>Zero</td>
<td>Always 0</td>
<td>No-op on write</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>Expr. Eval/func. Return</td>
<td></td>
</tr>
<tr>
<td>$4-7</td>
<td>$a0-a3</td>
<td>Proc./func. Call parameters</td>
<td></td>
</tr>
<tr>
<td>$8-15</td>
<td>$t0-t7</td>
<td>Temporaries; volatile</td>
<td>Not saved on proc. Calls</td>
</tr>
<tr>
<td>$16-23</td>
<td>$s0-s7</td>
<td>Temporaries</td>
<td>Should be saved on calls</td>
</tr>
<tr>
<td>$24-25</td>
<td>$t8-t9</td>
<td>Temporaries; volatile</td>
<td>Not saved on proc. Calls</td>
</tr>
<tr>
<td>$26-27</td>
<td>$k0-k1</td>
<td>Reserved for O.S.</td>
<td>Don’t use them</td>
</tr>
<tr>
<td>$28</td>
<td>$gp</td>
<td>Pointer to global static memory</td>
<td></td>
</tr>
<tr>
<td>$29</td>
<td>$sp</td>
<td>Stack pointer</td>
<td></td>
</tr>
<tr>
<td>$30</td>
<td>$fp</td>
<td>Frame pointer</td>
<td></td>
</tr>
<tr>
<td>$31</td>
<td>$ra</td>
<td>Proc./func return address</td>
<td></td>
</tr>
</tbody>
</table>
MIPS = RISC = Load-Store architecture

• Every operand must be in a register
  – Except for some small integer constants that can be in the instruction itself (see later)
• Variables have to be *loaded* in registers
• Results have to be *stored* in memory
• Explicit Load and Store instructions are needed because there are many more variables than the number of registers
Example

• The HLL statements
  
a = b + c
  
d = a + b

• will be “translated” into assembly language as:
  
load b in register rx
load c in register ry
rz <- rx + ry
store rz in a  # not destructive; rz still contains the value of a
rt <- rz + rx
store rt in d
MIPS Information units

- Data types and size:
  - Byte
  - Half-word (2 bytes)
  - Word (4 bytes)
  - Float (4 bytes; single precision format)
  - Double (8 bytes; double-precision format)

- Memory is **byte-addressable**
- A data type must start at an address evenly divisible by its size (in bytes)
- In the little-endian environment (the one we’ll use), the address of a data type is the address of its lowest byte
Big-endian vs. little endian

• Byte order within a word:

<table>
<thead>
<tr>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>Little-endian (we’ll use this)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>Big-endian</td>
</tr>
</tbody>
</table>
Addressing of Information units

- Byte address 0
- Half-word address 0
- Word address 0
- Byte address 8
- Half-word address 8
- Word address 8

- Byte address 2
- Half-word address 2
- Byte address 5
SPIM Convention

Words listed from left to right but little endians within words

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

Byte 7fffebd2  Word 7fffebd4  Half-word 7fffebde
Assembly Language programming or
How to be nice to your TAs

• Use lots of detailed comments
• Don’t be too fancy
• Use lots of detailed comments
• Use words (rather than bytes) whenever possible
• Use lots of detailed comments
• Remember: The address of a word is evenly divisible by 4
• Use lots of detailed comments
• The word following the word at address $i$ is at address $i+4$
• Use lots of detailed comments
MIPS Instruction types

• Few of them (RISC philosophy)
• Arithmetic
  – Integer (signed and unsigned); Floating-point
• Logical and Shift
  – work on bit strings
• Load and Store
  – for various data types (bytes, words,…)
• Compare (of values in registers)
• Branch and jumps (flow of control)
  – Includes procedure/function calls and returns
Notation for SPIM instructions

- Opcode: rd, rs, rt
- Opcode: rt, rs, immed
- where
  - rd is always a destination register (result)
  - rs is always a source register (read-only)
  - rt can be either a source or a destination (depends on the opcode)
  - immed is a 16-bit constant (signed or unsigned)