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
  - *M*icroprocessor without *I*nterlocked *P*ipe *S*tages
- 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 = *R*educed *I*nstruction *S*et *C*omputer
- 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

    add $8,$9,$10 # a = b + c
  - Easy to name (most computers have 32 (integer) registers visible to the programmer and their names are 0, 1, 2, …,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

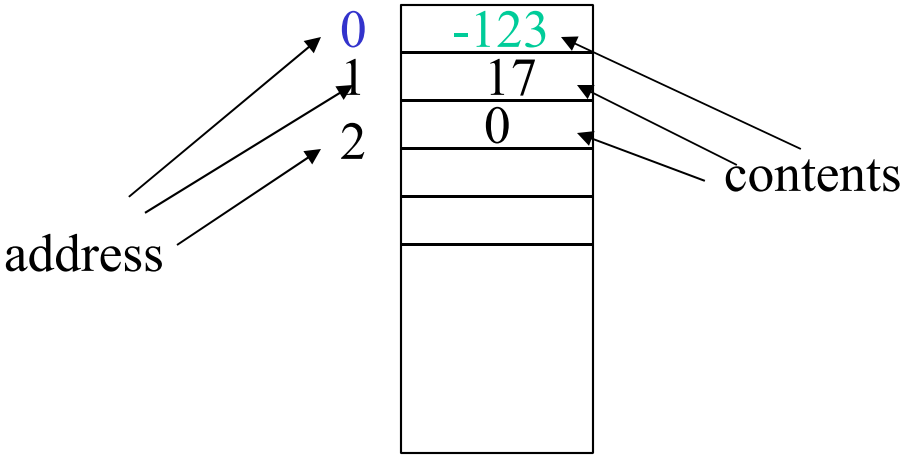
- Memory is a *hierarchy* of devices with faster and more expensive ones closer to CPU
  - Registers
  - Caches (hierarchy: on-chip, off-chip)
  - Main memory (DRAM)
  - Secondary memory (disks)

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

Register	Name	Function	Comment
$0	Zero	Always 0	No-op on write
$1	$at	Reserved for assembler	Don't use it
$2-3	$v0-v1	Expr. Eval/funct. Return	
$4-7	$a0-a3	Proc./func. Call parameters	
$8-15	$t0-t7	Temporaries; volatile	Not saved on proc. Calls
$16-23	$s0-s7	Temporaries	Should be saved on calls
$24-25	$t8-t9	Temporaries; volatile	Not saved on proc. Calls
$26-27	$k0-k1	Reserved for O.S.	Don't use them
$28	$gp	Pointer to global static memory	
$29	$sp	Stack pointer	
$30	$fp	Frame pointer	
$31	$ra	Proc./funct return address	

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:

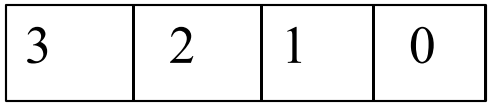
```
load b in register rx
load c in register ry
rz <- rx + ry
store rz in a        # not destructive; rz still contains the value of a
rt <- rz + rx
store rt in d
```

MIPS Information units

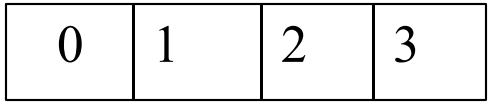
- Data types and size:
  - Byte
  - Half-word (2 bytes)
  - Word (4 bytes)
  - Float (4 bytes; single precision format)
  - Double (8 bytes; double-precision format)
- Memory is **byte-addressable**
- A data type must start at an address evenly divisible by its size (in bytes)
- In the little-endian 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:

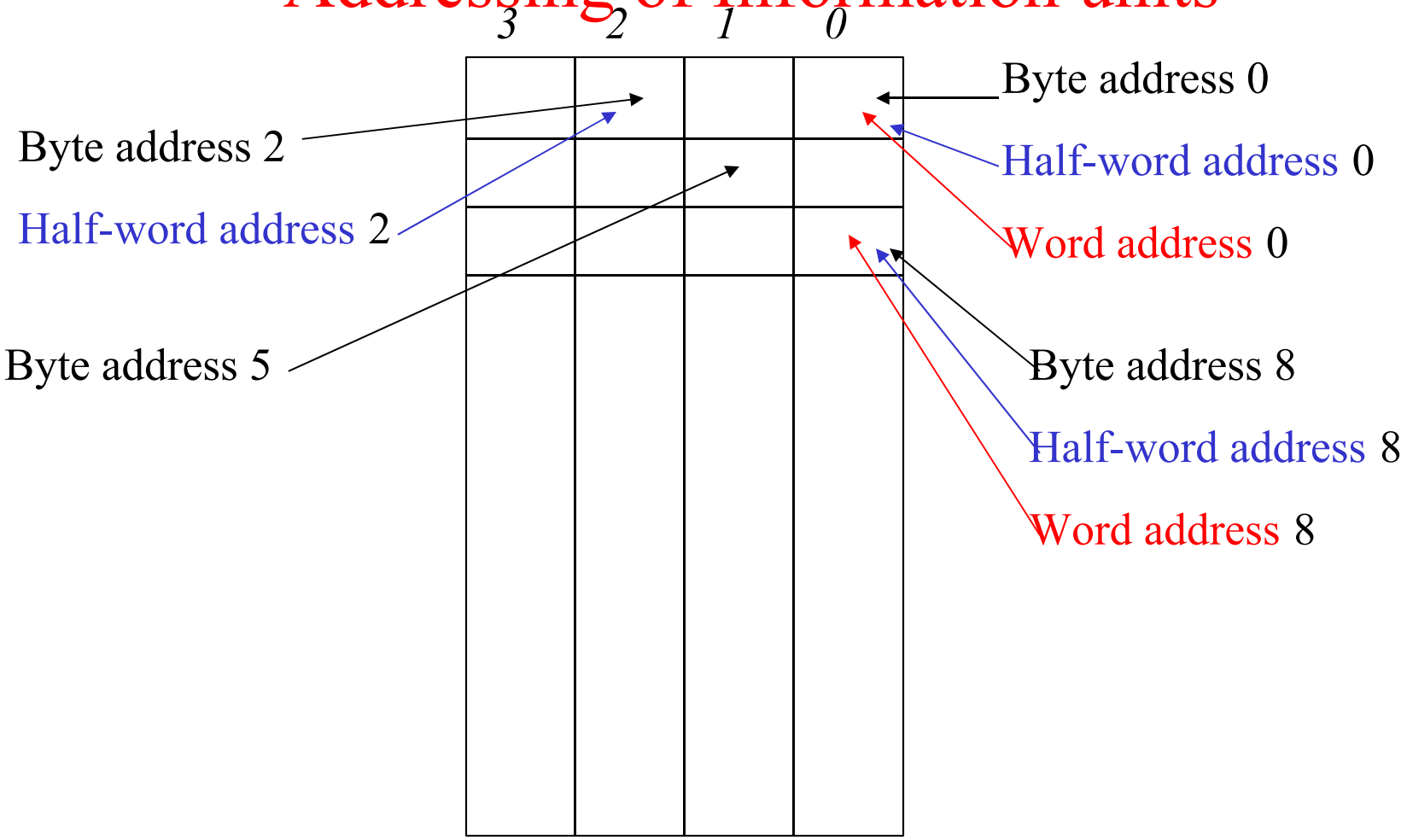


Little-endian
(we'll use this)



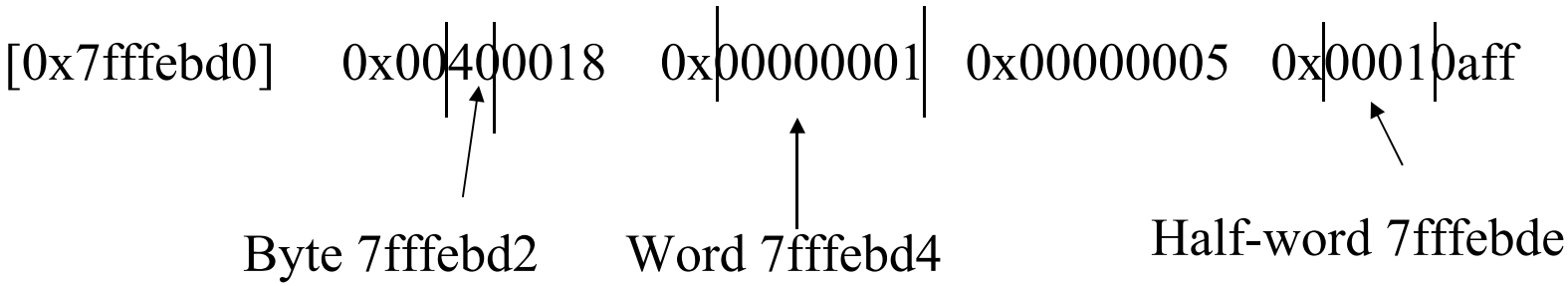
Big-endian

Addressing of Information units



SPIM Convention

Words listed from left to right but little endians within words



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)