MIPS Instruction Set

Dr. Prasenjit Chanak

Department of Computer Science and Engineering
Indian Institute of Technology (BHU), Varanasi
UP, 221005

Register Names and Purpose

Name	Register number	Usage
`$zero`	0	the constant value 0
`$v0-$v1`	2-3	values for results
`$a0-$a3`	4-7	arguments
`$t0-$t7`	8-15	temporaries
`$s0-$s7`	16-23	saved
`$t8-$t9`	24-25	more temporaries
`$gp`	28	global pointer
`$sp`	29	stack pointer
`$fp`	30	frame pointer
`$ra`	31	return address

MIPS Arithmetic

- Simplicity favours regularity
- Operands must be registers, only 32 registers
- provided (smaller is faster)

- Expressions need to be broken

C code	MIPS code
A = B + C + D;	add $t0, $s1, $s2
E = F - A;	add $s0, $t0, $s3 sub $s4, $s5, $s0

Memory Organization

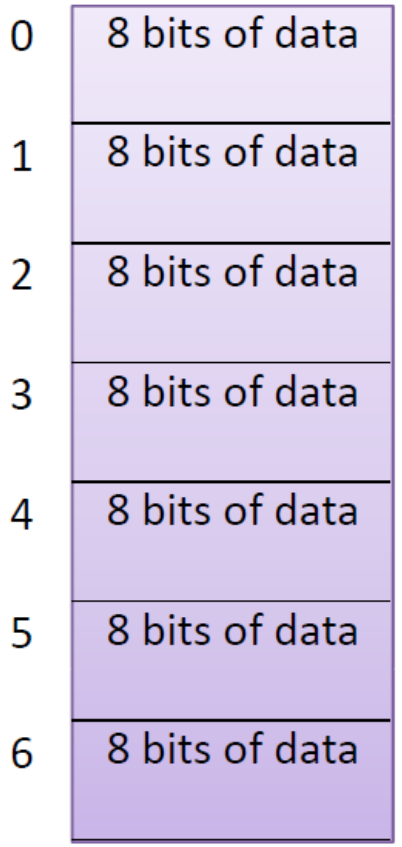
- Programming languages have simple variables that contain single data elements, as in these examples, but they also have more complex data structures—arrays and structures
- These complex data structures can contain many more data elements
- than there are registers in a computer. How can a computer represent and access such large structures?
- The processor can keep only a small amount of data in registers, but computer memory contains billions of data elements. Hence, data structures (arrays and structures) are kept in memory

Memory Organization

- As explained above, arithmetic operations occur only on registers in MIPS instructions; thus, MIPS must include instructions that transfer data between memory and registers. Such instructions are called **data transfer instructions**
- To access a word in memory, the instruction must supply the memory **address**.
- Memory is just a large, single-dimensional array, with the address acting as the index to that array, starting at 0

Memory Organization

- Viewed as a large, single-dimension array. With an address
- A memory address is an index into array
- “Byte addressing” means that the index points to a byte of memory



Words and Bytes

- 2^{32} bytes : byte addresses from 0 to 2^{32}-1
- 2^{30} words : byte addresses 0, 4, 8, ... 2^{32}-4

Big endian byte order

0	1	2	3
4	5	6	7

Little endian byte order

3	2	1	0
7	6	5	4

Non-aligned word

3	2	1	0
7	6	5	4

Memory Organization

- Since 8-bit *bytes* are useful in many programs, virtually all architectures today address individual bytes
- Therefore, the address of a word matches the address of one of the 4 bytes within the word, and addresses of sequential words differ by 4
- In MIPS, words must start at addresses that are multiples of 4
- This requirement is called an **alignment restriction**, and many architectures have it
- **Alignment Restriction:** A requirement that data be aligned in memory on natural boundaries

Memory Organization

- Computers divide into those that use the address of the left most or “big end” byte as the word address versus those that use the rightmost or “little end” byte
- MIPS is in the *big-endian* camp. Since the order matters only if you access the identical data both as a word and as four bytes, few need to be aware of the endianness

Instructions to access memory

- Load and store instructions
- Example:

C code: `A[8] = h + A[8];`

MIPS code:

```
lw   $t0, 32($s3)
add $t0, $s2, $t0
sw   $t0, 32($s3)
```