Instruction Set Architecture

- In computer science, an **instruction set architecture** (**ISA**), also called **computer architecture,** is an abstract model of a computer. A device that executes instructions described by that ISA, such as a central processing unit (CPU), is called an *implementation*.
- In general, an ISA defines the supported instructions, data types, registers, the hardware support for managing main memory, fundamental features (such as the memory consistency, addressing modes, virtual memory), and the input/output model of a family of implementations of the ISA.

Memory Locations and Addresses

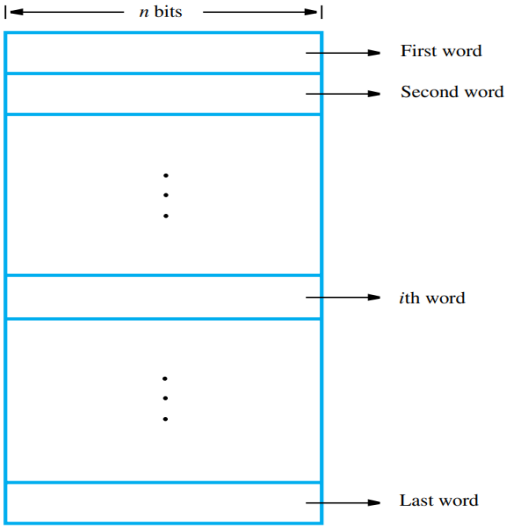
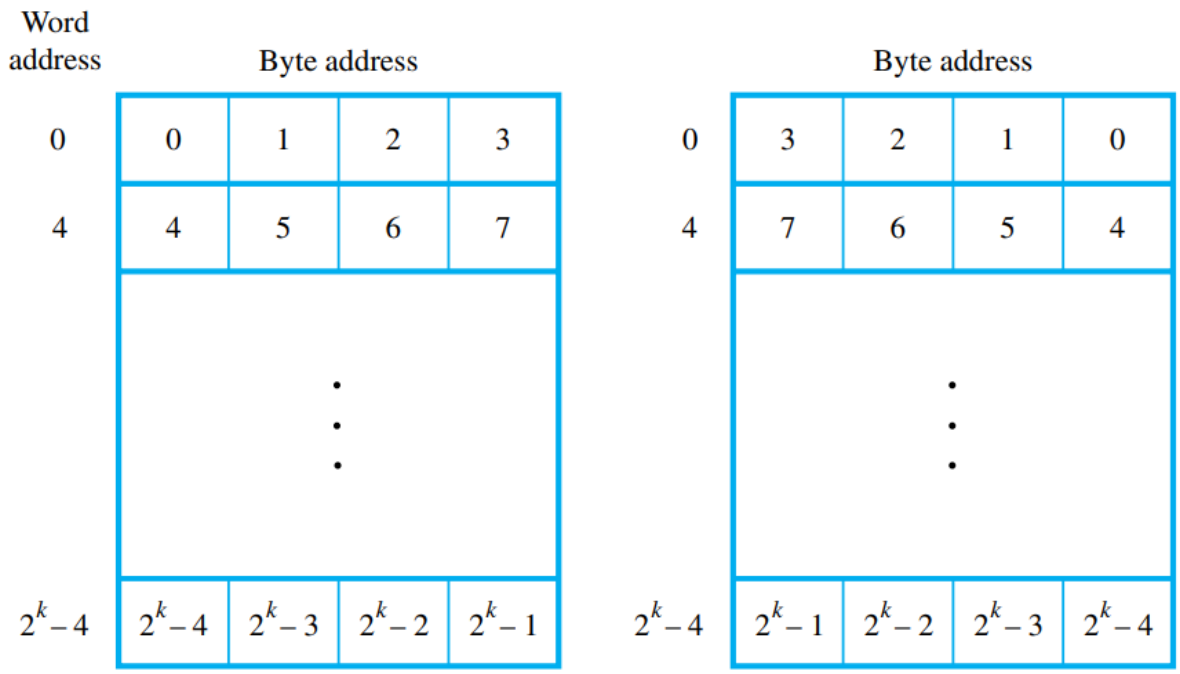


Figure 2.1 Memory words.

- The computer's memory is made of a silicon chip which has millions of storage cell, where each storage cell is capable to store a *bit* of information which value is either 0 or 1.
- The group of n bit is termed as ***word*** where n is termed as the *word length*. The word length of the computer has evolved from 8, 16, 24, 32 to 64 bits. General-purpose computers nowadays have 32 to 64 bits. The group of 8 bit is called a *byte*.
- To access the memory location either you must know the memory location by its unique name or it is required to provide a unique address to each memory location.

Big-Endian and Little-Endian Assignments in Byte Addresses



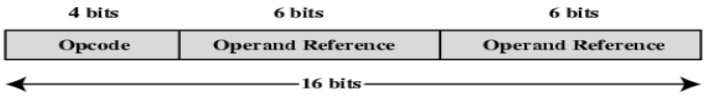
(a) Big-endian assignment (b) Little-endian assignment

In the **big-endian assignmen**t, the lower byte addresses are used for the leftmost bytes of the word. Observe the word 0 in the image below, the leftmost bytes of the word have lower byte addresses.
In the **little-endian assignment,** the lower byte addresses are used for the rightmost bytes of the word. Observe the word 0 in the image below the rightmost bytes of word 0 has lower byte addresses.

Word Alignment

- Word locations have aligned addresses if they begin at a byte address that is a multiple of the number of bytes in a word.
- For practical reasons associated with manipulating binary-coded addresses, the number of bytes in a word is a power of 2.
  Hence, if the word length is 16 (2 bytes), aligned words begin at byte addresses 0, 2, 4,..., and for a word length of 64 (8 bytes),aligned words begin at byte addresses 0, 8, 16.

Instruction



Machine language is built up from discrete *statements* or *instructions*. On the processing architecture, a given instruction may specify:

- opcode (the instruction to be performed) e.g. add, copy, test
- any explicit operands:
  - registers
  - literal/constant values
  - addressing modes used to access memory

Arithmetic Instructions

Instruction	Example	Meaning	Comments
add	`add $1,$2,$3`	$1=$2+$3	
subtract	`sub $1,$2,$3`	$1=$2-$3	
add immediate	`addi $1,$2,100`	$1=$2+100	"Immediate" means a constant number
add unsigned	`addu $1,$2,$3`	$1=$2+$3	Values are treated as unsigned integers, not two's complement integers
subtract unsigned	`subu $1,$2,$3`	$1=$2-$3	Values are treated as unsigned integers, not two's complement integers

Logical

Instruction	Example	Meaning	Comments
and	`and $1,$2,$3`	$1=$2&$3	Bitwise AND
or	`or $1,$2,$3`	$1=$2|$3	Bitwise OR
and immediate	`andi $1,$2,100`	$1=$2&100	Bitwise AND with immediate value
or immediate	`or $1,$2,100`	$1=$2|100	Bitwise OR with immediate value
shift left logical	`sll $1,$2,10`	$1=$2<<10	Shift left by constant number of bits
shift right logical	`srl $1,$2,10`	$1=$2>>10	Shift right by constant number of bits

Data Transfer

Instruction	Example	Meaning	Comments
load word	`lw $1,100($2)`	$1=Memory[$2+100]	Copy from memory to register
store word	`sw $1,100($2)`	Memory[$2+100]=$1	Copy from register to memory
load upper immediate	`lui $1,100`	$1=100x2^16	Load constant into upper 16 bits. Lower 16 bits are set to zero.
load address	`la $1,label`	$1=Address of label	*Pseudo-instruction* (provided by assembler, not processor!) Loads computed address of label (not its contents) into register
load immediate	`li $1,100`	$1=100	*Pseudo-instruction* (provided by assembler, not processor!) Loads immediate value into register