UNIVERSIDAD POLITECNICA DE VALENCIA

etsinf

DISCA

Fundamentos de computadores

Subject 6. Assembler

Aims

- Study the fundamental concepts for programming in assembly language of the MIPS R2000 processor
- Know the basic structure of a processor, which allows the execution of different instructions
- Know the elements of a program (data and instructions) written in assembler language and understand its location in memory
- Know the basic set of instructions of a current processor
- Understand MIPS 2000 assembler language instruction coding

Outline

- Introduction

- MIPS R2000
  - Basic concepts
  - Instruction Set
    - Arithmetic instructions
    - Logic instructions
    - Load and Store instructions
    - Pseudo-instructions li & la
    - Comparison instructions
    - Conditional statements
  - Coding of instructions

- Relation with others courses
  - Computer Languages
  - Operating Systems
  - Advanced processing
  - Multiprocessors & Supercomputers

Bibliografía

- MIPS Assembly Language Programming. Robert Briton
- See MIPS Run. Dominic Sweetman
- Introducción a los computadores. Julio Sahuquillo y otros

Outline

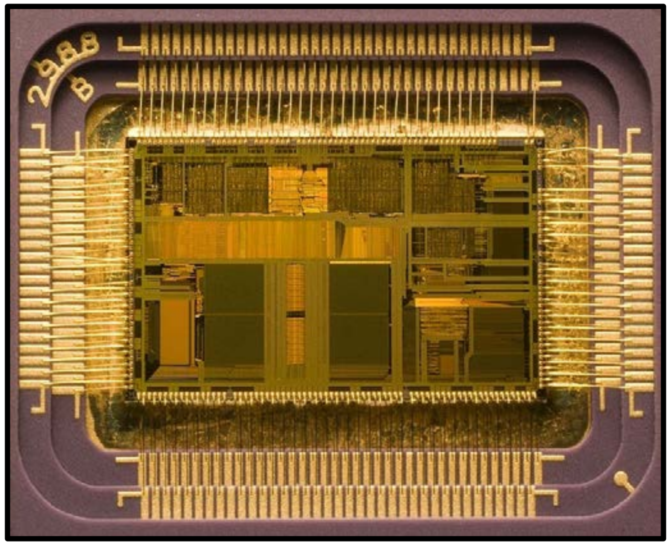
- Introduction
  - Functional Units
  - Stages in the execution of an instruction
  - Concept of ARCHITECTURE
- MIPS R2000
  - Basic concepts
  - Instruction Set
    - Arithmetic instructions
    - Logic instructions
    - Load and Store instructions
    - Pseudo-instructions li & la
    - Comparison instructions
    - Conditional statements
  - Coding of Instructions
- Relation with others courses
  - Computer Languages
  - Operating Systems
  - Advanced processing
  - Multiprocessors & Supercomputers

Functional Units of a Computer

- Functional Unit
  - **A set of circuits that perform a specific function differentiated from other functions**
- Basic funtional Units of a Computer
  - Central processing unit (UCP, CPU)
  - Memory unit
  - Input/Output unit (E/S, I/O)

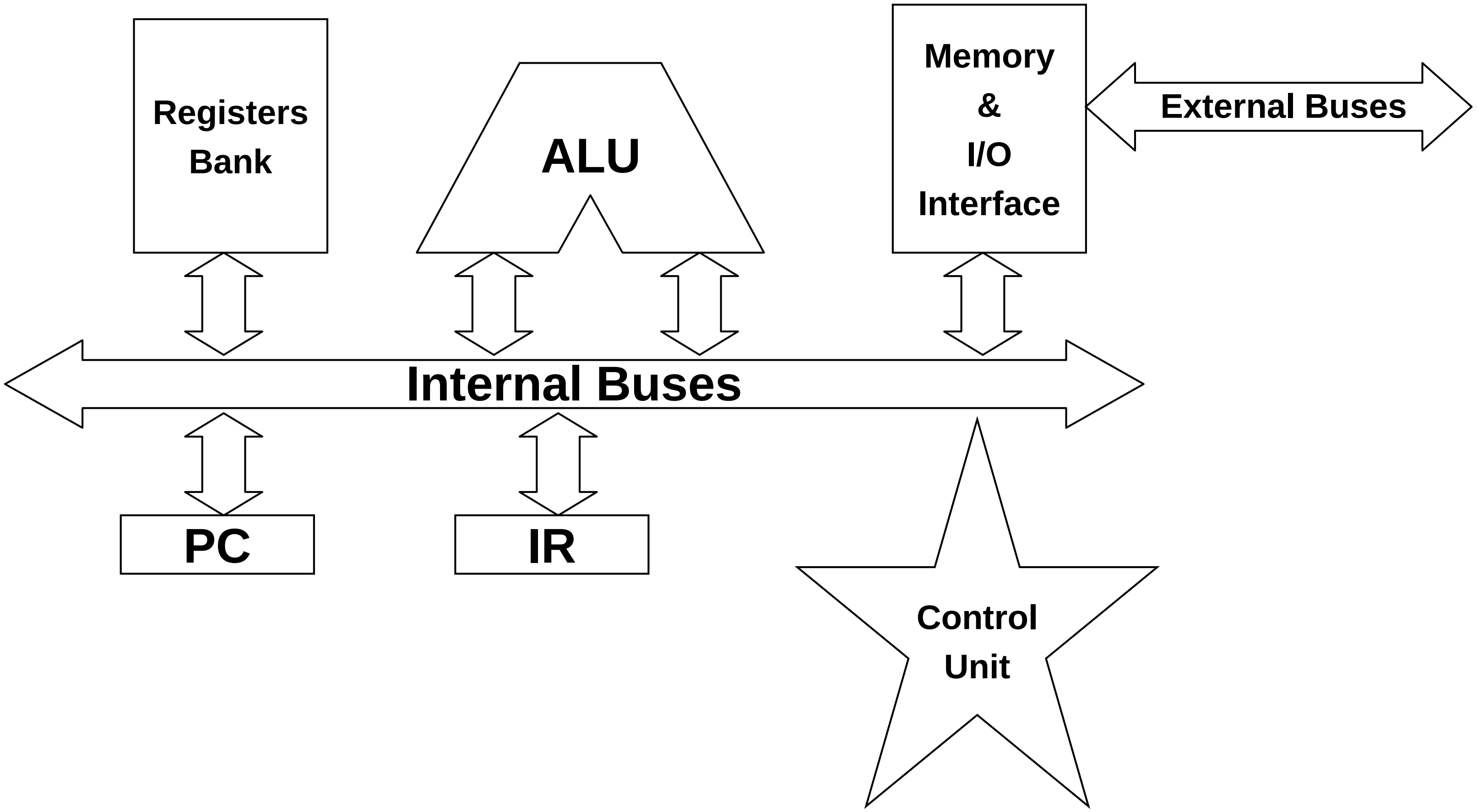
Functional Units of a Computer

- Central Processing Unit (CPU)
  - It is the component in a computer, which interprets the instructions and processes the data contained in the programs.
  - In other words, it executes the instructions



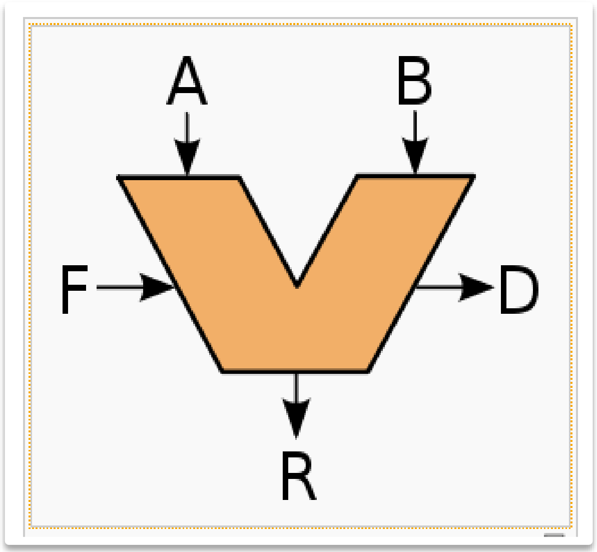
Functional Units of a Computer

- Central Processing Unit (CPU)



Functional Units of a Computer

- Arithmetic Logic Unit (ALU), performs arithmetic operations (addition, substraction, multiplication, etc.) & logic operations (equal to, less than, greather than, etc.)
  - A y B → Operands
  - F → Operation to be carried out
  - R → Result of the operation
  - D → State flags



Functional Units of a Computer

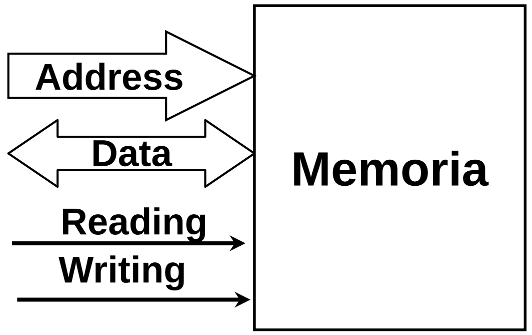
- Registers Bank
  - Used to store data temporarily
  - Memory with a very short access time
  - It is always inside the processor
  - The number of registers is between 8 & 512, it depends on the processor
  - The MIPS R2000 has 32 registers of 32 bits ($0 .. $31)
  - 1 writing port & 1 o 2 Reading ports
- Special registers
  - Program counter (PC) Contains the memory address of the next instruction to be executed.
  - Instruction register (IR) Contains the code of the instruction being executed.

Control of instructions execution

- Control Unit
  - Generates the signals needed by the CPU in order to execute the instructions properly
  - Indicates the type of operation that the ALU has to perform
  - Indicates which registers contain the data and where to store the result
  - Generates load signals of all registers when they have to store information
  - It is a sequential system. Its complexity depends on the complexity of the CPU and on the number and type of instructions to execute

Functional Units of a Computer

- Memory: Storage device
  (data + instructions)

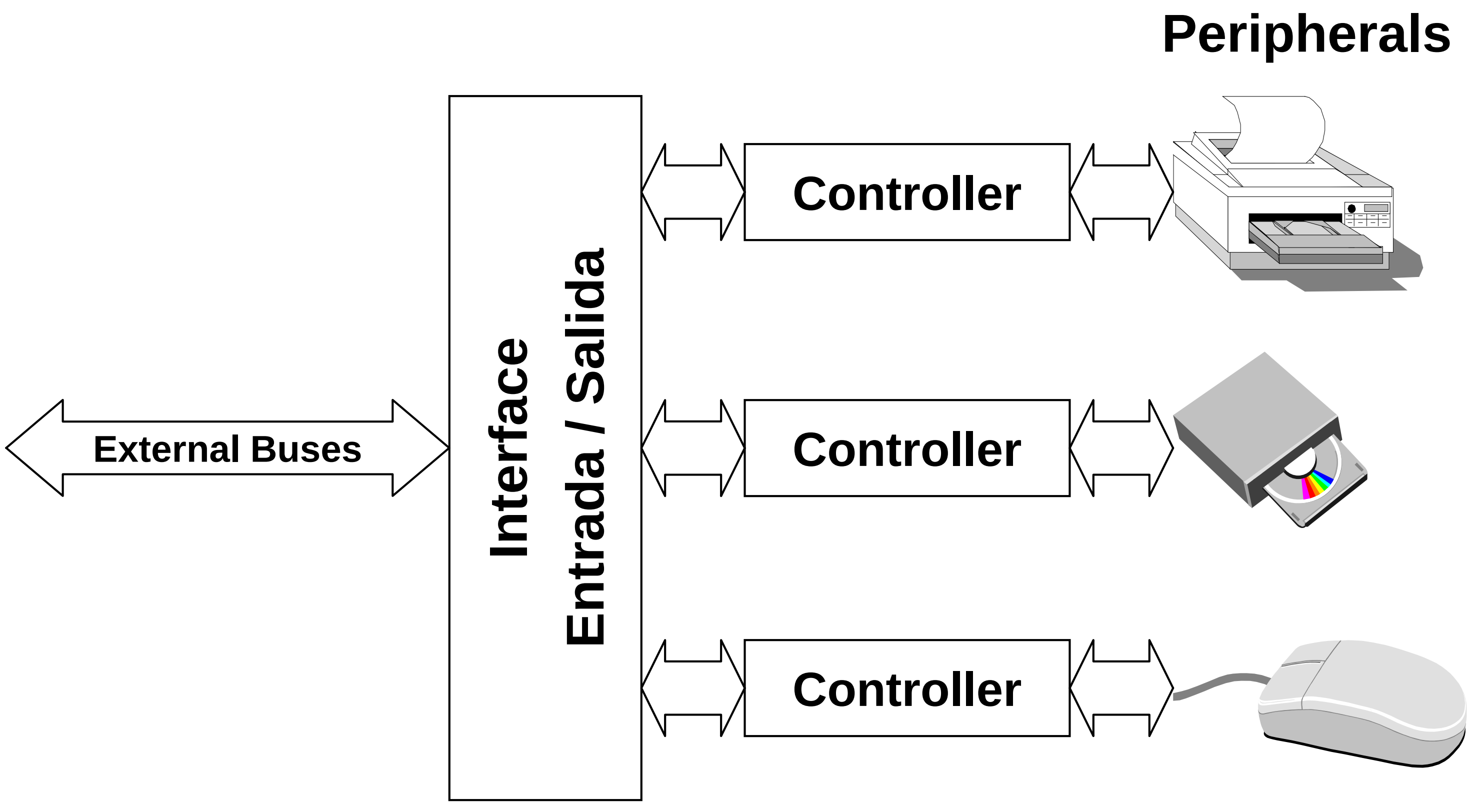


Units

- **1K (kilo) = 2^{10} = 1024**
- **1M (mega) = 2^{10}K = 2^{20}**
- **1G (giga) = 2^{10}M = 2^{20}K = 2^{30}**
- **1T (tera) = 2^{10}G = 2^{20}M = 2^{30}K = 2^{40}**
- **1P (peta) = 2^{10}T = 2^{20}G = 2^{30}M = 2^{40}K =2^{50}**
- **1E (exa) = 2^{10}P = 2^{20}T = 2^{30}G = 2^{40}M = 2^{50}K =2^{60}**

Functional Units of a Computer

- Input/Output System
  - It allows the communication of the CPU-memory system with the outside



Functional Units of a Computer

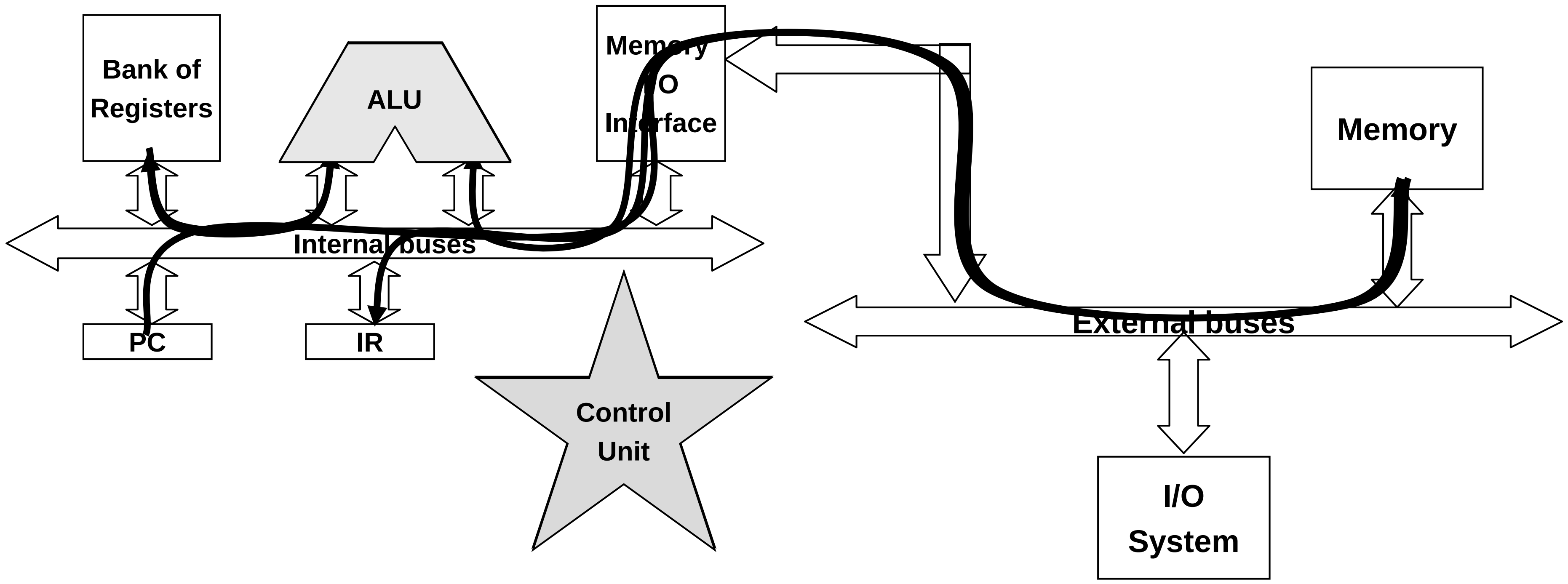
- Peripheral
  - Device that allows communication with the outside
    - Secondary memory (storage)
    - Data Input/Output (communication)
- Interface or controller
  - Hardware / software device that allows communication between the CPU (or the memory system) and the peripheral

Outline

- Introduction
  - Functional Units
  - Stages in the execution of an instruction
  - Concept of ARCHITECTURE
- MIPS R2000
  - Basic concepts
  - Instruction Set
    - Arithmetic instructions
    - Logic instructions
    - Load and Store instructions
    - Pseudo-instructions li & la
    - Comparison instructions
    - Conditional statements
  - Coding of Instructions
- Relation with others courses
  - Computer Languages
  - Operating Systems
  - Advanced processing
  - Multiprocessors & Supercomputers

Stages in the execution of an instruction

- Instruction Fetch (IF)
- Instruction decoding (ID)
- Operands fetch (OF)
- Operation execution
- Result storage



Outline

- Introduction
  - Functional Units
  - Stages in the execution of an instruction
  - Concept of ARCHITECTURE
- MIPS R2000
  - Basic concepts
  - Instruction Set
    - Arithmetic instructions
    - Logic instructions
    - Load and Store instructions
    - Pseudo-instructions li & la
    - Comparison instructions
    - Conditional statements
  - Coding of Instructions
- Relation with others courses
  - Computer Languages
  - Operating Systems
  - Advanced processing
  - Multiprocessors & Supercomputers

Instructions set architecture

- The Instruction Set Architecture (ISA) is the part of the processor that is visible to the programmer or compiler writer.
  - The ISA serves as the boundary between software and hardware.
- The ISA is determined by:
  - Set of registers
  - Data types
  - The addressable space of memory (organization of the memory)
  - Set of instructions
  - Addressing mode(Operands location)

Programs in assembly language

- Assembly language(asm):
  - Low level programming language
  - It consists of a set of mnemonics that represent instructions that the processor can execute.
  - In general, one instruction in assembly Language has a corresponding machine instruction.
  - Programs written in assembly Language are translated to machine code language using a special program called assembler programs.
  - The machine language or machine code is a set of instructions with its data encoded in binary, which can understand and execute a processor.
  - Both assembler language and machine code are specific to each processor model.

Outline

- Introduction
  - Functional Units
  - Stages in the execution of an instruction
  - Concept of ARCHITECTURE
- MIPS R2000
  - Basic concepts
  - Instruction Set
    - Arithmetic instructions
    - Logic instructions
    - Load and Store instructions
    - Pseudo-instructions li & la
    - Comparison instructions
    - Conditional statements
  - Coding of Instructions
- Relation with others courses
  - Computer Languages
  - Operating Systems
  - Advanced processing
  - Multiprocessors & Supercomputers

Outline

FCO

- Introducción
- **MIPS R2000**
  - Basic concepts
    - Bank of registers
    - Data Types
    - Arithmetic and Logic Unit
    - Memory
    - Structure of a program
  - Instruction Set
    - Arithmetic instructions
    - Logic instructions
    - Load and Store instructions
    - Pseudo-instructions li & la
    - Comparison instructions
    - Conditional statements
  - Coding of Instructions
- Relation with others courses
  - Computer Languages
  - Operating Systems
  - Advanced processing
  - Multiprocessors & Supercomputers

MIPS R2000: Bank of registers (BR)

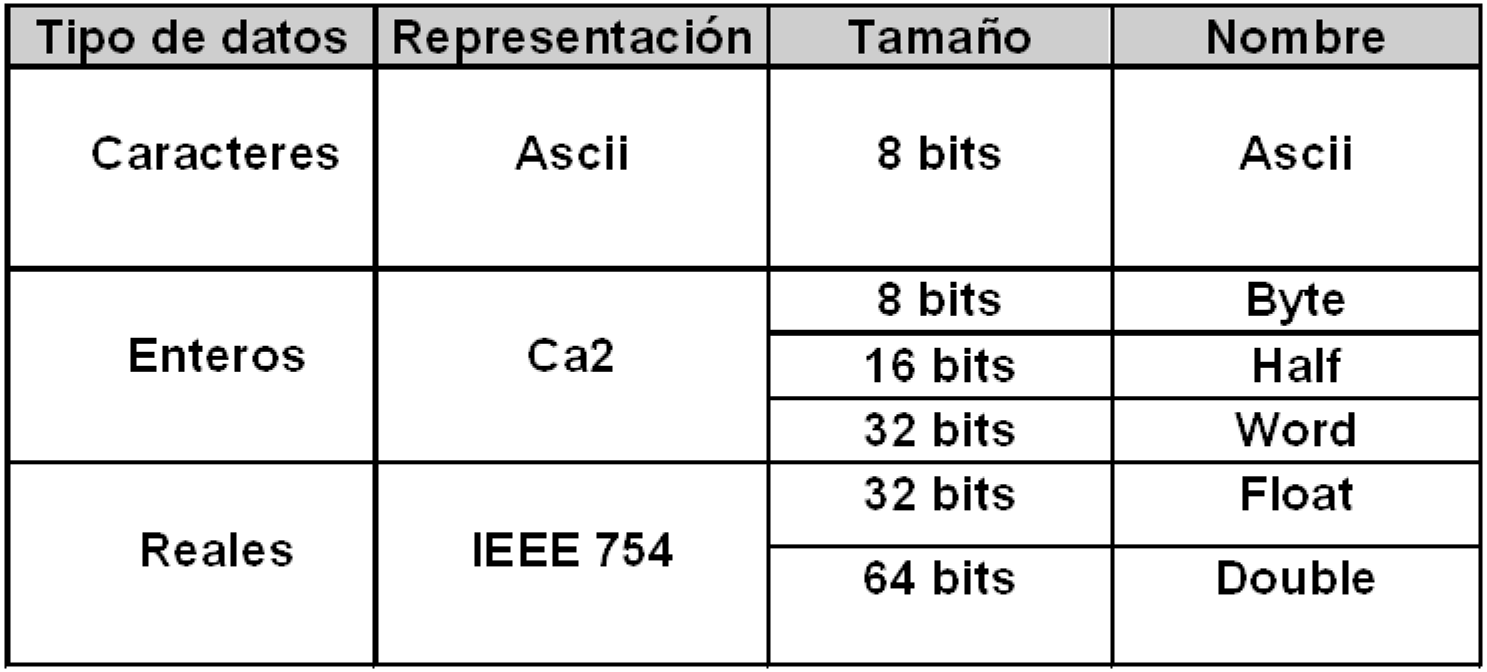
- Main characteristics of the MIPS R2000 bank of registers
  - Integer Bank of registers:
    - 32 general purpose Registers of 32 bits
      - Identified as \$0,\$1,……, \$31
      - Example: add \$2, \$3, \$4 → (\$2←\$3+\$4)
    - 2 special registers of 32 bits (HI y LO).
      - Used to store the result of multiplication and division operations
  - Floatin point Bank of registers
    - 32 Registers of 32 bits. Format IEEE 754 simple precision. Identified as: \$f0, \$f1, \$f2, ……., \$f31
    - 16 Registers of 64 bits. Format IEEE 754 double precision. Identified as: \$f0, \$f2, \$f4, …, \$f30
- The register \$0 has the value 0. It is cabled to ground.

Outline

- Introducción
- **MIPS R2000**
  - Basic concepts
    - Bank of registers
    - Data Types
    - Arithmetic and Logic Unit
    - Memory
    - Structure of a program
  - Instuction set
    - Arithmetic instructions
    - Logic instructions
    - Load and Store instructions
    - Pseudo-instructions li & la
    - Comparing instructions
    - Conditional statements
  - Instruction coding
- Relation with others courses
  - Computer Languages
  - Operating Systems
  - Advanced processing
  - Multiprocessors & Supercomputers

MIPS R2000: Data types & data representation FCO

- Data types supported by the MIPS R2000



Tipo de datos	Representación	Tamaño	Nombre
Caracteres	Ascii	8 bits	Ascii
Enteros	Ca2	8 bits	Byte
		16 bits	Half
		32 bits	Word
Reales	IEEE 754	32 bits	Float
		64 bits	Double

- Float : Real numbers represented using the standard IEEE 754 of single precision
- Double: Real numbers represented using the standard IEEE 754 of double precision

Outline

FCO

- Introducción
- MIPS R2000
  - Basic concepts
    - Bank of registers
    - Data Types
    - Arithmetic and Logic Unit
    - Memory
    - Structure of a program
  - Instuction set
    - Arithmetic instructions
    - Logic instructions
    - Load and Store instructions
    - Pseudo-instructions li & la
    - Comparing instructions
    - Conditional statements
  - Instruction coding
- Relation with others courses
  - Computer Languages
  - Operating Systems
  - Advanced processing
  - Multiprocessors & Supercomputers

MIPS R2000: Arithmetic and Logic Unit

- Main characteristics of the ALU
  - Responsible of execute the arithmetic and logic operations.
  - The input data can be integers represented in complement to two or natural binary (CBN)
  - One of the operands can be in the instruction (RI) as an immediate data of 16 bits
  - The registers HI and LO are used to store the results of multiplication and división operations.

Outline

- Introducción
- **MIPS R2000**
  - Basic concepts
    - Bank of registers
    - Data Types
    - Arithmetic and Logic Unit
    - Memory
    - Structure of a program
  - Instuction set
    - Arithmetic instructions
    - Logic instructions
    - Load and Store instructions
    - Pseudo-instructions li & la
    - Comparing instructions
    - Conditional statements
  - Instruction coding
- Relation with others courses
  - Computer Languages
  - Operating Systems
  - Advanced processing
  - Multiprocessors & Supercomputers

MIPS R2000: Memory organization

- **Addresable space of the MIPS R2000**

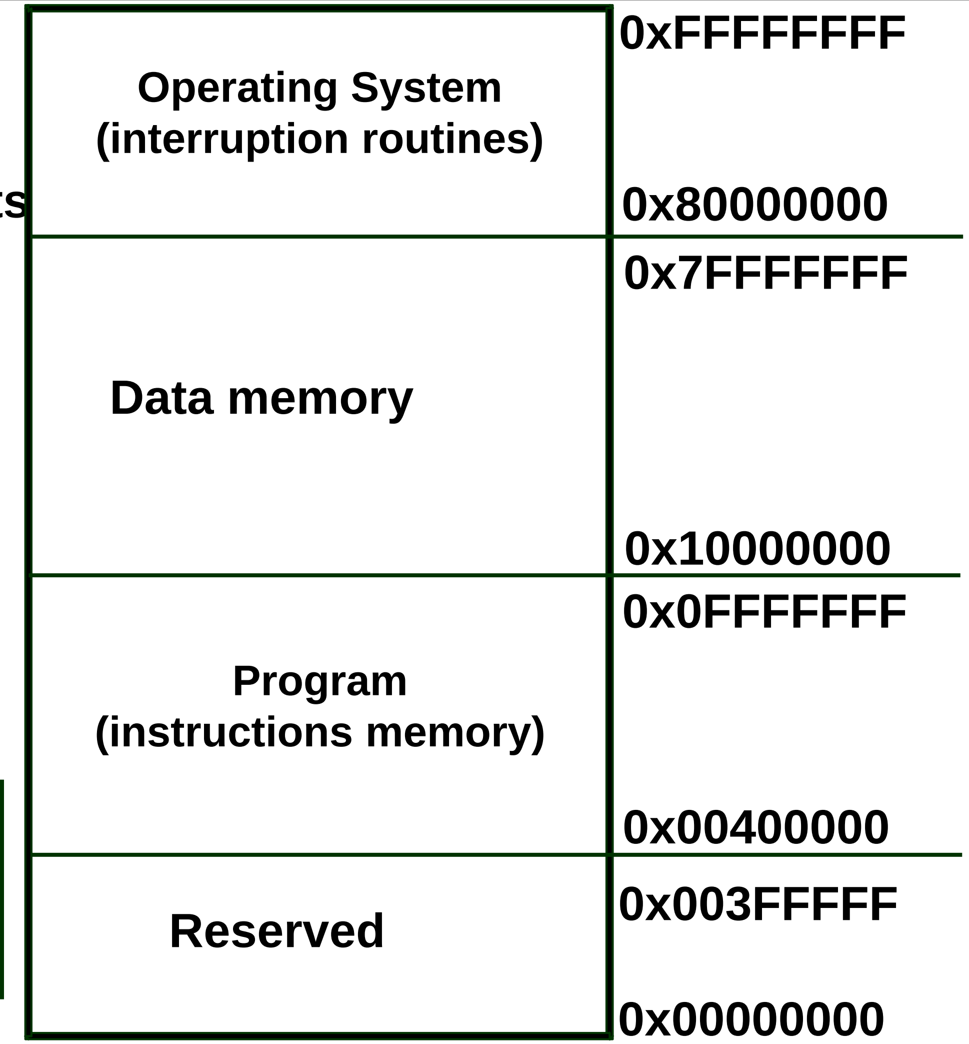
– **The width of the address bus is 32 bits**

– **4 Gbytes (4G x 8bits) =**
2 Ghalf = (2G x 16bits) =
1 Gword = (1G x 32bits).

– **2^{32} bytes addressed from**
0 to $2^{32} – 1$

– **2^{30} words (4 bytes) addesses: 0, 4, 8, ..., 2^{32} - 4**

The user can use the memory addesses in the range: [0x00400000, 0x7FFFFFFF]

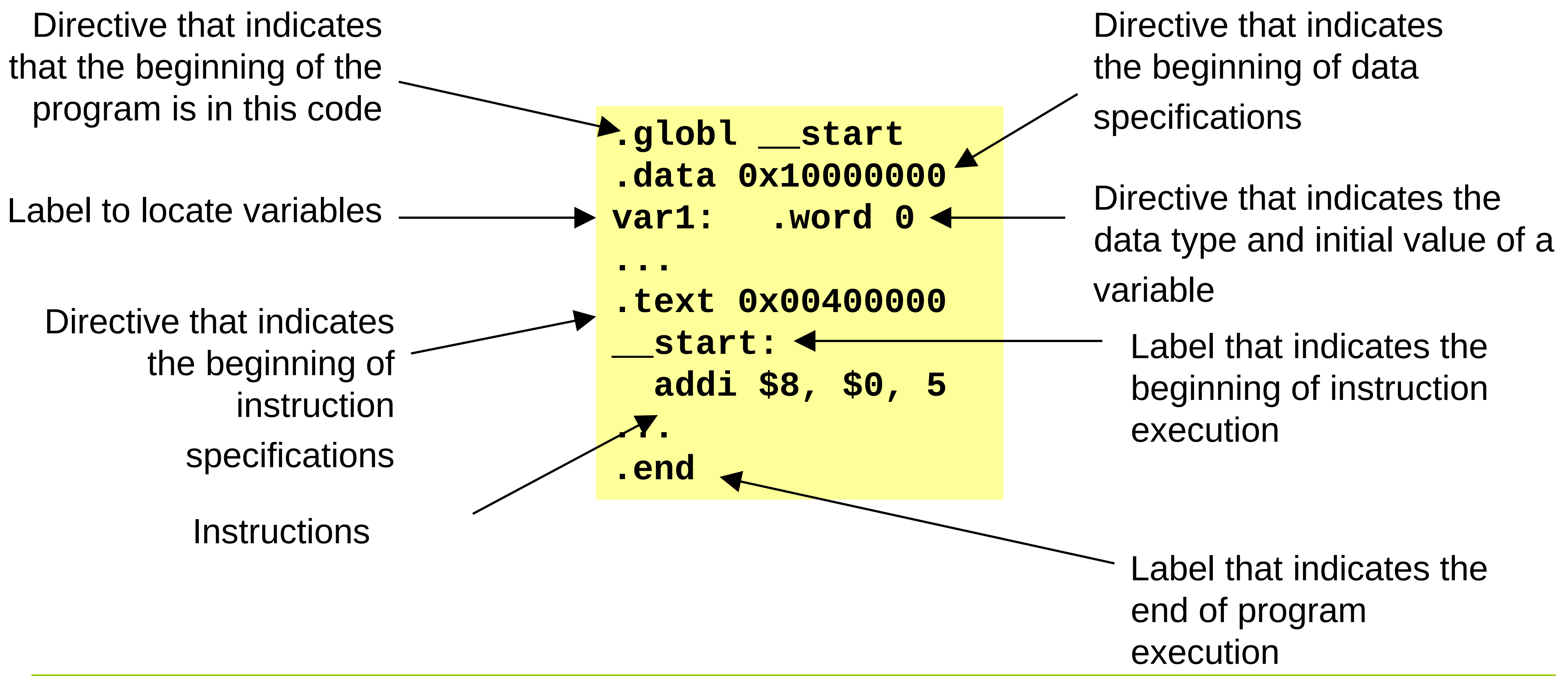


Outline

- Introducción
- **MIPS R2000**
  - Basic concepts
    - Bank of registers
    - Data Types
    - Arithmetic and Logic Unit
    - Memory
    - Structure of a program
  - Instuction set
    - Arithmetic instructions
    - Logic instructions
    - Load and Store instructions
    - Pseudo-instructions li & la
    - Comparing instructions
    - Conditional statements
  - Coding of Instructions
- Relation with others courses
  - Computer Languages
  - Operating Systems
  - Advanced processing
  - Multiprocessors & Supercomputers

Structure of a program

- Structure of a program written in assembler of the MIPS R2000



```
.globl __start
.data 0x10000000
var1:   .word 0
...
.text 0x00400000
__start:
  addi $8, $0, 5
...
.end
```

Directive that indicates that the beginning of the program is in this code

Label to locate variables

Directive that indicates the beginning of instruction specifications

Instructions

Directive that indicates the beginning of data specifications

Directive that indicates the data type and initial value of a variable

Label that indicates the beginning of instruction execution

Label that indicates the end of program execution

Structure of a program

- The assembler directives are used to locate data and instructions in the memory of the processor
  - Their names begin with a period and brackets are used to indicate that one or more arguments are optional

    .directory_name argument1 [, argument2]
  - They are grouped into three types:
    - Directives for allocation memory space
    - Directives to indicate the starting address space of data or the starting address space of instructions
    - General purpose directives
  - They are NOT instructions, they do not occupy memory.

Programs in assembler

- ".data" directive
  - Sintaxis: ".data address"
  - Locates data specified by data directives from the specified address
  - If the address is not it is used the address 0x10000000
- ".text" directive
  - Sintaxis ".text address"
  - Locates instructions from the specified address
  - If the address is not it is used the address 0x00400000
- ".end" directive
  - Sintaxis ".end"
  - Indicates the end of the program

Structure of a program

- Labels
  - Sintaxis: “etiqueta:”
  - They are used to point an important memory position, for instance: memory locations used to store variables or memory locations used in jump instructions, ….
- “__start”
  - Label indicating the start of the program. In other words, the first instruction to be executed when the program starts its execution.
  - Ina program, only one can exist an“__start” Label. For that purpouse itn is used the directive“.globl etiqueta”. The .globl indicates that the label is common to all the program.

Outline

- Introducción
- **MIPS R2000**
  - Basic concepts
    - Bank of registers
    - Data Types
    - Arithmetic and Logic Unit
    - Memory
    - Structure of a program
  - Instuction set
    - Arithmetic instructions
    - Logic instructions
    - Load and Store instructions
    - Pseudo-instructions li & la
    - Comparing instructions
    - Conditional statements
  - Instruction coding
- Relation with others courses
  - Computer Languages
  - Operating Systems
  - Advanced processing
  - Multiprocessors & Supercomputers

Instructions set

- Instructions
  - Computer language
  - The instruction set under study is very similar to the instruction set used in Sony and Nintendo
- Each instruction in assembler language has a format. The instruction is coded in machine language (strings of 0's and 1's)
- Instructions are grouped in:
  - Arithmetic Instructions
  - Logical Instructions
  - Store and load Instructions
  - Movement Instructions
  - Comparison Instructions
  - Conditional jump Instructions
  - Unconditional jump Instructions

Outline

FCO

- Introducción
- MIPS R2000
  - Basic concepts
    - Bank of registers
    - Data Types
    - Arithmetic and Logic Unit
    - Memory
    - Structure of a program
  - Instuction set
    - Arithmetic instructions
    - Logic instructions
    - Load and Store instructions
    - Pseudo-instructions li & la
    - Comparing instructions
    - Conditional statements
  - Instruction coding
- Relation with others courses
  - Computer Languages
  - Operating Systems
  - Advanced processing
  - Multiprocessors & Supercomputers

Arithmetic instructions

- Arithmetic instructions

Sintaxis	Format	Description
add rd, rs, rt	R	rd ← rs + rt
addi rt, rs, inm	I	rt ← rs + inm (immediato of 16 bits represened in two's complement)
sub rd, rs, rt	R	rd ← rs – rt
mult rs, rt	R	Multiply rs by rt the 32 least signifiant bits are stored in the register LO and the 32 most signifiant bits are stored in the register HI
div rs, rt	R	Divide rs by rt the cotient is stored in the register LO abd the reminder is stored in the register HI

- Data movement instructions

Sintaxis	Formato	Descripción
mfhi rd	R	rd ← HI
mflo rd	R	rd ← LO
mthi rs	R	HI ← rs
mtlo rs	R	LO ← rs

Arithmetic instructions

Adding and subtracting

```
.globl __start
.text 0x00400000
__start:
  addi $8, $0, 5
  addi $9, $0, 6
  add $10, $8, $9
  sub $11, $8, $9
.end
```

Multiplication

```
.globl __start
.text 0x00400000
__start:
  addi $8, $0, 2
  addi $9, $0, 3
  mult $8, $9
  mflo $10
  mfhi $11
.end
```

Division

```
.globl __start
.text 0x00400000
__start:
  addi $8, $0, 4
  addi $9, $0, 11
  div $9, $8
  mflo $10
  mfhi $11
.end
```

- What is the content of registers $10 and $11 after the execution of each segment of code?
- What is the content of registers $10 and $11 if the following instructions are replaced?

Replace `sub $11, $8, $9` **by** `sub $11, $9, $8`

Replace `div $9, $8` **by** `div $8, $9`

Outline

- Introducción
- **MIPS R2000**
  - Basic concepts
    - Bank of registers
    - Data Types
    - Arithmetic and Logic Unit
    - Memory
    - Structure of a program
  - Instuction set
    - Arithmetic instructions
    - Logic instructions
    - Load and Store instructions
    - Pseudo-instructions li & la
    - Comparison instructions
    - Conditional statements
  - Coding of Instructions
- Relation with others courses
  - Computer Languages
  - Operating Systems
  - Advanced processing
  - Multiprocessors & Supercomputers

Logic Instructions

Sintaxis	Format	Description
and rd, rs, rt	R	rd ← rs and rt, the logic operation indicated is performed bit by bit
nor rd, rs, rt	R	rd ← rs nor rt
xor rd, rs, rt	R	rd ← rs xor rt
or rd, rs, rt	R	rd ← rs or rt
andi rt, rs, inm	I	rt← rs and imm, (the immediate data is of 16 bits and extended with 16 zeros)
ori rt, rs, inm	I	rt← rs or inm
xori rt, rs, inm	I	rt← rs xor inm
sll rd, rt, disp	R	rd← rt << disp, shift to the left, stuffing with zero
srl rd, rt, disp	R	rd← rt >> disp, shift to the right, stuffing with zero

Logic Instructions

or & and

```
.globl __start
.text 0x00400000
__start:
  ori $8, $0, 0x000a
  ori $9, $0, 0x000c
  or $10, $8, $9
  and $11, $8, $9
.end
```

nor & xor

```
.globl __start
.text 0x00400000
__start:
  ori $8, $0, 0x000a
  ori $9, $0, 0x000c
  nor $10, $8, $9
  xor $11, $8, $9
.end
```

- Operations are performed bit by bit
- What is the content of registers $10 y $11 after the execution of each segment of code?
- How can be performed the NOT operation?
- How can be performed the NAND operation?

Logic Instructions

Shift to the left

```
.globl __start
.text 0x00400000
__start:
  addi $8, $0, 6
  sll $9, $8, 1
  sll $10, $8, 2
.end
```

Shift to the right

```
.globl __start
.text 0x00400000
__start:
  addi $8, $0, 6
  srl $9, $8, 1
  srl $10, $8, 2
.end
```

- What is the content of registers $9 y $10 after the execution of each segment of code?
- What arithmetic operation is implemented when it is performed the shift to the left operation?
- What arithmetic operation is implemented when it is performed the shift to the right operation?

Outline

- Introducción
- **MIPS R2000**
  - Basic concepts
    - Bank of registers
    - Data Types
    - Arithmetic and Logic Unit
    - Memory
    - Structure of a program
  - Instuction set
    - Arithmetic instructions
    - Logic instructions
    - Load and Store instructions
    - Pseudo-instructions li & la
    - Comparison instructions
    - Conditional statements
  - Instruction coding
- Relation with others courses
  - Computer Languages
  - Operating Systems
  - Advanced processing
  - Multiprocessors & Supercomputers

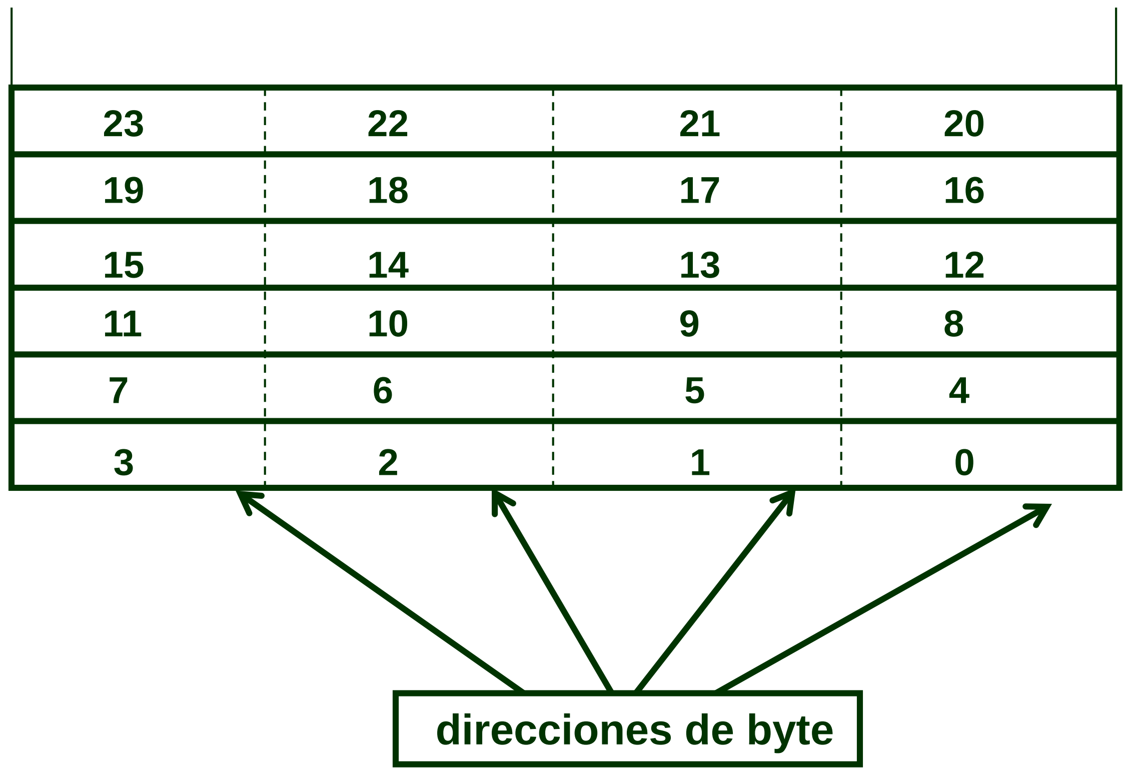
MIPS R2000: Organization of the memory

- Memory. Can be seen as a unidimensional vector.
  - Memory address. Index of the vector.
  - Memory is addressed byte by byte.
    - Indexes point to each memory.
    - The byte is the minimal addressable unit

Address	**Content**
3	1 data byte
2	1 data byte
1	1 data byte
0	1 data byte

MIPS R2000: Organización de la memoria

- **Acceso por byte (BYTE)**
  - 1 Byte en cualquier dirección



MIPS R2000: Organization of the memory

- Memory. Can be seen as a unidimensional vector.
  - Memory address. Index of the vector.
  - Memory is addressed byte by byte.
    - Indexes point to each memory.
    - The byte is the addressable unit

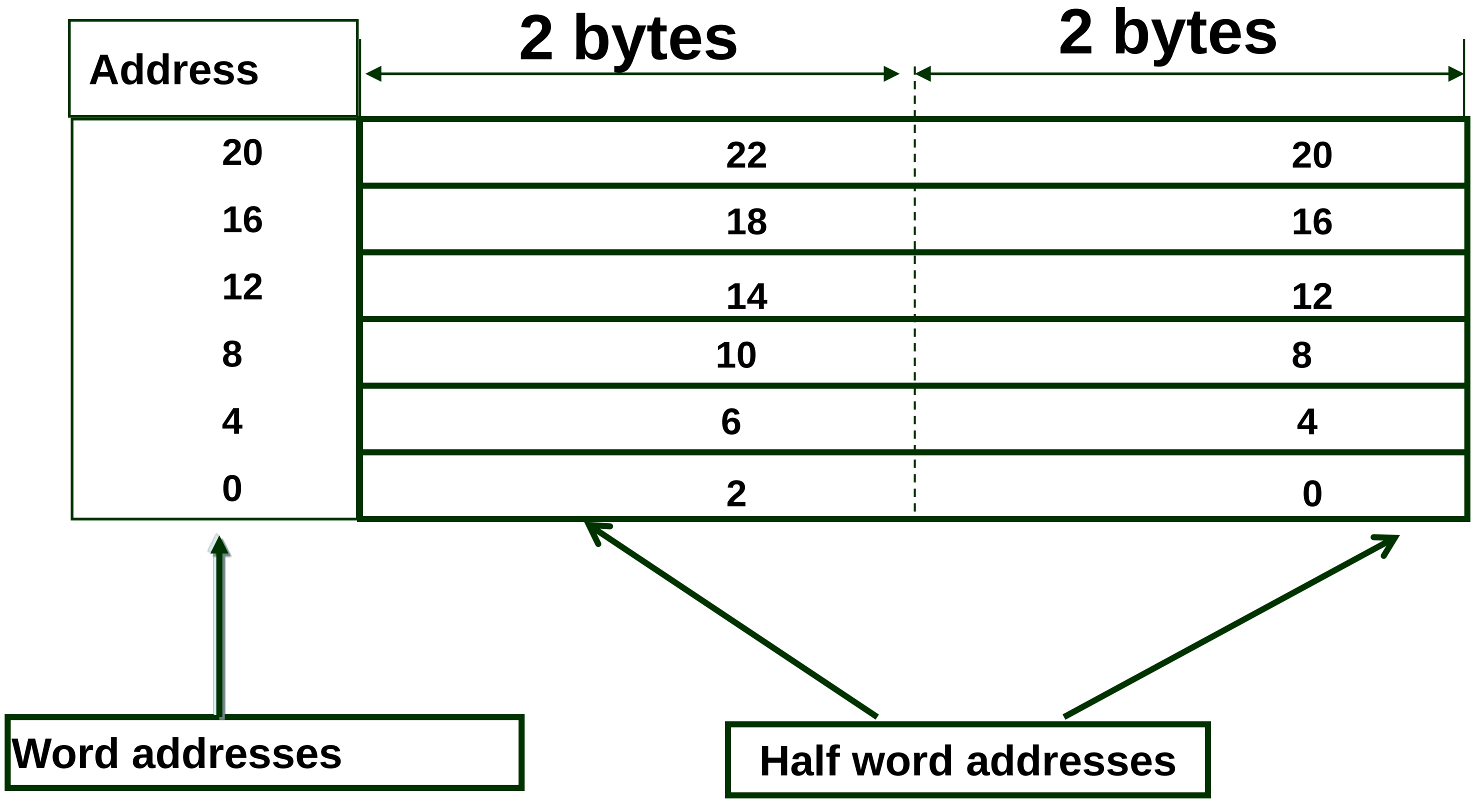
Address	Content
3	1 data byte
2	1 data byte
1	1 data byte
0	1 data byte

- In the case of the MIPS R2000 the memory is organized in words of 4 bytes.
  - Each word ocupies 4 memory positions (bytes) or adresses of memory
  - It is possible to access in the following ways:
    - By Byte → Any address
    - By Word → addresses (0, 4, 8, …)
    - By Half word → addresses (0, 2, 4, …)

MIPS R2000: Memory organization

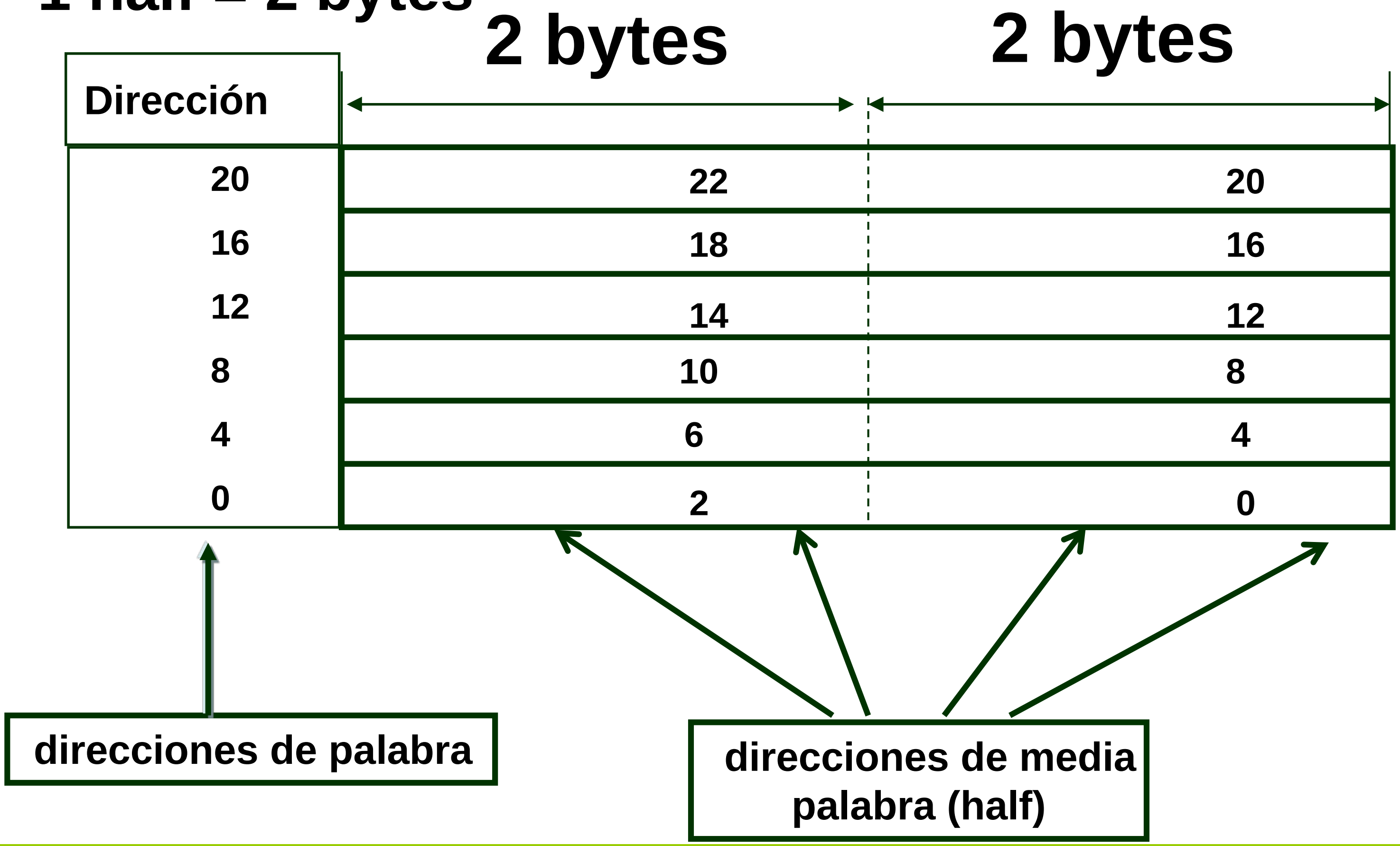
FCO

- Data access by half word
  - 1 half word = 2 bytes



MIPS R2000: Memory organization

- Acceso por media palabra (half)
  - 1 half = 2 bytes



MIPS R2000: Memory organization

- There are 2 ways to store information in memory:
  - big endian format
    -
    - The most significant byte of data or instruction is stored in the lowest memory address.
  - little endian format
    - The least significant byte of data or instruction is stored in the lowest memory address.

MIPS R2000: Memory organization

- **Example 1. Using the little endian and big endian format store the following words in memory:**

1st word:

0XA1	0XB2	0X33	0X22
31 24	23 16	15 8	7 0

2nd word:

0X55	0X55	0XC1	0X00
31 24	23 16	15 8	7 0

0X55	0X55	0XC1	0X00	4
0XA1	0XB2	0X33	0X22	0

Little Endian

0X00	0XC1	0X55	0X55	4
0X22	0X33	0XB2	0XA1	0

Big Endian

Assembler: Memory directives

- Directive “.data”
  - Sintaxis: “.data dirección”
  - It is used to start locating data from the specified address
  - If the address is omitted, the address 0x10000000 is used.

Assembler: Memory directives

- ".space" directive
  - Sintaxis: .space n
  - Allocate n bytes of memory and put 0x00 on each allocated byte
- ".ascii" and ".asciiz" directives
  - Sintaxis: .ascii 'string1' ,['string2', …' string n']
    .asciiz 'string', ['string2', …' string n']
  - Used to store character strings in memory. The characters are coded in ASCII.
  - The ".asciiz" directive adds an extra NULL (0x00) to the end of the string

Assembler: Memory directives

- “.byte” directive
  - Sintaxis: “.byte b1, [b2, b3, …]”
  - Initializes consecutive positions of memory with 1 byte integers coded in two’s complement
- “.half” directive
  - Sintaxis: “.half h1, [h2, h3, …]”
  - Initializes consecutive positions of memory with 2 byte integers coded in two’s complement
- “.word” directive
  - Sintaxis: “.word w1, [w2, w3, …]”
  - Initializes consecutive positions of memory with 4 byte integers coded in two’s complement
- In any case the data can be written in decimal or in hexadecimal.

Assembler: Memory directives

- ".float" directive
  - Sintaxis: ".float f1, [f2, f3, …]"
  - Initializes consecutive positions of memory with 4 byte real numbers coded using the standard IEEE 754 of single precision
- ".double" directive
  - Sintaxis: ".double d1, [d2, d3, …]"
  - Initializes consecutive positions of memory with 4 byte real numbers coded using the standard IEEE 754 of double precision
- In both cases the data can be written in decimal notation or in scientific notation.

Assembler: Memory directives

- Memory data alignment
  - .space, .byte, .ascii and .asciiz allocate memory space from any direction
  - .half allocates memory space from addresses evenly divisible by two
  - .word y .float allocate memory space from addresses evenly divisible by four
  - .double allocates memory space from addresses evenly divisible by eight
- The MIPS simulator used in LAB sessions uses the Little Endian format to store data

Assembler: Memory directives

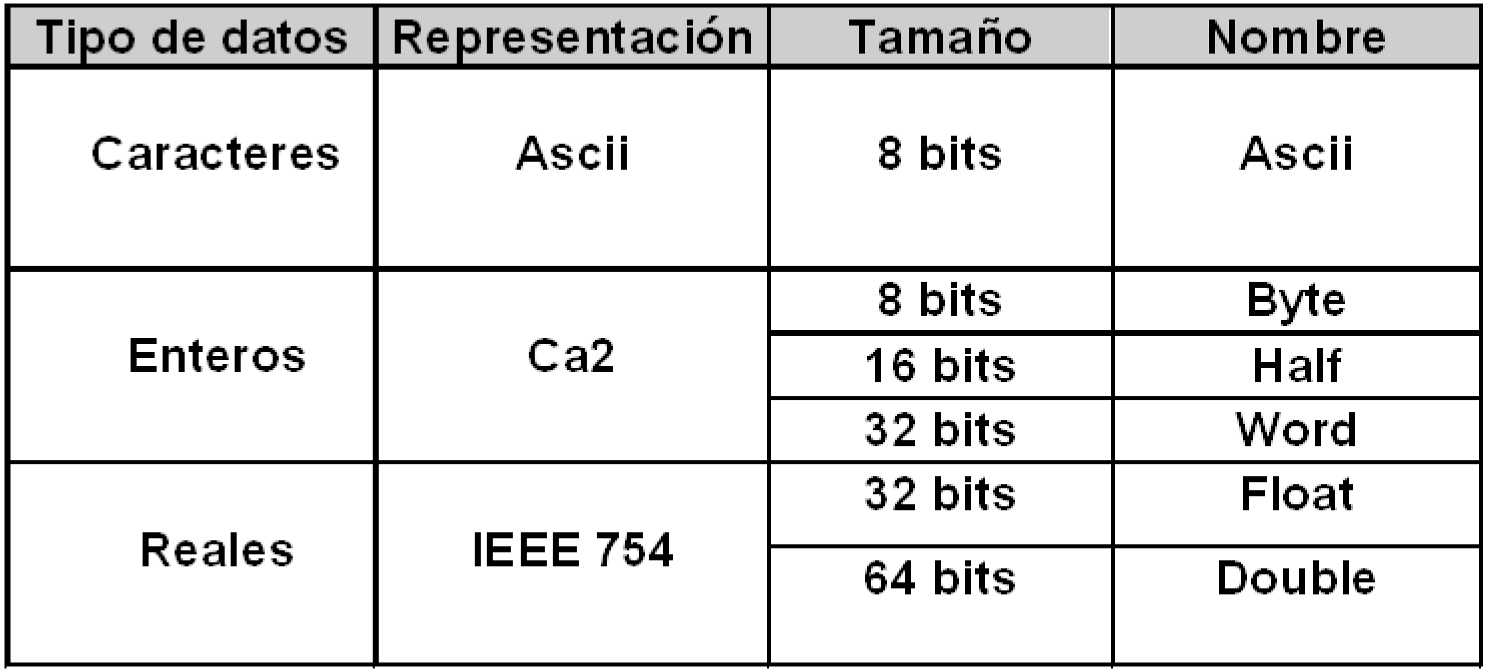
Data alignment example

```
.data 0x10000020
        .word  0x20000001
        .space 1
        .word 0x10
        .byte  0x1, 0x2, 0x3
        .double 1.0  #0x3FF0000000000000 IEEE double precision
```

Content in hexadecimal				Memory address
31 24	23 16	15 8	7 0	
0x20000001				0x10000020
			0x00	0x10000024
0x00000010				0x10000028
	0x03	0x02	0x01	0x1000002C
0x00000000				0x10000030
0x3FF00000				0x10000034

MIPS R2000: Data types & data representation FCO

- Data types supported by the MIPS R2000

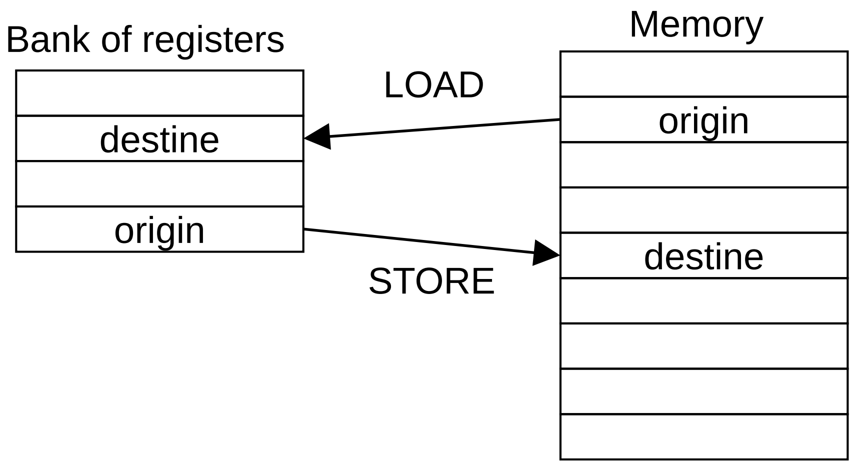


Tipo de datos	Representación	Tamaño	Nombre
Caracteres	Ascii	8 bits	Ascii
Enteros	Ca2	8 bits	Byte
		16 bits	Half
		32 bits	Word
Reales	IEEE 754	32 bits	Float
		64 bits	Double

- Float : Real numbers represented using the standard IEEE 754 of single precision
- Double: Real numbers represented using the standard IEEE 754 of double precision

Memory access instructions

- Load and store instructions
  - They are the only instructions who access memory
    - Load: reads the memory content indicated by a direction and stores it in a register of the bank of registers
    - Store: stores the content of a register in the memory address indicated



Memory access instructions

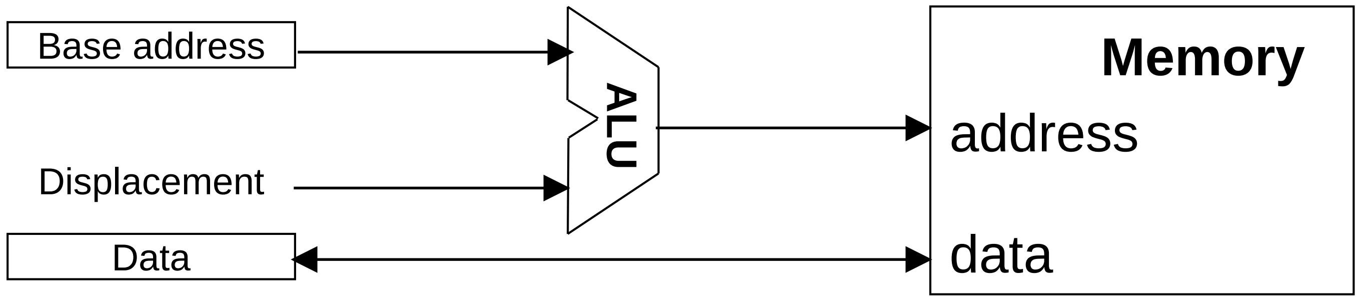
- In the description of the different instructions the following nomenclature will be :
  - rd, rs and rt represent general purpose registers
  - rt $_{31..16}$: MSB of the register rt
  - rt $_{15..0}$: LSB of the register rt
  - disp+rs, generates a memory address which is the result of adding the content of the register rs with the quantity called displacement
  - M[disp+rs] is the content of the memory position disp+rs

data	addresses
0x00000032	**0x00000010**
0x00007654	**0x0000000C**
0x00000543	**0x00000008**
0x00000000	**0x00000004**
0x00000100	**0x00000000**

$$\left.\begin{array}{l} rs = 0x00000004 \\ disp = 8 \end{array}\right\} \begin{array}{l} Disp+rs = 0x0000000C \\ M[disp+rs] = 0x7654 \end{array}$$

Memory access instructions

- Load and store instructions
  - Address calculation for both load and store instructions:
    - Base address: register with the memory base address
    - Displacement: 16 bits integer in two's complement written in the instruction
    - Memory effective address = base address + displacement



Memory access instructions

- Load and store instructions

Sintaxis	Format	Description
lw rt, desp(rs)	I	rt ← M[desp+rs],
lh rt, desp(rs)	I	rt ← M[desp+rs], loads half word (16 bits) extending sign bit
lb rt, desp(rs)	I	rt ← M[desp+rs], loads 1 byte (8 bits) extending sign bit
sw rt, desp(rs)	I	M[desp+rs] ← rt
sh rt, desp(rs)	I	M[desp+rs] ← rt stores the 16 least signifiant bits of register rt in memory
sb rt, desp(rs)	I	M[desp+rs] ← rt stores the least signifiant byte of register rt in memory
lui rt, inm	I	$rt_{31..16} \leftarrow$ inm $rt_{15..0} \leftarrow 0$

Instruction set

Memory reading

```
.globl __start
.data 0x10000000
var_a: .byte 2
var_b: .half -4
var_c: .word 6

.text 0x00400000
__start:
  lui $8, 0x1000
  lb $9, 0($8)
  lh $10, 2($8)
  lw $11, 4($8)
.end
```

- What is the memory content before executing the segment of code?
- What are the memory addresses where the variables “var_a”, “var_b” and “var_c” are stored?
- What is the content of registers $8, $9, $10 and $11 after executing the segment of code?

Instruction set

Writing in memory

```
.globl __start
.data 0x10000000
var_a: .space 1
var_b: .space 2
var_c: .space 4

.text 0x00400000
__start:
  addi $8, $0, -1
  addi $9, $0, 2
  addi $10, $0, 4
  lui $11, 0x1000
  sb $8, 0($11)
  sh $9, 2($11)
  sw $10, 4($11)
.end
```

- What is the memory content before executing the code shown?
- What is the memory content after executing the code shown?
- What is the content of the registers \$8, \$9, \$10 and \$11 after executing the segment of code shown?

Outline

FCO

- Introducción
- **MIPS R2000**
  - Basic concepts
    - Bank of registers
    - Data Types
    - Arithmetic and Logic Unit
    - Memory
    - Structure of a program
  - Instuction set
    - Arithmetic instructions
    - Logic instructions
    - Load and Store instructions
    - Pseudo-instructions li & la
    - Comparison instructions
    - Conditional statements
  - Instruction Coding of instructions
- Relation with others courses
  - Computer Languages
  - Operating Systems
  - Advanced processing
  - Multiprocessors & Supercomputers

Pseudo-instructions

- Definition:
  - Set of instructions that the assembler language incorporates besides the instructions machine of the processor
  - Pseudo-instructions are translated to machine instructions: Normally are required 1 to 4 machine instructions
  - The programmer uses the pseudoinstrucciones as machine instructions
  - The assembler is responsible to replace the pseudo-instructions with the corresponding machine instructions
  - The SPIM simulator uses register $1 for temporary storage
- Example:
  - Load a 32 bits address into a register
  - To cover all the possible comparisons or conditional jumps

Pseudo-instructions

- li: Load Immediate:
  - Sintaxis: li R, imm
  - Description: Stores in register R the signed integer coded in two’s complement the immediate value imm
  - Equivalent machine instructions: the pseudo-instruction li is equivalent to a lui instruction or to a lui instruction and an ori instruction. It depends on the value of imm
  - Examples
    - li $8, 1 → ori $8, $0, 1
    - li $9, 0x10000001 → lui $1, 0x1000; ori $9, $1, 0x0001
    - li $10, 0x10000000 → lui $10, 0x1000

Pseudo-instructions

- la: Load Address:
  - Sintaxis: la R, *etiqueta*
  - Description: Store in register E the memory address pointed by the label *etiqueta*
  - Equivalent machine instructions: the pseudo-instruction la is equivalent to a lui instruction or to a lui instruction and an ori. It depends on the address pointed by the label *etiqueta*
  - Examples
    - According to the following code →
    - la \$8, var_a → lui \$8, 0x1000
    - la \$9, var_b → lui \$1, 0x1000; ori \$9, \$1, 4

```
.data 0x10000000
var_a:  .word 1
var_b:  .word 2
```

Outline

- Introducción
- **MIPS R2000**
  - Basic concepts
    - Bank of registers
    - Data Types
    - Arithmetic and Logic Unit
    - Memory
    - Structure of a program
  - Instuction set
    - Arithmetic instructions
    - Logic instructions
    - Load and Store instructions
    - Pseudo-instructions li & la
    - Comparison instructions
    - Conditional statements
  - Instruction coding
- Relation with others courses
  - Computer Languages
  - Operating Systems
  - Advanced processing
  - Multiprocessors & Supercomputers

Comparison instructions

- Comparison instructions

Sintaxis	Format	Description
slt rd, rs, rt	R	If rs < rt then rd ← 1, if not rd ← 0
slti rt, rs, inm	I	If rs < inm then rt ← 1, if not rt ← 0

Inmediat Comparison

```
.globl __start
.text 0x00400000
__start:
  addi $8, $0, 1
  addi $9, $0, 3
  slti $10, $8, 2
  slti $11, $9, 2
.end
```

Comparison

```
.globl __start
.text 0x00400000
__start:
  addi $8, $0, 1
  addi $9, $0, 3
  slti $10, $8, $9
  slti $11, $9, $8
.end
```

- What is the content of registers $10 y $11 after the execution of each segment of code?

Outline

- Introducción
- MIPS R2000
  - Basic concepts
    - Bank of registers
    - Data Types
    - Arithmetic and Logic Unit
    - Memory
    - Structure of a program
  - Instuction set
    - Arithmetic instructions
    - Logic instructions
    - Load and Store instructions
    - Pseudo-instructions li & la
    - Comparison instructions
    - Conditional statements
  - Instruction coding
- Relation with others courses
  - Computer Languages
  - Operating Systems
  - Advanced processing
  - Multiprocessors & Supercomputers

Conditional statements

- Condicional jump Instructions

Sintaxis	Format	Description
beq rs, rt, etiqueta	I	If rs = rt then jump to the address pointed by the label etiqueta PC🡐 etiqueta
bne rs, rt, etiqueta	I	Si rs <> rt then jump to the address pointed by the label etiqueta PC 🡐 etiqueta

- Uncondicional jump Instructions

Sintaxis	Format	Description
j etiqueta	J	PC🡐 etiqueta, jump to the address pointed by the label etiqueta
jal etiqueta	J	Store the return address in register $31 and jump to the address pointed by the label etiqueta $31🡐 PC updated PC🡐 etiqueta
jr rs	R	PC 🡐 rs, jump to the address stored in the register rs

Conditional statements

Jump of equality

```
.globl __start
.text 0x00400000
__start:
  addi $8, $0, 1
  addi $9, $0, -1
  beq $8,$0,fin
  addi $9, $0, 1
fin:
.end
```

Jump of inequality

```
.globl __start
.text 0x00400000
__start:
  addi $8, $0, 1
  addi $9, $0, -1
  bne $8,$0,fin
  addi $9, $0, 1
fin:
.end
```

- What is the memory address pointed by the label “fin”?
- What is the content of the register $9 after the execution of the program?

Conditional statements

Uncondicional jump

```
.globl __start
.text 0x00400000
__start:
  addi $8, $0, 3
  addi $9, $0, 1
bucle:
  beq $8, $0, fin
  mult $9, $8
  mflo $9
  addi $8, $8, -1
  j bucle
fin:
.end
```

- Which is the memory address pointed by label “bucle”?
- What is the content of the registers $8 y $9 after the execution of the program?
- The program shown performs a well known operation? Which?
- What do registers $8 and $9 represent into the program?

Outline

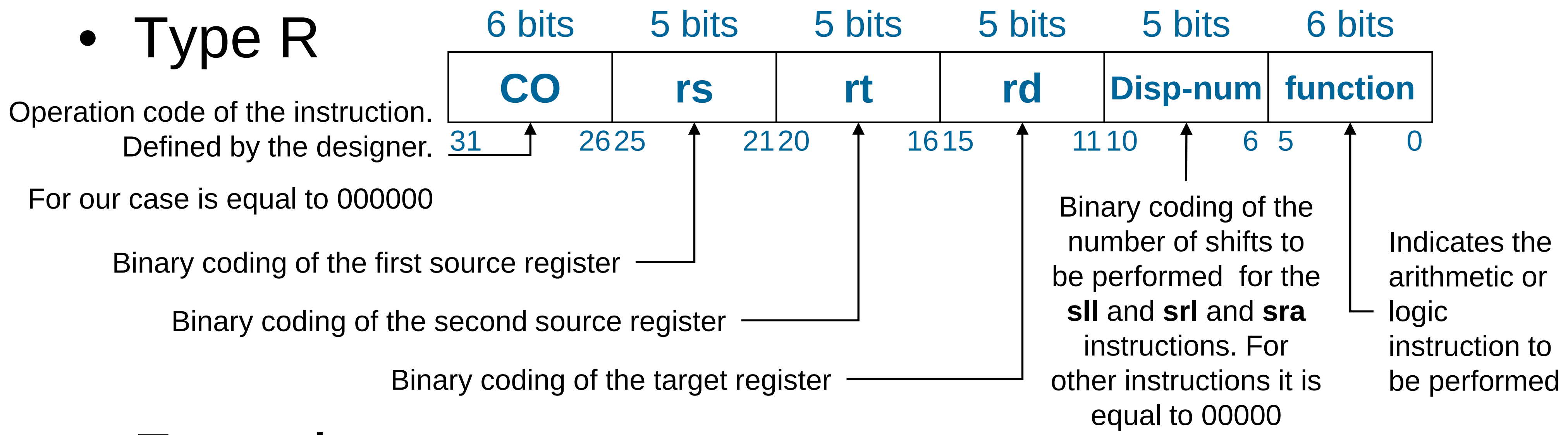
- Introducción
- **MIPS R2000**
  - Basic concepts
    - Bank of registers
    - Data Types
    - Arithmetic and Logic Unit
    - Memory
    - Structure of a program
  - Instuction set
    - Arithmetic instructions
    - Logic instructions
    - Load and Store instructions
    - Pseudo-instructions li & la
    - Comparison instructions
    - Conditional statements
  - Coding of Instruction
- Relation with other courses
  - Lenguajes informáticos
  - Sistemas Operativos
  - Procesamiento avanzado, redes

Coding of instructions

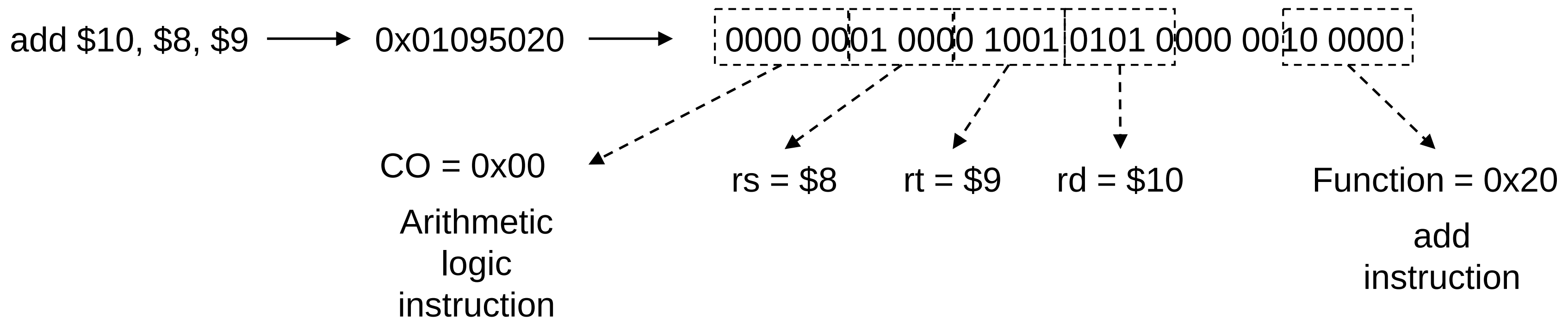
- The format of the instructions indicates the way in which the they are coded in machine's code
- All the instructions of the MIPS R2000 have a length of 32 bits
- MIPS's instructions are grouped in 3 types of instructions:
  - Type R (register type)
  - Type I (immediate type)
  - Type J (unconditional jump)
- The six most significant bits of all the MIPS instructions indicates the operation code

Coding of instructions

- Type R

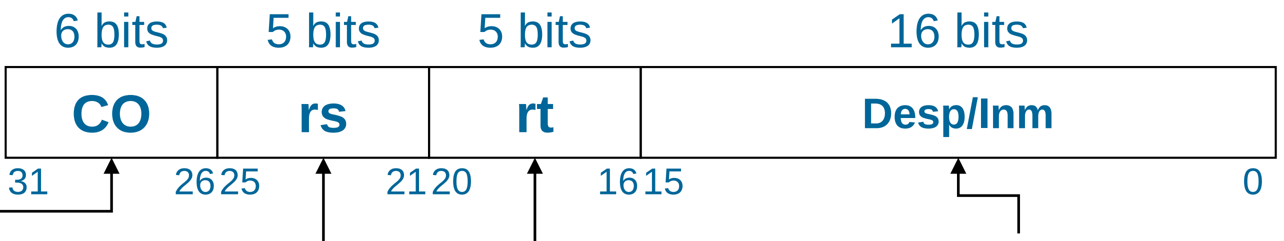

6 bits
5 bits
5 bits
5 bits
5 bits
6 bits
CO
rs
rt
rd
Disp-num
function
31
26
25
21
20
16
15
11
10
6
5
0
Operation code of the instruction. Defined by the designer.
For our case is equal to 000000
Binary coding of the first source register
Binary coding of the second source register
Binary coding of the target register
Binary coding of the number of shifts to be performed for the **sll** and **srl** and **sra** instructions. For other instructions it is equal to 00000
Indicates the arithmetic or logic instruction to be performed

- Examples



Coding of instructions

- Type I



Operation code of the instruction. Defined by the designer . Indicates the instruction type

Binary coding of the first source register

Binary coding of the second source register

Or the target register

16 bits coded in two's complemet:
- For the instructions **beq** and **bne** represents a displacement of words
- For the **load/store** instructions represents a displacement of bytes

- Example

addi $10, $8, -1 ⟶ 0x210aFFFE ⟶ 0010 0001 0000 1010 1111 1111 1111 1110

CO = 0x08 "addi" instruction

rs = $8

rt = $10

Inm = 0xFFFE

Representation of the integer value -2 in two's complement

Coding of instructions

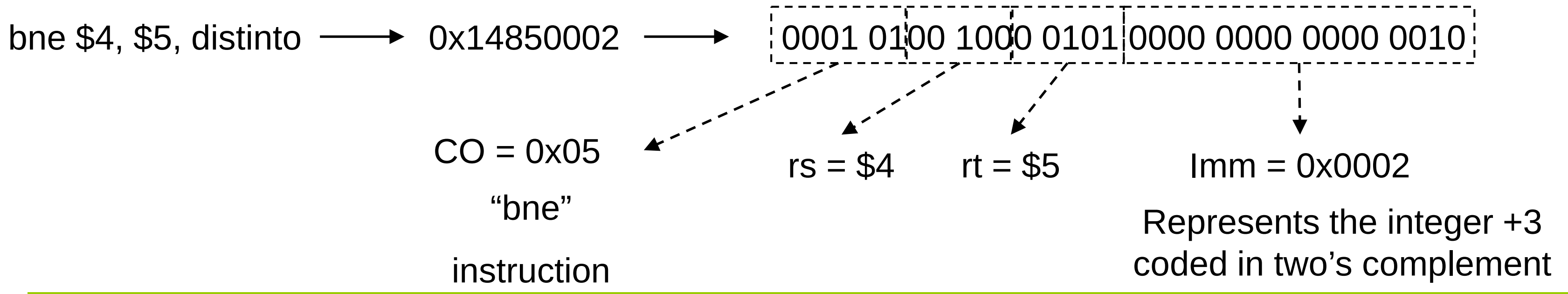
- Type I: Example

```
.text 0x0040000
__start:
  lw $4, 8($10)
  bne $4, $5, distinto
  sub $4, $4, $5
  j salida
distinto: add $4, $4, $5
salida:   addi $5, $5, 1
```

The program counter (PC) points to the instruction being executed (“bne”)

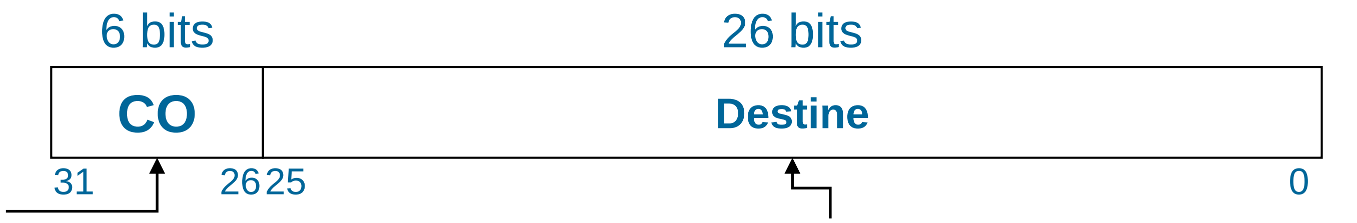
The label “distinto” point to the instruction “add” que located 2 instructions far away from the PC

Each instruction is coded using 32 bits. Each instruction ocupies 4 bytes in memory. The memory addresses where instructions are located are evenly divided by 4: 0, 4, 8, 12, 16, etc. The field displacement for the instruction bne (in this example) represents the number of memory words that must be jumped



Coding of instructions

- Tipo j



Operation code of the instruction. Defined by the designer. Defines the instruction

Memory address pointed by the label which indicates the destine of the jump

- Example (using the code segment of the previous slide)

j salida → 0x08100005 → 0000 1000 0001 0000 0000 0000 0000 0101

CO = 0x02

"j"

instruction

Imm 00 0001 0000 0000 0000 0000 0101

Added = 0000 00

Result = 0000 0000 0100 0000 0000 0000 0001 0100

Address destine= 0x00400014

Only 26 bits from the 32 bits of the Program Counter (PC) are modified by the j instruction . 6 bits remain unchanged
The address jump (32 bits) have to be coded in the firld destine (26 bits). For that purpose the 4 most significant bits are eliminated and the 2 least signifiant bits are eliminated.

Outline

- Introducción
- **MIPS R2000**
  - Basic concepts
    - Bank of registers
    - Data Types
    - Arithmetic and Logic Unit
    - Memory
    - Structure of a program
  - Instuction set
    - Arithmetic instructions
    - Logic instructions
    - Load and Store instructions
    - Pseudo-instructions li & la
    - Comparison instructions
    - Conditional statements
  - Instruction coding
- Relation with others courses
  - Computer Languages
  - Operating Systems
  - Advanced processing
  - Multiprocessors & Supercomputers

Computer languages

- A programming language is an artificial language designed to express computations that can be carried out by machines as the computers

Language	Code
Java language	a=b+c;
Assembler language of the MIPS	add $3,$2,$9
Machine language of the MIPS	0000 0000 0100 1001 0001 1000 0010 0000

Computer languages

- Software: It is the set of the programs of calculation, procedures, rules, documentation and associate information that form a part of the operations of a system of computation.
- Software types according to the standard IEEE 729
  - System Software
  - Programming Software
  - Application Sofware
- The Software adopts several forms in different moments of his life cycle
  - Source code
  - Object code
  - Runnable code

Operating Systems

- What is an operating System?
  - A program that acts as intermediary between the user of a computer and the hardware of the same one
- Objetivos del sistema operativo:
  - To execute programs and to facilitate the solution of the problems of the user
  - To do a suitable use of the computer
- To use the computer of efficient form
- To provide an extended virtual machine

Operating Systems

- Commercial OS:
  - Linux
  - Windows XP, Vista, 7
  - Mac OS X

- PDAS
  - Linux
  - Symbian
  - Windows Mobile
  - Palm OS
  - Android
  - Iphone os

Operating Systems: System calls

- It is the mechanism used by an application to request a service to the operating system.
- Interface between applications and the OS
  - Generally available as functions programmed in assembler
- Typical services of the OS
  - Management of processes
  - Management of light-weight processes
  - Management of signals,
  - Memory management
  - Files management
- Examples
  - read: allows to read data from a file
  - fork: allows to create a new process

Advanced processing

- To increase the power and the efficiency in the processing there are different models of more advanced processing
- The optimization is based on diverse concepts
  - Paralelización de procesos.
  - Utilization of shared resources
  - Specialization of components
- Diverse models exist
  - Multiprocessors
  - Supercomputers
  - Computers networks
  - Distributed systems

Multiprocessors and Supercomputers FCO

- Multiprocessors
  - Computer with two or more processors
  - They allow the processing in parallel processing simultaneously several "threads" belonging to the same process or of different processes
  - They generate certain difficulties of coordination specially in the access to memory
  - They require that the Operating system should be prepared to be able to obtain all the power that they offer.
- Supercomputers
  - Systems with multiple computers connected directly by means of dedicated connections
  - Provide a power of calculation much higher than common computers.

Computer networks and distributed systems

- Computers networks
  - Set of computers connected by communication media (cables, waves, etc.) that allows them to share resources (units of storage, printers, etc.)
  - Diverse models of implementation exist: ISO/OSI, TCP/IP, ATM, etc.
- Distributed systems
  - Systems based on computers networks where the computers communicate across services (model client - server).
  - A server can be a client and viceversa
  - They must be reliable: if a component fails, another component must be capable of replacing it (fault tolerance)

Fundamentos de computadores

Subject 6. Assembler