Tema 3 (I)
Fundamentos de la programación en ensamblador

Grupo ARCOS

Estructura de Computadores

Grado en Ingeniería Informática

Universidad Carlos III de Madrid

Contenidos

1. Introducción
   1. Motivación y objetivos
   2. Lenguaje ensamblador y arquitectura MIPS
2. Programación en ensamblador
   1. Arquitectura MIPS (I)
   2. Tipo de instrucciones (I)
   3. Formato de instrucciones

¡ATENCIÓN!

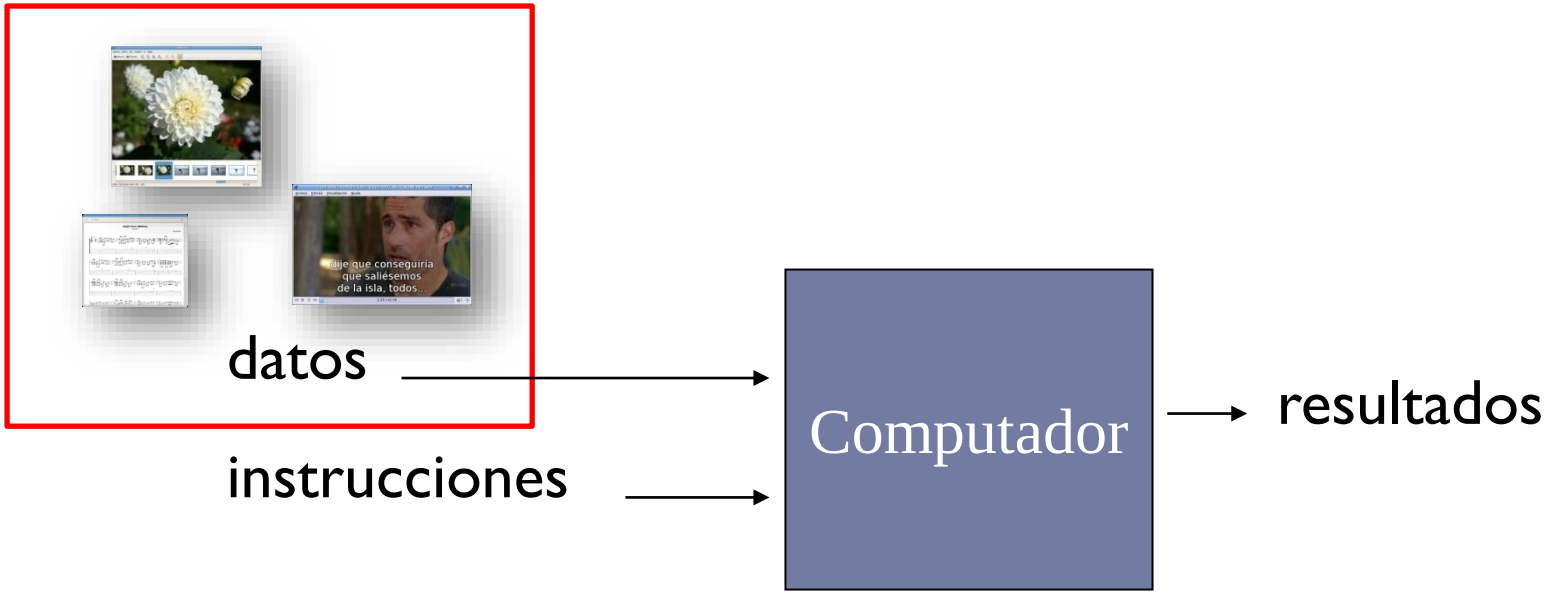
- Estas transparencias son un guión para la clase
- Los libros dados en la bibliografía junto con lo explicado en clase representa el material de estudio para el temario de la asignatura

Contenidos

1. **Introducción**
   1. **Motivación y objetivos**
   2. **Lenguaje ensamblador y arquitectura MIPS**
2. Programación en ensamblador
   1. Arquitectura MIPS (I)
   2. Tipo de instrucciones (I)
   3. Formato de instrucciones

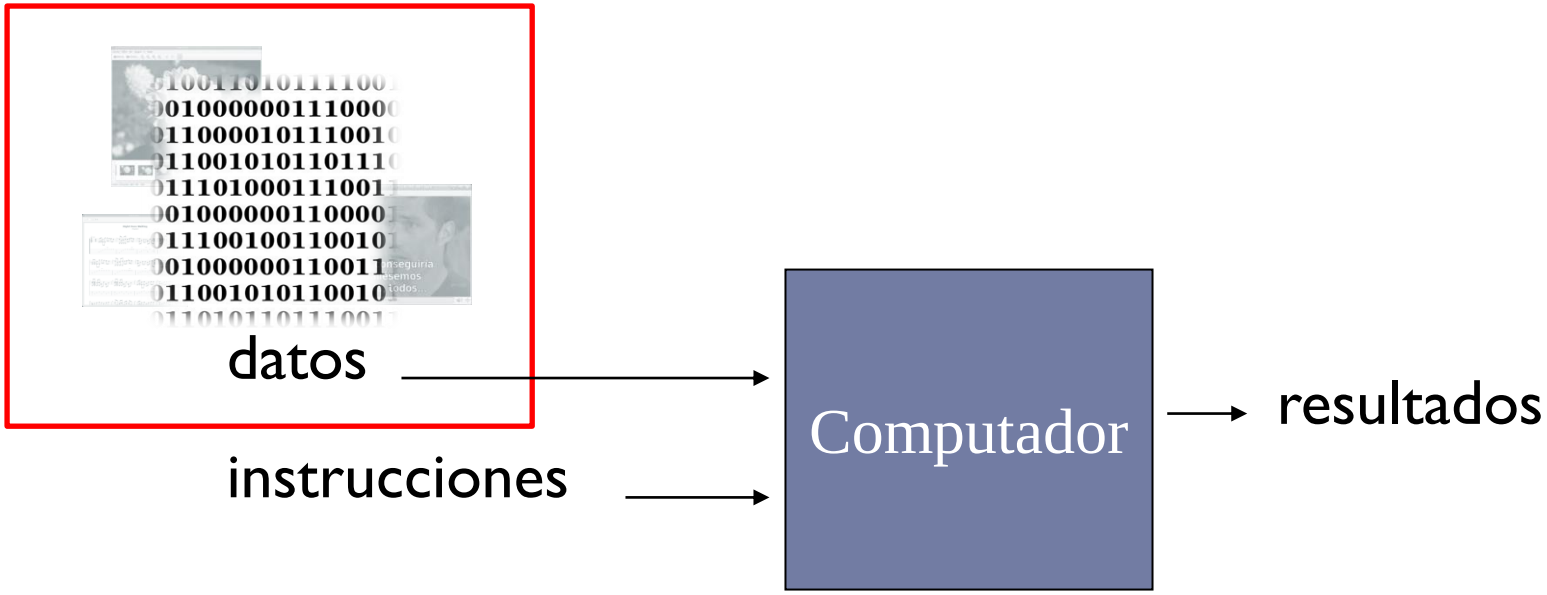
Tipos de información: instrucciones y datos

- Representación de datos ...



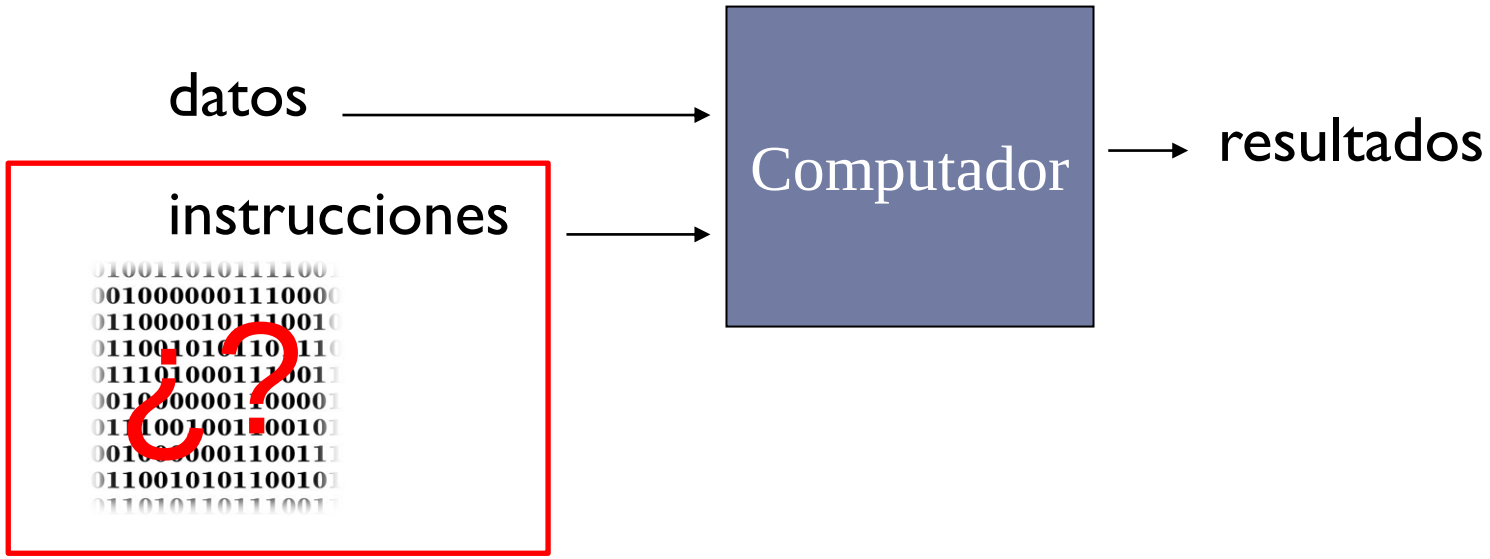
Tipos de información: instrucciones y datos

- Representación de datos en binario.



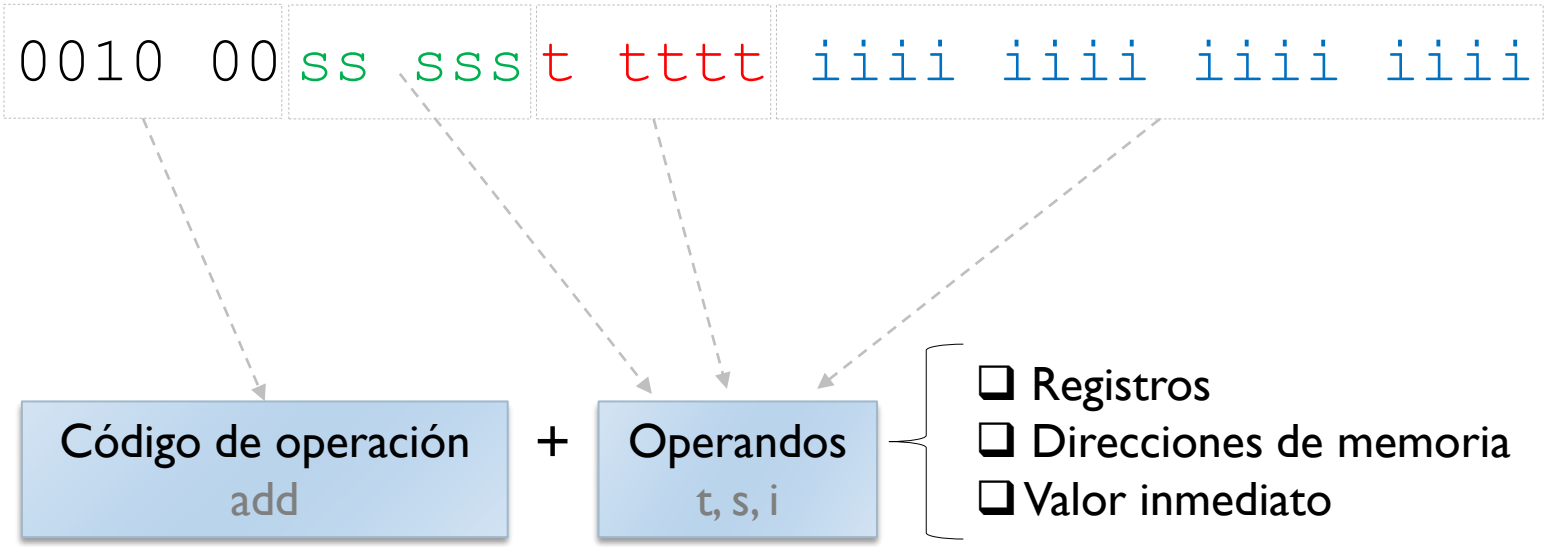
Tipos de información: instrucciones y datos

- ¿Qué sucede con las instrucciones?



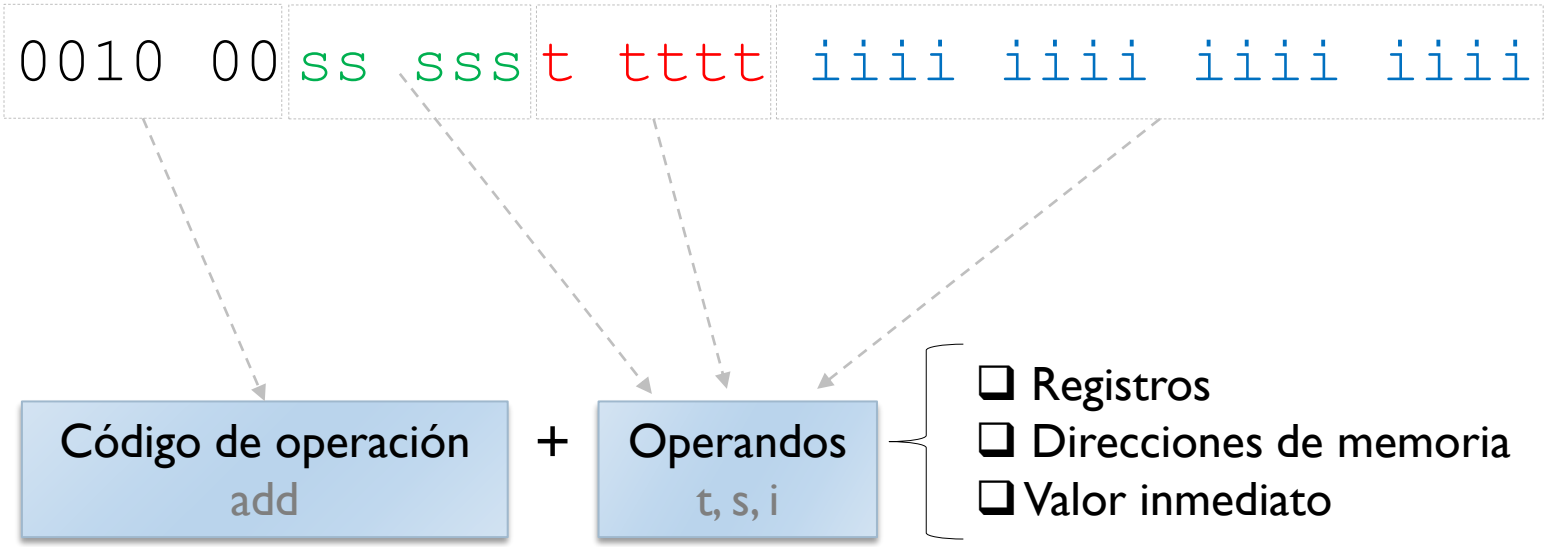
Instrucción máquina

- Ejemplo de instrucción en CPU MIPS:
  - Suma de un registro (s) con un valor inmediato (i) y el resultado de la suma se almacena en registro (t)



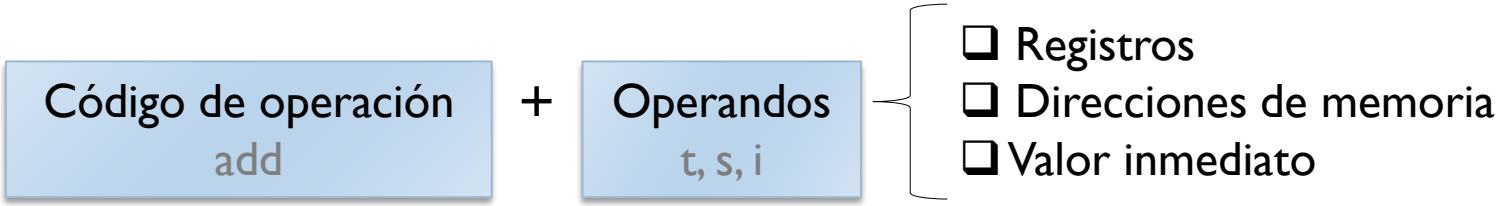
Propiedades de las instrucciones máquina

- Realizan una única y sencilla tarea
- Operan sobre un número fijo de operandos
- Incluyen toda la información necesaria para su ejecución



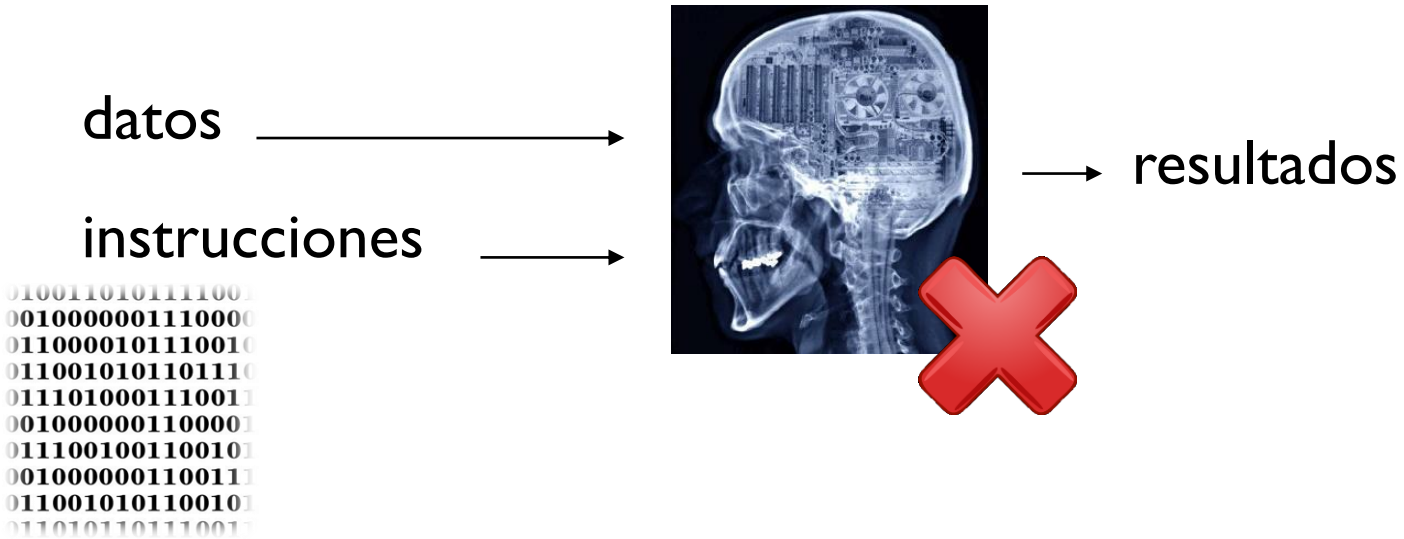
Información incluida en instrucción máquina

- La operación a realizar.
- Dónde se encuentran los operandos:
  - En registros
  - En memoria
  - En la propia instrucción (inmediato)
- Dónde dejar los resultados (como operando)
- Una referencia a la siguiente instrucción a ejecutar
  - De forma implícita, la siguiente instrucción
  - De forma explícita en las instrucciones de bifurcación (como operando)



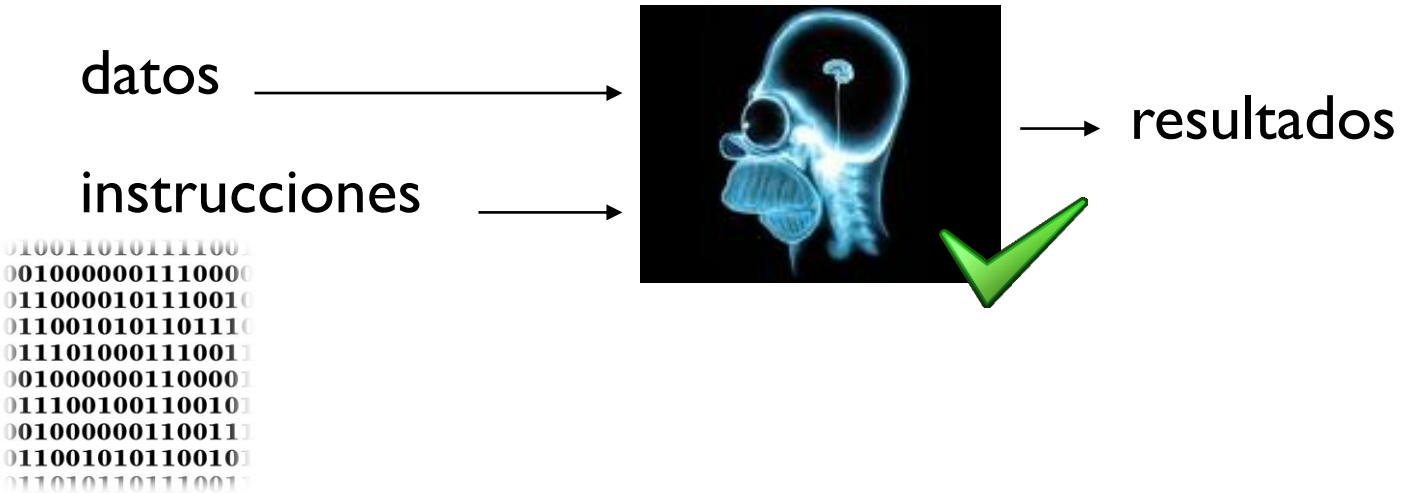
Instrucción máquina

- No son instrucciones complejas…



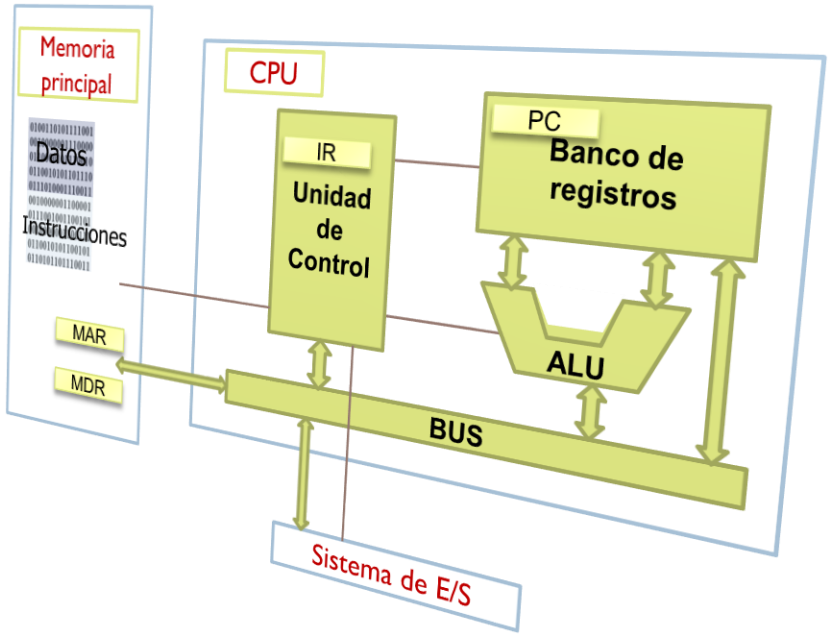
Instrucción máquina

- … sino sencillas tareas…



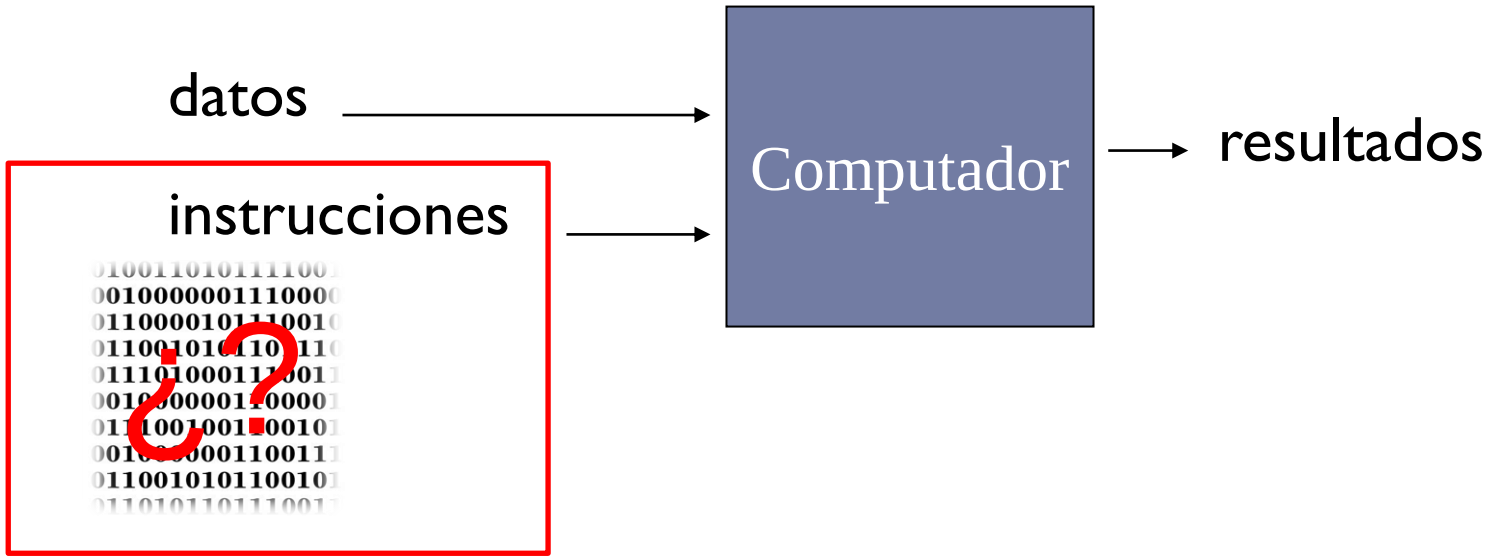
Instrucción máquina

- … realizadas por el procesador:
  - Transferencias de datos
  - Aritméticas
  - Lógicas
  - Conversión
  - Entrada/Salida
  - Control del sistema
  - Control de flujo



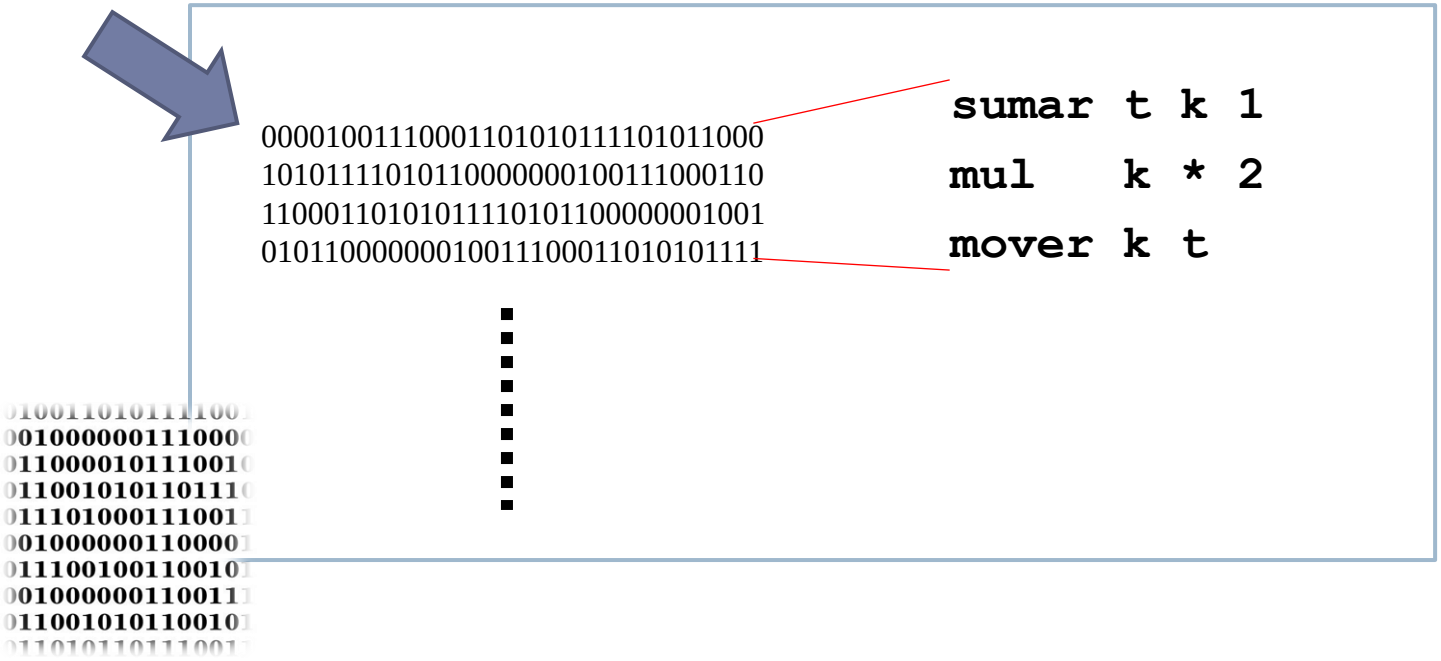
Tipos de información: instrucciones y datos

- ¿Qué sucede con las instrucciones?



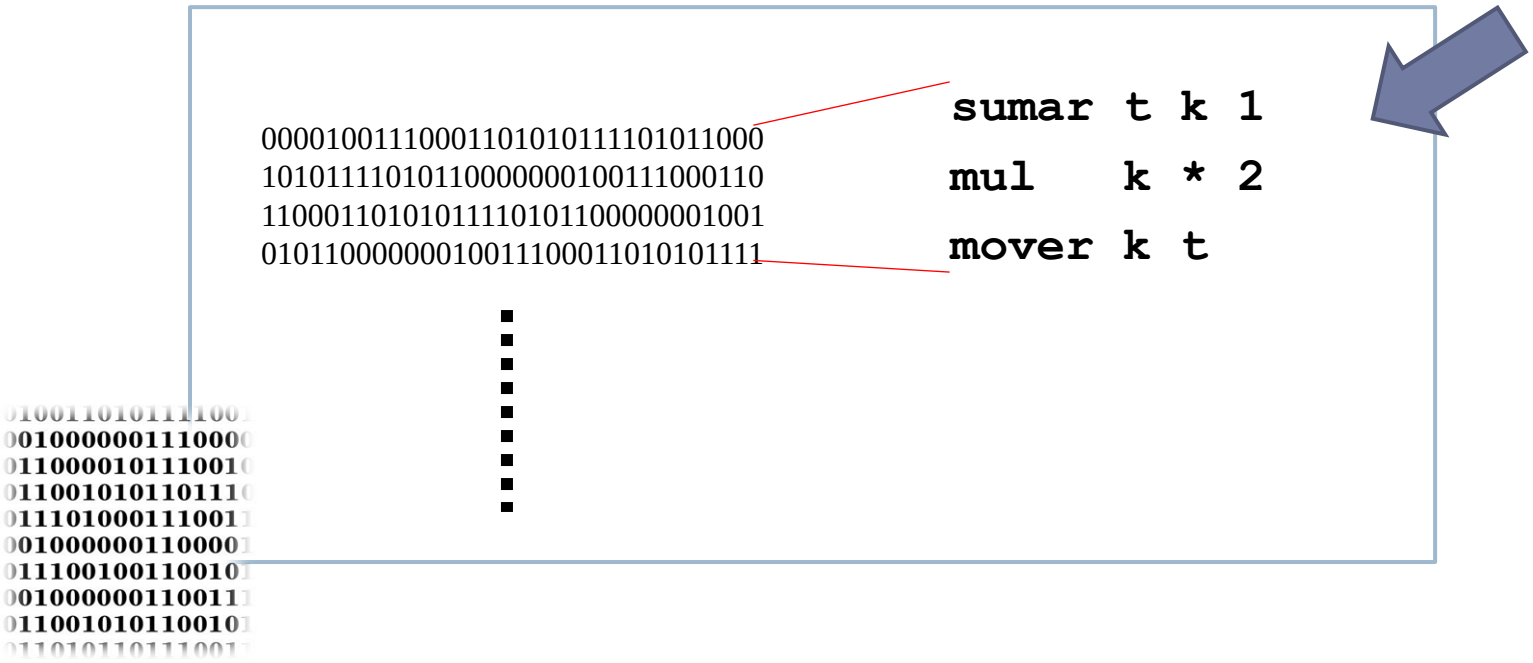
Definición de programa

- **Programa**: lista ordenada de instrucciones máquina que se ejecutan en secuencia por defecto.



Definición de lenguaje ensamblador

- **Lenguaje ensamblador**: lenguaje legible por un programador que constituye la representación más directa del código máquina específico de una arquitectura de computadoras.



Definición de lenguaje ensamblador

- **Lenguaje ensamblador**: lenguaje legible por un programador que constituye la representación más directa del código máquina específico de una arquitectura de computadoras.
  - Emplea códigos nemónicos para representar instrucciones
    - `add` – suma
    - `lw` – carga un dato de memoria
  - Emplea nombres simbólicos para designar a datos y referencias
    - `$t0` – identificador de un registro
  - Cada instrucción en ensamblador se corresponde con una instrucción máquina
    - `add $t1, $t2, $t3`

Proceso de compilación

Lenguaje de alto nivel

```
#include <stdio.h>

#define PI 3.1416
#define RADIO 20

int main ( )
{
   int l;

   l=2*PI*RADIO;
   printf("long: %d\n",l) ;
   return (0);
}
```

Lenguaje ensamblador

```
.data
   PI: .word 3.14156
   RADIO: .word 20

.text
   li $a0 2
   la $t0 PI
   lw $t0 ($t0)
   la $t1 RADIO
   lw $t1 ($t1)
   mul $a0 $a0 $t0
   mul $a0 $a0 $t1

   li $v0 1
   syscall
```

Lenguaje binario

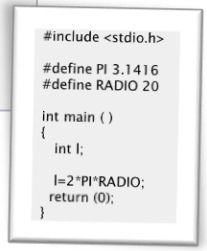
```
0100110101111001
0010000001110000
0110000101110010
0110010101101110
0111010001110011
0010000001100001
0111001001100101
0010000001100111
0110010101100101
0110101101110011
```

Compilación: ejemplo

- Edición de hola.c
  - gedit hola.c

hola.c

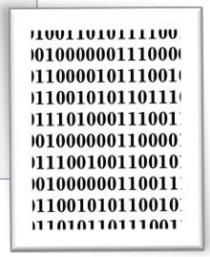
```
int main ( )
{
    printf("Hola mundo…\n") ;
}
```



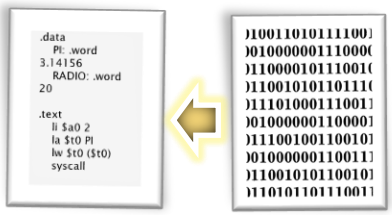
- Generación del programa hola:
  - gcc hola.c –o hola

hola

```
MZ      ÿÿ  ¸       @
€   º ´   Í!¸LÍ!This program cannot be run in DOS
mode.

$       PE  L ,UŽI   ù   à
8                    @                   P     ³Ð
@  ^.text   °                          `.rdata
@  @.bss    @    0                     €  À.idata  ^
```

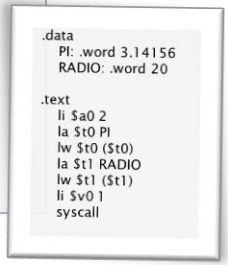


Compilación: ejemplo



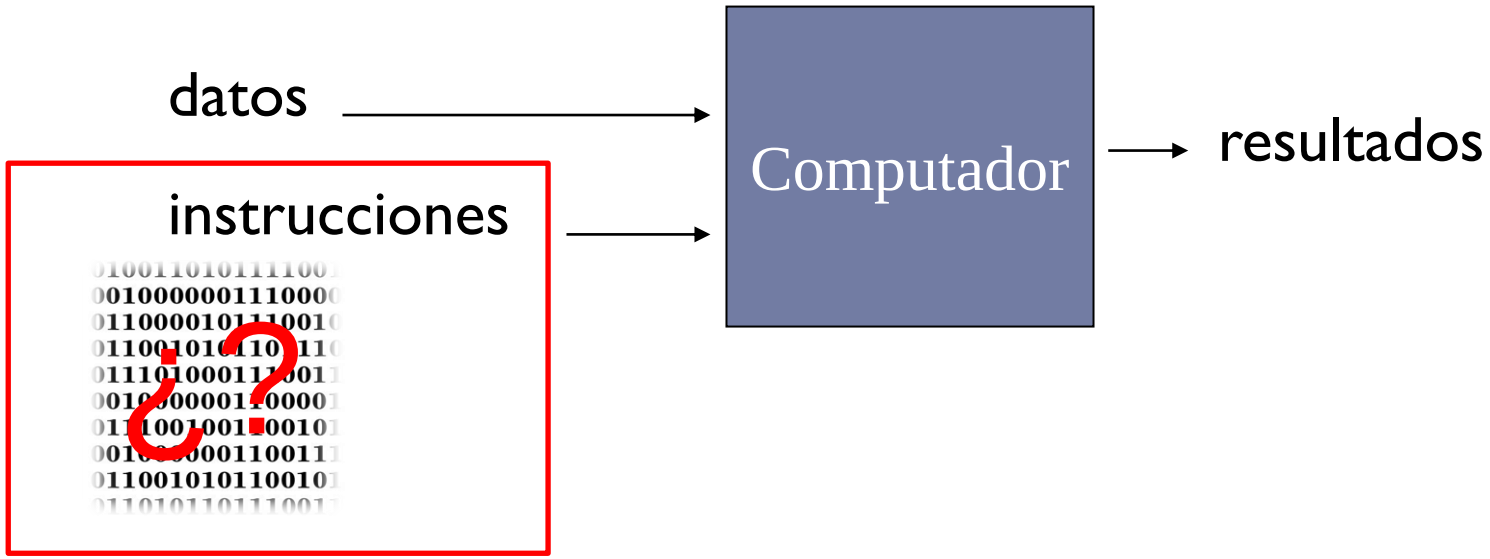
- Desensamblar hola:
  - objdump –d hola

```
hola.exe:     formato del fichero pei-i386

Desensamblado de la secci'on .text:

00401000 <_WinMainCRTStartup>:
  401000:       55                      push   %ebp
  ...
  40103f:       c9                      leave
  401040:       c3                      ret

00401050 <_main>:
  401050:       55                      push   %ebp
  401051:       89 e5                   mov    %esp,%ebp
  401053:       83 ec 08                sub    $0x8,%esp
  401056:       83 e4 f0                and    $0xfffffff0,%esp
  401059:       b8 00 00 00 00          mov    $0x0,%eax
  40105e:       83 c0 0f                add    $0xf,%eax
  401061:       83 c0 0f                add    $0xf,%eax
  401064:       c1 e8 04                shr    $0x4,%eax
  401067:       c1 e0 04                shl    $0x4,%eax
  40106a:       89 45 fc                mov    %eax,0xfffffffc(%ebp)
  40106d:       8b 45 fc                mov    0xfffffffc(%ebp),%eax
  401070:       e8 1b 00 00 00          call   401090 <___chkstk>
  401075:       e8 a6 00 00 00          call   401120 <___main>
  40107a:       c7 04 24 00 20 40 00    movl   $0x402000,(%esp)
  401081:       e8 aa 00 00 00          call   401130 <_printf>
  401086:       c9                      leave
  401087:       c3                      ret
  ...
```



Tipos de información: instrucciones y datos

- ¿Qué motivación hay para aprender ensamblador MIPS?



Motivación para aprender ensamblador

```
#include <stdio.h>

#define PI 3.1416
#define RADIO 20

int main ( )
{
   register int l;

   l=2*PI*RADIO;
   printf("long: %d\n",l) ;
   return (0);
}
```

- Para comprender que ocurre cuando un computador ejecuta una sentencia de un lenguaje de alto nivel.
  - C, C++, Java, PASCAL, …
- Para poder determinar el impacto en tiempo de ejecución de una instrucción de alto nivel.

Motivación para aprender ensamblador

```
.data
    PI: .word 3.14156
    RADIO: .word 20

.text
    li $a0 2
    la $t0 PI
    lw $t0 ($t0)
    la $t1 RADIO
    lw $t1 ($t1)
    mul $a0 $a0 $t0
    mul $a0 $a0 $t1

    li $v0 1
    syscall
```

- Porque es útil en dominios específicos.
  - Compiladores,
  - SSOO
  - Juegos
  - Sistemas empotrados
  - Etc.

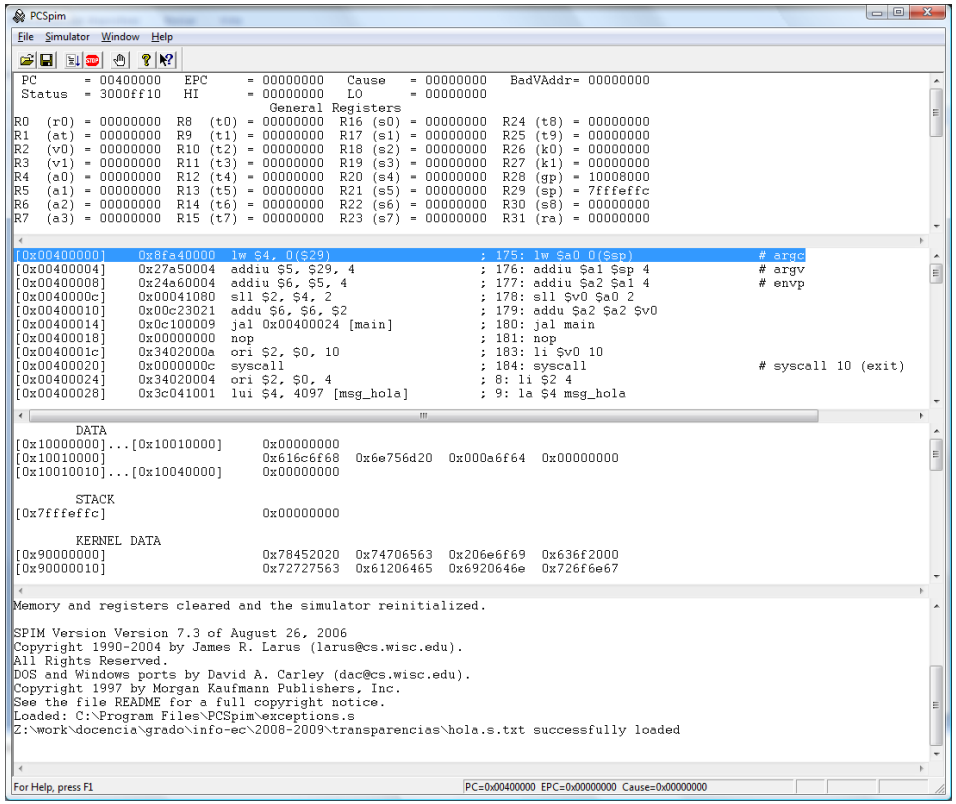
Motivación para usar MIPS

(Microprocessor without Interlocked Pipeline Stages)

- Arquitectura simple.
  - Facilidad de aprendizaje.
- Ensamblador similar al de otros procesadores.
- Usado en diversos dispositivos

Motivación para usar SPIM

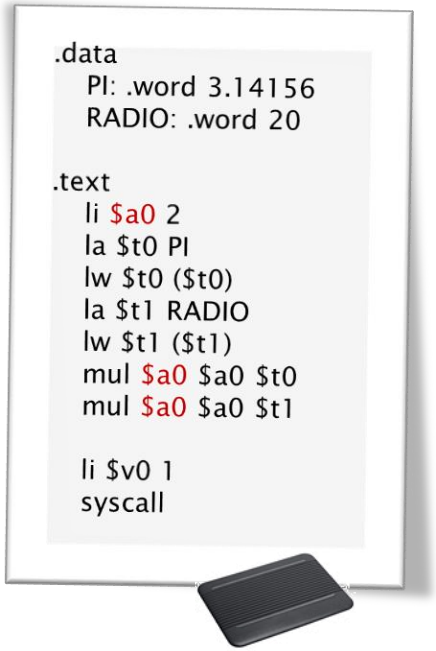
http://pages.cs.wisc.edu/~larus/spim.html



- SPIM es un simulador de una arquitectura MIPS
- Aplicación multiplataforma:
  - Linux
  - Windows
  - MacOS
- Permite simular una arquitectura MIPS

Objetivos

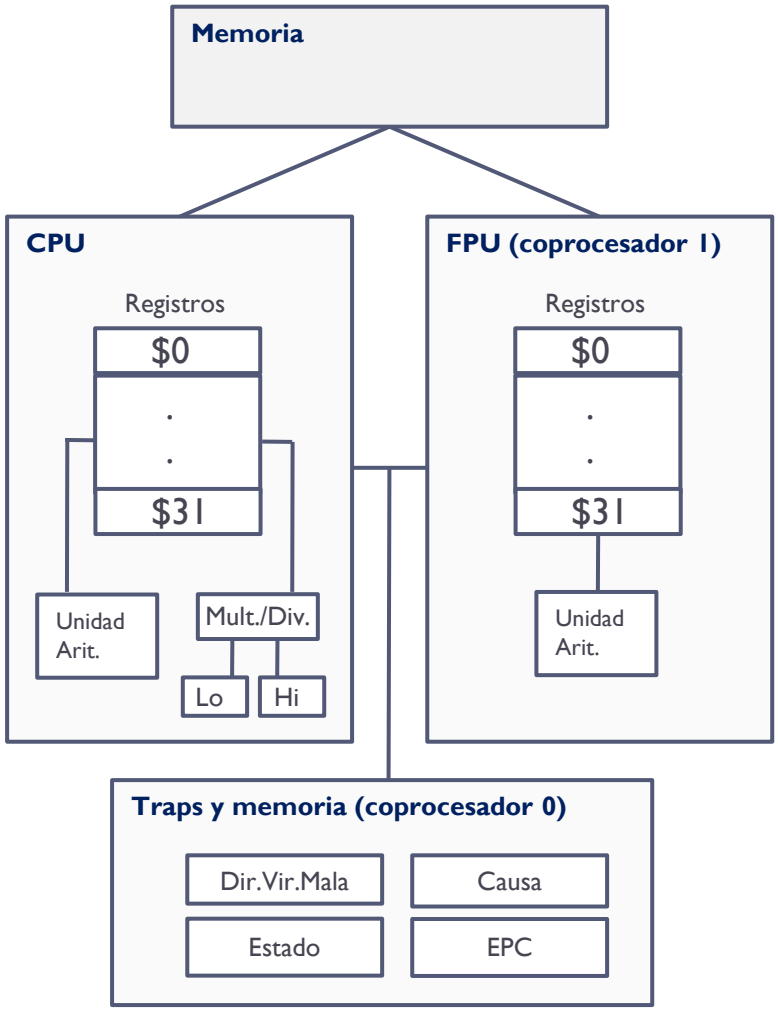
- Saber como los elementos de un lenguaje de alto nivel se pueden representar en ensamblador:
  - Tipos de datos (int, char, …)
  - Estructuras de control (if, while, …)
- Poder escribir pequeños programas en ensamblador.



Contenidos

1. Introducción
   1. Motivación y objetivos
   2. Lenguaje ensamblador y MIPS
2. **Programación en ensamblador**
   1. **Arquitectura MIPS (I)**
   2. Tipo de instrucciones (I)
   3. Formato de instrucciones

Arquitectura MIPS



- MIPS R2000/R3000
  - Procesador de 32 bits
  - Tipo RISC
  - CPU + coprocesadores auxiliares
- Coprocesador 0
  - excepciones, interrupciones y sistema de memoria virtual
- Coprocesador 1
  - FPU (Unidad de Punto Flotante)

http://es.wikipedia.org/wiki/MIPS_(procesador)

Banco de registros (enteros)

Nombre registro	Número	Uso
zero	0	Constante 0
at	1	Reservado para el ensamblador
v0, v1	2, 3	Resultado de una rutina (o expresión)
a0, …, a3	4, …, 7	Argumento de entrada para rutinas
t0, …, t7	8, …, 15	Temporal (NO se conserva entre llamadas)
s0, …, s7	16, …, 23	Temporal (se conserva entre llamadas)
t8, t9	24, 25	Temporal (NO se conserva entre llamadas)
k0, k1	26, 27	Reservado para el sistema operativo
gp	28	Puntero al área global
sp	29	Puntero a pila
fp	30	Puntero a marco de pila
ra	31	Dirección de retorno (rutinas)

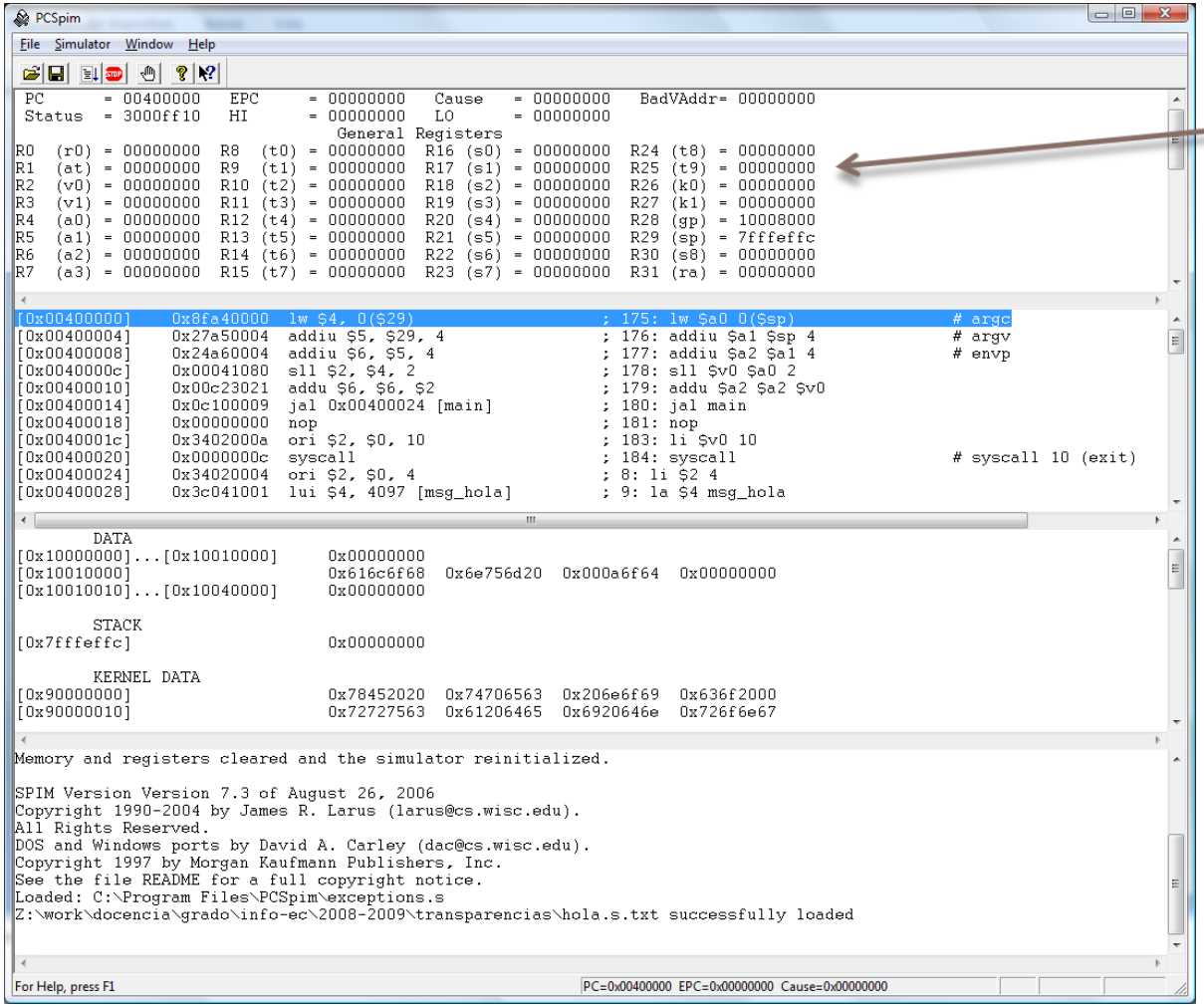
- Hay 32 registros
  - 4 bytes de tamaño (una palabra)
  - Se nombran con un $ al principio
- Convenio de uso
  - Reservados
  - Argumentos
  - Resultados
  - Temporales
  - Punteros

Banco de registros (flotantes)

Número	Uso
0, ….,3	Resultados (como los $v)
4, …, 11	Temporales (como los $t)
12, …, 15	Parámetros (como los $a)
16, …, 19	Temporales (como los $t)
20, …., 31	Preservados (como los $s)

- Hay 32 registros
  - 4 bytes de tamaño (una palabra)
  - Se nombran con un $ al principio
- Se pueden ver como 16 registros dobles
  - Se unen de dos en dos consecutivos
  - Ej.: $f0' = ($f0,$f1)

SPIM



Banco de registros

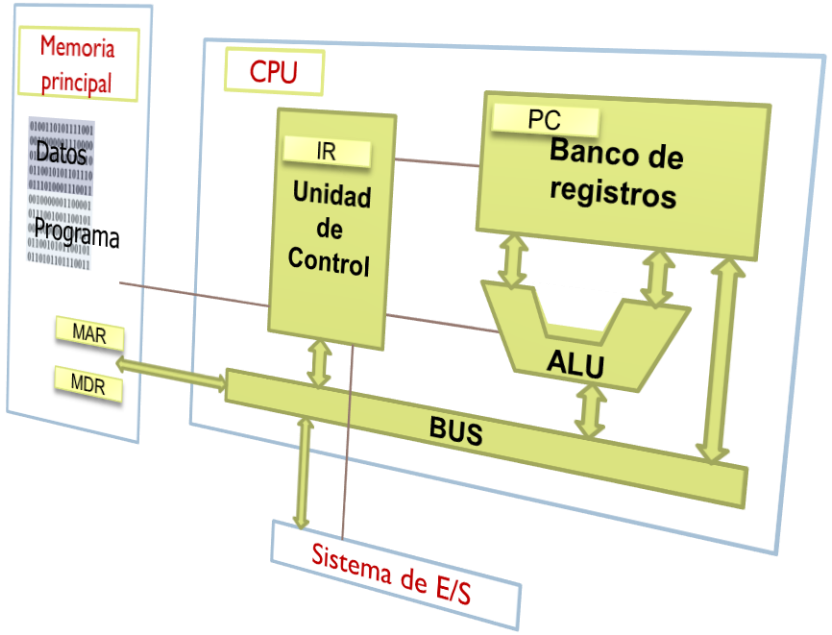
$0, $1, $2, …

$f0, $f1, …

Contenidos

1. Introducción
   1. Motivación y objetivos
   2. Lenguaje ensamblador y MIPS
2. **Programación en ensamblador**
   1. Arquitectura MIPS (I)
   2. **Tipo de instrucciones (I)**
   3. Formato de instrucciones

Tipo de instrucciones



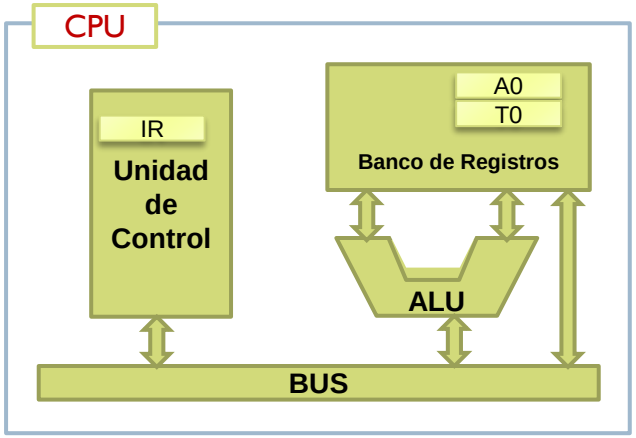
- Transferencias de datos
- Aritméticas
- Lógicas
- Conversión
- Control del sistema
- Control de flujo

Transferencia de datos (1)

- Transferencias de datos
- Aritméticas
- Lógicas
- Desplazamiento/Rotación
- Control del sistema
- Control de flujo

- Copia datos entre registros

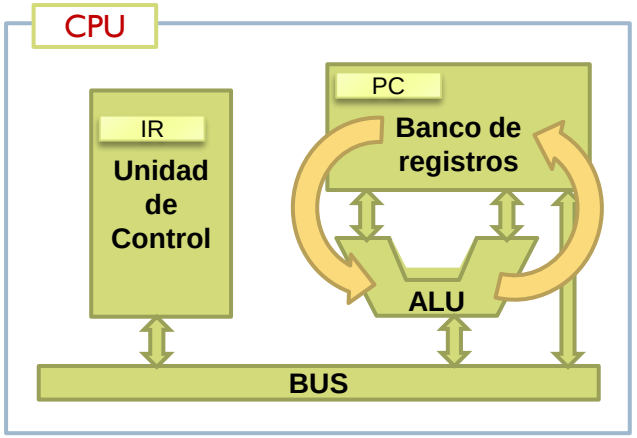
- Ejemplos:
  - Registro a registro
    move $a0 $t0
  - Carga inmediata
    li $t0 I




```
move   $a0 $t0      # BR[$a0] = BR[$t0]
li     $t0  I       # BR[$t0] = IR(li,$t0,I)
```

Aritméticas (1)

- Realiza operaciones aritméticas de enteros en la ALU
- Ejemplos:
  - Sumar
    add $t0 $t1 $t2
  - Restar
    sub $t0 $t1 $t2
  - Multiplicar
    mul $t0 $t1 $t2
  - División entera (5 / 2=2)
    div $t0 $t1 $t2
  - Resto de la división (5 % 2=1)
    rem $t0 $t1 $t2



Ejemplo

```
int a = 5;
int b = 7;
int c = 8;
int i;
```

```
i = a * (b + c)
```

```
.text
.globl main
 main:
```

```
li $t1 5
li $t2 7
li $t3 8
```

```
add  $t4 $t2 $t3
mul  $t4 $t4 $t1
```

Ejercicio

```
int a = 5;
int b = 7;
int c = 8;
int i;


i = -(a * (b - 10) + c)
```

```
.text
.globl main
 main:
```

Ejercicio

```
int a = 5;
int b = 7;
int c = 8;
int i;
```

```
i = -(a * (b - 10) + c)
```

```
.text
.globl main
 main:
    li $t1 5
    li $t2 7
    li $t3 8

    li   $t0 10
    sub  $t4 $t2 $t0
    mul  $t4 $t4 $t1
    add  $t4 $t4 $t3
    li   $t0 -1
    mul  $t4 $t4 $t0
```

Aritméticas (2)

- Aritmética en binario puro o en complemento a dos
- Ejemplos:
  - Suma con signo (ca2)
    ```
    add     $t0  $t1 $t2
    ```
  - Suma inmediata con signo
    ```
    addi    $t0 $t1 -1
    ```
  - Suma en sin signo (binario puro)
    ```
    addu    $t0 $t1 $t2
    ```
  - Suma inmediata sin signo
    ```
    addiu   $t0 $t1 2
    ```

- No **overflow**:

```
li $t0 0x7FFFFFFF
li $t1 1
addu  $t0  $t0 $t1
```

- Con **overflow**:

```
li $t0 0x7FFFFFFF
li $t1 1
add   $t0  $t0 $t1
```

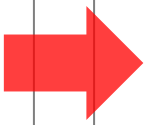
Ejercicio

```
.text
.globl main
 main:

        li $t1 5
        li $t2 7
        li $t3 8

        li   $t0 10
        sub  $t4 $t2 $t0
        mul  $t4 $t4 $t1
        add  $t4 $t4 $t3
        li   $t0 -1
        mul  $t4 $t4 $t0
```

¿Y usando las nuevas instrucciones?

Ejercicio

```
.text
.globl main
 main:

      li $t1 5
      li $t2 7
      li $t3 8

      li   $t0 10
      sub  $t4 $t2 $t0
      mul  $t4 $t4 $t1
      add  $t4 $t4 $t3
      li   $t0 -1
      mul  $t4 $t4 $t0
```



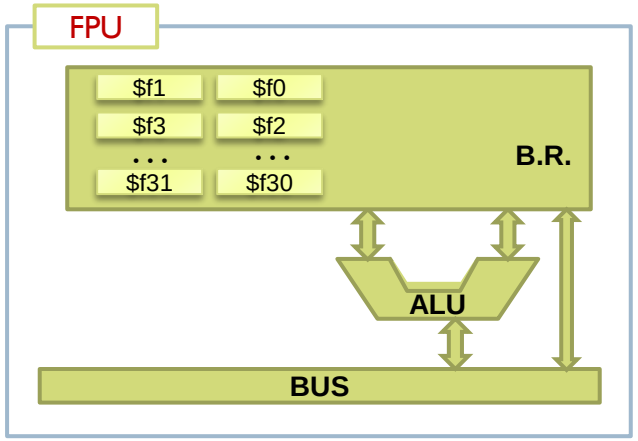
```
.text
.globl main
 main:

      li $t1 5
      li $t2 7
      li $t3 8

      addi  $t4 $t2 -10
      mul   $t4 $t4 $t1
      add   $t4 $t4 $t3
      mul   $t4 $t4 -1
```

Aritméticas (3)

- Aritmética de coma flotante IEEE 754 en la FPU
- Ejemplos:
  - Suma simple precisión
    add.s $f0 $f1 $f4

    f0 = f1 + f4
  - Suma doble precisión
    add.d $f0 $f2 $f4

    (f0,f1) = (f2,f3) + (f4,f5)
  - Otras instrucciones importantes:
    li.s, li.d, cvt.*x*.*y*, mfc*x*, mtc*x*, mul.s, div.s, sub.s, etc.



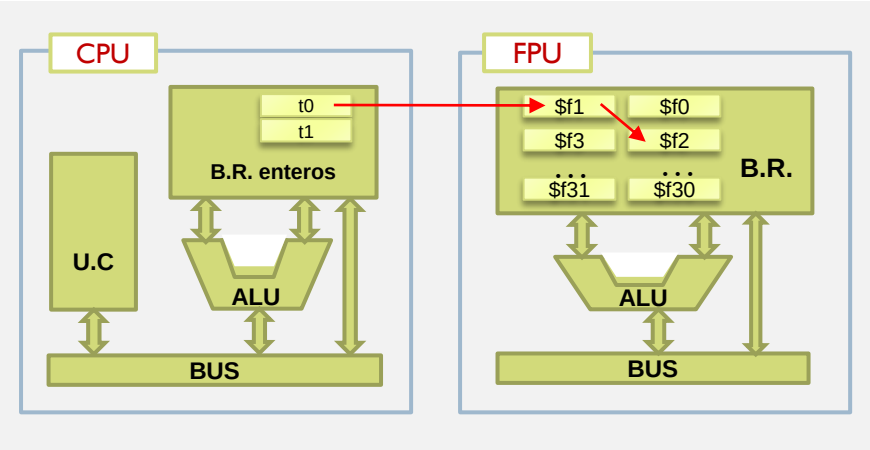
Ejemplo

```
float PI    = 3,1415;
int   radio = 4;
float longitud;

longitud = PI * radio;
```

```
.text
.globl main
 main:
   li.s     $f0  3.1415
   li       $t0 4
```

Ejemplo

```
float PI     = 3,1415;
int    radio = 4;
float longitud;

longitud = PI * radio;
```



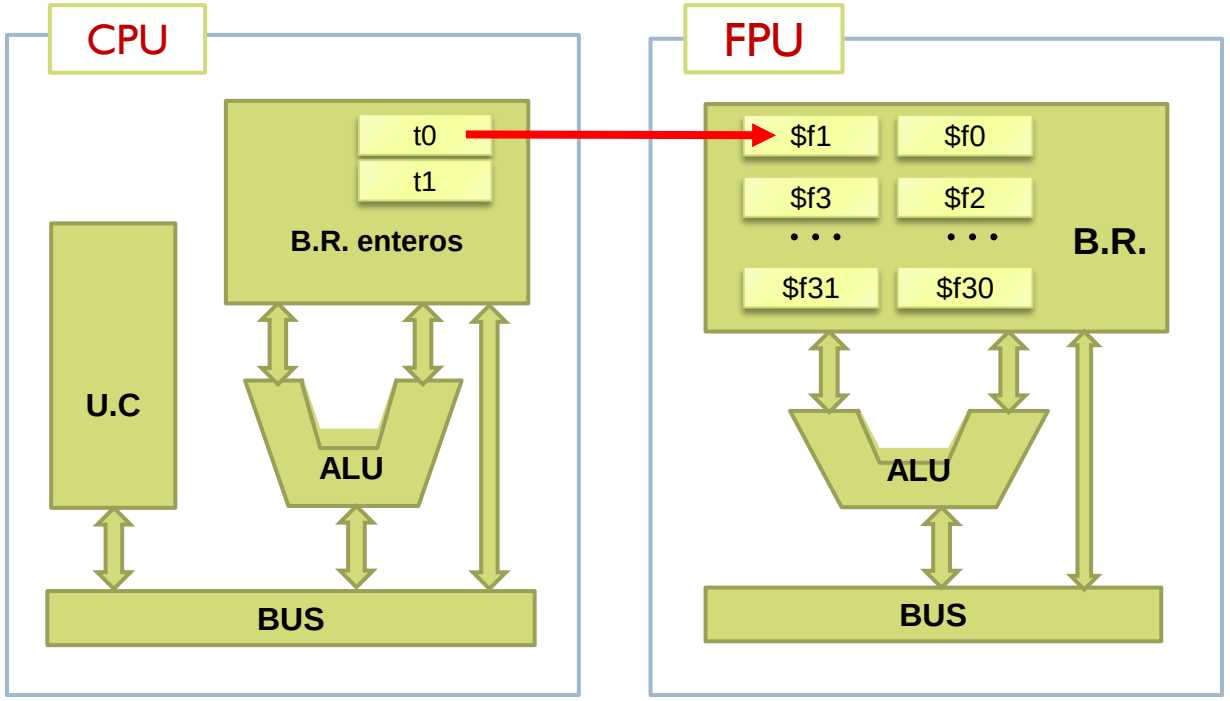

```
.text
.globl main
 main:
    li.s     $f0  3.1415
    li       $t0 4

    mtc1     $t0 $f1    # 4ca2
    cvt.s.w  $f2 $f1    # 4ieee754
    mul.s    $f0 $f2 $f1
```


http://www.d.umn.edu/~gshute/spimsal/malref.html#load

mtcx

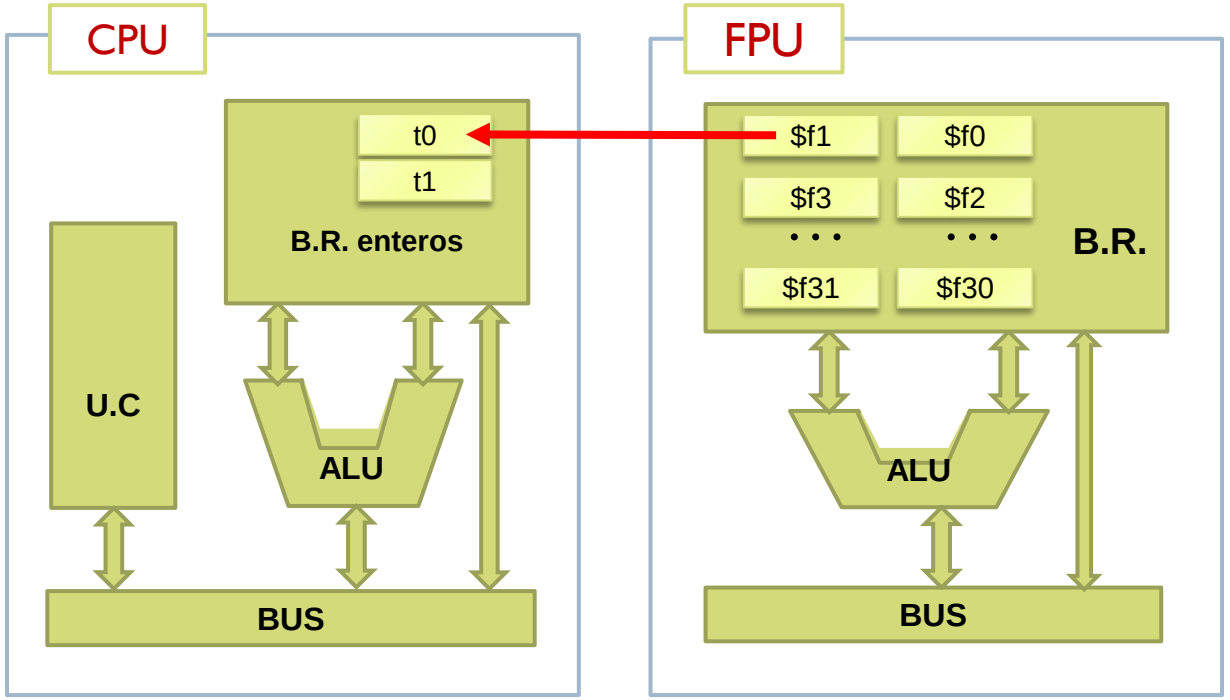
- Transferencias de datos
- Aritméticas
- Lógicas
- Desplazamiento/Rotación
- Control del sistema
- Control de flujo

- Move To Coprocessor 1 (FPU)



mfcx

- Move From Coprocessor 1 (FPU)

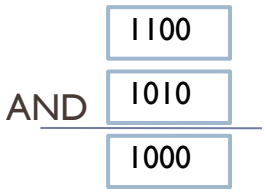


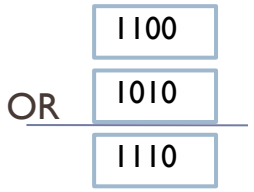
cvt.x.y

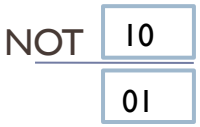
- cvt.s.w $f2 $f1
  - Convierte un entero ($f1) a simple precisión ($f2)
- cvt.w.s $f2 $f1
  - Convierte de simple precisión ($f1) a entero ($f2)
- cvt.d.w $f2 $f0
  - Convierte un entero ($f0) a doble precisión ($f2)
- cvt.w.d $f2 $f0
  - Convierte de doble precisión ($f0) a entero ($f2)
- cvt.d.s $f2 $f0
  - Convierte de simple precisión ($f0) a doble ($f2)
- cvt.s.d $f2 $f0
  - Convierte de doble precisión ($f0) a simple($f2)

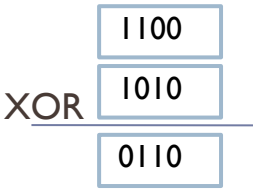
Lógicas

- Operaciones booleanas
- Ejemplos:
  - AND
    ```
    and $t0 $t1 $t2    ($t0 = $t1 && $t2)
    ```
  - OR
    ```
    or  $t0 $t1  $t2   ($t0 = $t1 ||  $t2)
    ori  $0  $t1   80  ($t0 = $t1 ||  80)
    ```
  - NOT
    ```
    not $t0 $t1        ($t0 = ! $t1)
    ```
  - XOR
    ```
    xor  $t0 $t1  $t2    ($t0 = $t1 ^ $t2)
    ```









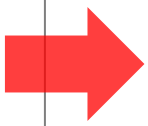
Ejercicio

```
.text
.globl main
 main:

      li $t0, 5
      li $t1, 8

      and $t2, $t1, $t0
```

¿Cuál será el valor en $t2?

Ejercicio

```
.text
.globl main
 main:

      li $t0, 5
      li $t1, 8

      and $t2, $t1, $t0
```



¿Cuál será el valor en $t2?

```
        00...0101   $t0
        00...1000   $t1
and     00...0000   $t2
```

Ejemplo

```
.text
.globl main
 main:

      li $t0, 5
      li $t1, 0x007FFFFF

      and $t2, $t1, $t0
```

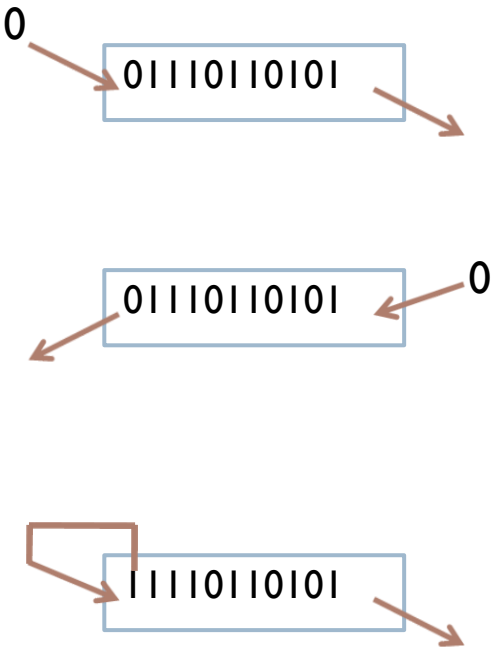
¿Qué me permite hacer un and con 0x007FFFFF?

Quedarme con los 23 bits menos significativos

La constante usada para la selección de bits se denomina máscara.

Desplazamientos

- De movimiento de bits
- Ejemplos:
  - Desplazamiento lógico a la derecha
    srl $t0 $t0 4 ($t0 = $t0 >> 4 bits)
  - Desplazamiento lógico a la izquierda
    sll $t0 $t0 5 ($t0 = $t0 << 5 bits)
  - Desplazamiento aritmético
    sra $t0 $t0 2 ($t0 = $t0 >> 2 bits)



Rotaciones

- De movimiento de bits (2)
- Ejemplos:
  - Rotación a la izquierda
    rol $t0 $t0 4 ($t0 = $t0 >> 4 bits)

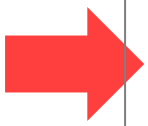
  - Rotación a la derecha
    ror $t0 $t0 5 ($t0 = $t0 << 5 bits)

Ejercicio

Realice un programa que detecte el signo de un número en $t0 y deje en $t1 un 1 si es negativo y un 0 si es positivo

Ejercicio

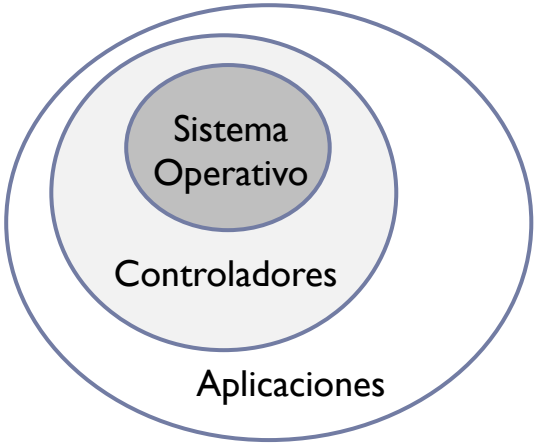
Realice un programa que detecte el signo de un número en $t0 y deje en $t1 un 1 si es negativo y un 0 si es positivo



```
.text
.globl main
 main:

   li   $t0 -3

   move $t1 $t0
   rol  $t1 $t1 1
   and  $t1 $t1 0x00000001
```

Control de Sistemas

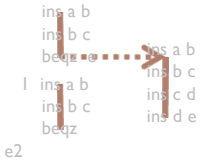
- Instrucciones privilegiadas
- CPU define varios niveles de privilegios



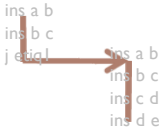
- Ejemplo en Intel:
  - Intel x86 define 4 niveles
  - Sistema operativo en nivel 0
  - AMD-V/i-VT define un nivel -1
  - sysenter (transición al anillo 0)
  - sysexit (transición al anillo 3 desde el anillo 0)

Control de Flujo

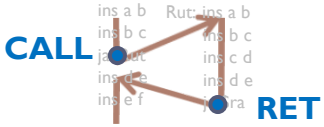
- Cambio de la secuencia de instrucciones a ejecutar (programadas)
- Distintos tipos:
  - Bifurcación o salto condicional:
    - Saltar a la posición x, si valor <=> a y
    - Ej: bne $t0 $t1 etiqueta1



  - Bifurcación o salto incondicional:
    - El salto se realiza siempre
      Ej: j etiqueta2

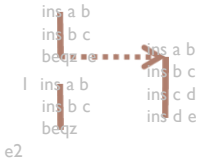


  - Llamada a procedimiento:
    - Salto con vuelta
    - Ej: jal subrutina1 …… jr $ra



Control de Flujo

- Cambio de la secuencia de instrucciones a ejecutar (programadas)
- Distintos tipos:
  - Bifurcación o salto condicional:
    - Saltar a la posición x, si valor <=> a y
    - Ej: `bne $t0 $t1 etiqueta1`



```
beq    $t0   $t1   etiq1   # salta a etiq1 si $t1  = $t0
bne    $t0   $t1   etiq1   # salta a etiq1 si $t1 != $t0
beqz   $t1         etiq1   # salta a etiq1 si $t1  = 0
bnez   $t1         etiq1   # salta a etiq1 si $t1 != 0
bgt    $t0   $t1   etiq1   # salta a etiq1 si $t1 >  $t0
bge    $t0   $t1   etiq1   # salta a etiq1 si $t1 >= $t0
blt    $t0   $t1   etiq1   # salta a etiq1 si $t1 <  $t0
ble    $t0   $t1   etiq1   # salta a etiq1 si $t1 <= $t0
```

Estructuras de control if... (1/3)

```
beq    $t1  = $t0
bnez   $t1  = 0
bne    $t1 != $t0
bgt    $t1 >  $t0
bge    $t1 >= $t0
blt    $t1 <  $t0
ble    $t1 <= $t0
```

```
int a=1;
int b=2;

main ()
{
  if (a < b) {
    a = b;
  }
  ...
}
```

```
        li   $t1 1
        li   $t2 2

if_1:   blt $t1 $t2 then_1
        b    fin_1

then_1: move $t1 $t2

fin_1:  ...
```

Estructuras de control if... (2/3)

```
beq    $t1  = $t0
bnez   $t1  = 0
bne    $t1 != $t0
bgt    $t1 >  $t0
bge    $t1 >= $t0
blt    $t1 <  $t0
ble    $t1 <= $t0
```

```
int a=1;
int b=2;

main ()
{
  if (a < b) {
    a = b;
  }
  ...
}
```

```
        li   $t1 1
        li   $t2 2

if_2:   bge $t1 $t2 fin_2

then_2: move $t1 $t2

fin_2: ...
```

Estructuras de control if... (3/3)

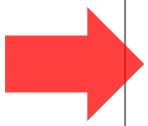
```
beq    $t1  = $t0
bnez   $t1  = 0
bne    $t1 != $t0
bgt    $t1 >  $t0
bge    $t1 >= $t0
blt    $t1 <  $t0
ble    $t1 <= $t0
```

```
int a=1;
int b=2;

main ()
{
  if (a < b){
     // acción 1
  } else {
     // acción 2
  }
}
```

```
        li   $t1 1
        li   $t2 2

if_3:   bge $t1 $t2 else_3

then_3: # acción 1
        b fi_3

else_3: # acción 2

fi_3: ...
```

Ejercicio

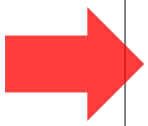
```
int b1 = 4;
int b2 = 2;


 if (b2 == 8) {
     b1 = 1;
 }
 ...
```



```
.text
.globl main
 main:
```

Ejercicio

```
int b1 = 4;
int b2 = 2;


 if (b2 == 8) {
     b1 = 1;
 }
 ...
```



```
.text
.globl main
 main:

        li   $t0 4
        li   $t1 2
        li   $t2 8

        bneq $t0 $t2  fin1
        li   $t1 1
fin1:   ...
```

Estructuras de control while…

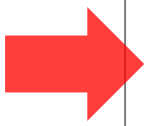
```
beq    $t1  = $t0
bnez   $t1  = 0
bne    $t1 != $t0
bgt    $t1 >  $t0
bge    $t1 >= $t0
blt    $t1 <  $t0
ble    $t1 <= $t0
```

```
int i;

main ()
{
    i=0;
    while (i < 10) {

      /* acción */
      i = i + 1 ;
    }
}
```

```
.text
.globl main
 main:

        li $t0 0
        li $t1 10

while_4:
        bge $t0 $t1 fin_4
        # acción
        addu $t0 $t0 1
        b while_4
fin_4: ...
```

Ejercicio

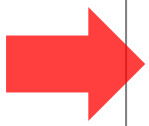
Realice un programa que calcule la suma de los diez primeros números y deje este valor en el registro $v0



```
.text
.globl main
 main:
```

Ejercicio

Realice un programa que calcule la suma de los diez primeros números y deje este valor en el registro $v0



```
.text
.globl main
 main:
    li $v0 0
    add $v0 $v0 1
    add $v0 $v0 2
    add $v0 $v0 3
    add $v0 $v0 4
    add $v0 $v0 5
    add $v0 $v0 6
    add $v0 $v0 7
    add $v0 $v0 8
    add $v0 $v0 9
```

NO vale ;-)

Ejercicio (solución)

Realice un programa que calcule la suma de los diez primeros números y deje este valor en el registro $v0

```
int i, s;

s=i=0;
while (i < 10)
{
    s = s + i ;
    i = i + 1 ;
}
```

```
.text
.globl main
 main:
```

Ejercicio (solución)

Realice un programa que calcule la suma de los diez primeros números y deje este valor en el registro $v0

```
int i, s;

s=i=0;
while (i < 10)
{
    s = s + i ;
    i = i + 1 ;
}
```

```
.text
.globl main
 main:

        li $t0 0
        li $v0 0
while1:
        bge $t0 10 fin1
        add $v0 $v0 $t0
        add $t0 $t0 1
        b while1
fin1:
```

Fallos típicos

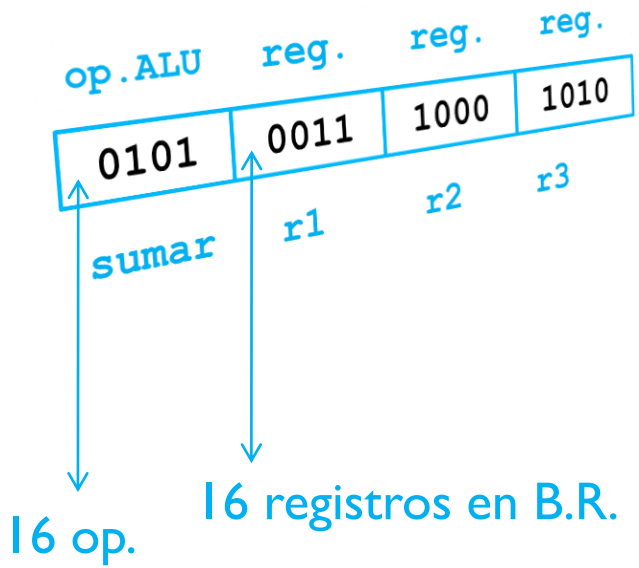
1) Programa mal planteado
   - No hace lo que se pide
   - Hace incorrectamente lo que se pide
2) Programar directamente en ensamblador
   - No codificar en pseudo-código el algoritmo a implementar
3) Escribir código ilegible
   - No tabular el código
   - No comentar el código ensamblador o
     no hacer referencia al algoritmo planteado inicialmente

Contenidos

1. Introducción
   1. Motivación y objetivos
   2. Lenguaje ensamblador y MIPS
2. **Programación en ensamblador**
   1. Arquitectura MIPS (I)
   2. Tipo de instrucciones (I)
   3. **Formato de instrucciones**

Información de una instrucción (1)

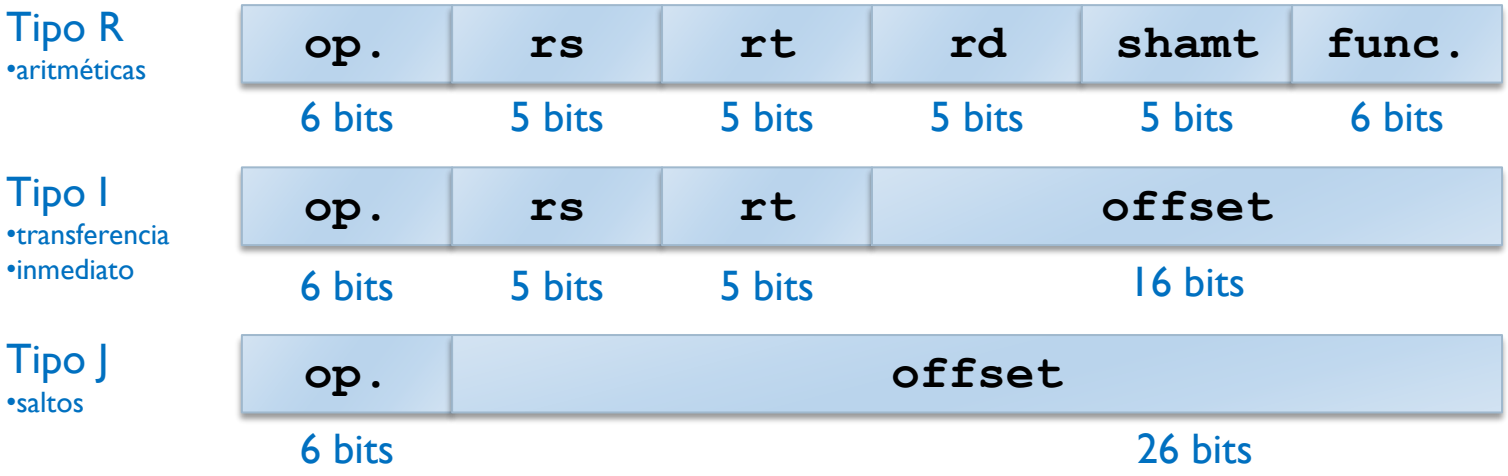
- El tamaño de la instrucción se ajusta al de palabra (o múltiplo)
- Se divide en campos:
  - Operación a realizar
  - Operandos a utilizar
    - Puede haber operando implícitos
- El formato de una instrucción indica los campos y su tamaño:
  - Uso de formato sistemático
  - Tamaño de un campo limita los valores que codifica



Información de una instrucción (2)

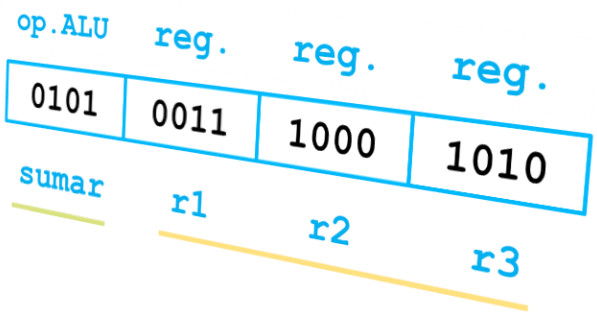
- Se utiliza unos pocos formatos:
  - Una instrucción pertenecen a un formato
  - Según el código de operación se conoce el formato asociado

- Ejemplo: formatos en MIPS

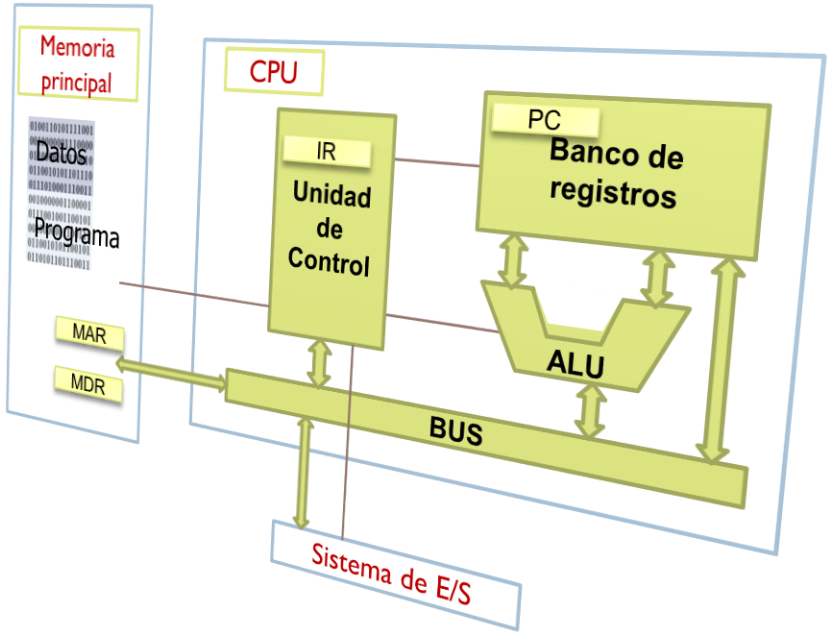


Campos de una instrucción

- En los campos se codifica:
  - Operación a realizar
    - Instrucción y formato de la misma
  - Operandos a utilizar
    - Ubicación de los operandos
    - Ubicación del resultado
    - Ubicación de la siguiente instrucción *(si op. salto)*
      - Implícito: PC ←—— PC + '1' (apuntar a la siguiente instrucción)
      - Explícito: j 0x01004 (modifica el PC)



Ubicaciones posibles



- En la propia instrucción
- En registros de la CPU (UCP)
- En memoria principal
- En unidades de Entrada/Salida (I/O)

Adelanto:
El modo de direccionamiento guía a la U.C. a llegar al operando

Ejercicio

4 minutos máx.

- Sea un computador de 16 bits de tamaño de palabra, que incluye un repertorio con 60 instrucciones máquina y con un banco de registros que incluye 8 registros.

  Se pide:
  Indicar el formato de la instrucción ADDx R1 R2 R3, donde R1, R2 y R3 son registros.

Solución

palabra -> 16 bits
60 instrucciones
8 registros (en BR)
ADDx R1(reg.), R2(reg.), R3(reg.)

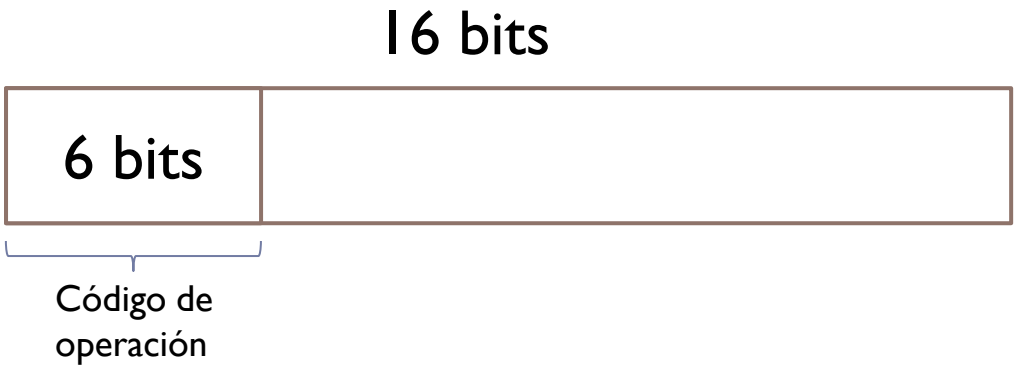
- Palabra de 16 bits define el tamaño de la instrucción

16 bits

Solución

palabra -> 16 bits
60 instrucciones
8 registros (en BR)
ADDx R1(reg.), R2(reg.), R3(reg.)

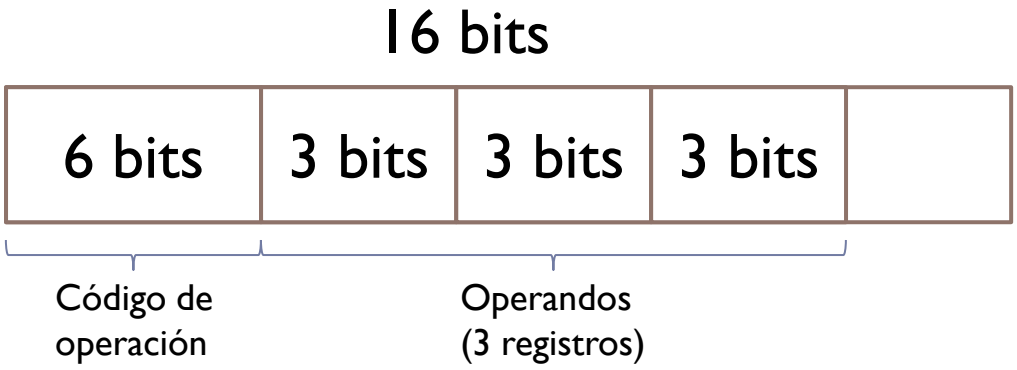
- Supongo un solo formato de instrucciones para todas
- Para 60 instrucciones necesito 6 bits (mínimo)



Solución

palabra -> 16 bits
60 instrucciones
8 registros (en BR)
ADDx R1(reg.), R2(reg.), R3(reg.)

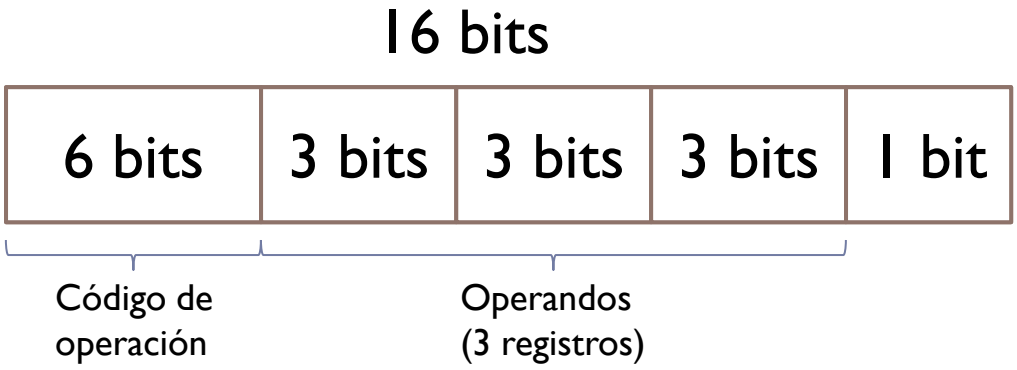
- Para 8 registros necesito 3 bits (mínimo)



Solución

palabra -> 16 bits
60 instrucciones
8 registros (en BR)
ADDx R1(reg.), R2(reg.), R3(reg.)

- Sobra 1 bit (16-6-3-3-3 = 1), usado de relleno



Fallos típicos

1) Formato mal planteado
   - Código de formato de instrucción
   - Código de instrucción
   - Operandos con límites (registros)
   - Operandos sin límites
2) Espacio reservado insuficiente
   - Error al calcular el número de bits para un campo
3) No usar dos palabras cuando es necesario
   - Si uno de los operandos es una dirección o valor inmediato que ocupa una palabra por si mismo

Tema 3 (I)
Fundamentos de la programación en ensamblador

Grupo ARCOS

Estructura de Computadores

Grado en Ingeniería Informática

Universidad Carlos III de Madrid