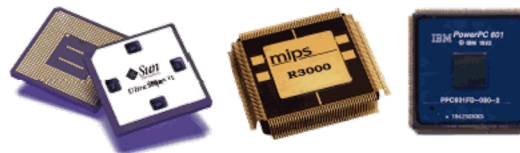


# Tema 2B: Ensamblador MIPS

## Fundamentos de Ordenadores y Sistemas Operativos

Mario Martínez Zarzuela  
marmar@tel.uva.es



# Contenidos

- Nuevas instrucciones para control de flujo
- Operaciones E/S: monitor y el teclado
- Multiplicaciones y divisiones

# Evaluar condiciones del tipo $>$ , $\geq$ , $<$ , $\leq$

- Tipo R: todos los operandos en registros (**r**)

`slt rd, rs, rt      #set rd to 1 if rs<rt`

- Tipo I: incluyen algún dato inmediato (**immediate**)

`slti rd, rs, imm      #set rd to 1 if rs<imm`

- Para control de flujo combinar con beq y bne


# Bucles: completa las instrucciones que faltan

```
for(i=0;i<3;i++){  
    ...  
}
```

```
i:      .data  
        .word    0  
for:    .text  
        lw       $t0,i($zero)  
  
        ...  
        addi     $t0,$t0,1  
        sw       $t0,i($zero)  
        j        for  
endfor: nop
```

# Bucles: completa las instrucciones que faltan

```
for(i=0;i<3;i++){  
    ...  
}
```

```
i:      .data  
        .word    0  
for:    .text  
        lw       $t0,i($zero)  
  
        ...  
        addi     $t0,$t0,1  
        sw       $t0,i($zero)  
        j        for  
endfor: nop
```

# Bucles: completa las instrucciones que faltan

```
for(i=0;i<3;i++){  
    ...  
}
```

```
i:      .data  
        .word    0  
for:    .text  
        lw       $t0,i($zero)  
  
        ...  
        addi     $t0,$t0,1  
        sw       $t0,i($zero)  
        j        for  
endfor: nop
```

# Estrategias de optimización de código (1)

```
for(i=0; i<3; i++){  
    ...  
}
```

**Los accesos a memoria son lentos**, ¿cómo conseguir una ejecución más rápida?

Solución A:

- No actualizar el valor de i en memoria en cada iteración del bucle
- Hacerlo solamente al salir del bucle
- Ahorro tiempo de almacenamiento

```
i:      .data  
        .word    0  
        .text  
for:  lw      $t0, i($zero)  
for:    slti     $t1, $t0, 3  
        beq      $t1, $zero, endfor  
        ...  
        addi     $t0, $t0, 1  
sw      $t0, i($zero)  
        j        for  
endfor: sw      $t0, i($zero)
```

# Estrategias de optimización de código (2)

```
for(i=0; i<3; i++){  
    ...  
}
```

**Los accesos a memoria son lentos**, ¿cómo conseguir una ejecución más rápida?

Solución B (más agresivo):

- No usar memoria para la variable i, ya que es auxiliar
- Ahorro tiempo de carga y de almacenamiento.

```
i: .data  
   .word 0  
   .text  
for: addi $t0,$zero,0      #lw $t0,i($zero)  
     slti $t1,$t0,3  
     beq  $t1,$zero,endifor  
     ...  
     addi $t0,$t0,1  
     sw $t0,i($zero)  
     j    for  
endifor: nop
```



# Laboratorio

- Ejecuta los códigos anteriores para estar seguro de que entiendes las tres variantes:
  - Cargar el valor de  $i$  en cada iteración del bucle, actualizarlo y salvarlo de nuevo (Diapositiva 6)
  - No cargar el valor de  $i$  en cada iteración del bucle y salvarlo solamente a la salida del bucle (Diapositiva 7)
  - No utilizar memoria para el valor de  $i$  (Diapositiva 8)



# Laboratorio

- Modifica los códigos de las diapositivas 6, 7 y 8 para que se utilicen **pseudoinstrucciones** en lugar de las instrucción **slti + beq**
- Comprueba en MARS en qué instrucciones se descompone esta pseudoinstrucción al ensamblar el código

# Bucles para recorrer vectores

## • Vector de cadenas

- Recorrer memoria byte a byte
- **Usar instrucción lb: equivalente a lw, pero carga un byte**
- Hasta encontrar byte con valor 0 (fin de cadena)

Data Segment			
Value (+0)	Value (+4)	Value (+8)	Value (+12)
0    t   x   e   t	\0 \0 \0 \0   o	\0 \0 \0 \0	\0 \0 \0 \0
32   \0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0	\0 \0 \0 \0

```

1      .data
2  cad: .asciiz  "texto"
3
4      .text
5  while: lb    $s1, cad($s0)
6          beq  $s1, $zero, fin
7
8          #...
9          addi $s0, $s0, 1
10         j    while
11  fin:  nop
  
```

## • Vector de enteros

- Recorrer memoria en saltos del número de bytes que ocupe cada dato (enteros ocupan 4 bytes)
- Hasta llegar alcanzar la longitud del vector (en otra variable)

Data Segment			
Value (+0)	Value (+4)	Value (+8)	Value (+12)
0    0    1    2    3			
32   0    0    0    0			

```

1      .data
2  vector: .word    0,1,2
3  veclon: .word    3
4
5      .text
6  add    $s0, $zero, $zero    #contador
7  main:  lw    $s1, veclon($zero) #límite
8  while: bge    $s0, $s1, finwh
9          sll   $t0, $s0, 2
10         lw    $s2, vector($t0)
11
12         #...
13         addi   $s0, $s0, 1
14         j      while
15  finwh: nop
  
```

# Laboratorio

- Ejercicio 2.5

- Carácter '\0' (NULL) tiene el valor 0 en tabla ASCII
- Utiliza el tipo de dato .ascii para inicializar la cadena

**lb**    **rt, imm(rs)**        **# load byte from memory**  
**sb**    **rt, imm(rs)**        **# store byte to memory**

```
1 void main()
2 {
3     char* cad="ensamblador";
4     int lon=0;
5
6     while(cad[lon]!='\0') {
7         lon++;
8     }
9
10 }
11
```

# Laboratorio

- Ejercicio 2.6
  - El valor final almacenado en sum es 27.

```
1 void main()  
2 {  
3     int num=4;  
4     int dato[]={4,9,8,6};  
5     int sum=0;  
6     int cont=0;  
7  
8     while (cont<num)  
9     {  
10        {  
11            sum+=dato[cont++];  
12        }  
13        {  
14            sum = sum + dato[cont];  
15            cont = cont + 1;  
16        }  
17    }  
18 }
```

# Contenidos

- Nuevas instrucciones para control de flujo
- Operaciones E/S: monitor y el teclado
- Multiplicaciones y divisiones

# Operaciones de E/S

- Se consiguen mediante llamadas al sistema
- Configurar registros según tabla e invocar **syscall**

Operación	\$v0 (Cód de función)	\$a0, \$a1 (Argumentos)		\$v0 (Resultado )
print int	1	\$a0 (integer)	syscall	
print string	4	\$a0 (dir. string)		
read int	5			\$v0 (integer)
read string	8	\$a0 (dir. string) \$a1 (longitud)		



# ¿Qué hacen los siguientes códigos?

```
1      .data
2  vector: .word    0,1,2
3  veclon: .word    3
4
5      .text
6  add $s0,$zero,$zero
7  main: lw  $s1,vecclon($zero)
8  while: bge $s0,$s1,finwh
9         sll $t0,$s0,2
10        lw  $s2,vector($t0)
11
12        add $a0, $zero, $s2
13        addi $v0, $zero, 1
14        syscall
15
16        addi $s0,$s0,1
17        j    while
18  finwh: nop
```

```
1      .data
2  vector: .word    0,1,2
3  veclon: .word    3
4
5      .text
6  add $s0,$zero,$zero
7  main: lw  $s1,vecclon($zero)
8  while: bge $s0,$s1,finwh
9         sll $t0,$s0,2
10        lw  $s2,vector($t0)
11
12        move $a0, $s2
13        li   $v0, 1
14        syscall
15
16        addi $s0,$s0,1
17        j    while
18  finwh: nop
```

- Observa estas nuevas **pseudoinstrucciones**

```
move  rx, ry      # copy ry to rx
li    rx, imm     # load immediate
```

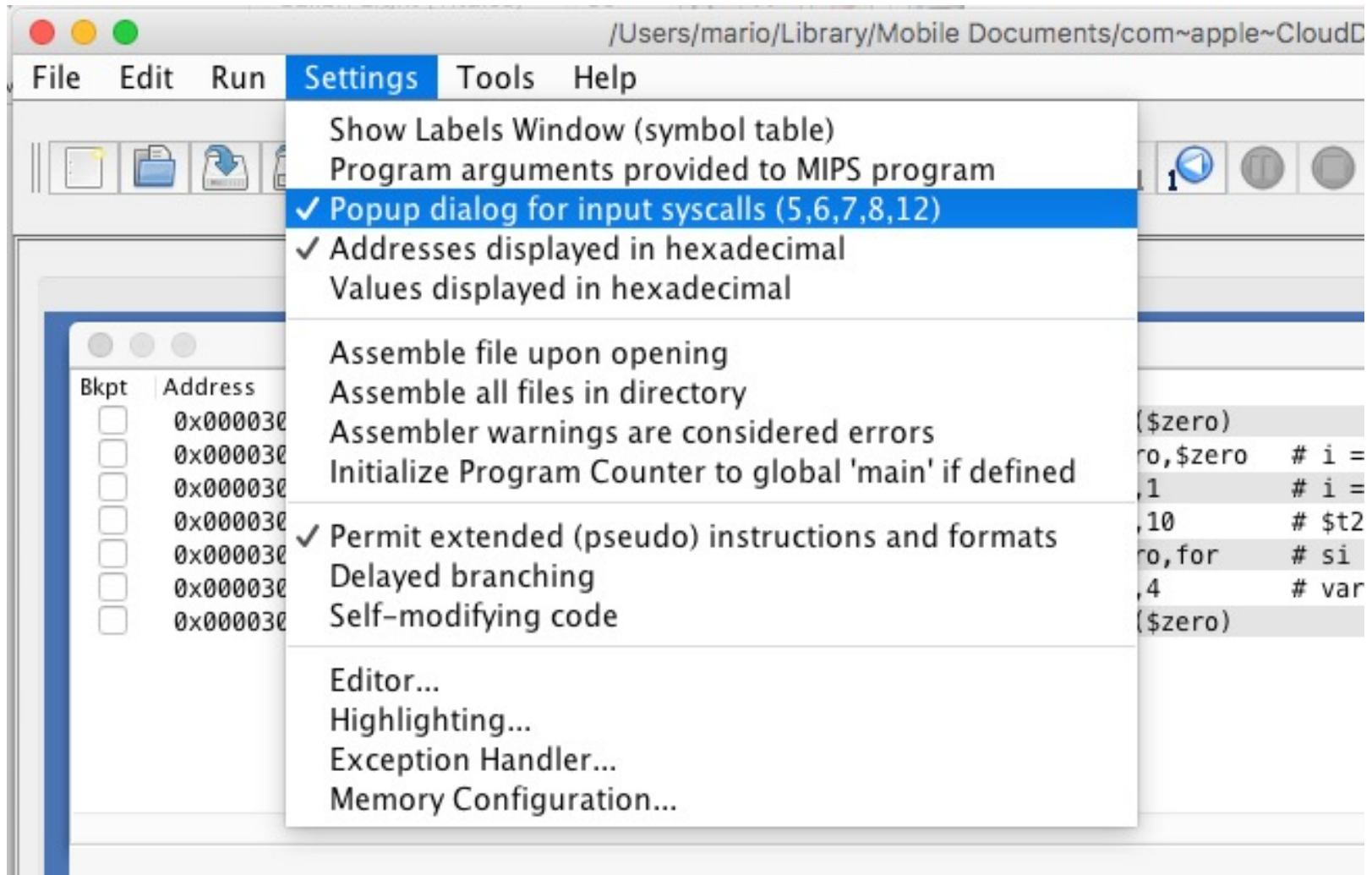
# Operaciones de E/S

- **.space** sirve para reservar un espacio en número de bytes

```
1 void main()  
2 {  
3     char cadena[64];  
4  
5     scanf("%s",cadena);  
6     printf("%s",cadena);  
7 }
```

```
1 .data  
2 cadena: .space 64  
3  
4 .text  
5 main:  
6 #scanf  
7 addi $v0, $zero, 8 #li $v0,8  
8 la $a0, cadena  
9 addi $a1, $zero, 64  
10 syscall  
11 #printf  
12 addi $v0, $zero, 4  
13 syscall  
14 end: nop
```

# Operaciones de E/S



# Laboratorio

- Ejercicio 2.7
  - Modifica el código del Ejercicio 2.3 para que:
    - Se solicite por teclado el valor de  $i$ 
      - Equivaldría a  $\rightarrow$  `scanf("%d", &i);`
    - Se muestre el valor final de  $A[i]$  por pantalla
      - Equivaldría a  $\rightarrow$  `printf("%d", A[i]);`

# Laboratorio

- Ejercicio 2.8
  - Modifica el código del ejercicio 2.7 para que se muestre más información por pantalla:
    - Se solicite por teclado el valor de  $i$ 
      - `printf("\nIntroduce i: ");`
      - `scanf("%d",&i);`
    - Se muestre el valor final de  $A[i]$  por pantalla
      - `printf("\nResultado: %d", A[i]);`

# Contenidos

- Nuevas instrucciones para control de flujo
- Operaciones E/S: monitor y el teclado
- Multiplicaciones y divisiones

# Operaciones de multiplicación y división

- El resultado de estas operaciones se almacena en dos registros especiales: **hi** y **lo**

- Multiplicación

```
mul    rd, rs, rt    # {hi,lo & rd} = rs * rt
mult   rs, rt         # {hi,lo} = rs * rt
```

- División

```
div    rd, rs, rt     # lo & rd = rs / rt
                     # hi = rs % rt
```

- Mover desde hi y lo

```
mfhi   rd             # rd ← hi
mflo   rd             # rd ← lo
```

# Operaciones de multiplicación y división

- Ejemplo de multiplicación

```
1      .data
2  A:   .word    3
3  B:   .word    4
4  C:   .word    0
5
6      .text
7      lw      $s0,A($zero)
8      lw      $s1,B($zero)
9
10     multipl: mult  $s0,$s1
11                mflo $s2
12                sw   $s2,C($zero)
13                nop
```



# Laboratorio

- Ejercicio 2.9
  - Declara dos vectores de números enteros A y B con 3 elementos cada uno (elige los valores que quieras)
  - Declara un vector de enteros C de la misma longitud con sus elementos inicializados a cero
  - En un bucle (para  $i=0,1,2$ ) realiza las siguientes operaciones
    - $C[i] = A[i] * B[i]$
    - Imprimir C[i] por consola

# Laboratorio

- Utiliza la información de la siguiente tabla para separar con espacios los valores de C[i] impresos en consola en el ejercicio anterior

Operación	\$v0	\$a0, \$a1 (Argumentos)	syscall	\$v0 (Resultado )
print character	11	\$a0 (character)		
read character	12			\$v0 (character)

- Para reservar espacio para un caracter utiliza **.byte** en el segmento de datos

```
char:    .data
char:    .byte  'a'
```