Programación de microprocesadores MIPS

Contenidos

- Características generales de los microprocesadores
- Juegos de instrucciones
- Modos de direccionamiento
- Programación práctica en el entorno MIPS

Motivación para aprender ensamblador

```
#include <stdio.h>

int main(int argc, char **argv)
{
  printf ("Hola Mundo\n");
  return (0);
}
```

- Comprender qué ocurre cuando un computador ejecuta una sentencia de alto nivel
- Determinar el tiempo de ejecución de una instrucción de alto nivel
- Útil en: Compiladores, SSOO, Juegos, Sistemas empotrados, etc.

Motivación para usar MIPS

- Arquitectura simple
- Ensamblador similar al de otros procesadores RISC
- Usado en diversos dispositivos

http://www.mips.com/

Diferentes niveles de lenguaje

Lenguaje de alto nivel

(Ej: C, C++)

```
temp = v[k];
v[k] = v[k+1];
v[k+1]=temp;
```

Lenguaje ensamblador

(Ej: MIPS, etc.)

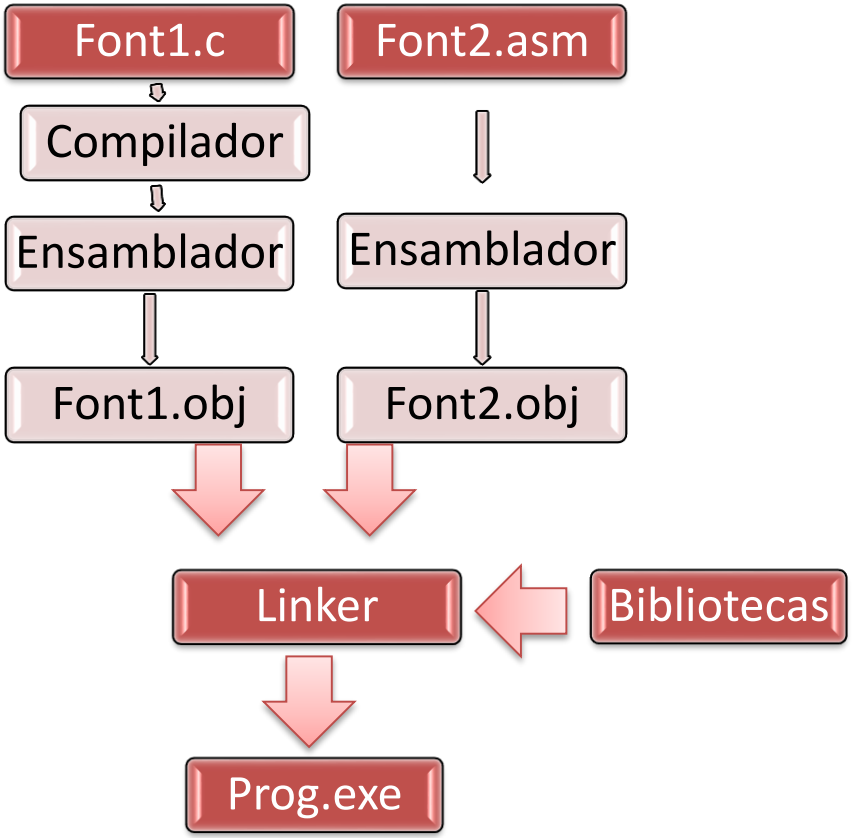
```
lw $t0,0($2)
lw $t1,4($2)
sw $t1,0($2)
sw $t0,4($2)
```

Lenguaje máquina

(Ej: MIPS, etc.)

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 000 10001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

Lenguaje ensamblador



Modelo de programación de un computador

- **Juego de instrucciones** (ensamblador)

  Una instrucción incluye:

  - Código de operación
  - Operandos (registros, valores inmediatos o direcciones de memoria)

Código de operación	Reg. destino	Reg. fuente	Reg. fuente

CAMPOS DE UNA INSTRUCCIÓN

(¡ojo: dependen del tipo de instrucción!)

Modelo de programación de un computador

- **Elementos de almacenamiento**
  - Registros generales (R2, $t2, etc)
  - Memoria (dirección 0x3F4B)
  - Registros de los controladores E/S
- **Modelos de ejecución**
  - Memoria-Memoria
  - Registro-Registro
  - Registro-Memoria
  - Pila

Modelo de memoria

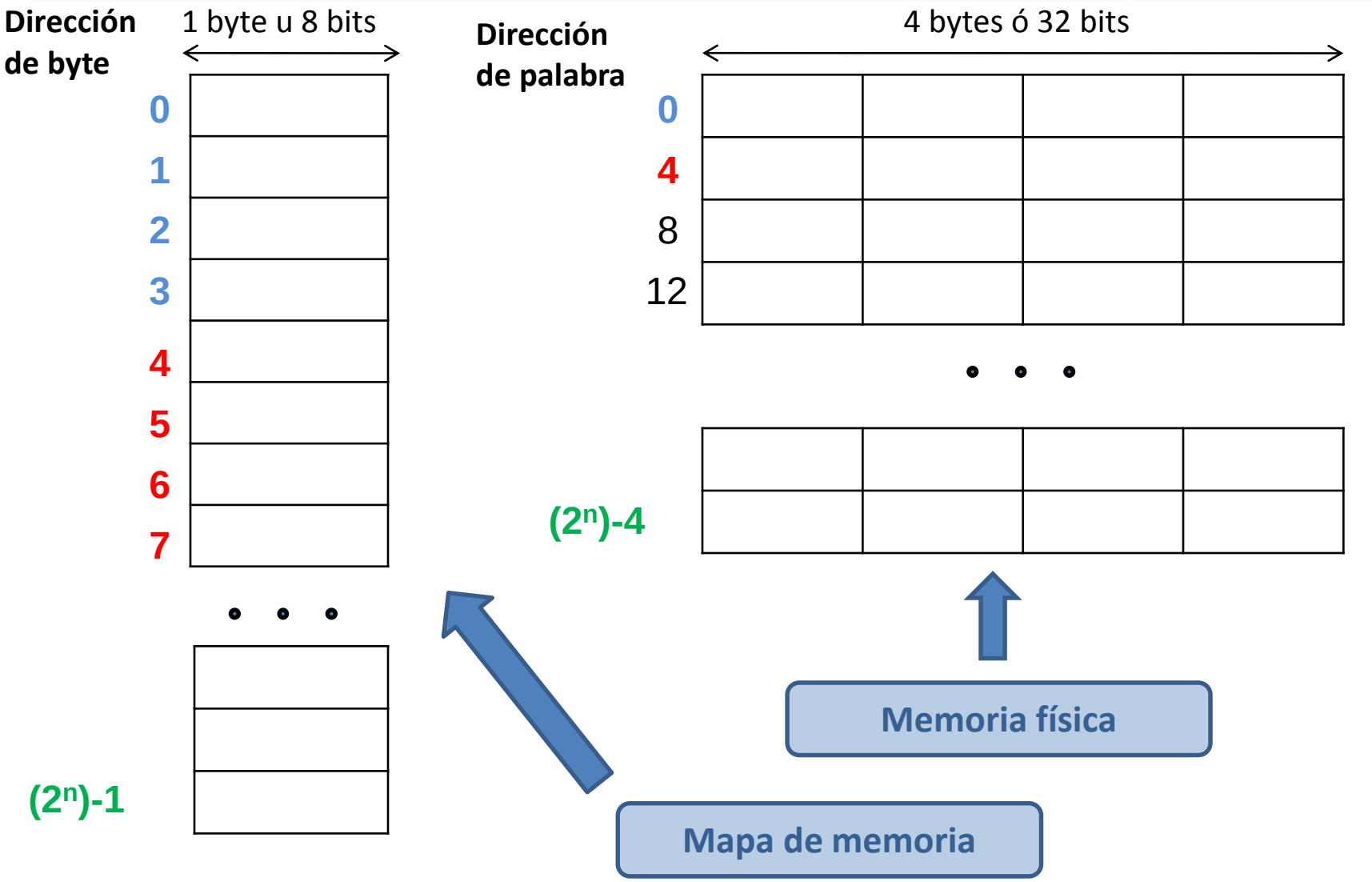
- Direccionamiento por bytes
- Procesador con n bits de direcciones puede manejar 2^n bytes de memoria

  p.e.: MIPS32 → n=32 Direcciona 2^{32} = 4GB de memoria
- Las direcciones generadas por el procesador se referirán siempre a bytes (mapa de memoria)
- Por razones de velocidad el acceso a memoria se realiza por palabras de 32 ó 64 bits (memoria física).

  p.e.: MIPS32 → Dirección de la palabra [A31:A2]

  → Dirección del byte en la palabra [A1:A0]

Espacio de direcciones y memoria física

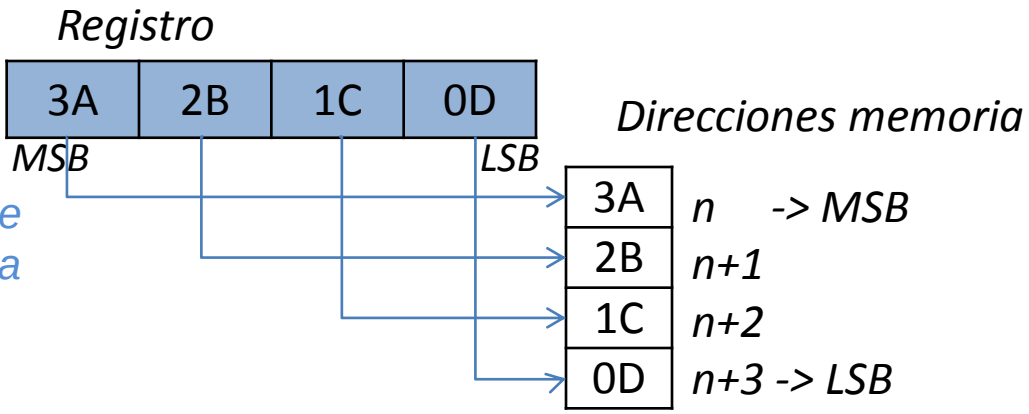


Modelo de memoria

- Ordenamiento de los bytes dentro de una palabra

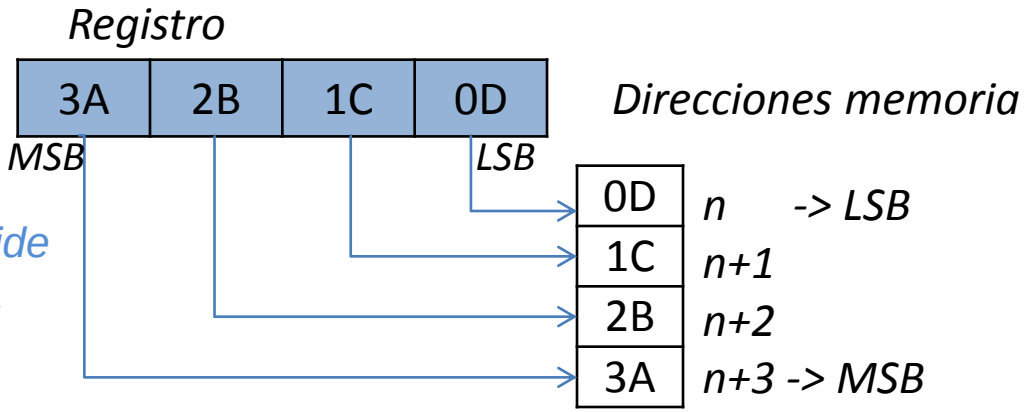
BIG-ENDIAN

Byte de más peso coincide con la dirección de palabra



LITTLE-ENDIAN

Byte de menos peso coincide con la dirección de palabra



Características MIPS (R2000/R3000)

- Procesador de 32 bits
- Arquitectura RISC
- 32 registros de 32 bits (ver manual de prácticas)
- Modelo de ejecución Registro-Registro
- Tres formatos de instrucciones: R, I y J de 32 bits

R	Op.	Rs	Rt	Rd	shamt	func
	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
I	Op.	Rs	Rt	offset		
	6 bits	5 bits	5 bits	16 bits		
J	Op.	Offset				
	6 bits	26 bits				

Características MIPS (R2000/R3000)

- Modelo de memoria
  - Acceso por bytes
  - Palabras de 32 bits
  - Little-Endian
  - Dirección múltiplo de 4 (Restricción de alineamiento)
  - Dispone de un rango de direcciones de memoria reservadas y otras válidas para el usuario-programador
- Acceso a memoria

  modos de direccionamiento: emplea seis para referirse a datos de memoria solo con LOAD y STORE.

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

http://www.mips.com/products/architectures/mips32

Tipos de instrucciones

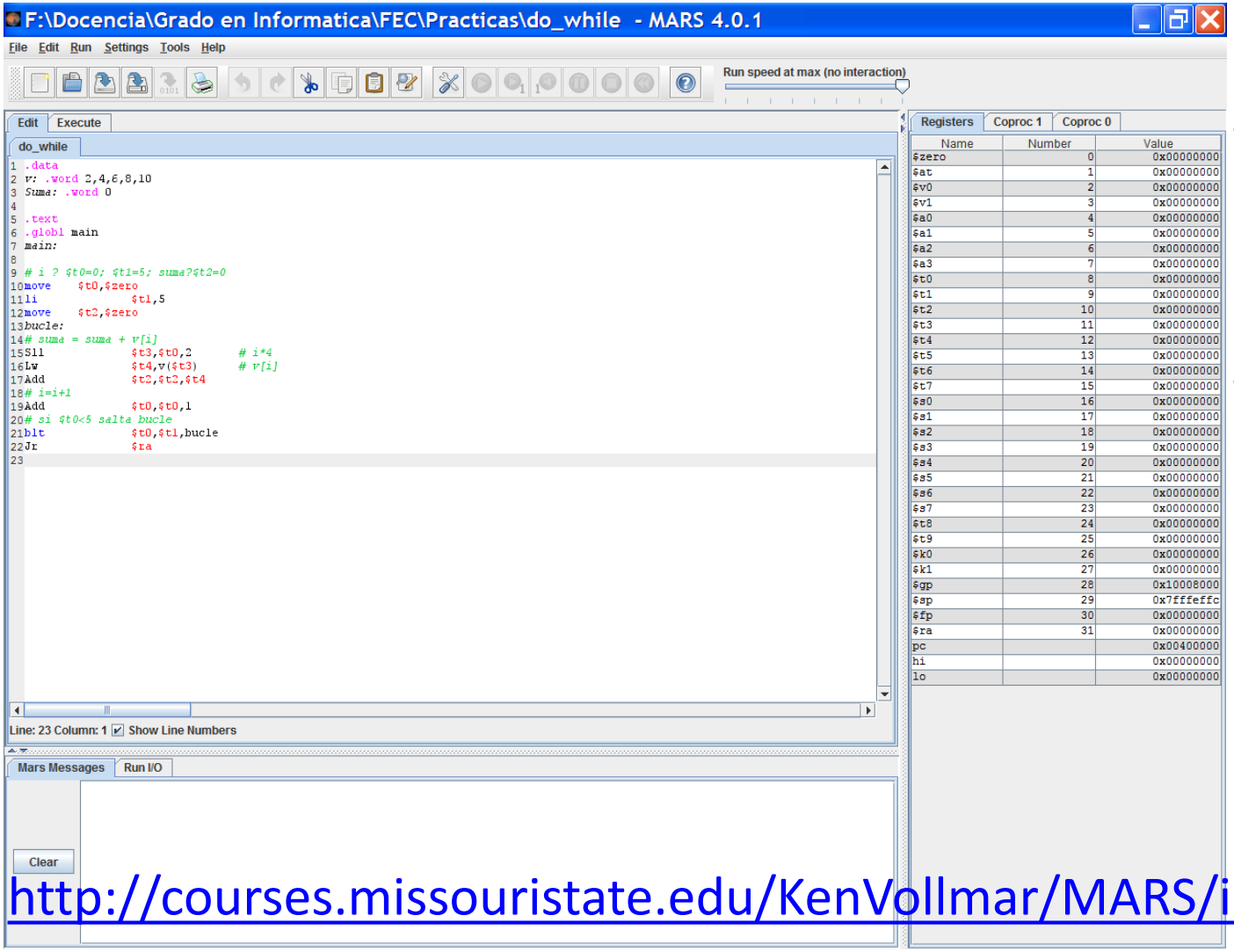
Tipo	Ejemplo
Aritméticas	add Rd,R1,R2 → Rd = R1 + R2
Lógicas	and Rd,R1,R2 → Rd = R1 & R2
Desplazamientos	srl Rd,R1,posiciones
Manipulación de constantes	lui Rt,inm → Rt = inm0..0
Carga	lw Rt,dir → Rt = [dir]
Almacenamiento	sw Rt,dir → [dir] = Rt
Comparaciones	slt Rd,R1,R2 → si R1<R2 Rd = 0x00000001 en caso contrario Rd = 0x00000000
Saltos condicionales	beq R1,R2,etiqueta → si R1=R2 entonces salta a etiqueta
Saltos incondicionales	j etiqueta → salta a etiqueta

Características ensamblador MIPS

- Conjunto de instrucciones soportadas por el hardware
- Pseudoinstrucciones
- Sintaxis básica
- Directivas del ensamblador
- Tipos de datos
- Llamadas al sistema

SPIM , MARS, SIMULA3M

Simulador MARS



- SSOO: Windows Linux
- Necesita Java

http://courses.missouristate.edu/KenVollmar/MARS/index.htm

Lenguaje ensamblador

- Emplea códigos nemónicos para representar instrucciones

  p.e.: add → Suma

  lw → Carga un dato de memoria

- Utiliza nombres simbólicos para referirse a datos, registros y otras referencias

  p.e.: $t0 → Identificador de un registro

- Cada instrucción en ensamblador se corresponde con una instrucción máquina

  p.e.:

  add $t1,$t2,$t3 → 0000 1010 1101 1001 1110 0000 1111 0100

Ejemplos de instrucciones MIPS
- Aritméticas -

- ADD rd, rs, rt → **rd = rs + rt**

0	rs	rt	rd	0	0x20
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Tipo R

$t1 = 0x00000007 y $t2 = 0x00000003

ADD $t0, $t1, $t2 → $t0 = 0x0000000A

- ADDI rt, rs, inmediato → **rt = rs + inmediato**

8	rs	rt	Inmediato
6 bits	5 bits	5 bits	16 bits

Tipo I

$t1 = 0x00000007

ADDI $t0, $t1, 0x2A30 → $t0 = 0x00002A37

Ejemplos de instrucciones MIPS
- Aritméticas -

- SUB rd, rs, rt → **rd = rs - rt**

0	rs	rt	rd	0	0x22
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Tipo R

$t1 = 0x00000007 y $t2 = 0x00000003

SUB $t0, $t1, $t2 → $t0 = 0x00000004

- MULT rs, rt → **HI-LO = rs * rt**

0	rs	rt	0	0x18
6 bits	5 bits	5 bits	10 bits	6 bits

Tipo R

$t1 = 0x00000007 y $t2 = 0x00000003

MULT $t0, $t1 → HI= 0x00000000 LO = 0x00000015

Ejemplos de instrucciones MIPS
- Lógicas -

- AND rd, rs, rt → **rd = rs AND rt**

0	rs	rt	rd	0	0x24
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Tipo R

$t1 = 0x00000007 y $t2 = 0x00000003

AND $t0, $t1, $t2 → $t0 = 0x00000003

- ANDI rt, rs, inmediato → **rt = rs AND inmediato**

0xC	rs	rt	Inmediato
6 bits	5 bits	5 bits	16 bits

Tipo I

$t1 = 0x00000007

ANDI $t0, $t1, 0x0030 → $t0= 0x00000000

Ejemplos de instrucciones MIPS
- Lógicas -

- NOR rd, rs, rt → **rd = rs NOR rt**

0	rs	rt	rd	0	0x27
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Tipo R

$t1 = 0x00000007 y $t2 = 0x00000003

NOR $t0, $t1, $t2 → $t0 = 0xFFFFFFF8

- OR rd, rs, rt → **rd = rs OR rt**

0	rs	rt	rd	0	0x25
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

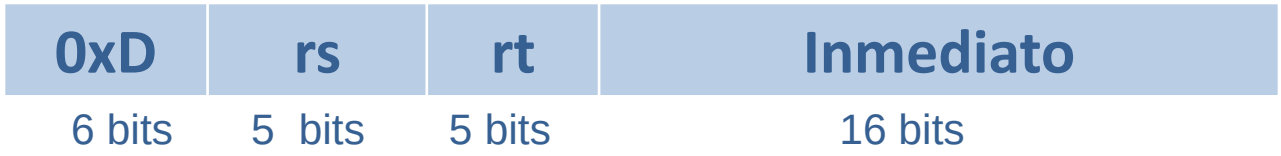
Tipo R

$t1 = 0x00000007 y $t2 = 0x00000003

OR $t0, $t1, $t2 → $t0 = 0x00000007

Ejemplos de instrucciones MIPS
- Lógicas -

- ORI rt, rs, inmediato → **rt = rs OR inmediato**



Tipo I

$t1 = 0x00000007

ORI $t0, $t1, 0x0031 → $t0 = 0x00000031

- XOR rd, rs, rt → **rd = rs XOR rt**

Tipo R

$t1 = 0x00000007 y $t2 = 0x00000003

OR $t0, $t1, $t2 → $t0 = 0x00000004

Ejemplos de instrucciones MIPS
- Lógicas -

- XORI rt, rs, inmediato → **rt = rs XOR inmediato**

0xE	Rs	Rt	Inmediato
6 bits	5 bits	5 bits	16 bits

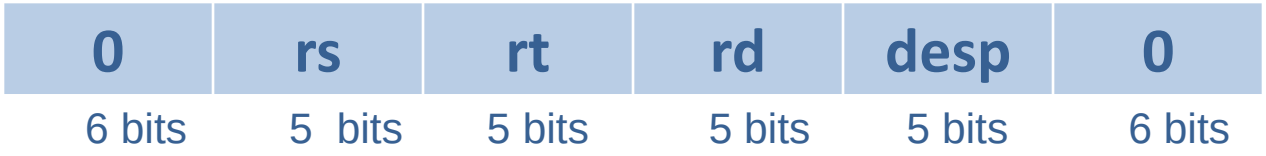
Tipo I

$t1 = 0x00000007

ORI $t0, $t1, 0x0001 → $t0 = 0x00000006

Ejemplos de instrucciones MIPS
- Desplazamientos Lógicos -

- SLL rd, rt, desp → **rd = rt << desp**

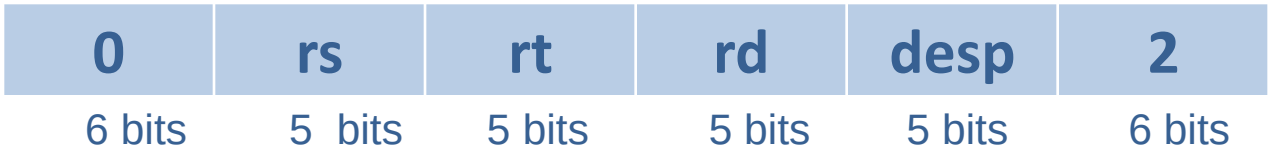


Tipo R

$t1 = 0x00000007

SLL $t0, $t1, 0x01 → $t0= 0x0000000E

- SRL rd, rs, desp → **rd = rt >> desp**



Tipo R

$t1= 0x00000007

SRL $t0, $t1, 0x01 → $t0 = 0x00000003

Ejemplos de instrucciones MIPS
- Desplazamientos Aritméticos -

- SRA rd, rt, desp → **rd = rt >> desp**

  desp: nº de posiciones a desplazar

0	rs	rt	rd	desp	3
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Tipo R

$t1 = 0x00000007

SRA $t0, $t1, 0x01 → $t0 = 0x0000000E

Ejemplos de instrucciones MIPS
- Manipulación de Constantes -

- LUI rt, inm(16 bits) → rt = inm0000000000000000

16 bits 16 bits

0xf	0	rt	inm
6 bits	5 bits	5 bits	16 bits

Tipo I

$t0 = 0x00000007

LUI $t0, 0x001A → $t0 = 0x001A0000

Ejemplos de instrucciones MIPS
- Carga -

- LW rt, dir → **rt = [dir]**

Estudiar los diferentes modos de direccionamiento:

LW $t0, 4($t1)
LW $t0, 0x0000001A
LW etiqueta
…

Tipo I

$t0 = 0x00000007 ; [0x000001A] = 0x1CC0DE03

LW $t0, 0x000001A → $t0 = 0x1CC0DE03

Ejemplos de instrucciones MIPS
- Almacenamiento -

- SW rt, dir → [dir] = rt

Estudiar los diferentes modos de direccionamiento:

SW $t0, 4($t1)
SW $t0, 0x0000001A
SW etiqueta
…

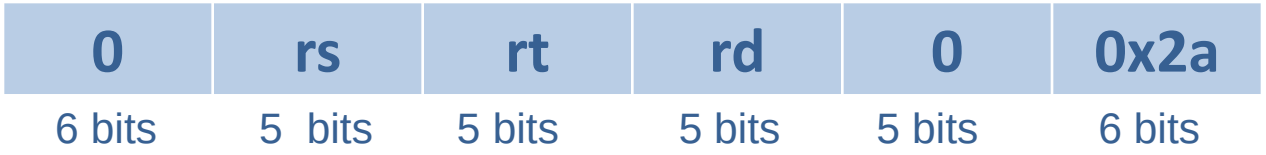
Tipo I

$t0 = 0x00000007 ; [0x0000001A] = 0x00000000

SW $t0, 0x0000001A → [0x0000001A] = 0x00000007

Ejemplos de instrucciones MIPS
- Comparaciones -

- SLT rd, rs, rt → si rs < rt rd = 0x00000001
  si no rd = 0x00000000

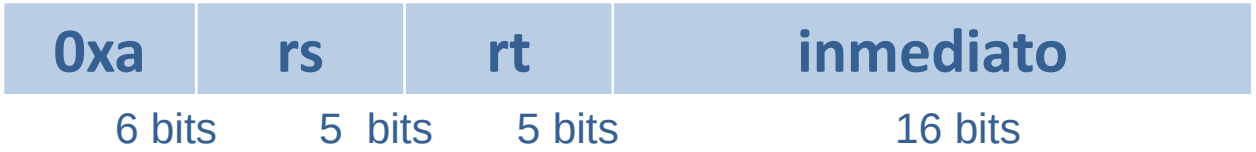


Tipo R

$t1 = 0x00000007 $t2 = 0x00000002

SLT $t0, $t1, $t2 → $t0 = 0x00000000

- SLTI rt, rs, inmediato → si rs < inm rd = 0x00000001
  si no rd = 0x00000000



Tipo I

$t1=0x00000007

SLTI $t0, $t1,0x000B → $t0 = 0x00000001

Ejemplos de instrucciones MIPS
- Saltos Condicionales -

- BEQ rs, rt, etiqueta → si rs=rt salta a etiqueta
  PC = PC + desp *4

0x4	rs	rt	Desplazamiento
6 bits	5 bits	5 bits	16 bits

Tipo I

$t1 = 0x00000007 $t2 = 0x00000007

BEQ $t0, $t1, inicio → salta a inicio

desp representa el nº de instrucciones que hay que avanzar o retroceder

- BNE rs, rt, etiqueta → si rs<>rt salta a etiqueta
  PC = PC + desp *4

0x5	rs	rt	Desplazamiento
6 bits	5 bits	5 bits	16 bits

Tipo I

$t1 = 0x00000007 $t2 = 0x00000005

BNE $t0, $t1, inicio → salta a inicio

Ejemplos de instrucciones MIPS
- Saltos Condicionales -

- BGEZ rs,etiqueta → si rs >= 0 salta a etiqueta

  PC = PC + desp *4

1	rs	1	Desplazamiento
6 bits	5 bits	5 bits	16 bits

Tipo I

$t0 = 0x00000007 etiq1 → desplazamiento = +10

BGEZ $t0, etiq1 → PC = PC+10*4

Ejemplos de instrucciones MIPS
- Saltos Incondicionales -

- J dir → PC = dir

2	Dirección
6 bits	26 bits

Tipo J

J inicio → PC = inicio

- JAL dir → 1º → ra = PC+4 2º → PC= dir

3	Dirección
6 bits	26 bits

Tipo J

JAL inicio → ra= PC+4; PC=inicio

Ejemplos de instrucciones MIPS
- Saltos Incondicionales -

- JR rs → PC = rs

0	rs	0	8
6 bits	5 bits	15 bits	6 bits

Tipo R

$t1 = 0x0000000F

JR $t1 → PC = 0x0000000F

Ejemplos de pseudoinstrucciones MIPS

- LA rt, dir → rt = dir

  32 bits

  LUI rt, dir alta (se almacena la parte alta)

  ORI rs, rt, inmediato (se almacena la parte baja)

```
LA $t1,0x10010001
LUI $t1,0x1001        → $t1 = 0x1001000
ORI $t1,$t1,0x0001    → $t1 = 0x10010000 OR 0x00000001
```

- LI rt, inmediato → rt = inmediato

  32 bits

  LUI rt, inmediato (se almacena la parte alta)

  16 bits

  ORI rs, rt, inmediato (se almacena la parte baja)

  16 bits

Sintaxis básica (Mars)

- Una sentencia por línea
- Las etiquetas van seguidas por dos puntos
- Nomenclatura de registros: $t1=$9
- Las cadenas de caracteres van entre “ “
- Representación decimal de números por defecto
- Para comentar líneas se usa: #
- Representación hexadecimal de números: **0x**010f3a5e

Directivas del ensamblador (MARS)

Directiva	Función
.text	Indica el inicio de código
.data	Indica el inicio de la declaración de datos globales
.globl	Convierte una etiqueta en global

```
.data                                   #Declaración de variables
     Declaración de variables

.text                                   #Comienzo del código
.globl main                             # Hace global la etiqueta “main”
main:                                   # Etiqueta obligatoria que marca el inicio
     Inicio de las instrucciones
```

Tipos de datos (Directivas. Anexo A3)

Directiva de datos	Tipo
.ascii	Cadena de caracteres
.asciiz	Cadena de caracteres + Carácter terminador
.float	Flotante de 32 bits
.double	Flotante de 64 bits
.space	Reserva de espacio en memoria
.word	Reserva una palabra de memoria para el dato especificado

Tipos de datos (Directivas)

```
.data                                   #Declaración de variables
    cadena:  .ascii   "HOLA"            #cadena="HOLA"  (32 bits)
    otra:    .asciiz  "HOLA"            #otra="HOLA" + 00 (40 bits)
    espacio: .space   3                 #3 bytes de memoria "en blanco"
    palabra: .word    124               #palabra=124 (32 bits)
    simple:  .float   12.4              #simple=12.4 (32 bits)
    doble:   .double  12.4              #doble=12.4 (64 bits)

.text                                   #Comienzo del código
.globl main                             #Donde empieza el programa
main:
    Inicio de las instrucciones
```

Tipos de datos (Directivas)

ASCII: "H" = 68 "O" = 6F "L" = 6C "A" = 61

Little endian

ETIQUETA	DIR. MEMORIA	DATO EN MEM.	BYTES
cadena(.ascii)	1001 0000	616C 6F68	4
otra (.asciiz)	1001 0004 1001 0008	616C 6F68 0000 0000	5
espacio (.space)			3
palabra (.word)	1001 000C	0000 007C	4
simple (.float)	1001 0010	4146 6666	4
doble (.double)	1001 0014 1001 0018	CCC CCCD 4028 CCCC	8

Llamadas al sistema (Syscall) Anexo A4

Directiva de datos	Código	Tipo
Print_int	1	Imprime como número entero lo que hay en el registro $a0
Print _string	4	Imprime como string lo que hay en la dirección guardada en $a0
Read_int	5	Solicita un entero que se almacenará en $v0
Read_String	8	Solicita un string que se almacena en $a0 y cuya longitud se guarda en $a1
Exit	10	Finaliza la ejecución

Llamadas al sistema (Syscall)

```
.data
texto: .asciiz "Hola Mundo“

.text
.globl main
main:
 la $a0,texto

#Llamada a PRINT_STRING
 li $v0,0x00000004
 syscall
```

```
#Llamada a PRINT_INT
 li $v0,0x00000001
 syscall

#Llamada a READ_INT
 li $v0,0x00000005
 syscall

#Llamada a EXIT
 li $v0,0x0000000A
 syscall
```

Registros MIPS (MARS)

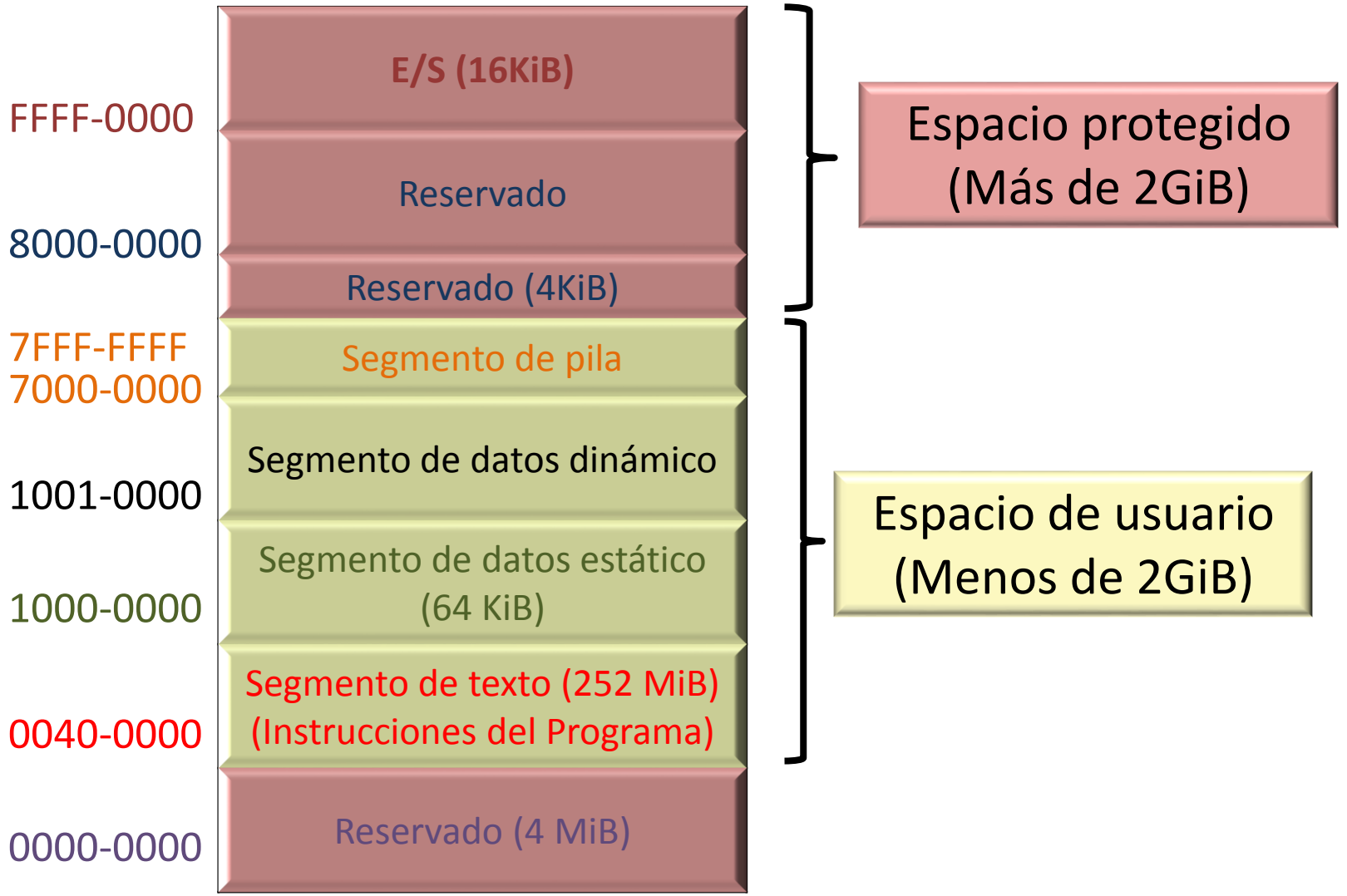
Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000000
$s1	17	0x00000000
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000
$t8	24	0x00000000
$t9	25	0x00000000
$k0	26	0x00000000
$k1	27	0x00000000
$gp	28	0x10008000
$sp	29	0x7fffeffc
$fp	30	0x00000000
$ra	31	0x00000000
pc		0x00400000
hi		0x00000000
lo		0x00000000

$zero	0	Valor 0
$at	1	Reservado
$v0-$v1	2-3	Resultados de expresiones
$a0-a3	4-7	Argumentos
$t0-t9	8-15 24-25	Registros temporales
$s0-s7	16-23	Registros salvados
$k0-$k1	26-27	Reservado S.O.
$gp	28	Puntero global
$sp	29	Puntero de pila
$fp	30	Puntero de marco de pila
$ra	31	Dirección de retorno

PC	Contador de programa
Hi-Lo	Unidos permiten 64bits

Memoria MIPS (MARS)



Memoria MIPS

```
.data
texto: .asciiz "Hola
Mundo"
#####################


.text
.globl main
main:

  la $a0,texto
  li $v0,0x00000004
  syscall
```

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)
0x10010000	0x616c6f48	0x6e754d20	0x00006f64
0x10010020	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000

ASCII: "h" = 48 "O" = 6F "L" = 6C "A" = 61 .....

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x3c011001	lui $1,0x1001	9: la $a0,texto
	0x00400004	0x34240000	ori $4,$1,0x0000	
	0x00400008	0x24020004	addiu $2,$0,0x0004	10: li $v0,0x00000004
	0x0040000c	0x0000000c	syscall	11: syscall

Estructuras de programación (MIPS)

1. Decisiones (IF)

```
.data
x:      .word 1
.text
.globl main
main:
...
#si   x>=0   salta a fin_if
lw      $t0,x
bgez   $t0,fin_if

#bloque   if
sw     $zero,x

fin_if:
...
```

```
Int x=1;

main()
 {
    ...
    If (x<0)
    {
        x=0;
    }
    ...
 }
```

Estructuras de programación (MIPS)

1. Decisiones (IF-ELSE)

```
.data
x:      .word 2
y:      .word 3
z:      .space 4

.text
.globl main
main:
```

```
main:
...
# si  x<=y  salta a else
lw      $t0,x
lw      $t1,y
ble     $t0,$t1,else

# bloque  if
sw      $t0,z

# termina bloque if
j       fin_if

# bloque  else
else:
sw      $t1,z
fin_if:
...
```

```
int x=2, y=3;
int z;
main()
{
    ...
    if (x>y)
    {
        z=x;
    }
    else
    {
        z=y;
    }
    ...
}
```

Estructuras de programación (MIPS)

2. Bucles (DO-WHILE)

```
.data
v: .word 2,4,6,8,10
suma: .word 0

.text
.globl main
main:
```

```
# Acumulador
move        $t2,$zero
# Longitud del vector
li          $t1,5
# Indice del vector
move        $t0, $zero

bucle:
# bloque do-while
sll         $t3,$t0,2    # i*4
lw          $t4,v($t3)   # v[i]
add         $t2,$t2,$t4
add         $t0,$t0,1
# si $t0<5 salta a bucle
blt         $t0,$t1,bucle
…
```

```
int v[5]={2,4,6,8,10};
int suma=0;

main()
{
    int i=0;
    do {
        suma+=v[i];
        i=i+1;
    } while (i<5);
    …
}
```

Estructuras de programación (MIPS)

3. Bucles (WHILE)

```
.data
v: .word 2,4,6,8,10
suma: .word 0

.text
.globl main
main:
```

```
# Acumulador
move        $t2,$zero
# Longitud del vector
li          $t1,5
# Indice del vector
move        $t0, $zero
bucle:  # bloque while
# si $t0>=5 no iterar más.
bge         $t0,$t1, fin
sll         $t3,$t0,2    # i*4
lw          $t4,v($t3)   # v[i]
add         $t2,$t2,$t4
add         $t0,$t0,1
j bucle     # reiterar
fin:   …
```

```
int v[5]={2,4,6,8,10};
int suma=0;

main()
{
    int i=0;
 while (i<5) {
        suma+=v[i];
        i=i+1;
    }
    …
 }
```

Estructuras de programación (MIPS)

4. Bucles (FOR)

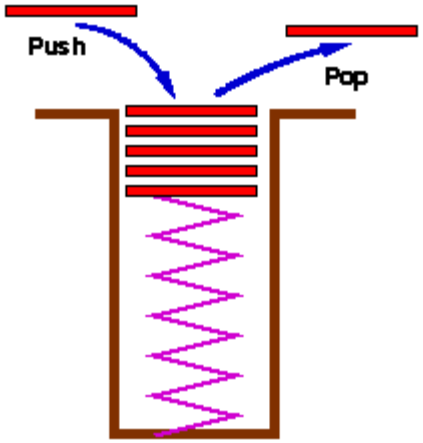
```
.data
v: .word 2,4,6,8,10
suma: .word 0

.text
.globl main
main:
```

```
# Acumulador
move        $t2,$zero
# Longitud del vector
li          $t1,5
# Indice del vector
move        $t0, $zero
bucle:
bge         $t0,$t1,fin_bucle
# bloque  for
sll         $t3,$t0,2    # i*4
lw          $t4,v($t3)   # v[i]
add         $t2,$t2,$t4
add         $t0,$t0,1
# salta a bucle
j           bucle
fin_bucle:
...
```

```
int v[5]={2,4,6,8,10};
int suma=0; int i;
main()
 {
     for(i=0;i<5;i++)
     {
          suma+=v[i];
      }
      ...
}
```

La pila (*stack*)

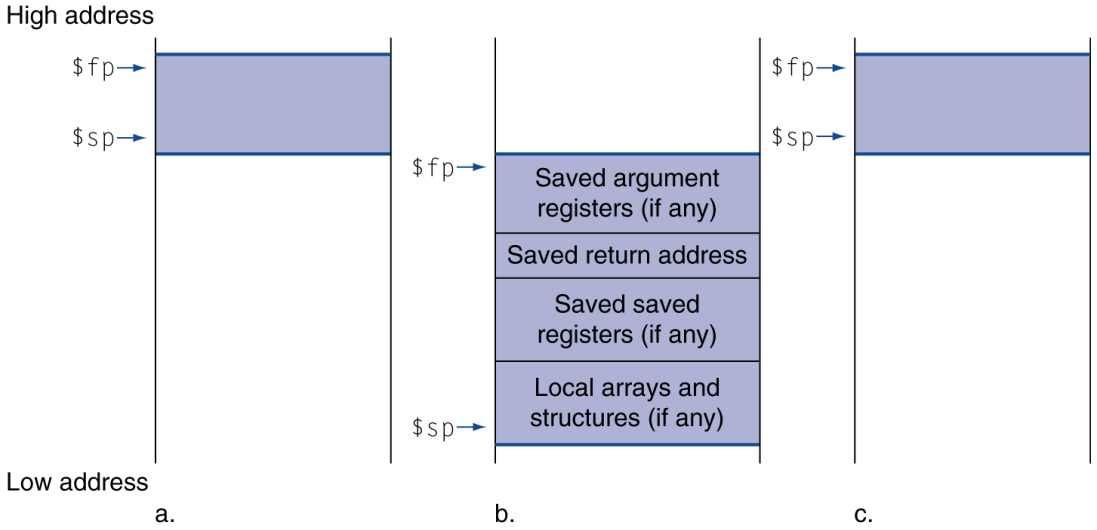


Pila: Espacio en memoria para almacenar datos temporales de forma dinámica.

LIFO = Last in, first out: El último dato en entrar es el primero en salir.

PUSH:
addi $sp, $sp, -4
sw $s1, 0($sp)

POP:
lw $s1, 0($sp)
addi $sp, $sp, +4



Rutinas/funciones (MIPS)

1. Llamada a rutina o función: versión sencilla

Código que ejecuta la llamada:

1. Guardar parámetros en registros ($a0..3).
2. Transferir control a la rutina: instrucción `JAL` (jump and link).

Después de la llamada:

Continuar el programa.

Función a la que se llama:

1. Ejecutar operaciones de la rutina (Con operandos en $a0..3).
2. Guardar result. en reg. $v0, $v1
3. Volver al punto de llamada con `JR $ra`

Rutinas/funciones (MIPS)

2. Llamada a rutina o función: versión (casi) completa

Código que ejecuta la llamada:

1. Guardar parámetros en registros ($a0..3) [y en *pila*].
2. Salvaguardar registros temporales ($t0..9)
3. Transferir control a la rutina: instrucción `JAL` (jump and link).

Después de la llamada:

1. Recuperar registros salvaguardados ($t0..9).

Función a la que se llama:

1. Salvaguardar registros que se vayan a modificar ($s0..7, ...)
2. Ejecutar operaciones de la rutina.
3. Guardar result. en reg. $v0, $v1
4. Recuperar registros salvaguardados ($s0..7, ...).
5. Volver al punto de llamada con `JR $ra`

Rutinas/funciones (MIPS)

3. Función que llama a otra función (o a sí misma)

- Es una función que es llamada por otra. Por tanto tendrá que hacer las operaciones propias de las rutinas:
  - Salvaguardar los registros $s0..7 que vaya a utilizar, y restaurarlos después de haber calculado los resultados.
  - Guardar resultados en $a0..3
- También es una función que llama a otras, y por tanto tendrá que hacer las operaciones relacionadas con una llamada a función:
  - Antes de llamar a otra función, salvaguardar los registros temporales $t0..9 y que utilice, y restaurarlos después de la llamada a función.
- **Y además tendrá que salvaguardar también los registros que intervienen en llamadas: $a0..3 (si fuera necesario), y $ra (siempre)**.

Rutinas/funciones (MIPS)

4. Ejemplo: factorial (implementación recursiva)

```
fact:
    addi $sp, $sp, -8       # adjust stack for 2 items
    sw   $ra, 4($sp)        # save return address
    sw   $a0, 0($sp)        # save argument
    slti $t0, $a0, 1        # test for n < 1
    beq  $t0, $zero, L1
    addi $v0, $zero, 1      # if so, result is 1
    addi $sp, $sp, 8        #   pop 2 items from stack
    jr   $ra                #   and return
L1: addi $a0, $a0, -1       # else decrement n
    jal  fact               # recursive call
    lw   $a0, 0($sp)        # restore original n
    lw   $ra, 4($sp)        #   and return address
    addi $sp, $sp, 8        # pop 2 items from stack
    mul  $v0, $a0, $v0      # multiply to get result
    jr   $ra                # and return
```