Arquitecturas y Organización de Computadoras I

2: Arquitectura MIPS

Rafael Ignacio Zurita

Depto. Ingeniería de Computadoras

September 16, 2020

Arquitectura MIPS

- Máquina MIPS general
- Hardware visible al programador: memoria y registros
- Cuando utilizar lenguaje ensamblador
- Conjunto de instrucciones
- Modos de direccionamiento
- lenguaje ensamblador vs lenguaje máquina

Arquitectura MIPS

- Máquina MIPS general
- Hardware visible al programador: memoria y registros
- Cuando utilizar lenguaje ensamblador
- Conjunto de instrucciones
- Modos de direccionamiento
- lenguaje ensamblador vs lenguaje máquina

» Introducción a MIPS

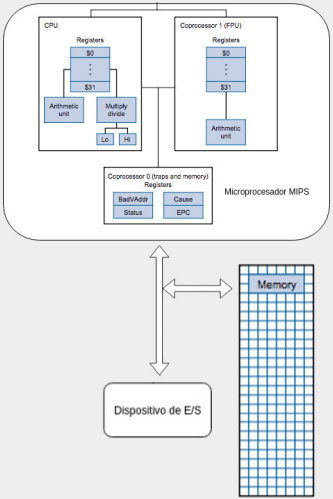


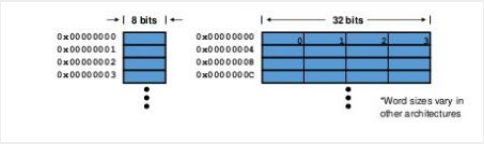
Diagrama de Bloques de una computadora MIPS

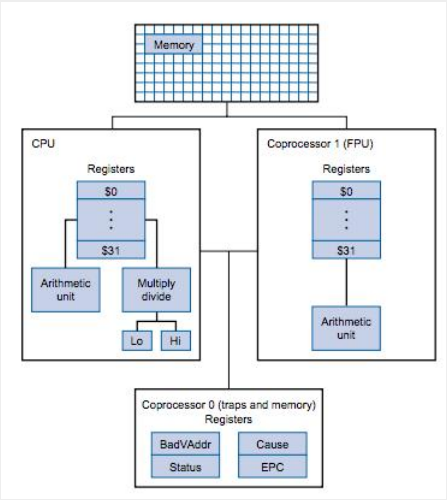
Arquitectura MIPS

- Máquina MIPS general
- Hardware visible al programador: memoria y registros
- Cuando utilizar lenguaje ensamblador
- Conjunto de instrucciones
- Modos de direccionamiento
- lenguaje ensamblador vs lenguaje máquina

» Organización de la Memoria principal en MIPS

- La memoria total máxima posible es de **4GB**.
- 2^{32} La memoria es **byte direccionable** (cada byte tiene una dirección de memoria 0, 1, 2 ...)
- 2^{30} palabras de 4-bytes, con direcciones 0, 4, 8 ...
- Las direcciones de datos con tamaño de palabra (word, float) deben ser múltiplos de 4.
- Las direcciones de las medias palabras (half) deben ser múltiplos de 2.
- Las direcciones de las dobles palabras (ej. double) deben ser múltiplos de 8.





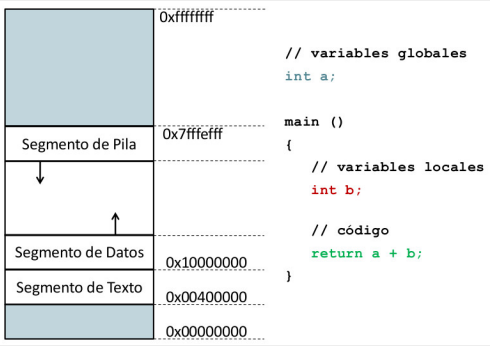
» Organización de la Memoria principal en MIPS

Un programa en memoria está dividido en segmentos lógicos, para organizar su contenido.

Código En el **segmento de código** se ubican las instrucciones máquina del programa.

Datos En el **segmento de datos** se ubican las variables globales del programa.

Pila En el **segmento de pila** se ubican las variables locales a una función.



» Registros dentro del procesador

Registros visibles al usuario.

- 32 Existen 32 registros de propósito general.
- 4 bytes Cada registro tiene el tamaño de la palabra en la arquitectura (32bits/4 bytes).
- Uso Se los menciona en lenguaje ensamblador colocando un signo $ (ej. $t2, o $5).

Otros registros.

- Existen muchos otros registros, pero son parte de la microarquitectura (buffers, PC, EPC, Cause, etc).

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)

Arquitectura MIPS

- Máquina MIPS general
- Hardware visible al programador: memoria y registros
- **Cuando utilizar lenguaje ensamblador**
- Conjunto de instrucciones
- Modos de direccionamiento
- lenguaje ensamblador vs lenguaje máquina

Arquitectura MIPS

- Máquina MIPS general
- Hardware visible al programador: memoria y registros
- Cuando utilizar lenguaje ensamblador
- **Conjunto de instrucciones**
- Modos de direccionamiento
- lenguaje ensamblador vs lenguaje máquina

» Tipos de Instrucciones

Tipos Principales de Instrucciones

- Instrucciones de transferencia de datos (memoria a registro o viceversa)
  - Carga y Almacenamiento
- Aritméticas y Lógicas
  - Enteros
  - Punto Flotante
- Flujo de control
  - Salto
  - Bifurcación condicional
  - Llamado y Retorno

» Instrucciones de transferencia de datos

- Permiten transferir datos entre la memoria y los registros del procesador:
  - **lw**: cargar una palabra desde la memoria a un registro
  - **sw**: almacenar una palabra desde un registro a la memoria
  - **lb**: cargar un byte desde la memoria a un registro (extendiendo el signo del byte)
  - **lbu**: cargar un byte desde la memoria a un registro (sin extender el signo)
  - **sb**: almacenar un byte desde un registro a la memoria
  - **lh**: cargar media palabra desde la memoria a un registro
  - **sh**: almacenar media palabra desde un registro a la memoria
- Formatos posibles (en lenguaje ensamblador):

```
.text


lw $3, etiqueta
lw $3, etiqueta + 23   # etiqueta + constante numérica
lw $3, 23($2)   # base (contenido del registro) + desplazamiento
lw $3, ($2)   # base (contenido del registro) sin desplazamiento
lw $3, 0x10004   # una dirección directa
lw $3, etiqueta+23($4) # base (contenido del registro) + etiqueta + desplazamiento
```

» Programación de la máquina: lenguaje ensamblador MIPS

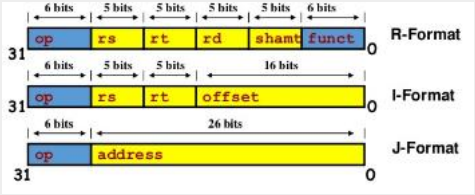
Conjunto básico de instrucciones MIPS

Instruction	Usage
Load upper immediate	`lui rt,imm`
Add	`add rd,rs,rt`
Subtract	`sub rd,rs,rt`
Set less than	`slt rd,rs,rt`
Add immediate	`addi rt,rs,imm`
Set less than immediate	`slti rd,rs,imm`
AND	`and rd,rs,rt`
OR	`or rd,rs,rt`
XOR	`xor rd,rs,rt`
NOR	`nor rd,rs,rt`
AND immediate	`andi rt,rs,imm`
OR immediate	`ori rt,rs,imm`
XOR immediate	`xori rt,rs,imm`
Load word	`lw rt,imm(rs)`
Store word	`sw rt,imm(rs)`
Jump	`j L`
Jump register	`jr rs`
Branch less than 0	`bltz rs,L`
Branch equal	`beq rs,rt,L`
Branch not equal	`bne rs,rt,L`

Pseudoinstruction	Usage
Move	`move regd,regs`
Load address	`la regd,address`
Load immediate	`li regd,anyimm`
Absolute value	`abs regd,regs`
Negate	`neg regd,regs`
Multiply (into register)	`mul regd,reg1,reg2`
Divide (into register)	`div regd,reg1,reg2`
Remainder	`rem regd,reg1,reg2`
Set greater than	`sgt regd,reg1,reg2`
Set less or equal	`sle regd,reg1,reg2`
Set greater or equal	`sge regd,reg1,reg2`
Rotate left	`rol regd,reg1,reg2`
Rotate right	`ror regd,reg1,reg2`
NOT	`not reg`
Load doubleword	`ld regd,address`
Store doubleword	`sd regd,address`
Branch less than	`blt reg1,reg2,L`
Branch greater than	`bgt reg1,reg2,L`
Branch less or equal	`ble reg1,reg2,L`
Branch greater or equal	`bge reg1,reg2,L`

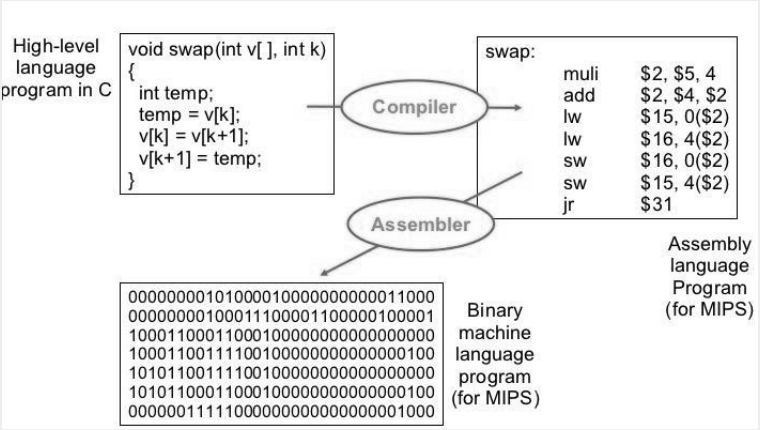
» Formato de instrucciones en MIPS

3 Formatos (fácil decodificación en hardware)



» Formato de instrucciones en MIPS

Un ejemplo completo de traducción al lenguaje de la máquina



» Tipos de Instrucciones

Tipos Principales de Instrucciones

- Aritméticas y Lógicas
  - Enteros
  - Punto Flotante
- Instrucciones de transferencia de datos (memoria)
  - Carga y Almacenamiento
- Flujo de control
  - Salto
  - Bifurcación condicional
  - Llamado y Retorno

» Tipos de Instrucciones

Instrucciones aritméticas en MIPS

- Las instrucciones aritméticas y lógicas más comunes tienen 3 operandos
- El orden de los operandos es fijo (el destino primero)
- Ejemplo:

```
Código C:        a = b + c;

Código MIPS:     add $s0, $s1, $s2
```

(el compilador asocia `$s0`, `$s1` y `$s2` a las variables a, b y c)

» Tipos de Instrucciones

Instrucciones aritméticas en MIPS

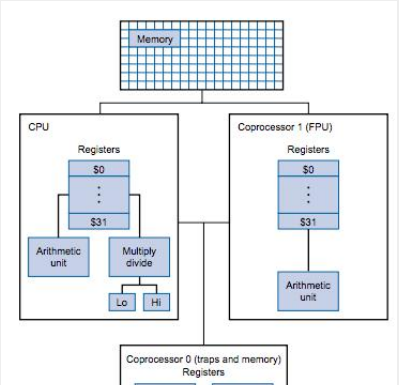
```
Código C:        a = b + c + d;
                 e = f - a;

Código MIPS:     add $t0, $s1, $s2
                 add $s0, $t0, $s3
                 sub $s4, $s5, $s0
```

* Los operandos deben estar en registros, y existen sólo 32
* Principio de diseño: Más pequeño es más rápido. Porqué? (viaje de las señales)

» Tipos de instrucciones en MIPS

Registros vs. Memoria

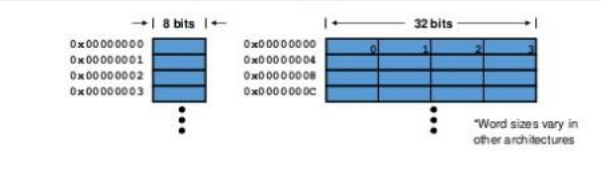
- Los operandos deben estar en registros, y existen sólo 32
- El compilador asocia variables a registros
- ¿Qué sucede con un programa con un montón de variables?



» Representación de datos a nivel máquina

Organización de la Memoria principal en MIPS

- 2^{32} bytes, con direcciones 0, 1, 2 ...
- 2^{30} 4-bytes palabras, con direcciones 0, 4, 8 ...
- Las direcciones de las palabras deben ser múltiplos de 4



» Tipos de Instrucciones

Registros vs. Memoria

- Los operandos deben estar en registros, y existen sólo 32
- El compilador asocia variables a registros
- ¿Qué sucede con un programa con un montón de variables?

» Tipos de Instrucciones

Asignación de Registros

- El compilador intenta asociar tantas variables a registros como sea posible
- Algunas variables no pueden ser alocadas
  - grandes arreglos (vectores)
  - variables accedidas con diferentes punteros
  - variables alocadas dinamicamente
    - heap
    - stack
- El compilador podría quedarse sin registros : **spilling**

» Tipos de Instrucciones

Instrucciones de transferencia de datos

* Instrucciones de **Carga** y **Almacenamiento**
* Ejemplo:

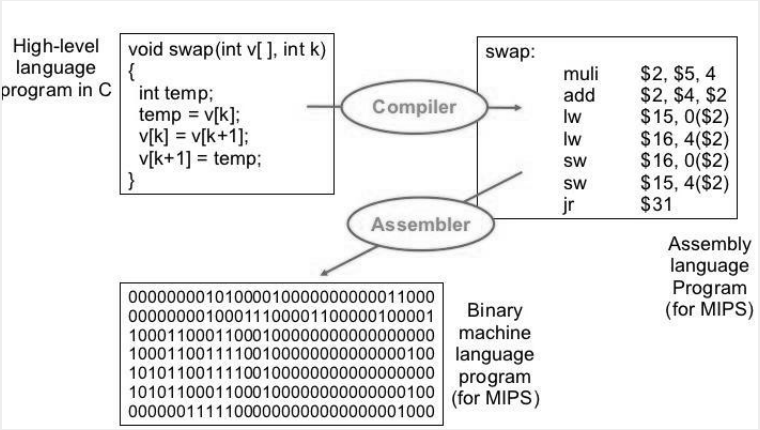
```
Código C:        a[8] = h + a[8];

Código MIPS:     lw  $t0, 32($s3)
                 add $t0, $s2, $t0
                 sw  $t0, 32($s3)
```

* Las operaciones de carga y almacenamiento no tienen operandos destino (registro)
* Recuerde: las operaciones aritméticas y lógicas operan sobre registros, no sobre elementos en la memoria!

» Formato de instrucciones en MIPS

Un ejemplo completo de traducción de C al lenguaje de la máquina

* ¿Puede interpretar la asignación de variables y el por qué del código en lenguaje ensamblador?



» Tipos de Instrucciones

Repaso

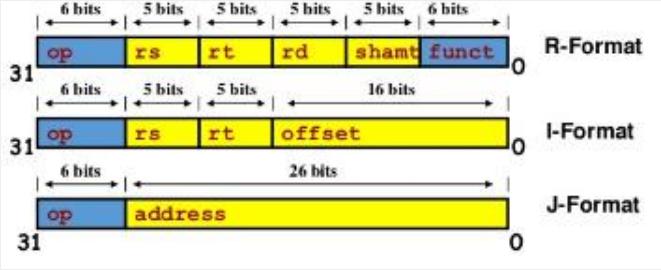
- MIPS
  - Cargamos palabras (words) pero utilizamos direccionamiento del primer byte
  - Las instrucciones aritméticas y lógicas operan unicamente con registros

```
Instrucción             Significado

add $s1, $s2, $s3       $s1 = $s2 + $s3
sub $s1, $s2, $s3       $s1 = $s2 - $s3
lw $s1, 100($s2)        $s1 = MEMORIA[$s2+100]
sw $s1, 100($s2)        MEMORIA[$s2+100] = $s1
```

» Lenguaje Máquina

Formato de Instrucciones en MIPS

- Las instruccciones, como los registros y la palabra, son de 32-bits
  - Los registros están numerados en el código máquina: del 0 al 31



» Tipos de Instrucciones

Instrucciones de transferencia de Control

- Instrucciones para realizar decisiones
  - Alterar el flujo de control de ejecución
  - y por lo tanto, cambiar la "próxima" instrucción a ser ejecutada
- Instrucciones de bifurcación condicional en MIPS

```
bne $t0, $t1, label
beq $t0, $t1, label
```

- Ejemplo:

```
Código en C:      if (i == j)  h = i + j;

Código en MIPS:         bne $s0, $s1, label
                        add $s3, $s0, $s1
                  label: ...
```

» Tipos de Instrucciones

Instrucciones de transferencia de Control

- Instrucciones de salto incondicional
  - j label
- Ejemplo:

Código en C:

```
if (i != j)
      h = i + j;
else
      h = i - j;
```

Código en MIPS:

```
      beq $s4, $s5, lab1
      add $s3, $s4, $s5
      j lab2
lab1: sub $s3, $s4, $s5
lab2:
```

» Tipos de Instrucciones

Repaso

Instrucción	Significado
`add $s1, $s2, $s3`	`$s1 = $s2 + $s3`
`sub $s1, $s2, $s3`	`$s1 = $s2 - $s3`
`lw $s1, 100($s2)`	`$s1 = MEMORIA[$s2+100]`
`sw $s1, 100($s2)`	`MEMORIA[$s2+100] = $s1`
`bne $s4, $s5, Etiq`	Próx. instr. está en Etiq si $s4 es distinto a $s5
`beq $s4, $s5, Etiq`	Próx. instr. está en Etiq si $s4 es igual a $s5
`j Etiqueta`	Próx. instr. está en Etiqueta

» Tipos de Instrucciones

Instrucciones de transferencia de Control

- Salto condicional: bne, beq
- ¿Qué sucede con saltar si es menor qué?
- Nueva instrucción:

```
slt $t0, $s1, $s2

Significado:      if $s1 < $s2 then
                      $t0 = 1
                  else
                      $t0 = 0
```

- Puede ser utilizada para construir

  `"blt $s1, $s2, Etiqueta"`

  - Se pueden construir estructuras de control de ejecución generales
- El ensamblador necesita utilizar un registro *temporal*
  - Convención de uso de registros

» Modelo de programación MIPS

Convención de uso de registros en MIPS

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)

» Modelo de programación MIPS

Constantes

- Pequeñas constantes son utilizadas muy frecuentemente (50 por ciento de los operandos son constantes)

```
Ejemplo:   a = a + 5;
           b = b + 1 ;
           c = c * 2 + 1;
```

- ¿Soluciones?
  - Poner las constantes típicas (1, 2, 10) en memoria y cargarlas
  - Crear registros cableados en hardware con un valor (hard-wired). Como el registro zero
  - o .....
- Colocar las constantes como parte de las Instrucciones (MIPS)

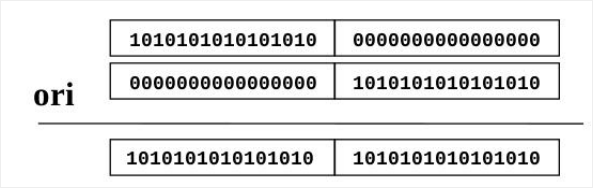
```
Ejemplo:   addi $29, $29 + 5
           addi $8, $8, 1
           andi $29, $29, 6
           ori $9, $9, 4
```

» Modelo de programación MIPS

¿Qué sucede con grandes Constantes?

- Las instrucciones son de 32-bits, por lo que no es posible colocar una constantes de 32-bits dentro de la instrucción.
- Solución: para cargar una constante de 32-bit en un registro se utilizan dos instrucciones

```
Ejemplo:   lui $t0, 0xAAAA
           ori $t0, 0xAAAA
```



» Programación de la máquina: lenguaje ensamblador MIPS

Conjunto básico de instrucciones MIPS

Instruction	Usage
Load upper immediate	`lui rt,imm`
Add	`add rd,rs,rt`
Subtract	`sub rd,rs,rt`
Set less than	`slt rd,rs,rt`
Add immediate	`addi rt,rs,imm`
Set less than immediate	`slti rd,rs,imm`
AND	`and rd,rs,rt`
OR	`or rd,rs,rt`
XOR	`xor rd,rs,rt`
NOR	`nor rd,rs,rt`
AND immediate	`andi rt,rs,imm`
OR immediate	`ori rt,rs,imm`
XOR immediate	`xori rt,rs,imm`
Load word	`lw rt,imm(rs)`
Store word	`sw rt,imm(rs)`
Jump	`j L`
Jump register	`jr rs`
Branch less than 0	`bltz rs,L`
Branch equal	`beq rs,rt,L`
Branch not equal	`bne rs,rt,L`

Pseudoinstruction	Usage
Move	`move regd,regs`
Load address	`la regd,address`
Load immediate	`li regd,anyimm`
Absolute value	`abs regd,regs`
Negate	`neg regd,regs`
Multiply (into register)	`mul regd,reg1,reg2`
Divide (into register)	`div regd,reg1,reg2`
Remainder	`rem regd,reg1,reg2`
Set greater than	`sgt regd,reg1,reg2`
Set less or equal	`sle regd,reg1,reg2`
Set greater or equal	`sge regd,reg1,reg2`
Rotate left	`rol regd,reg1,reg2`
Rotate right	`ror regd,reg1,reg2`
NOT	`not reg`
Load doubleword	`ld regd,address`
Store doubleword	`sd regd,address`
Branch less than	`blt reg1,reg2,L`
Branch greater than	`bgt reg1,reg2,L`
Branch less or equal	`ble reg1,reg2,L`
Branch greater or equal	`bge reg1,reg2,L`

» Modelo de programación MIPS

Resumen de la arquitectura MIPS

- Instrucciones sencillas de 32-bits de ancho
- Muy estructurado, no hay necesidad de grandes variaciones
- Confía en el compilador para ganar performance
- Hay que ayudar al compilador cuando sea necesario
- Únicamente 3 formatos de instrucciones

- ¿Cómo programar mejor? (cantidad de variables, constantes)

Arquitectura MIPS

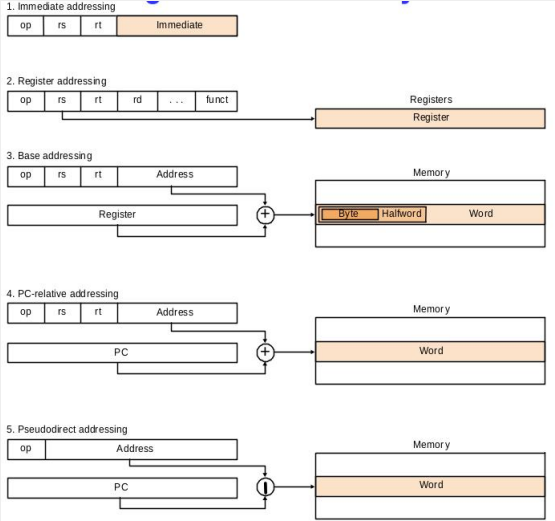
- Máquina MIPS general
- Hardware visible al programador: memoria y registros
- Cuando utilizar lenguaje ensamblador
- Conjunto de instrucciones
- **Modos de direccionamiento**
- lenguaje ensamblador vs lenguaje máquina

» Modos de direccionamiento

- Los modos de direccionamiento son las diferentes formas de especificar un operando dentro de una instrucción (a nivel código máquina).
- Es el CÓMO se especifican e interpretan las direcciones de memoria según las instrucciones
- NO CONFUNDIR modos de direccionamiento con los formatos de direccionamiento permitidos al programar en lenguaje ensamblador
- Absoluto, Inmediato, Directo por registro suelen ser los mas importantes

» Modos de direccionamiento

Modos de direccionamiento en MIPS



Arquitectura MIPS

- Máquina MIPS general
- Hardware visible al programador: memoria y registros
- Cuando utilizar lenguaje ensamblador
- Conjunto de instrucciones
- Modos de direccionamiento
- lenguaje ensamblador vs lenguaje máquina

» Modelo de programación MIPS

Lenguaje Ensamblador vs. Lenguaje Máquina?

- El lenguaje ensamblador provee una representación simbólica conveniente
  - Es mucho más sencillo que programar escribiendo números
  - Por ejemplo: el destino de una operación va primero
- El lenguaje máquina es la verdadera realidad
  - Por ejemplo: el destino ya no es lo primero que aparece en una operación va primero
- El lenguaje ensamblador puede proveer **pseudoinstrucciones**
  - Por ejemplo:
    ```
    move $t0, $t1
    ```
  - sería implementadao utilizando:
    ```
    add $t0, $t1, $zero
    ```
- Cuando se debe considerar la performance se deben contar las instrucciones reales

» Bibliografía

Libros

* Andrew S. Tanenbaum (2000), ORGANIZACIÓN DE COMPUTADORAS un enfoque estructurado, Editorial Prentice Hall. (10 copias en biblioteca)
* David. Patterson John L. Hennessy (1995), ORGANIZACIÓN Y DISEÑO DE COMPUTADORES La interfaz hardware/software, McGraw-Hill (8 copias en biblioteca).

Contenido electrónico

* **x86 assembly basis** Una introducción al lenguaje ensamblador x86. Disponible en PEDCO en formato PDF. `https://www.nayuki.io/page/a-fundamental-introduction-to-x86-assembly-programming`
* Apuntes elaborados por la cátedra, disponibles en PEDCO para impresión (pdf) o lectura online (html)
* Secciones de libros aptas para publicacion