Clase empieza a las 12:10

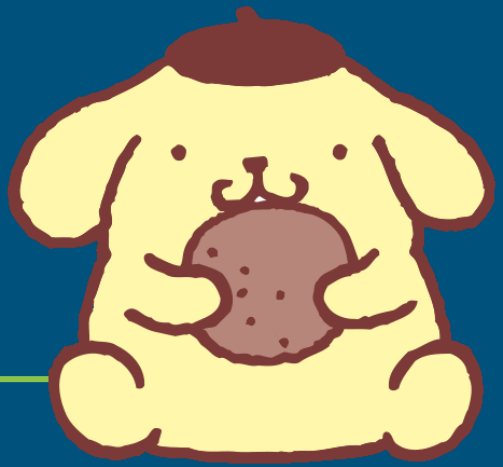
Introducción a Ensamblador

Organización y Arquitectura de Computadoras (2023-2)

Ricardo Enrique Pérez Villanueva

Esta presentación es un complemento a la información dicha en la práctica.

Checa también la práctica y ya de paso una Googleada si no entiendes algo.



Sobre MIPS

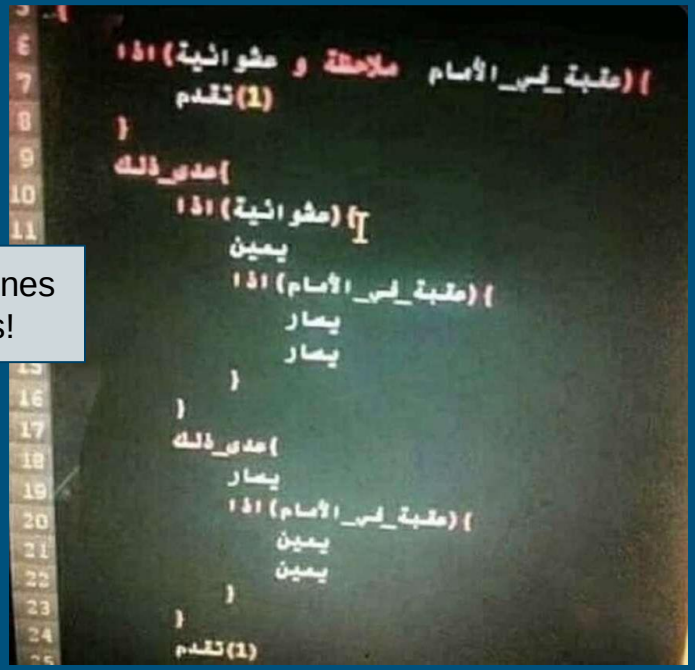
MIPS (Microprocessor without Interlocked Pipeline Stages) es una arquitectura diseñada para sistemas incrustados, sistemas operativos y otras aplicaciones de bajo nivel. MIPS usa un sistema de instrucciones simples que es fácil de entender y optimizar. En esta clase vamos a cubrir los puntos básicos de la programación de MIPS, incluyendo el conjunto de instrucciones, tipos de datos, control de flujo y llamadas al sistema.

Ventajas de programar en Ensamblador

- La optimización y velocidad del código es inigualable.
- Nos ayuda a tener control total sobre los recursos de memoria y sobre las llamadas que se realizan en el sistema.
- Las llamadas contactan directamente con el hardware del equipo, ya que estamos un nivel arriba de lenguaje máquina, con la facilidad de poder leer las instrucciones y entenderlas.

Conjunto de instrucciones de MIPS

Consiste de un número relativamente pequeño de instrucciones simples, cada una de ellas realiza una operación específica con los datos guardados en los registros de memoria. Las instrucciones de MIPS están codificadas en números binarios y cada instrucción están diseñadas para ser ejecutadas en un ciclo de reloj.



Tipos de datos

MIPS soporta diferentes tipos de datos, por ejemplo.

- word: Variable de 32 bits
- doubleword: Variable de 64 bits
- float: Numero de punto flotante
- asciiz: Cadena en formato ASCII

Los datos se pueden guardar en registro o memoria y MIPS nos da instrucciones para mover datos entre registros y memoria, además de realizar operaciones aritméticas y lógicas con los dichos datos.

Suma y Resta

add: Añade 2 valores y guarda el resultado en el registro especificado.

```
add $s1, $s0, $zero
# Añade $s0 y $zero para obtener el valor de $s1
```

sub: sustrae un valor del otro y guarda el resultado en el registro especificado.

```
sub $s1, $s0, $zero
# Resta $s0 y $zero para obtener el valor de $s1
```

Guardar “Palabras”

lw: Carga una palabra de 4 Bytes de memoria a un registro.

```
lw $t1, count
#Guarda la palabra count en $t1
```

sw: Guarda una palabra de registro en la memoria.

```
sw $t1, count
#Guarda el contenido de $t1 en count
```

Básicos del control de flujo

beq: Salta a una etiqueta si 2 registros son iguales. Si no son iguales, continua la ejecución.

bne: Salta a una etiqueta si 2 registros no son iguales. Si son iguales, continua la ejecución.

j: Salta a una etiqueta.

```
loop:
     bne $t1, 10, loop
     #Salta a loop si $t1 y 10 no son iguales.
```

Instrucciones de Control de Flujo

MIPS nos proporciona de varias instrucciones para controlar el flujo del programa, estas incluyen:

- beq: Salta a una etiqueta si 2 registros son iguales. Si no son iguales, continua la ejecución.
- bne: Salta a una etiqueta si 2 registros no son iguales. Si son iguales, continua la ejecución.
- j: Salta a una etiqueta.
- jal: Salta a una etiqueta y guarda la dirección de retorno en el registro especificado
- jr: Salta a la dirección del registro.

Estás operaciones son usadas para implementar loops, condicionales y llamadas a funciones en ensamblador.

Tipos de operaciones en MIPS

Existen 3 tipos de operaciones en MIPS:

- R-Type
- J-Type
- I-Type

En algunas fuentes, las instrucciones del coprocesador se cuentan como un cuarto tipo de operaciones, además, existen otros 2 tipos de operaciones todas tristes porque sólo afectan a los números flotantes, siendo estas FR-Type y FI-Type.

Operaciones de tipo R

Las instrucciones de tipo R se usan cuando todos los valores usados por la instrucción se encuentran en los registros. Las operaciones de tipo R tienen el siguiente formato: `OP rd, rs, rt` donde rs y rt son los registros de dónde se sacará la información, y rd es el registro donde se guardará la información.

Un ejemplo de esto sería la operación add.

op	rs	rt	rd	shamt	funct	
000000	10001	10010	10000	00000	100000	Example: add $s0, $s1, $s2

Operaciones de tipo I

Las instrucciones de tipo I se usan cuando se debe operar con un valor inmediato y un valor en el registro. Los valores inmediatos tienen un máximo de 16 bits. Números más grandes no pueden ser manipulados con operaciones inmediatas. Las instrucciones inmediatas tienen el siguiente formato:

```
OP rt, IMM(rs)
```

Donde rs es el inmediato. Excepto en las operacione bne y beq, donde hay que realizar una comparación entre un registro y un inmediato. Un ejemplo de instrucciones I-Type es addi.

op	rs	rt	immediate	
001000	10011	01010	0000000000000100	Example: addi $t2, $s3, 4

Operaciones de tipo J

Las instrucciones de tipo J son usadas para realizar saltos. Estas operaciones tienen mayor capacidad para manejar valores inmediatos, ya que las direcciones de memoria usan números grandes. Las operaciones de tipo J tienen el siguiente formato: `OP LABEL`

Donde OP es la operación a realizar y LABEL a donde se debe saltar. Un ejemplo de estas operaciones es la instrucción J.

op	address	
000010	00000000000000000100000001	Example: `j LOOP` (or `j 1028`)

Llamadas a Sistema

MIPS puede interactuar directamente con el sistema operativo mediante llamadas al sistema; estas son funciones especiales proporcionadas por el sistema operativo que permite al programa tomar input/outputs, locación de memoria y creación de procesos. Las llamadas a sistema más comunes son:

- syscall: Dado el valor del registro $v0, realiza la llamada al sistema especificada en dicho registro.
- li: Carga un valor inmediato a un registro
- la: Carga la dirección de una etiqueta a un registro.

Las llamadas al sistema son una parte esencial de escribir programas en MIPS
(Y extremadamente esenciales para usar MARS)

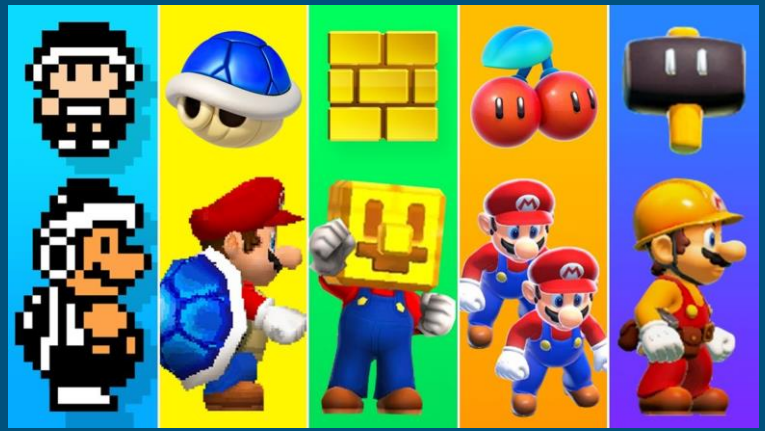
```
# Cuando $v0 es 10, la llamada al sistema
# que se realiza es salir del programa.
li $v0, 10
syscall
```

syscall y $v0

La operación syscall puede tener varios comportamientos, dependiendo del valor que se encuentre en la dirección de memoria $v0.

Imagina que syscall es el botón B en los juegos de Mario. Por sí solo, no hace nada, pero si tenemos un Power Up guardado, hace algo diferente dependiendo de lo que tenemos guardado.

Si queremos cambiar la función de syscall debemos sobreescribir el valor en $v0 y después volver a usar syscall.



Consejos para usar MIPS

- Mantenga su código simple y legible. Concéntrense en escribir código que sea fácil de leer y depurar, ya que las operaciones no nos dicen mucho además por si solas.
- Usa comentarios para explicar tu código. Los comentarios son esenciales para documentar su código y explicar cómo funciona. Asegúrese de incluir comentarios que expliquen el propósito de cada instrucción y cómo encaja en el programa general.
- Pruebe su código a fondo. Los programas MIPS pueden ser difíciles de depurar, por lo que es importante probar el código minuciosamente para asegurarse de que funciona como se espera. Utilice casos de prueba y herramientas de depuración para identificar y corregir errores en su código.
- Utilice las mejores prácticas para la optimización. MIPS está diseñado para optimizar el rendimiento, por lo que es importante utilizar las mejores prácticas para optimizar su código. Esto incluye usar las instrucciones más eficientes, minimizar el acceso a la memoria

Fuentes y más material chido 

PDF muy completo con muchas instrucciones de MIPS explicadas:
https://www.dsi.unive.it/~gasparetto/materials/MIPS_Instruction_Set.pdf

Hoja de referencias de MIPS:
https://inst.eecs.berkeley.edu/~cs61c/resources/MIPS_help.html

Wiki de MIPS:
https://en.wikibooks.org/wiki/MIPS_Assembly/Instruction_Formats