Proyecto Final

Realizado por:
Carlos Eduardo López Camey (#08107)
Héctor Hurtarte (#08119)
Mario Sánchez (#06089)

Catedrático Martha Ligia Naranjo

Universidad del valle de Guatemala
Organización de las Computadoras
CC3001

Mayo 2009

I. Introducción

MIPS, acrónimo de **Microprocessor without Interlocked Pipeline Stages**, es una arquitectura de procesadores tipo RISC desarrollada por MIPS Computer Systems Inc. Los diseños de MIPS se usan en las estaciones de trabajo de SGI, y tienen mucha implantación en sistemas empotrados, dispositivos que soportan Windows CE, y en los routers de Cisco. La consola Nintendo 64, la Sony PlayStation, la Sony PlayStation 2, y la consola portátil Sony PSP usan procesadores MIPS.

Estos procesadores poseen un set de instrucciones de lenguaje ensamblador, con lo cual logran realizar esta arquitectura de procesadores. Entre las MIPS conocidas se encuentran versiones como MIPS I, MIPS II, MIPS III, MIPS IV, y MIPS 32/64.

Este set de instrucciones definidas a continuación, se encuentran en la versión de MIPS I. a pesar de que todas las versiones son compatibles con la versión anterior a ellas, se eligió la primera versión, debido a que es interesante se modelar la primera versión que hubo de MIPS.

Se realizo un sistema de simulación de una máquina virtual, **en lenguaje PHP,** que recibe como entrada un programa escrito en el lenguaje ensamblador asignado, y despliega la ejecución del programa traducido paso a paso o en forma continua.

La pantalla de salida contiene: registros visibles a nivel ISA, códigos de control, memoria, instrucción que se va ejecutando y otros botones de utilidad para el usuario.

II. Conjunto de instrucciones del lenguaje ensamblador

Esta es una descripción del conjunto de instrucciones MIPS, su significado, la sintaxis, la semántica, y codificaciones. La sintaxis de cada instrucción se refiere a la sintaxis del lenguaje de montaje apoyado por el ensamblador MIPS.

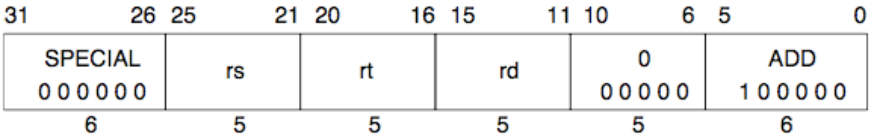
MIPS Instrucciones Assembler

Categoría	Instrucción	Ejemplo	Significado	Comentario
Aritmética	add	add $1 $2 $3	$1 = $2 + $3	3 operandos;
	add immediate	addi $1 $2 %100	$1 = $2 + 100	+ constante;
Lógica	and	and $1 $2 $3	$1 = $2 & $3	3 registros operando Lógica AND
	and immediate	and $1 $2 %100	$1 = $2 & 100	Lógica AND registro, constante
Transferencia de Datos	load word	lw $1 %100	$1 = Memory[$2+100]	Datos de memoria hacia registro
	store word	sw $1 %100	Memory[$2+100] = $1	Datos de memoria hacia registro
Branch Condicional	branch on equal	beq $1 $2 %100	if ($1 == $2) go to PC+4+100	Evaluacion igual; PC relative branch
	branch on not equal	bne $1 $2 %100	if ($1 != $2) go to PC+4+100	Evaluacion no igual; PC relative
Salto Incondicional	jump	j %10000	goto 10000	Salta hacia dirección de destino
	jump register	j $31	goto $31	
	jump and link	jal %100	$31 = PC + 4;ir a 100	Almacena dir retorno en $ra
	Jump and link register	jalr	$31 = PC + 4;ir a $sd	

Aritmética

ADD

(MIPS I)

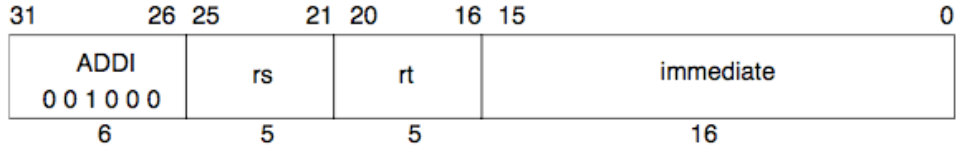


Formato: ADD rd, rs, rt
Descripción: rd = rs + rt

Suma el primer operando (Destino), y el segundo operando (Operando Fuente) y almacena el resultado en el operando de destino. Éste operando puede ser únicamente un registro;.Puede recibir 3 registros o 2 dos registros y 1 numero .

ADDI

(MIPS I)



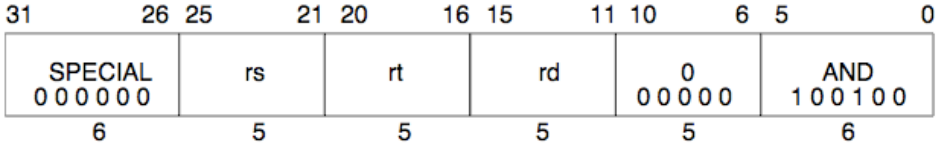
Formato: ADDI rt, rs, immediate
Descripcion: rt = rs + inmediato

El operando fuente puede ser inmediato (ADDI). Cuando un valor inmediato es usado como operando, se le aplica una extensión de signo para que tenga la longitud del formato del operando de destino. Puede recibir 3 registros o 2 dos registros y 1 numero y puede ocurrir una suma de números positivos y negativos.

Lógica

AND

(MIPS I)

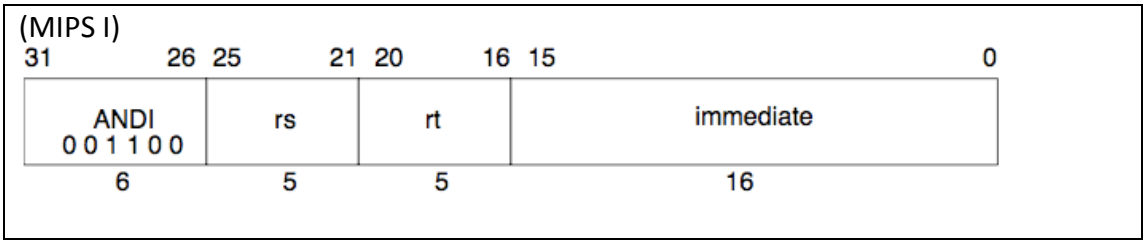


Formato: AND rd, rs, rt

Descripcion: rd rs AND rt

El contenido de GPRrs se combina con el contenido de GPRrt, en un una operación lógica AND bitwise. El resultado se coloca en GPRrd. . Puede recibir 3 registros o 2 dos registros y 1 numero.

ANDI

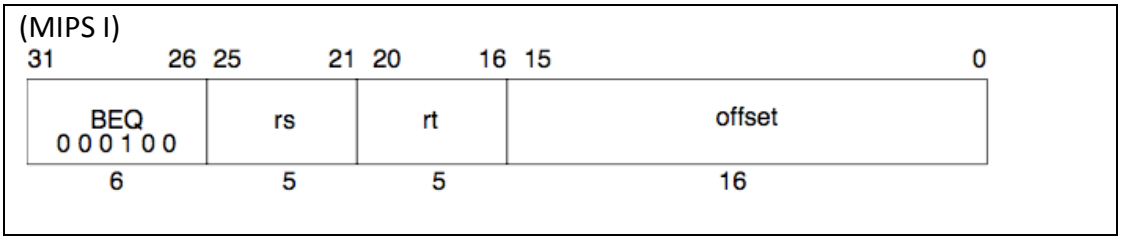


Formato: ANDI rt, rs, immediate
Descripcion: rt = rs AND immediate

Los 16-bit inmediatos son zero-extendido a la derecha y combinados con el contenido de rs en un AND. El resultado se guarda en rt.

Branch Condicional

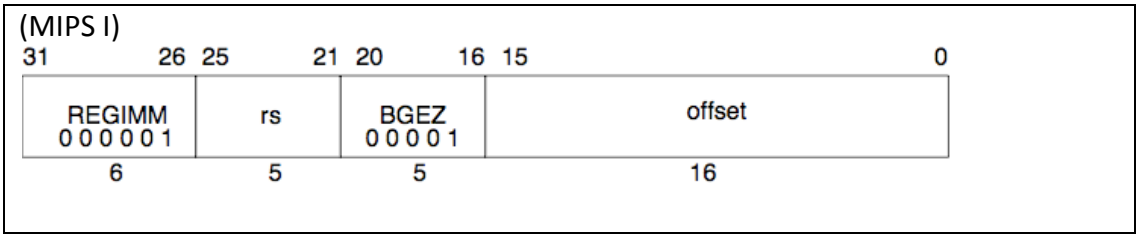
BEQ



Formato: BEQ rs, rt, offset
Descripcion: if rs == rt then branch

Un offset de 18/bit (el campo del offset de 16/bit se movió a la izquierda 2 bits) se agrega a la dirección de instrucción para formar una dirección meta efectiva para formar un dirección relativa al PC. Si los contenidos del rs y rt son iguales, se unen a la dirección efectiva meta, después de las instrucciones en el espacio del retraso se ejecuta.

BGEZ

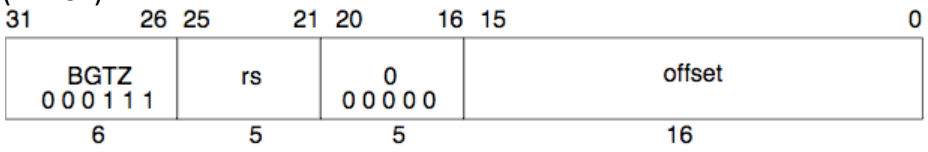


Formato: BGEZ rs, offset
Descripcion: if rs==0 then branch

Un offset con signo de 18-bit (el campo offset de 16-bit cambio a la izquierda 2 bits) se agrega a la dirección de la instrucción para formar una dirección neta relativa al PC efectiva. Si los contenidos del rs son mas grandes o iguales a cero (símbolo bit es 0).

BGTZ

(MIPS I)

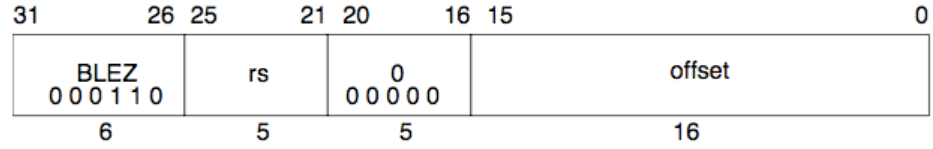


Formato: BGTZ rs, offset
Descripcion: if rs > 0 then branch

Un offset con signo de 18-bit (el campo offset de 16-bit cambio a la izquierda 2 bits) se agrega a la dirección de la instrucción para formar una dirección neta de PC-relative efectiva. Si los contenidos del GPR rs son mas grandes a cero (símbolo bit es 0, pero el valor no es 0).

BLEZ

(MIPS I)

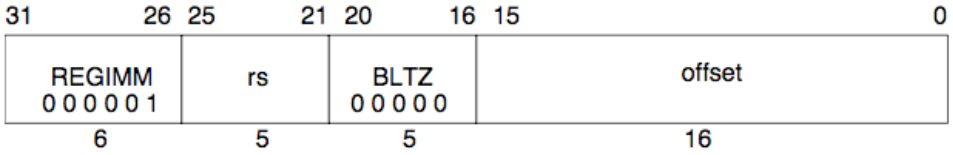


Formato: BLEZ rs, offset
Descripcion: if rs <=0 then branch

Un offset con signo de 18-bit (el campo offset de 16-bit cambio a la izquierda 2 bits) se agrega a la dirección de la instrucción para formar una dirección neta de PC-relative efectiva. Si los contenidos del GPR rs son mas grandes a cero (símbolo bit es 0, pero el valor no es 0).

BLTZ

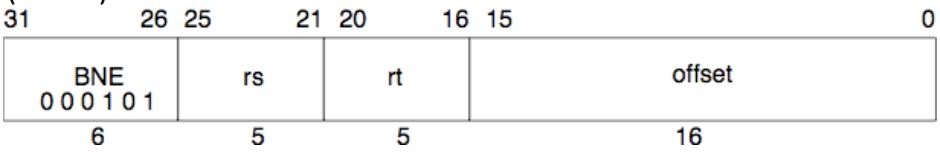
(MIPS I)



Formato: BLTZ rs, offset
Descripcion: if rs < 0 then branch

BNE

(MIPS I)



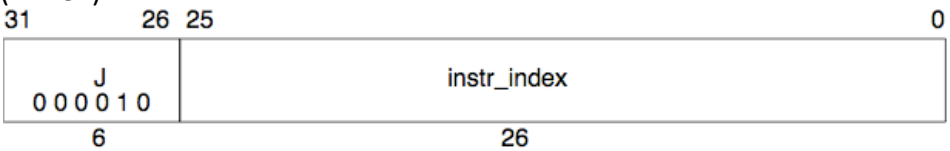
Formato: BNE rs, rt, offset
Descripcion: if rs <>rt then branch

Un offset con signo de 18-bit (el campo offset de 16-bit cambio a la izquierda 2 bits) se agrega a la dirección de la instrucción para formar una dirección neta de PC-relative efectiva. Si los contenidos del GPR rs son mas grandes a cero (símbolo bit es 0, pero el valor no es 0).

Salto Uncondicional

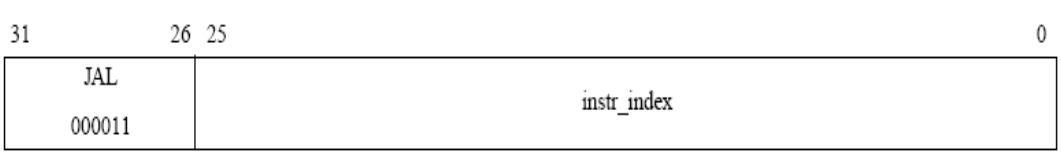
J

(MIPS I)



JAL

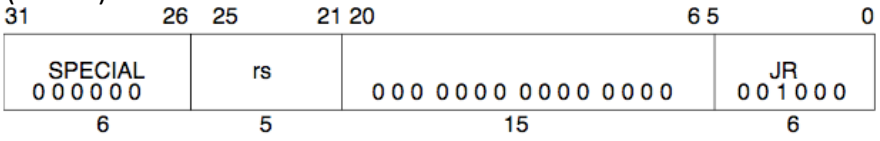
(MIPS I)



Coloca el link de la dirección de retorno en $ra. El link de retorno es la dirección de la segunda instrucción. Los 28 bits bajos de la dirección meta es el campo *instr_index* que cambio 2 bits a la izquierda. Los restantes bits arriba son los bits correspondientes a la dirección de la instrucción en el delay slot (no la rama en si misma). Salte a la dirección meta efectiva. Ejecute la instrucción que sigue al slot, en el delay slot de la rama, antes de ejecutar el salto en si mismo.

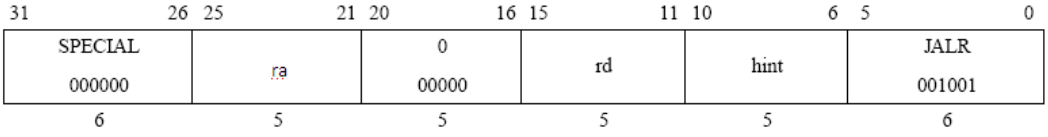
JR

(MIPS I)



JALR

(MIPS I)

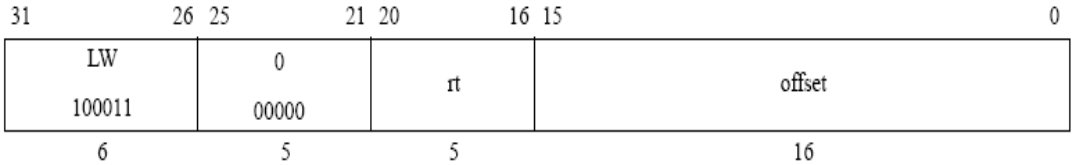


Formato: JALR rd
Descripcion: ra == return_addr, PC ==rd

Transferencia de Datos

LW

(MIPS I)

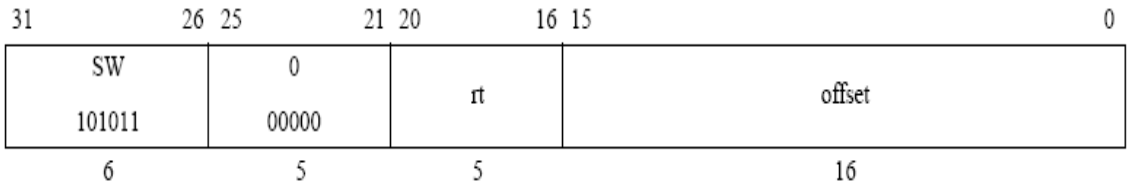


Formato: LW rt, offset(base)
Descripcion: rt ==memory[PC+offset]

Los contenidos de la palabra de 32.bit localizada en la memoria especificada por la dirección efectiva alineada están atraídos sign-extended a la extensión del registro GPR, y localizados en GPR rt. El signed offset de 16-bit se agrega a los contenidos de la base GPR para formar la dirección efectiva.

SW

(MIPS I)

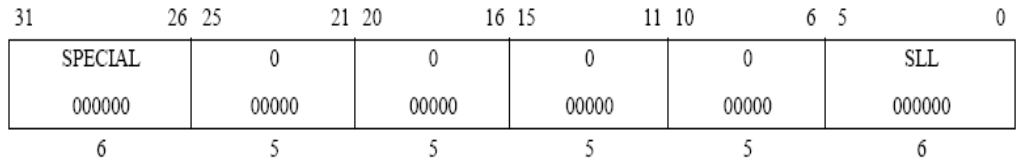


Formato: SW rt, offset
Descripcion: memory[PC+offset] rt

La palabra menos importante del registro rt de 32/bit se almacena en la memoria en un lugar especifico con la dirección alineada efectiva. El signed offset de 16-bit se agrega al contendio de la base GPR para formar la dirección efectiva.

Otros
NOP

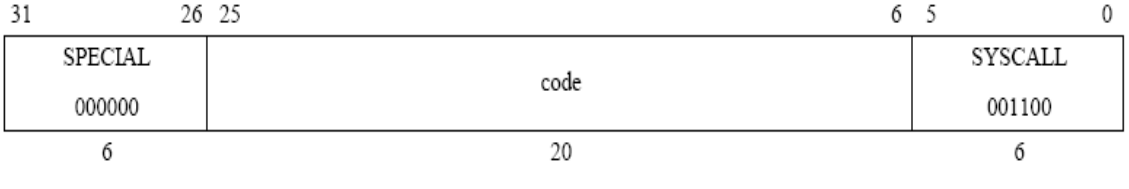
(MIPS I)



NOP es el assembler para denotar que no existe operación en la línea en procesamiento.

SYSCALL

(MIPS I)



Una llamada a Syscall se produce, de inmediato y sin condiciones. La elección del proceso a seguir por la llamada depende del contenido del registro $v1.

III. Análisis general del problema

Tipos de estructuras utilizados para realizar la simulación de memoria y registros, análisis y diseño de ISA

Se creó una clase general, esta clase se llama Filburt, esta es la principal, y de la que depende MIPS. Filburt posee un método llamado get_html(), este método es el encargado de desplegar toda la interfaz grafica, se encarga de mostrar "home.php", y "header.php". Por último Filburt posee varios objetos, entre estos: uno para manejar el lenguaje (de la clase "lang") y otro para traducir un archivo en lenguaje ensamblador (clase "traductor").

La clase "lang" se encarga únicamente del idioma, la instancia de este objeto en Filburt se encarga de poder elegir entre dos idiomas (español, ingles) para mostrarlo al usuario.

La mayoría del MIPS se basa en programación PHP, pero se utilizo otro lenguaje para la interacción con el usuario. Javascript fue esencial para tener una interacción con el usuario, y para lograr esto se utilizaron varias librerías, entre ellas el *framework* "prototype".

Para realizar la simulación de memoria se utilizo un modelo de vector bidimensional en php, después se mostro el simulador usando el contenido de ese vector (usando javascript).
Luego para modelar lo registros se utilizo otro vector, en este cada uno tendría la información de el contenido del registro, nombre de registro, referencia binaria (representación).

El principal problema es el de traducir bien a un lenguaje aun de más bajo nivel para poder interpretarlo.

Uno de los problemas encontrados para el diseño del simulador de la maquina virtual fue la variedad de maquinas *MIPS* existentes (al menos de la I a la IV). De la diversidad de instrucciones, y su consecuente aumento de complejidad, fue necesario escoger aquellas instrucciones que parecieran más adecuadas al proyecto, por lo que se dejaron fuera todas las relacionadas al soporte de números *float* como datos del sistema.

Poro otro lado, fue necesario escoger solo algunas de los procesos soportados por la instrucción *syscall* y algunas de las directivas a utilizar, y solo algunas de las directivas. Es necesario mencionar que esta escogencia y utilización de estas instrucciones y directivas se realizo en base al documento *MIPS Assembly Language Programming*, del autor Daniel J. Ellard.

Lenguaje de programación utilizado

El lenguaje utilizado para realizar MIPS, fue PHP con ayuda de Java Script. Se eligió este lenguaje debido a que era desafiante, así mismo que se quería aprender más acerca de este lenguaje.

IV. Descripción general de variables y módulos del programa

Etiquetas

Las etiquetas en la MIPS a implementar poseen un requerimiento, que el nombre de la etiqueta este separada por un espacio y que el contenido tiene que está entre comillas "" si es texto, porcentaje % si es decimal y x si es entero.

.DATA *dirección:* indica que lo siguiente son datos, por default inicia en la siguiente localidad de memoria disponible. Si el parámetro opcional *dirección* está presente, entonces los datos comienzan en esa dirección
.TEXT *dirección:* indica que lo siguiente son necesariamente instrucciones. y pasa lo mismo que con .DATA con respecto al parámetro opcional *dirección*.
.ASCII *string*: ensambla el *string* siguiente, un carácter en cada localidad de memoria, no lo termina con un caracter *null*.
.ASCIIZ *string*: ensambla el *string* siguiente, un carácter en cada localidad de memoria, si lo termina con un caracter *null*.
.SPACE *num*: reserva las siguientes *num* localidades de memoria.
.WORD *num*: llena la localidad de memoria con el contenido definido por el parámetro que debe de ser un número binario o hexadecimal

Registros utilizados

Registros de propósito general (GPRS) se indican con un signo de dólar ($)

Nombre simbólico	Numero	Uso
$v1	2-3	Registro de resultado
$a0-$a3	4-7	Registros de parámetro 1…..4
$t0-$t9	8-15 , 24-25	Registros temporales 0…..9
$s0-$s7	16-23	Guarda los registros 0….7
$ra	31	Regresa la dirección

La mayor parte de los elementos externos requeridos (listado de caracteres ASCII, registros, instrucciones, directivas) fueron empleados, durante la ejecución, como

vectores bidimensionales. De cualquier forma, para su portabilidad fueron almacenados en archivos de texto.

Bibliografía

- *MIPS R3000 Instruction Set Summary*. Consultado 1 de junio de 2009. Sitio web: http://www.xs4all.nl/~vhouten/mipsel/r3000-isa.html
- *MIPS 32TM Architecture for programmers VolumeII: The MIPS32TM Instruction* set. MIPS Technologies. Consultado 1 de junio de 2009. Sitio web: http://www.cs.tau.ac.il/~afek/MipsInstructionSetReference.pdf
- *MIPS Instruction Reference*. Consultado 30 de mayo de 2009. Sitio web: http://www.mrc.uidaho.edu/mrc/people/jff/digital/MIPSir.html
- Ellard, D. J. *MIPS Assembly Language Programming. CS50 Discussion and Project Book*. Septiembre, 1994 [en web].
- Price, C. *MIPS IV Instruction Set*. Septiembre, 1995 [en web].