

Simulador de una arquitectura tipo RISC= conjunto reducido de instrucciones. No hay manejo de pila, sus instrucciones son reducidas. Maneja ciclos por instrucción, cuando menor sea el CPI, mejor

Tiene instrucciones de 32 bits, registros de 64 bits (32 registros de enteros de propósito general y 32 de punto flotante) y sin flags de estados. Tiene una arquitectura Harvard, o sea tiene una memoria de programa y otra de datos con buses independientes

DIRECTIVA DEL COMPILADOR- DATOS E INSTRUCCIONES

.ASCII/.asciiz almacena una cadena ASCII, .byte almacena un byte, .word16 2 byte, .word32 4 bytes, .word almacena 64 bits, .double para punto flotante

Instrucciones de Transferencia de Datos			
lb	$r_d, \text{Inm}(r_i)$		Copia en r_d un byte (8 bits) desde la dirección ($\text{Inm}+r_i$) (con extensión del signo)
lbu	$r_d, \text{Inm}(r_i)$		Copia en r_d un byte (8 bits) desde la dirección ($\text{Inm}+r_i$) (sin extensión del signo)
sb	$r_s, \text{Inm}(r_i)$		Guarda los 8 bits menos significativos de r_s en la dirección ($\text{Inm}+r_i$)
lh	$r_d, \text{Inm}(r_i)$		Copia en r_d un half-word (16 bits) desde la dir. ($\text{Inm}+r_i$) (con extensión del signo)
lhu	$r_d, \text{Inm}(r_i)$		Copia en r_d un half-word (16 bits) desde la dir. ($\text{Inm}+r_i$) (sin extensión del signo)
sh	$r_s, \text{Inm}(r_i)$		Guarda los 16 bits menos significativos de r_s a partir de la dirección ($\text{Inm}+r_i$)
lw	$r_d, \text{Inm}(r_i)$		Copia en r_d un word (32 bits) desde la dir. ($\text{Inm}+r_i$) (con extensión del signo)
lwu	$r_d, \text{Inm}(r_i)$		Copia en r_d un word (32 bits) desde la dir. ($\text{Inm}+r_i$) (sin extensión del signo)
sw	$r_s, \text{Inm}(r_i)$		Guarda los 32 bits menos significativos de r_s a partir de la dirección ($\text{Inm}+r_i$)
ld	$r_d, \text{Inm}(r_i)$		Copia en r_d un double word (64 bits) desde la dirección ($\text{Inm}+r_i$)
sd	$r_s, \text{Inm}(r_i)$		Guarda r_s a partir de la dirección ($\text{Inm}+r_i$)
l.d	$f_d, \text{Inm}(r_i)$		Copia en f_d un valor en punto flotante (64 bits) desde la dirección ($\text{Inm}+r_i$)
s.d	$f_s, \text{Inm}(r_i)$		Guarda f_s a partir de la dirección ($\text{Inm}+r_i$)
mov.d	f_d, f_s		Copia el valor del registro f_s al registro f_d
mtc1	r_s, f_d		Copia los 64 bits del registro entero r_s al registro f_d de punto flotante
mfc1	r_d, f_s		Copia los 64 bits del registro f_s de punto flotante al registro r_d entero
cvt.d.l	f_d, f_s		Convierte a punto flotante el valor entero copiado al registro f_s , dejándolo en f_d
cvt.l.d	f_d, f_s		Convierte a entero el valor en punto flotante contenido en f_s , dejándolo en f_d
Instrucciones Aritméticas			
dadd	r_d, r_s, r_q		Suma r_s con r_q , dejando el resultado en r_d (valores con signo)
daddi	r_d, r_s, N		Suma r_s con el valor inmediato N, dejando el resultado en r_d (valores con signo)
daddu	r_d, r_s, r_q		Suma r_s con r_q , dejando el resultado en r_d (valores sin signo)
daddui	r_d, r_s, N		Suma r_s con el valor inmediato N, dejando el resultado en r_d (valores con signo)
add.d	f_d, f_s, f_q		Suma f_s con f_q , dejando el resultado en f_d (en punto flotante)
dsub	r_d, r_s, r_q		Resta r_q a r_s , dejando el resultado en r_d (valores con signo)
dsubu	r_d, r_s, r_q		Resta r_q a r_s , dejando el resultado en r_d (valores sin signo)
sub.d	f_d, f_s, f_q		Resta f_q a f_s , dejando el resultado en f_d (en punto flotante)
dmul	r_d, r_s, r_q		Multiplica r_s con r_q , dejando el resultado en r_d (valores con signo)
dmulu	r_d, r_s, r_q		Multiplica r_s con r_q , dejando el resultado en r_d (valores sin signo)
mul.d	f_d, f_s, f_q		Multiplica f_s con f_q , dejando el resultado en f_d (en punto flotante)
ddiv	r_d, r_s, r_q		Divide r_s por r_q , dejando el resultado en r_d (valores con signo)
ddivu	r_d, r_s, r_q		Divide r_s por r_q , dejando el resultado en r_d (valores sin signo)
div.d	f_d, f_s, f_q		Divide f_s por f_q , dejando el resultado en f_d (en punto flotante)
slt	r_d, r_s, r_q		Compara r_s con r_q , dejando $r_d=1$ si r_s es menor que r_q (valores con signo)
slti	r_d, r_s, N		Compara r_s con el valor inmediato N, dejando $r_d=1$ si r_s es menor que N (valores con signo)
c.lt.d	f_d, f_s		Compara f_d con f_s , dejando flag FP=1 si f_d es menor que f_s (en punto flotante)
c.le.d	f_d, f_s		Compara f_d con f_s , dejando flag FP=1 si f_d es menor o igual que f_s (en punto flotante)
c.eq.d	f_d, f_s		Compara f_d con f_s , dejando flag FP=1 si f_d es igual que f_s (en punto flotante)
Instrucciones Lógicas			
and	r_d, r_s, r_q		Realiza un AND entre r_s y r_q (bit a bit), dejando el resultado en r_d
andi	r_d, r_s, N		Realiza un AND entre r_s y el valor inmediato N (bit a bit), dejando el resultado en r_d
or	r_d, r_s, r_q		Realiza un OR entre r_s y r_q (bit a bit), dejando el resultado en r_d
ori	r_d, r_s, N		Realiza un OR entre r_s y el valor inmediato N (bit a bit), dejando el resultado en r_d
xor	r_d, r_s, r_q		Realiza un XOR entre r_s y r_q (bit a bit), dejando el resultado en r_d
xori	r_d, r_s, N		Realiza un XOR entre r_s y el valor inmediato N (bit a bit), dejando el resultado en r_d

Instrucciones de desplazamiento de bits		
dsll	r_d, r_s, N	Desplaza a izquierda N veces los bits del registro r_s , dejando el resultado en r_d
dsllv	r_d, r_s, r_N	Desplaza a izquierda r_N veces los bits del registro r_s , dejando el resultado en r_d
dsrl	r_d, r_s, N	Desplaza a derecha N veces los bits del registro r_s , dejando el resultado en r_d
dsrlv	r_d, r_s, r_N	Desplaza a derecha r_N veces los bits del registro r_s , dejando el resultado en r_d
dsra	r_d, r_s, N	Igual que dsrl pero mantiene el signo del valor desplazado
dsrav	r_d, r_s, r_N	Igual que dsrlv pero mantiene el signo del valor desplazado
Instrucciones de Transferencia de Control		
j	$offN$	Salta a la dirección rotulada $offN$
jal	$offN$	Salta a la dirección rotulada $offN$ y copia en r_{31} la dirección de retorno
jr	r_d	Salta a la dirección contenida en el registro r_d
beq	$r_d, r_s, offN$	Si r_d es igual a r_s , salta a la dirección rotulada $offN$
bne	$r_d, r_s, offN$	Si r_d no es igual a r_s , salta a la dirección rotulada $offN$
beqz	$r_d, offN$	Si r_d es igual a 0, salta a la dirección rotulada $offN$
bnez	$r_d, offN$	Si r_d no es igual a 0, salta a la dirección rotulada $offN$
bclif	$offN$	Salta a la dirección rotulada $offN$ si flag FP=0 (ó false) (en punto flotante)
bclt	$offN$	Salta a la dirección rotulada $offN$ si flag FP=1 (ó true) (en punto flotante)
Instrucciones de Control		
nop		Operación nula
halt		Detiene el simulador

ETAPAS CAUSE

- IF: recupera la instrucción de la memoria de instrucciones
- ID: decodifica a instrucción y guarda registros temporales
- EX: ejecuta instrucción
- MEM: guarda valor en la memoria de datos
- WB: guarda el resultado en el operando destino

ATASCOS RAW

TECNICA DE HARDWARE, forwarding: uso un registro temporal donde en la etapa MEM guardo el valor para que otras instrucciones accedan al dato sin que haya pasado por WB. El forwarding minimiza los atascos, no los elimina al 100%

Branch taken stall= atasco por rama equivocada

PILA

- Hagamos nuestro propio push y pop
 - Con **daddi**
 - y **ld/sd**
 - Usamos el registro **r29** como si fuese el SP
 - Inicializamos con: **daddi r29, r0, 0x400**
- push r1**

pop r1

→

→

daddi r29, r29, -8
sd r1, 0(r29)

ld r1, 0(r29)
daddi r29, r29, 8

Ejemplo

- Intercambiar el valor de los registros r4 y r6
- Utilizando la pila
 - Ubicada en 0x400
- Antes

```
push r4
push r6
pop r4
pop r6
```

```
daddi r29,r0,0x400; pila
daddi r4,r0,10
daddi r6,r0,20
;push de r4
daddi r29,r29,-8
sd r4, 0(r29)
;push de r6
daddi r29,r29,-8
sd r6,0(r29)
;pop a r4
ld r4, 0(r29)
daddi r29,r29,8
;pop a r6
ld r6,0(r29)
daddi r29,r29,8
```

Ejemplo

- Intercambiar el valor de los registros r4 y r6
- Utilizando la pila
 - Ubicada en 0x400
- Antes

```
push r4
push r6
pop r4
pop r6
```

```
daddi r29,r0,0x400; pila
daddi r4,r0,10
daddi r6,r0,20
;push de r4
daddi r29,r29,-8
sd r4, 0(r29)
;push de r6
daddi r29,r29,-8
sd r6,0(r29)
;pop a r4
ld r4, 0(r29)
daddi r29,r29,8
;pop a r6
ld r6,0(r29)
daddi r29,r29,8
```

Solución con Pila

```
.code
daddi $sp,$sp,0x400
jal subrutina1
halt
```

```
subrutina2: nop
           jr $ra
```

```
subrutina1: daddi $sp,$sp,-8 ;push $ra
           sd $ra, 0($sp)
```

```
           jal subrutina2
           nop
```

```
           ld $ra, 0($sp) ;pop $ra
           daddi $sp,$sp,8
           jr $ra
```

Calcular suma de enteros, mostrar en pant.

```
.data
DIR_CONTROL: .word 0x10000
DIR_DATA: .word 0x10008
A: .word 25
B: .word 50
```

```
.code
ld $t2,DIR_CONTROL($0)
ld $t3,DIR_DATA($0)

ld $t0,A($0)
ld $t1,B($0)
```

```
dadd $t5, $t0,$t1
sd $t5, 0($t3)
; 1 → Código de imprimir entero
daddi $t6, $0, 1
sd $t6, 0($t2)
halt
```

```
ld $t2,DIR_CONTROL($0)
ld $t3,DIR_DATA($0)
ld $t0,A($0)
ld $t1,B($0)
dadd $t5,$t0,$t1
sd $t5,0($t3)
daddi $t6,$0,1
sd $t6,0($t2)
halt
```



Mostrar en pantalla un vector de números

```
.data
DIR_CONTROL: .word 0x10000
DIR_DATA: .word 0x10008
vec: .word -25,10,-12,15,50,3
```

```
.code
ld $t5,DIR_CONTROL($0)
ld $t6,DIR_DATA($0)
; 2 → código imp. neg
daddi $t4, $0, 2
```

```
daddi $t1, $0, 6
daddi $t2, $0, 0
loop: ld $t0,vec($t2)
      ; guardar valor en DATA
      sd $t0, 0($t6)
      ; código 2 en CONTROL
      sd $t4, 0($t5)
      daddi $t2,$t2,8
      daddi $t1, $t1, -1
      bnez $t1, loop
halt
```

Pintar el pixel (24,10) de violeta

```
.data
    coorX: .byte    24; coordenada X de un punto
    coorY: .byte    10; coordenada Y de un punto
    color: .byte    255, 0, 255, 0; rojo + azul ⇒ violeta
    DIR_CONTROL: .word32 0x10000 ; Dirección de CONTROL
    DIR_DATA: .word32 0x10008 ; Dirección de DATA

.text
    lwu    $s6, DIR_CONTROL($0); $s6 = dir. de CONTROL
    lwu    $s7, DIR_DATA($0); $s7 = dir. de DATA
    lbu    $s0, coorX($0) ; $s0 = coordenada X
    sb     $s0, 5($s7) ; DATA+5 = coordenada X ←
    lbu    $s1, coorY($0) ; $s1 = coordenada Y
    sb     $s1, 4($s7) ; DATA+4 = coordenada Y
    lwu    $s2, color($0) ; $s2 = color
    sw     $s2, 0($s7) ; DATA = color
    daddi   $s3, $0, 5; $t0 = 5 → salida gráfica
    sd     $s3, 0($s6) ; CONTROL = 5 → salida gráfica
    halt
```

Los registros: sus nombres y sus usos

\$zero	Siempre tiene el valor 0 y no se puede cambiar
\$ra	<i>Return Address</i> – Dir. de retorno de subrutina. Debe ser salvado
\$v0-\$v1	Valores de retorno de la subrutina llamada
\$a0-\$a3	Argumentos pasados a la subrutina llamada
\$t0-\$t9	Registros temporarios
\$s0-\$s7	Registros que deben ser salvados
\$sp	<i>Stack Pointer</i> – Puntero al tope de la pila. Debe ser salvado
\$fp	<i>Frame Pointer</i> – Puntero de pila. Debe ser salvado
\$at	<i>Assembler Temporary</i> – Reservado para ser usado por el ensamblador
\$k0-\$k1	Para uso del kernel del sistema operativo
\$gp	<i>Global Pointer</i> – Puntero a zona de memoria estática. Debe ser salvado

(r0)
(r31)
(r2-r3)
(r4-r7)
(r8-r15 y r24-r25)
(r16-r23)
(r29)
(r30)
(r1)
(r26-r27)
(r28)

Mostrar en pantalla "Hola" 10 veces

```
.data
DIR_CONTROL: .word 0x10000
DIR_DATA: .word 0x10008
str: .asciiz "Hola\n"

.code
ld $t2, DIR_CONTROL($0)
ld $t3, DIR_DATA($0)
; t0 = dir de str
daddi $t0, $0, str
sd $t0, 0($t3)
; código de imprimir str
daddi $t1, $0, 4
```

```
daddi $t4, $0, 10
loop: sd $t1, 0($t2)
      daddi $t4, $t4, -1
      bnez $t4, loop
      halt
```

\n : código ASCII del Enter

