Instituto Politécnico Superior – AUS

Trabajo Practico N° 4

Intel x86

Giovani Bavera – 1 año, comisión 1

Resumen:

La principal diferencia entre el ensamblador de Intel x86 y MIPS radican en la filosofía que sigue cada una de las arquitecturas. Por un lado, MIPS se centra en hacer instrucciones más simples y rápidas por su arquitectura de instrucciones reducidas (RISC). Por otro lado, la arquitectura x86 (CISC) posee instrucciones más complejas y variadas, lo que permite realizar tareas más diversas. Podríamos decir que x86 es un lenguaje que brinda mas herramientas u opciones en comparación a su contraparte, lo que termina derivando en una mayor complejidad. Esto se ve reflejado en sus registros, donde Intel tiene muchos registros específicos para diversas tareas. Si bien MIPS también cuenta con algunos, la mayoría de sus registros están mas orientados a uso generales. Por ejemplo, x86 tiene 14 registros para propósito general mientras MIPS tiene más del doble (32).

MIPS propone una convención de uso de registros, pero no son obligatorios, a diferencia de x86 que tiene registros dedicados para ciertas instrucciones o modos de direccionamiento, como FLAGS, SP, BP o IP.

Name	Register number	Usage
$zero	0	the constant value 0
$v0-$v1	2-3	values for results and expression evaluation
$a0-$a3	4-7	arguments
$t0-$t7	8-15	temporaries
$s0-$s7	16-23	saved
$t8-$t9	24-25	more temporaries
$gp	28	global pointer
$sp	29	stack pointer
$fp	30	frame pointer
$ra	31	return address

Esta es una convención de como programar en MIPS donde, por ejemplo, registros como $to - $t9 se usan para cosas temporales asi como $s0 - $s9 para permanentes.

Operadores y registros:

En cuanto a la cantidad y formato de los operandos en las instrucciones entre x86 y MIPS, x86 tiene instrucciones de longitud variable que pueden ir de 1 a 18 bytes, mientras que MIPS tiene instrucciones de 32 bits, es decir, posee una longitud fija.

x86 posee una cualidad un poco particular que es la posibilidad de no ser textual en las operaciones, permitiendo ser “implícito”. Es decir, en operaciones no requieren la especificación explícita de operandos o direcciones de memoria en las instrucciones. En lugar de eso, se utilizan ubicaciones de memoria predefinidas o registros dedicados que se asumen de manera implícita, como lo puede ser el registro acumulador.

```
; Caso implícito x86
   mov eax, 5  ; eax = 5
   add 5       ; agrega implícitamente 5 a eax
   inc         ; incrementa a eax en 1
```

Cuando se habla del registro acumulador, en x86 se refiere a EAX y se utiliza de forma implícita para operaciones aritméticas y lógicas. Estas instrucciones operan un operando de memoria o de otro registro con el valor del acumulador, guardando el resultado en el mismo. En

cambio, en MIPS, el acumulador es el $v0 y se utiliza para almacenar los valores de retorno de funciones o subrutinas. A diferencia de x86, no se usa para operaciones aritméticas y lógicas debido a la existencia de muchos más registros que pueden cumplir esa función.

Las instrucciones aritméticas en MIPS son más simples en comparación con x86 en términos de homogeneidad. En MIPS, las instrucciones utilizan 3 registros <destino, operando, operador>, mientras que en x86 se usan uno, dos o tres registros para realizar operaciones. Por ejemplo, puede ser *add 5* en MIPS, donde implícitamente se suma 5 al acumulador. Por otro lado, en MIPS están organizadas en tres formatos (tipo R, tipo I, tipo J), mientras que las de x86 son mucho más complejas y variadas, con diferentes números de operandos y longitud variable.

```
;en x86 puede ser:
  add eax, 5
  add 5

#Mientras que en MIPS si o si tiene que ser:
  add $t2, $t1, $t0 #Donde $t2 es el reg destino
                    #$t1 y $t0 son los operadores
```

Pilas y subrutinas:

El control de la pila, en MIPS se utiliza el registro $sp para apuntar al inicio de la pila, gestionando esta mediante operaciones como addi y subi para generar más o menos espacios y almacenar los datos. En x86, se emplean instrucciones específicas como push y pop (equivalentes a lo que hace addi y subi en MIPS) para manejar la pila. El crecimiento de la pila en los dos lenguajes es opuesto; en MIPS crece de forma decreciente mientras que en Intel su crecimiento es de forma creciente. Ambas siguen el principio de LIFO, ultimo en entrar primero en salir.

```
; Caso Intel
push eax        ; Reservar espacio en la pila para un valor
mov eax, 10     ; Valor a almacenar
push eax        ; Colocar el valor de 'eax' en la pila
pop eax         ; Liberar el espacio de la pila

# Caso MIPS
addi $sp, $sp, -4  # Hace espacio en la pila
sw $s0, 0($sp)     # Guardar el valor de $s0 en la pila
addi $sp, $sp, 4   # Liberar el espacio de la pila
```

En cuanto a las subrutinas, hay diferencias significativas en cómo se pasan los argumentos. En x86 se pasan a través de la pila, a diferencia de MIPS, donde se usan los registros específicos como *$a0* a *$a3*. En caso de tener más de 4 argumentos, se usa la pila. Por otro lado, la dirección de retorno en MIPS utiliza otro registro especial *($ra)* para guardar la dirección de retorno. x86 por su parte lo hace en la pila. En algo en lo que coinciden es la forma en la que se retorno el valor de la funcion, en ambos casos en un registro (*$vo* en MIPS y *EAX* en Intel).

```
; Caso x86
section .text
   global suma
suma:
    ; Los argumentos se pasan a través de la pila
    ; Primer argumento está en [esp+8], segundo argumento está en [esp+12]
    mov eax, [esp+8]    ; Primer argumento
    add eax, [esp+12]   ; Suma el primer y segundo argumento y almacena el resultado en eax
    ret                 ; Retorna

# Caso MIPS
main:
    li $a0, 5
    li $a1, 5
    jal suma
suma:
    # Los argumentos se pasan en $a0 y $a1
    add $v0, $a0, $a1   # Suma los argumentos y almacena el resultado en $v0
    jr $ra              # Retorna a la dirección de retorno ($ra)
```

FLAGS y su homólogo en MIPS:

Las FLAGS son algo bastante útil que posee x86 siendo indicadores de estado después de hacer operaciones aritméticas o lógicas. Registros que se modifican automáticamente con el valor correspondiente en base a la operación realizada. Flags como Zero Flag (ZF) que indica si el resultado de la operación fue igual a 0 o, Sign Flag que indica el signo del valor operado. Posteriormente estas son utilizadas para los saltos condicionales como jl, Jne o je.

Esto en MIPS no existe como tal. En su lugar se utilizan instrucciones de comparación como bne o beq para posteriormente saltar. Esto quiere decir que mientras x86 guarda los valores en los indicadores para posteriormente realizar la comparación, MIPS NO guarda ningún valor de la comparación, sino que únicamente compara los dos valores para posteriormente decidir si hacer el salto o no.

```
# Caso ejemplo en x86
cmp eax, ebx    ; Compara eax y ebx
je son_iguales  ; Salta si son iguales (ZF se activará si son iguales)

-----------------------------
# Caso ejemplo en MIPS
# Supongamos que los valores a comparar están en $t0 y $t1
    li $t0, 10      # Primer valor
    li $t1, 10      # Segundo valor

# Comparación de valores
beq $t0, $t1, son_iguales   # Salto si son iguales
# Si no son iguales (seguir ejecutando aquí)
# Código si los valores no son iguales
```