***Arquitectura de las Computadoras***

***Trabajo Práctico N.º 4***

***Assembly de la arquitectura Intel x86***

***Matías Haspert***

* Diferencias en cantidad y uso obligatorio o no de los registros:

En Intel x86 tenemos pocos registros (de uso general o especial) y son de longitud variable por ejemplo tenemos el registro EAX de 32 bits que es una versión extendida de AX (16 bits inferiores de EAX), a su vez AX se puede descomprimir en registros de 8 bits, AH (8 bits superiores) y AL (8 bits inferiores). No hay forma de acceder a los 16 bits superiores de EAX. A diferencia de x86, en MIPS existen 32 registros de tamaño fijo y solo de propósito general, todos de 32 bits.

* Importancia del registro acumulador:

En Intel x86 si hay registro acumulador que es el registro EAX, éste se usa para hacer operaciones aritméticas y lógicas. Mientras que en MIPS no hay registro acumulador y se usan los registros $t0, $t1, $t2, $t3, $t4, $t5, $t6, $t7 para hacer operaciones aritméticas y lógicas.

* Cantidad y forma de los operandos en las instrucciones y su complejidad de formato:

En MIPS todas las instrucciones son de 32 bits y tienen 3 formatos R-Type (3 operandos que son registros), I-Type (3 operandos y uno es inmediato) y J-Type(1 operando). A diferencia de MIPS, Intel x86 en general cada instrucción en sí misma tiene un número fijo de operandos (0 a 3) y pueden variar de tamaños (8, 16, 32 o 64 bits). Los operandos pueden tener de tipo registro, memoria, inmediato e implícito.

* Diferencias en las instrucciones aritméticas:

En Intel x86 las instrucciones aritméticas son más complejas que a las de MIPS porque solo tienen 2 operandos y también como dicho antes la longitud puede variar, y en MIPS se hacen las operaciones aritméticas con 3 operandos y todas de 32 bits por ejemplo la instrucción addi que sirve para sumar un valor inmediato a un registro, addi destino, fuente, valor inmediato y en x86 addi no existe, sería add destino, valor inmediato.

* Diferencias en cantidad y modos de direccionamiento. Modo implícito:

En MIPS tenemos 5 modos de direccionamiento y en Intel x86 hay 6 modos de direccionamiento, el modo que está en x86 y no en MIPS es el implícito, este modo les permite a los registros comportarse como variables apuntador. Por ejemplo:

mov ax, [Data] ; Direccionamiento directo de memoria de una palabra

mov ebx, Data ; ebx = &Data

mov ax, [ebx] ; ax = \*ebx

* Importancia de los FLAGS y su análogo en MIPS (ALU):

En x86 los FLAGS son muy importantes porque son los que indican el resultado de una operación aritmética o lógica. En MIPS no hay FLAGS, pero se utiliza la ALU, éste indica el resultado de una operación aritmética o lógica en un registro. La ALU es el análogo de los FLAGS en MIPS.

* Diferencias de evaluación de condiciones y su implicación en los saltos:

En MIPS por ejemplo una instrucción que evalúa condiciones es la beq que evalúa si dos registros son iguales y hace un salto condicional a la dirección de una etiqueta. En Intel x86 las condiciones se evalúan con la instrucción cmp y el resultado de una comparación se almacenan en el registro FLAGS para usarlas luego y para un salto condicional se utiliza la instrucción por ejemplo jnle(jump no less equal) que chequea en el registro FLAGS para determinar si salta o no.

* Diferencias en el manejo de pila:

En MIPS tenemos el registro $sp que apunta al tope de la pila y se utiliza para su manejo. Al momento de realizar un push (añadir datos a la pila) se tienen que restar 4 unidades al puntero de pila y para realizar un pop (eliminar elementos de la pila) después de extraer el dato se debe sumar 4 unidades al puntero. A diferencia de MIPS, Intel x86 ya tiene implementadas las operaciones sobre la pila: PUSH y POP que automáticamente ajustan el valor de ESP que es el registro encargado del manejo y apunta al elemento del tope de la pila y el registro EBP que apunta a la base de la pila.

* Diferencias en la gestión de subrutinas:

En MIPS las subrutinas se gestionan con las instrucciones jal (jump and link) que salta a una función y almacena en el registro $ra la dirección de retorno (PC+4), y para el retorno desde una función existe la instrucción jr (jump register, jr $ra). Los parámetros se pasan en los registros $a0-$a3 si hay más se almacenan en la pila y el valor de retorno en $v0. En Intel x86 las subrutinas se gestionan con CALL y RET, los parámetros se pasan por la pila y si se retorna algún valor se devuelve por el registro EAX.