Trabajo Practica No 4 - Assembly de la arquitectura Intel x86

Luka Lattanzi

Instituto Politecnico Superior

Arquitectura de las Computadoras I

# Comparación entre las Arquitecturas de MIPS e Intel x86

En este informe se realizará un análisis comparativo detallado entre dos arquitecturas de computadoras: MIPS e Intel x86. El objetivo de este informe es proporcionar una comprensión de las diferencias y similitudes entre las arquitecturas MIPS e Intel x86.

## Diferencias en Cantidad y uso Obligatorio o no de los Registros

### Arquitectura MIPS

La arquitectura MIPS es una arquitectura RISC que cuenta con un conjunto de 32 registros de propósito general, estos son conocidos como r0 a r31. El uso de estos es obligatorio para la mayoría de operaciones aritméticas y lógicas. Las instrucciones RISC en MIPS están diseñadas para operar directamente sobre los registros, y no en memoria. Esto significa que cualquier operación que involucre datos debe cargarse primero en los registros desde la memoria.

### Arquitectura Intel x86

La arquitectura Intel x86, que es una arquitectura CISC, tiene un número menor de registros de propósito general. Los registros de propósito general son EAX, EBX, ECX, EDX, ESI, EDI, EBP, y ESP, cada uno de 32 bits en la arquitectura x86 de 32 bits. A diferencia de MIPS, el uso de registros en x86 no es tan estrictamente obligatorio. Las instrucciones en x86 pueden operar tanto en registros como directamente en memoria.

## Importancia del Registro Acumulador

### Arquitectura MIPS

En la arquitectura MIPS, no hay un registro específico designado exclusivamente como acumulador. Cualquiera de los 32 registros de propósito general puede ser utilizado como acumulador dependiendo del contexto de la operación. Las operaciones aritméticas y lógicas en MIPS pueden usar cualquier registro de propósito general como destino, actuando de manera similar a un acumulador.

### Arquitectura Intel x86

En la arquitectura Intel x86, el registro EAX o AX es comúnmente utilizado como el registro acumulador. Este registro tiene un papel prominente en muchas operaciones aritméticas y lógicas, y algunas instrucciones están optimizadas para operar específicamente con el registro acumulador.

## Cantidad y Forma de los Operandos en las Instrucciones

### Arquitectura MIPS

Las instrucciones en MIPS generalmente tienen tres operadores. Este diseño se sigue estrictamente debido a su naturaleza RISC. La mayoría de las instrucciones aritméticas y lógicas en MIPS siguen el formato: instrucción destino, operando 1, operando 2.

### Arquitectura Intel x86

Las instrucciones en Intel x86 tienen una variedad más amplia de formatos de operandos debido a su naturaleza CISC. Las instrucciones pueden tener de cero a tres operandos, y la cantidad específica depende de la instrucción en particular.

## Diferencias en las Instrucciones Aritméticas

### Arquitectura MIPS

MIPS tiene un conjunto reducido y simplificado de instrucciones aritméticas debido a su naturaleza RISC. Las operaciones aritméticas básicas incluyen adición, sustracción, multiplicación y división. Las instrucciones aritméticas en MIPS generalmente siguen un formato de tres operaciones: operación destino, fuente 1, fuente 2.

Ejemplo: ADD $t0, $t1, $t2 - Suma los valores de $t1 y $t2 y almacena el resultado en $t0.

### Arquitectura Intel x86

Intel x86 tiene un conjunto más extenso y complejo de instrucciones aritméticas debido a su naturaleza CISC. Las operaciones aritméticas incluyen adición, sustracción, multiplicación, división, y muchas más operaciones específicas. Las instrucciones aritméticas en x86 pueden tener dos o tres operadores y pueden operar directamente sobre registros, memoria, o valores inmediatos.

Ejemplo: ADD EAX, EBX - Suma el valor de EBX al valor en EAX.

## Diferencias en Cantidad y Modos de Direccionamiento

### Arquitectura MIPS

MIPS, siendo una arquitectura RISC, tiene un conjunto reducido y simplificado de modos de dirección. Esto facilita la decodificación de instrucciones y mejora el rendimiento.

Modos de Direccionamiento en MIPS:

Dirección Inmediata: Utiliza un valor constante en la instrucción.

Ejemplo: ADDI $t0, $t1, 10 - Suma 10 al valor en $t1 y almacena el resultado en $t0.

Dirección de Registro: Utiliza los valores de los registros.

Ejemplo: ADD $t0, $t1, $t2 - Suma los valores de $t1 y $t2, y almacena el resultado en $t0.

Dirección Base + Desplazamiento: Utiliza un registro como base y un valor constante como desplazamiento para acceder a la memoria.

Ejemplo: LW $t0, 4($t1) - Carga el valor en la dirección calculada sumando 4 al valor en $t1 y almacena el resultado en $t0.

En MIPS, no hay modos implícitos como tal. Todas las instrucciones especifican explícitamente sus operandos, lo que se alinea con el diseño RISC.

### Arquitectura Intel x86

Intel x86, siendo una arquitectura CISC, tiene una mayor variedad de modos de dirección. Esto proporciona flexibilidad y capacidad para escribir instrucciones más compactas y específicas.

Modos de Direccionamiento en x86:

Dirección Inmediata: Utiliza un valor constante en la instrucción.

Ejemplo: MOV EAX, 10 - Mueve el valor inmediato 10 al registro EAX.

Dirección de Registro: Utiliza los valores de los registros.

Ejemplo: ADD EAX, EBX - Suma el valor de EBX al valor en EAX.

Dirección Directa: Accede directamente a una dirección de memoria especificada.

Ejemplo: MOV EAX, [1234h] - Mueve el valor en la dirección 1234h al registro EAX.

Dirección Indirecta: Utiliza un registro para almacenar la dirección de memoria.

Ejemplo: MOV EAX, [EBX] - Mueve el valor en la dirección almacenada en EBX al registro EAX.

Dirección Base + Índice: Utiliza dos registros, uno como base y otro como índice, y opcionalmente un desplazamiento constante para acceder a la memoria.

Ejemplo: MOV EAX, [EBX + ESI] - Mueve el valor en la dirección calculada sumando los valores en EBX y ESI al registro EAX.

Dirección Base + Índice + Desplazamiento: Similar al anterior, pero con un desplazamiento constante adicional.

Ejemplo: MOV EAX, [EBX + ESI + 4] - Mueve el valor en la dirección calculada sumando los valores en EBX, ESI y 4 al registro EAX.

Dirección Escalada: Utiliza un registro multiplicado por un factor de escala, junto con una base y/o un desplazamiento.

Ejemplo: MOV EAX, [EBX + ESI\*4] - Mueve el valor en la dirección calculada sumando el valor en EBX y ESI multiplicado por 4 al registro EAX.

En x86, algunas instrucciones tienen operandos implícitos, lo que significa que ciertos registros o valores se utilizan automáticamente sin ser especificados en la instrucción.

Ejemplo: MUL EBX - Multiplica el valor en EBX por el valor en EAX y almacena el resultado en EAX (registro acumulador). Aquí, EAX se usa implícitamente.

## Complejidad del Formato de las Instrucciones

### Arquitectura MIPS

MIPS sigue el diseño RISC, que enfatiza la simplicidad y regularidad. Todas las instrucciones tienen un tamaño fijo de 32 bits, lo que facilita la decodificación y ejecución.

Formato R (Registro):

Estructura: Opcode (6 bits), rs (5 bits), rt (5 bits), rd (5 bits), shamt (5 bits), funct (6 bits).

Ejemplo: ADD $d, $s, $t se decodifica con un opcode para ADD, registros fuente $s y $t, registro destino $d, y un campo de función.

Formato I (Inmediato):

Estructura: Opcode (6 bits), rs (5 bits), rt (5 bits), inmediato (16 bits).

Ejemplo: ADDI $t0, $t1, 10 suma el valor inmediato 10 al registro $t1.

Formato J (Salto):

Estructura: Opcode (6 bits), dirección (26 bits).

Ejemplo: J 10000 salta a la dirección especificada.

### Arquitectura Intel x86

x86 sigue el diseño CISC, que permite una mayor variedad de instrucciones con tamaños y formatos variables. Las instrucciones pueden tener de 1 a 15 bytes de longitud.

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

### Arquitectura MIPS

En la arquitectura MIPS, no existe un registro FLAGS como en x86. En cambio, los resultados de las operaciones aritméticas y lógicas se manejan de manera más explícita. En lugar de depender de FLAGS, MIPS utiliza instrucciones de comparación y salto, como BEQ (Branch if Equal), BNE (Branch if Not Equal), BGEZ (Branch if Greater Than or Equal to Zero), etc. Estas instrucciones realizan comparaciones explícitas y toman decisiones basadas en los registros.

### Arquitectura Intel x86

El registro FLAGS (o EFLAGS en las versiones extendidas de 32 bits y RFLAGS en 64 bits) contiene una serie de bits de estado que reflejan los resultados de operaciones aritméticas y lógicas. Los FLAGS se utilizan en instrucciones de salto condicional, como JZ (Jump if Zero), JNZ (Jump if Not Zero), JO (Jump if Overflow), etc. Estas instrucciones permiten alterar el flujo de ejecución del programa basándose en el estado de los FLAGS.

## Diferencias de Evaluación de Condiciones y su Implicación en los Saltos

### Arquitectura MIPS

En MIPS, las condiciones se evalúan mediante instrucciones específicas de comparación, como SLT (Set on Less Than), que almacenan el resultado en un registro. Los saltos condicionales, como BEQ (Branch if Equal) y BNE (Branch if Not Equal), dependen directamente del resultado almacenado en los registros tras una comparación explícita. Esto resulta en una simplicidad y regularidad en la decodificación y ejecución, ya que la separación clara entre la evaluación de condiciones y los saltos condicionales facilita un control de flujo explícito y predecible en el programa.

### Arquitectura Intel x86

La arquitectura Intel x86 utiliza el registro FLAGS para almacenar los resultados de las operaciones aritméticas y lógicas. Bits específicos del registro FLAGS (como ZF, SF, CF) indican condiciones como cero, signo, acarreo y desbordamiento. Los saltos condicionales en x86, como JE (Jump if Equal), JNE (Jump if Not Equal), JG (Jump if Greater) y JL (Jump if Less), dependen del estado de estos bits en FLAGS. Aunque el uso de FLAGS permite que una sola instrucción afecte múltiples condiciones de estado, haciendo el código más compacto, también añade complejidad a la decodificación y ejecución de instrucciones, ya que el estado de FLAGS debe ser evaluado continuamente.

## Diferencias en el Manejo de Pila

### Arquitectura MIPS

En MIPS, el manejo de la pila es explícito y se realiza mediante instrucciones específicas de carga y almacenamiento. Los desarrolladores deben gestionar la pila manualmente utilizando registros específicos, como el registro $sp (stack pointer). Las instrucciones típicas incluyen SW (Store Word) y LW (Load Word) para almacenar y cargar datos en la pila. Por ejemplo, para empujar un valor en la pila, se decrementa el $sp y luego se usa SW para almacenar el valor en la nueva dirección de la pila. Este enfoque ofrece simplicidad y control explícito sobre la gestión de la pila, pero requiere que el programador manualmente el crecimiento y decrecimiento de la pila, aumentando la responsabilidad de la correcta gestión del espacio de la memoria.

### 

### 

### Arquitectura Intel x86

En x86, el manejo de la pila está más integrado en el conjunto de instrucciones, con instrucciones dedicadas como PUSH y POP para empujar y extraer valores de la pila respectivamente. Estas instrucciones automáticamente ajustan el registro ESP (Extended Stack Pointer) que apunta al tope de la pila. Además, x86 proporciona instrucciones como CALL y RET para las llamadas a procedimientos y retornos, que también gestionan automáticamente la pila almacenando y recuperando las direcciones de retorno. Este enfoque automatizado simplifica el manejo de la pila desde la perspectiva del programador, pero también introduce una capa de abstracción que puede ocultar detalles importantes de la gestión de la memor

## Diferencias en la Gestión de Subrutinas

### Arquitectura MIPS

En MIPS, la gestión de subrutinas es explícita y directa. La llamada a una subrutina se realiza mediante la instrucción JAL (Jump and Link), que guarda la dirección de retorno en el registro $ra (return address) y salta a la dirección de la subrutina. El retorno de la subrutina se efectúa utilizando la instrucción JR (Jump Register), que emplea la dirección de retorno almacenada en $ra para volver a la instrucción siguiente después de la llamada a la subrutina. En cuanto a la gestión de la pila, en MIPS esta es manual; el programador debe gestionar el almacenamiento y recuperación de registros necesarios en la pila mediante las instrucciones SW (Store Word) y LW (Load Word). Además, se siguen convenciones de llamada donde ciertos registros deben ser guardados por el llamador o la subrutina.

### 

### Arquitectura Intel x86

En la arquitectura Intel x86, la gestión de subrutinas está más integrada y automatizada. La llamada a una subrutina se realiza con la instrucción CALL, que automáticamente empuja la dirección de retorno en la pila y salta a la dirección de la subrutina. El retorno de la subrutina se realiza mediante la instrucción RET, que automáticamente saca la dirección de retorno de la pila y salta de vuelta a esa dirección. La gestión de la pila en x86 es también más automatizada, con las instrucciones CALL y RET manejando automáticamente la pila para empujar y sacar la dirección de retorno, respectivamente. Además, se utilizan las instrucciones PUSH y POP para gestionar otros registros si es necesario. Al igual que en MIPS, existen convenciones de llamada, pero la automatización en x86 reduce la carga sobre el programador.