Sistemas Digitales

Instrucciones:
Lenguaje de la
Computadora

¿Qué veremos en este capítulo?

- Las instrucciones. Desde el nivel más alto hasta el más bajo.
- Los lenguajes. Su diversidad y similitud
- Instrucciones. La representación mediante hardware, partiendo desde el lenguaje de alto nivel. Concepto de programa almacenado.
- Estándar a utilizar. A pesar que los otros estándares son muy parecidos.

Primer paso: Conocer el lenguaje

- Las instrucciones son las palabras del lenguaje que “habla” una computadora. El conjunto o set de instrucciones constituye el vocabulario de dicho lenguaje.
- Los set de instrucciones de los diferentes hardware son dialectos, es decir, básicamente todos hablan lo mismo pero en distintos formatos
- La aparente similitud se debe al diseño de las computadoras que emplean los mismos principios básicos.
- Las instrucciones son parte de la arquitectura de una computadora.

Lenguaje de Alto Nivel vs Hardware

- Los Lenguajes de Alto Nivel emplean un formato de escritura parecido al humano, pero para que el hardware pueda realizar las tareas escritas en este lenguaje, se debe traducir a un formato que el hardware conozca.
- El Lenguaje de Alto Nivel ofrece varios beneficios:
  - Permiten al programador usar un lenguaje natural, siendo los programas más parecidos al lenguaje real.
  - Incrementa la productividad del programador, pues un lenguaje natural hace concisa la escritura de programas.
  - Los programas pueden ser independientes de la computadoras, pues se puede tener un compilador para cada una de las diversas arquitecturas.

Lenguaje de Alto Nivel vs Hardware

- Los Lenguajes de Alto Nivel tienen muchos beneficios, pero en el diseño del hardware se debe tener en cuenta los lenguajes de más bajo nivel.
- Estos últimos llevan una forma de escritura cercana a los 0s y 1s, pero aún emplean símbolos más entendibles a nivel humano.
- Entonces los primeros programadores crearon un programa que hiciera la traducción automática de estos símbolos y así nació el assembler.
- Assembler constituye el lenguaje de más bajo nivel antes de pasar al lenguaje de máquina puro (0s y 1s)

Instrucciones: Operaciones

- Todas las computadoras deben realizar operaciones aritméticas.
- En un lenguaje de alto nivel se escribiría así
  - *a=b+c*
- y en MIPS
  - *add a,b,c*
- La representación limita el uso de tres elementos para la acción. ¿Qué pasa si deseamos sumar *b*, *c*, *d* y *e*; y el resultado colocarlo en *a*?

Instrucciones: Operaciones

- Usar tres elementos corresponde a la filosofía que el hardware sea sencillo y corresponde a uno de los principios de diseño de hardware.
- *Principio de Diseño 1:* La simplicidad favorece la regularidad.
- La palabra *add* se conoce como nmemónico.
- Los operandos fuente son *b* y *c*; y el destino es *a*.
- ¿Cuál sería la representación para?
  - *d=a-e*
  - *f=(g+h)-(i+j)*

Operandos

- A diferencia de los lenguajes de alto nivel, la cantidad de operandos de las instrucciones son limitados y deben estar construidos el hardware.
- Los elementos de hardware para este propósito se llaman registros.
- Estos registros son primitivas (elementos básicos) utilizados en el diseño del hardware.
- En MIPS los registros son 32 bits y la cantidad de estos en el hardware es de sólo 32.

Operandos

- Las operaciones aritméticas sólo pueden usar estos registros como sus operandos.
- La cantidad limitada de registro nos lleva al:
- *Principio de Diseño 2: Cuanto más pequeño, más rápido.*
- Si la cantidad de registros es mayor (como en la memoria) el acceso tendría un retardo muy alto.
- Además se contempla la cantidad de bits en la instrucción para identificar estos registros.

Registros según MIPS

Name	Number	Use
$0	0	the constant value 0
$at	1	assembler temporary
$v0-$v1	2–3	procedure return values
$a0-$a3	4–7	procedure arguments
$t0-$t7	8–15	temporary variables
$s0-$s7	16–23	saved variables
$t8-$t9	24–25	temporary variables
$k0-$k1	26–27	operating system (OS) temporaries
$gp	28	global pointer
$sp	29	stack pointer
$fp	30	frame pointer
$ra	31	procedure return address

Operandos

- Si queremos realizar la operación: a = b + c – d y la representamos en assembler sería:
  - `Sub t,c,d`
  - `Add a,b,t`
- Ahora lo escribiremos de la siguiente forma:
  - `Sub $t0,$s0,$s1`
  - `Add $s3,$s2,$t0`
- Los operandos con prefijo $ corresponden a los registros físicos. En el ejemplo los registros $s se conoce como saved y los $t como temporary.

Operandos

- A pesar que los registros tienen sus nombres, en el hardware la única forma de poder trabajar con ellos es seleccionándolo.
- El selector es equivalente al bus de direcciones de una memoria.

Operandos en Memoria

- Los lenguajes pueden almacenar los valores más usados en los registros, pero también manejan grandes y complejas estructuras de datos utilizando memorias.
- El banco de registros es pequeño y rápido mientras que la memoria es grande y lenta.
- Para trabajar con memoria, necesitamos instrucciones específicas, y se les llama *instrucciones de transferencia de datos*.
- Dos de las instrucciones usadas son `lw` (load word) y `sw` (save word).

Operandos de Memoria

- La instrucción `lw` carga un valor desde memoria a uno de los registros y la instrucción `sw` guarda un valor de uno de los registros en la memoria. En ambos casos se debe indicar la dirección de la posición en la memoria que se desea acceder.
- Tanto `lw` como `sw` deben especificar la dirección de memoria con la cual se trabajará. Para ello usa el formato: dirección = base + offset.
  - Base: Está contenido en un registro e indica la dirección de referencia para el acceso (punto de partida)
  - Offset: Cantidad de posiciones que debe desplazarse desde la base para llegar a la posición deseada.

Operandos de Memoria

- Por ejemplo si queremos utilizar de una tabla en memoria el octavo dato a partir de la referencia (base) se escribiría: g ← h + A[8].
- La RAM que usamos es de 8 bits por posición, pero los datos que se manejan en MIPS son de 32 bits entonces debemos emplear 4 posiciones de la RAM por cada dato utilizado en MIPS.
- Por ejemplo tenemos : `lw $t0,8($s3)` donde
  - En t0 se guarda el dato que viene de la memoria.
  - En s3 está la dirección base (referencia).
  - El número 8 indica el dato dos; pues para cada dato el sistema cuenta cuatro posiciones/direcciones.

Operandos de Memoria

- Al descomponer o recomponer el dato de 32 bits, el orden usado en la memoria es del formato Big Endian.
- Entonces en la memoria el primer byte que se almacena o lee es el byte más significativo del dato.
- Por ejemplo, como se escribiría en assembler la siguiente instrucción:

A[12] = h + A[8]

Operandos Constantes o Inmediatos

- En los programas también es común usar constantes en una operación.
- Una alternativa que evita cargar desde memoria implica que uno de los operandos sea una constante. A esta instrucción se le conoce como inmediata. En el caso de la suma tenemos:
  - `addi $s3,$s2,4` (s3 <= s2 + 4)
- Esta nueva forma de usar operandos implica que el caso común es más rápido.

Operandos Constantes o Inmediatos

- Las constantes son operandos que frecuentemente se utilizan y el colocarlos directamente en la instrucción simplifica su acceso, pues no hay que buscarlo en la memoria ni en los registros. El acceso a este dato implica simplemente leer la instrucción.
- Debido a la estructura de la arquitectura la constante que es de 32 bits sólo se puede escribir con 16 bits, por lo tanto el hardware debe convertirlo a 32 bits para que se pueda operar.
- Las constantes siempre se escriben en modo complemento a 2: el bit más significativo es el signo y los otros quince conforman el número.

Representación de las instrucciones

- Las instrucciones en una computadora están guardadas como un conjunto de señales electrónicas altas y bajas y se pueden interpretar como números (binarios).
- Cada parte de la que esta conformada una instrucción se considera un número y al unir esas partes obtenemos la instrucción.
- Por ejemplo, los registros como $s0, $s1, etc., a pesar de representarse como caracteres al escribir la instrucción, para el hardware sólo se puede identificar con su número en binario (por ejemplo $s0 es el registro 16)

Representación de las instrucciones

- Si tomamos como ejemplo la instrucción:
  - `add $t0,$s1,$s2`
- Al representarlo como números decimales:

0	17	18	8	0	32

- Cada división se llama campo y en el ejemplo:
  - El primer y último campo combinados indican que se realizará una suma.
  - Los números 17, 18 y 8 identifican los registros con los que trabaja la instrucción: $s1, $s2 y $t0.
  - El quinto (penúltimo campo) en este caso no se usa y se le asigna cero.
- Y el formato para escribirlo en binario es:

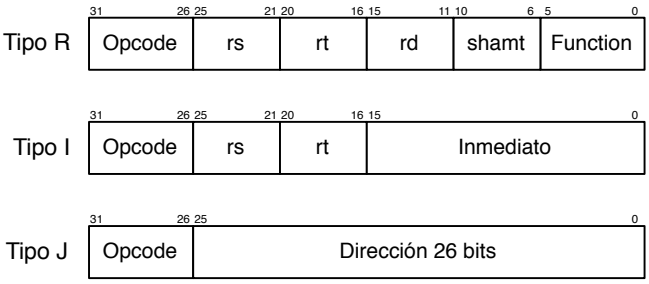
Representación de las instrucciones

- Para no confundir términos, las instrucciones en versión binaria se conocen como lenguaje de máquina mientras que si usan los nombres alfanuméricos será lenguaje ensamblador.
- En MIPS, una instrucción siempre consta de 32 bits pero se presenta un problema cuando se manejan diversos tipos y cantidades de operandos y formatos.
- Por ejemplo `add` y `sub` usan tres registros como operandos mientras que `lw` y `sw` usan dos registros y una constante.

Representación de las instrucciones

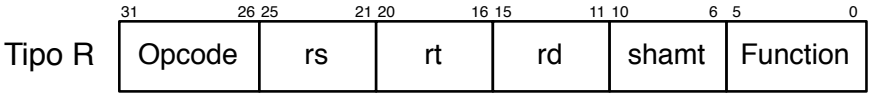
- Esto nos lleva al:
- *Principio de Diseño 4: Un buen diseño exige buenos compromisos.*
- En este caso el compromiso es mantener siempre la misma longitud (32 bits) para todas las instrucciones a pesar que requieren distintos campos.
- Por ese motivo existen tres formatos básicos para instrucciones cuyos campos difieren en tipo y tamaño, pero manteniendo siempre los 32 bits en total.

Campos



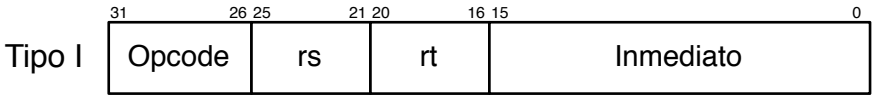
- Opcode: Código de operación.
- rs y rt: Normalmente son los operando fuentes, pero rt puede ser destino en la instrucción `sw`.
- rd : Operando destino.
- shamt : Usado para operaciones de desplazamiento.
- Function : En combinación con Opcode define la operación que se va a realizar.
- Inmediato : Número de 16 bits en complemento a 2.
- Dirección 26 bits : Usado para saltos.

Instrucción Tipo R



- Por ejemplo tenemos: add $t0,$s2,$s1
  - Opcode (000000): Operación aritmética.
  - rs (10010): Registro s2
  - rt (10001): Registro s1
  - rd (01000): Registro t0
  - shamt (00000): No hay desplazamiento.
  - Function (100000): Operación aritmética suma
- Código 00000010010100010100000000100000
- En hexadecimal 0x02514020

Instrucción Tipo I



- Por ejemplo tenemos: lw $s3,12($t1)
  - Opcode (100011): Cargar de memoria a registro.
  - rs (01001): Registro t1
  - rt (10011): Registro s3
  - Inmediato (0000000000001100): Desplazamiento de 12 posiciones en memoria (dato [3]).
- Código 10001101001100110000000000001100
- En hexadecimal 0x8D33000C

Almacenamiento de Programas

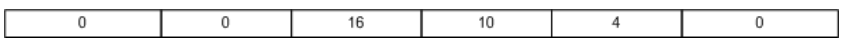
- Los programas se almacenan en memoria en formato binario, por ejemplo:
  - `lw $t2,32($t0)`
  - `add $s0,$s1,$s2`
  - `addi $t0,$s3,-12`
  - `sub $t0,$t3,$t5`
- Ocupará 16 posiciones en memoria de código.
- Estas instrucciones serán “recogidas” una a una para que el hardware lo ejecute.

Operaciones Lógicas

- A pesar que las operaciones aritméticas son fundamentales en un procesador, las operaciones lógicas son también bastante útiles por diversos motivos.
- Tenemos dentro de ellas a las básicas (AND, OR y NOR) que se realizan entre los operandos bit a bit.
- También tenemos las de desplazamiento que nos permiten modificar los datos en base a mover sus bits uno a uno.

Operaciones Lógicas

- La operación más sencilla es el desplazamiento. Consiste en desplazar los bits una cantidad de posiciones hacia la derecha o izquierda, agregando ceros donde quedan lugares vacios.
  - Por ejemplo para el siguiente número binario 00000000000000000000000000001001
  - Si lo desplazamos 4 posiciones a la izquierda resulta 00000000000000000000000010010000
- Y en instrucción sería `sll $t2,$s0,4` y su formato es (Tipo R):

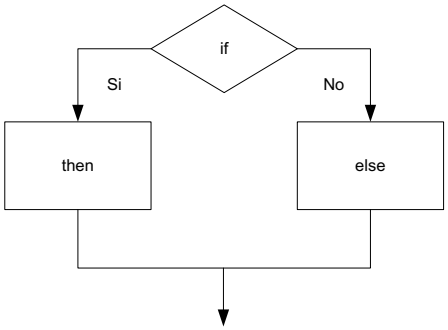


Operaciones Lógicas

- Las operaciones lógicas básicas (AND, OR, NOR) también son utilizadas.
  - Por ejemplo si ejecutamos la operación AND entre los número binarios 00000000000000000000110100000000 y 00000000000000000001111000000000
  - La operación se realiza bit a bit entre ambos números resultando 00000000000000000000110000000000
- Y la instrucción es and $t0,$t1,$t2

Instrucciones para Tomar Decisiones

- Una de las principales diferencias de una computadora respecto a una calculadora es su capacidad de tomar decisiones.
- Basado en una condición se pueden ejecutar diferentes instrucciones.



Instrucciones para Tomar Decisiones

- Las instrucciones para tomar decisiones son
  - beq registro1, registro2, etiqueta
  - bne registro1, registro2, etiqueta
- Estas instrucciones tienen como base la instrucción de alto nivel if. Por ejemplo si deseamos ejecutar:
  - if (i==j) then f = g + h; else f = g – h
- Las instrucciones para realizar esta tarea serían:

```
        bne $s3,$s4,Else
        add $s0,$s1,$s2
        j Exit
Else:   sub $s0,$s1,$s2
Exit:
```


Instrucciones para Tomar Decisiones

- El uso de las etiquetas Else y Exit evita que el programador tenga que realizar los cálculos para saltar a las líneas donde las instrucciones se van a ejecutar.

Modos de Direccionamiento

- Las múltiples formas de direccionamiento son llamados los modos de direccionamiento. En MIPS estos son:
  - *Inmediato*, donde el operando es una constante dentro de la instrucción.
  - *Registro*, donde el operando es un registro.
  - *Base* o *desplazamiento*, donde el operando está en la posición de memoria cuya dirección es la suma de un registro y una constante en la instrucción.
  - *Relativo a PC*, donde la dirección es la suma de la PC y una constante en la instrucción.
  - *Pseudodirecto*, donde la dirección de salto son los 26 bits de la instrucción concatenado con los bits altos del PC.