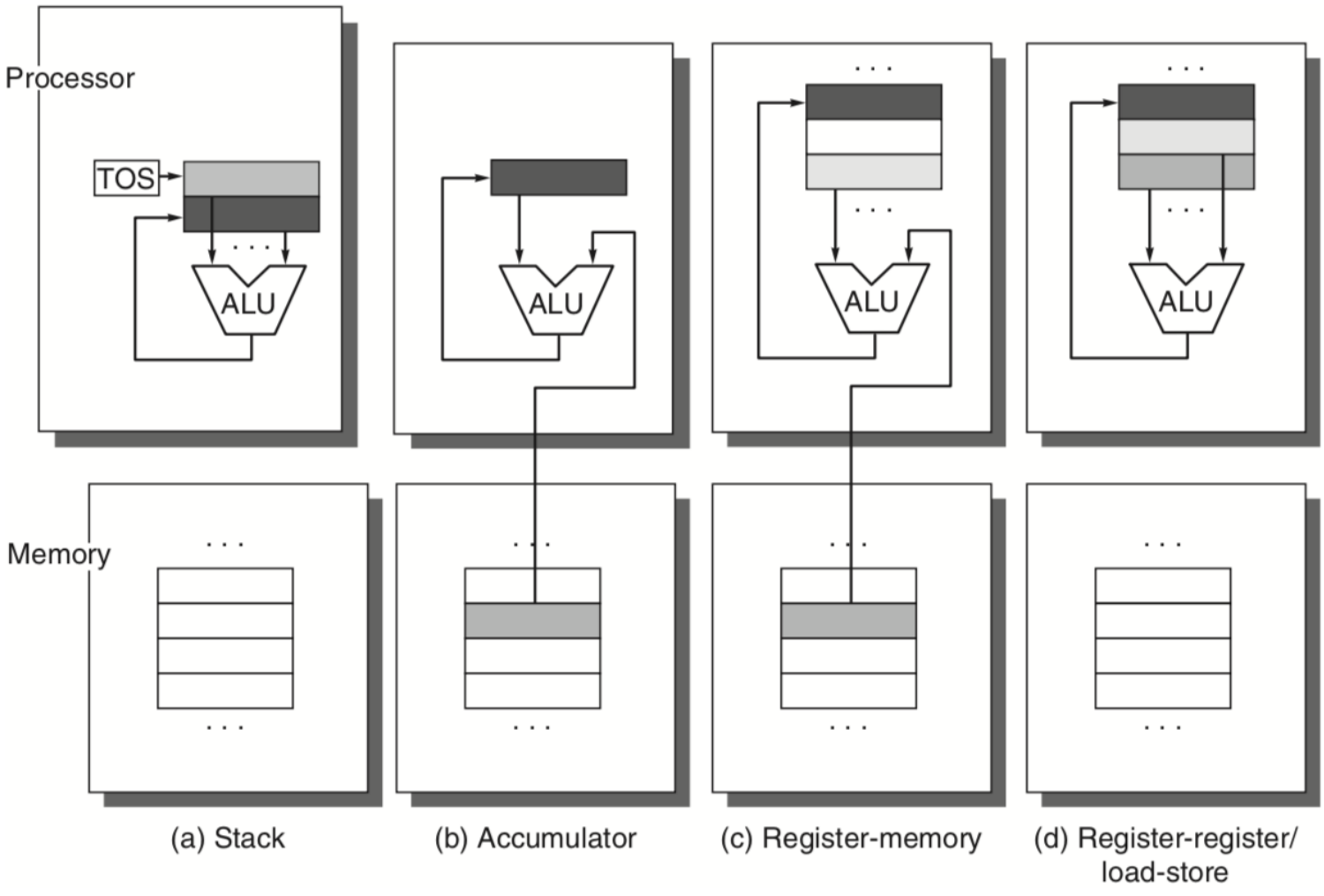
U.B.A. - Facultad de Ingeniería

66.20/86.37 Organización de Computadoras Arquitecturas

Práctica

1^{er} cuatrimestre 2020

Clasificación de ISAs



Clasificación de ISAs

- Modelo de pila: Los operandos están implícitamente en el tope del stack y el hardware debe evaluar la expresión en un solo orden y cargar un operando múltiples veces.
- Modelo de acumulador: En una arquitectura de acumulador un operando está implícito en el acumulador.
- Modelo de registros de propósito general: Aquí se tienen únicamente operandos explícitos, ya sea que estén ubicados en registros o en memoria.
- Modelo de carga y almacenamiento: En este tipo de arquitectura, la memoria solo puede ser accedida a través de instrucciones de carga y almacenamiento (load/store).

Tipo de instrucciones en MIPS

- Load / Store: únicas que acceden a memoria.
- Computational: realizadas en la ALU.
- Jump / Branch: saltos incondicionales y condicionales.
- Miscelaneas: Serialización de instrucciones, Excepciones (syscall), Moves condicionales, Prefetch, NOPs.
- Coprocessors y FPU.

Ejercicio 2.1 CAAQA 3ed

Asumir que los valores A, B, C, D y E residen en memoria, que el opcode de la instrucción es de 8 bits, las direcciones de memoria son de 64 bits y las direcciones de registro son de 6 bits.

a. Por cada arquitectura del conjunto de instrucciones mostradas en el primer slide ¿Cuántas direcciones o nombres aparecen en cada instrucción para calcular C = A + B y cuál es el tamaño total del código?

b. Algunas ISAs de la figura del primer slide destruyen los operando durante el cómputo y trae una consecuencia de performance. Para cada arquitectura escribir la secuencia de código para calcular C = A + B seguido de D = A - E. En el código, indicar cada operando que se destruye durante la ejecución e indicar en cada una el "overhead" que se incluye como solución. Dar el tamaño en total del código, los datos que son movidos de o hacia memoria y el overhead de cada secuencia de código.

Ejercicio 2.1 CAAQA 3ed - Resolución

a.

Stack	Accumulator	Register (register-memory)	Register (load-store)
Push A	Load A	Load R1,A	Load R1,A
Push B	Add B	Add R3,R1,B	Load R2,B
Add	Store C	Store R3,C	Add R3,R1,R2
Pop C			Store R3,C

$$C = A + B$$

Ejercicio 2.1 CAAQA 3ed - Resolución

b.

Code size = opcode + memory operand = 8 bits + 64 bits = 72 bits

Stack Code	Comment	Code size (bits)	Data size (bytes)
Push A	load from memory	72	n
Push B		72	n
Add	operands A and B destroyed	8	0
Pop C	save result to memory	72	n
Push E		72	n
Push A	overhead instruction; reloading A onto stack	72	n
Sub	operands A and E destroyed; order of operands on stack determines which is minuend and subtrahend	8	0
Pop D		72	n

El tamaño del código es 448 bits (56 bytes), el total de datos movidos de ó hacia memoria es 6n bytes. Hay una instrucción y un operando que generan overhead.

Ejercicio 2.1 CAAQA 3ed - Resolución

b.

Accumulator Code	Comment	Code size (bits)	Data size (bytes)
Load A		72	n
Add B	operands A and B destroyed	72	n
Store C		72	n
Load A	overhead instruction; reloading A	72	n
Sub E	operands A and E destroyed	72	n
Store D		72	n

El tamaño del código es 432 bits (54 bytes), el total de datos movidos de ó hacia memoria es 6n bytes. Hay una instrucción y un operando que generan overhead.

Ejercicio 2.1 CAAQA 3ed - Resolución

b.

Code size = opcode + register operand + memory operand = 8 bits + 6 bits + 64 bits = 78 bits

Register-Memory Code	Comment	Code size (bits)	Data size (bytes)
Load R1,A		78	n
Add R1,B	operands A and B destroyed	78	n
Store C, R1		78	n
Load R1,A	overhead instruction; reloading A	78	n
Sub R1,E	operands A and E destroyed	78	n
Store D,R1		78	n

El tamaño del código es 468 bits (59 bytes), el total de datos movidos de ó hacia memoria es 6n bytes. Hay una instrucción y un operando que generan overhead.

Ejercicio 2.1 CAAQA 3ed - Resolución

b.

Load-Store Code	Comment	Code size (bits)	Data size (bytes)
Load R1,A		78	n
Load R2,B		78	n
Add R3,R2,R1	no operands destroyed	26	0
Store C,R3		78	n
Load R4,E		78	n
Sub R5,R1,R4	no operands destroyed	26	0
Store D,R5		78	n

El tamaño del código es 442 bits (55 bytes), el total de datos movidos de ó hacia memoria es 5n bytes. No hay overhead en las instrucciones ni en los operando de datos.

Ejercicio 2.2 CAAQA 3ed

Algunas operaciones con dos operando no son conmutativas (la resta por ejemplo). ¿Qué ventajas y desventajas en las arquitecturas de pila, acumulador y carga-almacentamiento existen cuando se ejecuta operaciones no conmutativas?

Ejercicio 2.2 CAAQA 3ed - Resolución

- **Stack**
  - Ventajas: El tamaño de las instrucciones es pequeño porque no hay que ubicar los operando o el resultado.
  - Desventajas: En general los operando tienen que estar en el orden correcto. Una instrucción útil podría ser para intercambiar el tope de pila.
- **Acumulador**
  - Ventajas: La codificación de las instrucciones es pequeña ya que hay un solo operando.
  - Desventajas: Lo mismo que para la pila, los operando deben estar en el orden correcto. Una instrucción útil podría ser para intercambiar el contenido del acumulador con cualquier ubicación de un operando.
- **Load-Store**
  - Ventajas: Ya que los operando están en registros se pueden intercambiar cuando sea necesario.
  - Desventajas: Tamaño más grande para encodear las instrucciones.

Endianess

Todo el conjunto de instrucciones es direccionado por bytes y se provee acceso por bytes (8 bits), mitad de palabra (half words) (16 bits), por palabra (words) (32 bits).

Hay dos convenciones para ordenar los bytes en un objeto. **Little Endian** y **Big Endian**.

Endianess

El orden **Little Endian** coloca el byte cuya dirección "x . . . x000" en la posición menos significativa. Los bytes entonces son numerados:

El orden **Big Endian** coloca el byte cuya dirección es "x . . . x000" en la posición más significativa. Los bytes entonces son numerados:

Alineación

En varias computadoras, los accesos a objetos más grandes que un byte deben estar alineados.

- Un objeto de tamaño s bytes en la dirección de bytes A está alineado si A mod s $= 0$.
- Esto se fuerza en muchas arquitecturas porque el hardware accede a memoria típicamente alineado a múltiplos de word o double-word.

¿Qué puede ocurrir a nivel de accesos a memoria si se accede a datos no alineados y la arquitectura lo permite?

Modos de direccionamiento

Se presentan varios modos de direccionamiento:

- Register
- Immediate
- Displacement
- Register indirect
- Indexed
- Direct or Absolute
- Memory indirect
- Autoincrement
- Autodecrement
- Scaled

MIPS implementa 2 modos de direccionamiento: Immediate y Displacement, ambos con immediate de 16 bits.
Usando el registro 0 se consigue Absolute, y con displacement 0 se consigue Register indirect.

Modos de direccionamiento

```
Immediate:        add    $t1, $t2, 10
Displacement:     lw     $t1, 30($t2)
Absolute:         sb     $t1, 40($0)
Indirect:         sw     $t1, 0($t2)
```

Ejercicio 2.3 CAAQA 3ed

El valor representado por el número hexadecimal 434F 4D50 5554 4552 se almacena en una palabra doble alineada en 64 bits.

a. Utilizando el arreglo físico de la primera fila en la Figura 2.5, escribir el valor a ser almacenado utilizando el orden de byte Big Endian. Luego, interpretar cada byte como caracter ASCII y debajo de cada byte escribir el caracter correspondiente formando la cadena de caracteres que se almacenaría en el orden Big Endian.

b. Realizar el punto (a.) pero siguiendo el orden Little Endian.

Ejercicio 2.3 CAAQA 3ed

	Value of 3 low-order bits of byte address							
Width of object	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
1 byte (byte)	Aligned	Aligned	Aligned	Aligned	Aligned	Aligned	Aligned	Aligned
2 bytes (half word)	Aligned		Aligned		Aligned		Aligned	
2 bytes (half word)		Misaligned		Misaligned		Misaligned		Misaligned
4 bytes (word)	Aligned				Aligned			
4 bytes (word)		Misaligned				Misaligned		
4 bytes (word)			Misaligned				Misaligned	
4 bytes (word)				Misaligned				Misaligned
8 bytes (double word)	Aligned							
8 bytes (double word)		Misaligned						
8 bytes (double word)			Misaligned					
8 bytes (double word)				Misaligned				
8 bytes (double word)					Misaligned			
8 bytes (double word)						Misaligned		
8 bytes (double word)							Misaligned	
8 bytes (double word)								Misaligned

Fiigura 2.5: Direcciones de byte alineadas y no alineadas

Ejercicio 2.3 CAAQA 3ed - Resolución

43 4F 4D 50 55 54 45 52 almacenada en una palabra doble alineada en 64 bits.

a. Big Endian

43	4F	4D	50	55	54	45	52
C	O	M	P	U	T	E	R

b. Little Endian

52	45	54	55	50	4D	4F	43
R	E	T	U	P	M	O	C

Encoding MIPS

Hay 3 formatos de encoding posibles en MIPS

- I-type (Immediate)
- R-type (Register)
- J-type (Jump)

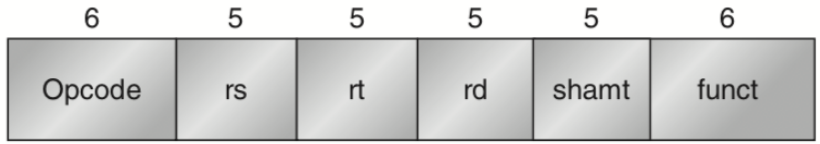
Encoding MIPS

I-type instruction

Encodes: Loads and stores of bytes, half words, words, double words. All immediates (rt ← rs op immediate)

Conditional branch instructions (rs is register, rd unused)
Jump register, jump and link register
(rd=0, rs=destination, immediate=0)

R-type instruction



Register-register ALU operations: rd ← rs funct rt
Function encodes the data path operation: Add, Sub, . . .
Read/write special registers and moves

J-type instruction

Jump and jump and link
Trap and return from exception

Encoding MIPS

Viendo el detalle de las instrucciones J-type vemos que hay un offset de 26 bits left-shifted 2 bits, para formar los 28 lsb de la direccion destino del salto (las regiones de salto están alineadas a 256MB, y la región está determinada por los 4 msb de PC).

¿Por qué se puede (y conviene) tomar los 26 bits como desplazados a la izquierda 2 bits?

Instrucción	Frecuencia
Load	26%
Store	10%
Aritméticas	14%
Comparaciones	13%
Saltos condicionales	16%
Saltos incondicionales	1%
Call / Return	2%
Shift	4%
Logicas	4%
Misc.	5%

Tabla 1: Mix. de instrucciones

Ejercicio 2.6 CAAQA 3ed

Varios investigadores han sugerido que incorporar un modo de direccionamiento registro-memoria a una máquina *Load/Store* puede ser conveniente. La idea es reemplazar la siguiente secuencia

```
LOAD   R1, 0(Rb)
ADD    R2, R2, R1
```

por

```
ADD    R2, 0(Rb)
```

Asumir que la nueva instrucción provoca un incremento en el ciclo de clock del 5%. Utilizar la frecuencia de instrucciones de la Tabla 1. La nueva instrucción afecta únicamente al ciclo de clock y no al CPI.

Ejercicio 2.6 CAAQA 3ed

a. ¿Qué porcentaje de loads deben ser eliminados en la máquina con la nueva instrucción para tener al menos el mismo rendimiento?

b. Mostrar una secuencia de instrucciones donde un load de R1 seguido inmediatamente por el uso de R1 (con algún tipo de opcode) no podría ser reemplazado por una única instrucción como la propuesta, asumir que el mismo opcode existe.

Ejercicio 2.6 CAAQA 3ed - Resolución

a.

Vamos a partir de la ecuación de desempeño antes vista y calcularemos primero reduciendo las instrucciones en su total y no solo los loads.

$$CPUTime = CPI \times CC \times IC$$

$$\begin{aligned} \text{CPU Time}_{old} &= CPI_{old} \times CC_{old} \times IC_{old} \\ \text{CPU Time}_{new} &= CPI_{new} \times CC_{new} \times IC_{new} \\ &= CPI_{old} \times (CC_{old} \times 1.05) \times (IC_{old} - R) \end{aligned}$$

Despejando R, tenemos que cantidad de IC_{old} habría que reducir

$$R = 0.048 \times IC_{old}$$

Entonces tenemos que reemplazar el $\frac{0.048 \times IC_{old}}{IC_{old} \times 0.26\text{loads}} = 18.4\%$ loads por la nueva instrucción.

Ejercicio 2.6 CAAQA 3ed - Resolución

b.

Consideremos la secuencia

```
LOAD    R1, 0(R1)
ADD     R1, R1, R1
```

El resultado que se escribe en R1 es $MEM[0 + R1] + MEM[0 + R1]$
En el caso de reemplazar esa secuencia por la instrucción

```
ADD     R1, 0(R1)
```

El resultado en R1 es $R1 + MEM[0 + R1]$, que si en R1 tenemos $MEM[0 + R1]$ daría lo mismo pero es algo que no se puede garantizar.

Ejercicio 2.11 CAAQA 3ed

Calcular el CPI Efectivo para MIPS utilizando la Tabla 1. Asumir que se obtuvieron los siguientes CPI para instrucciones.

Instrucción	Ciclos
ALU	$1,0$
Load/Store	$1,4$
Saltos condicionales	
Tomados	$2,0$
No Tomados	$1,5$
Saltos	$1,2$

Asumir que el 60% de los saltos son tomados.

Ejercicio 2.11 CAAQA 3ed - Resolución

Vamos a llevar el mix de instrucciones de la Tabla 1 a estas cuatro categorías para calcular el CPI. Tenemos que el 53% de las instrucciones son ALU (tardan 1 ciclo en promedio), las instrucciones L/S son el 36 % y tardan 1.4 ciclos, saltos condicionales 16% (depende si se toman o no) y saltos son el 1 % y tardan 1.2 ciclos. Con lo cual podemos calcular el CPI efectivo

$$\overline{CPI} = \frac{\sum_i IC_i \times CPI_i}{IC}$$

$$\overline{CPI} = 0.53 \times 1 + 0.36 \times 1.4 + 0.01 \times 1.2 + \underbrace{0.16 \times (\underbrace{0.6 \times 2}_{tomados} + (1 - 0.6) \times 1.5)}_{saltos}$$

Ejercicio 2.12 CAAQA 3ed

Considerar el agregado de un nuevo modo de acceso a MIPS. El nuevo modo suma dos registros y un valor de offset de 11 bits con signo para obtener la dirección efectiva. Utilizar el porcentaje de instrucciones indicado en la Tabla 1. El compilador pasa de las instrucciones:

```
ADD     R1, R1, R2
LW      Rd, 100(R1)    (or Store)
```

A utilizar:

```
LW    Rd, 100(R1,R2)
```

Ejercicio 2.12 CAAQA 3ed

a. Asumir que el nuevo modo de acceso es usado por el 10% de los *loads* y *stores*. ¿Cuál es el porcentaje de IC nuevo comparado con la tasa original?

b. Si el nuevo modo de direccionamiento aumenta en 5% el tiempo de clock, ¿Cuál computadora será más rápida y por cuanto?

c. Describa una situación que limite el porcentaje de reemplazos realizado.