LICENCIATURA EN INFORMÁTICA
INGENIERÍA EN INFORMÁTICA
Facultad de Ciencias Exactas y Tecnología
Universidad Nacional de Tucumán

ARQUITECTURA Y ORGANIZACIÓN DE COMPUTADORAS II
TRABAJO PRÁCTICO
ISA
RODRIGUEZ DEL BUSTO, SOFIA

Problema 1

Muestre cómo se almacenarán los siguientes valores en máquinas con palabras de 32 bits, utilizando el formato little endian y luego el big endian. Suponga que cada valor comienza en la dirección 1016. Dibuje un diagrama de memoria para cada uno, colocando los valores apropiados en las ubicaciones de memoria correctas (y etiquetadas).

Address ⟶	10_{16}	11_{16}	12_{16}	13_{16}
Big Endian				
Little Endian				

1. $456789A1_{16}$

Address	10_{16}	11_{16}	12_{17}	13_{16}
Big Endian	45	67	89	A1
Little Endian	A1	89	67	45

2. $0000058A_{16}$

Address	10_{16}	11_{16}	12_{17}	13_{16}
Big Endian	8A	05	00	00
Little Endian	00	00	05	8A

3. 14148888_{16}

Address	10_{16}	11_{16}	12_{17}	13_{16}
Big Endian	14	14	88	88
Little Endian	88	88	14	14

Problema 2

Se dispone de una computadora donde se puede observar la siguiente porción de memoria. Si hay un valor entero (32bits) guardador en dirección x00F2A3DE, el valor es positivo o negativo? ¿Qué implica la pregunta previa?, explique su razonamiento.

Dirección	x00F2A3DE	x00F2A3DF	x00F2A3E0	x00F2A3E1	x00F2A3E2	x00F2A3E3	x00F2A3E4	x00F2A3E5
Dato	F2	A4	50	0C	C7	B4	57	FB

Si la palabra en memoria es [F2][34][A7][0C], en Big Endian el byte más significativo es F2, quedando la palabra 0xF234A70C, que en binario es 11110010 00110100 10100111 00001100. Como el bit más alto es 1, el número es negativo y su valor decimal con signo es -227.610.868**.** En cambio, en Little Endian el byte más significativo es 0C, quedando la palabra 0x0CA734F2, que en binario es 00001100 10100111 00110100 11110010. Como el bit más alto es 0, el número es positivo y su valor decimal es 212.163.954.

Problema 3

Convierta las siguientes expresiones de notación infija a notación posfija (Polaca Inversa). Haciendo referencia al modelo de memoria de la MIC-1, figuras 4-8 y 4-9 del libro de Tanenbaum, diagrame como es la secuencia de operaciones de cada expresión.

1. X * Y + W * Z + V * U

VALOR EVALUADO	EXPRESIÓN POSFIJA	PILA
X	X	
*	X	*
Y	X Y	*
+	X Y *	+
W	X Y * W	+
*	X Y * W	+ *
Z	X Y * W Z	+ *
+	X Y * W Z * +	+
V	X Y * W Z * + V	+
*	X Y * W Z * + V	+ *
U	X Y * W Z * + V U	+ *
	X Y * W Z * + V U * +	

2. W * X + W * (U * V + Z)

VALOR EVALUADO	EXPRESIÓN POSFIJA	PILA
W	W	
*	W	*
X	W X	*
+	W X *	+
W	W X * W	+
*	W X * W	+ *
(	W X * W	+ * (
U	W X * W U	+ * (
*	W X * W U	+ * (*
V	W X * W U V	+ * (*
+	W X * W U V *	+ * (+
Z	W X * W U V * Z	
)	W X * W U V * Z +	+ *
	W X * W U V * Z + * +	

3. (W * (X + Y * (U * V))) / (U * (X + Y))

VALOR EVALUADO	EXPRESIÓN POSFIJA	PILA
(		(
W	W	(
*	W	(*
(	W	(* (
X	W X	(* (
+	W X	(* (+
Y	W X Y	(* (+
*	W X Y	(* (+ *
(	W X Y	(* (+ * (
U	W X Y U	(* (+ * (
*	W X Y U	(* (+ * (*

V	W X Y U V	(* (+ * (*
)	W X Y U V *	(* (+ *
)	W X Y U V * * +	(*
)	W X Y U V * * + *	
/	W X Y U V * * + *	/
(	W X Y U V * * + *	/ (
U	W X Y U V * * + * U	/ (
*	W X Y U V * * + * U	/ (*
(	W X Y U V * * + * U	/ (* (
X	W X Y U V * * + * U X	/ (* (
+	W X Y U V * * + * U X	/ (* (+
Y	W X Y U V * * + * U X Y	/ (* (+
)	W X Y U V * * + * U X Y +	/ (*
)	W X Y U V * * + * U X Y + *	/
	W X Y U V * * + * U X Y + * /	

Problema 4

Convierta lo siguiente en notación posfija.

X = X - (Y + 3) + Z + (W - 2)

VALOR EVALUADO	EXPRESIÓN POSFIJA	PILA
X	X	
=	X	=
X	X X	=
-	X X	= -
(	X X	= - (
Y	X X Y	= - (
+	X X X Y	= - (+
3	X X Y 3	= - (+
)	X X Y 3 +	= -
+	X X Y 3 + -	= +
Z	X X Y 3 + - Z	= +
+	X X Y 3 + - Z +	= +
(	X X Y 3 + - Z +	= + (
W	X X Y 3 + - Z + W	= + (
-	X X Y 3 + - Z + W	= + (-
2	X X Y 3 + - Z + W 2	= + (-
)	X X Y 3 + - Z + W 2 -	= +
	X X Y 3 + - Z + W 2 - + =	

Usando la ISA de la IJVM, escriba un programa que evalúe esta expresión. Hagan la suposición que cada variable ya se encuentra en el marco de los variables locales en las siguientes posiciones en referencia al LV:

- X está en posición 1
- Y está en posición 2
- Z está en posición 3
- W está en posición 4

Hint: Van a tener que usar BIPUSH, IADD, ILOAD, ISTORE, e ISUB.

ILOAD 2 // carga Y en la pila

BIPUSH 3 // carga 3 en la pila

IADD // suma Y y 3

ILOAD 1 // carga X en la pila

ISUB // resta (Y+3) de X

ISTORE 1 // almacena resultado temporal en X

ILOAD 1 // carga X en la pila

ILOAD 3 // carga Z en la pila

IADD // suma X y Z

ILOAD 4 // carga W en la pila

BIPUSH 2 // carga 2 en la pila

ISUB // resta 2 de W

IADD // suma (X+Z) y (W-2)

ISTORE 1 // almacena resultado final en X

Problema 5

¿Cuál es la diferencia entre utilizar el modo de direccionamiento directo y el indirecto? Dé un ejemplo.

El direccionamiento directo consiste en que la instrucción contiene explícitamente la dirección efectiva de memoria del operando. De esta forma, el procesador accede directamente a esa posición en memoria para leer o escribir el dato. Es muy útil para variables globales o valores pequeños, ya que permite un acceso rápido sin pasos intermedios. La dirección de memoria queda fija en la instrucción, por lo que si bien el contenido en esa posición puede modificarse las veces que se quiera, la ubicación en memoria de la variable no puede cambiar. Un ejemplo claro es cuando declaramos una variable simple: podemos actualizar su valor, pero la dirección física donde está almacenada permanecerá la misma durante la ejecución del programa.

En el direccionamiento indirecto, en lugar de contener la dirección del operando, la instrucción señala un registro que guarda un puntero hacia la dirección efectiva en memoria. Esto lo hace muy flexible, ya que modificando el valor del registro se puede cambiar fácilmente la ubicación a la que se accede, lo que lo vuelve ideal para trabajar con arreglos o estructuras dinámicas. La gran ventaja es esta capacidad de recorrer o modificar referencias sin tener que cambiar

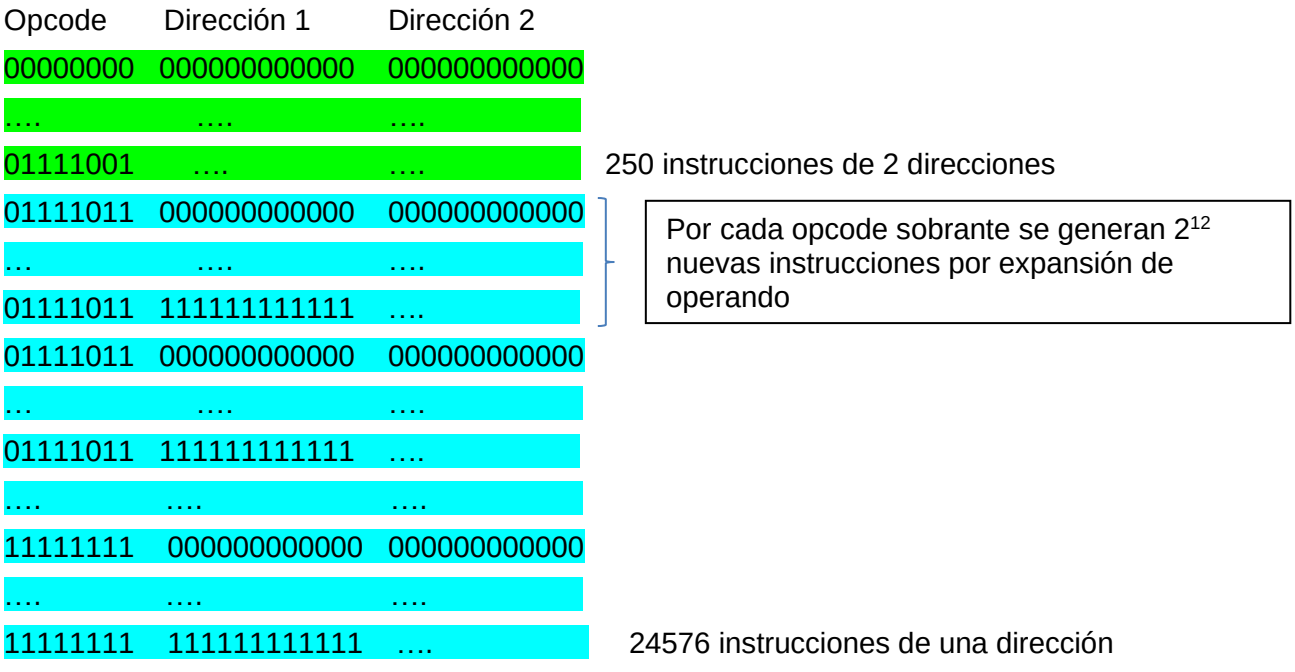
la instrucción, aunque a costa de más accesos a memoria ya que primero se debe leer la dirección desde el registro y luego acceder al dato. Un ejemplo es cuando declaramos un puntero o un arreglo: el registro almacena la dirección base y, al incrementar el puntero, podemos movernos por diferentes posiciones de memoria.

Problema 6

Un ordenador tiene instrucciones de longitud fija de 32 bits donde el tamaño de un operando es de 12 bits. Supongamos que hay 250 instrucciones de 2 direcciones. ¿Si usamos expansión de operando, cuántas instrucciones de 1 dirección se pueden formular? Explica tu respuesta.

Para instrucciones de dos direcciones, sabiendo que el tamaño de un operando es de 12 bits, necesitamos 24 bits para los operandos. Luego, como la instrucción tiene una longitud de 32 bits, nos quedan 31 – 24 = 8 bits para representar los opcodes. Por lo tanto, la cantidad de instrucciones de 2 direcciones que podemos representar es 2^8 = 256 direcciones.

Como existen 250 instrucciones de dos direcciones entonces tenemos 256 – 250 = 6 códigos de 8 bits que podemos utilizar para la expansión de operando para representar instrucciones de una dirección. Como ahora solo se necesita representar una dirección de 12 bits, tenemos 12 bits extras para realizar la expansión. Por lo tanto, por cada opcode no utilizado en las instrucciones de dos direcciones vamos a poder direccionar 2^{12} direcciones de una dirección. Luego, obtenemos que la cantidad de instrucciones de una dirección que podemos formular es igual a $6*2^{12}$ = 24576 instrucciones.



Problema 7

En un formato de instrucción de ordenador, la longitud de la instrucción es de 11 bits y el tamaño de un campo de dirección para un operando es de 4 bits.

1. ¿Con el esquema mencionado, es posible implementar lo siguiente?
    - 5 instrucciones de 2 direcciones
      Como cada dirección se representa con 4 bits, hay 11 bits – 2*4 bits = 3 bits disponibles para los opcodes.
      Por lo tanto, la cantidad de instrucciones que se pueden formular son: $2^3 = 8$ instrucciones.
      Se necesitan 5 instrucciones por lo que nos queda 3 cadenas (opcodes no necesitados) de 3 bits para la expansión de operando.
    - 45 instrucciones de 1 dirección
      Como ahora solo se busca representar 1 dirección, hay 4 bits disponibles para realizar la expansión. Por cada uno de los opcodes no utilizados anteriormente, se utilizarán los 4 bits extras para formular nuevas instrucciones. Por lo tanto, la cantidad de instrucciones de una dirección que se pueden formular son $3*2^4 = 48$ instrucciones.
      Como se necesitan 45 instrucciones de una dirección, hay 3 de ellas disponibles para la nueva expansión.
    - 32 instrucciones de 0 direcciones
      Ahora se necesitarán 0 direcciones por lo que se suman 4 bits para realizar la expansión. Por cada una de las instrucciones de una dirección no utilizadas anteriormente, se utilizarán los 4 bits extras para formular nuevas instrucciones. Por lo tanto, la cantidad de instrucciones de una dirección que se pueden formular son $3*2^4 = 48$ instrucciones.
      Como se necesitan 32 instrucciones podemos afirmar que se puede implementar lo mencionado en este esquema.

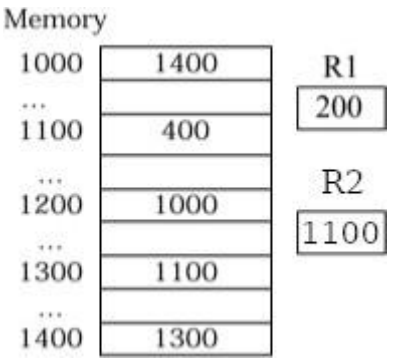
2. Supongamos que un arquitecto de computadoras ya ha diseñado 6 instrucciones de dos direcciones y 24 de dirección cero utilizando el formato de instrucción anterior. ¿Cuál es el número máximo de instrucciones de una dirección que se puede añadir al conjunto?

Sabemos que la cantidad máxima de instrucciones de dos direcciones que se pueden generar son 8. Como se diseñaron 6 instrucciones, hay dos de ellas que se pueden utilizar en la expansión de operando. Por cada una de las anteriores, se utilizarán los 4 bits extras para formular nuevas instrucciones. Por lo tanto, la cantidad de instrucciones de una dirección que se pueden formular son $2*2^4 = 32$ instrucciones.

Sin embargo, este número está limitado por la cantidad de instrucciones de cero direcciones ya que se obtienen mediante la expansión de operando de las instrucciones de una dirección. Como por cada instrucción de una dirección se pueden obtener 2^4 direcciones de 0 direcciones, la cantidad de instrucciones de cero direcciones = cantidad de instrucciones de una dirección libre * 2^4. Es decir, $24 = x*2^4$, $x = 24/16 = 1.5$. Redondeando obtenemos que se necesitan que dos de l total de instrucciones de una dirección queden

libres. Podemos concluir que la cantidad máxima de instrucciones de 1 dirección que se pueden definir es igual a 32 – 2 = 30 instrucciones.

Supongamos que tenemos varias instrucciones de LOAD (Carga) con diferentes modos de direccionamiento. Dada la siguiente porción de memoria y los valores en registros R1 y R2.



Determine el valor real cargado en R3:

Modo	**Instrucción**	**Valor cargado en R3**
Inmediato	LOAD R3, 1400	1400
Directo	LOAD R3, (1100)	400
Indirecto	LOAD R3, (R2)	400
Indexado	LOAD R3, (R1 + R2)	1100
Desplazamiento	LOAD, R3, 1000(R1)	1000

Problema 9

Una computadora tiene una unidad de memoria de 24 bits por palabra. El conjunto de instrucciones consta de 150 operaciones diferentes. Todas las instrucciones tienen una parte de código de operación (opcode) y una parte de dirección (que permite una sola dirección). Cada instrucción se almacena en una palabra de memoria.

1. ¿Cuántos bits es lo mínimo necesarios para el opcode?

   Dada una cantidad de bits n para representar el opcode, podemos representar 2^n opcodes diferentes. Como necesitamos 150 opcodes diferentes, 150 >= 2^n. Luego, n = $\log_2 150$ = 7,22 bits. Redondeando, necesitamos 8 bits para representar el opcode.

2. ¿Cuántos bits quedarían para el operando?
   La cantidad de bits que quedan para el operando es 24 bits – 8 bits = 16 bits.
3. ¿Cuál es la dirección máxima de memoria?

La dirección máxima de memoria que se puede direccionar con 16 bits es $2^{16} - 1 = 65535$.

Problema 10

Investiguen la arquitectura MIPS (32 Release 5) y detallen los siguientes conceptos:

- **El modelo de memoria.**

El MIPS32 utiliza un modelo de memoria de espacio de direcciones planas y se basa en la arquitectura Load-Store (Cargar-Almacenar).

- **Espacio de Direcciones**: La memoria principal se organiza como un único espacio de direcciones virtuales de 4 Gigabytes (232bytes), ya que la arquitectura es de 32 bits.
- **Orientación por Bytes**: La memoria está direccionada por bytes (cada byte tiene una dirección única), aunque las instrucciones suelen manipular datos de 32 bits (palabras).
- **Load-Store**: La CPU no puede manipular datos directamente en la memoria. Las únicas instrucciones que acceden a la memoria son las de carga **(Load),** para traer datos de la memoria a un registro, y almacenamiento **(Store),** para escribir datos de un registro a la memoria. Todas las demás operaciones (aritméticas, lógicas, etc.) se realizan solo con datos ubicados en los registros de la CPU.
- **Alineación de datos**: Los datos (palabras de 32 bits, medias palabras de 16 bits, etc.) deben estar alineados en la memoria según su tamaño. Por ejemplo, una palabra (4 bytes) debe comenzar en una dirección divisible por 4.
- **Endianness (Orden de Bytes)**: MIPS puede implementarse como Big-Endian (el byte más significativo se almacena en la dirección más baja) o Little-Endian (el byte menos significativo se almacena en la dirección más baja), siendo la elección una característica de la implementación.

- **Los tipos de datos.**

El MIPS32 funciona principalmente con los siguientes tipos de datos, todos con un tamaño base de 32 bits:

- **Enteros:**
  - Byte (8 bits): Usado para caracteres.
  - Media Palabra (Halfword, 16 bits): Enteros cortos.
  - Palabra (Word, 32 bits): Es el tamaño estándar del registro y el más común para todas las operaciones.
- **Punto Flotante:** La arquitectura MIPS incluye un Coprocesador 1 (COP1) para operaciones de punto flotante (FPU).
  - Precisión Simple (Single-Precision, 32 bits): Estándar IEEE 754.
  - Precisión Doble (Double-Precision, 64 bits): Estándar IEEE 754 (utiliza un par de registros de punto flotante).

- **Los modos de direccionamiento usados.**

La arquitectura MIPS se caracteriza por su reducido y simplificado conjunto de modos de direccionamiento para facilitar la segmentación (pipelining).

- **Direccionamiento por Registro (Register):** El operando se encuentra directamente en uno de los 32 registros de propósito general.
- **Direccionamiento Inmediato(Immediate):** El valor del operando es una constante de 16 bits que está incrustada directamente en la instrucción.
- **Direccionamiento Base/Desplazamiento:** Es el único modo de direccionamiento de memoria. La dirección efectiva se calcula sumando el contenido de un registro base (un registro de propósito general) y una constante de desplazamiento (offset) de 16 bits con signo incrustado en la instrucción.
- **Direccionamiento Relativo al PC (PC-Relative):** Utilizado para instrucciones de bifurcación condicional (beq, bne). La dirección de destino del salto se calcula a partir del valor del Program Counter (PC) sumado a un desplazamiento incrustado en la instrucción (que es de 16 bits, pero se extiende para direccionar instrucciones de 32 bits).
- **Direccionamiento Pseudo-Directo:** Utilizado para las instrucciones de salto incondicional (j, jal). La dirección de destino se construye combinando los 26 bits del campo de dirección de la instrucción con los bits superiores del Program Counter (PC).

- **Lista de los tipos de operaciones que se pueden hacer.**

El conjunto de instrucciones de MIPS32 se clasifica ampliamente en las siguientes categorías:

- **Aritméticas:**
  - Suma (add, addi), Resta (sub).
  - Multiplicación (mult, mul), División (div).
- **Lógicas:**
  - Y (and, andi), O (or, ori), XOR (xor, xori), NOR (nor).
- **Transferencia de Datos (Memoria):**
  - Cargar Palabra (lw), Media Palabra (lh), Byte (lb).
  - Almacenar Palabra (sw), Media Palabra (sh), Byte (sb).
  - Carga Inmediato Superior (lui- Load Upper Immediate) para construir constantes de 32 bits.
- **Desplazamiento (Shift):**
  - Desplazamiento Lógico Izquierda (sll), Desplazamiento Lógico Derecha (srl).
  - Desplazamiento Aritmético Derecha (sra).
- **Operaciones de punto flotante**: add.s, sub.s, mul.s, div.s, y operaciones de conversión entre enteros y punto flotante.

- **Control de Flujo**

El control de flujo en MIPS se realiza mediante **instrucciones de salto y de rama condicional**:

- **Saltos incondicionales**: Realice un salto si se cumple una condición (comparando el contenido de los registros). Utiliza Dirección Relativa al PC .
  - j target: Salta a una dirección específica.
  - jr $ra: Salta a la dirección almacenada en un registro (comúnmente usado en retorno de subrutinas).

- **Saltos condicionales**: El salto se ejecuta siempre. Utiliza Dirección Pseudo-Directo .
  - beq $rs, $rt, label: Si los registros $rs y $rt son iguales, salta a la etiqueta.
  - bne $rs, $rt, label: Si los registros $rs y $rt no son iguales, salta a la etiqueta.

- **Subrutinas**:
  - jal label: Llama a una subrutina y guarda la dirección de retorno en $ra.

- **Tipo de operación**

  Las operaciones se dividen en los siguientes tipos:
  - **Instrucciones aritméticas**: add, sub, mul, div, etc.
  - **Operaciones de carga y almacenamiento**: lw, sw, lb, sb.
  - **Operaciones lógicas y de bits**: and, or, xor, nor, sll, srl.
  - **Operaciones de control de flujo**: j, jal, beq, bne, jr.
  - **Operaciones de punto flotante**: add.s, sub.s, mul.s, div.s, y operaciones de conversión entre enteros y punto flotante.

- **Formato de instrucción**

  Todas las instrucciones MIPS32 tienen un tamaño fijo de 32 bits . Existen tres formatos principales, definidos por el campo de Opcode (bits 31-26).

  - **Formato R (Registro)**: Utilizado para operaciones aritméticas y lógicas que requieren tres registros.
    - Ejemplo: add $rd, $rs, $rt
    - Campos: opcode (6 bits), rs (5 bits), rt (5 bits), rd (5 bits), shamt (5 bits), funct (6 bits).

  - **Formato I (Inmediato)**: Utilizado para operaciones con valores inmediatos o accesos a memoria.
    - Ejemplo: addi $rt, $rs, immediate
    - Campos: opcode (6 bits), rs (5 bits), rt (5 bits), immediate (16 bits).

  - **Formato J (Salto)**: Utilizado para las instrucciones de salto.

- Ejemplo: j target
- Campos: opcode (6 bits), address (26 bits).