**Ejercicio 1**

El computador A utiliza ordenación **little-endian**. Dispone de registros de propósito general de 64-bits (x1, x2, x3....). Para leer datos de la memoria se usa **direccionamiento directo**. Estas son las instrucciones de carga (load) disponibles:

* load\_byte reg, dir : Carga del **byte** almacenado en la dirección dir en el registro reg
* load\_half reg, dir : Carga de la **media palabra** (16-bits) almacenada en la dirección dir en el registro reg
* load\_word reg, dir : Carga de la **palabra** (32-bits) almacenada en la dirección dir en el regisgtro reg
* load\_dword reg, dir : Carga de la **doble palabra** (64-bits) almacenada en la dirección dir en el registro reg

Se ha realizado un **volcado de la memoria** (en bytes) a partir de la dirección **0x10010000**. Esto es lo obtenido:

Tabla

Descripción generada automáticamente con confianza media

Inidicar el valor (en hexadecimal) del dato que se almacena en el registro x1 al ejecutar las siguientes instrucciones:

* a) load\_byte x1, 0x10010003

**0x1D**

* b) load\_byte x1, 0x10010005

**0XCA**

* c) load\_half x1, 0x10010003

**0X001D**

* d) load\_half x1, 0x10010005

**0X00BACA**

* e) load\_word x1, 0x10010002

**0XCA001D2C**

* f) load\_word x1, 0x10010005

**0XDEFEBACA**

* g) load\_dword x1, 0x10010001

**0XDEFEBACA001D2C3B**

**Ejercicio 2**

El **computador B** es igual que el computador A del ejercicio 1, pero su ordenación es de tipo **Big Endian**. Si el volcado de memoria es el mismo que el del ejercicio 1, inidicar el valor (en hexadecimal) del dato que se almacena en el registro x1 al ejecutar las mismas instrucciones de los apartados a-g del ejercicio 1

* a) load\_byte x1, 0x10010003

**0x1D**

* b) load\_byte x1, 0x10010005

**0XCA**

* c) load\_half x1, 0x10010003

**0X1D00**

* d) load\_half x1, 0x10010005

**0X00CABA**

* e) load\_word x1, 0x10010002

**0X2C1D00CA**

* f) load\_word x1, 0x10010005

**0XCABAFEDE**

* g) load\_dword x1, 0x10010001

**0X3B2C1D00CABAFEDE**

**Ejercicio 3**

El **computador C** tiene una **arquitectura de 32-bits**. La memoria que usa tiene por tanto una **anchura de 32-bits**, lo que le permite leer o escribir palabras de 32-bits con un **único acceso**. Dispone de instrucciones de load y store para acceder a datos de tipo **byte**, **media palabra** y **palabra**. Estas instrucciones usan **direccionamiento directo**. Las **instrucciones de load** son las siguientes:load\_byte reg, dir : Carga del **byte** almacenado en la dirección dir en el registro reg

* load\_half reg, dir : Carga de la **media palabra** (16-bits) almacenada en la dirección dir en el registro reg
* load\_word reg, dir : Carga de la **palabra** (32-bits) almacenada en la dirección dir en el regisgtro reg

El computador C **permite acceso** tanto a **datos alineados** como **no alineados**. Indica la **cantidad de accesos a memoria** para la **lectura de los datos** que se realizan al ejecutarse estas instrucciones

* a) load\_byte x1, 0x2003

1 acceso a memoria

* b) load\_byte x1, 0x2000

1 acceso a memoria

* c) load\_half x1, 0x2000

Es direccion alineada porque esta en una dirección con numeor par, asi que 1 acceso.

* d) load\_half x1, 0x2007

Es dirección no alineada. Dos accesos.

* e) load\_half x1, 0x200A

Es direccion alineada porque esta en una dirección con numeor par, asi que 1 acceso.

* f) load\_word x1, 0x2000

ES DIRECCION ALINEADA 1 ACCESO

* g) load\_word x1, 0x2004

Es dirección alineada, un acceso

* h) load\_word x1, 0x2007

Es dirección no alineada, no es multiplo de 4. Dos accesos

* i) load\_word x1, 0x200A

Es dirección no alineada, no es multiplo de 4. Dos accesos

**Ejercicio 4**

El **computador D** es de **32 bits**. La **anchura** de su memoria es de **32-bits**. Utiliza ordenación **Little-endian** y puede acceder a datos situados en direcciones **alineadas** y **no alineadas**. Utiliza **direccionmiento directo** para el acceso a memoria. Estas son las **instrucciones de almacenamiento** (store) disponibles:

* store\_byte reg, dir : Almacenar el byte del registro reg en la dirección dir
* store\_half reg, dir : Almacenar la media palabra (16-bits) del registro reg en la dirección dir
* store\_word reg, dir : Almacenar la **palabra** (32-bits) del registro reg en la dirección dir

Inicialmente, los registros del procesador contienen estos valores:

| **Registro** | **Valor** |
| --- | --- |
| x1 | 0xFF |
| x2 | 0x00 |
| x3 | 0xCAFE |
| x4 | 0xBEBA |
| x5 | 0xD0D1 |
| x6 | 0xFEDEDABA |
| x7 | 0x01020304 |
| x8 | 0xABCDEF12 |

Tras ejecutarse este programa:

store\_byte x1, 0xBA05

store\_byte x1, 0xBA1F

store\_byte x2, 0xBA15

store\_half x3, 0xBA07

store\_half x4, 0xBA0A

store\_half x5, 0xBA1D

store\_word x6, 0xBA01

store\_word x7, 0xBA0F

store\_word x8, 0xBA18

Indicar:

* a) Los valores que se han almacenado en cada posición de memoria (Rellena el dibujo)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|  | BA | DA | DE | FE | FF |  | FE | CA |  | BA | BE |  |  |  | 04 |
| 03 | 02 | 01 |  |  | 00 |  |  | 12 | EF | CD | AB |  | D1 | D0 | FF |

Gráfico

Descripción generada automáticamente con confianza media

* b) ¿Cuantos accesos de escritura a memoria se han realizado en total?

|  |  |  |
| --- | --- | --- |
|  | ALINEADA | LECTURAS |
| store\_byte x1, 0xBA05 | **si** | **1** |
| store\_byte x1, 0xBA1F | **si** | **1** |
| store\_byte x2, 0xBA10 | **si** | **1** |
| store\_half x3, 0xBA07 | **no** | **2** |
| store\_half x4, 0xBA0A | **si** | **1** |
| store\_half x5, 0xBA1D | **no** | **2** |
| store\_word x6, 0xBA01 | **no** | **2** |
| store\_word x7, 0xBA0F | **no** | **2** |
|  |  | **13** |

**Ejercicio 5**

En un computador con arquitectura base **RV64I** (Risc-V de 64 bits) se ha definido un array de **dobles palabras** en un lenguaje de alto nivel. Implementar en ensamblador de ese computador las instrucciones que permite calculas las siguientes **expresiones**. Supón que la **dirección base** del **array A** se encuentra en el **registro s0**, y la **variable h** en el **registro s1**

* a) a = h + A[8]. Utiliza el **registro t0** para la variable **a**
* b) A[12] = h + A[8]. El resultado de h + A[8] se almacena en el propio array A, en el índice 12

**Ejercicio 6**

En un computador con arquitectura base **RV64IM** (Risc-V de 64 bits) se han definido un **array A** en un lenguaje de alto nivel. Se quiere implementar la siguiente expresión

A[12] = h + A[i]

Supón que la **dirección base** del **array A** se encuentra en el **registro s3**, la **variable h** en el **registro s2** y la **variable i** en el **registro s1**

Implementar en **ensamblador** de ese computador las instrucciones que permite calcular esa expresión para los siguientes **tipos de datos** almacenados en el array:

a) Los elementos del array son dobles palabras  
b) Los elementos del array son palabras  
c) Los elementos del array son medias palabras  
d) Los elementos del array son bytes

**Ejercicio 7**

En un computador con arquitectura base **RV64IM** (Risc-V de 64 bits con la extensión de multiplicación y división de enteros) se han definido dos Arrays: A y B en un lenguaje de alto nivel. Se quiere implementar la siguiente expresión

B[12] = h - A[i]

Supón que la **dirección base** del **array B** se encuentra en el **registro s4**, la dirección base del **array A** en el **registro s3**, la **variable h** en el **registro s2** y la **variable i** en el **registro s1**

Implementar en **ensamblador** de ese computador las instrucciones que permite calcular esa expresión para los siguientes **tipos de datos** almacenados en el array:

a) Los elementos del array son dobles palabras  
b) Los elementos del array son palabras  
c) Los elementos del array son medias palabras  
d) Los elementos del array son bytes