**GUÍA DEL ALUMNO**

***SESIÓN 1: Introducción al DLX***

# Recursos

* Documentos “**Repertorio ESENCIAL del DLX”** y “**Mitos y Leyendas: variables”**

Diapositivas que acompañan las explicaciones de la profesora.

* El procesador DLX se encuentra descrito con detalle en:

o Arquitectura de Computadores. un enfoque cuantitativo" de Hennessy- Patterson, Mc Graw Hill, 1ª Edición 1993. Capítulos 5 y 6, Págs 172 (repertorio), y 273 en adelante (características del cauce), cuya lectura es ampliamente recomendada.

* DLXVSim para Windows: un ensamblador y simulador del procesador DLX. Disponible en la máquina virtual que utilizamos. Si no está instalado previamente, simplemente descomprime DLXV3.1.zip en una carpeta y ejecuta el programa de instalación dejándolo en c:\DLXV que suele ser la ubicación por defecto. Es útil definir un acceso directo desde el Escritorio. El "**HELP**" que incluye la instalación detalla las directivas y demás aspectos de la herramienta y el ensamblador.

# Procedimiento

En esta primera sesión nos familiarizaremos con el Repertorio de instrucciones del DLX, el procesador con el que estudiaremos algunos aspectos de la segmentación de cauce. Para trabajar un poco con el DLX, utilizaremos una herramienta de compilación (ensamblado) y simulación llamada DLXVSIM. Describiremos sus funciones básicas y la notación, directivas y sintaxis del lenguaje ensamblador del DLX. Más adelante utilizaremos una segunda herramienta para analizar otros aspectos de la segmentación y sus riesgos.

# Programa ejemplo

Este ejemplo muestra la suma de los n (10) primeros números enteros. Escríbelo y guárdalo en un fichero de nombre **sumn.s** La extensión "s" identifica a una fuente en ensamblador.

|  |
| --- |
| **.data 100 n: .word 10 suma: .space 4**  **.text 1000 ini: lw r7,n(r0) ; esto es UN COMENTARIO r7 <-- M(r0+n)=M(0+100)=10 r7 tendrá un 10 xor r1,r1,r1**  **loop: add r1,r1,r7 subir7,r7,1 bnezr7,loop**  **nop sw suma(r0),r1 trap #6** |

**.data** y **.text** son directivas del ensamblador para ubicar en esas direcciones a los segmentos de datos (variables) y texto (programa). Ver la AYUDA de la herramienta.

**.word** y **.space** también son directivas del ensamblador, como **.float** o **.double**:

* **.word** se usa para asignar un nombre lógico (**n**) a la dirección 100 (en este caso ya que es la primera variable del segmento de datos) donde se ha reservado un espacio de una palabra con el valor 10.
* **.space** se usa para reservar bytes. El el ejemplo, 4 bytes a partir de la dirección suma, que en este caso es 104 ya que una palabra tiene 4 bytes.

**ini:** y **loop:** son etiquetas que nos hacen más cómodo la referencia a las direcciones 1000 y 1008 (las instrucciones miden todas 4 bytes en esta máquina).

Punto y coma denota **comentario**.

**trap #6** hace que se detenga la ejecución pero que se completen las instrucciones que entraron al cauce. El resto es el repertorio propiamente dicho debes asegurarte de interpretarlo y entenderlo. Usaremos este programa para demostrar el uso del DLXVSim.

# DLXVSIM

La profesora describirá la herramienta. Al ejecutar paso a paso DLXVSim ejecuta primero una instrucción y para antes de ejecutar la siguiente mostrando esta última. En consecuencia la primera instrucción del programa **no se ve**.

Una vez instalado, crea con el editor (UltraEdit) un programa, es decir, un fichero de texto con la extensión ".s" en el que escribirás el código del programa ejemplo proporcionado. Asegúrate de que puedes:

* Ensamblarlo en DLXVSim,
* Ejecutarlo paso a paso,
* Examinar los registros,
* Examinar la memoria de datos (comprobar el resultado escrito en memoria)
* Ver las estadísticas.

1. Ver el documento “**Repertorio ESENCIAL del DLX”** y fijarse en los **trucos** proporcionados al final. Leer con atención este documento y también “**Mitos y Leyendas: variables”.**
2. Modifica el programa para que realice la suma de **los n primeros números PARES** (no hasta ***n*** si no los ***n*** primeros, por ej. los 9 primeros pares son 2 4 6 8 10 12 14 16 y 18 y no hasta 9 que solo llegaría hasta 8). Deberás usar la instrucción de desplazamiento SLLI (consulta la documentación).
3. EN CASA: Modificarlo de nuevo para que haga la suma de los impares.
4. Implementa el siguiente algoritmo, ejecútalo y examina el resultado. Definir un área de datos que comience en la dirección 100 que tenga un entero “nterm” de valor 8. Reservar a continuación espacio para 8 enteros etiquetando “fibo” a la dirección 104. Desarrollar el programa a partir de la dirección 2000. Ejecutarlo mostrando en la ventana de datos toda la zona de datos hasta 2 enteros más allá del último que escriba el programa.

|  |  |  |
| --- | --- | --- |
| **R1 <-- R0+R0**  **R2<-- R0+1**  **R3<-- R0+1**  **R4<-- M(R0+nterm)**  **R5<-- R5 xor R5** | **bucle:**  **R3<-- R1+R2**  **M(fib+R5)<-- R3**  **R1<-- R2**  **R2<-- R3** | **R5<-- R5+4 R4<-- R4-1**  **if (R4 != 0) GOTO bucle** |

1. Definir un área de datos con dos listas de enteros A y B de 12 números (N =12 debe ser otra variable) y hacer un programa que los sume y guarde en otra lista C. Para dicha lista C debe reservarse sitio en el área de datos con”.space”.

Usar dos registro: uno para contador del bucle y otro como índice, como se usaron R4 y R5 en el programa del ejercicio anterior. Desarrolla dos variantes del programa:

* 1. La primera que recorra los elementos en orden creciente (reg. índice 0,4,8,..4(N-1))
  2. La segunda que recorra los elementos en orden decreciente (reg. índice 4(N-1), 4(N-2), … 4,0)

Ejecutarlo paso a paso observando cómo se producen los resultados en el área de datos y las estadísticas de ciclos consumidos (ventana Estadísticas)

1. Reescribe el siguiente algoritmo con el repertorio del DLX dando los valores a las variables de entrada que desees (por ejemplo, listas de n=6 elementos):

**k=5;**

**for (i=0;i<n;i++)**

**{ c[i] = k + b[i]; k = k + a[i];**

**}**

Necesitarás Registros para:

* CONTADOR DE BUCLE(cargar con número de elementos, normalmente variable enmemoria)
* ÍNDICE de listas para acceso a memoria. Valores crecientes desde 0 o decrecientes(incr./decr. ±4)
* ACUMULADOR para k
* Cargar a[i] y b[i] y para operar y luego guardar resultados en c[i]

1. Escribe y ejecuta el siguiente programa cambiando el valor de n (declararlo). Observa el valor de los registros. Responde las preguntas al lado para cada valor de n utilizado, aprendiendo a contabilizar los ciclos de ejecución.

Con n = 12

|  |  |  |  |
| --- | --- | --- | --- |
| **lw**  **addi r1,r0,n sub r**  **loop:**  **subi add nop add bnez nop nop trap #6** | **r7,n(r0)**  **2,r2,r2**  **r7,r7,1 r3,r1,r2**  **r4,r1,r2 r7,loop** | a)  b)  c)  d)  e)  f) | Instrucciones ejecutadas antes del bucle 3  Instrucciones ejecutadas en el bucle 72  Instrucciones ejecutadas después del bucle 2  Total isnt. Ejecutadas 77 Total ciclos 77  CPI= 1  Compara con los ciclos en Estadísticas (DLXVSim)\_\_\_\_ |

1. Escribe y ejecuta el siguiente programa cambiando el valor de n y a(i). Definir valores 0 y diferentes de 0 para a. Declarar las variables necesarias. Observa el valor de los registros.

Responde las preguntas al lado para cada valor de n utilizado

|  |  |  |
| --- | --- | --- |
| **lw r7,n(r0) xor r2,r2,r2 xor r3,r3,r3**  **loop:**  **lw r1,a(r2) subi r7,r7,1 addi r2,r2,4**  **beqz r1,xxx nop**  **add r3,r3,r1 xxx:**  **bnez r7,loop nop**  **sw nzval(r0),r3 trap #6** | a)  b)  c)  d)  e)  f)  g)  h)  i) | ¿Qué hace el programa?  N.º de valores 0 en a \_\_\_  N.º de valores distintos de 0 en a \_\_\_  Instrucciones ejecutadas antes del bucle \_\_\_\_\_\_  Instrucciones ejecutadas en el bucle \_\_\_\_\_  Instrucciones ejecutadas después del bucle  Total isnt. ejecutadas\_\_\_\_\_\_\_\_\_ Total ciclos\_\_\_\_\_\_  CPI= \_\_\_\_\_\_  Compara con los ciclos en Estadísticas (DLXVSim)\_\_\_\_ |

1. El programa hace un sumatorio de los elementos de a.
2. 1 (El primero)
3. 11
4. 3
5. 95
6. 2
7. 100 100
8. 1

La primera instrucción de salto implementa in “if” condicional (Bifurcación) mientras que la segunda implementa un bucle ¿Por qué? ¿Cuál es la diferencia entre ambas?

La primera bifurcación compara el valor de r1 (a(0)), y si el primer elemento de la lista a es un 0, no suma los valores de a(i), ahorrándose 1 instrucción de ejecución. La diferencia entre ellas es la lógica de implementación en el programa, la primera salta hacia adelante (condición) y la segunda vuelve a un punto anterior del programa (normalmente bucle).