



Prácticas Arquitectura de Computadores



Bloque I

Cronograma

- ▶ Bloque I
 - ▶ Programación DLX nivel básico
 - ▶ Repertorio de Instrucciones DLX
 - ▶ Programa Windlx
 - ▶ Ejemplos básicos
 - ▶ Guiones
-
- ▶ Programación DLX nivel básico
 - ▶ Cargas y Almacenamientos
 - ▶ Operaciones ALU
 - ▶ Lazos
 - ▶ Tiempo de ejecución
 - ▶ **Prueba de validación: 9/10/2017 y 10/10/2017**

Pruebas de Validación

- ▶ **Objetivo: Validar conocimientos visto anteriormente**

- ▶ **Individuales**

- ▶ **Validaciones – Solamente durante el cuatrimestre**

- 1. Juego de Instrucciones DLX / Procesador Secuencial: 0.5p
 - 2. Procesador Segmentado DLX y Ganancia: 1.25p
 - 3. Procesador Superscalar / SuperDLX y Ganancia: 1.25p
 - 4. Optimización de Código y Ganancia: 1p

- ▶ **Presentación Trabajo: 1 punto**

- Convocatoria ordinaria - extraordinaria

Pruebas de Validación

- ▶ Procedimiento de validaciones
 - ▶ Describir prueba: 10 minutos
 - ▶ Desarrollar prueba: 80 minutos
 - Realizar un programa y resolver determinadas cuestiones
 - Subir material a ILIAS
 - ▶ Validar prueba: 30 minutos
 - Chequear programa y cuestiones
 - ▶ Material
 - Juego de Instrucciones
 - Manual del procesador
 - No se permiten código de ejercicios

Introducción

- ▶ **Bloque I – Programación DLX nivel básico**
 - ▶ Conocimientos teóricos de todo el bloque
 - ▶ Repertorio de Instrucciones DLX
 - ▶ Ciclos de ejecución en computador secuencial
 - ▶ Programa Windlx
 - ▶ Material
 - ▶ Manual de la UNED
 - ▶ Programa Windlx

Bloque 1 – Programación DLX nivel básico

Repertorio de Instrucciones

Introducción

- ▶ **Arquitectura orientada a registro (de 32 bits):**
 - ▶ 32 registros de propósito general (GPR): R0, R1, ...
 - ▶ R0 contiene siempre el valor 0.
 - ▶ 32 registros de Punto Flotantes (FPR): F0, F1, ... (Simple precisión)
 - ▶ Utilizables como registros de 64 bits por pares:
 - F0, F2, ... (Doble precisión)
 - ▶ 16 Registros especiales
- ▶ **Tipos de datos (en bits):**

▶ 8: Byte	-128 ... +255
▶ 16: Media palabra	-32768 + 65535
▶ 32: Palabra	-2147483648 + 2147483647
▶ 64: Doble palabra	

Registros Especiales

- ▶ **FPSR** (Floating-Point Status Register).
 - ▶ Es un registro de estado de 1 bit de longitud
 - ▶ Comparaciones y excepciones de coma flotante.
 - ▶ Todos los movimientos desde y hacia este registro se realizan a través de los registros de propósito general.
 - ▶ Las comparaciones en punto flotante asignan el bit de este registro
 - ▶ Disponibles instrucciones de salto que basan su resultado en el valor del bit (1 cierto, 0 falso).
- ▶ **PC** (Program Counter).
 - ▶ Contiene la dirección de la próxima instrucción que va a ser ejecutada.
 - ▶ Los saltos y las bifurcaciones pueden cambiar el contenido del mismo.
- ▶ **IMAR** (Instruction Memory Address Register).
 - ▶ Este registro es inicializado con el contenido del contador de programa en la etapa IF a causa de que está conectado con el sistema de memoria, mientras que el PC no.
- ▶ **IR** (Instruction Register).
 - ▶ En la etapa IF es cargado con la próxima instrucción a ejecutarse.
- ▶ **A, B.**
 - ▶ Son cargados en la etapa ID y sus valores son enviados a los operandos de la unidad aritmético lógica en la siguiente etapa la EX. En WinDLX, además existen los pseudo-registros
 - ▶ AHI y BHI que contienen los 32 bits superiores para valores en coma flotante de doble precisión.

Registros Especiales

- ▶ **BTA (*Branch Target Address*).**
 - ▶ En la etapa ID, la dirección de salto/bifurcación es calculada y escrita en este registro
- ▶ **ALU (*Arithmetic Logical Unit*).**
 - ▶ El resultado de una operación en la ALU es transferido a este registro.
 - ▶ En WinDLX existe un pseudo-registro llamado ALUHI que contiene los 32 bits superiores para valores en coma flotante de doble precisión.
- ▶ **DMAR (*Data Memory Address Register*).**
 - ▶ La dirección de memoria a la que se va acceder es transferida a este registro en la etapa EX.
 - ▶ En la etapa MEM, el acceso a la memoria para lectura o escritura es efectuado con el valor almacenado en este registro.
- ▶ **SDR (*Store Data Register*).**
 - ▶ El dato que se va a escribir en memoria por medio de una instrucción es almacenado previamente en este registro.
 - ▶ En WinDLX existe un pseudo-registro llamado SDRHI que contiene los 32 bits superiores para valores en coma flotante de doble precisión.
- ▶ **LDR (*Load Data Register*).**
 - ▶ El dato que es leído de memoria se almacena en este registro.
 - ▶ En WinDLX existe un pseudo-registro llamado LDRHI que contiene los 32 bits superiores para valores en coma flotante de doble precisión.

Introducción

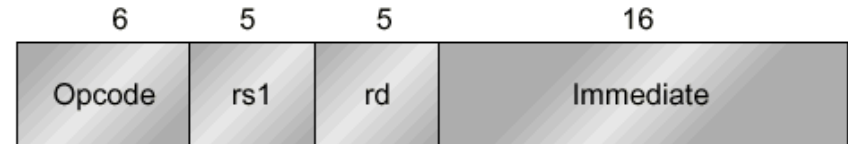
- ▶ Registros especiales: PC, IR
- ▶ Memoria direccionable al byte.
- ▶ Ancho de memoria: 32 bits (bus de datos).
- ▶ Bus de direcciones: 32 bits
- ▶ Organización Big Endian.
- ▶ Modos de direccionamiento:
 - ▶ Registro Ej.: `ADD R1,R2,R3`
 - ▶ Inmediato Ej.: `SUBI R1,R2,#5`
 - ▶ Desplazamiento Ej.: `LW R1,30(R2)`

Formato de la Instrucciones DLX

▶ 3 formatos de instrucciones

- ▶ Igual longitud
- ▶ Igual tamaño de opcode

I - type instruction



Encodes: Loads and stores of bytes, words, half words
All immediates ($rd \leftarrow rs1 \text{ op immediate}$)

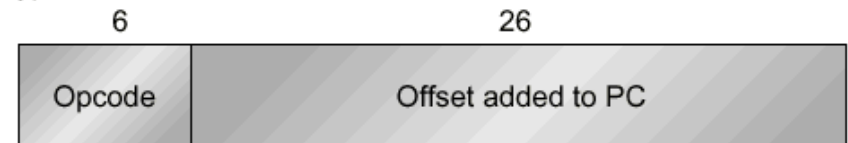
Conditional branch instructions (rs1 is register, rd unused)
Jump register, jump and link register
($rd = 0$, rs = destination, immediate = 0)

R - type instruction



Register-register ALU operations: $rd \leftarrow rs1 \text{ func } rs2$
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

Operaciones de acceso a memoria en el DLX

Example instruction	Instruction name	Meaning
LW R1, 30 (R2)	Load word	$\text{Regs}[\text{R1}] \leftarrow_{32} \text{Mem}[30 + \text{Regs}[\text{R2}]]$
LW R1, 1000 (R0)	Load word	$\text{Regs}[\text{R1}] \leftarrow_{32} \text{Mem}[1000 + 0]$
LB R1, 40 (R3)	Load byte	$\text{Regs}[\text{R1}] \leftarrow_{32} (\text{Mem}[40 + \text{Regs}[\text{R3}]]_0)^{24} \text{##} \text{Mem}[40 + \text{Regs}[\text{R3}]]$
LBU R1, 40 (R3)	Load byte unsigned	$\text{Regs}[\text{R1}] \leftarrow_{32} 0^{24} \text{##} \text{Mem}[40 + \text{Regs}[\text{R3}]]$
LH R1, 40 (R3)	Load half word	$\text{Regs}[\text{R1}] \leftarrow_{32} (\text{Mem}[40 + \text{Regs}[\text{R3}]]_0)^{16} \text{##} \text{Mem}[40 + \text{Regs}[\text{R3}]] \text{##} \text{Mem}[41 + \text{Regs}[\text{R3}]]$
LF F0, 50 (R3)	Load float	$\text{Regs}[\text{F0}] \leftarrow_{32} \text{Mem}[50 + \text{Regs}[\text{R3}]]$
LD F0, 50 (R2)	Load double	$\text{Regs}[\text{F0}] \text{##} \text{Regs}[\text{F1}] \leftarrow_{64} \text{Mem}[50 + \text{Regs}[\text{R2}]]$
SW 500 (R4), R3	Store word	$\text{Mem}[500 + \text{Regs}[\text{R4}]] \leftarrow_{32} \text{Regs}[\text{R3}]$
SF 40 (R3), F0	Store float	$\text{Mem}[40 + \text{Regs}[\text{R3}]] \leftarrow_{32} \text{Regs}[\text{F0}]$
SD 40 (R3), F0	Store double	$\text{Mem}[40 + \text{Regs}[\text{R3}]] \leftarrow_{32} \text{Regs}[\text{F0}] ;$ $\text{Mem}[44 + \text{Regs}[\text{R3}]] \leftarrow_{32} \text{Regs}[\text{F1}]$
SH 502 (R2), R3	Store half	$\text{Mem}[502 + \text{Regs}[\text{R2}]] \leftarrow_{16} \text{Regs}[\text{R3}]_{16..31}$
SB 41 (R3), R2	Store byte	$\text{Mem}[41 + \text{Regs}[\text{R3}]] \leftarrow_8 \text{Regs}[\text{R2}]_{24..31}$

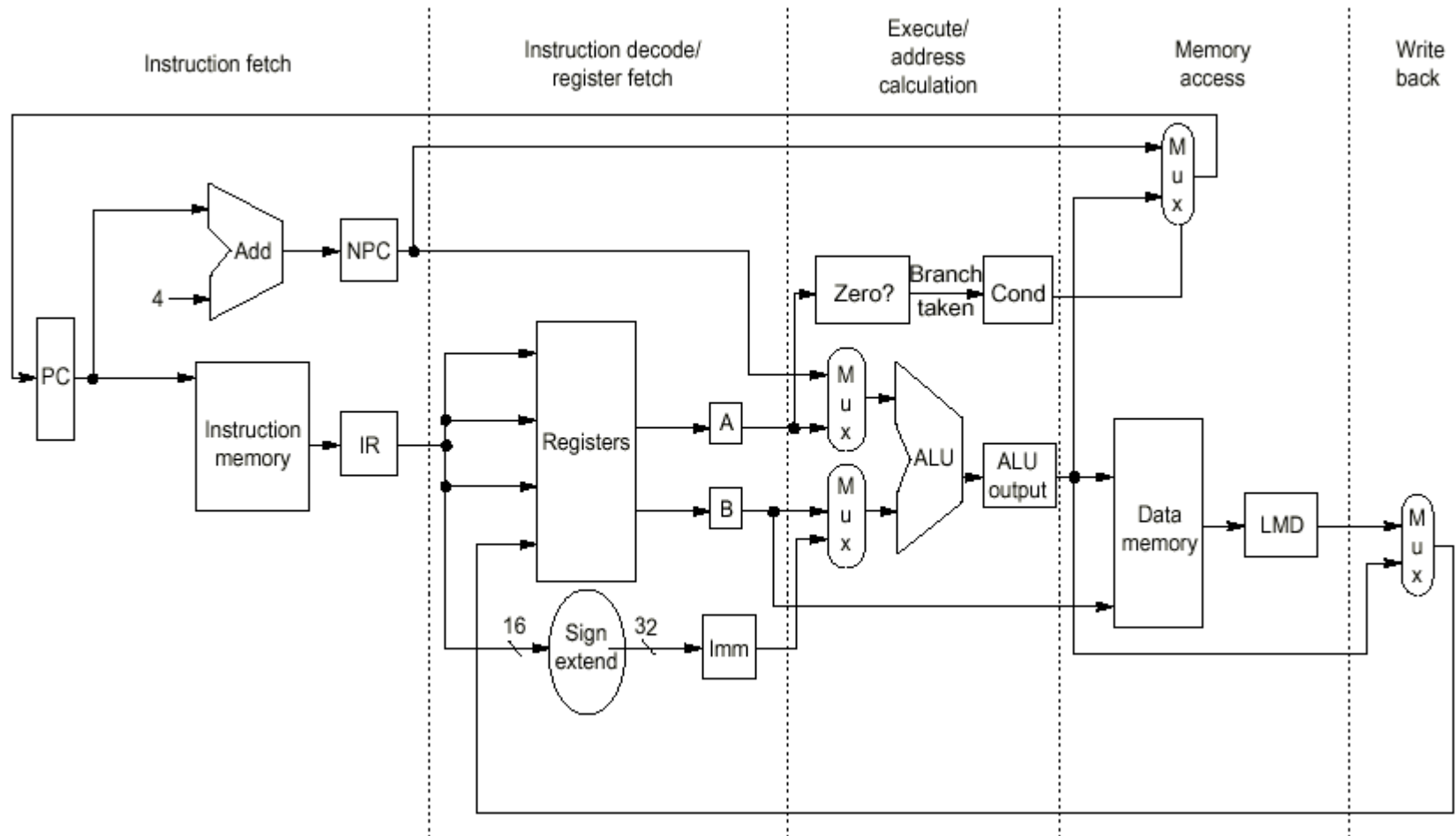
Operaciones aritmético-lógicas en el DLX

Example instruction	Instruction name	Meaning
ADD R1, R2, R3	Add	$\text{Regs}[\text{R1}] \leftarrow \text{Regs}[\text{R2}] + \text{Regs}[\text{R3}]$
ADDI R1, R2, #3	Add immediate	$\text{Regs}[\text{R1}] \leftarrow \text{Regs}[\text{R2}] + 3$
LHI R1, #42	Load high immediate	$\text{Regs}[\text{R1}] \leftarrow 42 \# \# 0^{16}$
SLLI R1, R2, #5	Shift left logical immediate	$\text{Regs}[\text{R1}] \leftarrow \text{Regs}[\text{R2}] \ll 5$
SLT R1, R2, R3	Set less than	$\text{if } (\text{Regs}[\text{R2}] < \text{Regs}[\text{R3}])$ $\text{Regs}[\text{R1}] \leftarrow 1 \text{ else } \text{Regs}[\text{R1}] \leftarrow 0$

Operaciones de bifurcación en el DLX

Example instruction	Instruction name	Meaning
J name	Jump	$PC \leftarrow name; ((PC+4) - 2^{25}) \leq name < ((PC+4) + 2^{25})$
JAL name	Jump and link	$R31 \leftarrow PC+4; PC \leftarrow name; ((PC+4) - 2^{25}) \leq name < ((PC+4) + 2^{25})$
JALR R2	Jump and link register	$Regs[R31] \leftarrow PC+4; PC \leftarrow Regs[R2]$
JR R3	Jump register	$PC \leftarrow Regs[R3]$
BEQZ R4, name	Branch equal zero	if $(Regs[R4] == 0)$ $PC \leftarrow name; ((PC+4) - 2^{15}) \leq name < ((PC+4) + 2^{15})$
BNEZ R4, name	Branch not equal zero	if $(Regs[R4] != 0)$ $PC \leftarrow name; ((PC+4) - 2^{15}) \leq name < ((PC+4) + 2^{15})$

Ruta de datos del DLX



Etapas en la ejecución de instrucciones

- ▶ **Etapas y ciclos de reloj. Máquina secuencial**
 - ▶ **IF: Búsqueda de la instrucción**
 - ▶ 1 ciclo de reloj
 - ▶ **ID: Decodificación**
 - ▶ 0.8 ciclo de reloj
 - ▶ **EX: Ejecución**
 - ▶ Depende de la unidad funcional
 - ▶ **MEM: Acceso a memoria**
 - ▶ 1 ciclo de reloj
 - ▶ **WB: Escritura de resultados**
 - ▶ 0.8 ciclo de reloj

Etapa IF

- ▶ Se carga la instrucción en curso en el *Registro de Instrucción* (RI)
- ▶ (IMAR) Instruction Memory Address Register.
- ▶ Se prepara el contador de programa para apuntar a la siguiente instrucción.

$$IMAR \leftarrow PC$$

$$IR \leftarrow Mem[PC]$$

$$NPC \leftarrow PC + 4$$

Etapa ID

- ▶ Se decodifica la instrucción
- ▶ Se cargan los diferentes campos en los registros especiales A, B, Imm.

$$A \leftarrow \text{Regs}[IR6...10]$$

$$B \leftarrow \text{Regs}[IR11...15]$$

$$\text{Imm} \leftarrow ((IR16)16 \text{ \#\# } IR16...31)$$

Etapa EX

- ▶ Acceso a memoria (cálculo de la dirección efectiva)

$$ALUOutput \leftarrow A + Imm$$

- ▶ Reg-Reg ALU (operación especificada por *func*)

$$ALUOutput \leftarrow A \text{ func } B$$

- ▶ Reg-Imm ALU (operación especificada por *op*)

$$ALUOutput \leftarrow A \text{ op } Imm$$

- ▶ Salto (cálculo del PC destino y de la condición)

$$ALUOutput \leftarrow NPC + Imm$$

$$Cond \leftarrow (A \text{ op } 0)$$

Etapa MEM

- ▶ Acceso a memoria (lectura-Load o escritura-Store en memoria)

$$LMD \leftarrow Mem[ALUOutput]$$
$$Mem[ALUOutput] \leftarrow B$$

- ▶ Salto (carga de la dirección efectiva en el PC)

if (cond)

$$PC \leftarrow ALUOutput$$

else

$$PC \leftarrow NPC$$

Etapas WB

- ▶ Reg-Reg ALU:

$$\text{Regs}[\text{IR} / 6 \dots 20] \leftarrow \text{ALUOutput}$$

- ▶ Reg-Imm ALU:

$$\text{Regs}[\text{IR} / 1 \dots 15] \leftarrow \text{ALUOutput}$$

- ▶ Instrucción de carga (Load):

$$\text{Regs}[\text{IR} / 1 \dots 15] \leftarrow \text{LMD}$$

Directivas

- ▶ Dos punteros:
 - ▶ Datos (.data). Por defecto → \$DATA=0x1000
 - ▶ Instrucciones (.text). Por defecto → \$DATA=0x100
- ▶ ***align n*** . Ocasiona que el próxima dato o instrucción sea cargado en la próxima dirección con los *n* bits de más bajo peso a 0 (la dirección más cercana que sea mayor o igual a la dirección actual que sea múltiplo de 2_n).
 - ▶ Por ejemplo, si *n* es 2, la siguiente dirección sobre la que se escribirá será la inmediatamente siguiente que sea múltiplo de 4.
- ▶ ***.ascii "string1", "..."*** Almacena en memoria las cadenas "strings" indicadas en la directiva como una lista de caracteres. Las cadenas no se completan con un byte 0.
- ▶ ***.asciiz "string1", "..."*** Similar a *.ascii*, excepto que cada cadena es terminada por un byte 0 de forma similar a los strings en C.
- ▶ ***.byte byte1, byte2, ...*** Almacena secuencialmente en memoria los bytes indicados en la directiva.
 - ▶ Por ejemplo, *.byte 0x1, 0x2, 0x3* almacena a partir de la última dirección utilizada en el segmento de datos los valores 0x1, 0x2 y 0x3 consecutivamente utilizando un byte para cada uno de ellos.

Directivas

- ▶ **.data [address]** Ocasiona que el código o datos que sigue a esta directiva sea almacenado en el área de datos
- ▶ **.double number1,...** Almacena secuencialmente en memoria los números indicados en la directiva en doble precisión.
- ▶ **.global label** Hace pública la etiqueta para que pueda ser referenciada por código perteneciente a archivos cargados en memoria después de éste.
- ▶ **.space size** Mueve size bytes hacia adelante el actual puntero de almacenamiento con el fin de dejar libre algún espacio en memoria.
- ▶ **.text [address]** Ocasiona que el siguiente código o dato que aparezca en el fichero sea almacenado en el área de texto (código).
- ▶ **.word word1, ,...** Almacena secuencialmente en memoria las direcciones de los símbolos indicados en la directiva.
 - ▶ Si por ejemplo, el símbolo *PrintFormat* hace referencia a la dirección 1017, se almacenará en memoria este valor.

Trap

▶ **Excepciones (TRAPs).**

- ▶ La interfaz entre los programas DLX y el sistema de entradas/salidas se realiza a través del uso de excepciones (TRAPs). En WinDLX hay cinco TRAPs definidos
 - ▶ Trap #1: Abrir fichero
 - ▶ Trap #2: Cerrar fichero
 - ▶ Trap #3: Leer un bloque del fichero
 - ▶ Trap #4: Escribir un bloque en el fichero
 - ▶ Trap #5: Salida formateada a través de la salida estándar
- ▶ El valor 0 para un TRAP (TRAP #0) no está permitido, y se utiliza para terminar un programa..

Etapas en la ejecución de instrucciones

- ▶ Etapas y ciclos de reloj. **Máquina secuencial**
 - ▶ IF: Búsqueda de la instrucción
 - ▶ 1 ciclo de reloj
 - ▶ ID: Decodificación
 - ▶ 0.8 ciclo de reloj
 - ▶ EX: Ejecución
 - ▶ Depende de la unidad funcional
 - ▶ MEM: Acceso a memoria
 - ▶ 1 ciclo de reloj
 - ▶ WB: Escritura de resultados
 - ▶ 0.8 ciclo de reloj

Ejemplo 1

```
.data 0
.global x
x:    .word 1
.global y
y:    .word 2
.global z
z:    .word 3
.global out
out:  .word 3
```

```
.text 256
```

```
lw r1,x
lw r2,y
lw r3,z
add r4,r1,r2
add r4,r4,r3
sw out,r4
```

```
lw r5,out
```

```
trap #0
```

Ejemplo 1 – Sumar 3 números (Word)

```
.data 0
    .global x
x:    .word 1
    .global y
y:    .word 2
    .global z
z:    .word 3
    .global out
out:  .word 3
```

```
.text 256
```

```
lw r1,x
lw r2,y
lw r3,z
add r4,r1,r2
add r4,r4,r3
sw out,r4
```

```
lw r5,out
```

```
trap #0
```

Etapas:

CPI: Lw: IF;ID;EX;MEM;WB= 4.6

CPI: Add: IF;ID;EX;WB= 3.6

CPI=Sw: IF;ID;EX;MEM= 3.8

Num de ciclos: $4 \times 4.6(Lw) + 2 \times 3.6(Add) + 3.8(Sw) = 18.4 + 7.2 + 3.8 = 29.4$

Ejemplo 2– Sumar 3 números (Word – Desplazamiento)

```
.data 0
.global x
x:     .word 1,2,3
       .global out
out:   .word 3

.text 256

add r7,r0,x ;r7 puntero para moverme por direcciones
lw r1,0(r7)
lw r2,4(r7) ; me muevo con el desplazamiento
lw r3,8(r7)
add r4,r1,r2
add r4,r4,r3
sw out,r4   ; almaceno valor
lw r5,out   ; compruebo en r5

trap #0
```

Ejemplo 3– Sumar 3 números (Doubles – Desplazamiento)

```
.data 0
.global x
x: .double 1,2,3
.global out
out:.double 23

.text 256

ld f2,x      ; cargo primer dato
add r7,r0,x ;r7 puntero para moverme por direcciones

ld f4,8(r7) ; cargo segundo dato
addi r7,r7,16 ; aumento puntero

ld f6,0(r7) ; desplazamiento 0 porque he aumentado puntero

add f8,f2,f4 ; primera suma
add f8,f8,f6 ; segunda suma
add r7,r0,out ;r7 puntero para moverme por direcciones
sd 0(r7),f8    ; almaceno

ld f10,out     ; compruebo valor

trap #0
```

Etapas:

•Ld: IF;ID;EX;MEM;WB= 4.6

•Addd: IF;ID;EX-EX;WB= 4.6

•Add: IF;ID;EX;WB= 3.6

•Sd: IF;ID;EX;MEM= 3.8

Num de ciclos: $4 \times 4.8(\text{Ld}) + 2 \times 4.6(\text{Addd}) + 3 \times 3.6(\text{Add}) + 3.8(\text{Sd}) = 19.2 + 9.2 + 10.8 + 3.8 = 43$

Bloque 1 – Programación DLX nivel básico

Ejecución WinDLX

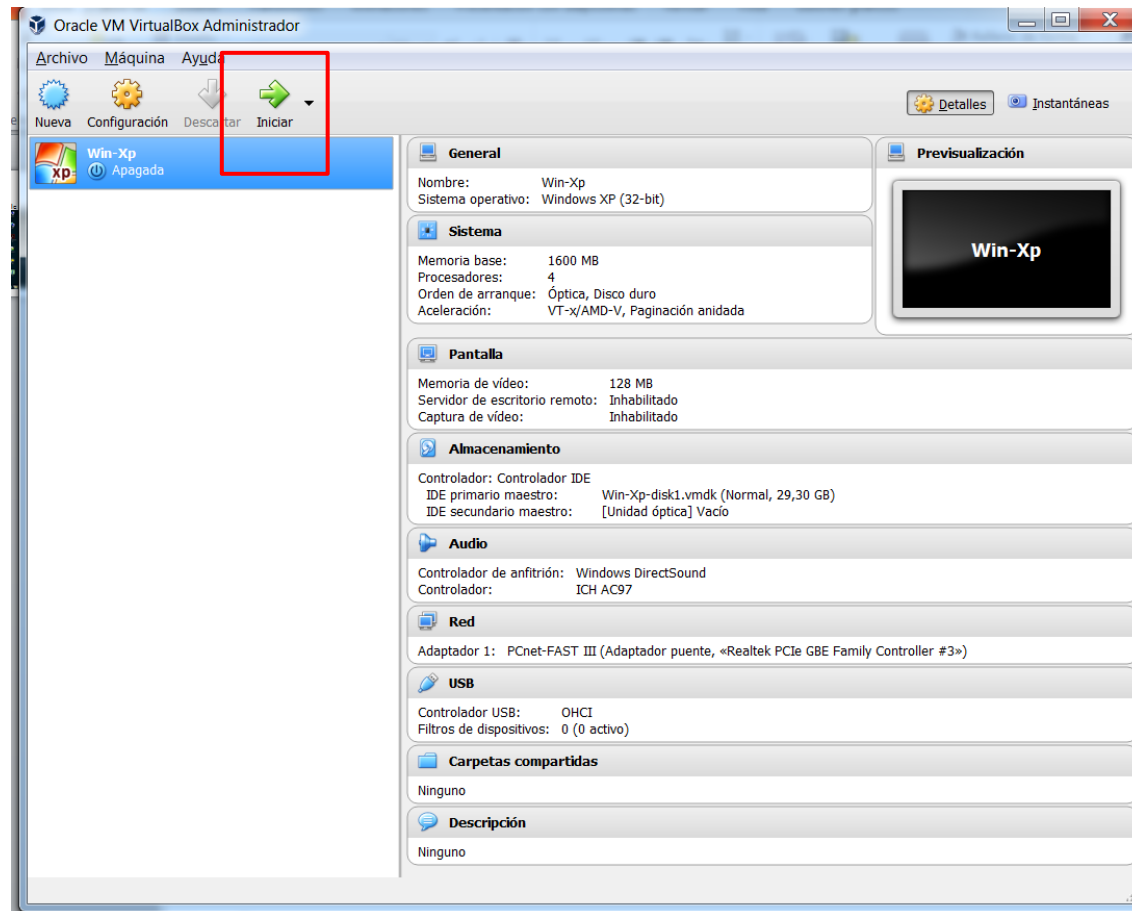
Ejecución WinDLX

► Oracle virtual box



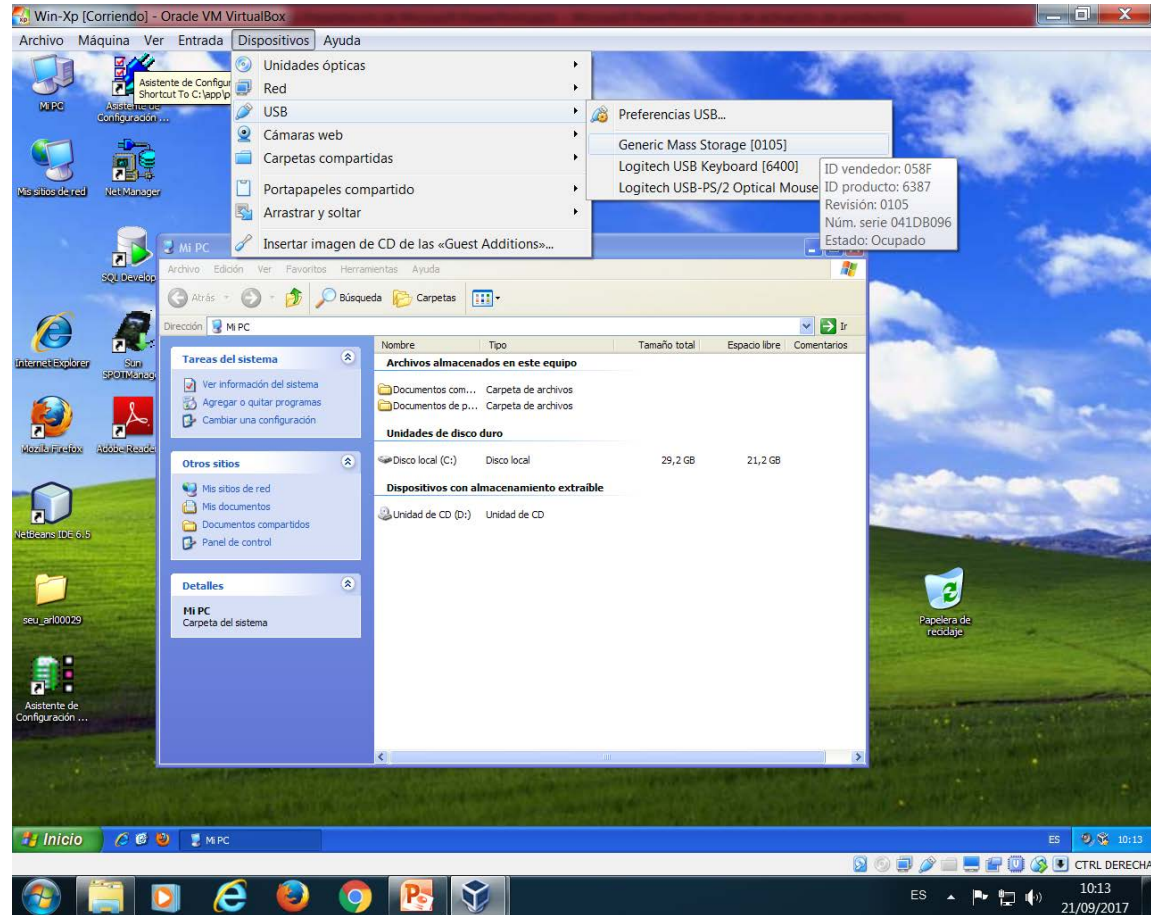
Ejecución WinDLX

- ▶ Iniciar Win-Xp → Iniciar Windows normalmente



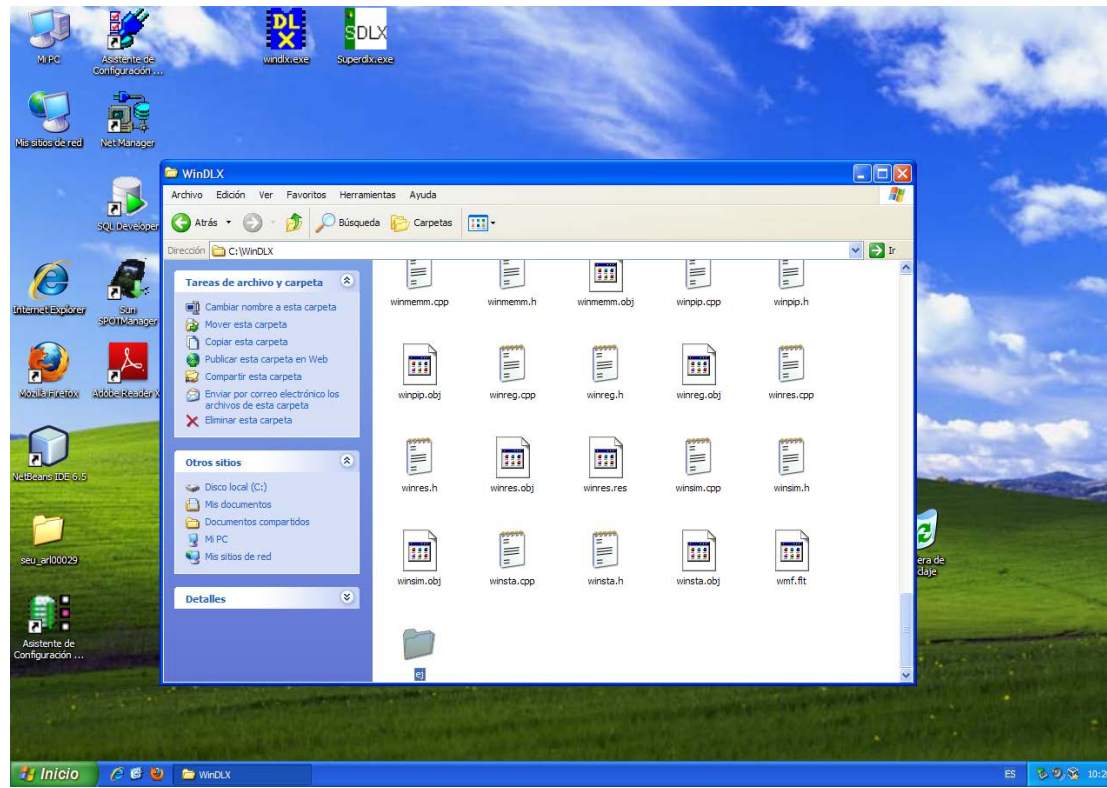
Ejecución WinDLX

► Montar un dispositivo conectado USB



Ejecución WinDLX

- Crear carpeta ej en la carpeta WinDLX



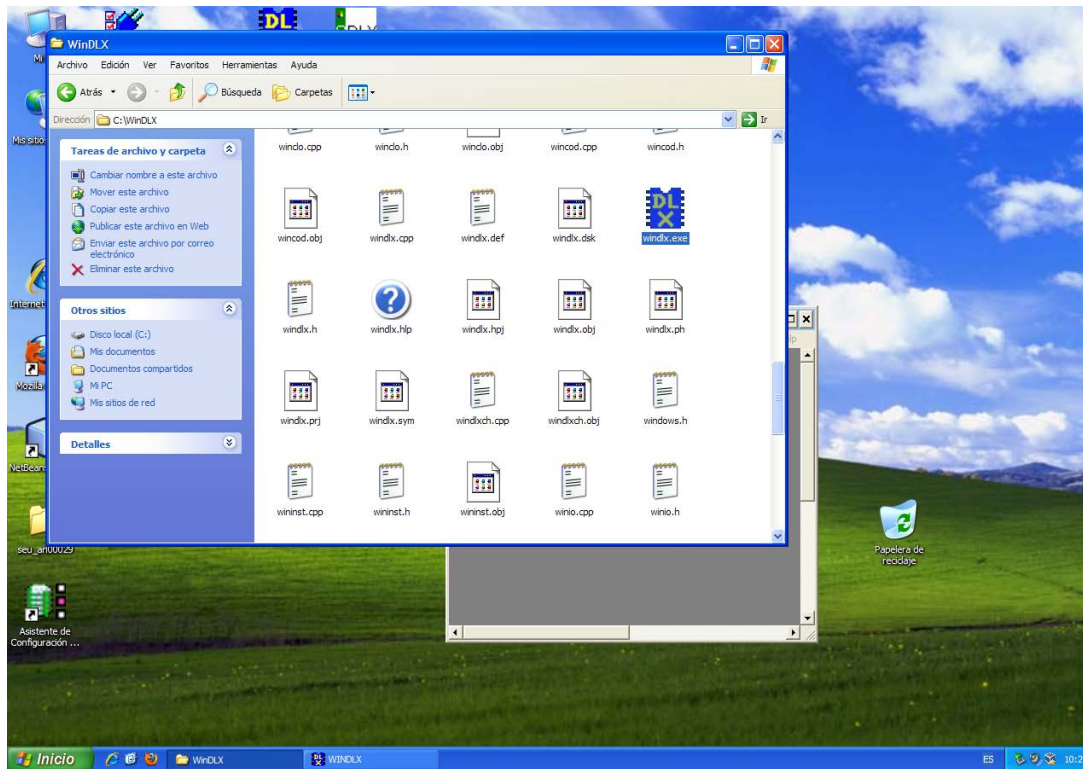
Ejecución WinDLX

- ▶ Iniciar el programa WinDLX
 - ▶ No desde el escritorio



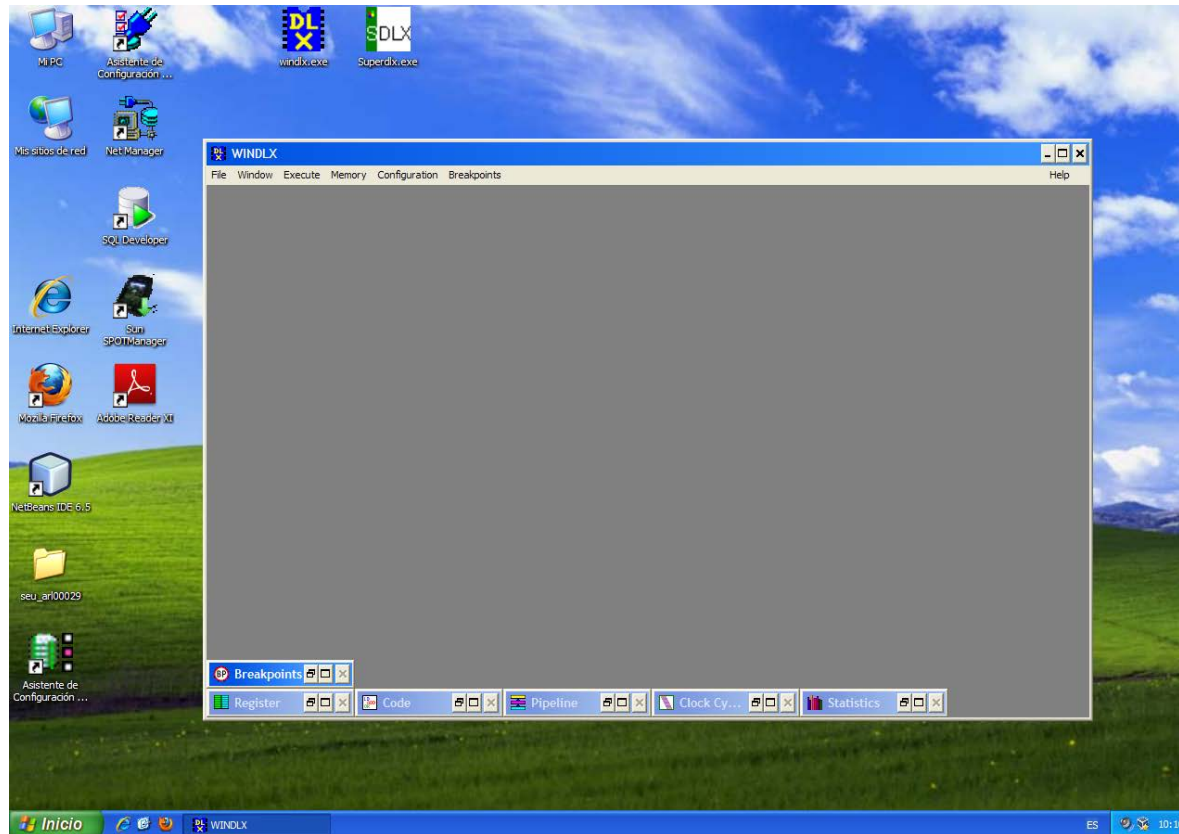
Ejecución WinDLX

- ▶ Iniciar el programa WinDLX
 - ▶ Desde C:/WinDLX



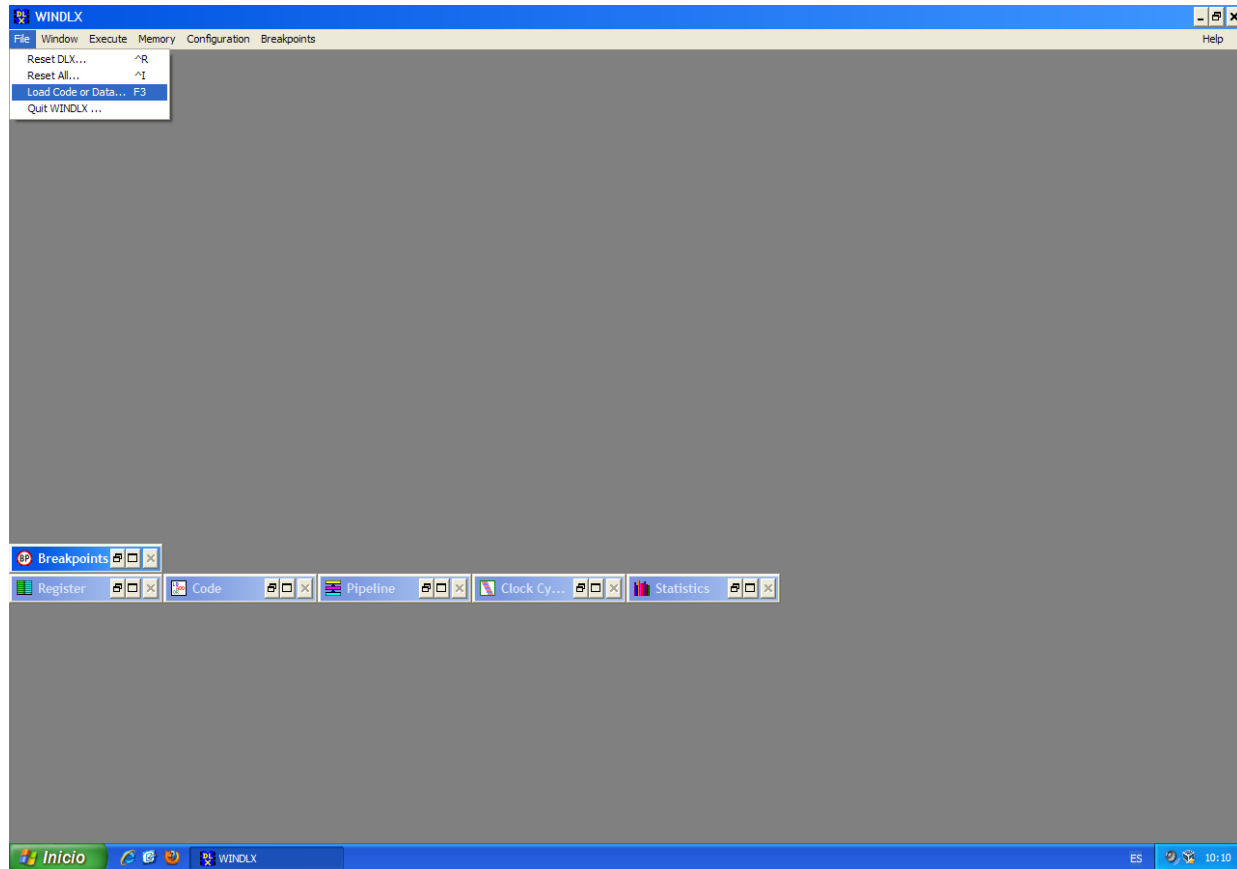
Ejecución WinDLX

► Interfaz WinDLX



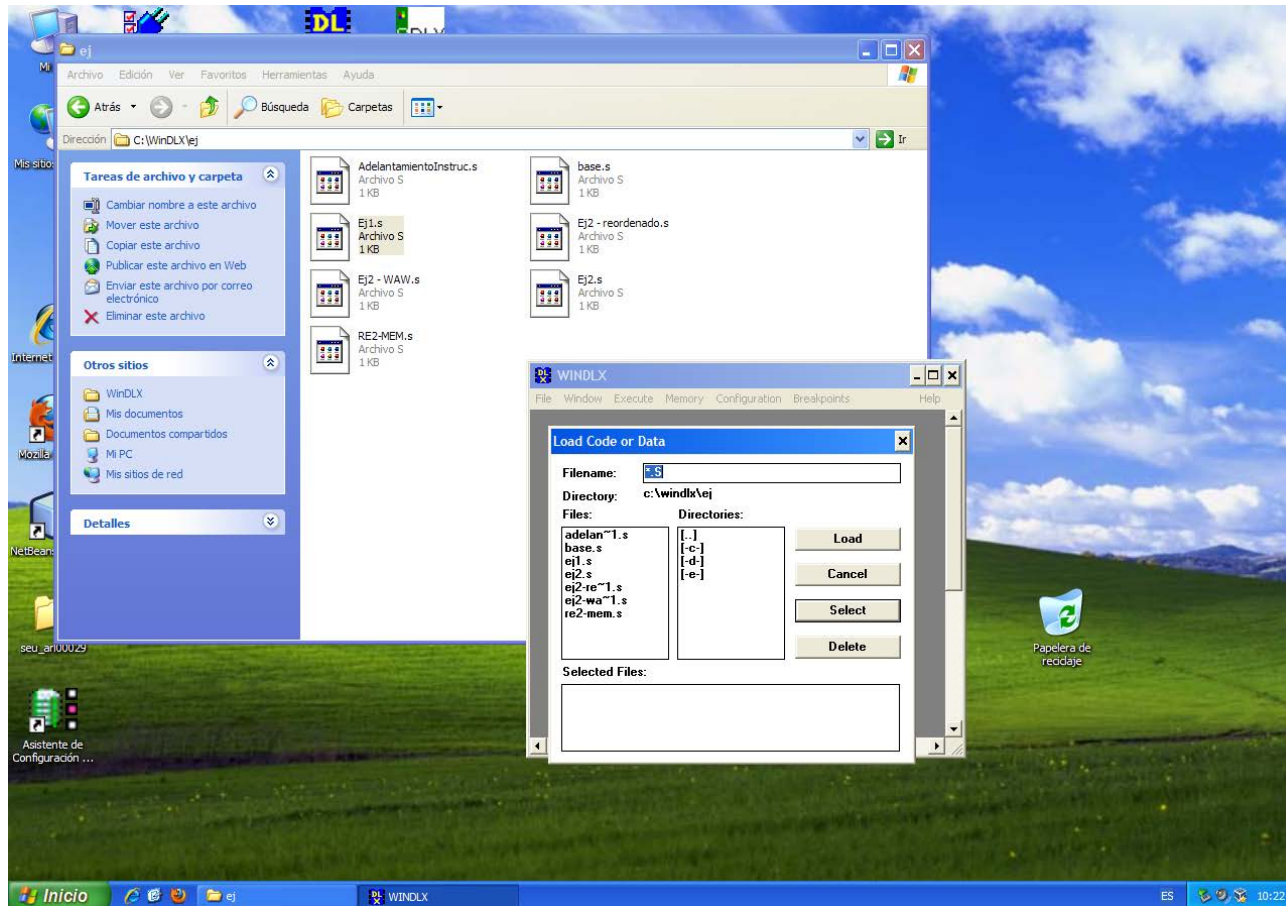
Ejecución WinDLX

► Cargar código



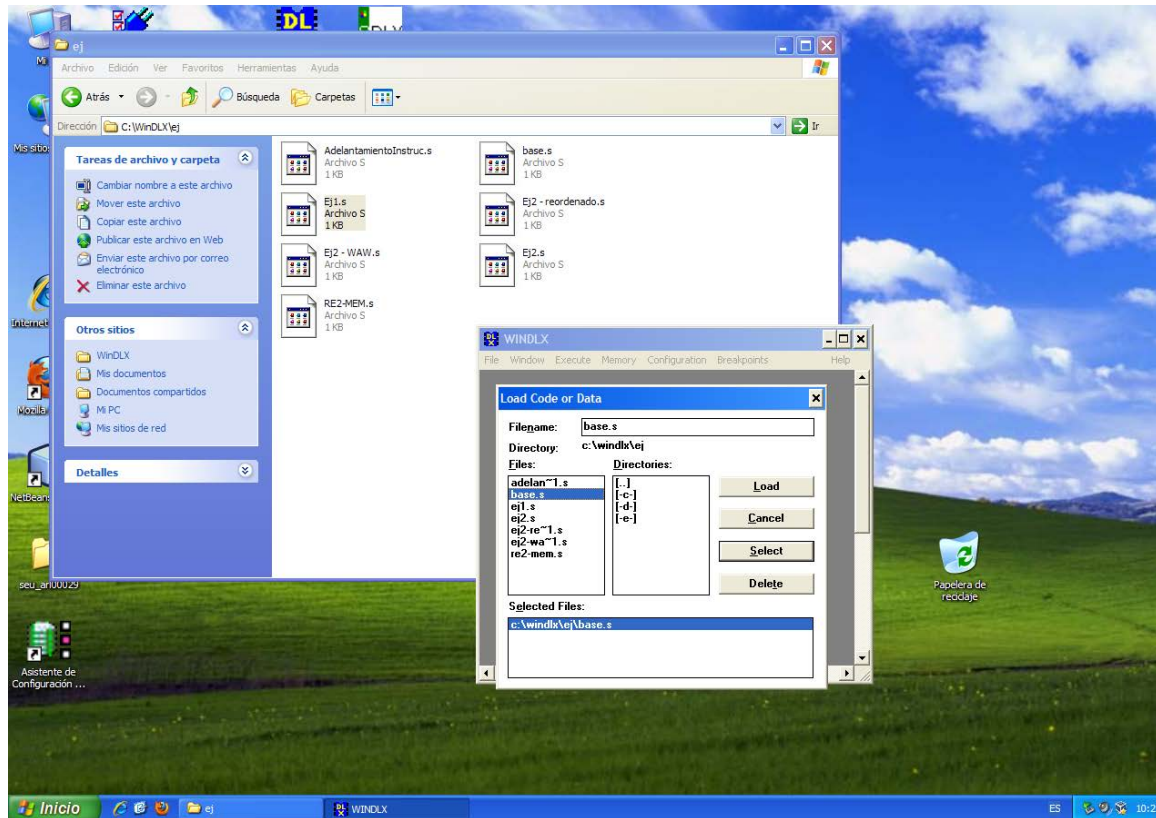
Ejecución WinDLX

- Buscar el archivo. Por ejemplo base.s



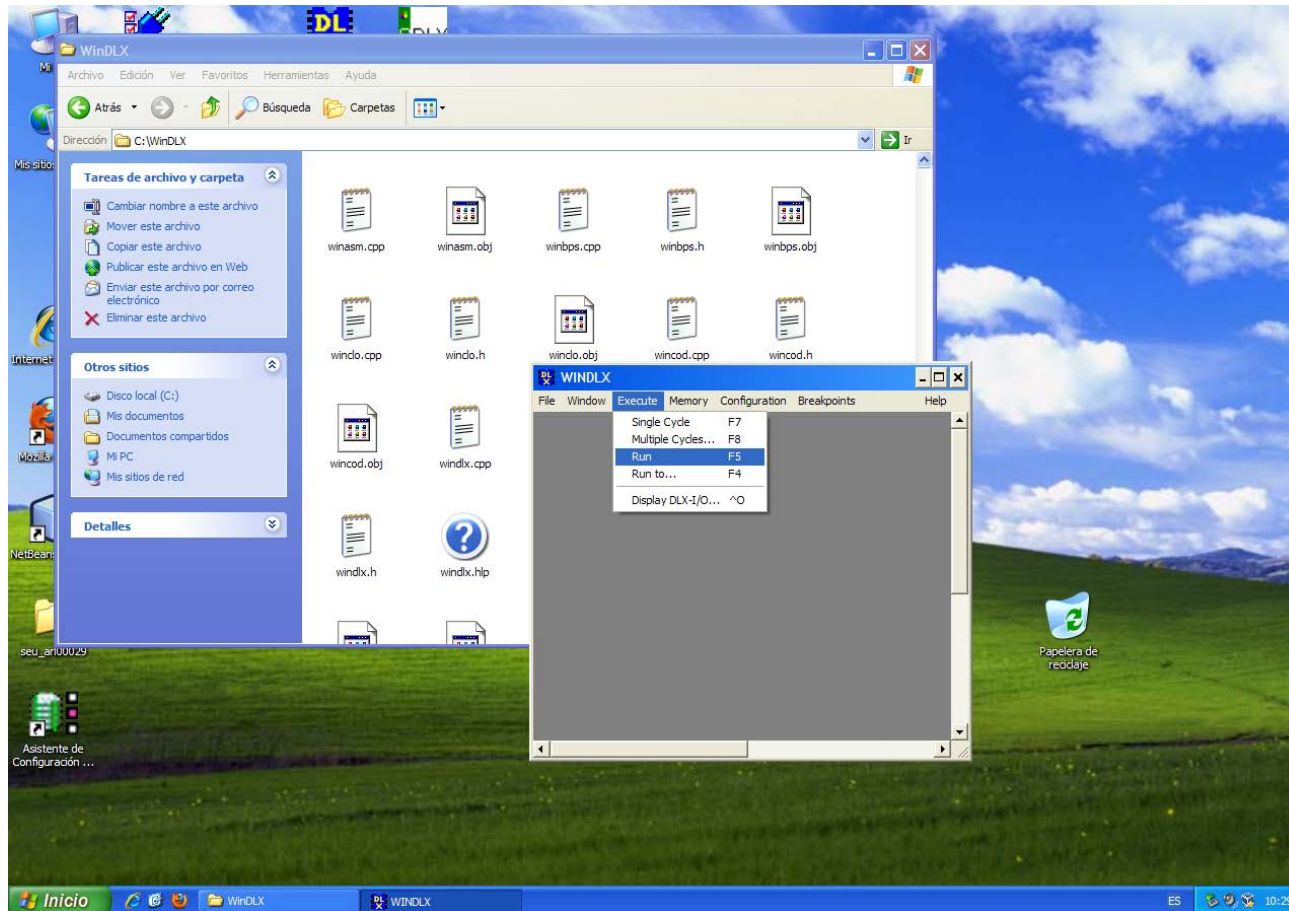
Ejecución WinDLX

► Seleccionar y pulsar Load



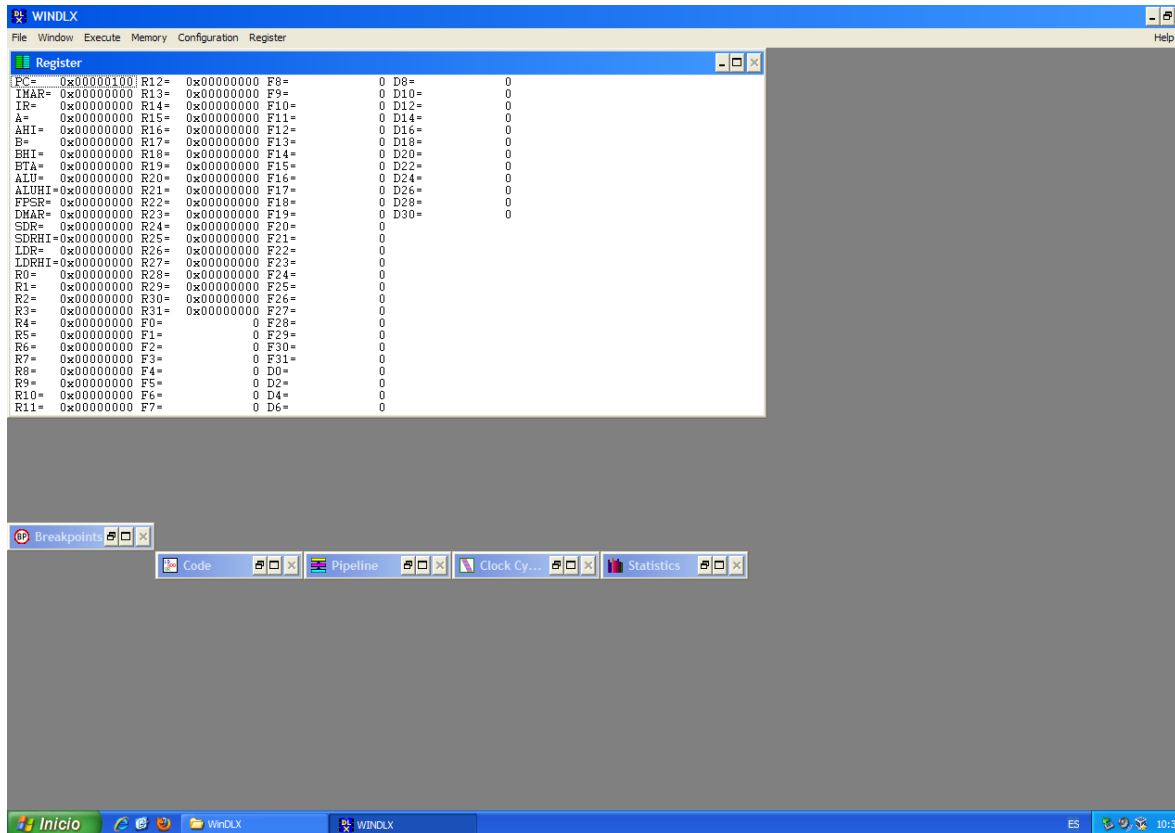
Ejecución WinDLX

► Ejecutar el programa cargado



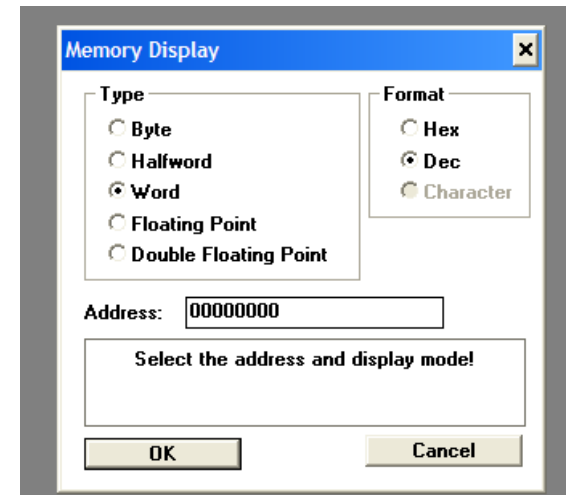
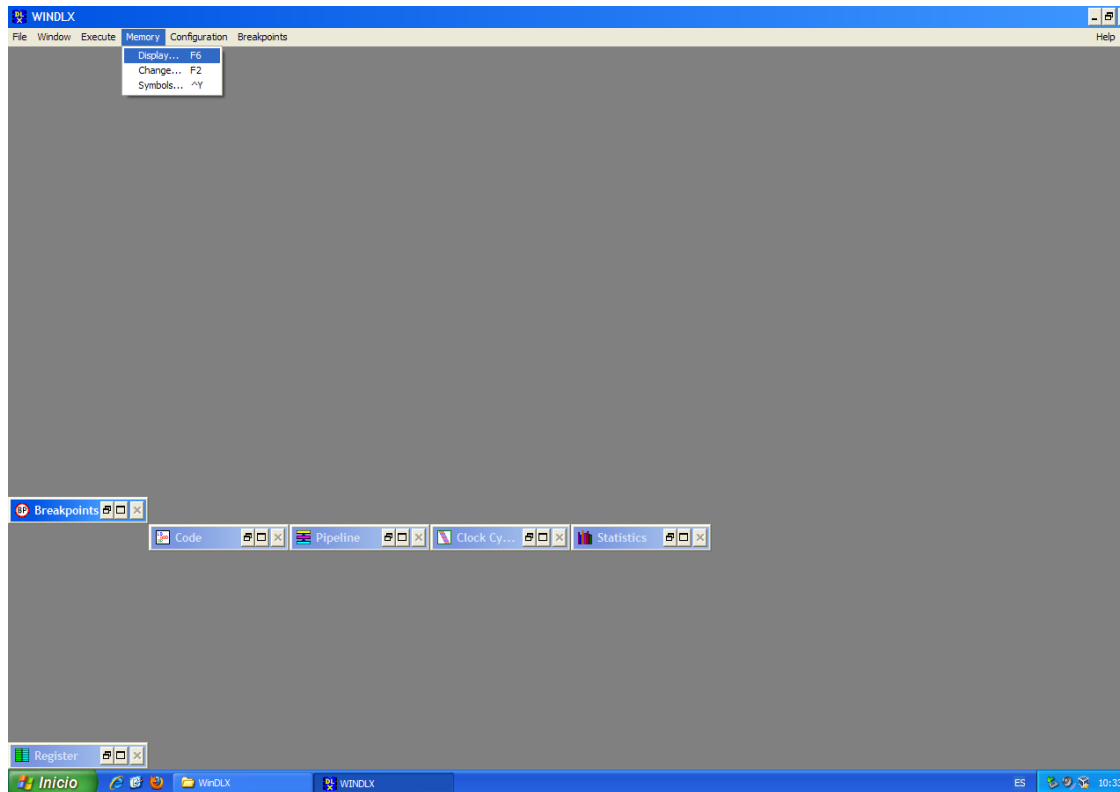
Ejecución WinDLX

► Comprobar registros



Ejecución WinDLX

► Comprobar memoria



► Comprobar memoria

The screenshot shows the WINDLX debugger's Memory-1 window. The window title is "Memory-1". The main area displays a list of memory addresses and their values. The address 0x00000000 is highlighted. The values are mostly zeros, with some non-zero values at higher addresses. The interface includes a menu bar (File, Window, Execute, Memory, Configuration, Memory Display) and a status bar at the bottom showing "Inicio" and "WINDLX".

Address	Value
0x00000000	0
0x00000001	0
0x00000002	0
0x00000003	0
0x00000004	0
0x00000005	0
0x00000006	0
0x00000007	0
0x00000008	0
0x00000009	0
0x0000000a	0
0x0000000b	0
0x0000000c	0
0x0000000d	0
0x0000000e	0
0x0000000f	0
0x00000010	0
0x00000011	0
0x00000012	0
0x00000013	0
0x00000014	0
0x00000015	0
0x00000016	0
0x00000017	0
0x00000018	0
0x00000019	0
0x0000001a	0
0x0000001b	0
0x0000001c	0
0x0000001d	0
0x0000001e	0
0x0000001f	0
0x00000020	0
0x00000021	0
0x00000022	0
0x00000023	0
0x00000024	0
0x00000025	0
0x00000026	0
0x00000027	0
0x00000028	0
0x00000029	0
0x0000002a	0
0x0000002b	0
0x0000002c	0
0x0000002d	0
0x0000002e	0
0x0000002f	0
0x00000030	0
0x00000031	0
0x00000032	0
0x00000033	0
0x00000034	0
0x00000035	0
0x00000036	0
0x00000037	0
0x00000038	0
0x00000039	0
0x0000003a	0
0x0000003b	0
0x0000003c	0
0x0000003d	0
0x0000003e	0
0x0000003f	0
0x00000040	0
0x00000041	0
0x00000042	0
0x00000043	0
0x00000044	0
0x00000045	0
0x00000046	0
0x00000047	0
0x00000048	0
0x00000049	0
0x0000004a	0
0x0000004b	0
0x0000004c	0
0x0000004d	0
0x0000004e	0
0x0000004f	0
0x00000050	0
0x00000051	0
0x00000052	0
0x00000053	0
0x00000054	0
0x00000055	0
0x00000056	0
0x00000057	0
0x00000058	0
0x00000059	0
0x0000005a	0
0x0000005b	0
0x0000005c	0
0x0000005d	0
0x0000005e	0
0x0000005f	0
0x00000060	0
0x00000061	0
0x00000062	0
0x00000063	0
0x00000064	0
0x00000065	0
0x00000066	0
0x00000067	0
0x00000068	0
0x00000069	0
0x0000006a	0
0x0000006b	0
0x0000006c	0
0x0000006d	0
0x0000006e	0
0x0000006f	0
0x00000070	0
0x00000071	0
0x00000072	0
0x00000073	0
0x00000074	0
0x00000075	0
0x00000076	0
0x00000077	0
0x00000078	0
0x00000079	0
0x0000007a	0
0x0000007b	0
0x0000007c	0
0x0000007d	0
0x0000007e	0
0x0000007f	0
0x00000080	0
0x00000081	0
0x00000082	0
0x00000083	0
0x00000084	0
0x00000085	0
0x00000086	0
0x00000087	0
0x00000088	0
0x00000089	0
0x0000008a	0
0x0000008b	0
0x0000008c	0
0x0000008d	0
0x0000008e	0
0x0000008f	0
0x00000090	0
0x00000091	0
0x00000092	0</

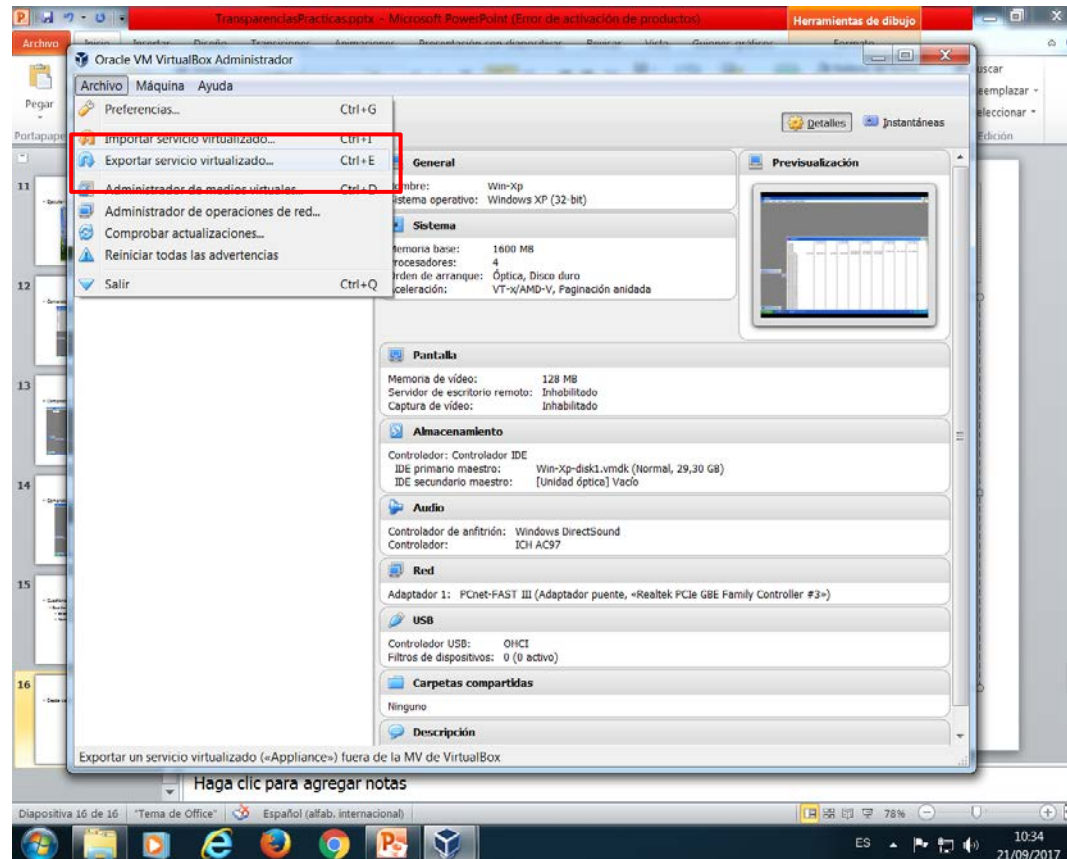
Ejecución WinDLX

- ▶ Cuestiones en consideración:
 - ▶ Guardar ejercicios en WinDLX/ej/
 - ▶ Extension .s
 - ▶ Nombre corto: ej1.s, ej2.s



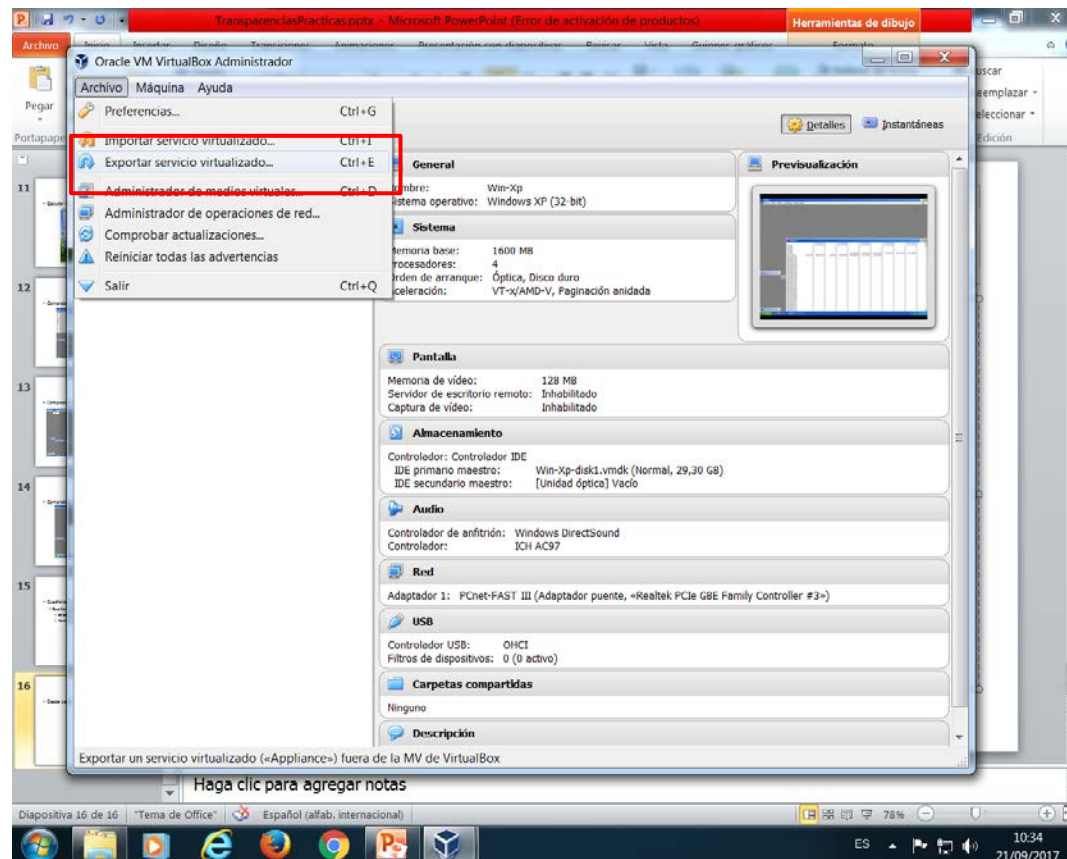
Ejecución WinDLX

- ▶ Desde casa → Exportar el servicio virtualizado
- ▶ Genera archivo .ova



Ejecución WinDLX

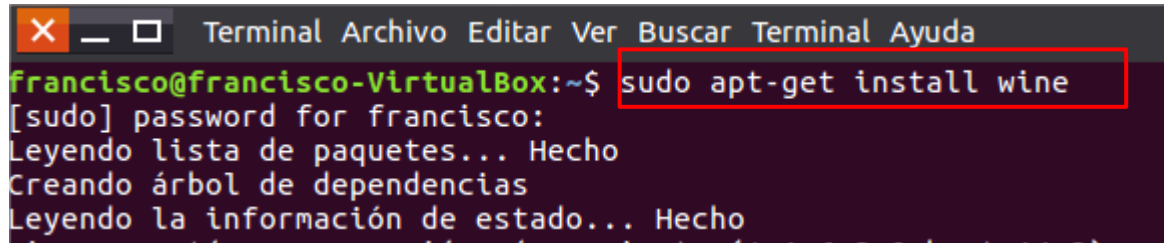
- ▶ Desde casa → Importar servicio virtualizado
- ▶ archivo.ova



Ejecución WinDLX en GNU/Linux

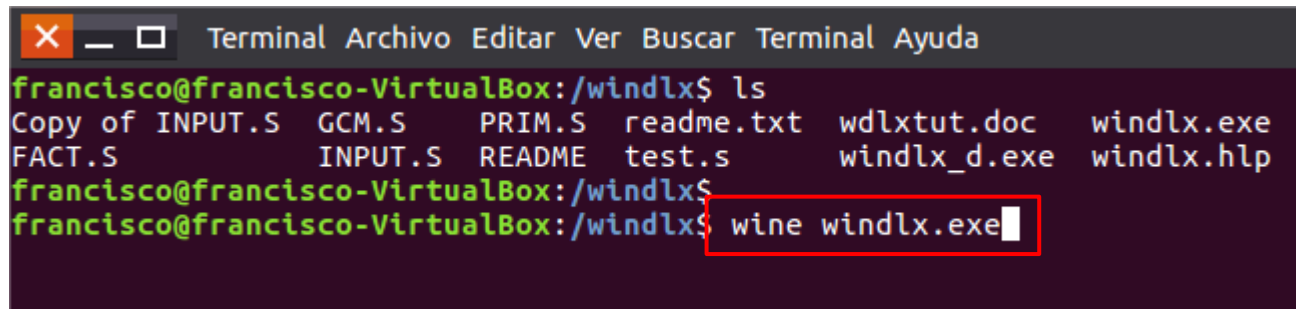
- ▶ Método alternativo → Ejecución desde GNU/Linux

- ▶ Instala el software “wine” en tu distribución de Linux



```
francisco@francisco-VirtualBox:~$ sudo apt-get install wine
[sudo] password for francisco:
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
```

- ▶ Crea una carpeta /windlx en la raíz del sistema de archivos y copia en ella el software WinDLX



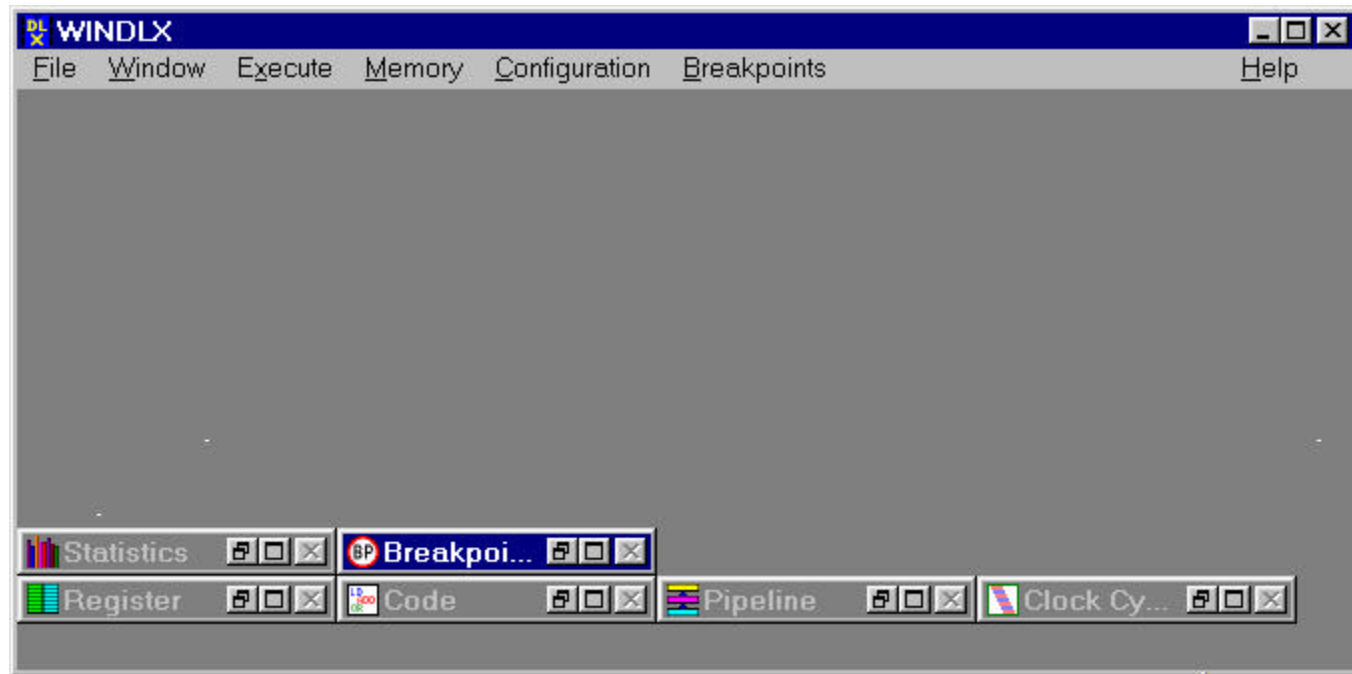
```
francisco@francisco-VirtualBox:/windlx$ ls
Copy of INPUT.S  GCM.S  PRIM.S  readme.txt  wdlxtut.doc  windlx.exe
FACT.S          INPUT.S  README  test.s      windlx_d.exe  windlx.hlp
francisco@francisco-VirtualBox:/windlx$
francisco@francisco-VirtualBox:/windlx$ wine windlx.exe
```

- ▶ Introduce “wine windlx.exe” para ejecutar WinDLX

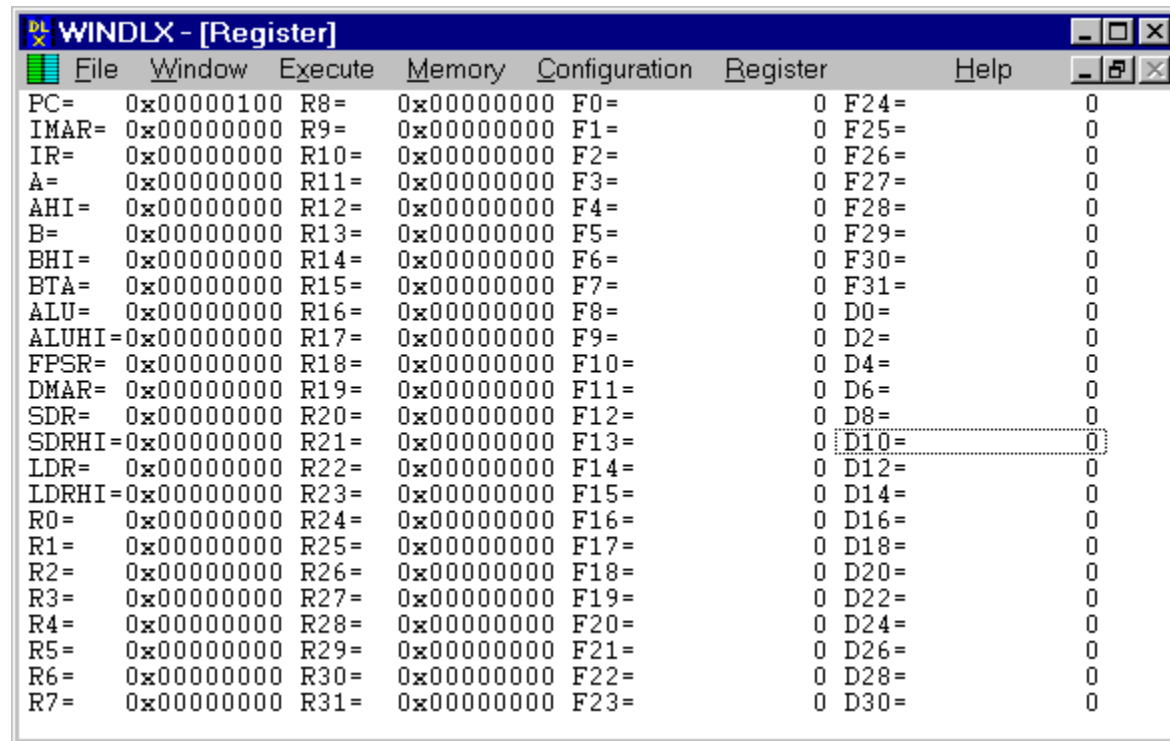
Bloque 1 – Programación DLX nivel básico

Entorno de Simulación - WinDLX

Entorno de Simulación



Entorno de Simulación. Register

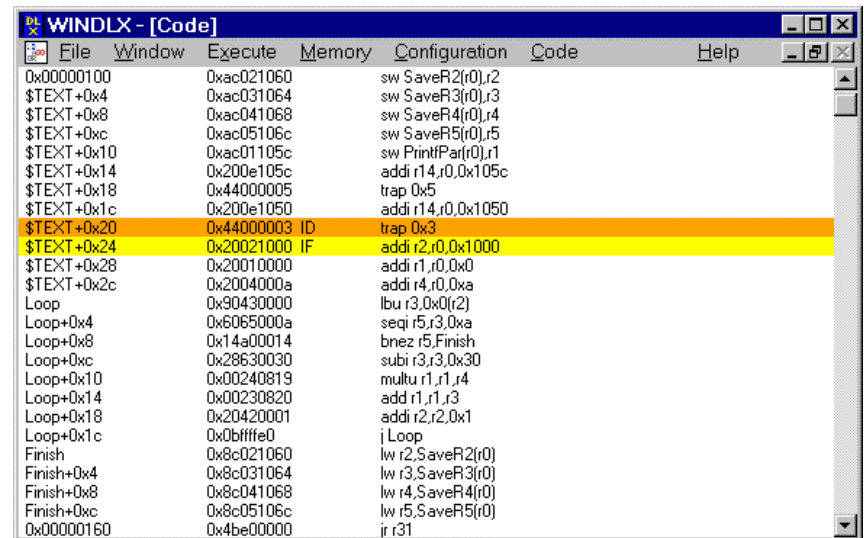


The screenshot shows a window titled "WINDLX - [Register]" with a menu bar containing "File", "Window", "Execute", "Memory", "Configuration", "Register", and "Help". The main area displays a table of CPU registers and their current values. The registers are organized into three columns: PC, R (Registers), and F (Floating Point Registers). The values are shown in hexadecimal format, with some registers having a value of 0.

PC=	0x00000100	R8=	0x00000000	F0=	0	F24=	0
IMAR=	0x00000000	R9=	0x00000000	F1=	0	F25=	0
IR=	0x00000000	R10=	0x00000000	F2=	0	F26=	0
A=	0x00000000	R11=	0x00000000	F3=	0	F27=	0
AHI=	0x00000000	R12=	0x00000000	F4=	0	F28=	0
B=	0x00000000	R13=	0x00000000	F5=	0	F29=	0
BHI=	0x00000000	R14=	0x00000000	F6=	0	F30=	0
BTA=	0x00000000	R15=	0x00000000	F7=	0	F31=	0
ALU=	0x00000000	R16=	0x00000000	F8=	0	D0=	0
ALUHI=	0x00000000	R17=	0x00000000	F9=	0	D2=	0
FPSR=	0x00000000	R18=	0x00000000	F10=	0	D4=	0
DMAR=	0x00000000	R19=	0x00000000	F11=	0	D6=	0
SDR=	0x00000000	R20=	0x00000000	F12=	0	D8=	0
SDRHI=	0x00000000	R21=	0x00000000	F13=	0	D10=	0
LDR=	0x00000000	R22=	0x00000000	F14=	0	D12=	0
LDRHI=	0x00000000	R23=	0x00000000	F15=	0	D14=	0
R0=	0x00000000	R24=	0x00000000	F16=	0	D16=	0
R1=	0x00000000	R25=	0x00000000	F17=	0	D18=	0
R2=	0x00000000	R26=	0x00000000	F18=	0	D20=	0
R3=	0x00000000	R27=	0x00000000	F19=	0	D22=	0
R4=	0x00000000	R28=	0x00000000	F20=	0	D24=	0
R5=	0x00000000	R29=	0x00000000	F21=	0	D26=	0
R6=	0x00000000	R30=	0x00000000	F22=	0	D28=	0
R7=	0x00000000	R31=	0x00000000	F23=	0	D30=	0

Entorno de Simulación. Code

- ▶ Visualización:
 - ▶ Instrucciones
 - ▶ Puntos de ruptura (breakpoints)
- ▶ Instrucción está ejecutándose en una etapa determinada del pipeline:
 - ▶ un color característico de cada etapa
 - ▶ Aparece una etiqueta de la etapa.

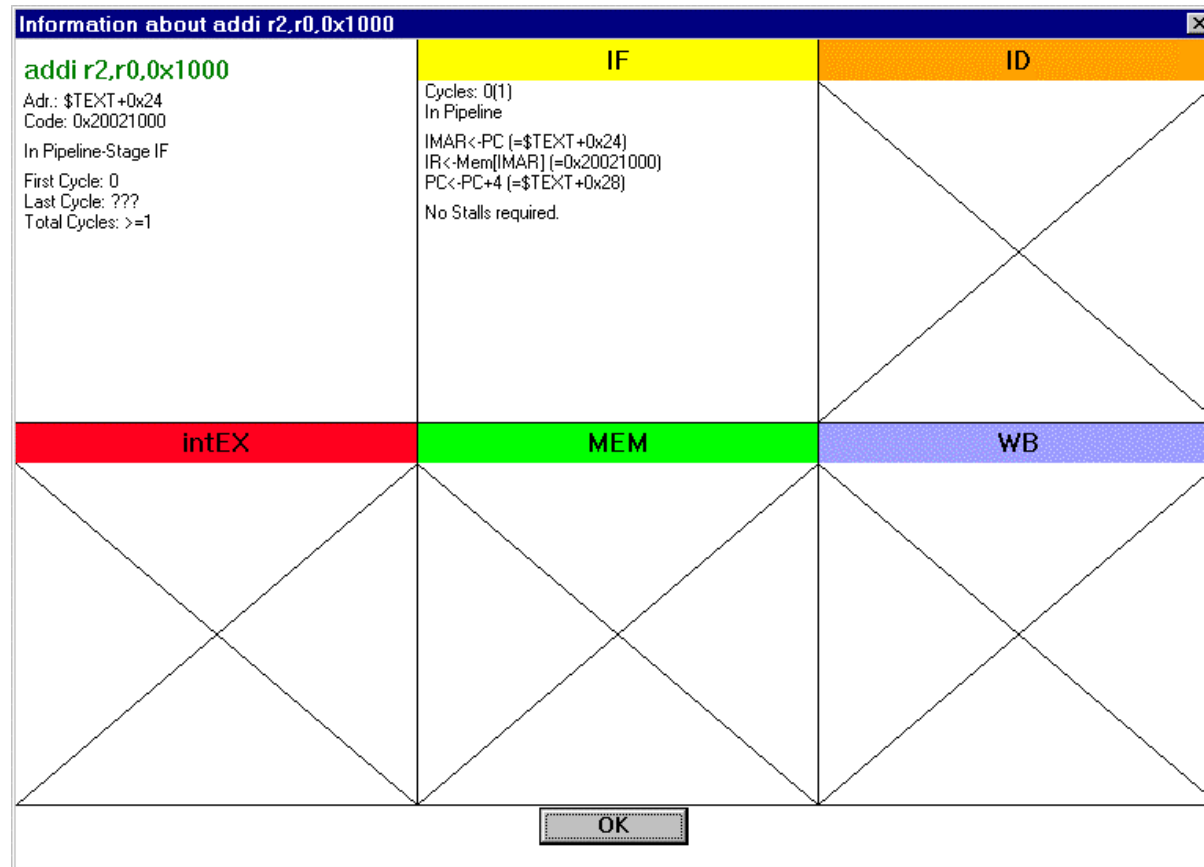


The screenshot shows a window titled "WINDLX - [Code]" with a menu bar (File, Window, Execute, Memory, Configuration, Code, Help) and a list of instructions. Each instruction is displayed with its address, hex value, and the pipeline stage it is currently in. The stages are color-coded: ID (orange), IF (yellow), and others (white). The instruction at address 0x20021000 is in the IF stage, and the instruction at address 0x44000003 is in the ID stage.

Address	Hex Value	Pipeline Stage	Instruction
0x00000100	0xac021060		sw SaveR2(r0),r2
\$TEXT+0x4	0xac031064		sw SaveR3(r0),r3
\$TEXT+0x8	0xac041068		sw SaveR4(r0),r4
\$TEXT+0xc	0xac05106c		sw SaveR5(r0),r5
\$TEXT+0x10	0xac01105c		sw PrintPar(r0),r1
\$TEXT+0x14	0x200e105c		addi r14,r0,0x105c
\$TEXT+0x18	0x44000005		trap 0x5
\$TEXT+0x1c	0x200e1050		addi r14,r0,0x1050
\$TEXT+0x20	0x44000003	ID	trap 0x3
\$TEXT+0x24	0x20021000	IF	addi r2,r0,0x1000
\$TEXT+0x28	0x20010000		addi r1,r0,0x0
\$TEXT+0x2c	0x2004000a		addi r4,r0,0xa
Loop	0x90430000		lbu r3,0x0(r2)
Loop+0x4	0x6065000a		seqi r5,r3,0xa
Loop+0x8	0x14a00014		bnez r5,Finish
Loop+0xc	0x28630030		subi r3,r3,0x30
Loop+0x10	0x00240819		multu r1,r1,r4
Loop+0x14	0x00230820		add r1,r1,r3
Loop+0x18	0x20420001		addi r2,r2,0x1
Loop+0x1c	0x0bffffe0		j Loop
Finish	0x8c021060		lw r2,SaveR2(r0)
Finish+0x4	0x8c031064		lw r3,SaveR3(r0)
Finish+0x8	0x8c041068		lw r4,SaveR4(r0)
Finish+0xc	0x8c05106c		lw r5,SaveR5(r0)
0x00000160	0x4be00000		jr r31

Entorno de Simulación. Code

► Información detallada de las instrucciones:



Entorno de Simulación.

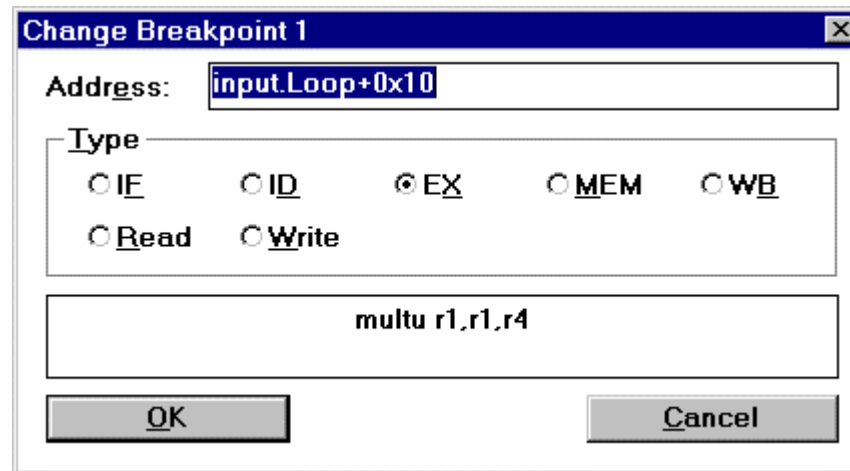
Ventana y menú Pipeline

- ▶ Visualizan las etapas por las que pasan las instrucciones dentro de la estructura del pipeline del procesador.
- ▶ El menú Pipeline:
 - ▶ Display Floating point stages.
 - ▶ Activo:
 - ☐ las etapas en coma flotante
 - ▶ Desactivado
 - ☐ Las cinco etapas básicas del pipeline del DLX

Entorno de Simulación.

Ventana y menú Breakpoints

- ▶ Conjunto de instrucciones que tienen puntos de ruptura asignados.
- ▶ Número máximo de puntos de ruptura es 20.



Menú Principal

► **File.**

- Permite borrar e inicializar simulaciones (opciones Reset DLX y Reset All), cargar los ficheros con los programas en ensamblador y datos (opción Load Code or Data) y salir del programa (Quit WINDLX).

Menú Principal

▶ **Window.**

- ▶ Es la misma que en cualquier aplicación Windows. Permite abrir y modificar las subventanas, que también se pueden manejar directamente sobre los correspondientes iconos de la ventana principal.

Menú Principal

► **Execution.**

- Permite controlar la ejecución de los programas por parte del simulador y mostrar la ventana de E/S de WinDLX donde aparecen los mensajes correspondientes a la simulación (opción **Display DLX-I/O**). Los programas se pueden ejecutar hasta el final (opción **Run**), o hasta un Breakpoint que se haya insertado (opción **Run to**), ciclo a ciclo (opción **Single Cycle**), un número de ciclos prefijado (opción **Multiple Cycles**).

Menú Principal

▶ **Memory.**

- ▶ Permite crear ventanas para visualizar el contenido de la memoria (opción **Display**), ver y cambiar el contenido de ciertas direcciones de memoria (opción **Change**) y manipular los símbolos de las variables (opción **Symbols**).

Menú Principal

► **Configuration.**

- Permite cambiar ciertas características del cauce, concretamente:
 - El número de unidades funcionales y sus latencias (opción **Floating Point Stages**) y la posibilidad de utilizar caminos de bypass o no (opción **Enable Forwarding**);
 - El tamaño de la memoria (opción **Memory Size**);
 - y otras características propias del simulador como:
 - ☐ la utilización de nombres simbólicos o no (opción **Symbolic Addresses**),
 - ☐ la utilización de tiempos absolutos o relativos al ciclo de reloj actual (opción **Absolute Cycle Count**),
 - ☐ el almacenamiento de las características de la configuración en un fichero – con la extensión por defecto .wdc - (opción **Store**), o el establecimiento de una configuración almacenada en un fichero (opción **Load**) .

Menú Principal

► **Help.**

- Permite acceder a una ayuda en inglés bastante completa del simulador WinDLX, las características del cauce, el repertorio de instrucciones, etc.