ESTRUCTURA DE COMPUTADORES

PRÁCTICA Nº 1 DEL TEMA2

INTRODUCCIÓN AL SIMULADOR WINMIPS64

Estas prácticas se basan en las desarrolladas por el Departamento

de Informática Aplica de la Universidad Politécnica de Madrid (http://www.dia.eui.upm.es)

INTRODUCCIÓN

Este documento contiene el enunciado de la práctica de introducción al simulador WinMIPS64.

OBJETIVO

El objetivo perseguido es la familiarización del alumno con el simulador WinMIPS64. Para conseguirlo se llevarán a cabo varias actividades que irán introduciendo de manera gradual los diferentes aspectos del simulador, desde la edición de programas hasta la ejecución de los mismos.

INSTALACIÓN DEL SIMULADOR

El simulador únicamente funciona en entornos Microsoft Windows. Está disponible, en formato ZIP para su instalación, en la web de la asignatura.

Una vez descargado es necesario utilizar una herramienta como WinZIP, para poder extraer su contenido en un directorio destino (p.e. `C:\Winmips64`).

Junto con el simulador (`winmips64.exe`) están disponibles los ficheros siguientes:

- asm.exe — Programa que permite comprobar la sintaxis de un fichero con código MIPS64. Para utilizarlo es necesario una ventana de comandos (cmd.exe), admitiendo como parámetros de entrada el nombre del fichero con extensión ".s". Ejemplo de utilización: `C:\Winmips64> asm prueba.s` (Suponiendo que tanto `asm` como el fichero `prueba.s` están situados en el mismo directorio)
- winmipstut.doc — Fichero en formato MS Word con el Manual de Usuario del simulador en inglés (existen traducciones al español en Internet)
- ejemplos — Programas de ejemplo listos para ser cargados y ejecutados en el simulador.

EL PRIMER PROGRAMA

Antes de mostrar la funcionalidad del simulador se va a describir brevemente el formato de un programa en ensamblador listo para ser cargado en el simulador.

El programa debe editarse con un editor de texto y consta de dos secciones diferenciadas: definición de variables y código.

En la sección de definición de variables se declaran todas las variables que van a ser usadas por el programa. En realidad la definición de variables no es más que reservar un espacio de memoria para alojarlas y asignarlas un nombre simbólico (nombre de la variable) para referenciarlas de una manera más sencilla en el código.

Para definir variables el ensamblador proporciona una serie de directivas que facilitan la declaración de las variables. Conviene que la declaración de cada variable comience en una nueva línea en su columna 0.

El código (las instrucciones) comienza tras la directiva "`.text`" o "`.code`"). Las primeras columnas de la línea se reservan para situar una etiqueta como posible destino de una instrucción de salto.

Disponéis del siguiente programa a modo de ejemplo (p1_ej1.s):

```
; Práctica de introducción al simulador
.data
i:    .word  0
j:    .word  0
.text
          daddi R2,R0,0 ; comentario
          daddi R3,R0,0 ;
          daddi R5,R0,10 ;
WHILE:    slt R6,R2,R5
          beqz R6,ENDWHILE
          daddi R3,R3,5
          sw R3,j(R0)
          daddi R2,R2,1
          sw R2,i(R0)
          j WHILE
ENDWHILE: nop
          halt
```

Programa 1

Este programa corresponde al programa C siguiente:

```
int main ()
{
 int i = 0;
 int j = 0;

 while ( i < 10) {
    j = j + 5;
    i = i + 1;
  }
}
```

Programa 2

En el programa fuente (ensamblador) las directivas "`.data`" y "`.text`" indican el comienzo de la declaración de variables y de código respectivamente. La otra directiva presente "`.word`" indica que la variable ocupara 64 bits de memoria. El identificador que precede a "`.word`" es el nombre simbólico asignado a la variable seguido del carácter ":" (en este caso se han declarado dos variables de nombre "i" y "j"). Por último, el número que aparece tras la directiva es el valor inicial que tendrá la variable. Resumiendo, se han declarado dos variables (i y j) de 32 bits y con valor inicial 0.
El código del programa consta de trece instrucciones y tres etiquetas (`MAIN`, `WHILE` y `ENDWHILE`). Como ya se ha comentado antes la declaración de etiquetas se realiza en el comienzo de la línea (columna 0) terminando su nombre con el carácter ":".

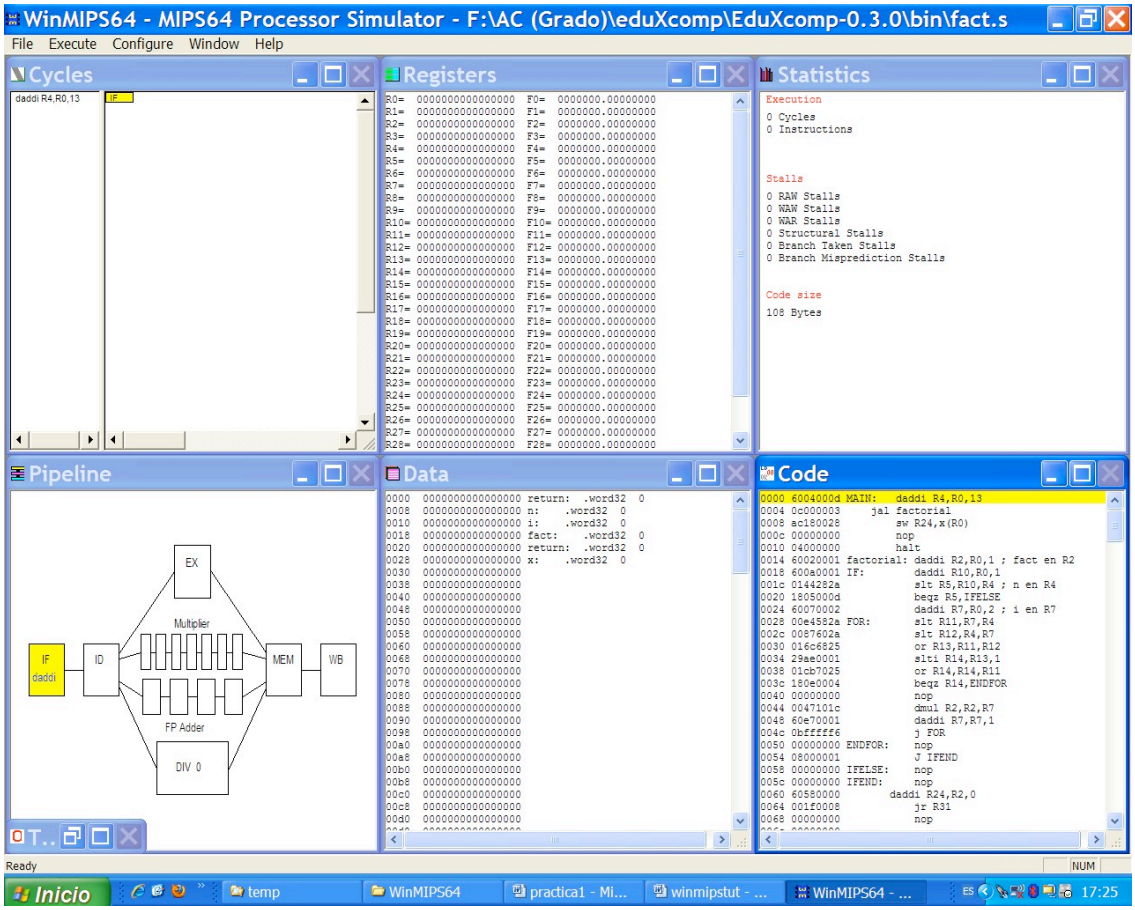
Las etiquetas WHILE y ENDWHILE son destinos de las instrucciones de salto (`beqz` y `j`). El nombre de la etiqueta suele ser representativo del código en el que se utiliza, en este caso se trata de un bucle *while*.

En el fichero también pueden aparecer comentarios, se considera como tal cualquier texto que aparezca a la derecha del carácter ";"

Las constantes numéricas pueden expresarse en base decimal o bien en hexadecimal anteponiendo al valor los caracteres (0x). Un ejemplo sería `daddi r1,r0,0xf` (inicializa el registro r1 con el valor hexadecimal F).

ARRANQUE DEL SIMULADOR

Para arrancar el simulador basta con replicar en el icono correspondiente al fichero **winmips64.exe** desde el explorador de ficheros. El aspecto del simulador es:



Consta de siete ventanas:

1. **Pipeline** Muestra la estructura del cauce con las etapas y las diferentes unidades funcionales disponible. Al ejecutarse un programa va mostrando qué instrucciones están en cada etapa.

2. **Code** Muestra el contenido del segmento de memoria. En la primera columna aparece la dirección de memoria en la que se encuentra cargada la instrucción, a continuación el código máquina y, por último, la instrucción en ensamblador tal y como se editó. Si en el código no se ha indicado nada, el simulador carga el código a partir de la dirección 0 del segmento de código. El contenido de las direcciones se muestra en palabras de 32 bits.

3. **Data** Contenido del segmento de datos con las variables declaradas en el programa. A diferencia de la ventana de código, el contenido de las direcciones se muestra en palabras de 64 bits. El simulador utiliza ordenación *little-endian* para las variables de tamaño superior a un octeto.

4. **Register** Contenido de los registros de propósito general (desde R0 a R31) y de coma flotante (F0-F31). El contenido de los registros puede modificarse en cualquier momento repicando sobre su nombre.

5. **Cycles** Muestra el contenido del cauce a lo largo del tiempo de ejecución. El tamaño del buffer histórico está limitado. Cada etapa del cauce tiene asignado un color y un nombre.

6. **Statistics** Contiene información sobre la ejecución del programa (nº instrucciones ejecutadas, ciclos empleados, paradas clasificadas por tipo de riesgo, etc.).

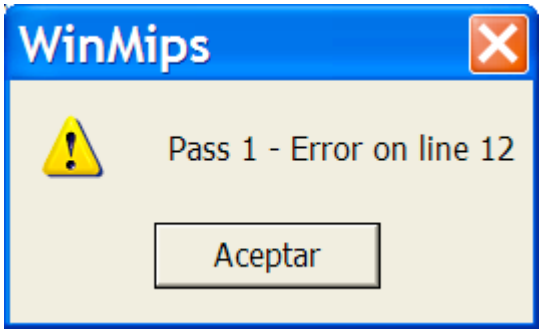
7. **Terminal** Simula el comportamiento de un terminal de E/S, permitiendo a los programas leer y escribir información desde/hacia el exterior.

Todos los valores numéricos mostrados en las anteriores ventanas están expresados en hexadecimal.

En la barra de estado de la ventana principal del simulador se van mostrando informaciones de cómo transcurre la ejecución de un programa en el modo paso a paso (Single Cycle)

CARGA DE UN PROGRAMA

Una vez que tengamos editado el programa en un fichero, se procede a su carga en el simulador utilizando la opción **Open** del menú **File**. Si tras seleccionar el fichero y dar al botón **Abrir**[1] no aparece la ventana siguiente es que el programa es sintácticamente correcto y se puede proceder a ejecutarlo:



Una vez cargado, las ventanas de código (Code) y datos (Data) aparecen rellenas a partir de la información que contenía el fichero que se acaba de cargar.

[1] *El nombre del botón aparece en español al ser esté el idioma disponible en el sistema operativo en el que se ejecuta el simulador*

Cuando aparece la ventana anterior indica que existe al menos un error, pero desgraciadamente muchas veces los programas contienen más de un error siendo deseable corregirlos todos y no necesitar sucesivas cargas del programa en el simulador para detectarlos todos y proceder a corregirlos.

Cómo ya se ha indicado anteriormente junto con el simulador está disponible el programa asm.exe que permite comprobar la validez de un programa, mostrando todos los errores que contenga.

EJECUCIÓN DE UN PROGRAMA

La ejecución de un programa puede realizarse de tres formas distintas: Ciclo-a-Ciclo (Single Cycle), Multi-Ciclo (Multi-Cycle) y Hasta-Breakpoint (Run-to).

Ciclo-a-Ciclo permite ir ejecutando el programa comprobando que sucede cada vez que transcurre un nuevo ciclo de reloj. Este modo nos permite ver con gran nivel de detalle el funcionamiento del cauce, desde cuántas instrucciones están activas, riesgos en los que incurren, paradas, etc. Para acceder a este modo hay que utilizar la opción ***Single cycle*** *del menú* ***Execute*** *o utilizar la tecla de función* ***F7****.*

En el modo Multi-ciclo la ejecución del programa avanza cinco ciclos (valor por defecto que puede modificarse a través de la opción ***Multi Step*** *del menú* ***Configure*** *o bien pulsando la combinación de teclas* ***ctrl.+T****). Para acceder a este modo se puede utilizar la opción* ***Multi Cycle*** *del menú Execute o bien la tecla de función* ***F8****.*

El modo Hasta-breakpoint ejecuta el programa hasta encontrar un punto de parada o hasta el final del programa (normalmente la instrucción `halt` *es la última, procediendo a detener el procesador). Este modo está disponible mediante la opción* ***Run to*** *del menú* ***Execute*** *o la tecla de función* ***F4****.*

Los puntos de parada se fijan repicando con el ratón en una instrucción de la ventana de código pasando ésta a color azul para indicar que se ha fijado un punto de parada. Para quitarlos se procederá de idéntica forma, es decir, repicando en la instrucción que lo tenga *fijado, pasando a continuación del color azul al negro original.*

En cualquier momento de la simulación podemos reiniciar el simulador mediante las opciones ***Reset MIPS64*** *y* ***Full Reset*** *del menú* ***File****. La primera opción simula un reset hardware del procesador, la segunda además borra el contenido de la memoria.*

OTRAS COSAS DEL SIMULADOR

El simulador tiene mucha más funcionalidad que iremos viendo en las siguientes prácticas.

TAREAS A REALIZAR

Para llevar a cabo las tareas es necesario configurar el simulador con las opciones Forwarding y Delay Slot inhibidas (menú **Configure** opciones **Enable Forwarding** y **Enable Delay** Slot)

Una vez visto el funcionamiento básico del simulador se proponen las tareas siguientes:

1. Editar y ejecutar p1_ej1.s en el simulador el que hemos visto como ejemplo.
   Respondiendo a las cuestiones siguientes:
   a. ¿Qué registros se utilizan en el programa para trabajar con las variables i y j?

   b. ¿Qué hace la instrucción `slt R6,R2,R5`? ¿Qué ocurre si intercambias los dos últimos registros de la instrucción?

   c. ¿Qué registro tiene almacenado el número de veces que tiene que ejecutarse el bucle?

   d. Comprueba en la memoria que valores tienen las variables i y j tras finalizar la ejecución del programa.

   e. ¿Qué estrategia de salto se está utilizando para los saltos condicionales e incondicioneles? ¿En qué etapa se produce la terminación del salto en ambos casos?

2. En el fichero p1_ej2.s encontrarás un programa que almacena en las 20 posiciones de un vector A el valor 2^4. La función POWER realiza el cálculo de la potencia de dos realizando sucesivos productos (dmul) por 2. El tamaño de los elementos del array es de 32 bits.
   a. Comprobar el tiempo de ejecución (consulta los ciclos en la ventana statistics) empleado en la ejecución total del programa, así como el que corresponde a instrucciones de la función POWER.

   b. Si sustituimos toda la función POWER por el uso de la instrucción dsll (dsll R3,R3,4) emplearíamos un solo ciclo en el cálculo de la potencia de 2. Determina los parámetros de la ley de Amdahl, y aplícala para predecir cuál será la aceleración que conseguiremos al ejecutar el programa.

   c. Cambia el programa, sustituyendo la llamada a POWER por la instrucción de desplazamientos a la izquierda, dsll (dsll R3,R3,4). Comprueba el tiempo de ejecución (ciclos) para esta segunda versión. ¿Es consistente con la predicción que realizaste en el apartado anterior?

HOJA DE RESPUESTAS

Nombre y Apellidos:

Cuestiones

1.

a)

b)

c)

d)

e)

2.

a)

b)

c)