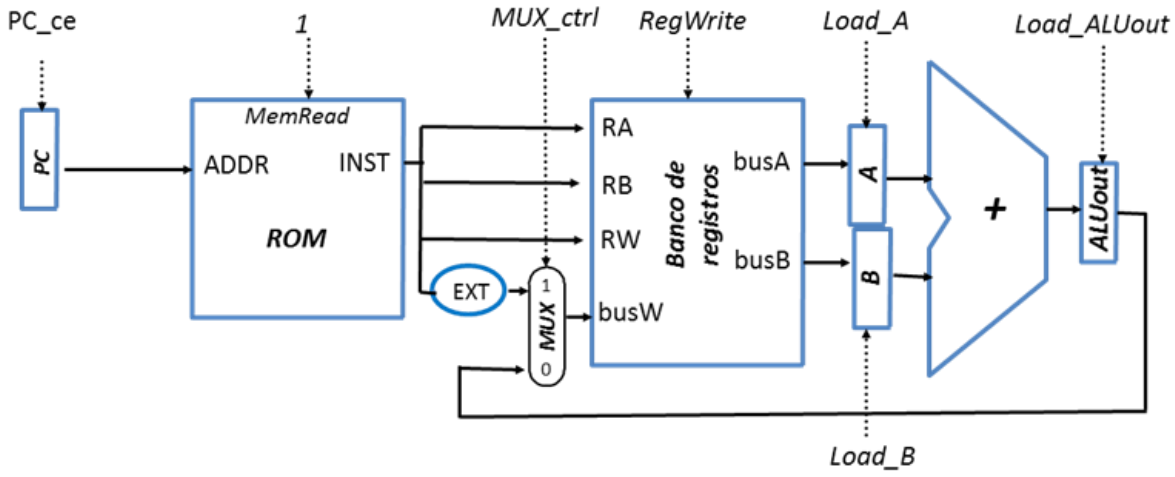
DISEÑO DE UN PROCESADOR CON BANCO DE REGISTROS

Práctica 2



Arquitectura y Organización de Computadores 2
2º Grado Ingeniería Informática

Escuela de Ingeniería y Arquitectura
Universidad Zaragoza

Departamento de Informática e Ingeniería de Sistemas
Universidad Zaragoza

1 RESUMEN

En esta segunda sesión vamos a aprender a trabajar con un lenguaje de descripción de hardware y un entorno de simulación profesional basado en bancos de prueba y cronogramas. Para ello vamos a diseñar un procesador muy sencillo que trabaje con el banco de registros y un sumador. Lo programaremos para que realice una serie de escrituras y sumas. Una vez diseñado realizaremos un análisis temporal para averiguar la frecuencia máxima a la que puede funcionar, y visualizaremos la ejecución de las instrucciones en un cronograma.

***Al entrar al laboratorio** coloca el **trabajo previo del apartado 2.1** encima de la mesa, de forma visible. Antes de empezar a trabajar conéctate a Moodle desde uno de los equipos del laboratorio y realiza el **control de asistencia a la sesión**.*

*La práctica finaliza cuando el procesador funciona correctamente y ha sido entregado a través de **Moodle**. También hay que entregar los **resultados del apartado 3**.*

2 DISEÑO DE UN PROCESADOR CON BANCO DE REGISTROS

2.1 TRABAJO PREVIO: DISEÑO EN PAPEL

a) Baja los ficheros de la práctica[1] y estudia las entradas y salidas de los módulos VHDL que vas a usar en la práctica. Dibuja sobre papel los siguientes componentes: un banco de registros **BR32** *(BReg_delays.vhd*), una ROM (*ROM_I_delay.vhd*), un contador (*counter_delay.vhd*), un sumador de 32 bits (*adder32_delay.vhd*), un multiplexor (*mux2_1_delay.vhd*), tres registros *A, B* y *ALUout (REGISTER_delay)*, y una unidad de control (*UC_Mov_add.vhd*) que genera las dos señales de control: ***RegWr,*** para escribir en el banco de registros; ***MUX_ctrl***, para elegir qué se escribe en el banco de registros, ***PC_ce***, para que el pc avance, y las señales de carga de los registros. En la portada tienes un esquema de la ruta de datos, en la que sólo faltan algunos detalles (como los bits que se usan en algunas conexiones).

b) Interconecta los componentes, teniendo en cuenta que:
   - En cada dirección de la ROM se escribirá una instrucción.
   - El contador se utilizará para direccionar la ROM.
   - Con el sumador se calculará la suma de los dos registros leídos del **BR32**.
   - El multiplexor seleccionará el dato adecuado a escribir en el **BR32**.
   - El procesador debe soportar las siguientes instrucciones:

```
mov K,rd          ; SignExt(K)-> BR(rd);pc++     K es un inmediato de 16 bits
add ra,rb,rd      ; BR(ra) + BR(rb)-> BR(rd); pc++
halt              ; la ejecución se detiene
```

c) Define el formato de instrucción y la codificación de las instrucciones.
   - Presta atención a dónde hemos codificado K en el esqueleto de ruta de datos que te damos en *Mov_add.vhd*
   - El campo para el registro destino debe de ser el mismo en mov y add
   - En principio puedes colocar el código de operación donde quieras, pero sería conveniente colocarlo en los bits de más peso del formato de instrucción.

[1] Están en Moodle pero desde el lab o con ssh a hendrix los puedes copiar a tu cuenta desde /misc/practicas/aoc2 y descomprimir con 'unzip AOC2_práctica2.zip'

d) Dibuja el autómata de la unidad de control siguiendo un esquema **Moore**. Como existen registros intermedios, cada instrucción se ejecutará en varios ciclos. En cada estado especifica sus acciones RTL (no es necesario que indiques el valor de las señales de control).
e) Utilizando las instrucciones anteriores escribe un programa ensamblador (NIA.asm) que guarde en los registros r1 – r7 los valores obtenidos en webdiis.unizar.es/~luisma/aoc2/nia1.php a partir de tu NIA y calcule la suma de todos ellos en r0.

2.2 TRABAJO EN EL LABORATORIO: DISEÑO EN VHDL

a) Copia los ficheros vhd de la práctica y sigue las instrucciones del flujo de trabajo con GHDL que encontrarás en Moodle.
b) Simula el banco de pruebas *BR_test.vhd* el tiempo necesario para poder ver todas las pruebas. Abre la simulación con GTKWave, y carga la configuración de cronograma *br_testbench.gtkw* (*File / Read Save File*).
c) Comprueba que las distintas operaciones de lectura y escritura funcionen correctamente. Localiza en el cronograma los retardos de propagación $d_{RA \rightarrow busA}$ y d_{busA}.
d) Abre el procesador *Mov_add.vhd* con el editor de texto que prefieras. No modifiques el nombre del archivo, tan sólo **rellena la información de la cabecera con tus datos**. Los componentes ya están definidos e instanciados, pero debes conectarlos.
e) Describe en VHDL la UC que has diseñado. Sigue el esquema de *UC_mov_add.vhd*. No modifiques el nombre del archivo, tan sólo **rellena la información de la cabecera con tus datos.** Asigna un nombre descriptivo a cada estado, especifica qué señales se activan en cada estado (UC tipo Moore), y define el estado siguiente en función del estado actual y del código de operación.
f) Traduce tu programa del apartado 2.1.d a **hexadecimal** y programa la ROM. Para ello edita *ROM_I_delay.vhd* y sustituye los 0x00000000 por las instrucciones necesarias.
g) Simula con GHDL el test *Mov_Add_test.vhd* el tiempo necesario para ejecutar tus instrucciones. Abre la simulación con GTKWave y carga la configuración de cronograma *mov_add_testbench.gtkw*. Revisa la ejecución de las instrucciones ciclo a ciclo. ¿Se están ejecutando las instrucciones que esperabas? ¿Realizan las escrituras correctas en BR? Ten en cuenta que los módulos que utilizas tienen retardos asociados, puedes ver sus retardos en el código, o en el apartado de análisis temporal.
h) Finalmente entrega tu circuito a través del recurso Moodle 'Entrega práctica 2'. Debes entregar también capturas de pantalla del autómata de la unidad de control y del cronograma de la ejecución.

3 ANÁLISIS TEMPORAL

a) Los módulos VHDL tienen los siguientes retardos asignados: d_{REG} = 5 ns; $tsetup_{REG}$ = 1 ns; d_{ROM} = 8 ns; d_{CONT} = 6 ns; $tsetup_{CONT}$ = 1 ns; $d_{RA \rightarrow busA}$ = 7 ns; d_{busA} = 5 ns; $tsetup_{RW}$ = $tsetup_{RegWr}$ = 3 ns; d_{SUM} = 26 ns; $d_{MUX\_X \rightarrow Z}$ = 2 ns; $d_{MUX\_S \rightarrow Z}$ = 3 ns; d_{UC} = 3 ns. Se asume que los tiempos de *setup* no indicados son despreciables y no los modelamos.
b) Identifica el camino crítico del procesador multiciclo y su retardo (*dmax*). ¿Cuál es su frecuencia máxima (MHz) de funcionamiento (*fmax*)?
c) Abre *Mov_add_test.vhd* y ajusta el tiempo de ciclo a *fmax* editando la línea:
constant CLK_period : time := 100 ns;. Comprueba que todo sigue funcionando correctamente.
d) ¿Cuánto tarda el procesador en ejecutar tu código (tex)?
e) Cuando tengas todos los cálculos realizados introdúcelos a través del recurso Moodle 'Análisis temporal práctica 2'.