ESTRUCTURA DE COMPUTADORES
Grado en Ingeniería Informática

Sesión de laboratorio número 2

La ruta de datos y control

1. Objetivos

- Afianzar el funcionamiento de la ruta de datos del MIPS R2000
- Analizar cómo se decodifican y ejecutan instrucciones de diversos formatos.

2. Desarrollo

En esta práctica vamos a trabajar sobre el simulador ProcSim (v2.0) de la ruta de datos del procesador MIPS. El objetivo de la práctica es familiarizarse con el simulador y comprender el ciclo completo de ejecución de diferentes instrucciones. El simulador se ejecuta sobre una máquina virtual Java.

3. Toma de contacto con el simulador: Instrucciones de tipo R

Actividad: Iniciar el simulador y cargar una arquitectura y un código en ensamblador.

Para realizar esta práctica debemos iniciar el simulador. El simulador está en una carpeta que debemos copiar desde poliformaT en el escritorio del ordenador[1]. Para ejecutar el simulador debemos lanzar el programa *ProcSim.exe*. Una vez iniciado, nos aparece una ventana donde podemos seleccionar la ruta de datos a utilizar (hay diferentes versiones de la ruta de datos) y el código ensamblador a ejecutar sobre la ruta de datos. El diálogo que nos aparece y las diferentes rutas de datos y código se muestran en las siguientes figuras.

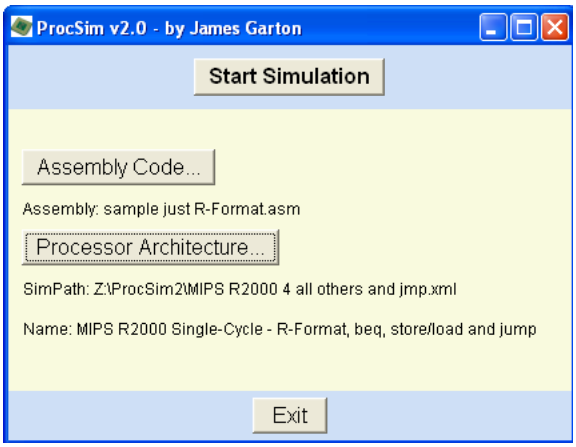


Ilustración 1. Diálogo principal del simulador.

[1] La carpeta (ProcSim) se puede copiar en cualquier otra ubicación local del ordenador, pero, atención: deben evitarse rutas de acceso largas (ej: C:\MisDocumentos\...\...\ETC\Prac3**ProcSim**), pues el simulador da problemas al acceder a las rutas de datos.

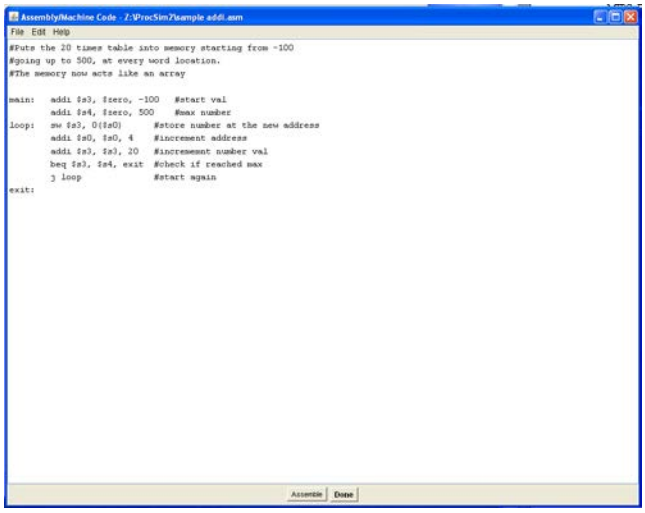


Ilustración 2. Diálogo de edición y selección de código ensamblador.

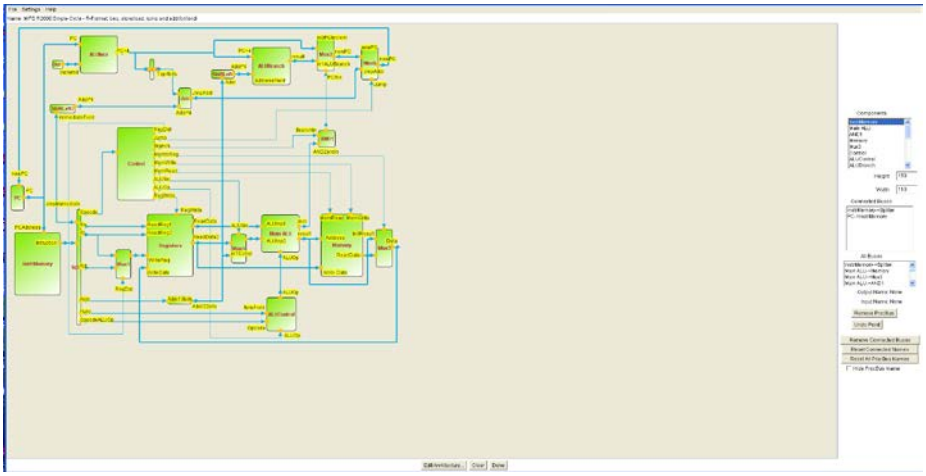
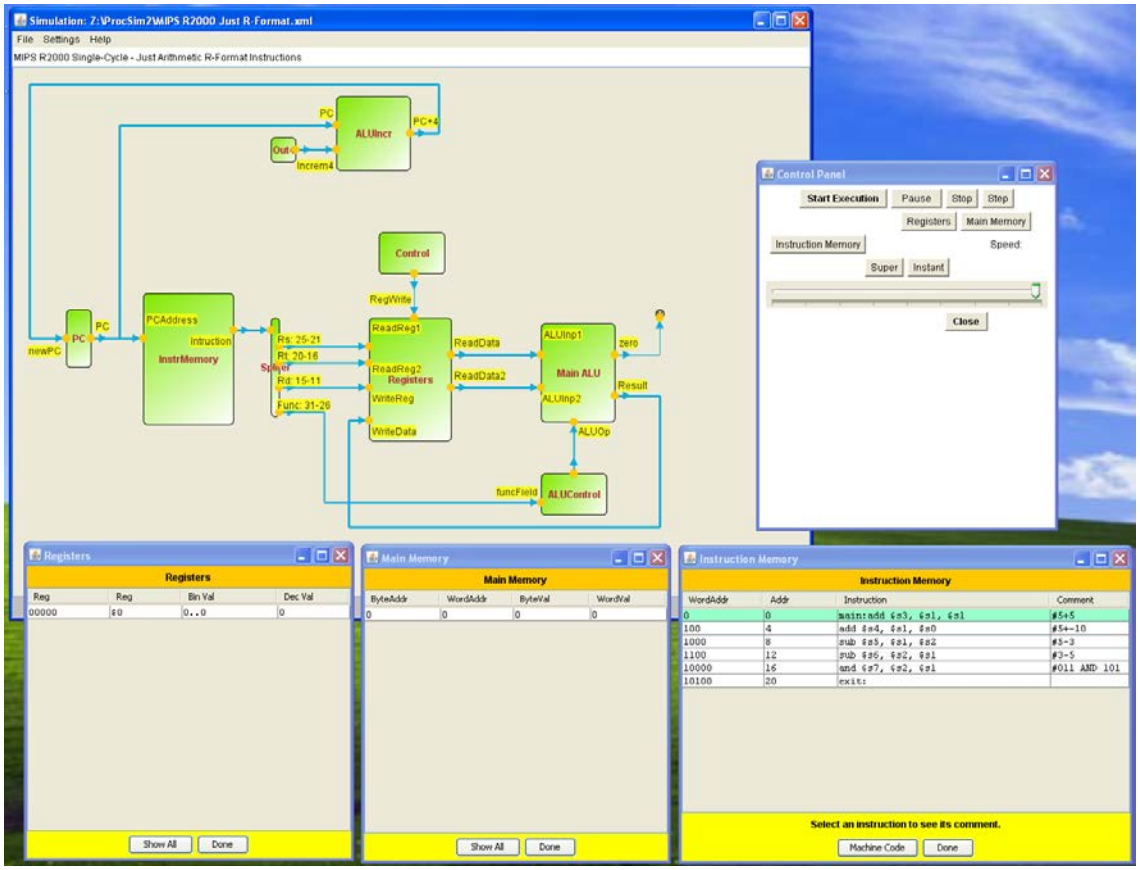


Ilustración 3. Diálogo de selección y edición de la ruta de datos a simular.

Las rutas de datos disponibles son subconjuntos de la ruta de datos completa vista en las sesiones de teoría, a pesar de que algunos de los componentes reciben nombres distintos.

Selecciona el fichero de código ensamblador "sample just R-Format.asm", ensambla el código (botón *Assemble*), cierra el cuadro de diálogo (botón *Done*) y guarda el cambio (*Save Changes)*. Seguidamente, selecciona la ruta de datos "MIPS R2000 Just R-Format" y a continuación cierra la correspondiente ventana (botón *Done*). Esta ruta de datos muestra los componentes para la ejecución de instrucciones con formato de tipo R.

Una vez seleccionados el código a simular y la ruta de datos sobre la que se ejecuta, desde el diálogo principal podemos lanzar el simulador haciendo clic en el botón "*Start Simulation*". Nos aparecen en ese momento diferentes diálogos, algunos de ellos los podemos activar con los botones en el "*Control Panel*". La siguiente figura muestra los diferentes diálogos con la ruta de datos, el banco de registros, la memoria de datos (*Main Memory*), y la memoria de instrucciones con el código ensamblado.



Actividad: Identifica y familiarízate con los diferentes componentes de la ruta de datos. Recuerda que la ruta de datos utiliza solamente los componentes que se utilizan al ejecutar instrucciones de tipo "R". Los componentes son:

- Memoria de instrucciones, *InstrMemory*.
- Contador de programa, *PC*.
- Banco de registros, *Registers*.
- Unidad aritmético-lógica, *Main ALU*.
- Unidad de control, formada por dos componentes: Control y ALUControl.
- ALU para el autoincremento del contador de programa, *ALUIncr*.

Todos estos componentes están interconectados formando la ruta de datos. En concreto tenemos conexiones entre la Memoria de instrucciones, el contador de programa y la ALU de autoincremento para el registro PC. A su vez, tenemos el banco de registros conectado directamente con la ALU. Identifica en el diálogo de la ruta de datos los diferentes componentes y cómo se encuentran interconectados entre sí.

Una vez cargado el simulador ya podemos empezar a ejecutar el código en ensamblador instrucción a instrucción. Para cada instrucción se visualiza el avance de todas las señales de control y de datos por la ruta de datos. La ejecución se controla a través del panel de control (*Control Panel*) con los botones:

- *Start Execution*. Realiza la ejecución de todas las instrucciones mostrando de forma animada el avance de todas las señales de control.
- *Pause/Resume*. Realiza una pausa (*pause*) o reanuda la simulación (*resume*).
- *Stop*. Finaliza la simulación.

- *Step*. Avanza la simulación un paso, representando el avance de un subconjunto de señales de datos y de control. Para ejecutar una instrucción por completo debemos realizar varios pasos.
- *Speed*. Podemos modificar la velocidad de la simulación con los botones y la barra asociada.

Actividad: Utilizando el simulador describe los nueve pasos en los que se ejecuta la primera instrucción del código (`add $s3, $s1, $s1`). Indica para cada paso cuáles son las acciones realizadas.

Paso	Acciones sobre la ruta de datos
1	Carga el PC con 0 y lo lleva al ALUIncr y a la instruccion de memoria
2	Carga la instruccion de memoria codificada y la envia al Spliter
3	La unidad de control envia un 1 a RegWrite
4	El spliter envia los datos de la instrucción a cada registro y carga la función
5	El ALUControl con el dato de la función carga 010 (suma) en la muxALU
6	El bando de registros (BR) carga los registros y los envia a la ALU (para sumar)
7	Envia zero a la salida superior y el resultado a la entrada de escritura del BR
8	En el out se envia una señal al ALUIncr cargando el valor anterior del PC
9	El ALUIncr le suma 4bytes al valor anterior del PC y lo lleva a PC

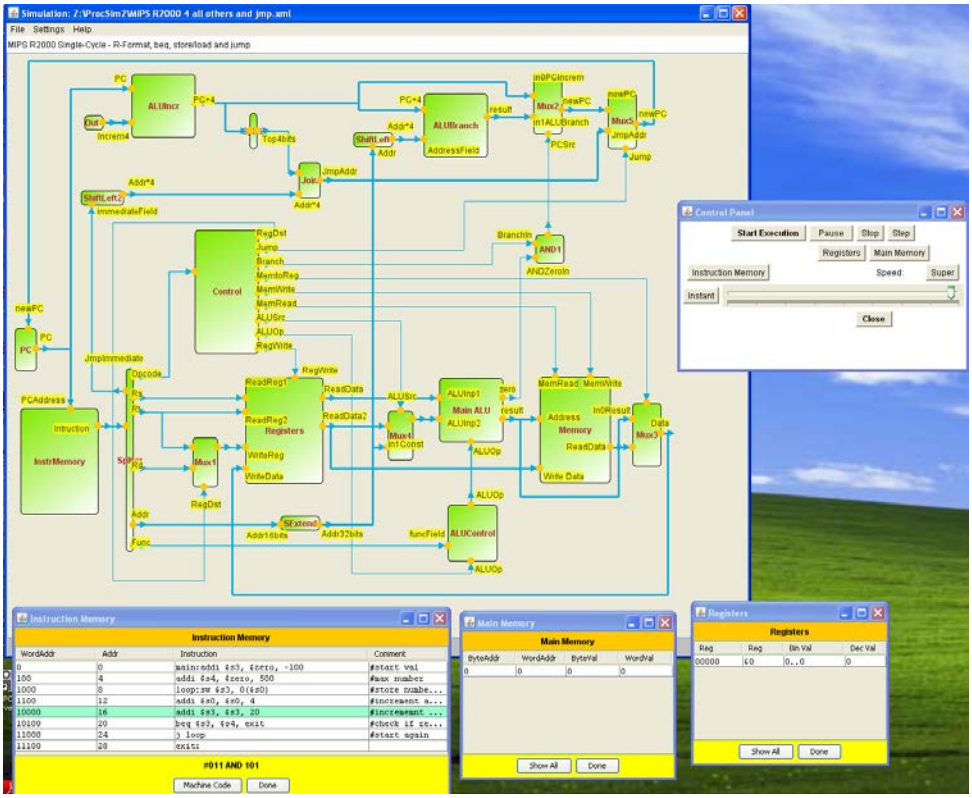
Nótese que estamos ante una ruta de datos monociclo, en el que todas las acciones correspondientes a la ejecución de una instrucción tendrían lugar en realidad en el mismo ciclo de reloj. El orden en que se muestran las acciones en el simulador es arbitrario y simplemente pretende indicar de forma legible la activación de las señales de control y el camino que siguen las instrucciones por la ruta.

Actividad: Realiza la simulación completa de las instrucciones restantes, paso a paso, e identifica en la tabla siguiente los valores de las señales indicadas:

Instrucción	Rs	Rt	Rd	ALUOp	ReadData	ReadData2	Result	Zero
add $s4, $s1, $s0	1 0 0 0 1	1 0 0 0 0	1 0 1 0 0	0 1 0	5	-10	-5	0
sub $s5, $s1, $s2	1 0 0 0 1	1 0 0 1 0	1 0 1 0 1	1 1 0	5	3	2	0
sub $s6, $s2, $s1	1 0 0 1 0	1 0 0 0 1	1 0 1 1 0	1 1 0	3	5	-2	0
and $s7, $s2, $s1	1 0 0 1 0	1 0 0 0 1	1 0 1 1 1	0 0 0	3	5	1	0

4. Ejecución de instrucciones de tipo I y de salto

Una vez trabajada la ruta de datos para instrucciones de tipo R, ahora nos centramos en la ruta de datos completa, la cual soporta también instrucciones de tipo I e instrucciones de tipo J. Para ello, cierra la simulación (botón *Close*) y desde el diálogo principal carga el código "sample addi" (recuerda ensamblar el código con el botón "*Assemble*") y la ruta de datos "MIPS R2000 5 all jmp and addi". Una vez cargados haz clic en el botón "Start Simulation". De nuevo nos aparecen las ventanas con la ruta de datos (con más componentes en este caso), el panel de control y los valores almacenados en el banco de registros, la memoria de instrucciones y la memoria de datos. Redimensiona los diálogos para poder ver toda la información de una manera cómoda, tal como se muestra en la siguiente figura.



En esta ruta de datos aparecen muchos más componentes y ahora el simulador mostrará la ejecución de cada instrucción en 21 pasos. Como elementos adicionales tenemos:

- Multiplexores:
    - Mux1, multiplexor que selecciona el registro destino en el banco de registros (denominado MxDST o RegDst en la ruta de datos de teoría).
    - Mux3, multiplexor que selecciona el resultado a escribir en el banco de registros (denominado MxER o MemToReg en la ruta de datos de teoría).
    - Mux4, selecciona la entrada de la UAL (denominado MxUAL2 o ALUSrc en la ruta de datos de teoría).
    - Mux2 y Mux5, multiplexores que seleccionan el nuevo valor del registro PC (denominado MxPC o PCMux en la ruta de datos de teoría).
- ALUBranch, calcula la dirección de salto de las instrucciones de salto condicional.
- Operador de desplazamiento (Shiftleft), realiza un desplazamiento de dos bits a la izquierda (multiplicación por cuatro).
- Operador Join, une dos señales de entrada, componiendo un bus que concatena ambas entradas.
- Operador AND, realiza una operación AND lógica sobre las dos señales de entrada.
- El operador Splitter ya aparecía en la anterior ruta de datos y sirve para separar los campos de un bus, en nuestro caso la instrucción actual.

Actividad: Ejecuta las instrucciones paso a paso y escribe el valor de las señales de control (generadas por la unidad de control) para cada instrucción.

Instrucción	RegDst	Jump	Branch	MemtoReg	MemWrite	MemRead	ALUSrc	ALUop	RegWrite
addi $s3, $zero, -100	0	0	0	0	0	0	1	11	1
addi $s4, $zero, 500	0	0	0	0	0	0	1	11	1
sw $s3, 0($s0)	0	0	0	0	1	0	1	00	0
addi $s0, $s0, 4	0	0	0	0	0	0	1	11	1
addi $s3, $s3, 20	0	0	0	0	0	0	1	11	1
beq $s3, $s4, exit	0	0	1	0	0	0	0	01	0
j loop	0	1	0	0	0	0	0	00	0

Como se puede observar, en las instrucciones de salto condicional (`beq`) y de salto incondicional (`j`), la ruta de datos sigue un camino distinto para el cálculo de la nueva dirección del PC. En la figura siguiente podemos ver la lógica de la ruta de datos para el cálculo de la dirección de salto.

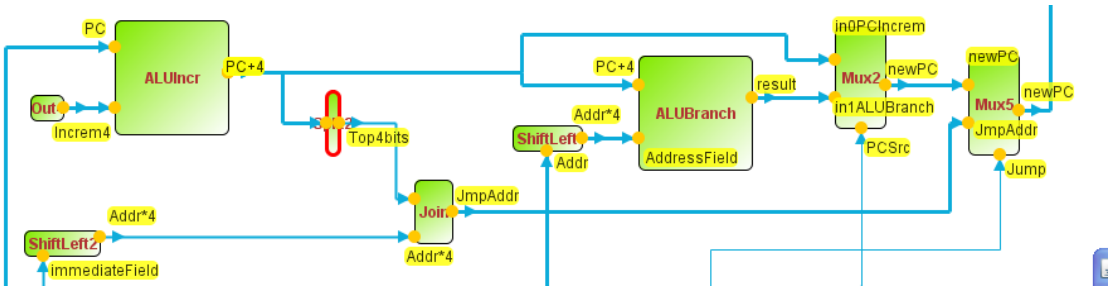


Ilustración 4. Parte de la ruta de datos para el cálculo del nuevo contador de programa.

Actividad: Indica los valores de entrada y salida para los elementos siguientes, tanto para la instrucción `beq` como para la instrucción `j`.

Instrucción	Señales	Valor
Instrucción `beq`	PC+4	0...11000(32 bits)
	Addr32bits (salida SExtend)	1
	Addr*4 (salida shiftleft)	4
	result (salida ALUBranch)	0...11100(32 bits)
	PCSrc (salida AND1)	0
	newPC (salida Mux2)	0...11000(32 bits)
	Jump (entrada Mux5)	0
	newPC (salida Mux5)	24
Instrucción `j`	JmpImmediate	0...10(26 bits)
	Addr*4 (salida ShiftLeft2)	0...1000(28 bits)
	Top4bits	0000(4 bits)
	JmpAddr	0...1000(32 bits)
	Jump (entrada Mux5)	1
	newPC (salida Mux5)	8

Actividad: Justifica el valor de dichas señales, indicando el porqué de esos valores en función de la dirección de salto de las instrucciones.

1- En el caso de la instrucción beq carga el valor PC en AluIncr, saliendo como resultado PC+4, dicho valor llega a MUX2 y como PCSrc es 0 el resultado de MUX2 es PC+4 que ahora pasará a llamarse newPC y dicho valor llega a MUX5 y al ser Jump = 0, la salida de MUX5 es newPC.

2- En el caso de la instrucción jump tenemos JumpInmediate que ocupa los últimos 26 bits de la intrucción y estos van hacia ShiftLeft2, donde se les agregan 2 bits al final, ocupando 28 bits y ahora pasará a llamarse Addr*4, dicho valor llega a Join donde se le agregan los primeros 4 bits del PC+4 y dando como resultado JumpAddr. En MUX5 se tiene en la segunda entrada el valor de JumpAddr y al ser Jump=1, JumpAddr pasa a ser newPC.