Implementación Procesador RISC-V

Román F. Muñoz
Facultad de Ingeniería y Ciencias Hidricas- Universidad Nacional del Litoral

***Resumen*—En el siguiente informe se describe la consigna del trabajo práctico integrador, asi como se detalla la implementación realizada. A grandes rasgos la misma se basa en un datapath, que es el encargado de comunicar los distintos modulos del procesador. Estos modulos son: El Contador de Programa, la Memoria de Instrucciones, el El Banco de Registros, la Extensión de Signo, la Unidad Aritmetico-Lógica y la Memoria de Datos. Además se cuenta con multiplexores y sumadores al servicio de administrar el flujo de la información en el sistema. Y el pipeline es acompañado por una unidad de control. Esta se compone de dos decodificadores, uno principal, encargado de configurar las señales de selección en los multiplexores y las de escritura en las memorias, y otro dedicado a la unidad aritmetico-lógica, seleccionando la operacion a realizar.**

I. Introducción

La actividad consistia en simular un programa de assembly en la implementación en verilog del procesador RISC-V realizada en las clases de práctica. RISC es la abreviación de Reduced Instruction Set Computing. Se trata de una arquitectura de conjunto de instucciones abierta y libre.

El procesador monociclo en cuestion se compone de un datapath (camino de datos) y una unidad de control.

La implementación soporta seis tipos de instricciones: **lw (lectura de memoria)**, **sw(escritura en memoria)**, **R(operaciones aritmeticas-lógicas)**, **beq (ramificar si es igual)**, **addi (sumar un valor inmediato)** y **jal (salto)**.

II. Camino de Datos

A continuacioón se detallan los modulos construidos:

PC: Tiene como inputs la señal de reloj (*clk(*, un reset, el valor inicial (*pcInput*) y el siguiente valor del contador (*pcNext*). Cada ciclo de reloj actualiza el valor del contador de programa. Si la señal de reset se encuentra en alto entonces se asigna el valor de *pcInput*, es decir se resetea. Su salida es el contador de programa.

IM: Recibe la dirección de la instrucción (*adressIm*) en la memoria de instrucciones, esta última se define como un arreglo de 32 palabras de 32 bits. Su salida es la instrucción seleccionada.

Aqui se carga en codigo el programa a ejecutar.

Las instrucciones se obtuvieron dumpeando en memoria el programa de assembly. Se selecciono el formato hexadecimal.

BR: Recibe la señal de reloj, los indices de los dos registros a leer (**a1** y **a2**), el indice del registro a escribir (**a3**), el valor a escribir (**wd3**) y la señal de escritura (**we**). Se inicializa el registro en 0 y se asegura que no se sobreescriba este valor. Cada ciclo de reloj, si la señal *we* se halla en alto, se escribe *wd3* en la posición *a3* del banco de registros, este se define de manera identica a la memoria de instrucciones. Las dos salidas (**rd1** y **rd2**) se corresponden a los valores de los registros en las direcciones *a1* y *a2.*

SE: Ingresa al modulo la instruccion sin el campo op (**inm**) de 25 bits y la señal de control (**src**). Esta última determina donde buscar los valores del campo inmediato de acuerdo al tipo de instrucción. Se concatena correctamente los valores recibidos y se realiza la extension de signo correspondiente para generar la salida de 32 bits.

ALU: Recibe los dos operandos (**srcA** y **srcB**) y la señal de control (**ALUControl**). Esta última determina la operación aritmetica o lógica a realizar entre los operandos. El resultado (**result**) es una de las salidas, la otra es una señal **zero** que determina si el resultado es o no zero.

DM: Toma la señal de reloj, asi como la direccion de memoria (**addresDM**), el valor a guardar (**wd**) y la señal de escritura (**we**). Cada ciclo de reloj, si la señal *we* se halla en alto se procede a guardar *wd* en *addresDM* de la memoria de datos. Esta memoria fue definida identicamente al banco de registros y la memoria de instrucciones.

Adder: Recibe dos palabras de 32 bits y las suma, esta suma es la salida. Se emplea en dos ocasiones. Una sumandole cuatro al contador de programa . Otra sumandole una cantidad proveniente de una instruccion de ramificacion o salto.

Mux2x1: Multiplexor de dos entradas, una señal de control y una salida. Se emplea en dos ocasiones. En primer lugar recibiendo la señal *pcSrc* que indica si se debe tomar el pc aumentado en cuatro, es decir curso normal del programa, o si debe realizar un salto o ramificacion tomando la segunda entrada.
En segundo lugar para determinar, dado el valor de *AluSrc*, si el segundo operando de la alu es la salida *rd2* del banco de registros, o bien la salida del modulo de extension de signo.

Mux4x1: Multiplexor de cuatro entradas, una señal de control y una salida. Se emplea para, dado el valor de *resSrc*, determinar si la entrada *wd3* del banco de registros es la lectura de memoria (*rd)*, la salidad de la ALU, o la suma entre pc y 4.

Datapath: Tiene como entradas la señal de reloj, un reset, el valor inicial del pc y las señales provenientes de la Unidad de Control. Este modulo se encarga de definir correctamente todos los modulos anteriores. Presenta detalles que merecen ser mencionados como la señal constante *four* que se cablea al adder *apc.*

En el cableado con la memoria de instrucciones se descartan del contador los dos bits menos signifivos. De este modo se obtiene un avance unitario. La operación es equivalante a dividir el *pc* por 4.

Además a partir de la instruccion se cablean a los modulos las entradas *a1, a2 y a3 ([19:15],[24:20] y [11:7])* del banco de registros, la entrada al modulo de extension de signo *([31:7]).*

Las salidas, tomadas tambien de la instrucción, son la señal *f7, f3 y op ([30], [14:12] y [6:0]).* Se tiene tambien la salida zero que se corresponde a la señal zero de la alu.

III. Unidad de Control

AluDeco: su señales de entrada son *op, f7, f3 y aluOp.* Se encarga de decodificar estas y retornar el codigo de operacion correspondiente a la alu *(ALUControl)*.

MainDeco: su entradas es el codigo de operación *op*. A partir de este genera las señales de control correspondientes a la instruccion asociada al *op.* Las señales de salida son *branch, resSrc, memWrite (we), aluSrc, inmSrc, regWrite (we) y aluOp.*

Unidad de Control: Aqui se definen los dos decodifcadores anteriores. Cabe destacar la lógica correspondiente a la señal pcSrc.

```
45    assign pcSrc = (zero & branch) | resSrc[1];
```

Figura 1: asignación pcSrc.

De este modo, el *pc* obtendra el valor proveniente del segundo sumador cuando sea alto tanto la señal de *branch* como sea zero la resta de la condicion. Ó cuando el valor *resSrc[1]* sea alto, y esto se da así cuando se esta en una instrucción de tipo jal.

Finalmente tanto el Datapath como la Unidad de control se integraron en un modulo superior llamado rv32i.
Este modulo superior recibe la señal de reloj, reset y el valor inicial del pc.

IV. Programa

El programa de prueba provisto cuenta con las seis instrucciones implementadas, se implemento en la Memoria de Instrucciones de manera hexadecimal. Se realizaron correciones pertinentes al hecho que, a diferencia del RISC-V, nuestra implementación cuenta con el inicio de su memoria de datos en la posición 0.

V. Simulacion

Mediante apio y gtkwave se simulo el codigo de verilog. En primer lugar se procedio a verificar el funcionamiento individual de cada modulo para una mejor organización, para eso se realizaron testbenches individuales para cada modulo.

Luego se simulo el procesador en su conjunto, y se estudio si el mismo respondia correctamente ante cada posible instrucción que se programó.

Verificado esto último se simulo el programa brindado y se corroboro que se obtuvieron los resultados esperados a partir de analizar que las distintas señales se correspondieran con la instruccion que se ejecutaba en ese momento, que las lecturas y escrituras se realizaran, y que los valores sean los correctos, es decir, equivalentes a ejecutar el codigo ensamblador en el RARS.