Proyecto de Arquitectura de Computadoras

Curso 2013-2014

Isabel I de Castilla

Introducción

El proyecto consiste en diseñar en LogiSim un procesador que implemente la arquitectura ISABEL-1. El ISA (Instruction Set Architecture) de ISABEL-1 es el siguiente:

```
MOV R1,K
MOV R1,R2
MOV R1,[R2]
MOV [R2],R1
ADD R1,R2,R3
SUB R1,R2,R3
MUL R1,R2,R3
NEG R1,R2
AND R1,R2,R3
OR  R1,R2,R3
XOR R1,R2,R3
NOT R1,R2
JMP K
JZ K
JN K
JMP R1
HALT
```

El procesador debe tener 8 registros nombrados R0, R1, ..., R7 de 8 bits cada uno. En las instrucciones que lo llevan el argumento constante K tiene 8 bits.

El procesador utilizará una RAM de 256 entradas de 1 byte cada una, solo para datos. Y una ROM de 256 entradas de 16 bits cada una, solo para instrucciones. La ROM contendrá una instrucción por entrada. Cada instrucción ocupa, por tanto, 16 bits.

Instrucciones

MOV R1,K

Esta instrucción mueve el valor constante K de 8 bits hacia un registro

La instrucción `MOV RN,K` se codifica así:

OpCode				Registro RN			*	K							
0	0	0	0	N_2	N_1	N_0	0	K_7	K_6	K_5	K_4	K_3	K_2	K_1	K_0

*No se usa

Los 3 bits que aparecen después del OpCode representan el registro destino de la instrucción. Al registro R0 le corresponden los bits 000, a R1 los bits 001, a R2 los bits 010 y así hasta R7, al que corresponden los bits 111.

Por ejemplo:

`MOV R3,18` se codificaría como 0000 011 0 00010010

`MOV R4,-5` se codificaría como 0000 100 0 11111011

MOV R1,R2

Esta instrucción copia el valor almacenado en un registro hacia otro.

La instrucción `MOV RA,RB` se codifica así:

OpCode				Registro RA			Registro RB			*	*	*	*	*	*
0	0	0	1	A_2	A_1	A_0	B_2	B_1	B_0	0	0	0	0	0	0

*No se usa

Ejemplos:

`MOV R3,R1` se codificaría 0001 011 001 000000

`MOV R4,R0` se codificaría 0001 100 000 000000

MOV R1,[R2]

Esta instrucción copia el valor almacenado en la dirección R2 de la RAM hacia el registro R1.

La instrucción `MOV RA,[RB]` se codifica así:

OpCode				Registro RA			Registro RB			*	*	*	*	*	*
0	0	1	0	A_2	A_1	A_0	B_2	B_1	B_0	0	0	0	0	0	0

*No se usa

Ejemplos:

`MOV R3,[R1]` se codificaría 0010 011 001 000000

`MOV R4,[R0]` se codificaría 0010 100 000 000000

MOV [R1],R2

Esta instrucción copia hacia la dirección R1 de la RAM el valor del registro R2.

La instrucción `MOV [RA],RB` se codifica así:

OpCode				Registro RA			Registro RB			*	*	*	*	*	*
0	0	1	1	A_2	A_1	A_0	B_2	B_1	B_0	0	0	0	0	0	0

*No se usa

Ejemplos:

`MOV [R3],R1` se codificaría 0011 011 001 000000
`MOV [R4],R0` se codificaría 0011 100 000 000000

ADD R1,R2,R3
SUB R1,R2,R3
MUL R1,R2,R3
AND R1,R2,R3
OR R1,R2,R3
XOR R1,R2,R3

Estas instrucciones realizan las operaciones lógicas o aritméticas indicadas usando R2 y R3 como argumentos y almacenando el resultado en R1. Estas instrucciones, con argumentos `RA,RB,RC`, se codifican así:

OpCode (debajo)				Registro RA			Registro RB			Registro RC			*	*	*
-	-	-	-	A_2	A_1	A_0	B_2	B_1	B_0	C_2	C_1	C_0	0	0	0

*No se usa

El OpCode para cada una es:

ADD 0100
SUB 0101
MUL 0110
AND 1000
OR 1001
XOR 1010

Ejemplos:

`ADD R7,R4,R7` se codificaría 0100 111 100 111 000
`XOR R0,R1,R2` se codificaría 1010 000 001 010 000

NEG R1,R2

Esta instrucción almacena en R1 el opuesto aritmético de R2 (el complemento a 2). Por ejemplo, tras las dos instrucciones

```
MOV R2,8
NEG R1,R2
```

el registro R1 tiene almacenado 11111000 (-8). La instrucción `NEG RA,RB` se codifica así:

OpCode				Registro RA			Registro RB			*	*	*	*	*	*
0	1	1	1	A_2	A_1	A_0	B_2	B_1	B_0	0	0	0	0	0	0

*No se usa

Ejemplos:

`NEG R3,R3` se codificaría 0111 011 011 000000

`NEG R4,R0` se codificaría 0111 100 000 000000

NOT R1,R2

Esta instrucción almacena en R1 la negación bit a bit de R2 (el complemento a 1). Por ejemplo, tras las dos instrucciones

```
MOV R2,8
NOT R1,R2
```

el registro R1 tiene almacenado 11110111 (-9). La instrucción `NOT RA,RB` se codifica así:

OpCode				Registro RA			Registro RB			*	*	*	*	*	*
1	0	1	1	A_2	A_1	A_0	B_2	B_1	B_0	0	0	0	0	0	0

*No se usa

Ejemplos:

`NOT R3,R3` se codificaría 1011 011 011 000000
`NOT R4,R0` se codificaría 1011 100 000 000000

JMP K

Esta instrucción salta incondicionalmente K instrucciones. El valor de la constante K es relativo a la instrucción actual. K puede ser negativo, lo que provocaría un salto hacia atrás.

Esta instrucción se codifica así:

OpCode				*	*	*	*	K							
1	1	0	0	0	0	0	0	K_7	K_6	K_5	K_4	K_3	K_2	K_1	K_0

*No se usa

Ejemplos:

`JMP  8` se codificaría 1100 0000 00001000
`JMP -8` se codificaría 1100 0000 11111000

JZ K
JN K

Estas instrucciones implementan saltos condicionales. JZ salta si el resultado de la última operación aritmética o lógica fue cero (0). JN salta si el resultado de la última operación aritmética o lógica tiene el bit más significativo en 1. El argumento K tiene la misma interpretación que en la instrucción JMP.

Estas instrucciones se codifican así:

OpCode (debajo)				*	*	*	*	K							
-	-	-	-	0	0	0	0	K_7	K_6	K_5	K_4	K_3	K_2	K_1	K_0

*No se usa

Los OpCodes de estas instrucciones son:

JZ 1101
JN 1110

Ejemplos:

`JZ  8` se codificaría 1101 0000 00001000

`JN -8` se codificaría 1110 0000 11111000

JMP R1

Esta instrucción salta incondicionalmente un número de instrucciones igual al valor del registro R1. Como la instrucción `JMP K`, el valor almacenado en el registro se interpreta relativo a la instrucción actual. Si es negativo se salta hacia atrás.

La instrucción `JMP RA` se codifica así:

OpCode				Registro RA			*	*	*	*	*	*	*	*	*
1	1	1	1	A_2	A_1	A_0	0	0	0	0	0	0	0	0	0

*No se usa

Ejemplos:

`JMP R3` se codificaría 1111 011 000000000

`JMP R7` se codificaría 1110 111 000000000

HALT

La instrucción `HALT` no tiene un OpCode propio y no lleva argumentos. La función de esta instrucción es activar una salida *Stop* en el procesador que indica que la ejecución debe detenerse. Esta salida se activa cuando se hace un salto incondicional de cero (0) instrucciones. Dicho de otra forma, `HALT` es lo mismo que `JMP 0`.

OpCodes

Como referencia, a continuación se listan los OpCodes de todas las instrucciones:

No.	OpCode	Instrucción
0	0000	`MOV R1,K`
1	0001	`MOV R1,R2`
2	0010	`MOV R1,[R2]`
3	0011	`MOV [R2],R1`
4	0100	`ADD R1,R2,R3`
5	0101	`SUB R1,R2,R3`
6	0110	`MUL R1,R2,R3`
7	0111	`NEG R1,R2`
8	1000	`AND R1,R2,R3`
9	1001	`OR R1,R2,R3`
10	1010	`XOR R1,R2,R3`
11	1011	`NOT R1,R2`
12	1100	`JMP K` y `HALT`
13	1101	`JZ K`
14	1110	`JN K`
15	1111	`JMP R1`

Interfaz e Implementación

El procesador debe ser implementado como un subcircuito de LogiSim con nombre ISABEL. En el sitio de la asignatura, y adjunto en este PDF, hay un archivo de LogiSim con dos subcircuitos: *Main Board* e *ISABEL*. El circuito *Main Board* contiene la RAM, la ROM, el reloj y una entrada RESET; e incluye al módulo *ISABEL* que implementa al procesador. El procesador debe tener una salida Stop que se activa si se ejecuta la instrucción `HALT` (`JMP 0`).

La figura 1 muestra la interfaz externa de ISABEL y sus conexiones con la RAM, la ROM y los demás pines de control.

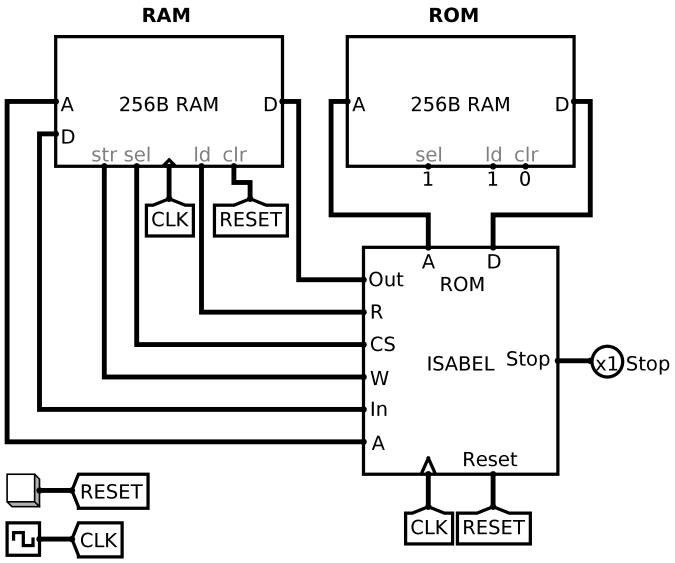

Figura 1: Main Board. Interfaz externa de ISABEL.

La figura 2 muestra la interfaz interna del procesador. Note que los pines circulares o redondeados son de salida y los rectangulares son de entrada. Las salidas de ISABEL son: ROM Addr, RAM Addr, RAM Data In, RAM Chip Select, RAM Read, RAM Write y Stop. Las entradas son: ROM Data, RAM Data Out, Clock y Reset.

El circuito *Main Board* no necesita ser modificado. Toda la implementación del procesador ocurre dentro del subcircuito *ISABEL*, que naturalmente, tendrá sus propios subcircuitos.

Ensamblaje

En el sitio de la asignatura, y adjunto en este PDF, está un sencillo ensamblador de ISABEL implementado en Python 3, que traduce un archivo de texto con código en ISABEL-1 a un archivo de datos de LogiSim con las instrucciones codificadas. Este archivo se debe cargar en la ROM (click derecho sobre la ROM y *Load Image...*) del circuito *Main Board* para comenzar la ejecución de un programa.

Verificación y Evaluación

El proyecto debe realizarse en equipos de no más de dos (2) estudiantes. El plazo para entregar los proyectos es el viernes 25 de abril de 2014. Después de la entrega se realizará una revisión oral con un profesor y los miembros de cada equipo. Se puede entregar y revisarse en cualquier momento antes de la expiración del plazo.

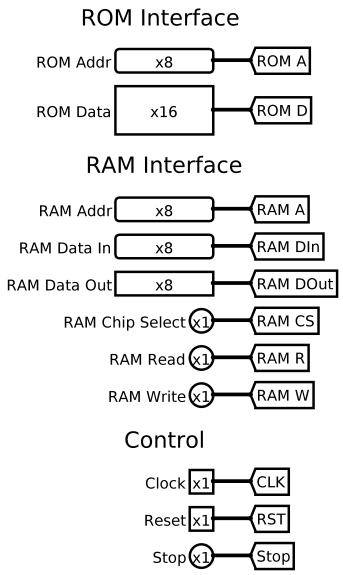

Figura 2: ISABEL. Interfaz interna.

La entrega es el archivo `.circ` descrito aquí, con las modificaciones necesarias al subcircuito *ISABEL* y los subcircuitos adicionales que hagan falta. El nombre del archivo `.circ` debe contener los nombres, apellidos y grupos de los integrantes del equipo.

En la página http://cherry.matcom.uh.cu/isabel-1/ estará disponible un verificador para los procesadores. El verificador recibirá como entrada el archivo `.circ` descrito aquí, con los cambios que se le hayan hecho y ejecutará un conjunto de pruebas sobre el circuito, reportando los resultados.

Pasar todas las pruebas del verificador será un requisito para hacer la revisión oral. El proyecto no se aprueba si el circuito no ha pasado todas la pruebas del verificador antes de o en el día de la entrega.

La nota del proyecto se decide en la revisión oral.