Cartilla de Referencia - `OrgaSmallI`

Procesador `OrgaSmallI`

`OrgaSmallI` es un procesador diseñado e implementado sobre la herramienta *Logisim*. Cuenta con las mismas características que el procesador **OrgaSmall** más un controlador de interrupciones, nuevas instrucciones y señales:

- Nuevo flag para indicar si las interrupciones están habilitadas (flag_I), que comienza en cero.
- Dirección reservada: 0xFF, donde se almacena la dirección de retorno antes de comenzar la rutina de la interrupción.

¿Qué pasa cuando interrumpen al CPU?

En el caso de un procesador `OrgaSmallI`, si el dispositivo de E/S activa la señal de interrupción y el flag I vale 1, termina de ejecutar la instrucción en curso y realiza **atómicamente** los siguientes pasos:

- Coloca [0xFF] = PC
- Coloca I=0 para evitar que el procesador vuelva a interrumpirse
- Coloca PC = [0x00]
- Activa la señal INTA para indicarle al dispositivo que atenderá su pedido

Luego, comienza a ejecutarse la rutina de atención de la interrupción propiamente dicha.

Instrucciones

Instrucción	CodOp	Formato	Acción
`ADD Rx, Ry`	00001	A	`Rx` ← `Rx + Ry`
`ADC Rx, Ry`	00010	A	`Rx` ← `Rx + Ry + flag_C`
`SUB Rx, Ry`	00011	A	`Rx` ← `Rx - Ry`
`AND Rx, Ry`	00100	A	`Rx` ← `Rx and Ry`
`OR Rx, Ry`	00101	A	`Rx` ← `Rx or Ry`
`XOR Rx, Ry`	00110	A	`Rx` ← `Rx xor Ry`
`CMP Rx, Ry`	00111	A	Modifica *flags* de `Rx - Ry`
`MOV Rx, Ry`	01000	A	`Rx` ← `Ry`
`SIG Rx`	01001	B	`Rx` ← `Rx + 1`
`NEG Rx`	01010	B	`Rx` ← `0 - Rx`
`MIX Rx, Ry`	01011	A	`Rx` ← $B_1A_6B_3A_4B_5A_2B_7A_0$
`STR [M], Rx`	10000	D	`Mem[M]` ← `Rx`
`LOAD Rx, [M]`	10001	D	`Rx` ← `Mem[M]`
`STR [Rx], Ry`	10010	A	`Mem[Rx]` ← `Ry`
`LOAD Rx, [Ry]`	10011	A	`Rx` ← `Mem[Ry]`
`JMP M`	10100	C	`PC` ← `M`
`JC M`	10101	C	Si `flag_C=1` entonces `PC` ← `M`
`JZ M`	10110	C	Si `flag_Z=1` entonces `PC` ← `M`
`JN M`	10111	C	Si `flag_N=1` entonces `PC` ← `M`
`INC Rx`	11000	B	`Rx` ← `Rx + 1`
`DEC Rx`	11001	B	`Rx` ← `Rx - 1`
`SHR Rx, t`	11010	A	`Rx` ← `Rx << t`
`SHL Rx, t`	11011	A	`Rx` ← `Rx >> t`
`STI M`	11100	E	`flag_I` ← `1`
`CLI M`	11101	E	`flag_I` ← `0`
`IRET M`	11110	E	`PC` ← `[0xFF]`, `flag_I` ← `1`
`SET Rx, M`	11111	D	`Rx` ← `M`

Las instrucciones están codificadas en 16 bits. Los primeros 5 bits son el `opcode` de la instrucción, el resto de los bits indican los parámetros. Existen 4 posibles codificaciones de parámetros:

Formato	Codificación
A	`OOOOO XXX YYY-----`
B	`OOOOO XXX ---------`
C	`OOOOO --- MMMMMMMM`
D	`OOOOO XXX MMMMMMMM`
E	`OOOOO --- ---------`

XXX codifica el número del registro X (`Rx`), vale entre 0 y 7.

YYY codifica el número del registro Y (`Ry`) o un inmediato, vale entre 0 y 7.

MMMMMMMM Dirección de memoria o valor inmediato, número de 8 bits.

Las posiciones indicadas con `-` deben ser cero. Las instrucciones de `opcode`: 12, 13, 14, 15 son instrucciones reservadas.

Micro-instrucciones

Los datos almacenados en la memoria de micro-instrucciones corresponden a señales para los distintos componentes del sistema. La siguiente tabla indica a qué bit de micro-instrucción corresponde cada señal.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reset_microOp	load_microOp	IC_inta	IC_cli	IC_sti	DE_loadH	DE_loadL	DE_enOutImm	-	PC_enOut	PC_inc	PC_load	I_microOp	JN_microOp	JZ_microOp	JC_microOp	ALU_OP	ALU_OP	ALU_OP	ALU_OP	ALU_opW	ALU_enOut	ALU_enB	ALU_enA	-	MM_enAddr	MM_load	MM_enOut	RB_selectIndexOut	RB_selectIndexIn	RB_enOut	RB_enIn

La codificación de las operaciones de la unidad artimético lógica es la siguiente:

Código	Operación
0000	Reservada
0001	ADD
0010	ADC
0011	SUB

Código	Operación
0100	AND
0101	OR
0110	XOR
0111	CMP

Código	Operación
1000	SHR
1001	SHL
1010	-
1011	MIX

Código	Operación
1100	cte0x00
1101	cte0x01
1110	cte0x02
1111	cte0xFF

Herramientas

Ensamblador (`assembler.py`)

El ensamblador toma como entrada un archivo de texto con la lista de mnemónicos de instrucciones y genera el código binario del programa. Las instrucciones soportadas son todas las descriptas por la arquitectura.
Además el ensamblador soporta el uso de etiquetas y comentarios. Las etiquetas pueden ser cualquier cadena de caracteres finalizada en ":". Una vez declarada, puede ser utilizada en cualquier parte del código, incluso como parámetro o valor inmediato. Para declarar un comentario, se utiliza el caracter ";". Todo el texto luego de la primera aparición de este será considerado comentario.
El ensamblador también soporta la declaración de valores inmediatos. Para esto se utiliza la palabra reservada `DB` seguida de un valor numérico inmediato. Éste será incluido directamente en el código generado.

Generador de Micro-instrucciones (`buildMicroOps.py`)

El generador de micro-instrucciones toma como entrada un archivo de descripción de señales que respeta la siguiente sintaxis:

```
<binary_opcode>:
<signal_1> <signal_2> ... <signal_n>
...
<signal_1> <signal_2> ... <signal_n>
```

donde `<binary_opcode>` corresponde al valor de *opcode* a codificar en binario y `<signal>` a un nombre de señal. Las señales pueden ser indicadas por el nombre de la misma o como: `<signal>=<x>` donde `x` indica el valor de la señal.
Para señales de más de un bit, como `ALU_OP` se debe utilizar el número entero decimal correspondiente. En particular, para la señal mencionada, se puede utilizar directamente el nombre de la operación en la `ALU`. Si no se indica el valor que debe tomar la señal, se utiliza 1.
Por ejemplo, para la codificación de la instrucción `ADD`:

```
00001: ; ADD
    RB_enOut  ALU_enA  RB_selectIndexOut=0   ; ALU_A := Rx
    RB_enOut  ALU_enB  RB_selectIndexOut=1   ; ALU_B := Ry
    ALU_OP=ADD ALU_opW                       ; ALU_ADD
    RB_enIn   ALU_enOut RB_selectIndexIn=0   ; Rx := ALU_OUT
    reset_microOp
```

Notar que la señal `ALU_OP` es completada con un texto que indica la operación de la ALU a realizar.

Uso

Una vez dentro de *Logisim* se debe cargar el circuito de `OrgaSmallI`. Para cargar un programa, se debe entrar en el circuito `memory` y una vez seleccionada la memoria, utilizar el comando *load* para cargar un nuevo programa.
Para modificar el código de micro-instrucciones, se debe entrar al circuito `controlUnit` y con el mismo procedimiento anterior, cargar el nuevo código de micro-instrucciones. Recordar que el `PC` comienza siempre en cero y apunta a la primera instrucción a ejecutar.

> **Nota:** se deberá guardar la dirección de la rutina de la interrupción en la dirección 0x00 antes de habilitar las interrupciones.