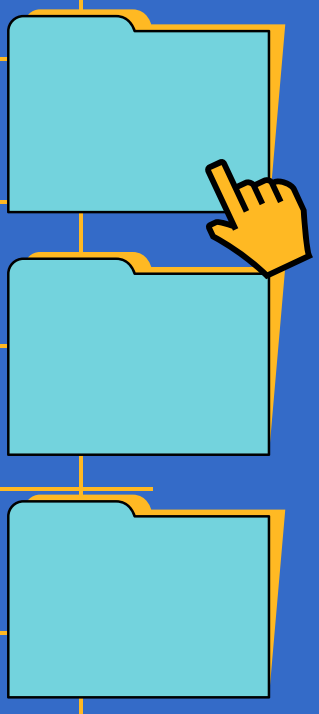
UNIVERSIDAD TORCUATO DI TELLA - ESCUELA DE NEGOCIOS

TRABAJO PRÁCTICO 1

MICROARQUITECTURA EN ORGASMALL



Gonzalo García Vence

Ezequiel Grinblat

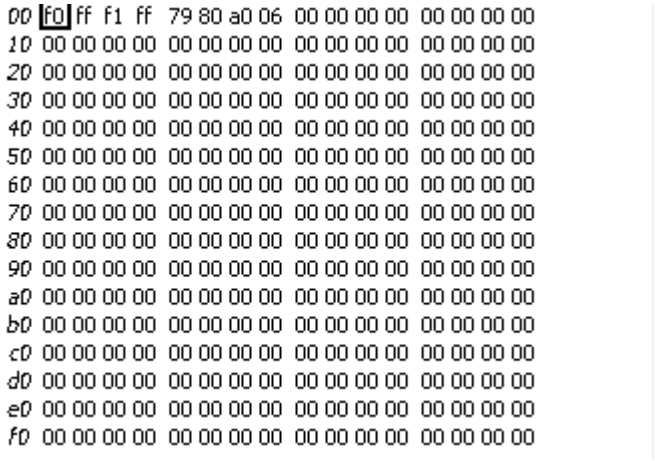
Francisco Frusto Alvarado

Trabajo Práctico 1 “Microarquitectura”:

Introducción:

1.

a. En el caso presentado del trabajo práctico, donde trabajaremos con el procesador Orgasmall, encontramos una memoria de 256 bytes de tamaño (byte = 8 bits). Esta memoria será direccionable a byte.

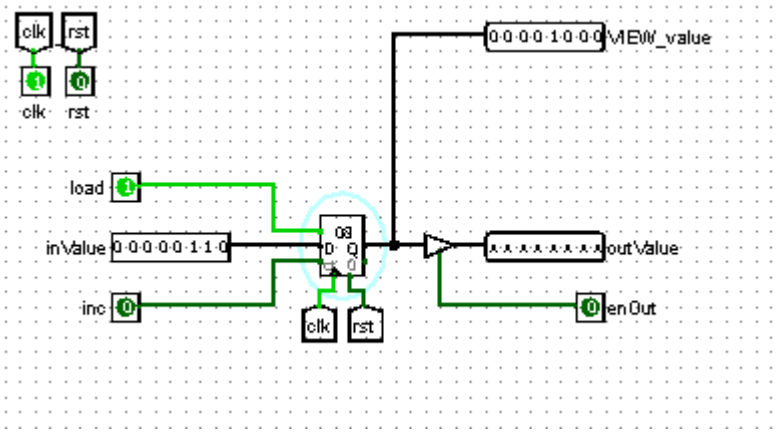


(Captura de pantalla de la memoria representada en Logisim)

b. Como dice la documentación, en Orgasmall tenemos 3 instrucciones de **opcode** libres para crear alguna instrucción a placer. En este caso, son las instrucciones 14, 15 y 31.

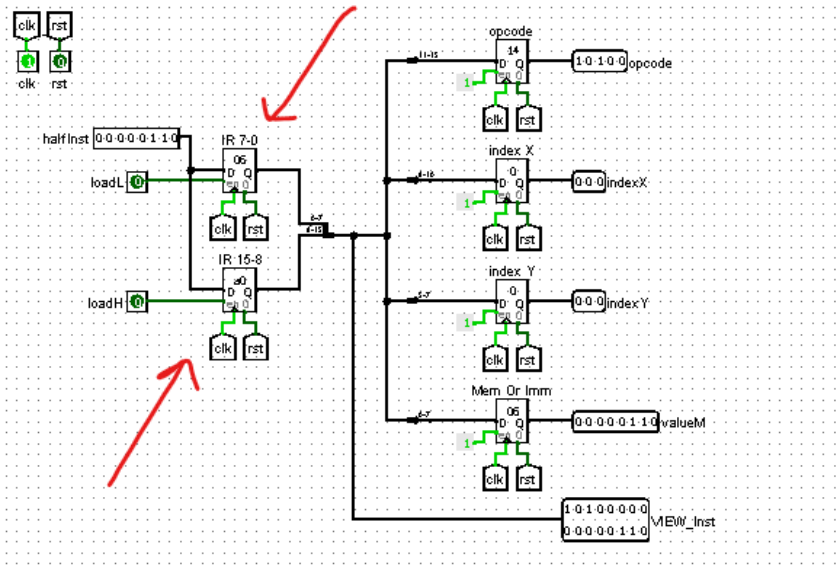
Dirección	Inst.	Dirección	Inst.	Dirección	Inst.	Dirección	Inst.
00000xxxx	fetch	01000xxxx	MOV	10000xxxx	STR	11000xxxx	JO
00001xxxx	ADD	01001xxxx	PUSH	10001xxxx	LOAD	11001xxxx	SHRA
00010xxxx	ADC	01010xxxx	POP	10010xxxx	STR*	11010xxxx	SHR
00011xxxx	SUB	01011xxxx	CALL	10011xxxx	LOAD*	11011xxxx	SHL
00100xxxx	AND	01100xxxx	CALL*	10100xxxx	JMP	11100xxxx	READF
00101xxxx	OR	01101xxxx	RET	10101xxxx	JC	11101xxxx	LOADF
00110xxxx	XOR	01110xxxx	-	10110xxxx	JZ	11110xxxx	SET
00111xxxx	CMP	01111xxxx	-	10111xxxx	JN	11111xxxx	-

c. El **Program Counter** de Orgasmall tiene 8 bits.



(Captura de pantalla del PC representado en Logisim)

d. El **IR** se encuentra en el Decode y está formado por dos registros (IR-L e IR-H) de 8 bits, por lo que su tamaño es de 16 bits.



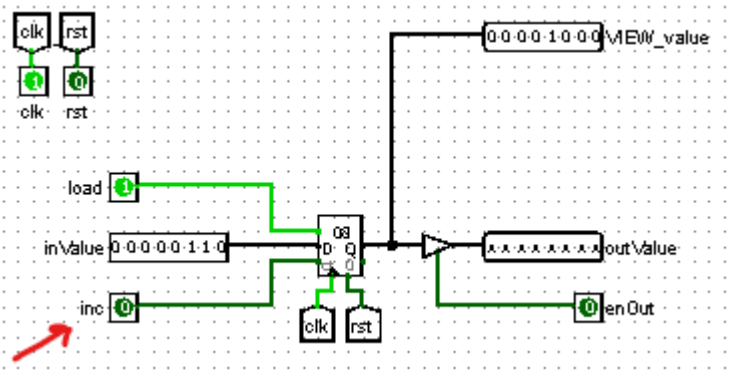
(Captura de pantalla del IR representado en Logisim, las flechas indican tanto el IR como el IL)

e. El tamaño de la memoria de **microinstrucciones** es de 2048 bytes.

Analizar:

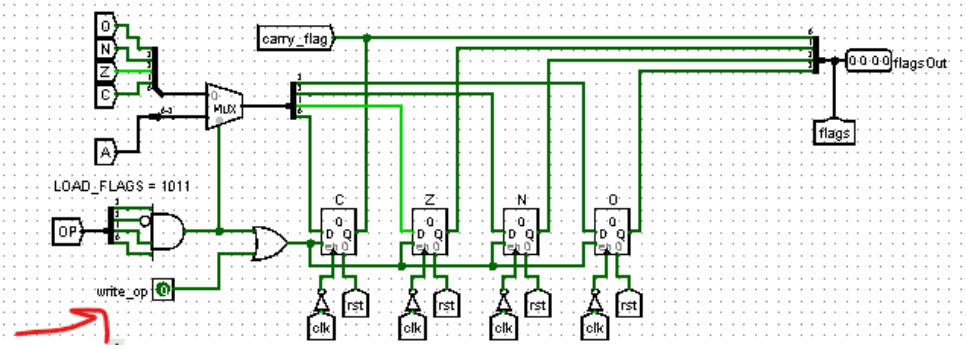
2.

a. La señal **inc** en el PC lo que hace es modificar la dirección de memoria de la siguiente instrucción a ejecutar aumentando su valor en 1.



(Captura de pantalla del PC. La flecha indica la posición de la señal inc)

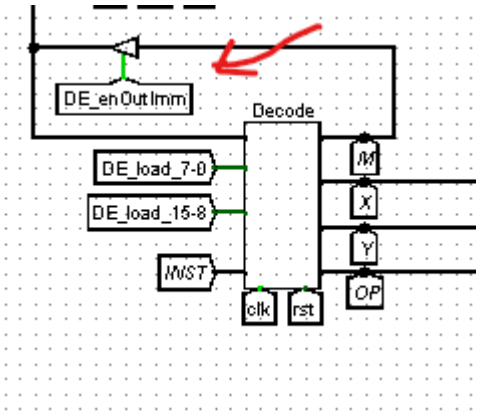
b. La señal **op** va guardando los flags y el **opW** indica si hay que guardarlos.



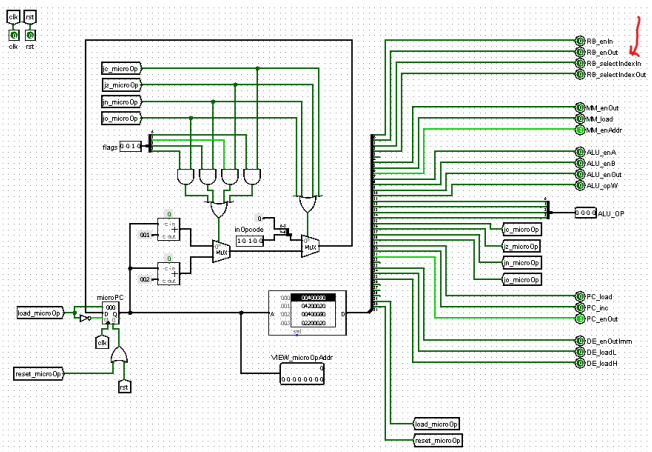
(Captura de pantalla de la ALU. La flecha indica la posición de la señal opW)

c. Los saltos condicionales se realizan con las instrucciones **JZ, JC, JO, JN**. Estas instrucciones modifican el PC al valor que se ponga si el flag de la instrucción (Negative, Overflow, Zero o Carry) está prendido.

d. La señal **DE_enOutImm** permite la escritura de valores inmediatos en el bus. La señal que selecciona el indice del registro es RB_selectIndexIn, ubicada en el UC.



(Captura de pantalla del DataPath, con flecha está indicado la señal **DE enOutImm**)



(Captura de pantalla del UC, con flecha está indicado la señal **RB_selectIndexIn**)

Ensamblar y ejecutar:

3.

a. Lo que hace el programa presentado es que en cada ejecución del ciclo le suma a R0 3 y luego lo va comparando con el valor de R1 que, si bien se modifica momentáneamente durante add3, siempre sigue valiendo 9. En cada comparación, si R0 y R1 son distintos, es decir que la resta no es cero y no se prende el flag de zero, se repite el ciclo. Cuando R0 y R1 valen ambos 9, este flag se activa, por lo que salta a fin y luego sigue en halt y el programa se queda en loop. Hasta llegar a este loop, el ciclo da 3 “vueltas”.

b.

Etiqueta	Dirección de memoria
inicio:	00
ciclo:	06
fin:	0e
halt:	0e
add3:	10

c. Para que llegue a la instrucción **JMP halt** es necesario que pasen 241 ciclos de clock. Lo calculamos manualmente desde logisim.

d.

- Para la instrucción **ADD**, se ejecutan 4 microinstrucciones

```
00001: ; ADD
    ALU_enA RB_enOut RB_selectIndexOut=0 ; A <- Rx 1
    ALU_enB RB_enOut RB_selectIndexOut=1 ; B <- Ry 2
    ALU_OP=ADD ALU_opW                           3
    RB_enIn RB_selectIndexIn=0 ALU_enOut ; Rx <- Rx + Ry4
    reset_microOp
```

- Para la instrucción **JZ**, se ejecutan 1 microinstrucción dependiendo de que el carry este prendido o no

```
10110: ; JZ
    JZ_microOp load_microOp ; if Z then microOp+2 else microOp+1
    reset_microOp
    PC_load    DE_enOutImm  ; PC <- M
    reset_microOp
```

- Para la instrucción **JMP** se ejecutan 1 microinstrucción

```
10100: ; JMP
    PC_load     DE_enOutImm    ; PC <- M
    reset_microOp
```

e.

```
01001: ; PUSH |Rx|, Ry
    MM_enAddr                   RB_enOut RB_selectIndexOut=0   ; addr <- Rx
    MM_load                     RB_enOut RB_selectIndexOut=1   ; [Rx] <- Ry
    ALU_enA                     RB_enOut RB_selectIndexOut=0   ; A <- Rx
    ALU_enB                     ALU_enOut ALU_OP=cte0x01       ; B <- 1
    RB_enIn RB_selectIndexIn=0  ALU_enOut ALU_OP=SUB           ; Rx <- Rx - 1
    reset_microOp
```

El funcionamiento de la instrucción **PUSH** es la siguiente:

- Guarda el valor de **Rx** en **addr**
- Guarda el valor de **Ry** en la direccion de memoria de **Rx**
- Reduce el valor de **Rx** en 1 con la **ALU**

```
01010: ; POP  |Rx|, Ry
    ALU_enA                     RB_enOut RB_selectIndexOut=0   ; A <- Rx
    ALU_enB                     ALU_enOut ALU_OP=cte0x01       ; B <- 1
    RB_enIn RB_selectIndexIn=0  ALU_enOut ALU_OP=ADD           ; Rx <- Rx + 1
    MM_enAddr                   RB_enOut RB_selectIndexOut=0   ; addr <- Rx
    RB_enIn RB_selectIndexIn=1  MM_enOut                       ; Ry <- [Rx]
    reset_microOp
```

El funcionamiento de la instrucción **POP** es la siguiente:

- Suma 1 al valor del **Rx** con la **ALU**
- Guarda el valor de **Rx** en **addr**
- Guarda el valor de memoria de **Rx** en **Ry**

En conjunto, estas instrucciones nos permiten guardar temporalmente valores en la memoria para no “pisar” los valores de esos registros en futuras operaciones.

```
01100: ; CALL |Rx|, M
    MM_enAddr                   RB_enOut RB_selectIndexOut=0   ; addr <- Rx
    MM_load                     PC_enOut                       ; [Rx] <- PC
    ALU_enA                     RB_enOut RB_selectIndexOut=0   ; A <- Rx
    ALU_enB                     ALU_enOut ALU_OP=cte0x01       ; B <- 1
    RB_enIn RB_selectIndexIn=0  ALU_enOut ALU_OP=SUB           ; RX <- RX - 1
    PC_load                     DE_enOutImm                    ; PC <- M
    reset_microOp
```

El funcionamiento de la instrucción **CALL** es la siguiente:

- Guarda el valor de **Rx** en **addr**
- Guarda el valor de **PC** en la direccion de memoria de **Rx**
- Resta 1 al valor de **Rx**
- Guarda el valor de **M** en **PC**

```
01101: ; RET  |Rx|
    ALU_enA                     RB_enOut RB_selectIndexOut=0   ; A <- Rx
    ALU_enB                     ALU_enOut ALU_OP=cte0x01       ; B <- 1
    RB_enIn RB_selectIndexIn=0  ALU_enOut ALU_OP=ADD           ; RX <- RX + 1
    MM_enAddr                   RB_enOut RB_selectIndexOut=0   ; addr <- Rx
    PC_load                     MM_enOut                       ; PC <- [Rx]
    reset_microOp
```

El funcionamiento de la instrucción **RET** es la siguiente:

- Suma 1 al valor de **Rx**

- Guarda el valor de **Rx** en **addr**
- Guarda el valor de la direccion de memoria de **Rx** en la **PC**

En conjunto, estas instrucciones nos permiten cambiar el PC a disposicion temporalmente para luego volver mediante el **RET**