Trabajo Práctico 3

Microarquitectura

Tecnología Digital II

G24 - Wolodarsky Salem Steinmeier

Ejercicios

1. Introducción - Leer la hoja de datos y responder:
    a. ¿Cuál es el tamaño de la memoria en cantidad de bytes?

El tamaño de la memoria en cantidad de bytes es de 256 , direccionable a Byte.

    b. ¿Cuántas instrucciones sin operandos se podrían agregar al formato de instrucción?

Tenemos tres opcode libres, estos son: 01110, 01111 y 11111. Como cada uno de estos opciones es seguido de 11 bits para completar la instrucción, en realidad tenemos 2^11 opciones de instrucción por cada opcode. Finalmente tenemos un total de:

$$2^{11} * 3 \text{ instrucciones} = 6{,}144$$

    c. ¿Qué tamaño tiene el PC?

El tamaño del PC (Program Counter) es de 8 bits de entrada y salida, es decir 1 Byte.

    d. ¿Dónde se encuentra y que tamaño tiene el IR?

El IR se encuentra en el Decodificador, está separado en dos partes: High y Low. Cada uno con un tamaño de entrada de 8 Bits (limitado por el Bus), y salida conjunta de 16 Bits.

    e. ¿Cuál es el tamaño de la memoria de microinstrucciones? ¿Cuál es su unidad direccionable?

Las micro-instrucciones son almacenadas en una memoria en el componente UC. La misma contiene memoria en palabras de 32 bits (2^5) y direcciones de 9 (2^9) bits. Cada una direccionable a Byte. Dándonos un tamaño total de: 2^9 * 2^5 = 2^14 = 16KB

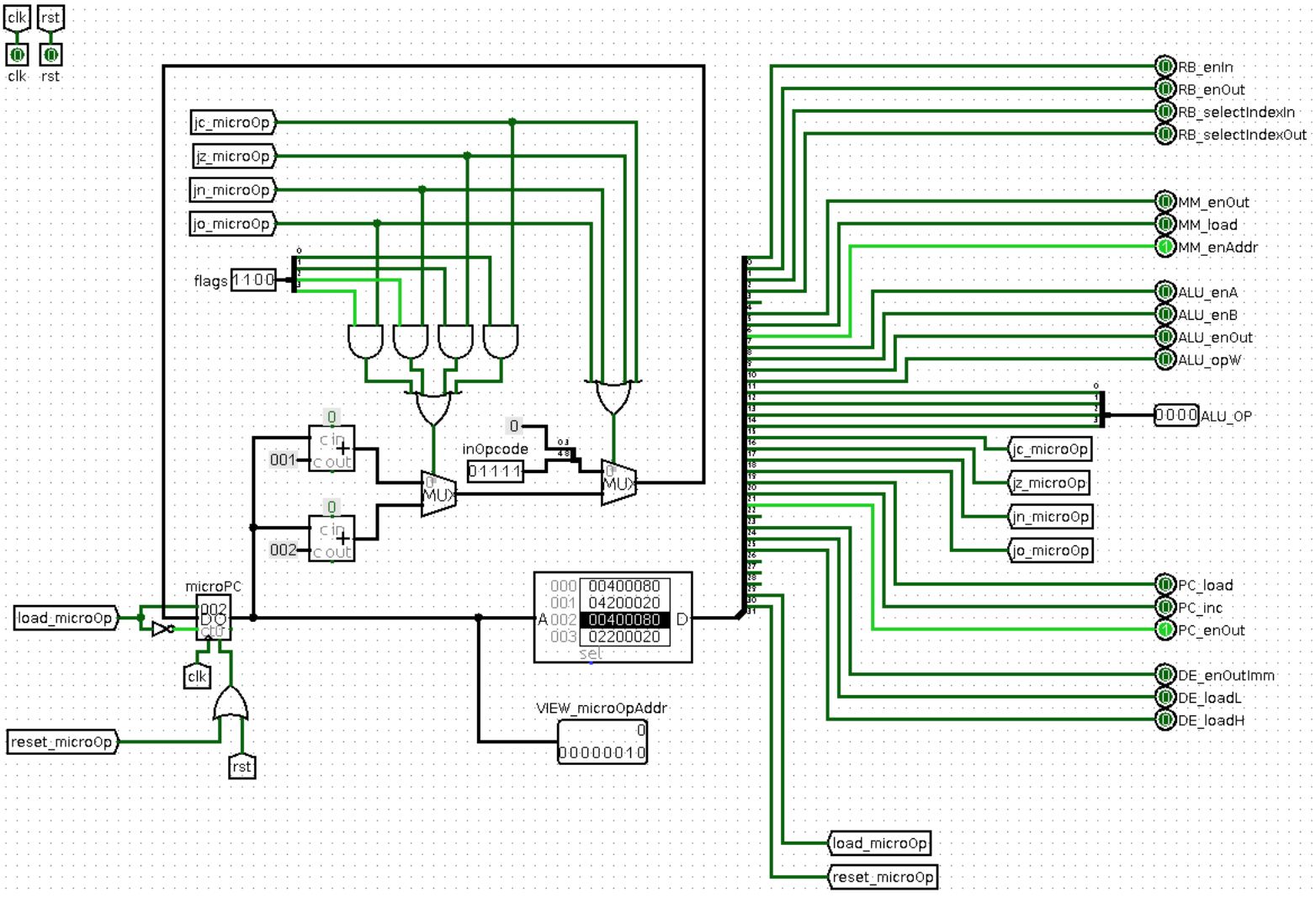
2. Analizar - Estudiar el funcionamiento de los circuitos indicados y responder las siguientes preguntas:
    a. PC (Contador de Programa): ¿Que función cumple la señal **inc**?

La función **inc** sirve para incrementar en 1 el valor actual del Program Counter.

    b. ALU (Unidad Aritmético Lógica): ¿Qué función cumple la señal **opW**?

La ALU toma como entradas operandos e indicaciones y retorna el resultado de la operación junto a sus flag. La señal Opw indica si se deben escribir esos flags que se construyen a partir del estado del número (si es negativo, cero, no representable, etc).

    c. Control Unit (Unidad de control): ¿Cómo se resuelven los saltos condicionales? Describir detalladamente el mecanismo, incluyendo la forma en que interactúan las señales jc_microOp, jz_microOp, jn_microOp y jo_microOp, con los flags.



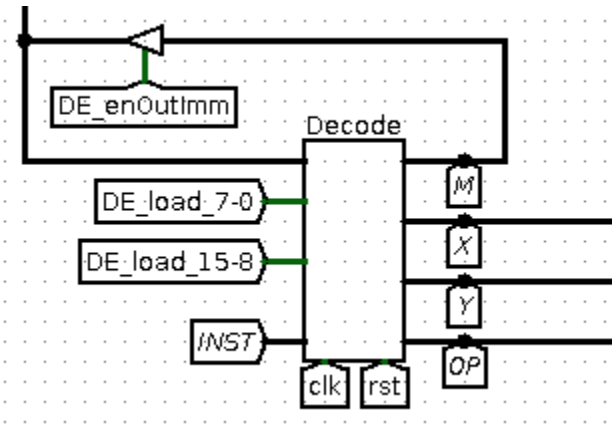
En la Unidad de Control, se resuelven todas las operaciones que tengan que ver con los flags.

Como vemos en la imagen, en la parte superior izquierda se encuentran los tags utilizados para las microoperaciones con los flags, están conectados a dos compuertas, por un lado a una compuerta and, junto a una entrada con los flags activados y desactivados (en este caso vemos que están prendidos los flags de overflow y neg.) esta compuerta and lleva a la entrada de control de un multiplexor, que luego lleva a otro multiplexor, en este caso controlado por las señales las señales jc_microOp, jz_microOp, jn_microOp y jo_microOp. La salida de este multiplexor nos lleva a la entrada de data del Micro PC. Es esta entrada que a través de los distintos multiplexores nos deja hacer los jumps condicionales.

```
10110: ; JZ

        JZ_microOp  load_microOp  ;  if  Z  then  microOp+2  else
microOp+1

    reset_microOp

    PC_load    DE_enOutImm  ; PC <- M

    reset_microOp
```

Como vemos en el código del salto condicional JZ (es muy similar para todos) , primero se prende la jz_microOp, y se carga la micro operación, esto viaja hacia los multiplexores y si logra atravesar ambos, se le sumará un 2 a la microoperación, efectivamente haciendo el salto, de otra manera, se continúa el ciclo.

d. microOrgaSmall (DataPath): ¿Para qué sirve la señal **DE_enOutImm**? ¿Qué parte del circuito indica cuál índice del registro a leer y escribir?



Como vemos en la imagen, la señal DE_enOutImm controla un buffer de tres estados que habilita la entrada y la escritura en el bus de valores desde el decoder. Por otro lado, el Control Unit es el componente que indica cuál índice del registro a leer y escribir con las señales: RB_enIn(habilita la entrada de los registros) y RB_enOut(habilita la salida de los registros.).

3. Ensamblar y ejecutar - Escribir el siguiente archivo, compilarlo y cargarlo en la memoria de la Máquina:
    a. Previamente a ejecutar el programa, describir con palabras el comportamiento esperado del mismo. No se debe explicar instrucción por instrucción, la idea es entender que hace el programa y que resultado genera.

El programa inicia Seteando los registros R7,R0,R1,R2 y R3 a los valores FF(255), 00(0), 50(80), 14(20), 02(2) respectivamente.

Luego inicia el ciclo que se repite hasta que eventualmente el R2 sea Igual al R0 y ahí salta al "fin" donde entra en un loop infinito.

Dentro del “ciclo”se llama a una subrutina “add3” que guardan los registros R0,R1,R3 en la pila .Se guarda en memoria en la ubicación del R1 el valor del R0, se aumentan estos registros en 1 y se vuelve a guardar en memoria en la ubicación del R1 el valor del R0. Finalmente se hace un pop devolviendo los valores originales a R0,R1,R3 y se vuelve al "ciclo".

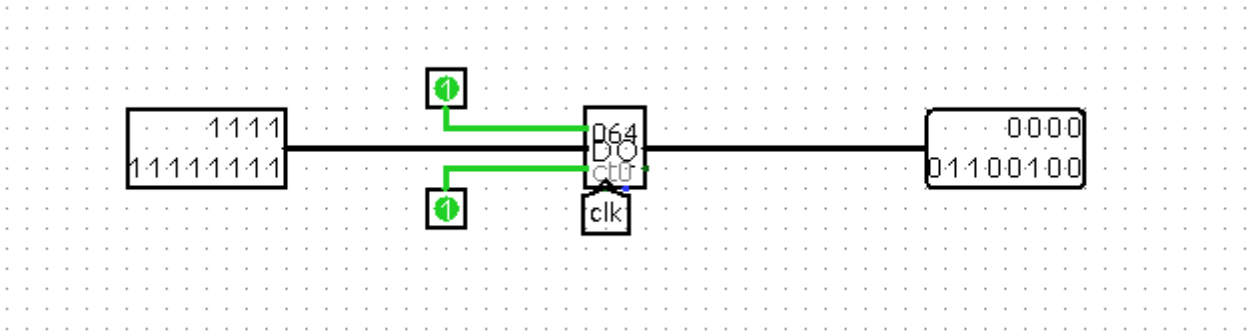
    b. Identificar la dirección de memoria de cada una de las etiquetas del programa.

```
inicio |00| SET R7 , 0xFF
       |02| SET R0 , 0x00
       |04| SET R1 , 0x50
       |06| SET R2 , 0x14
       |08| SET R3 , 0x02
 ciclo |0a| CALL | R7 | , add3
       |0c| ADD R1 , R3
       |0e| ADD R0 , R3
       |10| CMP R0 , R2
       |12| JZ fin
       |14| JMP ciclo
fin, halt |16| JMP halt
   add3 |18| PUSH | R7 | , R3
       |1a| PUSH | R7 | , R1
       |1c| PUSH | R7 | , R0
       |1e| STR [ R1 ] , R0
```

|20| SET R3 , 0x01
|22| ADD R1 , R3
|24| ADD R0 , R3
|26| STR [R1] , R0
|28| POP | R7 | , R0
|2a| POP | R7 | , R1
|2c| POP | R7 | , R3
|2e| RET | R7 |

c. Ejecutar e identificar cuántos ciclos de clock son necesarios para que el programa llegue a la instrucción JMP halt.

Para este ejercicio empleamos un circuito contador como el que vemos en la captura posterior. Este es seteado con 12 bits en 1 al comienzo del ciclo, y luego el mismo va descontando del lado derecho con cada clock. De esta manera pudimos calcular la diferencia entre el lado derecho y el izquierdo al final del ciclo, y esto nos dio: 1587 clocks.



PS:(La imagen no representa los 1587 clocks.)

d. ¿Cuántas microinstrucciones son necesarias para ejecutar la instrucción ADD? ¿Cuantas para la instrucción JZ? ¿Cuántas para la instrucción JMP?

5 microinstrucciones.

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

10110: JZ : 4 MicroInstrucciones
JZ_microOp load_microOp ; if Z then microOp+2 else microOp+1
reset_microOp
PC_load DE_enOutImm ; PC <- M
reset_microOp

10100: JMP : 2 MicroInstrucciones
PC_load DE_enOutImm ; PC <- M
reset_microOp

e. Describir detalladamente el funcionamiento de las instrucciones PUSH, POP, CALL y RET.

Push: La instrucción “Push” tiene dos parámetros, el primero es |Rx| que es la dirección de memoria donde está actualmente el tope de la pila y el otro es Ry que es el dato que se quiere guardar. Por lo tanto lo que hace la instrucción es guardar el dato Ry en la ubicación que indica el Rx y posteriormente restarle 1 al registro Rx para que apunte al próximo lugar libre de la pila.

Pop: La instruccion “Pop” tiene dos parametros, el primero es |Rx| que es la dirección de memoria donde está actualmente el tope de la pila y el otro es Ry que es el registro al cual le queremos poner el valor del dato guardado en la dirección del Rx. Consecuentemente, una vez transferido el dato al registro, se procede a sumarle 1 al registro Rx para que apunte al nuevo último dato de la pila. Vale aclarar que el dato que se transfirio al registro no fue borrado de la pila, pero al pasar a estar por encima de donde apunta el registro Rx, ese dato paso a valer “Basura”.

Call: La instrucción “Call” tiene dos parámetros, el primero es |Rx| que es la dirección de memoria del próximo lugar libre de la pila, el otro es Ry que es la ubicación de la memoria de la próxima instrucción que queremos ejecutar. Cuando llamamos a la instrucción call lo que se hace es guardar el la dirección Rx la dirección de memoria que está actualmente en el PC (La dirección de la próxima instrucción en memoria). Luego se le resta 1 al Rx para que pase a apuntar al próximo lugar libre de la pila y por último se

setea el PC con el valor de Ry asi en el próximo “Fetch” salte directamente a la instrucción indicada.

Ret: El Ret toma un solo parámetro, el |Rx|. Lo que hace el ret es simplemente sumarle 1 al |Rx| y guardar en el PC el último valor de la Pila así la próxima instrucción a ejecutar es la la que habíamos guardado el call.

El funcionamiento de todas las instrucciones proviene de una estructura en memoria administrada por el procesador llamada **Stack.** Esta pila se utiliza para guardar el contexto de ejecución de una función o datos y contiene ciertas instrucciones que utiliza para llevar a cabo su función. La instrucción PUSH guarda un dato en la pila, la instrucción POP quita un dato de la pila, la instrucción CALL salta a una subrutina y guarda la dirección de retorno en la pila y la instrucción RET toma de la pila la dirección de retorno y regresa a la rutina. Las instrucciones CALL y RET permiten llamar a funciones mientras que PUSH y POP salvan información de la pila.

4. Programar - Escribir en ASM las siguientes funciones:
   a) Escribir la función mft3 que calcula dos expresiones distintas dependiendo de una condición.

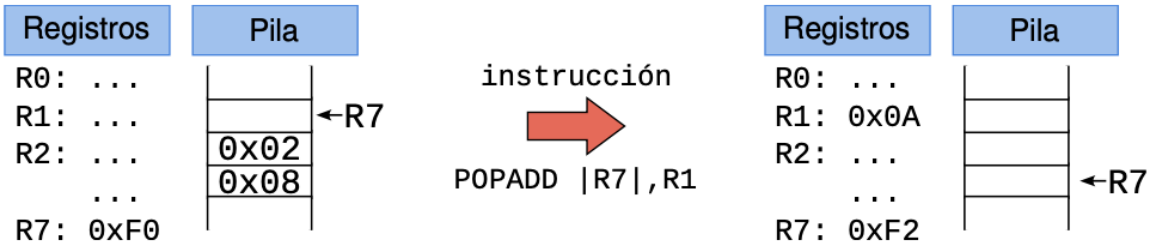
Adjunto en la entrega.

   b) Escribir la función swapping que invierte la posición de los datos dentro de un arreglo de números a partir de una dirección de memoria *mem*.

Adjunto en la entrega.

5. Ampliando la máquina - Agregar las siguientes nuevas instrucciones:
   a. Sin agregar circuitos nuevos, agregar la instrucción POPADD que toma dos números almacenados en la pila, los suma y retorna el resultado en un registro. Esta instrucción debe alterar los flags según la operación que se realizó y dejar la pila consistente, es decir, quitando dos datos de la pila. Se recomienda utilizar como código de operación el 0x0F.

Adjunto en la entrega.



   b. Agregar la instrucción FLIPBITS, que intercambia los bits dentro de un registro, almacenando el resultado en el mismo registro. Para implementar esta instruccion se debe modificar el circuito de la ALU para la operación 14. Bajo este código se debe agregar la operación de la ALU que realiza el intercambio de bits. Se recomienda utilizar como código de operación el 0x0E. La forma de intercambiar bits debe respetar el siguiente orden:

Adjunto en la entrega.