

**Tecnología Digital 2: Sistemas de Computación - TP1**

**Autores:** Arechaga, Facundo

Ahn, Peter

Bergman, Franco

**2. Ensamblar y ejecutar -** Escribir el siguiente archivo, compilarlo y cargarlo en la memoria de la máquina:

1. Previamente a ejecutar el programa, describir con palabras el comportamiento esperado del

mismo. No se debe explicar instrucción por instrucción, la idea es entender qué hace el programa y qué resultado genera.

El programa realiza un bucle principal en la etiqueta “aca” en el cual compara los valores de los registros “R3” y “R2”. Si son iguales, se ejecuta el código en la etiqueta “fin”. Mientras “R3” y “R2” no sean iguales, se llama a “coso2” que ejecuta un conjunto de instrucciones, guardando un valor en memoria, y luego se incrementa el valor de “R2” hasta en algún momento alcanzar el valor de “R3”. Luego de entrar a la etiqueta “fin”, se realiza un JMP al inicio del programa, repitiéndose infinitamente, haciendo que nunca se llegue al halt.

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

Teniendo en cuenta que las instrucciones ocupan 2 bytes de memoria (exceptuando la instrucción DB que solo ocupa 1 byte):

* main: 0x00
* aca: 0x0a
* coso2: 0x14
* fin: 0x20
* halt: 0x22

1. Ejecutar e identificar de ser posible cuantos ciclos de clock son necesarios para que el programa llegue a la instrucción JMP halt.

El programa nunca llega al *Halt*. Esto debido a que cuando el programa llega a “fin” se hace un JMP (mediante la instrucción DB) al principio del programa (repitiéndose nuevamente).

Por lo tanto, no hay una cantidad de ciclos de clock, dado que nunca se llega a esa sección del programa, pero sí podemos calcular cuantos ciclos de clock se tarda en llegar a “fin”:

Aproximadamente, se ejecutan 1226 microinstrucciones, lo cual equivale a 1226 ciclos de clock (para llegar a “fin”)

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

* **ADD:** requiere 4 microinstrucciones
* **JZ:** requiere 2 microinstrucciones
* **JMP:** requiere 1 microinstrucción

1. ¿El programa utiliza la pila?, ¿Qué datos son almacenados en la pila?

Si, el programa utiliza la pila, específicamente cuando hace la instrucción CALL |R7|, donde almacena en ésta, el valor de R7, el cual es 0xFF. Luego al hacer la instrucción PUSH |R7|, R0, se almacena el valor de R0 en la pila como último valor (primero de salida).   
Pero el programa vuelve a hacer función de la pila al llamar la instrucción POP |R7|, R0, devolviéndole el valor (0x00) a R0. Y para finalizar la instrucción RET |R7| le devuelve su valor (0xFF) al registro.

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

PUSH:

La instrucción PUSH utiliza a R7 cómo SP (por convención) y almacena en la pila el valor de R0 en la dirección de memoria apuntada, luego R7 apunta a la siguiente posición disponible de pila, decrementando su valor.

POP:

Básicamente hace lo contrario a la instrucción PUSH. Esta instrucción actualiza el puntero de pila, “volviendo para atrás”, incrementando, para luego recuperar el valor de pila almacenado previamente. En este caso el de R0, para volver a cargarlo en el mismo registro

CALL:

La instrucción CALL sirve para llamar a una función o una subrutina, saltando automáticamente la dirección de ésta misma. Pero lo primero que hace es guardar la dirección de la siguiente instrucción al CALL como dirección de retorno en la pila, para que, al terminar la subrutina, volver exactamente a donde se había quedado el programa.

En el caso de nuestro programa, al utilizar el CALL, la PC apunta a la subrutina *coso2*,y al terminar, vuelve a donde se quedó para pasar a la instrucción ADD R2, R1.

RET:

La instrucción RET sirve para regresar o volver de una subrutina que fue llamada por un CALL, utilizando la dirección de retorno almacenada en la pila para poder continuar el programa. El puntero, en ese caso R7, se incrementa para poder retornar la dirección almacenada haciendo que R7 vuelva a tener su valor original y que la dirección se cargue en el PC

**3. Ampliando la máquina -** Agregar las siguientes nuevas instrucciones:

01110: ; ADDINMEM Rx

    ALU\_enA     RB\_enOut RB\_selectIndexOut=0    ; A <- Rx

    ALU\_enB     ALU\_enOut ALU\_OP=cte0x01        ; B <- 1

    MM\_enAddr   ALU\_enOut ALU\_OP=ADD            ; addr <- Rx + 1

    ALU\_enA     MM\_enOut                        ; A <- [Rx + 1]

    MM\_enAddr   RB\_enOut RB\_selectIndexOut=0    ; addr <- Rx

    ALU\_enB     MM\_enOut                        ; B <- [Rx]

    MM\_load     ALU\_enOut ALU\_OP=ADD            ; [Rx] <- [Rx + 1]

    reset\_microOp

01111: ; ADDE10S Rx, Ry

    ALU\_enA RB\_enOut RB\_selectIndexOut=0                ; A <- Rx

    ALU\_enB RB\_enOut RB\_selectIndexOut=1                ; B <- Ry

    ALU\_OP=ADD

    ALU\_enA ALU\_enOut                                   ; A <- Rx + Ry

    ALU\_enB ALU\_enOut ALU\_OP=cte0x0a                    ; B <- 10

    ALU\_OP=SUB ALU\_opW

    RB\_enIn RB\_selectIndexIn=0 ALU\_enOut                ; Rx <- [Rx + Ry] - 10

    JO\_microOp load\_microOp                             ; if O then microOp+2 else microOp+1

    reset\_microOp

    RB\_enIn RB\_selectIndexIn=0 ALU\_enOut ALU\_OP=cte0xFF ; Rx <- FF

    reset\_microOp

**4. Programar -** Escribir en ASM las siguientes funciones:

main:

    SET R7, 0xFF    ;stack

    SET R0, p       ;p

    SET R1, 0x10    ;size

    PUSH |R7|, R2

    PUSH |R7|, R1

    PUSH |R7|, R0

    SET R2, 0x02

    CALL |R7|, processArray

    POP |R7|, R0

    POP |R7|, R1

    POP |R7|, R2

halt:

    JMP halt

processArray:

    ADDINMEM R0

    ADD R0, R2

    SUB R1, R2

    CMP R1, R2

    JN fin

    JMP processArray

fin:

    RET |R7|

p:

DB 0x01

DB 0x02

DB 0x04

DB 0x08

DB 0x03

DB 0x03

DB 0xA1

DB 0xC0

DB 0xFF

DB 0x40

DB 0x55

DB 0xCC

DB 0xBD

DB 0x45

DB 0x9A

DB 0xEE