Práctica 2

# Objetivo

Identificar la organización de una computadora de propósito general, para comprender su estructura y funcionamiento, analizando la interconexión de sus componentes básicos, con una actitud crítica, propositiva y visionaria.

# Desarrollo

Responda los siguientes cuestionamientos acerca del simulador MARIE.

1. Complete la Tabla 1 describiendo con sus propias palabras las acciones que realizan cada una de las instrucciones del simulador.

|  |  |
| --- | --- |
| **Instrucción** | **Descripción** |
| Add X | Suma el valor encontrado en la dirección “x” al AC y lo guarda en AC (Acumulador) |
| Subt X | Resta el valor encontrado en la dirección “x” al AC y lo guarda en AC |
| AddI X | Suma el valor encontrado en la dirección que indica el valor encontrado en “x” al AC y lo guarda en AC |
| Clear | “Borra” el contenido encontrado en el acumulador (AC = 0) |
| Load X | Carga el valor encontrado en la dirección “x” a AC |
| Store X | Guarda el contenido de AC a la dirección “x” |
| Input | Solicita una entrada al usuario |
| Output | Imprime el valor de AC |
| Jump X | Salta a la dirección “x” |
| Skipcond (C) | Dependiendo del valor en C se salta la dirección que le procede (C = 800) Salta si AC es mayor a 0 (C = 800) Salta si AC es igual a 0 (C = 000) Salta si AC es menor a 0 |
| JnS X | Guarda la dirección de la instrucción que le procede después de que fue llamada y salta a la dirección “x” |
| JumpI X | Salta a la dirección “x” y la guarda en el PC (Contador de programa) |
| StoreI | Guarda AC en la dirección que indica el valor encontrado en “x” |
| LoadI | Carga el valor encontrado en la dirección que indica el valor encontrado en “x” |
| Halt | Detiene el procesador |

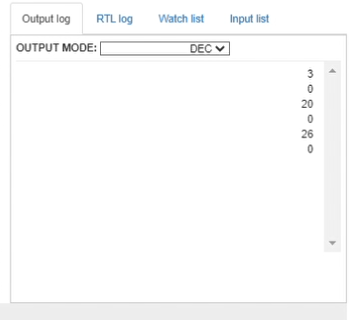
Tabla 1. Conjunto de instrucciones de MARIE.

1. Complete la Tabla 2 indicando la función de cada uno de los registros.

|  |  |
| --- | --- |
| **Registro** | **Función** |
| MAR | Carga o guarda los valores en la dirección indicada |
| PC | Guarda la posición actual de la instrucción |
| MBR | Guarda los valores cuando son transferidos desde o hacia memoria |
| AC | Guardar temporalmente los datos para uso inmediato del procesador |
| IN | Para introducir valores |
| OUT | Para mantener los valores introducidos por el usuario |
| IR | Para mantener los valores desplegados en la salida |

Tabla 2. Registros de MARIE.

1. ¿Qué tamaño en bits tiene el bus de direcciones? **12 bits**
2. ¿Qué tamaño en bits tiene el bus de datos? **16 bits**
3. ¿Qué tamaño en bits tiene el código de operación (opcode) de las instrucciones?  **16 bits**
4. ¿Cuál es la dirección máxima de memoria que se puede acceder? **4095 (DEC) o FFF (HEX)**
5. ¿Por qué el código de operación de los registros es de 3 bits? **Porque son 7 registros y con esos 3 bits se puede acceder a cada uno de ellos**
6. ¿Por qué el registro MAR es de 12 bits? **Porque guarda la dirección (12 bits)**
7. ¿Por qué el registro MBR es de 16 bits? **Porque guarda los datos (16 bits)**
8. Escriba un programa que contenga la subrutina **División**, la cual recibe dos números en las variables **A** y **B** y almacena en la variable **R** el resultado de **A/B**. En el código principal, solicite al usuario dos números y despliegue la división del primero entre el segundo.

**El programa creado pide dividendo, divisor y en resultados muestra el cociente así como el residuo**

**15 / 5 = Cabe 3 veces sin dejar residuo**

**100 / 5= Cabe 20 veces sin dejar residuo**

**234 / 9 = Cabe 9 veces sin dejar residuo**

Input /Obtenemos el dividendo

Store x

Input /Obtenemos el divisor

Store y

JnS XYDiv

Input /Obtenemos el dividendo

Store x

Input /Obtenemos el divisor

Store y

JnS XYDiv

Input /Obtenemos el dividendo

Store x

Input /Obtenemos el divisor

Store y

JnS XYDiv

Halt

/Subrutina para obtener dos numeros y dividirlos

XYDiv, HEX 000

JnS Div /Ejecutamos la division

Load Cicle /Obtenemos el cociente de la division

Output

Load Rem /Obtenemos el residuo

Output

Clear /Reseteamos variables a 0

Store Temp

Store Rem

Store Cicle

JumpI XYDiv /Terminamos subrutina

/Subrutina division

Div, HEX 000

Load x

Store Temp /Usamos la variable temp para evitar modificar x

Loop, Subt y /Restamos el divisor

Store Temp

Load Cicle

Add One /Aumentamos de valor el cociente

Store Cicle

Load Temp

Skipcond 800 /Si es mayor que 0

Jump Less

Jump Loop /Seguir restando

Less, Skipcond 000 /Si es menor que 0

JumpI Div

Add y /Nos pasamos, asi que regresamos un ciclo

Store Rem /Guardamos el residuo

Load Cicle

Subt One /Restamos el ciclo que se hizo de mas

Store Cicle

JumpI Div /Acaba subrutina

One, dec 1

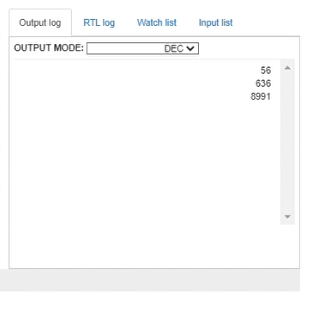
Cicle, dec 0

Rem, dec 0

Temp, dec 0

x, dec 0

y, dec 0

1. ****Escriba un programa que contenga la subrutina Multiplicar, la cual recibe dos números en las variables A y B y almacena en la variable R el resultado de A\*B. En el código principal, solicite al usuario dos números y despliegue la multiplicación del primero por el segundo. Grabar un video corto donde ejecutan al programa con 3 multiplicaciones consecutivas, no finalicen el programa, sino que pongan algo como lo siguiente, de esto toman capturas de pantalla para su reporte y grabaran el video de esa pequeña prueba, en el video debe visualizarse correctamente como funciona su programa y los valores que este regresa. A, B, C y D deben ser números enteros positivos diferentes entre sí.

**7\*8 = 56**

**156\*4 = 636**

**999\*9 = 8991**

/Multiplicación

/X\*Y

//Llamar a la subrutina multiplicar 3 veces

Input

Store x

Input

Store y

JnS Multiply

Input

Store x

Input

Store y

JnS Multiply

Input

Store x

Input

Store y

JnS Multiply

Halt

/Subrutina para obtener dos numeros y mutiplicarlos

Multiply, HEX 000

JnS AddXYTimes /Ejecutamos la multiplicacion

Load xtemp /Obtenemos el resultado

Output

Clear /Reseteamos variables a 0

Store xtemp

Store ytemp

JumpI Multiply /Terminamos subrutina

/Subrutina multiplicacion

AddXYTimes, HEX 000

Load y /Usamos variables ytemp y xtemp

Store ytemp /para evitar modificar 'y' y x

Loop, Load xtemp /Iniciamos la sumatoria en 0

Add x

Store xtemp

Load ytemp /ytemp sera el contador del numero de ciclos a ejecutar

Subt One

Store ytemp

Skipcond 800 /Mientras ytemp sea mayor que 0, seguimos en el ciclo

JumpI AddXYTimes /Terminar ciclo

Jump Loop /repetir ciclo

One, dec 1

xtemp, dec 0

ytemp, dec 0

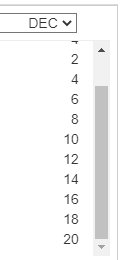
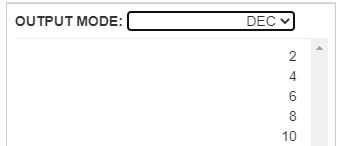
x, dec 0

y, dec 0

1. Escriba un programa que contenga la subrutina **Pares**, la cual despliega en pantalla los primeros **N** números pares. **N** es un número ingresado por el usuario en el código principal.

**Pares**

**5 2 10**

********

//Pares

Input

Store x

JnS NPairs

Input

Store x

JnS NPairs

Input

Store x

JnS NPairs

Halt

NPairs, HEX 000

load x

loop, Skipcond 800

Jump end /Terminar subrutina

Load temp

Add TWO

Output /Imprimir pares

Store temp

Load x

Subt ONE

Store x

Jump loop /Loopear subrutina

end, Clear

Store temp

JumpI NPairs

temp, DEC 0

ONE, DEC 1

TWO, DEC 2

x, DEC 0

1. El plan de una compañía celular incluye 100 minutos de llamadas nacionales y 70 mensajes de texto por $80 al mes. Cada minuto adicional de llamadas nacionales cuesta

$1.00, cada mensaje adicional $2.00 y las llamadas internacionales se cobran a $3 el minuto. Escriba un programa que solicite al usuario tres números que representan los minutos usados en llamadas nacionales e internacionales, y los mensajes enviados.

Finalmente, calcule y despliegue el total a pagar.

|  |  |  |  |
| --- | --- | --- | --- |
| Minutos Nacionales | Mensajes Enviados | Minutos Internacionales | Total |
| 50 | 50 | 0 | **100** |
| 150 | 50 | 0 | **130** |
| 50 | 100 | 0 | **140** |
| 50 | 50 | 100 | **380** |
| 150 | 100 | 0 | **190** |
| 150 | 50 | 100 | **430** |
| 50 | 100 | 100 | **440** |
| 150 | 150 | 150 | **740** |
| 0 | 0 | 0 | **80** |

Input

Store national

Input

Store messagesSent

Input

Store international

Clear

Add plan

Store total

JnS AddNational

JnS AddMessages

JnS AddInternational

Load total

Output

Clear

Store total

Halt

AddMessages, HEX 000

Load messagesSent

Subt mesLimit

Store messagesSent

loopMes, Skipcond 800

JumpI AddMessages

Load total

Add two

Store total

Load messagesSent

Subt one

Store messagesSent

Jump loopMes

AddNational, HEX 000

Load national

Subt minLimit

Store national

loopNat, Skipcond 800

JumpI AddNational

Load total

Add one

Store total

Load national

Subt one

Store national

Jump loopNat

AddInternational, HEX 000

Load international

loopInt, Skipcond 800

JumpI AddInternational

Load total

Add three

Store total

Load international

Subt one

Store international

Jump loopInt

one, dec 1

two, dec 2

three, dec 3

minLimit, dec 100

mesLimit, dec 70

plan, dec 80

total, dec 0

national, dec 0

international, dec 0

messagesSent, dec 0

min, dec 100

mes, dec 70

# Conclusiones y comentarios

Marie es una manera muy práctica de aprender el funcionamiento interno de un procesador y para irse familiarizando con el lenguaje ensamblador. Es bastante amigable y su uso de instrucciones (opcodes) limitado te ayuda a ver lo problemas desde otro ángulo y resolverlos de distintas formas.

# Dificultades en el desarrollo

La única dificultad que se me presentó al desarrollar los códigos fue el uso del opcopde skipcond

Esta instrucción es una instrucción condicional, gracias a esto puede ser convertida en un if o en while, o en cualquier cosa que puedas hacer funcionar utilizando la lógica.

**Anexos**

Carpeta a Drive con los videos y códigos fuente de los programas

<https://drive.google.com/drive/folders/18EcQWiGSVOoYgY2xSwAGJJZYy5xaLyak?usp=sharing>