Práctica 2

Interconexión de Elementos en la Organización de una Computadora de Propósito General

---

Luis Eduardo Galindo Amaya
1274895

Asignatura	Organización de Computadoras (331)
Docente	Arturo Arreola Alvarez
Fecha	2022-08-29

Interconexión de Elementos en la Organización de una Computadora de Propósito General

Luis Eduardo Galindo Amaya
1274895

2022-08-29

Cuestionario

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[1].
4. ¿Cuál es la dirección máxima de memoria que se puede acceder?
   - El bus de memoria es solo de 12 bits por lo que la maxima posicion de memoria accesibles es $2^12 = 4096$ bits[2].
5. ¿Por qué el registro `MAR` es de 12 bits?
   - `MAR` es el acronimo de "Memory Address Registerz sirve para acumular la posicion memoria que se vaya a operar, por lo tanto esta conectada al bus de direcciones, y el bus de direcciones es de 12 bits [1].
6. ¿Por qué el registro `MBR` es de 16 bits?
   - El `MBR` esta conectado al bus de datos y solo contiene el valor del buffer de datos,

[1] Los primeros 4 bits son para la intruccion y los siguientes 12 son para representar direcciones.
[2] 1000 en hexadecimal. En la linea 65 en código de marie.js esta explícitamente definido.

Programa: Lista de Arrays

```
1
2  / ESCRIBA UN PROGRAMA QUE CONTENGA LA SUBRUTINA CAPTURARARREGLO,
3  / LA CUAL PERMITE AL USUARIO CAPTURAR DATOS EN UN ARREGLO EN
4  / MEMORIA. EL PROGRAMA PRINCIPAL DEBE PEDIR AL USUARIO UN NÚMERO
5  / QUE CORRESPONDE AL TAMAÑO DEL ARREGLO. SIGUIENTE, SE DEBE
6  / LLAMAR A SU SUBRUTINA LA CUAL PERMITIRÁ AL USUARIO CAPTURAR
7  / "N" NÚMEROS Y COMENZARÁ A ALMACENARLOS A PARTIR DE LA
8  / DIRECCIÓN 300H, ES DECIR, SI EL USUARIO INGRESA EL NÚMERO 5,
9  / SU SUBRUTINA PEDIRÁ AL USUARIO 5 NÚMEROS Y LOS ALMACENARÁ EN
10 / LAS DIRECCIONES 300H, 301H, 302H, 303H Y 304H.
11
12 / INICIO DEL PROGRAMA ————————————————————————————————
13 INPUT
14 STORE VAR–ARREGLO–SIZE          / CAPTURA EL TAMAÑO DEL ARREGLO
15
16 LOAD CONST–RANGO–INICIO         / CARGA LA POSICION DE INICIO
17 STORE VAR–PTR
18
19 JNS SUB–CAPTURAR–ARREGLO        / CAPTURA LOS DATOS EN EL ARREGLO
20
21 HALT / FINAL DEL PROGRAMA ————————————————————————————
22
23 / SUBRUTINA CAPTURAR ARREGLO ——————————————————————————
24 SUB–CAPTURAR–ARREGLO, HEX 000
25 X–CAPTURAR–ARREGLO–INI, LOAD VAR–ITERADOR
26     SUBT VAR–ARREGLO–SIZE       / RESTA EL TAMAÑO DEL ARREGLO AL
27     SKIPCOND 000                / ITERADOR, SI AC < 0 TERMINA
28     JUMPI SUB–CAPTURAR–ARREGLO
29
30     LOAD CONST–RANGO–INICIO     / OBTIENE LA POSICION ACTUAL DEL
31     ADD VAR–ITERADOR            / PUNTERO Y LO ALMACENA
32     STORE VAR–PTR
33
34     INPUT                       / CAPTURA EL VALOR
35
36     STOREI VAR–PTR              / LO ALMACENA EN LA POSICION DE
37                                 / MEMORIA CORRESPONDIENTE
38
39     LOAD VAR–ITERADOR           / INCREMENTA EL ITERADOR EN 1
40     ADD CONST–UNO
41     STORE VAR–ITERADOR
42     JUMP X–CAPTURAR–ARREGLO–INI
43
44 / CONSTANTES ——————————————————————————————————————
45 CONST–UNO, DEC 1
```

```
46 CONST-RANGO-INICIO , HEX 300
47
48 / VARIABLES ----------------------------------------------------------------
49 VAR-ARREGLO-SIZE , DEC 10
50 VAR-ITERADOR, DEC 0
51 VAR-PTR, HEX 0
```

Capturas

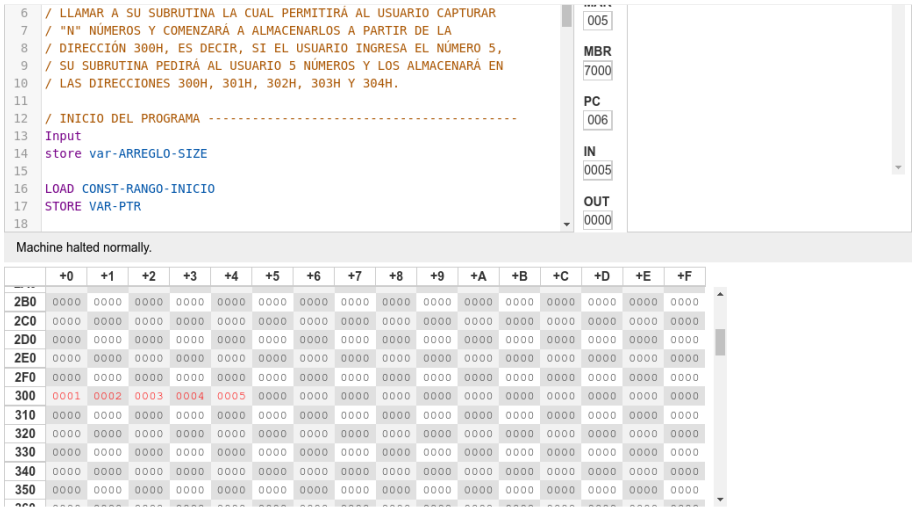

Figura 1: Captura de 5 valores, en color rojo.

Programa: Área Del Triangulo

```
1
2  / ESCRIBA UN PROGRAMA QUE CONTENGA LA SUBRUTINA AREATRIANGULO,
3  / LA CUAL RECIBE DOS NÚMEROS EN LAS VARIABLES B Y H, QUE
4  / REPRESENTAN LA BASE Y LA ALTURA DE UN TRIÁNGULO,
5  / RESPECTIVAMENTE, Y ALMACENA EN LA VARIABLE A EL ÁREA DEL
6  / TRIÁNGULO. EN EL CÓDIGO PRINCIPAL, SOLICITE AL USUARIO DOS
7  / NÚMEROS Y DESPLIEGUE EL ÁREA DE UN TRIÁNGULO CALCULADA CON LOS
8  / DATOS INGRESADOS. REALICE CAPTURAS DE PANTALLA DONDE SE
9  / MUESTRE EL FUNCIONAMIENTO DE LOS PROGRAMAS.
10
11 / INICIO DEL PROGRAMA ----------------------------------------------
12 INPUT
13 STORE VAR-ALTURA              / CAPTURA LA ALTURA
14 OUTPUT
15
16 INPUT
17 STORE VAR-BASE                / CAPTURA LA BASE
18 OUTPUT
19
20 / MULTIPLICACIÓN ---------------------------------------------------
21
22 STORE VAR-ITERADOR            / EL ITERADOR DETERMINA CUANTAS
23                                / VECES SE SUMARA ALTURA
24 JNS SUB-MULT-ALTURA
25 LOAD VAR-ACUMULADOR           / CARGAR EL RESULTADO DE LA
26                                / MULTIPLICACIÓN
27
28 STORE VAR-DIVIDENDO           / GUARDAR EL VALOR EN EL DIVIDENDO
29 OUTPUT
30
31 / REINICIAR LAS VARIABLES ------------------------------------------
32 LOAD CONS-CERO
33 STORE VAR-ITERADOR
34 STORE VAR-ACUMULADOR
35
36 / DIVISION ENTRE DOS -----------------------------------------------
37 JNS SUB-DIV-DOS                / NO OCUPAMOS CARGAR EL VALOR DEL
38                                / DIVIDENDO, YA QUE LO GUARDAMOS
39                                / ANTES
40 LOAD VAR-ITERADOR
41 OUTPUT
42
43 HALT / FINAL DEL PROGRAMA ------------------------------------------
44
45 / SUBRUTINA MULTIPLICACIÓN -----------------------------------------
```

```
46 SUB–MULT–ALTURA, HEX 000       / SUMA "VAR–ALTURA" LA CANTIDAD DE
47                                / VECES QUE DIGA "VAR–ITERADOR" Y
48                                / LO ALMACENA EN "VAR–ACUMULADOR"
49
50 ADD CONS–UNO                   / INCREMENTAR EN 1 A
51                                / "VAR–ITERADOR", ESTO SIMPLIFICA
52                                / LA LÓGICA DEL SKIPCOND, PARA QUE
53                                / SOLO SALTE SI AC ES MAYOR A 0
54 STORE VAR–ITERADOR
55
56 X–MULT–ALTURA–INI, LOAD VAR–ITERADOR
57     SUBT CONS–UNO              / RESTAR UNO AL ACUMULADOR
58
59     SKIPCOND 800               / DETENER LA ITERACIÓN SI AC ES
60     JUMPI SUB–MULT–ALTURA      / MAYOR A 0
61
62
63     STORE VAR–ITERADOR         / ALMACENA EL ITERADOR PARA LA
64                                / SIGUIENTE ITERACION
65     LOAD VAR–ACUMULADOR        / CARGA EL ACUMULADOR
66     ADD VAR–ALTURA             / SUMA "VAR–ALTURA"
67         STORE VAR–ACUMULADOR   / LO ALMACENA EN "VAR–ACUMULADOR"
68                                / Y REPITE HASTA QUE
69                                / "VAR–ITERADOR" ES 0
70     JUMP X–MULT–ALTURA–INI
71
72 / SUBRUTINA DIVISIÓN ————————————————————————————————————
73 SUB–DIV–DOS, HEX 000           / DIVISIÓN ENTERA DE
74                                / "VAR–DIVIDENDO" ENTRE DOS Y LO
75                                / ALMACENA EN "VAR–ITERADOR"
76
77 X–DIV–DOS–INI, LOAD VAR–DIVIDENDO
78     SUBT CONS–DOS              / SE RESTA 2 DE "VAR–DIVIDENDO"
79     SKIPCOND 800               / SI EL RESULTADO DE LA RESTA ES
80     JUMP X–DIV–DOS–TERMINA     / MAYOR A 0 CONTINUA ITERANDO
81
82 X–DIV–DOS–INC–ITN, STORE VAR–DIVIDENDO
83     LOAD VAR–ITERADOR
84     ADD CONS–UNO               / INCREMENTA EL ITINERADOR
85     STORE VAR–ITERADOR
86     JUMP X–DIV–DOS–INI
87
88 / SI LA DIVISIÓN ES EXACTA SE REPITE EL SUBPROCESO PARA
89 / INCREMENTAR EL ITERADOR, AL TERMINAR ESE SUBPROCESO REGRESA
90 / AL INICIO DE "SUB–DIV–DOS" ENTONCES SE VUELVE A RESTAR 2 AL
91 / DIVIDENDO Y AHORA EL NUMERO ES MENOR A 0 Y DIFERENTE A 0 Y
92 / AHORA SI TERMINA EL SUBPROCESO "SUB–DIV–DOS"
93
```

```
X_DIV_DOS_TERMINA, SKIPCOND 400
        JUMPI SUB_DIV_DOS
        JUMP X_DIV_DOS_INC_ITN

/ CONSTANTES ------------------------------------------------------------------
CONS_CERO, DEC 0
CONS_UNO, DEC 1
CONS_DOS, DEC 2

/ VARIABLES -------------------------------------------------------------------
VAR_BASE, DEC 0
VAR_ALTURA, DEC 0
VAR_ITERADOR, DEC 0
VAR_ACUMULADOR, DEC 0
VAR_DIVIDENDO, DEC 0
```

Captura

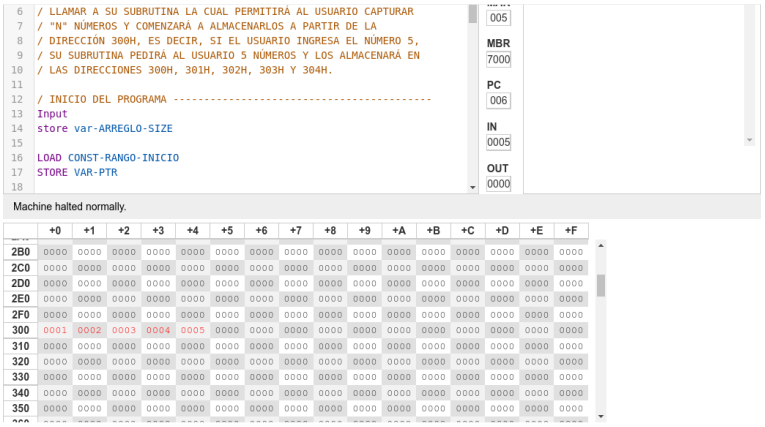

Figura 2: Con base 10, altura 5.

Conclusión

A modo de conclusión me gustaría poner mis reflexiones, Al tener operaciones muy limitadas en marie.js es muy complicado de obtener el valor de una división, se tiene que recurrir a métodos mas sencillos para poder realizar las operaciones más complejas, recuerdo que algunos profesores siempre nos dicen: *"Las computadoras son tontas"* y al programar en assembly, recuerdo lo sencillas que realmente son es nuestro trabajo como programadores expandir las limitadas capacidades de estos dispositivos para que sean realmente útiles.