# Ejercitación manual

Se ha tipeado en un lenguaje de alto nivel

INTEGERS R = P+ Q – T

P=‐227D Q=774D T=‐20D

1) Este punto no se realiza para números enteros.

2a) Para las siguientes variables utilice las mismas direcciones del TPR Nº 5.A

R=2277

P=2279

Q=227B

T=227D

2b) Ídem Trabajo Práctico Nº 5.A

P=‐227 (en decimal) = 1111 1111 0001 1101 (en binario) = FF1D (en hexa)

Q= 774 (en decimal) = 0000 0011 0000 0110 (en binario) = 0306 (en hexa)

T= ‐20 (en decimal) = 1111 1111 1110 1100 (en binario) = FFEC (en hexa)

PARA P:

|  |  |  |
| --- | --- | --- |
| **DEC A BIN** | | |
| **PESA** | **DEC** | **BIN** |
| 2^7 | 128 | 1 |
| 2^6 | 64 | 1 |
| 2^5 | 32 | 1 |
| 2^4 | 16 | 0 |
| 2^3 | 8 | 0 |
| 2^2 | 4 | 0 |
| 2^1 | 2 | 1 |
| 2^0 | 1 | 1 |
| **DEC:** | | **227** |
| **BIN:** | | **0000 0000 1110 0011** |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **227** | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 |  | 1 | 1 | 1 | 0 |  | 0 | 0 | 1 | 1 |
| **-227** | 1 | 1 | 1 | 1 |  | 1 | 1 | 1 | 1 |  | 0 | 0 | 0 | 1 |  | 1 | 1 | 0 | 1 |

|  |  |  |
| --- | --- | --- |
| **BIN NEG A HEXA** | | |
| **BIN** | **DEC** | **HEXA** |
| 1111 | 15 | F |
| 1111 | 15 | F |
| 0001 | 1 | 1 |
| 1101 | 13 | D |
| **ENTERO:** | | **-227** |
| **HEXA:** | | **FF1D** |

PARA Q:

|  |  |  |
| --- | --- | --- |
| **DEC A BIN** | | |
| **PESA** | **DEC** | **BIN** |
| 2^9 | 512 | 1 |
| 2^8 | 256 | 1 |
| 2^7 | 128 | 0 |
| 2^6 | 64 | 0 |
| 2^5 | 32 | 0 |
| 2^4 | 16 | 0 |
| 2^3 | 8 | 0 |
| 2^2 | 4 | 1 |
| 2^1 | 2 | 1 |
| 2^0 | 1 | 0 |
| **DEC:** | | **774** |
| **BIN:** | | **0000 0011 0000 0110** |

|  |  |  |
| --- | --- | --- |
| **BIN POS A HEXA** | | |
| **BIN** | **DEC** | **HEXA** |
| 0000 | 0 | 0 |
| 0011 | 3 | 3 |
| 0000 | 0 | 0 |
| 0110 | 6 | 6 |
| **DECIMAL:** | | **774** |
| **HEXA:** | | **0306** |

PARA T:

|  |  |  |
| --- | --- | --- |
| **DEC A BIN** | | |
| **PESA** | **DEC** | **BIN** |
| 2^4 | 16 | 1 |
| 2^3 | 8 | 0 |
| 2^2 | 4 | 1 |
| 2^1 | 2 | 0 |
| 2^0 | 1 | 0 |
| **DEC:** | | **20** |
| **BIN:** | | **0000 0000 0001 0100** |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **20** | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 1 |  | 0 | 1 | 0 | 0 |
| **-20** | 1 | 1 | 1 | 1 |  | 1 | 1 | 1 | 1 |  | 1 | 1 | 1 | 0 |  | 1 | 1 | 0 | 0 |

|  |  |  |
| --- | --- | --- |
| **BIN NEG A HEXA** | | |
| **BIN** | **DEC** | **HEXA** |
| 1111 | 15 | F |
| 1111 | 15 | F |
| 1110 | 14 | E |
| 1100 | 12 | C |
| **ENTERO:** | | **-20** |
| **HEXA:** | | **FFEC** |

2c) Ídem TPR Nº5.A

P=‐227 (en decimal) = 1111 1111 0001 1101 (en binario) = FF1D (en hexa)

Q= 774 (en decimal) = 0000 0011 0000 0110 (en binario) = 0306 (en hexa)

T= ‐20 (en decimal) = 1111 1111 1110 1100 (en binario) = FFEC (en hexa)

INTEGERS R = P+ Q – T

P=‐227 Q=774 T=‐20

R = –227 + 774 –(–20) = –227 + 774 + 20 = 567D = 0237H

|  |  |  |
| --- | --- | --- |
| MEM | VAR | VAL |
| 2277 | R | 37 |
| 2278 | R | 02 |
| 2279 | P | 1D |
| 227A | P | FF |
| 227B | Q | 06 |
| 227C | Q | 03 |
| 227D | T | EC |
| 227E | T | FF |

2d) Ídem TPR Nº5.A {}… A1

|  |  |
| --- | --- |
| MEM | INS |
| 7420 | A1 |
| 7421 | 79 |
| 7422 | 22 |
| 7423 | 03 |
| 7424 | 06 |
| 7425 | 7B |
| 7426 | 22 |
| 7427 | 2B |
| 7428 | 06 |
| 7429 | 7D |
| 742A | 22 |
| 742B | A3 |
| 742C | 77 |
| 742D | 22 |

2e) Comparar el paso 2d) de este TP con el mismo paso del TP anterior. ¿Qué conclusiones resultan y por qué los códigos de operación son iguales en el TPR Nº5.A y en el TPR Nº 5.B? **(La respuesta no es porque en ambos casos se hace R = P + Q ‐ T y porque las direcciones son las mismas, dado que para reales también se hace la misma suma algebraica, pero los códigos de operación son distintos)**

Comparemos, en términos prácticos, lo que se le pidió al procesador que opere en ambos casos:

En el práctico anterior: P Q T

(227) + (774) – (20)

En este práctico: P Q T

(-227) + (774) – (-20)

Como puede apreciarse, lo que cambia es el valor de las variables PQT, pero también que en el primer caso nosotros “enviamos” naturales a la UAL, y en el segundo, “enteros”. Sin embargo, la UAL en esencia opera con naturales. Es decir, los binarios con bit de signo, en la UAL, se operan como naturales. La UAL no sabe que está operando con enteros. Somos nosotros, nuestro programa, los que interpretamos el resultado haciendo un análisis de los FLAGS. Como puede verse en este trabajo, la UAL siempre genera cuatro indicadores (SZVC) luego de cada operación aritmética, pero nosotros (nuestro programa) tomará en cuenta solo las banderas SVZ si estamos operando con enteros, y solo las CZ si lo estamos haciendo con naturales. Es por eso por lo que, en las capturas de pantalla del DEBUG (que luego veremos) y a los fines de este trabajo (en el que “enviamos” enteros), el SET de CARRY termina no resultando pertinente.

Resumiendo, los códigos de operación son iguales porque, en general, ejecutamos de una misma manera la operación algebraica a pesar de que el tipo de resultado que esperamos no es el mismo. Y se hace así porque, una vez entendido que el procesador, la UAL, tiene una y tan solo una mecánica de operación, es el ingenio humano el que le da uso y lo adapta según el caso.

3) Ídem TPR Nº 5.A, indicando además el valor de los flags SZVC cuando interviene la UAL.

P=‐227 (en decimal) = 1111 1111 0001 1101 (en binario) = FF1D (en hexa)

Q= 774 (en decimal) = 0000 0011 0000 0110 (en binario) = 0306 (en hexa)

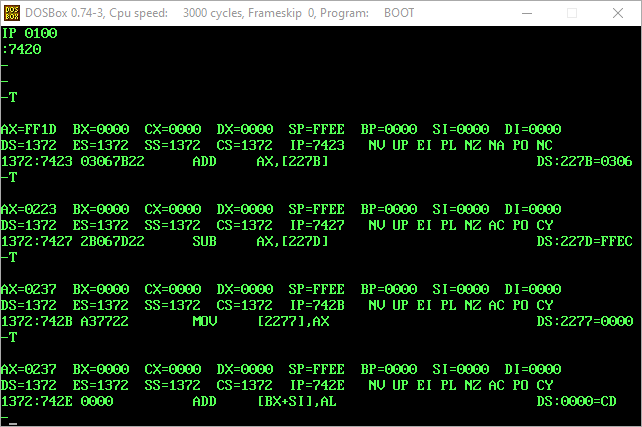
T= ‐20 (en decimal) = 1111 1111 1110 1100 (en binario) = FFEC (en hexa)

R= 567 (en decimal) = 0000 0010 0011 0111 (en binario) = 0237 (en hexa)

|  |  |  |  |
| --- | --- | --- | --- |
| VAR | DEC | BIN | HEX |
| P | -227 | 1111 1111 0001 1101 | FF1D |
| Q | 774 | 0000 0011 0000 0110 | 0306 |
| T | -20 | 1111 1111 1110 1100 | FFEC |
| P+Q | 547 | 0000 0010 0010 0011 | 0223 |
| P+Q-T | 567 | 0000 0010 0011 0111 | 0237 |

ESTADO DE LAS FLAGS[[1]](#footnote-1):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **INSTRUCCIÓN** | **V** | **S** | **Z** | **C** |
| 1 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 1 |
| 3 | 0 | 0 | 0 | 1 |
| 4 | 0 | 0 | 0 | 1 |

[[2]](#footnote-2)

**ADENDA**

Puede verse en la imagen (luego del TRACE) los siguientes valores para los FLAGS:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **INS** | **Overflow** | **Direction** | **Interrupt** | **Sign** | **Zero** | **Auxiliary Carry** | **Parity** | **Carry** |
| 1 | NV | UP | EI | PL | NZ | NA | PO | NC |
| 2 | NV | UP | EI | PL | NZ | AC | PO | CY |
| 3 | NV | UP | EI | PL | NZ | AC | PO | CY |
| 4 | NV | UP | EI | PL | NZ | AC | PO | CY |

Cuya interpretación debiera ser:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **INS** | **Overflow** | **Sign** | **Zero** | **Carry** |
| 1 | CLEAR (0-bit) No oVerflow | CLEAR (0-bit) Plus | CLEAR (0-bit) Not Zero | CLEAR (0-bit) No Carry |
| 2 | CLEAR (0-bit) No oVerflow | CLEAR (0-bit) Plus | CLEAR (0-bit) Not Zero | SET (1-bit) CarrY |
| 3 | CLEAR (0-bit) No oVerflow | CLEAR (0-bit) Plus | CLEAR (0-bit) Not Zero | SET (1-bit) CarrY |
| 4 | CLEAR (0-bit) No oVerflow | CLEAR (0-bit) Plus | CLEAR (0-bit) Not Zero | SET (1-bit) CarrY |

Y que numéricamente tiene este aspecto:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **INS** | **Overflow** | **Sign** | **Zero** | **Carry** |
| 1 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 1 |
| 3 | 0 | 0 | 0 | 1 |
| 4 | 0 | 0 | 0 | 1 |

1. Después de ejecutar I1 resulta AX = P = FF1D
2. Después de ejecutar I2 resulta AX = P+Q = 0223
3. Después de ejecutar I3 resulta AX = P+Q–T = 0237
4. Después de ejecutar I4 resulta AX = = 0237

Realizar la suma en binario (16 bits) que hace la UAL, que permite determinar este valor de AX, que es:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P | 1 | 1 | 1 | 1 |  | 1 | 1 | 1 | 1 |  | 0 | 0 | 0 | 1 |  | 1 | 1 | 0 | 1 |
| Q | 0 | 0 | 0 | 0 |  | 0 | 0 | 1 | 1 |  | 0 | 0 | 0 | 0 |  | 0 | 1 | 1 | 0 |
| **P+Q** | **0** | **0** | **0** | **0** |  | **0** | **0** | **1** | **0** |  | **0** | **0** | **1** | **0** |  | **0** | **0** | **1** | **1** |
| T | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 1 |  | 0 | 1 | 0 | 0 |
| **P+Q-T** | **0** | **0** | **0** | **0** |  | **0** | **0** | **1** | **0** |  | **0** | **0** | **1** | **1** |  | **0** | **1** | **1** | **1** |

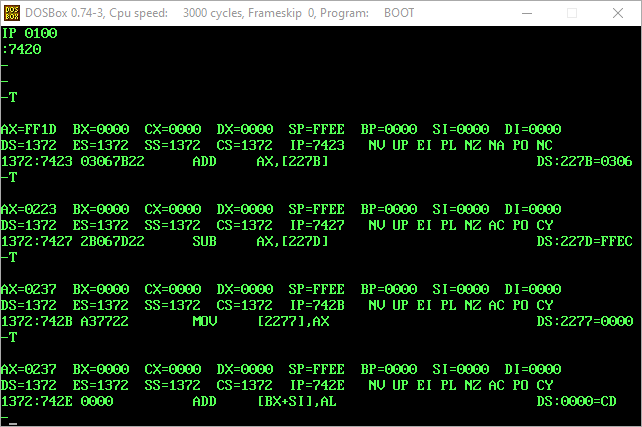
Recuerde pasar el resultado de P + Q de binario a decimal y verificar que es el signo y valor esperado.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **BINARIO A DECIMAL** | | | | | | | | | | | | | | | | | | | | |
| **PESA** | | | | | | | | | | | | | | | | | | | **POT** | **DEC** |
| **1** | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 | 15 |  |
| 0 | **1** | 0 | 0 |  | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 | 14 |  |
| 0 | 0 | **1** | 0 |  | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 | 13 |  |
| 0 | 0 | 0 | **1** |  | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 | 12 |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 0 | 0 | 0 | 0 |  | **1** | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 | 11 |  |
| 0 | 0 | 0 | 0 |  | 0 | **1** | 0 | 0 |  | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 | 10 |  |
| 0 | 0 | 0 | 0 |  | 0 | 0 | **1** | 0 |  | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 | 9 | 512 |
| 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | **1** |  | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 | 8 |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 |  | **1** | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 | 7 |  |
| 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 |  | 0 | **1** | 0 | 0 |  | 0 | 0 | 0 | 0 | 6 |  |
| 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 |  | 0 | 0 | **1** | 0 |  | 0 | 0 | 0 | 0 | 5 | 32 |
| 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | **1** |  | 0 | 0 | 0 | 0 | 4 |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 |  | **1** | 0 | 0 | 0 | 3 |  |
| 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 |  | 0 | **1** | 0 | 0 | 2 |  |
| 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 |  | 0 | 0 | **1** | 0 | 1 | 2 |
| 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | **1** | 0 | 1 |
| **DECIMAL:** | | | | | | | | | | | | | | | | | | | | **547** |
| **0** | **0** | **0** | **0** |  | **0** | **0** | **1** | **0** |  | **0** | **0** | **1** | **0** |  | **0** | **0** | **1** | **1** | **P+Q** | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **BINARIO A DECIMAL** | | | | | | | | | | | | | | | | | | | | |
| **PESA** | | | | | | | | | | | | | | | | | | | **POT** | **DEC** |
| **1** | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 | 15 |  |
| 0 | **1** | 0 | 0 |  | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 | 14 |  |
| 0 | 0 | **1** | 0 |  | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 | 13 |  |
| 0 | 0 | 0 | **1** |  | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 | 12 |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 0 | 0 | 0 | 0 |  | **1** | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 | 11 |  |
| 0 | 0 | 0 | 0 |  | 0 | **1** | 0 | 0 |  | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 | 10 |  |
| 0 | 0 | 0 | 0 |  | 0 | 0 | **1** | 0 |  | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 | 9 | 512 |
| 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | **1** |  | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 | 8 |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 |  | **1** | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 | 7 |  |
| 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 |  | 0 | **1** | 0 | 0 |  | 0 | 0 | 0 | 0 | 6 |  |
| 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 |  | 0 | 0 | **1** | 0 |  | 0 | 0 | 0 | 0 | 5 | 32 |
| 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | **1** |  | 0 | 0 | 0 | 0 | 4 | 16 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 |  | **1** | 0 | 0 | 0 | 3 |  |
| 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 |  | 0 | **1** | 0 | 0 | 2 | 4 |
| 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 |  | 0 | 0 | **1** | 0 | 1 | 2 |
| 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | **1** | 0 | 1 |
| **DECIMAL:** | | | | | | | | | | | | | | | | | | | | **567** |
| **0** | **0** | **0** | **0** |  | **0** | **0** | **1** | **0** |  | **0** | **0** | **1** | **1** |  | **0** | **1** | **1** | **1** | **P+Q-T** | |

3b) Agregar el valor de los flags SZVC en el punto 3 del TP anterior, en los casos en que intervenga la UAL y verificar en el Debug del TPR 5ºB que el valor de SZVC coincide con el calculado.

Pantalla del TRACE:



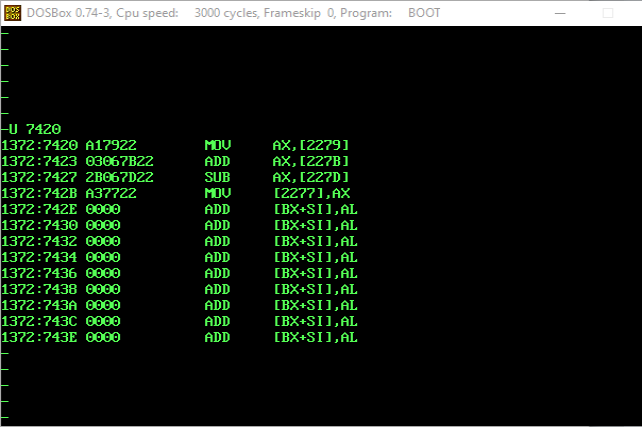
Estado de los FLAGS[[3]](#footnote-3):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **INS** | **V** | **S** | **Z** | **C** |
| 1 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 1 |
| 3 | 0 | 0 | 0 | 1 |
| 4 | 0 | 0 | 0 | 1 |

Significado:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **INS** | **Overflow** | **Sign** | **Zero** | **Carry** |
| 1 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 1 |
| 3 | 0 | 0 | 0 | 1 |
| 4 | 0 | 0 | 0 | 1 |

Explicación de CARRY:



Como puede verse, en la 2da instrucción se activa CARRY. ¿Qué sucede en esa instrucción? De la captura de pantalla puede verse que, después de asignar el valor al acumulador, pide que se sume el contenido de 227B al contenido del acumulador. Aquí en detalle la operación con los valores (en binario, obviamente) cargados en memoria:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| P |  |  | 1 | 1 | 1 | 1 |  | 1 | 1 | 1 | 1 |  | 0 | 0 | 0 | 1 |  | 1 | 1 | 0 | 1 |
| Q |  |  | 0 | 0 | 0 | 0 |  | 0 | 0 | 1 | 1 |  | 0 | 0 | 0 | 0 |  | 0 | 1 | 1 | 0 |
| P+Q | **1** |  | 0 | 0 | 0 | 0 |  | 0 | 0 | 1 | 0 |  | 0 | 0 | 1 | 0 |  | 0 | 0 | 1 | 1 |
| T |  |  | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 1 |  | 0 | 1 | 0 | 0 |
| P+Q-T |  |  | 0 | 0 | 0 | 0 |  | 0 | 0 | 1 | 0 |  | 0 | 0 | 1 | 1 |  | 0 | 1 | 1 | 1 |

Esa segunda instrucción (sumar P+Q) produce un acarreo (en rojo). En los procesadores de computadora, la bandera de acarreo es un solo bit en un registro de bandera que se utiliza para indicar cuándo se ha generado un acarreo o préstamo aritmético.

**ADENDA**

¿Por qué el DEBUG siempre tira valores en las FLAGS, aunque no haya habido operaciones aritméticas? Véase en el UNASSEMBLER y compárese con el TRACE. Según el TRACE (tal cual lo muestra el UNASSEMBLER) la primera operación fue una asignación, un valor copiado a una dirección de memoria. Sin embargo, el DEBUG lee las banderas y muestra su estado. Debo asumir que se trata, simplemente, de valores por defecto. Luego, en la segunda instrucción (I2) sí que interviene la UAL porque hay una operación aritmética. Es obvio y esperable que, de acuerdo con la operación, haya (o no) un cambio en las FLAGS. En este caso lo hubo pues así lo requería la operación. Hasta ahí, todo correcto. Pero esos estados de las FLAGS se mantienen durante I3 e I4. Se puede presumir que los estados, si no es necesario, no vuelven a sus valores por defecto (por así llamarlo) y que cambiarán la próxima vez que opere la UAL y que emita los valores correspondientes. Mientras tanto, el DEBUG siempre va a estar consultando el estado de las banderas. Y debe ser simplemente porque el DEBUG tampoco entiende las operaciones que los comandos le piden hacer al procesador. El DEBUG, simplemente, siempre lee los FLAGS.

4) Ídem TPR Nº5A.

|  |  |  |
| --- | --- | --- |
| MEM | VAR | VAL |
| 2277 | R | 37 |
| 2278 | R | 02 |

5) Ídem TPR Nº5.A XX R

|  |  |  |
| --- | --- | --- |
| CHAR | HEX | ASCII |
| R | 52 | 82 |
| = | 3D | 61 |
| 5 | 35 | 53 |
| 6 | 36 | 54 |
| 7 | 37 | 55 |

# Ejecución experimental

Los puntos 6a), 6b) y 6c) no deben realizarse.

7a) Escritura zona datos: luego de haber realizado el paso 6c en el último renglón de la pantalla se verá un guión indicador que el Debug está esperando un comando 3. Al lado de este guión se escribirá:

E 1234 ↵ (E 2848 para el DNI ejemplificado)

(Recuerde que las primeras 4 cifras del DNI indicaban la dirección del comienzo de la zona de datos paso 2a).

Luego de pulsar Enter (↵), el Debug responderá indicando el contenido de la dirección que acompaña al comando E. Dicho contenido va seguido de un punto. A la derecha de este punto el Debug está esperando que se escriba (si se desea) el nuevo contenido con que se quiere reemplazar el contenido indicado. Este nuevo contenido a escribir es el que aparece en la dirección simbolizada 1234 en la tabla determinada en 2a.

Después de escribir el nuevo contenido se debe pulsar la barra espaciadora, para evitar tipear nuevamente el comando E con la dirección siguiente. Así, el Debug mostrará automáticamente el contenido de la dirección siguiente a la modificada a continuación del último contenido modificado. Nuevamente aparecerá un valor que no interesa, acompañado por un punto. A la derecha de este punto, co mo se hizo anteriormente, hay que escribir el nuevo contenido que reemplazará al que se muestra, de acuerdo con el segundo renglón de la tabla determinada en 2ª.

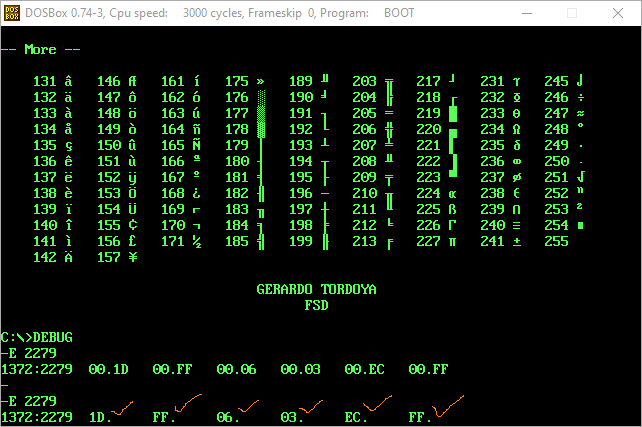
Seguidamente se debe pulsar la barra espaciadora, repitiendo el proceso anterior para cada sucesiva dirección de memoria, hasta escribir todos los datos de la tabla del paso 2a, en cuyo caso pulsar

Enter (↵). En caso de duda consultar capítulo 1.6 de La PC por dentro.

7b) Volver a tipear E 1234↵, y luego pulsar sucesivamente la barra espaciadora de modo que vayan apareciendo todos los contenidos escritos en 7a.

(Verificar con una tilde que sean los mismos que figuran en la tabla determinada en 2a).

Pantalla y tabla:



|  |  |  |
| --- | --- | --- |
| MEM | VAR | VAL |
| 2279 | P | 1D |
| 227A | P | FF |
| 227B | Q | 06 |
| 227C | Q | 03 |
| 227D | T | EC |
| 227E | T | FF |

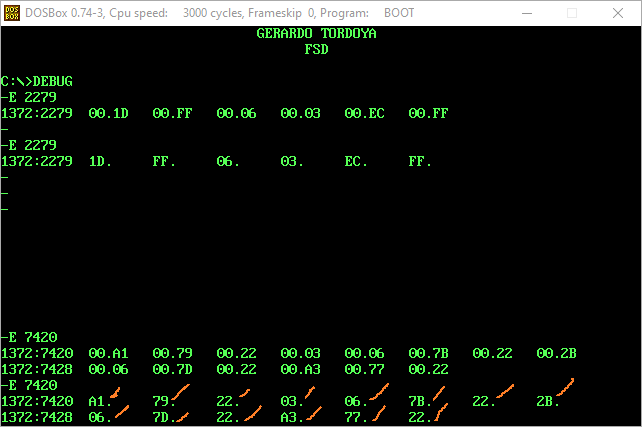
7c) Escritura zona instrucciones: escribir al lado del guión del Debug:

E 5678 ↵

(Siendo que las últimas 4 cifras del DNI indicaban la dirección del comienzo de la zona de instrucciones (paso 2d). Luego proceder como en 7a) pero ahora para escribir en memoria la tabla del paso 2d).

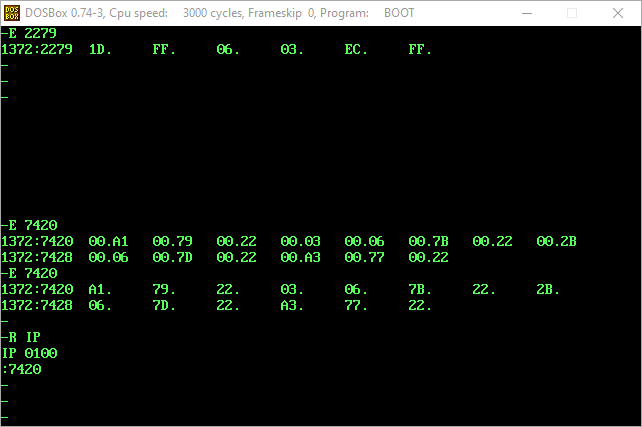
7d) Volver a tipear E 5678↵, y luego pulsar sucesivamente la barra espaciadora de modo que vayan apareciendo todos los contenidos escritos en 7c.

(Verificar con una tilde que sean los mismos que figuran en la tabla determinada en 2d.)



|  |  |
| --- | --- |
| MEM | INS |
| 7420 | A1 |
| 7421 | 79 |
| 7422 | 22 |
| 7423 | 03 |
| 7424 | 06 |
| 7425 | 7B |
| 7426 | 22 |
| 7427 | 2B |
| 7428 | 06 |
| 7429 | 7D |
| 742A | 22 |
| 742B | A3 |
| 742C | 77 |
| 742D | 22 |

7e) Escribir al lado del guión del Debug el comando RIP↵ .Debajo de este comando aparecerá el valor actual del IP seguido de dos puntos, debiendo escribirse en el tercer renglón la dirección donde comienza la primera instrucción, establecida en el paso 2d., seguida de ↵.



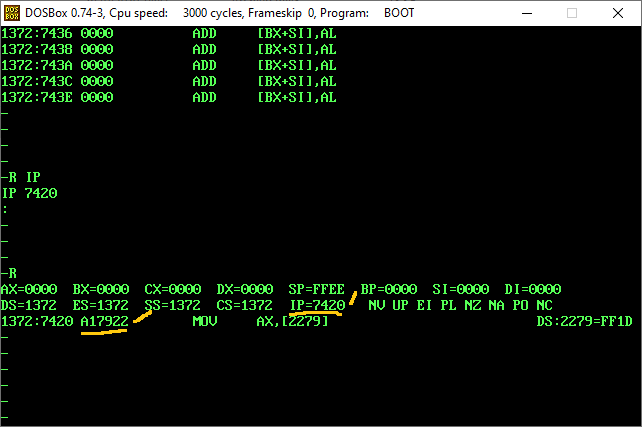
7f) Escribir al lado del guión el comando R ↵, aparecerán en pantalla 3 renglones con el estado de distintos registros de la UCP. De esta información en este paso sólo interesa el valor del registro IP y en el tercer renglón a la izquierda el código de máquina de la primer instrucción de la secuencia.

ANOTAR A CONTINUACION ESTOS VALORES:

IP=7420

CODIGO DE LA PROXIMA INSTR A EJECUTAR = A17922

Verificar: INDICANDO EN EL DEBUG CON UN TILDE O CON RESALTADOR



|  |  |
| --- | --- |
| MEM | INS |
| 7420✓ | A1✓ |
| 7421 | 79✓ |
| 7422 | 22✓ |

7f.1) Si el valor del IP coincide con la dirección donde comienza la primera instrucción (I1) establecida en 2d. ✓

7f.2) Si el código de máquina de la primera instrucción a ejecutar es el mismo que el anotado más arriba. ✓

7g) Escribir al lado del guión el comando T ↵, se ejecutará la instrucción I1 que apareció en pantalla al realizar el paso 7d. El Debug, como en el paso 7d, mostrará en pantalla tres renglones de información. (El comando T siempre ejecuta el código de máquina de la instrucción anterior)

De esta información anotar a continuación los siguientes valores: I1

AX=FF1D

IP=7423

CODIGO DE LA PROXIMA INSTR A EJECUTAR = 03067B22

Verificar INDICANDO EN EL DEBUG CON UN TILDE O CON RESALTADOR. **← IMAGEN EN I4**

7g.1) Si el valor de AX y de los flags coincide con el calculado en el punto 3 después de ejecutar I1. ✓

**NOTA ACLARATORIA**

La UAL dispara siempre esos controladores (las banderas), según dice la teoría. O al menos, el DEBUG las lee (es materia de investigación, supongo). Pero, de todas maneras, evidentemente hay valores por defecto[[4]](#footnote-4). Y si una asignación (un MOV) no hizo saltar (cambiar) el estado de las banderas, bueno, armo esta respuesta a partir de ahí. Si del análisis de las operaciones se le pidió al procesador coinciden los valores que se suponen tienen que tirar esos FLAGS (ya sea cuando son valores default o cuando por alguna operación aritmética se haya modificado el estado de alguna de las banderas), el resultado, según lo calculado, coincide en el DEBUG con lo que se puede corroborar al hacer correr esos comandos y valores (según se puede ver en las capturas de pantalla)[[5]](#footnote-5).

7g.2) Si el valor de IP coincide con la dirección de comienzo de I2, establecida en 2d. ✓

7g.3) Si el código de la próxima instrucción a ejecutar (I2) es el que aparece a partir del tercer renglón, observando la tabla del punto 2d. ✓

7h) Ídem punto 7g, con lo cual se ejecutará I2 volcando los siguientes valores: I2

AX=0223

IP=7427

CODIGO DE LA PROXIMA INSTR A EJECUTAR = 2B067D22

Verificar INDICANDO EN EL DEBUG CON UN TILDE O CON RESALTADOR, si el valor de AX y de los flags coincide con el valor calculado, y si IP y la próxima instrucción a ejecutar coinciden con lo establecido en 2d. ✓ **← IMAGEN EN I4**

7i) Ídem punto 7g, se ejecutará I3 volcando los siguientes valores: I3

AX=0237

IP=742B

CODIGO DE LA PROXIMA INSTR A EJECUTAR = A37722

Verificar INDICANDO EN EL DEBUG CON UN TILDE O CON RESALTADOR si AX coincide con el valor calculado, y si IP y la próxima instrucción a ejecutar coinciden con lo establecido en 2d. ✓ **← IMAGEN EN I4**

7j) Ídem 7g, con lo cual se ejecutará I4 volcando los siguientes valores: I4

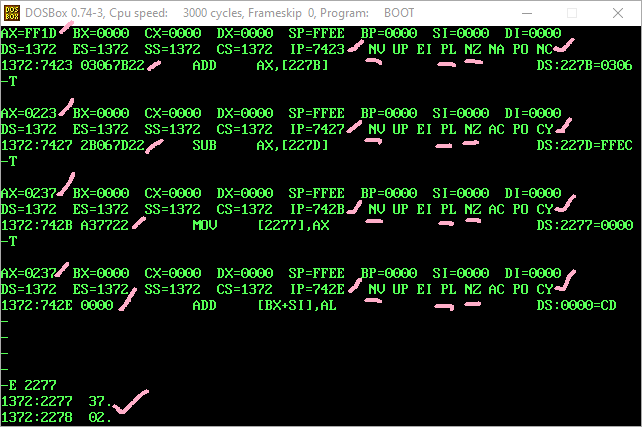
AX=0237

IP=742E

CODIGO DE LA PROXIMA INSTR A EJECUTAR = 0000

Verificar INDICANDO EN EL DEBUG CON UN TILDE, si AX coincide con el valor calculado (también indicado en el paso 4), y si IP y la próxima instrucción a ejecutar coinciden con lo establecido en 2d. ✓

Después de este paso, en la pantalla deberán verse cuatro comandos T, en correspondencia con la ejecución de las 4 instrucciones que componen la secuencia.



7k) La instrucción I4 de código de operación A3 ordena pasar el contenido de AX a la dirección de memoria que acompaña a A3. Para verificar que se ejecutó esta instrucción se debe examinar esta dirección de memoria y la siguiente, correspondientes a la ubicación de la variable R. Esto implica que en el Debug debe hacerse, de acuerdo con el paso 2a:

‐E dirección de la variable R (primeras 4 cifras del DNI) ↵

EL RESULTADO SE HA SEÑALADO EN LA IMAGEN ANTERIOR ↑

REPRODUCCIÓN DE LA TABLA PARA CORROBORAR:

|  |  |  |
| --- | --- | --- |
| MEM | VAR | VAL |
| 2277 | R | 37 |
| 2278 | 02 |
| 2279 | P | 1D |
| 227A | FF |
| 227B | Q | 06 |
| 227C | 03 |
| 227D | T | EC |
| 227E | FF |

(El valor de esta dirección y el de la siguiente debe coincidir con el indicado en el paso 4, en cuyo caso se debe resaltar en el Debug esta coincidencia. Caso contrario debe buscarse el error cometido.)

ADJUNTAR A ESTAS PAGINAS TODOS LOS PASOS REALIZADOS CON EL DEBUG.

1. **AYUDA-MEMORIA:**

   |  |  |  |  |  |  |
   | --- | --- | --- | --- | --- | --- |
   | **FLAGS** | **TIPO** | **ESTAD-0** | **CARG-0** | **ESTAD-1** | **CARG-1** |
   | oVerflow | Enteros | NV | V=0 | OV | V=1 |
   | Sign | Enteros | PL | S=0 | NG | S=1 |
   | Zero | Enteros + Naturales | NZ | Z=0 | ZR | Z=1 |
   | Carry | Naturales | NC | C=0 | CY | C=1 |

   [↑](#footnote-ref-1)
2. Los cuatro registros principales de 16 bits de un procesador 8086 se llaman:

   |  |  |  |  |
   | --- | --- | --- | --- |
   | AX: | A**H** | A**L** | Accumulator |
   | BX: | B**H** | B**L** | Base Register |
   | CX: | C**H** | C**L** | Count Register |
   | DX: | D**H** | D**L** | Data Register |
   |  | **8** bits | **8** bits |  |
   |  | **1 byte** | **1 byte** |  |

   Los otros registros de 16 bits (2 bytes) del procesador 8086 son:

   |  |  |
   | --- | --- |
   | **SP** | Stack Pointer |
   | **BP** | Base Pointer |
   | **SI** | Source Index |
   | **DI** | Destination Index |
   | **CS** | Code Segment |
   | **DS** | Data Segment |
   | **SS** | Stack Segment |
   | **ES** | Extra Segment |
   | **IP** | Instruction Pointer |

   Este es un registro de 8 bits que contiene 1 bit de datos para cada de las ocho "Banderas" que deben interpretarse de la siguiente manera:

   Abreviaturas para el nombre de las banderas: => **of df if sf zf af pf cf**

   Si las banderas están configuradas en 1 se ven así: OV DN EI NG ZR AC PE CY

   Si las banderas están configuradas en 0 se ven así: NV UP DI PL NZ NA PO NC

   **FLAGS SET (a 1-bit) CLEARed (a 0-bit)**

   --------------- --------------- -------------------

   Overflow **of =** OV (OVerflow) NV [No oVerflow]

   Direction **df =** DN (decrement) UP (increment)

   Interrupt **if =** EI (enabled) DI (disabled)

   Sign **sf =** NG (negative) PL (positive)

   Zero **zf =** ZR [zero] NZ [ Not zero]

   Auxiliary Carry **af =** AC NA [ No AC ]

   Parity **pf =** PE (even) PO (odd)

   Carry **cf =** CY [Carry] NC [ No Carry]

   Las abreviaturas individuales tienen estos significados:

   OV = Overflow NV = No oVerflow

   DN = DowN UP = (up)

   EI = Enable Interupt DI = Disable Interupt

   NG = NeGative PL = PLus

   ZR = ZeRo NZ = Not Zero

   AC = Auxiliary Carry NA = Not Auxiliary carry

   PE = Parity Even PO = Parity Odd

   CY = CarrY NC = No Carry [↑](#footnote-ref-2)
3. AYUDA-MEMORIA:

   |  |  |  |  |  |  |
   | --- | --- | --- | --- | --- | --- |
   | **FLAGS** | **TIPO** | **ESTAD-0** | **CARG-0** | **ESTAD-1** | **CARG-1** |
   | oVerflow | Enteros | NV | V=0 | OV | V=1 |
   | Sign | Enteros | PL | S=0 | NG | S=1 |
   | Zero | Enteros + Naturales | NZ | Z=0 | ZR | Z=1 |
   | Carry | Naturales | NC | C=0 | CY | C=1 |

   [↑](#footnote-ref-3)
4. A menos que una ulterior investigación afirme que, por ejemplo, en un MOV, también interviene la UAL. [↑](#footnote-ref-4)
5. Véase además ADENDA al punto 3B. [↑](#footnote-ref-5)