EGCP 2110-01
Microprocessors
Laboratory #3

Multi-Byte Logic and Arithmetic

Prepared By:
Alex Laird
Collin Barrett

on

Saturday, September 19th, 2009

Source Code

Below is the assembly code that we ensured worked in the lab. We wrote it prior to having access to the Z-80 Microprocessor to test it. There were no syntax errors, and the code worked the first time we ran in.

It's worth noting that our program allowed the user to enter the three-byte hexadecimal number properly (most significant to least significant), the program accounted for the need to add the numbers from least significant to most significant in its calculations. It then displayed the output in the correct format (most significant to least significant).

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Created by: Alex Laird and Collin Barrett
;Date: Sept. 14, 2009
;Class: Microprocessors
;Lab 3: Multi-Byte Logic and Arithmetic
;Purpose: To explore the implementation of multi-byte logic and
;         arithmetic on the Z80.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;define some constants needed for the ROM subroutine calls
HEXTO7:     EQU   0FF1H       ;calling address for HEXTO7
DISPV:      EQU   1F12H       ;calling address for DISPV
KBRD:       EQU   0FEBH       ;calling address for KBDR

;point the program start to the beginning of memory
      ORG   1800H

;display the initial values of INV1
      LD    IX, INV1
      LD    A, (IX+0)         ;indexed
      LD    (DISPV+0), A
      LD    A, (IX+1)
      LD    (DISPV+1), A
      LD    A, (IX+2)
      LD    (DISPV+2), A
      CALL  HEXTO7
      CALL  KBRD
;display the initial values of INV2
      LD    IX, INV2
      LD    A, (IX+0)
      LD    (DISPV+0), A
      LD    A, (IX+1)
      LD    (DISPV+1), A
      LD    A, (IX+2)
      LD    (DISPV+2), A
      CALL  HEXTO7
      CALL  KBRD

;add INV1 and INV2
      LD    IX, INV1
      LD    IY, ADDOUT
```

```
      LD    IX, INV1
      LD    A, (IX+2)
      LD    IX, INV2
      ADC   A, (IX+2)
      LD    (IY+2), A

      LD    IX, INV1
      LD    A, (IX+1)
      LD    IX, INV2
      ADC   A, (IX+1)
      LD    (IY+1), A

      LD    IX, INV1
      LD    A, (IX+0)
      LD    IX, INV2
      ADC   A, (IX+0)
      LD    (IY+0), A

;display addition results
      LD    IX, ADDOUT
      LD    A, (IX+0)
      LD    (DISPV+0), A
      LD    A, (IX+1)
      LD    (DISPV+1), A
      LD    A, (IX+2)
      LD    (DISPV+2), A
      CALL  HEXTO7
      CALL  KBRD

;subtract INV1 and INV2
      LD    IX, INV1
      LD    IY, SUBOUT

      LD    IX, INV1
      LD    A, (IX+2)
      LD    IX, INV2
      SBC   A, (IX+2)
      LD    (IY+2), A

      LD    IX, INV1
      LD    A, (IX+1)
      LD    IX, INV2
      SBC   A, (IX+1)
      LD    (IY+1), A

      LD    IX, INV1
      LD    A, (IX+0)
      LD    IX, INV2
      SBC   A, (IX+0)
      LD    (IY+0), A

;display subtraction results
      LD    IX, SUBOUT
      LD    A, (IX+0)
      LD    (DISPV+0), A
      LD    A, (IX+1)
      LD    (DISPV+1), A
```

```
      LD    A, (IX+2)
      LD    (DISPV+2), A
      CALL  HEXTO7
      CALL  KBRD

;logical AND INV1 and INV2
      LD    IX, INV1
      LD    IY, ANDOUT

      LD    IX, INV1
      LD    A, (IX+2)
      LD    IX, INV2
      AND   (IX+2)
      LD    (IY+2), A

      LD    IX, INV1
      LD    A, (IX+1)
      LD    IX, INV2
      AND   (IX+1)
      LD    (IY+1), A

      LD    IX, INV1
      LD    A, (IX+0)
      LD    IX, INV2
      AND   (IX+0)
      LD    (IY+0), A

;display logical AND results
      LD    IX, ANDOUT
      LD    A, (IX+0)
      LD    (DISPV+0), A
      LD    A, (IX+1)
      LD    (DISPV+1), A
      LD    A, (IX+2)
      LD    (DISPV+2), A
      CALL  HEXTO7
      CALL  KBRD

;logical OR INV1 and INV2
      LD    IX, INV1
      LD    IY, OROUT

      LD    IX, INV1
      LD    A, (IX+2)
      LD    IX, INV2
      OR    (IX+2)
      LD    (IY+2), A

      LD    IX, INV1
      LD    A, (IX+1)
      LD    IX, INV2
      OR    (IX+1)
      LD    (IY+1), A

      LD    IX, INV1
      LD    A, (IX+0)
      LD    IX, INV2
```

```
      OR    (IX+0)
      LD    (IY+0), A

;display logical OR results
      LD    IX, OROUT
      LD    A, (IX+0)
      LD    (DISPV+0), A
      LD    A, (IX+1)
      LD    (DISPV+1), A
      LD    A, (IX+2)
      LD    (DISPV+2), A
      CALL  HEXTO7
      CALL  KBRD

;logical XOR INV1 and INV2
      LD    IX, INV1
      LD    IY, XOROUT

      LD    IX, INV1
      LD    A, (IX+2)
      LD    IX, INV2
      XOR   (IX+2)
      LD    (IY+2), A

      LD    IX, INV1
      LD    A, (IX+1)
      LD    IX, INV2
      XOR   (IX+1)
      LD    (IY+1), A

      LD    IX, INV1
      LD    A, (IX+0)
      LD    IX, INV2
      XOR   (IX+0)
      LD    (IY+0), A

;display logical XOR results
      LD    IX, XOROUT
      LD    A, (IX+0)
      LD    (DISPV+0), A
      LD    A, (IX+1)
      LD    (DISPV+1), A
      LD    A, (IX+2)
      LD    (DISPV+2), A
      CALL  HEXTO7
      CALL  KBRD

;define storage locations
VARS        ORG   1D00H ;start variable storage at 1D00
INV1:       DEFS  3     ;first input vector
INV2:       DEFS  3     ;second input vector
ADDOUT:     DEFS  3     ;addition result storage
SUBOUT:     DEFS  3     ;subtraction result storage
ANDOUT:     DEFS  3     ;AND logic result
OROUT:      DEFS  3     ;OR logic result
XOROUT:     DEFS  3     ;XOR logic result
```

Results

The results we received from our inputs, INV1 and INV2, are shown in Table 1. The first two numbers were chosen specifically to test extreme boundary conditions. The rest of the numbers were chosen with the intent to gain a wide variety of diverse numbers to manipulate, testing other simpler boundary conditions, for instance, the carrying boundary from one byte to the next.

INV1	INV2	ADDOUT	SUBOUT	ANDOUT	OROUT	XOROUT
FFFFFF	000002	000001	FFFFFD	000002	FFFFFF	FFFFFD
000005	000007	00000C	FFFFFE	000005	000007	000002
A72330	0021FB	A7452B	A70135	002130	A723FB	A702CB
013A4C	009D1A	01D760	009D32	001808	01BFSE	01A756
02440F	31247A	336889	D11F95	00040A	33647F	336075
BC0DEE	012FFC	BD3DEA	BADDF2	000DEC	BD2FFE	BD2212

Table 1 Numbers Used and Results Received

Problems Encountered

When we first assembled and ran our program, we received valid output for the first three equations and the wrong output for the fourth equation. Our board flicked numerous times as well, so we thought a short may have occurred. We unplugged the board, cleared the memory, and downloaded our code to memory again. We had to do this numerous times as something on the board kept glitching and giving us improper output, but our program *did* return the correct results when the board kept power.

Conclusions

We used the ADC and SBC commands as opposed to standard ADD and SUB commands in order to compensate for the carries across the three bytes (from least significant to most significant bytes).

As a whole, we had little to no trouble with the assignment. Except for the few hardware hiccups we had when we got to the lab, the assignment was challenging but manageable.