EGCP 2110-01
Microprocessors
Laboratory #5

Functions

Prepared By:
Alex Laird
Collin Barrett

on

Monday, October 1st, 2009

Low-Level 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.

Upon execution, the program displays a U1 to prompt the user to input the first three-bit number. After the user enters this number, the program prompts for the second with U2. The user can then press NEXT, PREV, or GO. NEXT performs an addition, PREV performs a subtraction, and GO performs a multiplication. The multiplication subroutine performs the multiply using shifting and addition techniques instead of repeated addition. After performing the respective routine, the program displays the answer to the user and awaits the next command. To end the program, the user can press the STEP key.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Created by: Alex Laird and Collin Barrett
;Date: Sept. 28, 2009
;Class: Microprocessors
;Lab 5: Functions
;Purpose: To study the process of implementing functions
;         through the development of a calculator program.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Dear Colegue,                                                        ;
; This  is the beginning of my calculator program.  The main keyboard ;
; input routine is written, it will fetch two 3-nibble values from the;
; user via the keypad and store them into memory.  It then calls the  ;
; keypad one more time to get the operator (reverse polar notation).  ;
; This is where I was when I took my vocation.  You need to finish    ;
;writing the logic that will decide which function to            ;
; call based on the key that was pressed for the function.  Here are  ;
; the function mappings given by management:                          ;
;                                                                      ;
; Key     KBRD-Value    Function                                      ;
; NEXT       10H        Add U1 to U2                                  ;
; PREV       11H        Subtract U2 from U1                           ;
; GO         12H        Multiply U1 x U2                              ;
; any other             Sound tone, prompt for new operation key      ;
;                                                                      ;
; After displaying the result, you must call KBRD to determine if you ;
; should run the program again.  If you receive the STEP key from the ;
; keyboard after displaying the result, you should terminate the      ;
; program any other key at this point should rerun the program.       ;
; The STEP key returns the value 13H                                  ;
;                                                                      ;
; Our supervisor has already done the software review on the functions;
; I have written and they have passed QA.  Therefore, you are not     ;
; permitted to make any changes to the existing code.  Your changes   ;
; must appear in the areas designated.                                ;
;                                                                      ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
KBRD  EQU   0FEBH       ; The system equates defined by ZAD
HEXTO7      EQU   0FF1H
MESOUT      EQU   0FF4H
TONE  EQU   0FFAH
DISPV EQU   1F12H

      ORG   1800H
      JP    MAIN        ; Jump over my functions and get to main
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                DO NOT MODIFY ANY EXISTING CODE                    ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; GETVAL: Function to fetch a 12-bit number from the user
; ENTRY:  User must provide in the IY a pointer to a 2-byte location
;         where they want the 12-bit number stored (little indian)
; EXIT:   GETVAL will obtain a 12-bit (3 nibble) value from the user
;         in HEX format and store it at the location pointed to by the
;         IY register.  All registers are returned un-changed
; NOTE:   Calls function GETNIB which actually gets one nibble
;         at a time
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

GETVAL:
      PUSH  AF          ; Save all registers
      PUSH  BC
      PUSH  DE

      PUSH  HL
      PUSH  IX
      LD    IX,DISPV
      PUSH  IY
      LD    A,0
      LD    (IX),A
      LD    (IX+1),A
      LD    (IX+2),A
      LD    B,0         ; Ask GETNIB for 1st nibble
      CALL  GETNIB
      LD    B,1         ; Ask GETNIB for 2nd nibble
      CALL  GETNIB
      LD    B,2         ; Ask GETNIB for 3rd nibble

      CALL  GETNIB
      LD    HL,40D            ; Sound tone indicating value was received
      CALL  TONE
      LD    A,(DISPV+2) ; Copy 12-bit value from storage in DISPV to
      POP   IY          ;    the variable passed in by pointer
      LD    (IY),A
      LD    A,(DISPV+1)
      LD    (IY+1),A
      CALL  HEXTO7            ; Display the final value so the user can see
      CALL  KBRD        ;   it before proceeding
      POP   IX          ; Restore the registers to original state
      POP   HL
      POP   DE
      POP   BC
      POP   AF
      RET
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; GETNIB:  This is an internal function called by GETVAL to fetch a 1-digit
;          HEX nibble.  It checks for valid values accepting only 0-F keys.
;          This function assembles a 12-bit (3-nibble) value and stores it in
;          the DIPSV variable.  The calling function (GETVAL) must copy the
;          final value from DISPV.
; ENTRY:   The calling function must specify in the B register which nibble is
;          being fetched.  0=first value, 1=second, 2=third.  The 3-nibble value
;          is assembled by assuming the calls are made in sequential order.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

GETNIB:
      CALL  HEXTO7            ;Display current value and get next digit
      CALL  KBRD
      CP    10H         ;Check key press to verify between 0-F
      JP    M,NIBOK
      LD    HL,250D
      CALL  TONE
      JP    GETNIB            ;If not valid digit, sound tone and re-enter digit
NIBOK:
      LD    C,A         ;Hold the value aside
      LD    A,B         ;Which nibble is this supposed to be?
      CP    0
      JP    NZ,NOT0           ;If 1st nibble then
      LD    (IX+2),C    ;   save as first nibble
      RET
NOT0:
      CP    1           ;Else If 2nd Nibble Then
      JP    NZ,NOT1
      LD    A,(IX+2)    ;   Get 1st nibble from memory
      RLC   A           ;   Shift it over to left to make room for 2nd
      RLC   A
      RLC   A
      RLC   A
      OR    C           ;   Append 2nd nibble behind first
      LD    (IX+2),A    ;   Save 1st and 2nd nibble as a byte
      RET
NOT1:
      LD    A,(IX+2)    ;Else If 3rd Nibble Then
      SRL   A           ;   Get first byte and pull out 1st nibble
      SRL   A
      SRL   A
      SRL   A           ;
      LD    (IX+1),A    ;   Save 1st nibble as New Byte  (the MSB)
      LD    A,(IX+2)    ;   Get first byte again
      SLA   A           ;   Shift out the first nibble making room for 3rd
      SLA   A
      SLA   A
      SLA   A
      OR    C           ;   Stick 3rd nibble into second making new LSB
      LD    (IX+2),A    ;   Save the LSB
      RET

;;;;;;;;;;;;;;; END OF MY FUNCTIONS ;;;;;;;;;;;;;;;;;
```

```
MAIN:
      LD    IX,MSG1           ; Prompt user for the first 12-bit value
      CALL  MESOUT
      CALL  KBRD
      LD    IY,U1       ; Point to where the 12-bits should be stored
      CALL  GETVAL            ; Make call for 12-bit input

      LD    IX,MSG2           ; Prompt user for the second value
      CALL  MESOUT
      CALL  KBRD
      LD    IY,U2       ; Point to storoage for 2nd value
      CALL  GETVAL            ; Call for the value

PROMPT:
      LD    IX,MSG3           ; Prompt user for Operation
      CALL  MESOUT
      CALL  KBRD        ; A reg has the function code
      LD    IX, U1
      LD    IY, U2
;if the user pushed Next, add
      CP    10H
      CALL  Z, ADD
      JP    Z, MAIN
;if the user pushed Prev, subtract
      CP    11H
      CALL  Z, SUB
      JP    Z, MAIN
;if the user pushed Go, multiply
      CP    12H
      CALL  Z, MULT16
      JP    Z, MAIN
      CP    13H
      CALL  Z, 0000H
;no valid input was given, so alert/prompt again
      CALL  TONE
      LD    IX, MSG4
      CALL  MESOUT
      CALL KBRD
      CP    12H
      JP    Z, PROMPT

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; ADDITION SUBROUTINE
; Adds the number stored in HL with the number stored in DE and
; stores the sum in HL before returning.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
ADD:
      PUSH  AF
      LD    H, (IX+1)
      LD    L, (IX+0)
      LD    D, (IY+1)
      LD    E, (IY+0)
      ADC   HL, DE
      CALL  DISPS
      POP   AF
```

```
      RET

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; END SUBROUTINE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; SUBTRACTION SUBROUTINE
; Subtracts the number stored in DE from the number stored in HL and
; stores the difference in HL before returning.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
SUB:
      PUSH  AF
      LD    H, (IX+1)
      LD    L, (IX+0)
      LD    D, (IY+1)
      LD    E, (IY+0)
      SBC   HL, DE
      CALL  DISPS
      POP   AF
      RET
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; END SUBROUTINE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; MULTIPLICATION SUBROUTINE
; Using shift commands instead of repeated addition, multiplies the
; number stored in HL by the number stored in DE and stores the
; product in HL before returning.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
MULT16:
      PUSH  AF
      PUSH  IX
      PUSH  IY
      LD    H, (IX+1)
      LD    L, (IX+0)
      LD    D, (IY+1)
      LD    E, (IY+0)
      LD    A, H        ;CHECK FOR ZEROS INITIALLY
      OR    L
      JP    Z, MZERO    ;ESCAPE WITH ZERO RESULT IF ZERO
      LD    A, D
      OR    E
      JP    Z, MZERO    ;ESCAPE WITH ZERO RESULT IF ZERO
      LD    C, 16H            ;COUNTER FOR 16 BITS
      LD    IX, MULTIPLIER    ;POINT TO 4 BYTE MULTIPLIER (SHIFTED)
      LD    IY, MRESULT ;POINT TO 4 BYTE RESULT (ACCUM)
      LD    A, 00H            ;CLEAR PREVIOUS RESULT
      LD    (IY+0), A
      LD    (IY+1), A
      LD    (IY+2), A
      LD    (IY+3), A
      LD    (MULTIPLIER), DE;SAVE INITIAL MULTIPLIER IN LOW BYTES
      LD    (IX+2), A
      LD    (IX+3), A
MTOP:
```

```
      RR    H           ;ROTATE THE MULTIPLICAND
      RR    L           ;CARRY WILL CONTAIN THE PREV LSB
      JP    NC, NOADD   ;IF NO CARRY DO NOT ADD MULTIPLIER
      LD    A, (IY)           ;PERFORM 32 BIT ADDITION
      ADD   A, (IX)
      LD    (IY), A
      LD    A, (IY+1)
      ADC   A, (IX+1)
      LD    (IY+1), A
      LD    A, (IY+2)
      ADC   A, (IX+2)
      LD    (IY+2), A
      LD    A, (IY+3)
      ADC   A, (IX+3)
      LD    (IY+3), A
NOADD:
      SLA   (IX)        ;SHIFT MULTIPLIER LEFT TO BRING 0
      RL    (IX+1)
      RL    (IX+2)
      RL    (IX+3)
      DEC   C           ;SEE IF MORE BITS TO DO
      JP    NZ, MTOP
MPDONE:
      LD    DE, (MRESULT)     ;FETCH RESULTS
      LD    HL, (MRESULT+2)
      CALL  DISPM
      POP   IY
      POP   IX
      POP   AF
      RET
MZERO:
      LD    DE, 0000H   ;IF EITHER INPUT WAS ZERO, RETURN
      LD    HL, 0000H   ;ZERO
      CALL  DISPM
      POP   IY
      POP   IX
      POP   AF
      RET

;declare variables for multiplication
MULTIPLIER: DEFS  4
MRESULT:    DEFS  4
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; END SUBROUTINE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; DISPLAY-STANDARD SUBROUTINE
; Displays the sumation or difference stored in the HL registers
; to the 7-segment display.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
DISPS:
      LD    IX, DISPV
      LD    (IX+1), H
      LD    (IX+2), L
      CALL  HEXTO7
      CALL  KBRD
```

```
      RET

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; END SUBROUTINE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; DISPLAY-MULTIPLICATION SUBROUTINE
; Displays the product stored in the HL and DE register to the
; 7-segment display.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
DISPM:
      LD    IX, DISPV
      LD    (IX+2), E
      LD    (IX+1), D
      LD    (IX+0), L
      CALL  HEXTO7
      CALL  KBRD
      RET
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; END SUBROUTINE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;status variable declarations
MSG1: DEFM  'U1    '
MSG2: DEFM  'U2    '
MSG3: DEFM  'OPCODE'
MSG4: DEFM  'INVALD'

;variables for storing the input values
U1:   DEFS  2
U2:   DEFS  2
```

Description of the Code

Functions were used for the addition, subtraction, and multiplication. To pass the parameters into addition, subtraction, and multiplication functions, pointers are stored in the IX and IY register. The actual result values are returned in the HL register for the addition and subtraction functions, and the multiplication register returns the actual values in the HL and DE registers.

To implement these functions with negative numbers, the code could be modified to check the first bit each time since the first bit is the determining factor for a negative number in two's compliment. If the number is set to be negative, corrections could be made before mathematical calculations are performed on the numbers.

Conclusions

Using functions with simple call and return commands, this simple calculator was easy to implement. It gave a good insight into the structure and implementation of subroutines on the Z80 Microprocessor.