EGCP 2110-01
Microprocessors
Laboratory #4

Branches and Loops

Prepared By:
Alex Laird
Collin Barrett

on

Monday, September 28th, 2009

High-Level (Java) Source Code

Below is the high-level source code written using the Java programming language. The program simply asks the user to input a hexadecimal value between 00H and FFH. If the user inputs 00H, the program terminates, otherwise it enters an inner loop in which in adds all odd numbers from 01H to the input value.

```
package microlab4;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

/**
 * A program that adds the odd integers from 1 to the users input.
 *
 * @author Alex Laird
 * @author Collin Barrett
 */
public class Main
{
    // declare variables for reading from console
    static final InputStreamReader isr = new InputStreamReader(System.in);
    static final BufferedReader cin = new BufferedReader(isr);

    /**
     * The main class from which the program executes.
     *
     * @param args The command-line arguments.
     */
    public static void main(String[] args) throws IOException
    {
        // the variable to hold the sum
        int sum;

        // get valid input for the first time
        System.out.print("Enter an integer between 01H and 0FFH: ");
        String line = cin.readLine();
        // drop the H for hex off if the user gave it
        if(line.toUpperCase().endsWith("H"))
        {
            line = line.substring(0, line.length() - 1);
        }
        int input = Integer.parseInt(line, 16);

        // verify input
        if(input < 0 || input > 255)
        {
            System.out.println("\nThe number you entered was not between 01H”
                               + “ and 0FFH!");
            return;
        }

        // loop until the user enters 0
        while(input != 0)
```

```
        {
            sum = 0;

            // loop until odd meets the input value
            int odd = 1;
            while(odd <= input)
            {
                sum += odd;
                odd += 2;
            }

            // get valid input for the next time
            System.out.println("Sum: "
                               + Integer.toHexString(sum).toUpperCase());
            System.out.print("\nEnter an integer between 01H and 0FFH: ");
            line = cin.readLine();
            // drop the H for hex off if the user gave it
            if(line.toUpperCase().endsWith("H"))
            {
                line = line.substring(0, line.length() - 1);
            }
            input = Integer.parseInt(line, 16);

            // verify input
            if(input < 0 || input > 255)
            {
                System.out.println("\nThe number you entered was not between”
                                   + “ 01H and 0FFH!");
                return;
            }
        }

        System.out.println("\nDONE!");
    }
}
```

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 waits for the user to input two hexadecimal numbers. Once the full four-bit number was received, the program waits for the user to press the “GO” key on the keypad. The application then performs the addition of all odd numbers from 01H up to the input value (inclusive) and displays the result. To exit the program, the user may input 00H as their two hexadecimal numbers.

The program implements an inner loop and an outer loop. The outer continues to ask the user for input and sum the odd numbers from that input until 00H is received. The inner loop performs the summation until the input value is reached.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Created by: Alex Laird and Collin Barrett
;Date: Sept. 21, 2009
;Class: Microprocessors
;Lab 4: Branches and Loops
;Purpose: To explore the use of branches and loops in programming for 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
SCAN:       EQU   0FFEH       ;calling address for SCAN

;point the program start to the beginning of memory (assurance call)
      ORG   1800H

;init 7-segment memory locations
      LD    A, 00H
      LD    (DISPV+0), A
      LD    (DISPV+1), A
      LD    (DISPV+2), A

;get the two input values from the user
LOOP1:
      CALL  KBRD
      LD    (DISPV+2), A
      LD    B, A
      LD    A, 00H            ;clear A to clear displays
      LD    (DISPV+0), A      ;set display for clear
      LD    (DISPV+1), A      ;set display for clear
      CALL  HEXTO7
      CALL  KBRD
      SLA   B
      SLA   B
      SLA   B
      SLA   B
      OR    B
      SUB   00H               ;check if exit condition met (00H)
      JP    Z, END            ;zero flag will be set if exit
      LD    (ORIG), A         ;load input into permanent memory location
      LD    (DISPV+2), A
      CALL  HEXTO7
GO:
      CALL  KBRD              ;waits for user to press "GO" key
      SUB   12H
      JP    NZ, GO

;if input is 01H, jump to end
      LD    A, (ORIG)
      SUB   01H
      JP    NZ, CONT
      LD    H, 00H
      LD    L, 01H
      JP    DISP
```

```
;initialize the count and sum
CONT:
      LD    C, 01H            ;register C will be our counter
      LD    HL, 0000H         ;HL will hold our sum total

;initialize final comparison value to even number
      LD    B, (ORIG)         ;load register B with input value
      LD    A, B
      AND   01H
      JP    NZ, LOOP2
      INC   B                 ;register B will hold the incremented even
;representation of the input value. The actual ;input value is stored permanently in
ORIG.

;loop to add the odd integers from 1 up to the input value
LOOP2:
      LD    E, C
      ADD   HL, DE
      INC   C
      INC   C                 ;increment to the next odd number
      LD    A, B              ;add our upper (even) limit to A
      CP    C                 ;compare current count with A
      JP    NZ, LOOP2         ;continue looping if we C and A weren't equal

;add one more if odd for inclusive
      LD    A, (ORIG)
      AND   01H
      JP    Z, DISP
      LD    E, C
      ADD   HL, DE

;setup for sum display
DISP:
      LD    A, H
      LD    (DISPV+0), A
      LD    A, L
      LD    (DISPV+1), A
      CALL  HEXTO7
;the sum will be displayed after the jump when KBRD is called

;end of first loop, so we jump back to capture input and potentially terminate
      JP    LOOP1

;return control to the ROM-based monitor and return to initial data entry mode
END:
      JP    0000H

;define storage locations
ORIG: DEFS  1                 ;the permanent storage location for the input
```

High-Level to Low-Level Comparison

A high-level language can do essentially the same things a low-level language can, it's just a number of steps further away from direct communication with the processor. However, being further away, its simple functionality can be greatly improved. Therefore, the power of the high-level program is significantly greater than that of the low-level program (interface, error catching, etc.).

Of course, the low-level implementation is significantly faster. Additionally, the low-level language did not require too many more lines of code to accomplish the same task.

Where the high-level language had more friendly and safer structured looping mechanisms, loops had to be mimicked using jump statements (which is essentially what a high-level loop is) in the low-level implementation.

Conclusions

The range of the sums was from 01H to 4000H. The implementations for the high and low level versions of this program were different only in interface implementation. The high-level clearly had more safety features and was more intuitive, but the low-level was far quicker and more straight forward (once you knew how to use it).