EGCP 2110-01 Microprocessors Laboratory #7

Stopwatch

Prepared By:
Alex Laird
Collin Barrett

on

Saturday, October 24th, 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 a few syntax and runtime errors, which we resolved quickly. Upon presentation, our code ran correctly.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Created by: Alex Laird and Collin Barrett
;Date: Oct. 19, 2009
;Class: Microprocessors
;Lab 7: Stopwatch
;Purpose: To implement a stop watch program using the Z-80
;         microprocessor and the circuitry implemented in
;         Lab 6.
;Overview: The first switch is START. If it is active, the
;          stop watch will start counting. If it is inactive,
;          the stop watch display will halt immediately. The
;          second switch is RESET. It is only functional when
;          START is not active. If RESET is set active when
;          START is inactive, the display will drop back to
;          zeros.
;          The B register is used to hold the current time
;          on the stop watch for intermediate steps and when
;          the stop watch is paused.
;          The Z80 processor clock runs at  1.789773 MHz.
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

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

;set memory locations, then clear all registers to 00H
RESET:
      LD    HL, 83E9H
      LD    (COUNT), HL
      LD    A, 03H
      LD    (FIX), A
      LD    A, 00H             ;the initial value sent out should be zeros
      LD    B, A               ;clear all registers
      LD    C, A
      LD    D, A
      LD    E, A
      LD    H, A
      LD    L, A
;wait until START is set active
WAIT:
      OUT   (0C0H), A
      IN    A, (0C0H)
      BIT   0, A               ;the first switch is START
      JP    NZ, CHKRST
      BIT   1, A               ;the second switch is RESET
      JP    NZ, RESET
      JP    WAIT
;ensure that reset is not active before starting
CHKRST:
      BIT   1, A
      JP    NZ, RESET
      JP    LDCOUNT

;pause until START is set active again
PAUSE:
      LD    A, B
      OUT   (0C0H), A
      IN    A, (0C0H)
      BIT   0, A
      JP    NZ, START
      BIT   1, A
      JP    NZ, RESET
      JP    PAUSE

;load the counter with COUNT, number of loops in 1 second
LDCOUNT:
      LD    HL, (COUNT)       ;T-states: 16
;loop for 1 second while START is active
START:
      IN    A, (0C0H)         ;T-states: 11
      BIT   0, A              ;T-states: 8
;pause if START is set inactive, but keep states
      JP    Z, PAUSE          ;T-states: 10
      DEC   HL                ;T-states: 6
      LD    A, H              ;T-states: 4
      OR    L                 ;T-states: 4
      JP    NZ, START         ;T-states: 10
;increment the time now that 1 second is reached
      LD    A, B              ;T-states: 4
      INC   A                 ;T-states: 4
      LD    B, A              ;T-states: 4
;check for tens rollover
TEN:
      AND   0AH               ;T-states: 7
      CP    0AH               ;T-states: 4
      LD    A, B              ;T-states: 4
      JP    NZ, SIXTY         ;T-states: 10
      ADD   A, 06H            ;T-states: 7
;check for sixty rollover
SIXTY:
      CP    60H               ;T-states: 4
      JP    NZ, FINAL         ;T-states: 10
      LD    A, 00H            ;T-states: 7
;output, adjust COUNT with FIX, and return to 1 second loop
FINAL:
      OUT   (0C0H), A         ;T-states: 12
      PUSH  AF                ;T-states: 10
      LD    A, (FIX)          ;T-states: 13
      LD    C, A              ;T-states: 4
      LD    B, 00H            ;T-states: 7
;load count for next 1 second loop
      LD    HL, (COUNT)       ;T-states: 16
;clear the carry
      XOR   A                 ;T-states: 4
;subtract FIX for time spent in the incrementation
      SBC   HL, BC            ;T-states: 15
      POP   AF                ;T-states: 10
      LD    B, A              ;T-states: 4
      JP    START             ;T-states: 10

;define storage locations
VARS   ORG  1D00H             ;start variable storage at 1D00H
COUNT: DEFS 2                 ;the number of times to loop
FIX:   DEFS 1                 ;the correction after increment
```

Program Overview

The circuit designed in Lab 6 was used to produce Binary Coded Decimal (BCD) output, receive input from switches on the trainer board, and communicate with a program running on the Z80 microprocessor.

The program we designed and ran on the Z80 microprocessor continually reads input from the two lowest-byte switches on the trainer board. When the first switch is set active, this gives the program the START command, causing the stopwatch to start incrementing once per second. While START is active, the second switch is ignored. If the START switch is set inactive, the second switch sends the RESET command, resetting all variables and states. If START is set inactive while the stopwatch is running, the program goes into a paused state. When the START switch is set active again, it continues counting exactly where it left off.

The biggest process in writing this program was implementing a delay that would cause the program to hold for an entire second before incrementing each time. To do this, we analyzed the T-state delays given by each command within our loop and calculated the number of loops needed to delay for one second.

T-State Analysis

To generate a properly delayed loop, we added the T-states within the loop together to achieve this result of T_L.

$T_L = 11 + 8 + 10 + 6 + 4 + 4 + 10 = 53$

From T_L, we can calculate the number of loops needed to delay for one second. We can calculate this from our knowledge that the Z80 microprocessor clock runs at a frequency of 1.789773 MHz. The number of loops necessary is stored in the COUNT variable.

$53/1{,}789{,}773 = 2.96 * 10^{-5}x$

$= 1$ second

$x = 1/(2.96 * 10^{-5})$

$= 33{,}769$

$= 83E9H$

Outside the loop, in the incrementation step that gets executed once each second, T_0 is the delay of the loop

$T_0 = 4 + 4 + 4 + 7 + 4 + 4 + 10 + 7 + 4 + 10 + 7 + 12 + 10 + 13 + 4 + 7 + 16 + 4 + 15 + 10 + 4 + 10+16$
$= 186$

$T_0/T_L \approx 3.51$, so we know that when the incrementation step happens, approximately 3.51 extra loops have happened. We rounded this down and set FIX to 3, subtracting our FIX off of COUNT each time incrementation step happens.

Conclusions

This lab provided an example of the way delays can be used and specifically timed delays can be generated on any given processor. The lab was both insightful and a lot of fun to implement.