**Experiment #6: Count Up/Down Timer – One More Time!**

**ECE 367 – Microprocessor Design (Spring 2013)**

**PROFESSOR:** Robert Becker

**T.A.:** Chenjie Tang

MWF – 10:00AM – 11:50PM

T Lab: 8:00AM – 10:50AM

Prepared by: Mitchell Hedditch

UIN: 677318273

Date Prepared: Wednesday, March 7th, 2013

Date Submitted: Thursday, March 8th, 2013

1. Logic Diagram
2. Schematic Diagram
   1. See attached sheet following this page.
3. 9S12 Assembler Program

; University of Illinois at Chicago, Dept. of Electrical and Computer Engineering

; ECE 367 -Microprocessor-Based Design

; Semester: Spring 2013

; Experiment Title: Count Up/Down Timer - One More Time!

; Experiment Description: This system is a timer that is capable of starting and

; pausing operation, as well as reversing the count direction.

; It will count from 00 to 99 or 99 to 00 and when it reaches 00

; again, it will blink 3 times and then reset the system. You

; can enter number values on the keypad at any time to change the

; timing values.

; Date: 2/21/2013

; Updated: 3/7/2013

; Version: 1

; Programmer: Mitchell Hedditch

; Lab Session: Tuesday 8AM-10:50AM

; Define symbolic constants

REGBAS EQU $0000 ; REGISTER BLOCK STARTS AT $0000

PortA EQU $0000 ; PortA address (relative to Regbase i.e. offset)

DDRA EQU $0002 ; PortA Data Direction control register offset

PortM EQU $0250 ; PortM offset (actual address of PortM)

DDRM EQU $0252 ; PortM Data Direction control register offset

PortT EQU $0240 ; PortT offset (actual address of PortT)

DDRT EQU $0242 ; Actual Data Direction Register for PortT

PortE EQU $0008 ; PortE LABEL (XIRQ' INTERRUPT)

; TIMER SYMBOLIC CONSTANTS

TSCR1 EQU $0046 ; TIMER SYSTEM CONTROL REGISTER - WITH FAST FLAGS

TSCR2 EQU $004D ; TIMER SYSTEM CONTROL REGISTER 2 - NO FAST FLAGS

TFLG1 EQU $004E ; TIMER INTERRUPT FLAG1 REGISTER

TFLG2 EQU $004F ; TIMER INTERRUPT FLAG2 REGISTER

TIOS EQU $0040 ; TIMER INTERRUPT OUTPUT COMPARE

TCNT EQU $0044 ; TIMER COUNTER REGISTER - 16 BIT, INPUT CAPTURE/OUTPUT COMPARE REQUIRED

TC0 EQU $0050 ; TIME I/O COMPARE SELECT 0 REGISTER TO LOCATION $50 HEX

TC1 EQU $0052 ; TIME I/O COMPARE SELECT 1 REGISTER TO LOCATION $52 HEX

TIE EQU $004C ; TIMER TCi INTERRUPT ENABLE REGISTER

; INTERRUPT CONSTANTS

IRQCR EQU $001E ; IRQ CONTROL REGISTER ADDRESS LABEL

;UNKNOWN

INITRG EQU $0011

INITRM EQU $0010

PLLCTL EQU $003A

; CLOCKS

CLKSEL EQU $0039

CRGFLG EQU $0037

SYNR EQU $0034

REFDV EQU $0035

COPCTL EQU $003C ; COMPUTER OPERATING PROPERLY CONTROL LOCATION

TEST EQU $3800 ; DEFINE LOCATION FOR TEST BYTE STORAGE FOR DEBUGGING

SAVE\_X EQU $3802 ; Defines location for the storage of the X index register

SAVE\_Y EQU $3804 ; Defines location for the storage of the Y index register

DIR\_FLAG EQU $3806 ; DEFINES LOCATION FOR STORAGE OF COUNTER DIRECTION FLAG

; FOR INTERRUPTS FLAG = 0->COUNT UP; 1->COUNT DOWN

PAUSE EQU $3808 ; DEFINES LOCATION FOR STORAGE OF START/PAUSE FLAG

; FLAG = 0->PAUSE; 1->COUNT

TMR\_FLAG EQU $3810 ; DEFINES LOCATION FOR STORAGE OF TIMER FLAG

; FLAG= 0->NOTHING; 1->TIMER FIRED

INVALID\_KEY EQU $3812 ; DEFINES LOCATION FOR STORAGE OF TIMER FLAG 2

; FLAG= 0->NOTHING; 1->TIMER FIRED

TIME\_COUNT EQU $3814 ; MEM ADDRESS TO STORE TIME FOR SECONDS

XIRQ\_FLAG EQU $3816 ; PAUSE FOR XIRQ (1 MSEC)

NUM\_FLAG EQU $3818 ; A FLAG THAT GOES TO 1 IF A KEY IS PRESSED ON THE PAD

CUR\_PAD\_VAL EQU $3820 ; USED TO HOUSE THE VALUE FOR THE CURRENT KEYPAD ITERATION

COUNT\_VAL EQU $3822 ; STORE THE COUNT VALUE HERE

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; The ORG statment below is followed by variable definitions

; THIS IS THE BEGINNING SETUP CODE

;

ORG $3800 ; Beginning of RAM for Variables

;

; The main code begins here. Note the START Label

;

ORG $4000 ; Beginning of Flash EEPROM

START LDS #$3FC0 ; Top of the Stack

SEI ; Turn Off Interrupts

MOVB #$00, INITRG ; I/O and Control Registers Start at $0000

MOVB #$39, INITRM ; RAM ends at $3FFF

;

; We Need To Set Up The PLL So that the E-Clock = 24MHz

;

BCLR CLKSEL,$80 ; disengage PLL from system

BSET PLLCTL,$40 ; turn on PLL

MOVB #$2,SYNR ; set PLL multiplier

MOVB #$0,REFDV ; set PLL divider

NOP ; No OP

NOP ; NO OP

PLP BRCLR CRGFLG,$08,PLP ; while (!(crg.crgflg.bit.lock==1))

BSET CLKSEL,$80 ; engage PLL

CLI ; TURN ON ALL INTERRUPTS

;

; End of setup code. You will always need the above setup code for every experiment

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; Begin Code

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; Initialize the 68HC11

LDY #REGBAS ; Initialize register base address

; Note that Regbas = $0000 so now <Y> = $0000

SEI ; TURN OFF INTERRUPTS

; INITIALIZE ALL SYSTEM PORTS/INTERRUPTS/DDRS/FLAGS/ETC

; SETUP S BIT ON INTERRUPTS

MOVB #$C0, IRQCR ; TURN ON IRQ' INTERRUPT AND SET TO EDGE TRIGGERED

ANDCC #$BF ; SET THE X-BIT TO USE XIRQ' AS A STANDARD INTERRUPT

; SETUP THE DATA DIRECTON REGISTERS AND INITIALIZE PORT A & PORT T

MOVB #$03,DDRM ; SET PortM PINS 0&1 TO OUTBOUND

MOVB #$00,PortM ; SET ALL PortM PINS TO LOW (0)

MOVB #$FF,DDRT ; SET ALL PortT PINS TO OUTBOUND

MOVB #$00,PortT ; SET ALL PortT PINS TO LOW

; SET UP TIMER COUNT INFORMATION AND PRESCALE INITIALIZE THE COUNTER

MOVB #$06,TSCR2 ; CONFIGURE PRESCALE FACTOR 64

MOVB #$01,TIOS ; ENABLE OC0 FOR OUTPUT COMPARE

MOVB #$90,TSCR1 ; ENABLE TCNT & FAST FLAGS CLEAR

MOVB #$01,TIE ; ENABLE TC1 INTERRUPT

LDD TCNT ; FIRST GET CURRENT TCNT

ADDD #3750 ; INCREMENT TCNT COUNT BY 3750 AND STORE INTO TC0

STD TC0 ; WE WILL HAVE A SUCCESSFUL COMPARE IN 375 CLICKS

MOVB #$01,TFLG1 ; OF TCNT. BETTER BE SURE FLAG C0F IS CLEAR TO START

JSR INITIALIZE ; INITIALIZE ALL OF OUR VARIABLES, FLAGS, ETC.

; ALL VARIABLES ARE INITIALIZED SO WE'RE READY FOR INTERRUPTS

CLI ; TURN ON INTERRUPTS

POLL: MOVB #$00,INVALID\_KEY ; RESET INVALID KEY FLAG

MOVB #$00,NUM\_FLAG ; CLEAR THE NUM FLAG TO WAIT FOR A NEW KEY

JSR GET\_KEY ; CHECK THE KEYPAD FOR A PRESSED VALUE

BRCLR NUM\_FLAG,$01,NO\_KEY ; IF NO KEY HAS BEEN PRESSED THEN MOVE ON THE THE NO\_KEY LINE

JSR CHECK\_KEY ; CHECK TO SEE IF THE KEY IS VALID

BRSET INVALID\_KEY,$01,POLL ; GO BACK AND POLL AGAIN IF WE'VE GOT A BAD KEY

JSR LOAD\_NUMBER ; IF A KEY HAS BEEN PRESSED THEN LOAD THE NEW NUMBER

MOVB #$00,NUM\_FLAG ; CLEAR THE NUM FLAG TO WAIT FOR A NEW KEY

NO\_KEY BRSET PAUSE,$01,POLL ; WAIT AT POLL WHILE THE IRQ' (PAUSE) INTERRUPT FLAG IS SET

BRCLR PortE,$01,\* ; BRANCH HERE UNTIL THE XIRQ PORT IS HIGH AGAIN

BRCLR TMR\_FLAG,$01,POLL ; IF THE TIME FLAG ISN'T SET BRANCH BACK TO POLL

BRSET DIR\_FLAG,$01,CDOWN ; IF THE DIRECTION FLAG IS SET, THEN COUNT DOWN

JSR COUNT\_UP ; INCREMENT THE COUNT VALUE

BRA UPDATE\_DISP ; BRANCE TO CONTINUE

CDOWN JSR COUNT\_DOWN ; DECREMENT THE COUNT VALUE

UPDATE\_DISP JSR PREP\_VALS ; PREPARE OUR COUNT VALUE FOR OUTPUT

JSR UPDATE\_ONES\_DISPLAY ; UPDATE THE ONES DISPLAY LED

JSR UPDATE\_TENS\_DISPLAY ; UPDATE THE TENS DISPLAY LED

MOVB #$00,TMR\_FLAG ; CLEAR THE TIMER FLAG

LDD COUNT\_VAL ; LOAD THE COUNT VALUE INTO D

CPD #$00 ; CHECK TO SEE IF WE'RE AT ZERO

BNE CONTINUE ; IF WE'RE NOT AT ZERO THEN CONTINUE

JSR BLINK ; BLINK 3 TIMES IF WE'RE AT ZERO

JSR INITIALIZE ; RESTART OUR SYSTEM

CONTINUE BRA POLL ; GO BACK START PROCESSING AT POLL AGAIN!

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; PROGRAM INITIALIZATION

INITIALIZE: MOVB #$01,PAUSE ; INITIALIZE IN THE SYSTEM IN PAUSE MODE

MOVB #$00,DIR\_FLAG ; INITIALIZE THE SYSTEM IN COUNT UP MODE

MOVB #$00,TMR\_FLAG ; INITIALIZE THE TIMER FLAG TO LOW

LDD #$0000 ; INITIALIZE THE COUNT TO 0

STD COUNT\_VAL ; STORE THE VALUE OF D TO MEMORY

MOVB #$00,TIME\_COUNT ; SET TIME\_COUNT TO 0

MOVB #$00,NUM\_FLAG ; SET NUM\_FLAG TO 0 TO

MOVB #$01,PortM ; TURN TENS ENABLE ON

MOVB #$79,PortT ; OUTPUT AN E ON THE LED

MOVB #$02,PortM ; TURN ONES ENABLE ON

MOVB #$37,PortT ; OUTPUT AN N ON THE LED

MOVB #$00,INVALID\_KEY ; RESET INVALID KEY FLAG

RTS ; RETURN FROM SUBROUTINE

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; PURPOSE: TO RETRIEVE A PRESSED KEY FROM A MATRIX KEYBOARD, IF THIS ACTION HAPPENS, SET A FLAG

; AND STORE THE VALUE

GET\_KEY: MOVB #$00,PortM ; TURN THE LATCH ENABLES OFF!!

LDX #KP\_VALUE ; LOAD X WITH MEM ADDRESS FOR KP\_VALUE

STX CUR\_PAD\_VAL ; STORE THE ADDRESS OF THE FIRST KEYPAD VALUE

LDX #ROW ; LOAD X WITH THE INITIAL VALUE AT THE ROW ADDRESS

LDY #COLUMN ; LOAD Y WITH THE INITIAL VALUE AT THE COLUMN ADDRESS

; NOW WE BEGIN OUR LOOPING

NEXT\_ROW LDAA 1,X+ ; LOAD ACCUM A WITH CURRENT ROW VALUE POST INCREMENT

NEXT\_COLUMN LDAB 1,Y+ ; LOAD ACCUM Y WITH CURRENT COLUMN VALUE POST INCREMENT

STAA PortT ; SET THE CURRENT ROW TO HIGH VALUE

NOP ; WAIT SOME TIME FOR PIN TO GO HI

NOP ; WAIT SOME TIME FOR PIN TO GO HI

NOP ; WAIT SOME TIME FOR PIN TO GO HI

CMPB PortM ; COMPARE THE VALUE IN B TO PortM

BEQ KEY\_PRESSED ; IF THE KEY IS PRESSED THEN MAKE IT SO!

PSHA ; PUSH ONTO THE STACK OR IT WILL BE LOST

PSHB ; PUSH B ONTO THE STACK OR IT WILL BE LOST

LDD CUR\_PAD\_VAL ; LOAD THE CUR\_PAD\_VAL INTO D

ADDD #1 ; ADD 1 TO D

STD CUR\_PAD\_VAL ; STORE D BACK INTO THE PAD VALUE

PULB ; GET B BACK FROM THE STACK FIRST

PULA ; NOW RESTORE A FROM THE STACK

CPY #COLUMN+4 ; CHECK TO SEE IF WE'RE AT THE END OF THE COLUMNS

BNE NEXT\_COLUMN ; IF NOT, THEN GO BACK AND TRY NEXT COLUMN

LDY #COLUMN ; IF WE ARE THEN RESET THE COLUMNS

CPX #ROW+4 ; CHECK TO SEE IF WE'RE AT THE END OF THE ROWS

BNE NEXT\_ROW ; IF WE'RE NOT AT END OF ROWS, GO TO NEXT ROW

RTS ; RETURN FROM THE SUBROUTINE IF WE'VE PROCESS ALL ROWS AND COLUMNS

KEY\_PRESSED MOVB #$01,NUM\_FLAG ; SET NUM\_FLAG SINCE A NUMBER WAS PRESSED

JSR KEY\_RELEASE ; NOW WE NEED TO WAIT UNTIL THE KEYS ARE RELEASED

RTS ; RETURN FROM SUBROUTINE

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; PURPOSE: WAIT UNTIL A PRESSED KEY IS RELEASED TO ELIMINATE BOUNCE AND DOUBLE PRESSING

KEY\_RELEASE: MOVB #$F0,PortT ; SET ROWS 4,5,6,7 OF PortT TO HIGH

NOP ; SHORT TIME WAITING FOR PINS TO GO HIGH

BRCLR PortM,$3C,FINISH ; WHEN COLUMN 1-4 (PM2-PM5) IS CLEAR THEN ALL KEYS

; HAVE BEEN RELEASED

BRA KEY\_RELEASE ; BRANCH BACK TO KEY RELEASE

FINISH RTS ; RETURN FROM SUBROUTINE

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; PURPOSE: TO CHECK AND MAKE SURE WE HAVE A VALID KEY PRESSED

CHECK\_KEY: LDX CUR\_PAD\_VAL ; GET THE CURRENT KEYPAD VALUE ADDRESS

LDAA X ; LOAD THE KEYPAD VALUE ADDRESS

CMPA #$09 ; WAS THIS KEY AN INVALID KEY?

BGT INVALID ; IF IT WAS THEN SET THE FLAG

RTS ; RETURN FROM SUBROUTINE

INVALID MOVB #$01,INVALID\_KEY ; SET THE INVALID KEY FLAG

RTS ; RETURN FROM SUBROUTINE

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; PURPOSE: THIS SUBROUTINE IS USED TO LOAD A NEW DIGIT INTO THE LED AND THE COUNT VALUE

LOAD\_NUMBER: JSR PREP\_VALS ; LETS LET THE PREP\_VALS SUB SPLIT THE NUMBER

TFR Y,X ; TRANSFER THE ONES VALUE INTO THE X INDEX

LDD #$000A ; LOAD D WITH DECIMAL 10

EMUL ; MULTIPLY THE TENS VALUE BY 10 AND PLACE IN D

STD COUNT\_VAL ; STORE OUR TENS VALUE INTO COUNT\_VAL

LDY CUR\_PAD\_VAL ; LOAD THE EFFECTIVE ADDRESS INTO Y (NEW VALUE)

LDAA #$00 ; CLEAR A OUT BY WRITING ZEROS TO IT

LDAB Y ; LOAD A WITH THE ADDRESS IN Y

ADDD COUNT\_VAL ; ADD OUR KEYPAD VALUE TO THE TENS VALUE

STD COUNT\_VAL ; STORE THE NEW VALUE INTO COUNT\_VAL

JSR PREP\_VALS ; GO BACK TO THE PREP\_VALS SUB TO GET READY FOR OUTPUT

JSR UPDATE\_ONES\_DISPLAY ; UPDATE THE ONES DISPLAY LED

JSR UPDATE\_TENS\_DISPLAY ; UPDATE THE TENS DISPLAY LED

RTS ; RETURN FROM SUBROUTINE

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; NOTE: DO NOT CHANGE THIS SUBROUTINE UNLESS YOU MODIFY LOAD\_NUMBER!!

; PURPOSE: TAKE THE VALUE IN COUNT\_VAL AND PARSE ITS ONES AND TENS DIGIT INTO THE X AND Y INDEX

; FOR USE IN THE DISPLAYS

PREP\_VALS: LDD COUNT\_VAL ; LOAD THE COUNT VALUE INTO D

CPD #$000A ; COMPARE X TO 10

LBLO UNDR\_TEN ; IF IT'S LESS THEN 10 MAKE IT ZERO

LDX #0010 ; PLACE TEN IN D

IDIV ; DIVIDE OUR NUMBER BY 10

TFR D,Y ; TRANSFER THE REMAINDER INTO Y

RTS ; RETURN FROM SUBROUTINE

UNDR\_TEN LDX 0 ; LOAD ZERO INTO X

TFR D,Y ; WE LEAVE Y AS IT IS

RTS ; RETURN FROM SUBROUTINE

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; PURPOSE: TAKE THE VALUE IN THE X INDEX AND DISPLAY IT IN THE TENS LCD

UPDATE\_TENS\_DISPLAY:

MOVB #$01,PortM ; ENABLE THE TENS LATCH

TFR X,B ; MOVE X INTO B

LDX #TABLE ; LOAD THE BEGINNING ADDRESS OF TABLE INTO X

ABX ; ADD B TO THE X INDEX

LDAA X ; LOAD THE ADDRESS OF INDEX X INTO ACCUM A

STAA PortT ; OUTPUT IT TO THE LED

MOVB #$00,PortM ; DISABLE THE LATCHES

RTS ; RETURN FROM SUBROUTINE

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; PURPOSE: TAKE THE VALUE IN THE Y INDEX AND DISPLAY IT IN THE ONES LCD

UPDATE\_ONES\_DISPLAY:

MOVB #$02,PortM ; ENABLE THE TENS LATCH

TFR Y,B ; MOVE Y INTO B SO WE CAN USE Y

LDY #TABLE ; LOAD THE BEGINNING ADDRESS OF TABLE INTO Y

ABY ; ADD B TO THE Y INDEX

LDAA Y ; LOAD THE ADDRESS OF INDEX Y INTO ACCUM A

STAA PortT ; OUTPUT IT TO THE LED

MOVB #$00,PortM ; ENABLE THE TENS LATCH

RTS ; RETURN FROM SUBROUTINE

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; PURPOSE INCREMENT THE VALUE OF COUNT\_VAL UNTIL WE REACH 99, THEN RESET TO 00

COUNT\_UP:

LDY COUNT\_VAL ; LOAD THE COUNT VALUE INTO Y

INY ; INCREMENT THE NUMBER BY 1

CPY #100 ; COMPARE IT TO 100

BEQ U\_RESET ; IF WE'RE ATT 100, THEN RESET THE NUMBER

STY COUNT\_VAL ; STORE THE COUNT VALUE BACK INTO Y

RTS ; RETURN FROM SUBROUTINE

U\_RESET LDY 0 ; RESET Y TO 0

STY COUNT\_VAL ; STORE THE COUNT VALUE BACK INTO Y

RTS ; RETURN FROM SUBROUTINE

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; PURPOSE: DECREMENT THE VALUE OF COUNT\_VAL UNTIL WE REACH 0, THEN RESET TO 99

COUNT\_DOWN:

LDY COUNT\_VAL ; LOAD THE COUNT VALUE INTO Y

CPY #0 ; COMPARE IT TO 0

BEQ D\_RESET ; IF WE'RE AT ZERO, THEN RESET IT

DEY ; DECREMENT THE NUMBER BY Y

STY COUNT\_VAL ; STORE THE COUNT VALUE BACK INTO Y

RTS ; RETURN FROM SUBROUTINE

D\_RESET LDY 99 ; RESET Y TO 0

STY COUNT\_VAL ; STORE THE COUNT VALUE BACK INTO Y

RTS ; RETURN FROM SUBROUTINE

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

BLINK: LDY #0 ; SET Y TO ZERO

BLINKING BSET PortM,$03 ; SET BOTH Portm PINS TO HIGH FOR OUTPUT

LDAA #$00 ; LOAD A WITH ZERO SO DISPLAY GOES BLANK

STAA PortT ; OUTPUT VALUE IN A TO PortT

BRCLR TMR\_FLAG,$01,\* ; WAIT A SECOND HERE

MOVB #$00,TMR\_FLAG ; CLEAR THE TIMER FLAG

LDAA TABLE ; LOAD A WITH TABLE VALUE TO DISPLAY 00

STAA PortT ; OUTPUT VALUE IN A TO PortT

BRCLR TMR\_FLAG,$01,\* ; WAIT A SECOND HERE

MOVB #$00,TMR\_FLAG ; CLEAR THE TIMER FLAG

INY ; INCREMENT Y

CPY #3 ; SEE IF WE'VE BLINKED 3 TIMES

BNE BLINKING ; IF NOT THEN BLINK AGAIN!

RTS ; RETURN FROM SUBROUTINE

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; TC0 INTERRUPT SUBROUTINE

ISR\_TC0: LDD TC0 ; INTERRUPT READS THE FLAG SO THIS WRITE CLEARS THE FLAG

ADDD #3750 ; ADD THE EQUIVALENT .1 SECOND CNT TO REGISTER D

STD TC0 ; UPDATE TC0 MEMORY TO NEW VALUE

BRSET PAUSE,$01,PAUSED ; IF PAUSED DON'T UPDATE TIME\_COUNT!!

PSHA ; SAVE A ON THE STACK

LDAA TIME\_COUNT ; LOAD THE VALUE OF TIME\_COUNT INTO A

CMPA #100 ; IF TIME\_COUNT = 100 THEN WE HAVE 1 SECOND

BNE TMR\_UPDATE ; IF WE'RE NOT AT 100 YET, GOTO TMR\_UPDATE LINE

MOVB #$01,TMR\_FLAG ; TURN ON OUR TIMER FLAG

MOVB #$00,TIME\_COUNT ; RESET OUR TIMER COUNT BACK TO ZERO

PULA ; PUL A BACK OFF THE STACK

PAUSED RTI ; RETURN FROM THE INTERRUPT

TMR\_UPDATE ADDA #01 ; INCREMENT THE VALUE IN A

STAA TIME\_COUNT ; STORE A BACK INTO TIME\_COUNT

PULA ; PULL A BACK OFF THE STACK

RTI ; RETURN FROM THE INTERRUPT

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; IRQ' INTERRUPT SUBROUTINE

ISR\_IRQ: COM PAUSE ; TOGGLE THE START/PAUSE FLAG

RTI ; RETURN FROM INTERRUPT

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; XIRQ' INTERRUPT SUBROUTINE

ISR\_XIRQ: COM DIR\_FLAG ; TOGGLE THE DIRECTION FLAG

RTI ; RETURN FROM INTERRUPT

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ORG $FFF2 ; IRQ' VECTOR ADDRESS

FDB ISR\_IRQ ; ISR\_IRQ IS A LABEL FOR THE INTERRUPT SUBROUTINE

ORG $FFF4 ; XIRQ' VECTOR ADDRESS

FDB ISR\_XIRQ ; ISR\_XIRQ' IS A LABEL FOR THE INTERRUPT SUBROUTINE

ORG $FFEE ; VECTOR ADDRESS FOR TC0 INTERRUPT

FDB ISR\_TC0 ; ISR\_TIMER IS A LABEL FOR THE INTERRUPT SUBROUTINE

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

; Have the Assembler put the solution data in the look-up table

ORG $5500 ; The look-up table is at $5000

TABLE: DC.B $3F, $06, $5B, $4F, $66 ; Define data table of mappings to each of the

DC.B $6D, $7D, $07, $7F, $6F ; segments of the 7-segment LED displays

DC.B $5C, $3C, $39, $5F, $7B ; Memory locations correspond to their values

DC.B $71 ; i.e. $5500 = 0, $5501 = 1, etc

ROW: DC.B $10, $20, $40, $80 ; PortT OUTPUT VALUES FOR MATRIX KEYPAD ROWS

COLUMN: DC.B $04, $08, $10, $20 ; PortM INPUT VALUES FOR MATRIX KEYPDA COLUMNS

KP\_VALUE: DC.B $01, $02, $03, $0A ; KEY VALUES FROM KEYPAD FOR ITERATING THROUGH

DC.B $04, $05, $06, $0B

DC.B $07, $08, $09, $0C

DC.B $00, $0F, $0E, $0D

; End of code

; Define Power-On Reset Interrupt Vector - Required for all programs!

; AGAIN - OP CODES are at column 9

ORG $FFFE ; $FFFE, $FFFF = Power-On Reset Int. Vector Location

FDB START ; Specify instruction to execute on power up

END ; (Optional) End of source code

; Labels start in the first column (left most column = colunm 1)

; OP CODES are at column 9

; COMMENTS follow a ";" symbol

; Blank lines are allowed (Makes the code more readable)

1. USER MANUAL
   1. Start Up
      1. To start this system, use the USB to miniUSB cable and connect to computer. If you have a USB-outlet adapter, power may be supplied this way as well.
      2. Verify the system is in “Run” mode (the switch on the microcontroller board).
      3. Press the reset button to begin (left-most black button).
   2. Operation
      1. After pressing the reset button, the system will display “EN” letting you know that it is awaiting input. The system is initialized in the “Paused” and “Count Up” modes.
      2. Enter a number using the numeric portion of the keypad by pressing and releasing a button. Each time a new keypad number is entered, the system will take the value entered and move it into the ones place LED. If the ones place LED has a value in it, it will be moved to the tens place LED. A new number can be entered at any time during operation.
      3. If no values are input, the system will initialize to 00 and begin counting up after the Start/Pause button is pressed.
      4. The system operates using the “Start/Pause” button which is the right black button. Each time the “Start/Pause” button is pressed; the system will either Start/Resume or Pause the countdown.
      5. The direction of the count (up or down) may be changed at any time by pressing and releasing the middle black button.
      6. When the shot clock reaches “00” the display will blink “00” 3 times and then reset to “EN” indicating it is ready for input again.
      7. The system can be restarted/reinitialized at any time by simply pressing the reset button during operation.
      8. Button Layout:
         1. Left: Reset
         2. Middle: Change Direction
         3. Right: Start/Pause
   3. Shut Down
      1. To shut the system down, disconnect the power source (USB cable) from the breadboard.
2. Conclusion.
   1. How well does your project meet the specifications?
      1. It meets project requirements as explained on the ECE 367 website for experiment #6.
   2. What were the most difficult issues in realizing the system?
      1. My most difficult issue was the keypad, due to the fact that I implemented my own keypad subroutine. I had a major bug that took me a great deal of debugging time to find.
   3. Were you able to add extra features? If so, explain them.
      1. Instead of a blank startup, the system starts up with “EN”, short for “Enter Number.”
      2. Rather than use the lengthy code for the matrix, I instituted an iterative subroutine and a table with values that correspond to the matrix keypad layout. With this method, the system cycles through each row, column by column. It can easily be adapted to different sized keypads. This will allow for much faster implementation in the future and rapid code-reuse.
   4. What would you have done differently if you were to do this project again?
      1. I probably would have added better user prompts to let the user know where in the program they are.
   5. What did you learn from working on this project?
      1. The first thing that I learned from this project is how to set up the timer counter code, including the fast flags and cycle frequency. Using flags I created I was able to signal the main program when the interrupt subroutine determined a second passed by.
      2. I learned that you must be careful while implementing new subroutines not to overwrite accumulators that you are currently using! This was the reason my keypad would not function correctly. I fixed it by pushing accum a and b onto the stack and then pulling them back off.