**Objective**

The objective of this lab was to drive two pairs of LEDs at independent rates. Cooperative multitasking was used and each task was designed as a finite state machine. The LEDs blinked in the following order:

G\_LED ON and R\_LED OFF

G\_LED OFF and R\_LED OFF

G\_LED OFF and R\_LED ON

G\_LED OFF and R\_LED OFF

G\_LED ON and R\_LED ON

G\_LED OFF and R\_LED OFF

**Tasks**

Task 1: Pattern One

This task established the blinking order of the first pair of LEDs. There are seven states. State 0 ensures that all LEDs are off when Task 1 is initialized. States one through six turn on the LEDs in the pattern stated above.

Task 2: Timing of first LED pair

This task counts down the first LED pair. This task has 3 states: state 0, state 1, and state 2. State 0 initializes Task 2 and sets the next state. State 1 initializes COUNT\_1, decrements COUNT\_1 and stores it. State 2 tests to see if COUNT\_1 is already zero, counts down COUNT\_1, decrements COUNT\_1 and stores it. If it is done, the done flag is set, if it is not done then it returns.

Task 3: Delay

This task sets the delay time of 1.00 ms. There are two states, State 0 and State 1. State 0 initializes Task 3 and State 1 sets the time delay of 1.00 ms.

Task 4: Pattern Two

This task established the blinking order of the second pair of LEDs. There are seven states and they are the same as Task 4, but applied to the second pair of LEDs and the associated variables.

Task 5: Timing of second LED pair

This task counts down the second LED pair. This task has 3 states and they are the same as Task 2, but applied to the second LED pair and the associated variables.

**Inter-task Communication Variables**

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable** | **Description** | **Set Location** | **Clear location** |
| DONE\_1 | The varibale communicates when to switch to the next step in TASK 1. | TASK\_2, setdone\_1 | TASK\_2,t2state0 |
| DONE\_2 | The varibale communicates when to switch to the next step in TASK 4. It is constantly checked in TASK\_4. | TASK\_5,setdone\_2 | TASK\_5,t4state0 |

**Finite State Machines**

See Attached.

**Source Code**

;Assembler equates

PORTS = $00D6 ; output port for LEDs

DDRS = $00D7

LED\_MSK\_1 = 0b00000011 ; LED\_1 output pins

R\_LED\_1 = 0b00000001 ; red LED\_1 output pin

G\_LED\_1 = 0b00000010 ; green LED\_1 output pin

LED\_MSK\_2 = 0b00001100

R\_LED\_2 = 0b00000100

G\_LED\_2 = 0b00001000

; RAM area

.area bss

TICKS\_1:: .blkb 2 ; use this space to explain each of your variables

COUNT\_1:: .blkb 2

DONE\_1:: .blkb 1

TICKS\_2:: .blkb 2

COUNT\_2:: .blkb 2

DONE\_2:: .blkb 1

t1state:: .blkb 1

t2state:: .blkb 1

t3state:: .blkb 1

t4state:: .blkb 1

t5state:: .blkb 1

;code area

.area text

;

;==============================================================================

;

; main program

\_main::

clr t1state ; initialize all tasks to state0

clr t2state

clr t3state

; Normally no code other than that to clear the state variables and call the tasks

; repeatedly should be in your main program. However, this week we will make an

; exception. The following code will allow the user to set TICKS\_1 and TICKS\_2 in

; the debugger.

movw #200, TICKS\_1 ; set default for TICKS\_1

movw #500, TICKS\_2 ; set default for TICKS\_2

bgnd ; stop in DEBUGGER to allow user to alter TICKS

TOP: ;bgnd

jsr TASK\_1

;bgnd

jsr TASK\_2

;bgnd

jsr TASK\_3

;bgnd

jsr TASK\_4

jsr TASK\_5

bra TOP

; end main

;=============================================================================

;

; Subroutine TASK\_1 ; pattern\_1

TASK\_1: ldaa t1state ; get current t1state and branch accordingly

beq t1state0

deca

beq t1state1

deca

beq t1state2

deca

beq t1state3

deca

beq t1state4

deca

beq t1state5

deca

beq t1state6

rts ; undefined state - do nothing but return

t1state0: ; init TASK\_1

bclr PORTS, LED\_MSK\_1 ; ensure that LEDs are off when initialized

bset DDRS, LED\_MSK\_1 ; set LED\_MSK\_1 pins as PORTS outputs

movb #$01, t1state ; set next state

rts

t1state1: ; G, not R

bset PORTS, G\_LED\_1 ; set state1 pattern on LEDs

tst DONE\_1 ; check TASK\_1 done flag

beq exit\_t1s1 ; if not done, return

movb #$02, t1state ; if done, set next state

exit\_t1s1:

rts

t1state2: ; not G, not R

bclr PORTS, G\_LED\_1 ; set state2 pattern on LEDs

tst DONE\_1 ; check TASK\_1 done flag

beq exit\_t1s2 ; if not done, return

movb #$03, t1state ; if done, set next state

exit\_t1s2:

rts

t1state3: ; not G, R

bset PORTS, R\_LED\_1 ; set state3 pattern on LEDs

tst DONE\_1 ; check TASK\_1 done flag

;bgnd

beq exit\_t1s3 ; if not done, return

movb #$04, t1state ; if done, set next state

exit\_t1s3:

rts

t1state4: ; not G, not R

bclr PORTS, LED\_MSK\_1 ; set state4 pattern on LEDs

tst DONE\_1 ; check TASK\_1 done flag

beq exit\_t1s4 ; if not done, return

movb #$05, t1state ; if done, set next state

exit\_t1s4:

rts

t1state5: ; G, R

bset PORTS, LED\_MSK\_1 ; set state5 pattern on LEDs

tst DONE\_1 ; check TASK\_1 done flag

;bgnd

beq exit\_t1s5 ; if not done, return

movb #$06, t1state ; if done, set next state

exit\_t1s5:

rts

t1state6: ; not G, not R

bclr PORTS, LED\_MSK\_1 ; set state6 pattern on LEDs

tst DONE\_1 ; check TASK\_1 done flag

beq exit\_t1s6 ; if not done, return

movb #$01, t1state ; if done, set next state

exit\_t1s6:

rts

; end TASK\_1

;

;=============================================================================

;

; Subroutine TASK\_2 ; count down LED\_1 pair

TASK\_2: ldaa t2state ; get current t2state and branch accordingly

beq t2state0

deca

beq t2state1

deca

beq t2state2

rts ; undefined state - do nothing but return

t2state0: ; initialization for TASK\_2

clr DONE\_1

movb #$01, t2state ; set next state

t2state1: ; (re)initialize COUNT\_1

movw TICKS\_1, COUNT\_1

ldx COUNT\_1

dex ; decrement COUNT\_1

stx COUNT\_1 ; store decremented COUNT\_1

clr DONE\_1

movb #$02, t2state ; set next state

;rts

t2state2: ; count down COUNT\_1

ldx COUNT\_1

beq setdone\_1 ; test to see if COUNT\_1 is already zero

dex ; decrement COUNT\_1

stx COUNT\_1 ; store decremented COUNT\_1

bne exit\_t2s2 ; if not done, return

setdone\_1:

movb #$01, DONE\_1 ; if done, set DONE\_1 flag

movb #$01, t2state ; set next state

exit\_t2s2:

rts

; end TASK\_2

;

;=============================================================================

; Subroutine TASK\_4 ; pattern\_2

TASK\_4: ldaa t4state ; get current t4state and branch accordingly

beq t4state0

deca

beq t4state1

deca

beq t4state2

deca

beq t4state3

deca

beq t4state4

deca

beq t4state5

deca

beq t4state6

rts ; undefined state - do nothing but return

t4state0: ; init TASK\_1

bclr PORTS, LED\_MSK\_2 ; ensure that LEDs are off when initialized

bset DDRS, LED\_MSK\_2 ; set LED\_MSK\_1 pins as PORTS outputs

movb #$01, t4state ; set next state

rts

t4state1: ; G, not R

bset PORTS, G\_LED\_2 ; set state1 pattern on LEDs

tst DONE\_2 ; check TASK\_1 done flag

beq exit\_t4s1 ; if not done, return

movb #$02, t4state ; if done, set next state

exit\_t4s1:

rts

t4state2: ; not G, not R

bclr PORTS, G\_LED\_2 ; set state2 pattern on LEDs

tst DONE\_2 ; check TASK\_1 done flag

beq exit\_t4s2 ; if not done, return

movb #$03, t4state ; if done, set next state

exit\_t4s2:

rts

t4state3: ; not G, R

bset PORTS, R\_LED\_2 ; set state3 pattern on LEDs

tst DONE\_2 ; check TASK\_1 done flag

;bgnd

beq exit\_t4s3 ; if not done, return

movb #$04, t4state ; if done, set next state

exit\_t4s3:

rts

t4state4: ; not G, not R

bclr PORTS, LED\_MSK\_2 ; set state4 pattern on LEDs

tst DONE\_2 ; check TASK\_1 done flag

beq exit\_t4s4 ; if not done, return

movb #$05, t4state ; if done, set next state

exit\_t4s4:

rts

t4state5: ; G, R

bset PORTS, LED\_MSK\_2 ; set state5 pattern on LEDs

tst DONE\_2 ; check TASK\_1 done flag

;bgnd

beq exit\_t4s5 ; if not done, return

movb #$06, t4state ; if done, set next state

exit\_t4s5:

rts

t4state6: ; not G, not R

bclr PORTS, LED\_MSK\_2 ; set state6 pattern on LEDs

tst DONE\_2 ; check TASK\_1 done flag

beq exit\_t4s6 ; if not done, return

movb #$01, t4state ; if done, set next state

exit\_t4s6:

rts

; end TASK\_4

;

;=============================================================================

;

; Subroutine TASK\_5 ; count down LED\_2 pair

TASK\_5: ldaa t5state ; get current t5state and branch accordingly

beq t5state0

deca

beq t5state1

deca

beq t5state2

rts ; undefined state - do nothing but return

t5state0: ; initialization for TASK\_2

clr DONE\_2

movb #$01, t5state ; set next state

t5state1: ; (re)initialize COUNT\_2

movw TICKS\_2, COUNT\_2

ldx COUNT\_2

dex ; decrement COUNT\_2

stx COUNT\_2 ; store decremented COUNT\_2

clr DONE\_2

movb #$02, t5state ; set next state

;rts

t5state2: ; count down COUNT\_2

ldx COUNT\_2

beq setdone\_2 ; test to see if COUNT\_2 is already zero

dex ; decrement COUNT\_2

stx COUNT\_2 ; store decremented COUNT\_2

bne exit\_t5s2 ; if not done, return

setdone\_2:

movb #$01, DONE\_2 ; if done, set DONE\_2 flag

movb #$01, t5state ; set next state

exit\_t5s2:

rts

; end TASK\_2

;

;=============================================================================

;

; Subroutine TASK\_3 ; delay 1.00ms

TASK\_3: ldaa t3state ; get current t3state and branch accordingly

beq t3state0

deca

beq t3state1

rts ; undefined state - do nothing but return

t3state0: ; initialization for TASK\_3

; no initialization required

movb #$01, t3state ; set next state

rts

t3state1:

jsr DELAY\_1ms

rts

; end TASK\_3

;

;=============================================================================

;

; Subroutine Delay\_1ms delays for ~1.00ms

;

DELAY\_1ms:

ldy #$0262

INNER: ; inside loop

cpy #0

beq EXIT

dey

bra INNER

EXIT:

rts ; exit DELAY\_1ms

; end subroutine DELAY\_1ms

;

;==============================================================================

.area interrupt\_vectors (abs)

.org $FFFE ; at reset vector location

.word \_\_start ; load starting address