x86 Assembly RoboTrike Project (EE/CS 51)

Sung Hoon Choi

California Institute of Technology

Table of Contents

0. Functional Specifications
   - Functional Specification for Remote module
   - Functional Specification for Motors module
1. Description for Code Updates
2. Shared Program Files
   - Events.asm
     : Contains necessary functions to initialize and handle EventQueue
   - Events.inc
     : Contains the definitions for the all events of Robotrike.
   - ChipSel.asm
     : Contains the functions used to initialize the chip selects for RoboTrike.
   - ChipSel.inc
     : Contains the definitions for the ChipSel.asm.
   - InitPB.asm
     : Contains the functions for initializing parallel port B of RoboTrike.
   - InitPB.inc
     : Contains the definitions for the InitPB.asm.
   - INTFunc.asm
     : Contains the interrupt vector initialization functions.
   - INTFunc.inc
     : Contains the definitions for general interrupts and the interrupt controller.
   - Timer1.asm
     : Contains the functions for handling the timer1 and timer1 events in Robotrike
   - Timer2.asm
     : Contains the functions for handling the timer2 and timer2 events in Robotrike
   - Timer.inc
     : Contains the definitions for the Timer.asm.
   - general.inc
     : Contains definitions for general constants.
3. Individual Program Files
   - HW2
     - Converts.asm
       : Converts 16-bit signed or unsigned values into decimal or hexadecimal and save them as strings.

    - Converts.inc
      : Contains definitions for the constants of Converts.asm.
- HW3
    - Queue.asm
      :Contains the functions initializing and handling queues.
    - Queue.inc
      :Contains definitions for the constants of Queue.asm
- HW4
    - Display.asm
      : Contains the functions necessary for displaying number or strings on the 7-Segment LED digits.
    - Display.inc
      :Contains definitions for the constants of Display.asm.
- HW5
    - Keypad.asm
      : Contains the functions necessary for handling keypad inputs.
    - Keypad.inc
      :Contains definitions for the constants of Keypad.asm.
- HW6
    - Motors.asm
      : Contains the functions necessary for handling DC motors and laser.
    - Motors.inc
      :Contains definitions for the constants of Motors.asm
- HW7
    - Serial.asm
      :Contains the functions necessary for handling serial communications.
    - Serial.inc
      :Contains definitions for the constants of Serial.asm
- HW8
    - Parser.asm
      :Contains the functions necessary for parsing the commands.
    - Parser.inc
      :Contains definitions for the constants of Parser.asm
- HW9
    - Remote.asm
      :Contains the main function which initializes and runs the Remote module.
    - Remote.inc
      :Contains definitions for the constants of Remote.asm

- HW10
    - MtrMain.asm
      :Contains the main function which initializes and runs the Motors module.
    - MtrMain.inc
      :Contains definitions for the constants of MtrMain.asm

Functional Specification

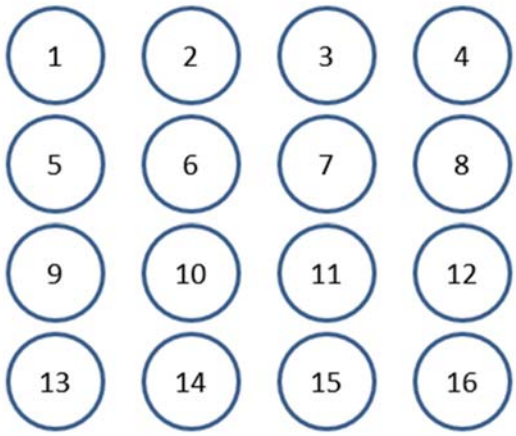
For Remote Module

Sung Hoon Choi

Description: The remote module consists of eight 7-segment LED digit displays and 16 (4 keys per row) keys. When the user pushes a key, the corresponding command will be transmitted to the Motors module through the serial channel. Also, every time a command is transmitted to the Motors module, the changed status(speed, direction, laser) will be displayed on the LED digits. If the command did not change any status of the RoboTrike, LED digits will keep displaying the old message. It also displays various error messages in case of errors and resets in case of the system failure. See the Error Handling section for details. Note that the auto-repeat is implemented for the keys, so the user can keep pressing a key to execute a command for a large number of times.

Global Variables: None

Input: Major inputs are given through the keys. There are 16 (4x4) keys on the Remote module. See the following descriptions for each key.



Key Number	Key Name	Description
1	Set Half Maximum Speed	Set the RoboTrike's speed to the half of maximum value.
2	Stop	Stop the RoboTrike's motors.
3	Change direction 45 deg clockwise	Alter the direction 45 degrees clockwise.
4	Change direction 45 deg Anti-clockwise	Alter the direction 45 degrees anti-clockwise.
5	Increment Speed (Delicate)	Increment the speed a little bit. (Increments 1.5% the MAX speed each time a key is pressed. Use Auto-Repeat if you want a huge change in speed. The speed will not increment if it reached the MAX speed)
6	Increment Speed (Rough)	Increment the speed considerably. (Increments 7.5% the MAX speed each time a key is pressed. Use Auto-Repeat if you want a huge change in speed. The speed will not increment if it reached the MAX speed)
7	Decrement Speed (Delicate)	Decrement the speed a little bit. (Decrements 1.5% the MAX speed each time a key is pressed. Use Auto-Repeat if you want a huge change in speed. The speed will not decrement if it reached 0.)
8	Decrement Speed (Rough)	Decrement the speed considerably. (Decrements 7.5% the MAX speed each time a key is pressed. Use Auto-Repeat if you want a huge change in speed. The speed will not decrement if it reached 0.)
9	Turn Turret 45 deg Clockwise (To be implemented)	Turn the turret's angle 45 degrees clockwise. (To be implemented)
10	Turn Turret 45 deg	Turn the turret's angle 45 degrees anti-

	Anti-Clockwise (To be implemented)	clockwise. (To be implemented)
11	Turn Turret Elevation 30 deg up (To be implemented)	Turn the turret's elevation 30 degrees upward. (To be implemented)
12	Turn Turret Elevation 60 deg up (To be implemented)	Turn the turret's elevation 60 degrees upward. (To be implemented)
13	Turn Turret Elevation 30 deg down (To be implemented)	Turn the turret's elevation 30 degrees downward. (To be implemented)
14	Turn Turret Elevation 60 deg down (To be implemented)	Turn the turret's elevation 60 degrees downward. (To be implemented)
15	Turn Laser On	Turn the laser on.
16	Turn Laser Off	Turn the laser off.

The Remote module also receives Error Message strings from the Motors module as an input. For actual error messages, please see the Error Handling section.

Output: Eight 7-segment LED digits are used to display the recently updated status of

the RoboTrike. For example, let's say the current direction of the RoboTrike is +45 degrees(from the center line of the Motors Module). Now, you pressed Key#3. Then, the LED digits will display 'D00090'. Note that the LED digits always display 6 characters. Thus, zeroes will be padded if the number has less than 5 digits. The format of the status message on LED is

"Status Type" + "Status Value"

While there are three status types:

S - Speed

D - Direction

L – Laser

Note that the range of speed is [0, 65534] and the range of direction is [0, 359]. For the laser, Status Value of 1 indicates that the laser is turned on while Status Value of 0 indicates that the laser is turned off.

Also, it displays 'SEri_Err' for a serial error, 'PArS_Err' for a parser error, and 'SYS_FAIL' for the system failure. See Error Handling section for details.

User Interface: Users can send commands to the Motors module by pressing the keys on the Remote module. Once the user presses the key, the altered status of the RoboTrike will be displayed on the LED digits. If the command did not change any status of RoboTrike, the message displayed on the LED digits won't change. The user can order multiple commands without pressing keys tremendous times since the Auto-Repeat is implemented on the keys. The user can keep pressing the key and the same command will be transmitted about 5 times per second. The user can notice if there's an error on the RoboTrike by reading the message on LED digits. The LED digits display proper error messages for each type of error. If the system failure occurs, the system will reset automatically.

Error Handling: Error messages will be displayed on the LED digits in case of errors.

Error Type	Error Message	Error Detail	How to resolve it
System Failure	'SYS_FAIL'	System Failure occurs when the EventQueue of RoboTrike is full.	The system will reset automatically once there's a system failure.
Serial Error	'SEri_Err'	Serial Error occurs when there's an error on the serial communication.	Check the parity and baud rates and fix them if they are incorrect.
Parser Error	'PArS_Err'	Parser Error occurs when the parser function generates an error. It means that an illegal	Check the parity and baud rates and fix them if they are incorrect.

		command has been ordered.	

Algorithms: 7-segment LED digits use multiplexing to display proper data on LED.

Data Structures: Event queues were used to handle the events happening on the RoboTrike.

Also, the RemoteDisplayBuffer was used to handle the strings to be displayed on LED digits.

Limitations: Since there is no feedback in this system, there's no way to tell if the system moved the correct distance or direction. Also, since the RoboTrike uses 7-segment LED digits, it cannot display certain types of characters.

Known Bugs: None

Special Notes: None

Functional Specification

For Motors Module

Sung Hoon Choi

Description: The Motors module has three omni-wheels driven by DC motors, one laser diode, and one turret driven by a servo motor and a stepper motor. The omni-wheels allow the RoboTrike to move all directions without rotating. It uses a vector calculation to move with the desired speed and direction. The wheels are placed 120 degrees from each other on the circular board. The turret's rotation is controlled by a stepper motor while the elevation is controlled by a servo motor. However, the turret's movements are not implemented.

It receives commands from the Remote module through the serial cable. It sends back the altered status to the Remote module if there's any. It also transmits various error messages to the Remote module in case of errors. See the Error Handling Section for details.

Global Variables: None

Input: Inputs are given through the serial cable (channel). The inputs are the commands in a string format. Thus, the Motors module parses the received command and executes a proper action.

Output: The outputs of the Motors module are the DC motors(wheels), laser, and serial.

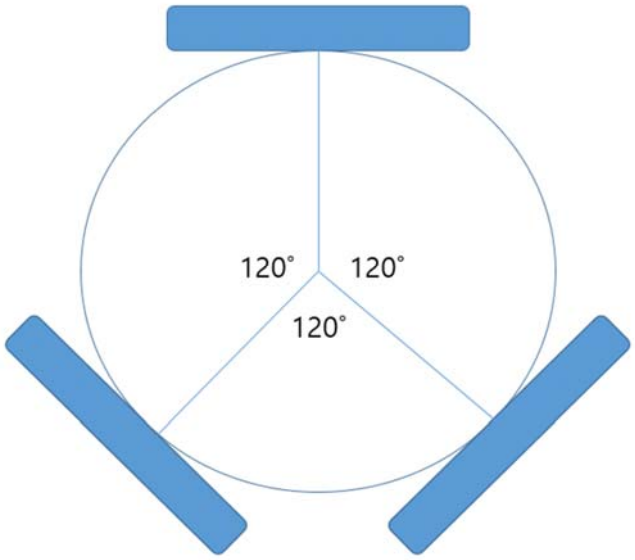
Three DC motors are placed 120 degrees from each other on the circular board. Since the RoboTrike uses omni-wheels, a vector calculation is applied to move in a desired speed and direction. It calculates the pulse width for each speed and direction by a vector calculation and then uses PWM(Pulse

Width Modulation) to actually control the motors. The equation used to obtain the pulse width is:

$$PulseWidth = \ Force_X * velocity * Cos(Angle) + \ Force_Y * velocity * Sin(Angle)$$

The Motors module turns the laser on if it gets the command to turn it on. It turns the laser off if it gets the command to turn it off.

Also, it transmits the altered status of the RoboTrike back to the Remote module after executing an action. For example, if the speed has been changed by the last command, it sends the updated speed value back to the Remote module so that the status can be displayed on LED digits. See the *Functional Specification for Remote* to check actual messages shown on LED display. It also sends error messages to the Remote module in case of errors. There are three types of errors: System Failure, Serial Error, and Parser Error. See the Error Handling section for details.



(The diagram which shows how the omni-wheels are placed on the Motors module)

User Interface: The user cannot control the Motors module directly.

They should use the Remote module to control the Motors module.

Error Handling: Errors messages will be sent to the Remote Module in case of errors.

Error Type	Error Message	Error Detail	How to resolve it
System Failure	'SYS_FAIL'	System Failure occurs when the EventQueue of RoboTrike is full.	The system will reset automatically once there's a system failure.
Serial Error	'SEri_Err'	Serial Error occurs when there's an error on the serial communication.	Check the parity and baud rates and fix them if they are incorrect.
Parser Error	'PArS_Err'	Parser Error occurs when the parser function generates an error. It means that an illegal command has been ordered.	Check the parity and baud rates and fix them if they are incorrect.

Algorithms: It uses the vector calculation and Pulse Width Modulation to control the DC motors. Also, it uses the Finite State Machine to parse the received commands.

Data Structures: StatusBuffer and MotorTxBuffer were used to transmit the updated status value and error messages to the Remote module. Also, various states of the Finite State Machine were used for parsing the commands.

Limitations: Since there is no feedback in this system, there's no way to tell if the system moved the correct distance or direction. Also, since the Motor module is connected to the Remote module by a wire(serial cable), the RoboTrike cannot move freely.

Known Bugs: None

Special Notes: Wireless communication between the Motors module and the Remote module should be implemented to enable free movements of the RoboTrike.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                              ;
;                                Events.asm                                    ;
;                               Sunghoon Choi                                  ;
;                                                                              ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; Description:
;     This file contains the functions necessary to initialize and handle EventQueue.
; Table of Contents:
;    InitEventQueue      -    Initializes the EventQueue.
;    EnqueueEvent        -    Enqueues an event to EventQueue.
;    DequeueEvent        -    Dequeues an event from EventQueue.
;    CheckSystemFail     -    Checks if system failure has occurred.
; Revision History:
;    12/01/2016      Sunghoon Choi         Created
;    12/02/2016      Sunghoon Choi        Initial Compilation
;    12/02/2016      Sunghoon Choi        Corrected minor syntax errors.
;    12/02/2016      Sunghoon Choi        Added comments


$INCLUDE(general.inc)          ;Include the .inc file which contains general constants
$INCLUDE(queue.inc)            ;Include the .inc file which contains constants for Queue.asm

EXTRN   QueueFull:NEAR         ;Import QueueFull to check if EventQueue is full.
EXTRN   QueueEmpty:NEAR        ;Import QueueEmpty to check if EventQueue is empty.
EXTRN   QueueInit:NEAR         ;Import QueueInit to initialize EventQueue.
EXTRN   Enqueue:NEAR           ;Import Enqueue to insert an event to EventQueue.
EXTRN   Dequeue:NEAR           ;Import Dequeue to pop an event from EventQueue.

CGROUP GROUP CODE
DGROUP GROUP DATA
CODE SEGMENT PUBLIC 'CODE'

        ASSUME  CS:CGROUP, DS:DGROUP


; InitEventQueue
;
; Description:
;     It initializes the EventQueue which contains Event Types and Event Values.
; Operation:
;    It sets SI to the address of EventQueue and set BL to WORD_SIZE. Then, it calls
;    QueueInit to initialize the EventQueue. Finally, it resets SystemFailFlag and exits.
; Arguments:
;    None
; Return Value:
;   None
; Local Variables:
;   SI(EventQueueAddress)    -    The address of the EventQueue.
;    BL(EventQueueType)      -    The size of the elements of EventQueue.
; Shared Variables:
;    EventQueue        -    [Write]    -    The queue which contains event type and event value for
;                                            each events
;    SystemFailFlag    -    [Write]    -    Indicates whether system failure has occurred or not.
; Global Variables:
;   None
; Input:
;   None
; Output:
;   None
; Error Handling:
```

```
;   None
; Algorithms:
;   None
; Data Structures:
;   None
; Registers Changed:
;   SI, BX
; Limitations:
;   None
; Known bugs:
;   None
; Special Notes:
;   None
; Author:
;   Sunghoon Choi
; Revision History:
;     12/01/2016    Sunghoon Choi      Created
;     12/02/2016    Sunghoon Choi      Initial Compilation
;     12/02/2016    Sunghoon Choi      Updated documentation

InitEventQueue  PROC    NEAR
                PUBLIC  InitEventQueue

InitializeEventQueue:
    MOV SI, OFFSET(EventQueue)          ;Sets SI to the offset of EventQueue to
    initialize it
    MOV BL, WORD_SIZE                   ;Set the EventQueue to a WORD queue.

    CALL QueueInit                      ;Call QueueInit to initialize the EventQueue.
ResetSystemFailFlag:
    MOV SystemFailFlag, 0               ;Reset SystemFailFlag.
EndInitEventQueue:
    RET                                 ;End of InitEventQueue.
InitEventQueue  ENDP




; EnqueueEvent
;
; Description:
;     It enqueues an event to EventQueue.
; Operation:
;    It first sets SI to the offset of the EventQueue. Then, it calls QueueFull to
check if
;    the EventQueue is full. If it is not full, it inserts the argument event to the
;    EventQueue by calling Enqueue. If it is full, set the SystemFailFlag and exit.
; Arguments:
;    AH(Event Type)    -    The type of the event to be enqueued
;    AL(Event Value)   -    The value of the event to be enqueued
; Return Value:
;   None
; Local Variables:
;    SI(EventQueueAddress)  -    The address of EventQueue
; Shared Variables:
;    EventQueue      -   [Write]  -  The queue which contains event type and event
value for
;                                    each events
;    SystemFailFlag  -   [Write]  -  Indicates whether a system failure has occurred.
; Global Variables:
;   None
; Input:
;   None
; Output:
;   None
; Error Handling:
```

```
;   None
; Algorithms:
;   None
; Data Structures:
;   None
; Registers Changed:
;   AX, BX, SI, Flags
; Limitations:
;   None
; Known bugs:
;   None
; Special Notes:
;   None
; Author:
;   Sunghoon Choi
; Revision History:
;     12/01/2016    Sunghoon Choi      Created
;     12/02/2016    Sunghoon Choi      Initial Compilation
;     12/02/2016    Sunghoon Choi      Updated documentation

EnqueueEvent    PROC    NEAR
                PUBLIC  EnqueueEvent
    PUSH SI                              ;Save the address of EventQueue
CheckEventQueueFull:
    MOV SI, OFFSET(EventQueue)           ;Set SI to the offset of EventQueue to check if
    it
                                         ;is full.
    PUSHA                                ;push registers since QueueFull changes them.
    CALL QueueFull                       ;Check if EventQueue is full.
    POPA                                 ;Retrieve registers since QueueFull is done.
    JNZ  InsertEvent                     ;If EventQueue is not full, go insert the event.
    ;JZ  SetSystemFailFlag               ;If it is full, set the system failure flag.
SetSystemFailFlag:
    MOV SystemFailFlag, TRUE             ;Set the SystemFailFlag since the EventQueue is
    full.
    JMP EndEnqueueEvent                  ;Exit EnqueueEvent procedure.
InsertEvent:
    CALL Enqueue                         ;Insert the argument event on EventQueue.
EndEnqueueEvent:
    POP SI                               ;Retrieve the address of EventQueue.
    RET                                  ;End of EnqueueEvent.
EnqueueEvent    ENDP


; DequeueEvent
;
; Description:
;     It dequeues an event from EventQueue.
; Operation:
;    It first sets SI to the offset of the EventQueue. Then, it calls QueueEmpty to
    check if
;    the EventQueue is empty. If EventQueue is empty, it sets the carry flag and exit.
    If
;    EventQueue is not empty, it dequeues an event from EventQueue and clear the carry
    flag,
;    and exits.
; Arguments:
;   None
; Return Value:
;   AH(Event Type)    -    The type of the event dequeued from EventQueue.
;    AL(Event Value)  -    The value of the event dequeued from EventQueue.
;    Carry Flag       -    Set if EventQueue is empty.
;                          Reset if EventQueue is not empty.
; Local Variables:
;    SI(EventQueueAddress)   -    The address of EventQueue
; Shared Variables:
```

```
188 ;    EventQueue      -    [Read]  -  The queue which contains event type and event value
    for
189 ;                                    each events
190 ; Global Variables:
191 ;   None
192 ; Input:
193 ;   None
194 ; Output:
195 ;   None
196 ; Error Handling:
197 ;   None
198 ; Algorithms:
199 ;   None
200 ; Data Structures:
201 ;   None
202 ; Registers Changed:
203 ;   AX, BX, SI, Flags
204 ; Limitations:
205 ;   None
206 ; Known bugs:
207 ;   None
208 ; Special Notes:
209 ;   None
210 ; Author:
211 ;   Sunghoon Choi
212 ; Revision History:
213 ;     12/01/2016    Sunghoon Choi      Created
214 ;     12/02/2016    Sunghoon Choi      Initial Compilation
215 ;     12/02/2016    Sunghoon Choi      Updated documentation
216
217 DequeueEvent    PROC    NEAR
218                 PUBLIC  DequeueEvent
219
220 CheckEventQueueEmpty:
221     MOV SI, OFFSET(EventQueue)    ;Set SI to the offset of EventQueue to check if it is
        empty
222     CALL QueueEmpty               ;Check if EventQueue is empty.
223     JZ  SetCarryFlag              ;If EventQueue is empty, go set the carrfy flag and
        exit.
224    ;JNZ PopEvent                  ;If EventQueue is not empty, dequeue an event.
225 PopEvent:
226     CALL Dequeue                  ;CALL Dequeue to dequeue an event from the EventQueue.
227     CLC                           ;If an event was dequeued, clear the carry flag.
228     JMP EndDequeueEvent           ;Exit the procedure.
229 SetCarryFlag:
230     STC                           ;Since EventQueue is empty, set the carry flag
231 EndDequeueEvent:
232     RET                           ;End of DequeueEvent
233 DequeueEvent    ENDP
234
235
236 ; CheckSystemFail
237 ;
238 ; Description:
239 ;     It checks if a system failure has occurred
240 ; Operation:
241 ;    It returns the value of SystemFailFlag in AX.
242 ; Arguments:
243 ;   None
244 ; Return Value:
245 ;   AX(SystemFailFlag)    -    Indicates whether a system failure has occurred or not.
246 ; Local Variables:
247 ;    None
248 ; Shared Variables:
249 ;    SystemFailFlag       -  [Read] -  Indicates whether a system failure has occurred
    or not.
```

```
; Global Variables:
;   None
; Input:
;   None
; Output:
;   None
; Error Handling:
;   None
; Algorithms:
;   None
; Data Structures:
;   None
; Registers Changed:
;   AX
; Limitations:
;   None
; Known bugs:
;   None
; Special Notes:
;   None
; Author:
;   Sunghoon Choi
; Revision History:
;     12/01/2016    Sunghoon Choi      Created
;     12/02/2016    Sunghoon Choi      Initial Compilation
;     12/02/2016    Sunghoon Choi      Updated documentation

CheckSystemFail     PROC    NEAR
                    PUBLIC  CheckSystemFail

    MOV AX, SystemFailFlag        ;Return the value of SystemFailFlag in AX
    RET                           ;End of CheckSystemFail

CheckSystemFail ENDP


CODE ENDS


DATA    SEGMENT PUBLIC  'DATA'

EventQueue  QueueModule     <>  ;The queue which contains event type and event value for
                                ;each events
SystemFailFlag  DW  ?           ;Flag that indicates whether a system failure has
occurred
                                ;or not.

DATA  ENDS

END
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                              ;
;                                  Events.inc                                  ;
;                                 Sunghoon Choi                                ;
;                                                                              ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Description:
;  This file contains the definitions for the all events of Robotrike.
;
; Revision History:
;     10/30/2016  Sunghoon Choi    Created
;     11/3/2016   Sunghoon Choi    Updated documentation(comments) for constants.
;     11/19/2016  Sunghoon Choi    Events for Serial have been updated.
;     12/02/2016  Sunghoon Choi    Added BLANK_EVENT

BLANK_EVENT           EQU 00H     ;No event. Exists as a blank slot
                                  ;for the jump table in Remote.asm
KEY_EVENT             EQU 01H     ;Keypad pressed event
SERIAL_RECEIVED_EVENT EQU 02H     ;Serial Data Received Event
SERIAL_ERROR_EVENT    EQU 03H     ;Serial Error Event
```

```
NAME    ChipSel
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                            ;
;                                    ChipSel                                 ;
;                                 Sunghoon Choi                              ;
;                                                                            ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Description:
;    This file contains the functions used to initialize the chip selects for
;    80188 based RoboTrike system.
;
; Table of Contents:
;    InitCS - Initialize the peripheral chip selects
;
; Revision History:
;    10/25/2016    Sunghoon Choi    Created
;    10/28/2016    Sunghoon Choi    Updated documentation


$INCLUDE (ChipSel.inc)  ;include the file which contains constants for
                        ;ChipSel.asm

CGROUP GROUP CODE

CODE SEGMENT PUBLIC 'CODE'

        ASSUME  CS:CGROUP



; InitCS
;
; Description:       Initialize the Peripheral Chip Selects on the 80188.
;
; Operation:         Write the initial values to the PACS and MPCS registers.
;
; Arguments:         None.
; Return Value:      None.
;
; Local Variables:   None.
; Shared Variables:  None.
; Global Variables:  None.
;
; Input:             None.
; Output:            None.
;
; Error Handling:    None.
;
; Algorithms:        None.
; Data Structures:   None.
;
; Registers Changed: AX, DX
; Stack Depth:       0 words
;
; Limitations:       None
; Known bugs:        None
; Special Notes:     None
; Author:            Glen George, Sunghoon Choi
;
; Revision History:  07/12/2010  Last modified by Glen Geroge
;                    10/28/2016  PCS and MPCS register variable's name
;                                changed by Sunghoon Choi
InitCS  PROC    NEAR
        PUBLIC  InitCS

```

```
67
68          MOV     DX, PCSCtrl     ;write to PACS register
69          MOV     AX, PACSval
70          OUT     DX, AL
71
72          MOV     DX, MPCSctrl    ;write to MPCS register
73          MOV     AX, MPCSval
74          OUT     DX, AL
75
76
77          RET                     ;done so return
78
79
80    InitCS  ENDP
81
82    CODE ENDS
83
84    END
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                              ;
;                                   ChipSel.inc                                ;
;                                  Sunghoon Choi                               ;
;                                                                              ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


; Description:
;  This file contains the definitions for the ChipSel.asm
;
; Revision History:
;    10/25/2016    Sunghoon Choi    Created
;    10/28/2016    Sunghoon Choi    Updated documentation(comments) for constants.

PCSctrl EQU 0FFA4H      ;The address of PCS control register
MPCSctrl EQU 0FFA8H     ;The address of MPCS register

PACSval EQU 00003H      ;PCS base at 0,    3 wait states
                        ;0000000000------  starts at address 0
                        ;----------000---  reserved
                        ;-------------0--  wait for RDY inputs
                        ;--------------11  3 wait states
                        ;PCS in I/O space, use PCS5/6, 3 wait states
MPCSval EQU 00183H      ;0---------000---  reserved
                        ;-0000001--------  MCS is 8KB
                        ;--------1-------  output PCS5/PCS6
                        ;---------0------  PCS in I/O space
                        ;-------------0--  wait for RDY inputs
                        ;--------------11  3 wait states
```

```
NAME InitPB
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                            ;
;                                   InitPB                                   ;
;                                Sunghoon Choi                               ;
;                                                                            ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Description:
;  This file contains the functions for initializing parallel port B of RoboTrike.
;
; Table of Contents:
;    InitParallelB      - Initialize parallel port B
;
; Revision History:
;     11/6/2016      Sunghoon Choi        Created
;     11/8/2016      Sunghoon Choi        Initial Compilation
;     11/11/2016     Sunghoon Choi        Updated documentation

$INCLUDE(InitPB.inc)    ;Include the.inc file which contains constatns for InitPB.asm

CGROUP GROUP CODE

CODE SEGMENT PUBLIC 'CODE'

        ASSUME  CS:CGROUP

InitParallelB   PROC    NEAR
                PUBLIC  InitParallelB

MOV DX, PARALLEL_B_CTRL   ;Get the address of control word for parallel port B.
MOV AL, PARALLEL_B_VAL    ;Get the configuration value for parallel port B.
OUT DX, AL                ;Configure parallel port B with the prepared settings.

RET                       ;End of InitParallelB procedure.

InitParallelB ENDP

CODE ENDS

END
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                               ;
;                                  InitPB.inc                                   ;
;                                 Sunghoon Choi                                 ;
;                                                                               ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Description:
;  This file contains the definitions for the InitPB.asm
;
; Revision History:
;    11/6/2016    Sunghoon Choi    Created
;    11/11/2016   Sunghoon Choi    Updated documentation(comments) for constants.

PARALLEL_B_CTRL     EQU 183H         ;The address of control word for parallel port B
PARALLEL_B_VAL      EQU 10000000B    ;1------- MODE SET FLAG
                                     ;-00----- MODE 0
                                     ;---0---- Port A Output
                                     ;----0--- Port C Output
                                     ;-----0-- MODE 0
                                     ;------0- Port B Output (Important)
                                     ;-------0 Port C Output


```

```
NAME  INTFunc

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                              ;
;                                  INTFunc                                     ;
;                          Interrupt related functions                         ;
;                                Sunghoon Choi                                 ;
;                                                                              ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Description:
;     This file contains the interrupt initialization functions for LEDs.
; Table of Contents:
;     ClrIRQVectors       - clears the interrupt vector table
;     IllegalEventHandler - handler for illegal (uninitialized) interrupts
;
; Revision History:
;     10/25/2016    Sunghoon Choi    Created
;     10/29/2016    Sunghoon Choi    Updated Documentation




$INCLUDE(INTFunc.INC)


CGROUP GROUP CODE

CODE SEGMENT PUBLIC 'CODE'

        ASSUME  CS:CGROUP



; ClrIRQVectors
;
; Description:      This functions installs the IllegalEventHandler for all
;                   interrupt vectors in the interrupt vector table.  Note
;                   that all 256 vectors are initialized so the code must be
;                   located above 400H.  The initialization skips  (does not
;                   initialize vectors) from vectors FIRST_RESERVED_VEC to
;                   LAST_RESERVED_VEC.
;
; Operation:        All vectors are initialized to IllegalEventHandler in a
;                   loop.  The vectors from FIRST_RESERVED_VEC to
;                   LAST_RESERVED_VEC are skipped.
;
; Arguments:        None.
; Return Value:     None.
;
; Local Variables:  CX    - vector counter.
;                   ES:SI - pointer to vector table.
; Shared Variables: None.
; Global Variables: None.
;
; Input:            None.
; Output:           None.
;
; Error Handling:   None.
;
; Algorithms:       None.
; Data Structures:  None.
;
; Registers Used:   flags, AX, CX, SI, ES
; Stack Depth:      1 word
```

```
;
; Limitations:      None
; Known bugs:       None
; Special Notes:    None
; Author:           Glen George, Sunghoon Choi
; Revision History:     07/12/2010    last modified by Glen George
;                       10/28/2016    last modified by Sunghoon Choi

ClrIRQVectors   PROC    NEAR
                PUBLIC  ClrIRQVectors


InitClrVectorLoop:              ;setup to store the same handler 256 times

        XOR     AX, AX          ;clear ES (interrupt vectors are in segment 0)
        MOV     ES, AX
        MOV     SI, 0           ;initialize SI to the first vector

        MOV     CX, NUM_IRQ_VECTORS     ;number of vectors to initialize


ClrVectorLoop:                                  ;loop clearing each vector
                                                ;check if should store the vector
        CMP     SI, 4 * FIRST_RESERVED_VEC
        JB      DoStore                         ;if before start of reserved field - store it
        CMP     SI, 4 * LAST_RESERVED_VEC
        JBE     DoneStore                       ;if in the reserved vectors - don't store it
        ;JA     DoStore                         ;otherwise past them - so do the store

DoStore:                                        ;store the vector
        MOV     ES: WORD PTR [SI], OFFSET(IllegalEventHandler)
        MOV     ES: WORD PTR [SI + 2], SEG(IllegalEventHandler)

DoneStore:                                      ;done storing the vector
        ADD     SI, 4                           ;update pointer to next vector

        LOOP    ClrVectorLoop                   ;loop until have cleared all vectors
        ;JMP    EndClrIRQVectors;and all done


EndClrIRQVectors:                               ;all done, return
        RET


ClrIRQVectors   ENDP




; IllegalEventHandler
;
; Description:      This procedure is the event handler for illegal
;                   (uninitialized) interrupts.  It does nothing - it just
;                   returns after sending a non-specific EOI (in the 80188
;                   version).
;
; Operation:        Send a non-specific EOI (80188 version only) and return.
;
; Arguments:        None.
; Return Value:     None.
;
; Local Variables:  None.
; Shared Variables: None.
; Global Variables: None.
;
; Input:            None.
```

```
; Output:            None.
;
; Error Handling:    None.
;
; Algorithms:        None.
; Data Structures:   None.
;
; Registers Changed: None
; Stack Depth:       2 words
;
; Limitations:       None
; Known bugs:        None
; Special Notes:     None
; Author:            Glen George, Sunghoon Choi
; Revision History:  07/12/2010  Last modified by Glen George
;                    10/28/2016  Last modified by Sunghoon Choi

IllegalEventHandler     PROC    NEAR


        NOP                             ;do nothing (can set breakpoint here)

        PUSH    AX                      ;save the registers
        PUSH    DX

        MOV     DX, INTCtrlrEOI         ;send a non-specific EOI to the
        MOV     AX, NonSpecEOI          ;interrupt controller to clear out
        OUT     DX, AL                  ;the interrupt that got us here

        POP     DX                      ;restore the registers
        POP     AX


        IRET                            ;and return


IllegalEventHandler     ENDP


CODE ENDS



        END
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                               ;
;                                   INTFunc.inc                                 ;
;                          Interrupt Controller Constants                       ;
;                                  Sunghoon Choi                                ;
;                                                                               ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Description:
;   This file contains the definitions for general interrupts and the interrupt
;   controller for the Microprocessor-Based Clock.
;
; Revision History:
;   12/10/2007  Glen George      initial revision
;   10/29/2016  Sunghoon Choi    Deleted unused constants.



; Interrupt Controller

; Addresses
INTCtrlrCtrl    EQU     0FF32H          ;address of interrupt controller for timer
INTCtrlrEOI     EQU     0FF22H          ;address of interrupt controller EOI register

; Register Values
INTCtrlrCVal    EQU     00001H          ;set priority for timers to 1 and enable
                                        ;000000000000----  reserved
                                        ;------------0---  enable timer interrupt
                                        ;-------------001  timer priority

TimerEOI        EQU     00008H          ;Timer EOI command (all timers same)
NonSpecEOI      EQU     08000H          ;Non-specific EOI command



; General Interrupt Definitions

FIRST_RESERVED_VEC      EQU     1       ;reserve vectors 1-3
LAST_RESERVED_VEC       EQU     3
NUM_IRQ_VECTORS         EQU     32      ;number of interrupt vectors

```

```
NAME  Timer1

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                              ;
;                                    Timer1                                    ;
;                           Timer1 related functions                           ;
;                                 Sunghoon Choi                                ;
;                                                                              ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Description:
;  This file contains the functions for handling the timer1 and timer1 events in
;  Robotrike.
;
; Table of Contents:
;    InitTimer1            - Initialize timer 1 interrupts and variables
;    InstallTimer1Handler - Install the timer 1 event handler
;    Timer1EventHandler    - Timer 1 Event Handler which calls
;                            MotorLaserEventHandler
;
; Revision History:
;     11/6/2016     Created                     -   Sunghoon Choi
;     11/9/2016     Changed Timer1 interrupt    -   Sunghoon Choi
;                   frequency to 4KHz
;     11/11/2016    Updated Documentation       -   Sunghoon Choi




$INCLUDE(Timer.inc)       ;include .inc file which contains constants for Timer

CGROUP GROUP CODE
CODE SEGMENT PUBLIC 'CODE'

        ASSUME  CS:CGROUP
        EXTRN MotorLaserEventHandler:NEAR


; InitTimer1
;
; Description:       This function initializes timer 1
;
; Operation:         Timer 1 is initialized to generate interrupts at every 0.25ms.
;                    The interrupt controller is also initialized
;                    to allow the timer interrupts.  Timer #1 is used to scale
;                    the internal clock from 2.304 MHz to 4 KHz and generate the
;                    interrupts.
;
; Arguments:         None.
; Return Value:      None.
;
; Local Variables:   None.
; Shared Variables:  None.
; Global Variables:  None.
;
; Input:             None.
; Output:            Timer #1 and the Interrupt Controller are initialized.
;
; Error Handling:    None.
;
; Algorithms:        None.
; Data Structures:   None.
;
; Registers Changed: flags, AX, DX
; Stack Depth:       0 word
```

```
;
; Limitations:      None
; Known bugs:       None
; Special Notes:    None
; Author:            Glen George, Sunghoon Choi
; Revision History:  10/11/1998  - Last modified   by Glen George
;                    11/11/2016  - Last modified   by Sunghoon Choi
InitTimer1       PROC    NEAR
                 PUBLIC  InitTimer1



        MOV     DX, Tmr1Count   ;initialize the count register to 0
        XOR     AX, AX
        OUT     DX, AL

        MOV     DX, Tmr1MaxCnt  ;setup max count for 0.25ms counts
        MOV     AX, KHZ_4_CNT   ;4KHz
        OUT     DX, AL

        MOV     DX, Tmr1Ctrl    ;setup the control register
        MOV     AX, Tmr1CtrlVal
        OUT     DX, AL


        MOV     DX, INTCtrlrCtrl;setup the interrupt control register
        MOV     AX, INTCtrlrCVal
        OUT     DX, AL

        MOV     DX, INTCtrlrEOI ;send an EOI to turn off any pending interrupts
        MOV     AX, TimerEOI
        OUT     DX, AL


        RET                     ;done so return


InitTimer1       ENDP




; InstallTimer1Handler
;
; Description:       Install the event handler for the timer1 interrupt.
;
; Operation:         The event handler address is written to the timer1
;                    interrupt vector.
;
; Arguments:         None.
; Return Value:      None.
;
; Local Variables:   None.
; Shared Variables:  None.
; Global Variables:  None.
;
; Input:             None.
; Output:            None.
;
; Error Handling:    None.
;
; Algorithms:        None.
; Data Structures:   None.
;
; Registers Changed: flags, AX, ES
```

```
; Stack Depth:       0 words
;
; Limitations:      None
; Known bugs:       None
; Special Notes:    None
; Author:            Glen George, Sunghoon Choi
; Revision History:  01/28/2002  - Last modified by Glen George
;                    11/11/2016  - Comments revised by Sunghoon Choi

InstallTimer1Handler     PROC    NEAR
                         PUBLIC  InstallTimer1Handler


        XOR     AX, AX                  ;clear ES
                                        ;(interrupt vectors are in segment 0)
        MOV     ES, AX
                                        ;store the vector
        MOV     ES: WORD PTR (4 * Tmr1Vec), OFFSET(Timer1EventHandler)
        MOV     ES: WORD PTR (4 * Tmr1Vec + 2), SEG(Timer1EventHandler)


        RET                             ;all done, return


InstallTimer1Handler     ENDP



; Timer1EventHandler
;
; Description:       This procedure is the event handler for the timer 1
;                    interrupt. It calls MotorLaserEventHandler to output to motors and
;                    laser.
;
; Operation:         First, it pushes all register and flags.
;                    Then, it calls MotorLaserEventHandler to output to the motors and
;                    laser. When MotorLaserEventHandler is done, it sends EOI to the
;                    interrupt controller.
;
; Arguments:         None.
; Return Value:      None.
;
; Local Variables:   None.
; Shared Variables:  None
; Global Variables:  None.
;
; Input:
; Output:            Motors and laser (indirectly)
;
; Error Handling:    None.
;
; Algorithms:        None.
; Data Structures:   None.
;
; Registers Changed: None
; Stack Depth:       None
;
; Limitations:      None
; Known bugs:       None
; Special Notes:    None
; Author:            Glen George, Sunghoon Choi
; Revision History:     10/11/1998 - Last modified - by Glen George
;                       11/6/2016  - Replaced the existing eventhandler with
;                                    MotorLaserEventHandler. - by Sunghoon Choi
;                       11/11/2016 - Updated documentation.  - by Sunghoon Choi

```

```
Timer1EventHandler   PROC    NEAR


        PUSHA                     ;save any registers that are used
CallEventHandlers:

     CALL MotorLaserEventHandler    ;Output to motors and laser.

TimerEventEOI:                          ;send the timer EOI to the interrupt controller
        MOV     DX, INTCtrlrEOI         ;only in 80188 version
        MOV     AX, TimerEOI
        OUT     DX, AL

EndTimerEventHandler:                   ;done taking care of the timer
        POPA                            ;restore the registers
        IRET
Timer1EventHandler ENDP


CODE ENDS

END
```

```
NAME  Timer2

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                               ;
;                                    Timer2                                     ;
;                            Timer2 related functions                           ;
;                                 Sunghoon Choi                                 ;
;                                                                               ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Description:
;  This file contains the functions for handling the timer2 and timer2 events in
;  Robotrike.
;
; Table of Contents:
;    InitTimer2           - Initialize timer 2 interrupts and variables
;    InstallTimer2Handler - Install the timer 2 event handler
;    Timer2EventHandler   - Timer 2 Event Handler which calls
;                           DisplayEventHandler and KeypadEventHandler
;
; Revision History:
;    10/25/2016     Sunghoon Choi   Created
;    10/25/2016     Sunghoon Choi   Revised Timer2EventHandler by
;                                   adding the call to DisplayEventHandler
;    10/26/2016     Sunghoon Choi   Updated documentation for
;                                   Timer2EventHandler
;    10/31/2016     Sunghoon Choi   Added the call to KeypadEventHandler
;    11/4/2016      Sunghoon Choi   Changed the file and function names
;                                   InitTimer -> InitTimer2
;                                   InstallTimerHandler -> InstallTimer2Handler
;                                   Timer.asm -> Timer2.asm



$INCLUDE(Timer.inc)       ;include .inc file which contains constants for Timer

CGROUP GROUP CODE
CODE SEGMENT PUBLIC 'CODE'

        ASSUME  CS:CGROUP
        EXTRN KeypadEventHandler:NEAR   ;The eventhandler which reads the keys from
                                        ;keypad and enqueues the key events to EventBuf.
        EXTRN DisplayEventHandler:NEAR  ;The eventhandler which outputs
                                        ;the patterns to 7-segments LED displays

; InitTimer2
;
; Description:       This function initializes timer 2
;
; Operation:         Timer 2 is initialized to generate interrupts every 1 ms.
;                    The interrupt controller is also initialized
;                    to allow the timer interrupts.  Timer #2 is used to scale
;                    the internal clock from 2 MHz to 1 KHz and generate the
;                    interrupts.
;
; Arguments:         None.
; Return Value:      None.
;
; Local Variables:   None.
; Shared Variables:  None.
; Global Variables:  None.
;
; Input:             None.
; Output:            Timer #2 and the Interrupt Controller are initialized.
;
```

```
67  ; Error Handling:    None.
68  ;
69  ; Algorithms:        None.
70  ; Data Structures:   None.
71  ;
72  ; Registers Changed: flags, AX, DX
73  ; Stack Depth:       0 word
74  ;
75  ; Limitations:      None
76  ; Known bugs:       None
77  ; Special Notes:    None
78  ; Author:            Glen George, Sunghoon Choi
79  ; Revision History:  10/11/1998  - Last modified   by Glen George
80  ;                    10/28/2016  - Last modified   by Sunghoon Choi
81  InitTimer2       PROC    NEAR
82                   PUBLIC  InitTimer2
83
84
85
86          MOV     DX, Tmr2Count   ;initialize the count register to 0
87          XOR     AX, AX
88          OUT     DX, AL
89
90          MOV     DX, Tmr2MaxCnt  ;setup max count for 1ms counts
91          MOV     AX, ONE_MS_CNT
92          OUT     DX, AL
93
94          MOV     DX, Tmr2Ctrl    ;setup the control register
95          MOV     AX, Tmr2CtrlVal
96          OUT     DX, AL
97
98
99          MOV     DX, INTCtrlrCtrl;setup the interrupt control register
100         MOV     AX, INTCtrlrCVal
101         OUT     DX, AL
102
103         MOV     DX, INTCtrlrEOI ;send an EOI to turn off any pending interrupts
104         MOV     AX, TimerEOI
105         OUT     DX, AL
106
107
108         RET                     ;done so return
109
110
111 InitTimer2       ENDP
112
113
114
115
116
117 ; InstallTimer2Handler
118 ;
119 ; Description:       Install the event handler for the timer2 interrupt.
120 ;
121 ; Operation:         The event handler address is written to the timer2
122 ;                    interrupt vector.
123 ;
124 ; Arguments:         None.
125 ; Return Value:      None.
126 ;
127 ; Local Variables:   None.
128 ; Shared Variables:  None.
129 ; Global Variables:  None.
130 ;
131 ; Input:             None.
132 ; Output:            None.
```

```
;
; Error Handling:    None.
;
; Algorithms:        None.
; Data Structures:   None.
;
; Registers Changed: flags, AX, ES
; Stack Depth:       0 words
;
; Limitations:      None
; Known bugs:       None
; Special Notes:    None
; Author:            Glen George, Sunghoon Choi
; Revision History:  01/28/2002  - Last modified by Glen George
;                    10/28/2016  - Comments revised by Sunghoon Choi

InstallTimer2Handler     PROC    NEAR
                         PUBLIC  InstallTimer2Handler


        XOR     AX, AX                  ;clear ES
                                        ;(interrupt vectors are in segment 0)
        MOV     ES, AX
                                        ;store the vector
        MOV     ES: WORD PTR (4 * Tmr2Vec), OFFSET(Timer2EventHandler)
        MOV     ES: WORD PTR (4 * Tmr2Vec + 2), SEG(Timer2EventHandler)


        RET                             ;all done, return


InstallTimer2Handler     ENDP



; Timer2EventHandler
;
; Description:       This procedure is the event handler for the timer
;                    interrupt.  It calls KeypadEventHandler to read the keys
;                    and calls DisplayEventHandler to output patterns on LEDs. .
;
; Operation:         First, it pushes all register and flags.
;                    Then, it calls KeypadEventHandler to read the keys from keypad and
;                    calls DisplayEventHandler to output the pattern values
;                    to 7-Segment LED digits.
;                    When KeypadEventHandler and DisplayEventHandler are done,
;                    it sends EOI to the interrupt controller.
;
; Arguments:         None.
; Return Value:      None.
;
; Local Variables:   None.
; Shared Variables:  None
; Global Variables:  None.
;
; Input:             Keypad
; Output:            7-Segment LED displays.
;
; Error Handling:    None.
;
; Algorithms:        None.
; Data Structures:   None.
;
; Registers Changed: None
; Stack Depth:       None
;
```

```
; Limitations:      None
; Known bugs:       None
; Special Notes:    None
; Author:            Glen George, Sunghoon Choi
; Revision History:     10/11/1998 - Last modified - by Glen George
;                       10/29/2016 - changed the EventHandler being called
;                                    to DisplayEventHandler - by Sunghoon Choi
;                       10/31/2016 - added KeypadEventHandler - by Sunghoon Choi
;                       11/4/2016  - Revised functional specification for keypad
routines.

Timer2EventHandler   PROC    NEAR


        PUSHA                     ;save any registers that are used
CallEventHandlers:

     CALL KeypadEventHandler        ;reads the keys from keypad and enqueues the key
     events
                                    ;to EventBuf.

     CALL DisplayEventHandler       ;outputs the patterns to 7-segments LED displays

TimerEventEOI:                          ;send the timer EOI to the interrupt controller
        MOV     DX, INTCtrlrEOI         ;only in 80188 version
        MOV     AX, TimerEOI
        OUT     DX, AL

EndTimerEventHandler:                   ;done taking care of the timer
        POPA                            ;restore the registers
        IRET
Timer2EventHandler ENDP


CODE ENDS

END
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                               ;
;                                    Timer.inc                                  ;
;                               Timer related functions                         ;
;                                    Sunghoon Choi                              ;
;                                                                               ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Description:
;  This file contains the definitions for the Timer.asm
;
; Revision History:
;    10/25/2016     Sunghoon Choi     Created
;    10/28/2016     Sunghoon Choi     Updated documentation(comments) for constants.
;    11/6/2016      Sunghoon Choi     Added timer1 related addresses and values.

; Timer Defintions

; Addresses
Tmr1Ctrl        EQU     0FF5EH          ;address of Timer 1 Control Register
Tmr1MaxCnt      EQU     0FF5AH          ;address of Timer 1 Max Count Register
Tmr1Count       EQU     0FF58H          ;address of Timer 2 Count Register

Tmr1CtrlVal     EQU     1110000000000001B



Tmr2Ctrl        EQU     0FF66H          ;address of Timer 2 Control Register
Tmr2MaxCnt      EQU     0FF62H          ;address of Timer 2 Max Count Register
Tmr2Count       EQU     0FF60H          ;address of Timer 2 Count Register

; Control Register Values
Tmr2CtrlVal     EQU     1110000000000001B ;Timer 2 Control Register value
                       ;1---------------  enable timer
                       ;-1--------------  write to control
                       ;--1-------------  enable interrupts
                       ;----000000-0000-  reserved
                       ;---0------0-----  read only
                       ;---------------1  continuous mode

; Interrupt Vectors
Tmr1Vec         EQU     18              ;interrupt vector for Timer 1
Tmr2Vec         EQU     19              ;interrupt vector for Timer 2


; Interrupt Controller

; Addresses
INTCtrlrCtrl    EQU     0FF32H          ;address of Timer interrupt Control Register

INTCtrlrEOI     EQU     0FF22H          ;address of End_of_Interrupt Register

; Register Values
INTCtrlrCVal    EQU     00001H          ;The value to be given to TCUCON Register.
                                        ;000000000000----  These are reserved bits.
                                        ;------------0---  enables the Timer Interrupt
                                        ;-------------001  sets timer priority to 1.
TimerEOI        EQU     00008H          ;Command for Timer EOI

; General Timing Definitions


KHZ_4_CNT       EQU     576             ;Counts for generating 4KHz timer interrupts.
                                        ;18.432 / 2 / 4 /4000
ONE_MS_CNT      EQU     2304            ;Counts for generating 1KHz timer interrupts.
                                        ;18.432/ 2 / 4 / 1000
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                               ;
;                                  General.inc                                  ;
;                                   Homework3                                   ;
;                                 Sunghoon Choi                                 ;
;                                                                               ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Description:
;     This file contains definitions for general constants.
;
; Revision History:
;          10/21/2016    Sunghoon Choi    Initial Revision
;          10/22/2016    Sunghoon Choi    Added Comments
;          11/26/2016    Sunghoon Choi    Added DECIMAL_BASE

TRUE            EQU     1       ;LOGIC TRUE
FALSE           EQU     0       ;LOGIC FALSE
ZERO            EQU     0       ;INTEGER 0
MULT_BY_2       EQU     1       ;The constant used in SHL operation to multiply by 2
DECIMAL_BASE    EQU     10      ;Base of decimal numbers.

ASCII_NULL      EQU     0       ;ASCII Null
CARRIAGE_RETURN EQU     13      ;ASCII Carriage Return
STARTING_INDEX  EQU     0       ;Starting index for arrays
```

```
NAME    CONVERTS

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                            ;
;                                Converts.asm                                ;
;                                 Homework2                                  ;
;                                Sunghoon Choi                               ;
;                                                                            ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Description:      This program converts 16-bit signed or unsigned values to
;                   convert them into decimal or hexadecimal and store them as
;                   strings.
; Table of Contents:
;                   Dec2String - converts a 16-bit signed value to convert to
;                                decimal and store it as a string
;                   Hex2String - converts a 16-bit unsigned value to convert to
;                                hexadecimal and store it as a string
;
; Input:            None.
; Output:           None.
;
; User Interface:   None.
; Error Handling:   None.
;
; Algorithms:       Keep dividing the value by 10 or 16 and add ASCII offset to
;                   convert the value to digit string.
; Data Structures:  None.
;
; Revision History:
;        10/14/2016  17:00 Sunghoon Choi Started writing functional specification
;        10/14/2016  18:00 Sunghoon Choi initial compilation
;        10/14/2016  21:00 Sunghoon Choi Corrected stack pointer error
;        10/15/2016  04:00 Sunghoon Choi updated documentation



$INCLUDE(converts.INC)    ;include file for constants


CGROUP  GROUP   CODE

CODE SEGMENT PUBLIC 'CODE'

    ASSUME CS:CGROUP


; Dec2String
;
; Description:       This function is passed a 16-bit signed value to convert to
;                    decimal and store as a string. The string is 5 digits plus
;                    negative sign if the argument is negative. The string will
;                    have leading zeros if the number of digits are less than 5.
;                    The string contains the decimal representation of the argum
;                    ent in ASCII and terminates with <null>. The string is stor
;                    ed starting at the memory location indicated by the
;                    passed address.
;
; Operation:         The function checks if the argument(AX) is negative. If it
;                    is, store the minus sign '-' in DS:[SI] and negate the
;                    argument so that we can get the absolute value of it. If
;                    the argument is positive, skip the negating procedure and
;                    start the loop for extracting digits. Inside the loop, the
;                    function divides the argument by power-of-10, adds '0' to
;                    the quotient for ASCII conversion, and store it in DS:[SI].
;                    The remainder will be used for updating the argument(AX)
```

```
;                   later. Note that the power-of-10 is initialized
;                   to 10000(for 5 digit number) before the beginning of the
;                   loop. Now, after storing the converted digit string
;                   at DS:[SI], the function increments SI for the next loop,
;                   and updates the power-of-10 by dividing the power-of-10
;                   by 10. Now, update the argument(AX) to the remainder of
;                   'arg / pwr10' division, and go back to the beginning of the
;                   loop. The loop will end when the power-of-10 becomes zero.
;                   When all the digits are converted and stored, the function
;                   adds <NULL> to DS:[SI] and ends.
;
; Arguments:        AX - 16 bit signed value to convert to decimal string
;                   SI - The starting memory location where the converted
;                        string will be stored.
; Return Value:     None
;
; Local Variables:  AX - 16 bit signed value to convert to decimal string
;                   SI - The starting memory location where the converted
;                        string will be stored.
; Shared Variables: None
; Global Variables: None
;
; Input:            None
; Output:           None
;
; Error Handling:   None
;
; Algorithms:       Repeat dividing the argument by power of 10 and use the
;                   quotients to convert them to ASCII string and store them.
; Data Structures:  None.
;
; Registers Changed: AX, BX, CX, DX, SI, flags
;
; Limitations:      None
; Known bugs:       None
; Special Notes:    None


; Author:           Sunghoon Choi
; Last Modified:    10/15/2016
; Revision History:
;       10/14/2016  17:00 Sunghoon Choi Started writing functional specification
;       10/14/2016  18:00 Sunghoon Choi initial compilation
;       10/14/2016  21:00 Sunghoon Choi Corrected stack pointer error
;       10/15/2016  04:00 Sunghoon Choi updated documentation




Dec2String PROC NEAR
           PUBLIC Dec2String

InitDec2String:

CheckSign:                  ;Dec2String gets passed a 16-bit signed value.
                            ;Thus, we need to check sign first
CMP AX, 0                   ;Is the argument positive or equal to 0?

JGE Prepare10pow            ;Yes. Skip the negating procedure and start
                            ;extracting digits.
;JL HandlerNegArg           ;No. Store negative sign and get the absolute value.

HandleNegArg:
MOV BYTE PTR [SI], '-'      ;store negative sign first
INC SI                      ;increment the index to store other digits
```

```
133   NEG AX                    ;get the absolute value of the argument so that we
134                             ;can convert it.
135
136
137   Prepare10pow:             ;Now we start extracting and converting digits.
138   MOV CX, INITIAL_PWR10     ;Since the result string is 5 digits plus negative
139                             ;sign(if applicable),
140                             ;we initialize the dividing power-of-10 to 10000.
141
142   DecConvertStoreLoop:      ;start dividing the argument by power of 10 to
143                             ;extract digits.
144   MOV DX, 0                 ;clear DX for dividing procedure.
145   DIV CX                    ;divide the argument by the power of 10 to get a
146                             ;digit.
147                             ;AX<-quotient(digit),
148                             ;DX<-remainder(will be used to update argument)
149
150   ADD AX, ASCII_CODE_ZERO   ;add the ASCII offset to the quotient to convert
151                             ;it into ASCII.
152   MOV BYTE PTR [SI], AL     ;store the converted string at DS:[SI]
153
154   INC SI                    ;increment the index for next digit
155
156
157   PUSH DX                   ;save the remainder of previous division so that we
158                             ;can use it for updating the argument(AX) later
159   MOV DX, 0                 ;clear DX for dividing procedure. We will update the
160                             ;power of 10 here.
161   MOV AX, CX                ;move the power-of-10 to AX for division process.
162   MOV BX, 10                ;We should divide the power of 10 by 10 to update it
163   DIV BX                    ;divide the power-of-10 by 10 to update it.
164
165   POP DX                    ;retrieve the remainder of previous division to
166                             ;update the argument(AX)
167
168   CMP AX, 0                 ;Is the updated power-of-10 zero?
169
170   JE EndDec2String          ;if the updated power-of-10 is zero, it means we
171                             ;have completed converting the value.
172                             ;if the updated power-of-10 is still not zero,
173                             ;we keep the digit-converting loop.
174
175   MOV CX, AX                ;move the updated power-of-10 to CX so that we can
176                             ;continue digit-converting process.
177
178
179   MOV AX, DX                ;update the argument to the remainder of
180                             ;(argument/power-of-10) and
181                             ;repeat the digit converting process.
182
183
184   JMP DecConvertStoreLoop   ;continue repeating the digit-converting
185
                              ;process.

186
187
188   EndDec2String:            ;We finished converting and storing the strings.
189
190   MOV  BYTE PTR[SI], 0      ;Attach <NULL> to the end of the string.
191
192   RET                       ;return to where this function was called
193   Dec2String ENDP
194
195
196
```

```
197
198   ; Hex2String
199   ;
200   ; Description:       This function is passed a 16-bit unsigned value to convert
201   ;                     to hexadecimal and store as a string. The string is
202   ;                     4 digits(plus <NULL>) The string will have leading zeros if
203   ;                     the number of digits are less than 4. The string contains
204   ;                     the hexadecimal representation of the input value in ASCII
205   ;                     and terminates with <null>.
206   ;                     The string is stored starting at the memory location
207   ;                     indicated by the passed address.
208   ;
209   ; Operation:         The function starts a loop of extracting and converting the
210   ;                     digits. Inside the loop, it divides the argument by
211   ;                     power-of-16. If the quotient is bigger than or equal to 10,
212   ;                     it adds '0' to the quotient for ASCII conversion.
213   ;                     Otherwise, it adds 'A'-10.
214   ;                     Then, it stores the quotient in DS:[SI]. The remainder will
215   ;                     be used for updating the argument(AX) later. Note that the
216   ;                     power-of-16 is initialized to 4096(for 4 digit number)
217   ;                     before the beginning of the loop.
218   ;                     Now, after storing the converted digit string at DS:[SI],
219   ;                     the function increments SI for the next loop, and updates
220   ;                     the power-of-16 by dividing the power-of-16 by 16. Now,
221   ;                     update the argument(AX) to the
222   ;                     remainder of 'arg / pwr16' division, and go back to the
223   ;                     beginning of the loop. The loop will end when the
224   ;                     power-of-16 becomes zero.
225   ;                     When all the digits are converted and stored, the function
226   ;                     adds <NULL> to DS:[SI] and ends.
227   ;
228   ; Arguments:         AX - 16 bit unsigned value to convert to hexadecimal string
229   ;                     SI - The starting memory location where the converted
230   ;                          string will be stored.
231   ; Return Value:      None
232   ;
233   ; Local Variables:   AX - 16 bit unsigned value to convert to hexadecimal string
234   ;                     SI - The starting memory location where the converted
235   ;                          string will be stored.
236   ; Shared Variables:  None
237   ; Global Variables:  None
238   ;
239   ; Input:             None
240   ; Output:            None
241   ;
242   ; Error Handling:    None
243   ;
244   ; Algorithms:        Repeat dividing the argument by power of 16 and use the
245   ;                     quotients to convert them to ASCII string and store them.
246   ; Data Structures:   None.
247   ;
248   ; Registers Changed: AX, BX, CX, DX, SI, flags
249   ;
250   ; Limitations:       None
251   ; Known bugs:        None
252   ; Special Notes:     None
253
254
255   ; Author:            Sunghoon Choi
256   ; Last Modified:     10/15/2016
257   ; Revision History:
258   ;       10/14/2016  17:00 Sunghoon Choi Started writing functional specifications
259   ;       10/14/2016  18:00 Sunghoon Choi initial compilation
260   ;       10/14/2016  21:00 Sunghoon Choi Corrected stack pointer error
261   ;       10/15/2016  04:00 Sunghoon Choi updated documentation
262
```

```
263
264   Hex2String PROC NEAR
265              PUBLIC Hex2String
266
267
268   Prepare16pow:
269   MOV CX, INITIAL_PWR16       ;initialize the power-of-16 to 4096(16^3) since
270                               ;the argument will be converted to 4 digit string.
271
272   HexConvertStoreLoop:        ;start dividing the argument by power of 16 to
273                               ;extract digits
274
275   MOV DX, 0                   ;clear DX for dividing procedure.
276
277   DIV CX                      ;divide the argument by the power of 16 to get a
278                               ;digit.
279                               ;AX<-quotient(digit),
280                               ;DX<-remainder(will be used to update argument)
281
282   CheckDigitOver10:           ;check if the digit is larger than or equal to 10
283                               ;since 10 is 'A' and the required ASCII offset will
284                               ;become different.
285
286   CMP AX, 10                  ;Is the digit larger than or equal to 10?
287   JB SaveLessThan10Ascii      ;No. add '0' to it and store it.
288   ;JAE SaveBiggerThanOrEqualto10Ascii   ;Yes. add 'A'-10 to it and store it.
289
290
291   SaveBiggerThanOrEqualTo10Ascii:
292   ADD AX, ASCII_CODE_AMINUSTEN  ;add 'A'-10 to the digit to convert it into ASCII
293   MOV BYTE PTR[SI], AL          ;save the ascii string at DS:[SI]
294   INC SI                        ;increment the index for next digit
295   MOV AX, DX                    ;update the argument(AX) to the remainder of
296                                 ;previous division. So, now the number of digits
297                                 ;of the argument is decreased by 1
298
299   JMP UpdatePwr16               ;Now that we have converted a digit,
300                                 ;we should update the power-of-16
301
302
303   SaveLessThan10Ascii:
304   ADD AX, ASCII_CODE_ZERO       ;add the ASCII offset to the quotient to convert
305                                 ;it into ASCII.
306   MOV BYTE PTR[SI], AL          ;store the converted string at DS:[SI]
307   INC SI                        ;increment the index for next digit
308   MOV AX, DX                    ;update the argument(AX) to the remainder of
309                                 ;previous division.
310                                 ;So, now the number of digits decreased by 1.
311
312   UpdatePwr16:
313
314   PUSH DX                 ;Save the remainder of previous division so that we can
315                           ;use it for updating the argument(AX) later
316
317   MOV DX, 0               ;Clear DX for dividing procedure. We will update the
318                           ;power of 16 here.
319   MOV AX, CX              ;Move the power-of-16 to AX for division process.
320   MOV BX, 16              ;We should divide the power-of-16 by 16 to update it.
321   DIV BX                  ;Divide the power-of-16 by 16 to update it.
322
323   POP DX                  ;Retrieve the remainder of previous division to update
324                           ;the argument(AX)
325
326   CMP AX, 0               ;Is the updated power-of-16 zero?
327
328   JE EndHex2String        ;If the updated power-of is zero, it means we have
```

```
                        ;completed converting the value.
                        ;If the updated power-of-16 is still not zero,
                        ;we keep the digit-converting loop.

    MOV CX, AX          ;Move the updated power-of-16 to CX so that we can
                        ;continue digit-converting process.

    MOV AX, DX          ;Update the argument to the remainder of
                        ;(argument/power-of-16) and repeat the digit converting
                        ;process

    JMP HexConvertStoreLoop;Continue the digit-converting
    process.


    EndHex2String:      ;We finished converting and storing the strings.

    MOV  BYTE PTR[SI], 0    ;Attach <NULL> to the end of the string.


    RET                 ;return to where this function was called
    Hex2String ENDP




    UnsignedDec2String PROC NEAR
                       PUBLIC UnsignedDec2String

    UnsignPrepare10pow:         ;Now we start extracting and converting digits.
    MOV CX, INITIAL_PWR10       ;Since the result string is 5 digits plus negative
                                ;sign(if applicable),
                                ;we initialize the dividing power-of-10 to 10000.

    UnsignDecConvertStoreLoop:  ;start dividing the argument by power of 10 to
                                ;extract digits.
    MOV DX, 0                   ;clear DX for dividing procedure.
    DIV CX                      ;divide the argument by the power of 10 to get a
                                ;digit.
                                ;AX<-quotient(digit),
                                ;DX<-remainder(will be used to update argument)

    ADD AX, ASCII_CODE_ZERO     ;add the ASCII offset to the quotient to convert
                                ;it into ASCII.
    MOV BYTE PTR [SI], AL       ;store the converted string at DS:[SI]

    INC SI                      ;increment the index for next digit


    PUSH DX                     ;save the remainder of previous division so that we
                                ;can use it for updating the argument(AX) later
    MOV DX, 0                   ;clear DX for dividing procedure. We will update the
                                ;power of 10 here.
    MOV AX, CX                  ;move the power-of-10 to AX for division process.
    MOV BX, 10                  ;We should divide the power of 10 by 10 to update it
    DIV BX                      ;divide the power-of-10 by 10 to update it.

    POP DX                      ;retrieve the remainder of previous division to
                                ;update the argument(AX)

    CMP AX, 0                   ;Is the updated power-of-10 zero?

    JE EndUnsignedDec2String    ;if the updated power-of-10 is zero, it means we
                                ;have completed converting the value.
                                ;if the updated power-of-10 is still not zero,
                                ;we keep the digit-converting loop.
```

```
394
395   MOV CX, AX                    ;move the updated power-of-10 to CX so that we can
396                                 ;continue digit-converting process.
397
398
399   MOV AX, DX                    ;update the argument to the remainder of
400                                 ;(argument/power-of-10) and
401                                 ;repeat the digit converting process.
402
403
404   JMP UnsignDecConvertStoreLoop ;continue repeating the digit-converting
405
                                    ;process.

406
407
408   EndUnsignedDec2String:        ;We finished converting and storing the strings.
409
410   MOV  BYTE PTR[SI], 0          ;Attach <NULL> to the end of the string.
411
412   RET                           ;return to where this function was called
413   UnsignedDec2String ENDP
414
415   CODE ENDS
416
417
418   END
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                              ;
;                                Converts.inc                                  ;
;                                 Homework2                                    ;
;                                Sunghoon Choi                                 ;
;                                                                              ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;



; Constants

INITIAL_PWR10 EQU 10000         ;initial value for the dividing factor of
                                ;dec2string
ASCII_CODE_ZERO EQU '0'         ;ASCII offset required to convert 0 to '0'
INITIAL_PWR16 EQU 4096          ;initial value for the dividing factor of
                                ;hex2string
ASCII_CODE_AMINUSTEN EQU 55     ;ASCII offset required to convert 10 to 'A'.
                                ;This is equal to 'A'-10
```

```
NAME QUEUE

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                               ;
;                                  Queue.asm                                    ;
;                                  Homework3                                    ;
;                                Sunghoon Choi                                  ;
;                                                                               ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


; Description:
;       This file contains a number of functions to initialize, enqueue, and
;       dequeue the queue. Also, it contains functions for checking the status
;       of the queue.

; Table of Contents:
;    QueueInit(a,l,s) - Initialize the queue, HeadIndex, CurrentElemNum, ElemSize
;    QueueEmpty(a)    - Checks if the queue is empty. Sets the zero flag if the
;                       queue is empty and resets the zero flag otherwise.
;    QueueFull(a)     - Checks if the queue is full. Sets the zero flag if the queue
;                       is full. Resets the zero flag otherwise.
;    Enqueue(a,v)     - Waits until there's a slot in the queue and add a byte or
;                       a word
;    Dequeue(a)       - Waits until there's a value to take from the queue and
;                       return a byte or a word.
;
;
; Revision History:
;       10/19/2016    Sunghoon Choi  Started writing functional specifications
;       10/20/2016    Sunghoon Choi  Initial Compilation
;       10/20/2016    Sunghoon Choi  Corrected Stack Pointer error
;       10/21/2016    Sunghoon Choi  Corrected Head Index derivation error
;       10/21/2016    Sunghoon Choi  Updated documentations
;       10/31/2016    Sunghoon Choi  Revised the length initialization
;       11/31/2016    Sunghoon Choi  Changed to fixed length queue.


$INCLUDE(Queue.inc)      ;include file for queue structure and related constants
$INCLUDE(general.inc)    ;include file for general constants

CGROUP GROUP CODE


CODE SEGMENT PUBLIC 'CODE'

     ASSUME CS:CGROUP



; QueueInit
;
; Description:
;            Initializes the queue of element size(s) at the passed address(a).
;            After calling this procedure, the queue must
;            be empty and ready to accept the values. It initializes the
;            head index of the queue to zero. Note that the element
;            size(s) specifies whether each entry in queue is a byte or a word.
;            If s is TRUE(Non-zero), the entries are words.
;            If s is FALSE(EQU 0), the entries are bytes. The address(a) is passed
;            in by SI. It means that the queue starts at DS:SI. The element
;            size(s) is passed from BL. The length of the queue is fixed to
;            MAX_QUEUE_SIZE(EQU 2^8) bytes.
;
;
; Operation:  Initialize the HeadIndex to the starting location of the queue.
```

```
67  ;             Initialize the CurrentElemNum to zero since there's no elements
68  ;             when a queue is just created.
69  ;             Check if the arguemtn BL, the size of the queue's element, is
70  ;             equal to WORD_SIZE(EQU 2) or BYTE_SIZE(EQU 1). Set the elemNum
71  ;             to WORD_SIZE if BL is WORD_SIZE, and sets the elemNum to BYTE_SIZE
72  ;             if BL is BYTE_SIZE.
73  ;
74  ; Arguments:  SI - address of the queue
75  ;             BL - size of the queue's element.
76  ;
77  ; Return Value:    None
78  ;
79  ; Local Variables:   a  -  (SI)  -  address of the queue
80  ;                    s  -  (BL)  -  size of the queue's element
81  ;                        (Non-zero(TRUE) value means the elements are words)
82  ;                        (0(FALSE) value means the elements are bytes)
83  ;
84  ; Shared Variables:
85  ;    QueueModule                - (SI) - [Write] - The structure for the queue module
86  ;    QueueModule.HeadIndex      - (SI) - [Write] - Current Head Index of the queue
87  ;    QueueModule.CurrentElemNum - (SI) - [Write] - Current number of elemnts in the
    queue.
88  ;

89  ; Global Variables:  None
90  ;
91  ; Input:             None
92  ; Output:            None
93  ;
94  ; Error Handling:    None
95  ;
96  ; Algorithms:        None
97  ;
98  ; Data Structures:   Queue (MyQueue struct in Queue.inc)
99  ;
100 ; Registers Changed: Zero Flag
101 ;
102 ; Limitations:       The queue can only have a length of a power of 2.
103 ; Known bugs:        None
104 ; Special Notes:     None
105
106
107 ; Author:            Sunghoon Choi
108 ; Revision History:
109 ;       10/14/2016  17:00 Sunghoon Choi Started writing functional specifications
110 ;       10/14/2016  18:00 Sunghoon Choi initial compilation
111 ;       10/14/2016  21:00 Sunghoon Choi Corrected stack pointer error
112 ;       10/15/2016  04:00 Sunghoon Choi updated documentation
113 ;       10/31/2016  10:13 Sunghoon Choi revised the length initialization
114 ;       11/15/2016  21:00 Sunghoon Choi changed CMP BL, TRUE to CMP BL, FALSE
115
116
117 QueueInit       PROC    NEAR
118                 PUBLIC QueueInit
119
120
121 InitializeQueue:
122     MOV [SI].HeadIndex, QUEUE_INIT_CON
123                                     ;when queue is created, the Head Index and
124                                     ;tail index are on the very front.
125                                     ;tail index is not included in the
126                                     ;queueModule as an element since it can
127                                     ;be calculated by
128                                     ;(HeadIndex+Num of Elements) mod
                                        length
129
```

```
130
131         MOV [SI].CurrentElemNum, QUEUE_INIT_CON
132                                         ;When queue is created, there's no element
133                                         ;in the queue.
134
135         CMP BL, FALSE                   ;Check if the queue to be initialized is
136                                         ;a byte queue or a word queue
137
138         JNZ InitWordQueue               ;if the queue is a word queue, go set
139                                         ;the size of element to the size of a word
140                                         ;in bytes, which is WORD_SIZE(EQU 2).
141         ;JZ InitByteQueue               ;if the queue is a byte queue, go set
142                                         ;the size of element to the size of a byte
143                                         ;in bytes, which is BYTE_SIZE(EQU 1).
144
145     InitByteQueue:
146         MOV [SI].ElemSize, BYTE_SIZE    ;initialize the ElemSize to the size of
147                                         ;a byte in bytes, which is
148                                         ;BYTE_SIZE(EQU 1)
149
150         MOV AX, 256
151         MOV [SI].len, AX
152
153         JMP EndQueueInit
154
155     InitWordQueue:
156         MOV [SI].ElemSize, WORD_SIZE    ;Set the element's size to the size of
157                                         ;a word in bytes, which is
158                                         ;WORD_SIZE(EQU 2)
159
160         MOV AX, 256
161         MOV [SI].len, AX
162
163     EndQueueInit:                       ;finish the intialization.
164
165         RET
166
167     QueueInit ENDP
168
169
170     ; QueueEmpty
171     ;
172     ; Description:
173     ;        Checks if the queue is empty. It sets zero flag if the queue is empty.
174     ;        It resets zero flag if queue is not empty. The address (a) of the queue
175     ;        is passed by SI.
176     ;
177     ; Operation:  Check if QueueModule.CurrentElemNum is zero. If so, set ZF.
178     ;             Otherwise, reset ZF.
179     ;
180     ; Arguments:  SI - address of the queue
181     ;
182     ; Return Value: Zero Flag - set if the queue is empty
183     ;                         - reset if the queue is not empty.
184     ;
185     ; Local Variables: a  -  (SI) -  address of the queue
186     ;
187     ; Shared Variables:
188     ;    QueueModule                - (SI) - [Read] - The structure for the queue module
189     ;    QueueModule.CurrentElemNum - (SI) - [Read] - Current number of elemnts in the
190     ;                                                 queue.
191     ;
192     ; Global Variables:  None
193     ;
194     ; Input:             None
195     ; Output:            None
```

```
;
; Error Handling:    None
;
; Algorithms:        None
;
; Data Structures:   Queue    (MyQueue struct in Queue.inc)
;
; Registers Changed: Zero Flag
;
; Limitations:       The queue can only have a length of a power of 2.
; Known bugs:        None
; Special Notes:     None


; Author:            Sunghoon Choi
; Revision History:
;       10/14/2016   Sunghoon Choi Started writing functional specifications
;       10/14/2016   Sunghoon Choi initial compilation
;       10/14/2016   Sunghoon Choi Corrected stack pointer error
;       10/15/2016   Sunghoon Choi updated documentation


QueueEmpty      PROC    NEAR
                PUBLIC    QueueEmpty

CheckEmpty:
    CMP [SI].CurrentElemNum, QUEUE_INIT_CON
                                        ;check if the current number elements
                                        ;in the queue is zero.
                                        ;if so, set zero flag.
                                        ;Otherwise, reset zero flag.

EndQueueEmpty:

    RET

QueueEmpty ENDP


; QueueFull
;
; Description:
;       Checks if the queue is full. If the queue is full, set the zero flag.
;       Otherwise, reset the zero flag.
;
; Operation:
;       Compare the length of the queue and current number of elements.
;       If they are equal, it means the queue is full. So, set ZF.
;       If they are not equal, it means the queue is not full. So, reset ZF.
;
; Arguments:  SI - address of the queue
;
; Return Value:  Zero Flag - set if the queue is full
;                          - reset if the queue is not full.
;
; Local Variables:   a            -  (SI)  -  address of the queue
;                    currElemNumb -  (AX)   - Current Number of elements in the queue
;
; Shared Variables:
;    QueueModule               - (SI) - [Read] - The structure for the queue module
;
;   QueueModule.CurrentElemNum - (SI) - [Read] - Current number of elemnts in the
;                                                queue.
;   QueueModule.len            - (SI) - [Read] - The length of the queue
;
```

```
; Global Variables:  None
;
; Input:             None
; Output:            None
;
; Error Handling:    None
;
; Algorithms:        None
;
; Data Structures:   Queue    (MyQueue struct in Queue.inc)
;
; Registers Changed: AX, ZF
;
; Limitations:       The queue can only have a length of a power of 2.
; Known bugs:        None
; Special Notes:     None


; Author:           Sunghoon Choi
; Revision History:
;       10/14/2016  Sunghoon Choi Started writing functional specifications
;       10/14/2016  Sunghoon Choi initial compilation
;       10/14/2016  Sunghoon Choi Corrected stack pointer error
;       10/15/2016  Sunghoon Choi updated documentation


QueueFull   PROC    NEAR
            PUBLIC    QueueFull

CheckFull:
    MOV AX, [SI].CurrentElemNum     ;AX <- Current number of elements in the
                                    ;queue
    CMP AX, [SI].len                ;current number of elements in the
                                    ;queue is being equal to the queue's length
                                    ;means that the queue is full.
                                    ;If queue is full, set zero flag.
                                    ;otherwise, reset zero flag.
EndQueueFull:

        RET
QueueFull ENDP




; Dequeue
;
; Description:
;       The function removes a byte(80bit) or a word(16-bit) from the head of
;       queue and returns it in AL(if it's a byte) or AX(if it's a word). If the
;       queue is empty, it blocks until the queue receives a value to be removed
;       and returned. It will not return until a value is taken from the queue.
;       Note that the address of the queue is passed by SI.
;
; Operation:
;       It first checks if the queue is empty by calling QueueEmpty.
;       If the queue is empty, repeat calling QueueEmtpy.
;       If the queue is not empty, go check the size of the queue.
;       If the queue is a byte queue, go take a byte from the queue.
;       The function takes a byte value from the location of head index in the
;       queue.
;       If the queue is a word queue, it must first translate the head index
;       into the head index of word version. It multiplies the HeadIndex by
;       WORD_SIZE(EQU 2) and take a modulus of length for wrapping.
;       Now that we have the HeadIndex for word-version, we take the word value
```

```
;       from the location of the translated Head Index.
;       Once the function is done with taking a value from the queue, it
;       updates the HeadIndex. It adds BYTE_VALUE(EQU 1) to the HeadIndex
;       and take modulus of length for wrapping to get the new Head_ndex.
;       Update the QueueModule's HeadIndex to our new HeadINdex.
;       Finally, it decrements the current number of elements in the Queue by
;       the size of the queue's element. If the queue is a word queue, it
;       decrements the number of elements by WORD_SIZE(EQU 2). If the queue
;       is a byte queue, it decrements the number of elements by BYTE_SIZE.
;
;
; Arguments:      SI - address of the queue
;
; Return Value:   AL - a dequeued value (If byte)
;                 AX - a dequeued value (If word)
;
; Local Variables:  HeadIndex_WordVer - BX     - The HeadIndex which is translated
;                                               for a word queue
;                   UpdatedHeadIndex -  DX     - The new HeadIndex after dequeue
operation.
;                   a                -  SI     - The address of the queue
;                   DeqValue         -  AL(AX) - The dequeued value to be returned.
;
; Shared Variables:
;   QueueModule                - (SI) - [R/W]  - The structure for the queue module
;   QueueModule.ElemSize       - (SI) - [Read] - The size of the queue's entries
;   QueueModule.CurrentElemNum - (SI) - [W]    - Current number of elemnts in the
;                                               queue.
;   QueueModule.len            - (SI) - [Read] - The length of the queue
;   QueueModule.HeadIndex      - (SI) - [R/W]  - The head index of the queue.
;

; Global Variables:  None
;
; Input:             None
; Output:            None
;
; Error Handling:    If the queue is empty, stay in the loop and wait.
;
; Algorithms:        None
;
; Data Structures:   Queue (MyQueue struct in Queue.inc)
;
; Registers Changed: AX, BX, CX, DX, Zero Flag
;
; Limitations:       The queue can only have a length of a power of 2.
; Known bugs:        None
; Special Notes:     None


; Author:            Sunghoon Choi
; Revision History:
;       10/14/2016  Sunghoon Choi Started writing functional specifications
;       10/14/2016  Sunghoon Choi initial compilation
;       10/15/2016  Sunghoon Choi updated documentation



Dequeue     PROC     NEAR
            PUBLIC Dequeue

CheckQueueEmpty:            ;checks if the queue is empty
    CALL QueueEmpty         ;call QueueEmpty function to check if the queue
                            ;is empty
    JZ CheckQueueEmpty      ;if the queue is empty, wait until the queue
```

```
                            ;has a value to be taken. It's a blocking function
    ;JNZ DeqCheckElemSize   ;If the queue is not empty, go check the queue's
                            ;type

DeqCheckElemSize:
    MOV AL, BYTE_SIZE       ;Checks if the queue is a byte queue or a word queue
    CMP [SI].ElemSize, AL   ;Is the element size BYTE_SIZE(EQU 1)?
    JE GetByteValue         ;Yes. Let's go take a byte value
    ;JNE GetWordValue       ;No. Let's go take a word value

GetWordValue:
    XOR AX, AX              ;We are going to insert HeadIndex value in AL.
                            ;Thus, for division operation, we need to clear
                            ;the high bits (AH)
    XOR BX, BX              ;Clear BX to prevent unexpected error, just in case
    MOV AL, [SI].HeadIndex  ;Save HeadIndex value in AL for division operation
    SHL AX,  MULT_BY_2      ;We have to double the HeadIndex since
                            ;it's a word queue. The head index should move
                            ;with a step of WORD_SIZE(EQU 2), since it's a word
                            ;queue.
    MOV BX, [SI].len        ;Save the length of the queue in BX for division.
                            ;Since the length is in a word, we have to save it
                            ;in BX, not BL.
    XOR DX,DX               ;We have to clear DX because when the divisor size
                            ;is 2 byte, the dividend is DX:AX.
    DIV BX                  ;HeadIndex for Word = WORD_SIZE*HeadIndex mod length
                            ; "mod length" is needed since multiplying the
                            ;Head Index may lead to exceeding the maximum
                            ;array index.
                            ;The head index for word is stored in DX.
    MOV BX, DX              ;move the head index for word to BX to access array.
    MOV AX, WORD PTR [SI].Queue[BX]    ;Head's location = OFFSET of QueueModule+
                                       ;Offset of queue + new head index
                                       ;We take a word from head's location


    PUSH AX                            ;Save the dequeued word value since we have to
                                       ;use AX register for updating the head index.
    JMP UpdateHeadIndex


GetByteValue:
    XOR BX,BX                   ;Since we are going to store HeadIndex in BL, we have
                                ;to clear the high bits(BH) for future array accessing.
    MOV BL, [SI].HeadIndex      ;Store the Head Index in BL as an array pointer.

    MOV AL, BYTE PTR [SI].Queue[BX]    ;Head's location = OFFSET of QueueModule+
                                       ;Offset of queue + head index
                                       ;We take a byte from head's location
    PUSH AX                     ;save the dequeued value since we are going to
                                ;use AX for updating head index.

UpdateHeadIndex:
    XOR AX, AX                  ;Since we are going to store HeadIndex in AL,
                                ;we clear the high bits(AH).
    MOV AL, [SI].HeadIndex      ;AL is the current HeadIndex
    ADD AL, BYTE_SIZE           ;New HeadIndex = HeadIndex+BYTE_SIZE mod length
    XOR DX, DX                  ;Clear DX since the dividend for 2byte division
                                ;is DX:AX.
    MOV BX, [SI].len            ;BX = length of the queue.
    DIV BX                      ;DX = HeadIndex + BYTE_SIZE mod length
                                ;= new HeadIndex
    MOV [SI].HeadIndex, DL      ;Update the HeadIndex with the new HeadIndex
    POP AX                      ;retrieve the dequeued value
DecCurrentElemNum:
    XOR DX, DX                  ;clear the high bits since we are going to
```

```
                                ;store ElemSize in DL.
    MOV DL, [SI].ElemSize       ;The Current Element Number should be decreased
                                ;by WORD_SIZE if the queue is a word queue and
                                ;should be decreased by BYTE_SIZE if the queue
                                ;is a byte queue.
    SUB [SI].CurrentElemNum, DX ;Decrease the current element number
                                ;by BYTE_SIZE if the queue is a byte queue.
                                ;Decrease the current element number by
                                ;WORD_SIZE if the queue is a word queue.

EndDequeue:

    RET

Dequeue ENDP



; Enqueue
;
; Description:
;       The function adds the passed byte or word to the tail of the queue.
;       If the queue is full, it waits until the queue has a slot to add the
;       value. It does not return until it adds the value to the queue. Note
;       that the address of the queue is passed by SI while the value to add
;       is passed by AL(if the queue's entries are bytes) or AX(if the queue's
;       entries are words.
;
; Operation:
;       The function checks if the queue is full. If the queue is full,
;       stay in the loop and repeatedly call QueueFull until there is a slot
;       to enqueue.
;       If the queue is not full, it checks the size of the value to be added.
;       If it's a byte, go calculate the tail index of the queue.
;       The tail index is calculated by TailIndex=HeadIndex+CurrentElemNum mod
;       length for wrapping. Then, it adds the byte value to the tail of the
;       queue. If the arguemnt is a word, go calculate the tail index of the
;       queue with the headIndex for word version. It multiplies the
;       headIndex by WORD_SIZE(EQU 2) and takes a modulus of length for wrapping
;       to get the tail index of the word queue. Then, it adds the word value
;       to the tail of the queue.
;       Once adding the values to the queue is done, it increments the current
;       number of elements in the queue by the size of the entries.
;       It increments the number of elements by BYTE_SIZE if the argument was
;       a byte, and increments the number of elements by WORD_SIZE if the
;       argument was a word.
;
;
; Arguments:    SI - address of the queue
;               AX - the value to add(if word)
;               AL - the value to add(If byte)
;
; Return Value: None
;
; Local Variables:
;           HeadIndex                - AL    - The head index of the queue
;           TailIndexWithoutWrapping - AX    - Calculated Tail Index
;                                              wihtout wrapping
;           TailIndex                - DX,BX - Calculated Tail Index with wrapping
;           a                        - SI    - address of the queue
;           EnqueueVal               - AX,AL - The value to be added to the queue
;
; Shared Variables:
;   QueueModule                 - (SI) - [R/W] - The structure for the queue module
;   QueueModule.ElemSize        - (SI) - [Read]- The size of the queue's entries
```

```
;   QueueModule.CurrentElemNum  - (SI) - [W]   - Current number of elemnts in the
;                                                queue.
;   QueueModule.len             - (SI) - [Read]- The length of the queue
;   QueueModule.HeadIndex       - (SI) - [R/W] - The head index of the queue.
;
;
; Global Variables:  None
;
; Input:             None
; Output:            None
;
; Error Handling:   If the queue is full, stay in the loop and wait.
;
; Algorithms:        None
;
; Data Structures:   Queue (MyQueue struct in Queue.inc)
;
; Registers Changed: AX, BX, CX, DX, Zero Flag
;
; Limitations:      The queue can only have a length of a power of 2.
; Known bugs:       None
; Special Notes:    None


; Author:            Sunghoon Choi
; Revision History:
;        10/14/2016  Sunghoon Choi Started writing functional specifications
;        10/14/2016  Sunghoon Choi initial compilation
;        10/15/2016  Sunghoon Choi updated documentation




Enqueue     PROC NEAR
            PUBLIC Enqueue

    PUSH AX         ;Save AX so that we can pop it later
                    ;and add to the queue later.

CheckQueueFull:
    CALL QueueFull          ;Checks if the queue is full.
    JZ CheckQueueFull       ;if the queue is full, stay in the loop and wait.
                            ;this is a blocking function
    ;JNZ EnqCheckElemSize   ;if the queue is not full, go check the element size.

EnqCheckElemSize:
    MOV AL, BYTE_SIZE       ;Checks if the queue is a byte queue or a word queue
    CMP [SI].ElemSize, AL   ;Is the element size BYTE_SIZE(EQU 1)?
    JNE GetWORDTailIndex    ;No, go find the tail index of the word queue.
    ;JE GetByteTailIndex    ;Yes, go find the tail index of the byte queue.


GetByteTailIndex:
     XOR AX,AX              ;clear the high bits of AX since we are going to
                            ;store the HeadIndex in AL.
    MOV AL, [SI].HeadIndex  ;Get the head index of the queue to find the tail
                            ;index.
    ADD AX, [SI].CurrentElemNum ;HeadIndex+CurrentElemNum is the tail index
                                ;without wrapping
    XOR DX,DX               ; We have to clear DX since when the divisor is 2bytes,
                            ;the dividend is DX:AX.
    DIV [SI].len            ;DX = TailIndex = (HeadIndex + CurrentElem) mod len

AddByteValue:
    MOV BX, DX             ;Move the tailIndex to BX for array accessing.
    POP AX                 ;retrieve the value to be enqueued
```

```
591         MOV BYTE PTR [SI].Queue[BX], AL ;Add the argument byte to the tail of the
592                                         ;queue.
593         JMP IncCurrentElemNumByte       ;Now that we have added the value,
594                                         ;go increment the current number of elments.
595
596     GetWORDTailIndex:
597         XOR AX,AX                   ;clear the high bits of AX since we are going to
598                                     ;store the HeadIndex in AL.
599         MOV AL, [SI].HeadIndex      ;Get the head index of the queue to find the tail
600                                     ;index.
601         SHL AX, MULT_BY_2           ;We have to double the headIndex
602                                     ;to translate it into the head index for word version
603                                     ;In word queue, head index should proceed by
604                                     ;a step of WORD_SIZE(EQU 2)
605         ADD AX, [SI].CurrentElemNum ;TailIndex = headIndexWordVer + CurrentElemNum
606         XOR DX,DX                   ;We have to clear DX since when the divisor is 2bytes,
607                                     ;the dividend is DX:AX.
608         DIV [SI].len                ;DX = TailIndex = (HeadIndex*2+CurrentElemNum) mod
            len
609
610
611     AddWordValue:
612
613         MOV BX, DX                          ;Move the tailIndex to BX for array accessing.
614         POP AX                              ;retrieve the value to be enqueued
615         MOV WORD PTR [SI].Queue[BX], AX     ;Add the argument word to the tail of the
            queue.
616
617         JMP IncCurrentElemNumWord           ;Now that we have added the value,
618                                             ;go increment the current number of elments.
619
620
621     IncCurrentElemNumByte:
622         ADD [SI].CurrentElemNum, BYTE_SIZE  ;since we added a byte,
623                                             ;current number of elements must be
624                                             ;increased by BYTE_SIZE(EQU 1)
625         JMP EndEnqueue
626
627     IncCurrentElemNumWord:
628         ADD    [SI].CurrentElemNum, WORD_SIZE  ;since we added a word,
629                                                ;current number of elements must be
630                                                ;increased by WORD_SIZE(EQU 2)
631
632
633     EndEnqueue:
634
635         RET
636
637     Enqueue ENDP
638
639     CODE ENDS
640
641     END
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                               ;
;                                   Queue.inc                                   ;
;                                   Homework3                                   ;
;                                  Sunghoon Choi                                ;
;                                                                               ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Description:
;    This file contains definitions of constants and structures for (queue.asm)
;
;
; Revision History:
;   2016/10/21    Sunghoon Choi    Initial Revision
;   2016/10/22    Sunghoon Choi    Changed MAX_QUEUE_SIZE value to 256 from 1024


BYTE_SIZE         EQU    1            ;The size of a byte in bytes.
WORD_SIZE         EQU    2            ;The size of a word in bytes.
MAX_QUEUE_SIZE    EQU    256          ;The maximum size, or length for the queue.
QUEUE_INIT_CON    EQU 0               ;Initial condition for queue elements

QueueModule STRUC
    Queue        DB    MAX_QUEUE_SIZE DUP (?) ;Even number slots required since words
    can be inserted.
    HeadIndex    DB   ?
    len          DW   ?                     ;Final Index is 255. The Index starts
    from zero
    ElemSize     DB   ?
    CurrentElemNum DW ?
QueueModule ENDS
```

```
NAME Display
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                            ;
;                                   Display                                  ;
;                                  Homework 4                                ;
;                                 Sunghoon Choi                              ;
;                                                                            ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Description:
;    This file contains the functions necessary for displaying number or strings
;    on 7-Segment LED digits.
;
; Table of Contents:
;    InitDisplay         - Initializes the necessary buffers and constants for
;                          Display routine.
;    Display             - Converts the passed strings to appropriate pattern values
;                          and save them in a buffer.
;    DisplayNum          - Receive a decimal value, convert it to a string, and convert
;                          the string to its 7-Segment pattern and save it in a buffer.
;    DisplayHex          - Receive a hexadecimal value, convert it to a string,
;                          and convert the string to its 7-Segment pattern and
;                          save it in a buffer.
;    DisplayEventHandler - This eventhandler takes a pattern value from
;                          PatternBuffer and output it to the current LED digit.
;                          It is called by Timer2 at every 1ms.
;
;
; Revision History:
;    10/25/2016   Sunghoon Choi      Created
;    10/25/2016   Sunghoon Choi      Fixed PatternBuffer's index error
;    10/25/2016   Sunghoon Choi      Fixed the error caused by not pushing
;                                    the SI value before calling Dec2String
;                                    and Hex2String.
;    10/28/2016   Sunghoon Choi      Updated functional specifications
;    10/29/2016   Sunghoon Choi      Revised comments.



$INCLUDE (Display.inc)    ;include the .inc file which contains constants for
                          ;Display.asm


EXTRN   Dec2String:NEAR         ;import Dec2String for string conversion
EXTRN   Hex2String:NEAR         ;import Hex2String for string conversion
EXTRN   ASCIISegTable:BYTE      ;reference to the ASCIISegTable
                                ;allows us to convert a character
                                ;to its proper 7-Segment LED pattern.


CGROUP GROUP CODE
DGROUP GROUP DATA

CODE SEGMENT PUBLIC 'CODE'

        ASSUME  CS:CGROUP, DS:DGROUP



; InitDisplay
;
; Description:
;       The function fills the StringBuffer with <NULL>s and fills the
;       PatternBuffer with BLANK_PATTERNs. Also, it initializes the
;       Current_Digit which is an index that indicates the next digit to be
;       displayed to the first digit(which has the index of 0)
```

```
; Operation:
;       Since lengths of StringBuffer and PatternBuffer are both NUM_DIGITS(EQU
;       8), it first sets the iteration counter of initialization loop to
;       NUM_DIGITS(EQU 8). Then, it starts inserting <NULL> to StringBuffer and
;       BLANK_PATTERN to PatternBuffer with increasing the buffer index per
;       each loop. Finally, it initializes the CurrentDigit to the index of
;       first LED digit, which is zero.
; Arguments:
;       None
; Return Value:
;       None
; Local Variables:
;       LoopIndex (CX)      -  A counter for initializing loop
;       BufferIndex (SI)    -  An index used for initializing StringBuffer
;                              and PatternBuffer's elements.
; Shared Variables:
;       StringBuffer   - [Write]  - A buffer that receives the strings converted
;                                   by Dec2String and Hex2String. The maximum
;                                   length of string it can contain at each time
;                                   is NUM_DIGITS(EQU 8)
;       PatternBuffer  - [Write]  - A buffer that contains the pattern values
;                                   for the ASCII characters to be displayed on
;                                   LED.
;       CurrentDigit   - [Write]  - The index which indicates the next digit
;                                   to be displayed.
; Global Variables:
;       None
; Input:
;       None
; Output:
;       None
; Error Handling:
;       None
; Algorithms:
;       None
; Data Structures:
;       Buffers
; Registers Changed:
;       CX, SI
; Limitations:
;       None
; Known bugs:
;       None
; Special Notes:
;       None
; Author:
;       Sunghoon Choi
; Revision History:
;       10/25/2016  -   Sunghoon Choi   Started writing functional specification
;       10/25/2016  -   Sunghoon Choi   Initial Compilation
;       10/28/2016  -   Sunghoon Choi   Updated documentation

InitDisplay PROC    NEAR
            PUBLIC  InitDisplay

    MOV CX, NUM_DIGITS          ;We have to initialize the buffers by
                                ;initializing each buffer element per loop.
                                ;So, set the loop counter to NUM_DIGITS(EQU 8)
                                ;since the lengths of the buffers are NUM_DIGITS
    XOR SI,SI                   ;Initialize the buffers from their first element
                                ;So, we have to set the buffer index to zero.
FillNullChar:
    MOV StringBuffer[SI], 0     ;Initialize the StringBuffer with NULLs.
FillBlankPattern:
    MOV PatternBuffer[SI], BLANK_PATTERN
                                ;Fill the PatternBuffer with BLANK_PATTERNs.
```

```
                                ;When BLANK_PATTERNs are output to LEDs,
                                ;corresponding LED will stay turned off.
    INC SI                      ;increment the buffer index to continue
                                ;initialize next element.
    LOOP FillNullChar           ;We continue initializing the buffers until
                                ;the buffers are completely filled.
SetToFirstDigit:
    MOV CurrentDigit, 0         ;We are going to turn on the LED digits from the
                                ;leftmost one. Thus, we initialize the
                                ;CurrentDigit to the index of first, leftmost
                                ;digit.
EndInitDisplay:
    RET                         ;End initialization process
InitDisplay ENDP




; Display
;
; Description:
;       The function is passed a string of NUM_DIGITS(EQU 8)length including
;       NULL. If the passed string is shorter than NUM_DIGITS, pad BLANK_PATTERN
;       for the remaining LEDs.
; Operation:
;       It takes a character from ES:[SI], the location of StringBuffer, to
;       check if the received character is NULL. If the character is NULL, fill
;       BLANK_PATTERNs to the remaining slots of PatternBuffer. If the character
;       is not NULL, it converts the character to the corresponding pattern
;       by using the ASCIISegTable. Note that ES is used for string transfer
;       to leave the string in code segment without changing DS.
; Arguments:
;       str - ES:[SI] - The string to be converted to a pattern(and to be
;                       displayed in the end)
; Return Value:
;       PatternBuffer will get values returned from Display function.
; Local Variables:
;      string          - ES:[SI] - The string to be converted to a pattern(and to be
;                                  displayed in the end
;      PatternBufIndex - DI      - The index of PatternBuffer.
;                                  Used for storing the converted patterns in
;                                  the Pattern Buffer.
;      StringBufIndex  - SI      - The index of StringBuffer.
;                                  Used for taking the strings from StringBuffer.
; Shared Variables:
;       StringBuffer  - [Read]  - A buffer that receives the strings converted
;                                 by Dec2String and Hex2String. The maximum
;                                 length of string it can contain at each time
;                                 is NUM_DIGITS(EQU 8)
;       PatternBuffer  - [Write] - A buffer that contains the pattern values
;                                  for the ASCII characters to be displayed on
;
LED.

; Global Variables:
;       None
; Input:
;       None
; Output:
;       None
; Error Handling:
;       None
; Algorithms:
```

```
;       None
; Data Structures:
;       Buffers
; Registers Changed:
;       AX, BX, DI, SI
; Limitations:
;       The function can fill the buffer just once per call.
;       Thus, it cannot convert and fill more than NUM_DIGITS(EQU 8) characters
;       per each call.
; Known bugs:
;       None
; Special Notes:
;       None
; Author:
;       Sunghoon Choi
; Revision History:
;       10/25/2016  -   Sunghoon Choi   Started writing functional specification
;       10/25/2016  -   Sunghoon Choi   Initial Compilation
;       10/25/2016  -   Sunghoon Choi   Fixed Index error of PatternBuffer
;       10/28/2016  -   Sunghoon Choi   Updated documentation


Display PROC    NEAR
        PUBLIC  Display

    XOR DI,DI                           ;We are going to store the patterns in
                                        ;PatternBuffer from the beginning(index 0).
                                        ;Thus, set the index to zero.
CheckNull:
    XOR AX,AX
    MOV AL, BYTE PTR ES:[SI]            ;Bring a character from StringBuffer
    CMP AL, 0                           ;Is the character null?
    JE FillBlanks                       ;yes, fill the PatternBuffer with
                                        ;BLANK_PATTERNs.
    ;JNE ConvertAndStorePattern         ;No, convert and store patterns.
ConvertAndStorePattern:
    MOV BX, OFFSET(ASCIISegTable)       ;We have to refer to ASCIISegTable
                                        ;to obtain the pattern converted from
                                        ;the string.

    XLAT CS:ASCIISegTable               ;Converted pattern will be stored in
                                        ;AL. Since the elements of
                                        ;ASCIISegTable are in the order of
                                        ;ASCII strings, we can use the
                                        ;string(from StringBuffer) as an index
                                        ;for looking up the table.

    MOV PatternBuffer[DI], AL           ;Stores the converted pattern in
                                        ;PatternBuffer[PatternBufIndex].

    INC DI                              ;We have to store the next pattern
                                        ;which will be used for the next digit.
    INC SI                              ;We should obtain the next string from
                                        ;StringBuffer which will be displayed
                                        ;on the next LED digit.

    CMP DI, NUM_DIGITS                  ;Was that the pattern for
                                        ;the last digit?
    JGE EndDisplay                      ;Yes, finish filling PatternBuffer.
    JL CheckNull                        ;No, go convert the next character.

FillBlanks:
    MOV PatternBuffer[DI], BLANK_PATTERN    ;Since the character was NULL,
                                            ;display BLANK_PATTERN on the digit.
                                            ;BLANK_PATTERN will display
                                            ;nothing on the LED.
```

```
263
264         INC DI                              ;Go display BLANK_PATTERN on the next
265                                             ;digit too.
266         CMP DI, NUM_DIGITS                  ;Was that the last digit?
267         JL  FillBlanks                      ;No, keep padding BLANK_PATTERN for
268                                             ;the remaining LED digits.
269         ;JGE EndDisplay                     ;Yes, the LED digits are completely
270                                             ;padded with blank patterns.
271                                             ;So, finish the process.
272     EndDisplay:
273         RET                                 ;End of Display Process.
274     Display ENDP
275
276
277
278
279     ; DisplayNum
280     ;
281     ; Description:
282     ;       The function is passed a 16-bit signed value to output in decimal
283     ;       of 5 digits plus sign. If the number has less than 5 digits,
284     ;       digit 0 will be padded for the remaining spots.
285     ; Operation:
286     ;        First, it makes DS and ES point to the same segment.
287     ;       Then, it obtains the address of StringBuffer where the string converted
288     ;       by Dec2String will be stored. Next, it calls Dec2String to store the
289     ;       converted string in StringBuffer and calls Display function to convert
290     ;       the string into patterns.
291     ; Arguments:
292     ;       n   -   AX  -   The 16-bit signed value to be output in decimal string.
293     ; Return Value:
294     ;       none
295     ; Local Variables:
296     ;       StringBufAddr   -   SI  -   The address of StringBuffer where
297     ;                                   the converted strings will be saved.
298     ; Shared Variables:
299     ;        StringBuffer   - [Write] - A buffer that receives the strings converted
300     ;                                   by Dec2String and Hex2String. The maximum
301     ;                                   length of string it can contain at each time
302     ;                                   is NUM_DIGITS(EQU
        8)
303     ; Global Variables:
304     ;        None
305     ; Input:
306     ;        None
307     ; Output:
308     ;        None
309     ; Error Handling:
310     ;        None
311     ; Algorithms:
312     ;        None
313     ; Data Structures:
314     ;        Buffers
315     ; Registers Changed:
316     ;        BX, ,DS, ES, SI
317     ; Limitations:
318     ;        None
319     ; Known bugs:
320     ;        None
321     ; Special Notes:
322     ;        None
323     ; Author:
324     ;        Sunghoon Choi
325     ; Revision History:
326     ;       10/25/2016  -   Sunghoon Choi   Started writing functional specification
327     ;       10/25/2016  -   Sunghoon Choi   Initial Compilation
```

```
;       10/25/2016  -   Sunghoon Choi   Fixed the SI related error by
;                                       pushing SI before calling Dec2String
;                                       and popping SI before calling Display.
;       10/28/2016  -   Sunghoon Choi   Updated documentation

DisplayNum  PROC    NEAR
            PUBLIC  DisplayNum

AdjustSegmentNum:
    MOV BX, DS                      ;We should make DS and ES point to the
    MOV ES, BX                      ;same segment since Dec2String returns
                                    ;the converted string to DS:SI while
                                    ;Display gets the string from ES:SI.

    MOV SI, OFFSET(StringBuffer)    ;set the destination for converted strings
                                    ;The string converted by Dec2String will
                                    ;be stored in StringBuffer.

ConvertAndStoreNum:
    PUSH SI                         ;We need to save the address of StringBuffer
                                    ;since it will be used by Display in the
                                    ;future.
                                    ;Dec2String alters the value of SI.
    CALL Dec2String                 ;The converted decimal string will be stored
                                    ;in StringBuffer.
    POP SI                          ;Return the address of StringBuffer since
                                    ;Dec2String has changed the value of SI.
    CALL Display                    ;Now that we have string values in
                                    ;StringBuffer, go convert the string to
                                    ;proper patterns.
EndDisplayNum:
    RET                             ;Finsih the process and return to where
                                    ;this process was called.
DisplayNum ENDP




; DisplayHex
;
; Description:
;       The function is passed a 16-bit unsigned value to output in hexadecimal
;       of 4 digits. If the number has less than 4 digits,
;       digit 0 will be padded for the remaining spots.
; Operation:
;       First, it makes DS and ES point to the same segment.
;       Then, it obtains the address of StringBuffer where the string converted
;       by Hex2String will be stored. Next, it calls Hex2String to store the
;       converted string in StringBuffer and calls Display function to convert
;       the string into patterns.
; Arguments:
;       n   -   AX  -   The 16-bit unsigned value to be output
;                       in hexadecimal string.
; Return Value:
;        none
; Local Variables:
;         StringBufAddr -   SI    -   The address of StringBuffer where
;                                       the converted strings will be saved.
;
; Shared Variables:
;       StringBuffer   - [Write]  -   A buffer that receives the strings converted
;                                     by Dec2String and Hex2String. The maximum
;                                     length of string it can contain at each time
```

```
;                                   is NUM_DIGITS(EQU
8)

; Global Variables:
;       none
; Input:
;       none
; Output:
;       none
; Error Handling:
;       none
; Algorithms:
;       none
; Data Structures:
;       Buffers
; Registers Changed:
;       BX, DS, ES, SI
; Limitations:
;       none
; Known bugs:
;       none
; Special Notes:
;       none
; Author:
;       Sunghoon Choi
; Revision History:
;       10/25/2016  -    Sunghoon Choi  Started writing functional specification
;       10/25/2016  -   Sunghoon Choi   Initial Compilation
;       10/25/2016  -   Sunghoon Choi   Fixed the SI related error by
;                                       pushing SI before calling Hex2String
;                                       and popping SI before calling Display.
;       10/28/2016  -   Sunghoon Choi   Updated documentation

DisplayHex  PROC    NEAR
            PUBLIC  DisplayHex

AdjustSegmentHex:
    MOV BX, DS                      ;We should make DS and ES point to the
    MOV ES, BX                      ;same segment since Hex2String returns
                                    ;the converted string to DS:SI while
                                    ;Display gets the string from ES:SI.

    MOV SI, OFFSET(StringBuffer)    ;set the destination for converted strings
                                    ;The string converted by Hex2String will
                                    ;be stored in StringBuffer.

ConvertAndStoreHex:
    PUSH SI                         ;We need to save the address of StringBuffer
                                    ;since it will be used by Display
                                    ;in the future.
                                    ;Hex2String alters the value of SI.
    CALL Hex2String                 ;The converted hexadecimal string will be
                                    ;stored in StringBuffer.
    POP SI                          ;Return the address of StringBuffer since
                                    ;Hex2String has changed the value of SI.
    CALL Display                    ;Now that we have string values in
                                    ;StringBuffer, go convert the string to
                                    ;proper patterns.

EndDisplayHex:
    RET                             ;Finsih the process and return to where
                                    ;this process was called.
DisplayHex ENDP
```

```
458
459
460
461   ; DisplayEventHandler
462   ;
463   ; Description:
464   ;       It is an EventHandler for displaying on LED digits.
465   ;       It is called by Timer interrupts to take patterns from PatternBuffer and
466   ;       output them to the actual LED(hardware).
467   ;       It goes through each digit from the leftmost digit to rightmost digit
468   ;       by being called once per 1ms so that human cannot notice that the LEDs
469   ;       are actually blinking.
470   ; Operation:
471   ;       It obtains each patterns digit by digit from PatternBuffer and output
472   ;       the patterns to LEDs. After outputting to LEDs, it increments the
473   ;       CurrentDigit with wrapping to display on the next LED digit.
474   ; Arguments:
475   ;       none
476   ; Return Value:
477   ;       none
478   ; Local Variables:
479   ;       CurrentLEDAddr    -    DX    -    The address of the current LED digit to
480   ;                                         be displayed.
481   ;       PatternVal        -   AL     -    The pattern to be displayed on the current
482   ;                                         LED digit.
483   ; Shared Variables:
484   ;      PatternBuffer  - [Write]      -    A buffer that contains the pattern values
485   ;                                         for the ASCII characters to be displayed on
486   ;                                         LED.
487   ;      CurrentDigit   - [Read]       -    The index which indicates the next digit
488   ;                                         to be
      displayed.

489   ; Global Variables:
490   ;       none
491   ; Input:
492   ;       none
493   ; Output:
494   ;       7-Segment LED digits.(00H~07H)
495   ; Error Handling:
496   ;       none
497   ; Algorithms:
498   ;       none
499   ; Data Structures:
500   ;       Buffers
501   ; Registers Changed:
502   ;       AX, BX, CX, DX
503   ; Limitations:
504   ;       The eventhandler can display only one LED digit per call.
505   ; Known bugs:
506   ;       none
507   ; Special Notes:
508   ;       none
509   ; Author:
510   ;       Sunghoon Choi
511   ; Revision History:
512   ;      10/25/2016  -   Sunghoon Choi   Started writing functional specification
513   ;      10/25/2016  -   Sunghoon Choi   Initial Compilation
514   ;      10/25/2016  -   Sunghoon Choi   Fixed index calculation.
515   ;      10/28/2016  -   Sunghoon Choi   Updated documentation
516
517
518   DisplayEventHandler PROC    NEAR
519                       PUBLIC  DisplayEventHandler
520
521       XOR AX, AX
```

```
    XOR BX, BX          ;We need to clear the high bits of BX since
                        ;we are going to use BX as an index for PatternBuffer
                        ;while its maximum value is NUM_DIGITS(EQU 8), a byte
                        ;value obtained from CurrentDigit.
    XOR DX, DX          ;The LED digit's address starts at 00H.
    MOV DL, LED_ADDR    ;So clear DH and set the address of output port
                        ;to LED_ADDR(EQU 00H)


HandlerFillPattern:
    MOV BL, CurrentDigit        ;We are going to output to the LED digits
                                ;one by one, by moving from left to right.
                                ;Thus, load the current digit's index.
    ADD DL, BL                  ;Current digit's address equals
                                ;First LED's address + digit count
    MOV AL, PatternBuffer[BX]   ;Bring the pattern for current digit.
    OUT DX, AL                  ;Output the pattern to the current LED digit.

HandlerUpdateDigit:
    MOV AL, CurrentDigit        ;Since we have outputted to the current
                                ;LED, we should proceed to output to the
                                ;next LED digit.
    INC AL                      ;Move to the next digit.
    MOV CL, NUM_DIGITS          ;We need to wrap when incrementing
                                ;the digit index.
    DIV CL                      ;CurrentDigit = (CurrentDigit + 1) mod NUM_DIGITS
    MOV CurrentDigit, AH        ;Update the CurrentDigit.
EndDisplayEventHandler:
    RET                         ;End of DisplayEventHandler.

DisplayEventHandler ENDP


CODE ENDS

DATA    SEGMENT PUBLIC  'DATA'
      StringBuffer  DB  NUM_DIGITS  DUP (?) ;The buffer which contains
                                           ;string(a list of characters)
                                           ;to be converted to patterns.
      PatternBuffer DB  NUM_DIGITS  DUP (?) ;The buffer which contains
                                           ;patterns to be outputted to
                                           ;7-Segment LED digits.
      CurrentDigit  DB  ?                  ;The digit to be outputted next.
DATA    ENDS


END
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                               ;
;                                Display.inc                                    ;
;                                 Homework 4                                    ;
;                                Sunghoon Choi                                  ;
;                                                                               ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Description:
;  This file contains the definitions for the Display functions for RoboTrike.
;
; Revision History:
;    10/25/2016    Sunghoon Choi    Created
;     10/28/2016   Sunghoon Choi    Updated documentation(comments) for constants.

NUM_DIGITS      EQU     8         ;The number of LED digits on the board.
BLANK_PATTERN   EQU     00000000B ;The pattern for empty digit.
LED_ADDR        EQU     00H       ;The starting address of the 7-Segment LEDs.
                                  ;(The leftmost LED digit)
DISPLAY_START_INDEX EQU 0         ;The starting index of display.
```

```
NAME Keypad

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                              ;
;                                 Keypad.asm                                   ;
;                                 Homework5                                    ;
;                               Sunghoon Choi                                  ;
;                                                                              ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Description:
;     This file contains the functions necessary for handling keypad inputs.
;
; Table of Contents:
;    InitKeypad          - Initializes the necessary buffers and constants for
;                          Keypad routine.
;    KeypadEventHandler  - Reads keys from the keypad and enqueues the key event
;                          to EventBuf.
;                          It is called by Timer2 at every 1ms.
;
;
; Revision History:
;    10/30/2016     Sunghoon Choi         Created
;    10/31/2016     Sunghoon Choi         Initial Compilation
;    11/3/2016      Sunghoon Choi         Revised Comments
;    11/3/2016      Sunghoon Choi         Revised the DS value from DATA to DGROUP



$INCLUDE(Keypad.inc)        ;Include the .inc file which contains constants for keypad.asm
$INCLUDE(Events.inc)        ;Include the .inc file which contains the list of events for
                            ;Robotrike system.


EXTRN EnqueueEvent:NEAR     ;import EnqueueEvent for enqueueing the keypad events
                            ;to EventBuf

CGROUP GROUP CODE
DGROUP GROUP DATA
CODE SEGMENT PUBLIC 'CODE'

        ASSUME  CS:CGROUP, DS:DGROUP



; InitKeypad
;
; Description:
;        Initializes the KeyDebounceCounter array with KEY_COUNTER_MAX
; Operation:
;        It inserts KEY_COUNTER_MAX to each element of KeyDebounceCounter array by
;        incrementing the array index every loop. It repeats the loop until
;        the index reaches NUM_KEYS.
; Arguments:
;      None
; Return Value:
;       None
; Local Variables:
;      KeyArrayIndex(DI)   -  The array index used for initializing KeyDebounceCounter array.
; Shared Variables:
;       KeyDebounceCounter - [Write] -  An array of size NUM_KEYS which contains the
;                                       debounce counters for NUM_KEYS keys. Elements of
;                                       the array has the counter for each key.
; Global Variables:
;       None
```

```
65  ; Input:
66  ;       None
67  ; Output:
68  ;       None
69  ; Error Handling:
70  ;       None
71  ; Algorithms:
72  ;       None
73  ; Data Structures:
74  ;       None
75  ; Registers Changed:
76  ;       DI, Flags
77  ; Limitations:
78  ;       None
79  ; Known bugs:
80  ;       None
81  ; Special Notes:
82  ;       None
83  ; Author:
84  ;       Sunghoon Choi
85  ; Revision History:
86  ;        10/30/2016   Sunghoon Choi    Created
87  ;        10/31/2016   Sunghoon Choi    Initial Compilation
88  ;        11/3/2016    Sunghoon Choi    Revised Comments
89
90  InitKeypad  PROC    NEAR
91              PUBLIC  InitKeypad
92
93  InitKeypadIndex:
94      MOV DI, FIRST_KEY        ;We begin initializing the array from its first element
95
96  InitKeyCounter:
97      MOV KeyDebounceCounter[DI], KEY_COUNTER_MAX ;Initialize the KeyDebounceCounter array
98                                                  ;to the state of no buttons having been
99                                                  ;pressed. Thus, each counter will have
100                                                 ;its maximum value.
101     INC DI                                      ;Go to the next element for
        initialization
102     CMP DI, NUM_KEYS                            ;Is this the end of this array?
103     JB InitKeyCounter                           ;No, continue initialization.
104 ;    JGE EndInitKeypad                          ;Yes, finish initialization.
105 EndInitKeypad:
106      RET                                        ;End of InitKeypad procedure.
107
108 InitKeypad  ENDP
109
110
111
112
113 ; KeypadEventHandler
114 ;
115 ; Description:
116 ;       This eventhandler reads values from the keypad and debounces the current key
    if it
117 ;       is pressed. It sets a counter for each key. The counter indicates the number of
118 ;       interrupts(KeypadEventHandler) the system need to call for enqueuing a key
    pressed
119 ;       event. If a button is pressed, the counter will be decremented at every
    interrupt
120 ;       call(1 ms) and when the counter reaches zero, the key-pressed event will be
    enqueued
121 ;       and the counter will be set to Auto-Repeat counter value.
122 ;       If a certain key was found to be not pressed, it will reset the counter to its
123 ;       maximum value. KeypadEventHandler is called by Timer2EventHandler every 1ms.
124 ; Operation:
125 ;       It receives a value from the current keypad row.Our keypad has NUM_ROWS rows
```

```
    and each
126 ;       row has KEYS_IN_ROW keys. Although the value read from the row is originally
127 ;       BITS_IN_BYTE bits, it extracts only the KEYS_IN_ROW bits of current key row by
    using
128 ;       mask bits to remove the high (BITS_IN_BYTE - KEYS_IN_ROW) bits of AL.
129 ;       Next, it checks if the first key of the current row is pressed by using a
    column mask
130 ;       If it is not pressed, reset the key's debounce counter to the original maximum
    value.
131 ;       If the key is pressed, decrement the debounce counter of the current key.
132 ;       The debounce counter is stored in an array of NUM_KEYS elements. Each element
133 ;       contains the counter for each key.
134 ;       Now, if the debounce counter of the current key has reached zero, it calls
135 ;       EnqueueEvent with the arguments of KEYPAD_EVENT and the current key's
    identification
136 ;       number. Also, the debounce counter for the pressed key will be set to
    AUTO_REPEAT
137 ;       Counter value. After a key is checked(whether debounced or reset), it checks
    the next
138 ;       key by shifting the column mask to left. When it is done with checking the
139 ;       current row, it goes to the next row by incrementing the Keypad address since
140 ;       the address of each row is separated by one byte.
141 ;       After incrementing the keypad address, it goes back to check the first key of
    the row
142 ;       by retreiving and using the initial column mask.
143 ;       The eventhandler ends when it has gone through all NUM_KEYS keys
144 ;       (until the last key of NUM_ROWth row)
145 ; Arguments:
146 ;      None
147 ; Return Value:
148 ;       None
149 ; Local Variables:
150 ;      RowAddr(DX)            -    The address of the current key row
151 ;       CurrentRowValue(AL)   -    The values read from the RowAddr
152 ;       CurrentKeyValue(AL)   -    The value of the current key
153 ;       KeypadColMask(BL)     -    A mask used for selecting the current key inside the
    current row.
154 ;       KeyIndex(DI)          -    The index for current key(=column index of the key)
155 ; Shared Variables:
156 ;       KeyDebounceCounter   - [R/W] - An array of size NUM_KEYS which contains the
157 ;                                      debounce counters for NUM_KEYS keys.
158 ;                                      Elements of the array has the counter for each
    key.
159 ; Global Variables:
160 ;       None
161 ; Input:
162 ;       Keypad of NUM_KEYS keys.
163 ; Output:
164 ;       None
165 ; Error Handling:
166 ;       None
167 ; Algorithms:
168 ;       None
169 ; Data Structures:
170 ;       None
171 ; Registers Changed:
172 ;       AX, BX, CX, DI, Flags
173 ; Limitations:
174 ;       None
175 ; Known bugs:
176 ;       None
177 ; Special Notes:
178 ;       None
179 ; Author:
180 ;       Sunghoon Choi
181 ; Revision History:
```

```
;       10/30/2016   Sunghoon Choi    Created
;       10/31/2016   Sunghoon Choi    Initial Compilation
;       11/3/2016    Sunghoon Choi    Revised comments


KeypadEventHandler  PROC    NEAR
                    PUBLIC  KeypadEventHandler


InitKeyRowAndCol:
    MOV DX, KEY_FIRST_ROW_ADDR      ;The KeypadEventHandler scans the keypad row by row.
                                    ;First, it scans the first row.
    MOV BL, KEYPAD_COLUMN_MASK      ;Column mask will be used to indicate a specific key
                                    ;inside a row.
    XOR DI, DI                      ;The eventhandler will scan the keypad and each key
    has
                                    ;its own index. The eventhandler handles the first
                                    key
                                    ;(For an array, index 0 is the first element) at the
                                    ;beginning.

GetKeyRowVal:
    IN AL, DX                       ;Read a byte from the current row.
    AND AL, KEYPAD_ROW_MASK         ;Remove unnecessary high bits and leave only the
                                    ;bits of the current row.
    MOV CL, AL                      ;Need to save the value of current row since
                                    ;AL register will be altered by the loop beginning
                                    with
                                    ;GetAKeyVal.
GetAKeyVal:
    MOV AL, CL                      ;Retrieve the bits of current row to continue
                                    ;handling the current row.
    AND AL, BL                      ;Get a specific key inside current row by using
                                    ;the column mask.
    CMP AL, KEY_PRESSED             ;Is the key pressed?
    JE StartDebounce                ;Yes, the key is pressed. Start deboucing
    ;JNE ResetCounter               ;No, the key is not pressed. Reset the debounce
    counter.
ResetCounter:
    MOV KeyDebounceCounter[DI], KEY_COUNTER_MAX ;The key is not pressed. So we need to
                                                ;reset the counter and start the
                                                debouncing
                                                ;from the very beginning again.
    JMP GetNextKey                              ;Done with handling an unpressed key.
                                                ;So go to handle the next key.

StartDebounce:
    DEC KeyDebounceCounter[DI]                  ;Since the key was pressed, we need to
                                                ;decrement the debounce counter of the
                                                ;current key. The key will be considered
                                                ;"pressed" when the counter reaches zero.
    JNZ GetNextKey                              ;If the debounce counter has not
    reached zero
                                                ;yet, do not enqueue the key event and
                                                go
                                                ;check the next key.

    ;JZ    EnqueueKeyEvent                      ;The debounce counter has reached zero.
                                                ;Thus, we should consider this key
                                                "pressed"
                                                ;and enqueue this event to EventBuf.
EnqueueKeyEvent:
    MOV KeyDebounceCounter[DI], REPEAT_COUNTER  ;Since we confirmed that the current
    key is
                                                ;pressed, we need to check if the key is
                                                ;pressed consistently. If the key is
```

```
                                        pressed
239                                     ;for REPEAT_COUNTER ms, it treats the key
240                                     ;as being pressed again and enqueue the
241                                     ;event again.
242         MOV AX, DI                  ;AL is an argument for EnqueueEvent.
243                                     ;It contains the key index, or key value.
244         MOV AH, KEY_EVENT           ;AH is also an argument for EnqueueEvent.
245                                     ;It contains the KEY_EVENT.
246
247         PUSHA                       ;save all general-purpose registers since
248                                     ;calling EnqueueEvent may change the
                                        values
249                                     ;of them.
250         CALL EnqueueEvent           ;Enqueue the event that the current key
            has
251                                     ;been pressed to EventBuf.
252         POPA                        ;Retrieve all general-purpose registers
253                                     ;since the program has returned from
254                                     ;EnqueueEvent procedure.
255
256     GetNextKey:
257         INC DI                      ;Get ready to save the next key's
258                                     ;debounce counter in the next element of
259                                     ;KeyDebounceCounter array(If pressed).
260         SHL BL, COL_INTERVAL        ;We have to check the next key in the
261                                     ;current row. So shift the column index
262                                     ;to left to get the key of next column of
263                                     ;same row.
264
265         AND BL, KEYPAD_ROW_MASK     ;Limit the column index to the number of
266                                     ;keys in a row. The column index will
267                                     ;become zero if it exceeds the number of
268                                     ;keys in a row.
269         CMP BL, COL_OVER_LIMIT      ;Was the key handled on the last loop
270                                     ;the last key of current row?
271         JNE GetAKeyVal              ;No. There's still keys to be checked in
272                                     ;the current row. Go handle the next key
273                                     ;in the same row.
274     ;   JE  GoNextRow               ;Yes, we're done with this row.
275                                     ;Go handle the next row.
276     GoNextRow:
277         INC DX                      ;Get the next keypad row's address
278                                     ;to read from next row.
279         MOV BL, KEYPAD_COLUMN_MASK  ;Since we are going to the next row,
280                                     ;We should begin checking from the first
281                                     ;column again.
282     CheckLastRow:
283         CMP DX,  KEY_LAST_ROW_ADDR  ;Was the previous loop hnadling the
            last row?
284         JBE GetKeyRowVal            ;No. Go read the next row.
285         ;JE EndKeypadEventHandler   ;Yes. We've handled all rows and all
            keys.
286
287     EndKeypadEventHandler:
288         RET                         ;Since we've handled all keys,
289                                     ;finish the KeypadEventHandler.
290     KeypadEventHandler ENDP
291
292
293
294
295
296     CODE ENDS
297
298
299
```

```
DATA    SEGMENT PUBLIC  'DATA'
      KeyDebounceCounter    DB  NUM_KEYS    DUP (?)    ;KeyDebounceCounter contains
                                                      ;the debounce counters for each keys.
DATA    ENDS


END
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                               ;
;                                  Keypad.inc                                   ;
;                                  Homework5                                    ;
;                                 Sunghoon Choi                                 ;
;                                                                               ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Description:
;  This file contains the definitions for the Keypad functions for RoboTrike.
;
; Revision History:
;    10/30/2016   Sunghoon Choi    Created
;    11/3/2016    Sunghoon Choi    Updated documentation(comments) for constants.

KEY_COUNTER_MAX     EQU     50         ;Initial value of the counter which is used for
                                       ;debouncing each key.

FIRST_KEY           EQU     0          ;The index of the first key on
keypad
COL_INTERVAL        EQU     1          ;Number of bit shift needed to move to next
column key

REPEAT_COUNTER      EQU     250        ;Initial value of the counter which is used for
                                       ;handling the consistently pressed key.
NUM_KEYS            EQU     16         ;Number of keys on the board.

KEY_PRESSED         EQU     0          ;the value read from a pressed key
COL_OVER_LIMIT      EQU     00000000B  ;indicates that the column index has exceeded the
                                       ;number of keys in a row

KEY_FIRST_ROW_ADDR  EQU     80H        ;The address of the first key row
KEY_LAST_ROW_ADDR   EQU     83H        ;The address of the last key row
KEYPAD_ROW_MASK     EQU     00001111B  ;The mask used to extract only the key values
                                       ;and remove all other bits
KEYPAD_COLUMN_MASK  EQU     00000001B  ;The mask used to indicate a specific key
                                       ;inside a row.
```

```
NAME Motors
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                              ;
;                                    Motors                                    ;
;                                  Homework 6                                  ;
;                                 Sunghoon Choi                                ;
;                                                                              ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; Description:
;      This file contains the functions necessary for handling DC motors and laser.
;
; Table of Contents:
;     InitMotorLaser     -    Initializes all variables and arrays related to Motors routine.
;     SetMotorSpeed      -    Sets the speed and angle of motors of RoboTrike.
;     GetMotorSpeed      -    Gets the current speed setting of RoboTrike.
;     GetMotorDirection -     Gets the current direction of movement setting for RoboTrike.
;     SetLaser           -    Sets the laser setting to turn it on or turn it off.
;     GetLaser           -    Gets the current laser status of RoboTrike.
;    MotorLaserEventHandler    - Output values to parallel port B to control motors and laser.
;                              - It is called by Timer1 of 4KHz frequency.
;
;
; Revision History:
;      11/6/2016     Sunghoon Choi       Created
;      11/7/2016     Sunghoon Choi       Corrected syntax error for Force tables.
;      11/8/2016     Sunghoon Choi       Initial Compilation
;      11/11/2016    Sunghoon Choi       Updated documentation



$INCLUDE(Motors.inc)        ;Include the.inc file which contains constatns for Motors.asm

EXTRN   Sin_Table:NEAR      ;import Sin_Table to calculate the dot product of force and
                            ;velocity.
EXTRN   Cos_Table:NEAR      ;import Cos_Table to calculate the dot product of force and
                            ;velocity.

CGROUP GROUP CODE
DGROUP GROUP DATA

CODE SEGMENT PUBLIC 'CODE'

        ASSUME  CS:CGROUP, DS:DGROUP

; XForces
;
; Description:
;        This is the table for X component of force vectors of each motors.
;        The values are in Q0.15 format.
; Notes:
;        This table is declared PRIVATE to prevent other codes accessing the table.
;        Also, READ ONLY tables should always be in the code segment so that in a standalone
;        system it will be located in the ROM with the code.
;
; Author:      Sunghoon Choi
; Revision history:    11/6/2016        Sunghoon Choi    Created
;                      11/7/2016        Sunghoon Choi    Corrected syntax error
;                      11/11/2016       Sunghoon Choi    Updated documentation

XForces    LABEL   WORD

    DW    07FFFH    ;XForce_Motor1
```

```
64      DW    0C000H    ;XForce_Motor2
65      DW    0C000H    ;XForce_Motor3
66
67
68  ; YForces
69  ;
70  ; Description:
71  ;       This is the table for Y component of force vectors of each motors.
72  ;       The values are in Q0.15 format.
73  ; Notes:
74  ;       This table is declared PRIVATE to prevent other codes accessing the table.
75  ;       Also, READ ONLY tables should always be in the code segment so that in a
    standalone
76  ;       system it will be located in the ROM with the code.
77  ;
78  ; Author:       Sunghoon Choi
79  ; Revision history:    11/6/2016        Sunghoon Choi    Created
80  ;                      11/7/2016        Sunghoon Choi    Corrected syntax error
81  ;                      11/11/2016       Sunghoon Choi    Updated
    documentation
82
83  YForces     LABEL  WORD
84
85      DW    00000H    ;YForceMotor1
86      DW    09127H    ;YForceMotor2
87      DW    06ED9H    ;YForceMotor3
88
89
90  ; BackDirMask
91  ;
92  ; Description:
93  ;       This is a table of the masks to be used to reverse the directions of motors.
94  ;
95  ; Notes:
96  ;       This table is declared PRIVATE to prevent other codes accessing the table.
97  ;       Also, READ ONLY tables should always be in the code segment so that in a
    standalone
98  ;       system it will be located in the ROM with the code.
99  ;
100 ; Author:       Sunghoon Choi
101 ; Revision history:    11/6/2016        Sunghoon Choi    Created
102 ;                      11/7/2016        Sunghoon Choi    Corrected syntax error
103 ;                      11/11/2016       Sunghoon Choi    Updated documentation
104
105 BackDirMask     LABEL  BYTE
106
107     DB 00000001B       ;ReverseMaskMotor1
108     DB 00000100B       ;ReverseMaskMotor2
109     DB 00010000B       ;ReverseMaskMotor3
110
111 ; TurnOnMask
112 ;
113 ; Description:
114 ;       This is a table of the masks to be used to turn on each motors.
115 ;
116 ; Notes:
117 ;       This table is declared PRIVATE to prevent other codes accessing the table.
118 ;       Also, READ ONLY tables should always be in the code segment so that in a
    standalone
119 ;       system it will be located in the ROM with the code.
120 ;
121 ; Author:       Sunghoon Choi
122 ; Revision history:    11/6/2016        Sunghoon Choi    Created
123 ;                      11/7/2016        Sunghoon Choi    Corrected syntax error
124 ;                      11/11/2016       Sunghoon Choi    Updated documentation
125
```

```
TurnOnMask      LABEL  BYTE

    DB 00000010B       ;TurnOnMaskMotor1
    DB 00001000B       ;TurnOnMaskMotor2
    DB 00100000B       ;TurnOnMaskMotor3


; InitMotorLaser
;
; Description:
;      Initializes all variables and arrays related to Motor routine.
; Operation:
;      Inserts INIT_PULSE_WIDTH to all elements of pulseWidths by incrementing the
;      array index every loop. It repeats the loop until the index reaches NUM_MOTORS.
;      Inserts INIT_PULSE_COUNTER to pulseWidthCounter.
;      Inserts INIT_SPEED to driveSpeed.
;      Inserts INIT_ANGLE to driveAngle.
;      Inserts INIT_LASER_STAT to laserStatus.
; Arguments:
;     None
; Return Value:
;     None
; Local Variables:
;     MotorIndex(DI)      -    The index of current motor
; Shared Variables:
;      driveSpeed(DS)     - [Write] - The speed at which the RoboTrike is to move
;      driveAngle(DS)     - [Write] - The angle at which the RoboTrike is to move in
degrees
;      pulseWidths(DS)    - [Write] - An array which contains the pulse widths for
;                                     NUM_MOTORS motors.
;      pulseWidthCounter(DS)  - [Write] - A counter used for Pulse Width Modulation
control
;      laserStatus(DS)        - [Write] - Indicates whether the laser is on or off.
; Global Variables:
;     None
; Input:
;     None
; Output:
;     None
; Error Handling:
;     None
; Algorithms:
;     None
; Data Structures:
;     None
; Registers Changed:
;     DI, Flags
; Limitations:
;     None
; Known bugs:
;     None
; Special Notes:
;     None
; Author:
;     Sunghoon Choi
; Revision History:
;     11/6/2016    Sunghoon Choi      Created
;     11/8/2016    Sunghoon Choi      Initial Compilation
;     11/11/2016   Sunghoon Choi      Updated documentation


InitMotorLaser  PROC    NEAR
                PUBLIC  InitMotorLaser

    XOR DI, DI                  ;We begin initializing pulseWidths array with its
                                ;first element.
```

```
InitPulseWidths:
    MOV pulseWidths[DI], INIT_PULSE_WIDTH     ;pulse widths for each motors are
                                              ;initialized to INIT_PULSE_WIDTH.
    INC DI                                    ;Proceed to initialize the pulse width of
    next motor.
    CMP DI, NUM_MOTORS                        ;Is this the end of pulseWidths array?
    JB  InitPulseWidths                       ;No, go back to the beginning of the loop
    and
                                              ;initialize next pulse width.
;   JGE InitPulseWidthCnt                     ;Yes, we are done with initializing pulse
widths.
                                              ;Go initialize the pulse width counter.

InitPulseWidthCnt:
    MOV pulseWidthCounter, INIT_PULSE_COUNTER ;Initialize pulseWidthCounter to
                                              ;INIT_PULSE_COUNTER

InitSpeednAngle:
    MOV driveSpeed, INIT_SPEED         ;Initialize driveSpeed to INIT_SPEED
    MOV driveAngle, INIT_ANGLE         ;Initialize driveAngle to INIT_ANGLE

InitLaserStatus:
    MOV laserStatus, INIT_LASER_STAT   ;Initialize laserStatus to INIT_LASER_STAT
EndInitMotorLaser:
    RET                                ;End of InitMotorLaser procedure.

InitMotorLaser   ENDP




; SetMotorSpeed
;
; Description:
;    The function is called with two arguments. First argument, speed(unsigned word),
;    is passed in AX. Speed of IGNORE_SPEED indicates the current speed should not be
changed
;    If the speed is not IGNORE_SPEED, driveSpeed will be updated.
;    RoboTrike is at its full speed when the speed is equal to MAX_SPEED.
;    The second argument, angle, is a signed value and is passed in BX.
;    STRAIGHT_ANGLE degree indicates that the Robotrike should move straight ahead
relative to
;    RobotTrike orientation while an angle of IGNORE_ANGLE indicates the current
direction of
;    travel should not be changed. If the angle is not IGNORE_ANGLE, driveAngle will be
updated.
;    The angle will be normalized to the range of [STRAIGHT_ANGLE,FULL_ANGLE-1] and
speed will be
;    normalized to range of [MIN_SPEED,NORM_MAX_SPEED] for dot product calculation in
the procedure.
; Operation:
;    It first checks if the speed is IGNORE_SPEED. If it is, skip updating driveSpeed.
;    If the speed is not IGNORE_SPEED, it updates driveSpeed with the argument-speed.
;    Then, it normalizes the speed by shifting speed value to right by one bit to make
the
;    highest bit zero. This normalization is needed since the function is going to use
IMUL
;    instruction to multiply the speed with force vectors and cosine or sine
;    values. Shifting the speed to right by 1 prevents speed being treated as
;    negative value by IMUL while degrading the precision. The range of speed gets
halved.
;    It saves the normalized speed in driveSpeed.
;    Now it checks if the angle is IGNORE_ANGLE. If so, skip updating angle.
;    If the angle is not IGNORE_ANGLE, it normalizes the angle to the range of
;    [STRAIGHT_ANGLE,FULL_ANLGE-1]. If the angle is positive, it executes
```

```
;    "Normalized Angle = angle mod FULL_ANGLE".
;    If the angle is negative, it executes
;    "Normalized Angle = FULL_ANGLE - {abs(angle) mod FULL_ANGLE}".
;    It saves the normalized angle in dirveAngle.
;    Now that we have normalized both speed and angle, the function performs
;    the dot product of motors' forces and the velocity vectors using the Sin_Table and
;    Cos_Table as: "DotProductResult = XForces[MotorIndex] * NormSpeed *
Cos_Table[TrigIndex]
;   + YForces[MotorIndex] * NormSpeed * Sin_Table[TrigIndex]".
;    Q0.15 fixed point value operations are used for dot product calculation.
;    Thus, duplicated sign bits are removed at the end of calculation.
;    The calculated pulseWidth is stored in the pulseWidths array. When calculating and
;    storing the pulseWidth for current motor is done, increment the motor index and
repeat
;    the same process for remaining motors. When it has gone through all NUM_MOTORS
motors,
;    the procedure ends.
; Arguments:
;     speed - AX - The absolute(unsigned) speed at which the RoboTrike is to move
;    angle - BX - The signed angle at which the RoboTrike is to move in degrees
; Return Value:
;   None
; Local Variables:
;    speed         - AX - The absolute(unsigned) speed at which the RoboTrike is to move
;    angle         - BX - The signed angle at which the RoboTrike is to move in degrees
;    TrigIndex     - SI - The index used to obtain cos or sin value for each angle from
;                         Cos_Tables and Sin_Tables
;    MotorIndex    - DI - The index of current motor
;    NormSpeed     - AX - The normalized speed.
; Shared Variables:
;    pulseWidths(DS) - [Write] - An array which contains the pulse widths for
;                                NUM_MOTORS motors.
;    driveSpeed(DS)  - [Read/Write] - The speed at which the RoboTrike is to move
;    driveAngle(DS)  - [Read/Write] - The angle at which the RoboTrike is to move in
;                                     degrees
; Global Variables:
;    None
; Input:
;    None
; Output:
;    None
; Error Handling:
;    None
; Algorithms:
;    None
; Data Structures:
;    None
; Registers Changed:
;    AX, BX, CX, DX, DI, SI, Flags
; Limitations:
;    The precision of speed goes down due to shifting the speed to right by one bit for
IMUL
;    instruction. Also, there is a precision loss in dot product due to
truncation.
; Known bugs:
;    None
; Special Notes:
;    None
; Author:
;    Sunghoon Choi
; Revision History:
;     11/6/2016    Sunghoon Choi      Created
;     11/8/2016    Sunghoon Choi      Initial Compilation
;     11/11/2016   Sunghoon Choi      Updated documentation

SetMotorSpeed   PROC    NEAR
```

```
305                 PUBLIC SetMotorSpeed
306
307     XOR DI, DI          ;MotorIndex is initialized to F_INDEX_MOTOR1 to calculate the
308                         ;dot product of vectors for first motor.
309                         ;(For an array, index 0 is the first element)
310     XOR SI, SI          ;TrigIndex is initialized to STRAIGHT_ANGLE.
311                         ;TrigIndex will be updated to the current angle setting.
312 CheckIgnoreSpeed:
313     CMP AX, IGNORE_SPEED ;Is the speed equal to IGNORE_SPEED?
314     JE CheckIgnoreAngle  ;Yes, skip updating the shared variable driveSpeed
315                         ;and go check the angle.
316 ;   JNE UpdateDriveSpeed ;No. Update driveSpeed with the argument value.
317
318 UpdateDriveSpeed:
319     MOV driveSpeed, AX   ;Update driveSpeed with the argument-speed's value.
320
321 CheckIgnoreAngle:
322     CMP BX, IGNORE_ANGLE ;Is the angle IGNORE_ANGLE?
323     JE  CalcDotProduct   ;Yes, skip updating driveAngle and start calculating pulse
    widths.
324 ;   JNE NormAngle        ;No. Start normalizing the angle.
325 NormAngle:
326     CMP BX, STRAIGHT_ANGLE    ;Is the angle negative?
327     JL NormNegAngle           ;Normalize negative angle
328 ;   JGE NormPosAngle          ;Normalize positive angle
329 NormPosAngle:
330     MOV AX, BX                ;Set the dividend to angle.
331     MOV BX, FULL_ANGLE        ;Set the divisor to FULL_ANGLE.
332     XOR DX,DX                 ;Clear DX for DIV instruction.
333     DIV BX                    ;DX = normalized angle = angle mod FULL_ANGLE
334                               ;Now the normalized angle is in the range of
                                  [0,FULL_ANGLE-1]
335     MOV driveAngle, DX        ;Update driveAngle with the normalized angle
336     JMP CalcDotProduct        ;Now that we're done with normalizing speed and angle,
337                               ;go calculate the pulse widths.
338
339 NormNegAngle:
340     NEG BX                    ;Get the absolute value of the negative angle.
341     MOV AX, BX                ;Set the dividend to the absolute value of the angle.
342     MOV BX, FULL_ANGLE        ;Set the divisor to FULL_ANGLE
343     XOR DX,DX                 ;Clear DX for DIV instruction.
344     DIV BX                    ;DX = abs(angle) mod FULL_ANGLE
345     NEG DX                    ;DX = - (abs(angle) mod FULL_ANGLE)
346
347     ADD DX, FULL_ANGLE        ;DX = normalized angle = FULL_ANGLE-(abs(angle) mod
    FULL_ANGLE)
348     MOV driveAngle, DX        ;update driveAngle with the noramlized angle.
349
350
351 CalcDotProduct:               ;In this loop, it calculates the pulse widths for each
    motor
352                               ;by calculating the dot product of force and velocity.
353
354     SHL DI,NUM_SHIFT_DOUBLE   ;Double MotorIndex since XForces and YForces tables
355                               ;are WORD tables.
356     MOV  AX, driveSpeed       ;Retrieve the updated driveSpeed to AX.
357
358     SHR AX, NUM_SHIFT_ERASE_SIGN
359                                 ;Get rid of the sign bit of AX to obtain the absolute
                                    value
360                                 ;of driveSpeed for normalization.
361                                 ;Now, the normalized speed, normSpeed, is in the range of
362                                 ;[STRAIGHT_ANGLE, FULL_ANGLE]
363
364     PUSH AX                     ;Save NormSpeed since AX will be changed.
365
```

```
366         MOV  BX, CS:XForces[DI]     ;Obtain the X component of force vector of current motor
367         IMUL BX                     ;DX:AX = Fx * v while
368                                     ;       Fx = X component of force vector of current
                                        motor
369                                     ;       v = NormSpeed
370
371         MOV  AX, DX                 ;Truncate to DX for normalization.
372                                     ;AX = Fx * v
373         MOV SI, driveAngle          ;Obtain the current angle setting to do the trigonometric
374                                     ;calculation. Since driveAngle is in the range of
375                                     ;[STRAIGHT_ANGLE,FULL_ANGLE-1], we don't need further
376                                     ;normalization. Thus, driveAngle = NormAngle
377         SHL SI, NUM_SHIFT_DOUBLE    ;Double the angle since Cos table and Sin table are WORD
378                                     ;tables.
379         MOV  BX, CS:Cos_Table[SI]   ;BX = Cos(NormAngle)
380         IMUL BX                     ;DX:AX = Fx*v*Cos(NormAngle)
381         MOV CX, DX                  ;Truncate to DX for normalization.
382                                     ;CX = Fx * v * Cos(NormAngle) = X factor of pulse width
383
384         POP AX                      ;Retrieve NormSpeed
385
386         MOV BX, CS:YForces[DI]      ;Obtain the Y component of force vector of current motor
387         IMUL BX                     ;DX:AX = Fy * v while
388                                     ;       Fy = Y component of force vector of current
                                        motor
389                                     ;       v = NormSpeed
390         MOV AX, DX                  ;Truncate to DX for normalization.
391                                     ;AX = Fy*v
392
393         MOV BX, CS:Sin_Table[SI]    ;BX = Sin(NormAngle)
394         IMUL BX                     ;DX:AX = Fy*v*Sin(NormAngle)
395         MOV BX, DX                  ;Truncate to DX for normalization.
396                                     ;BX = Fy * v * Sin(NormAngle) = Y factor of pulse width
397
398
399         ADD CX, BX                  ;pulseWidth = Fx*v*Cos(NormAngle) + Fy*v*Sin(NormAngle)
400
401         SAL CX, DUPSIGN_0DOT15_MUL2 ;Remove duplicated extra sign bits caused by
402                                     ;Q0.15 multiplications
403                                     ;Thus, remove the extra sign bits by shifting the
                                        result to
404                                     ;left by DUPSIGN_0DOT15_MUL2.
405
406         SHR DI, NUM_SHIFT_HALF      ;Halve MotorIndex back since pulseWidhts is a BYTE array.
407         MOV pulseWidths[DI], CH     ;Truncate the pulseWidth of current motor to CH and
408                                     ;save it in the pulseWidths array.
409
410         INC DI                      ;Proceed to handle next motor
411
412         CMP DI, NUM_MOTORS          ;Is this the last motor?
413         JL CalcDotProduct           ;No, handle the next motor.
414     ;   JGE EndSetMotorSpeed        ;Yes, finish SetMotorSpeed
415
416     EndSetMotorSpeed:
417         Ret                         ;End of SetMotorSpeed procedure.
418     SetMotorSpeed ENDP
419
420
421
422     ; GetMotorSpeed
423     ;
424     ; Description:
425     ;    The function is called with no arguments and returns the current speed setting of
426     ;    RoboTrike in AX. A speed of MAX_SPEED indicates the maximum speed and a value of
427     ;    MIN_SPEED indicates that the RoboTrike is stopped.
428     ; Operation:
```

```
;    Return driveSpeed in AX.
; Arguments:
;   None
; Return Value:
;   AX(driveSpeed)   -    The speed at which the RoboTrike is to move
; Local Variables:
;   None
; Shared Variables:
;    driveSpeed(DS)  - [Read] -    The speed at which the RoboTrike is to move
; Global Variables:
;   None
; Input:
;   None
; Output:
;   None
; Error Handling:
;   None
; Algorithms:
;   None
; Data Structures:
;   None
; Registers Changed:
;   AX
; Limitations:
;   None
; Known bugs:
;   None
; Special Notes:
;   None
; Author:
;   Sunghoon Choi
; Revision History:
;      11/6/2016    Sunghoon Choi      Created
;      11/8/2016    Sunghoon Choi      Initial Compilation
;      11/11/2016   Sunghoon Choi      Updated documentation


GetMotorSpeed   PROC    NEAR
                PUBLIC  GetMotorSpeed

MOV AX, driveSpeed    ;Return driveSpeed to AX
RET                   ;End of GetMotorSpeed procedure
GetMotorSpeed ENDP



; GetMotorDirection
;
; Description:
;    The function is called with no arguments and returns the current direction of
movement
;    setting for the RoboTrike as an angle in degrees in AX. An angle of STRAIGHT_ANGLE
indicates
;    straight ahead relative to the RoboTrike orientation and angles are measured
clockwise.
;    The value returned will always be between STRAIGHT_ANGLE and FULL_ANGLE-1
inclusively.
; Operation:
;    Returns driveAngle in AX.
; Arguments:
;   None
; Return Value:
;   AX(driveAngle)  - The angle at which the RoboTrike is to move in degrees
; Local Variables:
;   None
; Shared Variables:
```

```
491   ;    driveAngle(DS)  - [Read] - The angle at which the RoboTrike is to move in degrees
492   ; Global Variables:
493   ;   None
494   ; Input:
495   ;   None
496   ; Output:
497   ;   None
498   ; Error Handling:
499   ;   None
500   ; Algorithms:
501   ;   None
502   ; Data Structures:
503   ;   None
504   ; Registers Changed:
505   ;    AX
506   ; Limitations:
507   ;    None
508   ; Known bugs:
509   ;    None
510   ; Special Notes:
511   ;    None
512   ; Author:
513   ;   Sunghoon Choi
514   ; Revision History:
515   ;     11/6/2016    Sunghoon Choi   Created
516   ;     11/8/2016    Sunghoon Choi   Initial Compilation
517   ;     11/11/2016   Sunghoon Choi   Updated documentation
518
519
520   GetMotorDirection   PROC    NEAR
521                       PUBLIC  GetMotorDirection
522
523   MOV AX, driveAngle          ;Return driveAngle to AX.
524   RET                         ;End of GetMotorDirection procedure.
525   GetMotorDirection ENDP
526
527
528
529   ; SetLaser
530   ;
531   ; Description:
532   ;    The function is passed a single argument (onoff) in AX that indicates whether to
      turn
533   ;    the RoboTrike laser on or off. Value of LASER_OFF turns the laser off and a value
534   ;    other than LASER_OFF turns it on.
535   ; Operation:
536   ;    Inserts the value of argument AX into laserStatus variable.
537   ; Arguments:
538   ;   laserPowerArg(AX) - The configuration value to turn the laser on or off.
539   ; Return Value:
540   ;   None
541   ; Local Variables:
542   ;   None
543   ; Shared Variables:
544   ;    laserStatus(DS)  - [Write] - Indicates whether the laser is on or off.
545   ; Global Variables:
546   ;   None
547   ; Input:
548   ;   None
549   ; Output:
550   ;   None
551   ; Error Handling:
552   ;   None
553   ; Algorithms:
554   ;   None
555   ; Data Structures:
```

```
;   None
; Registers Changed:
;   None
; Limitations:
;   None
; Known bugs:
;   None
; Special Notes:
;   None
; Author:
;   Sunghoon Choi
; Revision History:
;     11/6/2016    Sunghoon Choi   Created
;     11/8/2016    Sunghoon Choi   Initial Compilation
;     11/11/2016   Sunghoon Choi   Updated documentation

SetLaser        PROC    NEAR
                PUBLIC  SetLaser
MOV laserStatus, AX     ;Sets laserStatus with the argument laserPowerArg.
RET                     ;End of SetLaser procedure.
SetLaser ENDP



; GetLaser
;
; Description:
;    The function is called with no arguments and returns the status of the RoboTrike
laser
;    in AX. A value of LASER_OFF indicates the laser is off and a value other than
LASER_OFF
;    indicates the laser is on.
; Operation:
;    Returns laserStatus in AX.
; Arguments:
;    None
; Return Value:
;    laserStatus(AX)  -   Indicates whether the laser is on or off.
; Local Variables:
;    None
; Shared Variables:
;     laserStatus(AX) - [Read] -  Indicates whether the laser is on or off.
; Global Variables:
;    None
; Input:
;    None
; Output:
;    None
; Error Handling:
;    None
; Algorithms:
;    None
; Data Structures:
;    None
; Registers Changed:
;    AX
; Limitations:
;    None
; Known bugs:
;    None
; Special Notes:
;    None
; Author:
;    Sunghoon Choi
; Revision History:
;     11/6/2016    Sunghoon Choi   Created
```

```
;     11/8/2016    Sunghoon Choi   Initial Compilation
;     11/11/2016   Sunghoon Choi   Updated documentation

GetLaser        PROC    NEAR
                PUBLIC GetLaser

MOV AX, laserStatus             ;Return laserStatus to AX.
RET                             ;End of GetLaser procedure.
GetLaser ENDP

; SetTurretAngle
;
; Description:
;
; Operation:
;    Returns laserStatus in AX.
; Arguments:
;    None
; Return Value:
;
; Local Variables:
;    None
; Shared Variables:
;
; Global Variables:
;    None
; Input:
;    None
; Output:
;    None
; Error Handling:
;    None
; Algorithms:
;    None
; Data Structures:
;    None
; Registers Changed:
;    AX
; Limitations:
;    None
; Known bugs:
;    None
; Special Notes:
;    None
; Author:
;    Sunghoon Choi
; Revision History:


SetTurretAngle        PROC    NEAR
                      PUBLIC  SetTurretAngle

    RET

SetTurretAngle ENDP

; SetRelTurretAngle
;
; Description:
;
; Operation:
;    Returns laserStatus in AX.
; Arguments:
;    None
; Return Value:
;
```

```
; Local Variables:
;    None
; Shared Variables:
;
; Global Variables:
;    None
; Input:
;    None
; Output:
;    None
; Error Handling:
;    None
; Algorithms:
;    None
; Data Structures:
;    None
; Registers Changed:
;    AX
; Limitations:
;    None
; Known bugs:
;    None
; Special Notes:
;    None
; Author:
;    Sunghoon Choi
; Revision History:


SetRelTurretAngle        PROC    NEAR
                         PUBLIC  SetRelTurretAngle

    RET

SetRelTurretAngle ENDP

; SetTurretElevation
;
; Description:
;
; Operation:
;    Returns laserStatus in AX.
; Arguments:
;    None
; Return Value:
;
; Local Variables:
;    None
; Shared Variables:
;
; Global Variables:
;    None
; Input:
;    None
; Output:
;    None
; Error Handling:
;    None
; Algorithms:
;    None
; Data Structures:
;    None
; Registers Changed:
;    AX
; Limitations:
;    None
```

```
752  ; Known bugs:
753  ;    None
754  ; Special Notes:
755  ;    None
756  ; Author:
757  ;    Sunghoon Choi
758  ; Revision History:
759
760
761  SetTurretElevation        PROC    NEAR
762                            PUBLIC  SetTurretElevation
763
764      RET
765
766  SetTurretElevation ENDP
767
768
769  ; MotorLaserEventHandler
770  ;
771  ; Description:
772  ;    This is an eventhandler which activates motors and laser on parallel port B.
773  ;    It checks if the pulseWidthCounter has reached the pulseWidth value of each motor
774  ;    and turns the motors on or off accordingly. Once done with all motors, it checks the
775  ;    laserStatus and turns the laser on or off accordingly. This function will be called
776  ;    by Timer1 at every PORTB_TIMER_MS miliseconds to enable PWM control.
777  ;    The period of pulses is PERIOD_PWM_MOTORS ms and it has PRECISION_RATE_PWM bits of
778  ;    precision.
779  ; Operation:
780  ;    First it clears ParallelBOutput(The value to be output to parallel port B) to
781  ;    DEFAULT_PORTB_OUTPUT
782  ;   DEFAULT_PORTB_OUTPUT bit map:
783  ;    -------0: Motor1 Direction Forward
784  ;    ------0-: Motor1 Turned Off
785  ;    -----0--: Motor2 Direction Forward
786  ;    ----0---: Motor2 Turned Off
787  ;    ---0----: Motor3 Direction Forward
788  ;    --0-----; Motor3 Turned Off
789  ;    0-------: Laser Turned Off
790  ;    Next, it checks if the pulseWidth of current motor is negative. If it is negative,
791  ;    save the absolute value of the negative pulseWidth in a register(AL) and set the
792  ;    direction bit of current motor to 1 to activate backward direction. If it is
     positive,
793  ;    we don't need to change the direction bit since the default direction bit of
794  ;    ParallelBOutput was MOTOR_DIRECTION_FORWARD.
795  ;    Next, check if the pulseWidthCounter has reached the pulseWidth of current motor.
796  ;    If it has not reached the pulseWidth yet, set the power bit of current motor.
797  ;    If it has already reached the pulseWidth of current motor, we don't need to change
     the
798  ;    power bit since the default value of power bit of ParallelBOutput is
     MOTOR_TURN_OFF.
799  ;    Repeat this procedure for all motors.
800  ;    Once the procedure has gone through all motors, the function updates
     pulseWidthCounter.
801  ;    We use the equation "New pulseWidthCounter = (pulseWidthCounter+1) mod
     COUNTER_MAX" for
802  ;    wrapping.
803  ;    Finally, it sets the laser bit if laserStatus is not LASER_OFF. It resets the
     laser bit
804  ;    otherwise.
805  ; Arguments:
806  ;   None
807  ; Return Value:
808  ;   None
809  ; Local Variables:
810  ;   ParallelBOutput - BL - The value to be output to parallel port B.
811  ;    MotorIndex     - DI - The index of current motor
```

```
812 ; Shared Variables:
813 ;      driveSpeed(DS)    - [Read]  - The speed at which the RoboTrike is to move
814 ;      driveAngle(DS)    - [Read]  - The angle at which the RoboTrike is to move in
    degrees
815 ;      pulseWidths(DS)   - [Read]  - An array which contains the pulse widths for
816 ;                                    NUM_MOTORS motors.
817 ;      pulseWidthCounter(DS)  - [Read/Write] - A counter used for PWM control.
818 ;      laserStatus(DS)        - [Read] - Indicates whether the laser is on or off.
819 ; Global Variables:
820 ;    None
821 ; Input:
822 ;    None
823 ; Output:
824 ;    Parallel portB: Motors and laser.
825 ; Error Handling:
826 ;    None
827 ; Algorithms:
828 ;    None
829 ; Data Structures:
830 ;    None
831 ; Registers Changed:
832 ;    AX, BX, DI, Flags
833 ; Limitations:
834 ;    None
835 ; Known bugs:
836 ;    None
837 ; Special Notes:
838 ;    None
839 ; Author:
840 ;    Sunghoon Choi
841 ; Revision History:
842 ;     11/6/2016    Sunghoon Choi   Created
843 ;     11/8/2016    Sunghoon Choi   Initial Compilation
844 ;     11/11/2016   Sunghoon Choi   Updated documentation
845 MotorLaserEventHandler  PROC    NEAR
846                         PUBLIC  MotorLaserEventHandler
847
848 InitMotorIndex:
849     XOR DI, DI          ;Initializing the motor index.
850                         ;The eventhandler handles motor1 first.
851
852 InitParallelValues:
853     XOR BX, BX          ;Initialize ParallelBOutput(BL) to DEFAULT_PORTB_OUTPUT
854                         ;DEFAULT_PORTB_OUTPUT bit map
855                         ;-------0: Motor1 Direction Forward
856                         ;------0-: Motor1 Turned Off
857                         ;-----0--: Motor2 Direction Forward
858                         ;----0---: Motor2 Turned Off
859                         ;---0----: Motor3 Direction Forward
860                         ;--0-----; Motor3 Turned Off
861                         ;0-------: Laser Turned Off
862
863 CheckNegPulse:
864     MOV AL, pulseWidths[DI]     ;First, we need to check if the pulseWidth is negative
865                                 ;to determine the direction.
866     CMP AL, 0                   ;Is the pulse negative?
867     JGE  CheckPCounter          ;No, it's positive.
868                                 ;Skip reversing the direction and go check if the counter
869                                 ;has reached pulseWidth.
870     ;JGE SetBackDir             ;Yes, it's negative. Change the direction to backward.
871 SetBackDir:
872     NEG AL                      ;Get the absolute value of the negative pulseWidth
873     OR  BL, CS:BackDirMask[DI]  ;Reverse the direction of current motor.
874     JMP CheckPCounter           ;Now that we are done with handling direction,
875                                 ;proceed to check the counter.
876 CheckPCounter:
```

```
877         CMP pulseWidthCounter, AL   ;Has the counter reached the pulse width of current
            motor?
878         JGE HandleNextMotor         ;Yes. Halt the current motor and check the next motor.
879     ;   JL  TurnCurrMotorOn         ;No. Continue activating the current motor.
880
881     TurnCurrMotorOn:
882         OR BL, CS:TurnOnMask[DI]    ;Turn the current motor on.
883     HandleNextMotor:
884         INC DI                      ;Increment index to handle the next motor
885         CMP DI, NUM_MOTORS          ;Was that the last motor?
886         JL CheckNegPulse            ;No. Go back to the beinning of loop and
887                                     ;check the pulseWidth of next motor.
888
889     ;   JGE UpdatePCounter          ;Yes. We went through all NUM_MOTORS motors.
890                                     ;Now update the pulse width counter.
891     UpdatePCounter:
892         INC pulseWidthCounter       ;Increment the pulseWidthCounter since we handled all
893                                     ;motors once.
894         XOR AX, AX                  ;Clear AH for DIV instruction.
895         MOV AL, pulseWidthCounter   ;Dividend = pulseWidthCounter+1
896         XOR DX, DX                  ;Clear DX for DIV instruction.
897         MOV CX, COUNTER_MAX         ;Divisor = COUNTER_MAX
898         DIV CX                      ;New pulseWidthCounter(DL)
899                                     ; = (pulseWidthCounter + 1) mod COUNTER_MAX.
900                                     ; Modulus of COUNTER_MAX is done for wrapping.
901         MOV pulseWidthCounter, DL   ;Update pulse counter with its new value.
902
903     CheckLaser:
904         CMP laserStatus, LASER_OFF  ;Is the laser setting LASER_OFF??
905         JE  OutputParallel          ;Yes, turn laser off
906     ;   JNE TurnLaserOn             ;No, turn laser on
907
908     TurnLaserOn:
909         OR BL, LASER_POWER_MASK     ;Set the laser bit of ParallelBOutput.
910
911     OutputParallel:
912         MOV DX, PARALLEL_B_ADDR    ;Get the address of parallel port B
913         MOV AL, BL                 ;Get the final value of ParallelBOutput.
914         OUT DX, AL                 ;Output the ParallelBOutput to parallel port B.
915
916
917     EndMotorLaserEventHandler:
918         RET                        ;End of MotorLaserEventHandler
919     MotorLaserEventHandler ENDP
920
921
922     CODE ENDS
923
924
925
926
927
928     DATA SEGMENT PUBLIC 'DATA'
929         pulseWidthCounter       DB  ?  ;A counter used for Pulse Width Modulation control
930         pulseWidths             DB  NUM_MOTORS DUP (?) ;An array which contains the
931                                                        ;pulse widths for NUM_MOTORS motors.
932         driveSpeed              DW  ?  ;The speed at which the RoboTrike is to move
933         driveAngle              DW  ?  ;The angle at which the RoboTrike is to move in
            degrees
934         laserStatus             DW  ?  ;Indicates whether the laser is on or off.
935     DATA ENDS
936
937
938
939     END
940
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                               ;
;                                Motors.inc                                     ;
;                                Homework 6                                     ;
;                               Sunghoon Choi                                   ;
;                                                                               ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Description:
;  This file contains the definitions for the Motors functions for RoboTrike.
;
; Revision History:
;    11/06/2016 Sunghoon Choi     Created
;    11/24/2016 Sunghoon Choi     Added MAX_SPEED, MIN_SPEED, and MAX_TURRET_ELEVATION.
;    11/26/2016 Sunghoon Choi     Changed the constant name MIN_ANGLE TO STRAIGHT_ANGLE.

INIT_PULSE_WIDTH    EQU    0       ;Initial pulse width
INIT_PULSE_COUNTER  EQU 0          ;Initial pulse width counter
INIT_SPEED          EQU 0          ;Initial speed
INIT_ANGLE          EQU 0          ;Initial angle
INIT_LASER_STAT     EQU 0          ;Initial laser status
LASER_ON            EQU 1          ;laserStatus value to turn laser on
LASER_OFF           EQU 0          ;laserStatus value to turn laser off


NUM_MOTORS          EQU 3          ;Total number of DC motors for omniwheels on RoboTrike.
LASER_POWER_MASK    EQU 10000000B ;Mask to turn the laser on.
PARALLEL_B_ADDR     EQU 181H       ;The address of parallel port B
COUNTER_MAX         EQU 128        ;Maximum value of the pulse width counter.

IGNORE_SPEED        EQU 65535      ;The speed when RoboTrike won't change its current
speed
IGNORE_ANGLE        EQU -32768     ;The angle when RoboTrike won't change its current
angle

FULL_ANGLE          EQU 360        ;Full angle. The angle will be normalized to
                                   ;[MIN_ANGLE, FULL_ANGLE-1]
STRAIGHT_ANGLE      EQU 0          ;The angle of moving straightforward.

NUM_SHIFT_ERASE_SIGN  EQU 1        ;Number of bit shift required to erase a sign value.
NUM_SHIFT_DOUBLE      EQU 1        ;Number of bit shift required to double a value.
NUM_SHIFT_HALF        EQU 1        ;Number of bit shift required to halve a value.
DUPSIGN_0DOT15_MUL2   EQU    2     ;Number of duplicated extra sign bits generated by
                                   ;multiplying three Q0.15 values. Since we have
                                   ;three sign bits, we must remove two extra sign bits
                                   ;from them.
MAX_SPEED              EQU 65534   ;Maximum speed of RoboTrike
MIN_SPEED              EQU 0       ;Minimu speed of RoboTrike

POS_TURRET_ELEV_BOUND  EQU 60      ;The positive bound of turret elevation angle
NEG_TURRET_ELEV_BOUND  EQU -60     ;The negative bound of turret elevation angle
```

```
NAME Serial

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                            ;
;                                   Serial                                   ;
;                                 Homework 7                                 ;
;                               Sunghoon Choi                                ;
;                                                                            ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; Description:
;    This file contains the functions necessary to initialize and enable serial communication
;    of the RoboTrike.
; Table of Contents:
;    InitSerial         -   Initializes shared variables related to Serial routine
;                           and initializes settings of serial transmission.
;    InitINT2           -   Initializes INT2 interrupt.
;    SetBaudRate        -   Set the baud rate of serial communication.
;    SetParity          -   Set the parity of serial communication.
;    SerialPutChar      -   Stores the passed character in TxQueue.
;   SerialEventHandler  -   Identify the interrupt occured at serial channel and execute
;                           appropriate actions for each type of interrupt.
;                           It is called by INT2 interrupt.
;    SerialPutString    -   Stores the passed string in TxQueue.
; Revision History:
;     11/16/2016      Sunghoon Choi     Created
;     11/16/2016      Sunghoon Choi     Initial Compilation
;     11/16/2016      Sunghoon Choi     Corrected LCR configuration error
;     11/18/2016      Sunghoon Choi     Added SetParity function
;     12/02/2016      Sunghoon Choi     Added SerialPutString function

CGROUP GROUP CODE
DGROUP GROUP DATA

$INCLUDE(Serial.inc)     ;Include the .inc file which contains constatns for Serial.asm
$INCLUDE(Events.inc)     ;Include the .inc file which contains the list of events for RoboTrike
$INCLUDE(Queue.inc)      ;Include the .inc file which contains constants for Queue.asm
$INCLUDE(general.inc)    ;Include the .inc file which contains general constants for RoboTrike

EXTRN QueueInit:NEAR     ;Import QueueInit to initialize TxQueue.
EXTRN Enqueue:NEAR       ;Import Enqueue to enqueue characters to TxQueue.
EXTRN Dequeue:NEAR       ;Import Dequeue to dequeue characters from Dequeue.
EXTRN QueueFull:NEAR     ;Import QueueFull to check if TxQueue is full.
EXTRN QueueEmpty:NEAR    ;Import QueueEmpty to check if TxQueue is empty.
EXTRN EnqueueEvent:NEAR  ;Import EnqueueEvent to enqueue serial events to EventBuf.


CODE SEGMENT PUBLIC 'CODE'

        ASSUME  CS:CGROUP, DS:DGROUP

; BaudRateTable
;
; Description:
;        This is the table of divisors to enable desired baud rates of serial communication
;        for RoboTrike.
; Notes:
;        This table is declared PRIVATE to prevent other codes accessing the table.
;        Also, READ ONLY tables should always be in the code segment so that in a standalone
;        system it will be located in the ROM with the code.
;
```

```
62  ; Author:             Sunghoon Choi
63  ; Revision history:   11/16/2016    Sunghoon Choi    Created
64  ;                     11/18/2016    Sunghoon Choi    Updated documentation
65
66  BaudRateTable   LABEL   WORD
67  DW         120  ;divisor for 4800 baud rate.  9.216MHz/16/4800
68  DW          60  ;divisor for 9600 baud rate.  9.216MHz/16/9600
69  DW          40  ;divisor for 14400 baud rate. 9.216MHz/16/14400
70  DW          30  ;divisor for 19200 baud rate. 9.216MHz/16/19200
71  DW          15  ;divisor for 38400 baud rate. 9.216MHz/16/38400
72
73
74
75  ; ParityTypeTable
76  ;
77  ; Description:
78  ;        This is the jump table for setting parity.
79  ; Notes:
80  ;        This table is declared PRIVATE to prevent other codes accessing the table.
81  ;        Also, READ ONLY tables should always be in the code segment so that in a
    standalone
82  ;        system it will be located in the ROM with the code.
83  ;
84  ; Author:           Sunghoon Choi
85  ; Revision history:   11/16/2016    Sunghoon Choi    Created
86  ;                     11/18/2016    Sunghoon Choi    Updated documentation
87  ParityTypeTable LABEL   WORD
88  DW  DisableParity          ;Jump to disable parity
89  DW  EnableEvenParity       ;Jump to enable even parity
90  DW  EnableOddParity        ;Jump to enable odd parity
91  DW  TransmitParityAndClear ;Jump to transmit parity and check as cleared.(Stick Parity)
92  DW  TransmitParityAndSet   ;Jump to transmite parity and check as set.(Stick Parity)
93  DW  EnableBreak            ;Jump to force a break condition.(Break control)
94
95
96
97  ; SerialINTTypeTable
98  ;
99  ; Description:
100 ;        This is the jump table used for executing appropriate actions for each type of
101 ;        serial interrupts.
102 ; Notes:
103 ;        This table is declared PRIVATE to prevent other codes accessing the table.
104 ;        Also, READ ONLY tables should always be in the code segment so that in a
    standalone
105 ;        system it will be located in the ROM with the code.
106 ;
107 ; Author:           Sunghoon Choi
108 ; Revision history:   11/16/2016    Sunghoon Choi    Created
109 ;                     11/18/2016    Sunghoon Choi    Updated documentation
110
111 SerialINTTypeTable  LABEL   WORD
112 DW  ModemStatusInterrupt           ;Jump to handle the Modem Status Interrupt
113 DW  TransmitterEmptyInterrupt      ;Jump to handle Transmitter Empty Interrupt
114 DW  ReceivedDataAvailInterrupt     ;Jump to handle Received Data Available Interrupt
115 DW  ReceiverLineStatusInterrupt    ;Jump to handle Reciever Line Status Interrupt
116
117
118
119
120 ; InitSerial
121 ;
122 ; Description:
123 ;    Initializes the TxQueue, kickstart, and registers of the serial chip.
124 ;    Initialization of serial chip's registers includes baud rate setting and parity
    setting.
```

```
;    Installs interrupt vector and enable the interrupt for serial I/O and send
SerialEOI.
; Operation:
;    First, it initializes the TxQueue by calling QueueInit.  When this is
;    done, it sets LCR register's value and enables the serial hardware interrupt by
;    setting the bits of IER register. Next, it calls SetBaudRate to
;    set the baud rate of serial I/O and calls SetParity to set the parity.
;   SetBaudRate and SetParity do not change any bits other than baud bits and
;   parity bits since they use AND, OR instructions with masks to prevent other
;   bits being changed. Thus, the initialized values for LCR and IER are safe.
;   Finally, it installs SerialEventHandler in INT2 vector and enables INT2 by
;   calling InitINT2 function.
; Arguments:
;     None
; Return Value:
;     None
; Local Variables:
;     TxQueueAddr(SI) -  The address of TxQueue
; Shared Variables:
;      TxQueue(DS)    -  [Write] - The queue which contains the characters to be
transmitted
;                                  to serial channel.
;      kickstart(DS)  -  [Write] - Indicates whether kickstart is needed to reactivate
;                                  THRE interrupt to conitnue data transmission.
; Global Variables:
;     None
; Input:
;     None
; Output:
;     LCR(Line Control Register), IER(Interrupt Enable Register),
;      EOI(End of Interrupt) Register
; Error Handling:
;     None
; Algorithms:
;     None
; Data Structures:
;     Queue(TxQueue)
; Registers Changed:
;     AX, BX, DX, SI, Flags
; Limitations:
;     None
; Known bugs:
;     None
; Special Notes:
;     None
; Author:
;     Sunghoon Choi
; Revision History:
;     11/16/2016    Sunghoon Choi      Created
;     11/16/2016    Sunghoon Choi      Initial Compilation
;     11/18/2016    Sunghoon Choi      Updated documentation

InitSerial  PROC    NEAR
            PUBLIC  InitSerial

InitTxQueue:
    MOV SI, OFFSET(TxQueue) ;We have to initialize TxQueue before using it as a
                            ;transmission queue which contains characters to be
                            ;transmitted. Thus, get the address of it.
    MOV BL, FALSE           ;Initialize TxQueue to a byte-type queue.
                            ;BL is an argument for QueueInit. If it is FALSE, TxQueue
                            ;gets initialized to a byte queue. Otherwise, it will get
                            ;intialized to a word queue.
    PUSHA                   ;Save all register values since QueueInit function changes
                            ;the values of registers.
    CALL QueueInit          ;Initialize TxQueue by using QueueInit function.
```

```
189     POPA                    ;Retrieve all register values since we are back from
        QueueInit
190
191 InitLineControlRegister:
192     MOV DX, LCR_ADDR        ;We have to control the formate of asynchronous data
193                             ;communication exchange through the LCR.
194                             ;Thus, get the address of LCR register.
195     MOV AL, LCR_VAL         ;Prepare the value to be written to LCR register.
196                             ;0------- DLAB bit off. Access RBR, THR
197                             ;-0------ Break condition disbaled
198                             ;--0----- No Parity
199                             ;---0---- No Parity
200                             ;----0--- No Parity
201                             ;-----0-- One Stop bit
202                             ;------11 8 Bits Word length
203     OUT DX, AL              ;Set up the LCR by writing the prepared configuration value
204                             ;to LCR register's address.
205
206 InitINTEnableRegister:
207     MOV DX, IER_ADDR        ;We have to enable the interrupts of serial communication.
208                             ;Thus, get the address of IER register.
209     MOV AL, IER_VAL         ;Prepare the value to be written to IER register.
210                             ;0000---- Bits4-7 of IER are ALWAYS cleared
211                             ;----1--- Modem Status Interrupt Enabled
212                             ;-----1-- Receiver Line Status Interrupt Enabled
213                             ;------1- THRE interrupt enabled
214                             ;-------1 Received Data Availalbe Interrupt enabled
215     OUT DX, AL              ;Set up the IER by writing the prepared configuration value
216                             ;to IER reigster's address.
217 InitBaudRate:
218     MOV BX, BAUD_RATE_INDEX ;Choose a desired baud rate so that SetBaudRate function can
219                             ;get a proper divisor value from BaudRateTable and set
220                             ;the proper baud rate.
221     CALL SetBaudRate        ;Sets baud rate of serial communication to the desired value
222                             ;Note that SetBaudRate does not change any bits except
223                             ;those bits related to baud rate.
224
225 InitParity:
226     MOV BX, DISABLE_PARITY  ;We disable parity for now.
227                             ;Thus, give DISABLE_PARITY as an argument of SetParity.
228     CALL SetParity          ;Disables parity by calling SetParity with DISABLE_PARITY
229                             ;as an argument.
230                             ;Note that SetParity does not change any bits except
231                             ;those bits related to parity setting.
232 InitINT2Interrupt:
233     CALL InitINT2           ;Installs SerialEventHandler in INT2 vector and
234                             ;enables INT2 interrupt by calling InitINT2.
235 InitKickstart:
236     MOV kickstart, KICKSTART_OFF  ;Resets the kickstart flag for intialization.
237
238 EndInitSerial:
239     RET
240 InitSerial ENDP
241
242
243
244 ; InitINT2
245 ;
246 ; Description:
247 ;   This function initializes INT2 interrupt.
248 ; Operation:
249 ;   Installs SerialEventHandler in INT2 vector.
250 ;   Then, Interrupt Control Register of INT2 is initialized to enable INT2
251 ;   interrupt with INT2_PRIORITY_LEVEL priority.
252 ; Arguments:
253 ;   None
```

```
; Return Value:
;   None
; Local Variables:
;   None
; Shared Variables:
;   None
; Global Variables:
;   None
; Input:
;   None
; Output:
;   INT2 and the Interrupt Controller are initialized.
; Error Handling:
;   None
; Algorithms:
;   None
; Data Structures:
;   None
; Registers Changed:
;   AX, DX, Flags
; Limitations:
;   None
; Known bugs:
;   None
; Special Notes:
;   None
; Author:
;   Sunghoon Choi
; Revision History:
;     11/16/2016    Sunghoon Choi      Created
;     11/16/2016    Sunghoon Choi      Initial Compilation
;     11/18/2016    Sunghoon Choi      Updated documentation


InitINT2    PROC    NEAR
            PUBLIC  InitINT2

InstallINT2:

    XOR     AX, AX          ;clear ES
    MOV     ES, AX          ;(interrupt vectors are in segment 0
                            ;store the INT2 vector
    MOV     ES: WORD PTR (4 * Int2Vec), OFFSET(SerialEventHandler)
    MOV     ES: WORD PTR (4 * Int2Vec + 2), SEG(SerialEventHandler)

EnableINT2:
    MOV DX, INT2_ICR_ADDR   ;We have to enable INT2 interrupt and set its priority level.
    MOV AL, INT2_ICR_VAL
    OUT DX, AL              ;Write value to I2CON Register to set up the INT2 interrupt.

EndInitINT2:
    RET
InitINT2  ENDP




; SetBaudRate
;
; Description:
;    Sets the baud rate of Serial communication for RoboTrike.
; Operation:
;    Before the procedure starts, it saves the value stored in LCR register.
;    Next, it sets the DLAB bit of LCR register to enable access to Divisor Latches of
the
```

```
;    Baud Generator. Then, it writes the desired baud rate divisor to Divisor Latch by
;    refering to the baud rate table. When setting baud rate is done, it retrieves the
;    original value of LCR register and write in LCR register.
; Arguments:
;   BaudRateIndex(BX)n - The index used to get the divisor value for desired baud rate
from
;                        BaudRateTable.
; Return Value:
;   None
; Local Variables:
;   LCRValue(AL) n     - The value to be written to LCR.
;    BaudRateIndex(BX) - The index used to get the divisor value for desired baud rate
from
;                        BaudRateTable.
; Shared Variables:
;    None
; Global Variables:
;   None
; Input:
;   LCR(Line Control Register)
; Output:
;   LCR(Line Control Register)
; Error Handling:
;   None
; Algorithms:
;   None
; Data Structures:
;   None
; Registers Changed:
;   AX, BX, CX, DX, Flags
; Limitations:
;   None
; Known bugs:
;   None
; Special Notes:
;   None
; Author:
;   Sunghoon Choi
; Revision History:
;     11/16/2016    Sunghoon Choi      Created
;     11/16/2016    Sunghoon Choi      Initial Compilation
;     11/18/2016    Sunghoon Choi      Updated documentation

SetBaudRate   PROC    NEAR
              PUBLIC  SetBaudRate

BaudGetLCRVal:
    MOV DX, LCR_ADDR        ;We should save the original value of LCR register
    IN AL, DX               ;to prevent other bits being changed.
    MOV CL, AL              ;Save the value in CL so that we can use it when
EnableDLAB:
    PUSHF                   ;Critical code starts. Save flags.
    CLI                     ;Disable interrupts.
    OR AL, ENABLE_DLAB_MASK ;Do the OR instruction with a mask to set DLAB bit of LCR to
                            ;enable access to Divisor Latch.
                            ;1-------: Enables access to Divisor Latch.
    OUT DX, AL              ;Sets DLAB bit.
SetBaudVal:
    SHL BX, MULT_BY_2       ;Double the BaudRateIndex since BaudRateTable is a WORD
    table.
    MOV DX, DIV_LATCH_ADDR  ;Get the address of Divisor Latch to set baud rate.
    MOV AX, CS:BaudRateTable[BX]  ;Get the divisor value for desired baud rate.
    OUT DX, AL                    ;Writes the divisor value to Divisor Latch to set
    baud rate

RetrieveOriginalLCR:
```

```
381         MOV AL, CL                      ;Retrieve the original LCR value.
382         AND AL, NOT(ENABLE_DLAB_MASK)   ;Disable access to divisor latch by resetting DLAB bit.
383                                         ;DLAB bit must be cleared to access receiver buffer,
384                                         ;THR, or the IER.
385
386         MOV DX, LCR_ADDR                ;Get the address of LCR to set LCR value to its
            original
387                                         ;value.
388         OUT DX, AL                      ;Write the original LCR value to LCR.
389         POPF                            ;End of Critical code. Retrieve flags and enable
            interrupt
390
391     EndSetBaudRate:
392         RET                             ;End of SetBaudRate
393     SetBaudRate ENDP
394
395
396     ; SetParity
397     ;
398     ; Description:
399     ;   Sets the parity of Serial communication for RoboTrike.
400     ; Operation:
401     ;   Before the procedure starts, it saves the value of LCR register.
402     ;   Then, it uses a jump table(ParityTypeTable) to set the desired parity.
403     ;   It uses OR and AND instructions to keep other bits unchanged.
404     ;   Finally, it outputs the value to LCR and exits.
405     ; Arguments:
406     ;   parityIndex(BX) - Index for desired parity setting. It will be used with
407     ;                     ParityTypeTable to set the desired parity setting.
408     ; Return Value:
409     ;   None
410     ; Local Variables:
411     ;   LCRValue(AL)   - The value to be written to LCR.
412     ;   parity(BX)     - Desired parity setting. It will be used as an index for
        ParityTypeTable
413     ;                    to set the desired parity setting.
414     ; Shared Variables:
415     ;    None
416     ; Global Variables:
417     ;   None
418     ; Input:
419     ;   LCR(Line Control Register)
420     ; Output:
421     ;   LCR(Line Control Register)
422     ; Error Handling:
423     ;   None
424     ; Algorithms:
425     ;   None
426     ; Data Structures:
427     ;   None
428     ; Registers Changed:
429     ;   AX, BX, DX, Flags
430     ; Limitations:
431     ;   None
432     ; Known bugs:
433     ;   None
434     ; Special Notes:
435     ;   None
436     ; Author:
437     ;   Sunghoon Choi
438     ; Revision History:
439     ;     11/16/2016    Sunghoon Choi      Created
440     ;     11/16/2016    Sunghoon Choi      Initial Compilation
441     ;     11/18/2016    Sunghoon Choi      Updated documentation
442
443     SetParity   PROC    NEAR
```

```
444             PUBLIC  SetParity
445
446     PUSHF                           ;Critical code starts. Save all flags
447     CLI                             ;Disable Interrupt
448 SelectParityType:
449     MOV DX, LCR_ADDR                ;Save the original value of LCR register.
450     IN AL, DX                       ;We do not want to change bits not related to parity.
451     JMP CS:ParityTypeTable[BX]      ;Jump to the proper parity setting procedure.
452 DisableParity:
453     AND AL, NOT(ENABLE_PARITY_MASK) ;disable parity by clearing Parity Enable bit.
454     JMP EndSetParityTable           ;Go write the value in LCR.
455 EnableEvenParity:
456     OR AL, ENABLE_PARITY_MASK       ;Enable parity
457     OR AL, EVEN_PARITY_MASK         ;Enable even parity
458     JMP EndSetParityTable           ;Go write the value in LCR.
459 EnableOddParity:
460     OR AL, ENABLE_PARITY_MASK       ;Enable parity
461     AND AL, NOT(EVEN_PARITY_MASK)   ;Enable odd parity
462     JMP EndSetParityTable           ;Go write the value in LCR.
463 TransmitParityAndClear:
464     OR AL, TRANSMIT_CLR_MASK        ;Transmit parity bit and check as cleared.
465     JMP EndSetParityTable
466 TransmitParityAndSet:
467     OR AL, TRANSMIT_CLR_MASK        ;bit 3 and bit5 set and bit 4 clear is the condition
468     AND AL, TRANSMIT_SET_MASK       ;to transmit parity bit and check as set.
469     JMP EndSetParityTable
470 EnableBreak:
471     OR AL, BREAK_CTRL_MASK          ;Force a break condition
472     JMP EndSetParityTable
473 EndSetParityTable:
474     OUT DX, AL                      ;Write the configuration value in LCR.
475     POPF                            ;Critical code ends.
476     RET                             ;End of SetParity procedure.
477 SetParity ENDP
478
479
480
481
482 ; SerialPutChar
483 ;
484 ; Description:
485 ;    It outputs the passed character to the serial channel although what it actually does
486 ;    is putting the character in TxQueue. It returns with the carry flag reset if the
487 ;    character has been output(put in the TxQueue, not necessarily sent over the serial
488 ;    channel) and set otherwise (TxQueue is full). The character is passed by value in
    AL.
489 ; Operation:
490 ;    It checks if TxQueue is full by calling Queuefull. If TxQueue is full, set the
    carry
491 ;    flag and exit the procedure. If it is not full, the procedure enqueues TxQueue
    with the
492 ;    given character. Then, it checks if kickstart flag is set. If it is, disable THRE
493 ;    interrupt and enable THRE interrupt to reactivate THRE interrupt.
494 ;    If Transmitter Empty Interrupt happened but the system could not write
495 ;    the character to THR, the system cannot resolve the Transmitter Empty Interrupt.
    Thus,
496 ;    THRE interrupt will not be generated unless THRE interrupt gets kickstarted by
497 ;    "reset interrupt-and-set interrupt" procedure.
498 ;    When kickstarting THRE interrupt is done,  it resets the kickstart flag.
499 ;    If kickstart flag was not set, there's no extra procedure to be done.
500 ;    Finally, it resets the carry flag and exits.
501 ; Arguments:
502 ;   Character(AL)     - character to be passed to serial channel.
503 ; Return Value:
504 ;   Carry Flag        - Set if TxQueue is full.
505 ;                     - Reset if TxQueue is not full and character has been output
```

```
;                       (put in the TxQueue)
; Local Variables:
;    TxQueueAddr(SI)  -  The address of TxQueue
;    IERValue(AL)     -  The value of IER(Interrupt Enable Register)
; Shared Variables:
;    TxQueue(DS)       -  [R/W] - The queue which contains the characters to be
transmitted
;                                 to serial channel.
;    kickstart(DS)     -  [R/W] - Indicates whether kickstart is needed to reactivate
;                                 THRE interrupt to conitnue data transmission.
; Global Variables:
;   None
; Input:
;   None
; Output:
;   IER(Interrupt Enable Register)
; Error Handling:
;   None
; Algorithms:
;   None
; Data Structures:
;   None
; Registers Changed:
;   AX(If the character has been output), BX, DX, SI, Flags
; Limitations:
;   None
; Known bugs:
;   None
; Special Notes:
;   None
; Author:
;   Sunghoon Choi
; Revision History:
;     11/16/2016    Sunghoon Choi      Created
;     11/16/2016    Sunghoon Choi      Initial Compilation
;     11/18/2016    Sunghoon Choi      Updated documentation

SerialPutChar   PROC    NEAR
                PUBLIC  SerialPutChar

CheckTxQueueFull:
    PUSH AX                     ;Save the argument(character) since it should be enqueued
                                ;to TxQueue later.
    MOV SI, OFFSET(TxQueue)     ;Get the address of TxQueue to check if it is full.

    CALL QueueFull              ;Check if TxQueue is full by calling QueueFull.
    JNZ EnqueueTx               ;if TxQueue is not full, enqueue the character to TxQueue.
   ;JZ SetCarry                 ;If TxQueue is full, set the carry flag.
SetCarry:
    POP AX                      ;Retrieve the character.
    STC                         ;Set the carry flag as a return value.
    JMP EndSerialPutChar        ;Exit the procedure.
EnqueueTx:
    POP AX                      ;Retreive the character.
    PUSHA                       ;Save all registers since calling Enqueue changes
    registers' values.

    CALL Enqueue                ;Enqueue the character to TxQueue
    POPA                        ;Retreive all registers' values.
    CMP  kickstart, KICKSTART_ON  ;Is kickstart flag set?
    JNE  ResetCarry             ;If it is not set, we don't need to kickstart(reactivate)
                                ;THRE interrupt. Thus, go reset the carry flag and exit.

   ;JE   ReActivateTHRE         ;If it is set, we have to kickstart(reactivate) THRE
   interrupt.
                                ;If Transmitter Empty Interrupt happened but the system
```

```
                                could
569                             ;not write the character to THR, the system cannot
                                resolve the
570                             ;Transmitter Empty Interrupt. Thus, THRE interrupt will
                                not be
571                             ;generated unless THRE interrupt gets kickstarted by
572                             ;"reset interrupt-and-set interrupt" procedure.
573 ReActivateTHRE:
574     MOV DX, IER_ADDR            ;We should keep the original value of IER. So,
575                                 ;Get the address of IER to obtain its current value.
576     IN AL, DX                   ;Get original IER value from IER_ADDR.
577     AND AL, DISABLE_THRE_MASK   ;Disable THRE interrupt
578     OUT DX,AL
579
580     OR AL, NOT(DISABLE_THRE_MASK)   ;Enable THRE interrupt
581     PUSHF
582     CLI
583     OUT DX, AL
584 ClearKickstart:
585
586     MOV kickstart, KICKSTART_OFF    ;Reset kickstart flag since we
        kickstarted(reactivated)
587                                 ;THRE interrupt.
588     POPF
589 ResetCarry:
590     CLC                         ;Reset the carry flag as a return value.
591 EndSerialPutChar:
592     RET                         ;End of SerialPutChar procedure.
593 SerialPutChar ENDP
594
595
596
597
598
599
600 ; SerialEventHandler
601 ;
602 ; Description:
603 ;    Identify the interrupt occured at serial channel and execute appropriate actions for
604 ;    each type of interrupt. SerialEventHandler is called by INT2 interrupt.
605 ; Operation:
606 ;    It reads a value from IIR(Interrupt Identification Register) and refer to the
607 ;    SerialINTTypeTable. It will go to a label corresponding to the type of interrupt.
608 ;
609 ;    1. Receiver Line Status Interrupt:    A serial error has occured. Read the LSR
    register
610 ;                                          to reset the interrupt. Enqueue the event by
611 ;                                          calling EnqueueEvent with event type and
    event
612 ;                                          value.
613 ;
614 ;    2. Received Data Available Interrupt: Receiver data is available. Read the receiver
615 ;                                          buffer to reset the interrupt. Enqueue the
    event
616 ;                                          by calling EnqueueEvent with event type and
617 ;                                          event value.
618 ;
619 ;    3. Transmitter Empty Interrupt:       THR is empty. Check if TxQueue is empty. If
    it is,
620 ;                                          set the kickstart flag. If it is not empty,
621 ;                                          dequeue a value from TxQueue and write it in
    THR.
622 ;
623 ;    4. Modem Status Interrupt:            Read MSR(Modem Status Register)to reset the
624 ;                                          interrupt.
625 ;
```

```
626   ;    The program flow should not exit from SerialEventHandler until all serial
      interrupts
627   ;    are reset. When all serial interrupts are handled and reset, SerialEventHandler
      sends
628   ;    SerialEOI and exit.
629   ; Arguments:
630   ;   None
631   ; Return Value:
632   ;   None
633   ; Local Variables:
634   ;    InterruptIdentity(AL)
635   ; Shared Variables:
636   ;   TxQueue(DS)   -  [Read]  -  The queue which contains the characters to be transmitted
637   ;                               to serial channel.
638   ;   kickstart(DS) -  [Write] -  Indicates whether kickstart is needed to reactivate
639   ;                               THRE interrupt to conitnue data transmission.
640   ; Global Variables:
641   ;   None
642   ; Input:
643   ;   IIR(Interrupt Identification Register), LSR(Line Status Register)
644   ;   RBR(Receiver Buffer Register)
645   ; Output:
646   ;   THR(Transmitter Empty Register)
647   ; Error Handling:
648   ;   None
649   ; Algorithms:
650   ;   None
651   ; Data Structures:
652   ;   Queue(TxQueue)
653   ; Registers Changed:
654   ;   AX, BX, DX, SI, Flags
655   ; Limitations:
656   ;   None
657   ; Known bugs:
658   ;   None
659   ; Special Notes:
660   ;   None
661   ; Author:
662   ;   Sunghoon Choi
663   ; Revision History:
664   ;     11/16/2016    Sunghoon Choi      Created
665   ;     11/16/2016    Sunghoon Choi      Initial Compilation
666   ;     11/18/2016    Sunghoon Choi      Updated documentation
667
668
669   SerialEventHandler  PROC    NEAR
670                       PUBLIC  SerialEventHandler
671
672
673       PUSHA                   ;Save all register values since it's an interrupt event
          handler
674   InitSerialEventHandler:
675
676       XOR AX, AX              ;Clear AX since we will move AX's value to BX and use it as
          an
677                               ;index for SerialINTTypeTable.
678
679       MOV DX, IIR_ADDR        ;Read a value from IIR(Interrupt Identification Register) to
680       IN AL, DX                ;identify current serial interrupt.
681                               ;-----001: No Interrupt
682                               ;-----110: Priority 1, Receiver line status
                                  interrupt
683                               ;-----100: Priority 2, Received data available interrupt
684                               ;-----010: Priority 3, Transmitter holding register empty
685                               ;                       interrupt
686                               ;-----000: Priority 4, Modem status interrupt
```

```
687     CMP AL, NO_INTERRUPT    ;Was there no interrupt?
688     JE SendSerialEOI        ;There was no interrupt. Go send SerialEOI to announce the
        end
689                             ;of SerialEventHandler
690     ;JNE JumpINTTypeHandler ;There was an interrupt. Go handle each type of interrupt.
691
692 JumpINTTypeHandler:
693     MOV BX, AX                  ;Move interrupt type into BX to use it as an index for
694                                 ;jump table.
695     JMP CS:SerialINTTypeTable[BX] ;Go to the corresponding interrupt handling routine of
696                                 ;current interrupt.
697
698 ReceiverLineStatusInterrupt:    ;Interrupt: Overrun error, parity error, framing error
699                                 ;or break interrupt has occured.
700     MOV DX, LSR_ADDR            ;Get the address of LSR register and
701     IN AL, DX                   ;read the value of LSR to identify the error type.
702
703     AND AL, ERROR_MASK          ;Extract the error type from LCR value by using
        ERROR_MASK
704     MOV AH, SERIAL_ERROR_EVENT  ;Save the event type(SERIAL_ERROR_EVENT) in AH
705     CALL EnqueueEvent           ;Enqueue the event value(error type) and
706                                 ;event type(error event) to EventBuf.
707     JMP InitSerialEventHandler  ;Since we reset Receiver Line Status Interrupt,
708                                 ;go back and check if there's any other interrupts.
709
710 ReceivedDataAvailInterrupt:     ;Interrupt: Receiver data is available.
711     MOV DX, RBR_ADDR            ;Read the receiver buffer register to reset
712     IN AL, DX                   ;the Received Data Available Interrupt.
713     MOV AH, SERIAL_RECEIVED_EVENT ;AL = Read data(Event value)
714                                 ;AH = SERIAL_RECEIVED_EVENT(Event type)
715                                 ;These two arguments(event value and event type) will
                                    be
716                                 ;used to enqueue the event to EventBuf.
717     CALL EnqueueEvent           ;Enqueue the event value and event type to EventBuf
718     JMP InitSerialEventHandler  ;Since we reset Received Data Available Interrupt,
719                                 ;go back and check if there's any other interrupts.
720
721 TransmitterEmptyInterrupt:      ;Interrupt: Transmitter holding register is empty.
722     MOV SI, OFFSET(TxQueue)     ;Get the address of TxQueue to dequeue a value from it
723                                 ;and fill it in THR(Transmitter Holding Register)
724
725     PUSHF                       ;Critical code starts. Save all flags.
726     CLI                         ;Disable interrupts
727
728     CALL QueueEmpty             ;Check if TxQueue is empty.
729     JNZ WriteTHR                ;If TxQueue is not empty, go take a value from it
730                                 ;and write it in THR.
731     ;JZ SetKickstart            ;If TxQueue is empty, we cannot write any value in THR.
732                                 ;So now, although Transmitter Empty Interrupt has
                                    occured,
733                                 ;no value can be written in THR. Thus, Transmitter
                                    Empty
734                                 ;Interrupt will not be generated unless we kickstart it
735                                 ;or reactivate it. Thus, go set the kickstart flag.
736 SetKickStart:
737     MOV kickstart, KICKSTART_ON ;Set the kickstart flag to reactivate THRE interrupt.
738
739     POPF                        ;Critical code ends. Retrieve all flags and enable
740                                 ;interrupts.
741
742     JMP InitSerialEventHandler  ;Since we reset the Transmitter Empty Interrupt,
743                                 ;go back and check if there's any other interrupts.
744
745 WriteTHR:
746     POPF                        ;Critical code ends. Retrieve all flags and enable
747                                 ;interrupts.
```

```
748         CALL Dequeue                    ;Dequeue a character from TxQueue to write it in THR.
749         MOV DX, THR_ADDR                ;Get the address of THR to write a character.
750         OUT DX, AL                      ;Write the dequeued character in THR.
751         JMP InitSerialEventHandler      ;Since we reset the Transmitter Empty Interrupt,
752                                         ;go back and check if there's any other interrupts.
753
754     ModemStatusInterrupt:               ;Interrupt: Modem Status Interrupt
755         MOV DX, MSR_ADDR                ;Read modem status register to reset the
756         IN  AL, DX                      ;Modem Status Interrupt.
757         JMP InitSerialEventHandler      ;Since we reset the Modem  Interrupt,
758                                         ;go back and check if there's any other
                                            interrupts.
759
760     SendSerialEOI:
761        MOV DX, INTCtrlrEOI              ;Send SerialEOI to announce the end of
           SerialEventHandler
762        MOV AL, SerialEOI
763        OUT DX, AL
764
765     EndSerialEventHandler:
766        POPA                             ;Retrieve all original register values
767        IRET                             ;End of SerialEventHandler
768
769     SerialEventHandler ENDP
770
771
772     ; SerialPutString
773     ;
774     ; Description:
775     ;    It sends a string to the serial channel although it actually does is storing a
        string in
776     ;    TxQueue by repeatedly calling SerialPutChar.
777     ; Operation:
778     ;    It gets passed the address of the string to be sent(through the serial channel) in
        SI.
779     ;    It checks every character that belongs to the string.
780     ;    First, it checks if the character is NULL. If it is NULL, it sends Carriage Return
        by
781     ;    calling SerialPutChar after setting AL to CarriageReturn. If it is not NULL, send
        the
782     ;    character by calling SerialPutChar. Then, it checks if TxQueue is full by checking
        the
783     ;    carry flag which is returned from SerialPutChar.  If the carry flag is set, exit
784     ;    SerialPutString. If the carry flag is not set, increment the index of the
        character(in
785     ;    the string) and go back to fetch the next character.
786     ; Arguments:
787     ;   SI(StringAddress)    -    The address of the string to be sent.
788     ; Return Value:
789     ;   None
790     ; Local Variables:
791     ;    Character(AL)       -    The current character to be enqueued to TxQueue.
792     ;    CharAddrees(SI)     -    The address of the character to be sent.
793     ;                             It starts from the starting address of the string but gets
794     ;                             incremented every loop to handle all characters.
795     ; Shared Variables:
796     ;
797     ; Global Variables:
798     ;   None
799     ; Input:
800     ;   None
801     ; Output:
802     ;   None
803     ; Error Handling:
804     ;   None
805     ; Algorithms:
```

```
;   None
; Data Structures:
;   None
; Registers Changed:
;   AX, SI, Flags
; Limitations:
;   None
; Known bugs:
;   None
; Special Notes:
;   None
; Author:
;   Sunghoon Choi
; Revision History:
;     12/01/2016    Sunghoon Choi      Created
;     12/02/2016    Sunghoon Choi      Initial Compilation
;     12/02/2016    Sunghoon Choi      Updated documentation


SerialPutString     PROC     NEAR
                    PUBLIC  SerialPutString


FetchCharacter:
    MOV AL, ES:[SI]          ;fetch a character from the source string address.
    CMP AL, 0                ;Is the character NULL?
    JNE SendChar             ;No, send the character
    ;JE SendCarriageReturn   ;Yes, send carriage return
SendCarriageReturn:
    MOV AL, 13               ;Let's send Carriage Return. Insert CR in AL.
    PUSH SI                  ;Save the address of the current character.
    CALL SerialPutChar       ;Call SerialPutChar to store Carriage Return in TxQueue.
    POP SI                   ;Retrieve the address of the current character to conitnue
                             ;fetching characters.
    JMP EndSerialPutString   ;Since we've sent Carriage Return, SerialPutString ends.
SendChar:
    PUSH SI                  ;Save the address of the current character.
    CALL SerialPutChar       ;Call SerialPutChar to store the character in TxQueue.
    POP SI                   ;Retrieve the address of the current character to continue
                             ;fetcing characters.
FindTxQueueFull:
    JC  EndSerialPutString   ;Check if TxQueue is full. If it is, exit the process.
   ;JNC SendNextChar         ;Since TxQueue is not full, proceed to send the next
   character.
SendNextChar:
    INC SI                   ;increment the index to fetch the next character in the
    string.
    JMP FetchCharacter       ;repeat fetching characters until it gets a Carriage Return
                             ;or the TxQueue is full.
EndSerialPutString:
    RET                      ;End of SerialPutString
SerialPutString ENDP



CODE ENDS




DATA SEGMENT PUBLIC 'DATA'
    TxQueue     QueueModule     <>  ;The queue which contains the characters to be
                                    ;transmitted to serial channel.
    kickstart   DB              ?   ;Indicates whether kickstart is needed to reactivate
                                    ;THRE interrupt to conitnue data transmission.
```

```
870   DATA ENDS
871
872   END
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                               ;
;                                   Serial.inc                                  ;
;                                   Homework 7                                  ;
;                                  Sunghoon Choi                                ;
;                                                                               ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Description:
;  This file contains the definitions for the Serial.asm
;
; Revision History:
;    11/16/2016    Sunghoon Choi    Created
;    11/18/2016    Sunghoon Choi    Documentation Updated


;Serial Definitions

;Addresses
SERIAL_BASE EQU     100H                    ;Base address of Serial

LCR_ADDR    EQU     SERIAL_BASE+03H         ;Address of Line Control Register
IER_ADDR    EQU     SERIAL_BASE+01H         ;Address of Interrupt Enable Register
IIR_ADDR    EQU     SERIAL_BASE+02H         ;Address of Interrupt Identification Register
LSR_ADDR    EQU     SERIAL_BASE+05H         ;Address of Line Status Register
THR_ADDR    EQU     SERIAL_BASE+00H         ;Address of Transmitter Holding Register
RBR_ADDR    EQU     SERIAL_BASE+00H         ;Address of Receiver Buffer Register
MSR_ADDR    EQU     SERIAL_BASE+06H         ;Address of Modem Status Register
DIV_LATCH_ADDR  EQU SERIAL_BASE+00H         ;Address of Divisor Latch


;Values
NO_INTERRUPT    EQU 00000001B   ;No Interrupt indicated by IIR

LCR_VAL     EQU     00000011B   ;The value to be written to Line Control Register.
                   ;0------- DLAB bit off. Access RBR, THR
                   ;-0------ Break condition disbaled
                   ;--0----- Parity Off
                   ;---0---- Parity Off
                   ;----0--- Parity Off
                   ;-----0-- One Stop bit
                   ;------11 8 Bits Word length

IER_VAL     EQU     00001111B   ;The value to be written to Interrupt Enable Register.
                   ;0000---- Bits4-7 of IER are ALWAYS cleared
                   ;----1--- Modem Status Interrupt Enabled
                   ;-----1-- Receiver Line Status Interrupt Enabled
                   ;------1- THRE interrupt enabled
                   ;-------1 Received Data Availalbe Interrupt enabled

INT2_ICR_VAL   EQU  0000000000000001B ;The value to be written to I2CON Register.
                   ;00000000000-----  These are reserved bits.
                   ;-----------0----  Edge trigger
                   ;------------0---  enables the INT2 Interrupt
                   ;-------------001  sets INT2 priority to 1.


DISABLE_PARITY      EQU 0           ;Argument of SetParity for disabling parity
EVEN_PARITY         EQU 2           ;Argument of SetParity for enabling even parity(not
used)
ODD_PARITY          EQU 4           ;Argument of SetParity for enabling odd parity(not
used)

KICKSTART_OFF       EQU 0           ;Turn off kickstart flag
KICKSTART_ON        EQU 1           ;Turn on kickstart flag

;Masks
```

```
ENABLE_DLAB_MASK    EQU 10000000B ;Mask used to set DLAB bit

DISABLE_THRE_MASK   EQU 11111101B ;Mask used to disable THRE interrupt

ERROR_MASK          EQU 00011110B ;Mask used to extract error

ENABLE_PARITY_MASK  EQU 00001000B ;Mask used to enable parity bit.
EVEN_PARITY_MASK    EQU 00010000B ;Mask used to enable even parity
TRANSMIT_CLR_MASK   EQU 00111000B ;Mask used to transmit the parity and check as cleared.
TRANSMIT_SET_MASK   EQU 11101111B ;Mask used to transmit the parity and check as set.
                                  ;Must be used in pair with TRANSMIT_CLR_MASK.
BREAK_CTRL_MASK     EQU 01000000B ;Mask used to force a break condition.

;Table Indices
BAUD_RATE_INDEX     EQU 1         ;Index used to obtain proper divisor value for
                                  ;the desired baud rate.

;Interrupt Definitions

;Vectors
Int2Vec             EQU 14        ;Interrupt vector for INT2

;Addresses

PERIPHERAL_BASE EQU     0FF00H                ;Base Address of PCB
INT2_ICR_ADDR   EQU     PERIPHERAL_BASE+3CH   ;Address of INT2 Interrupt Control
Register
INTCtrlrEOI     EQU     PERIPHERAL_BASE+22H   ;Address of End_of_Interrupt Register

;Values
SerialEOI       EQU 000EH                     ;EOI to clear the In-Service bit for INT2
```

```
NAME Parser

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                              ;
;                                   Parser                                     ;
;                                 Homework 8                                   ;
;                                Sunghoon Choi                                 ;
;                                                                              ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Description:
;    This file contains functions necessary for handling characters of serial input.
;    It executes various commands through finite state machine.
;
; Table of Contents:
;    ParseSerialChar        -    Get passed a character and process it as a command.
;    InitParser             -    Initializes shared variables related to parsing routine
;    ResetHandle            -    Reset the shared variables related to the arguments of parsing
;                                routine.
;    GetToken               -    Returns the token value and token type.
;    DoSetAbsMotorSpeed     -    Sets the absolute value of motor speeds.
;    DoSetRelMotorSpeed     -    Increases or decreases the motors speeds.
;    DoSetMotorDirection    -    Sets the direction of RoboTrike.
;    DoSetLaserOn           -    Turns the laser on.
;    DoSetLaserOff          -    Turns the laser off.
;    DoSetAbsTurretAngle    -    Sets the absolute value of the turret angle.
;    DoSetRelTurretAngle    -    Increases or decreases the angle of the turret.
;    DoSetTurretElevation   -    Sets the elevation of the turret.
;    DoAddDigit             -    Adds a digit to the shared argument value(ArgNum)
;    DoSetError             -    Returns PARSER_ERROR.
;    DoNOP                  -    Does nothing. Created for State Machine.
;    DoSetNegSign           -    Set the negative sign of shared argument value (ArgSign)

; Revision History:
;    11/23/2016      Sunghoon Choi     Created
;    11/25/2016      Sunghoon Choi     Initial Compilation
;    11/26/2016      Sunghoon Choi     Updated Documentation

$INCLUDE(general.inc)              ;Include the .inc file which contains general constants for
                                   ;RoboTrike system.
$INCLUDE(Parser.inc)               ;Include the .inc file which contains constants for parsing
                                   ;routine.
$INCLUDE(Motors.inc)               ;Include the .inc file which contains constatns for motors
                                   ;and laser turrets.
EXTRN   SetMotorSpeed:NEAR         ;Import SetMotorSpeed for setting motor speeds.
EXTRN    GetMotorSpeed:NEAR        ;Import GetMotorSpeed for obtaining current motor speeds.
EXTRN    GetMotorDirection:NEAR    ;Import GetMotorDirection for obtaining current direction.
EXTRN    SetLaser:NEAR             ;Import SetLaser for turning laser on.
EXTRN    SetTurretAngle:NEAR       ;Import SetTurretAngle for setting the turret's angle.
EXTRN    SetRelTurretAngle:NEAR    ;Import SetRelTurretAngle for increasing or decreasing the
                                   ;turret's angle.
EXTRN    SetTurretElevation:NEAR   ;Import SetTurretElevation for setting the elevation of
                                   ;the turret.

CGROUP GROUP CODE
DGROUP GROUP DATA

CODE SEGMENT PUBLIC 'CODE'

```

```
60          ASSUME  CS:CGROUP, DS:DGROUP
61
62  ; ParseSerialChar
63  ;
64  ; Description:
65  ;       The function is passed a character (c) which is presumed to be from the serial
    input.
66  ;       The character (c) is passed by value in AL. It executes proper functions of
    current
67  ;       transition and proceed from the current state to the next state. The function
    returns
68  ;       the status of the parsing operation in AX. PARSER_SUCCESS is returned if there
    is no
69  ;       parsing error and PARSER_ERROR is returned if there is any parsing error.
70  ; Operation:
71  ;       It first calls GetToken to obtain the token value and token type. Next, it
    calculates
72  ;      the offset for current transition by using the StateTable and the token type.
73  ;       Then, it calls the action function of current transition with token value as an
74  ;       argument. When the action function is done, it proceeds ParseState to the next
    state.
75  ;       Finally, it checks if there was an error with parsing and executing commands. If
76  ;       there was, it resets the ParseState to ST_INIT and resets all shared variables
77  ;       related to parsing routine.
78  ; Arguments:
79  ;      character(AL) - character to be parsed
80  ; Return Value:
81  ;      ParserErrorFlag(AX)     -     PARSER_SUCCESS  if there's no parsing error
82  ;                              -     PARSER_ERROR    if there's a parsing error
83  ; Local Variables:
84  ;      TokenType(DH)     -  The type of the token
85  ;      TokenValue(CH)    -  The value of the token
86  ;      TableOffset(BX)   -  The offset for state table
87  ; Shared Variables:
88  ;       ParserState      -   [R/W]   -  The current state of parsing state
    machine
89  ; Global Variables:
90  ;      None
91  ; Input:
92  ;      None
93  ; Output:
94  ;      None
95  ; Error Handling:
96  ;      PARSER_ERROR will be returned in AX if there's a parsing error.
97  ; Algorithms:
98  ;      None
99  ; Data Structures:
100 ;      Finite State Machine
101 ; Registers Changed:
102 ;      AX, BX, CX, DX, Flags
103 ; Limitations:
104 ;      None
105 ; Known bugs:
106 ;      None
107 ; Special Notes:
108 ;      None
109 ; Author:
110 ;      Sunghoon Choi
111 ; Revision History:
112 ;      11/24/2016    Sunghoon Choi    Created
113 ;      11/25/2016    Sunghoon Choi    Initial Compilation
114 ;      11/26/2016    Sunghoon Choi    Revised Comments
115
116 ParseSerialChar PROC    NEAR
117                 PUBLIC  ParseSerialChar
118
```

```
119 GetTheToken:
120     CALL GetToken                ;Get the token value and token
        type
121     MOV DH, AH                   ;DH = Token type
122     MOV CH, AL                   ;CH = Token Value
123 GetNextTableIndex:
124      MOV    AL, NUM_TOKEN_TYPES  ;find row in the table
125     MUL    ParserState           ;AX is start of row for current state
126     ADD    AL, DH                ;get the actual transition
127     ADC    AH, 0                 ;propagate low byte carry into high byte
128
129     IMUL    BX, AX, SIZE TRANSITION_ENTRY   ;BX = Table Offset
130
131 DoNextAction:
132     MOV AL, CH                          ;Retrieve the token value to use it as an argument
133                                         ;for functions to be called by the transition.
134     CALL CS:StateTable[BX].ACTION       ;Do the proper action(function) for the transition.
135                                         ;All action functions will return the parsing
                                            result
136                                         ;in AX.
137 GoNextState:
138     MOV BL, CS:StateTable[BX].NEXT_STATE ;Go to the next state of the state machine.
139     MOV ParserState, BL                 ;Update ParserState with the new state.
140 CheckParserError:
141     CMP AX, PARSER_ERROR                ;Was there any error with parsing and executing
142                                         ;functions?
143     JNE EndParserSerialChar             ;There was no error. Finish ParseSerialChar
        procedure.
144 ;JE ResetParserState                   ;There was an error. Reset the state to the
    initial
145                                         ;state and reset ArgNum and ArgSign.
146 ResetParserState:
147     MOV ParserState, ST_INITIAL         ;Reset the state back to the ST_INITIAL.
148     CALL ResetHandle                    ;Reset ArgNum and ArgSign.
149 EndParserSerialChar:
150     RET                                 ;End of ParserSerialChar
151 ParseSerialChar ENDP
152
153
154
155
156 ; InitParser
157 ;
158 ; Description:
159 ;      It initializes all shared variables related to parsing routine.
160 ; Operation:
161 ;      It first initializes ParserState to ST_INITIAL.
162 ;      Then, it calls ResetHandle to initialize ArgNum and ArgSign.
163 ; Arguments:
164 ;     None
165 ; Return Value:
166 ;     None
167 ; Local Variables:
168 ;     None
169 ; Shared Variables:
170 ;      ParserState     -    [Write]  -  The current state of parsing state
    machine
171 ; Global Variables:
172 ;     None
173 ; Input:
174 ;     None
175 ; Output:
176 ;     None
177 ; Error Handling:
178 ;     None
179 ; Algorithms:
```

```
;     None
; Data Structures:
;     Finite State Machine
; Registers Changed:
;     None
; Limitations:
;     None
; Known bugs:
;     None
; Special Notes:
;     None
; Author:
;     Sunghoon Choi
; Revision History:
;     11/24/2016    Sunghoon Choi    Created
;     11/25/2016    Sunghoon Choi    Initial Compilation
;     11/26/2016    Sunghoon Choi    Revised Comments

InitParser  PROC    NEAR
            PUBLIC  InitParser


InitParserState:
    MOV ParserState, ST_INITIAL ;Initializes ParserState to the initial state of
                                ;finite state machine.
ResetParserHandle:
    CALL ResetHandle            ;Calls ResetHandle to reset ArgNum and ArgSign.

EndInitParser:
    RET                        ;End of InitParser

InitParser  ENDP



; ResetHandle
;
; Description:
;      It resets shared variables(ArgNum, ArgSign) of parsing routine.
; Operation:
;      It resets ArgSign to ARG_POSITIVE and ArgNum to ARG_INIT.
; Arguments:
;     None
; Return Value:
;     None
; Local Variables:
;     None
; Shared Variables:
;      ArgSign    -    [Write]  -  The sign of the argument for action functions.
;      ArgNum     -   [Write]   -  The value of the argument for action functions.
; Global Variables:
;     None
; Input:
;     None
; Output:
;     None
; Error Handling:
;     None
; Algorithms:
;     None
; Data Structures:
;     Finite State Machine
; Registers Changed:
;     None
; Limitations:
;     None
```

```
246   ; Known bugs:
247   ;     None
248   ; Special Notes:
249   ;     None
250   ; Author:
251   ;     Sunghoon Choi
252   ; Revision History:
253   ;     11/24/2016    Sunghoon Choi   Created
254   ;     11/25/2016    Sunghoon Choi   Initial Compilation
255   ;     11/26/2016    Sunghoon Choi   Revised Comments
256
257   ResetHandle PROC    NEAR
258               PUBLIC  ResetHandle
259
260   InitArguments:
261       MOV ArgSign, ARG_POSITIVE    ;Resets argument's sign to positive.
262
263       MOV ArgNum,  ARG_INIT        ;Resets the argument value to ARG_INIT.
264   EndResetHandle:
265       RET                          ;End of ResetHandle
266   ResetHandle ENDP
267
268
269   ; GetToken
270   ;
271   ; Description:
272   ;     This procedure returns the token class and token value for the passed character.
273   ;     The character is truncated to 7-bits.
274   ; Operation:
275   ;      First, it clears the unused high bit by using a mask.
276   ;      Then, it obtains the token type and token value of the passed character by
277   ;      referring to TokenTypeTable and TokenValueTable.
278   ; Arguments:
279   ;     character(AL)    - The character to look up.
280   ; Return Value:
281   ;     TokenValue(AL) - token value of the character
282   ;     TokenType(AH)  - token type of the character.
283   ; Local Variables:
284   ;     None
285   ; Shared Variables:
286   ;      None
287   ; Global Variables:
288   ;     None
289   ; Input:
290   ;     None
291   ; Output:
292   ;     None
293   ; Error Handling:
294   ;     None
295   ; Algorithms:
296   ;     None
297   ; Data Structures:
298   ;     Finite State Machine
299   ; Registers Changed:
300   ;     AX, BX, Flags
301   ; Limitations:
302   ;     None
303   ; Known bugs:
304   ;     None
305   ; Special Notes:
306   ;     None
307   ; Author:
308   ;     Sunghoon Choi
309   ; Revision History:
310   ;     11/24/2016    Sunghoon Choi   Created
311   ;     11/25/2016    Sunghoon Choi   Initial Compilation
```

```
;     11/26/2016    Sunghoon Choi   Revised Comments

GetToken    PROC    NEAR
            PUBLIC  GetToken

MaskTokenBits:
    AND         AL, TOKEN_MASK              ;strip unused high bits
    MOV         AH, AL                      ;and preserve value in AH

GetTokenType:
    MOV     BX, OFFSET(TokenTypeTable)      ;BX points at table
    XLAT    CS:TokenTypeTable               ;Store token type in AL

    XCHG    AH, AL                          ;token type in AH, character in AL

GetTokenValue:
    MOV     BX, OFFSET(TokenValueTable)     ;BX points at table
    XLAT    CS:TokenValueTable              ;Store token value in AL

EndGetToken:
    RET                                     ;End of GetToken
GetToken ENDP


; DoSetAbsMotorSpeed
;
; Description:
;      It sets the absolute speed of the motors and returns PARSER_SUCCESS.
; Operation:
;      It sets AX to ArgNum and BX to IGNORE_ANGLE and calls SetMotorSpeed to set the
;      absolute motor speed. After updaating speed is done, it sets ErrorFlag to
;      PARSER_SUCCESS to notify that this action function was executed successfully.
;      Finally, it calls ResetHandle to reset shared variables and exits.
;      Note that BX's value is pushed in the beginning and popped in the end to keep its
;      value unchnaged. This is because BX's value has to be used as an index for finding
;      the next state in ParseSerialChar.
; Arguments:
;     None
; Return Value:
;     ParserErrorFlag(AX) = PARSER_SUCCESS
; Local Variables:
;     NewSpeed(AX)    -    The speed of motors to be set.
;      Angle(BX)      -   The direction angle of motors to be set.
; Shared Variables:
;      ArgNum     -   [Read]  -  The value of the argument for action functions.
; Global Variables:
;     None
; Input:
;     None
; Output:
;     None
; Error Handling:
;     None
; Algorithms:
;     None
; Data Structures:
;     Finite State Machine
; Registers Changed:
;     AX, Flags
; Limitations:
;     None
; Known bugs:
;     None
; Special Notes:
;     None
; Author:
```

```
;     Sunghoon Choi
; Revision History:
;     11/24/2016    Sunghoon Choi    Created
;     11/25/2016    Sunghoon Choi    Initial Compilation
;     11/26/2016    Sunghoon Choi    Revised Comments

DoSetAbsMotorSpeed  PROC    NEAR
                    PUBLIC  DoSetAbsMotorSpeed

   PUSH BX                  ;Save BX since BX's value has to be used as an index for
                            ;finding the next state in ParseSerialChar.
SetSpeedAngle:
   MOV AX, ArgNum           ;Speed(Argument 1 for SetMotorSpeed) = ArgNum
   MOV BX, IGNORE_ANGLE     ;Angle(Argument 2 for SetMotorSpeed) = IGNORE_ANGLE
                            ;Since we are setting the absolute speed, we should not change
                            ;the direction of motors.
   CALL SetMotorSpeed       ;With the two arguments, calls SetMotorSpeed to set the
                            ;absolute speed of motors.
EndDoSetAbsMotorSpeed:
   MOV AX, PARSER_SUCCESS   ;Since this action function is over, return PARSER_SUCCESS.
   CALL ResetHandle         ;Since the action function is over, we have to reset ArgNum
                            ;and ArgSign to execute the functions of next transition.
   POP BX                   ;Retrieve BX to find the next state in ParseSerialChar.
   RET                      ;End of DoSetAbsMotorSpeed
DoSetAbsMotorSpeed ENDP


; DoSetRelMotorSpeed
;
; Description:
;      It increases or decreases the motor speed by the value of ArgNum.
;      If the updated speed exceeds the upper limit or lower limit, it saturates the
;      NewSpeed to MAX_SPEED and MIN_SPEED. It returns PARSER_SUCCESS when setting
speed is
;      done.
; Operation:
;      It first calls GetMotorSpeed to obtain the current motor speed. Then, it checks
;      ArgSign to figure out whether ArgNum is positive or negative. If ArgNum is
positive,
;      it adds ArgNum to the current speed. Now, if a carry flag is set or the new speed
;      exceeds the upper limit(MAX_SPEED), it saturates the new speed to MAX_SPEED.
;      If ArgNum is negative, it subtracts ArgNum from the current speed.  If carry
flag is
;      set or the new speed exceeds lower limit(MIN_SPEED), it saturates the new speed
to
;      MIN_SPEED. Finally, it sets the angle to IGNORE_ANGLE and calls SetMotorSpeed to
;      change the motor speed. Before the procedure ends, it returns PARSER_SUCCESS in
AX
;      and calls ResetHandle to reset shared variables.
;      Note that BX's value is pushed in the beginning and popped in the end to keep its
;      value unchnaged. This is because BX's value has to be used as an index for finding
;      the next state in ParseSerialChar.
; Arguments:
;       None
; Return Value:
;     ParserErrorFlag(AX) = PARSER_SUCCESS
; Local Variables:
;     NewSpeed(AX)    -   The speed of motors to be set.
;      Angle(BX)      -   The direction angle of motors to be set.
; Shared Variables:
;      ArgSign     -  [Read]  -  The sign of the argument for action functions.
;      ArgNum      -  [Read]  -  The value of the argument for action
functions.
; Global Variables:
;     None
; Input:
```

```
;     None
; Output:
;     None
; Error Handling:
;     None
; Algorithms:
;     None
; Data Structures:
;     Finite State Machine
; Registers Changed:
;     AX, Flags
; Limitations:
;     None
; Known bugs:
;     None
; Special Notes:
;     None
; Author:
;     Sunghoon Choi
; Revision History:
;     11/24/2016    Sunghoon Choi   Created
;     11/25/2016    Sunghoon Choi   Initial Compilation
;     11/26/2016    Sunghoon Choi   Revised Comments

DoSetRelMotorSpeed  PROC    NEAR
                    PUBLIC  DoSetRelMotorSpeed

    PUSH BX                     ;Save BX since BX's value has to be used as an index for
                                ;finding the next state in ParseSerialChar.
ParserGetMotorSpeed:
    CALL   GetMotorSpeed        ;AX = CurrentSpeed [0,65534]
CheckSpeedSign:
    CMP ArgSign, ARG_POSITIVE   ;Is the argument positive?
    JNE SubtractSpeed           ;No, it's negative. So we have to decrease the speed.
    ;JE AddSpeed                ;Yes, it's positive. So we have to add the speed.
AddSpeed:
    ADD AX, ArgNum              ;AX = NewSpeed = CurrentSpeed + ArgNum
    JC SaturateMAXSpeed         ;If a carry was generated due to adding the speeds,
                                ;we have to saturate the speed to MAX_SPEED since it
                                ;indicates that the calculation result has exceeded the
                                ;maximum value of WORD data.
    CMP AX, MAX_SPEED           ;Does NewSpeed exceed MAX_SPEED?
    JBE SetRelMotorSpeed        ;No. Thus, we don't need to saturate the speed.
                                ;Go update the speed.
    ;JG SaturateMAXSpeed        ;Yes. Thus, we have to saturate the speed to MAX_SPEED
                                ;before updating the speed.
SaturateMAXSpeed:
    MOV AX, MAX_SPEED           ;Saturate the NewSpeed to MAX_SPEED.
    JMP SetRelMotorSpeed        ;Now that we are done with calculating the NewSpeed,
                                ;let's go update the speed.
SubtractSpeed:
    NEG ArgNum                  ;Since we are going to subtract speed, we need to first
                                ;get the absolute value of negative ArgNum.
    SUB AX, ArgNum              ;New Speed = CurrentSpeed - Abs(ArgNum)
    JC SaturateMINSpeed         ;If a carry was borrowed for the subtraction, it means
                                ;that the calculated speed was below zero. Thus, we have
                                ;to saturate the speed to MIN_SPEED.
    CMP AX, MIN_SPEED           ;Is NewSpeed below MIN_SPEED?
    JAE SetRelMotorSpeed        ;No. Thus, we don't need to saturate the NewSpeed to
                                ;MIN_SPEED. Go update the speed.
    ;JB SetRelMotorSpeed        ;YES. Thus, we have to saturate the speed to MIN_SPEED
                                ;before updating the speed.
SaturateMINSpeed:
    MOV AX, MIN_SPEED           ;Saturate the NewSpeed to MIN_SPEED.
    JMP SetRelMotorSpeed        ;We're done with calculating NewSpeed.
                                ;Thus, go update the speed.
```

```
504
505   SetRelMotorSpeed:
506       MOV BX, IGNORE_ANGLE        ;Since we are updating only the speed, we should keep
          the
507                                   ;direction unchanged.
508                                   ;AX = NewSpeed BX = IGNORE_ANGLE
509       CALL SetMotorSpeed            ;Call SetMotorSpeed to update the speed.
510
511   EndDoSetRelMotorSpeed:
512       MOV AX, PARSER_SUCCESS      ;We're done with setting relative motor speed.
513                                   ;Thus, return PARSER_SUCCESS
514       CALL ResetHandle            ;Since the action function is over, we have to reset
          ArgNum
515                                   ;and ArgSign to execute the functions of next transition.
516       POP BX                      ;Retrieve BX to find the next state in ParseSerialChar.
517       RET                         ;End of SetRelMotorSpeed
518   DoSetRelMotorSpeed  ENDP
519
520
521   ; DoSetMotorDirection
522   ;
523   ; Description:
524   ;      It increases or decreases the current angle of RoboTrike's direction by the degree
525   ;      of argument value. It returns PARSER_SUCCESS when setting angle is done.
526   ; Operation:
527   ;      It first calls GetMotorDirection to obtain the current direction angle of
      RoboTrike.
528   ;      Then, it starts normalizing ArgNum. If ArgNum is positive, it normalizes ArgNum by
529   ;      "ArgNum = ArgNum mod FULL_ANGLE". If ArgNum is negative, it normalizes ArgNum by
530   ;      "ArgNum = FULL_ANGLE - (abs(ArgNum) mod FULL_ANGLE)". When the normalization is
      done,
531   ;      it adds ArgNum to the current angle of RoboTrike. Since ArgNum is normalized to
      the
532   ;      range of [SRAIGHT_ANGLE, FULL_ANGLE-1], we don't need to subtract ArgNum from the
533   ;      current angle. Then, it sets AX(speed) to IGNORE_SPEED and calls SetMotorSpeed to
534   ;      update the direction of RoboTrike. Finally, it returns PARSER_SUCCESS in AX and
      calls
535   ;      ResetHandle to reset ArgNum and ArgSign.
536   ;      Note that BX's value is pushed in the beginning and popped in the end to keep its
537   ;      value unchnaged. This is because BX's value has to be used as an index for finding
538   ;      the next state in ParseSerialChar.
539   ; Arguments:
540   ;       None
541   ; Return Value:
542   ;     ParserErrorFlag(AX)      =   PARSER_SUCCESS
543   ; Local Variables:
544   ;     Speed(AX)                -   The speed of motors to be set.
545   ;      NormalizedArgNum(DX)    -   Normalized value of ArgNum
546   ;      NewAngle(BX)            -   The direction angle of motors to be set.
547   ; Shared Variables:
548   ;      ArgSign     -  [Read]   -   The sign of the argument for action functions.
549   ;      ArgNum      -  [R/W]    -   The value of the argument for action
      functions.
550   ; Global Variables:
551   ;     None
552   ; Input:
553   ;     None
554   ; Output:
555   ;     None
556   ; Error Handling:
557   ;     None
558   ; Algorithms:
559   ;     None
560   ; Data Structures:
561   ;     Finite State Machine
562   ; Registers Changed:
```

```
;     AX, CX, DX, Flags
; Limitations:
;     None
; Known bugs:
;     None
; Special Notes:
;     None
; Author:
;     Sunghoon Choi
; Revision History:
;     11/24/2016    Sunghoon Choi    Created
;     11/25/2016    Sunghoon Choi    Initial Compilation
;     11/26/2016    Sunghoon Choi    Revised Comments

DoSetMotorDirection PROC    NEAR
                    PUBLIC  DoSetMotorDirection

    PUSH BX                 ;Save BX since BX's value has to be used as an index for
                            ;finding the next state in ParseSerialChar.
SetParserMotorDirection:
    CALL GetMotorDirection  ;Obtain the current direction of motors in AX.
    MOV CX, AX              ;Save the current direction in CX.

    MOV AX, ArgNum          ;Start normalizing ArgNum before setting the direction.
                            ;Save ArgNum in AX to start normalization.
ParserNormAngle:
    CMP AX, STRAIGHT_ANGLE  ;Is ArgNum negative?
    JL ParserNormNegAngle   ;Yes. Start normalizing negative angle
;   JGE NormPosAngle        ;No. Start normalizing positive angle
ParserNormPosAngle:
    MOV BX, FULL_ANGLE      ;Divisor(BX)  = FULL_ANGLE
                            ;Dividend(AX) = ArgNum
    XOR DX,DX               ;Clear DX for DIV instruction
    DIV BX                  ;DX = ArgNum mod FULL_ANGLE
    MOV ArgNum, DX          ;Update ArgNum with the normalized ArgNum
    JMP ParserUpdateDirection    ;Now that we're done with normalizing ArgNum,
                                 ;go update the direction.
ParserNormNegAngle:
    NEG AX                  ;Get the absolute value of negative ArgNum.
    MOV BX, FULL_ANGLE      ;Divisor(BX)  = FULL_ANGLE
                            ;Dividend(AX) = ArgNum
    XOR DX,DX               ;Clear DX for DIV instruction.
    DIV BX                  ;DX = abs(ArgNum) mod FULL_ANGLE
    NEG DX                  ;DX = - (abs(ArgNum) mod FULL_ANGLE)

    ADD DX, FULL_ANGLE      ;DX = normalized angle = FULL_ANGLE-(abs(angle) mod
    FULL_ANGLE)
    MOV ArgNum, DX          ;update ArgNum with the noramlized angle.
ParserUpdateDirection:
    MOV BX, CX              ;Retrive current direction.
    ADD BX, ArgNum          ;Whether ArgNum was negative or positive, it is now
    normalized to
                            ;non-negative range, [STRAIGHT_ANGLE, FULL_ANGLE-1]. Thus,
                            just
                            ;adding normalized ArgNum to current direction will set the
                            new
                            ;direction appropriately.
                            ;NewAngle = Current Direction + Normalized ArgNum
ParserSetMotorDirection:
    MOV AX, IGNORE_SPEED    ;Since this function is changing the direction, we should not
                            ;change the speed. Thus, set the speed argument to
                            IGNORE_SPEED.
                            ;AX = IGNORE_SPEED BX = NewAngle
    CALL SetMotorSpeed      ;Call SetMotorSpeed to update the direction.

EndDoSetMotorDirection:
```

```
624         MOV AX, PARSER_SUCCESS  ;We're done with increasing or decreasing the motors angle.
625                                 ;Thus, return PARSER_SUCCESS
626         CALL ResetHandle        ;Since the action function is over, we have to reset ArgNum
627                                 ;and ArgSign to execute the functions of next transition.
628         POP BX                  ;Retrieve BX to find the next state in ParseSerialChar.
629         RET                     ;End of DoSetMotorDirection
630     DoSetMotorDirection ENDP
631
632
633
634     ; DoSetLaserOn
635     ;
636     ; Description:
637     ;       Turns the laser on and returns PARSER_SUCCESS
638     ; Operation:
639     ;       It sets AX to LASER_ON and calls SetLaser to turn the laser on.
640     ;       Then, it returns PARSER_SUCCESS in AX and calls ResetHandle to reset ArgNum and
641     ;       ArgSign. Note that BX's value is pushed in the beginning and popped in the end to
642     ;       keep its value unchnaged. This is because BX's value has to be used as an index
        for
643     ;       finding the next state in ParseSerialChar.
644     ; Arguments:
645     ;        None
646     ; Return Value:
647     ;     ParserErrorFlag(AX) = PARSER_SUCCESS
648     ; Local Variables:
649     ;     LaserArg(AX)        -    Indicates whether laser should be turned on or not.
650     ;                              It is an argument for SetLaser.
651     ; Shared Variables:
652     ;       None
653     ; Global Variables:
654     ;     None
655     ; Input:
656     ;     None
657     ; Output:
658     ;     None
659     ; Error Handling:
660     ;     None
661     ; Algorithms:
662     ;     None
663     ; Data Structures:
664     ;     Finite State Machine
665     ; Registers Changed:
666     ;     AX, Flags
667     ; Limitations:
668     ;     None
669     ; Known bugs:
670     ;     None
671     ; Special Notes:
672     ;     None
673     ; Author:
674     ;     Sunghoon Choi
675     ; Revision History:
676     ;     11/24/2016    Sunghoon Choi    Created
677     ;     11/25/2016    Sunghoon Choi    Initial Compilation
678     ;     11/26/2016    Sunghoon Choi    Revised Comments
679     DoSetLaserOn    PROC    NEAR
680                     PUBLIC  DoSetLaserOn
681
682         PUSH BX                 ;Save BX since BX's value has to be used as an index for
683                                 ;finding the next state in ParseSerialChar.
684     TurnLaserOn:
685         MOV AX, LASER_ON        ;AX is an argument for SetLaser function.
686                                 ;Set it to LASER_ON.
687         CALL SetLaser           ;Calls SetLaser to turn the laser on.
688     EndDoSetLaserOn:
```

```
689         MOV AX, PARSER_SUCCESS  ;Since this action function was handled successfully,
690                                 ;return PARSER_SUCCESS.
691         CALL ResetHandle        ;Resets ArgNum and ArgSign for executing the functions of
692                                 ;next transition.
693
694         POP BX                  ;Retrieve BX to find the next state in ParseSerialChar.
695         RET                     ;End of DoSetLaserOn
696     DoSetLaserOn    ENDP
697
698
699     ; DoSetLaserOff
700     ;
701     ; Description:
702     ;       Turns the laser off and returns PARSER_SUCCESS
703     ; Operation:
704     ;       It sets AX to LASER_OFF and calls SetLaser to turn the laser off.
705     ;       Then, it returns PARSER_SUCCESS in AX and calls ResetHandle to reset ArgNum and
706     ;       ArgSign. Note that BX's value is pushed in the beginning and popped in the end to
707     ;       keep its value unchnaged. This is because BX's value has to be used as an index
        for
708     ;      finding the next state in ParseSerialChar.
709     ; Arguments:
710     ;        None
711     ; Return Value:
712     ;      ParserErrorFlag(AX) =    PARSER_SUCCESS
713     ; Local Variables:
714     ;      LaserArg(AX)       -    Indicates whether laser should be turned on or not.
715     ;                               It is an argument for SetLaser.
716     ; Shared Variables:
717     ;       None
718     ; Global Variables:
719     ;      None
720     ; Input:
721     ;      None
722     ; Output:
723     ;      None
724     ; Error Handling:
725     ;      None
726     ; Algorithms:
727     ;      None
728     ; Data Structures:
729     ;      Finite State Machine
730     ; Registers Changed:
731     ;      AX, Flags
732     ; Limitations:
733     ;      None
734     ; Known bugs:
735     ;      None
736     ; Special Notes:
737     ;      None
738     ; Author:
739     ;      Sunghoon Choi
740     ; Revision History:
741     ;      11/24/2016    Sunghoon Choi   Created
742     ;      11/25/2016    Sunghoon Choi   Initial Compilation
743     ;      11/26/2016    Sunghoon Choi   Revised Comments
744     DoSetLaserOff    PROC    NEAR
745                      PUBLIC  DoSetLaserOff
746
747         PUSH BX                 ;Save BX since BX's value has to be used as an index for
748                                 ;finding the next state in ParseSerialChar.
749     TurnLaserOff:
750         MOV AX, LASER_OFF       ;AX is an argument for SetLaser function.
751                                 ;Set it to LASER_OFF to turn laser off.
752         CALL SetLaser           ;Calls SetLaser to turn the laser off.
753     EndDoSetLaserOff:
```

```
754         MOV AX, PARSER_SUCCESS  ;Since this action function was handled successfully,
755                                 ;return PARSER_SUCCESS.
756         CALL ResetHandle        ;Resets ArgNum and ArgSign for executing the functions of
757                                 ;next transition.
758         POP BX                  ;Retrieve BX to find the next state in ParseSerialChar.
759         RET                     ;End of DoSetLaserOff
760     DoSetLaserOff    ENDP
761
762
763     ; DoSetAbsTurretAngle
764     ;
765     ; Description:
766     ;      Sets the turret's absolute angle and return PARSER_SUCCESS.
767     ; Operation:
768     ;      It sets AX to ArgNum, which is the absolute angle of the turret to be set.
769     ;      Then, it calls SetTurretAngle to set the absolute angle of the turret.
770     ;      Finally, it returns PARSER_SUCCESS in AX and calls ResetHandle to reset ArgNum and
771     ;      ArgSign. Note that BX's value is pushed in the beginning and popped in the end to
772     ;      keep its value unchnaged. This is because BX's value has to be used as an index
        for
773     ;     finding the next state in ParseSerialChar.
774     ; Arguments:
775     ;       None
776     ; Return Value:
777     ;     ParserErrorFlag(AX) = PARSER_SUCCESS
778     ; Local Variables:
779     ;     TurretAngle(AX)   -   Angle of Turret to be set.
780     ;                           Argument for SetTurretAngle
781     ; Shared Variables:
782     ;      ArgNum      -  [Read]  -  The value of the argument for action
        functions.
783     ; Global Variables:
784     ;     None
785     ; Input:
786     ;     None
787     ; Output:
788     ;     None
789     ; Error Handling:
790     ;     None
791     ; Algorithms:
792     ;     None
793     ; Data Structures:
794     ;     Finite State Machine
795     ; Registers Changed:
796     ;     AX, Flags
797     ; Limitations:
798     ;     None
799     ; Known bugs:
800     ;     None
801     ; Special Notes:
802     ;     None
803     ; Author:
804     ;     Sunghoon Choi
805     ; Revision History:
806     ;     11/24/2016    Sunghoon Choi   Created
807     ;     11/25/2016    Sunghoon Choi    Initial Compilation
808     ;     11/26/2016    Sunghoon Choi    Revised Comments
809
810     DoSetAbsTurretAngle    PROC    NEAR
811                            PUBLIC  DoSetAbsTurretAngle
812         PUSH BX                 ;Save BX since BX's value has to be used as an index for
813                                 ;finding the next state in ParseSerialChar.
814     ParserSetTurretAngle:
815         MOV AX, ArgNum          ;Set AX to ArgNum, which is the angle of turret to be set,
            as
816                                 ;an argument for SetTurretAngle
```

```
    CALL SetTurretAngle     ;Call SetTurretAngle with the argument AX(=ArgNum) to set
                            ;the turret's angle.
EndDoSetAbsTurretAngle:
    MOV AX, PARSER_SUCCESS  ;Since this action function was handled successfully,
                            ;return PARSER_SUCCESS.
    CALL ResetHandle        ;Resets ArgNum and ArgSign for executing the functions of
                            ;next transition.
    POP BX                  ;Retrieve BX to find the next state in ParseSerialChar.
    RET                     ;End of DoSetAbsTurretAngle
DoSetAbsTurretAngle    ENDP



; DoSetRelTurretAngle
;
; Description:
;      Sets the turret's relative angle and return PARSER_SUCCESS.
; Operation:
;      It sets AX to ArgNum, which is the relative angle of the turret to be set.
;      Then, it calls SetRelTurretAngle to set the relative angle of the turret.
;      Finally, it returns PARSER_SUCCESS in AX and calls ResetHandle to reset ArgNum and
;      ArgSign. Note that BX's value is pushed in the beginning and popped in the end to
;      keep its value unchnaged. This is because BX's value has to be used as an index
for
;     finding the next state in ParseSerialChar.
; Arguments:
;       None
; Return Value:
;     ParserErrorFlag(AX) = PARSER_SUCCESS
; Local Variables:
;     TurretRelAngle(AX)     -    Relative angle of Turret to be set.
;                                 Argument for SetRelTurretAngle
; Shared Variables:
;      ArgNum      -  [Read] -    The value of the argument for action
functions.
; Global Variables:
;     None
; Input:
;     None
; Output:
;     None
; Error Handling:
;     None
; Algorithms:
;     None
; Data Structures:
;     Finite State Machine
; Registers Changed:
;     AX, Flags
; Limitations:
;     None
; Known bugs:
;     None
; Special Notes:
;     None
; Author:
;     Sunghoon Choi
; Revision History:
;     11/24/2016    Sunghoon Choi   Created
;     11/25/2016    Sunghoon Choi   Initial Compilation
;     11/26/2016    Sunghoon Choi   Revised Comments
DoSetRelTurretAngle    PROC    NEAR
                       PUBLIC  DoSetRelTurretAngle
    PUSH BX                 ;Save BX since BX's value has to be used as an index for
                            ;finding the next state in ParseSerialChar.
ParserSetRelTurretAngle:
```

```
    MOV AX, ArgNum          ;Set AX to ArgNum, which is the relative angle of turret to
                            ;be set as an argument for SetRelTurretAngle
    CALL SetRelTurretAngle  ;Call SetRelTurretAngle with the argument AX(=ArgNum) to set
                            ;the turret's relative angle.
EndDoSetRelTurretAngle:
    MOV AX, PARSER_SUCCESS  ;Since this action function was handled successfully,
                            ;return PARSER_SUCCESS.
    CALL ResetHandle        ;Resets ArgNum and ArgSign for executing the functions of
                            ;next transition.
    POP BX                  ;Retrieve BX to find the next state in ParseSerialChar.
    RET                     ;End of DoSetRelTurretAngle
DoSetRelTurretAngle    ENDP


; DoSetTurretElevation
;
; Description:
;      Sets the elevation angle of the turret. If the elevation angle is not in the range of
;      [NEG_TURRET_ELEV_BOUND, POS_TURRET_ELEV_BOUND], it returns PARSER_ERROR.
; Operation:
;      First, it checks if ArgNum is in the range of [NEG_TURRET_ELEV_BOUND, POS_TURRET_ELEV
;      _BOUND]. If it is not, returns PARSER_ERROR in AX and exits the procdeure.
;      If it is in the range of [NEG_TURRET_ELEV_BOUND, POS_TURRET_ELEV_BOUND], it sets
;      AX to ArgNum and calls SetTurretElevation to set the turret elevation angle.
;      Finally, it returns PARSER_SUCCESS in AX and calls ResetHandle to reset ArgNum and
;      ArgSign. Note that BX's value is pushed in the beginning and popped in the end to
;      keep its value unchnaged. This is because BX's value has to be used as an index for
;     finding the next state in ParseSerialChar.
; Arguments:
;     None
; Return Value:
;     ParserErrorFlag(AX)    -    PARSER_SUCCESS  if there's no parsing error
;                           -    PARSER_ERROR    if there's a parsing error
; Local Variables:
;     ElevationAngle(AX)     -    The angle of turret elevation
; Shared Variables:
;      ArgNum      -  [Read] -    The value of the argument for action functions.
; Global Variables:
;     None
; Input:
;     None
; Output:
;     None
; Error Handling:
;     It returns PARSER_ERROR if the elevation angle is not in the range of
;      [NEG_TURRET_ELEV_BOUND, POS_TURRET_ELEV_BOUND].
; Algorithms:
;     None
; Data Structures:
;     Finite State Machine
; Registers Changed:
;     AX, Flags
; Limitations:
;     None
; Known bugs:
;     None
; Special Notes:
;     None
; Author:
;     Sunghoon Choi
; Revision History:
;     11/24/2016    Sunghoon Choi   Created
```

```
;     11/25/2016    Sunghoon Choi   Initial Compilation
;     11/26/2016    Sunghoon Choi   Revised Comments

DoSetTurretElevation   PROC   NEAR
                       PUBLIC  DoSetTurretElevation

    PUSH BX            ;Save BX since BX's value has to be used as an index for
                       ;finding the next state in ParseSerialChar.
CheckTurretPosBound:
    CMP ArgNum, POS_TURRET_ELEV_BOUND     ;Does ArgNum(Turret Elevation Angle) exceed
                                          ;POS_TURRET_ELEV_BOUND?
    JG TurretElevationError               ;Yes. The ArgNum is illegal. Thus, go return
                                          ;the PARSER_ERROR.
   ;JLE CheckTurretNegBound               ;No. Now check the lower boundary.
CheckTurretNegBound:
    CMP ArgNum, NEG_TURRET_ELEV_BOUND     ;Is ArgNum(Turret Elevation Angle) less than
                                          ;NEG_TURRET_ELEV_BOUND?
    JL  TurretElevationError              ;Yes. The ArgNum is illegal. Thus, go return
                                          ;the PARSER_ERROR.
   ;JGE ParserSetTurretElevation          ;No. The ArgNum is legal. Thus, go set
                                          ;the turret elevation angle.
ParserSetTurretElevation:
    MOV AX, ArgNum                        ;Set AX to ArgNum(Turret Elevation Angle) as
                                          ;an argument for SetTurretElevation
    CALL SetTurretElevation               ;Call SetTurretElevation to set the elvation
                                          ;angle of the turret.
TurretElevationSuccess:
    MOV AX, PARSER_SUCCESS                ;Since this action function was handled
                                          ;successfully, return PARSER_SUCCESS.
    JMP EndParserSetTurretElevation       ;Go reset ArgNum and ArgSign for next
    transitions

TurretElevationError:
    MOV AX, PARSER_ERROR                  ;Setting turret elevation was not processed
                                          ;successfully. Thus, return PARSER_ERROR.
EndParserSetTurretElevation:
    CALL ResetHandle                      ;Resets ArgNum and ArgSign for executing the
                                          ;functions of next transition.
    POP BX                                ;Retrieve BX to find the next state in
                                          ;ParseSerialChar.
    RET                                   ;End of DoSetTurretElevation
DoSetTurretElevation    ENDP



; DoAddDigit
;
; Description:
;      It adds a digit to the shared variable ArgNum. It returns PARSER_ERROR if an
overflow
;      or an underflow occurs while adding the digit. It returns PARSER_SUCCESS if there
;      was no overflow or underflow.
; Operation:
;      First, it multiplies current ArgNum by DECIMAL_BASE. Then, if ArgSign is
ARG_POSITIVE,
;      it adds the digit to ArgNum. If it is ARG_NEGATIVE, it subtracts the digit from
;      ArgNum. If an overflow or underflow occurs in the process of addition or
subtraction,
;      it returns PARSER_ERROR. If there's no overflow or underfow, it returns
PARSER_SUCCESS.
;      Note that BX's value is pushed in the beginning and popped in the end to
;      keep its value unchnaged. This is because BX's value has to be used as an index
for
;     finding the next state in ParseSerialChar.
; Arguments:
;     Digit(AL)         -    The digit to be added to ArgNum
```

```
; Return Value:
;     ParserErrorFlag(AX)     -    PARSER_SUCCESS  if there's no parsing error
;                             -    PARSER_ERROR    if there's a parsing error
; Local Variables:
;      DecMultipledArg(AX)    -    The value of ArgNum multiplied with DECIMAL_BASE
;     Digit(CX)               -    The new digit to be added to ArgNum
; Shared Variables:
;      ArgSign     -  [Read]  -    The sign of the argument for action functions.
;      ArgNum      -  [R/W]   -    The value of the argument for action
functions.
; Global Variables:
;     None
; Input:
;     None
; Output:
;     None
; Error Handling:
;     It returns PARSER_ERROR if there's an overflow or underflow in calculation.
; Algorithms:
;     None
; Data Structures:
;     Finite State Machine
; Registers Changed:
;     AX, CX, DX, Flags
; Limitations:
;     None
; Known bugs:
;     None
; Special Notes:
;     None
; Author:
;     Sunghoon Choi
; Revision History:
;     11/24/2016    Sunghoon Choi   Created
;     11/25/2016    Sunghoon Choi   Initial Compilation
;     11/26/2016    Sunghoon Choi   Revised Comments

DoAddDigit    PROC    NEAR
              PUBLIC    DoAddDigit

    PUSH BX                 ;Save BX since BX's value has to be used as an index for
                            ;finding the next state in ParseSerialChar.
    MOV CX, AX              ;Save the input digit in CL.
MultArgNumDec:
    MOV BX, DECIMAL_BASE    ;We are going to multiply ArgNum with DECIMAL_BASEE to add
                            ;a digit to it. Set BX to DECIMAL_BASE
    MOV AX, ArgNum          ;Set AX to ArgNum so that we can multiply ArgNum and
                            ;DECIMAL_BASE
    XOR DX,DX               ;Clear DX for IMUL instruction.
    IMUL BX                 ;DX:AX = ArgNum * DECIMAL_BASE
    MOV ArgNum, AX          ;Update ArgNum. "ArgNum = ArgNum * DECIMAL_BASE"
    JO    AddDigitError     ;If there was an overflow or underflow by mutliplication,
                            ;return PARSER_ERROR.
;    JNO AddDigitCheckSign  ;If there was no overflow(or underflow), go check ArgSign.
AddDigitCheckSign:
    CMP ArgSign, ARG_POSITIVE   ;Is ArgSign ARG_POSITIVE? (Is ArgNum positive?)
    JNE    SubtractDigitArg     ;No. Subtract the digit from ArgNum.
    ;JE    AddDigitArg          ;Yes. Add the digit to ArgNum.
AddDigitArg:
    XOR BX, BX              ;Clear BH since we will save the digit in BX.
    MOV BL, CL              ;Retrieve the digit in BL
    ADD ArgNum, BX          ;"New ArgNum = Decimal Multiplied ArgNum + digit"
    JO    AddDigitError     ;Return PARSER_ERROR if there was an overflow or
    underflow.
  ;JNO  AddDigitSuccess     ;If there was no overflow or underflow, return the
    JMP    AddDigitSuccess  ;PARSER_SUCCESS.
```

```
SubtractDigitArg:
    XOR BX, BX                ;Clear BH since we will save the digit in BX.
    MOV BL, CL                ;Retrieve the digit in BL
    SUB ArgNum, BX            ;"New ArgNum = Decimal Multiplied ArgNum - digit"
    JO    AddDigitError       ;Return PARSER_ERROR if there was an overflow or
    underflow.
  ;JNO  AddDigitSuccess       ;If there was no overflow or underflow, return the
    JMP    AddDigitSuccess    ;PARSER_SUCCESS.
AddDigitSuccess:
    MOV AX, PARSER_SUCCESS    ;Return PARSER_SUCCESS in AX.
    JMP EndDoAddDigit         ;Exit the procedure.
AddDigitError:
    MOV AX, PARSER_ERROR      ;Return PARSER_ERROR in AX.
EndDoAddDigit:
    POP BX                    ;Retrieve BX to find the next state in
    ParseSerialChar.
    RET
DoAddDigit    ENDP



; DoSetError
;
; Description:
;      It returns PARSER_ERROR in AX.
; Operation:
;      It sets AX to PARSER_ERROR.
; Arguments:
;     None
; Return Value:
;     ParserErrorFlag(AX) = PARSER_ERROR
; Local Variables:
;      None
; Shared Variables:
;      None
; Global Variables:
;     None
; Input:
;     None
; Output:
;     None
; Error Handling:
;     None
; Algorithms:
;     None
; Data Structures:
;     Finite State Machine
; Registers Changed:
;     AX
; Limitations:
;     None
; Known bugs:
;     None
; Special Notes:
;     None
; Author:
;     Sunghoon Choi
; Revision History:
;     11/24/2016    Sunghoon Choi    Created
;     11/25/2016    Sunghoon Choi    Initial Compilation
;     11/26/2016    Sunghoon Choi    Revised Comments

DoSetError  PROC    NEAR
            PUBLIC  DoSetError

ParserSetError:
```

```
1131      MOV AX, PARSER_ERROR       ;Returns PARSER_ERROR in AX.
1132    EndDoSetError:
1133      RET                        ;End of DoSetError.
1134    DoSetError  ENDP
1135
1136
1137    ; DoNOP
1138    ;
1139    ; Description:
1140    ;      It does nothing but returns PARSER_SUCCESS
1141    ; Operation:
1142    ;      It sets AX to PARSER_SUCCESS
1143    ; Arguments:
1144    ;     None
1145    ; Return Value:
1146    ;     ParserErrorFlag(AX) = PARSER_SUCCESS
1147    ; Local Variables:
1148    ;     None
1149    ; Shared Variables:
1150    ;      None
1151    ; Global Variables:
1152    ;     None
1153    ; Input:
1154    ;     None
1155    ; Output:
1156    ;     None
1157    ; Error Handling:
1158    ;     None
1159    ; Algorithms:
1160    ;     None
1161    ; Data Structures:
1162    ;     Finite State Machine
1163    ; Registers Changed:
1164    ;     AX
1165    ; Limitations:
1166    ;     None
1167    ; Known bugs:
1168    ;     None
1169    ; Special Notes:
1170    ;     None
1171    ; Author:
1172    ;     Sunghoon Choi
1173    ; Revision History:
1174    ;      11/24/2016    Sunghoon Choi    Created
1175    ;      11/25/2016    Sunghoon Choi    Initial Compilation
1176    ;      11/26/2016    Sunghoon Choi    Revised Comments
1177
1178    DoNOP    PROC    NEAR
1179             PUBLIC  DoNOP
1180
1181    ParserDoNOP:
1182      MOV AX, PARSER_SUCCESS        ;Returns PARSER_SUCCESS in AX.
1183    EndDoNOP:
1184       RET                         ;End of DoNOP
1185    DoNOP    ENDP
1186
1187
1188    ; DoSetNegSign
1189    ;
1190    ; Description:
1191    ;      It sets the sign flag of argument, ArgSign, to ARG_NEGATIVE.
1192    ; Operation:
1193    ;      It updates ArgSing with ARG_NEGATIVE. Then, it returns PARSER_SUCCESS in AX.
1194    ; Arguments:
1195    ;     None
1196    ; Return Value:
```

```
;     ParserErrorFlag(AX) = PARSER_SUCCESS
; Local Variables:
;     None
; Shared Variables:
;      ArgSign     -    [Write]  -  The sign of the argument for action
functions.
; Global Variables:
;     None
; Input:
;     None
; Output:
;     None
; Error Handling:
;     None
; Algorithms:
;     None
; Data Structures:
;     Finite State Machine
; Registers Changed:
;     None
; Limitations:
;     None
; Known bugs:
;     None
; Special Notes:
;     None
; Author:
;     Sunghoon Choi
; Revision History:
;     11/24/2016    Sunghoon Choi    Created
;     11/25/2016    Sunghoon Choi    Initial Compilation
;     11/26/2016    Sunghoon Choi    Revised Comments
DoSetNegSign    PROC    NEAR
             PUBLIC  DoSetNegSign

ParserSetNegSign:
   MOV ArgSign, ARG_NEGATIVE         ;Set ArgSign to ARG_NEGATIVE
EndDoSetNegSign:
   MOV AX, PARSER_SUCCESS            ;Return PARSER_SUCCESS in AX.
   RET
DoSetNegSign   ENDP



; StateTable
;
; Description:
;      This is the state transition table for the state machine.
;      Each entry consists of the next state and the action for that transition.
; Notes:
;       This table is declared PRIVATE to prevent other codes accessing the table.
;       Also, READ ONLY tables should always be in the code segment so that in a
standalone
;       system it will be located in the ROM with the code.
;
; Author:
;       Sunghoon Choi
; Revision history:
;      11/24/2016    Sunghoon Choi    Created
;      11/25/2016    Sunghoon Choi    Initial Compilation
;      11/26/2016    Sunghoon Choi    Revised Comments

TRANSITION_ENTRY        STRUC           ;structure used to define table
    NEXT_STATE   DB      ?              ;the next state for the transition
    ACTION       DW      ?              ;action for the transition
TRANSITION_ENTRY      ENDS
```

```
1261
1262
1263    ;define a macro to make table a little more readable
1264    ;macro just does an offset of the action routine entries to build the STRUC
1265    %*DEFINE(TRANSITION(nxtst, act))  (
1266        TRANSITION_ENTRY< %nxtst, OFFSET(%act) >
1267    )
1268
1269
1270    StateTable    LABEL    TRANSITION_ENTRY
1271
1272        ;Current State = ST_INITIAL                         Input Token Type
1273        %TRANSITION(ST_SPD,      DoNOP)                      ;TOKEN_SPEED
1274        %TRANSITION(ST_REL_SPD, DoNOP)                      ;TOKEN_RELATIVE_SPEED
1275        %TRANSITION(ST_DIR,      DoNOP)                       ;TOKEN_DIRECTION
1276        %TRANSITION(ST_LSR_ON,     DoNOP)                        ;TOKEN_LASER_ON
1277        %TRANSITION(ST_LSR_OFF,     DoNOP)                        ;TOKEN_LASER_OFF
1278        %TRANSITION(ST_TURRET_ANGLE, DoNop)                      ;TOKEN_TURRET_ANGLE
1279        %TRANSITION(ST_TURRET_ELEVATION, DoNop)                  ;TOKEN_TURRET_ELEVATION
1280        %TRANSITION(ST_INITIAL,     DoSetError)                    ;TOKEN_SIGN_POSITIVE
1281        %TRANSITION(ST_INITIAL, DoSetError)                     ;TOKEN_SIGN_NEGATIVE
1282        %TRANSITION(ST_INITIAL, DoSetError)                     ;TOKEN_DIGIT
1283        %TRANSITION(ST_INITIAL, DoNOP)                          ;TOKEN_IGNORE
1284        %TRANSITION(ST_INITIAL, DoNOP)                          ;TOKEN_RETURN
1285        %TRANSITION(ST_INITIAL, DoSetError)                     ;TOKEN_OTHER
1286
1287        ;Current State = ST_SPD                              Input Token Type
1288        %TRANSITION(ST_INITIAL,  DoSetError)                 ;TOKEN_SPEED
1289        %TRANSITION(ST_INITIAL,  DoSetError)                ;TOKEN_RELATIVE_SPEED
1290        %TRANSITION(ST_INITIAL,  DoSetError)                ;TOKEN_DIRECTION
1291        %TRANSITION(ST_INITIAL,  DoSetError)                ;TOKEN_LASER_ON
1292        %TRANSITION(ST_INITIAL,  DoSetError)                ;TOKEN_LASER_OFF
1293        %TRANSITION(ST_INITIAL,  DoSetError)                ;TOKEN_TURRET_ANGLE
1294        %TRANSITION(ST_INITIAL,  DoSetError)                ;TOKEN_TURRET_ELEVATION
1295        %TRANSITION(ST_SPD_SIGN, DoNOP)                        ;TOKEN_SIGN_POSITIVE
1296        %TRANSITION(ST_INITIAL,  DoSetError)                ;TOKEN_SIGN_NEGATIVE
1297        %TRANSITION(ST_SPD_DIGIT,DoAddDigit)                ;TOKEN_DIGIT
1298        %TRANSITION(ST_SPD,      DoNOP)                        ;TOKEN_IGNORE
1299        %TRANSITION(ST_INITIAL,  DoSetError)                ;TOKEN_RETURN
1300        %TRANSITION(ST_INITIAL,  DoSetError)                ;TOKEN_OTHER
1301
1302
1303        ;Current State = ST_SPD_SIGN                         Input Token Type
1304        %TRANSITION(ST_INITIAL, DoSetError)                   ;TOKEN_SPEED
1305        %TRANSITION(ST_INITIAL,    DoSetError)                   ;TOKEN_RELATIVE_SPEED
1306        %TRANSITION(ST_INITIAL, DoSetError)                   ;TOKEN_DIRECTION
1307        %TRANSITION(ST_INITIAL,    DoSetError)                   ;TOKEN_LASER_ON
1308        %TRANSITION(ST_INITIAL,    DoSetError)                   ;TOKEN_LASER_OFF
1309        %TRANSITION(ST_INITIAL, DoSetError)                   ;TOKEN_TURRET_ANGLE
1310        %TRANSITION(ST_INITIAL, DoSetError)                   ;TOKEN_TURRET_ELEVATION
1311        %TRANSITION(ST_INITIAL,    DoSetError)                   ;TOKEN_SIGN_POSITIVE
1312        %TRANSITION(ST_INITIAL, DoSetError)                   ;TOKEN_SIGN_NEGATIVE
1313        %TRANSITION(ST_SPD_DIGIT, DoAddDigit)                ;TOKEN_DIGIT
1314        %TRANSITION(ST_SPD_SIGN, DoNOP)                       ;TOKEN_IGNORE
1315        %TRANSITION(ST_INITIAL, DoSetError)                   ;TOKEN_RETURN
1316        %TRANSITION(ST_INITIAL, DoSetError)                   ;TOKEN_OTHER
1317
1318
1319        ;Current State = ST_SPD_DIGIT                         Input Token Type
1320        %TRANSITION(ST_INITIAL, DoSetError)                   ;TOKEN_SPEED
1321        %TRANSITION(ST_INITIAL,    DoSetError)                   ;TOKEN_RELATIVE_SPEED
1322        %TRANSITION(ST_INITIAL, DoSetError)                   ;TOKEN_DIRECTION
1323        %TRANSITION(ST_INITIAL,    DoSetError)                   ;TOKEN_LASER_ON
1324        %TRANSITION(ST_INITIAL,    DoSetError)                   ;TOKEN_LASER_OFF
1325        %TRANSITION(ST_INITIAL, DoSetError)                   ;TOKEN_TURRET_ANGLE
1326        %TRANSITION(ST_INITIAL, DoSetError)                   ;TOKEN_TURRET_ELEVATION
```

```
1327        %TRANSITION(ST_INITIAL,    DoSetError)                ;TOKEN_SIGN_POSITIVE
1328        %TRANSITION(ST_INITIAL, DoSetError)                  ;TOKEN_SIGN_NEGATIVE
1329        %TRANSITION(ST_SPD_DIGIT, DoAddDigit)               ;TOKEN_DIGIT
1330        %TRANSITION(ST_SPD_DIGIT, DoNOP)                    ;TOKEN_IGNORE
1331        %TRANSITION(ST_INITIAL, DoSetAbsMotorSpeed)          ;TOKEN_RETURN
1332        %TRANSITION(ST_INITIAL, DoSetError)                  ;TOKEN_OTHER
1333
1334
1335        ;Current State = ST_REL_SPD                        Input Token Type
1336        %TRANSITION(ST_INITIAL,  DoSetError)               ;TOKEN_SPEED
1337        %TRANSITION(ST_INITIAL,    DoSetError)                ;TOKEN_RELATIVE_SPEED
1338        %TRANSITION(ST_INITIAL, DoSetError)                  ;TOKEN_DIRECTION
1339        %TRANSITION(ST_INITIAL,    DoSetError)                ;TOKEN_LASER_ON
1340        %TRANSITION(ST_INITIAL,    DoSetError)                ;TOKEN_LASER_OFF
1341        %TRANSITION(ST_INITIAL, DoSetError)                  ;TOKEN_TURRET_ANGLE
1342        %TRANSITION(ST_INITIAL, DoSetError)                  ;TOKEN_TURRET_ELEVATION
1343        %TRANSITION(ST_REL_SPD_SIGN, DoNOP)                  ;TOKEN_SIGN_POSITIVE
1344        %TRANSITION(ST_REL_SPD_SIGN, DoSetNegSign)           ;TOKEN_SIGN_NEGATIVE
1345        %TRANSITION(ST_REL_SPD_DIGIT, DoAddDigit)           ;TOKEN_DIGIT
1346        %TRANSITION(ST_REL_SPD, DoNOP)                       ;TOKEN_IGNORE
1347        %TRANSITION(ST_INITIAL, DoSetError)                  ;TOKEN_RETURN
1348        %TRANSITION(ST_INITIAL, DoSetError)                  ;TOKEN_OTHER
1349
1350        ;Current State = ST_REL_SPD_SIGN                    Input Token Type
1351        %TRANSITION(ST_INITIAL, DoSetError)                  ;TOKEN_SPEED
1352        %TRANSITION(ST_INITIAL,    DoSetError)                ;TOKEN_RELATIVE_SPEED
1353        %TRANSITION(ST_INITIAL, DoSetError)                  ;TOKEN_DIRECTION
1354        %TRANSITION(ST_INITIAL,    DoSetError)                ;TOKEN_LASER_ON
1355        %TRANSITION(ST_INITIAL,    DoSetError)                ;TOKEN_LASER_OFF
1356        %TRANSITION(ST_INITIAL, DoSetError)                  ;TOKEN_TURRET_ANGLE
1357        %TRANSITION(ST_INITIAL, DoSetError)                  ;TOKEN_TURRET_ELEVATION
1358        %TRANSITION(ST_INITIAL,    DoSetError)                ;TOKEN_SIGN_POSITIVE
1359        %TRANSITION(ST_INITIAL, DoSetError)                  ;TOKEN_SIGN_NEGATIVE
1360        %TRANSITION(ST_REL_SPD_DIGIT, DoAddDigit)           ;TOKEN_DIGIT
1361        %TRANSITION(ST_REL_SPD_SIGN, DoNOP)                  ;TOKEN_IGNORE
1362        %TRANSITION(ST_INITIAL, DoSetError)                  ;TOKEN_RETURN
1363        %TRANSITION(ST_INITIAL, DoSetError)                  ;TOKEN_OTHER
1364
1365
1366        ;Current State = ST_REL_SPD_DIGIT                   Input Token Type
1367        %TRANSITION(ST_INITIAL, DoSetError)                  ;TOKEN_SPEED
1368        %TRANSITION(ST_INITIAL,    DoSetError)                ;TOKEN_RELATIVE_SPEED
1369        %TRANSITION(ST_INITIAL, DoSetError)                  ;TOKEN_DIRECTION
1370        %TRANSITION(ST_INITIAL,    DoSetError)                ;TOKEN_LASER_ON
1371        %TRANSITION(ST_INITIAL,    DoSetError)                ;TOKEN_LASER_OFF
1372        %TRANSITION(ST_INITIAL, DoSetError)                  ;TOKEN_TURRET_ANGLE
1373        %TRANSITION(ST_INITIAL, DoSetError)                  ;TOKEN_TURRET_ELEVATION
1374        %TRANSITION(ST_INITIAL,    DoSetError)                ;TOKEN_SIGN_POSITIVE
1375        %TRANSITION(ST_INITIAL, DoSetError)                  ;TOKEN_SIGN_NEGATIVE
1376        %TRANSITION(ST_REL_SPD_DIGIT, DoAddDigit)           ;TOKEN_DIGIT
1377        %TRANSITION(ST_REL_SPD_DIGIT, DoNOP)                ;TOKEN_IGNORE
1378        %TRANSITION(ST_INITIAL, DoSetRelMotorSpeed)          ;TOKEN_RETURN
1379        %TRANSITION(ST_INITIAL, DoSetError)                  ;TOKEN_OTHER
1380
1381
1382
1383        ;Current State = ST_DIR                               Input Token Type
1384        %TRANSITION(ST_INITIAL, DoSetError)                  ;TOKEN_SPEED
1385        %TRANSITION(ST_INITIAL,    DoSetError)                ;TOKEN_RELATIVE_SPEED
1386        %TRANSITION(ST_INITIAL, DoSetError)                  ;TOKEN_DIRECTION
1387        %TRANSITION(ST_INITIAL,    DoSetError)                ;TOKEN_LASER_ON
1388        %TRANSITION(ST_INITIAL,    DoSetError)                ;TOKEN_LASER_OFF
1389        %TRANSITION(ST_INITIAL, DoSetError)                  ;TOKEN_TURRET_ANGLE
1390        %TRANSITION(ST_INITIAL, DoSetError)                  ;TOKEN_TURRET_ELEVATION
1391        %TRANSITION(ST_DIR_SIGN, DoNOP)                      ;TOKEN_SIGN_POSITIVE
1392        %TRANSITION(ST_DIR_SIGN, DoSetNegSign)               ;TOKEN_SIGN_NEGATIVE
```

```
1393        %TRANSITION(ST_DIR_DIGIT, DoAddDigit)             ;TOKEN_DIGIT
1394        %TRANSITION(ST_DIR, DoNOP)                         ;TOKEN_IGNORE
1395        %TRANSITION(ST_INITIAL, DoSetError)                  ;TOKEN_RETURN
1396        %TRANSITION(ST_INITIAL, DoSetError)                 ;TOKEN_OTHER
1397
1398
1399        ;Current State = ST_DIR_SIGN                      Input Token Type
1400        %TRANSITION(ST_INITIAL, DoSetError)                 ;TOKEN_SPEED
1401        %TRANSITION(ST_INITIAL,    DoSetError)                 ;TOKEN_RELATIVE_SPEED
1402        %TRANSITION(ST_INITIAL, DoSetError)                 ;TOKEN_DIRECTION
1403        %TRANSITION(ST_INITIAL,    DoSetError)                 ;TOKEN_LASER_ON
1404        %TRANSITION(ST_INITIAL,    DoSetError)                 ;TOKEN_LASER_OFF
1405        %TRANSITION(ST_INITIAL, DoSetError)                 ;TOKEN_TURRET_ANGLE
1406        %TRANSITION(ST_INITIAL, DoSetError)                 ;TOKEN_TURRET_ELEVATION
1407        %TRANSITION(ST_INITIAL, DoSetError)                 ;TOKEN_SIGN_POSITIVE
1408        %TRANSITION(ST_INITIAL, DoSetError)                 ;TOKEN_SIGN_NEGATIVE
1409        %TRANSITION(ST_DIR_DIGIT, DoAddDigit)             ;TOKEN_DIGIT
1410        %TRANSITION(ST_DIR_SIGN, DoNOP)                     ;TOKEN_IGNORE
1411        %TRANSITION(ST_INITIAL, DoSetError)                   ;TOKEN_RETURN
1412        %TRANSITION(ST_INITIAL, DoSetError)                 ;TOKEN_OTHER
1413
1414
1415
1416        ;Current State = ST_DIR_DIGIT                      Input Token Type
1417        %TRANSITION(ST_INITIAL, DoSetError)                 ;TOKEN_SPEED
1418        %TRANSITION(ST_INITIAL,    DoSetError)                 ;TOKEN_RELATIVE_SPEED
1419        %TRANSITION(ST_INITIAL, DoSetError)                 ;TOKEN_DIRECTION
1420        %TRANSITION(ST_INITIAL,    DoSetError)                 ;TOKEN_LASER_ON
1421        %TRANSITION(ST_INITIAL,    DoSetError)                 ;TOKEN_LASER_OFF
1422        %TRANSITION(ST_INITIAL, DoSetError)                 ;TOKEN_TURRET_ANGLE
1423        %TRANSITION(ST_INITIAL, DoSetError)                 ;TOKEN_TURRET_ELEVATION
1424        %TRANSITION(ST_INITIAL, DoSetError)                 ;TOKEN_SIGN_POSITIVE
1425        %TRANSITION(ST_INITIAL, DoSetError)                 ;TOKEN_SIGN_NEGATIVE
1426        %TRANSITION(ST_DIR_DIGIT, DoAddDigit)             ;TOKEN_DIGIT
1427        %TRANSITION(ST_DIR_DIGIT, DoNOP)                  ;TOKEN_IGNORE
1428        %TRANSITION(ST_INITIAL, DoSetMotorDirection)      ;TOKEN_RETURN
1429        %TRANSITION(ST_INITIAL, DoSetError)                 ;TOKEN_OTHER
1430
1431        ;Current State = ST_LSR_ON                          Input Token Type
1432        %TRANSITION(ST_INITIAL, DoSetError)                 ;TOKEN_SPEED
1433        %TRANSITION(ST_INITIAL,    DoSetError)                 ;TOKEN_RELATIVE_SPEED
1434        %TRANSITION(ST_INITIAL, DoSetError)                 ;TOKEN_DIRECTION
1435        %TRANSITION(ST_INITIAL,    DoSetError)                 ;TOKEN_LASER_ON
1436        %TRANSITION(ST_INITIAL,    DoSetError)                 ;TOKEN_LASER_OFF
1437        %TRANSITION(ST_INITIAL, DoSetError)                 ;TOKEN_TURRET_ANGLE
1438        %TRANSITION(ST_INITIAL, DoSetError)                 ;TOKEN_TURRET_ELEVATION
1439        %TRANSITION(ST_INITIAL, DoSetError)                 ;TOKEN_SIGN_POSITIVE
1440        %TRANSITION(ST_INITIAL, DoSetError)                 ;TOKEN_SIGN_NEGATIVE
1441        %TRANSITION(ST_INITIAL, DoSetError)                 ;TOKEN_DIGIT
1442        %TRANSITION(ST_LSR_ON,  DoNOP)                     ;TOKEN_IGNORE
1443        %TRANSITION(ST_INITIAL, DoSetLaserOn)             ;TOKEN_RETURN
1444        %TRANSITION(ST_INITIAL, DoSetError)                 ;TOKEN_OTHER
1445
1446        ;Current State = ST_LSR_OFF                       Input Token Type
1447        %TRANSITION(ST_INITIAL, DoSetError)                 ;TOKEN_SPEED
1448        %TRANSITION(ST_INITIAL,    DoSetError)                 ;TOKEN_RELATIVE_SPEED
1449        %TRANSITION(ST_INITIAL, DoSetError)                 ;TOKEN_DIRECTION
1450        %TRANSITION(ST_INITIAL,    DoSetError)                 ;TOKEN_LASER_ON
1451        %TRANSITION(ST_INITIAL,    DoSetError)                 ;TOKEN_LASER_OFF
1452        %TRANSITION(ST_INITIAL, DoSetError)                  ;TOKEN_TURRET_ANGLE
1453        %TRANSITION(ST_INITIAL, DoSetError)                 ;TOKEN_TURRET_ELEVATION
1454        %TRANSITION(ST_INITIAL, DoSetError)                 ;TOKEN_SIGN_POSITIVE
1455        %TRANSITION(ST_INITIAL, DoSetError)                 ;TOKEN_SIGN_NEGATIVE
1456        %TRANSITION(ST_INITIAL, DoSetError)                 ;TOKEN_DIGIT
1457        %TRANSITION(ST_LSR_OFF,  DoNOP)                     ;TOKEN_IGNORE
1458        %TRANSITION(ST_INITIAL, DoSetLaserOff)             ;TOKEN_RETURN
```

```
1459        %TRANSITION(ST_INITIAL, DoSetError)                ;TOKEN_OTHER
1460
1461        ;Current State = ST_TURRET_ANGLE                  Input Token Type
1462        %TRANSITION(ST_INITIAL, DoSetError)                ;TOKEN_SPEED
1463        %TRANSITION(ST_INITIAL,    DoSetError)                ;TOKEN_RELATIVE_SPEED
1464        %TRANSITION(ST_INITIAL, DoSetError)                ;TOKEN_DIRECTION
1465        %TRANSITION(ST_INITIAL,    DoSetError)                ;TOKEN_LASER_ON
1466        %TRANSITION(ST_INITIAL,    DoSetError)                ;TOKEN_LASER_OFF
1467        %TRANSITION(ST_INITIAL, DoSetError)                ;TOKEN_TURRET_ANGLE
1468        %TRANSITION(ST_INITIAL, DoSetError)                ;TOKEN_TURRET_ELEVATION
1469        %TRANSITION(ST_TURRET_ANGLE_SIGN, DoNOP)          ;TOKEN_SIGN_POSITIVE
1470        %TRANSITION(ST_TURRET_ANGLE_SIGN, DoSetNegSign)    ;TOKEN_SIGN_NEGATIVE
1471        %TRANSITION(ST_TURRET_ABS_DIGIT, DoAddDigit)      ;TOKEN_DIGIT
1472        %TRANSITION(ST_TURRET_ANGLE,  DoNOP)              ;TOKEN_IGNORE
1473        %TRANSITION(ST_INITIAL, DoSetError)                ;TOKEN_RETURN
1474        %TRANSITION(ST_INITIAL, DoSetError)                ;TOKEN_OTHER
1475
1476
1477        ;Current State = ST_TURRET_ABS_DIGIT               Input Token Type
1478        %TRANSITION(ST_INITIAL, DoSetError)                    ;TOKEN_SPEED
1479        %TRANSITION(ST_INITIAL,    DoSetError)                ;TOKEN_RELATIVE_SPEED
1480        %TRANSITION(ST_INITIAL, DoSetError)                ;TOKEN_DIRECTION
1481        %TRANSITION(ST_INITIAL,    DoSetError)                 ;TOKEN_LASER_ON
1482        %TRANSITION(ST_INITIAL,    DoSetError)                ;TOKEN_LASER_OFF
1483        %TRANSITION(ST_INITIAL, DoSetError)                ;TOKEN_TURRET_ANGLE
1484        %TRANSITION(ST_INITIAL, DoSetError)                ;TOKEN_TURRET_ELEVATION
1485        %TRANSITION(ST_INITIAL, DoSetError)                ;TOKEN_SIGN_POSITIVE
1486        %TRANSITION(ST_INITIAL, DoSetError)                ;TOKEN_SIGN_NEGATIVE
1487        %TRANSITION(ST_TURRET_ABS_DIGIT, DoAddDigit)      ;TOKEN_DIGIT
1488        %TRANSITION(ST_TURRET_ABS_DIGIT,  DoNOP)          ;TOKEN_IGNORE
1489        %TRANSITION(ST_INITIAL, DoSetAbsTurretAngle)      ;TOKEN_RETURN
1490        %TRANSITION(ST_INITIAL, DoSetError)                ;TOKEN_OTHER
1491
1492
1493        ;Current State = ST_TURRET_ANGLE_SIGN               Input Token Type
1494        %TRANSITION(ST_INITIAL, DoSetError)                ;TOKEN_SPEED
1495        %TRANSITION(ST_INITIAL,    DoSetError)                ;TOKEN_RELATIVE_SPEED
1496        %TRANSITION(ST_INITIAL, DoSetError)                ;TOKEN_DIRECTION
1497        %TRANSITION(ST_INITIAL,    DoSetError)                ;TOKEN_LASER_ON
1498        %TRANSITION(ST_INITIAL,    DoSetError)                ;TOKEN_LASER_OFF
1499        %TRANSITION(ST_INITIAL, DoSetError)                ;TOKEN_TURRET_ANGLE
1500        %TRANSITION(ST_INITIAL, DoSetError)                ;TOKEN_TURRET_ELEVATION
1501        %TRANSITION(ST_INITIAL, DoSetError)                ;TOKEN_SIGN_POSITIVE
1502        %TRANSITION(ST_INITIAL, DoSetError)                ;TOKEN_SIGN_NEGATIVE
1503        %TRANSITION(ST_TURRET_REL_DIGIT, DoAddDigit)      ;TOKEN_DIGIT
1504        %TRANSITION(ST_TURRET_ANGLE_SIGN, DoNOP)          ;TOKEN_IGNORE
1505        %TRANSITION(ST_INITIAL, DoSetError)                ;TOKEN_RETURN
1506        %TRANSITION(ST_INITIAL, DoSetError)                ;TOKEN_OTHER
1507
1508
1509        ;Current State = ST_TURRET_REL_DIGIT               Input Token Type
1510        %TRANSITION(ST_INITIAL, DoSetError)                ;TOKEN_SPEED
1511        %TRANSITION(ST_INITIAL,    DoSetError)                ;TOKEN_RELATIVE_SPEED
1512        %TRANSITION(ST_INITIAL, DoSetError)                ;TOKEN_DIRECTION
1513        %TRANSITION(ST_INITIAL,    DoSetError)                ;TOKEN_LASER_ON
1514        %TRANSITION(ST_INITIAL,    DoSetError)                ;TOKEN_LASER_OFF
1515        %TRANSITION(ST_INITIAL, DoSetError)                ;TOKEN_TURRET_ANGLE
1516        %TRANSITION(ST_INITIAL, DoSetError)                ;TOKEN_TURRET_ELEVATION
1517        %TRANSITION(ST_INITIAL, DoSetError)                ;TOKEN_SIGN_POSITIVE
1518        %TRANSITION(ST_INITIAL, DoSetError)                ;TOKEN_SIGN_NEGATIVE
1519        %TRANSITION(ST_TURRET_REL_DIGIT, DoAddDigit)      ;TOKEN_DIGIT
1520        %TRANSITION(ST_TURRET_REL_DIGIT, DoNOP)            ;TOKEN_IGNORE
1521        %TRANSITION(ST_INITIAL, DoSetRelTurretAngle)      ;TOKEN_RETURN
1522        %TRANSITION(ST_INITIAL, DoSetError)                ;TOKEN_OTHER
1523
1524
```

```
1525
1526        ;Current State = ST_TURRET_ELEVATION             Input Token Type
1527        %TRANSITION(ST_INITIAL, DoSetError)               ;TOKEN_SPEED
1528        %TRANSITION(ST_INITIAL,    DoSetError)               ;TOKEN_RELATIVE_SPEED
1529        %TRANSITION(ST_INITIAL, DoSetError)               ;TOKEN_DIRECTION
1530        %TRANSITION(ST_INITIAL,    DoSetError)               ;TOKEN_LASER_ON
1531        %TRANSITION(ST_INITIAL,    DoSetError)               ;TOKEN_LASER_OFF
1532        %TRANSITION(ST_INITIAL, DoSetError)               ;TOKEN_TURRET_ANGLE
1533        %TRANSITION(ST_INITIAL, DoSetError)               ;TOKEN_TURRET_ELEVATION
1534        %TRANSITION(ST_TURRET_ELEV_SIGN, DoNOP)           ;TOKEN_SIGN_POSITIVE
1535        %TRANSITION(ST_TURRET_ELEV_SIGN, DoSetNegSign)    ;TOKEN_SIGN_NEGATIVE
1536        %TRANSITION(ST_TURRET_ELEV_DIGIT, DoAddDigit)     ;TOKEN_DIGIT
1537        %TRANSITION(ST_TURRET_ELEVATION, DoNOP)           ;TOKEN_IGNORE
1538        %TRANSITION(ST_INITIAL, DoSetError)               ;TOKEN_RETURN
1539        %TRANSITION(ST_INITIAL, DoSetError)               ;TOKEN_OTHER
1540
1541
1542        ;Current State = ST_TURRET_ELEV_SIGN              Input Token Type
1543        %TRANSITION(ST_INITIAL, DoSetError)               ;TOKEN_SPEED
1544        %TRANSITION(ST_INITIAL,    DoSetError)               ;TOKEN_RELATIVE_SPEED
1545        %TRANSITION(ST_INITIAL, DoSetError)               ;TOKEN_DIRECTION
1546        %TRANSITION(ST_INITIAL,    DoSetError)               ;TOKEN_LASER_ON
1547        %TRANSITION(ST_INITIAL,    DoSetError)               ;TOKEN_LASER_OFF
1548        %TRANSITION(ST_INITIAL, DoSetError)               ;TOKEN_TURRET_ANGLE
1549        %TRANSITION(ST_INITIAL, DoSetError)               ;TOKEN_TURRET_ELEVATION
1550        %TRANSITION(ST_INITIAL, DoSetError)               ;TOKEN_SIGN_POSITIVE
1551        %TRANSITION(ST_INITIAL, DoSetError)               ;TOKEN_SIGN_NEGATIVE
1552        %TRANSITION(ST_TURRET_ELEV_DIGIT, DoAddDigit)     ;TOKEN_DIGIT
1553        %TRANSITION(ST_TURRET_ELEV_SIGN, DoNOP)           ;TOKEN_IGNORE
1554        %TRANSITION(ST_INITIAL, DoSetError)               ;TOKEN_RETURN
1555        %TRANSITION(ST_INITIAL, DoSetError)               ;TOKEN_OTHER
1556
1557        ;Current State = ST_TURRET_ELEV_DIGIT             Input Token Type
1558        %TRANSITION(ST_INITIAL, DoSetError)               ;TOKEN_SPEED
1559        %TRANSITION(ST_INITIAL,    DoSetError)               ;TOKEN_RELATIVE_SPEED
1560        %TRANSITION(ST_INITIAL, DoSetError)               ;TOKEN_DIRECTION
1561        %TRANSITION(ST_INITIAL,    DoSetError)               ;TOKEN_LASER_ON
1562        %TRANSITION(ST_INITIAL,    DoSetError)               ;TOKEN_LASER_OFF
1563        %TRANSITION(ST_INITIAL, DoSetError)               ;TOKEN_TURRET_ANGLE
1564        %TRANSITION(ST_INITIAL, DoSetError)               ;TOKEN_TURRET_ELEVATION
1565        %TRANSITION(ST_INITIAL, DoSetError)               ;TOKEN_SIGN_POSITIVE
1566        %TRANSITION(ST_INITIAL, DoSetError)               ;TOKEN_SIGN_NEGATIVE
1567        %TRANSITION(ST_TURRET_ELEV_DIGIT, DoAddDigit)     ;TOKEN_DIGIT
1568        %TRANSITION(ST_TURRET_ELEV_DIGIT, DoNOP)          ;TOKEN_IGNORE
1569        %TRANSITION(ST_INITIAL, DoSetTurretElevation)     ;TOKEN_RETURN
1570        %TRANSITION(ST_INITIAL, DoSetError)               ;TOKEN_OTHER
1571
1572
1573    ; Token Tables
1574    ;
1575    ; Description:
1576    ;       This creates the tables of token types and token values.
1577    ;      Each entry corresponds to the token type and the token value for a character.
1578    ;       Macros are used to actually build two separate tables - TokenTypeTable for token
1579    ;       types and TokenValueTable for token values.
1580    ; Notes:
1581    ;       This table is declared PRIVATE to prevent other codes accessing the table.
1582    ;       Also, READ ONLY tables should always be in the code segment so that in a
    standalone
1583    ;       system it will be located in the ROM with the code.
1584    ; Author:
1585    ;       Sunghoon Choi
1586    ; Revision history:
1587    ;      11/24/2016   Sunghoon Choi   Created
1588    ;       11/25/2016    Sunghoon Choi    Initial Compilation
1589    ;       11/26/2016    Sunghoon Choi    Revised Comments
```

```
1590
1591   %*DEFINE(TABLE)  (
1592           %TABENT(TOKEN_OTHER, 0)              ;<null>
1593           %TABENT(TOKEN_OTHER, 1)              ;SOH
1594           %TABENT(TOKEN_OTHER, 2)              ;STX
1595           %TABENT(TOKEN_OTHER, 3)              ;ETX
1596           %TABENT(TOKEN_OTHER, 4)              ;EOT
1597           %TABENT(TOKEN_OTHER, 5)              ;ENQ
1598           %TABENT(TOKEN_OTHER, 6)              ;ACK
1599           %TABENT(TOKEN_OTHER, 7)              ;BEL
1600           %TABENT(TOKEN_OTHER, 8)              ;backspace
1601           %TABENT(TOKEN_IGNORE, 9)            ;TAB (TOKEN_IGNORE)
1602           %TABENT(TOKEN_OTHER, 10)            ;new line
1603           %TABENT(TOKEN_OTHER, 11)            ;vertical tab
1604           %TABENT(TOKEN_OTHER, 12)            ;form feed
1605           %TABENT(TOKEN_RETURN, 13)            ;carriage return
1606           %TABENT(TOKEN_OTHER, 14)            ;SO
1607           %TABENT(TOKEN_OTHER, 15)            ;SI
1608           %TABENT(TOKEN_OTHER, 16)            ;DLE
1609           %TABENT(TOKEN_OTHER, 17)            ;DC1
1610           %TABENT(TOKEN_OTHER, 18)            ;DC2
1611           %TABENT(TOKEN_OTHER, 19)            ;DC3
1612           %TABENT(TOKEN_OTHER, 20)            ;DC4
1613           %TABENT(TOKEN_OTHER, 21)            ;NAK
1614           %TABENT(TOKEN_OTHER, 22)            ;SYN
1615           %TABENT(TOKEN_OTHER, 23)            ;ETB
1616           %TABENT(TOKEN_OTHER, 24)            ;CAN
1617           %TABENT(TOKEN_OTHER, 25)            ;EM
1618           %TABENT(TOKEN_OTHER, 26)            ;SUB
1619           %TABENT(TOKEN_OTHER, 27)            ;escape
1620           %TABENT(TOKEN_OTHER, 28)            ;FS
1621           %TABENT(TOKEN_OTHER, 29)            ;GS
1622           %TABENT(TOKEN_OTHER, 30)            ;AS
1623           %TABENT(TOKEN_OTHER, 31)            ;US
1624           %TABENT(TOKEN_IGNORE, ' ')           ;space
1625           %TABENT(TOKEN_OTHER, '!')            ;!
1626           %TABENT(TOKEN_OTHER, '"')            ;"
1627           %TABENT(TOKEN_OTHER, '#')            ;#
1628           %TABENT(TOKEN_OTHER, '$')            ;$
1629           %TABENT(TOKEN_OTHER, 37)            ;percent
1630           %TABENT(TOKEN_OTHER, '&')            ;&
1631           %TABENT(TOKEN_OTHER, 39)            ;'
1632           %TABENT(TOKEN_OTHER, 40)            ;open paren
1633           %TABENT(TOKEN_OTHER, 41)            ;close paren
1634           %TABENT(TOKEN_OTHER, '*')            ;*
1635           %TABENT(TOKEN_SIGN_POSITIVE, +1)      ;+ (positive sign)
1636           %TABENT(TOKEN_OTHER, 44)            ;,
1637           %TABENT(TOKEN_SIGN_NEGATIVE, -1)      ;- (negative sign)
1638           %TABENT(TOKEN_OTHER, '.')            ;. (decimal point)
1639           %TABENT(TOKEN_OTHER, '/')            ;/
1640           %TABENT(TOKEN_DIGIT, 0)              ;0 (digit)
1641           %TABENT(TOKEN_DIGIT, 1)              ;1 (digit)
1642           %TABENT(TOKEN_DIGIT, 2)              ;2 (digit)
1643           %TABENT(TOKEN_DIGIT, 3)              ;3 (digit)
1644           %TABENT(TOKEN_DIGIT, 4)              ;4 (digit)
1645           %TABENT(TOKEN_DIGIT, 5)              ;5 (digit)
1646           %TABENT(TOKEN_DIGIT, 6)              ;6 (digit)
1647           %TABENT(TOKEN_DIGIT, 7)              ;7 (digit)
1648           %TABENT(TOKEN_DIGIT, 8)              ;8 (digit)
1649           %TABENT(TOKEN_DIGIT, 9)              ;9 (digit)
1650           %TABENT(TOKEN_OTHER, ':')            ;:
1651           %TABENT(TOKEN_OTHER, ';')            ;;
1652           %TABENT(TOKEN_OTHER, '<')            ;<
1653           %TABENT(TOKEN_OTHER, '=')            ;=
1654           %TABENT(TOKEN_OTHER, '>')            ;>
1655           %TABENT(TOKEN_OTHER, '?')            ;?
```

```
1656            %TABENT(TOKEN_OTHER, '@')               ;@
1657            %TABENT(TOKEN_OTHER, 'A')               ;A
1658            %TABENT(TOKEN_OTHER, 'B')               ;B
1659            %TABENT(TOKEN_OTHER, 'C')              ;C
1660            %TABENT(TOKEN_DIRECTION, 'D')           ;D (Set Direction)
1661            %TABENT(TOKEN_TURRET_ELEVATION, 0)  ;E (Set Turret Elevation)
1662            %TABENT(TOKEN_LASER_ON, 'F')         ;F (Turn Laser On)
1663            %TABENT(TOKEN_OTHER, 'G')            ;G
1664            %TABENT(TOKEN_OTHER, 'H')            ;H
1665            %TABENT(TOKEN_OTHER, 'I')            ;I
1666            %TABENT(TOKEN_OTHER, 'J')            ;J
1667            %TABENT(TOKEN_OTHER, 'K')            ;K
1668            %TABENT(TOKEN_OTHER, 'L')            ;L
1669            %TABENT(TOKEN_OTHER, 'M')            ;M
1670            %TABENT(TOKEN_OTHER, 'N')            ;N
1671            %TABENT(TOKEN_LASER_OFF, 'O')        ;O (Turn Laser Off)
1672            %TABENT(TOKEN_OTHER, 'P')            ;P
1673            %TABENT(TOKEN_OTHER, 'Q')            ;Q
1674            %TABENT(TOKEN_OTHER, 'R')            ;R
1675            %TABENT(TOKEN_SPEED, 'S')            ;S (Set Absolute Speed)
1676            %TABENT(TOKEN_TURRET_ANGLE, 'T')    ;T (Set Turret Angle)
1677            %TABENT(TOKEN_OTHER, 'U')            ;U
1678            %TABENT(TOKEN_RELATIVE_SPEED, 'V')    ;V (Set Relative Speed)
1679            %TABENT(TOKEN_OTHER, 'W')            ;W
1680            %TABENT(TOKEN_OTHER, 'X')            ;X
1681            %TABENT(TOKEN_OTHER, 'Y')            ;Y
1682            %TABENT(TOKEN_OTHER, 'Z')            ;Z
1683            %TABENT(TOKEN_OTHER, '[')            ;[
1684            %TABENT(TOKEN_OTHER, '\')            ;\
1685            %TABENT(TOKEN_OTHER, ']')            ;]
1686            %TABENT(TOKEN_OTHER, '^')            ;^
1687            %TABENT(TOKEN_OTHER, '_')            ;_
1688            %TABENT(TOKEN_OTHER, '`')            ;`
1689            %TABENT(TOKEN_OTHER, 'a')            ;a
1690            %TABENT(TOKEN_OTHER, 'b')            ;b
1691            %TABENT(TOKEN_OTHER, 'c')            ;c
1692            %TABENT(TOKEN_DIRECTION, 'd')        ;d (Set Direction)
1693            %TABENT(TOKEN_TURRET_ELEVATION, 'e');e (Set Turret Elevation)
1694            %TABENT(TOKEN_LASER_ON, 'f')         ;f (Turn Laser On)
1695            %TABENT(TOKEN_OTHER, 'g')            ;g
1696            %TABENT(TOKEN_OTHER, 'h')            ;h
1697            %TABENT(TOKEN_OTHER, 'i')            ;i
1698            %TABENT(TOKEN_OTHER, 'j')            ;j
1699            %TABENT(TOKEN_OTHER, 'k')            ;k
1700            %TABENT(TOKEN_OTHER, 'l')            ;l
1701            %TABENT(TOKEN_OTHER, 'm')            ;m
1702            %TABENT(TOKEN_OTHER, 'n')            ;n
1703            %TABENT(TOKEN_LASER_OFF, 'o')        ;o (Turn Laser Off)
1704            %TABENT(TOKEN_OTHER, 'p')            ;p
1705            %TABENT(TOKEN_OTHER, 'q')            ;q
1706            %TABENT(TOKEN_OTHER, 'r')            ;r
1707            %TABENT(TOKEN_SPEED, 's')            ;s (Set Absolute Speed)
1708            %TABENT(TOKEN_TURRET_ANGLE, 't')    ;t (Set Turret Angle)
1709            %TABENT(TOKEN_OTHER, 'u')            ;u
1710            %TABENT(TOKEN_RELATIVE_SPEED, 'v')    ;v (Set Relative Speed)
1711            %TABENT(TOKEN_OTHER, 'w')            ;w
1712            %TABENT(TOKEN_OTHER, 'x')            ;x
1713            %TABENT(TOKEN_OTHER, 'y')            ;y
1714            %TABENT(TOKEN_OTHER, 'z')            ;z
1715            %TABENT(TOKEN_OTHER, '{')            ;{
1716            %TABENT(TOKEN_OTHER, '|')            ;|
1717            %TABENT(TOKEN_OTHER, '}')            ;}
1718            %TABENT(TOKEN_OTHER, '~')            ;~
1719            %TABENT(TOKEN_OTHER, 127)            ;rubout
1720    )
1721
```

```
1722  ; token type table - uses first byte of macro table entry
1723  %*DEFINE(TABENT(tokentype, tokenvalue))  (
1724          DB      %tokentype
1725  )
1726
1727  TokenTypeTable    LABEL   BYTE
1728          %TABLE
1729
1730
1731  ; token value table - uses second byte of macro table entry
1732  %*DEFINE(TABENT(tokentype, tokenvalue))  (
1733          DB      %tokenvalue
1734  )
1735
1736  TokenValueTable    LABEL       BYTE
1737          %TABLE
1738
1739  CODE ENDS
1740
1741  DATA SEGMENT PUBLIC 'DATA'
1742      ArgNum        DW      ?    ;The argument value for the action functions
1743      ArgSign        DB        ? ;Sign of ArgNum
1744      ParserState DB        ?    ;Current state of the parsing state
         machine.
1745  DATA ENDS
1746
1747  END
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                               ;
;                                   Parser.inc                                  ;
;                                   Homework 8                                  ;
;                                  Sunghoon Choi                                ;
;                                                                               ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Description:
;  This file contains the definitions for the Parser.asm
;
; Revision History:
;    11/23/2016   Sunghoon Choi     Created
;    11/26/2016   Sunghoon Choi     Updated Documentation


;Argument constants
ARG_INIT        EQU 0    ;Initial value of ArgNum
ARG_POSITIVE    EQU 0    ;Indicates that ArgNum is positive.
ARG_NEGATIVE    EQU 1    ;Indicates that ArgNum is negative.


;Parsing error flag constants
PARSER_SUCCESS    EQU    0    ;Indicates that the action function was processed
successfully.
PARSER_ERROR      EQU    1    ;Indicates that there was an error processing the action
function.


;States
ST_INITIAL                EQU        0     ;Initial State
ST_SPD                    EQU        1     ;State for setitng speed
ST_SPD_SIGN               EQU        2     ;State for setting speed's sign
ST_SPD_DIGIT              EQU        3     ;State for setting speed's value
ST_REL_SPD                EQU        4     ;State for setting relative speed
ST_REL_SPD_SIGN           EQU        5     ;State for setting relative speed's sign
ST_REL_SPD_DIGIT          EQU        6     ;State for setting relative speed's value
ST_DIR                    EQU        7     ;State for setting direction
ST_DIR_SIGN               EQU        8     ;State for setting direction's sign
ST_DIR_DIGIT              EQU        9     ;State for setting direction's value
ST_LSR_ON                 EQU        10    ;State for turning laser on
ST_LSR_OFF                EQU        11    ;State for turning laser off
ST_TURRET_ANGLE           EQU        12    ;State for setting turret angle
ST_TURRET_ABS_DIGIT       EQU        13    ;State for setting turret angle's absolute
value
ST_TURRET_ANGLE_SIGN      EQU        14    ;State for setting turret angle's sign
ST_TURRET_REL_DIGIT       EQU        15    ;State for setting turret angle's relative
value
ST_TURRET_ELEVATION       EQU        16    ;State for setting turret's elevation
ST_TURRET_ELEV_SIGN       EQU        17    ;State for setting turret's elevation's sign
ST_TURRET_ELEV_DIGIT      EQU        18    ;State for setting turret's elevation's value


;Token types
TOKEN_SPEED               EQU        0     ;Command S
TOKEN_RELATIVE_SPEED      EQU        1     ;Command V
TOKEN_DIRECTION           EQU        2     ;Command D
TOKEN_LASER_ON            EQU        3     ;Command F
TOKEN_LASER_OFF           EQU        4     ;Command O
TOKEN_TURRET_ANGLE        EQU        5     ;Command T
TOKEN_TURRET_ELEVATION    EQU        6     ;Command E
TOKEN_SIGN_POSITIVE       EQU        7     ;Sign +
TOKEN_SIGN_NEGATIVE       EQU        8     ;Sign -
TOKEN_DIGIT               EQU        9     ;Digits 0~9
TOKEN_IGNORE              EQU        10    ;space bar
TOKEN_RETURN              EQU        11    ;Carriage Return
```

```
TOKEN_OTHER             EQU     12    ;Other charactrs

NUM_TOKEN_TYPES         EQU     13    ;Total number of token types

;Token mask
TOKEN_MASK              EQU     01111111B ;Mask off the highest bit of token


```

```
NAME REMOTE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                             ;
;                                  Remote.asm                                 ;
;                                  Homework 9                                 ;
;                                 Sunghoon Choi                               ;
;                                                                             ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Table of Contents
;   RemoteMain              -   Main function for Remote module
;   ResetRemote             -   It initializes(resets) all configurations and shared
;                               variables related to queues, display, keypad,
;                               serials, timers, and interrupts.
;   KeyInputHandler         -   It handles the inputs from keypad and send command to Motors
;                               module through serial.
;   SerialReceivedHandler   -   It displays the strings received from the motors module
;                               through serial channel.
;   SerialErrorHandler      -   It displays serial error message when a serial error occurs.
;
;   RemoteEventHandlerTable - The table of handling routines for each type of events
;   SystemFailMsg           - The string to be displayed when system failure occurs.
;   SerialErrorMsg          - The string to be displayed when serial error occurs.
;   KeyCommandTable         - The table of commands for each keys.
;
; Description:
;   This file contains the main function which initializes all hardware configurations
;   of the remote module and executes functionalities of the remote board.
;   It receives input from keypad and send an appropriate command to the Motors module.
;   Also, if it receives a string from the Motors module through serial channel, it
;   displays the received string on the LED display. If a serial error occurs, it displays
;   a serial error message on LED. If a system failure(critical error) occurs, it displays
;   system failure message on LED. Refer to the Functional Specification document to
;   see more detailed version of description.
;
; Input:    Keypad, Serial
; Output:   Display, Motors, Serial
;
; User Interface:   User can use the keypad to send desired commands to Motors module.
;                   Keypad(key indices)-----------------------------
;                   0   1   2   3
;                   4   5   6   7
;                   8   9   10  11
;                   12  13  14  15
;                   Commands(for each key)---------------------------
;                   0:S65534  1:S0  2:D+45 3:D-45 4:V+1000 5:V+5000 6:V-1000 7:V-5000
;                   8:T+45  9:T-45  10:E+30  11:E+60  12:E-30  13:E-60  14:F  15:O
;                   [S: Set Speed] [D:Set Direction] [V:Change velocity]
;                   [T:Set Turret direction] [E:Set Turret Elevation]
;                   [F: Turn laser on]  [O:Turn laser off]
;                   The 7-digit LED display shows the string received from Motors module.
;                   It displays error message if a system failure or serial error occurs.
;                   To see more detailed version of user interface, refer to the
;                   Functional Specification.
; Error Handling:   It displays error message on LED for serial errrors and system failure.
;
; Algorithms:       None.
; Data Structures:  None
;
; Known Bugs:       None.
; Limitations:      None.
```

```
;
; Revision History:
;     12/01/2016    Sunghoon Choi      Created
;     12/02/2016    Sunghoon Choi      Initial Compilation
;     12/02/2016    Sunghoon Choi      Updated documentation



CGROUP  GROUP   CODE
DGROUP  GROUP   DATA, STACK


$INCLUDE(general.inc) ;Include the .inc file which contains general constants
$INCLUDE(Queue.inc)   ;Include the .inc file which contains constants for Queue.asm
$INCLUDE(Events.inc)  ;Include the .inc file which contains the list of events for
RoboTrike
$INCLUDE(Remote.inc)  ;Include the .inc file which contains constatns for Remote.asm
$INCLUDE(Display.inc) ;Include the .inc file which contains constants for Display.asm
$INCLUDE(Parser.inc)  ;Include the .inc file which contains constatns for Parser.asm

CODE    SEGMENT PUBLIC 'CODE'


        ASSUME  CS:CGROUP, DS:DGROUP

        ;external function declarations
        EXTRN CheckSystemFail:NEAR  ;Import CheckSystemFail to check system failure
        EXTRN InitDisplay:NEAR      ;Import InitDisplay to initialize display routine
        EXTRN Display:NEAR          ;Import Display to output digits(characters) on LED.
        EXTRN InitKeypad:NEAR       ;Import InitKeypad to initialize keypad routine
        EXTRN InitEventQueue:NEAR   ;Import InitEventQueue to initialize the EventQueue.
        EXTRN InitSerial:NEAR       ;Import InitSerial to initialize serial
        communication.
        EXTRN InitTimer2:NEAR       ;Import InitTimer2 to initialize timer2 interrupts.
        EXTRN InstallTimer2Handler:NEAR ;Import InstallTimer2Handler to install the
                                        ;Timer2handler on the interrupt vector.
        EXTRN SerialPutString:NEAR  ;Import SerialPutString to send string through
        serial.
        EXTRN EnqueueEvent:NEAR     ;Import EnqueueEvent to enqueue an event to
        EventQueue
        EXTRN DequeueEvent:NEAR     ;Import DequeueEvent to dequeue an event from
        EventQueue
        EXTRN InitCS:NEAR           ;Initializes the chip select
        EXTRN ClrIRQVectors:NEAR    ;Installs IllegalEventHandler for all interrupt
        vectors.


; RemoteMain
;
; Description:
;   It first initializes(resets) all configurations and shared variables related to
queues,
;   display, keypad, serials, timers, and interrupts. Then, it dequeues an event from
;   EventQueue and call a proper handler for the event.
; Operation:
;   It calls ResetRemote to initialize all configurations for Remote routine.
;   Then, it checks the system failure. If the system failure has occurred, it displays
;   System Failure message on the LED and call ResetRemote to reset the configurations.
;   If the system failure has not occurred, it dequeues an event from the EventQueue
;   and uses the RemoteEventHandlerTable to jump to a proper handler of the dequeued
event.
;   After handling the event, it goes back to the beginning of the loop.
; Arguments:
;   None
; Return Value:
;   None
```

```
; Local Variables:
;   AH(Event Type)             -    The type of the event
;   AL(Event Value. Key Index) -    The value of the event
; Shared Variables:
;   None
; Global Variables:
;   None
; Input:
;   None
; Output:
;   None
; Error Handling:
;   It displays System Failure message on LED if system failure occurs.
;   It displays serial error message on LED if a serial error occurs.
; Algorithms:
;   None
; Data Structures:
;   None
; Registers Changed:
;   AX, BX, CX, DX, SI, Flags
; Limitations:
;   None
; Known bugs:
;   None
; Special Notes:
;   None
; Author:
;   Sunghoon Choi
; Revision History:
;     12/01/2016    Sunghoon Choi      Created
;     12/02/2016    Sunghoon Choi      Initial Compilation
;     12/02/2016    Sunghoon Choi      Updated documentation

START:

MAIN:
        MOV     AX, DGROUP                  ;initialize the stack pointer
        MOV     SS, AX
        MOV     SP, OFFSET(DGROUP:TopOfStack)

        MOV     AX, DGROUP                  ;initialize the data segment
        MOV     DS, AX

        CALL InitCS                         ;Initializes the chip select

        CALL ClrIRQVectors                  ;Installs IllegalEventHandler for all
                                            ;interrupt vectors.


        CALL ResetRemote                    ;Initializes EventQueue, Display, Keypad,
                                            ;Serial, Timer2, and shared variables for
                                            ;Remote routine.

        STI                                 ;Since we are done with configuring the
                                            ;interrupt settings, enable the interrupts
                                            ;to run RoboTrike.


CheckSystemFailure:                         ;The first thing to do is to check the
                                            ;critical error.
        CALL  CheckSystemFail               ;CALL CheckSystemFFail to check if a system
                                            ;failure has occurred.
        CMP   AX, TRUE                      ;Has system failure occurred?
        JNE   CheckEventQueueEmpty          ;No. Go check if the EventQueue is empty.
       ;JE    DisplaySystemFailure         ;Yes. Display the system failure message on
                                            ;LED digits.
```

```
186
187   DisplaySystemFailure:
188           MOV   AX, CS                        ;Since SystemFailMsg is in the CS segment,
189           MOV   ES, AX                        ;set CS=ES to call Display function.
190           MOV   SI, OFFSET(SystemFailMsg)     ;Set SI to the address of SystemFailMsg so
              that
191                                               ;Display function can output the string.
192           CALL  Display                       ;Call Display to display SystemFailMsg on LED
193           CALL  ResetRemote                   ;To handle the system failure, we have to
194                                               ;reset the remote module.
195           JMP   CheckSystemFailure            ;Now that we've handled the systemfailure,
              go
196                                               ;back and check system failure again.
197
198   CheckEventQueueEmpty:
199           CALL  DequeueEvent                  ;Dequeue an event from the EventQueue.
200           JC    CheckSystemFailure            ;If the EventQueue is empty, we cannot
              dequeue
201                                               ;an event. Thus, go back to the beginning of
202                                               ;the loop.
203          ;JNC   HandleRoboEvents              ;If the EventQueue is not empty, go handle
              the
204                                               ;dequeued event.
205
206   CheckRemoteEvents:
207           CMP AH, KEY_EVENT                   ;Is the event KEY_EVENT?
208           JE  HandleRemoteEvents              ;Yes, go to the RemoteEventHandlerTable to
209                                               ;handle it.
210
211           CMP AH, SERIAL_RECEIVED_EVENT       ;Is the event SERIAL_RECEIVED_EVENT?
212           JE  HandleRemoteEvents              ;Yes, go to the RemoteEventHandlerTable to
213                                               ;handle it.
214
215           CMP AH, SERIAL_ERROR_EVENT          ;Is the event SERIAL_ERROR_EVENT?
216           JE  HandleRemoteEvents              ;Yes, go to the RemoteEventHandlerTable to
217                                               ;handle it.
218          JNE  CheckSystemFailure              ;If the event is an invalid event,
219                                               ;go back to the beginning of the loop and
                                                  check
220                                               ;the system failure.
221
222   HandleRemoteEvents:
223          XOR  BX, BX                          ;Clear BX since we are going to update BL.
224          MOV  BL, AH                          ;BL = Event Type of the dequeued event.
225          SHL  BX, MULT_BY_2                   ;We use the event type as index for the
226                                               ;RemoteEventHandlerTable. Since it is an word
227                                               ;table, double the index.
228          CALL CS:RemoteEventHandlerTable[BX]  ;Using the RemoteEventHandlerTable, handle
229                                               ;the event by using an appropriate handler.
230          JMP CheckSystemFailure               ;SInce we're done with handling the current
231                                               ;event, go back to the beginning of the loop
232                                               ;and check the system failure.
233
234
235
236   ; RemoteEventHandlerTable
237   ;
238   ; Description:
239   ;       This is the jump table used for executing appropriate handlers for each type of
240   ;       events.
241   ; Notes:
242   ;       This table is declared PRIVATE to prevent other codes accessing the table.
243   ;       Also, READ ONLY tables should always be in the code segment so that in a
      standalone
244   ;       system it will be located in the ROM with the code.
245   ;
```

```
246   ; Author:           Sunghoon Choi
247   ; Revision history:   12/01/2016    Sunghoon Choi   Created
248   ;                     12/02/2016    Sunghoon Choi   Updated documentation
249   RemoteEventHandlerTable LABEL   WORD
250       DW  BLANK_EVENT                     ;No event is assigned for this event type.
251       DW  KeyInputHandler                 ;Go handle the key pressed event
252       DW  SerialReceivedHandler           ;Go handle the serial received event
253       DW  SerialErrorHandler              ;Go handle the serial error event.
254
255
256   ; SystemFailMsg
257   ;
258   ; Description:
259   ;       This is the string to be shown on LED displays when a system failure occurs.
260   ; Notes:
261   ;       This table is declared PRIVATE to prevent other codes accessing the table.
262   ;       Also, READ ONLY tables should always be in the code segment so that in a
      standalone
263   ;       system it will be located in the ROM with the code.
264   ;
265   ; Author:           Sunghoon Choi
266   ; Revision history:   12/01/2016    Sunghoon Choi   Created
267   ;                     12/02/2016    Sunghoon Choi   Updated documentation
268   SystemFailMsg   LABEL   BYTE
269       DB 'SYS_FAIL', 0
270
271
272   ; SerialErrorMsg
273   ;
274   ; Description:
275   ;       This is the string to be shown on LED displays when a serial error occurs.
276   ; Notes:
277   ;       This table is declared PRIVATE to prevent other codes accessing the table.
278   ;       Also, READ ONLY tables should always be in the code segment so that in a
      standalone
279   ;       system it will be located in the ROM with the code.
280   ;
281   ; Author:           Sunghoon Choi
282   ; Revision history:   12/01/2016    Sunghoon Choi   Created
283   ;                     12/02/2016    Sunghoon Choi   Updated documentation
284
285
286   SerialErrorMsg  LABEL   BYTE
287       DB  'SEri_Err', 0
288
289
290   ; KeyCommandTable
291   ;
292   ; Description:
293   ;       This is the table of commands assigned for each keys.
294   ;       The map of key indices:
295   ;       0   1   2   3
296   ;       4   5   6   7
297   ;       8   9   10  11
298   ;       12  13  14  15
299   ; Notes:
300   ;       This table is declared PRIVATE to prevent other codes accessing the table.
301   ;       Also, READ ONLY tables should always be in the code segment so that in a
      standalone
302   ;       system it will be located in the ROM with the code.
303   ;
304   ; Author:           Sunghoon Choi
305   ; Revision history:   12/01/2016    Sunghoon Choi   Created
306   ;                     12/02/2016    Sunghoon Choi   Updated documentation
307
308   KeyCommandTable  LABEL  BYTE
```

```
309   DB  'S65534',0,0            ;key 0
310   DB  'S0',0,0,0,0,0,0        ;key 1
311   DB  'D+45',0,0,0,0          ;key 2
312   DB  'D-45',0,0,0,0          ;key 3
313
314   DB  'V+1000',0,0            ;key 4
315   DB  'V+5000',0,0            ;key 5
316   DB  'V-1000',0,0            ;key 6
317   DB  'V-5000',0,0            ;key 7
318
319   DB  'T+45',0,0,0,0          ;key 8
320   DB  'T-45',0,0,0,0          ;key 9
321   DB  'E+30',0,0,0,0          ;key 10
322   DB  'E+60',0,0,0,0          ;key 11
323
324   DB  'E-30',0,0,0,0          ;key 12
325   DB  'E-60',0,0,0,0          ;key 13
326   DB  'F',0,0,0,0,0,0,0       ;key 14
327   DB  'O',0,0,0,0,0,0,0       ;key 15
328
329
330   ; ResetRemote
331   ;
332   ; Description:
333   ;   It initializes(resets) all configurations and shared variables related to queues,
334   ;   display, keypad, serials, timers, and interrupts. Also, it initializes
335   ;   the RemoteDisplayIndex.
336   ; Operation:
337   ;   It calls InitEventQueue, InitDisplay, InitKeypad, InitSerial, InitTImer2,
338   ;   InstallTimer2Handler. Then, it initializes RemoteDisplayBufferIndex to
339   ;   DISPLAY_START_INDEX.
340   ; Arguments:
341   ;   None
342   ; Return Value:
343   ;   None
344   ; Local Variables:
345   ;   None
346   ; Shared Variables:
347   ;   RemoteDisplayBufferIndex    -   [Write] -   The index used to take a value from
348   ;                                                RemoteDisplayBuffer
349   ; Global Variables:
350   ;   None
351   ; Input:
352   ;   None
353   ; Output:
354   ;   None
355   ; Error Handling:
356   ;   None
357   ; Algorithms:
358   ;   None
359   ; Data Structures:
360   ;   None
361   ; Registers Changed:
362   ;   AX, BX, CX, DX, SI, Flags
363   ; Limitations:
364   ;   None
365   ; Known bugs:
366   ;   None
367   ; Special Notes:
368   ;   None
369   ; Author:
370   ;   Sunghoon Choi
371   ; Revision History:
372   ;     12/01/2016    Sunghoon Choi      Created
373   ;     12/02/2016    Sunghoon Choi      Initial Compilation
374   ;     12/02/2016    Sunghoon Choi      Updated documentation
```

```
ResetRemote         PROC    NEAR
                    PUBLIC  ResetRemote

    CALL    InitEventQueue          ;Initializes the EventQueue.

    CALL    InitDisplay             ;Initializes display routine

    CALL    InitKeypad              ;Initializes keypad routine

    CALL    InitSerial              ;Initializes serial communication.
                                    ;InitSerial includes InitINT2,
                                    InstallINT2EventHandler.

    CALL    InitTimer2              ;Initializes Timer2

    CALL    InstallTimer2Handler    ;Installs Timer2Handler on the interrupt vector.

    MOV     RemoteDisplayBufferIndex, DISPLAY_START_INDEX
                                    ;Reset RemoteDisplayBufferIndex.

    RET                             ;End of ResetRemote
ResetRemote         ENDP


; KeyInputHandler
;
; Description:
;   It uses the Event Value of Keypad event to find an appropriate command string in
;   KeypadCommandTable and send the string to Motor module through serial channel.
; Operation:
;   It multiplies the Event Value (= Key Value) with the length of each command string
to
;   find the index in KeypadCommandTable. Then, it adds the calculated index to the
offset
;   of KeypadCommandTable to find the exact address of the target command. Finally, it
;   calls SerialPutString to send the command string to Motor module through serial
channel.
; Arguments:
;   AH(Event Type)             -    The type of the event to be enqueued
;   AL(Event Value. Key Index) -    The value of the event to be enqueued.
;                                   Here, it means the index of the pressed key.
; Return Value:
;   None
; Local Variables:
;   KeyCommandAddr(SI)  -   The address of the command string for current key input.
; Shared Variables:
;   None
; Global Variables:
;   None
; Input:
;   None
; Output:
;   None
; Error Handling:
;   None
; Algorithms:
;   None
; Data Structures:
;   None
; Registers Changed:
;   AX, BX, DX, SI, Flags
; Limitations:
;   None
; Known bugs:
;   None
; Special Notes:
```

```
437   ;   None
438   ; Author:
439   ;   Sunghoon Choi
440   ; Revision History:
441   ;     12/01/2016    Sunghoon Choi      Created
442   ;     12/02/2016    Sunghoon Choi      Initial Compilation
443   ;     12/02/2016    Sunghoon Choi      Updated documentation
444
445   KeyInputHandler         PROC    NEAR
446                           PUBLIC  KeyInputHandler
447
448       XOR AH,AH                     ;AL = Key's index.
449                                     ;To multiply AX with COMMAND_LENGTH, we have to clear
                                        AH.
450       MOV BX, COMMAND_LENGTH        ;It multiplies Key_Index with COMMAND_LENGTH to find
          the
451                                     ;Key Command's index in KeyCommandTable.
452       XOR DX, DX                    ;Clear DX for multiplication.
453       MUL BX                        ;DX:AX = KeyCommand's Index inside KeyCommandTable
454
455       ADD AX, OFFSET(KeyCommandTable) ;KeyCommand's Address
456                                       ; = OFFSET(KeyCommandTable) + KeyCommand index in
                                          table
457
458       MOV SI, AX                    ;SI = KeyCommand's address
459       MOV AX, CS                    ;Since the key command table is in CS segment,
460       MOV ES, AX                    ;CS=ES must be performed to call SerialPutString.
461       CALL SerialPutString          ;Send the Key Command to Motor module through the
462                                     ;serial channel.
463
464       RET                           ;End of KeyInputHandler
465
466   KeyInputHandler     ENDP
467
468
469   ; SerialReceivedHandler
470   ;
471   ; Description:
472   ;   It displays the characters received from the serial channel on LED digits.
473   ; Operation:
474   ;   It first checks if the received character is Carriage Return. If it is, display the
475   ;   characters in RemoteDisplayBuffer by attaching NULL in the end of the string. After
476   ;   calling Display function to display the string, it resets RemoteDisplayIndex to
477   ;   the starting index of RemoteDisplayBuffer.
478   ;   If the received character is not the Carriage Return, it stores the received
      character in
479   ;   RemoteDisplayBuffer at RemoteDisplayIndex. Then, it increments RemoteDisplayIndex.
480   ;   If the incremented RemoteDisplayIndex is larger than or equal to the length of
481   ;   RemoteDisplayBuffer, fix RemoteDisplayIndex at DISPLAY_BUFFER_LEN-1 so that it can
482   ;   display the truncated characters when a carriage return is received later.
483   ; Arguments:
484   ;   AH(Event Type)  -   The type of the event to be enqueued (unused in this function)
485   ;   AL(Event Value, ReceivedCharacter) -    The value of the event to be enqueued
486   ; Return Value:
487   ;   None
488   ; Local Variables:
489   ;   ReceivedCharacter(AL)   -   The event value of SERIAL_RECEIVED_EVENT.
490   ;                               It will be enqueued to RemoteDisplayBuffer
491   ; Shared Variables:
492   ;   RemoteDisplayBuffer - [Write] - The buffer which contains the characters to be
      displayed
493   ;   RemoteDisplayIndex  - [R/W] -   The index for RemoteDisplayBuffer.
494   ; Global Variables:
495   ;   None
496   ; Input:
497   ;   None
```

```
; Output:
;   None
; Error Handling:
;   None
; Algorithms:
;   None
; Data Structures:
;   None
; Registers Changed:
;   AX, BX, SI, Flags
; Limitations:
;   None
; Known bugs:
;   None
; Special Notes:
;   None
; Author:
;   Sunghoon Choi
; Revision History:
;     12/01/2016    Sunghoon Choi      Created
;     12/02/2016    Sunghoon Choi      Initial Compilation
;     12/02/2016    Sunghoon Choi      Updated documentation

SerialReceivedHandler       PROC    NEAR
                            PUBLIC  SerialReceivedHandler

CheckCarriageReturn:
    CMP AL, CARRIAGE_RETURN         ;Is the character Carriage Return?
    JE  DisplayReceivedStr          ;Yes. Display the string in RemoteDisplayBuffer.
    ;JNE  UpdateDisplayBuffer       ;No. Store the current character in
    RemoteDisplayBuffer.
UpdateDisplayBuffer:
    MOV BX, RemoteDisplayBufferIndex ;Save RemoteDisplayBufferIndex in BX so that we can
                                     ;store the character in a proper index of
                                     ;RemoteDisplayBuffer.
    MOV RemoteDisplayBuffer[BX], AL  ;Save the character in RemoteDisplayBuffer.
IncrementBufferIndex:
    INC RemoteDisplayBufferIndex     ;Proceed to storing next character by incrementing
                                     ;RemoteDisplayBufferIndex.
    CMP RemoteDisplayBufferIndex, REMOTE_BUFFER_LEN-1
                                    ;Is RemoteDisplayBufferIndex >= REMOTE_BUFFER_LEN ?
    JNA EndSerialReceivedHandler     ;No. We still have rooms to store characters in
                                    ;RemoteDisplayBuffer.
    ;JA WaitUntilCR                 ;Yes. We we have to fix the index at the last
    character
                                    ;of RemoteDisplayBuffer and wait until it Carriage
                                    ;Return arrives.
WaitUntilCR:
    MOV RemoteDisplayBufferIndex, REMOTE_BUFFER_LEN-1
                                    ;Fix the index at the last index of
                                    RemoteDisplayBuffer
                                    ;and wait until it receives Carriage Return.
    JMP EndSerialReceivedHandler     ;Exit the procedure without displaying the string
                                    ;since we haven't received Carriage Return.
DisplayReceivedStr:
    MOV BX, RemoteDisplayBufferIndex ;Save RemoteDisplayBufferIndex in BX so that we can
                                     ;store the character in a proper index of
                                     ;RemoteDisplayBuffer.
    MOV RemoteDisplayBuffer[BX], 0   ;String must end with NULL.
    MOV AX, DS                       ;Since RemoteDisplayBuffer is in Data segment,
    MOV ES, AX                       ;we have to set DS = ES to call Display.
    MOV SI, OFFSET(RemoteDisplayBuffer) ;Set SI to the offset of RemoteDisplayBuffer
                                        ;to call display to print the string in
                                        ;RemoteDisplayBuffer.
    CALL Display                        ;Display the string stored in
    RemoteDisplayBuffer.
```

```
560         MOV RemoteDisplayBufferIndex, STARTING_INDEX
561                                             ;Since we displayed the string in
                                                RemoteDisplayBuffer,
562                                             ;reset the index to the beginning index of the
                                                buffer.
563     EndSerialReceivedHandler:
564         RET                                  ;End of SerialReceivedHandler.
565
566     SerialReceivedHandler    ENDP
567
568     ; SerialErrorHandler
569     ;
570     ; Description:
571     ;   It displays SerialErrorMsg on LED digits when received a serial error.
572     ; Operation:
573     ;   After setting SI to the offset of SerialErrorMsg, it calls Display function to
        display
574     ;   the serial error message on LED digits when a serial error is received.
575     ; Arguments:
576     ;   None
577     ; Return Value:
578     ;   None
579     ; Local Variables:
580     ;   None
581     ; Shared Variables:
582     ;   None
583     ; Global Variables:
584     ;   None
585     ; Input:
586     ;   None
587     ; Output:
588     ;   None
589     ; Error Handling:
590     ;   None
591     ; Algorithms:
592     ;   None
593     ; Data Structures:
594     ;   None
595     ; Registers Changed:
596     ;   AX, SI, Flags
597     ; Limitations:
598     ;   None
599     ; Known bugs:
600     ;   None
601     ; Special Notes:
602     ;   None
603     ; Author:
604     ;   Sunghoon Choi
605     ; Revision History:
606     ;     12/01/2016    Sunghoon Choi      Created
607     ;     12/02/2016    Sunghoon Choi      Initial Compilation
608     ;     12/02/2016    Sunghoon Choi      Updated documentation
609
610     SerialErrorHandler      PROC    NEAR
611                             PUBLIC  SerialErrorHandler
612
613     DisplaySerialErrorMsg:
614         MOV AX, CS                      ;Since SerialErrorMsg is in the code segment,
615         MOV ES, AX                      ;we have to set CS=ES to call Display.
616         MOV SI, OFFSET(SerialErrorMsg)  ;Set SI to the offset of SerialErrorMsg to call
617                                         ;Display to display the error message on LED digits.
618         CALL Display                    ;Call Display to display the error message on LED.
619
620     EndSerialErrorHandler:
621         RET                             ;End of SerialErrorhandler
622
```

```
623   SerialErrorHandler   ENDP
624
625
626   CODE ENDS
627
628
629
630
631
632
633   DATA SEGMENT PUBLIC 'DATA'
634
635       RemoteDisplayBufferIndex    DW  ?
636                                       ;The index of current character for
                                        RemoteDisplayBuffer.
637       RemoteDisplayBuffer         DB  REMOTE_BUFFER_LEN DUP (?)
638                                       ;The buffer which contains the characters to be
                                        displayed
639
640   DATA ENDS
641
642
643
644
645   STACK   SEGMENT STACK  'STACK'
646
647                   DB      80 DUP ('Stack ')       ;240 words
648
649   TopOfStack      LABEL   WORD
650
651   STACK   ENDS
652
653   END START
654
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                             ;
;                                  Remote.inc                                 ;
;                                  Homework 9                                 ;
;                                 Sunghoon Choi                               ;
;                                                                             ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Description:
;  This file contains the definitions for the constants of Remote.asm
;
; Revision History:
;     12/01/2016    Sunghoon Choi      Created
;     12/02/2016    Sunghoon Choi      Initial Compilation
;     12/02/2016    Sunghoon Choi      Updated documentation

COMMAND_LENGTH      EQU      8  ;Length of each RoboTrike motor commands
REMOTE_BUFFER_LEN   EQU      9  ;The length of RemoteDisplayBuffer

```

```
NAME MTRMAIN
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                              ;
;                                 MtrMain.asm                                  ;
;                                 Homework 10                                  ;
;                                Sunghoon Choi                                 ;
;                                                                              ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Table of Contents
;   MotorMain               - Main function for Motor Module
;   ResetMotorMain          - It initializes(resets) all configurations and shared variables
;                             related to Motors, Serial, queues, timers, and interrupts.
;   SerialReceivedHandler   - It parses the received command and send back the status of the
;                             motor module.
;   SerialErrorHandler      - It transmits the serial error message to the remote module
;                             when a serial error happens.
;
;   MotorEventHandlerTable  - The table of handling routines for each type of events
;   SystemFailMsg           - The string to be transmitted when system failure occurs
;   SerialErrMsg            - The string to be transmitted when serial error occurs.
;   ParserErrMsg            - The string to be transmitted when parser error occurs.
; Description:
;   This file contains the main function which initializes all hardware/software
;   configurations of motor module and executes functionalities of the motors board.
;   It receives commands from the remote moudle through serial channel and executes
;   appropriate actions. After executing an action, it sends back the information of the
;   changed status to the remote module through the serial channel. If a serial error occurs,
;   it sends a serial error message to the remote module so that it can display the message.
;   If a parser error occurs, it sends a parser error message to the remote module.
;   If a system failure occurs, it sends the system failure message to the remote module
;   and reset all variables and configurations of the motors module.
;   Refer to the Functional Specification document to see more detailed version of
;   description.
; Input:
;   Serial
; Output:
;   Motors, Serial
;
; User Interface:
;   The user can check the changed status of the motors module by reading
;   the LED digits on the remote module. All motors and laser are controlled
;   by the Remote module.
; Error Handling:
;   It transmits serial error message, parser error message, and system failure message to
;   the remote module when corresponding error occurs.
; Algorithms:       None.
; Data Structures:  None
;
; Known Bugs:       None.
; Limitations:      It does not have a feedback control.
;
; Revision History:
;     12/07/2016    Sunghoon Choi      Created
;     12/08/2016    Sunghoon Choi      Initial Compilation
;     12/08/2016    Sunghoon Choi      Updated documentation



CGROUP  GROUP   CODE
DGROUP  GROUP   DATA, STACK
```

```
62
63
64  $INCLUDE(general.inc) ;Include the .inc file which contains general constants
65  $INCLUDE(Queue.inc)   ;Include the .inc file which contains constants for Queue.asm
66  $INCLUDE(Events.inc)  ;Include the .inc file which contains the list of events for
    RoboTrike
67  $INCLUDE(MtrMain.inc) ;Include the .inc file which contains constatns for MtrMain.asm
68  $INCLUDE(Parser.inc)  ;Include the .inc file which contains constatns for Parser.asm
69
70  CODE    SEGMENT PUBLIC 'CODE'
71
72
73          ASSUME  CS:CGROUP, DS:DGROUP
74
75          ;external function declarations
76          EXTRN CheckSystemFail:NEAR      ;Import CheckSystemFail to check system failure
77          EXTRN InitEventQueue:NEAR       ;Import InitEventQueue to initialize the
            EventQueue.
78          EXTRN InitSerial:NEAR           ;Import InitSerial to initialize serial
            communication.
79          EXTRN InitParallelB:NEAR        ;Import InitParallelB to initialize parallel
            port B.
80          EXTRN InitMotorLaser:NEAR       ;Import InitMotorLaser to initialize motors.
81          EXTRN GetLaser:NEAR             ;Import GetLaser to get the laser status.
82          EXTRN GetMotorSpeed:NEAR        ;Import GetMotorSpeed to get the motor speed.
83          EXTRN GetMotorDirection:NEAR    ;Import getMotorDirection to get the current
            direction.
84          EXTRN ParseSerialChar:NEAR      ;Import ParseSerialChar to parse commands.
85          EXTRN Dec2String:NEAR           ;Import Dec2String to convert decimal numbers
            to strings
86          EXTRN UnsignedDec2String:NEAR   ;Import UnsignedDec2String to convert decimal
            numbers
87                                          ;to strings in unsigned format.
88          EXTRN InitParser:NEAR           ;Import InitParser to initialize parser routine.
89          EXTRN InitTimer1:NEAR           ;Import InitTimer1 to initialize timer1
            interrupts.
90          EXTRN InstallTimer1Handler:NEAR ;Import InstallTimer1Handler to install the
91                                          ;Timer1Handler on the interrupt vector.
92          EXTRN SerialPutString:NEAR      ;Import SerialPutString to send string through
            serial.
93          EXTRN EnqueueEvent:NEAR         ;Import EnqueueEvent to enqueue an event to
            EventQueue
94          EXTRN DequeueEvent:NEAR         ;Import DequeueEvent to dequeue an event from
            EventQueue
95          EXTRN InitCS:NEAR               ;Initializes the chip select
96          EXTRN ClrIRQVectors:NEAR        ;Installs IllegalEventHandler for all interrupt
            vectors.
97
98
99  ; MotorMain
100 ;
101 ; Description:
102 ;   It first initializes(resets) all configurations and shared variables related to
    queues,
103 ;   serials, motors, timers, and interrupts. Then, it dequeus an event from EventQueue
    and
104 ;   call a proper handler for the event.
105 ; Operation:
106 ;   It calls ResetMotorMain to initialize all configurations for Motors routine.
107 ;   Then, it checks the system failure. If the system failure has occurred, it
    transmits the
108 ;   system failure message to the Remote Module and call ResetMotorMain to reset the
109 ;   configurations. If the system failure has not occurred, it dequeues an event from
    the
110 ;   EventQueue and uses the MotorEventHandlerTable to jump to a proper handler of the
111 ;   dequeued event. After handling the event, it goes back to the beginning of the loop.
```

```
; Arguments:
;   None
; Return Value:
;   None
; Local Variables:
;   AH(Event Type)              -    The type of the event
;    AL(Event Value)            -    The value of the event
; Shared Variables:
;    None
; Global Variables:
;   None
; Input:
;   None
; Output:
;   None
; Error Handling:
;   It transmits System Failure message to the RemoteModule if the system failure occurs.
;   It transmits Serial Error message to the RemoteModule if a serial error occurs.
;   It transmits Parser Error message to the RemoteModule if a parser error occurs.
; Algorithms:
;   None
; Data Structures:
;   None
; Registers Changed:
;   AX, BX, CX, DX, SI, Flags
; Limitations:
;   None
; Known bugs:
;   None
; Special Notes:
;   None
; Author:
;   Sunghoon Choi
; Revision History:
;     12/07/2016    Sunghoon Choi      Created
;     12/08/2016    Sunghoon Choi      Initial Compilation
;     12/08/2016    Sunghoon Choi      Updated documentation

START:

MAIN:
        MOV     AX, DGROUP                  ;initialize the stack pointer
        MOV     SS, AX
        MOV     SP, OFFSET(DGROUP:TopOfStack)

        MOV     AX, DGROUP                  ;initialize the data segment
        MOV     DS, AX

        CALL InitCS                         ;Initializes the chip select

        CALL ClrIRQVectors                  ;Installs IllegalEventHandler for all
                                            ;interrupt vectors.

debugpoint1:
        CALL ResetMotorMain                 ;Initializes EventQueue, Serial, Parser,
        Motors,
                                            ;Timer1, INT2 and shared variables for
                                            ;Motor Module's main routine.
debugpoint2:
        STI                                 ;Since we are done with configuring the
                                            ;interrupt settings, enable the interrupts
                                            ;to run RoboTrike.


CheckSystemFailure:                         ;The first thing to do is to check the
                                            ;critical error.
```

```
177             CALL  CheckSystemFail                ;CALL CheckSystemFFail to check if a system
178                                                  ;failure has occurred.
179             CMP   AX, TRUE                       ;Has system failure occurred?
180             JNE   CheckEventQueueEmpty           ;No. Go check if the EventQueue is empty.
181            ;JE    SendSystemFailure             ;Yes. Display the system failure message on
182                                                  ;LED digits.
183
184   SendSystemFailure:
185             MOV   AX, CS                         ;Since SystemFailMsg is in the CS segment,
186             MOV   ES, AX                         ;set CS=ES to call SerialPutString
                function.
187             MOV   SI, OFFSET(SystemFailMsg)      ;Set SI to the address of SystemFailMsg
                so that
188                                                  ;SerialPutString function can output the
                                                     string.
189             CALL  SerialPutString                ;Call SerialPutString to send the system
190                                                  ;failure message to the remote module.
191             CALL  ResetMotorMain                 ;reset the remote module.
192
193             JMP   CheckSystemFailure             ;Now that we've handled the
                systemfailure, go
194                                                  ;back and check system failure again.
195
196   CheckEventQueueEmpty:
197             CALL  DequeueEvent                   ;Dequeue an event from the EventQueue.
198             JC    CheckSystemFailure             ;If the EventQueue is empty, we cannot
                dequeue
199                                                  ;an event. Thus, go back to the beginning
                                                     of
200                                                  ;the loop.
201            ;JNC   HandleRoboEvents               ;If the EventQueue is not empty, go
               handle the
202                                                  ;dequeued event.
203
204   CheckMotorsEvents:
205
206             CMP AH, SERIAL_RECEIVED_EVENT        ;Is the event SERIAL_RECEIVED_EVENT?
207             JE  HandleMotorsEvents               ;Yes, go to the MotorEventHandlerTable to
208                                                  ;handle it.
209
210             CMP AH, SERIAL_ERROR_EVENT           ;Is the event SERIAL_ERROR_EVENT?
211             JE  HandleMotorsEvents               ;Yes, go to the MotorEventHandlerTable to
212                                                  ;handle it.
213            JNE  CheckSystemFailure               ;If the event is an invalid event,
214                                                  ;go back to the beginning of the loop and
                                                     check
215                                                  ;the system failure.
216
217   HandleMotorsEvents:
218            XOR  BX, BX                           ;Clear BX since we are going to update BL.
219            MOV  BL, AH                           ;BL = Event Type of the dequeued event.
220            SHL  BX, MULT_BY_2                    ;We use the event type as index for the
221                                                  ;MotorEventHandlerTable. Since it is an
                                                     word
222                                                  ;table, double the index.
223            CALL CS:MotorEventHandlerTable[BX]    ;Using the MotorEventHandlerTable, handle
224                                                  ;the event by using an appropriate handler.
225            JMP CheckSystemFailure                ;Since we're done with handling the
               current
226                                                  ;event, go back to the beginning of the
                                                     loop
227                                                  ;and check the system failure.
228
229
230
231   ; MotorEventHandlerTable
```

```
232   ;
233   ; Description:
234   ;       This is the jump table used for executing appropriate handlers for each type of
235   ;       events.
236   ; Notes:
237   ;       This table is declared PRIVATE to prevent other codes accessing the table.
238   ;       Also, READ ONLY tables should always be in the code segment so that in a
      standalone
239   ;       system it will be located in the ROM with the code.
240   ;
241   ; Author:           Sunghoon Choi
242   ; Revision history:   12/01/2016    Sunghoon Choi    Created
243   ;                     12/02/2016    Sunghoon Choi    Updated documentation
244   MotorEventHandlerTable LABEL   WORD
245       DW  BLANK_EVENT                     ;No event is assigned for this event type.
246       DW  BLANK_EVENT                     ;No event is assigned for this event type
247       DW  SerialReceivedHandler           ;Go handle the serial received event
248       DW  SerialErrorHandler              ;Go handle the serial error event.
249
250
251   ; SystemFailMsg
252   ;
253   ; Description:
254   ;       This is the string to be shown on LED displays when a system failure occurs.
255   ; Notes:
256   ;       This table is declared PRIVATE to prevent other codes accessing the table.
257   ;       Also, READ ONLY tables should always be in the code segment so that in a
      standalone
258   ;       system it will be located in the ROM with the code.
259   ;
260   ; Author:           Sunghoon Choi
261   ; Revision history:   12/01/2016    Sunghoon Choi    Created
262   ;                     12/02/2016    Sunghoon Choi    Updated documentation
263   SystemFailMsg   LABEL   BYTE
264       DB 'SYS_FAIL', 0
265
266
267   ; SerialErrorMsg
268   ;
269   ; Description:
270   ;       This is the string to be shown on LED displays when a serial error occurs.
271   ; Notes:
272   ;       This table is declared PRIVATE to prevent other codes accessing the table.
273   ;       Also, READ ONLY tables should always be in the code segment so that in a
      standalone
274   ;       system it will be located in the ROM with the code.
275   ;
276   ; Author:           Sunghoon Choi
277   ; Revision history:   12/01/2016    Sunghoon Choi    Created
278   ;                     12/02/2016    Sunghoon Choi    Updated documentation
279
280
281   SerialErrorMsg  LABEL   BYTE
282       DB  'SEri_Err', 0
283
284
285   ; ParserErrorMsg
286   ;
287   ; Description:
288   ;       This is the string to be shown on LED displays when a parser error occurs.
289   ; Notes:
290   ;       This table is declared PRIVATE to prevent other codes accessing the table.
291   ;       Also, READ ONLY tables should always be in the code segment so that in a
      standalone
292   ;       system it will be located in the ROM with the code.
293   ;
```

```
; Author:           Sunghoon Choi
; Revision history:   12/01/2016    Sunghoon Choi    Created
;                     12/02/2016    Sunghoon Choi    Updated documentation


ParserErrorMsg  LABEL   BYTE
    DB  'PArS_Err', 0



; ResetMotorMain
;
; Description:
;     It initializes(resets) all configurations and shared variables related to queues,
;    display, keypad, serials, timers, and interrupts. Also, it initializes
;    the RemoteDisplayIndex.
; Operation:
;    It calls InitEventQueue, InitDisplay, InitKeypad, InitSerial, InitTImer2,
;    InstallTimer2Handler. Then, it initializes RemoteDisplayBufferIndex to
;    DISPLAY_START_INDEX.
; Arguments:
;   None
; Return Value:
;   None
; Local Variables:
;    None
; Shared Variables:
;    RemoteDisplayBufferIndex    -    [Write]   -  The index used to take a value from
;                                                  RemoteDisplayBuffer
; Global Variables:
;   None
; Input:
;   None
; Output:
;   None
; Error Handling:
;   None
; Algorithms:
;   None
; Data Structures:
;   None
; Registers Changed:
;   AX, BX, CX, DX, SI, Flags
; Limitations:
;   None
; Known bugs:
;   None
; Special Notes:
;   None
; Author:
;   Sunghoon Choi
; Revision History:
;     12/01/2016    Sunghoon Choi      Created
;     12/02/2016    Sunghoon Choi      Initial Compilation
;     12/02/2016    Sunghoon Choi      Updated documentation
ResetMotorMain      PROC    NEAR
                    PUBLIC  ResetMotorMain

    CALL    InitEventQueue          ;Initializes the EventQueue.


    CALL    InitSerial              ;Initializes serial communication.
                                    ;InitSerial includes InitINT2,
                                    InstallINT2EventHandler.

    CALL    InitTimer1              ;Initializes Timer1
```

```
359
360         CALL    InstallTimer1Handler    ;Installs Timer1Handler on the interrupt vector.
361
362         CALL    InitMotorLaser
363
364         CALL    InitParser
365
366         CALL    InitParallelB
367
368
369         RET                             ;End of ResetMotorMain
370     ResetMotorMain          ENDP
371
372
373
374
375
376     ; SerialReceivedHandler
377     ;
378     ; Description:
379     ;
380     ; Operation:
381     ;
382     ; Arguments:
383     ;   AH(Event Type)                      -    The type of the event to be enqueued
384     ;                                            (unused in this function)
385     ;   AL(Event Value, ReceivedCharacter) -    The value of the event to be enqueued
386     ; Return Value:
387     ;   None
388     ; Local Variables:
389     ;    ReceivedCharacter(AL)              -    The event value of SERIAL_RECEIVED_EVENT.
390     ;                                            It will be enqueued to RemoteDisplayBuffer
391     ; Shared Variables:
392
393     ; Global Variables:
394     ;   None
395     ; Input:
396     ;   None
397     ; Output:
398     ;   None
399     ; Error Handling:
400     ;   None
401     ; Algorithms:
402     ;   None
403     ; Data Structures:
404     ;   None
405     ; Registers Changed:
406     ;   AX, BX, SI, Flags
407     ; Limitations:
408     ;   None
409     ; Known bugs:
410     ;   None
411     ; Special Notes:
412     ;   None
413     ; Author:
414     ;   Sunghoon Choi
415     ; Revision History:
416
417
418     SerialReceivedHandler       PROC    NEAR
419                                 PUBLIC  SerialReceivedHandler
420
421
422     SaveCurrentStatus:                  ;Save current status to check what's changed in
        future.
423
```

```
424         PUSH AX
425         XOR BX, BX
426         CALL GetLaser
427         MOV StatusBuffer[BX], AX        ;StatusBuffer[0] = LaserStatus
428
429         ADD BX, WORD_SIZE
430         CALL GetMotorSpeed
431         MOV StatusBuffer[BX], AX        ;StatusBuffer[2] = Speed
432
433         ADD BX, WORD_SIZE
434         CALL GetMotorDirection
435         MOV StatusBuffer[BX], AX        ;StatusBuffer[4] = Direction
436         POP AX
437
438     ParseCommand:
439
440         CALL  ParseSerialChar
441         CMP   AX, PARSER_SUCCESS        ;Has parsing succeeded?
442         JNE   SendParserError           ;No. Go send parser error
443        ;JE      FindChangedStatus       ;Yes. Find what's changed.
444
445     FindChangedStatus:                  ;Find what's been changed.
446
447         XOR BX, BX
448         CALL GetLaser
449         CMP AX, StatusBuffer[BX]        ;Has laser status been changed?
450         JNE GetChangedLaserVal
451
452         ADD BX, WORD_SIZE
453         CALL GetMotorSpeed
454         CMP AX, StatusBuffer[BX]        ;Has speed been changed?
455         JNE GetChangedSpeedVal
456
457         ADD BX, WORD_SIZE
458         CALL GetMotorDirection
459         CMP AX, StatusBuffer[BX]        ;Has direction been changed?
460         JNE GetChangedDirectionVal
461
462         JMP EndSerialReceivedHandler    ;If nothing has changed, exit SerialReceivedHandler.
463
464     GetChangedLaserVal:
465                                         ;update the laser in MotorTxBuffer
466         MOV MotorTxBuffer, LASER_CHAR
467         MOV SI, OFFSET(MotorTxBuffer)
468         INC SI
469         CALL GetLaser
470         CALL Dec2String                 ;MotorTxBuffer is filled with 'L0000'
471         JMP SendChangedStatus
472
473     GetChangedSpeedVal:                 ;update the speed in MotorTxBuffer
474         MOV MotorTxBuffer, SPEED_CHAR
475         MOV SI, OFFSET(MotorTxBuffer)
476         INC SI
477         CALL GetMotorSpeed
478         CALL UnsignedDec2String         ;MotorTxBuffer is filled with 'S0000'
479         JMP SendChangedStatus
480
481     GetChangedDirectionVal:             ;update the direction in MotorTxBuffer
482         MOV MotorTxBuffer, DIRECTION_CHAR
483         MOV SI, OFFSET(MotorTxBuffer)
484         INC SI
485         CALL GetMotorDirection
486         CALL Dec2String                 ;MotorTxBuffer is filled with 'D0000'
487         JMP SendChangedStatus
488
489     SendChangedStatus:
```

```
490         MOV AX, DS
491         MOV ES, AX
492         MOV SI, OFFSET(MotorTxBuffer)
493         CALL SerialPutString            ;Send the changed status to the remote module.
494
495         JMP EndSerialReceivedHandler
496
497     SendParserError:
498
499         MOV AX, CS                      ;Since ParserErrorMsg is in the code segment,
500         MOV ES, AX                      ;we have to set CS=ES to call SerialPutString.
501         MOV SI, OFFSET(ParserErrorMsg)
502
503         CALL SerialPutString            ;Send Parser Error Message to Retmote.
504
505     EndSerialReceivedHandler:
506         RET                             ;End of SerialReceivedHandler.
507
508     SerialReceivedHandler    ENDP
509
510
511
512     ; SerialErrorHandler
513     ;
514     ; Description:
515     ;     It displays SerialErrorMsg on LED digits when received a serial error.
516     ; Operation:
517     ;    After setting SI to the offset of SerialErrorMsg, it calls Display function to
        display
518     ;   the serial error message on LED digits when a serial error is received.
519     ; Arguments:
520     ;   None
521     ; Return Value:
522     ;   None
523     ; Local Variables:
524     ;    None
525     ; Shared Variables:
526     ;   None
527     ; Global Variables:
528     ;   None
529     ; Input:
530     ;   None
531     ; Output:
532     ;   None
533     ; Error Handling:
534     ;   None
535     ; Algorithms:
536     ;   None
537     ; Data Structures:
538     ;   None
539     ; Registers Changed:
540     ;   AX, SI, Flags
541     ; Limitations:
542     ;   None
543     ; Known bugs:
544     ;   None
545     ; Special Notes:
546     ;   None
547     ; Author:
548     ;   Sunghoon Choi
549     ; Revision History:
550     ;     12/01/2016    Sunghoon Choi      Created
551     ;     12/02/2016    Sunghoon Choi      Initial Compilation
552     ;     12/02/2016    Sunghoon Choi      Updated documentation
553
554     SerialErrorHandler      PROC    NEAR
```

```
555                         PUBLIC  SerialErrorHandler
556
557   DisplaySerialErrorMsg:
558       MOV AX, CS                      ;Since SerialErrorMsg is in the code segment,
559       MOV ES, AX                      ;we have to set CS=ES to call Display.
560       MOV SI, OFFSET(SerialErrorMsg)  ;Set SI to the offset of SerialErrorMsg to call
561                                       ;Display to display the error message on LED digits.
562       CALL SerialPutString            ;Call Display to display the error message on LED.
563
564   EndSerialErrorHandler:
565       RET                             ;End of SerialErrorhandler
566
567   SerialErrorHandler   ENDP
568
569
570   CODE ENDS
571
572
573
574
575
576
577   DATA SEGMENT PUBLIC 'DATA'
578       StatusBuffer                DW  STATUS_BUFFER_LEN   DUP
          (?)
579       MotorTxBuffer               DB  MOTOR_TX_BUFFER_LEN DUP (?)
580                                       ;The buffer which contains the characters to be sent.
581
582   DATA ENDS
583
584
585
586
587   STACK   SEGMENT STACK  'STACK'
588
589                   DB      80 DUP ('Stack ')       ;240 words
590
591   TopOfStack      LABEL   WORD
592
593   STACK   ENDS
594
595   END    START
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                               ;
;                                  MtrMain.inc                                  ;
;                                  Homework 10                                  ;
;                                 Sunghoon Choi                                 ;
;                                                                               ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; Description:
;  This file contains the definitions for the constants of MtrMain.asm
;
; Revision History:
;     12/01/2016    Sunghoon Choi      Created
;     12/02/2016    Sunghoon Choi      Initial Compilation
;     12/02/2016    Sunghoon Choi      Updated documentation

MOTOR_TX_BUFFER_LEN     EQU      9  ;The length of MotorTxBuffer
STATUS_BUFFER_LEN       EQU      8  ;The length of StatusBuffer

LASER_CHAR          EQU     'L'
SPEED_CHAR          EQU     'S'
DIRECTION_CHAR      EQU     'D'
```