CEG 3136 – COMPUTER ARCHITECTURE II

LAB 2

HARDWARE INTERFACING

STUDENT NAME:

1) HAZIQ HAFIZI BIN MD YUSOF (8346453)
2) HANIF BIN ARIFFIN (8346480)

DATE: SEPTEMBER 18,2017

TA: Anas Alakhras

OBJECTIVES

MAIN OBJECTIVE:

- Learn how to write assembly code
- Learn about Pull Up Control Register
- Learn how to scan keypad on Dragon 12 board

SUB-OBJECTIVE:

- Get to know the shape of the Dragon 12 and the lab for CEG 3136 works

EQUIPMENTS AND COMPONENTS

- Dragon 12 Plus Trainer
- Windows PC
- MiniIDE

SOFTWARE / HARDWARE DESIGN

KEYPAD.ASM

```
;----------------------------------------------------------------------
; File: Keypad.asm
; Author:

; Description:
;  This contains the code for reading the
;  16-key keypad attached to Port A
;  See the schematic of the connection in the
;  design document.
;
;  The following subroutines are provided by the module
;
; char pollReadKey(): to poll keypad for a keypress
;                 Checks keypad for 2 ms for a keypress, and
;                 returns NOKEY if no keypress is found, otherwise
;                 the value returned will correspond to the
;                 ASCII code for the key, i.e. 0-9, *, # and A-D
; void initkey(): Initialises Port A for the keypad
;
; char readKey(): to read the key on the keypad
;                 The value returned will correspond to the
;                 ASCII code for the key, i.e. 0-9, *, # and A-D
;---------------------------------------------------------------------

; Include header files
 include     "sections.inc"
 include     "reg9s12.inc"  ; Defines EQU's for Peripheral Ports
 include     "delay.inc"

**************EQUATES**********


;-----Conversion table
NUMKEYS             EQU         16           ; Number of keys on the keypad
BADCODE      EQU          $FF          ; returned of translation is unsuccessful
NOKEY        EQU          $00          ; No key pressed during poll period
POLLCOUNT    EQU          1            ; Number of loops to create 1 ms poll time

 SWITCH globalConst  ; Constant data

 ; codes for scanning keyboard

KEY_1 EQU %11101110
KEY_2 EQU %11101101
KEY_3 EQU %11101011
KEY_A EQU %11100111
KEY_4 EQU %11011110
KEY_5 EQU %11011101
KEY_6 EQU %11011011
KEY_B EQU %11010111
KEY_7 EQU %10111110
KEY_8 EQU %10111101
KEY_9 EQU %10111011
KEY_C EQU %10110111
KEY_ASTERISK EQU %01111110
KEY_0 EQU %01111101
KEY_HASHTAG EQU %01111011
KEY_D EQU %01110111

 SWITCH code_section  ; place in code section
;-----------------------------------------------------------
; Subroutine: initKeyPad
;
; Description:
```

```
;      Initiliases PORT A
;-----------------------------------------------------------
initKeyPad:

       ;-- Set the DDRA and PUCR to enable input from the board
       MOVB   #$FF,  DDRA
       MOVB   #$01,  PUCR
       rts

;-----------------------------------------------------------
; Subroutine: ch <- pollReadKey
; Parameters: none
; Local variable:
; Returns
;       ch: NOKEY when no key pressed,
;       otherwise, ASCII Code in accumulator B

; Description:
;  Loops for a period of 2ms, checking to see if
;  key is pressed. Calls readKey to read key if keypress
;  detected (and debounced) on Port A and get ASCII code for
;  key pressed.
;-----------------------------------------------------------
; This routine does not require any local variable (no stack usage)


pollReadKey: PSHX

       ;-- The amount of time to loop, depending on the length of our loop cycle, set POLLCOUNT
       ;-- appropriately to cause 1ms delay.
       LDX    #POLLCOUNT

       ;-- Set PORTA to default value of $0F
       MOVB   #$0F, PORTA

pollReadKey_check_difference:

       ;-- Load the value of PORTA into AC B
       LDAB   PORTA

       ;-- Compare the value of PORTA( now in AC B) with #$0F
       CMPB   #$0F

       ;-- If equal it means no difference then we go to loop_check_difference_end to
       ;-- decrement POLLCOUNT and try again
       BEQ    pollReadkey_decrement_POLLCOUNT

       ;-- Else, prepare to delay for 1 ms to check for debouncing
       LDD    #1
       JSR    delayms

;-- Check if PORTA is still not equal to $0F.
;-- If it is equal, then that means that it was just an anomaly. We decrement POLLCOUNT and try
again until
;-- polllcount is zero, then we return NOKEY if we continue to fail.
;-- If it is NOT equal, then we pass the debouncing test. We will now let readKey to read the
input.
;-- After readKey routine is finished,
pollReadKey_check_after_debouncing:
       LDAB   PORTA;
       CMPB   #$0F
       BEQ pollReadkey_decrement_POLLCOUNT

       ;-- If the value in PORTA is not #$0F, meaning that it is in
       ;-- fact a key press (and not some random anomaly) then we
       ;-- go to readkey to evaluate the actual value of the key
       JSR readkey

       ;-- Restore value of REG X
       PULX
       RTS ;-- return from pollReadKey

;-- Decrement POLLCOUNT.
;-- If POLLCOUNT is zero then return NOKEY (by putting it in AC B) and exit subroutine.
;-- Else return with whatever the value obtained from readKey and exit subroutine
pollReadkey_decrement_POLLCOUNT:
       DEX
       BNE pollReadKey_check_difference
       LDAB NOKEY
       PULX
       RTS ;-- return from pollReadKey

;-----------------------------------------------------------
; Subroutine: readKey
; Arguments: none
; Local variable:
;      ASCII value in AC B

; Description:
;-- After pollReadKey have guaranteed that the change in voltage was not an anomaly, we now
;-- further guarantee that the input given is consistent for some period of time before we are
fully
;-- confident the change is exactly an input by the user. The translation from PORTA --> ASCII is
done
;-- by the translate_to_ASCII subroutine
;-----------------------------------------------------------
; Stack Usage
       OFFSET 0  ; to setup offset into stack
READKEY_CODE       DS.B 1 ; code variable
READKEY_VARSIZE:
READKEY_RA         DS.W 1 ; return address

;-- push AC A because we are going to use it in this subroutine
readKey:psha

       ;-- Reserve stack space for our local variables
    LEAS     -READKEY_VARSIZE,   SP

;-- Readkey_main is a reference to beginning to the module where we will set PORTA to the default
value and
;-- see if anything changes (possibly under the same assumption that the keypress will keep
overwriting the value in PORTA(??).
;--

readkey_main:
       ;-- Set PORTA to the default value of $0F
    MOVB     $0F,                       PORTA

;-- While PORTA == 0x0F, we will continue to loop indefinitely. This is probably not a good idea in
general (?).
;-- But won't necessarily cause any hickups, the user just have to press the keypad and it will
exit this infinite loop.
;-- Else it will continue to perform the same check that we see in pollReadKey but this time it is
not checking for
;-- debouncing (?) and delays for a lot longer (10ms). Instead of checking if PORTA is the same as
the default value,
;-- this one checks if the (old_keypress) is the same as the (new_keypress). If the value in PORTA
is the
;-- same as the one obtained from before then we will proceed to call the subroutine to translate
this value and
;-- convert it into ASCII(hexadecimal) value.
readkey_obtain_value_from_PORTA:
    LDAB     PORTA
    CMPB     #$0F
    BEQ      readkey_obtain_value_from_PORTA

    ;-- Since we have exited the readkey_obtain_value_from_PORTA loop,
    ;-- we are now under the assumption that something changed in PORTA,
    ;-- we will now delay for 10ms before checking again to see if we have the same value as
before. Before we
```

```
    ;-- do that, we store the value obtained from PORTA into the variable READKEY_CODE (which is in
the stack).
    MOVB     PORTA,                     READKEY_CODE, SP
    LDD      #10
    JSR      delayms

    ;-- Get and compare the new content PORTA after the 10ms delay. If the resulting
    ;-- substraction (which is how CMPB works) is not equal to zero (hence we are using BNE)
    ;-- then the program will loop back to readkey_main and perform the same check as
    ;-- before again. We go back to readkey_main because there could
    ;-- be something weird that caused the input to change during the 10ms delay and let the
    ;-- user try again. Even better, if the user's hand is slow enough,
    ;-- this is just reducing the probability that the hardware messes up.
    LDAB     PORTA
    CMPB     READKEY_CODE,       SP
    BNE      readkey_main

    ;-- Since we have confirmed that the value in PORTA is indeed consistent for
    ;-- some time, we can guarantee that this is indeed a keypress.
    ;-- Call the translate_to_ASCII subsroutine to parse the value in PORTA. That
    ;-- will store the return value in B, we then put this
    ;-- value in the variable for this subsroutine.
    JSR      translate_to_ASCII
    STAB     READKEY_CODE,       SP

;-- To ensure that the processing of the keypad press is only evaluated AFTER the
;-- user have released the key, we perform an infinite loop
;-- that will only be broken when the value in PORTA is equal to the default value that
;-- is set in the beginning of that infinite loop. To ensure that the user have indeed released the
key,
;-- we will introduce a delay of 10ms (the debouncing time) into the infinite loop. This will
likely
;-- cause a slow program, but its better than having a buggy program.
;--
;-- ________________ISSUES___________________:
;-- If the value in PORTA is being continuously updated indefinitely, the program will get stuck
here and there is no recovering from this error.
readkey_check_release_key:
    MOVB     #$0F,                      PORTA

    ;-- Call delayms subroutine to delay for 10ms
    LDD      #10
    JSR      delayms

    ;-- Check if the value that we have set into PORTA is still 0x0F.
    ;-- If it is not equal to 0x0F, then the user is still pressing the key and PORTA is still
being overwritten by that.
    LDAB     PORTA
    CMPB     #$0F
    BNE      readkey_check_release_key

    ;-- Load the value that we obtained from translate_to_ASCII and put it into AC B to be
translated by the translate_keypad subroutine.
    LDAB     READKEY_CODE,       SP

    ;-- Recover the stack memory that this subroutine uses.
    LEAS     READKEY_VARSIZE,    SP
    PULA
    RTS

;------------------------------------------------------------
; Subroutine: translate_to_ASCII
; Arguments: No argument
; Returns:
    The value in AC B

; Description: Depending on the key pressed, it will return the corresponding ASCII value
;------------------------------------------------------------
;-- This subroutine does not use local variables so no stack is needed
;-- translate_to_ASCII is just a bunch of if's statement that will check each the given value
obtained with the rows.
translate_to_ASCII:

       ;-- Check if PORTA is KEY_0
    cmpb       #KEY_0
    bne        translate_to_ASCII_found_key_0

       ;-- Check if PORTA is KEY_1
    cmpb       #KEY_1
    bne        translate_to_ASCII_found_key_1

       ;-- Check if PORTA is KEY_2
       cmpb    #KEY_2
    bne        translate_to_ASCII_found_key_2

    ;-- Check if PORTA is KEY_3
    cmpb       #KEY_3
    bne        translate_to_ASCII_found_key_3

       ;-- Check if PORTA is KEY_4
    cmpb       #KEY_4
    bne        translate_to_ASCII_found_key_4

       ;-- Check if PORTA is KEY_5
       cmpb    #KEY_5
    bne        translate_to_ASCII_found_key_5

    ;-- Check if PORTA is KEY_6
    cmpb       #KEY_6
    bne        translate_to_ASCII_found_key_6

       ;-- Check if PORTA is KEY_7
    cmpb       #KEY_7
    bne        translate_to_ASCII_found_key_7

       ;-- Check if PORTA is KEY_8
       cmpb    #KEY_8
    bne        translate_to_ASCII_found_key_8

    ;-- Check if PORTA is KEY_9
    cmpb       #KEY_9
    bne        translate_to_ASCII_found_key_9

       ;-- Check if PORTA is KEY_A
    cmpb       #KEY_A
    bne        translate_to_ASCII_found_key_A

       ;-- Check if PORTA is KEY_B
       cmpb    #KEY_B
    bne        translate_to_ASCII_found_key_B

    ;-- Check if PORTA is KEY_C
    cmpb       #KEY_C
    bne        translate_to_ASCII_found_key_C

       ;-- Check if PORTA is KEY_D
    cmpb       #KEY_D
    bne        translate_to_ASCII_found_key_D

       ;-- Check if PORTA is KEY_ASTERISK
       cmpb    #KEY_ASTERISK
    bne        translate_to_ASCII_found_key_asterisk

    ;-- Check if PORTA is KEY_HASHTAG
       cmpb    #KEY_HASHTAG
    bne        translate_to_ASCII_found_key_hashtag

    ;-- All check failed for whatever reason, returns BADCODE
    LDAB BADCODE
    RTS

translate_to_ASCII_found_key_0:
       LDAB #'0'
       RTS
translate_to_ASCII_found_key_1:
       LDAB #'1'
       RTS
translate_to_ASCII_found_key_2:
       LDAB #'2'
       RTS
translate_to_ASCII_found_key_3:
       LDAB #'3'
       RTS
translate_to_ASCII_found_key_4:
       LDAB #'4'
       RTS
translate_to_ASCII_found_key_5:
       LDAB #'5'
       RTS
translate_to_ASCII_found_key_6:
       LDAB #'6'
       RTS
translate_to_ASCII_found_key_7:
       LDAB #'7'
       RTS
translate_to_ASCII_found_key_8:
       LDAB #'8'
       RTS
translate_to_ASCII_found_key_9:
       LDAB #'9'
       RTS
translate_to_ASCII_found_key_A:
       LDAB #'a'
       RTS
translate_to_ASCII_found_key_B:
       LDAB #'b'
       RTS
translate_to_ASCII_found_key_C:
       LDAB #'c'
       RTS
translate_to_ASCII_found_key_D:
       LDAB #'d'
       RTS
translate_to_ASCII_found_key_hashtag:
       LDAB #'#'
       RTS
translate_to_ASCII_found_key_asterisk:
       LDAB #'*'
       RTS
```

DELAY.ASM

```
;-------------------------------------------------------
; Alarm System Simulation Assembler Program
; File: delay.asm
; Description: The Delay Module
; Author: Gilbert Arbez
; Date: Fall 2010
;-------------------------------------------------------

; Some definitions
    SWITCH code_section

;-------------------------------------------------------
; Subroutine setDelay
; Parameters: cnt - accumulator D
; Returns: nothing
; Global Variables: delayCount
; Description: Intialises the delayCount
;              variable.
;-------------------------------------------------------
;-- This subroutine does not use local variables so no stack is needed
setDelay:
         STD       delayCount
    rts


;-------------------------------------------------------
; Subroutine: polldelay
; Parameters:  none
; Returns: TRUE when delay counter reaches 0 - in accumulator A
; Local Variables
;   retval - acc A cntr - X register
; Global Variables:
;      delayCount
; Description: The subroutine delays for 1 ms, decrements delayCount.
;              If delayCount is zero, return TRUE; FALSE otherwise.
;   Core Clock is set to 24 MHz, so 1 cycle is 41 2/3 ns
;   NOP takes up 1 cycle, thus 41 2/3 ns
;   Need 24 cyles to create 1 microsecond delay
;   8 cycles creates a 333 1/3 nano delay
;      DEX - 1 cycle
;      BNE - 3 cyles - when branch is taken
;      Need 4 NOP
;   Run Loop 3000 times to create a 1 ms delay
;-------------------------------------------------------
;-- This subroutine does not use local variables so no stack is needed

polldelay: pshb
       pshx
       pshy

       ;-- Obtain the delayCount value set from setDelay and put it into register X
    LDX                   delayCount
       LDAA         #FALSE
       LDY          #3000

polldelay_main:
       NOP
       NOP
       NOP
       NOP

       ;-- A usual loop function, decrement #3000 by 1 and loop back to polldelay_main until its 0
       DEY
       BNE          polldelay_main

       ;-- Decrement delayCount (in X) and check if its zero. If its zero, that means that the loop
finished counting. I
       ;-- If its not, that means that while counting, something is interrupting the process (??)
       DEX
       BNE          polldelay_end_loop
       LDAA         #TRUE;

polldelay_end_loop:
       ;-- restore registers and stack
       puly
       pulx
       pulb
       rts


;-------------------------------------------------------
; Subroutine delayms
; Parameters: num - accumulator D
; Returns: nothing
; Global Variables:
; Description: Set delay for num ms
;-------------------------------------------------------
```

```
;-- This subroutine does not use local variables so no stack is needed
delayms:
         ;-- Initialize our delayCounter to be whatever value this function is given in D
         JSR        setDelay
    JSR             polldelay

       ;-- Test if AC A is zero, if it is is, then skip BNE, if it is then BNE back and request a
pollDelay again.
       ;-- AC A should content the boolean returned from pollDelay during this sequence.
       TSTA
       BNE          delayms
       rts


;-------------------------------------------------------
; Global variables
;-------------------------------------------------------
   switch globalVar
delayCount ds.w 1   ; 2 byte delay counter
```

Conclusion

Lab 2 has succeeded although he faced a lot of problem. We tried to run our original code on the board but it could not read the keypad due to some mistake that we made in our code. However, we able to resolve the problem after we consulted with the TA and we able to demo the project within the lab period.