@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

@ ECE 371 – PROGRAMMING PROJECT II

@----------------------------------------------------------------------------

@ INTERRUPT CONTROLLER ( Extra Credit)

@ In the third part of this project, I will develop an interrupt procedure that

@ services an interrupt request from a debounced, push-button switch. There are

@ 4 states for pressing:

@ 00 -> LEDs rotate .

@ 01 -> LED turn off.

@ 10 -> LED count up by binary

@ 11 -> LED count down binary

@ After 4th pressing,if I continue to press the button ( 5th press), it will return to @ the State 00 ( LEDS rotate).

@

@----------------------------------------------------------------------------

@ Hai Dang Hoang

@ Reference: Douglas V. Hall and Leela Yadlapalli

@ DECEMBER 18th, 2014

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

**.text**

**.global** \_start

**.global** INT\_DIRECTOR

**\_start:**

LDR R13, =STACK @ Point to base of Stack

ADD R13, R13, #0x1000 @ Point to top of Stack

LDR R0, =0x4804C000 @ Base Address for GPIO1 registers

ADD R4, R0, #0x190 @ Address of GPIO1\_ClearDATAOUt register

MOV R7, #0x01E00000 @ Load value to turn off LED on GPIO1\_21

STR R7,[R4] @ Write to GPIO1\_ClearDataOUt register

@ Program GPIO1\_21 as output

ADD R1,R0,#0x0134 @ Make GPIO1\_OE register address

LDR R6,[R1] @ Read current GPIO1 Output Enable register

LDR R7, =0xFE1FFFFF @ Word to enable GPIO1\_21 as output ( 0 enables)

AND R6,R7,R6 @ Clear bit 21 ( Modify)

STR R6, [R1] @ Write to GPIO1 Output Enable register

@ Detect falling edge on GPIO1\_31 and enable to assert POINTRPEND1

ADD R1, R0, #0x14C @ R1 = address of GPIO1\_FallingDetect register

LDR R2,=0x80000000 @ Load value for bit 31

LDR R3,[R1] @ Read GPIO1\_FallingDetect register

ORR R3,R3,R2 @ Modify ( set bit 31)

STR R3, [R1] @ Write back

ADD R1,R0,#0x34 @ Create address of GPIO1\_IRQSTATUS\_SET\_0 register

STR R2,[R1] @ enable GPIO1\_31 request on POINTRPEND1

@ Initialize INTC

LDR R1, =0x482000E8 @ Address of INTC\_MIR\_Clear3 register

MOV R2,#0x04 @ Value to unmask INTC INT 98 GPIOINT1A

STR R2,[R1] @ Write to INTC\_MIR\_CLEAR3 register

@ Load status LED\_Display

MOV R5, #0x0 @ R5 Register(Index) control status of LED\_Display array

LDR R7,=0x01E00000 @ Value for clear all 4 USR LEDS

@ Make sure processor IRQ enabled in CPSR

MRS R3, CPSR @ Copy CPSR to R3

BIC R3,#0x80 @ Clear bit 7

MSR CPSR\_c, R3 @ Write back to CPSR

@ Wait for interrupt

**LOOP:**

LDR R2,=Flag @ Load pointer of Flag to check current state

LDR R3,[R2] @ Load value from Flag memory

CMP R3,#0x0 @ Compare with 0

LDREQ R1, =LED\_Display @ Load Pointer LED\_display for rotating LED

BLEQ RotatingLED @ Call Procedure Rotating LED

CMP R3,#0x01 @ Compare with 1 ( Check Pressing button for LED turn of

BLEQ Turn\_offLED @ If equals, turn off LED

CMP R3,#0x02 @ Compare with 2 ( Check Pressing button for LED turn on by binary)

LDREQ R1, =LED\_Display\_Binary @ Load Pointer LED\_display\_Binary for count on by binary

BLEQ Count\_up @ If Equals -> Count up by binary

BLNE Count\_down @ Else, Count down by binary

B LOOP

**RotatingLED:**

BL LED\_on @ Call Procedure LED turn on

BL Delay @ Call delay 2s procedure

STR R7,[R4] @ Clear all 4 USR LEDS

CMP R5,#12 @ Check index, if index >12 -> index = 0

MOVEQ R5,#0 @ If equals -> Reset index =0

ADDNE R5,R5,#4 @ If not, increment Pointer by 4

B LOOP

**Turn\_offLED:**

STR R7,[R4] @ Clear all 4 USR LEDS

B LOOP

**Count\_up:**

BL LED\_on @ Call Procedure LED turn on

BL Delay @ Call delay 2s procedure

STR R7,[R4] @ Clear all 4 USR LEDS

CMP R5,#60 @ Check index, if index >60 (15x4) -> index = 0

MOVEQ R5,#0 @ If equals -> Reset index =0

ADDNE R5,R5,#4 @ If not, increment Pointer by 4

B LOOP

**Count\_down:**

BL LED\_on @ Call Procedure LED turn on

BL Delay @ Call delay 2s procedure

STR R7,[R4] @ Clear all 4 USR LEDS

CMP R5,#0 @ Check index, if index=0 -> index = 60

MOVEQ R5,#60 @ If equals -> Reset index =60

SUBNE R5,R5,#4 @ If not, Decrement Pointer by 4

B LOOP

@ Procedure for rotating 4 USR LEDS

**LED\_on:**

STMFD R13!,{R2-R5,R14} @ Store uses registers on Stack

ADD R2,R0,#0x194 @ Load address of GPIO1\_SetDataOut

LDR R3,[R1,R5] @ Load value for turn in USR LED Depending on R5 (Index)

STR R3,[R2] @ Turn on led by storing the value into GPIO1\_SET\_DATA\_OUT

LDMFD R13!,{R2-R5,R14} @ Restore saved resigisters

MOV PC,R14 @ Return to mainline

@ Procedure for delaying 1s

**Delay:**

STMFD R13!,{R9,R14} @ Save uses registers

LDR R9,=0x00400000 @ Intialize delay loop counter

**NEXT:**

SUBS R9,R9,#0x1 @ Decrement loop counter

BNE NEXT @ Until loop counter equal 0

LDMFD R13!,{R9,R14} @ Restore values for saved registers

MOV PC, R14 @ Return to mainline

@ Interrupt

**INT\_DIRECTOR:**

STMFD SP!,{R0-R3,LR} @ Push registers on stack

LDR R0,=0x482000F8 @ Address of INTC-PENDING\_IRQ3 register

LDR R1,[R0] @ Read INTC-PENDING\_IRQ3 register

TST R1,#0x00000004 @ TEST BIT 2

BEQ PASS\_ON @ Not from GPIOINT1A, go to back to wait loop, Else

LDR R0,=0x4804C02C @ Load GPIO1\_IRQSTATUS\_0 register address

LDR R1,[R0] @ Read Status register

TST R1,#0x80000000 @ Check if bit 31=1

BNE BUTTON\_SVC @ If bit 31=1, then button pushed

BEQ PASS\_ON @ If bit 31=0, then go to back to wait loop

**PASS\_ON:**

LDMFD SP!,{R0-R3,LR} @ Restore registers

SUBS PC,LR,#4 @ Pass execution on to wait LOOP for now

**BUTTON\_SVC:**

LDR R1,=0x80000000 @ Value to turn off GPIO1\_31 Interrupt request

@ This will turn off INTC interrupt request also

STR R1,[R0] @ Write to GPIO1\_IRQSTATUS\_0 register

@ Turn off NEWIRQA bit in INTC\_CONTROL, so processor can respond to new IRQ

LDR R0,=0x48200048 @ Address of INTC\_CONTROL register

MOV R1,#0x1 @ Value to clear bit 0

STR R1,[R0] @ Write to INTC\_CONTROL register

MOV R1,#0x01E00000 @ Load value to turn ON or OFF for 4 USR LED

LDR R0,=Flag @ Load pointer of Flag to check current state

LDR R3,[R0] @ Load value from Flag memory

CMP R3,#0x0 @ Compare with 0

MOVEQ R3,#0x1 @ If equal, Change flag to 1 ( State 1)

STREQ R3,[R0] @ Store it into memory flag

BEQ Done @ Done

CMP R3,#0x01 @ Check flag = 1 or not

MOVEQ R3,#0x2 @ If equal, Change Flag to 2 ( State 2)

STREQ R3,[R0] @ Store it in Memory

MOVEQ R5,#0 @ Reset R5 register for controlling display LED

BEQ Done @ Done

CMP R3,#0x02 @ Check flag = 2 or not

MOVEQ R3,#0x3 @ If equal, Change Flag to 3 ( State 3)

STREQ R3,[R0] @ Store it in Memory

SUBEQ R5,R5,#4 @ Need to subjact R5 by 4 because it added 4 from State 2 before checking

BEQ Done @ Done

MOV R3,#0x0 @ Currentlly, flag=3 and change to 0 ( State 0)

STR R3,[R0] @ Store it in memory.

MOV R5,#-4 @ Reset R5 register for controlling display LED

**Done:** LDMFD SP!,{R0-R3,LR} @ Restore registers

SUBS PC,LR,#4 @ Return from IRQ interrupt procedure

**.align** 2

**.data**

**LED\_Display:** **.word** 0x00200000, 0x00400000, 0x00800000, 0x01000000

@ Values for turn in 4 USR LEDS

@ 0x00200000: USR1 GPIO1\_21 on

@ 0x00400000: USR2 GPIO1\_22 on

@ 0x00800000: USR3 GPIO1\_23 on

@ 0x01000000: USR4 GPIO1\_24 on

**LED\_Display\_Binary:** **.word** 0x00000000, 0x00200000, 0x00400000, 0x00600000

**.word** 0x00800000, 0x00A00000, 0x00C00000, 0x00E00000

**.word** 0x01000000, 0x01200000, 0x01400000, 0x01600000

**.word** 0x01800000, 0x01A00000, 0x01C00000, 0x01E00000

**Flag:** **.word** 0x0 @ Aside a memory to check condition (State)

@ 00 -> LEDs rotate

@ 01 -> LED turn off

@ 10 -> LED count up binary

@ 11 -> LED count down binary

**.align** 2

**STACK:** .rept 1024

**.word** 0x0

.endr

.end