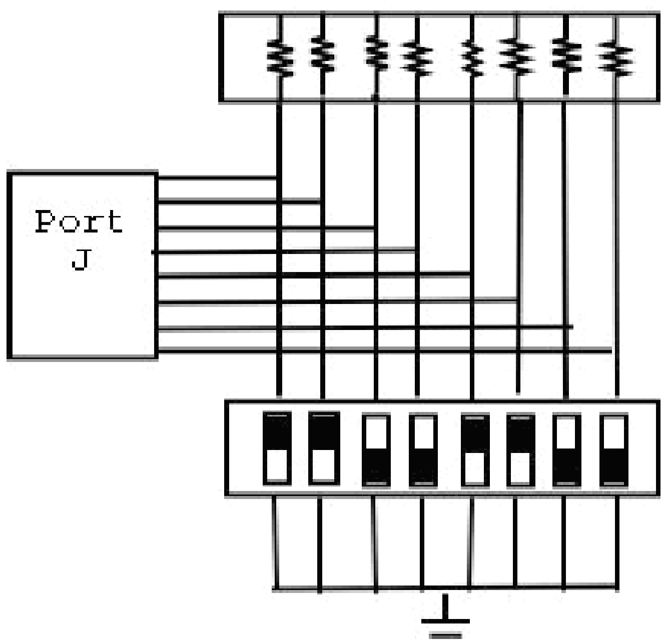
EECS 388 HW #4

Due: April 1

1. (30) Write a program to do matrix addition and subtraction of two 5x5 matrices M1 and M2 based on the position of an eight-position DIP switch. Use port J to read the switch positions. If the number of ones is even, do the addition operation M1+M2. Otherwise, do the subtraction operation M1-M2. The results of either operation will be stored in M3. Your program will assume that each matrix is stored as five consecutive column vectors (i.e., each vector is 5 by 1). Each matrix will be stored at the following locations:

M1: $4500_$4519
M2: $4520_$4539
M3: $4540_$4559

Your program must use a subroutine to do the arithmetic of two column vectors, and either return or store the result. Essentially, this function will add or subtract each 5 item "row" of the matrix. Arguments should be passed to the subroutine using the stack. The arguments should include a value to represent the arithmetic operator, and either the values for a length 5 vector, or a starting address for the vector. The function may then either return or store the result. The program must be well-commented. The DIP switch is pictured below:



To address this problem, first we need to look up the relevant addresses for Port J. Port J data is at address $28, and the data direction is at address $29. We want to configure all bits to input, so we'll want to initialize the direction with mask '$00'. Here is the code:

```
PORTJ       EQU       $0028       ; location of Port J
```

```
DDRJ        EQU     $0029       ; location of DDRJ register
DDRJ_INI    EQU     $00         ; all of Port J as input
M1          EQU     $4500       ; Matrix 1
M2          EQU     $4520       ; Matrix 2
M3          EQU     $4540       ; Matrix 3
ADD         EQU     $00         ; flag for addition
SUB         EQU     $01         ; flag for subtraction

            ORG     $5000
            LDS     #$8000      ; declare stack
            LDAA    #DDRJ_INI   ; load initialization value to A
            STAA    #DDRJ       ; initialize DDRJ

WORK        LDAA    PORTJ       ; read switch positions
            LDX     #$0008      ; initialize X to 8 before counting
            LDAB    #$00        ; initialize B to 0 to count the number of 1's

COUNT       CPX     #$00        ; check if we've looped 8 times
            BEQ     MAT_OP
            DEX                 ; decrement X
            BITA    #$01        ; check LSB of A to see if it is a 1
            BEQ     COUNT       ; if not, loop again
            INCB                ; LSB was a 1 so increment B by 1
            BRA     COUNT

MAT_OP      BITB    #$01        ; check LSB of B to see if it is a 1 (odd)
            BEQ     ADDITION    ; if even, do addition
            LDAA    #SUB        ; load A with the flag for subtraction
            BRA     DISPATCH
ADDITION    LDAA    #ADD        ; load A with the flag for addition

DISPATCH    LDAB    #$05        ; we need to loop 5 times (5 rows)
            STAB    $6000       ; save loop counter
            LDX     #M3         ; load X with address of M3
            PSHX                ; push M3 onto the stack
            LDX     #M1         ; load X with address of M1
            LDY     #M2         ; load Y with address of M2

MAT_LOOP    LDAB    $6000       ; load loop counter
            CMPB    #$00        ; check if we've looped 5 times
            BEQ     WORK        ; if yes, go back to beginning
            DECB                ; decrement loop counter
            STAB    $6000       ; store loop counter
            PSHX                ; push X (current M1 index) onto the stack
            PSHY                ; push Y (current M2 index) onto the stack
            PSHA                ; push A (flag) onto the stack
```

```
        JSR          MAT_FUN     ; jump to the matrix subroutine
        BRA MAT_LOOP             ; loop again

MAT_FUN PULA                     ; pull flag from the stack
        PULY                     ; pull M2 from the stack
        PULX                     ; pull M1 from the stack
        LDAB         #$05        ; we need to loop 5 times
        STAB         $6500       ; store loop counter value

OP_LOOP LDAB         $6500       ; load loop counter value
        CMPB         #$00        ; have we looped 5 times?
        BEQ          DONE
        DECB                     ; decrement loop counter
        STAB         $6500       ; store loop counter
        LDAB         1,X+        ; load next M1 value into B
        CMPA         #ADD        ; compare A with flag for addition
        BEQ          MAT_ADD     ; if it is, perform the addition
        SUBB         1,Y+        ; subtract M2 value from M1 value
        BRA          RESULT      ; skip addition
MAT_ADD ADDB         1,Y+        ; add M1 value to M2 value
RESULT  STX          $7000       ; store X temporarily
        PULX                     ; pull current M3 index from stack
        STAB         1,X+        ; store result of matrix operation
        PSHX                     ; push next M3 index onto stack
        LDX          $7000       ; restore X to M1 index
        BRA          OP_LOOP     ; loop again

DONE    RTS
```

2. (20) An eight-position DIP switch is connected to PORT A of the 68HC12. Provide the codes to read the position of the DIP switches when an IRQ interrupt occurs.

Let us write the program and interrupt service routine that would allow us to read the position of the DIP switches when an IRQ interrupt occurs:

```
INTCR      EQU     $001E          ; address of INTCR register
INTCR_INI  EQU     $60            ; initial value of INTCR
DDRA       EQU     $0002          ; location of DDRA register
DDRA_INI   EQU     $00            ; all 8 bits are input
PORTA      EQU     $0000          ; location of Port A

           ORG     $FFF1          ; address of IRQ vector
           FDB     $IRQ_DIP       ; name of our ISR
           ORG     $4000
           LDS     #$8000
```

```
        LDAA    #INTCR_INI      ; load init values to A
        STAA    INTCR           ; initialize INTCR
        LDAA    #DDRA_INI       ; load input direction to A
        STAA    DDRA            ; initialize DDRA
        CLI

        ORG     $9000
IRQ_DIP LDAA    PORTA           ; reads the dip switch positions
        …                       ; do work with it
        RTI
```

3. (30) Extend the battery backup supply example for one primary battery and two backup batteries. Show all initialization steps in the main code and the interrupt service routine. Use PORTG[4:5] for input signals (00: primary battery in use, 01: the first backup, 11: the second backup) and PORTG[0:1] for out signals (again 00-01-11 for primary-1st backup-2nd backup).

We can reuse much of the code from the book and add a few things to support the 2nd backup battery:

```
STACKTOP  EQU   $3FFF       ; equate STACKTOP with $3FFF
INTCR     EQU   $001E       ; address of INTCR register
INTCR_INI EQU   $60         ; initial value of INTCR
DDRG      EQU   $0033       ; location of DDRG register
DDRG_INI  EQU   $03         ; [4:5] input and [0:1] output
PORTG     EQU   $0031       ; location of Port G
PRIMARY   EQU   $00         ; test mask for primary
BACK_UP1  EQU   $10         ; test mask for 1st back up
BACK_UP2  EQU   $30         ; test mask for 2nd back up

          ORG   $FFF2       ; address of IRQ vector
          FDB   $IRQ_DIP    ; name of our ISR
          ORG   $2000
          LDS   #STACKTOP
          LDAA  #INTCR_INI  ; load init values to A
          STAA  INTCR       ; initialize INTCR
          LDAA  #DDRG_INI   ; load input direction to A
          STAA  DDRG        ; initialize DDRG
          CLI

          ORG   $9000       ; interrupt service routine
IRQ_DIP   LDAA  PORTG       ; determine battery in use
          ANDA  #BACK_UP2   ; mask out unneeded bits
          CMPA  #BACK_UP2   ; check if backup 2 is in use
```

```
          BNE     SWAP_BU2      ; back up 2 not in use
          LDAA    #PRIMARY      ; swap to primary
          STAA    PORTG
          BRA     DONE
SWAP_BU2  CMPA    #BACK_UP1     ; check if backup 1 is in use
          BNE     SWAP_BU1      ; back up 1 not in use (primary is)
          LDAA    #BACK_UP2     ; swap to backup 2
          STAA    PORTG
          BRA     DONE
SWAP_BU1  LDAA    #BACK_UP1     ; swap to backup 1
          STAA    PORTG
          RTI
```

4. (10) Page 290, Fundamental #2
In the preceding question, if the MCLK was 2 MHz and the prescaler bits PR[2:1:0] were set to 000, how much time in seconds transpired between the two input capture events?

First, we need to solve the preceding question, where we get (assuming no rollover occurred):

$FF20 - $1037 = $EEE9

which is 61,161 in decimal. We are told that the clock runs at 2 MHz, so each tick of the clock takes 500 ns. With the prescaler bits set to 000, we know the divisor is 1, so each tick of the clock increments the counter. Therefore, the calculation is simple:

number of seconds: 0.0000005 _ 61161 = 0.0305805 sec

or about 31 ms.

5. (10) Page 290, Fundamental #3
Repeat the preceding question if PR[2:1:0] were set to 101.

In this case, the prescaler bits are set to 101, which corresponds to a divisor of 32. Therefore we need only multiply the result from the previous question by 32 to get our answer here:

number of seconds: 0.0305805 _ 32 = 0.978576 sec

or about 1 sec.