

ANTES QUESTIONS:

1. What are the steps that the 68HC11 processor takes in servicing an interrupt?
(Refer Lecture 7 & 8)

ANSWER

A device requiring attention from the CPU can activate an interrupt service routine simply by asserting its interrupt output.

The interrupt causes the CPU to :

1. If the interrupt mask flag is cleared, then interrupts are sampled after each instruction completes. If interrupts are masked, program execution continues in a normal manner.
When the interrupt mask is cleared, and an interrupt is asserted the CPU will:
2. Complete the current instruction (in the current program)
-i.e. delay interrupt servicing until the current instruction completes - the delay time varies from 1 cycle (ABA instruction) to 41 (IDIV instruction) clock cycles
3. Push all registers - PC, X, Y, AccA, AccB & CCR
4. Mask further interrupts by setting the I bit in CCR
5. Fetch the interrupt vector
(assuming an IRQ interrupt - vector is fetched from \$FFF2 & \$FFF3)
6. Jump to the address fetched from the int. vector

7. In the interrupt service routine we perform the minimum necessary functions required to manage the peripheral and then reset the bit in the appropriate register to de-assert the interrupt (if we don't do this, then the device will immediately be interrupted again)

8. Finally execute a RTI instruction that will pop all previously pushed registers.
Two notable side effects of popping the registers are:
 1. The I bit is re-set to its previous status, re-enabling interrupts.
 2. The program counter is set to execute the next instruction in the current program.

Q2. Create a software Time delay for 1 second on a 68HC11 CPU with an 8MHz crystal.

ANSWER

An 8MHz crystal => 2 MHz E clock.

So each instruction cycle takes 0.5 microseconds = 0.5×10^{-6} seconds.

If we use a busy wait loop, we would need to waste 1000000 us or 2000000 E clock cycles.

We could code that in this manner

```
jsr    delay1s    * 5 cycles
```

delay1s:

```
    psha          * 3 cycles
```

```
    pshb          * 3 cycles
```

```
    pshx          * 4 cycles
```

```
    pshy          * 5 cycles
```

```
    ldy    #39999    * 5 cycles
```

* 25 cycles so far

loopMore

```
    nop          * waste 2 cycles
```

```
    idiv         * waste 41 clock cycles - modifies D & X
```

```
    dey          * 4 cycles
```

```
    bne    loopMore    * 3 cycles
```

* 50 cycles per loop for above code

```
    aba          * waste 1 cycle
```

```
    puly         * 6 cycles
```

```
    pulx         * 5 cycles
```

```
    pulb         * 4 cycles
```

```
    pula         * 4 cycles
```

```
    rts          * 5 cycles
```

* 25 cycles for above 6 lines (makes the arithmetic easier)

$2000000 - 25 - 25 = 1999950$

so execute inner loop 39999 times => $199950 + 25 + 25 =$

2000000 cycles = Near enough to 1 second.

(There are many possible solutions to this problem).

Q3. Write a program to find the largest (unsigned) byte in a block of data.
 It should store the largest byte in address \$0000
 Address \$0001 holds the block length in bytes (1..255)
 Address \$0002 is the address of the start block

```

LDAB  0      ; acc B becomes our loop counter
LDX   2      ; x is our pointer to the block of data
ldaa  0,x    ; fetch the first data item
staa  0      ; save it as our largest value so far

loop:
ldaa  0      ; fetch the largest value so far
cmpa  0,x    ; compare it to the data
bhs   not_bigger ; unsigned compare - Branch if Higher or Same
                ; we get here if the data is Above our previous largest value
ldaa  0,x    ; load into acca
staa  0      ; store new largest value in memory at location 0

not_bigger:
decb          ; decrease our loop counter (Zero flag is set when B becomes 0)
bne  loop    ; loop again, unless the Zero Flag is set
rts                ; finished so return from subroutine

```

Q4. What is a stack ? Use diagrams to explain .
 Describe in detail the main stack operations in a 68HC11
 Provide assembly example of stack usage
 What is a frame pointer

A stack is an area of memory set aside for temporary storage of data, variables and return addresses. The stack is pointed to by the Stack Pointer register. You Add data to the stack using the Push instruction. For example the psha instruction copies the data in Accumulator A to memory where the stack pointer is pointing to, then decrements the stack pointer. The pshx instruction pushes 2 bytes from the index register x, decrementing the stack twice.

The reverse operation is pull (or pop in intel parlance). The pula instruction first increments the stack pointer then copies the data from memory where the stack pointer is pointing, into accumulator A. The number of pushes and pulls must be equal within a subroutine or else the rts (return from subroutine) instruction will pull an invalid return address off the stack.

A good example is how parameters are passed using printf(“%d \n”, myInteger);

```

ldx   myInteger
pshx
ldx   #formatStatement
pshx
jsr   printf
pulx
pulx

```

Q5. Write a program to properly initialise the internal A/D sub-system in a 68HC11A1 to read 256 conversions per channel from analog channels AN0..AN3 scanning ans saving them in memory buffer starting at \$3000 then stop by calling buffalo with a SWI instruction.

```

* 68HC11Ax Analog to digital converter software
*
* Assemble with the command:
* as11 adc.asm -l > asm.lst
*
REGS EQU $1000 * Register stack
TOF EQU %10000000 * Timer overflow flag
N1 EQU 300
TFLG2 EQU $25
OPTION EQU $39
ADCTL EQU $30
ADR1 EQU $31
ADSET EQU %00000100 * A/D input on PE-4
CCF EQU %10000000 * Conversion complete flag
ADPU EQU %10000000 * A/D power-up bit
* * Monitor Equates
OUTLHF EQU $FFB2 Print left half
OUTRHF EQU $FFB5 Print right half
CRLF EQU $FFC4 Print CRLF
* * Memory Map Equates
CODE EQU $2000
DATA EQU $4000
STACK EQU $0040
ORG CODE
LDX #REGS
* Power up the A/D converter charge pump
BSET OPTION,x ADPU
JSR delay100us * must wait for charge pump voltage to rise
FOREVER * Beginning of LOOP-FOREVER
* Start the conversion SCAN=0, MULT=0
LDAA #ADSET * Select ADC input from Port E-4
STAA ADCTL,x * Signal ADC to start conversion
* Wait here until conversion complete flag is set to a 1
convwait:
BRCLR ADCTL,x CCF convwait
* Get the input and print it using the BUFFALO Monitor
LDAA ADR1, x * Get ADC converted input voltage value
TAB * save it (A) in B
*
JSR OUTLHF * output high nybble
TBA * restore it to A
JSR OUTRHF * output low nybble
JSR CRLF * output carriage return - line feed
spin1:
TST TFLG2,x
BPL spin1
*
DECB
BNE delay1
BRA FOREVER * loop forever
* This subroutine delays (a little over) 100 micro-seconds
delay100us:
psha * save Accumulator A
LDAA #40 * 40 loops for 200 clock cycles at 0.5us/clock
DELAYLOOP:
DECA *
BNE DELAYLOOP
pula * restore Accumulator A
RTS *

```