Homework 3

Name

1. **DSP Processor Concepts.** In a few sentences each, briefly describe the following and how/when it’s used in our lab.

   (a) Analog interface circuit

   (b) Anti-aliasing (input) filter

   (c) Smoothing (output) filter

   (d) Circular addressing
2. **Setting Sampling Rate.** Recall the following expressions,

\[
F_S = \frac{10 \times 10^6}{2AB} \quad (1)
\]

\[
F_{LP} = \frac{10 \times 10^6}{80A} \quad (2)
\]

modify the section of initialization code for the 'AC01 chip show below

\[
\ldots
\]

\[
\text{REG1 .set 112h}
\]

\[
\text{REG2 .set 212h}
\]

\[
\text{REG3 .set 300h}
\]

\[
\text{REG4 .set 405h}
\]

\[
\ldots
\]

to implement, as close as possible,

- sampling frequency, \(F_S\), of 10kHz and
- a low-pass filter cut-off frequency, \(F_{LP}\), of 4kHz

and because you may have rounded A and B, what is the resulting \(F_S\) and \(F_{LP}\)?
3. Sampling Rate.

(a) Many wireless LAN products today are based on the IEEE 802.11g standard. An IEEE 802.11g compliant transmission is about 20MHz wide. Sampling theorem tells us that we should sample this incoming signal at 40MHz or higher. Suppose that our DSP board can receive and transmit wireless signals, can we program the AIC to sample at 40MHz? Why or why not?

(b) Suppose that our AIC can sample at 40MHz. Recall that our DSP, the C542, runs at 40MHz and has 40MIPS. Suppose that we were to write a program to process the wireless signal, how many instruction cycles are available to process the current sample before the next sample arrives?
4. **Programming Exercise.** We will modify Experiment B’s `sample.asm` to implement a half wave rectifier. In other words, whenever an input sample from `trcv` is negative, output 0; if positive, output that sample.

Example:

- **Input sequence:** 0 2 4 2 0 -2 -4 -2 0 2 4 2 0 ...
- **Output sequence:** 0 2 4 2 0 0 0 0 0 2 4 2 0 ...

```
.mmregs
.setsect "text", 0x500,0
.setsect "data", 0x800,0
.setsect "vectors", 0x180,0
.sect "vectors"
.copy "vectors.asm"
.data
; ***** add .word's here if you need any
```
.text
start:  intm = 1          ; globally disable interrupts
  call AC01INIT
  pmst = #01a0h          ; set up iptr
  sp = #0ffah            ; init stack pointer.
  a = #0                ; initialize a to zero
  imr = #280h           ; ready to rcv int's

; ***** initialize pointers here if you need any

wait
  nop
  goto wait
; -------------- Receive Interrupt Routine ---------------------
XINT:
    b = trcv ; load acc b with input

    ; ***** your code starts here

    b = #OFFCh & b
    tdxr = b ; transmit the data.
    return_enable

    ; ***** your code ends here

; ----------------------- end ISR -----------------------------
.copy "ac01init.asm"
.end