SIGNAL PROCESSING

Note: 2 more files have been or will be added to your home directory for this lab. They are "adL4.com" and "lab4.com". This lab is a direct extension of the things you did in lab #3.

PURPOSE: This lab will introduce you to the use of the arithmetic instructions, arrays, various forms of indirect addressing, and a basic method of processing analog signals.

BACKGROUND INFORMATION:

In lab #3 you used a simulated analog to digital converter (ADC) to sample a sine wave and display it on the debugger screen. The amplitude of the signal that you sampled in lab #3 conveniently lay within the bounds of your display window, so that you could simply sample the data and display it on the screen. In real systems where computers are used to sample analog signals, it is not really this simple.

ADC's are normally configured so that they can sample a range of voltages. One common configuration is to set the range of allowed voltages to be between -5 volts and +5 volts. This means that the analog to digital converter is capable of sensing any voltage level within that range and generating a corresponding 16-bit number.

The ADC simulated in this lab uses 16-bit two's complement numbers to represent voltages between -5 and +5. For instance, if the value in the ADCDR were $8000, the corresponding input voltage must have been -5 volts. Similary, if the value in the ADCDR were $7FFF, the input voltage must have been +5 volts. Notice that if the input voltage is zero volts, the ADCDR will contain $0000.

Since the values in the ADCDR will now range from $8000 - $7FFF, you won't be able to simply take a sample value and display it in the display window as you did in lab #3. For this lab, you will have to scale the sampled value, and add some offset to it so that when you display the data, it will be centered in the display window.

Computer systems that sample analog signals are often used to process the signals in some way. Examples are: removing noise from the signal, interpreting information contained in the signal, filtering out parts of the signal (like a graphic equalizer on your stereo), or changing the shape of the signal. In this lab you will write a program that is to perform simple signal processing on an input sampled waveform. In particular, you will compute a weighted moving average of the input waveform in this lab.

NOTE: For those of you that have some signals and systems background, your program will perform a discrete convolution of the input sampled waveform and a raised-cosine low pass filter. You will study this next semester in EEAP 244.

A moving average is an average of only a certain portion of the data. Your program will compute the average of the first ten sampled data points (i.e it will average samples 1 -

10) and store that value in a buffer, i.e. a memory location. Then it will compute the average of samples 2 - 11 and store that value in the next location of the buffer. Your program will continue like this until it computes the last value, i.e. the average of samples 68-77. This is why it is called a moving average - because you are moving through the input data and computing an average of only a few samples.

You will not compute a normal average in this lab. Instead, it will be a weighted average. For example, imagine that you are computing a weighted average of the first four samples of your input data using the coefficients 1,2,2,1. It would be done in the following manner:

```
wtd avg = [(samp #1)(1)+(samp #2)(2)+(samp #3)(2)+(samp
#4)(1))]/4
```

where the middle two samples are weighted more heavily than the first and last sample. Your program will compute the weighted average using ten coefficients: 1, 2, 3, 5, 8, 8, 5, 3, 2, 1. It will generate an output waveform that consists of 68 data points, where each point is the weighted average of ten points in the input waveform.

CAUTION: If the quotient of a division using a DIVU is going to be greater than $FFFF (or outside $8000 to $7FFF for DIVS), then the V bit gets set, and the register is unchanged. This can cause unpredictable results if you divide after adding all ten values together. In fact, if you are using 10, you are pretty much guarnteed to have an overflow at some point in your program, even if you don't use a DIV instruction.

Reference: Tom Ulrich, “A Digital Filtering Primer,” The Computer Applications Journal, Issue #52, November 1994, pp. 14-18.

LAB ASSIGNMENT:

Your program should:

1. Sample 77 data points of the input waveform and display it in the display window as in lab #3.
2. Store the 77 data points of the sampled waveform in an array.
3. Use the weighted moving average of the stored sampled waveform to generate an output waveform that consists of 67 data points
4. Display the resulting waveform obtained from the averaging. It should be displayed in the same window as the input waveform, using a different character from the one used to display the input waveform.

Please Note: The debugger command file for lab 4 is different from the one you used for lab3. In order for your program to read the correct data for this lab, you must run the debugger using the following command:

```
db68k -c lab4.com <your_prog_name>
```

Questions:

1. Provide a flowchart and/or pseudocode if you didnt put one in the description section of your writeup.
2. Explain exactly how you performed the scaling and offsetting of both waveforms so that they displayed correctly.
3. How would you describe the shape of the input waveform ? The output waveform ?
4. If you assume that the range of allowed input voltages is between -5 and +5 volts, what voltage range does the input signal lie in ?
5. Describe the array(s) that you used in your program.
6. What forms of indirect addressing did you use ? What did you use them for ?

Lab Report Format and Grading:

Format (5 pts)
Reports should be in neat, concise form and turned in on time. The best reports will cover all vital information in detail without excessive verbiage on unnecessary details. All lab reports should be typed on 8-1/2 by 11 inch white paper. Macintosh computers can be easily accessed in Sears, Frieberger, and Smith computer lab, so it is hoped that all diagrams will be produced using a computer. You will be graded on how well you conform to this format.

Title Page (15 pts total)
Each laboratory report submitted should include on its title page:

a. Your name (typed) and signature
b. Lab assignment number and title
c. Abstract
d. Checkout signature (only for specified lab assignments)
e. Statement of authenticity. (labs will not be graded without this. See example title pg.)

Abstract (10 of the above 15 pts)
A 2 or 3 sentence description of the lab assignment and an overview of your solution to it.

Introduction (15 pts)
This should describe the problem in detail and the algorithm you used to solve the problem. The introduction should not describe your specific program.

Description of main body of your program (15 pts)
Discuss in detail exactly how your program works. This includes the use of pseudocode and flowcharts, if necessary. This portion of the lab report should also discuss input/output specifications, memory requirements, and register and memory locations used.

Description of any program subroutines (none in this case)
Should include a flow chart or pseudocode (if appropriate), input/output specifications, memory requirements, register and memory locations used.

Discussion (5 pts)
Explain any errors that exist (or existed) in your program. Also discuss the advantages and disadvantages of the algorithm used, ways it could be improved, and alternative methods of solving the problem.

Questions (5 pts each)
Answers to any questions posed in the lab assignment.

Conclusion (5 pts)

A 4 or 5 sentence summary of the report.

Program Listings (10 pts total)
Listings of source code and the assembly listing file (with the symbol table) should be included. When specified, you should also include sample output of the program.

Total: 100 pts.

Emulation Code Use

The following is an explanation of some of the parts of the emulation code written for the db68k debugger. In order to make use of these added routines, the first line of your program must be:

```
include display.s
```

In addition, you should run the debugger using the following command:

```
db68k -c lab4.com <your_file_name>
```

Using the Analog-Digital Converter

The Analog to Digital converter simulator has two word-sized registers associated with it. The A-D Converter Status Register, at memory location $11000, will contain a one anytime data is ready to be read in from the converter and a zero otherwise. The A-D Converter Data Register, at memory location $11002, will contain the data to be read in from the converter. Reading this memory location will clear the A-D Status Register. If an attempt is made to read data from the A-D converter when the status register is not a one, the A-D converter data register will contain zero.

Example:

```
ADCSR   EQU             $11000
ADCDR   EQU             $11002
LOOP            TST.W   ADCSR
                BEQ             LOOP
                MOVE.W  ADCDR,D0
```

The above code segment will poll the A-D status register until it is nonzero. When it becomes nonzero, the value in the A-D data register will be moved to data register D0.

Using the debugger display macro

The display macro will plot any character inside of the display window at a specified cursor location. To use the macro, do the following:

1. Put the X-coordinate into data register D0
2. Put the Y-coordinate into data register D1
3. Put the ASCII value of the character into data register D2.
4. JSR display

Example:

```
include display.s
```

```
MOVE.W  #4,D0
MOVE.W  #3,D1
MOVE.W  #40,D2
JSR     display
```

The above code segment will print the character with ASCII code $40 (@) in the fourth column, third row of the display window.

DIVU problems
Joseph S. Bui

Ok, I switched to using DIVU instead of DIVS, and I have the same problem. Here is a sample of some stuff:

```
* EEAP 282 - Fall 1994
* Lab #4 A/D Converter with weighted average
* Joseph S. Bui

* This is a scaling subrountine. 9363=$FFFF/7 (rounded up to an integer).
SCALE   DIVU    #9363,D1        ; scale number from $0..$FFFF to $0..$6
        RTS                     ; return from sub-routine

* This is the beginning of the weighted average loop
* All the data has been read into an array and it is already converted to
* unsigned values ranging from $0000 to $FFFF, with $0000 being the most
* positivei ADCDR value ($7FFF) and $FFFF map to the most negative ADCDR
* value ($8000). $4000 ends up being $3FFF and $C000 ends up being $BFFF,
* incidentally.
        MOVE    #4,D0           ; set (D0)=4 (X value and WEIGHT loop
counter)
        MOVE    #SYMBOL2,D2     ; set D2 to ASCII value of '+' for display.s
        LEA     ARRAY,A0        ; set A0 to starting address of ARRAY
        LEA     COEFS,A1        ; set A1 to starting address of COEFS
        CLR     D3              ; clear D3 (array pointer loop counter)
        CLR.L   D1              ; clear D1.L (Y value)
* Weighted average loop
WEIGHT  MOVE.W  (A0,D3),D4      ; set D4 to next array value
        MULU.W  (A1,D3),D4      ; weight D4
        ADD.L   D4,D1           ; put D4 in sum
        ADDQ    #2,D3           ; increment D3 by 2 (LOAD loop counter)
        CMP     #20,D3          ; set Z=1 if (D3)=20
        BNE     WEIGHT          ; if Z=0 go to LOAD
* This part above this works fine. It adds up the values, multiplying by
* appropriate cooeffcient. I've checked the results by hand for the first
* calculation.
        DIVU    #10,D1          ; average the 10 values in D1
* This DIVU line does not work. It does not reflect on the screen, nor does
* doing (M)emory (R)egister @D1 come up with the right value.
        JSR     SCALE           ; jump to value conversion routine
* Funny enough, the DIVU in the SCALE sub-routine does work. It comes up
* with correct values during the polling loop as well. It does this division
* correctly, but does not get the right answer, of course (garbage in,
* garbage out).
        JSR     display         ; jump to display conversion routine
        CLR     D3              ; clear D3 (LOAD loop counter)
        CLR.L   D1              ; clear D1 (Y value)
        ADDA.L  #2,A0           ; increase A0 by 1 word
        ADDQ    #1,D0           ; increment D0 (WEIGHT loop counter)
        CMP     #72,D0          ; set Z=1 if (D0)=68
        BNE     WEIGHT          ; if Z=0 go to WEIGHT
        END     MAIN
```

Everything works fine, except the DIVU #10,D1. The DIVU #9636,D1 works. I do not understand. I even decoded the

instuction that is displayed in the debugger: 82FC 000A
which is in fact immediate mode DIVU acting on D1.

Any ideas?
-Joe

Re: DIVU problems
Dennis Lee

Joseph S. Bui (jsb11@po.CWRU.Edu) wrote:

```
: Ok, I switched to using DIVU instead of DIVS, and I have the same problem.

<snip>    <snip>    <snip>

: WEIGHT        MOVE.W  (A0,D3),D4      ; set D4 to next array value
:       MULU.W  (A1,D3),D4      ; weight D4
:       ADD.L   D4,D1           ; put D4 in sum
:       ADDQ    #2,D3           ; increment D3 by 2 (LOAD loop counter)
:       CMP     #20,D3          ; set Z=1 if (D3)=20
:       BNE     WEIGHT          ; if Z=0 go to LOAD
: * This part above this works fine. It adds up the values, multiplying by
: * appropriate cooeffcient. I've checked the results by hand for the first
: * calculation.
:       DIVU    #10,D1          ; average the 10 values in D1
: * This DIVU line does not work. It does not reflect on the screen, nor does
: * doing (M)emory (R)egister @D1 come up with the right value.
:       JSR     SCALE           ; jump to value conversion routine

<more junk>

: Any ideas?
: -Joe
```

Did you remember to get rid of the remainder after the DIVU? Remember, the DIVU command places the quotient in the lower word of the data register and the remainder in the upper word (see pg 4-96 in Programmer's Reference Manual). In order to get rid of the remainder, use AND $0000FFFF,D1 after the DIVU (mask it). Nice program though ... wish I had thought of that ... I did it by brute force ... some 26 lines worth of MULUs, ADDs, and MOVEs (sigh). Hope this helps.

- Dennis

Re: DIVU problems
Dennis Lee

Joseph S. Bui (jsb11@po.CWRU.Edu) wrote:

```
: * This part above this works fine. It adds up the values, multiplying by
: * appropriate cooeffcient. I've checked the results by hand for the first
: * calculation.
:       DIVU    #10,D1          ; average the 10 values in D1
: * This DIVU line does not work. It does not reflect on the screen, nor does
: * doing (M)emory (R)egister @D1 come up with the right value.
: Joe
```

And are you sure it shouldn't be DIVU #38,D1? This question has been been brought up before in this newsgroup, but still ... if we divide the weighted average by #10, then the resulting value is no longer within the range $0000 to $FFFF (use ten $FFFF's as your data points, it's pretty obvious). If this is true, then dividing this result by $9363 makes no sense, right?

Dennis

Re: DIVU problems
Joseph S. Bui

In a previous article, dkl@happy.EEAP.CWRU.Edu (Dennis Lee) says:

```
>Joseph S. Bui (jsb11@po.CWRU.Edu) wrote:
>: * This part above this works fine. It adds up the values, multiplying by
>: * appropriate cooeffcient. I've checked the results by hand for the first
>: * calculation.
>:      DIVU    #10,D1          ; average the 10 values in D1
>: * This DIVU line does not work. It does not reflect on the screen, nor
does
>: * doing (M)emory (R)egister @D1 come up with the right value.
>       And are you sure it shouldn't be DIVU #38,D1?  This question has been
>been brought up before in this newsgroup, but still ... if we divide the
>weighted average by #10, then the resulting value is no longer within the
>range $0000 to $FFFF (use ten $FFFF's as your data points, it's pretty
>obvious).  If this is true, then dividing this result by $9363 makes no
>sense, right?
```

I think the 38 vs. 10 debate is something that is left up to us. Since the assignment contained an example in which there was a gain dependant on the coefficients, I chose to use 10. I had my lab checked out today, and I didn't get any negative reaction.

I also found my problem with that DIVU (or a DIVS, if you want to use that, which is what I ended up doing). If the quotient of the division is going to be greater than $FFFF (or outside $8000 to $7FFF for DIVS), then the V bit gets set, and the register is unchanged. This can cause unpredictable results if you divide after adding all ten values together. In fact, if you are using 10, you are pretty much guarnteed to have an overflow at some point in your program, even if you don't use a DIV instruction.
-Joe