Parallel Moves (brief)

Motorola Training Notes 3-1 through 3-16.

Note that in a parallel move involving address registers, one register must come from the upper file while the other from the lower file. For example,

```
move x:(r0),x0   y:(r4),y0
move x:(r4),x0   y:(r0),y0
```

are valid, however,

```
move x:(r0),x0   y:(r1),y0
```

would not be valid. Also in a parallel move, movements from x-memory must go/come to/from either register x0 or x1 while those from y-memory must go/come to/from either y0 or y1. (IS THIS REALLY TRUE?)

Example 1:

```
move x:(r4),x0
move a,y0
```

Can be replaced with

```
move x:(r4),x0   a,y0
```

Example 2:

```
add x0,a
move x:(r4)-,x1
move b,y0
```

Can be replaced with

```
add x0,a   x:(r4)-,x1   b,y0
```

Instruction Set I (brief)

Motorola Training Notes 4-4 through 4-32

Circular Queues (a.k.a. FIFOs, Circular Arrays, etc.)

Figure: FIFO (implemented as a circular queue)

Assume the current sample is stored in x0. Write a code to:
1) allocate data memory for a circular queue of length 4
2) initialize the queue (clear queue, initialize address and modifier registers, etc)
3) store the most recent 4 input samples. (assume this code is called everytime a sample moves from the A/D to x0)


```
;*********************************************************
;This code allocates memory for circular code of length 4
;*********************************************************
N equ 4

;X Memory
   org x:$0
input dsm N

;*********************************************************
;This code initializes the queue
;*********************************************************
   move #input,r0   ;point to input sample queue
   move #N-1,m0     ;set modifer reg for modulo addressing
   move #$0,x0      ;clear x0
   rep #N
   move x0,x:(r0)+  ;clear out queue

;*********************************************************
;This code stores a sample in x0 into a circular queue
;*********************************************************
   move x0,x:(r0) ;move sample in x0 into queue
   move (r0)-     ;point to new, oldest sample in queue (combine w/above)
```

Figure: circular queue in action

Simple Programs

PASS.ASM – Audio Pass Through Program

The starting point for programming DSPs will be a program typically called **`pass.asm`**. This program, which simply passes samples from the codec to the DSP and back, is included with virtually all DSP evaluation modules and development boards. We've already seen the basic outline:

```
initialize_DSP
initialize_codec
initialize_memory
main
      wait_for_input_sample
      process_sample
      send_output_sample
      goto main
```

Motorola Pass Pack

The **`pass.asm`** program is our starting point since all sample processing code will of course lie somewhere

between the code which passes samples from the codec to the DSP and the code which passes samples from the DSP to the codec.

For the DSP5630*x*EVM, this program is actually a collection of several files which we collectively call "Motorola Pass Pack." The Motorola Pass Pack for the DSP56302EVM is comprised of the following files:

```
pass.asm        intequ.asm
ada_init.asm    ioequ.asm
ada_equ.asm     vectors.asm
```

and are listed in Appendix A of the text. In addition, they can be found at

`http://www.ece.nmsu.edu/~pdeleon/Teaching/EE592/SampleMotorolaCodes`

Modified Pass Pack

In order to make the Motorola Pass Pack more flexible for our needs, we will make several file modifications (in bold) to the Motorola code. The "Modified Pass Pack" for the DSP56302EVM is comprised of the following files:

pass.asm	intequ.asm
pass.dat	ioequ.asm
ada_init.asm	vectors.asm
ada_equ.asm	**proginit.asm**
procster.asm	

and are listed in Appendix B of the text. In addition, they can be found at

`http://www.ece.nmsu.edu/~pdeleon/EE592/Freescale_DSP56K_Code.html`

Among the modifications are:

- Partitioning of external memory into two 16K banks of X- and Y-data memory beginning at addresses $010000
- Location of software stack in on-chip X-memory after last user data
- Addition of a **`pass.dat`** file where assembler directives for the program and data memory layouts can be stored
- Addition of a **`proginit.asm`** file where user code for initializaztion can be stored
- Addition of a **`procster.asm`** file where process_stereo subroutine is stored

All real-time DSP codes in this text will use the Modified Pass Pack v 1.7b

