

EE442 / EE592 Real-Time Digital Signal Processing
Quiz #1
 Allowed: *DSP56300Family Manual*, Textbook, Calculators

The following table may be useful for Problems 1 - 4

%0000	\$0	0	2 ⁸	256.0	2 ⁻⁸	0.00390625000000
%0001	\$1	1	2 ⁷	128.0	2 ⁻⁹	0.00195312500000
%0010	\$2	2	2 ⁶	64.0	2 ⁻¹⁰	0.00097656250000
%0011	\$3	3	2 ⁵	32.0	2 ⁻¹¹	0.00048828125000
%0100	\$4	4	2 ⁴	16.0	2 ⁻¹²	0.00024414062500
%0101	\$5	5	2 ³	8.0	2 ⁻¹³	0.00012207031250
%0110	\$6	6	2 ²	4.0	2 ⁻¹⁴	0.00006103515625
%0111	\$7	7	2 ¹	2.0	2 ⁻¹⁵	0.00003051757812
%1000	\$8	8	2 ⁰	1.0	2 ⁻¹⁶	0.00001525878906
%1001	\$9	9	2 ⁻¹	0.50000000000000	2 ⁻¹⁷	0.00000762939453
%1010	\$A	10	2 ⁻²	0.25000000000000	2 ⁻¹⁸	0.00000381469727
%1011	\$B	11	2 ⁻³	0.12500000000000	2 ⁻¹⁹	0.00000190734863
%1100	\$C	12	2 ⁻⁴	0.06250000000000	2 ⁻²⁰	0.00000095367432
%1101	\$D	13	2 ⁻⁵	0.03125000000000	2 ⁻²¹	0.00000047683716
%1110	\$E	14	2 ⁻⁶	0.01562500000000	2 ⁻²²	0.00000023841858
%1111	\$F	15	2 ⁻⁷	0.00781250000000	2 ⁻²³	0.00000011920929

1. Write the hexadecimal values for the following 24-bit fixed-point decimal fractions. The first one is done for you.

+0.5000000	\$400 000
+0.0468750	\$
-0.0234375	\$

2. Write the 24-bit fixed-point, decimal fractions (at least 7 digits of precision) for the following hexadecimal values. The first one is done for you.

+0.2500000	\$200 000
	\$4B0 000
	\$A30 000

3. Write the hexadecimal values for the following 56-bit fixed-point accumulator (decimal integer/fraction). The first one is done for you.

+128.0000000	\$40 000 000 000 000
+ 2.0390625	\$
- 7.0000000	\$

4. For the following hexadecimal value in accumulator b, determine the resulting 6-digit hexadecimal value in register x1, when the following instruction is executed:

move b,x1

b	x1
\$00 101010 800000	\$
\$FC 100000 000000	\$

5. When moving data from register x0 to accumulator b, the data from x0 will be found in bits (more than one answer may be correct)

- (a) 23 to 0, bits 24 to 55 are not affected
- (b) 47 to 24, bits 48 to 55 are either all zeros or all ones, and bits 23 to 0 are zeros
- (c) 23 to 0, bits 24 to 55 are undefined
- (d) 47 to 24, bits 48 to 55 are sign extended, and bits 23 to 0 are zeros
- (e) 47 to 24, bits 48 to 55 are zeros, and bits 23 to 0 are zeros
- (f) None of the above

6. Write a short code which does the following steps:

- Step 1) multiplies the data in X:\$00 with the data in Y:\$01
- Step 2) multiplies the data in X:\$01 with the data in Y:\$00
- Step 3) adds the results from Step 1 and Step 2

7. Fill in the missing pieces of code and comments denote with “?” in order to compute the following equation

$$[(0.2+0.3) \times 0.4]^2$$

```

move  #0.2,?           ;a = 0.2
move  #0.3,?           ;?
add   x0,?             ;a = 0.2+0.3
move  ?,?              ;x0 = a
move  ?,y0             ;y0 = 0.4
mpyr  ?,?,?           ;a = (0.2+0.3)*0.4
move  ?,?              ;?
mpy   x0,x0,?         ;a = [(0.2+0.3)*0.4]^2

```

For the following questions, determine the register, accumulator, and memory states after execution of the instruction. Put 6 hexadecimal digits in each box (or 2 in the case of a2 and b2). If you leave the box blank, we assume the state does not change. If the instruction cannot be executed, write "CANNOT BE EXECUTED"

8. `move x:(r3)+n3,x0`

Before Execution

x1	x0	r2	n2	
030201	908070	000100	000001	
y1	y0	r3	n3	
030203	908090	000101	000001	
a2	a1	a0	r4	n4
13	579BDF	02468A	000102	000002
b2	b1	b0	r5	n5
24	68ACE0	13579B	000103	000002

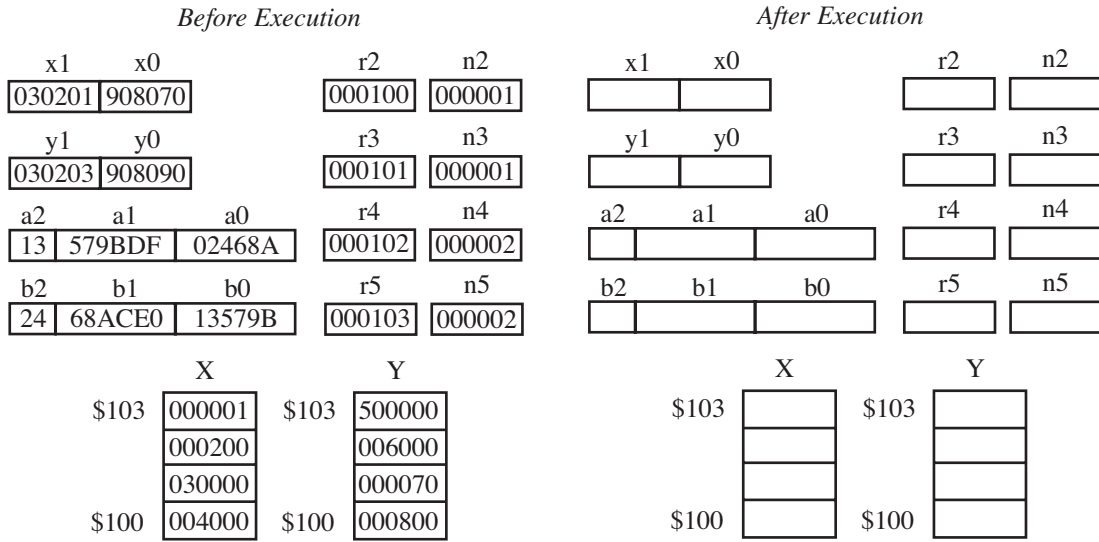
After Execution

x1	x0	r2	n2	
y1	y0	r3	n3	
a2	a1	a0	r4	n4
b2	b1	b0	r5	n5

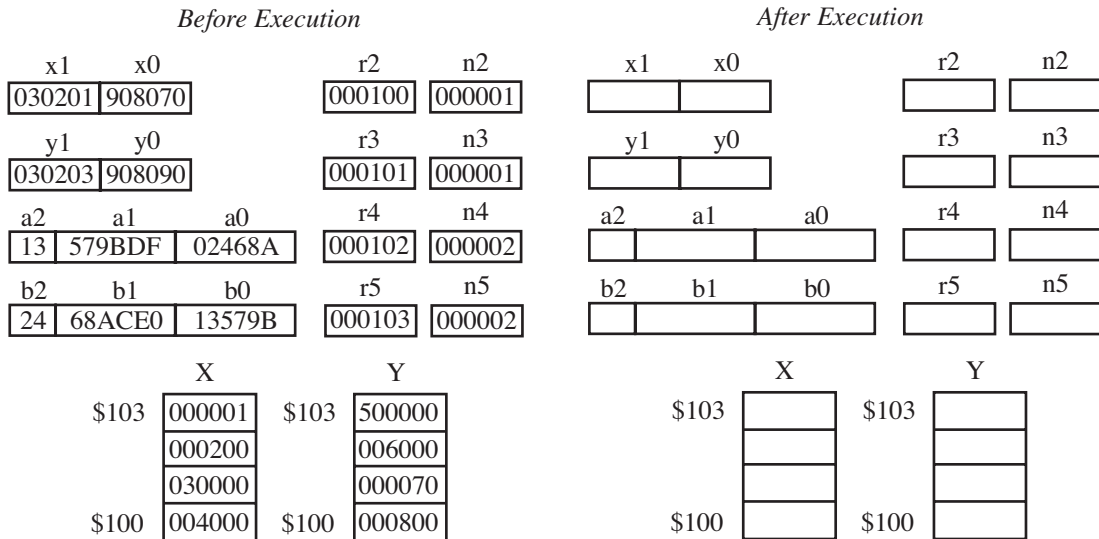
	X		Y
\$103	000001	\$103	500000
	000200		006000
	030000		000070
\$100	004000	\$100	000800

	X		Y
\$103		\$103	
\$100		\$100	

9. `move y:(r4)+n4,a`



10. `move b,x:(r2)-`



This page EE592 only

11. (+20 points) For the following code, determine the register, accumulator, and memory states after assembly and execution of the code. If you leave a box blank, we assume the state does not change. The next page contains some information regarding assembler directives which you may find useful.

```

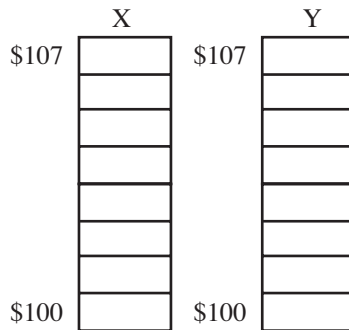
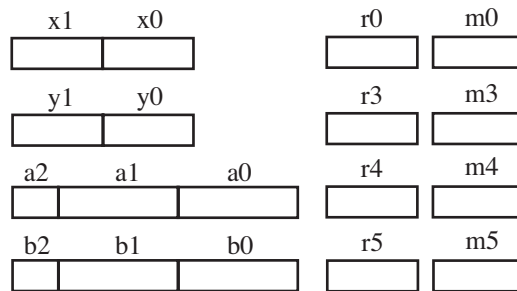
N      equ    3

      org x:$101
siga  dsm    N
      org x:siga
      dc     0.125, 0.125, 0.25

      org y:$102
sigb  dc     N
      org y:sigb
      dc     0.25, 0.25, 0.125

      org p:$100
      move  #siga,r0
      move  #N-1,m0
      move  #sigb,r5
      clr   a      x:(r0)+,x1      y:(r5)+,y1
      rep   #N-2
      mac   x1,y1,a      x:(r0)+,x1      y:(r5)+,y1
    
```

After Execution



Define Constant (DC)

```
<label>      DC      <arg>[,<arg>,...,<arg>]
```

The DC directive allocates and initializes a word of memory for each argument, <arg>. Arguments may be a numeric constant, a single or multiple character string constant, a symbol, and/or an expression. The DC directive may have one or more arguments separated by commas. Multiple arguments are stored in successive address locations.

<label>, if present, will be assigned the value of the runtime location counter at the start of the directive processing. Integer arguments are stored “as is”; floating point numbers are converted to binary values.

Example:

```
COEFS      dc      0.1,0.2,0.3,0.4,0.5      ;coefficients stored in memory
```

Define Storage Modulo (DSM)

```
<label>      DSM      <expression>
```

The DSM directive reserves a block of memory the length of which in words is equal to the value in <expression>. If the runtime location counter is not zero, this directive first advances the runtime location counter to a base address that is a multiple of 2^k , where $2^k \geq \text{<expression>}$. Next the runtime location counter is advanced by the value of the integer expression in the operand field. <expression> can have any memory space attribute. The block of memory reserved is not initialized to any given value.

<label>, if present, will be assigned the value of the runtime location counter at the start of the directive processing.

Example:

```
CIRCULAR_BUFFER  dsm      16      ;16 words allocated for circ. Buffer
```

Originate (ORG)

```
ORG      <rms>:[<expr1>]
```

The ORG directive is used to specify addresses and to indicate memory space and mapping changes. Two of the parameters used with the ORG directive are:

<rms>---which memory space (X, Y, L, P, or E) will be used as the runtime memory space. If the memory space is L, any allocated datum with a value greater than the target word size will be extended to two words; otherwise it is truncated. If the memory space is E, then depending on the memory space qualifier, any generated words will be split into bytes, one byte per word, or a 16/8-bit combination.

<exp1>---initial value assigned to the runtime counter used as the <rlc>. If <exp1> is an absolute expression the assembler uses the absolute location counter. If <exp1> is not specified, then the last value and mode that the counter had will be used.

Example:

```
ORG      x:$00000a      ;lay things out in x-memory starting at $000a
```