NAME: _____ GRADE: _____

EE442 / EE592 Real-Time Digital Signal Processing
Quiz #1
Allowed: *DSP56300Family Manual*, 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
+0.0937500	$
-0.9375000	$

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
	$120 000
	$ED0 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
+ 48.0312500	$
- 23.3750000	$

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
$AB 123000 000000	$

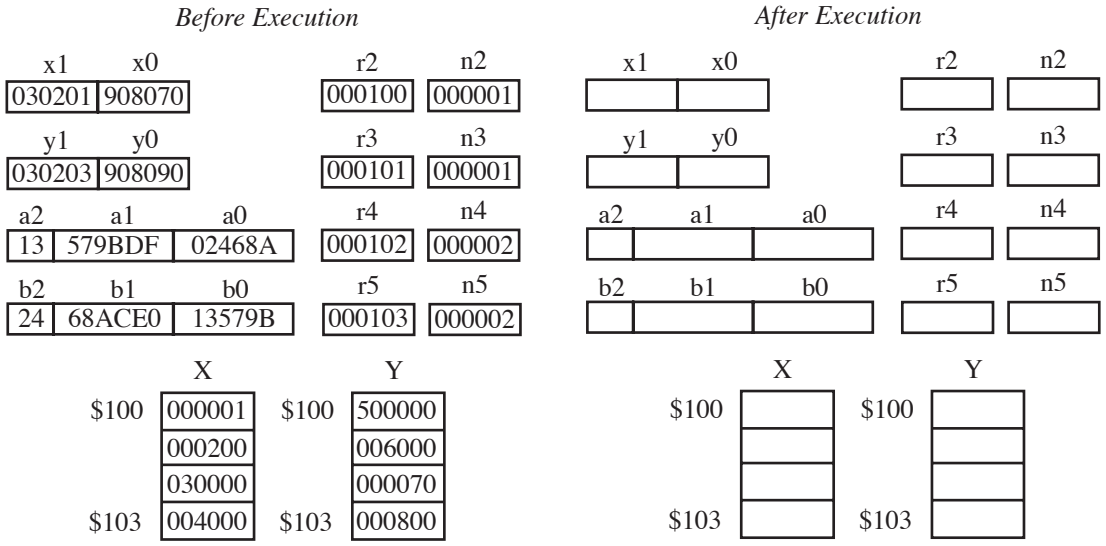
5. Write a short code to compute the following equation (as a programmer you may not assist the DSP with doing the calculation but you may rearrange the calculation into a more convenient form)

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

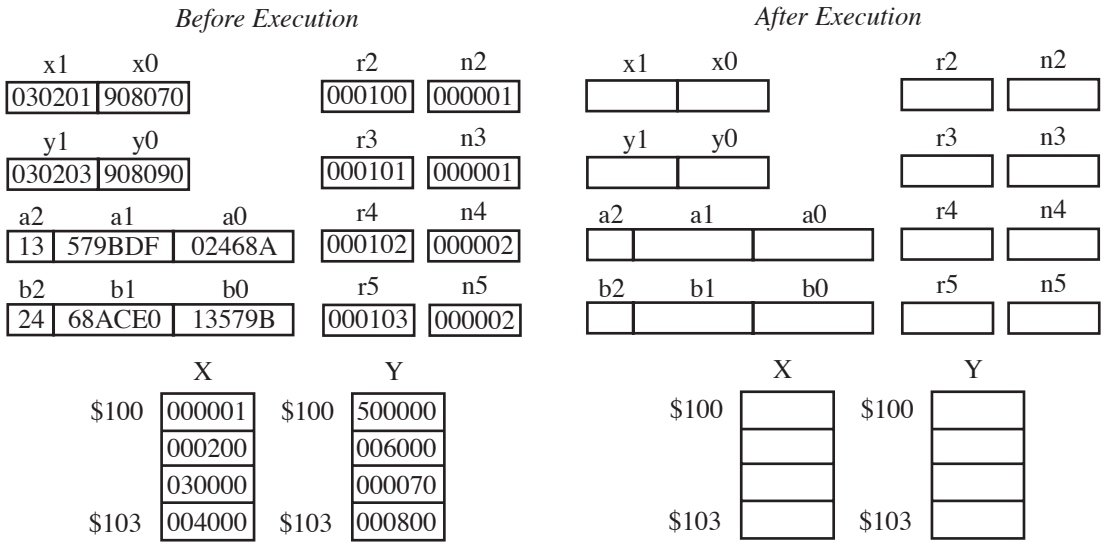
Five or fewer lines of correct code will yield full credit; six or more lines of correct code will yield half credit. Hint: use the distributive property, the MPYI/MPYRI and MACI/MACRI instructions, and take care with rounding.

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"

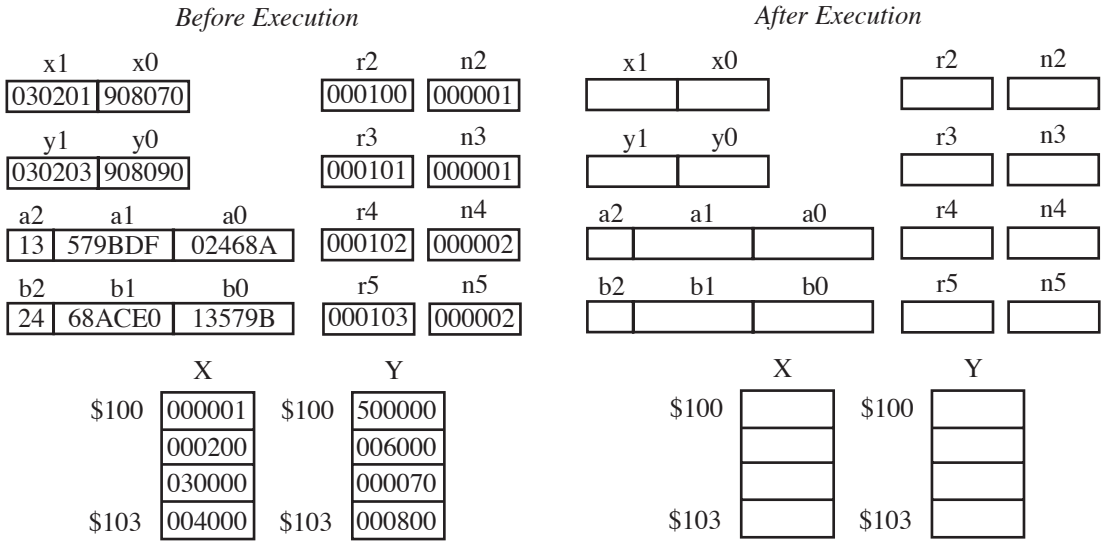
6. `move x:(r5-n5),y0`



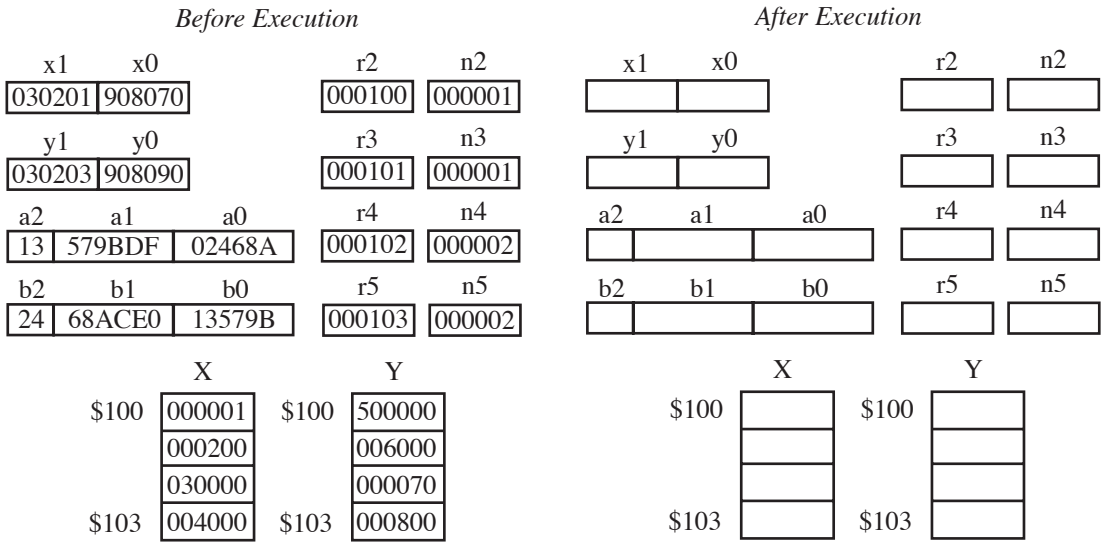
7. `move y1,y:(r4)+n4`



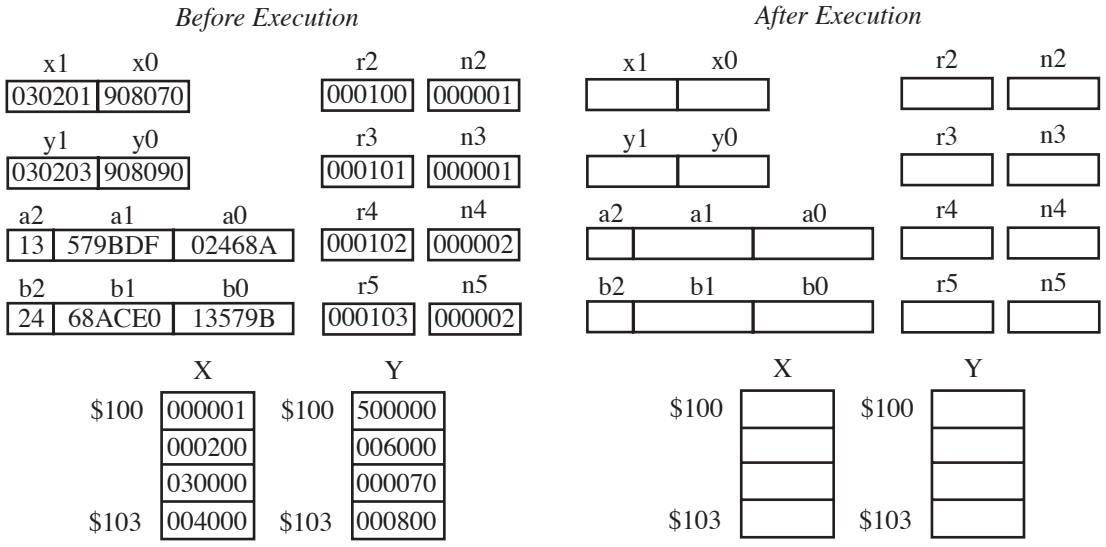
8. `move b,y:(r5)-`



9. `move a1,x:+(r2)`



10. `move x:(r4)+,y0`



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.

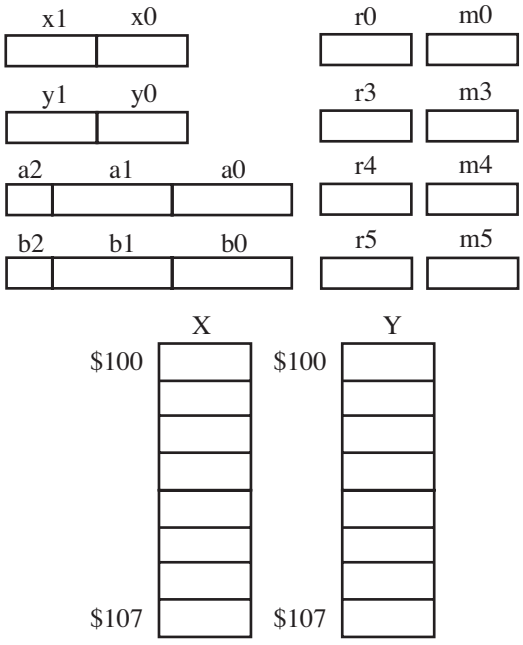
```
N      equ   4
Nby2   equ   N/2

       org x:$101
veca   dsm   N
       org x:veca
       dc    0.25, 0.0, 0.0, 0.0

       org y:$102
vecb   dc    N
       org y:vecb
       dc    0.5, 0.6, 0.7, 0.8

       org p:$100
       move  #veca,r3
       move  #N-1,m3
       move  #vecb,r4
       clr   a     x:(r3)+,x0        y:(r4)+,y0
       rep   #Nby2
       mac   x0,y1,a     x:(r3)+,x0        y:(r4)+,y0
```

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 ≥ <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
```