Seat _____ **Serial Number _____**

CSE 30 Final Winter 2018

_____ (Signature)

_____ (Last Name)

cs30x _____

Student ID _____

_____ (First Name)

Affirm your adherence to the principle of Academic Integrity by writing the following statement:
"**I Excel with Integrity**"

Page 1	**_____**	**(33 points)**
Page 2	**_____**	**(27 points)**
Page 3	**_____**	**(15 points)**
Page 4	**_____**	**(10 points)**
Page 5	**_____**	**(22 points)**
Page 6	**_____**	**(21 points)**
Page 7	**_____**	**(44 points)**
Page 8	**_____**	**(34 points)**
Page 9	**_____**	**(7 points)**
Page 10	**_____**	**(17 points)**
Page 11	**_____**	**(11 points)**
Page 12	**_____**	**(32 points)**
Total	**_____**	**(273 points)**

260 points = 100%
(5% Extra Credit - 13 points Extra Credit)

By filling in the above and signing my name, I confirm I will complete this exam with the utmost integrity and in accordance with the Policy on Integrity of Scholarship.

This exam is to be taken by yourself with closed books, closed notes, no electronic devices.
You are allowed both sides of an 8.5"x11" sheet of paper handwritten by you.

Page 1 (33 points)

Convert **0xFB99** (2's complement, 16-bit word) to the following. (6 points)

binary_____ (straight base conversion to binary)

octal_____ (straight base conversion)

decimal_____ (convert to signed decimal)

Convert **264** to the following (assume 16-bit word). **Express answers in <u>hexadecimal</u>.** (3 points)

sign-magnitude_____

1's complement_____

2's complement_____

Convert **-685** to the following (assume 16-bit word). **Express answers in <u>hexadecimal</u>.** (6 points)

sign-magnitude_____

1's complement_____

2's complement_____

Rt-Lt Rule

Using the C Rt-Lt Rule, define a variable named fubar that is a multi-dimensional array of 21 rows and 12 columns where each element is a pointer to a function that takes a single argument of type pointer to char and returns a pointer to an array of 17 elements where each element is of type pointer to struct foobar. (6 points)

Indicate what the condition code bits are when adding the following 8-bit 2's complement numbers. (12 points)

```
 10011101        01010110        10110100
+10101001       +00111011       +01001100
---------       ---------       ---------
```

```
  N   Z   C   V       N   Z   C   V       N   Z   C   V
-----------------   -----------------   -----------------
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |
-----------------   -----------------   -----------------
```

Page 2 Branching (27 points)

Translate the C code below into the equivalent **unoptimized** ARM Assembly code. Just perform a direct translation – no optimizations. Assume the local variables `a` and `b` have been allocated on the stack properly as discussed in class. This is not an entire function - just translate the code fragment using the standard local variable stack allocation scheme we used this quarter. Use the control flow specified in class.

C

```
int a;       /* First local variable on the stack */
int b;       /* Second local variable on the stack */
/* Some other code here assigning values to a and b. */
/* Translate just this code below.
   Use r3 for manipulating var a; Use r2 for manipulating var b. */
while ( a > b ) {
  if ( (b % 42) == 0 ) {
    a = b-- + 42;
  } else {
    b = ++a - 42;
  }
  b = b + 24;
}
/* Some other code here */
```

Note this is unoptimized:
Always write values back to their allocated stack space on any assignment.
Always read values from their allocated stack space for every variable access.
Don't assume a value is still in a register from a previous statement.

Possible assembly instructions:

```
A1) ldr  r3, [fp, -8]
A2) ldr  r2, [fp, -12]
A3) str  r3, [fp, -8]
A4) str  r2, [fp, -12]
A5) ldr  r0, [fp, -8]
A6) ldr  r1, [fp, -8]
A7) ldr  r0, [fp, -12]
A8) ldr  r1, [fp, -12]

B1) cmp  r3, r2
B2) cmp  r0, 0
B3) cmp  r1, 0
B4) cmp  r2, 0
B5) cmp  r3, 0

C1) ble  loop
C2) ble  else
C3) ble  endif
C4) ble  endloop

D1) bgt  loop
D2) bgt  else
D3) bgt  endif
D4) bgt  endloop

E1) blt  loop
E2) blt  else
E3) blt  endif
E4) blt  endloop

F1) bge  loop
F2) bge  else
F3) bge  endif
F4) bge  endloop
```

Possible assembly instructions:

```
G1) b    loop
G2) b    else
G3) b    endif
G4) b    endloop

H1) bl   div
H2) bl   mod
H3) bl   rem
H4) bl   quot

I1) bne  loop
I2) bne  else
I3) bne  endif
I4) bne  endloop

J1) beq  loop
J2) beq  else
J3) beq  endif
J4) beq  endloop

K1) add  r3, r3, 42
K2) add  r3, r2, 42
K3) add  r2, r2, 24
K4) add  r2, r2, 1
K5) add  r3, r3, 1

L1) sub  r2, r2, 42
L2) sub  r2, r3, 42
L3) sub  r3, r3, 24
L4) sub  r2, r2, 1
L5) sub  r3, r3, 1

M1) mov  r0, 42
M2) mov  r1, 42
M3) mov  r2, 42
M4) mov  r3, 42
```

ARM ASSEMBLY

Put the code corresponding to the correct instruction in the correct order here:

```
        _____
        _____
        _____
        _____
loop:
        _____
        _____
        _____
        _____
        _____
        _____
        _____
        _____
        _____
        _____
        _____
else:
        _____
        _____
        _____
        _____
        _____
endif:
        _____
        _____
        _____
        _____
        _____
        _____
        _____
endloop:
```

Page 3 (15 points)

What is the value in r0 after each statement is executed? **Express your answers as 8 hexadecimal digits.**
(All 32 bits. Be sure to specify any leading or trailing zeros.)

```
ldr  r0, =0xCAFEBABE
ldr  r1, =0x87654321
and  r0, r0, r1
```

Value in r0 is _____ (2 points)

```
ldr  r0, =0xCAFEBABE
asr  r0, r0, 9
```

Value in r0 is _____ (2 points)

```
ldr  r0, =0xCAFEBABE
lsl  r0, r0, 11
```

Value in r0 is _____ (2 points)

```
ldr  r0, =0xCAFEBABE
ldr  r1, =0x????????
eor  r0, r0, r1                  ! Value in r0 is now 0x87654321
```

Value set in r1 must be this bit pattern _____ (3 points)

```
ldr  r0, =0xCAFEBABE
ldr  r1, =0x87654321
orr  r0, r0, r1
```

Value in r0 is _____ (2 points)

```
ldr  r0, =0xCAFEBABE
lsr  r0, r0, 6
```

Value in r0 is _____ (2 points)

Fill in the blanks for this program to determine whether the system this code is executing on is a Big-Endian or a Little-Endian architecture. (0 or 2 points for both correct)

```
#include <stdio.h>

int main( void ) {
  int word = 0x41;  // 0x41 = 'A'

  if ( (* (char *) &word) == 'A' )

    printf( "_____-Endian\n" );

  else if ( (* ((char *) &word) + 3) == 'A' )

    printf( "_____-Endian\n" );

  return 0;
}
```

Page 4 (10 points) Given **main.s** and **fubar.s**, what gets printed when executed? Yes … Draw stack frames!

```
        .cpu    cortex-a53                  /* main.s */
        .syntax unified

        .section        .rodata
code:   .byte   0x43, 0x6A, 0x61, 0x53, 0x4A, 0x76, 0x45, 0x71, 0x61, 0x33, 0x51, 0x4A
        .byte   0x30, 0x74, 0x2D, 0x2D, 0x4E, 0x3E, 0x52, 0x62, 0x2D, 0x75, 0x53, 0x2B
        .byte   0x6C, 0x73, 0x2B, 0x65, 0x42, 0x43, 0x73, 0x00, 0x00, 0x00, 0x00, 0x33

        .global main
        .text
        .align  2

main:
        push    {fp, lr}
        add     fp, sp, 4

        ldr     r0, =code
        mov     r1, 0
        bl      fubar

        sub     sp, fp, 4
        pop     {fp, pc}
```

```
        .cpu    cortex-a53                  /* fubar.s */
        .syntax unified

        .section        .rodata
fmt:
        .asciz  "%c"

        .global fubar
        .text
        .align  2

fubar:
        push    {fp, lr}
        add     fp, sp, 4

        sub     sp, sp, 8       @ save space for local var of type char

        sub     sp, sp, 8       @ save space for formal params 1 and 2
        str     r0, [fp, -16]
        str     r1, [fp, -20]

        ldr     r1, [fp, -20]
        add     r1, r1, 1
        str     r1, [fp, -20]

        ldr     r0, [fp, -16]
        cmp     r0, 0
        beq     end

        ldrb    r3, [r0, r1]
        cmp     r3, 0
        beq     end

        strb    r3, [fp, -5]

        ldr     r0, [fp, -16]
        ldr     r1, [fp, -20]
        add     r1, r1, 2
        bl      fubar

        ldr     r0, =fmt
        ldrb    r1, [fp, -5]
        sub     r1, r1, 1
        bl      printf

end:

        sub     sp, fp, 4
        pop     {fp, pc}
```

What gets printed? _____

Page 5 (22 points)

Here is a C function that allocates a few local variables, performs some assignments and returns a value. Don't worry about any local variables not being initialized before being used. Just do a direct translation. Assume struct foo is defined as:

```
struct foo {
  int   s1[2];
  char  s2;
  int   s3;
  char  s4[4];
};
```

C

```
int
fubar( int a, int b ) {
    int *      local_stack_var1;
    int        local_stack_var2;
    struct foo local_stack_var3;

    local_stack_var3.s4[0] = local_stack_var_3.s2;          /* stmt 1 */
    local_stack_var2 = a - local_stack_var3.s3++;           /* stmt 2 - Use r2 to hold result of sub */
    local_stack_var3.s1[1] = local_stack_var3.s1[0] + b;    /* stmt 3 */

    return ( *++local_stack_var1 + 42 );                    /* stmt 4 */
}
```

Write the equivalent full **unoptimized** ARM assembly language module to perform the equivalent. **All local variables are allocated on the Runtime Stack.**

Treat each statement independently. Use the instructions provided below.

Possible assembly instructions:

```
A1) sub  fp, fp, 4
A2) sub  fp, sp, 4
A3) sub  sp, fp, 4
A4) add  sp, fp, 4
A5) add  fp, sp, 4
A6) add  fp, fp, 4

B1) sub  sp, sp, 28
B2) sub  sp, sp, 32
B3) sub  sp, sp, 36
B4) sub  sp, sp, 40

C1) ldr  r3, [fp, -4]
C2) ldr  r3, [fp, -8]
C3) ldr  r3, [fp, -12]
C4) ldr  r3, [fp, -16]
C5) ldr  r3, [fp, -20]
C6) ldr  r3, [fp, -24]
C7) ldr  r3, [fp, -28]
C8) ldr  r3, [fp, -32]
C9) ldr  r3, [fp, -36]

D1) str  r3, [fp, -4]
D2) str  r3, [fp, -8]
D3) str  r3, [fp, -12]
D4) str  r3, [fp, -16]
D5) str  r3, [fp, -20]
D6) str  r3, [fp, -24]
D7) str  r3, [fp, -28]
D8) str  r3, [fp, -32]
D9) str  r3, [fp, -36]

E1) sub  r2, r0, r3
E2) sub  r2, r1, r3
E3) sub  r2, r2, r3

F1) str  r2, [fp, -4]
F2) str  r2, [fp, -8]
F3) str  r2, [fp, -12]
F4) str  r2, [fp, -16]
F5) str  r2, [fp, -20]
F6) str  r2, [fp, -24]
F7) str  r2, [fp, -28]
F8) str  r2, [fp, -32]
F9) str  r2, [fp, -36]

G1) add  r3, r3, 1
G2) add  r3, r3, 2
G3) add  r3, r3, 3
G4) add  r3, r3, 4
```

Possible assembly instructions:

```
H1) ldrb r3, [fp, -4]
H2) ldrb r3, [fp, -8]
H3) ldrb r3, [fp, -12]
H4) ldrb r3, [fp, -16]
H5) ldrb r3, [fp, -20]
H6) ldrb r3, [fp, -24]
H7) ldrb r3, [fp, -28]
H8) ldrb r3, [fp, -32]
H9) ldrb r3, [fp, -36]

I1) add  r3, r3, r0
I2) add  r3, r3, r1
I3) add  r3, r3, r2
I4) add  r3, r3, r3

J1) strb r3, [fp, -4]
J2) strb r3, [fp, -8]
J3) strb r3, [fp, -12]
J4) strb r3, [fp, -16]
J5) strb r3, [fp, -20]
J6) strb r3, [fp, -24]
J7) strb r3, [fp, -28]
J8) strb r3, [fp, -32]
J9) strb r3, [fp, -36]

K1) pop  {fp, lr}
K2) pop  {fp, pc}
K3) push {fp, lr}
K4) push {fp, pc}

L1) add  r0, r3, 42
L2) add  r1, r3, 42
L3) add  r2, r3, 42
L4) add  r3, r3, 42

M1) ldr  r0, [r0]
M2) ldr  r1, [r1]
M3) ldr  r2, [r2]
M4) ldr  r3, [r3]

N1) .text
N2) .bss
N3) .section .rodata
N4) .data

O1) .align 1
O2) .align 2
O3) .align 3
O4) .align 4
```

Put the code corresponding to the correct instruction in the correct order here:

```
        .cpu    cortex-a53
        .syntax unified
        .global fubar

        _____

        _____

fubar:

        _____

        _____

        _____

-------------------------------

        _____   ! stmt 1

        _____

-------------------------------

        _____   ! stmt 2

        _____

        _____

        _____

        _____

-------------------------------

        _____   ! stmt 3

        _____

        _____

-------------------------------

        _____   !stmt 4

        _____

        _____

        _____

        _____

        _____

        _____
```

Page 6 (21 pts)

Given the following program, order the line numbers so the memory addresses are printed from smallest to largest when compiled and run on our ARM pi-cluster Linux system. For example, which line will print the lowest memory address, then the next higher memory address, etc. up to the highest memory address? This is identifying where the different parts of a C program live in the run time environment.

```
#include <stdio.h>
#include <stdlib.h>

void foo1( int *, int ); /* Function Prototype */
void foo2( int, int *, int, int, int, int ); /* Function
Prototype */

int a;

int main( int argc, char *argv[] ) {

   int b;
   double c;

   foo2( a, &b, 42, -99, 17, 37 );

/*  1 */ (void) printf( "1: argc --> %u\n", &argc );
/*  2 */ (void) printf( "2: c --> %u\n", &c );
/*  3 */ (void) printf( "3: argv --> %u\n", &argv );
/*  4 */ (void) printf( "4: malloc --> %u\n", malloc(50) );
/*  5 */ (void) printf( "5: b --> %u\n", &b );
}

void foo1( int *d, int e ) {

   static struct foo {int a; int b;} f = { 1, 2 };
   int g;

/*  6 */ (void) printf( "6: f.b --> %u\n", &f.b );
/*  7 */ (void) printf( "7: d --> %u\n", &d );
/*  8 */ (void) printf( "8: e --> %u\n", &e );
/*  9 */ (void) printf( "9: f.a --> %u\n", &f.a );
/* 10 */ (void) printf( "10: foo2 --> %u\n", foo2 );
/* 11 */ (void) printf( "11: g --> %u\n", &g );
}

void foo2( int h, int *i, int j, int k, int l, int m ) {

   int n = 411;
   int o[3];

   foo1( i, j );

/* 12 */ (void) printf( "12: o[1] --> %u\n", &o[1] );
/* 13 */ (void) printf( "13: h --> %u\n", &h );
/* 14 */ (void) printf( "14: a --> %u\n", &a );
/* 15 */ (void) printf( "15: i --> %u\n", &i );
/* 16 */ (void) printf( "16: o[0] --> %u\n", &o[0] );
/* 17 */ (void) printf( "17: j --> %u\n", &j );
/* 18 */ (void) printf( "18: l --> %u\n", &l );
/* 19 */ (void) printf( "19: k --> %u\n", &k );
/* 20 */ (void) printf( "20: m --> %u\n", &m );
/* 21 */ (void) printf( "21: n --> %u\n", &n );
}
```

_____ this line # will print the smallest memory address

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____ this line # will print the largest memory address

Page 7 (44 points)

Convert 124.875 (decimal fixed-point) to binary fixed-point (**binary**) and single-precision IEEE floating-point (**hexadecimal**) representations.

binary fixed-point _____ (2 points)

IEEE floating-point _____ (4 points)

Convert 0xC346C000 (single-precision IEEE floating-point representation) to fixed-point decimal.

fixed-point decimal _____ (6 points)

Translate the following instructions into ARM machine code. Use **hexadecimal** values for your answers. If an instruction is a branch, specify the + or - number of instructions away for the target (vs. a Label).

```
adds   r3, r2, 5
```
_____ (5 points)

```
bge    Label
```
_____ (5 points)
(where Label is 3 instructions above the bge Label instruction)

Translate the following ARM machine code instructions into ARM assembly instructions.

```
0xE51B3008
```
_____ (5 points)

```
0xE1A01787
```
_____ (5 points)

If an odd-ball computer had 900MB of memory, how many bits would be needed in an address register to address any byte in this system?

_____ (1 point)

Which part of the entire compilation sequence clear through to program execution is responsible for:

____ expanding # directives

____ reporting syntax errors

____ getting the executable image from disk into memory

____ translating assembly source code into object target code

____ reporting this error -- undefined reference to 'foo' … ld returned 1 exit status

____ creating an executable image from multiple object files

____ translating C source code into assembly target code

____ having the operating system report a segmentation fault (cored dumped) message

____ resolving undefined external symbols with defined global symbols across modules

____ ensuring the bss segment is set up and zero-filled

____ reporting multiply-defined symbols across multiple files

A) C Preprocessor
B) C Compiler
C) Assembler
D) Linkage Editor
E) Loader
F) Program Execution

Page 8 (34 points)

For the following program fragment, specify in which C runtime area/segment each symbol will be allocated or pointing:

```
static int a = -17;                                  _____ ( a )

int b;                                               _____ ( b )

static int c = 37;                                   _____ ( c )

int d;                                               _____ ( d )

int foo( int e ) {            _____ ( e )            _____ ( foo )

    static double f = 42.42;                         _____ ( f )

    static int *g;                                   _____ ( g )

    g = (int *) malloc( e );                         _____ (where g is pointing)

    int (*h)(int) = foo;      _____ ( h )            _____ (where h is pointing)

    double i = 30.30;                                _____ ( i )

        ...
}
```

Fill in the letter corresponding to the correct scoping/visibility for each of the variables:

A) Global across all modules/functions linked with this source file (global scope).
B) Global just to this source file (file scope).
C) Local to function foo() (block scope).

a _____
b _____
c _____
d _____
e _____
f _____
g _____
h _____
i _____
foo _____

Fill in the letter corresponding to the correct lifetime for each of the variables:

A) Exists from the time the program is loaded to the point when the program terminates.
B) Exists from the time function foo() is called to the point when foo() returns.

a _____
b _____
c _____
d _____
e _____
f _____
g _____
h _____
i _____
foo _____

If the function foo() above is called 10 times, indicate how many times will `f` be initialized? _____

If the function foo() above is called 10 times, indicate how many times will `i` be initialized? _____

Page 9 (7 points)

Specify the full 32 bit hex values after each line has been fully executed.

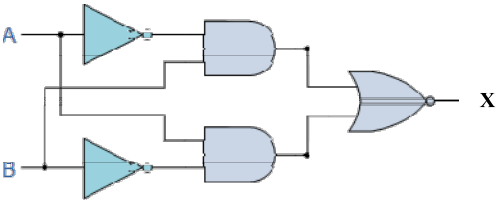
All answers need to be in the form 0xXXXXXXXX

```
        .cpu    cortex-a53
        .syntax unified

        .global main

        .section        .rodata
fmt:
        .asciz  "0x%08X\n"

        .data
c:      .byte   0xBB

        .align  1
s:      .hword  0xDEAD          @ Remember: stored as little-endian in memory

        .align  2
i1:     .word   0x24680ACE      @ Remember: stored as little-endian in memory
i2:     .word   0x24680ACE      @ Remember: stored as little-endian in memory

        .text
        .align  2
main:
        push    {fp, lr}
        add     fp, sp, 4

        ldr     r0, =i1         @ load the 32-bit addr of i1 into r0
        ldr     r1, =s          @ load the 32-bit addr of s into r1


        ldrsh   r2, [r1]          _____________________________ Hex value in r2


        strb    r2, [r0, 2]       _____________________________ Hex value in word labeled i1
                                                                  (memory so little-endian)

        lsr     r2, r2, 4         _____________________________ Hex value in r2


        strh    r2, [r0]          _____________________________ Hex value in word labeled i1
                                                                  (memory so little-endian)
        ldr     r1, [r0]
        ldr     r0, =fmt
        bl      printf          @ prints the int in word labeled i1 as hex value (as big-endian)


        ldr     r0, =i2         @ load the 32-bit addr of i2 into r0
        ldr     r1, =c          @ load the 32-bit addr of c into r1


        ldrb    r2, [r1]          _____________________________ Hex value in r2


        strh    r2, [r0]          _____________________________ Hex value in word labeled i2
                                                                  (memory so little-endian)

        strb    r2, [r0, 3]       _____________________________ Hex value in word labeled i2
                                                                  (memory so little-endian)
        ldr     r1, [r0]
        ldr     r0, =fmt
        bl      printf          @ prints the int in word labeled i2 as hex value (as big-endian)


        mov     r0, 0
        sub     sp, fp, 4
        pop     {fp, pc}
```

Page 10 (17 points)

Complete the truth table for the following logic diagram:

A	B	X
0	0	___
0	1	___
1	0	___
1	1	___



A) De Morgan Logic Circuit
B) Combinational Logic Circuit
C) Bi-Modal Logic Circuit
D) Synchronous Sequential Logic Circuit
E) Tri-State Logic Circuit
F) Asynchronous Sequential Logic Circuit

What kind of logic circuit is this? _____ (From table above - answer one of A-F)

Draw the logic circuit for the following boolean expression expressed with C bitwise operators:

```
~(a | b) & (c ^ d)
```

(Use 3 logic gates - Do not use inverters in the logic diagram!)

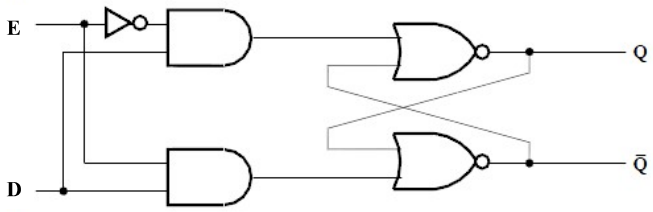
What value must input D and E be in order to set output Q to the value 1 independent of what value Q may have been previously? **Use the numbers in the box below to answer this and the next question.**

D ____

E ____

0) 0
1) 1
2) Either 0 or 1

What value of D will keep the output Q the same independent of what value E is? ____



What kind of logic circuit is this? _____ (From table above - answer one of A-F)

C requires all initialized global and static variables to be initialized with a constant expression known at compile time. But C++ allows global and static variables to be initialized with a variable expression that may only be known at run time (such as the return value of a function call or an expression involving a formal parameter). The C++ compiler will set up such run time dependent initializations so they occur only once - in particular, the first time the function is called for any internal static variables initialized with a variable expression. With this knowledge, what is the output of the following C++ program? [Stack Frames? - You bet!]

```
#include <stdio.h>

void foo( int x ) {
  static int y = x - 2;

  printf( "%d\n", ++y );

  if ( x >= 2 && y <= 5 )
    foo( y++ - 2 );
  else
    printf( "Stop!\n");

  printf( "%d\n", ++y );
}

int main() {

  foo( 5 );

  return 0;
}
```

Put answer here

Page 11 (11 points)

What is the value of each of the following expressions taken sequentially based on changes that may have been made in previous statements?

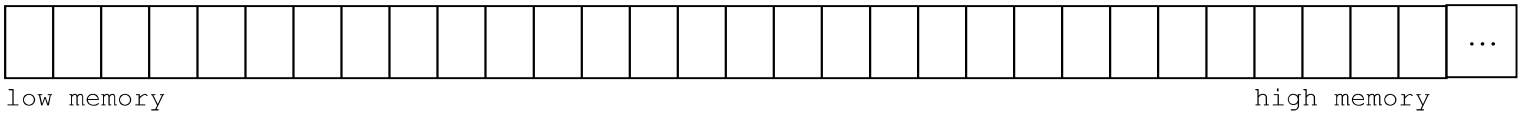
```
#include <stdio.h>

int
main() {
  char a[] = "ARM-RULES!";
  char *ptr = a;

  printf( "%c\n", *ptr++ );       _____

  printf( "%c\n", ++*ptr );       _____

  printf( "%c\n", --*++ptr );     _____

  ptr = ptr + 2;

  printf( "%c\n", (*ptr)++ );     _____

  printf( "%c\n", *++ptr );       _____

  printf( "%c\n", ++*ptr++ );     _____

  printf( "%ld\n", ptr - a );     _____

  printf( "%s\n", a );            _____________________________

  return 0;
}
```

Given the C array declaration

```
      int a[2][4];
```
C

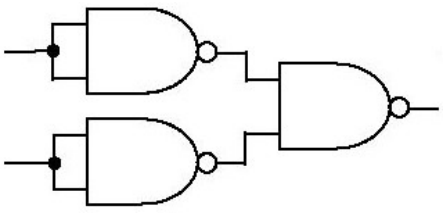
Mark with an **A** the memory location(s) where we would find

```
      a[1][1]
a:
```



Each box represents a byte in memory.

What is Rick's favorite Disney movie? _____



The logic circuit to the left is equivalent to what single logic gate?

_____ gate

Page 12 (32 points)
Give the order of the typical C compilation stages and on to actual execution as discussed in class

0 – prog.exe/a.out (Executable image)
1 – Program Execution
2 – Object file (prog.o)
3 – Loader
4 – as (Assember)
5 – Segmentation Fault (Core Dump) / General Protection Fault
6 – cpp (C preprocessor)
7 – ld (Linkage Editor)
8 – Assembly file (prog.s)
9 – ccomp (C compiler)
10 – Source file (prog.c)

gcc ____ –> ____ –> ____ –> ____ –> ____ –> ____ –> ____ –> ____ –> ____ –> ____ –> ____

Fill in the blanks appropriately. **Do not use . or [] operators. Use only -> and * and sizeof operators.**

```
#include <stdio.h>
#include <stdlib.h>

#define MAX_CHARS 20

struct foo { int *a; char *b; }; // Type definition for struct foo

int main( int argc, char *argv[] ) {

  struct foo *ptr;

  /* Dynamically allocate space automatically initialized to 0 for a struct foo */

  ptr = (struct foo *) _________________ ( 1, ____________________________________ ); // No magic #s

  /* Dynamically allocate uninitialized space for an int and assign it to struct field a */
                                                                                  // No magic #s
  _________________ = (int *) _________________ ( ____________________________________ );

  _________________ = 42; // Set the value struct field a points to in the above allocated space to 42

  printf( "%d\n", _________________ ); // Prints the value 42 via ptr - Do not use 42 for the answer

  /* Dynamically allocate space automatically initialized to 0 for MAX_CHARS chars and assign to
     struct field b */

  _________________ = (char *) _________________ ( MAX_CHARS, ____________________ ); // No magic #s

  /* Dynamically expand the memory for struct foo so there is now memory allocated for two struct
     foo's in contiguous memory pointed to by newPtr. */

  struct foo *newPtr = (struct foo *) _________________ (____________ ,____________________ * 2 );

  /* Check that the above allocation succeeded and if so assign newPtr to ptr, otherwise output
    "ERROR" to standard error. */

  if ( newPtr != _____________ ) {

    _____________ = _____________ ;
  } else {

    _____________ (_____________ , "ERROR\n" );
  }

  free(_________________ );  // Free memory allocated for the int that struct field a points to
                             //   in the first struct foo
  free(_________________ );  // Free memory allocated for the 20 chars that struct field b points to
                             //   in the first struct foo
  free(_________________ );  // Free memory allocated for the two struct foo's that ptr points to

  return 0;
}
```

Hexadecimal - Character

00 NUL	01 SOH	02 STX	03 ETX	04 EOT	05 ENQ	06 ACK	07 BEL
08 BS	09 HT	0A NL	0B VT	0C NP	0D CR	0E SO	0F SI
10 DLE	11 DC1	12 DC2	13 DC3	14 DC4	15 NAK	16 SYN	17 ETB
18 CAN	19 EM	1A SUB	1B ESC	1C FS	1D GS	1E RS	1F US
20 SP	21 !	22 "	23 #	24 $	25 %	26 &	27 ’
28 (	29)	2A *	2B +	2C ,	2D -	2E .	2F /
30 0	31 1	32 2	33 3	34 4	35 5	36 6	37 7
38 8	39 9	3A :	3B ;	3C <	3D =	3E >	3F ?
40 @	41 A	42 B	43 C	44 D	45 E	46 F	47 G
48 H	49 I	4A J	4B K	4C L	4D M	4E N	4F O
50 P	51 Q	52 R	53 S	54 T	55 U	56 V	57 W
58 X	59 Y	5A Z	5B [	5C \	5D]	5E ^	5F _
60 ‘	61 a	62 b	63 c	64 d	65 e	66 f	67 g
68 h	69 i	6A j	6B k	6C l	6D m	6E n	6F o
70 p	71 q	72 r	73 s	74 t	75 u	76 v	77 w
78 x	79 y	7A z	7B {	7C |	7D }	7E ~	7F DEL

A portion of the Operator Precedence Table

Operator	Associativity
++ postfix increment	L to R
-- postfix decrement	
[] array element	
() function call	
-> struct/union pointer	
. struct/union member	
* indirection	R to L
++ prefix increment	
-- prefix decrement	
& address-of	
sizeof size of type/object	
(type) type cast	
* multiplication	L to R
/ division	
% modulus	
+ addition	L to R
- subtraction	
.	
.	
.	
= assignment	R to L