1. Convert the following binary numbers into hexadecimal and octal

a. 1111→ 0x__
b. 1100→ 0x__
c. 1010→ 0x__
d. 1110→ 0x__
e. 1101→ 0x__
f. 1011→ 0x__

2. Convert the following base-10 decimal numbers to unsigned binary numbers

a. 42

b. 63

c. 229

d. 845

3. Express -45 base-10 in binary (2's complement, 8-bits)

4. What gets printed by each line

```
int
main()
{
  char a[] = "ARMV7";
  char *p = a;
  printf( "%c", p[3] + 1 );                    _____

  printf( "%c", *(a+4) = *p );                 _____

  printf( "%c", *p++ );                        _____

  printf( "%c", *p-- );                        _____

  printf( "%c", *++p = *(a+2) + 3 );           _____

  printf( "%c", *(p+2) );                      _____

  printf( "%d", ++p - a );                     _____

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

  return 0;
}
```

If we have a stack pointer that starts at the address 0x30000200 and has two words pushed on to it with the following code:

```
ldr r3 = 0xBABEFACE
ldr r4 = 0xBEEFBEEF
push {r3, r4}
```

What is the address contained in the stack pointer?

Assume we have a char array that stores 10 elements. Register r0 holds the array base address, and the register r1 plays the role of index i. How do we get the i-th element in the array into r2?

Assume we have an int array that stores 50 elements. The register r0 holds the array base address, and the register r1 holds address of the last element in the array. How should we increment every integer of the array by 5?

```
// r0 =array base address
// initialization code
  add r1, r0, 200
  b .Ltest
.Lloop:
  __________
  __________
  __________
  __________
.Ltest:
  cmp r0, r1
  bl .Lloop
```

Q3

What is the problem, if any, with the following function?

```
void fn (int sz){
  double *dblBuf = malloc(sizeof(double) * sz);
}
```

- There is no problem with this code
- The code allocates a potentially large buffer in stack memory
- The code creates a memory leak
- None of the other choices is correct.
- The code creates a dangling pointer

Q8

We have a floating-point number format with the following field widths:

Sign	Exponent	Mantissa
1	6	7

The exponent bias is calculated as

$2^{k-1} - 1$

where k = the number of bits for the exponent.

Translate this 14-bit number, which is in the above format, to decimal. (The number has been left-padded with 2 `0`s for the shown hex.)

```
0x31AF
```

- -43.75
- 5.875
- 0.078125
- -21.875

For each of the three C statements that you need to match in this question, assume that:

- r0 contains the address of x
- r1 contains the address of y
- the statements are executed in isolation (i.e. the first does not affect the second, and vice versa)

What is the equivalent in ARM of each of the following C statements? Match each statement with the letters corresponding to the ARM instruction(s) above.

Here are the available options and their respective letters:

A.
`ldr r1, [r0]`
	B.
	`ldr r2, [r0]`
	`str r2, [r1]`
		C.
		`ldr r2, [r0]`
		`ldr r1, [r2]`

D.
`ldr r2, [r0]`
`ldr r2, [r2]`
`str r2, [r1]`
	E.
	`ldr r2, [r0]`
	`ldr r3, [r1]`
	`str r3, [r1]`
		F.
		`str r1, [r0]`

G. `str r0, [r1]`	**H.** `mov r1, r0`	**I.** `mov r0, r1`
J. None of the other choices is correct.		

y = x;	[Choose]
y = &x;	[Choose]
y = *x;	[Choose]

Create code that prints the names in a singly-linked list in reverse order using recursion. The function `printRev` is initially passed a pointer (possibly NULL) to the head of the linked list; if the list is empty, then the function should print nothing.

Fill in the boxes for the 5 lines using any of the 10 code snippets listed below. These code snippets may be reused. **You must write the code to adhere to the available brackets provided below (i.e. you cannot add brackets that are not part of a given snippet, or remove any given brackets).**

If there are any blank lines, fill in the corresponding boxes with the letter for the blank line. **All blank lines, if any, should be listed last.**

(Hint: it might help to write the code first and then select the proper answers.)

```
struct rec {
    char *name;
    char *homeAddress;
    int  studentId;
    struct rec *nxt;   // next node in the linked list
};

// print the linked list of names in reverse order using re
cursion.
// The last record will have its nxt pointer equal NULL.

void printRev(struct * rec ptr){
   <A>
   <B>
   <C>
   <D>
   <E>
   } //Take note of this lone closing bracket
}
```

The available code snippets are:

```
A. printf("%s\n", *ptr);
B. ptr = ptr->homeAddress;
C. printf("%s\n", *ptr->name);
D. printf("%s\n", ptr->name);
E. printRev(ptr);
F. printRev(ptr->nxt);
G. if (ptr->nxt != NULL) {
H. if (*ptr != NULL) {
I. if (ptr != NULL) {
J. // blank line
```

1. What are the hex values of register r0 after executing each line? (each line depends on the line before it)
Assume that `r1=0x00001234, r2=0x0000FACE.`

```
and r0, r1, r2    r0=_____
lsl r0, r0, #6    r0=_____
asr r0, r0, #3    r0=_____
lsr r0, r0, #5    r0=_____
```

Suppose you are working with 4 bit 2's complement signed numbers. Which pairs of numbers, added, will produce the wrong result (overflow). Check ALL that apply.

- [ ] -5+-5
- [ ] -2+6
- [ ] -1 + 1
- [ ] 4+4
- [ ] -4 + -4

Consider the following C code :

```
int* f() {
    int* p = malloc(sizeof(int) * 5);
    return p;
}
```

Which of the following is true? **Select one option.**

- ◯ `p` is stored on the stack, `*p` is stored in the heap
- ◯ both `p` and `*p` are stored in the stack
- ◯ both `p` and `*p` are stored in the heap
- ◯ `p` is stored in the heap, `*p` is stored on the stack

```c
#include <stdio.h>
#include <stdlib.h>
void mystrcpy(char *dest, char *src) {
    if ((src == NULL) || (dest == NULL)) return;
    while (*src != '\0') {
        *dest++ = *src++;
    }
}
int main(void) {
    char str[10] = "spam";
    mystrcpy(str + 5, str);
    printf("%s", str);
    return EXIT_SUCCESS;
}
```

When you run the program, select the closest description of what happens. **Select one option.**

- ◯ It prints: `spam`
- ◯ Infinite loop. It never completes execution
- ◯ It prints: `spamspam`
- ◯ Segmentation fault
- ◯ None of the above

What is the C equivalent of the following ARM code? Assume variable `x` is stored in `r2`, and `A` is an array of 4-byte `int`s, with `A`'s address stored in `r3`. **Select one option.**

```
ldr r0, [r3, 32]
add r0, r0, r2
str r0, [r3, 48]
```

- ◯ `A[48] = A[32] + x;`
- ◯ `A[12] = A[8] + x;`
- ◯ `A[6] = A[4] + x;`
- ◯ None of the above

Given the following C loop:

```
while ((r1 <= 10) && (r2 != 0)) {
     r1++;
     r2 = r2 + r3;
}
```

At each location where there is a blank, fill in the ARM 32-bit branch instruction necessary to achieve **exactly** the same operation as implemented by the C language loop above

```
    cmp         r1, 10

    _________   .L2          //
    cmp         r2, 0

    _________   .L2          /
.L1:
    add         r1, r1, 1
    add         r2, r2, r3
    cmp         r1, 10

    _________   .L2          // 20c)
    cmp         r2, 0

    _________   .L1          // 20d)
.L2:
```

What does the following code do to the value in `r0`?

```
mov r1, #32
eor r1, r1, #0x30
orr r0, r0, r1
```

- ◯ Flips bit 4 of `r0`
- ◯ Flips bit 30 of `r0`
- ◯ Sets bit 30 of `r0` to 0
- ◯ Sets bit 4 of `r0` to 1
- ◯ Sets bit 4 of `r0` to 0