Q1. **(05 points)** Circle the correct answer

1. How many bits does a ***WORD*** have? 4B→32b
   A. 4 B. 8 C. 16 D. 32 E. 64
2. How many bits does a ***register*** have in the ARM processor?
   A. 4 B. 8 C. 16 D. 32 E. 64 8 4B→32b
3. Memory address is always in terms of ***bytes***.
   A. TRUE B. FALSE
4. What is the result of **y** in binary? **y = 0x11<<2** 0001 0001
   A. 0b1100 B. 0b1000100 C. 0b0011 D. 0b0000 E. 0b0101
   0b0001 0001 LSL 2 0100 0100
5. If the physical memory address has 32 bits, the maximum amount of memory it can access is 2^{32}
   A. 4GB B. 4MB C. 1GB D. 2GB E. 2MB
   2^{22} 1 0 0 0 0 0 0 0 0 1 GB = 1000 MB $= 2^{10}$
6. In a C program, what does the following code do? **x &= ~(1<<k);**
   A. Clear a bit B. Set a bit C. Toggle a bit D. Check a bit
7. In a C program, what does the following code do? **x |= 1<<k;**
   A. Clear a bit B. Set a bit C. Toggle a bit D. Check a bit
8. In a C program, what does the following code do? **x ^= 1<<k;**
   A. Clear a bit B. Set a bit C. Toggle a bit D. Check a bit
9. In a C program, a ***pointer*** is actually a memory address.
   A. TRUE B. FALSE
10. If you do not follow the protocol, your program might still run correctly. But your assembly program cannot be called by a standard C procedure or by another assembly program written by a different programmer.
    a. TRUE b. FALSE

Q2. **(03 points)** For the 4-bit binary representations in the table below, show the equivalent decimal values when the data is interpreted as unsigned binary or signed binary.

Binary representation	Signed Decimal Value	Unsigned Decimal Value
0000	0	0
0111	7	7
1111	-1	15

0001

Q3. **(06 points)** Assume an array of 30 integers (each integer is a word). A compiler associates variables x and y with registers r0 and r1, respectively. Assume that the base address for the array is located in register r2. Translate this C statement into ARM assembly language.

4 Bytes

```
x = array[7] + y;
```

LDR r0, [r2, #28] ; x = array[7] → each int = 4B ∴ offset = 7×4 = 28
ADD r0, r0, r1 ; x = x+y
r1 = y

MOV r0, x ; r0 = x
MOV r1, y ; r1 = y

assume this is implied since only given 2 lines

array arr[30];

LDR → load reg to word

6

Q4. **(08 points)** Suppose we have a hypothetical processor, of which each register has only five bits. The contents of registers r0 and r1 are initialized as follows

- r0 = **0b11101**
- r1 = **0b10110.**

What are the N, Z, C, and V flags of the following instructions? Assume initially N = 0, Z = 0, C = 1, V = 0, and these instructions are executed independently and separately after the above initializations (*i.e.*, they are NOT part of a program).

(1) ADDS r3, r0, r1

```
  11101
+ 10110
-------
  10011
```

N	Z	C	V
1	0	1	0

neg+neg = neg

(2) SUBS r3, r0, r1

```
  11101  (-3)
- 10110  (-10)
-------
  00111
```

N	Z	C	V
1	0	1	0

borrow

(3) EOR r3, r0, r1

```
  11101
⊕ 10110
-------
  01011
```

N	Z	C	V
0	0	1	1

(4) ANDS r3, r1, r1, LSL #3

```
  10110
& 10000
-------
  10000
```

N	Z	C	V
1	0	1	0

7

Q5. **(10 points)** ARM Data Addressing. Suppose r0 = 0x00008000, and the memory layout is presented in this table.

Address	Data	
0x00008000	0x1A	LE
0x00008001	0x2C	
0x00008002	0xEB	
0x00008003	0x0D	BE
0x00008004	0xFD	
0x00008005	0xA3	
0x00008006	0xCD	
0x00008007	0x79	

i. (04 points) ARM processors can be configured as big-endianness or little-endianness. What is the value of r1 after running

LDR r1, [r0]?

a. If little-endianness is used:

r1 = 0x0DEB2C1A

b. If big-endianness is used:

r1 = 0x1A2CEB0D

ii. (06 points) Suppose the system is based on little-endianness. What are the values of r1 and r0 if the following instructions are executed separately and independently after initialization of r0 = 0x00008000?

LDR r1,[r0,#4]

r0 = 0x00008000

r1 = 0x79CDA3FD

LDR r1,[r0,#4]!

r0 = 0x00008004

r1 = 0x79CDA3FD

Q6. **(06 points)** Suppose r0 = 0x2000,0000 and r1 = 0x1234,5678. All bytes in memory are initialized to 0x00.

```
1. STR r1, [r0], #4    <- post-index ya dummy!
2. STR r1, [r0, #8]!
3. STR r1, [r0]
```

Assume the computer uses Little Endian to store data. Show the memory content when the above program completes successfully.

STR r1, [r0], #4 ; [r0] = r1+4
0x1234567C

STR r1, [r0, #8]!
↳ store original r1 starting @ 0x2000,0008, save this location as r0

STR r1, [r0]
↳ store r1 starting at 0x2000,0008 (already stored here – will overwrite to same values)

Little Endian!!

Memory Address	Memory Content
0x2000,0013	
0x2000,0012	
0x2000,0011	
0x2000,0010	
0x2000,000F	~~78~~ 12
0x2000,000E	~~56~~ 34
0x2000,000D	~~34~~ 56
0x2000,000C	~~12~~ 78
0x2000,000B	~~78~~
0x2000,000A	~~56~~
0x2000,0009	~~34~~
0x2000,0008	~~12~~
0x2000,0007	
0x2000,0006	
0x2000,0005	
0x2000,0004	
0x2000,0003	~~7C~~ 12
0x2000,0002	~~56~~ 34
0x2000,0001	~~34~~ 56
0x2000,0000	~~12~~ 78

1. r0 = 0x2000,0004

2. r0 = 0x2000,000C

Q7. **(12 points)** What do the following assembly programs calculate?

Program 1:

```
f       MOV r2, #1      //1
        MOV r1, #1      //2
loop    CMP r1, r0      //3
        BGT done        //3
        MUL r2, r1, r2  //4
        ADD r1, r1, #1
        B   loop
done    MOV r0, r2      //3.1
```

1: Store value 1 in r2
2: store value 1 in r1
3: if r0 > 1, branch to "done"
   3.1: set r0 equal to 1
4: if r0 ≤ 1,
   set r2 = r2 x r1 (r2 x r1)
   increment r1 by 1
   return to beginning of "loop"

```
function (c) {
    int a, b = 1;
    while (a <= c) do {
        b = a * b;
        a++;
    }
    c = b;
}
```

We have a function that takes in some number (c). Within our function we have two vars (a & b), both set to 1. While the number we passed in, c, is greater than or equal to a, we will set b = a x b and increment a. As soon as a is greater than c, we set the value of c equal to b.

Program 2:

```
        AREA my code, CODE
        EXPORT __main
        ALIGN
        ENTRY
__main PROC

        MOVS   r0,#0 ; r0=0
        MOVS   r1,#15 ; r1=15
        MOVS   r2,#0 ; r2=0

loop
        CMP    r2,r1 ; r2-r1=-15
        BGT    stop
        MLA    r0,r2,r2,r0 ; r0=r2xr2+r0 = 0+0=0
        ADDS   r2,r2,#1 ; r2=r2+1 = 0+1 =1
        B      loop

        % The final result is saved in register r0

stop
        B      stop
        ENDP
        END
```

36
55

r0 = ~~0~~ + ~~5~~ ~~14~~ ~~30~~ ~~55~~ ~~91~~ ~~140~~ ~~204~~ ~~285~~ ~~385~~ ~~506~~ ~~650~~ ~~819~~ 1015 1271

r2 = ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ ~~7~~ ~~8~~ ~~9~~ ~~10~~ ~~11~~ ~~12~~ ~~13~~ ~~14~~ 15 16

```
int a=0, b=15, c=0;
while(c≤b){
    a= b*b+a;
    b++;
}
return a;
```

We have a function with 3 variables: a=0, b=15, c=0. While c≤b, or c≤15, we will change the value of a to b^2+a and increment the value of b. When c is finally greater than b, or 15, we will save the final value (1271) to register r0, and the program ends.

ok

385
121
506
144
1650
169
1819
196
1015
256
1271

14
14
16
40
40
100
196

16
16
36
60
60
400
256