Digital Computer Design

1. Consider the following C code snippet.

```
// C code
void setArray(int num) {
     int i=5;
     compare(4, i);
}

int compare(int a, int b) {
     if (sub(a, b) >= 0)
          return 1;
     else
     return 0;
}

int sub(int a, int b) {
     return a - b;
}
```

Implement the C code snippet in ARM assembly language. Use R4 to hold the variable i.

2. Convert the following ARM assembly code into machine language. Write the instructions in hexadecimal.

```
MOV R10, #63488
LSL R9, R6, #7
STR R4, [R11, R8]
ASR R6, R7, R3
```

3. Convert the following ARM assembly code into machine language. Write the instructions in hexadecimal.

```
ADD R8, R0, R1
LDR R11, [R3, #4]
SUB R5, R7, #0x58
LSL R3, R2, #14
```

4. Convert the following program from machine language into ARM assembly language. The numbers on the left are the instruction addresses in memory, and the numbers on the right give the instruction at that address. Then reverse engineer a high-level program that would compile into this assembly language routine and write it. Explain in words what the program does. R0 and R1 are the input, and they initially contain positive numbers, a and b. At the end of the program, R0 is the output.

```
0x00008008 0xE3A02000
0x0000800C 0xE1A03001
0x00008010 0xE1510000
```

```
0x00008014 0x8A000002
0x00008018 0xE2822001
0x0000801C 0xE0811003
0x00008020 0xEAFFFFFA
0x00008024 0xE1A00002
```

5. Convert the following program from machine language into ARM assembly language. The numbers on the left are the instruction addresses in memory, and the numbers on the right give the instruction at that address. Then reverse engineer a high-level program that would compile into this assembly language routine and write it. Explain in words what the program does. R0 and R1 are the inputs. R0 contains a 32-bit number and R1 is the address of a 32-element array of characters (char).

```
0x00008104 0xE3A0201F
0x00008108 0xE1A03230
0x0000810C 0xE2033001
0x00008110 0xE4C13001
0x00008114 0xE2522001
0x00008118 0x5AFFFFFA
0x0000811C 0xE1A0F00E
```

6. Convert the following branch instructions into machine code. Instruction addresses are given to the left of each instruction.

(a)
```
0x0000A000          BEQ LOOP
0x0000A004          ...
0x0000A008          ...
0x0000A00C LOOP     ...
```

(b)
```
0x00801000          BGE DONE
                    ...
0x00802040 DONE     ...
```

(c)
```
0x0000B10C BACK     ...
...                 ...
0x0000D000          BHI BACK
```

(d)
```
0x00103000          BL FUNC
...                 ...
0x0011147C FUNC     ...
```

(e)
```
0x00008004 L1       ...
...                 ...
0x0000F00C          B L1
```

7. Consider the following ARM assembly language snippet. The numbers to the left of each instruction indicate the instruction address.

```
0x000A0028 FUNC1          MOV R4, R1
0x000A002C                ADD R5, R3, R5, LSR #2
0x000A0030                SUB R4, R0, R3, ROR R4
0x000A0034                BL FUNC2
...                       ...
0x000A0038 FUNC2          LDR R2, [R0, #4]
0x000A003C                STR R2, [R1, -R2]
0x000A0040                CMP R3, #0
0x000A0044                BNE ELSE
0x000A0048                MOV PC, LR
0x000A004C ELSE           SUB R3, R3, #1
0x000A0050                B FUNC2
```

(a) Translate the instruction sequence into machine code. Write the machine code instructions in hexadecimal.
(b) List the addressing mode used at each line of code.