1. Recall the format of ARM data processing instructions (full details provided in original document).

a) What is the purpose of the I bit?

When the I bit is 0, the second operand is the contents of a register.
When the I bit is 1, the second operand is an immediate value.

b) What is the purpose of the S bit?

When the S bit is 0, the NZCV bits are not updated.
When the S bit is 1, the NZCV bits are updated.

c) What is the smallest immediate value which can be used? The largest?

The 8-bit field contains an unsigned integer value. Thus, the smallest value is 0x00 and the largest value is 0xff.

d) Convert each of the ARM machine language instructions shown below (in hexadecimal) into ARM assembly language instructions.

```
e086a007    add  r10, r6, r7
e096a007    adds r10, r6, r7
e286a007    add r10, r6, #7
e1520003    cmp r2, r3
ela01003    mov r1, r3
```

e) Convert each of the ARM assembly language instructions shown below into ARM machine language instructions (in hexadecimal).

```
subs r5, r4, r3    e0545003
cmn r6, #12        e376000c
orr r8, r7, #0xff   e38780ff
mov r5, r4          ela05004
eor r11, r10, r9    e02ab009
```

f) Recall that an assembler is the program which converts assembly language instructions into machine language instructions. Assume that an ARM assembler is to be extended to accept the following assembly language instruction:

```
inc r5
```

The purpose of that instruction is to increment the contents of R5 by 1. Give the equivalent ARM machine language instruction.

Equivalent assembly language instruction: add r5, r5, #1

Equivalent machine language instruction: e2855001
2. Consider the following ARM assembly language statements. For each statement which contains a comment, give the requested information after the instruction is executed. Give the value of the register in hexadecimal and the value of the condition code bits (if requested) in binary.

```
ldr r0, =0xb60000e5
ldr r1, =0x7c00008d
eor r4, r0, r1      @  r4:  ____ca000068____________
adds r5, r0, r1      @  r5:  ____32000172____________
          @ NZCV: ____0010____
omr r6, r0, r1      @  r6:  ____fe0000ed____________
subs r7, r0, r1      @  r7:  ____3a000058____________
          @ NZCV: ____0011____
asr r8, r0, #8      @  r8:  ____ff600000____________
lsl r9, r0, #12     @  r9:  ____000e5000____________
          @ NZCV: ____0011____
asm r10, r1, #16    @  r10: ____00007c00____________
lsl r11, r1, #20    @  r11: ____08d00000____________
```

3. Give the ARM assembly language code segment which is equivalent to the following C statements. Assume that all variables are integers, the value of "AAA" is in register r4, and the value of "BBB" is in register r5.

```
if (AAA >= 200 && BBB < 150)                 cmp r4, #200
{                                            blt else
       AAA = AAA - 15;                            cmp r5, #150
}                                            bge else
else                                         sub r4, r4, #15
{                                            b      endif
    BBB = BBB + 25;                    else:    add r5, r5, #25
}                                            add r4, r4, #15
```

4. Give the ARM assembly language code segment which is equivalent to the following C statements. Assume that all variables are integers, the value of "AAA" is in register r4, and the value of "BBB" is in register r5.

```
AAA = 1;
while (AAA < 10)
{                                            mov r4, #1
    BBB = BBB + AAA;
    AAA++;
}                                            cmp r4, #10
loop:                                        bge endloop
```

```
endloop:
add r5, r5, r4
add r4, r4, #1
b      loop
```
5. Consider the C source code statements shown below.

```c
int test( int*, int );
int X, Y, Z;
int main()
{
    Z = test( &X, 8*Y );
}
```

Complete the ARM assembly language function definition below so that it is equivalent to the C definition of function "main" (shown above).

```assembly
.data
.balign 4
X: .skip 4
Y: .skip 4
Z: .skip 4

.global main
.text
.balign 4
main:
    push  {lr}
    ldr  r0, =X
    ldr  r1, =Y
    lsl  r1, [r1]
    bl   test
    ldr  r1, =Z
    str  r0, [r1]
    pop  {lr}
    bx   lr
```
6. Consider the ARM assembly language code segment shown below.

```assembly
.data
.balign 4
LocA:
.byte   0x55, 0x66, 0x77, 0x88, 0x99, 0xAA
.byte   0xBB, 0xCC, 0xDD, 0xEE, 0xFF, 0x00
.byte   0x11, 0x22, 0x33, 0x44, 0x55, 0x66
.byte   0x77, 0x88
LocB:
.word   0xffffffff, 0xeceeeedde
.word   0xddddddd, 0xcccccccc
.word   0xbbbbbbb
.text
.balign 4
ldr     r0, =0x3456789A
ldr     r1, =LocA
ldr     r2, =LocB
ldr     r4, [r1, #0]  @ r4: ___88766555_______
ldrsb   r5, [r1, #2]  @ r5: ___00000777_______
ldrb    r6, [r1, #4]  @ r6: ___00000999_______
ldrsh   r7, [r1, #6]  @ r7: ___FFFFCBBB_______
ldrh    r8, [r1, #8]  @ r8: ___0000EEDD_______
strb    r0, [r2, #2]
strh    r0, [r2, #6]
str     r0, [r2, #12]
```

a) The contents of the 20 bytes of memory starting at "LocA" are shown below (in hexadecimal).

In the spaces provided next to the assembly language instructions, give the four-byte hexadecimal value in each of the indicated registers after the instructions execute.

LocA: 55 66 77 88 99 aa bb cc dd ee ff 00 11 22 33 44 55 66 77 88

b) The contents of the 20 bytes of memory starting at "LocB" are shown below (in hexadecimal).

In the memory dump below, give the hexadecimal value in each of the indicated memory locations after the instructions execute. You only need to give the value for memory locations which are changed by the instructions.

LocB: ff ff 9a ff ee ee 9a 78 dd dd dd dd 9a 78 56 34 bb bb bb bb
7. Consider the C source code statements shown below.

```c
struct player
{
    char name[30];
    unsigned int goals;
    unsigned int assists;
    unsigned int points;
};

// Function total: calculate points for exactly one player in array
void total( struct player roster[], unsigned int n )
{
    roster[n].points = roster[n].goals + roster[n].assists;
}
```

Complete the ARM assembly language function definition below so that it is equivalent to the C definition of function "total" (shown above).

```assembly
.global total
.text
.balign 4
total:
    push    {lr}
    mov     r2, #44
    mul     r1, r1, r2
    add     r0, r0, r1
    ldr     r2, [r0, #32]
    ldr     r3, [r0, #36]
    add     r2, r2, r3
    str     r2, [r0, #40]
    pop     {lr}
    bx      lr
```