4.1 Conditional Constructs

(a) Give the equivalent C language code segment that describes the ARM assembly code segment given in Fig. 4.1a. Assume the address labels of memory variables **R1**, **R2** and **R3** represent corresponding names of integer variables in your C equivalent code.

```
            MOV R3,#0
            CMP R1,R2
            BEQ Label_1
            B   Label_2
Label_1     MOV R3,#1
Label_2     MOV R1,#0
```

Fig. 4.1a – A segment of ARM assembly code

(b) Optimize the ARM assembly code given in Fig 4.1a. The new code must produce identical results as the original code under all conditions. (Hint: Use conditional instructions)

(c) Give the ARM assembly code that implements the C language code segment given in Fig 4.1b. Assume X,Y, and Z are mapped to R0, R1, and R2, respectively.

```
if ((X > Y) && (X <= Z))
{
    Y = -Y;
}
else
{
    Z = -Z;
}
```

Fig. 4.1b – A segment of C language code

4.2 Loop Constructs

(a) Is the loop construct given in Fig. 4.2 a pre-test or post-test loop?

11 04

```
while (X <= Y)
{
  X = X + 2;
  Y = Y - 1;
}
```

Fig. 4.2 – A segment of C language code

(b) Give the ARM assembly code that implements the C code shown in Fig 4.2. Assume X and Y are mapped to R0 and R1, respectively.

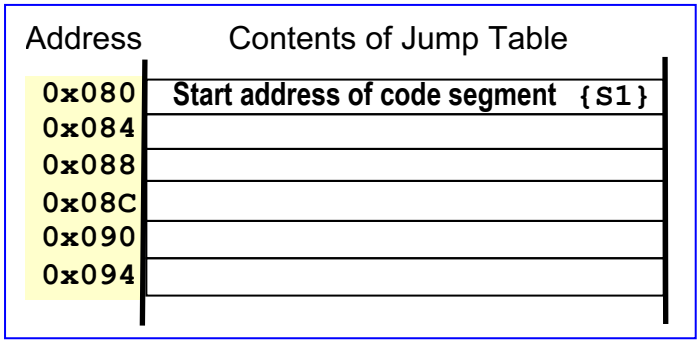
(c) Optimize your solution in question 4.2(b) such that your code segment would run faster in situations where the loop segment is expected to execute many times.

4.3 The Switch Construct

Jump Tables can be used to efficiently implement Switch constructs like the one in Fig 4.3(a).

```
switch( X )
{
   case -1:
     {S1};
     break;
   case 1:
     {S2};
     break;
   case 2:
     {S3};
     break;
   default :
     {S4}
     break;
}
```

(a)



(b)

Fig. 4.3 – (a) A segment of C pseudo-code implementing a Switch construct. (b) The partially completed Jump Table used to implement the Switch construct in (a).

(a) Assuming you have created a Jump Table starting at address 0x080, describe how you would fill the entry of this Jump Table in order to handle the case values shown in Fig. 4.3(a). You do not need to know the exact start addresses of the case code segments {S1} to {S4}, only state which start address resides in which entry of the Jump Table, as shown in Fig. 4.3(b).

(b) Based on the Jump Table you have setup in part (a), write the segment of ARM assembly code to implement the Switch construct shown in Fig. 4.3(a). Assume the C integer variable **x** is a memory variable (stored in address 0x200) and has the numeric address of **x** in memory.