4.1 Conditional Constructs

(a) Give the equivalent C language code segment that describes the VIP assembly code segment given in Fig. 4.1a. Assume the address labels of memory variables **`Var1`**, **`Var2`** and **`Var3`** represent corresponding names of integer variables in your C equivalent code.

```
            MOV [Var3],#0
            CMP [Var1],[Var2]
            JEQ Label_1
            JMP Label_2
Label_1     MOV [Var3],#1
Label_2     MOV [Var1],#0
```

Fig. 4.1a – A segment of VIP assembly code

(b) Optimize the VIP assembly code given in Fig 4.1a. The new code must produce identical results as the original code under all conditions. (Hint: avoid using the unconditional jump)

(c) Give the VIP assembly code that implements the C language code segment given in Fig 4.1b. Use the names of the C integer variables **X**, **Y** and **Z** to represent the numeric value of the addresses where each of the respective variables is stored in memory as a word-sized value.

```
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?

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

Fig. 4.2 – A segment of C language code

(b) Give the VIP assembly code that implements the C code shown in Fig 4.2. Use the names of the C integer variables **X**, **Y** and **Z** to represent the numeric value of the addresses where each of the respective variables is stored in memory as a word-sized value.

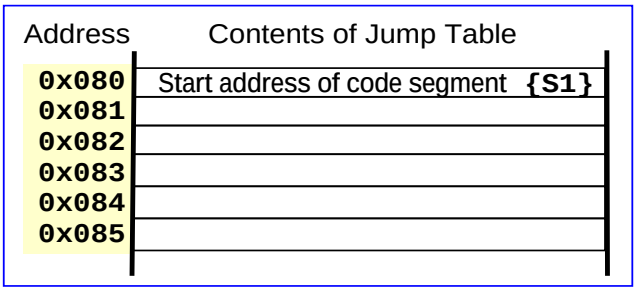
(c) Optimize your solution in question 4.2(b) such that your code segment would run faster in situations where the loop segment is expected to execute many times.

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 VIP assembly code to implement the Switch construct shown in Fig. 4.3(a). Assume the C integer variable **X** is a memory variable and has the numeric address of **X** in memory.