4. Translate each of the following C statements into a corresponding sequence of assembly language instructions, where *ch* is the label on an 8-bit memory location whose content is an ASCII character, and *x*, *y*, and *z* are labels on 32-bit variables of type int32_t:

(a) `if (x < y && y < z) z = 6; else z = x;`

```
        LDR     R0,x        // x < y ?
        LDR     R1,y
        CMP     R0,R1
        BGE     Else
        LDR     R0,z        // y < z ?
        CMP     R1,R0
        BGE     Else
Then:   LDR     R0,=6       // z = 6 ;
        STR     R0,z
        B       EndIf
Else:   LDR     R0,x        // z = x ;
        STR     R0,z
EndIf:
```

```
        if (x < y && y < z) goto Then ;
        goto Else ;
Then:   z = 6 ;
        goto EndIf ;
Else:   z = x ;
EndIf:

        if (!(x < y && y < z)) goto Else ;
Then:   z = 6 ;
        goto EndIf ;
Else:   z = x ;
EndIf:

        if (x >= y) goto Else ;
        if (y >= z) goto Else ;
Then:   z = 6 ;
        goto EndIf ;
Else:   z = x ;
EndIf:
```

(b) `if (-10 < x && x < +10) goto L1 ;`

```
        LDR     R0,x            // x > -10 ?
        CMP     R0,-10
        BLE     EndIf
        CMP     R0,10           // x < +10 ?
        BGE     EndIf
        B       L1              // goto L1 ;
EndIf:
```

(c) `if (x < 10 || x > 20) y = 0 ; else y = 1 ;`

```
Then:   LDR     R0,x            // x < 10 ?
        CMP     R0,10
        BLT     Then
        CMP     R0,20           // x > 20 ?
        BLE     Else
Then:   LDR     R0,=0           // y = 0 ;
        STR     R0,y
        B       EndIf
Else:   LDR     R0,=1           // y = 1 ;
        STR     R0,y
```

```
        if (x < 10 || x > 20) goto Then ;
        goto Else ;
Then:   y = 0 ;
        goto EndIf ;
Else:   y = 1 ;
EndIf:

        if (x < 10) goto Then ;
        if (x <= 20) goto Else ;
Then:   y = 0 ;
        goto EndIf ;
Else:   y = 1 ;
EndIf:
      EndIf:
```

(d) `if ('a' <= ch & & ch <= 'z')`
`ch = ch - 'a' + 'A' ;`

```
        LDRB    R0,ch       // ch >= 'a'
        CMP     R0,'a'
        BLT     EndIf
        CMP     R0,'z'      // ch <= 'z' ?
        BGT     EndIf
Then:   SUB     R0,R0,'a'   // ch = ch-'a'+'A'
        ADD     R0,R0,'A'
        STRB    R0,ch
EndIf:
```

```
    if (!(ch >= 'a' && ch <= 'z') ) goto EndIf ;
    ch = ch – 'a'   'A' ;
                ⇩
EndIf:

    if (ch < 'a' || ch > 'z' ) goto EndIf ;
    ch = ch – 'a'   'A' ;
                ⇩
EndIf:

    if (ch < 'a' ) goto EndIf ;
    if (ch > 'z' ) goto EndIf ;
    ch = ch – 'a' + 'A' ;
EndIf:
```

(e) `x = y / 5 ;`

```
        LDR     R0,y
        LDR     R1,=5
        SDIV    R0,R0,R1
        STR     R0,x
```

(f)
```
uint32_t u32 ;
int32_t s32 ;

if (u32 > 10) s32 = s32  - 1 ;
else s32 = s32 + 1 ;
```

```
        LDR     R0,u32      // u32 > 10 ?
        CMP     R0,10
        BLS     Else
Then:   LDR     R0,s32      // s32 = s32 – 1 ;
        SUB     R0,R0,1
        STR     R0,s32
        B       EndIf
Else:   LDR     R0,s32      // s32 = s32 + 1 ;
        ADD     R0,R0,1
        STR     R0,s32
EndIf:
```

```
LDR     R0,s32
LDR     R1,u32
CMP     R1,10
ITE     HI
SUBHI   R0,R0,1
ADDLS   R0,R0,1
STR     R0,s32
```

```
(g) int32_t s32 ;

    if (-10 < s32 && s32 < +10) s32 = 0 ;

                LDR     R0,s32          // s32 > -10
                CMP     R0,-10
                BLE     EndIf

                CMP     R0,10           // s32 < +10 ?
                BGE     EndIf

        Then:   LDR     R0,=0           // s32 = 0 ;
                STR     R0,s32

        EndIf:
```

```
        if (s32 > -10 && s32 < +10) goto Then;
        goto EndIf
Then:   s32 = 0 ;
EndIf:

        if (s32 <= -10 || s32 >= +10) goto EndIf;
Then:   s32 = 0 ;
EndIf:

        if (s32 <= -10)  goto EndIf;
        if (s32 >= +10) goto EndIf;
Then:   s32 = 0 ;
EndIf:
```

```
(h) uint32_t u32, min, max ;

    if (u32 < min || u32 > max) u32 = 0 ;

                LDR     R0,u32          // u32 < min ?
                LDR     R1,min
                CMP     R0,R1
                BLO     Then

                LDR     R1,max          // u32 > max ?
                CMP     R0,R1
                BLS     EndIf

        Then:   LDR     R0,=0           // u32 = 0 ;
                STR     R0,u32

        EndIf:
```

```
        if (u32 < min || u32 > max) goto Then ;
        goto EndIf ;
Then:   u32 = 0 ;
EndIf:

        if (u32 < min) goto Then ;
        if (u32 <= max) goto EndIf ;
Then:   u32 = 0 ;
EndIf:
```

5. Write a function in assembly language to find and return the minimum value in an array. The function prototype is:

```
int32_t Minimum(int32_t data[], int32_t count) ;
```

```
int32_t Minimum(int32_t data[],int32_t count)
        {
        int32_t min, index, temp ;

        min = data[0] ;
        for (index = 1; index < count; index++)
               {
               temp = data[index] ;
               if (temp < min) min = temp ;
               }
        return min ;
        }
```

```
Minimum: // R0  = &data[0]
         // R1  = count
         // R2  = min
         // R3  = index
         // R12 = temp
```

There are certainly better solutions that use fewer instructions, but I thought this one was the closest to the C version of the function, and thus the easiest to understand.

```
         LDR    R2,[R0]               // R2 (min) ← data[0]
         LDR    R3,=1                 // R3 (index) ← 1
L1:      CMP    R3,R1                 // is R3 (index) < R1 (count) ?
         BGE    L2                    // if not, all done
         LDR    R12,[R0,R3,LSL 2]     //    R12 (temp) ← data[index]
         CMP    R12,R2                //    is R12 (temp) < R2 (min) ?
         IT     LT                    //    if yes, then update min
         MOVLT  R2,R12                //       R2 (min) ← R12 (temp)
         ADD    R3,R3,1               //    R3 (index) ++
         B      L1                    // repeat the loop
L2:      MOV    R0,R2                 // return value in R2 (min)
         BX     LR
```