Understanding the code generated by the GNU C compiler

In this file, you can compare the C code for the function "countWhites" of Pract4a to the corresponding assembly code that has been generated after the compilation and linking steps.

The assembly code has been taken directly from the disassembly window of the Eclipse environment after launching the program with the "Pract4a" Debug Configuration. However, the code generated by the compiler does not include any comments (of course, they are no needed for this fully automated process, which is usually transparent to the programmer). Therefore, the teacher has inserted all the comments to make your understanding easier. You must follow the assembly code, understand the meaning of every instruction, and correlate them with the original C code.

As you can see in Eclipse, and the assembly code below, the linker has assigned the address 0x0c0204e0, as the staring address of the function *countWhites*.

In the next page, the blue box encloses the code corresponding to the "*loop i*". Similarly, the green box contains the code of the "*loop j*", which is repeated for each iteration of the *loop i*.

```
//C code for the function "countWhites"


void countWhites(unsigned char mat[N][M], unsigned char vector[N]) {
    int i,j;

    for (i=0;i<N;i++) {
        vector[i]=0;
        for (j=0; j<M; j++) {
            if ( mat[i][j] == 255) vector[i]++;
        }
    }
}
```

```
//Assembly code

//36      void countWhites(unsigned char mat[N][M], unsigned char vector[N]) {
countWhites:
0c0204e0:  push {r11}           @ (str r11, [sp, #-4]!)  PROLOG (recall that r11 is the fp)
0c0204e4:  add r11, sp, #0
0c0204e8:  sub sp, sp, #20      @reserve space for 5 local variables
0c0204ec:  str r0, [r11, #-16]  @store initial address of "mat" (1st parameter) in fp-16
0c0204f0:  str r1, [r11, #-20]  @store initial address of "vector" (2nd parameter) in fp-20

// 39       for (i=0;i<N;i++) {

0c0204f4:  mov r3, #0
0c0204f8:  str r3, [r11, #-8]                 @i=0 (i located in fp-8)
0c0204fc:  b 0xc020584 <countWhites+164>
```

```
// 40            vector[i]=0;
0c020500:     @loop i
             ldr r3, [r11, #-8]       @r3=i
0c020504:  ldr r2, [r11, #-20]     @r2=initial address of vector
0c020508:  add r3, r2, r3          @r3=address of vector[i]
0c02050c:  mov r2, #0              @observe the reuse of registers!!
0c020510:  strb r2, [r3]           @vector[i]=0

// 41            for (j=0; j<M; j++) {
0c020514:  mov r3, #0
0c020518:  str r3, [r11, #-12]     @j=0 (located in fp-12)
0c02051c:  b 0xc02056c <countWhites+140>

// 42              if ( mat[i][j] == 255) vector[i]++;
0c020520:       @loop j
             ldr r3, [r11, #-8]       @r3=i
0c020524:  lsl r3, r3, #7          @r3 = i*128
0c020528:  ldr r2, [r11, #-16]     @r2=initial address of mat
0c02052c:  add r2, r2, r3          @r2=initial address of mat + i*128
0c020530:  ldr r3, [r11, #-12]     @r3=j
0c020534:  add r3, r2, r3          @r3 = (initial address of mat + i*128)+j
0c020538:  ldrb r3, [r3]           @r3=mat[i] [j]
0c02053c:  cmp r3, #255    ; 0xff
0c020540:  bne 0xc020560 <countWhites+128> @if mat[i][j] not white, go to endif

                                   @we have detected a “white”
0c020544:  ldr r3, [r11, #-8]      @r3=i
0c020548:  ldr r2, [r11, #-20]     @r2=initial address of vector
0c02054c:  add r3, r2, r3          @r3=address of vector[i]
0c020550:  ldrb r2, [r3]           @r2=vector[i]
0c020554:  add r2, r2, #1          @r2=vector[i]++
0c020558:  and r2, r2, #255        @ reduce r2 to just 8 bits (“and” with 0xff)
0c02055c:  strb r2, [r3]           @store r2 into vector[i]

// 41            for (j=0; j<M; j++) {   (code for end of loop j)
0c020560:       @endif
             ldr r3, [r11, #-12]      @r3=j
0c020564:  add r3, r3, #1          @r3++
0c020568:  str r3, [r11, #-12]     @store r3 into j
0c02056c:  ldr r3, [r11, #-12]     @r3=j
0c020570:  cmp r3, #127            @if j <= 127 execute loop again
0c020574:  ble 0xc020520 <countWhites+64> @branch to loop j

// 39          for (i=0;i<N;i++) {     (code for end of loop i)
0c020578:  ldr r3, [r11, #-8]
0c02057c:  add r3, r3, #1
0c020580:  str r3, [r11, #-8]
0c020584:  ldr r3, [r11, #-8]
0c020588:  cmp r3, #127    ; 0x7f
0c02058c:  ble 0xc020500 <countWhites+32> @branch to loop i
// 45       }
0c020590:   add sp, r11, #0        @ EPILOG
0c020594:   pop {r11}              ; (ldr r11, [sp], #4)
0c020598:   bx lr
```

<u>EXERCISE</u>

Write the equivalent assembly code, using the following “equ” declarations:

```
.equ     i        -8
.equ     j        -12
.equ     mat      -16
.equ     vector   -20
```

What are the main advantages of the new version?