[416] Explanation Q&A : Assignment 5

Kim, Charles <ckim@Howard.edu>
Wed 11/29/2023 02:59 PM

(Q) Hi Dr.Kim, I'm still having trouble with assignment 5. I wish I could attend your office hours but I'm unable to. I'm having trouble with the counting part from 0 to 999. I'm still confused about how it's supposed to work. I read the guide you gave us but I'm still confused. Is there any pseudo code I could follow? Thank you.
(A) Referring to the "Coding Design" I sent before,
(a) To display on seven2 seg, we need the following pattern in a register (which holds 4 bytes): 00 00 XX YY, where XX is a 2-digit hexa number for the first letter (of your name), and YY a 2-digit hexa number for the second letter. In the "Design", XX is 0x5F (for 'a') and YY 0x7C (for 'b'). So if we store these together in to r2, as r2 = 00 00 5F 7C. Now we store this to seven2 register, _ _ ab will be displayed. Since this letter pattern does not change, I do this very early, and do not do/visit again.

(b) To display the 3rd letter ('c" in the "Design") along with 3 numbers, we need to allocate a register to hold, again, 4 bytes, each representing a letter or a digit. Since 7-seg pattern for 'c' is (as in "Design") 0x58 and this must be stored at the highest byte as r3 = 58 00 00 00, and the trailing 3 bytes of 0's are to be determined for a certain number such as 235. Once we find corresponding pattern for 2, 3, and 5, we put these patterns in the 00 00 00 part of the bytes. Therefore, if AA is the seg pattern for '2', BB for '3', and CC for '5', then the final value for r3 would be, r3 = 58 AA BB CC. If we store r3 to seven1, c 2 3 5 would be displaced. Since _ _ ab was already in display, the overall display on six 7-segs would look like:
a b c 2 3 5.

(c) Now, how do we get 2, 3, 5 and corresponding AA, BB, and CC from a number 235? First we apply, successive subtraction as division to find the quotient from 235/100 = 2(Q1) -- 35(R). Let's store Q1=2 in to r4 = 00 00 00 02. Now we do 35/10 = 3(Q2) --5(R). Let's store Q2 and R in to r5 and r6, respectively, : r5 = 00 00 00 03 and r6 = 00 00 00 05. We found 2, 3, and 5 from 235. The next step is to find AA for '2' and BB for '3' and CC for '5'.

(d) Since we used a lookup table for segPattern as

```
.data
segPattern:
.word 0x3F, 0x06, 0x5B, 0x4F,0x66, 0x6D, etc etc
for    '0'    '1'      '2'    '3'    '4'     '5'
```

The array segPattern indicates and is the same as the address of the first element (which is 0x3F for

'0'). The data for '1' is 0x06 and it is stored in the memory location at { segPattern + 4}, remembering that .word means 4 bytes, which in turn means 4 memory locations. The data for '4' is 0x66 and is stored at {segPattern + 16} location. The numbers 4 and 16 are called offset (from the based memory location of segPattern array). And if we compare a number (such as 2) and its offset for its pattern (such as '2'), we get offset = number*4. In other words, for 3, its pattern '3' is located at the memory offset by 3*4 = 12. For 5, its pattern '5' is located at offset 5*4 = 20. Since multiplication by 4 is the same as Logical Shift Left (LSL) twice, we could do LSL r?, r?, #2 to be equivalent to MUL r?, r?, #4.

(e) Now, for a pattern (AA, BB, or CC as explained above) for each number stored in r4, r5, r6, can be loaded from the segPattern by,
LDR r11, =segPattern //r11 holds the base address of the segPattern arrary ('0')
For '2', LDR r8, [r11, r4, LSL #2] // load a word from the location made by r11 + r4*4, i.e. base addr + offset.
For '3', LDR r10, [r11, r5, LSL #2] //load a word from r11 + r5*4 location
For '5', LDR r12, [r11, r6, LSL #2] //load a word from r11 + r6*4 location
Now r8 holds 0x00 00 00 5B for '2', r10 0x00 00 00 4F for '3', and r12 0x00 00 00 6D for '5'.

(f) Now we need some arrangement so that in 7-segments, they are displayed as _ 2 3 5. So while we keep r12 (which is '5' occupying the units position), we need to shift r10 (which is '3' occupying the tens position) to left 8 times (a byte shift left) so that new r10 = 0x 00 00 4F 00. And we do similarly for r8 (which is '2' occupying the hundreds position) (shifting left 16 times) so that the new r8 = 0x00 5B 00 00. If we add all three, r8 and r10 and r12 to r12, then r12= 0x 00 5B 4F 6D.
Now if we add the 3rd letter stored in r3, by adding r12 and r3, we have r12 = 0x58 5B 4F 6D representing c '2' '3' '5'.
Finally we send r12 to seven1 to display c 2 3 5.

(g) If your code works for this specific number, only then try to make a loop so that the number starts from 0 to 999.

(h) Good luck!

Dr. Kim