ECS502U Lab: Week 3

The point of this lab is to make you solve small problems using 8051 assembler code. You will be using the MCU8051 IDE for this lab. On the QMPlus module page you can find 8051 assembler instruction documentation, including a brief "cheat sheet." We suggest you create a directory for all your code for ECS502U, and then a sub-directory for this week.

QUESTION 1 Implement an 8051 program adding two 16-bit numbers. It should read the numbers to be added via the ports of the microcontroller. Store the result in registers R0, R1, R2. Then answer the following questions:

1. Provide your code:
2. Describe your algorithm, using written text or visualisation as appropriate:
3. Why is this an appropriate solution? To what degree is it optimal?

QUESTION 2 Add a sequence of 5 8-bit numbers, reading each number from port P1. Output the final result via P2 and P3. You may find the following code fragment (also available as question2.asm) useful to wait for a number of clock ticks.

```
ORG 0H
START: LCALL DELAY ; time delay to wait for next input value
       ; your code
       SJMP START ; keep doing this for ever
; ----------the delay subroutine
ORG 300H ; put the subroutine at 300H
DELAY: MOV R5, #0FFH ; R5 = 255
AGAIN: DJNZ R5, AGAIN ; repeat until R5 is zero
       RET ; return to caller (when R5 = 0)
```

Then answer the following questions:

1. Provide your code:
2. Describe your algorithm, using written text or visualisation as appropriate:
3. Why is this an appropriate solution? To what degree is it optimal?

QUESTION 3

(a) Which of the 8051's memory banks is active initially after power-up, before executing any instructions? Explain how you can switch to memory bank 1 and why you should be very careful if you use a memory bank other than memory bank 0 if also using push and pop instructions.

(b) Explain the role of the individual bits in the Program Status Word (in the 8051's special function registers).

Solutions

The following solutions are for verification purposes, i.e. so as to compare with your own ones. Note, though, that some exercises may have more than one (correct) solution.

QUESTION 1

1. **ORG** 0H
   NOP *; no−op just to have time to input test values into the simulator*
   MOV A, P0 *; less significant bytes are read from P0 and P1, respectively*
   ADD A, P1 *; add up the lower bytes*
   MOV R0, A
   MOV A, P2 *; most significant bytes are read from P2 and P3, respectively*
   ADDC A, P3 *; add with carry*
   MOV R1, A
   CLR A
   ADDC A, #0 *; add carry to 0*
   MOV R2, A

2. The above program reads the two 16-bit numbers from P2/P0 and P3/P1, respectively. As the 8051 can only handle 8-bit numbers, the lower and higher bytes are added up independently. This just requires handling potential overflow, which will result in the carry-flag being set. Hence ADDC is used.

3. Simulating using the values 5612H, 7834H yields the expected value, so seemingly it works fine. The solution is free from branching and thus fast, but it may be possible to implement a more memory-efficient program by using a loop instead of multiple ADD/ADDC instructions.

QUESTION 2

1. **ORG** 0H
   START: **CLR** A *; clean initial state*
   MOV R0, #0
   MOV R1, #0
   MOV R2, #5 *; 5 numbers to be added*
   MORE: **LCALL** DELAY *; time delay to wait for next input value*
   ADD A, P1 *; add the new value onto the existing value*
   MOV R1, A *; back up A into R1*
   CLR A
   ADDC A, R0 *; handle a possible carry*
   MOV R0, A *; maintain it in R0*
   MOV A, R1 *; put R1 back into A*
   DJNZ R2, MORE *; loop until R2 reaches 0*
   MOV P2, R1 *; output values − lower byte*
   MOV P3, R0 *; high byte*
   SJMP START *; keep doing this for ever*
   ; −−−−−−−−−−the delay subroutine
   ORG 300H *; put the subroutine at 300H*
   DELAY: **MOV** R5, #0FFH *; R5 = 255*

```
AGAIN: DJNZ R5, AGAIN ; repeat until R5 is zero
       RET ; return to caller (when R5 = 0)
```

2. The program loops forever, and in each iteration first resets all storage. Then 5 values are read from P1, added, and stored in R1, with a possibly carry stored in R0. Once the inner loop terminates, the values from R1 (lower byte) and R0 (collated carries) are written to P2 and P3, respectively.

3. Simulation using 08H (five times) and 40H (five times) yields the appropriate values. The loop avoid copy&pasting the same code multiple times, thus reducing the code size.

QUESTION 3

(a) Bank 0 is initially selected.

Bits 3 and 4 in the program status register determine the register bank to be used. For bank 0 bits 3 and 4 = 0. For bank 1 bit 3 = 1 and but 4 = 0.

Pop and push instructions store data on the stack and the stack uses memory locations starting at 08H which is R0 for register bank 1. Exactly how much memory is used by the stack depends on how many items are pushed onto the stack, but it can occupy all 24 memory addresses for the register banks 1, 2 and 3. Data in these banks can be overwritten by stack operations. The solution is to adjust the stack pointer to a higher region of memory (i.e. 30H)

(b)

- CY PSW.7 Carry flag
- AC PSW.6 Auxiliary carry flag
- F0 PSW.5 Available for general use by the user
- RS1 PSW.4 Register bank selector bit 1
- RS0 PSW.3 Register bank selector bit 0
- OV PSW.2 Overflow flag
- – PSW.1 User definable bit
- P PSW.0 Parity flag. This is set or cleared on each cycle depending on the number of 1 bits in the accumulator (even parity)