CS 342302 Operating Systems

Fall Semester 2021

Prof. Pai H. Chou

Weekly Review 2

The questions here serve the purpose of reviewing concepts from the lecture, and expect the concepts to be tested on the midterm and final. However, they are by no means exhaustive. Anything covered in the lecture and projects can be tested.

## 1. Definitions and Short Answers - week 2 (9/20 lectures)

1. What is the **kernel** part of an OS?
2. What is a **system program**?
3. Is a **web browser** part of an OS? part of the kernel? part of system programs?
4. **Middleware** is a set of software frameworks that provide additional services to what kind of users? What would be some example features?
5. As a resource manager, what kinds of resources does an OS manage?
6. What does **API** stand for? What kinds of API does an OS provide?
7. What is a **bootloader**? How is it related to an OS?
8. How does a bootloader load an OS?
9. How does a CPU **send a command** to a device controller?
10. From textbook (page 7) A **device controller** maintains some "**local buffer storage**" and a set of **special-purpose registers**". Where do these storage and registers reside? On CPU? in main memory? in cache? outside CPU?
11. What is a **device driver**? hardware or software? part of the OS or in an application program? What does it do?
12. What does **IRQ** stand for, and what is its purpose?
13. How does a program for 8051 know when the UART has received a byte?
14. What does **ISR** stand for, and when is it invoked?
15. What is an **interrupt vector**?
16. What is an **interrupt vector table**?
17. How does a processor decide which ISR to execute when there are multiple I/O devices?
18. How does a processor continue executing the user program after an ISR finishes?
19. What is a **nested interrupt**?
20. What is **interrupt chaining**? (page 10 in textbook) Why is it useful?
21. How does the OS protect CPU time as a resource by preventing a user program from hogging the CPU without making a system call?
22. What is **volatile memory** vs. **nonvolatile memory**? What are examples of each kind?
23. What is the purpose of a **cache**? Can a processor run without a cache?
24. What is the principle of **locality**? What are **two kinds of locality**?
25. What does **DMA** stand for, and why is it used?
26. What are steps in a DMA setup, transfer, and completion?

## EdSim51 and 8051 - week 2 (9/22 lecture)

1. What are the **(direct) addresses** of the 8051 general-purpose I/O (GPIO) ports P0, P1, P2, and P3?

A: The direct addresses of the 8051 GPIO ports P0, P1, P2 and P3 are respectively 80H, 90H, A0H, B0H.

1. What is the meaning of each of the following instructions in 8051?
   1. MOV A, #1
   2. MOV A, 1
   3. MOV A, R1
   4. MOV A, @R1
   5. MOV R1, A
   6. MOV R1, #1
   7. MOV R1, 34H
   8. MOV @R1, A
   9. MOV @R1, #1
   10. MOV @R1, 34H
   11. MOV 34H, A
   12. MOV 34H, #56H
   13. MOV 34H, 56H
   14. MOV 34H, R1
   15. MOV 34H, @R1

a. Immediate addressing mode, take the decimal value 1 (0x01) and move it to the accumulator A.

b. Direct addressing mode, take the data at decimal IDATA address 1(0x01) and move it to register A.

c. Register addressing mode, transfer the data at R0 (and selected bank) into the accumulator A.

d. Indirect addressing mode, use the value stored in R1 as IDATA address and transfer the data at this address to the accumulator A.

e – o Are variations of the cases described in. a – d.

1. Are the following allowed? If not, what assembly code does the intended o

peration, if any?

* 1. MOV R1, R2
  2. MOV A, A
  3. MOV #20, R3

a. This is not allowed. The intended operation can be written as follows in assembly:

CLR A

MOV A, R2

MOV R1,A

b. This is useless and is not allowed. No equivalent assembly code.

c. This is nonsense and is not allowed. No equivalent assembly code.

1. What is a **port latch**?

A:

1. What does it mean that 8051 GPIO ports are **bit addressable**?

A: It means that they may be addressed as bytes or individual bits.

1. What is the Intel assembly notation for "bit 3 of port P2"?

A: P2.3

1. What is the meaning of  
   SETB P1.1  
   CLR P2.3

A: Set the value of bit 1 of P1 as 1. Clear the value of bit 2 in P3.   
and why can't the same effect be achieved using a MOV instruction?

A: Because the MOV instruction is byte addressable. Not bit addressable.

1. If you want to set individual bits of a GPIO port without using SETB and CLR instructions, what instruction can you use? Hint: [8051 User's Guide](https://drive.google.com/file/d/1FJEKnCuO2oDEANp8tDqkzQHOFy4-SKB6/view), Table 3 on page 1-11, "Logical Instructions", find a combination of ANL, ORL, XRL instructions. What **addressing modes** should be used with these instructions? Use these instructions to implement the two-bit set/clear instructions in the previous question.
2. What is a **label** in an assembly language program?

A: A label is a symbolic name for the address where the following instruction is located.

1. Given the sample program for setting P3<4:3>:  
   Top: SETB P3.4  
    SETB P3.3  
    MOV P1, #24H  
    CLR P3.3  
    MOV P1, #24H  
    CLR P3.4  
    SETB P3.3  
    MOV P1, #24H  
    CLR P3.3  
    MOV P1, #24H  
    SJMP Top  
   Rewrite it in C by filling in the blank \_\_\_\_ below:  
    **int** i;  
    **for** (i = 0; i < 4; i++) {  
    P3 = \_\_\_\_\_; // you may change = into &=, |=, etc  
   INT0 }
2. What does the following Intel 8051 assembly code do?  
   Data1: DB "Hello world”

A: DB is an assembler directive that stands for Define Byte. It is used to define 8-bit data.

* 1. Does it occupy any memory? In which space? (CODE? IDATA? XDATA?)

A: It occupies CODE memory.

* 1. What is the closest equivalent statement in C?

A: const char Data1[] = “Hello World”

* 1. Is the assembly version null-terminated? How do you find out?

A: No, it is not null terminated. Consider the following two DB statements:

ORG OOOOH

DB “HELLO WORLD”

DB “HELLO”

When running the program, we see the following in code memory

48H 45H 4CH 4CH 4FH 20H 57H 4FH 52H 4CH 44H 48H 45H 4CH 4CH 4FH 20H 57H 4FH 52H 4CH 44H

The ASCII code for “H” is 0x48 AND FOR “D”0x44. However, in the blue-colored hex digits above, we don’t see the null terminator 00H after the ASCII code corresponding to “D”.

1. Given the Intel 8051 assembly code:  
   Data2: DB 25
   1. How many bytes does the 25 data occupy?

A: 1 byte

* 1. What kind of address is Data2? In other words, what space (CODE, IDATA, XDATA, etc) and how many bytes?

A: It occupies CODE memory.

1. Given the Intel 8051 assembly code  
    COUNT EQU 25
   1. How many bytes does the above line occupy in the assembled code, if any and in which memory?

A: It occupies no space.

* 1. What is the equivalent statement in C?

A: #define COUNT 25

1. If you want to display the digit "7" on the seven-segment LED with the additional vertical line on the upper left instead of just an upside-down L, what value do you write to P1?

A: MOV P1, #0D8 (dot off) or #58 (dot on)

1. What is the advantage of using CLR A instruction over MOV A, #0 instruction, which does the same thing (assign A = 0)? Hint: look up how these instructions are encoded.

A: From the 8051 manual, the CLR A instruction takes 1 byte to execute while the MOV instruction takes 2 bytes.

1. What is the difference between A and ACC in 8051 assembly? Why do I have to say PUSH ACC and cannot say PUSH A? (Similarly POP ACC but not POP A)?

A: A and ACC refer to the same register however they do so in different. Addressing modes. ACC is the IDATA name for the accumulator and PUSH and POP instructions must use the direct address of the accumulator register ACC.

1. What is the effect of PUSH and POP on the stack? Explain in terms of the stack, i.e., SP and the memory location pointed by SP.

A: PUSH and POP increment and decrement the SP by 1 respectively.

1. What does LCALL Display do (where Display is a code label)? Explain in terms of the program counter and the stack.

A: The LCALL Display instruction calls the Display subroutine. When LCALL is called, it pushes the return address of the next instruction after LCALL into the stack.

1. What does the RET instruction do? Does RET know if the subroutine was originally called by LCALL or ACALL instruction?

A: Returns from subroutine. The RET instruction returns control to the caller after finishing execution of the subroutine. There is no difference. if the call originated from an LCALL or ACALL instruction, RET only returns control to caller.

1. What does the following instruction do:  
   MOV DPTR, #LEDdata  
   Assume LEDdata corresponds to address 0x2468, what is the new value of DPL and DPH?

A: DPL = 0x68, DPL = 0x68

1. What is the meaning of the instruction  
   MOVC A, @A+DPTR  
   and how is it different from  
   MOV A, @Ri  
   where Ri is either R0 or R1?
2. On a UART, what does RxD stand for? What is TxD?

A: RxD stands for receiver while TxD stands for transmitter.

1. How should the RxD and TxD signals of one system be connected to the RxD and TxD signals of another system that it communicated with?

A: The RxD of one system should be connected to the TxD of the other system and vice-versa.

1. On the 8051, what is the purpose of the SBUF special-function register? What happens when you move data **to** SBUF? move data **from** SBUF?
2. How does a program know when there is valid data to be read from SBUF?
3. Why is it necessary to clear RI bit after reading from SBUF?
4. What is the meaning of the code  
   Here: JNB RI, Here

Rewrite this assembly code as a C statement.

A: while(!RI){

//LOOP

}

1. What does **4800 baud** mean for a UART?
2. On the 8051, what is the **interrupt vector** (i.e., address of the ISR) for Reset?
3. The interrupt vector for interrupt is at 03H, which leaves only 3 bytes for the interrupt vector for Reset. Why is 3 bytes sufficient?
4. The interrupt vector after 03H is 0BH, leaving only 8 bytes for the the INT0. Why is 8 bytes sufficient?
5. When is the TI flag set to 1 by the UART?
6. When the UART's ISR is invoked, is it caused by RI or TI becoming 1? Can it be both?
7. What is the purpose of the EA bit, which stands for "enable all (interrupts)"?

## 

## 3. Short Assembly Programs

1. Modify the Display subroutine so that it not only displays the digit but also switches to the next 7-segment LED. It just needs to a different digit at a time; does not need to keep all four.  
     
   Display: MOV DPTR, #LEDdata  
    MOVC A, @A+DPTR  
    ;; your code to increment the LED ID,  
    ;; which is defined by P3.3 and P3.4  
    MOV P1, A ;; light up the selected 7-segment LED  
    RET  
   LEDdata: DB 0c0H,0F9H,0A4H,0B0H,99H,92H,82H,0F8H,80H,90H

Test your code with polling version of UART (e.g., slide 12 of 02-EdSim51-IO.pdf). Remember to initialize UART and Timer as well as P3.3 and P3.4. Try it out by typing digit characters into the Tx window and click send.

1. Convert your code to interrupt version, shown on slide, based on slide 26-27.