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.

@: take the register content as pointer and dereference it

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

1. What is the **kernel** part of an OS?

the core part that is resident and governs os functionality

1. What is a **system program**?

work at system level, serve the entire system, tightly associated with the os

1. Is a **web browser** part of an OS? part of the kernel? part of system programs?

middleware

1. **Middleware** is a set of software frameworks that provide additional services to what kind of users? What would be some example features?

developers, general enough for apps to be built on it

1. As a resource manager, what kinds of resources does an OS manage?

CPU time, storage space, use of io devices

1. What does **API** stand for? What kinds of API does an OS provide?

application programming interface

use of memory, io devices, storage, communication

1. What is a **bootloader**? How is it related to an OS?

ROM code executed on power up, set up enough to load the os kernel into memory

closely related but usually not considered part of the os

1. How does a bootloader load an OS?

communicate with disk or network, loads part of the os, the rest can be loaded by part of the os

1. How does a CPU **send a command** to a device controller?

bus transaction, interrupt request

1. 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?

outside the processor not part of the instruction set

1. What is a **device driver**? hardware or software? part of the OS or in an application program? What does it do?

software, talk to the device controller

1. What does **IRQ** stand for, and what is its purpose?

interrupt request, a way for device controller to notify the CPU

1. How does a program for 8051 know when the UART has received a byte?

RI==1

1. What does **ISR** stand for, and when is it invoked?

interrupt service routine

when there is an interrupt or upon data ready

1. What is an **interrupt vector**?
2. What is an **interrupt vector table**?

=interrupt vector

array of addresses of ISRs, indexed by interrupt number

1. How does a processor decide which ISR to execute when there are multiple I/O devices?

interrupt number indexed into interrupt vector

1. How does a processor continue executing the user program after an ISR finishes?

save the address of the routine before jumping to ISR

restore the following line’s address from stack

1. What is a **nested interrupt**?

when running interrupt, another interrupt could happen

due to devices of different priority levels

1. What is **interrupt chaining**? (page 10 in textbook) Why is it useful?

each element in the interrupt vector points to the head of a list of interrupt handlers.

When an interrupt is raised, the handlers on the corresponding list are called one by one,

until one is found that can service the request.

This structure is a compromise between the overhead of a huge interrupt table and the inefficiency of dispatching to a single interrupt handler.

1. How does the OS protect CPU time as a resource by preventing a user program from hogging the CPU without making a system call?

timer, generate interrupt every specific time

1. What is **volatile memory** vs. **nonvolatile memory**? What are examples of each kind?

volatile: reg, cache, memory

nonvolatile: solid-state disk, hard disk, optical disk, magnetic tapes

1. What is the purpose of a **cache**? Can a processor run without a cache?

get faster access to memory

1. What is the principle of **locality**? What are **two kinds of locality**?

temporal, spatial

1. What does **DMA** stand for, and why is it used?

direct memory access

transfer a block of data is more efficient than that of a byte or a word

1. What are steps in a DMA setup, transfer, and completion?

set up: set up a shared memory structure so the device controller on the bus can read/write a region of memory on its own

transfer blocks of data between device and memory

completion: controller sends an interrupt back to the processor

## 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?

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
2. Are the following allowed? If not, what assembly code does the intended operation, if any?
   1. MOV R1, R2

go through A or use immediate

* 1. MOV A, A
  2. MOV #20, R3

1. What is a **port latch**?
2. What does it mean that 8051 GPIO ports are **bit addressable**?

can access the port bit by using P1.1

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

P2.3

1. What is the meaning of  
   SETB P1.1  
   CLR P2.3  
   and why can't the same effect be achieved using a MOV instruction?

set port 1 bit 1

clears port 2 bit 3

MOV is for byte access, not for bit access

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.

SETB: ORL <byte> #0FFH

CLR: ANL <byte> #0H

indirect addressing mode

2-bit set: ……

2-bit clear: ……

1. What is a **label** in an assembly language program?

symbolic name for target location

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?

DB = define byte   
Data1: DB "Hello world"

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

Yes

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

const char DATA1[] = “Hello world”

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

yes

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

1

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

1

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?

0

* 1. What is the equivalent statement in C?

#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?

0xD8

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

take less 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)?

push, pop param only in direct addressing mode

can only push idata address

ACC: idata name for the accumulator(direct address)

A, ACC: refer to same thing but in different addressing mode

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.

save & restore the value

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

push the return address(the address of the next instruction) onto the stack and transfer control

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

LCALL = long call, ACALL = absolute call

return from subroutine

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(8 bit)?

move the actual address of LEDdata into DPTR(16 bits)

DPH = 0x24, DPL = 0x68

1. What is the meaning of the instruction

MOVC = mov code memory  
MOVC A, @A+DPTR  
and how is it different from  
MOV A, @Ri  
where Ri is either R0 or R1?

1. On a UART, what does RxD stand for? What is TxD?

receive data, transmit data

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?

crossed, RxD <-> TxD

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?

SBUF = serial buffer

to read or write

TI==1, RI==1

1. How does a program know when there is valid data to be read from SBUF?

check if RI==1

1. Why is it necessary to clear RI bit after reading from SBUF?

wouldnt know if you are ready when next time polling, might access old 1

1. What is the meaning of the code  
   Here: JNB RI, Here

Rewrite this assembly code as a C statement.

JNB = jump if not bit

while(!RI);

1. What does **4800 baud** mean for a UART?

clock rate 4800 on both sides

1. On the 8051, what is the **interrupt vector** (i.e., address of the ISR) for Reset?

0000H

1. The interrupt vector for interrupt is at 03H, which leaves only 3 bytes for the interrupt vector for Reset. Why is 3 bytes sufficient?
2. The interrupt vector after 03H is 0BH, leaving only 8 bytes for the the INT0. Why is 8 bytes sufficient?
3. When is the TI flag set to 1 by the UART?

when there is valid data to be transmitted to SBUF

when ready for next byte

1. When the UART's ISR is invoked, is it caused by RI or TI becoming 1? Can it be both?

Yes, Yes

1. What is the purpose of the EA bit, which stands for "enable all (interrupts)"?

all the ones that are individually enabled are actually enabled

quick way to turn on all interrupts, and restore all enabled interrupts without affecting the

individual ones

## 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.