CS 342302 Operating Systems

Fall Semester 2021

Prof. Pai H. Chou

Weekly Review 6

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. EdSim51 SDCC

1. What is the difference between a subroutine and a function? Which one is more complicated?
2. What instruction can be used for calling a subroutine?
3. What instruction is used for returning from a subroutine call?
4. The subroutine-call instruction needs to specify the address of the subroutine to call, but the return instruction just returns without operand. How does it know where to return to?
5. What is the power-on value of the stack pointer SP?
6. Which way does the 8051 stack grow? That is, when you PUSH, does the processor increment or decrement the stack pointer?
7. if SP is currently pointing to address 8H, after a PUSH instruction, what address contains the newly pushed value, and what is the new value of SP?
8. if SP is currently pointing at address 9H, what address is the POP value read from?
9. Can the program counter (PC) be modified by a MOV instruction? If not, what instruction (or sequence of instructions) can be used
   1. to assign a constant address to the PC?
   2. to assign a variable address in DPTR?
10. Is 8051 a big-endian or little-endian architecture?
11. If you do an LCALL to target address 1234H, and the SP is currently pointing at address 7H, what are the new values at address 8H and at address 9H after the call, and what is the new value of SP?
12. What is the addressing mode accepted by PUSH and POP instructions on 8051?
13. Does 8051 allow PUSH A or POP A? If not, how do you push a value from the accumulator to the stack or pop a value from the stack into the accumulator?
14. Does 8051 allow PUSH R2 or POP R2? If not, how does SDCC generate code that pushes a value from register R2 or pop a value into register R2?
15. Why does C compiler need to save registers before a function call and restore registers after a function return?
16. What does JC instruction do?
17. How does SDCC pass parameter to a function that expects
    1. a single-byte parameter?
    2. two single-byte parameters?
    3. a single two-byte parameter?
18. How does SDCC return a value from a function:
    1. a single byte return value?
    2. a two-byte return value?

2. Cooperative Threads

1. Why is 8051 called a **Harvard architecture**?
2. What defines the context of a thread on an 8051?
3. What does this C declaration do?  
   typedef void (\*FunctionPtr)(void);
4. In the testcoop.c file, what does this code do:  
   void \_sdcc\_gsinit\_startup(void) {   
    \_\_asm  
     ljmp \_Bootstrap  
     \_\_endasm;  
    }
5. Why is the C function Bootstrap named \_Bootstrap in assembly (i.e., prepended with an underscore)?
6. In the Producer-Consumer example, what are the different ways threads can be created to run the Producer and Consumer functions?
7. In the memory allocation scheme for cooperative threads, what is the **address range of the stack** for thread 0? thread 1? thread 2? thread 3?
8. Register banks
   1. Which bits in which register do you set to select the register bank?
   2. When <RS1:RS0> = 2 (i.e., PSW<4:3> = 2), which RAM addresses get mapped to registers R0-R7?
9. What are the purposes of the macros
   1. SAVESTATE?
   2. RESTORESTATE?
   3. Is it necessary to push the registers R0-R7 onto the stack during SAVESTATE and pop them back into R7-R0 during RESTORESTATE?
10. Memory for variables
    1. What region of memory is available for use by the thread manager and application program as global variables after the register banks and stacks have been allocated?
    2. What is the purpose of the **thread bitmap**, which is a (global) variable maintained by the thread manager?
11. What are the four major steps of the Bootstrap routine?
12. Bootstrapping
    1. [slide 27] How should Bootstrap initialize the thread bitmap?
    2. [slide 28] When Bootstrap calls ThreadCreate(main) to create the context for main, **which stack is used**, and what is the **content of that stack** when ThreadCreate is first entered? How is the parameter passed and what does it mean?
    3. Does ThreadCreate() modify the current thread ID?
    4. [slide 32] How does Bootstrap() start running the main() function whose context has been set up by ThreadCreate(main)?
    5. After Bootstrap starts running main's thread, what happens to the **initial stack** that Bootstrap used (from part b of this question)?
13. Thread creation
    1. If you can assume all threads must **explicitly** call ThreadExit() if they ever exit (and never **implicitly** call ThreadExit() when the function finishes), to create the new thread-0's context that can be restored to run main from the beginning, how should the ThreadCreate() function initialize the content of stack-0 (and set its SP value) and why?
    2. How should ThreadCreate() initialize a thread's stack content before RESTORESTATE in order to support **implicit call** to ThreadExit() upon function return?
14. [slide 33] When main() calls ThreadCreate(Producer),
    1. which **stack** does main use to make this call?
    2. what is that stack's **content** upon entering the ThreadCreate function, assuming we do not need to support implicit ThreadExit (and therefore need not push ThreadExit's address upon ThreadCreate)?
15. [slide 34] By the time main() returns from ThreadCreate(Producer), which returns the thread ID of 1 for the newly created thread,
    1. What is on stack 0?
    2. What is on stack 1? and what is its savedSP value?
    3. What is the value of SP?
    4. What is the value of DPL and why?
    5. stack1 contains value 08H at address 56H, which is intended to be restored into PSW when thread1 resumes. Why value 08H?
16. Why should main() call Consumer() instead of creating another thread to run Consumer?
17. [slide 35] When main() calls Consumer(), what is the content of the stack upon entering Consumer?
18. [slide 36] When Consumer() calls ThreadYield(), what is the content of the stack upon entering ThreadYield()?
19. [slide 37] What are the **three major steps** in ThreadYield()?
20. What does the thread0's stack look like after ThreadYield() calls SAVESTATE?
21. When ThreadYield() is picking the next thread to context switch to,
    1. what **data structure** does it check to know what threads are available to choose from?
    2. what **stack** is it running on during choosing the next thread?
    3. what happens if an **interrupt** happens during this time? Can an interrupt cause any problems?
22. [slide 39-40] How does the SP change from the time ThreadYield() **just before** RESTORESTATE to **just after** RESTORESTATE?
23. [slide 40] After ThreadYield() does RESTORESTATE, where and how does it "continue" running the thread it is restoring?