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?

A: A subroutine assumes a hardware-supported stack and PC while a function is a compiler-supported wrapper around a subroutine. Functions save registers, passing parameters and auto-locals. Functions are more complicated.

1. What instruction can be used for calling a subroutine?

A: LCALL is the native instruction used to call a subroutine.

1. What instruction is used for returning from a subroutine call?  
   A: RET is the instruction used to returning from a subroutine call.
2. 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?

A: It automatically returns to the next instruction (e.g.from LCALL instruction to LJMP main) pointed by the stack. The return address is then the next instruction that was pushed in the stacked before the subroutine call and popped from the stack after RET is called.

1. What is the power-on value of the stack pointer SP?

A: At power-on the value of the SP is 07H.

1. Which way does the 8051 stack grow? That is, when you PUSH, does the processor increment or decrement the stack pointer?

A: It increments the stack pointer by 1 whenever you PUSH.

1. 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?

A: SP = 08H +1 = 09H

1. if SP is currently pointing at address 9H, what address is the POP value read from?

A: 09H since it is post-decrement. The decrement occurs after the POP operation.

1. 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? MOV can be used.

A: No, the PC may not be modified by a MOV instruction. You may use instructions such as LCALL, LJMP, RET or RETI.

1. Is 8051 a big-endian or little-endian architecture?

A: 8051 has a little-endian architecture.

1. 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?

A: Since the 8051 has a little-endian architecture then the address 8H would contain the lower-order byte 34H and the address 9H would contain the higher-order byte 12H.

1. What is the addressing mode accepted by PUSH and POP instructions on 8051?

A: The PUSH and POP instructions only accept the direct addressing mode.

1. 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?

A: 8051 does not allow register addressing so PUSH A and POP A are allowed. To push or pop a value from the stack into the accumulator you must use the following: PUSH 0E0H or POP 0E0H where E0 is the address of register A. We can also use PUSH ACC or PUSH ACC.

1. 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?

A: No, it is not allowed. SDCC then uses the symbolic name for register R2 as “ar2”to directly address the R2 register.

1. Why does C compiler need to save registers before a function call and restore registers after a function return?

A: To continue execution in the next instruction after control is returned. to the calling code. Saving registers save the state of the calling code before calling the function while restoring registers allows execution to continue.

1. What does JC instruction do?

A: The JC instruction branches to the specified address if the carry bit or carry flag is set. Otherwise continue with the next instruction.

1. How does SDCC pass parameter to a function that expects
   1. a single-byte parameter? – Uses DPL
   2. two single-byte parameters? - Uses only DPL and the second single-byte parameter is allocated in the stack.
   3. a single two-byte parameter? – Uses DPL and DPH
2. How does SDCC return a value from a function:
   1. a single byte return value? – Uses DPL
   2. a two-byte return value? - Uses DPL and DPH

2. Cooperative Threads

1. Why is 8051 called a **Harvard architecture**?

A: Because it separates address spaces for code and data.

1. What defines the context of a thread on an 8051?

A: In 8051 the context of a thread is defined by its register bank, stack pointer and PC value. Given that the 8051 has 4 register banks we can manage exactly 4 threads. To do context switching we must change register bank.

1. What does this C declaration do?  
   typedef void (\*FunctionPtr)(void);

A: Defines a function pointer to a function.

1. In the testcoop.c file, what does this code do:  
   void \_sdcc\_gsinit\_startup(void) {   
    \_\_asm  
     ljmp \_Bootstrap  
     \_\_endasm;  
    }

A: At startup make an LJMP to the bootstrap C function.

1. Why is the C function Bootstrap named \_Bootstrap in assembly (i.e., prepended with an underscore)?

A: The assembler prepends the underscore \_ to denote a C identifier, e.g. A C function name.

1. In the Producer-Consumer example, what are the different ways threads can be created to run the Producer and Consumer functions? ‘

A: Three threads, Two threads and shared-memory communication.

1. 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?

A: Thread 0: 40H – 4FH, Thread 1: 50H – 5FH, Thread 2: 60H – 6FH, Thread 0: 70H – 7FH.Each thread gets 16 bytes! Total gets 64 bytes or half of the data space.

1. Register banks
   1. Which bits in which register do you set to select the register bank?

A: You must set bits 3 and 4 (RS0 and RS1) in the PSW register to select the register bank.

* 1. When <RS1:RS0> = 2 (i.e., PSW<4:3> = 2), which RAM addresses get mapped to registers R0-R7?

A: Since bank 2 is selected the RAM addresses 10H to 17H are mapped to registers R0-R7.

1. What are the purposes of the macros
   1. SAVESTATE? – Save the stack pointer of the current thread. PUSH ACC,B, DPTR PSW on the stack. Saves register bank.
   2. RESTORESTATE? – Reverse operation of SAVESTATE
   3. Is it necessary to push the registers R0-R7 onto the stack during SAVESTATE and pop them back into R7-R0 during RESTORESTATE?

A: No, it is not necessary. As PSW is saved and restored, it automatically saves and restores the register banks

1. 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?

A: Depends if you push the SP for each thread to the first byte of the stack.34H-36H if no. Else 30-32H.

* 1. What is the purpose of the **thread bitmap**, which is a (global) variable maintained by the thread manager?

A: Its purpose is to determine which thread is active/available.

1. What are the four major steps of the Bootstrap routine?

A: 1.Initialize thread manager variables

2.Create a thread for main.

3.Set current thread ID

4.Restore

1. Bootstrapping
   1. [slide 27] How should Bootstrap initialize the thread bitmap?

A: It should initialize the thread bitmap as O000B;

* 1. [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?

A: The initial stack is still used (08H-09H) and it contains the return address of Bootstrap(3) 09H. Parameters are passed through DPL and DPR since main is 2 bytes.

* 1. Does ThreadCreate() modify the current thread ID?

A: No, It doesn’t. It is modified when Bootstrap returns from ThreadCreate(). However, it passes the value of DPL = 0 to the stack which is set in the next step in bootstrap.

* 1. [slide 32] How does Bootstrap() start running the main() function whose context has been set up by ThreadCreate(main)?

A: It stores main() address in the stack and goes there once the ACC,B and other registers are initialized.

* 1. After Bootstrap starts running main's thread, what happens to the **initial stack** that Bootstrap used (from part b of this question)?

A: The initial hardware stack is discarded.

1. 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?

A: It should first push the address of ThreadExit to the stack to support an implicit return.

1. [slide 33] When main() calls ThreadCreate(Producer),
   1. which **stack** does main use to make this call?

A: Stack 0 is used to make this call.

* 1. 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)?

A: The return address of the Consumer call in main (return address =main(2))

1. [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?

The return address of main(2) or the Consumer call.

* 1. What is on stack 1? and what is its savedSP value?

A: The initial values of stack 1 so it can later be restored. savedSP is implementation dependent (Example is 56H).

* 1. What is the value of SP?

A: 3FH in stack 0 where the consumer call in 40H will be made once it returns to main.

* 1. What is the value of DPL and why? DPL = 1 which is returned by thread create 1.
  2. stack1 contains value 08H at address 56H, which is intended to be restored into PSW when thread1 resumes. Why value 08H?

A: To select register bank 1 we need PSW.4 RS1 = 0 and PSW.3 RSO = 1 which is then 08H = 0000-1000B which sets the appropriates bits to select register bank 1.

1. Why should main() call Consumer() instead of creating another thread to run Consumer?

A: Because it is more economical. If we create another thread for the Consumer we would be wasting the main thread.

1. [slide 35] When main() calls Consumer(), what is the content of the stack upon entering Consumer?

A: The return address of main(3) or the instruction after we call the consumer.

1. [slide 36] When Consumer() calls ThreadYield(), what is the content of the stack upon entering ThreadYield()?

A: The stack now contains the return address of main(3) and the return address of consumer(2).

1. [slide 37] What are the **three major steps** in ThreadYield()?

A: SAVE STATE, pick next thread and RESTORESTATE.

1. What does the thread0's stack look like after ThreadYield() calls SAVESTATE?

A: The stack now contains the return address of main(3) and the return address of consumer(2) and the stack register values (ACC, B DPTR, PSW).

1. 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?

A: It checks the threadBitmap to see which threads are available. Finally, it checks savedSP array since when a thread is selected it should call RESTORESTATE and pop the values from the stack.

* 1. what **stack** is it running on during choosing the next thread?

A: Stack 0

* 1. what happens if an **interrupt** happens during this time? Can an interrupt cause any problems?

A:

1. [slide 39-40] How does the SP change from the time ThreadYield() **just before** RESTORESTATE to **just after** RESTORESTATE?

A: The SP now points to the selected thread (thread 1) and RESTORES its state. As soon as the SP changes you switch the threads.

1. [slide 40] After ThreadYield() does RESTORESTATE, where and how does it "continue" running the thread it is restoring?