

# CS 331: Embedded Systems, Fall 2002 Midterm

Name:
NetID:

---

- Do **NOT** open the test until told to do so.
  - Print your name and NetID in the boxes above. Also print your NetID on the upper right corner of every page.
  - You may use one  $8\frac{1}{2}'' \times 11''$  double-sided handwritten sheet of notes and a calculator.
  - If anything is unclear, write down your assumptions.
  - Show your work when appropriate. Clearly indicate your answers by circling them.
  - You have exactly 1 hour and 15 minutes to complete the exam.
  - Good luck!
- 

#	Points	Score	Grader
1	10		
2	6		
3	6		
4	18		
5	12		
6	12		
7	12		
8	10		
9	14		
Total	100		

1. (10 points) Give short answers to the following questions.

- (a) (3 points) Under POSIX, many different processes can use timers simultaneously. However, there is only one hardware timer interrupt. When the timer interrupt is raised, how can POSIX figure out which processes to send a SIGALRM signal? Draw a figure if it helps in illustrating your answer.

**Solution:**

Make a generic timer ISR handler. In addition, create a linked list of software timers (counters). When the generic ISR is invoked, it decrements the counter on each software timer. When a counter reaches zero, it is reset and a SIGALRM signal is sent to the corresponding process.



- (b) (3 points) Should we use `usleep( $n$  microseconds)` to implement a periodic task with a period of  $n$  microseconds? Explain why or why not.

**Solution:**

No. The `usleep()` function cannot be used to accurately implement periodic tasks. A process inside `usleep()` could be preempted and that would cause drift. In addition, using `usleep()` implies that I/O is done in the middle of the program. Preemption during any I/O work results in jitter.

- (c) (4 points) In the Interrupt Model presented in class, the hardware does not automatically save all of the data registers when an interrupt is generated. Instead, the programmer must manually save them as part of the Interrupt Service Routine (ISR). Explain why this makes sense. Give an example if it helps in illustrating your answer.

**Solution:**

Saving data registers that will not be modified by the ISR is unnecessary and wastes resources. Additionally, the breakpoint interrupt may be used with a debugger to change register values in the middle of the program.

2. (6 points) You are trying to observe a periodic signal with frequency components of 8Hz and 10Hz. But there is a high frequency noise spike at 80Hz.

- (a) (3 points) What is a practical lower bound on the sampling frequency in Hz if Digital Signal Processing is used to filter the noise? Show your work.

**Solution:**

$$80 \times 3 = 240\text{Hz}.$$

- (b) (3 points) What is a practical lower bound on sampling frequency in Hz if an analog anti-aliasing filter is used to remove the noise? Show your work.

**Solution:**

$$10 \times 3 = 30\text{Hz}.$$

3. (6 points) Is the following system stable, marginally stable, or unstable? Why? Show your work.

$$x'' - 9x' - 8x = 0$$

**Solution:**

Take the Laplace transform of the differential equation:  $s^2 - 9s - 8 = 0$

Then solve for  $s$ , the eigenvalues:  $s = \frac{9 \pm \sqrt{(-9)^2 - 4 \times 1 \times -8}}{2 \times 1} = \frac{9 \pm \sqrt{113}}{2} \approx \{-0.815, 9.815\}$

Using the quadratic formula we see that one eigenvalue is equal to  $-0.815$  while another is equal to  $9.815$ . Since the real part of one of them is positive, the system is unstable.

4. (18 points) An obscure A/D – D/A card uses 11-bit data registers consisting of a high byte and a low byte. The 11 bits are divided into the most significant 6 bits and the least significant 5 bits. The 6 bits are stored in the most significant bits of the high byte, and the 5 bits are stored in the most significant bits of the low byte. The port address of the low byte is 300H and the high byte is 301H. The card is configured to use a  $-10 \dots +30$  volt range for both input and output.

- (a) (6 points) Write C or C++ code to read in the two bytes and assemble the 11-bit value in an unsigned int. Use the unsigned char `hw_in(int port)` function from the lab.

**Solution:**

```
unsigned char low = hw_in(300H);
unsigned char high = hw_in(301H);
unsigned int final = ((high << 6) | low) >> 3;
```

- (b) (6 points) You just read in the 11-bit value “011 0010 1001” from an A/D channel. What voltage does it represent? Show your work.

**Solution:**

11 bits of resolution gives us a range of  $2^{11} = 2048$  values ( $0 \dots 2047$ ). The conversion formula from a raw value  $r$  to a voltage  $v$  is therefore  $v = \frac{r}{2047} \times 40 - 10$ . The given binary number is  $2^9 + 2^8 + 2^5 + 2^3 + 2^0 = 809$  in decimal. Hence, the represented voltage is  $\frac{809}{2047} \times 40 - 10 \approx 5.8085$  volts.

- (c) (6 points) What 11-bit value represents  $-2.75$  volts? Show your work.

**Solution:**

11 bits of resolution gives us a range of  $2^{11} = 2048$  values ( $0 \dots 2047$ ). The conversion formula from a voltage  $v$  to a raw value  $r$  is therefore  $r = \frac{v+10}{40} \times 2047$ . Hence,  $-2.75$  volts is represented by  $r = \frac{-2.75+10}{40} \times 2047 = 371.01875$ . Rounding off the small fractional part and converting to binary we get “001 0111 0011”.

5. (12 points) Suppose that we have two new assembly commands named **STAQADD** and **STAQSUB**. **STAQADD** pops the 2 most recently pushed values, adds them, and pushes the result back onto the stack. **STAQSUB** pops the 2 most recently pushed values, subtracts them, and pushes the result back onto the stack. See the Intel assembly code below.

STAQADD	STAQSUB
POP AX POP BX ADD AX, BX // AX := AX + BX PUSH AX	POP AX POP BX SUB AX, BX // AX := AX - BX PUSH AX

Suppose we have a stack that starts at segment 1000H (SS = 100H) and has a size of 200H (SP = 200H). Some values are pushed onto the stack and the result is shown here:

Address	Content
11FFH	11H
11FEH	02H
11FDH	13H
11FCH	10H
11FBH	12H
11FAH	52H
11F9H	
⋮	⋮
1001H	
1000H	


SP →

- (a) (6 points) The `STAQADD` instruction is then executed. What will the stack look like afterwards? Annotate the picture below with your solution by filling in the stack contents and drawing an arrow to indicate where `SP` points.

	Address	Content
	11FFH	
	11FEH	
	11FDH	
	11FCH	
	11FBH	
	11FAH	
	11F9H	
	⋮	⋮
SP —	1001H	
	1000H	

**Solution:**

	Address	Content
	11FFH	11H
	11FEH	02H
	11FDH	25H
	11FCH	62H
	11FBH	
	11FAH	
	11F9H	
	⋮	⋮
SP —	1001H	
	1000H	



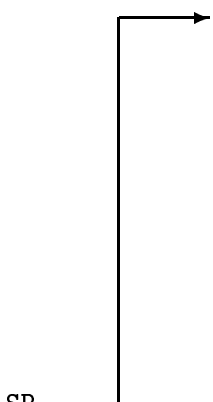
- (b) (6 points) Immediately after executing `STAQADD` in part (a), the `STAQSUB` instruction is executed. What will the stack look like afterwards? Annotate the picture below with your solution by filling in the stack contents and drawing an arrow to indicate where `SP` points.

	Address	Content
	11FFH	
	11FEH	
	11FDH	
	11FCH	
	11FBH	
	11FAH	
	11F9H	
	⋮	⋮
SP —	1001H	
	1000H	

**Solution:**

	Address	Content
	11FFH	14H
	11FEH	60H
	11FDH	
	11FCH	
	11FBH	
	11FAH	
	11F9H	
	⋮	⋮
	1001H	
	1000H	

SP —



## 6. (12 points) PID Control

- (a) (4 points) If a system is moving to the set point too slowly, which gain should you change and how should you change it?

**Solution:**

Increase proportional gain or decrease derivative gain.

- (b) (4 points) If a system has a steady state error (i.e., it cannot reach the set point), what should you do?

**Solution:**

Slowly increase integral gain, but watch out for overshoot and oscillation.

- (c) (4 points) If a system's error tends to grow larger and larger, what can you say about the eigenvalues of the system?

**Solution:**

Real part of one or more eigenvalue(s) is positive.

7. (12 points) Square waves from Lab 2

- (a) (2 points) What is the period in milliseconds of a 160 Hz square wave?

**Solution:**

A 160 Hz square wave should complete a full period in  $\frac{1}{160} = 6.25$  milliseconds.

- (b) (10 points) Suppose we tried to generate this 160 Hz square wave using our Lab 2 code on a new computer system that allows square waves to have a maximum frequency of 200 Hz. What is the period in milliseconds of the wave seen on the oscilloscope screen? Is your answer the same as that from part (a)? Why or why not?

**Solution:**

You will not observe the theoretical period in part (a). Since generating a full period of the square wave requires two transitions, the clock resolution on this new computer system is  $2 \times 200 = 400$  Hz. The period between clock interrupts is therefore  $\frac{1}{400} = 2.5$  milliseconds. A 160 Hz square wave would complete a full period in  $\frac{1}{160} = 6.25$  milliseconds and transition every  $\frac{1}{2 \times 160} = 3.125$  milliseconds. However, since 3.125 is not a multiple of 2.5, each transition will need to wait until the next timer interrupt  $2 \times 2.5 = 5$  milliseconds after the previous transition. Therefore, the period of the full wave is  $2 \times 5 = 10$  milliseconds. You will end up seeing a 100 Hz square wave on the oscilloscope.

8. (10 points) The function `yowza()`, which uses extended AT&T inline assembly, is defined below. What will be printed to the screen when “`yowza(5, 10, 20)`” is called?

Hint: Note the positions of the colons and their meaning for read-only (“input”) and write-only (“output”) variables.

```

1 void yowza(int x, int y, int z)
2 {
3     __asm__ ("movl %2, %1\n\t"
4             "addl %2, %1\n\t" /* adds %2 to %1 and stores the result in %1 */
5             "movl %1, %0\n\t"
6             : "=a" (x)
7             : "b" (y), "c" (z));
8
9     cout << "x=" << x << endl;
10    cout << "y=" << y << endl;
11    cout << "z=" << z << endl;
12 }
```

**Solution:**

x=40

y=10  
z=20

9. (14 points) The Butterworth filter will decrease the magnitude of a signal according to the formula below; where  $\omega$  is the original frequency,  $\omega_{cutoff}$  is the cutoff frequency,  $n$  is the order of the filter, and  $\theta$  is the attenuated signal (as a percentage of the original).

$$\frac{1}{\sqrt{\left(\frac{\omega}{\omega_{cutoff}}\right)^{2n} + 1}} = \theta \quad (1)$$

Suppose that we have a signal with a frequency of  $x$  and noise with a frequency of  $5x$ . Compute the cutoff frequency and the *minimal* number of stages of the filter such that, after filtering, more than 90% of the signal is left and at most 10% of the noise is left? Note that your cutoff frequency will be in terms of  $x$ .

**Solution:**

We solve for  $\omega_{cutoff}$  in equation (1):

$$\sqrt{\left(\frac{\omega}{\omega_{cutoff}}\right)^{2n} + 1} = \theta^{-1} \quad (2)$$

$$\left(\frac{\omega}{\omega_{cutoff}}\right)^{2n} = \theta^{-2} - 1 \quad (3)$$

$$\left(\frac{\omega}{\omega_{cutoff}}\right) = (\theta^{-2} - 1)^{\frac{1}{2n}} \quad (4)$$

$$\omega_{cutoff} = \frac{\omega}{(\theta^{-2} - 1)^{\frac{1}{2n}}} \quad (5)$$

Using equation (5), we can then plug in our target attenuations and original frequencies.

Order	Signal	Noise
1	$\omega_{cutoff} > \frac{x}{(.9^{-2}-1)^{\frac{1}{2}}} \approx 2.065x$	$\omega_{cutoff} < \frac{5x}{(.1^{-2}-1)^{\frac{1}{2}}} \approx .503x$
2	$\omega_{cutoff} > \frac{x}{(.9^{-2}-1)^{\frac{1}{4}}} \approx 1.437x$	$\omega_{cutoff} < \frac{5x}{(.1^{-2}-1)^{\frac{1}{4}}} \approx 1.585x$

Thus a 2nd order filter with a cutoff frequency between  $1.437x\text{Hz}$  and  $1.585x\text{Hz}$  will meet our requirements.