CPE355 - Real time embedded kernels - Spring'11

Prof. Nuno Alves (nalves@wne.edu), College of Engineering

Midterm #1

Date: 11am, Thursday, March 1st 2012

Question #1 - Pointers (30 points)

For the following declarations and statements,

```
// declarations
int i,j,k[10]={2,4,6,8,10,12,14,16,18,20};
int *ip;
char a,b,c[10]={11,12,13,14,15,16,17,18,19,20};
char *cp;
void *vp;

struct stst
{
    int i;
    char *p;
} x;

// assignment statements
ip  = &k[4];
vp  = ip;
cp  = &c[0];
x.p = cp+3;
x.i = 27;
```

a) Evaluate the following expressions (if the expression is an error, indicate that with "X"):

- `*(ip+2) + c[3] ;` _unknown, but code compiles___
- `(*cp +1) ;` _12_____
- `*(ip) + 6 ;` _16_____
- `vp = &x ; ((st*)vp)->p ;` _X_____
- `*(ip-2) ;` _6_____

b) Using a pointer `struct stst *z;` how would you update the member **i** of the structure **x**?

`z=&x ; (*x).i = <somevalue>`

Question #2 - Logical operators (20 points)

Evaluate the following C expressions:

```
#define A 0x33
#define B 0x20
#define C 0xB7

unsigned char a,b,c,d,e;

a = (A|B) & C;
b = ~(A & B);
c = A^C;
d = B<<3;
e = C | B | ~A;
```

Give answers in binary:

- a _____
- b _____
- c _____
- d _____
- e _____

```
a = 0x33 = 0000000000110011
b = 0xdf = 0000000011011111
c = 0x84 = 0000000010000100
d = 0x0 = 0000000000000000
e = 0xff = 0000000011111111
```

Question #3 - Typical interview questions (30 points)

Write very brief answers to the following questions:

- What is does it mean for a signal to have 30% duty cycle?

It means that 30% of the time the signal is at a high-state.

- What is a universal gate? Give one example of a universal gate?

It's a type of gate than can be combined with itself to generate any logic function. Both NAND and NOR gates are universal.

- What is the difference between a latch and a flip-flop?

In a latch, an input gets propagated to an output when the clock is either high or low. In a FF, the input only gets propagated to an output at a clock edge (either rising, falling or both).

- What is an watchdog timer and how does it work?

Its a computer hardware or software timer. If it is not restarted then the system will reboot. This is to prevent system hangs.

- In Arduino, give an example of a non-maskable interrupt.

Maskable interrupt means you can't disable it. The reset button is a good example.

- What is the difference between an interrupt request (IRQ) and interrupt service routine (ISR)?

ISR is the code that will be executed whenever an IRQ is triggered.

- Assuming we are working with 8 bit quantities, convert -15_{10} into base-2 using 2's complement.

11110001_2

- Convert 25_{10} into base-16.

0x19

• What is a decoupling capacitor? When shall we use it?

It's a capacitor we add to the circuit to prevent sudden voltage drops.

• How does the microprocessor know where to find the ISR?

Depends on the architecture. Recent microprocessors normally have the interrupt vector table at a well defined memory location. The interrupt vector table contains the addresses where the interrupt service routines are located.

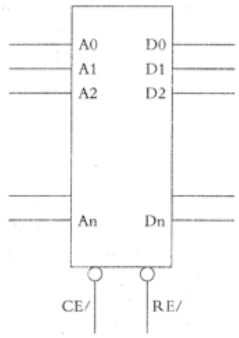
• What is an atomic section of the code?

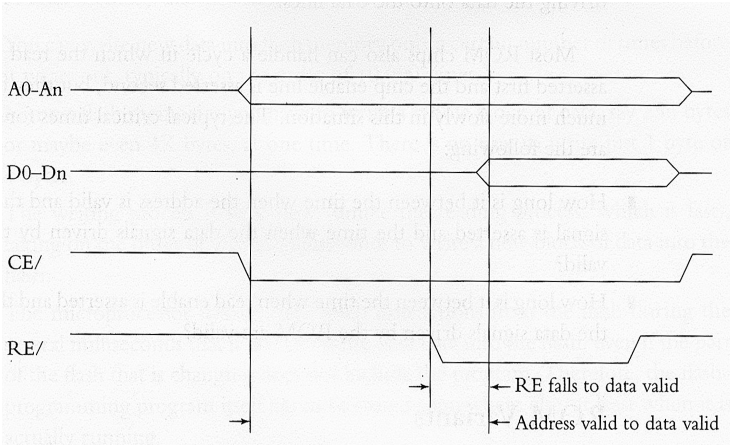
It's a section where interrupts cannot happen.

• When shall you use polling instead of an interrupt?

When we want to use a simpler code implementation in which we don't to deal with shared data problems. Polling is normally a bad idea.

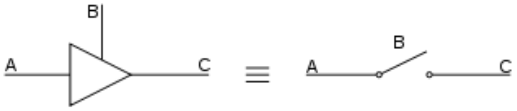
• Draw the timing diagram that reads data from the following ROM cell.





• Write the truth table for the following component. What is this component called?

INPUT		OUTPUT
A	B	C
0	1	0
1		1
X	0	Z



This is a tri-state buffer.

Last updated on February 28, 2012 1:30 PM

• When interrupts are enabled what is the problem with this code?

```
static int iTemperatures[2];

void interrupt vReadTemperatures  (void)
{
  iTemperatures[0] = !! read data from hardware
  iTemperatures[1] = !! read data from hardware
}

void main (void) {
  int iTemp0, iTemp1;
  while (TRUE) {
      iTemp0 = iTemperatures[0];
      iTemp1 = iTemperatures[1];
      if (iTemp0 != iTemp1){ !! Set off howling alarm }
  }
}
```

If an interrupt happens right after the line

"iTemp0 = iTemperatures[0];"

Then we may have a different value for iTemp0 and iTemp1. This is the most basic behavior for a shared data problem.

Question #4 - Interrupt latency (10 points)

• We have a system with 2 interrupts (intHIGH and intLOW) and 2 different task codes (taskA and taskB).

• When taskA or taskB are running, interrupts are disabled.

• The interrupt intHIGH has the highest priority.

• It takes taskA 125µsec to run, while taskB takes 250µsec.

• The interrupt intHIGH takes 300µsec to run, while intLOW requires 150µsec.

• When intHIGH is triggered, it needs finish executing its associated routine within 650µsec.

a) Assuming that interrupt nesting is allowed, is it possible to implement this system?

b) Assuming that interrupt nesting is not allowed, is it possible to implement this system?

a) Yes. The worst case interrupt latency happens when taskB just started running and intHIGH is triggered. So taskB needs to finish first (250µsec) and then intHIGH will run (300µsec). The total execution time is 550µsec which is well below the required 650µsec.
b) Yes. Nothing really changes from part a). Interrupt nesting is just the ability of an interrupt to happen inside an interrupt.

Question #5 - Software architectures (10 points)

Complete the following table:

	Priorities available	**Worst time response for task code**	**Stability of response when code changes**	**Simplicity**
Round-robin	None.	Sum of all task code.	Poor.	Very simple.
Round-robin with interrupts	Interrupt routines in priority order, then all task code with the same priority.	Total of execution time for all task code (plus execution time for interrupt routines).	Good for interrupt routines. Poor for task code.	Must deal with shared data problems between interrupt routines and task code.
Function queue-scheduling	Interrupt routines in priority order, then all task code in priority order.	Execution time for the longest function (plus execution time for interrupt routines).	Relatively good.	Must deal with shared data and must write function queue code.
Real time operating system	Interrupt routines in priority order, then all task code in priority order.	Zero (plus execution time for interrupt routines).	Very good.	Most complex (although much of the complexity is in the operating system itself).