CS 61C Summer 2019

I/O, RAID, ECC

Discussion 14: August 12, 2019

1 Polling & Interrupts

1.1 Fill out this table that compares polling and interrupts.

Operation	**Definition**	**Pro/Good for**	**Con**
Polling			
Interrupts			

2 Memory Mapped I/O

2.1 For this question, the following addresses correspond to registers in some I/O devices and not regular user memory.

- `0xFFFF0000`—Receiver Control: LSB is the ready bit (in the context of polling), there may be other bits set that we dont need right now.
- `0xFFFF0004`—Receiver Data: Received data stored at lowest byte.
- `0xFFFF0008`—Transmitter Control: LSB is the ready bit (in the context of polling), there may be other bit set that we dont need right now.
- `0xFFFF000C`—Transmitter Data: Transmitted data stored at lowest byte.

Recall that receiver will only have data for us when the corresponding ready bit is 1, and that we can only write data to the transmitter when its ready bit is 1.

Write RISC-V code that reads byte from the receiver (busy-waiting if necessary) and writes that byte to the transmitter (busy-waiting if necessary).

3 RAID

3.1 Fill out the following table:

	Configuration	**Pro/Good for**	**Con/Bad for**
RAID 0			
RAID 1			
RAID 2			
RAID 3			
RAID 4			
RAID 5			

4 Hamming ECC

Recall the basic structure of a Hamming code. We start out with some bitstring, and then add parity bits at the indices that are powers of two (1, 2, 8, etc.). We

don't assign values to these parity bits yet. **Note that the indexing convention used for Hamming ECC is different from what you are familiar with.** In particular, the 1 index represents the MSB, and we index from left-to-right. The ith parity bit $P\{i\}$ covers the bits in the new bitstring where the *index* of the bit under the aforementioned convention, j, has a 1 at the same position as i when represented as binary. For instance, 4 is `0b100` in binary. The integers j that have a 1 in the same position when represented in binary are 4, 5, 6, 7, 12, 13, etc. Therefore, $P4$ covers the bits at indices 4, 5, 6, 7, 12, 13, etc. A visual representation of this is:

Bit position		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	
Encoded data bits		p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8	d9	d10	d11	p16	d12	d13	d14	d15	
Parity bit coverage	p1	✗		✗		✗		✗		✗		✗		✗		✗		✗		✗		
	p2		✗	✗			✗	✗			✗	✗			✗	✗			✗	✗		...
	p4				✗	✗	✗	✗					✗	✗	✗	✗					✗	
	p8								✗	✗	✗	✗	✗	✗	✗	✗						
	p16																✗	✗	✗	✗	✗	

Source: `https://en.wikipedia.org/wiki/Hamming_code`

4.1 How many bits do we need to add to 0011_2 to allow single error correction?

4.2 Which locations in 0011_2 would parity bits be included?

4.3 Which bits does each parity bit cover in 0011_2?

4.4 Write the completed coded representation for 0011_2 to enable single error correction. Assume that we set the parity bits so that the bits they cover have even parity.

4.5 How can we enable an additional double error detection on top of this?

4.6 Find the original bits given the following SEC Hamming Code: 0110111_2. Again, assume that the parity bits are set so that the bits they cover have even parity.

4.7 Find the original bits given the following SEC Hamming Code: 1001000_2