Hamming Codes
Recall the basic structure of a Hamming code. Given bits 1...,m, the bit at position 2^n (starting at n=0, the first bit) is parity for all the bits with a 1 in position n. For example, the first bit is chosen such that the sum of all odd-numbered bits is even.

1. Suppose you had the bits 0011 and we want to add some bits to allow single error correction.
a) How many bits do we have to add? **3**

b) Which bits are parity bits?
Parity bits are at positions labeled with x: **XX0X011**

c) Which bits does each parity bit cover?
bit 1 1,3,5,7
bit 2 2,3 6,7
bit 4 4,5,6,7

d) Write the completed coded representation for this bit string.
1000011
e) What do we need to do to make this into a SEC-DED code?
Add another parity bit over the whole sequence.

2. Find the original bits given the following SEC Hamming Code.
a) 0110111
Group 1 error
Group 2 ok
Group 4 error
The fifth bit should be a 0. 1011

b) 1001000
Group 1 error
Group 2 ok
Group 4 error
Fifth bit should be a 1. 0100

3. If we didn't need SEC but wanted SED, how bits would we need to add for 4 bits? For 16 bits?

1, 1

RAID

Big disks are expensive (and dangerous). We can use an array of smaller disks to simulate the behavior of one larger disk with a more reasonable cost.

RAID 0	Data striping
RAID 1	Disk mirroring
RAID 2	Bit-striping with ECC disks
RAID 3	Byte–striping with single-parity disk
RAID 4	Block-striping with dedicated parity disk
RAID 5	Block-striping with interleaved parity
RAID 6	Block-striping with two interleaved parity disks for DEC

Memory Mapped I/O

Certain memory addresses correspond to registers in I/O devices and not normal memory.

Control Register: Indicates if it is okay to read/write data register

Data Register: Contains I/O data

Register	Location	Contains
Receiver Control	0xffff0000	Lowest bit is Ready Bit
Receiver Data	0xffff0004	Received data stored at lowest byte
Transmitter Control	0xffff0008	Lowest bit is Ready Bit
Transmitter Data	0xffff000c	Transmitted data stored at lowest byte

Write MIPS code to read a byte from the receiver and immediately send it to the transmitter.

```
        lui $t0 0xffff
receive_wait:
        lw $t1 0($t0)
        andi $t1 $t1 1
        beq $t1 $0 receive
        lb $t2 4($t0)
transmit_wait:
        lw $t1 8($t0)
        andi $t1 $t1 1
        beq $t1 $0 transmit_wait
        sb $t2 12($t0)
```

Polling and Interrupts

Operation	Definition	Pro/Con	Good For
Polling	Pretty much the above; forces hardware to wait on ready bits (alternatively, if timing of device is known – the ready bit can be polled at the frequency of the device). It basically means manually checking the ready bit	PRO: -easy to write -poll handler does not have excessively high overhead -deterministic -doesn't require additional hardware CON: -unfeasable on hardware with fast transfer rates that is actually rarely ready	-Slow devices: Mouse, Keyboard
Interrupt	Hardware fires an "exception" when it becomes ready. CPU changes $PC to execute code in the interrupt handler when this occurs.	PRO: -Necessary for fast devices that are rarely CON: -nondeterministic when interrupt occurs -interrupt handler has some overhead (saves all registers), meaning polling can actually be faster for slow, often ready devices such as mice ready.	Fast devices: Hard drives Network cards