Discussion 13: I/O, ECC/Parity, RAID

I/O

1. Fill this table of polling and interrupts.

Operation	Definition	Pro/Good for	Con
Polling	Forces the hardware to wait on ready bit (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 regularly.	PRO: -easy to write -poll handler does not have excessively high overhead -deterministic -doesn't require additional hardware Good for: -Mouse, keyboard	Infeasible on hardware with fast transfer rates that is actually rarely ready (e.g. Ethernet card receiver)
Interrupts	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 ready. Good for: Fast devices - Hard drives, Network cards	-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

2. Memory Mapped I/O
   Certain memory addresses correspond to registers in I/O devices and not normal memory.
   0xFFFF0000 – Receiver Control:
   Lowest two bits are interrupt enable bit and ready bit.
   0xFFFF0004 – Receiver Data:
   Received data stored at lowest byte.
   0xFFFF0008 – Transmitter Control
   Lowest two bits are interrupt enable bit and ready bit.
   0xFFFF000C – Transmitter Data
   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: #poll on ready of receiver
        lw $t1 0($t0)
        andi $t1 $t1 1
        beq $t1 $zero receive_wait
        lb $t2 4$t0) #load data
```

```
transmit_wait: #poll on ready of transmitter
        lw $t1 8($t0)
        andi $t1 $t1 1
        beq $t1 $zero transmit_wait
        #write to transmitter
        sb $t2 12($t0)
```

Hamming ECC

Recall the basic structure of a Hamming code. Given bits 1, . . . , m, the bit at position 2n 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.

Bit	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Data	P1	P2	D1	P4	D2	D3	D4	P8	D5	D6	D7	D8	D9	D10	D11
P1	X		X		X		X		X		X		X		X
P2		X	X			X	X			X	X			X	X
P4				X	X	X	X					X	X	X	X
P8								X	X	X	X	X	X	X	X

1. How many bits do we need to add to 00112 to allow single error correction?
   Parity Bits: 3
2. Which locations in 00112 would parity bits be included?
   Using P for parity bits: $PP0P011_2$
3. Which bits does each parity bit cover in 00112?
   Parity bit #1: 1, 3, 5, 7
   Parity bit #2: 2, 3, 6, 7
   Parity bit #3: 4, 5, 6, 7
4. Write the completed coded representation for 00112 to enable single error correction.
   1000011_2
5. How can we enable an additional double error detection on top of this?
   Add an additional parity bit over the entire sequence.
6. Find the original bits given the following SEC Hamming Code: 01101112
   Parity group 1: error
   Parity group 2: okay
   Parity group 4: error
   Incorrect bit: 1 + 4 = 5, change bit 5 from 1 to 0: 0110011_2
   $0110011_2 \rightarrow 1011_2$
7. Find the original bits given the following SEC Hamming Code: 10010002
   Parity group 1: error
   Parity group 2: okay
   Parity group 4: error
   Incorrect bit: 1 + 4 = 5, change bit 5 from 1 to 0: 1001100_2
   $1001100_2 \rightarrow 0100_2$
8. Find the original bits given the following SEC Hamming Code: 0100110100001102
   Parity group 1: okay
   Parity group 2: error
   Parity group 4: okay
   Parity group 8: error
   Incorrect bit: 2 + 8 = 10, change bit 10 from 0 to 1: 010011010100110_2
   $010011010100110_2 \rightarrow 01100100110_2$

RAID

Fill out the following table:

	Configuration	Pro / Good for…	Con / Bad for…
RAID 0	Data disks without check information	No overhead Fast read / write	Reliability
RAID 1	Mirrored Disks: Extra copy of disks	Fast read / write Fast recovery	High overhead → Expensive
RAID 2	Hamming ECC: One check disk per parity group	Smaller overhead	Redundant check disks
RAID 3	Single check disk for error correction (Disk controllers can detect failures) Transfer units are spread over all disks in a group (bit interleaving)	Smallest check information overhead	Read all data disks for small accesses to detect errors
RAID 4	Transfer units = a sector within a single disk Errors are detected within a single transfer unit Independent reads/writes per disks	Higher throughput of small reads	Still slow small writes (A single check disk is a bottleneck)
RAID 5	Check information is distributed across all disks in a group.	Higher throughput of small writes	

Small accesses = an access to a single disk in a group