CS 61C: Great Ideas in Computer Architecture (Machine Structures)

Dependability

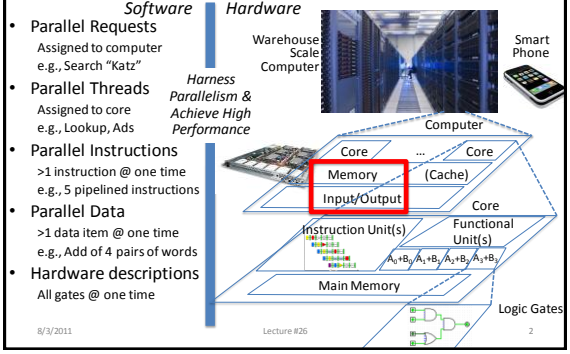
Instructor:
Michael Greenbaum

New-School Machine Structures (It's a bit more complicated!)

- Parallel Requests
  Assigned to computer
  e.g., Search "Katz"
- Parallel Threads
  Assigned to core
  e.g., Lookup, Ads
- Parallel Instructions
  >1 instruction @ one time
  e.g., 5 pipelined instructions
- Parallel Data
  >1 data item @ one time
  e.g., Add of 4 pairs of words
- Hardware descriptions
  All gates @ one time

Software / *Hardware*

Harness Parallelism & Achieve High Performance



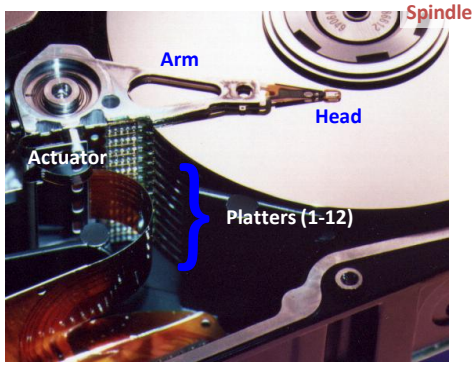
Agenda

- Disks
- Reliability
- Administrivia
- RAID
- Error Correcting Codes (If time)

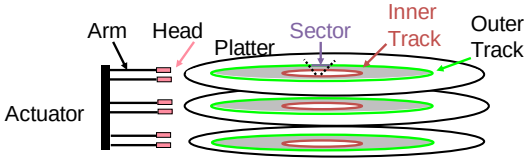
Magnetic Disks

- A kind of memory
  - Information stored by magnetizing ferrite material on surface of rotating disk
- Nonvolatile storage
  - retains its value without applying power to disk.
  - Main memory is volatile - only stores data when power is applied.
- Two Types
  - Floppy disks – slower, less dense, removable, non-existent today
  - Hard Disk Drives (HDD) – faster, more dense, non-removable.
- Purpose in computer systems (Hard Drive):
  - Long-term, inexpensive storage for files.
  - Layer in the memory hierarchy beyond main memory.

Photo of Disk Head, Arm, Actuator

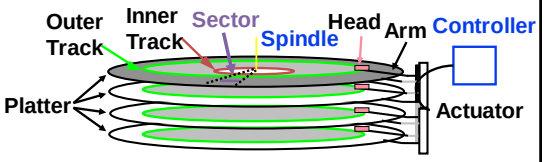


Disk Device Terminology



- Several platters, with information recorded magnetically on both surfaces (usually)
- Bits recorded in tracks, which in turn divided into sectors (e.g., 512 Bytes)
- Actuator moves head (end of arm) over track ("seek"), wait for sector rotate under head, then read or write

Disk Device Performance (1/2)



- Disk Latency = Seek Time + Rotation Time + Transfer Time + Controller Overhead
  - Seek Time? depends on no. tracks to move arm, speed of actuator
  - Rotation Time? depends on speed disk rotates, how far sector is from head
  - Transfer Time? depends on data rate (bandwidth) of disk (f(bit density,rpm)), size of request

Disk Device Performance (2/2)

- Average distance of sector from head?
- 1/2 time of a rotation
  - 7200 Revolutions Per Minute ⇒ 120 Rev/sec
  - 1 revolution = 1/120 sec ⇒ 8.33 milliseconds
  - 1/2 rotation (revolution) ⇒ 4.17 ms
- Average no. tracks to move arm?
  - Disk industry standard benchmark:
    - Sum all time for all possible seek distances from all possible tracks / # possible
    - Assumes average seek distance is random
- Size of Disk cache can strongly affect perf!
  - Cache built into disk system, OS knows nothing

Disk Drive Performance Example

- 7200 RPM drive, 4 ms seek time, 20 MB/sec transfer rate. Negligible controller overhead. Latency to read 100 KB file?
- Rotation time = 4.17 ms (from last slide)
- Transfer time = .1 MB / 20 (MB/sec) = 5 ms
- Latency = 4 + 4.17 + 5 = 13.17 ms.
- Throughput = 100 KB / 13.17 ms = 7.59 MB / sec
- How do numbers change when reading bigger/smaller file? File fragmented across multiple locations?

Where does Flash memory come in?

- Microdrives and Flash memory (e.g., CompactFlash) are going head-to-head
  - Both non-volatile (no power, data ok)
  - Flash benefits: durable & lower power (no moving parts, need to spin µdrives up/down)
  - Flash limitations: finite number of write cycles (wear on the insulating oxide layer around the charge storage mechanism). Most ≥ 100K, some ≥ 1M W/erase cycles.
- How does Flash memory work?
  - NMOS transistor with an additional conductor between gate and source/drain which "traps" electrons. The presence/absence is a 1 or 0.

`en.wikipedia.org/wiki/Flash_memory`

What does Apple put in its iPods?

Agenda

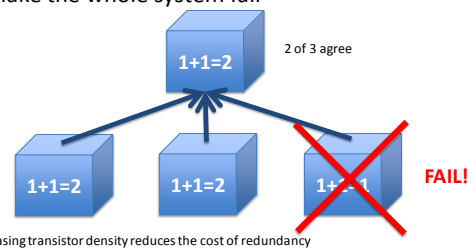
- Disks
- Reliability
- Administrivia
- RAID
- Error Correcting Codes (If time)

Review - 6 Great Ideas in Computer Architecture

1. Layers of Representation/Interpretation
2. Moore's Law
3. Principle of Locality/Memory Hierarchy
4. Parallelism
5. Performance Measurement & Improvement
6. **Dependability via Redundancy**

Great Idea #6: Dependability via Redundancy

- Redundancy so that a failing piece doesn't make the whole system fail



Increasing transistor density reduces the cost of redundancy

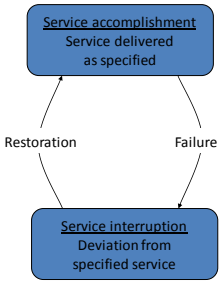
Great Idea #6: Dependability via Redundancy

- Applies to everything from datacenters to memory
  - Redundant datacenters so that can lose 1 datacenter but Internet service stays online
  - Redundant routes so can lose nodes but Internet doesn't fail
  - Redundant disks so that can lose 1 disk but not lose data (Redundant Arrays of Independent Disks/RAID)
  - Redundant memory bits of so that can lose 1 bit but no data (Error Correcting Code/ECC Memory)

Explain a lot of buzzwords

- Parity (even, odd)
- ECC
- SEC/DED
- Hamming Distance
- MTTF
- MTBF
- MTTR
- Nines of availability
- Hot Spares

Dependability



- Fault: failure of a component
  - May or may not lead to system failure

Dependability Measures

- Reliability: Mean Time To Failure (MTTF)
- Service interruption: Mean Time To Repair (MTTR)
- Mean time between failures (MTBF)
  - MTBF = MTTF + MTTR
- Availability = MTTF / (MTTF + MTTR)
- Improving Availability
  - Increase MTTF: More reliable hardware/software + Fault Tolerance
  - Reduce MTTR: improved tools and processes for diagnosis and repair

Reliability Measures

- MTTF, MTBF usually measured in hours
  - E.g., average MTTF is 100,000 hours
- Another measure is average number of failures per year
  - E.g., 1000 disks with 100,000 hour MTTF
  - 365 days * 24 hours == 8760 hours
  - (1000 disks * 8760 hrs/year) / 100,000 = 87.6 failed disks per year on average
  - 87.6/1000 = 8.76% annual failure rate

Availability Measures

- Availability = MTTF / (MTTF + MTTR) as %
- Since hope rarely down, shorthand is "number of 9s of availability per year"
- 1 nine: 90% => 36 days of repair/year
- 2 nines: 99% => 3.6 days of repair/year
- 3 nines: 99.9% => 526 minutes of repair/year
- 4 nines: 99.99% => 53 minutes of repair/year
- 5 nines: 99.999% => 5 minutes of repair/year

Dependability Design Principle

- Design Principle: No single points of failure
- "Chain is only as strong as its weakest link"
- Dependability Corollary of Amdahl's Law
  - Doesn't matter how dependable you make one portion of system
  - Dependability limited by part you do not improve

Administrivia

- No lab Thursday - Work on project, study for final, etc. TAs will be there.
- Sign up for Project 3 face-to-face grading at: http://inst.eecs.berkeley.edu/~cs61c/su11/signups/signup.cgi
  - Scheduled for Wednesday, 8/10 in 200 SD
- HKN will be conducting course survey Tuesday, 8/9.

Cs61c in … Social Media

- Are C's null-terminated strings the costliest mistake in CS history?
- Performance costs: "If the source string is NUL terminated, however, attempting to access it in units larger than bytes risks attempting to read characters after the NUL. "
- Security costs: "Despite 15 years of attention, over- and under-running string buffers is still a preferred attack vector for criminals"
- What are the pros/cons of storing a String as a length/data vector instead?

http://queue.acm.org/detail.cfm?id=2010365

Agenda

- Disks
- Reliability
- Administrivia
- RAID
- Error Correcting Codes (If time)

Evolution of the Disk Drive

IBM 3390K, 1986

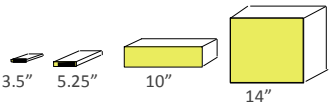
IBM RAMAC 305, 1956

Apple SCSI, 1986

Arrays of Small Disks

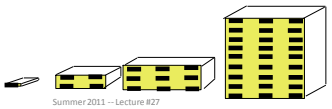
Can smaller disks be used to close gap in performance between disks and CPUs?

Conventional:
4 disk types



Low End → High End

Disk Array:
1 disk type



Replace Small Number of Large Disks with Large Number of Small Disks! (1988 Disks)

	IBM 3390K	IBM 3.5" 0061	x70	
Capacity	20 GBytes	320 MBytes	23 GBytes	
Volume	97 cu. ft.	0.1 cu. ft.	11 cu. ft.	9X
Power	3 KW	11 W	1 KW	3X
Data Rate	15 MB/s	1.5 MB/s	120 MB/s	8X
I/O Rate	600 I/Os/s	55 I/Os/s	3900 IOs/s	6X
MTTF	250 KHrs	50 KHrs	??? Hrs	
Cost	$250K	$2K	$150K	

Disk Arrays have potential for large data and I/O rates, high MB per cu. ft., high MB per KW, but what about reliability?

Replace Small Number of Large Disks with Large Number of Small Disks! (1988 Disks)

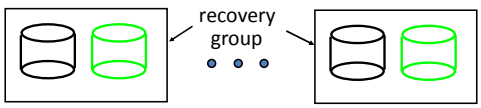
	IBM 3390K	IBM 3.5" 0061	x70	
Capacity	20 GBytes	320 MBytes	23 GBytes	
Volume	97 cu. ft.	0.1 cu. ft.	11 cu. ft.	9X
Power	3 KW	11 W	1 KW	3X
Data Rate	15 MB/s	1.5 MB/s	120 MB/s	8X
I/O Rate	600 I/Os/s	55 I/Os/s	3900 IOs/s	6X
MTTF	250 KHrs	50 KHrs	~700 Hrs	
Cost	$250K	$2K	$150K	

Disk Arrays have potential for large data and I/O rates, high MB per cu. ft., high MB per KW, but what about reliability?

RAID: Redundant Arrays of (Inexpensive) Disks

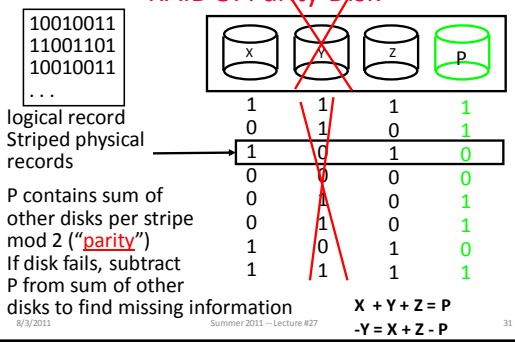
- Files are "striped" across multiple disks
- Redundancy yields high data availability
  - Availability: service still provided to user, even if some components failed
- Disks will still fail
- Contents reconstructed from data redundantly stored in the array
  - Capacity penalty to store redundant info
  - Bandwidth penalty to update redundant info

Redundant Arrays of Inexpensive Disks RAID 1: Disk Mirroring/Shadowing

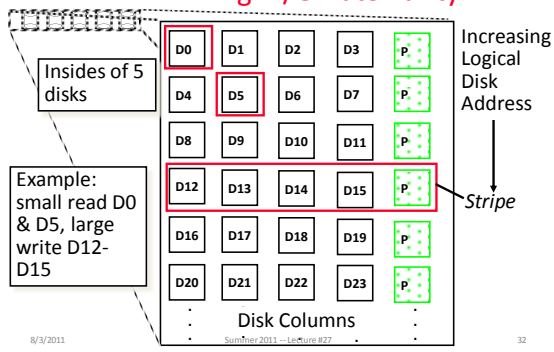


- Each disk is fully duplicated onto its “mirror”
  Very high availability can be achieved
- Bandwidth sacrifice on write:
  Logical write = two physical writes
  Reads may be optimized
- Most expensive solution: 100% capacity overhead

Redundant Array of Inexpensive Disks RAID 3: Parity Disk

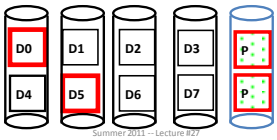


Redundant Arrays of Inexpensive Disks RAID 4: High I/O Rate Parity

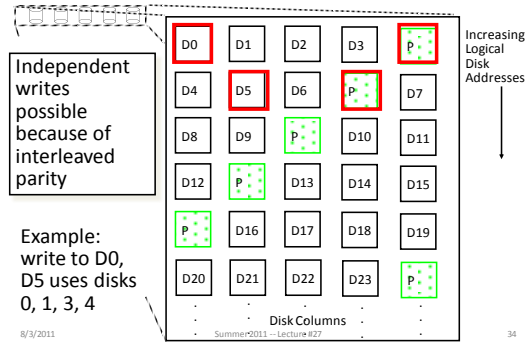


Inspiration for RAID 5

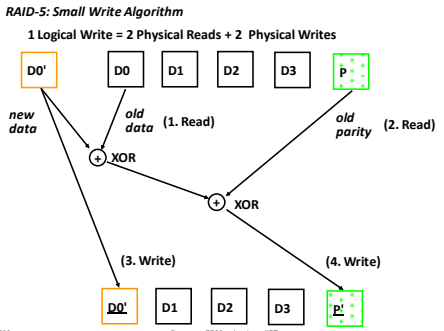
- When writing to a disk, need to update Parity Disk:
  - Option 1: read other data disks, create new sum and write to Parity Disk
  - Option 2: since P has old sum, compare old data to new data, add the difference to P
- Small writes are bottlenecked by Parity Disk: Write to D0, D5 both also write to P disk



RAID 5: High I/O Rate Interleaved Parity



Problems of Disk Arrays: Small Writes



Tech Report Read 'Round the World (December 1987)

A Case for Redundant Arrays of Inexpensive Disks (RAID)

David A. Patterson, Garth Gibson, and Randy H. Katz

Google scholar Case for Raid Search Advanced Scholar Search

Scholar Articles and patents anytime include citations Create email alert

A case for redundant arrays of inexpensive disks (RAID)
DA Patterson, G Gibson… - Proceedings of the 1988 ACM …, 1988 - portal.acm.org
Abstract Increasmg performance of CPUs and memorres wrll be squandered If not matched by a sunrlm peformance ourease m II0 Whde the capactty of Smgle Large Expenstve D&T (SLED) has grown rapuily, the performance rmprovement of SLED has been modest …
Cited by 2413 - Related articles - Library Search - All 186 versions

magnetic disk technology developed for personal computers, offers an attractive alternative to SLED, promising improvements of an order of magnitude in performance, reliability, power consumption, and scalability.
This paper introduces five levels of RAIDs, giving their relative cost/performance, and compares RAIDs to an IBM 3380 and a Fujitsu Super Eagle.

RAID-I

- RAID-I (1989)
  - Consisted of a Sun 4/280 workstation with 128 MB of DRAM, four dual-string SCSI controllers, 28 5.25-inch SCSI disks and specialized disk striping software

RAID II

- 1990-1993
- Early Network Attached Storage (NAS) System running a Log Structured File System (LFS)
- Impact:
  - $25 Billion/year in 2002
  - Over $150 Billion in RAID device sold since 1990-2002
  - 200+ RAID companies (at the peak)
  - Software RAID a standard component of modern OSs

RAID II

RAID Summary

- Logical-to-physical block mapping, parity striping, read-modify-write processing
- Orchestrating data staging between network interfaces, parity hardware, and file server interfaces
- Failed disk replacement, hot spares, background copies and backup
- Embedded log-structured file systems, compression on the fly
- Software complexity dominates hardware!

Agenda

- Disks
- Reliability
- Administrivia
- RAID
- Error Correcting Codes

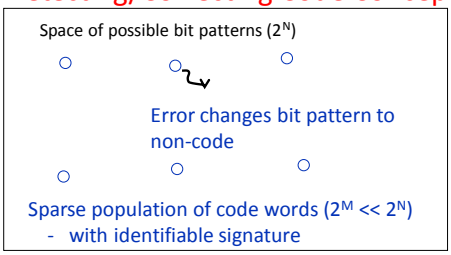
Error Detection/Correction Codes

- Memory systems generate errors (accidentally flipped-bits)
  - DRAMs store very little charge per bit
  - "Soft" errors occur occasionally when cells are struck by alpha particles or other environmental upsets
  - "Hard" errors can occur when chips permanently fail.
  - Problem gets worse as memories get denser and larger

Error Detection/Correction Codes

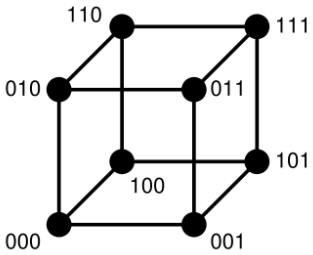
- Memories protected against failures with EDC/ECC
- Extra bits are added to each data-word
  - Extra bits are a function of the data
  - In general, each possible data word value is mapped to a special "*code word*". A fault changes a valid code word to an invalid one - which can be detected.

Detecting/Correcting Code Concept



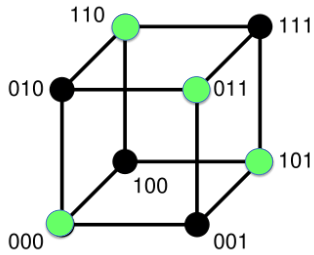
- Detection: bit pattern fails codeword check
- Correction: map to nearest valid code word

3 Bit Example



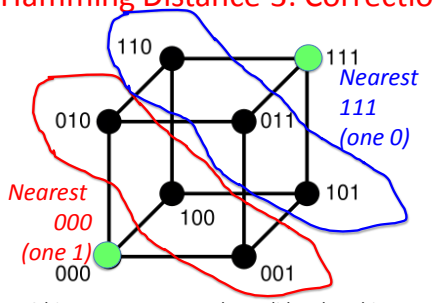
Hamming Distance: Number of bits that differ between two values.

Hamming Distance 2: Detection



- No 1 bit error goes to another valid code
- ½ codes are valid

Hamming Distance 3: Correction



- No 2 bit error goes to another valid code; 1 bit error near
- 1/4 codes are valid

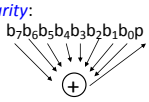
Block Code Principles

- Hamming distance = difference in # of bits
- p = 011011, q = 001111, Ham. distance (p,q) = 2
- p = 011011,
  q = 110001,
  distance (p,q) = ?
- Can think of extra bits as creating a code with the data
- What if minimum distance between members of code is 2 and get a 1 bit error?

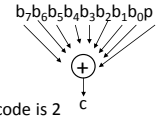
Richard Hamming, 1915-98
Turing Award Winner[48]

Parity: Simple Error Detection Coding

- Each data value, before it is written to memory is "tagged" with an extra bit to force the stored word to have *even parity*:



- Each word, as it is read from memory is "checked" by finding its parity (including the parity bit).



- Minimum Hamming distance of parity code is 2
- A non-zero parity indicates an error occurred:
  - 2 errors (on different bits) is not detected (nor any even number of errors)
  - odd numbers of errors are detected.

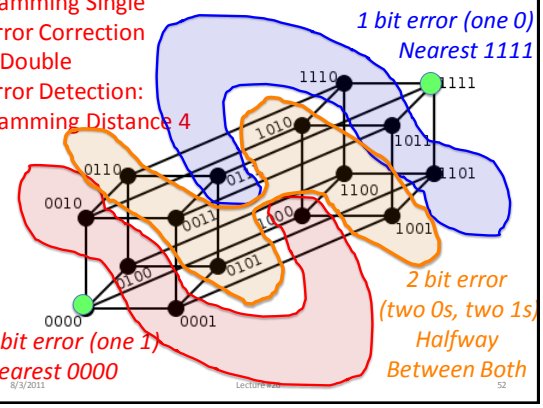
Parity Example

- Data 0101 0101
- 4 ones, even parity now
- Write to memory: 0101 0101 0 to keep parity even
- Data 0101 0111
- 5 ones, odd parity now
- Write to memory: 0101 0111 1 to make parity even
- Read from memory 0101 0101 0
- 4 ones => even parity, so no error
- Read from memory 1101 0101 0
- 5 ones => odd parity, so error
- What if error in parity bit?

Suppose want to Correct 1 Error?

- Can we correct if minimum distance is 2?
- What must minimum Hamming Distance be?
- Richard Hamming came up with simple to understand mapping to allow Error Correction at minimum distance of 3
- Called Hamming ECC for Error Correction Code
- We won't go over the details during lecture, check bonus slides at end.

Hamming Single Error Correction + Double Error Detection: Hamming Distance 4



"In Conclusion.."

- Disks - Work by positioning head over spinning platters. Very slow relative to CPU.
- Great Idea: Redundancy to Get Dependability
- Reliability: MTTF & Annual failure rate
- Availability: % uptime (MTTF-MTTR/MTTF)
- RAID: Redundant Arrays of Inexpensive Disks
- Memory:
  - Hamming distance 2 Parity for Single Error Detect
  - Hamming distance 3 Single Error Correction Code + encode bit position of error
  - Hamming distance 4 SEC/Double Error Detection

Additional Slides

- You are NOT responsible for knowing the material on the following slides.
- They are here to provide more detail on some of the topics we've covered.

Hamming Error Correction Code

- Use of extra parity bits to allow the position identification of a single error

1. Mark all bit positions that are powers of 2 as parity bits. (positions 1, 2, 4, 8, 16, …)
   - Start numbering bits at 1 at left, not at 0 on right
2. All other bit positions are for the data to be encoded. (positions 3, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, …)

Hamming ECC

3. Each parity bit calculates the parity for some of the bits in the code word

- The position of parity bit determines sequence of bits that it checks
- Bit 1 (0001_2): checks bits (1,3,5,7,9,11,...)
- Bit 2 (0010_2): checks bits (2,3,6,7,10,11,14,15,...)
- Bit 4 (0100_2): checks bits (4-7, 12-15, 20-23, ...)
- Bit 8 (1000_2): checks bits (8-15, 24-31, 40-47 ,...)

Hamming ECC

4. Set parity bits to create even parity for each group

- A byte of data: 10011010
- Create the coded word, leaving spaces for the parity bits:
- _ _ 1 _ 0 0 1 _ 1 0 1 0

  0 0 0 0 0 0 0 0 0 1 1 1

  1 2 3 4 5 6 7 8 9 0 1 2
- Calculate the parity bits

Hamming ECC

Student Roulette

- Position 1 checks bits 1,3,5,7,9,11:
  ? _ **1** _ **0** 0 **1** _ **1** 0 **1** 0. set position 1 to a _:
  _ _ **1** _ **0** 0 **1** _ **1** 0 **1** 0
- Position 2 checks bits 2,3,6,7,10,11:
  0 **?** **1** _ 0 **0** **1** _ 1 **0** **1** 0. set position 2 to a _:
  0 **_** **1** _ 0 **0** **1** _ 1 **0** **1** 0
- Position 4 checks bits 4,5,6,7,12:
  0 1 1 **?** **0** **0** **1** _ 1 0 1 **0**. set position 4 to a _:
  0 1 1 **_** **0** **0** **1** _ 1 0 1 **0**

Hamming ECC

- Position 1 checks bits 1,3,5,7,9,11:
  ? _ **1** _ **0** 0 **1** _ **1** 0 **1** 0. set position 1 to a 0:
  0 _ **1** _ **0** 0 **1** _ **1** 0 **1** 0
- Position 2 checks bits 2,3,6,7,10,11:
  0 **?** **1** _ 0 **0** **1** _ 1 **0** **1** 0. set position 2 to a 1:
  0 **1** **1** _ 0 **0** **1** _ 1 **0** **1** 0
- Position 4 checks bits 4,5,6,7,12:
  0 1 1 **?** **0** **0** **1** _ 1 0 1 **0**. set position 4 to a 1:
  0 1 1 **1** **0** **0** **1** _ 1 0 1 **0**

Hamming ECC

- Position 8 checks bits 8,9,10,11,12:
  0 1 1 1 0 0 1 **?** **1** **0** **1** **0**. set position 8 to a _:
  0 1 1 1 0 0 1 **0** **1** **_** **1** **0**
- Final code word: 011100101010
- Data word: 1 001 1010

Hamming ECC

- Position 8 checks bits 8,9,10,11,12:
  0 1 1 1 0 0 1 **? 1 0 1 0**. set position 8 to a 0:
  0 1 1 1 0 0 1 **0 1 0 1 0**
- Final code word: 011100101010.

Agenda

- Definition of Dependability, Reliability, Availability and metrics to evaluate them
- Codes for Redundancy
- Administrivia
- Error Detection in Memory (Parity)
- Error Correction in Memory: Encoding
- Technology Break
- Error Correction in Memory: Correcting

Hamming ECC

- **Finding and fixing a corrupted bit:**
- Suppose receive 011100101110
  123456789012
- Parity 1_, Parity 2_, Parity 4_, Parity 8_

Bits numbers $xxx1_{two}$, $xx1x_{two}$, $x1xx_{two}$, $1xxx_{two}$)

- Parity bits 2 and 8 incorrect. As 2 + 8 = 10, bit position 10 is location of bad bit: flip value!
- Corrected value: 011100101010
- Why does Hamming ECC work?

Hamming ECC on your own

- Test if these Hamming-code words are correct. If one is incorrect, indicate the correct code word. Also, indicate what the original data was.
- 010101100011
- 111110001100
- 000010001010

Hamming Error Correcting Code

- Overhead involved in single error correction code
- Let *p* be total number of parity bits and *d* number of data bits in *p + d* bit word
- If p error correction bits are to point to error bit (*p + d* cases) + indicate that no error exists (1 case), we need:

  $2^p >= p + d + 1$,

  thus $p >= \log(p + d + 1)$

  for large *d*, *p* approaches log(*d*)
- *8 bits data => d = 8,* $2^p = p + 8 + 1 => p = 4$
- *16 data => 5 parity, 32 data => 6 parity, 64 data => 7 parity*

Hamming Single Error Correction, Double Error Detection (SEC/DED)

- Adding extra parity bit covering the entire word provides double error detection as well as single error correction

  1 2 3 4 5 6 7 8

  p_1 p_2 d_1 p_3 d_2 d_3 d_4 p_4
- Hamming parity bits H (p_1 p_2 p_3) are computed (even parity as usual) plus the even parity over the entire word, p_4:

  H=0 p_4=0, no error

  H≠0 p_4=1, correctable single error (odd parity if 1 error => p_4=1)

  H≠0 p_4=0, double error occurred (even parity if 2 errors=> p_4=0)

  H=0 p_4=1, an error occurred in p_4 bit, not in rest of word

Typical modern codes in DRAM memory systems:
64-bit data blocks (8 bytes) with 72-bit code words (9 bytes).

What if More Than 2 Bit Errors?

- Network transmissions, disks, distributed storage common failure mode is bursts of bit errors, not just one or two bit errors
  - contiguous sequence of *B* bits in which first, last and any number of intermediate bits are in error
  - caused by impulse noise or by fading in wireless
  - effect is greater at higher data rates

Cyclic Redundancy Check

- for block of *k* bits, transmitter generates an *n-k* bit frame check sequence
- Transmits *n* bits exactly divisible by some number
- Receiver divides frame by that number
  - If no remainder, assume no error
  - Easy to calculate division for some binary numbers with shift register
- Disks detect *and correct* blocks of 512 bytes with called Reed Solomon codes ≈ CRC (see last slide)

(In More Depth: Code Types)

- Linear Codes: $\vec{C} = \vec{G} \cdot \vec{d}$ $\vec{S} = \vec{H} \cdot \vec{C}$
  Code is *generated* by G and in *null-space* of H
- Hamming Codes: Design the H matrix
  - d = 3 ⇒ Columns nonzero, Distinct
  - d = 4 ⇒ Columns nonzero, Distinct, Odd-weight
- Reed-solomon codes:
  - Based on polynomials in GF(2^k) (I.e. k-bit symbols)
  - Data as coefficients, code space as values of polynomial:
  - $P(x)=a_0+a_1x^1+\ldots a_{k-1}x^{k-1}$
  - Coded: P(0),P(1),P(2)….,P(n-1)
  - Can recover polynomial as long as get *any* k of n
  - Alternatively: as long as no more than n-k coded symbols erased, can recover data.
- Side note: Multiplication by constant in GF(2^k) can be represented by k×k matrix: a·x
  - Decompose unknown vector into k bits: $x=x_0+2x_1+\ldots+2^{k-1}x_{k-1}$
  - Each column is result of multiplying a by 2^i