CS252
Graduate Computer Architecture
Lecture 18

Error Correction

John Kubiatowicz

Electrical Engineering and Computer Sciences

University of California, Berkeley

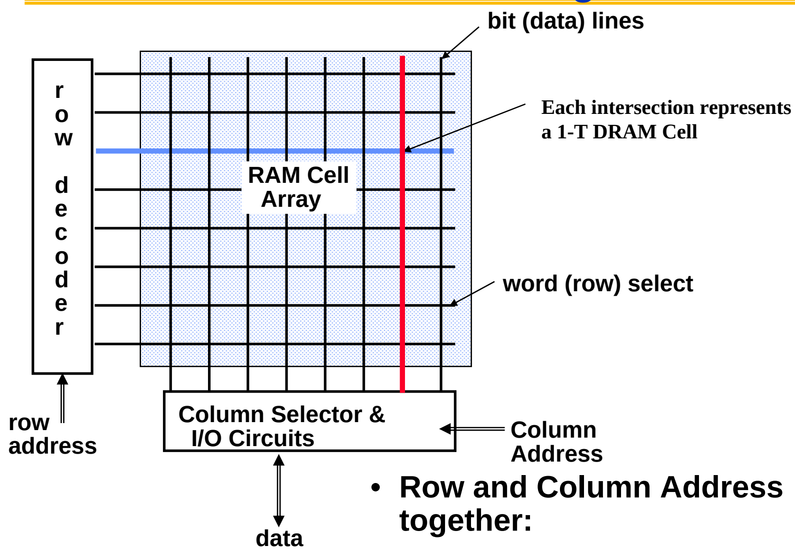
http://www.eecs.berkeley.edu/~kubitron/cs252

http://www-inst.eecs.berkeley.edu/~cs252

Review: Main Memory Background

- **Performance of Main Memory:**
  - **Latency: Cache Miss Penalty**
    - » ***Access Time*: time between request and word arrives**
    - » ***Cycle Time*: time between requests**
  - **Bandwidth: I/O & Large Block Miss Penalty (L2)**
- **Main Memory is *DRAM*: Dynamic Random Access Memory**
  - **Dynamic since needs to be refreshed periodically (8 ms, 1% time)**
  - **Addresses divided into 2 halves (Memory as a 2D matrix):**
    - » ***RAS* or *Row Address Strobe***
    - » ***CAS* or *Column Address Strobe***
- **Cache uses *SRAM*: Static Random Access Memory**
  - **No refresh (6 transistors/bit vs. 1 transistor**
    ***Size*: DRAM/SRAM - *4-8*,**
    Cost/Cycle time*: SRAM/DRAM - *8-16

Review: Classical DRAM Organization

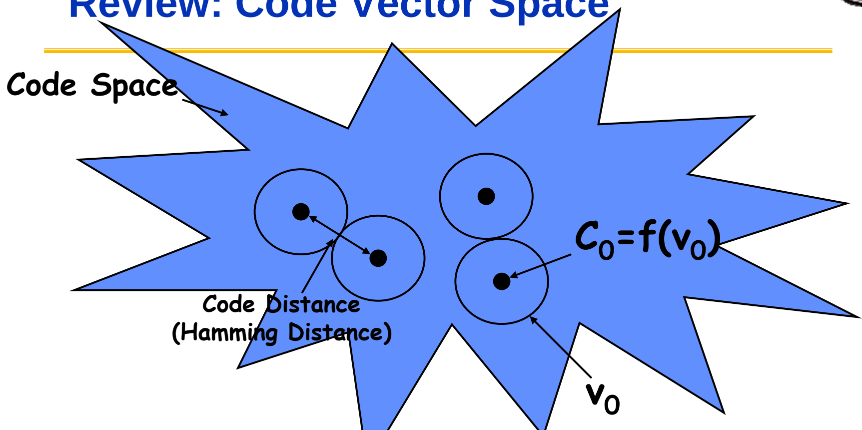


- **Row and Column Address together:**
  - **Select 1 bit a time**

Review: Need for Error Correction!

- **Motivation:**
  - **Failures/time *proportional* to number of bits!**
  - **As DRAM cells shrink, more vulnerable**
- **Went through period in which failure rate was low enough without error correction that people didn't do correction**
  - **DRAM banks too large now**
  - **Servers always corrected memory systems**
- **Basic idea: add redundancy through parity bits**
  - **Common configuration: Random error correction**
    - » **SEC-DED (single error correct, double error detect)**
    - » **One example: 64 data bits + 8 parity bits (11% overhead)**
  - **Really want to handle failures of physical components as well**
    - » **Organization is multiple DRAMs/DIMM, multiple DIMMs**
    - » **Want to recover from failed DRAM and failed DIMM!**
    - » **"Chip kill" handle failures width of single DRAM chip**

Review: Code Vector Space



- Not every vector in the code space is valid
  - If code space of size 2^n and data space of size 2^k, called an (n, k) code
- Hamming Distance (d):
  - Minimum number of bit flips to turn one code word into another
- Number of errors that we can detect: (d-1)
- Number of errors that we can fix: ½(d-1)

Galois Field Elements

- Definition: *Field:* a complete group of elements with:
  - Addition, subtraction, multiplication, division
  - Completely *closed* under these operations
  - Every element has an additive inverse
  - Every element except zero has a multiplicative inverse
- Examples:
  - Real numbers
  - Binary, called GF(2) ⇐ Galois Field with base 2
    - » Values 0, 1. Addition/subtraction: use xor. Multiplicative inverse of 1 is 1
  - Prime field, GF(p) ⇐ Galois Field with base p
    - » Values 0 … p-1
    - » Addition/subtraction/multiplication: modulo p
    - » Multiplicative Inverse: every value except 0 has inverse
    - » Example: GF(5): 1×1 ≡ 1 mod 5, 2×3 ≡ 1mod 5, 4×4 ≡ 1 mod 5
  - General Galois Field: GF(p^m) ⇐ base p (prime!), dimension m
    - » Values are vectors of elements of GF(p) of dimension m
    - » Add/subtract: vector addition/subtraction
    - » Multiply/divide: more complex
    - » Just like read numbers but finite!
    - » Common for computer algorithms: GF(2^m)

Hamming Bound, symbols in GF(2)

- Consider an (n,k) code with distance d
  - How do n, k, and d relate to one another?
- First question: How big are spheres?
  - For distance d, spheres are of radius ½ (d-1),
    - » i.e. all error with weight ½ (d-1) or less must fit within sphere
  - Thus, size of sphere is at least:
    1 + Num(1-bit err) + Num(2-bit err) + …+ Num(½(d-1) – bit err) ⇒

$$Size = \sum_{e=0}^{\frac{1}{2}(d-1)} \binom{n}{e}$$

- Hamming bound reflects bin-packing of spheres:
  - need 2^k of these spheres within code space

$$2^k \cdot \sum_{e=0}^{\frac{1}{2}(d-1)} \binom{n}{e} \le 2^n \quad \Rightarrow \quad 2^k \cdot (1+n) \le 2^n, d = 3$$

How to *Generate* code words?

- Consider a *linear* code. Need a *Generator* Matrix.
  - Let v_i be the data value (k bits), C_i be resulting code (n bits):

$$\overline{C_i} = \mathbf{G} \cdot \overline{v_i} \quad \longleftarrow \quad \text{G must be an n×k matrix}$$

- Are there 2^k unique code values?
  - Only if the k columns of G are linearly independent!
- Of course, need some way of decoding as well.

$$\overline{v_i} = f_d\left(\overline{C_i^{'}}\right)$$

  - Is this linear??? Why or why not?
- A code is *systematic* if the data is directly encoded within the code words.
  - Means Generator has form: $\mathbf{G} = \begin{pmatrix} \mathbf{I} \\ \mathbf{P} \end{pmatrix}$
  - Can always turn non-systematic code into a systematic one (row ops)

Implicitly Defining Codes by Check Matrix

- **But – what is the distance of the code? Not obvious**
- **Instead, consider a parity-check matrix H (n×[n-k])**
  - **Compute the following *syndrome* S_i given code element C_i:**

$$S_i = \mathbf{H} \cdot C_i$$

  - **Define valid code words C_i as those that give S_i=0 (null space of H)**
  - **Size of null space? (n-rank H)=k if (n-k) linearly independent columns in H**
- **Suppose you transmit code word C, and there is an error. Model this as vector E which flips selected bits of C to get R (received):**

$$\overline{R} = \overline{C} \oplus \overline{E} \longleftarrow \text{Error vector}$$

- **Consider what happens when we multiply by H:**

$$\overline{S} = \mathbf{H} \cdot \overline{R} = \mathbf{H} \cdot (\overline{C} \oplus \overline{E}) = \mathbf{H} \cdot \overline{E}$$

- **What is distance of code?**
  - ***Code has distance d if no sum of d-1 or less columns yields 0***
  - ***I.e. No error vectors, E, of weight ≤ d have zero syndromes***
  - ***Code design: Design H matrix with these properties***

How to relate G and H (Binary Codes)

- **Defining H makes it easy to understand distance of code, but hard to generate code (H defines code implicitly!)**
- **However, let H be of following form:**

$$\mathbf{H} = (\mathbf{P} \mid \mathbf{I}) \longleftarrow$$

P is (n-k)×k, I is (n-k)×(n-k)
Result: H is (n-k)×k

- **Then, G can be of following form (maximal code size):**

$$\mathbf{G} = \begin{pmatrix} \mathbf{I} \\ \mathbf{P} \end{pmatrix} \longleftarrow$$

P is (n-k)×k, I is k×k
Result: G is n×k

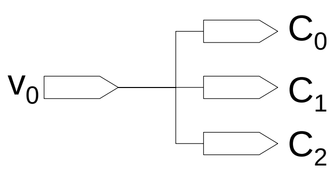
- **Notice: G generates values in null-space of H**

$$\overline{S}_i = \mathbf{H} \cdot \left(\mathbf{G} \cdot \overline{v}_i\right) = \left((\mathbf{P} \mid \mathbf{I}) \cdot \begin{pmatrix} \mathbf{I} \\ \mathbf{P} \end{pmatrix} \right) \cdot \overline{v}_i \equiv 0$$

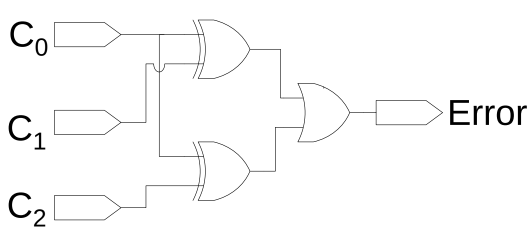
Simple example: Repetition (voting)

- **Repetition code (1-bit):**

$$\mathbf{G} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$



$$\mathbf{H} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$$

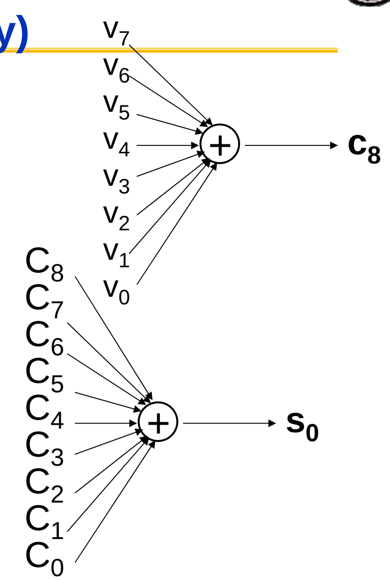


Simple example (Parity)

- **Parity code (8-bits):**

$$\mathbf{G} = \begin{pmatrix} 1&0&0&0&0&0&0&0 \\ 0&1&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0 \\ 0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&1 \\ 1&1&1&1&1&1&1&1 \end{pmatrix}$$

$$\mathbf{H} = (111111111)$$



- **Note: Complexity of logic depends on number of 1s in row!**

Example, d=4 code (SEC-DED)

- **Design H with:**
  - **All columns non-zero, odd-weight, distinct**
    - » **Note that odd-weight refers to *Hamming Weight,* i.e. number of zeros**
- **Why does this generate d=4?**
  - **Any single bit error will generate a distinct, non-zero value**
  - **Any double error will generate a distinct, non-zero value**
    - » **Why? Add together two distinct columns, get distinct result**
  - **Any triple error will generate a non-zero value**
    - » **Why? Add together three odd-weight values, get an odd-weight value**
  - **So: need four errors before indistinguishable from code word**
- **Because d=4:**
  - **Can correct 1 error (*Single Error Correction, i.e.* SEC*)***
  - **Can detect 2 errors (*Double Error Detection, i.e.* DED)**
- **Example:**
  - **Note: log size of nullspace will be (columns – rank) = 4, so:**
    - » **Rank = 4, since rows independent, 4 cols indpt**
    - » **Clearly, 8 bits in code word**
    - » **Thus: (8,4) code**

$$\begin{pmatrix} S_0 \\ S_1 \\ S_2 \\ S_3 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \\ C_4 \\ C_5 \\ C_6 \\ C_7 \end{pmatrix}$$

Tweeks:

- **No reason cannot make code shorter than required**
- **Suppose n-k=8 bits of parity. What is max code size (n) for d=4?**
  - **Maximum number of unique, odd-weight columns: 2^7 = 128**
  - **So, n = 128. But, then k = n – (n – k) = 120. Weird!**
  - **Just throw out columns of high weight and make 72, 64 code!**
- **But – shortened codes like this might have d > 4 in some special directions**
  - **Example: Kaneda paper, catches failures of groups of 4 bits**
  - **Good for catching chip failures when DRAM has groups of 4 bits**