CS252
Graduate Computer Architecture
Lecture 22

Memory Technology
Error Correction Codes
April 16th, 2012

John Kubiatowicz
Electrical Engineering and Computer Sciences
University of California, Berkeley

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

Review: Zoo of hardware primitives

- test&set (&address) { /* most architectures */
    result = M[address];
    M[address] = 1;
    return result;
  }
- swap (&address, register) { /* x86 */
    temp = M[address];
    M[address] = register;
    register = temp;
  }
- compare&swap (&address, reg1, reg2) { /* 68000 */
    if (reg1 == M[address]) {
      M[address] = reg2;
      return success;
    } else {
      return failure;
    }
  }
- load-linked&store conditional(&address) {
    /* R4000, alpha */
    loop:
      ll r1, M[address];
      movi r2, 1; /* Can do arbitrary comp */
      sc r2, M[address];
      beqz r2, loop;
  }

Recall: Transactional Memory

- Transaction-based model of memory
  - Interface:
    start transaction();
    read/write data
    commit transaction():
  - If conflicts detected, commit will abort and must be retried
  - What is a conflict?
    - » If values you read are written by others in middle of transaction
    - » If values you write are written by others in middle of transaction
- Hardware support for transactions
  - Typically uses cache coherence protocol to help process
  - How to detect conflict?
    - » Set R/W flags on cache line when access
    - » Conflicts detected when cache line invalidates (and/or interventions) notice bits set
  - Eager Conflict detection:
    - » Newer transaction is assumed to conflict with older one

LogTM: Log-based Transactional Memory

- LogTM: Log-based Transactional Memory
  - Kevin Moore, Jayaram Bobba, Michelle Moravan, Mark Hill & David Wood
  - Use of Cache Coherence protocol to detect transaction conflicts
- Transactional Interface:
  - `begin_transaction()`: Request that subsequent statements for a transaction
  - `commit_transaction()`: Ends successful transaction begun by matching `begin_transaction()`. Discards any transaction state saved for potential abort
  - `abort_transaction()`: Transfers control to a previously register conflict handler which should undo and discard work since last `begin_transaction()`

```
for(i=0; i<10000; ++i){
  begin_transaction();
    new_total = total.count + 1;
    private_data[id].count++;
    total.count = new_total;
  commit_transaction();
  think();
}
```

Figure 4. Shared-counter microbenchmark (main loop)

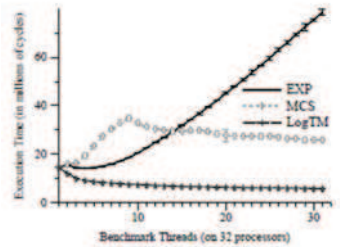
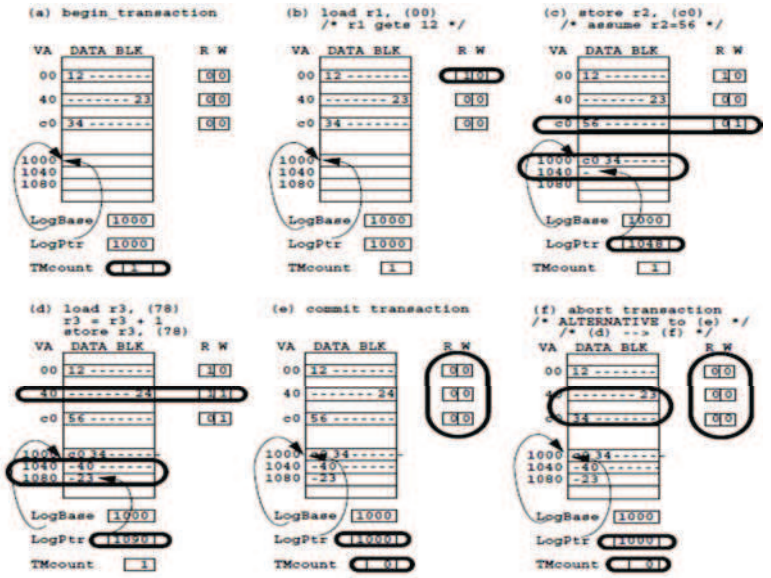


Figure 5. Execution time for LogTM transactions, test-and-test-and-set locks with exponential backoff (EXP) and MCS locks (MCS).

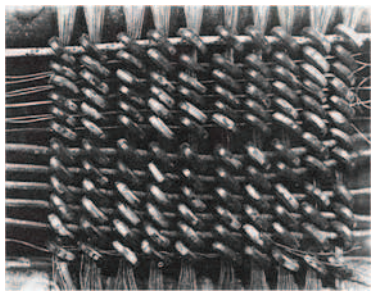
Specific Logging Mechanism



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*

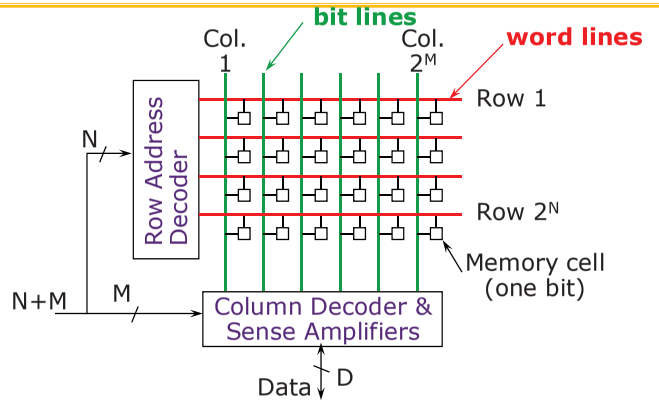
Core Memories (1950s & 60s)



The first magnetic core memory, from the IBM 405 Alphabetical Accounting Machine.

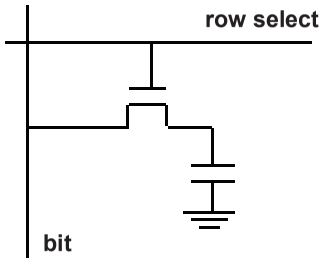
- Core Memory stored data as magnetization in iron rings
  - Iron "cores" woven into a 2-dimensional mesh of wires
  - Origin of the term "Dump Core"
  - Rumor that IBM consulted Life Saver company
- See: http://www.columbia.edu/acis/history/core.html

DRAM Architecture



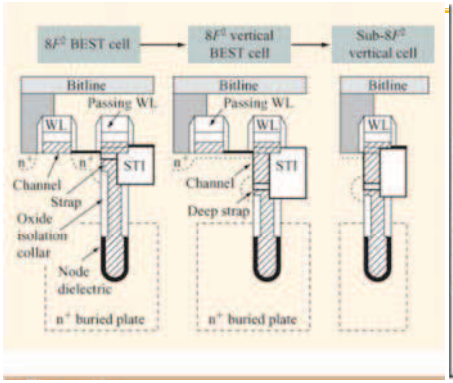
- Bits stored in 2-dimensional arrays on chip
- Modern chips have around 4 logical banks on each chip
  - each logical bank physically implemented as many smaller arrays

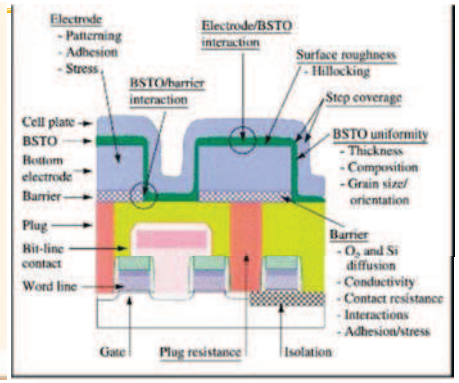
1-T Memory Cell (DRAM)



- **Write:**
  - 1. Drive bit line
  - 2.. Select row
- **Read:**
  - 1. Precharge bit line to Vdd/2
  - 2.. Select row
  - 3. Cell and bit line share charges
    - » Very small voltage changes on the bit line
  - 4. Sense (fancy sense amp)
    - » Can detect changes of ~1 million electrons
  - 5. Write: restore the value
- **Refresh**
  - 1. Just do a dummy read to every cell.

DRAM Capacitors: more capacitance in a small area





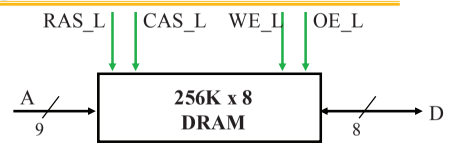
- **Trench capacitors:**
  - Logic ABOVE capacitor
  - Gain in surface area of capacitor
  - Better Scaling properties
  - Better Planarization
- **Stacked capacitors**
  - Logic BELOW capacitor
  - Gain in surface area of capacitor
  - 2-dim cross-section quite small

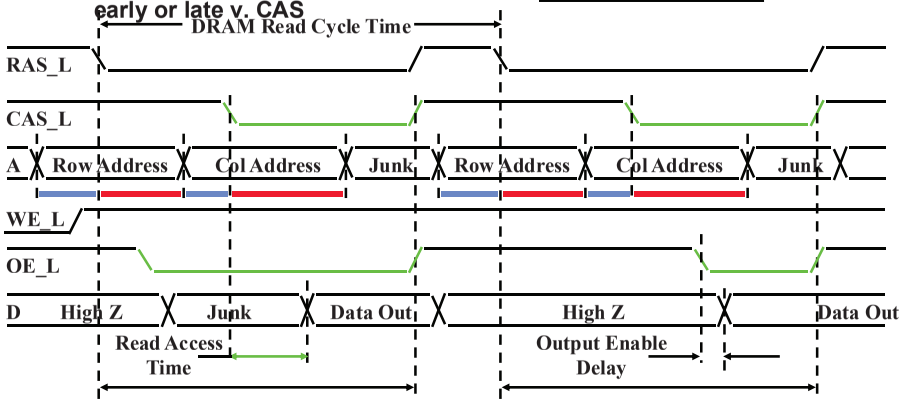
DRAM Operation: Three Steps

- **Precharge**
  - charges bit lines to known value, required before next row access
- **Row access (RAS)**
  - decode row address, enable addressed row (often multiple Kb in row)
  - bitlines share charge with storage cell
  - small change in voltage detected by sense amplifiers which latch whole row of bits
  - sense amplifiers drive bitlines full rail to recharge storage cells
- **Column access (CAS)**
  - decode column address to select small number of sense amplifier latches (4, 8, 16, or 32 bits depending on DRAM package)
  - on read, send latched bits out to chip pins
  - on write, change sense amplifier latches. which then charge storage cells to required value
  - can perform multiple column accesses on same row without another row access (burst mode)

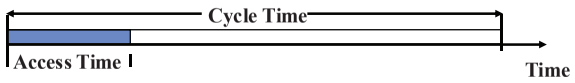
DRAM Read Timing (Example)

- **Every DRAM access begins at:**
  - The assertion of the RAS_L
  - 2 ways to read: early or late v. CAS



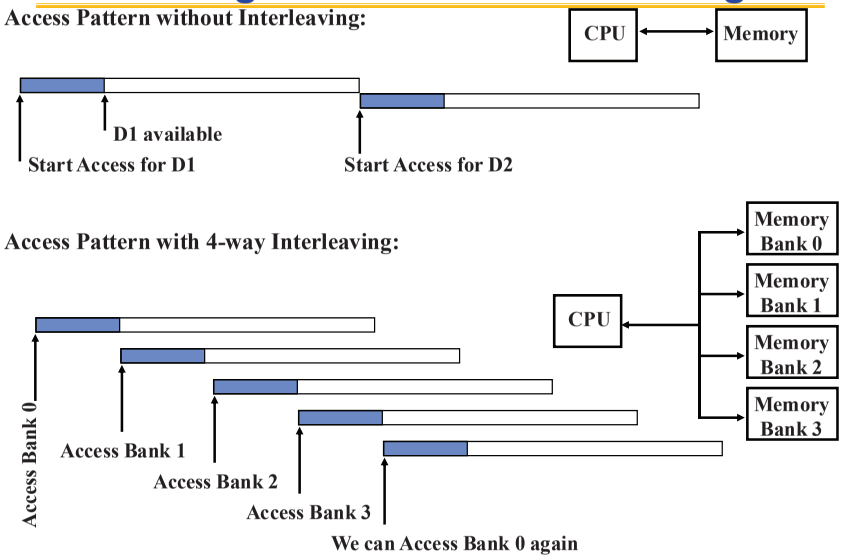


Main Memory Performance

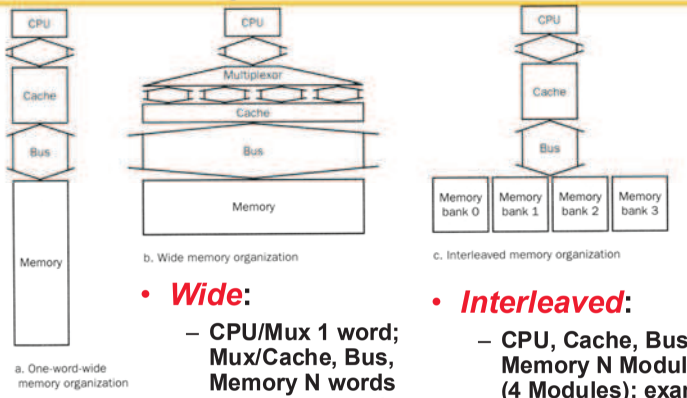


- DRAM (Read/Write) Cycle Time >> DRAM (Read/Write) Access Time
  - 2:1; why?
- DRAM (Read/Write) Cycle Time :
  - How frequent can you initiate an access?
  - Analogy: A little kid can only ask his father for money on Saturday
- DRAM (Read/Write) Access Time:
  - How quickly will you get what you want once you initiate an access?
  - Analogy: As soon as he asks, his father will give him the money
- DRAM Bandwidth Limitation analogy:
  - What happens if he runs out of money on Wednesday?

Increasing Bandwidth - Interleaving



Main Memory Performance



- *Wide*:
  - CPU/Mux 1 word; Mux/Cache, Bus, Memory N words (Alpha: 64 bits & 256 bits)
- *Interleaved*:
  - CPU, Cache, Bus 1 word: Memory N Modules (4 Modules); example is *word interleaved*
- *Simple*:
  - CPU, Cache, Bus, Memory same width (32 bits)

Quest for DRAM Performance

1. Fast Page mode
   - Add timing signals that allow repeated accesses to row buffer without another row access time
   - Such a buffer comes naturally, as each array will buffer 1024 to 2048 bits for each access
2. Synchronous DRAM (SDRAM)
   - Add a clock signal to DRAM interface, so that the repeated transfers would not bear overhead to synchronize with DRAM controller
3. Double Data Rate (DDR SDRAM)
   - Transfer data on both the rising edge and falling edge of the DRAM clock signal ⇒ doubling the peak data rate
   - DDR2 lowers power by dropping the voltage from 2.5 to 1.8 volts + offers higher clock rates: up to 400 MHz
   - DDR3 drops to 1.5 volts + higher clock rates: up to 800 MHz

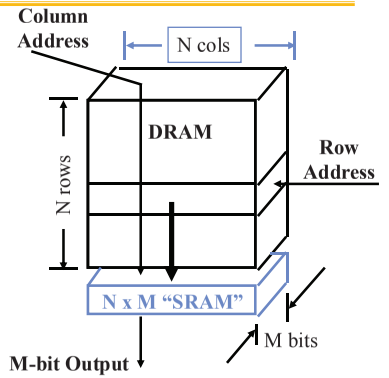
- Improved Bandwidth, not Latency

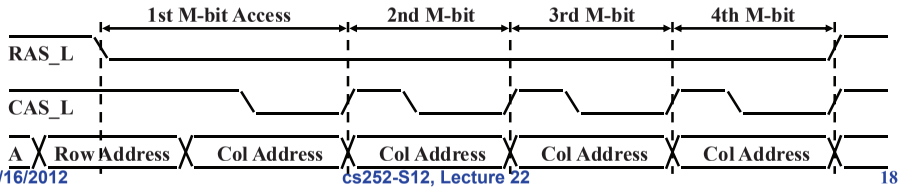
Fast Memory Systems: DRAM specific

- Multiple CAS accesses: several names (page mode)
  - *Extended Data Out (EDO)*: 30% faster in page mode
- Newer DRAMs to address gap; what will they cost, will they survive?
  - *RAMBUS*: startup company; reinvented DRAM interface
    - » Each Chip a module vs. slice of memory
    - » Short bus between CPU and chips
    - » Does own refresh
    - » Variable amount of data returned
    - » 1 byte / 2 ns (500 MB/s per chip)
  - *Synchronous DRAM*: 2 banks on chip, a clock signal to DRAM, transfer synchronous to system clock (66 - 150 MHz)
    - » DDR DRAM: Two transfers per clock (on rising *and* falling edge)
  - Intel claims *FB-DIMM* is the next big thing
    - » Stands for "Fully-Buffered Dual-Inline RAM"
    - » Same basic technology as DDR, but utilizes a serial "daisy-chain" channel between different memory components.

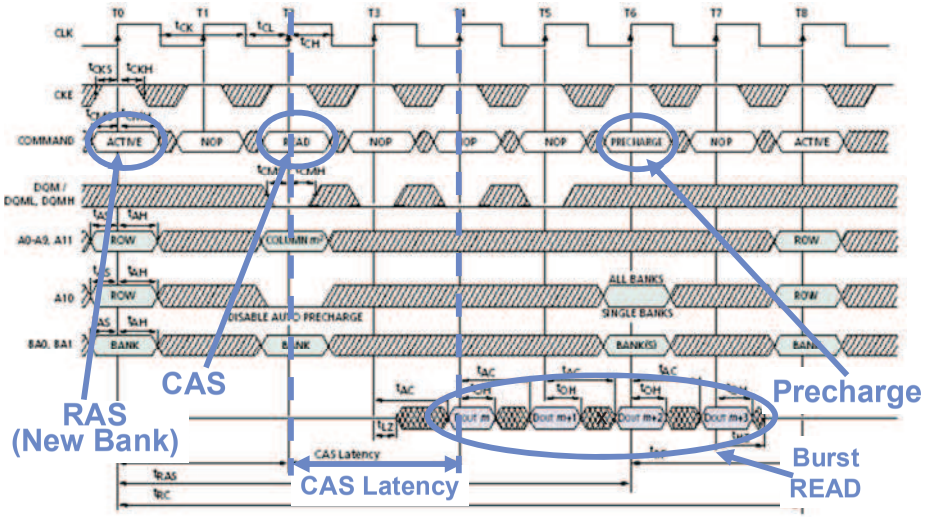
Fast Page Mode Operation

- Regular DRAM Organization:
  - N rows x N column x M-bit
  - Read & Write M-bit at a time
  - Each M-bit access requires a RAS / CAS cycle
- Fast Page Mode DRAM
  - N x M "SRAM" to save a row
- After a row is read into the register
  - Only CAS is needed to access other M-bit blocks on that row
  - RAS_L remains asserted while CAS_L is toggled



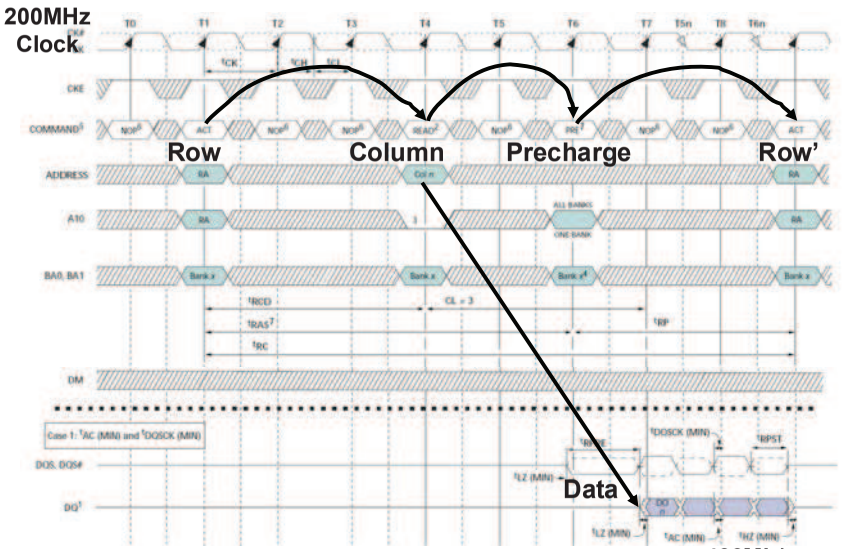


SDRAM timing (Single Data Rate)



- Micron 128M-bit dram (using 2Meg×16bit×4bank ver)
  - Row (12 bits), bank (2 bits), column (9 bits)

Double-Data Rate (DDR2) DRAM

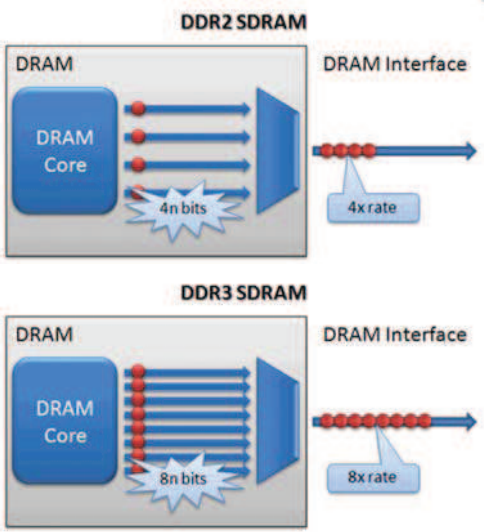


[Micron, 256Mb DDR2 SDRAM datasheet]

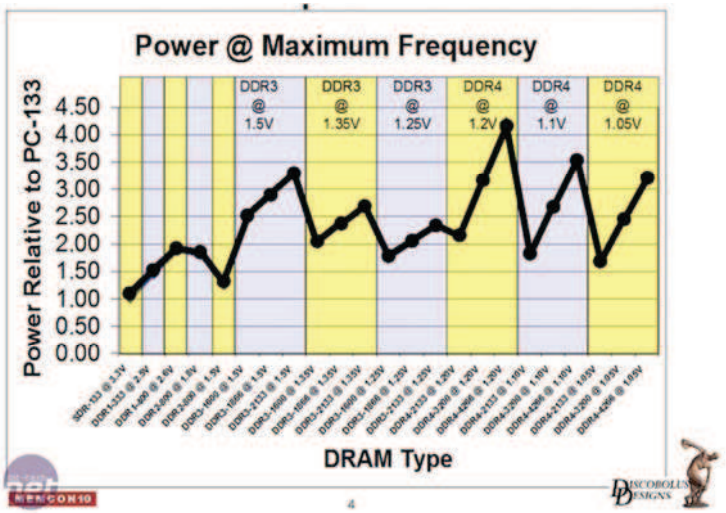
400Mb/s Data Rate

DDR vs DDR2 vs DDR3 vs DDR4

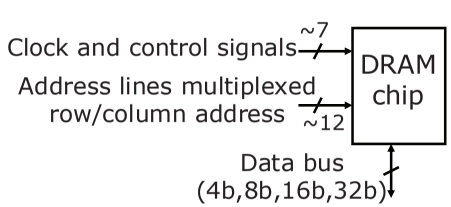
- All about increasing the rate at the pins
- Not an improvement in latency
  - In fact, latency can sometimes be worse
- Internal banks often consumed for increased bandwidth
- DDR4 (January 2011)
  - Samsung,…
  - Currently 2.13Gb/sec
  - Target: 4 Gb/sec



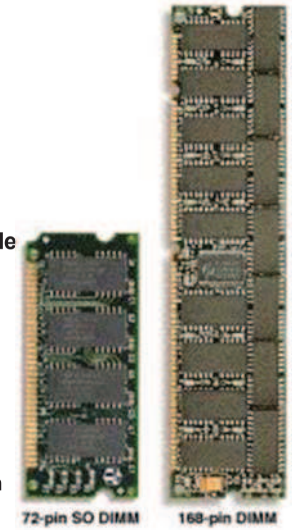
DRAM Power: Not always up, but…



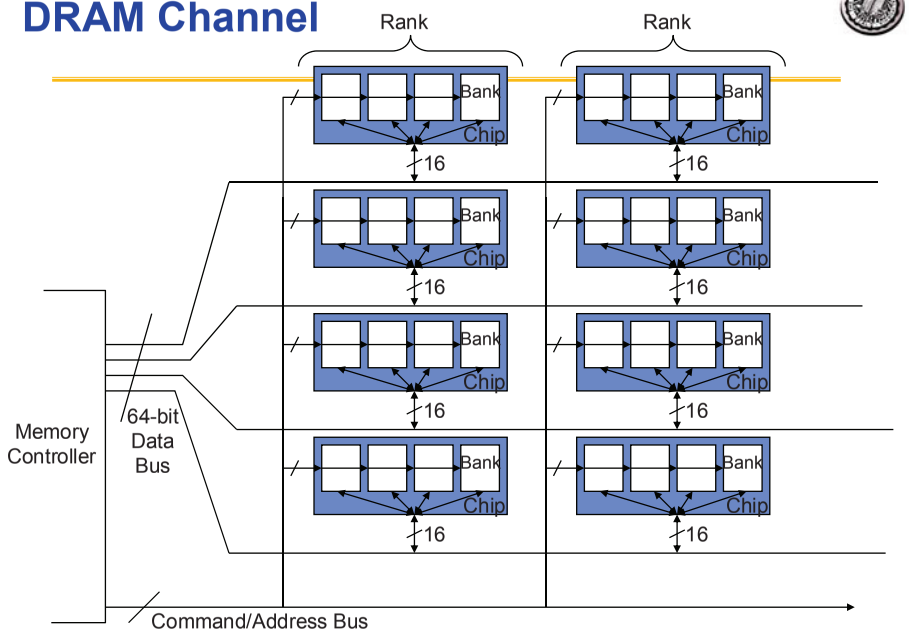
DRAM Packaging



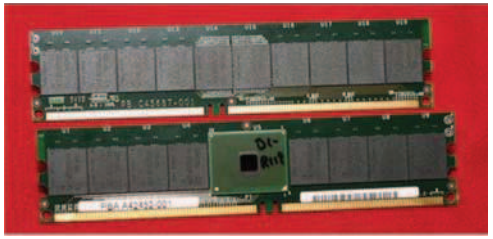
- DIMM (Dual Inline Memory Module) contains multiple chips arranged in "ranks"
- Each rank has clock/control/address signals connected in parallel (sometimes need buffers to drive signals to all chips), and data pins work together to return wide word
  - e.g., a rank could implement a 64-bit data bus using 16x4-bit chips, or a 64-bit data bus using 8x8-bit chips.
- A modern DIMM usually has one or two ranks (occasionally 4 if high capacity)
  - A rank will contain the same number of banks as each constituent chip (e.g., 4-8)



DRAM Channel



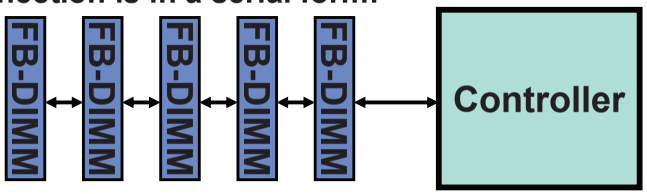
FB-DIMM Memories



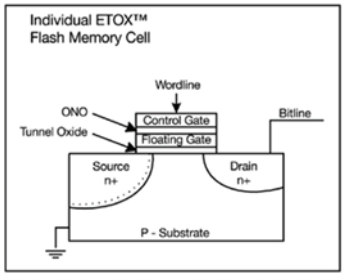
Regular DIMM

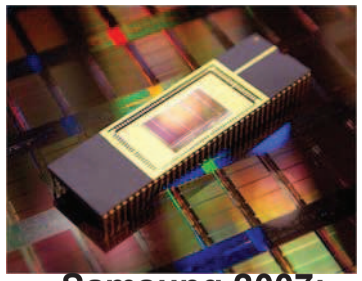
FB-DIMM

- Uses Commodity DRAMs with special controller on actual DIMM board
- Connection is in a serial form:



FLASH Memory

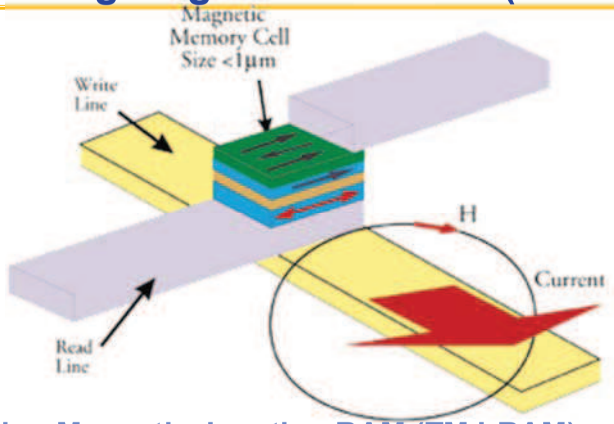




Samsung 2007: 16GB, NAND Flash

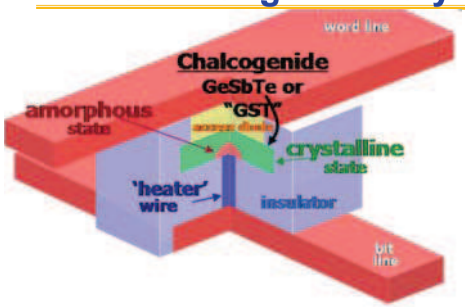
- Like a normal transistor but:
  - Has a floating gate that can hold charge
  - To write: raise or lower wordline high enough to cause charges to tunnel
  - To read: turn on wordline as if normal transistor
    - » presence of charge changes threshold and thus measured current
- Two varieties:
  - NAND: denser, must be read and written in blocks
  - NOR: much less dense, fast to read and write

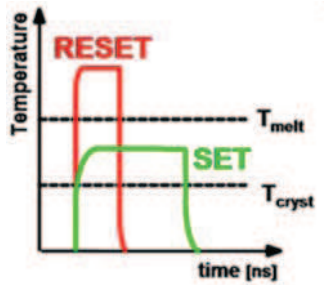
Tunneling Magnetic Junction (MRAM)



- Tunneling Magnetic Junction RAM (TMJ-RAM)
  - Speed of SRAM, density of DRAM, non-volatile (no refresh)
  - "Spintronics": combination quantum spin and electronics
  - Same technology used in high-density disk-drives

Phase Change memory (IBM, Samsung, Intel)





- Phase Change Memory (called PRAM or PCM)
  - Chalcogenide material can change from amorphous to crystalline state with application of heat
  - Two states have very different resistive properties
  - Similar to material used in CD-RW process
- Exciting alternative to FLASH
  - Higher speed
  - May be easy to integrate with CMOS processes

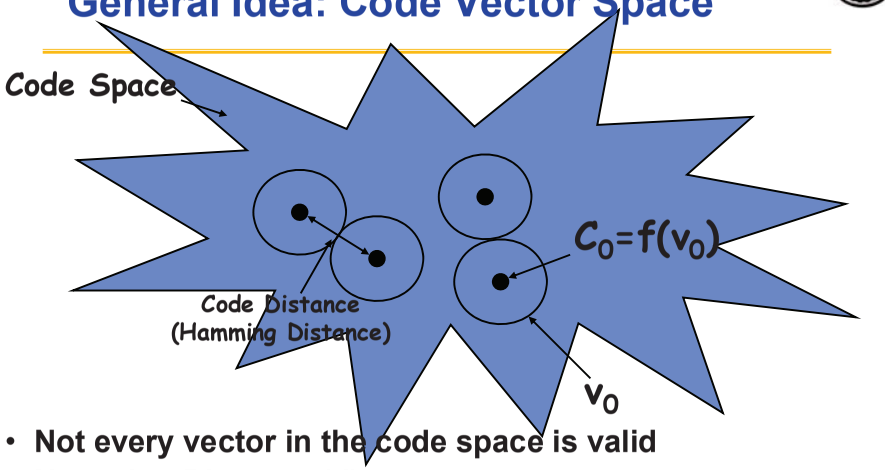
Error Correction Codes (ECC)

- 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.
  - Less frequently, "hard" errors can occur when chips permanently fail.
  - Problem gets worse as memories get denser and larger
- Where is "perfect" memory required?
  - servers, spacecraft/military computers, ebay, …
- Memories are protected against failures with ECCs
- Extra bits are added to each data-word
  - used to detect and/or correct faults in the memory system
  - in general, each possible data word value is mapped to a unique "code word". A fault changes a valid code word to an invalid one - which can be detected.

ECC Approach: Redundancy

- Approach: Redundancy
  - Add extra information so that we can recover from errors
  - Can we do better than just create complete copies?
- Block Codes: Data Coded in blocks
  - k data bits coded into n encoded bits
  - Measure of overhead: Rate of Code: K/N
  - Often called an (n,k) code
  - Consider data as vectors in GF(2) [i.e. vectors of bits]
- Code Space is set of all 2^n vectors, Data space set of 2^k vectors
  - Encoding function: $C=f(d)$
  - Decoding function: $d=f(C')$
  - Not all possible code vectors, *C, are valid!*

General Idea: Code Vector Space



- Not every vector in the code space is valid
- 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)

Some Code Types

- Linear Codes: $\vec{C} = \vec{G} \cdot \vec{d} \quad \vec{S} = \vec{H} \cdot \vec{C}$

  Code is *generated* by G and in *null-space* of H
  - (n,k) code: Data space 2^k, Code space 2^n
  - (n,k,d) code: specify distance d as well
- Random code:
  - Need to both identify errors *and* correct them
  - Distance d ⇒ correct ½(d-1) errors
- Erasure code:
  - Can correct errors if we know which bits/symbols are bad
  - Example: RAID codes, where "symbols" are blocks of disk
  - Distance d ⇒ correct (d-1) errors
- Error detection code:
  - Distance d ⇒ detect (d-1) errors
- Hamming Codes
  - d = 3 ⇒ Columns nonzero, Distinct
  - d = 4 ⇒ Columns nonzero, Distinct, Odd-weight
- Binary Golay code: based on quadratic residues mod 23
  - Binary code: [24, 12, 8] and [23, 12, 7].
  - Often used in space-based schemes, can correct 3 errors

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):

$$\overline{C_i} = \mathbf{G} \cdot \overline{v_i}$$ ← 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:
  - Can always turn non-systematic code into a systematic one (row ops)

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

- But – What is distance of code? Not Obvious!

Implicitly Defining Codes by Check Matrix

- Consider a parity-check matrix H (n×[n-k])
  - Define valid code words Ci as those that give S_i=0 (null space of H)

$$\overline{S_i} = \mathbf{H} \cdot \overline{C_i} = \overline{0}$$

  - Size of null space?
    (null-rank H)=k if (n-k) linearly independent columns in H
- Suppose we transmit code word C with error:
  - Model this as vector E which flips selected bits of C to get R (received):

$$\overline{R} = \overline{C} \oplus \overline{E}$$

  - Consider what happens when we multiply by H:

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

- 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
  - So – Code design is designing H matrix

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:

$$\mathbf{H} = (\mathbf{P} \mid \mathbf{I})$$ ← P is (n-k)×k, I is (n-k)×(n-k)
Result: H is (n-k)×n

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

$$\mathbf{G} = \begin{pmatrix} \mathbf{I} \\ \mathbf{P} \end{pmatrix}$$ ← P is (n-k)×k, I is k×k
Result: G is n×k

- Notice: G generates values in null-space of H *and* has k independent columns so generates 2^k unique values:

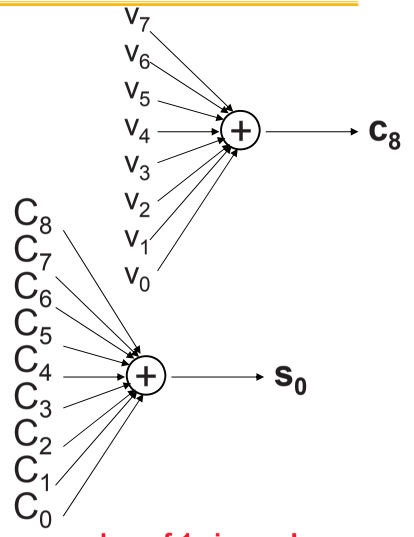
$$\overline{S_i} = \mathbf{H} \cdot \left(\mathbf{G} \cdot \overline{v_i}\right) = \left((\mathbf{P} \mid \mathbf{I}) \cdot \begin{pmatrix} \mathbf{I} \\ \mathbf{P} \end{pmatrix}\right) \cdot \overline{v_i} \equiv 0$$

Simple example (Parity, d=2)

- Parity code (8-bits):

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

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



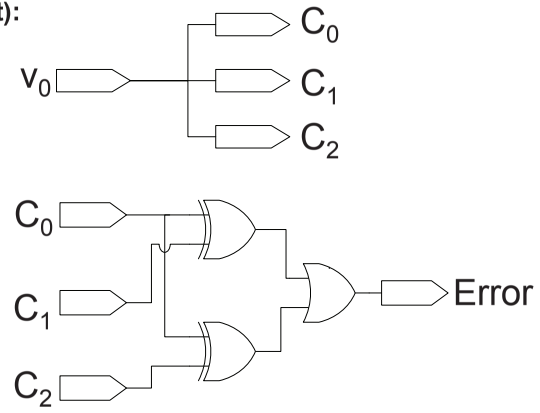
- Note: Complexity of logic depends on number of 1s in row!

Simple example: Repetition (voting, d=3)

- Repetition code (1-bit):

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

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



- Positives: simple
- Negatives:
  - Expensive: only 33% of code word is data
  - Not packed in Hamming-bound sense (only D=3). Could get much more efficient coding by encoding multiple bits at a time

Example: Hamming Code (d=3)

- Binary Hamming code meets Hamming bound
- Recall bound for d=3:

$$2^k \cdot (1+n) \le 2^n \Rightarrow n \le 2^{n-k} - 1$$

- So, rearranging:

$$k \le 2^c - (c+1), c = n - k$$

- Thus, for:
  - c=2 check bits, k ≤ 1 (Repetition code)
  - c=3 check bits, k ≤ 4
  - c=4 check bits, k ≤ 11, use k=8?
  - c=5 check bits, k ≤ 26, use k=16?
  - c=6 check bits, k ≤ 57, use k=32?
  - c=7 check bits, k ≤ 120, use k=64?
- H matrix consists of all unique, non-zero vectors
  - There are 2^c-1 vectors, c used for parity, so remaining 2^c-c-1

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

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

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!
- Circuit optimization: if throwing out column vectors, pick ones of highest weight (# bits=1) to simplify circuit
- 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
- What about EVENODD code?
  - Can be used to handle two erasures
  - What about two dead DRAMs? Yes, if you can really know they are dead

How to correct errors?

- Consider a parity-check matrix H (n×[n-k])
  - Compute the following *syndrome* S_i given code element C_i:

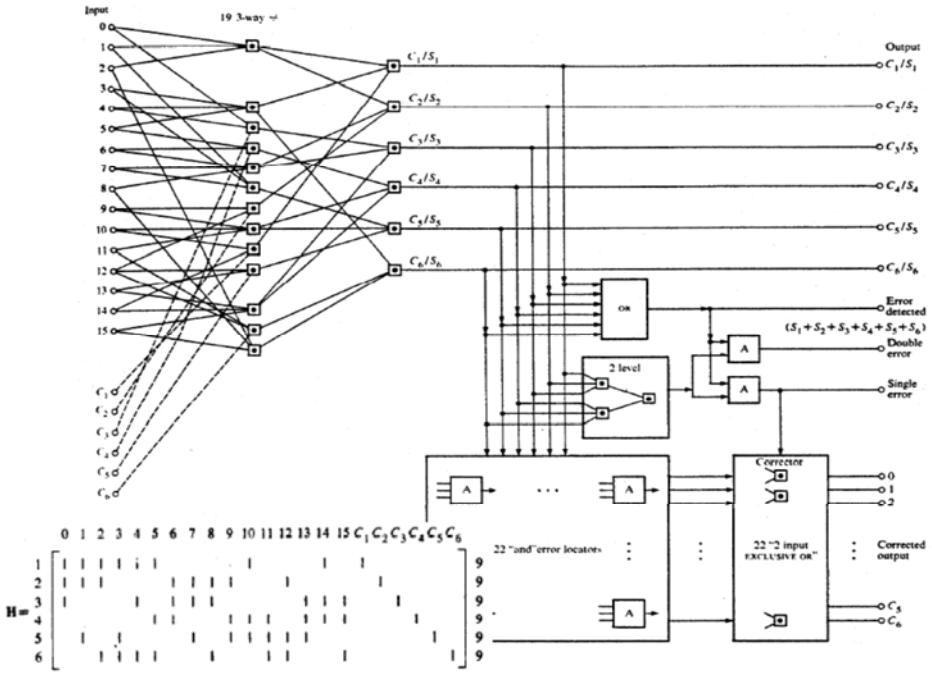
$$\overline{S_i} = \mathbf{H} \cdot \overline{C_i} = \mathbf{H} \cdot \overline{E}$$

- Suppose that two *correctable* error vectors E_1 and E_2 produce same syndrome:

$$\mathbf{H} \cdot \overline{E_1} = \mathbf{H} \cdot \overline{E_2} \Rightarrow \mathbf{H} \cdot \left(\overline{E_1} + \overline{E_2}\right) = 0$$

$$\Rightarrow \overline{E_1} + \overline{E_2} \text{ has d or more bits set}$$

- But, since both E_1 and E_2 have ≤ (d-1)/2 bits set, $E_1 + E_2 \leq$ d-1 bits set so this conclusion cannot be true!
- So, syndrome is unique indicator of correctable error vectors



Conclusion

- Main memory is Dense, Slow
  - Cycle time > Access time!
- Techniques to optimize memory
  - Wider Memory
  - Interleaved Memory: for sequential or independent accesses
  - Avoiding bank conflicts: SW & HW
  - DRAM specific optimizations: page mode & Specialty DRAM
- ECC: add redundancy to correct for errors
  - (n,k,d) ⇒ n code bits, k data bits, distance d
  - Linear codes: code vectors computed by linear transformation
- Erasure code: after identifying "erasures", can correct