

# EE 5900\_04 Lecture Notes - Spring '01

## EE 5900\_04 Memory System Design & Simulation MWF 13:05-13:55

### Wednesday 1/17- Class # 1

## Introduction

### • Who Am I

- Lecture Style
- From outline - so missing a lecture can seriously disrupt flow
- Please interrupt for questions!

### • My Contact Information

- My Email
  - btdavis@mtu.edu
- All Handouts via Web
  - [http://www.ee.mtu.edu/faculty/btdavis/courses/mtu\\_5900\\_spr01/](http://www.ee.mtu.edu/faculty/btdavis/courses/mtu_5900_spr01/)
- Office Hours - knock

### • Walk thru of Syllabus

- Prereqs!!!! - How many have them?
- Which Prereq?
- PreReq exam on Friday

### • Class movement issues

- Move from Fridays or Mondays to enable 3 day weekends
- ??? Question : Friday 2/2 -> Thursday 2/1 ???

### • Book Issues - 1st vs 2nd Edition

- Use for only the first 1/3 of course
- Suggested Texts have been requested at the library

## Focus & Intent of course

### • Distinction between Computer Engineering & Computer Science

- Computer Science - Efficient use of fixed computer system via software methods
- Computer Engineering - Determination & design of computer system architecture from device specification and requirements

- Co-design - Computer Engineering design task with regard for the range of software tasks to which the hardware will be applied

- **Examination of Computer SYSTEM**

- It is my belief that the primary avenues for increasing computer performance lie in the improvement of memory technologies and the communication mechanisms between components
- Majority of computer engineers do not end up designing processors, but a good fraction do end up on systems
  - Motherboard of desktop workstations <---> microprocessor controlled toaster
  - System design & specification tasks
- How to use available “off-the-shelf” devices to compile the best system for the design specification
  - Using existing bussing techniques - as devices are spec’ed out
  - Using FPGA’s or discrete logic where devices are not available

- **Processor Elements pertaining to Memory Hierarchy**

- Caches & Variants : Associativity, linesize, victim cache, trace cache
- Memory disambiguation
- Load/store operations under speculative executions

- **Memory Technologies & Configurations**

- Many Technologies available
- PROM(Flash) - Easy to read, power intensive to write, slow, small
  - non-volatile ; used for seldom changed programs and/or data
  - EPROM -> EEPROM -> FLASH
- SRAM
  - Widest grouping
  - Register file on mProc, Cache, device buffers
  - volatile
- DRAM
  - Main Memory : SDRAM, PC100, PC133, DDR266, DDR2, DRDRAM
  - volatile & requires refresh

- **Simulation**

- Tradeoffs at the system level are often non-intuitive
  - When extra die area becomes available is it best spent on larger cache, larger tlb, trace cache, victim cache, faster multiplier? - These questions can be answered via simulation
- Simulation can happen at many levels - Spim, Dinero, SimpleScalar, SimOS
- The more accurate your simulation model, the more accurate the results
- A 100% accurate model may be as/more difficult than developing the system

- Simulation Guided research/exploration
  - Use low-accuracy models to focus the area of concentration for more accurate models
  - low-accuracy Architectural level simulations done to examine feasibility
  - Design constraints established & functional blocks partitioned
  - Circuit-level simulations done on functional blocks to verify they meet spec
  - Feedback results into arch-level sim for more accurate results
- We will only be examining a single stage of simulation, as we are not a 100+ person design team

**READING : 4.1-4.4 of H&P by Monday**

## **Friday 1/19 - Class # 2**

### **PreReq Exam - 30 minutes**

## **Monday 1/22 - Class # 3**

- **Reminder of reading assignment**
- **Discuss Lab #1 Assignment**

## **Memory Hierarchy**

- **Pyramid Figure**
  - DRAW DIAGRAM (Pyramid)
  - Reg-File
  - On-Chip Cache
  - Off-Chip Cache
  - DRAM
  - Swap space (Disk)
- **Characteristics**
  - Cost (\$ / bit)
  - Access Latency
  - Quantity (bits / system)
- **8086 - First PC's**
  - DIAGRAM (PC\_XT Arch)
  - Direct Wired to the DRAM - same clock frequency
  - This impacts the ISA's of the era
    - low register counts
    - no prefetch or explicit cache management

- allow for self-modifying code
- Frequency Differential is now 60:1 to 200:1

- **Evolution of Memory Hierarchy**

- When processor  $\triangleleft$  memory freq differentials increased to where it was impacting performance, caches were added
- First on the motherboard - while these cache could maintain freq parity w/ proc
- Then in a multi-level design when only onchip-cache could keep pace w/ proc freq

- **P4 (Willamette) Memory (on-chip) Diagram**

- DIAGRAM (SysOpt webpage)
- Two levels of data cache on-chip
- Two levels of Instruction Cache on-chip
- First level cache is not shared - and is very different
- Third level of cache off chip - dedicated bus

- **System Architectural Diagram**

- DIAGRAM (Via Slot 1 chipset)
- North-Bridge/South-Bridge/AGP
  - Where is memory located in this figure?
  - Many places - just about every component
  - But the processor has the abstraction of a unified memory space?

## Wednesday 1/24 - Class # 4

- BANDWIDTH vs. LATENCY
  - bandwidth - throughput (bytes/sec) of an interconnection
    - Potential - pure calculations (bus-width \* frequency)
    - Effective (or realized) - bandwidth utilized in an interconnection
  - Latency - Time from desire for data/object to retrieval of data/object
    - Many different latencies For Example DRAM latency
    - DRAM advertised latency - from chip boundary to response
      - RAS / CAS / Data retrieval
    - Processor boundary - from access leaving processor to return to processor
      - +++ FSB occupancy ; DRAM Ctrl queueing ; DRAM bus occupancy
    - Processor observed latency - from Core data request to data availability
      - +++ Multiple level miss detection ; synchronization delay @ boundary

# Processor Attributes

## • Pipelining basics (4.1)

- Pipeline CPI = Ideal Pipeline CPI + Structural stalls + RAW stalls + WAR stalls + WAW stalls + Control stalls
  - Goal - to get Pipeline CPI as close to ideal as possible - reduce all other terms to 0
  - Structural stalls  $\rightarrow$  0 as enough resources (transistors) are applied to the problem
  - WAR & WAW stalls  $\rightarrow$  0 when using OO with store buffer
  - Remains : Ideal + RAW stalls (data dependencies) + control stalls (branches)
- Loop Unrolling
  - changing instruction ordering/mix to make more efficient use of functional units and reduce stall cycles
- Data Dependencies
  - EXAMPLE
    - LD            F0, 0(R1)
    - ADDD        F4, F0, F2
    - SD            0(R1), F4
  - Data dependencies cause pipeline stalls (classified in RAW stalls above)
  - Increase CPI
  - Worst case is that a LD has to go all the way out to the DRAM memory system

## Friday 1/26 - Class # 5

### READ : 4-5 through 4.11 for 2/1 - Thursday rescheduled class

- Name Dependencies
  - EXAMPLE
    - LD            F0, 0(R1)
    - ADDD        F4, F0, F2
    - ADDD        F2, F4, F6
    - LD            F6, 0(R2)
  - Can be avoided by register renaming either statically or dynamically
  - Static - Compiler optimized renaming
    - increases register pressure, and is dependant upon ISA - difficult in x86
  - Dynamic - "Register Renaming"
    - increased pipeline depth
- Control Dependencies
  - Subsequent instructions dependant upon preceeding branch

- Contributes to control stalls
- Motivates use of branch prediction (4.3) or speculative execution (

### • **Dynamic Scheduling (4.2)**

- a.k.a out-of-order execution
- not a “new” concept - done in ‘60’s era mainframes
- added to microprocessors when the transistors were available (early ‘90s)
- ?What is your level of understanding? - Not a “single” way to do OO
- DIAGRAM
  - Reservation Stations
  - pipes & RegFile
  - Reorder Buffer
- speedup factor : 1.7-2.5 for CDC6600
- Increases NAME dependancies problems
  - motivates use of a dynamically enamed registers
- ?Why is this critical to understanding the Memory Hierarchy?
  - Without OO techniques, a memory system need only support a single outstanding load
  - With OO techniques, the memory system must support multiple concurrent accesses
  - These techniques also allow the core to continue operation after/during a long-latency memory access - at least until the re-order buffers are full, or all instructions in the reservation stations are data dependant

## **Monday 1/29 - Class # 6**

### • **Speculative Execution**

- Impromptu material

### • **Load-Store Queue**

- Impromptu material

### • **Branch Prediction (4.3)**

- Alot of research has gone into this field - as with highly pipelined designs, branch misprediction significantly impacts performance - Could easily occupy 3 lectures
- Static Branch Prediction
  - Based on “hint” bits encoded into branch instructions
  - Ideal case 2 hint bits - (a) use (b) take/not take
  - Depends upon code analysis by compiler
- Dynamic Branch Prediction
  - Depends upon past history of branches

- one-bit predictor, two-bit predictor, dynamic multi-level predictors

## Wednesday 1/31 - Class # 7

### Reminder - Chapter 4 to have been read by tomorrow

#### • Branch Prediction (4.3) - continued

- Up to 94% accuracy - when allocated enough resources - for SPEC95
- Multiple Level predictors
  - History -> Pattern -> Prediction
  - History can be allocated Globally or per address
  - Amount of history maintained is bits of history
  - Pattern table can be allocated globally or per address
  - number of patterns is determined by the bits of the history register
  - Prediction can be adaptive (2-bit) or static
  - Nomenclature : GAg, GSg, PAg, PAp
- Resources required - increase rapidly, exponentially in some cases
  - When designing a processor, determinations have to be made, where are resources (i.e. transistors or die area) best allocated for optimal performance
  - Storage requirements are significant especially when using ?Ap or PAp?
  - Requirements can be arithmetically determined - but this is not a course on BP
- GShare
  - Use the address of the branch as a partial hash for the history
  - Reduces some of the aliasing, allowing global schemes to approach per-address perf
  - The history is XORed with the history before going to the pattern history table
- Hybrid Predictors
  - Motivated by the realization that 60% of branches have a static outcome
  - Error checking branches, etc - Why let these pollute the dynamic structures
  - MANY hybrid predictors are proposed - two (or more) components predictors chosen by a “meta-predictor”
  - Typically involve a simple two-bit saturating, and a complex two-level
  - Requires still more resources, and complexity, but can increase performance

## Thursday 2/1 - Class # 8 - Rescheduled 3pm in EERC316

- Branch Target Buffer
  - Only accessed by branches “predicted taken”
  - Functions as a predictor of the taken address

- Eliminates the necessity for address generation of branches, which may be difficult due to data dependencies, and certainly requires a cycle
- Eliminates the stall cycles necessary from AG
- Also requires resources
- Return Address Stack
  - RETURNS are Branches with known outcome
  - But target of the branch is not static - thus poorly predicted by BTB
  - Return address stack holds the addresses from which subroutines were invoked
  - allows RET to be 100% predicted w/ 100% accuracy
  - depth of the stack is a parameter, and stack must be able to be pushed to a memory struct

- **SIMD execution**

- impromptu response to questions
- Vector Processors
- ISA Extensions (MMX, SSE, 3DNow!)

## Monday 2/5 - Class #9

- **SuperScalar pipelines (4.4)**

- SuperScalar is essentially putting multiple pipeline in parallel
- ? DIAGRAM - superscalar stairway ?
- ? DIAGRAM - OO superscalar core ?
- Typically not all pipelines are general purpose - and there may be more pipelines than there are available issue slots
- Brings into design parameters a number of new variables
  - Pipeline Types and numbers
    - Parameterizable w/in SimpleScalar
    - ialu, imult, memport, fpalu, fpmult - sometimes branches also - alternatively “branch folding”
    - number of each depends upon instruction mix - fp pipes are larger in die area
  - Fetch Width
    - Determined by bandwidth of connection between L1 cache & memory
    - Even still - frequently limited by alignment issues
    - Variable instruction length (x86) makes this even more difficult
    - Fetch double the bytes required for issue
  - Issue Width
    - Determined by architectural constraints & dependency checking hardware

- Involves identification of instructions in res-stations w/ no unresolved dependancies & transferring from res-stations to pipeline
- This sets Upper bound on “ideal” CPI
- revisit  $CPI = ideal + Structural\ stalls + RAW\ stalls + WAR\ stalls + WAW\ stalls + Control\ stalls$
- Instructions Retired per cycle
  - Upper bound on req'd # is Issue width
  - Actual CPI cooresponds to AVERAGE Instructions retired/cycle over time
  - Involves identification of oldest instructions in reorder buffer - determination if they are complete - then writeback/commit of thier results
  - Easier to impliment than Issue, so less of an architectural constraint
- Pipelines vary in depth & function - best integrated into an OO core

## Wednesday 2/7 - Class #10

**READ : Kroft Paper - "Lockup-Free Instruction Fetch/Prefetch Cache Organization." - By Wednesday 2/14**

**READ : Patel TechReport/Thesis - Pages 1-22 by Wednesday 2/14**

**Please write down questions pertaining to these papers as you are reading them.**

- Limitations to pipeline width
  - dependancy checking grows exponentially
  - Ports to the register files & memory system
  - inherent limitation in ILP
  - difficulty in building devices of that scale
- VLIW - Very long Instruction Word
  - Each “instruction word” contains multiple instructions in the current nomenclature
  - The compiler is responsible for scheduling the dependant instructions such that they can execute without stall
  - Eliminates the dependancy checking within a “word”
  - The EPIC or IA64 architecture is a variant of a VLIW architecture

### • Load-Store Pipeline(s)

- One of the required pipeline types in a heterogeneous superscalar core
- Can be a unified L/S pipe - or discrete Load & Store pipes - ADV/DIS
- For x86 instruction set - one of every 5 instructions is a load/store - less for ISA's with more registerers
- A pipeline is required for address generation, and subsequent access stages

- Must have a mechanism for holding Cache-Miss accesses w/o blocking the pipeline
  - MSHR - miss information/status holding register

## Monday 2/12 - Class # 11

- Speculation - Loads can be done speculatively, but Stores can not
  - Store Queue handles this problem
  - Frequently integrated into a load/store queue to eliminate load stalls in the load-pipe(s)
  - Allow stores to be delayed until the commitment of the instruction
  - Allow loads to compare values against preceeding stores
    - Requires Content Addressable Memory - low # of MSHRs/Queue entries
  - Allow loads to stall for high-latency (DRAM, L3 cache) accesses w/o stalling pipeline(s)
  - Entries into the store queue - parameter to the architecture
  - MSHRs are equivalent in functionality to the load/store queues we have been discussing in abstract
- Ports to Memories
  - What is a dual ported memory structure
  - Latency of a dual ported memory structure is higher than that of a single ported
  - This is one advantage to splitting the I/D side L1 caches

### • Trace Cache

- Modern Cache approach - attempts to respond to the Reservation Station FILL problem
- Paper to be read describes implimentation in a RISC context - slightly different usage in a CISC/ hardware emulated context - Examin RISC context first
- Problem : branches & small basic blocks make it difficult to utilize fetch bandwidth
- Solution : cache instructions by basic block & sequence rather than by address
- A sequence of basic blocks in a single trace-cache line is called a segment

## Wednesday 2/14 - Class # 12

- DIAGRAM - Pages 8 & 9 of Patel
- Indexed by fetch address of first basic block & (partially) branch prediction
  - should be associative - more than one block may contain the same first basic block
  - should allow for partial match - if branch predictions are only partially correct
- Trace cache lines are gathered by the Fill Unit
  - Fill Unit is an algorithmic challenge in itself - Patel Suggests 4 rules for segment completion
  - 1. segment contains 16 instructions
  - 2. segment contains 3 conditional branches
  - 3. segment contains an indirect jump, return or trap

- 4. merging the next incoming block would result in segment length > 16
- Implications
  - Instructions may be cached in multiple places in the trace cache
  - The trace cache line must contain target for all branch outcomes - to expedite fetch, and because otherwise basic block addresses must be kept to calculate branch targets - and storage overhead is equivalent.
  - Tag is address (partial if not FA) and branch outcomes
- CISC advantages
  - CISC Decode typically requires conversion from Complex ISA -> Internal ISA
  - True of P3/P4/Athlon - Somewhat true of Crusoe, but not the whole story
  - In this case, the decoded Internal ISA (ROP) instructions can be cached in the trace cache
  - Other advantages - increase fetch effectiveness - are consistent
- Branch Predictor & Instruction Cache
  - These functional units are primarily unchanged from a non Trace-Cache implementation
  - The Branch predictor must be capable of predicting three branches into the future (a separate and complex issue) - generates a set of (3) 2-bit predictors for each branch address
  - The Instruction Cache remains primarily unchanged, but might not require as long a line-size as a non Trace Cache implementation

### • **Compiler Optimizations (4.5) - Not the focus**

- Elimination of Dependancies
  - Dependancies impact performance by restricting the issue of the processor
  - Compilers should attempt to eliminate dependancies where they can be identified
  - loop-carried dependancy - each iteration dependant upon the preceeding iteration - can not be eliminated - but it's
  - register/name dependancy - can be eliminated by compiler techniques if enough registers are available

## **Friday 2/16 - Class # 13**

- Increase parallelism
  - Subtle differences between these techniques
  - Loop Unrolling
    - Unrolling a loop such that multiple iterations of the original loop occur in a single iteration of the unrolled loop
  - Software Pipelining
    - Unrolling a loop such that a single iteration of the original loop is carried across multiple iterations of the unrolled loop - effective as a memory prefetch technique
  - Trace Scheduling

- Critical Path Scheduling & Trace Compaction
- Attacks same problem as trace cache
- Cache line alignment of branch targets / basic blocks
- Makes the frequently executed basic blocks (plural) in a common cache line
- Put infrequently executed branch targets or basic blocks off the fall-through path

• **Hardware Support for Extracting ILP (4.6) - covered somewhat 1/29**

- Techniques requiring support in the: compiler, ISA and hardware
- This means changes to the ISA - can be done by start-from-scratch or ISA additions
- Predicated Execution (rather than branches)
  - Branches have a negative impact upon performance because they change the fetch path
  - Predicated instructions do not change the fetch path, but writeback conditional upon a predicate
  - Many novel architectures include support for predicates (IA64)
  - Common & Simple instructions are frequently predicated, CMOV
  - EXAMPLE
    - if (b!=0) a = a + b ; else a = a + 1
    - ld            r1, 0(r16)
    - pcmp.eq      p1,r1,0
    - add           r0, r0, r1      (p1)
    - addi          r0, r0, 1      (!p1)
  - Increases basic block size - increases fetch bandwidth
  - Book discusses two approaches
    - Predicated instructions w/ predicate explicit - requires many register specifiers and is high latency
    - Poison bits for exception - exceptions can be high latency
    - Most approaches being proposed or implemented now use explicit predicates which can be calculated & used (repeated) in an instruction encoding
- Dynamic Register Renaming
  - Dynamic Register Renaming has been mentioned before - it reduces the pressure on a small register file - or eliminates structural hazards in an OO-execution core
  - Goal : eliminate the structural hazards/stalls caused by name dependencies - with “infinite” register resources these hazards are completely eliminated
  - Allows for multiple versions of a single register NAME depending

**Monday 2/19 - Class # 14**

- May either eliminate completely the physical register file, or may place the physical register file into a background or shadowed role (retirement register file).

- Allows for a different number of “physical registers” from “architected registers”
  - physical registers - number of registers in the register file(s) of the processor, available for use w/o memory access - can be different between implimentation of same ISA
  - architected registers - number of registers available to the instructions - encoded in the ISA
- Requires an additional stage in instruction decode (done IN ORDER) which maps the (architected) instruction encoded register specifiers into (physical) register specifiers - the physical register specifiers then proceed with the instruction through the remainder of the pipeline.
- The state of the register mappings are maintained in the RAT - Register Alias Table - this structure is accessed by the remapping stage, and the ReOrderBuffer - the re-order buffer only needs to access it to determine which physical register to commit to, and how to “roll-back” the RAT in the case of miss-speculation
- EXAMPLE - Dynamic Register Renaming / Name Dependance
  - DIAGRAM - showing DECODE/OO\_CORE/Retirement
  - $r1 = r2 + r3$
  - $r4 = r1 + r5$
  - $r1 = r6 + r7$
  - $r8 = r1 + r4$
  - Architected registers vs. physical registers. Register renaming?
  - Rename the previous example where the Register Alias Table (RAT) is initially:
    - $r1 \rightarrow p12$     $r2 \rightarrow p6$     $r3 \rightarrow p9$     $r4 \rightarrow p15$
    - $r5 \rightarrow p1$     $r6 \rightarrow p10$     $r7 \rightarrow p8$     $r8 \rightarrow p14$
    - Free List:  $p5, p11, p13, p4$ .
- Allows the elimination of all false/Name dependancies (WAR & WAW)
- RAW - “true” dependancies remain
- Explicit prefetch (ISA must support)
  - Allows a known high-latency access to be initiated early, such that other instructions can execute in parallel
  - Can already be done with “load hoisting” why are new instructions necessary
  - Prefetches do not : Pollute registers, cause exceptions
  - Prefetches can be : predicated, predicted address
  - Goal : Eliminates the stalls associated with dependancies upon high-latency loads

## Wednesday 2/21 - Class # 15

### READ : 5.1 - 5.4 by Monday 2/26

- Speculation - Putting it all together

- Branch Prediction - Register Renaming - Reservation Stations - Store Queues - ReOrder Buffers
- DIAGRAM - How these parts fit into an OO superscalar core
- This is the baseline for most modern microprocessors - Not everything is here, but this gives you a good framework for understanding the other portions of the system
- Pipeline's w/in pipelines - Pipelines are a key to faster clock speeds and higher ILP
- Current designs save speculation resolution until the Commit/ROB stage to make recovery easier to implement
- Possible to speculate on multiple contexts - predict down multiple branches
- Possible to contain state for two concurrent threads (SMT) - next generation
  - Maintains highest level of throughput for any one thread
  - Uses extra resources for other threads
- Basic framework for "typical" modern microprocessors - High exploitation of ILP requires

### • The Limits to ILP (4.7)

- ILP - Instruction Level Parallelism
- Assumptions to maximize ILP
  - Register Renaming - infinite registers - all WAR & WAW (name dependencies) are avoided
  - Branch Prediction - perfect branch prediction
  - Jump (Branch Target) prediction - perfect target address prediction
  - Memory disambiguation - all load/store addresses are known and can be resolved OO w/o waiting for address generation
  - Infinite Fetch & Decode bandwidth
- Available ILP : 17-150 for SPEC
- Book goes through constraints on these #'s due to "real" fetch/issue/BP/Registers/Memory Disambiguation
- I am not going to spend time on that - but I do want to address the inherent limits of the way that we encode programs
- Are the limits encoded into our ISA's
- ISA's assume sequential execution of programs
- Sequential execution - regulated by the program counter - of programs makes the "critical path" time through the program longer
- Dataflow - One example which breaks this mode, not the only one
  - DIAGRAM (car assembly?)
  - Focus on the critical path of execution
  - Break workload into component tasks
  - Minimize the dependencies between these tasks

- Execute each of these component tasks upon a unique processor core (may be same chip)
- How do we encode a dataflow program?
- Thread-level-parallelism in Software terminology
- Requires additional annotations to describe multiple starting points or parallel tasks
- Popular in MultiMedia/Graphics engines
- Experimental/Proof-of-concept machines have been built which follow the Dataflow model

## Homework #1 - 4.10 & 4.20 due 2/28

## Memory Technologies

### • Motivation for Virtual Memory - Impromptu

- Use of larger memory space than physically available
- Protection - allows O/S to prevent one process from “mucking with” the data of another process or the O/S
- Relocation - allows O/S to provide each app with it’s own virtual memory space, using whatever virtual addresses the process desire(s)

## Friday 2/23 - Class Cancelled (health)

## Monday 2/26 - Class # 16

### • Impromptu

- Write allocation & write-back/through policies

### • Return to Memory Hierarchy

- DIAGRAM
- What is not shown?
  - Flash BIOS - on MBoard & Controller cards
  - Disk Drive ROM & Caches
  - Graphics Card ROM & Buffers
  - Network Card Buffers
  - Essentially I/O device memories
  - Should be included : Tape, Optical Disk, Magnetic disk, FeCore, Magnetic Drum

### • Memory Characteristics

- Principle of Locality
  - locality of reference : programs tend to reuse data and instructions they have used recently
    - Especially true on stack data types or array processing
    - This is the property which makes long cache lines useful
- Temporal locality

- Items accessed recently in time are likely to be accessed in the near future
- This is the property which makes caches useful
- Spatial locality
  - Items whose addresses are close together are likely to be accessed in close temporal proximity
- Volatile vs. Non-Volatile
  - non-volatile - will retain data even in the absence of supplied power
  - volatile - will retain data only while constraints upon the supply power, and other possible constraints (refresh) are maintained
- Sequential access vs Random Access vs pseudo-random access
  - Sequential access - data locations are read out in address order - changing this order increases the subsequent access times - access time dependant upon distance from last address read
    - Drum, Tape, disk(to some degree)
  - Random access - data locations can be read out in any order - access time is equal for the next byte or a byte 1/2 way across the memory space - completely uniform access time
    - SRAM, \*ROM, Flash, first-generation DRAM
  - Pseudo-random access - non-uniform access time - precise parameters dependant upon specific architecture
    - FPM DRAM, EDO DRAM, SDRAM, DRDRAM
    - note : despite RAM in name - these technologies are not random access

## Wednesday 2/28 - Class # 17

### • SRAM

- Diagram - Logical diagram (printout) - build up gate by gate
  - bistable flip-flop
  - How many transistors to implement?
- Diagram - SRAM Cell (Sharma Fig 2.4) - CMOS(a) & Dual-Port(c)
  - 6-transistor basic cell - higher density than the logical diagram
  - Read Operation - Both Bit &  $\overline{\text{Bit}}$  held high, then enable word
  - Write Operation - Bit &  $\overline{\text{Bit}}$  set to be appropriate for the data to be written, then enable word
- Diagram - SRAM Device (Sharma Fig 2.5)
  - Many “cells” organized in an array
  - Multiple arrays can be organized in “banks”
- Many technologies have been used for Cache design - I have focussed upon CMOS here because it is the most common general purpose process
  - Familiarity w/ NMOS vs CMOS vs PMOS?
  - nMOS - early, fewer transistors, higher power consumption

- CMOS - Complimentary, more transistors, lower power consumption
- BiPolar (DCTL, ECL, BiCMOS) - faster, more power consumption, less mature processes

## Monday 3/12 - Class # 18

- Interface
  - As with most memory devices - while the core can remain constant the interface may vary
  - On chip SRAM can have whatever timing diagrams the designers desire (i.e. Alpha dual-speed on-chip cache)
  - Many varieties of SRAM are available (asynch, synch, synch-burst, sequential)
  - Asynch Read Timing Diagram (TI printout) - Diagram & Block Diagram
  - Flow-through Read Timing Diagram (TI printout) - Diagram & Block Diagram
  - Piplelined Read Timing Diagram (TI printout) - Digarm & Block Diagram
  - SRAM Write Timings
    - Default(TI printout)- why is this bad?
    - Late Write & Zero Bus Turnaround (TI printout) - why better?
- SRAM placement
  - How is SRAM different than say a register file - not - 32 bit lines w/ many ports
  - Mproc structures - Branch Target Buffers, caches, reservation stations, ROB all implimented as SRAM
  - Frequently have “sub-block” writes, multi-port, or other use-specific designs

## Wednesday 3/14 - Class # 19

- **Cover Lab # 3 & Review for exam**

## Friday 3/16 - Exam # 1

## Monday 3/19 - Class # 20

- **Caches**
  - Caches can be built of any memory structure - but are almost always SRAM
  - Cache line & cache block are typically synonymous
  - Caches by definition cover only a portion of the entire memory space
  - This means that they require TAGS to determine which portions of the memory spare are covered
  - Critical Word first - how it affects latency
  - Where can a block be placed in a cache
    - Anywhere - Fully Associative
    - A single location - Direct Mapped (1-way set associative)
    - N possile locations (N-way set associative)
  - How is a block found in the cache

- Physical Address components
  - byte-in-block (block offset) - low order
  - index - set of bits from above byte-in-block (not necessarily low-order or contiguous)
  - tag - remaining bits
- Typical for index to be low order, but not necessary
- Retrieve all blocks in set & compare tags in parallel
- Block allocation/Replacement
  - When a set is completely populated, yet a new block must be brought into the set, how do we determine which block is ejected
  - clean first- replace clean blocks in the set first, minimizing bus traffic - does it? - can be integrated with any of the policies listed below
  - LRU - least recently used - most typically implemented as “pseudo lru”
  - FIFO - first in first out
  - random - randomly select a block to be ejected
  - Ideal - longest latency to next use
  - Policy decision - all will work correctly, seeking the policy yielding the best performance. May be different for different benchmarks/workloads
- Write Policy
  - Covered w/ regard to write allocation on 2/26
  - Write Through - every write goes through to the lower level cache, thus a line does not require a write when ejection occurs
  - Write Back - dirty bytes/blocks are maintained in the cache - requiring dirty bit(s) for the cache block or sub-blocks. These dirty lines must then be written back upon ejection
  - Write Back vs. Write Through - Write back yields lower traffic on the bus below the cache being discussed, but can yield higher latencies when a block must be written back prior to retrieval of a freshly allocated block
  - Write Allocation - When a write occurs to a line not presently in the cache, the cache designers can choose to allocate that line into the cache, or pass the write down the memory hierarchy to be written at that level.
  - Cache coherence - in SMP environments, cache coherence requires that processors be aware of what (dirty) cache lines are being maintained by their peer processors. This allows the processors to maintain a consistent state of the shared memory. One protocol which is designed for this purpose is the MESI protocol.

## • Cache Misses

- The three C's of Cache misses
- Compulsory - cold start misses - first access to block, so block must be retrieved - can be reduced by larger cache lines or via prefetching

- Capacity - block desired was previously in the cache, but ejected due to cache capacity limitations
  - can be reduced by increasing cache size
- Conflict - block desired was previously in the cache, but ejected due to capacity limitations within the set of this block - can be reduced by increasing cache associativity

## End of Material for Exam # 1

### • Cache Performance

- Cache performance is based upon the benchmarks being run
- That being said there are a number of techniques used to potentially increase performance
- Block Size
  - Increasing block sizes effectively increases prefetching to spatially local addresses
  - For a fixed cache size - reduces the # of blocks which can be housed in the cache
  - Larger block size is preferred for I-side caches
  - Can increase the latency of an access unless the cache fetches “critical word first”
- Associativity
  - Increased associativity can reduce the number of conflict misses in a cache
  - Associativity typically slows down the cache access - but this can be compensated for using different architectural layouts
  - Higher associativity becomes less useful as the overall cache size becomes larger, as there are fewer potential blocks mapping into the same set
  - Higher associativity is preferred for D-side caches
- Victim Caches
  - Low associativity can result in significant conflict misses
  - Victim caches function as a FIFO of the most recently ejected cache lines, from ANY set
  - Accesses must thus access the traditional (L1) cache and the Victim cache in parallel prior to an L2 access

## Wednesday 3/21 - Class #21

### • Aside : Discussion of Lab # 3

- Devices/Banks/Buswidths/memory space - how to determine relationships
- Addressing bit usage

## Friday 3/23 - Class # 22

- Victim Caches - Continued
  - Typically Victim caches are fully associative, but very small 4-16 entries
  - They provide elimination of conflict misses in “hot sets” in the cache
  - Because of the small size & high associativity Victim caches require significant tag bits
- Prefetching

- Prefetching is the retrieval of data from the memory system, into a buffer, or higher level memory prior to it's use by the compute engine
- Intent is to reduce the AVERAGE latency of accesses
- Hardware Controlled Prefetching
  - The hardware should identify some patterns to the accesses that it has seen in the past
  - From this pattern, the hardware generates accesses to the memory system, prior to requests by the core
  - These requests can be performed speculatively because they do not affect "correctness" of the program
- Stream Buffers
  - Proposed by Norm Jouppi
  - Another "bandaid" for caches - accessed in parallel with the primary cache
  - Prefetches the addresses following the missed address of the main cache
  - Stream buffers reduce compulsory as well as capacity misses
  - Each stream buffer can have a low number of lines (1-4)
  - Typically multiple streams are constructed (2-8) but not limited to power of two
  - FIFO - style construction w/ head & tail
  - Policy decisions on when to fetch next line
- Hardware based I-side prefetching (sequential cache lines) is common
  - I-side Stream buffers - Multiple streams (LRU allocated)
  - Policy decisions on when to request - when line is accessed
- Hardware based D-side prefetching is more complex
  - D-side stream buffers - less useful than I-side
  - Stride-based D-side prefetching - good for matrix code
    - Can be based on a stream buffer with an offset
    - More complex to detect than sequential - must keep track of last missed address for each load instruction
    - Can be complicated by unrolled loops
- Run-ahead prefetching
  - Proposed by Jim Dundas
  - What to do when the processor is idle (or has idle issue slots) waiting for data
  - Allow the processor core to continue executing those instructions which can execute, ignoring those which can-not, and issuing prefetches when load operations are encountered
  - No specific prefetch instructions in ISA - std load instructions in "run-ahead" mode
  - Requires significant hardware augmentation - control & commit



- desired data is muxed out using column address bits

## Wednesday 3/28 - Class # 24

**READ : Chapter 5 of Betty Prince text - on Reserve in Library - by Mon 4/2**

### • No class on Monday 4/9 - Optional Make-up class?

- Memory Modules (SLIDE from BREY)
  - 30-pin SIMM (9-wide data)
  - 72-pin SIMM (32/32 - wide data)
  - 168-pin DIMM (64/72/80 - wide data)
    - Keyed to prevent over-voltage destruction - but not correct operation
    - DIAGRAM - keying in 168pin DIMMs
    - Two sided pins in DIMM - both sides not identical
  - Many other kinds of SIMM/DIMM - but these are the most common
  - 168 Pin DIMM Spec has limit on number of “words/lines” of 64M
  - Std/Parity/ECC bits per SIMM
  - Bus-width vs xIMM coverage - when xIMMs need to be installed in multiples
- Conventional DRAM
  - DIAGRAM : device architecture (latches & such)
  - DIAGRAM : Access pattern (i.e. timing diagram - RAS/CAS/ADDR/DATA)
  - There are other signals in the interface, but these are sufficient to understand what’s happening
  - Multiplexed address
  - Memory array requires only 1/2 of address at a time
  - Row Address (Strobe)
  - Column Address (Strobe)
- FPM DRAM
  - Fast Page mode DRAM
  - eliminates the Row Strobe requirement when Row is constant
  - Can ONLY perform FPM accesses between two temporally adjacent accesses to the same DRAM row
  - DIAGRAM : timing digram
  - First DRAM “controller policy” - open page vs. close page - in which state to leave the most recently accessed page?
  - Only difference to the DRAM device is in the on-chip control

## Friday 3/29 - Class # 25

- EDO DRAM

- EDO (Extended Data Out) DRAM
- Adds a latch which allows for faster cycle times
- Allows memory array precharge & output to be overlapped (pipelined)
- DIAGRAM : timing diagram
- BEDO DRAM
  - Burst EDO DRAM
  - Adds control to enable multiple FPM data output cycles from a single “col” address
  - DIAGRAM : timing diagram
  - Reduces the traffic on the Addr portion of the bus - thus slightly reducing power consumption, and easing controller design, but does not improve throughput or performance

- **More Memory Modules**

## Monday 4/2 - Class # 30

- **Interleaving - here???**

- Memory system organization issue
- Think in the mindset of a long-latency, low data-bus occupancy
- Allows simultaneous accesses to non-Synchronous long-latency memory devices
- Increases bus throughput and adds ability for pipelined accesses
- DIAGRAM : Basic Interleaving
  - Requires additional control signals for Address Latches / Data Enables
  - Common with high-performance pre-SDRAM memory systems
  - In addition to increasing throughput, can perform multiple requests
  - Interleaving factor is a substantial parameter to performance - need not be  $2^n$
  - Can only perform parallel accesses to unique banks
  - Low-order interleaving - use low order bits to select bank - can provide performance improvements
  - High-order interleaving - use high-order bit to select bank - typically provides less performance improvements, but this is typically done by default in those memory systems utilizing multiple DIMM slots
  - Commonly used in Vector computers - which require a unique data element each cycle after the vector operation has begun
- DIAGRAM : CRAY X-MP-2 Memory Organization
  - Multi-level/Multi-port interleaving
  - Allows multiple porting and interleaving / high throughput
  - Much higher level of control complexity
- Interleaved functionality is essentially being replicated internally by modern synchronous devices

- Very uncommon to see interleaved organization with Synchronous interface devices

- **Discussions of Address Remapping, Synchronous DRAM bandwidths, etc..**

## **Wednesday 4/4 - Class # 31**

- **DRAM - continued**

- SDRAM
  - All prior DRAM are Asynchronous - no clock in the interface
  - All Subsequent DRAM are Synchronous - clock in the interface
  - Unfortunately “SDRAM” has come to refer to a specific type of SDRAM
  - Advantages to a synchronous interface - only a single signal (clock) with sharp edges, i.e. high power drivers
  - Reduces the complexity of the timing constraints (set-up & hold times)
  - Allows for highly pipelined accesses - higher throughput
  - Downside - accesses must synchronize to a bus cycle before being issued to the device
- PC66/PC100/PC133 SDRAM
  - DIAGRAM : Timing diagram
  - Latency Timings a-b-c
    - a : CL : CAS latency
    - b : Trcd : RAS-to-CAS delay
    - c : Trp : precharge time
    - 3-3-2 and 2-2-2 are common, but not the only possible
  - Burst Modes are supported
  - Same device can be run at many different clock speeds
  - Same device can be setup to function with multiple CL clock latencies (2 or 3)
  - How to read Data Sheets?
  - Multiple Banks per device - not strictly a SDRAM issue, but most helpful when mated with the SDRAM interface
    - Limited by the number of banks for which the controller can maintain state

## **Monday 4/9 - Class # 32**

- DDR1 SDRAM
  - Address & Control signals are unchanged from PC133 SDRAM - only timing & interaccess constraints
  - Data signals transition at twice the clock frequency (i.e. latched by both clock edges)
  - Differential clock signals - one more clock pin, latched at sharper edge -typically falling
  - Twice the data throughput
  - Naming Convention - PC2100 - not 2100 MHz - 2.1 GB/s throughput @ 133MHz

- Thought to be capable of 166-200MHz clock frequency max, pushing theoretical limit with current signalling technology & bus/socket/DIMM capacitances
- DIAGRAM : Timing Diagram
- Higher power consumption - justified by bandwidth
- DDR2 SDRAM
  - Targetted at achieving frequencies starting at 200MHz, with better bus utilization than DDR1
  - Signalling level is lower (SSTL) - allowing for higher switching frequencies - with lower power consumption
  - WL = RL-1
    - Recall distinction between burst SRAM, late-write SRAM & Zero-Bus-Turnaround SRAM
    - DRAM bus can not have ZBT - especially at high frequencies, but one cycle is doable
    - Based upon a 1-cycle "bus-turnaround" latency
    - allows for more efficiently pipelined access requests

### **Wednesday 4/11 - Class # 33**

- Fixed Burst-Size of 4 bus-widths
  - Increased activity on the Address Bus
  - Reduced complexity in the controller and DRAM devices
- Refresh Discussed - Impromptu
  - Many different approaches to Refresh
  - Standby : Auto-refresh - requires time to enter & exit, only useful for computer shutdown mode
  - Shared refresh
    - All rows at once
    - time interspersed
      - rows / refresh time
      - most common approach - best performance w/ simplest controller implementation
    - grouped
      - impacts latency to microprocessor substantially
  - Individual refresh
    - High
  - Pathologic refresh-not-necessary cases
    - Video card frame buffer

### **Friday 4/13 - Class # 34**

- DDR2 SDRAM - Continued
  - Posted-CAS

- Places a delay between the CAS and the actual issuing of the CAS to the core
- Only useful in conjunction with
- DIAGRAM : Timing Diagram(s)
- Current termination of the “Evolutionary approaches”
  - Why is Evolution preferred - less technical & monetary risk
  - Examine the problems Intel has had with thier Rambus Chipsets
- ESDRAM (Enhanced SyncDRAM)
  - ESDRAM adds a full row of cache between the sense amps and the column decoder. This is not to be confused with EDO which adds a few bits of latch, enough to cover the data output between column decoder and device boundary
  - DIAGRAM : Architectural

## Monday 4/16 - Class # 35

- Essentially, places a small amount of cache on the DRAM die
- ALWAYS enables precharge to be hidden - i.e. not affect latency
- The advantages of both Open-Page and Close-Page-Autoprecharge in a single device/ architecture
- Sponsored by “Enhanced Memory System” - Currently being produced, small volumes
- Adopted as an enhancement to the DDR2 standard
- DIAGRAM - CAS Latency only, even when performing inner-page accesses
- Adds a new mode called “No-Write-Transfer” - which is a write to the array without disturbing the contents of the “row cache” or latches following the sense-amps
- Controller issues
  - OP / CPA issue is eliminated - always use CPA - use the “row caches” to handle OP hits
  - New controller issue - on write, use No-Write-Transfer, or Write-Transfer?
- VC (Virtual Channel) DRAM
  - A different approach for putting cache on the DRAM die
  - More associative in concept than ESDRAM
  - DIAGRAM : Architectural
  - Currently supported by some mainstream chipsets - but requires significantly more controller complexity
    - Controller must have tag storage for 16 shorter & associative (i.e. more tag bits) cache lines
    - Controller must have complexity to handle more tags, as well as management of on-chip caches or “virtual channels”
  - Controller issues
    - Channel Allocation - Bus Master, LRU, ???
    - Channel Writeback

## Wednesday 4/18 - Class # 36

- Conceptual Advantages
  - Virtual Channels could be assigned based upon bus master
  - Flipping between pages in the same bank does not eliminate the open-page advantage
  - Faster CL because of shorter cache lines in close proximity to external bus
- DIMMs carry significant price overhead - but are available
- ESDRAM vs. VC
  - Neither are tied to a interface architecture - both have been applied to PC100, and both are proposed for application to DDR2
  - Sequential accesses - EMS wins due to larger “cache” lines
  - Random conflicting accesses - VC wins due to higher associativity
  - Controller is more complex for VC - less complex for EMS
- Rambus (Gen 1)
  - Not the “Rambus” you are used to hearing about
  - For All Rambus, and DRAM for the most part, the changes are in the interface, not the core array
  - Used in ORIGINAL Nintendo
  - Interface
    - Shows the primary motivation of the Rambus line - signal pins
    - Double Data Rate - all signals - only family (Rambus) of devices to use DDR for address & control
    - 13 signal pins TOTAL - plus power/ground/Vref
    - 9 signal pins in the interface, shared between address & data
    - Address & Data signals are time multiplexed
    - All information transmitted in 8-transition “octcycle” packets - must be alligned
    - Clocks are send in both directions
      - such that outgoing data sync’s with the outgoing clock, and incoming data sync’s with the incoming clock
    - DIAGRAM : Clock distribution

## Friday 4/20 - Class # 37

- A single chip covers the entire data bus
  - more *effective* smaller sense-amp rows (i.e. cache lines) for a given memory space
  - Possible because of the reduced # of pins in the interface - would be pad/package expensive in a SDRAM device
  - DIAGRAM : Memory Space / chip coverage RDRAM vs SDRAM

- Architecture
  - The Gen 1 Rambus device is an older technology design : 4Mbit / 2 banks
- HUGE departure for experienced designers - everything is new
  - Increased design time, likelihood of errors (Intel 820) and additional non-recurring costs (testers, etc)
- Direct Rambus
  - Rambus Gen 3 - after the Intel input has been integrated
    - Gen 2 was “Concurrent Rambus” - it never left the drawingboard
  - Used in Nintendo64 & Desktop PC’s
  - Intel’s support of both standard, and company are contingent upon modifications
  - If volumes reach a mark, and Intel chipset usage reaches a mark, RMBS pays-off INTL
  - Interface
    - 18 Data wires ; 3 Row wires ; 5 Column wires - HIGH bus concurrency
    - All signals are double data rate
    - Refresh is handled on Row wires
    - 32/34 signals total
    - Will run at 300, 350, 400, 533 MHz
    - Clocks run in 2 directions like base Rambus
    - Information still transmitted in “Octcycle” packets - but they no longer need to be aligned
    - DIAGRAM - Packet formats
    - High levels of concurrency - 3 transactions in parallel
    - Lower latency than previous Rambus - non aligned octcycles - but still subject to packetized interface penalties

## Monday 4/23 - Class #38

- Power = freq \* capacitance \* Voltage<sup>2</sup>
- Architecture
  - 128 Mbit / 32 banks
  - Sense-amps are shared accross banks - saves space - sacrifices concurrency
  - DIAGRAM - Internal device Architectural
  - Each device covers entire system bus
  - Multiple devices on RIMM acts as multiple banks in SDRAM configuration - each device handles a transaction individually
  - DIAGRAM - System level architecture (RIMM’s, bus coverage, etc)
  - Yields larger # of smaller *effective* sense-amp rows - better cache utilization
- Timing

- DIAGRAM - timing, show non-aligned Octycles
- Clock rates of 300, 350, 400, 533 MHz mean bit-rates per signal of twice that
- End-to-end latency is higher than for fast SDRAM
- Bandwidth is comparable to fast SDRAM

- Bandwidth Chart

Device name	Dimm Name	Data Bus Signals	Interface Signals	64-Bit Bus BW
PC100	PC100	64 / 72 / 80	92 / 100 / 108	800 MB/s
PC133	PC133	64 / 72 / 80	92 / 100 / 108	1064 MB/s
PC150	PC150	64 / 72 / 80	92 / 100 / 108	1200 MB/s
PC166	PC166	64 / 72 / 80	92 / 100 / 108	1328 MB/s
DDR266	PC2100	64 / 72 / 80	92+	2128 MB/s
DDR300	PC2400	64 / 72 / 80		2400 MB/s
DDR333	PC2600	64 / 72 / 80		2664 MB/s
DDR2 - 200	???	64 / 72 / 80	92++	3200 MB/s
Rambus (250)	???	8 / 9	13	500 MB/s
Direct Rambus (400)	RIMM800	16 / 18	32 / 34	1600 MB/s
Direct Rambus (533)	RIMM1066	16/18	32/34	2132 MB/s

- Other DRAM Technologies:
  - FCDRAM - pipelined w/in the array
    - DIAGRAM - where are the latches
  - MBDRAM - many small banks - banks are disable-able for higher yields
    - DIAGRAM - banking architecture
  - Embedded DRAM
- Future Directions:
  - Advanced DRAM Technology (ADT) Group - High Volume by 2003
    - Intel, Micron, Infineon, etc..

## Wednesday 4/25 - Class # 39

### • EPROM - EEPROM & FLASH

- Non-volatile memories

- Maintain information even in absence of power
- Some device with this characteristic is necessary in all systems - bootstrap
- Distinction between OTP - one time programmable, and re-programmable
- Embedded App / BIOS / controller program
- Typically do not want to write frequently to NV memories
  - Finite # (1-1,000,000) of write cycles
  - High power consumption for write (as compared w/ SRAM or DRAM)
- Evolution of Read Only Memories
- ROM (masked Read-Only Memories)
  - data is permanently written during manufacture (OTP)
  - Not typically used as discrete component any longer - but may be a functional block on a processor or circuit
  - The Pentium FDIV bug was caused by an incorrect entry in an on-chip ROM
  - There are common regular approaches to generating ROM, NOR array, NAND array, and such
    - but these are VERY dated approaches
  - More modern approach is no different than a random logic synthesis - but the usage of ROM is limited in modern applications
- PROM (Programmable Read Only Memories)
  - Manufactured in both Bipolar & CMOS technologies
  - Typically Fusible links - which can be blown
  - Fusible links are blown by passing excessive current through the link
  - Figure 3-5 : Sharma Pg. 88
  - PROM are less flexible (OTP), less expensive, and more tamper-proof than equivalent multiple-program devices.
  - When you attempt to program a PROM for the 2nd time - data will be corrupted

## **Friday 4/27 - Class # 40**

- EPROM (uv Erasable Programmable Read Only Memories)
  - Developed in 1970 by Frohman-Bentchkowsky
  - Ceramic package w/ quartz window for UV access - expensive
  - Uses a “floating gate”
    - Electrons are transported to the “floating gate” by Fowler-Nordheim tunnelling
    - This gate can be “biased” by putting a charge on the floating gate
    - The device non-volatility relies upon no leakage from the “floating gate”
    - Requires a HIGH voltage for programming (12-25V)
    - Figure 3-11 : Sharma Pg 93

- Floating gate is common to EPROM / EEPROM / FLASH - methods for putting charge onto floating gate differ
- Finite # of erases
- Data retention can be a problem for UV-EPROM
- Threshold Voltage ( $V_t$ ) is changed via the floating gate, and thus can make the gate open or closed for a known gate voltage
- EEPROM (Electrically Erasable Programmable Read Only Memories)
  - Fabricated in:
    - Metal-Nitride-Oxide-Silicon (MNOS) memories
    - Silicon-Oxide-Nitride-Oxide Semiconductor (SONOS) memories
    - Floating-Gate Tunneling Oxide (FLOTOX) technology
  - EEPROM purchased today are most likely to be FLOTOX devices
  - Requires a 12V supply for programming
  - Allow individual byte erasure
  - EEPROM is still available - but has been almost completely superseded by FLASH
- FLASH
  - Flash could be considered A “electrically erasable programmable read only memory”
  - However, the interface and voltage supplies required are different from the EEPROM standard
  - Contents of all memory array cells can be erased simultaneously
  - Can be programmed from only a 5V supply
  - Limited Endurance - ~100,000 write cycles
  - Slower write times than SRAM/DRAM - comparable read times
  - 1.4X denser than DRAM (for same design rules)
  - 2X more expensive per bit than DRAM
  - Based upon same “floating gate” basic design as EPROM or EEPROM

## Monday 4/30/01 - Class # 41

- ERASE (write 0)
  - Places electrons onto floating gate via Fowler-Nordheim tunnelling
  - ~40mS
  - Performed on a large-block basis
- PROGRAM (write 1)
  - Removes electrons from floating gate
  - ~15microS
  - Performed on a cell-by-cell selected basis
- READ

- power to the gate, no voltage on drain - threshold voltage is  $>$  gate voltage : read 0
- power to gate, voltage on drain - threshold voltage is  $<$  gate voltage : read 1
- Reliability
  - Finite # of write cycles - most manufacturers claim  $\sim 100,000$  - probably more
  - Retention - electrons stay in floating gate for  $\sim 10$  years
- Multi-level FLASH
  - Intel StrataFLASH multi-level (multi threshold) flash circuit
  - ADV : density, price
  - DIS : increased read time, 5% area penalty for controller, significantly more complex

## • MGRAM

- Magnetic Ram predates Static/Dynamic - but at a discrete component level - core
- The current MGRAM initiatives are trying to move the same technologies into a fabricated integrated circuit
- Uses Magnetic rather than electrical charges to store bits
- ADV:
  - Speed of SRAM
  - Density of DRAM
  - Non-volatility of Flash
- Many companies are pursuing MGRAM : IBM, Infineon, Motorola, Intel, Honeywell, HP, Toshiba, Siemens & Bosch
- Specifics of how the bits are stored are not the same between proponents
- IBM has patents on a Magnetic Tunnel Junction device - which could be used in a MGRAM

**Wednesday 5/2/2001 - Last Class**

## **Final Exam Review**