Main Memory

• Main memory generally utilizes Dynamic RAM (DRAM), which use a single transistor to store a bit, but require a periodic data refresh by reading every row (~every 8 msec).

• Static RAM may be used if the added expense, low density, power consumption, and complexity is feasible (e.g. Cray Vector Supercomputers)

• Main memory performance is affected by:
  – Memory latency: Affects cache miss penalty. Measured by:
    • Access time: The time it takes between a memory access request is issued to main memory and the time the requested information is available to cache/CPU.
    • Cycle time: The minimum time between requests to memory (greater than access time in DRAM to allow address lines to be stable)
  – Memory bandwidth: The sustained data transfer rate between main memory and cache/CPU.
Logical DRAM Organization (4 Mbit)

- Square root of bits per RAS/CAS

1. Address Buffer
2. Row Decoder
3. Column Decoder
4. Sense Amps & I/O
5. Memory Array (2,048 x 2,048)
6. Data In
7. Data Out
8. Word Line
9. Bit Line
10. Storage Cell
Logical Diagram of A Typical DRAM

Control Signals (RAS_L, CAS_L, WE_L, OE_L) are all active low

Din and Dout are combined (D):
- WE_L is asserted (Low), OE_L is disasserted (High)
  - D serves as the data input pin
- WE_L is disasserted (High), OE_L is asserted (Low)
  - D is the data output pin

Row and column addresses share the same pins (A)
- RAS_L goes low: Pins A are latched in as row address
- CAS_L goes low: Pins A are latched in as column address
Four Key DRAM Timing Parameters

- **t\textsubscript{RAC}:** Minimum time from RAS (Row Access Strobe) line falling to the valid data output.
  - Usually quoted as the nominal speed of a DRAM chip
  - For a typical 4Mb DRAM $t\textsubscript{RAC} = 60$ ns

- **t\textsubscript{RC}:** Minimum time from the start of one row access to the start of the next.
  - $t\textsubscript{RC} = 110$ ns for a 4Mbit DRAM with a $t\textsubscript{RAC}$ of 60 ns

- **t\textsubscript{CAC}:** Minimum time from CAS (Column Access Strobe) line falling to valid data output.
  - 15 ns for a 4Mbit DRAM with a $t\textsubscript{RAC}$ of 60 ns

- **t\textsubscript{PC}:** Minimum time from the start of one column access to the start of the next.
  - About 35 ns for a 4Mbit DRAM with a $t\textsubscript{RAC}$ of 60 ns
DRAM Performance

- A 60 ns ($t_{RAC}$) DRAM chip can:
  - Perform a row access only every 110 ns ($t_{RC}$)
  - Perform column access ($t_{CAC}$) in 15 ns, but time between column accesses is at least 35 ns ($t_{PC}$).
    - In practice, external address delays and turning around buses make it 40 to 50 ns
- These times do not include the time to drive the addresses off the CPU or the memory controller overhead.
Every DRAM access begins at:

- The assertion of the RAS_L
- 2 ways to write: early or late v. CAS

**DRAM WR Cycle Time**

Early Wr Cycle: WE_L asserted before CAS_L

Late Wr Cycle: WE_L asserted after CAS_L
Every DRAM access begins at:
- The assertion of the RAS_L
- 2 ways to read: early or late v. CAS

Early Read Cycle: OE_L asserted before CAS_L
Late Read Cycle: OE_L asserted after CAS_L
Simplified Asynchronous DRAM Read Timing

Source: http://arstechnica.com/paedia/r/ram_guide/ram_guide.part2-1.html
Page Mode DRAM: Motivation

° 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 “register” to save a row
Page Mode DRAM: Operation

° 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
Simplified Asynchronous Fast Page Mode (FPM) DRAM Read Timing

Typical timing at 66 MHZ: 5-3-3-3
For bus width = 64 bits = 8 bytes cache block size = 32 bytes
It takes = 5+3+3+3 = 14 memory cycles or 15 ns x 14 = 210 ns to read 32 byte block
Read Miss penalty for CPU running at 1 GHZ = 15 x 14 = 210 CPU cycles
Simplified Asynchronous Extended Data Out (EDO) DRAM Read Timing

- Extended Data Out DRAM operates in a similar fashion to Fast Page Mode DRAM except the data from one read is on the output pins at the same time the column address for the next read is being latched in.

**EDO Read**

**EDO DRAM speed rated using tRAC ~ 40-60ns**

Simplified Asynchronous Extended Data Out (EDO) DRAM Read Timing

- Typical timing at 66 MHZ: 5-2-2-2
- For bus width = 64 bits = 8 bytes  Max. Bandwidth = 8 x 66 / 2 = 264 Mbytes/sec
- It takes = 5+2+2+2 = 11 memory cycles or 15 ns x 11 = 165 ns to read 32 byte cache block
- Read Miss penalty for CPU running at 1 GHZ = 11 x 15 = 165 CPU cycles

**Source:** http://arstechnica.com/paedia/r/ram_guide/ram_guide.part2-1.html
## Synchronous DRAM Interface Characteristics Summary

<table>
<thead>
<tr>
<th></th>
<th>PC100</th>
<th>DDR266 (PC2100)</th>
<th>DDR2</th>
<th>DRDRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Potential Bandwidth</td>
<td>0.8 GB/s</td>
<td>2.133 GB/s</td>
<td>3.2 GB/s</td>
<td>1.6 GB/s</td>
</tr>
<tr>
<td>Interface Signals</td>
<td>64(72) data 168 pins</td>
<td>64(72) data 168 pins</td>
<td>64(72) data 184 pins</td>
<td>16(18) data 184 pins</td>
</tr>
<tr>
<td>Interface Frequency</td>
<td>100 MHz</td>
<td>133 MHz</td>
<td>200 MHz</td>
<td>400 MHz</td>
</tr>
<tr>
<td>Latency Range</td>
<td>30-90 nS</td>
<td>18.8-64 nS</td>
<td>17.5-42.6 nS</td>
<td>35-80 nS</td>
</tr>
</tbody>
</table>
Synchronous Dynamic RAM, SDRAM Organization

Diagram of Synchronous Dynamic RAM Organization
Simplified SDRAM Read Timing

Typical timing at 133 MHZ (PC133 SDRAM) : 4-1-1-1
For bus width = 64 bits = 8 bytes     Max. Bandwidth = 133 x 8 = 1064 Mbytes/sec
It takes = 4+1+1+1 = 8 memory cycles or 7.5 ns x 8 = 60 ns to read 32 byte cache block
Read Miss penalty for CPU running at 1 GHZ = 7.5 x 8 = 60 CPU cycles
# DRAM “Near” Future: 1 Gbit Chips

<table>
<thead>
<tr>
<th></th>
<th>Mitsubishi</th>
<th>Samsung</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blocks</td>
<td>512 x 2 Mbit</td>
<td>1024 x 1 Mbit</td>
</tr>
<tr>
<td>Clock</td>
<td>200 MHz</td>
<td>250 MHz</td>
</tr>
<tr>
<td>Data Pins</td>
<td>64</td>
<td>16</td>
</tr>
<tr>
<td>Die Size</td>
<td>24 x 24 mm</td>
<td>31 x 21 mm</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>– Sizes will be much smaller in production</td>
<td></td>
</tr>
<tr>
<td>Metal Layers</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Technology</td>
<td>0.15 micron</td>
<td>0.16 micron</td>
</tr>
</tbody>
</table>

Mitsubishi and Samsung are two companies that are currently working on developing 1 Gbit chips. The table above compares some of the specifications of these chips. Mitsubishi’s chip has a block size of 512 x 2 Mbit, a clock speed of 200 MHz, 64 data pins, and a die size of 24 x 24 mm. Samsung’s chip has a block size of 1024 x 1 Mbit, a clock speed of 250 MHz, 16 data pins, and a die size of 31 x 21 mm. Mitsubishi’s chip has three metal layers, while Samsung’s chip has four. Mitsubishi uses 0.15 micron technology, while Samsung uses 0.16 micron technology. Both companies are expected to produce much smaller chips in the future.
Memory Bandwidth Improvement Techniques

- **Wider Main Memory:**
  
  Memory width is increased to a number of words (usually the size of a cache block).
  
  \[ \Rightarrow \text{Memory bandwidth is proportional to memory width.} \]
  
  e.g. Doubling the width of cache and memory doubles memory bandwidth

- **Simple Interleaved Memory:**
  
  Memory is organized as a number of banks each one word wide.
  
  - Simultaneous multiple word memory reads or writes are accomplished by sending memory addresses to several memory banks at once.
  
  - Interleaving factor: Refers to the mapping of memory addressees to memory banks.
    
    e.g. using 4 banks, bank 0 has all words whose address is:
    
    \[ \text{(word address mod} 4) = 0 \]
Three examples of bus width, memory width, and memory interleaving to achieve higher memory bandwidth

Simplest design: Everything is the width of one word

Wider memory, bus and cache

Narrow bus and cache with interleaved memory
Memory Interleaving

Access Pattern without Interleaving:

- D1 available
- Start Access for D1
- Start Access for D2

Access Pattern with 4-way Interleaving:

- Access Bank 0
- Access Bank 1
- Access Bank 2
- Access Bank 3

We can Access Bank 0 again

EECC551 - Shaaban
### Four way interleaved memory

<table>
<thead>
<tr>
<th>Address</th>
<th>Bank 0</th>
<th>Address</th>
<th>Bank 1</th>
<th>Address</th>
<th>Bank 2</th>
<th>Address</th>
<th>Bank 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>1</td>
<td></td>
<td>2</td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>5</td>
<td></td>
<td>6</td>
<td></td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>9</td>
<td></td>
<td>10</td>
<td></td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td>13</td>
<td></td>
<td>14</td>
<td></td>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>

Three memory banks address interleaving:
- Sequentially interleaved addresses on the left, address requires a division
- Right: Alternate interleaving requires only modulo to a power of 2
Memory Width, Interleaving: An Example

Given the following system parameters with single cache level $L_1$:

- Block size = 1 word
- Memory bus width = 1 word
- Miss rate = 3%
- Miss penalty = 32 cycles
  (4 cycles to send address, 24 cycles access time/word, 4 cycles to send a word)
- Memory access/instruction = 1.2
- Ideal CPI (ignoring cache misses) = 2
- Miss rate (block size = 2 word) = 2%
- Miss rate (block size = 4 words) = 1%

- The CPI of the base machine with 1-word blocks = $2 + (1.2 \times 0.03 \times 32) = 3.15$
- Increasing the block size to two words gives the following CPI:
  - 32-bit bus and memory, no interleaving = $2 + (1.2 \times 0.02 \times 2 \times 32) = 3.54$
  - 32-bit bus and memory, interleaved = $2 + (1.2 \times 0.02 \times (4 + 24 + 8)) = 2.86$
  - 64-bit bus and memory, no interleaving = $2 + (1.2 \times 0.02 \times 1 \times 32) = 2.77$

- Increasing the block size to four words; resulting CPI:
  - 32-bit bus and memory, no interleaving = $2 + (1.2 \times 0.01 \times 4 \times 32) = 3.54$
  - 32-bit bus and memory, interleaved = $2 + (1.2 \times 0.01 \times (4 + 24 + 16)) = 2.53$
  - 64-bit bus and memory, no interleaving = $2 + (1.2 \times 0.01 \times 2 \times 32) = 2.77$
Computer System Components

CPU Core
500 MHZ - 2.0 GHZ
4-way Superscaler
RISC or RISC-core (x86):
  Deep Instruction Pipelines
  Dynamic scheduling
  Multiple FP, integer FUs
  Dynamic branch prediction
  Hardware speculation

SDRAM
PC100/PC133
100-133MHZ
64-128 bits wide
2-way interleaved
~ 900 MBYTES/SEC (64bit)

Double Data Rate (DDR) SDRAM
PC2100
266MHZ
64-128 bits wide
4-way interleaved
~2.1 GBYTES/SEC (64bit)

RAMbus DRAM (RDRAM)
400-800MHZ
16 bits wide
~ 1.6 GBYTES/SEC

CPU

Caches

Memory

System Bus

L1
L2
L3

Memory Controller

Controllers

NICs

Disks
Displays
Keyboards

I/O Devices:

Examples:
Alpha, AMD K7: EV6, 200-266MHZ
Intel PII, PIII: GTL+ 100MHZ
Intel P4 400MHZ

Networks

I/O Buses

Example: PCI, 33MHZ
32 bits wide
133 MBYTES/SEC

EECC551 - Shaaban
X86 CPU Cache/Memory Performance Example

AMD Athlon T-Bird Vs. Intel PIII

AMD Athlon T-Bird 1GHz
- L1: 64K INST, 64K DATA (3 cycle latency), both 2-way
- L2: 256K 16-way 64 bit
  Latency: 7 cycles
  L1,L2 on-chip

Intel PIII GHZ
- L1: 16K INST, 16K DATA (3 cycle latency), both 4-way
- L2: 256K 8-way 256 bit, Latency: 7 cycles
  L1,L2 on-chip

Memory:
- PC2100
  133MHz DDR SDRAM 64bit
  Peak bandwidth: 2100 MB/s

- PC133
  133MHz SDRAM 64bit
  Peak bandwidth: 1000 MB/s

- PC800
  Rambus DRDRAM
  400 MHz DDR 16-bit
  Peak bandwidth: 1600 MB/s


Intel 840 uses two PC800 channels
X86 CPU Cache/Memory Performance Example:
AMD Athlon T-Bird Vs. Intel PIII

This Linpack data size range causes L2 misses and relies on main memory

X86 CPU Cache/Memory Performance Example:
AMD Athlon T-Bird Vs. Intel PIII, Vs. P4

AMD Athlon T-Bird 1GHZ
L1: 64K INST, 64K DATA (3 cycle latency), both 2-way
L2: 256K 16-way 64 bit
Latency: 7 cycles
L1,L2 on-chip

Intel P4, 1.5 GHZ
L1: 8K INST, 8K DATA (2 cycle latency)
both 4-way
96KB Execution Trace Cache
L2: 256K 8-way 256 bit, Latency: 7 cycles
L1,L2 on-chip

Intel PIII 1 GHZ
L1: 16K INST, 16K DATA (3 cycle latency)
both 4-way
L2: 256K 8-way 256 bit, Latency: 7 cycles
L1,L2 on-chip

X86 CPU Cache/Memory Performance Example: AMD Athlon T-Bird Vs. Duron

AMD Athlon T-Bird
750MHZ-1GHZ
L1: 64K INST, 64K DATA,
both 2-way
L2: 256K 16-way 64 bit
  Latency: 7 cycles
  L1,L2 on-chip

Memory:
PC2100
133MHZ DDR SDRAM 64bit
  Peak bandwidth: 2100 MB/s

PC1600
100MHZ DDR SDRAM 64bit
  Peak bandwidth: 1600 MB/s

AMD Athlon Duron
750MHZ-1GHZ
L1: 64K INST, 64K DATA
  both 2-way
L2: 64K 16-way 64 bit
  Latency: 7 cycles
  L1,L2 on-chip

Source: http://www1.anandtech.com/showdoc.html?i=1345&p=10
A Typical Memory Hierarchy

- **Processor**
  - Control
  - Datapath
  - Registers
  - On-Chip Level
  - One Cache
  - $L_1$
  - Second Level Cache
  - (SRAM) $L_2$
  - Main Memory (DRAM)
  - Virtual Memory, Secondary Storage (Disk)
  - Tertiary Storage (Tape)

- **Speed (ns):**
  - 1s
  - 10s
  - 100s
  - 10,000,000s (10s ms)
  - 10,000,000,000s (10s sec)

- **Size (bytes):**
  - 100s
  - Ks
  - Ms
  - Gs
  - Ts

**Faster**

**Larger Capacity**
Virtual Memory

- Virtual memory controls two levels of the memory hierarchy:
  - Main memory (DRAM).
  - Mass storage (usually magnetic disks).
- Main memory is divided into blocks allocated to different running processes in the system:
  - Fixed size blocks: Pages (size 4k to 64k bytes).
  - Variable size blocks: Segments (largest size 216 up to 232).
- At any given time, for any running process, a portion of its data/code is loaded in main memory while the rest is available only in mass storage.
- A program code/data block needed for process execution and not present in main memory result in a page fault (address fault) and the block has to be loaded into main main memory from disk.
- A program can be run in any location in main memory or disk by using a relocation mechanism controlled by the operating system which maps the address from virtual address space (logical program address) to physical address space (main memory, disk).
Virtual Memory

Benefits

- Illusion of having more physical main memory
- Allows program relocation
- Protection from illegal memory access

Virtual address

31 30 29 28 27 15 14 13 12 11 10 9 8 3 2 1 0

Virtual page number          Page offset

Translation

29 28 27 15 14 13 12 11 10 9 8 3 2 1 0

Physical page number          Page offset

Physical address
Paging Versus Segmentation

<table>
<thead>
<tr>
<th></th>
<th>Page</th>
<th>Segment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Words per address</td>
<td>One</td>
<td>Two (segment and offset)</td>
</tr>
<tr>
<td>Programmer visible?</td>
<td>Invisible to application programmer</td>
<td>May be visible to application programmer</td>
</tr>
<tr>
<td>Replacing a block</td>
<td>Trivial (all blocks are the same size)</td>
<td>Hard (must find contiguous, variable-size, unused portion of main memory)</td>
</tr>
<tr>
<td>Memory use inefficiency</td>
<td>Internal fragmentation (unused portion of page)</td>
<td>External fragmentation (unused pieces of main memory)</td>
</tr>
<tr>
<td>Efficient disk traffic</td>
<td>Yes (adjust page size to balance access time and transfer time)</td>
<td>Not always (small segments may transfer just a few bytes)</td>
</tr>
</tbody>
</table>
Virtual → Physical Address Translation

Contiguous virtual address space of a program

Physical location of blocks A, B, C
Mapping Virtual Addresses to Physical Addresses Using A Page Table

Virtual address

Virtual page number  Page offset

Page table

Physical address

Main memory
Virtual Address Translation

Virtual page number

Page table
Physical page or disk address

Valid

Physical memory

Disk storage

EECC551 - Shaaban
Two memory accesses needed:
• First to page table.
• Second to item.
## Typical Parameter Range For Cache & Virtual Memory

<table>
<thead>
<tr>
<th>Parameter</th>
<th>First-level cache</th>
<th>Virtual memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block (page) size</td>
<td>16–128 bytes</td>
<td>4096–65,536 bytes</td>
</tr>
<tr>
<td>Hit time</td>
<td>1–2 clock cycles</td>
<td>40–100 clock cycles</td>
</tr>
<tr>
<td>Miss penalty</td>
<td>8–100 clock cycles</td>
<td>700,000–6,000,000 clock cycles</td>
</tr>
<tr>
<td>(Access time)</td>
<td>(6–60 clock cycles)</td>
<td>(500,000–4,000,000 clock cycles)</td>
</tr>
<tr>
<td>(Transfer time)</td>
<td>(2–40 clock cycles)</td>
<td>(200,000–2,000,000 clock cycles)</td>
</tr>
<tr>
<td>Miss rate</td>
<td>0.5–10%</td>
<td>0.00001–0.001%</td>
</tr>
<tr>
<td>Data memory size</td>
<td>0.016–1MB</td>
<td>16–8192 MB</td>
</tr>
</tbody>
</table>
Virtual Memory Issues/Strategies

- **Main memory block placement:** Fully associative placement is used to lower the miss rate.
- **Block replacement:** The least recently used (LRU) block is replaced when a new block is brought into main memory from disk.
- **Write strategy:** Write back is used and only those pages changed in main memory are written to disk (dirty bit scheme is used).
- To locate blocks in main memory a page table is utilized. The page table is indexed by the virtual page number and contains the physical address of the block.
  - In paging: Offset is concatenated to this physical page address.
  - In segmentation: Offset is added to the physical segment address.
- To limit the size of the page table to the number of physical pages in main memory a hashing scheme is used.
- Utilizing address locality, a translation look-aside buffer (TLB) is usually used to cache recent address translations and prevent a second memory access to read the page table.
Speeding Up Address Translation: Translation Lookaside Buffer (TLB)

- TLB: A small on-chip fully-associative cache used for address translations.
- If a virtual address is found in TLB (a TLB hit), the page table in main memory is not accessed.

![Diagram of TLB and page translation](image-url)
Data TLB During Address Translation

Virtual address

Valid

Read Permission

Write Permission

TLB = 32 blocks
Data cache = 256 blocks
TLB access is usually pipelined
**TLB & Cache Operation**

**Virtual address**

- **TLB access**
  - **TLB hit?**
    - **Yes**
    - **Physical address**
      - **Write?**
        - **Yes**
          - **Write data into cache**, update the tag, and put the data and the address into the write buffer
        - **No**
          - **Write access bit on?**
            - **Yes**
              - **Write protection exception**
            - **No**
              - **Cache hit?**
                - **Yes**
                  - **Deliver data to the CPU**
                - **No**
                  - **Cache miss stall**

- **No**
  - **TLB miss**
    - **use page table**

**Cache is physically-addressed**

**Cache operation**

- **Cache miss stall**
- **Cache hit?**
  - **Yes**
    - **Deliver data to the CPU**
  - **No**
    - **Try to read data from cache**
      - **Write?**
        - **Yes**
          - **Write data into cache**, update the tag, and put the data and the address into the write buffer
        - **No**
          - **Write access bit on?**
            - **Yes**
              - **Write protection exception**
            - **No**
              - **Cache hit?**
                - **Yes**
                  - **Deliver data to the CPU**
                - **No**
                  - **Cache miss stall**
CPU Performance with Real TLBs

When a real TLB is used with a TLB miss rate and a TLB miss penalty is used:

\[ CPI = CPI_{\text{execution}} + \text{mem stalls per instruction} + \text{TLB stalls per instruction} \]

Where:

Mem Stalls per instruction = Mem accesses per instruction \times mem stalls per access

Similarly:

TLB Stalls per instruction = Mem accesses per instruction \times TLB stalls per access

TLB stalls per access = TLB miss rate \times TLB miss penalty

Example:

Given: \( CPI_{\text{execution}} = 1.3 \) \quad \text{Mem accesses per instruction} = 1.4

Mem stalls per access = 0.5 \quad \text{TLB miss rate} = 0.3\% \quad \text{TLB miss penalty} = 30 \text{ cycles}

What is the resulting CPU CPI?

Mem Stalls per instruction = 1.4 \times 0.5 = 0.7 \text{ cycles/instruction}

TLB stalls per instruction = 1.4 \times (\text{TLB miss rate} \times \text{TLB miss penalty})

= 1.4 \times 0.003 \times 30 = 0.126 \text{ cycles/instruction}

CPI = 1.3 + 0.7 + 0.126 = 2.126
### Event Combinations of Cache, TLB, Virtual Memory

<table>
<thead>
<tr>
<th>Cache</th>
<th>TLB</th>
<th>Virtual Memory</th>
<th>Possible?</th>
<th>When?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Miss</td>
<td>Hit</td>
<td>Hit</td>
<td>Possible, no need to check page table</td>
<td>Tlb miss, found in page table</td>
</tr>
<tr>
<td>Hit</td>
<td>Miss</td>
<td>Hit</td>
<td>TLB miss, cache miss</td>
<td>Tlb miss, found in page table</td>
</tr>
<tr>
<td>Miss</td>
<td>Miss</td>
<td>Hit</td>
<td>TLB miss, cache miss</td>
<td>Tlb miss, found in page table</td>
</tr>
<tr>
<td>Miss</td>
<td>Miss</td>
<td>Miss</td>
<td>Page fault</td>
<td>Tlb miss, found in page table</td>
</tr>
<tr>
<td>Miss</td>
<td>Hit</td>
<td>Miss</td>
<td>Impossible, cannot be in TLB if not in memory</td>
<td>Tlb miss, found in page table</td>
</tr>
<tr>
<td>Hit</td>
<td>Hit</td>
<td>Miss</td>
<td>Impossible, cannot be in TLB or cache if not in memory</td>
<td>Tlb miss, found in page table</td>
</tr>
<tr>
<td>Hit</td>
<td>Miss</td>
<td>Miss</td>
<td>Impossible, cannot be in cache if not in memory</td>
<td>Tlb miss, found in page table</td>
</tr>
</tbody>
</table>