DRAM Technology

- Data stored as a charge in a capacitor
  - Single transistor used to access the charge
  - Must periodically be refreshed
    - Read contents and write back
    - Performed on a DRAM “row”
Advanced DRAM Organization

- Bits in a DRAM are organized as a rectangular array
  - DRAM accesses an entire row
  - Burst mode: supply successive words from a row with reduced latency
- Double data rate (DDR) DRAM
  - Transfer on rising and falling clock edges
- Quad data rate (QDR) DRAM
  - Separate DDR inputs and outputs
### DRAM Generations

<table>
<thead>
<tr>
<th>Year</th>
<th>Capacity</th>
<th>$/GB</th>
</tr>
</thead>
<tbody>
<tr>
<td>1980</td>
<td>64Kbit</td>
<td>$150000</td>
</tr>
<tr>
<td>1983</td>
<td>256Kbit</td>
<td>$500000</td>
</tr>
<tr>
<td>1985</td>
<td>1Mbit</td>
<td>$200000</td>
</tr>
<tr>
<td>1989</td>
<td>4Mbit</td>
<td>$50000</td>
</tr>
<tr>
<td>1992</td>
<td>16Mbit</td>
<td>$15000</td>
</tr>
<tr>
<td>1996</td>
<td>64Mbit</td>
<td>$10000</td>
</tr>
<tr>
<td>1998</td>
<td>128Mbit</td>
<td>$4000</td>
</tr>
<tr>
<td>2000</td>
<td>256Mbit</td>
<td>$1000</td>
</tr>
<tr>
<td>2004</td>
<td>512Mbit</td>
<td>$250</td>
</tr>
<tr>
<td>2007</td>
<td>1Gbit</td>
<td>$50</td>
</tr>
</tbody>
</table>

![Graph showing the trend of DRAM capacity and cost over years](image)
DRAM Performance Factors

- **Row buffer**
  - Allows several words to be read and refreshed in parallel

- **Synchronous DRAM**
  - Operates in Synchrony with external clock
  - Allows for consecutive accesses in bursts without needing to send each address
  - Improves bandwidth

- **DRAM banking**
  - Interleaves Memory Banks
  - Allows simultaneous access to multiple DRAMs
  - Improves bandwidth
Increasing Memory Bandwidth

- **4-word wide memory**
  - Miss penalty = 1 + 15 + 1 = 17 bus cycles
  - Bandwidth = 16 bytes / 17 cycles = 0.94 B/cycle

- **4-bank interleaved memory**
  - Miss penalty = 1 + 15 + 4×1 = 20 bus cycles
  - Bandwidth = 16 bytes / 20 cycles = 0.8 B/cycle
Interleaved Memory Banks (Διαφύλλωση Μνήμης)

Requests (Addresses)

(Optional Request Queues)

in case of Bank Conflicts

Mem. Bank0

Mem. Bank1

Mem. Bank2

Mem. Bank3

Data back

<table>
<thead>
<tr>
<th>Addr</th>
<th>00</th>
<th>10</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>E0</td>
<td>F0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Addr</th>
<th>04</th>
<th>14</th>
<th>24</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>E4</td>
<td>F4</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Addr</th>
<th>08</th>
<th>18</th>
<th>28</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>E8</td>
<td>F8</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Addr</th>
<th>0C</th>
<th>1C</th>
<th>2C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EC</td>
<td>FC</td>
<td></td>
</tr>
</tbody>
</table>

e.g.: 100 ns per Bank Access, one new Request every 25 ns

"Split Transactions" on request/reply "bus"... pipelining

...multiple transactions interleaved in time
"Split Transactions":
- Do not "hold" the "bus" exclusively between request and reply; release it for other transactions.
- Need Transaction ID with every request and its corresponding reply, especially in case of replied out-of-order.
Disk Storage

- Nonvolatile, rotating magnetic storage
Disk Sectors and Access

- Each sector records
  - Sector ID
  - Data (512 bytes, 4096 bytes proposed)
  - Error correcting code (ECC)
    - Used to hide defects and recording errors
  - Synchronization fields and gaps

- Access to a sector involves
  - Queuing delay if other accesses are pending
  - Seek: move the heads
  - Rotational latency
  - Data transfer
  - Controller overhead
Disk Access Example

- **Given**
  - 512B sector, 15,000rpm, 4ms average seek time, 100MB/s transfer rate, 0.2ms controller overhead, idle disk

- **Average read time**
  - 4ms seek time
  - $\frac{1}{2} \div (15,000/60) = 2\text{ms} \text{ rotational latency}$
  - $512 \div 100\text{MB/s} = 0.005\text{ms} \text{ transfer time}$
  - $0.2\text{ms} \text{ controller delay}$
  - $= 6.2\text{ms}$

- If actual average seek time is 1ms
  - Average read time = $3.2\text{ms}$
Disk Performance Issues

- Manufacturers quote average seek time
  - Based on all possible seeks
  - Locality and OS scheduling lead to smaller actual average seek times
- Smart disk controller allocate physical sectors on disk
  - Present logical sector interface to host
  - SCSI, ATA, SATA
- Disk drives include caches
  - Prefetch sectors in anticipation of access
  - Avoid seek and rotational delay
Amortize cost over large data blocks
Instruction Set Architecture for I/O

- Some machines have special input and output instructions

- Alternative model (used by MIPS):
  - Input: ~ reads a sequence of bytes
  - Output: ~ writes a sequence of bytes

- Memory also a sequence of bytes, so use loads for input, stores for output
  - Called “Memory Mapped Input/Output”
  - A portion of the address space dedicated to communication paths to Input or Output devices (no memory there)
Memory Mapped I/O

- Certain addresses are not regular memory

- Instead, they correspond to registers in I/O devices
Example: keyboard... if only a Data Register:

What did the user type?

<table>
<thead>
<tr>
<th>Time (s)</th>
<th>Read Data Reg.</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>= 't'</td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>= 't'</td>
<td></td>
</tr>
<tr>
<td>1.0</td>
<td>= 'o'</td>
<td></td>
</tr>
<tr>
<td>1.5</td>
<td>= 'o'</td>
<td></td>
</tr>
<tr>
<td>2.0</td>
<td>= 'o'</td>
<td></td>
</tr>
</tbody>
</table>
Example: keyboard... if only a Data Register:

What did the user type?

```
0.0 0.5 1.0 1.5 2.0
<table>
<thead>
<tr>
<th>read Data Reg.</th>
<th>read Data Reg.</th>
<th>read Data Reg.</th>
<th>read Data Reg.</th>
<th>read Data Reg.</th>
</tr>
</thead>
<tbody>
<tr>
<td>='t'</td>
<td>='t'</td>
<td>='o'</td>
<td>='o'</td>
<td>='o'</td>
</tr>
</tbody>
</table>
```

"to"

"too"

"two"

"ttoo00"

"tttoo"
Processor Checks Status before Acting

- Path to device generally has 2 registers:
  - 1 register says it’s OK to read/write (I/O ready), often called **Control Register**
  - 1 register that contains data, often called **Data Register**

- Processor reads from Control Register in loop, waiting for device to set Ready bit in Control reg to say its OK (0 → 1)

- Processor then loads from (input) or writes to (output) data register
  - Load from device/Store into Data Register resets Ready bit (1 → 0) of Control Register

* "Polling" "Busy wait" if done continuously; else, poll multiple devices on every interrupt from the real-time clock (usu. 50-120 Hz)
I/O Address Pages must be non-cacheable!

If they were allowed to be cached...

- Transitional ("non-coherent"...) caching does **NOT** work when other devices (I/O, other cores) access memory independently.

- Note: write-through is a "half-solution": works for output but not for input...
I/O/Communication Registers ≠ Normal Memory Semantics (non-shared)

Normal Memory:
- Write \( x \)
- Read \( x \times \) (potentially ≠)
- Successive reads from a same location (without any inter-reading writes from processor) may yield different values!

I/O/Communication Registers:
- Write here
- Other words NOT changed
- Data status
  - Read/write here may change other words too!

"Side-Effects"

Read always yields the last written value

Writes only affect the word being written
Memory Consistency

write: from input device or communication from other processor(s)

data 1

data 2

data 3

ready flag → 1

too fast? → old

interconnection network in-order or out-of-order delivery?

e.g. what if these reside on different memory banks in an interleaved memory?

reader:

wait to see the flag be come 1 →

then read data 2

old ??
What is the alternative to polling?

° Wasteful to have processor spend most of its time “spin-waiting” for I/O to be ready

° Wish we could have an unplanned procedure call that would be invoked only when I/O device is ready

° Solution: use exception mechanism to help I/O. Interrupt program when I/O ready, return when done with data transfer
I/O Interrupt

° An I/O interrupt is like an overflow exceptions except:
  • An I/O interrupt is “asynchronous”
  • More information needs to be conveyed

° An I/O interrupt is asynchronous with respect to instruction execution:
  • I/O interrupt is not associated with any instruction, but it can happen in the middle of any given instruction
  • I/O interrupt does not prevent any instruction from completion
Definitions for Clarification

° **Exception**: signal marking that something “out of the ordinary” has happened and needs to be handled

° **Interrupt**: asynchronous exception

° **Trap**: synchronous exception

° **Note**: These are different from the book’s definitions.
Interrupt Driven Data Transfer

1. I/O interrupt
2. save PC
3. interrupt service addr
4. read, store, etc.
5. jr

Memory

- add
- sub
- and
- or

User program

Interrupt service routine
Questions Raised about Interrupts

° Which I/O device caused exception?
  • Needs to convey the identity of the device generating the interrupt  Cause register, or Vectored Interrupts

° Can avoid interrupts during the interrupt routine?
  • What if more important interrupt occurs while servicing this interrupt?
  • Allow interrupt routine to be entered again?

° Who keeps track of status of all the devices, handle errors, know where to put/supply the I/O data?
Fast Devices need I/O **Buffer** — not just a **Register**

... Amortize the cost of interrupt over many data

**Example:**

(just) **1 Gbit/s**

↓

1 bit every 1 ns

↓

32 bits every 32 ns

one "Register"

- Cost to poll status register (non-cacheable, off-chip)
  usually ≥ DRAM access
  usually ~ 100 ns (or more)

- Then read the data register
  similarly ~ 100 ns

- Cost of interrupt + kernel interrupt handlers usually ~ 1 µs (1000 ns)!

**I/O Buffer**

<table>
<thead>
<tr>
<th>e.g.</th>
<th>4 kBy or more</th>
</tr>
</thead>
<tbody>
<tr>
<td>I/O</td>
<td>4 kBy</td>
</tr>
</tbody>
</table>

or "Double Buffering"

(fill one buffer by I/O, while servicing the other by the processor)

4 By every 32 ns

↓

4 kBy every 32 µs

but this may still be a problem if transfer done word-by-word by the processor (load-store loop) OK ratio
Direct Memory Access (DMA)

Alternatives for cacheability:
- DMA onto non-cacheable memory pages ... too slow when processor processes the I/O data
- Flush the cache before/after I/O DMA ... quite expensive operation (total flush?)
- Cache-Coherent DMA ← good! → next chapter...

"bus"

(write-through only solves half the problem)

Burst Transactions

copy, or copy →