SJSU SAN JOSÉ STATE UNIVERSITY

Charles W. Davidson College of Engineering

Department of Computer Engineering

Real-Time Embedded System Co-Design
CMPE 146 Section 1
Fall 2024

Memory

SRAM Basics

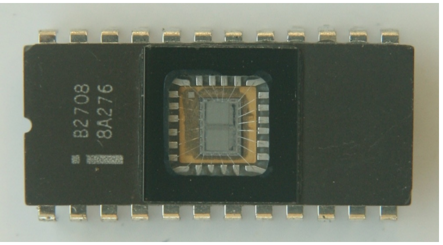
- SRAM stands for Static Random Access Memory
- Volatile
  - When power is gone, data are gone
- Why static?
  - Data are maintained without any kind of extra operations
  - As long as power to the device is maintained

DRAM Basics

- DRAM stands for Dynamic Random-Access Memory
- Volatile
  - When power is gone, data are gone
- Why dynamic?
  - Data are maintained with periodic refresh operations
  - Just maintaining power to the device is not sufficient
- Comparison to SRAM
  - DRAM cell is smaller
  - Less expensive per bit
  - SRAM is faster
  - DRAM requires more peripheral circuitry
- Hard to integrate with logic circuitry on chip
  - Fabrication process is different
  - Usually integrated off-chip

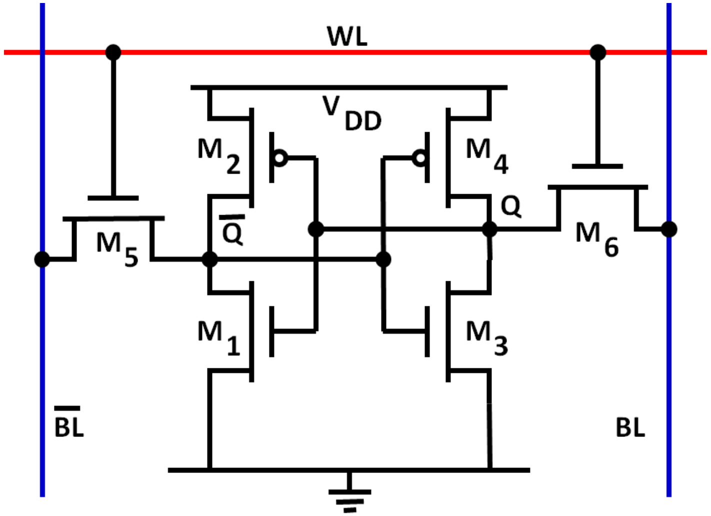
ROM Basics

- ROM stands for Read-Only Memory
- Non-volatile
  - When power is gone, stored data are still retained
- Some types are programmable and non-erasable
  - PROM (Programmable Read-Only Memory)
  - Once programmed, memory contents cannot be changed
- Some types are programmable and erasable
  - EPROM (Erasable Programmable Read-Only Memory)
    - Erasing and programming are done off board
    - Erase with ultra-violet light (in 15-20 minutes)
    - Program with high voltage (12 V)
  - EEPROM (Electrically Erasable Programmable Read-Only Memory)
    - Basically an older version of flash memory
    - Erasing and programming are done on board
    - Erasing is much slower and needs higher voltage (20 V)
    - Lower cell density

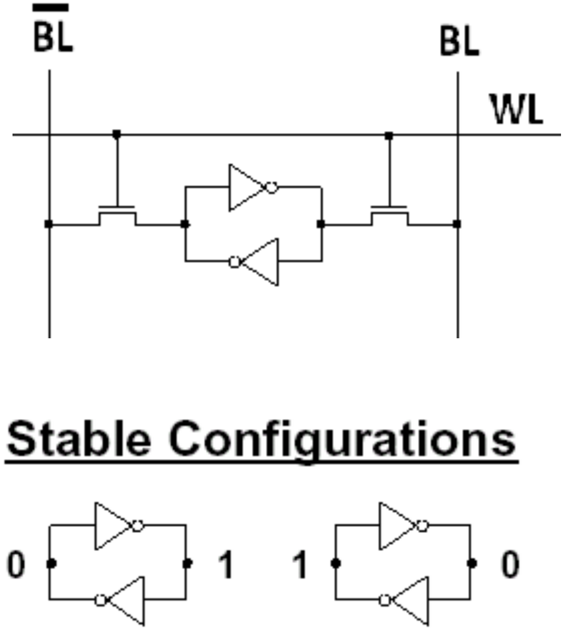


SRAM Cell

- 6 transistors



BL: Bit Line, carries data
WL: Word Line, selects cells



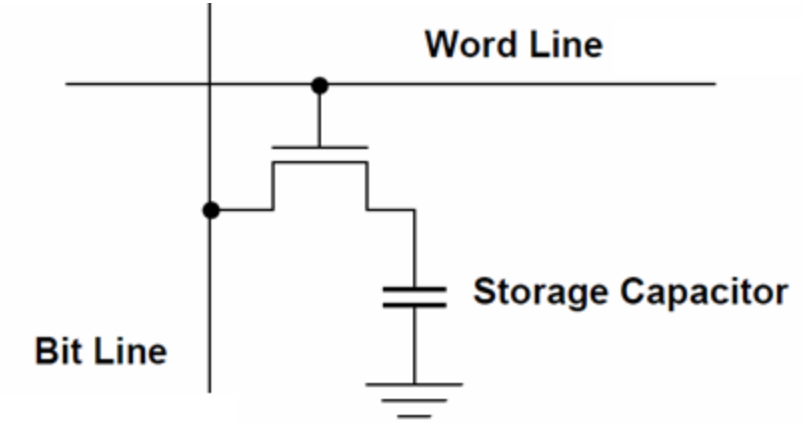
- On write
  - Set BLs to new data value
  - Raise WL to High
  - Cell is set to new state

- On read
  - Set BLs High (precharge)
  - Set WL High and float BLs
  - One of the BLs goes Low
  - Sense the difference of BLs

Diagram source: Digital Integrated Circuits, A Design Perspective, Rabaey, et al., Prentice hall, 2002

DRAM Cell

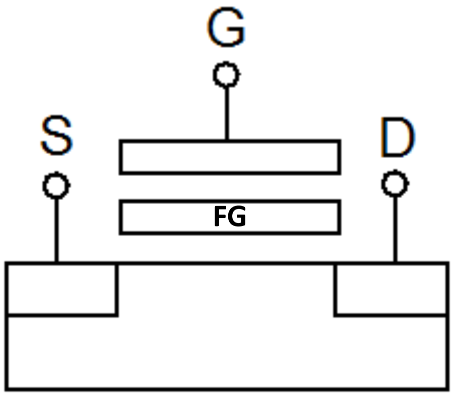
- 1 transistor, 1 capacitor
- On write
  - Set BL High or Low
  - Raise WL to High
  - Cell is set to new state



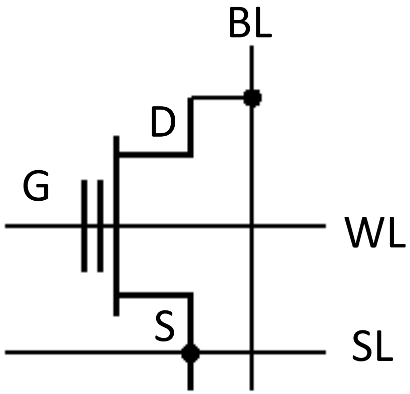
- On Read
  - Precharge BL to half of V_{dd}
  - Set WL High and float BL
  - Capacitor shares charge with BL, causing a change of voltage in BL
  - Sense BL voltage
    - Higher than half of V_{dd}: High is read
    - Lower than half of V_{dd}: Low is read
  - Restore charge state in capacitor (recharge or drain)
    - Execute a write operation of the same data just read

Flash Memory Cell

- 1 transistor with floating gate (FG)
  - Electrons at FG means transistor cannot be conducting
    - Neutralize the effect of an application of voltage at gate
- NOR flash (used for program execution) operations:
- On Erase
  - Float BL, set WL Low and apply 12 V to SL
  - Effect: Clear all electrons at FG
    - Cell stores a 1 (transistor can be turned ON)
- On Write/Program
  - Apply ~12 V to WL and set SL to Low
  - Store 0: Apply ~7 V to BL ➔ Electrons at FG
  - Otherwise, set BL to Low ➔ No change at FG
- On Read
  - Apply ~5V to WL, ~ 1V to BL and set SL to Low
    - Stored 0 (electrons at FG): Transistor is turned off, no current from D to S
    - Stored 1 (no electrons at FG): Transistor is turned on, some current from D to S
  - Presence of current flow in BL means a stored 1



G: Gate, D: Drain, S: Source

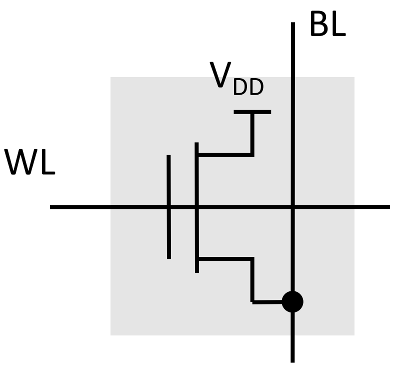


SL: Source Line

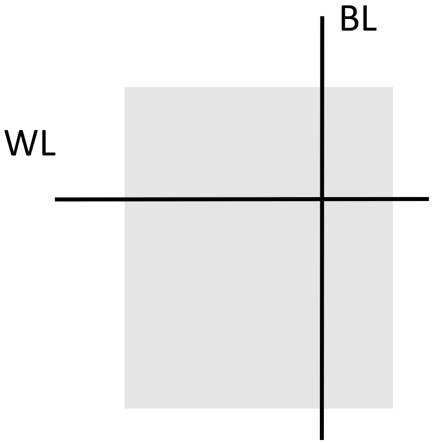
ROM Cell

- Implementation with at most 1 transistor
  - Data “written” during chip design

- Cell containing a transistor stores a 1



- Cell containing no transistor stores a 0



- On Read
  - Set WL High
  - BL becomes High or Low
    - With BL attached to a pull-down resistor

Memory Cell Organization

- Cells are arranged in a 2-D array
- Word line can be various lengths
  - 8, 16, 32, 64 or 128 bits
- Sense amplifiers
  - Detect changes on bit lines
  - Differential type for SRAM
    - Memory cell has two bit lines
- Achieves high density of cells

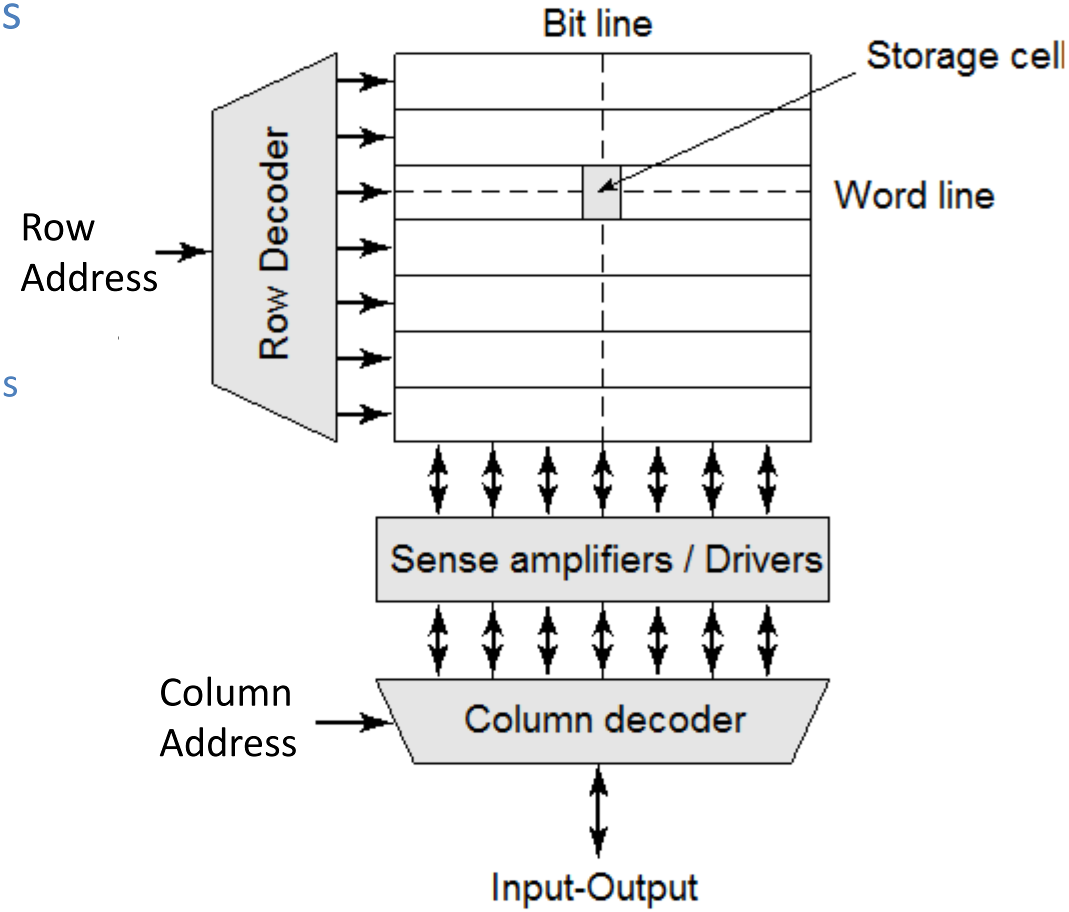


Diagram source: Digital Integrated Circuits, A Design Perspective, Rabaey, et al., Prentice hall, 2002

Memory Controller

A controller manages the operations on one or more memory cell arrays. It has a number of functions.

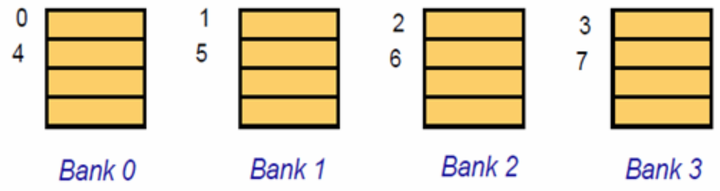
- Interfaces with the rest of the system
  - For example, connects to the AHB (Advanced High-Performance Bus)
- Provides various operation modes
  - Asynchronous or synchronous (to the system clock)
  - Burst data transfer
  - Low power mode (for example, only retains data)
- Control
  - Command decoding
  - Configuration (from processor)
    - Number of wait states
    - Erase protection bits (for flash memory)
  - Data buffering
    - Provides caching scheme to improve access performance

Memory Controller (cont'd)

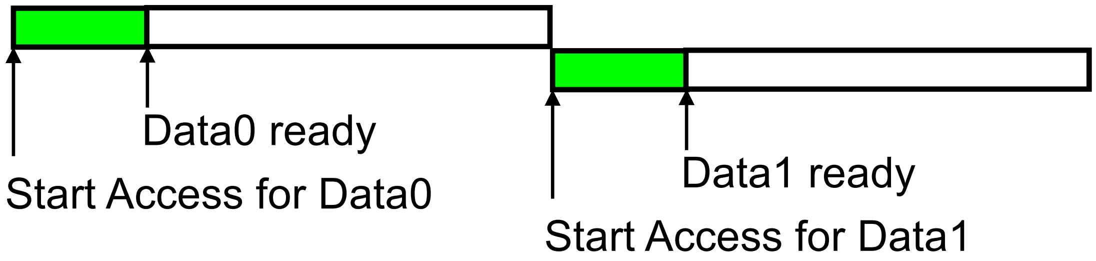
- Timing
  - Read, write, erase (for flash memory)
    - May need to insert wait states with high-frequency bus clocks
  - Refresh (for DRAM)
  - DDR (Double Data Rate) operations (for DRAM)
    - Uses both the falling and rising edges of clock
  - Command pipelining
- Address decoding
  - Maps input address to into bank, row, column
- Error detection and correction
  - Uses parity (single bit) or ECC (Error Correction Code, multiple bits)
- Interfaces with target memory (SRAM, DRAM, flash, ROM)
  - Row and column address decoders
  - Sense amplifiers and drivers (on bit lines)
  - Higher voltages (for flash memory)

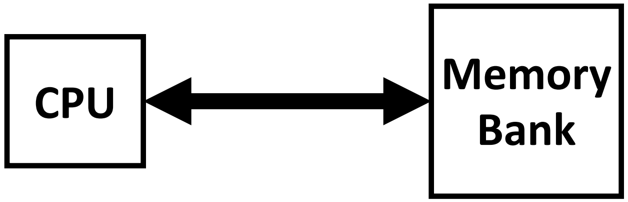
Memory Interleaving

- Put a line of words in different memory banks
  - To achieve better performance

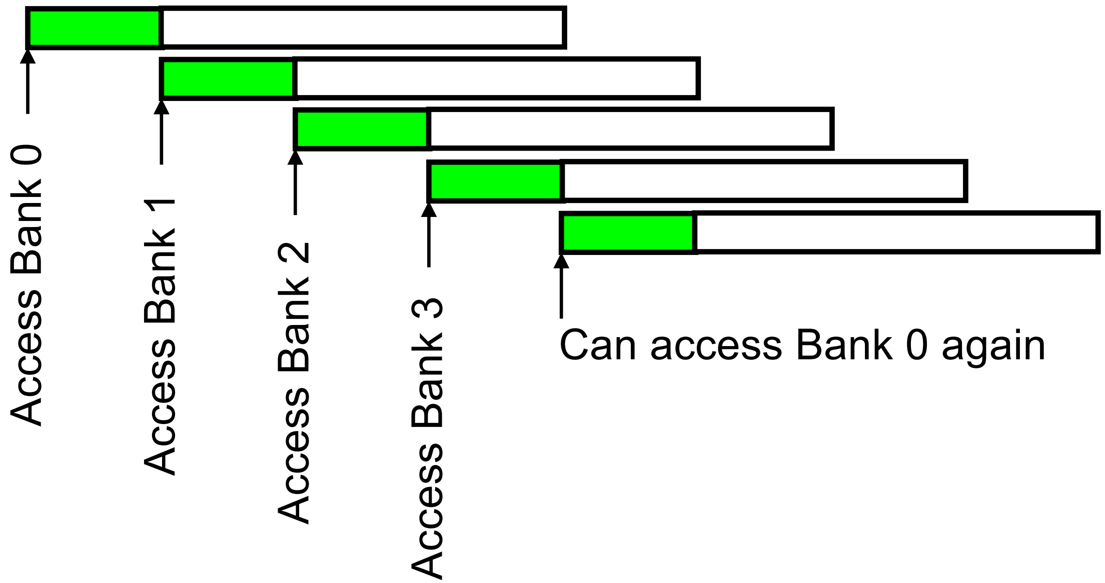


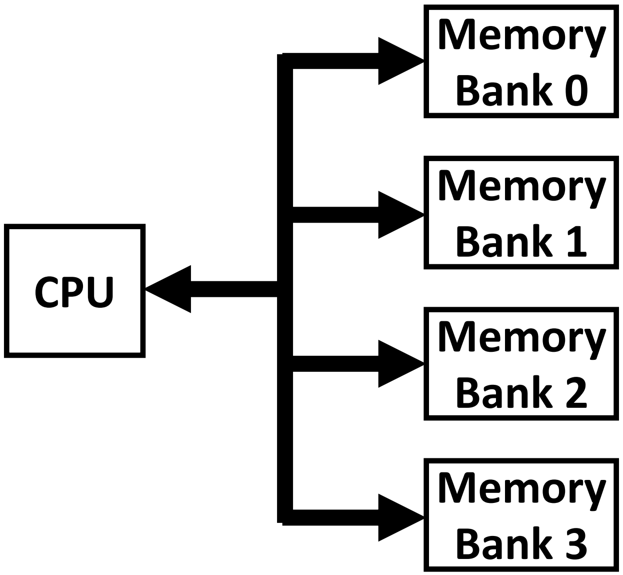
- Access pattern without interleaving:





- Access pattern with 4-way interleaving:





Memory Module Example

- Integrated Silicon Solution IS42/45X 512-Mb Synchronous DRAM

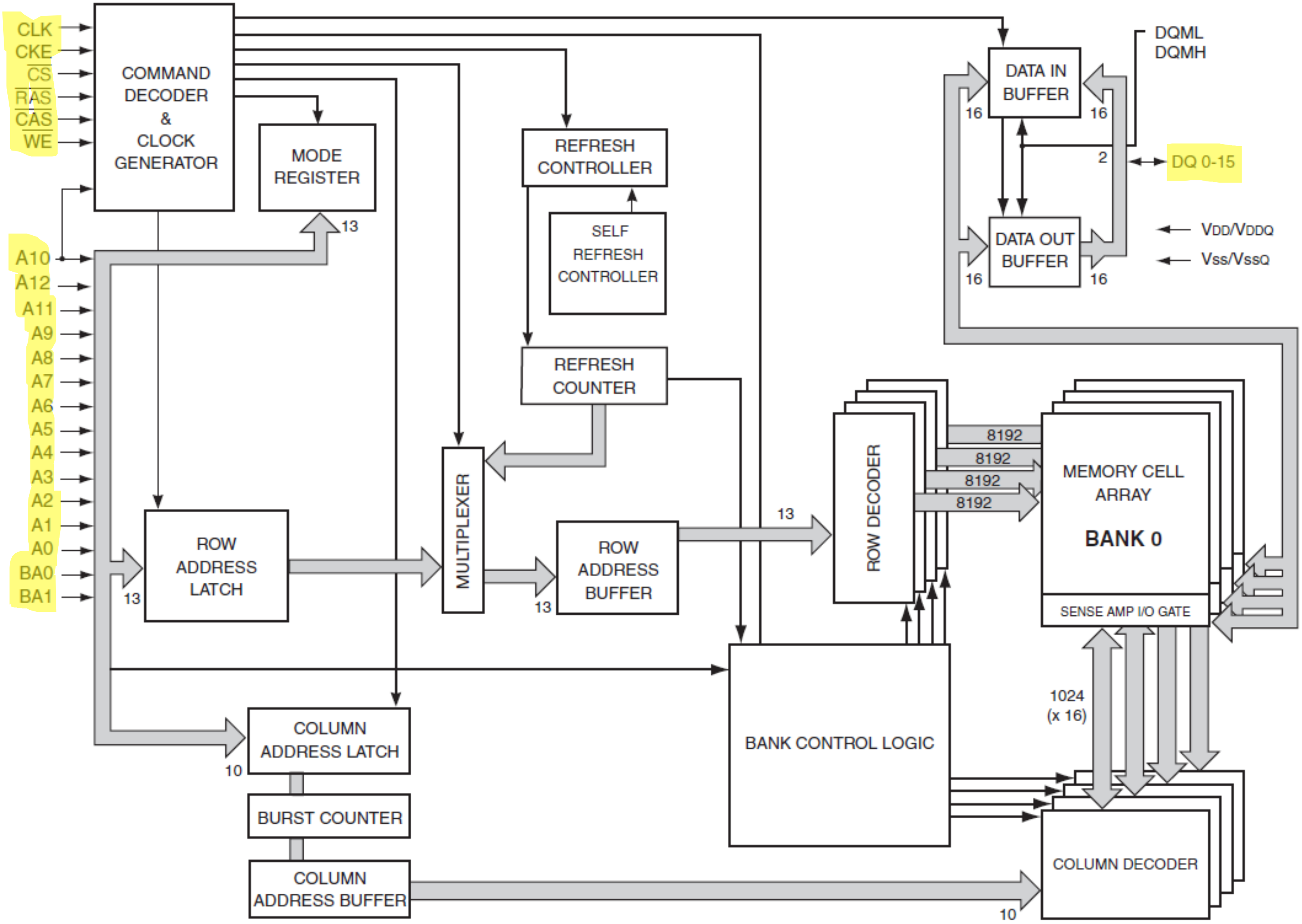
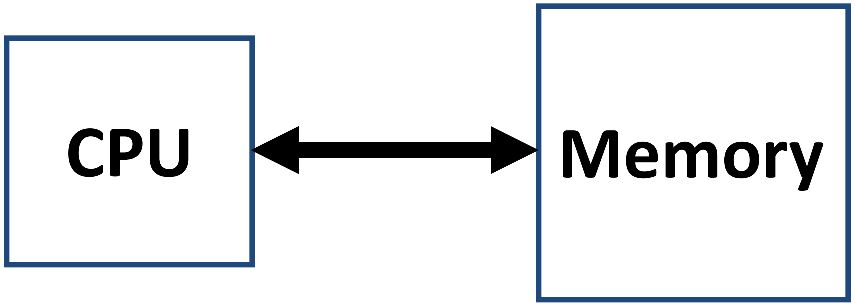


Diagram source: Integrated Silicon Solution IS42/45X 512-Mb Synchronous DRAM Datasheet

Computing Performance

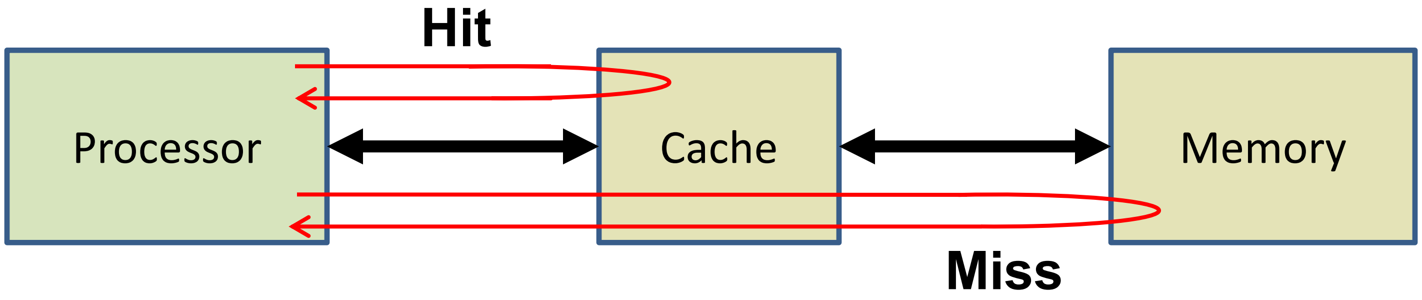


Computing performance is usually limited by memory latency and bandwidth

- Latency
  - Access time: Time between request made and data arrives
  - Affected by cycle time: Minimum time between requests to memory
- Bandwidth
  - Data transferred per unit time
- Modern embedded processors can run at hundreds of MHz or higher
  - Need very fast memory to avoid stalling the processor execution
    - Normal main memory with at least tens of ns access time is not adequate

Memory Cache

- A very effective solution to much narrow the gap between processor and memory speeds
- Create a very fast and small memory (cache) sitting between the processor and the main memory
  - Cache operates at (or very close to) the processor speed



- How it works?
  - When processor reads, if the data is not in the cache, it reads a block of data (cache line) from memory including the desired data into the cache [**Cache Miss**]
  - Returns the desired data to processor
  - Next time when processor reads, if the data is in the cache, data is returned to processor immediately [**Cache Hit**]
  - When processor writes, it can write to cache first immediately, without going all the way to main memory

Memory Hierarchy

- The closer to processor core, the faster the memory is
  - Amount of availability is also less
  - More expensive per bit
- Gives the illusion of a large, fast memory being presented to the processor
- Goal: To reduce the average memory access time

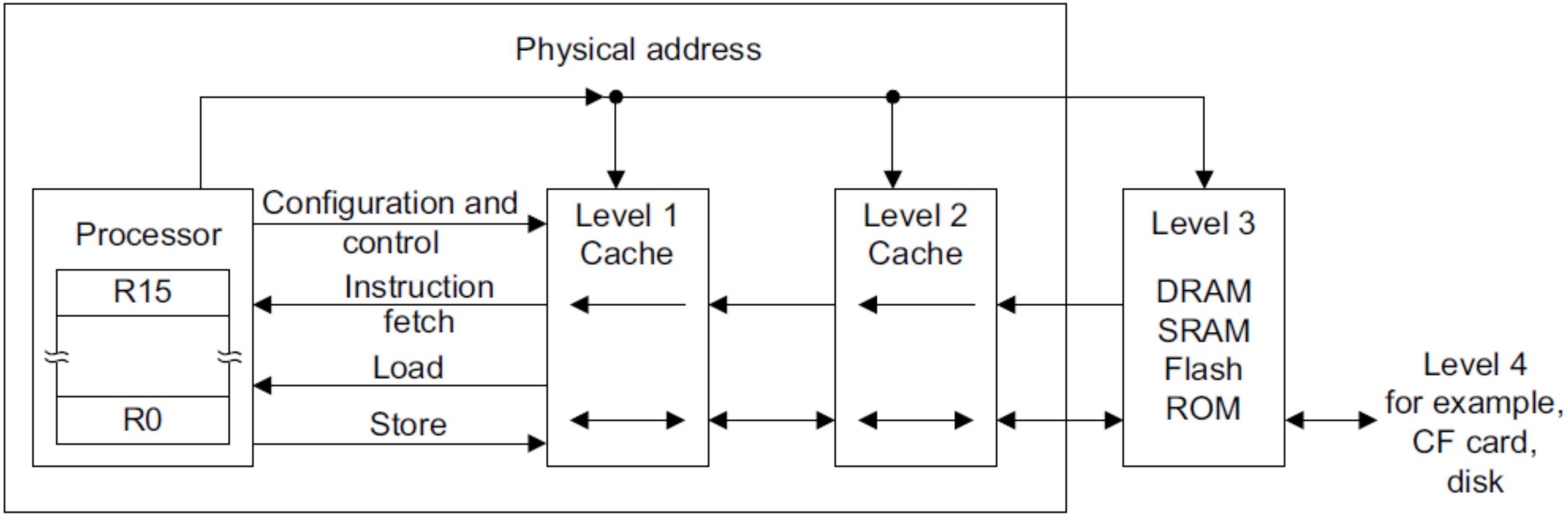
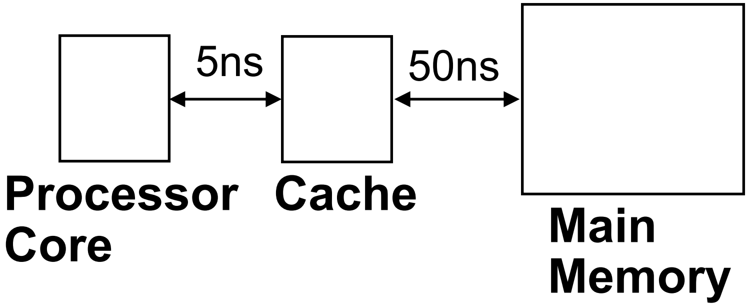


Diagram source: ARMv7-M Architecture Reference Manual

Why Cache Works?

- Most application executions exhibit two properties: Temporal and spatial localities
- Temporal locality
  - If a location is referenced, it is likely to be referenced again in the near future
  - For examples, small program loops, array index variable in a loop, table pointer, etc.
  - Cache “remembers” the contents of recently accessed locations
- Spatial locality
  - If a location is referenced, it is likely that locations near it will be referenced in the near future
  - For example, structured data such as arrays, matrices, queues, etc.
  - Cache also fetches data around recently accessed locations



```
sum = 0;
for (int i=0; i<N; i++)
   sum += data[i];
```

Challenges

- Need to manage potentially two copies of same data
  - Only modified data are problematic
  - Management complexity increases quickly with more processors in system
- DMA
  - Controller typically has no access to cache, which is tightly coupled with the processor, not attached to the system bus
  - Possible solutions
    - Introduce circuitry to detect potentially problematic conditions on bus
    - Make application program be aware of potential conflicts