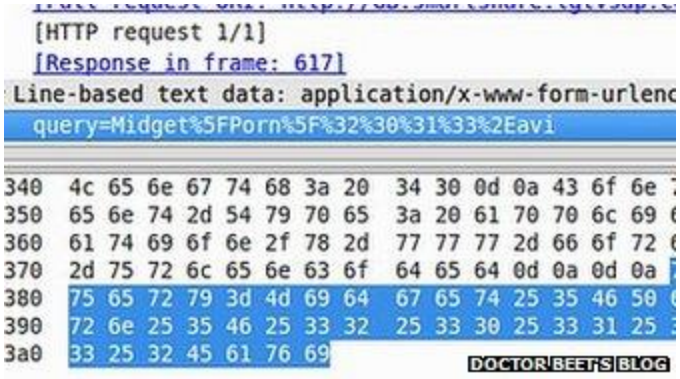
Embedded Hardware (2)

Kai Huang

Your Smart TV is Spying on You

- Digging into the TV's menu system, he had noticed that an option called "collection of watching info" had been switched on by default, he said.
- After switching it off, he had been surprised to find evidence that unencrypted details about each channel change had still been transmitted to LG's computer servers, but this time a flag in the data had been changed from "1" to "0" to indicate the user had opted out.



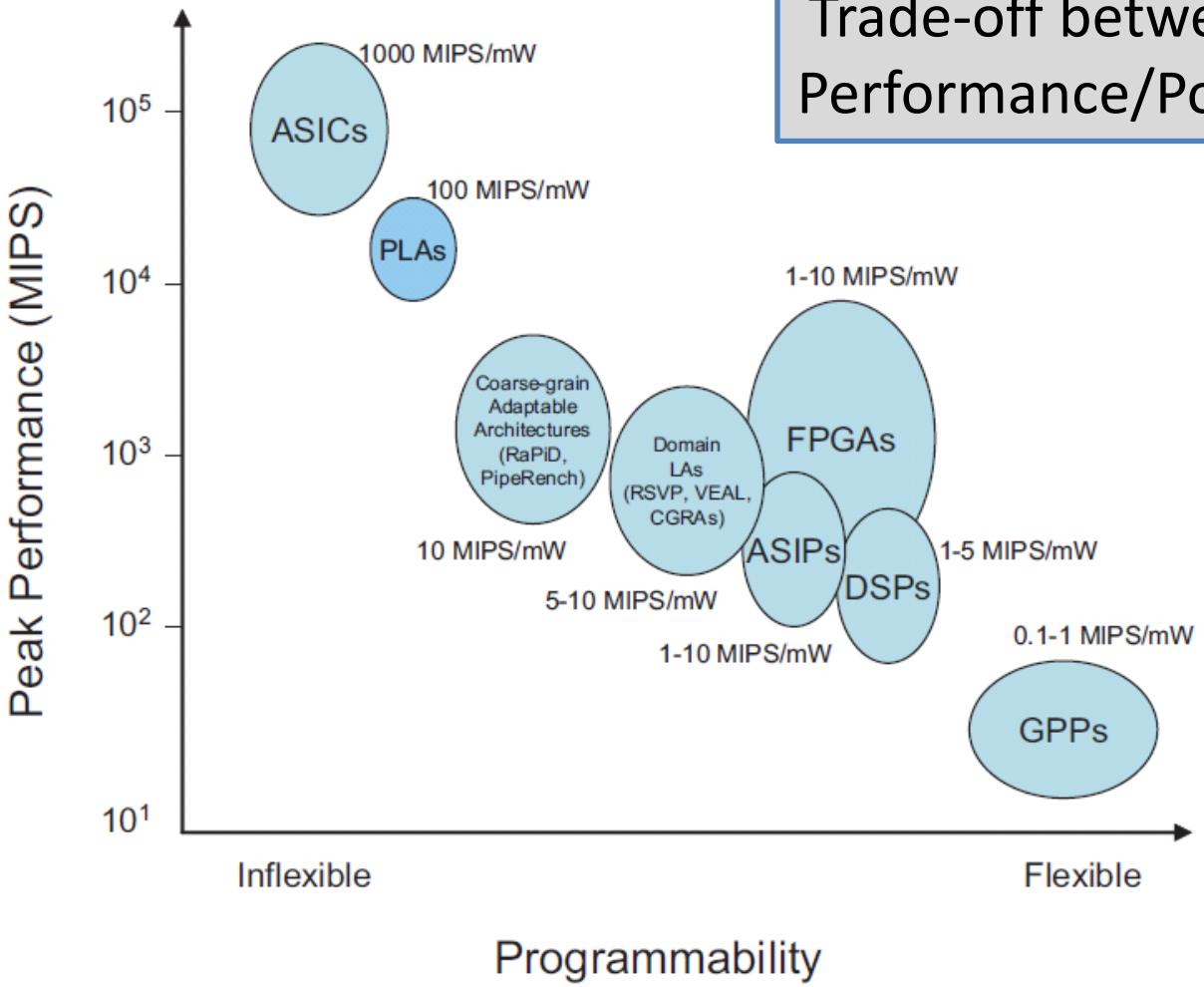
http://www.bbc.co.uk/news/technology-25018225

Outline

- Processor (recap)
- Memory
- I/O

Why Implementation Alternatives?

Trade-off between Flexibility and Performance/Power Efficiency



Outline

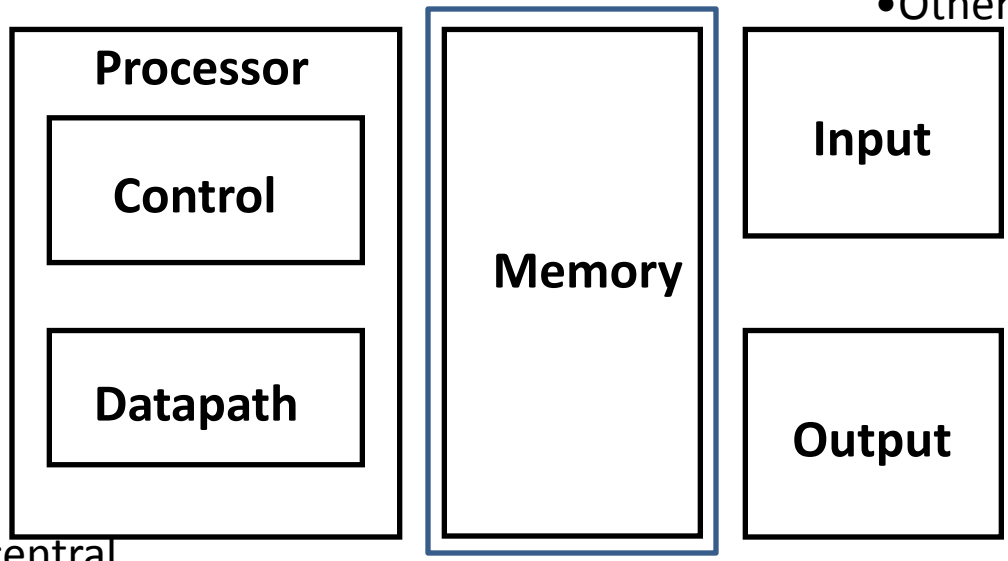
- Processor (recap)
- Memory
  - Basics
  - Memory Hierarchy
  - Cache
- I/O

The Big Picture

- Since 1946 all computers have had 5 components

Input unit accepts information:
- Human operators,
- Electromechanical devices
- Other computers

Control unit coordinates various actions:
- Input,
- Output
- Processing



Datapath:
- the part of the central processing unit (CPU) that does the actual computations

Stores information:
- Instructions,
- Data

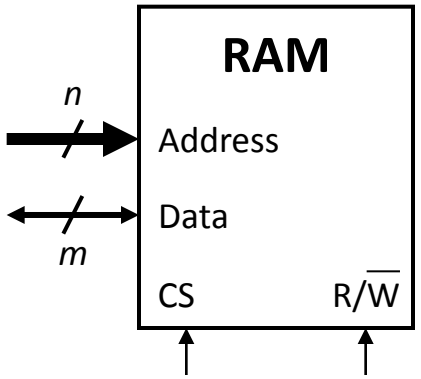
Output unit sends results of processing:
- To a monitor display,
- To a printer

What Is Memory?

Random Access Memory (RAM)

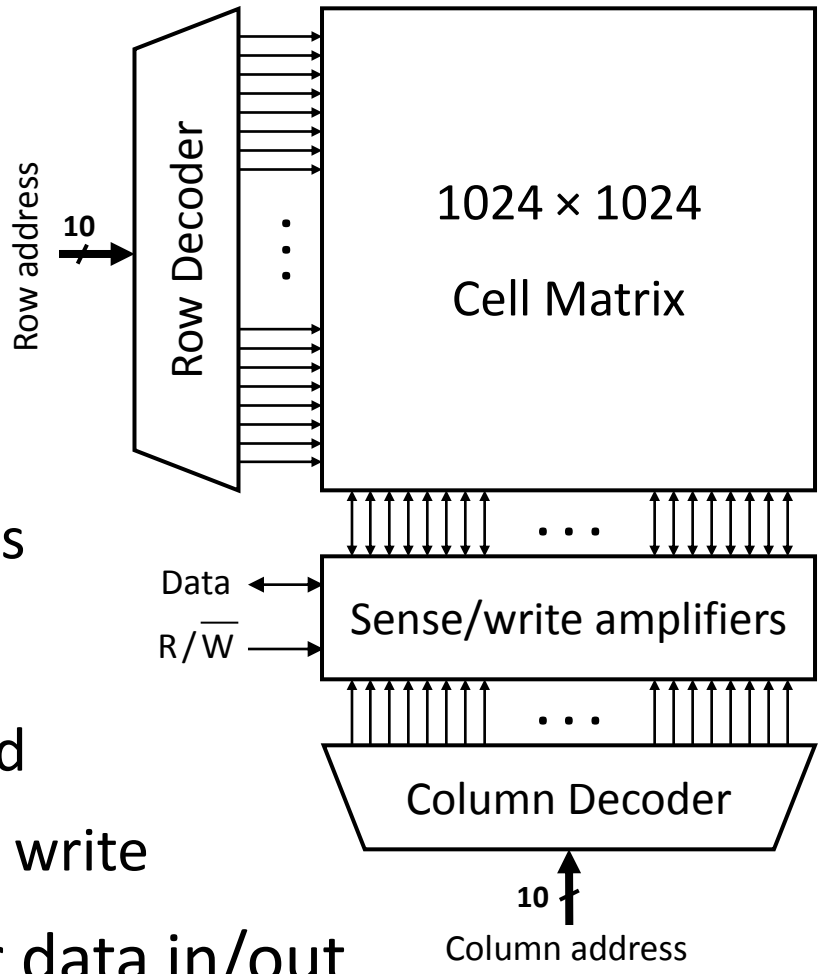
- Large arrays of storage cells
- Volatile memory
  - Hold the stored data as long as it is powered on
- Random Access
  - Access time is practically the same to any data on a RAM chip

- Chip Select (CS) control signal
  - Select RAM chip to read/write
- Read/Write (R/$\overline{W}$) control signal
  - Specifies memory operation
- $2^n \times m$ RAM chip: n-bit address and m-bit data

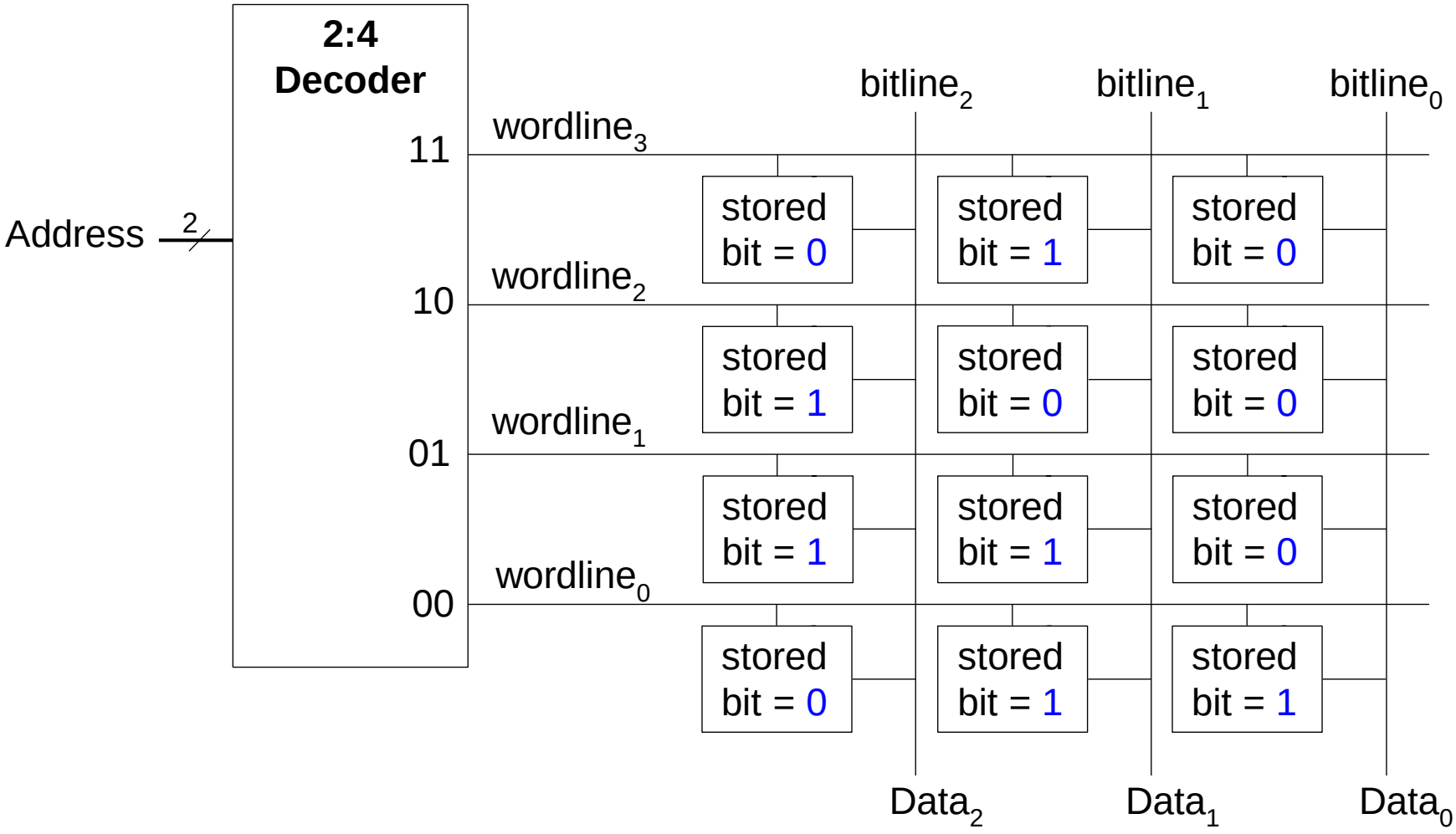


Typical Memory Structure

- Row decoder
  - Select row to read/write
- Column decoder
  - Select column to read/write
- Cell Matrix
  - 2D array of tiny memory cells
- Sense/Write amplifiers
  - Sense & amplify data on read
  - Drive bit line with data in on write
- Same data lines are used for data in/out

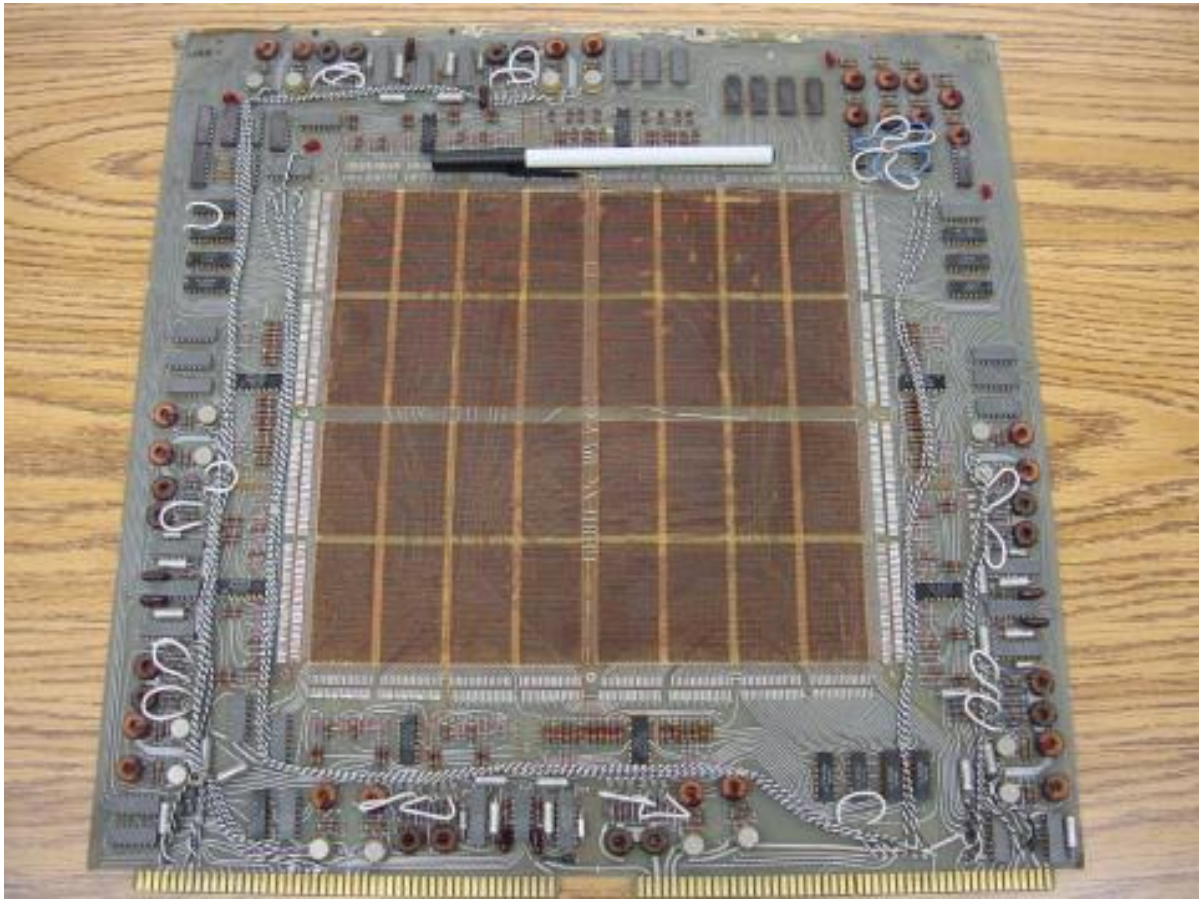


Memory Arrays



Memory Board at Early 1970's or Late 1960's

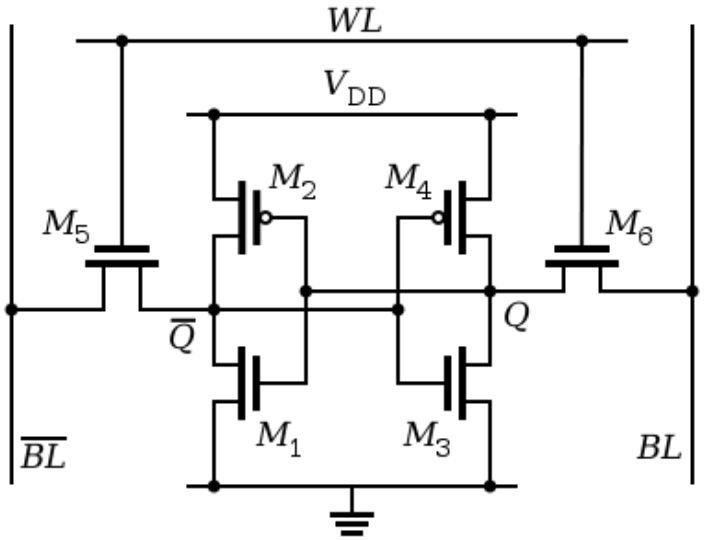
- 4Kbytes …



Static RAM Storage Cell

Static RAM (SRAM): fast but expensive RAM

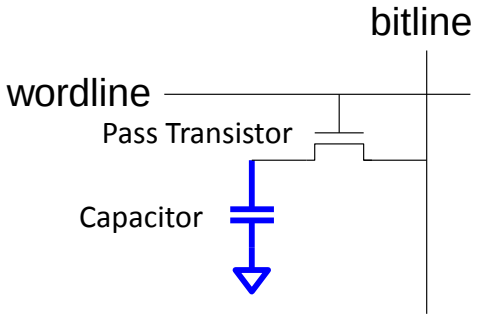
- 6-Transistor cell with no static current
- Typically used for caches
- Provides fast access time
- Cell Implementation:
  - Cross-coupled inverters store bit
  - Two pass transistors
  - Row decoder selects the word line
  - Pass transistors enable the cell to be read and written



Dynamic RAM Storage Cell

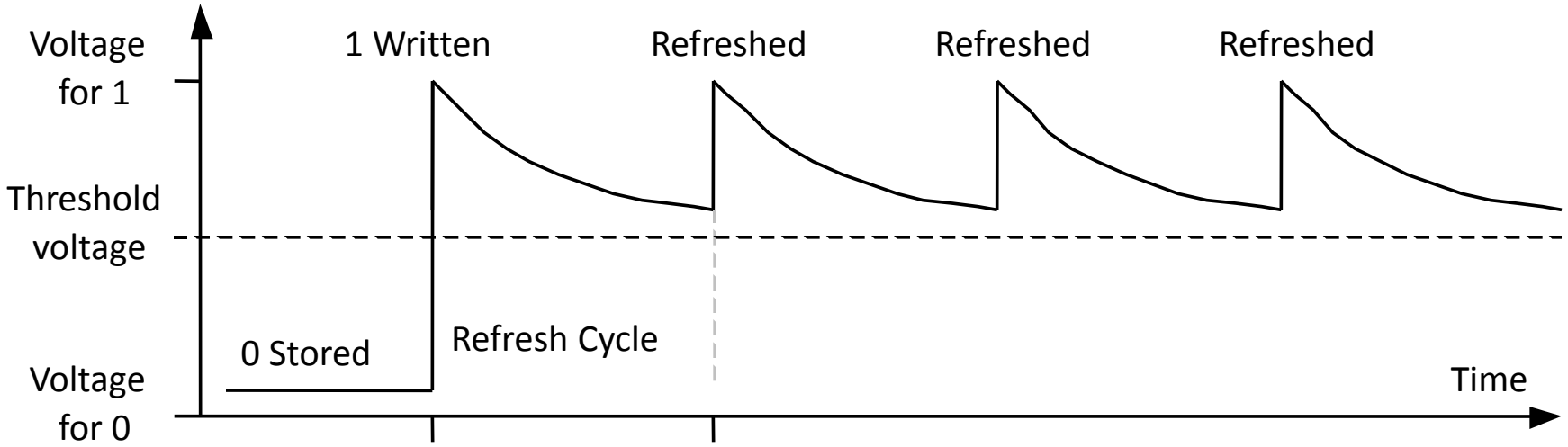
Dynamic RAM (DRAM): slow, cheap, and dense memory

- Typical choice for main memory
- Cell Implementation:
  - 1-Transistor cell (pass transistor)
  - Trench capacitor (stores bit)
- Bit is stored as a charge on capacitor
- Must be refreshed periodically
  - Because of leakage of charge from tiny capacitor
- Refreshing for all memory rows
  - Reading each row and writing it back to restore the charge



DRAM Refresh Cycles

- Refresh cycle is about tens of milliseconds
- Refreshing is done for the entire memory
- Each row is read and written back to restore the charge
- Some of the memory bandwidth is lost to refresh cycles



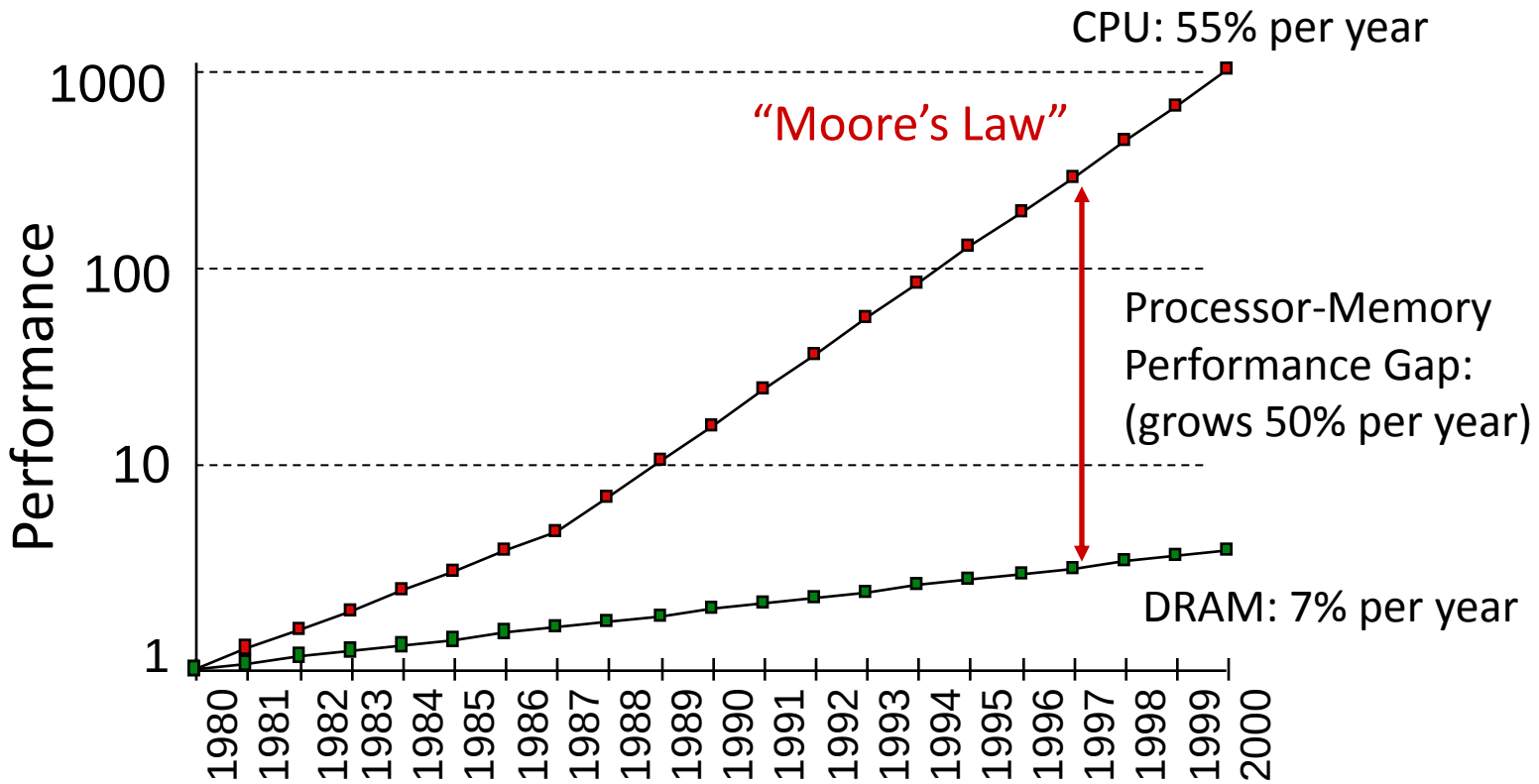
Loss of Bandwidth to Refresh Cycles

- Example:
  - A 256 Mb DRAM chip
  - Organized internally as a 16K × 16K cell matrix
  - Rows must be refreshed at least once every 50 ms
  - Refreshing a row takes 100 ns
  - What fraction of the memory bandwidth is lost to refresh cycles?
- Solution:
  - Refreshing all 16K rows takes: 16 × 1024 × 100 ns = 1.64 ms
  - Loss of 1.64 ms every 50 ms
  - Fraction of lost memory bandwidth = 1.64 / 50 = 3.3%

Outline

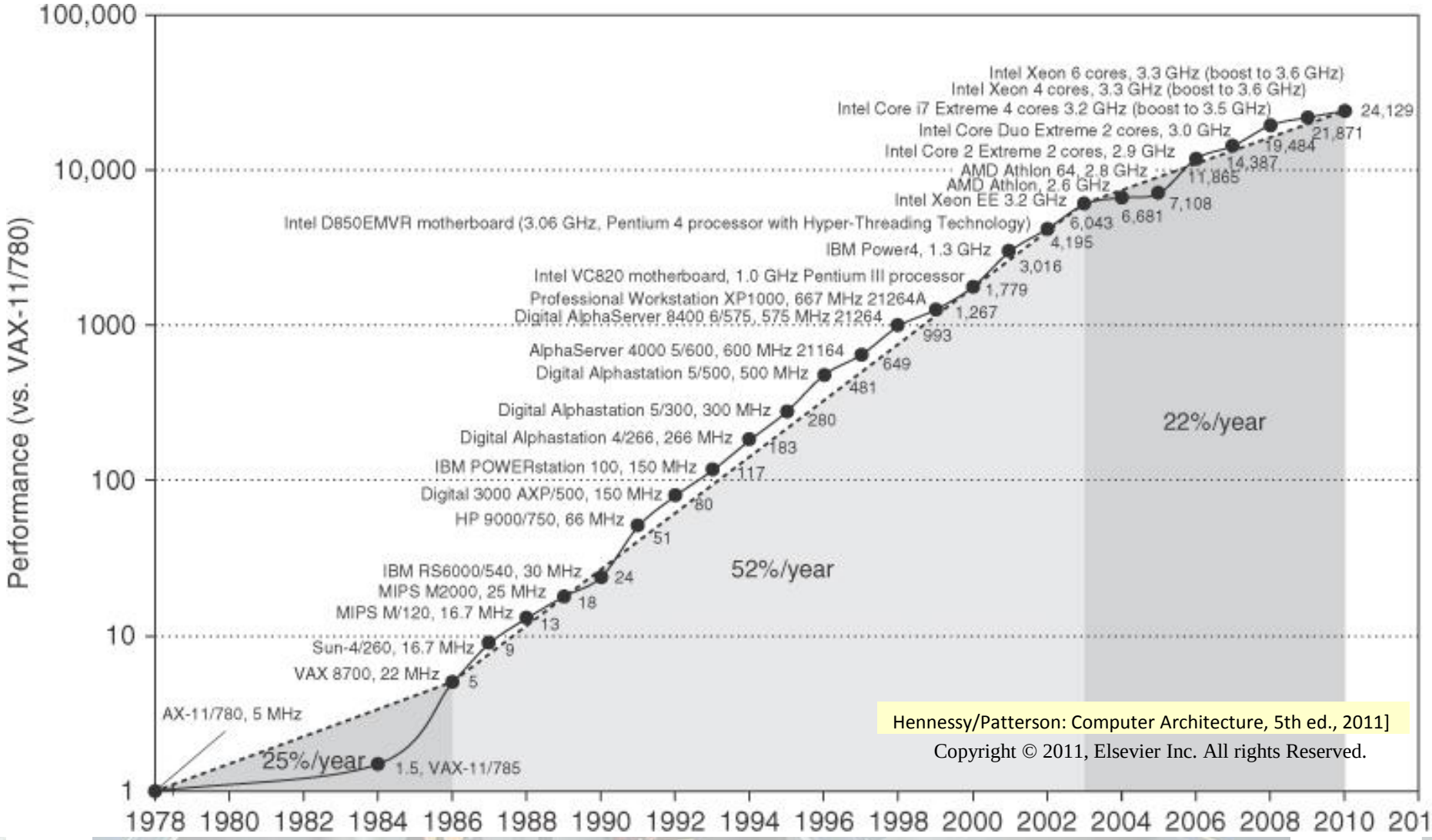
- Processor (recap)
- Memory
  - Basics
  - Memory Hierarchy
  - Cache
- I/O

Processor-Memory Performance Gap



- 1980 – No cache in microprocessor
- 1995 – Two-level cache on microprocessor

(Parallel) Performance Keeps Increasing



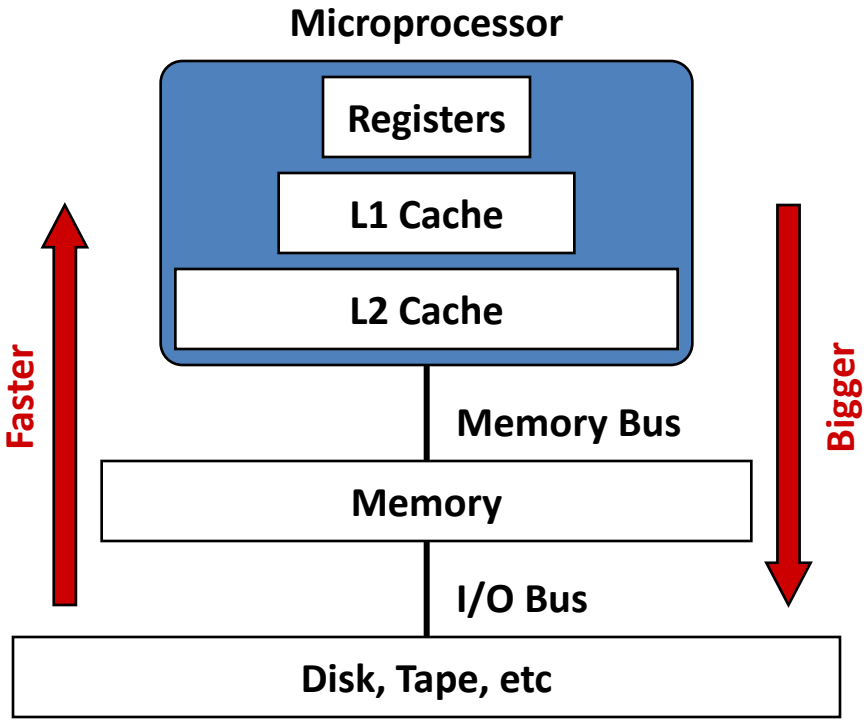
Hennessy/Patterson: Computer Architecture, 5th ed., 2011]

The Need for a Memory Hierarchy

- Widening speed gap between CPU and main memory
  - Processor operation takes less than 1 ns
  - Main memory requires more than 50 ns to access
- Each instruction involves at least one memory access
  - One memory access to fetch the instruction
  - A second memory access for load and store instructions
- Memory bandwidth limits the instruction execution rate
- Cache memory can help bridge the CPU-memory gap
- Cache memory is small in size but fast

Typical Memory Hierarchy

- Registers are at the top of the hierarchy
  - Typical size < 1 KB
  - Access time < 0.5 ns
- Level 1 Cache (8 – 64 KB)
  - Access time: 0.5 – 1 ns
- L2 Cache (512KB – 8MB)
  - Access time: 2 – 10 ns
- Main Memory (1 – 2 GB)
  - Access time: 50 – 70 ns
- Disk Storage (> 200 GB)
  - Access time: milliseconds



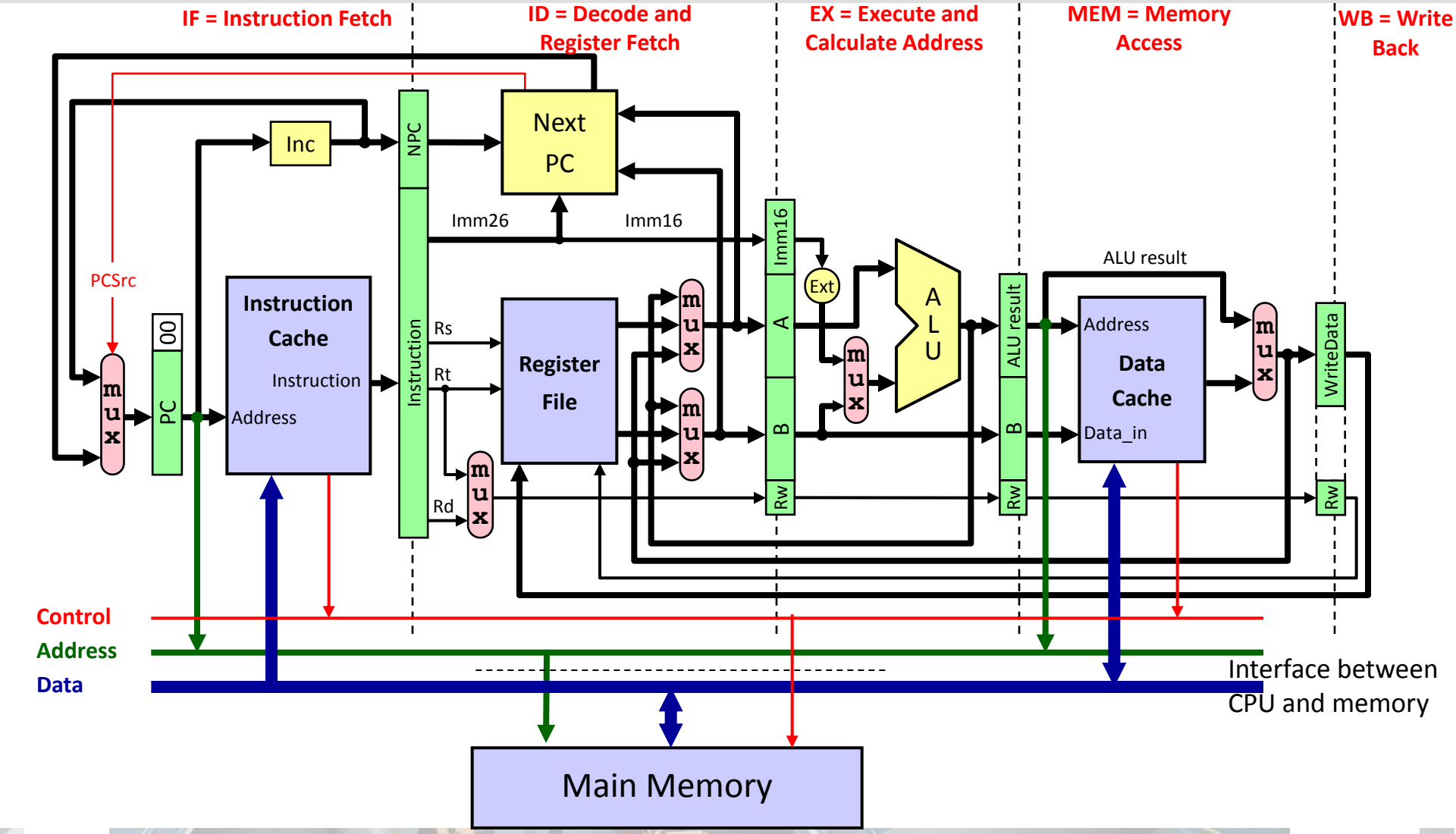
Principle of Locality of Reference

- Programs access small portion of their address space
  - At any time, only a small set of instructions & data is needed
- Temporal Locality (in time)
  - If an item is accessed, probably it will be accessed again soon
  - Same loop instructions are fetched each iteration
  - Same procedure may be called and executed many times
- Spatial Locality (in space)
  - Tendency to access contiguous instructions/data in memory
  - Sequential execution of Instructions
  - Traversing arrays element by element

What is a Cache Memory ?

- Small and fast (SRAM) memory technology
  - Stores the subset of instructions & data currently being accessed
- Used to reduce average access time to memory
- Caches exploit temporal locality by …
  - Keeping recently accessed data closer to the processor
- Caches exploit spatial locality by …
  - Moving blocks consisting of multiple contiguous words
- Goal is to achieve
  - Fast speed of cache memory access
  - Balance the cost of the memory system

Cache Memories in the Datapath



Almost Everything is a Cache !

- In computer architecture, almost everything is a cache!
- Registers: a cache on variables – software managed
- First-level cache: a cache on second-level cache
- Second-level cache: a cache on memory
- Memory: a cache on hard disk
  - Stores recent programs and their data
  - Hard disk can be viewed as an extension to main memory
- Branch target and prediction buffer
  - Cache on branch target and prediction information

Outline

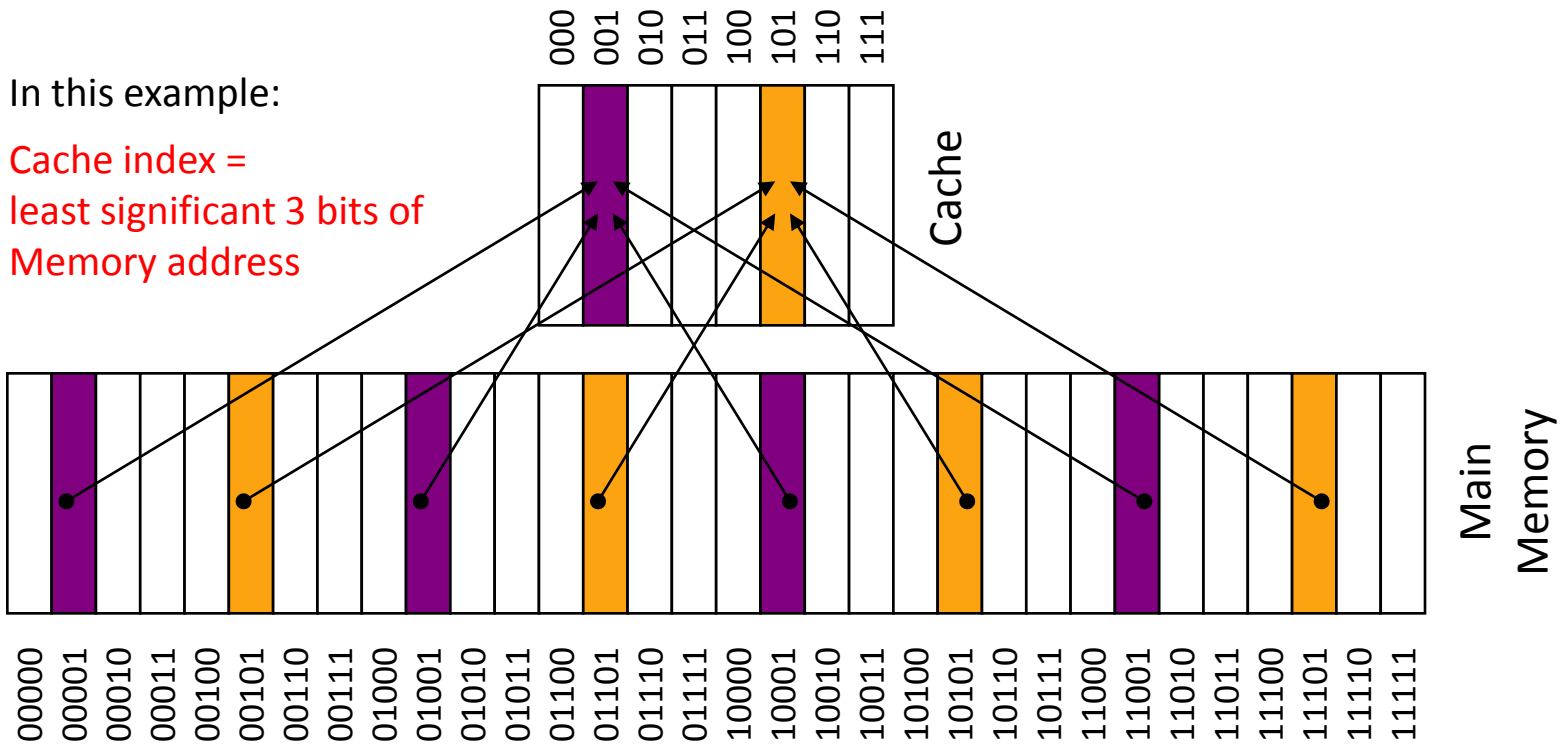
- Processor (recap)
- Memory
  - Basics
  - Memory Hierarchy
  - Cache
- I/O

Four Basic Questions on Caches

- Q1: Where can a block be placed in a cache?
  - Block placement
  - Direct Mapped, Set Associative, Fully Associative
- Q2: How is a block found in a cache?
  - Block identification/addressing
  - Block address, tag, index
- Q3: Which block should be replaced on a miss?
  - Block replacement
  - FIFO, Random, LRU
- Q4: What happens on a write?
  - Write strategy
  - Write Back or Write Through (with Write Buffer)

Block Placement: Direct Mapped

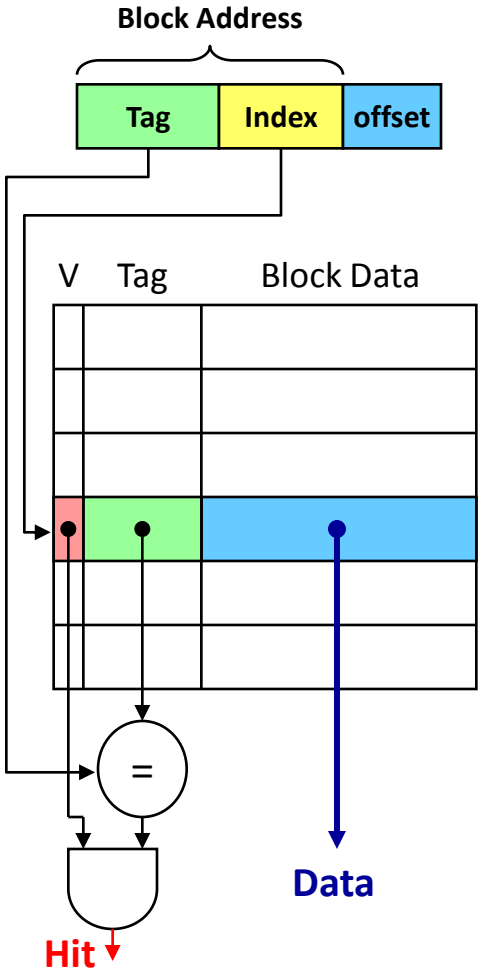
- Block: unit of data transfer between cache and memory
- Direct Mapped Cache:
  - A block can be placed in exactly one location in the cache



Direct-Mapped Cache

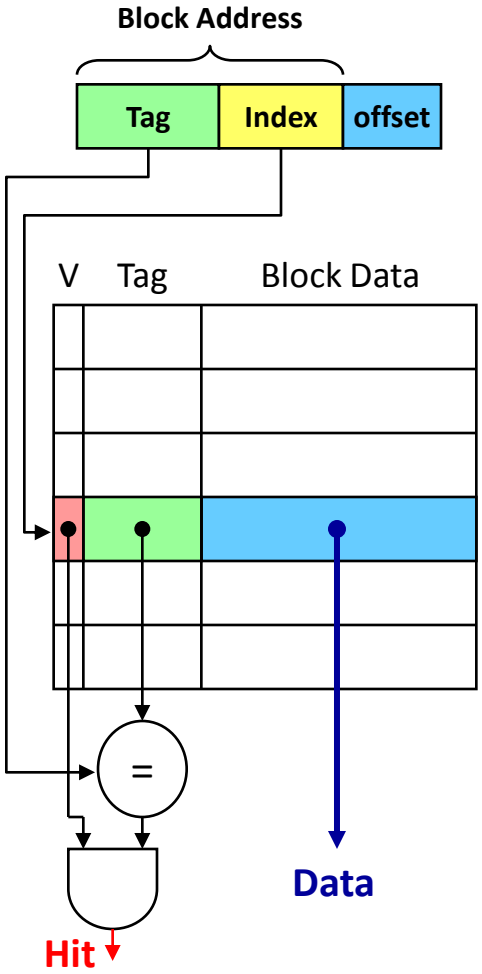
- A memory address is divided into
  - Block address: identifies block in memory
  - Block offset: to access bytes within a block
- A block address is further divided into
  - Index: used for direct cache access
  - Tag: most-significant bits of block address

  Index = Block Address **mod** *Cache Blocks*
- Tag must be stored also inside cache
  - For block identification
- A valid bit is also required to indicate
  - Whether a cache block is valid or not



Direct Mapped Cache – cont'd

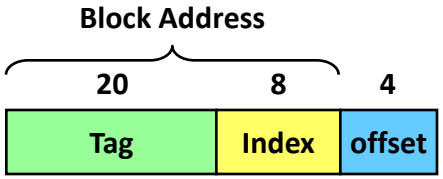
- Cache hit: block is stored inside cache
  - Index is used to access cache block
  - Address tag is compared against stored tag
  - If equal and cache block is valid then **hit**
  - Otherwise: **cache miss**
- If number of cache blocks is 2^n
  - n bits are used for the cache index
- If number of bytes in a block is 2^b
  - b bits are used for the block offset
- If 32 bits are used for an address
  - $32 - n - b$ bits are used for the tag
- Cache data size = 2^{n+b} bytes



Mapping an Address to a Cache Block

- Example
  - Consider a direct-mapped cache with 256 blocks
  - Block size = 16 bytes
  - Compute tag, index, and byte offset of address: 0x01FFF8AC

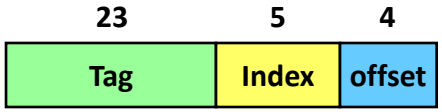
- Solution
  - 32-bit address is divided into:



    - 4-bit byte offset field, because block size = 2^4 = 16 bytes
    - 8-bit cache index, because there are 2^8 = 256 blocks in cache
    - 20-bit tag field
  - Byte offset = 0xC = 12 (least significant 4 bits of address)
  - Cache index = 0x8A = 138 (next lower 8 bits of address)
  - Tag = 0x01FFF (upper 20 bits of address)

Example on Cache Placement & Misses

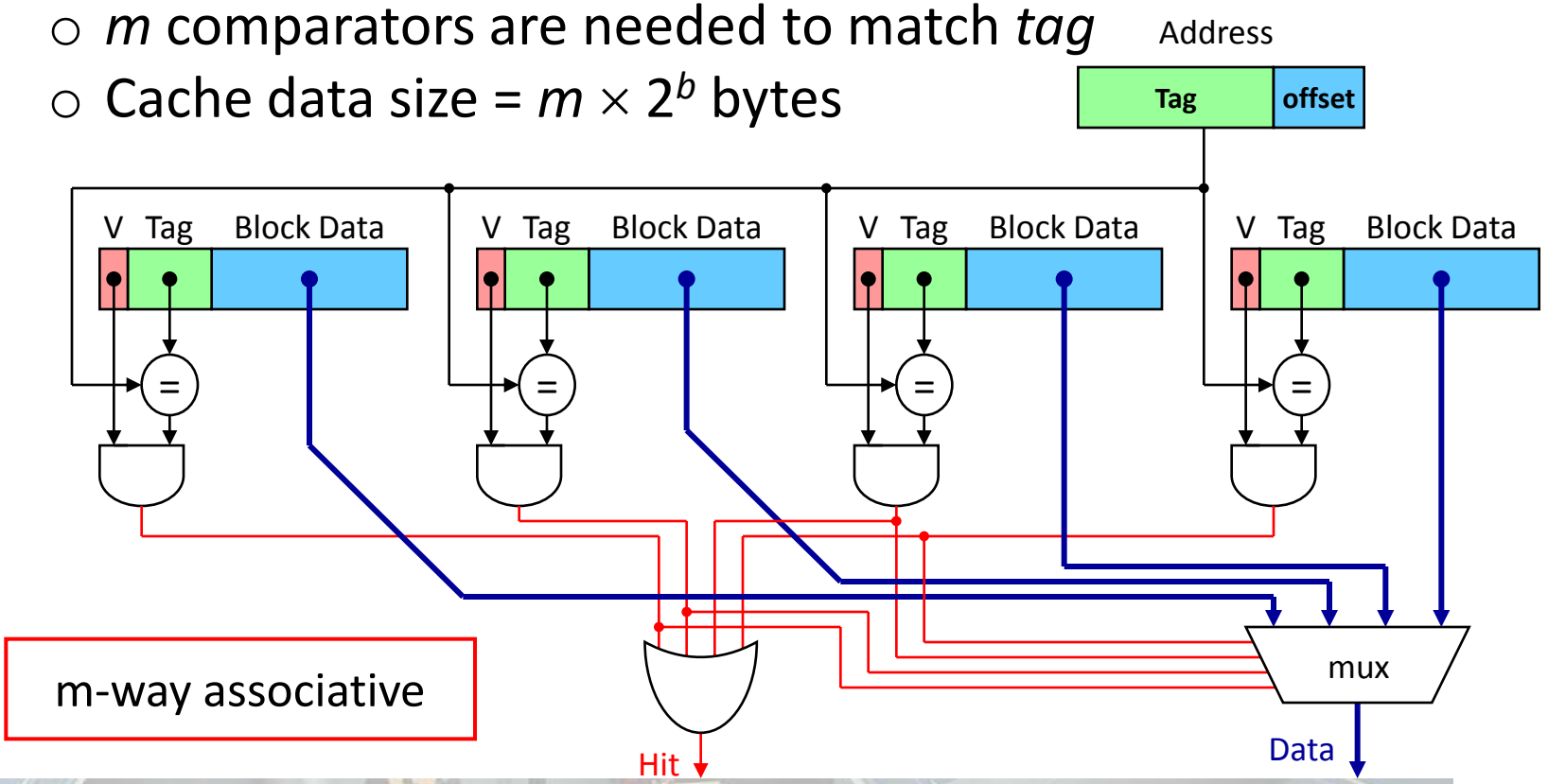
- Consider a small direct-mapped cache with 32 blocks
  - Cache is initially empty, Block size = 16 bytes
  - The following memory addresses (in decimal) are referenced: 1000, 1004, 1008, 2548, 2552, 2556.
  - Map addresses to cache blocks and indicate whether hit or miss
- Solution:



1000 = 0x3E8	cache index = 0x1E	Miss (first access)
1004 = 0x3EC	cache index = 0x1E	Hit
1008 = 0x3F0	cache index = 0x1F	Miss (first access)
2548 = 0x9F4	cache index = 0x1F	Miss (different tag)
2552 = 0x9F8	cache index = 0x1F	Hit
2556 = 0x9FC	cache index = 0x1F	Hit

Fully Associative Cache

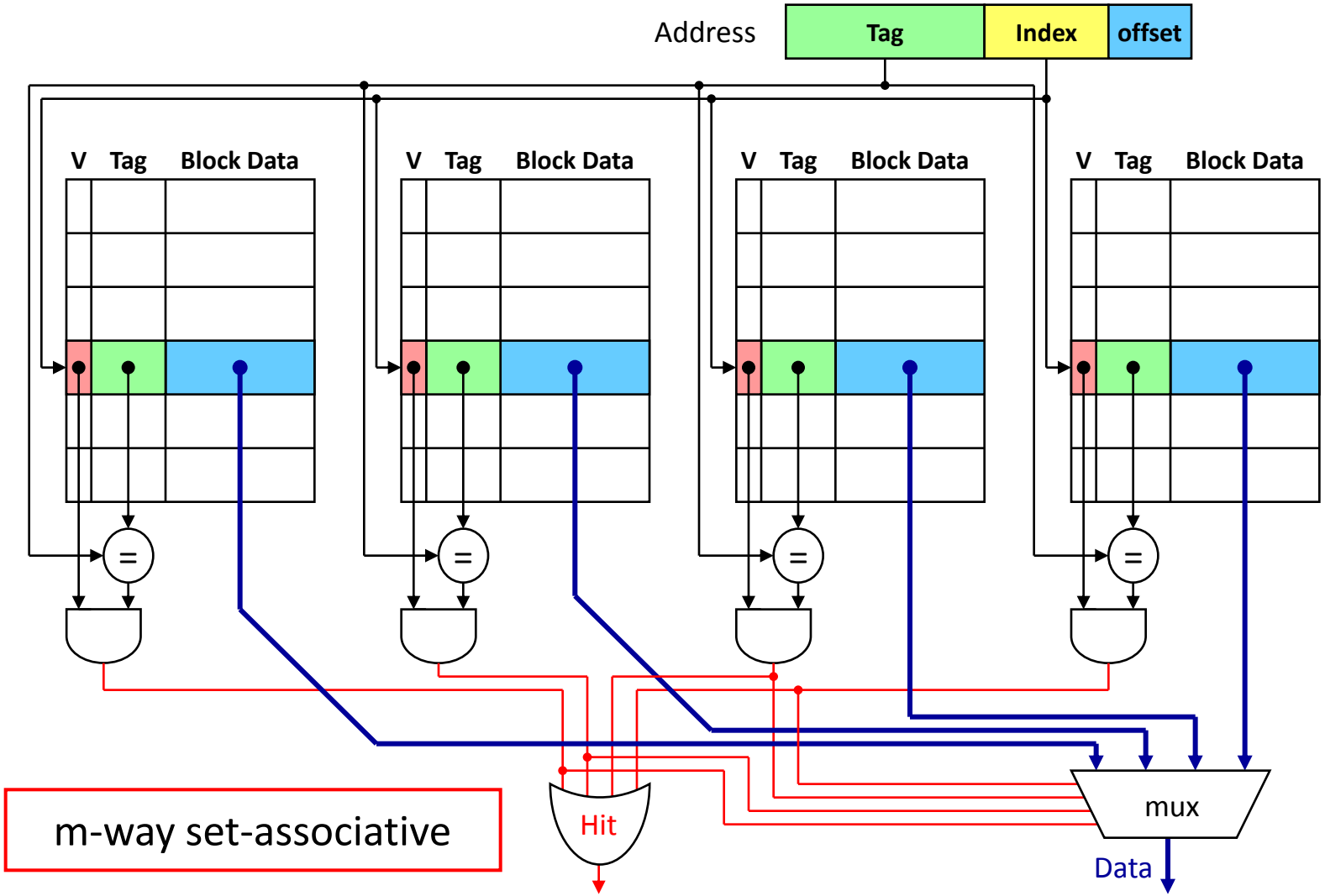
- A block can be placed anywhere in cache ⇒ no indexing
- If *m* blocks exist then
  - *m* comparators are needed to match *tag*
  - Cache data size = $m \times 2^b$ bytes



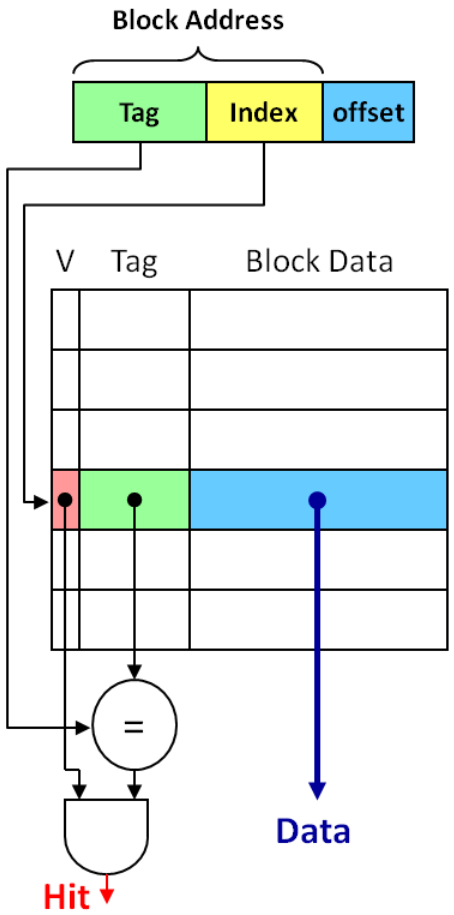
Set-Associative Cache

- A set is a group of blocks that can be indexed
- A block is first mapped onto a set
  - *Set index = Block address* **mod** *Number of sets in cache*
- If there are *m* blocks in a set (*m*-way set associative) then
  - *m* tags are checked in parallel using *m* comparators
- If 2^n sets exist then set index consists of *n* bits
- Cache data size = $m \times 2^{n+b}$ bytes (with 2^b bytes per block)
  - Without counting tags and valid bits
- A direct-mapped cache has one block per set ($m = 1$)
- A fully-associative cache has one set ($2^n = 1$ or $n = 0$)

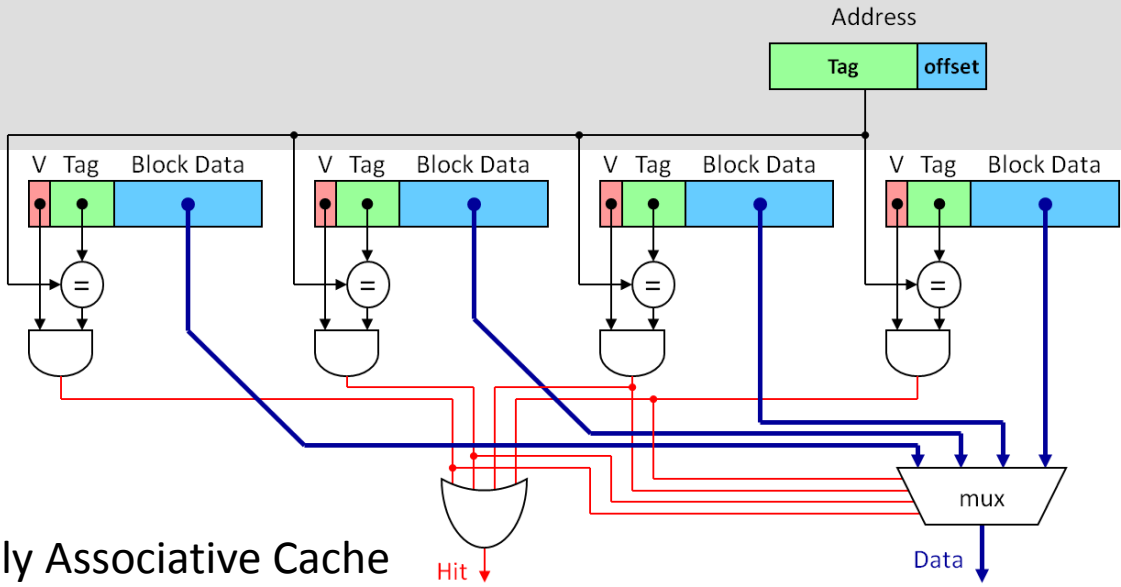
Set-Associative Cache Diagram



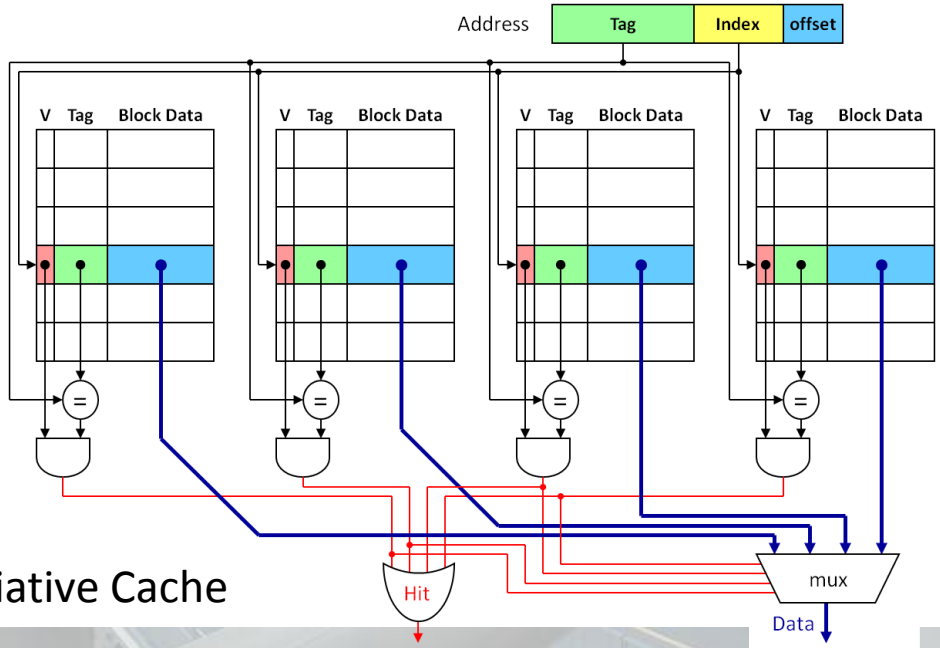
Comparison



a) Direct Mapped Cache



b) Fully Associative Cache



c) Set Associative Cache

What Happens on a Cache Miss?

- Cache sends a miss signal to stall the processor
- Decide which cache block to allocate/replace
  - One choice only when the cache is directly mapped
  - Multiple choices for set-associative or fully-associative cache
- Transfer the block from lower level memory to this cache
  - Set the valid bit and the tag field from the upper address bits
- If block to be replaced is modified then write it back
  - Modified block is moved into a Write Buffer
  - Otherwise, block to be replaced can be simply discarded
- Restart the instruction that caused the cache miss
- Miss Penalty: clock cycles to process a cache miss

Replacement Policy

- Which block to be replaced on a cache miss?
- No selection alternatives for direct-mapped caches
- *m* blocks per set to choose from for associative caches
- Random replacement
  - Candidate blocks are randomly selected
  - One counter for all sets (0 to *m* – 1): incremented on every cycle
  - On a cache miss replace block specified by counter
- First In First Out (FIFO) replacement
  - Replace oldest block in set
  - One counter per set (0 to *m* – 1): specifies oldest block to replace
  - Counter is incremented on a cache miss

Replacement Policy – cont'd

- Least Recently Used (LRU)
  - Replace block that has been unused for the longest time
  - Order blocks within a set from least to most recently used
  - Update ordering of blocks on each cache hit
  - With *m* blocks per set, there are *m*! possible permutations
- Pure LRU is too costly to implement when $m > 2$
  - $m = 2$, there are 2 permutations only (a single bit is needed)
  - m = 4, there are 4! = 24 possible permutations
  - LRU approximation are used in practice
- For large $m > 4$,

  Random replacement can be as effective as LRU

Write Policy

- Write Through:
  - Writes update cache and lower-level memory
  - Cache control bit: only a Valid bit is needed
  - Memory always has latest data, which simplifies data coherency
  - Can always discard cached data when a block is replaced
- Write Back:
  - Writes update cache only
  - Cache control bits: Valid and Modified bits are required
  - Modified cached data is written back to memory when replaced
  - Multiple writes to a cache block require only one write to memory
  - Uses less memory bandwidth than write-through and less power
  - However, more complex to implement than write through

Outline

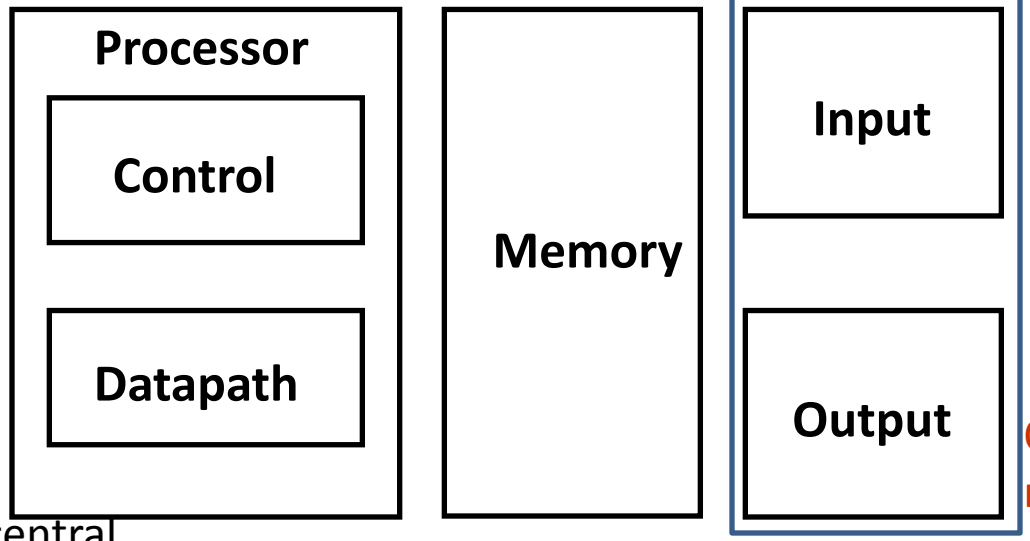
- Processor
- Memory
- I/O
  - PWM, GPIO, DAC, ADC
  - Interfacing I/O

The Big Picture

- Since 1946 all computers have had 5 components

Input unit accepts information:
- Human operators,
- Electromechanical devices
- Other computers

Control unit coordinates various actions:
- Input,
- Output
- Processing



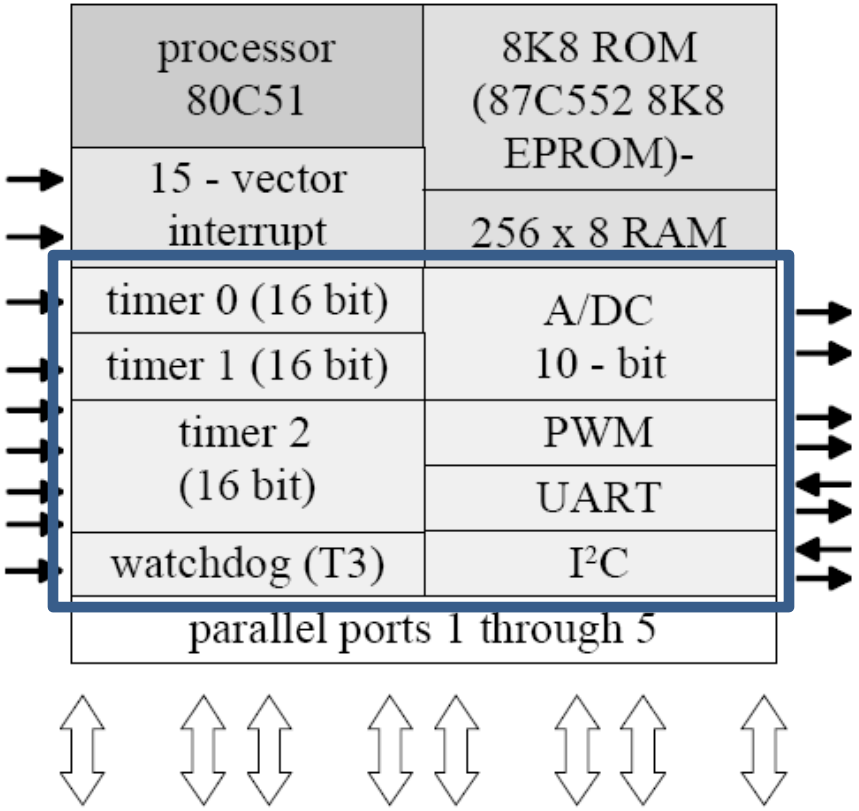
Datapath:
- the part of the central processing unit (CPU) that does the actual computations

Stores information:
- Instructions,
- Data

Output unit sends results of processing:
- To a monitor display,
- To a printer

Examples of I/O Devices

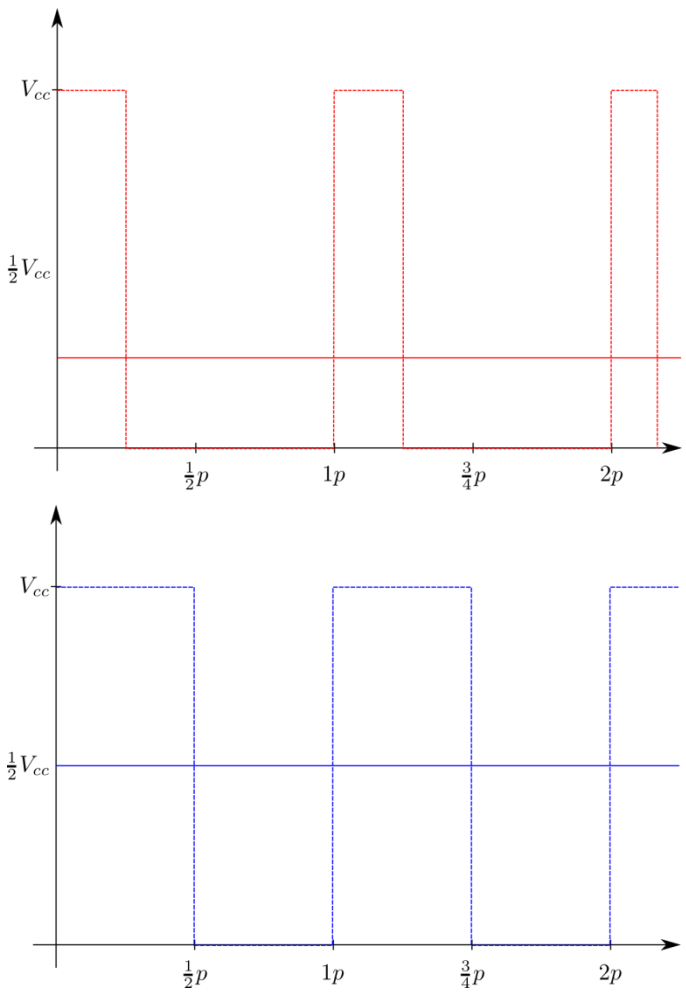
- Timers
- WatchDogs
- Pulse Width Modulators
- (PWM)
- Universal Asynchronous
- Receiver Transmitter
- (UART)
- Inter Integrated Circuit
- (I2C)
- ...



Philips 83C552:
8 bit-8051 based microcontroller

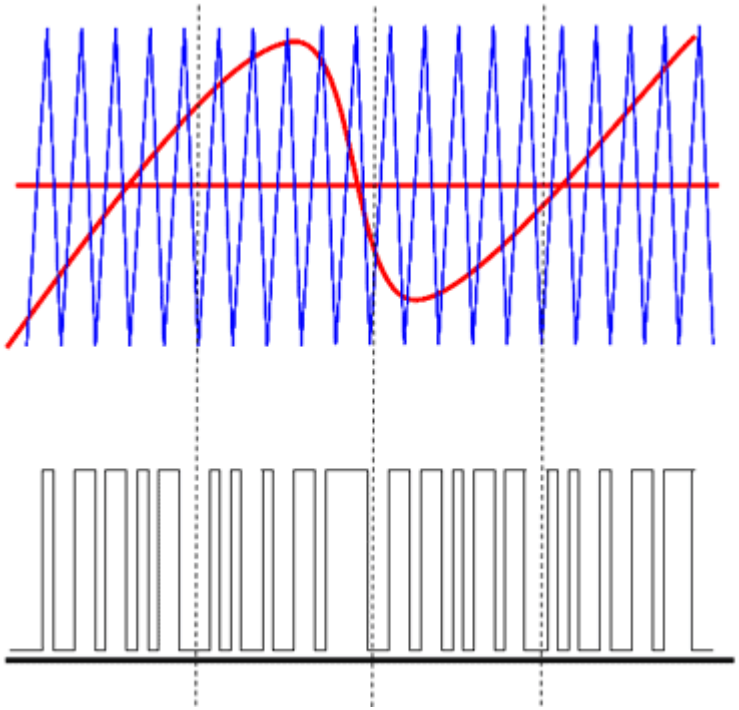
Pulse Width Modulation (PWM)

- Deliver variable amounts of power to external hardware (e.g. LED brightness, motor driver)
- Duty cycle: proportion of cycle time where voltage is high
- Programming through PWM interface



PWM cont‘d

- PWM with high frequencies (10-50kHz) to approximate signals
- Timer is configured to reload with fixed frequency (PWM frequency)
- Output Compare register is set to a value that is a percentage of the timer reload value (duty cycle)
- Output is set low when counter reaches Output Compare value

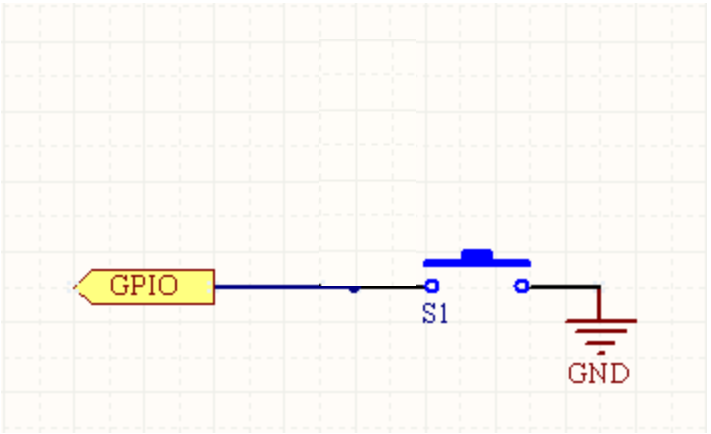


http://www.fairaudio.de

General Purpose IO (GPIO)

- General Purpose IO (GPIO) can be configured as digital input or output

- Digital output:
  Voltage of a pin can be set to high (VCC) or low (0V) by setting a bit in a register
- Digital input:
  State of pin (high or low) can be read by reading a bit in a register

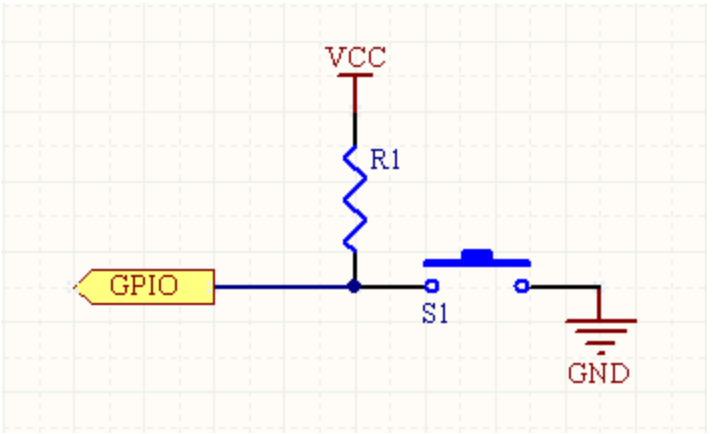


- What state does the pin have when switch is open?
  →floating (high-impedance/tri-stated)

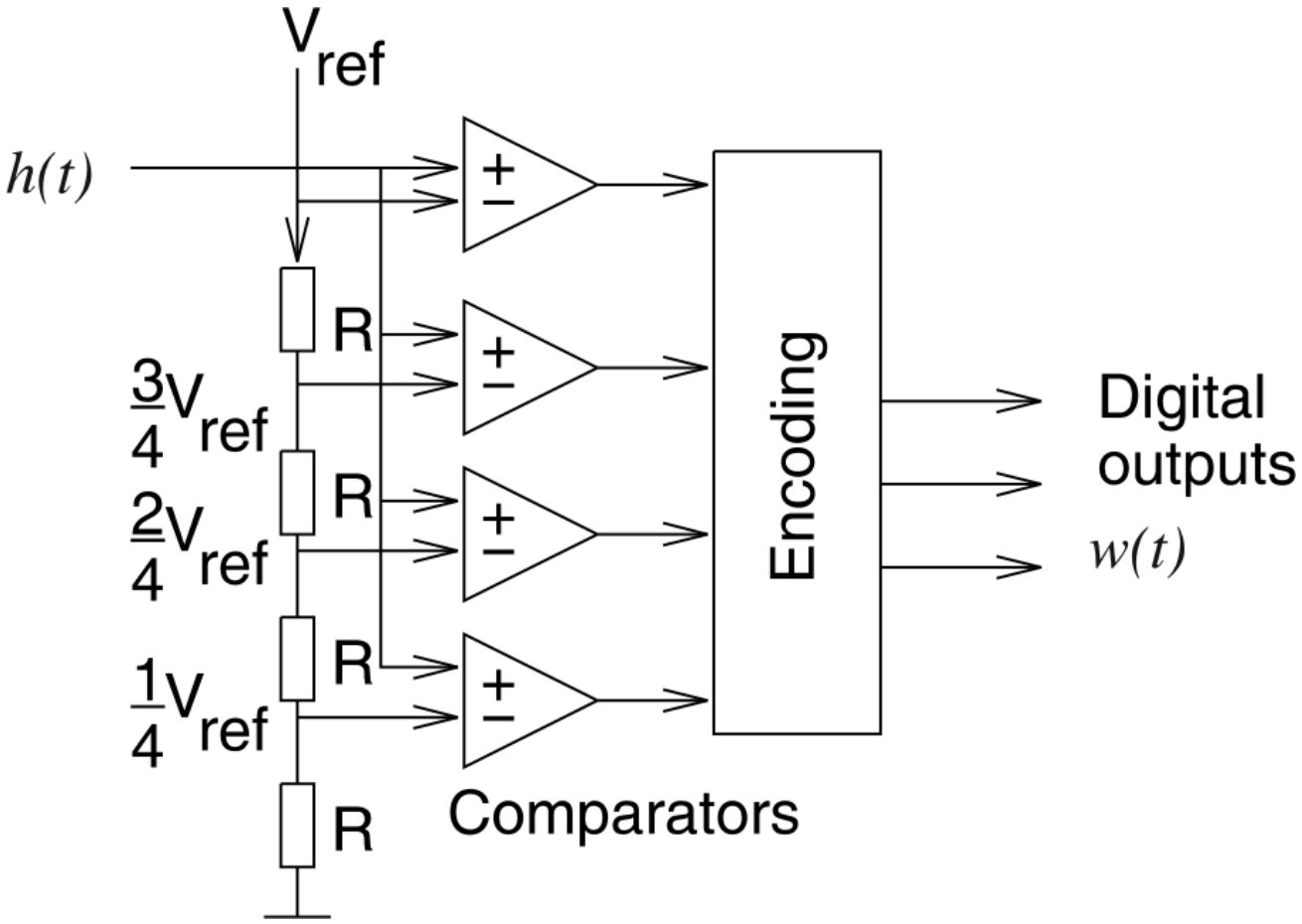
GPIO cont‘d

- Pull-up resistor is needed for switch-open-state

- Most microcontrollers offer configurable pull-up and/or pull-down resistors

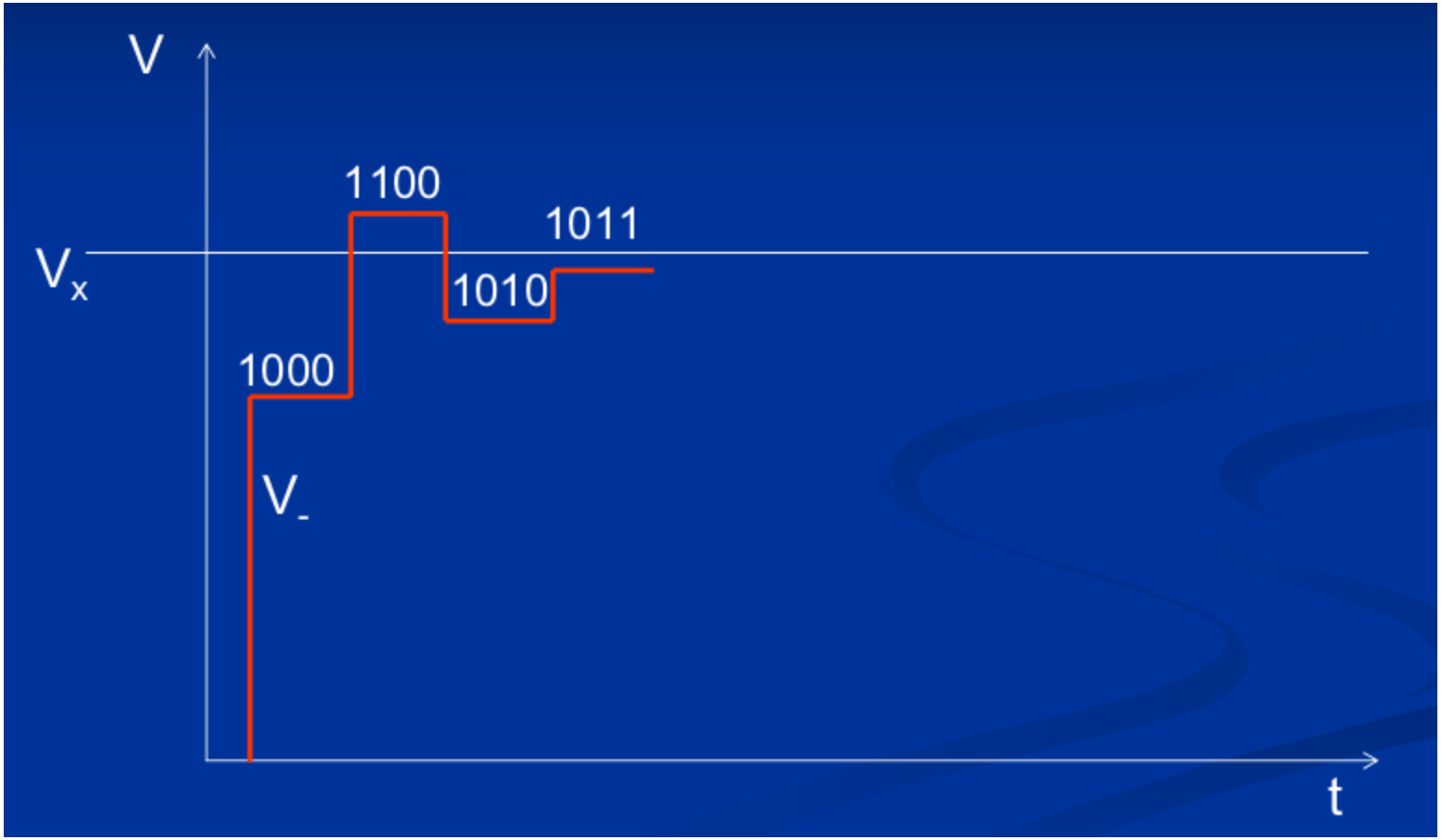


Analog-to-Digital Converter (ADC)



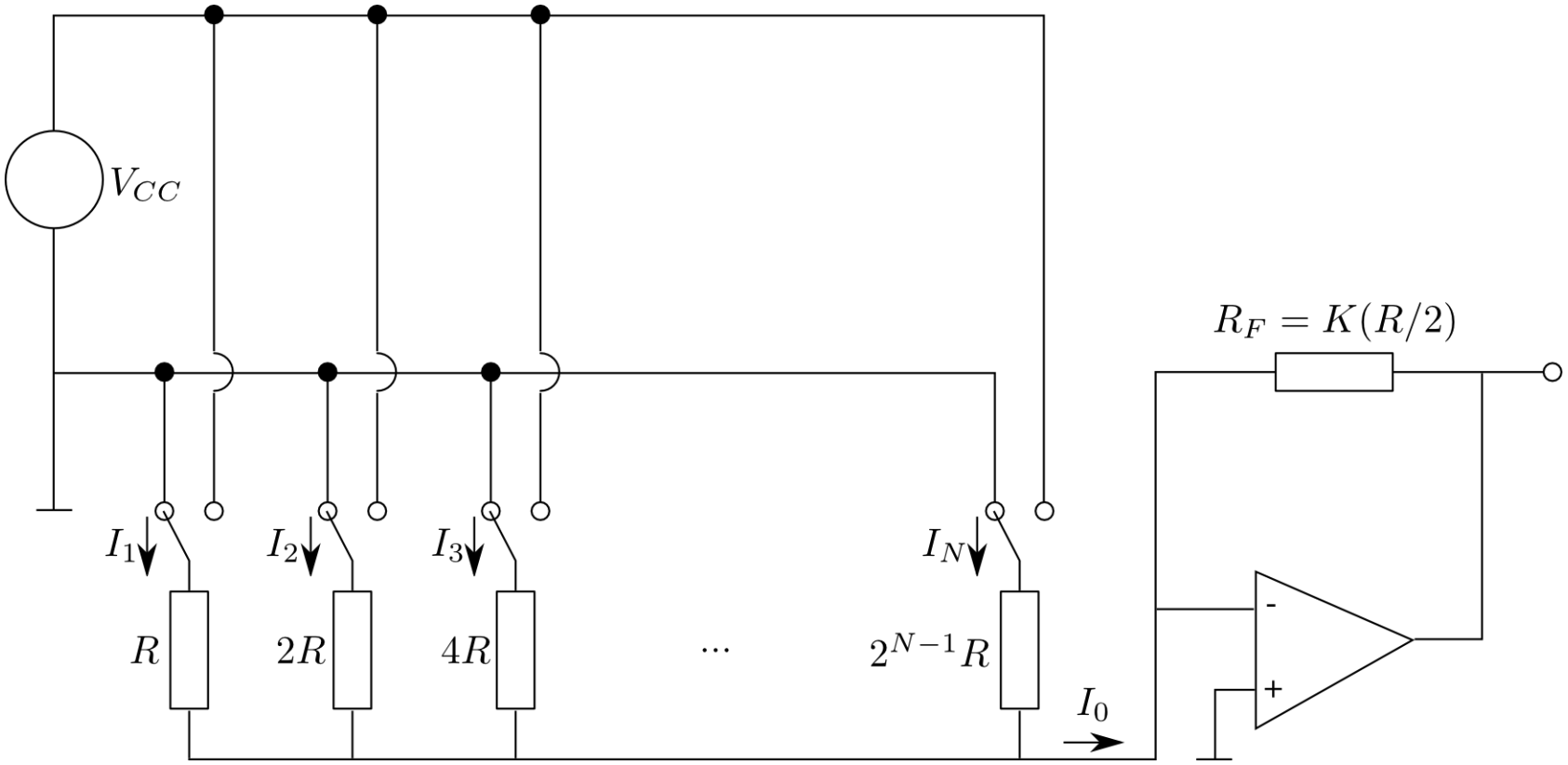
- Encodes input number of most significant ‘1’ as an unsigned number, e.g. “1111” -> “100”, “0111” -> “011”, “0011” -> “010”, “0001” -> “001”, “0000” -> “000” (priority encoder)

ADC: Successive Approximation

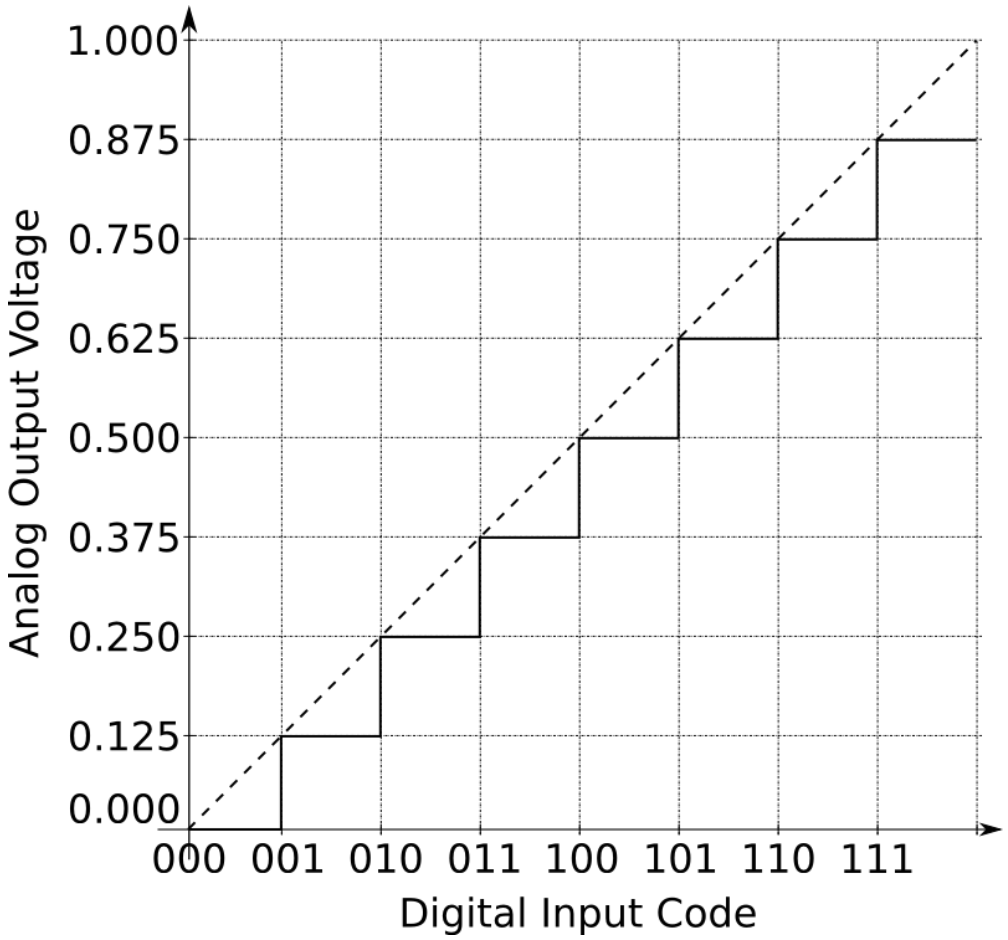


Digital-Analog Converter (DAC)

- Current Scaling: Binary-weighted resistor DAC implementation



DAC Example



- Example of a 3-bit DAC with normalized output voltage

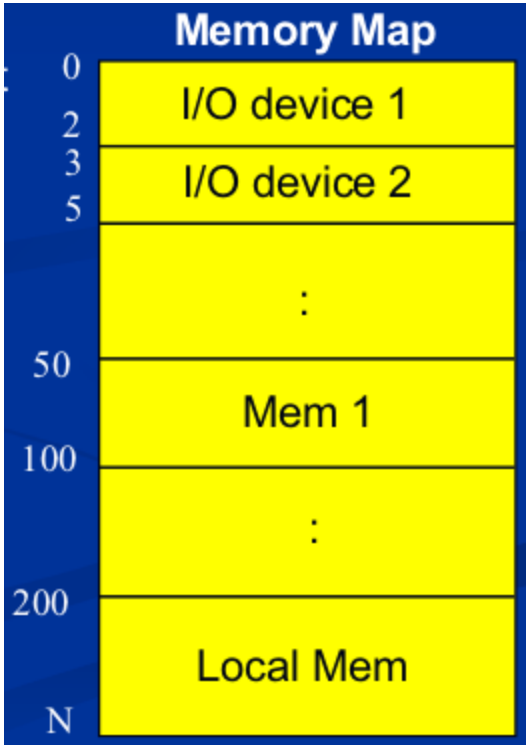
Outline

- Processor
- Memory
- I/O
  - PWM, GPIO, DAC, ADC
  - Interfacing I/O: polling vs. interrupt

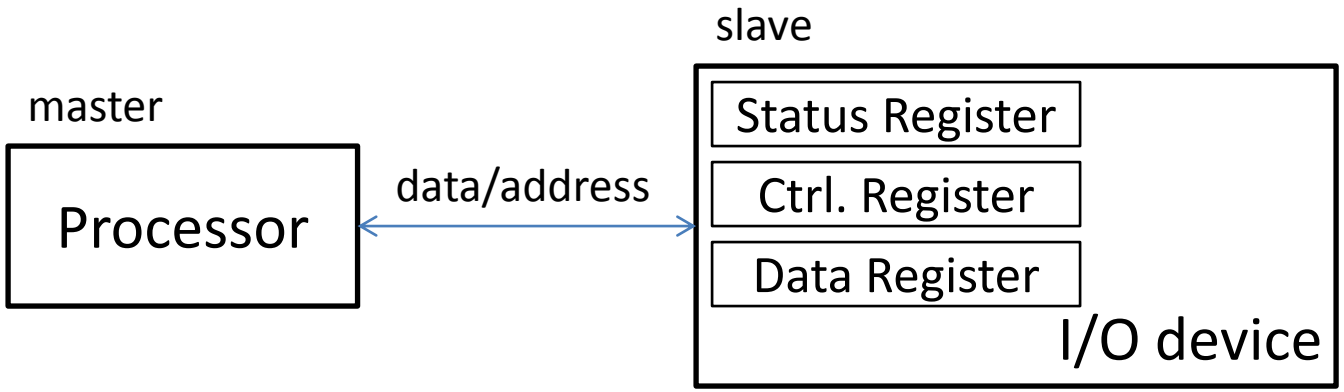
Interfacing I/O Devices to Processor

- I/O Devices are mapped in the address space of Computation Components (CC), i.e., CCs communicate with I/O devices simply by reading/writing from/to memory cells.

- Two general methods for access
  - Polling - includes, a kind of busy-wait loop
  - Interrupts - using interrupt line of CC to provide service to devices



Polling: Busy-Wait Interface



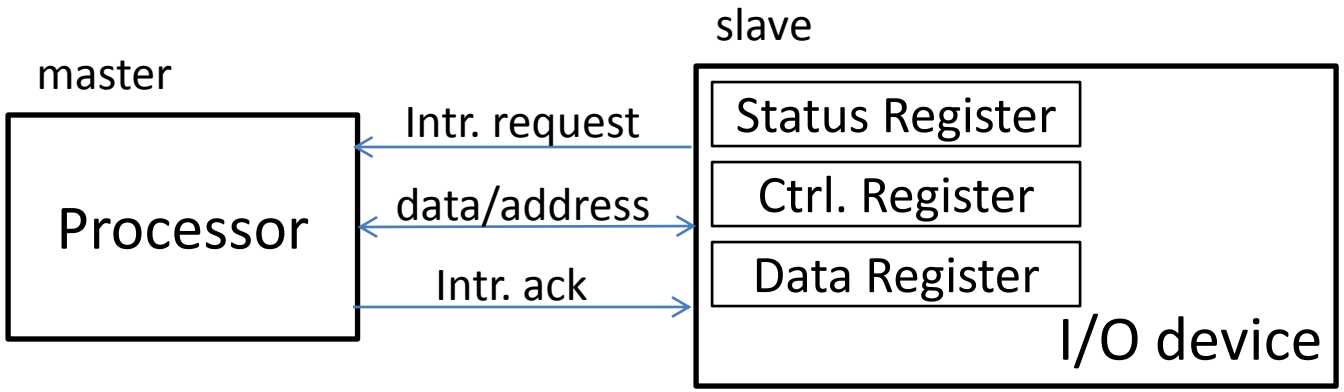
- Continuous polling

```
While (!device ready) {
    Check device
}
```

- Periodic polling

- Issues:
  - MASTER is tied up in communication with device until I/O operation is done
  - No other work can be accomplished by MASTER
  - Typically one MASTER in system, but many I/O devices
  - Only really useful if devices are fast
    - No time consuming context switches

Interrupt



- Procedure
  - When I/O device needs MASTER attention, an interrupt signal is sent
  - MASTER is "forced" to suspend its current task
    - Interrupts can be ignored (masked) when critical task is executed
  - MASTER acknowledges interrupt and jumps to interrupt service routine
  - When finished control is returned to the interrupted task

- Key observations:
  - MASTER is free to do something else until attention is needed
  - Improves utilization of MASTER, leading to a much better response
  - I/Os can proceed asynchronously
  - Time consuming context switches

Polling vs. Interrupt

Polling

- Pros
  - Short latencies (low number of IOs)
  - Many events do not block the normal execution of the program
- Cons
  - Most polls are unnecessary
  - High CPU usage
  - Reaction time depends on number of IOs

Interrupt

- Pros
  - Processor resources are only used when needed
- Cons
  - Program execution is interrupted in a non-deterministic manner