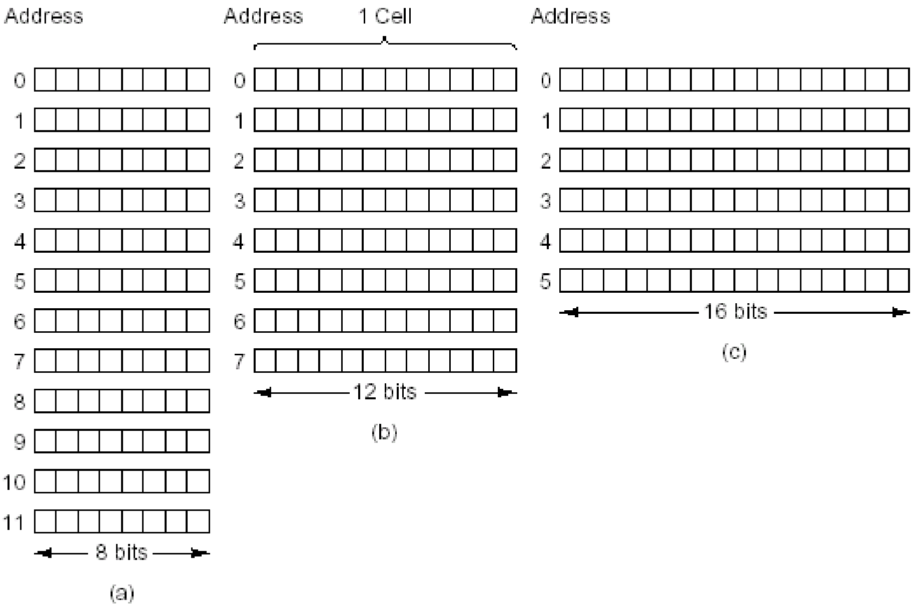
Memory System Organization and Architecture

Primary memory (RAM)

- Memory is the workspace for the processor
- Collection of storage cells, together with associated circuits needed to transfer information in and out of storage cells
- Programs and data they access must be in RAM before the processor can execute them
- RAM is temporary (volatile)
- Consists of a no. of cells, each having a number (address)
- *n* cells → addresses: 0 to *n*-1
- Same no. of bits in each cell
- Adjacent cells have consecutive addresses
- *m*-bit address ⇒ 2^m addressable cells

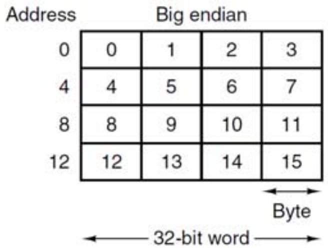
Example: (96-bit memory)



Byte ordering

- The bytes in a word can be numbered from left-to-right or right-to-left
- Big endian system
  - numbering begins at big (high-order) end
  - Used in IBM mainframes

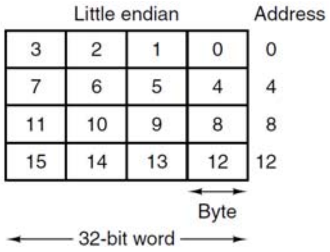
Example: Memory of a 32-bit computer



Address	Big endian			
0	0	1	2	3
4	4	5	6	7
8	8	9	10	11
12	12	13	14	15

- Little endian system
  - numbering begins at little (low-order) end
  - Used in Intel processors

Example: Memory of a 32-bit computer



Little endian				Address
3	2	1	0	0
7	6	5	4	4
11	10	9	8	8
15	14	13	12	12

Note:

- Both systems represent the value of a 32-bit integer in the rightmost *n* bits of a word and zeros in the leftmost (32-*n*) bits
- Example: representation of 12
  - Big endian: $\underset{0}{\underline{00000000}}\ \underset{1}{\underline{00000000}}\ \underset{2}{\underline{00000000}}\ \underset{3}{\underline{00001100}}$
    byte number
  - Little endian: $\underset{3}{\underline{00000000}}\ \underset{2}{\underline{00000000}}\ \underset{1}{\underline{00000000}}\ \underset{0}{\underline{00001100}}$
    byte number

Example:

A movie record represented in a 32-bit system

Title (string): The Matrix
Playing time in minutes (integer): 136
Year (integer): 1999

Big endian

0	T	h	e	
4	M	a	t	r
8	i	x	0	0
12	0	0	0	136
16	0	0	7	207

Little endian

	e	h	T	0
r	t	a	M	4
0	0	x	i	8
0	0	0	136	12
0	0	7	207	16

Types of memory

- ROM (Read-only memory)
  - Built in to the motherboard or video cards
- DRAM (Dynamic RAM)
  - This is the type you usually purchase and install
- SRAM (Static RAM)
  - Built in to the CPU

ROM

- Permanent (nonvolatile)
- Read-only operations
- Ideal place to put PC’s boot instructions
- Main BIOS is contained in a ROM chip
- ROM types: PROM, EPROM, EEPROM

DRAM

- Used as main memory in modern PCs
- High density and inexpensive
- Dynamic
  - Content changes with every keystroke or mouse swipe
- Constructed using array of cells, each consisting a transistor and a capacitor
- Capacitors can be charged or discharged, allowing 0s and 1s to be stored
- Electric charge tends to leak out ⇒ each bit must be reloaded (**refreshed**) every few milliseconds (15ms) to prevent data from leaking away
- Refreshing takes several CPU cycles to complete
  - 10% in older systems
  - 1% in modern computers

SRAM

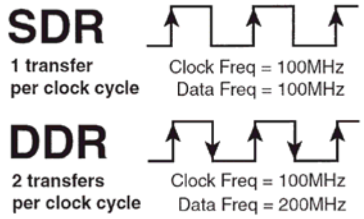
- No need to refresh as in DRAM
- Cluster of 6 transistors for each bit of storage
- Very fast, low density, expensive
- Contents are retained as long as power is kept on
- Used to construct cache memory

SDRAM (Synchronous DRAM)

- A type of DRAM
- Runs in synchronization with the system bus
- Driven by a single synchronous clock
- Used main memories
- One transfer per clock cycle

DDR (Double Data Rate) SDRAM

- An upgrade to standard SDRAM
- Performs 2 transfers per clock cycle without doubling clock rate
- Low power consumption (2.5v) than standard SDRAM
- Available in dual-channel mode



Memory errors

- Hard errors
  - Permanent failure
  - How to fix? (replace the chip)
- Soft errors
  - Non-permanent failure
  - Occurs at infrequent intervals
  - How to fix? (restart)

- Best way to deal with soft errors is to increase system's fault tolerance (implement ways of detecting and correcting errors)

Techniques used for fault tolerance

- Parity
- ECC (Error Correcting Code)

Parity checking

- 9 bits are used in the memory chip to store 1 byte (12% increase in cost)
- Extra bit (parity bit) keeps tabs on other 8 bits
- Parity can only detect errors, but cannot correct them

Odd parity standard for error checking

- Parity generator/checker is a part of CPU or located in a special chip on motherboard
- Parity checker evaluates 8 data bits by adding the no. of 1s in the byte
- If an even no. of 1s is found, parity generator creates a 1 and stores it as the parity bit
- If the sum is odd, parity bit would be 0

- If a (9-bit) byte has an even no. of 1s, that byte must have an error
- Cannot tell which bit or bits have changed
- If 2 bits changed, bad byte could pass unnoticed
- Multiple bit errors in a single byte are very rare

Example (even parity):

Consider the 8-bit number: 1 0 0 1 1 0 1 0

No. of 1s = 4 (even)
Parity bit = 0
9-bit number = 1 0 0 1 1 0 1 0 0

ECC

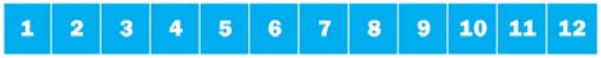
- Successor to parity checking
- Can detect and correct memory errors
- Only a single bit error can be corrected though it can detect double-bit errors
- Most standard PC processors, chipsets and memory modules do not support ECC
- Used in server systems

The Hamming Single Error Correcting Code:

- Invented by Richard Hamming
- No. of additional parity bits are required for single error correction
  - 4 check bits for a single byte
  - 7 check bits for a 4-byte word
  - 8 check bits for a 8-byte word

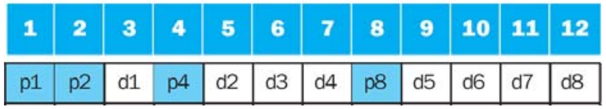
Steps to calculate Hamming ECC for a byte (4 parity bits):

- Start numbering bits from 1 on the left



1	2	3	4	5	6	7	8	9	10	11	12

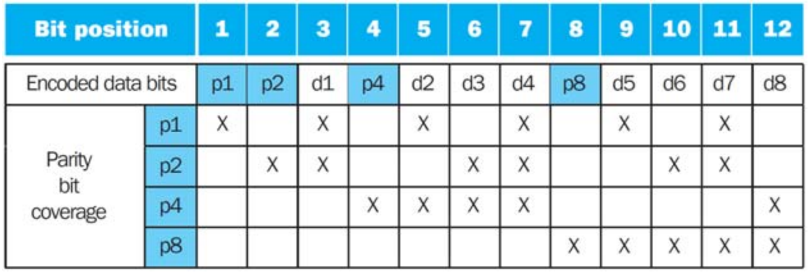
- Mark all bit positions that are power of 2 as parity bits (1, 2, 4, 8)
- All other bit positions are used for the data (3, 5, 6, 7, 9, 10, 11, 12)



1	2	3	4	5	6	7	8	9	10	11	12
p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8

Steps to calculate Hamming ECC for a byte (4 parity bits):

- Position of parity bits
  - Bit 1 (0001) check bits are at 1, 3, 5, 7, 9, 11, 13, 15 (rightmost bit of address is 1 – ie is 0001, 0011, 0101, 0111, 1001, 1011, 1101, 1111)
  - Bit 2 (0010) check bits are at 2, 3, 6, 7, 10, 11, 14, 15 (second bit to the right is 1)
  - Bit 4 (0100) check bits are at 4-7, 12-15 (third bit to the right is 1)
  - Bit 8 (1000) check bits are 8-15 (fourth bit to the right is 1)
- Set parity bits to create even parity for each group



Bit position		1	2	3	4	5	6	7	8	9	10	11	12
Encoded data bits		p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8
Parity bit coverage	p1	X		X		X		X		X		X	
	p2		X	X			X	X			X	X	
	p4				X	X	X	X					X
	p8								X	X	X	X	X

Example:

- Consider the byte: 1 1 0 0 1 0 0 1
- Insert parity positions (_): _ _ 1 _ 1 0 0 _ 1 0 0 1
- Bit 1 positions (red): _ _ 1 _ 1 0 0 _ 1 0 0 1 [odd 1s, parity = 1]
  - 1 _ 1 _ 1 0 0 _ 1 0 0 1
- Bit 2 positions (red): 1 _ 1 _ 1 0 0 _ 1 0 0 1 [odd 1s, parity = 1]
  - 1 1 1 _ 1 0 0 _ 1 0 0 1
- Bit 4 positions (red): 1 1 1 _ 1 0 0 _ 1 0 0 1 [even 1s, parity = 0]
  - 1 1 1 0 1 0 0 _ 1 0 0 1
- Bit 8 positions (red): 1 1 1 0 1 0 0 _ 1 0 0 1 [even 1s, parity = 0]
  - 1 1 1 0 1 0 0 0 1 0 0 1
- Now, invert bit 5: 1 1 1 0 0 0 0 0 1 0 0 1
- Parity bit 1: 1 1 1 0 0 0 0 0 1 0 0 1 [odd 1s, ERROR]
- Parity bit 2: 1 1 1 0 0 0 0 0 1 0 0 1 [even 1s, OK]
- Parity bit 4: 1 1 1 0 0 0 0 0 1 0 0 1 [odd 1s, ERROR]
- Parity bit 8: 1 1 1 0 0 0 0 0 1 0 0 1 [even 1s, OK]
- Parity bit 1 and 4 raise error
- Therefore, bit number 5 (= 1 + 4) has an error, and invert that bit to correct the error!

Cache memory:

- A high-speed, small memory
- Most frequently used memory words are kept in
- Based on the principle of locality
  - Programs access a relatively small portion of their address space at any instant of time
- Temporal locality
  - If an item is referenced, it will tend to be referenced again soon
- Spatial locality
  - If an item is referenced, items whose addresses are close by will tend to be referenced soon

Example:

- Computer programs exhibit both temporal and spatial locality

- **Cache lines** – blocks inside the cache
- **Cache hit** – requested information is available in the cache
- **Cache miss** - requested information is not available in the cache
- **Miss penalty** – no. of cycles CPU has to wait until the requested data are brought in on a cache miss

- Main memories and caches are divided into fixed sized blocks
- On a cache miss, entire cache line is loaded into cache from memory

Example:

- 64K cache can be divided into 1K lines of 64 bytes, 2K lines of 32 bytes etc
- In a 64-byte cache line size, a reference to memory address 270 would pull the line containing bytes 256 to 319 into cache line

    0 – 63, 64 – 127, 128 – 191, 192 – 255, 256 - 319

Unified cache

- instruction and data use the same cache

Split cache

- instructions in one cache and data in another

Levels of caches

- Level 1 (L1) cache
  - Directly built into processor die
  - Runs at full speed of the processor
  - A split cache (L1-D and L1-I)
  - Smallest cache

- Level 2 (L2) cache
  - Unified cache
  - High capacity than L1
  - Built into processor die in modern systems (runs at the full CPU speed)
- Level 3 (L3) cache
  - Present in Intel Core-i family of processors
  - Built into the die
  - Unified cache
  - Largest cache
  - Shared to all processors

Let

h – hit rate (fraction of all references that can be satisfied out of cache)

⇒ miss rate = 1- h

Average memory access time (AMAT)

= cache access time + miss rate x miss penalty

h = 1 ⇒ no memory references

h = 0 ⇒ all are memory references

Example:

Find AMAT for a processor with a 2 ns clock cycle time, a miss penalty of 80 cycles, a miss rate of 0.05 misses per instruction, and a cache access time of 1 clock cycle.

Answer:

AMAT = cache access time + miss rate x miss penalty

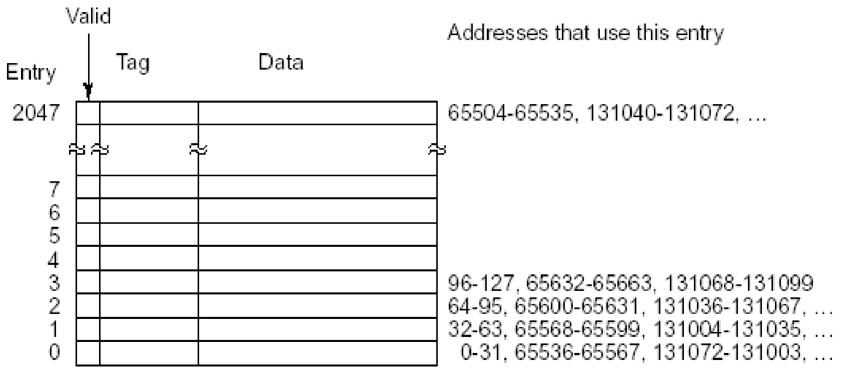
= 1 + 0.05 x 80

= 5 clock cycles

= 10 ns

Direct-mapped caches

- Single level direct-mapped cache



- Example cache contains 2048 entries
- Each entry (row) in cache can hold exactly one cache line from main memory

    32-byte cache line size ⇒ 64KB cache

- **Valid** bit: indicates whether there is valid data in this entry (invalid at boot time)
- **Tag** field: 16-bit value identifying the line of memory from which data came
- **Data** field contains a copy of data in memory (32 bytes here)
- Given memory word is stored in exactly one place within cache

- To store/retrieve data from cache, memory address is divided into 4 components **(32-bit virtual address)**

Bits	16	11	3	2
	TAG	LINE	WORD	BYTE

- **TAG** – corresponds to **Tag** bits stored in cache entry
- **LINE** – indicates which cache entry holds data
- **WORD** – tells which word within a line is referenced
- **BYTE** – if only a single byte is requested, it tells which byte within the word is needed

When CPU produces a memory address

- Hardware extracts 11 LINE bits from address, indexes into cache, finds corresponding cache entry
- Valid entry? → TAG field of memory is compared with Tag field of cache
  - If agree, word requested is extracted from cache
- Invalid entry or tags do not agree? → 32-byte cache line is fetched from memory and stored in cache entry, discarding what was there
- Puts consecutive memory lines in consecutive cache entries
  - Two memory lines that differ in their address by 64K cannot be stored in cache at the same time

Example:

Consider the following sequence of memory references of a 32 bytes memory to an eight-block cache with block size of 1 byte.

10110, 11010, 10110, 11010, 10000, 00011, 10000, 10010

Show the content of the cache for each cache miss.

Answer:

TAG: 2 bits, LINE: 3 bits

	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

Initially after power on

	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10	Memory (10110)
111	N		

Cache miss for memory access 10110

	V	Tag	Data
000	N		
001	N		
010	Y	11	Memory (11010)
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory (10110)
111	N		

Cache miss for memory access 11010

	V	Tag	Data
000	Y	10	Memory (10000)
001	N		
010	Y	11	Memory (11010)
011	N		
100	N		
101	N		
110	Y	10	Memory (10110)
111	N		

Cache miss for memory access 10000

	V	Tag	Data
000	Y	10	Memory (10000)
001	N		
010	Y	11	Memory (11010)
011	Y	00	Memory (00011)
100	N		
101	N		
110	Y	10	Memory (10110)
111	N		

Cache miss for memory access 00011

	V	Tag	Data
000	Y	10	Memory (10000)
001	N		
010	Y	10	Memory (10010)
011	Y	00	Memory (00011)
100	N		
101	N		
110	Y	10	Memory (10110)
111	N		

Cache miss for memory access 10010

Handling writes:

Write-through

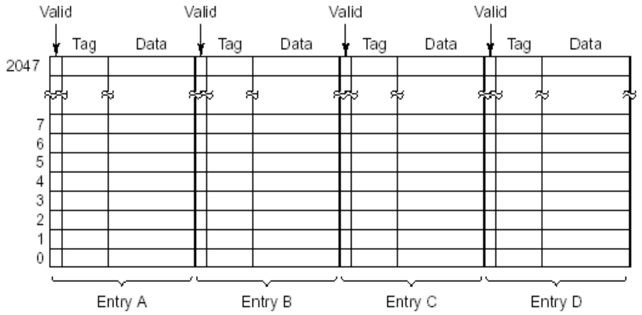
- Data are written to both the cache and the memory at the same time
  - Memory is up to date, Time consuming

Write-back

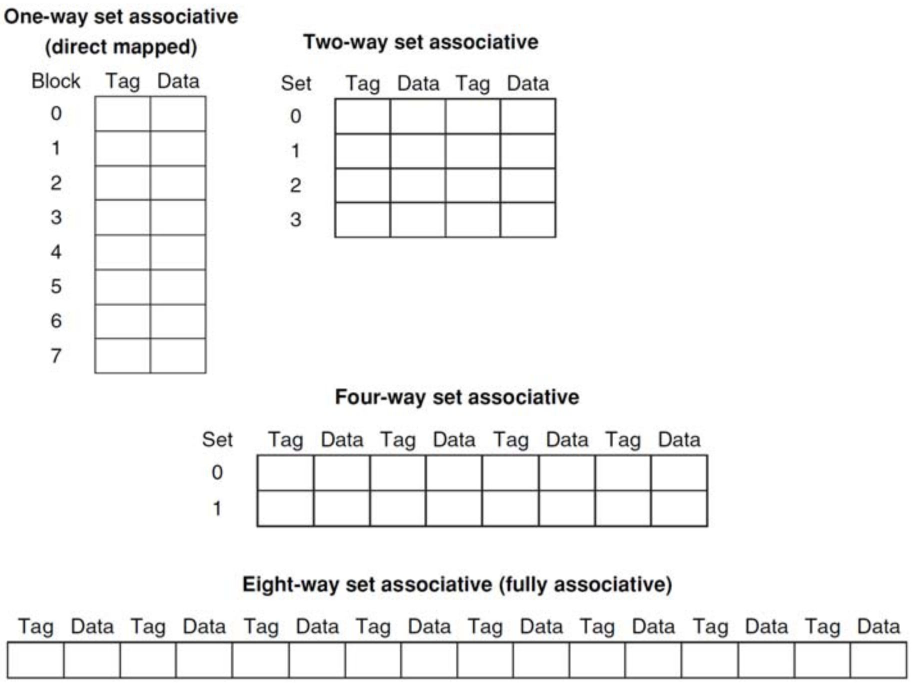
- Data are only written to main memory when it is forced out of cache

Set associative caches

- A cache with *n* possible entries for each address is called an *n*-way set associative cache
- 4-way set associative cache (2048 addresses)



- More complicated than a direct-mapped cache
  - n cache entries have to be checked
- 2-way, 4-way caches perform well enough to make this extra circuitry worthwhile
- Direct-mapped cache: one-way set associative
  - A block is placed in exactly one location in the cache
- Fully associative cache: N-way set associative
  - N - total no. of blocks in the cache
  - A block can be placed any location in the cache



When a new entry is brought into cache, which of present items should be discarded?

LRU (Least Recently Used) algorithm is used

- The block that has been unused for the longest time is replaced

CPU time = (CPU clock cycles + Memory stall cycles) x Clock cycle time

Memory stall cycles = No. of misses x Miss penalty

= IC x Misses per instruction x Miss penalty

Example:

Miss rate of instruction cache = 2%
Miss rate of data cache = 4%
Processor has a CPI of 2 without any memory stalls
Miss penalty = 100 cycles for all misses
Frequency of data access = 36%
How much faster is the processor with perfect cache that never misses?

Answer:

Instruction miss cycles = IC x 0.02 x 100 = 2 x IC

Data miss cycles = IC x 0.36 x 0.04 x 100 = 1.44 x IC

Memory stall cycles = 2 x IC + 1.44 x Ic = 3.44 x IC

CPU Time$_{\text{cache}}$ = (2 + 3.44) x IC x Cycle time = 5.44 x IC x Cycle time

CPU Time$_{\text{perfect cache}}$ = 2 x IC x Cycle time

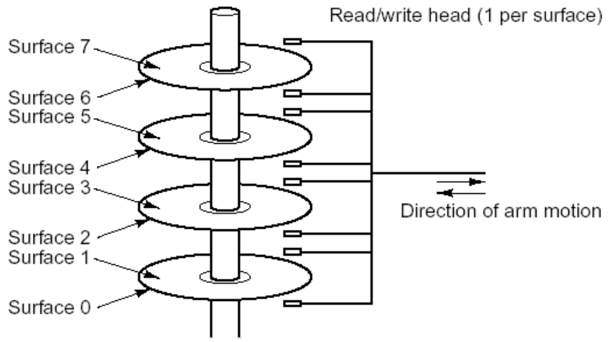
Speedup = 5.44/2 = 2.72

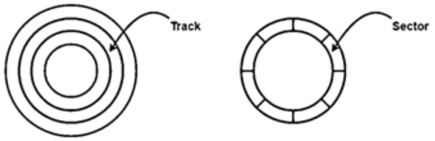
Secondary memory

- Nonvolatile memory used to store programs / data between runs
  - Magnetic disk (and SSD) in desktop computers and servers
  - Flash memory in mobile devices

Magnetic disks

- Disk platters are divided in to concentric rings called tracks
- Tracks are divided into sectors
- Cylinder: same set of contiguous tracks on each platter





Disk performance depends on

- **seek time** - time to move arm to desired track
- **rotational latency** – time needed for requested sector to rotate under head
  - Rotational speed: 5400, 7200, 10000, 15000 rpm

Average rotation time = $\frac{0.5}{\text{rpm}}\ \text{minutes} = \frac{0.5 \text{ x } 60 \text{ x } 1000}{\text{rpm}}\ \text{ms}$

- **transfer time** – time needed to transfer a block of bits under head (e.g., 100 MB/s)
- **Disk controller**
  - chip that controls the drive. Its tasks include accepting commands (READ, WRITE, FORMAT) from software, controlling arm motion, detecting and correcting errors
- **controller time**
  - overhead the disk controller imposes in performing an I/O access

Avg. disk access time = avg. seek time + avg. rotational delay
+ transfer time + controller overhead

Example:

Advertised average seek time of a disk is 5ms, transfer rate is 100 MB per second, and it rotates at 7,200 rpm. Controller overhead is 1ms. Calculate the average time to read a 512-byte sector.

Average time = $5 + \frac{0.5 \text{ x } 60 \text{ x } 1000}{7200} + \frac{512 \text{ x } 1000}{100 \text{ x } 1024 \text{ x } 1024} + 1$

= 5 + 4.167 + 0.005 + 1 = 10.172ms