Computer Memory

Michael Dann
School of Computing Technologies

What's next…

Computer Memory

Main takeaways:

- Memory = information storage. Lots of different mechanisms, but they all essentially do the same thing (store 1s and 0s).
- There's a trade-off between speed and capacity.
- Modern memory uses a lot of tricks to speed things up (DDR, pipelining, prefetching).

What are the main types of memory on a computer today?

NAME	SIZE	SPEED	VOLATILE?
1) CPU / CACHE	TINY	VERY FAST	YES
2) RAM	SMALL	FAST	YES
3) HDD / SDD	HUGE	VERY SLOW	NO

...

How is memory structured?

- Collection of bits (1s and 0s)
- Often more convenient to think in terms of "words" since this is how memory is addressed and transferred.
- Different machines have different word sizes (8-bit, 16-bit, 32-bit, 64-bit).

Question:

- How many megabytes are there in a gigabyte?

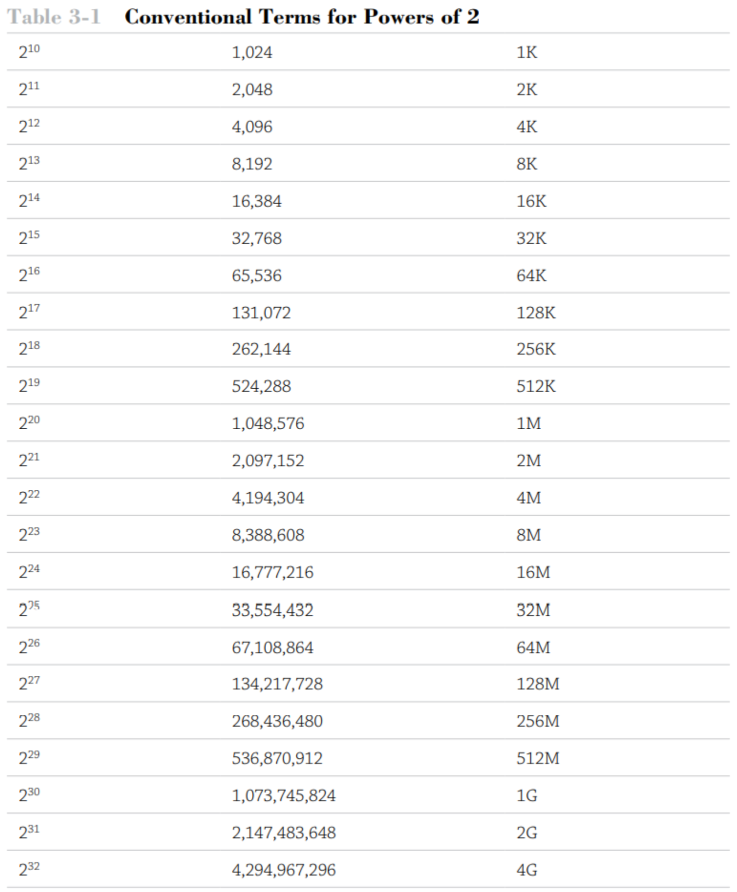


Table 3-1 **Conventional Terms for Powers of 2**

2^{10}	1,024	1K
2^{11}	2,048	2K
2^{12}	4,096	4K
2^{13}	8,192	8K
2^{14}	16,384	16K
2^{15}	32,768	32K
2^{16}	65,536	64K
2^{17}	131,072	128K
2^{18}	262,144	256K
2^{19}	524,288	512K
2^{20}	1,048,576	1M
2^{21}	2,097,152	2M
2^{22}	4,194,304	4M
2^{23}	8,388,608	8M
2^{24}	16,777,216	16M
2^{25}	33,554,432	32M
2^{26}	67,108,864	64M
2^{27}	134,217,728	128M
2^{28}	268,436,480	256M
2^{29}	536,870,912	512M
2^{30}	1,073,745,824	1G
2^{31}	2,147,483,648	2G
2^{32}	4,294,967,296	4G

Static RAM (SRAM) vs Dynamic RAM (DRAM)

- Historically, there have been two main types of RAM: Static RAM and Dynamic RAM.

- Both types are volatile – when power is switched off, their contents are lost.

- Static RAM – Common in the early days (60s).
  - "Static" = doesn’t need to be refreshed. Once you write data, it persists so long as power is on.
  - Faster to access than DRAM.
  - More expensive due to circuitry and hard to shrink size.
  - Today, this type of technique is used for cache and registers on CPU.

- Dynamic RAM – Cheaper and can be mass produced.
  - Must be periodically refreshed (read then immediately rewritten). This makes it more power hungry.
  - Today, used for computer's main memory.

Synchronous DRAM (SDRAM)

Older DRAM is “asynchronous” – as soon as an input control signal is sent, it’s acted upon.

SDRAM achieves a dramatic speed improvement through *pipelining* (overlapping operations to hide latency).

- Multiple “banks” that can be operated on independently.
- Can accept a new command for one bank while still processing a command to another bank.
- Timing of changes in control inputs needs to be consistent for this to work.
    - Coordinate input controls via a *clock* (regular electric pulses, essentially).
    - This is where the term “synchronous” comes from.

SDR vs DDR RAM

- SDRAM is synchronous, meaning it relies on a clock to synchronize signals, creating orderly cycles of data fetches and writes.
- Single Data Rate (SDR SDRAM)
  - Memory transfers occur on the rising part of the clock cycle only.
- Double Data Rate (DDR SDRAM)
  - Memory transfers happen on the rising and falling parts of the clock cycle.
- DDR2, DDR3, DDR4, …
  - Speed increased in various ways.

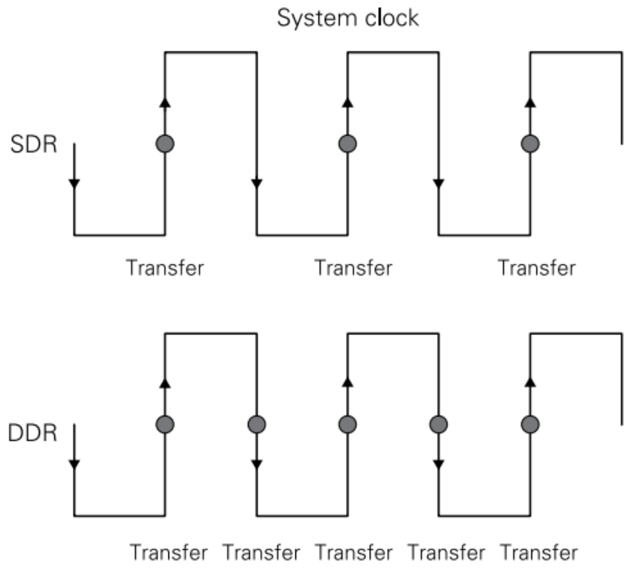


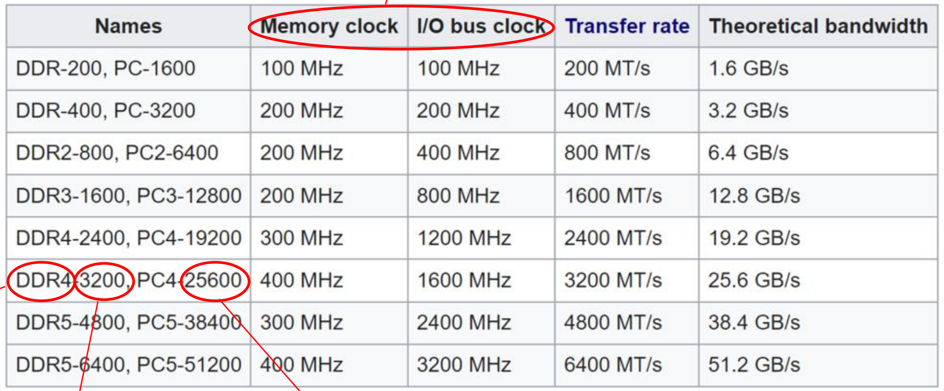
FIGURE 3-6: SDR vs. DDR timing

Understanding memory specs

Two main metrics for memory performance:

- **Bandwidth** (Amount of data that can be transferred per unit time).
- **Access time** (The time for a memory access to complete after the CPU requests it).

The difference is a bit subtle. It's due to *prefetching*, as explained in the textbook (pages 67-68).



Names	Memory clock	I/O bus clock	Transfer rate	Theoretical bandwidth
DDR-200, PC-1600	100 MHz	100 MHz	200 MT/s	1.6 GB/s
DDR-400, PC-3200	200 MHz	200 MHz	400 MT/s	3.2 GB/s
DDR2-800, PC2-6400	200 MHz	400 MHz	800 MT/s	6.4 GB/s
DDR3-1600, PC3-12800	200 MHz	800 MHz	1600 MT/s	12.8 GB/s
DDR4-2400, PC4-19200	300 MHz	1200 MHz	2400 MT/s	19.2 GB/s
DDR4-3200, PC4-25600	400 MHz	1600 MHz	3200 MT/s	25.6 GB/s
DDR5-4800, PC5-38400	300 MHz	2400 MHz	4800 MT/s	38.4 GB/s
DDR5-6400, PC5-51200	400 MHz	3200 MHz	6400 MT/s	51.2 GB/s

RAM generation.

Transfer rate (megatransfers per second).

Theoretical bandwidth (MB/s).

Access time typically specified in terms of *CAS latency* = how many clock cycles in it takes for the RAM module to access data in one of its columns.

Since it's expressed in clock cycles, need to take clock speed into account to get true latency.

Memory Cache

- Caching is a widely applicable technique in computing that isn't just used for memory.
    - For example, it's widely used on the web too.

- The basic idea is to store a copy of frequently-requested data in a place that's faster to access than the original source.

- It's effectively invisible to most programs.

- Modern microprocessors have multiple layers of cache, generally at least 2. Super fast memory, but expensive, so not much of it.

- The first layer, L1, is the smallest (only kilobytes) and fastest. The layers get progressively larger and slower down the hierarchy.

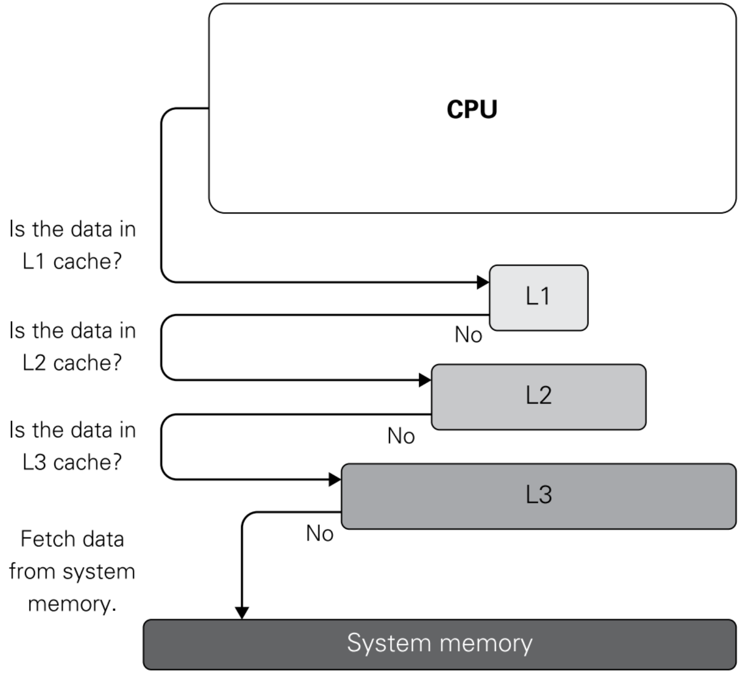


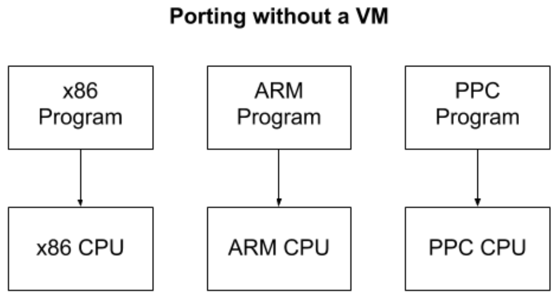
FIGURE 3-8: A multi-level cache

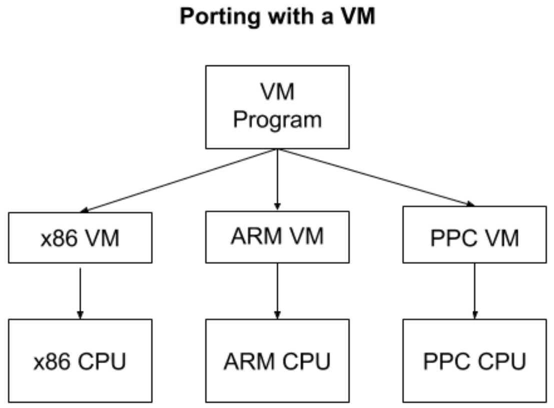
What is a virtual machine?

- **A VM is a piece of software that acts like a computer.**
- It simulates a CPU and possibly some other hardware components, allowing it to perform arithmetic, read and write to memory, etc, just like a physical computer.
- Why use VMs?
  - <u>Portability</u>
    Code only needs to run on one (imaginary) machine.
    Example: The Java Virtual Machine (JVM).
  - <u>Emulation</u>
    Mimic old hardware that is hard to come by.

Portability

- Imagine you want to write a program that can run on multiple computer architectures. VMs can provide a standard platform that provides portability for all of them.
- The most famous example is the Java Virtual Machine (JVM).
- There's a formal specification that describes what is required in a JVM implementation. Programmers need not worry about the idiosyncrasies of the underlying hardware platform – they just need to write for the JVM.
- Drawback: There's an overhead from running the VM, which doesn't exist with compiled code.
- Note that VMs and compilers address portability in different ways:
    - A compiler translates a high-level language to instructions for a target CPU architecture.
    - A VM creates one standard CPU architecture that is simulated on various hardware devices.





Emulation

- The aim of an emulator is to simulate all components of a real hardware device, such as an old video game console, as closely as possible.
- Focus is typically on *faithful* simulation, even if this includes “bad” features of the original device, such as slowdown when there are too many sprites on the screen.
- Allows you to play old Nintendo games on your PC!

Mesen - Super Mario Bros. 3 (USA) (Rev 1)
File Game Options Tools Debug Help
SUPER MARIO BROS. 3
TM
1 PLAYER GAME
2 PLAYER GAME
© 1988 Nintendo

Video game emulation

With *Mesen* (a NES emulator), not only can you play old Nintendo games, but you can also view the games' assembly instructions, the CPU status, the values stored in memory, etc.

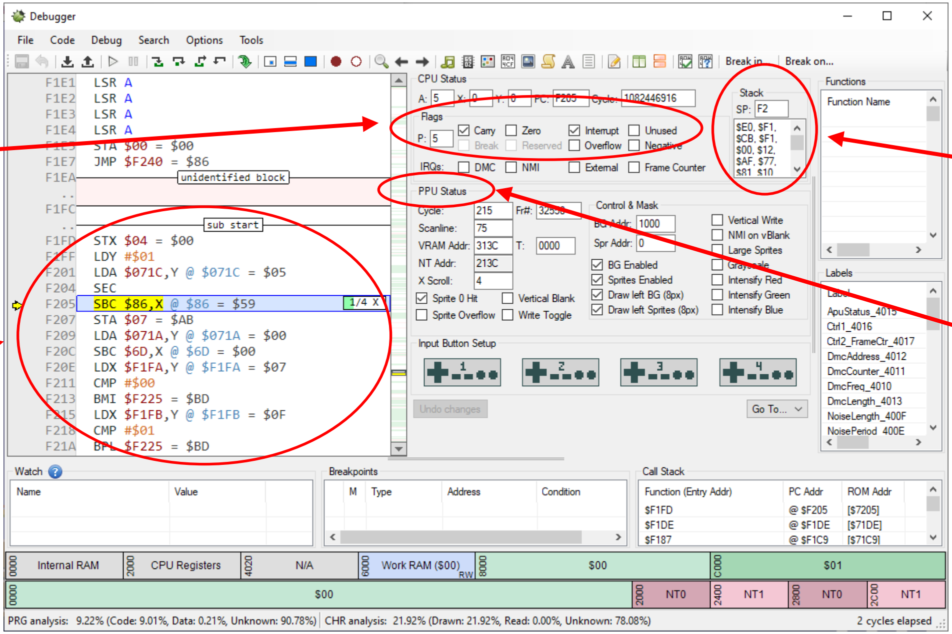
More flags than just n/z/p.

Note that NES instructions look a bit different to LC-3.

If you really know what you're doing, you can create your own modified ROMs.

Something we miss in LC-3!

PPU = "picture processing unit".



```
F1E1  LSR A
F1E2  LSR A
F1E3  LSR A
F1E4  LSR A
F1E5  STA $00 = $00
F1E7  JMP $F240 = $86
F1EA ---------- unidentified block ----------
  ..
F1FC
  ..  ---------- sub start ----------
F1FD  STX $04 = $00
F1FF  LDY #$01
F201  LDA $071C,Y @ $071C = $05
F204  SEC
F205  SBC $86,X @ $86 = $59          1/4 X
F207  STA $07 = $AB
F209  LDA $071A,Y @ $071A = $00
F20C  SBC $6D,X @ $6D = $00
F20E  LDX $F1FA,Y @ $F1FA = $07
F211  CMP #$00
F213  BMI $F225 = $BD
F215  LDX $F1FB,Y @ $F1FB = $0F
F218  CMP #$01
F21A  BPL $F225 = $BD
```

CPU Status — A: 5 X: 0 Y: 0 PC: F205 Cycle: 1082446916

Flags — P: 5 ☑ Carry ☐ Zero ☑ Interrupt ☐ Unused ☐ Break ☐ Reserved ☐ Overflow ☐ Negative

IRQs: ☐ DMC ☐ NMI ☐ External ☐ Frame Counter

Stack — SP: F2
$E0, $F1, $CB, $F1, $00, $12, $AF, $77, $81, $10

PPU Status — Cycle: 215 Fr#: 32550 Scanline: 75 VRAM Addr: 313C T: 0000 NT Addr: 213C X Scroll: 4
☑ Sprite 0 Hit ☐ Vertical Blank ☐ Sprite Overflow ☐ Write Toggle

Control & Mask — BG Addr: 1000 Spr Addr: 0
☑ BG Enabled ☑ Sprites Enabled ☑ Draw left BG (8px) ☑ Draw left Sprites (8px)
☐ Vertical Write ☐ NMI on vBlank ☐ Large Sprites ☐ Grayscale ☐ Intensify Red ☐ Intensify Green ☐ Intensify Blue

Functions — Function Name

Labels — Label: ApuStatus_4015, Ctrl1_4016, Ctrl2_FrameCtr_4017, DmcAddress_4012, DmcCounter_4011, DmcFreq_4010, DmcLength_4013, NoiseLength_400F, NoisePeriod_400E

Watch — Name | Value

Breakpoints — M | Type | Address | Condition

Function (Entry Addr)	PC Addr	ROM Addr
$F1FD	@ $F205	[$7205]
$F1DE	@ $F1DE	[$71DE]
$F187	@ $F1C9	[$71C9]

0000 Internal RAM | 2000 CPU Registers | 4020 N/A | 6000 Work RAM ($00) RW | 8000 $00 | C000 $01

0000 $00 | 2000 NT0 | 2400 NT1 | 2800 NT0 | 2C00 NT1

PRG analysis: 9.22% (Code: 9.01%, Data: 0.21%, Unknown: 90.78%) | CHR analysis: 21.92% (Drawn: 21.92%, Read: 0.00%, Unknown: 78.08%) 2 cycles elapsed

Cheat codes via memory hacking

You can also override memory values, which allows you to invent cheat codes!

Cheat name	Address (in hex)	Value (in hex)
Permanent fireballs	0756	02
Invincibility	079F	FF
Always 8 lives	075A	07
Swimming	0704	01
Max speed	0057	7F

Questions:

- Note that the value stored at each address is represented via two hexadecimal digits. Given this information, what is the word size of the NES (in bits)?
- Note that the maximum speed is given by the value 7F. Why this value, and not FF?

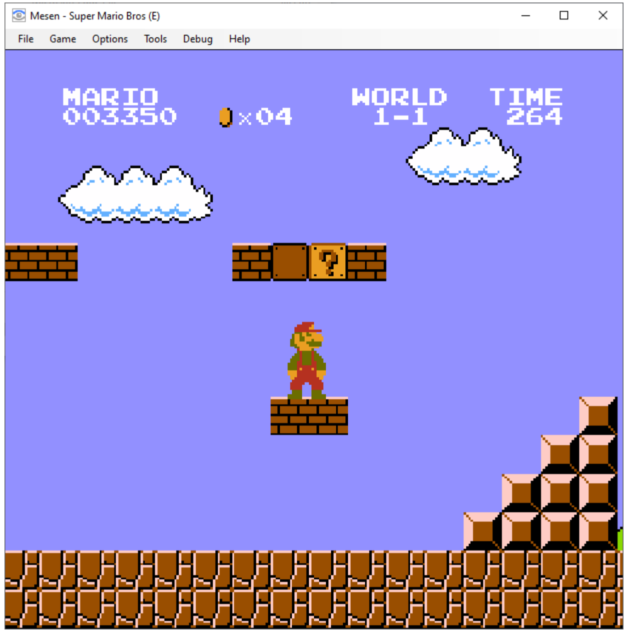
Cheat codes via memory hacking

It’s possible to find cheat codes online, but that’s not actually how we came up with the cheats on the previous slide…

Mesen allows you to scan / query all the current values in memory.

Questions:

- How can we use this to find the memory address where lives are stored?
- How can we use this to find the memory address where Mario’s x-velocity is stored?



Security measures

Obviously, memory hacking is undesirable from a security standpoint!

There are generally protections built into the operating system and the CPU to prevent you from reading and writing other programs’ data.

If your program tries to access a memory location that it isn’t meant to, the hardware will raise a segfault (“segmentation fault”).

Hackers are always coming up with new, clever ideas though…

Division

The integers X and Y are stored at locations x3100 and x3101, respectively.
Write an LC-3 assembler program that calculates X / Y.

Strategy: Keep subtracting Y from X, keep track of how long it takes to reach a negative value.

Part 1 (data validation):

- If X is negative, print out "X must be non-negative".
- If Y is not positive, print out "Y must be positive".
- Otherwise, print out "The inputs are valid".

Part 2:

- Store the quotient (the whole number part of the answer) at **x3102**.

Part 3:

- Store the remainder at **x3103**.

Question

Note that this division program uses quite a few registers!

If we put the code inside a subroutine so that we can call it repeatedly, we'll have to be careful because the subroutine will potentially overwrite register values from our main program.

Can we do anything about this?