The Memory System

February 3, 2025

Memory

- **Executing** code resides in memory
- The processor fetches instructions from memory
- Data used by **executing** code resides in memory
- Machine instructions read data from memory
- Machine instructions write data to memory

The Memory is **always** a target

Other components are initiators

- Processors (always initiator)
- DMA controllers

Before Woke Times

The memory was called passive. Be ware of the mob.

Memory vs. Storage

- What we call ‘memory’ is called by some ‘main memory’
- What we will call ‘storage’ is called by those some ‘secondary memory’
- Surprisingly we prefer our naming
- Technically the difference is this
  - Memory can be accessed with one machine instruction
  - Storage access needs rather large code and knowledge of the specific storage controller used

Memory Speed (lack of, actually)

Era	Proessor	Memory
Olden times	slow	fast
60s	equal	
2024	fast	slow

As of 2024

Processors are about 1000 faster than memory

Processors are starved for data

Programmer/Processor view of the Memory

The Memory as a Vector

- The memory is a 1-dimensional array of bytes
- Each byte is 8 bits wide
- Each byte has an address
- Each byte can be read or written independently
- There is no address to parts of a byte

A memory system as above is called **Byte Addressable**

All memory systems nowadays (2024) are byte addressable

Byte is not Enough

- Suppose we need a (C) `short`
- There are two bytes in a `short`
- Hence two addresses are needed to locate the `short`
- Assuming the bytes are consecutive, one address suffices

Nowadays norm

The minimal address in an item is **its address**

Ambiguity

```
char *p = malloc(2);
char b0 = *p;
char b1 = *(p+1);
```

Little Endian

```
(b1 << 8) + b0
```

Big Endian

```
(b0 << 8) + b1
```

The manufacturer decides the endianness

There are processors in which it is software controlled at the OS level

Endianness

Little Endian

Low address means low value

Big Endian

Low address means high value

```
long x = 0x01020304;
```

x	0x01020304			
	+0	+1	+2	+3
Big Endiann	0x01	0x02	0x03	0x04
Little Endiann	0x04	0x03	0x02	0x01

Endianess with Structure

```
struct {
  long f0;
  long f1;
} s;
s.f0 = 0x01020304;
s.f1 = 0x05060708;
```

	+0	+1	+2	+3	+4	+5	+6	+7
Big endian	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08
Little endian	0x04	0x03	0x02	0x01	0x08	0x07	0x06	0x05

The Drawings are for Us (1)

s.f0	0x01020304
s.f1	0x05060708

	+0	+1	+2	+3	+4	+5	+6	+7
Big endian	0x01	0x02	0x03	0x04	0x05	0x06	0x07	0x08
Little endian	0x04	0x03	0x02	0x01	0x08	0x07	0x06	0x05

	+7	+6	+5	+4	+3	+2	+1	+0
Big endian	0x08	0x07	0x06	0x05	0x04	0x03	0x02	0x01
Little endian	0x05	0x06	0x07	0x08	0x01	0x02	0x03	0x04

The Drawings are for Us (2)

s.f0	0x01020304
s.f1	0x05060708

	Big endian	Little endian
+0	0x01	0x04
+1	0x02	0x03
+2	0x03	0x02
+3	0x04	0x01
+4	0x05	0x08
+5	0x06	0x07
+6	0x07	0x06
+7	0x08	0x05

	Big endian	Little endian
+7	0x08	0x05
+6	0x07	0x06
+5	0x06	0x07
+4	0x05	0x08
+3	0x04	0x01
+2	0x03	0x02
+1	0x02	0x03
+0	0x01	0x04

Endianess Relevance

If there are large binary fields and either:

- Data is transfer between computers
  - No relevance: http is a character based protocol
  - Relevance: ipv4 is binary with `long` fields
  - Relevance: ipv6 is binary with `_BitInt(128) fields`
- Fields and their subfields are used in parallel

Communications

- The IP and TCP protocols are big endian
- x86's are small endian
- IBM mainframes are big endian

Subfields

```
short x = 1;
short *p = &x;
char  *q = (char *)p;
printf("%d %d", *q, *(q + 1));
```

Little endiann output:

```
1 0
```

Big endiann output:

```
0 1
```

Communicating with The Memory System

Reading/Writing in Principle

Transaction

- A read or write operation towards the memory
- (A special case of bus transaction)

Read transaction

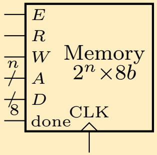
- Supply address
- Assert read command
- Get the data when ready

Write transaction

- Supply address and data
- Assert write command

Of course, the issues are in the details

$2^n \times 8b$ Read Transaction



Initiator

- Assert E, R and A
- Remove R and A. When done is asserted take the D lines and deassert E

Memory

- Wait for E and R to be asserted
- Take the A lines
- Get from the chips the byte
- Assert done and set D

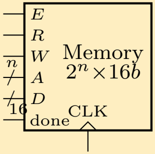
Reading multibyte

- An initiator might need `short` or longer from memory
- In this case the initiator will initiate transaction as needed
- Of course, this take time

Improving speed

- Wider D bus
- Burst mode

Wider D-bus



Twice the memory with the same A lines as before

On read always two bytes will be returned

Adding wires has its price

Not all accesses will be faster!

The Serial Revolution

- Serial communication has become amazingly fast
- Thus it becomes practical to use single wire link to the memory
- Then we work kind of like IP protocol with layers
- More on this in the bus chapter

Ideal Memory/Storage

- Non volatile
- Dense
- No power usage when not in use
- Cheap

We do not know (yet?!) how to manufacture this ideal system

Solution

We use different systems were parts of the above traits hold

Just Remember

This is a bug, not a feature...

Memory vs. Storage

Memory

One machine instruction fetches/stores from/into memory

Storage

Whole programs are needed to access storage

Memory vs. Registers

- Registers are memory units inside of the processor
- Very fast
- Cannot have too many of them: Context switch

The Memory Controller

- On the one side talks with the memory/system bus
  - Usually proprietery
- On the other side talks with the memory chips
  - Becomes standartized

The controller translates external transactions into internal ones

Memory System Organization

- Inside the memory system different configurations are possible
- There is a connection between the outside world expectation and the internal configuration
- This connection is not 1-1 as will be made clear later

Memory chips

- Mostly blacbox (for us) analog devices
- Size: Number of addresses $\times$ bits per address
- From very simple to highly complex

Read Only Memory

Read Only Memory (ROM)

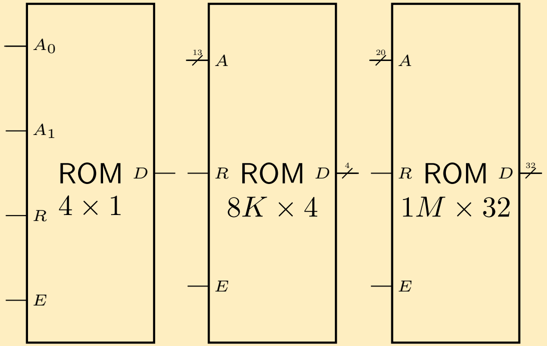
- Bad name
- The important property is being non-volatile
- The read onlyness is ‘a bug not a feature’

Evolution

- ROM
- PROM
- EPROM
- EEPROM

Computer systems always(!) have ROM

Asynchronous ROM chips block diagrams



General Picture

- High impedance while E is cleared
- Connect and output valus when E is asserted

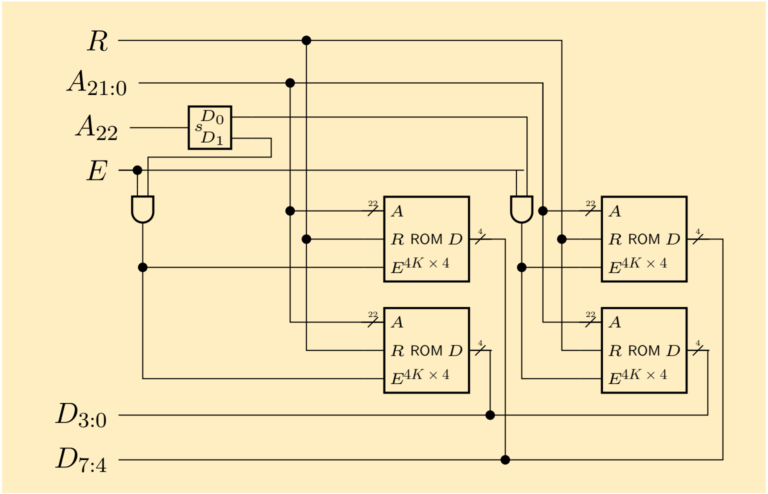
ROM Async Chip Read Protocol

- Set the address on the A lines
- Assert R (if there is one)
- Wait
- Set the E line
- Wait
- Copy the O lines
- Clear the E line
- Wait

Waiting time given by the chip producer

Chips Configuration

4M $\times$ 8 from 2M $\times$4 Chips



Random Access Memory

RAM

- Unfortunate name
- ROM is also random accessbile
- RWM would be a better name, but. . .
- Two families: Static RAM and Dynamic RAM

SRAM	DRAM
‘Forever’	Short term
Faster	Slower
Not so dense	Denser
Expensive	Less expensive
Simpler	Not so simple

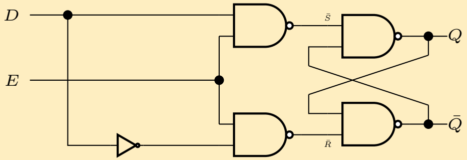
Nowadays (2024)

- SRAM: Mostly in caches
- DRAM: Main memory

SRAM

SRAM cell is logically a D-Latch

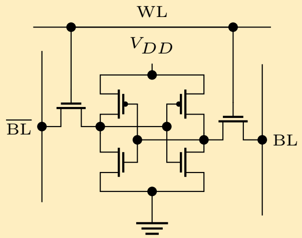
(recall) D-Latch



Lots of transistors!

SRAM 6T Cell

From our EE collegues



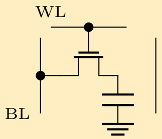
Read	Write
Assert WL	Set the BLs
Wait	Assert WL
Read BLs	Wait
Clear WL	Clear WL

- Very popular as of 2024
- In essence this is a D-latch
- It is analog, so we do not need to understand it
- Cells invented due to system needs: 4T,5T,6T,7T,...

Dynamic RAM

(Dennard 1966)

DRAM 1T1C Cell



- Value of the cell lost if not accessed for 64ms