A First Look at Microprocessors

using the

The General Prototype Computer (GPC) model

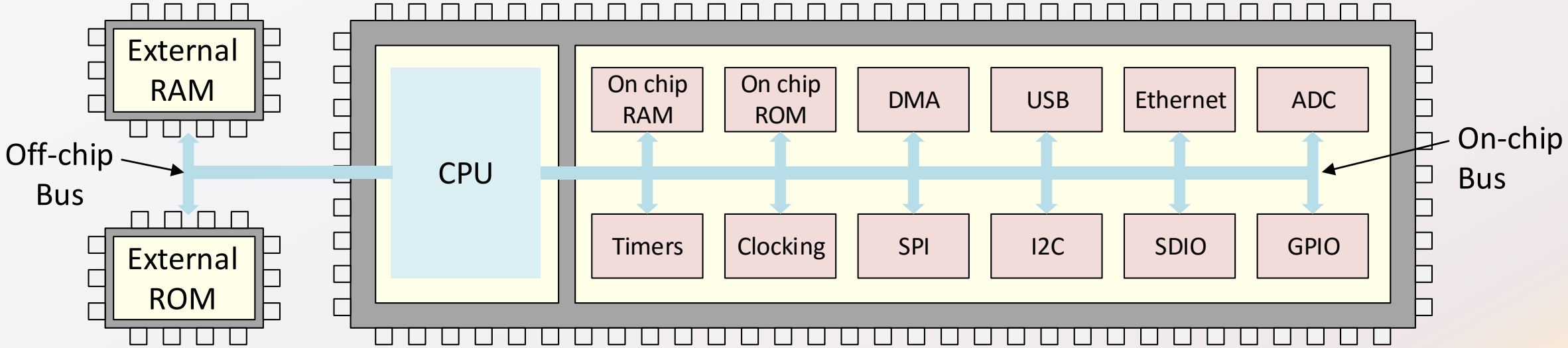
Part 3

CPU Ecosystem

CPUs by themselves cannot make a complete system – they need certain other “peripherals”, or support circuits, to create a functional system.

Typical peripherals include RAM, ROM, clocking, various ports, and other circuits.

Peripherals are connected to the CPU by a bus. A bus is a fixed-width set of transport signals that move data in and out of the CPU. A bus is often shared by more than one data producer and/or data consumer, and CPUs often have more than one bus.



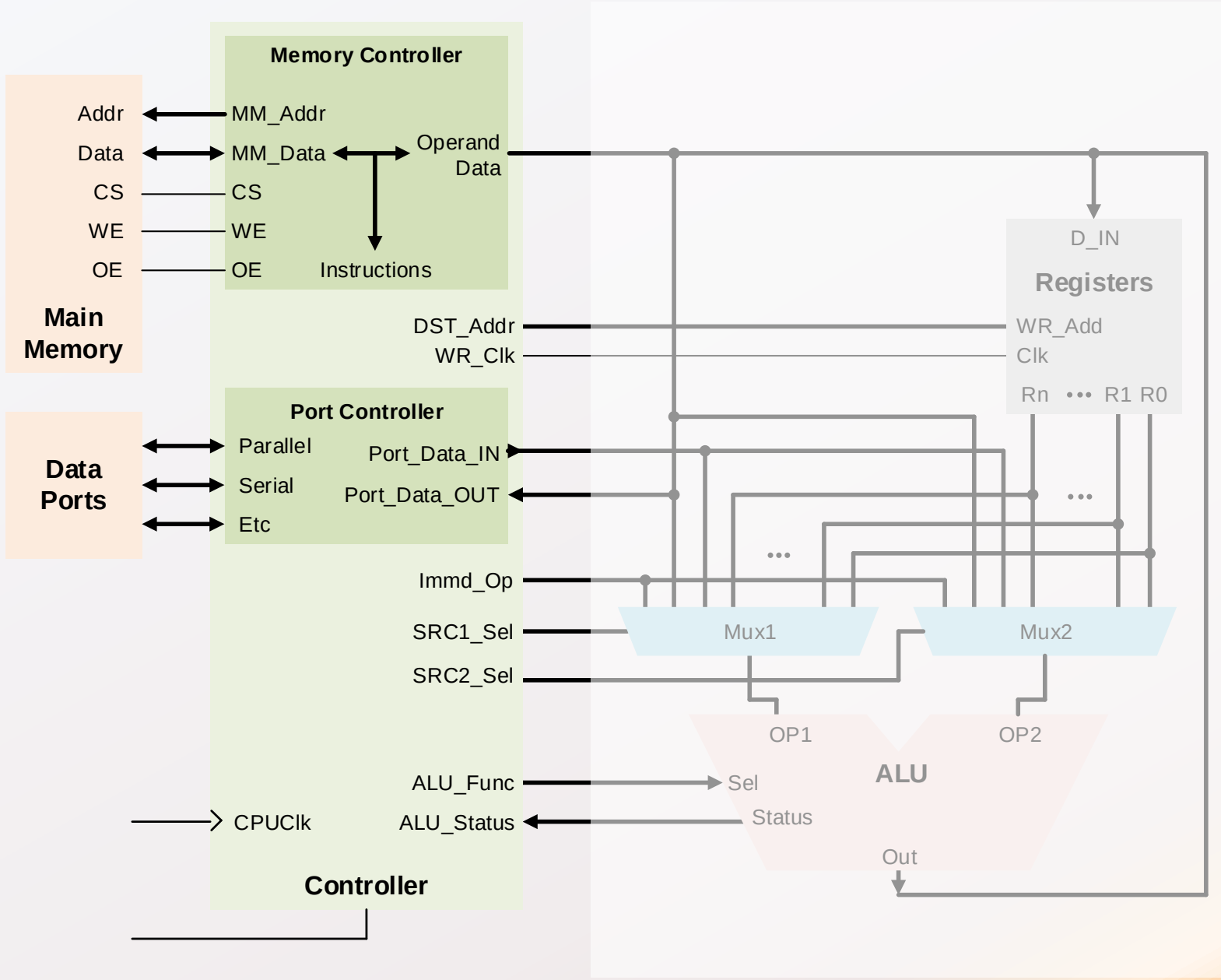
External Busses

Processors need external busses to communicate with external devices, like memories and ports.

Memories hold programming instructions and data. They are too large to fit on chip (except for smaller applications)

Ports bring new data into the processor, or move data out of the processor while a program is running.

Access to any external bus typically requires several clock cycles.

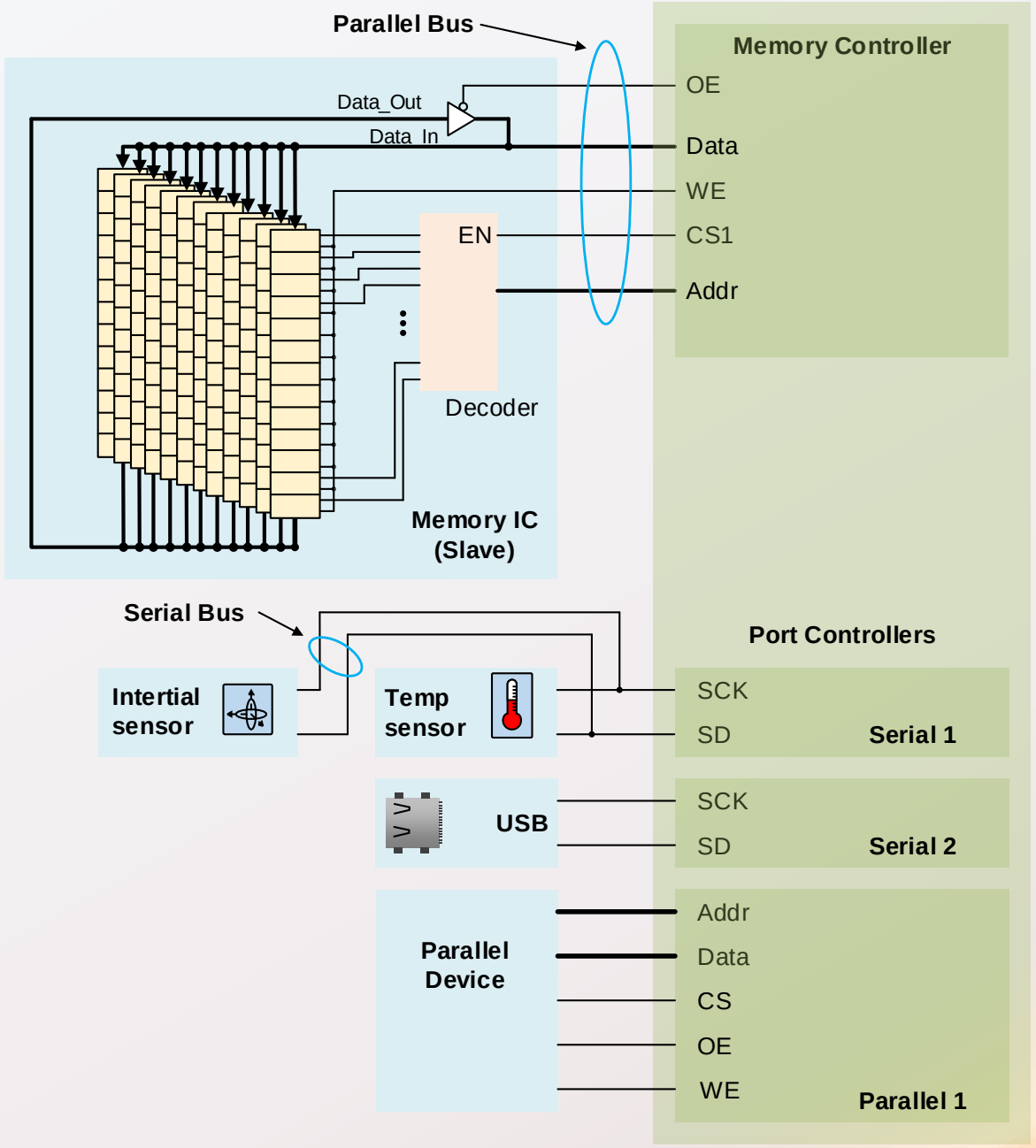


External Busses

An external bus needs several signals, including address, data, direction control (in/out), and timing. A parallel bus has separate wires for all signals; a serial bus uses time-multiplexing to transport all information on one wire (and sometimes a second “clock” signal for timing).

OE: Output Enable; turns on tristate buffers
WE: Latch signal for memory cells
CS1: Chip select; one chip select needed for each device that shares the bus (Why?)

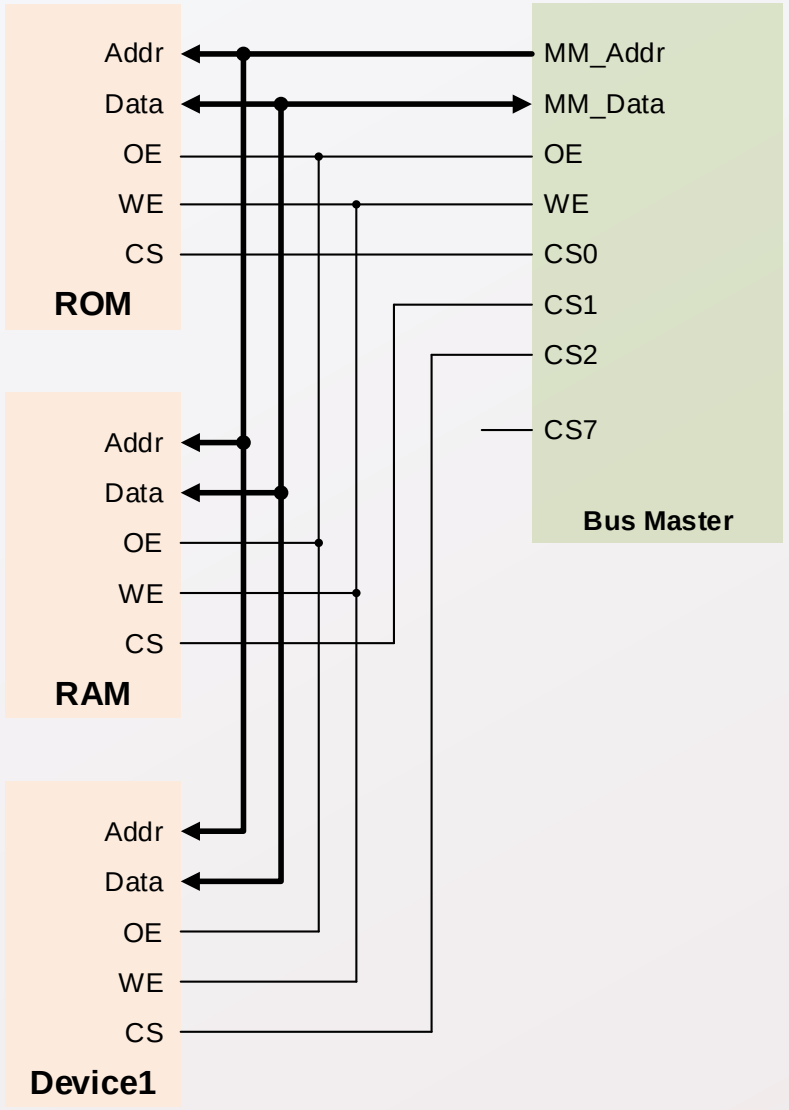
SCK: Serial Clock
SD: Serial Data

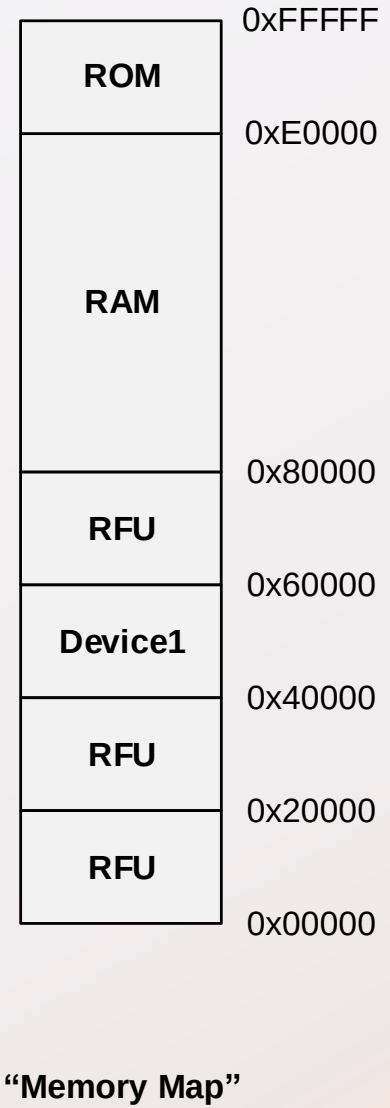


Parallel Busses

We have already encountered one parallel bus - main memory. In an actual computer system, several types of memory devices may share that bus, including RAM, ROM, and certain other “memory mapped” devices (Note: Memory mapped devices are so called because they share the same bus as memory uses.)

Every device on the shared bus uses the same address, data, and timing signals. But they each have a separate Chip Select (CS) so that only one device is enabled at a time.





“Memory Map”

Memory Map

A memory map shows the entirety of the addressable memory space, and the range of addresses that are associated with a given function.

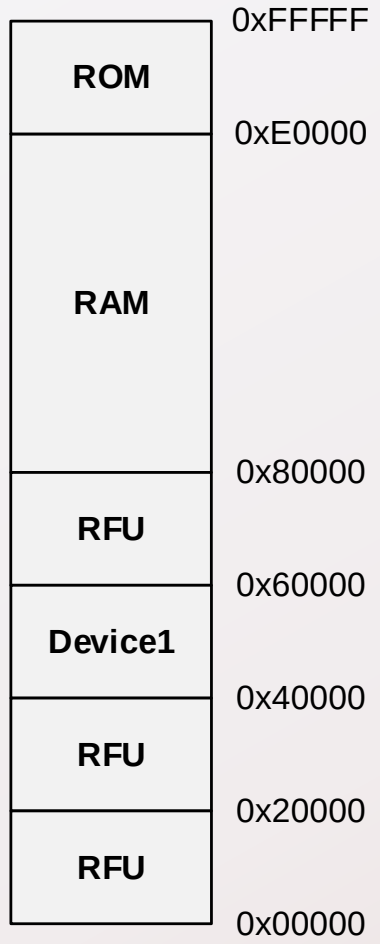
How large is this memory space (in bytes?)

Chip select pins are driven by address decoders. They can typically be programmed to cover a given memory range.

How large is each memory section (in bytes?)

How many address bits would the CS decoder decode?

If a system has more external memory mapped devices than there are chip selects to enable them, an external decoder can be used to create more chip selects for given address ranges. How?



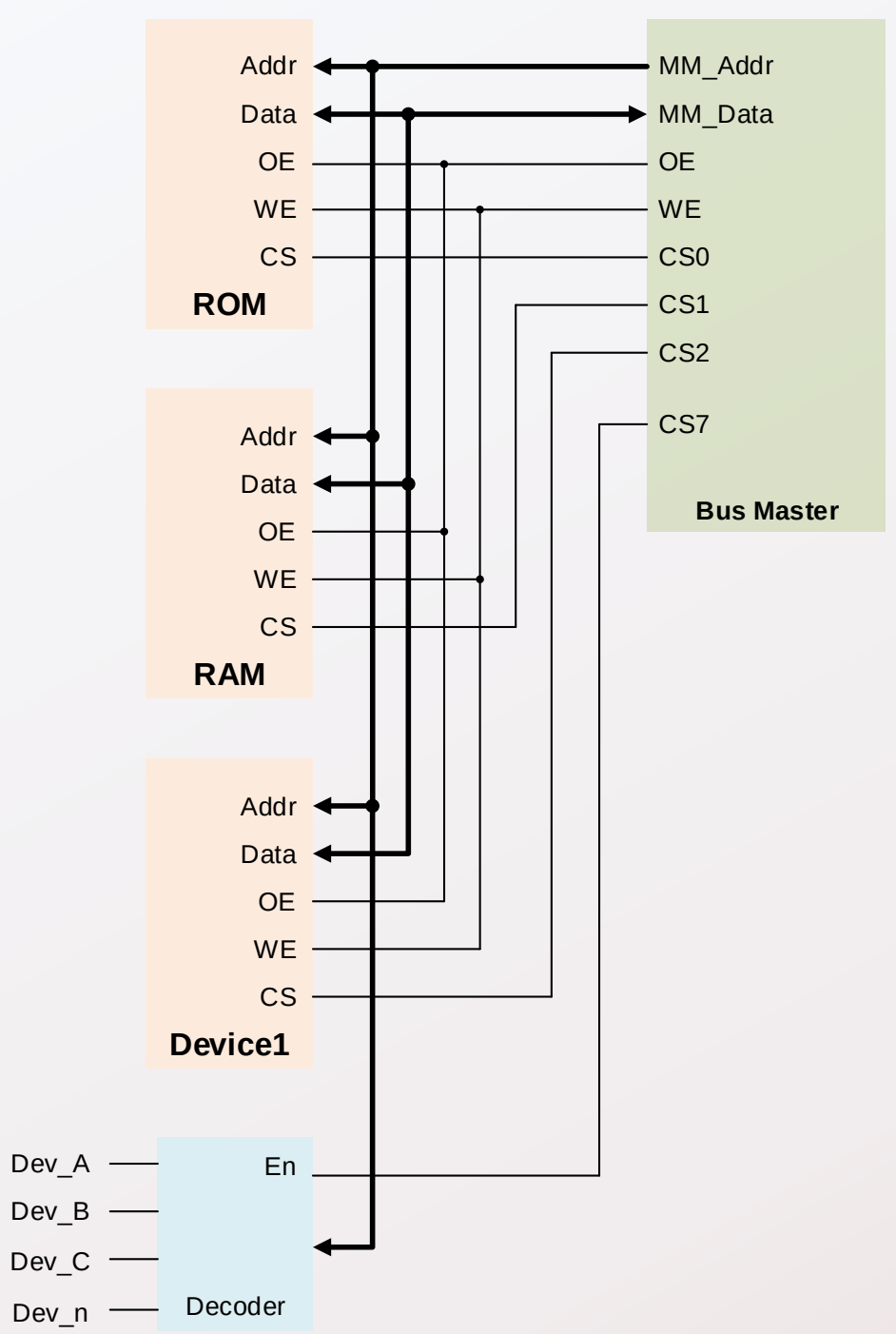
“Memory Map”

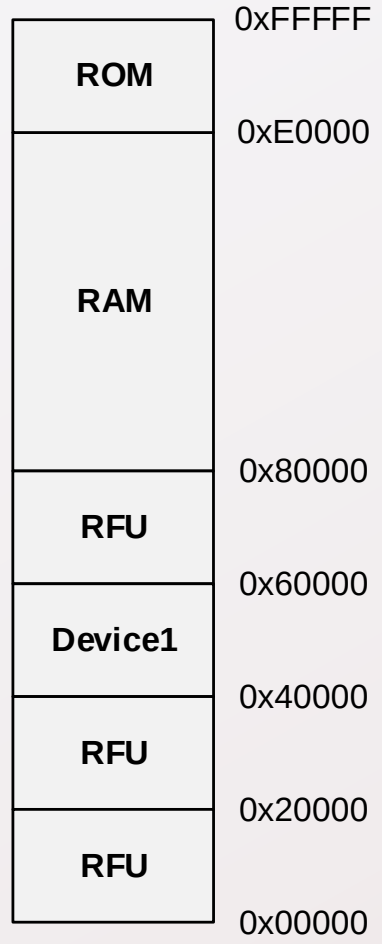
If a system has more external memory mapped devices than there are chip selects to enable them, an external decoder can be used to create more chip selects for given address ranges.

In the example shown, a decoder is used to further decode the memory map from 0x00000 to 0x20000. Which address bits are input to the decoder?

Which address bits are input to the decoder?

How many bytes are in each subrange of CS7?

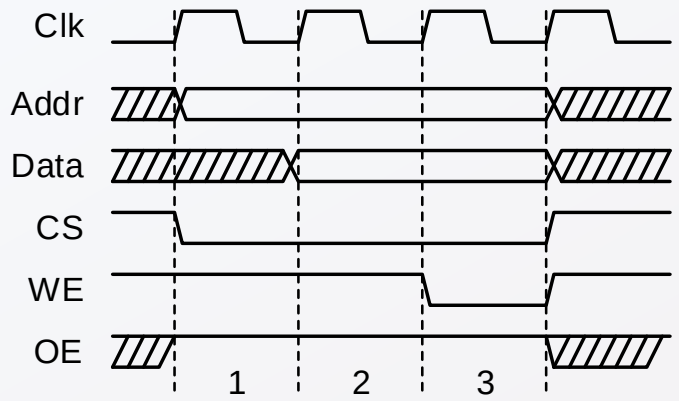




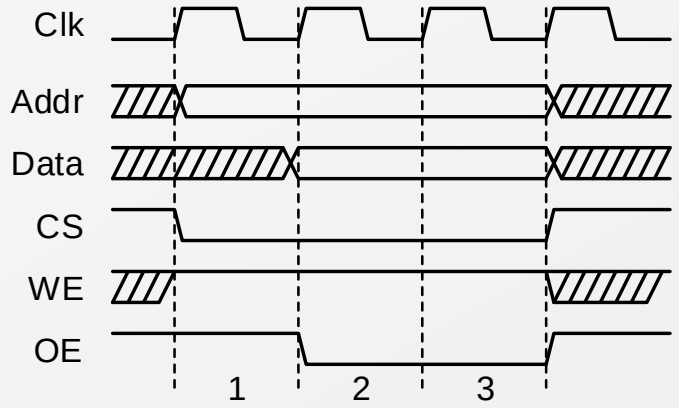
“Memory Map”

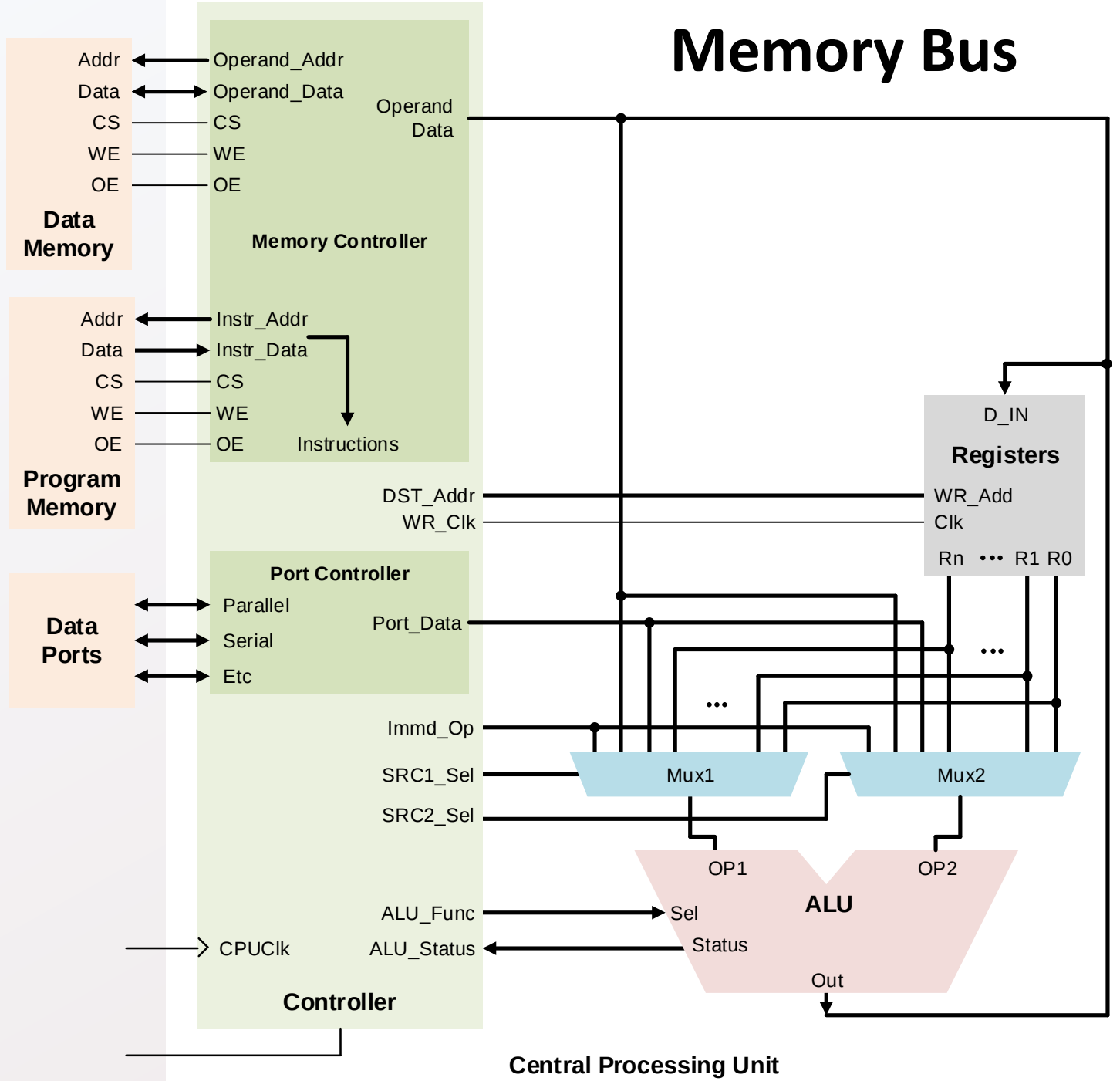
Registers can be read immediately, and written in one clock cycle. But memory requires a sequence of signals, and multiple clocks.

Memory Write Cycle

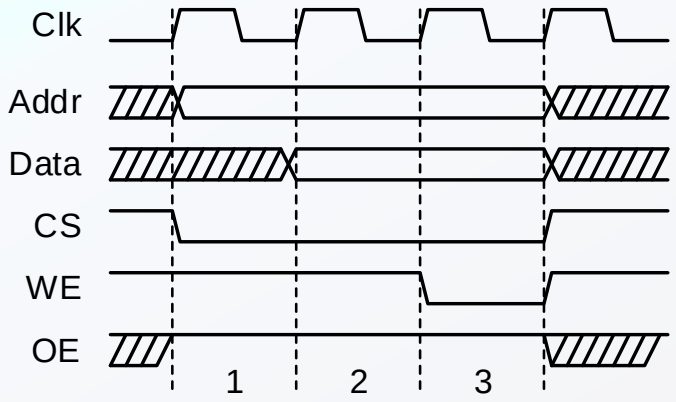


Memory Read Cycle

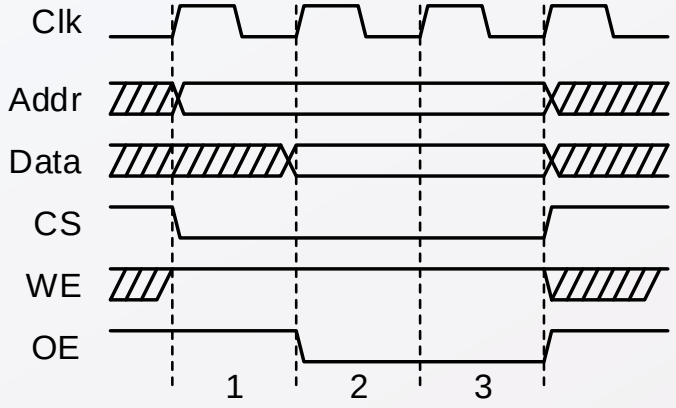




Memory Write Cycle



Memory Read Cycle



In both read and write cycles, CS and address are driven first, to “turn on” the chip that will be accessed, and establish its address. Typically, the address signals require some amount of “settling time” before a read or write can occur.

For a write cycle, the processor drives data onto the bi-directional data bus after CS and address have settled. Then WE is asserted (low), and data is latched into the memory on the rising edge of WE.

For a read cycle, OE is asserted (low) to turn on the memory side drivers, and so data flows from the memory towards the processor. After the data settles, it can be latched into the processor (on either the end of cycle 2 or cycle 3).

It may be possible to speed these operations up a little, but accessing main memory is always slower than accessing internal registers.

External Memories and Busses

System designers have many types of memory to choose from, including SRAM, Fast SRAM, Dynamic RAM (DDR), NAND Flash, NOR Flash, and many others.

The external memory bus in our example in intended to control a Static Random Access Memory, or SRAM. SRAM was the original memory used in processor systems. It is fast, but large arrays are expensive.

Many newer, higher-density, and faster memories (like Dynamic RAM) require more and different signals than are shown in our simple interface. These extra signals typically deal with more advanced timing signals, and more flexible data access options. The main concepts are the same – just the access implementation details change.

Some external busses are “multiplexed”, and share address and data lines (why)? These busses can be identified by their “ALE” signal.

When accessing memory, the address settling time is typically the longest and most critical. Once the address is stable, data can be read or written quickly.

So, most processors read several consecutive locations at once every time main memory is accessed, even though the data at the additional locations has not yet been requested. Most of the address pins will be the same so there will be less settling time. The "speculative" data is stored in temporary memory locations that are very like registers (they are fast, with single-cycle access). If the processer needs data stored in one of these temporary registers, it can get it immediately.

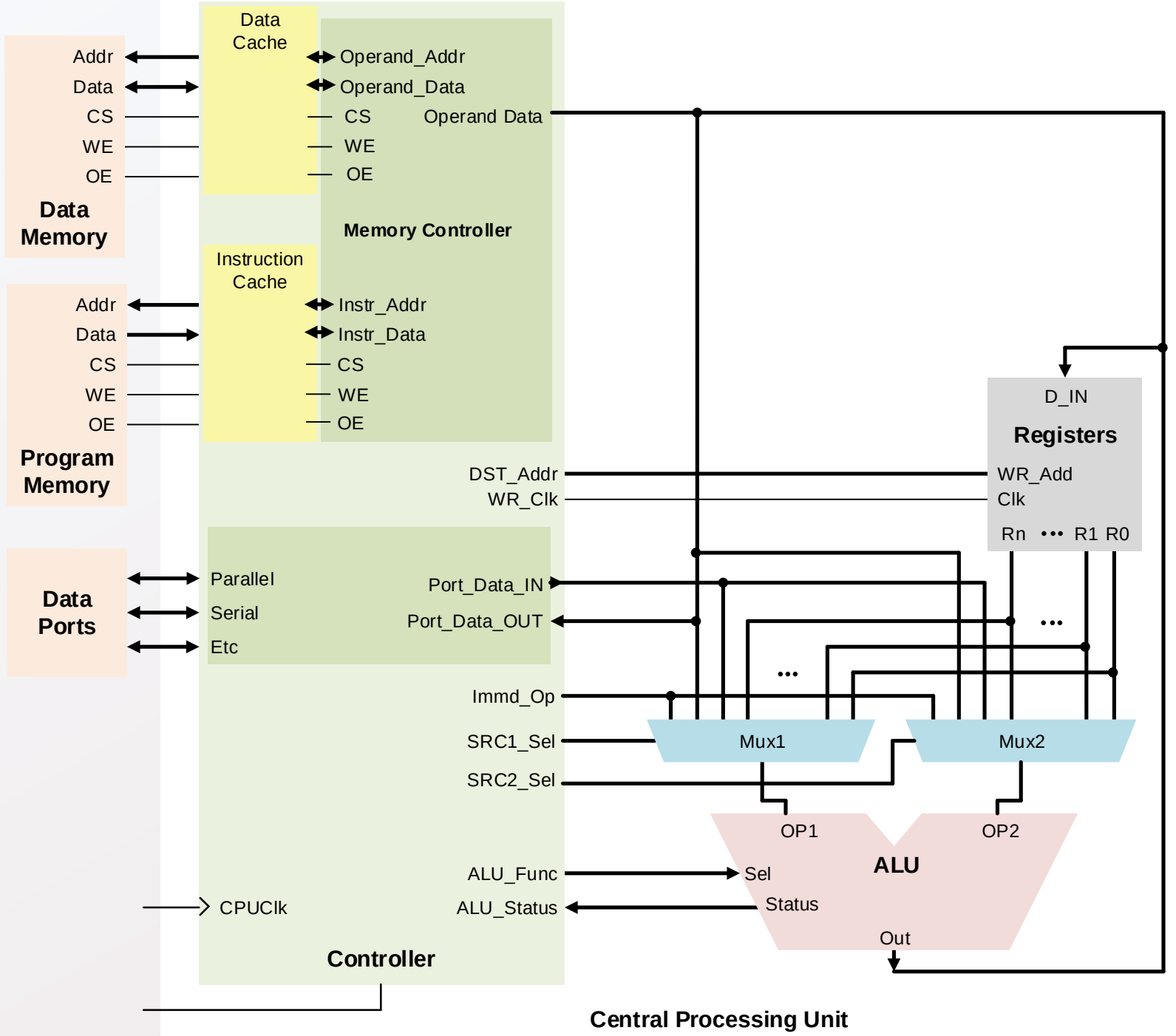
This is known as ***cache memory***, and it is included in most processors today.

A memory manager system manages the cache, keeps it current, and determines when external memory cycles are required.

Cache memories are on the processor IC. They are small, high-speed memories that hold current and likely-to-be-used data and instructions.

Every time the memory controller accesses external memory, it grabs a few extra bytes that are likely to be needed in the near future.

If the cached data is needed, it is available much more quickly than it would otherwise have been.

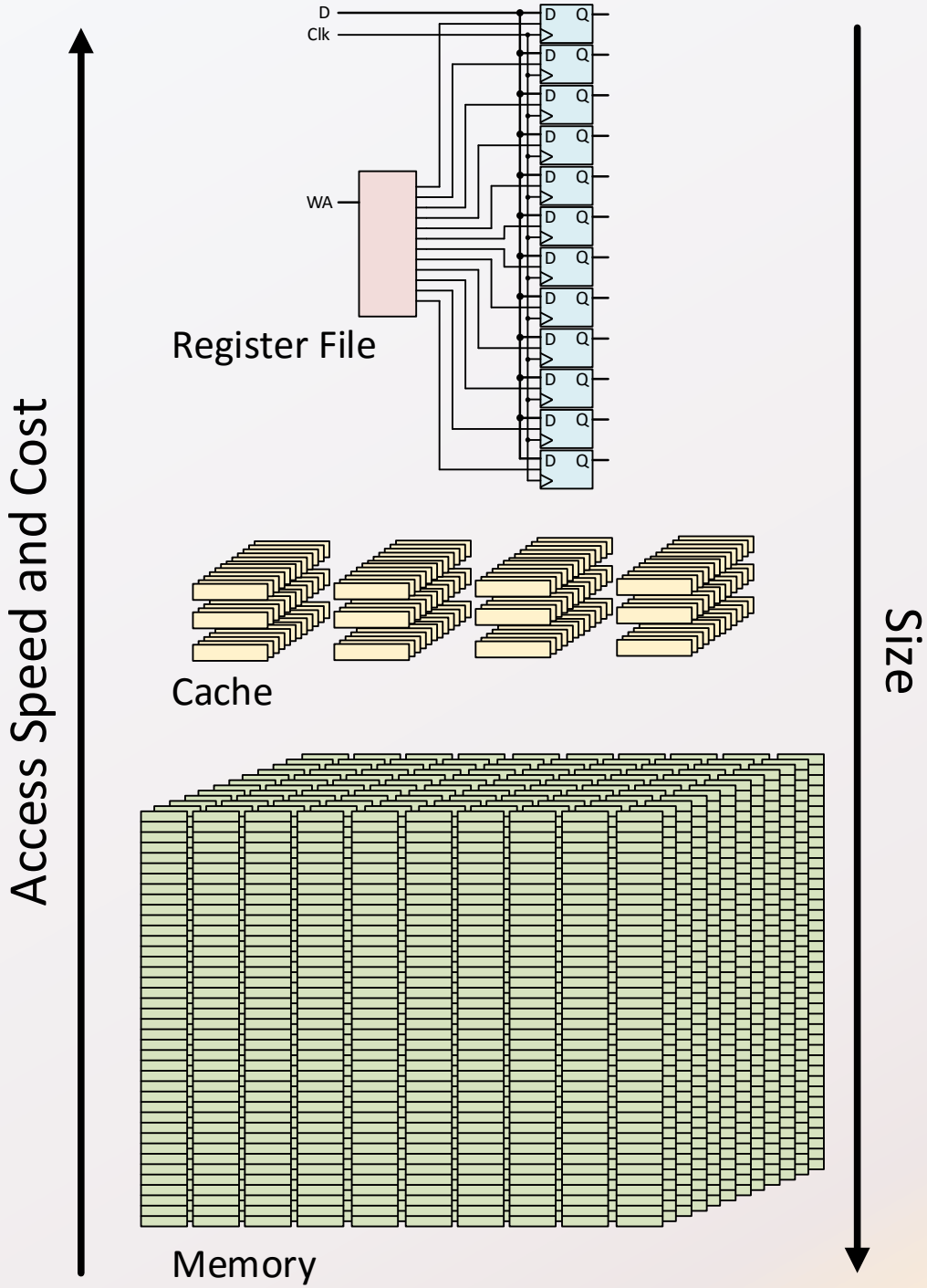


Because cache memory is much smaller than main memory, and because it is co-located on the same IC as the processor, it is much faster than main memory.

It is also more expensive, because it uses more area per bit than main memory (and, it is on the processor die).

Cache is accessed through a shared bus, and so cache requires more overhead than registers.

Registers are the fastest.



Cache

If a processor uses cache memory, every access to main memory can potentially come from cache (and so, take less time). Sometimes, the requested memory data will be in cache – this is called a cache “hit”. Sometimes, the requested memory data will not be in cache – this is a cache “miss”.

When a processor must access main memory, it reads a “Cache line” that includes a tag (address less the line address bits), the data, and flag(s). Cache lines are typically 8-128 bytes, and every time a miss occurs, an entire line is read (not just the needed word).

There are many options when designing a cache system, including “associativity”, “write policy”, and virtual memory issues (homonyms and synonyms)

Other Busses

In addition to the parallel memory bus, most processors have one or more “I/O busses”, or ports. These ports connect external devices like sensors, actuators, ADCs, or communications ICs to the processor. The majority of I/O ports use serial busses (SPI, I2C, UART, USB, etc. – we will look at these later).

Many processors also maintain external parallel busses in addition to the memory bus (memory busses have become more specialized and higher-speed, making it difficult to share them).

These “legacy” busses are used to drive displays (like smaller LCDs or OLEDs), or older memories, or other less demanding peripherals. They typically use a bus similar to the older, simpler memory bus shown in our illustrations.

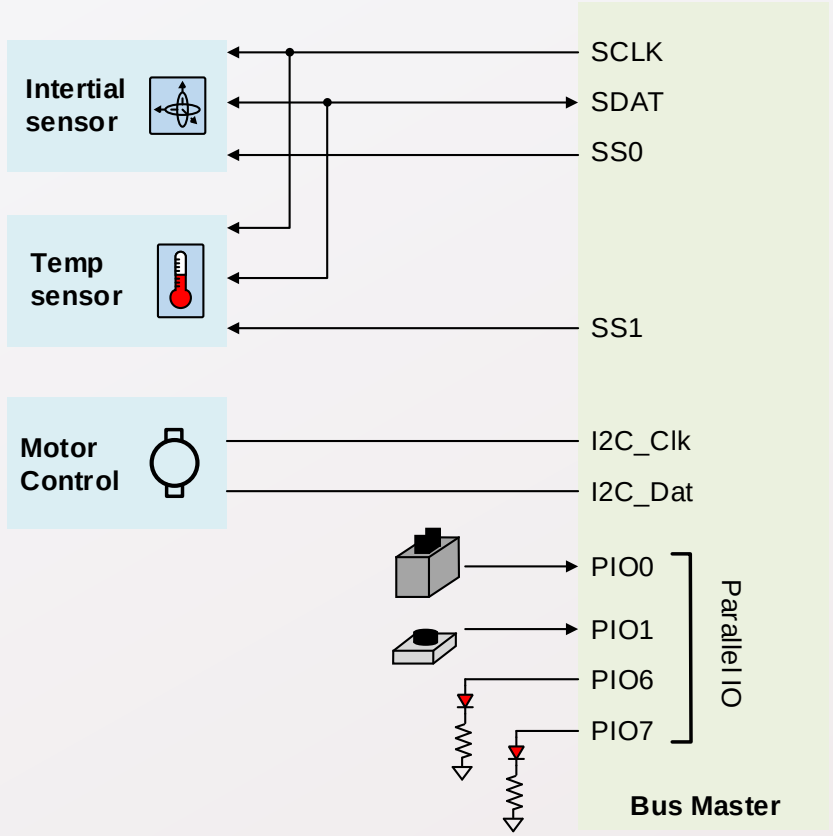
Serial Busses

In addition to the parallel memory bus, most processors have one or more “I/O busses”, or ports. These ports connect external devices like sensors, actuators, ADCs, or communications ICs to the processor. The majority of I/O ports use serial busses (SPI, I2C, UART, USB, etc. – we will look at these later).

Modern serial busses can move as much data as parallel busses of just a few years ago.

Some I/O busses are “one to one” with only two participants, a master and a slave. The master initiates all transactions, either writing data to the slave, or reading from the slave. The RS232/UART bus, commonly called a COM port (or just serial port) is an example.

Some I/O busses are “one to many”, and there are several protocols. The I2C (or TWI) bus is very common for low-speed peripherals. It is a multi-master, multi-slave, packet switched network. The SPI bus is common for low to medium speed peripherals, and it uses a star configuration.



External/off-chip busses are relatively basic and simple, because external pins are expensive.

Processors also use internal busses to move information between on-chip IP blocks. Internal busses are often more complex, with many more options and timing models. Later, we will look at the internal bus structure used by ARM: the Advanced Microcontroller Bus Architecture, (AMBA), and the more recent Advanced eXtensible Interface, or AXI.