A First Look at Microprocessors

using the

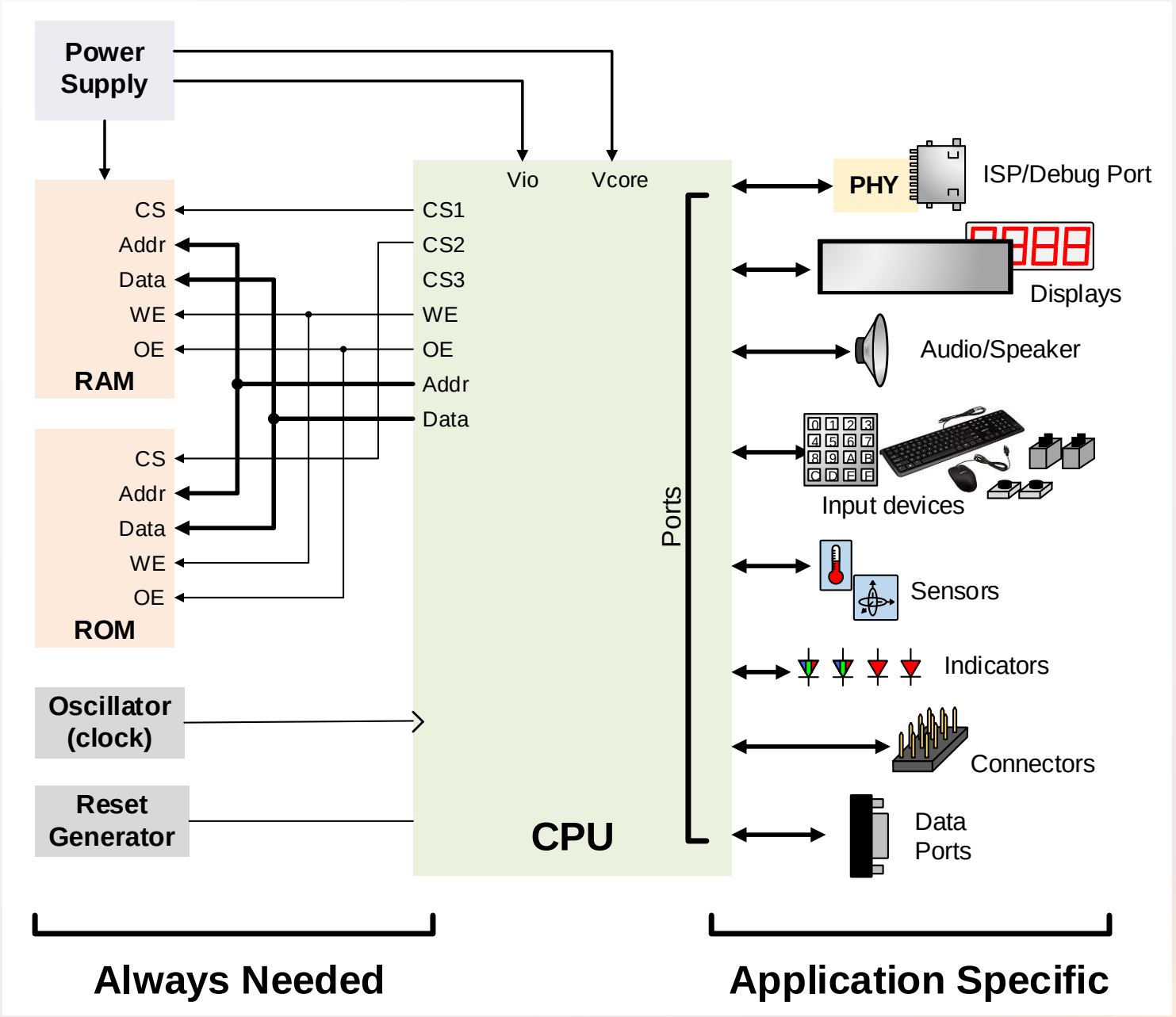
The General Prototype Computer (GPC) model

Part 4

CPU Ecosystem

All CPUs need RAM, ROM, a clock source and reset circuit, and power.

Depending on the application, CPUs will most typically have other connected devices as well. These other devices will use one of the serial busses, or a parallel bus, depending on the device itself.



As transistors have become less expensive and smaller, many peripherals that were once separate IC's have migrated to the processor IC. Examples include audio and video processors, bus controllers, timers, media interfaces, etc.

Many peripherals remain in separate ICs. Memory circuits are as large (or larger) than the processor – there is simply no room for them. Same for high-end graphics processors. Some circuits require specialized manufacturing that is not compatible with the processes used to make processors – for example, physical front-end chips (called PHYs) that drive cables like Ethernet or USB. Same for radios like Wifi and Bluetooth.

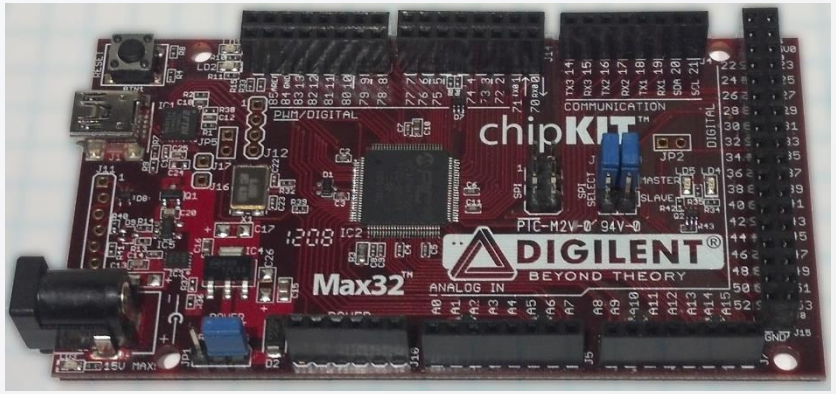
So, most processor systems will have a central microprocessor surrounded by a collection of support IC's that are application dependent.

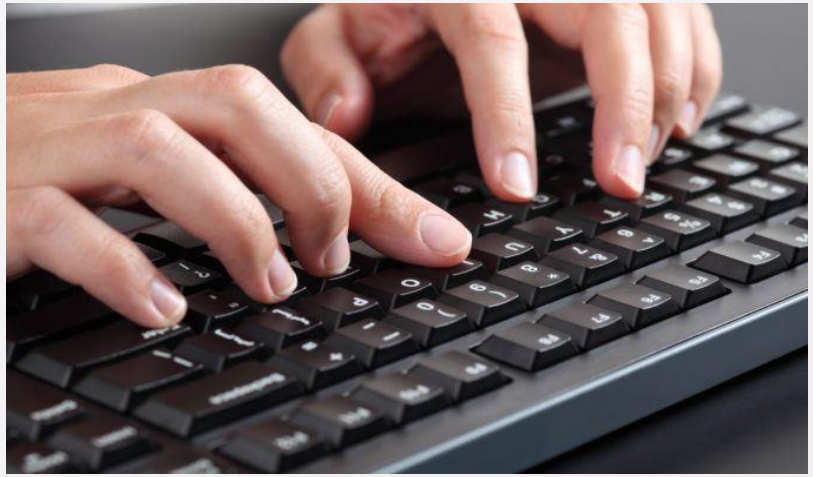
A desktop computer will have (at least) a microprocessor, external memory, a clock source, a radio, and an Ethernet driver. Expansion slots allow other peripherals to be added later.

An embedded microprocessor in an appliance might have a collection of on-board sensors and motor drivers particular to the application.

Processors typically use clocks that are in the 10MHz to 200MHz range. Many applications require a processor to interface with much slower signals from I/O devices, like human-actuated buttons or switches.

Consider a CPU with a 40MHz clock (25ns period); every 25ns, an instruction can be executed.

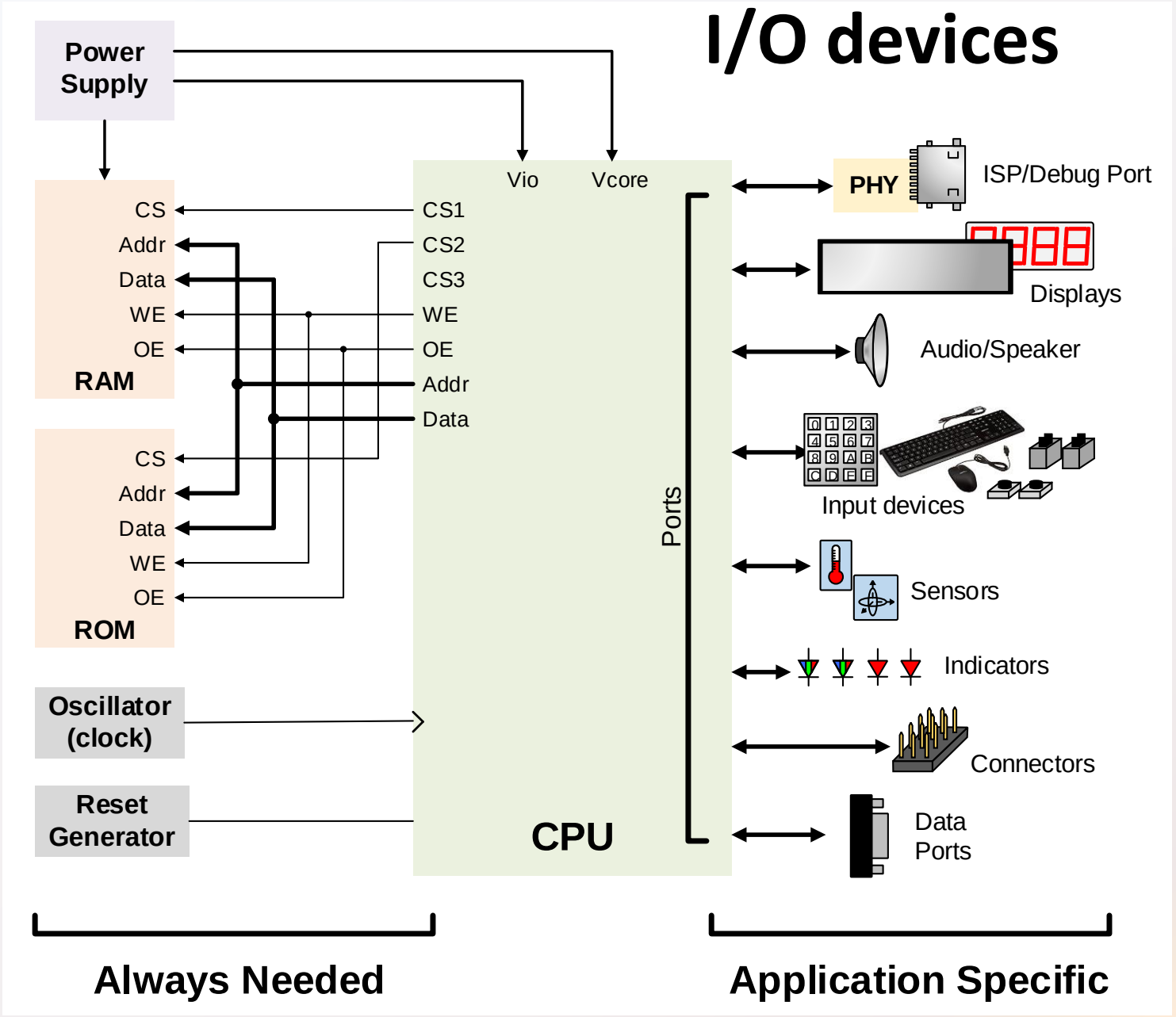




A fast typist can enter about four key strokes per second – one every 250ms.

This $6 processor can execute 10 million instructions between keystrokes!

Instead of waiting around, for slow peripherals to produce or consume data, the CPU does other functions. Then when a port needs service, it asserts an “interrupt request” signal.



Interrupts

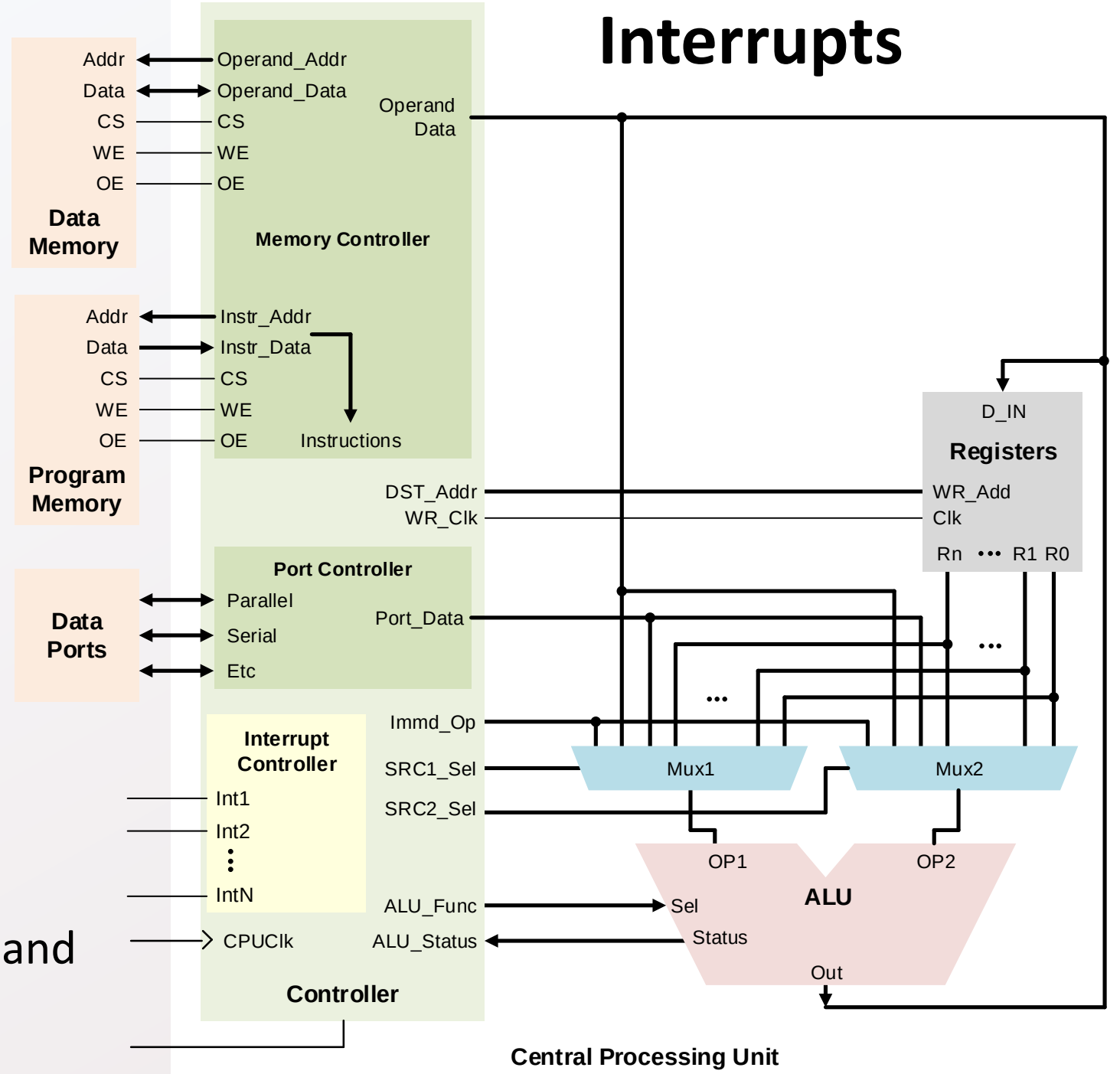
Interrupts are “exceptional event” signals that interrupt the normal flow of CPU operations.

Interrupts typically arise from peripheral devices that need service, or else data may be lost.

Examples include a keyboard, a disc, various ports, and a power fail.

The CPU may or may not respond to the interrupt request, depending on the interrupt priority.

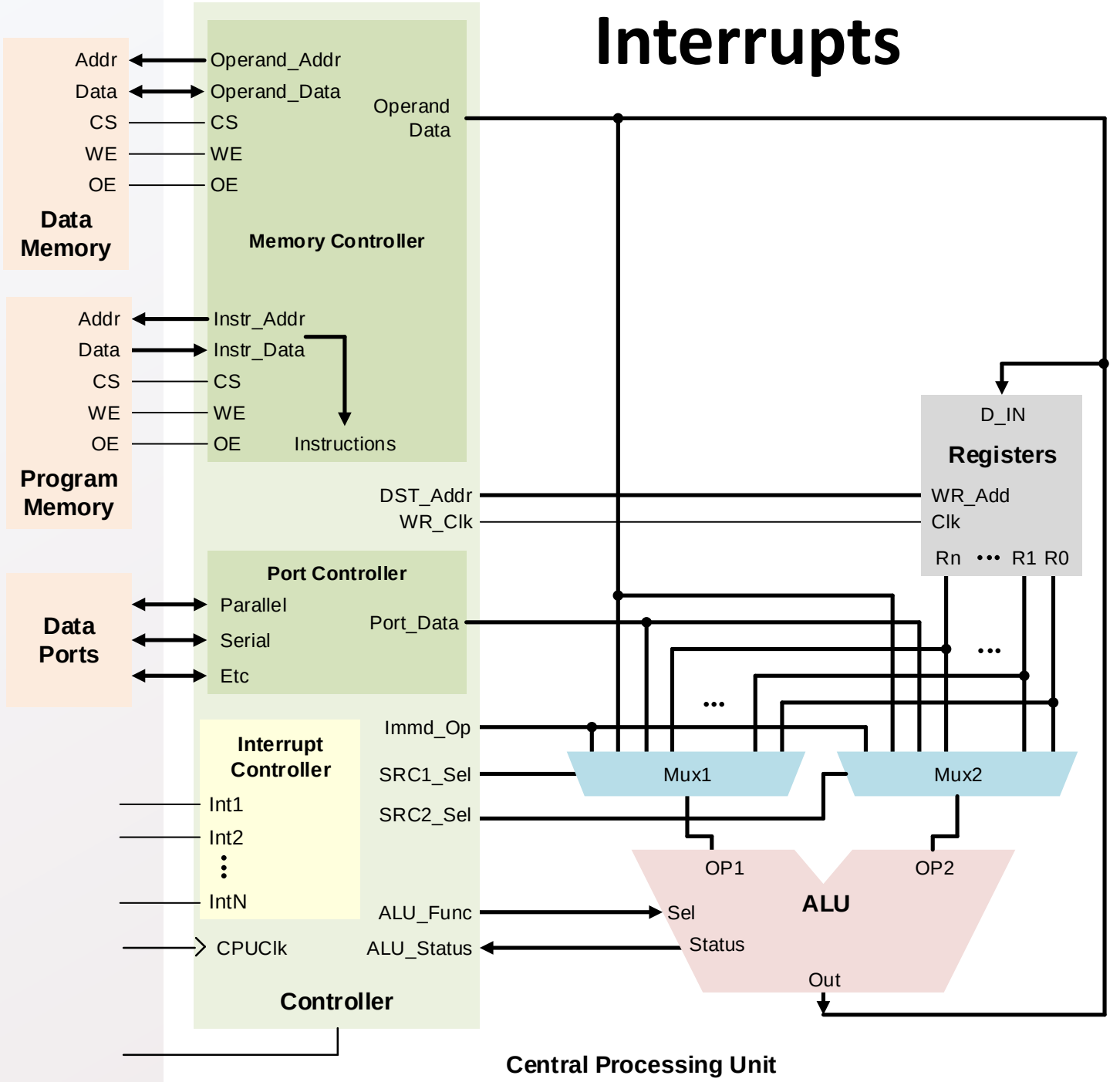
If the interrupt is taken, the CPU immediately stops its regular execution and runs the interrupt service routine (ISR).



Interrupts

When the ISR is complete, regular execution begins exactly where it left off. Since the main program was unexpectedly interrupted, the ISR must enable the main program to continue operating as if nothing happened.

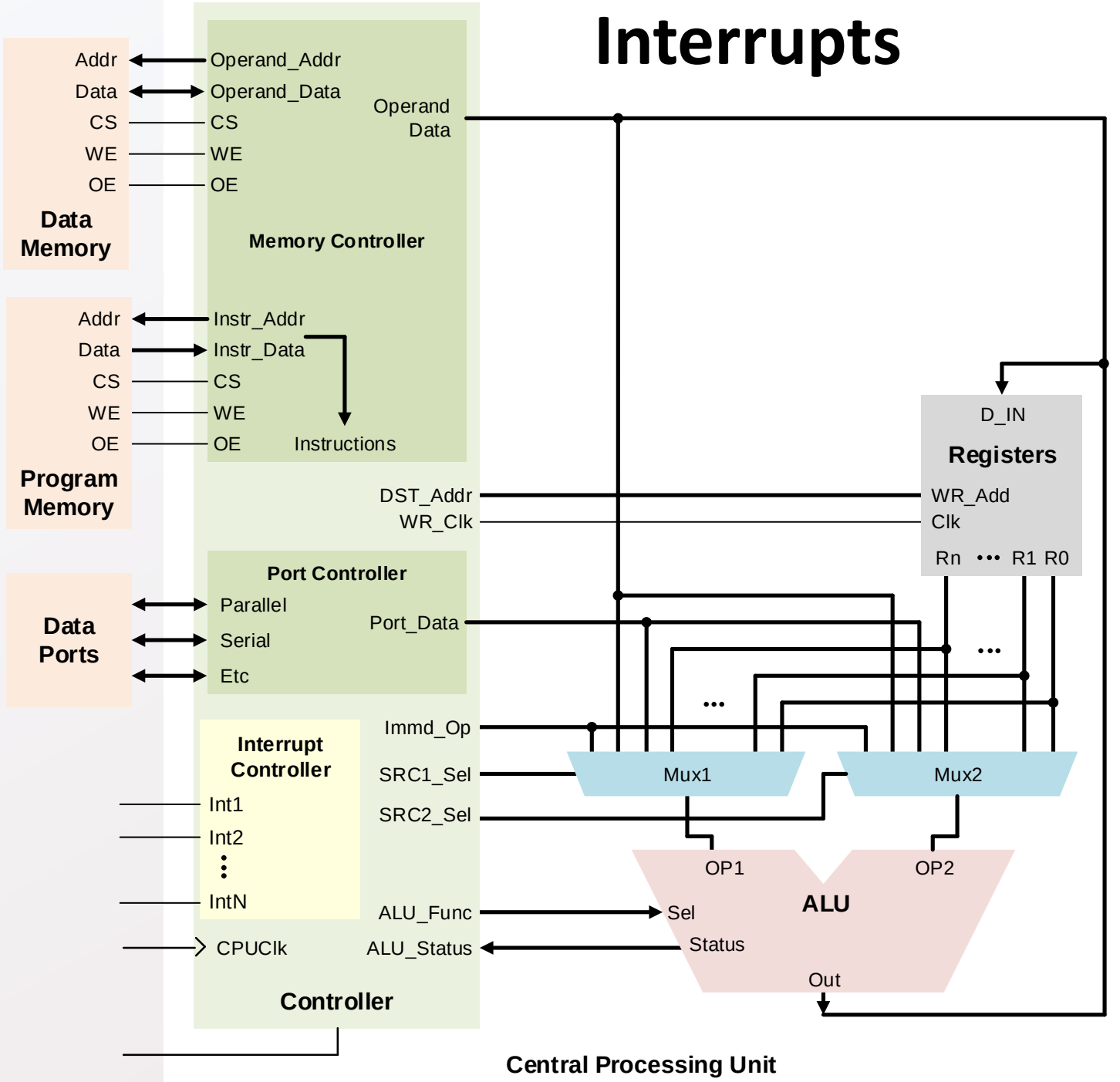
The ISR must save all registers, and optionally adjust priorities, before doing anything else. This essential “context switch” preserves the operating state of the processor, so it can resume execution right where it left off, as if the interrupt never occurred.



Interrupts

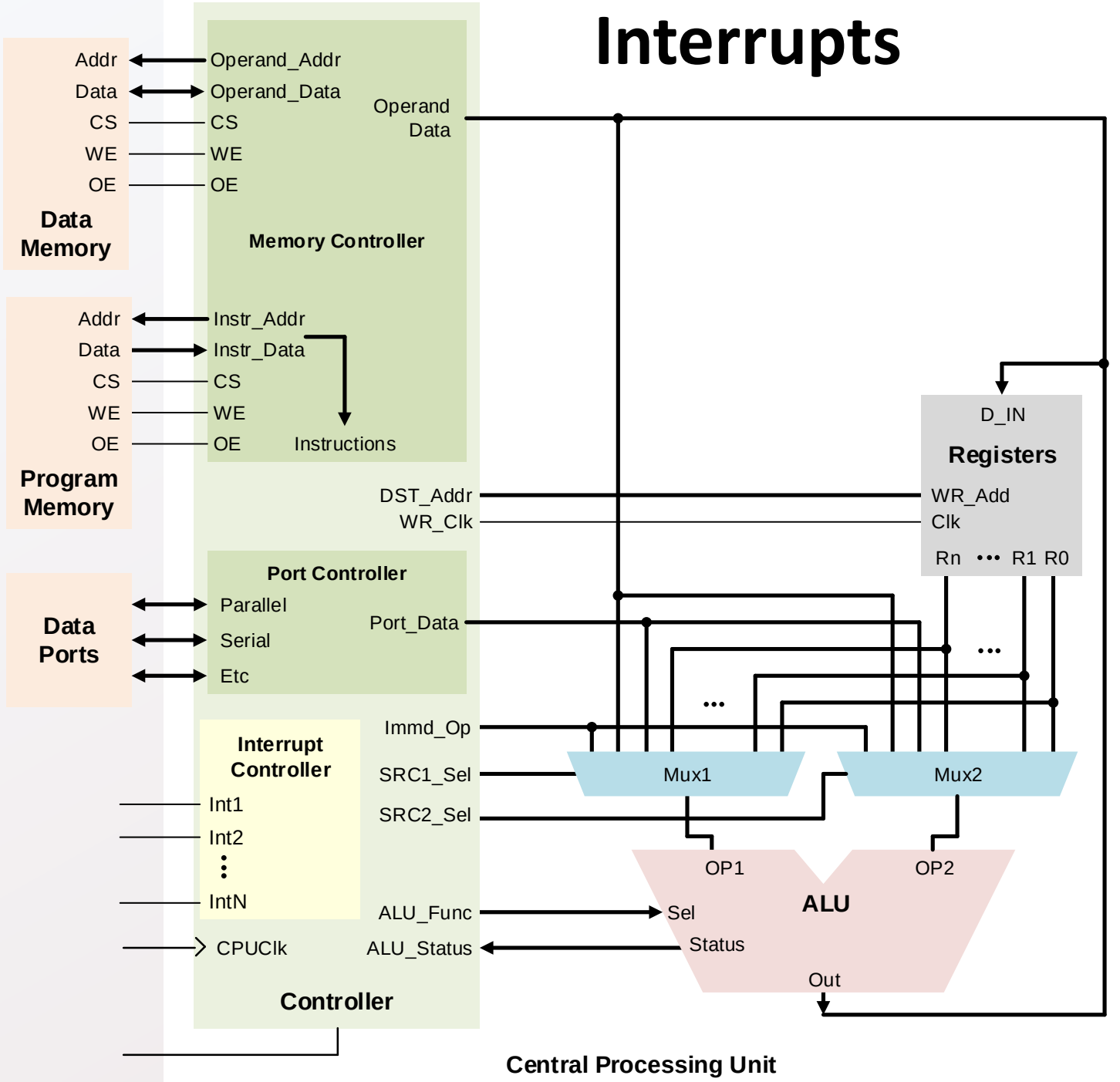
CPUs have “interrupt controllers” that receive exceptional event signals, and redirect the CPU as required.

Different interrupts are assigned a priority level by the programmer. Events that require quicker service are typically assigned higher priority. The CPU can set the interrupt level that it will respond to.



Interrupts

Interrupts are a much more efficient way to deal with infrequent events, rather than “polling” (i.e., constantly checking signal status).

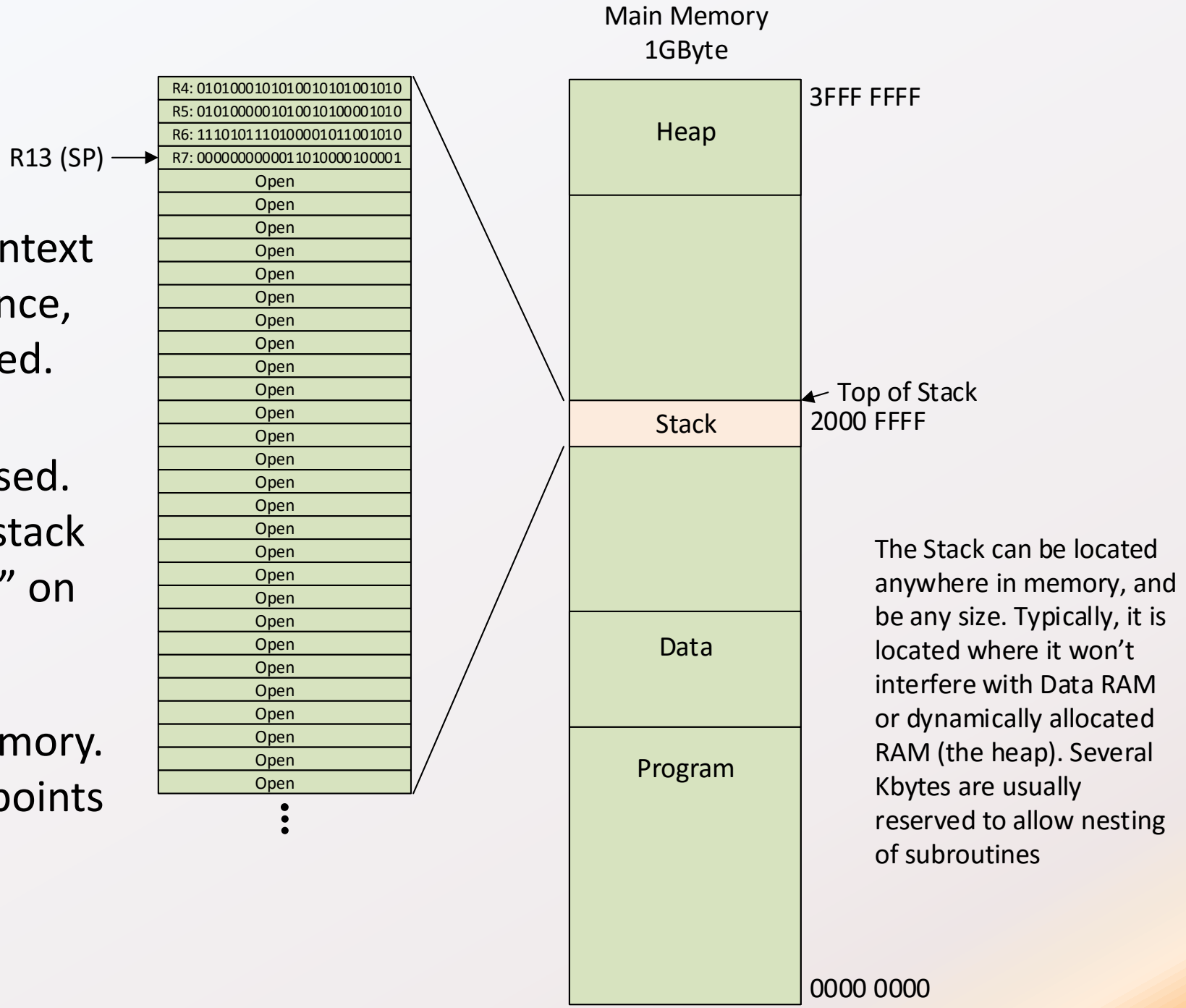


Stack

Saving and restoring context (“context switching”) is a common occurrence, so a standard method has emerged.

A FIFO “stack” data structure is used. Registers are “pushed” onto the stack on entry to the ISR, and “popped” on exit.

The stack is simply an area of memory. A stack pointer (register) always points at the most recent entry.



Events, Exceptions, and Interrupts

Events are expected signal transitions generated by peripherals that happen during the normal course of operations. For example, a peripheral device may have data ready, or an internal timer might have expired.

Exceptions are anomalous or exceptional conditions that occur during normal operations, but that require special and immediate attention. Examples include dividing by 0 (there is no suitable outcome), writing to “protected” memory, or overflowing certain memory structures.

Interrupts are signals that may result from events or exceptions. Typically, the sources of events or exceptions can be configured to generate interrupts if desired. Interrupts can be assigned a priority level, and the CPU can be configured to respond to interrupts of a given priority level.

Low-level programs

“Machine code” is a listing of the opcodes. For example, a simple program for our processor might be:

1101000100001101

1101001000000010

0000001100010010

0010010100110100

1110110000001101

This is not very user friendly. Mnemonics are.

A “Mnemonic” is a memory device that is easy to remember. It is associated with something that is not easy to remember, as an aid in learning and recall.

Assembly mnemonics

"Assembly code" is more or less a listing of the opcodes, but the binary numbers have been replaced with more readable "mnemonics". Mnemonics are one-to-one replacements for machine codes that are easier to use and remember. A typical instruction format is:

```
Instruction    Destination, Source1, Source2 (Source 2 may be a "Val")
```

Machine Listing	Assembly		Comments
1101000100001101	XFER	R1, data1	// load data1 into R1
1101001000000010	XFER	R2, data2	// load data 2 into R2
0000001100010010	ADD	R3, R1, R2	// add R1 and R2 into R3
0010010100110100	SUB	R5, R3, R4	// compare R3 to R4
1110110000001101	BZ	next	// If equal, branch to label "next"

Assembler

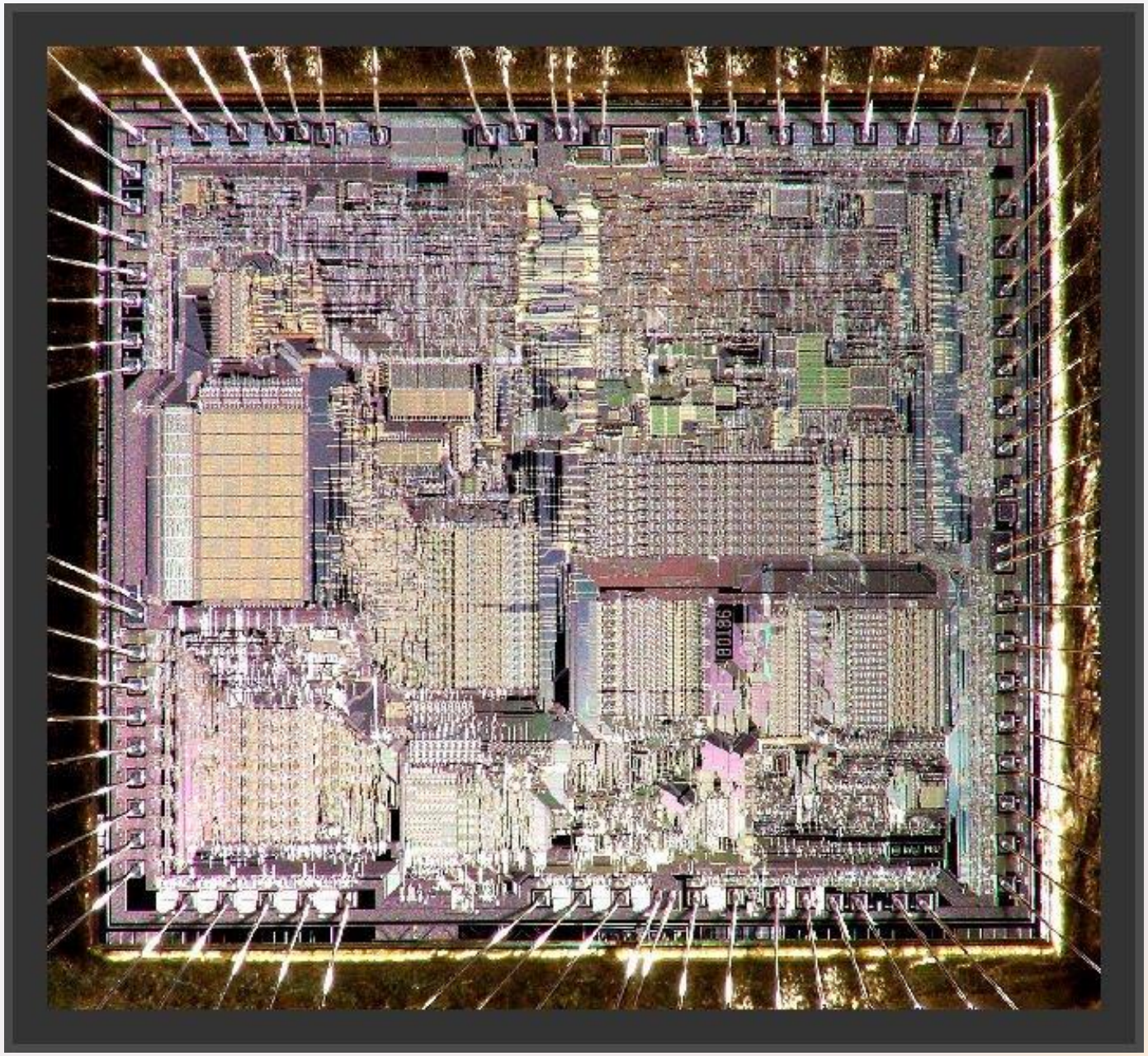
The Assembler program swaps out the assembler mnemonics for machine code. The machine code can then be forwarded to the linker, and then programmed into memory for execution.

The Assembler also offers mechanisms to define memory structures at a given location, to use text labels for reference points (as in the BZ example), to include external files, etc. We will learn about the ARM assembler throughout the class.

Microcontrollers

Transistors are relatively cheap; pins are relatively expensive.

Off-chip functions must minimize signal pins



Intel 80186 Circa 1983

Microcontrollers

Microcontrollers are low-to-midrange performance microprocessors, with lots of on-board peripherals.

Intended for “one chip” solutions for white goods, toys, and consumer products, and specialty markets like automotive, telecom, industrial controls, etc.

Typically emphasize low power, low cost, low system complexity

Typically don’t emphasize high performance