MIPS: the I/O interface

These transparencies are based on those provided with the book:
David Money Harris and Sarah L. Harris, “Digital Design and Computer Architecture”,
2nd Edition, 2012, Elsevier

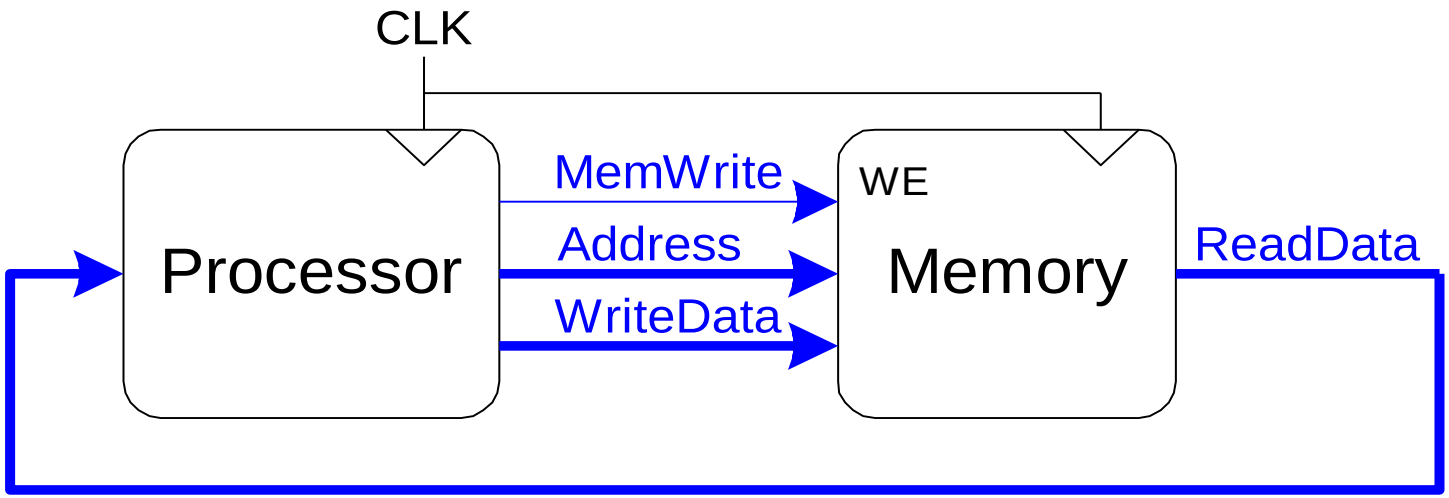
Memory-Mapped I/O

- Processor accesses I/O devices (like keyboards, monitors, printers) just like memory
- Each I/O device is assigned one or more address
- When that address is detected, data is read/written from/to an I/O device instead of memory
- A portion of the address space is dedicated to I/O devices (*memory-mapped I/O*)

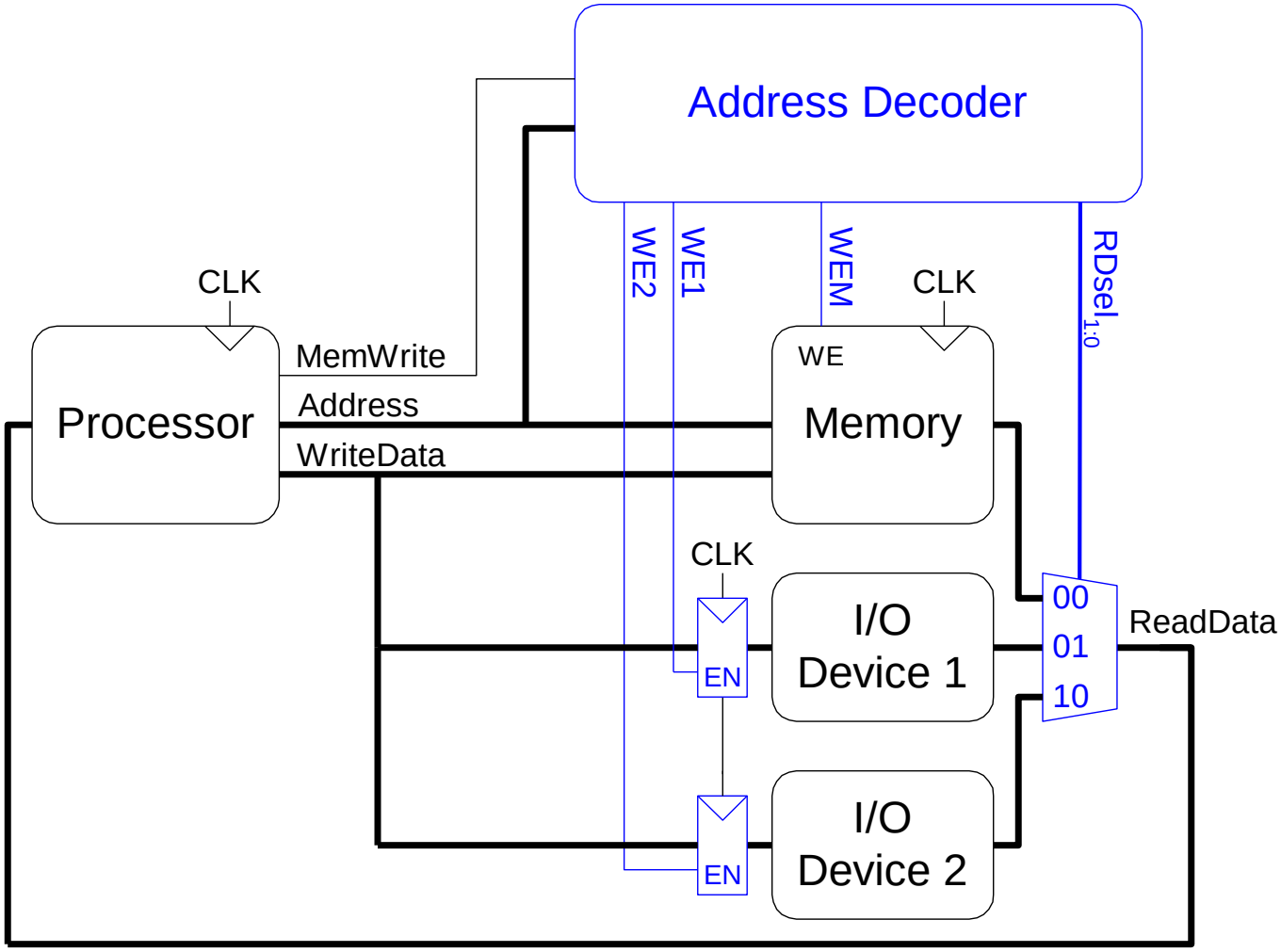
Memory-Mapped I/O Hardware

- **Address Decoder:**
  - Looks at address to determine which device/memory communicates with the processor
- **I/O Registers:**
  - Hold values written to the I/O devices
- **ReadData Multiplexer:**
  - Selects between memory and I/O devices as source of data sent to the processor

The Memory Interface



Memory-Mapped I/O Hardware



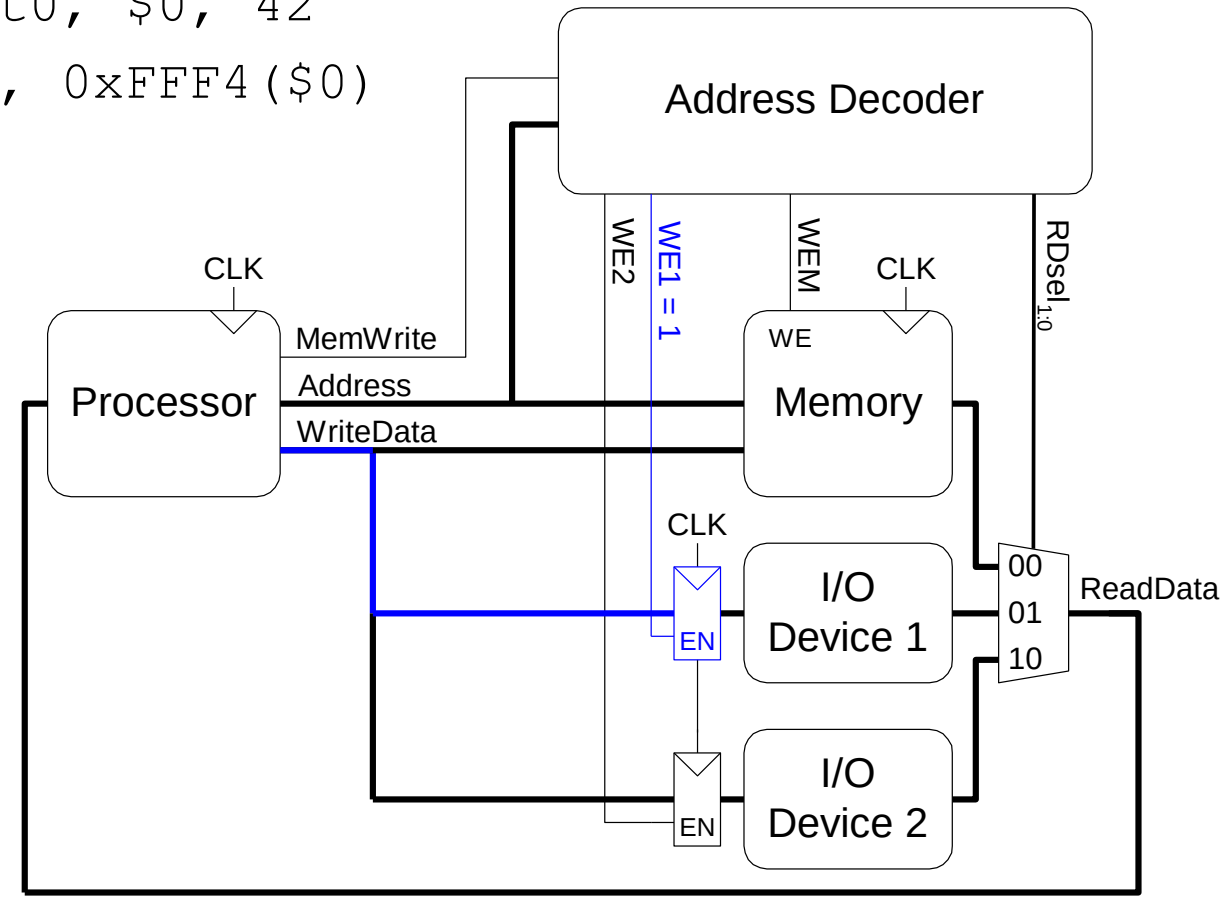
Memory-Mapped I/O Code

- Suppose I/O Device 1 is assigned the address 0xFFFFFFF4
  - **Write the value 42 to I/O Device 1**
  - **Read value from I/O Device 1 and place in `$t3`**

Memory-Mapped I/O Code

- **Write the value 42 to I/O Device 1 (0xFFFFFFF4)**

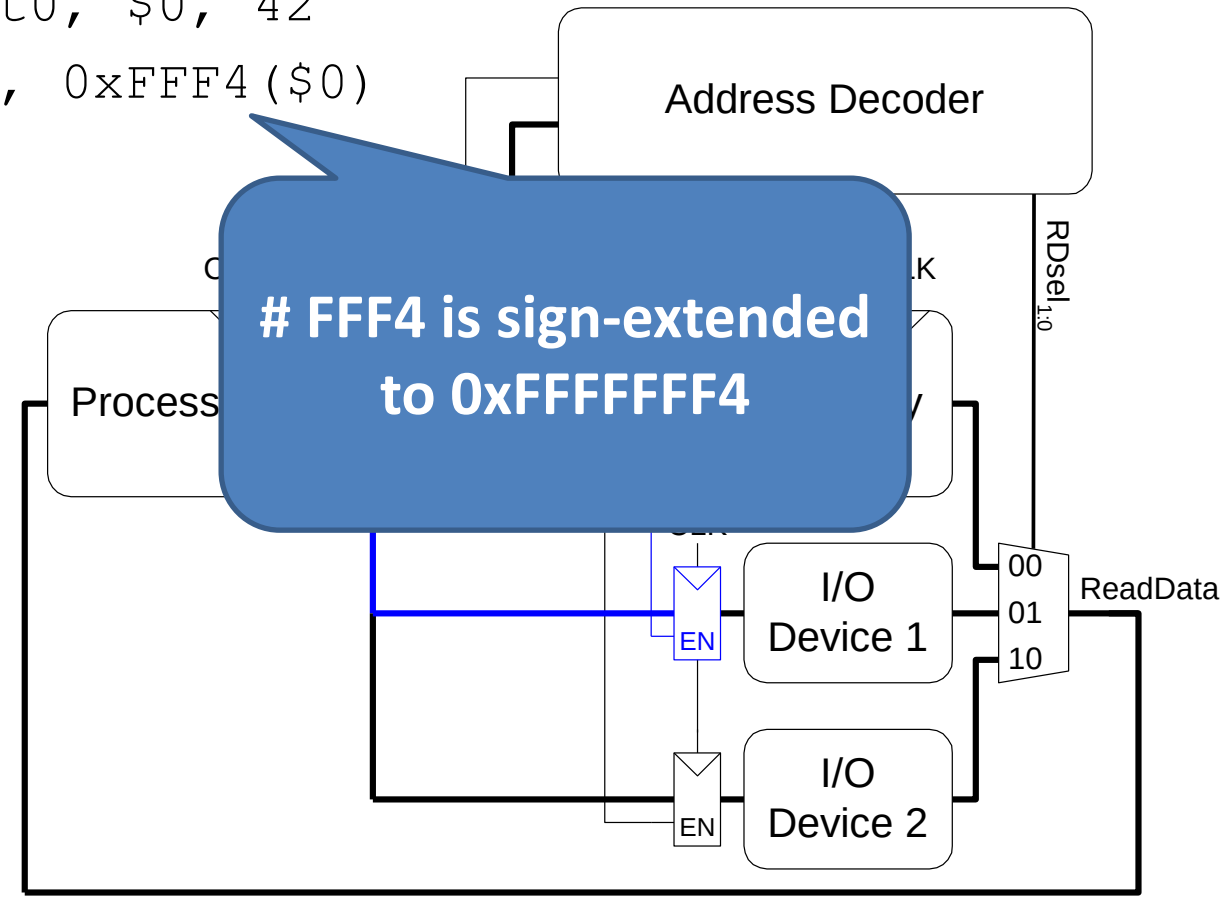
```
addi $t0, $0, 42
sw $t0, 0xFFF4($0)
```



Memory-Mapped I/O Code

- **Write the value 42 to I/O Device 1 (0xFFFFFFF4)**

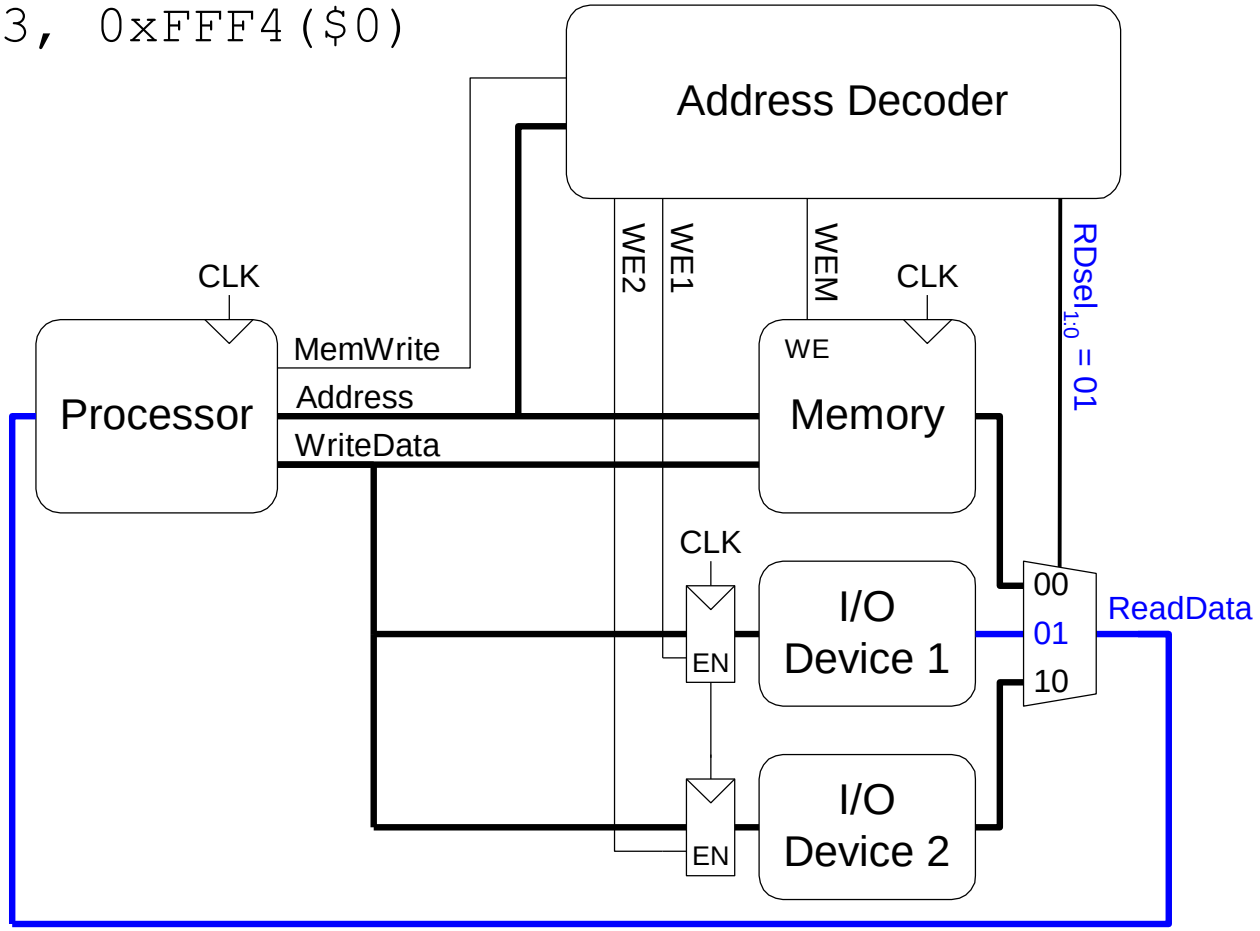
```
addi $t0, $0, 42
sw $t0, 0xFFF4($0)
```



Memory-Mapped I/O Code

- **Read the value from I/O Device 1 and place in `$t3`**

```
lw $t3, 0xFFF4($0)
```

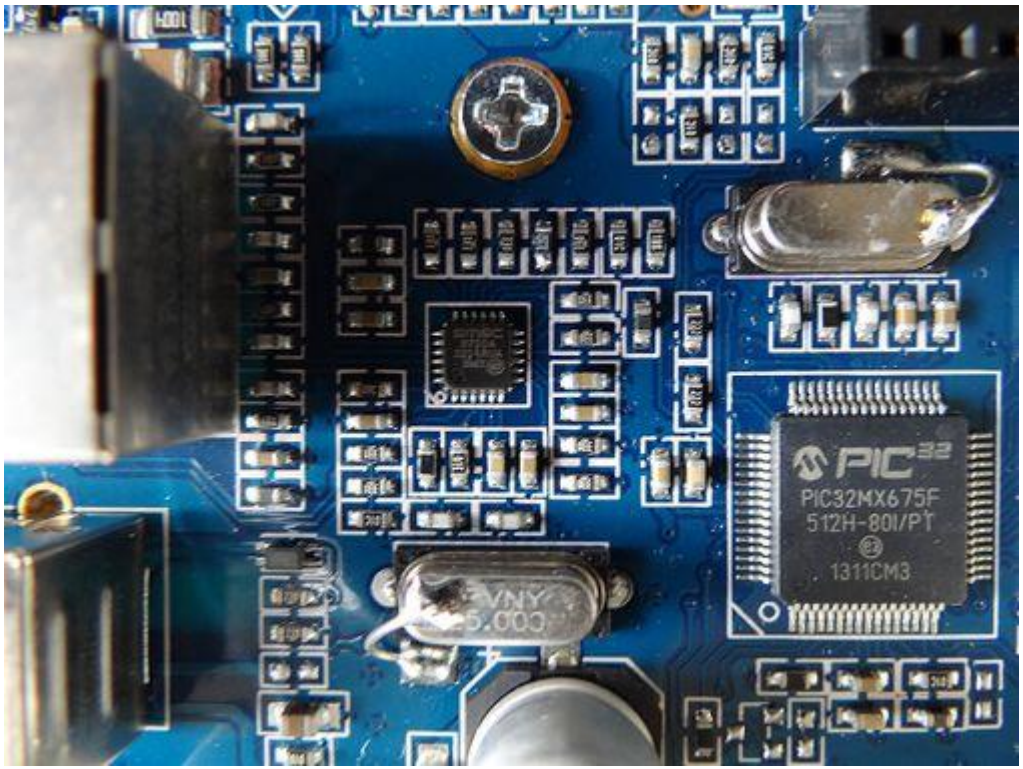


Embedded Systems

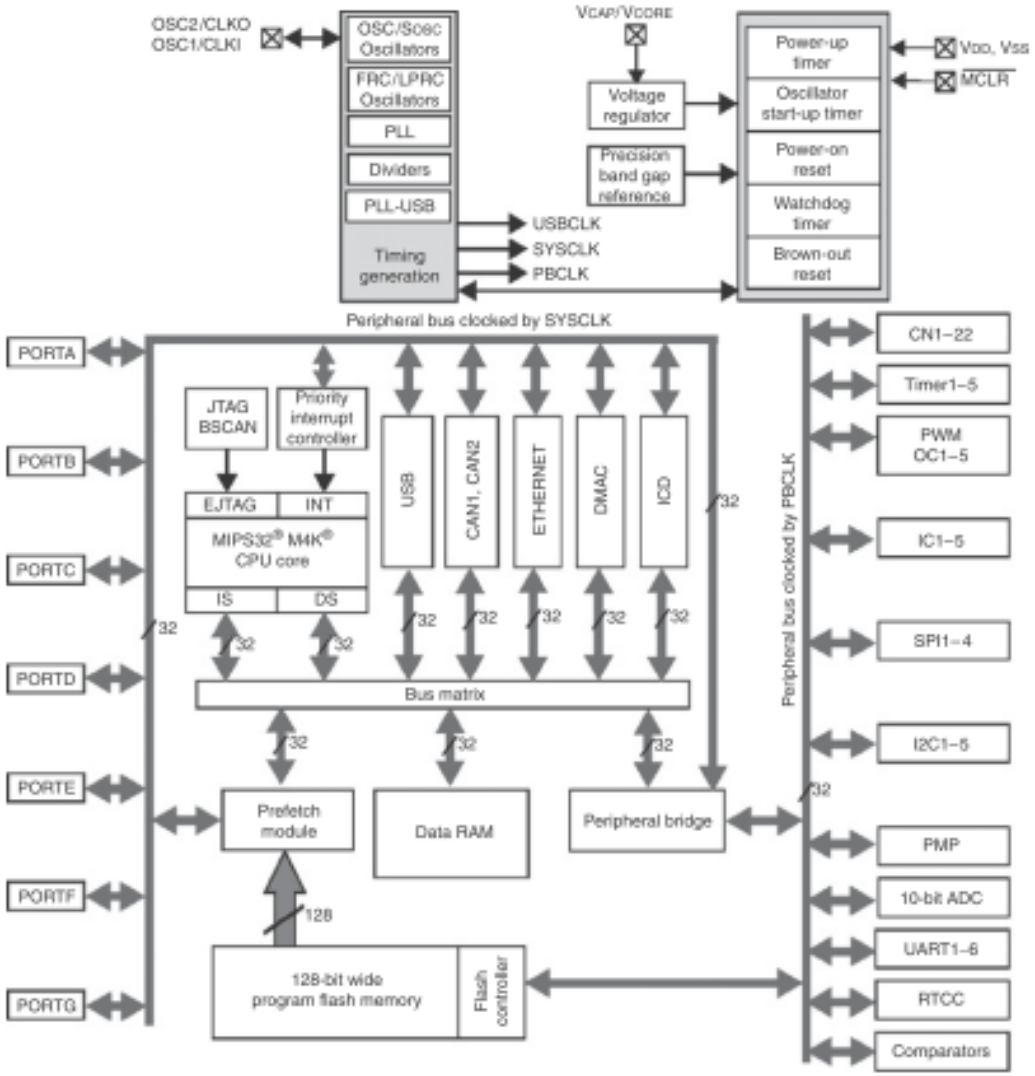
- Embedded systems use a processor to control interactions with the physical environment
- They are typically built around *microcontroller units* (MCUs) which combine a microprocessor with a set of easy-to-use peripherals such as general-purpose digital and analog I/O pins, serial ports, timers, etc.
- Microcontrollers are classified by the size of data that they operate upon
- 8-bit microcontrollers are the smallest and least expensive, while 32-bit microcontrollers provide more memory and higher performance.

PIC32MX675F512H

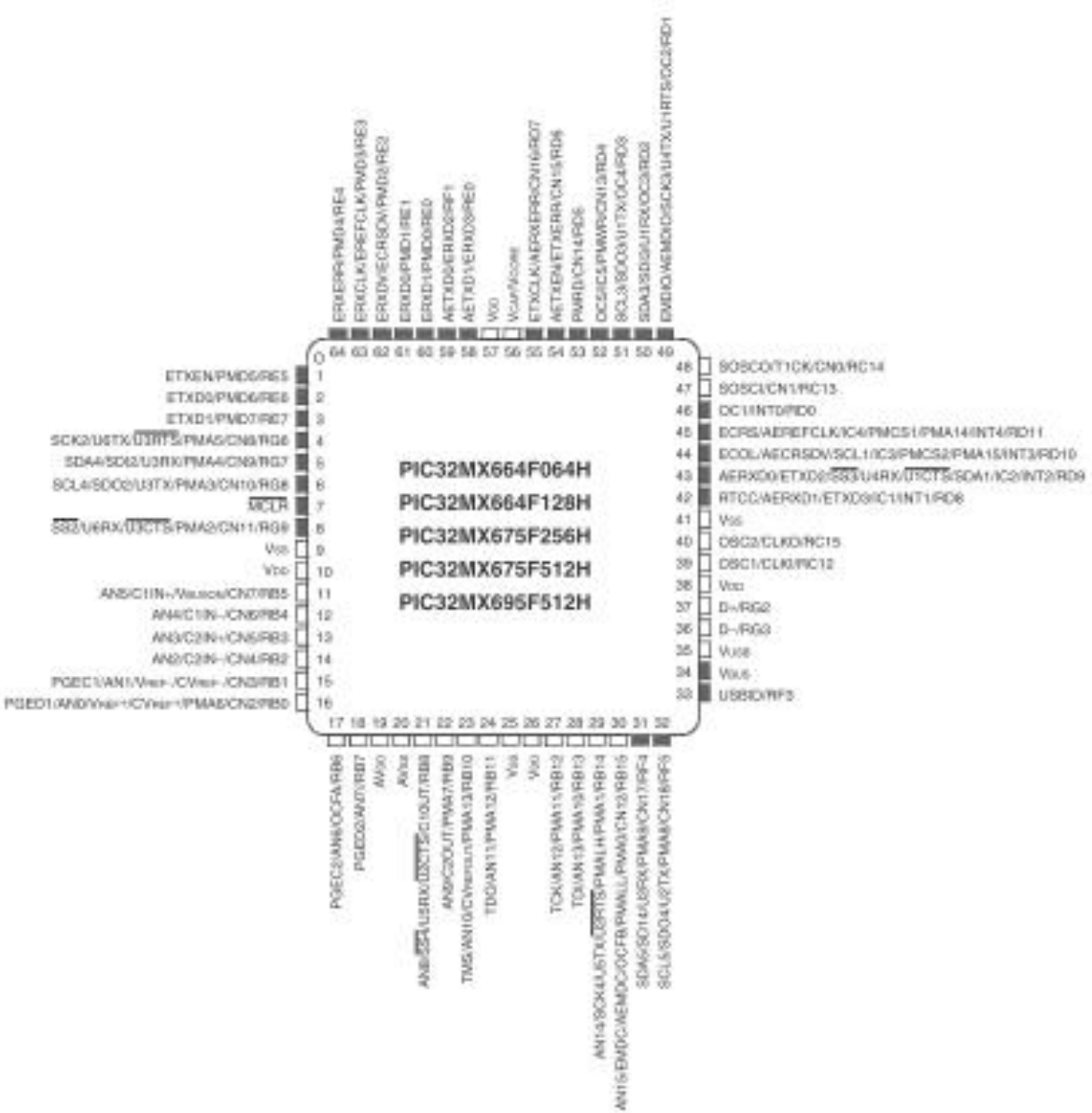
- We will focus on the PIC32MX675F512H, a member of Microchip's PIC32-series of microcontrollers based on the 32-bit MIPS microprocessor.



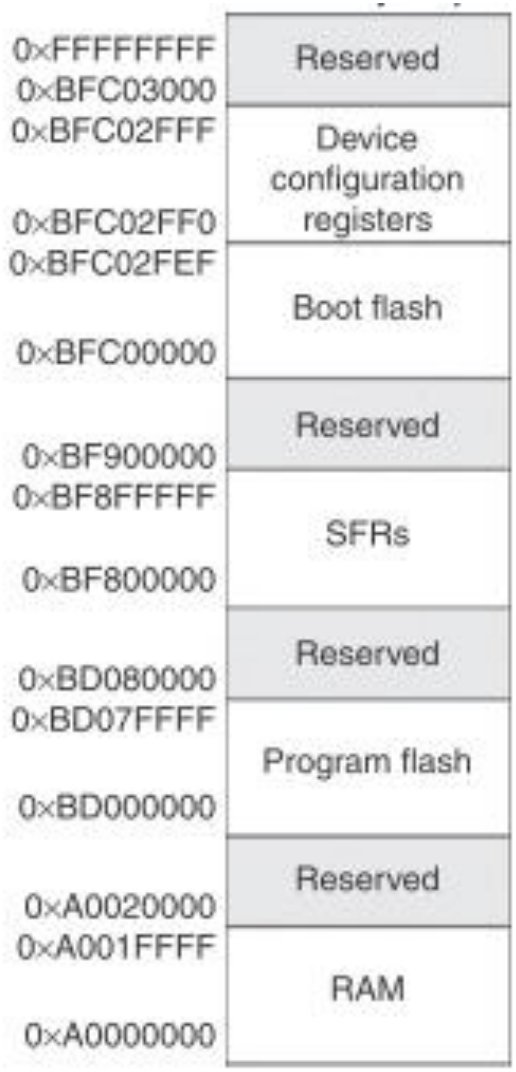
Block diagram



Pinout


PIC32MX664F064H
PIC32MX664F128H
PIC32MX675F256H
PIC32MX675F512H
PIC32MX695F512H

Memory map


0×FFFFFFFF
0×BFC03000
Reserved
0×BFC02FFF
0×BFC02FF0
Device configuration registers
0×BFC02FEF
0×BFC00000
Boot flash
0×BF900000
Reserved
0×BF8FFFFF
0×BF800000
SFRs
0×BD080000
Reserved
0×BD07FFFF
0×BD000000
Program flash
0×A0020000
Reserved
0×A001FFFF
0×A0000000
RAM

In-Circuit Debugger (ICD)

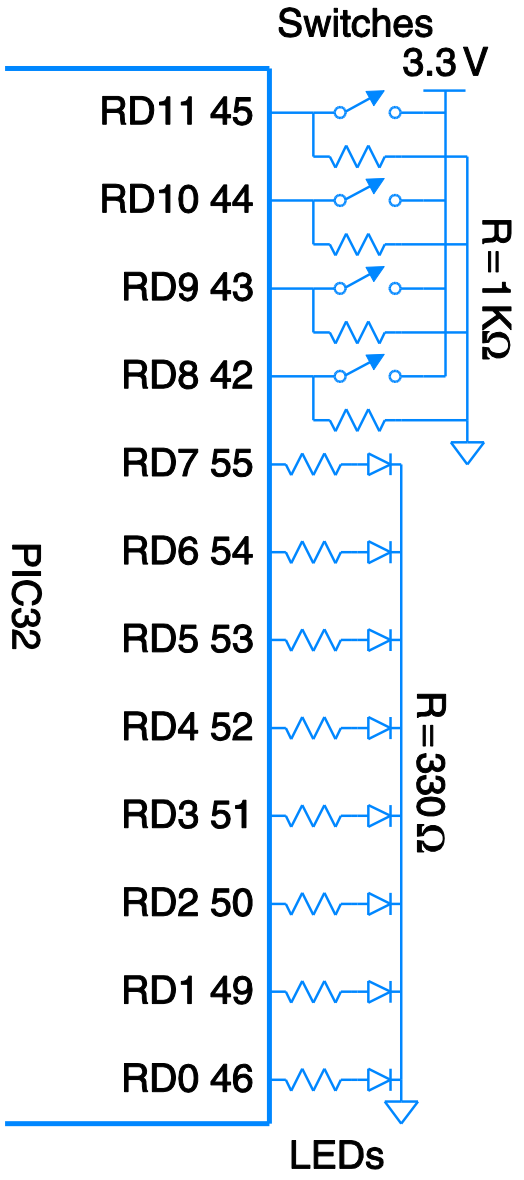
- The easiest way to program the microcontroller is with a Microchip In-Circuit Debugger (ICD).
- The ICD allows the programmer to communicate with thePIC32 from a PC to download code and to debug the program.
- The ICD connects to a USB port on the PC and to a six-pin RJ-11 modular
- connector on the PIC32 development board.
- The ICD3 communicates with the PIC32 over a 2-wire In-Circuit Serial Programming interface with a clock and a bidirectional data pin.
- You can use Microchip's free MPLAB Integrated Development Environment (IDE) to
  - write your programs in assembly language or C
  - debug them in simulation
  - download and test them on a development board by means of the ICD.

I/O pins

- Most of the pins of the PIC32 microcontroller default to function as general-purpose digital I/O pins.
- Because the chip has a limited number of pins, these same pins are shared for special-purpose I/O functions such as serial ports, analog-to-digital converter inputs, etc., that become active when the corresponding peripheral is enabled.
- It is the responsibility of the programmer to use each pin for only one purpose at any given time.

General Purpose I/O

- General-purpose I/O (GPIO) pins are used to read or write digital signals.
- The Figure shows eight light-emitting diodes (LEDs) and four switches connected to a 12-bit GPIO port.
- The schematic indicates the name and pin number of each of the port's 12 pins; this tells the programmer the function of each pin and the hardware designer what connections to physically make.
- The LEDs are wired to glow when driven with a 1 and turn off when driven with a 0.
- The switches are wired to produce a 1 when closed and a 0 when open.
- The microcontroller can use the port both to drive the LEDs and to read the state of the switches.



I/O ports

- The PIC32 organizes groups of GPIOs into *ports* that are read and written together.
- The PIC32 calls these ports RA, RB, RC, RD, RE, RF, and RG; they are referred to as simply port A, port B, etc.
- Each port may have up to 16 GPIO pins, although the PIC32 doesn't have enough pins to provide that many signals for all of its ports.

TRISx and PORTx

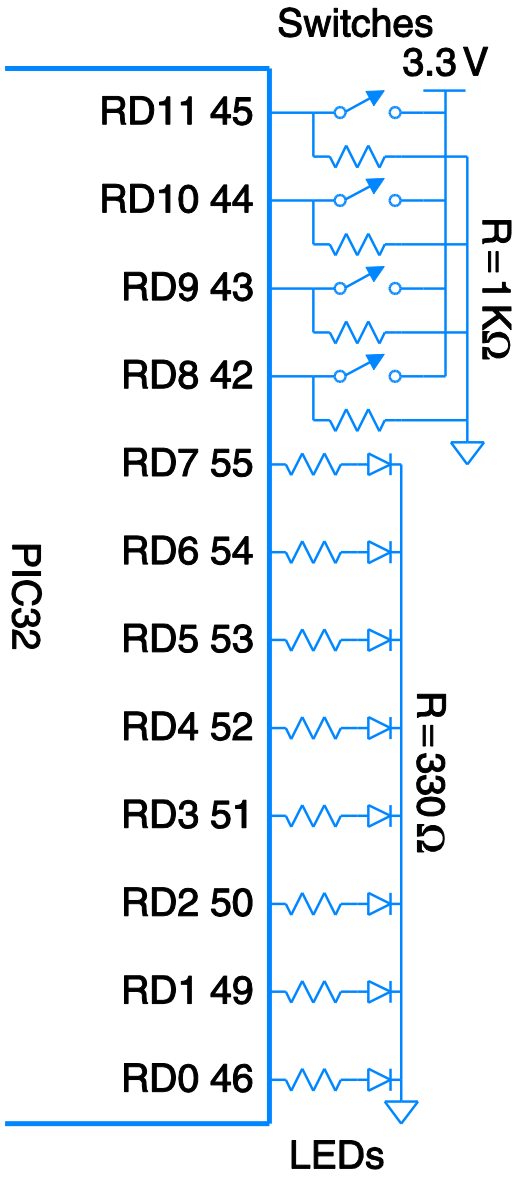
- Each port is controlled by two registers: TRISx and PORTx, where x is a letter (A-G) indicating the port of interest.
- The TRISx registers determine whether the pin of the port is an input or an output.
- The PORTx registers indicate the value read from an input or driven to an output.
- The 16 least significant bits of each register correspond to the sixteen pins of the GPIO port.
- When a given bit of the TRISx register is 0, the pin is an output, and when it is 1, the pin is an input.
- It is prudent to leave unused GPIO pins as inputs (their default state) so that they are not inadvertently driven to troublesome values.

Port addresses

- Each register is memory-mapped to a word in the Special Function Registers portion of the memory (0xBF80000-BF8FFFF).
- For example, TRISD is at address 0xBF8860C0 and PORTD is at address 0xBF8860D0.
- The p32xxxx.h header file declares these registers as 32-bit unsigned integers.
- Hence, the programmer can access them by name rather than having to look up the address.

Example

- Write a C program to read the four switches and turn on the corresponding bottom four LEDs using the shown hardware.
- Solution
- Configure TRISD so that pins RD[7:0] are outputs and RD[11:8] are inputs.
- Then read the switches by examining pins RD[11:8], and write this value back to RD[3:0] to turn on the appropriate LEDs.



Digital I/O

```
// C Code
#include <p3xxxx.h>

int main(void) {
  int switches;

  TRISD = 0xFF00;        // RD[7:0] outputs
                         // RD[11:8] inputs
  while (1) {
    // read & mask switches, RD[11:8]
    switches = (PORTD >> 8) & 0xF;
    PORTD = switches;   // display on LEDs
  }
}
```