Embedded systems are product sub-systems controlled by a special-purpose computer. The computer provides software-control by using transducers called actuators and sensors.

- **Sensors** are transducers that convert physical energy to electrical energy. In general, sensors ***meter*** the environment by measuring some change in a physical parameter.
- **Actuators** are transducers that convert electrical energy to physical energy. In general, actuators ***change*** the environment by using electricity to change a physical parameter.
- **Microcontrollers** are single-chip computers that are the brain of ***most*** embedded systems.
- **Figure 1** is a basic model used to diagram embedded systems that use microcontrollers.

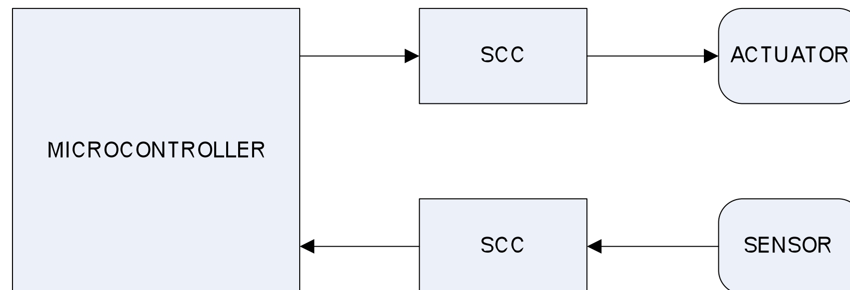

Figure 1: The Basic Embedded System Model

Microcontrollers contain built-in ***input and output devices*** (I/O) to coordinate the movement of sensor and actuator data. In general, the built-in I/O devices share the pins of the microcontroller to help reduce the total pin count. The ATmega32 pin interface is shown in Table 1.

TABLE 1: ATmega32 PORT PINS		
PORT NAME	**PINS AND DIRECTIONS**	**SHARED BY THESE I/O DEVICES**
PORTA	8 bidirectional pins	Analog-to-digital converter
PORTB	8 bidirectional pins	SPI serial, analog comparator, timer, external interrupts
PORTC	8 bidirectional pins	Timer, JTAG in-system programming, two-wire serial
PORTD	8 bidirectional pins	Timer, external interrupts, UART serial

I/O control registers hold binary numbers that allow user programmatic control of I/O devices. The embedded system software will use different types of control registers to configure and interact with the I/O devices.

- **Configuration control registers** configure I/O devices that use port pins.
- **Direction control registers** control the direction of the bidirectional port pins.
- **Input registers** sample sensor signals on command.
- **Output registers** control the voltage output to actuator signals.
- **Parameter registers** hold parameters needed by I/O device functions. Example include time values for timer functions and voltage values for analog comparators,
- **Result registers** hold results produced by some I/O devices.
- **Status registers** flag events for software notice.

 © Dr. Russ Meier, Milwaukee School of Engineering

I/O devices are independent hardware units that complete operating system or user program requests. Two methods are commonly implemented in hardware to allow monitoring of I/O device activity from software:

- **Completion flag bits** are provided in **status registers** so that polling loops can monitor for event completion. Polling loops read the status register flags and execute I/O device code only when flags announce the completion of the I/O request. ***Polling loops do not meet real-time constraints*** because the loop overhead suggests that an event could complete some number of instructions before the flags are checked again.
- **Interrupt signal voltages** are provided to **announce** the event completion in real-time. The microcontroller checks for I/O device interrupt voltages at the completion of each instruction and it automatically executes an **interrupt service subroutine** for each interrupt signal. Thus, the **interrupt abstraction** provides a method to **meet real-time constraints** by avoiding user program overhead.

Interrupt service subroutines (ISRs) are device driver subroutines written by the engineer to handle the completion of I/O events.

- **Device drivers** are sets of subroutines used to control I/O devices.
- **Device drivers** do not have to contain ISRs if the system software will only interact with the I/O device through completion flag polling.
- **ISR** starting addresses are stored in an **interrupt vector table** so that the microcontroller knows which ISR the engineer wants executed when the interrupt is signaled. The vector table is located at fixed memory locations and the engineer must write the starting address for the ISR to the table using assembly language instructions or appropriate constructs in high-level languages. **Refer** to the manufacturer datasheet for the microcontroller to find the vector table locations.

ISRs are special subroutines because they were not called by a user program but were instead executed automatically by the CPU in response to an I/O event signal. **Interrupt processing** is handled automatically by the microcontroller when an interrupt is signaled.

- Additional interrupts are disabled to prevent interrupting the interrupt and missing the real-time constraint.
- The current program counter is automatically saved to the stack so that software execution can resume at the same spot after the interrupt.
- The interrupt service routine address is found in the interrupt vector table and written to the program counter so that the first instruction of the ISR is executed.
- The interruption of the user program **requires** that the values of all CPU registers must be preserved through the ISR because the user program data is contained in the CPU registers. This is known as **preserving machine state**.

© Dr. Russ Meier, Milwaukee School of Engineering

- **Some microcontrollers** have interrupt hardware that **automatically preserve** machine state by pushing the registers to the stack and restoring them at the end of the ISR. For example, the Motorola MC68HC11 stacks all registers automatically.
- **Other microcontrollers** require the engineer to include appropriate state preservation instructions in the assembly language of the ISR. The ATmega32 is an example microcontroller that requires the engineer to write the machine preservation instructions into the ISR as push-pop pairs.
- The **status register** should also be preserved because it contains a snapshot of the last calculation instruction. **Remember: interrupts were not expected so the main program may need the status register information when it is restarted.** On the Atmega32, the status register can be saved as shown in the ISR skeleton:

```
ISR:
  push r16
  in r16,sreg ; save sreg in r16 – r16 pushed first
  push any others
  do ISR work
  pop any others
  out sreg,r16 ; restore
  pop r16
reti
```

- The final instruction of the ISR is a special assembly language instruction. This instruction causes the un-stacking of any automatically stacked machine state registers, the PC, and re-enables interrupts.

 © Dr. Russ Meier, Milwaukee School of Engineering