PDHonline Course E332 (4 PDH)

Microcontrollers: An Introduction

Instructor: Mark A. Strain, P.E.

2020

PDH Online | PDH Center

5272 Meadow Estates Drive
Fairfax, VA 22030-6658
Phone: 703-988-0088
www.PDHonline.com

An Approved Continuing Education Provider

Table of Contents

Introduction

Microcontrollers and microprocessors whether seen or unnoticed are an integral part of everyday life from the time you get up in the morning to the time you retire at night. You quite possibly encounter hundreds everyday.

Think about your day. Your alarm clock that woke you up contains a processor that controls the display, the clock and the radio. Your coffee maker that came on at a preset time contains a processor that controls the clock as well as your electric toothbrush that alerted you when to recharge the battery. The smart phone that you picked up to check your email and messages contains a handful of processors: one to control the display, keypad and user applications, one to control the GPS, one or two to control the phone radio system and one to control the Bluetooth transceiver. Your television that you turned on to get the news contains processors to control the display, receiver and remote as well as one for the cable box. After getting dressed and getting into the car you encountered a couple more: one in your remote garage door opener and one in the alarm system of your house. Your car in which you drove into work contains a dozen or so processors for the engine electronic control system, GPS, radio system, electronic compass, etc.

Needless to say these tiny little black silicon brains have embedded themselves as an inseparable part of everyday life. Seemingly complicated technology within the tiny black square, when broken down into its individual subsystems, the tiny device becomes easily understood.

Architecture

Microcontrollers are composed of the following main subsystems: the central processing unit (CPU) or core, a memory interface (for data memory and program memory) and peripherals.

There are two main categories of memory: nonvolatile and volatile. Nonvolatile memory is that which retains its memory after power is removed. Disk storage and flash memory are examples of nonvolatile memory. Volatile memory is that which is erased during a power cycle. Random access memory (RAM) is an example of volatile memory.

Program memory is memory where the program code or set of instructions for the machine resides. The program is usually written in a higher level language which is compiled and translated into machine code to be interpreted by the central processing unit (CPU). Program memory is permanent storage. It is nonvolatile; the data is persistent over power cycles.

Data memory is memory that the machine uses to temporarily store data and to store variables during execution. Sometimes the entire computer program is copied from program memory into data memory and execution begins in the area of data memory where the program is copied. Data memory is temporary storage. It is volatile; the data does not exist if power is removed.

Three types of processor architectures will be discussed in this course: the Harvard architecture, von Neumann architecture and the modified Harvard architecture. The differences in the three designs focus on the processor's access to program memory and data memory.

Harvard Architecture

The Harvard architecture is a microprocessor design with separate memory and memory busses for program and data memory. A machine with a Harvard architecture has separate data and address busses for program and data [4].

The advantage of a machine with a Harvard architecture is that simultaneous access to program and data is possible. Many microcontrollers use this architecture to speed processing by simultaneous access to program code and data. Microcontrollers typically have small amounts of program (usually flash memory) and data (RAM) memory. Examples include the Atmel AVR and Microchip PIC microcontrollers.

In a machine incorporating a Harvard architecture program and data memory can have different timing and bus width. Instruction fetches and data access do not share the same path from the CPU to memory, so there is no collision of data (or bus contention) and a data access operation does not interfere with an instruction fetch. This feature makes a microprocessor designed with a Harvard architecture faster than one with a single bus from the CPU to memory.

In a Harvard architecture the data address 0x00000000 is not the same as program address 0x00000000. Program address 0x00000000 is typically where a processor begins execution out of reset. Figure 1 illustrates the Harvard architecture.

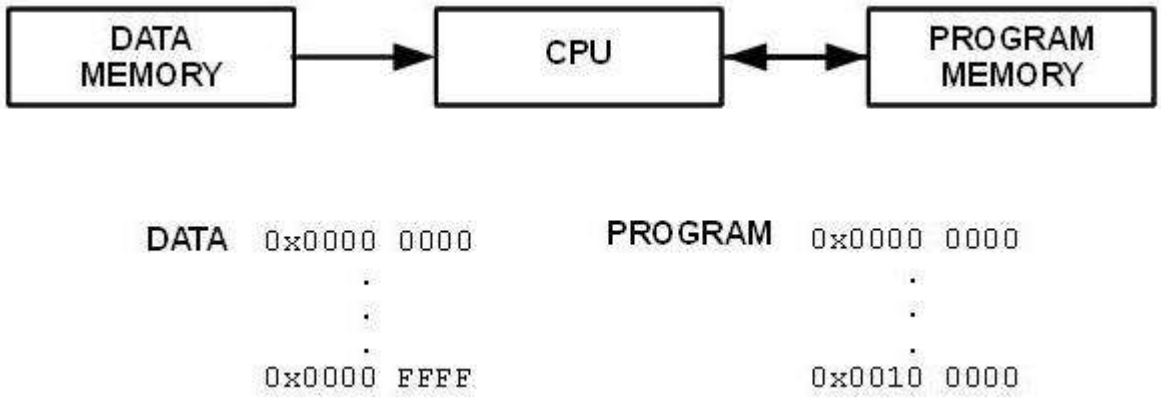

Figure 1 - Harvard Architecture

The Harvard architecture contains a program address 0x00000000 and a data address 0x00000000.

Von Neumann Architecture

The von Neumann architecture is named after John von Neumann (a renowned mathematician in the 1940s). The von Neumann architecture is a microprocessor design with the same memory bus for program and data memory. A machine with a von Neumann architecture stores data and program instructions in the same memory area – maybe even the same memory device [10].

The von Neumann architecture is called a stored-program architecture. This means that the machine's program instruction is read from some form of nonvolatile storage (program memory) such as a disk drive or flash memory device and written into volatile RAM (data memory). Once

the program is written to RAM execution is transferred to the program in RAM and the program runs from RAM. Here the program instructions and the data are located in the same memory device. Code can be treated as data and data can be treated as code. Figure 2 illustrates the von Neumann architecture.

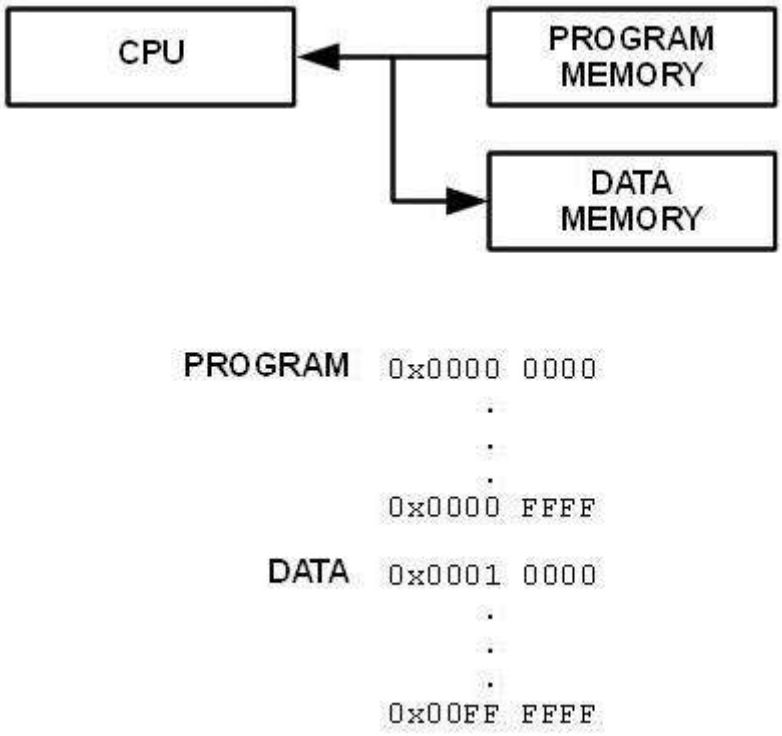

Figure 2 - von Neumann Architecture

The von Neumann architecture contains only one address 0x00000000.

Modified Harvard Architecture

The modified Harvard architecture is a design based on a combination of both the Harvard and von Neumann architectures. The modified Harvard architecture allows for the contents of program memory to be accessed as data memory. It allows concurrent access to both memory busses, thus relaxing the strict separation between instructions and data as in the Harvard architecture.

The following three characteristics describe processors based on the modified Harvard architecture [6]:

1. Program and data memories occupy different address spaces so that there is only one address 0x00000000.

   An address space defines every possible memory location for the processor. For example, the reset vector (or the first instruction that the processor executes out of reset) may be located at address 0x00000000, the first data memory address may be located at

0x00010000, and a peripheral like Timer1 may be located at address 0xF0010000. Therefore, program memory will have different address locations than data memory.

Processors based on von Neumann and modified Harvard architectures have a single address space.

2. Program and data memories have separate busses to the CPU.

   The separate busses allow program instructions and data memory to be accessed simultaneously, thus improving throughput. Pure Harvard machines have separate pathways for data and program memory. Modified Harvard machines have memory caches or other tightly coupled memory with separate paths for program instructions and data memories.

3. Instruction and data memory may be accessed in different ways.

   Program instructions may be stored in nonvolatile flash memory and the data may be located in some form of RAM memory device, but the access to each is uniform, i.e. the same width, 16 bits for example.

Examples of processors incorporating the modified Harvard architecture include the x86, ARM, MIPS, Blackfin and PowerPC [6].

Central Processing Unit

The CPU is sometimes called the “core”. It is called the core because it is the center or heart of the computational system. The CPU performs all of the calculations that occur and is the primary component carrying out the microcontroller’s or microprocessor’s functions.

The CPU is composed of the arithmetic logic unit, the control unit and registers.

Arithmetic Logic Unit

Almost all microcontrollers contain an arithmetic logic unit (ALU). It is a fundamental building block of the CPU. The ALU is a digital circuit that performs arithmetic and logical operations. The arithmetic operations include addition, subtraction, multiplication and division. These are integer operations, i.e., the inputs (or operands) are integers as well as the output. Some of the more complex processors will contain a floating point unit that handles fractional numbers in addition to integers. The logic operations include AND, OR, NOT and XOR as well as logical comparisons and bit shifting left and right.

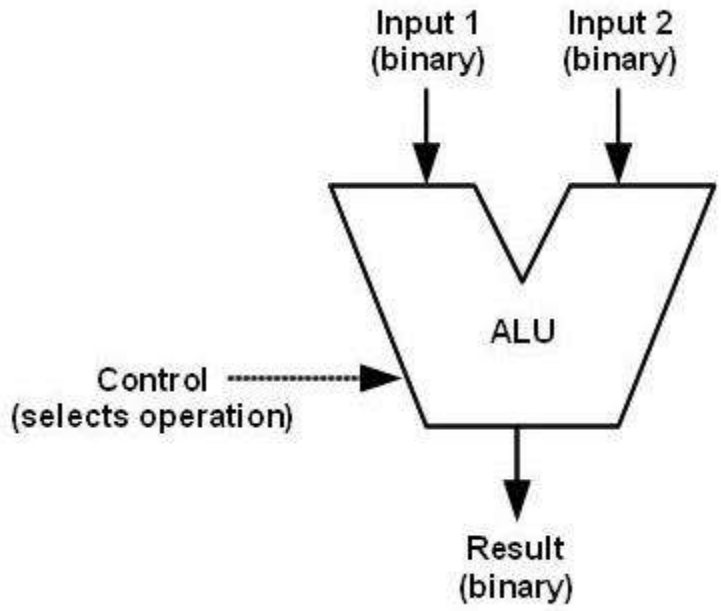

Figure 3 - Arithmetic Logic Unit

The ALU performs arithmetic and logic operations on two binary numbers resulting in another binary number. The numbers (the operands and the result) are represented using two's complement format. Two's complement allows subtraction to be performed by adding the complement of a number instead of subtracting the number. The utilization of two's complement greatly simplifies the ALU circuit since addition is a fundamental operation.

The following is an example of a two's complement operation (subtracting 109 from 202):

```
1100 1010  (202)
0110 1101  (109)
```

Take the two's complement of –109:

```
0110 1101  (109)
```

Invert all bits

```
1001 0010
```

Add one (this is the two's complement of –109)

```
1001 0011
```

Now add the two numbers:

```
1100 1010
1001 0011
----------
0101 1101
```

The result is 93.

Multiplication and division are performed by bit shifting. Multiplication is performed by bit shifting to the left and division is performed by bit shifting to the right. A shift left by one bit is equivalent to multiplying the number by 2:

```
58 x 2 = 116

0011 1010  (58)
```

Shift the number left by one bit:

```
0111 0100  (116)
```

A shift left by two bits is equivalent to multiplying by 4:

```
58 x 4 = 232

0011 1010  (58)
```

Shift the number left by two bits:

```
1110 1000  (232)
```

Similarly, a shift right by one bit is equivalent to dividing by 2:

```
172 / 2 = 86

1010 1100  (172)
```

Shift the number right by one bit:

```
0101 0110  (86)
```

A shift right by two bits is equivalent to dividing by 4:

```
172 / 4 = 43

1010 1100  (172)
```

Shift the number right by two bits:

```
0010 1011  (43)
```

The inputs to the ALU are the operands (or the data to be operated on) and the control lines to the ALU. Figure 3 shows the inputs and outputs of the ALU. The control lines pass operation codes to the ALU which select which operation to perform, e.g. addition. The output of the ALU is the result of the computation. The inputs will either come from a CPU register or a memory location. The result will also be stored either in a register or a memory location.

Control Unit

The control unit functions as a fundamental building block of the CPU just like the ALU. It contains digital circuitry that directs the processor to carry out stored program instructions. The control unit configures the ALU for a particular operation and as necessary the memory busses for a read or write operation. It then selects a particular register for a read or write operation.

The control unit directs the CPU to perform three basic functions: fetch, decode and execute. The fetch-decode-execute cycle characterizes the heart of the number crunching factory. The control unit is the facilitator of this cycle. The control unit fetches instructions from program memory. It then decomposes (or decodes) the instruction into its basic elements (opcode and operands).

During this phase it configures the ALU, memory and register bank for the specific operation. After decoding the instruction the control unit loads the ALU with a combination of register values and/or memory contents thereby executing the instruction. The control unit then sends the result either to a memory location or to another register.

The control unit of a CPU is essentially a complicated finite state machine. A state machine is a model used to describe the behavior of a system. The word "finite" is used because the state machine has a limited, i.e. not infinite number of states or modes. A state machine is composed of states, transitions and actions. It is a system whose output depends not only on the input to the system but also on the history of the system (or the current state of the system).

Registers

Just like the ALU and control unit the CPU registers are also a fundamental building block of the CPU. Registers are temporary storage areas for instructions and data. They do not contain a lot of memory space. They are only a few bytes wide, typically 8 bits, 16 bits, 32 bits or 64 bits wide. Registers are not part of main memory. They are special storage locations outside of program and data memory. They are not part of the processor's memory map.

Registers are an important component of the CPU. The control unit controls the reading and writing of the CPU registers. They hold and transfer instructions and data and hold data for the ALU to perform arithmetic and logic operations. Registers are typically made from individual flip-flops (one per bit), shown in Figure 4, or from high speed core memory. Registers are much faster than conventional memory like static and dynamic RAM. They have a much faster access time. The core benefits from the registers' fast access time in terms of processor speed.

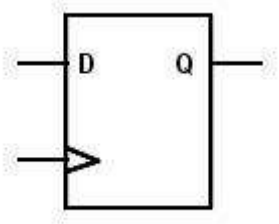

Figure 4 - D Flip-Flop

Figure 5 gives an example of an 8-bit register made of D flip-flops. The D0 through D7 values are the inputs to the register. The Q0 through Q7 values are the outputs of the register. The "CLK" input is the processor's internal clock. During a write operation, data will be presented to D0:D7. Data is clocked in during the next clock cycle. During a read operation, data is presented to the outputs during the next clock cycle. The register will be enabled or disabled by another control line controlled by the processor's control unit. Data that is written to the register is stored there until a new value is presented or power is lost to the register. Depending on the processors complexity, a processor register's width will depend on the processor's complexity. The more complex processors will have wider register banks.

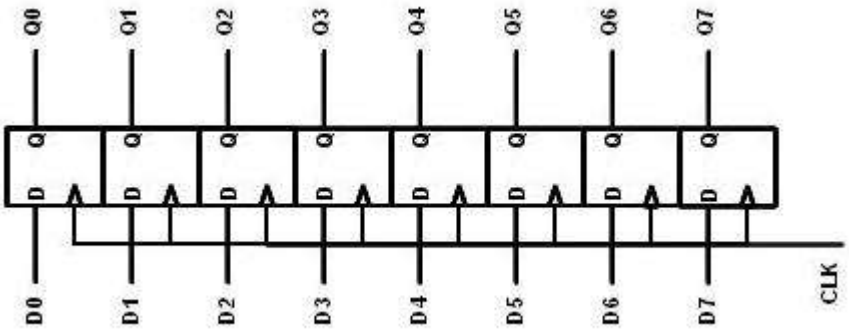

Figure 5 - An 8-bit Register Made of D Flip-Flops

Registers provide the CPU with the fastest method of accessing data. During a fetch-decode-execute cycle data is loaded from main memory into registers, manipulated and then stored back into memory. All arithmetic and logic operations occur in the CPU registers.

There are several types of registers. General purpose registers store both data and addresses. An accumulator register collects the results of computations from the ALU. An address register stores the location in memory of an instruction or data. Special purpose registers store program execution state, like the program counter, stack pointer and status register.

Cycle of the Central Processing Unit

The CPU performs all of the calculations that occur and is the primary component carrying out the microcontroller's or microprocessor's functions. The main building blocks of the CPU include the ALU, the control unit (instruction fetch and instruction decoder) and registers. Figure 6 shows a block diagram of the brains of a microcontroller: the CPU.

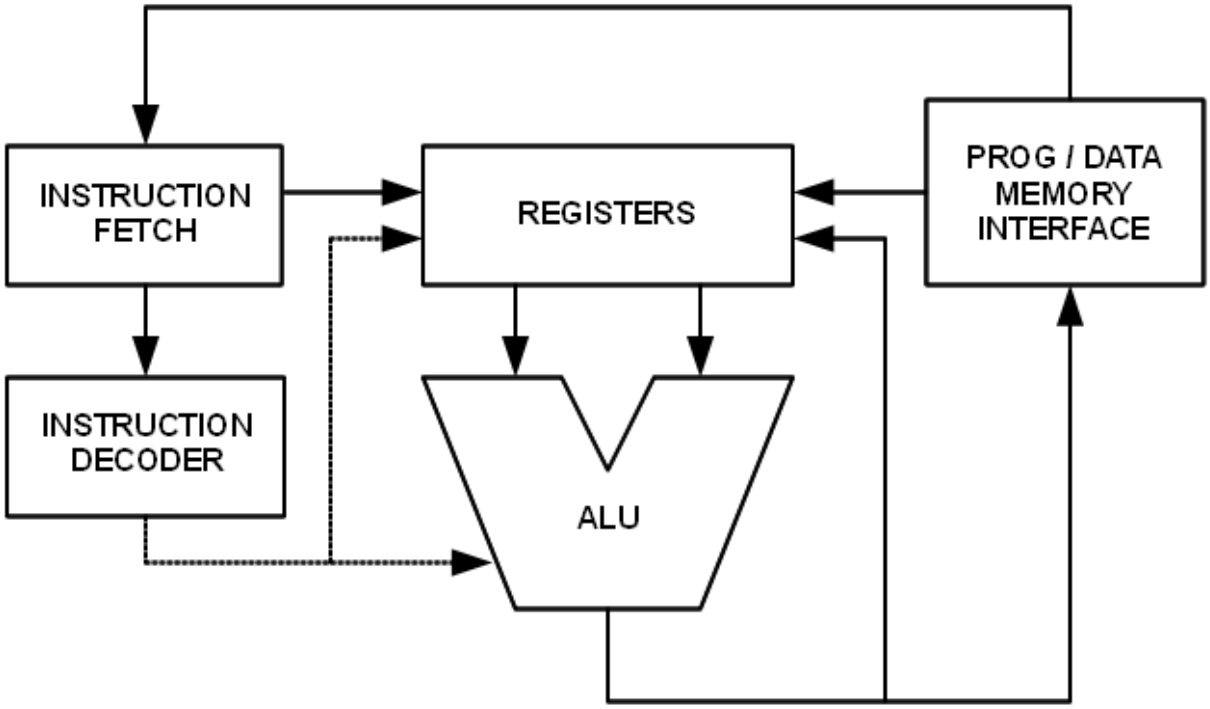

Figure 6 - Block Diagram of CPU Core

The control unit is a complete computational engine; it is a digital circuit that extracts instructions from memory, decodes the instructions and executes them. The CPU is the portion of the microcontroller or microprocessor that carries out the instructions of the stored computer program. The CPU contains a limited set of discrete states. The states are switched by millions of transistors on the processor chip.

The CPU's operation is governed by what is called the fetch-decode-execute cycle. The CPU executes a binary file that contains instructions in a particular sequence so that the processor steps through the instructions and executes (or runs) the program. The fetch-decode-execute cycle is analogous to loading, cocking and firing a gun.

Fetch Cycle

The fetch operation retrieves an instruction from program memory. The CPU stores the instruction in the instruction register. In our analogy, the gun is now loaded. The location of the next instruction in memory to be executed is stored in a register called the program counter.

Decode Cycle

After an instruction is fetched from memory it must be decoded or interpreted. The decoding operation determines how to configure the ALU for the particular operation. The gun is now cocked. The decode operation breaks up the instruction into pieces. The meaning of the different chucks of binary bits depends upon the processor's instruction set. The instruction set can be thought of as the processor's vocabulary. Part of the instruction to be decoded is called the opcode (or operation code). This indicates which instruction to perform, for example, an AND

operation. The remaining parts include the operands for the operation. These are the inputs to the command. They can be a constant value, a register or a memory location.

Execute Cycle

After an instruction is fetched and decoded it is then executed. The gun is now fired. During the execute phase different parts of the CPU are connected to perform the desired operation. For example, during an AND operation, the ALU will be configured to perform a logical AND. The two operands (a constant, a register location or memory location) will be presented to the inputs of the ALU. The control unit will present the proper binary code to the ALU's control section to prepare it for an AND operation and the output of the ALU will contain the result. The result is then written to an internal CPU register or some memory location.

At this phase the program counter is incremented and the CPU is reconfigured to fetch the next instruction and complete the next fetch-decode-execute cycle. In our analogy, the gun readies itself to fire again.

Microcontrollers and Microprocessors

The development of mass-produced, standardized processors has revolutionized modern life with the presence of all sorts of digital devices from desktop computes, cell phones, MP3 players, modern aircraft and automobiles and factory automation. Most households contain dozens of microcontrollers and are used as an inseparable part of everyday life.

What is the difference between a microprocessor and a microcontroller? Sometimes the names are used synonymously. Sometimes the term "processor" is used when referring to a microcontroller or microprocessor. Sometimes the term "CPU" is used to refer to the entire microprocessor or microcontroller, but this is not accurate. The CPU is actually a component of the microprocessor or microcontroller.

A microcontroller is basically a specialized microprocessor. Both microcontrollers and microprocessors contain a CPU (ALU, registers and control unit), but a microcontroller usually contains a small amount of program memory (flash) and data memory (RAM). With the integrated memory, a microcontroller does not need to interface with external memory. It is compact and integrated. Microcontrollers usually contain a suite of peripherals. These peripherals may include timers, a real-time clock and serial communication controllers (for example, UARTs and I^2C and SPI interfaces). Microcontrollers may also include an analog-to-digital converter or an analog comparator. All of this functionality is integrated onto a single chip called an application-specific integrated circuit (ASIC). This single chip concept provides for a compact, self-sufficient and cost-effective solution for electronically controlling devices. [13]

A microcontroller usually performs only one task, like control the remote control for your television or control the display and keypad interface for your microwave oven. A personal computer, on the other hand, incorporates a microprocessor. This microprocessor performs multiple tasks for programs: like word processors, spreadsheets, web browsers and video games. The microcontroller usually has only one program stored in it to perform a single task for its lifetime. A microprocessor is usually embedded in a system that performs many tasks, some not even conceived of at the time of its construction.

Instruction Set

Microcontrollers and microprocessors run a series of sequential instructions called a program. The program is usually written in a high level language like C or C++. The high level language is compiled or translated by a software tool called a compiler. The compiler translates the series of instructions into machine code. The program may also be written in a lower level assembly language. A software tool called an assembler translates or assembles the series of assembly instructions into machine code. The machine code is stored in the processor's program memory.

Machine code is simply a file containing a series of ones and zeros. Some programs may be very short, for example, an infinite loop controlling a couple of output pins flashing some LEDs in a child's toy. Other programs may be very long and complicated to the computer controller in a car's engine to the guidance system on a missile. No matter how simple or complex all computer programs are translated into a file containing a stream of ones and zeros called machine code.

Machine code is interpreted by the processor's CPU. Depending on the processor's complexity, the instruction width can be 16, 32 or 64 bits wide. A processor with 16-bit wide instructions will, for example, fetch instructions 16 bits at a time. The instruction will be stored in the instruction register. It will be decoded by the CPU in which the control unit will configure the ALU for the particular operation (ADD two registers, for example) and configure memory and/or some registers for a read operation. The CPU will then execute the instruction (ADD two registers) and store the result in another register or in data memory. The following examples demonstrate how an instruction for the Atmel AVR family of microcontrollers is translated into machine-readable machine code [2].

Here is an example of an ADD instruction:

Instruction:

ADD (add without carry)

Operation:

Rd ← Rd + Rr (Rr – source register; Rd – destination register)

Syntax:

ADD Rd, Rr

Machine Code:

15	8	7	0
0000	11rd	dddd	rrrr

For example, the instruction to add register R3 to register R2

ADD R2, R3

translates to the following binary machine code (note: d = 00010, r = 00011):

0000 1100 0010 0011

or in hexadecimal:

0C23

The code "0C23" is the machine code for the ADD R2, R3 instruction in the Atmel AVR instruction set [2].

Here is an example of an AND instruction:

Instruction:

AND

Operation:

Rd ← Rd ● Rr (Rr – source register; Rd – destination register)

Syntax:

AND Rd, Rr

Machine Code:

15 8	7 0		
0010	00rd	dddd	rrrr

For example, the instruction to AND register R3 to register R2

AND R2, R3

translates to the following binary machine code (note: d = 00010, r = 00011):

0010 0000 0010 0011

or in hexadecimal:

2023

The code "2023" is the machine code for the AND R2, R3 instruction in the Atmel AVR instruction set [2].

Here is an example of copy register (MOV or move) instruction:

Instruction:

Copy Register

Operation:

Rd ← Rr (Rr – source register; Rd – destination register)

Syntax:

MOV Rd, Rr

Machine Code:

15 8	7 0		
0010	11rd	dddd	rrrr

For example, the instruction to copy register R3 into register R2

MOV R2, R3

translates to the following binary machine code (note: d = 00010, r = 00011):

0010 1100 0010 0011

or in hexadecimal:

2C23

The code "2C23" is the machine code for the MOV R2, R3 instruction in the Atmel AVR instruction set [2].

Peripherals

A peripheral is a device that operates separately but in conjunction with the CPU such as an input/output device, timer or communications device. A microcontroller is not very useful if it cannot control a device or communicate with the outside world.

Input/Output

General Purpose Input/Output

Typically a microcontroller controls a device through some sort of external means. General purpose input/output (GPIO) pins are the simplest form of input and output device. GPIO pins are configured separately either as inputs or output by setting some internal peripheral registers. A processor can have as few as one or two or up to several dozen available GPIO pins. They are small signal, low voltage and low current pins.

A processor pin usually has multiple functions. For example, a particular pin on a microcontroller depending on the device could be configured as a GPIO input or output pin, a pulse-width modulator output, or as a transmit pin for one of its onboard communication peripherals.

Configured as an output, a GPIO pin would typically be connected to a small signal switching device such as a field-effect transistor (FET) or an electromechanical relay. These can switch higher current devices, such as a motor, a solenoid, an LED or a compressor motor on a refrigerator. Configured as an input, a GPIO pin would typically be connected to a button on a keypad that a user would interface with, such as the control buttons on a microwave oven.

GPIO pins are good for simple, single-ended control, i.e. one line. However, it is common for the need for more complicated input/output devices, such as a graphics display. In this case the processor's address and data bus along with a chip select could be interfaced to the display allowing for faster, parallel access to data. Another example is an onboard high-speed serial communications device, such as universal serial bus (USB), which could be interfaced to the device.

Timer

A timer is a device that measures elapsed time or controls events during a predetermined interval. Timers and clocks are very important to microcontrollers and microprocessors. The internal processor clock is the heartbeat of the entire system. It synchronizes every event that happens: from copying a memory location from data memory to a register to servicing a task such as refreshing a display or checking for any button presses on a keypad.

All timers have a clock as a timing source. A clock is a series of regular spaced pulses as shown in Figure 7 that are generated by an oscillator. Clocks have a rising edge and a falling edge. Data is clocked in or read out of a device either on the clock's rising edge or falling edge. The difference between a clock and a timer is that a clock is a subset of a timer. A timer has a clock source of known frequency and counts the number of clock pulses to keep track of elapsed time.

Figure 7 - Series of Clock Pulses

General Purpose Timer

General purpose timers function as an integral part of many embedded systems. Most microcontrollers include one or more timers as part of their compliment of peripherals. A timer contains a counter register which counts the pulses of the timer's clock input. When the counter register fills up this is called a timer overflow. Since the clock oscillates at a known, steady frequency, this overflow will occur over a predictable time interval. When an overflow event occurs, the processor is interrupted and a particular interrupt function is called in software. With this timer overflow and software interrupt system in place a microcontroller can perform regular periodic tasks like service a human machine interface, refresh a display, read a keypad or something as simple as flash an LED. This timing ability forms the basis of a real-time operating system.

A timer's counter register is typically 8, 16 or 32-bits wide. The wider counter registers create a timer which is more precise and can measure shorter intervals with a faster clock or longer intervals with a slower clock. The input clock can be scaled (or slowed down) by prescaler registers. The prescaler will divide the incoming clock to the timer by 2, 4, 8, 16, etc. to slow down the incoming clock to give the ability to measure longer time intervals.

A timer also may contain a comparison or match register. A value can be placed in the comparison register so that during every timer clock cycle, the counter register is compared to the comparison register. If the two match then the processor is interrupted and an interrupt function is called in software. This comparison register is used to give the timer measurement system more precision.

More complicated timers may contain pulse-width modulation (PWM) registers to form a particular waveform. A PWM waveform has many uses. It can be used to dim an LED or display, to regulate the speed of a servo motor or to produce a waveform for a communications system.

Real-Time Clock

A real-time clock is a special timer with an input frequency of 32768 Hz. This produces clock pulses easily divisible into seconds so that the timer can tick down the seconds and keep track of the time of day. The real-time clock is usually interfaced into the processor's power management system so that when the processor is put into sleep mode the real-time clock still runs and keeps time.

A real-time clock is used by a system that needs to keep track of the time of day and perform events at a preset time, like record a movie on a cable television box or turn on a sprinkler system or control a thermostat for an air conditioning system. Some real-time clocks have integrated calendars to keep track of the date as well as the time.

Communications

The ability of a microcontroller to control a device or interact with a system sometimes requires more than a few simple GPIO pins that flash a couple of LEDs or turn a relay on and off. Sometimes monitoring and controlling a system requires some sort of communications. Often times a processor needs to provide some sort of status or needs to monitor data from another source. For example, a temperature sensor can provide temperature data to a processor. Air flow and fuel mixture sensors provide crucial data for a car engine's electronic control unit. The engine's electronic control unit can provide diagnostic data through a cable to a nearby computer to monitor engine performance.

Serial communications is a cost effective form of communications. The term "serial" implies that the data comes in or goes out one bit at a time. Data can be full duplex where transmission and reception happen simultaneously, or data can be half duplex where data cannot be transmitted and received at the same time.

A data link can be synchronous or asynchronous. Synchronous data contains a clock on a separate line: clock on one line and data on another. Asynchronous data does not contain a clock line. The data rate is agreed upon by the sender and receiver, or the clock is recovered from the data by mathematical means.

Data links can be single ended, i.e. a communications link between two nodes only, or data links can have many nodes like in a network. In the case of many nodes on the same bus, a serial communications protocol must be established so that the nodes can communicate.

For this course three forms of serial communication common in many microcontrollers will be discussed: universal asynchronous receiver/transmitter (UART), serial peripheral interface (SPI) and inter-integrated circuit (I^2C).

Universal Asynchronous Receiver/Transmitter (UART)

A UART is an asynchronous serial communications interface that translates data from parallel to serial format and then back to parallel on the receiving end. The UART is a digital circuit incorporated into many microcontrollers. A UART converts data bytes into a stream of individual bits and transmits the bits on a single line to a remote UART. The receiving UART in turn takes the stream of bits and translates them into individual bytes to be read by the processor. The data link utilizing a UART on each end of the transmission may be either half duplex or full duplex. Figure 8 shows two UARTs connected together forming a data link.

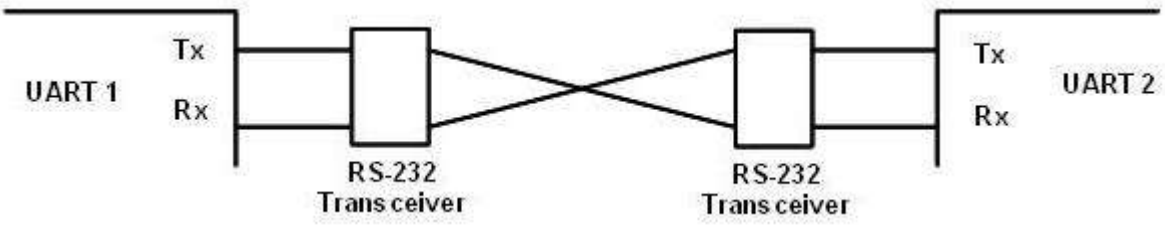

Figure 8 - UART Data Link

The signals generated by a UART are low voltage and current so external level converters are used to give the ability to transceiver longer distances (typically several feet). The level converter chips are most commonly RS-232, RS-422 or RS-485.

A UART is composed of shift registers, some special-purpose registers, and a baud rate generator. There are two shift registers, one for the transmitter and one for the receiver. The special-purpose registers are used to configure and control the UART. The baud rate generator generates the clock from the system clock using a divider circuit. The baud rate (or symbol rate) must be known by both the transmitter and the receiver. It is not derived or computed. If the transmitter and the receiver are not using the same baud rate, no valid data will be received by the receiver. A standard baud rate is typically chosen such as 1200, 2400, 4800, 9600,…, 15200, etc.

For every byte transmitted a start bit is attached to the beginning and a stop bit is attached to the end. The start bit is low (logic 0) and the stop bit is high (logic 1). A UART can be configured to have an additional stop bit. It can include parity checking, and can include an additional data bit. A UART can also be configured to control modem lines used for hardware handshaking.

Advantages of UARTs

- The data link is full duplex.
- Longer transmission distances are possible (several feet) with the addition of transceiver chips.
- Only 2 lines are required (plus a ground reference).
- Hardware handshaking is available.

Disadvantages of UARTs

- The transceiver chips consume relatively high current.
- The data rate must be agreed upon by both the transmitter and receiver.
- Hardware handshaking takes 2 to 4 additional lines.

Serial Peripheral Interface (SPI)

SPI is a synchronous serial communications interface integrated into many modern microcontrollers. The SPI standard was developed by Motorola. Like a UART a SPI peripheral also translates data from parallel to serial format and then back to parallel on the receiving end. A SPI interface is synchronous so a clock line must be used between the transmitter and receiver. A SPI interface operates in full duplex mode. A data link will have a master and a slave. The master will initiate all transmissions and a slave will respond. Multiple slave devices are allowed on the same bus with an individual chip select (or enable line) for each device. Figure 9 shows a SPI master connected to two slave devices.

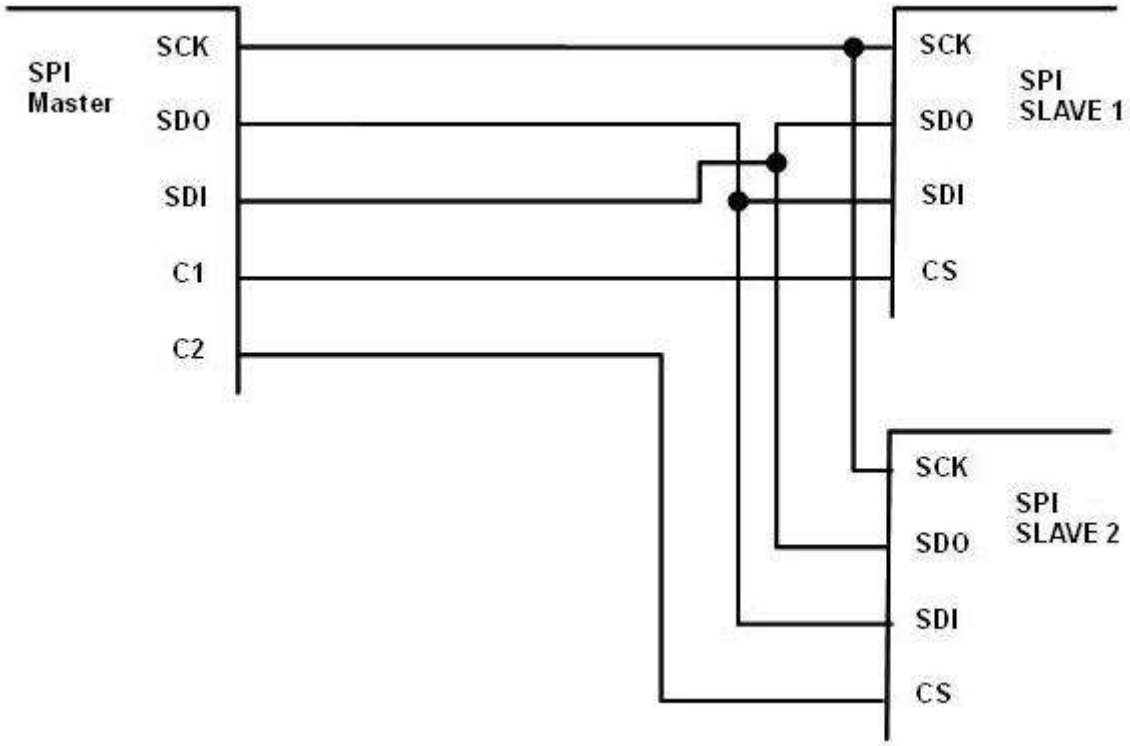

Figure 9 - SPI Data Link

The master and slaves have their clocks (SCLK or SCK) connected together. The master has its data out (SDO, DO or MOSI) line connected to the slaves' data in (SDI, DI or MISO) line. The master also has a dedicated output port pin (one for each slave) connected to the slaves' chip select (CS or SS) line. The SPI peripheral contains shift registers for the transmitter and receiver and registers that configure and control the device.

A slave will not respond unless it is selected, i.e. its chip select line is pulled low (logic 0) during the transmission. A SPI interface is typically utilized between devices on the same circuit board. For example, a microcontroller can communicate with multiple temperature sensors, an EEPROM and an analog-to-digital converter on the same circuit board on a SPI bus.

On the other hand UARTs, in conjunction with level converters, are used to communicate with devices not on the same board, like a computer capturing data from a piece of test equipment. A SPI interface does not use level converters. The SPI ports are interfaced directly.

Advantages of SPI

- The data link is full duplex.
- Slaves do not need a unique address.
- Transceiver chips are not needed.

Disadvantages of SPI

- Requires more pins than I^2C.
- A chip select (additional pin) is required for each SPI device.
- No slave acknowledgement (master does not know if the data was received).
- Only a short transmission distance is possible.

Inter-Integrated Circuit (I^2C)

I^2C (also called I2C or IIC) is a synchronous serial communications interface developed by Phillips in the early 1980s. Like a UART and SPI interface it also converts data from a parallel to serial format, transmits the data and converts it back to parallel on the receiving end. An I^2C interface is synchronous so a clock line (SCL) must be used between the transmitter and receiver. Like the SPI interface I^2C utilizes a master/slave configuration. It is used to communicate with devices on the same board like a processor communicating with a temperature sensor, analog-to-digital converter, EEPROM, etc. Figure 10 shows an I^2C master connected to two I^2C slaves.

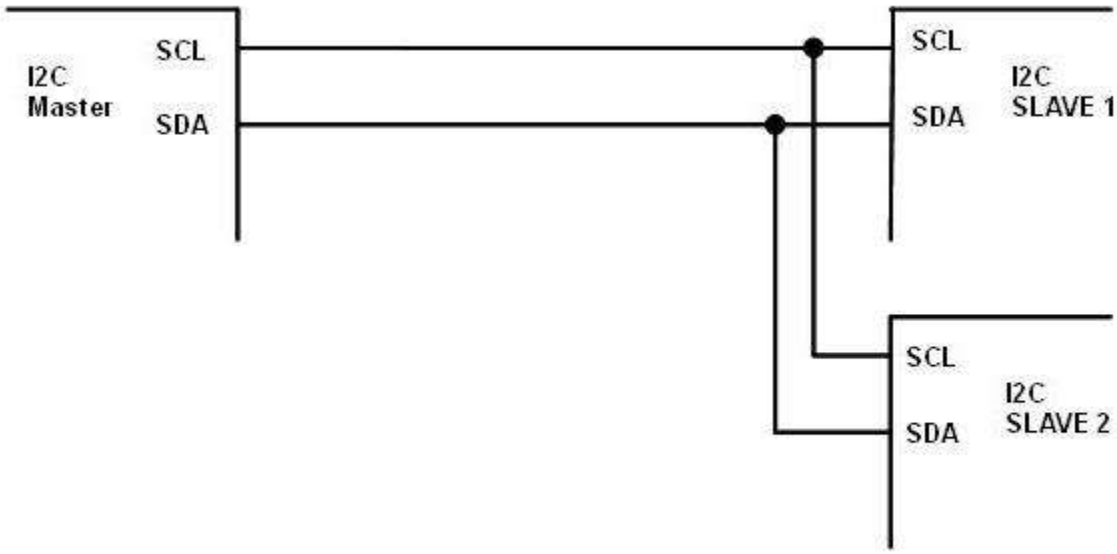

Figure 10 - I^2C Data Link

Unlike SPI devices I^2C devices do not have a chip select line. Each device is individually addressable by a unique address within the data stream. The data link is half duplex as the devices all transmit and receive data on a single data line (SDA). Each transmission is acknowledged by the receiver by the receiver pulling down the data line during a predetermined acknowledge clock pulse. With the acknowledge feature, the transmitter is ensured that the receiver received valid data.

Data transfers are made either in standard mode (100kb/s), fast mode (400kb/s) or high-speed mode (3.4Mb/s). A maximum of 112 nodes is possible due to the maximum number of addresses in its address space (7-bit address, with 16 reserved addresses). However, line capacitance may significantly reduce the allowable number of devices on the same line.

The master initiates a data transfer by generating a start condition (a high going low pulse on the data line while the clock is high) and starting the clock. The master terminates the data transfer

by generating a stop condition (a low going high pulse on the data line while the clock is high) and terminating the clock. The slave's address is sent after the initial start condition, whereby the slave acknowledges the request by holding the data line low on the 9th clock pulse.

Advantages of I^2C

- The interface only requires 2 pins.
- No separate chip select is required.
- Devices are individually addressable.

Disadvantages of I^2C

- The data link is half duplex.
- The user must configure the device for a given speed of 100kb/s, 400kb/s or 3.4Mb/s.
- Only a short transmission distance is possible.

Summary

Microcontrollers are specialized microprocessors. Three of the most popular processor architectures include the Harvard architecture where data memory and program memory are accessed separately, the von Neumann architecture where data memory and program memory are accessed from the same bus, and the modified Harvard architecture which is a combination of the previously mentioned two. The central processing unit (composed of the arithmetic logic unit, registers and the control unit) functions as the brains or core of the processor. The central processing unit processes machine code stored in memory to control all of its functions. The machine code is compiled or assembled from the processor's instruction set which defines all of the operations of the microcontroller. To complement their functionality, microcontrollers include a suite of peripherals such as input/output pins, timers, a real-time clock and communications controllers.

References

1. "Arithmetic Logic Unit - Wikipedia, the Free Encyclopedia." 4 July 2010 <http://en.wikipedia.org/wiki/Arithmetic_logic_unit>

2. "Atmel 8-bit AVR® Instruction Set, Rev 0856." June 2010 <http://www.atmel.com/dyn/resources/prod_documents/doc0856.pdf>

3. "Central Processing Unit - Wikipedia, the Free Encyclopedia." 2 July 2010 <http://en.wikipedia.org/wiki/Central_processing_unit>

4. "Harvard Architecture - Wikipedia, the Free Encyclopedia." 26 June 2010 <http://en.wikipedia.org/wiki/Harvard_architecture>

5. "HowStuffWorks 'How Microprocessors Work.'" visited 6 July 2010 <http://www.howstuffworks.com/microprocessor.htm>

6. "Modified Harvard Architecture - Wikipedia, the Free Encyclopedia." 30 June 2010 <http://en.wikipedia.org/wiki/Modified_Harvard_architecture>

7. "Serial Peripheral Interface Bus - Wikipedia, the Free Encyclopedia." 27 June 2010 <http://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus>

8. "The I^2C-Bus Specification, Version 2.1." January 2000 <http://www.nxp.com/acrobat_download2/literature/9398/39340011.pdf>

9. "Universal Asynchronous Receiver/Transmitter - Wikipedia, the Free Encyclopedia." 24 June 2010 <http://en.wikipedia.org/wiki/Universal_asynchronous_receiver/transmitter>

10. "Von Neumann Architecture - Wikipedia, the Free Encyclopedia." 24 June 2010 <http://en.wikipedia.org/wiki/Von_Neumann_architecture>

11. "What is ALU? - A Word Definition From the Webopedia Computer Dictionary." visited 1 July 2010 <http://www.webopedia.com/TERM/A/ALU.html>

12. "What is CPU? - A Word Definition From the Webopedia Computer Dictionary." visited 1 July 2010 <http://www.webopedia.com/TERM/C/cpu.html>

13. "WikiAnswers - What are the Differences between CPU and ALU." visited 2 July 2010 <http://wiki.answers.com/Q/What_are_the_Differences_between_CPU_and_ALU>

