Faculty of Engineering, Computing & Creative Industries

Edinburgh Napier UNIVERSITY

Module : Computer Systems 1

Module Number : CSN07101

Computer System Basics

Student Study Materials

Version 1.1 Author : Frank Greig December 17th , 2013

• Frank Greig • School of Computing • Edinburgh Napier University • Edinburgh •

Faculty of Engineering, Computing & Creative Industries

Edinburgh Napier UNIVERSITY

Module : Computer Systems 1

Module Number : CSN07101

Architecture Basics

Student Study Material

Version 1.1 Author : Frank Greig December 17th , 2013

BASIC PROCESSOR SYSTEM ARCHITECTURE & OPERATION

INTRODUCTION

Computer based technologies are now embedded into a increasing range of consumer, industrial and commercial products. Obvious examples are Personal Computers (PC), mobile phones, televisions, washing machines, digital cameras, MP3 players and USB pen drives and the number of products that now rely on a processor based technologies is rapidly increasing. The automotive industry, for example, is currently experiencing a rapid growth in the deployment processor based technologies and experts anticipate that the average car is likely to contain 40-80 embedded microprocessors, controlling anything from the engine and braking systems to the comfort of the seating.

These notes introduce the basic building blocks, architecture and operations that are common to any processor based system. PCs are one, very common example, of processor based system technology, however, as this course progresses, alternative forms of processor based technologies will be introduced for example microcontrollers, digital signal processors, and graphics processors. These 'Embedded Processor Systems' don't look like PCs and they are usually located somewhere deep within a unit, say for example, inside a stereo system, under the dashboard of a car, or crammed into a mobile phone. The key point is that, although their external appearance may be quite different, the core elements and fundamental operating principles of all processor based systems is exactly the same. Hopefully, this fact will become crystal clear as this series of lectures progresses.

The operating principles of processor based systems are quite straightforward: its not 'rocket science' and there is nothing like a hardware 'brain' in the system. An important fact to keep in mind is this - no matter what it gets called, PC, PDA, Microcomputer, Intelligent XXXX, Smart XXXX or simply Embedded Processor System - if it has a processor at its core then the same basic hardware architecture exists and same basic operating principles apply. What differs significantly between systems, is the scale of their operations, for example, the speed of the system clock, how much memory the system supports and the types of peripheral devices that are connected.

BASIC SYSTEM ARCHITECTURE

Figure 1 presents a basic block diagram of a processor system showing the main functional elements of MEMORY (composed of ROM and RAM), INPUT-OUTPUT(I/O) and the PROCESSOR UNIT(PU). The connection between the system elements is made by the SYSTEM BUSES and the system timing is governed by the SYSTEM CLOCK..

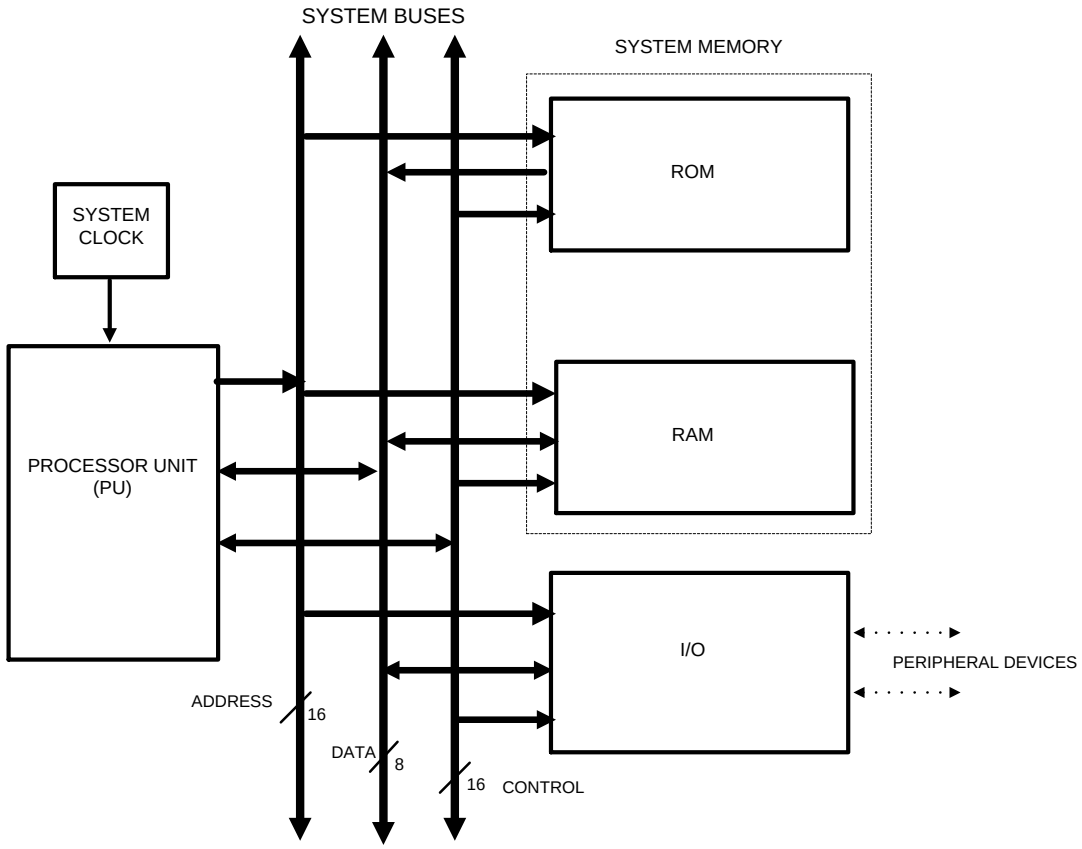


Figure 1. Basic Processor System Architecture

Before getting into the operational detail of computer systems it is appropriate to explore the functional behaviour of the individual system elements.

JARGON: The world of computing is full of jargon and hype and for the uninitiated this can be quite confusing and often discouraging. If you take the time to look closely you will soon see that very often complex sounding terminology is being used to describe very straightforward concepts. Don't be put off when facing terms like ALU, REIGISTERS, PORTS, MIPS,

Flash-ROM, SRAM, DRAM, DDRAM, CACHE, MNEMONIC etc, for the first time, these terms will become part of your own vocabulary in a very short time.

SYSTEM CLOCK

The system clock provides the basic timing reference for the system and all system activities, such as data transfers and the generation of bus signals are synchronised to the system clock.

A typical clock waveform is shown in Figure2. It is a graph of VOLTAGE (Y-axis) versus TIME (X-axis)

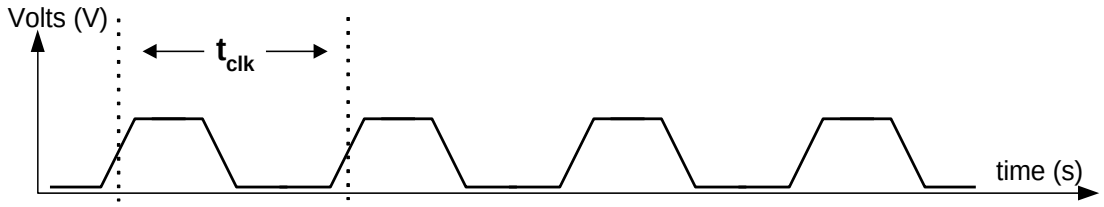


Figure 2 Clock Pulses

The clock signal is a good example of a binary signal (or digital signal) – it has only two possible values, 1 or 0, HIGH or LOW, ON or OFF. In this course, the actual voltage levels are unimportant, but they do illustrate that, deep inside the processor system electronics the 1's and 0's are actually voltage signals generated by semiconductor transistor switches opening and closing at very high frequencies.

The second important feature of the clock waveform is its frequency because it how fast the system operates. The relationship between time and frequency is simple :

frequency = 1/time or f = 1/t

where frequency is measure in Hertz (Hz) and time is in seconds (s).

As an example, a processor clock frequency of 100MHz produces 100 Million clock ticks per second and the interval between clock ticks is $1/100x10^6 = 10 \times 10^{-9}$ seconds, or 10 nanoseconds. Refer to Appendix A for a discussion on numerical notation.

The system clock is generated from a crystal oscillator (XTAL) that gives a highly stable frequency of oscillation. The XTAL frequency changes very little with changes in temperature, humidity and time (component ageing). Typical XTAL clock frequencies can range from 1MHz-250MHz and for practical reasons the XTAL frequency is often much lower than the actual processor system clock frequency. The processor system clock generator normally multiples the basic XTAL frequency to generate a much higher processor system clock. For example, a 2.5GHz system clock can be derived from a XTAL operating at 100MHz, using a x25 frequency multiplier.

In terms of system performance, a high clock frequency generally means faster data transfers and faster instruction processing. On this basis the clock frequency specification is often used to give a crude indication of 'processing power '. It should be noted however that the number of clock ticks required to execute an instructions does vary and complex instructions types, such as, multiplication and looping can take several clock cycles to execute. There is no one-to-one relationship between clock speed and system performance.

SYSTEM BUSES

In Figure 1 the system buses are shown as thick lines because they represent multiple signal connections. The (I_{16}) label attached to the ADDRESS Bus indicates that this bus has 16 signal lines and the (I_8) label on the DATA bus indicates that it has 8 signal connections. Physically the system buses are realised as Printed Circuit Board (PCB) tracks, or as a ribbon cable, or as multi-core cable.

Collectively, the system bus provides a uniform method for connecting the system components together. The great advantage of this approach is the flexibility. Bus compatible components can easily be added, or removed, from the system bus. For example, memory components and different types of I/O devices can be attached to the system bus to meet a variety of system application requirements.

Simply put a BUS is a group of signals that perform a common function and in the basic processor system shown in Figure 1 there are three buses - ADDRESS, DATA and CONTROL.

ADDRESS

The processor generates an address value to locate one item of data that is stored in memory (or at an input-output port). In normal circumstances the processor is the only device that generates addresses so the address bus is unidirectional . This is depicted in Figure 1 by the unidirectional arrow.

DATA

The data bus needs to be bi-directional because data needs to be written to and read from memory (or I/O) devices. Bus widths of 4, 8, 16 and 32 are common and the width of the data bus is used to categorise the processor, for example, an 8-bit processor has data bus width of 8 data lines and a 16-bit processor has 16 data lines and so on.

CONTROL

The control bus is made up of a group of signals that synchronise the transfer of addresses and data within the processor system. Unlike the address and data buses, where all the signal lines perform the same function, the control bus actually comprises a group of separate signals that collectively synchronise and control the transfer of data around the system. Common examples of processor system control signals are: memory read (MR), memory write (MW), RESET (RST) and INTERRUPT REQUEST (IRQ).

Collectively, the address, data and control buses are referred to as the **System Bus** and the system bus is used to interface the processor unit to both memory and input-output devices.

SYSTEM MEMORY

Memory in a processor system is used for two basic purposes. It stores program instructions and it stores the data generated when the program executes. Two fundamental memory types are required to accomplish program and data storage and they are called ROM and RAM respectively. Their basic organisation is shown in Figure 3 :

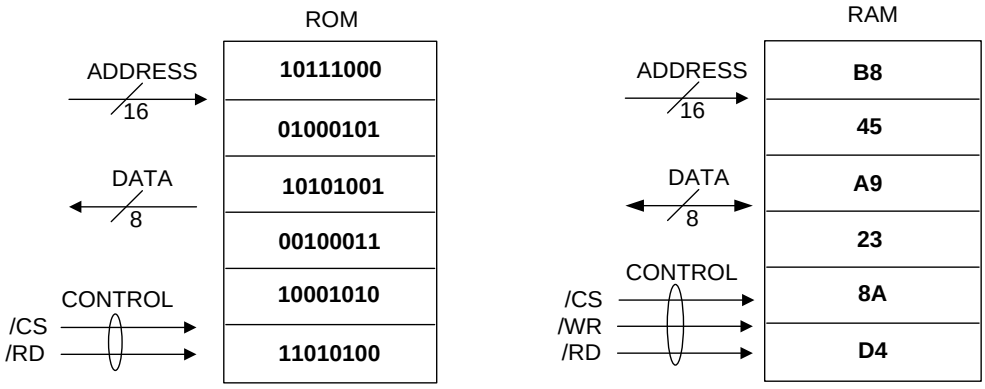


Figure 3. System Memory ROM and RAM

Memory devices function like a very large look-up-table where the input is provided by an address value which is used to 'look-up' data that is stored at the addressed memory location. The data stored at the addressed memory location is then output on the data bus. **The key point to remember is that every address input to a memory chip uniquely selects one storage location which in turn outputs a stored data value**.

READ ONLY MEMORY (ROM)

ROM stores program instructions and fixed data such as system constants and look-up tables. The contents of ROM must remain fixed when the power is removed, otherwise the system program is lost and the system does not function when power is re-applied. The term used to describe this type of memory is NON-VOLATILE and common types of ROM are MASKED ROM, PROM, UV-EPROM and FLASH-EPROM. Flash ROM now dominates the ROM memory market because it is electrically reprogrammable and devices such as MP3 players, USB pen-drives and digital cameras have Flash ROM at the core of their operation.

RANDOM ACCESS MEMORY (RAM)*

RAM stores the data that is generated during program execution. RAM is VOLATILE which means that when power is removed the stored data is lost. Data is read-from and written-to RAM therefore, its data pins are bi-directional. The processor control signals, memory-read and memory-write (RD and WR) determine the direction of data transfer. Note that the ROM chip has a read (RD) signal only.

There are two common types of RAM - SRAM and DRAM. DRAM is suited to large memory systems such as those found in PC's. The need-for-speed has led to the development of many DRAM derivatives – EDO, SDRAM, DDRAM, RAMDAC etc, and the various packaging options required for PC has introduced the SIMM and the DIMM. These technologies are described later.

*The term RAM is historical and is something on a misnomer. A more accurate term for RAM is Read/Write Memory (RWM) since both ROM and RAM provide random access to stored data.

The amount of data stored at a memory address is typically 8, 16, 32 or 64 bits - note that these are all power-of-2. Figure 3 shows a common memory organisation for ROM and RAM devices where each memory location stores 1-byte (8-bits) of data. The data stored in 8-bit can be represented 0-255 in decimal, 00000000-11111111 in binary and 00-FF in Hex. Although 8-bit data buses are very common in processor systems many newer systems, that demand more processing power, now use 16-bit, 32-bit or 64-bit architectures. For example, a Pentium type processor supports a 64-bit data bus.

Also notice in Figure 3 that the ROM shows data output only (read-only), and the RAM shows bi-directional data (i.e. both read and write)

Since the address inputs to a memory chip are binary then calculating the available number of memory locations in a memory chip is fairly straightforward. It is simply the number of address inputs raised to the power of 2.The table below shows the memory capacity versus the number of address inputs:

NUMBER of ADDRESS INPUTS	2^N	NUMBER of MEMORY LOCATIONS
1	2^1	2
4	2^4	16
8	2^8	256
10	2^{10}	1024 (1K)
16	2^{16} $(2^6) * (2^{10})$	65536 (64K)
20	2^{20}	1048576 (1M)
24	2^{24}	16 x 1048576 (16M)
30	2^{30} $(2^{10}) * (2^{20})$	1k x 1M = 1G (1 Giga)

The memory devices in Figure 3 are organised as 64kx 1byte or 64kB which may also be describes as a (64kx8) = 512kb (512kilobit) memory device. Not the difference B – meaning Byte and b – meaning bit.

PROCESSOR-MEMORY INTERFACE – BUS READ AND WRITE CYCLES

A data transfer from the processor to a memory (or I/O device) is called a WRITE cycle and a transfer from memory (or I/O device) is called a READ cycle. As is shown in Figure 4, the processor in conjunction with the system clock is responsible for sequencing and synchronising read and write bus operations. Specifically the processor provides the control signals, READ (/RD) and WRITE (/WR), which establish the direction of the data transfer. The (/) the in front of a signal name indicates that the signal is active low - the signal is active at logic 0 and it is disabled and when it is at logic 1. This signalling method is common in computer systems.

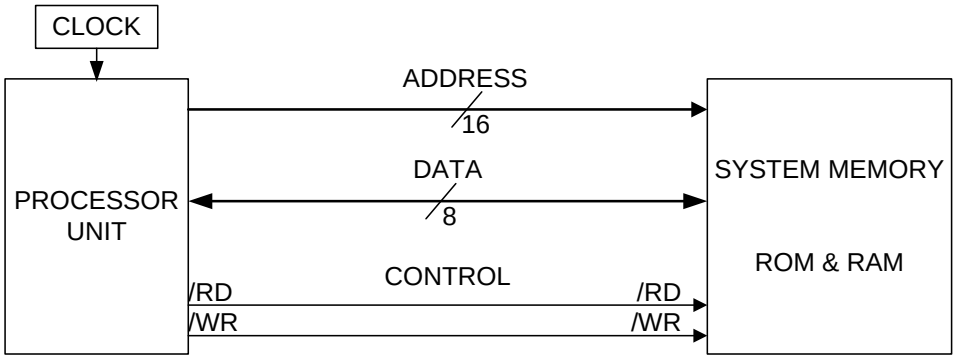


Figure 4 - A Basic Processor - Memory Interface and Data Transfer

A typical data transfer sequence for memory read and write transfers is as follows **:**

MEMORY READ CYCLES

A read cycle begins when the processor outputs an address value on to the address bus and asserts the memory (/RD) read signal. A short time later, allowing time for the memory access time to elapse, the memory device drives data (1's and 0's) on to the data bus. The processor then clocks the data from the data bus into an internal processor register.

MEMORY WRITE CYCLES

A write cycle begins when the processor outputs an address value on to the address bus and simultaneously outputs the data to be written on to the data bus. It then asserts the memory write signal and a short time later, again allowing time for signals to arrive at their destination and stabilise, the memory device stores the data at the addressed memory location.

At this point it is worth restating that the system clock synchronises every event during read and write data transfer operations. Bus read and write sequences are always synchronised to the system clock.

For simplicity the Chip Select (/CS) signal has been omitted at this stage. It is discussed later in the Memory and I/O Systems section.

INPUT – OUTPUT (I/O) INTERFACES AND PORTS

I/O interfaces provide a link between the processor system and peripheral devices. The processor communicates with I/O devices using the same read and write bus operations as for ROM and RAM data transfers. Figure 5 shows that the same bus address, bus data, and read and write signals are present, however, on the peripheral side of the interface things are quite different.

Communication between the processor system and external devices takes place through a hardware interface device. The I/O interface in Figure 5 matches the digital signal requirements of processor system bus to the external signal requirements of the peripheral device. Data transfers are implemented by reading from and writing to a series of internal registers within the interface device and data is transferred though **data ports (**as in a ships 'porthole'). Additionally, the format and synchronisation of data transfers is controlled via **control** and **status** registers.

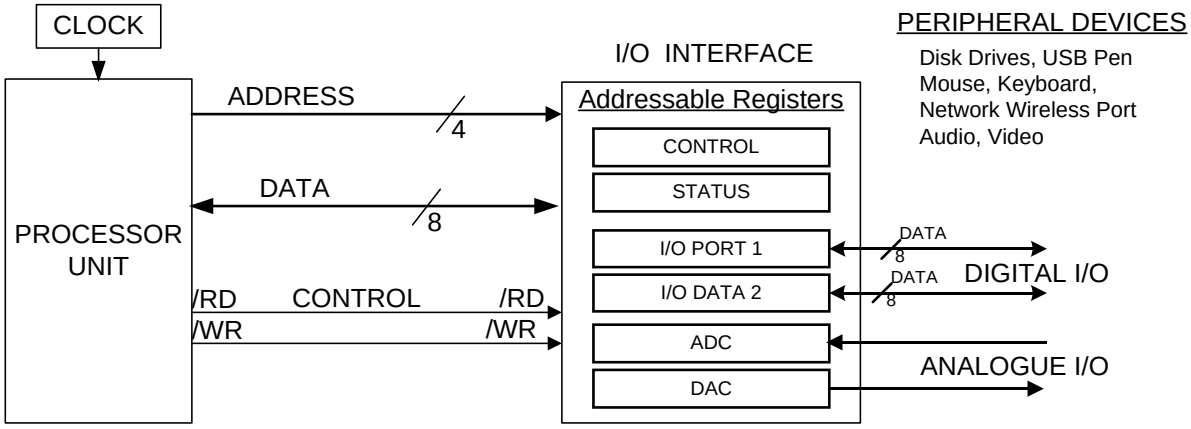


Figure 5 - Processor – I/O Interface.

I/O READ and WRITE CYCLES

This process is almost exactly the same as for memory read and write cycles. The notable difference is that there are far fewer address signals, only 4 in Figure 5, and they are used to address internal registers within the I/O device.

I/O INTERFACE TYPES

There is a vast array of peripheral devices that can be hooked-up to processor systems. Common PC system examples are, disk drives, sound cards and USB ports but in smaller embedded systems, for example, phones and MP3 players, the peripheral devices are more likely to be simple keypads, basic LCD Graphics displays, audio sounders or LEDs. Similarly, in measurement and instrumentation systems input sensors for measuring physical parameters such as temperature, flow, pressure, acceleration, direction and position are common peripheral devices.

Although the range of possible external devices is enormous their signal characteristics fall into just two categories: Analogue or Digital. A switch is obviously a digital device that produces either ON or OFF (1 or 0) signal, whereas a microphone is an analogue device that produces a signal output voltage that varies as audio signals are detected. Consequently, the switch requires a simple digital I/O interface and the microphone requires a more complex interface that converts the microphones analogue voltage into a stream of digital values

DIGITAL I/O

Since both sides of the interface are digital then the requirement for this interface are fairly straightforward – match the signal levels and the signal timing requirements on both sides of the interface.

ANALOGUE I/O

Analogue signal inputs must be converted into a digital form before they can be processed by the system and this is the function of the Analogue-to-Digital Converter (ADC) shown in Figure 5. Conversely, the function of the Digital-to-Analogue Converter (DAC) block is to convert digital signals (binary numbers) into analogue voltages

These interfaces are discussed in more detail in the I/O Interfaces section.

REGISTERS, FLAGS, BUFFERS AND PORTS

These terms describe storage elements within the processor and I/O interface devices. A key feature of registers and flags is that they are addressable and as such are directly accessible to system programmes.

REGISTER

A register is a named placeholder within a processor or I/O interface device. For example, the Control Register in Figure 5, is the register that sets the operating characteristics of the interface. Usually parameters such as data rates, data formats, signal directions and clock speeds can set (or tested) by writing (or reading) bits within the Control Register.

FLAG

A flag is a name single bit with a register. For example, the Status Register in an I/O interface device will often have a READY flag that indicates when the device is ready to send or receive data.

BUFFER REGISTER

A buffer register is a simple register that is often used to couple two system components together. For example in Figure 6 the function of the Memory Address Register (MAR) and the Memory Data Register (MDR) is to couple the internal processor buses to the external system address and data buses. In most cases buffer registers provide a hardware function and are 'hidden' from the system programmer.

PORT

The term **PORT*** is used to distinguish a normal memory location from an I/O location. Operationally an I/O interface can be viewed as enhanced memory device. Data written to an OUTPUT PORT is available at the output terminals as digital or analogue signal levels. Similarly, binary data read from an INPUT PORT represents the signal value at interface input terminals. Ports provide a structured mechanism for interfacing external devices to processor systems and they make it relatively simple to connect and communicate with external devices. I/O Interface devices for keyboards, printers, disks, mouse, modems, networks, graphics and audio systems are readily available from chip manufactures.

PROCESSOR UNIT (PU)

To avoid any confusion in the terminology, the terms Microprocessor Unit (MPU), Central Processor Unit (CPU) and Processor Unit (PU) mean the same thing. The term processor unit PU is used in these notes to describe the system component that is responsible for executing programs, which it achieves by implementing repeated instruction fetch-and-execute cycles. Figure 6 shows the internal architecture of a basic processor unit. It contain has three distinct elements:

- GENERAL PURPOSE REGISTERS (also called the REGISTER SET or Scratchpad registers).
- ARITHMETIC and LOGICAL UNIT (ALU);
- CONTROL UNIT.

Internally these functional blocks are linked together by a system of internal buses. The Memory Data Register(MDR) and the Memory Address Register(MAR), shown dotted in Figure 6, are non-addressable buffer registers that simply provide a link between the internal processor bus system and external system address and data buses.

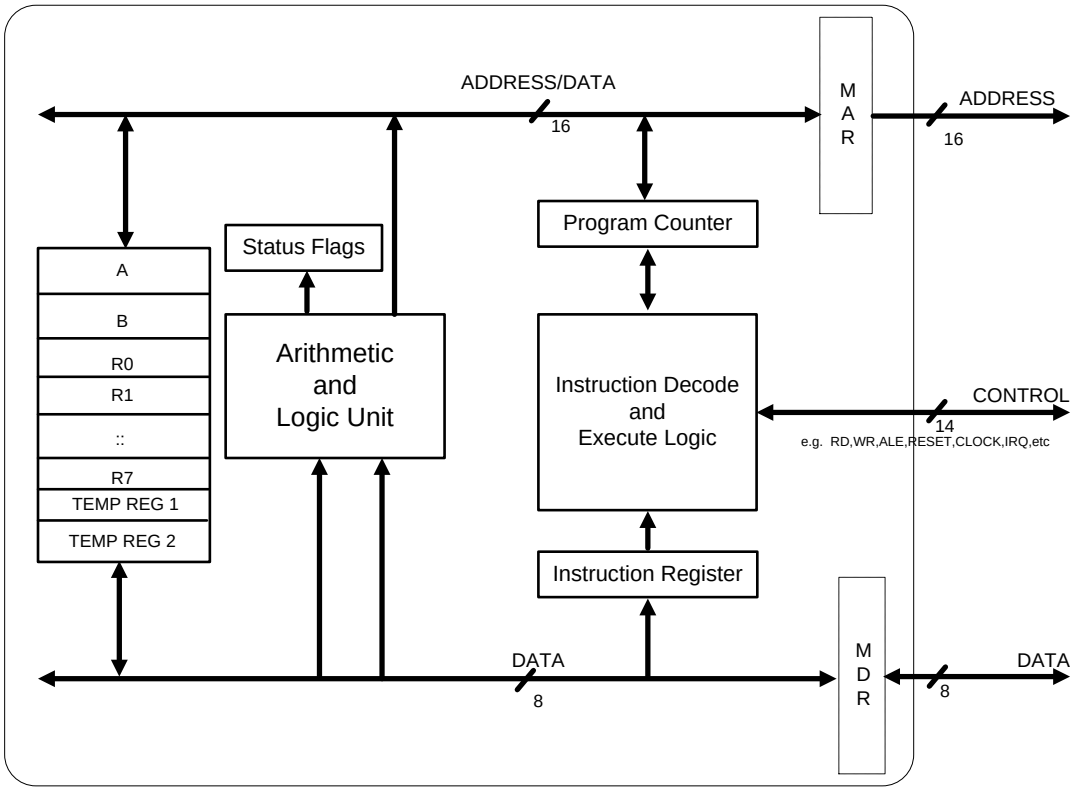


Figure 6. The Internal Architecture of a Simple Processor Unit

GENERAL PURPOSE REGISTERS

Processor resisters provide a limited area of storage locations that can be used to hold data while it is being processed. A register can be thought of as a memory location that is internal to the PU but it has a significant difference. Unlike a memory location which is identifiable by an address value, a register is identifiable by name, for example, R0, R1, R7, PSW etc. During program execution data is moved from external memory into the processor registers where it is held while it is being processed. Additionally, when the processor computes results they are temporarily stored in the processor registers before being written back to external memory when the processor is finished with then. Accessing data that is stored locally in the processor registers is much faster that than reading data from external memory devices, therefore, the important contribution made by the general purpose registers is that they significantly speed-up processing tasks.

ARITHMETIC AND LOGIC UNIT (ALU)

The **ALU** allows the processor to perform arithmetic and logical operations on data. Most processors provide the basic arithmetic operations of ADD, SUBTRACT, AND, OR, NOT and SHIFT. Instructions for MULTIPLY and DIVIDE are less common in 8-bit processors and when they are required, program subroutines have to written for these functions. More powerful 16-bit processors often have multiply and divide instructions as part of their instruction set but these are based on shift and add arithmetic and are very slow to execute. A typical arithmetic machine instruction would take the contents of two registers, ADD them together and then return the result to another register.

The ALU also has Status FLAGS associated with it - sometimes referred to as a Processor Status Word (PSW). Status flags are simply 1-bit storage elements that indicate the outcome of an arithmetic or logical operation. Typically, a ZERO flag becomes set when the outcome of an arithmetic operation is zero, say for example, after an instruction subtracts the contents of registers R0 and R1 and the result is 0.

Common status flags are ZERO(Z), CARRY(C), and NEGATIVE (N) and these are often used in conjunction with BRANCH and JUMP instructions to control program flow. Program loops (FOR and WHILE) and decisions (IF and SWITCH) are implemented using **conditional jump** instructions that rely on the processor status flags to make the decision.

CONTROL UNIT

The control unit is the most complex element of the processor architecture. It generates sequences of control signals that fetch and execute program instructions and it also provides the interface between the internal processing elements, the general purpose registers, the ALU, and external ROM, RAM and I/O devices. A processor may be able to execute hundreds, or even thousands, of different instructions with each instruction requiring a unique sequence of bus signals. It is the role of the control unit to generate these bus signal sequences during instruction execution in an orderly and precisely timed manner. To achieve this the control unit relies on the system clock to provide an accurate timing reference signal and all processor activities are synchronised the system clock ticks. This topic is examined in more detail when instruction fetch-execute cycles and program execution is discussed.

MACHINE INSTRUCTIONS AND THE PROCESSOR INSTRUCTION SET

A processor instruction set defines the range of instructions that a processor can execute. It is usually presented as a simple table that maps the instruction HEX code f alongside its **Mnemonic Code** (meaning an aid to memory).

For example:

HEX CODE	INSTRUCTION MNEMONIC	MEANING
	OPCODE OPERAND(S)	
76	ADD R0,R1	ADD the content of registers R0and R1 and store the result in register R0.
9A	INC R7	INCcrement (add 1) to some value
B4 1234	MOV R0, INDATA	MOVe data stored at address MYDATA (0x1234) to register R0.
80 000F	SJMP START	Short JUMP to the memory address START (0x000F).

A complete instruction is defined in two parts which can be summarised as follows -

- The operational part of the instruction, more formally called the **OPCODE (Operational Code),** defines the operation to be performed**, f**or example, MOV, ADD, JUMP INC (increment).

- The **OPERAND** refers to the location of the data to be operated on. In the example of an ADD instruction the operand identifies the location of the two numbers to be added and the eventual destination of the result. In the case of the MOV instruction the operand identifies the memory address or register name where the data is located and the destination of the result.

As a final point, note that there may be one or more instruction operands, for example, in the INC R0 instruction there is a single operand - R0, however, the source and destination of data is implicit in the instruction. The instruction simply takes the current value in the R0 register, moves it to the ALU, adds one to it and then stores the result back into the R0 register. Similarly, the move instruction MOV R0,INDATA has two operands. Operand 1 identifies the source of the data as the memory address labelled INDATA and Operand 2 identifies the register R0 as the destination of the data.

The identifiers (called labels in assembly language) START and INDATA are actually memory addresses - JUMP START, for example, could be JUMP 000F, assuming that the program starts at memory location 000F.

INSTRUCTION TYPES

When running applications such as Word or Excel on a PC, it is easy to lose sight of the fact that complex graphical operations, that take a fraction of a second to complete, are in fact simply the cumulative effect of executing many thousands of simple machine instructions. The simple fact is that processors are incapable of implementing complex tasks in a single operation. The instruction set defines exactly what the processor is capable of and it comprises a number of machine instructions that can be grouped together into the following instruction types:

DATA TRANSFER

These instructions transfer data between the processor and memory or the processor and input-output devices. Typical examples are: MOV -move, LDA - load, STA -store, IN – i/o input , OUT - i/o output .

ARITHMETIC & LOGICAL

Obviously these process data. Typical examples: ADD, SUB, MUL, DIV, AND, NOT, OR, LSHIFT, ROTATE.

FLOW CONTROL

These instructions interrupt the normal sequential flow of fetch-execute cycles and cause a jump, or branch to a new instruction address. These instructions implement program loops, decisions and subroutine calls.
Typical examples are :

Unconditional JUMP - JMP START*
Conditional JUMP - JZ – test Zero flag and jump if Z =1. (or JNZ test Z=0)
- JNC - test Carry flag and jump if = 0 (or JC test C= 1).
Subroutine call - CALL ADCSAMP * Subroutine return - RTS – return to the calling program.

MISCELLANEOUS

This group of instructions is very processor specific and typically includes:

Register Initialisation - At the start of a program it is normal to set internal system, control and integrated I/O registers to known states.
Interrupt Control - enable and disable interrupt requests.
Bit handling - SET and CLEAR individual bits on I/O ports.
No operation (NOP) - do nothing, simply increment the program counter.

PROGRAM EXECUTION AND THE INSTRUCTION FETCH-EXECUTE CYCLE

It has already been established that the processor unit is responsible for processing machine instructions and that it achieves this by repeatedly fetching machine instructions from memory (ROM) and executing them. Cumulatively, the process of repeated instruction processing is referred to as program execution, however, in order to get a clearer perspective on processor system operation it is essential that some grasp of low-level system operations is examined in detail. Debunking the myth of a 'computer brain' is a major objective here.

In order to bring these ideas together and to demonstrate how the hardware and software interact, the execution of a simple program is now analysed in monotonous detail. In the table below is a simple C program that adds the numbers, 5 and 7, and stores the result at memory address. It then loops back to repeat the addition. The actual object code (also called machine code) generated by the compiler is also shown.

SOURCE CODE – in C	OBJECT CODE - Machine Language

int main(void){
// Declare byte sized variables
unsigned char N1,N2,Result;
 	
while(1) // Loop forever
{
N1 = 5;
N2 = 7;
Result = N1 + N2;
}
}
	ADDRESS DATA MACHINE INSTRUCTION COMMENT
	
	C:0x000F 7F05 MOV R7,#0x05
	C:0x0011 7E07 MOV R6,#0x07
	C:0x0013 EF MOV A,R7
	C:0x0014 2E ADD A,R6
	C:0x0015 F508 MOV 0x08,A
	C:0x0017 80F6 SJMP 000F

The layout of the columns is typical of an Assembly Language program listing. The Address column identifies where the instruction is stored in memory; the Data column indicates the value stored in that memory location (in HEX); the instruction column shows, in mnemonic form, both the opcode and operand for the instruction, and finally there is an optional a comment.

INSTRUCTION FETCH-EXECUTE CYCLES

Although this code fragment does serve some useful purpose, its main function is to present a typical instruction sequence and to show how the basic instruction types are used. Data Transfer, Arithmetic and Branch/Jump instructions are all used in this short code fragment.

What is now to be considered is exactly how this sequence of program instructions is executed. Figure 7 shows the processor-memory interface with the object code for the program shown stored in consecutive memory addresses starting at the memory address 000F. Compare the listed sequence of hex codes with the object code listed in the previous table and note that some values stored in memory represent opcodes and others represent operands, for example, the opcode 7F is the instruction MOV for moving the data in the next memory location into register R7. In this case the value store in memory location 0x0010 (05) represents the operand for the MOV R7 instruction – the value to be moved into register R7.

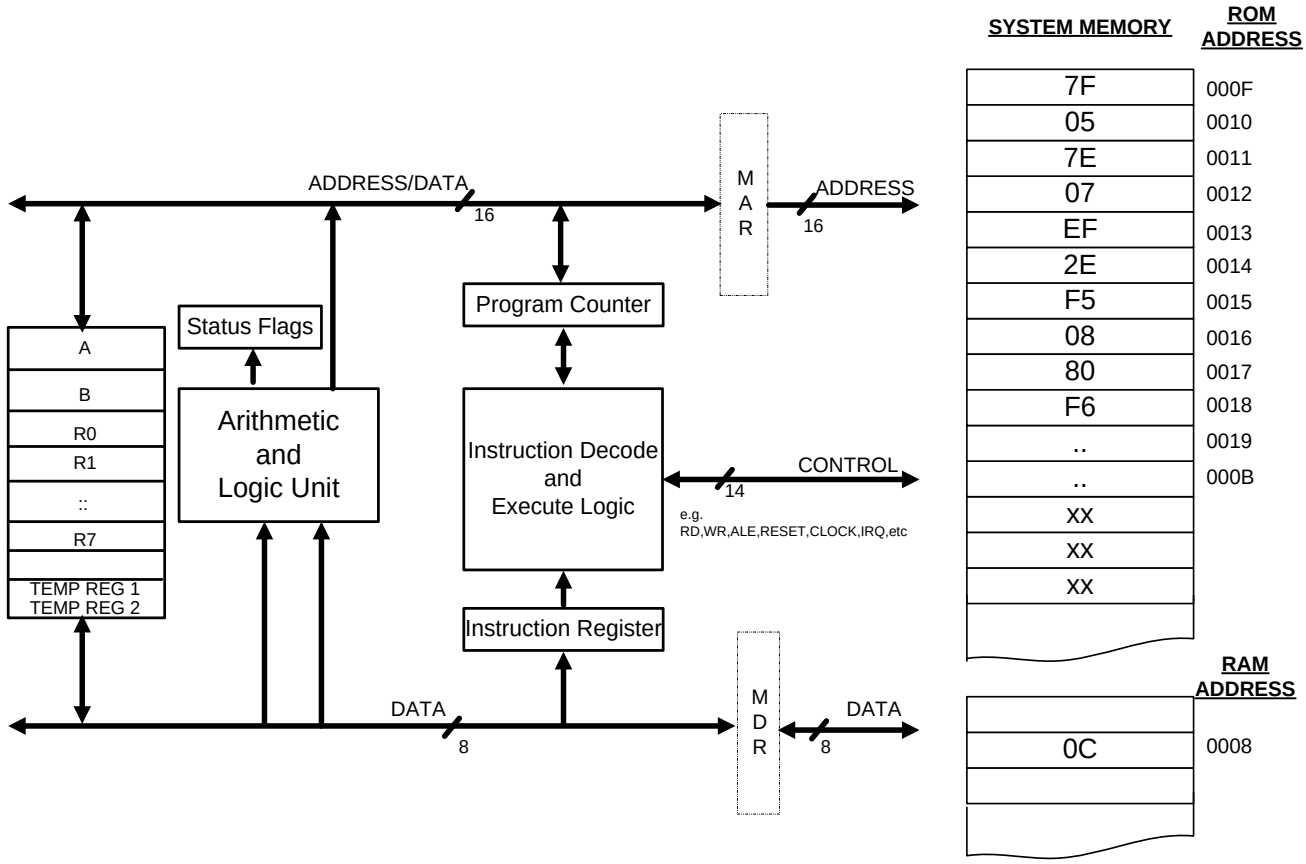


Figure 7. Program Execution and Instruction Fetch -Execute Cycles

In order to appreciate program execution as the cumulative effect of fetching and executing machine instructions a step-by-step account of the program execution is presented. This detailed analysis serves to demonstrate the repetitive nature of instruction FETCH - EXECUTE cycles and it should also illustrate that even complex programming tasks are implemented, at machine level, as the execution of a series of simple machine operations.

PROGRAM INITIALISATION

Assume that when power is first applied to the system, the processor is RESET to a known state. In this case assume that the Program Counter (PC) register is loaded with a reset value of 000F. As noted previously the processor system operates synchronously, therefore, all events occur in strict order and at regular clock tick intervals. In the description that follows it is the arrival of a clock pulse that causes the control unit to sequence from one event to the next:

INSTRUCTION - MOV R7, #05

The start of every instruction cycle (combined fetch and execute phases) begins with an instruction fetch cycle and ***since the INSTRUCTION FETCH SEQUENCE is the same for every instruction, it is presented here for the first instruction only and is assumed for all subsequent instructions***.

FETCH CYCLE

The control unit transfers the PC register contents (000F) to the address bus by copying the PC value to the memory address register MAR. On the next clock tick, the READ signal is asserted. At this point the addressed memory location is accessing the stored data and after the ***access time*** has elapsed the addressed data (7F) appears on the data bus. This data value is then ***latched*** into the Instruction Register (IR) - this completes the instruction fetch cycle.
The current PC value is now incremented to the next memory location (0010).

EXECUTE CYCLE - MOV R7, #05

In response to the IR value of 7F the control unit executes the following sequence:

The control unit transfers the PC register contents (0010) to the address bus and asserts the READ signal. Again the addressed memory location responds by outputting its data, the contents of location 0010 (05) onto the data bus. The control unit then transfers this data bus into the R7 register.

The current PC value is now incremented to the next memory location in sequence (0011).

The instruction fetch-execute cycle for MOV R&,#05 is complete. Now fetch the next instruction and repeat

INSTRUCTION - MOV R6,#07
FETCH CYCLE - as above but this time memory location 0011 is addressed and opcode (7E) is read into the IR register.
EXECUTE CYCLE - In response to the IR value of 7E the control unit executes the sequence:

- Transfer the PC register contents (0012) to the address bus and the READ signal is asserted.
- Memory responds by putting the contents of address location 0012 (07) onto the data bus.
- The control unit transfers the data bus value to the R6 register and the current PC value is incremented to the next memory location.

INSTRUCTION - MOV A,R7
FETCH CYCLE - as above but this time address 0x0013 is used and 0xEF is read into the IR register.
EXECUTE CYCLE - In response the control unit executes a sequence of operations which are completely internal to the processor. The R7 register contents (05) are transferred to the processor accumulator register (A).

INSTRUCTION - ADD A,R6
FETCH CYCLE - as above but this time address 0x0014 is used and 0x2E is read into the IR register.
EXECUTE CYCLE - In response the control unit executes a sequence of operations which again are completely internal to the processor:

- The R6 register contents (07) are transferred to the ALU input, the A register contents (05) are also transferred to the ALU input, an ALU ADD signal is generated.
- Finally the ALU output data (12) (0x0C in hex) is transferred to the A register.

INSTRUCTION - MOV 0x08,A
FETCH CYCLE - as above but this time address 0x 0015 is used and 0xF5 is read into the IR register.
EXECUTE CYCLE - In response the control unit executes the following sequence:

- The control unit transfers the PC register contents (0016) to the address bus and the READ signal is asserted.
- Memory responds by putting the contents of address location 0016 (08) onto the data bus and the control unit transfers this value to the MAR register low byte.
- The control unit then writes (00) to the high byte of the MAR. The current PC value is incremented to 0017.
- The control unit then transfers the A register contents to the MDR register, and then asserts the WRITE signal. The data is now written to memory location addressed by the MAR (0008).

INSTRUCTION - SJMP 000F
FETCH CYCLE - as above but this time address 0x 0017 is used and 0x80 is read into the IR register.
EXECUTE CYCLE - In response the control unit executes the following sequence:

- The control unit transfers the PC register contents (0018) to the address bus and the READ signal is asserted. Memory responds by putting the contents of address location 0018 (F6) onto the data bus and the PU transfers it into one of the temporary registers, say TEMP REG1 in Figure 7.
- The control unit increments the PC to 0x0019 and then adds this value to the contents of TEMP REG1 and returns the result to the PC register.

The processor has performed the following 2's complement addition – 0x0019 + 0x00F6 = 0x000F or in decimal 25 + (-10) = 15. The overall result is that PC value is now 0x000F and the next instruction is fetched from memory location 0x000F - the program has executed an unconditional jump instruction.

The sequence of instructions, from address 000F to 0019 repeats, and continues to repeat forever.
The program is 'STUCK' in an infinite loop. From a user perspective this would be observed as a system CRASH and only a system RESET or POWER-OFF is going to restore the system to its RESET state.

OBSERVATIONS ON MACHINE LEVEL OPERATIONS

Having successfully, if somewhat tediously, negotiated the execution of a simple machine code program it is worth drawing together the major points raised and relating them to processor systems in general :

- Programming tasks, however complex, are implemented at machine level as sequences of simple instructions, MOV, ADD, JMP etc. Comparisons with, or analogies to 'computer brains' are stupid and completely unfounded. A processor based system is a **'no-brainer'!**
- **The length of instructions is variable -** Instructions can occupy 1,2 or 3 bytes of memory.
- **Instruction execution times are variable -** Instructions take a different number of clock cycles to execute. The branch instruction takes many more steps and many more machine cycles than a simple register increment instruction for example.
- **The processor executes simple machine instructions in a few clock cycles -** Assuming that on average each instruction takes 2 clock cycles to execute then these five instructions take around 10 clock cycles to execute. If the clock frequency is 100MHz (10ns per clock pulse) then it takes 100ns to execute this section of code. That works-out as 10 Million-Instructions-Per-Second (MIPS).

- **Complex tasks appear to be performed instantaneously** – This is due to the fact that the processor executes in MIPS. Of course it takes time for a processor to execute programming tasks but our response times are so slow compared with the time that it takes to execute useful computer tasks. Computer system responses can in most cases appear to occur instantaneously. Even today with high-power processors some tasks, notably high-resolution graphics operations, can still appear painfully slow or are rough-around-the-edges.

PROGRAMMING LANGUAGES

Assembly Language Programming is lowest level of system programming and it has the great advantage that the system programmer has complete control over all the system resources. The registers used, the memory allocated and the instructions used are all decided by the programmer. In the hands of an experience assembly language programmer this can result in code that is extremely efficient(fast) and compact (uses the minimum amount of memory).

The downside of this freedom of resources is that assembly language programming is difficult to write, much more prone to error and much less productive than programming in high-level languages. It is much more productive to use languages such as C, C#, Java, Basic Pascal or Perl for applications programming. The benefit of using C or Java (High Level Languages) is that low-level issues are hidden from the programmer. Decisions such as which registers or instructions to use are not under the control of the programmer, these issues are resolved by the compiler. More importantly, HLLs offer the significant advantage that program code becomes, in theory anyway, machine/processor independent making HLL code is PORTABLE. In theory, a C program written for a PC should be ported directly to a Mac PC: In practice, however, this is rarely the case, some adjustment is always necessary.

Tools such as compilers are readily available to translate the HLL language syntax, FOR, DO, WHILE, IF THEN UNTIL etc, into low-level machine instructions and the availability of a compiler greatly simplifies the task of program development and improves productivity. Note, however, that the reliability and efficiency of the machine code running on the system is then almost entirely dependent on the quality of the compiler.

To bring some of these ideas together consider what low-level machine operations are required to implement the JAVA statement :

System.out.println (" Current Sample Value >>>> " (ADC_Sample/255) * 5.0) ;

At a global level the message - Current Sample Value >>>> 1.434 would appear on the system monitor but what are the low-level operations required to implement this task :

- Compute the value (ADC_Sample/255) * 5.0 – which results in a real number, say 1.4.
- Convert this value to an ASCII string.
- Append this string to the string Current Sample Value >>>> .
- Write the complete string into video memory so that it can be displayed on the system monitor.

Obviously this routine HLL operation is implemented by many, perhaps thousands, of simple machine level instructions. The advantage of using HLLs is obvious.

SOME FINAL THOUGHTS

- You will meet numerous programming languages (JAVA, C, C# etc) and applications (WORD, EXCEL etc) during your course. Try to keep this fact in mind when you sit down at a computer:

  No matter what you think is going on in front of you, the reality is - inside the box there is a processor, and it is systematically fetching and executing simple instructions such as MOV, ADD, JUMP and it is doing this very very fast, Millions-of Instructions Per second (MIPs).

  Its all too easy to lose sight of this fact when you enter the world of Windows, Java, Visual Programming, Object Models, Artificial Intelligence etc, etc.

- The function performed by a processor based system is defined by a program. A program, in the form of machine instructions, stored in the system memory defines the operation of the system. This concept, generally known as ***stored-program operation***, highlights the fundamental attraction of processor based systems and suggests a major reason why processor based systems are being used in an ever increasing number of applications - **FLEXIBILITY**. Changing the system operation by modifying the system software allows manufacturers to make changes to or enhance their existing products and to develop new products more quickly. Time-to-market is a key business consideration.

- The layman's view of a computer as a desktop box (or a tower) with monitor, keyboard and mouse is a very restricted view of what real computer systems actually are. PCs are only the tip-of-the-iceberg, processor based technologies are now embedded into a vast range of devices such as televisions, washing machines, toasters, toys and iPods, mobile phones. Additionally, there is very often more than one processor in the device: a mobile phone may use upto three processor cores; a PC definitely makes uses of several processor cores in addition to the main normal Pentium or Dual/Quad core processor. What do they do?

- IO interfaces and the availability of a wide range of sensing devices dramatically widens the range of computer system applications. New sensors and actuators make it possible to monitor and control many physical parameters such as temperature, pressure and flow, distance, acceleration, touch, proximity, gestures etc using processor based technology. The basics of IO interfaces are discussed in the next section.

- There is currently, much ongoing research and development into a wider range of applications for embedded processor based systems. Applications, such as, distributed sensing systems, smart buildings, mobile computing, wearable computing, telemedicine, grid energy metering are all currently hot topics. The list of embedded systems applications is growing rapidly and this indicates that there is a bright future for software and application developers with an understanding of low-level machine hardware operations.

Faculty of Engineering, Computing & Creative Industries

Edinburgh Napier UNIVERSITY

Module : Computer Systems 1

Module Number : CSN07101

MEMORY & INPUT OUTPUT INTERFACES

Student Study Material

Version 1.1 Author : Frank Greig December 17th , 2013

MEMORY & INPUT OUTPUT SYSTEMS

INTRODUCTION

The simplified computer system model introduced previously identified and described the main elements found in any basic processor system. In this section, processor system hardware is discussed in more detail and Figure 8 presents a more detailed view of the processor system showing the MEMORY and INPUT/OUTPUT subsystems with their associated control signals in place.

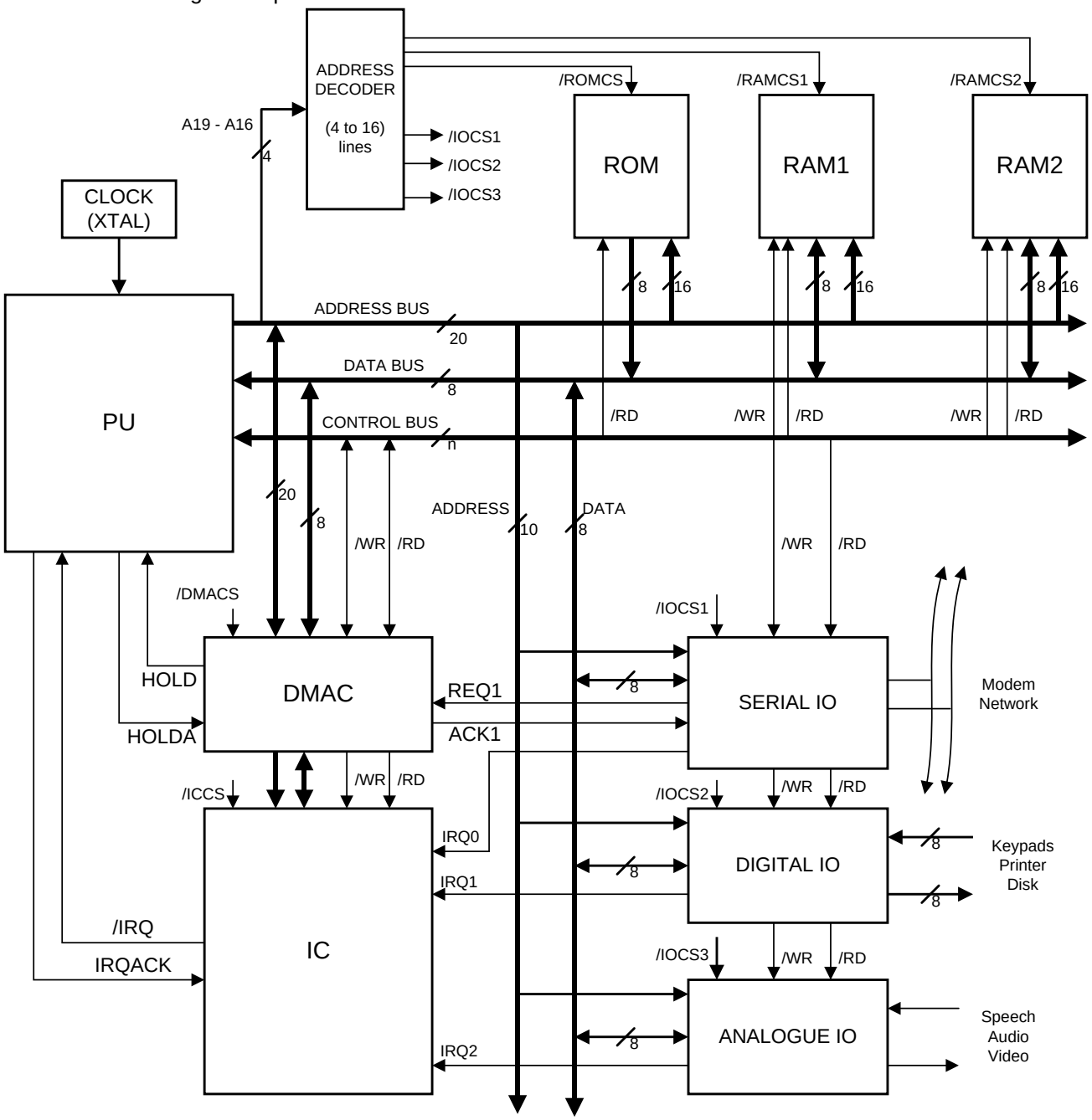


Figure 8. 8-bit Processor System showing Memory and I/O Subsystems

The memory sub-system is comprises a mixture of ROM and RAM chips and the Input-Output sub-system now shows a number of Digital and Analogue I/O interfaces. Also shown in Figure 8 is the detailed connection of the device Chip Select (/CS) control signals that when in conjunction with the ADDRESS DECODER chip ensures that the individual memory and I/O devices are uniquely addressed. This important mechanism avoids data bus contention issues – this is discussed in thee next section. To avoid cluttering the diagram the /IOCSx signal connections are broken but consider them as being physically connected in just the same manner as the ROM and RAM chip selects.

Figure 8 also shows two I/O Controller devices, an Interrupt Controller (IC) and a Direct Memory Access Controller (DMAC).These devices provide efficient mechanisms for implementing I/O systems and are introduced in a later section. High-speed data transfers between system memory and disk drives, or between the processor system and the network use the DMAC and IC to manage the data transfer.

DEVICE ADDRESSING - ADDRESS DECODING AND CHIP SELECTS

PC memory systems require large amounts of RAM memory, typically 0.5-1GB, and a number of I/O interfaces to support a broad range of peripherals, such as, disks, networks, USB, mouse, keyboard, audio and video. As Figure 8 shows all these devices share the common data bus and many devices must share the same address, data and control pins. If this common bus connection arrangement is to work then certain operating conditions must be observed :

- **Only one device should be able to drive the data bus at any time.** All devices in a bus connected system share a common data bus and if more than one device attempts to write data to the bus at the same time a collision of 1's and 0's occurs. This situation is known as a **'bus contention'** and it results in the corruption of data on the bus. Bus contentions may also result in damage to the devices that are attempting to simultaneously drive the bus. Contention issues are avoided by ensuring that when a device is not addressed it is effectively disconnected from the data bus. The is the function of the device chip-select (/CS) control signal which achieve connect and disconnect from the data bus using TRI-STATE operation. Simply put this means that all non-selected devices are effectively powered-down and disconnected from the data bus. It should also be understood that it is the function of the ADDDRESS DECODER block, in Figure 8, to decode the address bus signals such that only one chip select signal is active at any time ensures thereby ensuring that bus contention is avoided.

- **Every device connected to the bus system must be uniquely addressed.** The purpose of the address decoding circuit in Figure 8 is to ensure that only one device is active at any time. For any combination of address inputs only one device chip select output becomes active and only one memory or I/O device is enabled. The address decoding circuit effectively divides the available address space into blocks so that each device occupies a specific range of addresses.

Device addressing in Figure 8 has now become a little more complicated but it does demonstrate more clearly how the addressing system actually works. When the processor puts an address on the address bus the higher order address signals (A19 – A16) are used to enable a single device, the lower-order address (A15 – A0) signals are used to select the specific memory location or I/O register within the device. In reality, at the instant when the data transfer takes place only the processor and the select memory location are connected to the data bus – all other devices are switched-off.

MEMORY AND I/O DATA TRANSFERS

The previous discussion on data transfers was somewhat simplified so it is re-visited here this time with the inclusion of the system clock, device control signals and address decoding logic is taken into consideration. The intention is to generate a deeper understanding of processor system hardware operations and the data transfer process.

READING DATA

To read the data from a device the processor outputs an address value on the address bus and asserts the memory read signal(/RD). The address decoding circuit responds to this by activating one of it's chip select outputs (/CSx) and a short time later, allowing for memory access time, the selected memory or I/O device drives a data on to the data bus. The processor then clocks the data from the data bus into an internal register to complete the read cycle. Consider as an example reading data from the Digital I/O port. The sequence of events would be something like : the processor outputs the address of the digital I/O port and asserts the memory read signal(/RD); the address decoding circuit responds by activating /IOCS2, and a short time later, allowing for the device access time, the digital I/O device drives a data on to the data bus; finally the processor clocks the data from the data bus into an internal register to complete the read cycle.

WRITING DATA

Writing data from the processor to a memory or I/O device follows a similar pattern. The processor outputs an address value on the address bus and at the same time also outputs the data on the data bus. The address decoding circuit responds to this by activating one of it's chip select outputs (/CSx) so that a device and a unique location are now addressed. The processor then asserts the write signal (/WR) to begin writing the data into the device. A short time later, allowing time for the address, data and control signal to stabilise and the data to be written into the selected device, the processor removes the write signal to complete the write cycle.

In summary, the device chip select /CS signals are derived from the high-order address bus signals and are used to uniquely identify the device that is to be used for data transfer while the low-order address signals are used to identify the single memory location or I/O device register that is to be used in the data transfer. The read (/RD) and write (/WR) bus control signals are used establish the direction of data transfer between the processor, memory and I/O devices. It is again worth restating that the sequential activities described for read and write data transfers are synchronised to the system clock

Note again that all instruction *processing activity and data transfer operations are synchronised to the system clock.*

MEMORY TECHNOLOGY – ROM AND RAM

ROM and RAM are generic terms for semiconductor memory but these can be sub-divided into other types such as UVROM and Flash ROMs, and SRAM and DRAM RAMs. The discussion here is limited to the basic characteristics of memory and their applications but it is useful to establish the criteria that can be used for making comparisons of different memory types. Current memory technologies are limited in what they can achieved because they must operate within the following constraints:

- Memory cells occupy a finite area of silicon. They have a 'cell-size' that limits the number of memory cells that can be fitted into a single chip. The term **bit-density** relates to cell-size and is used as a parameter for comparing memory devices. It indicates how many cells can be built into a given area on the chip.
- Larger memory cells and faster memory cells consume more power and excessive power consumption means that more heat is generated – remember heat kills semiconductors. This also places a limit on the permissible number of cells on a chip.
- High-speed memory operation is clearly important for interfacing with fast processor systems. **Memory Access Time** is the timing parameter that defines how fast the device works. It is the time elapsed from addressing a memory location to the time when data is made available at the data outputs. A 50ns memory is specified as having 50ns access time. Also note that faster memory is more expensive.
- **Endurance** is the number time the device can be re-programmed – 100,000 cycles is typical for Flash ROMs.
- **Data retention** defines the time period that bits can be reliably stored. The device may start to lose its stored program after this period – 10years is typical for EPROM and Flash ROM types.

ROM

Read Only Memory is used to store permanent data. The contents of a ROM are fixed when it is programmed and in normal operation the contents of ROM cannot, and should not, be altered. The term ***non-volatile*** is applied to ROM to indicate that ROM retains it's stored data when the power is removed. ROMs are used for storing such things as program code, character generators and look-up tables. The following are the more common types of ROM:

MASK PROGRAMMABLE – ROM

To generate a Masked ROM a customer supplies the memory chip manufacturer with the binary data (1's and 0's) for programming into the chip. This is usually supplied as a binary file (HEX File) to the manufacturer who then produces a MASK - a pattern of the physical connections required to store the data in the device. This process is irreversible and is only viable for large scale production with stable systems. Mask-ROM becomes cost-effective when large numbers of devices are to be manufactured – think toys, set-top-boxes, automotive applications, games console etc.

UV ERASABLE PROGRAMMABLE READ ONLY MEMORY - UVEPROM

UVEPROM was the forerunner to Flash-EPROM and can now be considered as obsolete.

FLASH EPROM

Flash memory devices are now the preferred technology for non-volatile storage applications.

A flash memory cell stores a 1's or 0's as the presence or absence of a minute stored electrical charge but the key feature of Flash is that it is erased and reprogrammed electrically. This is very definite system advantage because devices can be erased and reprogrammed in situ.

Flash memory devices are structured a little like disk drives where the data is stored in SECTORS/BLOCKS. A typical 128MByte flash ROM would have, for example 16 x 8MB blocks. The use of sectors allows different areas of the memory chip to be programmed and erased separately, for example an MP3 track can be erased from one sector on leaving the system programme code, which resides in another sector, completely unaffected.

The explosive growth in the sales of media devices, such as, USB pen drives, MP3 players and digital cameras, is fuelled by the availability of low-cost of Flash Memory. The ability to erase and re-programme Flash devices electrically is a major convenience that opens-up a wealth of system possibilities, for example, remote system software upgrades and diagnostics can be implemented over the Internet, at minimal cost and maximum productivity.

RAM

Random Access Memory (RAM), or more accurately Read Write Memory (RWM), is used to store data. The data stored in a RAM must change each time a new value is written, therefore, the storage mechanism must be able to alter the stored '1' or '0' bits in each memory cell. RAM is ***volatile*** which means that data is lost when the power is removed. The two main RAM technologies are Static RAM (SRAM) and Dynamic RAM (DRAM).

STATIC RAM MEMORY - SRAM

SRAM memory cells use a flip-flop circuit to store a single bit. This circuit requires 4 transistors, therefore, the cell - size for SRAM is large and the bit density is low when compared with ROM or DRAM cell types. The power consumption of SRAM is also much greater (about 4 times) that of DRAM. These disadvantages are easily outweighed by the significant advantage that SRAM has - fast memory access times, which can be as low a 5ns.

The large cell structure and fast access time of SRAM makes it suitable for applications that require high-speed memory accesses - CACHE memory and I/O memory buffers are notable examples..

DYNAMIC MOS MEMORY - DRAM

The dynamic cell uses single MOS transistor and very small capacitor to store a bit. The storage mechanism relies on the fact that a capacitor stores charge. When a memory cell is selected for writing, the cell transistor is switched ON and the storage capacitor is charged-up to store a logic 1; If a logic 0 is to be stored the storage capacitor is discharged. DRAM has slower access times than SRAM, operating at with around 70-100ns access times. Another limitation of DRAM is that the storage capacitors are not perfect and the stored charge needs to be toped-up periodically. This operation is called a refresh cycle and is required every 2-4ms.

The small cell structure of DRAM – approximately ¼ of SRAM cell size- and lower power consumption of DRAM makes it suitable for large memory systems.

PC SYSTEM RAM

PC system memory uses DRAM devices that are supplied as Dual-In-line Memory Modules (DIMMs). The DIMM shown in Figure 9 shows an array of memory chips that stores 256MB of data.

As a more convenient/consistent of specifying memory access time, DIMM modules use the concept of memory bandwidth (data rate in Mbps) to indicate their speed of operation. In Figure 9 the DIMM supplies data to the processor at the rate 2100MBps and this a calculated as follows:

PC2-4200 - is 8B x 2 x 266MHz = 4200Mbps (64bit data bus is 8Bytes x 2 for double data rate x bus speed).

Bandwidth is consistent with network terminology and is perhaps a more relevant way to compare memory performance.

Figure 9 A DDR3 - DIMM

DDRAM - Double Data Rate Random Access Memory

DRAM is used in PC memory systems because a PC requires large amounts of memory and DRAM offers the best compromise between bit-density, data transfer rate, power consumption and cost. At present Double Data Rate RAM (DDRAM) is used because it provides data at the fastest rate. DDRAM operates with bus speeds of 200 - 400MHz and Double Data Rate RAM allows data to be delivered to the processor on both edges of the system clock thus doubling the available data transfer rate. For example a DDRAM system running with a 400MHz bus clock gets data at double the speed - 800MHz.

DDR3

The sole purpose of this DDRAM evolution is to supply data hungry processors with data at an ever faster rate and since the evolutionary process continues DDRAM has now progressed from DDR2 to DDR3 which offers data rates of up to 12.8Gbps.

CACHE MEMORY

There are numerous specialist memory devices used in processor based systems but cache memory is now so common that it deserves a separate mention. High-performance processor systems require large amounts of fast memory RAM which, from our previous discussion, is a requirement that is in direct conflict with the available SRAM and DRAM technologies. Large amounts of memory implies the use of DRAM because it has the highest bit-density, while high-performance implies the use of fast SRAM because it has faster access times than DRAM.. Remembering also that SRAM consumes more power than SRAM it is prohibited for large memory arrays.

CACHE MEMORY provides an ingenious solution to the problem of providing, at least as far as the processor sees it, very fast SRAM memory system without the overheads of increased power consumption and system costs. Cache Memory, as shown in Figure 10, provides a very efficient compromise that gives a vastly improved performance with little added cost. In Figure 10 a small amount (512k - 2M) of very-high speed SRAM placed between the processor and main memory. This is the memory CACHE and the key its success lies in an aspect of program operations called 'Locality of Reference'.
In general, program execution involves highly clustered operations which means that there is a high probability that the address used for the next memory access (instruction or data) is in close proximity to the current one. Consider the three basic program constructs :

- SEQUENCE – the next instruction is located in the next memory location.
- DECISION - the next instruction is located either in the next memory location or the program counter is loaded with a new address that may be within, at most a few thousand memory locations.
- REPITITION - same as DECISION.

In the case of data, it is normally processed sequentially, the next item to be processed is frequently the next one in a list, array or table and is, therefore, local to the current item.

Cache memory exploits locality by pre-fetching instructions and data from system memory into the cache memory thus ensuring that the processor fetches instructions and data from fast SRAM cache memory and rarely resorts to fetching instructions and data slower mains system memory (DDRAM).

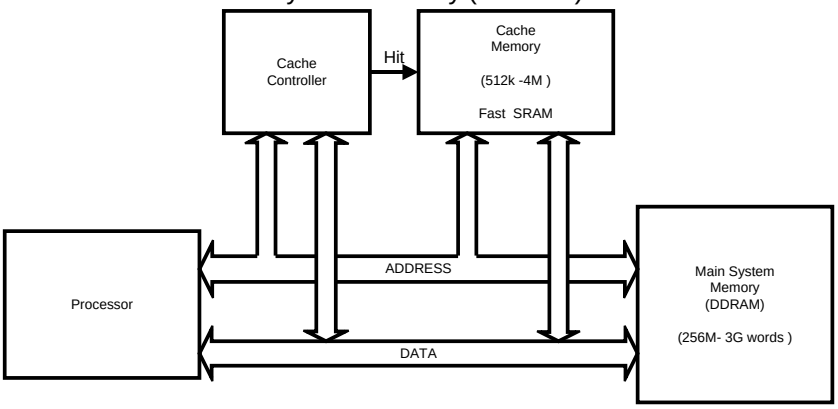


Figure 10 A basic Cache Memory interface.

The cache controller in Figure 10 ensures that the instructions and data currently in use by the processor are pre-loaded into the cache. It monitors the current address values and predicts the blocks of main memory that are about to be accessed, it then pre-fetches them into the cache memory. A successful prediction by the cache controller results in a data or instruction fetch from the fast cache memory and is called a **'hit'.** The hit signal shown in Figure 10 causes that the cache memory to supply data to the processor. If instruction or data is not currently in the cache a cache **'miss'** has occurred and the data is fetched from slower system memory.
The performance of a cache system is measured by the **'hit ratio'**, which is a measure of how many times the processor locates the source of data in the cache (a fast access) versus the number of times it finds the data in main memory (a cache miss and a slow access). Hit ratios can be as high as 98%. The policy used for making cache predictions is based on a knowledge of the memory addresses accessed at present, an awareness of the instructions that have been executed most recently and a prediction of those instructions that are most likely to be executed in the near future. Obviously, embedding this level of functionality into the cache controller makes it a somewhat complex device. In Figure 10, purely for reasons of clarity, the cache controller is shown as a separate block. In reality, the cache controller and memory is often integrated into the processor chip.

LEVEL 1 AND LEVEL 2 CACHE

Cache systems are often split into two or even three levels. Level 1 cache is integrated into the processor and is small (around 32kB) and is very fast. Level 2 cache, as shown in Figure 10, is external to the processor. This arrangement further improves the hit ratio which can be 90-95% in PC systems.

HOW MUCH CACHE?

It is tempting to think that the larger the cache the higher the hit ratio, however, cache sizes greater than 5-10% of the available system memory provide very little improvement in performance. The reason for this is that it is more difficult to manage the data in larger caches and it takes longer to determine when cache hits occur.

I/O INTERFACES AND EXTERNAL DEVICES

Computer systems are now required to interface to a vast range of I/O devices. Common examples are disk drives, display, printer, mouse and keyboard, however some other examples that are being used increasingly are microphones, cameras, gamepads, virtual reality headsets, Wi motion sensors. Although at first glance these devices would appear to be quite different, a closer examination quickly reveals that they fall into just two basic categories – they are either DIGITAL or ANALOGUE interfaces.

DIGITAL I/O

Digital interfaces are relatively simple to implement because the signals on both sides of the interface are in the same binary form. The processor side of the interface always uses parallel data transfers so that, in the simple case, where the required data is also in parallel format the main role of the interface is to match the signal levels. Many devices provide data in serial stream of bits and so this type of interface has a secondary function – it must translate the parallel data into serial data and vice versa. These two basic forms of digital data transmission are termed - PARALLEL and SERIAL.

PARALLEL INTERFACE

Parallel I/O data transfer is fast, requiring 1 clock pulse to transfer N-bits of data. On the downside however, it requires 1 signal line (copper cable, connector pin, pcb track) per signal to be transferred. This makes parallel data transfer an expensive option, so parallel data transfer is restricted to applications where fast data is required over short distances. Typical PC microcomputer parallel interfaces are, the disk drives and the printer port.

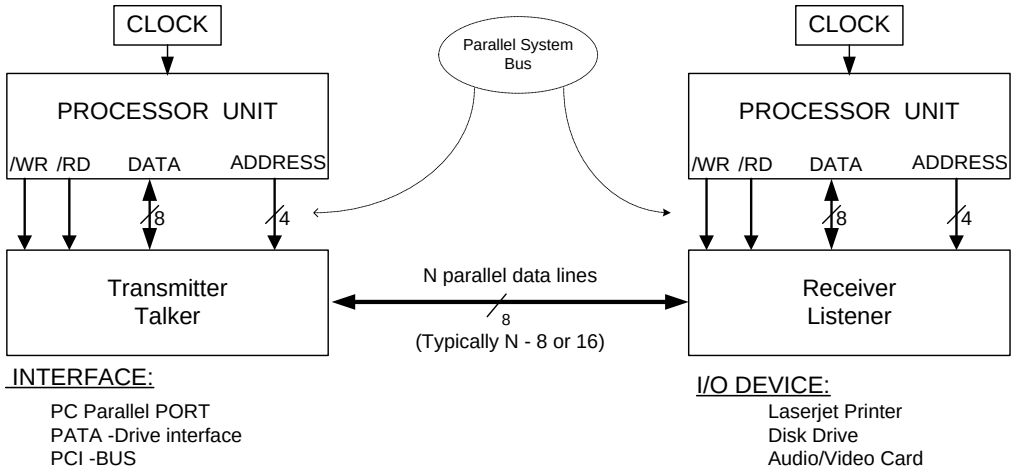


Figure 11. Parallel Data Transfer

SERIAL INTERFACE

Many applications require data to be transferred over a distance, say greater than a few metres and this situation is ideal for serial data transfer. Communications via the telephone or over a computer network use serial data transfer. Serial data is commonly transmitted over a single conductor, or a pair of wires, as shown Figure 12 , and the connection between the two communicating devices is simple. Furthermore, the connectors (plugs and sockets) required for the serial interface are simpler and less expensive than their parallel counterparts - compare the Parallel and Comports at the rear of a PC for example.

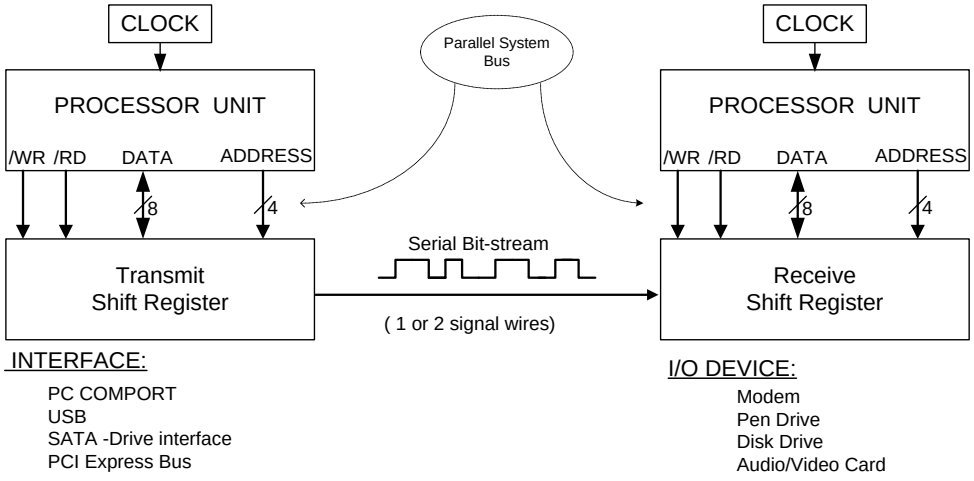


Figure 12. Serial Data Transfer

Since the processor system transfers data over the system bus in parallel binary format, serial data transfers require a mechanism for of converting parallel data into a serial bit-stream and this is accomplished using a simple electronic circuit called a **shift register**. When transmitting the shift register loads data from the parallel bus and clocks it out as a serial bit-stream over a single wire. At the receiving end of the serial link the serial bit-stream is clocked into a similar receive shift register which then makes the data available in parallel format for the receiving processor system data bus.

In the PC environment Universal Serial Bus (USB) is now increasingly used to replace existing COM and LPT ports, Serial –ATA is now becoming the standard for disk drive interfaces and PCI-Express is a high performance scalable serial bus that is the replacement for the older parallel PCI-X bus. The usual commercial motivating factors apply here - increased performance and reduced costs. The speed of serial data transfers is set by a clock and clock rates for serial devices can range from 480Mbps for USB to >1Gbps for Ethernet. High-speed serial interface schemes are now exploited to great effect in a whole new range of I/O devices. These serial interfaces are used for inter-chip data transfers. Inter-Integrated Circuit (I^2C) and Serial Peripheral Interface (SPI) are examples.
The obvious differences between parallel and serial data transfer schemes are :

- serial data transfer is slower but costs less than parallel data transfer - in terms of cabling cost (copper), whereas
- parallel data transfer is fast but requires more copper and is therefore more expensive.

In summary, if the distances involved are greater than a few metres serial data transfer is the appropriate choice and for short distance high-speed data transfers a parallel data transfer scheme is appropriate.

ANALOGUE I/O

Many sensors (and transducers) generate analogue signals that must be converted into binary data so that a processor system can digitally process the signal information. Figure 13 shows a typical analogue signal. Its notable characteristic is that it has a value at all instants in time and is said to be time continuous. To digitised the analogue signal it is sampled at regular instants as shown by the arrows at t1,t2,t3 in Figure 13. A numeric data value is assigned to each sample based on the height of the signal at the sampling point and the value is then stored in memory. The table, or more correctly ARRAY, shown in Figure 13 represents the digitised signal.

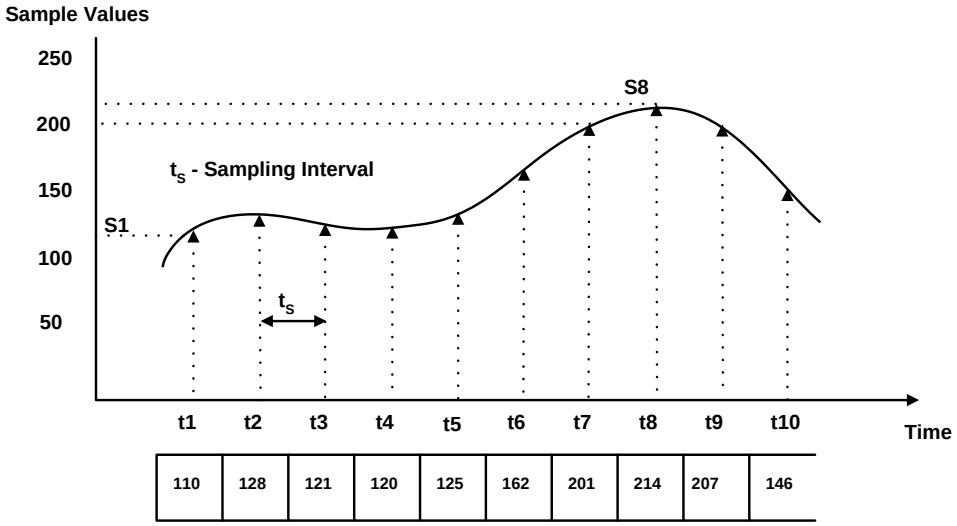


Figure 13. Analogue Signal Digitisation

The conversion between the analogue and digitised signals results in a loss of signal information – the digitised signal, as seem by the processor, has no knowledge of the signal between sample instants. In practice this presents no problem provided the analogue signal is sampled fast enough to record all signal details of interest.

The rule of thumb for estimating a suitable sampling rate is known as the Nyquist frequency and it states that :

The sampling frequency should be at least x2 the maximum frequency present in the analogue signal input.

In terms of audio sampling the input frequency rang is 20Hz-20kHz, which means that the sampling rate should be at least 40kHz – 44kHz is often quoted in CD quality sampling. For video applications, the signal frequency content extends from 0-5MHz therefore sampling rates in excess of 10MHz are required.

ANALOGUE INPUT - ANALOGUE-TO- DIGITAL CONVERSION (ADC)

The ADC interface converts analogue signals to a stream of binary sample values. It operates by comparing the input signal voltage to a internal reference voltage and producing a proportional binary weighted output value. For example in Figure 14 an input voltage of 0 volts generates a binary output value of 00000000; a value of 2.55 volts produces a value of 11111111; in-between a input value of 1.25 volts produces a binary value of 01111111. This particular ADC interface increases its output value by 1-bit for every 1mllivolt change in input voltage.

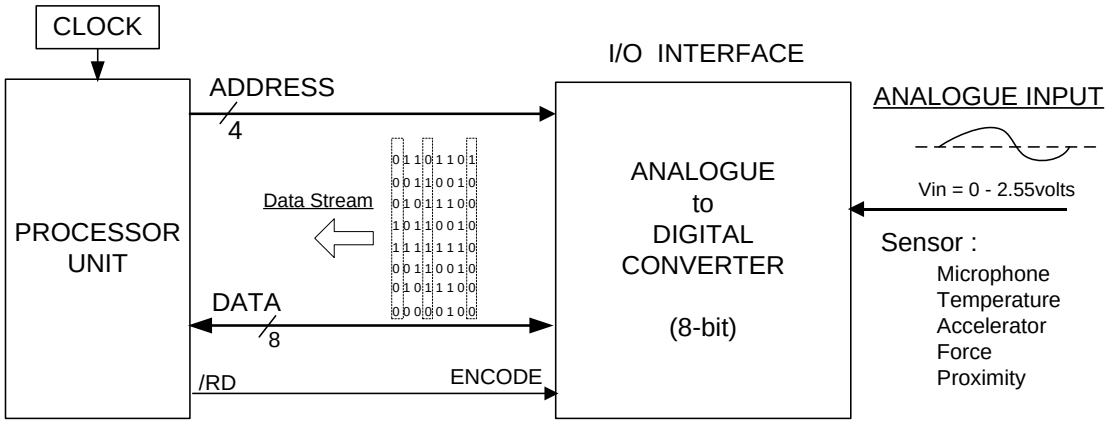


Figure 14. Analogue-to-Digital Conversion

The ENCODE signal in Figure 14 is used to initiate sampling events and is usually controlled by the processor. The Data Valid (DAV) signal is to signal the processor that a sample value is available. Very often this is connected to a processor interrupt pin.

Temperature, pressure, flow, position and acceleration are common examples of such analogue signal inputs.

Similarly, many output devices require analogue signals to control them. This requires an interface that converts the digital data generated by the processor system into an analogue voltage or current. Video displays, speech synthesisers and motors are typical examples of output devices that are controlled by analogue signal outputs.

ANALOGUE INPUT - DIGITAL-TO-ANALOGUE-CONVERSION (DAC)

A DAC performs the reverse process to the ADC. It converts binary sample values to a proportional analogue signal value.

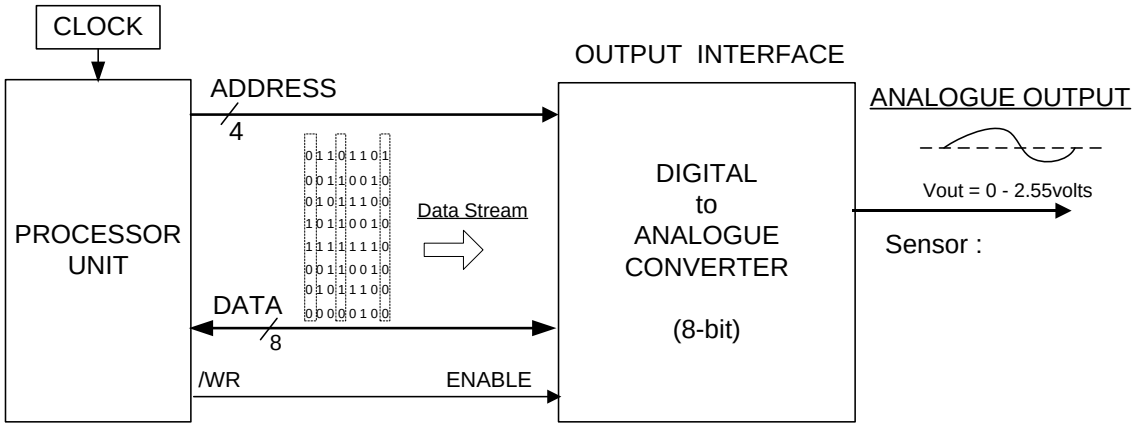


Figure 15. Digital-to-Analogue Conversion

The ENABLE signal is used by the processor system to 'tell' the DAC when to update the output signal.

It should be noted that the DAC output voltage is held fixed between sample updates. It the output sample rate is too low the output response can appear to have a stepped output profile. If this is not properly filtered, particularly in audio applications, the resulting sound contain high frequency 'clicks and pops' seriously degrading the sound quality.

Referring back to the ARRAY of sample points in Figure 13 it should be noted that the original signal can be reconstructed by replaying the sample values through a DAC interface at the same sampling interval (ts).

INTERRUPT AND DIRECT MEMORY ACCESS SYSTEMS

I/O data transfers often present a system 'bottleneck' which limits processor throughput and degrades system performance. The reasons for this are quite simple. In general, I/O devices operate at much slower speeds than processors. Processors can therefore, waste a lot of valuable processing time simply waiting for I/O processes to complete. In order to resolve this problem, processor systems commonly employ I/O controller devices to improve the efficiency of I/O data transfers. Two of these are discussed - INTERRUPT and DIRECT MEMORY ACCESS are introduced below.

INTERRUPTS AND THE INTERRUPT CONTROLLER

An interrupt system uses a combination of hardware and software to implement a more efficient I/O data transfers. The basic principle is that the processor continuously executes processing tasks and only responds to I/O devices when they request attention. Figure 16 illustrates the program flow that results when a programming task is interrupted and an interrupt service routine is executed.

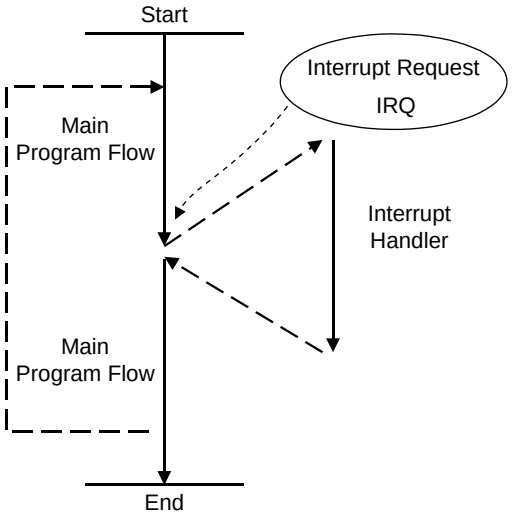


Figure 16. Program flow and Hardware Interrupt Response

In order to explain how the processor unit handles interrupt requests consider Figure 8 and the situation where the Digital I/O interface generates and interrupt request on IRQ1. It can also be assumed that the processor is busy at this point executing some other task. A typical generating and handling and interrupt request is as follows :

- The interface generates an interrupt request for the Interrupt Controller (IC) which responds by generating an IRQ signal for the processor unit (PU).

- The processor responds to the interrupt request by suspending the current processing task and jumping (vectoring) directly to an Interrupt Handler (also called an Interrupt Service Routine (ISR)) that processes the interrupt request - it this case the ISR reads data from the digital I/O port and stores it memory.

- When the ISR has completed this task the processor returns to its original task and resumes processing from the point at which the original task was suspended.

Computers are generally interfaced to a number of I/O devices , for example, keyboard, mouse, serial port, timer, disk, sound card so the interrupt system must be able to handle interrupt requests from several devices. Also, it must deal with the possibility of interrupt requests occurring at any time and in any order. For example, it has to resolve situations where a second interrupt request is made while a previous request is being processed. The function of the Interrupt controller (IC) is to deal with these more complex possibilities.

Analysis of the interrupt based system shows that it is the I/O device that initiates the request for data transfer. If the I/O device does not generate interrupt requests the processor ignores it and continues with other processing tasks. Since the processor only responds when interrupt requests are received then the processor is freed from the burden of repeatedly polling flags. Clearly this is a more efficient I/O data transfer technique.

Interrupt Controller

The system in Figure 8 shows three common interface devices any of which can generate interrupt requests. In normal operation, the serial interface would generate an interrupt when it receives a message from the serial data link, the digital interface would generate an interrupt when a key is pressed, and the analogue interface would generate an interrupt when a valid data sample is available. The Interrupt Controller (IC) handles interrupts from multiple peripherals, typically 8, as follows:

- When an interrupt is requested the IC signals the system processor by sending an IRQ signal. The PU responds by sending and an interrupt acknowledge signal IRQACK to the IC which then sends a code, via the data bus, that identifies the source of the interrupt. The processor uses this identifier to automatically generate an address the locates the first instruction in the appropriate Interrupt Handler routine.

- A second function of the IC is to prioritise interrupts. Typically, Interrupt 0 has the highest priority and Interrupt 7 the lowest. If during the execution of and interrupt handler, a second, higher priority, interrupt is received, the processor suspends execution of the current handler and immediately executes the higher priority interrupt handler. When this is completed the previously interrupted handler is resumed. For the case where a lower priority interrupt is requested the current interrupt handler is executed to completion before the lower priority interrupt handler is run

DIRECT MEMORY ACCESS (DMA) AND THE DMA CONTROLLER

Many data transfer applications, such high speed data acquisition, networks, disk drives and graphic displays require burst, or block, data transfers at high rates, say > 20MBytes/s. If all that is required of the I/O transfer is to move a block of data from one place to another then Direct Memory Access (DMA) is the appropriate I/O technique.

The essence of DMA is that the processor takes no part in the actual data transfer. It is effectively ' put to sleep' while the data transfer takes place and then 'awakened' when the transfer is completed. Figure 8 shows a DMA I/O subsystem with the Direct Memory Access Controller (DMAC) managing data transfers between the external I/O devices an the system RAM.

In Figure 8 the serial interface is shown connected to the DMAC and this is used to illustrate the most basic form of DMA transfer, a technique called 'Block Mode' transfer. The sequence of events require to implement a block transfer is as follows:

- The I/O interface requests a DMA data transfer by generating a DREQ1 signal.

- The DMAC signals this request to the processor by raising the processor's HOLD control signal.

- In response to the HOLD signal the processor completes its current bus cycle and then puts its address and data buses into their high-impedance (tri-state) condition. The processor then sends an acknowledgement to the DMAC via the hold acknowledge HOLDA signal.

- When the DMAC receives the acknowledge signal it sends an acknowledge (DACK1) signal to the peripheral and takes control of the address , data buses. The DMAC also takes control of the read and write control signals at this time.

- The DMA transfer now takes place - a block or byte of data is transferred from the peripheral to memory, from memory to the peripheral, or possibly, from memory to memory. The DMAC has complete control off all address, data and control signals.

- When the data transfer is complete the DMAC removes the HOLD signal and bus control is returned to the processor.

An extension to the I/O-memory transfers described above is a DMA transfer from Memory-to Memory. This is a situation required most frequently in graphical processing applications.

In PC systems DMA transfers are used extensively in network, audio and video/graphic processing.

Faculty of Engineering, Computing & Creative Industries

Edinburgh Napier UNIVERSITY

Module : Computer Systems 1

Module Number : CSN07101

Case Studies : System Architectures

Student Study Material

Version 1.1 Author : Frank Greig December 17th , 2013

COMPUTER SYSTEM ARCHITECTURES

INTRODUCTION

Semi-conductor fabrication techniques continue to improve and as a direct consequence transistors sizes continue to decrease and the number of transistors that can be integrated onto a single chip continues to increase. This means that higher levels of hardware integration are possible and clock speeds get faster (> 3GHz). Given that devices containing in excess of 50Million transistors are now routinely manufactured then it is possible to fabricate an entire computer system onto a single chip. The PU, ROM, RAM and I/O interfaces are now fabricated within a single – ***System-On-a-Chip*** (SOC) - solution. Alternatively, it is equally possible to enhance the processor architecture so that it supports 32/64-bit operations, or build-in instruction and data memory cache memories, or include instruction pipelines that speed-up fetch-execute cycles, or to extended the processor ALU to include floating point arithmetic operations that make number-crunching faster.

In order further develop these ideas and broaden awareness of processor system architectures, some popular alternative system architectures are presented here. These reflect some of the possibilities for hardware integration and each architecture should be analysed in terms of how it varies from the basic processor system model developed earlier. Also consider what market segments and areas of application are these processors targeted at.

EXAMPLE 1 : MICROCONTROLLER - ATMEL ATMEGA328 (8-BIT MCU)

A Microcontroller (MCU), is a intended for low-cost, single-chip, solutions for simple peripheral, instrumentation and control applications. MCUs can be found in USB device, such as Flash Pen Drives, keyboards, printers and scanners and they will also provide the user interface functions in mobile phones and MP3 players. In the wider field MCUs are used in metering, data-logging and smart-sensing applications or they may be found, in domestic appliances such as, a central heating controllers. It is worth noting that PCs use MCU's – the keyboard uses one to scan the keyboard and communicate keycodes serially and the PC motherboard uses an MCU to monitor processor temperature and control fans etc.

ARCHITECTURE AND MAIN FEATURES :

The example shown in Figure 17 is used in very low-cost (£1) sensor and control applications. It has been chosen here because it is also popular with many students and hobbyist in the form of an Arduino UNO development board. The MCU incorporates all the elements of a entire processor system into a single chip.

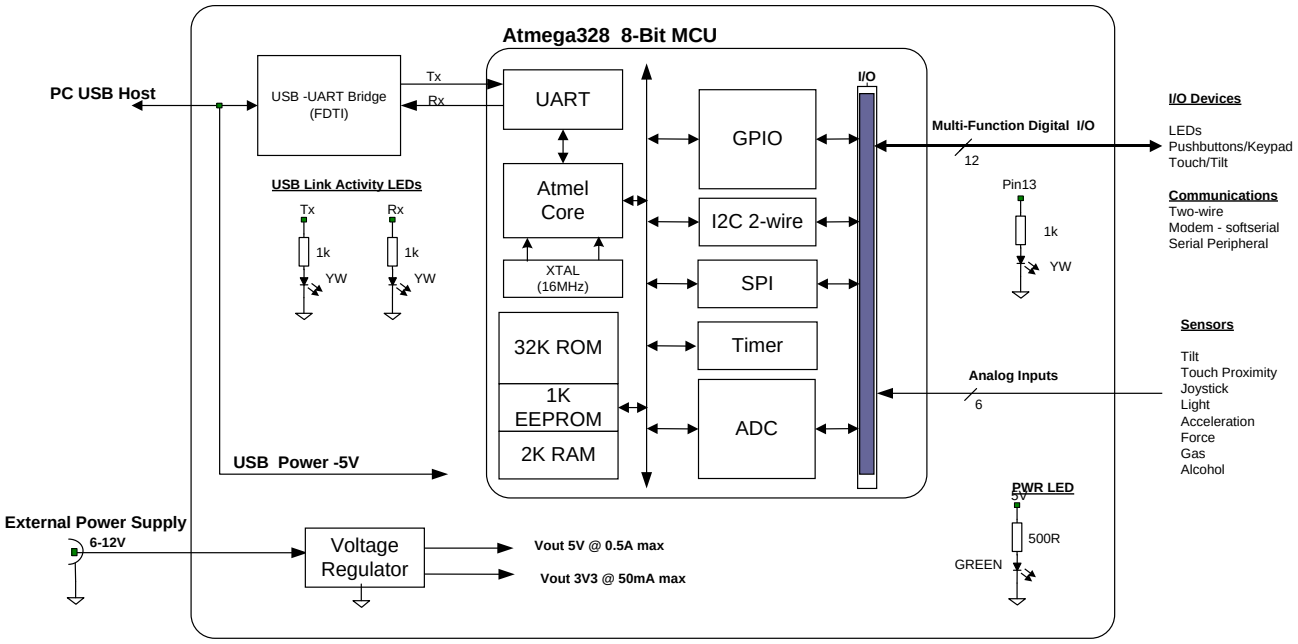


Figure 17. Microcontroller (MCU) Architecture

The device is manufactured by ATMEL and is a typical example of an 8-bit processor core. There are thousands of similar chips available form the hundred of chip manufacturers – Microchip, Freescale, Texas Instruments, Silicon Laboratories, Cypress Semi-Conductor etc. The key thing to notices is that all the building blocks of a computer system are intergrated into a single chip – PU core, Flash ROM, RAM, Clock and a plethora of Analogue and Digital IO interfaces. It provides a 'System-on-a-Chip' solution for small embedded system applications, for example, smart sensors, printer drives and domestic appliances such as heating and washing machine controllers high-level

The data sheet available at - http://www.atmel.com/Images/doc8161.pdf - summarises the main feature of the devices as :

PROCESSOR - 8-BIT PROCESSOR CORE ADVANCED RISC ARCHITECTURE

– 131 Powerful Instructions – Most Single Clock Cycle Execution
– 32 x 8 General Purpose Working Registers
– Fully Static Operation
– Up to 20 MIPS Throughput at 20 MHz
– On-chip 2-cycle Multiplier

MEMORY

– 4/8/16/32K Bytes of In-System Self-Programmable Flash program memory (ATmega48PA/88PA/168PA/328P)
– 256/512/512/1K Bytes EEPROM (ATmega48PA/88PA/168PA/328P)
– 512/1K/1K/2K Bytes Internal SRAM (ATmega48PA/88PA/168PA/328P)
– Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
– Data retention: 20 years at 85°C/100 years at 25°C

IO INTERFACE FEATURES

– 23 Programmable I/O Lines
– Two 8-bit Timer/Counters and One 16-bit Timer/Counter
– Real Time Counter with Separate Oscillator
– Six Pulse With Modulator (PWM) Channels
– 8-channel 10-bit ADC in TQFP and QFN/MLF package
– 6-channel 10-bit ADC in PDIP Package with Temperature Measurement
– Programmable Serial USART
– Master/Slave SPI Serial Interface
– Byte-oriented 2-wire Serial Interface (Philips I2C compatible)
– Programmable Watchdog Timer with Separate On-chip Oscillator
– On-chip Analog Comparator
– Interrupt and Wake-up on Pin Change

SPECIAL MICROCONTROLLER FEATURES

– Power-on Reset and Programmable Brown-out Detection
– Internal Calibrated Oscillator
– External and Internal Interrupt Sources
– Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby, and Extended Standby
– Optional Boot Code Section with Independent Lock Bits
– Programming Lock for Software Security

OPERATING VOLTAGE

– 1.8 - 5.5V for ATmega48PA/88PA/168PA/328P

POWER MANAGEMENT

Low Power Consumption at 1 MHz, 1.8V, 25°C for ATmega48PA/88PA/168PA/328P:
– Active Mode: 0.2 mA
– Power-down Mode: 0.1 µA
– Power-save Mode: 0.75 µA (Including 32 kHz RTC)

EXAMPLE 2 – ARM MICROCONTROLLER – (16/32-BIT) NXP – ARM7 LPCXXXX

ARM processor cores are now manufactured by a huge range of companies, including Philips, Atmel, Analogue Devices, Freescale, and Texas Instruments. ARM is widely recognised as the 16/32-bit replacement for the long-serving Intel 8051 family of 8-bit devices. As applications become ever more complex, more processing power is required and 8-bit processors struggle to meet the performance demands, 16 or 32-bit processors are therefore required. As shown Figure 18 the basic ARM Microcontroller (μC or MCU) architecture integrates a complete computer system into a single chip and is intended to provide low-cost, single-chip, intelligent solutions for instrumentation and control applications. Mobile phones, set-top boxes, media severs are typical application areas for ARM MCUs

It should also be stated here that the basic ARM has undergone many enhancements since its introduction. In particular, more flash memory and data RAM has been added; the clock speed has increased to around 100MHz on some variants; and the range of integrated peripheral interfaces continues to grow, typical examples would be ADC and DACs but there are more exotic versions with a CAN bus controllers integrated with the processor core.

PROCESSOR ARCHITECTURE – NXP LPC21XX ARM MCUS

Phillips (now NXP) is one of the very many chip manufacturers that manufacture a range MCUs that contain ARM cores. The silicon labs family of devices are given the generic label LPCxxxx and there are several device variants available. The MCU variant that is discussed here is based on the LPC2100 device and Figure 18 shows a simplified architecture. Full data sheets for the LPCxxxx processors are available for download from- www.nxp.com They contains a wealth of useful descriptions on the various interfaces and are supported a broad range of application notes.

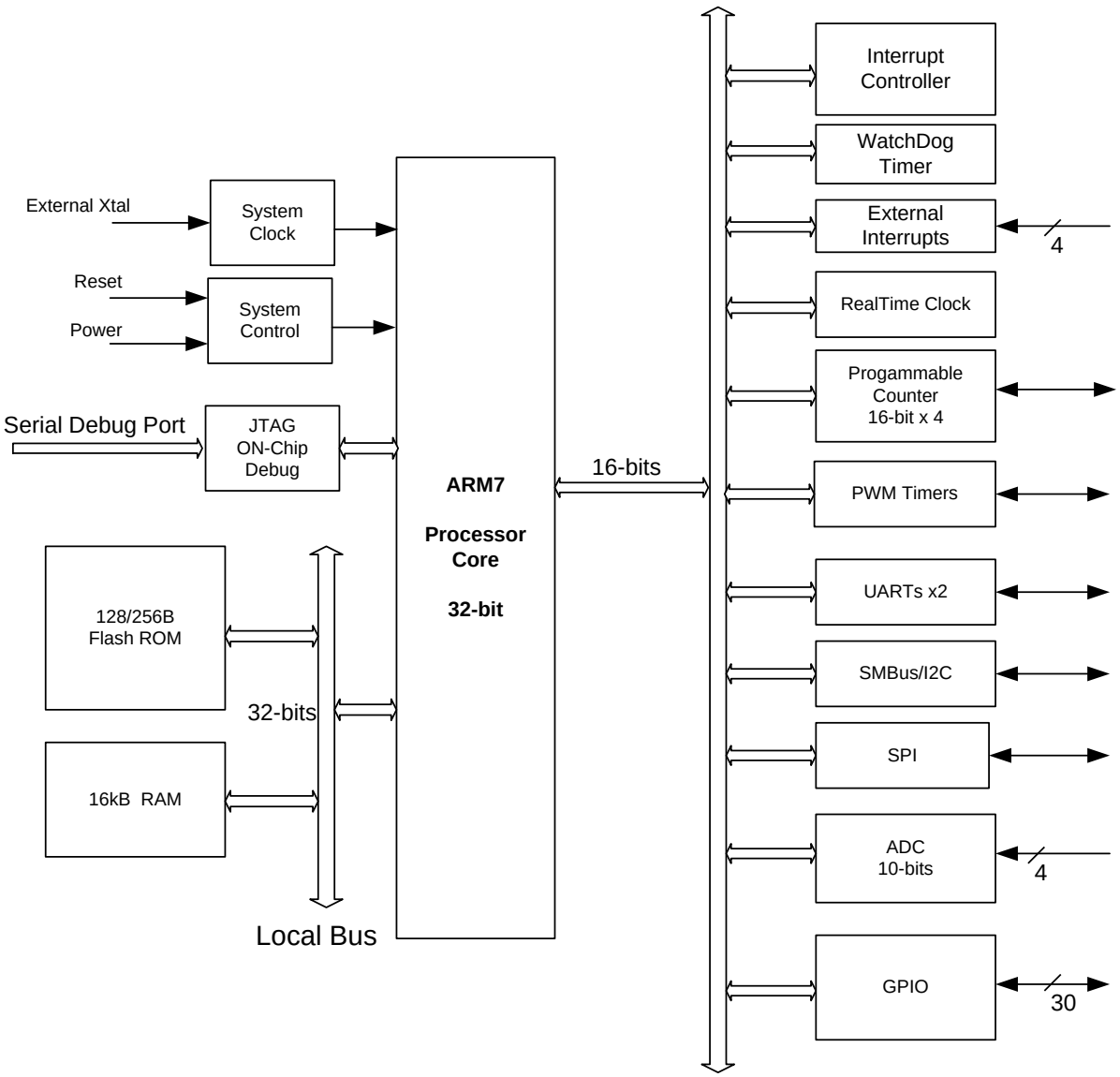


Figure 18. ARM 7 Microcontroller (MCU) Architecture.

The main features of this device are :

HIGH SPEED ARM μC CORE

- RISC based Architecture. Instruction Pipeline to speed-up execution cycles
- System Clock – 60MHz.
- Vectored Interrupt Controller with programmable priorities.

MEMORY

- ROM - 128k Bytes In-System Programmable FLASH Program Memory
- RAM - 16k Internal Data Static RAM

DIGITAL PERIPHERALS - ALL ARE 5V TOLERANT

- General Purpose I/O lines – GPIO Ports approximately 30 lines.
- Serial ports – UART1, UART 0 , SPI and SMBus/I2C.
- Pulse Width Modulator Module.
- Two General Purpose 16-bit Counter/Timers.
- Real-Time Clock module.
- Dedicated Watch-Dog Timer

ANALOG PERIPHERALS

10-bit ADC ±1LSB INL

- Programmable Throughput up to 200kSamples/s
- 4 External Inputs; Programmable as Single-Ended or Differential

CLOCK SOURCES

- Internal Programmable Oscillator – 1-30MHz
- External Oscillator: Crystal (XTAL) Clock to 60MHz

DUAL SUPPLY VOLTAGE

- Processor Core operates at 1.8V
- I/O Interfaces operate at 3.3V
- Multiple Power Saving Sleep and Shutdown Modes

ON-CHIP JTAG DEBUG

This full-speed non-intrusive debugging using low-cost debug tools. The JTAG interface is serial requiring only 4-pins.

EXAMPLE 3 – ADSP2181 DSP MICROCOMPUTER

APPLICATIONS :

This processor, manufactured by Analog Devices, is referred to as a Digital Signal Processor (DSP) Microcomputer This terminology is used to indicate that all the elements found in a typical microcomputer, PU, ROM, RAM and I/O, have been integrated into the device. Among many potential applications for this type of processor are – mobile phones, digital satellite set-top boxes, PC sound cards, modems, image processing, hard disk control. The market sector for this type of processor is vast and continues to grow rapidly. It should also be noted that many devices incorporate more than one processor, for example, a mobile phone will use an MCU to control the user interface - keypad, display, power management etc, and another DSP type to manage the radio communications. Similarly an MP3 player will use an MCU to manage the buttons, display, USB connection and flash files and another, specialised DSP, to manage the MP3 real-time decompression.

ARCHITECTURE AND MAIN FEATURES :

A key feature of digital signal processing applications is that they require large amounts of numerical processing. DSP processors always integrate a hardware multiplier into the processor core to speed-up the 'number crunching'.

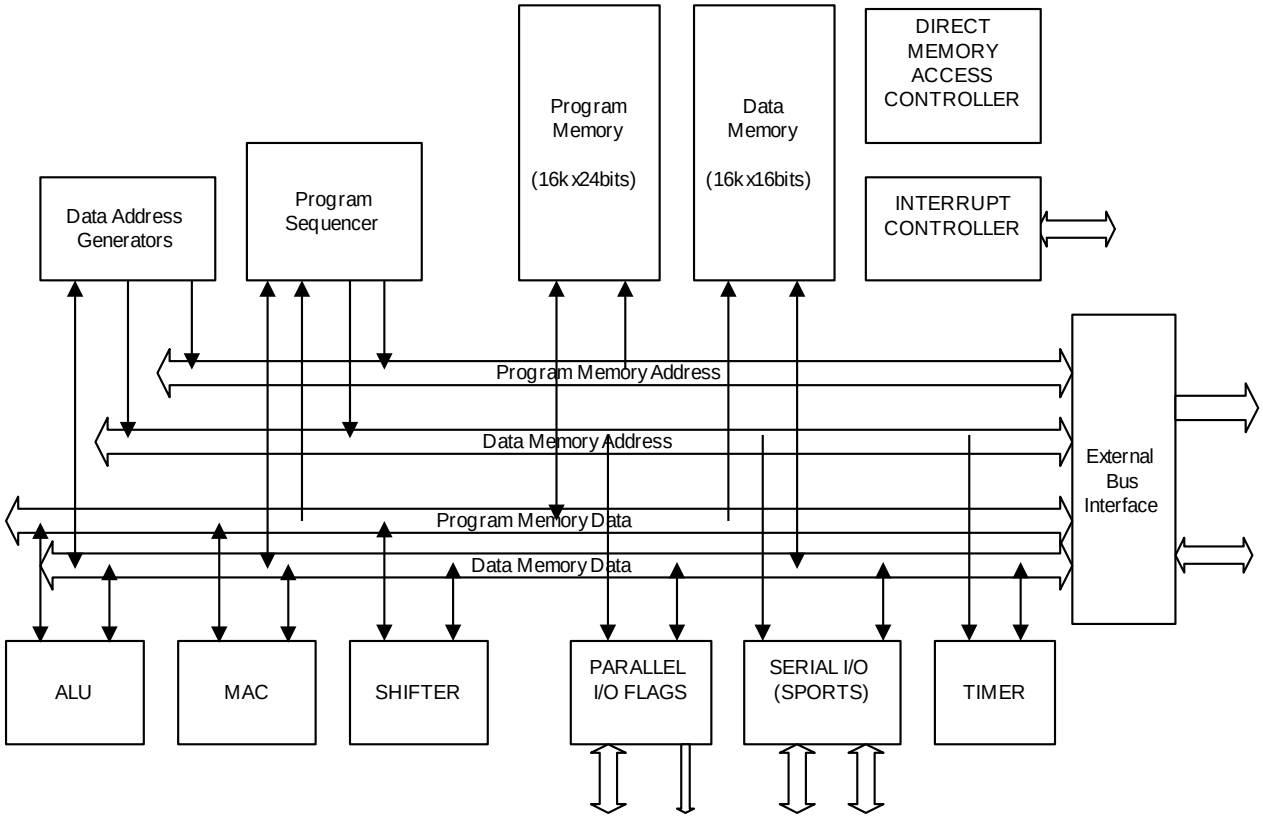


Figure 19 The ADSP2181 DSP Microcomputer

The main features of this device are :

- Separate program and data buses - 16-bit wide data path and 24-bit wide program/instruction path.
- 16k x24 On-chip program memory (PM)
- 16k x 16 On-chip data memory (DM)
- ON-chip peripherals - 2serial ports (SPORT), 11x parallel I/O lines, timer, interrupt and direct memory access controller.
- On-chip Multiplier(MACC) and Barrel Shifter - to speed-up numerically intensive applications.
- CMOS Technology with 33MHz clock.
- Operating Voltage range 3V - 5.5V. Power consumption 116mW @ 5V,33MHz.
- Standby Operation at 50μW@ 33MHz.

RELATED INFORMATION:

The is a wealth of information available on the ADSP family processors at - www.Analog.com. This includes a book on DSP techniques and applications, available in PDF format for free download. This site also makes available animated simulations which shows program execution on the DSP device.
Texas Instruments are also major players in the DSP market their web site also provides comprehensive DSP related information .

EXAMPLE 4 – PENTIUM PROCESSORS

PRIMARY MANUFACTURERS : INTEL AND ADVANCED MICRO DEVICES (AMD)

APPLICATIONS :

The Pentium family of processors are almost exclusively used in a desktop PCs, laptops, network servers. and games consoles (Xbox).

ARCHITECTURE AND MAIN FEATURES :

A basic Pentium processor is shown in Figure 20. It contains a core with the basic processor, Registers, ALU and Control Unit. However, in order to improve system throughput a number of enhancements are made to this basic processor core. Notably, in Figure 20 , cache memories are added to speed-up the instruction processing. A Floating Point Unit (FPU) has also been added to speed-up complex arithmetic operations, say for example the multiplication of Real Numbers (i.e. numbers in the form 1234.5678).

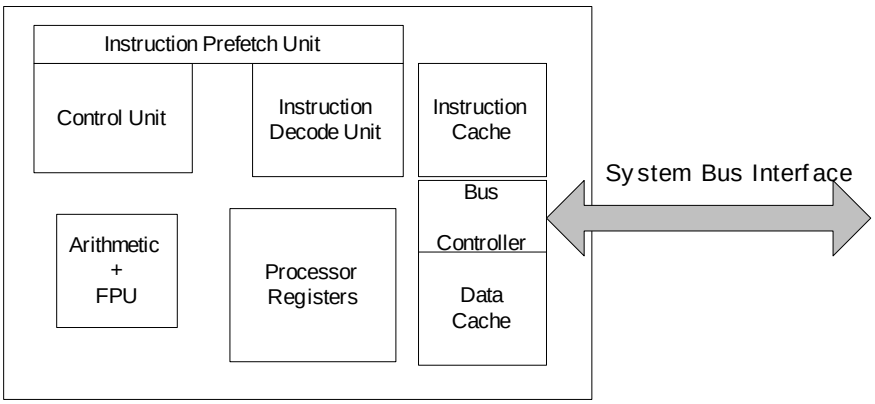


Figure 20. A simplified Pentium Processor Architecture

Intel was the originator of the 80xx family of processors. These have been used in PCs since the original PC-XT was introduced in the early eighties. Since that time the processor has evolved from 8088, to 80286,80386,80486 and now Pentium I, II, III and IV.

Since the PC market is huge other processor manufacturers, notably Advanced Micro Devices (AMD) and Cyrix, have introduced processors architectures to compete directly with Intel. Among the leading competition is AMD which produce and Athlon and Duron processors. There is continual battle to achieve better performance and lower-costs between these competitors. The actual complexity of the processor and the competitive nature of the PC market means that much of the detailed architecture for Pentium class processors is not readily available. The representation in Figure 20 is minimal and many variations exist, for example, iCore CPUs have three levels of cache memory integrated into the processor chip for increase performance and dual-core variants, those intended for laptops, are optimised for lower power consumption.

PROCESSOR PERFORMANCE IMPROVEMENTS

Improving processor performance in a continuous quest among processor manufacturers such as Intel, AMD and Cyrix and the several approaches have been used.

- The obvious method is to **increase the processor clock speed.** Although this approach has achieved remarkable results, with processors now running at an astounding 4.3GHz, it is not sustainable. Reducing the size of transistors in order to increase speed has reached the point where the transistors are now so small that they stop functioning as switches. Furthermore, the heat generated within the processor core due to faster clocks speed is becoming excessive and devices are now in great danger of failure by burning out.

- **Increase parallel operations** through the use of hyperthreading and multi-stage pipelines. These approaches are in widespread use but they are, at present, only applied within single processor systems.

- **Use multiple processor cores**. Dual and quad-core processors are now mainstream and this approach is set dominate the future. Simply put, this approach keeps the processor clock speed at manageable levels and adds additional processor cores. Processing tasks can now be run in parallel, for example, one core can manage a network task while others runs a tasks associated with an office application or a game. The multi-processor approach is not new but high manufacturing costs meant that they were not viable for low-cost PC systems. An extremely important aspect of realising the potential of multiprocessor technology is that it needs new software to take full advantage of it. New languages and design methods are be required to fully exploit the potential of parallel processing.

PC MICROCOMPUTER ARCHITECTURE

The PC microcomputer is now in such widespread use that is worthy of separate discussion. However, the subject is so large that it would require a full course to do it in any real depth. All that is attempted here is to present the key components, indicate the basic system layout, and to put some physical representation to the terminology.

To construct a PC from the basic Pentium processor requires the addition of a number of support chips and processor manufactures develop standard 'Chipsets ' which integrate all the circuitry required to interface the processor to memory and peripherals devices. Most chipsets use two sections, labelled in Figure 22 as 'Northbridge' and ' Southbridge'. These chips are complex devices which 'mop-up' all of the system interface functions into just two chips, reducing the cost and complexity of producing system motherboards.

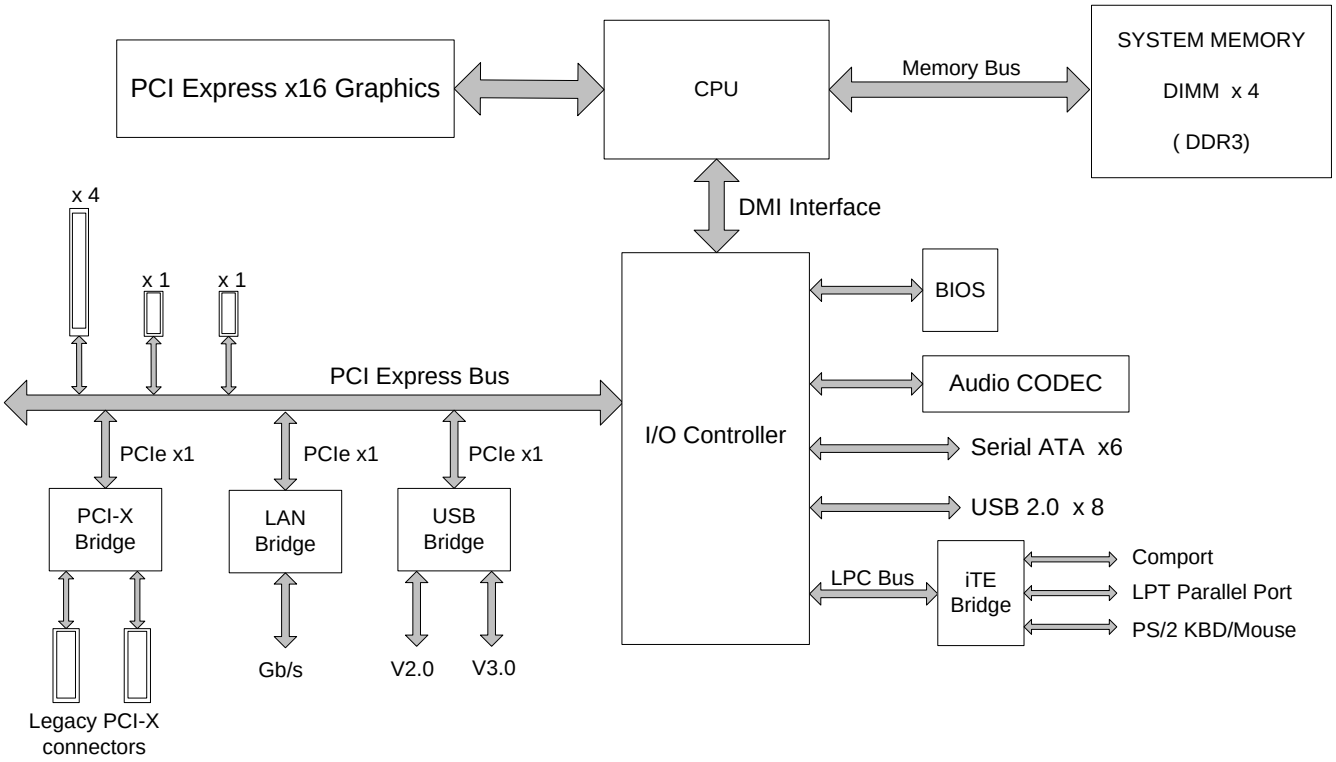


Figure 21 A Simplified PC Motherboard Architecture

In Figure 22 the NORTHBRIDGE(also referred to as Chipset 1) interfaces directly to the processor via the FSB (Front Side Bus) and handles the system memory (DDRAM) interface. It also provides a PCI Express (PCIe) x16 high-speed interface for a graphics card and provides a bridge to the Southbridge chip (also called Chipset 2). The Southbridge provides support for a variety of peripheral interfaces, for example, the IDE, ISA bus, USB, Comports Parallel port, and for the present time at least a legacy interface for PCI-x 2.0. Notice also that the BIOS boot-mode Flash ROM is interfaced through this chip.

It should not go unnoticed that numerous abbreviations have been used for the system blocks in Figure 22. This is unavoidable in 'PC speak' so a brief explanation of the terms used is given below.

BIOS - Basic Input Output System

BIOS is the program that executes when the PC is powered-up. It is usually a single Flash-EPROM device that executes on power-up to provide a basic user interface. BIOS allows various hardware configurations to be setup, for example, I/O port options, passwords, boot options and disk drives. On power-up BIOS performs a system memory test, configures the I/O system, including the hard disk drive, before loading the operating system and any application software from the hard disk.

DDRAM - Double Data Rate Random Access Memory

DRAM is used exclusively in PC memory systems because a PC systems continue to demand larger-and-larger amounts of memory. As discussed previously DRAM offers the best compromise between bit-density, data transfer rate, power consumption, and of course, cost. At present Double- Data Rate RAM (DDRAM) is used because it provides data at the fastest rate.

DDRAM is just the current stage of PC memory evolution that has gone from EDO DRAM, SDRAM and is now at DDRAM. The sole purpose of the evolution is to supply data hungry processors with data at an ever faster rate. DDRAM operates with bus speeds of 133 - 266MHz Double Data Rate RAM allows data to be delivered to the processor on both edges of the clock, doubling the data transfer rate for example 266MHz to 533MHz.

Memory Bandwidth

The rate at which data is transferred from the system memory to the processor is determined by the speed of the front-side bus (FSB), or more accurately, a fixed fraction, 1/2,1/4, etc. Bandwidth is also dependant of the number of bus cycles required to produce data from the memory chip, for example, Double Data Rate(DDR). The table below links the memory type, bus speed and the available bandwidth. It is assumed that the bus is 64bits (8Bytes).

Type	Bus Clock Speed (MHz)	Transfer Rate (Max MB/s)	Bandwidth (Bps)
DDR3-800	400	6,400	PC3-6400
DDR3-1066	533	8,500	PC3-6400
DDR3-1333	666	10,666	PC3-10600
DDR3-1600	12,800	12,800	PC3-12800

The DIMM Memory Module shown above has 4GB of memory and is capable of supplying a processor with data at 6.4GGB/s. This a calculated as follows:

PC3-6400 - is 8B x 2 x 400 = 6400MBytes/sec (64bits bus is 8Bytes x2 for double data rate).

TRENDS IN BUS SYSTEMS

There is now a strong move away form parallel system buses toward high-speed serial buses. Borrowing from advances in network communications technology, it is now possible to cost-effectively , deploy very high-speed, point-to-point, serial interfaces that match or surpass the performance existing parallel bus systems. The cost benefits of serial interfaces – less copper for cables, simpler connectors, small size – makes them the preferred choice in newer system designs.

PCI Express (PCIe) - Peripheral Component Interconnect Bus

PCIe borrows from Networking technology to provide increased speed and simpler connections. It is a high-speed scalable serial interface that operates at 500MBps per channel,(referred to as ***lanes*** in PCIe speak). The scalable nature of PCIe means that additional channels can be added to create bus speeds of 16GBps – PCIe x16 gives 16 x 500MBps x 2 = 16GBps (the x2 assumes transmission is in a single direction).. At this speed PCIe provides a much improved performance over the older parallel PCI-X technology and is more than capable of meeting the demands of high performance systems such as graphics interfaces, servers and RAID arrays.

As with all bus systems the PCI specification evolves and is now at PCI 3.0 which is capable of 32GB/s transfers over a 16 lane bus.

Disk Drives - Serial AT Attachment (SATA)

Serial – ATA also borrows from Networking technology to provide a high-speed interface with simpler connections and reduced costs. The SATA interface employs a bespoke protocol which is optimised for disk data transfers. SATA operates at 3Gb/s and 6Gb/s using a simple low-cost 7pin connector..

USB – Universal Serial Bus
USB was introduced to reduce PC costs by replacing all the connectors found on the back of a PC. Since connectors are expensive, replacing the Keyboard, Mouse, two Serial Comport and Parallel Printer port would result in a significant saving.

USB 2.0 is the current version of USB. It is a PC centric system (a PC must act as the host controller) that operates 480Mbits/s and allows up to 127 devices to be connected through a hierarchical tree of user devices and hubs. A strong feature of USB connections is that they include power signals that can be used to provide power for peripheral devices removing the need for separate power supplies or batteries.

USB continues to evolve :

- USB 3.0 is about to be introduced offering bus speeds of 5Gbps and more efficient power handling than USB 2.0.
- Wireless USB - capitalises on the wide-acceptance of the USB protocol and implements it using Ultra-Wideband (UWB) wireless technology to provide a high-speed desktop wireless protocol which will undoubtedly compete with Bluetooth.
- USB-OTG (On-The-Go) removes the requirement for a PC host and allows devices, such as, cameras and printers to be directly connected. Additionally, Wireless USB devices will start to appear in the very near future.

LEGACY INTERFACES

The need-for-speed in PC systems drives the requirement to make system interfaces run ever faster. To keep pace bus systems and interfaces must improve and evolve. In PC systems at present PCI Express (PCIe) is replacing PCI-X; SATA is replacing PATA; and USB is replacing LPT parallel and serial comports.

PCI – X - Peripheral Component Interconnect Bus
The PCI-bus is a high-speed parallel bus that allows high-spec peripheral devices to be interfaced to the system. Usually 3-5 slots for PCI-interface cards are provided on the system motherboard. The adoption of the PCI-bus was important step in the rationalisation of PC system architecture because the PCI is a bus 'STANDARD' which is widely used in computer systems - it is not just a PC bus.
PCI bus speeds have reached their practical limit and is now being phased out in favour of PCIe bus technology.

AGP – Advanced Graphics Port
When it was introduced AGP offered faster graphics processing at lower cost using a dedicated interface port which is tightly coupled to the processor to give high-speed data transfers – theoretically AGPx 8 operates at 2.1GBps. Note the AGP is a Port not a Bus – only one AGP device can be connected to the port. AGP bus speed has reached its practical limit and is now being phased out in favour of PCIe bus technology.

Disk Drives - Integrated Drive Electronics (IDE/PATA)
The IDE connector provides an interface for hard disk drives and CD-ROMs. There are usually two connectors, the Primary and Secondary, with each supporting two drives termed the Master and Slave. IDE is a parallel bus technology (PATA) has been around for a very long time) that operates at around 133MBps, using the familiar 40pin connectors and ribbon cables. PATA is now being superseded by a high-speed serial interface, Serial – ATA.

COMPORTS – Communications Port
The original purpose for serial ports is to connect communications equipment (a MODEM) to the PC, however, comports are now commonly used to provide remote access to a wide-range of embedded systems for example, set-top-boxes, and network equipment such as hubs and routers. Although these are being phased-out and being replaced by USB there is still very often a single serial port provided - PC Comport – COM1.

Parallel Port – (LPT)
The original purpose of the parallel port was to connect a printer but, with time, this is an 8-bit port that has evolved into a general purpose bi-directional port through which all sorts of devices are be interfaced for example, external disk drives, tape streamers and video cameras etc. Parallel port will eventually disappear to be replaced by USB.

These interfaces are now only seen in older systems. New systems will use the interfaces identified in Figure 22.

PC MOTHERBOARD - MSI 975X

The motherboard image presented in Figure 22 is intended to give some physical identity to the block diagram introduced in Figure 22. The modular structure of the motherboard allows flexibility in the system configuration. The potential for configuring different processors, amounts and types of system memory, hard disks, CDROMs, DVD drives, graphics plug-ins etc is seemingly endless. The key point to remember is that system components can be added or upgraded simply. With a minimum of common sense and a little confidence its a straightforward task to upgrade most parts of a PC - really its not difficult!

Note : A full description is available, in PDF document format, for the GIGABYTE GA-P67A is available from the website - http://www.gigabyte.com/products/product-page.aspx?pid=3759#ov

Figure 22 A typical modern PC Motherboard.

MOTHERBOARD SPECIFICATIONS

CPU

Supports LGA1155 packaged processors, for example, i3,i5,i7 , Pentium, Celeron Intel Core.

CHIPSET

- Intel® P67 Express Chipset

SYSTEM MEMORY

- 4 DIMM slots of 1.5 Volt DDR3 SDRAM
- Up to 32GB memory size
- Dual Channel DDR3 2133/1866/1600/1333/1066 MHz memory modules

BUS INTERFACES

- Two PCIe x16 slots + Two PCIe x 1 slots or One PCIe x4 slot
- SATA 2 x 6Gb/s + 4 x 3Gb/s + 2 x eSATA 6Gb/s on back panel; Support for RAID 0,1,5,10
- USB upto 14 USB V2.0 – 8 on back panel; upto 4 x USB3.0/2.0 2 on pack panel.
- 1 x Realtek RTL8111E 100/1000Mb/s Ethernet
- Audio Realtek ALC889 codec high definition audio.

MOBILE DEVICES - TABLET ARCHITECTURE

Tablets are now outselling PCs and this trend is set to continue for the foreseeable future. Perhaps the desktop PC is a thing-of-the-past and laptops and netbooks will follow suit in the near future. Discuss?

The primary design goals for any mobile device are high-performance coupled with long battery lifetimes, however, in electronic terms these are directly conflicting requirements - faster processor clock speeds and the more complex architectures require more battery power and hence a shorter battery lifetime between charges. For chip designers the trade-off between performance and power consumption is a constant battle that has led to very sophisticated power management systems being integrated into the device architectures in order to minimise power consumption. In essence these work on the principle that if something is not being used, even for a microsecond(or less) switch it off.

A basic tablet architecture is shown in Figure 23. The core component is a single chip – a highly complex multi-core processor. This example is based on Open Multimedia Applications Processor (OMAP) processor from Texas Instruments which integrates several processor cores including : dual Arm Cortex A9 processor units; a Graphic Processor Unit for 2D and 3D graphics; a Digital Signal Processor to handle Camera Image processing, several specialist processors (often referred to as 'engines' for handling audio, video encoding/decoding, security encryption and decryption. In addition to the multiple processor cores there is the usual on-chip ROM and RAM (limited in this case) plus I/O interfaces to provide support for wireless, audio and touchscreen devices.

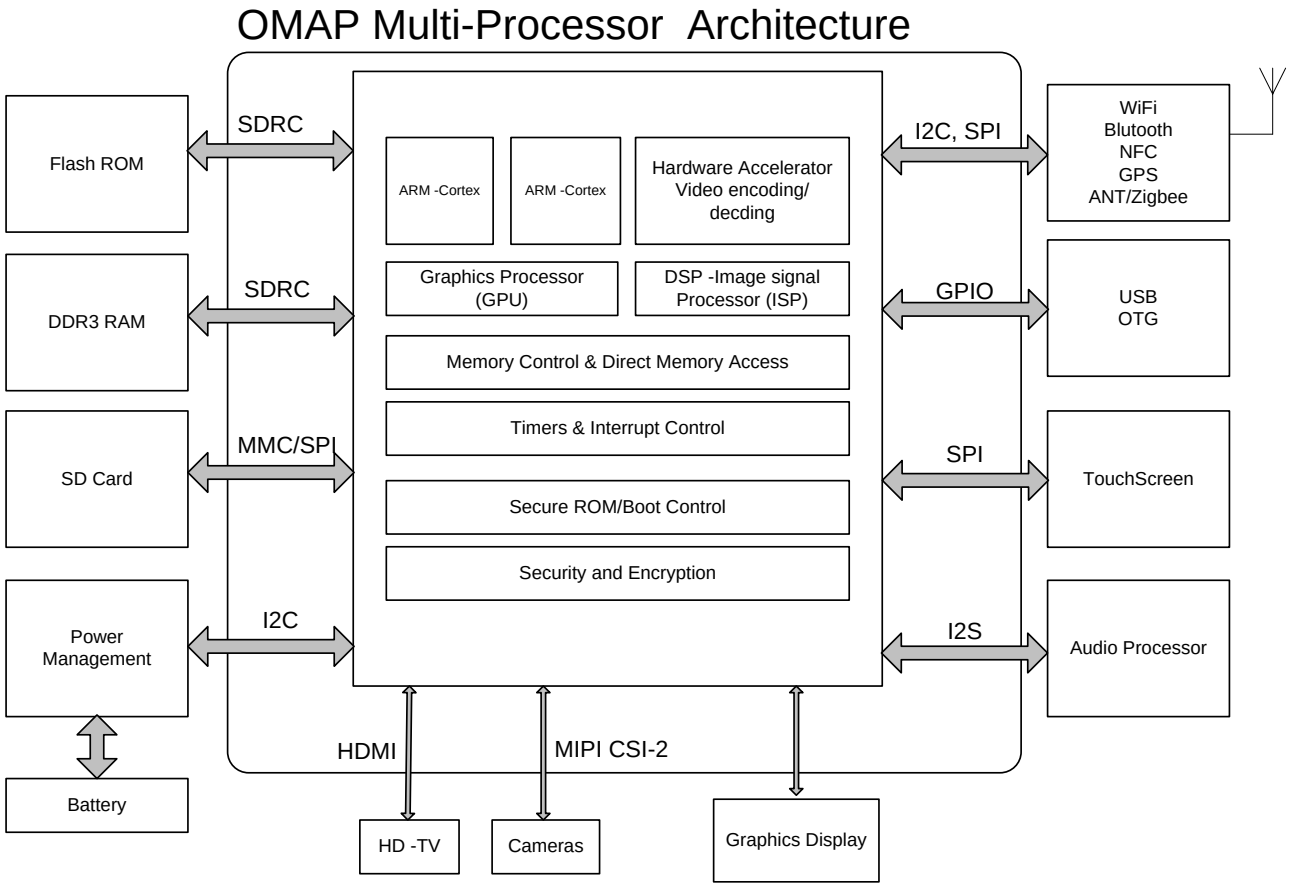


Figure 23 A very simplified Tablet Architecture

It should be noted that this is a very simplified view of the architecture that is intend to show that processor hardware architecture is now based in multiple core architecture that consist of several processors integrated into a single chip and also that a range of different processer types, DSP, MCU,GPU are intergrated into a single chip.

The full story of OMAP is available here http://www.ti.com/lit/ug/swpu235aa/swpu235aa.pdf - a full product description of around 5000 pages. Your welcome!

OMAP processors are used in the Kindle Fire and it should be noted that there are many similar type chips available from other manufactures – Apple, Samsung, Intel, Freescale, Broadcom (Raspberry PI) to name but a few..

Faculty of Engineering, Computing & Creative Industries

Edinburgh Napier UNIVERSITY

Module : Computer Systems 1

Module Number : CSN07101

Appendices:

- **A - Large & Small Numbers**
- **B - Binary & Hex Numbers**
- **C - Arithmetic & Logical Operations**
- **D - Specimen Test Paper**

APPENDIX A - HANDLING LARGE AND SMALL NUMBERS

INTRODUCTION

Current computers work at rates of about 3 GHz, with memory cycle times of perhaps 20 nanoseconds. What does this mean? In order to understand the operation of computers we often have to be able to understand very large (or small numbers) such as these. This short handout is an attempt to explain how such numbers can be manipulated.

WAYS OF REPRESENTING NUMBERS:

DECIMAL

Most people are used to the decimal system for everyday values. I might buy 500gms of sugar, and understand that this is the same as 0.5Kg. It is about 1.4 kilometres from my house to work. Such numbers are fairly easy to visualise. Unfortunately, the normal decimal system becomes cumbersome for handling very large and very small numbers. For instance, at the beginning I quoted the fact that a current PC has a processor that might run at 3 GHz. This literally means 3,000,000,000 times per second (1 Hz means once per second). The memory takes about 0.000000002 seconds to operate. These numbers can be very awkward; it is too easy to make a mistake. I said that the memory might take 0.00000002 of a second to operate. If I had written 0.0000002, I would have been wrong, but the difference is difficult to see (you have to count the numbers of zeroes).

SCIENTIFIC OR EXPONENTIAL NOTATION

If you have done some science or maths, you will probably be aware of this system. It is a powerful system for handling large and small numbers. The idea is to separate out the significant digits, and the information about the magnitude. So, 3,000,000,000 becomes $3x10^9$. This can be read as 'three times ten to the power nine'. Ten to the nine means 10 x 10 x 10 x 10 x 10 x 10 x 10 x 10 x 10. The system can be used for very small numbers too. 0.00000002 is the same as $2x10^{-8}$ which is another way of writing $2/10^8$ (three divided by ten raised to the power eight). This is a very widely used notation, and is often the easiest to use if you need to do calculations. It makes it easy to handle large numbers when you have to multiply and divide by factors of ten. To multiply by ten, you just add 1 to the power of 10. To divide by ten, you take one away. For instance $2x10^{-8}$ times 10 is $2x10^{-7}$. It has one problem as far as computing goes, you can't write it in plain text, as you need to be able to show the power of ten as a superscript[1]

POWERS OF A THOUSAND

This is probably the commonest method of representing large and small numbers, as it is just an extension of the system we use normally, but carried to bigger extremes than in normal usage. I can weigh my flour in grams or Kg (and I know that there is 1000 of one in the other). I said my house was about 1.4 Km from work, which I know is the same as 1400 metres. The essence of the system is to give a name to every unit that is a multiple of a thousand. Then you use the nearest named value as a prefix. If I want to measure my height, I could say it was 1.82 metres, not 0.00182 km. If I want to know the distance between my house and work, I could say it is 1400 metres, but the number is starting to get large, so I use the next magnitude up and say it is 1.4 kilometres. The system is not a strictly formal one, but if used consistently, can give a good balance between being comprehensible and being compact. Names have been given to many different magnitudes, each measure being separated by a factor of a thousand[2]:

SIZE	SCIENTIFIC	PREFIX	ABBREVIATION
0.000000001	10^{-9}	nano-	n
0.000001	10^{-6}	micro-	μ
0.001	10^{-3}	milli-	m
1	10^0	-	-
1000	10^3	kilo-	k
1000000	10^6	Mega-	M
1000000000	10^9	Giga-	G

[1] Actually, it is used in computing, but written as 2.10**8, for instance.

[2] The system is not strict. If metres are the fundamental unit of length, we should use kilometres and millimetres. However, we use small measures of length so often that we also use centimetres for convenience.

So, a PC that runs at $3x10^9$ Hz runs at 3 GHz. Early PCs ran with a clock speed of 10 MHz, which we can see means 10 million Hertz. A disk drive might take 10 ms to find the data. This is the same as 10x0.001 or 0.01 seconds. If we see a number quoted with a prefix, it should be relatively straightforward to work out what it means. Similarly, if we have a number that happens to fall at one of the named prefixes (10^3 or 10^{-6} for instance), we can work out the prefix. But what about numbers that fall in between, such as 10^5 or 10^{-2}? Then we have to work from a nearby named prefix. For example, the memory operates in 2.10^{-8} seconds. There is no name for 10^{-8} , so we find the nearest equivalent in the above table, 10^{-9}. Now, $2x10^{-8}$ is the same as $20x10^{-9}$, in other words 20 nanoseconds. Some serial data that took $5x10^{-5}$ seconds to be transmitted would have been transmitted in $50x10^{-6}$ seconds, or 50 microseconds.

CONVERTING POWERS OF TEN

In the last section, I converted powers of ten in a way that might not be familiar to you. For instance, I said that "$3x10^{-8}$ is the same as $30x10^{-9}$". If I need to adjust the powers of ten, I can do it by multiplying one by ten, and dividing the other by ten. After all, if I take a number, such as 6, and multiply it by ten (to get 60), then divide by ten, I end up with the number I started with. The same applies to the number in scientific notation. I start with 3 times 10^{-8}; I multiply the three by ten to get 30, and divide the 10^{-8} by ten to get 10^{-9} . Put them back together to get $30x10^{-9}$. I can also do it the other way round: divide the first number by 10, and multiply the second by 10. If a chip responds in $3x10^{-7}$ seconds, I can convert this to $0.3x10^{-6}$, or 0.3 microseconds. This is similar to saying that 300 gms of flour is the same as $3x10^2$ grams, which is the same as $0.3x10^3$ gms which is the same as 0.3 Kgs.

APPENDIX B - DECIMAL BINARY AND HEXADECIMAL NUMBERS

DECIMAL NUMBERS

We can easily comprehend a number like 246 because we use the decimal numbering system everyday: We don't think about the details of the number, or the fact that it operates in base-10 arithmetic. However, we are about to encounter Binary (base-2) and Hexadecimal (base-16) numbering systems, so a brief review of what we already know should provide a good starting point.
The decimal number system uses 10 symbols, 0,1,2,3,4,5,6,7,8,9 and all arithmetic operation are performed using only these symbols. No other symbols are allowed in base-10 arithmetic.
Reading number 246 aloud we would say - Two Hundred and forty six, or, Two hundred + Four tens + Six units. Expressing this in arithmetic notation as :

$$(2 \times 100) + (4 \times 10) + (6 \times 1) = 246$$

or, using base-10 notation

$$(2 \times 10^2) + (4 \times 10^1) + (6 \times 10^0) = 246$$

Notice that starting with the least significant digit, the units, the number base,10 in this case is incremented as we move from right-to-left, 10^0, 10^1, 10^2 and so on. This numeric representation is the general format of any numbering system and it applies to numbering systems using any base. We will only consider binary and hexadecimal systems here but the rules are applicable to any numeric base.

BINARY NUMBERS

The operation of a Digital Computer is based on switching circuits which can have only two states, ON or OFF, HIGH or LOW , 1 or 0, therefore, we need to operate on a numbering system with the base 2. In the binary number system the only allowable symbols are 1 and 0. The value 11110110 is a valid binary number which in its full arithmetic format is expressed as:

$$(1 \times 2^7) + (1 \times 2^6) + (1 \times 2^5) + (1 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (1 \times 2^1) + (0 \times 2^0)$$

OR

$$(1 \times 128) + (1 \times 64) + (1 \times 32) + (1 \times 16) + (0 \times 8) + (1 \times 4) + (1 \times 2) + (0 \times 1)$$

which is equivalent to decimal value 128 + 64 + 32 + 16 + 4 + 2 = 246

BITS, BYTES AND WORDS

A **B**inary dig**IT** - BIT is smallest unit of data stored in any digital system. A BIT can have only two values and is represented in the binary number system by the symbols '1' and '0'. Alternatives such as ON and OFF, TRUE and FALSE, and HIGH and LOW are often to describe switching networks and logic systems but their meaning is the same – they are binary values.

A grouping of 8 bits is known as a **BYTE**. To identify individual bits within a byte, each bit is labelled by the letter 'b' and a number which indicates the position of the bit within the number. As with normal decimal numbers, the **Most Significant Bit** (MSB) is on the left hand side and the **Least Significant Bit** (LSB) is on the right hand side.
For example, in the binary value shown below,b7, b3, b0 are 1. All other bits are 0..

b7	b6	b5	b4	b3	b2	b1	b0
1	0	0	0	1	0	0	1

This has the decimal equivalent of 128+8+1 = 137

A group of 16-bits is generally regarded as a **WORD,** although 32-bit and even 64-bit words are now used.

The table below shows the range of values covered by these groups of bits and also gives some useful signpost values.

NUMBER of BITS	COMMON TERM	2^N COMBINATIONS	NUMBER of COMBINATIONS
0		0	0
1	bit	2^1	2
2	snack	2^2 or 2x2	4
4	nibble	2^4 or 2x2x2x2	16
8	byte	2^8 or 2x2x2x2x2x2x2x2	256
10		2^{10} etc	1024 (1K)
16	word	2^{16} $(2^6) * (2^{10})$	65536 (64K)
20		2^{20}	1048576 (1M) - 1 Meg
32	long word	2^{32} $(2^2)* (2^{10}) * (2^{20})$	4 * 1k * 1M >> 4G - 4 Giga

For a practical application of these numbers, consider a microprocessor system described as having a 16-bit address bus and an 8-bit data bus. This means that the processor is capable of addressing 2^{16} which is 65536 in decimal or 64K memory terminology. The 8-bit data bus means that a 1 byte value can be stored at any memory address. The range of values stored is therefore, range 0-255 which is 2^8 or 256 values in total.

Also you may consider a Pentium style processor which supports a 32-bit address bus and a 32-bit data bus. What does this mean in terms of the maximum range of memory supported and the range of data values stored ?

BINARY >> DECIMAL CONVERSION

A convenient method to convert binary into its decimal equivalent to begin at b7 and accumulate the appropriate decimal equivalent,(power of 2), if a '1' is present in a bit position. Since b7 is the decimal equivalent of 128, moving from right-to-left decreases the decimal equivalent by a power of 2, thus, the equivalent decimal numbers are easy to remember - 128, 64,32, 16, 8, 4, 2, 1.

In the example above, bits b7 +b3 + b0 are present, so the equivalent the decimal is : 128(b7) + 8(b3) + 1(b0) = 137

DECIMAL >> BINARY CONVERSION

The textbook method for converting from decimal to binary repeatedly divide by 2 and note the remainder. The first remainder is the least-significant bit and the last remainder is the most-significant bit. For example, to convert decimal 246 to binary:

÷ 2	246	
÷ 2	123	r 0 << LSB
÷ 2	61	r 1
÷ 2	30	r 1
÷ 2	15	r 0
÷ 2	7	r 1
÷ 2	3	r 1
÷ 2	1	r 1
	0	r 1 << MSB

thus decimal 246 is represented as **11110110** in binary notation.

A more direct approach is to reverse the method described above, or even better use a calculator. This method is best explained by example, so lets return to the decimal equivalent of 246. Decimal 246 is made up from the binary weighted values (((128 + 64 = 192) + 32 = 224) + 16 = 240) + 4 +2 = 246. Thus 246 in decimal is 128+64+ 32 +16 +4 +2 which is expressed in binary as 11110110. It is best to practise this on a scrap of paper before attempting the mental arithmetic.
As another example, consider 79. This has the binary equivalent of :

79 = 64+8+4+2+1 = (0x 128) + (1x 64) + (0x 32) + (0x 16) + (1x 8) + (1x 4) + (1x 2) + (1x 1) or 01001111 in binary.

OR

USE A CALCULATOR

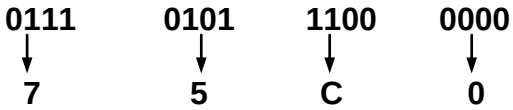
HEXADECIMAL NUMBERS

It is cumbersome and error-prone to read and write long binary numbers, therefore, the hexadecimal (base-16) number system is used. Hex is justified on the basis that it gives a compact representation for long binary values and that the conversion from hex to binary is simple. From our previous discussion it should be obvious that a base-16 number system needs 16 symbols to define all the allowable digits. The table below shows that in HEX the normal decimal digits 0..9 are used as the first ten symbols and then letters A..F are used for the numbers 11..15.

Decimal	Hex	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

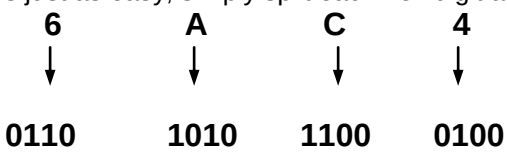
BINARY >> HEX

Since base-16 is equivalent to 2^4 then the conversion from binary to hex is simply a matter of grouping nibbles(4-bits) starting at the LSB and then converting each nibble into to its equivalent hex value. For example, to convert 0111010111000000 to hex:



HEX >> BINARY

Converting from hex to binary is just as easy, simply split each hex digit and then convert it into a four bit binary value.



These examples illustrate clearly the usefulness of hex notation. ***Hex is compact*** requiring only 4 digits to represent a 16-bit binary value and ***less prone to error***, '1' and '0' don't get mixed-up or omitted.

IDENTIFYING HEX NUMBERS

There are a number of different symbols used to identify a number as being hex. In Pascal, hex numbers are preceded by a dollar sign, for example $C4 indicates a hex byte. In C hex numbers are preceded with a 0x, for example 0x24. In assembler the letter 'H' is sometimes used immediately after the last digit, for example 55H.

APPENDIX C - ARITHMETIC AND LOGICAL OPERATIONS

A core requirement in any computer processing system is the ability to perform arithmetic operations. The most basic of these, addition and subtraction are covered here. The more complex operations of multiplication and division are not covered but by recalling your first attempts at multiplication using repeated additions then it is clear that multiplication is essentially a process of addition. Similarly, division is based on a repeated subtraction process.

Subtraction can be performed by the algebraic addition of negative numbers and this provides a further simplification for arithmetic operations. For example, the subtraction 5 – 3 = 2 is the same as the addition 5 + (-3) = 2. For this to work, however, there must be way to represent negative numbers in binary; it turns out that in the binary number system this is an easy thing to do using 2's complement numbers.

Since subtraction, multiplication and division can all be performed by addition the arithmetic hardware built into the processor unit (PU) only needs to perform addition. The obvious benefit being that it minimises the amount of circuitry required to implement the arithmetic unit.

BINARY ADDITION

The rules for binary addition are as follows :

		Carry	Sum
0 + 0	=	0	0
0 + 1	=	0	1
1 + 0	=	0	1
1 + 1	=	1	0
1 + 1 + 1	=	1	1

These rules are really no different to those for decimal addition, except that a carry is generated when the sum exceeds 2; thus 1+1 in binary generates a carry and a sum of 0. The following example should demonstrate the rules:

Decimal	Binary	Hex
56	00111000	38
+47	00101111	2F
103	01100111	67

For a quick check of the answer 0x67 is ((6 x16) = 96) + 7 = 103

BINARY SUBTRACTION

As was stated earlier the subtraction of binary numbers is performed by adding the negative, 2's complement, equivalent of the number to be subtracted. It is, therefore, necessary to understand the 2's complement numbers system before subtraction can be performed.

2'S COMPLEMENT – NEGATIVE NUMBERS

If a binary numbering system has only positive values it is referred to as unsigned binary system. A byte value can represent any value in the range 0-255. If negative numbers are required then the available 256 codes can be split into positive and negative values. This is exactly what happens in the 2's complement numbers system; the available 256 codes are split into the range -128 to +127. The table below illustrates how the assignment is made.

Decimal	2^s complement
-128	10000000
-127	10000001
-126	10000010
::	
-2	11111110
-1	11111111
0	00000000
1	00000001
2	00000010
::	
126	01111110
127	01111111

An important observation from the table above is that in a 2's complement numbering system all a negative values are identified by having a 1 in the Most Significant Bit position , b7 is always 1 for a negative value and 0 for a positive values. Note also that decimal zero 0 is represented by binary zero 00000000 in 2's complement representation.

The conversion from a positive value to its 2's complement equivalent is simple – **complement** all bits and then add 1. To complement a binary number simply change all 1's to 0's and all 0's to 1's. For example, the 2's complement of +65 is :

65	0100 0001	
1's Complement	1011 1110	
Add 1	0000 0001	
2's Complement	1011 1111	-65

thus -65 is 1011111 in 2s complement notation.

A more efficient method for the conversion process is to apply the following rule –

Starting with the LSB, copy upto and including the first '1' and then complement the remaining bits.

An important aspect of using 2's complement arithmetic is that the logic required to generate a 2's complement number is trivial; it is simply the NOT function with 1 added to the result.

As stated previously the attraction in using the 2's complement system becomes obvious when subtraction is to be performed. To subtract one number from another number, convert the negative number to its 2s complement form and then add the numbers. For example, to subtract 47 from 56, convert 47 to 2s complement (ie -47) then add 56+(-47):

Decimal		Binary	Hex
+47		00101111	2F
2's Complement	-47	11010001	D1
56		00111000	38
+ (-47)		11010001	D1
09		00001001	09

Note that, the carry generated during the Hex addition is ignored.

Lets consider another example, where the result is negative, say 47 – 56 = -09

Decimal		Binary	Hex
56		00111000	38
2's Complement	-56	11001000	C8
+47		00101111	2F
+ (-56)		11001000	C8
-09		11110111	F7

For a quick check, the binary result 11110111 is the 2's complement form of 00001001 which is binary 9. The result 11110111 is –9 in 2's complement format.

These results illustrate some important facts about computer arithmetic operations :

- If signed arithmetic is being used all negative numbers are stored in their 2's complement form.
- Addition is not affected by using 2's complement notation; positive values retain their true binary representation.
- Subtraction is performed by the addition of 2's complement numbers at no significant penalty in speed or hardware resources used..

LOGICAL OPERATORS

Decision making is a common occurrence in programming. Programming constructs like IF THEN, FOR DO and WHILE all require a decision to be taken. For example, the IF THEN construct takes a decision by selecting one of two program statements to be executed next; looping constructs, like WHILE, take a decision to determine whether the loop exits or whether the next iteration of the loop is executed.
The obvious question here is : How does a computer system make these decisions? The simple answer is that decisions are made on the basis that a bit, or a bit within a byte, or a particular binary value is detected. Logical operators allow individual bits, bytes or words to be tested and hence decisions can be made about what program instructions are executed next. The common logical operators are NOT, OR, AND and XOR.

NOT

The NOT operator has already been met, it is simply the complement operator. If a bit has the value 1 then NOT (1) is 0; similarly, if a bit has the value 0 then NOT (0) is 1. To apply the NOT operator to any binary value, change all zeros to ones and all ones to zeros. Simple!

This information is usually presented in a table called a TRUTH TABLE

INPUT	OUPUT
0	1
1	0

Read this as : - the output is the complement of the input.

OR

The OR operator shows 2 inputs and produces an output if either input is present . In truth table form this gives :

INPUTS		OUPUT
0	0	0
0	1	1
1	0	1
1	1	1

Read this as : - the output is 1 if any input is 1.

AND

The AND operator shows 2 inputs and produces an output only if both inputs are present . In truth table form this gives :

INPUTS		OUPUT
0	0	0
0	1	0
1	0	0
1	1	1

Read this as : - the output is 1 only if both inputs are 1.

XOR

The XOR operator shows 2 inputs and produces an output only if a single input is present but not both. In truth table form this gives :

INPUTS		OUPUT
0	0	0
0	1	1
1	0	1
1	1	0

Read this as : - the output is 1 if either input is present, but not both.

APPLYING LOGICAL OPERATORS

Logical operators are applied in exactly the same manner as the addition operator. However, there is a significant simplification : there is no CARRY to worry about. A few examples will make the process clear.

OR

OPERATOR	Binary	Hex
	00111000	38
OR	00101111	2F
RESULT	00111111	3F

AND

OPERATOR	Binary	Hex
	00111000	38
AND	00101111	2F
RESULT	00101000	28

XOR

OPERATOR	Binary	Hex
	00111000	38
XOR	00101111	2F
RESULT	00010111	17

It should be obvious from these results that applying different operators to the same numbers causes widely different results.

INTERPRETING THESE RESULTS

The significance of applying logical operators becomes clear when developing applications for Input-Output (I/O), peripheral control and image processing. The need to test that an input signal is 1or 0, or the need to SET, RESET and TOGGLE(simply change state) hardware control signals are common I/O programming tasks. Additionally, matching and comparing bits (using XOR) are fundamental operations in image processing and data encryption applications. The role of these **bitwise** operators are summarised below:

NOT – produces the complement of all data bits, 1>0 and 0>1.

OR with 0 Individual bits within a group of bits when or'd with 0 stay the same – 0 or 0 = 0; 0 or 1 =1
OR with 1 Forces individual bits within a group of bits to become SET (=1) – 1 or 0 = 1; 1 or 1 =1

AND with 1 Individual bits within a group of bits when and'd with 1 stay the same – 1 and 0 = 0; 1 and 1 =1
AND with 0 Forces individual bits within a group of bits to become RESET (=0) – 0 and 0 = 0; 0 and 1 =0

Bitwise AND is also used to test the state of individual bits within a group of bite - a process known as **BIT MASKING**

XOR with 1 - Forces bits to TOGGLE (1 > 0, 0 >1- change state).

Note that a detailed understanding of these operators is not necessary at this stage but any programming applications requiring I/O control need a firm understanding of logic operators .

APPENDIX D - SPECIMEN TEST PAPER

Attempt ALL Questions

1.	i. Convert the decimal value 231 to its 8-bit binary equivalent. ANS:	(2)
	ii. Convert the decimal value 231 to its hexadecimal equivalent. ANS:	(2)
	iii. Convert the hexadecimal value 0xECA75 to its 16-bit binary equivalent. ANS:	(1)

2.	Use 2's Complement binary addition to subtract 56 from 121. Note that all working must be clearly shown. ANS:	(5)

3.	Using the system Processor- Memory Reference Model (Appendix X) describe, as a step-by-step sequence, how the processor fetches and executes the following instruction : ADDRESS OBJECT CODE INSTRUCTION C:0x0010 C4 78 MOV R2,#0x78	(7)
	ANS:	

4.	i. Briefly describe the operation of Cache Memory. ANS:	(5)
	ii. Give an example of a Serial Interface and identify a peripheral device that would be connected to it? ANS	(3)

5.

For the PC system motherboard given below label clearly the following components :

i. Processor Socket.

ii. DRAM Memory DIMM slots.

iii. Intel P67 Chipset I/O Controller.

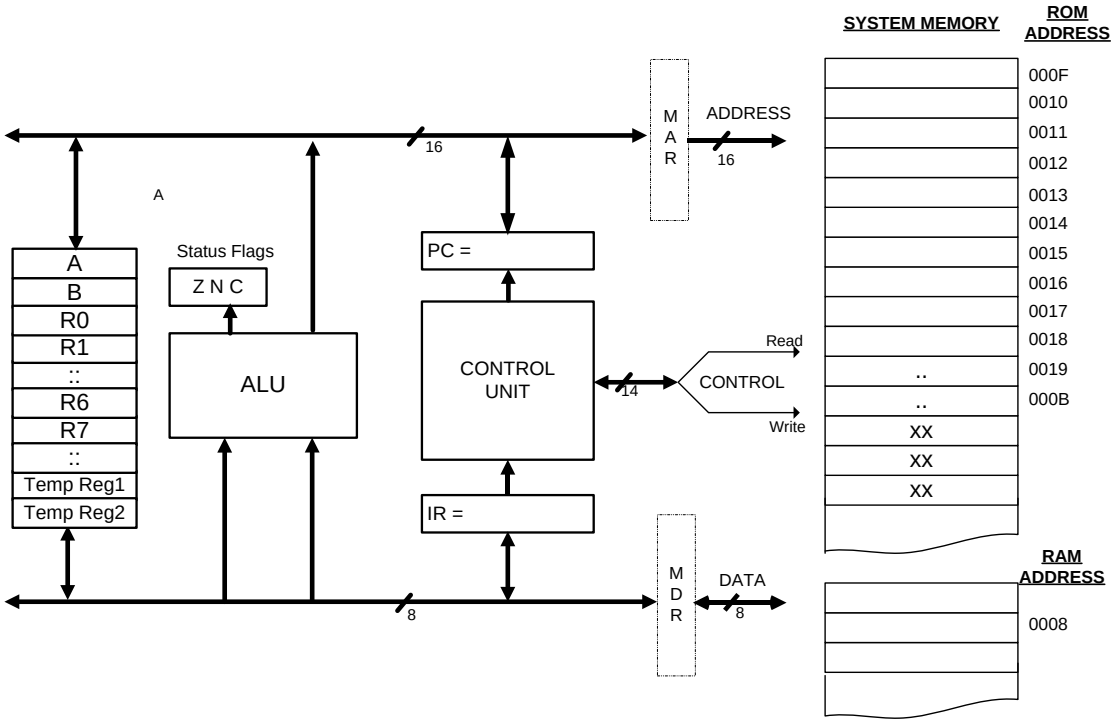
iv. x 16 PCI Express BUS slots.

v. SATA Disk Drive Connectors.

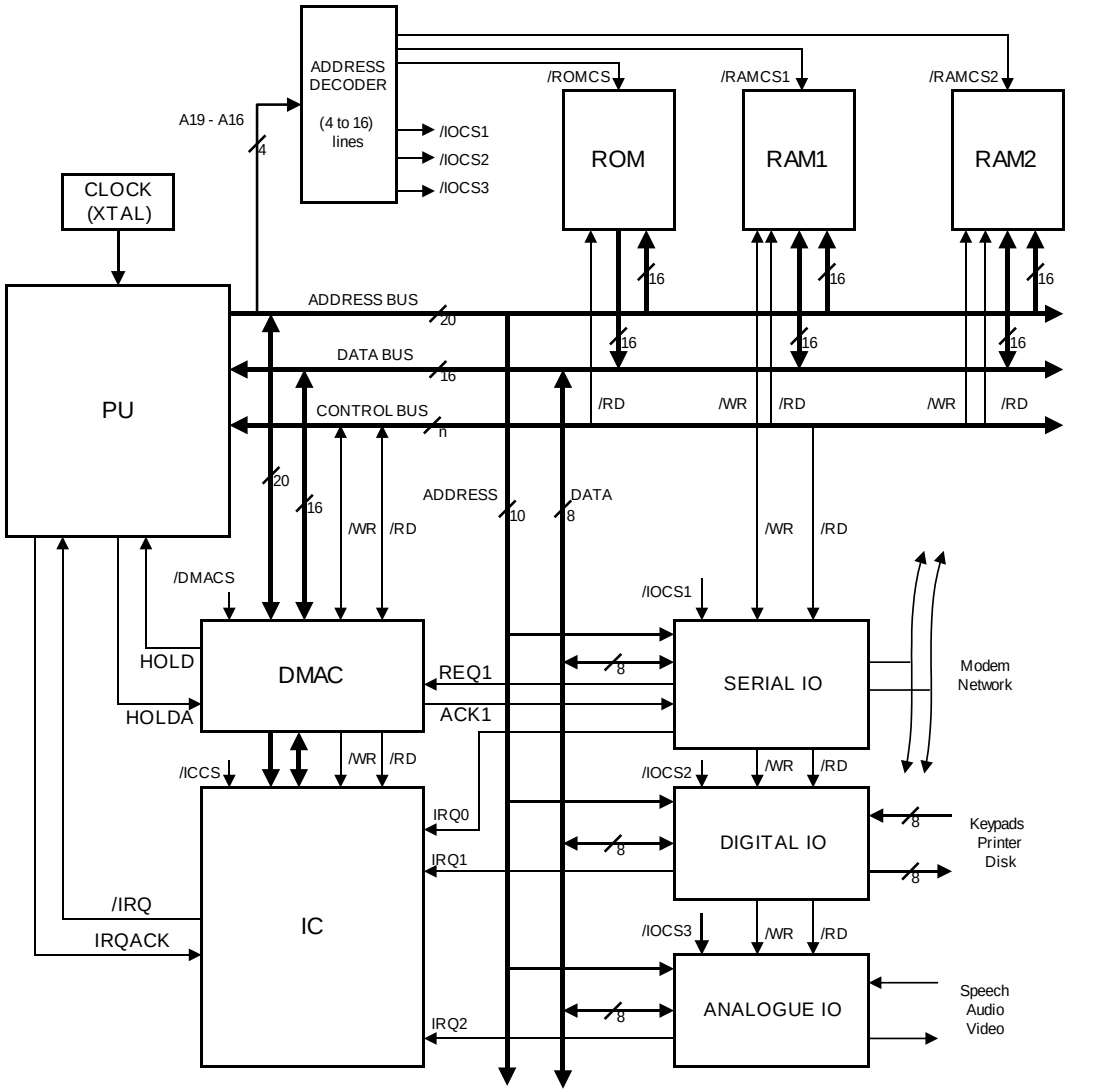
(5)

CLEARLY LABEL the DIAGRAM

APPENDIX X - PROCESSOR – MEMORY REFERENCE MODEL



APPENDIX Y - SYSTEM REFERENCE MODEL



Frank Greig • School of Computing • Edinburgh Napier University • Edinburgh •