10. SHARC has instruction word of 48 bits and 32-bit data word for integer and floating point operations and 40-bit extended floating point. SHARC functions as a VLIW (very large instruction word) processor. The word size is 48-bit for instructions, 32-bit for integers and standard floating-point (FP), and 40-bit for extended floating-point (EFP). Smaller 16 or 8-bit must also store as full 32-bit data. Therefore, the big endian or little endian data alignment is not considered during processing
11. SHARC also provides instructions for saturation integer operations. For example, the integer after operation should limit to a maximum value. These instructions are required in graphic processing.
12. SHARC permits parallel operations. It supports processing instruction level parallelism as well as memory access parallelism. Therefore, there can be multiple data accesses in a single instruction.

TigerSHARC **TigerSHARC** is a highest performance density family of processors from Analog Devices. The architecture provides precision high-performance integrated circuits used in analog and digital signal processing applications. A version of TigerSHARC is TigerSHARC ADSP-TS201.

TigerSHARC is designed for multiprocessing applications and for peak performance greater than BFLOPS (billion floating-point operations per second). MultipleTigerSHARCs can connect by serial communication at 1 GBps. ADSP-TS203SABP-050 processor processes using 250 MHz clock and on chip memory of 6 M bits and operates at 1.2 V/3.3 V. Low voltage design helps in processing with little power dissipation. Analog Devices TigerSHARCs have the highest performance per watt. A TigerSHARC version has 24 M bits ON-chip memory. TigerSHARC is available as the IP core also so that new applications with the core can be developed.

TigerSHARC finds applications in software baseband processing, 3G WCDMA baseband communication, cellular base stations and 14 Mbps HSDPA (High Speed Data Packets Access) networks for packet-based multimedia contents.

2.3.5 DSP

Advanced signal processor circuits consisting of MAC (Multiply and Accumulate) unit at a DSP provides fast multiplication of two operands and accumulates results at a single address. It computes fast an expression such as the following, $y_n = \Sigma (a_i \cdot x_{n-i})$, where the sum is made for i = 0, 1, 2, ..., N-1. Here i, n and N are the integers, a is a coefficient, x is independent variable or an input element and y is the dependent variable or an output element.

DSP processors invariably have Harvard architecture. Caches are also organized in Harvard architecture (separate I-cache and D-Cache).

Architecture of Digital Signal Processor The architecture of a DSP can be understood by considering an exemplary DSP of TMSC64x™ DSP generation.

The main structural units in a TMSC64x™ DSP generation and their functions are given in Table 2.4. Figure 2.21 shows the interconnections between twenty-five structural units by a block diagram for processor structure. Table 2.5 gives the additional structural units and the functions in the processors, TMS320C64x64™ VelociTI™, which is a VLIW architecture Extension.

2.4 PROCESSOR AND MEMORY ORGANIZATION

2.4.1 Processor Organization

Figure 2.22 shows a simple representation of organization of processor and memory in a system. The memory and IO devices interface the processor using buses. Figure 2.16 showed a detailed block diagram for internal

Table 2.4 Structural units and functions of processor in a DSP core

Structural Units in Core	*Functions*
Basic units	MDR, internal bus, data bus, address bus, control bus, bus interface unit, instruction fetch register, instruction decoder, control unit, instruction cache, data cache, multistage pipeline processing, multiline superscalar processing for processing speed higher than one instruction per clock cycle, program counter similar to Table 2.2.
Instruction dispatch	For dispatch of instructions to the appropriate units.
Control register	Control registers associated with the control unit of the processor.
Registers emulation unit	Emulation
Register File A	Set of on-chip registers used during processing instructions in data path 1. These are named A0... A 15 and A16 ...A31. A register file is a file that associates with a unit such as ALU or FLPU.
Register File B	Set of on-chip registers used during processing instructions in data path 2. These are named A0... A 15 and A16 ...A31.
Prefetch unit	For fetching eight 32-bit instructions at each cycle.
Processing unit	Two multipliers and six arithmetical units, highly orthogonal, compiler and assembly optimizer, execution resources.
Arithmetic logical subunit	Subunit to execute arithmetical or logical instruction according to current instruction fetched at IR.
Auxiliary Logic subunit	A subunit used during subtraction. [Finds 2's complement before addition and then adds in order to subtract]
Multiplier subunit	Multiply
Floating Point processing (FLP) subunit	Subunit in C67x™ distinct from the ALU and performs FLP operations.
Assembly Optimizer	Optimizer for assembled codes
C compiler	Highly efficient compilation

Table 2.5 Additional structural units and functions of processors in TMS320C64x64™ VelociTI™ VLIW architecture extension

Structural Unit in Core	*Functions*
Packed data processing	8-bit or 16-bit data packed and processed as 32-bit data
Parallel execution MAC units	Quad 16-bit MAC/Octal 8-bit MAC [Table 2.2]
Special instructions	Broadband and image processing using VLIWs
Level 2 cache	Enhances performance of each fetch cycle
Instruction packing unit	Instructions packed as VLIW, which executes in parallel without in between halts

units of processor and showed the buses. A processor has an ALU. A processor circuit does sequential operations and a clock guides these. A processor has the program counter and stack pointer, which point to the instruction to be fetched and top of the data pushed into the stack, respectively. Certain processors have on-chip memory

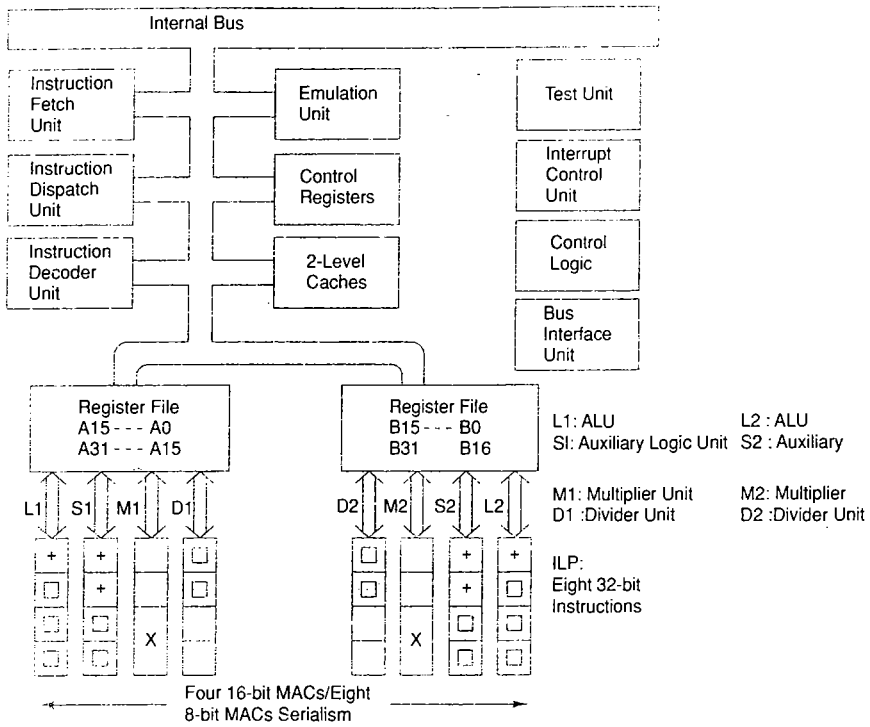

Fig. 2.21 Core and special structure units in DSP, TMS320C64x DSP
Note: Floating Point Units present in C67x

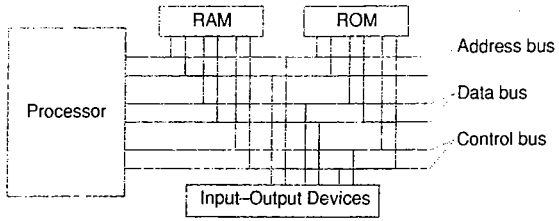

Fig. 2.22 A simple view of organization of processor, buses and memory in a system

management unit (MMU). A processor generally has general-purpose registers. Registers organize onto a common internal bus of the processor. A register is of 32, 16 or 8 bits depending on whether the ALU performs at an instance 32- or 16- or 8-bit operation.

A processor may have CISC (Complex Instruction Set Computer) or RISC (Reduced Instruction Set Computer) architecture. A CISC has the ability to process complex arithmetic and logic as well as other

instructions and processes complex data sets using fewer registers, as it provides for a large number of addressing modes. An RISC executes simpler instructions and in a single cycle per instruction. New RISC processors, such as ARM 7 and ARM9, also provide for a few most useful CISC instructions also. CISC converges to an RISC implementation because most instructions are hardwired and implement in a single clock cycle.

A processor provides for the inputs for external interrupts so that the external circuits can send the interrupt signals (Section 2.2.4). The processor may possess an internal interrupt controller (handler) to program service routine priorities and to allocate vector addresses. The internal interrupt controller is of great help in most applications.

A processor may provide for bit manipulation instructions. These instructions help in easy manipulation of bits at the ports and memory addresses. Certain processors possess FLPU and FRS units that perform floating-point operations fast. These permit higher computational capabilities in the processor; they are essential for signal processing and sophisticated control applications.

Certain processors provide for direct memory access (DMA) controller with multiple channels on chip. When there are a number of I/O devices and an I/O device needs to access a multibyte data set fast, the system memory on-chip DMA controller is of great help. Section 4.8 will describe the DMA in detail.

Table 2.6 lists the nineteen features for the CISC family of microcontrollers and microprocessors.

Table 2.6 Features in four CISC microcontroller and processor families

Capability	*Intel 8051 and Intel 8751*	*Motorola M68HC11E2 E2*	*Intel 80196KC*	*Intel Pentium*
Processor instruction cycle in microseconds (typical)	1	0.5	0.5	0.001[1]
Iinternal bus width in bits	8	8	16	64
CISC or RISC architecture	CISC	CISC	CISC	CISC with RISC feature[2]
Program counter bits with reset value	16 (0x0000)	16 |(0xFFFE)|	16 (0x2080)	32[3] (0xFFFF FFFF)
Stack pointer bits with initial reset value in case a processor defines these	8 (0x07)	16	16	32[3]
Atomic operations unit	No	No	No	No
Pipeline and super-scalar architecture	No	No	No	Yes
On-chip RAM and/or register file bytes[4]	128 and 128 RAM	512 RAM	256 and 232	No
Instruction cache	No	No	No	8 kB[5]
Data cache	No	No	No	8 kB[5]
Program memory EPROM/EEPROM	4 k	8 k	8 k	No
Program memory capacity in bytes	64 k[6]	64 k	64 k	4 GB
Data/ stack memory capacity in bytes	64 k[6]	64 k	64 k	4 GB
Main memory, Harvard or Princeton architecture (Section 2.4.2)	Harvard[6]	Princeton	Princeton	Princeton
External interrupts	2	2	2	1[7]
Bit manipulation instructions	Yes	Yes	Yes	Yes
Floating point processor	No	No	No	Yes

(*Contd*)

Capability	*Intel 8051 and Intel 8751*	*Motorola M68HC11E2 E2*	*Intel 80196KC*	*Intel Pentium*
Internal Interrupt controller	Yes	Yes	Yes	No
DMA controller channels	No	No	1 (PTS)[8]	4
On-Chip MMU	No	No	No	Yes

[1]It is maximum time in a typical Pentium 1 GHz version.
[2]Single clock-cycle hardwired implementation for most instructions implement like a RISC
[3]Stack Pointer ESP 32 bits together with the Stack Segment ES 16 bits point to physical stack address at the memory. It equals ES × 0x10000 + ESP.
[4]This is in standard version. In other versios, it may be different.
[5]This is in a typical version
[6]Program and data memory spaces are separate in Intel 8051 family members. It is common in others.
[7]Using the INTR pin and external programmable interrupt controller, up to 256 external interrupts can be handled
[8]PTS means there is a Peripheral Transactions Server providing a DMA-like feature.

Table 2.6 shows the memory addresses in hexadecimal. Thus 0x10000 means a hexadecimal memory address 10000; 0x100FF means hexadecimal memory address 100FF. The same is the convention in C. It helps in distinguishing a decimal number from hexadecimal number.

2.4.2 Memory Organization

The memory system (consisting of various units) acts as a storage receptacle for data and programs. Most systems have two types of memory—*read-only memory* (ROM) and *random-access memory* (RAM). A flash memory functions as the ROM. Examples of uses of flash are mobile phone, mobile-computer and digital camera.

Read Only Memory As its name suggests, contents of the ROM does not modify during running of computer or on power off but may be read. In general, the ROM is used to hold a program that is executed automatically by the system every time it is turned on or reset. This program is called bootstrap, or boot loader, which instructs the system to load its operating system from its hard disk or other I/O storage device. The name of this program comes from the idea that the system is "pulling itself up by its own bootstraps" by executing a program that tells it how to load its operating system. An example of ROM is as follows: A system has ROM unit(s) for the bootstrap program(s), basic input–output system (BIOS) program(s) and vector addresses of the interrupts (Section 4.4.1).

Random Access Memory Random-access memory, on the other hand, can be both read and written, and is used to hold the programs, operating system and data required by the system. For example, a mobile phone has 128 kB or 256 kB of RAM to hold the stack and temporary variables of the programs operating system and data. RAM is generally volatile, meaning that it does not retain the data stored in it when the system's power is turned off. Any data that needs to be stored while the system power is off must be written to a permanent storage device, such as flash memory or hard disk.

Addresses Memory (both RAM and ROM) is divided into a set of storage locations, each of which can hold 1-byte (8 bits) of data. The storage locations are numbered, and an assigned number is called *address*. It defines in a memory of system which location the processor wants to reference at a given instance. One of the important characteristics of a computer system is the width of the address lines (bus) it uses, which limits the amount of memory that the processor can address. Most current computers use either 32-bit or 64-bit addresses,

allowing them to access either 2^{32} or 2^{64} bytes of memory. Assume that an IBM PC has 1 MB memory (1024 × 1024 bytes). Its bootstrap program and BIOS ROM addresses are between 15×2^{16} (≡ 0xF0000) and $2^{20} - 1$ (≡ 0xFFFFF). RAM addresses are between 1×2^{16} (≡ 0x10000) and $15 \times 2^{16} - 1$ (≡ 0xEFFFF).

Random Access Model of Memory A simple model for RAM and ROM both is the random-access model of memory when all memory operations take the same amount of time independent of the address of byte or word in memory. Assume that the memory system will support two operations: load (read operation into processor from memory) and store (read operation from processor into memory). The random access model states as follows: From the memory, a data byte, a word, a double word, or a quad word may be accessed from or at any addressable location, and a similar process is used to access from all locations. There is equal access time for a read or write that is independent of a memory address location. This mode differs from another model, called serial access model.

Store and Load (Write and Read) Instructions Most high performance organizations allow more than 1-byte of memory (generally four bytes) to be loaded or stored at one time. Generally, a load or store operation operates on a quantity of data equal to the system's bus width, and the address sent to the memory system specifies the location of the lowest-addressed byte of data word(s) to be loaded or stored. Each instruction mostly has the opcode followed by operands. Store operations need two operands, a value to be stored and the address in which that the value should be stored. They place the specified value in the memory location specified by the address.

Load operations need an operand that specifies the address containing the value to be loaded and return (fetch) the contents of that memory location into their destination (register), which is specified by another operand.

Using this model, the memory can be thought of as functioning similar to a large sheet of lined paper, where each line on the page represents a 1-byte storage location. To write (store) a value into the memory, we count down from the top of the page until we reach the line specified by the address, erase the value written on the line and write in the new value. To read (load) a value, we count down from the top of the page until we reach the line specified by the address, and read the value written on that line.

Alignment of Multibyte Store and Load in a Memory Organization Some memory organization requires loads and stores to be "aligned". Assume that a 4-byte word has been aligned at address 0x000C or 0x1000, which is a multiple of 4. This simplifies the organization of the memory system as follows:

When a memory organization require loads and stores to be "aligned," it means that the address of a memory reference must be a multiple of the size of the data being loaded or stored, so a 4-byte load must have an address that is a multiple of 4, an 8-byte store must have an address that is a multiple of 8, and so on. Other systems allow unaligned loads and stores, but take significantly longer to complete such operations than aligned loads.

ARM processor memory addresses are aligned either in multiples of four or two or one byte addresses. ARM permits three data types: four bytes word or two byte half word or 1-byte word, which stores at addresses in multiple of 4 or 2 or 1, respectively.

Example 2.9

(a) Assume that a given memory organization require loads and stores to be "aligned". Then a 32-bit system loads or stores 32 bits (4 bytes) of data with each operation into the 4 bytes that start with the operation's address, so a load from location 0x424 would return a 32-bit word containing the bytes in locations 0x0424, 0x0425, 0x0426 ad 0x0427.

(b) Assume that a given organization require loads and stores to be not aligned. A 32-bit system loads or stores 32 bits (4 bytes) of data with each operation into the 4 bytes that start with the operation's address, so a load from location 0x423 would return a 32-bit word containing the bytes in location 0x0423 0x0424 0x0425 and 0x0426, as in such organizations the store or load address can be any number, not necessarily a multiple of 2 or 4.

Little Endian and Big Endian in a Memory Organization Some processor and memory organizations require little endian and other big endian aligned multiple bytes when there is store into the memory or load into the processor from memory. The ARM processor permits programming at the start and enables a programmer to define one of two possible word-alignments, little endian or big endian, at the beginning. It is important to know how organization orders the bytes written at the memory.

(a) In a little-endian system, the least-significant (smallest value) byte (8-bit) of a word (of 16 or 32-bit) is written into the lowest-addressed byte, and the other bytes are written in increasing order of significance.

(b) In a big-endian system, the byte order is reversed, with the most significant byte being written into the byte with the lowest address. The other bytes are written in decreasing order of significance.

Example 2.10

1. Two different ordering schemes are used in modem computers: *little endian* and *big endian*. Assume that a word of 32 bits is 0x90ABCDEF, and the address where the word stores when written is 0x1000. The following shows an example of how a little-endian system and a big-endian system would write a 32-bit (4-byte) data word to address 0x1000.

Little-endian system and a big-endian system

Address	*0x1000*	*0x1001*	*0x1002*	*0x1003*
Little Endian	EF	CD	AB	90
Big Endian	90	AB	CD	EF

In general, programmers do not need to know the endianness of the system they are working on, except when the same memory location is accessed using loads and stores of different lengths. For example, if a 1-byte store of 0 into location 0x1000 was performed on the 32-bit systems in Example 2.10, a subsequent 32-bit load from 0x1000 would return 0x90ABCD00 on the little-endian system and 0x00ABCDEF on the big-endian system. Endianness is often an issue when transmitting data between different computer systems, as big-endian and little-endian computer systems will interpret the same sequence of bytes as different words of data. To get around this problem, the data must be processed to convert it to the endianness of the computer that will read it.

Figures 2.10 and 11 described the memory, processor and IO units organized on the buses. It can be safely concluded that the memory organization has a tremendous impact on computer system performance and is often the limiting factor on how quickly an application executes. Both bandwidth (how much data can be loaded or stored in a given amount of time) and latency (how long a particular memory operation takes to complete) are critical to application performance.

Other important issues in memory system design include protection (preventing different programs from accessing each other's data) and how the memory system interacts with the IO system.

There may be on-chip memories as RAM and/or register files, windows, caches and ROM in a microprocessor.

The caches are the integral parts of the memory-organization within a system. The software designer should enable the use of caches by an appropriate instruction, to obtain greater performance during the run of a section of a program, while simultaneously disabling the remaining sections in order to reduce the power dissipation and minimize energy requirements. Hardware designers should select a processor with multiway cache units so that only that part of a cache unit gets activated that has the data necessary to execute a subset of instructions. This also reduces power dissipation.

Processor Memory Organization: Princeton Architecture Figure 2.23(a) shows processor and memory organization in Princeton architecture. 80x86 processors and ARM7 have Princeton architecture for main memory. Vectors, pointers, variables, program segments and memory blocks for data and stacks have different addresses in the program in Princeton memory architecture.

Processor Memory Organization: Harvard Architecture Figure 2.23(b) shows processor and memory organization in Harvard architecture. A processor having Harvard main-memory architecture has distinct address spaces, control signal(s), processor instructions, and data paths for the bytes for data and for program. (The 8051-family microcontrollers have Harvard architecture.)

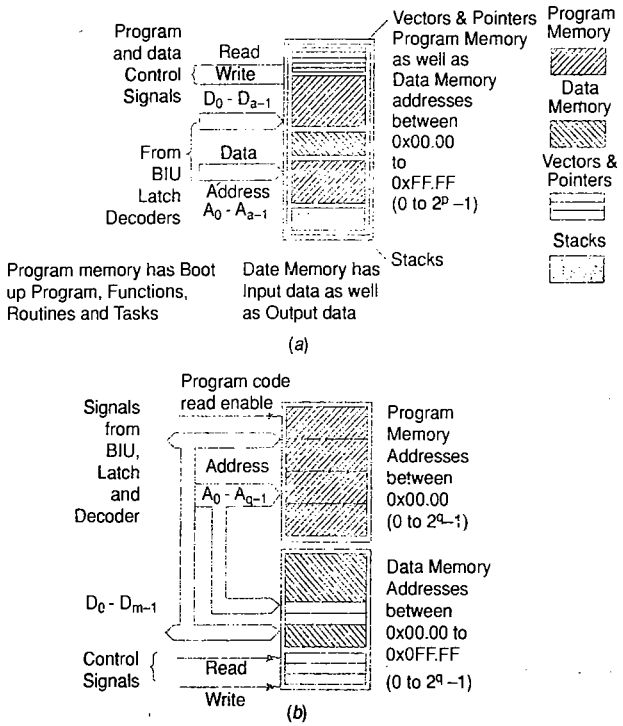

Fig. 2.23 (*a*) Processor and memory organization in Princeton architecture
(*b*) Processor and memory organization in Harvard architecture

Harvard architecture helps in handling streams of data that are required to be accessed in cases of single instruction multiple data type instructions and DSP instructions. Separate data buses ensure simultaneous accesses for instructions and data. Program segments and memory blocks for data and stacks have separate sets of addresses. Control signals and read-write instructions are also separate for accessing the program memory and data memory.

It must be remembered when coding in assembly and when organizing the main memories of certain processors, that their memory organization may be Harvard architecture. Program memory and data memory have separate set of addresses and have separate instructions for area accesses. A processor having Harvard architecture is needed for access to streams of data. Examples are (i) single instruction multiple data type instructions and (ii) DSP instructions.

For example, consider a DSP computation of the following expression in a 'Finite Impulse Response (FIR) filter'. An n-th filtered output sequence, $y_n = \Sigma(a_i.x_n - i)$, where the sum is made for i = 0, 1, 2, ..., N–1. Here i, n and N and the integers. If N = 10, then for each value of y, first one of the 10 coefficients, a_i, and one of the 10 input sequences, x, are multiplied and then the summation is done. The total computations for all 10 values of n will need 100 multiplications and 100 summations. Storing and accessing the coefficients from a separate set of memory-addresses in a separate memory will allow fast access by using a separate set of buses.

2.5 INSTRUCTION-LEVEL PARALLELISM

Several instructions can execute in parallel. Two or more instructions can execute in parallel as well as in sequence in pipelines. In the instruction level parallelism (ILP), two parallel pipelines in a processor and two instructions I_n and I_{n+1} execute in parallel at separate execution units. Figure 2.24 shows instruction-level parallelism in pentium processor.

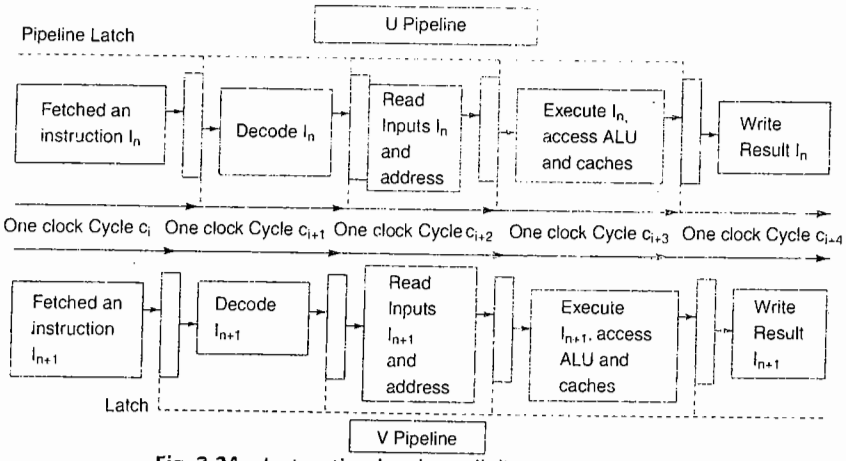

Fig. 2.24 Instruction level parallelism in a processor

2.5.1 Pipelined and Superscalar Units

High processor performance is required in many cases. For example, real-time signal processing. Pipelining and superscalar operations have now become essential. The hardware designer selects the processor as per the

required MIPS or MFLOPS performance. [A multi-processor system (Section 6.4.1) will be needed for very high performance requirements in mobile phones, digital camera, speech processing and video systems.]

Advanced processing units include instruction pipelining unit, which improves performance by processing instructions in multiple stages and the parallel units for superscalar execution, which improves performance on execution of two or more instructions in parallel execution units.

How do pipeline and superscalar units give such higher performance? Let us look at Example 2.11.

Example 2.11

Pipeline and Superscalar Execution

Step 1: Let us assume that the processor instruction cycle time is 0.02 μs (at 50 MHz operation) and that the processor executes an instruction in one clock cycle. The processor performance expected without advanced processing units will be 50 MIPS.

Step 2: Assume there is a three-stage pipeline as in ARM7. Let us, for the moment, ignore the effect of branching (called *branch penalty*). Three instructions will process in three clock cycles, but each clock cycle period can be made = 1/3 of the earlier period, as the division of processing units in stages divides the circuit also. The maximum expected performance of the processor without superscalar but with pipeline will be = 150 MIPS.

Step 3: Assume there is a two-line superscalar. Let us ignore the effects of unaligned data (*data dependency penalty*). Six instructions can process in single clock cycle with the three-stage pipeline and two superscalar units. The maximum performance will now be six times the processor cycle time, 300 MIPS.

We now explain the two terms used: branch penalty and data dependency penalty.

Branch penalty: If a branching instruction is encountered at a multistage pipeline, then the instructions executed in part at the preceding stages become redundant. These instructions have to be executed in full again later on after completion of the loop or return from a routine. The time required for re-processing these is called branch penalty.

Data dependency penalty: Assume that there are two instructions in two execution lines during a superscalar operation. Further, that one instruction depends on the data output of another. This is known as improper alignment. Thus, the two instructions are not aligned before putting them in separate lines. One instruction will now have to wait and cannot proceed further till the other instruction is executed. The waiting time is the data dependency penalty.

Superscalar processors possess hardware to extract instruction-level parallelism from sequential programs and possess hardware to efficiently take care of the collisions in execution unit and of data and control hazards.

During each cycle, the instruction issue logic of a superscalar processor examines the instructions in the sequential program to determine which instructions may be issued on that cycle. If enough instruction-level parallelism exists within a program, a superscalar processor can execute one instruction per execution unit per cycle, even if the program was originally compiled for execution on a processor that could only execute one instruction per cycle.

SHARC supports instruction level parallelism. The SHARC processor has instructions that are combined as single instruction word. [SHARC also supports memory parallelism; its processor has a modified Harvard structure called super Harvard structure (Figure 2.20). Multiple data can be fetched in a single instruction.]

ILP capability is one of the greatest advantages of superscalar processors and is the reason why virtually all high performance CPUs are superscalar processors. Superscalar processors can run programs that were originally compiled for purely sequential processors, and they can achieve better performance on these programs than processors that are incapable of exploiting the ILP.

Thus, users who work on new systems containing superscalar CPUs can install their old programs on those systems and see better performance on those programs than was possible on their old systems.

The use of high performance processor ICs and cores in embedded systems providing billion operations per second has become feasible due to the great advances in VLSI and in ILP and multi core processor design technology.

2.6 PERFORMANCE METRICS

Sophisticated embedded systems for high computing performance applications needs optimized use of resources, power, caches and memory. The following are the processor performance metrics:

1. (a) High MIPS, (b) high MFLOPS and (c) high *Dhrystone benchmark* program-based MIPS
2. Optimized compiler unit performance in the processor.

The above metrics are provided by the latest innovatively designed processors. A high-performance processor combines capabilities with optimized use of resources, power, caches, and memory.

A benchmarking program is called Dhrystone, developed in 1984 by Reinhold P. Weicker. It measures the performance of a processor for processing integers and strings (characters) both. It uses a benchmark program available in C, Pascal or Java. It benchmarks a CPU and not the performance of IO or OS calls. Dhrystones per second is the metric used to measure the number of times the program can run in a second. 1 MIPS = 1757 Dhrystone/s. [Why? VAX11/780, which executed 1 MIPS, ran the Dhrystone benchmark program 1757 times (refer to http://www.webopedia.com/ TERM/D/Dhrystone.html.)]

There is EDN Embedded Benchmark Consortium (EEMBC) [EDN is a group that publishes the International magazine EDN, which is dedicated to Embedded System information. Refer to http://www.e-insite.net/ednmag/]. EEMBC proposed five-benchmark program suites for 5 different areas of applications of embedded systems: (a) Telecommunications, (b) Consumer Electronics, (c) Automotive and Industrial Electronics, (d) Consumer Electronics, and (e) Office Automation. These program suites are also used for measuring and comparing embedded system processor performances.

Different systems require different processor performance in terms of processing speed. A hardware designer takes these into view and selects an optimum performance-giving processor.

2.7 MEMORY-TYPES, MEMORY-MAPS AND ADDRESSES

2.7.1 Memory in a System

Section 1.3.5 introduced the memory in a system. A simple credit–debit transaction card may require just 2 kB of memory. On the other hand, a smart card for secure transactions when embedding a Java program for cryptographic functions may require 32 kB (typical value) memory. A complex embedded system may need huge memory.

The following subsections explain and look at certain important aspects of the memory. The various memory are described from the point of view of an embedded systems hardware or software designer.

ROM: its Uses, Forms and Variants ROM non-volatility is a most important asset and it is extremely useful to embed codes and data in a system. ROM is a loosely used term. For a hardware designer, it may mean masked ROM, PROM, OTP-ROM, EPROM and EEPROM. In a strict sense, ROM means a masked ROM made at a foundry from the programmer's ROM image file (Section 1.4.1). ROM that embeds the software or an application logic circuit is in one of the three forms: masked ROM, PROM and EPROM. When ROM is to be programmed during runtime and is to hold the processed result, either an EEPROM or flash memory is used.

(i) Masked ROM A masked ROM is built from a circuit that has r inputs (A_0 to A_{r-1}) and 8 outputs (D_0 to D_7). [Byte storing at an address is most common.] The circuit for masked ROM is one of a set of 2^r combinational circuits. Appropriate masking gives the desired set of outputs at each combinational circuit. Certain links fuse and others that are masked do not fuse. [A combination circuit is a circuit made up of logic gates with a distinct set of output logic states during distinct input logic states. It has a distinct truth table for r inputs x 8 outputs. As soon as the inputs change (or withdraw), the output also changes in this circuit.].

The embedded software designer (after thorough testing and debugging) provides to a manufacturing foundry a file having a table of desired output bits for the various combinations of the input address bits. A program called *locator* creates this table. The manufacturer prepares the programming masks and then programs the ROM at a foundry. This ROM is returned to the system manufacturer.

Normally, one time masking charge could be very high. Generally, therefore, a system manufacturer will place the order, and the manufacturing foundry will accept the order for a minimum of 1000 pieces. The ROM is a cost effective solution to a bulk user of ROMs for the manufacture of embedded systems. An embedded system manufacturer using a masked ROM does not have to use a device programmer (ROM burner) each time a system ROM is made using EPROM or PROM or flash.

(ii) EPROM, E^2PROM and OTP ROM Special versions of ROM can be programmed at the designer's or manufacturer's site for an embedded system with the help of a device programmer. One version is EPROM. It is an ultraviolet ray *erasable* and device programmer *Programmable Read Only Memory*. Erasing the device means restoring 1 at each bit in the cell arrays at each ROM address. Another version is E^2PROM (EEPROM). It is an *Electrically Erasable*, and *Programmable Read Only Memory*. Examples of EPROM and EEPROM are 2732, a 4 kB EPROM, 28F256, a 32 kB EEPROM and 28F001 is 512 K × 16-bit EEPROM.

EEPROM erasing during an application-program run is done by sending all eight data bus bits as 1s for the write in the presence of inputs of V_{pp}, called programming voltage and short duration write pulse. Sending 1s and 0s in the byte by a write instruction results in EEPROM programming during a program run. Erasing of a byte must precede the write. The processor with the system program can do erasing and writing, as it is similar to the writing in a RAM. What then, is the difference between the EEPROM and RAM? The difference is that in RAM, the read and write timing cycles are identical. Here, the write cycle has to be longer than in case of RAM, and it must succeed the erase of the byte by writing 0xFF. Further, an addition voltage V_{pp} signal is needed when erase and write occurs to the EEPROM The number of times an EEPROM can be written is one million times plus. There is no limit for RAM, and a practically infinite number of writes is possible without first writing 1s in RAM.

Flash memory is a form of EEPROM in which a sector of bytes can be erased in a flash (very short duration corresponding to a single clock cycle). [Lately flashes of even ~3 V form and of capacity 2, 4 and 8 GB have become available even in card or stick form, which enables insertion in camera or pocket computer.] A sector can be from 256 B to 16 kB. The advantage over EEPROM is that the erasing of many bytes simultaneously saves time in each erase cycle that precedes the write cycles. The disadvantage is that once a sector is erased, each byte writes into it again one by one, and that takes too long a time. A new version of flash is **boot back flash**. A sector is reserved to store once only at the time of first boot. Later on it is protected from any further erase. In other words, it has an OTP sector also that can be used to store ROM images like in a ROM. Nowadays, flash has replaced EPROM in the systems.

PROM (an OTP ROM, a one time device programmer) is another form. A PROM once written is not erasable.

(iii) Uses of ROM or EEPROM or Flash Figure 1.5 showed what a ROM embeds—program codes for various tasks, interrupt service routines, operating system kernel, initialization (bootstrap program and data) and the standard data or table or constant strings.

An EEPROM is usable by erasing over one million times. It can be erased during runtime itself. Flash memory is usable about 10,000 times for repeated erasing followed by programming during the runtime. The PROM is written only once by a device programmer or the first system run.

Three examples of EEPROM memory applications are as follows. **(i)** Storing current date and time in a machine. **(ii)** Storing port statuses. **(iii)** Storing driving, malfunctions and failure history in an automobile for use by mechanics later on.

Three examples of flash memory applications are as follows. **(i)** Storing pictures in a digital camera. **(ii)** Storing voice compressed form in a voice recorder. [Recall of prerecorded message in a phone.] **(iii)** Storing messages and contacts in a mobile phone.

Examples of use of an OTP ROM are as follows. **(i)** Smart card identity number and user's personal information. **(ii)** Storing boot programs and initial data like a pictogram displaying a seal or monogram. **(iii)** ATM card or credit card or identity card. Once the various details are written at the bank and handed over to the account holder, there is no modification possible in the embedded PROM at the card. Just as a paper holds information permanently once written or printed, so also does a PROM.

A flash or PROM or ROM is not only used for program and data storage, but also for obtaining the preprogrammed logic outputs and output sequences for the given sets and sequences of inputs. [Inputs are given analogous to an address signal by the processor and outputs are obtained analogous to those obtained during a processor read cycle.] Assume that there are 8 inputs ($r = 8$). The truth table for it will have 256 combinations. 8 x 8 ROM can be programmed to generate 256 sets of 8-bit outputs for each combination. Examples of applications of preprogrammed logic outputs are as follows.

1. Used to hold language-specific bits for the fonts corresponding to each character in a printer.
2. Used to hold the image bits for a display. A pictogram generates from these bits. A ROM is used in a display circuit. It stores the bytes for the full bit-image corresponding to the pixels for a pictogram. Sequential changes at the inputs repeatedly generate the full pictogram.
3. In a CISC a control ROM at a micro-programmed unit is used. It stores sets of microinstructions for each processor instruction. Each set of microinstructions is stored in a sequence such that it specifies a set of signals for the various fetch and executing unit during execution of an instruction fetched from the memory.

RAM A system designer considers RAM devices of eight forms. These forms are 'SRAM', 'DRAM', 'NVRAM', 'EDO RAM', 'SDRAM', 'RDRAM', Parameterized Distributed RAM and Parameterized Block RAM.

(i) Uses of RAM RAM stores the variables during a program run and stores the stack (Section 1.3.5). It stores input and output buffers, for example, of speech or image. It can also store the application program and data when the ROM image is stored in a compressed format in an embedded system and decompression is done before the actual run of the system.

1. SRAM is used most commonly for designning caches and in embedded systems and microcontrollers.
2. DRAM is used mostly in high performance computers or high memory density systems.
3. EDO RAM is used in systems with buses to the devices when operating with clock rates up to 100 MHz; a zero-wait state is needed between two fetches, and there is single-cycle read or write.

4. SDRAM synchronizes the read operations and keeps the next word ready while the previous one is being fetched. This device is used when buses can fetch or send to the processor up to speed of 1 GHz.
5. RDRAM accesses in bursts the four successive words in a single fetch and thus gives above 1 GHz performance of the system.
6. Parameterized distributed RAM is the RAM distributes in various system subunits. IO buffers and transceiver subunits can have a slice of RAM each and the system stack can be at another slice. Distribution provides buffering of memory at the subunits before they are fetched and processed by the processor. It facilitates faster inputs from the IO devices than the processor system buses access the IOs using system memory.
7. Parameterised block RAM is used when a specific block of the RAM is dedicated for use by a subunit only, for example, MAC unit. A parameterized block RAM is used when an access by the system or IO or internal bus is slow compared to the processing speed of a subunit.

Different types of memory in varying capacities are available for use as per requirement. (1) Masked ROM or EPROM or flash stores the embedded software (ROM image). Masked ROM is for bulk manufacturing. (2) EPROM or EEPROM is used for testing and design stages. (3) EEPROM is used to store the results during the system program runtime. It is erased byte-by-byte and written during the system-run. It is useful to store modifiable bytes, for example, the runtime system status, time and date and telephone number. (4) Flash stores the results byte by byte during a system run after a full sector erase. (5) Flash is thus very useful when a processed image or voice is to be stored or a data set or system configuration data is to be stored, which can be upgraded as and when required. For example, a new image (after compressing and processing) can be stored and the old one erased from a sector in a single instruction cycle. (6) Boot block flash also has an OPT sector(s) to store the boot program and initial data or permanent system configuration data. It serves by storing the ROM image or its part in the OTP sector(s) and, at the same time, serves by storing as a flash in other sectors. (7) RAM is mostly used in SRAM form in a system. (8) Sophisticated systems use RAM in the form of a DRAM, EDO RAM, SDRAM or RDRAM. (9) Parameterized distributed RAM is used when the IO devices and subunits require a memory buffer and a fast write by another system. (10) Subunits like MAC when operating at fast speed use separate blocks of RAM.

2.7.2 Address Allocations in Memory

Figure 2.23 (a) showed memory addresses needed in the case of Princeton architecture in the system. Figure 2.23 (b) showed memory addresses needed in the case of Harvard architecture.

A *system memory allocation-map* is not only a reflection of addresses available to the memory blocks, and the program segments and addresses available to IO devices, but also reflects a description of the memory and IO devices in the system hardware. It maps guides to the actual presence of the memory at the various units, EPROM, PROM, ROM, EEPROM, flash Memory, SRAM (static RAM), DRAM (dynamic RAM) and IO devices. It reflects memory allocation for the programs, and data and IO operations by a *locator* program. It shows the memory blocks and ports (devices) at these addresses.

System IO devices map may be designed separately. This not only reflects the actual presence of the IO devices, but also guides the available addresses of the various device registers and port-data. [An example of a device is a timer. IO devices are the peripheral units of the system.]

The following are examples that describe memory allocation maps using the *locator*.

Example 2.12

Consider a memory map for an exemplary embedded system—a smart card needing a 2 kB memory, a 256 B RAM mainly for the stacks, EEPROM 512 B for storing the balance amount under credit or debit and the previous transaction records on the card. The memory locator or linker script program for this system to define a memory allocation map [Figure 2.25(a)] is as follows.

```
1. Memory
2. {ram : ORIGIN = 0x10000, LENGTH = 256
3. eeprom : ORIGIN = 0x20000, LENGTH = 512
4. rom : ORIGIN = 0x00000, LENGTH = 2K
5. }
```

Example 2.13

Consider a Java embedded card with software for encrypting and deciphering transactions. Assume that the system needs 32 kB ROM, RAM of 4 kB, and EEPROM 512B for storing not only the balance amount under credit or debit but also the cryptographic keys and previous transaction records on the card. So the memory locator or linker script program for this system defines memory map [Figure 2.25(b)] as follows.

Memory

```
1. { ram : ORIGIN = 0x10000, LENGTH = 4K
2.   eeprom : ORIGIN = 0x20000, LENGTH = 512
3.   rom : ORIGIN = 0x00000, LENGTH = 32K
4. }
```

One can also make the following important observation from Examples 2.12 and 2.13. There are memory address gaps between the origin of ROM, RAM and EEPROM in spite of the very small lengths of available memory. This gap is due to a design feature: the designer provides for expansion of these memories in future so no change will be needed in the interfacing decoder circuit between the memory and processor. Further, its software program has to make minimal changes. The changes will only be in length. This is because when there is no gap the origin will also change. This feature ensures that any future changes in the program code sizes and data sizes will not need change in the locator codes. One feature of a locator is also that it does not relocate the addresses of the special purpose ports that are dedicated to a particular IO task or dedicated to the device driver read and write operations.

The final step of the design process in an embedded system is that the bytes locate at the ROM from the image for the bootstrap (reset) program and data, the initialization data as well as the following standard data or table or constant strings device driver data and programs, the program codes for various tasks, interrupt service routines and operating system kernel. [The bootstrap program consists of the instructions that are executed on system reset. The bootstrap data example is for stack pointer initialization. The initialization data may be for defining initial state and system parameters. The constant strings may be for initial screen display.] There is a shadow segment in the ROM. The shadow segment has the initialization data, constant string, and the start-up codes that are copied into the RAM by a shadow segment copy program at system boot up. When a start-up code (booting) program is executed, a copy of the shadow segment from the ROM is generated in the RAM. The RAM also holds the data (intermediate and output data) and stack. *A compressed program format locates at the ROM in case of large ROM image is required for the system program. This is because decompression program plus compressed image will need less memory than the large ROM image. The start-up code task is to generate the decompressed program codes and store them into the RAM before system starts other programs. The processor executes all other programs subsequently by fetches from the RAM.*

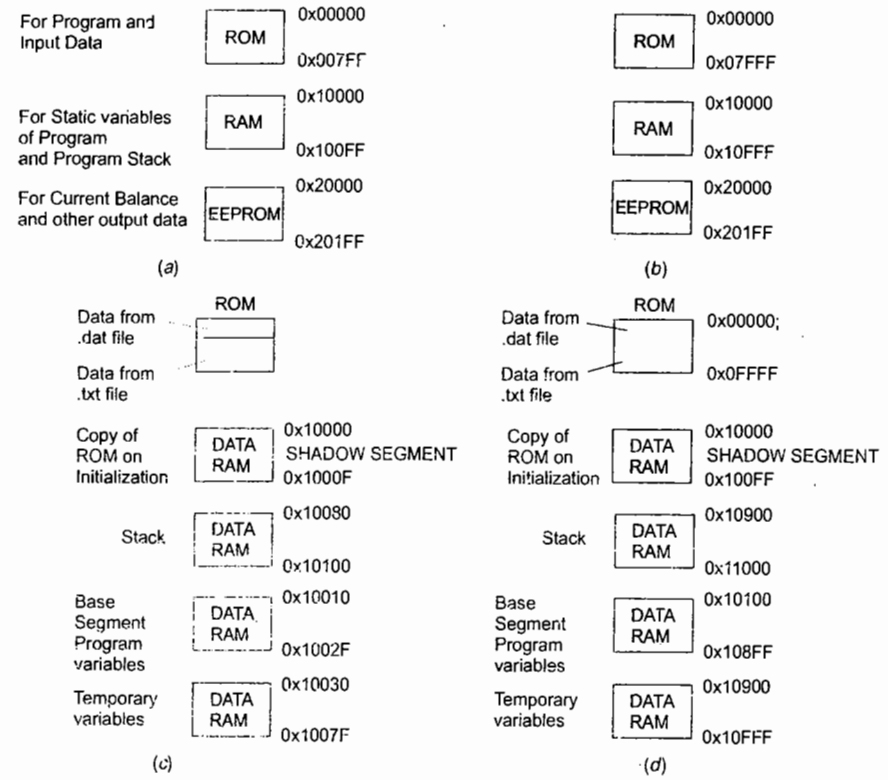

Fig. 2.25 Examples of Memory maps in embedded systems—(*a*) smart card needing 2 kB memory (*b*) Java embedded card with software for encrypting and deciphering transactions (*c*) memory map sections in a smart card (*d*) memory map sections in another smart card

A processor may have predefined memory locations for the initialization boot record. For example, in 80960, boot record consists of 12 words. The record is stored at ROM addresses, 0xFFFFFF00 to 0xFFFFFF2C.

Example 2.14

Consider memory map for an exemplary card in which there are sections at the memory allocation map. Consider its description in a *locator* program. The sections for an exemplary embedded system, smart card memory map [Figure 2.25(c)] may be defined as follows.

```
1. SECTIONS
```

```
2. {/* Stack Top Location for 128 B RAM*/
3. _TopOfStack = 0x10100;
4. /* Bottom of Heap */
5. _BottomOfHeap = 0x10080;
6. text rom :
7. {/* Debit-credit card program instructions are at the text file
   named here*/
8. * (----.txt)
9. }
10. data ram :
11. {
12. /* Shadow Segment for 16 byte of Initialised Data at RAM for a copy
   from ROM from */
13. _DataStart = 0x10000;
14. /* Debit-credit card shadow segment data at the data file named
   here*/
15. * (----.data)
16. _DataEnd = 0x1000F;
17. }
18. /* Command for copy into the RAM */
19. > rom
20. bss :
21. {
22. /* Base Segment for 32 byte of Program Variables Data at
   RAM */
23. _bssStart = 0x10010;
24. /* Smart card base segment data at the base segment data
   file named here*/
25. * (----.bss)
26. _bssEnd = 0x1002F;
27. }
28. }
```

Example 2.15

Consider another memory map [Figure 2.25(d)] for another exemplary card. The locator specifies different map sections as follows:

SECTIONS

```
1. {/* Stack Top Location for 4 kB RAM*/
2. _TopOfStack = 0x11000;
3. /* Bottom of Heap */
```



```
4. _BottomOfHeap = 0x10900;
5. text rom :
6. {/* Encrypting Java Card program instructions are at
   the text file named here*/
7. a. (----.txt)
8. data ram :
9. {
10. /* Shadow Segment for 256 bytes of Initialized Data at RAM for a copy
   from ROM from */
11. _DataStart = 0x10000;
12. /* The card shadow segment data at the data file named here*/
        a. (----.data)
13. _DataEnd = 0x100FF;
14. }
    /* Command for copy into the RAM */
15. a. rom
16. bss :
17. {
18. /* Base Segment for 2 kB Program Variables Data at RAM */
19. _bssStart = 0x10100;
20. /* Java card base segment data at the base segment data file
   named here*/
21. a. (----.bss)
22. _bssEnd = 0x108FF;
23. }
24. }
```

The memory map that includes the device IO addresses is designed after appropriate address allocations of the pointers, vectors, data sets and data structures. If the main memory is of Harvard architecture, the program memory map will be separate. For example, 8051 reads from the program memory by a separate set of instructions (input–output instructions).

2.8 PROCESSOR SELECTION

A hardware designer must take into account following processor-specific features:

1. A processor, which can operate at higher clock speed, processes more instructions per second.
2. A processor gives high computing performance when there exist (a) Pipeline(s) and superscalar architectures, (b) pre-fetch cache unit, caches, and register-files and MMU and (c) RISC architecture.
3. A processor with register-windows provides fast context switching in a multitasking system.
4. A power-efficient embedded system requires a processor that has programmable auto-shut down feature for its units and programmability for disabling use of caches when the processing need for a function

or instruction set is not constrained by limit on execution deadline. Processor uses Stop, Sleep and Wait instructions, and special cache design.

5. A processor that has a burst mode accesses external memories fast, reads fast and writes fast.
6. A processor with an atomic operation unit provides hardware solution to shared data problems when designing embedded software, else special programming skill and efforts are to be made when program uses shared variables and data buffers among multiple tasks.
7. When coding in assembly language or designing compiler or locator, data may store in big-endian mode in a system and the lower order bytes store at higher address: for example, in Motorola processors. Data may also store in little-endian mode in a system. Lower order bytes store at lower addresses and vice versa: for example, in Intel processors. A processor may also be *configure* at the initial program stage big-endian or little-endian storage of words: for example, the ARM processors.

The StrongArm family processors from *Intel* and TigerSHARC from *Analog Devices* have high power efficiency features.

The processor selection processes can be understood by considering four representative cases. Firstly a design-table similar to Table 2.7 is built. Then a processor having the required structural units and capable of giving the desired processor performance in system is chosen.

1. *Case 1:* Systems in which processor instruction cycle time ~ 1 μs and on-chip devices and memory can suffice. Examples are automatic chocolate vending machine, 56 kbps modem, robots, data acquisition systems like an ECG recorder or weather recorder or multipoint temperature and pressure recorder and real-time robotic controller.
2. *Case 2:* Systems in which processor instruction cycle time ~ 10 to 40 ns required, on-chip devices and memory do not suffice and medium processor performance is required. Examples are 2 Mbps router, image processing, voicedata acquisition, voice compression, video decompression, adaptive cruise control system with string stability and network gateway.
3. *Case 3:* Systems in which instruction cycle times of 5 to 10 ns required and high MIPS or MFLOPS performance is needed. Examples are multiport 100 Mbps network transceiver, fast 100 Mbps switches, routers, multichannel fast encryptions and decryptions systems.
4. *Case 4:* Systems in which instruction cycle time of even 1-ns does not suffice and multi-processor system is required along with use of the floating point and MAC units. Examples are voice processing, video processing, realtime audio or video processing and mobile phone systems.

Different systems require different processor features. A hardware designer takes these into view and selects an optimum performance-giving processor.

2.8.1 Microcontroller Selection

There are numerous versions of 8051. Additional devices and units are provided in these versions. A version and microcontroller is selected for embedded system design as per the application as well as its cost.

1. Embedded system in an automobile, for example, requires a CAN bus (Section 3.10.2). Then a version with CAN bus controller is selected.
2. An 8051 enhancement 8052 has an additional timer.
3. Philips P83C528 has I^2C serial bus (Section 3.10.1).
4. 8051 family member 83C152JA (and its sister JB, JC and JD microcontrollers) has two direct memory access (DMA) channels on-chip. (Section 4.8) The 80196KC has a PTS (Peripheral Transactions Server) that supports DMA functions. [Only single and bulk transfer modes are supported, not the burst transfer mode.] When a system requires direct transfer to memory from external systems, the DMA controller, improves the system performance by providing for a separate processing unit for the data transfers from and to the peripherals.

Table 2.7 Essential processor capabilities in four exemplary set of systems

Processor capability Required	*Case 1: Automatic Chocolate Vending Machine, Data Acquisition System, Real time Robotic Control*	*Case 2: Voice data acquisition, Voice-data Compression, Video Compression, Adaptive Cruise Control System with String Stability, Network Gateway*	*Case 3: Multi-port Network Transceiver, Fast Switches, Routers, Multi channel Fast Encryptions and decryptions*	*Case 4: Voice Processor, Video processing and Mobile Phone Systems*
Required processor	Microcontroller	Microprocessor	Multiprocessor System	Microprocessor +DSP based Multiprocessor System
Processor instruction cycle in μs (typical)	~ 0.5 to 1	0.01 – 0.04	0.0005 – 0.001	0.001 – 0.0005
Processor performance	Low suffices	Medium to high	High	Very High
Internal bus width in bits	8	32	32	64
CISC or RISC architecture	Any	RISC	RISC	RISC
Program counter and stack pointers	16	32	32	32
Stack at external or internal memory	External	External or internal	Internal	Internal
On-chip atomic operations unit	–	–	Yes[1]	–
Pipelined and super-scalar and pipelined architecture	No	Yes	Yes	Yes
Off-chip RAM in view of excessive RAM needs	No, on-chip suffices	Yes	Yes	Yes
On-chip register windows and files due to fast context switching needs	No	Yes	Yes	Yes
Interrupts handler internal in micro-controller or external to processor	Internal microcontroller	External	External	External

(Contd)

Processor capability **Required**	*Case 1: Automatic Chocolate Vending Machine, Data Acquisition System, Real time Robotic Control*	*Case 2: Voice data acquisition, Voice-data Compression, Video Compression, Adaptive Cruise Control System with String Stability, Network Gateway*	*Case 3: Multi-port Network Transceiver, Fast Switches, Routers, Multi channel Fast Encryptions and decryptions*	*Case 4: Voice Processor, Video processing and Mobile Phone Systems*
Instruction and data caches and MMU	No	Yes	Yes	Yes
On-chip memory flash or EPROM	Yes, on-chip suffices	No, on-chip does not suffice	No, on-chip does not suffice	No, on-chip does not suffice
External interrupts	1 to 16	1-2	128-256	16-32
Bit manipulation instructions	Used	Heavily used	Heavily used	Heavily used
Floating point processor	No	Yes	No	Yes
Streams of data requiring Harvard main memory architecture	No	Mostly Not necessary	May be Yes	Invariably Yes
DMA controller channels	No	No	Yes	May be Yes
Exemplary processor family	8051, 68HC11 or 12 or 16, 80196, PIC16F84	80x86, 80860, 80960	ARM7, Sunspare	ARM9, TMS family DSPs, PowerPC

[1]Needed when multiple ports and multichannel operations need data sharing.

Example 2.16 [illegible]

1. A robotic system motor needs signalling at the rate above 50 to 100 ms. Hence there is enough time available for signalling and real-time control of multiple motors at the robot when we use a processor with instruction cycle time ~1 μs.
2. The processor speed need not be very high and performance needed is much below 1 MIPS. So no caches and advanced processing units like pipeline and superscalar processing are required.
3. A four-coil stepper motor needs only a 4-bit input and a DC motor needs a 1-bit pulse width modulated output. Therefore an 8-bit processor suffices.
4. Frequent accesses and bit manipulations at IO ports are needed. CISC architecture therefore suffices.
5. The program can fit in 4 kB or 8 kB of internal ROM on-chip. Stack sizes needed in the program are small so that can be stacked in an on-chip 256 or 512-byte RAM. A microcontroller is thus needed. No floating-point unit is needed.

Microcontrollers appropriate for the above case are 8051, 68HC11, 68HC12, 68HC16 or 80196. Microcontrollers 68HC12 and 68HC16 can be used due to availability of large number of ports. The 68HC12's instruction cycle and clock cycle time equals 0.125 μs. Number of ports equals 12 in 68HC12. Therefore, 6 or more degree of freedom robot with 6 or more motors can be driven directly through these ports. STOP and WAIT instructions in the processor save power when the robot is at rest!]

Example 2.17 *Case Study of Voice Data Compression System*

1. Voice signals are pulse-code modulated. The rate at which bits are generated is 64 kbps. A suitable algorithm can process the data compression of these bits with an instruction cycle time of ~ 0.01 to 0.04 μs (100 to 25 MHz) when the processor uses advanced processing units and caches.
2. Let us assume that the processor instruction cycle time is 0.02 μs (50 MHz). With a three-stage pipeline and two-line superscalar architecture, the highest performance will be 300 MIPS. [Refer to Example 2.11 for an understanding of the computations of MIPS]. It suffices for not only for voice but also for video compression.
3. Frequent accesses and complex instructions may not be needed.
4. The program cannot fit in 4 kB or 8 kB of internal ROM on-chip, and stack sizes needed in the program are big. Instead large ROM and RAM as well as caches are needed.
5. No floating-point is needed as mostly the bit manipulation instructions are processed during compression.

Exemplary processors that are appropriate for the above case are 80x86 and ARM family processors.

Example 2.18 [illegible]

1. Transfer rates of 100 MHz plus are needed in fast switches on a network. Assuming 10 instructions per switching and transceiver action, instruction cycle time is ~ 0.001 plus. A multiprocessor system is needed for GHz transfer rates.
2. Let us assume that the processor instruction cycle time is 0.01 μs (100 MHz). With a five-stage pipeline and two-line superscalar architecture, the highest performance will be 1000 MIPS. [Example 2.11]. Multiprocessor system is thus needed for 1000 MHz plus switches.
3. The processor should have RISC architecture for single cycle instruction processing at each stage and line.
4. ROM and RAM as well as caches are required.
5. No floating-point is needed as mostly the bits are processed for IOs.

Exemplary processors that are appropriate for the above case are ARM7, ARM9 and Pentium.

Example 2.19 *Real-time video processing*

1. Real-time video processing requires fast compression of an image needing use of DSP. Many real-time tasks have to be processed: for instance, scaling and rotation of images, corrections for shadow, colour and hue, image sharpening and filter functions. In such cases, a multiprocessor system with DSP(s) and that has the best processing performances is required.

Exemplary processors that are appropriate for multiprocessor system are ARM9 integrated with TMS family DSP(s) or ARM11 or TigerSHARC.

2.9 MEMORY SELECTION

Once the software designer's coding is over and the ROM image file is ready, the hardware designer is faced with the questions of what type of memory and what size of each should be used. First a design-table, as in Table 2.8, is built. The memory having the required features and address space is chosen. Following are the case studies. The actual memory requirement is known only after coding as per the design functions and specifications. ROM and RAM allocations for various segments, data sets and structures will be available from the software design. However, a prior estimate of the memory type and size requirements can be made. [Remember, the memory are available as: 1 kB, 4 kB, 16 kB, 32 kB, 64 kB, 128 kB, 256 kB, 512 kB and 1 MB. Therefore, when 92 kB of memory is needed, then a device of 128 kB is selected.]

Example 2.20

(a) Case Study of an Automatic Washing machine

Consider an automatic washing machine system. Assume that machine is not saving the pictures and graphics. (a) An EEPROM's first byte is required to store the state (wash, rinse cycle 1, rinse cycle 2 and drying) that has been completed. The second byte is required to store the time in minutes already spent at the current stage. The third byte is needed to store the status of the user set buttons. Thus a 128 B EEPROM at best should suffice in microcontroller. (b) Embedded software can be within 4 kB ROM at the microcontroller. (c) RAM is needed only for a few variables and stacks. An internal RAM of 128 B should suffice. (d) Therefore, no external memory is required with the system when using a microcontroller.

(b) Case Study of a Robotic

Consider a robotic system. (a) EEPROM bytes are required to store the rest status of each degree of freedom. Thus 512 B EEPROM in the microcontroller at best should suffice. (b) Embedded software can be within 32 kB ROM in the microcontroller. (c) RAM need is only for the variables and only one stack is needed for the return address of the subroutine calls. Internal RAM of 512 B should suffice. Therefore, no external memory is required with the system when using a microcontroller.

Example 2.21

(a) Case Study of the Data Acquisition Systems for the sixteen-parameter channels and voice/image processing during acquisition

Consider a data acquisition system. Assume that there are sixteen channels and at each channel 4 B of data store every minute. (a) Bytes are to be stored in flash memory. Assume that the results are stored in flash memory for a day before it is printed or transferred to a computer. Thus 92 kB is the data acquired per day. A 128 kB flash memory will thus suffice. (b) Embedded software can be within 8 kB ROM in the microcontroller. (c) RAM is needed only for the variables and only one stack is needed for the return address of the subroutine calls. An internal RAM of 512 B will suffice. (d) Intermediate calculations are needed for storing ADC results in the proper format. Unit conversion functions need to be calculated, which may necessitate a RAM of about 4 kB to 8 kB. (e) Therefore, a microcontroller with 8 kB EPROM and 512 B RAM is required, and an external flash (or 5 V EEPROM) of 128 kB and external RAM of 4 to 64 kB are required with the system. For acquiring image or voice data on-line, RAM buffer requirement can be 512 MB.

(b) Case Study of the Data Acquisition Systems for the ECG waveforms

Consider another data acquisition system, which is used for recording the ECG waveforms. Let each waveform be recorded at 256 points. A 64 kB flash will be required for 256 patient records.

Table 2.8 Required memory in four exemple of systems

Memory Required	*Case 1: Automatic Chocolate Vending Machine or Real time Robotic Control system*	*Case 2: Data Acquisition System*	*Case 3: Multi-port Network Transceiver, Fast Switches, Routers, or Multi channel Fast or Encryption and decryption system*	*Case 4: Voice Processor or Video processing or Mobile Phone or Pocket PC System*	*Case 5: Digital Camera or Voice Recorder System*
Processor used	Micro-controller	Micro-controller	Multiprocessor system	Microprocessor + DSP-based Multiprocessor system	Micro-processor
Internal ROM or EPROM	4 to 32 kB	8 kB	–	–	–
Internal EEPROM	256 to 512 B	256 to 512 B	–	–	–
Internal RAM	256 to 512 B	256 to 512 B	–	–	–
ROM or EPROM device	No	No	64 kB	64 MB	64 kB
EEPROM or Flash device[1]	No	64 to 128 kB	512 B	32 kB to 256 kB 2 to 8 GB memory stick	Flash 16 MB to 8 GB memory stick
RAM device	No	64 kB to 512 MB	64 kB to 512 MB	8 MB	1 MB
Parameterised distributed RAM	No	No	Yes for IO buffers. 4 kB per channel		–
Parameterised Block RAM	No	No		Yes for MAC unit, Dialing IO unit	–

Note: [1]Flash with a boot block can be used to store the protected part of the boot program in its OTP sector(s).

Example 2.22 Case Study of a multichannel Fast Encryption cum decryption Transceiver Systems

1. Consider a system with multiple channels. There are encrypted inputs at each channel. These are decrypted for retransmission to other systems.
2. EEPROM is required for configuring ports and storing their statuses. Assume 16 channels. 512 kB will suffice for a 16 B need per channel.
3. Encryption and decryption algorithms can be in 64 kB ROM.

4. Multichannel data buffers are required before the caches process the algorithms. Therefore, 1 MB to 512 MB RAM may be required.
5. IO buffer storage of 4 kB per channel is needed. If a parameterised distributed RAM is employed at each channel, the system performance will be increased.
6. The system will thus need the following memory: 64 kB ROM, 512 B EEPROM, 1 MB RAM and 4 kB per channel distributed parameterised RAM.

Example 2.23 *Case Study of a Mobile Phone system*

1. As voice compression–decompression and encryption–decryption algorithms and DSP processing algorithms are required, the ROM image will be large. Assume it is can be taken as 64 MB. Now if the ROM image is stored in a compressed format, a boot-up program first runs a decompression program. The decompressed program and data first load at RAM and then the application program runs from there. The RAM is obviously of bigger size in these systems. ROM can be reduced as per compression factor.
2. A large RAM is also needed. It can be taken as 8 MB for storing the decompressed program and data and for the data buffers.
3. The phone memory for entering important telephone numbers can be in a 16 kB flash or EEPROM. A flash of 64 kB can be taken for recording messages. MMS pictures. A memory stick using SDIO port of 2 GB or 8 GB is required for recording songs and videos.
4. Parameterised block RAM at MAC subunit and other subunits will improve system performance.
5. The system will thus require memory of 1 MB ROM, 16 kB EEPROM, 16 kB Flash, 1 MB RAM and block RAM at the subunits.

Example 2.24 *Case Study of a Digital Camera system*

1. Assume a low-resolution uncoloured digital camera system. Images are to be recorded GIF (graphic image format) compressed format. (a) Assume that an image has a Quarter-CIF (Common Intermediate Format) of 144 x 176 pixels. Then, 25,344 pixels are to be stored per image. Assume that compression reduces the image by a factor of 8 then 3 kB per image will be needed. Flash required will be 0.2 M for 64 camera images. Therefore, a 256 kB flash will be required. (b) A 64 B digital uncoloured images camera system will thus be estimated to need memory of 64 kB ROM, 256 kB flash and 1 MB RAM. High resolution 6 M pixel digital camera need 16 MB 256 MB flash and 2 to 8 GB memory stick.
2. Assume a voice recording system. 64 kbps are required, assuming an 8-bit pulse code modulation of the voice signals. [Average frequency is taken as 8 kHz.] Assume voice data compression factor of 8, 1 kB flash is required per second. A is MB flash 4 required for each hour of recording.
3. Since voice compression–decompression algorithms have to be processed, the ROM image will be large. It can be taken as 1 MB. Using compression techniques, a 64 kB ROM can store the ROM image.
4. The RAM needed is large for storing the decompressed program. It can therefore be estimated as 1 MB.
5. A one-hour voice recorder system is estimated to require memory or 64 kB ROM, 4 MB flash and 1 MB RAM.

§ *Simple systems like automatic chocolate vending machines or robots needs no external memory. The designer selects a microcontroller that has an on-chip memory required by the system. The data acquisition system needs EEPROM or flash. A mobile phone or pocket PC or digital camera system needs 1 MB plus RAM device and 64 kB to 256 kB internal flash device plus memory stick. Image or voice or video recording systems require a large flash memory in the form of memory stick of 2 GB to 8 GB.*

Summary

- 8051 microcontroller has Harvard architecture for memory, has special function registers, internal RAM and ROM or flash. It has two timers, and SI interface for the half duplex synchronous and full duplex UART communication.
- Bus signals interface the processor, memory and devices. The interface circuit takes into account the timing diagram with reference to processor clock output. The circuit uses the processor control, address and data bus signals and takes into account the timing diagram for the bus signals. A PAL-, GAL- or FPGA-based circuit, called glue circuit, provides a single-core or chip solution for the latches, decoders, multiplexers, demultiplexers and other necessary interfacing circuits.
- The buses interface to stepper motor, LCD controller, A/D, D/A circuits using ports and appropriate interface circuit.
- An embedded system hardware designer must select an appropriate processor, appropriate set of memories for the system and design an appropriate interfacing circuit between the processor, memories and IO devices. This is done after taking into account the various available processors, structural units and architecture, memory types, sizes and speeds, bus signals and timing diagrams.
- The structural units of a processor that interconnect through a bus are memory address and data registers, system and arithmetic unit registers, control unit, instruction decoder, instruction register and arithmetic and logical unit. *Registers in processor* also called register set(s) window(s) or file(s) are important and are meant for various functions such as context switch to process another task or ISR.
- Advanced processors have following additional structural units—prefetch control unit, instruction queuing unit, caches for instruction, data and branch transfer, floating point registers and floating point arithmetic unit. Pipelining and superscalar features and caches in the processors are used in high performance systems. MIPS or MFLOPS or Dhrystone per second define the computing performance. The goal is to provide optimal computing performance at the least cost and power dissipation.
- ARM, SHARC, TigerSHARC and DSPs processors are used in high performance computations.
- A system needs ROM and RAM memory of various types and address spaces. Various forms of ROM are masked ROM, PROM, EPROM, EEPROM, flash, boot-back flash and memory stick or card. Basic details of the memories are the addresses available, speed for read and write operations, and modes of memory access.
- The Processors and Memory selection can be done using appropriate design tables.
- Each IO device has a distinct set of addresses. Each IO device also has a distinct set of device registers – data registers, control registers and status registers. At a device address, there may be more than one register. The device addresses depend on the system hardware. Based on the memory map with IO device addresses, a locator program is designed to locate the linked object code file and generate a ROM image.

Keywords and their Definitions

Absolute addressing mode : Define all the address bits in an instruction.

Accelerator : ASIC, IP core or FPGA, which accelerates the code execution and which may also include the bus interface unit, DMA, read and write units and registers with their cores.

Accumulator : A register that provides input to an ALU and that accumulates a resulting operand from the ALU.

AHB : A high-performance version of the AMBA used in ARM processors.

ALU : A unit to perform arithmetic and logic operations as per the instructions.

AMBA : An established open source specification for on-chip interconnects that serves as a framework for SoC designs and IP library development.

Arithmetic unit registers : Registers that hold the input and output operands and flags with the ALU.

ARM : A family of high performance reduced code density ARM7, ARM9, ARM10 and ARM 11 processors, which are used in embedded systems as a chip, or as a core in an ASIC.

ARM7 and ARM9 : Two families of RISC processor for an SoC from ARM and Texas Instruments, also available in single-chip CPU versions and in file versions for embedding at a VLSI chip. ARM 7 has Princeton architecture for the main memory and ARM9 has Harvard Architecture.

Asynchronous serial : Data bytes or frames not maintain uniform phase differences in serial

communication : communication.

Auto index : When after executing an instruction, the index register contents change automatically.

Base addressing : Addressing an address from where a first element of data structure starts.

Baud Rate : Rate at which serial bits are received at the line during a UART communication.

Boot back flash : A flash with a few sectors similar to an OTP device, to enable storage of bootup program and data.

Branch transfer cache : Cache to hold in advance the next set of instructions to be executed on the program branching to this set.

Burning-in : A process in which bits are modified from all 1s in erase form to the 1s and 0s in a device as per input 1s and 0s.

Bus interface unit : A unit to interconnect the internal buses with the external buses for control, address and data bits.

CISC : A complicated Instruction Set Computer that has one feature that provides a big instruction set for permitting multiple addressing modes for the source and destination operands in an instruction. The hardware executes the instructions in a different number of cycles, as per the addressing mode used in an instruction.

Code Compatibility : Usability of codes by various generations of a family.

Code composer studio : An IDE for TI DSP-specific code composing which provides an environment similar to MS Visual C++. It consists of the following: multilevel (C as well as DSP assembly) debugger, compiler, assembly optimiser, RISC-like assembly codes and RISC-like scheduling for optimum performance and efficiency probe points, file IO functions, comprehensive data visualization displays and GEL scripting language based on C.

CODEC : A unit for digital coding after ADC and other operations and decoding to get analog signals using DAC and other operations. It is used in processing audio or CCD device pixels and video signals.

Control unit : To control and sequence all the processing actions during an instruction execution.

DAA : Direct access arrangement; for example, a typical DAA serial in and out port directly transferring the analog input and output using up to 1 master and 7 slave CODECs.

Data cache : Cache to hold the data in content-addressable memory format.

DCT : Discrete Cosine Tranformation function used in a number of DSP functions, for example, the MPEG2/MPEG4 compression.

Device address : A device address used by processor to access its set of registers. At each address there may be one or more device registers.

Device programmer : A system or unit for programming a device by burning-in ROM image.

Device programming : Programming of bits by burning-in a memory of microcontroller or in a PLA, PAL, CPLD or any other device.

Device register : A register in a device for byte, word of data, flags or control bits. Several device registers may have a common address.

Device : A physical or virtual unit that has three sets of registers: data registers, control registers and status registers, and which the processor addresses it like a memory.

Dhrystone : A benchmarking program that measures the performance of a processor for processing integers and strings (characters). It uses a benchmark program available in C, Pascal or Java. It benchmarks a CPU, not the performance of the IO or OS calls. 1 MIPS = 1757 Dhrystone/s.

Digital Filtering : A filter for the signals that use DSP functions.

Direct address : A directly usable address in an instruction. It is usually the address on a page in the memory.

DMA : A direct memory access by a controller internal or external. DMA operations facilitate the peripherals and devices of the system to obtain access to the system memories directly, without the processor controlling the transfer of the bytes in a memory block.

DRAM : Dynamic RAM, which refreshes continuously by a device called DRAM refresh controller. Once programmed, it auto reads and writes the same set of bits repeatedly by scanning the DRAM memory cells.

Echo cancellation : A process of eliminating echoes.

Echo : A signal received after a delay and which superimposes over the original signal. For example, in a hall or at the hills we hear the original sound as well as the echoed sound. Similarly, there may be echo in electronic signal.

EDA : A powerful tool for Electronic Design Automation.

EEMBC : EDN Embedded Benchmark Consortium.

EEPROM : A type of memory each byte of which is erasable many times and then programmable by the instructions of a program as well as by a device programmer.

EPROM : A type of memory that is erasable many times by UV light exposure and programmable by a device programmer.

Erase time : Time taken for device erasing.

Fixed point arithmetic : Arithmetic using signed or unsigned integers employing processor registers or memory.

Flash : A memory in which a set of sectors erase simulatenously.

Flash : A type of memory in a sector of bytes that is erasable many times (maximum. ~10000) in a flash at the same instance in a single cycle. Each erased byte is then programmable by the write instruction of a program as well as by a device programmer.

Floating point arithmetic : Arithmetic using processor registers or memory, where the decimal numbers and fractional numbers are stored in a standard floating-point representation.

High-level language support : A supporting unit for given processor structure that facilitates program coding in C or other high level languages and enables their running like machine codes by an internal compilation.

Index register : A register holding a memory address of a variable in an array. queue. table or list.

Instruction cache : A cache to sequentially hold the instructions that have been prefetched for pipeline base parallel execution.

Instruction decoder : The circuit to decode the opcode of the instruction and direct the control unit accordingly.

Instruction format : Format of expressing an instruction.

Instruction queuing unit : A unit to hold a queue of instructions and place these into the cache.

Instruction register : A register to hold the current instruction for execution.

Instruction set : A definite set of executabie instructions in a processor.

Instruction set : A unique processor-specific set of instructions.

Interface circuit : A circuit consisting of the latches. decoders. multiplexers and demultiplexers.

Internal bus : A set of paths that carry in parallel the signals between various internal structural units of a processor. Its size is 64-bit in a 64-bit processor.

Java Accelerator : An accelerator that helps in the execution of Java codes faster than a JVM.

JVM : Machine codes that use the compiled byte codes of a Java program and run the program on a given system.

MAC unit : A unit used in DSP operations for fast calcuation of $\Sigma[ax_n + (b_i.y_j)]$ or similar expressions.

Master : A processor, device or system which synchronously or asynchronously controls the output to several different processors, devices and systems called slaves. The master can choose to send output to an addressed slave if a slave has a distinct address. The master can choose to receive an input from any slave selected by it at an instant.

McBSP : A high-speed communicating multichannel Buffered Serial Port.

Memory address register : A register that holds the address for a memory unit for placing it on the bus using bus interface unit.

Memory data register : A register that holds the data for or from a memory unit.

Memory management unit : A unit to manage the prefetch, paging and segmentation of memories.

Memory map : A memory addresses allocation table such that the map reflects the available memory addresses for various uses of the processor. A memory map defines the addresses of the ROMs and RAMs of the systems.

Microarchitecture : When a processor architecture refers specifically to the architectural instruction sets and programmers' models, the term microarchitecture refers specifically to the implementation of those architectures. A processor may have CISC architecture with an RISC microarchitecture implementation.

Noise elimination : A process that eliminates unrelated randomly introduced signal components.

OMAP 5910 processor : A TI processor of unique architecutre in DSP chips of high performance with low power consumption.

On-chip parallel port : The port on a chip which receives or sends 8 or 16 bits at an instance.

On-chip serial port : The port on a chip which receives or sends a bit serially at an instance with a definite rate in kbps [baud rate in UART].

Opcode : First byte of an instruction for the instruction decoder of the processor. It defines the operation or process to be performed on the operand(s).

Performance benchmarking : Metrics for evaluating the performance of a system.

Pipelining : There is pipelining also in the superscalar processor. It means than its ALU circuit divides into n subunits. If in its last part. the processing of a p-th instruction is taking place at an instant. then at the first part processing of (p + n) th instruction is taking place. There may be multiple pipelines in a processor to process in parallel.

Prefetch control unit : A unit to fetch instructions in advance and data in advance from the memory units.

Program counter : A processor register to hold the current instruction address to be executed after a fetch cycle on the buses.

Program flow instruction : An instruction in which the program counter or instruction pointer changes in a way different from its normal changes during program execution.

PROM or OTP : A type of memory which is programmable only once by a device programmer. OTP is a one-time programmable memory.

Pulse accumulator counter : A counter that counts the input pulses during a slelect interval. When used as a timer with a repeatledly loadable value, it functions as a pulse width modulator.

Real time video processing : Processing of video signals such that all or most incoming frames are processed in a time frame such that each processed frame maintains constant phase differeces in the intervals between them.

RISC with CISC functionality	:	A processor with RISC implementation but user programs it similar to a CISC.
RISC	:	A Reduced Instruction Set Computer that has one feature that provides a small instruction set and permits limited addressing modes for the source and destination operands in an arithmetic or logic instruction. The hardware executes each instruction in a single cycle.
Segment register	:	A register to point to the start of a segment for a program code or data set or string or stack.
Slave	:	A processor or device or system, which receives the input from the master processor or device or system. This slave is the one having a distinct address and is chosen by the master.
Special function register	:	A register in 8051 for special functions of accumulator, data pointer, timer control, timer mode, serial buffer, serial control, power down control, ports, etc.
Stack pointer	:	A register that hold an address to define the available memory address to where the processor can push the registers and variables on a stack operation and from where they can be popped.
Stack	:	A block to memory that holds the pushed values for last-in first-out data transfer on popping back the values.
Superscalar processor	:	A processor with the capacity to fetch, decode and execute more than one instruction in parallel at an instant.
Synchronous communication	:	Data bytes or frames maintain uniform phase differences in serial communication.
System register	:	Processor register.
Thumb® instruction set	:	An instruction set in which each instruction is of 16 bits on a 32-bit processor. It gives reduced code density. It is a 16-bit instruction set which enables 32-bit performance at 8/16-bit system cost. They are used by ARM processors.
Timing diagram	:	A diagram that reflects the relative time intervals of the signals on the external buses with respect to the processor clock pulses.
Video accelerator	:	An accelerator for the video output.
Watchdog Timer	:	A timer which is set in advance and program codes are made such that its overflow indicates that a process is stuck somewhere and therefore the processor resets and restarts.

Review Questions

1. Exlpain 8051 architectural features. What are the devices internally present in the classic 8051. How do you interface a programmable peripheral interface in 8051?
2. Describe serial interface, timer/counters and interrupts in 8051.
3. Describe real-word interfacing. Explain interfacing to keyboard.
4. Compare memory-mapped IO and IO-mapped IOs.
5. What are the common structure units in most processors?
6. Compare Harvard and Princeton memory organizations.
7. What are the special structural units in processors for digital camera systems, real-time video processing systems, speech compression systems, voice compression systems and video games?

8. How do having separate caches for instruction, data and branch-transfer help?
9. What is the advantage of having multiway cache units so that only that a part of the cache unit is activated, which has the necessary data to execute a subset of instructions? List four exemplary processors with multiway caches.
10. When do you need MAC units at a processor in the system?
11. Explain three-stage pipeline, superscalar processing and branch- and data-dependency penalties.
12. What are the advantages in Harvard architecture? Why is the ease of accessing stack and data-table at program memory less in Harvard memory architecture compared to Princeton memory architecture?
13. Explain three performance metrics of a processor: MIPS, MFLOPS and Dhrystone per second.
14. Why should a program be divided into functions (routines or modules) and each placed in different memory blocks or segments?
15. How do the ARM7, ARM 9, ARM 11 and StrongArm differ? When will you prefer ARM7, when ARM 9 and when ARM11?
16. How does a memory map help in designing a locator program?
17. What do you mean by the terms: Quarter-CIF, EDO RAM, RDRAM, peripheral transcations server, shadow segment, on-chip DMAC and time-division multiplexing.
18. How does a decoder help in memory and IO device interfacing? Draw four exemplary circuits.

Practice Exercises

19. Draw the memory organization in 8051.
20. How will you interface an 8051 to four servomotors in a robot using timer/counters and ports of 8051?
21. A two by three matrix multiplies by another three by two matrix. If data tranfer from a register to another takes 2 ns, addition takes 20 ns and multiplication takes 50 ns, what will be the execution time? How will a MAC unit help. Asume that these times are same in a DSP with a MAC unit?
22. An array has 10 integers, each of 32 bits. Let an integer be equal to its index in the array multiplied by 1024. Let the base address in memory be 0x4800. How will the bits be stored for the 0^{th}, 4^{th} and 9^{th} element in (a) big-endian mode (b) little-endian mode?
23. We can assume that the memory of an embedded system is also a device. List the reasons for it. [Hint: Use of pointers like access control registers and the concept of virtual file and RAM disk devices.]
24. Nowadays high-performance embedded systems use either an RISC processor or a processor with an RISC core with a code-optimized CISC instruction set. Why?
25. A circular queue has 100 characters at the memory addresses, each of 32 bits. What will be the total memory space required, including the space for both the queue pointers?
26. Estimate the memory requirement for a 500-image digital camera when the resolution is (a) 1024 x 768 pixels, (b) 640 x 480, (c) 320 x 240 and (d) 160 x 120 pixels and each image stores in compressed jpg format.
27. What are the special structural units in processors for digital camera, real-time video processing, speech compression and video game systems?