UNIT-II

ARM PROCESSOR (Advanced RISC Machine)

→ The ARM is a 32-bit Reduced Instruction Set Computer (RISC) processor.

→ ARM processors are suitable for low power applications.

→ This has made them dominant in the mobile and embedded electronics market.

→ ARM architecture comes in several versions.

→ CISC : Complex Instruction Set Computer
These machines provided a variety of instructions that may perform very complex tasks and it generally used a number of different instructions formats of varying lengths.

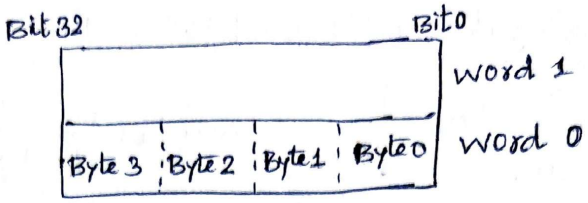
→ RISC : Reduced Instruction Set Computer
It is developed for high performance microprocessor & its instruction sets are executed in pipelined processors.
It can be used for any types of applications.

→ ARM 7 is a Von Neumann architecture & it supports two basic types of data.
1). The standard ARM word is 32 bit long.
2] The word may be divided into four (8-bit) bytes.

→ ARM 7 allows addresses upto 32 bit long.
here 1 word = 32 bit
word 0 has location 0
word 1 has location 4
word 2 has location 8 and so on.

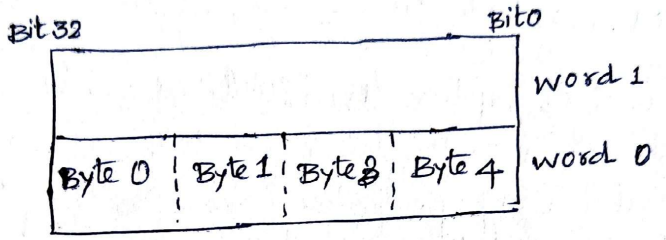
→ ARM processor can be configured at power to address the bytes in a word in either Little-endian or Big-endian mode.

→ In little-endian mode the lower order byte residing in the low order bits of the word.



Little-Endian
Intel x80

→ In big-endian mode the lowest order byte stored in the highest bits of the word.



Big-Endian
Motorola

→ Arithmatic & logical operations cannot be performed directly on memory locations.

→ ARM is a load store architecture and data operands must be loaded into the CPU and then store back to main memory to save the results.

→ ARM has CPSR (Current Program Status Register) in its programming model. This Register is set during every arithmatic, logical or shift operation.

→ When used in relation to the ARM

Byte means 8 bits - 1 byte
Half word means 16 bits - 2 bytes
word means 32 bits - 4 bytes

→ Most ARM's implement two instruction sets

32-bit ARM Instruction Set
16-bit Thum Instruction Set

→ Jazell cores can also execute Java byte code.

→ The ARM has seven basic modes of operations.
Each mode has access to its own stack space and a different subset of register.
Some operations can only be carried out in a privileged mode.

ARM Processor Modes

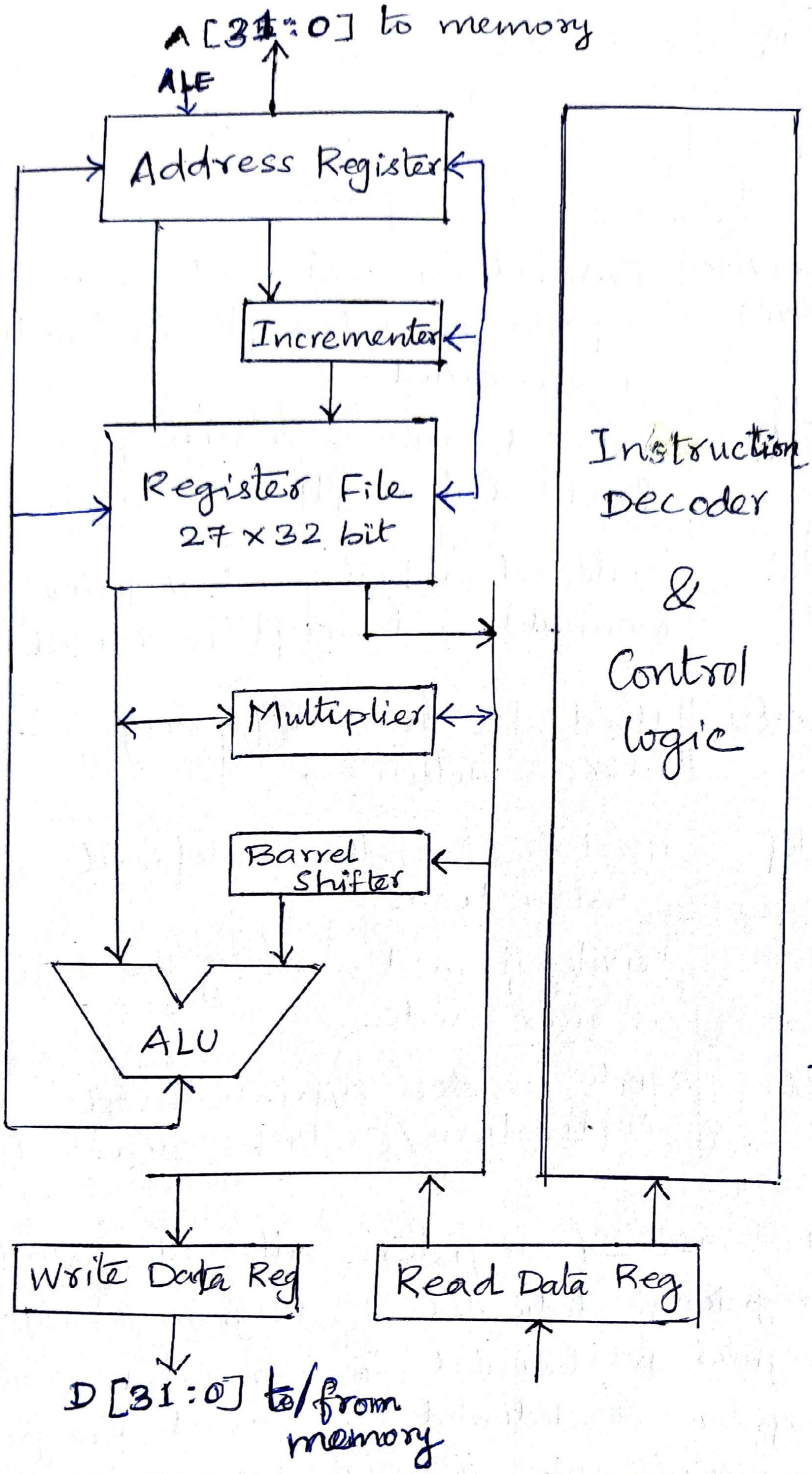
Mode	Description	Remarks
Supervisor (SVC)	Entered on reset and when a Simultaneous Interrupt instruction is executed.	Privileged Modes
FIQ	Entered when a high priority (fast) interrupt is raised.	
IRQ	entered when a low priority (normal) interrupt is raised.	
Abort	used to handle memory access violations	
Undef	used to handle undefined instructions	
System	Privileged mode using the registers as user mode	
User	Mode under which most Applications / OS tasks run	Unprivileged mode

Exception modes.

⇒ ARM has 37 registers all of which are 32-bits long
- 1 register is dedicated to Program counter.
- 1 register is dedicated to current program status register.
- 5 registers are dedicated to saved program status registers.
- 30 registers are general purpose registers.

⇒ The current processor mode governs which of several banks is accessible. Each mode can access
- a particular set of r0-r12 registers
- a particular r13 (the stack pointer, SP) & r14 (the link reg. lr)
- the program counter, r15 (PC)
- the current program status register (CPSR)

⇒ Privileged modes (except system) can also access
- a particular SPSR (saved program status register)



* large uniform register file
* Load/store architecture
* Simple addressing mode
* Uniform & fixed-length instruction fields.
* High performance
* low code size
* low power consumption
* low silicon area
* 16 visible R0-R15
* 31 general-purpose 32-bit reg

→ ARM is an RISC architecture and thereby the operands are not directly fetched into the ALU in fact that they will first store in the register file and then to the ALU.

→ In the code datapath Barrel shifter doing an important role such that it can processes the data & barrel shifter can shift the data into right or left according to the arbitrary value assigned in the code.

→ The barrel shifter differ from normal shift register in such a way that they can do all these shifts in a single cycle but shift registers can do the same depends on number of shift required.

→ The barral shifter are combination circuit as like shift registers.

→ Auto increment block can operate register independent of ALU.

→ The Auto incrementer block is followed by a incrementer bus which has access to both register bank (31*32 registers & 6 status registers).

→ The AI block can take the address from Address Register and can operate with the same without stay in queue for ALU to finish it's operation.

→ Fast Interrupt Response.

→ Excellent High level langnage support 'c' Language.

→ Simple & powerful Instruction set.

→ Load/Store multiple instructions.

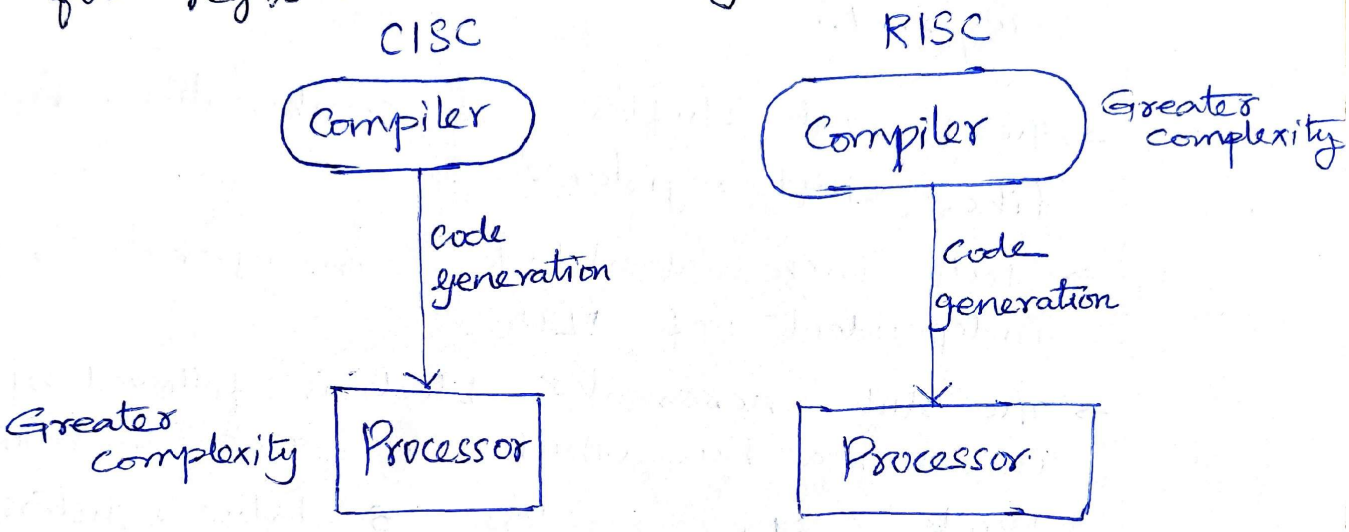
→ Conditional execution.

→ Auto Increment/Decrement addressing mode.

→ ARM7TDMI provides up to 120 Dhrystone MIPS and is known for its high code density & low power consumption, making it deal for Embedded devices.

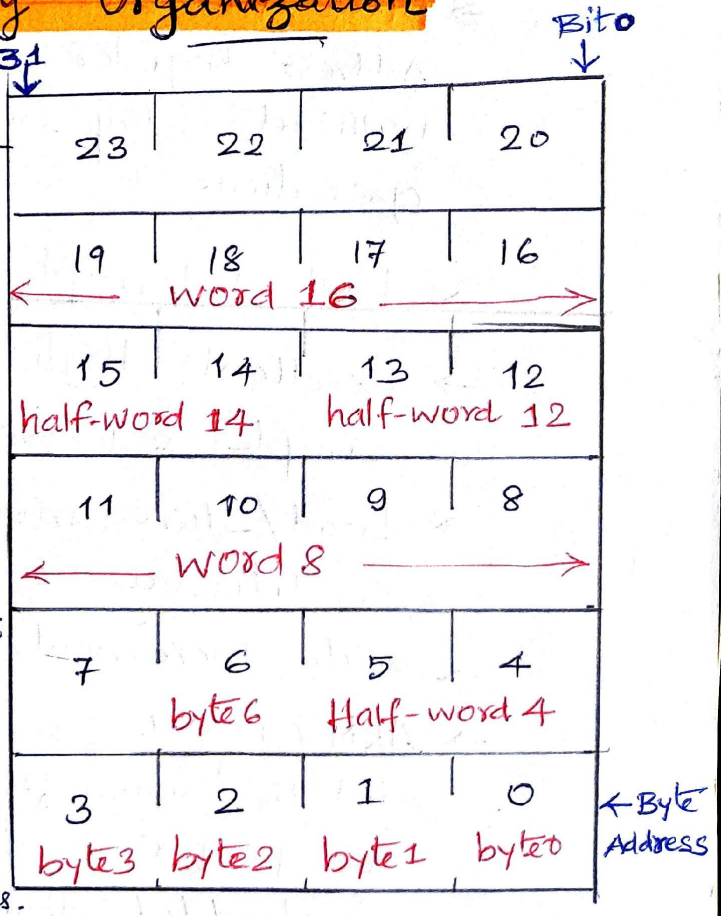
→ RISC machines have a large general-purpose register set. Any register can contain either data or address. Registers act as fast local memory store. In contrast, CISC processor have dedicated reg. for specific purpose.

→ RISC processor operates on data held in register banks. This will reduce multiple memory access and seperate load & store instructions are there for register bank-memory access.



ARM Memory Organization

→ The ARM7 is a Von-Neuman load/store airchitecture
→ only 32 bit data bus for both instruction & data
→ Memory is addressed as a 32 bit address space.
→ Data byte can be 8-bit bytes, 16-bit half words or 32 bit words and may be seen as a byte line folded into 4-byte words
→ Word will be alligned to a 4 byte boundaries.
→ Half-word must be alligned to 2 byte boundaries.



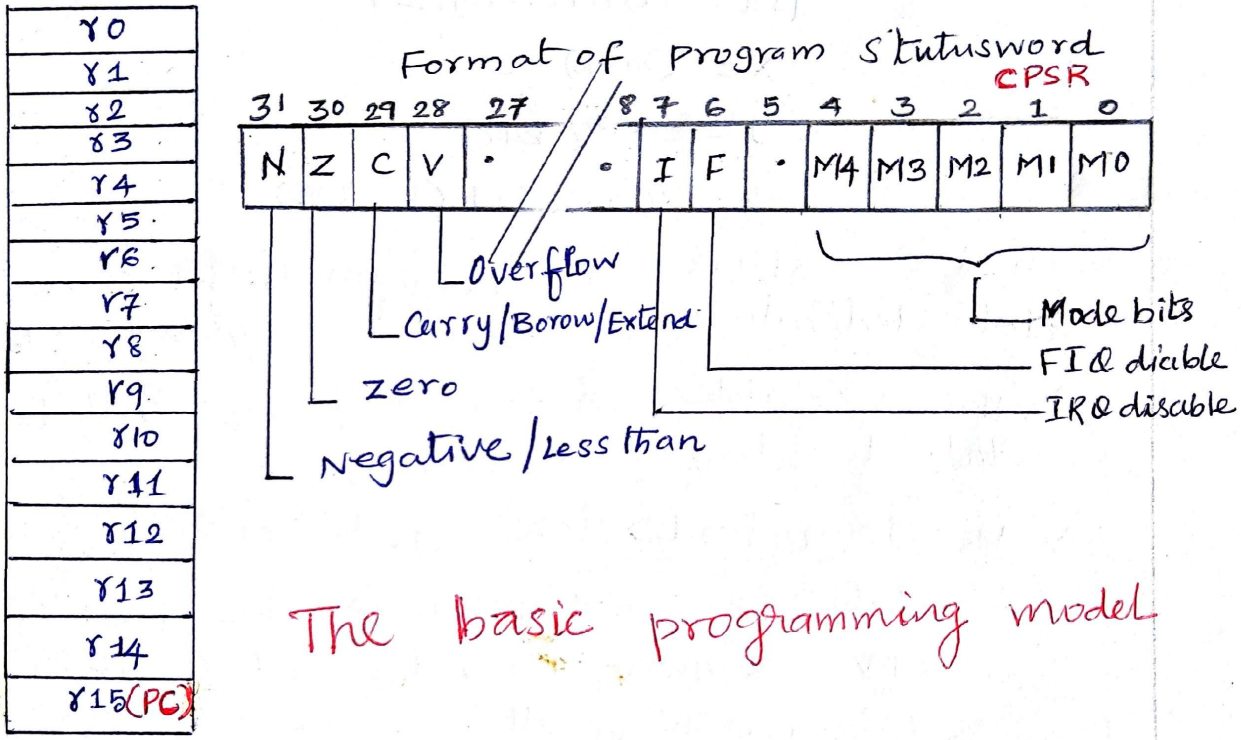
→ The memory controller supports all three access sizes.
→ Always ensure that memory controller supports all three access sizes.

⇨ The negative (N) bit is set when the result is negative in two's complement arithmatic

⇨ The zero (Z) bit is set when every bit of the result is zero.

⇨ The carry (C) bit is set where there is a carry out of the operation.

⇨ The overflow (V) bit is set when an arithmatic operation result in an overflow.



The basic programming model

⇨ These bits can be used to check easily the result of an arithmatic operation.

⇨ However, if a chain of arithmatic or logical operations is performed & the intermediate state of the CPSR bits are important then they must be checked at each step since the next operation changes the CPSR value.

The basic form of a data instruction is simple

ADD r0, r1, r2 [r0 ← r1 + r2]

ADD r0, r1, #2. [r0 ← r1 + #2]

Data Operations

⇒ Arithmatic and logic operations in C are performed by variable.

⇒ Variables are implemented as memory locations.

⇒ Therefore, to be able to write instructions to perform C expressions & assignments, & consider both arithmatic & logical instructions as well as instructions for reading & writing memory.

```
int a, b, c, x, y, z;
x = (a+b) - c
y = a*(b+c)
z = (a<<2) | (b & 15)
```

in the above 'C' programming, C code with data declerations & several assignment statements.

⇒ The variables a, b, c, x, y & z all became data locations in memory.

⇒ ARM is a load--store architecture

- data operand mush first be loaded into the CPU. & then stored back to main memory to save the results.

⇒ ARM has 16 general-purpose register $r0 \div r15$

⇒ r15 is program counter (PC)

⇒ PC always keep the track of next instruction to be executed.

⇒ The current Program status register (CPSR).

⇒ CPSR is set automatically during every arithmatic, logical, or shifting.

⇒ The top of the register holds arithmatic / logical operations Results.

⇒ The result of that arithmatic / logical operation

⇨ RSB performs a subtraction with the order of the two operands reversed

RSB r0, r1, r2 [r0 ← r2 − r1]

⇨ The bit-wise logical operations performs logical AND, OR, and XOR operations

⇨ BIC instruction stands for bit clear.

BIC r0, r1, r2 [r0 ← r1 and not r2] where a bit in the mask is 1 in second operand, the corresponding bit in the first source operand is cleared.

⇨ MUL instruction multiplies two values.

⇨ MLA instructions perform a multiply-accumulate operation, particularly used in matrix operations and signal processing

MLA r0, r1, r2, r3 [r0 ← r1 × r2 + r3]

⇨ Shift operations

LSL – logic Shift Left (filling the Least-significant bits with zeros)
LSR – logic shift Right

⇨ The arithmatic shift left is equal to LSL

i.e ASL = LSL

⇨ ASP – it copies the sign bit

⇨ The rotate modifier always rotate right LSB → MSB

RRX modifier performs a 33-bit rotate, with the CPSR's 'c' bit being inserted above the sign bit of the word. This allows the carry bit to be included in the rotation.

⇨ Comparision operation:

CMP r0, r1 [compute r0 − r1, sets the status bits NZCV & not result of subtraction

CMN uses an addition to set the status bits

⇨ TST performs a bit-wise AND on the operand
TEQ performs a bit-wise Exclusive-OR operation.

⇨ MOV r0, r1 [r0 ← r1]
MVN instruction complements the operand bits during the move.

Instruction Sets

ADD	Add
ADC	Add with carry
SUB	Subtract
SBC	Subtract with carry
RSB	Reverse Subtract
RSC	Reverse Subtract with carry
MUL	Multiply
MLA	Multiply & accumulate

Logical	
AND	Bit-wise AND
ORR	Bit-wise OR
EOR	Bit-wise - Exclusive-OR
BIC	Bit clear

Shift / rotate	
LSL	Logical shift left (Zero fill)
LSR	logical shift right (zero fill)
ASL	Arithmatic shift Left
ASR	Arithmatic shift Right
ROR	Rotate Right
RRX	Rotate Right extended with C

CMP	compare
CMN	negated compare
TST	Bit-wise test
TEQ	Bit-wise negated test

MOV	Move
MVN	Move negated

⇨ The values are transferred between registers & memory using load & store instructions.
⇨ LDRB & STRB load and store bytes rather than whole words
⇨ LDRH & SDRH operate on half words
⇨ LDRSH & SDRSH extends the sign bit on loading.
⇨ The ARM uses indirect addressing in load & store operations

⇨ LDR r0, [r1] : r1 hold the address, the data available in memory location mentioned in r1 is load into r0

STR r0, [r1] : store the contents of r0 in the memory location whose address is given in r1.

LDR r0, [r1-r2] : r0 ← (r1-r2)

LDR r0, [r1, #4] ; r0 ← (r1+4)

ADR r1, F00 ; r1 ← [0x100] ∵ 0x100 = F00

Pseudo operation

ARM load-store instruction and pseudo-operations.

LDR	load
STR	store
LDRH	load half-word
STRH	store half-word
LDRSH	load half-word signed
LDRB	load byte
STRB	store byte
ADR	set register to address

⇨ The ARM also support seveal form of base-plus-offset addressing.

LDR r0, [r1, #16] ; r0 ← [r1+16]

⇨ This addressing mode has two other variations: auto-indexing & post-indexing. Auto-indexing updates the base register, such that

LDR r0, [r1, #16]! (base register, off set value)

first adds 16 to the value of r1 & then uses that new value as the address.

! operator causes the base register to be updated with the computed address so that it can be used later.

⇨ Post-indexing does not perform the offset calculation until after the fetch has been performed

LDR r0, [r1], #16 : r0 ← [r1] then [r1] + 16

In this case, the post-indexed mode fetches different value than the other two examples but ends up with the same final value for r1 as does auto-indexing.

Flow control

⇨ The B (branch) instruction is the basic mechanism in ARM for changing the flow of control.

⇨ The branch destination address is called as **branch target**

⇨ When branch occurs the PC is updated to the branch target.

⇨ The offset is in words, but the ARM is byte addressable, the offset is multiplied by four (shift left two bits) to form a byte address.
Thus, the instruction

B #100 [will add 400 to the current PC value]

⇨ The conditional braches alway refers the condition codes as follows.

UNIT II EMBEDDED PROCESSORS & PERIPHERALS

The CPU BUS

⇨ The bus is the machanism by which the CPU communicates with memory and devices.

⇨ A bus is a collection of wires, protocol, memory and devices communicate.

⇨ The bus is to provide an interface to memory and I/O devices.

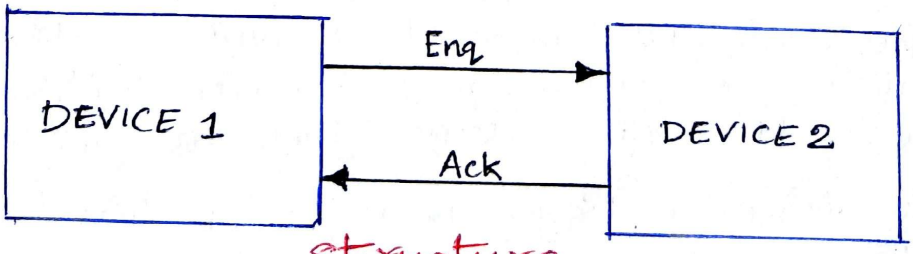
BUS PROTOCOLS

* The basic building block of most bus protocol is the four-cycle handshake
* The handshake ensures that when two devices want to communicate, one is ready to transmit & other is ready to receive.
* The two wires are dedicated to the handshake with enq (enquiry) and ack (acknowledge). Extra wires are used for data transmitting.

The four-cycles are described below:

1). Device 1 raises its output to signal an enquiry, which tells device 2 that it should get ready to listen for data

2) when device 2 is ready to receive, it raises its output to the signal an acknowledgment. At this point devices can transmit & receive.



structure

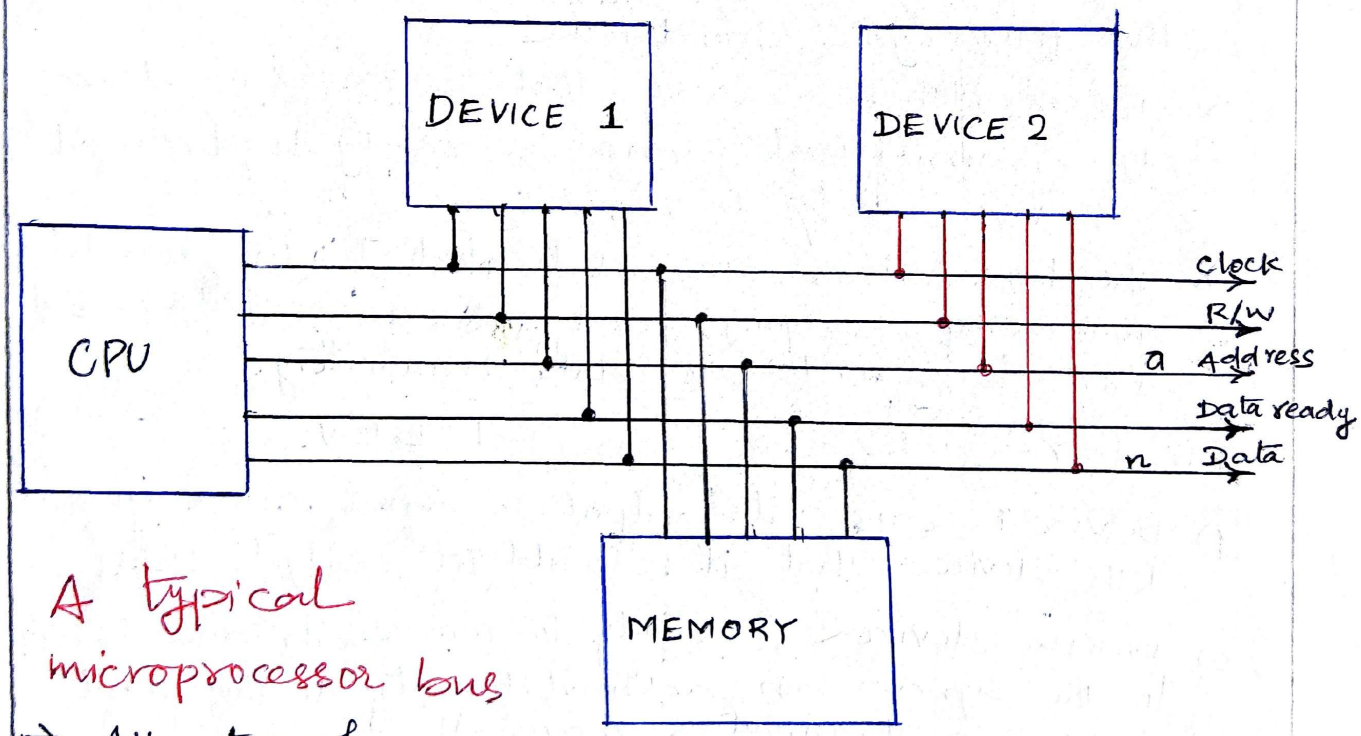
3) Once the data transfer is complete, device 2 lower its output, signaling that it has received the data.

4) After seeing the ack has been released, device 1 lower its o/p.

⇨ At the end of the handshake, both handshaking signals are low, just they were at the start of the handshake then the system is ready for another tranfer.

⇨ The below figure shows the structure of a typical bus that supports read & writes

* clock provides synchronization to the bus components.
* R/W is true when the bus is reading and false when the bus is writing.
* Address is true an a-bit bundle of signals that transmit the address for an access.
* Data is an n-bit bundle of signals that can carry data to or from the CPU &
* Data ready signals when the values on the data bundle are valid.



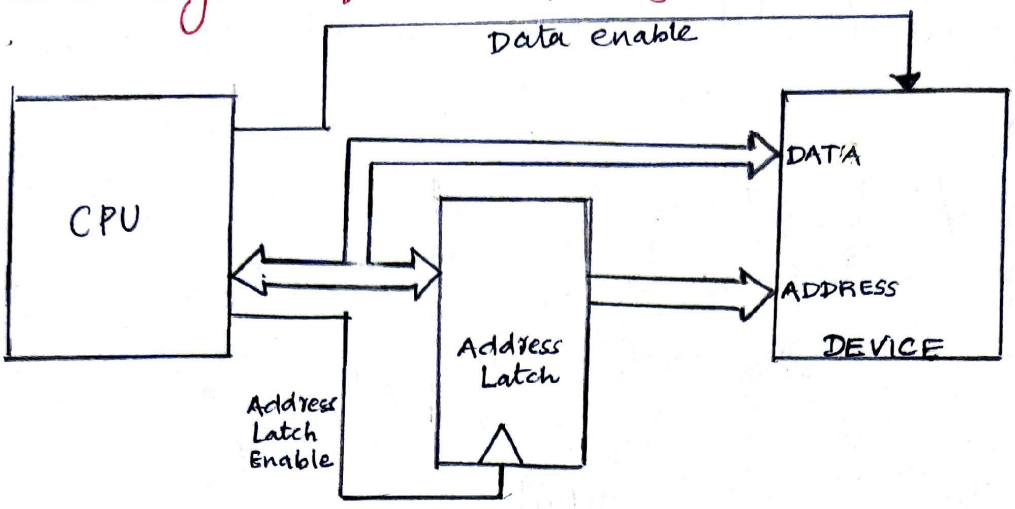
A typical microprocessor bus

⇨ All transfer on this basic bus are controller by the CPU - the CPU can read or write a device or memory but devices or memory cannot initiate a transfer.

⇨ The addres bus & R/W signals are unidirectional.

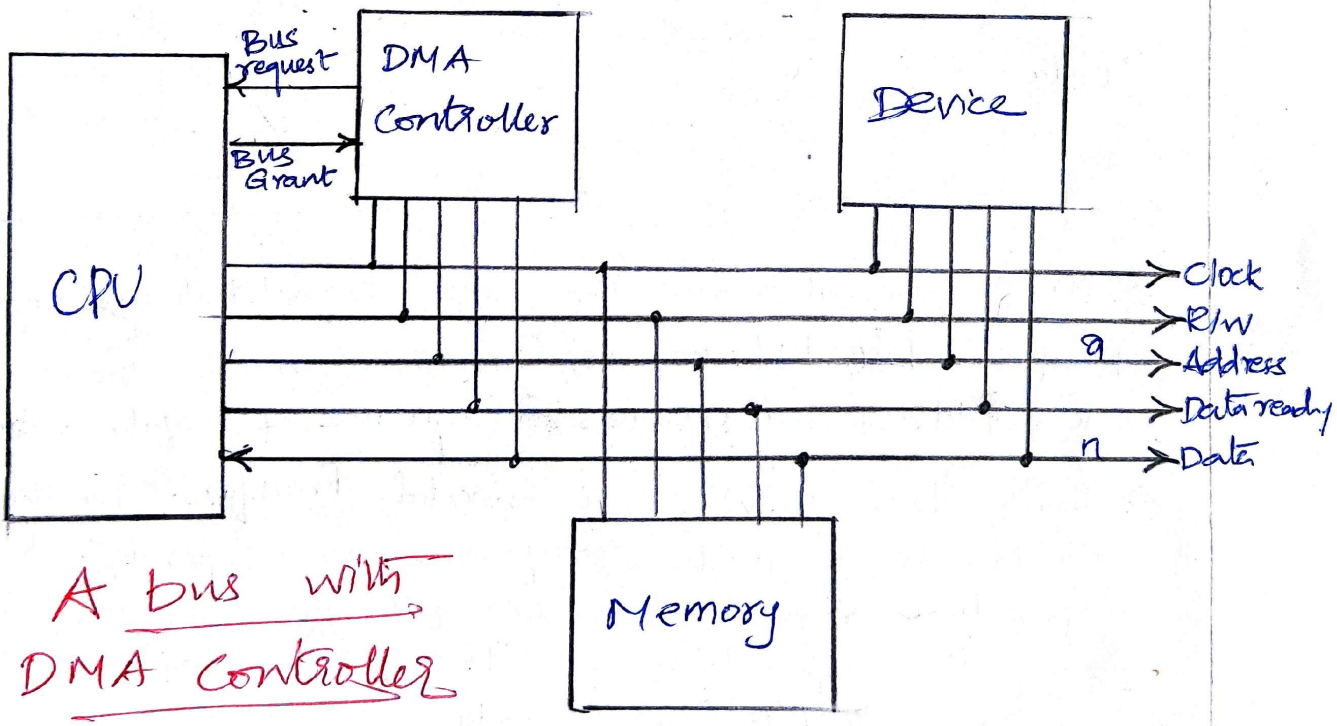
⇨ The timing diagram is a pictorial representation of bus signals, which clearly shows the individual signal state with respect to the clock signal.

Bus signals for multiplexing address & data



DMA : Direct Memory Access

For large data with high-speed transfer between memory and the I/O device without involving the CPU can be performed with the help of DMA controller.



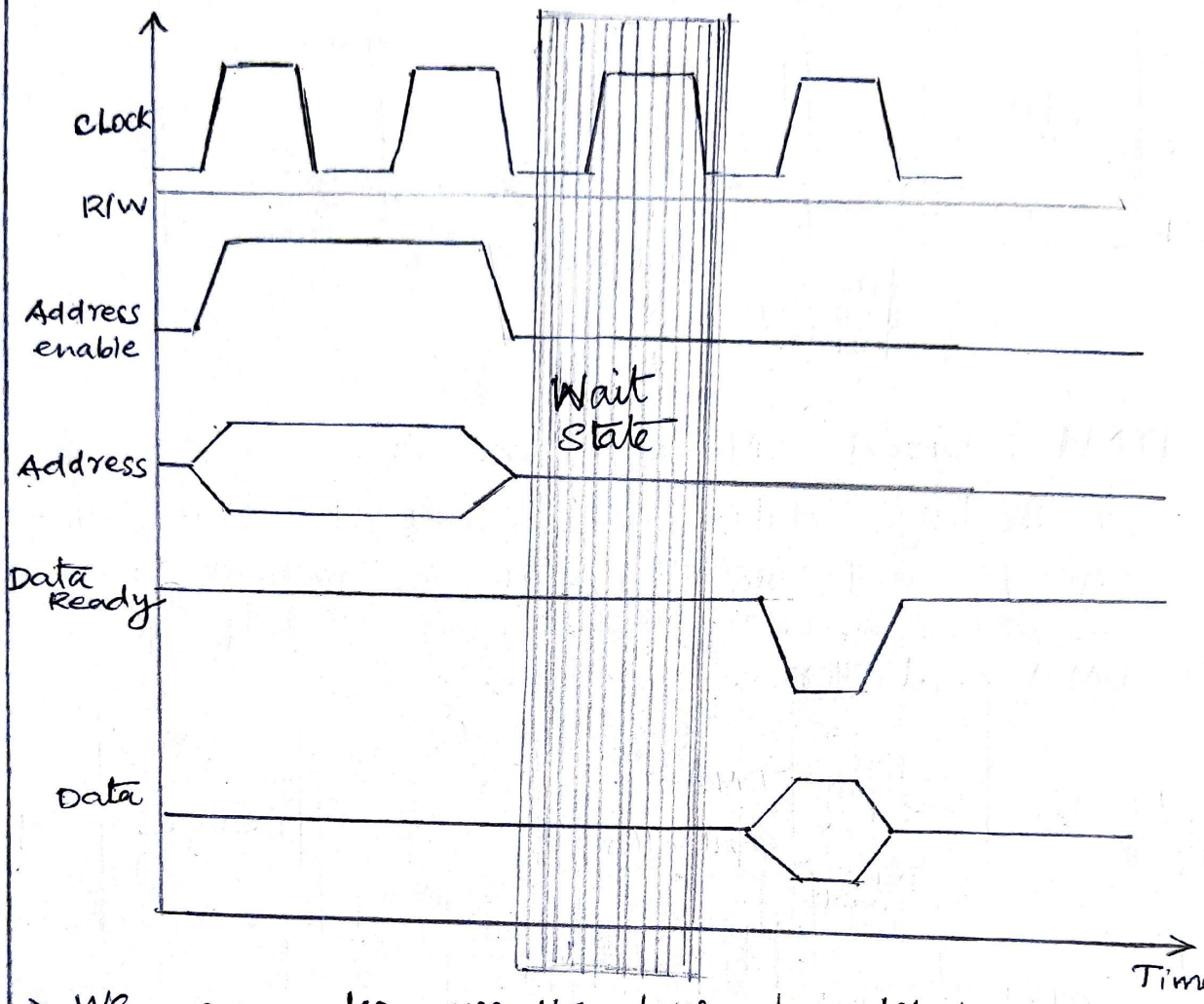
A bus with a DMA controller

⇒ DMA is a bus operation that allows reads & writes not controlled by the CPU but controlled by DMA controller which request the control of the bus from the CPU.

⇒ After gaining the control, the DMA controller performs read & write operation directly between devices & memory.

⇒ Bus request is used by the DMA controller for gaining the bus. Bus grant is used by the CPU to grant bus the DMA controller.

A wait state on the read operation



⇨ We can also use the bus handshaking signals to perform burst transfers, In this mode the CPU sends one address but receives a sequence of data values.

⇨ Some bus provids disconnected transfers. In these buses, the request & response are seperate. A first operation requests the transfer. The bus can then be used for other operations. The transfer is completed later, when the data are ready.

⇨ Some bus use multiplex address & data. Additional control lines are provided to tell whether the value on the address/data lines is an address or data

⇨ Usually, the address comes first followed by the data.

⇨ The address can be held (latch) in a register until the data arrive. so that both can be presented to the device at the same time.

System bus configurations

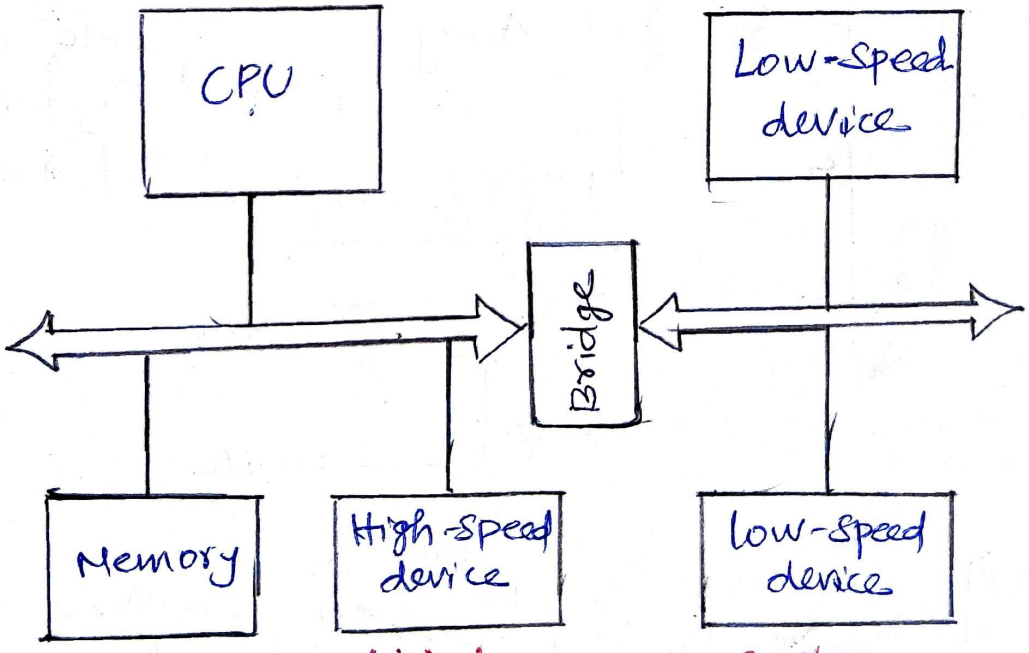
⇒ A microprocessor system often has more than one bus.

⇒ High-speed devices may be connected to a high-performance bus, while lower-speed devices are connected to a different bus.

⇒ A small block of logic known as **bridge** allows the buses to connect to each other

⇒ Advantages of bridges.

* High-speed buses may provide wider data connections
* A high-speed bus requires more expensive circuits & connections .. the cost of low-speed devices can be held down by using a low-speed low-cost bus.
* The bridge may allow the buses to operate independently, thereby providing some parallelism in I/O operations.



A Multiple Bus System

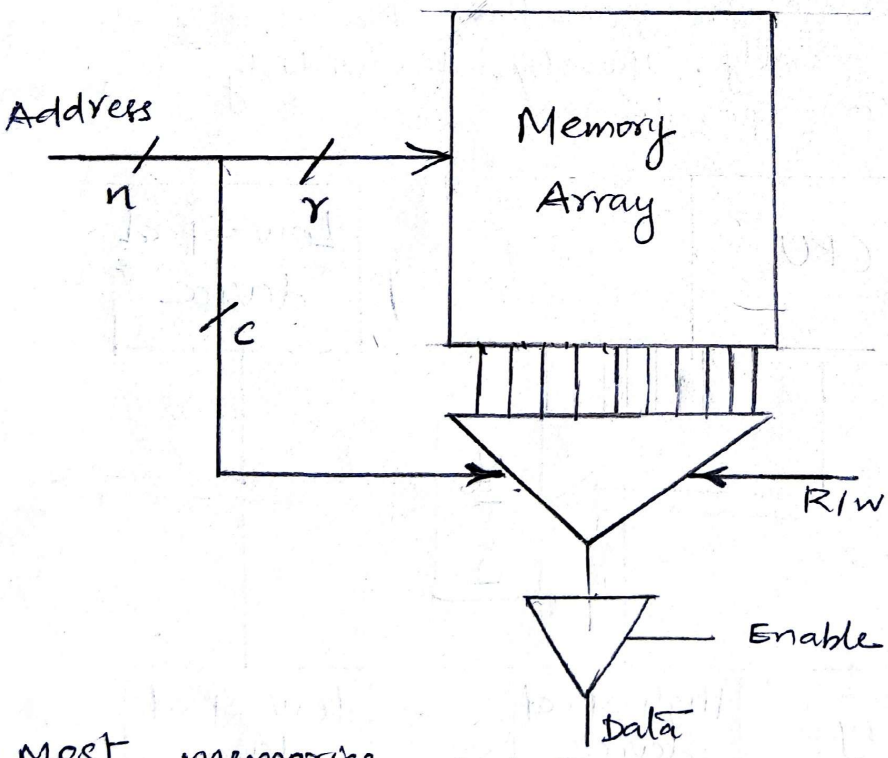
Memory Device Organization

⇨ The memory is characterized by its capacity. Such as 256 MB may be available in two version.

* As a 64M x 4-bit array, a single memory access obtained an 8-bit data item, with a maximum of 2^{16} different addresses.
* As a 32M x 8-bit array, a single memory access obtains a 1-bit data item with a maximum of 2^{13} different address.

⇨ The heigh/width ratio of a memory is known as its aspect ratio.

⇨ Internally, the data is stored in a two-dimentional array of memory cells. then its split as a row & a column address. (with $n = r + c$).



Internal Organization of a memory Device

⇨ Most memories includes an enable signal that controls the tri-stating of data onto the memory's pins.

⇨ A read/write signal (R/W) on read/write memories controls the direction of data transfer

Random-Access Memory (RAM)

⇨ RAM can be read and written.
⇨ The most bulk memory in modern systems is dynamic RAM (DRAM).
⇨ DRAM is very dense: it requires that its value be refreshed periodically since the value inside the memory cells decay over time. [every 10 ms refreshing needs]
⇨ The dominant form of DRAM is Synchronous DRAMs (SDRAM), which uses clocks to improve DRAM performance.

⇨ SDRAM use Row Address Select (RAS) & Column Address Select (CAS) to breaks the address into two parts, which select the proper row & column in the RAM array.
⇨ Signal transitions are relative to the SDRAM clock, which allows the internal SDRAM operations to be pipelined.
⇨ SDRAMs use a seperate refresh signal to control refresh signal to control refreshing.
⇨ Rather than refresh the entire memory at once. DRAMs refresh part of the memory at a time.
⇨ Refreshing slow down the speed of memory.
⇨ SDRAM supports burst modes that allows several sequential addresses to be accessed by sending one address.
⇨ SDRAM supports interleaved mode that exchanges pair of bytes.
⇨ Faster synchronous DRAM is known as Double-Data rate (DDR) SDRAM or DDR2 & DDR3 SDRAMs. are now in use.
⇨ DDRs simply use sophisticated circuit techniques to perform more operations per clock cycle.
⇨ SIMM (single in-line memory modules or) DIMM (double in-line memory modules) are generally used in PCs
⇨ A SIMM or DIMM a small circuit board that fits into a standard memory socket. Memory chips are soldered to the circuit board to supply the desired memory.

Read-only Memory (ROM)

⇨ ROMs are preprogrammed with fixed data. They are very useful in embedded system since a great deal of the code, and some data, does not change over time.

⇨ ROMs are also less sensitive to radiation-induced errors.

⇨ The basic types are factory-programmed ROM (or Mask-programmed ROM) & Field-programmable ROM.

⇨ Factory-programmed ROM are ordered from the factory with particular programming.

⇨ Field-programmable ROMs can be programmed in the lab.

⇨ Flash memory is the dominant form of field programmable ROM & is electrically erasable.

⇨ Flash memory use standard system voltage for erasing & programming, allowing it to be reprogrammed inside a typical system.

⇨ It supports automatic distribution of upgrades.

⇨ The modern Flash memory allows memory to be erased in blocks.

⇨ A common application is to keep the boot-up code in a protected block but allow updates to other memory blocks on the device.

⇨ As the result, this form of flash is commonly known as boot-block flash.

I/O DEVICES

⇨ Some of the I/O devices are used as on-chip devices on micro-controllers; others are implemented seperately.

⇨ common I/O devices used in embedded computing system

* Timers and Counters
* Watchdog Timer
* A/D and D/A Converters
* Keyboard
* LEDs
* Displays
* Touchscreens

Timers and Counters

⇒ Both are built from adder logic with registers to hold the current value, with an increment input that adds one to the current register value.

⇒ A timer has its count connected to a periodic clock signal to measure time intervals.

⇒ A counter has its count input connected to an aperiodic signal in order to count the number of occurances of some external event.

⇒ Because the same logic can be used for either purpose, the device is called a counter/timer.

A watchdog timer

⇒ It is an I/O device that is used for internal purpose of a system.

⇒ The watchdog timer is connected into the CPU bus & also to the CPU's reset line.

⇒ The CPU's software is designed to periodically reset the watchdog timer, before the timer (still) ever reaches its time-out limit.

⇒ If the watchdog timer still does reach that limit, its time-out action is to reset the processor.

⇒ Otherwise CPU will hang-up, rather than diagnose the problem, the system is reset to get it operational as quickly as possible.

A/D & D/A converter

⇒ These devices are used to interface non-digital devices to embedded systems.

⇒ A/D conversion requires sampling the analog input before converting it to digital form.

⇒ A control signal causes the A/D converter to take a sample & digitize it.

⇒ There are two different types of A/D converter circuit
1) Constant-time converter &
2) Variable-time converter.

⇨ A typical A/D interface has analog input and two digital inputs. A data port allows A/D registers to be read and written, & a clock input tells when to start the next convertion.

⇨ D/A conversion is relatively simple. It's includes only the data value. The input is continuously converted to analog form.

Keyboard

➡ A keyboard is an array of switches, it includes some internal logic to interface with microprocessors.

➡ The major problem with mechanical switches is bouncing.

➡ The bouncing problem is rectified by hardware method and software method.

➡ A hardware debouncing circuit can be built using a one-shot timer & the software can also be used debounce the switch input.

➡ The more key switches are orgonished using encoded keyboard.

➡ An encoded keyboard use some code to represent which switch is currently being depressed.

➡ The important part of the encoded key is scanned array of switches

➡ The scanned key board array reads only one row of switches at a time.

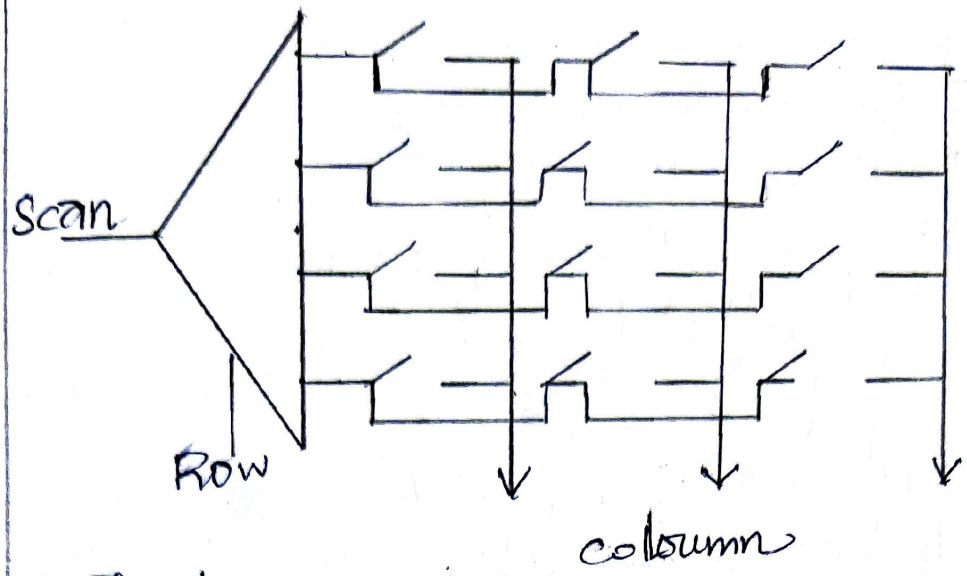
➡ The demultiplexer at the left side of the array selects the row to be read.

➡ When the key input is 1, that value is transmitted to one terminal of each key in the row

➡ If the switch in the column is depressed, the 1 is sensed at that column. Since only one switch in the column is activated, that value uniquely identifies a key.

➡ The row address & column output can be used for encoding, or circuit can be to give a different encoding.

A scanned key array

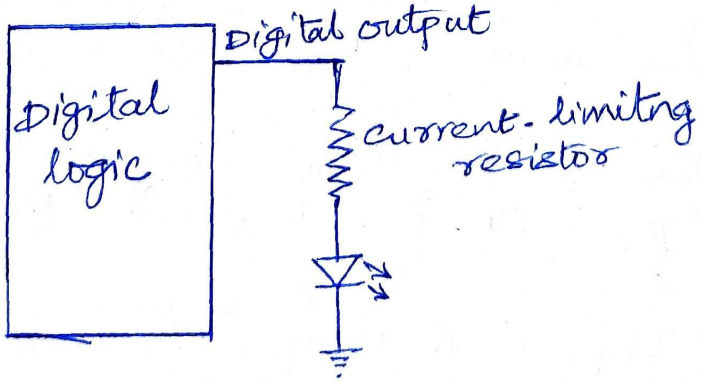


⇒ The keyboard microcontroller can be programmed to provide n-key rollover, so that rollover keys are sensed, put on a stack and transmitted in sequence as keys are released.

LEDs – (Light-Emitting Diodes)

⇒ It is a simple output device, a resistor is connected between the output pin & the LED to observe the voltage difference between the digital output voltage & the 0.7V drop across the LED.

⇒ When the digital output goes to 0, the LED voltage is in the device's off region and the LED is not on.



An LED connected to a digital output

Displays

⇨ A display device may be either directly driven or driven from a frame buffer.

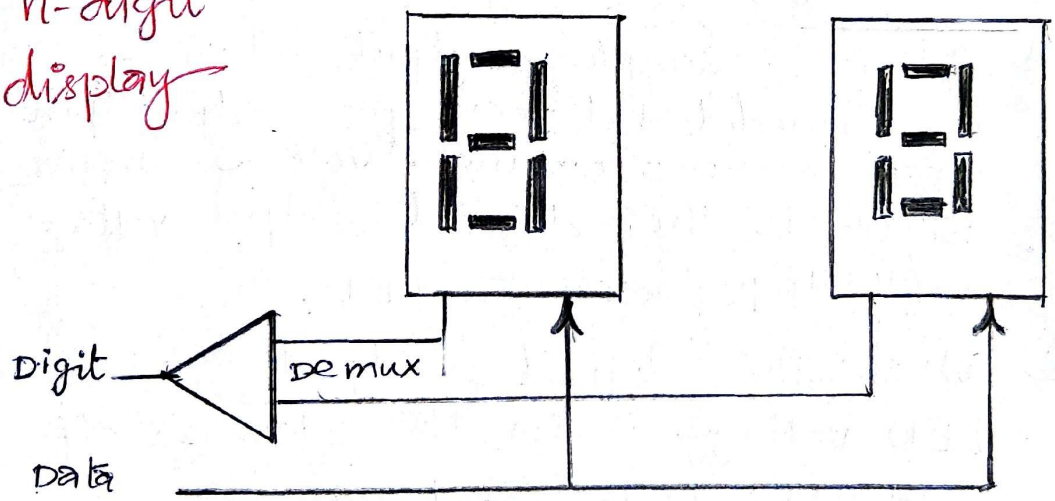
⇨ Small digits are driven by directly driven & large digits are driven by RAM frame buffer.

⇨ A single-digit display consists of seven segments; each segment may be either an LED or LCD element.

⇨ The digit input is used to choose which digit is currently being updated & the selected digit activates its display elements based on the current data value.

⇨ The display's driver is responsible for repeatedly scanning through the digits and presenting the current value of each to the display.

An n-digit display



⇨ A frame buffer is a RAM that is attached to the system bus.

⇨ The microprocessor writes values into the frame buffer in whatever order is desired.

⇨ The pixels in the frame buffer are generally written to the display in raster order by reading pixels sequentially.

⇨ Large displays are built using LCD. Each pixel in the display is formed by a single liquid crystal.

⇨ LCD display present a very different interface to the system because the array of pixel LCDs can be randomly accessed.

⇨ Early LCD panels were called passive matrix because they relied on a 2-Dimentional grid of wires to address the pixels.

⇨ Modern LCD panels use an active matrix system that puts a transistor at each pixel to control access to the LCD.

⇨ Active matrix displays provide higher constrast and a higher-quality display.

Touchscreens

➡ A touchscreen is an input device overlaid on an output device.

➡ The touchscreen registers the position of a touch to its surface.

➡ By overlaying this on a display, the user can react to information shown on the display.

➡ The two most common types of touchscreens are resistive & capacitive.

➡ A resistive touchscreen uses a 2-Dimentional volt-meter to sense position.

➡ The touchscreen consists of two conductive sheets seperated by spacer bar

➡ The top conductive sheet is flexible so that it can be pressed touch the bottom sheet

➡ A voltage is applied across the sheet; its resistance causes a voltage gradient to apper across the sheet

➡ The top sheet samples the conductive sheet's applied voltage at the contact position.

➡ The analog & digital converter is used to measure the voltage & resulting position.

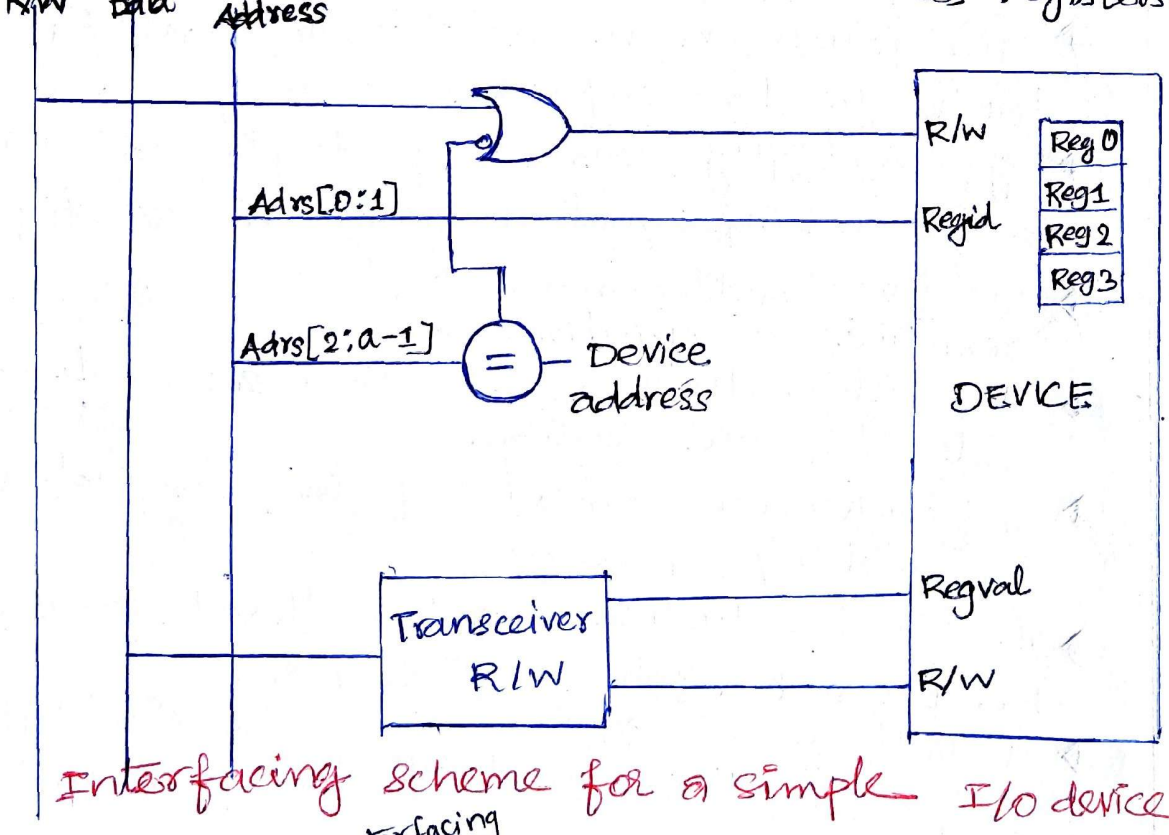
➡ The touchscreen alternates between x and y position sensing by alternately applying horizontal & vertical voltage gradients.

Component Interfacing

⇨ The component interfacing is including memory & I/O devices with bus system.

⇨ Device Interfacing

- Some I/O devices are designed to interface directly to a particular bus, forming glueless interfacing.
- The glue logic is required when a device is connected to a bus for which it is not designed.
- An I/O device typically requires a much smaller range of addresses than a memory, so addresses must be decoded much more finely.
- Some additional logic is required to cause the bus to read and write the device's registers.



Interfacing scheme for a simple I/O device interfacing

⇨ I/O devices may be obtained in two ways

1) Simple I/O
2) Memory mapped I/O

⇒ The device has four register that can be read & written by presenting the register number on the regid pins. Also asserting R/W as required and reading or writing the value on the regval pins.

⇒ To interface to the bus, the bottom two bits of the address are used to refer the registers within the device & the remaining bits are used to identify the device itself.

⇒ When the bus address can be set with switches & the bus address matches the device's.

⇒ The device's address can be set with switches to allow the address to be easily changed.

⇒ When the bus address matches the device's, the result is used to enable a transceiver for the data pins.

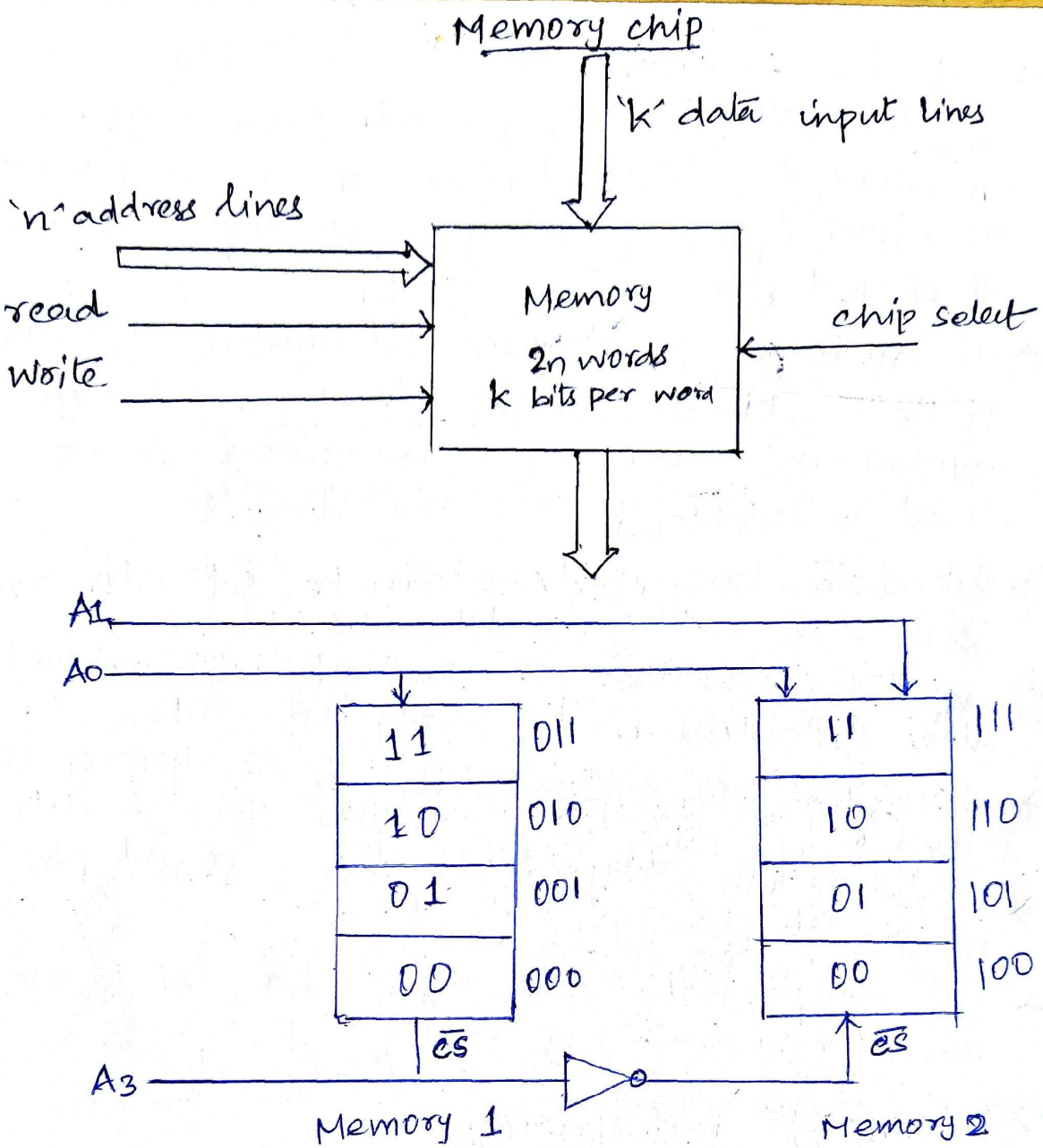
⇒ When the transceiver is disabled, the regval pin are disconnected from the data bus.

⇒ The comparator's output is also used to modify the R/W signal

⇒ Memory Interfacing

* If the memory size is exact then the interfacing is very simple.
* If we buy other than exact required size then we have to construct the memory to fit with the system

* In case of multiple chips simple circuit like NOT gate will not work
* In this case normally decoder circuits take 3-to-8 decoder. This circuit is called address decoder. There are two types
  1) partially decoding
  2) Fully decoding.



Designing with microprocessor

→ Architectures & Components
- Softwar
- Hardware

→ Some software is very hardware dependent.

→ Contains several elements in hardware platform they are
CPU
Bus
Memory
I/O devices : networking, sensors, actuators ect.

Software architecture

- Functional description must be broken into pieces
  - * division among people
  - * conceptual organization
  - * performance
  - * testability
  - * maintenance

Hardware and software architecture

- Hardware and software are intimately related.
- Software does not run without hardward.
- how much hardware you need is determined by the software requirements:
  - * speed
  - * memory

Evaluation boards

- Designed by CPU manufacturer or Others.
- It includes CPU, memory, some I/O devices,
- It may include prototyping section.
- CPU manufacturer often gives out evaluation board netlist
- It can be used as starting point for your custom board design.

Adding logic to a board

→ Programmable logic devices (CPLDs)
- provide low/medium density logic.

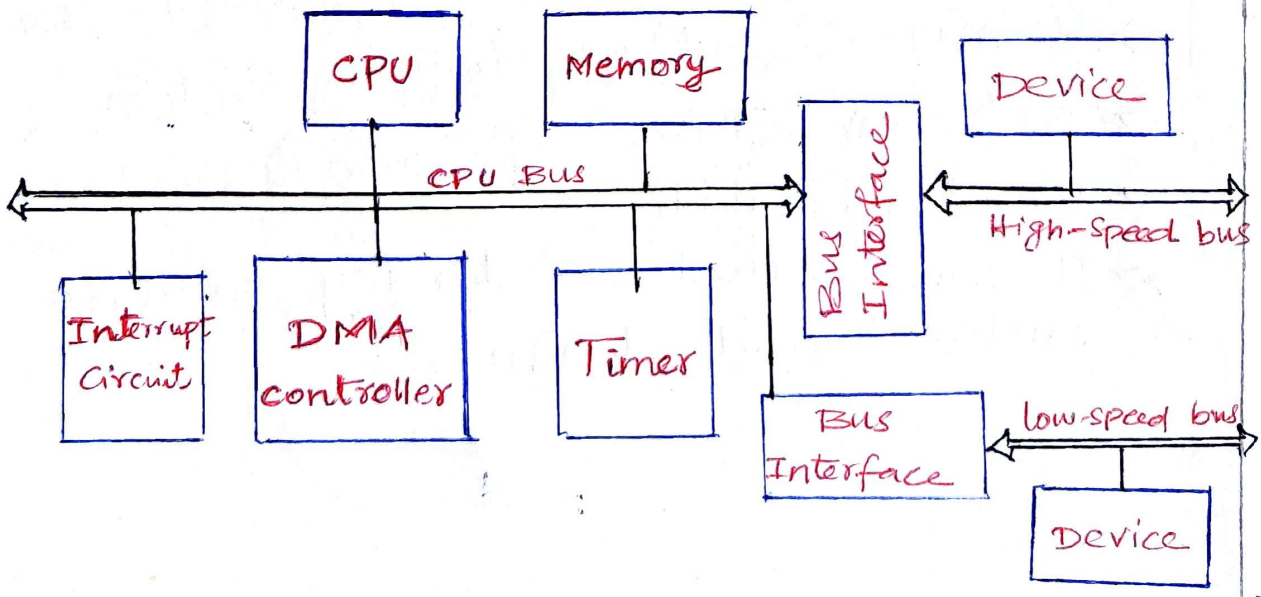
→ Field-programmable gate arrays (FPGAs)
- Provide more logic & multi-level logic.

→ Application-specific integrated circuits (ASICs)
- ASICs are manufactured for a single purpose.

→ If the PC is used as a platform, it takes the advantages
- cheap and easy to get
- rich and familiar software environment

and it takes the disadvantages
- requires a lot of hardware resources
- not well-adapted to real-time.

Typical PC hardware platform



Typical busses

➡ PCI : standard for high-speed interfacing
- 33 or 66 MHz
- PCI Express

➡ USB (Universal Serial Bus)
- Firewire
- relatively low-cost serial interface with high speed.

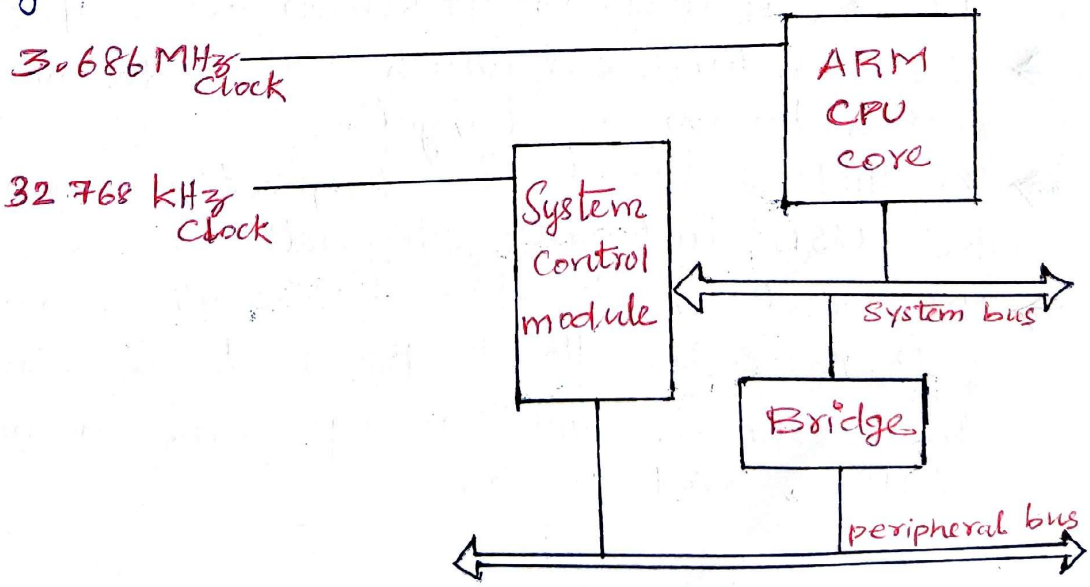
Software elements

➡ IBM PC uses BIOS (Basic Input Output System) to implement low-level functions :
- Boot-up process
- Minimal device drivers

➡ BIOS has become a generic term for the lowest-level system software.

Example : Strong ARM

➡ StrongARM system includes
* CPU chip (3.686 MHz clock)
* system control module (32.768 kHz clock)



→ The strong ARM provides a number of functions besides the ARM CPU.

→ The chip contains two on-chip buses:
  1) a high-speed system bus
  2) a lower-Speed peripheral bus.

→ The system control module contains the following peripheral devices.
  * A real-time clock
  * An operating system timer
  * 28 general-purpose I/Os (GPIOs)
  * An interrupt controller
  * A power manager controller
  * A reset controller that handles resetting the processor

→ The 32.768 kHz clock's frequency is choosen to be useful in timing real-time events.

→ The lower clock rate and therefore lower power consumption.

DEVELOPMENT AND DEBUGGING

Development Environments

→ A part of the software development on a PC or workstation is known as a **Host**.

→ The hardware on which the code will finally run is known as **target**.

→ The Host & target are frequently connected by the USB link like Ethernet.

→ The target must have a small amount of software to talk to the host system.
The software will take up some memory, interrupt vectors and so on.

➡ The host should be able to do the following.
* load programs into the target,
* Start and Stop program execution on the target
* examine memory & CPU register.

A Cross-Compiler : It is a compiler that runs on one type of machine but generates code for another.

- After compilation, the executable code is downloaded to the embedded system by a serial link or burned in a PROM & plugged in.

➡ In many causes, a test bench program can be built to help debug in embedded code.

Debugging Techniques

challegus :
- target system may be hard to observe;
- target may be hard to control;
- may be hard to generate realistic inputs;
- Setup sequence maybe complex.

⇨ The serial port found on most evalution bords is one of the most important hardware.

⇨ Breakpoint is an another important debugging tool.

⇨ A break point allows the user to stop execution examine system state and change state.

⇨ Replace the breakpointed instruction with a subroutine call to the monitor program.

ARM breakpoints

0X400	MUL r4, r6, r6	0X400	MUL r4, r6, r6
0X404	ADD r2, r2, r4	0X404	ADD r2, r2, r4
0X408	ADD r0, r0, #1	0X408	ADD r0, r0, #1
0X40c	B loop	0X40c	BL bkpoint

uninstrumented code — code with breakpoint

⇨ Breakpoint handler actions

- save registers.
- Allow user to examine machine.
- Before running, restore system state.
  + Safest way to execute the instruction is to replace it and execute in place.
  + Run another breakpoint after the replaced breakpoint to allow restoring the original breakpoint.

⇨ The microprocessor **in-circuit emulator** (ICE) is a specialized hardware tool that can help debug software in a working embedded system.

⇨ It allows you to stop execution, examine CPU state & modify the register.

⇨ The **logic analyzer** is an array of inexpensive oscilloscopes

⇨ The analyzer can sample many different signals simultaneously.

⇒ The logic analyzer records the values and then displays the result.

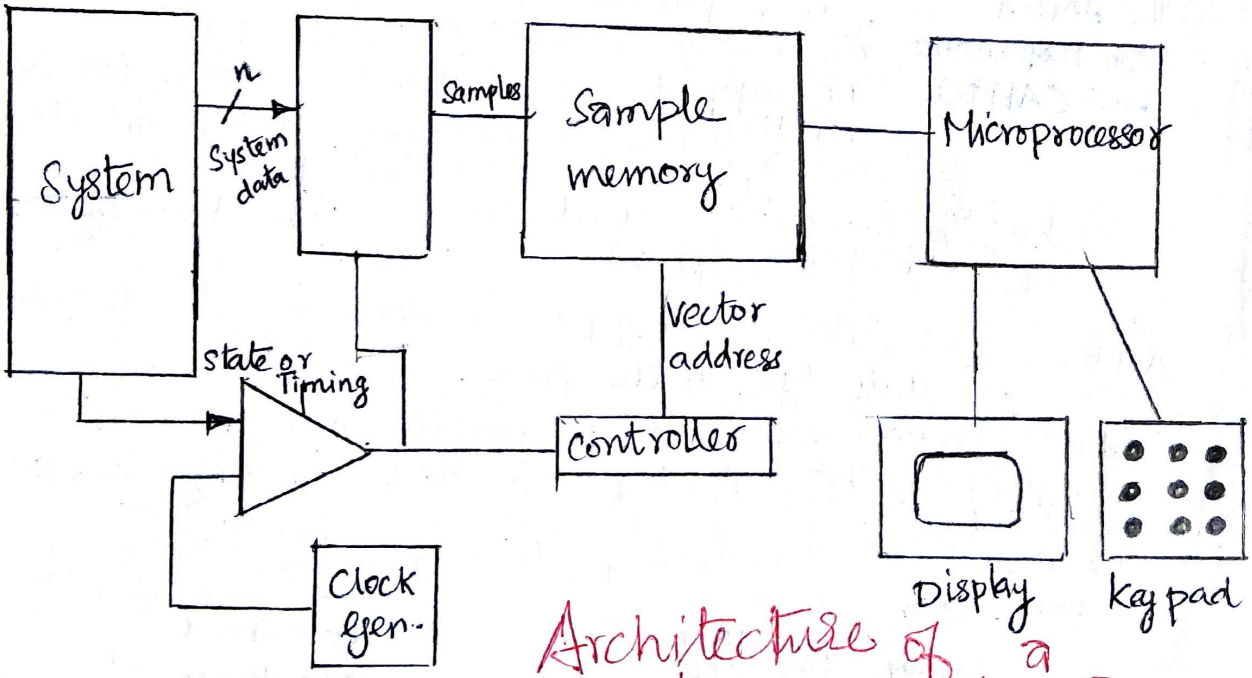
→ It can capture thousand of or even millions of samples of data.

→ It can acquire data in either of two modes that are called State & timing modes.

→ State mode uses the system's own clock to control sampling, so it samples each signal only once per clock cycle.

– Timing mode provides greater resolution in the signal for detecting glitches.

– state mode - used for sequentially oriented problems
– Timing mode - used for glitch-oriented debugging.



Architecture of a Logic Analyzer

⇒ logic analyzer gives a very good view of the externally visible signals. That information can be used for both functional and timing debugging.

Debugging challenges

→ logical errors in software can be hard to track down, but errors in real-time code can create problems that are even harder to diagnose

→ Bugs in drivers can cause non-deterministic behaviors in the foreground problem

→ Bugs may be timing-dependent.

ARM BUS

⇨ ARM has developed a seperate bus specification for single-chip systems

⇨ The AMBA bus supports CPUs, memories, & peripherals integrated in a system-on-silicon.

⇨ AMBA specification includes two buses.

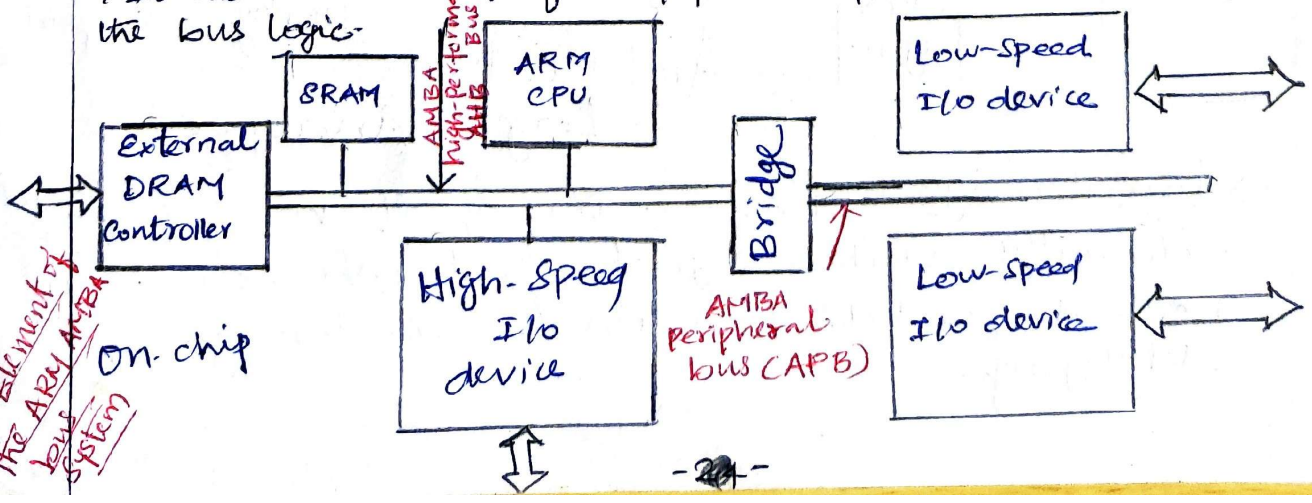
The AMBA High-Performance bus (AHB) : It is optimized for high-speed transfer & it is directly connected to the CPU. It supports several High-performance features: pipelining, burst transfer, split transactions & multiple bus master.

⇨ A bridge is can be used to connect AHB to an AMBA peripheral bus (APB)

AHB Bus : Simple & easy to implement : it also consumes relatively little power.

⇨ AHB assumes that all peripherals act as slave, simplifying the logic required in both peripherals & the bus controller.

⇨ It also does not perform pipelined operations, which simplyfies the bus logic.



* leading byte determines the outgoing port.
* MSB of each header byte distinguishes between host & switch packets.
* Variable payload length & 8-bit CRC.
* Switches; use blocking-cut-through routing
  Each contains two custum VLSI chips
  - crossbar-switch
  - Dual interface
* Topology is arbitrary.
* Myrinet is flexible & fast.
* It also affordable systems with acceptable performance.
* The data payload may be of variable length & the packet includes error correction codes.

Internets

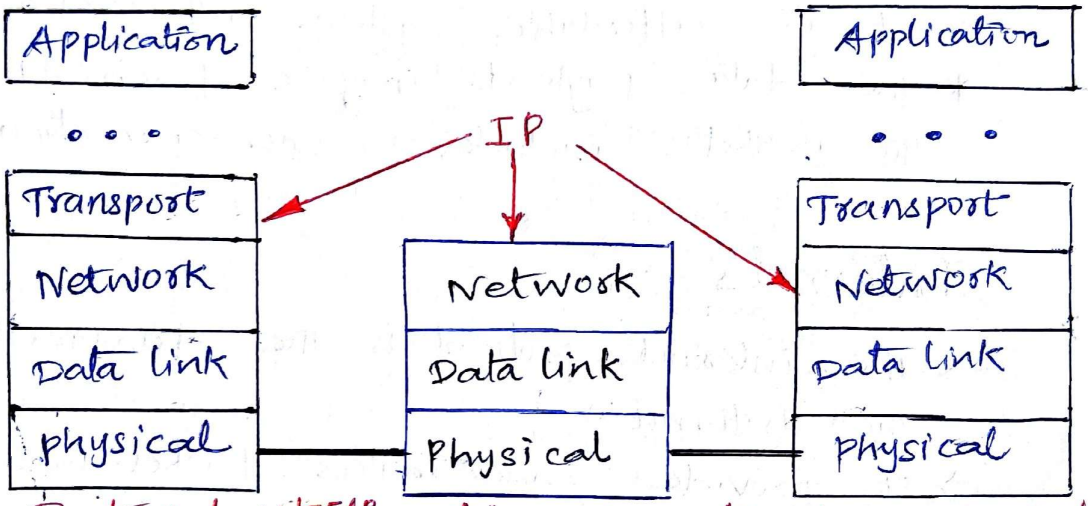
⇨ The internet protocol is the fundametal protocol on the Internet.
⇨ It provides connectionless, packet-based communication
⇨ Industrial automation uses a good application area for Internet-based embedded systems.
⇨ Internet protocol is very important in embedded computing.
⇨ The internet packet will travel over several different networks from source to destination.
⇨ The IP allows data to flow smoothly through these networks from one user to another.
⇨ IP works at network layer.
⇨ IP & individual networks are illustrated in figure present in next page
⇨ When node A wants to send data to node B, the application's data pass through several layers of the protocol stack to get into the Internet Protocol.

⇨ IP creates packets for routing to the destination, which are then send to the data link & physical layers.

⇨ A node that transmits data among different types of networks is known as a router.

⇨ In general, a packet may go to through several routers to get into its destination.

⇨ At the destination, the IP layer provides data to the transport layer & then application (layers).



Protocol utilization in Internet communication.

⇨ As the data pass through several layers of the protocol stack, the IP packet data are encapsulated in packet formats appropriate to each layer.

⇨ The basic format of an IP Packet is shown figure available in next page.

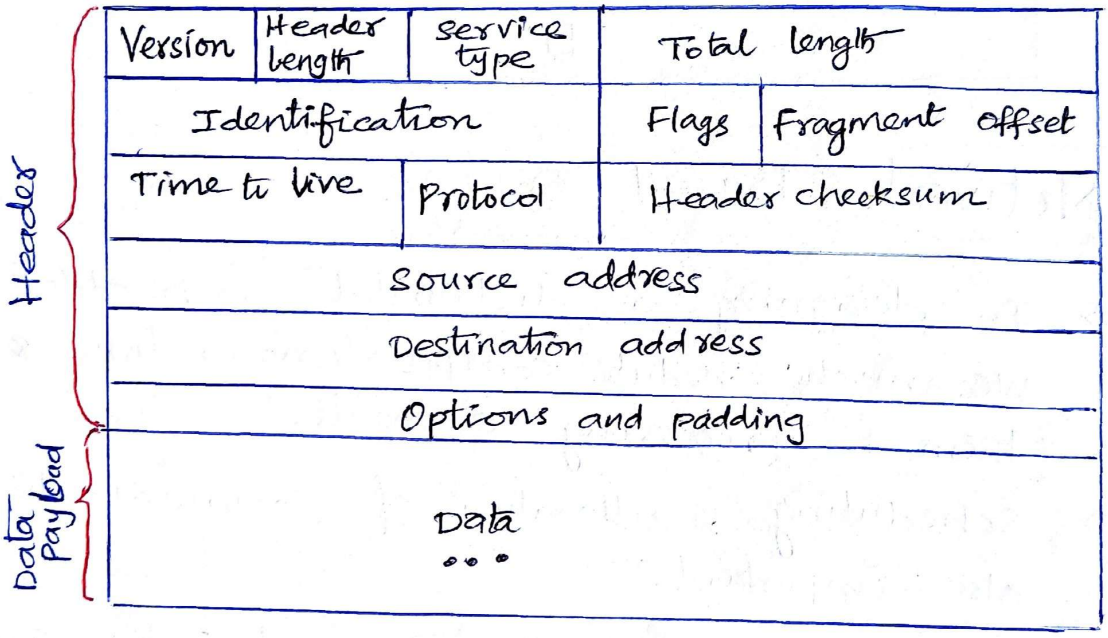
⇨ The header & data payload are both of variable length. The maximum total length of the header & data payload is 65,535 bytes.

⇨ The IP address is typically written in the form of xxx.xx.xx.xx The names by which users and applications typically referred to Internet nodes. such as foo.baz.com, are translated into IP addresses via calls to a Domain Name Server (DNS), one of the higher-level services built on top of IP.

⇒ IP works on network layer. so, it does not guarantee that a packet is delivered to its destination. Also packets may arrive out of order. This is referred to as best-effort routing.

⇒ Packets being routed along very different paths with different delays, real-time performance of IP can be hard to predict.

⇒ IP packet structure



⇒ The internet also provides higher-level services built on top of IP. Transmission control Protocol (TCP) is one such example.

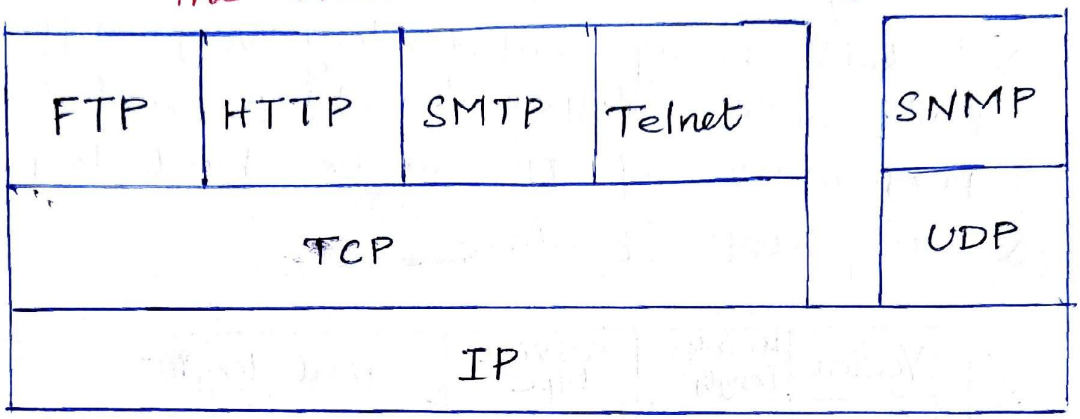
⇒ It provides connection-oriented service that ensure that data arrive in appropriate order.

⇒ The diagram in next page shows the relation-ship between IP & high-level Internet services.

⇒ Using IP as the foundation, TCP is used to provide File Transfer Protocol (FTP) for batch file transfers, Hypertext Text Transport protocol (HTTP) for World Wide Web services (WWW) Simple Mail Transfer Protocol (SMTP) for email. and Telnet for virtual terminals.

⇨ A seperate transport protocol, User Datagram Protocol (UDP), is used as the basic for the network management services provided by the Simple Network Management Protocol (SNMP).

The Internet service stack



Network-Based Design

⇨ In designing a distributed embedded system, we must schedule computations in time & accelarate them to processing elements.

⇨ scheduling & allocation of communications are also important.

⇨ Many embedded system networks are designed for low cost and therefore reduce the communication speed. It may lead the bottleneck in system design.

Communication Analysis

⇨ To analyze the performance of network we must determine the delay incurred by transmitting messages.

⇨ Let us assume for the moment that messages are sent reliably, no retransmit a message & no contention can be modeled as $t_m = t_x + t_n + t_r$

t_x = transmitter-side overhead
t_n = network transmission time
t_r = Receiver side overhead

Timers / counters:

Timers – to generate a time delay

Counters – to count events happening outside the microcontroller

In 8051 two counters / timers are there, T_0 and T_1. It is programmed to count internal clock pulses, acting as a timer or may be programmed to count external pulses acting as a counter.

Registers of timer 0:

16 bit - register divided into two 8 bit - registers (TL0) and (TH0)

Registers of timer 1:

16 bit - register divided int TL1 and TH1.

Control Registers:

Timer / counter action is controlled by

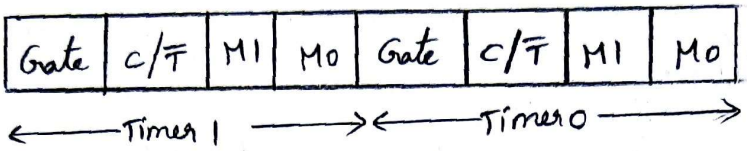
* TMOD register:

To set the various timer operation modes.

Dedicated solely to two timers and can be

considered to be two duplicate 4-bit registers, each of which controls the action of one of the timers.

It is not bit addressable.



Gate :

OR gate enable bit that controls RUN/STOP of timer 1/0

Timer/counter enabled while TR 1/0 is set and $\overline{INT}$ 1/0 is high

C/$\overline{T}$: (clock/Timer)

Decide whether the timer is used as timer/counter.

C/$\overline{T}$ = 0 → timer

C/$\overline{T}$ = 1 → counter

M1:

Timer/counter operating mode select bit 1.

Set/cleared by program to select mode

M0 :

Timer/counter operating mode select bit 0.

Set/cleared by program to select mode

TCON Register:

Has control bits and flags for

timer in upper nibble

external interrupts in lower nibble

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

TF1:

↳ Timer 1 overflow flag

↳ Set by hardware when timer/counter 1 overflows.

↳ cleared when processor vectors to execute ISR location 001B

TF0:

↳ Timer 0 overflow flag

↳ Set to enable timer/counter 0 overflow

↳ cleared → to execute ISR location 0001

IT1:

Interrupt 1 type control bit

Set → enable external interrupt 1 while falling edge signal

cleared → low-level signal on external interrupt 1.

IT0: Interrupt 0 type control bit

Set → enable interrupt 0 falling edge

cleared → low signal on external interrupt 0.

TR1:

↳ Timer 1 run control unit

↳ Set by program to enable timer to count

↳ cleared by program to halt the timer.

TR0:

↳ Timer 0 run control flag

↳ Set → enable timer to count

↳ cleared → to halt timer

IE1:

↳ External Interrupt 1 edge flag

↳ Set → high to low edge signal is received

↳ cleared → ISR located at 0013

IE0:

External interrupt 0 edge flag

Set → high to low edge signal is received on $\overline{INT_0}$

cleared → ISR located at 0003

Timer mode of operation:

4 modes are there

Timer Mode 0:

Has 13 bit timer

THx register – 8 bit counter & TLx register – 5 bit counter.

Values of 000H to 1FFFH. I/p divided by 32 in TL so TH counts original frequency reduced by 384.

Timer Mode 1:

Has 16 bit timer → THx register of 8 bit counter and TLx register as 8 bit counter allows values of 0000 to FFFF

Pulse input divided to 256 in TLx so THx counts frequency reduced by 3072.

Timer Mode 2:

Has 8 bit timer allow values of 00H to FFH to be loaded into timer's register THx. After THx is loaded with the 8 bit value, the microcontroller give copy of it to TLx. Then timer starts.

It is an auto reload feature: TLx will count up from the number in overflow and be initialized again with contents of THx.

Timer Mode 3:

Timer 0 and 1 may be programmed to be mode 0, 1 and 2. But mode 3 is chosen for timer 0. placing timer 1 in mode 3 causes it to stop counting, the control bit TR1 and timer 1 flag TF1 are then used by timer 0.

Timer 0 in mode 3 becomes two completely seperate 8-bit counters. TH0 receives the timer clock under the control of TR1 only and sets the TF1 flag when it overflows.

UART

* Universal Asynchronous Receiver / Transmitter.

* Traditional Definition : converts parallel (8 bit) data to serial data and vice versa.

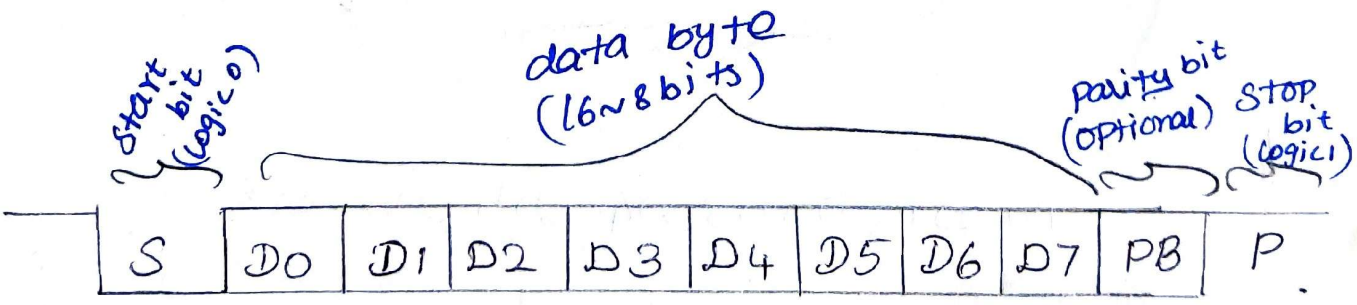
↳ UART is a simple half duplex, asynchronous serial Protocol.

↳ simple connections between two equivalent nodes.

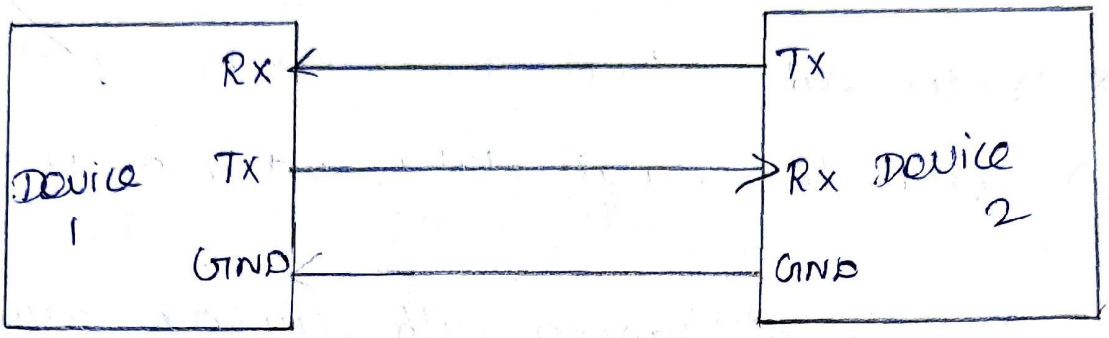
↳ Any node can initiate communication.

↳ Since connection is half duplex, the two lanes of communication are completely independent.

Packet format:



Connections for UART:



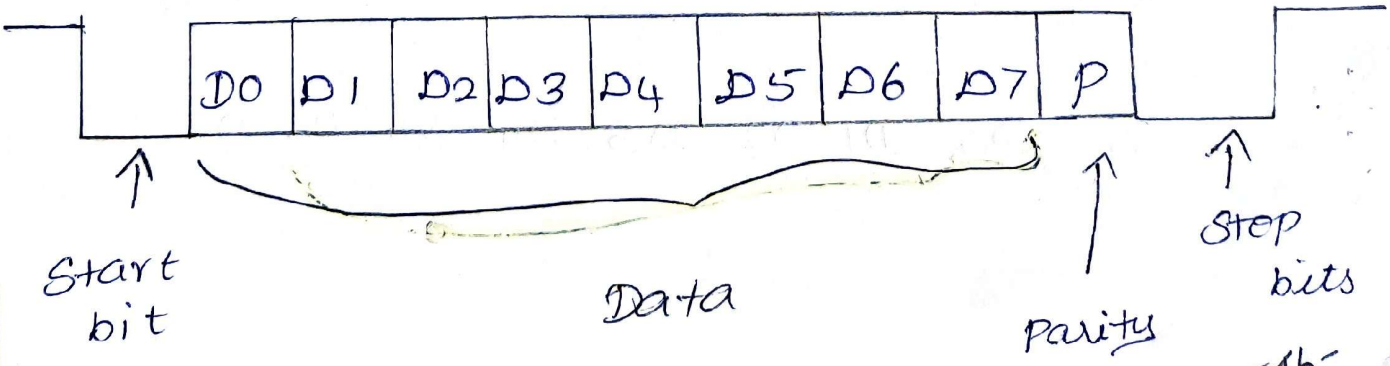
UART characterstics:

* The speed of communication (measured in bauds) is predetermined on both ends.

* A general rule of thumb is to use 9600 bauds for wired communication.

* UART implements error-detection in the form of parity bit.

Parity bit



* Parity bit is HIGH when number of 1's in the Data is odd.

* Respectively, it is LOW when number of 1's in the Data is even.

Baud Rate Generator:

* Used to generate the baud rates for both the transmitter and receiver.
* Not required for any other function including reads and writes.
* Crystal or External clock.
* 16 bit divisor Programmed in DLM / DLL registers.

$$\text{Baud Rate} = \frac{\text{clock frequency}}{(\text{Sampling Rate}) \times (\text{Divisor})}$$

Sampling Rate is 16.

Transmitter:

- Parallel to serial communication.
- Non FIFO mode
  - * Transmit holding Register (THR) and transmit shift register (TSR)
- FIFO mode
- 16x timing for bit shifting
- character framing
- Parity insertion
- TX FIFO interrupt and Status.

Receiver:

- Serial to Parallel conversion.
- Non FIFO mode (RHR and RSR)
- FIFO mode (RX FIFO and RSR)
- 16x timing clock for mid bit sampling.
- Start bit detection and verification.
- Rx FIFO is 11 bits wide
  - * 8 data bits
  - * 3 error or error tags.