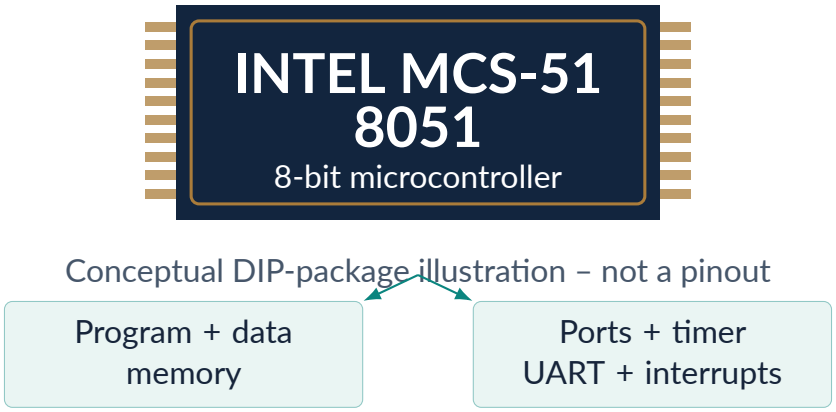
ZENITH STUDY HUB

Microprocessor & Micro-controller

UNIT III | INTEL 8051

Architecture, memory, timers, interrupts and serial communication



Official Unit III syllabus: Microprocessor vs Microcontroller; CISC vs RISC; 8051 Architecture; Memory Organization of 8051; 8051 Instruction Set & Addressing; Timers & Counters; Interrupts & Serial Communication.

Scope of THIS learning edition: all seven official Unit III headings are covered in the same PDF, with annotated diagrams, calculations and original instructional examples. This is a final visual and source-review candidate; native assembler and hardware validation remain outstanding.

Source basis: SPPU PCC302-ROA official TE 2024 Pattern syllabus; Atmel *8051 Microcontrollers Hardware Manual*, Rev. 4316E–8051–01/07; cross-reference with Intel MCS-51 family documentation.

Prepared by: Om Deshmukh **21 November 2026**

Study contract and roadmap

Reading route: 1. Build intuition with the shaded explanation. 2. Copy each labelled diagram. 3. Work through the numerical calculation. 4. Trace assembly one instruction at a time. 5. Write an exam answer without looking.

Contents

1 Microprocessor versus microcontroller 3

2 CISC versus RISC: what the words actually mean 5

3 Architecture of the classic Intel 8051 6

4 8051 memory organization: a visual map 8

5 8051 instruction set and addressing: first principles 10

6 Timers and counters: thinking in machine cycles 13

7 8051 interrupts: when hardware requests attention 16

8 Serial communication: sending useful bytes between devices 20

9 Integrated robotic-control example and revision 25

10 Practice questions and answer checkpoints 28

In simple words
The 8086 is a microprocessor: in a typical system it relies on external chips for RAM, program storage and I/O. The classic 8051 is a microcontroller: it combines a CPU with memory and interfaces in one package. That is

the main conceptual step from Unit II to Unit III.

Official Unit III scope	Coverage and status
Microprocessor versus microcontroller	Section 1, comparisons and applied examples
CISC versus RISC	Section 2, distinctions and limitations
8051 architecture	Section 3, functional diagram and registers
Memory organization	Section 4, annotated RAM/SFR maps and examples
Instruction set and addressing	Section 5, examples and short programs
Timers and counters	Section 6, TMOD/TCON, modes, reloads and programs
Interrupts	Section 7, sources, IE/IP, vectors and ISR
Serial communication	Section 8, UART modes, SCON, frames, baud and code

Hardware version matters. This guide introduces the *original/classic 8051 core*. Modern 8051-compatible chips can contain more ROM/RAM, extra timers, extra interrupt sources or different timing cores. Check the exact device data sheet before applying values to a real board.

1 Microprocessor versus microcontroller

In simple words
A microprocessor is a CPU intended to become part of a larger computer. A microcontroller is a small computer built around a CPU, usually with RAM, nonvolatile program storage and input/output peripherals on the same chip.

Property	8086 example	Classic 8051 example
Main function	General computation CPU	Embedded control CPU with peripheral blocks
Data width	16-bit ALU and registers	8-bit core/accumulator
Program storage	External memory in typical configuration	Original 8051 incorporates 4 KiB on-chip ROM; can use external code memory
Data storage	External RAM via system bus	128-byte internal data RAM plus external data-memory option
Peripherals	External companion chips required	Four 8-bit ports, two timers/counters and serial interface on the classic core
Common design goal	Computing with external system components	Read sensors and command outputs efficiently

A microcontroller is not automatically more powerful, nor is a microprocessor incapable of control. Both can run control software. The difference is primarily *integration* and the intended hardware/software environment. A modern system-on-chip may blur the boundary; these are concepts, not absolute marketing categories.

STEP-BY-STEP EXAMPLE: Choosing a controller for a small conveyor

A conveyor has a proximity switch, a status LED and a relay-driver command. A suitable microcontroller can read the switch on an input pin, run a control loop and switch the driver without a separate system bus. An 8086 system can also do that job, but it normally needs external memory and interface components. Real designs require isolated, protected drivers; a CPU or microcontroller pin must not power a motor directly.

Exam answer format: define both, draw the block diagrams, compare RAM/ROM/I-O integration, give one industrial example and explain why a microcontroller reduces board complexity for a narrow control task.

2 CISC versus RISC: what the words actually mean

In simple words
CISC generally offers a more varied instruction set with instructions of differing complexity. RISC designs emphasize comparatively regular, simple instructions and often load/store execution. These are design families, not a guarantee about speed or instruction count in every modern device.

The original 8051 is commonly classified as a CISC-style microcontroller: it supports rich addressing modes, one-bit Boolean operations, multi-byte instruction encoding, arithmetic, logic and branch forms. Some instructions operate directly on internally addressed memory or special-function registers. A modern RISC processor may use a uniform register-oriented design and rely on more simple instructions for a complex task.

Typical property	CISC-style approach	RISC-style approach
Instruction repertoire	Includes complex or specialized operations	Emphasizes simpler regular operations
Addressing	Often multiple addressing forms	Often more constrained memory access
Hardware realization	May involve more elaborate instruction decoding	Often optimized for regular decode/execution
Best caution	Not necessarily slower	Not necessarily faster

Exam trap: do not write “CISC needs one clock per instruction” or “RISC always completes everything in one cycle.” Cycle counts depend on the particular processor, instruction, memory system and implementation.

3 Architecture of the classic Intel 8051

In simple words
The 8051 repeatedly fetches instructions from program memory, decodes them in its 8-bit CPU, reads/writes data in internal RAM or special registers, and controls pins and peripherals. A program counter determines where the next instruction comes from.

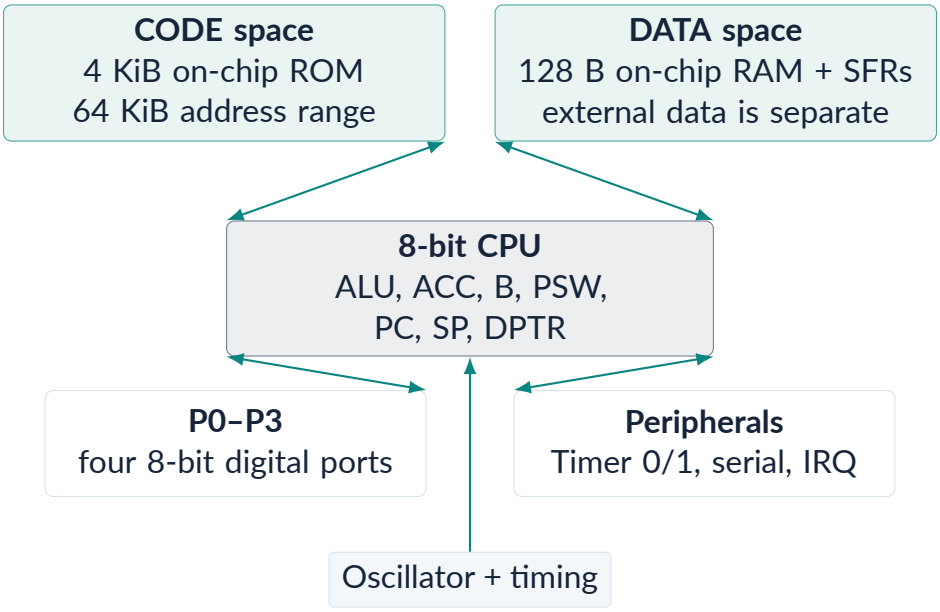


Figure 1. Original, simplified 8051 block diagram. The code and data memory spaces are logically separate; this is not a transistor-level drawing.

3.1 The essential CPU registers

Register	Width	Use
A (ACC)	8 bits	Main arithmetic and logic accumulator
B	8 bits	Multiply/divide companion; otherwise general data storage
PSW	8 bits	Carry, auxiliary carry, overflow, parity and bank-selection status
PC	16 bits	Program-memory address of the next instruction
DPTR	16 bits	Data pointer, made of DPH and DPL; used by MOVX/MOVC
SP	8 bits	Internal RAM stack pointer; reset default 07H on classic 8051
R0–R7	8 bits each	Current active register bank; R0/R1 also support indirect internal-RAM addressing

STEP-BY-STEP EXAMPLE: Accumulator, B and flags in a simple multiplication

The classic 8051 instruction `MUL AB` multiplies two unsigned 8-bit numbers in A and B and places the 16-bit result in B:A (B high byte, A low byte). If A=20H (=32) and B=10H (=16), their product is 0200H, so B=02H and A=00H. Reading only A incorrectly loses the high byte. Do not confuse this with 8086 MUL implicit register conventions.

3.2 Ports: 32 I/O pins are not motor-power outputs

Four nominally 8-bit ports P0, P1, P2 and P3 provide 32 digital port lines. Many P3 pins have alternate hardware functions: P3.0 RXD, P3.1 TXD, P3.2 INT0, P3.3 INT1, P3.4 T0, P3.5 T1, P3.6 WR and P3.7 RD. On classic external-memory interfaces, P0 and P2 help carry multiplexed low and high address/-data functions. P0 output-high behavior differs from the quasi-bidirectional P1/P2/P3 arrangements and often requires external pull-ups when used as ordinary I/O.

Robotics example: P1.0 can send a logic-level command to a suitable transistor/motor-driver input. A protective driver and separate motor supply provide current; the 8051 port only selects a control level. P3 serial pins can connect to a proper voltage-level transceiver, not directly to an arbitrary RS-232 voltage.

4 8051 memory organization: a visual map

In simple words
The 8051 has separate logical code and data address spaces. Inside the standard 8051 data map, lower addresses select internal RAM, while addresses 80H–FFH are special-function registers when accessed directly. Their uses are not the same as ordinary RAM bytes.

4.1 Classic 128-byte internal RAM layout

00H–1FH	Four register banks: R0–R7
20H–2FH	Bit-addressable: 128 bits within 16 bytes
30H–7FH	General-purpose internal RAM

Direct addresses 80H–FFH identify SFRs; they are not ordinary RAM on the original 8051.

Figure 2. Original 8051 internal DATA RAM map, showing actual hexadecimal boundaries and byte/bit distinction.

4.2 Register banks 00H–1FH

Internal addresses 00H–1FH form four banks of eight registers each. Bank 0 occupies 00H–07H, bank 1 08H–0FH, bank 2 10H–17H, and bank 3 18H–1FH. The two PSW bank-select bits RS1 and RS0 determine which physical bytes R0–R7 refer to. On reset, bank 0 is normally selected.

STEP-BY-STEP EXAMPLE: Which byte is R3 in bank 2?

Bank 2 starts at 10H. R3 is the fourth register, so its direct internal-RAM ad-

dress is 13H. The same instruction `MOV A,R3` reads a *different* physical byte if the PSW bank selection changes.

4.3 Bit-addressable RAM 20H–2FH

Sixteen bytes contain 16×8=**128 independently addressable bits**. This is useful for switch status, control flags and simple Boolean logic. It does not mean 128 additional bytes: the 128 individual bits occupy the same 16 internal-RAM bytes from 20H through 2FH.

STEP-BY-STEP EXAMPLE: An individual bit in RAM

A bit in the first bit-addressable byte 20H can be set or cleared by bit-oriented instructions. Changing a single bit does not require deliberately overwriting the other seven bits of the byte. However, a program still must consider whether an interrupt or other task accesses the same status field.

4.4 SFR addresses 80H–FFH

Special-function registers control hardware blocks and CPU state: P0=80H, SP=81H, DPL=82H, DPH=83H, PCON=87H, TCON=88H, TMOD=89H, TL0=8AH, TL1=8BH, TH0=8CH, TH1=8DH, P1=90H, SCON=98H and SBUF=99H. Other common SFR addresses include P2=A0H, IE=A8H, P3=B0H, IP=B8H, PSW=D0H, ACC=E0H and B=F0H in the classic core. Some addresses in 80H–FFH are unimplemented/reserved; do not assume every number names a usable SFR.

Crucial distinction. Direct `MOV A,90H` reads P1 (an SFR), not an ordinary internal-RAM byte at address 90H. Indirect access via `@R0`/`@R1` to upper internal RAM applies only to devices that actually provide that extra RAM. The original basic 8051 has 128 bytes of internal RAM.

4.5 Code memory versus external data memory

The classic architecture allows separate program code and external data spaces, each with a 16-bit address range of up to 64 KiB. The original 8051 implements 4 KiB on-chip program ROM, but external code memory can also be connected depending on EA pin and system configuration. `MOVC` fetches constants from code space; `MOVX` transfers to/from external data space. They are NOT interchangeable instructions.

Instruction	Address space	Meaning
MOV A,30H	Internal direct	Read internal RAM byte 30H into A
MOV A,@R0	Internal indirect	Read internal RAM byte whose address is in R0
MOVX A,@DPTR	External data	Read external data byte from 16-bit DPTR address
MOVC A,@A+DPTR	Code memory	Read constant byte from code address DPTR+A

5 8051 instruction set and addressing: first principles

In simple words
An addressing mode answers WHERE the operand comes from. An instruction tells you WHAT happens to it. The 8051 has register, immediate, direct, register-indirect and indexed code-memory forms.

Mode	Instruction example	Interpretation
Immediate	`MOV A,#25H`	Load literal byte 25H into A
Register	`MOV A,R3`	Read R3 in current register bank
Direct	`MOV A,30H`	Read byte from internal direct address 30H
Register indirect	`MOV A,@R0`	R0 holds an internal data address
Indexed code	`MOVC A,@A+DPTR`	Read program-memory lookup table byte
External data	`MOVX A,@DPTR`	Read external RAM/device mapped to DPTR

Assembler notation: the # sign marks immediate data in 8051 assembly, unlike the MASM hexadecimal suffix used for numeric literals in many

8086 examples. Using `MOV A,30H` instead of `MOV A,#30H` changes the meaning completely.

5.1 One picture: how addressing selects different hardware

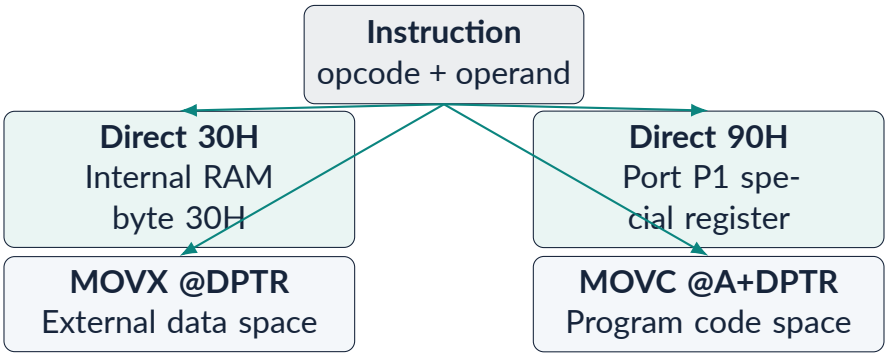


Figure 3. Effective addressing: the instruction mnemonic and operand form select RAM, SFR, external data or code memory.

5.2 Instruction group map

Data transfer: MOV, MOVC, MOVX, PUSH, POP, XCH. Arithmetic: ADD, ADDC, SUBB, INC, DEC, MUL AB, DIV AB. Logical: ANL, ORL, XRL, CPL, CLR, SETB. Control/branch: SJMP, LJMP, AJMP, LCALL, RET, RETI, DJNZ, CJNE, conditional bit jumps. Rotate/bit operations: RL, RLC, RR, RRC, JB, JNB and JBC. Each instruction has defined operand forms and flag effects; do not apply general x86 instruction syntax to an 8051.

5.3 Program A: add two bytes from internal RAM

Data contract: internal RAM at 30H contains F8H and 31H contains 15H. Store the low sum in 32H and a carry byte (00H/01H) in 33H. Assume appropriate memory has been initialized externally or by a surrounding program.

```
    MOV A,30H          ; A = first byte
    ADD A,31H          ; A = low sum, C = carry
    MOV 32H,A          ; save low byte
    MOV 33H,#00H       ; clear carry output byte
    JNC FINISH         ; if no unsigned carry, stop
    MOV 33H,#01H       ; otherwise mark carry
FINISH: SJMP FINISH    ; illustrative halt loop
```

STEP-BY-STEP EXAMPLE: Full trace

F8H+15H=10DH. The accumulator stores 0DH and the carry flag C=1. The MOV instructions do not modify carry, so JNC is not taken. Memory[32H]=0DH and memory[33H]=01H. A practical firmware program would return to a caller or continue control logic rather than sit forever in the teaching `SJMP FINISH` loop.

Worked answer: Result 10DH represented as 01H:0DH (high carry byte 01H, low byte 0DH). Check that the CPU and the intended application interpret the two output bytes correctly.

5.4 Program B: test a switch bit without disturbing others

A bit at internal bit-address 00H corresponds to the first bit of the bit-addressable RAM region. `JB 00H,ACTIVE` jumps when it is set, while `JNB 00H,INACTIVE` jumps when it is clear. A real switch on a port requires a specified pin, electrical pull-up, debouncing strategy and correct SFR/bit address; do not assume an arbitrary byte in RAM is physically connected to a switch.

```
    SETB P1.0       ; input: write 1 to P1.0 latch
    CLR  P1.1       ; output: driver command OFF
POLL: JNB  P1.0,ACTIVE  ; active-low sensor
      CLR  P1.1     ; OFF when sensor is high
      SJMP POLL
ACTIVE: SETB P1.1   ; ON when sensor reads low
      SJMP POLL
```

Circuit assumption: The P1.0 signal is active-low with appropriate input bias, and P1.1 only controls a protected logic-level driver. This is a polling example, not a motor safety circuit or debounced input implementation.

Figure 4. Program B uses P1.0 and P1.1 SFR bit addresses, unlike the preceding generic internal-RAM bit-address example.

6 Timers and counters: thinking in machine cycles

In simple words
A timer counts pulses generated by the microcontroller clock. A counter counts qualifying edges arriving at a dedicated external pin. The same hardware can perform either job; the C/T bit in TMOD selects which input is counted.

6.1 Start with a precise hardware assumption

An **original 12-clock-cycle 8051** increments a timer once per machine cycle when configured for timer operation: $f_{\text{tick}} = f_{\text{osc}}/12$ and $t_{\text{tick}} = 12/f_{\text{osc}}$. At $f_{\text{osc}} = 12$ MHz, one timer tick is $1\ \mu$s. At 11.0592 MHz, each tick is $1.0850694\ \mu$s. Modern single-cycle/6-clock 8051 derivatives can use different timing: *never reuse these formulas without checking the actual chip*. A polling loop also takes time, so an ideal counter overflow is not necessarily an exact wall-clock pin pulse.

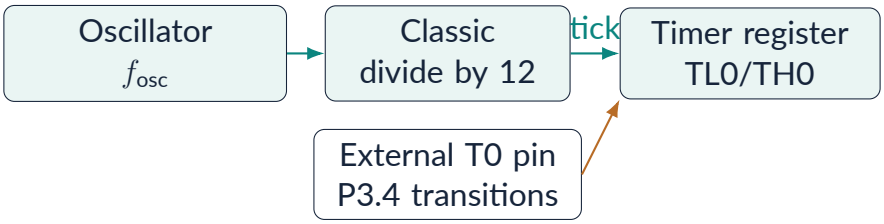


Figure 5. Timer clock sources. Classic timer mode counts oscillator/12; counter mode samples T0/T1 pin events.

6.2 TMOD and TCON: learn the bit meaning, not only the hex value

TMOD (89H) is divided into two four-bit fields: the upper nibble controls Timer 1, the lower nibble Timer 0. Each field is $(\text{GATE}, \text{C}/\overline{\text{T}}, \text{M1}, \text{M0})$ from its most significant to its least significant bit. GATE=0 lets TRx control running; GATE=1 additionally requires the external interrupt pin to be high. C/T=0 selects an internal timer; C/T=1 selects external events.



Figure 6. TMOD (89H) bit fields, drawn with readable bit numbers. C/T=0: timer clock; C/T=1: external event count.



Figure 7. TCON (88H): timer overflow/run flags at the left; external-interrupt status and trigger selection at the right.

TCON (88H) contains, from bit 7 to bit 0: TF1, TR1, TF0, TR0, IE1, IT1, IE0 and IT0. TFx records a timer overflow; TRx starts/stops that timer. IT0/IT1 select edge- versus level-triggered external interrupt behavior (subject to actual pin conditions). **Common confusion:** IE (A8H) is the interrupt-enable SFR; IE0 and IE1 are *flags* in TCON, not that SFR.

6.3 Compare the timer modes

Mode	Counter width	Behavior and engineering use
0	13-bit	Legacy 13-bit count, limited range; rarely preferred for new firmware.
1	16-bit	Counts from preload to FFFFH then overflows; useful for longer delays and event counts.
2	8-bit auto-reload	TLx increments and reloads from THx after overflow. Ideal for periodic events or a baud generator.
3	Split Timer 0	TL0 and TH0 behave as separate 8-bit timers with distinct control/overflow arrangements. Interacts with Timer 1 resources; consult the manual.

A subtle distinction: A 16-bit timer in mode 1 does *not* automatically reload itself. An ISR or polling code must write THx/TLx for the next inter-

val. Mode 2 is automatic, but its interval is at most 256 ticks on the classic device.

6.4 General delay equation, with units

For a b-bit up-counter, an initial preload P, and $N = 2^b - P$ ticks to overflow, the nominal delay is

$$t_{\text{delay}} = N \frac{12}{f_{\text{osc}}} \quad \Longrightarrow \quad P = 2^b - \frac{t_{\text{delay}} f_{\text{osc}}}{12}.$$

Choose N as an integer, verify $1 \le N \le 2^b$, and split a 16-bit preload into TH $= \lfloor P/256 \rfloor$ and TL $= P \bmod 256$. If the requested tick count is non-integer, quantify the rounding error rather than pretending it is exact.

STEP-BY-STEP EXAMPLE: Mode 1, nominal 1.000 ms at 12 MHz

1. $t_{\text{tick}} = 12/(12 \times 10^6) = 1\,\mu$s.
2. $N = (1\,\text{ms})/(1\,\mu\text{s}) = 1000$ ticks.
3. $P = 65536 - 1000 = 64536 =$ `FC18H`.
4. Load TH0=FCH and TL0=18H. The ideal counter reaches overflow after 1000 increments.

Worked answer: `TH0=FCH`, `TL0=18H` at 12 MHz. This is an ideal timer interval; instruction and ISR overhead affect repeated real-world pulses.

STEP-BY-STEP EXAMPLE: Mode 1, nominal 5 ms at 11.0592 MHz

Clock tick frequency $11{,}059{,}200/12 = 921{,}600$ Hz. For 5 ms: $N = 0.005 \times 921{,}600 = 4608 =$ `1200H`. Preload $65536 - 4608 = 60928 =$ `EE00H`. Hence TH0=EEH, TL0=00H, with exact ideal count-based delay at this clock.

Exam pattern: Write the oscillator assumption, divide by 12, compute ticks, subtract from 65536 (mode 1) or 256 (mode 2), convert to hexadecimal, load TH/TL, and explain where timing overhead enters.

6.5 Program C: poll a 1 ms Timer 0 overflow

This is original example code for a classic 12-clock 8051 at 12 MHz. A surrounding caller performs setup and uses a valid stack. The delay from setting

TR0 to overflow is nominal; execution overhead is additional.

```
    ANL TMOD,#0F0H   ; keep Timer 1, reset T0 field
    ORL TMOD,#01H    ; Timer 0 mode 1, internal clock
    CLR TR0
    CLR TF0
    MOV TH0,#0FCH
    MOV TL0,#018H
    SETB TR0         ; start T0
WAIT0: JNB TF0,WAIT0 ; poll overflow flag
    CLR TR0
    CLR TF0          ; clear flag for next call
```

6.6 Counter application: count parts on a conveyor

Set Timer 0 C/T=1 and mode 1 (lower TMOD bits $0101_2 = 5$). On classic hardware, edges arriving at T0 (P3.4) are sampled rather than arbitrarily fast; reliable edge counting requires electrical conditioning, valid pulse widths and appropriate maximum event rate. External counting should not be confused with decoding a noisy, unconditioned sensor wire.

7 8051 interrupts: when hardware requests attention

In simple words
Polling asks repeatedly: “Did an event happen?” An interrupt lets hardware signal a service request. The CPU temporarily suspends ordinary code, executes an interrupt-service routine (ISR), and returns to the interrupted instruction stream.

7.1 Standard 8051 interrupt vector table

Source	Vector	Enable and typical request
RESET entry	0000H	CPU starts here after reset
External INT0	0003H	EA=1, EX0=1; P3.2 trigger
Timer 0	000BH	EA=1, ET0=1; TF0 overflow
External INT1	0013H	EA=1, EX1=1; P3.3 trigger
Timer 1	001BH	EA=1, ET1=1; TF1 overflow
Serial port	0023H	EA=1, ES=1; TI and/or RI

Figure 8. Classic MCS-51 vector addresses. Reset is a vector but is not an interrupt.

The *original 8051* has five interrupt sources (not counting reset); later 8052-derived and vendor variants may have extra sources and vectors. Each vector is close to the next: put a short jump at the vector if the ISR code is larger. Do not copy a vendor-specific vector table without checking its family.

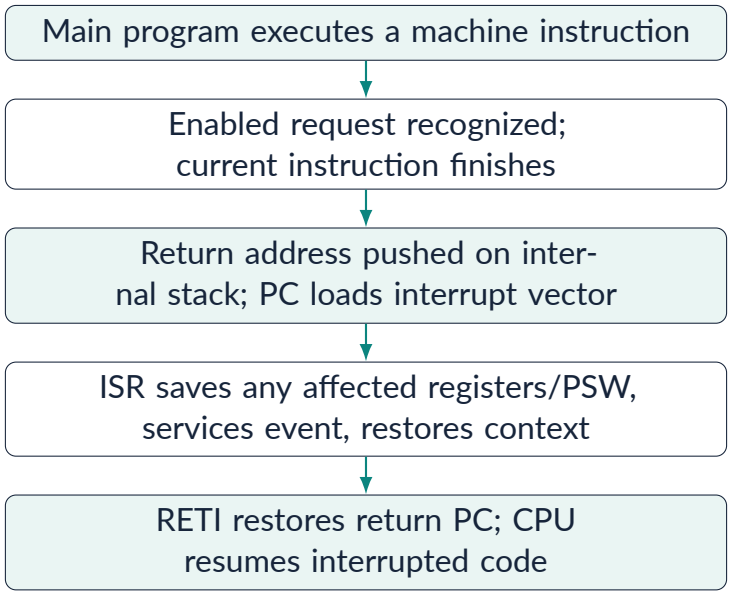


Figure 9. Interrupt service sequence. Hardware saves the return PC; software preserves any other working registers it changes.

7.2 Enabling, prioritizing, and acknowledging an interrupt

The IE register at A8H contains the global EA bit and source enables ES, ET1, EX1, ET0 and EX0. IP at B8H assigns high or low priority per source on the

original core. The classic 8051 supports two programmable priority levels; higher priority can pre-empt a lower-priority ISR, but an equal/lower one does not pre-empt the current ISR. Simultaneous requests within a priority level have a fixed internal polling order, not arbitrary order.

STEP-BY-STEP EXAMPLE: Why does IE=82H enable only Timer 0?

`82H = 1000 0010`$_2$: bit 7 EA=1 (global), bit 1 ET0=1. All other IE bits are 0. Timer 0 overflow can interrupt when TF0 becomes set and the core accepts the request. Setting ET0 alone with EA=0 cannot service an interrupt.

ISR programming rule: The interrupt mechanism saves the return address, not all CPU registers. If the ISR modifies ACC, B, DPTR or PSW, software must protect the used context. End interrupt handlers with RETI, not ordinary RET. On the classic 8051, timer-overflow flags TF0/TF1 are cleared by hardware when their ISR is accepted; external edge interrupt flags are likewise cleared on vectoring, whereas serial TI/RI remain set until software clears them. Other devices may differ. In particular, serial RI and TI must be cleared by software; do not assume all flags auto-clear.

7.3 Program D: interrupt-driven overflow counter

The following skeleton illustrates a main loop and a Timer 0 ISR that increments a software counter at internal RAM 30H. The timer must be initialized and started before enabling interrupts. It is intentionally simple; production code needs deterministic reloading and timing tests.

```
ORG 0000H
    LJMP MAIN
ORG 000BH          ; Timer 0 vector
    LJMP T0_ISR
ORG 0030H          ; code region, not data RAM 30H
MAIN:
    MOV 30H,#00H     ; software event count
    ANL TMOD,#0F0H  ; preserve Timer 1 fields
    ORL TMOD,#01H   ; configure Timer 0 mode 1
    MOV TH0,#0FCH
    MOV TL0,#018H
    CLR TF0
    MOV IE,#082H     ; EA=1, ET0=1
    SETB TR0
HERE: SJMP HERE
T0_ISR:
    PUSH ACC         ; save A, used below only for demo
    INC 30H          ; count overflows
    MOV TH0,#0FCH    ; manual reload for mode 1
    MOV TL0,#018H
    POP ACC
    RETI
END
```

Interpret this correctly: The vector at 000BH jumps to ISR code at 0030H *program memory*. Instruction `INC 30H` changes byte 30H in the separate *internal data memory*. Numeric address 0030H in code space and 30H in data RAM are not the same storage. Reload overhead means the resulting software count interval is *longer* than an exact 1 ms periodic real-time tick; compensate or use a carefully chosen timer scheme.

7.4 External interrupt practical example

A position-limit switch may cause INT0 to request immediate attention. Firmware should still confirm input conditioning and switch debounce, imple-

ment safe motor-driver shutdown and consider whether the desired safety function must be implemented independently in hardware. An ordinary microcontroller ISR is *not* a substitute for an industrial safety-rated stop mechanism.

8 Serial communication: sending useful bytes between devices

In simple words
Serial communication sends bits over time instead of sending a whole byte on eight separate wires. The classic 8051 includes a serial port with SBUF data registers and SCON control bits. In UART mode 1, it sends a start bit, eight data bits, and a stop bit for each character.

8.1 UART versus synchronous operation and the four modes

Mode	Format	Clocking and use
0	8-bit synchronous shift register	Fixed shift-clock derived from oscillator, clock signal on TXD.
1	8-bit UART frame	One start, eight data, one stop; baud adjustable via timer generation.
2	9-bit UART frame	One start, nine data, one stop; fixed oscillator-derived baud options.
3	9-bit UART frame	One start, nine data, one stop; adjustable baud as in mode 1.

The 8051's TXD and RXD are logic-level serial port pins (P3.1 and P3.0). USB and traditional RS-232 use different physical signaling and require appropriate transceivers/bridges. **Never wire the microcontroller directly to an RS-232 negative-voltage output.**

8.2 Draw the 8N1 frame: duration matters

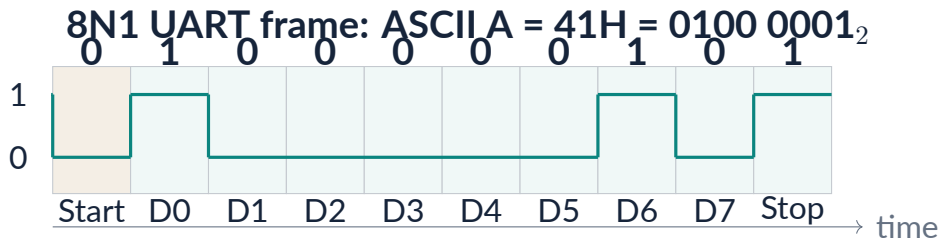


Figure 10. Original, labelled TTL-level UART waveform for ASCII A. Idle and stop are high; start is low; data bits transmit D0 first.

For 8N1 framing, one eight-bit character occupies ten bits on the wire: start + eight data + stop. At 9600 bits/s a frame takes $10/9600 = 1.041667$ ms; theoretical upper throughput is 960 characters/s if transmitted back to back. Throughput depends on framing, not just the nominal baud label.

8.3 SCON and SBUF: who controls what?

SCON (98H) bit fields from most to least significant are SM0, SM1, SM2, REN, TB8, RB8, TI and RI. In serial mode 1, SM0=0, SM1=1. REN=1 enables receiving. TI indicates a transmission-complete event, and RI indicates a received character. Both TI and RI are *software-clear* flags. SBUF (99H) is used to initiate transmission with a write and obtain received data with a read; underlying transmit/receive registers are logically distinct. A typical configuration `MOV SCON,#50H` selects mode 1 with receiving enabled.

8.4 Baud rate from Timer 1 mode 2

On the classic 12-clock 8051, mode-1 UART baud generated by Timer 1 in 8-bit auto-reload mode obeys

$$\text{baud} = \frac{2^{\text{SMOD}}}{32} \frac{f_{\text{osc}}/12}{256 - \text{TH1}},$$

where TH1 is interpreted as an *unsigned 8-bit reload number*, SMOD is the serial baud doubler bit in PCON, and Timer 1 is configured as an internal-clock timer in mode 2. It is not the formula for arbitrary hardware variants, UART modes or external T1 clock inputs.

STEP-BY-STEP EXAMPLE: 9600 baud, 11.0592 MHz, SMOD=0

1. Base timer rate: $f_{osc}/12 = 921{,}600$ ticks/s.
2. For SMOD=0, required Timer 1 overflow rate is $32 \times 9600 = 307{,}200$ overflows/s.
3. Auto-reload interval $N = 921{,}600/307{,}200 = 3$ ticks.
4. TH1 $= 256 - 3 = 253 =$ FDH. Timer 1 mode 2 corresponds to upper TMOD nibble 0010_2, so use TMOD=20H if Timer 0 need not be preserved.

Worked answer: TH1=FDH with 11.0592 MHz and SMOD=0 gives 9600 baud under the stated classic-core assumptions. The serial control byte is SCON=50H for 8-bit mode 1 and receiver enabled.

8.5 Program E: send ASCII "A" at 9600 baud

This illustrates a polled transmission on a correctly wired TTL-level UART. The oscillator is 11.0592 MHz and PCON.SMOD=0 as assumed above. An external terminal requires matching baud/framing and a suitable USB-to-UART or RS-232 interface.

```
    ANL PCON,#07FH    ; clear SMOD, preserve other bits
    CLR TR1           ; stop Timer 1 for setup
    ANL TMOD,#00FH    ; preserve Timer 0 setup
    ORL TMOD,#020H    ; configure Timer 1 mode 2
    MOV TH1,#0FDH     ; 9600 baud at 11.0592MHz
    MOV TL1,#0FDH
    MOV SCON,#050H    ; mode 1, receiver enabled
    CLR TI
    SETB TR1
    MOV SBUF,#041H    ; ASCII 'A'
TXWAIT:
    JNB TI,TXWAIT     ; wait for transfer flag
    CLR TI            ; acknowledge in software
DONE: SJMP DONE       ; teaching terminal loop
```

Review the API/hardware: This example is written in common 8051 assembly syntax. It does not claim native emulator execution. Some assemblers interpret numeric literal syntax differently, and controller derivatives may change timing and data buffering. Do not transmit before the physical level converter and common-ground arrangements are understood.

8.6 What changes when the serial port uses interrupts?

Set EA and ES in IE and place an ISR at vector 0023H. Both transmit-complete TI and receive-complete RI can request the *same* serial interrupt vector. The ISR must test which flag(s) caused it, read SBUF when RI is set, then clear RI (and clear TI when set) in software, and preserve any CPU registers it modifies. Never assume a single interrupt entry means “receive only.”

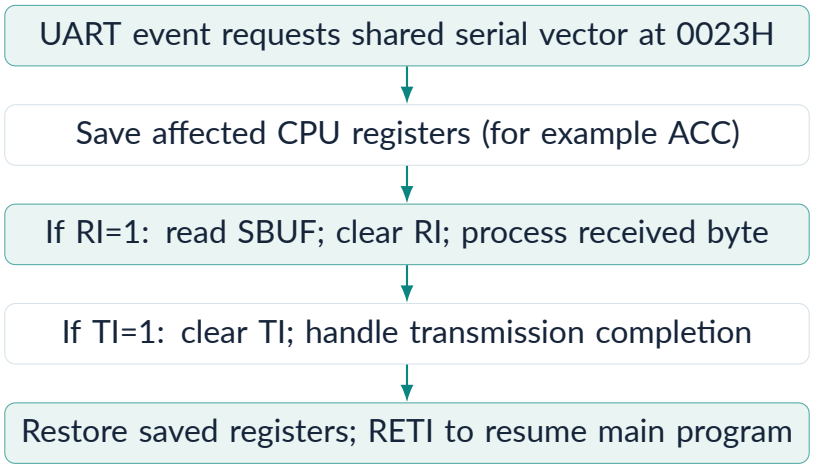


Figure 11. Serial ISR handling both possible sources on the same interrupt vector. Real firmware also needs buffering, error handling, and suitable UART initialization.

Remember: The serial interrupt is shared. If both RI and TI are set, a correct ISR can acknowledge *both* in a single entry; do not write an ISR that assumes only reception or only transmission.

8.7 Program F: a serial interrupt reads a received byte

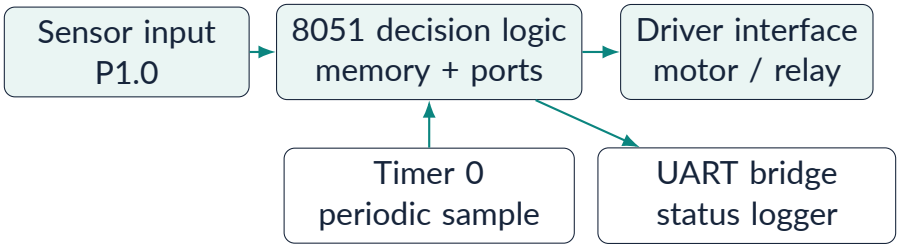
The UART uses a single serial interrupt vector for both RI (receive) and TI (transmit). This minimal example demonstrates the source test, clearing each flag, preserving ACC, and storing a received byte in internal RAM at 40H. The user must first configure Timer 1 for a matching UART baud rate as shown earlier. In the ISR, read the pending byte before clearing RI; otherwise, early acknowledgement could allow another receive event while the current byte is still being handled. Buffer overrun and back-to-back frames require a proper ring buffer in real firmware.

```
ORG 0000H
    LJMP START
ORG 0023H            ; shared serial interrupt
    LJMP SERIAL_ISR
ORG 0030H
START:
    MOV SCON,#050H   ; mode 1, receiver enabled
    MOV IE,#090H     ; global EA + serial ES
IDLE: SJMP IDLE
SERIAL_ISR:
    PUSH ACC
    JNB RI,CHECK_TX
    MOV A,SBUF       ; read received byte first
    CLR RI           ; acknowledge reception flag
    MOV 40H,A        ; store byte in internal RAM
CHECK_TX:
    JNB TI,EXIT_ISR
    CLR TI           ; acknowledge transmit flag
EXIT_ISR:
    POP ACC
    RETI
END
```

Hardware setup required: This intentionally omits complete Timer 1 baud-generator initialization and electrical interface circuitry; include both before real use. As with the other samples, the assembly is not yet verified in a native emulator or on a physical 8051.

9 Integrated robotic-control example and revision

In simple words
A useful embedded controller connects sensing, timing, decisions and communication. Think of a small belt conveyor with a photo-sensor, output driver, and serial status display. Each 8051 peripheral solves a different part of the same problem.



Concept-only; add debounce, electrical protection and safety controls.

9.1 A complete workflow with engineering checks

1. **Choose memory layout.** Reserve lower RAM bytes for registers and stack, a few bytes in 30H–7FH for samples and counts, and never treat SFR space as ordinary user data.
2. **Initialize ports.** For a classic quasi-bidirectional 8051 port, account for the latch/input behavior; use a properly designed sensor interface, pull-ups and driver stage.
3. **Configure the timer.** Compute reload from the actual oscillator and core cycle rate, set TMOD, THx/TLx, enable if needed and verify the real sample period.
4. **Handle an event.** Poll a flag or use an interrupt. Keep ISR code short, clear flags according to source, and preserve register context.
5. **Send diagnostics.** Configure SCON and Timer 1 baud generation; use SBUF and TI/RI. Check voltage levels and handshake/overflow behavior.
6. **Test safety.** A software timer or interrupt is not safety-certified equipment. Implement emergency stop, electrical isolation and independent protections as required.

9.2 Failure modes to explain in an exam

Observed failure	Likely conceptual cause
Timer delay too fast or too slow	Assumed wrong oscillator or wrong machine-cycle rate
Counter misses fast events	External pulses too short/noisy; sampling-rate limit exceeded
Timer 1 UART stopped working	TMOD accidentally overwritten by Timer 0 configuration
Serial never transmits again	TI flag not cleared or baud setup invalid
ISR runs once but never again	Uncleared source condition, bad RETI/stack or interrupt enable state
Program reads wrong byte	Confused immediate, direct RAM, code-memory or SFR addressing

9.3 Formula-and-vector quick-reference sheet

Classic 8051 12-clock timing: $f_{\text{tick}} = f_{\text{osc}}/12$; $N = tf_{\text{osc}}/12$; mode 1 preload $65536 - N$, mode 2 preload $256 - N$ (fit in 8 bits).
Core vectors: Reset 0000H; EX0 0003H; T0 000BH; EX1 0013H; T1 001BH; Serial 0023H.
UART mode 1/3 with Timer 1 mode 2: baud $= 2^{\text{SMOD}}(f_{\text{osc}}/12)/[32(256 - \text{TH1})]$. At 11.0592 MHz with SMOD=0, TH1=FDH gives 9600 baud.
8051 memory: register banks 00H–1FH; bit RAM 20H–2FH; data RAM 30H–7FH; SFR direct addresses 80H–FFH (not all assigned).
Safe practice: state the exact chip, quartz frequency, timer mode and interrupt assumptions before quoting timing.

9.4 Numerical practice with verified answers

1. A classic 12-clock 8051 uses a 12 MHz oscillator. Find a mode-1 preload for 2 ms. **Answer:** $N = 2000$ ticks, preload $65536 - 2000 = 63536 =$ F830H; TH0=F8H, TL0=30H.

2. With 11.0592 MHz and timer mode 1, how many ticks represent 10 ms?

Answer: $921600 \times 0.010 = 9216 =$ `2400H`, preload `DC00H`.

3. Timer 0 mode 2 needs exactly 200 ticks. Which reload? **Answer:** $256 - 200 = 56 =$ 38H; TH0=38H, TL0 initialized 38H for first interval.
4. Which IE bits does 90H set? **Answer:** EA=1 and ES=1, enabling serial interrupt requests globally; no other source enable is selected.
5. With 8N1 framing, how long does one character take at 4800 baud? **Answer:** $10/4800$ s ≈ 2.0833 ms.
6. With 11.0592 MHz, SMOD=0, TH1=FAH, what baud in serial mode 1? **Answer:** $921600/[32(256 - 250)] = 4800$ baud.

9.5 Exam revision: explain, draw, derive, trace

Explain microcontroller versus processor and CISC versus RISC; draw the functional 8051 blocks and memory map; demonstrate each addressing mode; derive the mode-1/mode-2 timer-preload formula, identify each TCON/TMOD field, draw the interrupt vector/ISR flow, show UART 8N1 framing and calculate TH1 for 9600 baud. Finally trace at least one program for each of timer delay, ISR event counter and serial transmission, including flag and stack behavior.

9.6 Putting the classic 8051 in a real robotic signal chain

In simple words
An 8051 pin reads logic, not an arbitrary sensor voltage. Its output latch also does not supply the current required by a motor or inductive relay. A protected input stage and motor-driver stage are essential components, not optional software features.

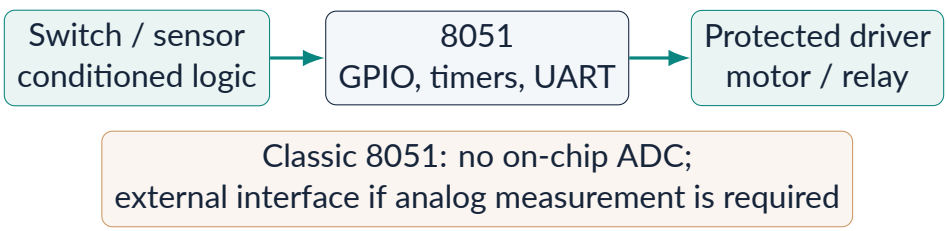


Figure 12. Original device-capability boundary diagram: a logic-level sensor or interface feeds the 8051, and a separate driver delivers actuator power.

Check the assumptions before copying code. Classic 8051 UART baud calculations often assume a 12-clock machine cycle and a specified crystal. Many derivatives have alternative clock dividers, additional peripherals or different interrupt priorities. Check oscillator, serial flags, port electrical properties and SFR locations in the exact device datasheet.

10 Practice questions and answer checkpoints

1. Compare 8086 and 8051 by memory integration, width and I/O responsibilities.
2. Explain why calling the 8051 a CISC-style architecture does not automatically determine performance.
3. Draw a labelled 8051 functional diagram including RAM, ROM, timers, UART, interrupt controller and four ports.
4. Explain ACC, B, PSW, PC, DPTR and SP using one instruction each.
5. Draw the four register banks and find R5 in bank 3. **Answer:** bank 3 starts 18H; R5 is 1DH.
6. Prove there are 128 bit-addressable RAM bits between 20H and 2FH. **Answer:** 16 bytes $\times$ 8 bits each.
7. Differentiate `MOV A,#30H`, `MOV A,30H`, `MOV A,@R0`, `MOVX A,@DPTR` and `MOVC A,@A+DPTR`.
8. Trace F8H+15H in the previous program, including the carry byte.
9. Explain why the original 8051 and modern enhanced 8051 derivatives can have different on-chip RAM capacities.

Candidate scope versus publication approval. This review PDF now covers all seven syllabus headings, but actual 8051 assembler/emulator execution, detailed technical sign-off, figure licence review and final page inspection are still required before marking it production-ready. Do not replace the live Unit III notes prematurely.

Corrections and verified hardware assumptions

Do not confuse these often-mixed cases:

1. **Original Intel 8051 vs 8052/enhanced cores:** 128 bytes RAM, two timers and five interrupt sources describe the basic 8051; an 8052 or modern derivative may add RAM, Timer 2 or other interrupt sources.
2. **Interrupt flags:** Timer TF0/TF1 generally clear on ISR vectoring on the classic device; `TI` and `RI` require explicit software clearing. Still consult the target hardware manual.
3. **Timer time vs exact wall-clock time:** an initial count of 1000 at 1 μs per tick represents nominally 1 ms; software reloads and interrupt-entry latency add timing error.
4. **Bit address vs byte address:** bit address `00H` is bit 0 of internal byte 20H, not byte address 00H itself. Port bit names such as `P1.0` are SFR bits, not internal RAM bit addresses.
5. **UART electrical level:** the graph shows TTL logic, not the opposite-polarity and higher-voltage RS-232 electrical standard. Use a proper bridge.

Figure provenance and reuse

All diagrams in this edition are original TikZ vectors derived from hardware facts; no third-party photo, book figure, or downloaded raster is embedded. A CC BY-SA 4.0 photo of an Intel C8751H was identified on Wikimedia Commons (author Mister rf), but its file bytes could not be downloaded and were therefore not included. The conceptual drawings are designed for legibility rather than acting as pin-level engineering schematics.

Source notes and limitations

Official scope: Savitribai Phule Pune University, TE Robotics & Automation Engineering (2024 Pattern), PCC302-ROA, Unit III, provided syllabus PDF pp.

16–17 (physical PDF pages 16–17).

Technical detail: Atmel, *8051 Microcontrollers Hardware Manual*, Rev. 4316E–8051–01/07, especially instruction-set section and hardware organization. This is a historical family-compatible reference; device variants may differ. Intel, *MCS-51 Microcontroller Family User's Manual*, accessible from MIT `https://web.mit.edu/6.115/www/document/8051.pdf`.

Access limitation: The two college Drive PDFs labelled MM Unit 3 notes and Unit III Final were discoverable by metadata but returned HTTP 403 on content read; neither has been treated as inspected evidence. No external photographs were embedded; diagrams are original and conceptual, not pin-accurate circuit schematics.

Final hardware-assumption checklist for exam solutions

Before concluding any 8051 calculation, write down *which 8051 variant*, the oscillator frequency, whether the timer sees a clock divided by 12, and the mode width (8-bit auto-reload, 16-bit or split timer). Explicit assumptions prevent the most common mistakes.

1. **Addressing:** direct and register-indirect addresses select different hardware paths; internal register banks change with PSW bits.
2. **Ports:** verify the actual 8051 port circuit; quasi-bidirectional behaviour and pull-up rules matter. External analog signals require conditioning and conversion.
3. **Timers:** subtract preload from the correct modulus. State if the result neglects ISR and software-reload overhead.
4. **Serial:** clear the correct RI/TI flag and read SBUF at the proper time; link UART logic to a correctly level-shifted electrical interface.
5. **Robot output:** the port sets a low-power command, while a rated driver provides motor or solenoid current. Protective shutdown must remain effective during a controller fault.