8051 Overview and Instruction Set

Curtis A. Nelson

Engr 355

Microprocessors vs. Microcontrollers

- **Microprocessors are single-chip CPUs used in microcomputers**
- **Microcontrollers and microprocessors are different in three main aspects: hardware architecture, applications, and instruction set features**
- **Hardware architecture: A microprocessor is a single chip CPU while a microcontroller is a single IC contains a CPU and much of remaining circuitry of a complete computer (e.g., RAM, ROM, serial interface, parallel interface, timer, interrupt handling circuit)**
- **Applications: Microprocessors are commonly used as a CPU in computers while microcontrollers are found in small, minimum component designs performing control oriented activities**

Microprocessors vs. Microcontrollers

- Instruction set:
  - Microprocessor instruction sets are processing intensive
    - Their instructions operate on nibbles, bytes, words, or even double words.
    - Addressing modes provide access to large arrays of data using pointers and offsets.
  - Microcontroller instruction sets cater to control of inputs and outputs
    - They have instructions to set and clear individual bits and perform bit operations.
    - They have instructions for input/output operations, event timing, enabling and setting priority levels for interrupts caused by external stimuli.
- Processing power of a microcontroller is much less than a microprocessor.

8051

- Today over fifty companies produce variations of the 8051.
- Several of these companies have over fifty versions of the 8051.
- 8051 cores are available for implementations in FPGA’s or ASIC’s.
- Over 100 million 8051’s are sold each year.
- The 8051 has been extremely successful, and has directly influenced many of the more recent microcontroller architectures.

MCS-51

- 8051 belongs to MCS-51 family of microcontrollers
- MCS-51 was developed by Intel but other manufacturers (e.g., Siemens, Philips) are second sources of this family.
- Summary of features of the standard 8051
  - 4K bytes internal ROM (program)
  - 128 bytes internal RAM (data)
  - Four 8-bit I/O ports
  - Two 16-bit timers
  - Serial interface
  - 64K external code memory space
  - 64K external data memory space
  - 210 bit-addressable locations

Memory

- 8051 implements a separate memory space for programs (code) and data.
- Both code and data may be internal, however, both expand using external components to a maximum of 64K code memory and 64K data memory.
- Internal memory consists of on-chip ROM and on-chip data RAM.
- On-chip RAM contains a rich arrangement of general purpose storage, bit addressable storage, register banks, and special function registers.
- In the 8051, the registers and input/output ports are memory mapped and accessible like any other memory location.
- In the 8051, the stack resides within the internal RAM, rather than in external RAM.

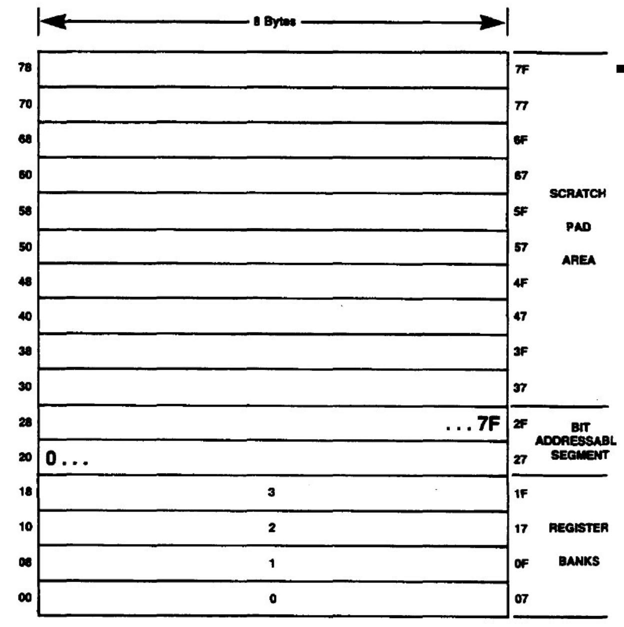


Figure 4. 128 Bytes of RAM Direct and Indirect Addressable

SFR MEMORY MAP

8 Bytes

F8									FF
F0	B								F7
E8									EF
E0	ACC								E7
D8									DF
D0	PSW								D7
C8	T2CON		RCAP2L	RCAP2H	TL2	TH2			CF
C0									C7
B8	IP								BF
B0	P3								B7
A8	IE								AF
A0	P2								A7
98	SCON	SBUF							9F
90	P1								97
88	TCON	TMOD	TL0	TL1	TH0	TH1			8F
80	P0	SP	DPL	DPH				PCON	87

↑
Bit
Addressable

Figure 5

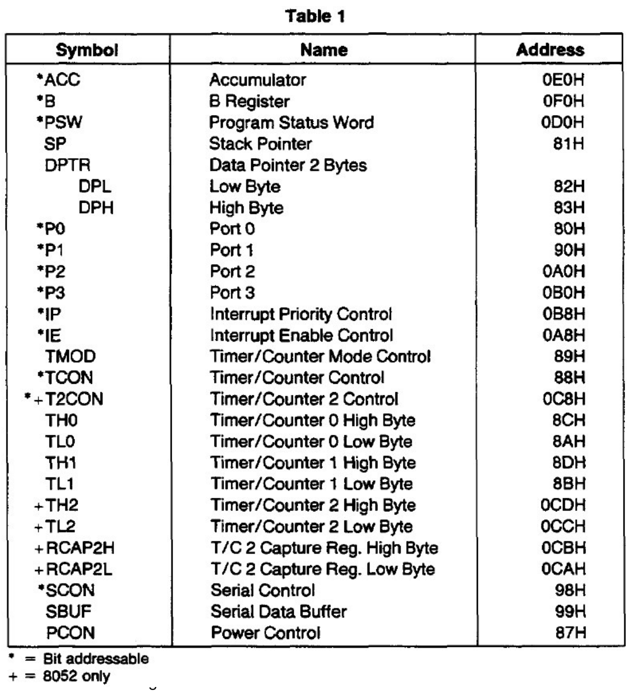


Table 1

Symbol	Name	Address
*ACC	Accumulator	0E0H
*B	B Register	0F0H
*PSW	Program Status Word	0D0H
SP	Stack Pointer	81H
DPTR	Data Pointer 2 Bytes	
DPL	Low Byte	82H
DPH	High Byte	83H
*P0	Port 0	80H
*P1	Port 1	90H
*P2	Port 2	0A0H
*P3	Port 3	0B0H
*IP	Interrupt Priority Control	0B8H
*IE	Interrupt Enable Control	0A8H
TMOD	Timer/Counter Mode Control	89H
*TCON	Timer/Counter Control	88H
*+T2CON	Timer/Counter 2 Control	0C8H
TH0	Timer/Counter 0 High Byte	8CH
TL0	Timer/Counter 0 Low Byte	8AH
TH1	Timer/Counter 1 High Byte	8DH
TL1	Timer/Counter 1 Low Byte	8BH
+TH2	Timer/Counter 2 High Byte	0CDH
+TL2	Timer/Counter 2 Low Byte	0CCH
+RCAP2H	T/C 2 Capture Reg. High Byte	0CBH
+RCAP2L	T/C 2 Capture Reg. Low Byte	0CAH
*SCON	Serial Control	98H
SBUF	Serial Data Buffer	99H
PCON	Power Control	87H

* = Bit addressable
+ = 8052 only

Bit Addressable RAM

- Individual accessing of bits is a powerful feature of microcontrollers
- Bits can be set, cleared, ANDed, ORed etc, with a single instruction
- 8051 ports are bit-addressable, simplifying the interface to single bit inputs and outputs
- The 8051 contains 210 bit-addressable locations
- 128 of these locations are at addresses 20_H to $2F_H$ and the rest are in the special function registers

Register Banks

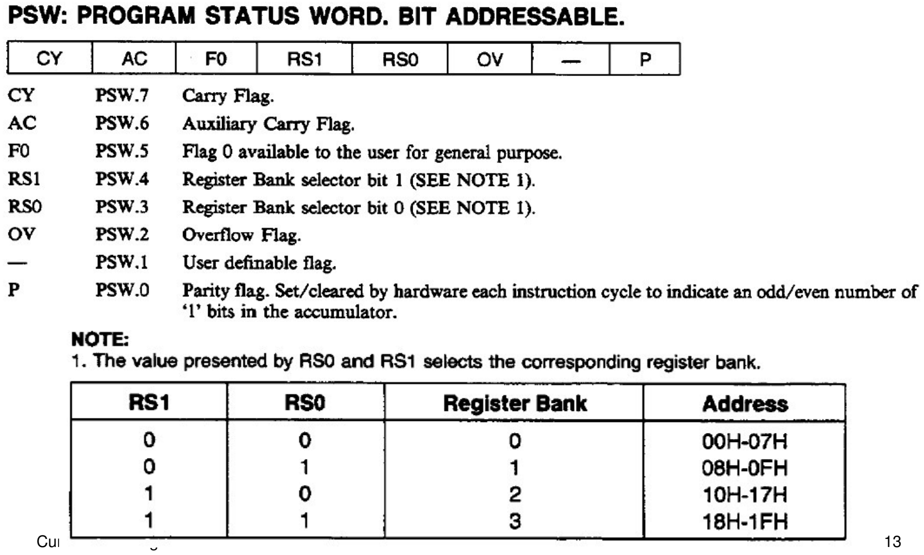
- The bottom 32 locations of internal memory contain the register banks
- 8051 supports 8 registers R0 to R7 and after a system reset (default) the registers are at address 00_H to 07_H
- MOV A, R5: reads the content of address 05_H into the accumulator
- MOV A,05_H will do the same thing
- The active register bank may be altered by changing the register bank select bits in the Program Status Word (PSW)
- Idea of register banks permits fast and effective context switching

Special Function Registers

- 8051 has 21 special function registers (SFRs) at the top of internal RAM from address 80H to FF_H.
- Most of the addresses from 80_H to FF_H are not defined, except for 21 of them.
- Some SFR’s are both bit-addressable and byte addressable, depending on the instruction accessing the register

Program Status Word

- Program status word (PSW) at address DO_H contains status bits as summarized in the following table



PSW: PROGRAM STATUS WORD. BIT ADDRESSABLE.

CY	AC	F0	RS1	RS0	OV	—	P

CY	PSW.7	Carry Flag.
AC	PSW.6	Auxiliary Carry Flag.
F0	PSW.5	Flag 0 available to the user for general purpose.
RS1	PSW.4	Register Bank selector bit 1 (SEE NOTE 1).
RS0	PSW.3	Register Bank selector bit 0 (SEE NOTE 1).
OV	PSW.2	Overflow Flag.
—	PSW.1	User definable flag.
P	PSW.0	Parity flag. Set/cleared by hardware each instruction cycle to indicate an odd/even number of '1' bits in the accumulator.

NOTE:
1. The value presented by RS0 and RS1 selects the corresponding register bank.

RS1	RS0	Register Bank	Address
0	0	0	00H-07H
0	1	1	08H-0FH
1	0	2	10H-17H
1	1	3	18H-1FH

Program Status Word

- Carry flag is also the "Boolean accumulator": 1 bit register for Boolean instructions
- Example: ANLC,25_H
- AND's bit 25_H with the carry flag and places the result back in the carry flag
- Auxiliary carry flag (AC): is set if a carry was generated out of bit 3 into bit 4 or if the result in the lower nibble is in the range $0A_H$ to $0F_H$
- AC is useful in arithmetic operations on binary coded decimal (BCD) values.

Program Status Word

- Flag 0 (F0): a general purpose flag bit available for user
- Register Bank Select Bits (RS0 and RS1): determine the active register bank
- Example: the following instructions enable register bank 3 and move the content of R7 (address $1F_H$) to the accumulator

```
SETB RS1
SETB RS0
MOV A,R7
```

Program Status Word

- Overflow flag (OV): is set after an addition or subtraction if there was an arithmetic overflow
- When signed numbers are added or subtracted this bit determines if the result is in the proper range
- Results greater than 127 or less than –128 will set OV bit
- When unsigned numbers are added OV can be ignored
- Example: What is the OV and the content of accumulator after the following instruction sequence:

  MOV R7, $\#FF_H$
  MOV A, $\#0F_H$
  ADD A,R7
  – Answer: OV=0, ACC=$0E_H$

Program Status Word

- Parity bit (p): is automatically set or cleared in each machine cycle to establish even parity in the accumulator
- Number of 1-bits in the accumulator plus P is always even
- P is used in serial port routines
- What is the state of P after execution of the following instruction?

  MOV A,#55H
  - Answer: P=0

B Register

- B register or accumulator B at address $F0_H$ is used along with the accumulator for multiply and divide operations
- MUL AB: multiplies 8 bit unsigned values in A and B and leaves the 16 bit result in A (low byte) and B (high byte)
- DIV AB: divided A by B, leaving the integer result in A and remainder in B
- B register is bit-addressable

Stack Pointer

- Stack pointer (SP) is an 8-bit register at address 81_H
- It contains the address of the data item currently on top of the stack.
- Stack operations include pushing data on the stack and popping data off the stack
- Pushing increments SP before writing the data
- Popping from the stack reads the data and decrements the SP
- 8051 stack is kept in the internal RAM
- Depending on the initial value of the SP, stack can have different sizes
- Example: MOV SP,#$5F_H$
- On 8051 this would limit the stack to 32 bytes since the uppermost address of on chip RAM is $7F_H$.

Stack and Data Pointers

- The default value of SP (after system reset) is 07_H.
- This result in the first stack write operation to store data in location 08_H which means that register bank 1 (and possible 2 and 3) are not available
- User may initialize the SP to avoid this
- Data pointer (DPTR): is used to access external data or code
- DPTR is a 16 bit register at addresses 82_H (low byte) and 83_H (high byte)
- Example: the following instructions write 55_H into external RAM location 1000_H:

```
MOV A,#55H
MOV DPTR,#1000H
MOVX @DPTR,A
```

Instruction Set

- 8051 instructions have 8-bit opcode
- There are 256 possible instructions of which 255 are implemented
- Some instructions have one or two additional bytes for data or address
- There are 139 1-byte instructions, 92 2-byte instructions, and 24 3-byte instruction
- Where does the data for an instruction come from?
  - Addressing modes

Addressing Modes

- There are eight addressing modes available in the 8051:
  - Register
  - Direct
  - Indirect
  - Immediate
  - Relative
  - Absolute
  - Long
  - Indexed

Register Addressing

- 8051 has access to eight working registers (R0 to R7)
- Instructions using register addressing are encoded using the three least significant bits of the instruction opcode to specify a register
- Example: ADD A,R7
- The opcode is 00101111. 00101 indicates the instruction and the three lower bits, 111, specify the register
- Some instructions are specific to a certain register, such as the accumulator, data pointer etc.
- Example: INC DPTR
  - A 1-byte instruction adding 1 to the data pointer
- Example: MUL AB
  - A 1-byte instruction multiplying unsigned values in accumulator and register B

Direct Addressing

- Direct addressing can access any on-chip memory location
- Example: ADD A,55_H
- Example: MOV P1, A
  - Transfers the content of accumulator to Port 1 (address 90_H)

Indirect Addressing

- How is a variable identified if its address is determined or modified while a program is running?
- 8051 solution is indirect addressing: R0 or R1 may operate as pointer registers (their content indicates an address in internal RAM where data are written or read)
- In 8051 assembly language, indirect addressing is represented by an @ before R0 or R1.
- Example: MOV A, @R0
  - Moves a byte of data from internal RAM at location whose address is in R0 to the accumulator
- Example:

```
            MOV R0, #60H
Loop:       MOV @R0,#0
            INC R0
            CJNE R0,#80H,Loop
```

Immediate Addressing

- When the source operand is a constant rather than a variable, the constant can be incorporated into the instruction as a byte of immediate address
- In assembly language, immediate operands are preceded by #
- Operand my be a numeric constant, a symbolic variable or an arithmetic expression using constants, symbols and operators.
- Assembler computes the value and substitutes the immediate data into the instruction
- Example: MOV A,#12

Immediate Addressing

- With one exception, all instructions using immediate addressing use 8-bit data
- Exception: when initializing the data pointer, a 16-bit constant is required
- Example: MOV DPTR, #8000_H

Relative Addressing

- Relative addressing is used with certain jump instructions
- Relative address (offset) is an 8-bit signed value (-128 to 127) which is added to the program counter to form the address of next instruction
- Prior to addition, the program counter is incremented to the address following the jump (the new address is relative to the next instruction, not the address of the jump instruction)
- This detail is of no concern to the user since the jump destinations are usually specified as labels and the assembler determines the relative offset
- Advantage of relative addressing: position independent codes

Absolute Addressing

- Absolute addressing is only used with ACALL and AJMP
- The 11 least significant bits of the destination address comes from the opcode and the upper five bits are the current upper five bits in the program counter (PC).
- The destination is in the same 2K (2^{11}) of the source

Long Addressing

- Long addressing is used only with the LCALL and LJMP instructions
- These 3-bytes instructions include a full 16-bit destination address as bytes 2 and 3
- The full 64K code space is available
- The instruction is long and position dependent
- Example: LJMP, $8AF2_H$
- Jumps to memory location $8AF2_H$

Indexed Addressing

- Indexed addressing uses a base register (either the program counter or data pointer) and an offset (the accumulator) in forming the effective address for a JMP or MOVC instruction
- Example: MOVC A, @A+DPTR
  - This instruction moves a byte of data from code memory to the accumulator. The address in code memory is found by adding the accumulator to the data pointer

Instruction Types

- 8051 instructions are divided among five groups:
  - Arithmetic
  - Logical
  - Data transfer
  - Boolean variable
  - Program branching

Arithmetic

- Since different addressing modes are available, an arithmetic instruction may be written in different ways.
- Example:

  ADD A,$7F_H$

  ADD A,@R0

  ADD A,R7

  ADD A,#35_H
- All arithmetic instructions are executed in one machine cycle except INC DPTR (two cycles) and MUL AB and DIV AB (four cycles)

Arithmetic

- Example: accumulator contains 63_H, R3 contains 23_H, and the PSW contains 00_H. What is the content of the accumulator and the PSW after execution of ADD A, R3 instruction?
  - Answer: ACC=86_H, C=0, OV=1, P=1 PSW=00000101
- Example: write code that subtracts content of R6 from R7 and leave the result in R7

```
MOV A,R7
CLR C
SUBB A, R6
MOV R7,A
```

  - Clearing the flag is necessary because the only form of subtraction in 8051 is SUBB (subtract with borrow). The operation subtracts from the accumulator source byte and carry bit.

Arithmetic

- Any memory location can be incremented or decremented using direct addressing without going through the accumulator.
- Example: INC $7F_H$
  - Increments the value in memory location $7F_H$
- INC instruction can also work on 16-bit data pointer
- A decrement data pointer is not provided and requires a sequence of instructions:

```
      DEC DPL
      MOV R7,DPL
      CJNE R7, #FFH, SKIP
      DEC DPH
SKIP: (continue)
```

Arithmetic

- MUL AB: multiplies 8 bit unsigned values in A and B and leaves the 16 bit result in A (low byte) and B (high byte). If the product is greater than 255 (FF_H), overflow flag is set.
- Example: ACC=55_H, B register contains 22_H, and PSW=00_H. What are the contents of these registers after execution of the MUL AB instruction?
  - Answer: ACC=$4A_H$, B=$0B_H$, P bit in PSW is set to one. Since the result is greater than 255, overflow flag is set.
- DIV AB: divided A by B, leaving the integer result in A and remainder in B

Arithmetic

- For BCD arithmetic, ADD and ADDC must be followed by a DA A (decimal adjust) operation to ensure the result is in range for BCD.
  - Note: ADDC simultaneously adds accumulator, the variable and the carry flag.
- Note that DA A will not convert a binary number to BCD
- Example: If ACC contains BCD value of 59 then:

```
ADD A, #1
DA A
```

  - First adds 1 to A, leaving 5A and then adjust the result to correct BCD value 60.

Arithmetic

- Example: Two 4-digit BCD numbers are in internal memory at locations 40_H, 41_H and 42_H, 43_H. The most significant digits are in locations 40_H and 42_H. Add them and store the BCD result in locations 40_H and 41_H.

MOV A, 43_H
ADD A, 41_H
DA A
MOV 41_H, A
MOV A, 42_H
ADDC A, 40_H
DA A
MOV 40_H,A

  - An example of multi-precision arithmetic

Logical Instructions

- 8051 logical instructions perform Boolean operations on bytes of data on a bit-by-bit basis .
- Example: let's assume A=00110101$_B$. Instruction ANL A,#01010011$_B$ will leave 00010001 in accumulator
- Different modes for logical instructions:

  ANL A,55$_H$
  ANL A,@R0
  ANL A,R6
  ANL A,#33$_H$
- Logical operations can be performed on any byte in internal memory without going through the accumulator
- Example: XRL P1,#FF$_H$
- Eight bits in Port 1 are read, each bit exclusive ORed. The result is written back to Port 1.

Logical Instructions

- Rotate instructions (RL A, RR A) shift the accumulator one bit to the left or right. For a left rotation, MSB rolls into LSB position. For a right rotation, LSB rolls into MSB position.
- RLC A and RRC A perform similar operations on concatenation of carry flag and accumulator.
- Example: A=00$_H$ and C=1. After RRC A, what will be in A and C?
  - A=80$_H$ and C=0
- SWAP A: exchanges high and low nibbles within accumulator

Logical Instructions

- Illustrate two ways to rotate contents of accumulator three positions to the left. Discuss each method in terms of memory and execution speed

```
RL A
RL A
RL A

SWAP A
RR A
```

  – All instruction are 1-byte, 1-cycle. So first solution uses three bytes of memory and takes three CPU cycles and second solution uses only two bytes of memory and executes in two cycles

Logical Instructions

- Example: write an instruction sequence to reverse the bits in the accumulator.

```
        MOV R7,#8
LOOP:   RLC A
        XCH A,B
        RRC A
        XCH A,B
        DJNZ R7,LOOP
        XCH A,B
```

  – XCH A,B exchanges the content of accumulator with B register

Data Transfer Instructions - Internal

- MOV <destination>, <source>: allows data to be transferred between any two internal RAM or SFR locations
- Stack operations (pushing and popping data) are also internal data transfer instructions
- Pushing increments SP before writing the data
- Popping from the stack reads the data and decrements the SP
- 8051 stack is kept in the internal RAM
- Example: stack pointer contains 07_H and A contains 55_H and B contains $4A_H$. What internal RAM locations are altered and what are their new values after the following instructions?
  PUSH ACC
  PUSH $F0_H$
  Answer: address 08_H will have 55_H, address 09_H will have $4A_H$ and address 81_H (SP) will have 09_H.

Data Transfer Instructions - Internal

- Instruction XCH A, <source> causes the accumulator and the address byte to exchange data
- Instruction XCHD A, @Ri causes the low-order nibbles to be exchanged.
- Example: if A contains $F3_H$, R1 contains 40_H, and internal RAM address 40_H contains $5B_H$, instruction XCHD A, @R1 leaves A containing FB_H and internal RAM location 40_H containing 53_H.

Data Transfer Instructions - External

- Data transfer instructions that move data between internal and external memory use indirect addressing
- The address could be one byte (residing in R0 or R1) or two bytes (residing in DPTR)
- 16-bit addresses uses all Port 2 for high-byte and this port cannot be used for I/O
- 8-bit addresses allow access to a small external memory
- MOVX is used for external data transfer
- Example: Read the content of external RAM locations $10F4_H$ and $10F5_H$ and place values in R6 and R7, respectively.

  MOV DPTR,#$10F4_H$
  MOVX, A,@DPTR
  MOV R6,A
  INC DPTR
  MOVX A,@DPTR
  MOV R7,A

Look-Up Tables

- MOVC loads the accumulator with a byte from code (program) memory
- The address of the byte fetched is the sum of the original unsigned 8-bit accumulator contents and the content of a 16-bit register (either the data pointer or PC). In the latter case, the PC is incremented to the address of the following instruction before being added to the accumulator

```
MOVC A, @A+DPTR
MOVC A,@A+PC
```

- This instruction is useful in reading data from LUT's.
- DPTR or PC is initialized to the beginning of the LUT and the index number of the desired entry is loaded into the accumulator.

Look-Up Tables

- Example: write a subroutine called SQUARE to compute the square of an integer between 0 and 9. Write two versions of the subroutine (a) using LUT and (b) without using LUT
  - Using LUT

```
SQUARE:             INC A
                    MOVC A, @A+PC
                    RET
TABLE:              0,1,4,9,16,25,36,49,64,81
```

  - Not using LUT

```
SQUARE:     PUSH F0H
            MOV F0H,A
            MUL AB
            POP F0H
            RET
```

  - Calling the subroutine:

```
MOV A,#6
CALL SQUARE
```

  - First approach 13 bytes, 5 cycles. Second approach 8 bytes and 11 cycles

Boolean Instructions

- 8051 contains a complete Boolean processor for single-bit operations.
- All bit accesses use direct addressing
- Bits may be set or cleared in a single instruction
- Example: SETB P1.7 CLR P1.7
- Carry bit in PSW is used as a single-bit accumulator for Boolean operations.
- Bit instructions that refer to carry bit as C are assembled as carry-specific instructions
- Carry also has a mnemonic representation (CY) which can be used in connection with non-carry-specific instructions.
- Example:

```
CLR C
CLR CY
```

Both do the same. First one is 1 byte and the second one is 2-bytes

Boolean Instructions

- Example: Compute the logical AND of the input signals on bits 0 and 1 of Port 1 and output the result to bit 2 of Port 1.

```
LOOP:    MOV C, P1.0    (1 cycle)
         ANL C,P1.1     (2 cycle)
         MOV P1.2,C     (2 cycle)
         SJMP LOOP      (2 cycle)
```

  – Worst case delay is when one of the inputs changes right after the first instruction. The delay will be 11 CPU cycles (for a 12 MHZ clock, this is 11 us)

Branching Instructions

- There are three versions of JMP instruction: SJMP, LJMP and AJMP.
- SJMP instruction specifies destination address as a relative offset. This instruction is 2 bytes and jump distance is limited to -128 to 127.
- LJMP specifies the destination address as a 16-bit constant. The destination address can be anywhere in the 64K program memory space
- AJMP specifies the destination address as an 11-bit constant. Destination must be within a 2K block of AJMP.
- In all cases, programmer specifies the destination address to the assembler (using label or a 16-bit constant) and the assembler puts destination address into correct format.

Subroutines and Interrupts

- There are two versions of the CALL instruction: ACALL and LCALL using absolute and long addressing
- Generic CALL may be used if the programmer does not care which way the address is coded
- Either instruction pushes the contents of the PC on the stack and loads the PC with the address specified in the instruction
- Note that the PC will contain the address of the instruction following the CALL instruction when it gets pushed on the stack
- The PC is pushed on the stack low-byte first, high-byte second

Subroutines and Interrupts

- Example: Instruction LCALL COSINE is in code memory at addresses 0204_H through 0206_H, and subroutine COSINE begins at code memory address $043A_H$. Assume stack pointer contains $3A_H$ before this instruction. What internal RAM locations are altered and what will their new values be after this instruction is executed?

Address	Contents
$3B_H$	02_H
$3C_H$	07_H
81_H	$3C_H$

Subroutines and Interrupts

- Subroutines should end with an RET instruction
- RET pops the last two bytes off the stack and places them in the PC
- Jumping in or out of a subroutine any other way usually fouls up the stack and causes the program to crash

Conditional Jump

- The 8051 offers a variety of conditional jump instructions
- JZ and JNZ tests the accumulator for a particular condition
- DJNZ (decrement and jump if not zero) is a useful instruction for building loops
- To execute a loop N times, load a register with N and terminate the loop with a DJNZ to the beginning of the loop
- CJNE (compare and jump if not equal) is another conditional jump instruction
- CJNE: two bytes in the operand field are taken as unsigned integers. If the first one is less than the second one, the carry is set
- Example: It is desired to jump to BIG if the value of the accumulator is greater than or equal to 20_H

```
CJNE A,#20H,$+3
JNC BIG
```

  - $ is an assembler symbol representing the address of the current instruction
  - Since CJNE is a 3-byte instruction, $+3 is the address of next instruction JNC

Summary

- 8051 overview
- Hardware
- Instruction set