Instruction Set Summary
Introduction

• The MCS-51 instruction set is optimized for 8-bit control applications.
  – It provides a variety of fast, compact address modes for accessing
    the internal RAM to facilitate operations on small data structures.
• The 8051 instructions have 8-bit opcodes $2^8 = 256$ instructions.
  – 255 are implemented and 1 is undefined.
  – There are 139 one-byte instructions, 92 two-byte instructions, and
    24 three-byte instructions. (139+92+24=255)
• Addressing modes are in integral part of each computer’s instruction set. There are 8 modes available:
  – Register, Direct, Indirect, Immediate, Relative, Absolute, Long, and Indexed.
Register Addressing Mode

- **Ex1: Add the contents of Register 7 to the accumulator.**
  - ADD A, R5
  - ADD A, Rn
    - 1 Byte and 1 Cycle
    - Encoding: 00101rrr (00101 indicates the instruction, rrr indicate the register) $2^3=8 \Rightarrow n=0-7$
    - Operation: $(A) \leftarrow (A)+(Rn)$

- **ADD A, R5 (Encoding: 00101101B)**
  - ADD (00101)
    - Encoding: 00101 0 0 1 0 1 1 0 1
  - MOV (11101)
    - Encoding: 1 1 1 1 0 1 1 0 1
Register Addressing Mode

- Ex2: What is the opcode for the following instruction?
  - MOV A,R7
  - Solution: EFH (=11101111B)

- Ex3: What is the opcode for the following instruction?
  - INC DPTR
  - Solution: A3H (=10100011B)

- Ex4: What is the opcode for the following instruction?
  - MUL AB
  - Solution: A4H (=10100100B); B<=high-byte, A<=low-byte.
Register Addressing Mode

- Error

\[
\begin{align*}
\text{MOV} & \quad \text{DPTR, A} \quad ; \text{DPTR (16 bits) but A (8 bits)} \\
\text{MOV} & \quad \text{R4, R7} \quad ; \text{MOV A, R7 or MOV R4, A} \\
\text{MUL} & \quad \text{R4R2} \quad ; \text{MUL AB} \\
\text{DIV} & \quad \text{R1A} \quad ; \text{DIV AB} \\
\text{ADD} & \quad \text{R4, R2} \quad ; \text{ADD A, R2}
\end{align*}
\]
Direct Addressing Mode

- Direct dressing can access any on-chip variable or hardware register.
- Q1: Move the content of the accumulator to Port 1.
  - Solution: MOV P1, A
    - General form: MOV direct, A
    - Opcode: 11110101 aaaaaaaa ➔ 11110101B 10010000B (F5H,90H)

- Ex3-3: What are the machine-language bytes for the following instruction?
  - Solution: MOV SCON, #55H
    - General form: MOV direct, #data
    - Opcode: 01110101 aaaaaaaa dddddd ➔ (75H,98H,55H)
Direct Addressing Mode

- Some examples

\[
\begin{align*}
\text{MOV} & \quad R0, \quad 40H \\
\text{MOV} & \quad 56H, \quad A \\
\text{MOV} & \quad R4, \quad 7FH \\
\text{MOV} & \quad A, \quad 4 \quad ; \quad \text{MOV} \quad A, \quad R4 \\
\text{MOV} & \quad A, \quad 7 \quad ; \quad \text{MOV} \quad A, \quad R7 \\
\text{MOV} & \quad 7, \quad 3 \quad ; \quad \text{MOV} \quad R7, \quad R3 \\
\text{MOV} & \quad B, \quad 5 \quad ; \quad \text{MOV} \quad B, \quad R5
\end{align*}
\]
Direct Addressing Mode

- The SFR can be accessed by their name or by their addresses
  
  ```
  MOV 0E0H, #55H ; MOV A, #55H
  MOV 0F0H, #25H ; MOV B, #25H
  ```

- Stack and direct addressing mode
  
  ```
  PUSH A ; This is an error!!
  PUSH 0E0H ; This is correct!!!
  PUSH 3 ; Push R3 to the stack
  ```

- Q2: Show the code to push R5, R6, A onto the stack and pop them back into R2, R3, and B.
Indirect Addressing Mode

• Register or direct addressing can not be used since they require operand addresses to be known at assemble-time.

• Q1: if R1 contains 40H and the internal memory address 40H contains 55, the instruction
  \[ \text{MOV A, } @\text{R1} \]
  moves 55H into the accumulator.

• Ex3-4: (a) What is the opcode for the following instruction (b) what does the instruction do?
  \[ \text{MOV A, } @\text{R0} \]

• Q2: Clear internal RAM from address 60H to 7FH.
Indirect Addressing Mode

- Register is used as a “pointer” to the data ➔ make accessing data dynamic!!
- If the data is inside the CPU, only R0 and R1 are used (i.e. R2 ~R7 can NOT used for this case).
  \[
  \text{MOV A, @R0} \\
  \text{MOV @R1, B}
  \]
  If the data is outside the CPU, DPTR is used.
- Q3: Write a program to copy the value 55H to RAM memory locations 40H to 45H using (a) Direct addressing mode, (b) indirect addressing mode without a loop, and (c) indirect addressing mode with a loop.
Indirect Addressing Mode

- Q4: Write a program to clear 16 RAM locations starting at RAM address 60H.
- Q5: Write a program to copy a block of 10 bytes of data from RAM locations starting at 35H to RAM locations starting at 60H.
Immediate Addressing Mode

- When a source operand is a constant (e.g. #5AH) rather than a variable.
- Q1: What are the opcode for the following instruction?
  MOV A, #12
  MOV DPTR, #8000H

- Ex3-5: What are the hexdecimal and binary machine language bytes for the following instruction?
  ADD A, #15
Immediate Addressing Mode

- Q2:

  MOV A, #12
  MOV R4, #62H
  MOV B, #01010011B
  MOV DPTR, #8000H \( \Rightarrow DPH=80H, DPL=00H \)

- Q3:

  COUNT EQU 30
  MOV R4, #COUNT
  MOV DPTR, #MYDATA (or #0200H)

  ORG 200H

  MYDATA: DB “America”
Relative Addressing Mode

- Relative addressing is used only with certain jump instructions. A relative address (or offset) is an 8-bit signed value, which is added to the program counter (PC) to form the address of the next instruction executed. (Range: -128 ~ +127)
- Jump “Ahead” and Jump “Back”.
- Ex3-6: The instruction

  **SJMP 9030H**

  is in memory location 9000H and 9001H. What are the machine language bytes for the instructions.
Relative Addressing Mode

• Ex3-7: An SJMP instruction with a machine language representation of 80H, F6H is in memory location 0802H and 0803H. To what address will the jump occur?
Absolute Addressing Mode

- Absolute addressing is used only with the ACALL and AJMP instructions. These 2 bytes allow branching within the current 2K page of code memory by providing the 11 least-significant bits of the destination address in the opcode (A10-A8) and byte 2 of the instruction (A7-A0).

- Q1: if the label of THERE represents an instruction at address 0F46H, and the instruction

  \[
  \text{AJMP THERE}
  \]

  is in memory locations 0900H and 0901H. What are the machine code for the AJMP instruction?

- EX3-8: An ACALL instruction is in memory locations 1024H and 1025H. The subroutine to which the call is directed begins in memory location 17A6H. What are the machine language bytes for the ACALL instruction?
Long Addressing Mode

- Long addressing is used only with the LCALL and LJMP instructions. These 3-byte instructions include a full 16-bit destination address. The advantage is that the full 64K code space may be used.
- Ex3-9: What are the machine language bytes for the following instruction?

  LJMP 8AF2H
Index Addressing Mode

- Indexed addressing mode and on-chip ROM access is used in accessing data elements of look-up table entries located in the program ROM.
- Indexed addressing uses a base register (PC or DPTR) and an offset (ACC) in forming the effective address for a JMP or MOVC instruction. Jump tables or look-up tables are easily created using indexed addressing.
- MOVC: this is because the data element is stored in ROM rather than RAM. Here ‘C’ denotes Code (ROM).
- Ex3-10: What is the opcode for the following instruction? MOVC A, @A+DPTR
Index Addressing Mode

- **Q1**: In this program, assume that the word ‘USA’ is burned into ROM locations starting at 200H, and that the program is burned into ROM location starting at 0H.
- **Q2**: Assuming that ROM space starting at 250H contains “America”, write a program to transfer the bytes into RAM locations starting at 40H.
Look-up Tables and the Use of Index Addressing Mode

- Q3: Write a program to get the x value from P1 and send $x^2$ to P2 continuously.
Arithmetic Instructions

- **Ex3-11**: The accumulator contains $63H$, R3 contains $23H$ and the PSW contains $00H$: (a) what is the hexadecimal content of the accumulator and the PSW after execution of the following instruction?
  
  **ADD A, R3**
  
  (b) What is the content of the accumulator in decimal after execution of this instruction?

- **Ex3-12**: Illustrate an instruction sequence to subtract the content of R6 from R7 and leave the result in R7.

- **Q1**: **INC 7FH?**
Arithmetic Instructions

- Ex3-13: Suppose the 8051 did not have an instruction to directly increment an internal RAM location. How could this operation be achieved?

- Ex3-14: The accumulator contains 55H, the B register contains 22H, and the PSW contains 00H. What are the contents of these registers after execution of the following instruction?

- Ex3-15: The accumulator contains 1FH. What is the largest value that could be in register B such that the OV bit would not be set after executing the following instruction?

- Ex3-16: illustrate how to add two 4-digit binary-coded decimal (BCD). The first is in internal memory locations 40H and 41H, and the second is in locations 42H and 43H. The most significant digits are in locations 40H and 42H. Place the BCD result in locations 40H and 41H.
Signed and Unsigned Numbers

- Signed and unsigned numbers
- Unsigned addition: 00~FFH (0~255 decimal) for 8-bit data.
  \[ \text{ADD} \ A, \ \text{source} \]

- Q2: Show how the flag register is affected by the following instructions
  
  \[ \text{MOV} \ A, \ #0F5H \]
  \[ \text{ADD} \ A, \ #0BH \]

- Q3: Find the sum of the values of locations 40 to 44 and put low byte to A, and high byte to R7 (hint: using JNC)
ADDC: Addition with a Carry

- ADDC is used when adding two 16 bit data operands, we need to be concerned with the propagation of a carry from the low byte to the high byte.
- Q4: Write a program to add two 16-bit numbers. The numbers are 3CE&H and 3B8DH. Place the sum in R7 (high byte) and R6 (low byte).
BCD: Binary Coded Decimal

- **Unpacked BCD (1 byte data)**
  - \( 9 \rightarrow 0000\ 1001B \) (1 byte)
  - \( 5 \rightarrow 0000\ 0101B \) (1 byte)

- **Packed BCD (1 byte data): it is twice as efficient in storing data compared to unpacked BCD.**
  - \( 59H \rightarrow 0101\ 1001B \) (1 byte)

- **Problem with adding BCD numbers**
  - `MOV A, #17H 0001 0111`
  - `ADD A, #28H 0010 1000`
  - The sum is `0011 1111 =3FH`
  - \( \Rightarrow \) This is NOT BCD number!!

- A BCD number only have digits from `0000` to `1001` (0 ~9).
BCD: Binary Coded Decimal

- Method 1: $17H + 28H = 45H$ (the calculation is the same as decimal) = 0100 0101  ➔ this is a BCD number
- Method 2: $17H + 28H = D9H$ (hexdecimal calculation). Since there is D (>9) in high nibble, we add 6 for that.
  
  \[ \text{D9H} + 60H = 139H \]
- DA: Decimal Adjust is designed to correct the BCD addition problem. DA will add 6 to the lower nibble or higher nibble if necessary. (nibble=4 bits)

\[
\begin{align*}
\text{MOV} & \quad A, \#47H \\
\text{MOV} & \quad B, \#25H \\
\text{ADD} & \quad A, B \quad \text{➔ A=6CH} \\
\text{DA} & \quad A \quad \text{➔ A=72H (BCD value)}
\end{align*}
\]
BCD: Binary Coded Decimal

- Error: Instruction DA can only be located after ADD.
  - \textit{INC} \hspace{1em} A
  - \textit{DA} \hspace{1em} (x)
- Algorithm:
  - If (lower nibble > 9) \text{ or } (AC=1)  
    - Add 0110 to the nibble for BCD
  - If (higher nibble > 9) \text{ or } (CY=1)  
    - Add 0110 to the nibble for BCD
- Q5: Assume that 5 BCD data item are stored in RAM starting at 40H. Write a program to sum all numbers. The result should be BCD.
  \[ 40=(71H), \hspace{1em} 41=(11H), \hspace{1em} 42=(65H), \hspace{1em} 43=(59H), \hspace{1em} \text{and} \hspace{1em} 44=(37H) \]
SUBB: Subtract with Borrow

- **SUBB** A, Source ; A=A-Source-CY
- **SUBB** with CY=0
  - 2’s complement for source operand
  - Add it to A
  - Invert the carry
- Q6: Show the steps involved in the following.
  
  ```
  CLR C
  MOV A, #3FH
  MOV R3, #23H
  SUBB A, R3
  ```
SUBB: Subtract with Borrow

- Q7: Analyze the following program.

  ```
  CLR C
  MOV A, #4CH
  SUBB A, #6EH
  JNC Next
  CPL A
  INC A
  Next: MOV R1, A
  ```

- Note: after SUBB with CY=0 ➔ the result is positive.
  CY=1 ➔ the result is negative.
SUBB: Subtract with Borrow

- SUBB with CY=1 for multi-byte numbers.
- Q8: Analyze the following programs.

```
CLR C
MOV A, #62H
SUBB A, #96H
MOV R7, A
MOV A, #27H
SUBB A, #12H
MOV R6, A
```
Unsigned Multiplication and Division: Need A and B

- Q9:
  
  \[
  \text{MOV} \ A, \ #25H \\
  \text{MOV} \ B, \ #65H \\
  \text{MUL} \ AB
  \]

- Q10:
  
  \[
  \text{MOV} \ A, \ #95 \\
  \text{MOV} \ B, \ #10 \\
  \text{DIV} \ AB
  \]

- Instruction ‘DIV AB’
  - if the numerator \( \neq 0 \) \( \Rightarrow \) always makes \( CY=0, \ OV=0 \)
  - If numerator = 0 \( \Rightarrow \) \( CY=0, \ OV=1 \) (indicates an error).
ADC Application

- Q11: Write a program to get hex data in the range of 00~FFH from P1 and convert it to decimal. Save the digits in R7 (LSB), R6, and R5. Analyze the program, assuming that P1 has a value of FDH for data. (hex converts to decimal using DIV AB)
# Signed Number

<table>
<thead>
<tr>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

If D7 = 1 → Negative

-128: 1000 0000
-127: 1000 0001
-126: 1000 0002

If D7 = 0 → Positive

+0 = 0000 0000
+1 = 0000 0001

-1: 1111 1111

127: 0111 1111
Overflow Problem

• Q12: Show how the 8051 would represent –5.
• Overflow Occurs
  A carry from D6 to D7, CY=0
  CY=1, but NO carry from D6 to D7.

• Q13:
  \[ MOV \ A, \ #96 \]
  \[ MOV \ R1, \ #70 \]
  \[ ADD \ A, \ R1 \]

• Q14:
  \[ MOV \ A, \ #-128 \]
  \[ MOV \ R4, \ #-2 \]
  \[ ADD \ A, \ R4 \]
Overflow Problem

• Q15:
  
  \[
  \text{MOV A, } \#-2 \\
  \text{MOV R1, } \#-5 \\
  \text{ADD A, R1}
  \]

• Q16:
  
  \[
  \text{MOV A, } \#7 \\
  \text{MOV R4, } \#18 \\
  \text{ADD A, R4}
  \]

• **Unsigned** Number (0~255)
  
  Addition ➔ we must monitor the CY (using JNC and JC)

• **Signed** Number (-128 ~+127)
  
  Addition ➔ we must monitor OV (using JB PSW.2 and JNB PSW.2)
Logical Instructions and Programs

- Ex3-17: Illustrate two ways to rotate the content of the accumulator three positions to the left.
- Q1:
  - ANL (AND operation)
  - ORL (OR operation)
  - XRL (XOR operation)
  - CPL (complement)
- Q2: Read the test P1 to see whether it has the value of 45H, if it does, send 99H to P2; otherwise, it stay cleared.
- 2’s complement
  - (CPL A) and (ADD A, #1)
  - (CPL A) and (INC A)
CJNE: Compare and Jump if NOT Equal

- **CJNE**
  
  ```
  CJNE A, #67H, Next ; if A!=67H and Jump
  : ; A still has its original value
  Next: …
  ```

- **CJNE** destination, source ; destination can be A or Reg
  
  - destination > source ➔ CY=0
  - destination < source ➔ CY=1
  
  ```
  CJNE R5, #80, NOT_equal
  
  statement1 ; R5=80
  NOT_equal: JNC Next ; jump if CY=0
  
  statement2 ; R5 <80
  
  Next: statement2 ; R5 >80
  ```
CJNE: Compare and Jump if NOT Equal

• Q5: Assume P1 is an input port connected to a temperature sensor. Write a program to read the temperature and test it for the value 75.

  if T = 75  \Rightarrow A=75
  T < 75  \Rightarrow R1=T
  T > 75  \Rightarrow R2=T
CJNE: Compare and Jump if NOT Equal

- **Q6:** Write a program to monitor P1 continuously for the value 63H, it should get out of the monitoring only if P1 = 63H.

- **Q7:** Assume internal RAM memory locations 40H~44H contain the daily temperature for 5 days. Such as 40H=(76), 41H=(79), 42H=(69), 43H=(65), 44H=(62). Search to see if only if the value equals to 65. If value 65 does exist in the table, give its location to R4; otherwise, make R4=0.
Rotate Right

- RR A: rotate right A

- Q8:
  
  MOV  A, #36H
  RR  A
  RR  A
  RR  A
Rotate Left

- RL A: rotate left A

- Q9:
  
  MOV A, #72H
  RL A
  RL A
Rotate Right with Carry

- RRC A: rotate right thru carry

![Diagram showing MSB, LSB, and CY connections]

- Q10-1:
  
  CLR C
  MOV A, #26H
  RRC A
Rotate Left with Carry

- **RLC A**: rotate left thru carry

![Diagram of rotate left with carry](image)

- **Q10-2:**
  - CLR C
  - MOV A, #5AH
  - RLC A
• Q11:
  MOV  A, #72H
  SWAP A

• Q12: Write a program that finds the number of 1’s in a given byte.
## BCD & ASCII

<table>
<thead>
<tr>
<th>Key</th>
<th>ASCII</th>
<th>Binary</th>
<th>BCD (unpacked)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>30</td>
<td>011 0000</td>
<td>0000 0000</td>
</tr>
<tr>
<td>1</td>
<td>31</td>
<td>011 0001</td>
<td>0000 0001</td>
</tr>
<tr>
<td>2</td>
<td>32</td>
<td>011 0010</td>
<td>0000 0010</td>
</tr>
<tr>
<td>3</td>
<td>33</td>
<td>011 0011</td>
<td>0000 0011</td>
</tr>
<tr>
<td>4</td>
<td>34</td>
<td>011 0100</td>
<td>0000 0100</td>
</tr>
<tr>
<td>5</td>
<td>35</td>
<td>011 0101</td>
<td>0000 0101</td>
</tr>
<tr>
<td>6</td>
<td>36</td>
<td>011 0110</td>
<td>0000 0110</td>
</tr>
<tr>
<td>7</td>
<td>37</td>
<td>011 0111</td>
<td>0000 0111</td>
</tr>
<tr>
<td>8</td>
<td>38</td>
<td>011 1000</td>
<td>0000 1000</td>
</tr>
<tr>
<td>9</td>
<td>39</td>
<td>011 1001</td>
<td>0000 1001</td>
</tr>
</tbody>
</table>
BCD ➔ ASCII

- Real Time Clock: RTC (Time and Date)
  Need BCD ➔ ASCII for display

- Packed BCD to ASCII
  - DS5000T microcontroller has a RTC. The RTC provides the time of day (Hour:Minute:Second) and the date regardless of whether power off or on.

  Packed BCD ➔ Unpacked BCD ➔ ASCII

  29H (0010 1001) ➔ 02H (0000 0010) ➔ 32H (0011 0010)
  & 09H (0000 1001) & 39H (0011 1001)
### ASCII ➞ Packed BCD

<table>
<thead>
<tr>
<th>Key</th>
<th>ASCII</th>
<th>Unpacked BCD</th>
<th>Packed BCD</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>34</td>
<td>0000 0100</td>
<td>0100 0111</td>
</tr>
<tr>
<td>7</td>
<td>37</td>
<td>0000 0111</td>
<td></td>
</tr>
</tbody>
</table>
Example: Packed BCD $\rightarrow$ ASCII

- Q13: Assume the register A has packed BCD, write a program to convert packed BCD to two ASCII numbers and place them in R2 and R6.
Data Transfer Instructions

- **Ex3-22**: illustrate an instruction sequence to read the content of external RAM locations 10F4H and 10F5H and place the values read in R6 and R7, respectively.
- **Ex3-23**: Write a subroutine called SQUARE to compute the square of an integer between 0 and 9. Enter the subroutine with the integer in A, and return with the square of the integer in A. Write two versions of the subroutine: (a) using a look-up table, and (b) without using a look-up table. Then (c) illustrate a call sequence to convert 6 to its square, 36.
Boolean Instructions

Q1: Write an instruction sequence for the AND gate.