

EmbeddedCraft
crafting intelligent systems

IMBUENT
Innovative Microcontroller-Based User Experience Technologies

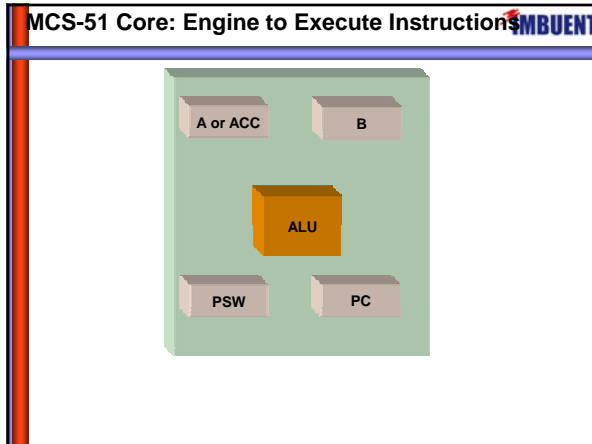
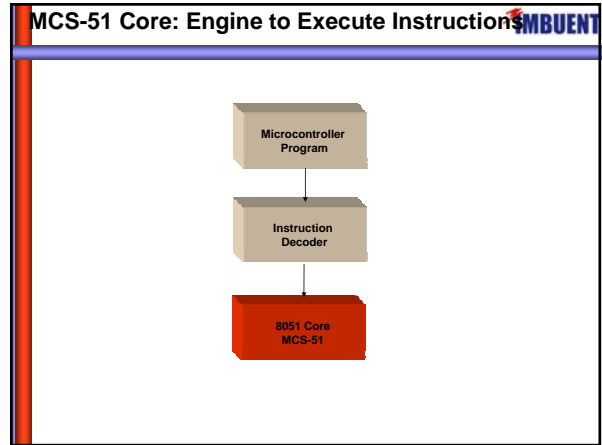
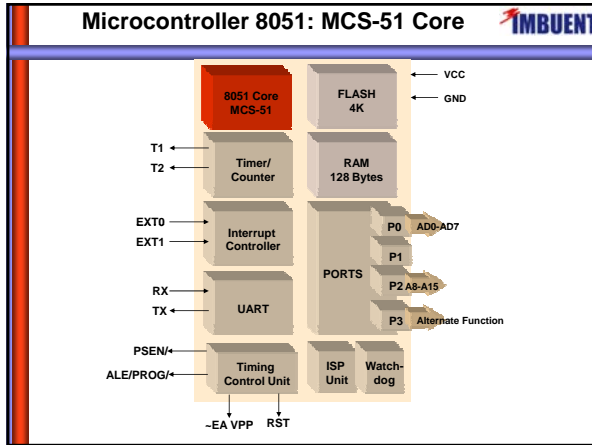
8051 MICROCONTROLLER

SESSION - 5

Embedded System-Session 5

IMBUENT

- >> Remaining Instruction Set
- >> LED Blinking
- >> Interfacing of Switch
- >> Other Example Program



MCS-51 Core: Engine to Execute Instructions

A	E0H	B	F0H	PSW	D0H	PC (16 bits)
8 bit		8 bit		8 bit		

A : Main Accumulator
Always store first operand (for A/L operation)

B :
used only in DIV and MUL instructions
can be used as general storage

PSW: Program Status Word (PSW) contains status bits that reflect the current state of the CPU

PC: Sequence the execution of Instruction of program
Always point to next instruction .

MUL and DIV

A is 135 decimal, B is 36 decimal.
What would be the value in each register after executing the instruction MUL AB?

A x B = 4860 = 0001 0010 1111 1100 (12FC H)
A <= FCH B <= 12H

```

$MOD51
ORG 0000H
MOV A, #135
MOV B, #36
MUL AB
END
    
```

MUL and DIV

A = 135, B = 36. What would be the value in each register after execution of the instruction DIV AB?

A <= Quotient = 3, B <= Remainder = 27 (1B H)

```

$MOD51
ORG 0000H
MOV A, #135
MOV B, #36
DIV AB
END
    
```

CY/AC/OV Flags

```

$MOD51
ORG 0000H
MOV A, #39H
MOV R1, #99H
ADD A, R1
END
    
```

A: 0011 1001
+ R1: 1001 1001

1101 0010

PSW							
CY	AC	FO	RS1	RS0	OV	F1	P
0	1	0	0	0	0	0	0

CY/AC/OV Flags

```

$MOD51
ORG 0000H
MOV A, #0A9H
MOV R1, #99H
ADD A, R1
END
    
```

A: 1010 1001
+ R1: 1001 1001

1 0100 0010

PSW							
CY	AC	FO	RS1	RS0	OV	F1	P
1	1	0	0	0	0	0	0

CY/AC/OV Flags

OV ← C7 ⊕ C6

```

$MOD51
ORG 0000H
MOV A, #0A9H (-87)
MOV R1, #0D9H (-39)
ADD A, R1
END
    
```

A: 1010 1001
+ R1: 1101 1001

1 1000 0010

PSW							
CY	AC	FO	RS1	RS0	OV	F1	P
1	1	0	0	0	0	0	0

Conclusion of OV and CY flag

CY flag: overflow situation in Unsigned Number CY ← C7

OV flag: overflow in case of Signed Number OV ← C7 ⊕ C6

- Example One Byte Register
- Signed No: -128 to +127
- Unsigned No: 0 to 255

7	6	5	4	3	2	1	0
---	---	---	---	---	---	---	---

BCD Numbers...

Each decimal digit is stored in a four-bit **nibble**.

Decimal:	0	1	2	3	4	5	6	7	8	9
BCD:	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001

Un Packed No
 -Each digit is stored into one Byte
 -Ex: storing 1234

RAM

BCD Numbers...

Packed No
 - Two BCD digits is stored into one Byte
 - Ex: storing 1234
 MOV 30H, #12H
 MOV 31H, #34H

- Second digit may also be used to store sign of digit
 1100 For + Positive
 1101 For - Negative

Ex: +4 = 1100 0100 [C4]
 -4 = 1101 0100 [D4]

Practical implementation in Matrix Keyboard

RAM

Instruction Type

8051 instructions are divided among five groups:

- Data transfer
- Arithmetic
- Logical
- Boolean variable
- Program branching

Assumptions:
 # = Immediate No
 @ = Indirect
 M = Memory
 R = Register

Data transfer instructions (internal)

MOV A, [# / R / @ / M]
 Example:
 MOV A, #45H A ← 45H
 MOV A, R0 A ← R0
 MOV A, @R1 A ← M[R1]
 MOV A, 30H A ← M[30H]

Mnemonic	Operation	Addressing Modes				Execution Time @ 12MHz (µs)
		Dir	Ind	Reg	Imm	
MOV A, <src>	A ← <src>	X	X	X	X	1
MOV <dest>, A	<dest> ← A	X	X	X	X	1
MOV <dest>, <src>	<dest> ← <src>	X	X	X	X	2
MOV DPTR, # data 16	DPTR ← 16-bit immediate constant				X	2
PUSH <src>	INC SP; MOV 'SP', <src>	X				2
POP <dest>	MOV <dest>, 'SP'; DEC SP	X				2
XCH A, <byte>	ACC and <byte> Exchange Data	X	X	X	X	1
XCH A, @Ri	ACC and @ Ri exchange low nibbles	X				1

MOV R, [# / M]
 Example:
 MOV R1, #45H R1 ← 45H
 MOV R1, 20H R1 ← M[20H]

MOV @, [# / M]
 Example:
 MOV @R1, #45H M[R1] ← 45H
 MOV R1, 20H M[R1] ← M[20H]

MOV DPTR, #4567H

Data transfer instructions (internal)

XCHA A, [M / R / @]
 Example:
 XCHA A, R1 A ↔ R1
 XCHA A, 20H A ↔ M[20H]
 XCHA A, @R0 A ↔ M[R0]

XCHD A, @
 Exchange Low order Nibble
 Example:
 XCHD A, @R1

Suppose A = 34H and M[20] = 98H
 Let R0 = 20H

XCHD A, @R0
 A = 38H and M[20] = 94H

Mnemonic	Operation	Addressing Modes				Execution Time @ 12MHz (µs)
		Dir	Ind	Reg	Imm	
MOV A, <src>	A ← <src>	X	X	X	X	1
MOV <dest>, A	<dest> ← A	X	X	X	X	1
MOV <dest>, <src>	<dest> ← <src>	X	X	X	X	2
MOV DPTR, # data 16	DPTR ← 16-bit immediate constant				X	2
PUSH <src>	INC SP; MOV 'SP', <src>	X				2
POP <dest>	MOV <dest>, 'SP'; DEC SP	X				2
XCH A, <byte>	ACC and <byte> Exchange Data	X	X	X	X	1
XCH A, @Ri	ACC and @ Ri exchange low nibbles	X				1

PUSH and POP

Stack:
 Used to store data temporarily in memory locations of RAM

Used in parameter passing, Subroutine, Interrupt call etc.

SP : 8 bit register address : 81H
 After Reset SP ← 07H
 Suppose SP = 30, and R8 = 78H
 PUSH R8

PUSH direct [SP] < --- [SP] + 1
 [SP+1] < --- [direct]

POP direct [direct] < --- [SP+1]
 [SP] < --- [SP] - 1

RAM

PUSH and POP

PUSH direct [SP] < --- [SP] + 1
 [SP:M] < --- [direct]

Suppose SP = 30, and R8 = 78H
 PUSH R8

SP + 1

SP

40
3F
:
33
32
31
30

RAM

PUSH and POP

PUSH direct [SP] < --- [SP] + 1
 [SP:M] < --- [direct]

POP direct [direct] < --- [SP:M]
 [SP] < --- [SP] - 1

Suppose SP = 30, and R8 = 78H
 PUSH R8

SP + 1
M[SP] <<< R8

SP

40
3F
:
33
32
31
30

RAM

PUSH and POP

PUSH direct [SP] < --- [SP] + 1
 [SP:M] < --- [direct]

POP direct [direct] < --- [SP:M]
 [SP] < --- [SP] - 1

Suppose SP = 30, and R8 = 78H
 PUSH R8

SP + 1
M[SP] <<< R8

SP

40
3F
:
33
32
31
30

RAM

PUSH and POP

- Use PUSH in the beginning of subroutine to store content of A, B,
- and before returning back from subroutine call POP Instruction in
- Let SP at 07H,

HANDLER: PUSH ACC

SP

40
B
:
A
9
8
7

RAM

PUSH and POP

- Use PUSH in the beginning of subroutine to store content of A, B,
- and before returning back from subroutine call POP Instruction in
- Let SP at 07H,

HANDLER: PUSH ACC
 PUSH B

SP

40
B
:
A
9
8
7

RAM

PUSH and POP

- Use PUSH in the beginning of subroutine to store content of A, B,
- and before returning back from subroutine call POP Instruction in
- Let SP at 07H,

HANDLER: PUSH ACC
 PUSH B
 PUSH PSW

SP

40
B
:
A
9
8
7

RAM

PUSH and POP

- Use PUSH in the beginning of subroutine to store content of A, B,
- and before returning back from subroutine call POP Instruction in
- Let SP at 07H,

HANDLER: PUSH ACC
 PUSH B
 PUSH PSW

RAM

PUSH and POP

- Use PUSH in the beginning of subroutine to store content of A, B,
- and before returning back from subroutine call POP Instruction in
- Let SP at 07H,

HANDLER: PUSH ACC
 PUSH B
 PUSH PSW

POP PSW

RAM

PUSH and POP

- Use PUSH in the beginning of subroutine to store content of A, B,
- and before returning back from subroutine call POP Instruction in
- Let SP at 07H,

HANDLER: PUSH ACC
 PUSH B
 PUSH PSW

POP PSW
 POP B
 POP B

RAM

PUSH and POP

- Use PUSH in the beginning of subroutine to store content of A, B,
- and before returning back from subroutine call POP Instruction in
- Let SP at 07H,

HANDLER: PUSH ACC
 PUSH B
 PUSH PSW

POP PSW
 POP B
 POP ACC

RAM

PUSH and POP

- Use PUSH in the beginning of subroutine to store content of A, B,
- and before returning back from subroutine call POP Instruction in
- Let SP at 07H,

HANDLER: PUSH ACC
 PUSH B
 PUSH PSW

POP PSW
 POP B
 POP ACC

RAM

Arithmetic Instruction...

- One operand is always A

ADD A, [# / R / M / @]

Example:

ADD A, #34H A ← A + 34H

ADD A, R0 A ← A + R0

ADD A, 40H A ← A + M [40]

ADD A, @R0 A ← A + M [R0]

Mnemonic	Operation	Addressing Modes			Execution Time in X1 Mode (1/2 Mhz per)	
		Dr	Ind	Reg	In	M
ADD A, #d16	A ← A + d16	X	X	X	X	

Arithmetic Instruction...

- One operand is always A

ADDC A, [# / R / M / @]

Example:

ADDC A, #34H $A \leftarrow A + 34H + C$
 ADDC A, R0 $A \leftarrow A + R0 + C$
 ADDC A, 40H $A \leftarrow A + M[40] + C$
 ADDC A, @R0 $A \leftarrow A + M[R0] + C$

Mnemonic	Operation	Addressing Modes				Execution Time in X1 Mode (@12 MHz (µs))
		Dir	Ind	Reg	Im	
ADD A, <i>dst</i> # <i>val</i>	$A \leftarrow A + \text{dst} \# \text{val}$	X	X	X	X	
ADDC A, <i>dst</i> # <i>val</i>	$A \leftarrow A + \text{dst} \# \text{val} + C$	X	X	X	X	1

Arithmetic Instruction...

- One operand is always A

SUBB A, [# / R / M / @]

Example:

SUBB A, #34H $A \leftarrow A + 34H - C$
 SUBB A, R0 $A \leftarrow A + R0 - C$
 SUBB A, 40H $A \leftarrow A + M[40] - C$
 SUBB A, @R0 $A \leftarrow A + M[R0] - C$

Mnemonic	Operation	Addressing Modes				Execution Time in X1 Mode (@12 MHz (µs))
		Dir	Ind	Reg	Im	
ADD A, <i>dst</i> # <i>val</i>	$A \leftarrow A + \text{dst} \# \text{val}$	X	X	X	X	
ADDC A, <i>dst</i> # <i>val</i>	$A \leftarrow A + \text{dst} \# \text{val} + C$	X	X	X	X	1
SUBB A, <i>dst</i> # <i>val</i>	$A \leftarrow A + \text{dst} \# \text{val} - C$	X	X	X	X	1

Arithmetic Instruction...

- One operand is always A

INC [A/R/M/@]

Example:

INC A $A \leftarrow A + 1$
 INC R3 $R3 \leftarrow R3 + 1$
 INC 20H $M[20] \leftarrow M[20] + 1$
 INC @R0 $M[R0] \leftarrow M[R0] + 1$

Mnemonic	Operation	Addressing Modes				Execution Time in X1 Mode (@12 MHz (µs))
		Dir	Ind	Reg	Im	
ADD A, <i>dst</i> # <i>val</i>	$A \leftarrow A + \text{dst} \# \text{val}$	X	X	X	X	
ADDC A, <i>dst</i> # <i>val</i>	$A \leftarrow A + \text{dst} \# \text{val} + C$	X	X	X	X	1
SUBB A, <i>dst</i> # <i>val</i>	$A \leftarrow A + \text{dst} \# \text{val} - C$	X	X	X	X	1
INC A	$A \leftarrow A + 1$					Accumulator only 1
INC <i>dst</i>	$\text{dst} \leftarrow \text{dst} + 1$	X	X	X	X	1
INC DPTR	$\text{DPTR} \leftarrow \text{DPTR} + 1$					Data Pointer only 2

Arithmetic Instruction...

- One operand is always A

INC [A/R/M/@]

Example:

INC A $A \leftarrow A + 1$
 INC R3 $R3 \leftarrow R3 + 1$
 INC 20H $M[20] \leftarrow M[20] + 1$
 INC @R0 $M[R0] \leftarrow M[R0] + 1$

Mnemonic	Operation	Addressing Modes				Execution Time in X1 Mode (@12 MHz (µs))
		Dir	Ind	Reg	Im	
ADD A, <i>dst</i> # <i>val</i>	$A \leftarrow A + \text{dst} \# \text{val}$	X	X	X	X	
ADDC A, <i>dst</i> # <i>val</i>	$A \leftarrow A + \text{dst} \# \text{val} + C$	X	X	X	X	1
SUBB A, <i>dst</i> # <i>val</i>	$A \leftarrow A + \text{dst} \# \text{val} - C$	X	X	X	X	1
INC A	$A \leftarrow A + 1$					Accumulator only 1
INC <i>dst</i>	$\text{dst} \leftarrow \text{dst} + 1$	X	X	X	X	1
INC DPTR	$\text{DPTR} \leftarrow \text{DPTR} + 1$					Data Pointer only 2

Arithmetic Instruction...

- One operand is always A

DEC [A/R/M/@]

Example:

DEC A $A \leftarrow A - 1$
 DEC R3 $R3 \leftarrow R3 - 1$
 DEC 20H $M[20] \leftarrow M[20] - 1$
 DEC @R0 $M[R0] \leftarrow M[R0] - 1$

Mnemonic	Operation	Addressing Modes				Execution Time in X1 Mode (@12 MHz (µs))
		Dir	Ind	Reg	Im	
ADD A, <i>dst</i> # <i>val</i>	$A \leftarrow A + \text{dst} \# \text{val}$	X	X	X	X	
ADDC A, <i>dst</i> # <i>val</i>	$A \leftarrow A + \text{dst} \# \text{val} + C$	X	X	X	X	1
SUBB A, <i>dst</i> # <i>val</i>	$A \leftarrow A + \text{dst} \# \text{val} - C$	X	X	X	X	1
INC A	$A \leftarrow A + 1$					Accumulator only 1
INC <i>dst</i>	$\text{dst} \leftarrow \text{dst} + 1$	X	X	X	X	1
INC DPTR	$\text{DPTR} \leftarrow \text{DPTR} + 1$					Data Pointer only 2
DEC A	$A \leftarrow A - 1$					Accumulator only 1
DEC <i>dst</i>	$\text{dst} \leftarrow \text{dst} - 1$	X	X	X	X	1

Arithmetic Instruction...

- One operand is always A

MUL AB

DIV AB

Mnemonic	Operation	Addressing Modes				Execution Time in X1 Mode (@12 MHz (µs))
		Dir	Ind	Reg	Im	
ADD A, <i>dst</i> # <i>val</i>	$A \leftarrow A + \text{dst} \# \text{val}$	X	X	X	X	
ADDC A, <i>dst</i> # <i>val</i>	$A \leftarrow A + \text{dst} \# \text{val} + C$	X	X	X	X	1
SUBB A, <i>dst</i> # <i>val</i>	$A \leftarrow A + \text{dst} \# \text{val} - C$	X	X	X	X	1
INC A	$A \leftarrow A + 1$					Accumulator only 1
INC <i>dst</i>	$\text{dst} \leftarrow \text{dst} + 1$	X	X	X	X	1
INC DPTR	$\text{DPTR} \leftarrow \text{DPTR} + 1$					Data Pointer only 2
DEC A	$A \leftarrow A - 1$					Accumulator only 1
DEC <i>dst</i>	$\text{dst} \leftarrow \text{dst} - 1$	X	X	X	X	1
MUL AB	$AB \leftarrow AB \times B$					ACC and B only 4
DIV AB	$A \leftarrow A / B$ $B \leftarrow M[A/B]$					ACC and B only 4

Arithmetic Instruction...

- One operand is always A

MUL AB

DIV AB

Mnemonic	Operation	Addressing Modes			Exe. Time (ns)	Mnemonics (12 MHz)
		Dir	Ind	Reg		
ADD A, r8/r16	A ← A + r8/r16	X	X	X	1	
ADDC A, r8/r16	A ← A + r8/r16 + C	X	X	X	1	
SUBB A, r8/r16	A ← A - r8/r16 - C	X	X	X	1	
INC A	A ← A + 1			Accumulator only	1	
INC r8/r16	r8/r16 ← r8/r16 + 1	X	X	X	1	
INC DPTR	DPTR ← DPTR + 1			Data Pointer only	2	
DECA	A ← A - 1			Accumulator only	1	
DEC r8/r16	r8/r16 ← r8/r16 - 1	X	X	X	1	
MUL AB	A ← A * B B ← Mod(A/B)			ACC and B only	4	
DIV AB	A ← int(A/B) B ← Mod(A/B)			ACC and B only	4	
DA A	Decimal Adjust			Accumulator only	1	

Arithmetic Instruction BCD operation...

- To adjust the result as per BCD operation
- 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.

59 H

A

+ 1

5A H

A

+DA A

60 H

A

- If Least Significant Byte is > 9 or AC = 1
 - ADD 06 in answer
- If Most Significant Byte is > 9 then
 - ADD 60 in answer

Arithmetic Instruction BCD operation...

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

```

MOV 40H,#12H
MOV 41H,#79H
MOV 42H,#21H
MOV 43H,#18H
MOV A, 43H
ADD A, 41H
DA A
MOV 41H, A
MOV A, 42H
ADDC A, 40H
DA A
MOV 40H, A
    
```

```

      12-79
    + 21-18
    -----
      33-97
    
```

40	7F
42	
44	
46	
48	

RAM

Question

- Two 16-digit BCD numbers are in internal memory at locations 40H, 41H and 42H, 43H. The most significant bytes are in locations 40H and 42H. Add them and store the result in locations 44H and 45H.

40	7F
42	
44	
46	
48	

RAM

Instructions that affect flags

Instruction	Flag			Instruction	Flag		
	C	OV	AC		C	OV	AC
ADD	X	X	X	CLR C	0		
ADDC	X	X	X	CPL C	X		
SUBB	X	X	X	ANL C, #0	X		
MUL	0	X		ANL C, #0	X		
DIV	0	X		ORL C, #0	X		
DA	X			ORL C, #0	X		
RRNC	X			MOV C, #0	X		
RLC	X			CJNE	X		
SETB C	1						



<http://www.embeddedcraft.org>



<http://www.imbuent.com>

END OF SESSION 5