8051 instruction set summary

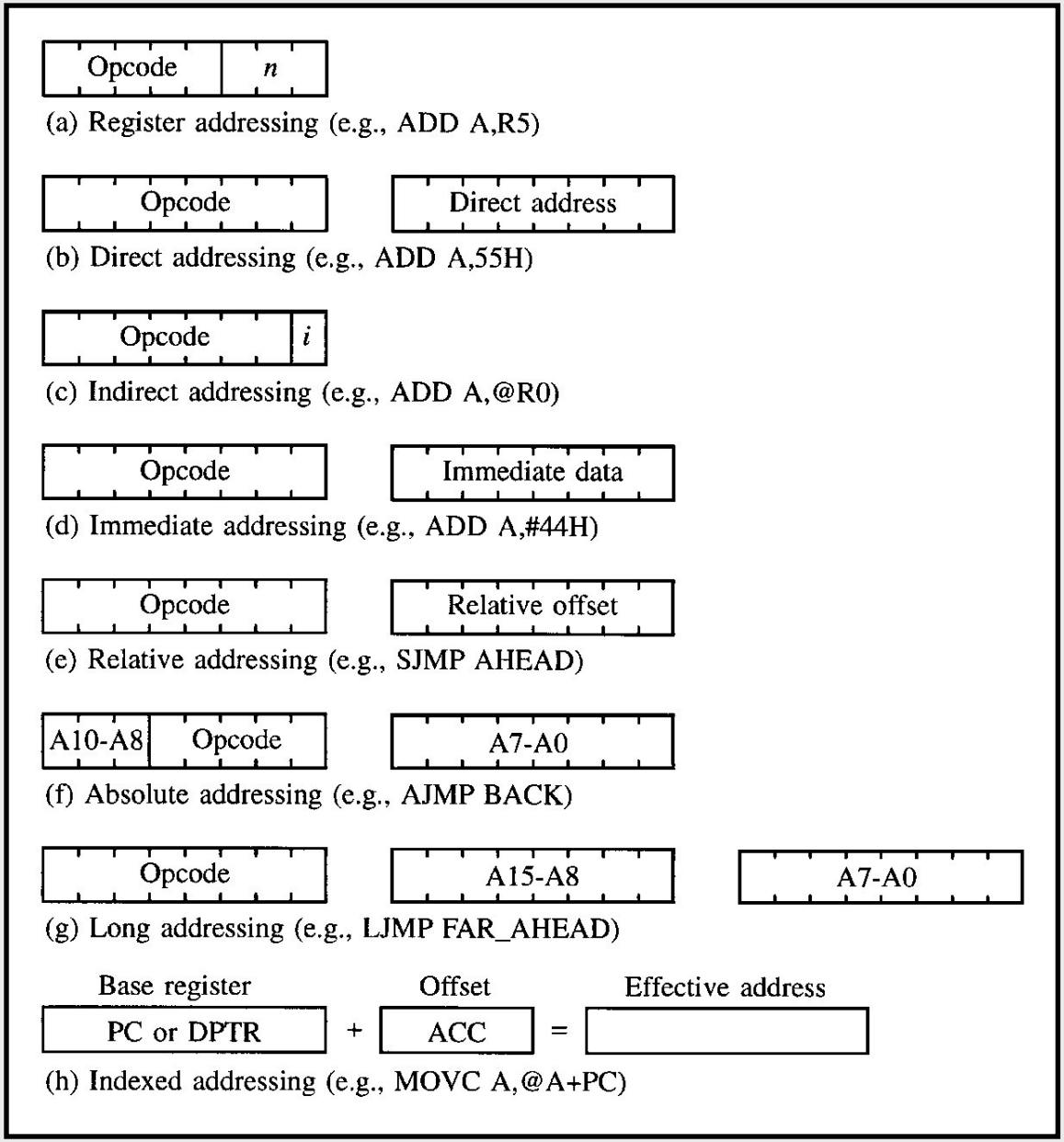
introduction

- optimized for 8-bit control applications
- fast and compact addressing modes
- extensive support for 1-bit variables
- 8-bit opcodes
- 255 defined instructions from possible 256
- 139 1-byte instructions
- 92 2-byte instructions
- 24 3-byte instructions

addressing modes

- *where is the data?*
- 8 addressing modes in 8051:
  - register
  - direct
  - indirect
  - immediate
  - relative
  - absolute
  - long
  - indexed

addressing modes (cont'd)



register addressing

```
ADD     A, R7     ; opcode: 00101111B (1 cycle)
```

- register-specific instructions
  - accumulator ; A
  - data pointer ; DPTR
  - program counter ; PC
  - carry flag ; C
  - accumulator-B register pair ; AB

```
INC     DPTR      ; opcode: 10100011 (1 cycle)
```

direct addressing

- access any on-chip variable or hardware register (2-byte instruction)

```
MOV     P1, A ; 1st byte    : 11110101 (opcode)
              ; 2nd byte    : 10010000 (address of P1)
              ; 1 cycle
```

indirect addressing

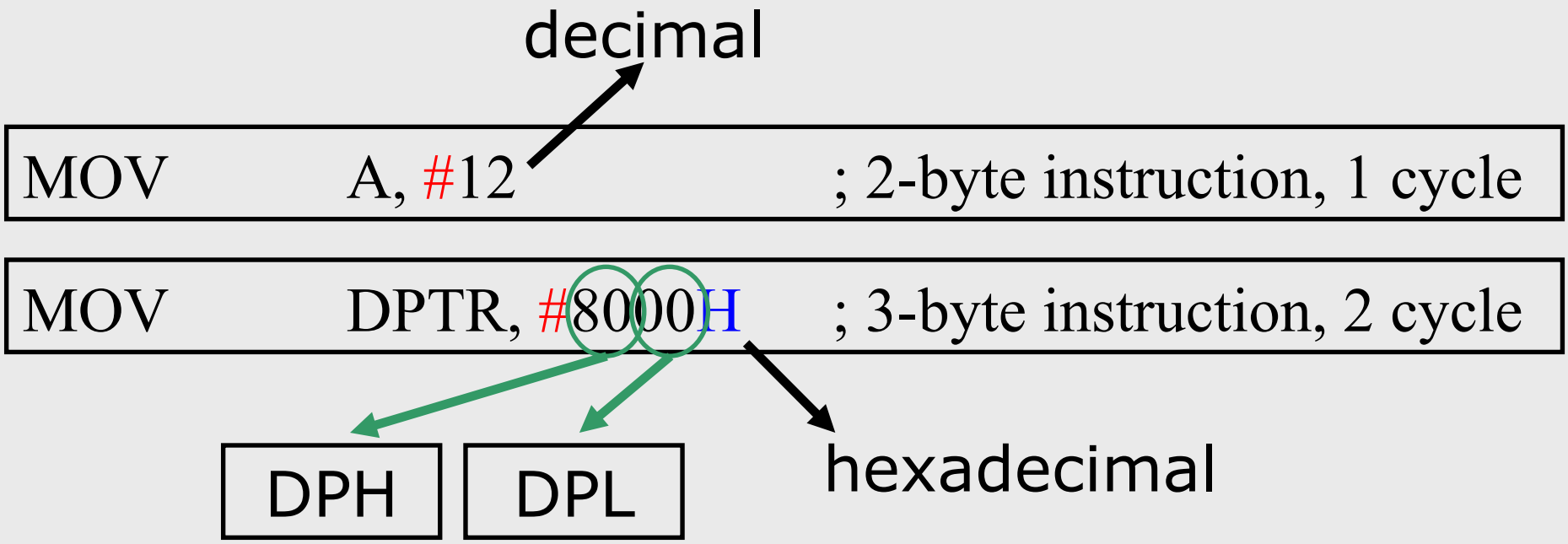
- LSB of instruction selects between R0 and R1 registers as pointers
  - manipulating sequential memory locations
  - indexed entries within tables in RAM
  - multiple precision numbers
  - character strings

```
MOV     A, @R1      ; opcode: 1110011 1 (1 cycle)
```


```
LOOP:   MOV     R0, #30H
        MOV     @R0, #0     ; 2-byte instr.
        INC     R0
        CJNE    R0, #80H, LOOP
```

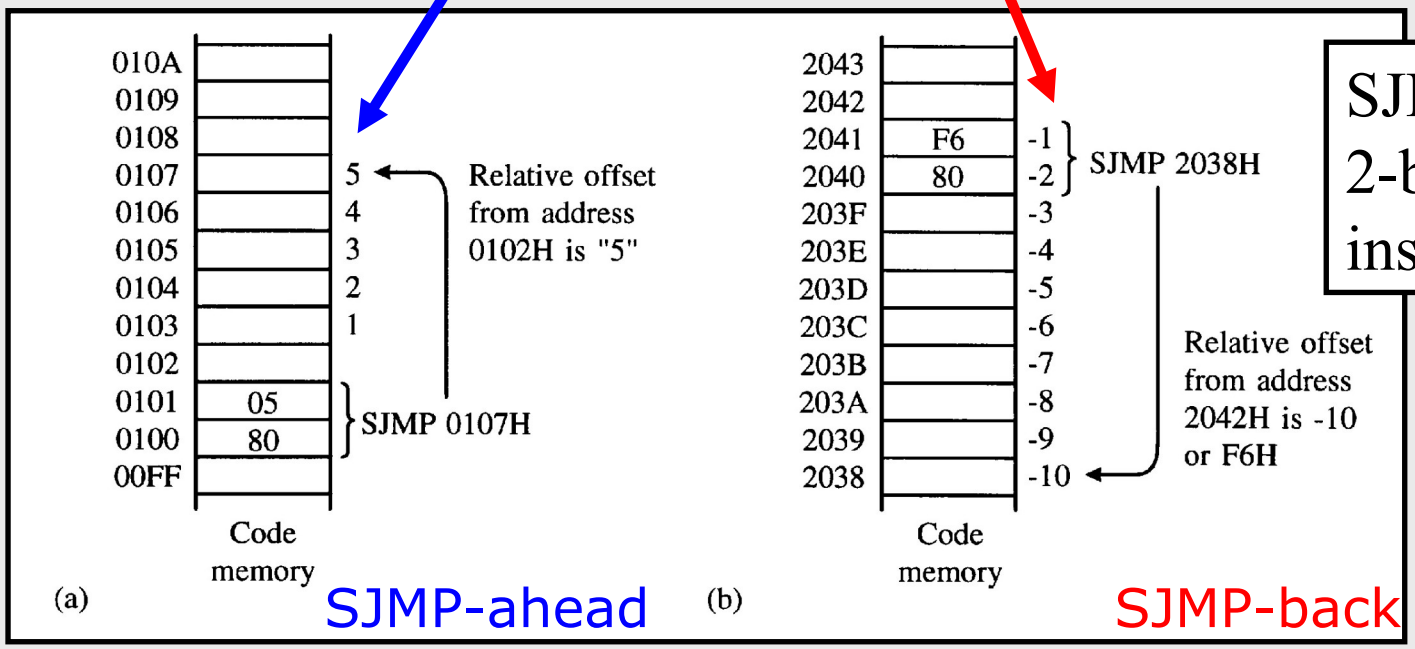
immediate addressing

- the source operand may be; a numeric constant, a symbolic variable, or an arithmetic expression using constants, symbols, and operators



relative addressing

- a relative address (offset) is an 8-bit signed value which is added to the PC to form the address of the next instruction
- relative addressing is used only with certain jump instructions
- jumping -128 to +127 locations

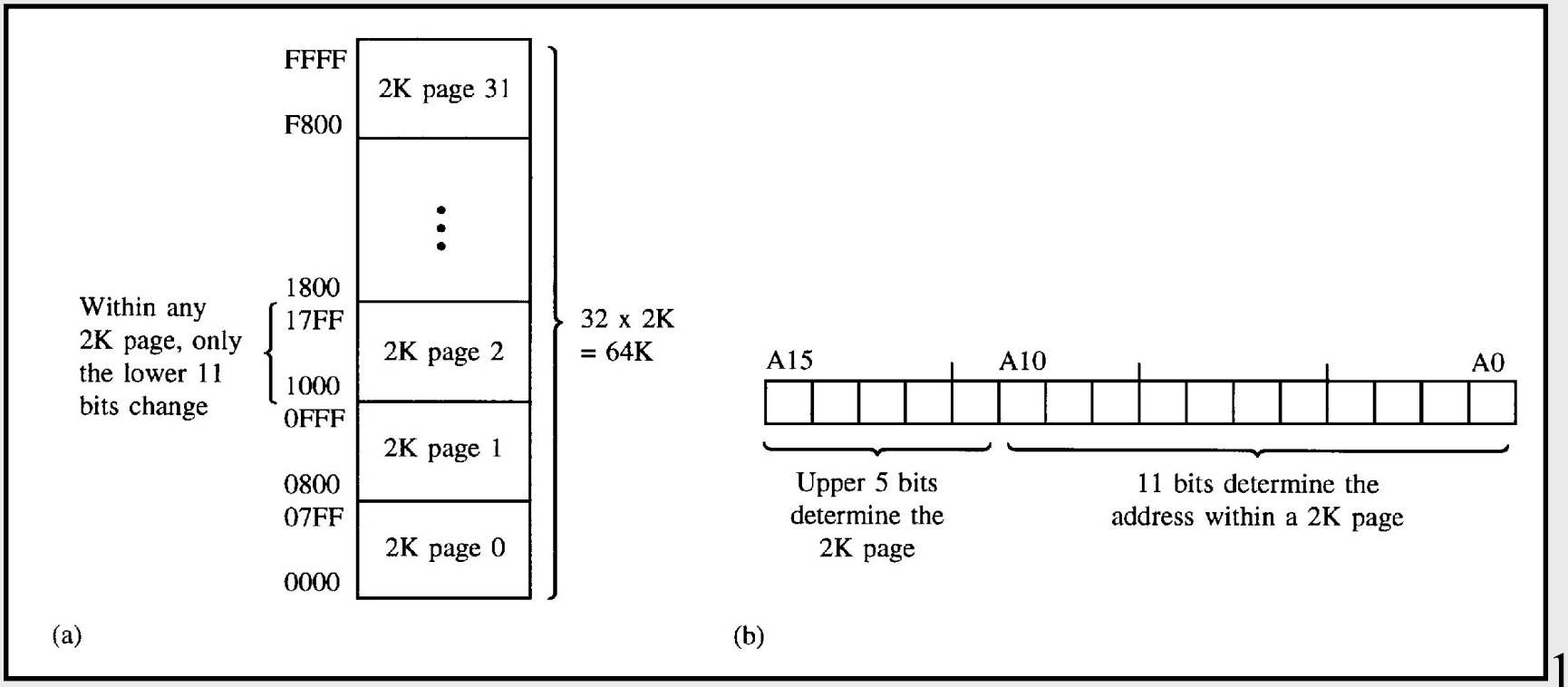


SJMP LOOP ;
2-byte, 2-cycle instruction

absolute addressing

- absolute addressing is used only with ACALL and AJMP instructions (2-byte, 2-cycles)

AJMP THERE	; 1st byte	: aaa00001 (A10-A8 + opcode)
	; 2nd byte	: aaaaaaaa (A7-A0)



long addressing

- long addressing is used only with LCALL and LJMP (*JMP*) instructions (3-bytes, 2-cycles)
- full 64K code space can be used
- instructions are position-dependent

```
JMP        2040H        ; always 2040H
```

indexed addressing

- indexed addressing uses a base register (PC or DPTR) and an offset (A) in forming the effective address for a JMP or MOVC instruction (1 byte, 2-cycles)
- jump tables

```
        MOV     DPTR, #JMP_TBL
        MOV     A, INDEX_NUMBER
        RL      A
        JMP     @A+DPTR ; A = {0,2,4,6}
JMP_TBL: AJMP   LABEL 0
        AJMP    LABEL 1
        AJMP    LABEL 2
        AJMP    LABEL 3
```

indexed addressing (cont'd)

- indexed addressing uses a base register (PC or DPTR) and an offset (A) in forming the effective address for a JMP or MOVC instruction (1 byte, 2-cycles)
- or look-up tables

```
MOV     A, ENTRY_NUMBER
CALL    REL_PC
```

subroutine

```
REL_PC: INC     A
        MOVC    A, @A+PC    ; A = {0,1,2,3}
        RET
        DB      66H
        DB      77H
        DB      88H
        DB      99H
```

instruction types

- 5 functional groups of instructions in 8051
  - arithmetic
  - logical
  - data transfer
  - boolean variable
  - program branching

instruction code summary

Instruction Code Summary

L \ H	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NOP	JBC bit, rel	JB bit, rel	JNB bit, rel	JC rel	JNC rel	JZ rel	JNZ rel	SJMP rel	MOV DPTR, # data 16	ORL C, /bit	ANL C, /bit	PUSH dir	POP dir	MOVX A, @DPTR	MOVX @DPTR, A
1	AJMP (P0)	ACALL (P0)	AJMP (P1)	ACALL (P1)	AJMP (P2)	ACALL (P2)	AJMP (P3)	ACALL (P3)	AJMP (P4)	ACALL (P4)	AJMP (P5)	ACALL (P5)	AJMP (P6)	ACALL (P6)	AJMP (P7)	ACALL (P7)
2	LJMP addr16	LCALL addr16	RET	RETI	ORL dir, A	ANL dir, A	XRL dir, A	ORL C, bit	ANL C, bit	MOV bit, C	MOV C, bit	CPL bit	CLR bit	SETB bit	MOVX A, @R0	MOVX @R0, A
3	RR A	RRC A	RL A	RLC A	ORL dir, # data	ANL dir, # data	XRL dir, # data	JMP @A+DPTR	MOVC A, @A+PC	MOVC A, @A+DPTR	INC DPTR	CPL C	CLR C	SETB C	MOVX A, @R1	MOVX @R1, A
4	INC A	DEC A	ADD A, # data	ADDC A, # data	ORL A, # data	ANL A, # data	XRL A, # data	MOV A, # data	DIV AB	SUBB A, # data	MUL AB	CJNE A, # data, rel	SWAP A	DA A	CLR A	CPL A
5	INC dir	DEC dir	ADD A, dir	ADDC A, dir	ORL A, dir	ANL A, dir	XRL A, dir	MOV dir, # data	MOV dir, dir	SUBB A, dir		CJNE A, dir, rel	XCH A, dir	DJNZ dir, rel	MOV A, dir	MOV dir, A
6	INC @R0	DEC @R0	ADD A, @R0	ADDC A, @R0	ORL A, @R0	ANL A, @R0	XRL A, @R0	MOV @R0, # data	MOV dir, @R0	SUBB A, @R0	MOV @R0, dir	CJNE @R0, # data, rel	XCH A, @R0	XCHD A, @R0	MOV A, @R0	MOV @ R0, A
7	INC @R1	DEC @R1	ADD A, @R1	ADDC A, @R1	ORL A, @R1	ANL A, @R1	XRL A, @R1	MOV @R1, # data	MOV dir, @R1	SUBB A, @R1	MOV @R1, dir	CJNE @R1, # data, rel	XCH A, @R1	XCHD A, @R1	MOV A, @R1	MOV @ R1, A
8	INC R0	DEC R0	ADD A, R0	ADDC A, R0	ORL A, R0	ANL A, R0	XRL A, R0	MOV R0, # data	MOV dir, R0	SUBB A, R0	MOV R0, dir	CJNE R0, # data, rel	XCH A, R0	DJNZ R0, rel	MOV A, R0	MOV R0, A
9	INC R1	DEC R1	ADD A, R1	ADDC A, R1	ORL A, R1	ANL A, R1	XRL A, R1	MOV R1, # data	MOV dir, R1	SUBB A, R1	MOV R1, dir	CJNE R1, # data, rel	XCH A, R1	DJNZ R1, rel	MOV A, R1	MOV R1, A
A	INC R2	DEC R2	ADD A, R2	ADDC A, R2	ORL A, R2	ANL A, R2	XRL A, R2	MOV R2, # data	MOV dir, R2	SUBB A, R2	MOV R2, dir	CJNE R2, # data, rel	XCH A, R2	DJNZ R2, rel	MOV A, R2	MOV R2, A
B	INC R3	DEC R3	ADD A, R3	ADDC A, R3	ORL A, R3	ANL A, R3	XRL A, R3	MOV R3, # data	MOV dir, R3	SUBB A, R3	MOV R3, dir	CJNE R3, # data, rel	XCH A, R3	DJNZ R3, rel	MOV A, R3	MOV R3, A
C	INC R4	DEC R4	ADD A, R4	ADDC A, R4	ORL A, R4	ANL A, R4	XRL A, R4	MOV R4, # data	MOV dir, R4	SUBB A, R4	MOV R4, dir	CJNE R4, # data, rel	XCH A, R4	DJNZ R4, rel	MOV A, R4	MOV R4, A
D	INC R5	DEC R5	ADD A, R5	ADDC A, R5	ORL A, R5	ANL A, R5	XRL A, R5	MOV R5, # data	MOV dir, R5	SUBB A, R5	MOV R5, dir	CJNE R5, # data, rel	XCH A, R5	DJNZ R5, rel	MOV A, R5	MOV R5, A
E	INC R6	DEC R6	ADD A, R6	ADDC A, R6	ORL A, R6	ANL A, R6	XRL A, R6	MOV R6, # data	MOV dir, R6	SUBB A, R6	MOV R6, dir	CJNE R6, # data, rel	XCH A, R6	DJNZ R6, rel	MOV A, R6	MOV R6, A
F	INC R7	DEC R7	ADD A, R7	ADDC A, R7	ORL A, R7	ANL A, R7	XRL A, R7	MOV R7, # data	MOV dir, R7	SUBB A, R7	MOV R7, dir	CJNE R7, # data, rel	XCH A, R7	DJNZ R7, rel	MOV A, R7	MOV R7, A

2Byte	3Byte
2Cycle	4Cycle

arithmetic instructions

```
ADD     A, 7FH      ; direct addressing
ADD     A, @R0      ; indirect addressing
ADD     A, R7       ; register addressing
ADD     A, #35H     ; immediate addressing
```

```
INC   7FH
```

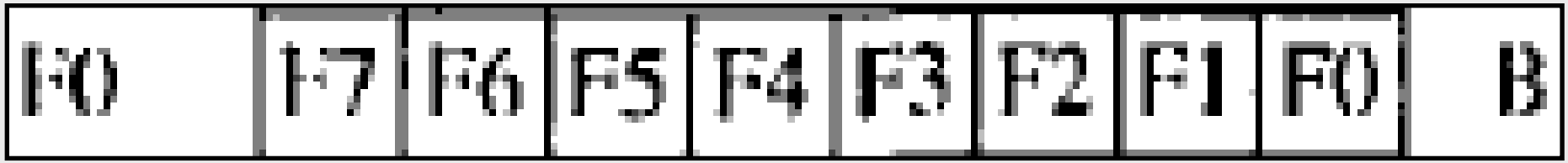
```
INC   DPTR
```

```
        DEC     DPL     ; DPTR--
        MOV     R7, DPL
        CJNE    R7, #0FFH, SKIP
        DEC     DPH
SKIP:   (continue)
```

BCD arithmetic

Arithmetic Operations			
ADD	A,source		add source to A
ADD	A,#data		
ADDC	A,source		add with carry
ADDC	A,#data		
SUBB	A,source		subtract from A
SUBB	A,#data		with borrow
INC	A		increment
INC	source		
DEC	A		decrement
DEC	source		
INC	DPTR	2	increment DPTR
MUL	AB	4	multiply A & B
DIV	AB		divide A by B
DA	A		decimal adjust A

arithmetic instructions (cont'd)




```
MUL     AB      ; A(8-bit) X B(8-bit) = 16-bit result (high-byte, low-byte)
```

```
DIV     AB      ; A(8-bit) / B(8-bit) = 8-bit integer result, 8-bit remainder
```

logical instructions

- all logical instructions using A as one of the operands execute in 1 cycle (others in 2 cycles)
- direct, indirect, register, and immediate addressing variations
- RL and RR are 8-bit rotate; RLC and RRC are 9-bit rotate operations

Logical Operations		
ANL	A,source	logical AND
ANL	A,#data	
ANL	direct,A	
ANL	direct,#data	
ORL	A,source	logical OR
ORL	A,#data	
ORL	direct,A	
ORL	direct,#data	
XRL	A,source	logical XOR
XRL	A,#data	
XRL	direct,A	
XRL	direct,#data	
CLR	A	clear A
CPL	A	complement A
RL	A	rotate A left
RLC	A	(through C)
RR	A	rotate A right
RRC	A	(through C)
SWAP	A	swap nibbles

```
XRL   P1, #0FFH      ; invert bits
```

BCD conversion

(A, known to be less than 100 in binary form)

```
MOV    B, #10
 DIV   AB
SWAP   A
 ADD   A, B
```

data transfer instructions

Data Transfer Operations		
MOV	A,source	move source
MOV	A,#data	to destination
MOV	dest,A	
MOV	dest,source	
MOV	dest,#data	
MOV	DPTR,#data16	
MOVC	A,@A+DPTR	move from code
MOVC	A,@A+PC	memory
MOVX	A,@Ri	move from data
MOVX	A,@DPTR	memory
MOVX	@Ri,A	
MOVX	@DPTR,A	
PUSH	direct	push onto stack
POP	direct	pop from stack
XCH	A,source	exchange bytes
XCHD	A,@Ri	exchange low order digits

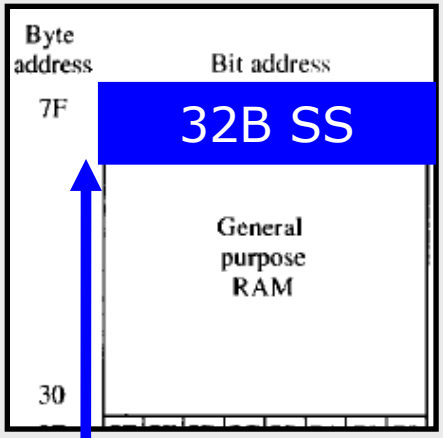
internal RAM DT instructions

- the instructions that move data within the internal memory spaces (execute in either 1 or 2 cycles)
- MOV <destination>, <source> format allows data to be transferred between two internal RAM locations without going through the accumulator
- stack resides in on-chip RAM and grows upward in memory

MOV	DPTR, #8000H	
XCH	A, <source>	; exchange data
XCHD	A, @R<i>	; *i = {0,1}* exchange low-order nibbles

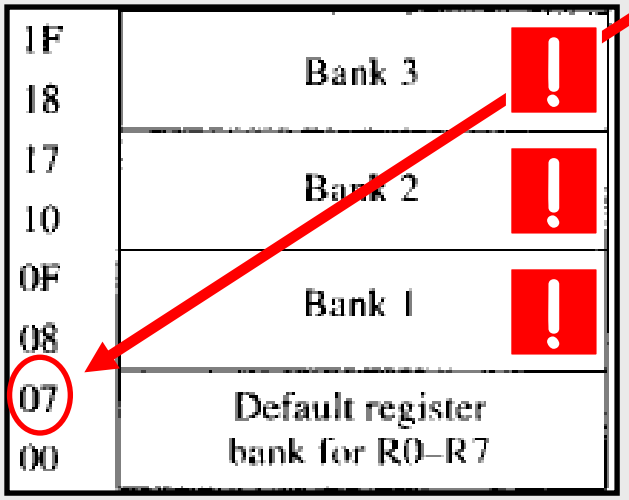
internal RAM DT instructions (cont'd)

81	not bit addressable	SP



- push : first SP+1, then write data
- pop : first read data, then SP-1
- default by system reset : 07H

```
MOV     SP, #5FH        ; beginning address of stack = 60H
```



be **very careful** of what you are doing if you do not initialize your own stack space but still use context switching, and PUSH, POP, ACALL, LCALL, RET, RETI instructions on stack space (SS)

external RAM DT instructions

- indirect addressing
- @Ri (1-byte address) or @DPTR (2-byte address)
- 16-bit address choice uses all the 8-bits of Port 2 as the high-byte of the address bus
- 8-bit address choice allows access to fewer RAM, but does not sacrifice Port 2
- all external RAM data transfer instructions execute in 2 machine cycles and use the accumulator either as source or destination operand
- read/write strobes to external RAM are activated only during a MOVX instruction (normally they are high) (they could be made available as I/O lines if no external memory is used)

look-up tables DT instructions

- the look-up tables can only be read, not updated
- MOVC (MOVe Constant) uses either PC or DPTR as the base register, and accumulator as offset

```
MOVC        A, @A+DPTR    ; can accomodate up to a table of 255 entries
```

```
        MOV     A, <entry number>
        CALL    LOOK_UP
```

```
LOOK_UP: INC    A
         MOVC   A, @A+PC
         RET
TABLE:   DB     <data0>
         DB     <data1>
         DB     <data2>
         DB     <data3>
```

boolean instructions

- complete Boolean processor for single-bit operations
- bit-addressable port lines, complete repertoire of bit-instructions

```
        SETB    P1.7
        CLR     P1.7
bit-bit MOV     C, FLAG
        MOV     P1.0, C
        CLR     C
        MOV     C, BIT1
XOR     JNB     BIT2, SKIP
        CPL     C
SKIP:   (continue)
```

bit-testing

Boolean Variable Manipulation

CLR	C	clear bit
CLR	bit	
SETB	C	set bit
SETB	bit	
CPL	C	complement bit
CPL	bit	
ANL	C,bit	AND bit with C
ANL	C,/bit	AND NOT bit with C
ORL	C,bit	OR bit with C
ORL	C,/bit	OR NOT bit with C
MOV	C,bit	move bit to bit
MOV	bit,C	
JC	rel	jump if C set
JNC	rel	if C not set
JB	bit,rel	jump if bit set
JNB	bit,rel	if bit not set
JBC	bit,rel	if set then clear

program branching instructions

- SJMP (relative offset, -128 to +127)
- AJMP (11-bit constant, 2K code memory blocks)
- LJMP (16-bit destination address)
- "destination out of range" error if instruction does not support destination address

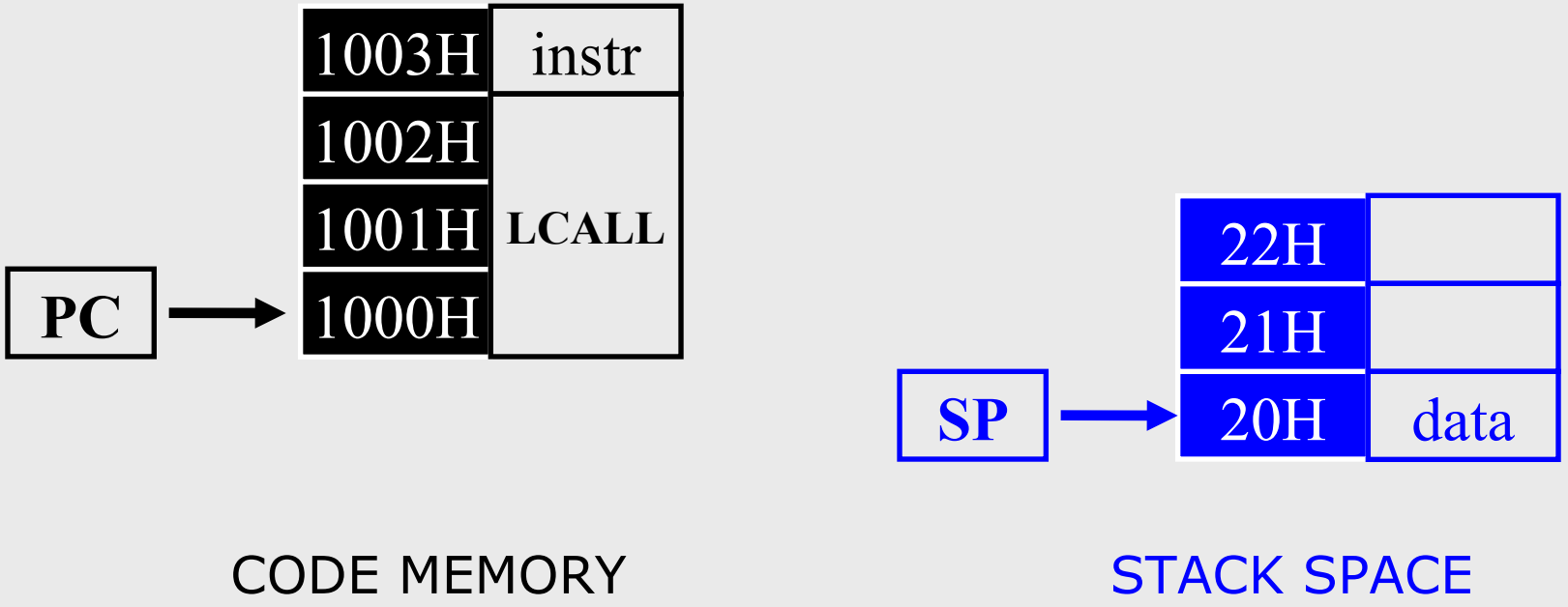
Program Branching		
ACALL	addr11	call subroutine
LCALL	addr16	
RET		return from sub.
RETI		from interrupt
AJMP	addr11	jump
LJMP	addr16	
SJMP	rel	
JMP	@A+DPTR	
JZ	rel	jump if A = 0
JNZ	rel	if A not = 0
CJNE	A,direct,rel	compare and jump
CJNE	A,#data,rel	if not equal
CJNE	Rn,#data,rel	
CJNE	@Ri,#data,rel	
DJNZ	Rn,rel	decrement and jump
DJNZ	direct,rel	if not zero
NOP		no operation

jump tables

```
                MOV     DPTR, #JUMP_TABLE
                MOV     A, INDEX_NUMBER
                RL      A       ; index even (2-byte address in table)
                JMP     @A+DPTR
JUMP_TABLE:     AJMP    CASE0   ; index = 0 -> RL -> 0
                AJMP    CASE1   ; index = 1 -> RL -> 2
                AJMP    CASE2   ; index = 2 -> RL -> 4
                AJMP    CASE3   ; index = 3 -> RL -> 6
```

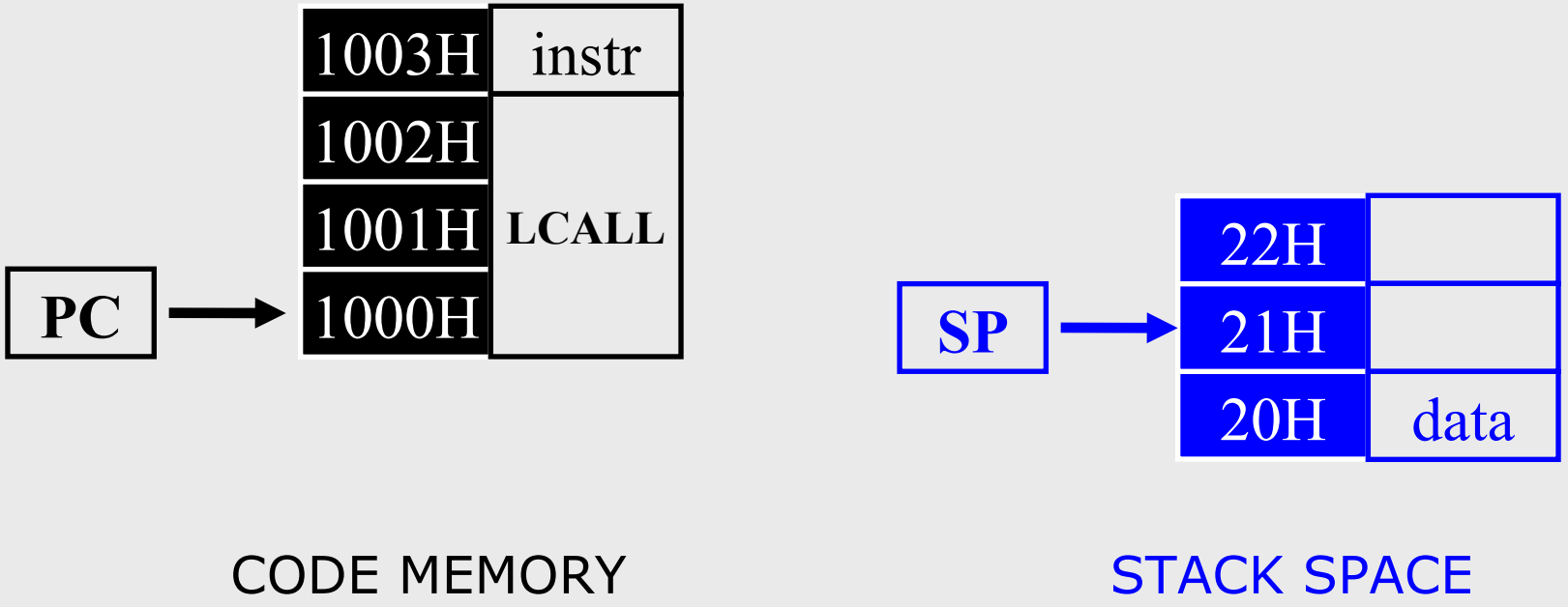
subroutines and interrupts

- ACALL (absolute addressing), LCALL (long addressing)



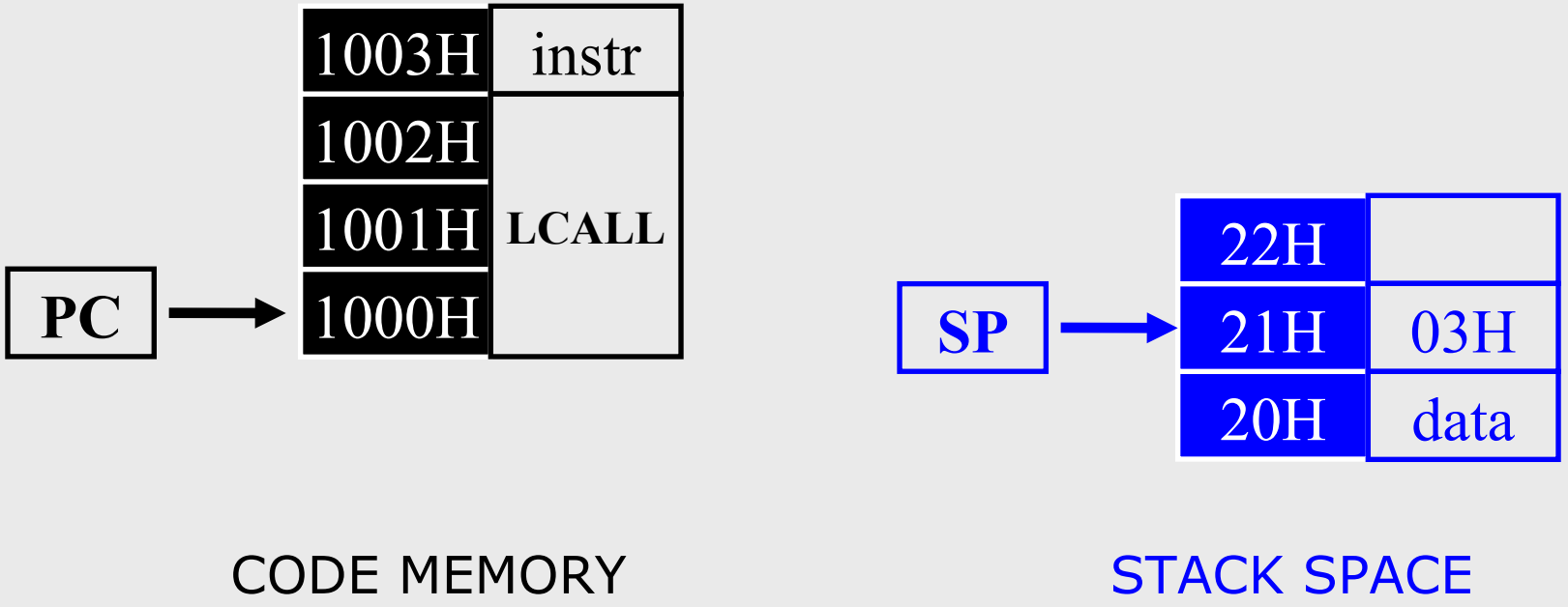
subroutines and interrupts

- ACALL (absolute addressing), LCALL (long addressing)



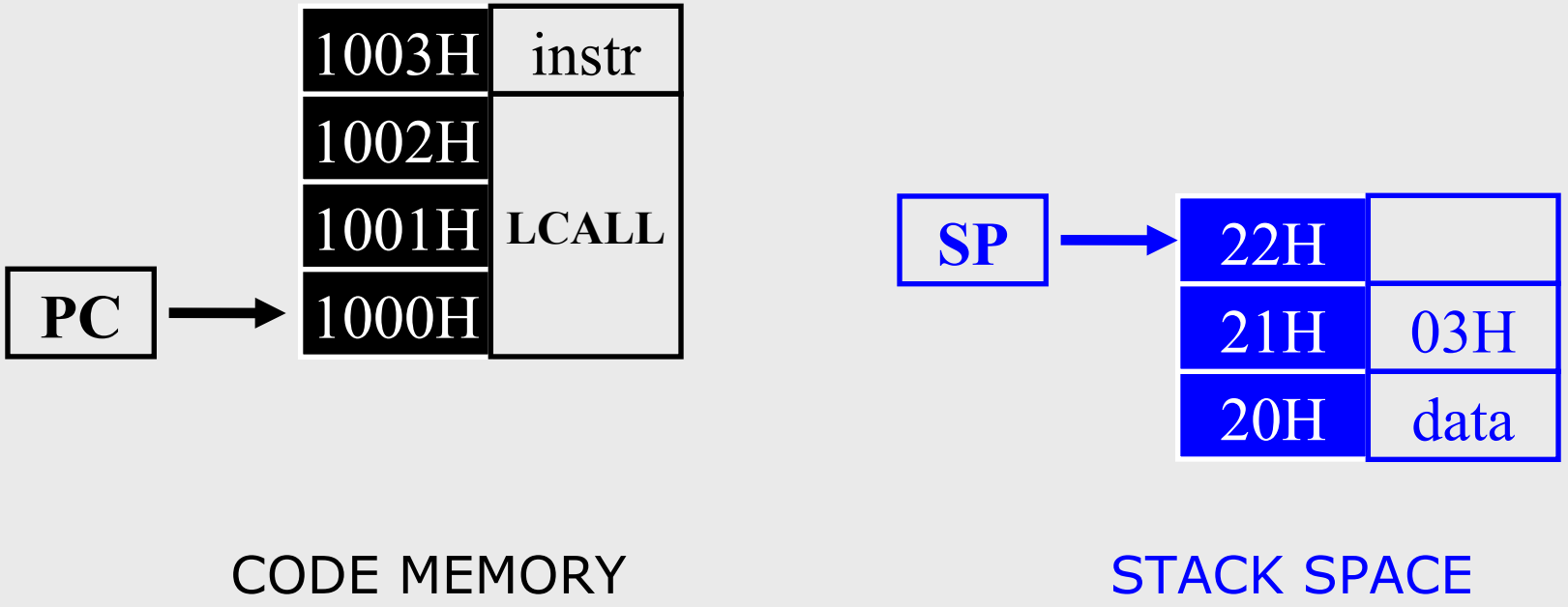
subroutines and interrupts

- ACALL (absolute addressing), LCALL (long addressing)



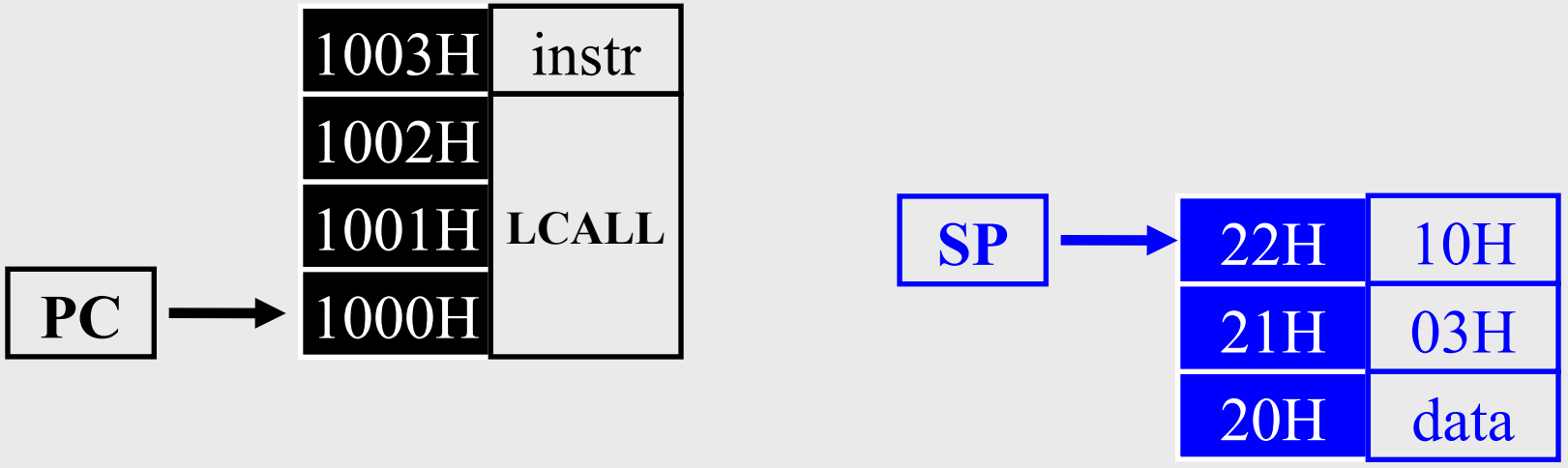
subroutines and interrupts

- ACALL (absolute addressing), LCALL (long addressing)



subroutines and interrupts

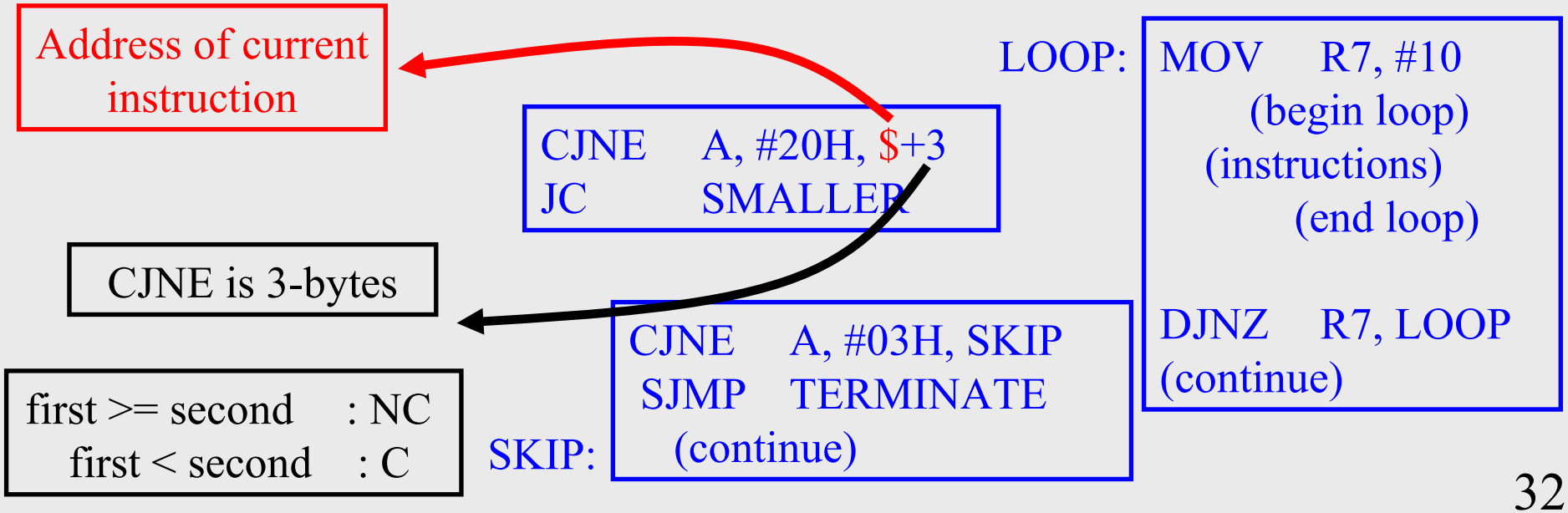
- ACALL (absolute addressing), LCALL (long addressing)



- subroutines should end with a RETurn instruction
- RETI RETurns from Interrupt Service Routine
- RETI == RET if no other pending interrupts **(but!)**

conditional jumps

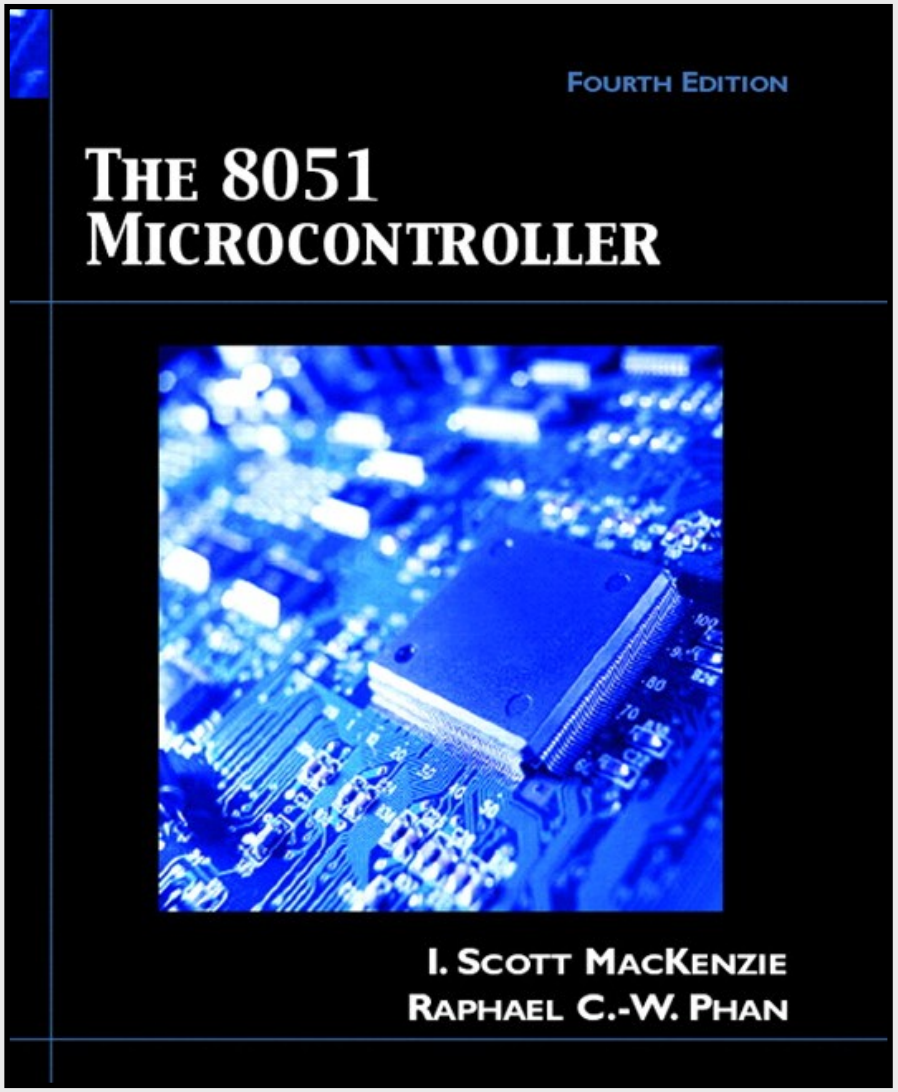
- conditional jumps use relative addressing (distance : -127 to +128)
- no 0-bit in PSW, JZ and JNZ tests A
- DJNZ (Decrement & Jump if Not Zero)
- CJNE (Compare & Jump if Not Equal)



summary

- 8051 instruction set overview
- addressing modes
- instruction types

references



Google™

WIKIPEDIA
The Free Encyclopedia