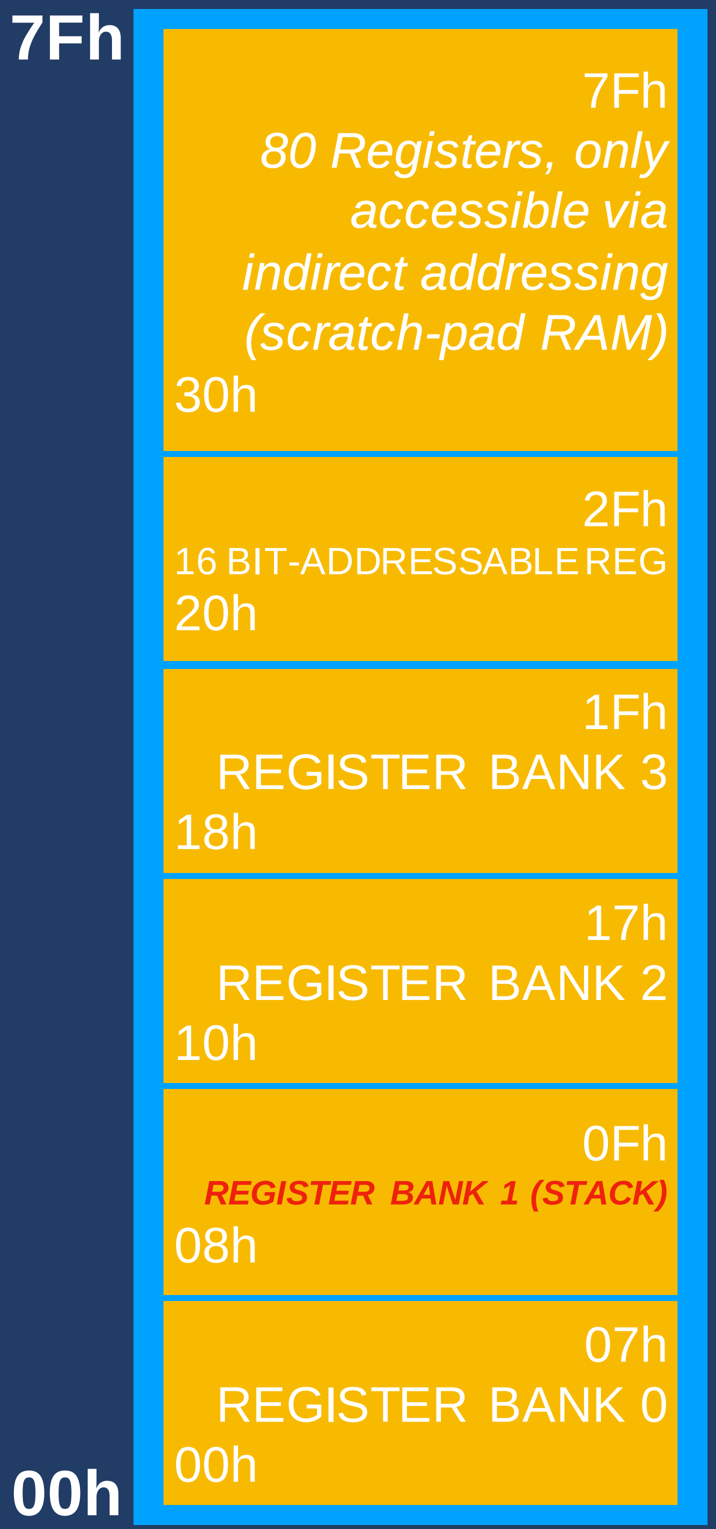
BIT-ADDRESSABLE REGISTERS



LOWER 128 BYTES, NOT TO SCALE

- The 8051 has 16 bit-addressable registers.
  - Bit-addressable: each bit within each register may be independently manipulated.
    - [_ _ _ _ _ _ _ _] < each of those bits may be toggled, allowing fine-grained control.
  - In a byte-addressable register, only the whole byte can be manipulated.
    - If you wish to manipulate a byte-addressable register bit-by-bit, you typically must move it to a bit addressable register, edit it, and then move it back.
  - R0-R7 in each of the register banks are byte-addressable, as is the region from 0x30 to 0x7F
    - Region from 0x20 to 0x2F is bit addressable.
      - Some of the SFR's (special function registers) are also bit-addressable. This allows us to edit settings (e.g., PSW) bit-by-bit using the SETB instruction and the CLR instruction.

Bit Addressing

- Bit Addressing with symbols is by dot notation

```
CLR 97H        ;Clear bit 7 of port 1 in SFR (bit addressing)
CLR P1.7       ;Same as the above (symbol with dot notation)
```

  - The assembler performs translation and the machine code contains the appropriate operand value.

  - Generally, any mnemonic listed in the datasheet can be used as a symbol in an assembly language program.

Example

- Find the contents of the destination operand after execution of each of the following instructions.

```
MOV R5, #10H          ; R5 = 10H
INC R5                ; R5 = 11H
INC R5                ; R5 = 12H
MOV R0, #20H          ; R0 = 20H
MOV A, #0FFH          ; A = FFH
MOV 20H, A            ; (20H) = FFH
MOV @ R0, #10H        ; (20H) = 10H
INC A                 ; A = 00H
MOV 20H, #00H         ; (20H) = 00H
INC 20H               ; (20H) = 01H
```

Example

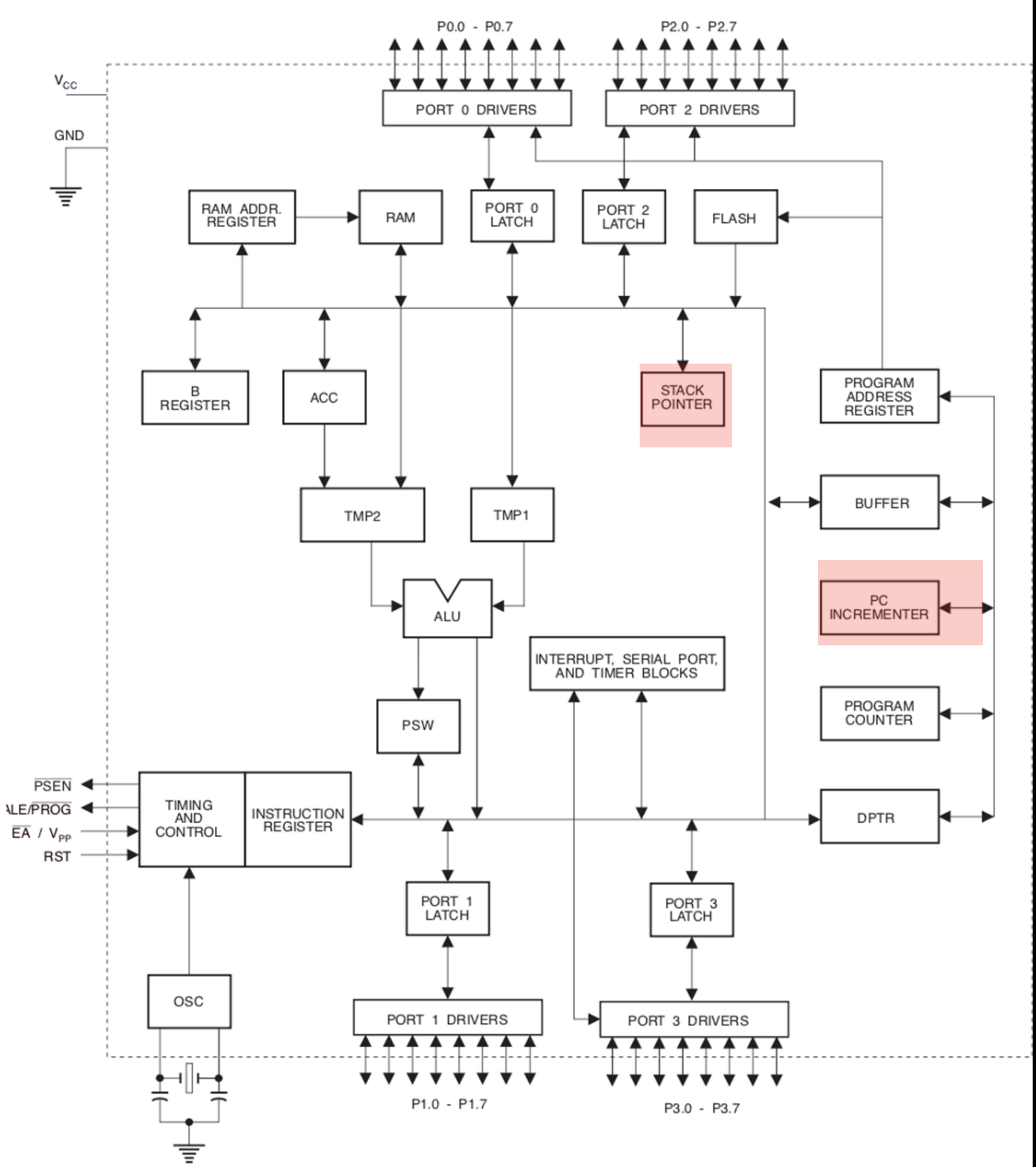
- Write a program to find the square of a number stored at the internal RAM address 50H. Store the result at address 60H (LSByte) and 61H (MSByte). If the number is AAH, what will be the result and status of the OV flag after finding the square of that number?

```
MOV A, 50H ; copy the number at address 50H into A
MOV B, A   ; copy the same number into B
MUL AB     ; find the square by multiplication
MOV 60H, A ; copy the result (LSByte) into address 60H
MOV 61H, B ; copy the result (MSByte) into address 61H
```

- If the number is AA, then result will be 70E4H. Since result is greater than FFH the overflow flag will be set, i.e. OV=1 after multiplication.

Today

Block Diagram



- Memory Usage:
- Stack

- Most important registers today are the Stack Pointer and the Program Counter

On-Chip Data Memory Organisation

Bit Addressable

Byte Address	Bit Address							
7F	General Purpose RAM							
30								
2F	7F	7E	7D	7C	7B	7A	79	78
2E	77	76	75	74	73	72	71	70
2D	6F	6E	6D	6C	6B	6A	69	68
2C	67	66	65	64	63	62	61	60
2B	5F	5E	5D	5C	5B	5A	59	58
2A	57	56	55	54	53	52	51	50
29	4F	4E	4D	4C	4B	4A	49	48
28	47	46	45	44	43	42	41	40
27	3F	3E	3D	3C	3B	3A	39	38
26	37	36	35	34	33	32	31	30
25	2F	2E	2D	2C	2B	2A	29	28
24	27	26	25	24	23	22	21	20
23	1F	1E	1D	1C	1B	1A	19	18
22	17	16	15	14	13	12	11	10
21	0F	0E	0D	0C	0B	0A	09	08
20	07	06	05	04	03	02	01	00
1F	Bank 3							
18								
17	Bank 2							
10								
0F	Bank 1							
08								
07	Default Register Bank for R0 – R7							
00								

Byte Address	Bit Address								
FF									
F0	F7	F6	F5	F4	F3	F2	F1	F0	B
E0	E7	E6	E5	E4	E3	E2	E1	E0	ACC
D0	D7	D6	D5	D4	D3	D2	-	D0	PSW
B8	-	-	-	BC	BB	BA	B9	B8	IP
B0	B7	B6	B5	B4	B3	B2	B1	B0	P3
A8	AF	-	-	AC	AB	AA	A9	A8	IE
A0	A7	A6	A5	A4	A3	A2	A1	A0	P2
99	Not bit-addressable								SBUF
98	9F	9E	9D	9C	9B	9A	99	98	SCON
90	97	96	95	94	93	92	91	90	P1
8D	Not bit-addressable								TH1
8C	Not bit-addressable								TH0
8B	Not bit-addressable								TL1
8A	Not bit-addressable								TL0
89	Not bit-addressable								TMOD
88	8F	8E	8D	8C	8B	8A	89	88	TCON
87	Not bit-addressable								PCON
83	Not bit-addressable								DPH
82	Not bit-addressable								DPL
81	Not bit-addressable								SP
80	87	86	85	84	83	82	81	80	P0

Most 8051 internal registers are mapped to on-chip RAM and, therefore, have an address:

- Stack Pointer
- Data Pointer
- PSW, TMOD, etc.
- ACC, B, R0-R7 etc.

Exceptions:

- Program Counter
- Instruction Register

Reason: little point in addressing or manipulating these registers directly

STACKs

- Important point about stacks:
  - Stack Pointer (SP) is a register memory-mapped to location 81H.
  - Pushing to the stack increments the SP *before* writing data;
    Popping from the stack reads data and then *decrements* the SP
  - The 8051 stack is kept in internal RAM and is limited to addresses accessible by indirect addressing (The first 128/256 bytes)
  - It is possible to relocate the stack by changing the value of the Stack Pointer.
  - The System uses the stack to manage program flow, both user (CALL) and interrupts

The system stack

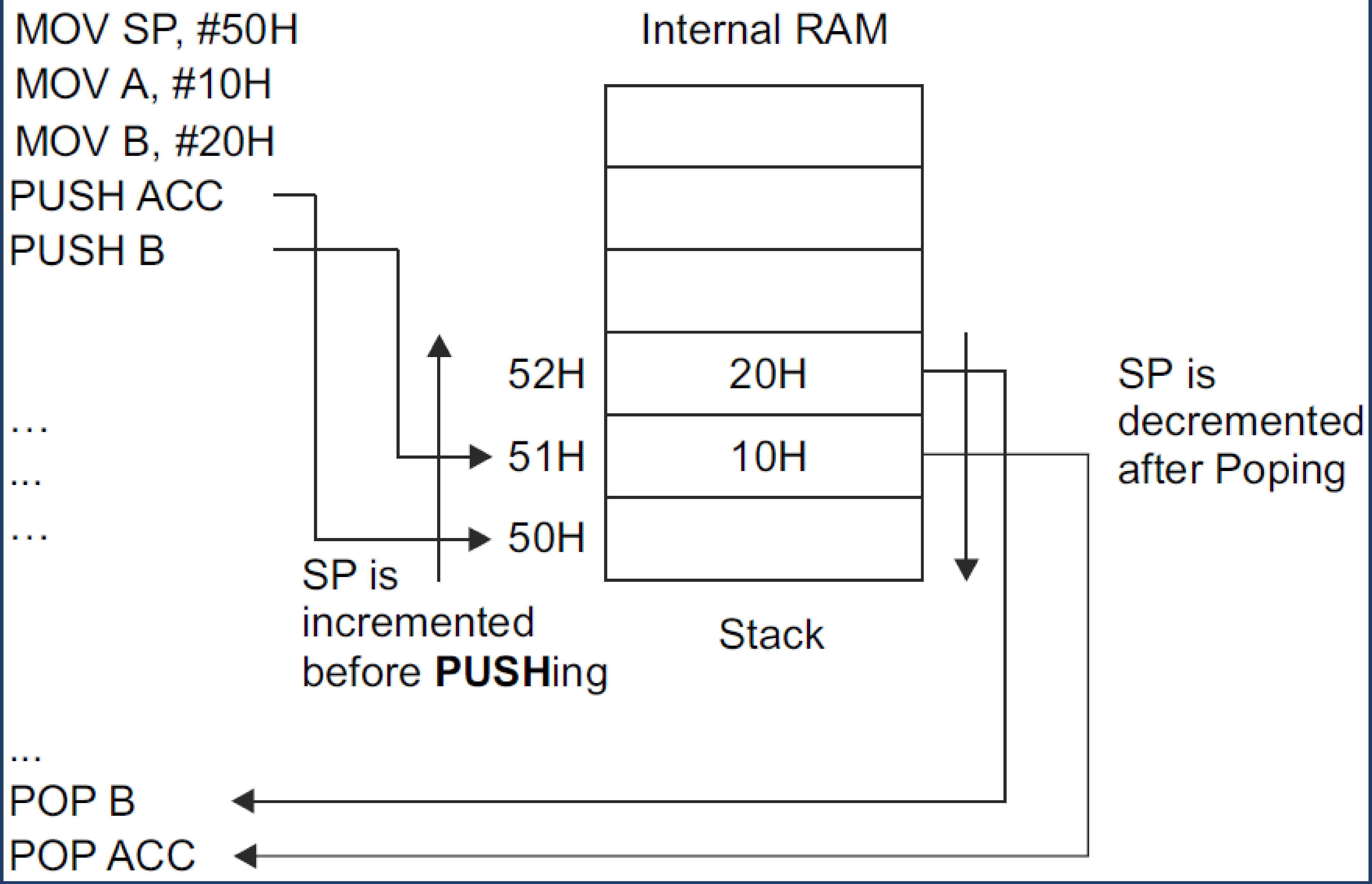
- **When calling a subroutine or serving an interrupt, it is necessary to preserve the return address**
- **Also, often, we need to preserve the contents of other registers.**
- **Then, the stack is a special area of data memory for temporary storage. It is a LIFO (Last In, First Out) Structure.**
- **A special Stack Pointer (SP) register is used to store the address of the top of the stack.**
- **The reset value of the SP is 07H, which is just after the first register bank**

The system stack

- **The maximum available memory for the 8051 stack is 128 bytes.**
- **This is not a lot, so we need to be careful that we do not run out of memory and cause an overflow**
- **Therefore, we must avoid recursive-type programs**
- **PUSH and POP are special instructions associated with the stack**

The system stack

- **Explain how the contents of the Accumulator and B registers can stored and retrieved form the stack**



The system stack

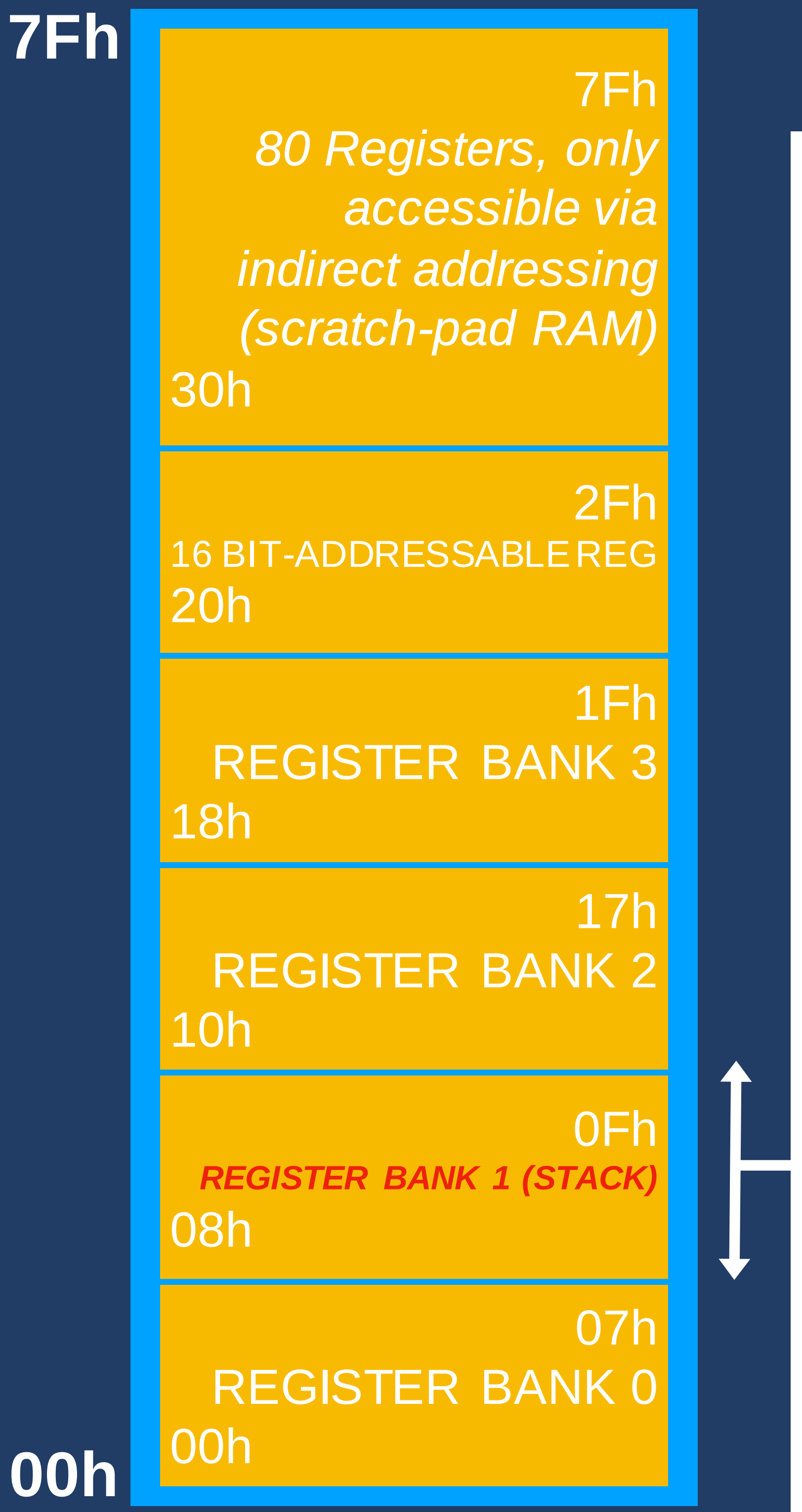
- **With a call to a subroutine, for example, "ACALL." The operation will cause the PC to increase by 2. Then, it pushes the 16-bit PC value onto the stack (low-order bytes first) and increments the stack pointer twice.**

- **At the end of the subroutine, the RET instruction pops the high byte and low byte address of the PC from the stack and decrements the SP by 2. The execution of the instruction will result in the program resuming from the location just after the "CALL" instructions**

- **A similar procedure occurs with interrupt**

- **The user can also use the stack for temporary storage, which is often a way to pass variables to and from subroutines.**

STACK



LOWER 128 BYTES, NOT TO SCALE

- The 8051 features a stack. Common to many computers.
  - Stack: memory region to which data may be 'pushed' and from which data may be 'popped'
    - Push: data is added to the stack.
    - Pop (AKA 'Pull'): the most recently added data is removed from the stack.
  - A stack is a "LIFO" structure: Last In, First Out
    - Last data in becomes the first data removed
  - The computer must keep track of the top of the stack: as the stack grows, this memory address will also grow.
    - This address is stored by the stack pointer.
      - As the stack grows, the stack pointer holds the address of the most recently added item.
- Stacks are used to temporarily store memory addresses while the computer does something else.
  - E.g., memory addresses before jumps to subroutines may be stored on the stack.
- Many CISC computers have a hardware stack; most RISC machines have stacks implemented in software.

8051 STACK

- The 8051's CPU features an 8-bit stack pointer
  - By default, the stack pointer points to address 0x07
  - The stack pointer increments by 1 (counting up).
    - This is the address immediately below the stack.
      - Register Bank 1, R0 (Address 0x08) is therefore the default start of the stack.
- Note that Register Bank 1 and the stack share the same space.
  - If we need to use Bank 1, we can relocate the stack.
    - **`MOV SP, #2FH ;Load 0x2F to stack pointer.   ; ;Stack now starts at 0x30.`**
  - Some instructions that use the stack:
    - PUSH
      - `PUSH addr` ; 2 cycles, increments stack pointer (SP) by 1 and then moves addr to the address within SP.
    - POP
      - `POP addr` ; 2 cycles. First, addr is loaded with value pointed to by SP. SP is then decremented by 1.
    - ACALL (discussed in prev. slides): address of line following the ACALL is stored to the stack (2 bytes).
      - RET: return from subroutine. Two byte address stored in ACALL is popped from the top of the stack (MSB then LSB) into PC.
- For more details on the registers and the stack, see https://what-when-how.com/8051-microcontroller/8051-register-banks-and-stack/