XMUT202 Digital Electronics

Instruction Sets

Week 10 Lecture 2

School of Engineering and Computer Science
Victoria University of Wellington

Today's topic

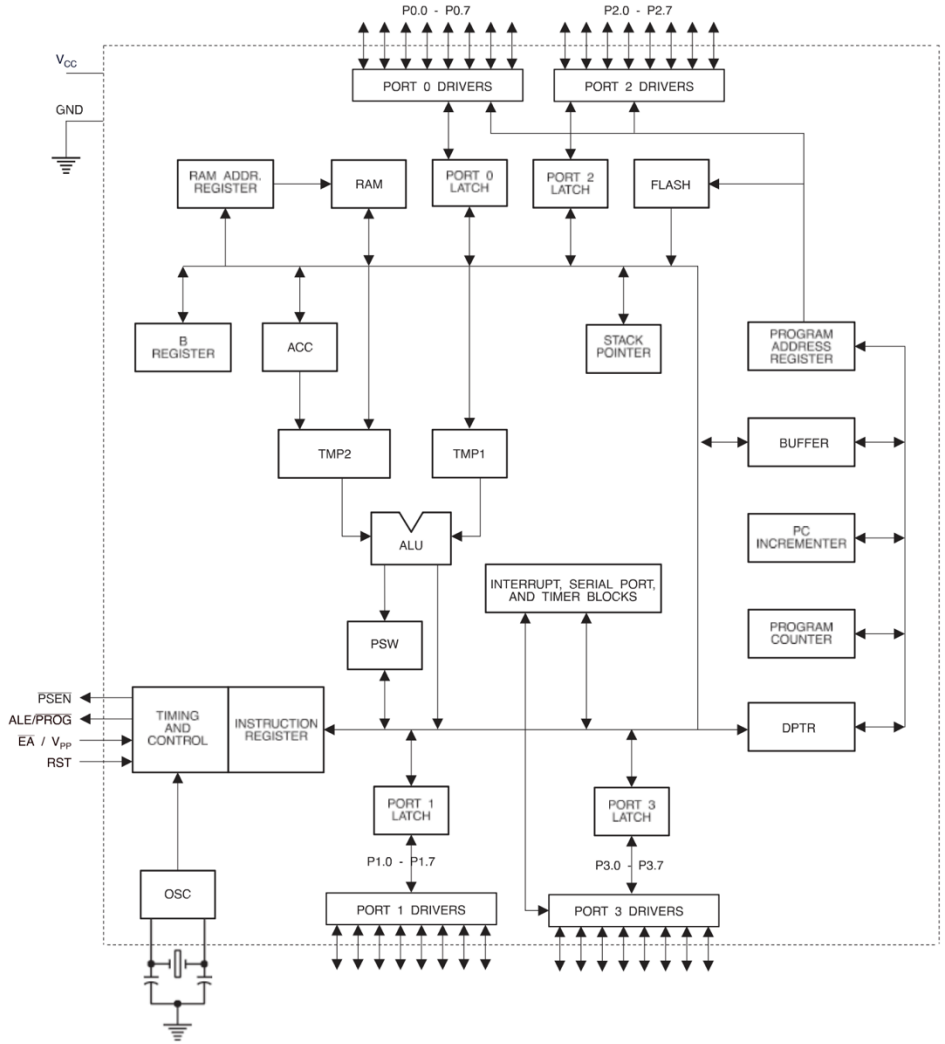
- (Review) 8051 architecture and memory organisation.
- (Review) Data memory and mapping.
- Special function registers.
- Types of 8051 standard instruction sets:
- Arithmetic operations.
- Logical operations.
- Data transfer operations.
- Boolean variable operations.
- Program branching operations.

Intro to Instruction Sets

- A computer instruction is made up of an operation code (op- code) followed by either zero, one or two bytes of operands.
- The op-code identifies the type of operation to be performed while the operands identify the source and destination of the data.
- The operand can be:
  - The data value itself.
  - A CPU register.
  - A memory location.
  - An I/O port.
- If the instruction is associated with more than one operand, the format is always:

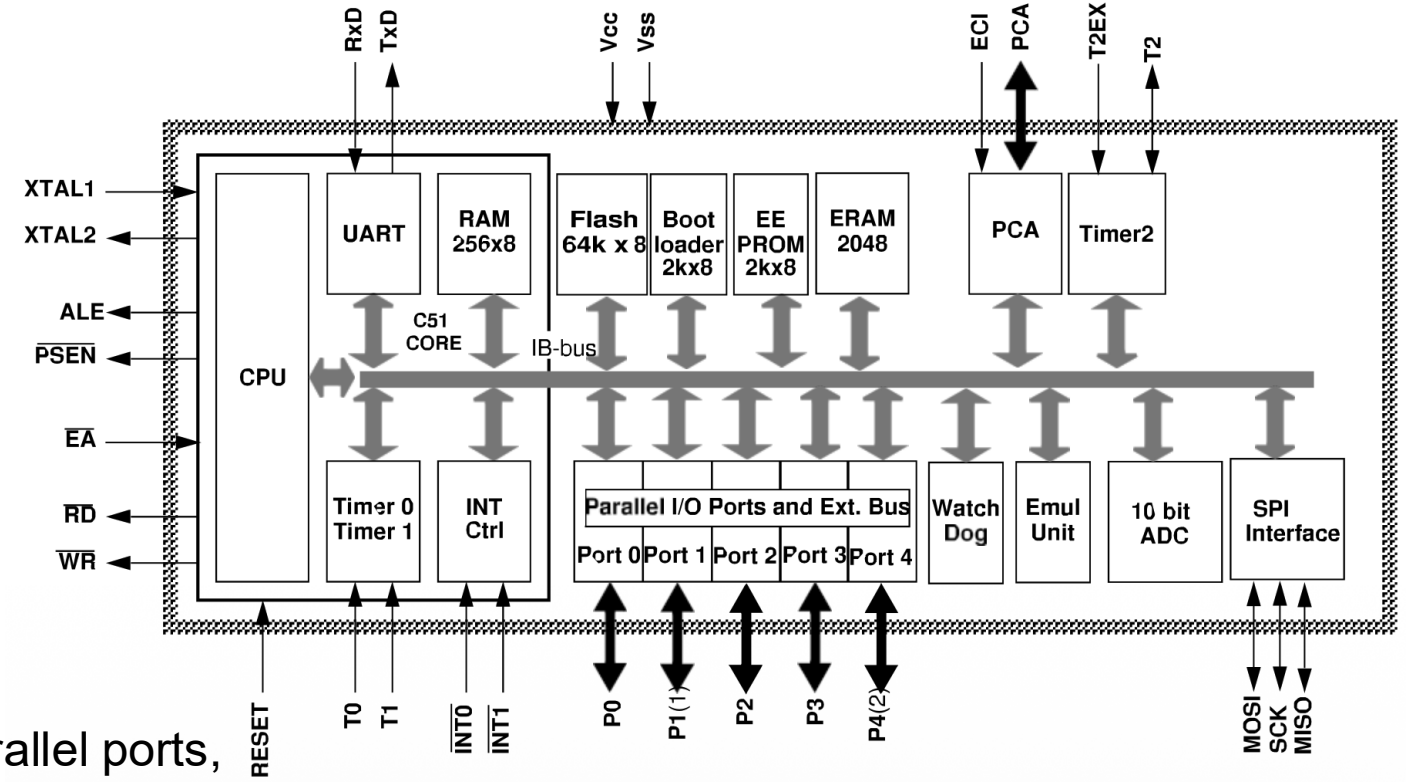
```
Instruction Destination, Source
```

8051 Architecture Review



Storage systems:

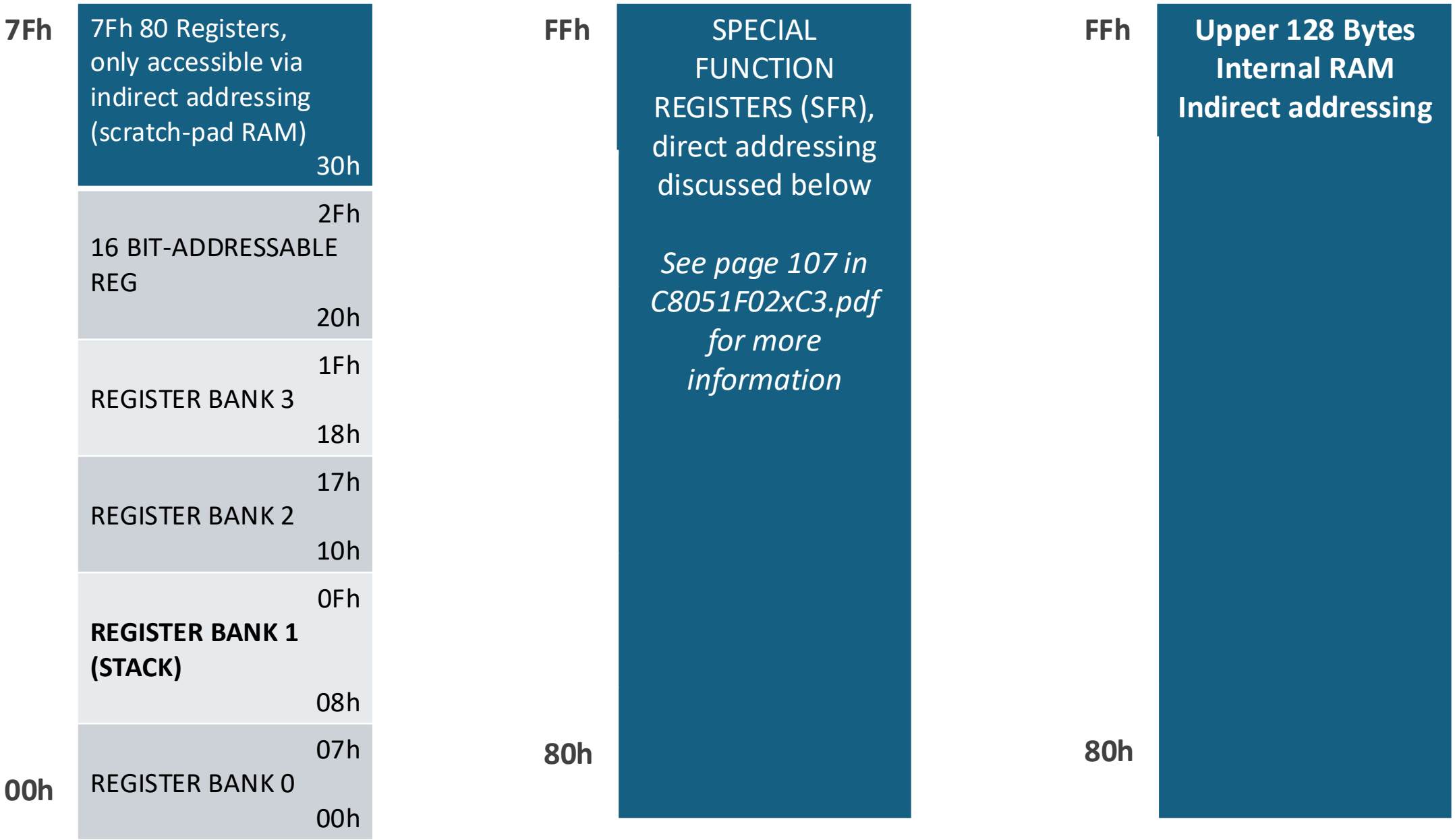
- RAM (256 x 8) – Data and Registers.
- Flash (64k x 8) – Program/code.
- Boot Loader (2k x 8) – Bootstrap code.
- EEPROM (2k x 8) – Firmware code.
- ERAM (2048) – Extended memory.



Components of CPU:

- ALU, PC, registers (A, B, etc.), series/parallel ports, timers, interrupt, etc.

8051 Data Memory Map

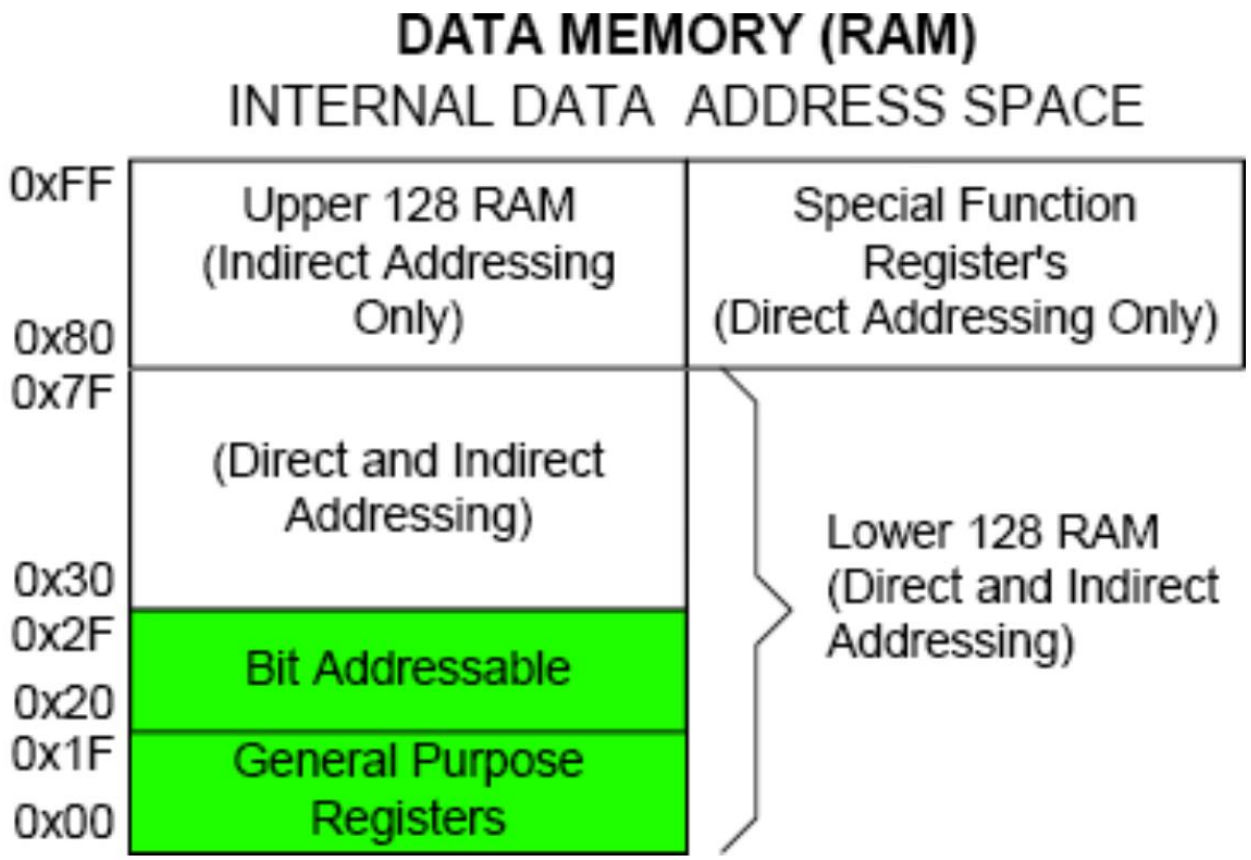


8051 Data Memory (RAM)

Internal Data Memory space is divided into three sections:

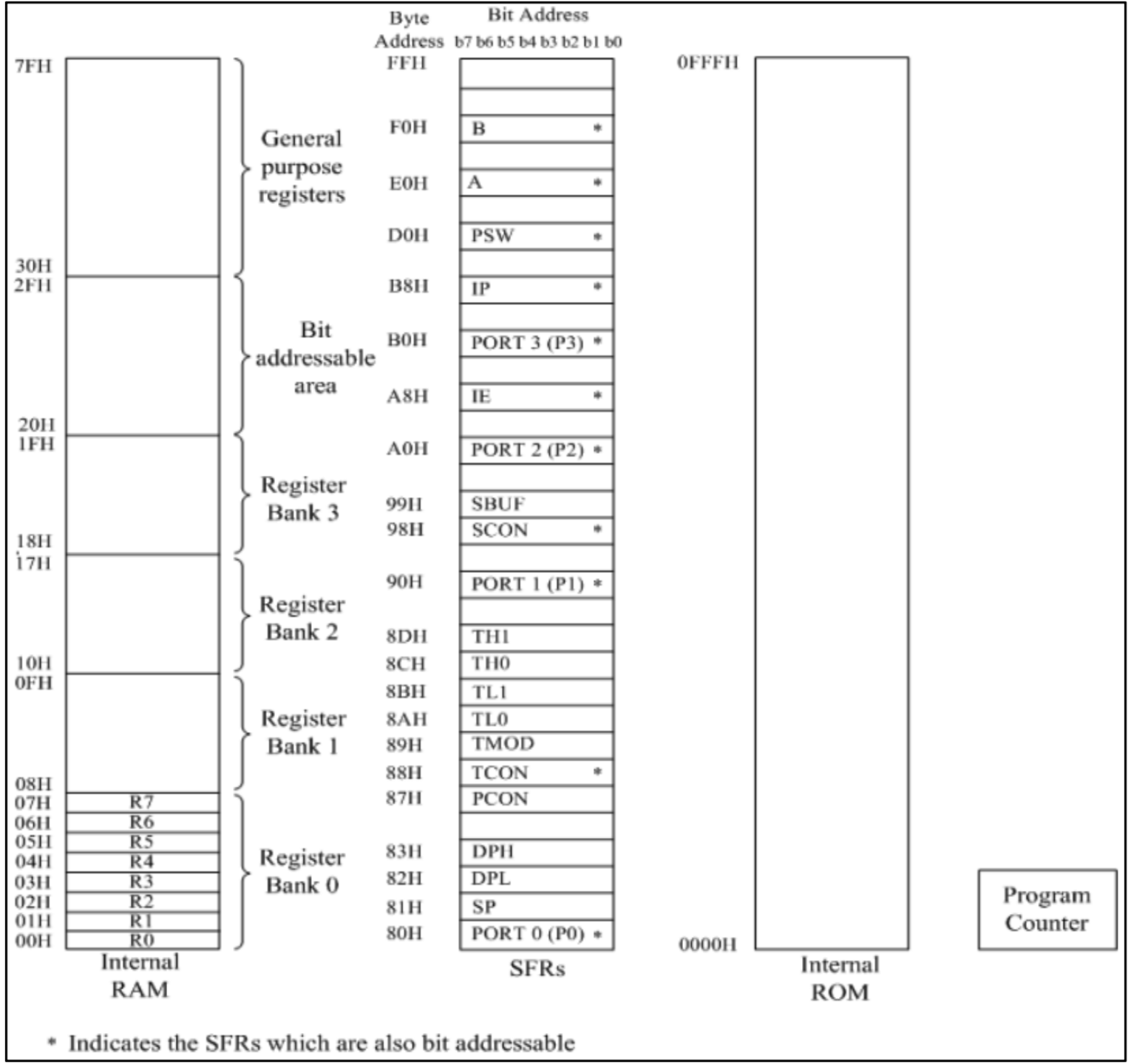
- Lower 128.
- Upper 128.
- Special function register (SFR).

- There are 384 bytes of memory space physically, though the Upper 128 and SFRs share the same addresses from location 80H to FFH.



- Appropriate instructions should be used to access each memory block.

8051 Data Memory (RAM)



Lower 128 - Register Banks and RAM

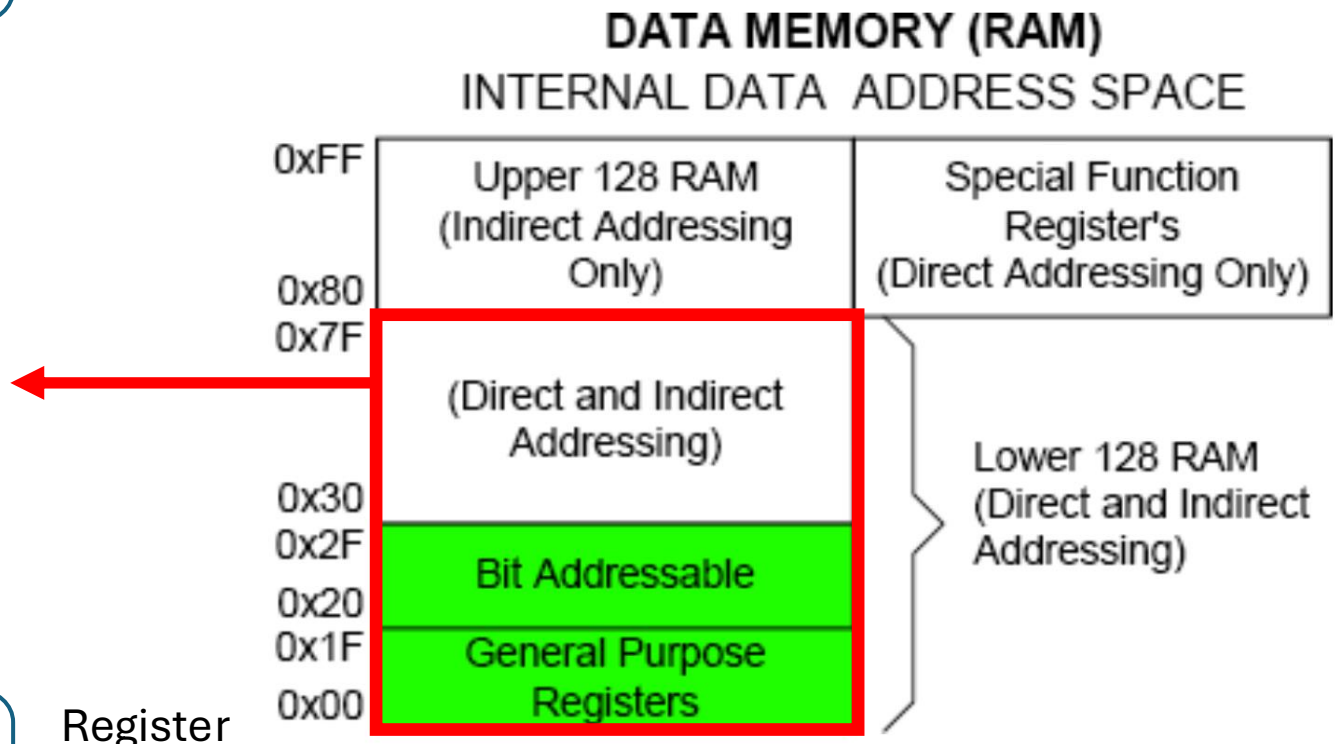
Byte Address	Bit Address							
7F	General Purpose RAM							
30								
2F	7F	7E	7D	7C	7B	7A	79	78
2E	77	76	75	74	73	72	71	70
2D	6F	6E	6D	6C	6B	6A	69	68
2C	67	66	65	64	63	62	61	60
2B	5F	5E	5D	5C	5B	5A	59	58
2A	57	56	55	54	53	52	51	50
29	4F	4E	4D	4C	4B	4A	49	48
28	47	46	45	44	43	42	41	40
27	3F	3E	3D	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								

General Purpose RAM (80 bytes)

Bit-addressable Area (16 bytes)

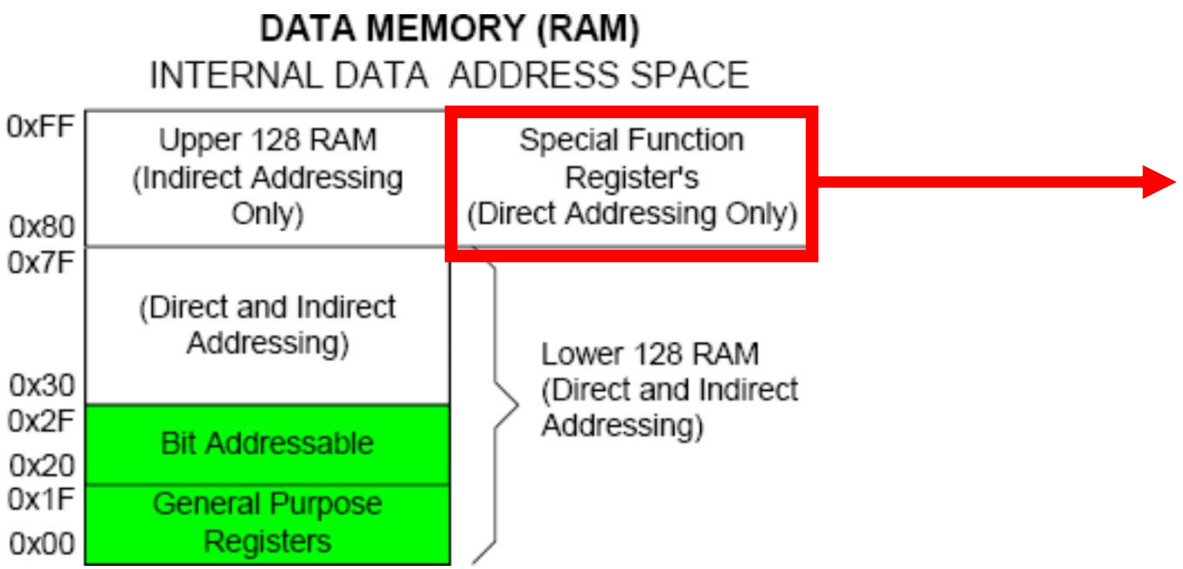
Register Banks (8 bytes per bank; 4 banks)

Lower 128 bytes memory area is for register banks, bit-addressable area (16 bytes) and general- purpose RAM



Notice that the general purpose of RAM is for direct and indirect addressing.

Upper 128 – Special Function Registers (SFRs)

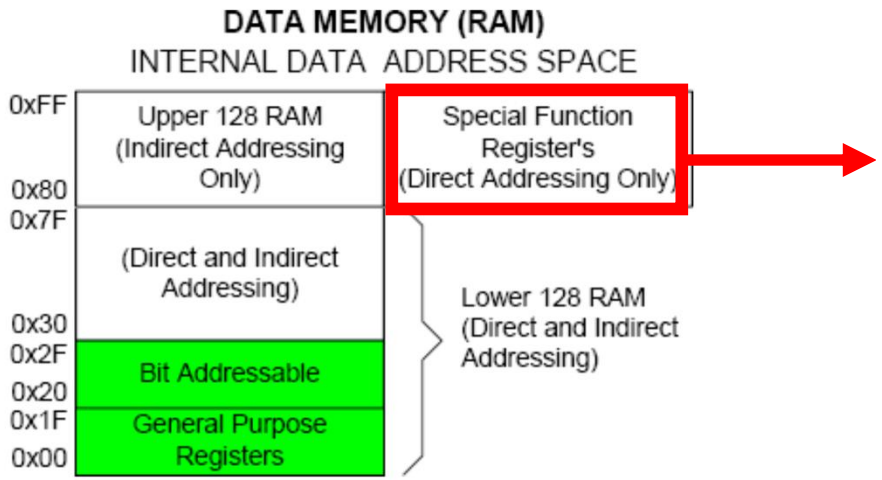


- SFRs provide control and data exchange with the microcontroller's resources and peripheral (not all are shown here, only the ones common to the core).
- Registers which have their byte addresses ending with 0H or 8H are byte- as well as bit-addressable.
- Some registers are not bit- addressable. These include the stack pointer (SP) and data pointer register (DPTR).

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	D1	D0	PSW
B8	BF	BE	BD	BC	BB	BA	B9	B8	IP
B0	B7	B6	B5	B4	B3	B2	B1	B0	P3
A8	AF	AE	AD	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	
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

AR

AT89C51AC3 Special Function Registers (SFRs)



- Alternative view looking at the SFRs.
- SFRs are arranged as a matrix of memory addresses.

Table 1. SFR Mapping

	0/8[2]	1/9	2/A	3/B	4/C	5/D	6/E	7/F	
F8h	IPL1 xxxx x0x0	CH 0000 0000	CCAP0H 0000 0000	CCAP1H 0000 0000	CCAP2H 0000 0000	CCAP3H 0000 0000	CCAP4H 0000 0000		FFh
F0h	B 0000 0000		ADCLK xxx0 0000	ADCON x000 0000	ADDL 0000 0000	ADDH 0000 0000	ADCF 0000 0000	IPH1 xxxx x0x0	F7h
E8h	IEN1 xxxx x0x0	CL 0000 0000	CCAP0L 0000 0000	CCAP1L 0000 0000	CCAP2L 0000 0000	CCAP3L 0000 0000	CCAP4L 0000 0000		EFh
E0h	ACC 0000 0000								E7h
D8h	CCON 0000 0000	CMOD 00xx x000	CCAPM0 x000 0000	CCAPM1 x000 0000	CCAPM2 x000 0000	CCAPM3 x000 0000	CCAPM4 x000 0000		DFh
D0h	PSW 0000 0000	FCON 0000 0000	EECON xxxx xx00	FSTA xxxx xx00	SPCON 0001 0100	SPSCR 0000 0000	SPDAT xxxx xxxx		D7h
C8h	T2CON 0000 0000	T2MOD xxxx xx00	RCAP2L 0000 0000	RCAP2H 0000 0000	TL2 0000 0000	TH2 0000 0000			CFh
C0h	P4 xxx1 1111								C7h
B8h	IPL0 x000 0000	SADEN 0000 0000							BFh
B0h	P3 1111 1111							IPH0 x000 0000	B7h
A8h	IEN0 0000 0000	SADDR 0000 0000							AFh
A0h	P2 1111 1111		AUXR1 xxxx 00x0				WDTRST 1111 1111	WDTPRG xxxx x000	A7h
98h	SCON 0000 0000	SBUF 0000 0000						CKCON1 xxxx xxx0	9Fh
90h	P1 1111 1111								97h
88h	TCON 0000 0000	TMOD 0000 0000	TL0 0000 0000	TL1 0000 0000	TH0 0000 0000	TH1 0000 0000	AUXR x001 0100	CKCON0 x00 0000	8Fh
80h	P0 1111 1111	SP 0000 0111	DPL 0000 0000	DPH 0000 0000				PCON 00x1 0000	87h
	0/8[2]	1/9	2/A	3/B	4/C	5/D	6/E	7/F	

8051 General-Purpose Registers

The lowest addresses of the 8051's internal RAM are a number of General-Purpose Registers.

- There are four banks of these registers, with each bank having 8 registers (R0 through R7).
  - 32 Bytes total.
- By default, Bank 0 is enabled.
  - As default, MOV A,R7 will move Bank 0's contents to accumulator.
  - We can switch register banks using bits 3 and 4 of the PSW flag register.

Bank 0	Bank 1	Bank 2	Bank 3
R7	R7	R7	R7
R6	R6	R6	R6
R5	R5	R5	R5
R4	R4	R4	R4
R3	R3	R3	R3
R2	R2	R2	R2
R1	R1	R1	R1
R0	R0	R0	R0

	RS1	RS0
BANK 0	0	0
BANK 1	0	1
BANK 2	1	0
BANK 3	1	1

8051 Flags

- The flag is raised when a specific condition of the operation of the program is met.
- Arithmetic flags:
  - Carry flag.
  - Auxiliary carry flag.
  - Overflow flag.
  - Parity flag.
- Specific purpose flag:
  - Specific register flag: timer overflow flag, external interrupt flag, etc.
  - User-defined flag.

PSW.7 CY (Carry flag, raised when the processor needs to carry in addition)	PSW .6 AC (Aux. carry, used during BCD math)	PSW.5 F0 (User-assignable flag)	PSW.4 RS1 (Register bank selector, don't worry about for now)	PSW.3 RS2 (Register bank selector, don't worry about for now)	PWS.2 OV (Overflow, raised when a signed number overflows into the sign bit)	PSW.1 - (User assignable)	PSW.0 P (Parity: 0 if acc. holds an even number of 1's)

8051 Flags - How Flags are Used?

1. Conditional Execution Flags are used in conditional jump instructions (like `JC` – Jump if Carry set, `JZ` – Jump if Zero set). These instructions allow the program to execute different code blocks based on the status of the flags.

2. Interrupt Handling: Flags like `IE` and `IF` are critical for managing interrupts. When an interrupt occurs, the processor checks the flags to determine if it should respond to the interrupt.

3. Error Detection: Flags like `O` (Overflow) and `C` (Carry) can be used to detect and handle arithmetic errors.

4. Status Reporting: Flags provide a way for the processor to report the status of operations to the rest of the program.

Example

Adding two numbers, and you want to check if the result overflows. You would:

1. Perform the addition.
2. Check the Overflow flag (O). If O is set, it means an overflow occurred, and you can take appropriate action (e.g., display an error message).

Overflow occurs when the result of an arithmetic operation (addition, subtraction, multiplication, or division) is too large (positive) or too small (negative) to be represented by the number of bits available in the data type being used

Example 2 - Register Programming

The following assembly language program illustrates programming with the general purpose register.

```
MOV R0,#FFH             ;Load R0 with 0xFF (immediate addressing)
MOV R1,R0               ;Copy R0 contents to R1 (register addressing)
MOV P0,R1               ;2 cycles! Direct move from R1 to P0
MOV 00,#FFH             ;Address 00 is R0 (direct addressing)

;;Now, we'll switch reg. banks to bank 3 (PSW.4 HI, PSW.3 HI).
SETB PSW.4              ;Sets the register bank select bit RS1 high
SETB PSW.3              ;Sets the register bank select bit RS2 high


;;R0-R7 now refer to bank 3 rather than bank 0 (default)
MOV R0,#FFH
MOV 00,#FFH             ;Same operation as code on top, different bank
```

Instruction Types

The 8051 instructions are divided into five functional groups:

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

Arithmetic Operations

Mnemonic	Description
ADD A. Rn	A=A+(Ro]
ADD A, direct	A= A + [direct memory]
ADD A @Ri	M=A+ (memory pointed to by Ril
ADD A #data	A= A + immediate data
ADD C A.Rn	A-A+ [Rn] + CY
ADD C A, direct	A = A + [direct memory] + CY
ADD C A@Ri	A-A+ (memory pointed to by Ril + CY
ADD C A#data	A - A + immediate data + CY
SUBB A, Rn	A - A- (Rn]- CY
SUBB A, direct	A= A - [direct memory| - CY
SUBB A@RI	A=A-(@Ri] -CY
SUBB A,#data	A - A- immediate data - CY
NC A	A-A +1
INC Rn	(Rn] = [Rn] + 1
INC direct	[direct] = [direct] + 1
INC QRi	@Ri] = @Ri] + 1
DEC A	A=A-1
DEC Rn	[Rn] = [Rn)1
DEC direct	[direct] = [direct] - 1
DEC @Ri	QRT= QR]-1
MUL AB	Multiply A & B
DV AB	Divide A by B
DA A	Decimal adjust A

With arithmetic instructions, the 8051 CPU has no special knowledge of the data format (e.g. signed binary, unsigned binary, binary coded decimal, ASCII, etc.).

- The appropriate status bits in the PSW are set when specific conditions are met, which allows the user software to manage the different data formats.

[@Ri] implies the contents of the memory location pointed to by R0 or R1.

- Rn refers to registers R0-R7 of the currently selected register bank.
- Example:

```
ADD A, R1      ;add A with R1
SUB A, x01H    ;substract A 1 bit
INC A          ;increment A by 1 bit
DEC A          ;decrement A by 1 bit
```

Logical Operations

Mnemonic	Description
ANL A, Rn	A = A & [Rn]
ANL A, direct	A = A & (direct memory)
ANL A@Ri	A - A & [memory pointed to by Ri]
ANL A,#data	A= A & immediate data
ANL direct A	[direct) = (direct] & A
ANL direct,#data	[direct] = [direct] & immediate data
ORL A, Rn	A = A OR [Rn]
ORL A, direct	A = A OR (direct]
ORL A@Ri	A= A OR (@RI]
ORL A,#data	A = A OR immediate data
ORL direct,A	[direct] = [direct] OR A
ORL direct,#data	[direct] = [direct] OR immediate data
XRL A, Rn	A = AXOR [Rn]
XRL A, direct	A = A XOR [direct memory]
XRL A,@Ri	A = AXOR (@Ri]
XRL A.#data	A = A XOR immediate data
XRL direct,A	[direct] = [direct] XOR A
XRL direct,#data	[direct] = [direct] XOR immediate data
CLR A	Clear A
CPL A	Complement A
RL	Rotate A left
RL A	Rotate A left (through C)
RR	Rotate A right
RRC A	Rotate A right (through C)
SWAP A	Swap nibbles

- Logical instructions perform Boolean operations (AND, OR, XOR, and NOT) on data bytes on a bit-by-bit basis.

- Example:

```
ANL A, #02H ;Mask bit 1
ORL TCON, A ;TCON=TCON-OR-A
```

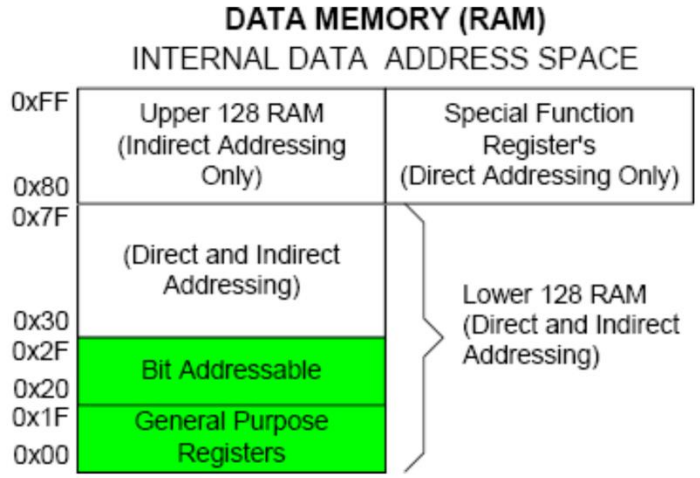
Data Transfer

Mnemonic	Description
MOV @Ri, direct	[@Ri] = [direct]
MOV @Ri, #data	[@Ri] = immediate data
MOV DPTR, #data 16	[DPTR] = immediate data
MOVC A,@A+DPTR	A = Code byte from [@A+DPTR]
MOVC A,@A+PC	A = Code byte from [@A+PC]
MOVX A,@Ri	A = Data byte from external ram [@Ri]
MOVX A,@DPTR	A = Data byte from external ram [@DPTR]
MOVX @Ri, A	External[@Ri] = A
MOVX @DPTR,A	External[@DPTR] = A
PUSH direct	Push into stack
POP direct	Pop from stack
XCH A,Rn	A = [Rn], [Rn] = A
XCH A, direct	A = [direct], [direct] = A
XCH A, @Ri	A = [@Rn], [@Rn] = A
XCHD A,@Ri	Exchange low order digits[AR]

- Data transfer instructions can be used to transfer data between an internal RAM location and an SFR location without going through the accumulator.
- It is also possible to transfer data between the internal and external RAM by using indirect addressing.
- Mostly data transfer is by using MOV opcode.
- PUSH and POP are used when dealing with data transfer in the stack.

Data Transfer

- The upper 128 bytes of data RAM are accessed only by indirect addressing.
- The SFRs are accessed only by direct addressing.



- Example:

```
MOV A,x21H ;assign A with value of 21H
MOV A, 1AH ;assign A with content stored
at address 1AH
MOV @R1, x20H ;assign R1 with 20H
```

Boolean Variable Instructions

Mnemonic	Description
CLR C	Clear C
CLR bit	Clear direct bit
SETB C	Set C
SETB bit	Set direct bit
CPL C	Complement c
CPL bit	Complement direct 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	MOV bit to C
MOV bit,C	MOV C to bit
JC rel	Jump if C set
JNC rel	Jump if C not set
JB bit,rel	Jump if specified bit set
JNB bit,rel	Jump if specified bit not set
JBC bit,rel	if specified bit set then clear it and jump

The 8051 processor can perform single-bit operations.

- The operations include *set, clear*, *and, or* and *complement* instructions.
- Also included are bit–level moves or conditional jump instructions.
- All bit accesses use direct addressing.

- Examples:

```
SETB TR0  ;Start Timer0.
POLL: JNB TR0, POLL
;Wait till timer overflows.
```

Program Branching Instructions

Mnemonic	Description
ACALL addr11	Absolute subroutine call
LCALL addr16	Long subroutine call
RET	Return from subroutine
RETI	Return from interrupt
AJMP addr11	Absolute jump
LJMP addr16	Long jump
SJMP rel	Short jump
JMP @A+DPTR	Jump indirect
JZ rel	Jump if A=0
JNZ rel	Jump if A NOT=0
CJNE A,direct, rel	Compare and Jump if Not Equal
CJNE A,#data,rel	
CJNE Rn, #data,rel	
CJNE @Ri,#data, rel	
DJNZ Rn,rel	Decrement and Jump if Not Zero
DJNZ direct,rel	
NOP	No Operation

- Program branching instructions are used to control the flow of program execution.
- Some instructions provide decision making capabilities before transferring control to other parts of the program (conditional branches).

- Example:

```
ACALL label1        ; jump to XX
LCALL delay         ;jump to XX
RET
RETI

SJMP loop1          ;jump to XX
LJMP main ;         jump to XX
```

Example 3 – Content Operand

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

```
MOV R5, #10H     ;R5 = 10H
INC R5
INC R5
MOV R0, #20H
MOV A, #0FFH
MOV 20H, A
MOV @R0, #10H
INC A
MOV 20H, # 00H
INC 20H
```

Example 3 – Content Operand

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, #FFH                 ;A = FFH
MOV 20H, A                  ;(20H) = FFH
MOV @R0, #10H         ;(20H) = 10H
INC A                       ; A = OOH
MOV 20H, # 00H        ;(20H) = 00H
INC 20H               ;(20H) = 01H
```

Example 4 – Square Number

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).

Note:

- Square arithmetic operation could be implemented a multiplication of the a number with itself
- Multiplication and division could result in a big number.
- Consider storing the results of the operation as 16 bits value.

Example 4 – Square Number

SquareNumber.asm

```
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 result (LSByte) into address 60H
MOV 61H, A              ;Copy result (MSByte) into address 6aH
```

Example 4 – Square Number

If the number is AAH, what will be the result and status of the OV flag after finding the square of that number?

PSW.7	PSW.6	PSW.5	PSW.4	PSW.3	PSW.2	PSW.1	PSW.0
CY	AC	F0	RS1	RS0	OV	--	P

Bit	Symbol	Flag name and description		
7	C (or CY)	Carry; Used in arithmetic, logic and Boolean operations		
6	AC	Auxiliary carry ; useful only for BCD arithmetic		
5	F0	Flag 0; general purpose user flag		
4	RS1	Register bank selection bit 1		
3	RS0	Register bank selection bit 0		
		RS1	RS0	
		0	0	Bank 0
		0	1	Bank 1
		1	0	Bank 2
		1	1	Bank 3
2	0V	Overflow; used in arithmetic operations		
1	--	Reserved; may be used as a general purpose flag		
0	P	Parity; set to 1 if A has odd number of ones, otherwise reset to 0		

Example 4 – Square Number

If the number is AAH, what will be the result and status of the OV flag after finding the square of that number?

MUL AB

and stores the total of 16-bit result as:

- Low byte → A (8 bit)
- High byte → B (8 bit)

Example 4 – Square Number

If the number is AAH, what will be the result and status of the OV flag after finding the square of that number?

Convert Hex to decimal

Hex numbers use powers of 16.
$AAH = A \times 16^1 + A \times 16^0$

Substitute A = 10:
$= 10 \times 16 + 10 \times 1$
$= 160 + 10$
$= 170$

So:
AAH = 170 decimal

Convert Hex to binary to decimal

AAH in binary is:
10101010_2

Convert to decimal:
128 + 32 + 8 + 2 = 170

Example 4 – Square Number

If the number is AAH, what will be the result and status of the OV flag after finding the square of that number?

AAH = 170 decimal

Then its square is:
AA × AA = 170 × 170 = 28900 decimal = 70E4H
So after multiplication:
Result: 70E4H

to hexadecimal:
Divide repeatedly by 16.

Division	Quotient	Remainder
28900 ÷ 16	1806	4
1806 ÷ 16	112	14 = E
112 ÷ 16	7	0
7 ÷ 16	0	7

Example 4 – Square Number

If the number is AAH, what will be the result and status of the OV flag after finding the square of that number?

Register	Value
A	E4H (low byte)
B	70H (high byte)

Result: 70E4H
Since the high byte (B = 70H) is nonzero, the result **does not fit within 8 bits**.
Therefore:
OV = 1 and
CY = 0
Final answer:
Square of AAH = 70E4H OV flag = 1

Example 4 – Square Number

- If the number is AAH, then result will be 70E4H.
- Since result is greater than FFH the overflow flag will be set,

i.e. OV=1 after multiplication.

Review

1. Convert Hex to Binary
3AH → Binary

2. Convert Decimal to Hex
255_{10} → Hex

3. Convert Binary to Hex
10101111_2 → Hex

4. Convert Hex to Decimal
7CH → Decimal

5. Convert Decimal to Hex
100_{10} → Hex

6. Convert Binary to Hex
11110000_2 → Hex

7. Convert Hex to Binary
B5H → Binary

8. Convert Decimal to Hex
170_{10} → Hex

9. Convert Binary to Decimal
11001010_2 → Decimal

10. Convert Hex to Decimal
1F4H → Decimal

Review

1. Convert Hex to Binary

3AH → Binary

Answer

3AH = 00111010_2

2. Convert Decimal to Hex

255_{10} → Hex

Answer

255_{10} = FFH

3. Convert Binary to Hex

10101111_2 → Hex

Answer

10101111_2 = AFH

4. Convert Hex to Decimal

7CH → Decimal

Answer

7CH = 124_{10}

Calculation:

7×16 + 12 = 112 + 12 = 124

5. Convert Decimal to Hex

100_{10} → Hex

Answer

100_{10} = 64H

Review

6. Convert Binary to Hex

11110000_2 → Hex

Answer

11110000_2 = F0H

7. Convert Hex to Binary

B5H → Binary

Answer

B5H = 10110101_2

8. Convert Decimal to Hex

170_{10} → Hex

Answer

170_{10} = AAH

9. Convert Binary to Decimal

11001010_2 → Decimal

Answer

$11001010_2 = 202_{10}$

Calculation:

128 + 64 + 8 + 2 = 202

10. Convert Hex to Decimal

1F4H → Decimal

Answer

$1F4H = 500_{10}$

Calculation:

$1\times16^2 + 15\times16 + 4 = 256 + 240 + 4 = 500$