# 8051 Addressing Mode and Instruction Set

# 8051 Instruction Set

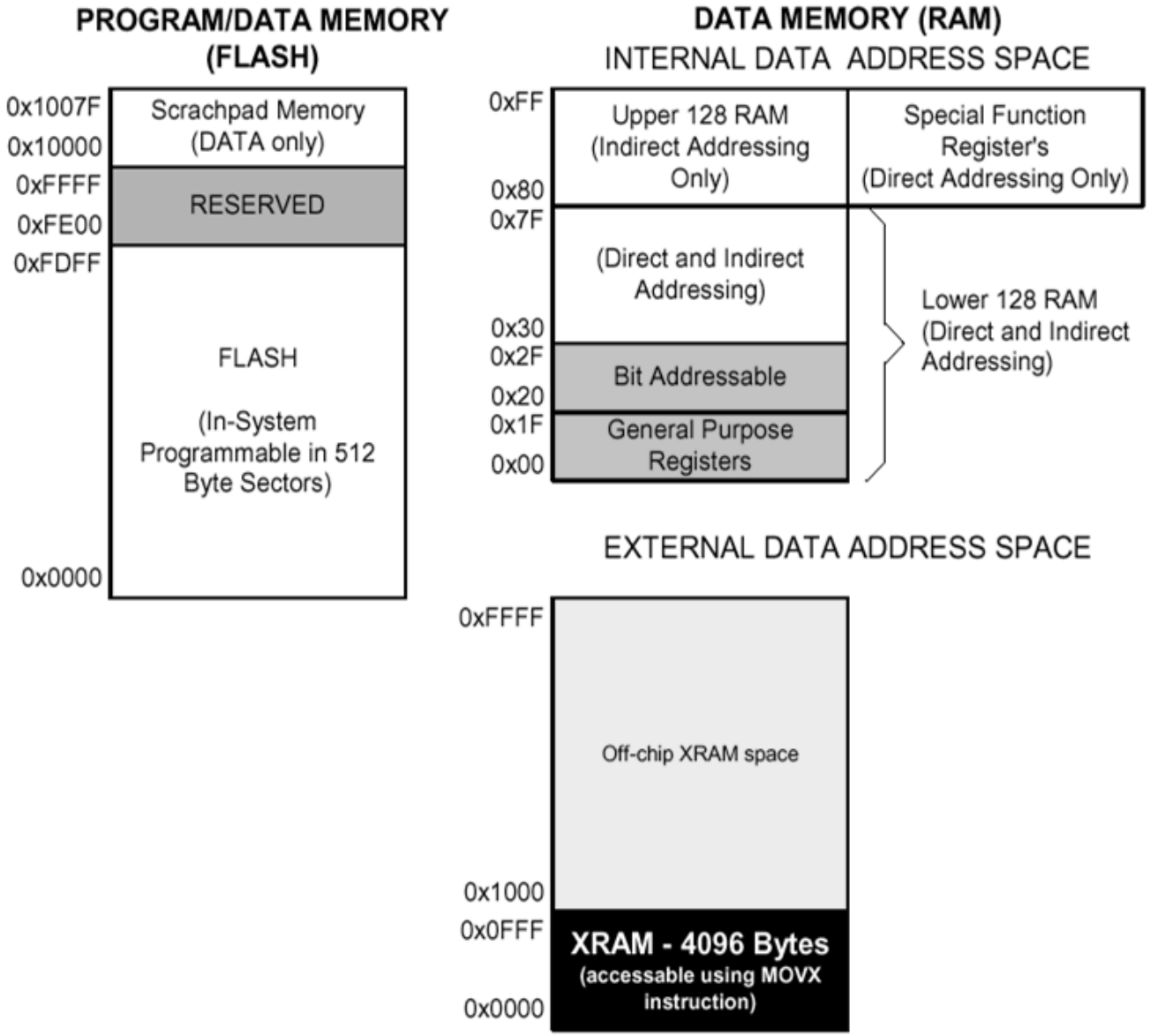
- Addressing Modes
  - Register addressing
  - Direct addressing
  - Indirect addressing
  - Immediate constant addressing
  - Relative addressing
  - Absolute addressing
  - Long addressing
  - Indexed addressing
- Instruction Types
  - Arithmetic operations
  - Logical operations
  - Data transfer instructions
  - Boolean variable instructions
  - Program branching instructions

# Introduction

- 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:

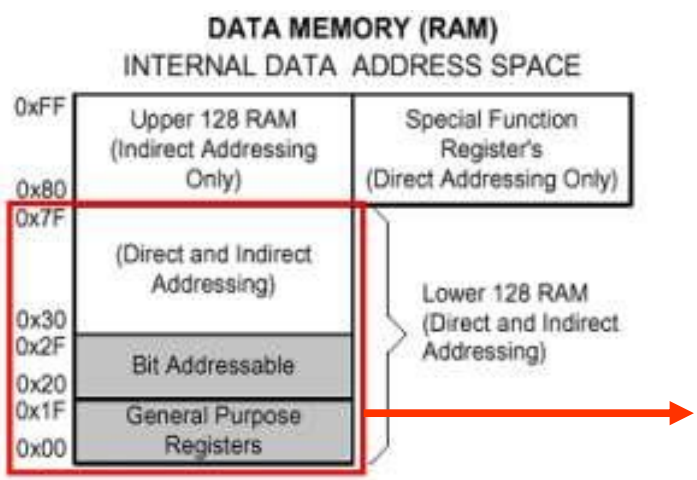
  *Instruction Destination, Source*

# Memory Organization



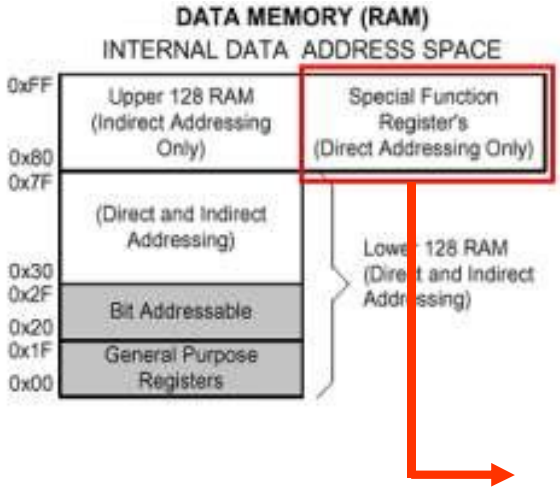

- The memory organization of C8051F020 is similar to that of a standard 8051
- Program and data memory share the same address space but are accessed via different instruction types

# Internal Data Memory




| | Byte Address | Bit Address | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 7F | General Purpose RAM | | | | | | | |
| | 30 | | | | | | | | |
| B | 2F | 7F | 7E | 7D | 7C | 7B | 7A | 79 | 78 |
| i | 2E | 77 | 76 | 75 | 74 | 73 | 72 | 71 | 70 |
| t | 2D | 6F | 6E | 6D | 6C | 6B | 6A | 69 | 68 |
| | 2C | 67 | 66 | 65 | 64 | 63 | 62 | 61 | 60 |
| A | 2B | 5F | 5E | 5D | 5C | 5B | 5A | 59 | 58 |
| d | 2A | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 |
| d | 29 | 4F | 4E | 4D | 4C | 4B | 4A | 49 | 48 |
| r | 28 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 |
| e | 27 | 3F | 3E | 3D | 3C | 3B | 3A | 39 | 38 |
| s | 26 | 37 | 36 | 35 | 34 | 33 | 32 | 31 | 30 |
| s | 25 | 2F | 2E | 2D | 2C | 2B | 2A | 29 | 28 |
| a | 24 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 |
| b | 23 | 1F | 1E | 1D | 1C | 1B | 1A | 19 | 18 |
| l | 22 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 |
| e | 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 | | | | | | | | |

# Special Function Registers




| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| F8 | SPI0CN | PCA0H | PCA0CPH0 | PCA0CPH1 | PCA0CPH2 | PCA0CPH3 | PCA0CPH4 | WDTCN |
| F0 | B | SCON1 | SBUF1 | SADDR1 | TL4 | TH4 | EIP1 | EIP2 |
| E8 | ADC0CN | PCA0L | PCA0CPL0 | PCA0CPL1 | PCA0CPL2 | PCA0CPL3 | PCA0CPL4 | RSTSRC |
| E0 | ACC | XBR0 | XBR1 | XBR2 | RCAP4L | RCAP4H | EIE1 | EIE2 |
| D8 | PCA0CN | PCA0MD | PCA0M0 | PCA0CPM1 | PCA0CPM2 | PCA0CPM3 | PCA0CPM4 | |
| D0 | PSW | REF0CN | DAC0L | DAC0H | DAC0CN | DAC1L | DAC1H | DAC1CN |
| C8 | T2CON | T4CON | RCAP2L | RCAP2H | TL2 | TH2 | | SMB0CR |
| C0 | SMB0CN | SMB0STA | SMB0DAT | SMB0ADR | ADC0GTL | ADC0GTH | ADC0LTL | ADC0LTH |
| B8 | IP | SADEN0 | AMX0CF | AMX0SL | ADC0CF | P1MDIN | ADC0L | ADC0H |
| B0 | P3 | OSCXCN | OSCICN | | | P74OUT | FLSCL | FLACL |
| A8 | IE | SADDR0 | ADC1CN | ADC1CF | AMX1SL | P3IF | SADEN1 | EMI0CN |
| A0 | P2 | EMI0TC | | EMI0CF | P0MDOUT | P1MDOUT | P2MDOUT | P3MDOUT |
| 98 | SCON0 | SBUF0 | SPI0CFG | SPIODAT | ADC1 | SPI0CKR | CPT0CN | CPT1CN |
| 90 | P1 | TMR3CN | TMR3RLL | TMR3RLH | TMR3L | TMR3H | P7 | |
| 88 | TCON | TMOD | TL0 | TL1 | TH0 | TH1 | CKCON | PSCTL |
| 80 | P0 | SP | DPL | DPH | P4 | P5 | P6 | PCON |
| | 0(8) Bit addressable | 1(9) | 2(A) | 3(B) | 4(C) | 5(D) | 6(E) | 7(F) |

# Addressing Modes

- Eight modes of addressing are available with the C8051F020
- The different addressing modes determine how the operand byte is selected

| Addressing Modes | Instruction |
|---|---|
| **Register** | **MOV A, B** |
| **Direct** | **MOV 30H,A** |
| **Indirect** | **ADD A,@R0** |
| **Immediate Constant** | **ADD A,#80H** |
| **Relative*** | **SJMP AHEAD** |
| **Absolute*** | **AJMP BACK** |
| **Long*** | **LJMP FAR_AHEAD** |
| **Indexed** | **MOVC A,@A+PC** |

* Related to program branching instructions

SILICON LABS

# Register Addressing

- The register addressing instruction involves information transfer between registers

- *Example*:

```
MOV     R0, A
```

- The instruction transfers the accumulator content into the R0 register. The register bank (Bank 0, 1, 2 or 3) must be specified prior to this instruction.

# Direct Addressing

- This mode allows you to specify the operand by giving its actual memory address (typically specified in hexadecimal format) or by giving its abbreviated name (e.g. P3)

  Note: Abbreviated SFR names are defined in the “C8051F020.inc” header file

- *Example*:

```
MOV     A, P3       ;Transfer the contents of
                    ;Port 3 to the accumulator
MOV     A, 020H     ;Transfer the contents of RAM
        ;location 20H to the accumulator
```

# Indirect Addressing

- This mode uses a pointer to hold the effective address of the operand
- Only registers R0, R1 and DPTR can be used as the pointer registers
- The R0 and R1 registers can hold an 8-bit address, whereas DPTR can hold a 16-bit address

- ***Examples*:**

```
MOV     @R0,A          ;Store the content of
                       ;accumulator into the memory
                       ;location pointed to by
        ;register R0. R0 could have an
;8-bit address, such as 60H.

MOVX    A,@DPTR        ;Transfer the contents from
                       ;the memory location
                       ;pointed to by DPTR into the
                       ;accumulator. DPTR could have a
        ;16-bit address, such as 1234H.
```

SILICON LABS

# Immediate Constant Addressing

- This mode of addressing uses either an 8- or 16-bit constant value as the source operand
- This constant is specified in the instruction, rather than in a register or a memory location
- The destination register should hold the same data size which is specified by the source operand

- *Examples*:

```
ADD A,#030H          ;Add 8-bit value of 30H to
                     ;the accumulator register
                     ;(which is an 8-bit register).

MOV DPTR,#0FE00H     ;Move 16-bit data constant
                     ;FE00H into the 16-bit Data
                     ;Pointer Register.
```

# Relative Addressing

- This mode of addressing is used with some type of jump instructions, like SJMP (short jump) and conditional jumps like JNZ
- These instructions transfer control from one part of a program to another
- The destination address must be within -128 and +127 bytes from the current instruction address because an 8-bit offset is used ($2^8 = 256$)
- *Example*:

```
GoBack:     DEC     A         ;Decrement A
            JNZ     GoBack    ;If A is not zero, loop back
```

# Absolute Addressing

- Two instructions associated with this mode of addressing are ACALL and AJMP instructions
- These are 2-byte instructions where the 11-bit absolute address is specified as the operand
- The upper 5 bits of the 16-bit PC address are not modified. The lower 11 bits are loaded from this instruction. So, the branch address must be within the current 2K byte page of program memory ($2^{11}$ = 2048)
- *Example*:

```
            ACALL   PORT_INIT       ;PORT_INIT should be
                                    ;located within 2k bytes.

PORT_INIT: MOV      P0, #0FH        ;PORT_INIT subroutine
```

# Long Addressing

- This mode of addressing is used with the LCALL and LJMP instructions
- It is a 3-byte instruction and the last 2 bytes specify a 16-bit destination location where the program branches
- It allows use of the full 64 K code space
- The program will always branch to the same location no matter where the program was previously
- *Example*:

```
        LCALL TIMER_INIT          ;TIMER_INIT address (16-bits
                                  ;long) is specified as the
                                ;operand; In C, this will be a
                ;function call: Timer_Init().
TIMER_INIT: ORL TMOD,#01H       ;TIMER_INIT subroutine
```

# Indexed Addressing

- The Indexed addressing is useful when there is a need to retrieve data from a look-up table
- A 16-bit register (data pointer) holds the base address and the accumulator holds an 8-bit displacement or index value
- The sum of these two registers forms the effective address for a JMP or MOVC instruction

- *Example*:

```
MOV     A,#08H          ;Offset from table start
MOV     DPTR,#01F00H    ;Table start address
MOVC    A,@A+DPTR       ;Gets target value from the table
                        ;start address + offset and puts it
                        ;in A.
```

- After the execution of the above instructions, the program will branch to address 1F08H (1F00H+08H) and transfer into the accumulator the data byte retrieved from that location (from the look-up table)