Principle and Interface Techniques of Microcontroller

--8051 Microcontroller and Embedded Systems Using Assembly and C

LI, Guang (李光) Prof. PhD, DIC, MIET

WANG, You (王酉) PhD, MIET

杭州 • 浙江大学 • 2015

Chapter 2
MCS-51 Assembly Language

Outline

- 2.1 Introduction to Assembly Language
- 2.2 8051 CPU
- 2.3 8051 Memory Space and Registers
- 2.4 Data transfer instructions

Introduction to Assembly Language

- In the early days of the computer, programmers coded in machine language, consisting of 0s and 1s

  Tedious, slow and prone to error

- Assembly languages, which provided mnemonics for the machine code instructions, plus other features, were developed

  An Assembly language program consist of a series of lines of Assembly language instructions

- Assembly language is referred to as a low-level language

  It deals directly with the internal structure of the CPU

Introduction to Assembly Language

- **Assembly language instruction includes**

  1) a mnemonic (abbreviation easy to remember)

  the commands to the CPU, telling it what those to do with those items

  2)optionally followed by one or two operands

  the data items being manipulated

- **A given Assembly language program is a series of statements, or lines**

  Assembly language instructions

  Tell the CPU what to do

  Directives (or pseudo-instructions)

- Give directions to the assembler

Structure of Assembly Language

- An Assembly language instruction consists of four fields:

[label:] Mnemonic [operands] [;comment]

```
        ORG   0H                ;start(origin) at location0
        MOV   R5, #25H          ;
        MOV   R7, #34H          ;
        MOV   A,#0             ;load 0 into A
        ADD   A, R5             ;add contents of R5 to A.
                                ; now A
                                ;add c
                                ;now A = A + R7
                                ;add to A value 12H, now A = A + 12H

HERE: SJMP HERE          ;stay in this loop
        END
```

Directives do not generate any machine code and are used only by the assembler

Mnemonics produce opcodes

Comments may be at the end of a line or on a line by themselves.
The assembler ignores comments

The label field allows the program to refer to a line of code by name

Assembler Directives

The DB directive is the most widely used data directive in the assembler.

It is used to define the 8-bit data

When DB is used to define data, the numbers can be in decimal, binary, hex, ASCII formats

```
        ORG   500H
DATA1:  DB  28            ;DECIMAL (1C
DATA2:  DB  00110101B     ;BINARY (35 in Hex)
DATA3:  DB  39H           ;HEX
        ORG 510H
DATA4:  DB  "2591"        ;ASCII NUM
        ORG 518H
TA6:    DB  "My name is Joe"  ;ASCII  CHARACTERS
```

The "D" after the decimal number is optional, but using "B" (binary) and "H" (hexadecimal) for the others

The Assembler will convert the numbers into hex

Place ASCII in quotation marks. The Assembler will assign ASCII code for the numbers or characters

Define ASCII strings larger than two characters

Assembler Directives

- ORG (origin)

  The ORG directive is used to indicate the beginning of the address

  The number that comes after ORG can be either in hex and decimal

  If the number is not followed by H, it is decimal and the assembler will convert it to hex

- END

  This indicates to the assembler the end of the source (asm) file

  The END directive is the last line of an 8051 program

  Mean that in the code anything after the END directive is ignored by the assembler

- EQU (equate)

  This is used to define a constant without occupying a memory location

  The EQU directive does not set aside storage for a data item but associates a constant value with a data label

  When the label appears in the program, its constant value will be substituted for the label

  Assume that there is a constant used in many different places in the program, and the programmer wants to change its value throughout

  By the use of EQU, one can change it once and the assembler will change all of its occurrences

```
COUNT   EQU   25
... ....
MOV   R3, #COUNT
```

Use EQU for the counter constant

The constant is used to load the R3 register

Assembling and Running an 8051 Program

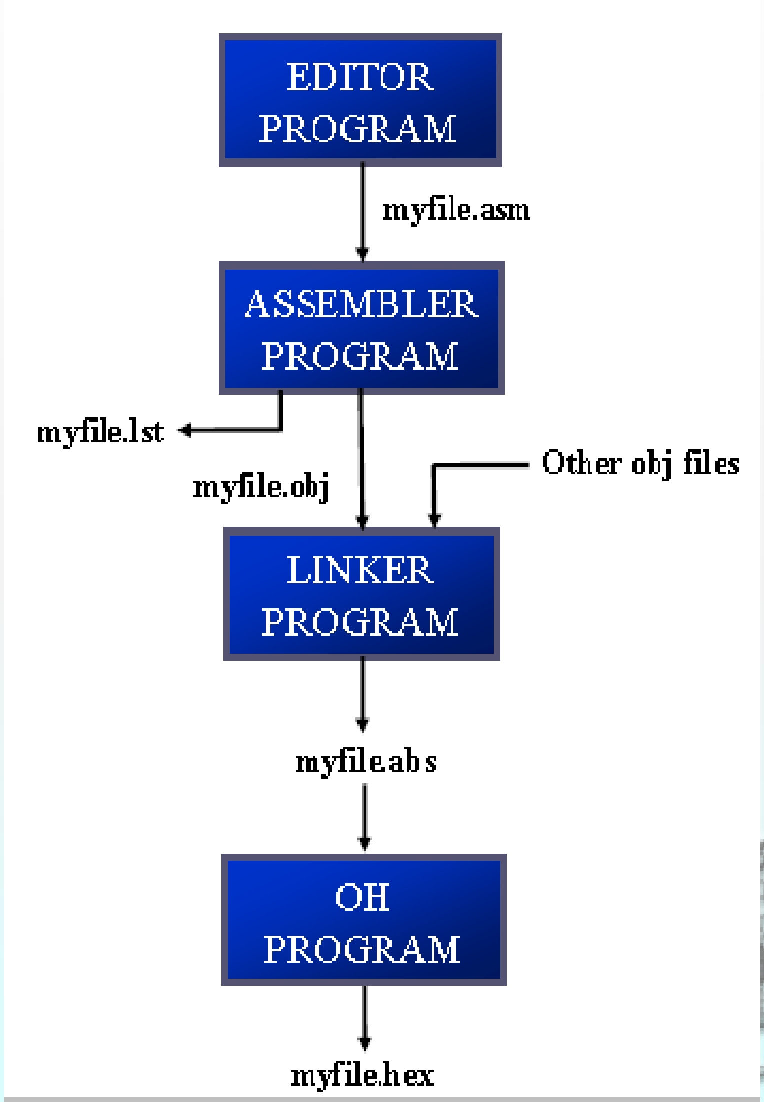
The step of Assembly language program are outlines as follows:

1)Use an editor to type a program

2) The “asm” source file containing the program code created in step 1 is fed to an 8051 assembler

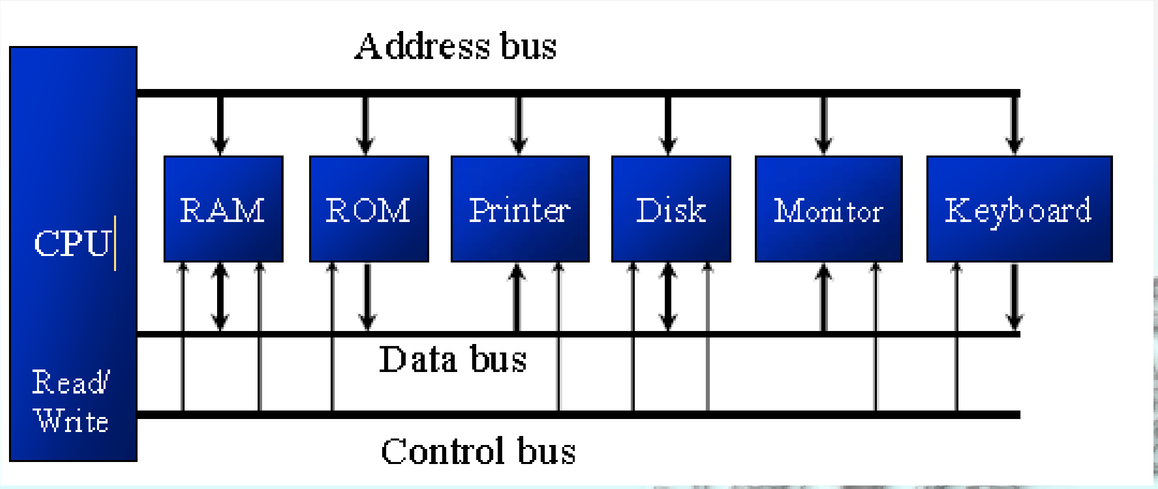
3) Assembler require a third step called linking

4) Next the “abs” file is fed into a Program called “OH” (object to hex converter)which creates a file with extension “hex” that is ready to burn into ROM



8051 CPU

- The CPU is connected to memory and I/O through strips of wire called a bus
  - Carries information from place to place:
  - Address bus
  - Data bus
  - Control bus



- **Address bus**

  For a device (memory or I/O) to be recognized by the CPU, it must be assigned an address

  The address assigned to a given device must be unique

  The CPU puts the address on the address bus, and the decoding circuitry finds the device

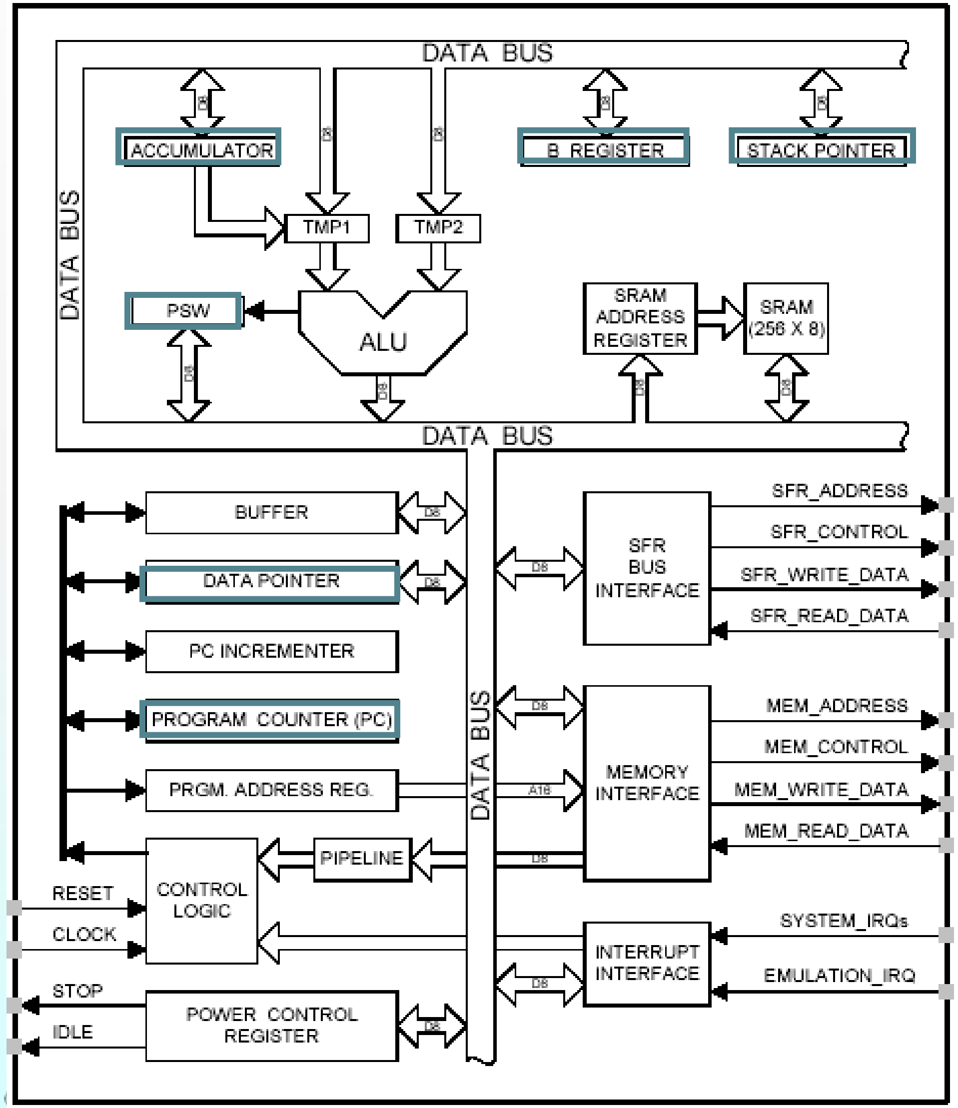
- **Data bus**

  The CPU either gets data from the device or sends data to it.

- **Control bus**

  Provides read or write signals to the device to indicate if the CPU is asking for information or sending it information

8051 CPU

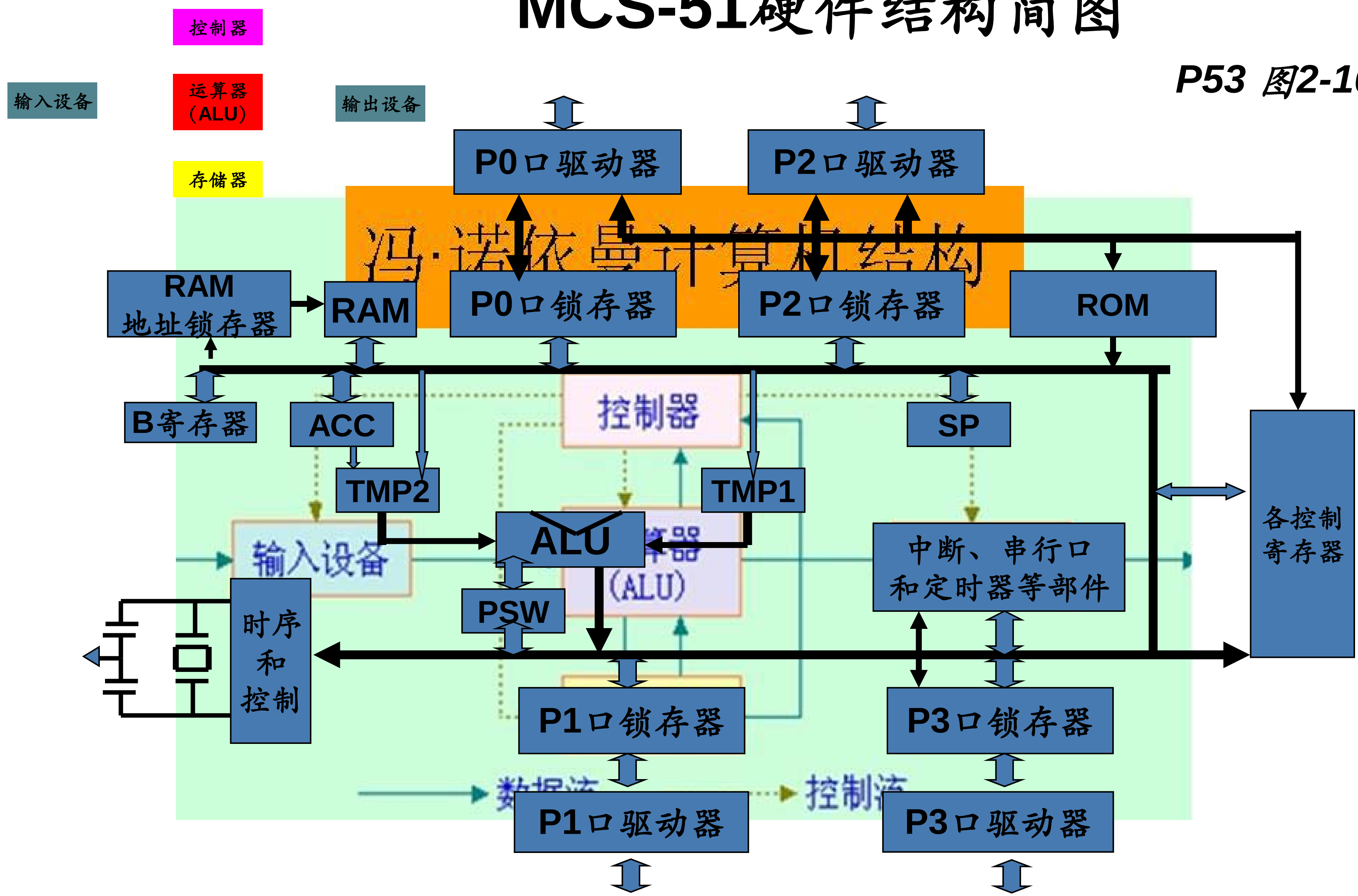


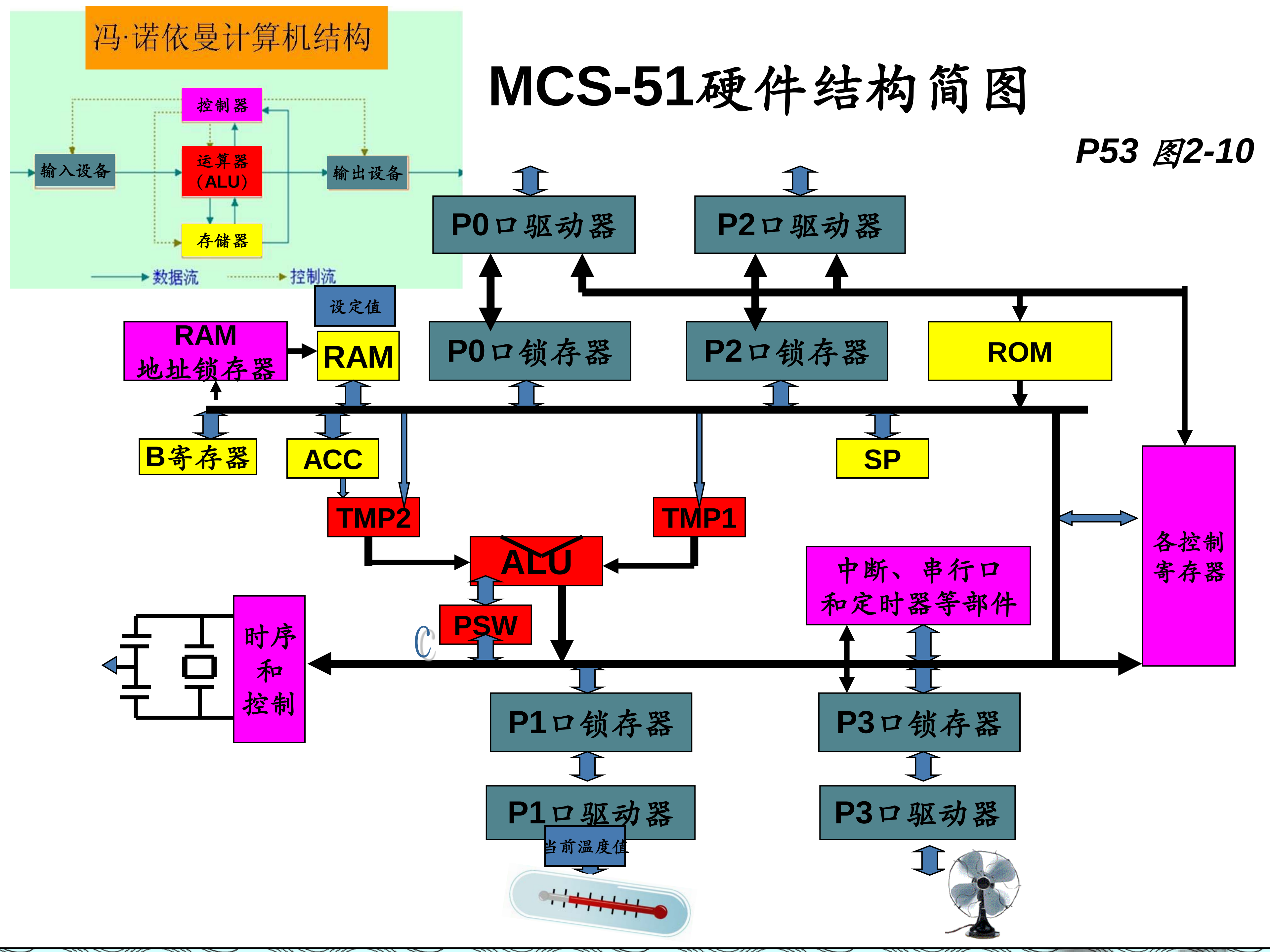
A (Accumulator)
B
PSW (Program Status Word)
SP (Stack Pointer)
PC (Program Counter)
DPTR (Data Pointer)

Used in assembler instructions

MCS-51硬件结构简图

P53 图2-10




冯·诺依曼计算机结构
控制器
输入设备
运算器（ALU）
输出设备
存储器
数据流
控制流
MCS-51硬件结构简图
P53 图2-10
P0口驱动器
P2口驱动器
设定值
RAM 地址锁存器
RAM
P0口锁存器
P2口锁存器
ROM
B寄存器
ACC
SP
TMP2
TMP1
ALU
PSW
各控制寄存器
中断、串行口和定时器等部件
时序和控制
P1口锁存器
P3口锁存器
P1口驱动器
P3口驱动器
当前温度值

- ALU: Accomplish arithmetic operation, logic operation, bit manipulation with cooperation of related registers (A, B, PSW).

A (ACC): For all arithmetic and logic instructions

B: For multiplication and division

PSW, also referred to as the flag register, is an 8 bit register. Only 6 bits are used

D7	D6	D5	D4	D3	D2	D1	D0
Cy	AC	F0	RS1	RS0	OV	_	P

PSW

D7	D6	D5	D4	D3	D2	D1	D0
Cy	AC	F0	RS1	RS0	OV	_	P

Cy (Carry): Carry flag

AC (Auxiliary Carry): Auxiliary carry flag

F0 (Flag): Available to the user for general purpose

RS1、 RS0： Register Bank selector

OV (Overflow) : Overflow flag

P (Parity): Parity flag. Set/cleared by hardware each instruction cycle to indicate an odd/even number of 1 bits in the accumulator.

Instructions that affect flag bits

P54

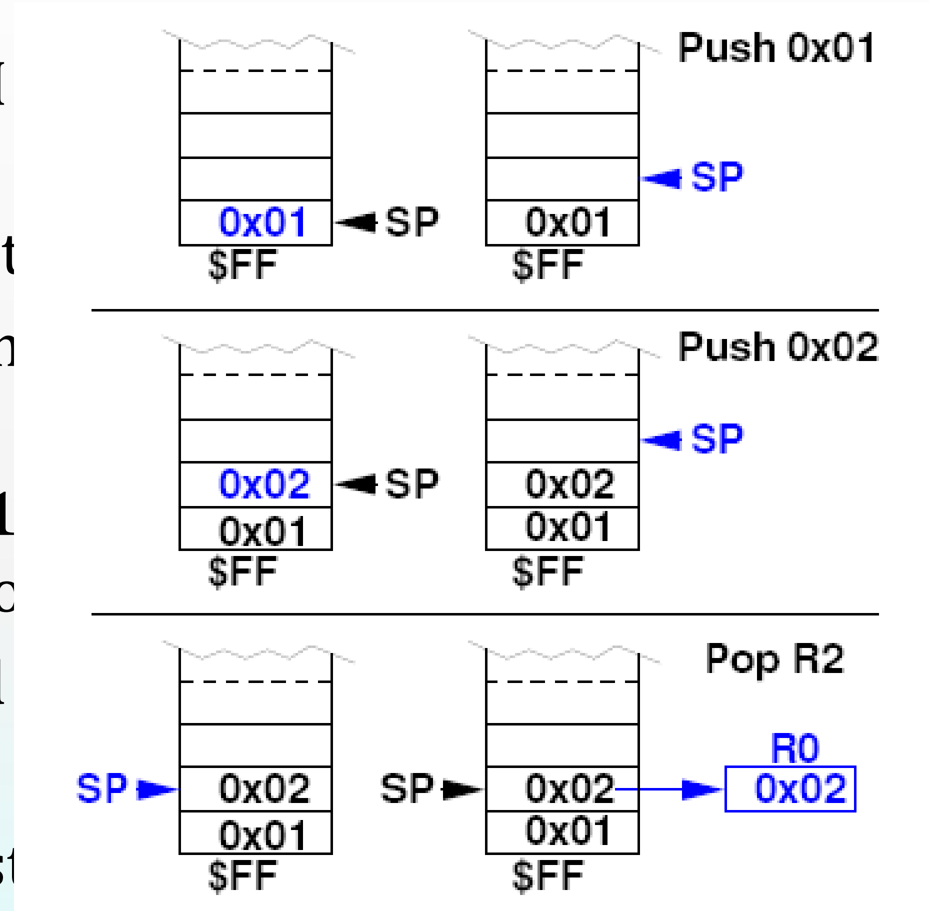
Instruction	CY	OV	AC
ADD	X	X	X
ADDC	X	X	X
SUBB	X	X	X
MUL	0	X	
DIV	0	X	
DA	X		
RRC	X		
RLC	X		
SETB C	1		
CLR C	0		
CPL C	X		
ANL C, bit	X		
ANL C, /bit	X		
ORL C, bit	X		
ORL C, /bit	X		
MOV C, bit	X		
CJNE	X		

Stack

- The stack is a section of RAM
  information temporarily

  This information could be dat
- The register used to access th
  pointer) register

  The stack pointer in the 8051
  means that it can take value c
  - When the 8051 is powered
    value 07
  - RAM location 08 is the first
    stack by the 8051




```
PUSH byte   ;increment stack pointer,
            ;move byte on stack
POP byte    ;move from stack to byte,
            ;decrement stack pointer
```

PC

- PC (program counter) points to the address of the next instruction to be executed

  As the CPU fetches the opcode from the program ROM, PC is increasing to point to the next instruction

- PC is 16 bits wide which means that it can access program addresses 0000 to FFFFH, a total of 64K bytes of code

- All 8051 members start at memory address 0000 when they're powered up

  PC has the value of 0000

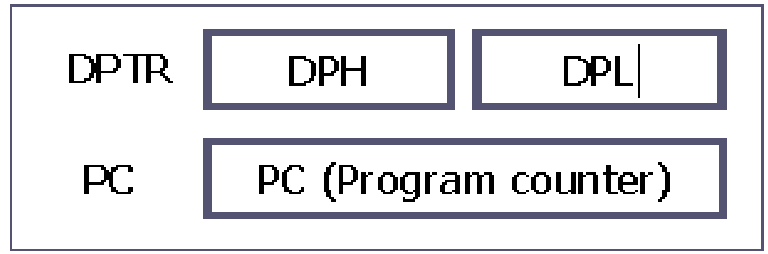
  The first opcode is burned into ROM address 0000H,

  since this is where the 8051 looks for the first

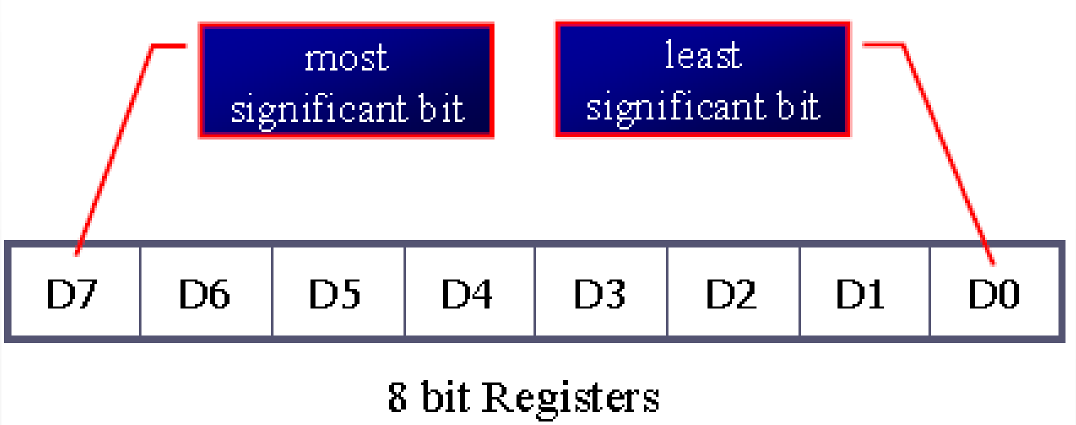
  instruction when it is booted

  We achieve this by the ORG statement in the source program

- DPTR (Data Pointer) is used to store16 bits address when the CPU access external 64 KB RAM



8051 Registers



- **Register are used to store information temporarily, while the information could be:**

  a byte of data to be processed, or

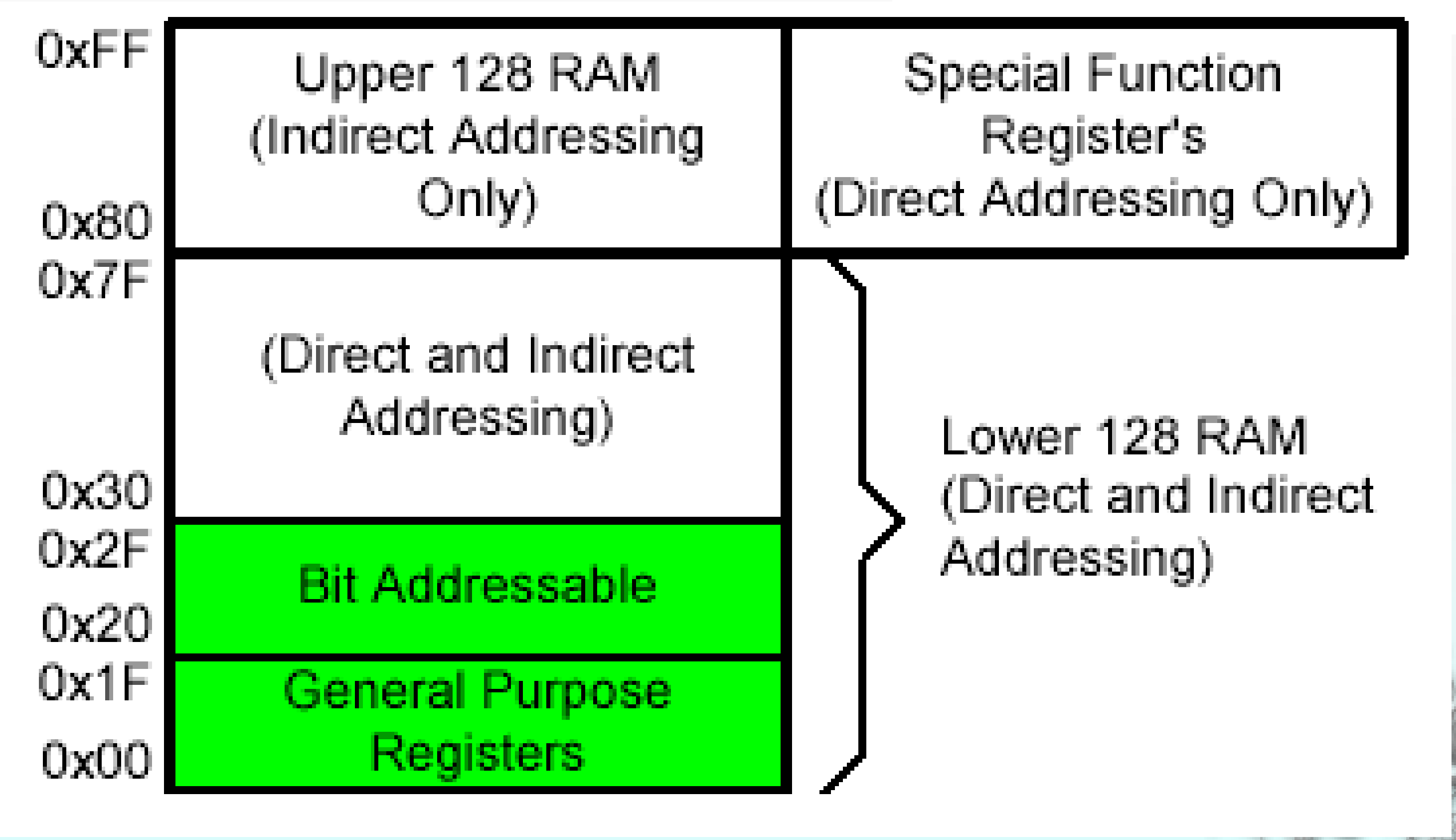
  an address pointing to the data to be fetched

- **The vast majority of 8051 register are 8-bit registers**

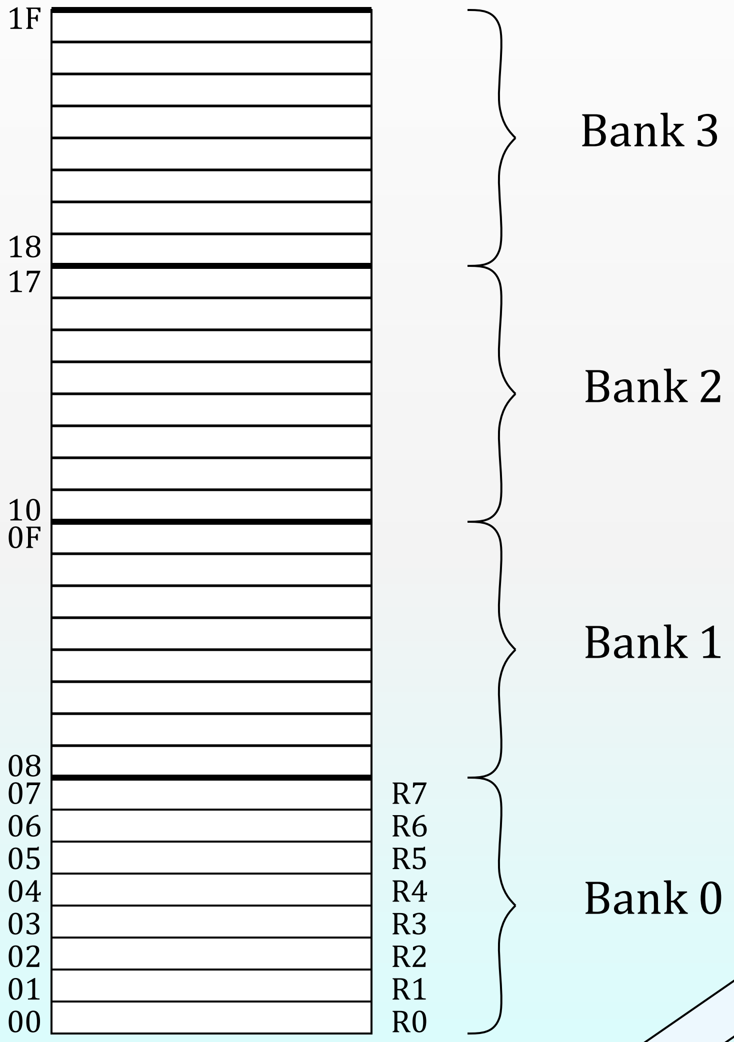
  There is only one data type, 8 bits, any data larger than 8 bits must be broken into 8-bit chunks before it is processed

RAM Memory Space Allocation

Internal RAM



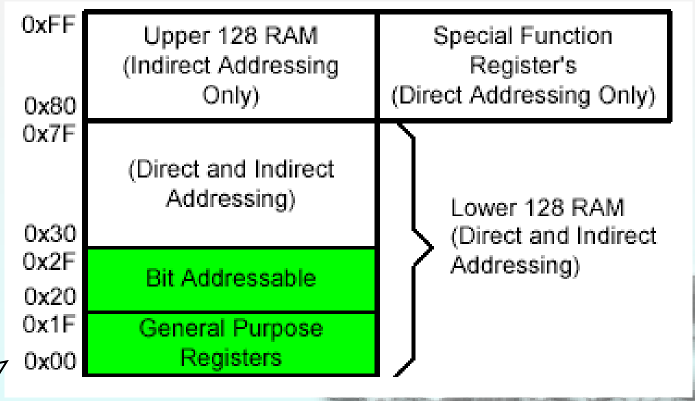
2.2 8051 Registers Banks



Four Register Banks
Each bank has R0-R7

We can switch to other banks by use of the PSW register

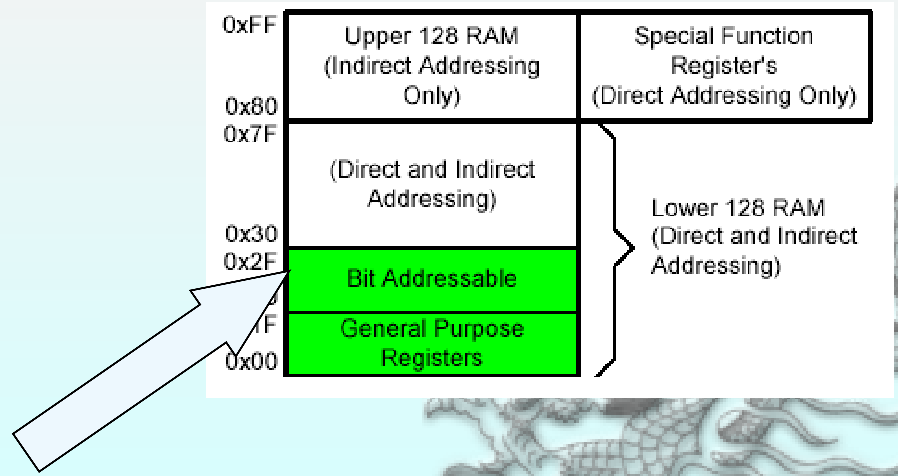
Register bank 0 is the default when 8051 is powered up.



Bit Addressable Memory

2F	7F							78
2E								
2D								
2C								
2B								
2A								
29								
28								
27								
26								
25								
24								
23						1A		
22								10
21	0F							08
20	07	06	05	04	03	02	01	00

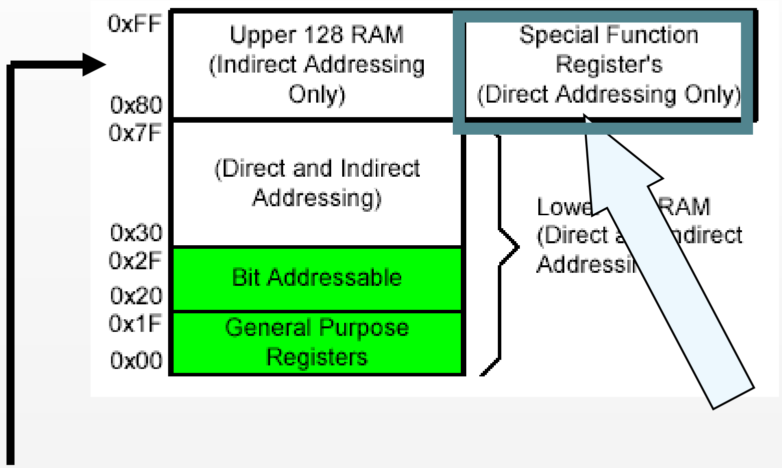
20h – 2Fh (16 locations X 8-bits = 128 bits)



Special Function Registers

DATA registers

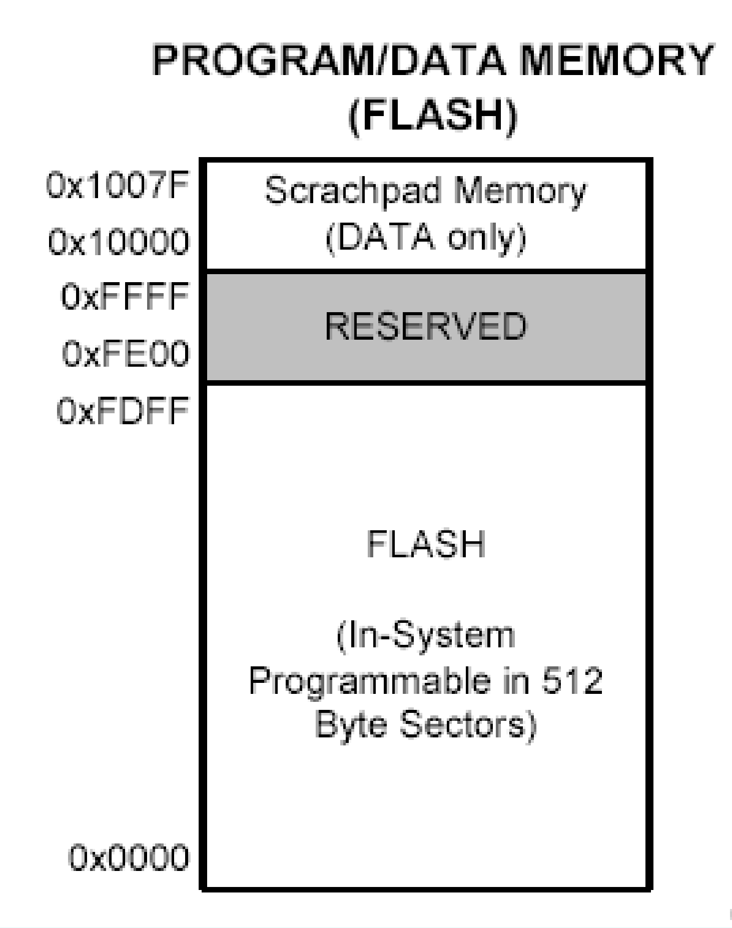
CONTROL registers

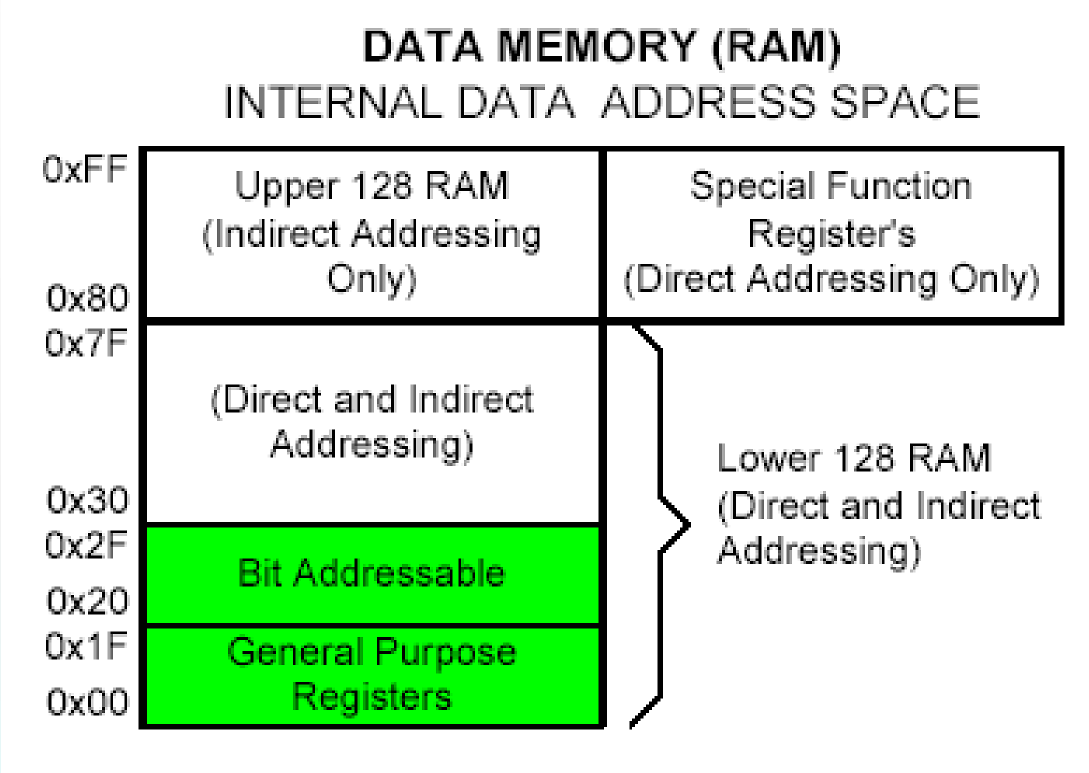


Addresses 80h – FFh

Direct Addressing used to access SPRs

On-Chip Memory: Program/Data





Data Transfer Instructions

--- MOV Instructions

MOV destination, source ; copy source to dest.

The instruction tells the CPU to move (in reality, COPY) the source operand to the destination operand.

“#” signifies that it is a value

6 basic types:

```
MOV A,#55H      ;load value 55H into reg. A
MOV R0,A        ;copy contents of A into R0
                ;(now A=R0=55H)
MOV R1,A        ;copy contents of A into R1
                ;(now A=R0=R1=55H)
MOV R2,A        ;copy contents of A into R2
                ;(now A=R0=R1=R2=55H)
MOV R3,#95H     ;load value 95H into R3
                ;(now R3=95H)
MOV A,R3        ;copy contents of R3 into A
                ;now A=R3=95H
```

Data Transfer Instructions

Notes on programming

- Value (proceeded with #) can be loaded directly to registers A, B, or R0 – R7

  MOV A, #23H

  MOV R5, #0F9H

If it's not preceded with #, it means to load from a memory location

- If values 0 to F moved i ... er, the rest of the bits are assumed all ze

  "MOV A, #5", the result ... A= 00000101 in binary

Add a 0 to indicate that F is a hex number and not a letter

- Moving a value that is too large into a register will cause an error

  MOV A, #7F2H ; ILLEGAL: 7F2H>8 bits (FFH)

Other Data Transfer Instructions

- Stack instructions

```
PUSH byte
POP byte
```

- Exchange instructions

```
XCH A, byte  ;exchange accumulator and
             ;byte
XCHD A, byte ;exchange low nibbles of
             ;accumulator and byte
```

Register A as the destination

MOV	A， Rn	； (A)←（Rn）
MOV	A， direct	； (A)←（direct）
MOV	A， @Ri	； (A)←（（Ri））
MOV	A， #data	； (A)←data

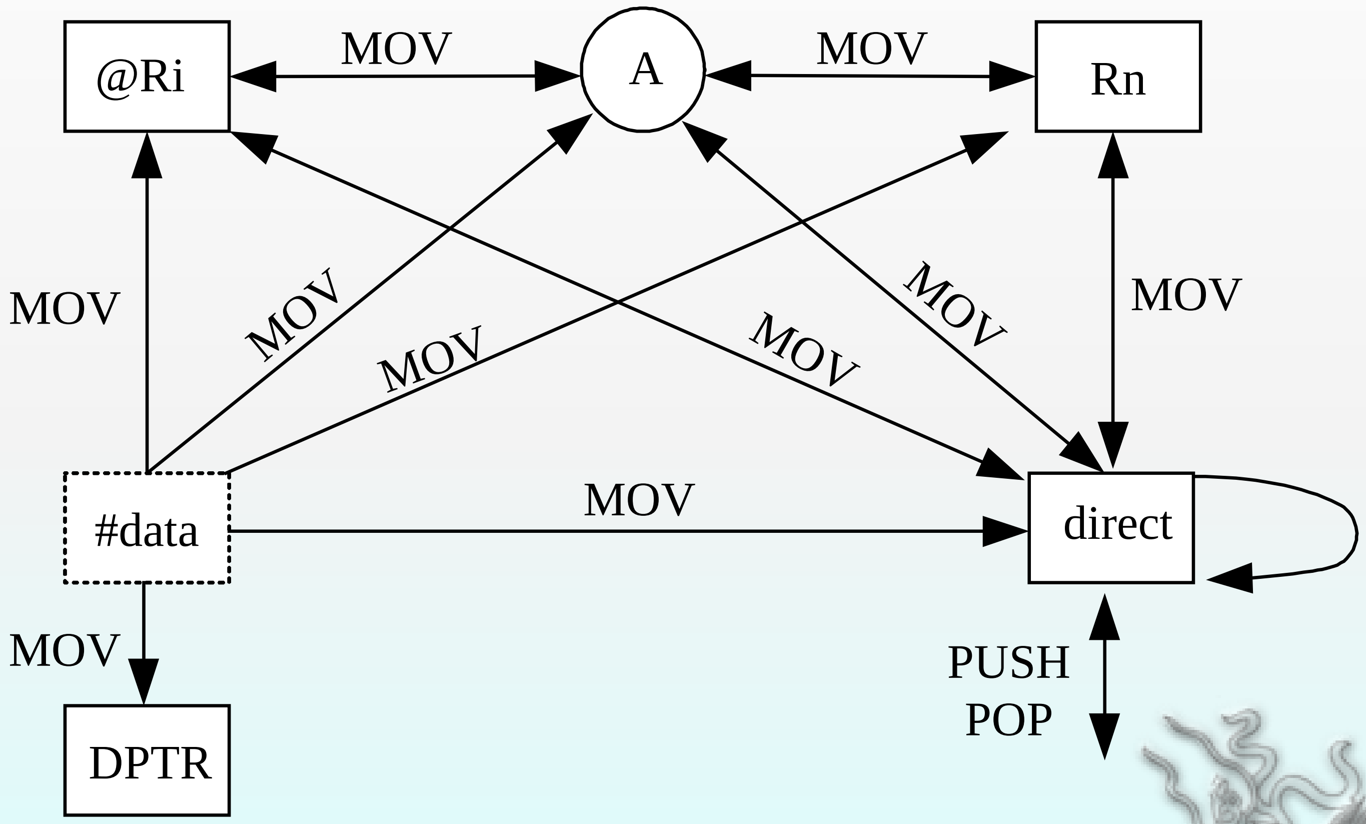
Rn as the destination

MOV	Rn， A	； (Rn)←（A）
MOV	Rn， direct	； (Rn)←（direct）
MOV	Rn， #data	； (Rn)←data

Internal RAM or SFR as the destination

MOV	direct， A	; (direct)←(A)
MOV	direct， Rn	; (direct)←(Rn)
MOV	direct1， direct2	; (direct) 1 ←(direct2)
MOV	direct， @Ri	; (direct)←((Ri))
MOV	direct， #data	; (direct)←#data
MOV	@Ri， A	; ((Ri))←(A)
MOV	@Ri， direct	; ((Ri))←(direct)
MOV	@Ri， #data	; ((Ri))←data
MOV	DPTR， #data16	; DPTR←data16

Internal data transfer instructions



ADD Instruction

ADD A, source ;ADD the source operand
;to the accumulator

- The ADD instruction tells the CPU to add the source byte to register A and put the result in register A
- Source operand can be either a register or immediate data, but the destination must always be register A

```
MOV   A, #25H      ;load 25H into A
MOV   R2, #34H     ;load 34H into R2
ADD   A, R2        ;add R2 to Accumulator
                   ;(A = A + R2)
```

There are always many ways to write the same program, depending on the registers used

```
MOV   A, #25H      ;load one operand
                   ;into A (A=25H)
ADD   A, #34H      ;add the second
                   ;operand 34H to A
```

Chapter 3
JUMP, LOOP and CALL Instructions

Looping

- Repeating a sequence of instructions a certain number of times is called a loop

Loop action is performed by

DJNZ reg , Label

The register is decremented

If it is not zero, it jumps to the target address referred to by the label

Prior to the start of loop the register is loaded with the counter for the number of repetitions

Counter can be R0 – R7 or RAM location

A loop can be repeated a maximum of 255 times, if R2 is FFH

```
;This program adds value 3 to the ACC ten times
          MOV   A, #0          ;A=0, clear ACC
          MOV   R2, #10        ;load counter R2=10
AGAIN:    ADD   A, #03         ;add 03 to ACC
          DJNZ  R2,AGAIN       ;repeat until R2=0,10 times
          MOV   R5,A           ;save A in R5
```

Nested Loop

- If we want to repeat an action more times than 256, we use a loop inside a loop, which is called nested loop
  - *We use multiple registers to hold the count*

Write a program to (a) load the accumulator with the value 55H, and (b) complement the ACC 700 times

```
        MOV   A,#55H        ;A=55H
        MOV   R3,#10        ;R3=10, outer loop count
NEXT:   MOV   R2,#70        ;R2=70, inner loop count
AGAIN:  CPL   A             ;complement A register
        DJNZ  R2,AGAIN      ;repeat it 70 times
        DJNZ  R3,NEXT
```

Conditional Jumps

- Jump only if a certain condition is met

  JZ label ;jump if A=0

```
        MOV   A,R0      ;A=R0
        JZ    OVER      ;jump if A = 0
        MOV   A,R1      ;A=R1
        JZ    OVER      ;jump if A = 0
        ...
OVER:
```

Can be used only for register A, not any other register

Determine if R5 contains the value 0. If so, put 55H in it.

```
        MOV  A,R5       ;copy R5 to A
        JNZ  NEXT       ;jump if A is not zero
        MOV  R5,#55H
NEXT:   ...
```

JNC label ;jump if no carry, CY=0

- If CY = 0, the CPU starts to fetch and execute instruction from the address of the label
- If CY = 1, it will not jump but will execute the next instruction below JNC

Find the sum of the values 79H, F5H, E2H. Put the sum in registers R0 (low byte) and R5 (high byte).

```
          MOV   A,#0          ;A=0
          MOV   R5,A          ;clear R5
          ADD   A,#79H        ;A=0+79H=79H
;         JNC   N_1           ;if CY=0, add next number
;         INC   R5            ;if CY=1, increment R5
N_1:      ADD   A,#0F5H       ;A=79+F5=6E and CY=1
          JNC   N_2           ;jump if CY=0
          INC   R5            ;if CY=1,increment R5 (R5=1)
N_2:      ADD   A,#0E2H       ;A=6E+E2=50 and CY=1
          JNC   OVER          ;jump if CY=0
          INC   R5            ;if CY=1, increment 5
OVER:     MOV   R0,A          ;now R0=50H, and R5=02
```

MOV R5,#0

Conditional Jumps

- 8051 conditional jump instructions

Instructions	Actions
JZ	Jump if A = 0
JNZ	Jump if A ≠ 0
DJNZ	Decrement and Jump if A ≠ 0
CJNE A,byte	Jump if A ≠ byte
CJNE reg,#data	Jump if byte ≠ #data
JC	Jump if CY = 1
JNC	Jump if CY = 0
JB	Jump if bit = 1
JNB	Jump if bit = 0
JBC	Jump if bit = 1 and clear bit

- All conditional jumps are short jumps

The address of the target must within -128 to +127 bytes of the contents of PC

Unconditional Jumps

- The unconditional jump is a jump in which control is transferred unconditionally to the target location

  LJMP (long jump)

  3-byte instruction

  First byte is the opcode

  Second and third bytes represent the 16-bit target address

  –- Any memory location from 0000 to FFFFH

  SJMP (short jump)

  2-byte instruction

  First byte is the opcode

  Second byte is the relative target address

  –- 00 to FFH (forward +127 and backward -128 bytes from the current PC)

Calculating Short Jump Address

- To calculate the target address of a short jump (SJMP, JNC, JZ, DJNZ, etc.)
  - *The second byte*
  - *immediately bel*
- If the target addres
  from the address b
  - *The assembler v*
  - *is out of range*

Line	PC	Opcode		Mnemonic	Operand
01	0000			ORG	0000
02	0000	7800		MOV	R0,#0
03	0002	7455		MOV	A,#55H
04	0004	6003		JZ	NEXT
05	0006	08		INC	R0
06	0007	04	AGAIN:	INC	A
07	0008	04		INC	A
08	0009	2477	NEXT:	ADD	A,#77H
09	000B	5005		JNC	OVER
10	000D	E4		CLR	A
11	000E	F8		MOV	R0,A
12	000F	F9		MOV	R1,A
13	0010	FA		MOV	R2,A
14	0011	FB		MOV	R3,A
15	0012	2B	OVER:	ADD	A,R3
16	0013	50F2		JNC	AGAIN
17	0015	80FE	HERE:	SJMP	HERE
18	0017				END

CALL INSTRUCTIONS

- Call instruction is used to call subroutine

  Subroutines are often used to perform tasks that need to be performed frequently

  This makes a program more structured in addition to saving memory space

LCALL (long call)

3-byte instruction

First byte is the opcode

Second and third bytes are used for address of target subroutine

- Subroutine is located anywhere within 64K byte address space

ACALL (absolute call)

2-byte instruction

11 bits are used for address within 2K-byte range

LCALL(讲到此处)

- When a subroutine is called, control is transferred to that subroutine, the processor
  - *Saves the address of the instruction immediately on the stack below the LCALL*
  - *Begins to fetch instructions form the new location*

- After finishing execution of the subroutine
  - *The instruction RET transfers control back to the caller*
    - Every subroutine needs RET as the last instruction

```
         ORG 0
BACK:    MOV   A ,#55H      ;load A with 5
         MOV   P1, A        ;send 55H to
         LCALL  DELAY       ;time delay
         MOV   A, #0AAH     ;load A with AA (in hex)
         MOV   P1, A        ;send AAH to port 1
         LCALL  DELAY
         SJMP  BACK         ;keep doing this indefinitely

;---------- this is delay subroutine
         ORG 300H           ;put DELAY at address 300H
DELAY:   MOV   R5, #0FFH ;R5=255 (FF in hex), counter
AGAIN:   DJNZ  R5, AGAIN ;stay here until R5 become 0
         RET                ;return to caller (when R5 =0)
         END
```

Upon executing "LCALL DELAY", the address of instruction below it, "MOV A,#0AAH" is pushed onto stack, and the 8051 starts to execute at 300H.

The counter R5 is set to FFH; so loop is repeated 255 times.

When R5 becomes 0, control falls to the RET which pops the address from the stack into the PC and resumes executing the instructions after the CALL.

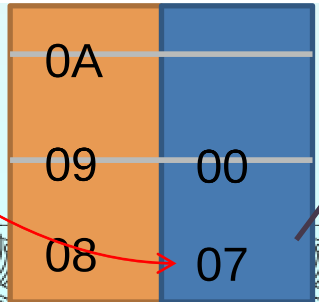
The amount of time delay depends on the frequency of the 8051

CALL Instruction and Stack

```
0000                      ORG  0
0000  7455     BACK:      MOV  A, #55H        ;load A with 55H
0002  F590                MOV  P1, A          ;send 55H to p1
0004  120300              LCALL  DELAY        ;time delay
0007  74AA                MOV  A, #0AAH       ;load A with AAH
0009  F590                MOV  P1, A          ;send AAH to p1
000B  120300              LCALL  DELAY
000E  80F0                SJMP  BACK          ;keep doing this
0010
0010   ;-------this is the delay subroutine------
0300                      ORG  300H
0300            DELAY:
0300  7DFF                MOV R5, #0FFH       ;R5=255
0302  DDFE      AGAIN:    DJNZ R5, AGAIN      ;stay here
0304  22                  RET                 ;return to caller
0305                      END                 ;end of asm file
```

Stack frame after the first LCALL

SP=09



Low byte goes first and high byte is last.

Calling Subroutines

```
;MAIN program calling subroutines
        ORG   0
MAIN:   LCALL             SUBR_1
        LCALL             SUBR_2
        LCALL             SUBR_3

HERE:   SJMP              HERE
;-----------end of MAIN

SUBR_1: ...
        ...
        RET
;-----------end of subroutine1

SUBR_2: ...
        ...
        RET
;-----------end of subroutine2

SUBR_3: ...
        ...
        RET
;-----------end of subroutine3
        END               ;end of the asm file
```

It is common to have one main program and many subroutines that are called from the main program

This allows you to make each subroutine into a separate module

- Each module can be tested separately and then brought together with main program
- In a large program, the module can be assigned to different programmers

ACALL

- The only difference between **ACALL** and **LCALL** is
  The target address for LCALL can be anywhere within the 64K byte address.
  The target address of ACALL must be within a 2K- byte range.
- The use of ACALL instead of LCALL can save a number of bytes of program ROM space

```
        ORG  0
BACK:   MOV   A, #55H      ;load A with 55H
        MOV  P1, A         ;send 55H to port 1
        LCALL   DELAY      ;time delay
        MOV   A, #0AAH     ;load A with AA (in hex)
        MOV  P1,A          ;send AAH to port 1
        LCALL  DELAY
        SJMP  BACK         ;keep doing this indefinitely
        ...
        END                ;end of asm file
```

A rewritten program which is more efficiently

```
        ORG 0
        MOV  A, #55H       ;load A with 55H
BACK:   MOV  P1,A          ;send 55H to port 1
        ACALL DELAY        ;time delay
        CPL  A             ;complement reg A
        SJMP  BACK         ;keep doing this indefinitely
        ...
        END
```

Time Delay for Various 8051 Chips

- CPU executing an instruction takes a certain number of clock cycles
  These are referred as to as machine cycles
- The length of machine cycle depends on the frequency of the crystal oscillator connected to 8051
- In original 8051, one machine cycle lasts 12 oscillator periods

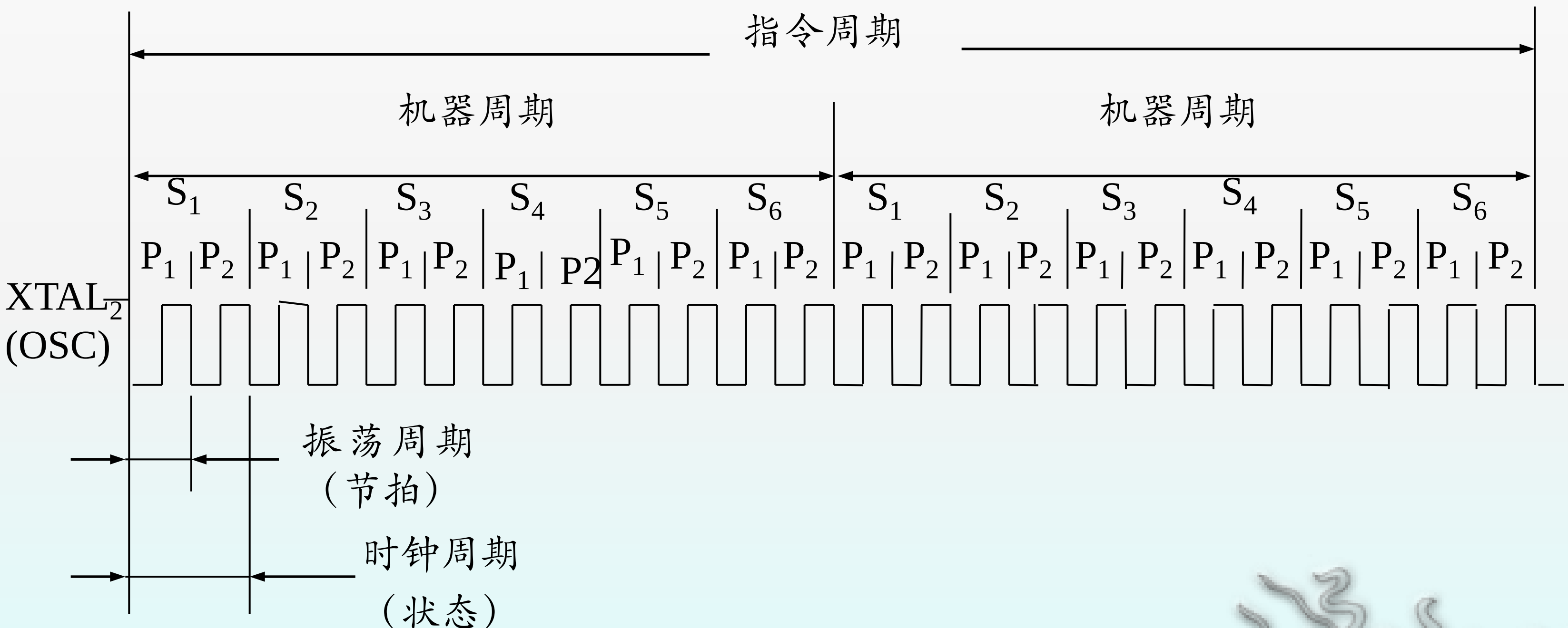
Find the period of the machine cycle for 11.0592 MHz crystal frequency

Solution:

11.0592/12 = 921.6 kHz;

machine cycle is 1/921.6 kHz = 1.085μs

Timing



指令周期
机器周期
机器周期
S1 S2 S3 S4 S5 S6 S1 S2 S3 S4 S5 S6
P1 P2 P1 P2 P1 P2 P1 P2 P1 P2 P1 P2 P1 P2 P1 P2 P1 P2 P1 P2 P1 P2 P1 P2
XTAL2
(OSC)
振荡周期
（节拍）
时钟周期
（状态）

Time Delay for Various 8051 Chips

For 8051 system of 11.0592 MHz, find how long it takes to execute each instruction.

(a) MOV R3, #55 (b) DEC R3 (c) DJNZ R2 target

(d) LJMP (e) SJMP (f) NOP (g) MUL AB

Solution:

	Machine cycles	***Time to execute***
(a)	1	1x1.085μs = 1.085μs
(b)	1	1x1.085μs = 1.085μs
(c)	2	2x1.085μs = 2.17μs
(d)	2	2x1.085μs = 2.17μs
(e)	2	2x1.085μs = 2.17μs
(f)	1	1x1.085μs = 1.085μs
(g)	4	4x1.085μs = 4.34μs

Delay Calculation

Find the size of the delay in following program, if the crystal frequency is 11.0592MHz.

```
            MOV A,#55H
AGAIN:      MOV P1,A
            ACALL DELAY
            CPL A
            SJMP AGAIN
;---time delay-------
 DELAY:     MOV R3,#200
HERE:       DJNZ R3,HERE
            RET
```

Solution:

		Machine cycle
DELAY:	MOV R3, #200	1
HERE:	DJNZ R3, HERE	2
	RET	2

Therefore, [(200x2)+1+2]x1.085μs = 436.255μs.

Increasing Delay Using NOP

Find the size of the delay in following program, if the crystal frequency is 11.0592MHz.

			Machine Cycle
DELAY:	MOV	R3, #250	1
HERE:	NOP		1
	NOP		1
	NOP		1
	NOP		1
	DJNZ	R3, HERE	2
	RET		2

Solution:

The time delay inside HERE loop is

[250(1+1+1+1+2)]x1.085μs = 1627.5μs.

Adding the two instructions outside loop we have

1627.5μs + 3 x 1.085μs = 1630.755μs

Large Delay Using Nested Loop

Find the size of the delay in following program, if the crystal frequency is 11.0592MHz.

		Machine Cycle
DELAY:	MOV R2, #200	1
AGAIN:	MOV R3, #250	1
HERE:	NOP	1
	NOP	1
	DJNZ R3, HERE	2
	DJNZ R2, AGAIN	2
	RET	2

Notice in nested loop, as in all other time delay loops, the time is approximate since we have ignored the first and last instructions in the subroutine.

Solution:

For HERE loop, we have (4x250)x1.085μs＝1085μs.

For AGAIN loop repeats HERE loop 200 times, so we have 200x1085μs＝217000μs.

But “MOV R3, #250” and “DJNZ R2, AGAIN” at the start and end of the AGAIN loop add (3x200x1.805)=651μs.

As a result we have 217000+651=217651μs.

Delay Calculation for Other 8051

- **Two factors can affect the accuracy of the delay**

  Crystal frequency

  The duration of the clock period of the machine cycle is a function of this crystal frequency

  8051 design

  The original machine cycle duration was set at 12 clocks

  Advances in both IC technology and CPU design in recent years have made the 1-clock machine cycle a common feature

Clocks per machine cycle for various 8051 versions

Chip/Maker	Clocks per Machine Cycle
AT89C51 Atmel	12
P89C54X2 Philips	6
DS5000 Dallas Semi	4
DS89C420/30/40/50 Dallas Semi	1

Delay Calculation for Other 8051

Find the period of the machine cycle (MC) for various versions of 8051, if XTAL=11.0592 MHz.

(a) AT89C51 (b) P89C54X2 (c) DS5000 (d) DS89C4x0

Solution:

(a) 11.0592MHz/12 = 921.6kHz;
MC is 1/921.6kHz = 1.085μs = 1085ns

(b) 11.0592MHz/6 = 1.8432MHz;
MC is 1/1.8432MHz = 0.5425μs = 542ns

(c) 11.0592MHz/4 = 2.7648MHz ;
MC is 1/2.7648MHz = 0.36μs = 360ns

(d) 11.0592MHz/1 = 11.0592MHz;
MC is 1/11.0592MHz = 0.0904μs = 90ns

Delay Calculation for Other 8051

Instruction	8051	DSC89C4x0
MOV R3,#55	1	2
DEC R3	1	1
DJNZ R2 target	2	4
LJMP	2	3
SJMP	2	3
NOP	1	1
MUL AB	4	9

For an AT8051 and DSC89C4x0 system of 11.0592 MHz, find how long it takes to execute each instruction.

(a) MOV R3,#55 (b) DEC R3 (c) DJNZ R2 target
(d) LJMP (e) SJMP (f) NOP (g) MUL AB

Solution:

	AT8051	***DS89C4x0***
(a)	$1 \times 1085\text{ns} = 1085\text{ns}$	$2 \times 90\text{ns} = 180\text{ns}$
(b)	$1 \times 1085\text{ns} = 1085\text{ns}$	$1 \times 90\text{ns} = 90\text{ns}$
(c)	$2 \times 1085\text{ns} = 2170\text{ns}$	$4 \times 90\text{ns} = 360\text{ns}$
(d)	$2 \times 1085\text{ns} = 2170\text{ns}$	$3 \times 90\text{ns} = 270\text{ns}$
(e)	$2 \times 1085\text{ns} = 2170\text{ns}$	$3 \times 90\text{ns} = 270\text{ns}$
(f)	$1 \times 1085\text{ns} = 1085\text{ns}$	$1 \times 90\text{ns} = 90\text{ns}$
(g)	$4 \times 1085\text{ns} = 4340\text{ns}$	$9 \times 90\text{ns} = 810\text{ns}$

THANK YOU!!