SPECIAL FUNCTION REGISTERS

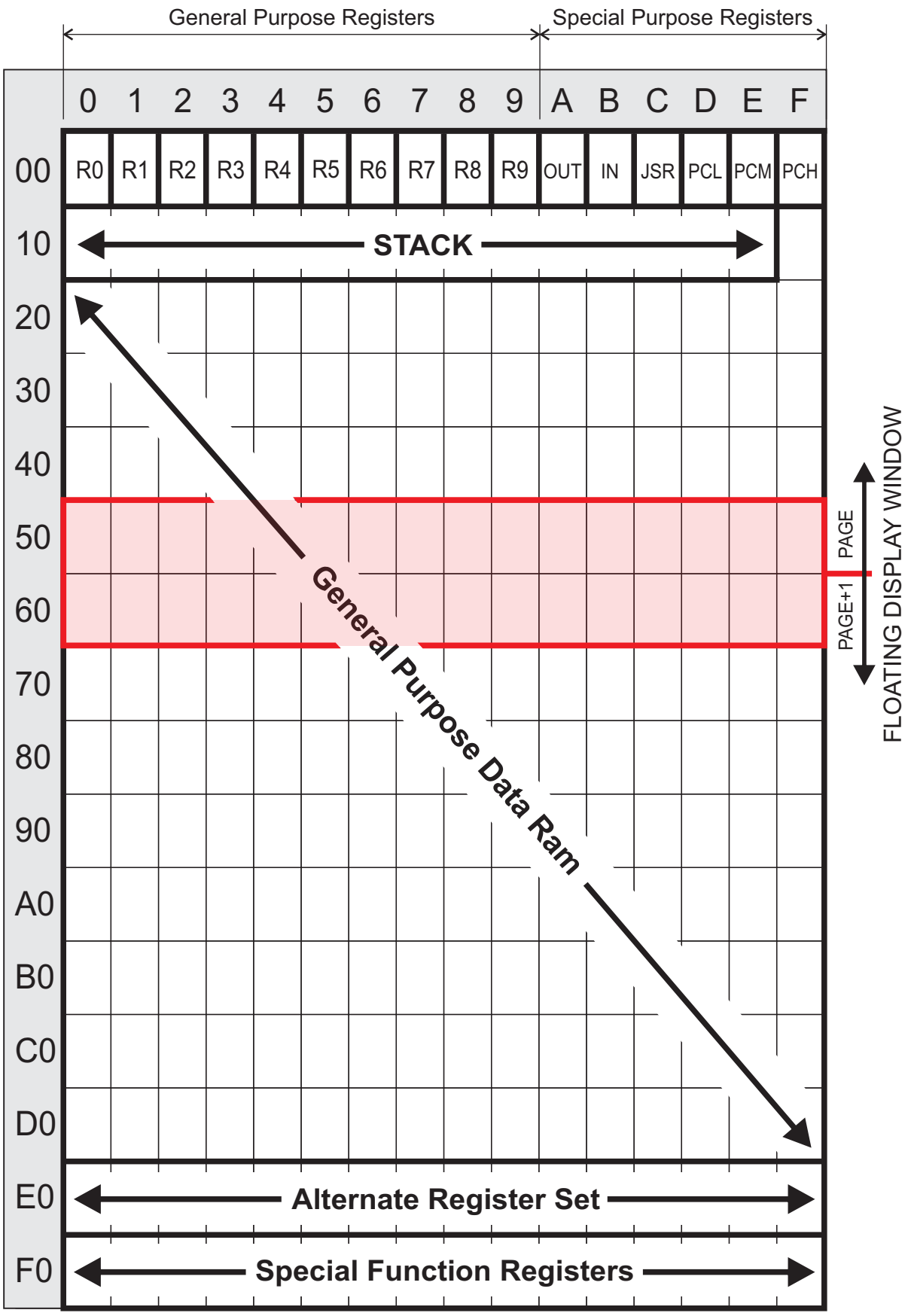
Revision 4a
Nov-03-2022

特殊功能

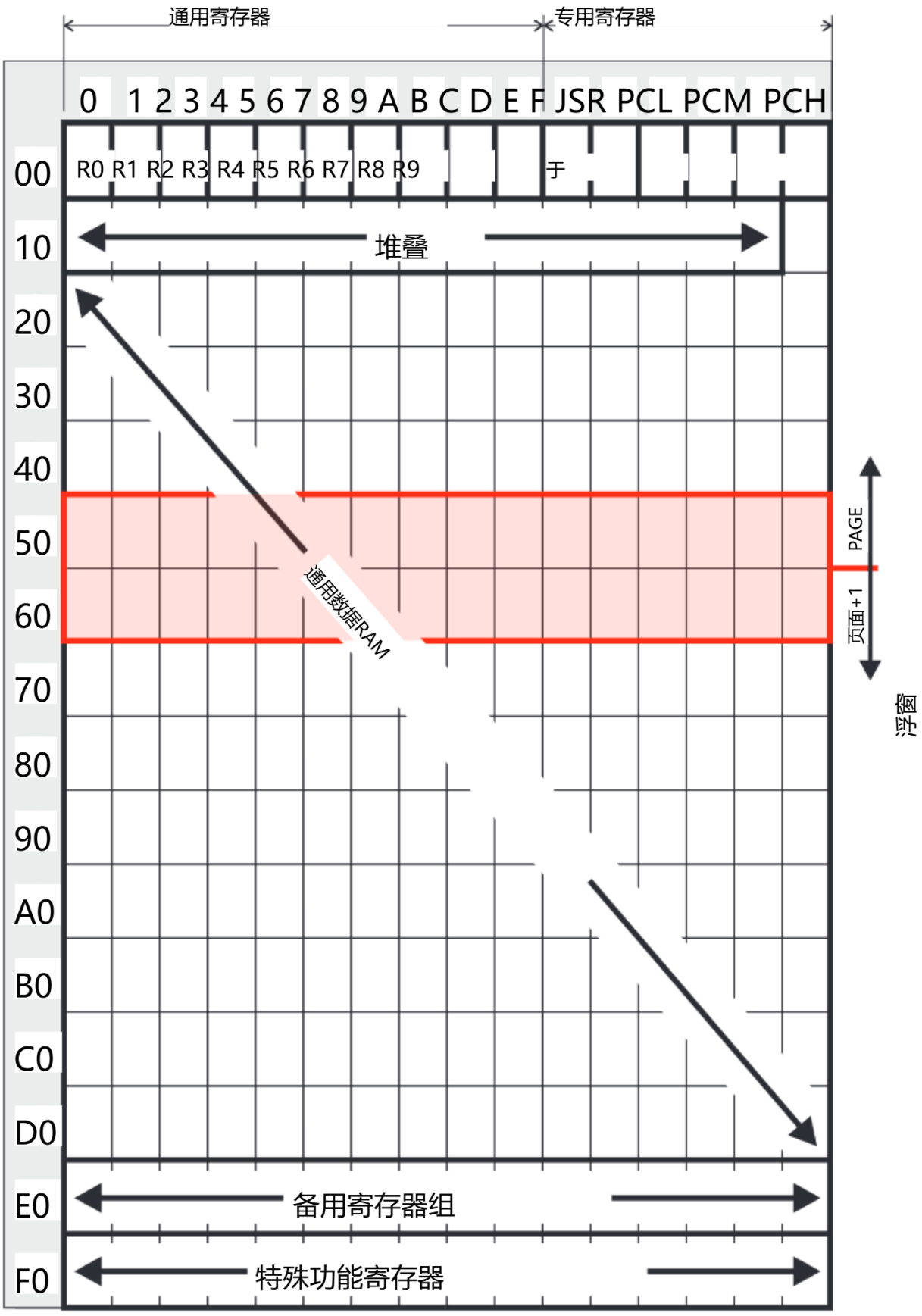
寄存器

修订版4a
2022年11月3日

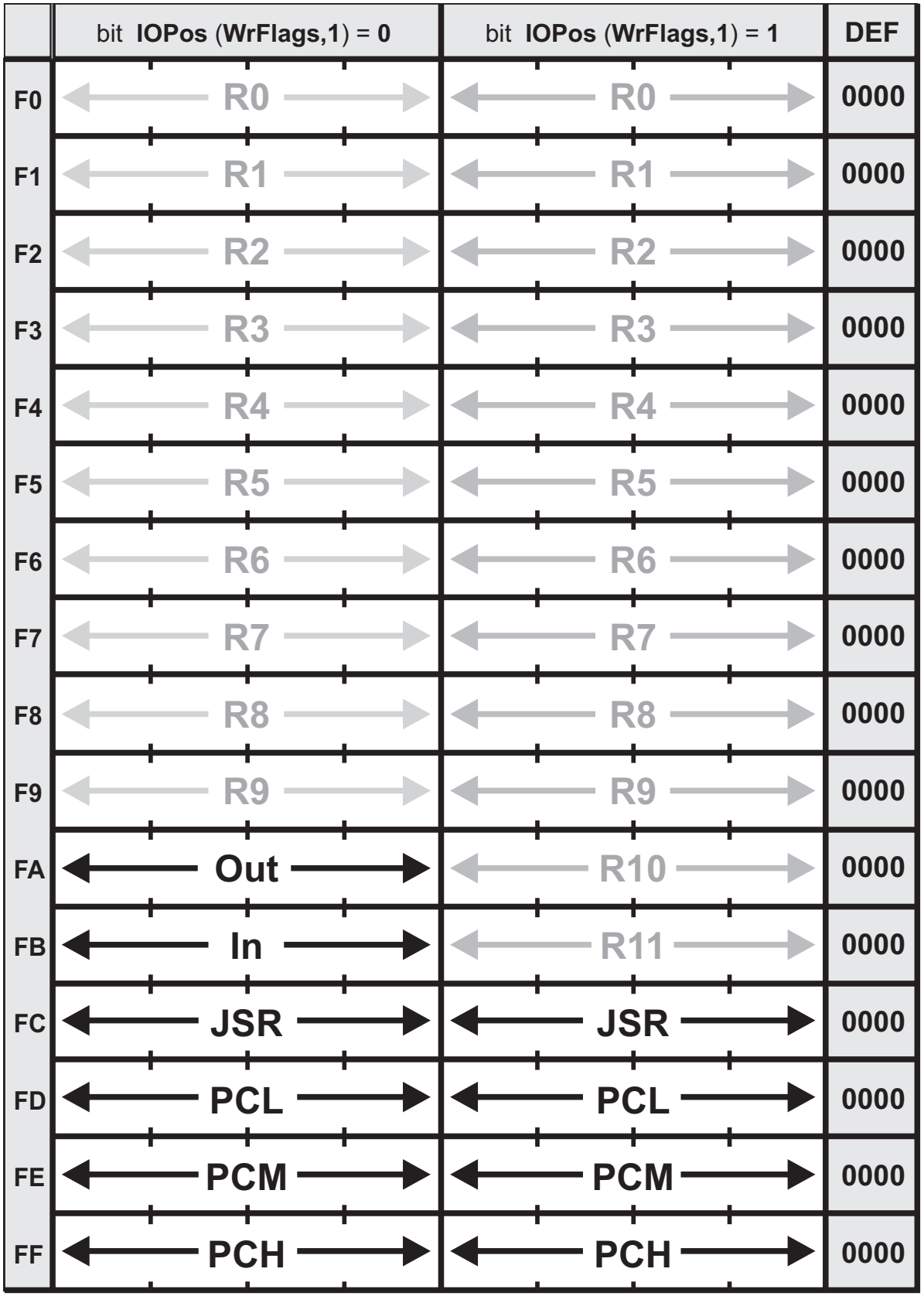
Data Memory Organization



数据存储器组织

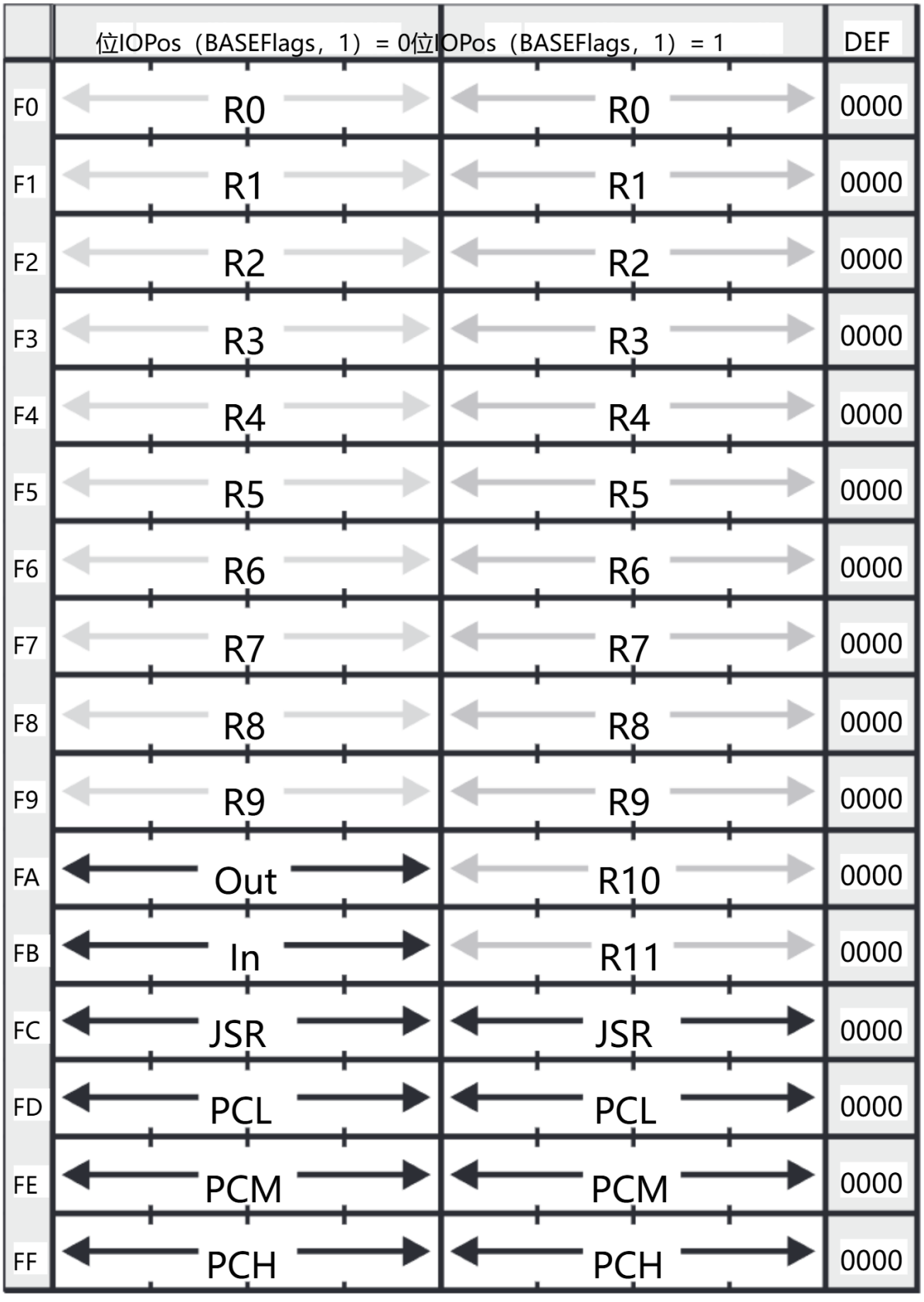

通用寄存器
专用寄存器
0 1 2 3 4 5 6 7 8 9 A B C D E F JSR PCL PCM PCH
00
10
20
30
40
50
60
70
80
90
A0
B0
C0
D0
E0
F0
R0 R1 R2 R3 R4 R5 R6 R7 R8 R9
于
堆叠
通用数据RAM
PAGE
页面+1
浮窗
备用寄存器组
特殊功能寄存器

Special Function Registers on Page 0



Note: Registers R0-R9 (or R0-R11) are general purpose registers

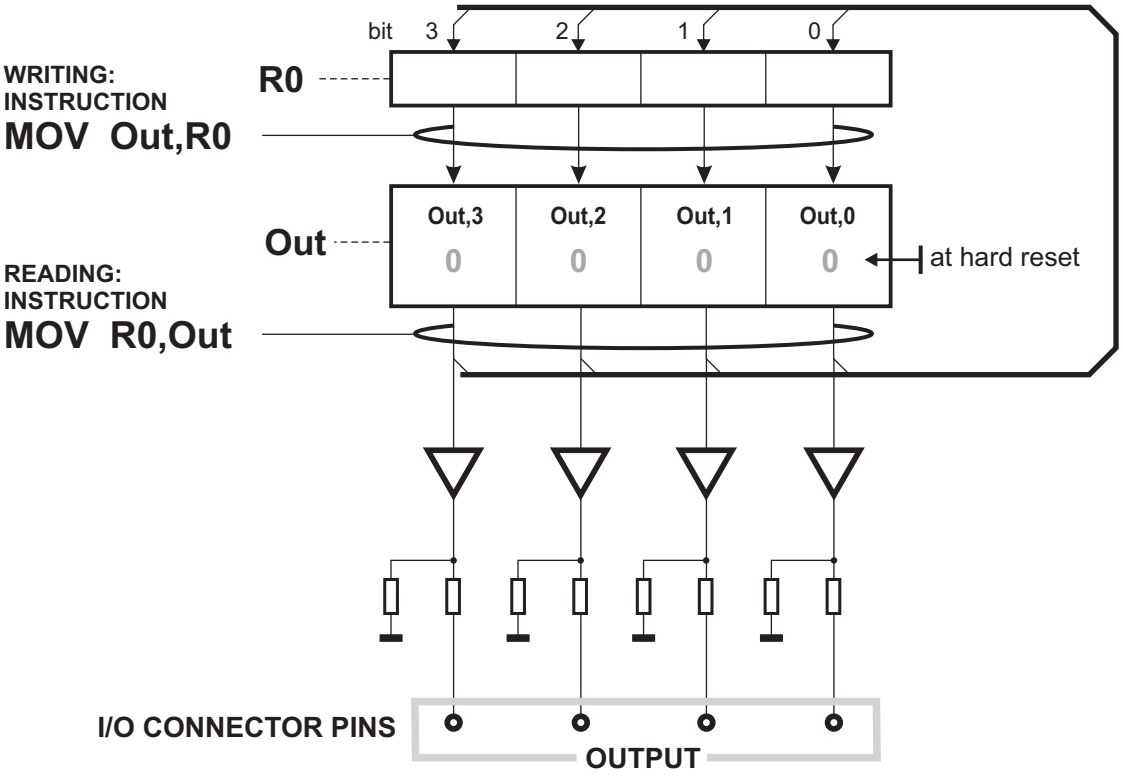
第0页上的特殊功能寄存器



注：寄存器R 0-R9（或R 0-R11）是通用寄存器

Special Function Register SFR0A (Out) 0x0A

Description: Register **Out** is the 4-bit latch which drives output ports, active when the **WrFlag,1** bit (**IOPos**) is cleared.

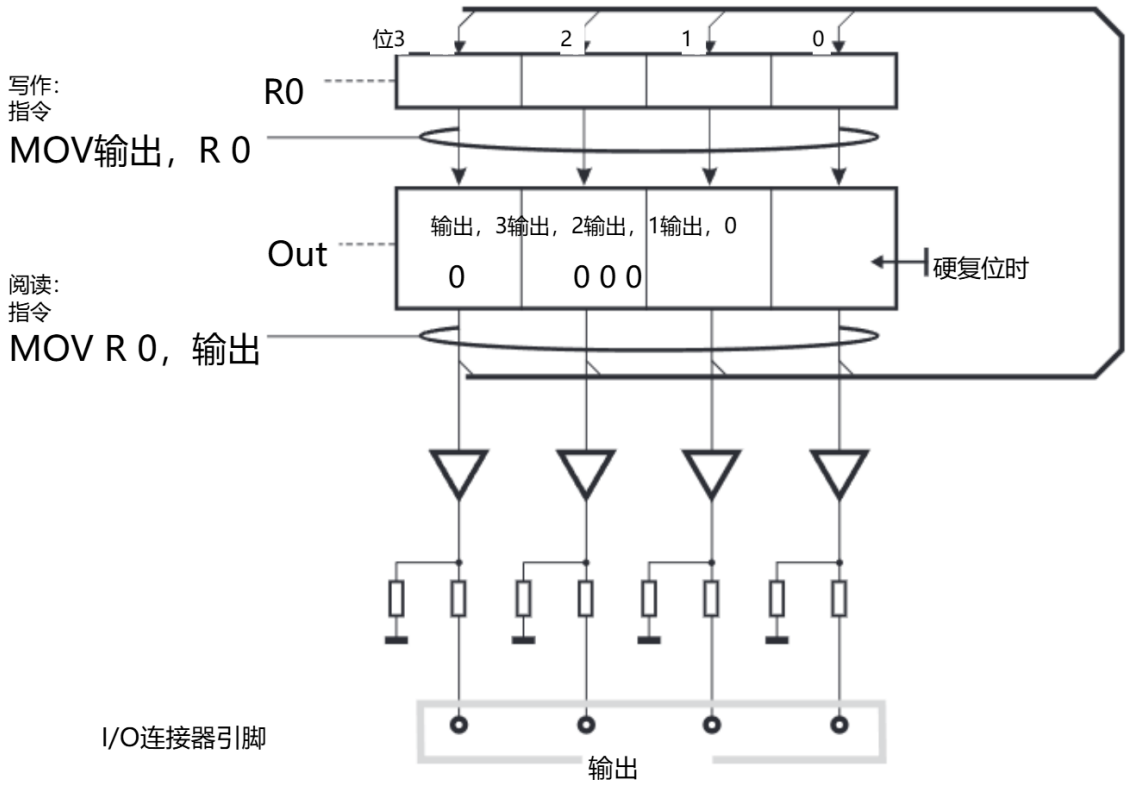


Operation: This register is the output latch. Contents written in the register **Out** will appear on the connector output pins as logic levels.

Note: This register is active only when the **WrFlag,1** bit (**IOPos**) is cleared. Otherwise, this register serves as the General Purpose Register **R10**, and the register **OutB** (address **0xFA**) is active.

特殊功能寄存器SFR0A（输出） 0x0A

产品描述： Register Out是一个4位锁存器，用于驱动输出端口，当1位IOPos（IOPos）清零时，该锁存器处于活动状态。

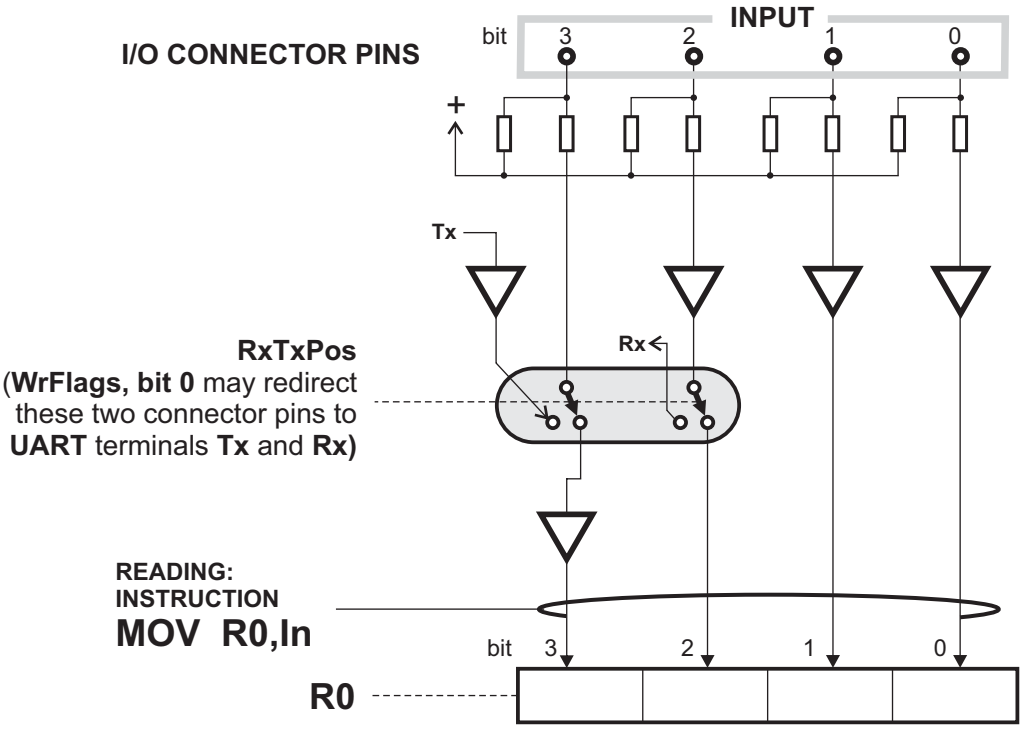


操作：该寄存器是输出锁存器。写入寄存器Out的内容将作为逻辑电平显示在连接器输出引脚上。

注：此寄存器仅在1位IOPos标志清零时有效。
否则，该寄存器用作通用寄存器R10，寄存器OutB（地址0xFA）有效。

Special Function Register SFR0B (In) 0x0B

Description: Register **In** is the input port register, active when the **WrFlag,1** bit (**IOPos**) is cleared.



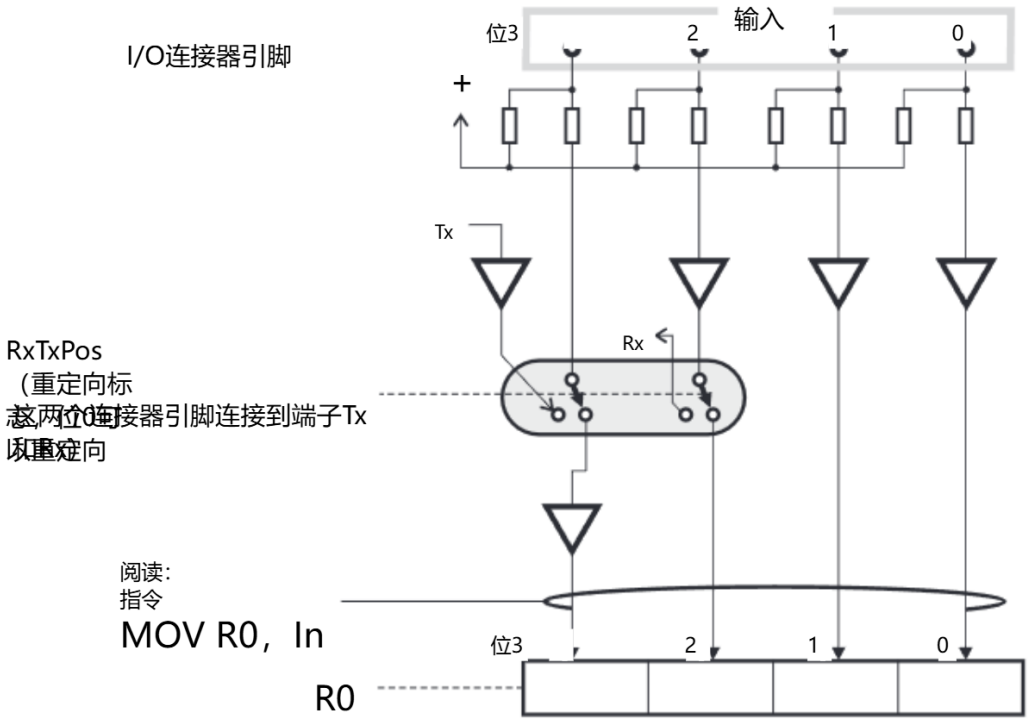
Operation: This register is the input port register. Logic levels on the connector input pins are always transferred to the register **In**.

Note: This register is active only when the **WrFlag,1** (bit **IOPos**) is cleard. Otherwise, this register serves as the General Purpose Register **R11**, and the register **InB** (address **0xFB**) is active.

Note 1: When the register **In** is selected as the input port (when the **WrFlag,1** (bit **IOPos**) is cleared), writing to this register is possible, but the contents will be instantly overwritten. So it makes no sense, except for dummy writes.

Note 2: If the **RxTxPos** (**WrFlags, bit 0**) is set, then bit **3** of the input port (connector pin **2**) is output, not input!

产品描述： 寄存器In是输入端口寄存器，在清零1位IOPos标志时有效。



操作：此寄存器是输入端口寄存器。连接器输入引脚上的逻辑电平始终传输到寄存器In。

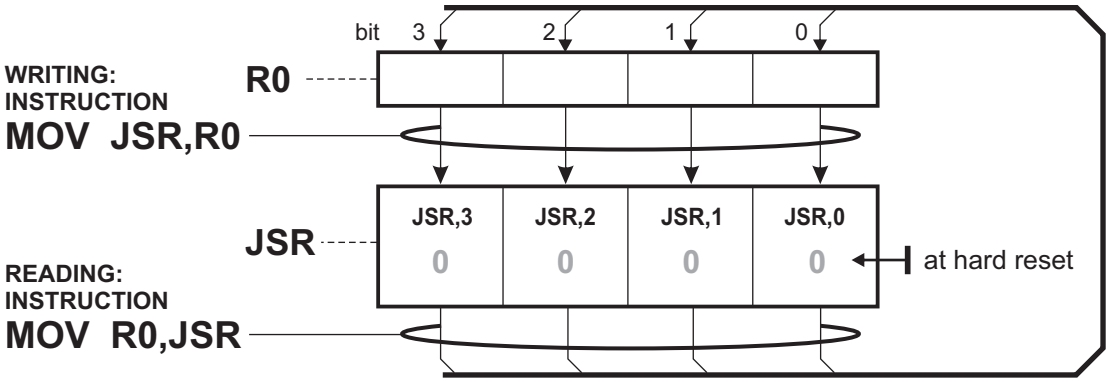
注意：仅当WrFlag，1（位IOPos）被清除时，该寄存器才有效。否则，该寄存器用作通用寄存器R11，寄存器InB（地址0xFB）有效。

注一：当选择寄存器In作为输入端口时（当清除了RISK Flag，1（位IOPos）时），可以写入该寄存器，但内容将立即被覆盖。
所以它没有意义，除了虚拟写入。

注2：如果设置了RxTxPos，则输出寄存器的位3 (连接器引脚2)
是输出，不是输入！

Special Function Register SFR0C (JSR) 0x0C

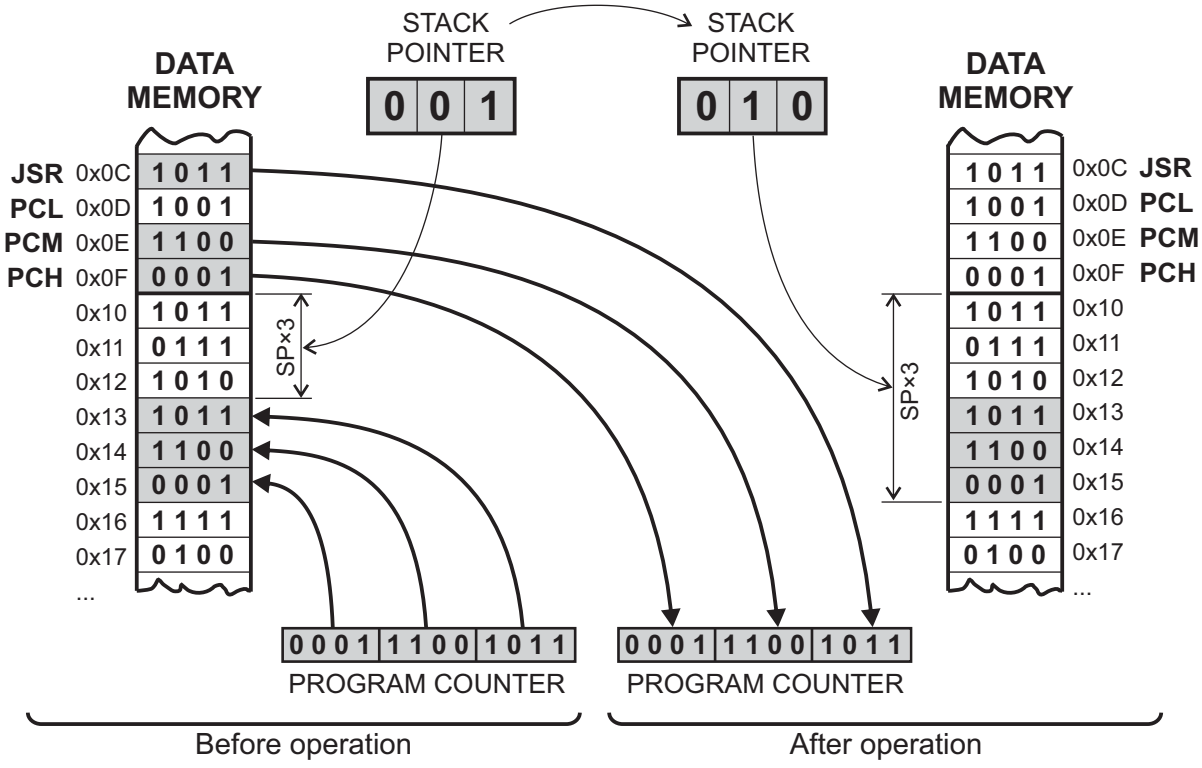
Description: Register **JSR** is the low-nibble address register for a subroutine call.



Operation: Reading from this register is a simple read operation, but when the program performs writing to this register, the subroutine call is automatically initiated. First, the **Program Counter** contents are moved to **Stack,** in the **Data Memory** addressed by the **Stack** register, on the three locations starting from the address **0x10+([SP]×3)**. After that, **SP** register is incremented by one, and the contents of locations **JSR** (low nibble), **PCM** and **PCH** (high nibble) are loaded to **Program Counter**.

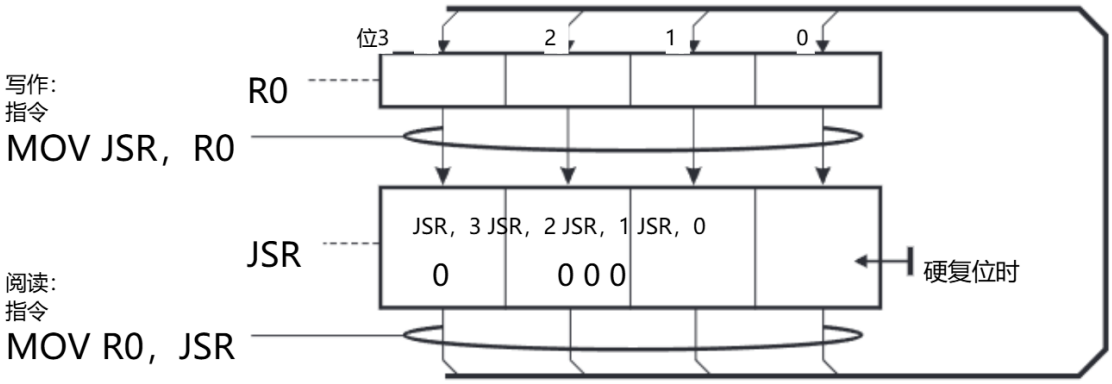
Note that only the instructions **MOV RX,RY**, **MOV RX,N**, **INC RY** and **DEC RY** will initiate the subroutine call. Modifying or writing to **PCL** by any other instruction will be treated as the simple memory modify or write.

Note: In the following example, program calls the **2th level** of subroutine from the **1st level**, that's why the instruction increments **SP** from **1** to **2**. On the next **RET** execution, return address **0001 1100 1011** will be reloaded back to the **Program Counter**, and the program will return to the level **1** of **Stack Pointer**.



特殊功能寄存器SFR0C（JSR） 0x0C

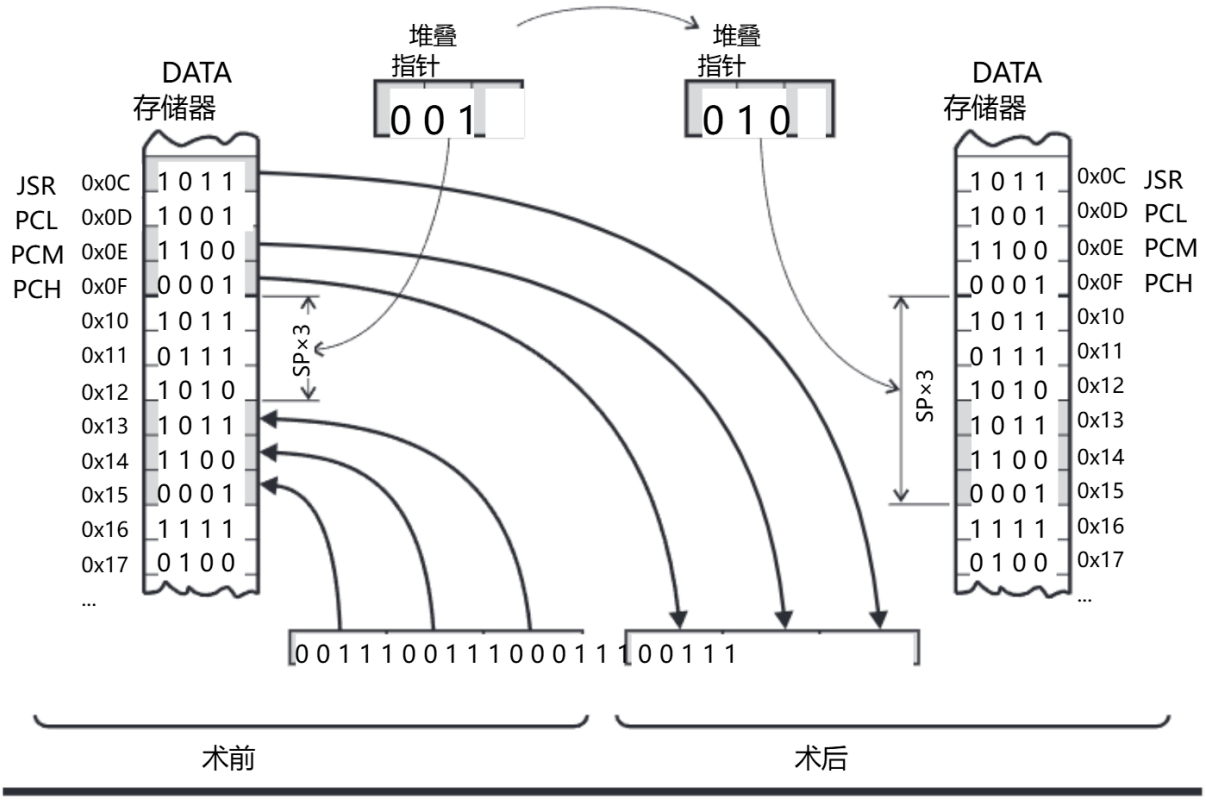
产品描述： 寄存器JSR是用于子例程调用的低半字节地址寄存器。



操作方式：从该寄存器的阅读是一个简单的读操作，但是当程序对此寄存器执行写操作时，子例程调用会自动启动。首先，将程序计数器内容移至堆栈寄存器寻址的数据存储器中的堆栈，位于从地址0x10+（[SP]×3）开始的三个位置。之后，SP寄存器递增1，位置JSR（低半字节）、PCM和PCH（高半字节）的内容加载到程序计数器。

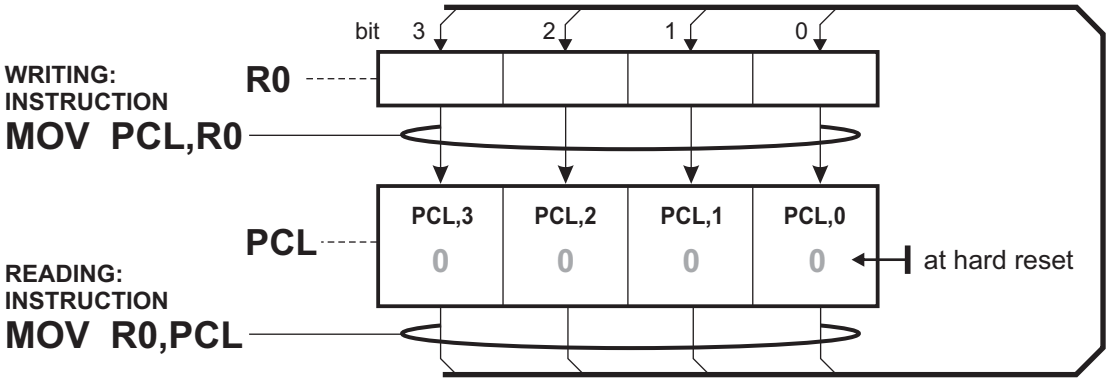
注意，只有指令MOV RX、RY、MOV RX、N、INC RY和DEC RY将启动子例程调用。通过任何其他指令修改或写入PCL将被视为简单的内存修改或写入。

注意：在下面的例子中，程序从第一级调用第二级子例程，这就是为什么指令将SP从1递增到2。在下一次RET执行时，返回地址0001 1100 1011将重新加载回程序计数器，并且程序将返回到堆栈指针的级别1。



Special Function Register SFR0D (PCL) 0x0D

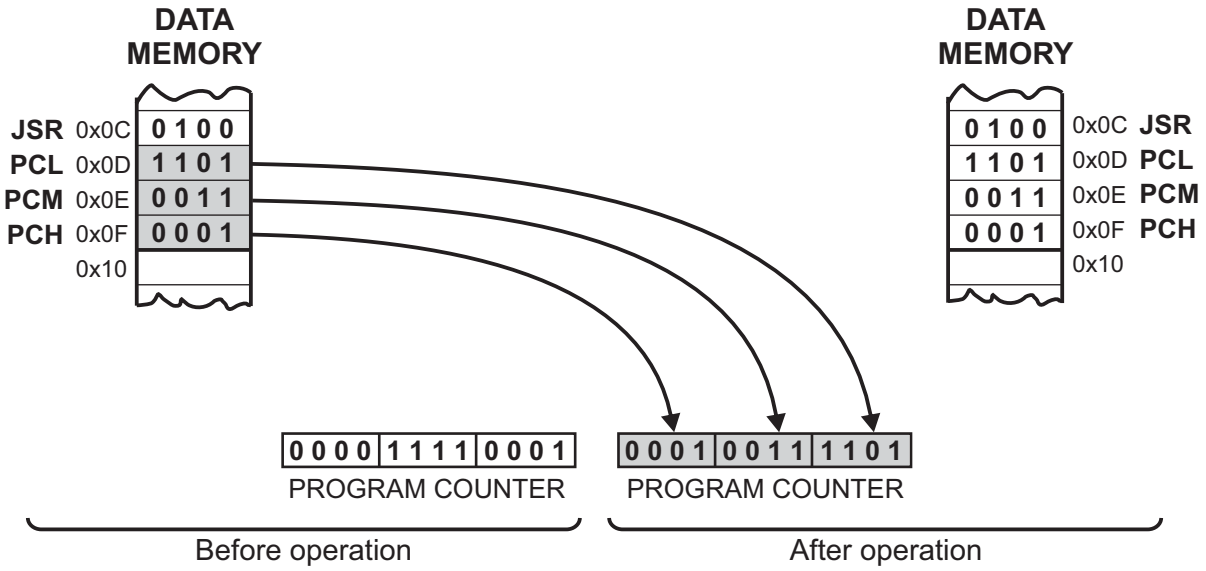
Description: Register **PCL** is the low-nibble address register for a program jump.



Operation: Reading from this register is a simple read operation, but when the program performs writing to this register, the program jump is automatically initiated. The contents of locations **PCL** (low nibble), **PCM** and **PCH** (high nibble) are loaded to **Program Counter**.

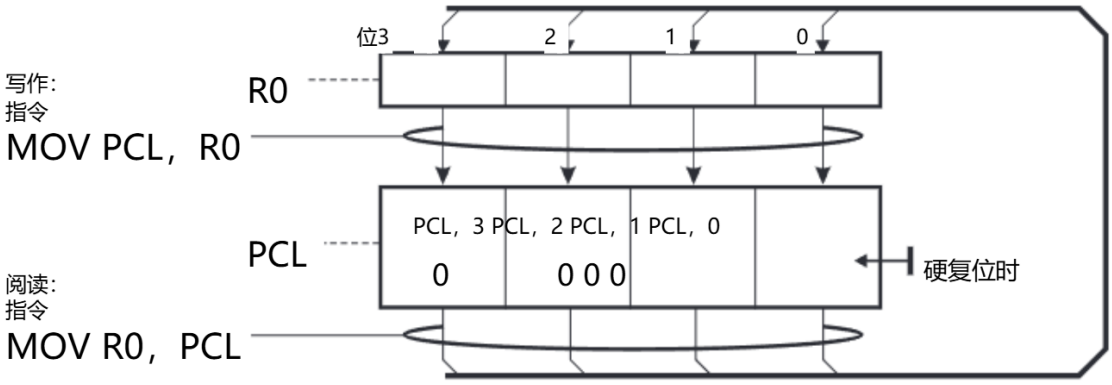
Note that only the instructions **MOV RX,RY**, **MOV RX,N**, **INC RY** and **DEC RY** will initiate the program jump. Modifying or writing to **PCL** by any other instruction will be treated as the simple memory modify or write.

Note: In the following example, program jumps from the address **0001 0011 1100** (**Program Counter** always points to the next address) to the location **0001 0011 1101**, determined by registers **PCH** (high nibble), **PCM** and **PCL** (low nibble). Only **Program Counter** is modified, the contents of all other registers are unaffected.



特殊功能寄存器SFR0D（PCL） 0x0D

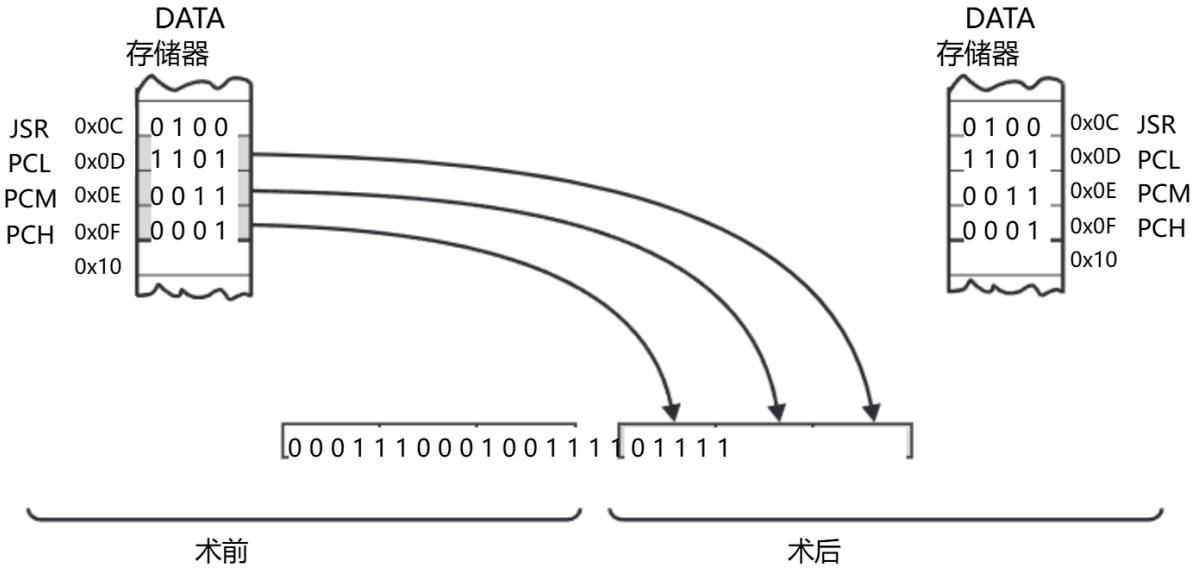
产品描述： 寄存器PCL是用于程序跳转的低半字节地址寄存器。



操作方式：从该寄存器的阅读是一个简单的读操作，但是当程序执行对此寄存器的写操作时，程序跳转自动启动。位置PCL（低半字节）、PCM和PCH（高半字节）的内容被加载到程序计数器。

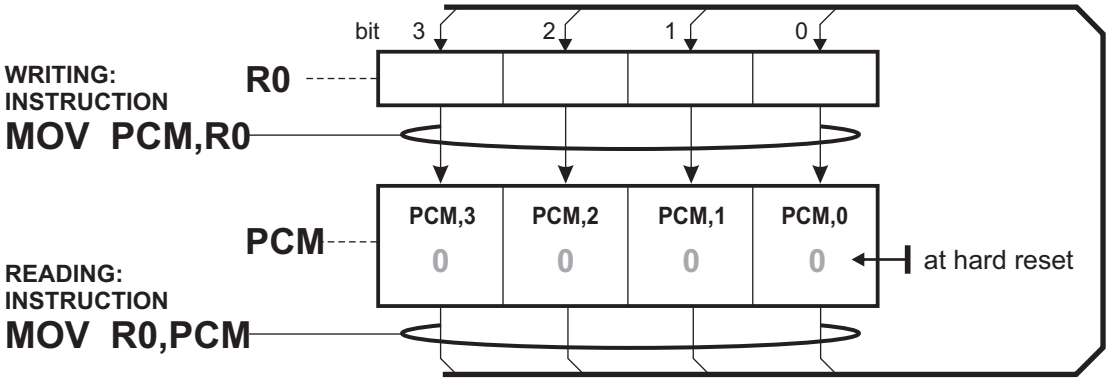
注意，只有指令MOV RX、RY、MOV RX、N、INC RY和DEC RY将启动程序跳转。通过任何其他指令修改或写入PCL将被视为简单的内存修改或写入。

注意事项：在以下示例中，程序从地址0001 0011 1100（程序计数器始终指向下一个地址）跳转到由寄存器PCH（高半字节）、PCM和PCL（低半字节）确定的位置0001 0011 1101。只有程序计数器被修改，所有其他寄存器的内容不受影响。



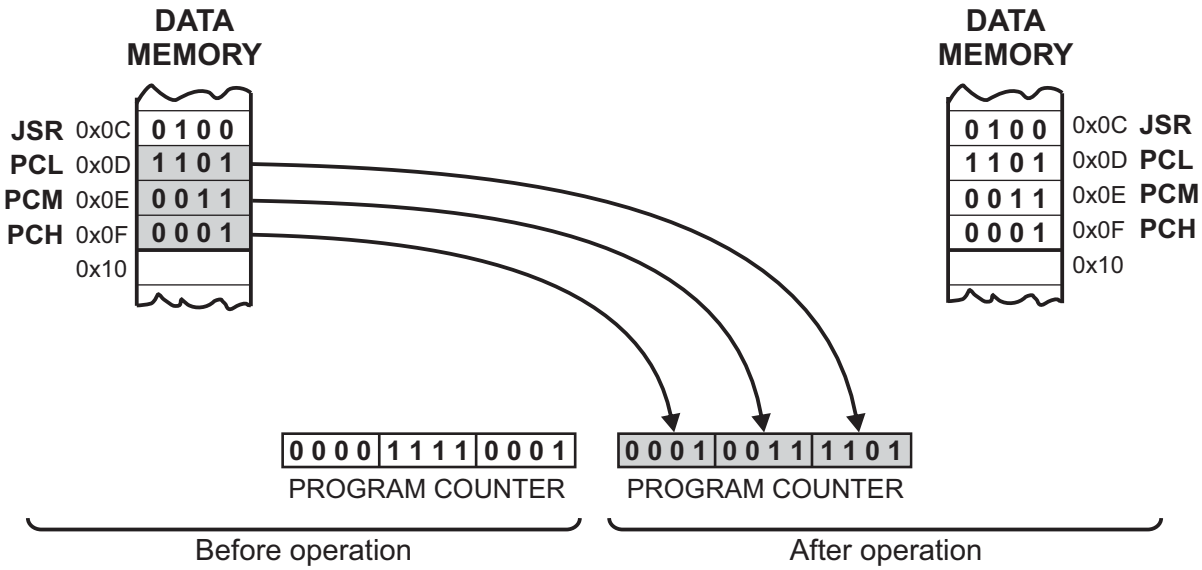
Special Function Register SFR0E (PCM) 0x0E

Description: Register **PCM** is the next-nibble address register for program jump or subroutine call.



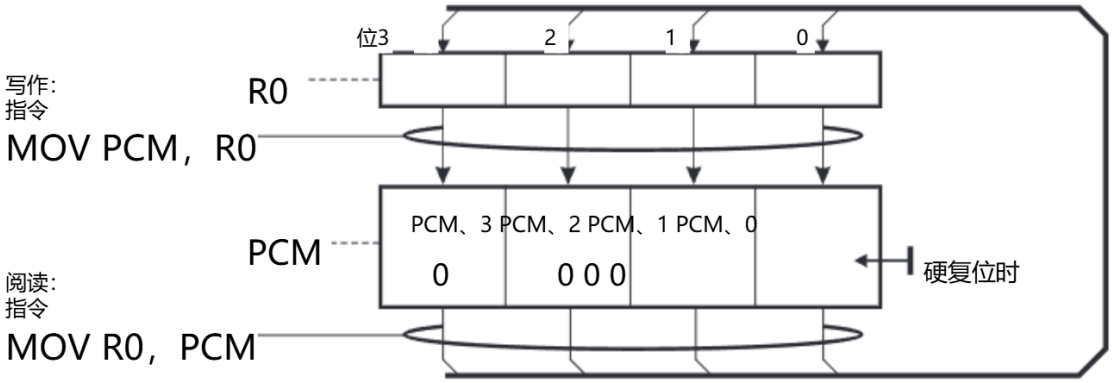
Operation: Reading from or writing to this register is a simple read or write operation, so it does not affect the program flow. However, this register will take action when program jump (writing to **PCL)** or subroutine call (writing to **JSR**) occurs, as it will be a part of the **Program Counter** contents.

The following example shows how **PCM** affects the final **Program Counter** contents during the program jump. The effect is similar in Subroutine Call, only the lower nibble of the **Program Counter** is not loaded from **PCL**, but from **JSR**.



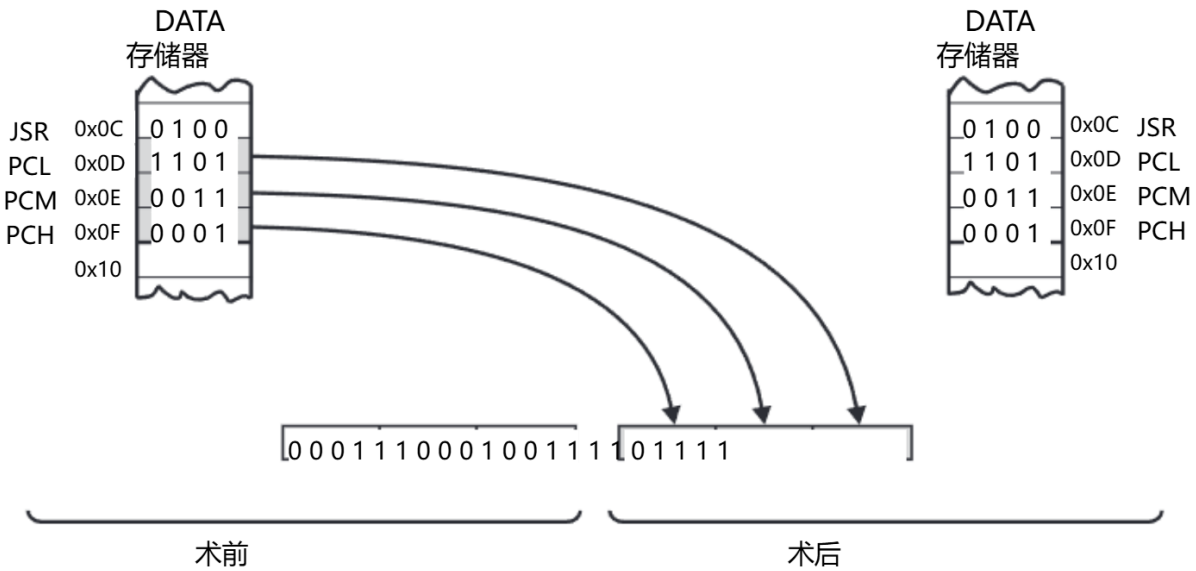
特殊功能寄存器SFR0E（PCM）　　0x0E

产品描述：　　PCM寄存器是程序跳转或子程序调用的下一个半字节地址寄存器。



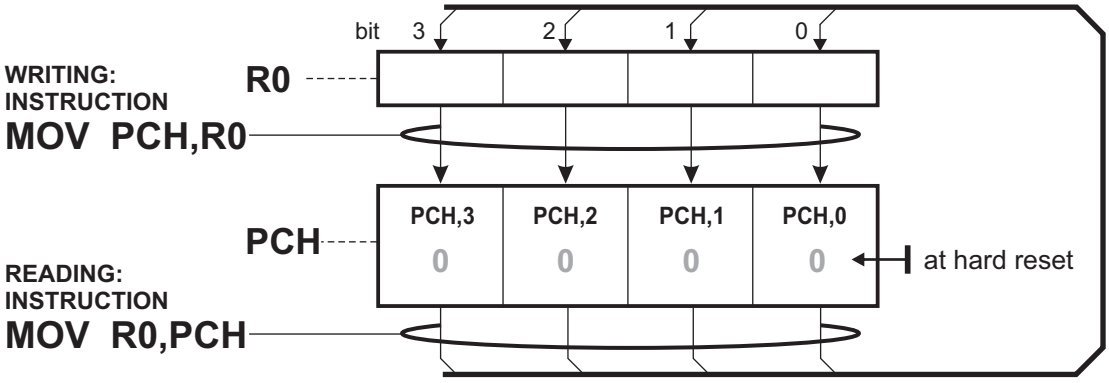
操作：阅读或写此寄存器是一个简单的读或写操作，因此不会影响程序流。但是，当程序跳转（写入PCL）或子例程调用（写入JSR）发生时，该寄存器将采取行动，因为它将成为程序计数器内容的一部分。

以下示例显示了PCM在程序跳转期间如何影响最终程序计数器内容。在Subroutine Call中的效果类似，只是程序计数器的下半字节不是从PCL加载，而是从JSR加载。



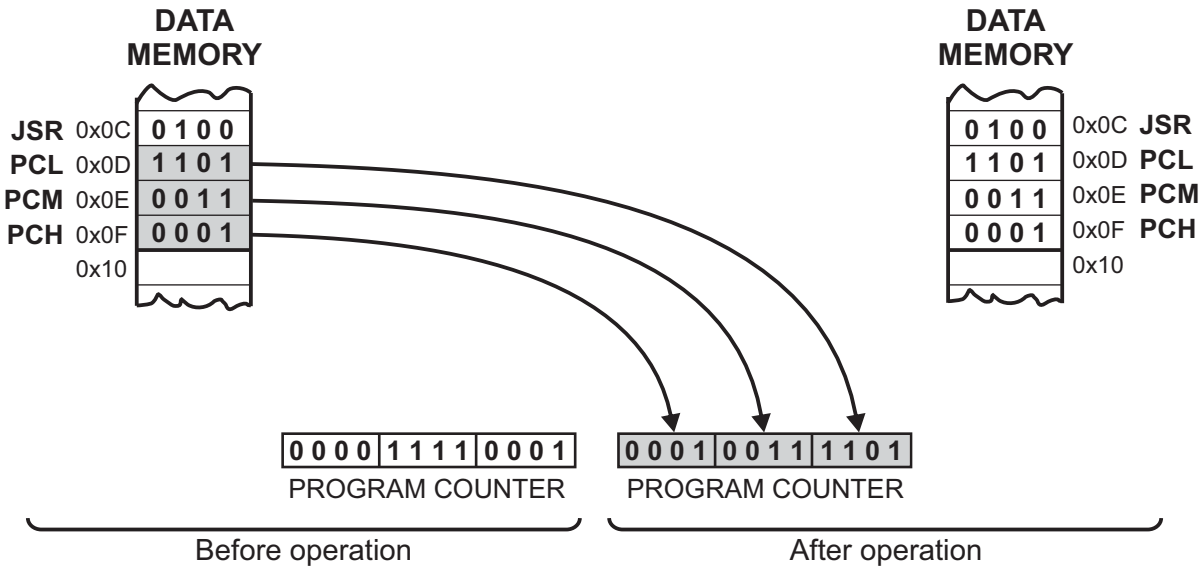
Special Function Register SFR0F (PCH) 0x0F

Description: Register **PCH** is the next-nibble address register for program jump or subroutine call.



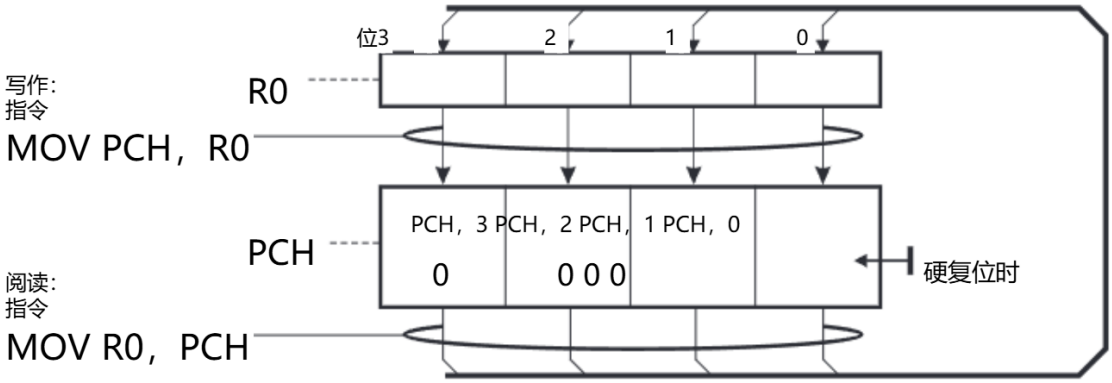
Operation: Reading from or writing to this register is a simple read or write operation, so it does not affect the program flow. However, this register will take action when program jump (writing to **PCL)** or subroutine call (writing to **JSR**) occurs, as it will be a part of the **Program Counter** contents.

The following example shows how **PCH** affects the final **Program Counter** contents during the program jump. The effect is similar in Subroutine Call, only the lower nibble of the **Program Counter** is not loaded from **PCL**, but from **JSR**.



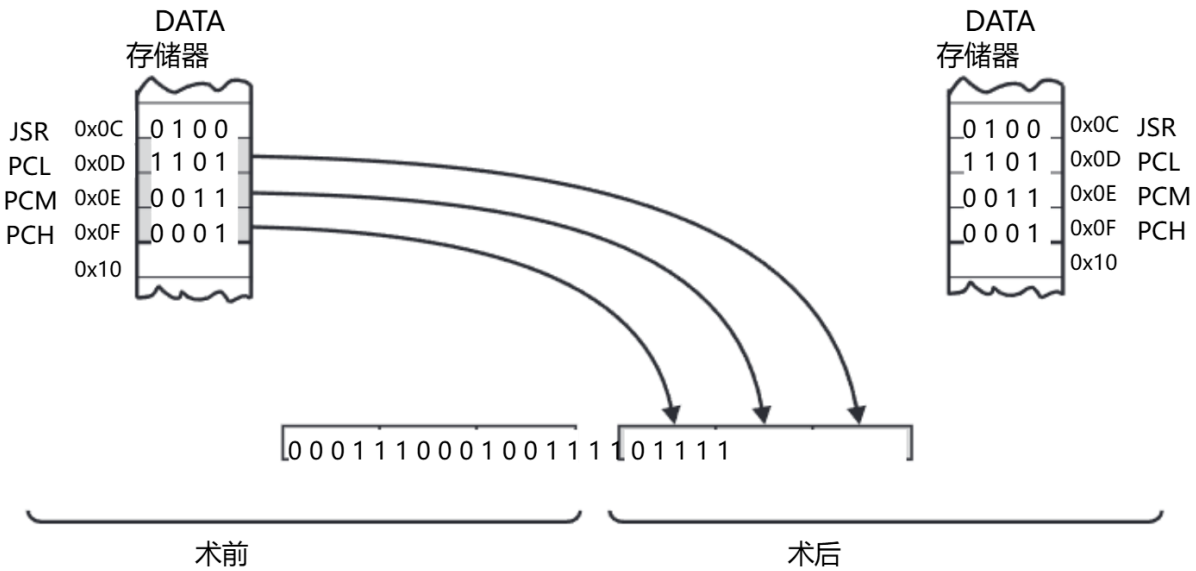
特殊功能寄存器SFR0F（PCH） 0x0F

产品描述： 寄存器PCH是程序跳转或子程序调用的下一个半字节地址寄存器。



操作：阅读或写此寄存器是一个简单的读或写操作，因此不会影响程序流。但是，当程序跳转（写入PCL）或子例程调用（写入JSR）发生时，该寄存器将采取行动，因为它将成为程序计数器内容的一部分。

以下示例显示PCH如何在程序跳转期间影响最终程序计数器内容。在Subroutine Call中的效果类似，只是程序计数器的下半字节不是从PCL加载，而是从JSR加载。



Special Function Registers on Page 15

	NAME	bit 3	bit 2	bit 1	bit 0	DEF
F0	Page	← PAGE →				0000
F1	Clock	← CLOCK →				0000
F2	Sync	← SYNC →				0000
F3	WrFlags	LedsOff	MatrixOff	InOutPos	RxTxPos	0000
F4	RdFlags	Reserved	Reserved	Vflag	UserSync	0000
F5	SerCtrl	RxError	← BaudRate →			0011
F6	SerLow	← SER LOW →				0000
F7	SerHigh	← SER HIGH →				0000
F8	Received	← RECEIVED →				0000
F9	AutoOff	← AUTO OFF →				0010
FA	OutB	← OUT B →				0000
FB	InB	← IN B →				0000
FC	KeyStatus	AltPress	AnyPress	LastPress	JustPress	0000
FD	KeyReg	← KEY REG →				0000
FE	Dimmer	← DIMMER →				0000
FF	Random	← RANDOM →				0000

Note: Bits with Flag symbol are automatically cleared after register reading

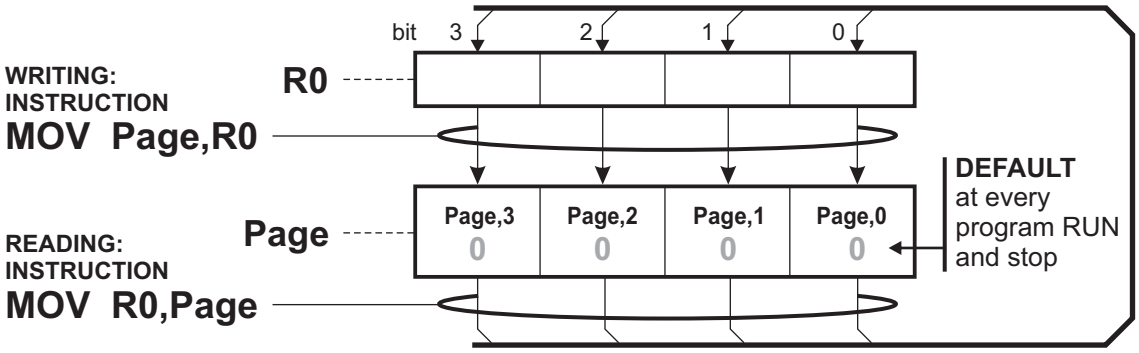
第15页上的特殊功能寄存器

	NAME	位3	位2	位1	位0	DEF
F0	Page	PAGE				0000
F1	时钟	时钟				0000
F2	Sync	SYNC				0000
F3	中国国旗	LedsOff	矩阵Off	输入输出	位置RxTxPos	0000
F4	RdFlags	保留	保留	Vflag	UserSync	0000
F5	SerCtrl	Rx错误	波特率			0011
F6	塞洛	SER低				0000
F7	SerHigh	SER高				0000
F8	接收	接收				0000
F9	自动关闭	自动关闭				0010
FA	OutB	输出B				0000
FB	InB	IN B				0000
FC	KeyStatus	AltPress	AnyPress	LastPress	JustPress	0000
FD	KeyReg	注册码				0000
FE	调光	调光				0000
FF	随机	随机				0000

注：寄存器阅读后，带标志符号的位自动清零

Special Function Register SFRF0 (Page) 0xF0

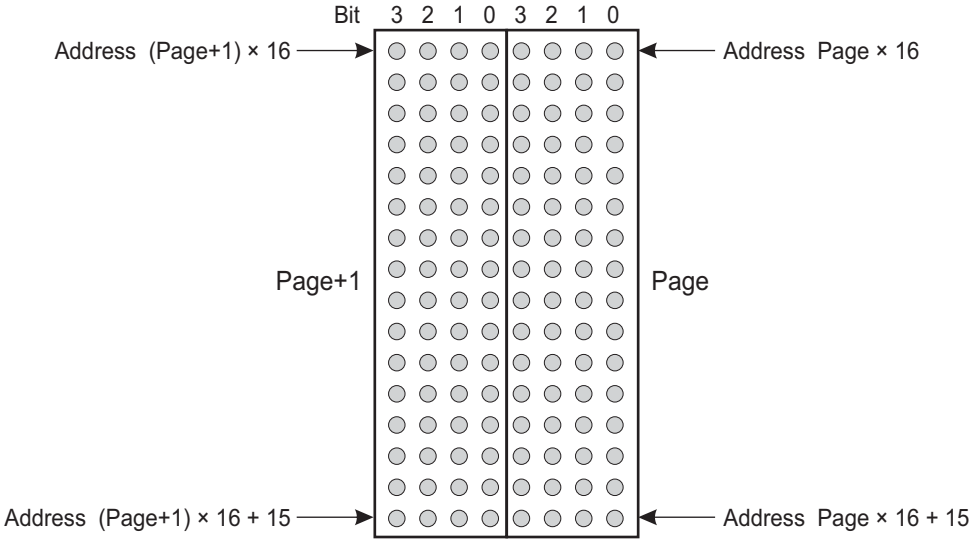
Description: Register **Page** selects which pages from the data memory will be dipslayed.



Operation: LED matrix with the resolution **8×16** displays two pages of data memory, 16 nibbles each. Register **Page** selects which page will be displayed. Selected page is on the right halve of the display, with the first nibble (address **xxxx 0000**) at the top, and the last one (**xxxx 1111**) at the bottom. Bit **3** is in the left column, and the bit **0** in the right. The next page is displayed on the left halve, with the same order.

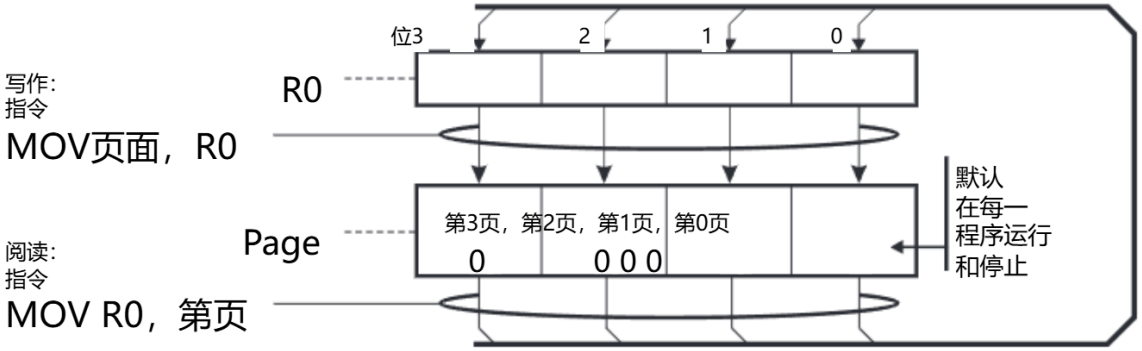
Special case: If the selected page is **15** (which is the last page), the **Next Page** (displayed on the left halve of the matrix) will be page 0. This can be used to watch the main register set (**Page 0**) and the **SFR** set (**Page 15**) at the same time.

Note: Contents of SFR register **Page** can be easily modified by holding button **ALT** and using buttons in **Operand Y** field to select the value. While button **ALT** is depressed, only **Page 0** is displayed (which can be used to see the main register set on **Page 0** promptly), but when the button **ALT** is released, the matrix will display the selected page.



特殊功能寄存器SFRF 0（第页） 0xF0

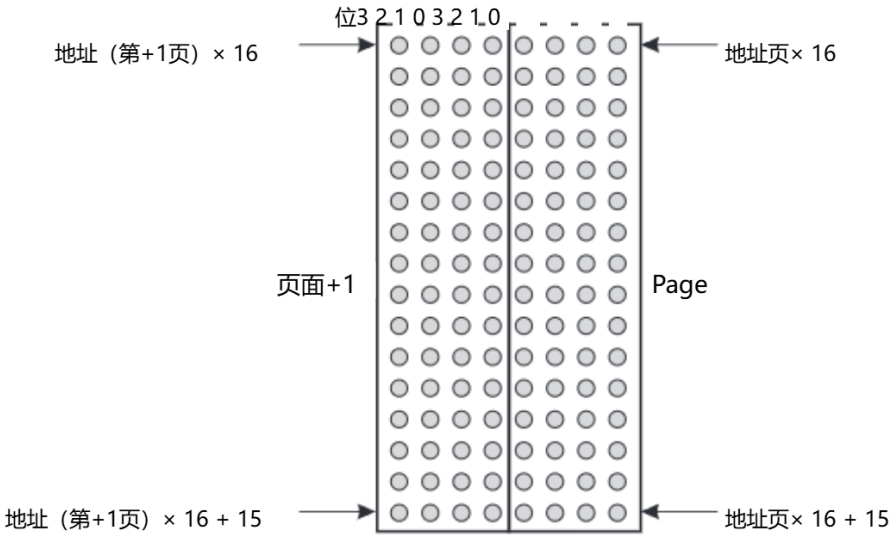
产品描述： 寄存器页选择数据存储器中的哪些页将被双列存储。



操作：8×16分辨率的LED矩阵显示两页数据存储器，每页16个半字节。注册页面选择要显示的页面。所选页面位于显示屏的右半部分，第一个半字节（地址xxxx 0000）位于顶部，最后一个半字节（地址xxxx 1111）位于底部。第3位在左列，第0位在右列。下一页显示在左半部分，顺序相同。

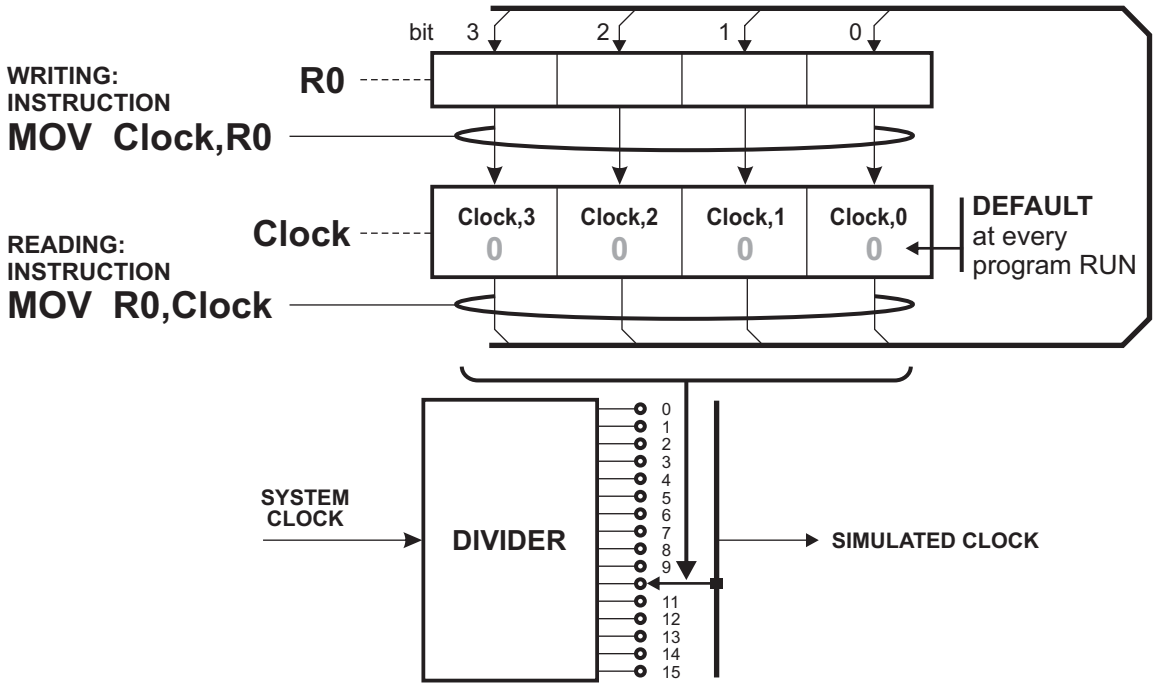
特殊情况：如果选择的页面为15（即最后一页），则下一页（显示在矩阵的左半部分）将是第0页。这可用于同时监视主寄存器组（第0页）和SFR组（第15页）。

注：通过按住ALT按钮并使用操作数Y字段中的按钮选择值，可以轻松修改SFR寄存器页面的内容。按下ALT按钮时，仅显示第0页（可用于立即查看第0页上设置的主寄存器），但松开ALT按钮时，矩阵将显示所选页面。



Special Function Register SFRF1 (Clock) 0xF1

Description: Register **Clock** selects one of 16 available processor speeds.



Operation: All functions of the hypothetical processor are simulated by the 16-bit microcontroller **PIC24FJ256GA704-I/PT**, running at **16 MHz**. As every instruction needs to be executed by a group of the microcontroller instructions, the maximum execution speed is not constant and it greatly depends on the instructions used in the user's program.

SFR1	Frequency	Period
0	~250 KHz	~4 µs
1	100 KHz	~10 µs
2	30 KHz	33 µs
3	10 KHz	100 µs
4	3 KHz	333 µs
5	1 KHz	1 ms
6	500 Hz	2 ms
7	200 Hz	5 ms
8	100 Hz	10 ms
9	50 Hz	20 ms
10	20 Hz	50 ms
11	10 Hz	100 ms
12	5 Hz	200 ms
13	2 Hz	500 ms
14	1 Hz	1 s
15	0.5 Hz	2 s

Here is the table of available frequencies and periods. After every instruction, the execution is waiting for the synchronisation with the selected output of the divider chain, except when **Clock=0**. In that case, the execution is the fastest possible, which is about **0.25 MIPS** (about **4 µs** for an average instruction).

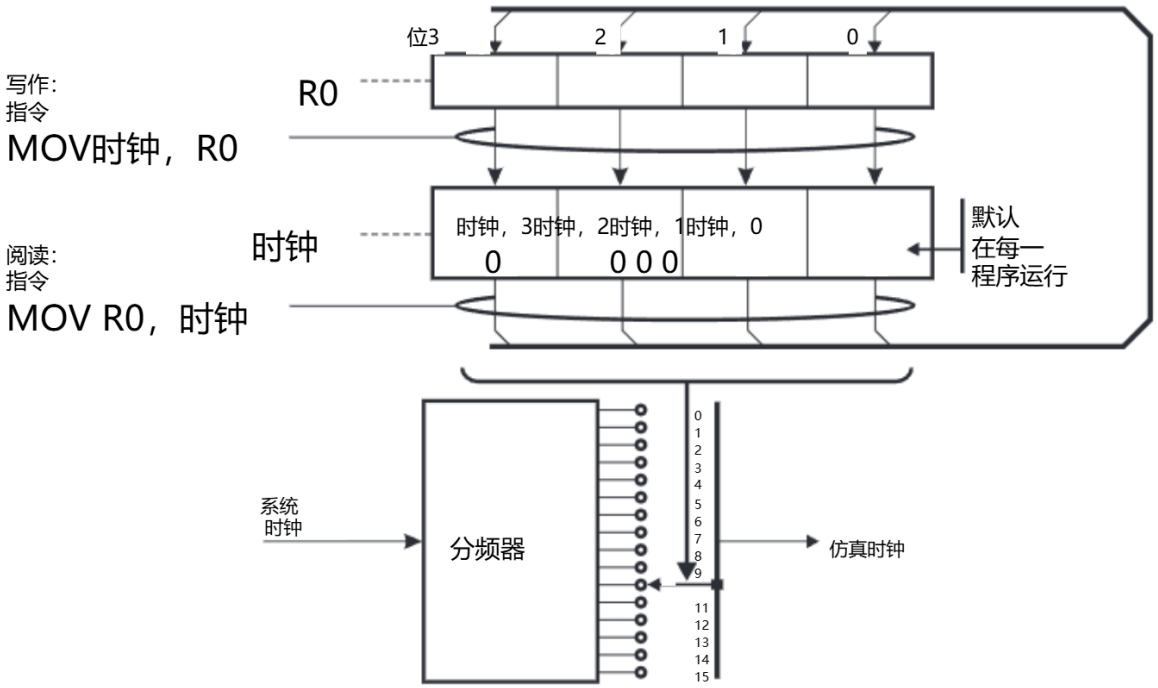
Note 1: Contents of SFR register **Clock** can be easily modified by holding button **ALT** and using buttons in **Operand X** field to select the value.

Note 2: On every **RUN** command, Clock parameter is initialized to **0000** (**250 KHz**).

Note 3: In **SS** mode, Clock has no effect, but it is internally initialized to **0001** for internal program reasons.

特殊功能寄存器SFRF 1（时钟） 0xF1

产品描述： 寄存器时钟选择16个可用处理器速度之一。



操作：所有功能的假设处理器是由16位微控制器模拟PIC 24 FJ 256 GA 704-I/PT，在16 MHz运行。由于每个指令都需要由一组微控制器指令执行，因此最大执行速度不是恒定的，它在很大程度上取决于用户程序中使用的指令。

SFR1频率周期		
0	约250 KHz	约4 µs
1	100 KHz	约10 µs
2	30 KHz	33 µs
3	10 KHz	100 µs
4	3 KHz	333 µs
5	1 KHz	1 ms
6	500 Hz	2 ms
7	200 Hz	5 ms
8	100 Hz	10 ms
9	50 Hz	20毫秒
10	20 Hz	50 ms
11	10 Hz	100 ms
12	5 Hz	200 ms
13	2 Hz	500 ms
14	1 Hz	1 s
15	0.5 Hz	2 s

这是可用频率和周期的表格。在每条指令之后，执行都会等待与分频器链的选定输出同步，除非Clock=0。在这种情况下，执行速度是最快的，约为0.25 MIPS（平均指令约为4 µs）。

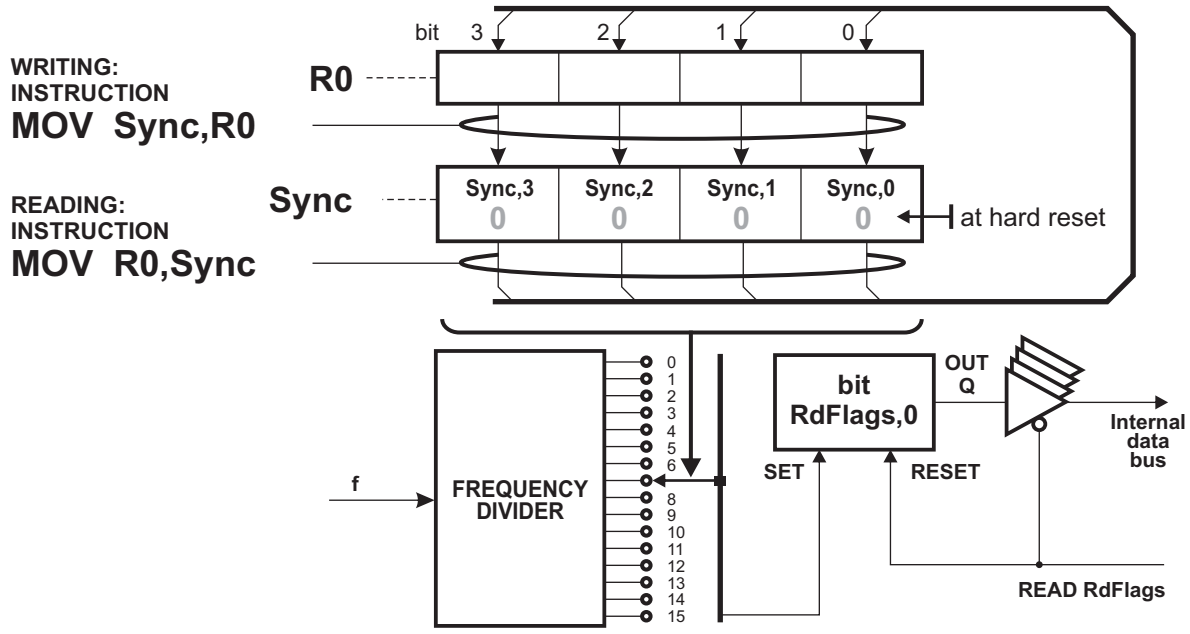
注1：通过按住ALT按钮并使用操作数X字段中的按钮选择值，可以轻松修改SFR寄存器Clock的内容。

注2：每次执行RUN命令时，时钟参数初始化为0000（250 KHz）。

注3：在SS模式下，时钟没有影响，但由于内部程序原因，它在内部被初始化为0001

Special Function Register SFRF2 (Sync) 0xF2

Description: Register **Sync** selects one of 16 fixed timings used in the frequency divider which sets the special **UserSync** flag in the register **RdFlags**.



Operation: Uniform clock is divided by the factor determined by the register **Sync** and the overflow pulse sets the **bit #0** (**UserSync**) in the register **RdFlags**. Program can read the **bit #0** in the register **RdFlags** (using instruction **MOV R0,RdFlags**) and thus synchronize the program flow with the real-time ticking. Reading from register **RdFlags** automatically resets bit **0** (**UserSync**).

Here is the table of available frequencies and periods:

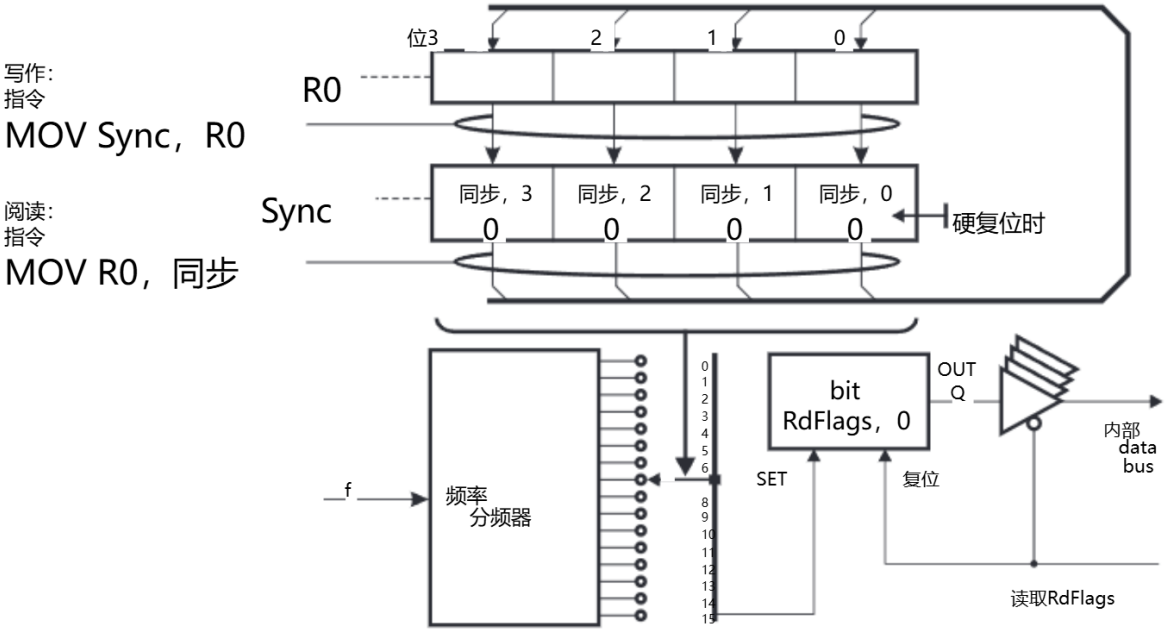
Sync	Frequency	Period
0	1000 Hz	1 ms
1	600 Hz	1.67 ms
2	400 Hz	2.5 ms
3	250 Hz	4 ms
4	150 Hz	6.67 ms
5	100 Hz	10 ms
6	60 Hz	16.7 ms
7	40 Hz	25 ms
8	25 Hz	40 ms
9	15 Hz	66.7 ms
10	10 Hz	100 ms
11	6 Hz	167 ms
12	4 Hz	250 ms
13	2.5 Hz	400 ms
14	1.5 Hz	667 ms
15	1 Hz	1 sec

Program example:

```
; initialization (only once)

  MOV   R0,12       ; 250 ms period
  MOV   [252],R0    ; write to reg Sync

  MOV   R0,[244]    ; get status
  BIT   R0,0        ; test User Sync bit
  SKIP  3,1         ; if NZ, skip 1 line
  BRA   -4          ; loop

  ADD   R3,R6       ; program goes on...

; if this is the part of the other loop,
; program will wait here every time for
; synchronization on 250 ms, and then
; continue execution.
```

特殊功能寄存器SFRF 2（同步） 0xF2

产品描述： 寄存器同步选择分频器中使用的16个固定时序之一，分频器设置寄存器RdFlags中的特殊UserSync标志。



操作方式：均匀时钟除以寄存器Sync确定的因子，溢出脉冲设置寄存器RdFlags中的位#0（UserSync）。程序可以读取寄存器RdFlags中的位#0（使用指令MOV R0，RdFlags），从而使程序流与实时计时同步。阅读寄存器RdFlags会自动复位位0（UserSync）。

以下是可用的频率和周期表：

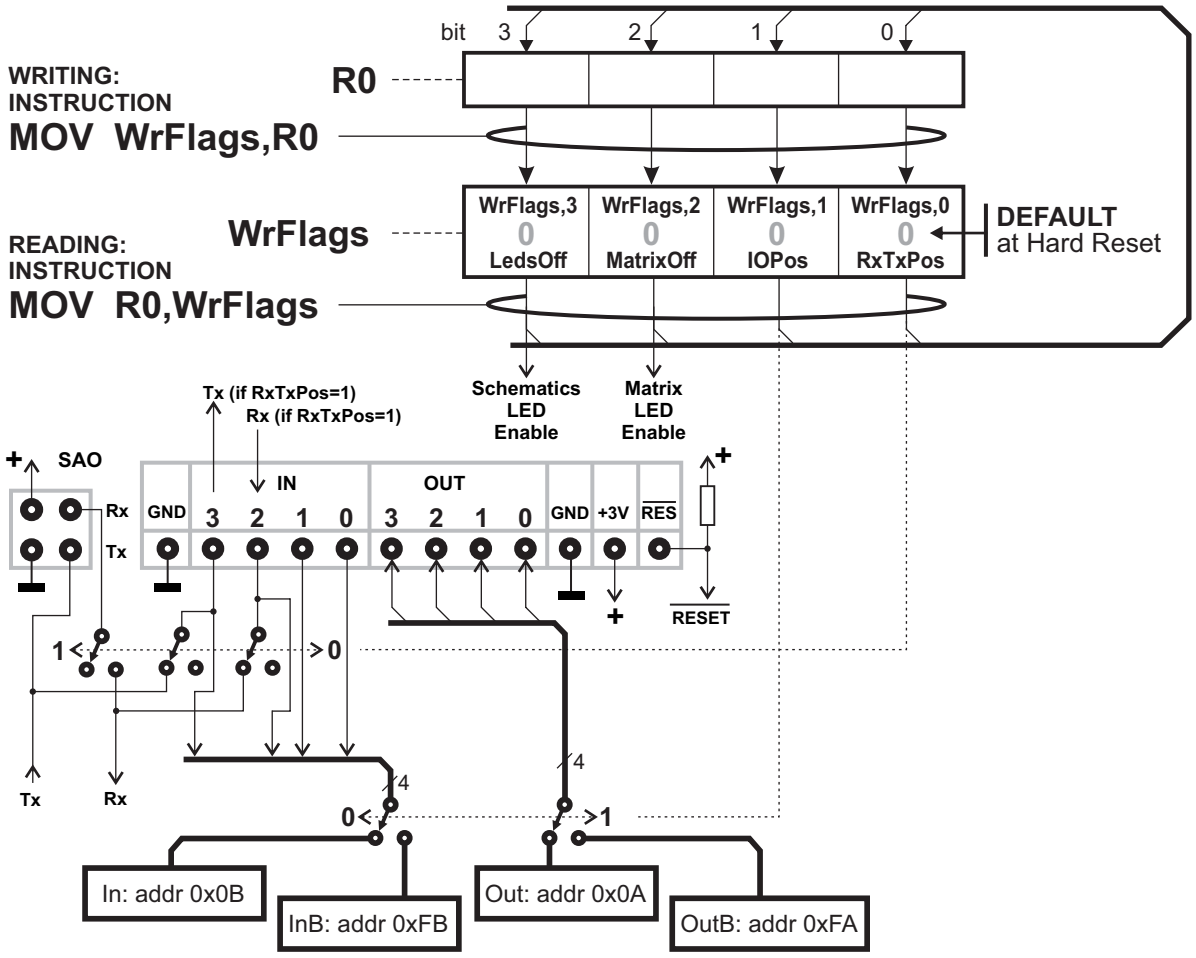
Sync	频率	期间
0	1000 Hz	1 ms
1	600 Hz	1.67 ms
2	400 Hz	2.5 ms
3	250 Hz	4 ms
4	150 Hz	6.67 ms
5	100 Hz	10 ms
6	60 Hz	16.7 ms
7	40 Hz	25 ms
8	25 Hz	40 ms
9	15 Hz	66.7 ms
10	10 Hz	100 ms
11	6 Hz	167毫秒
12	4 Hz	250 ms
13	2.5 Hz	400 ms
14	1.5 Hz	667毫秒
15	1 Hz	1秒

程序示例：

```
;初始化（仅一次）

    MOV R 0，12 ; 250 ms周期MOV [252]，R 0;写入reg Sync

    MOV R 0，[244] ;获取状态BIT R 0，0 ;测试用户同步位
    SKIP 3，1;如果NZ，跳过1行BRA -4 ;循环

    添加R3，R6 ;程序继续...

;如果这是另一个循环的一部分，;程序每次都会在这里等待;同步250 ms，然后;继续执行。
```

Special Function Register SFRF3 (WrFlags) 0xF3

Description: Register **WrFlags** contains individual control flags.



Operation: Every bit in the Register **WrFlags** has its own function:

Bit 3: **LedsOff = 0** LEDs on the **ALU** schematics diagram are enabled.
LedsOff = 1 LEDs on the **ALU** panel schematics diagram are turned off, except LEDs CLK and $\overline{\text{CLK}}$, which are always on.

Bit 2: **MatrixOff = 0** LEDs on the **Data memory Display Matrix** are enabled.
MatrixOff = 1 LEDs on the **Data memory Display Matrix** are turned off.

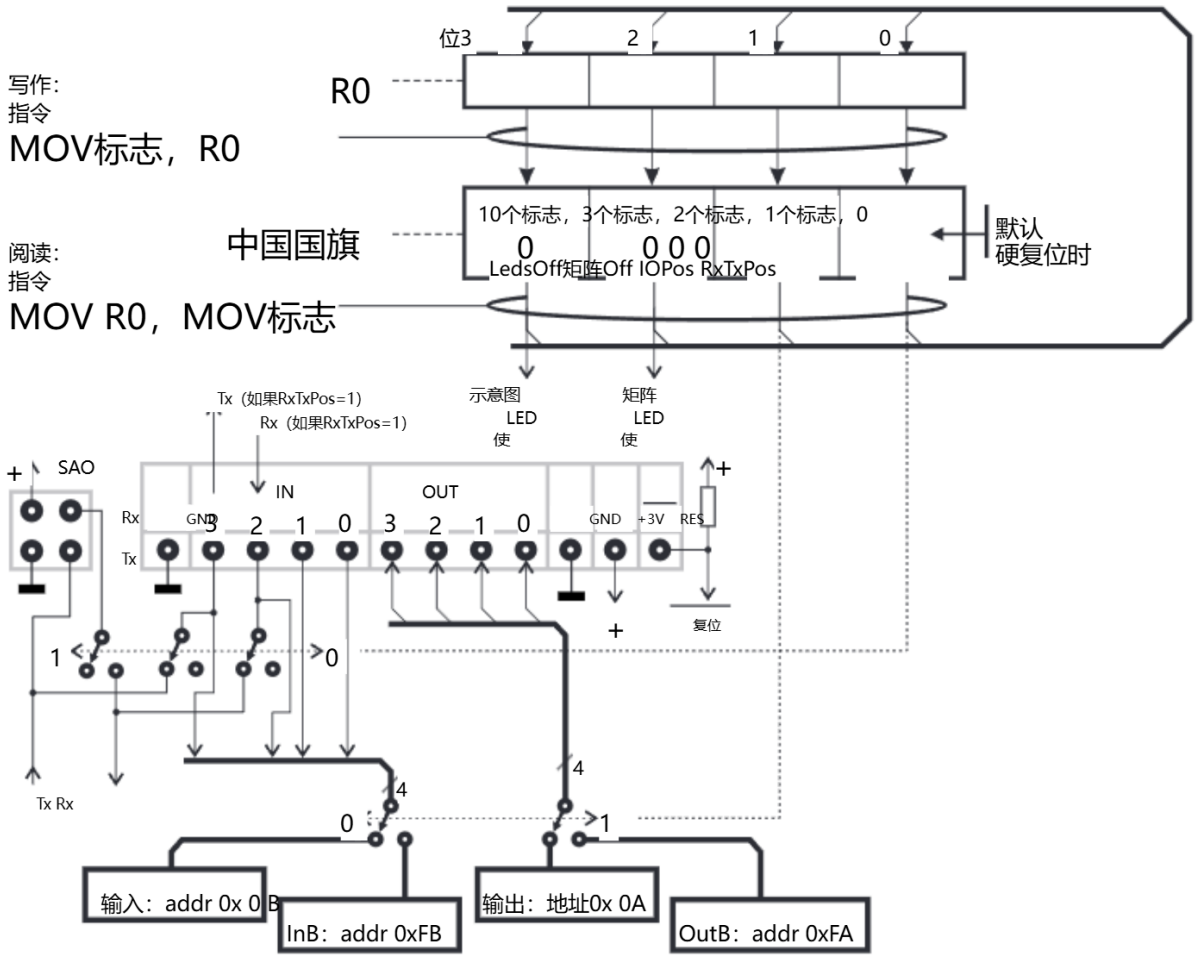
Bit 1: **IOpos = 0** Out register is at address **0x0A**, In register is at address **0x0B**
IOPos = 1 Out register is at address **0xFA**, In register is at address **0xFB**

Bit 0: **RxTxPos = 0** **UART** ports **TXD** and **RXD** are available on the **SAO** connector, and all Input and Output ports are on the main **I/O** connector.

RxTxPos = 1 **UART** ports **TXD** and **RXD** are on the main connector (pins **2** and **3** from the left), **RX** pin on the **SAO** connector is inactive. **TX** pin on **SAO** connector still has **TX** function. Input port (which is always readable on the address 0x0A or 0XFA) is normally active, reading bit0 and bit1, and also current states from bit2 (Rx) and bit3 (Tx).

特殊功能寄存器SFRF 3（触发标志） 0xF3

产品描述： 寄存器控制标志包含单独的控制标志。



操作：寄存器中的每个位都有自己的功能：

位3：LedsOff = 0
LedsOff = 1

ALU原理图上的LED已启用。ALU面板原理图上的LED关闭，但LED CLK和CLK始终亮起。

位2：MatrixOff = 0
矩阵关闭= 1

数据存储器显示矩阵上的LED已启用。
数据存储器显示矩阵上的指示灯熄灭。

位1：IOpos = 0
IOPos = 1

输出寄存器位于地址0x0A，输入寄存器位于地址0x0B输出寄存器位于地址0xFA，输入寄存器位于地址0xFB

位0：RxTxPos = 0

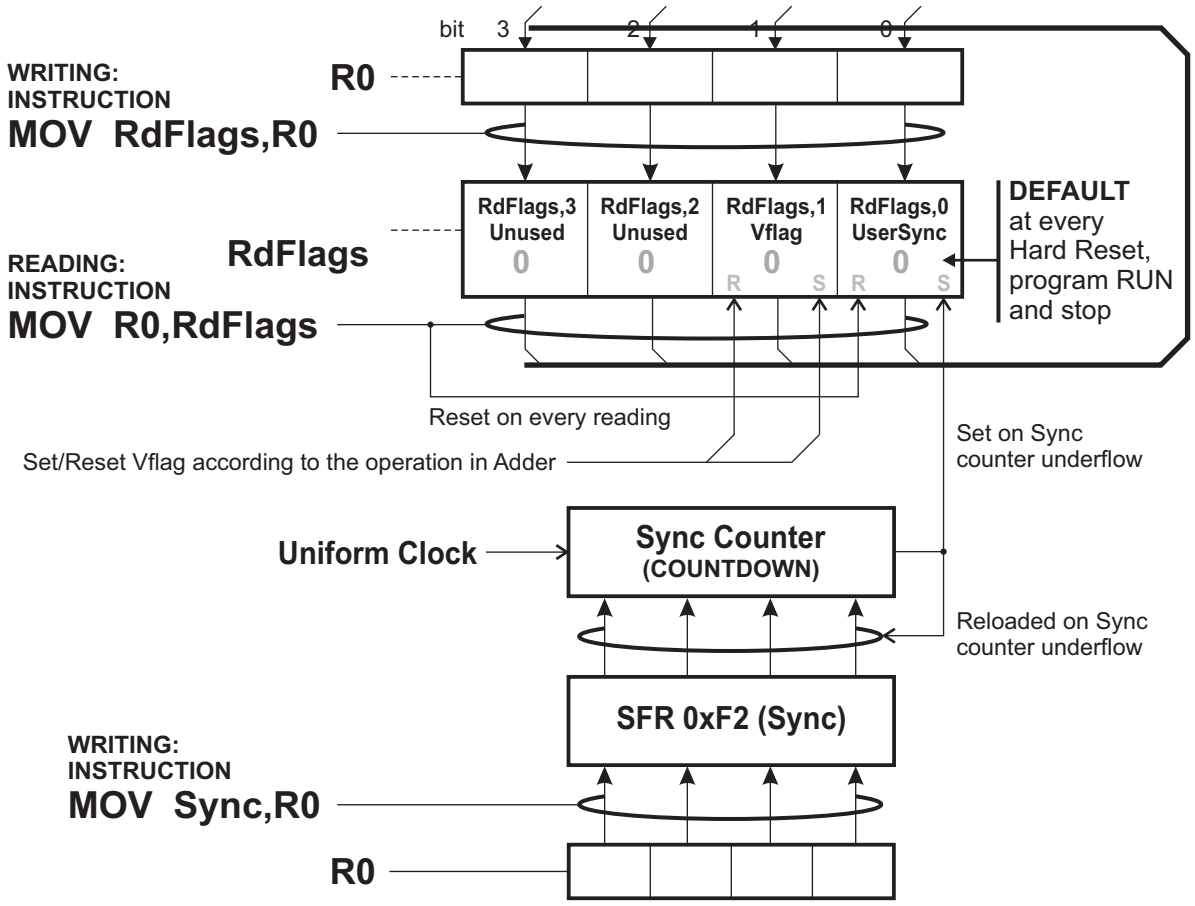
SAO连接器上提供了端口TXD和RXD，所有输入和输出端口都位于主I/O连接器上。

RxTxPos = 1

USB端口TXD和RXD位于主连接器上（从左数第2和第3针），SAO连接器上的RX针处于非活动状态。SAO连接器上的TX引脚仍具有TX功能。输入端口（在地址0x 0A或0XFA上始终可读）通常处于活动状态，阅读位0和位1，以及位2（Rx）和位3（Tx）的当前状态。

Special Function Register SFRF4 (RdFlags) 0xF4

Description: Register **RdFlags** contains individual status flags.



Operation: Register **RdFlags** signifies statuses for the following events:

Bit 3: **Unused**

Bit 2: **Unused**

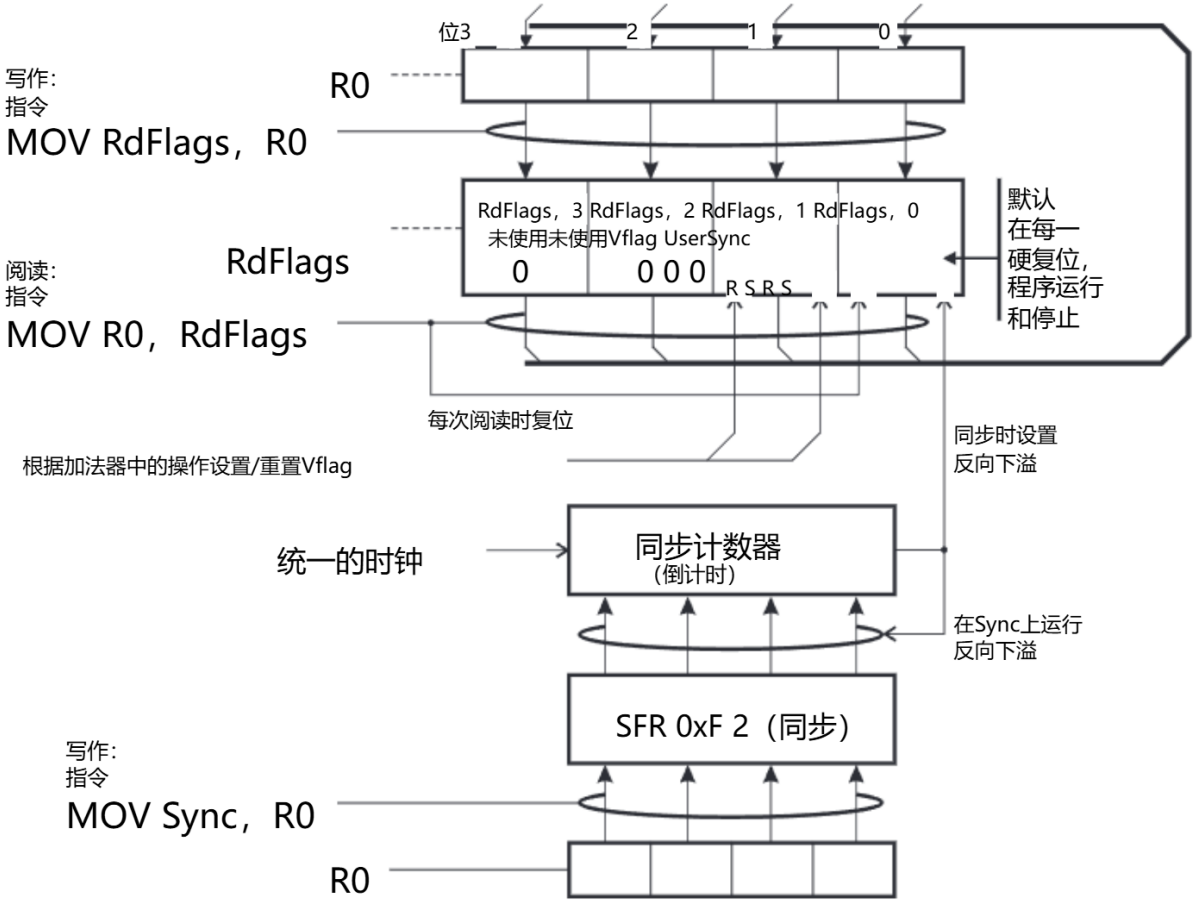
Bit 1: **Vflag = 0** **V flag** not set (signed arithmetic operation was in range)
Vflag = 1 **V flag** is set (signed arithmetic operation was out of range and the result of the operation is incorrect)

Bit 0: **UserSync = 0** **User Sync** didn't occur yet.
UserSync = 1 **User Sync** occurred. This bit is automatically reset upon reading of the register **RdFlags**.

Note: Period for **UserSync** function can be selected by writing the value to the register **Sync.**

特殊功能寄存器SFRF 4（RdFlags） 0xF4

描述：寄存器RdFlags包含单个状态标志。



操作：注册RdFlags表示以下事件的状态：

第3位：未使用

第2位：未使用

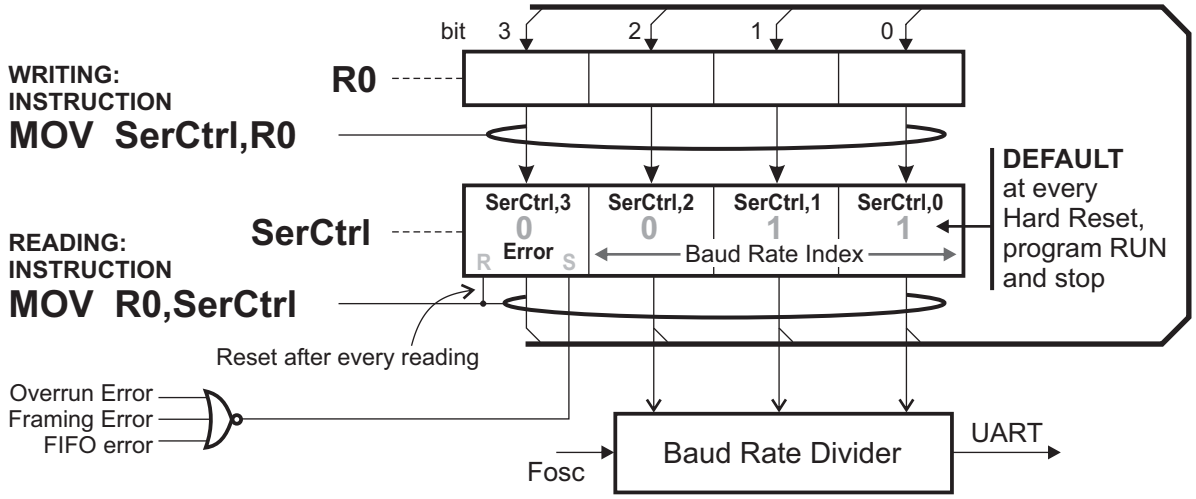
位1：Vflag = 0　　未设置V标志（带符号算术运算在范围内）已设置V标志（带符号算术
　　Vflag = 1　　运算超出范围，运算结果不正确）

位0：UserSync = 0　用户同步尚未发生。
　　UserSync = 1　发生用户同步。阅读寄存器RdFlags时，该位自动复位。

注：用户同步功能的周期可以通过将值写入同步寄存器来选择。

Special Function Register SFRF5 (SerCtrl) 0xF5

Description: Register **SerCtrl** contain the **ERROR** bit for the **Serial Port** and determines the **Baud Rate**.



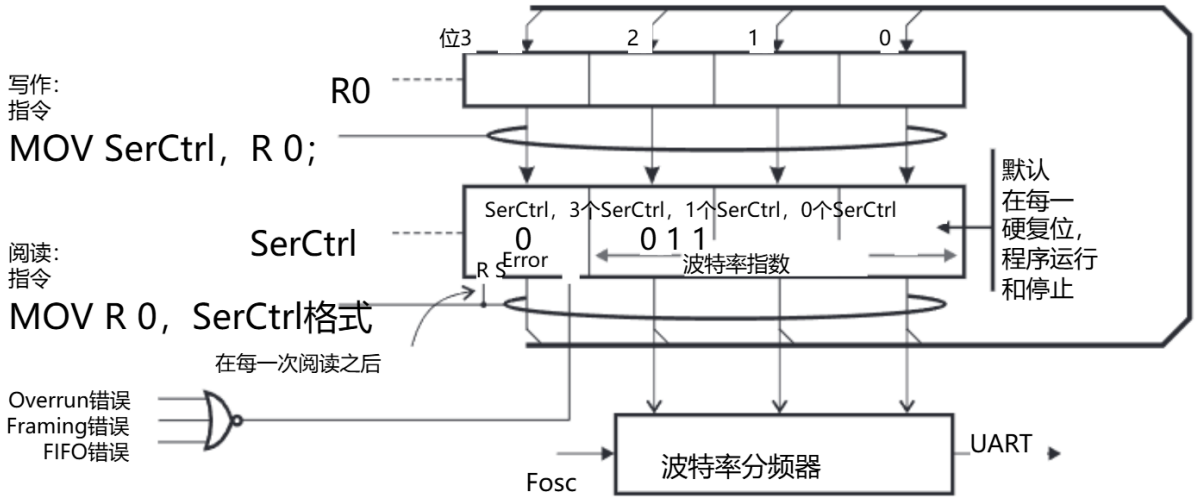
Operation: Bit 3 is used to determine if the error occurred during UART operation. This bit is automatically reset on every reading of register RdFlags.

Bits **#2**, **#1** and **#0** determine the **Baud Rate** for the **UART**.

Here is the table of the available Baud rates:

#	BIT #2	BIT #1	BIT #0	Baud	
0	**0**	**0**	**0**	1200	
1	**0**	**0**	**1**	2400	
2	**0**	**1**	**0**	4800	
3	**0**	**1**	**1**	9600	← **DEFAULT**
4	**1**	**0**	**0**	19200	
5	**1**	**0**	**1**	38600	
6	**1**	**1**	**0**	57600	
7	**1**	**1**	**1**	115200	

产品描述： 寄存器SerCtrl包含串行端口的错误位，并确定波特率。



操作：第3位用于确定是否在重复操作期间发生错误。每次阅读寄存器RdFlags时，该位自动复位。

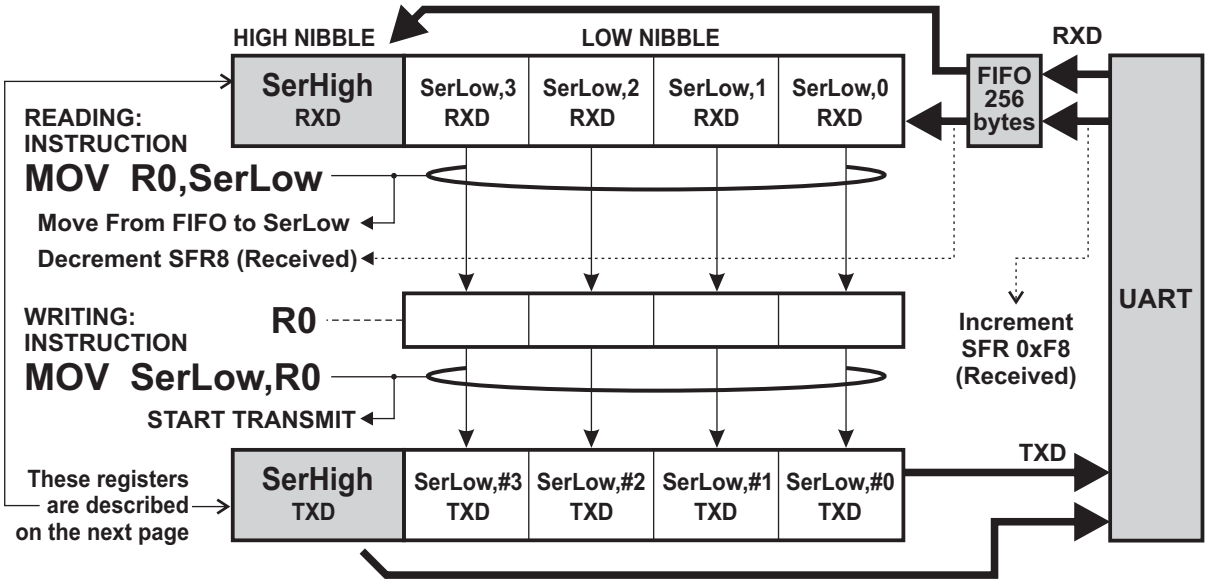
位#2、#1和#0决定了ADC的波特率。

下面是可用波特率的表格：

#	bit bit #2	#1	#0	Baud	
0	0	0	0	1200	
1	0	0	1	2400	
2	0	1	0	4800	
3	0	1	1	9600	← 默认
4	1	0	0	19200	
5	1	0	1	38600	
6	1	1	0	1 57600	
7	1	1		115200	

Special Function Register SFRF6 (SerLow) 0xF6

Description: Register **SerLow** consists of two **4-bit registers** which share the same address, but one of them is read-only, and the other one write-only. The first one is automatically loaded with the low nibble when the serial data byte is received, and another one is used for writing the low nibble of the data byte which will be immediately and automatically transmitted. There is the similar register pair **SFRF7** which is used for the high nibble of data byte.



Power On and **Run** default values for all registers represented here are **0000.**

Operation: This is not a single register but a pair of registers. They are on the same address, but one of them is read-only, and the other one is write-only. We will call them **SerLow-r** and **SerLow-w** here, although they share the same name.

Transmitting is a simple operation, as the handshaking protocol between the processor and **UART** is straightforward. Processor tests if the transmitter is ready (if the last byte was transmitted) and, if so, writes **SerLow-w** (low nibble) and **SerHigh-w** (high nibble) to the **8-bit transmit buffer**. If it's not ready, it will wait until the transmission is finished. This can cause certain slowing down in program flow.

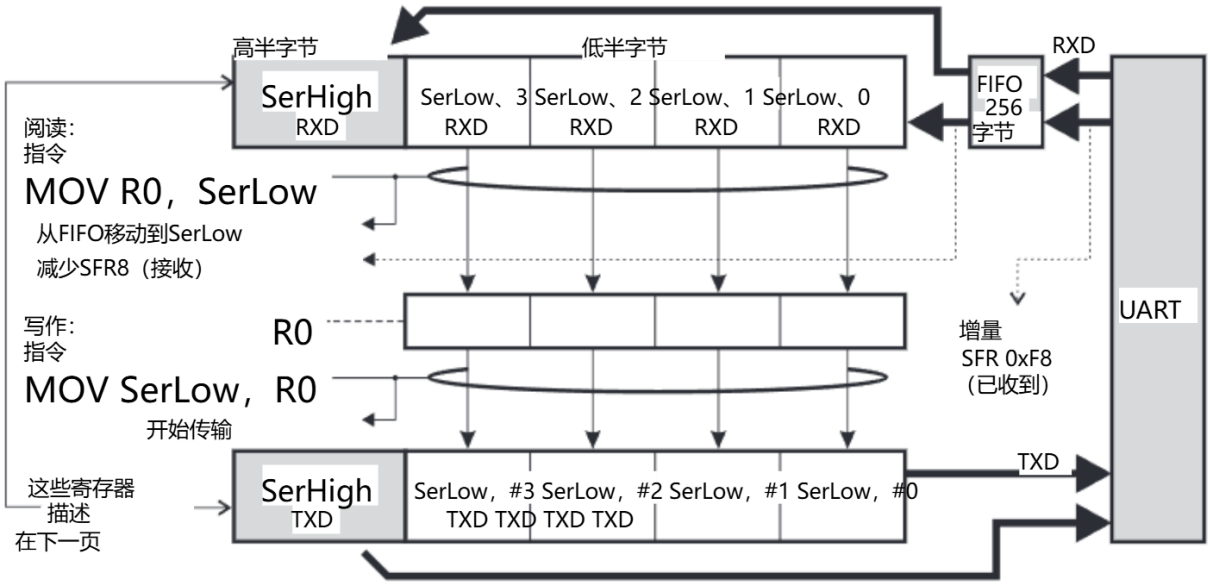
Note that the automatic transmission occurs only when the low nibble is written to **SerLow-w**, and not when the high nibble is written to **SerHigh-w**. For that reason, it is a good practice to write to the high nibble register (**SerHigh-w**) first.

When **UART** receives the data byte, it writes the byte into the **256-byte FIFO** (First In - First Out) buffer. If the whole **FIFO** buffer is full (**256** unread bytes), the following bytes are lost and the **Error** (**SerCtrl,#3**) flag is set. In the concurrent process, if the **SerLow-r** register is empty (data nibble was read), the new byte is written to **SerLow-r** (low nibble) and **SerHigh-r** (high nibble). At that moment, the register **SFR8** (**Received**) will be automatically decremented.

Note that all that happens automatically. All that the user's program has to do, is to test if the register **Received** is greater than **0** and, if it is (which is the sign that the new byte arrived from the serial port), read the high nibble (**SerHigh-r**) first, and then the low nibble (**SerLow-r**). Reading the low nibble (**SerLow-r**) resets the flag **RxRdy** automatically. This is not the case with high nibble reading (**SerHigh-r**), so it is a good practice to read the high nibble (**SerHigh-w**) first.

特殊功能寄存器SFRF 6（SerLow） 0xF6

描述：寄存器SerLow由两个4位寄存器组成，它们共享相同的地址，但其中一个是只读的，另一个是只写的。当接收到串行数据字节时，第一个自动加载低半字节，另一个用于写入将立即自动传输的数据字节的低半字节。有一个类似的寄存器对SFRF 7，用于数据字节的高位半字节。



此处表示的所有寄存器的上电和运行默认值为0000。

操作：这不是单个寄存器，而是一对寄存器。它们位于同一地址，但其中一个是只读的，另一个是只写的。我们在这里将它们称为SerLow-r和SerLow-w，尽管它们共享相同的名称。

传输是一个简单的操作，因为处理器和处理器之间的握手协议很简单。处理器测试发送器是否准备就绪（如果最后一个字节已发送），如果是，则将SerLow-w（低半字节）和SerHigh-w（高半字节）写入8位发送缓冲器。如果尚未准备好，它将等待传输完成。这可能会导致程序流程的某些减慢。

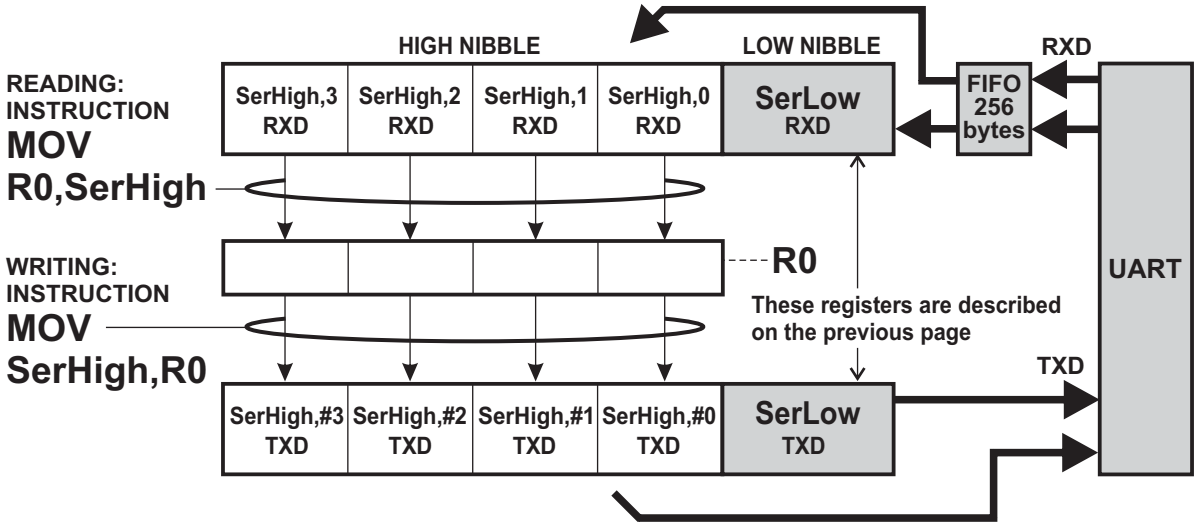
注意，仅当低半字节被写入SerLow-w时才发生自动传输，而当高半字节被写入SerHigh-w时不发生自动传输。因此，最好先写入高位半字节寄存器（SerHigh-w）。

当UART接收到数据字节时，它会将该字节写入256字节FIFO（先进先出）缓冲区。如果整个FIFO缓冲区已满（256个未读字节），则丢失后续字节，并设置错误（SerCtrl，#3）标志。在并发进程中，如果SerLow-r寄存器为空（数据半字节已读取），则新字节将写入SerLow-r（低半字节）和SerHigh-r（高半字节）。此时，寄存器SFR 8（已接收）将自动递减。

请注意，所有这些都是自动发生的。用户程序所要做的就是测试寄存器Received是否大于0，如果大于0（这是新字节从串行端口到达的标志），则首先读取高位半字节（SerHigh-r），然后读取低位半字节（SerLow-r）。阅读低半字节（SerLow-r）自动复位标志RxRdy。对于高半字节阅读（SerHigh-r），情况并非如此，因此最好先读取高半字节（SerHigh-w）。

Special Function Register SFRF7 (SerHigh) 0xF7

Description: Register **SerHigh** consists of two **4-bit registers** which share the same address, but one of them is read-only, and the other one write-only. The first one is automatically loaded with the high nibble when the serial data byte is received, and another one is used for writing the high nibble of the data byte which will be transmitted. There is the similar register pair **SFR6** which is used for the low nibble of data byte.



Power On and **Run** default values for all registers represented here are **0000.**

Operation: This is not a single register but a pair of registers. They are on the same address, but one of them is read-only, and the other one is write-only. We will call them **SerHigh-r** and or **SerHigh-w** here, although they share the same name.

Transmitting is a simple operation, as the handshaking protocol between the processor and **UART** is straightforward. Processor tests if the transmitter is ready (if the last byte was transmitted) and, if so, writes **SerHigh-w** (high nibble) and **SerLow-w** (low nibble) to the **8-bit transmit buffer**. If it's not ready, it will wait until the transmission is finished. This can cause certain slowing down in program flow.

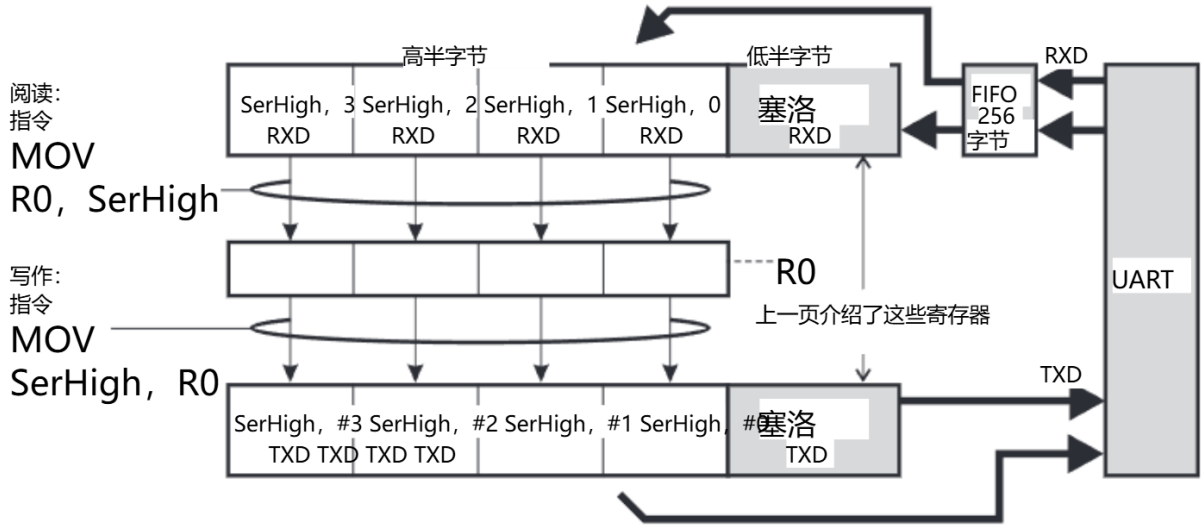
Note that the automatic transmission occurs only when the low nibble is written to **SerLow-w**, and not when the high nibble is written to **SerHigh-w**. For that reason, it is a good practice to write to the high nibble register (**SerHigh-w**) first.

When **UART** receives the data byte, it writes the byte into the **256-byte FIFO** (First In - First Out) buffer. If the whole **FIFO** buffer is full (**256** unread bytes), the following bytes are lost and the **Error** (**SerCtrl,#3**) flag is set. In the concurrent process, if the **SerLow-r** register is empty (data nibble was read), the new byte is written to **SerLow-r** (low nibble) and **SerHigh-r** (high nibble). At that moment, the register **Received** will be automatically decremented.

Note that all that happens automatically. All that the user's program has to do, is to test if the register **Received** is greater than **0** and, if it is (which is the sign that the new byte arrived from the serial port), read the high nibble (**SerHigh-r**) first, and then the low nibble (**SerLow-r**). Reading the low nibble (**SerLow-r**) resets the flag **RxRdy** automatically. This is not the case with high nibble reading (**SerHigh-r**), so it is a good practice to read the high nibble (**SerHigh-w**) first.

特殊功能寄存器SFRF 7（SerHigh） . 0xF7

产品描述： 寄存器SerHigh由两个共享同一地址的4位寄存器组成，但其中一个为只读，另一个为只写。当接收到串行数据字节时，第一个自动加载高半字节，另一个用于写入将要传输的数据字节的高半字节。有一个类似的寄存器对SFR 6，用于数据字节的低半字节。



此处表示的所有寄存器的上电和运行默认值为0000。

操作：这不是单个寄存器，而是一对寄存器。它们位于同一地址，但其中一个是只读的，另一个是只写的。我们在这里将它们称为SerHigh-r和/或SerHigh-w，尽管它们共享相同的名称

传输是一个简单的操作，因为处理器和处理器之间的握手协议很简单。处理器测试发送器是否准备就绪（如果最后一个字节已发送），如果是，则将SerHigh-w（高半字节）和SerLow-w（低半字节）写入8位发送缓冲器。如果它还没有准备好，它会等到传输完成。这可能会导致程序流程的某些减慢。

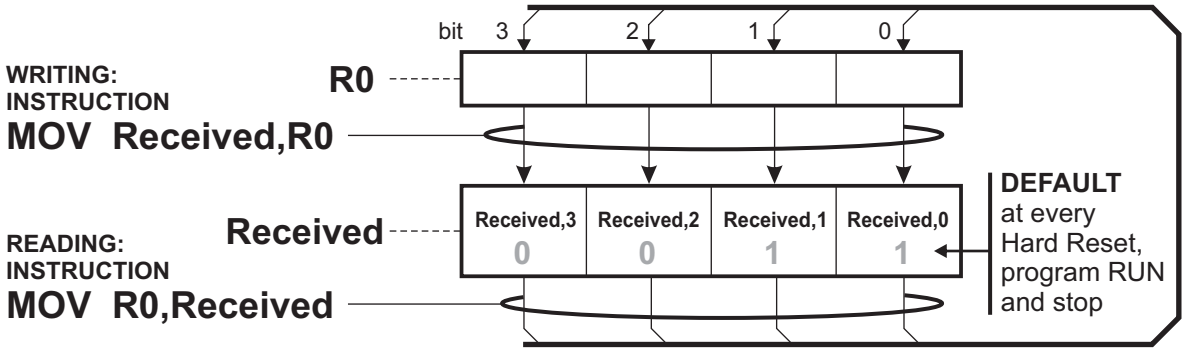
注意，仅当低半字节被写入SerLow-w时才发生自动传输，而当高半字节被写入SerHigh-w时不发生自动传输。因此，最好先写入高位半字节寄存器（SerHigh-w）。

当UART接收到数据字节时，它会将该字节写入256字节FIFO（先进先出）缓冲区。如果整个FIFO缓冲区已满（256个未读字节），则丢失后续字节，并设置错误（SerCtrl，#3）标志。在并发进程中，如果SerLow-r寄存器为空（数据半字节已读取），则新字节将写入SerLow-r（低半字节）和SerHigh-r（高半字节）。此时，接收的寄存器将自动递减。

请注意，所有这些都是自动发生的。用户程序所要做的就是测试寄存器Received是否大于0，如果大于0（这是新字节从串行端口到达的标志），则首先读取高位半字节（SerHigh-r），然后读取低位半字节（SerLow-r）。阅读低半字节（SerLow-r）自动复位标志RxRdy。对于高半字节阅读（SerHigh-r），情况并非如此，因此最好先读取高半字节（SerHigh-w）。

Special Function Register SFRF8 (Received) 0xF8

Description: Register **Received** contains 4-bit count of bytes in the **FIFO** queue plus one byte in **SerLow**-**SerHigh** pair (only **SerLow** contains the internal handshaking logic, and **SerHigh** is assumed). The capacity of FIFO and SerLow - SerHigh pair is **257** bytes total, so register **Received** shows the real number for **0-14** bytes, and state **"15"** means **"15 or more"**.



Operation: register **Received** is incremented every time when the byte is received and transferred from th **UART** to the **FIFO** memory, which is in the **Data Memory** of the microcontroller, and driven by the firmware. When the nibble is read from the register **SerLow**, register **Received** is decremented. So it shows the number of received, but unread bytes.

Handshaking between **SerLow**-**SerHigh** pair and **FIFO** buffer is performed internally, and the firmware takes care about it. All that user's program has to do is to read the state of register **Received** and test its zero-state: if it is greater than **0**, reading is allowed and user is sure that it will not be read more than once.

The capacity of **FIFO** and **SerLow**-**SerHigh** pair is **257** bytes total, and the register **Received** shows the real number for **0-14** bytes, and state **"15"** means **"15 or more"**.

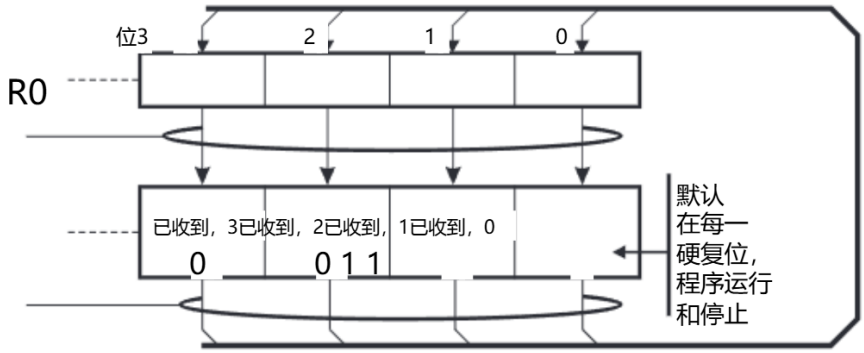
Note: After reading from **SerLow**, register **Received** will be decremented automatically. User's program should never attempt to modify it.

特殊功能寄存器SFRF 8（接收） 0xF8

描述：接收寄存器包含FIFO队列中的4位字节计数，
SerLow-SerHigh对中的一个字节（只有SerLow包含内部握手逻辑，而SerHigh是假定的）。FIFO和SerLow-SerHigh对的容量总共为257字节，因此寄存器Received显示0-14字节的真实的数，状态“15”表示“15或更多”。

写作：
指令
MOV接收，R 0

阅读：
指令
MOV R 0，已接收



操作方式：每次接收到字节并将其从FIFO存储器传输到FIFO存储器时，寄存器Received递增，FIFO存储器位于微控制器的数据存储器中，并由固件驱动。当从寄存器SerLow读取半字节时，寄存器Received递减。所以它显示已接收但未读的字节数。

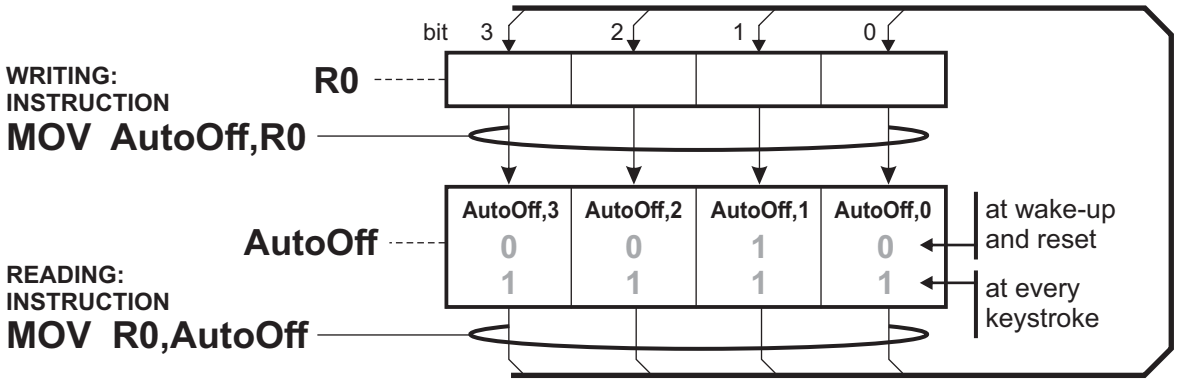
SerLow-SerHigh对和FIFO缓冲器之间的握手是在内部执行的，固件会处理它。所有用户的程序要做的就是读取寄存器Received的状态并测试其零状态：如果它大于0，则允许阅读，并且用户确信它不会被读取多次。

FIFO和SerLow-SerHigh对的容量总共为257字节，寄存器Received显示0-14字节的真实的数，状态“15”表示“15或更多”。

注：从SerLow阅读后，寄存器Received将自动递减。
用户的程序永远不应该试图修改它。

Special Function Register SFRF9 (AutoOff) 0xF9

Description: Register **AutoOff** is the 4-bit down-counter which is decremented automatically at every **10 minutes**. When it reaches zero, the device is internally rendered to **SLEEP** mode.



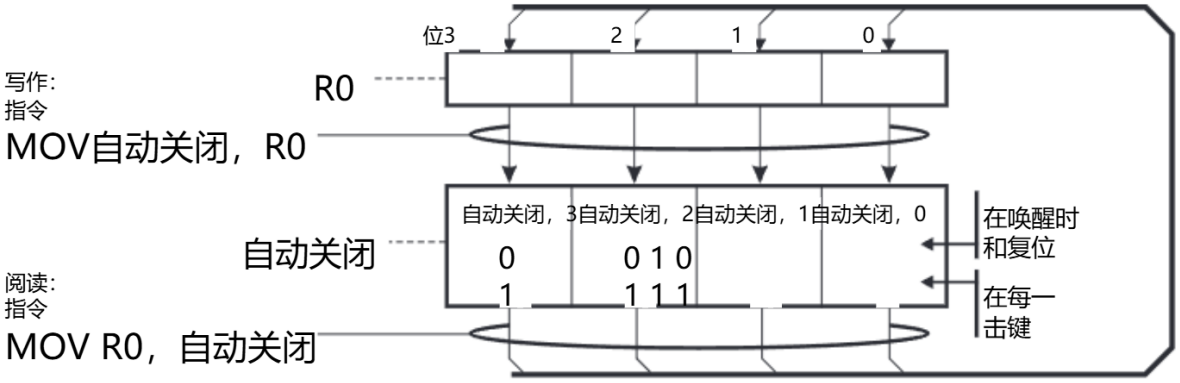
Operation: Register **AutoOff** is the **4-bit counter** which is decremented by **1** on every **10 minutes**. It is driven by the internal prescaler which divides the system oscillator frequency and thus generates one pulse at every **10 minutes**. Every time when the register **AutoOff** is loaded with the new value, the prescaler is preset to the full **10 minutes** period. The only exception is when the user's program writes 0 to the register **AutoOff**, then the prescaler is is not pre-loaded, which means that the unit will be shut off (rendered to **SLEEP** mode) immediately.

When the register **AutoOff** reaches zero, the device is internally rendered to **SLEEP** mode. It is the equivalent of switching the unit off by pressing **OFF-ON** button.

Register **AutoOff** allows that the user's program can keep the unit always **ON** (writing some high value, e.g. **1111** often enough) or to switch it off immediately, writing the zero value.

特殊功能寄存器SFRF 9（自动关闭） 0xF9

产品描述： 寄存器AutoOff是4位递减计数器，每10分钟自动递减一次。当它达到零时，设备在内部呈现为睡眠模式。



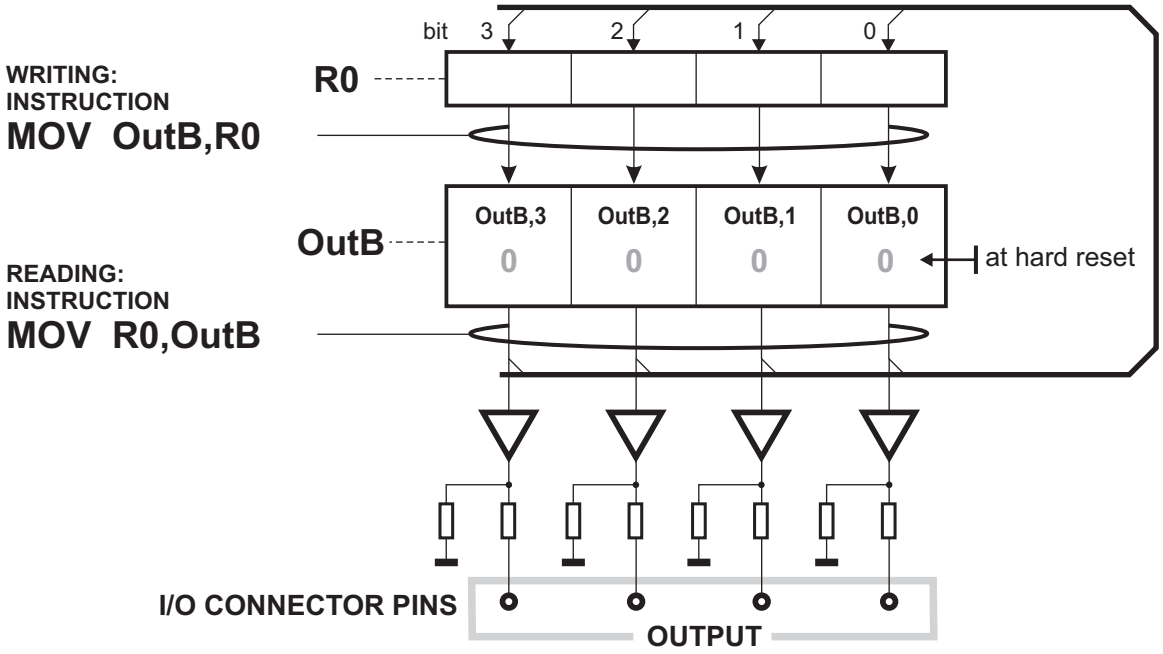
操作：寄存器AutoOff是4位计数器，每10分钟递减1。它由内部预分频器驱动，预分频器对系统振荡器频率进行分频，因此每10分钟产生一个脉冲。每次寄存器AutoOff加载新值时，预分频器预设为完整的10分钟周期。唯一的例外是，当用户程序将0写入寄存器AutoOff时，预分频器未预加载，这意味着该单元将立即关闭（呈现为SLEEP模式）。

当寄存器AutoOff达到零时，器件内部呈现为SLEEP模式。这相当于按下关闭-打开按钮关闭装置。

寄存器AutoOff允许用户的程序保持设备始终处于ON状态（经常写入某个高值，例如1111），或者立即将其关闭，写入零值。

Special Function Register SFRFA (OutB) 0xFA

Description: Register **OutB** is the alternate 4-bit latch for output port, active when the **WrFlag,1** bit (**IOPos**) is set.

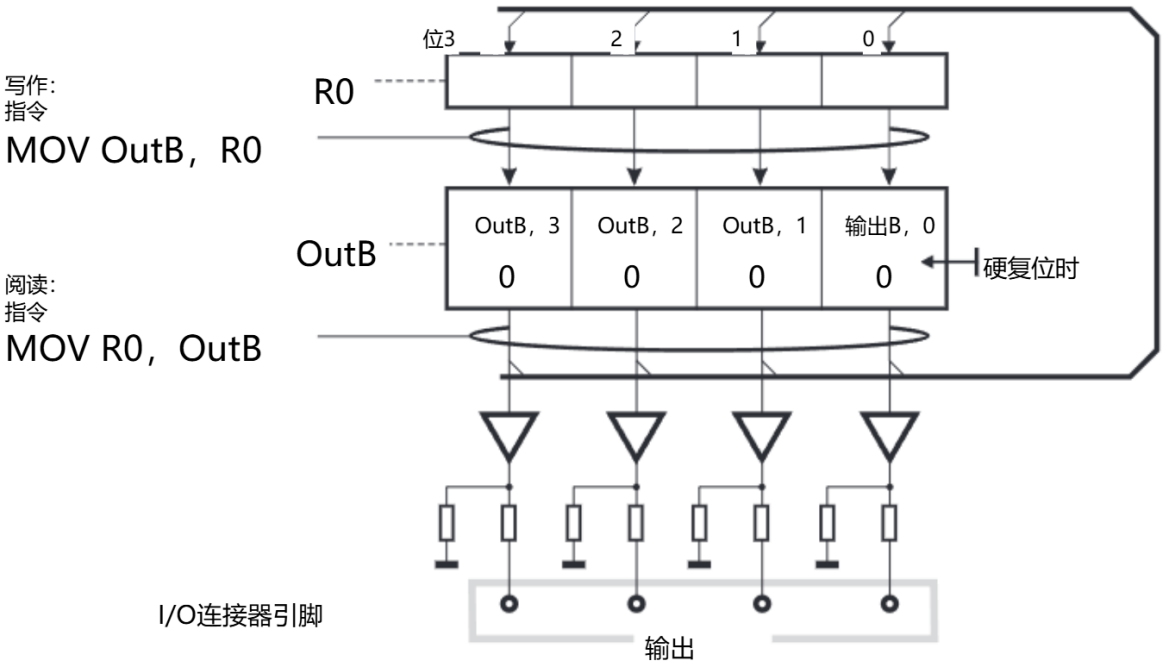


Operation: This register is the output latch. Contents written in the register **OutB** will appear on the connector output pins as logic levels.

Note: This register is active only when the **WrFlag,1** bit (**IOPos**) is set. Otherwise, this register may be used as the normal memory location, and the register **Out** (address **0x0A**) is active.

特殊功能寄存器SFRFA（OutB） 0xFA

产品描述： 寄存器OutB是输出端口的备用4位锁存器，在设置1位IOPos标志时有效。

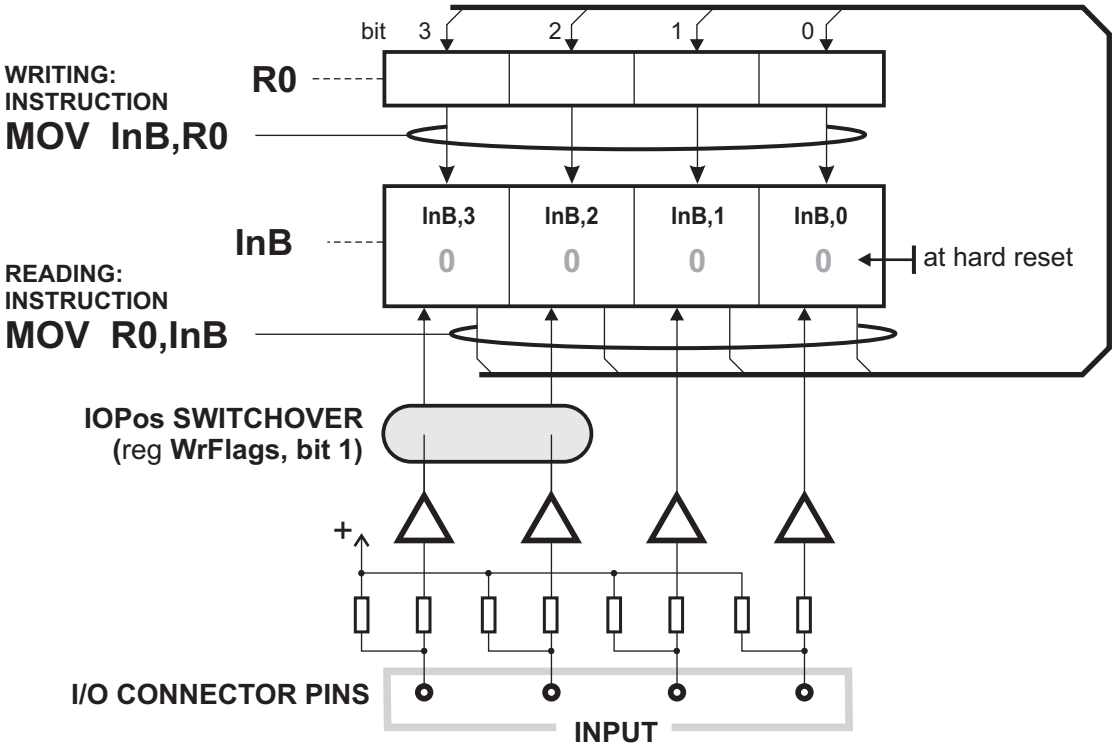


操作：该寄存器是输出锁存器。写入寄存器OutB的内容将作为逻辑电平出现在连接器输出引脚上。

注：此寄存器仅在1位IOPos标志置1时有效。否则，此寄存器可用作正常存储器位置，寄存器Out（地址0x0A）有效。

Special Function Register SFRFB (InB) 0xFB

Description: Register **InB** is the alternate **In** register, active when the **WrFlag,1** bit (**IOPos**) is set.



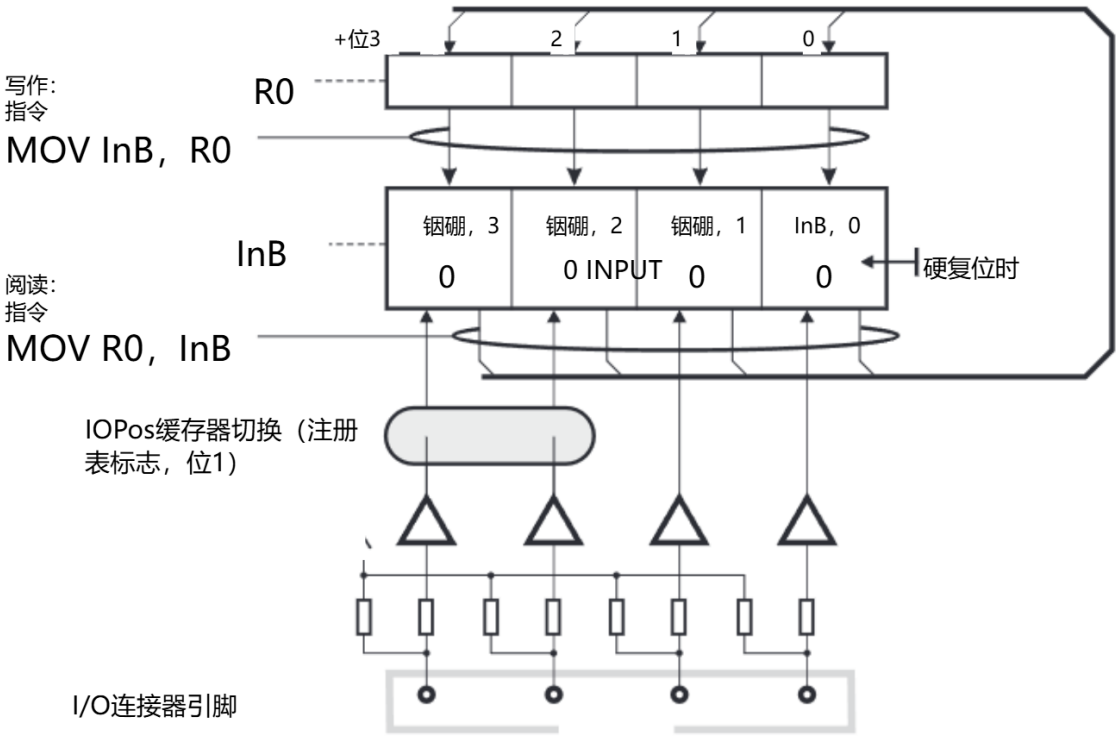
Operation: This register is the input port register. Logical levels on the connector input pins are always transferred to the register **InB**.

Note: This register is active only when the **WrFlag,1** (bit **IOPos**) is set. Otherwise, this register may be used as the normal memory location. If the **WrFlag,1** (bit **IOPos**) is set, the register **In** (address **0x0B**) is active.

Note: When the register **InB** is selected as the input port (when the **WrFlag,1** (bit **IOPos**) is set), writing to this register is possible, but the contents will be instantly overwritten. So it makes no sense, except for dummy writes.

特殊功能寄存器SFRFB（InB） 0xFB

产品描述： 寄存器InB是备用In寄存器，在1位IOPos标志置1时有效。



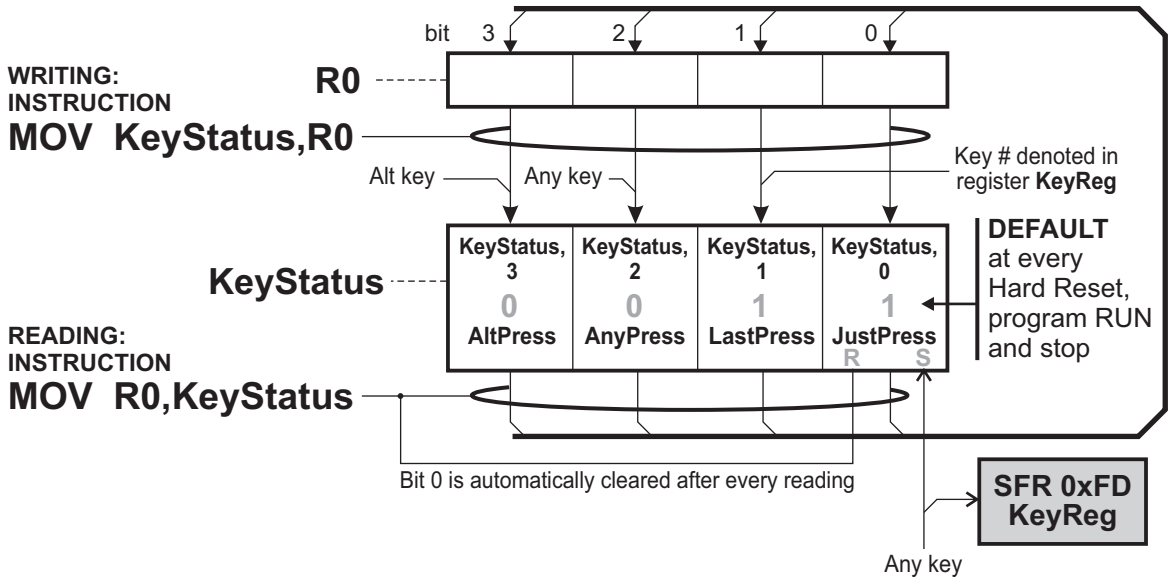
操作：此寄存器是输入端口寄存器。连接器输入引脚上的逻辑电平始终传输到寄存器InB。

注：此寄存器仅在设置了“I/O标记”1（位IOPos）时有效。否则，该寄存器可用作正常存储器位置。如果设置了寄存器标志1（位IOPos），则寄存器In（地址0x 0 B）有效。

注意事项：当选择寄存器InB作为输入端口时（当设置了RISK Flag，1（位IOPos）时），可以写入该寄存器，但内容将立即被覆盖。所以它没有意义，除了虚拟写入。

Special Function Register SFRFC (KeyStatus) 0xFC

Description: Register **RS3** contains individual statuses for registers **RS0, RS1** and **RS2** in bits **RS3,#2, RS3,#1** and **RS3,#0**. If the contents of these registers is **0000**, the corresponding bit will be **0**. Otherwise, the corresponding bit will be **1**.



Operation: Register **KeyStatus** allows the program to read the status of certain buttons of the keyboard.

Bit 3: **AltPress = 0** Button ALT not pressed (debounced, 5 ms delay)
AltPress = 1 Button ALT pressed (debounced, 5 ms delay)

Bit 2: **AnyPress = 0** No button is pressed (buttons ALT and ON-OFF not tested)
AnyPress = 1 At least one button is pressed (buttons ALT and ON-OFF not tested)

Bit 1: **LastPress = 0** The last pressed button (written in register KeyReg) is not pressed
LastPress = 1 The last pressed button (written in register KeyReg) is pressed

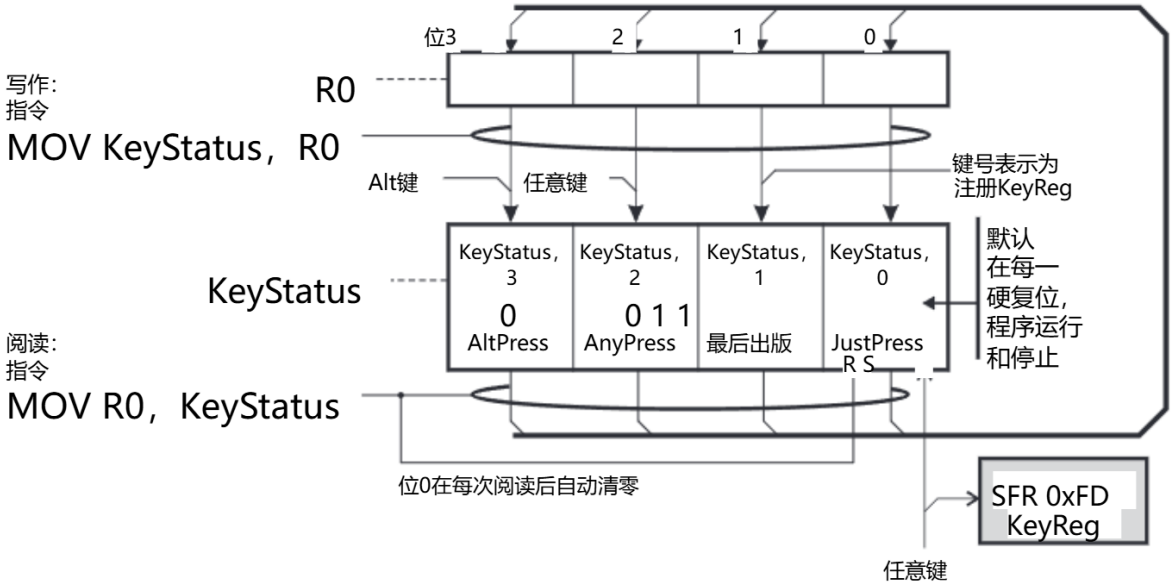
Bit 0: **JustPress = 0** No button was pressed after last reading of this register.
JustPress = 1 Button (which is written in register KeyReg) was pressed, and this bit is automatically reset after reading this register. So this bit can be read only once after every keypress.

Bits **3**, **2** and **1** show the momentarily state of buttons and every of them is automatically reset to **0** as soon as the condition for **1** is not satisfied any more

Bit **0** is the handshaking bit: once it is set, it will not be reset until this register is read. Then it is automatically reset, which means that the state “1” of this bit can be read only once after every keypress.

特殊功能寄存器SFRFC（KeyStatus） 0xFC

产品描述： 寄存器RS3在位RS3，#2、RS3，#1和RS3，#0中包含寄存器RS 0、RS 1和RS 2的各个状态。如果这些寄存器的内容为0000，则相应的位为0。否则，对应的位将为1。



操作：注册KeyStatus允许程序读取键盘上某些按钮的状态。

位3：AltPress = 0 未按下ALT按钮（去抖动，5 ms延迟）按下ALT按钮
AltPress = 1 （去抖动，5 ms延迟）

位2：AnyPress = 0 未按下按钮（未测试ALT和ON-OFF按钮）
AnyPress = 1 至少按下一个按钮（未测试ALT和ON-OFF按钮）

位1：LastPress = 0 未按下最后按下的按钮（写入寄存器KeyReg）按下最后按下的按钮（写入寄存器KeyReg）
最后按下= 1

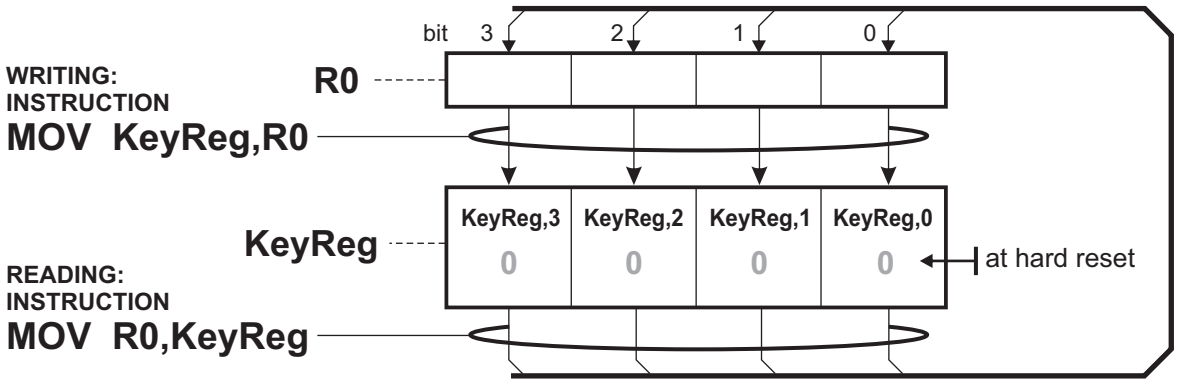
位0：JustPress = 0 最后一次阅读该寄存器后，未按下按钮。按下按钮（写入寄存器KeyReg），阅读此寄存器后，此位自动复位。所以这个位在每次按键后只能读一次。
JustPress = 1

位3、2和1显示按钮的瞬时状态，一旦不再满足1的条件，每个按钮都会自动重置为0

位0是握手位：一旦置1，在读取该寄存器之前不会复位。然后它会自动复位，这意味着该位的状态“1”在每次按键后只能读取一次。

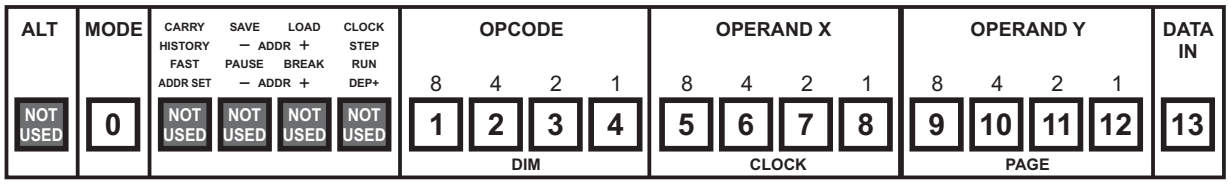
Special Function Register SFRFD (KeyReg) 0xFD

Description: Register **KeyReg** contains the number of the last pressed button.



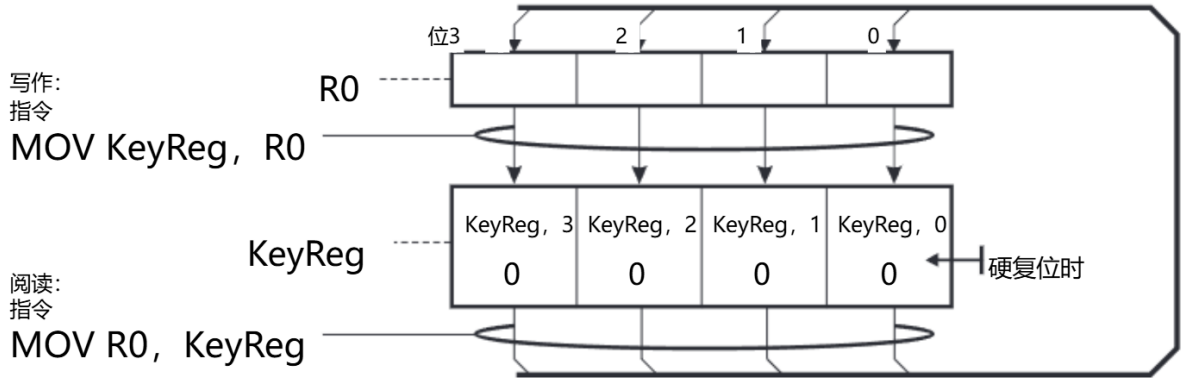
Operation: This register contains the number of the last pressed button. The contents will not change when the button is released, but only when the next button is pressed.

Here is the list of buttons, with the corresponding numbers. Pressing **On-Off**, **ALT** or any button in the Mode Specific Command group do not affect this register. These buttons are marked as “NOT USED” here.



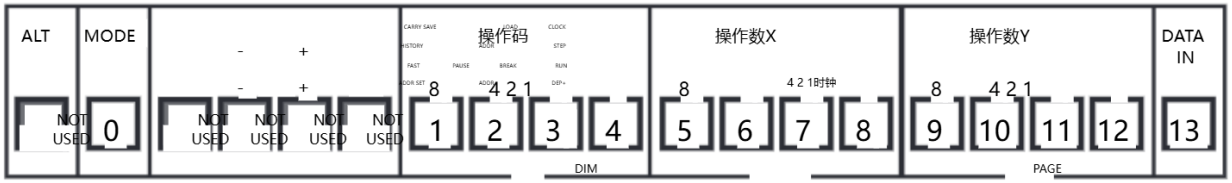
特殊功能寄存器SFRFD（KeyReg）　　0xFD

产品描述：　　Register KeyReg包含最后按下的按钮的编号。



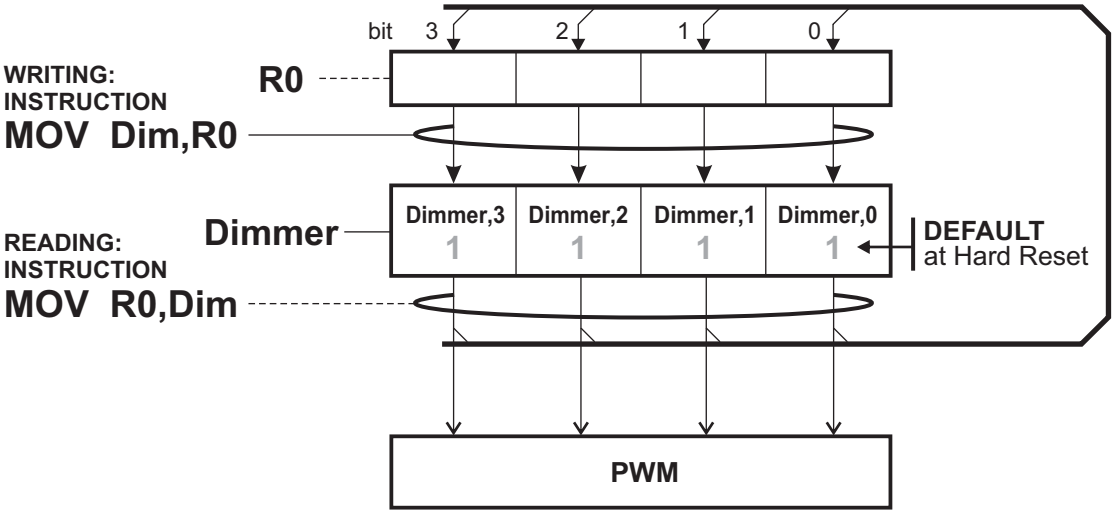
操作：该寄存器包含最后按下的按钮的编号。释放按钮时，内容不会改变，只有按下一个按钮时才会改变。

下面是按钮列表，以及相应的编号。按下开-关、ALT或模式特定命令组中的任何按钮不会影响此寄存器。这些按钮在此处标记为“未使用”。



Special Function Register SFRFE (Dimmer) 0xFE

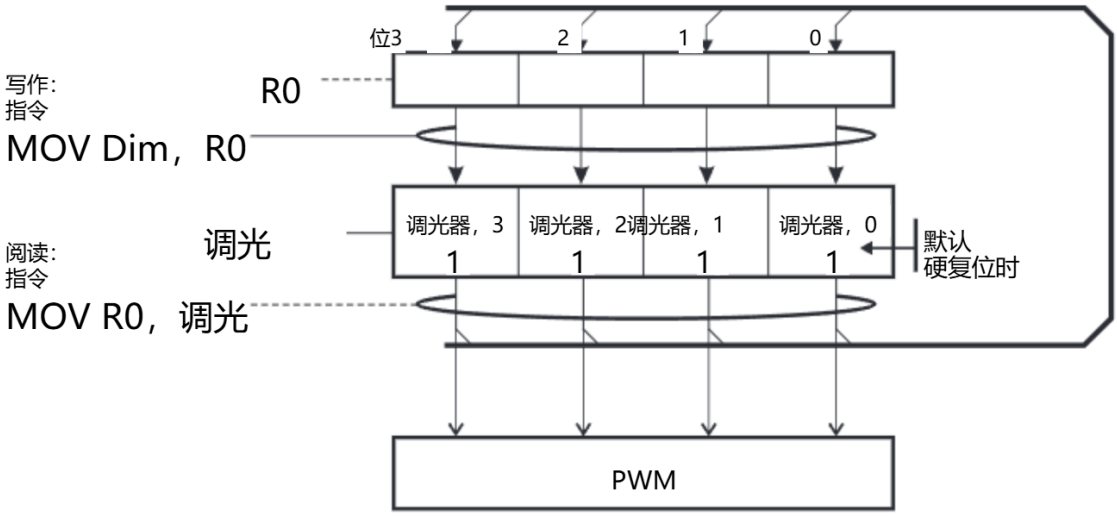
Description: Register **Dim**mer contains 4-bit value for the PWM LED intensity control.



Operation: All LEDs are arranged in 16×17 matrix, which is under the software control of the system microcontroller **PIC24FJ256GA704-I/PT**. Register **Dimmer** controls the **Pulse Width Modulation** ratio for **ON/OFF** state of all LEDs.

Value 1111 results in the 100% duty cycle, which is the brightest setting. Value "0000" results in the lowest light level.

产品描述： 寄存器调光器包含用于PWM LED亮度控制的4位值。

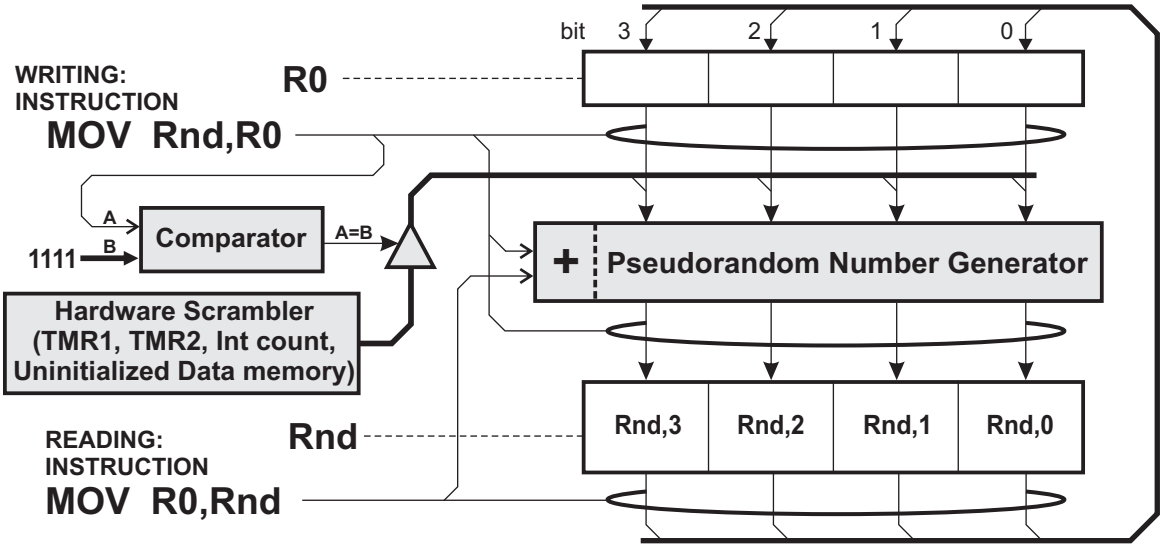


操作：所有LED均以16×17矩阵排列，由系统微控制器PIC 24 FJ 256 GA 704-I/PT软件控制。寄存器调光器控制所有LED的开/关状态的脉宽调制比。

值1111导致100%占空比，这是最亮的设置。值“0000”导致最低的光照水平。

Special Function Register SFRFF (Rnd) 0xFF

Description: Register **Rnd** contains the dynamic **4-bit** pseudorandom value.



Operation: **4**-bit pseudorandom value is generated by the **32**-bit **Linear Congruential Generator**, which uses the formula:

$$X_{n+1}=(a \times X_n + c) \bmod 2^{32} \qquad (a = 0x41C64E6D,\ c = 0x6073)$$

Seed X_0 is generated at **Master Reset**, using the uninitialized **Data Memory** data at startup. The same **seed** can be reinitialized to the known value by writing to the register **Rnd**. This value contains only **4** bits, but it is written in all **8 nibbles** of **Rnd**, so all the subsequent values are predictable. For instance, if the value written to **Rnd** is **0001** binary, the value of **32**-bit Seed will be

00010001 00010001 00010001 00010001.

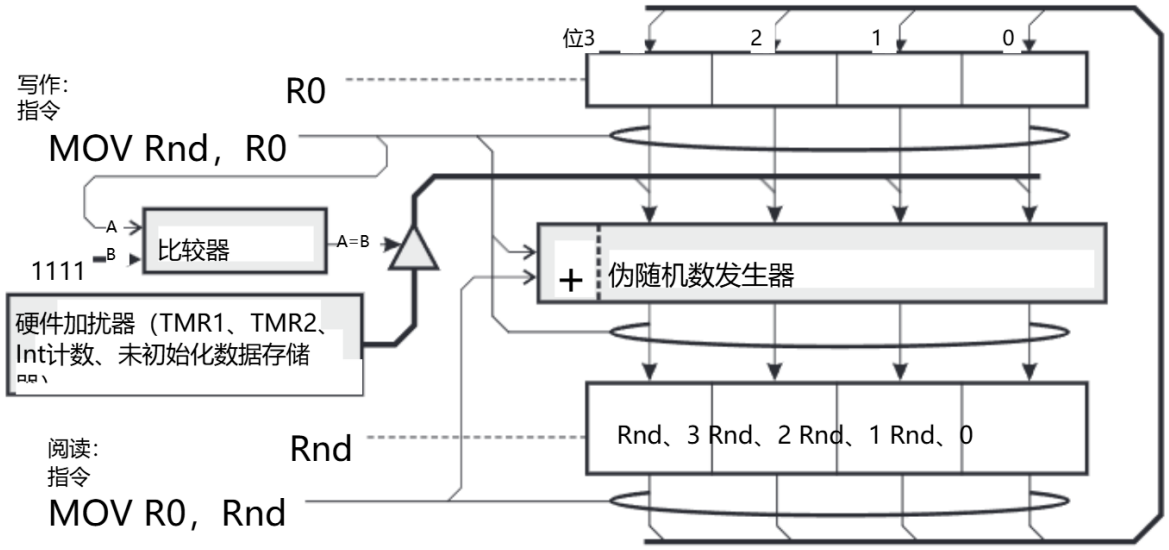
There is an exception, when the binary value written to the register **Rnd** is **1111** (decimal **15**), the full scrambled value will be rewritten to all **32** bits of **Seed**, and all the subsequent numbers will be unpredictable. This is the default behaviour of the **Rnd** register.

After every read from the SFR register **Rnd**, the new pseudorandom cycle $X_{n+1}=(X_n \times 0x41C64E6D + 0x6073) \bmod 2^{32}$ is executed and bits **3-0** of **Seed** are rewritten to the register **Rnd**.

There is one more a way to initiate the Pseudorandom cycle and write the new unpredictable value to the register **Rnd**. When the unit is in **SS** mode, and the register **Page** is **1111** (decimal **15**), every press of the button **Data In** will cause the new value in the register **Rnd**.

Note: Default value for **Rnd** register is randomly preset at power-up.

产品描述： 寄存器Rnd包含动态4位伪随机值。



操作：4位伪随机值由32位线性同余发生器生成，它使用以下公式：

$$X_{n+1} = (a \times X + c) \bmod 2^{32} \qquad (a = 0x41C64E6D，c = 0x6073)$$

种子X在主复位时生成，使用启动时未初始化的数据存储器0数据。通过写入寄存器Rnd，可以将同一种子重新初始化为已知值。该值仅包含4位，但写入Rnd的所有8个半字节，因此所有后续值都是可预测的。例如，如果写入Rnd的值是0001二进制，则32位种子的值将为

00010001 00010001 00010001 00010001.

有一个例外，当写入寄存器Rnd的二进制值为1111（十进制15）时，完整的加扰值将被重写为Seed的所有32位，并且所有后续数字都将不可预测。这是Rnd寄存器的默认行为。

每次读取SFR寄存器Rnd后，新的伪随机周期执行$X = (X \times 0x\ 41\ C64\ E6\ D + 0x\ 6073) \bmod 2^{32}$，种子的位3-0 aren+1 n重写到寄存器Rnd。

还有一种方法可以启动伪随机循环并将新的不可预测值写入寄存器Rnd。当装置处于SS模式且寄存器Page为1111（十进制15）时，每次按下按钮Data In都会在寄存器Rnd中产生新值。

注：Rnd寄存器的默认值在上电时随机预设。