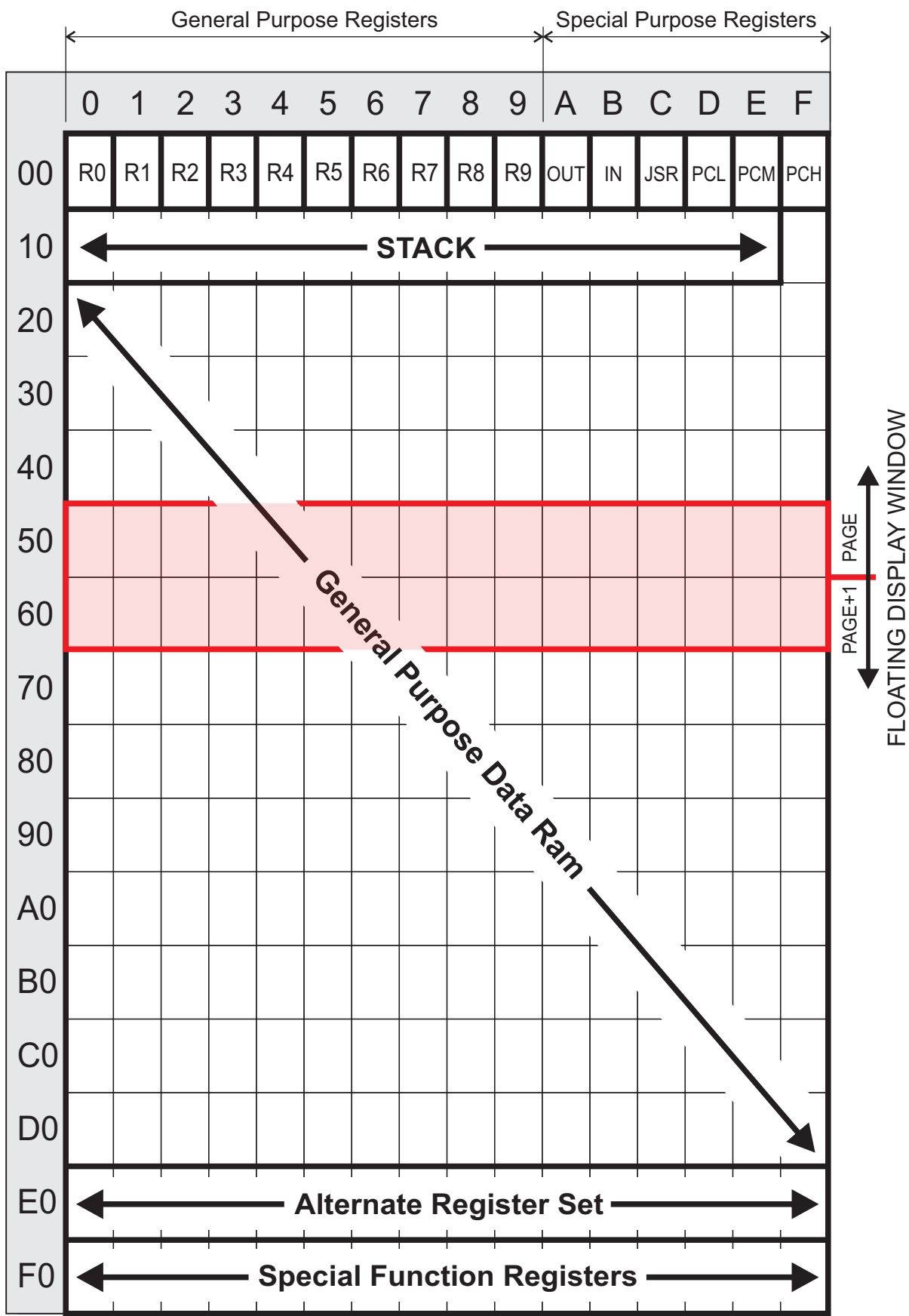


# **SPECIAL FUNCTION REGISTERS**

**Revision 2**  
Oct.12.2022

# Data Memory Organization



## Special Function Registers on Page 0

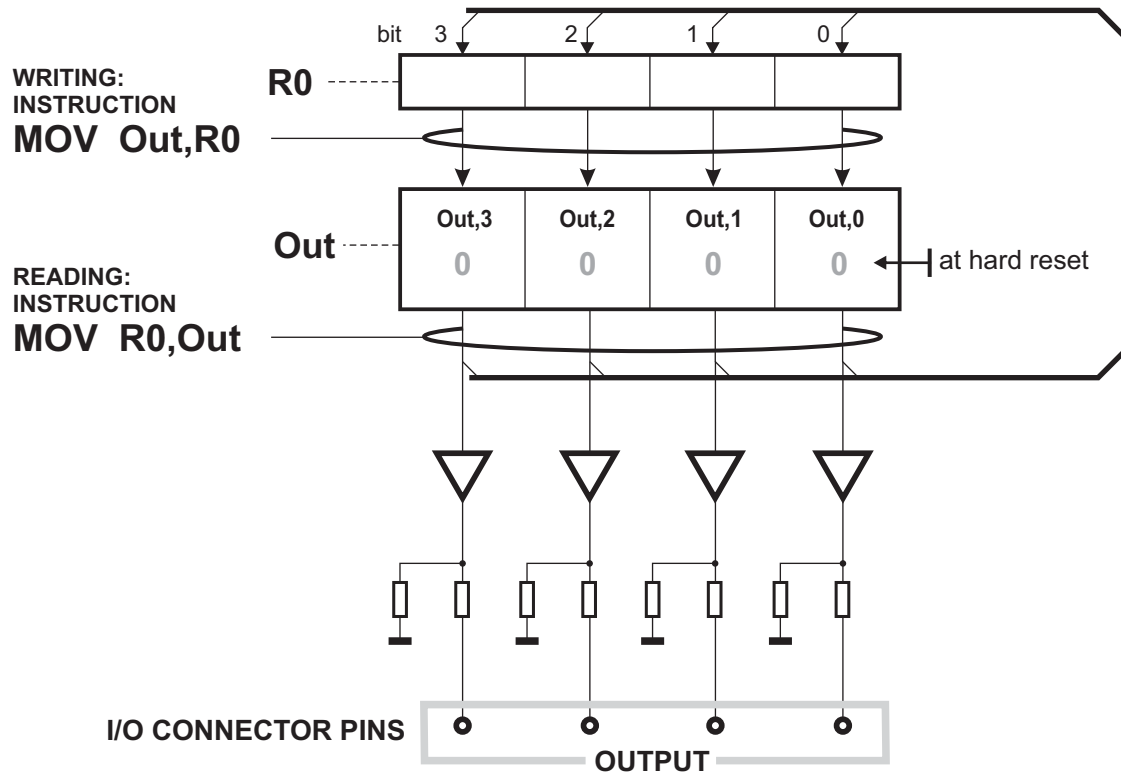
	bit IOPos (WrFlags,1) = 0	bit IOPos (WrFlags,1) = 1	DEF
F0	← R0 →	← R0 →	0000
F1	← R1 →	← R1 →	0000
F2	← R2 →	← R2 →	0000
F3	← R3 →	← R3 →	0000
F4	← R4 →	← R4 →	0000
F5	← R5 →	← R5 →	0000
F6	← R6 →	← R6 →	0000
F7	← R7 →	← R7 →	0000
F8	← R8 →	← R8 →	0000
F9	← R9 →	← R9 →	0000
FA	← Out →	← R10 →	0000
FB	← In →	← R11 →	0000
FC	← JSR →	← JSR →	0000
FD	← PCL →	← PCL →	0000
FE	← PCM →	← PCM →	0000
FF	← PCH →	← PCH →	0000

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

## 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.



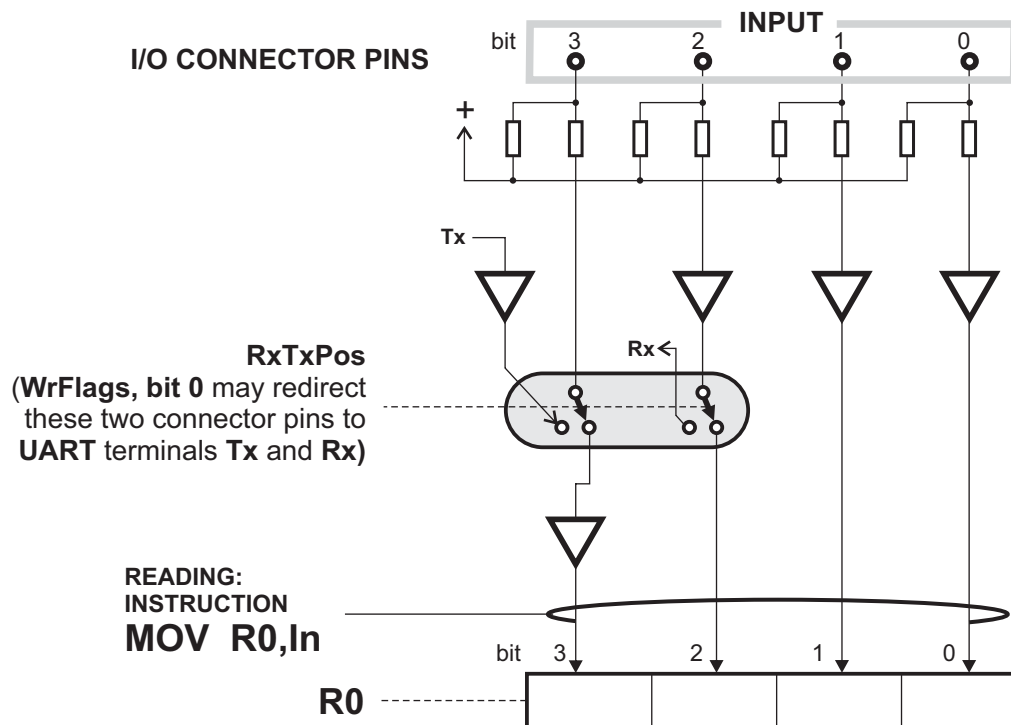
**Operation:** This register is the output latch. Contents written in the register **Out** will appear on the connector output pins as logic levels.

**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.

## 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 cleared. 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!

0x0C

The diagram illustrates the execution of the JSR instruction. It shows two main stages: **WRITING: INSTRUCTION** and **READING: INSTRUCTION**.

**WRITING: INSTRUCTION**  
**MOV JSR,R0**  
 The R0 register (bits 3, 2, 1, 0) is written into the JSR instruction. The JSR instruction is a 4-byte word where each byte contains the instruction code (JSR) and a 4-bit address (3, 2, 1, 0). At hard reset, the JSR register is initialized to 0.

**READING: INSTRUCTION**  
**MOV R0,JSR**  
 The JSR instruction is read from memory, and the 4-bit address (3, 2, 1, 0) is loaded into the R0 register.

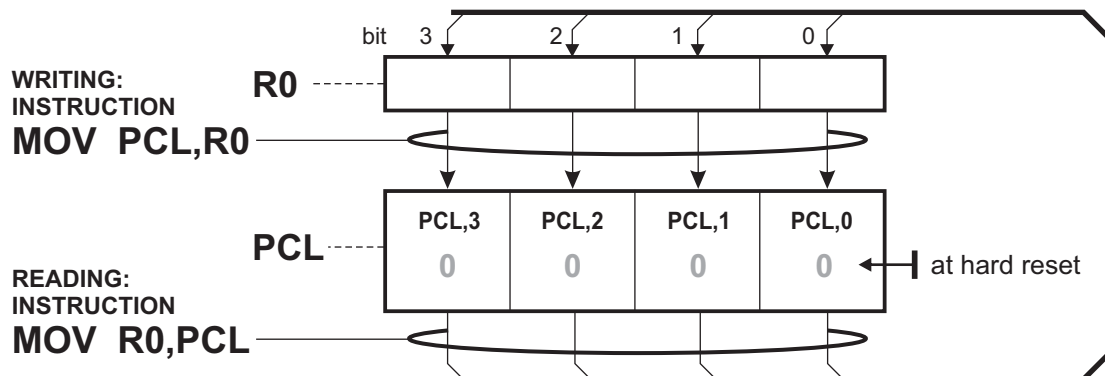
Note: In the following example, program calls the **2th level** of subroutine from the **1st level**, that's why the instruction increments **SP** from **1** to **2**. On the next **RET** execution, return address **0001 1100 1011** will be reloaded back to the **Program Counter**, and the program will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



## Special Function Register SFR0D (PCL)

0x0D

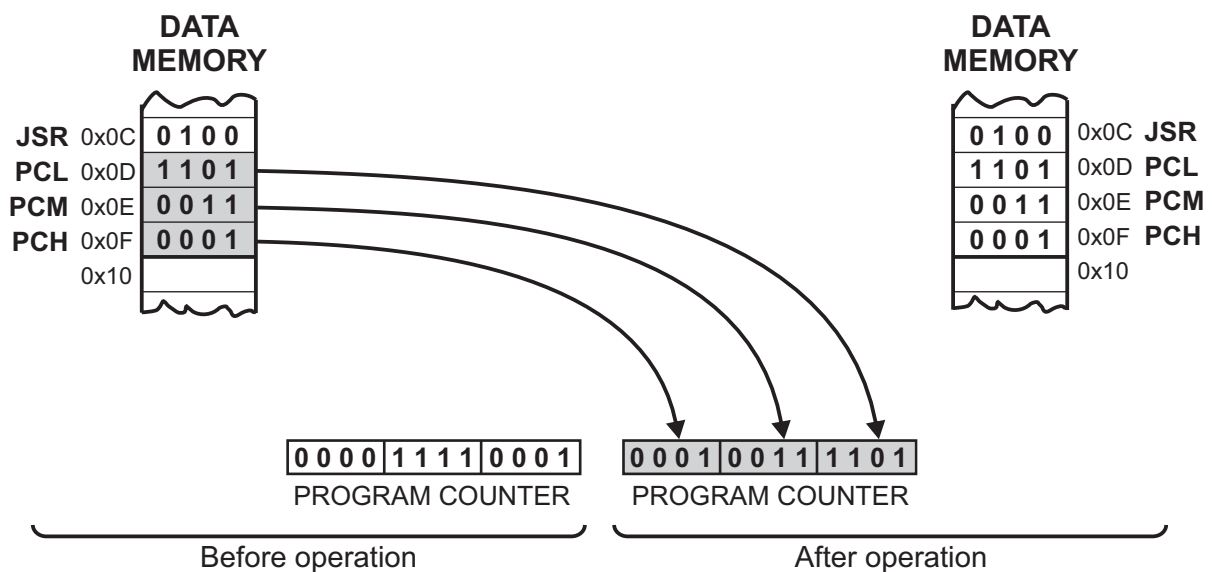
**Description:** Register **PCL** is the low-nibble address register for a program jump.



**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.

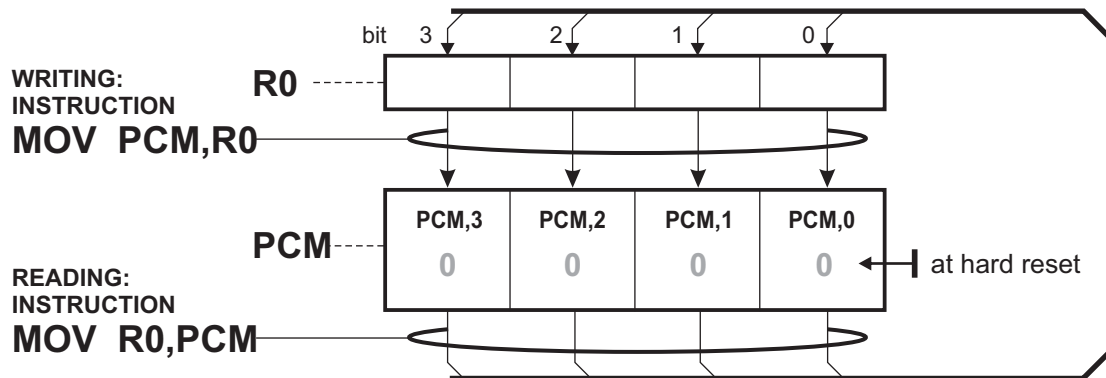
Note: In the following example, program jumps from the address **0001 0011 1100** (**Program Counter** always points to the next address) to the location **0001 0011 1101**, determined by registers **PCH** (high nibble), **PCM** and **PCL** (low nibble). Only **Program Counter** is modified, the contents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



## Special Function Register SFR0E (PCM)

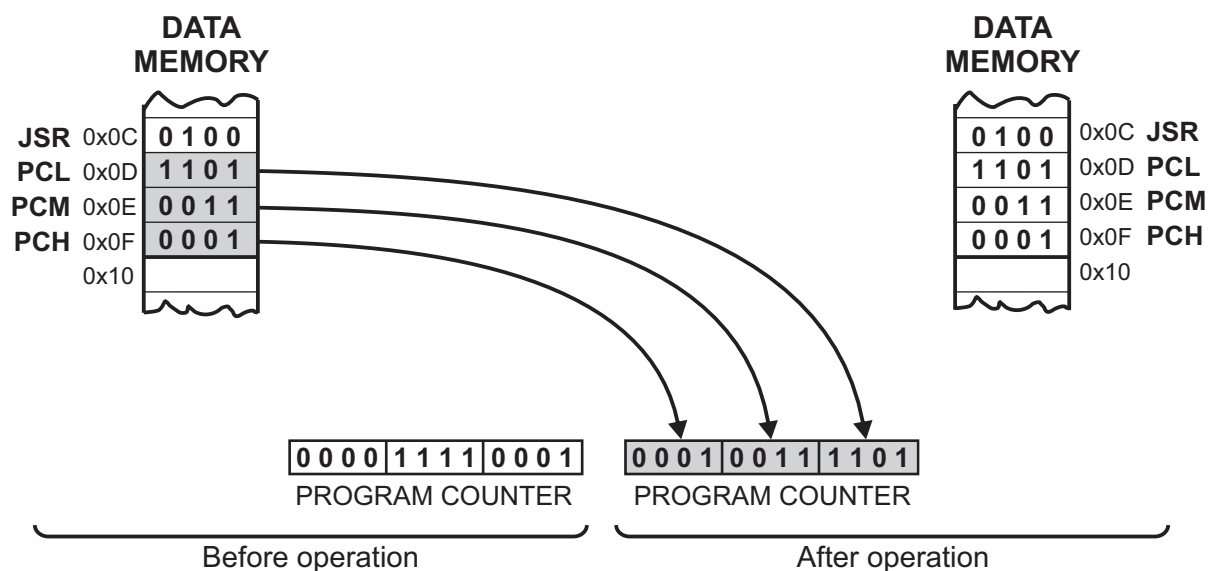
0x0E

**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 **PCM**) 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**.

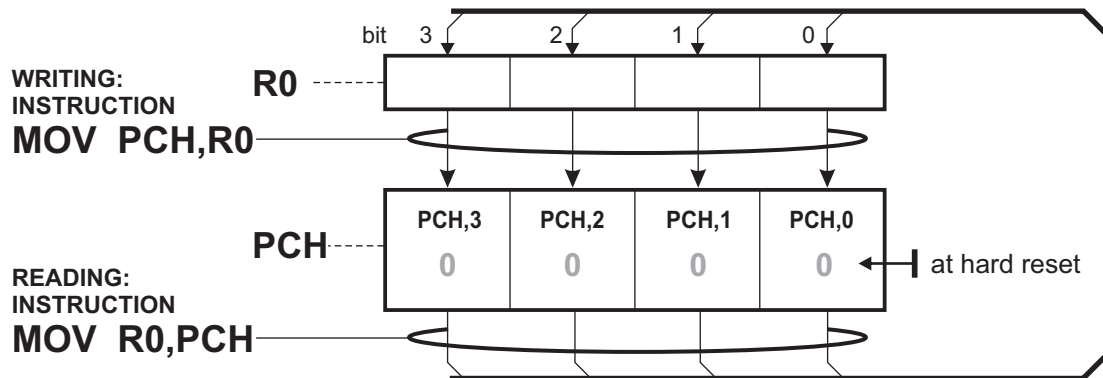




## Special Function Register SFR0F (PCH)

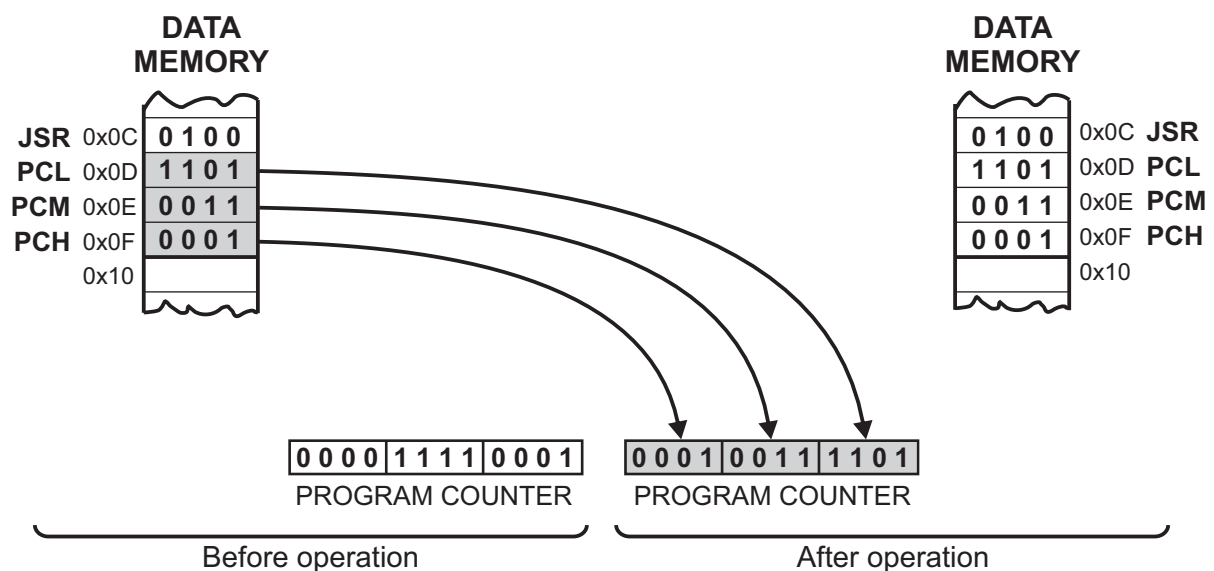
0x0F

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







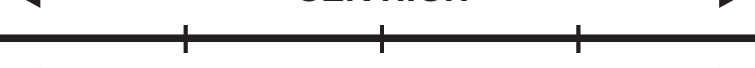








**Operation:** Reading from or writing to this register is a simple read or write operation, so it does not affect the program flow. However, this register will take action when program jump (writing to **PCH**) or subroutine call (writing to **JSR**) occurs, as it will be a part of the **Program Counter**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

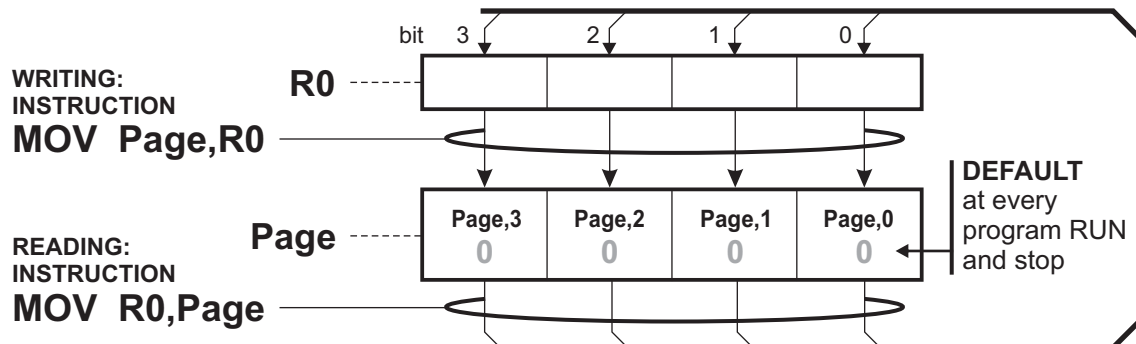


## Special Function Registers on Page 15

	NAME	bit 3	bit 2	bit 1	bit 0	DEF
F0	Page					0000
F1	Speed					0000
F2	Sync					0000
F3	WrFlags	LedsOff	MatrixOff	InOutPos	RxTxPos	0000
F4	RdFlags	Reserved	Reserved	Vflag	UserSync	0000
F5	SerCtrl	RxError				0011
F6	SerLow					0000
F7	SerHigh					0000
F8	Received					0000
F9	AutoOff					0010
FA	OutB					0000
FB	InB					0000
FC	KeyStatus	AltPress	AnyPress	LastPress	JustPress	0000
FD	KeyReg					0000
FE	Dimmer					0000
FF	Random					0000

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

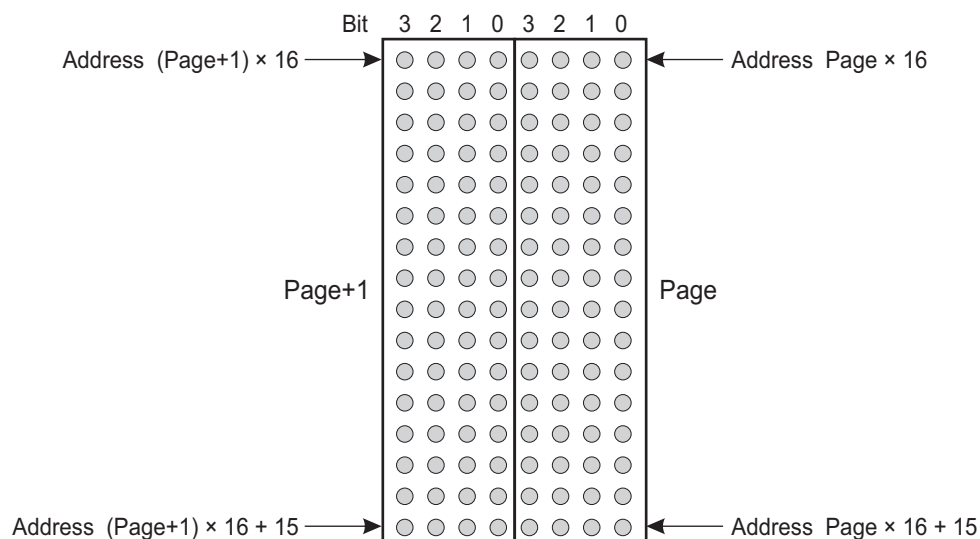
**Description:** Register **Page** selects which pages from the data memory will be displayed.



**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 half 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 half, with the same order.

Special case: If the selected page is **15** (which is the last page), the **Next Page** (displayed on the left half 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.

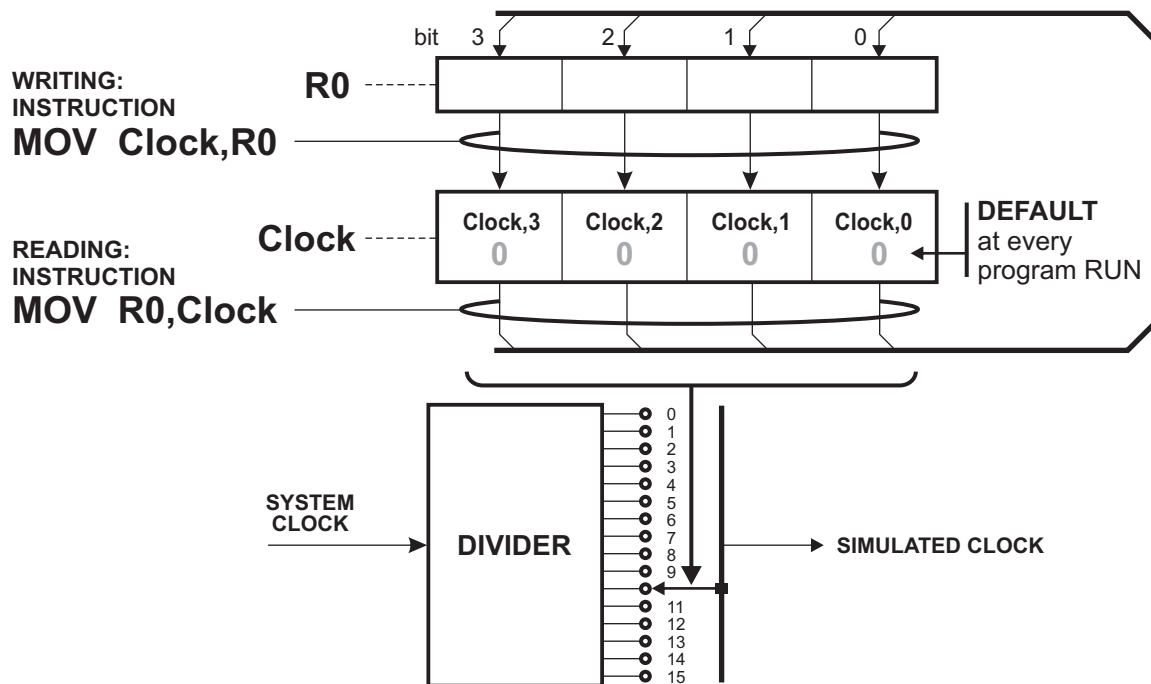
Note: Contents of SFR register **Page** can be easily modified by holding button **ALT** and using buttons in **Operand Y** field to select the value. While button **ALT** is depressed, only **Page 0** is displayed (which can be used to see the main register set on **Page 0** promptly), but when the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



## 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 $\mu$ s
1	100 KHz	~10 $\mu$ s
2	30 KHz	33 $\mu$ s
3	10 KHz	100 $\mu$ s
4	3 KHz	333 $\mu$ s
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  $\mu$ s** 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.

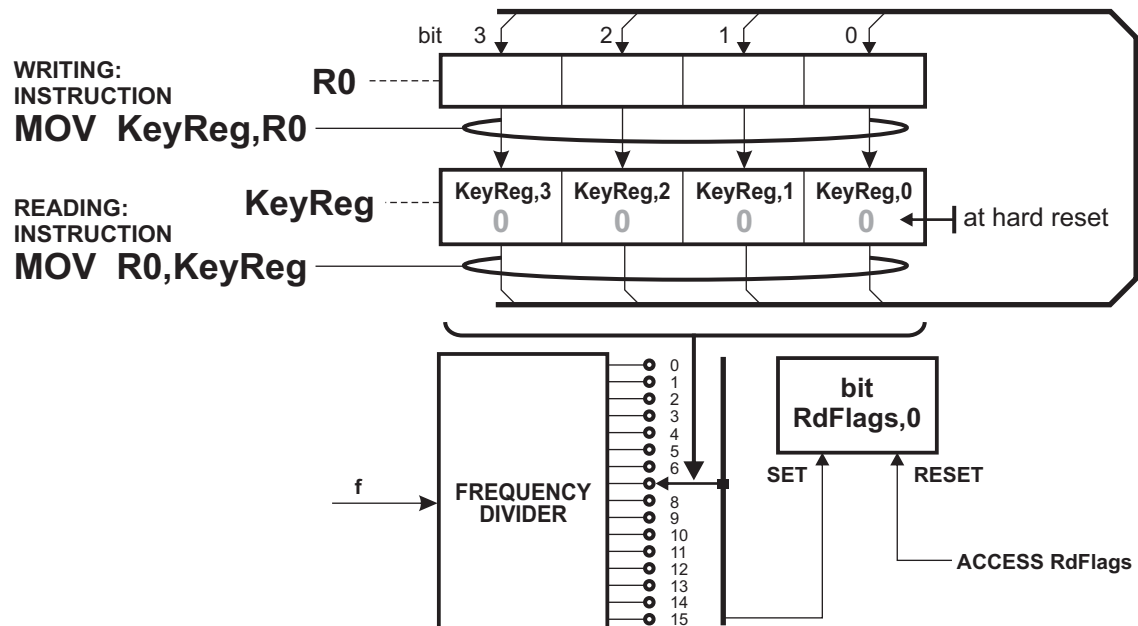
Note 2: On every **RUN** command, Clock parameter is initialized to **0000 (250 KHz)**.

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

## 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

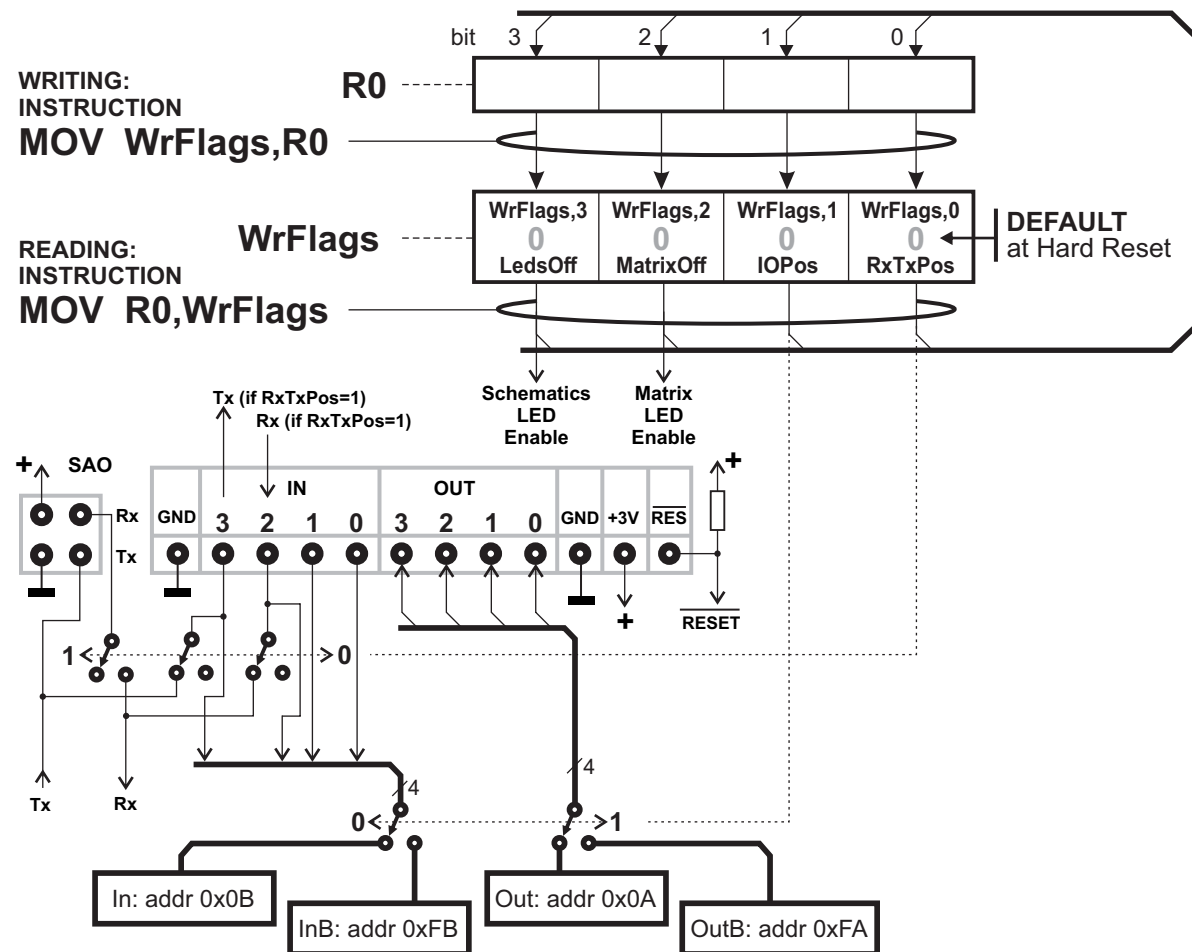
ADD    R3,R6      ; program goes on...
```

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

# Special Function Register SFRF3 (WrFlags)

0xF3

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



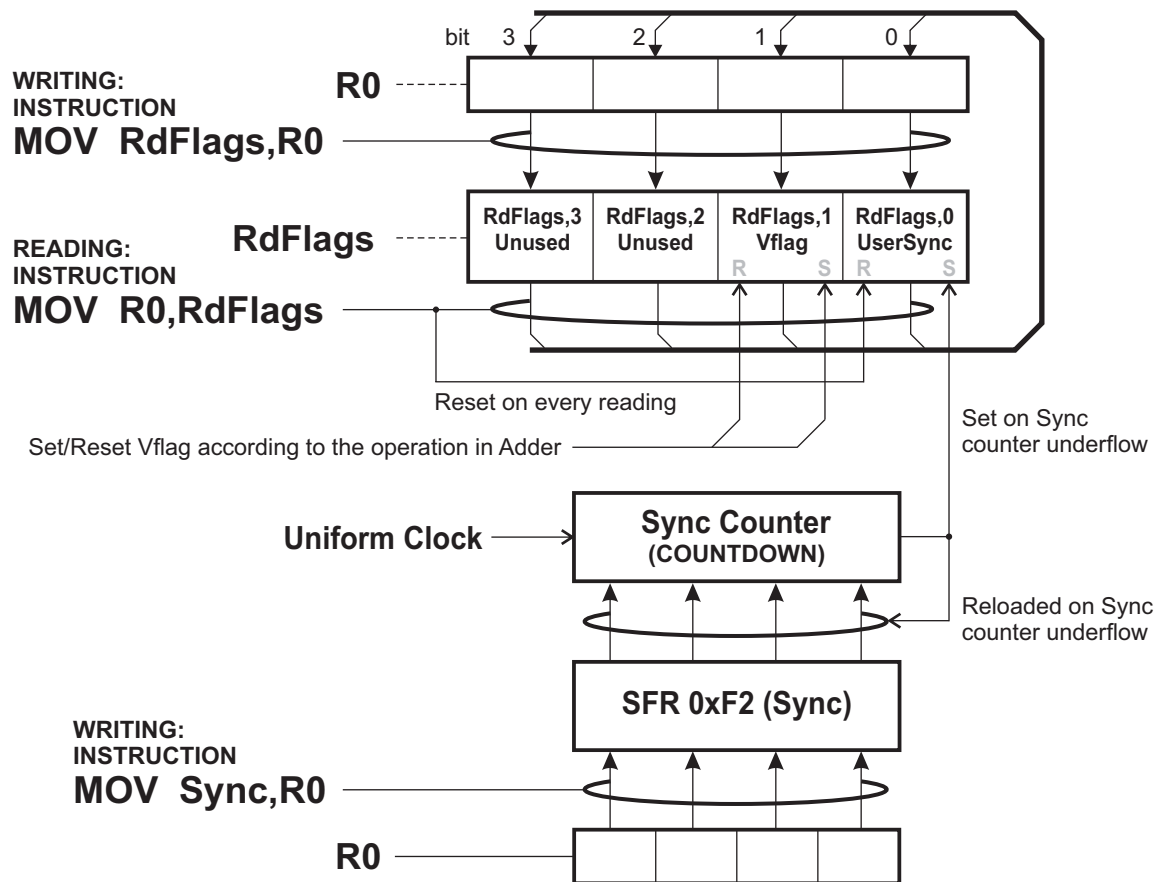
**Operation:** Every bit in the Register **WrFlags** has its own function:

- 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).

## 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)

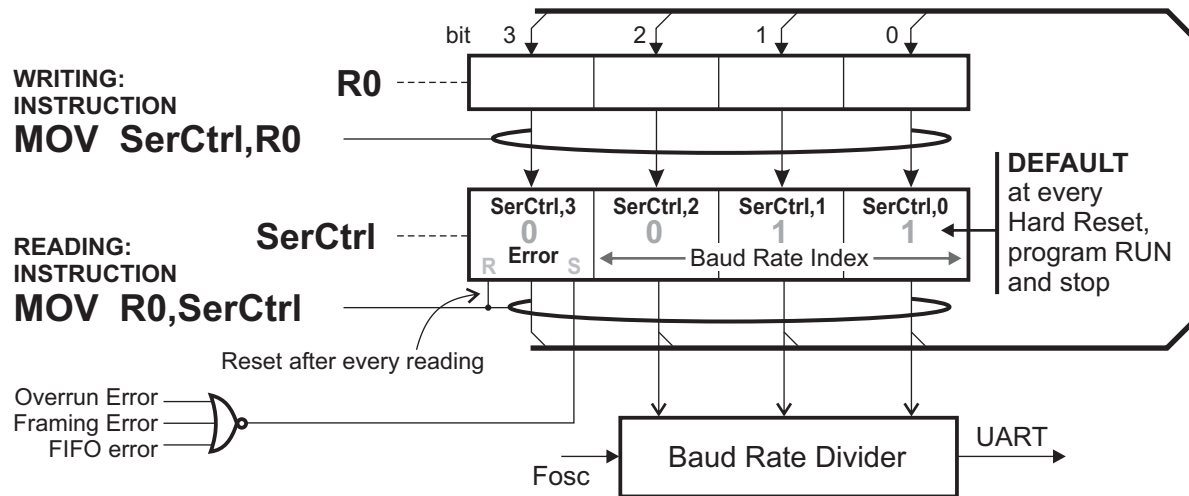
Bit 0: **UserSync = 0** **User Sync** didn't occur yet.  
**UserSync = 1** **User Sync** occurred. This bit is automatically reset upon reading of the register **RdFlags**.

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

## 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**.

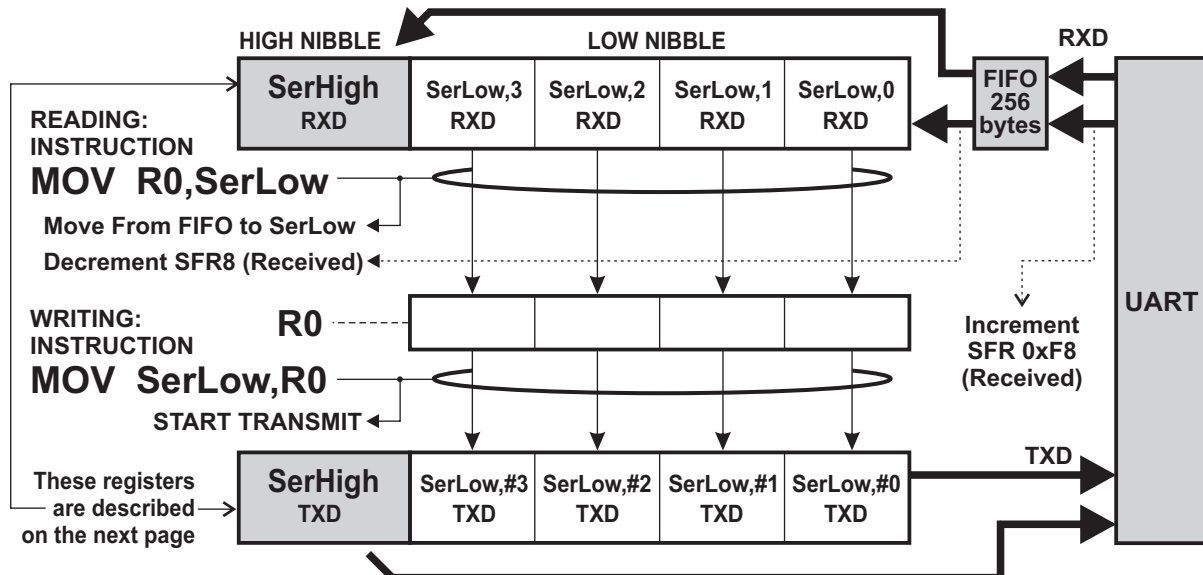
Here is the table of the available Baud rates:

#	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
4	1	0	0	19200
5	1	0	1	38600
6	1	1	0	57600
7	1	1	1	115200

← **DEFAULT** (points to row 3, Baud 9600)



**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 **SFR7** which is used for the high nibble of data byte.



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.

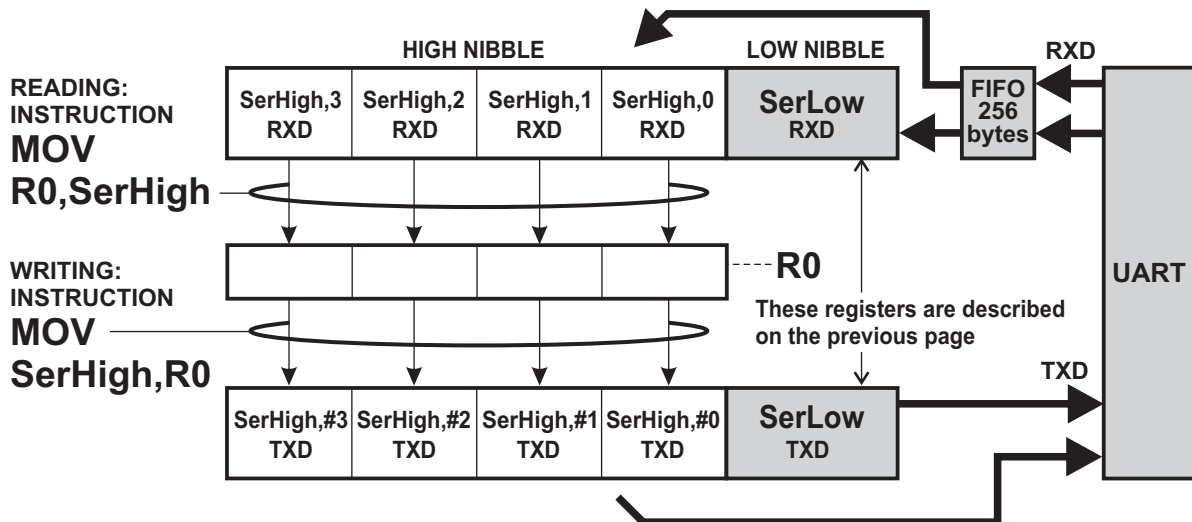
**Transmitting** is a simple operation, as the handshaking protocol between the processor and **UART** is straightforward. Processor tests if the transmitter is ready (if the last byte was transmitted) and, if so, writes **SerLow-w** (low nibble) and **SerHigh-w** (high nibble) to the **8-bit transmit 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.

**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.



**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.

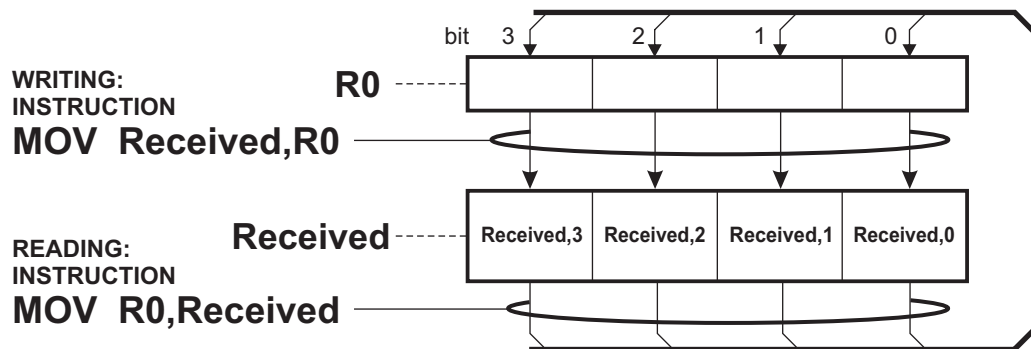
When **UART** receives the data byte, it writes the byte into the **256-byte FIFO** (First In - First Out) buffer. If the whole **FIFO** buffer is full (**256** unread bytes), the following bytes are lost and the **Error (SerCtrl,#3)** flag is set. In the concurrent process, if the **SerLow-r** register 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.

## 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 the **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.

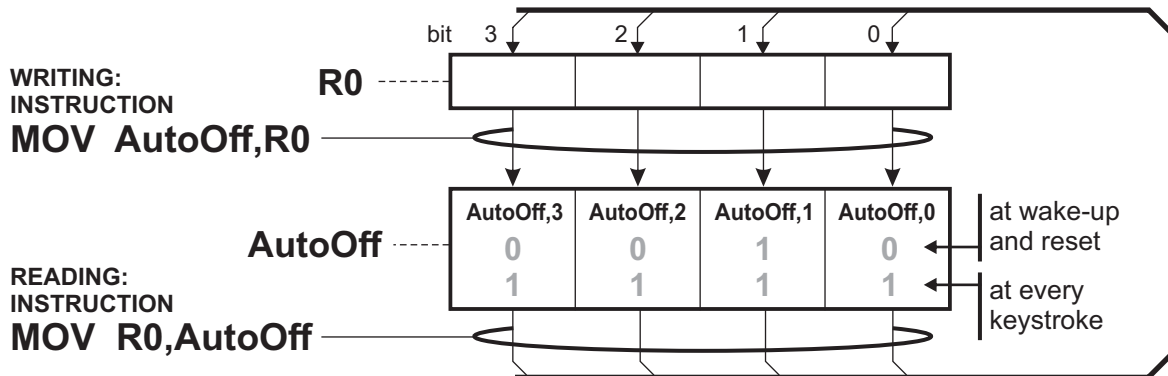
The capacity of **FIFO** and **SerLow-SerHigh** pair is **257** bytes total, and the register **Received** shows the real number for **0-14** bytes, and state “**15**” means “**15 or more**”.

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

## 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 not pre-loaded, which means that the unit will be shut off (rendered to **SLEEP** mode) immediately.

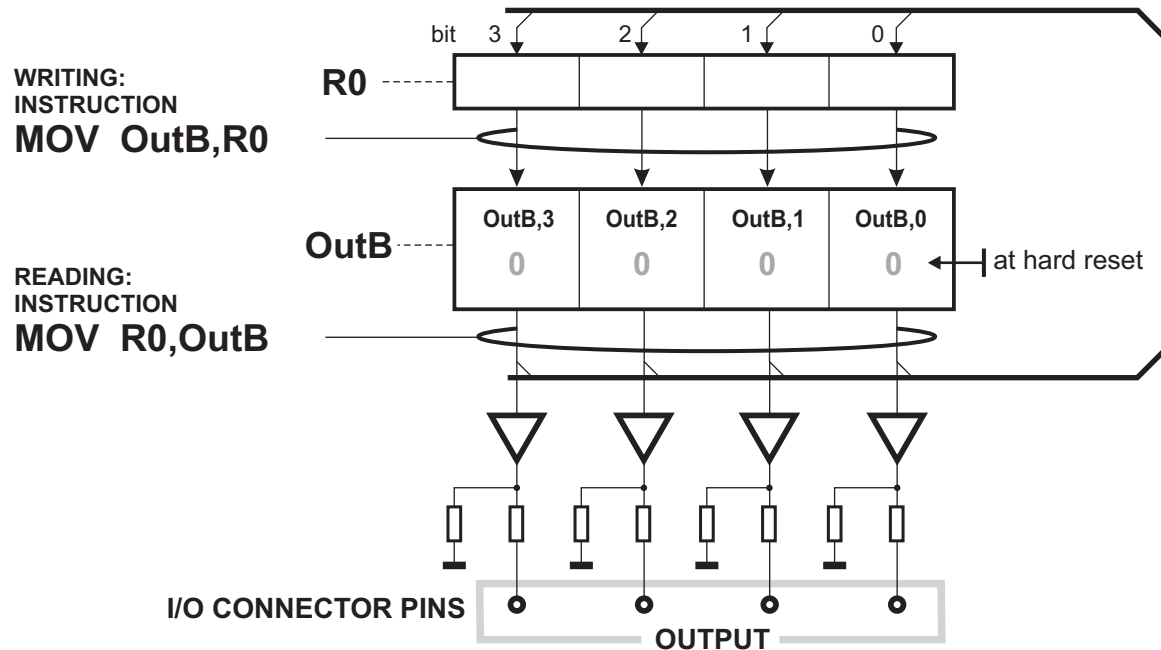
When the register **AutoOff** reaches zero, the device is internally rendered to **SLEEP** mode. It is the equivalent of switching the unit off by pressing **OFF-ON** button.

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.

## 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.



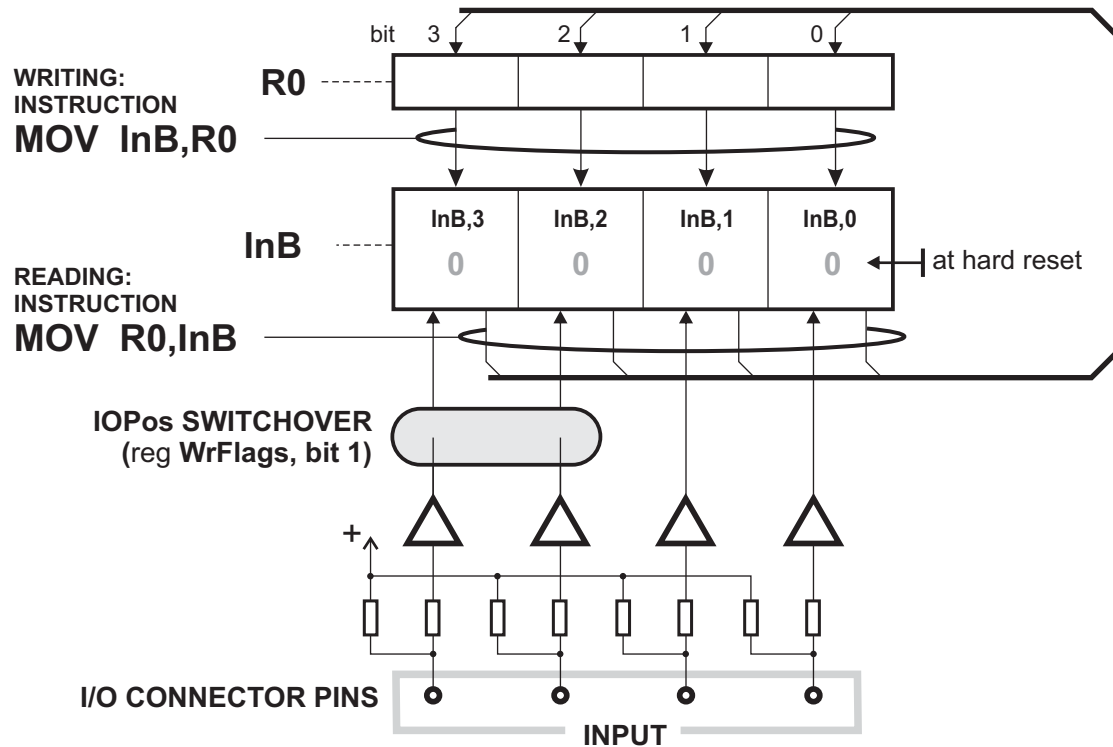
**Operation:** This register is the output latch. Contents written in the register **OutB** will appear on the connector output pins as logic levels.

**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.

## 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**.

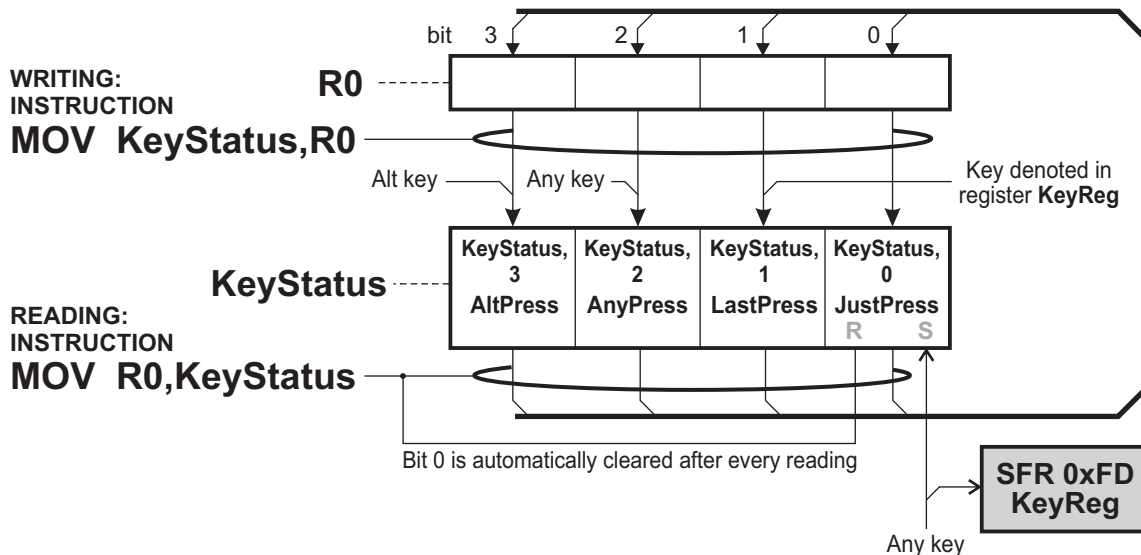
**Note:** This register is active only when the **WrFlag,1** (bit **IOPos**) is set. Otherwise, this register may be used as the normal memory location. If the **WrFlag,1** (bit **IOPos**) is set, the register **In** (address **0x0B**) is active.

**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.

## 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.

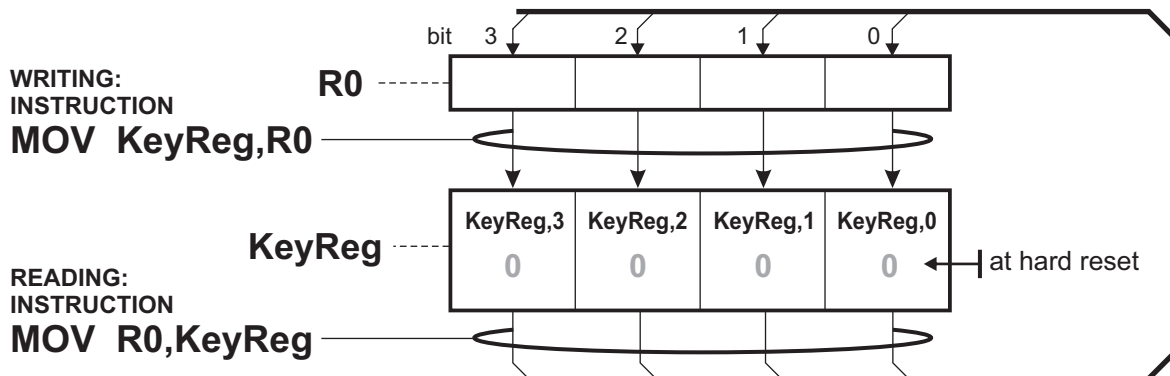
Bits **3**, **2** and **1** show the momentarily state of buttons and every of them is automatically reset to **0** as soon as the condition for **1** is not satisfied any more

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.

## 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 of **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.

ALT	MODE	CARRY HISTORY	SAVE FAST ADDR SET	LOAD - ADDR + PAUSE - ADDR +	CLOCK STEP RUN DEP+	OPCODE				OPERAND X				OPERAND Y				DATA IN
8	4	2	1	8	4	2	1	8	4	2	1	8	4	2	1			
NOT USED	1	NOT USED	NOT USED	NOT USED	NOT USED	2	3	4	5	6	7	8	9	10	11	12	13	14
						DIM				CLOCK				PAGE				

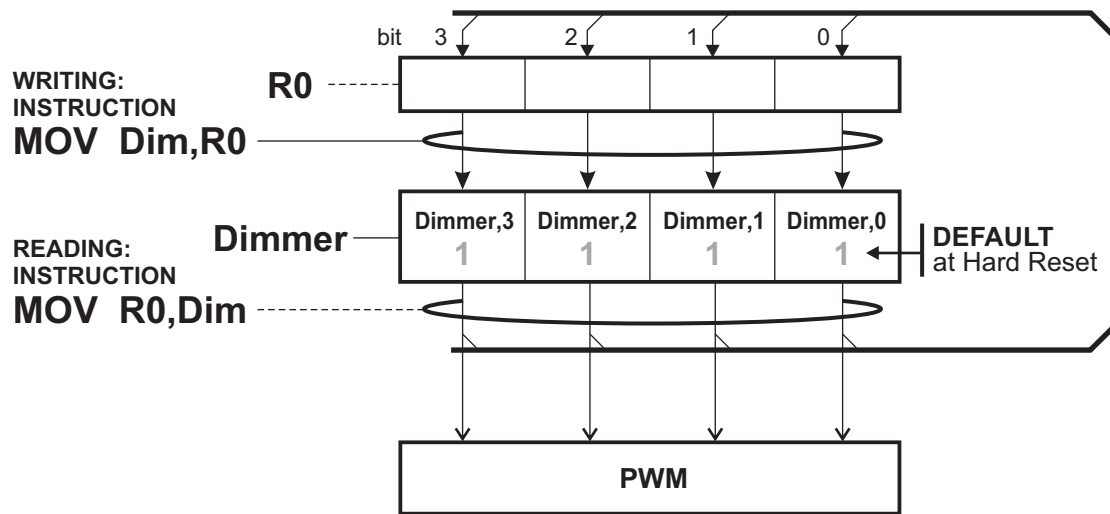
NOT USED	ON OFF
-------------	-----------



## Special Function Register SFRFE (Dimmer)

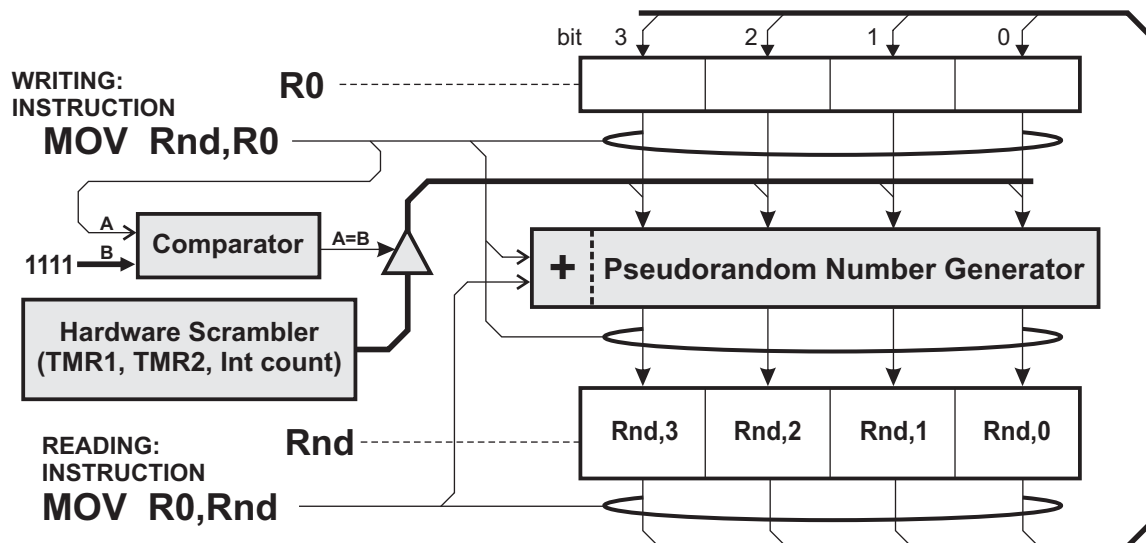
0xFE

**Description:** Register **Dimmer** 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.

**Description:** Register **Random** 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:

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

**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.

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**.