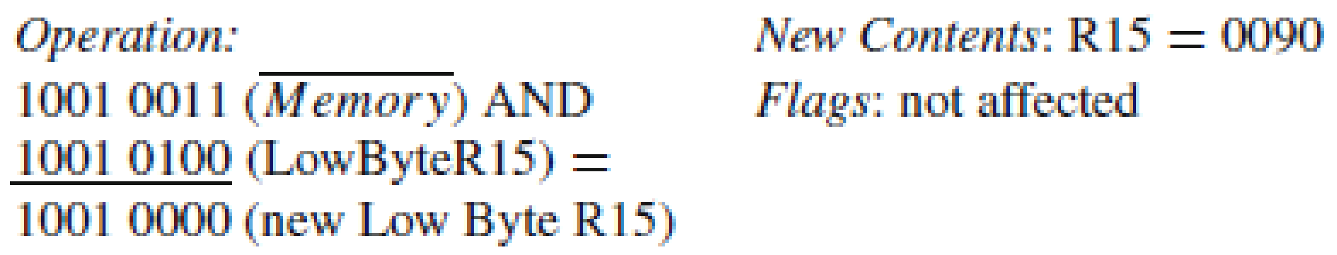
Bit Clear, Bitwise *Logic Instructions(Continuing)*

bic(.B or .W) src,dst; dst←(NOTsrc)ANDdst, clear bits in dst with mask src. **Flag is not effected.**

Ex. For the initial conditions, R12=25A3H, R15= 8B94H and [25A5H]= 6CH, what will be the content of R15 after the execution of the following program?

```
bic.b 2(R12),R15; R15←(R12+2)'AND R15
```



Operation:

1001 0011 ($\overline{Memory}$) AND

1001 0100 (LowByteR15) =

1001 0000 (new Low Byte R15)

New Contents: R15 = 0090

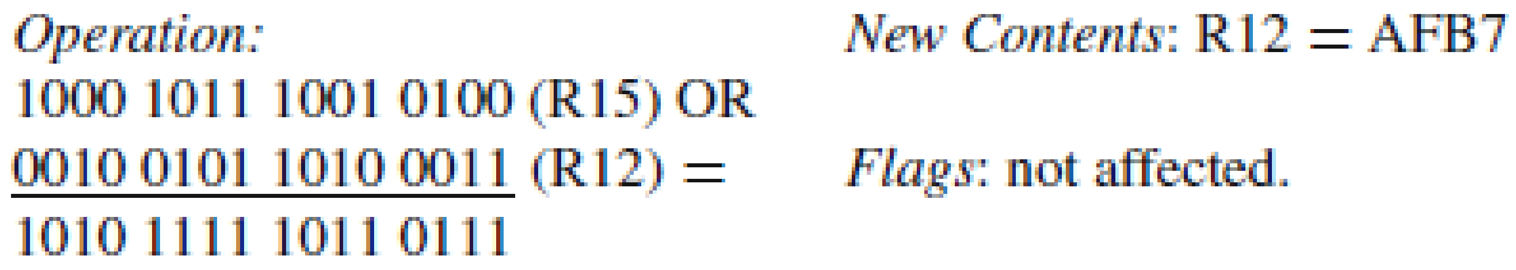
Flags: not affected

Bit Set

`bis(.B or .W) src,dst;` dst←srcORdst, set bits in dst with mask src. **Flag is not effected.**

Ex. For the initial conditions, R12=25A3H, R15= 8B94H and [25A5H]= 6CH, what will be the content of R12 after the execution of the following program?

```
bis R15,R12; R12←R12 OR R15
```



Operation:

1000 1011 1001 0100 (R15) OR

$\underline{0010\ 0101\ 1010\ 0011}$ (R12) =

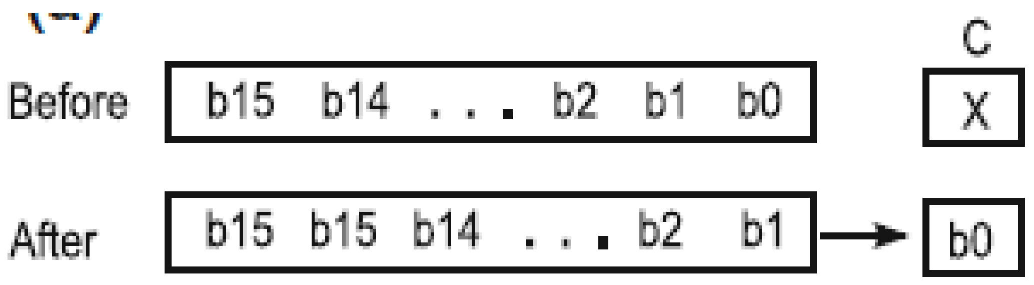
1010 1111 1011 0111

New Contents: R12 = AFB7

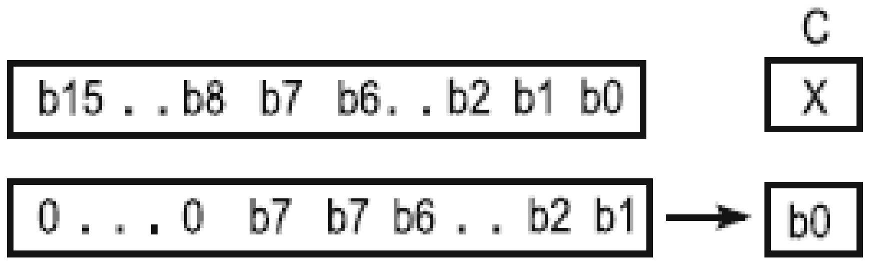
Flags: not affected.

Roll Right Arithmetically

`rra(.B or .W) dst;` Shift all bits to the right, C←LSB



rra.w; C=LSB



rra.b; C=LSB

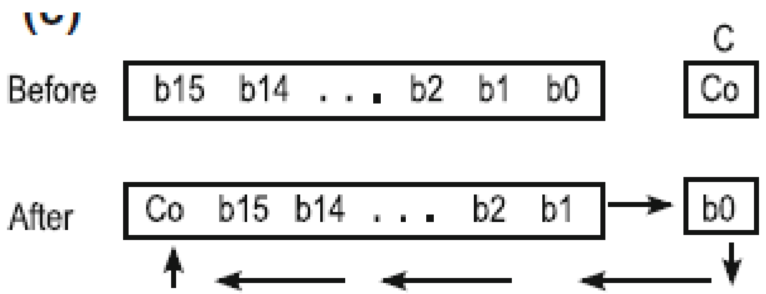
Ex. If the initial content of R5 is 8EF5H. What will be the content of R5 and the Carry value after the following codes individually?

rra.w R5; R5= C77AH, C=1

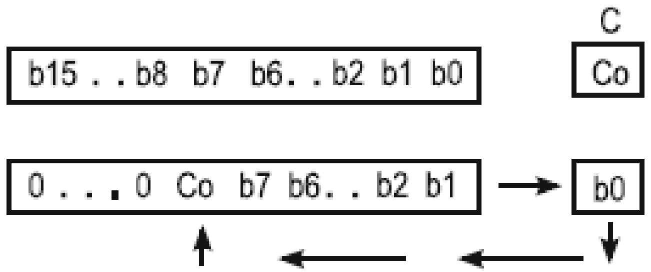
rra.b R5; R5= 00FAH, C=1

Rotate Right through Carry

`rrc(.B or .W) dst;` Shift all bits to the right, C←LSB



`rrc.w`; C=LSB



`rrc.b`; C=LSB

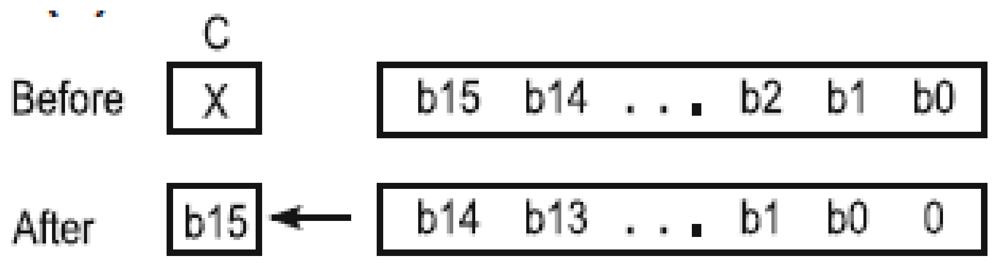
Ex. If the initial content of R5 is 8EF5H and C=0. What will the content of R5 and the new Carry value be after the following codes individually?

`rrc.w` R5; R5= 477AH, C=1

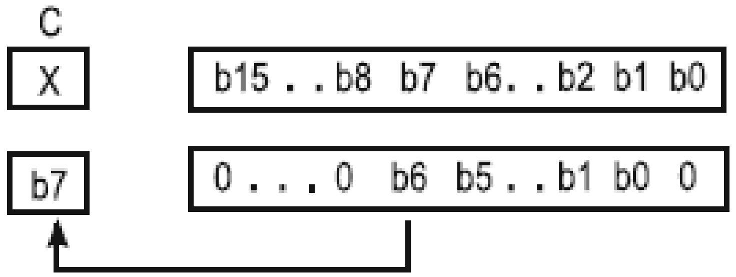
`rrc.b` R5; R5= 007AH, C=1

Roll Left Arithmetically

rla(.B or .W) dst; Shift all bits to the left



rla.w; C=b15



rla.b; C=b7

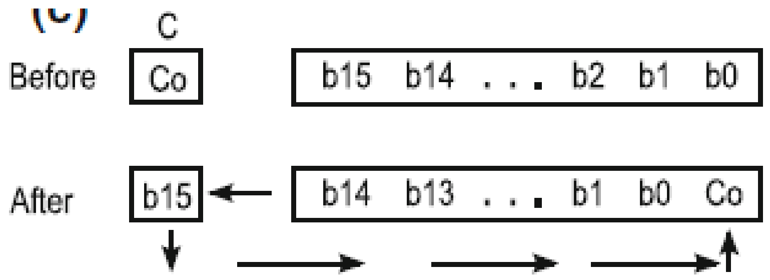
Ex. If the initial content of R5 is 8EF5H. What will the content of R5 and the new values of C, Z, N and V bits be after the following codes individually?

rla.w R5; R5= 1DEAH, C=1,Z=0,N=0,V=1

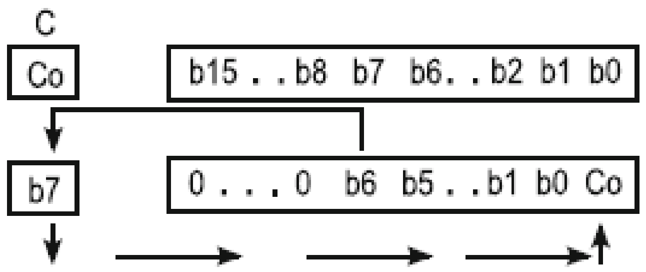
rla.b R5; R5= 00EAH, C=1,Z=0, N=1,V=0

Rotate Left through Carry

`rlc(.B or .W) dst;` Shift all bits to the left



`rlc.w`; C=b15



`rlc.b`; C=b7

Ex. If the initial content of R5 is 8EF5H and C=0. What will the content of R5 and the new values of C be after the following codes individually?

`rlc.w` R5; R5= 1DEAH, C=1

`rlc.b` R5; R5= 00EAH, C=1

Program Flow Instructions, Unconditional Jump

Unconditional jumps are realized with the jump instruction `jmp label`.

- When the program flow sees the `jmp label` instruction. Program flow will continue from the point the label indicates.
- Text for the label may be anything in English characters such as abc, xyz, etc.

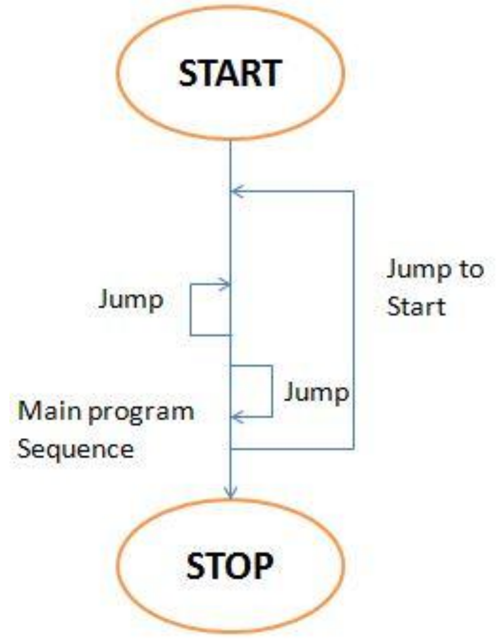


Figure. Unconditional jump

Conditional Jump

Conditional jumps are realized with different types of the jump instructions such as

`jz label, jnz label, jc label, jnc label, jl label,`

`jge label and jn label.`

- When the program flow sees the “jump” instruction and also the specific condition is satisfied, program flow will continue from the point the label indicates otherwise program will continue.
- Text for the label may be anything in English characters such as abc, xyz, etc.

Unconditional Jump

`JMP dst;` Program flow jumps to destination label without any condition.

Ex.

```
    mov.w #0x1234, r5; load R5 with 1234H
    mov.w #0x5678, r6; load R6 with 5678H
    jmp xyz; jump to label xyz
    mov.w #0xEEEE, r7; skip this line
    mov.w #0x2222, r8; skip this line
xyz mov.w #0x9999, r9 ; load R9 with 9999H
    mov.w #0xABCD, r10 ; load R10 with ABCDH
```

Program flow skips next 2 lines after `jmp` xyz

Unconditional Jump

Jumping procedure may also be nested…

```
Ex.  mov.w #0x1234, r5; load R5 with 1234H
     mov.w #0x5678, r6; load R6 with 5678H
     jmp xyz; jump to label xyz
     mov.w #0xEEEE, r7; skip this line
     mov.w #0x2222, r8; skip this line
 xyz mov.w #0x9999, r9 ; load R9 with 9999H
     mov.w #0xABCD, r10 ; load R10 with ABCDH
     jmp abc; jump to label abc
     mov.w #0xABCD, r3 ; skip this line
 abc mov.w #0x3333, r14; load R14 with 3333H
```

Unconditional Jump

Jumping direction can also be backward…

Ex.
```
     mov.w #0x1234, r5 ;load R5 with 1234H
     mov.w #0x5678, r6 ;load R6 with 5678H
     jmp abc ;jump to label abc
     mov.w #0xEEEE, r7 ;skip this line
     mov.w #0x2222, r8 ;skip this line
 xyz mov.w #0x9999, r9 ;load R9 with 9999H, skipped at first jump
     mov.w #0xABCD, r10 ;load R10 with ABCDH, skipped at first jump
     mov.w #0xABCD, r3 ;load R3 with ABCDH, skipped at first jump
 abc mov.w #0x3333, r14 ;load R14 with 3333H
     jmp xyz ;jump to label xyz
```

Conditional Jump

`JZ dst, JEQ dst;` Jumps to destination label if Z=1.

Ex.

```
      mov.w #0x1234, r5 ;load R5 with 1234H
      mov.w #0x1234, r6 ;load R6 with 1234H
      sub.w r6,r5 ;subtract R6 from R5, save to R5
      jz zero ;Z=1, jump to label zero
      mov.w #0xEEEE, r7 ;skip this line
      mov.w #0x2222, r8 ;skip this line
 zero mov.w  #0x1111, r9 ;load R9 with 1111H
      add.w #0x2222, r9 ;add 2222H to R9 and save in R9
```

Conditional Jump

`JNZ dst, JNE dst;` Jumps to destination label if Z=0.

```
Ex.    mov.w #0x1234, r5 ;load R5 with 1234H
       mov.w #0x4567, r6 ;load R6 with 4567H
       sub.w r5, r6 ;subtract R5 from R6, save to R6
       jnz nzero ;Z=0, jump to label nzero
       mov.w #0xEEEE, r7 ;skip this line
       mov.w #0x2222, r8 ;skip this line
 nzero mov.w #0x1111, r9 ;load R9 with 1111H
       xor.w #0x2222, r9 ;XOR 2222H with R9 and save in R9
```

Conditional Jump

`JN dst;` Jumps to destination label if N=1.

Ex.

```
         mov.w #0x1234, r5 ;load R5 with 1234H
         mov.w #0x4567, r6 ;load R6 with 4567H
         sub.w r6,r5 ;subtract R6 from R5, save to R5
         jn negative ;N=1, jump to label negative
         mov.w #0xEEEE, r7 ;skip this line
         mov.w #0x2222, r8 ;skip this line
negative mov.w  #0x1111, r9 ;load R9 with 1111H
         and.w #0x2222, r9 ;AND 2222H with R9 and save in R9
```

Conditional Jump

`JC dst;` Jumps to destination label if C=1.

Ex.

```
        mov.w #0xA234, r5 ;load R5 with A234H
        mov.w #0xB567, r6 ;load R6 with B567H
        add.w r6,r5 ;add R6 to R5, save to R5
        jc carry ;C=1, jump to label carry
        mov.w #0xEEEE, r7 ;skip this line
        mov.w #0x2222, r8 ;skip this line
carry   mov.w  #0x1111, r9 ;load R9 with 1111H
        and.w #0x2222, r9 ;AND 2222H with R9 and save in R9
```

Conditional Jump

`JNC dst;` Jumps to destination label if C=0.

Ex.

```
        mov.w #0x1234, r5 ;load R5 with 1234H
        mov.w #0x2567, r6 ;load R6 with 2567H
        add.w r6,r5 ;add R6 to R5, save to R5
        jnc ncarry ;C=0, jump to label ncarry
        mov.w #0xEEEE, r7 ;skip this line
        mov.w #0x2222, r8 ;skip this line
ncarry  mov.w  #0x1111, r9 ;load R9 with 1111H
        and.w #0x2222, r9 ;AND 2222H with R9 and save in R9
```

Conditional Jump

What if the jump condition is not satisfied…

Ex:

```
         mov.w #0x1234, r5 ;load R5 with 1234H
         mov.w #0x4567, r6 ;load R6 with 4567H
         sub.w r5,r6 ;subtract R5 from R6, save to R6
         jn negative ;N=0, DO NOT jump to label negative, just continue!
         mov.w #0xEEEE, r7 ;load R7 with EEEEH
         mov.w #0x2222, r8 ;load R8 with 2222H
negative mov.w  #0x1111, r9 ;load R9 with 1111H
         or.b #0x33, r9 ;OR 33H with R9 and save in R9
```

Conditional Jump

`JL dst;` Jumps to destination label if N and V bits are different.

```
 Ex.  mov.w #0xABCD, r5 ;load R5 with ABCDH
      mov.w #0x9876, r6 ;load R6 with 9876H
      add.w r6,r5 ;R5=4443H, V=1, N=0
      jl bjk ;jump to label bjk
      mov.w #0xEEEE, r7 ;skip this line
      mov.w #0x2222, r8 ;skip this line
bjk   mov.w  #0xAAAA, r9 ;load R9 with AAAAH
      and.b #0x44, r9 ;AND 44H with R9 and save in R9
```

Conditional Jump

`JGE dst;` Jumps to destination label if N and V bits are same.

Ex.
```
    mov.w #0x2345, r5 ;load R5 with ABCDH
    mov.w #0x6789, r6 ;load R6 with 6789H
    add.w r6,r5 ;V=1, N=1
    jge bjk ;jump to label bjk
    mov.w #0xEEEE, r7 ;skip this line
    mov.w #0x2222, r8 ;skip this line
bjk mov.w  #0xAAAA, r9 ;load R9 with AAAAH
    and.b #0x44, r9 ;AND 44H with the content of R9, save to R9
```

GPIO
General Purpose Input Output

Pinout

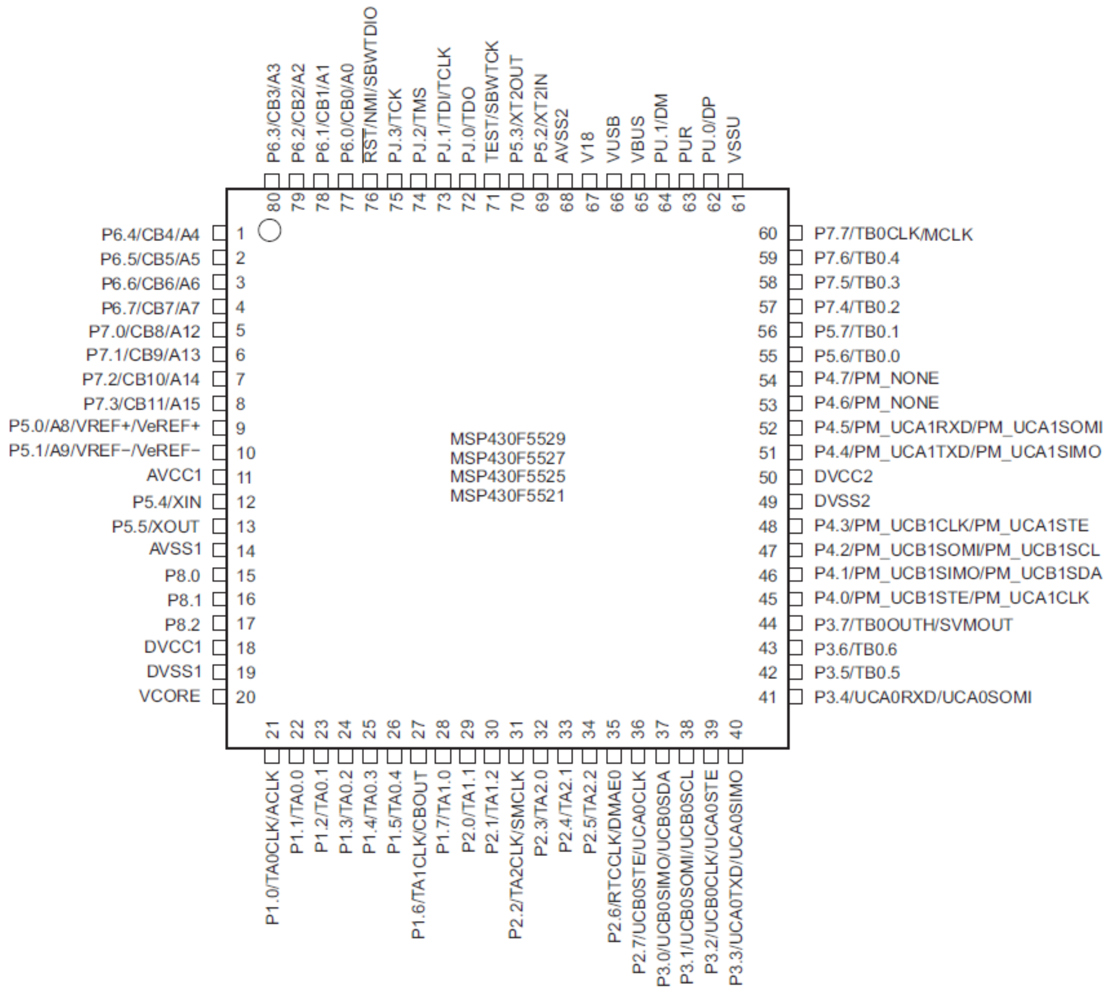


Figure. Pinout of MSP430F5529

* As can be seen from the figure, some pins of the microcontroller has multiple functions, these functions can be set through the software!

* Our MCU has totally 8 ports (P1,…P7, each port is 8-bit but P8 is 3-bit) that can be configured for different purposes.

MSP430 LaunchPad Evaluation Kit

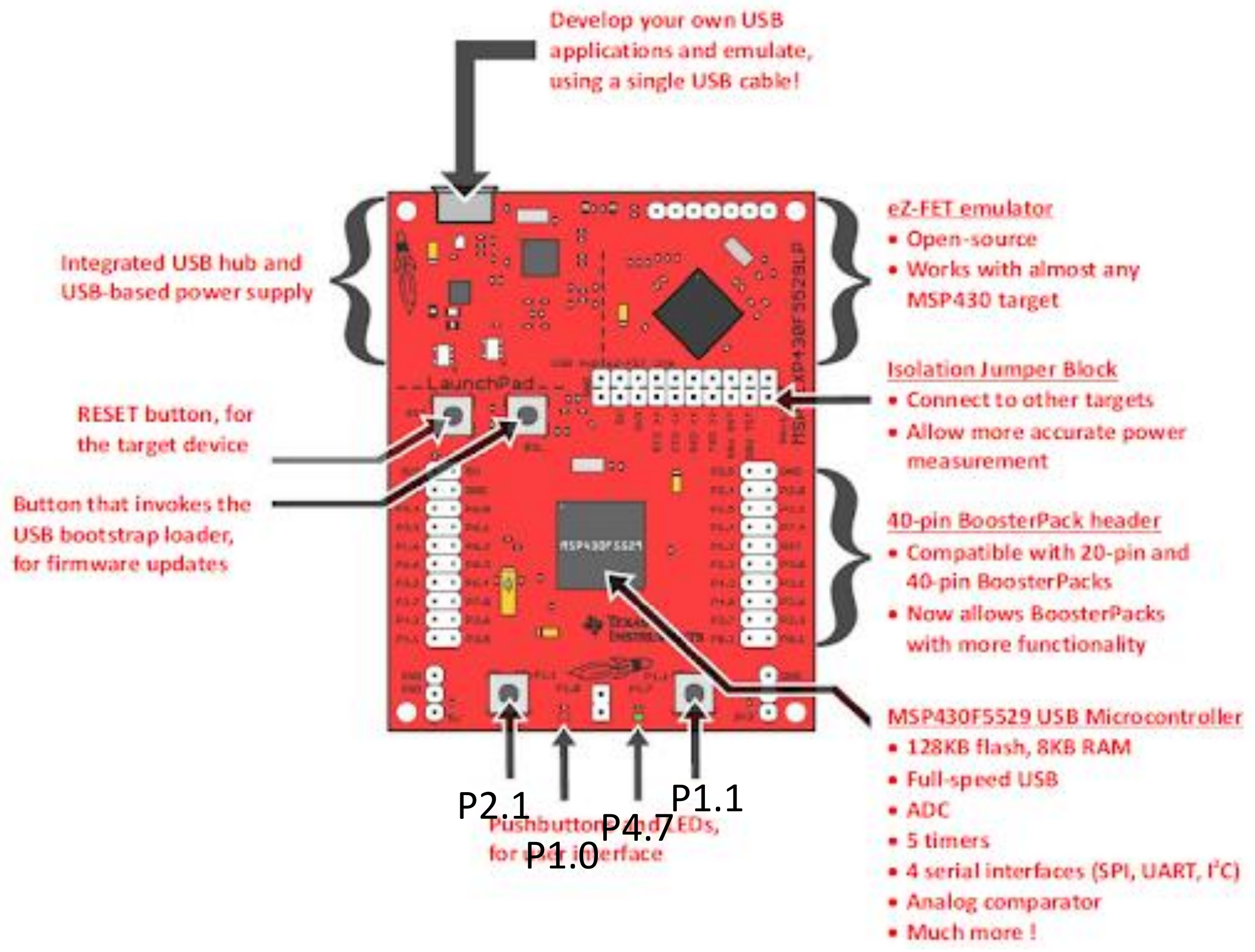
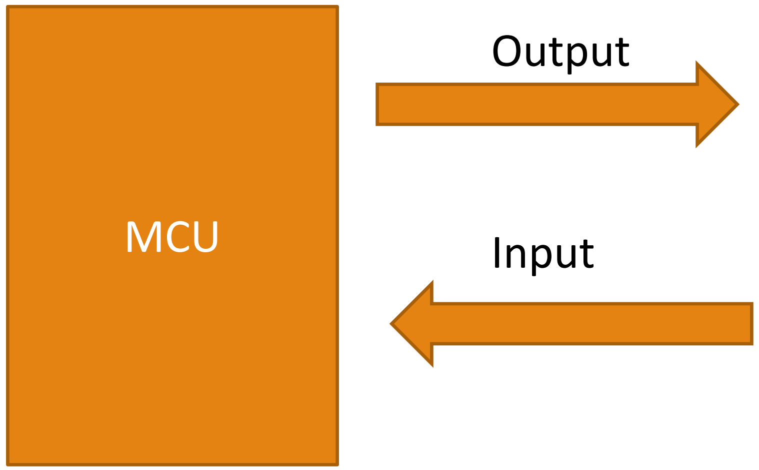

Develop your own USB applications and emulate, using a single USB cable!
Integrated USB hub and USB-based power supply
RESET button, for the target device
Button that invokes the USB bootstrap loader, for firmware updates
eZ-FET emulator
• Open-source
• Works with almost any MSP430 target
Isolation Jumper Block
• Connect to other targets
• Allow more accurate power measurement
40-pin BoosterPack header
• Compatible with 20-pin and 40-pin BoosterPacks
• Now allows BoosterPacks with more functionality
MSP430F5529 USB Microcontroller
• 128KB flash, 8KB RAM
• Full-speed USB
• ADC
• 5 timers
• 4 serial interfaces (SPI, UART, I²C)
• Analog comparator
• Much more !
Pushbuttons and LEDs, for user interface
P2.1
P1.0
P4.7
P1.1

Figure. MSP430F5529 LaunchPad

Input and Output



Input refers the data transfer **TO** the microcontroller(MCU)
Output refers the data transfer **FROM** the microcontroller(MCU)

* There are special function registers that allow the Ports to be configured as input and/or output
* Moreover, while some pins of a port can be configured as inputs others can be configured as outputs.

Port Registers

Direction Registers, PxDIR

It allows the user to configure the target port as an Input and/or Output. It is 8-bit register.

'x' is the port number (from 1 to 8)

Bit = 1: The port pin is set up as an output;

Bit = 0: The port pin is set up as an input.

Ex. Write the program that configures the Port 1's all bits as output

```
mov.b #0xff, r5 ;load R5's LSB with FFH
mov.b r5, P1DIR ;PIDIR=FFH
```

** Therefore, all pins of Port 1 are outputs

(P1.0, P1.1, P1.2, P1.3, P1.4, P1.5, P1.6, P1.7)

While the number before '.' shows port number, the one after '.' shows bit(pin) number.

Port Registers

Output Registers, PxOUT

It allows the user to send the desired data to the output port. Its width is 8-bit.

‘x’ is the port number (from 1 to 8)

Ex. Write the program that turns on the LED on P4.7

```
mov.b #0x80, r5 ;load R5's LSB with 80H (10000000)
mov.b r5, P4DIR ;P4DIR=80H, Only P4.7 is output, others are inputs
mov.b #0x80, P4OUT ;Turn ON P4.7, 10000000.
```

Port Registers

Output Registers, PxOUT

Ex. Write the program that toggles (ON and OFF) the P1.0 continuously.

```
     mov.b #0x01, r5 ;load R5's LSB with 01H
     mov.b r5, P1DIR ;P1DIR=01H, Only P1.0 is output, others are inputs
OFF  mov.b #0x00, P1OUT ;Turn OFF P1.0
     mov.b #0x01, P1OUT ;Turn ON P1.0
     jmp OFF ;jump to label OFF
```

Port Registers

Input Registers, PxIN

It allows the user to receive the desired data from the input port. Its width is 8-bit.

‘x’ is the port number (from 1 to 8)

It is read-only register, which means data inside the registers can be read but not written.

PxIN configuration:

Bit = 1: The input is high;

Bit = 0: The input is low;

Port Registers

Ex. Run the following program and discuss about the sense.

```
     mov.b #0xFF, P1DIR ;Entire Port1 is output
     mov.b #0xFF, P4DIR ;Entire Port4 is output
     mov.b #0x00, P4OUT ;Clear Port4, recommended to clear at start
     mov.b #0x00, P1OUT ;Clear Port1
     mov.b #0x00, P2DIR ;Entire Port2 is input
     mov.b P2IN, r9
     cmp  #0xFD, r9
     jz zero
     mov.b #0xFF, P4OUT
     jmp Done
zero mov.b #0xFF, P1OUT
Done
```

MSP430F5529 LaunchPad has logic 1 at its input pins (P2.1, P1.0, P4.7 and P1.1) as default.

Therefore, if no button pressed at P2, the data that is read in P2IN is **FFH.**

If the button at P2.1 is pressed (logic 0), the data that is read in P2IN is **FDH**

Program flow is controlled by the state of the button at P2.1 on the board

Port Registers

Port Function Select Registers (PxSEL1 and PxSEL0)

We use the Port Function Select (PxSEL) registers to tell the MCU which function to use, including whether to make the signal pin a digital input/output. The MSP430F5529 has more than two functions assigned to most of its pins, so it requires two bits to control the function selection.

PxSEL1	PxSEL0	Function
0	0	Digital I/O (Default)
0	1	Primary Function
1	0	Reserved
1	1	Secondary Function

Since PxSEL registers have 0 default value, we don’t have to configure them for GPIO applications.