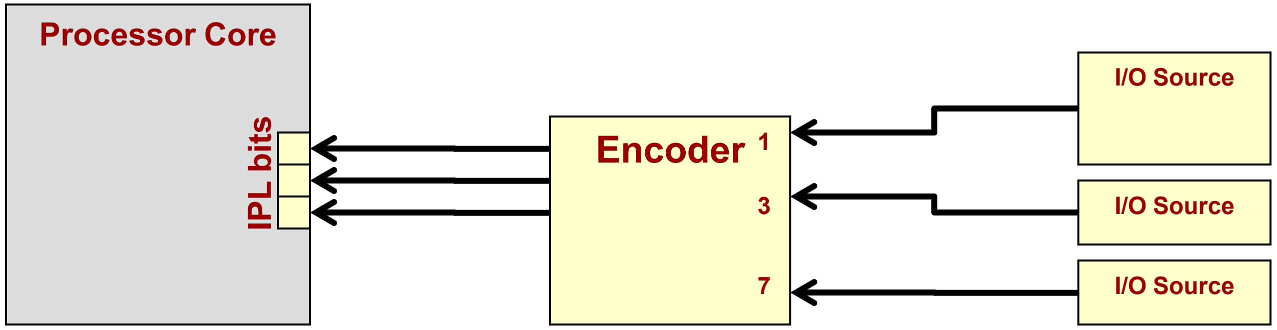
EE 357 Unit 10b

Interrupts

Timers

Coldfire / M68K Interrupts

- Coldfire interrupt architecture is based on original M68K
- 3-bit input (IPL[2:0]) indicating interrupt requests/priorities
  - 000 = No interrupt
  - 001-111 = Device 1-7 requesting interrupt



Masking Interrupts in the Processor Core

- May be times when we want the processor to execute important code and ignore (mask) interrupts
- I-bits bits in SR accomplish this
- Interrupt n will be ignored if n ≤ I-bits

SR:	0	0	1	0	0	101	000	1	0	1	0	1
	T		S	M		I-bits		X	N	Z	V	C

I-bits=5: Ignore interrupts [1-5]

SR:	0	0	1	0	0	000	000	1	0	1	0	1
	T		S	M		I-bits		X	N	Z	V	C

I-bits=0: Enable all interrupts

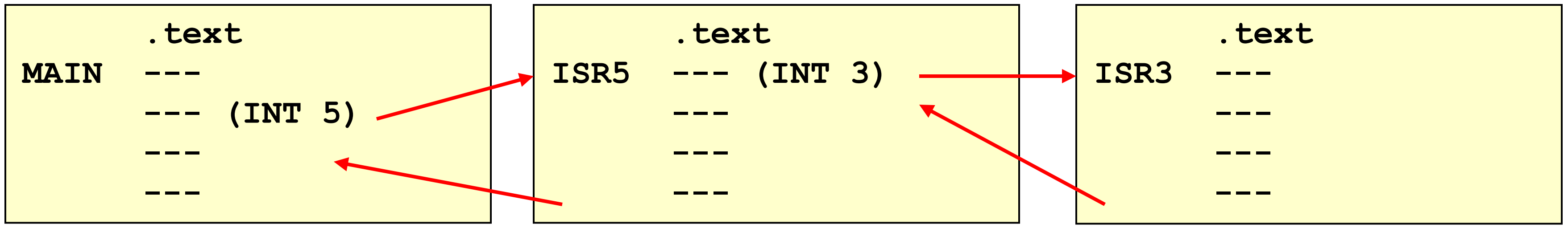
Non-Maskable Interrupt (NMI)

- I-bits = 7 would normally mean ignore all interrupts (n will always be ≤ I)
- Coldfire defines INT 7 as non-maskable
  - Cannot be ignored even if I = 7
- NMI is a safe guard to ensure some device can cause an interrupt no matter what.

Priority Inversion Problem

- Normally, higher priority interrupt should be handled before lower priority interrupts
  - Example: INT 5 should be processed before INT 3
- Priority inversion occurs when a lower priority interrupt occurs during handling of a higher priority interrupt

Priority Inversion Problem

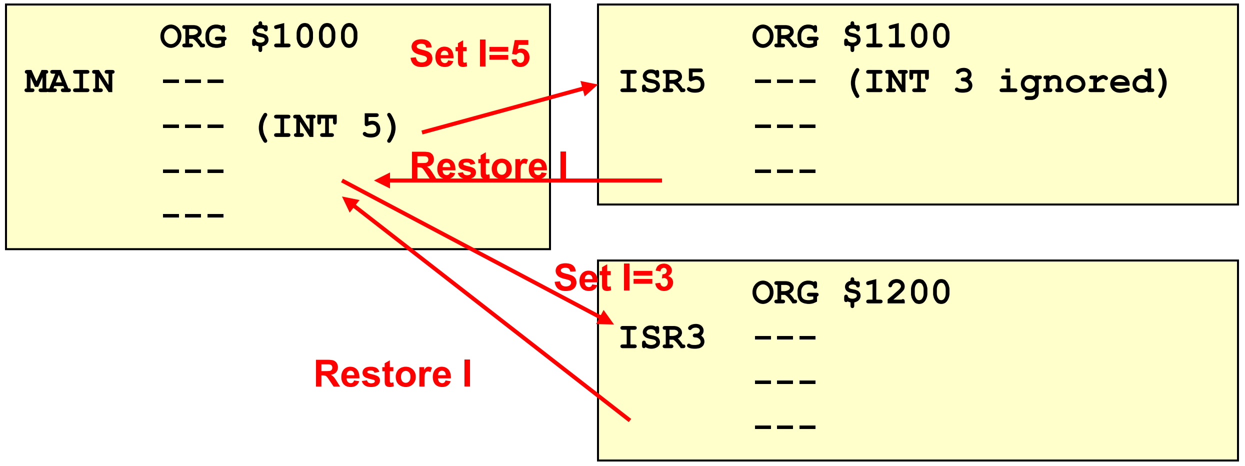


```
        .text
MAIN    ---
        --- (INT 5)
        ---
        ---
```

```
        .text
ISR5    --- (INT 3)
        ---
        ---
        ---
```

```
        .text
ISR3    ---
        ---
        ---
        ---
```

INT 3 interrupts ISR 5 and in effect INT 3 is handled before ISR 5 thus inverting the normal priority scheme.



```
        ORG  $1000
MAIN    ---
        --- (INT 5)
        ---
        ---
```

```
        ORG  $1100
ISR5    --- (INT 3 ignored)
        ---
        ---
```

```
        ORG  $1200
ISR3    ---
        ---
        ---
```

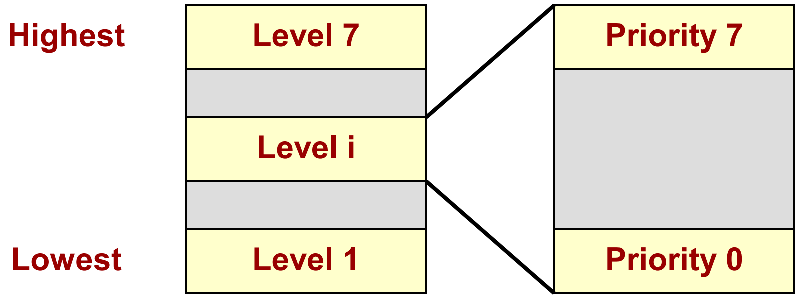
Solution: Raise I-bits = n on interrupt n (Raise it to 5 on INT 5 so that INT 3 will be ignored until ISR 5 completes)

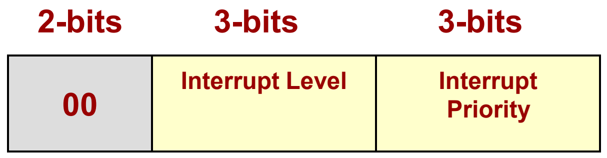
Interrupt Processing

- When an interrupt occurs, the CPU finishes the current instruction and then automatically goes through a 6 step process:
  1. Ignore interrupt if n ≤ I-bits
  2. Make a copy of the SR and Return Address/PC
  3. Raise I-bits = n
  4. Set S=1, T=0
  5. Push Return Address and Copy of SR onto stack
  6. Load PC with address from Exception Vector Table
- After handler finishes and calls RTE, original/copied SR will restore I-bit value

52259 Interrupt Priority Scheme

- 7 levels of priority for interrupts (higher priority interrupts are handled first) [Level 0 = No interrupt]
- MCF52259 has > 70 interrupt sources
  - How to map over 70 sources to 7 levels?
  - Break each level into 8 sub-priorities
  - Allows a total ordering of interrupt sources
  - Levels/Priorities are user-settable via control registers (i.e. timer0 can be set to level: 4, pri.: 1 or level: 6, pri.: 3)





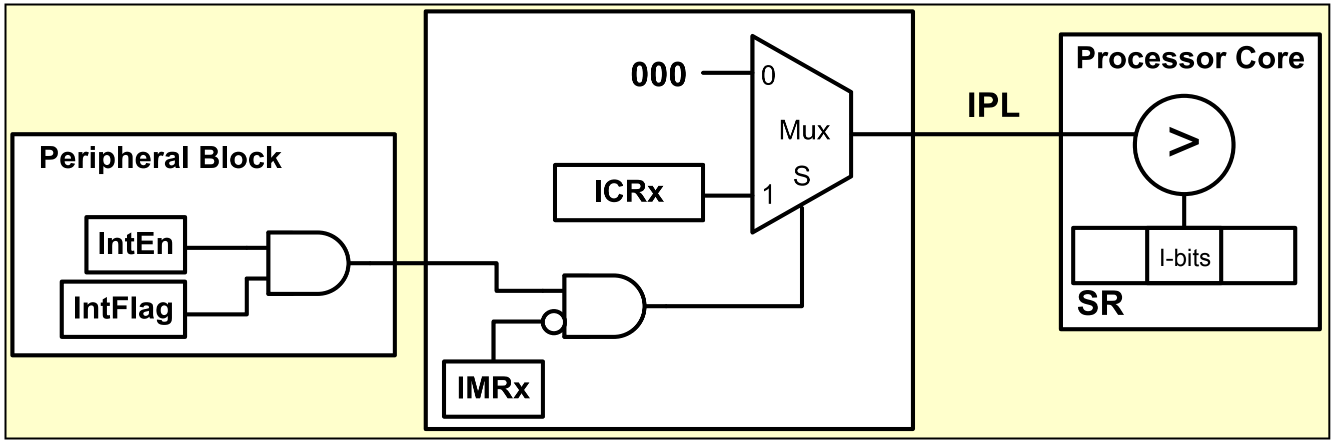
ICRxx Register
(Specifies the level and priority of interrupt source xx)

Masking Interrupts

- MCF52259 provides several mechanisms to enable or disable (mask) interrupts
- Processor Core
  - 3-bit value (Interrupt Priority Mask) in its SR (Status Register) allows only HIGHER priority interrupts (int n > I-bits of SR)
- Interrupt Controller Module
  - IMR Register's Individual Interrupt Mask bits (1=Disable / 0=Enable) – Clear this bit to enable that interrupt source
  - IMR Register's Master Disable (Mask) bit – Set this bit and ALL interrupts are disabled
  - ICR Register – Can set the level and priority for each interrupt source
- I/O Peripheral Block
  - Interrupt Enable bit – Must be set to '1' to allow that peripheral to attempt to generate interrupts

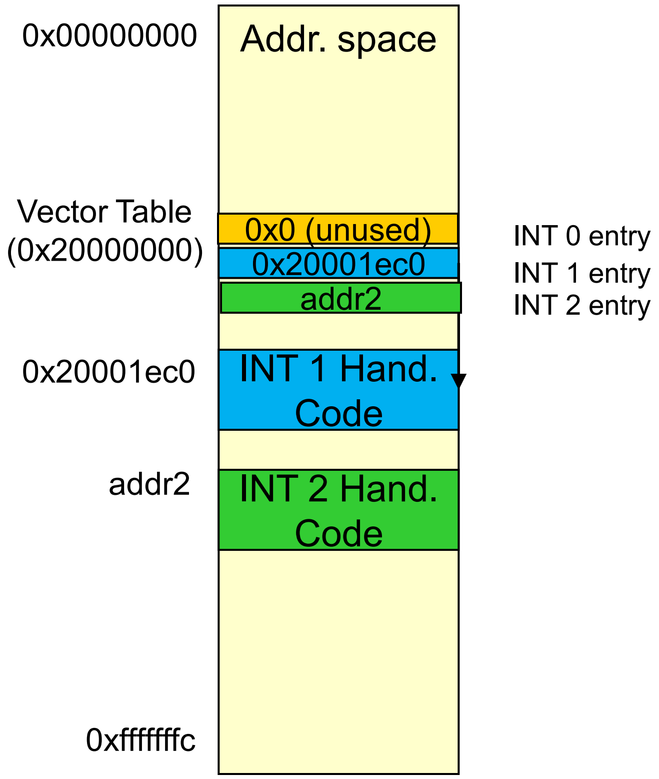
Interrupt Masking Control Bits

- Peripheral Block
  - Interrupt Flag (IF) is the interrupt source
- IMRxx = Interrupt Mask Reg.
  - (1=masks/ignores int., 0=allow int.)
- ICRxx = Interrupt Control Reg.
  - Contains the desired 3-bit level (IPL value)



Registering Interrupt Handlers

- Each interrupt source is assigned a number (so we know which mask bit and ICR to set)
  - See 52259 Reference Manual Table 12-16 (Sec. 16.3.8.1) for source numbers
- Vector table is an array with one entry per interrupt source
  - Each entry holds the starting address of a subroutine (handler) to be called when the interrupt occurs
- We will have to initialize the appropriate entry at startup



Interrupt Configuration Process

Initialization

1. Register handler routine in EVT
2. Enable local peripheral IE (int. en.)
3. Set the ICR to desired level
4. Clear the appropriate IMR mask bit
5. Set I-bits in SR (usually to 0)

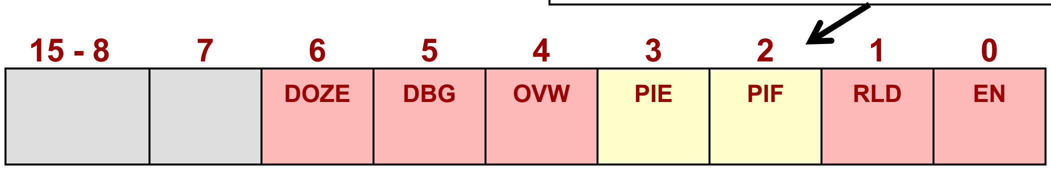
Operation

- Acknowledge interrupt (usually by writing a ‘1’ to the interrupt flag bit)

Acknowledging Interrupts

- As the first step in the interrupt handler you must clear the interrupt flag (source) to acknowledge that it is being handled and prevent it from generating more interrupt requests
  - Sometimes accomplished by writing a ‘0’ to a flag bit
  - Sometimes accomplished by writing a ‘1’ to a flag bit

When timer interrupt occurs, the PIF bit is set to ‘1’. We clear the PIF bit (set to ‘0’) by writing a ‘1’ to it



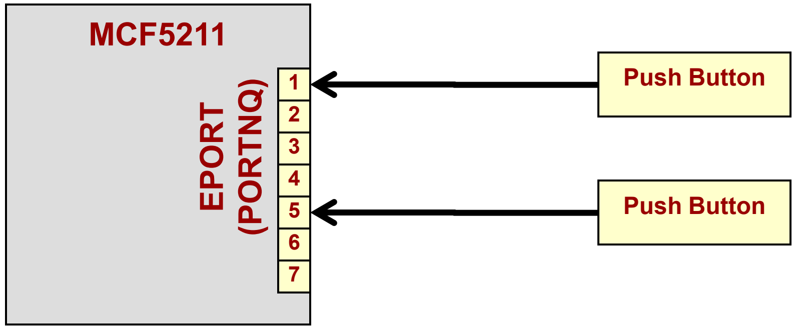
15 - 8	7	6	5	4	3	2	1	0
		DOZE	DBG	OVW	PIE	PIF	RLD	EN

Programmable Interrupt Timer Control/Status Register (PCSR)

```
void pit0_handler()
{
  MCF_PIT0_PCSR = MCF_PIT0_PCSR |
                  0x0004;
  ...
}
```

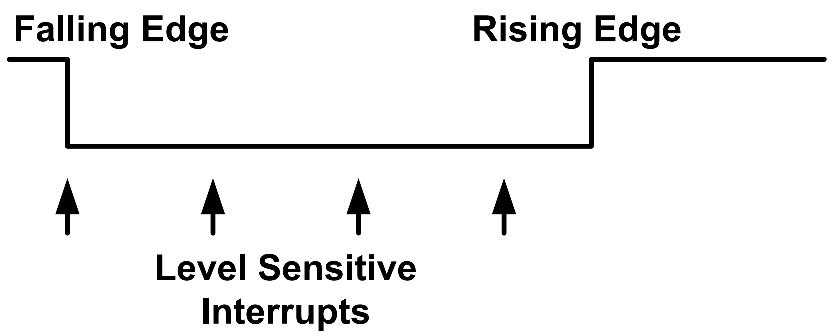
EPORT (External) Interrupts

- Edge Port (EPORT) allows user-defined signals to generate an interrupt
  - For our purposes, push buttons generating
- EPORT is really PORTNQ[7:1]
  - Rather than using it as GPIO we can use it to sense interrupt conditions
  - There are 7 pins (labeled from 7 to 1 rather than 6 to 0)



Edge Port Interrupts

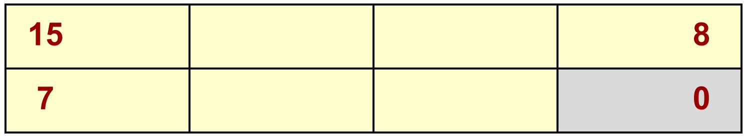
- Can be set to generate an interrupt on
  - A rising edge (transition from 0 to 1)
  - A falling edge (transition from 1 to 0)
  - Both rising and falling edges
  - Level sensitive (generate an interrupt whenever signal is low ‘0’)



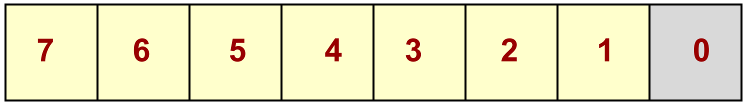
- For “edge” settings
  - Triggers only on an edge (1 interrupt no matter how long the signals is active)
  - An edge will be recorded for handling later if the processor is busy with a higher priority INT
- For “level” sensitive…
  - Generates an interrupt whenever signal is active
  - Will not be recorded (if processor is busy with higher interrupt, a level-sensitive interrupt will be lost)

Edge Port Registers

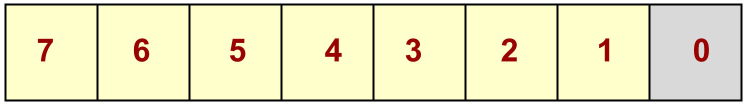
- EPPAR – Pin Assignment
  - Sets edge or level-sensitivity
- EPIER – Interrupt Enables
  - Must be set to allow the interrupt sources to generate interrupts to the processor core
- EPDDR – Data Direction
  - Must set to input (‘0’) to use as interrupt inputs
- EPFR – Flag Register
  - Indicates an interrupt has been generated. Must be cleared in the interrupt handler by writing a ‘1’ to the appropriate bit



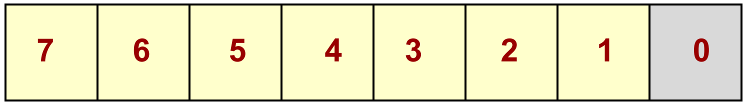
EPPAR (0=Level Sens. / 1=Falling-Edge / 2=Rising-Edge / 3=Both edges)



EPIER (1=Enable / 0=Disable)



EPDDR (1=output / 0=input)



EPFR (1=output / 0=input)

Programmable Interrupt Timer

- MCF52259 provides (2) programmable interrupt timers (PIT0 and PIT1)
- Operation
  - Set a 16-bit counter with a value
  - It will count down and when it reaches 0 it will generate an interrupt
  - It can automatically reset to the start value and count down again for regular interrupts or simply stop and wait to be used again

Timer Control

- PMR (PIT Modulus Reg.)
  - Counter initialization value
- PCNTR (PIT Counter)
  - A Read to this returns the 16-bit counter value
- PCSR (PIT Control/Status Reg.)
  - EN – enables counter to count down
  - Prescaler: $(F_{Sysclk} / 2) / 2^{Prescale}$
  - **F_{Sysclk} = 8 MHz in the MCF52259**
  - OVW – overwrite counter value when you write to PMR
  - RLD – Reload counter with PMR when it reaches 0
  - PIE – Interrupt Enable
  - PIF – Flag generates IRQ; Write of 1, clears it.

15	PMR (High)	8
7	PMR (Low)	0

PMR (16-bit value)

15	PMR (High)	8
7	PMR (Low)	0

PCNTR (16-bit value)

				PRE			
	DOZE	DBG	OVW	PIE	PIF	RLD	EN

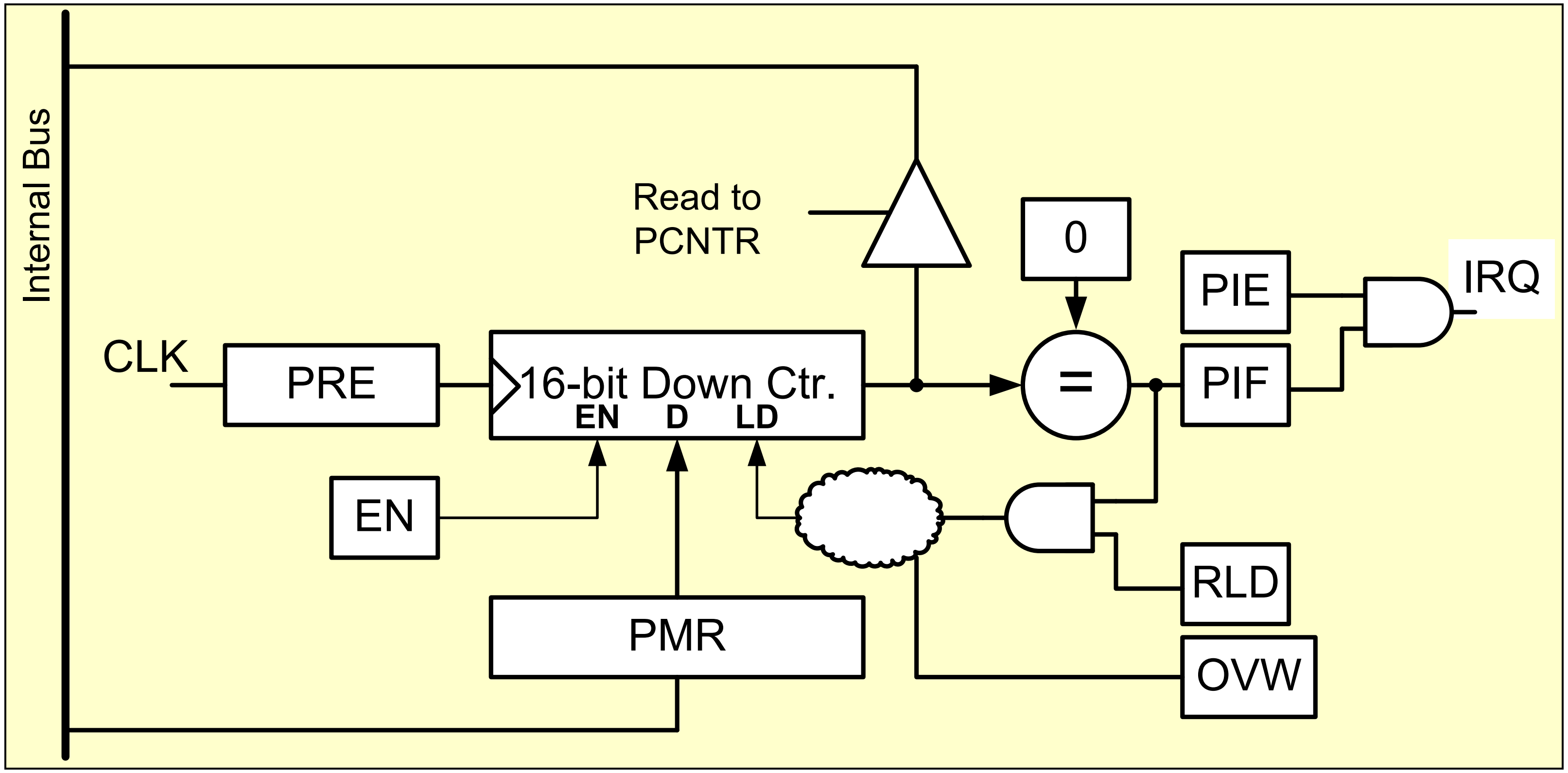
PCSR (16-bit value)

Timer Prescalar

- System clock runs at 8 MHz and we only have 16-bit down counter (i.e. max count of 65535)
- At 8 MHz, an interrupt would be generated in 65535 / (8E6 / 2) = 65535 / 4E6 = 16.384 ms
- Prescalar allows us to divide down the clock rate going into the counter
  - New counter clock rate 4 MHz / 2^{PRE}
- To generate an interrupt for a specific interval, use the above formula selecting a prescalar [0-15], solve for PMR and ensure it is less than 65535. If it is greater than that, pick a larger prescalar and iterate.

$$\text{Interval} = \text{PMR} * 2^{PRE} / 4*10^6$$

Timer Operation


Internal Bus
Read to
PCNTR
0
PIE
IRQ
CLK
PRE
16-bit Down Ctr.
EN
D
LD
=
PIF
EN
RLD
PMR
OVW