Chapter 6
Interrupt Programming

Why Interrupt?

- A computer is much more than the CPU
  - Keyboard, mouse, screen, disk drives, scanner, printer, sound card, camera, etc.
- These devices occasionally need CPU service
  - But we can’t predict when
- Solution 1: CPU periodically checks each device to see if it needs service – Polling

Disadvantage of Polling

- “Polling is like picking up your phone every few seconds to see if you have a call.”
- You may get a few phone calls a day → 99% of your effort will be wasted checking the phone.
- Takes CPU time even when no request pending

Example of Polling in Chapter 5

Create a square wave of 50% duty cycle on RB5. *Here the CPU spends most of the time checking the interrupt flag.*

```
        bcf     TRISB, 5        ; Set PB5 as output
        bsf     PORTB, 5
        movlw   B '00001000'    ; 16-bit, int clk, no prescaler
        movwf   T0CON
Here:   movlw   0xFF            ; TMR0H = 0xYY
        movwf   TMR0H
        movlw   0xF2            ; TMR0L = 0xXX
        movwf   TMR0L
        bcf     INTCON, TMR0IF  ; Clear timer interrupt flag
        bsf     T0CON, TMR0ON   ; start Timer 0
Again:  btfss   INTCON, TMR0IF
        bra     Again
        bcf     T0CON, TMR0ON
        btg     PORTB, 5
        bra     Here            ; load TMR0H and TMR0L again
```

Interrupt

- Whenever a device needs the CPU's service, the device notifies it by sending an interrupt signal.
- CPU then stops and serves the device.
- "Polling is like picking up your phone every few seconds to see if you have a call. Interrupts are like waiting for the phone to ring."

MCU Response to Interrupts

- If the interrupt request is present, the microcontroller:
  - Completes the execution of the current instruction
  - Saves the address of the program counter on the stack
- The microcontroller is redirected to the memory location where the interrupt request can be met.
- The set of instructions written to meet the request is called an interrupt service routine (ISR). *The interrupt flag has to be reset in ISR*.
- Once the request is accomplished, the MCU should find its way back to the next instruction, where it was interrupted.

PIC18 Interrupt

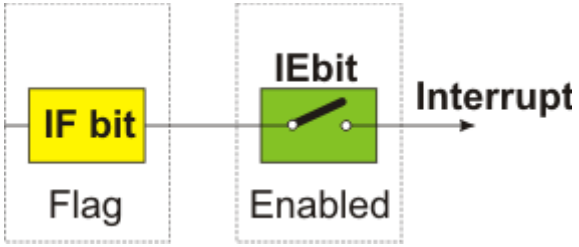
- PIC18 Microcontroller family
  - Has multiple sources that send interrupt requests, which can be classified into:
    - Internal Events
    - External Events
  - Has a priority scheme that divided interrupt requests into two groups:
    - High Priority
    - Low Priority

PIC18 Interrupt Sources

- Internal peripheral sources
  - Examples: Timers, A/D Converter, UART
- External sources
  - Three pins of PORTB: RB0, RB1 and RB2
    - These pins can be used to connect external interrupting sources such as keyboards or switches
    - Can select whether a rising or falling edge triggers an interrupt.
  - Change in logic levels of any of the pins RB4 to RB7 of PORTB can be recognized as interrupts

PIC18 Interrupt Sources

- Each interrupt source has:
  - A flag bit to indicate whether the interrupt has occurred (e.g., TMR0IF)
  - An enable bit to enable/disable the interrupt source (e.g., TMR0IE)
  - A priority bit to select priority. This bit has effect only when the priority scheme is enabled. (e.g., TMR0IP)



SFRs involved in interrupts

- RCON register sets up global priority.
- INTCON registers deal primarily with external interrupt sources.
  - INT0, INT1, INT2: Pins connecting external interrupting sources
  - Change in logic levels of any of the pins RB4-RB7
- PIR: Flag bits of internal peripheral interrupt sources.
- PIE: Enable bits of internal peripheral interrupt sources.
- IPR: Priority bits of internal peripheral interrupt sources.

Interrupt Priorities

- The interrupt priority feature is enabled by Bit 7 (IPEN) in RCON register.
- 2 Cases:
  - IPEN = 0: Interrupt priority is disabled.
  - IPEN = 1: Interrupt priority is enabled.
    - Interrupts are associated with high and low priorities.
    - A high-priority interrupt can interrupt a low-priority interrupt in progress.

Interrupt Priorities Not Set (IPEN = 0)

- All interrupts are treated as high-priority interrupts.
- Core group are under the control of a *two-level* enabling scheme
  - Global Interrupt Enable (GIE) bit
  - The enable bit of the interrupt source
- Peripheral interrupts are under the control of *a three-level* enabling scheme
  - Global Interrupt Enable (GIE) bit
  - Peripheral Interrupt Enable (PEIE) bit
  - The enable bit of the interrupt source
- Interrupt sources that are in the core group include:
  - INT0, INT1, INT2 external interrupts
  - Timer 0 overflow interrupt
  - PORT B Level-Change interrupts
- Other interrupt sources are considered peripheral sources

Interrupt Priorities Set (IPEN = 1)

- High-priority interrupts are under the control of a *two-level* enabling scheme
  - Global High-Priority Interrupt Enable (GIEH) bit
  - The enable bit of the interrupt source
- Low-priority interrupts are under the control of *a three-level* enabling scheme
  - Global High-Priority Interrupt Enable (GIEH) bit
  - Global Low-Priority Interrupt Enable (GIEL) bit
  - The enable bit of the interrupt source
- GIEH has the same location as the GIE bit and GIEL has the same location as the PEIE bit. They are interpreted differently when IPEN = 1.

Interrupt Service Routine (ISR)

- When an interrupt occurs, the microcontroller:
  - Completes the instruction being executed
  - Disables global interrupt enable (GIE/GIEH/GIEL)
  - Places the address from the program counter on the stack
  - WREG, STATUS and BSR are saved to corresponding built-in shadow registers
  - call ISR
- The starting address of the ISR (as called as *interrupt vector*) is predefined:
  - High Priority Interrupt – 0x000008
  - Low Priority Interrupt – 0x000018

Interrupt Service Routine (ISR)

- Attends to the request of an interrupting source
  - Clears the interrupt flag
  - Should save register contents that may be affected by the code in the ISR
  - Must be terminated with the instruction `retfie`
  - WREG, STATUS and BSR are optionally restored from corresponding built-in shadow register by `retfie 1`.

retfie

- retfie
  - Return from interrupt
  - [PC] = [TOS]
  - Enable the global interrupt bit (GIE/GIEH/GIEL).
  - WREG, STATUS and BSR are optionally restored from corresponding built-in shadow register by `retfie 1`.

RETFIE	**Return from Interrupt**
Syntax:	RETFIE {s}
Operands:	s ∈ [0,1]
Operation:	(TOS) → PC, 1 → GIE/GIEH or PEIE/GIEL; if s = 1, (WS) → W, (STATUSS) → STATUS, (BSRS) → BSR, PCLATU, PCLATH are unchanged
Status Affected:	GIE/GIEH, PEIE/GIEL.
Encoding:	0000 0000 0001 000s
Description:	Return from interrupt. Stack is popped and Top-of-Stack (TOS) is loaded into the PC. Interrupts are enabled by setting either the high or low-priority global interrupt enable bit. If 's' = 1, the contents of the shadow registers, WS, STATUSS and BSRS, are loaded into their corresponding registers, W, STATUS and BSR. If 's' = 0, no update of these registers occurs (default).

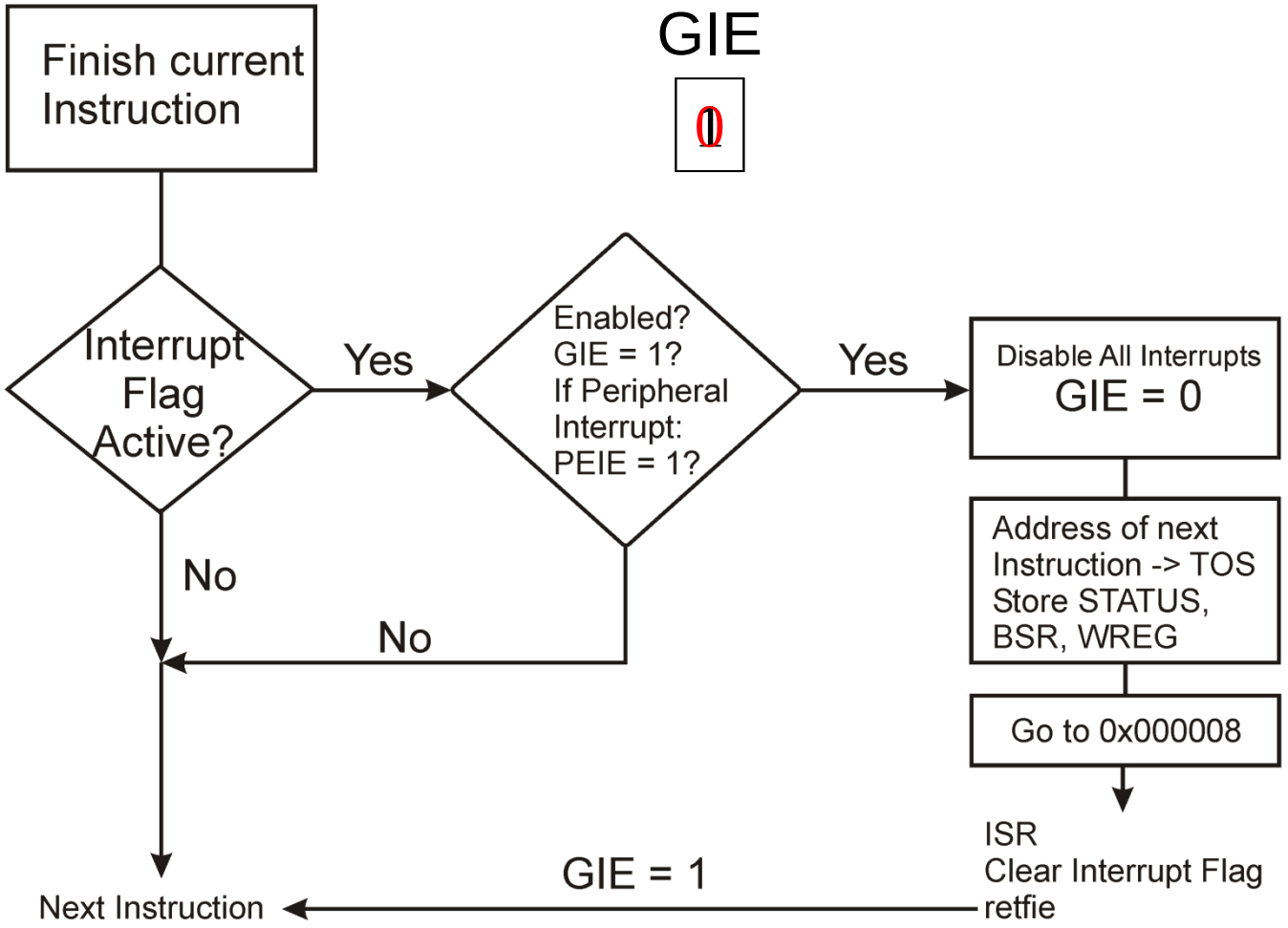
`retfie` **vs.** `return`

- Both instructions pops the TOS into PC.
- Recall: The global interrupt enable flag (GIE/GIEH/GIEL) has been disabled when an interrupt is evoked to prevent multiple interrupts.
- `retfie` enables the global interrupt enable, while `return` does not.

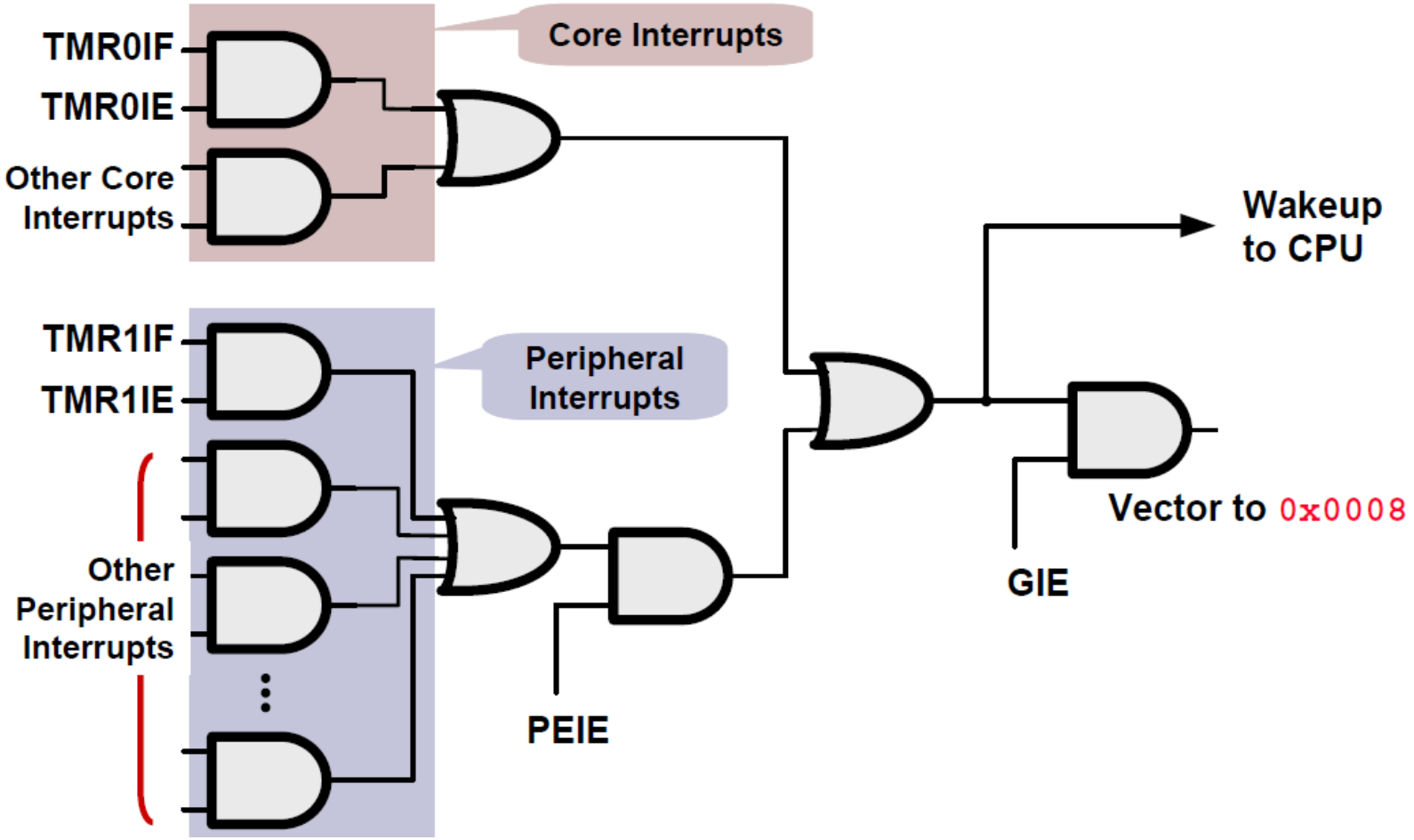
Shadow Registers

- An interrupt can occur at any point in the execution of a program.
  - The program may be in the middle of an arithmetic operation.
- The ISR will, at a minimum, change W and STATUS, and probably the BSR
  - These must be saved on ISR entry, and restored on ISR exit
- These registers are stored in corresponding built-in shadow registers before entry to ISR.
- Contents from these shadow registers are optionally restored by `retfie 1`.

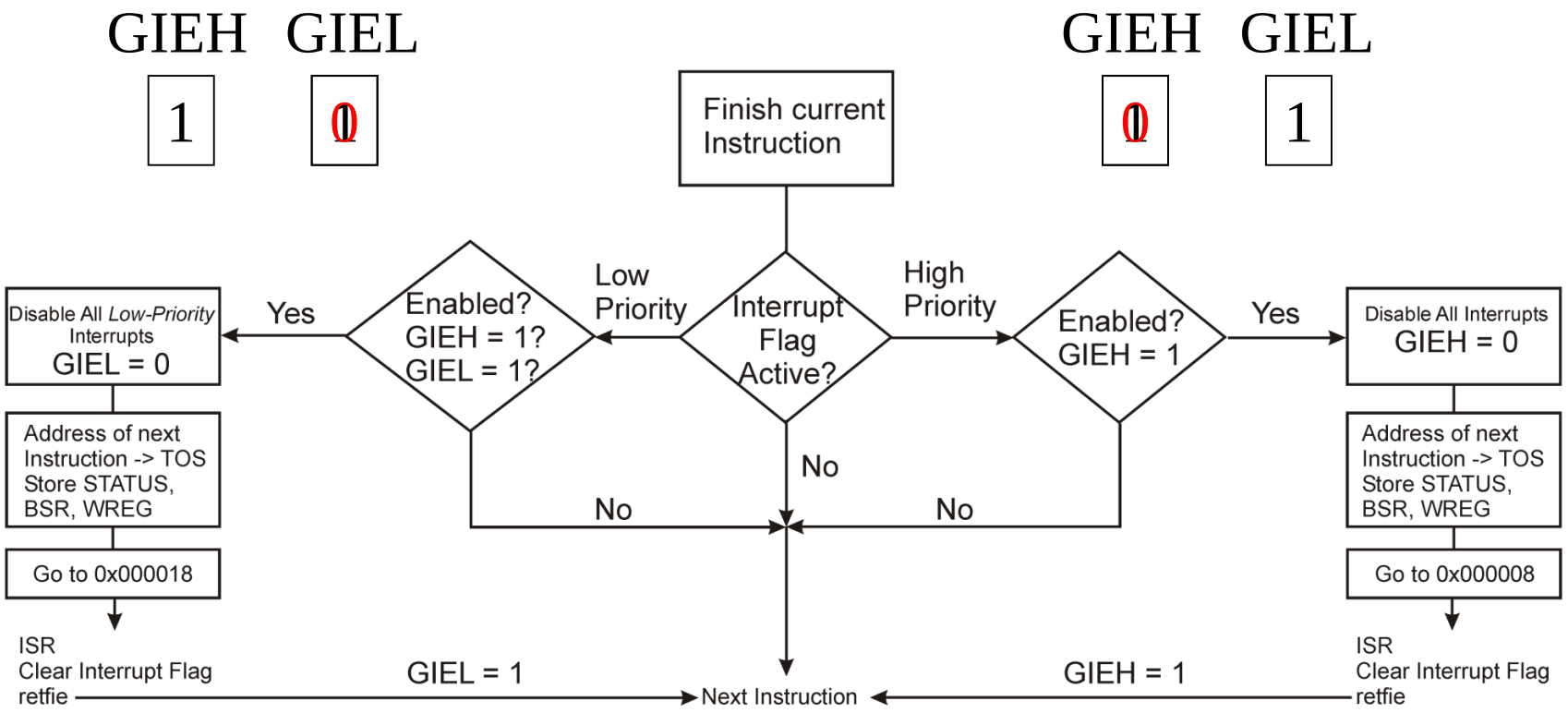
Interrupt Priorities Not Set (IPEN = 0)



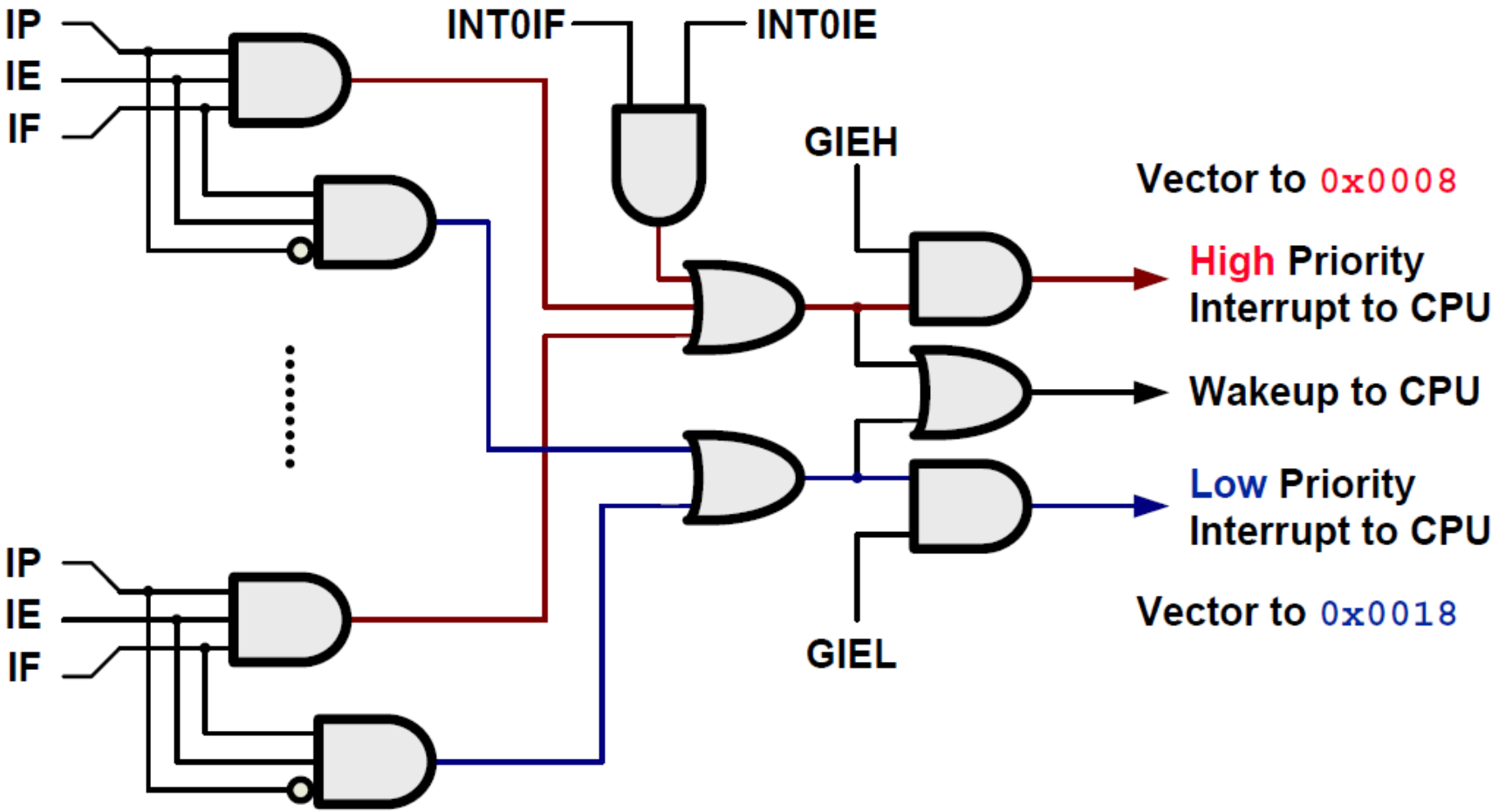
IPEN = 0 Interrupt Circuit



Interrupt Priorities Set (IPEN = 1)



IPEN = 1 Interrupt Circuit



Example with Interrupt Priorities Disabled

- We have used polling to create a square wave of 50% duty cycle on RB5 bit.
- In this example, we implement the same thing using interrupt with interrupt priorities disabled (IPEN = 0).

Example with Interrupt Priorities Disabled

```
                ORG 0x000000
                goto Main
                ORG 0x000008
                goto T0_ISR
                ORG 0x000100
Main:           bcf TRISB, 5
                 bsf PORTB, 5
                movlw 0x08; Timer0, 16-bit, no prescale, internal ck
                movwf T0CON
                movlw 0xFF
                movwf TMR0H
                movlw 0xF2
                movwf TMR0L; load FFF2 to TMR0
                bcf       RCON, IPEN; Disable Interrupt Priorities
                bcf       INTCON, TMR0IF; Clear interrupt flag bit
                bsf       INTCON, TMR0IE; enable Timer0 interrupt
                bsf       INTCON, GIE; enable global interrupt
                bsf       T0CON, TMR0ON; start Timer0
                bra       $
```

Example with Interrupt Priorities Disabled

```
                ORG 0X000200
T0_ISR:         btfss INTCON, TMR0IF
                bra   EXIT
                bcf   INTCON, TMR0IF
                btg   PORTB, 5; toggle PortB.5 to create sq. wave
                movlw 0xFF
                movwf TMR0H
                movlw 0xF2
                movwf TMR0L; Reinitialize TMR0 to FFF2
EXIT:           retfie 1
                END
```

Example: Observations

- We avoided using the memory space allocated to the interrupt vector table.
  - When awakened at address 0x000000, PIC18 executed the `goto` instruction, which redirects the controller away from the interrupt vector table.
- In Main, we enabled:
  - Timer 0 interrupt: `bsf INTCON, TMR0IE`
  - Global interrupt: `bsf INTCON, GIE`
- In Main, we initialized the Timer 0 registers and then enter an infinite loop. We waited for the TMR0IF flag to be raised when Timer0 rolls over.

Example: Observations

- When TMR0IF flag is raised, the microcontroller gets out of the infinite loop in Main and goes to 0x000008 to execute the ISR.
- The GIE bit is disabled, **locking out further interpret requests.**
- The ISR for Timer 0 is located starting at memory location 0x000200 because it is too large to fit into address space 0x000008-000017, the address allocated to high-priority interrupts.
- Upon execution of `retfie`, the program gets back to where the interrupt occurred and GIE is re-enabled so that new interrupts can be served.
- `retfie 1` restores the contents of WREG, STATUS and BSR from corresponding built-in registers.

Example of a low-priority interrupt

- In this example, we implement the same thing as the last example with
  - interrupt priorities enabled (IPEN = 1).
  - interrupt set to low priority

Example of a low-priority interrupt

```
                cblock 0x7D
                w_temp, status_temp, bsr_temp
                endc

                ORG 0x000000
                goto Main
                ORG 0x000008
                retfie 1; high-priority interrupt
                ORG 0x000018
                goto T0_ISR; low-priority interrupt
                ORG 0x000100
Main:           bcf TRISB, 5
                bsf PORTB, 5
                movlw 0x08
                movwf T0CON
                movlw 0xFF
                movwf TMR0H
                movlw 0xF2
                movwf TMR0L
                bsf   RCON, IPEN;   enable priority interrupt
                bcf   INTCON, TMR0IF; clear TMR0 interrupt flag
                bsf   INTCON, TMR0IE; enable TMR0 overflow interrupt
                bcf   INTCON2, TMR0IP; set TMR0 interrupt to low
                              ; priority
                bsf   INTCON, GIEH; enable high priority interrupt
                bsf   INTCON, GIEL; enable low priority interrupt
                bsf   T0CON, TMR0ON
                bra   $
```

Example of a low-priority interrupt

```
                ORG 0X000200
T0_ISR:         btfss INTCON, TMR0IF
                bra  EXIT
                movwf w_temp ; preserve context
                movff STATUS, status_temp
                movff BSR, bsr_temp
                bcf  INTCON, TMR0IF
                btg  PORTB, 5
                movlw 0xFF
                movwf TMR0H
                movlw 0xF2
                movwf TMR0L
                movff bsr_temp, BSR ; restore context
                movf w_temp, w
                movff status_temp, STATUS
EXIT:           retfie
                END
```

Example: Observations

- To activate a low-priority interrupt, we need to enable three bits:
  - Global High-Priority Interrupt Enable (GIEH) bit `(bsf INTCON, GIEH)`
  - Global Low-Priority Interrupt Enable (GIEL) bit `(bsf INTCON, GIEL)`
  - The enable bit of the interrupt source `(bsf INTCON, TMR1IE)`
- Low priority interrupt is served at memory location 0x000018.

Example: Observations

- W, BSR and STATUS registered must be explicitly saved and restored in a low-priority ISR.
- Even though these registers are saved in corresponding built-in shadow registers upon entry to the low-priority ISR, contents in the built-in shadow registers are overwritten if the low priority ISR is interrupted by a high-priority interrupt.
- Cannot restore from built-in registers → Use `retfie` (instead of `retfie 1`)

Interrupt program template

```
        CBLOCK 0x7D
        w_temp, status_temp, bsr_temp
        endc
        org 0x008
         goto  isr_high_priority
        org 0x0018
         goto  isr_low_priority
        org 0x????
isr_high_priority
        ;;; ISR high priority code
        retfie 1   ;; use shadow reg

isr_low_priority
        movwf w_temp          ; context
        movff  STATUS,status_temp
        movff BSR, bsr_temp
        ;;....ISR CODE ...
        ;;.... . . .
        movff bsr_temp,bsr   ; restore context
        movf  w_temp,w
        movff  status_temp,STATUS
        retfie
```

ISR Assembly

space for w, status, bsr

high priority ISR can use shadow registers (fast stack)

low priority must explicitly save the processor context

restore context, do not use shadow registers on 'retfie'

An application: Display the digits you enter from a keypad

- You may ask: In the examples, the main function is doing nothing anyways, why not just use it for polling?
- There are situations when interrupt must be used because there are two things that must be handled simultaneously, e.g.:
  - Detecting what keys are entered from the keypad.
  - Displaying the digits that have been entered in the 4-digit 7-segment LED.

An application: Display the digits you enter from a keypad

- Main Program:
  - Declare variables to store the four digits that was entered previously. Initialize the four digits to be say "----" when nothing has been entered yet.
  - Initialize the 4-digit 7-segment LED to display the first digit.
  - Program Timer 0 to generate a 5ms delay. Turn on Timer 0.
  - Perform keypad scanning operation.
- Use of Interrupt:
  - TMR0IF is raised at the end of 5ms.
  - Get out of the main function to serve the interrupt request when TMR0IF = 1.
  - Program ISR so that the next digit is displayed.

You should be able to ....

- Identify the difference between polling and interrupt
- List the advantages of interrupts
- Enable and disable PIC18 interrupts
- Explain how interrupts are handled when global interrupt priority is disabled and enabled
- Define the priorities of different interrupt sources in PIC18
- Write interrupt service routines (ISR).