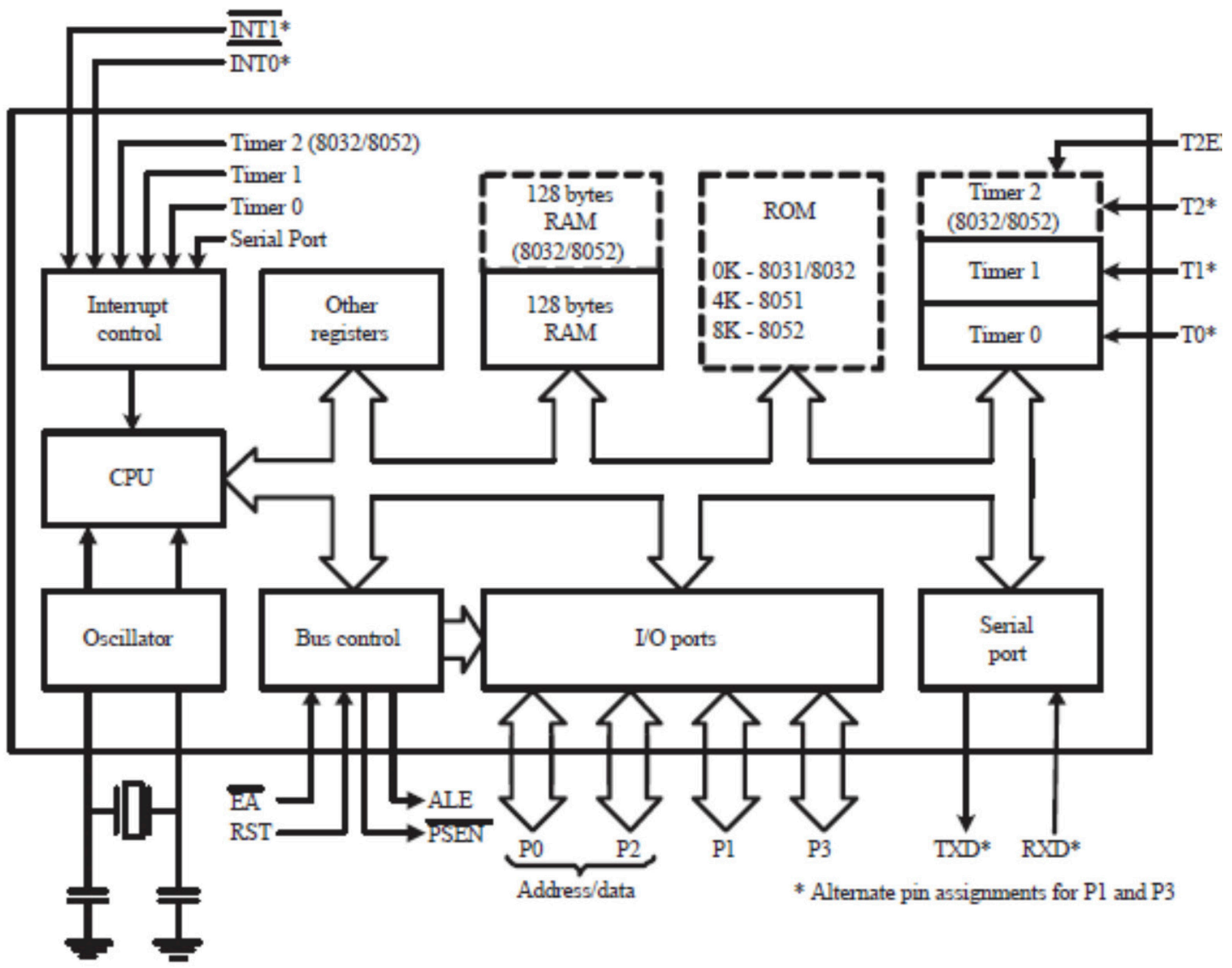
ECEN202



Mohammad Nekooei
mohammad.nekooei@vuw.ac.nz
School of Engineering and Computer Science
Victoria University of Wellington

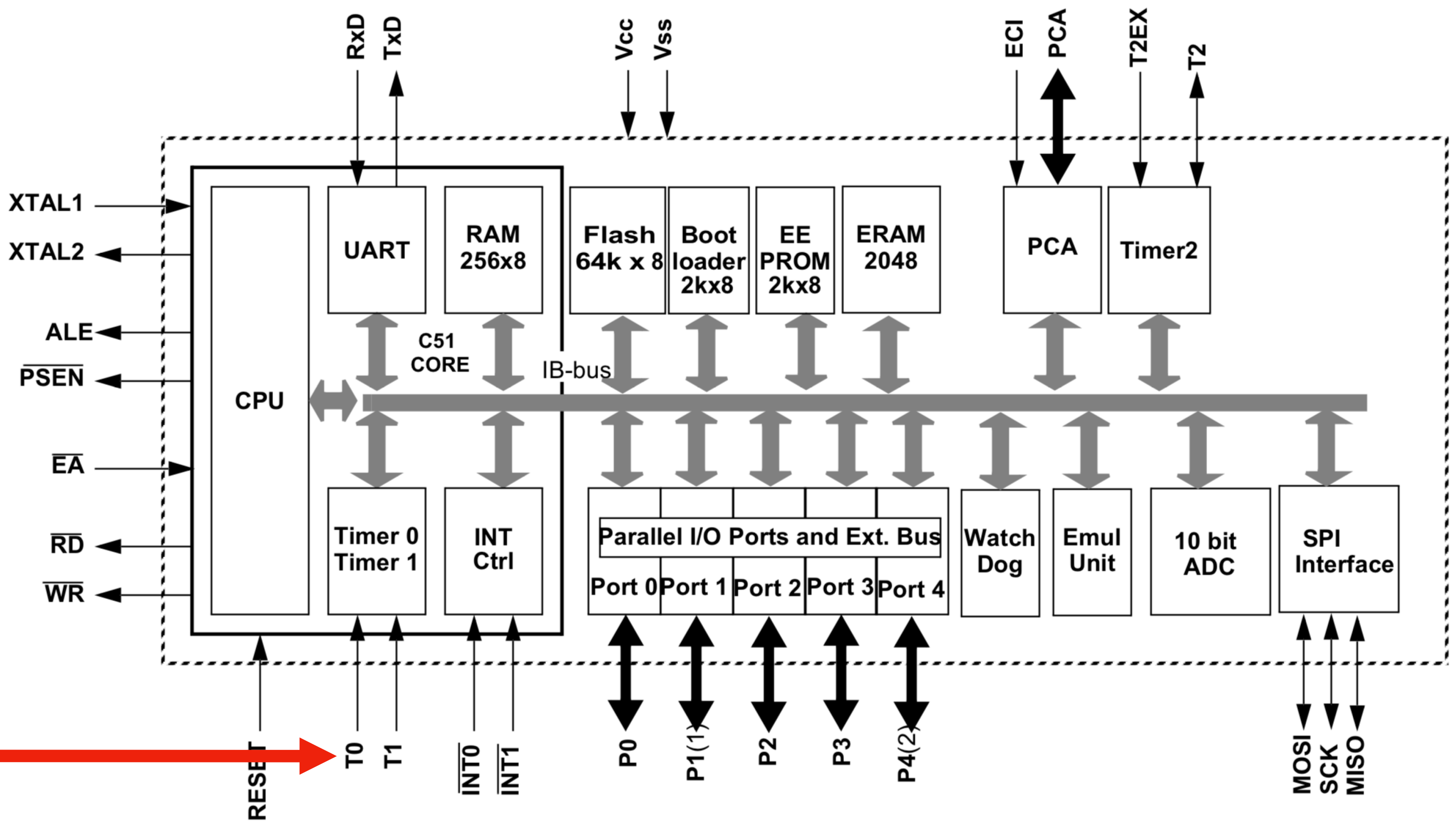
TODAY

- **Timers as counters**
- **Interrupts**

TIMERS AS COUNTERS

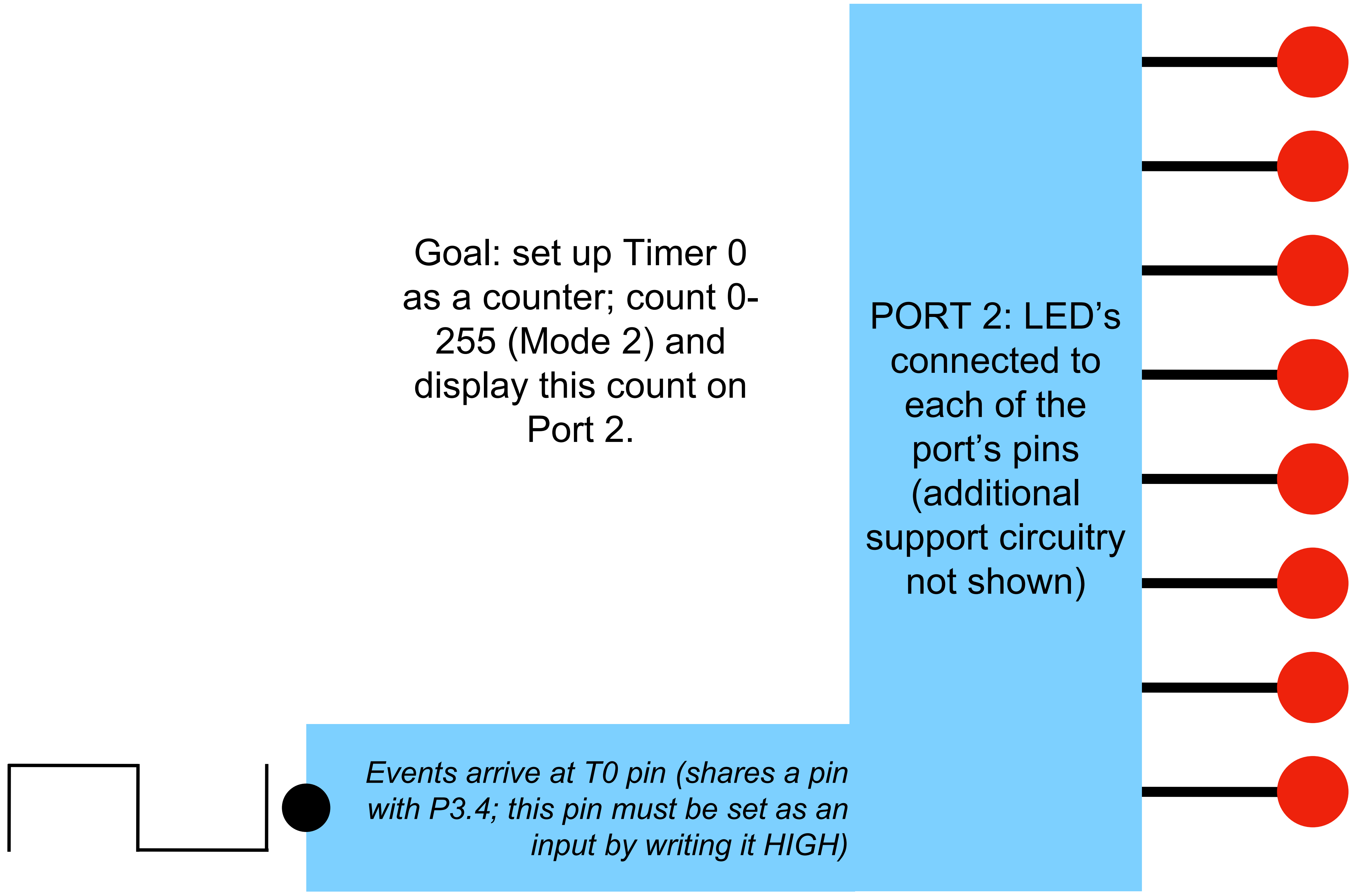
- Timers work by counting events and iterating each time an event arrives.
  - In the traditional timer role (as a ‘delay generator’), these events come from the microcontroller’s crystal oscillator (XTAL).
- We can also configure 8051 timers to count up each time another external event arrives.
  - This way, we can count things like button presses, times that sensors have exceeded thresholds, waveform edges, etc.

When T0 is in Counter mode, events arriving at T0 pin iterate the counter.
To enable the counters, set TMOD.6 or TMOD.2 to 1.



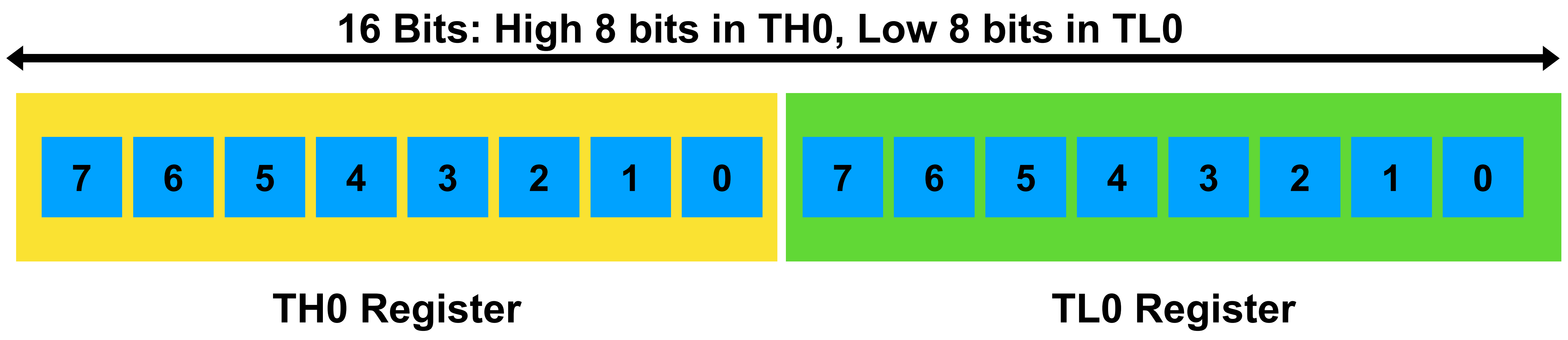
COUNTER EXAMPLE CODE

Goal: set up Timer 0 as a counter; count 0-255 (Mode 2) and display this count on Port 2.


PORT 2: LED’s connected to each of the port’s pins (additional support circuitry not shown)
Events arrive at T0 pin (shares a pin with P3.4; this pin must be set as an input by writing it HIGH)

TMOD REGISTER

TMOD.7	TMOD.6	TMOD.5	TMOD.4	TMOD.3	TMOD.2	TMOD.1	TMOD.0
GATE	C/T	M1	M0	GATE	C/T	M1	M0
When 1, timer only counts when TR1 bit is high and there is an external interrupt at INT0	When 0, Timer1 serves as XTAL-driven delay generator (timer); When 1, Timer1 counts external events	Timer 1 Mode bit 1 (see next slides for timer mode info.)	Timer 1 Mode bit 0 (see next slides for timer mode info.)	When 1, timer only counts when TR0 bit is high and there is an external interrupt at INT1	When 0, Timer0 serves as XTAL-driven delay generator (timer); When 1, Timer0 counts external events	Timer 0 Mode bit 1 (see next slides for timer mode info.)	Timer 0 Mode bit 0 (see next slides for timer mode info.)



COUNTER EXAMPLE CODE

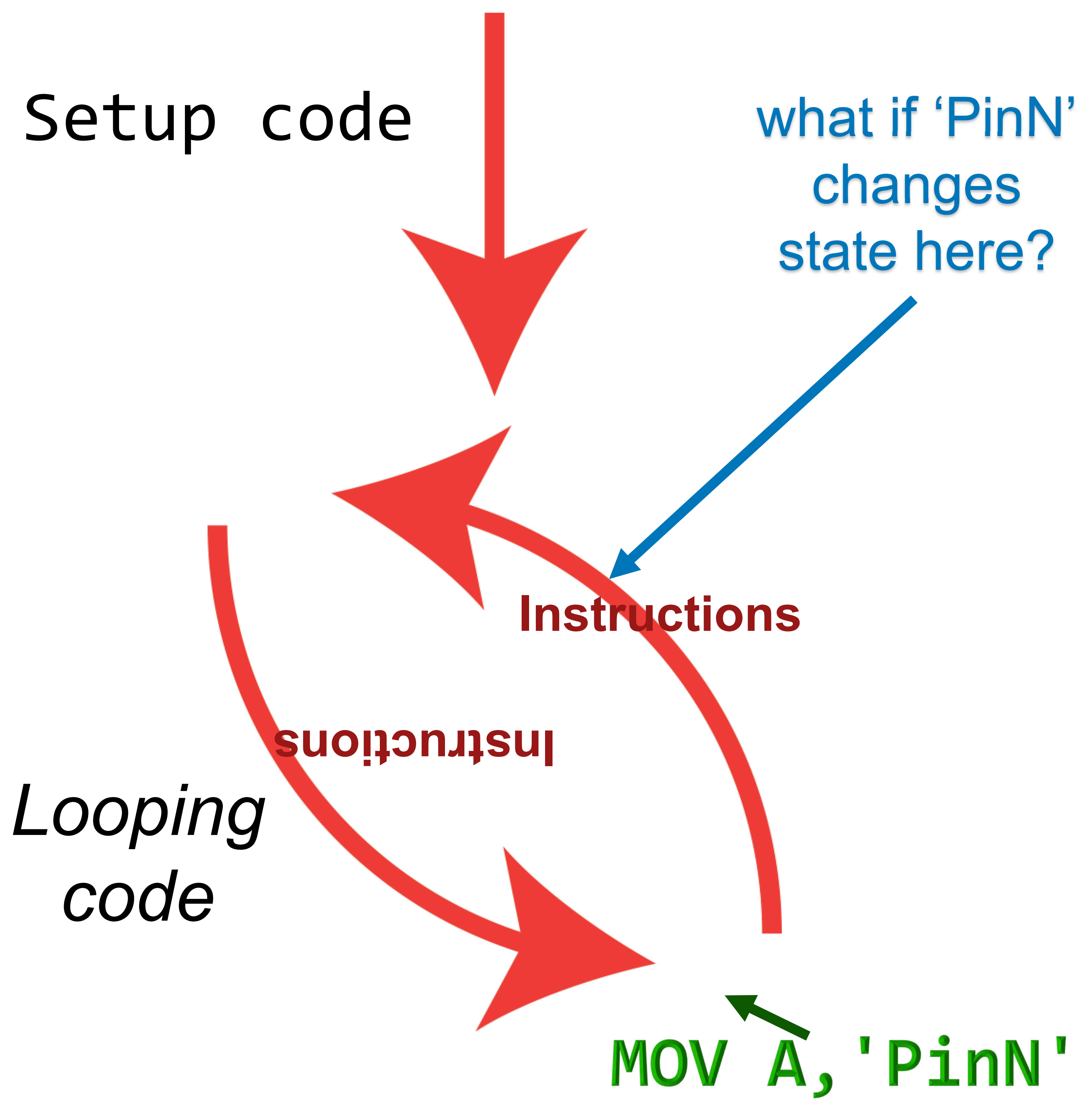
```
   MOV TMOD,#00000110B;Set timer 0 as a counter in mode2.
   MOV TH0,#0          ;Clear TH0
   SETB P3.4           ;Set T0 pin (shared with P3.4) to input
START:
   SETB TR0            ;Start timer 0

LOOP:
   MOV A,TL0           ;Grab how many events TL0 holds
   MOV P2,A            ;Output the binary count of event numbers
   JNB TF0,LOOP        ;Keep polling the TL0 port as long TF0=0
   CLR TR0             ;Stop the counter
   CLR TF0             ;Reset the event arrived flag
   SJMP START          ;Restart the timer, repeat
```

Example based upon:
https://what-when-how.com/8051-microcontroller/counter-programming/

WATCHDOG TIMER

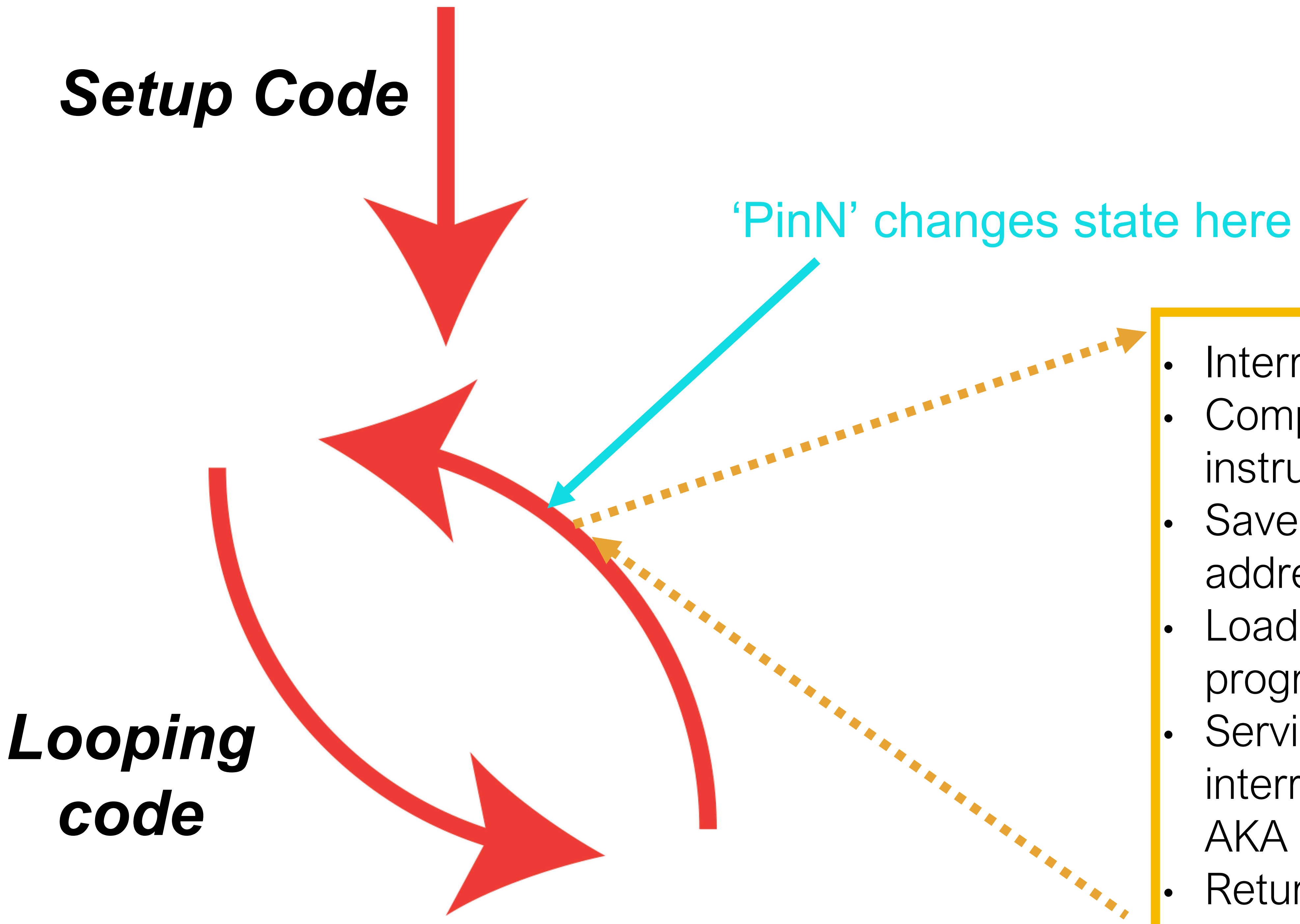
- In many critical applications, it can be beneficial to have an “emergency reset” switch.
  - This will allow us to reset our microcontroller even if the program hangs up on some routine.
    - Useful to let us automatically recover from unexpected ‘crashes.’
  - Microcontrollers’ watchdog timers (WDT’s) will automatically reset the system (moving PC to 0, etc.) if they are allowed to overflow.
    - In a normally operating system, WDTs are reset in software every so often. If this reset does not occur, then we know that the system is hanging and is in need of a reset.
- The 8051 features a watchdog timer.
  - The C8051F02x has a 21 bit timer of which 7 bits can be set, allowing custom time-out intervals (of 16 ms to 2 s).
    - We won’t study this in detail, but keep watchdog timers in mind when building systems that need to automatically recover from unexpected errors.
    - For more information on setting up and using the 8051’s watchdog timer, see page 129 of C8051F02x.pdf.pdf.

BLOCKING CODE



What if ‘PinN’ changes during one of the blocks of instructions?
And what if it’s *very* important that we not miss ‘PinN’ changing state?
We might solve this by configuring an interrupt to call some code and leave the pre-defined instructions whenever the pin changes.

INTERRUPT-DRIVEN CODE



- Interrupt flag raised
- Complete current instruction
- Save program counter address of next inst.
- Load interrupt vector into program counter
- Service interrupt (ISR = interrupt service routine), AKA 'handler'
- Return to loop (restore program counter), RETI

handling an interrupt

Introduction to Interrupts

- An interrupt is the occurrence of a condition that causes a temporary suspension of a program while the condition is serviced by another (sub) program

- Interrupts are important because they allow a system to respond asynchronously to an event and deal with the event while in the middle of performing another task

- An **interrupt-driven system** gives the illusion of doing many things simultaneously

- The (sub) program that deals with an interrupt is called an **interrupt service routine (ISR) or interrupt handler**

Introduction

- The ISR executes in response to the interrupt and generally performs an input or output operation to a device
- When an interrupt occurs, the main program temporarily suspends execution and branches to the ISR
- The ISR executes, performs the desired operation, and terminates with a "return from interrupt" (RETI) instruction
  - The RETI instruction is different from the normal "RET" instruction

Interrupt Organization

- The 8051 supports many interrupt sources, including:
  - external interrupts
  - timer interrupts
  - serial port interrupts
  - Each interrupt source has one or more associated **interrupt-pending** flag(s) located in an SFR
- When a peripheral or external source meets a valid interrupt condition, the associated interrupt-pending flag is set to 1
  - These interrupt flags are “level sensitive” in that if the flag is not cleared in the ISR by either hardware or software, the interrupt will trigger again, even if the event that originally caused the interrupt did not occur again
- All interrupts are disabled after a system reset and enabled individually by software

INTERRUPTS

- Interrupts allow special routines to be run when the microcontroller enters certain states.
  - We'll discuss the 8051-specific states below, but interrupts are often associated with microcontrollers' hardware peripherals:
    - When some hardware peripheral (timer, serial port, external I/O port, etc.) changes state, main program flow can be set to be interrupted to handle this change in a timely manner.
- Interrupts disrupt the running of normal code.
  - It is important to follow best practices with interrupts to keep from disrupting normally-running code:
    - Keep interrupt service routines short.
    - Try to avoid calling subroutines within your interrupt service routine.
      - Ask what might happen if an interrupt is itself interrupted…
        - Interrupts are handled in order of a defined priority.
        - Consider interrupt priority: which interrupt is most important?
    - If particularly time-sensitive code is used (e.g., bit-banging emulation of a serial port), consider disabling interrupts during this routine.
      - Re-enable after this routine is finished.

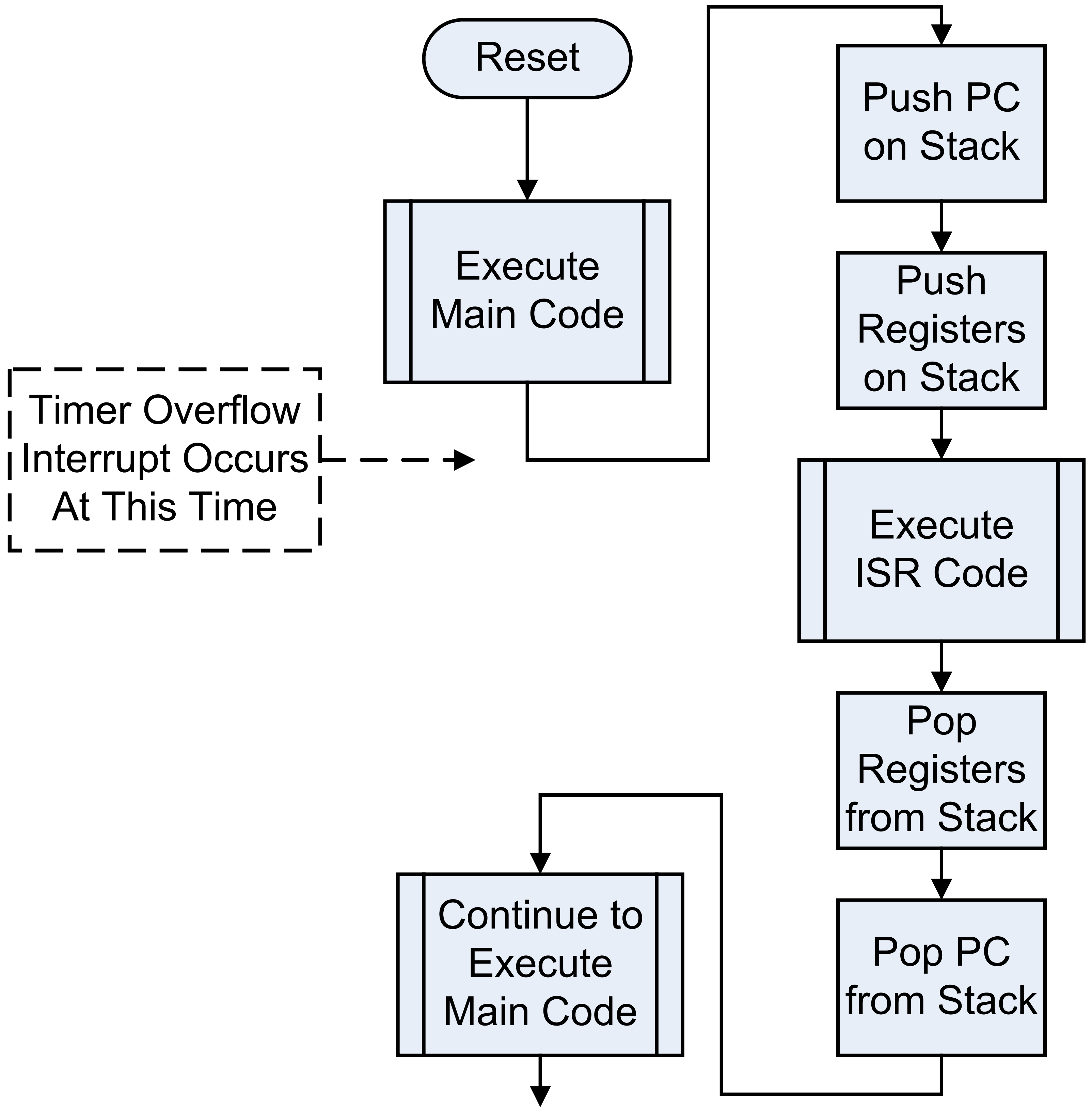
INTERRUPTS

- The ISR executes in response to the interrupt and generally performs an input or output operation to a device

- When an interrupt occurs, the main program temporarily suspends execution and branches to the ISR

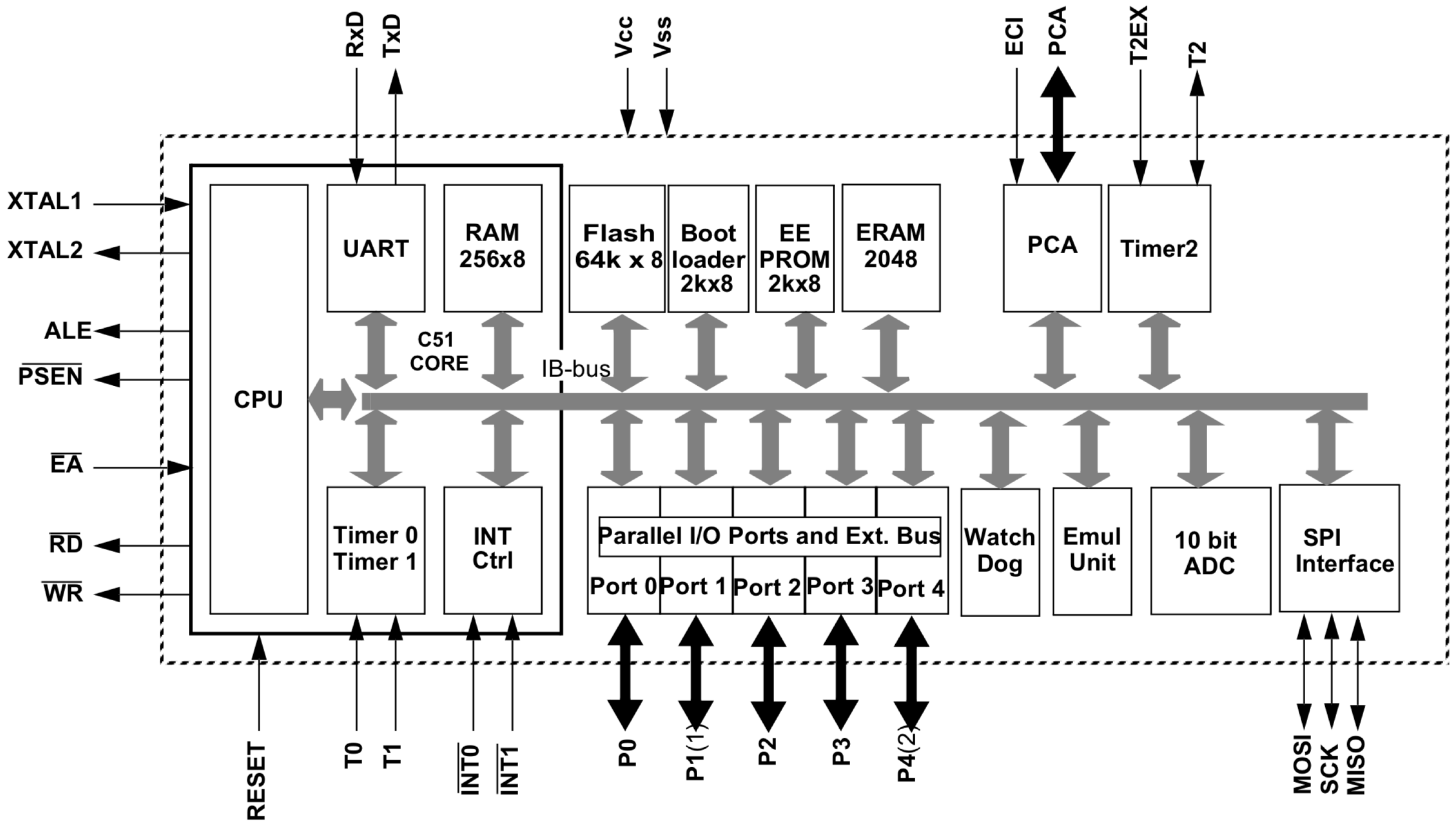
- The ISR executes, performs the desired operation, and terminates with a "return from interrupt" (RETI) instruction
  - The RETI instruction is different from the normal "RET" instruction

INTERRUPTS


Reset
Execute Main Code
Timer Overflow Interrupt Occurs At This Time
Push PC on Stack
Push Registers on Stack
Execute ISR Code
Pop Registers from Stack
Pop PC from Stack
Continue to Execute Main Code

8051 INTERRUPTS

- The original 8051 has four interrupt types.
  - RESET: When reset , PC is loaded with ROM address 0x0000
    - Unlike other interrupts, prior addresses aren't loaded to stack, etc.
  - TIMER INTERRUPTS: one interrupt for Timer 0 and one for Timer 1.
    - Timer 0 interrupt: PC is vectored to 0x000B
    - Timer 1 interrupt: PC is vectored to 0x001B
  - EXTERNAL INTERRUPTS: Interrupts that are called in response to external signals.
    - External interrupt 0 - INT0 (AKA EXT1): PC vectored to ROM 0x0003.
    - External interrupt 1 - INT1 (AKA EXT2): PC vectored to ROM 0x0013.
  - SERIAL INTERRUPT: Interrupt that is called in response to serial events.
    - Serial interrupt: PC is vectored to ROM 0x0023.
- The C8051F020x has additional interrupts; we'll look at these later.
- If the ISR is short, it can fit in the 8 bytes allocated to the interrupts.
  - If the ISR is longer, the vector address holds an LJMP instruction that points to another memory location, allowing for a longer ISR.

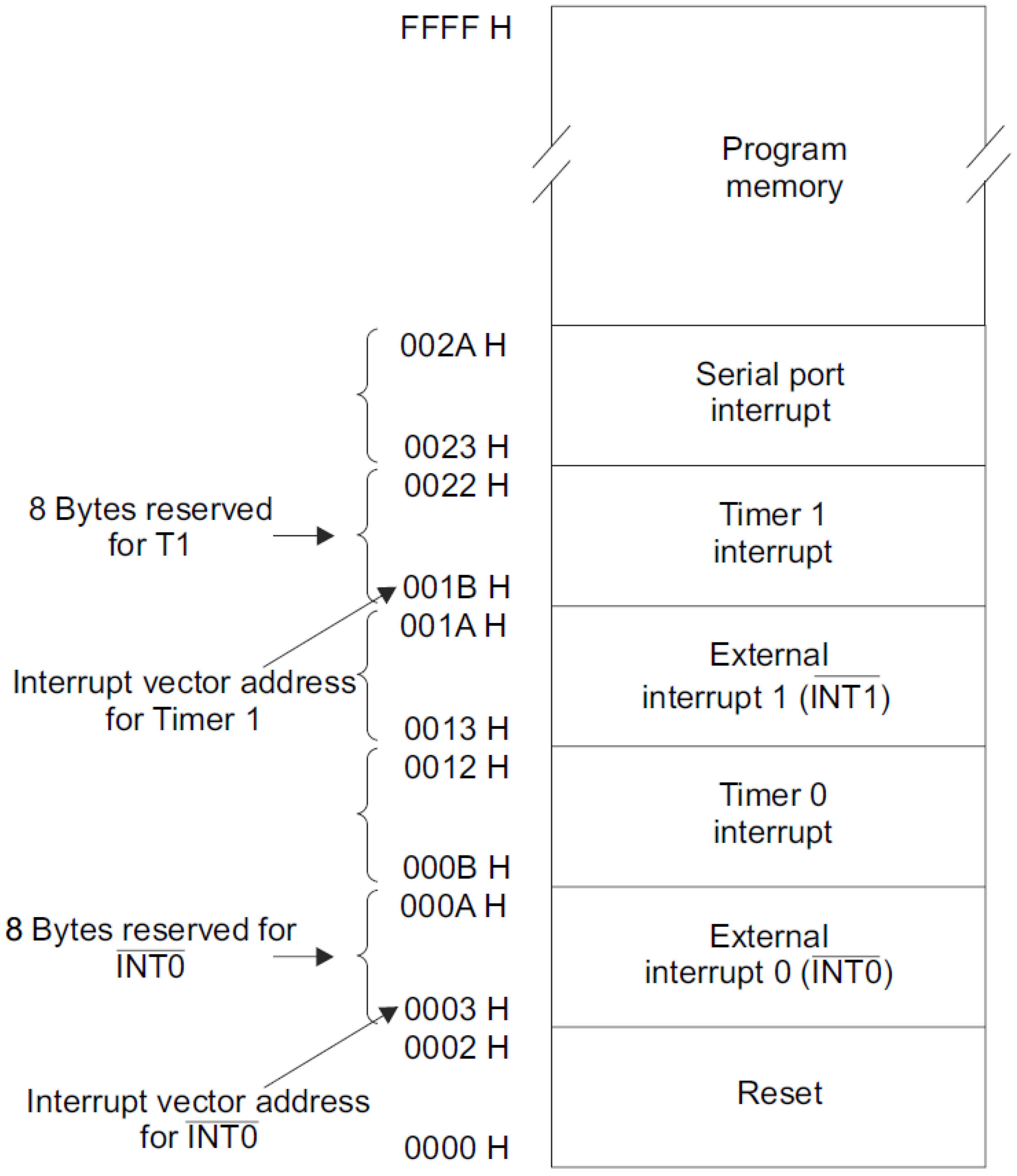


RxD
TxD
Vcc
Vss
ECI
PCA
T2EX
T2
XTAL1
XTAL2
ALE
PSEN
EA
RD
WR
CPU
UART
RAM 256x8
C51 CORE
IB-bus
Flash 64k x 8
Boot loader 2kx8
EE PROM 2kx8
ERAM 2048
PCA
Timer2
Timer 0 Timer 1
INT Ctrl
Parallel I/O Ports and Ext. Bus
Port 0
Port 1
Port 2
Port 3
Port 4
Watch Dog
Emul Unit
10 bit ADC
SPI Interface
RESET
T0
T1
INT0
INT1
P0
P1(1)
P2
P3
P4(2)
MOSI
SCK
MISO

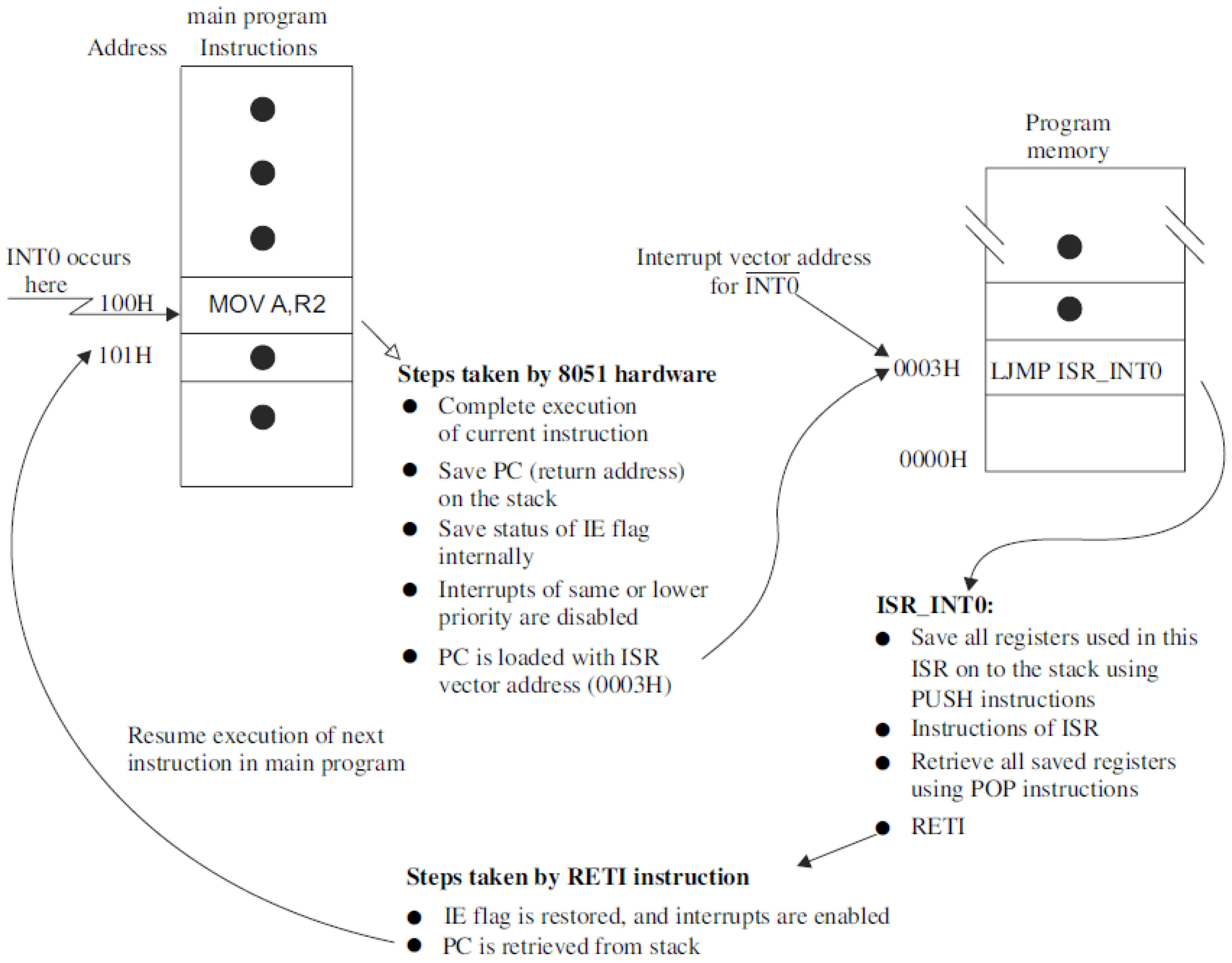
INTERRUPT VECTOR TABLE

INTERRUPT	ROM ADDRESS (for start of ISR)	INTERRUPT PIN	NOTES
RESET	0x0000	9	Note only 3 bytes of ROM between this and INT0. See example, next slide.
INT0	0x0003	P3.2	Interrupt flag auto-clears
Timer0	0x000B	N/A	Interrupt flag auto-clears
INT1	0x0013	P3.3	Interrupt flag auto-clears
Timer1	0x001B	N/A	Interrupt flag auto-clears
Serial Port	0x0023	N/A	Software must clear this interrupt flag.

INTERRUPT VECTOR TABLE


FFFF H
Program memory
002A H
0023 H
Serial port interrupt
0022 H
8 Bytes reserved for T1
Timer 1 interrupt
001B H
001A H
Interrupt vector address for Timer 1
External interrupt 1 ($\overline{INT1}$)
0013 H
0012 H
Timer 0 interrupt
000B H
000A H
8 Bytes reserved for $\overline{INT0}$
External interrupt 0 ($\overline{INT0}$)
0003 H
0002 H
Reset
Interrupt vector address for $\overline{INT0}$
0000 H

INTERRUPT VECTOR TABLE



Steps taken by 8051 hardware

- Complete execution of current instruction
- Save PC (return address) on the stack
- Save status of IE flag internally
- Interrupts of same or lower priority are disabled
- PC is loaded with ISR vector address (0003H)

ISR_INT0:

- Save all registers used in this ISR on to the stack using PUSH instructions
- Instructions of ISR
- Retrieve all saved registers using POP instructions
- RETI

Steps taken by RETI instruction

- IE flag is restored, and interrupts are enabled
- PC is retrieved from stack

Resume execution of next instruction in main program

RESET

- Note the low ROM addresses for the interrupt vector table addresses.
  - Also, note how Reset has only three bytes of ROM (0x0000-0x0002).
    - If we're not careful, our PC will just step through the other ISR's and execute them. Instead, we must quickly LJMP past these ISR addresses into an un-used memory space.
    - We can use the ORG directive to tell our assembler to assemble the code to make sure that we bypass this vector table.

```
ORG 0     ;This is our RESET address.
LJMP BODY ;We jump past the interrupt vector table addresses

ORG 30H ;We use org directive to associate the following
        ; instructions with addresses past 30H
BODY:
;Our non-ISR code goes here, since we set ORG past our ISR
;vector table addresses
END
```

THE IE (IEN0) REGISTER

- The special function register that allows for interrupt control is the IE register.
  - IE = interrupt enable.
- Allows for bit-by-bit control of each individual interrupt as well as all interrupts at once.
- Original 8051's have a single IE register; the C8051F020x has two.
  - We'll focus on the first one, called IEN0 on the C8051F020x.

IEN0.7	IEN0.6	IEN0.5	IEN0.4	IEN0.3	IEN0.2	IEN0.1	IEN0.0
EA	EC	ET2	ES	ET1	EX1	ET0	EX0
Enable all interrupts 1: interrupts may be enabled individually 0: All interrupts are disabled	**PCA interrupt enable** When 1, PCA interrupt is enabled. A C8051F020 specific interrupt, discussed later	Timer 2 Overflow Interrupt Enable When 1, Timer 2 interrupt is enabled.	Serial Port interrupt enable bit. When 1, serial port interrupt is enabled.	Timer 1 Overflow Interrupt Enable When 1, Timer 1 interrupt is enabled.	External interrupt 1 enable bit. When 1, External Interrupt 1 is enabled.	Timer 0 Overflow Interrupt Enable When 1, Timer 0 interrupt is enabled.	External interrupt 0 enable bit. When 1, External Interrupt 0 is enabled.

Interrupt enable registers

IEN0 (S:A8h)
Interrupt Enable Register

7	6	5	4	3	2	1	0
EA	**EC**	**ET2**	**ES**	**ET1**	**EX1**	**ET0**	**EX0**

Bit Number	**Bit Mnemonic**	**Description**
7	EA	**Enable All Interrupt bit** Clear to disable all interrupts. Set to enable all interrupts. If EA=1, each interrupt source is individually enabled or disabled by setting or clearing its interrupt enable bit.
6	EC	**PCA Interrupt Enable** Clear to disable the PCA interrupt. Set to enable the PCA interrupt.
5	ET2	**Timer 2 Overflow Interrupt Enable bit** Clear to disable Timer 2 overflow interrupt. Set to enable Timer 2 overflow interrupt.
4	ES	**Serial Port Enable bit** Clear to disable serial port interrupt. Set to enable serial port interrupt.
3	ET1	**Timer 1 Overflow Interrupt Enable bit** Clear to disable timer 1 overflow interrupt. Set to enable timer 1 overflow interrupt.
2	EX1	**External Interrupt 1 Enable bit** Clear to disable external interrupt 1. Set to enable external interrupt 1.
1	ET0	**Timer 0 Overflow Interrupt Enable bit** Clear to disable timer 0 overflow interrupt. Set to enable timer 0 overflow interrupt.
0	EX0	**External Interrupt 0 Enable bit** Clear to disable external interrupt 0. Set to enable external interrupt 0.

Reset Value = 0000 0000b
bit addressable

INTERRUPT PRIORITY

- If we're implementing an interrupt-rich application, we need to consider what happens if multiple interrupts arrive at the same time (interrupt during ISR).
  - Older 8051's had less flexible interrupt priority control.
    - For more on original 8051 priority, see https://what-when-how.com/8051-microcontroller/interrupt-priority-in-the-805152/
    - The C8051F020x has a much more modern interrupt priority configuration capability, which we'll focus upon.
    - Upon boot-up, the interrupts are given default priority hierarchy.
    - We can change this priority hierarchy by manipulating two SFR's.
      - IPL0 and IPH0 registers.
    - By default, the interrupt priority is:

DECREASING PRIORITY

1. External interrupt 0 (INT0)
2. Timer 0 (TF0)
3. External interrupt 1 (INT1)
4. Timer 1 (TF1)
5. Programmable counter array *(discussed later)* (CF)
6. Serial UART (RI or TI)
7. Timer 2 (TF2)
8. ADC *(discussed later)* (ADCI)
9. SPI *(discussed later)*

SETTING INTERRUPT PRIORITY

- Each interrupt has two bits that can set its priority.
    - These bits are in the IPH0 and IPL0 registers.
        - These two bits give four possible levels of interrupt priority.
- If two interrupts are configured with the same priority level, then they are handled in order of their ranking in the default priority level scheme (prev. slide).

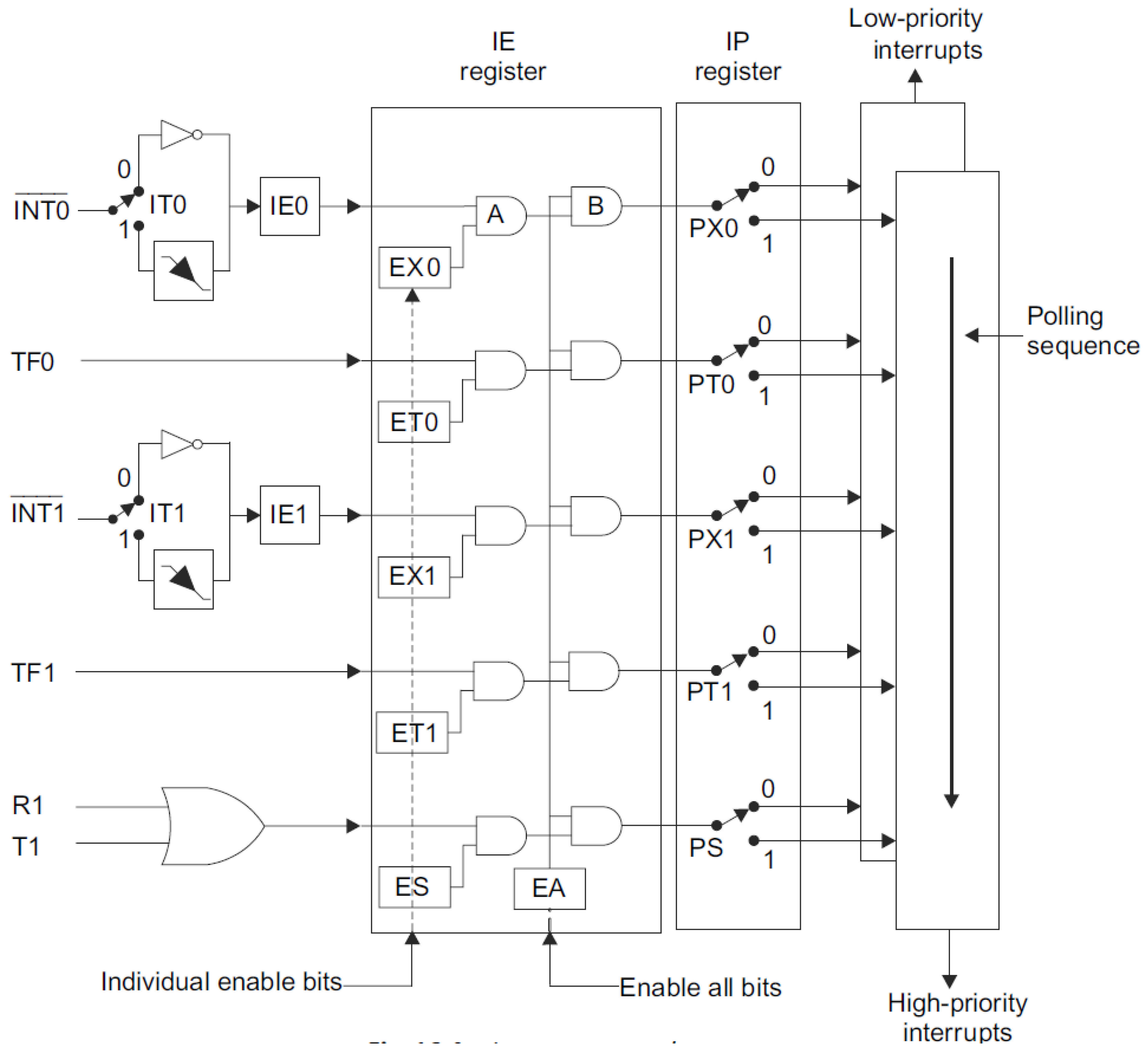
IPH.x	**IPL.x**	**Priority level (0: lowest, 3: highest)**
0	0	0
0	1	1
1	0	2
1	1	3

THE IPL AND IPH REGISTERS

IPH0	IPL0
7. (Reserved)	7. (Reserved)
6. PPCH (PCA Int)	6. PPC (PCA Int)
5. PT2H (Timer2)	5. PT2 (Timer2)
4. PSH (Serial port)	4. PS (Serial port)
3. PT1H (Timer 1)	3. PT1 (Timer 1)
2. PX1H (INT1)	2. PX1 (INT1)
1. PT0H (Timer 0)	1. PT0 (Timer 0)
0. PX0H (INT0)	0. PX0 (INT0)

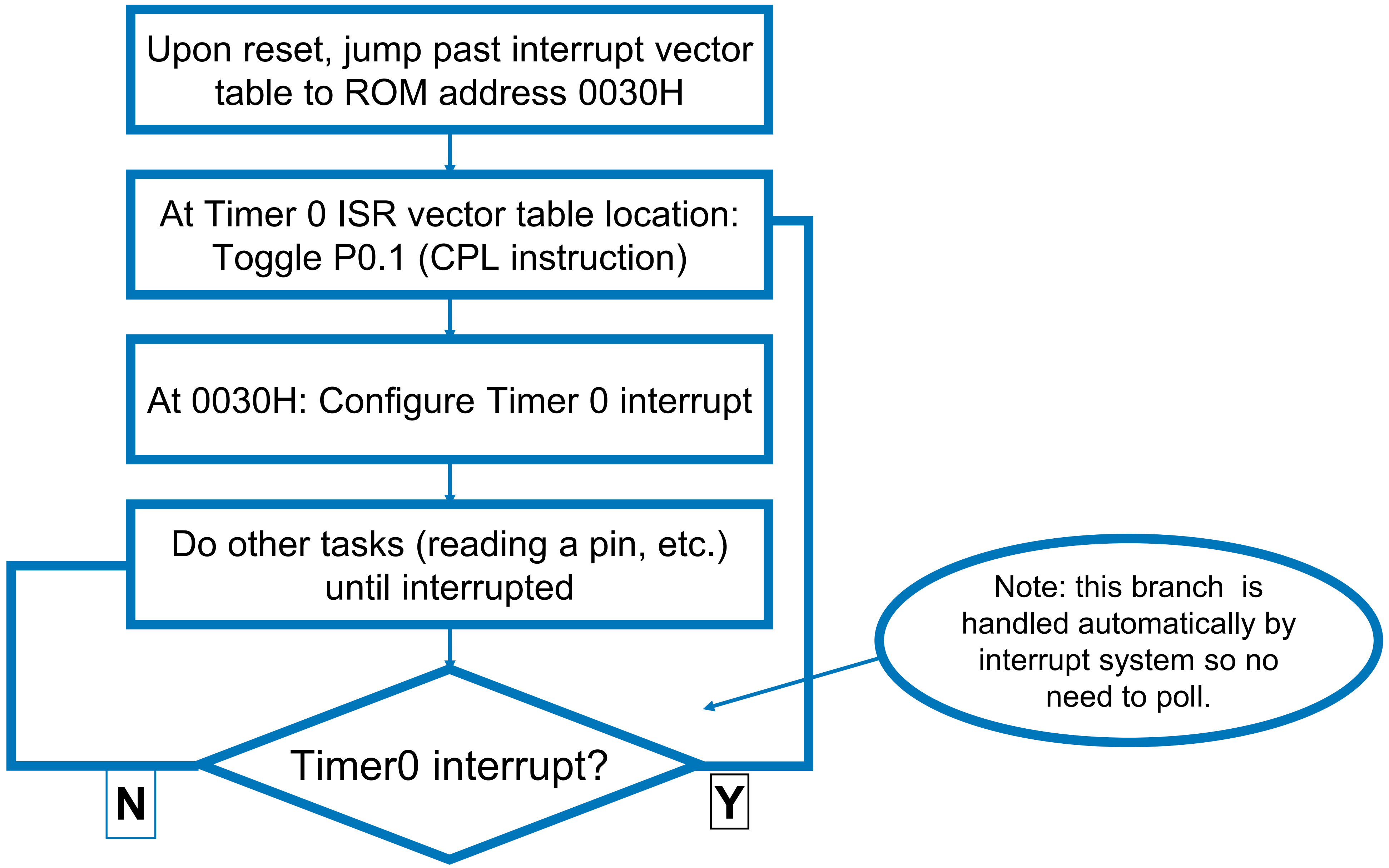
IPH.x	IPL.x	Priority level (0: lowest, 3: highest)
0	0	0
0	1	1
1	0	2
1	1	3

Interrupt control structure



TIMER INTERRUPTS

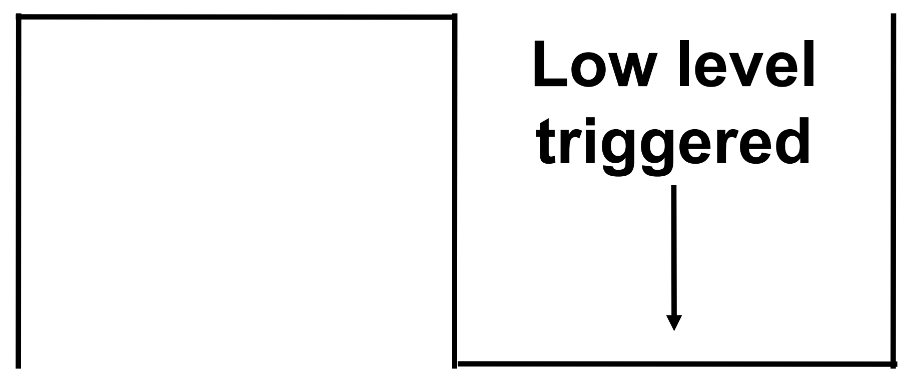
- Goal: replicate the "EXAMPLE SQUARE WAVE" code from a previous slide, replacing polling approach with an interrupt-driven approach.
  - Advantages: we can free up the CPU from having to monitor the TFx flag.
- For this example, we'll keep the ISR short so it will fit within the 8 Bytes of ROM allocated by the Interrupt Vector Table.

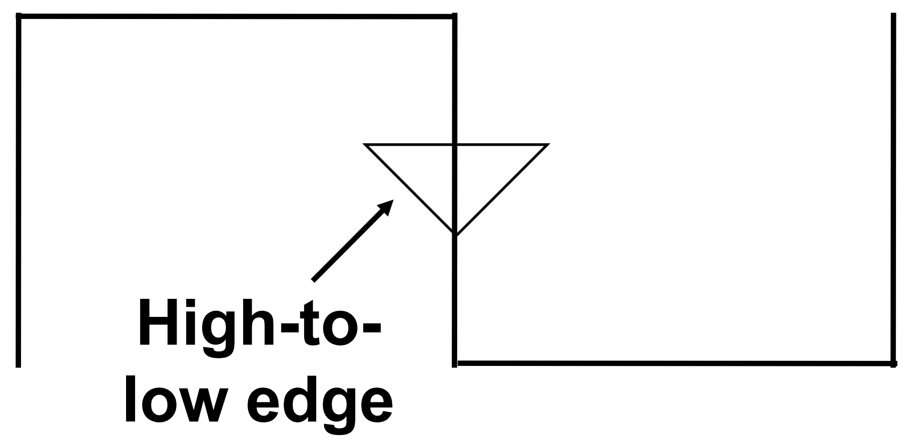

Upon reset, jump past interrupt vector table to ROM address 0030H
At Timer 0 ISR vector table location: Toggle P0.1 (CPL instruction)
At 0030H: Configure Timer 0 interrupt
Do other tasks (reading a pin, etc.) until interrupted
Timer0 interrupt?
N
Y
Note: this branch is handled automatically by interrupt system so no need to poll.

TIMER INTERRUPT EXAMPLE

```
;We'll be using the interrupt address space, so we need to make
;sure that our main program bypasses this.
ORG 0000H
LJMP MAIN     ;Jump past the interrupt vector table.
;——ISR for Timer 0. Will generate a square wave at Port 0.1
ORG 000BH      ;Assembler will place code at Timer0 vector adr.
CPL P0.1       ;Toggle Port 0 pin 1
MOV TL0,#00    ;Must reload timer values manually in this mode
MOV TLH,#80H
RETI           ;Return from int.; pops PC and resets interrupt logic
;—— Main body of program. Start by configuring Timer 0
ORG 0030H      ;Directs assembler past interrupt vector table
MAIN: MOV TMOD, #00000001B ;Timer 0, mode 1
MOV TL0,#00    ;Initially load timer values
MOV TLH,#80H
MOV IEN0, #10000010B ;Enable timer 0 interrupt
SETB TR0       ;Start the timer
IDLE: SJMP IDLE ;Loop here until interrupt. No polling of int flag!
END
```

EXTERNAL INTERRUPTS





- In addition to interrupting normal program flow in response to a timer, external pin activations can be set to interrupt program flow.
- On the 8051, interrupts can be set to be triggered in response to two different types of pin states/state changes:
  - Low level triggered: when the external interrupt pin is LOW, the interrupt is triggered.
    - The interrupt will be re-triggered if the pin remains low after the ISR!
  - Edge triggered interrupt: interrupt is triggered on falling edge of signal at external interrupt pin.
    - Falling edge: when pin goes from HIGH to LOW.
    - Low-level or falling edge mode: configured in TCON register.
    - Need to trigger on rising edge? Use an inverter chip (e.g., 7404)

EXTERNAL INTERRUPT CONFIGURATION

- Low 4 bits of TCON register: configure/examine external interrupts.
  - TCON.3 (IE1) and TCON.1 (IE0): ‘external interrupt occurred’ flags.
  - TCON.2 (IT1) and TCON.0 (IT0): set interrupt trigger type.

TCON.7	TCON.6	TCON.5	TCON.4	TCON.3	TCON.2	TCON.1	TCON.0
TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
Timer 1 Overflow Flag 1 when overflow occurs. Must be cleared in software; auto. cleared when leaving ISR	Timer 1 run bit 1: Start timer 0: Stop timer (Software controlled)	Timer 0 Overflow Flag 1 when overflow occurs. Must be cleared in software; auto. cleared when leaving ISR	Timer 0 run bit 1: Start timer 0: Stop timer (Software controlled	Ext. interrupt1 edge flag. 1: external interrupt occurred. 0: External interrupt processed. (Hardware controlled; no need to edit this)	Interrupt1 trigger type select bit. 1: Interrupt occurs on the falling edge of INT1. 0: Interrupt occurs on INT1’s level being LOW.	Ext. interrupt0 edge flag. 1: external interrupt occurred. 0: External interrupt processed. (Hardware controlled; no need to edit this)	Interrupt0 trigger type select bit. 1: Interrupt occurs on the falling edge of INT1. 0: Interrupt occurs on INT1’s level being LOW.

EXTERNAL INTERRUPT EXAMPLE

- Goal: Toggle an LED connected to Port 1, pin 3.
  - This LED will be toggled when an external interrupt (INT1) arrives.

```
ORG 0000H
LJMP MAIN     ;Jump past the interrupt vector table.

;– ISR: Interrupt 1, toggles LED when new interrupt arrives.
ORG 0013H     ;Location in vector table of INT1
CPL P1.3      ;Toggle Port 1 pin 3
RETI          ;Reset PC and clear interrupt flags

;Set up interrupts at ROM location past vector table
ORG 0030H
MAIN: SETB TCON.2 ;Interrupt is falling edge triggered
MOV IEN0 #10000100B ;Enable interrupts, INT1

IDLE: SJMP IDLE ;Other code could go here. Idle main CPU for now.
END
```