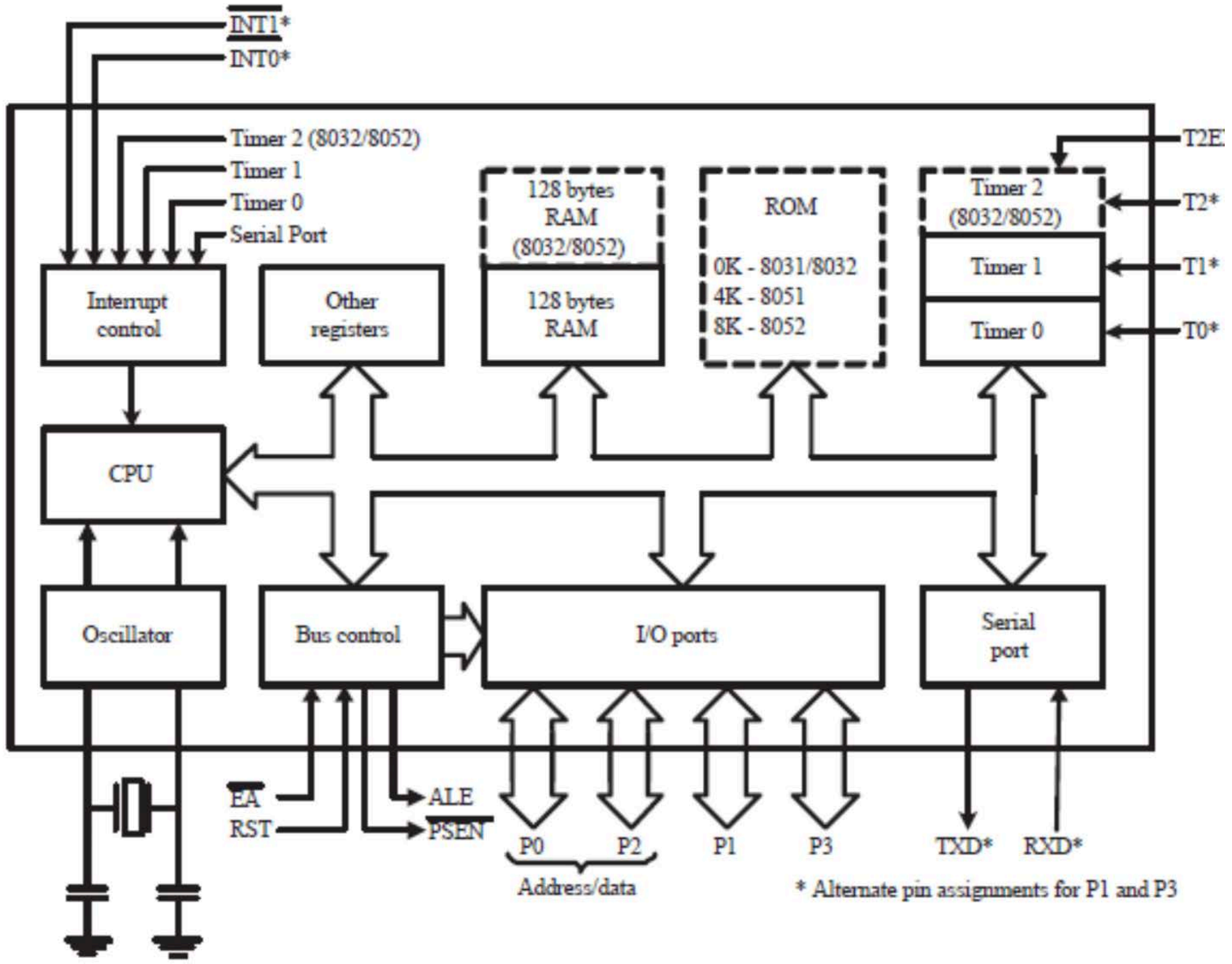
# ECEN202

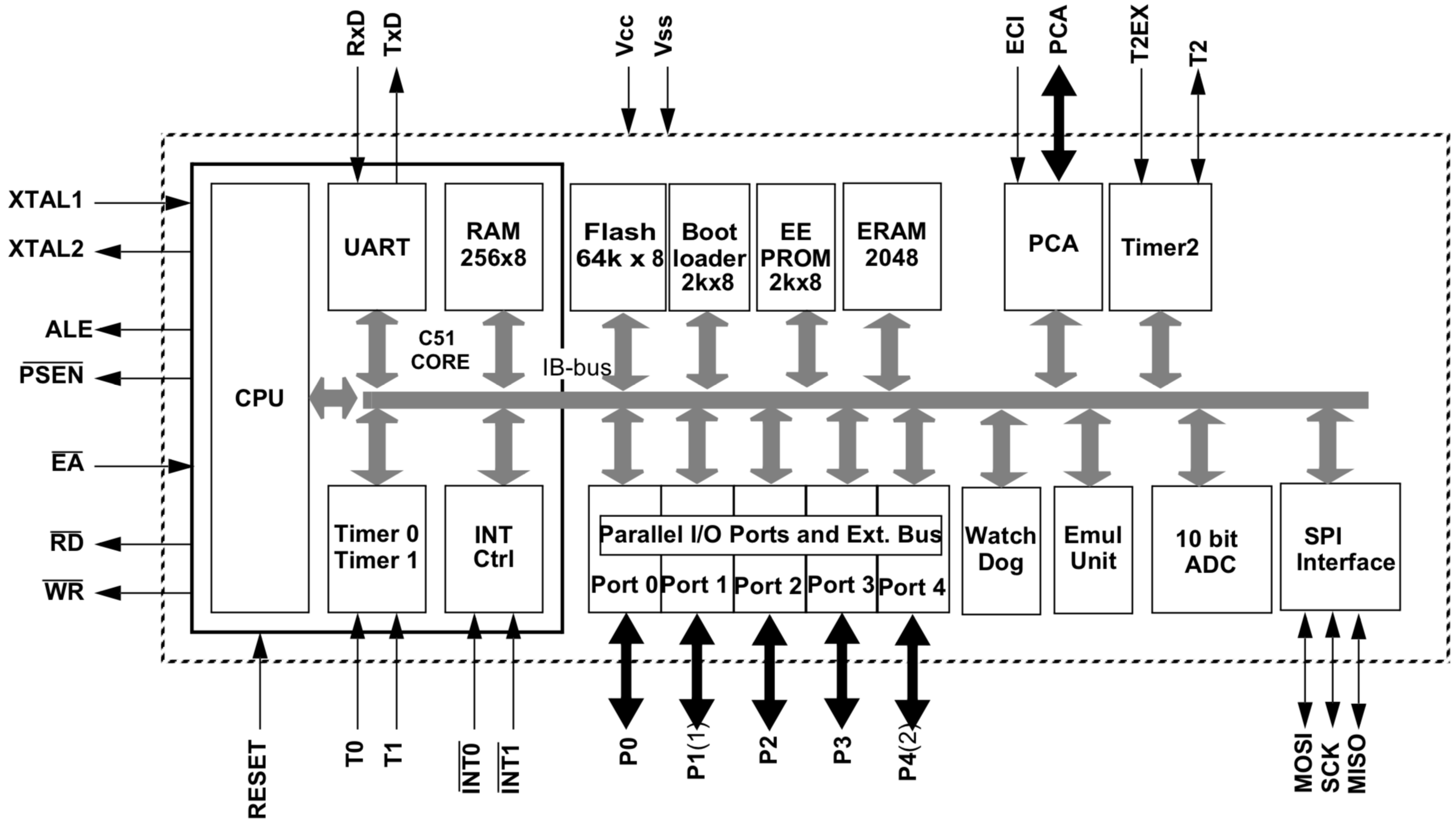


Mohammad Nekooei
mohammad.nekooei@vuw.ac.nz
School of Engineering and Computer Science
Victoria University of Wellington

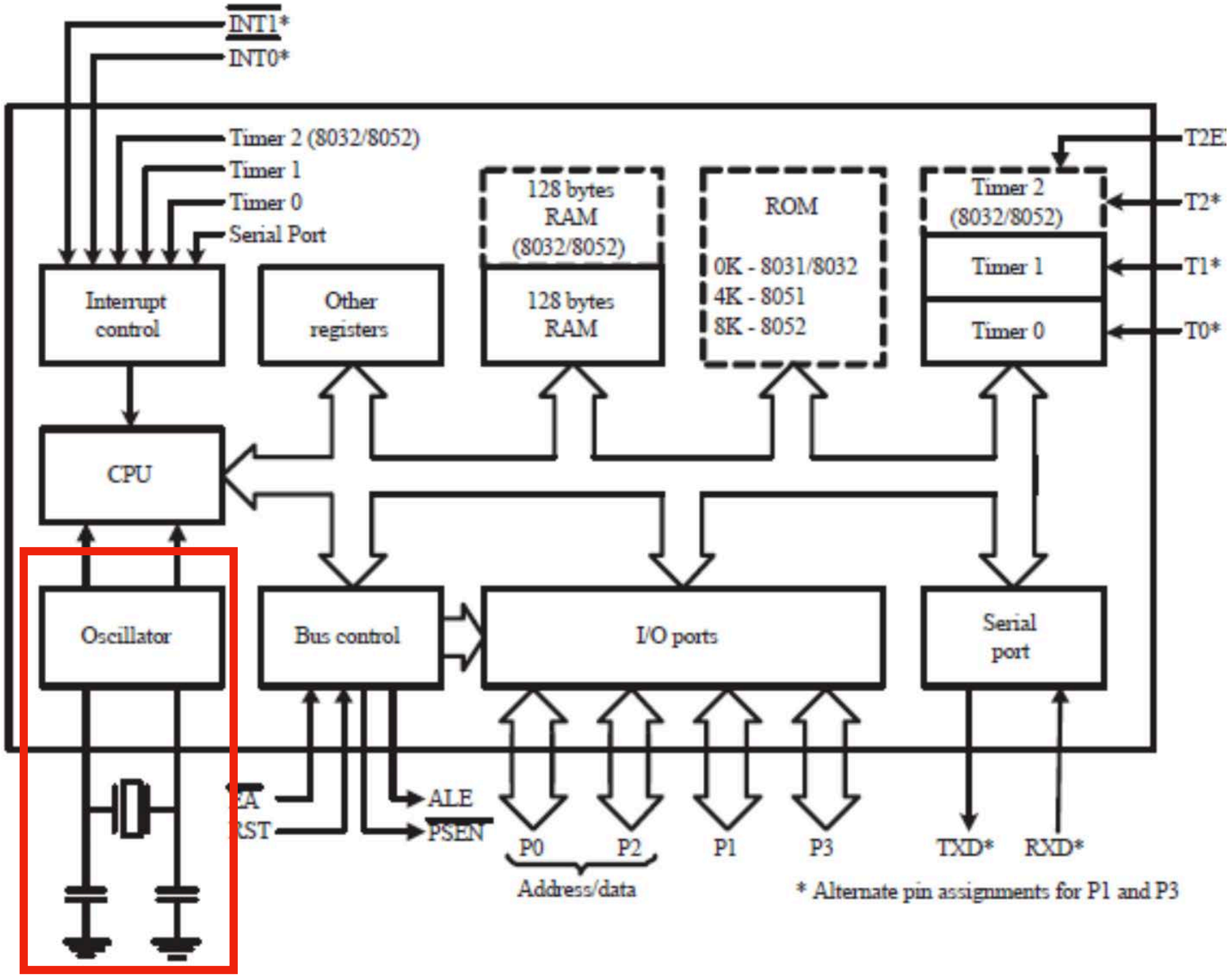
## TODAY

- **Timers & Counters**
- **Interrupts**
- ***Relatively big topic, so we might overflow to next week on these items.***

# 8051: BLOCK DIAGRAM



# 8051: BLOCK DIAGRAM

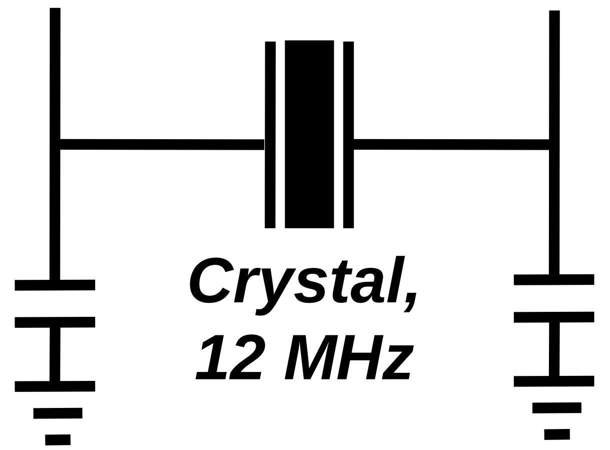


CLOCK SOURCE

# Recap: CLOCK SOURCE

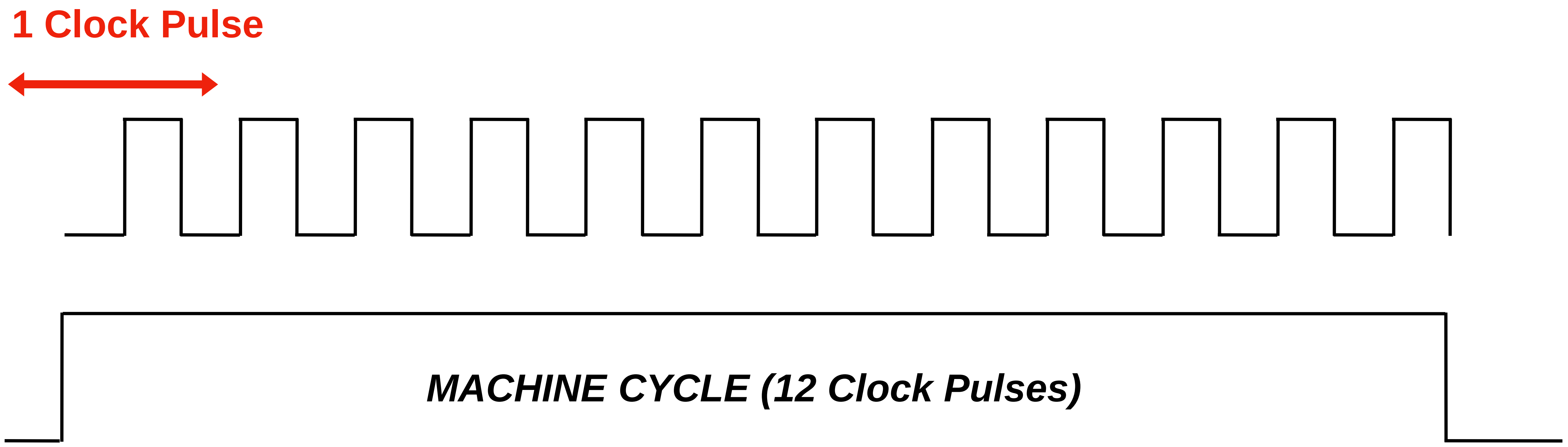
- 8051 programs are run sequentially, one instruction executing after the previous one.
  - A clock source is needed to increment a program counter (discussed later).
  - Other functions within the microcontroller (timers, communications, etc.) need to operate synchronously as well.
    - To make sure that all of these operate synchronously, the 8051 uses a master oscillator.
- The 8051's oscillator outputs a square wave. A relatively stable crystal (external) determines this square wave's period.

OSCILLATOR HARDWARE
*(internal, generates square wave from crystal's oscillation)*



- Many modern processors can execute one instruction per clock cycle. (or more!)
  - The 8051 requires 12 clock cycles per "machine cycle".
    - Instructions on the 8051 require 1 or 2 machine cycles
  - Given a 12 MHz clock, this means that we can execute 0.5-1 million instructions per second (MIPS).

# Recap: MACHINE CYCLE



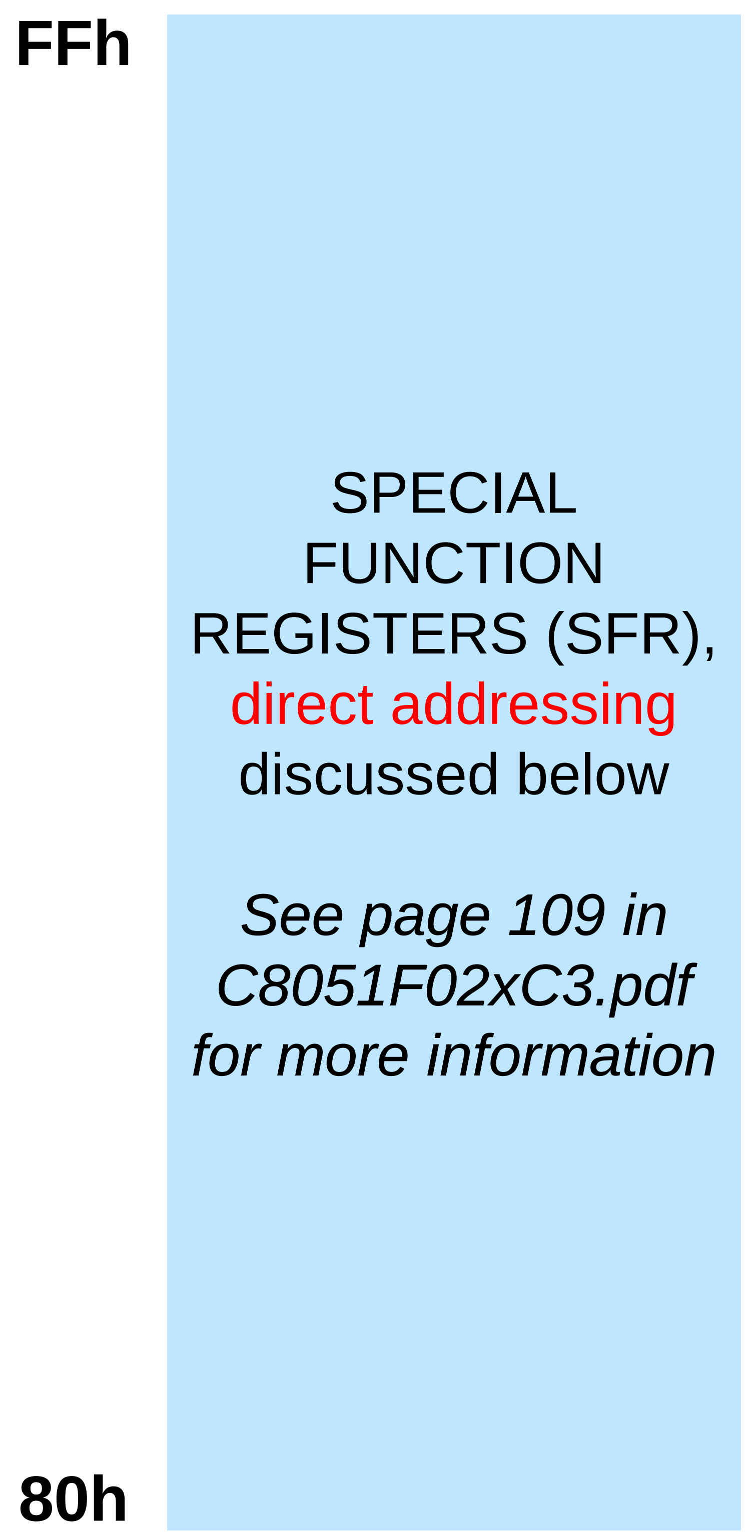
Given a 40 MHz Crystal, find the time required for one machine cycle.

40 MHz / 12 = 3.33 MHz
1/3.33 MHz = 0.30 µs

The oscillator generates clock pulses which are converted to machine cycles (1/12 clock pulses).
The CPU (etc.) is sequenced by these machine cycles.

# SPECIAL FUNCTION REGISTERS

- Microcontrollers have various modes, settings, and functionality that can be enabled.
  - The ‘toggle switches’ that hold these states and values are, in the case of the 8051, stored in the Special Function Registers (SFR, 0x80 to 0xFF)
  - The SFR also provides access to special-purpose registers such as I/O ports.
    - Some of the 8051’s SFR are bit addressable.
- Understanding a microcontroller’s special function registers is *greatly* aided by reading the datasheet.
  - Do this in conjunction with consulting pages such as http://www.keil.com/support/man/docs/c51/c51_le_sfrs.htm
  - SFR’s vary from device to device. Check the specific device’s data sheet!
- We will use the special function registers extensively when setting up timers, counters, and interrupts.



SPECIAL FUNCTION REGISTERS (SFR), direct addressing discussed below

*See page 109 in C8051F02xC3.pdf for more information*

# MICROCONTROLLER TIMERS

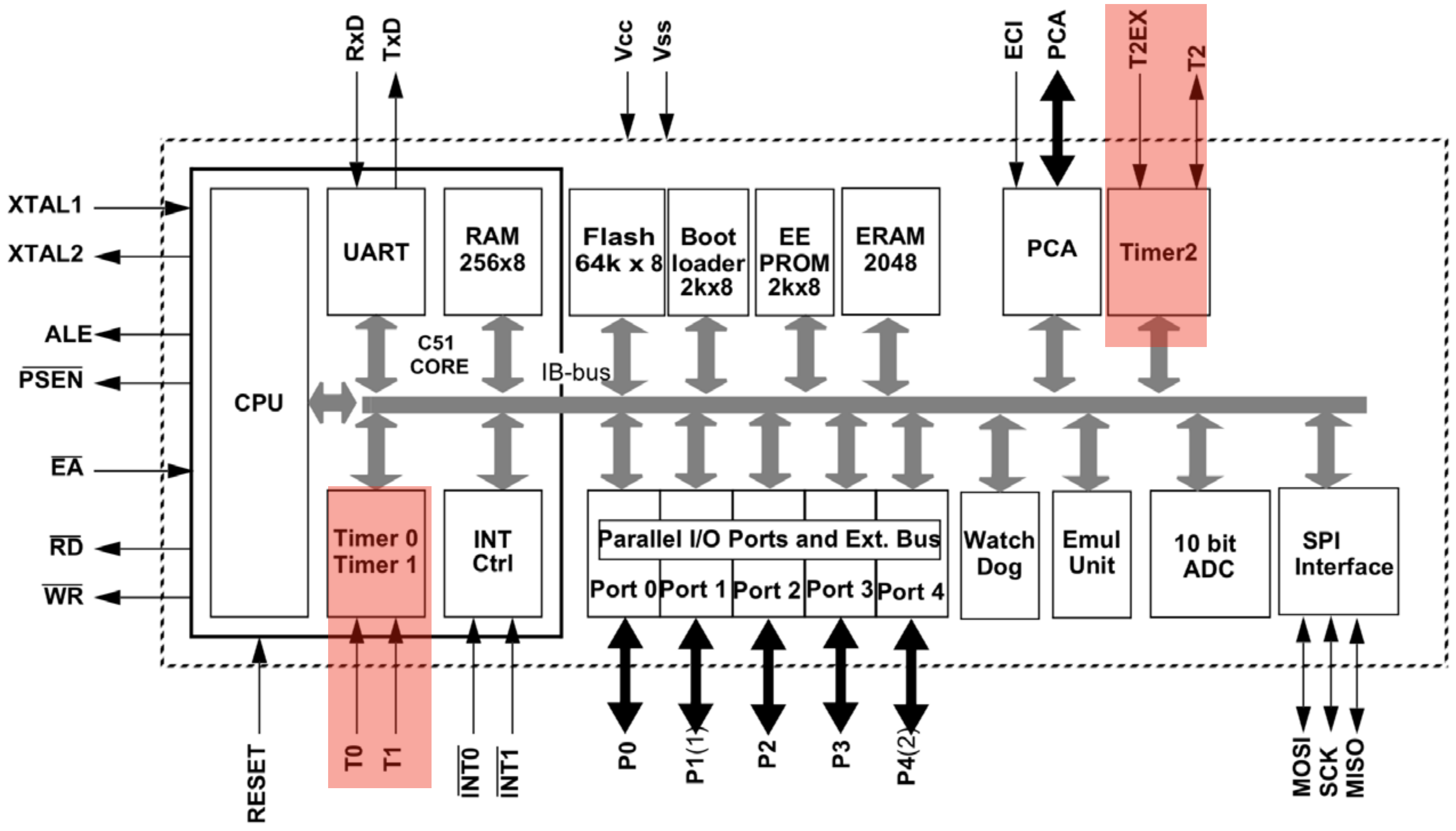
- When working with microcontrollers, we often wish for events to occur at a known rate.
  - Communications clocks (e.g., SPI)…
  - Waveform generation…
  - Periodic execution of code…
- While we can generate precise delays using the main CPU (lots of NOP’s, DJNZ, etc.), this is ‘blocking’ code, preventing the CPU from easily doing other tasks while these timed events occur.
  - Such blocking code is generally considered sloppy and best avoided.
    - Instead, the microcontroller’s hardware timers should be used, freeing up the CPU.

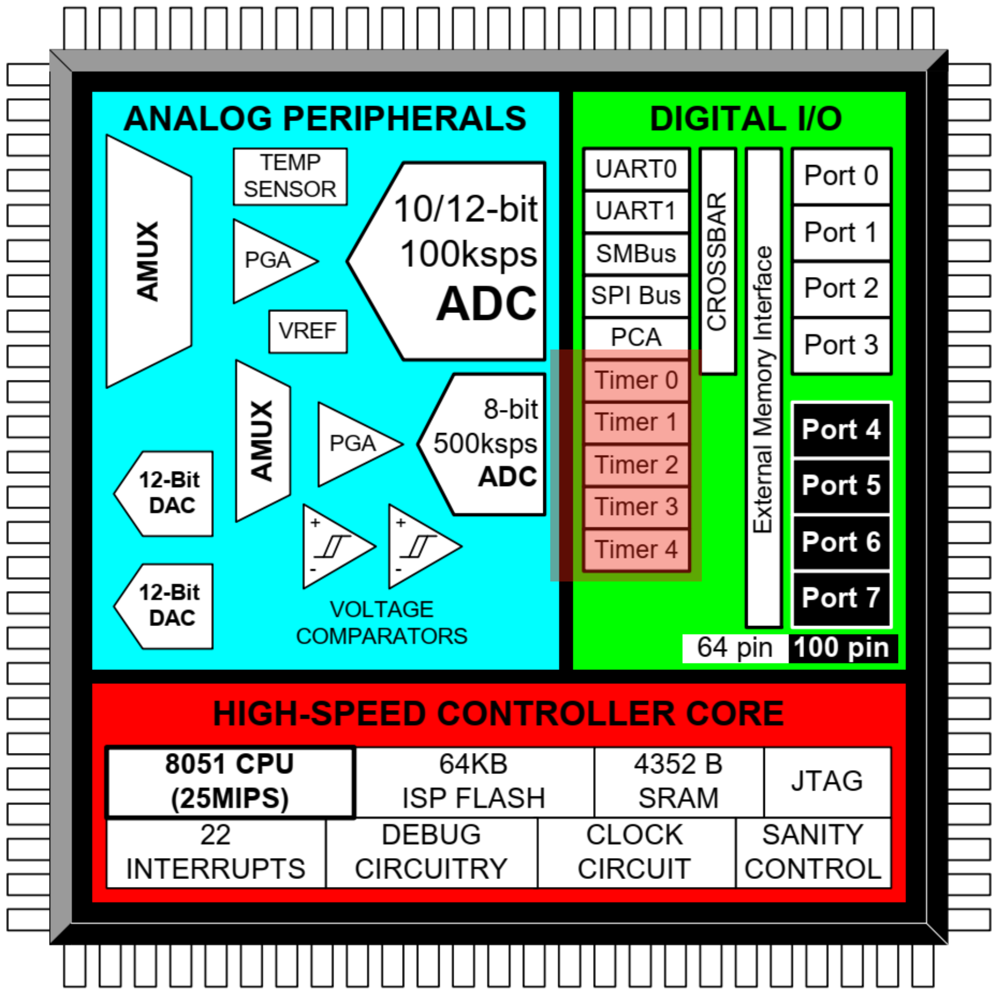
# MICROCONTROLLER TIMERS, CONTINUED

- The timer/counter is a separate component of the microcontroller’s hardware.
  - Timers are typically configured using configuration registers.
    - These registers allow the timer’s rate, counting behaviour, and overflow behaviour to be specified (amongst other things!)
  - Timers work by counting up/down at a known rate.
    - Given a known clock rate and a specific value to count to, very flexible timing options are made available.
    - Hardware timers on entry-level microcontrollers are usually either 8-bit, 16-bit, or 32-bit.
      - More bits == more precision timing options and lower frequencies.
      - Very basic microcontrollers might have no hardware timers. Modern advanced ones (e.g., ARM Cortex F7) might have 10+ timers with varying capabilities.
        - Timers (such as those on the 8051) may often also be used as counters, counting the numbers of events that occur.

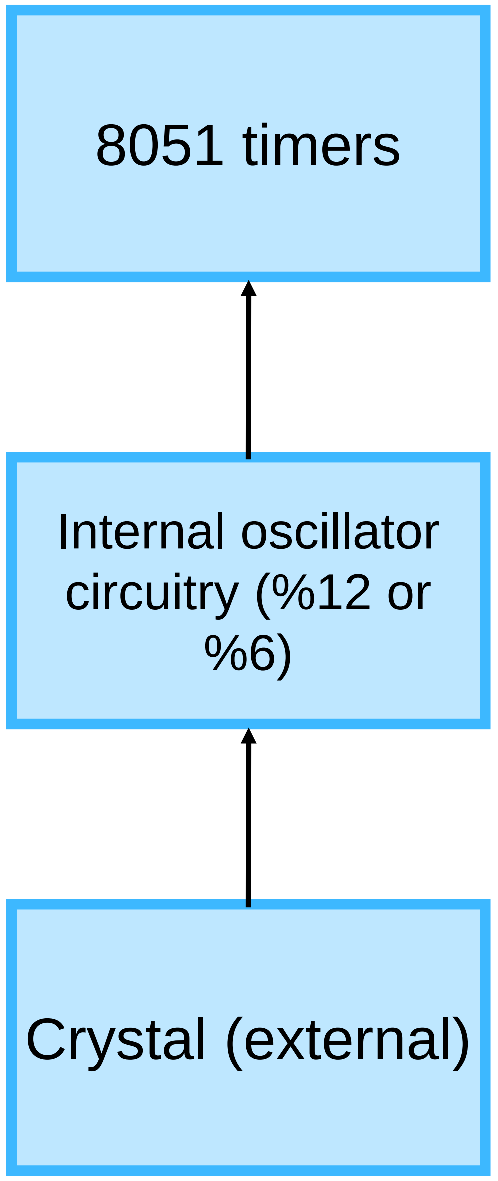
# TIMERS ON THE 8051

- The standard 8051 CPU has two timers (Timers 0 and 1).
- Enhanced binary-compatible ones (like the AT89C51AC3) might have more.
- The AT89C51AC3 features a third timer (Timer2); we'll focus on Timer 0 and Timer 1\
- The C8051F020 features five timers in total



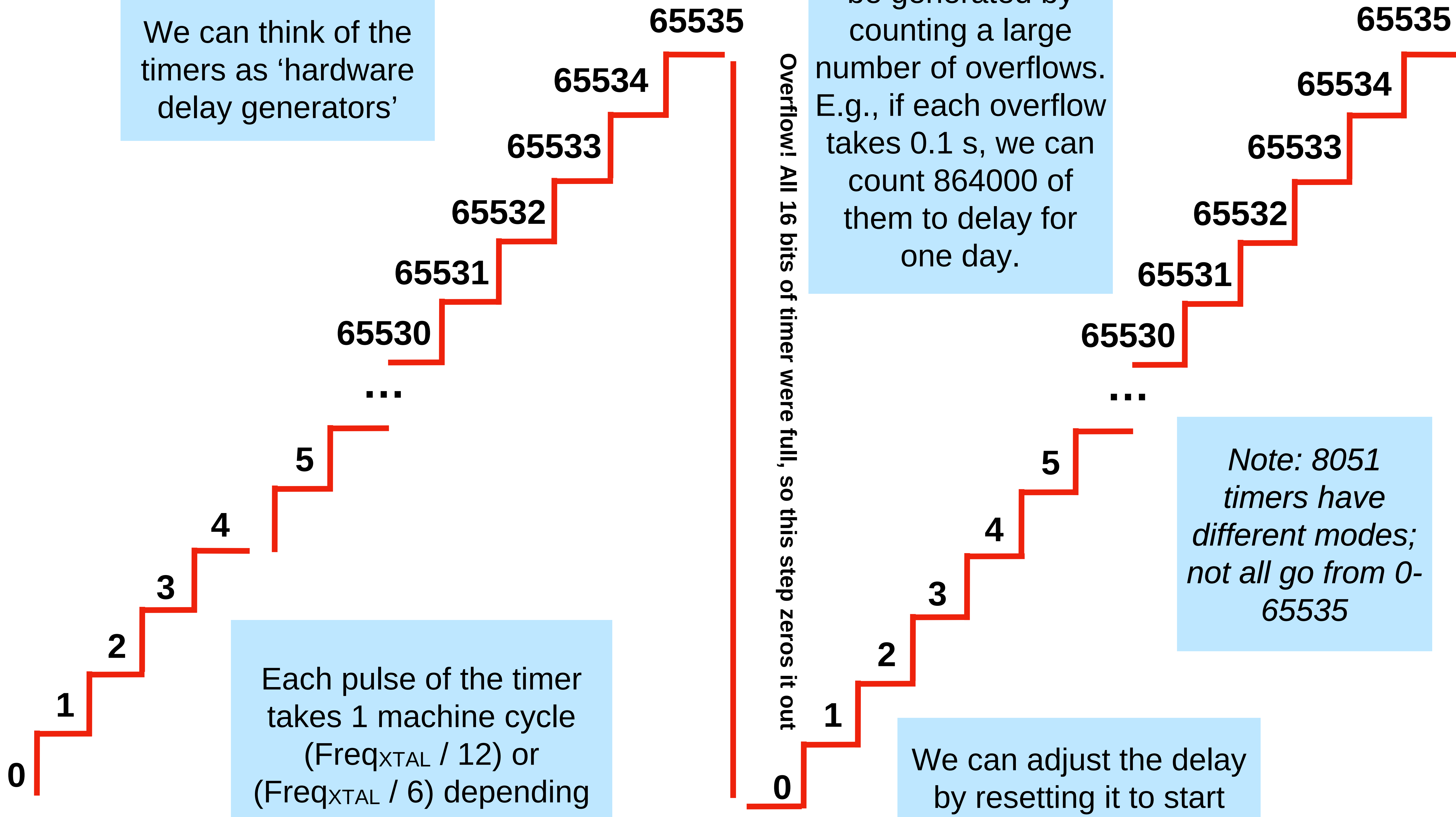


# 8051 TIMER SPECIFICATIONS



- Timer 0 and Timer 1:
  - 16 bits: can count from 0d0 to 0d65535
  - Since the 8051 is an 8-bit microcontroller, the 16 bit timers hold their values in two adjacent 8-bit registers.
    - TH0 and TL0 (Timer 0 High and Timer 0 Low)
    - TH1 and TL1 (Timer 1 High and Timer 1 Low)
  - To specify the timers' modes and control the timers, two special function registers are used.
    - TMOD register: sets various timer modes.
    - TCON register: Timer Control register, contains settable flags that show the timers' statuses.
- When the timer goes from the highest count (e.g., 65535) to 0, it can call an interrupt which results in special-purpose code being executed.
  - Known as a 'timer interrupt.'
  - We'll talk more about interrupts later.

We can think of the timers as 'hardware delay generators'



Each pulse of the timer takes 1 machine cycle ($Freq_{XTAL}$ / 12) or ($Freq_{XTAL}$ / 6) depending on the 8051

Longer delays can be generated by counting a large number of overflows. E.g., if each overflow takes 0.1 s, we can count 864000 of them to delay for one day.

*Note: 8051 timers have different modes; not all go from 0-65535*

We can adjust the delay by resetting it to start from a value other than 0 after it has overflowed…

# TCON REGISTER

- The TCON (Timer Control) register lets us:
  - Know when the timers have overflowed (TCON.7 and TCON.5)
  - Start and stop the timers (TCON.6 and TCON.4)
  - Specify external interrupt settings (TCON.3 - TCON.0)
    - Interrupt settings will be discussed more in a future slide.

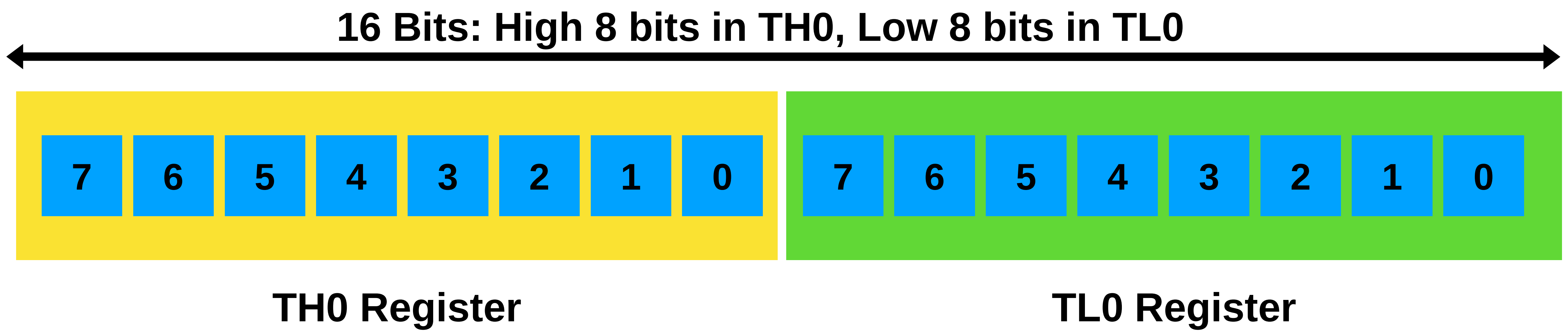
| TCON.7 | TCON.6 | TCON.5 | TCON.4 | TCON.3 | TCON.2 | TCON.1 | TCON.0 |
|---|---|---|---|---|---|---|---|
| TF1 | TR1 | TF0 | TR0 | IE1 | IT1 | IE0 | IT0 |
| Timer 1 Overflow Flag 1 when overflow occurs. Must be cleared in software; auto. cleared when leaving ISR | Timer 1 run bit 1: Start timer 0: 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. |

# TMOD REGISTER

- The TMOD (Timer Mode) special function register lets us:
  - Set modes for Timer0 and Timer1
  - Specify whether the timer always runs or only runs when some outside condition it met.
  - Specify whether the timer serves as a delay generator (timer) or as an event counter.

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

# THx & TLx REGISTERS

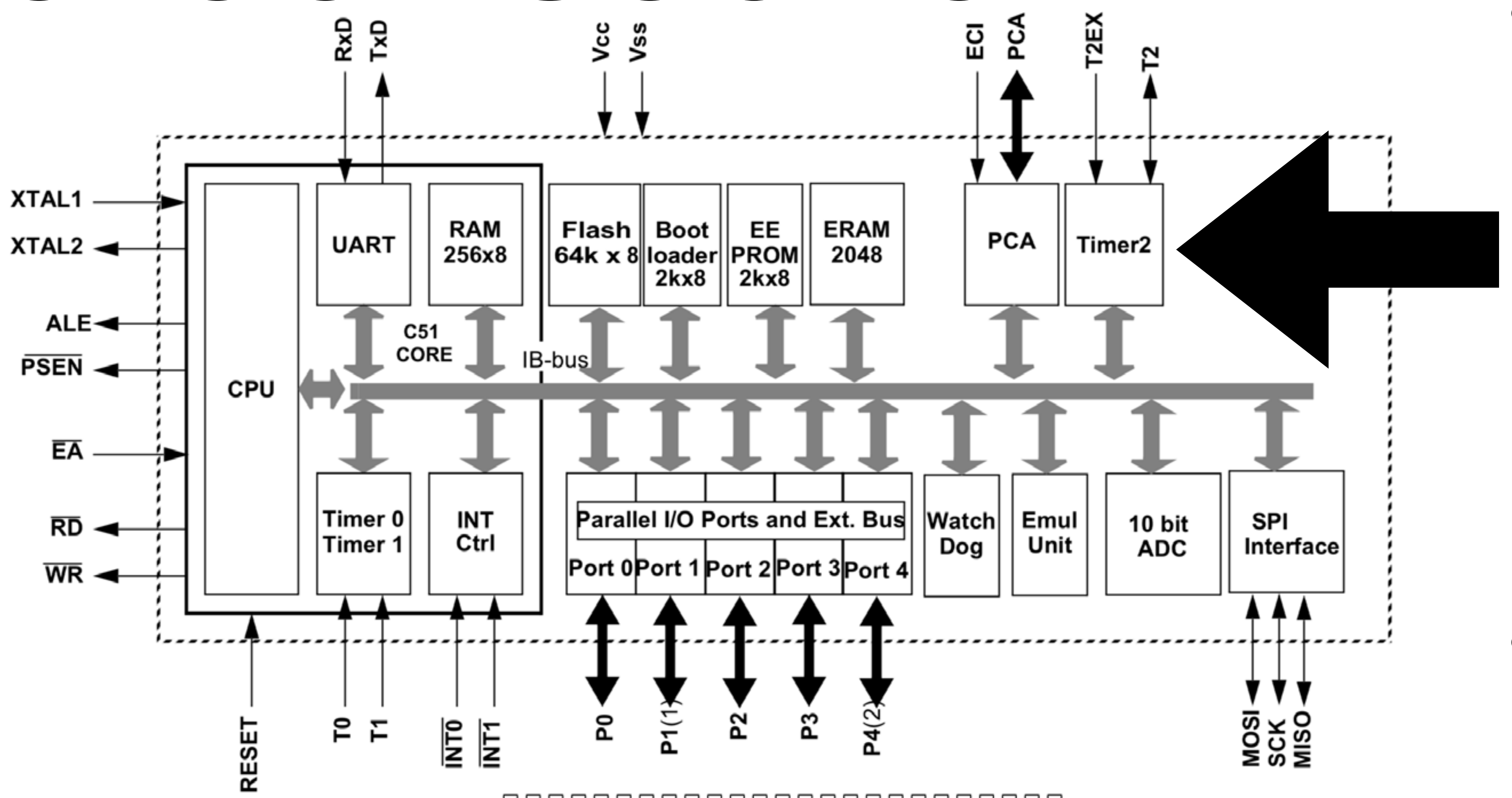


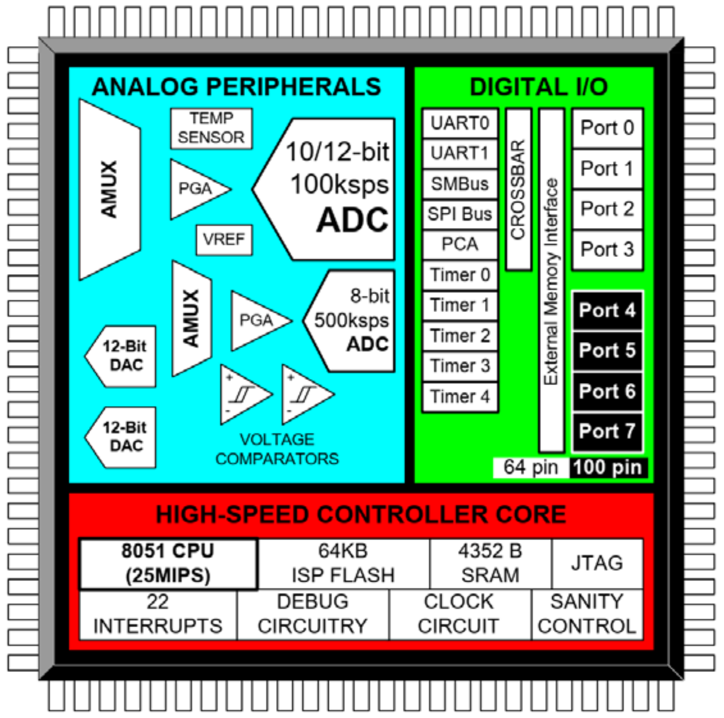
- Every time the timer iterates, the count of the timer register (really 2 ‘joined’ registers) increases by 1.
  - Once all bits are filled with 1’s (2^16), the next machine cycle will zero everything out (overflow).
    - This overflow will set a flag in TCON: TF0 or TF1
      - The TH0 and TL0 registers may then be re-set with a value between 0-2^16, and the count will begin again from this point.

# TIMER OPERATING MODES

| | **Bit values in TMOD M1, M0** | **Mode name** | **Notes** |
|---|---|---|---|
| MODE 0 | 0, 0 | 13-bit timer mode: 8 bits of THx and 5 bits of TLx | Don’t use this for new projects! (Allows for compatibility with old systems) |
| MODE 1 | 0, 1 | 16-bit timer mode. TLx counts 0-255; on overflow, this adds 1 to THx | A very common mode for generating delays and counting events. |
| MODE 2 | 1, 0 | 8-bit timer mode. TLx auto-reloads with THx value. | In non auto-reload modes, software must reload value after overflow; this does it automatically. |
| MODE 3 | 1, 1 | “Split timer” mode: THx is one 8-bit timer, and TLx is another. | When Timer0 is in split timer mode, TH0 becomes Timer1 and TL0 becomes Timer2. Meanwhile, the ‘real’ Timer1 just counts away. Useful if you need two ‘smart’ delays and one ‘dumb’ one. |

# TIMER2 AS A PROGRAMMABLE CLOCK SOURCE





- Many binary-compatible 8051 clone have additional timers. The AT89C51AC3's Timer2 behaves similarly to the other timers.
  - Timer2 is a 16 bit timer configured by the T2MOD register.
    - The T2CON register is equivalent to the TCON register.
- Timer 2 can be configured to work as a 50% duty cycle square wave clock pulse generator.
  - See page 238 in C8051F02x.pdf for more information.
- Timer 2 can also be set as an auto-reload 16 bit timer (meaning that it auto-restarts after overflow, no software intervention needed).

## MONITORING TIMER OVERFLOW

- Timers iterate at one tick per machine cycle.
  - When the timer overflows, we know that a certain amount of time has elapsed since the timer was started.
    - In order to know exactly when the timer has overflowed, we can do two things:
      - "Polling" approach: in the main program, regularly and rapidly check whether the TFx flag has gone from 0 to 1.
        - Advantages: Easy to program.
        - Disadvantages: The eats up CPU resources, requiring us to constantly inspect a memory location instead of using those CPU cycle for other things.
          - ALSO: if we're doing something else (e.g., displaying text on an LCD), we might miss the exact moment of the overflow.
      - "Interrupt-driven" approach: the 8051's program counter is vectored to the start address of interrupt service routine (ISR) when an overflow occurs.
        - We'll implement some examples of this in a future slide.
        - Advantage: CPU can do other things while the timer counts up, only needing to service the timer when an overflow occurs.
          - Much, much more efficient! Less deterministic.

# EXAMPLE: SQUARE WAVE GENERATOR, POLLING

```
;Generate a 50% duty cycle square wave on a 12-cycle 8051.
;We’ll count from 2^15 to 2^16, a total of 32768 values.
;XTAL = 12 MHz, machine cycle = 1 MHz, delay time = 0.033 s
;Output this square wave to P1.0
;Set up our timer: we’ll use Timer 0, Mode 1 (16 bit timer)
MOV TMOD,#01          ;TMOD reg: 0 0 0 0 0 0 0 1

;Now we load 0d32768 (0x8000) into the TL0 and TH0 registers
TIMR:
MOV TL0,#00H ;00000000 (Load up the timer registers with values to count)
MOV TH0,#80H ;10000000
CPL P1.0     ;Compliment P1.0 (flips bits - inverts it; if 0, then 1)
ACALL START
SJMP TIMR

START:
SETB TR0      ;Start timer0

POLLER: JNB TF0, POLLER ;Jump if bit not set. Waits for overflow
CLR TR0       ;Stop timer 0
CLR TF0       ;Clear the overflow flag
RET           ;Jump back to ACALL
```

https://what-when-how.com/8051-microcontroller/programming-8051-timers/