XMUT202 Digital Electronics

Timers and Counters

Week 11 Lecture 2

Agatha Rachmat

School of Engineering and Computer Science

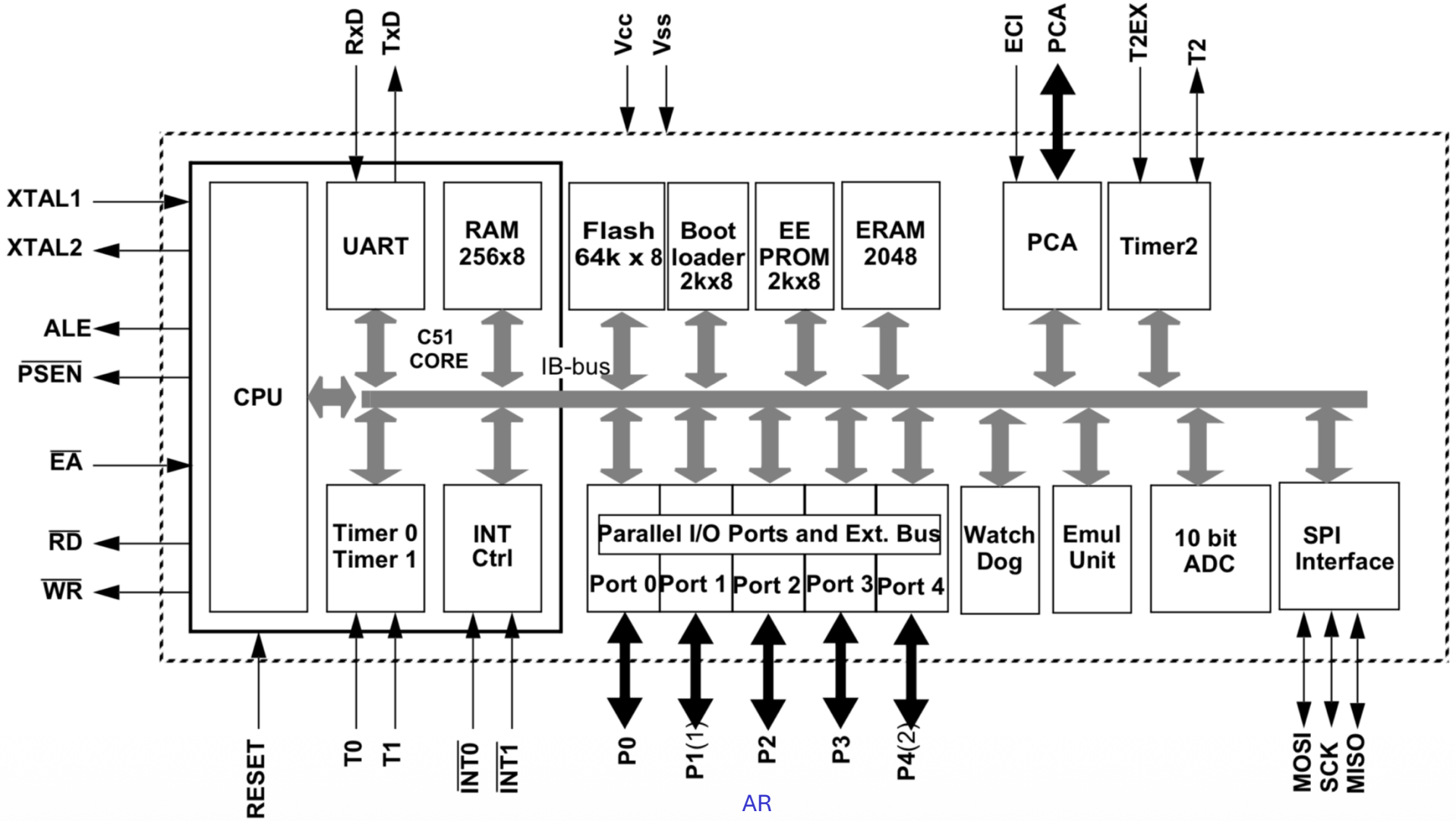
Victoria University of Wellington

Today's topic

- Intro to timers.
- Timers in 8051.
- Timer's registers.
- Programmable clock source.
- Timer as a counter.
- Watchdog timer.

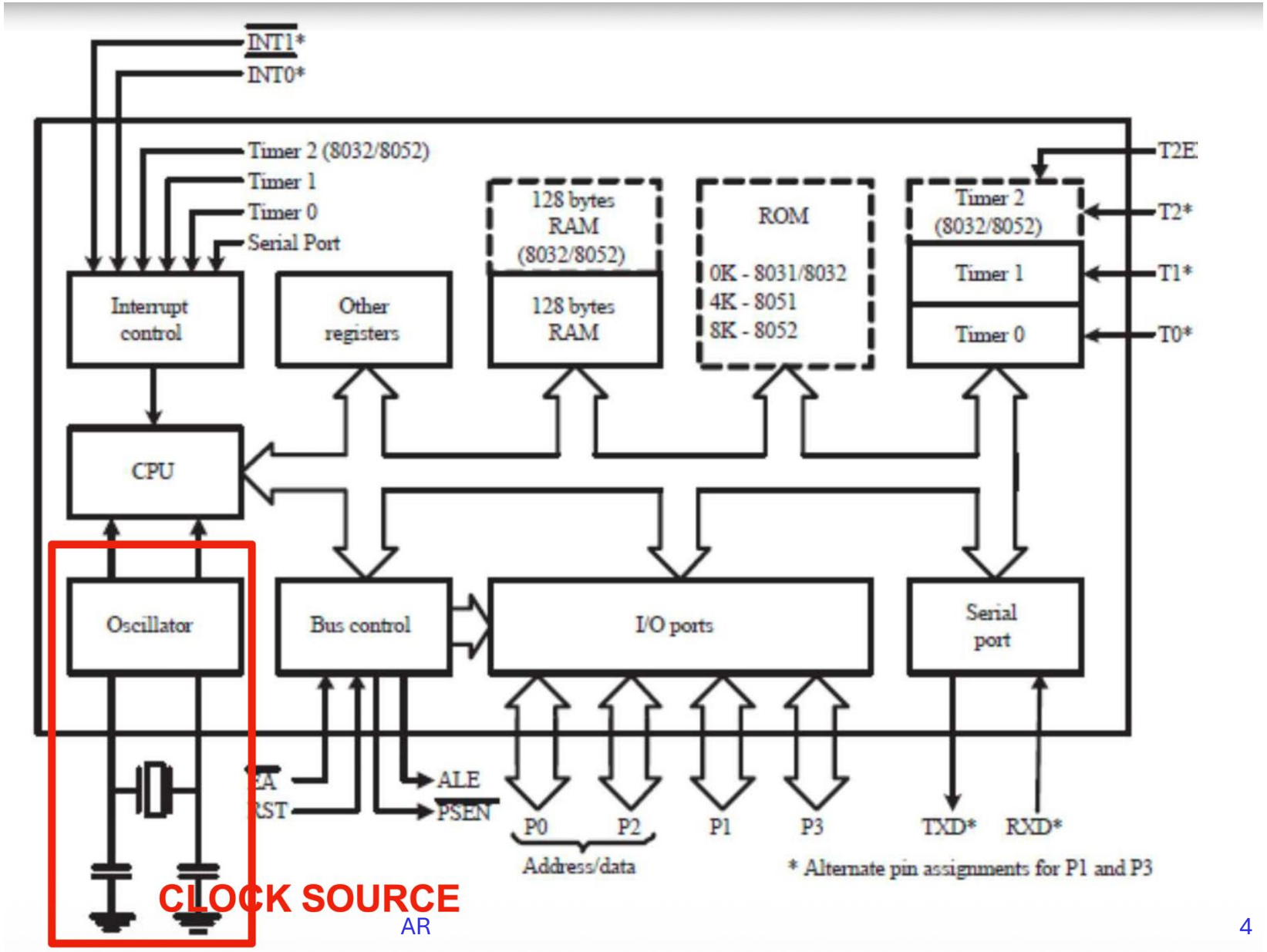
Timers in Block Diagram of 8051

- Timers in 8051: Timer 0 and Timer 1 (standard) and Timer 2.



Timers in Block Diagram of 8051

- Clock in Timers in 8051 is derived from XTAL oscillator.

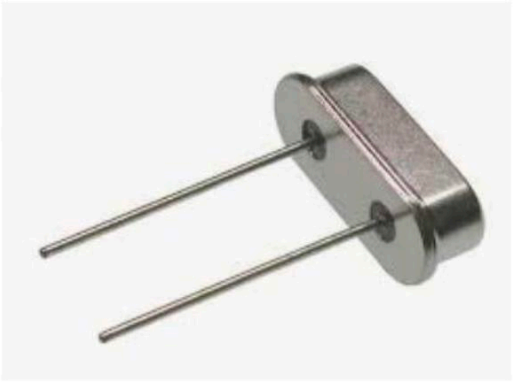


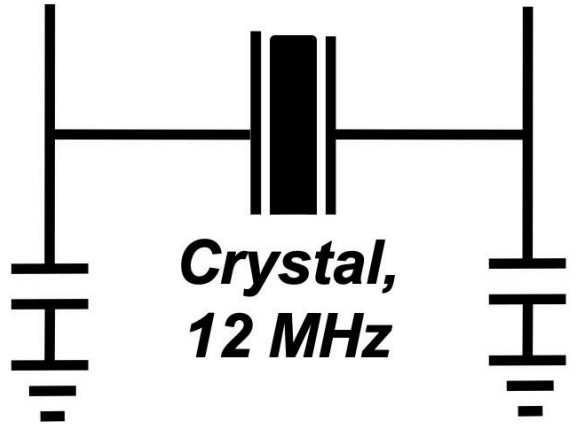
Recap: Clock Source

- 8051 programs are executed sequentially, with each instruction executing after the previous one.
- A clock source is needed to increment a program counter (discussed later).
- Other functions within the microcontroller (i.e. 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.

Recap: Clock Source

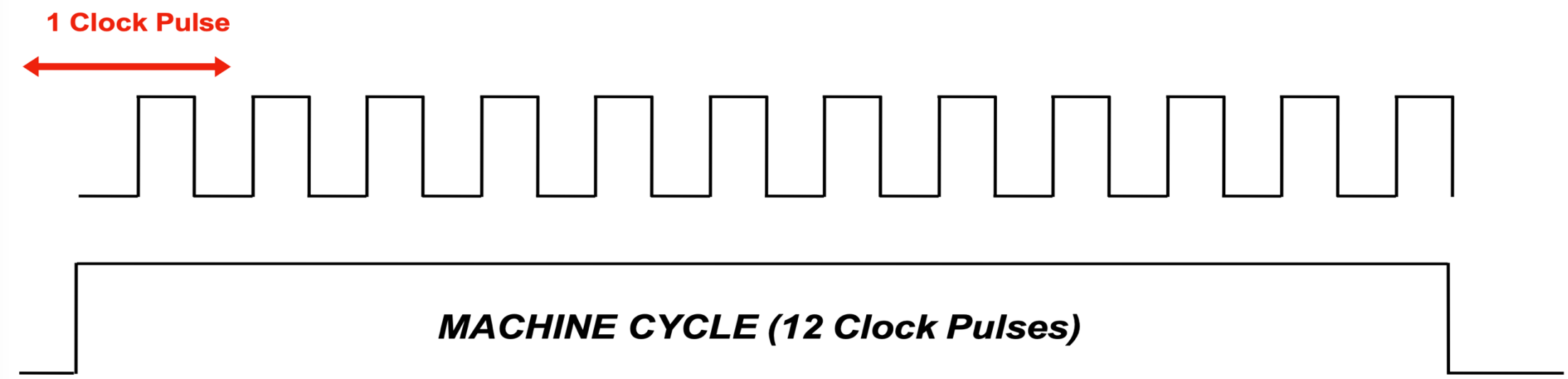
- Many modern processors can execute one instruction per clock cycle. (or more!)
- The 8051 requires **12 clock** cycles per "machine cycle".
- Instructions on the 8051 require 1 or 2 machine cycles.
- Given a 12 MHz clock, this means that we can execute 0.5-1 million instructions per second (MIPS).





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

Recap: Machine Cycle



- Given a 40 MHz crystal oscillator, 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.

Example 1 – Timer Delay Example

See the previous examples of implementation of a timer program in 8051 using the iterative method:

- SimpleTimer.asm.
- MultiTimer.asm.

Example 1 – Timer Delay Example

```
START:
MOV A,#0FFH          ; Move 0xFF(1)to accumulator
MOV P1,A             ; Move accumulator value to P1

;TODO: delay for 1 ms!
MOV A,#00H           ; Move 0x0(0) to accumulator
MOV P1, A            ; Move accumulator value to P1

;TODO: delay for 1 ms again
SJMP START           ; Jump back to 'START'
```

SimpleDelay.asm.

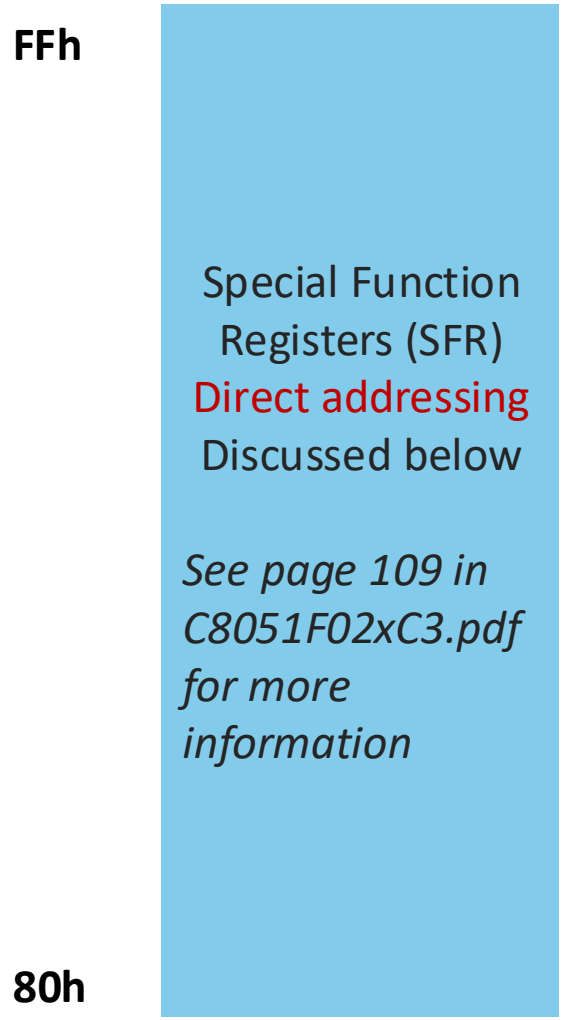
Example 1 – Timer Delay Example

```
START:
MOV A,#0FFH              ;Move 0xFF(1) to accumulator
MOV P1,A                 ;Move accumulator value to P1
ACALL DELAY              ;Calls subroutine at 'delay'
MOV A,#00H               ;Move 0x0(0) to accumulator
MOV P1,A                 ;Move accumulator value to P1
ACALL DELAY              ;Delay for another 1 ms
SJMP START               ;Jump back to 'START'

DELAY:          ;1 ms delay Subroutine
MOV R6,#250D             ;Place 0d250 into Register 6
MOV R7,#250D             ;Place 0d250 into Register 7

DEL1:
DJNZ R6,DEL1             ;DJNZ: Decrement R6 & jump if not 0

DEL2:
DJNZ R7,DEL2             ;DJNZ is 2-cycles, 2 μS to run. 2X500us=1ms
RET                      ;Return to ACALL
```

MultiDelay.asm.

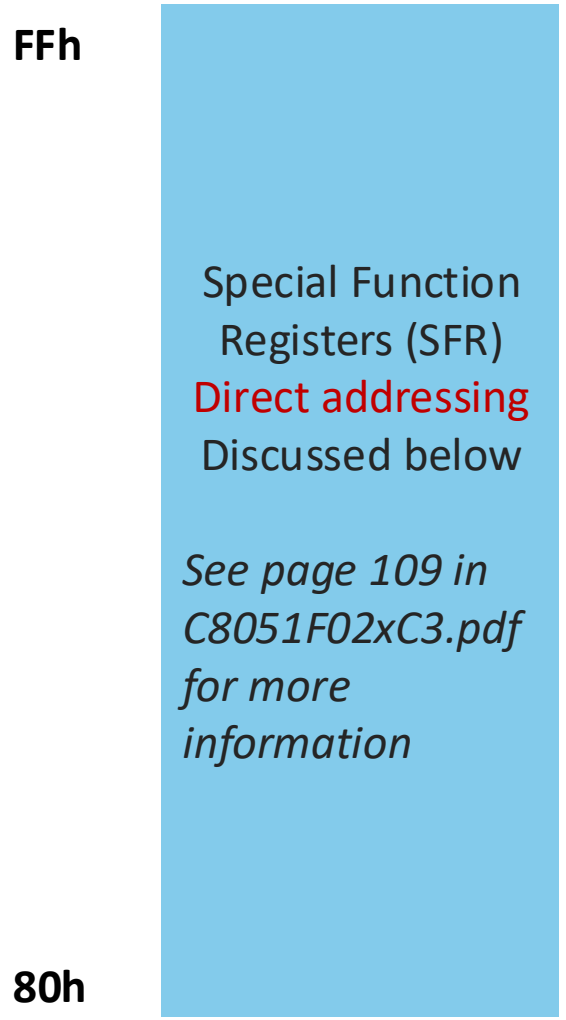
Special Function Registers with Timers

- 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.



Special Function Registers with Timers

- 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 (SFR) extensively when setting up timers, counters, and interrupts.



Microcontroller Timers Uses

- 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 (i.e. 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 Uses

- The timer or 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)

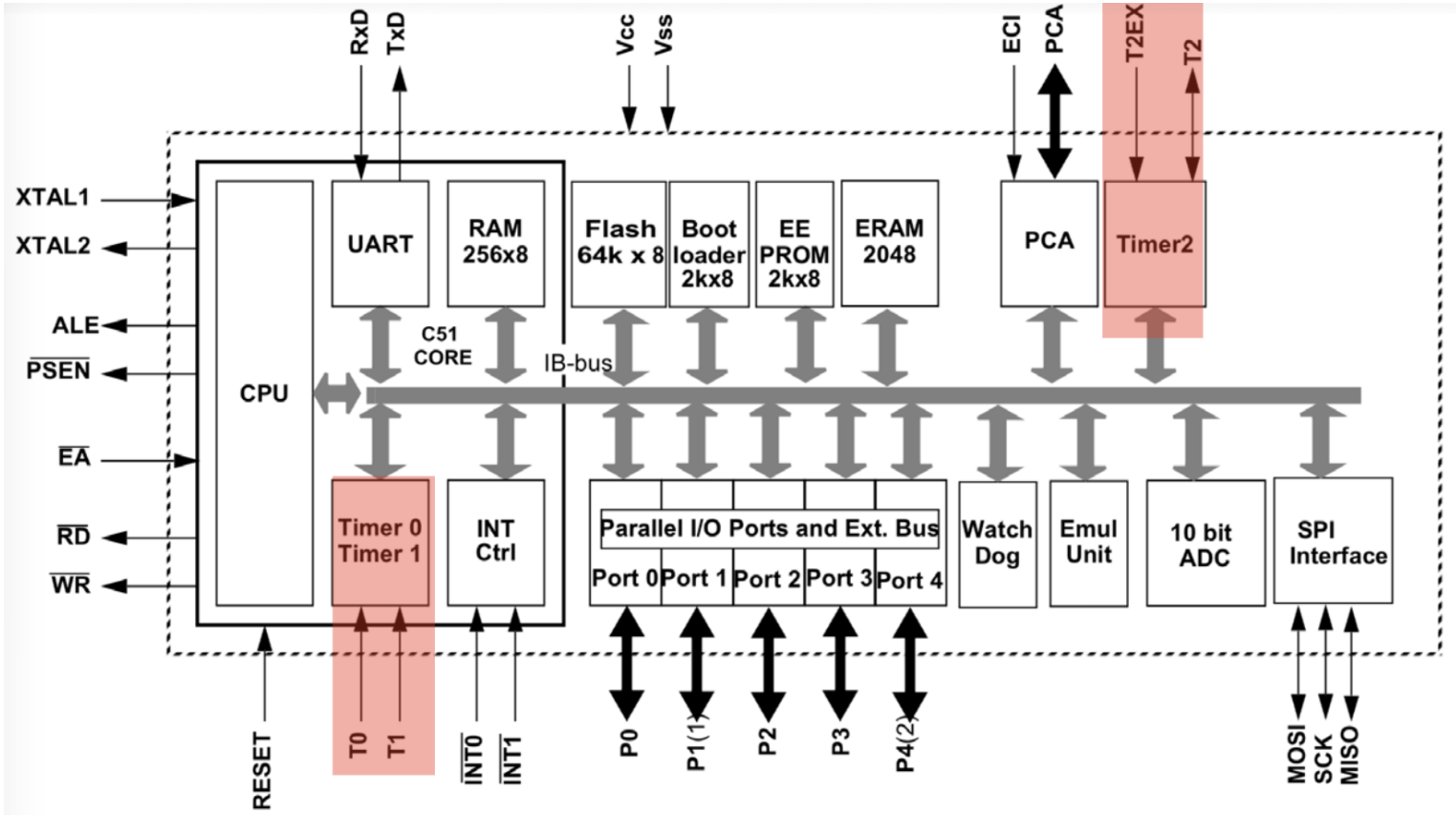
8D	Not bit-addressable								TH1
8C	Not bit-addressable								TH0
8B	Not bit-addressable								TL1
8A	Not bit-addressable								TL0
89	Not bit-addressable								TMOD
88	8F	8E	8D	8C	8B	8A	89	88	TCON
87	Not bit-addressable								PCON

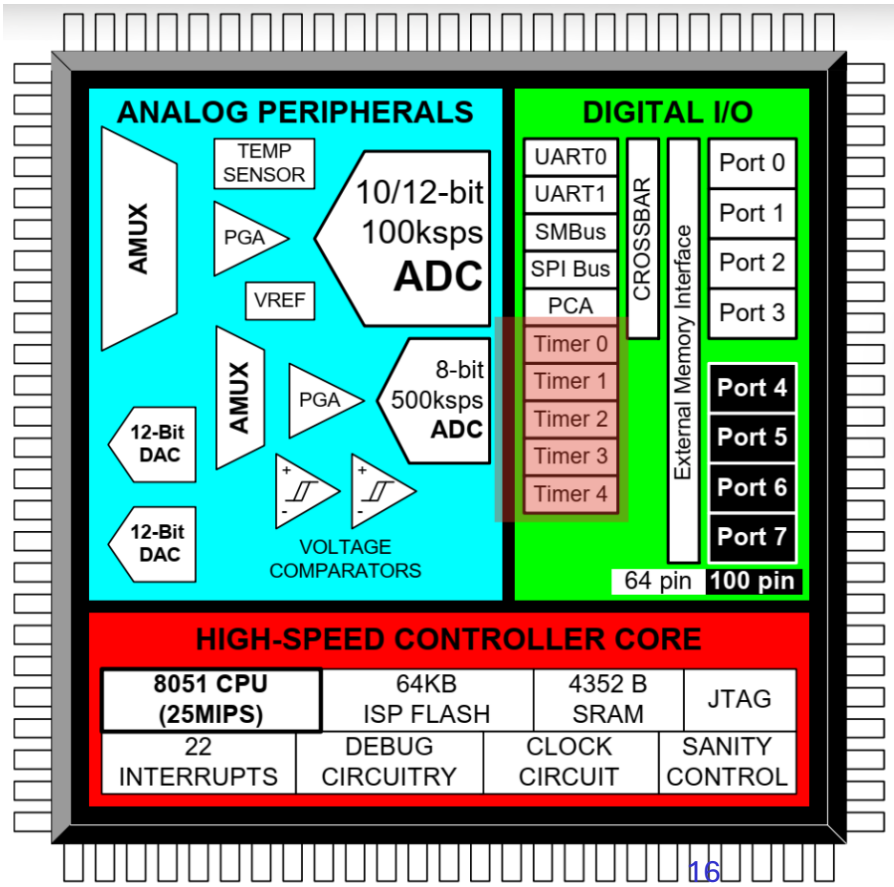
Microcontroller Timers Uses

- Timers work by counting up or 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 number 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 (Timer 2); 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.

A 16-bit register means the timer uses:

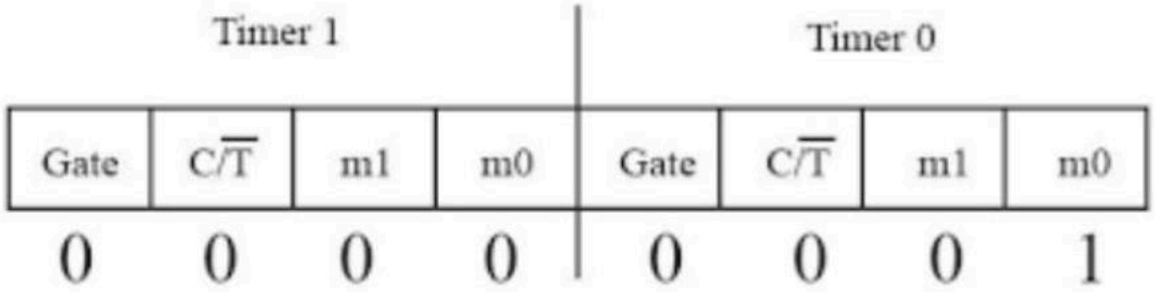
$$2^{16} = 65{,}536 \ possible\ values$$

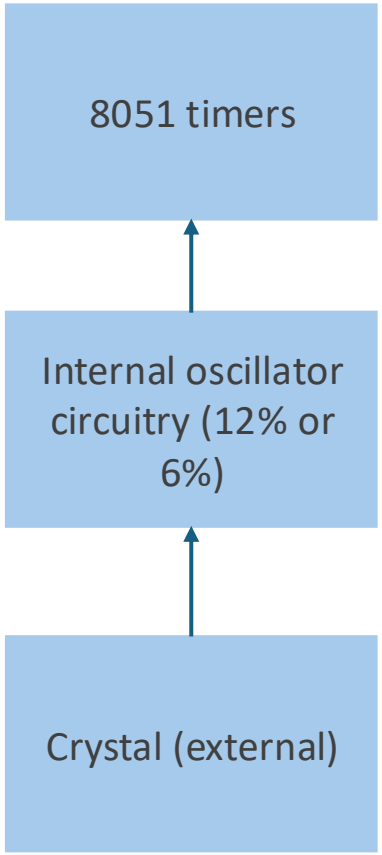
So the count range is:

•Minimum: **0000H = 0 (decimal 0)**

•Maximum: **FFFFH = 65535 (decimal 65535)**

That's why we say it counts from **0 to 65535**.

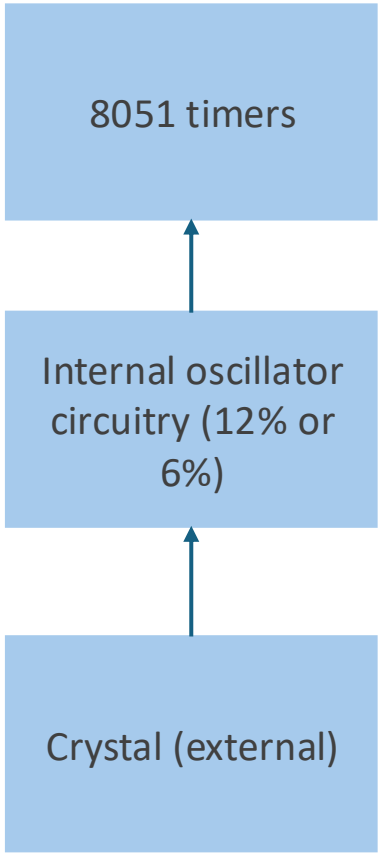




8051 Timer Specifications

Timer 0 and Timer 1:

- Since the 8051 is an 8-bit microcontroller, the 16-bit timers hold their values in two adjacent 8-bit registers.



The timer uses **two 8-bit registers**:
TH0 and TL0 (Timer 0 High and Timer 0 Low).
TH1 and TL1 (Timer 1 High and Timer 1 Low).
•THx = High byte
•TLx = Low byte

Together:

$$\text{Timer Value} = THx{:}TLx$$

Example:

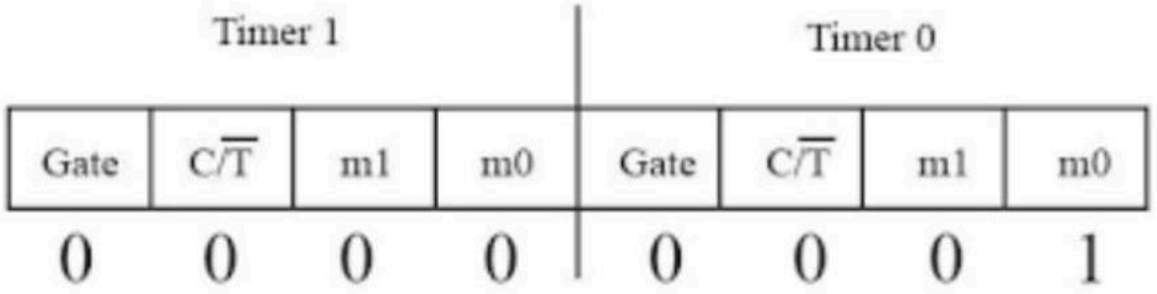
```
TH0 = 12H
TL0 = 34H
→ Timer value = 1234H
```

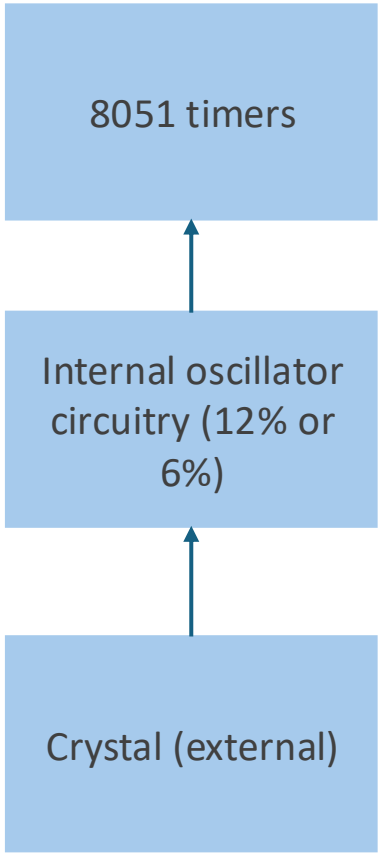
8051 Timer Specifications

Timer 0 and Timer 1:

- The timer increments by **1 every machine cycle** (if configured as timer mode)
- It counts like a binary counter:

`0000H → 0001H → 0002H → ... → FFFFH`

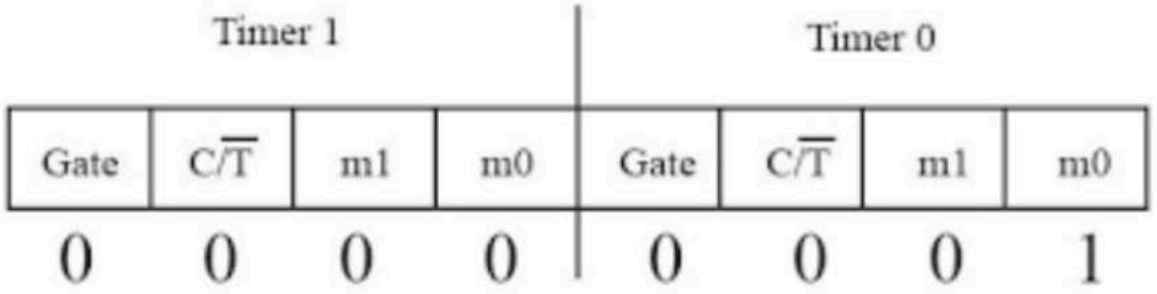


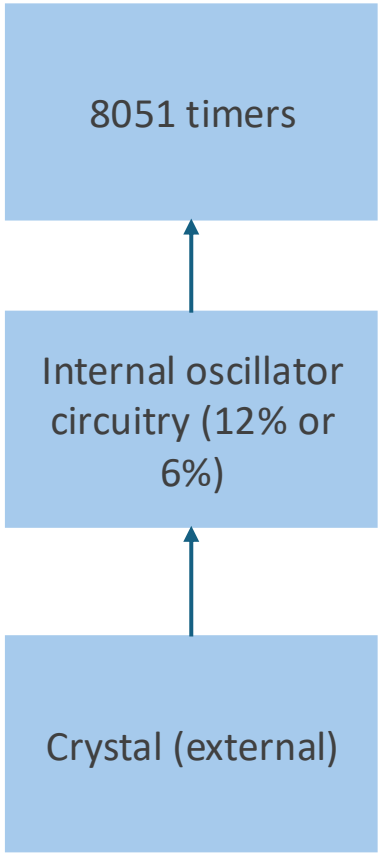


8051 Timer Specifications

Overflow : When the timer reaches its maximum:
FFFFH + 1 → 0000H (overflow)

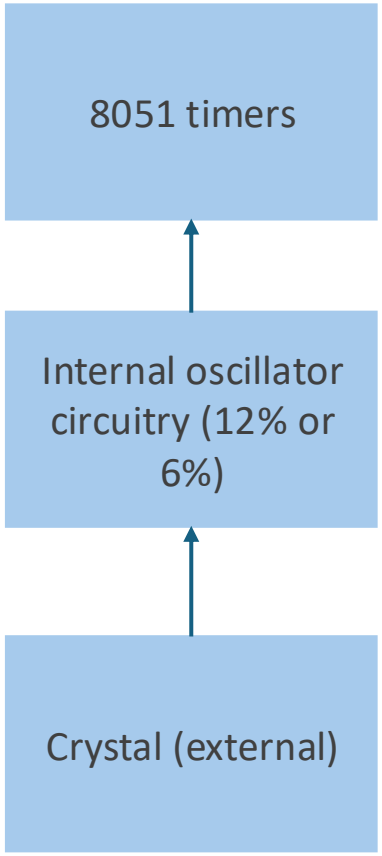
When overflow happens:
TF0 / TF1 flag is set



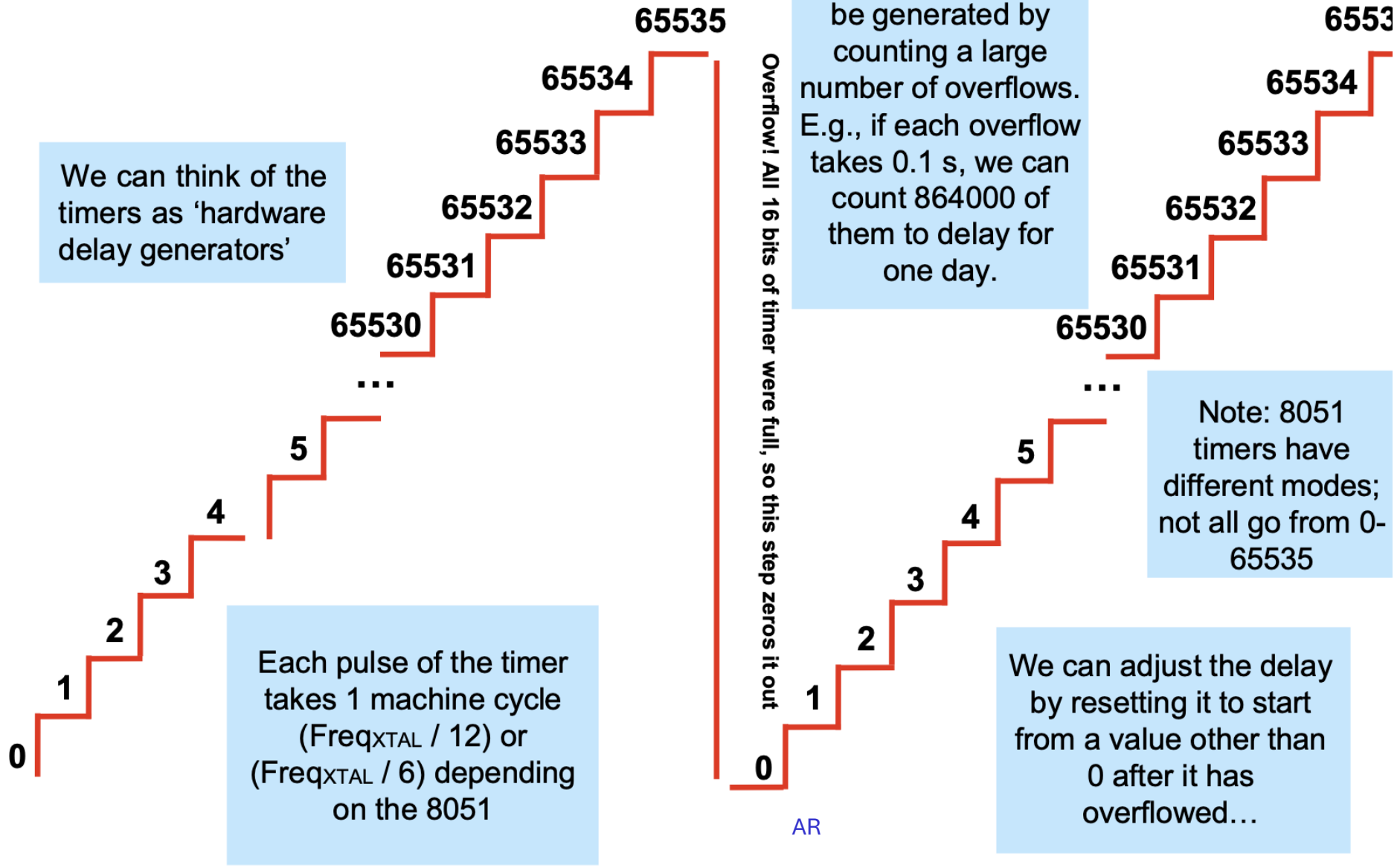


8051 Timer Specifications

- 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.



8051 Timer Specifications



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).

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

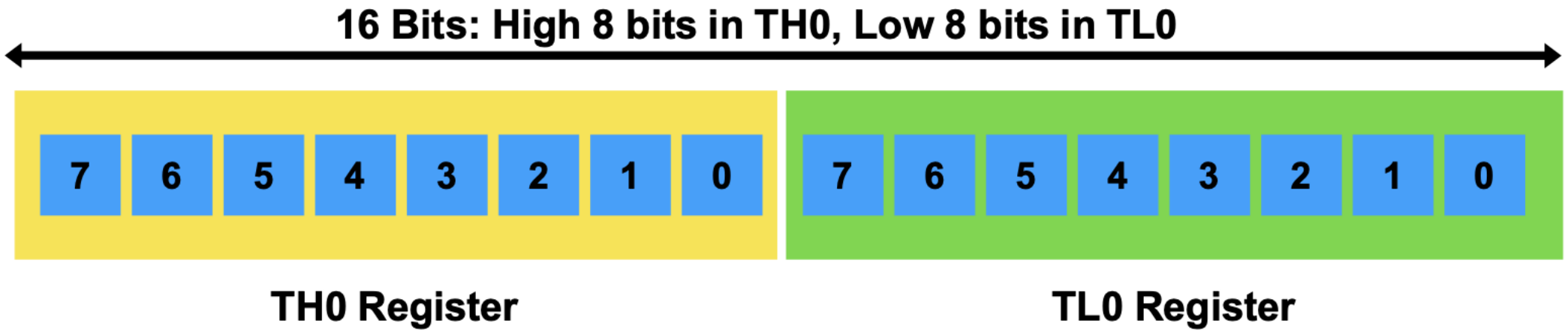
The TMOD (Timer Mode) special function register lets us:

- Set modes for Timer 0 and Timer 1.
- Specify whether the timer always runs or only runs when some outside condition is met.
- Specify whether the timer serves as a delay generator (timer) or as an event counter.

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

THx & TLx Registers

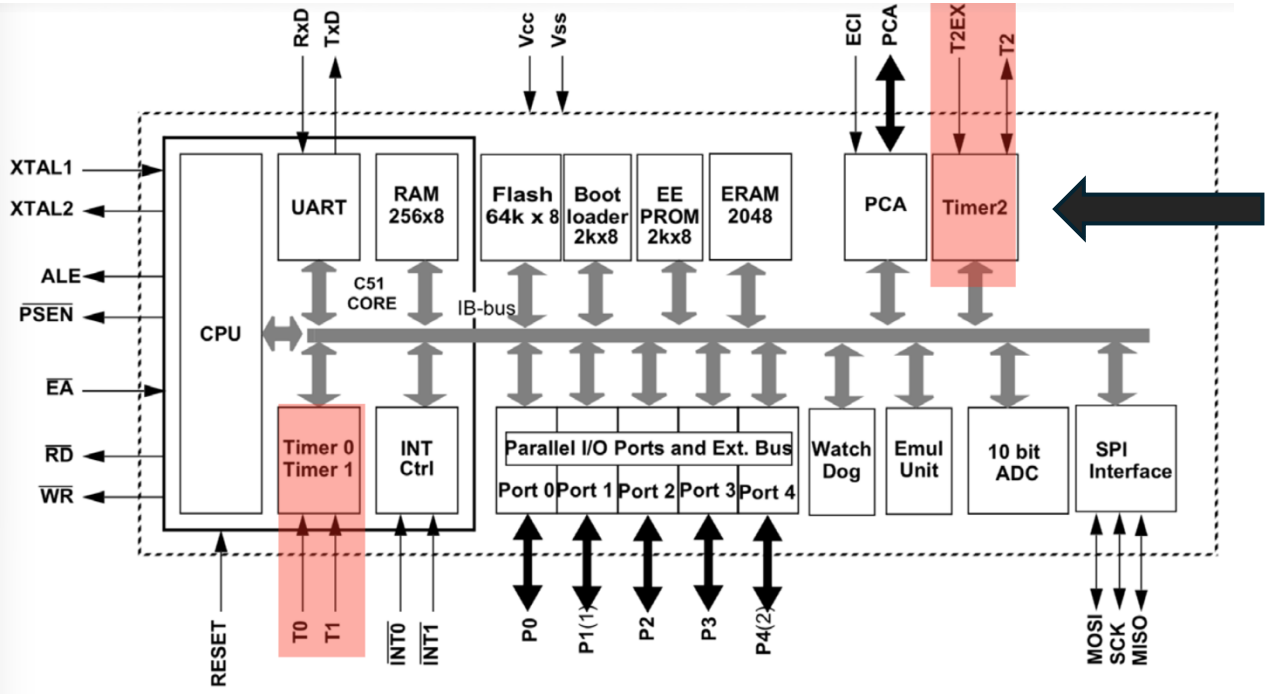
- Every time the timer iterates, the count of the timer register(really 2 ‘joined’ registers) increases by 1.
- Once all bits are filled with 1’s (2^16), the next machine cycle will zero everything out (overflow).
- This overflow will set a flag in TCON: TF0 or TF1.
- The TH0 and TL0 registers may then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

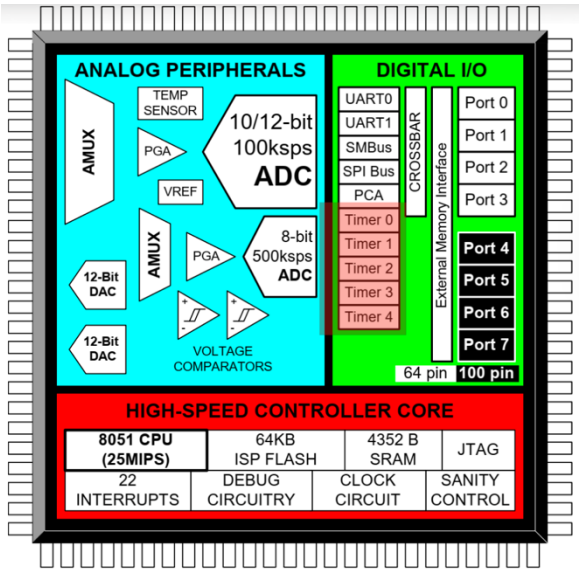


Timer Operating Modes

Mode	Bit values in TMD M1, M0	Mode name	Notes
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)
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.
2	1,0	8-bit timer mode. TLx auto-reloads with THx value.	In nonauto-reload modes, software must reload value after overflow; this does it automatically.
3	1,1	"Split timer" mode: THx is one 8-bit timer, and TLx is another.	When Timer 0 is in split timer mode, TH0 becomes Timer 1 and TL0 becomes Timer 2. Meanwhile, the 'real' Timer 1 just counts away. Useful if you need two 'smart' delays and one 'dumb' one.

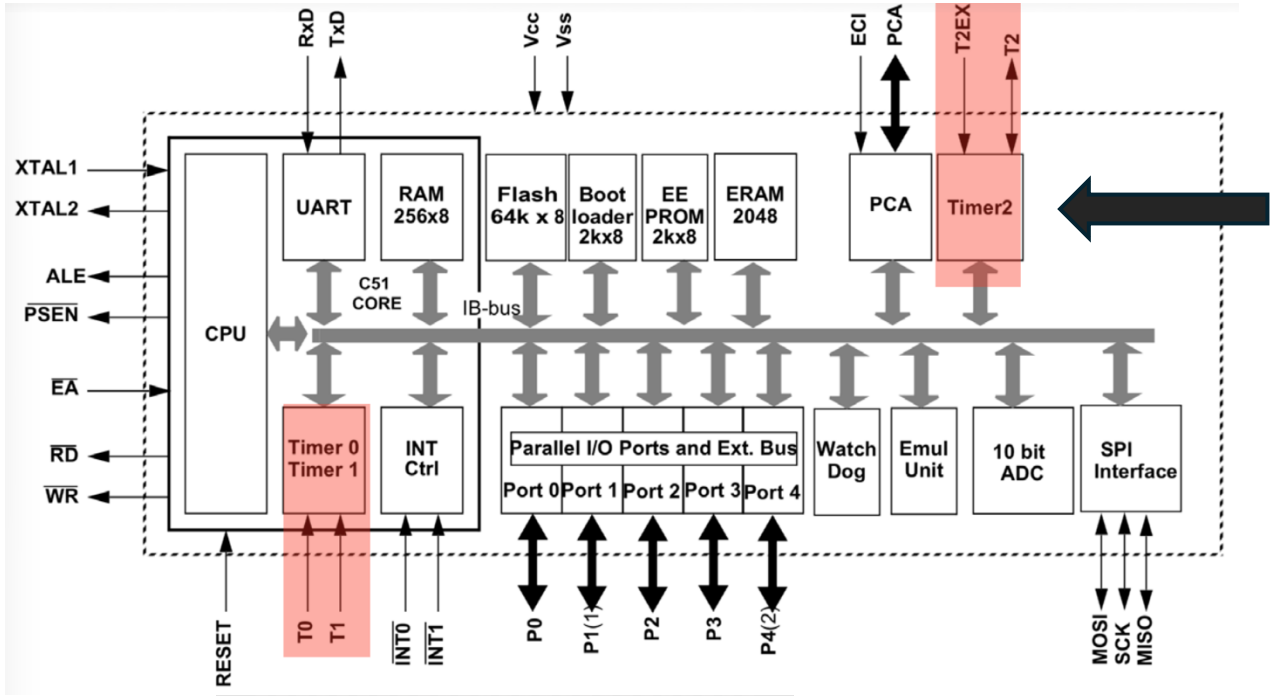
Timer 2 As A Programmable Clock Source

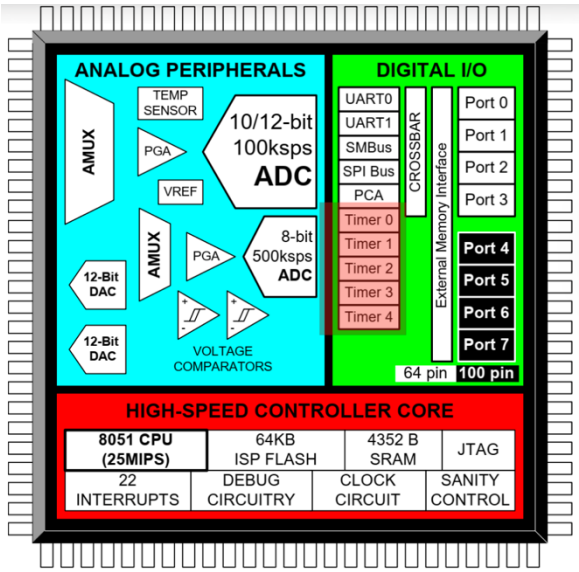




- Many binary- compatible 8051 clone have additional timers.
- The AT89C51AC3's Timer 2 behaves similarly to the other timers.
- Timer 2 is a 16 bit timer configured by the T2MOD register.
- The T2CON register is equivalent to the TCON register.

Timer 2 As A Programmable Clock Source





- 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 (i.e.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
  - Interrupt Driven

Monitoring Timer Overflow

1. “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: It consumes CPU resources, requiring us to constantly monitor a memory location instead of using those cycles for other tasks. ALSO: if we’re doing something else (e.g., displaying text on an LCD), we might miss the exact moment of the overflow.

2. “Interrupt-driven” approach: the 8051’s program counter is vectored to the start address of the 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 2: Square Wave Generator (Polling)

Write an assembly language code in 8051 that will generate a square wave using polling with the timer programming as specified below:

- Generate a 50% duty cycle square wave on a 12-cycle 8051.
- We will count from 2^15 to 2^16, a total of 32,768 values.
- XTAL = 12 MHz, machine cycle = 1 MHz, delay time = 0.033 s.
- Output this square wave to P1.0

Example 2: Square Wave Generator (Polling)

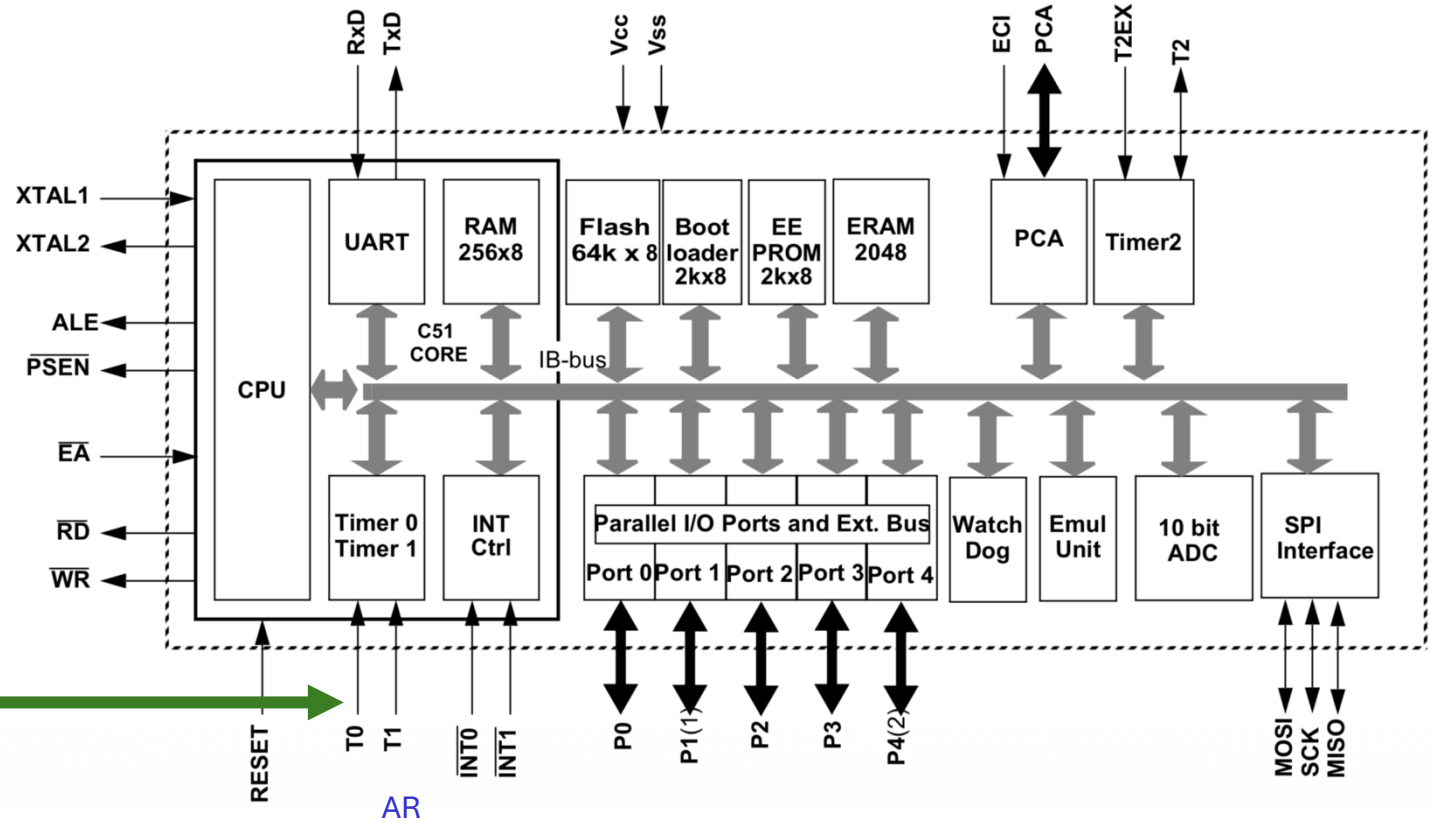
```
;Set up our timer: we'll use Timer 0, Mode 1 (16-bit timer)
MOV TMOD,#01          ;TMOD reg:00000001

;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 Timer 0
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
```

SquareWave.asm

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 when sensors exceed thresholds, waveform edges, etc.

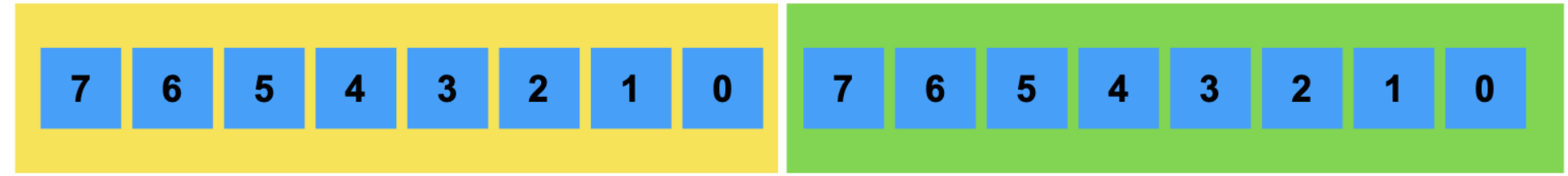
When T0 is in Counter mode, events arriving at T0 pin iterate the counter.
To enable the counters, set TMOD.6 or TMOD.2 to 1.



TMOD Register

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

16 Bits: High 8 bits in TH0, Low 8 bits in TL0



TH0 Register **TL0 Register**

TMOD Register

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

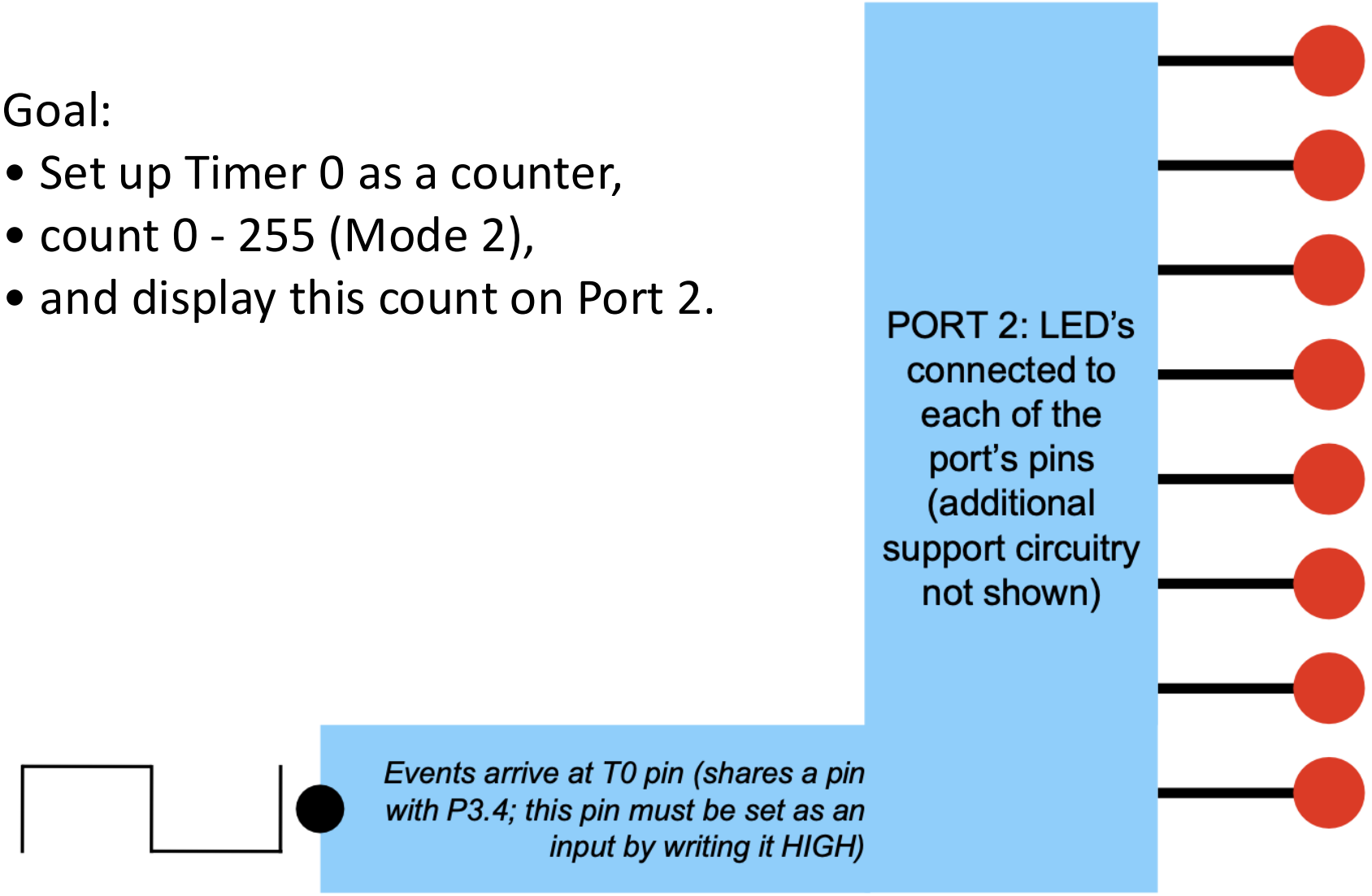
Review

- 00 → Mode 0
- 01 → Mode 1 (16-bit)
- 10 → Mode 2 (auto reload)
- 11 → Mode 3 (split timer)

Example 3: Timer as Counter

Goal:

- Set up Timer 0 as a counter,
- count 0 - 255 (Mode 2),
- and display this count on Port 2.



Example 3: Timer as Counter

```
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
```

Counter.asm

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

Example 4: Timer

Write code that sets Timer 0 to Mode 1 and Timer 1 to Mode 3.
Assume that TMOD holds other values that must not be overwritten.

Example 4: Timer

In 8051: 8 bit

| GATE1 | C/T1 | M1 | M0 | GATE0 | C/T0 | M1 | M0 |

Timer 1 Timer 0

Lower 4 bits → Timer 0
Upper 4 bits → Timer 1

Bits	Meaning
M1 M0	Select timer mode

Timer Modes

Mode	**M1 M0**
Mode 0	00
Mode 1	01
Mode 2	10
Mode 3	11

Example 4: Timer

We need:

- **Timer 0 → Mode 1**
- **Timer 1 → Mode 3**

Timer 0

Mode 1 = 01

Lower bits become:

0000 0001

Timer 1

Mode 3 = 11

Upper mode bits are bits 5 and 4:

0011 0000

Timer1 | Timer0

0011 0001

Hex	Binary
3	0011
1	0001

```
MOV TMOD,#31H
```

Timer 0

Bit	Value
GATE0	0
C/T0	0
M1	0
M0	1

Timer 1

Bit	Value
GATE1	0
C/T1	0
M1	1
M0	1

Example 5: Timer as counter

How can a timer be configured as a counter?

C/T = 1 in TMOD register

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.

Watchdog Timer

A **watchdog timer (WDT)** in the context of the **8051 microcontroller** is a **hardware safety timer** that automatically resets the system if the program stops executing correctly.

A **watchdog timer** is like a “guardian timer” that:
Keeps counting continuously

Must be regularly “reset” by the program
If not reset → it assumes the system is stuck → **forces a reset**

Watchdog Timer

How the watchdog works Step-by-step:

1.WDT starts counting automatically

2.The program must periodically reset it (feed it)

3.If program:

- crashes
- gets stuck in an infinite loop
- hangs due to noise or error

  Then WDT reaches a timeout, System resets automatically

It protects against:

•Infinite loops

•Software bugs

•System hang-ups

•External noise interference

Watchdog Timer

Typical watchdog operation in 8051 variants
Although registers differ by chip, the general flow is:

```
CLR WDT    ; reset watchdog timer (feed it)
```

or sometimes:

```
MOV WDT_CON, #reset_value
```

What happens if WDT is NOT cleared?

- Timer reaches overflow
- CPU reset is triggered
- Program restarts from 0000H

Review

1. What is a timer in 8051?
2. How many timers are there in 8051?
3. What are the modes of 8051 timers?
4. What is TMOD register?
5. Explain Mode 1 operation of 8051 timer.
6. What is the function of TH and TL registers?
7. Write steps to configure Timer 0 in Mode 1.
8. What is a Watchdog Timer?
9. Why is a Watchdog Timer used in embedded systems?
10. What happens if the watchdog timer is not reset in time?
11. Explain the working of a Watchdog Timer in 8051.