EECS 113
Lec 13: Timing part 2

Dept. of EECS, UC Irvine

Review: Timer 0, Timer 1

Timer 1	Timer 0	purpose
TMOD<7:4>	TMOD<3:0>	timer mode
TH1, TL1	TH0, TL0	high/low bytes
TR1 (=TCON.6)	TR0 (=TCON.4)	start(1), stop(0)
TF1 (=TCON.7)	TF0 (=TCON.5)	rollover flag

TMOD.7

Timer 1				Timer 0			
gate	c/t	M1	M0	gate	c/t	M1	M0

TMOD.0

Timer modes

Mode 0	Mode 1	Mode 2	Mode 3
13-bit	16-bit	8-bit auto	8-bit

Longest possible delay of 16-bit timer?

- Start from 0000H, to FFFFH and roll over
  = 65536 cycles
- At timer resolution of 1.085μs,
  - 65536 x 1.085μs = 71.1065ms
- Can get more delay with software
- Actually, overhead should be included
  (start/stop, load timer register, ...)

How precise can you make timing?

- e.g., 5ms at 11.0592MHz oscillator freq
  - 5ms / 1.085µs = 4608 clocks (exactly)
    - actually, 1.085µs comes from 12/11.0592
    - timer value = hex(2^16 - 4608) = EE00
- How about 250µs? 500µs?
  - uneven division: 230.4 or 460.8 cycles

Mode 0 vs Mode 1

- Mode 0: 13-bit timer
  - Range: 0000 to 1FFF hex
    - max steps = (8192 decimal)
- Mode 1: 16-bit timer
  - Range: 0000 to FFFF hex
- Manual reload required

Timer Mode 2:
8-bit, auto-reload

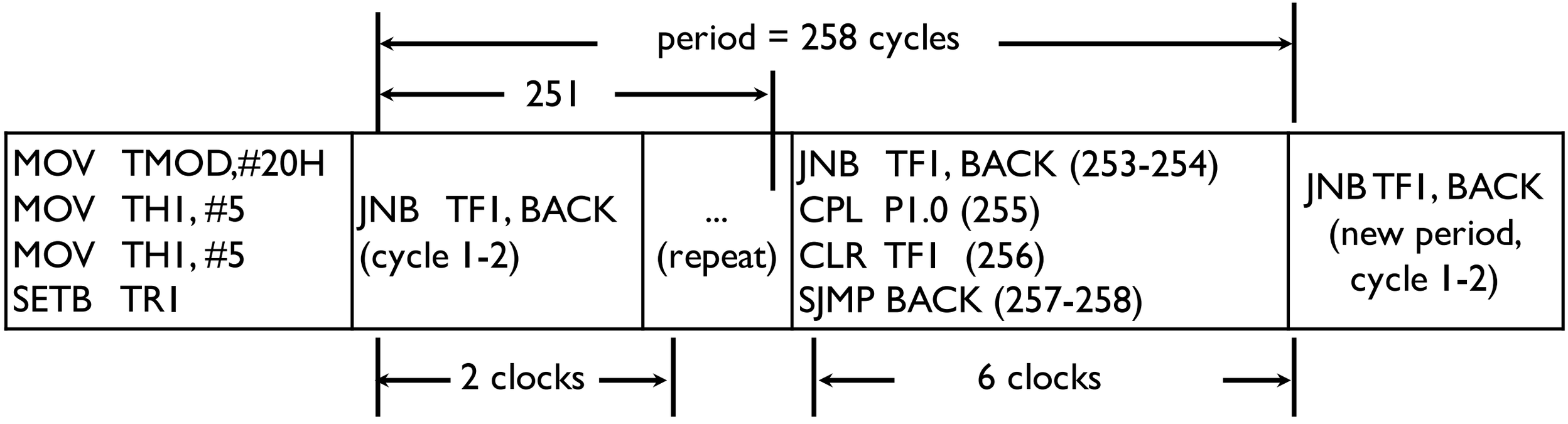
- Load 00-FFH into TH* (* = 0 or 1)
  - in mode 2, TL* gets a copy of TH*
- Start by SETB TR*, stop by CLR TR*
- Rollover => sets TF* just like before
- Difference: TL* gets auto-reloaded w/ TH*

Why auto-reload?

- Very convenient for periodic timing
  - same value each time, saves instructions
- No software overhead in the loop!
  - need instructions to setup and start/stop
  - Once running, works on its own
    easy to compute period,
    independent of instruction timing

Example: square wave

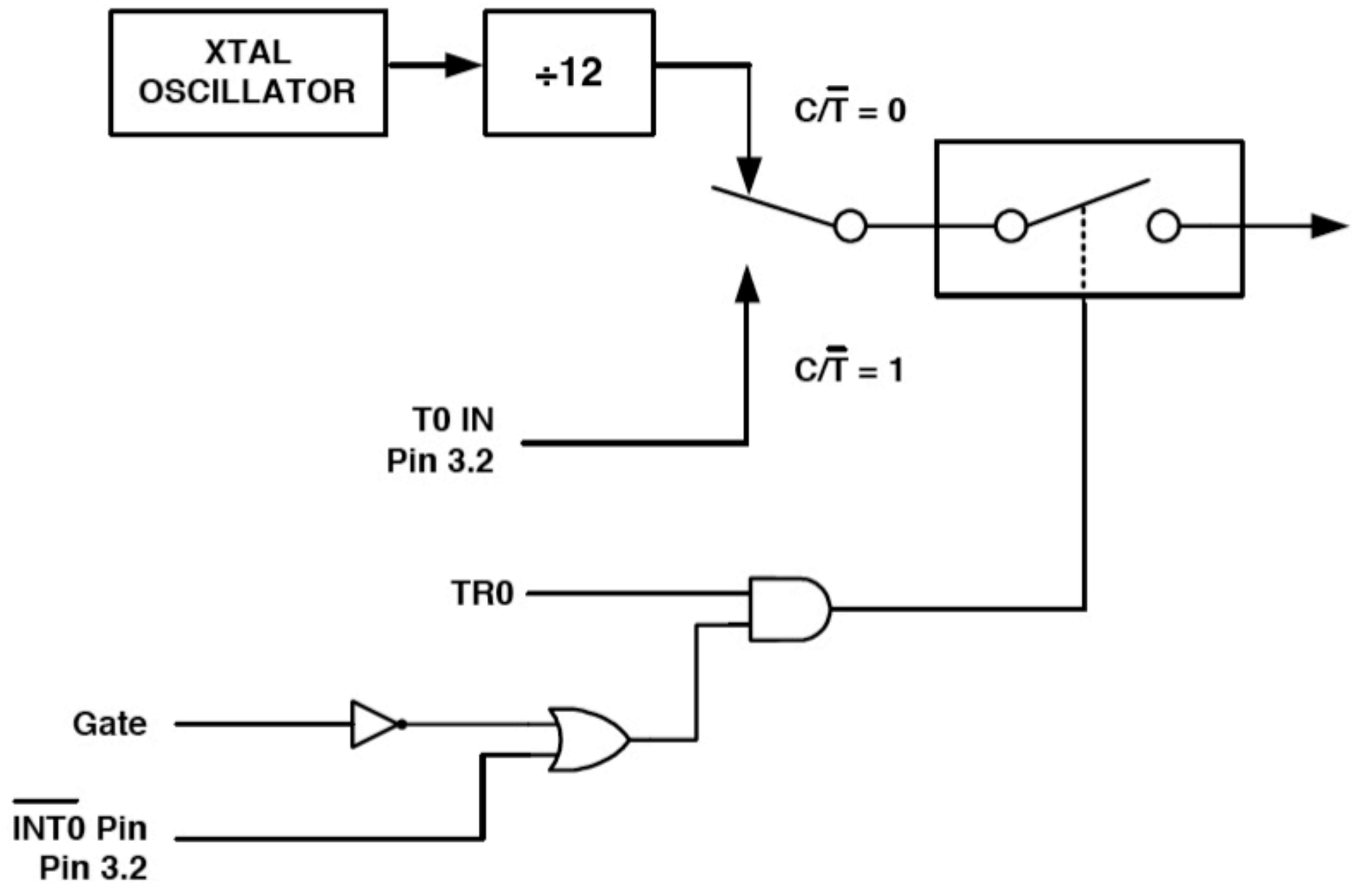
```
                                                          cycles
        MOV   TMOD, #20H    ;; timer 1 mode 2               2
        MOV   TH1, #5       ;; 251 clocks (=256-5)          2
        MOV   TL1, #5       ;; 251 clocks (=256-5)          2
        SETB  TR1           ;; start timer                  1
BACK:   JNB   TF1, BACK     ;; poll till roll over          2
        CPL   P1.0          ;; complement output bit        1
        CLR   TF1           ;; clear flag; auto-reloaded    1
        SJMP  BACK          ;; loop                         2
```



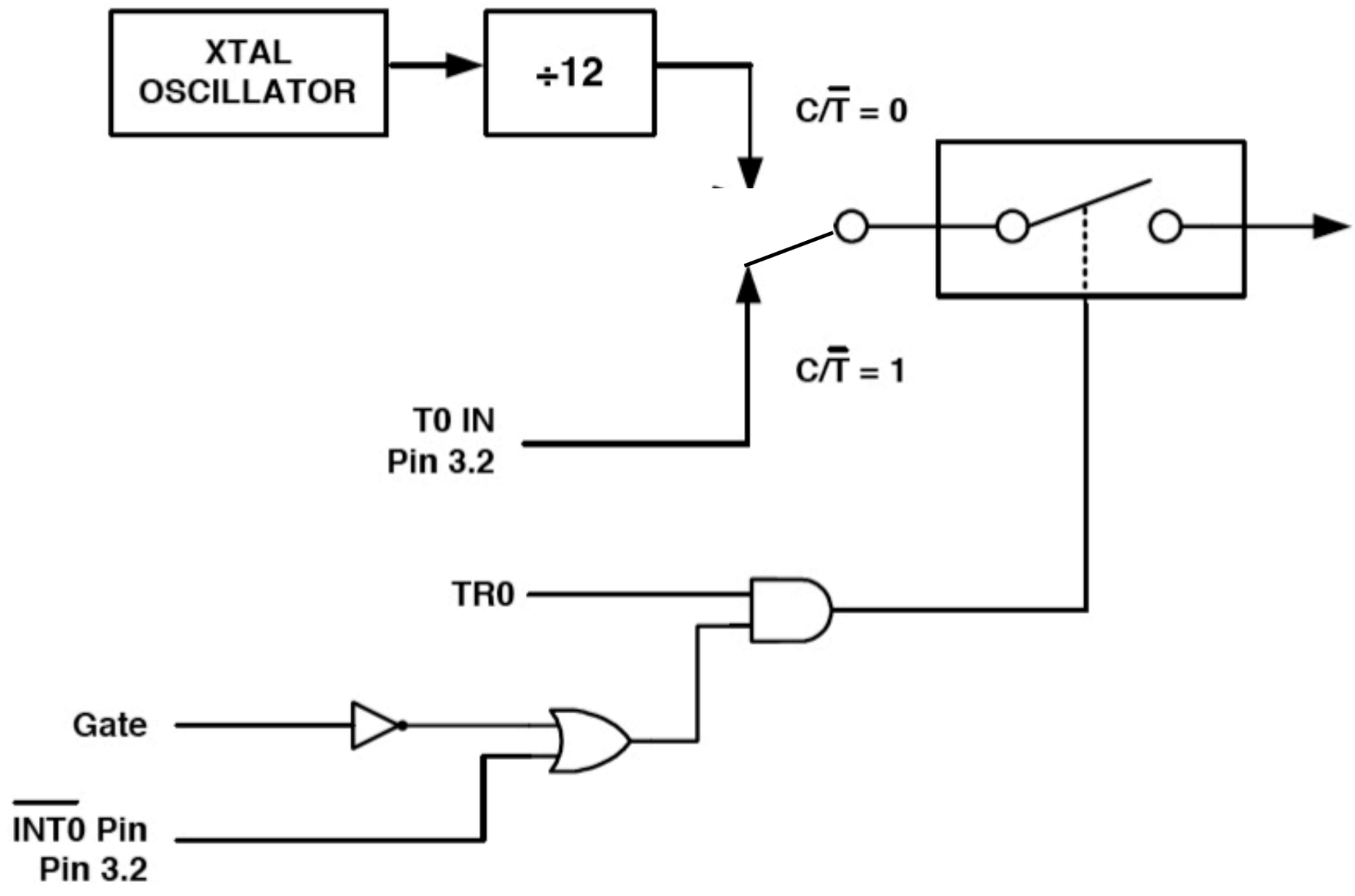
Counter vs. Timer, gate bit

- Counter & hardware use the same, diff src
  - Timer: clock from crystal oscillator/clk
  - Counter: input from T0, T1
- Gate bit:
  - 0: software SETB/CLR TR0 or TR1
  - 1: external pins /INT0, /INT1

Clock vs External T0/1

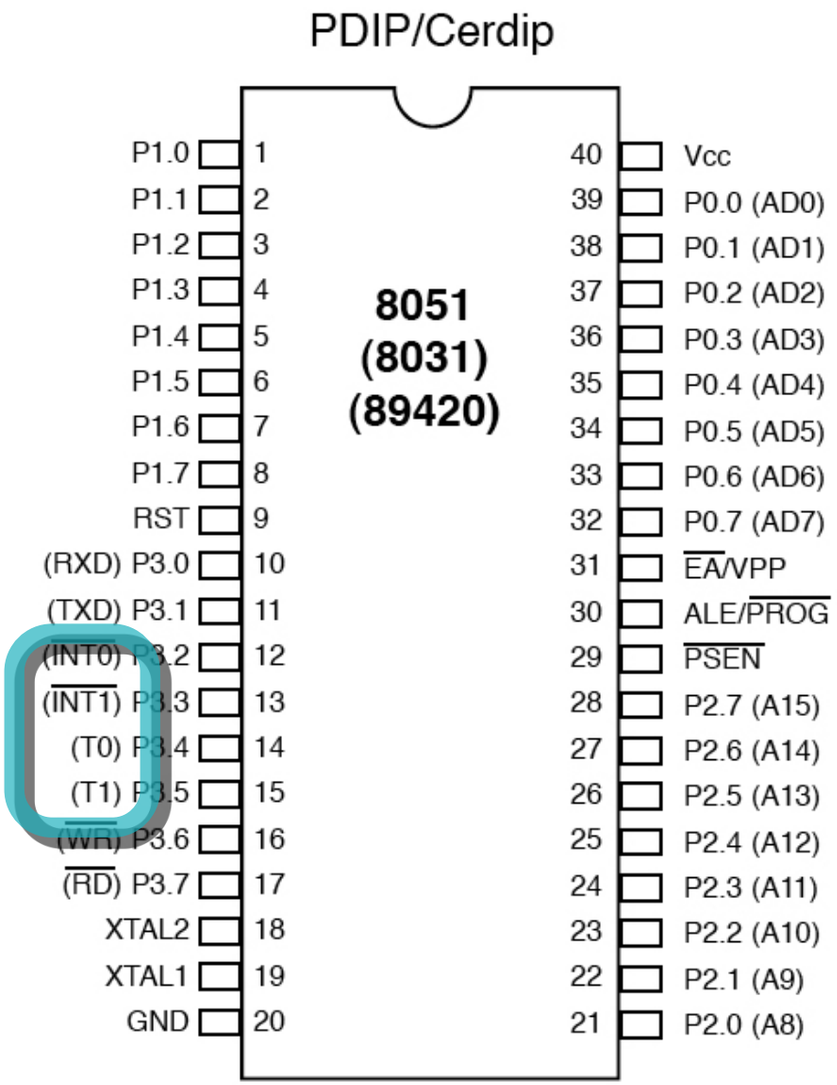


Clock vs External T0/1



Pins on 8051 (40-pin)

- Two timer/counter pins
  - T0, T1
  - Input pins
- Gate bit
  - /INT0, /INT1



TCON register (SFR)

- timer control
  - the TF* and TR* flags
- interrupt control
  - IE1, IT1, IE0, IT0

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0

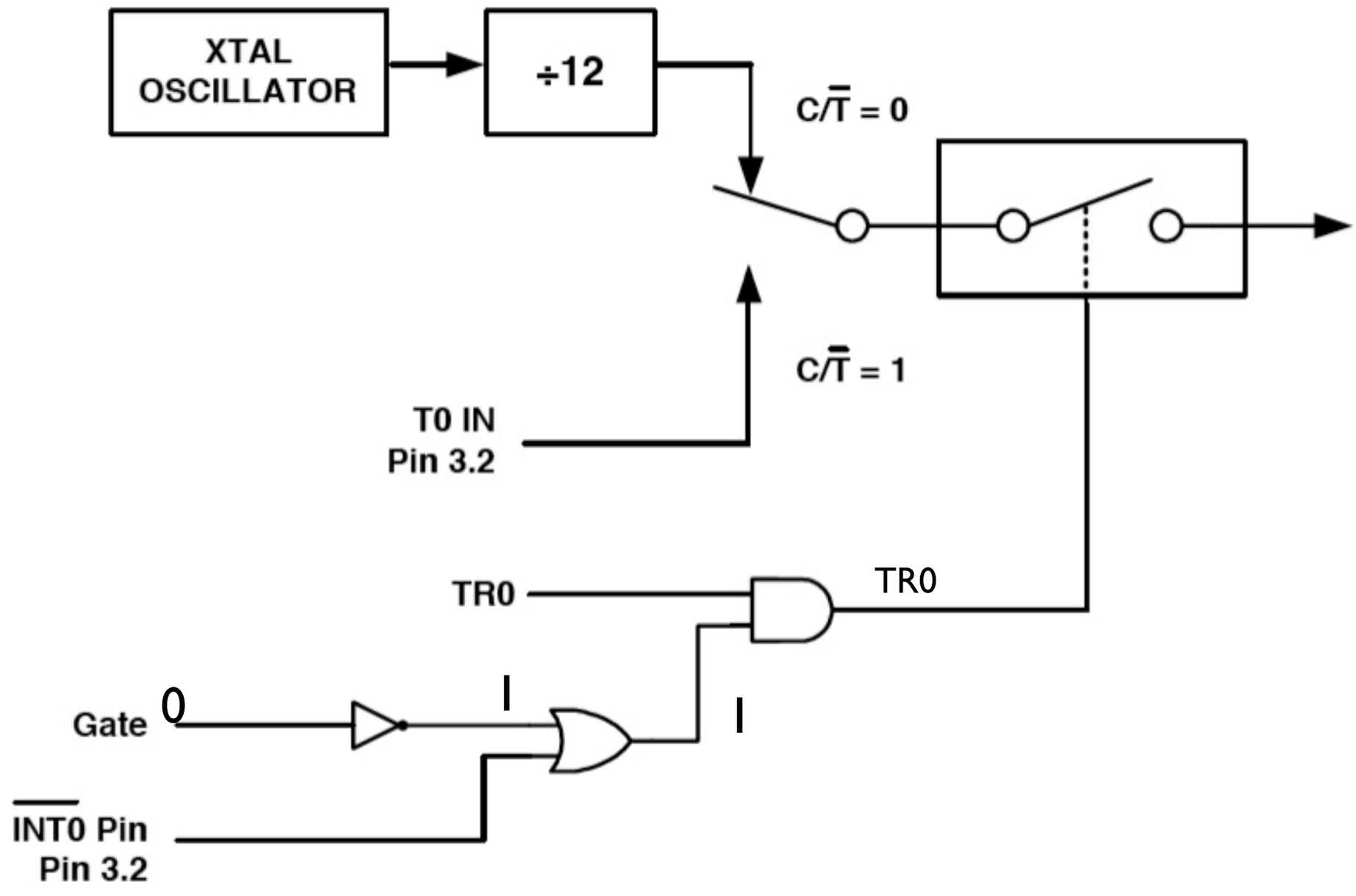
more on GATE bit in TMOD

- 0: use internal (software) to start/stop
- 1: enables /INT0 or /INT1 pin to start/stop, while TR0 or TR1 is enabled.

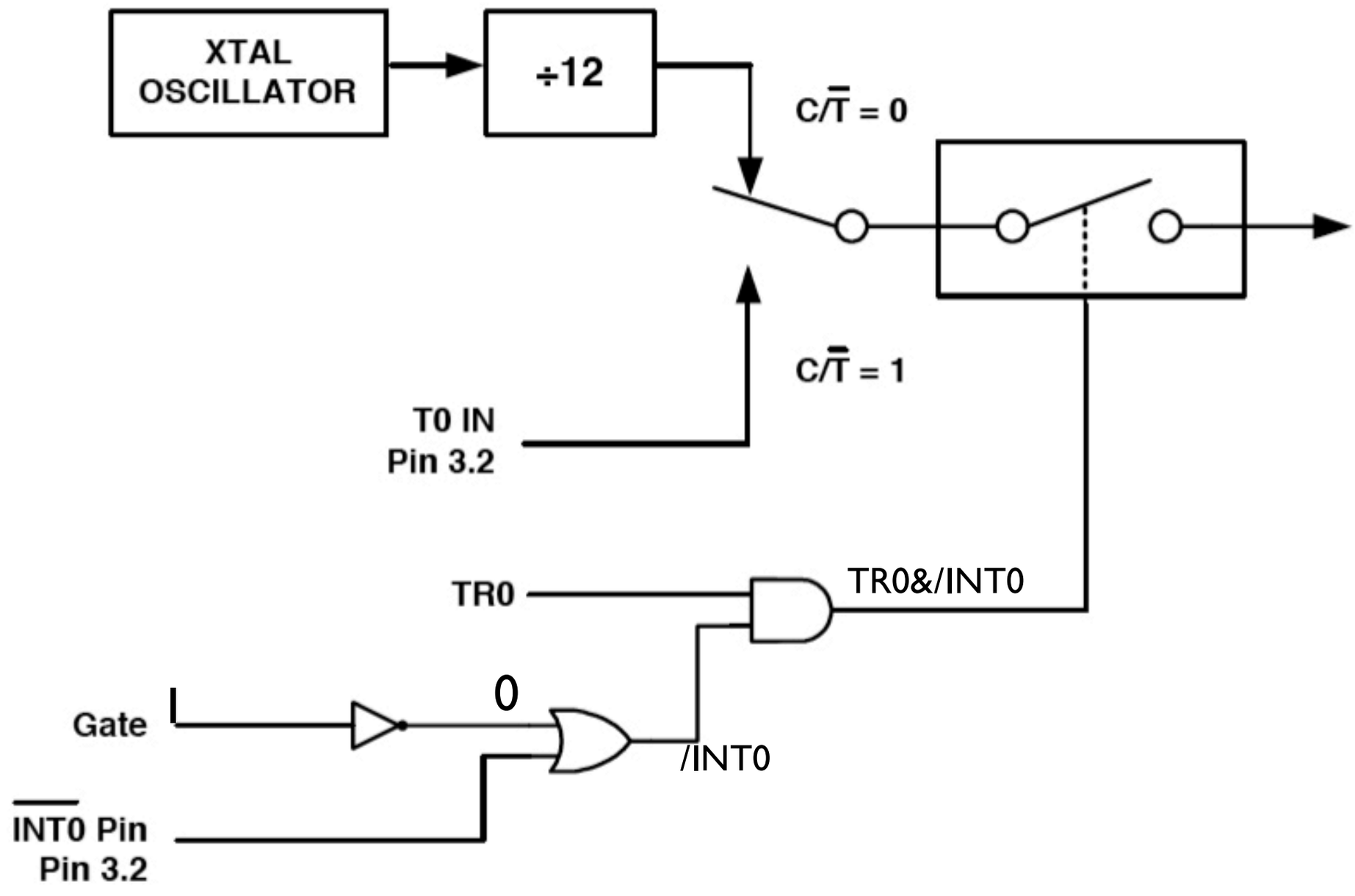
Timer 1				Timer 0			
gate	c/t	M1	M0	gate	c/t	M1	M0

TMOD.7 TMOD.0

Schematic for the GATE, /INT0, TR0



Schematic for the GATE, /INT0, TR0



C programs for Timer

- Fundamentally not too different from asm
- assignment statement instead of MOV
- **while (**TF0**==0) ; to poll** TF0 **flag.**
- TR0 **= 1 to start,** TR0 **= 0 to stop**
- Delay accomplished by count-up roll-over

Example 1: delay1.c (1/2)

```
#include <8051.h>
void T0Delay(void);
void main(void) {
    P1 = 0x55;
    while(1) {
        T0Delay();
        P1 = ~P1;
} }
```

```
void T0Delay(void) {
    TMOD = 0x01;
    TL0 = 0x00;
    TH0 = 0x35;
    TR0 = 1;
    while (!TF0) ;
    TR0 = 0;
    TF0 = 0;
}
```

Example 1: delay1.c (2/2)

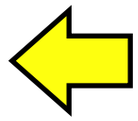
- You may need to define some functions
  - void _sdcc_gsinit_startup(void) { main();}
  - void _mcs51_genRAMCLEAR(void){}
  - void _mcs51_genXINIT(void){}
  - void _mcs51_genXRAMCLEAR(void){}
- Compile with
  - sdcc --nostdlib --idata-loc 0x40 -Wl '-b CSEG=0x40' delay1.c

Periodic execution

- Delay: relative to a point in time
  - Mode 0, Mode 1 (13-bit, 16-bit delay)
    - not as good to use Delay for periodic
- Periodic (e.g., "do task every 500ms")
  - Mode 2 (with auto reload) may be better
  - however, need to be careful with overhead, since timer reg is only 8 bits

Ex. 2: Auto-Reload

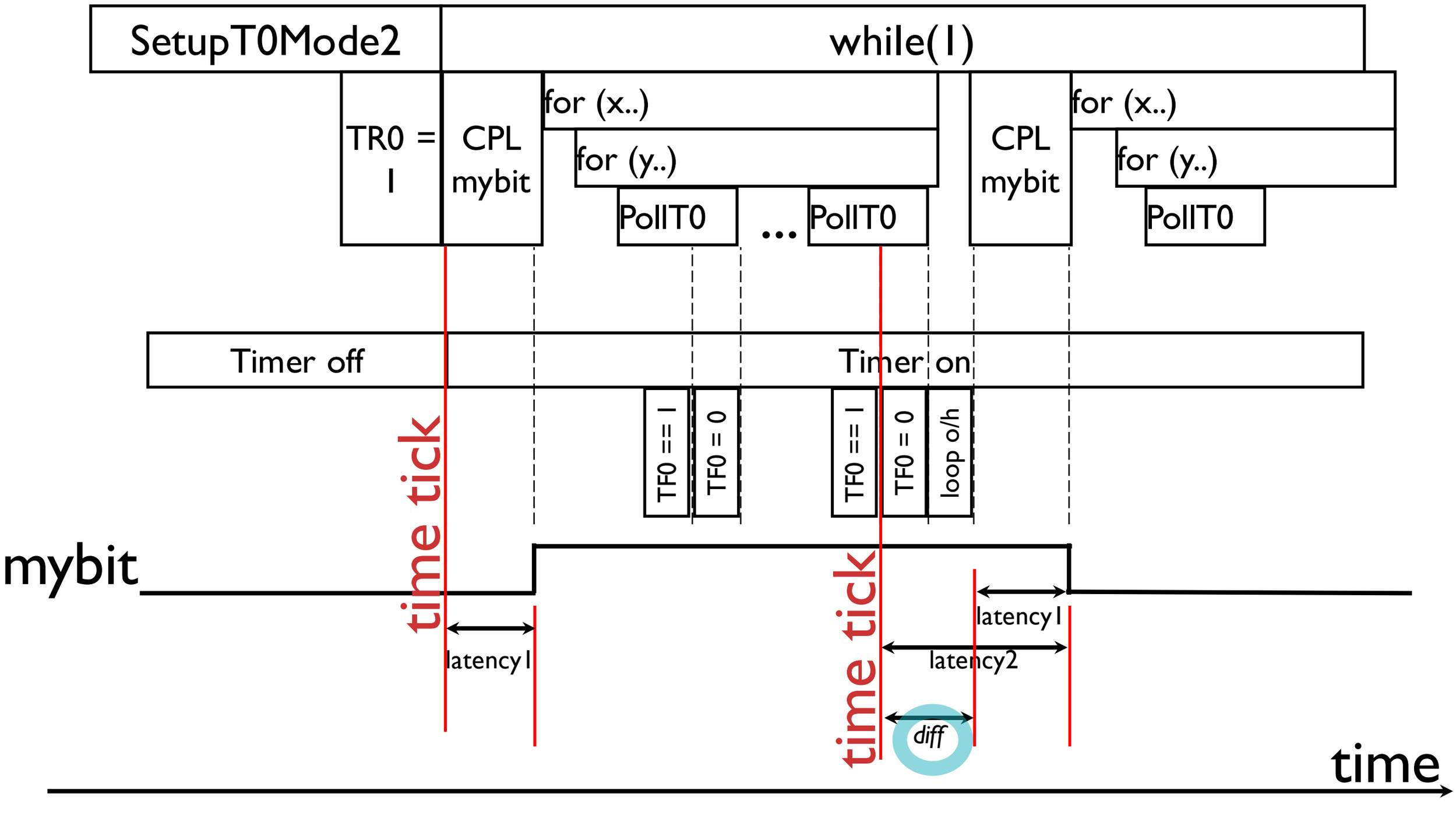
```
#include <8051.h>
void SetupT0M2(void), PollT0(void);
void main(void) {unsigned char x, y;
    SetupT0M2();
    while (1) {
        P1 = ~P1;
        for (x = 0; x < 250; x++) {
            for (y = 0; y < 36; y++) {
                PollT0();
        }   }
}   }
```



```
void SetupT0M2(void) {
    TMOD = 0x02;
    TH0 = -23;
    TR0 = 1;
}

void PollT0(void) {
    while (TF0 == 0) ;
    TF0 = 0;
}
```

Timing for better Ex 2



Summary of Ex 2

- Use auto-reload to absorb overhead
  - don't disable/re-enable timer on reloads!
- Line up with timer as precisely as possible
  - Immediately after start running timer
  - Immediately after polling TF going high
- 1st high interval longer by *diff*; others exact
  - easy to fix by padding nops