CpE 213
Digital Systems Design
C for the 8051
Timers/Counters

Lecture 18

Tuesday 10/28/2005

Overview

- C for the 8051
- Timers/Counters

C for the 8051

References

- See “Writing C Code for the 8051” on Blackboard.
- Also see Chapters 4 and 5 of the Keil uVision2 “Getting Started” manual.
- See me if you need a handout on C programming in general.

Memory Types

- Memory types (optional)
  - may define the type of memory in which variables are placed.
- Examples:
  - `unsigned char data x;`
  - `char code emsg[ ] = “ERROR”;`
  - `int xdata *data ptr; //ptr stored in data`
    `// points to int in xdata`
- Compiler decides if you don’t specify.
  - This is generally the best choice.
  - Function arguments and automatic variables that cannot be located in registers are also stored in the default memory area.

Memory Types

Memory Type	Description
Code	Program memory (64 Kbytes) accessed by opcode MOVC @A+DPTR.
Data	Directly addressable internal data memory; fastest access to variables (128 bytes).
Idata	Indirectly addressable internal data memory; accessed across the full internal address space (256 bytes).
Bdata	Bit-addressable internal data memory; allows bit and byte access (16 bytes).
Xdata	External data memory (64Kbytes); accessed by opcode MOVX @DPTR.
Pdata	Paged (256 bytes) external data memory; accessed by opcode MOVX @Rn.

- Place frequently used variables in internal data memory and less frequently used variables in external data memory.

Location

- Specifying the location of variables is optional. Compiler decides if unspecified.
- See examples in lect18_example.c
- Two ways of specifying:
  - global variables: _at_ addr

```
char data x _at_ 0x2A; //Keil requires 0x
int xdata y _at_ 0x5280;
```

  - direct access within code using built-in commands
    - XBYTE, XWORD for external data memory
    - DBYTE, DWORD for internal data memory
    - CBYTE, CWORD for code memory
    - See example on next slide.

Example

```
//include built-in memory access commands
#include <absacc.h> //REQUIRED!
main(){
  ⇣
  //read from XMEM location 0x5280
  x = XBYTE[0x5280];
  //write x to data memory location 0x42
  DBYTE[0x42] = x;
  ⇣
}
```

New Data Types

type	bits	range	description
bit	1	0-1	bit in bit memory
sbit	1	0-1	SFR bit at specified location
sfr	8	0-255	SFR byte at specified location
sfr16	16	0-64K	16-bit SFR beginning at specified location

- Last 3 types must be global and must specify address.
- Compiler automatically converts between data types when the result implies a different data type.

Example

```
sfr P0 = 0x80;       //sfr P0 located at addr 0x80
sfr seven_seg = 0x90;
sfr16 DPTR = 0x82;  //16-bit sfr named DPTR at 0x82
sbit z = P0^3;       // z is bit sfr located at bit 3 of P0
sbit carry = 0xD7;  // carry is at bit addr 0xD7

main(){
   bit y;                              // a bit variable
   char bdata x = 0xFF;     // a byte located in bit mem
   P0 = 0x00; // clear sfr P0
   y = 1;               // set bit y
   z = y;               // set sbit z equal to bit y
}
```

After executing:

P0 = 0x00

y = 1, z = y => P0 = 0000 1000 = 0x08

Memory Models

- Three basic models: small, compact, large.
- SMALL: Total RAM 128 bytes
  - Will support code sizes up to about 4K but a constant check must be kept on stack usage.
  - The number of global variables must be kept to a minimum.
- COMPACT: Total RAM 256 bytes off-chip, or 128 or 256 bytes on-chip
  - Suitable for programs where, for example, the on-chip memory is applied to an operating system.
  - Rarely used on its own, usually with SMALL.
  - Especially useful for programs with a large number of medium speed 8 bit variables, for which the instructionMOVX A, @R0 is very suitable.

Memory Models

- LARGE: Total RAM up to 64Kb, 128 or 256 bytes on-chip
  - Permits slow access to a very large memory space and is perhaps the easiest model to use.
  - Again, not often used on its own but in combination with SMALL.
  - As with COMPACT, register variables are still used, so efficiency remains reasonable.
- Specify model with directive: **`#pragma SMALL`** or set in project properties in Keil.

Functions

- Specification:

```
return_type name(args) mem_model reentrant
  interrupt n using n {
    //contents of function
}
```

  - everything other than `return_type, name, and args` is optional; do NOT use in prototype

- Examples:

```
char myfun(char x) small {
    //contents of myfun
}
void afun(void) interrupt 0 using 3{
    //contents of afun
 }
```

- Also see lect16_example.c

Function Specification

- interrupt n: function is called in response to interrupt n (n can be 0,1,2,…)
- using n: function will use register bank n exclusively (unless you specify otherwise)
- re-entrant: function may call itself
  - useful for recursion
  - all variables often saved on stack
  - stack may be external
  - default is NOT re-entrant
- memory model can be: small, compact, large

```
/***************************************
lect18_example.c
D. Beetner

A quick bit of code to show how to do a
few things in C for the 8051.  This program
isn't really useful for anything else
(maybe as a template?).

***************************************/

// include special function register defs
// -- note: commented out for this example (so don't
// -------- redefine some SFRs
// #include <reg51.h>

// include mem-access functions (XBYTE[], etc)
#include <absacc.h>

// Declare some SFRs -- MUST BE GLOBAL!
        sfr P0 = 0x80;          //sfr named P0 located at addr 0x80
        sfr P1 = 0x90;

        sfr16 DPTR = 0x82;      // 16-bit sfr named DPTR at addr 0x82

        sbit z = P0^3;          // z is bit 3 of P0
        sbit carry = 0xD7;      // carry is at bit addr 0xD7

// Declare bit-addressable memory
// (must use sbits to access individual bits, hence global)
        unsigned char bdata bits;       // a byte in bit-mem
        sbit bit4 = bits^4;                     // bit4 is bit 4 of bits

// Declare some variables located _at_ a particular address
// MUST BE GLOBAL to use _at_
        char data myabs _at_ 0x42; // located at data addr 42H
        char xdata seven_seg _at_ 0x5280;       // at xdata addr 5280H

// function prototypes (declarations)
        char myfun(char);
        void afun(void);

// main routine
void main(void){

        //local variables
        bit y;                          // a bit variable
        unsigned char bdata x = 0x2A;   // a byte located in bit mem
                                        // initialized as 0x2A
        int data aword;                 // in data mem
        char code emsg[] = "ERROR";     // in code mem
        unsigned char xdata     xsensor;        // in xdata

        char blah;                      // mem space SPEC'D BY COMPILER!
        int blab;                               // MUCH EASIER ON PROGRAMMER!
        float blob;

//      sfr P2 = 0xA0;  // NOT ALLOWED

        // Some bit operations
        P0 = 0x00;      // clear sfr P0
        y = 1;          // set bit y
        z = y;          // set sbit z equal to bit y
```

```
        bits = 0;       // set variable bits (in bit mem) to 0
        bit4 = 1;       // set bit bit4 to 1


        // play with some variables at known locations
        myabs = 0x2A;   // set location 42 (myabs) to 0x2A
        DBYTE[0x42] = 0;

        // call a function
        myabs = myfun(blah);

}


// Define contents of function myfun
char myfun(char x) small{       // runs using a small memory model
        return x--;
}

// Define afun as a function responding to interrupt 0
// and using register bank 3
void afun(void) interrupt 0 using 3{
        static char x;

        x++;
}
```

Timers and Counters for the 8051

Slides adapted from Mr. K. T. Ng

What Is a Timer/Counter?

- A counter is a device that generates binary numbers in a specified count sequence when triggered by an incoming clock pulse.
- Timers are counters that count pulses. If the pulses are “clock” pulses, then the timers count time.

Timers/Counter on the 8051

- The 8051 comes with two 16-bit timers/counters, both of which may be controlled, set, read, and configured individually.
- The 8051 timers have three general functions:
  - Keeping time and/or calculating the amount of time between events (Timer Mode).
  - Counting the events themselves (Event Counter Mode)
  - Generating baud rates for the serial port.
- Always count UP irrespective of function as a timer or a counter.

Difference between a Timer and a Counter

- When the incoming clock frequency is known, we can generate a fixed period of time known to the designer by setting a preloaded value. This is called a "timer".
- It is also called an "interval timer" since it is measuring the time of the interval between two events.
- When the incoming clock is irregular and we are only interested in the number of occurrences of the pulse, this is called a "counter". Since we are counting events, is also known as an "event counter".
- In 8051, timer gets its clock from the oscillator (crystal that connected to the CPU) frequency (1/12 of it).
- Counter gets the clock from an external pin:
  - P3.4 for timer 0, and
  - P3.5 for timer 1.

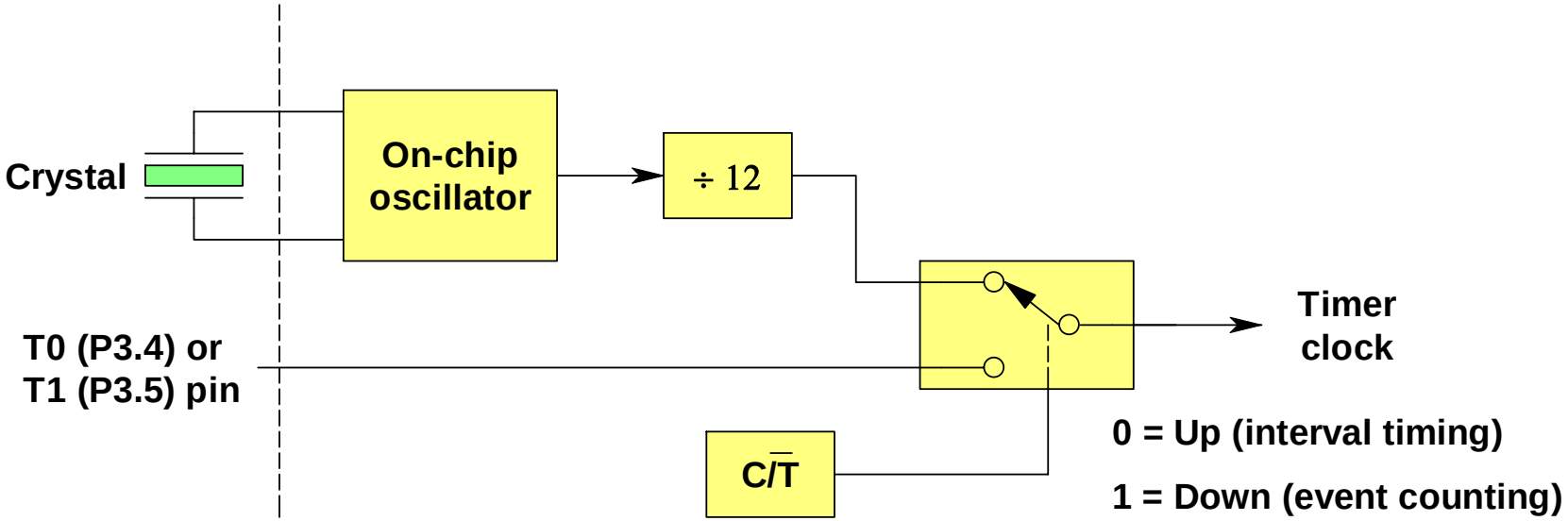
Difference between a Timer and Counter (Cont.)

- Timer mode (if the timer control bit is set to timer, the timer uses the system clock)
  - Increments by 1 every machine cycle.
  - A single machine cycle consists of 12 crystal pulses.
- Event mode (if the timer control bit is set to event, the timer/counter uses a port bit)
  - Increments by 1 for each pulse on P3.4 for Timer 0, and P3.5 for Timer 1, respectively.

Applications of Timers/Counters

- Generating time delays.
- Measuring pulse duration or timing intervals.
- Counting pulses or events.
- Generating baud rate clock for the internal 8051 serial I/O port.
- Generating interrupts.

Timer/Counter Clocking Source



Clocking Source

Timer Special Function Registers (SFRs)

- As mentioned before, the 8051 has two timers that essentially function the same way.
- One timer is called TIMER0 and the other is TIMER1.
- The two timers share two SFRs (TMOD and TCON) that control the timers.
- Each timer also has two SFRs dedicated solely to itself (TH0/TL0 and TH1/TL1).
- It is common practice to use given SFR names to refer to them, but in reality an SFR has a numeric address.
- When you enter the name of an SFR into an assembler, it internally converts it to the specified SFR address.

Location of Timer SFRs

SFR Name	Description	Address
TH0	Timer 0 High Byte	8CH
TL0	Timer 0 Low Byte	8AH
TH1	Timer 1 High Byte	8DH
TL1	Timer 1 Low Byte	8BH
TCON	Timer Control	88H
TMOD	Timer Mode	89H

Timer Control Register (TCON)

- TCON has the following functions:
  - Turns Timer 0 and Timer 1 on or off.
  - Sets trigger type (edge or level) for interrupt 0 and interrupt 1.
  - Sets flags when timer 0 or timer 1 overflow.

Summary of TCON

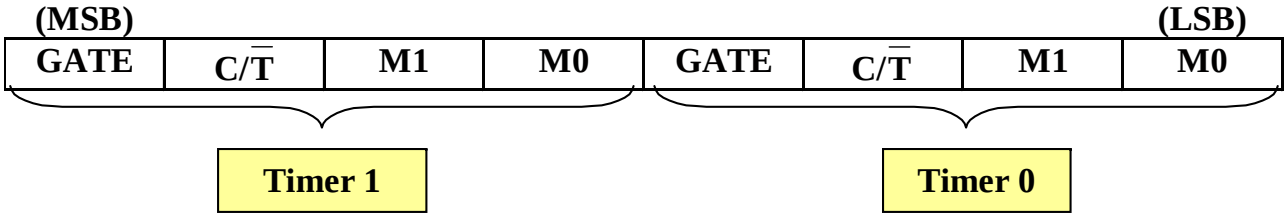
Bit	Symbol	Bit Address	Description
TCON.7	TF1	8FH	Timer 1 overflow flag. Set by hardware upon overflow; cleared by software, or by hardware when processor vectors to interrupt service routine
TCON.6	TR1	8EH	Timer 1 run-control bit. Set/cleared by software to turn timer on/off
TCON.5	TF0	8DH	Timer 0 overflow flag
TCON.4	TR0	8CH	Timer 0 run-control bit
TCON.3	IE1	8BH	External interrupt 1 edge flag. Set by hardware when a falling edge is detected on $\overline{\text{INT1}}$; cleared by software, or by hardware when CPU vectors to interrupt service routine
TCON.2	IT1	8AH	External interrupt 1 type flag. Set/cleared by software for falling edge/low-level activated external interrupt
TCON.1	IE0	89H	External interrupt 0 edge flag
TCON.0	IT0	88H	External interrupt 0 type flag

- Only the upper 4 bits are related to timers; the lower 4 bits have nothing to do with timers - they have to do with interrupts.
- TCON is a bit-addressable register.

Timer Mode Control (TMOD) Register

- TMOD controls timer 0 and timer 1 functions.
- The register's upper nibble controls Timer 1 and the lower nibble controls timer 0.
- The four bits for each timer control a gate function, selection of counter or timer, and timer mode.
- The timer or counter function is selected by control bits C/T; these two timers/counters have 4 operating modes, selected by appropriate M0 and M1 bits in the TMOD register.
- TMOD is not a bit-addressable register. It has to be manipulated as a byte.

Timer Mode (TMOD) Register Summary



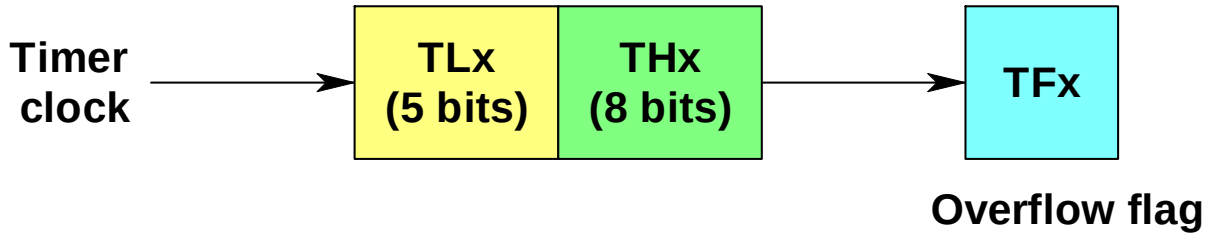
GATE If GATE = 1, timer x will run only when TRx = 1 and INTx = 1. If GATE = 0, timer x will run whenever TRx = 1.

C/$\overline{T}$ Timer mode select. If C/$\overline{T}$ = 1, timer x runs in counter mode taking its input from Tx pin. If C/$\overline{T}$ = 0 , timer x runs in timer mode taking its input from the system clock.

M1	M0	Operating Mode
0	0	13-bit timer mode (8048 mode).
0	1	16-bit timer mode.
1	0	8-bit auto-reload mode. "THx" holds a value which is to be reloaded into "TLx" each time it overflows.
1	1	(Timer 0) TL0 is an 8-bit Timer/Counter controlled by the standard Timer 0 control bits. TH0 is an 8-bit timer only controlled by Timer 1 control bits.
1	1	(Timer 1) Timer/Counter 1 is stopped.

Mode 0 - 13 Bit Counter

- TH, TL = XXXXXXXX, 000XXXXX
- 8 bits from TH
- 5 bits from LSB of TL
- For compatibility with older microcontroller (8048), not very useful now.
- Range = 0 to 8191.

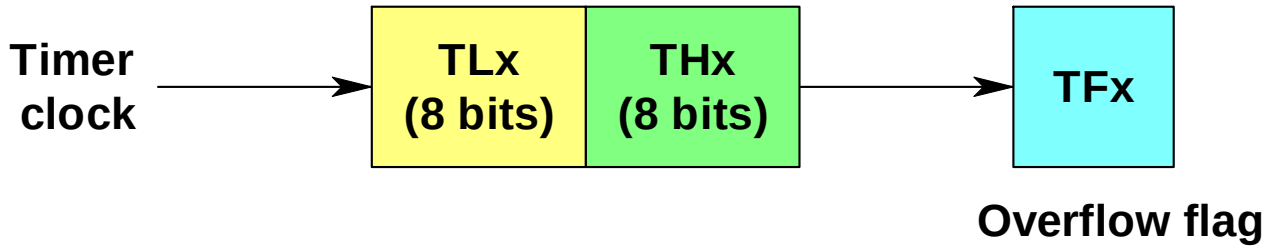


8051 Timer Mode 0

Timer Mode 1 - 16 Bit Counter

- 8 bits from TH, another 8 bits from TL.
- Each byte loaded separately
- Counts UP, so initialize with negative value.
- The counters start counting when enabled (by a bit in the SFR).
- Counts time (when acting as timer) until a preset number as specified by the THx and TLx registers is reached.
- Generates an overflow (interrupt) when timed out.
- In order to repeat the process the registers TH and TL must be reloaded with the original value and the timer overflow flag must be reset under the control of the program.

Timer Mode 1 (Cont.)



8051 Timer Mode 1

- Example

- Why doesn’t -1000/256 work?

Finding Values to be Loaded into the Timer

- Assume that we know the desired amount of timer delay, t_{delay}. To calculate the values to be loaded into the TL and TH registers, using crystal frequency of $X_{crystal}$ MHz, the following steps are needed:
  - Step 1: find n, where n = $\frac{t_{delay} \bullet X_{crystal}}{12}$
  - Step 2: Perform 65536 - n
  - Step 3: Convert step 2 result to hex, assume this result is yyxx
  - Step 4: Set TL = xx and TH = yy.

A Better Way

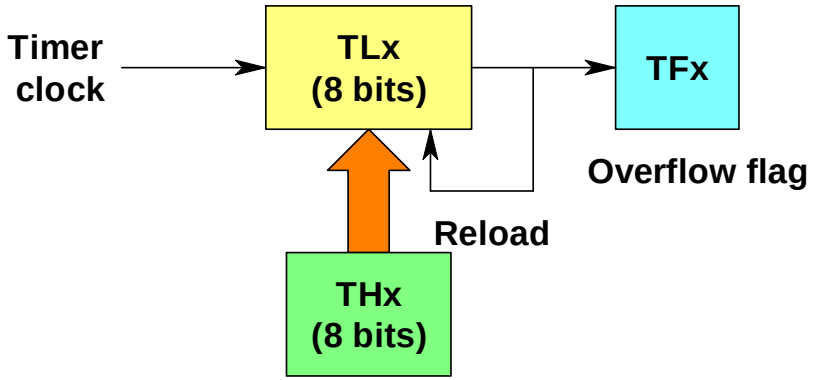
- A better way is to let the assembler do the arithmetic.
- Suppose the required delay count number, n, is 4608. When using timer0, we can load this value into TH0 and TL0 as follows:

```
MOV TH0,#-4068 SHR 8    ;MSB of -4068
MOV TL0,#-4068          ;LSB of -4068
```

- What does SHR 8 mean ?
  - It means the assembler would take the 16-bit number and shift it right by 8 bits. This gives us the MSB of -4068.
- The assembler automatically selects only the bottom 8 bits of the value, so the LSB of -4068 goes into TL0.

Mode 2 - 8 Bit Auto-Reload

- TL0 or TL1 is an 8 bit counter/timer
- TH0 or TH1 holds an initialization value
- TL reloaded from TH on 255 to 0 transition
- The reload leaves TH unchanged.
- Used to generate UART baud rate clock; a periodic flag or interrupt.

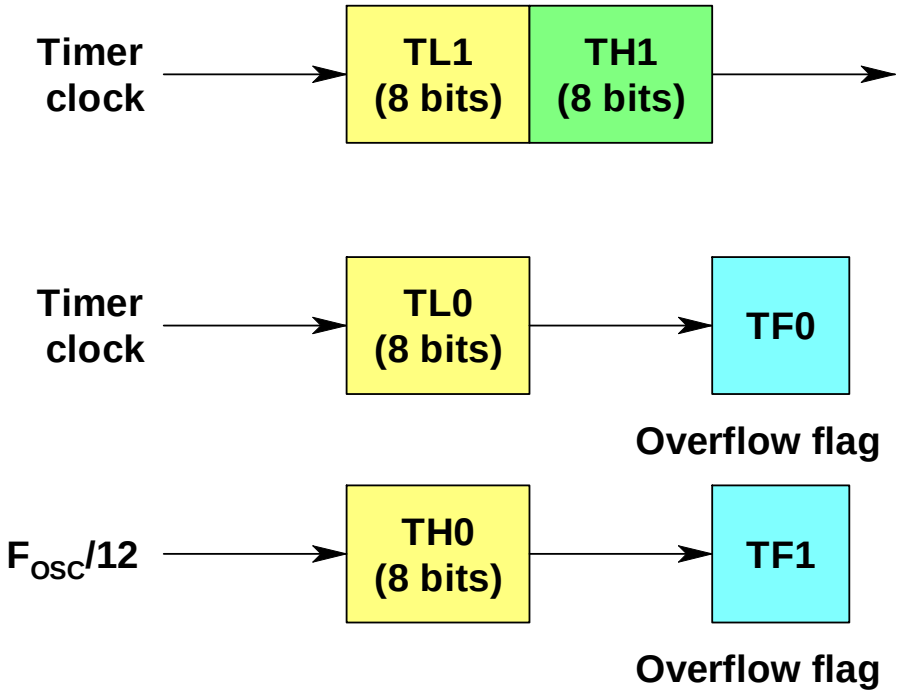


8051 Timer Mode 2

Timer Mode 3 - Three from Two

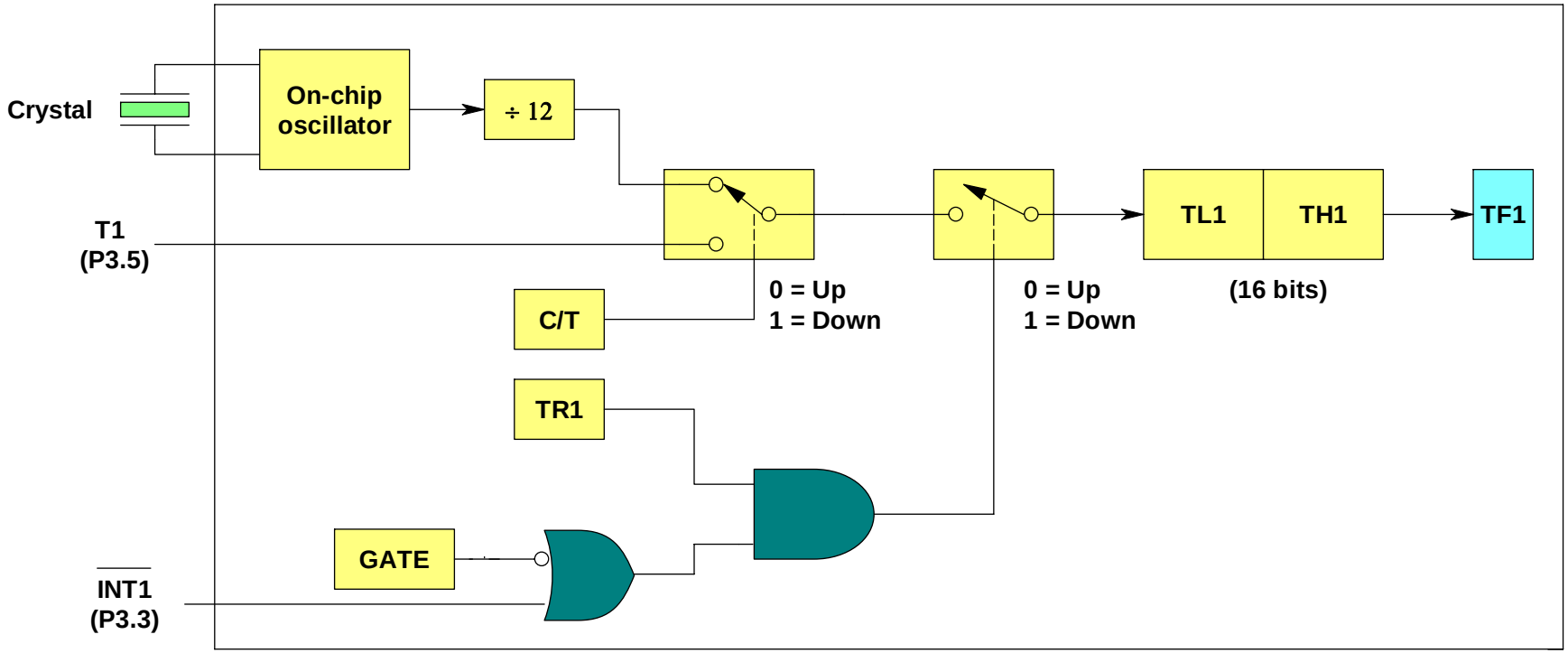
- T0 is split into two 8-bit counter/timers.
- First counter (TL0) act like mode 0.
- Second counter (TH0) counts CPU cycles.
  - Uses TR1 (timer 1 run bit) as enable.
  - Uses TF1 (timer 1 overflow bit) as flag.
  - The clock frequency is $F_{osc}/12$
- T1 becomes a 16-bit timer that can be started and stopped at any time by the mode operating bits M0 & M1, but cannot cause an interrupt.
- T1 is normally in mode 2 when T0 in mode 3
- T1 is used for baud rate clock while T0 provides two 8-bit counters.

Timer Mode 3 (Cont.)



8051 Timer Mode 3

C/T’ and GATE Bits



Timer 1 operating in mode 1

C/T’ and GATE Bits (Cont.)

- C/T’ is used as the timer or counter select bit. It is cleared for timer operation (input from internal system clock), and set for counter operation (input from Tn input pin).
- When the Gate control is set, Timer/Counter n is enabled only while INTn pin is high and TRn control pin is set. When cleared, Timer n is enabled whenever TRn control pin is set.
- Setting Gate = 1 allows us a way of controlling the stop and start of the timer externally at any time via a simple switch.

Summary

- Timers/counter implemented in HW
- Counts instruction cycles (timer) or
- External events (counter)
- 8, 13, or 16 bit counters available
- All count UP
- Overflow (interrupt) when MAXCOUNT reached
- Gated and auto-reload variations

Other variations

- Three counters on 8052 (T0,T1,T2)
- Watchdog timer
- Capcon (Capture/Comparator) registers
- PWM

Programming in Mode 1

- To generate a time delay using timer mode 1, the following steps are taken.
  1. Load the TMOD value register indicating which timer (0 or 1) is to be used and which timer mode is selected.
  2. Load registers TL and TH with initial count values.
  3. Start the timer by the instruction "`SETB TR0`" for timer 0 and "`SETB TR1`" for timer 1.
  4. Keep monitoring the timer flag (TF) with the "`JNB TFx,target`" instruction to see if it is raised. Get out of the loop when TF becomes high.

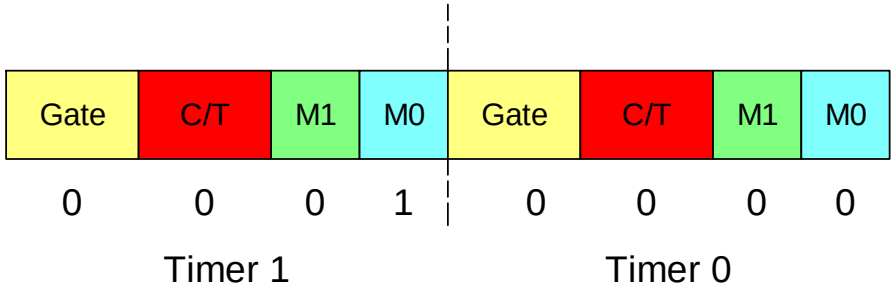
Programming in Mode 1 (Cont.)

1. Stop the timer with the instructions "`CLR  TR0`" or "`CLR  TR1`", for timer 0 and timer 1, respectively.
2. Clear the TF flag for the next round with the instruction "`CLR  TF0`" or "`CLR  TF1`", for timer 0 and timer 1, respectively.
3. Go back to step 2 to load TH and TL again.

- The programming techniques mentioned here are also applicable to counter/timer mode 0. The only difference is in the number of bits of the initialization value.

Example of Time Delay Generation Using Mode 1

- Generate a time delay of 1 ms Timer 1 in mode 1. Assume crystal frequency to be 12 MHz.
- Solution:
  - First calculate the value n:
  - n = (1 × 10^-3 × 12 × 10^6) / 12 =1000
  - 65536 - 1000 = 64536 = FC18H.
- The initial values to be loaded into TH1 and TL1 are FC and 18, respectively.
- For timer 1 to be set to mode 1, the configuration of the TMOD register should be 10H.



Time Delay Using Mode 1 (Cont.)

The time delay program segment is as follows:

```
        MOV   TMOD,#10H    ;Timer 1, mode 1 (16-bit mode)
HERE:   MOV   TL1,#18H     ;TL1 = 18H
        MOV   TH1,#0FCH    ;TH1 = FCH
        SETB  TR1          ;Start timer 1
AGAIN:  JNB   TF1,AGAIN    ;Monitor timer flag 1
                           ;until it rolls over
        CLR   TR1          ;Stop timer 1
        CLR   TF1          ;Clear timer 1 flag
```

Generating a Square Wave Using Timer 0

- Write a program fragment to continuously generate a square wave of 2 kHz frequency on pin P1.5 using timer 0. Assume the crystal oscillator frequency to be 12 MHz.
- Solution: Initial calculations are as follows.
- The period of the square wave is T = 1/(2 kHz) = 500 μs. Each half = 250 μs.
- The value n for 250 μs is:
  - (250 × 10^(-6) × 12 × 10^6) / 12 = 250
- 65536 - 250 = FF06H.
- TL = 06H and TH = 0FFH.

Square Wave Generation on Pin P1.5 (Cont.)

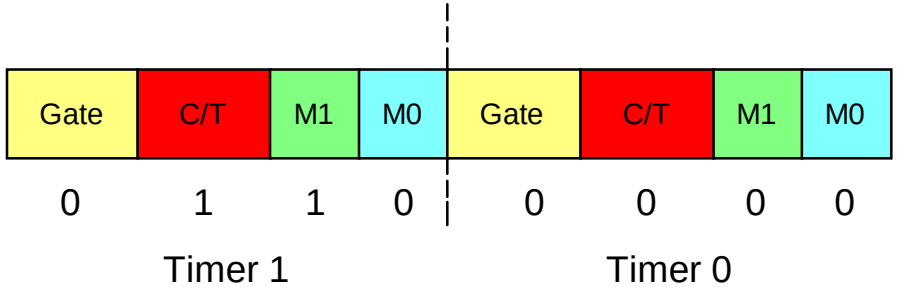
```
        MOV   TMOD,#01    ;Timer 0, mode 1
AGAIN:  MOV   TL0,#06H    ;TL0 = 06H
        MOV   TH0,#0FFH   ;TH0 = FFH
        SETB  TR0         ;Start timer 0
BACK:   JNB   TF0,BACK    ;Stay until timer
                          ;rolls over
        CLR   TR0         ;Stop timer 0
        CPL   P1.5        ;Complement P1.5 to
                          ;get Hi, Lo
        CLR   TF0         ;Clear timer flag 0
        SJMP  AGAIN       ;Reload timer since
                          ;mode 1 is not
                          ;auto-reload
```

Steps to Program in Mode 2

- To generate a time delay using the timer mode 2, the following steps are taken:
  1. Load the TMOD value register indicating which timer (0 or 1) is to be used; select timer mode 2.
  2. Load TH register with the initial count value. As it is an 8-bit timer, the valid range is from 00 to FFH.
  3. Start the timer.
  4. Keep monitoring the timer flag (TFx) with the "`JNB TFx,target`" instruction to see if it is raised. Get out of the loop when TFx goes high.
  5. Clear the TFx flag.
  6. Go back to step 4, since mode 2 is auto-reload.

Program Example for Mode 2

- Write a program segment that uses timer 1 in mode 2 to toggle P1.0 once whenever the counter reaches a count of 100. Assume the timer clock is taken from external source P3.5 (T1).
- Solution:



- The TMOD value is 60H
- The initialization value to be loaded into TH1 is
- 256 - 100 = 156 = 9CH

Program Example in Mode 2 (Cont.)

```
        MOV   TMOD,#60h    ;Counter1, mode 2, C/T'= 1
                           ;external pulse
        MOV   TH1,#9Ch     ;Counting 100 pulses
        SETB  P3.5         ;Make T1 input
        SETB  TR1          ;Start timer 1
BACK:   JNB   TF1,BACK     ;Keep doing it if TF = 0
        CPL   P1.0         ;Toggle port bit
        CLR   TF1          ;Clear timer overflow flag
        SJMP  BACK         ;Keep doing it
```

- Note in the above program the role of the instruction "SETB P3.5". Since ports are set up for output when the 8051 is powered up, we make P3.5 an input port by making it high.

Reading the Value of a Timer

- If the timer is in an 8-bit mode - that is, either 8-bit auto-reload mode (mode 2) or in split timer mode (mode 3), then reading the value of the timer is simple. Simply read the 1-byte value of the timer and that's it.
- For reading the value of a 16-bit timer, two common ways are:
  - Read the actual value of the timer as a 16-bit number.
  - Detect when the timer has overflowed.
- When reading a 13-bit or 16-bit timer, there is a potential problem of "phase error" if the low-byte overflows into the high byte between the two read operations

Reading the Value of a Timer (Cont.)

- Consider the case that the timer value was 14/255 (high byte 14, low byte 255) but you read 15/255. Why ?
- The solution to this phase error problem is:
  - Read the high byte of the timer first.
  - Then read the low byte.
  - Then read the high byte again.
  - If the high byte read the second time is not the same as the high byte read the first time, we read both bytes again.
- The code that appears in here is known as reading a timer “on the fly.”

Reading the Value of a Timer (Cont.)

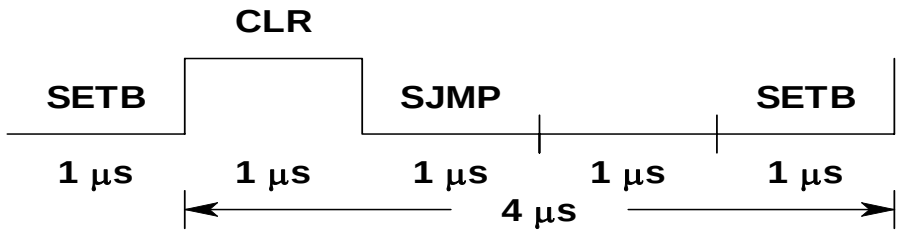
```
REPEAT:   MOV  A,TH0
          MOV  R0,TL0
          CJNE A,TH0,REPEAT
          MOV  R7,A
```

- When the code terminates we will have the low byte of the timer in R0 and the high byte in R7.
- A much simpler alternative is to turn off the timer run bit (i.e. `CLR TRx`), read the timer value, and then turn on the timer run bit (i.e. `SETB TRx`).
- The drawback of this simple approach is that the timer will be stopped for a few machine cycles, which may not be tolerable in some cases.

Short Timing Interval Using Software Loops

- Very short intervals can not be generated using timers because of the overhead needed to start and stop the timers.
- Such short timing intervals are usually generated using tight software loops.
- The following code segment generates a 250 KHz wave with 25% duty cycle on P1.0. Assume XTAL = 12 MHz.

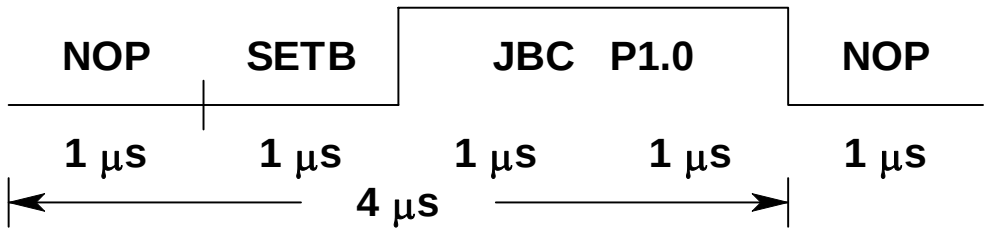
```
LOOP:    SETB  P1.0  ;1 machine cycle
         CLR   P1.0  ;1 machine cycle
         SJMP  LOOP  ;2 machine cycles
```



Short Timing Interval (Cont.)

- The following code segment generates a 250 KHz square wave on P1.0. Assume XTAL = 12 MHz.

```
LOOP:     NOP                ;1 machine cycle
          SETB  P1.0         ;1 machine cycle
          JBC   P1.0,LOOP    ;2 machine cycles
```



Summary of Techniques

Time Interval	Technique
No limit	Software loops
No limit	16-bit plus software loops
65536 machine cycles	16-bit timer
256 machine cycles	Auto-reload (8-bit)
<~ 10 machine cycles	Tight software tuning