CG3002

Embedded Systems Design Project

80188/PC Programming and Software Architecture

colintan@nus.edu.sg

NUS
National University
of Singapore

School of Computing

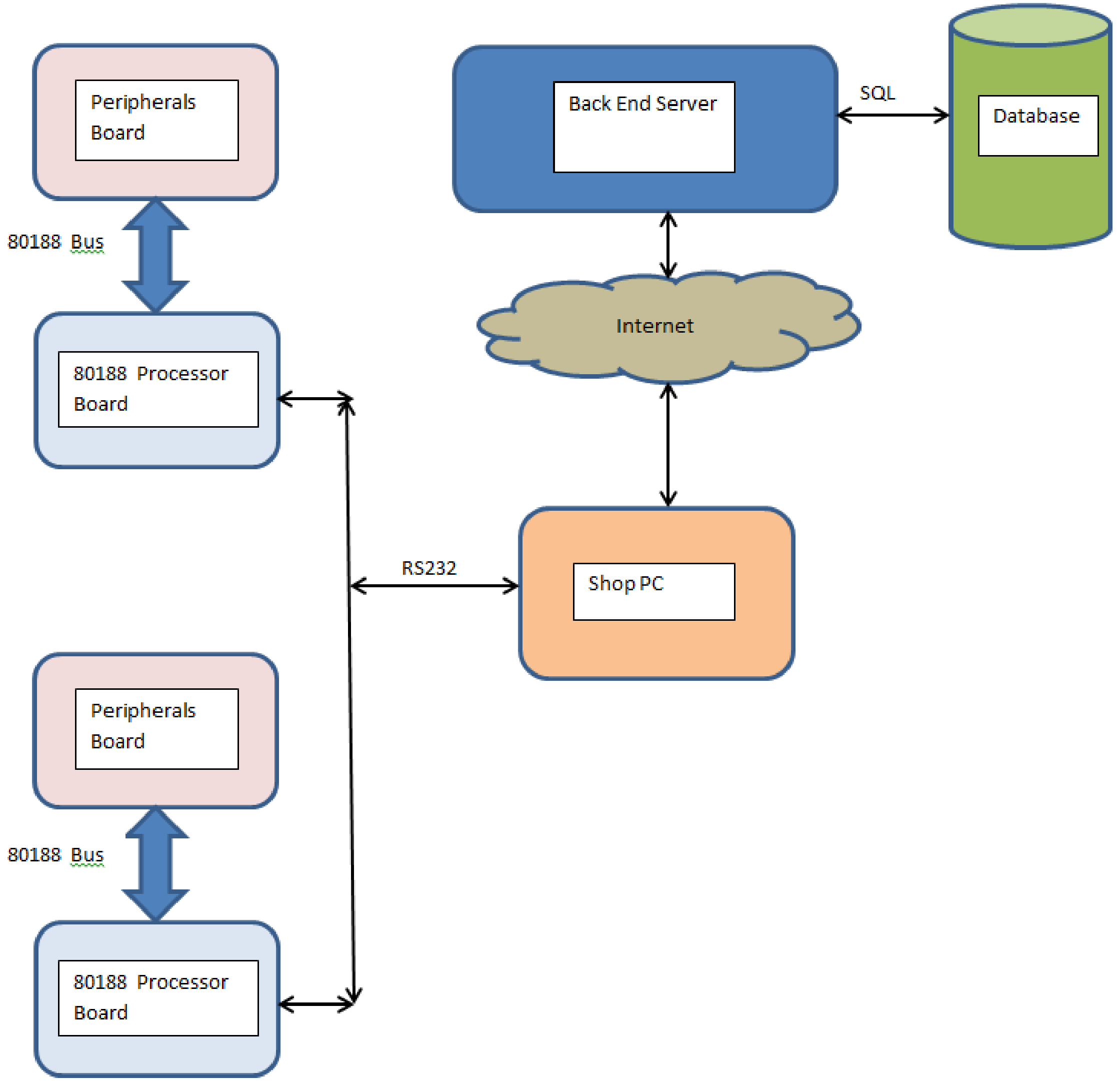
Learning Objectives

- **By the end of this lecture you will be able to:**
  - ▪Understand overall how your entire point-of-sales system (POS) integrates together.
  - ▪Understand how the 80188 system (the cash register) integrates with the store PC.
  - ▪Be able to write a program on a standard Windows PC to access the serial port.
  - ▪Use the serial port on your 80188 system to send and receive data.
  - ▪Use the timer on your 80188 system.

80188/PC Programming and Software Architecture

SYSTEM OVERVIEW

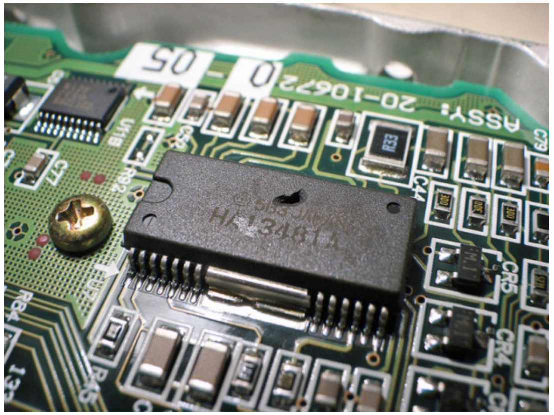
POS Overview



Essential Steps

- **User keys in the following information (and maybe more):**
  - ▪Bar code number.
  - ▪Quantity.
- **The following happens (baseline):**
  - ▪Speech synthesizer echoes what the user keys in.
    - ✓**E.g. "Product Code 12134 Quantity 3"**
  - ▪LEDs echo what the user keyed in.
  - ▪LCD shows purchase information.
  - ▪Product code and quantity are sent to the shop PC.
  - ▪The shop PC updates the back-end database.

Technical Issues



- **LEDs**
  - ▪You need to update the 6 LEDs at a rate of 50 Hz. Otherwise they will flicker.
- **Serial Port**
  - ▪Data arrives at a rate of about 960 characters per second.
  - ▪Must read within 1/960th of a second of the interrupt or data might be lost.
  - ▪We are using a multi-drop arrangement. If >1 person transmits at any time, voltage jump on the shared lines may cause magic smoke to escape from your 16C450.
    - ✓**You will need a new 16C450 if this happens.**

Technical Issues

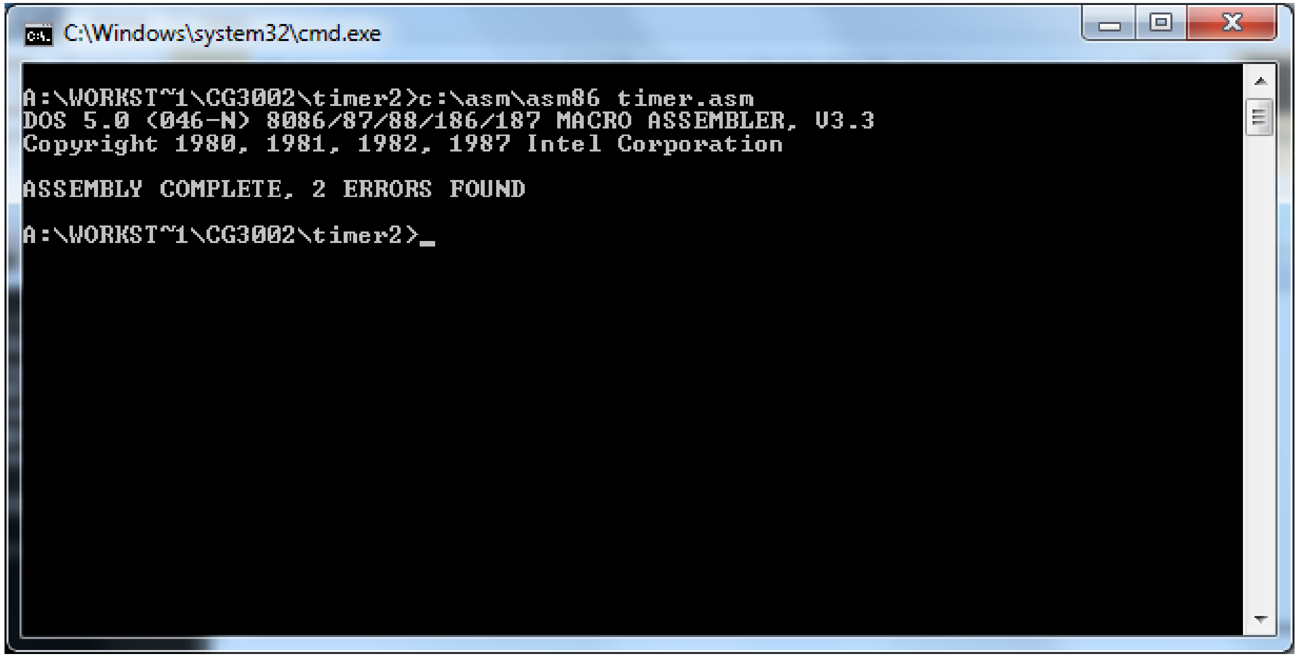
- ▪We will follow a polling arrangement.
  - ✓**Each cash register is assigned an id. E.g. 0, 1, 2, etc.**
  - ✓**PC will send an ID over the serial port. E.g. 0, 1, 2, etc.**
  - ✓**If the ID the PC sends matches your ID, send over your data. Otherwise keep quiet.**
- **Keypad**
  - ▪Interpreting X-Y coordinates into complete product codes and prices and prices might take some number of cycles.
  - ▪Always fixed format? I.e. first 6 digits is always product code, next 5 is always price?
  - ▪Or can it be more intelligent?

What We’ll Be Covering

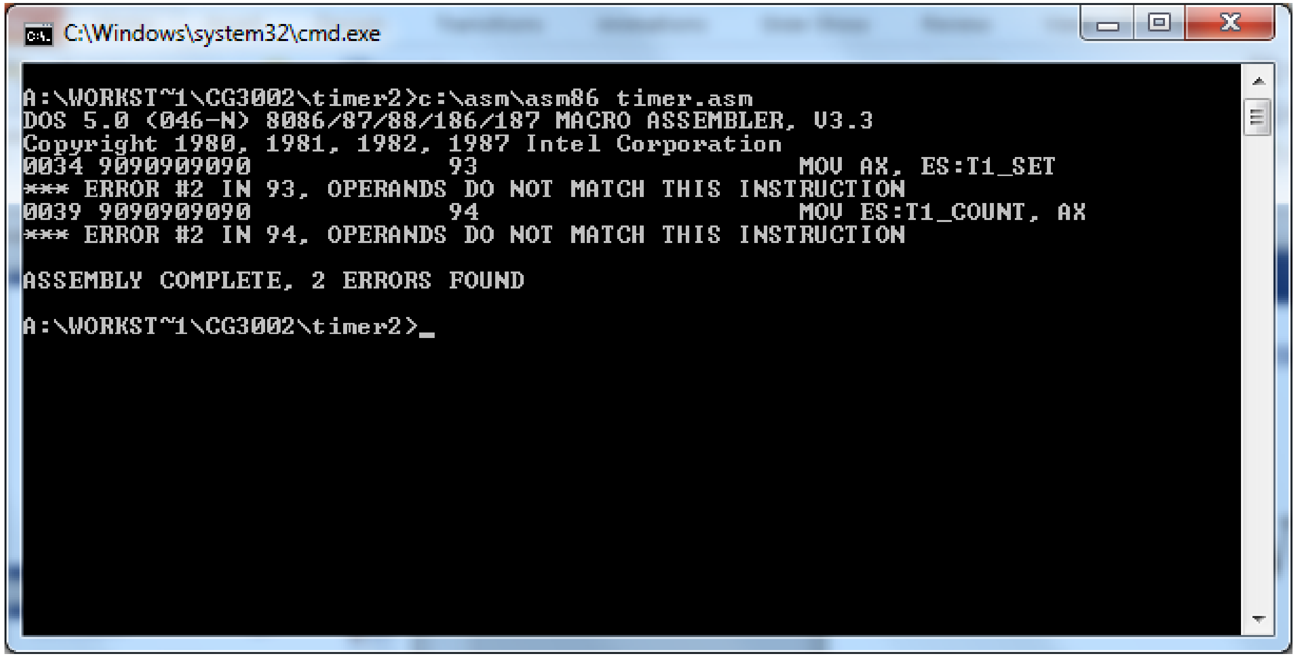
- **To help you build your project, today we’ll look at:**
  - What’s inside 80188.inc, misc.asm and timers.asm.
  - How to program the serial port on the shop PC.
  - Software architecture on the 80188.

Before We Start:

- **Before we start messing around with the assembly code, do the following:**
  - Open up TIMER.ASM and MISC.ASM
  - At the top of each file locate the line that says $mod186.
  - Immediately after that line, add in $EP
  - This causes the assembler to print error messages instead of just reporting the number of errors!



```
A:\WORKST~1\CG3002\timer2>c:\asm\asm86 timer.asm
DOS 5.0 (046-N) 8086/87/88/186/187 MACRO ASSEMBLER, V3.3
Copyright 1980, 1981, 1982, 1987 Intel Corporation

ASSEMBLY COMPLETE, 2 ERRORS FOUND

A:\WORKST~1\CG3002\timer2>_
```



```
A:\WORKST~1\CG3002\timer2>c:\asm\asm86 timer.asm
DOS 5.0 (046-N) 8086/87/88/186/187 MACRO ASSEMBLER, V3.3
Copyright 1980, 1981, 1982, 1987 Intel Corporation
0034 9090909090              93                   MOV AX, ES:T1_SET
*** ERROR #2 IN 93, OPERANDS DO NOT MATCH THIS INSTRUCTION
0039 9090909090              94                   MOV ES:T1_COUNT, AX
*** ERROR #2 IN 94, OPERANDS DO NOT MATCH THIS INSTRUCTION

ASSEMBLY COMPLETE, 2 ERRORS FOUND

A:\WORKST~1\CG3002\timer2>_
```

80188/PC Programming and Software Architecture

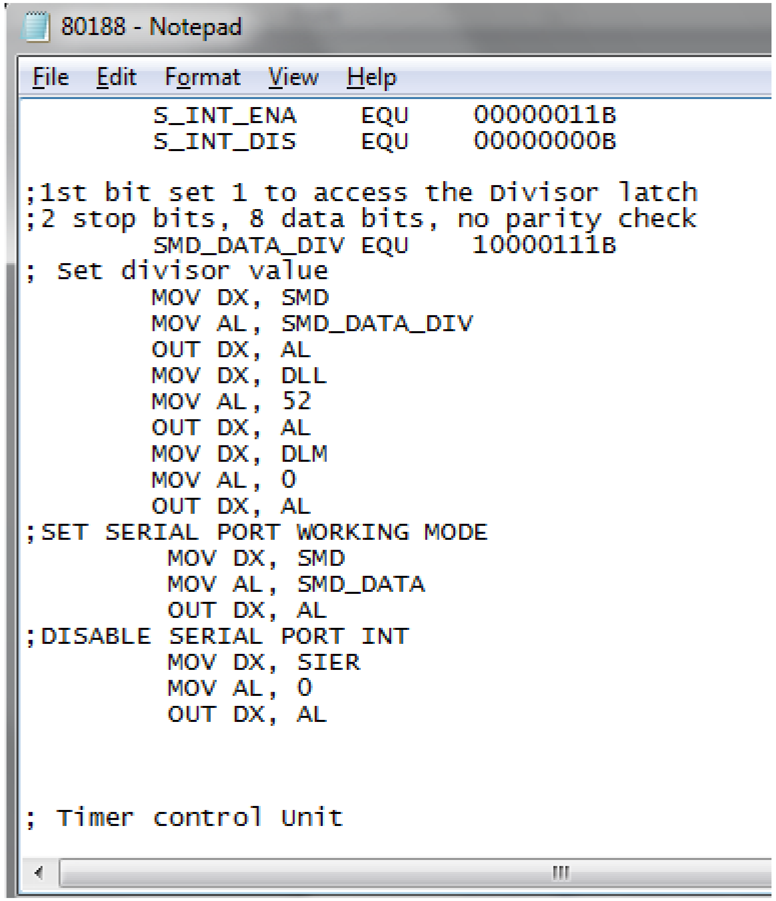
THE TIMERS TEMPLATE

The Timers Template

- **The template consists of 3 files:**
  - ▪80188.inc, containing hardware configuration register definitions and some initial register setup.
  - ▪misc.asm, containing hardware setup routines, utility routines and ISRs for the timer and serial port.
  - ▪timer.asm, where you will do most of your coding.

What’s Inside 80188.inc?

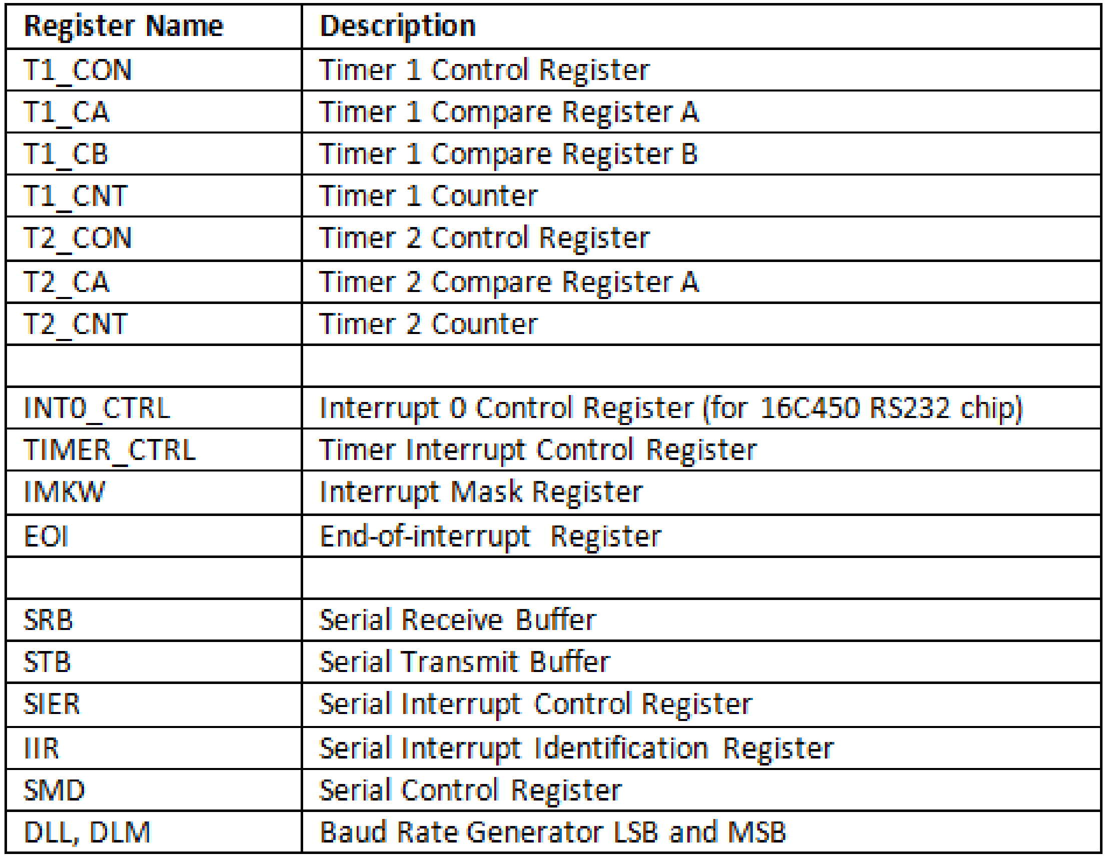
- **This file contains definitions for some of the peripheral control registers you need.**
  - Note: The control registers you need to access your LEDs, keypad, etc. are not included. You have to define these yourself.
  - It also contains some initial configurations for the peripheral chip select and the serial port.



```
        S_INT_ENA     EQU     00000011B
        S_INT_DIS     EQU     00000000B

;1st bit set 1 to access the Divisor latch
;2 stop bits, 8 data bits, no parity check
        SMD_DATA_DIV EQU     10000111B
; Set divisor value
        MOV DX, SMD
        MOV AL, SMD_DATA_DIV
        OUT DX, AL
        MOV DX, DLL
        MOV AL, 52
        OUT DX, AL
        MOV DX, DLM
        MOV AL, 0
        OUT DX, AL
;SET SERIAL PORT WORKING MODE
         MOV DX, SMD
         MOV AL, SMD_DATA
         OUT DX, AL
;DISABLE SERIAL PORT INT
         MOV DX, SIER
         MOV AL, 0
         OUT DX, AL



; Timer control Unit
```

What’s Inside 80188.inc?

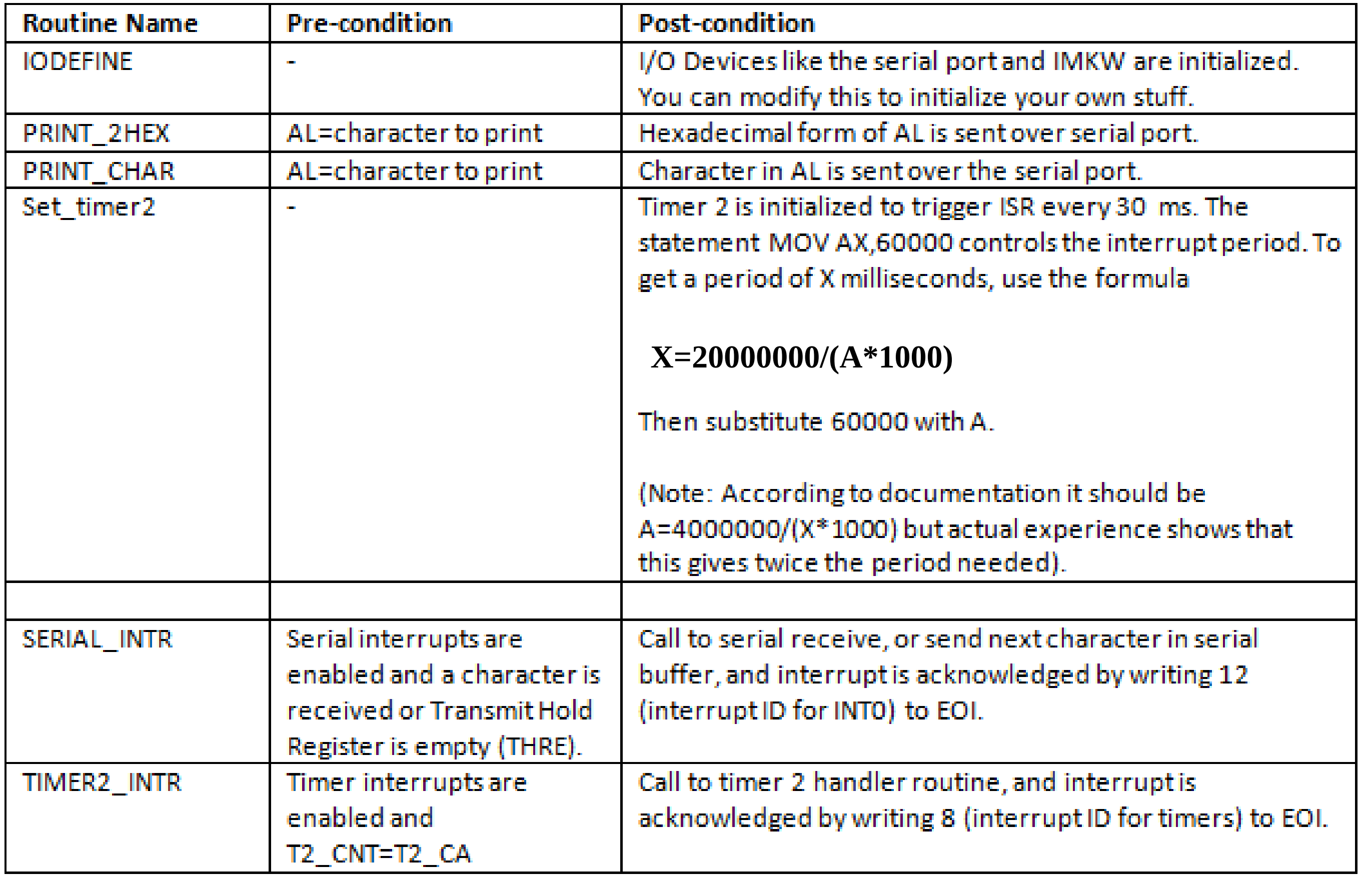
- **The more important thing for you however are the hardware configuration registers.**



Register Name	Description
T1_CON	Timer 1 Control Register
T1_CA	Timer 1 Compare Register A
T1_CB	Timer 1 Compare Register B
T1_CNT	Timer 1 Counter
T2_CON	Timer 2 Control Register
T2_CA	Timer 2 Compare Register A
T2_CNT	Timer 2 Counter
INT0_CTRL	Interrupt 0 Control Register (for 16C450 RS232 chip)
TIMER_CTRL	Timer Interrupt Control Register
IMKW	Interrupt Mask Register
EOI	End-of-interrupt Register
SRB	Serial Receive Buffer
STB	Serial Transmit Buffer
SIER	Serial Interrupt Control Register
IIR	Serial Interrupt Identification Register
SMD	Serial Control Register
DLL, DLM	Baud Rate Generator LSB and MSB

What’s Inside MISC.ASM?

- **This contains the interrupt vector table, useful routines and interrupt handlers.**



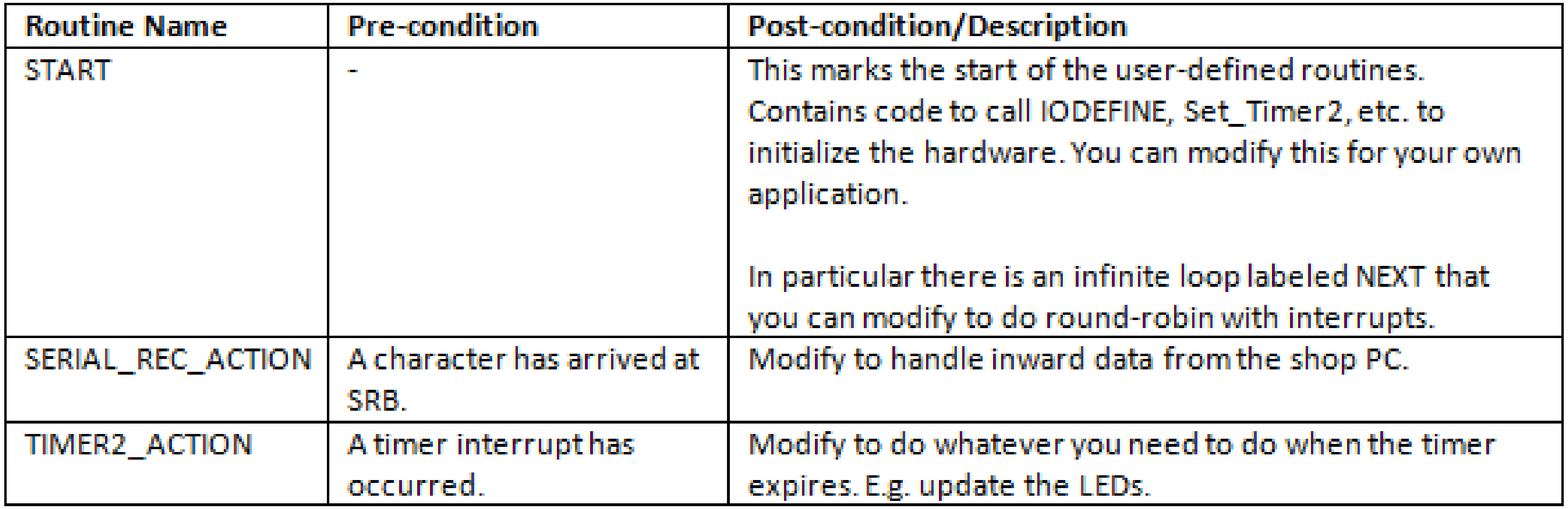
Routine Name	Pre-condition	Post-condition
IODEFINE	-	I/O Devices like the serial port and IMKW are initialized. You can modify this to initialize your own stuff.
PRINT_2HEX	AL=character to print	Hexadecimal form of AL is sent over serial port.
PRINT_CHAR	AL=character to print	Character in AL is sent over the serial port.
Set_timer2	-	Timer 2 is initialized to trigger ISR every 30 ms. The statement MOV AX,60000 controls the interrupt period. To get a period of X milliseconds, use the formula **X=20000000/(A*1000)** Then substitute 60000 with A. (Note: According to documentation it should be A=4000000/(X*1000) but actual experience shows that this gives twice the period needed).
SERIAL_INTR	Serial interrupts are enabled and a character is received or Transmit Hold Register is empty (THRE).	Call to serial receive, or send next character in serial buffer, and interrupt is acknowledged by writing 12 (interrupt ID for INT0) to EOI.
TIMER2_INTR	Timer interrupts are enabled and T2_CNT=T2_CA	Call to timer 2 handler routine, and interrupt is acknowledged by writing 8 (interrupt ID for timers) to EOI.

What’s Inside MISC.ASM?

- **Important thing about serial interrupts:**
  - ▪There is only ONE interrupt triggered by the serial port.
    - ✓**However that ONE interrupt can be caused by several things.**
    - ✓**Must look in the interrupt ID register (IIR) to see what caused the interrupt.**
  - ▪There are two we’re interested in, indicated by bits 2-0 of IIR.
    - ✓**100b = Data available at receiver.**
    - ✓**010b = Transmit holding register is empty.**
  - ▪The THRE interrupt triggers when the STB (serial transmit buffer is empty), not when you have something to transmit.

What’s Inside TIMER.ASM?

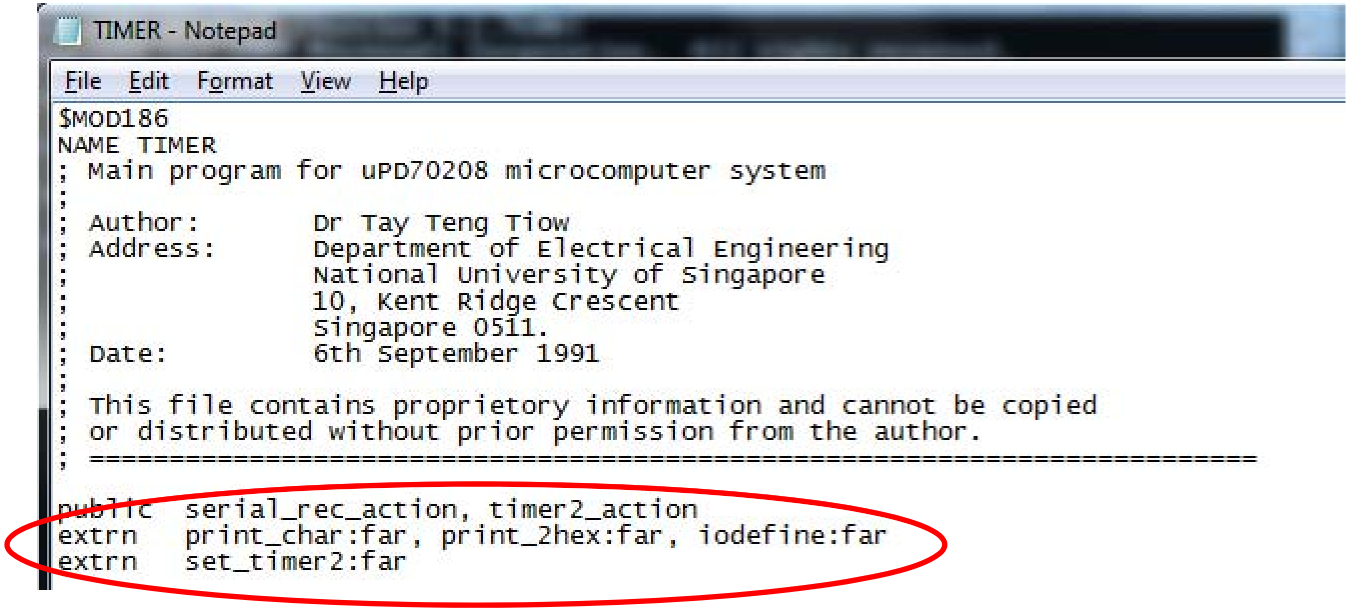
- **This file contains all the “handler routines”. You can place most of your logic inside here.**



Routine Name	Pre-condition	Post-condition/Description
START	-	This marks the start of the user-defined routines. Contains code to call IODEFINE, Set_Timer2, etc. to initialize the hardware. You can modify this for your own application. In particular there is an infinite loop labeled NEXT that you can modify to do round-robin with interrupts.
SERIAL_REC_ACTION	A character has arrived at SRB.	Modify to handle inward data from the shop PC.
TIMER2_ACTION	A timer interrupt has occurred.	Modify to do whatever you need to do when the timer expires. E.g. update the LEDs.

Importing and Exporting Routines.

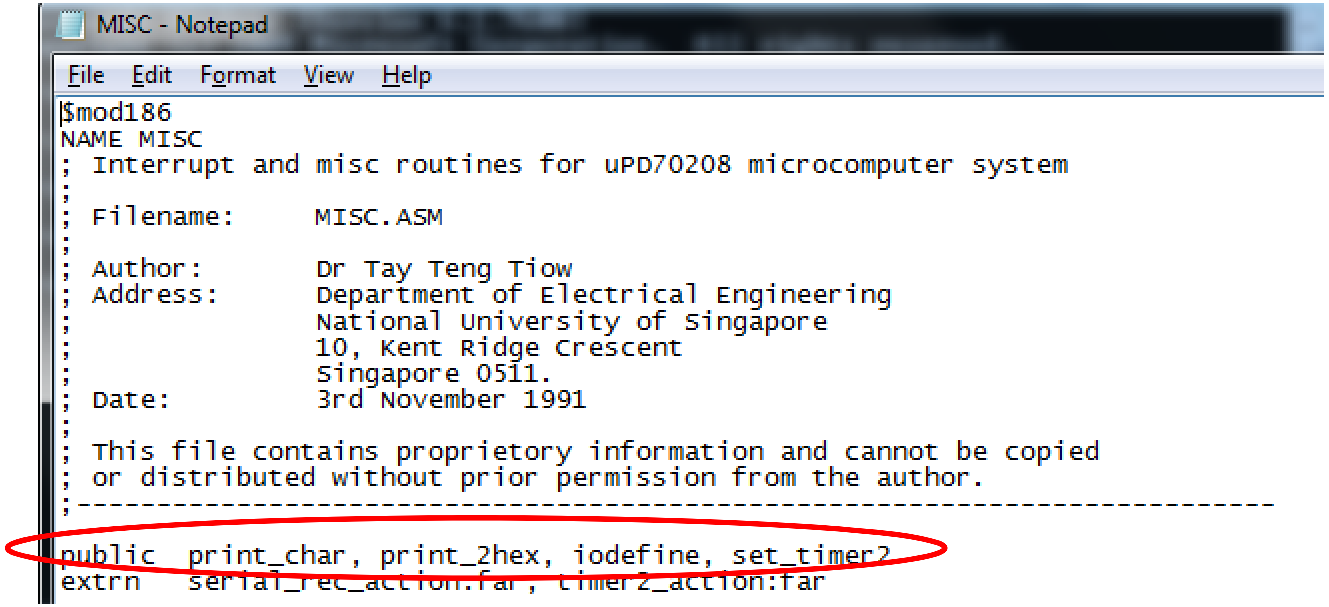
- **If you are bringing in routines from another source file, you need to import the routine using “extrn”.**
  - ▪E.g. you want to call PRINT_CHAR from within TIMER.ASM, and PRINT_CHAR is defined in MISC.ASM:



```
$MOD186
NAME TIMER
; Main program for uPD70208 microcomputer system
;
; Author:        Dr Tay Teng Tiow
; Address:       Department of Electrical Engineering
;                National University of Singapore
;                10, Kent Ridge Crescent
;                Singapore 0511.
; Date:          6th September 1991
;
; This file contains proprietory information and cannot be copied
; or distributed without prior permission from the author.
; =======================================================================

public  serial_rec_action, timer2_action
extrn   print_char:far, print_2hex:far, iodefine:far
extrn   set_timer2:far
```

Importing and Exporting Routines

- **Similarly to allow routines in other files to use your routines, you must export the routines using “public”.**
  - In our example MISC.ASM must export PRINT_CHAR.



```
$mod186
NAME MISC
; Interrupt and misc routines for uPD70208 microcomputer system
;
; Filename:     MISC.ASM
;
; Author:       Dr Tay Teng Tiow
; Address:      Department of Electrical Engineering
;               National University of Singapore
;               10, Kent Ridge Crescent
;               Singapore 0511.
; Date:         3rd November 1991
;
; This file contains proprietory information and cannot be copied
; or distributed without prior permission from the author.
;---------------------------------------------------------------------------

public  print_char, print_2hex, iodefine, set_timer2
extrn   serial_rec_action:far, timer2_action:far
```

80188/PC Programming and Software Architecture

PROGRAMMING THE PC’S SERIAL PORT

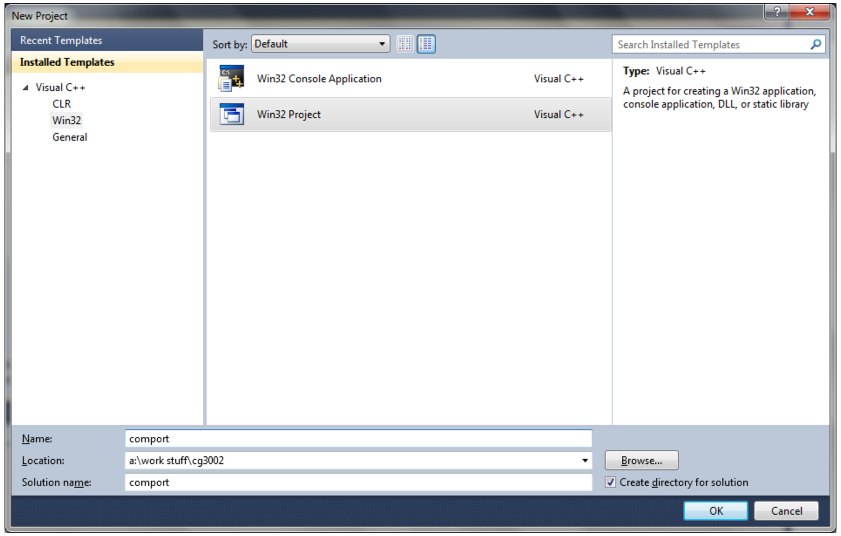
Programming the PC's Serial Port

- **The PC will poll for data from the cash registers by sending over an ID.**
- **The cash registers respond by sending back bar code and quantity information.**
- **The PC uses this information to update the back-end.**
- **So we need to know:**
  - How to access the PC's COM ports.
  - How to configure the COM ports.
  - How to send and receive data over the COM ports.

Programming the PC’s Serial Port

- You will need to download and install Microsoft Visual C++ Studio Express Edition. Available for free from Microsoft.
- Stuff in these notes work for Visual Studio 2010. Untested on Visual Studio 2012.
  - ✓**Meant primarily for Windows 8 development.**
- Create a win32 console application, but DO NOT click the “Empty Project” checkbox.
  - ✓**In fact make sure it’s not ticked.**

Programming the PC's Serial Port Creating the Project



New Project

Recent Templates

Installed Templates

- Visual C++
  - CLR
  - Win32
  - General

Sort by: Default

Search Installed Templates

Win32 Console Application	Visual C++
Win32 Project	Visual C++

Type: Visual C++

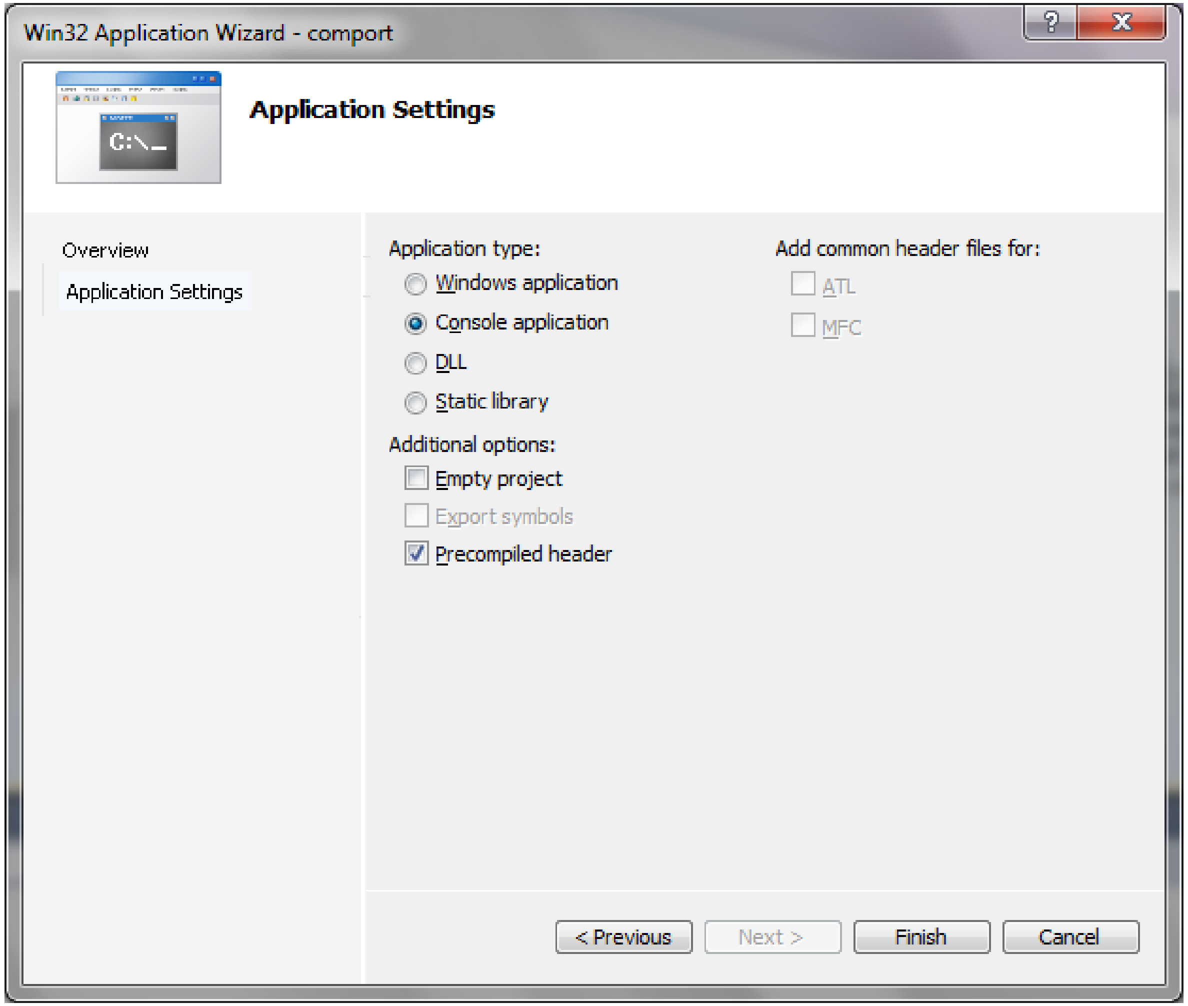
A project for creating a Win32 application, console application, DLL, or static library

Name: comport

Location: a:\work stuff\cg3002

Solution name: comport

☑ Create directory for solution



Win32 Application Wizard - comport

Application Settings

Overview

Application Settings

Application type:

○ Windows application

● Console application

○ DLL

○ Static library

Additional options:

□ Empty project

□ Export symbols

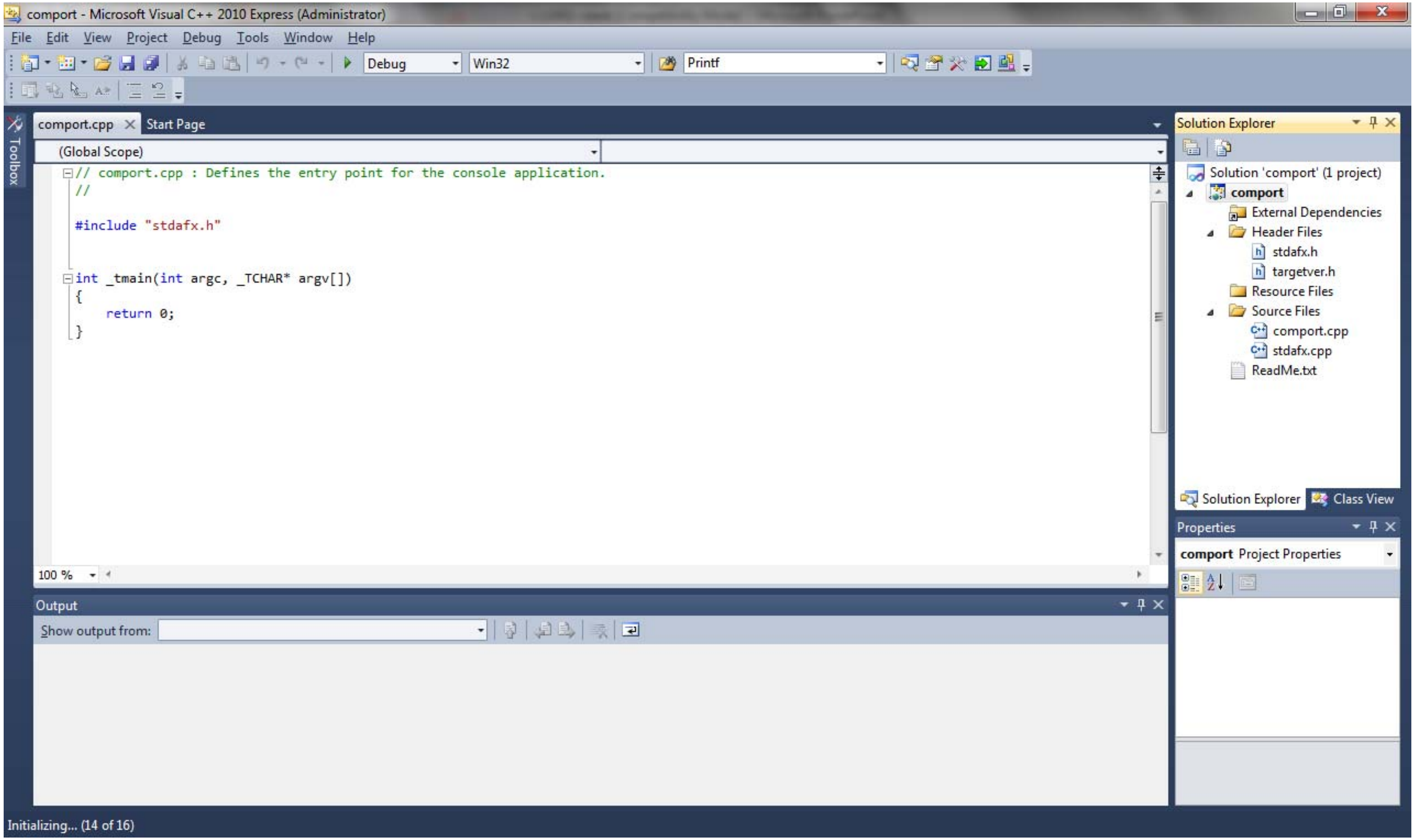
☑ Precompiled header

Add common header files for:

□ ATL

□ MFC

Programming the PC’s Serial Port
Creating the Project



```
// comport.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"


int _tmain(int argc, _TCHAR* argv[])
{
    return 0;
}
```

Programming the PC’s Serial Port

- **Steps:**
  - Open the serial port.
  - Configure it:
    - ✓**Standard configuration: 9600, no parity bits, 2 stop bits.**
  - Set a timeout:
    - ✓**This is so that we don’t wait forever for data.**
  - Read/write the serial port.
  - Close it.

Programming the PC's Serial Port
Declaring a File Handle

- **To use the serial port you need to do the following:**
  - ▪#include both stdafx.h (header file created for you by VC++) and Windows.h.
    - ✓**Windows.h contains declarations for creating Win32 programs**

```
#include "stdafx.h"
#include <Windows.h>
```

  - ▪Create a file handle of type HANDLE and an error code variable of type DWORD:

```
HANDLE hSerial;
DWORD err;
```

Programming the PC’s Serial Port
Opening the Serial Port

- **Now open the COM port using CreateFile:**

```
hSerial=CreateFile(L"COM6", GENERIC_READ | GENERIC_WRITE, 0, 0,
                 OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, 0);
```

  - Note the “L” prefix for the filename. This casts the ASCII filename “COM6” into Unicode.
    - ✓**Win32 functions do not understand ASCII!**
  - Remainder of arguments specify:
    - ✓**Open for read and write.**
    - ✓**No share and security attributes (both 0)**
    - ✓**Do not create a new file if it does not exist.**
    - ✓**File has normal attributes and no template.**

Programming the PC's Serial Port
Opening the Serial Port

- Check whether the creation was successful. Use GetLastError to get the error code if not successful.
    - ✓ **See http://msdn.microsoft.com/en-us/library/windows/desktop/ms681381(v=vs.85).aspx to understand error codes.**

```
if(hSerial==INVALID_HANDLE_VALUE)
{
        err=GetLastError();
        if(err==ERROR_FILE_NOT_FOUND)
                fprintf(stderr, "No such serial port!\n");
        else
                fprintf(stderr, "Error code %d\n", err);
        return -1;
}
```

Programming the PC’s Serial Port
Configuring the Serial Port

- Now configure the serial port by creating a “device control block”, reading the current configuration, and writing the new configuration back to the serial port:

```
LPDCB dcbSerialParams=new(DCB);

if(!GetCommState(hSerial, dcbSerialParams))
{
        fprintf(stderr, "Cannot get serial port state!\n");
        return -1;
}
```

Programming the PC’s Serial Port
Configuring the Serial Port

```
dcbSerialParams->BaudRate=CBR_9600;
dcbSerialParams->ByteSize=8;
dcbSerialParams->Parity=NOPARITY;
dcbSerialParams->StopBits=TWOSTOPBITS;

if(!SetCommState(hSerial, dcbSerialParams))
{
        fprintf(stderr, "Cannot set port state!\n");
        return -1;
}
```

Programming the PC’s Serial Port
Setting Timeout

- **Setting a timeout means that we don’t wait forever for more data to come in.**

```
LPCOMMTIMEOUTS timeouts=new(COMMTIMEOUTS);
// Maximum interval between characters
timeouts->ReadIntervalTimeout=50;
// Total timeout = ReadTotalTimeoutMultiplier * # of characters read
//      + ReadTotalTimeoutConstant.
timeouts->ReadTotalTimeoutConstant=50;
timeouts->ReadTotalTimeoutMultiplier=10;
timeouts->WriteTotalTimeoutConstant=50;
timeouts->WriteTotalTimeoutMultiplier=10;
if(!SetCommTimeouts(hSerial, timeouts))
{
        fprintf(stderr, "Cannot set timeouts!\n");
        return -1;
}
```

Programming the PC’s Serial Port
Reading and Writing the Serial Port

- **Reading and writing is done using ReadFile and WriteFile**

```
WriteFile(handle, buffer, num_of_characters_to_write,
        &num_of_characters_written, NULL);

ReadFile(handle, buffer, num_of_characters_to_read,
        &num_of_characters_read, NULL);
```

- **Both functions return 0 if you can’t read or write successfully.**

Programming the PC’s Serial Port
Reading and Writing the Serial Port

```
DWORD bytesRead;
char buffer[128]={0};

while(1)
{
        // Read in the input
        scanf("%s", buffer);
        if(!WriteFile(hSerial, buffer, strlen(buffer),
                &bytesRead, NULL))
        {
                fprintf(stderr, "Unable to write serial port\n");
        }
```

Programming the PC’s Serial Port

Reading and Writing the Serial Port

```
if(!ReadFile(hSerial, buffer, 128, &bytesRead, NULL))
{
        fprintf(stderr, "Unable to read serial port\n");
        bytesRead=0;
}
else
        if(bytesRead>0)
                printf("%s\n", buffer);
}

CloseHandle(hSerial);
```

- **The full code is given in serialcon.cpp.**

80188/PC Programming and Software Architecture

SOFTWARE ARCHITECTURE ON THE 80188

Software Architecture Basic

- **Basic architecture is essentially round-robin with interrupts.**
  - Main loop cycles through:
    - ✓**Reading the keypad.**
    - ✓**Working the speech synthesizer.**
    - ✓**Writing to the LCD.**
  - ISRs handle:
    - ✓**I/O through the serial port interrupt.**
    - ✓**Keeping the LEDs lit through the timer interrupt.**

Software Architecture
Basic

```
SERIAL_ISR:
        !! If received interrupt.
                !! Read the buffer.
                !! Append to ID_code
                !! If ID_code complete and ID_code==my_ID
                        !! Enable THRE interrupt.
                !! Else exit.
        !! If THRE interrupt.
                !! If character in buffer, send.
                !! Else disable THRE interrupt.
TIMER_ISR:
        !! If X milliseconds have passed.
                !! Update all the LEDs.
```

Software Architecture
Basic

```
Main_Loop:
        !! Read keypad.
                !! Interpret X,Y coordinates.
                !! Add digit read to item code or quantity.
                !! Echo digit read to speech synthesizer and LEDs.
                !! If item code and quantity fully read:
                        !! Write information to serial queue.
                        !! Other stuff like repeat order on speech
                                synth, etc.
        !! Update LCD
        !! Goto Main_Loop
```

Software Architecture Intermediate

- **Some enhancements:**
  - Some tasks may be slightly more timing critical:
    - ✓**E.g. reading the keypad too slowly will result in a system that doesn't respond quickly enough.**
  - You can think about moving such code into the ISR for the timer. This gives you more certainty over when things get done.

```
TIMER_ISR:
      !! If X1 milliseconds have passed
               !! Update all the LEDs.
      !! If X2 milliseconds have passed
               !! Check the keypad for button presses.
      !! If X3 milliseconds have passed
               !! Perform step 1 of interpreting the keypresses.
      !! …
```

Software Architecture
Higher Intermediate

- **You can also implement a timer-driven multi-tasking kernel for even better performance.**
  - ▪Idea:
    - ✓**Provide every task with its own stack space.**
    - ✓**This will be used to store the task's context.**
    - ✓**The timer interrupt will then switch between tasks depending on who needs to run at time X.**
  - ▪When interrupt is triggered the 80188 automatically saves the return PC onto the stack.

Software Architecture
Higher Intermediate

```
Task1_stack segment
        stack dw 256 dup (?)
Task1_stack ends

Task1   proc far
        push dx
        push ss
        mov dx, Task1_stack
        mov ss, dx
        …

Task2_stack segment
        stack dw 256 dup(?)
Task2_stack ends

Task2 proc far
        …
```

Software Architecture
Higher Intermediate

```
TIMER_ISR:
        !! Push AX, BX, CX, DX, CS, DS, ES, SI, DI, PSW onto stack.
        !! Save SP and SS onto TASKx_SS and TASKx_SP
        !! Figure out which task to restore.
        !! Load SP and SS from TASKy_SS and TASKy_SP
        !! Pop PSW, DI, SI, ES, DS, CS, DX, CX, BX and AX
        !! Return from interrupt.
```

80188/PC Programming and Software Architecture

OPEN DESIGN ISSUES

Open Design Issues

- **There are many issues that we haven't addressed to:**
  - The LCD shows purchase information like product code, quantity and price.
    - ✓**Where do you get the price from? Shop PC?**
  - What information is exchanged between the shop PC and the back-end?
  - How do you decide re-stock levels in the shops?
  - Do you assume that stock is always available? What if the shop cannot fulfill an order keyed in?
    - ✓**How do you coordinate between the PC and cash register?**
- **You might need to solve these issues in your system.**

Next Two Weeks

- **I will randomly pick 11 groups each week to:**
  - Give a 5-7 minute presentation.
  - Have 3 minutes Q&A.
- **What you should talk about:**
  - An overview of your system, including interaction with shop PC and backend server.
    - ✓**Address the open issues here as well.**
  - The software architecture(s) you intend to choose on the cash register and price tag.
  - What are the timing constraints you will face? Show an analysis.
  - How will you guarantee the timing constraints?