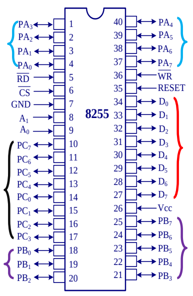
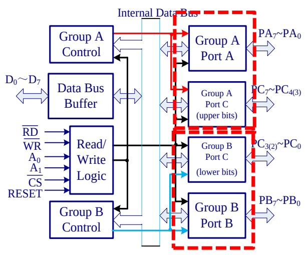
Lecture 07: 8255 PPI Chip

Package & Internal Structure





Internal Structure and Pins

Three data ports: A, B, and C

- Port A (PA_0~PA_7): can be programmed **all** as input/output
- Port B (PB_0~PB_7): can be programmed **all** as input/output
- Port C (PC_0~PC_7): can be split into two separate parts *PCU* and *PCL*; any bit can be programmed **individually**

Control register (CR)

- Internal register: used to setup the chip

Group A, Group B and control logic

- Group A (PA & PCU)
- Group B (PB & PCL)

Internal Structure and Pins

Data bus buffer

- An interface between CPU and 8255
- Bidirectional, tri-state, 8-bit

Read/Write control logic

- Internal and external control signals
- **RESET**: high-active, clear the control register, all ports are set as input port
- **~CS**, **~RD**, **~WR**
- **A_1, A_0:** port selection signals

~CS	A_1	A_0	~RD	~WR	Function
0	0	0	0	1	PA->Data bus
0	0	1	0	1	PB->Data bus
0	1	0	0	1	PC->Data bus
0	0	0	1	0	Data bus->PA
0	0	1	1	0	Data bus->PB
0	1	0	1	0	Data bus->PC
0	1	1	1	0	Data bus->CR
1	×	×	1	1	D_0~D_7 in float

Operation Modes

Input/output modes

- Mode 0, simple I/O mode:
  - **PA**, **PB**, **PC**: PCU{PC_4~PC_7}, PCL{PC_0~PC_3}
  - No Handshaking: *negotiation between two entities before communication*
  - Each port can be programmed as input/output port
- Mode 1:
  - **PA**, **PB** can be used as input/output ports with *handshaking*
  - PCU{PC_3~PC_7}, PCL{PC_0~PC_2} are used as handshake lines for PA and PB, respectively
- Mode 2:
  - Only **PA** can be used for *bidirectional handshake* data transfer
  - PCU{PC_3~PC_7} are used as handshake lines for PA

Bit set/reset (BSR) mode

- Only **PC** can be used as output port
- Each line of PC can be set/reset individually

Control Register & Op. Modes

Control Register

- A 8-bit internal register in 8255
- Selected when A_1=1, A_0=1
- Mode selection word
  - Input/output modes

1	D_6	D_5	D_4	D_3	D_2	D_1	D_0

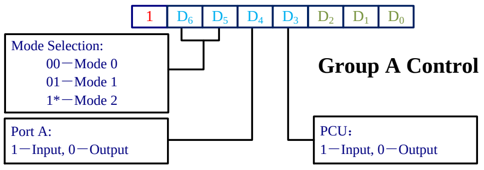
$\underbrace{D_6\ D_5\ D_4\ D_3}_{\text{Group A control}}\ \underbrace{D_2\ D_1\ D_0}_{\text{Group B control}}$

  - BSR mode

0	D_6	D_5	D_4	D_3	D_2	D_1	D_0

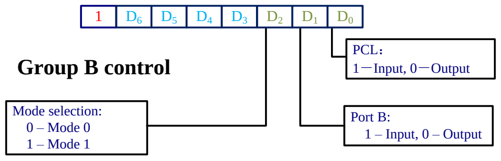
Select Input/output Modes

Input/output modes



Select Input/output Modes

Input/output modes



Select Input/output Mode Examples

1. Write ASM instructions for setting the 8255 in simple I/O mode with PA and PB being output port and PC being input port.

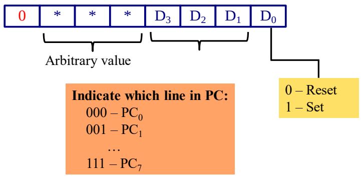
```
MOV  AL, 10001001B
MOV  DX, ContralPort
OUT  DX, AL
```

2. Assume that the address of the control register of the 8255 is 63H, give out the instructions that set up the 8255 in mode 0 where PA, PB and PCU are used as input ports and PCL is used as output port.

```
MOV  AL, 10011010B
OUT   63H, AL
```

Select BSR Mode

BSR Mode



Select BSR Mode Examples

Assume PC is used as an output port which connects to 8 LED segments, now turn off the second LED segment with the rest unchanged (e.g., for LED segments in this case, 1-on, 0-off).

```
MOV AL, 10000000
OUT  ContralPort, AL
…
IN  AL, CPORT
AND  AL, 11111101B
OUT   CPORT, AL
```

Using BSR mode instead:

```
MOV  AL, 00000010B
OUT   ControlPORT, AL
```

Select BSR Mode Examples

Assume the address range for a 8255 is 60H~63H, PC_5 is outputting a low level, write code to generate a positive pulse.

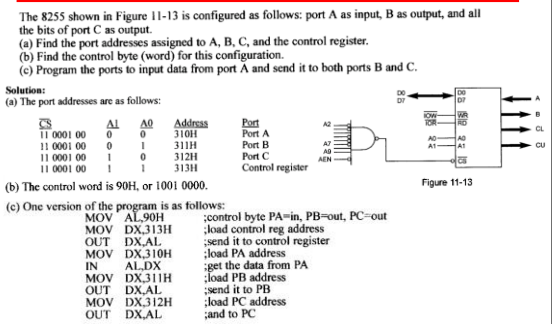
```
MOV  AL, 00001011B     ; set PC5 high level
OUT   63H, AL
MOV  AL, 00001010B     ; set PC5 low level
OUT   63H, AL
```

Mode 0 (Simple I/O)

For simple input/output scenario
- No handshaking needed
- Any port of PA, PB and PC (PCU, PCL) can be programmed as input or output port independently
- PCU=PC_4~PC_7, PCL=PC_0~PC_3
- CPU directly read from or write to a port using IN and OUT instructions
- Input data are **not** latched, Output data are latched
- E.g., setting up the control register for Mode 0

D_7=1	0	0	*	*	0	*	*

Mode 0 Example



The 8255 shown in Figure 11-13 is configured as follows: port A as input, B as output, and all the bits of port C as output.
(a) Find the port addresses assigned to A, B, C, and the control register.
(b) Find the control byte (word) for this configuration.
(c) Program the ports to input data from port A and send it to both ports B and C.

Solution:
(a) The port addresses are as follows:

$\overline{CS}$	A1	A0	Address	Port
11 0001 00	0	0	310H	Port A
11 0001 00	0	1	311H	Port B
11 0001 00	1	0	312H	Port C
11 0001 00	1	1	313H	Control register

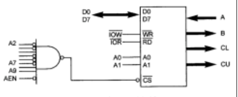


Figure 11-13

(b) The control word is 90H, or 1001 0000.

(c) One version of the program is as follows:
```
MOV   AL,90H       ;control byte PA=in, PB=out, PC=out
MOV   DX,313H      ;load control reg address
OUT   DX,AL        ;send it to control register
MOV   DX,310H      ;load PA address
IN    AL,DX        ;get the data from PA
MOV   DX,311H      ;load PB address
OUT   DX,AL        ;send it to PB
MOV   DX,312H      ;load PC address
OUT   DX,AL        ;and to PC
```

Mode 1 (Strobe I/O)

For handshake input/output scenario
- PA and PB can be used as input or output ports
- PCU=PC_3~PC_7, used as handshake lines for PA
- PCL=PC_0~PC_2, used as handshake lines for PB
- Both input and output data are latched

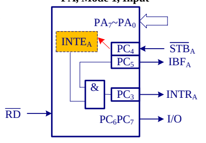
Mode 1: As Input Ports

PC_3~PC_5 and PC_0~PC_2 are used as handshake lines for PA and PB, respectively
- **~STB:** the strobe *input* signal from input device loads data into the port latch
- **IBF:** Input Buffer Full *output* signal to the device indicates that the input latch contains information (can also be used for programmed I/O)
- **INTR:** Interrupt request is an output to CPU that requests an interrupts (for interrupted I/O)

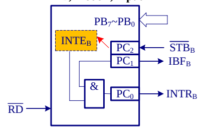
PC_6 and **PC_7** can be used as separate I/O lines for any purpose

INTE: the interrupt enable signal is neither an input nor an output; it is an internal bit programmed via the PC_4 (port A) or PC_2 (port B); 1-allowed, 0-forbidden

PA, Mode 1, Input

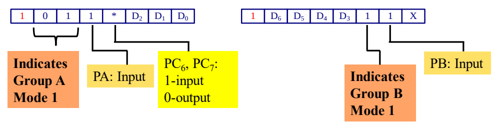


PB, Mode 1, Input

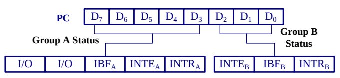


Mode 1: As Input Ports

Control register



PC stores all status information



Timing in Mode 1 Input

- Input device first puts data on PA_0~PA_7, then activates ~STB_A, data is latched in Port A;
- 8255 activates IBF_A which indicates that the device that the input latch contains information but CPU has not taken it yet. So device cannot send new data until IBF_A is cleared;
- When IBF_A, ~STB_A and $INTE_A$ are all high, 8255 activates $INTR_A$ to inform CPU to take data in PA by interruption;
- CPU responds to the interruption and read in data from PA; the ~RD signal will clear $INTR_A$ signal;
- After CPU finishes reading data from PA (i.e., ~RD signal goes high), the IBF_A signal is cleared.

Mode 1: As Output Ports

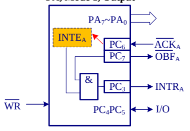
PC_3, PC_6, PC_7 and PC_0~PC_2 are used as handshake lines for PA and PB, respectively

- **~OBF** : Output buffer full is an *output* signal that indicates the data has been latched in the port
- **~ACK** : The acknowledge *input* signal indicates that the external device has taken the data
- **INTR:** Interrupt request is an output to CPU that requests an interrupts

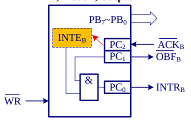
PC_4 and **PC_5** can be used as separate I/O lines for any purpose

INTE: the interrupt enable signal is neither an input nor an output; it is an internal bit programmed via the PC_6 (port A) or PC_2 (port B); 1-allowed, 0-forbidden

PA, Mode 1, Output

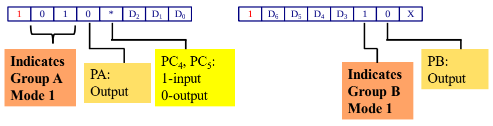


PB, Mode 1, Output

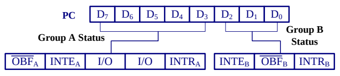


Mode 1: As Output Ports

Control register



PC stores all status information



Timing in Mode 1 Output

If $INTR_A$ active, CPU responds to the interruption and writes data to PA and clears the $INTR_A$ signal;

When data has been latched in PA, 8255 activates $\sim OBF_A$ which informs the output device to pick up data;

After the output device has taken the data, it sends $\sim ACK_A$ signal to 8255 which indicates that the device has received the data, and also makes $\sim OBF_A$ go high, indicating CPU can write new data to 8255;

When $\sim OBF_A$, $\sim ACK_A$ and $INTE_A$ are all high, 8255 sends an $INTR_A$ to inform CPU to write new data to PA by interruption.

Mode 2 (Bidirectional Bus)

For bidirectional handshake input/output scenario

- Only PA can be used as *both input and output port*
- PCU=PC_3~PC_7, used as handshake lines for PA
- Both input and output data are latched

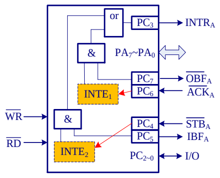
Mode 2: As Input & Output Port

PC_3 ~ PC_7 are used as handshake lines for PA

$\sim OBF_A$, $\sim ACK$, IBF_A, $\sim STB_A$, $INTR_A$

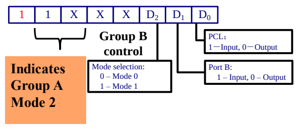
PC_0 ~ **PC_2** can be used as separate I/O lines for any purpose, or as handshake lines for PB

When CPU responds to an interrupt of 8255 working in Mode 2, it has to check the **$\sim OBF_A$** and **IBF_A** in order to tell whether the input process or the output process is generating the interrupt.

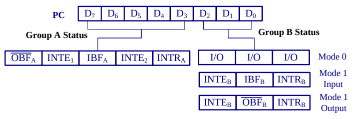


Mode 2: As Input & Output Port

Control register



PC stores all status information



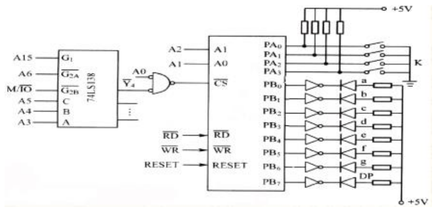
Polling vs. Interruptions

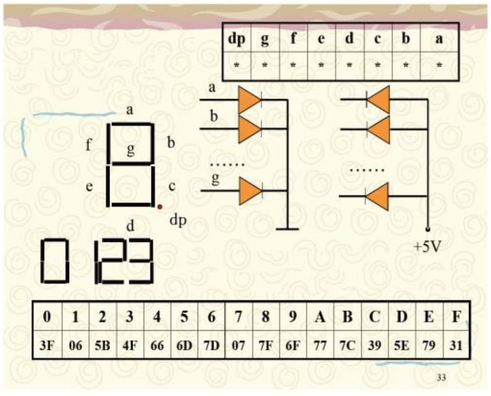
In Mode 1 and 2, PC stores status of Group A and/or Group B

By reading from PC using **IN** instruction, you can use polling method to check the state of I/O devices

Programming with 8255

As shown in the figure, PA and PB of the 8255 are working in mode 0. PA used as input port connects to 4 switches, and PB used as output port connects to a 7-segment LED. Write a program to display a hex digit that the switches can represent.



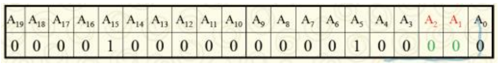


dp	g	f	e	d	c	b	a
*	*	*	*	*	*	*	*

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
3F	06	5B	4F	66	6D	7D	07	7F	6F	77	7C	39	5E	79	31

Address Decoding

What are the addresses of ports and the control register?



A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0

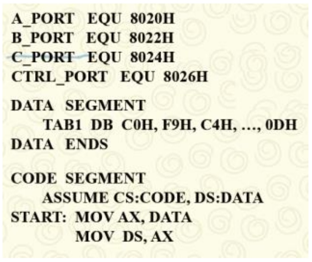
PA : 8020H

PB : ?

PC : ?

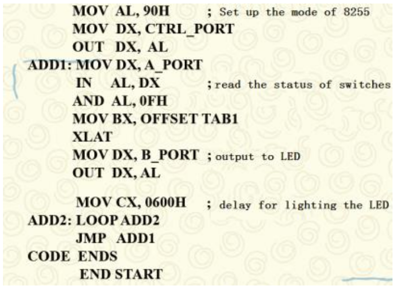
CR : ?

A Solution Program



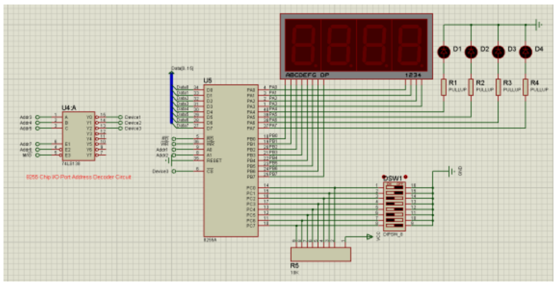
```
A_PORT   EQU  8020H
B_PORT   EQU  8022H
C_PORT   EQU  8024H
CTRL_PORT  EQU  8026H

DATA  SEGMENT
     TAB1  DB  C0H, F9H, C4H, ..., 0DH
DATA  ENDS

CODE  SEGMENT
     ASSUME CS:CODE, DS:DATA
START:  MOV AX, DATA
        MOV  DS, AX
```

A Solution Program



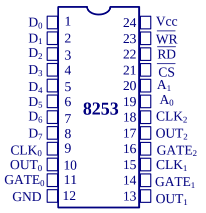
```
       MOV  AL, 90H       ; Set up the mode of 8255
       MOV  DX, CTRL_PORT
       OUT  DX, AL
ADD1: MOV DX, A_PORT
       IN   AL, DX        ;read the status of switches
       AND AL, 0FH
       MOV BX, OFFSET TAB1
       XLAT
       MOV DX, B_PORT  ; output to LED
       OUT  DX, AL

       MOV CX, 0600H     ; delay for lighting the LED
ADD2: LOOP ADD2
       JMP  ADD1
CODE  ENDS
       END START
```

Experiment 2



Lecture 08: 8253/4 Timer

Package & Internal Structure



The 8253/54 Programmable interval timer is used to generate a lower frequency for various uses

e.g.,

Event counter

Accurate time delays

Software:

setting up a timing loop

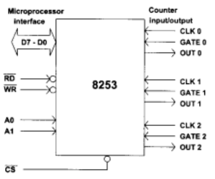
```
      MOV  CX, N
AGAIN: LOOP AGAIN
```

Hardware:

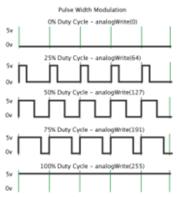
using 8253 to count out the delay and interrupt the CPU

Pros and cons?

Interface to the System

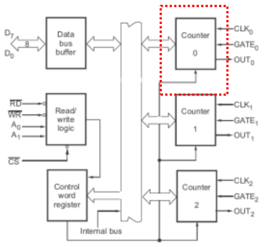


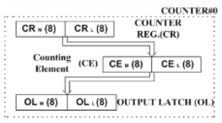
- ➢There are three independent counters.
- ➢The input frequency can be divided from 1 to 65536 (Binary), or from 1 to 10000 (BCD)
- ➢Shape of the output frequency:
  - ❖Square-wave
  - ❖One-shot
  - ❖Square-wave with various duty cycles.



- ➢Gate is used to enable (High) or disable (Low) the counter.
- ➢If bidirectional bus D0-D7 is connected to D0-D7 of the system bus, even addresses in 8086 system.

Internal Structure





To operate a 16-bit down counter

❖a 16-bit count is loaded in the counter

❖begins to decrement the count until it reaches 0

❖generates a pulse that can be used to interrupt the CPU

Features

Three independent 16-bit down counters

8254 can handle inputs from DC to 10 MHz (5MHz 8254-5 8MHz 8254 10MHz 8254-2) whereas 8253 can operate up to 2.6 MHz

Three counters are identical and pre-settable, and can be programmed for either binary or BCD count

Counter can be programmed in six different modes

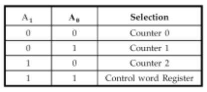
Compatible with all Intel and most other microprocessors

8254 has powerful command called READ BACK command which allows the user to check the count value, programmed mode and current mode and current status of the counter

Internal Structure & Pins

Data bus buffer

- interface the 8253/4 to the system data bus
- Bi-directional, tri-state, 8-bit



A_1	A_0	Selection
0	0	Counter 0
0	1	Counter 1
1	0	Counter 2
1	1	Control word Register

Read/Write control logic

- **~CS**
  - Tied to a decoded address
- **~RD**, **~WR**
  - In isolated I/O: ~IOR, ~IOW
  - Memory-mapped I/O: ~MEMR, ~MEMW
- **A_1, A_0**
  - Select the control word register and counters
  - usually connected to address lines A_1, A_0

/CS	/RD	/WR	A1A0	FUNCTION
0	1	0	00	Write counter0 (to CR0)
0	1	0	01	Write counter1 (to CR1)
0	1	0	10	Write counter2 (to CR2)
0	1	0	11	Write control port
0	0	1	00	Read counter0 (from OL0)
0	0	1	01	Read counter1 (from OL1)
0	0	1	10	Read counter2 (from OL2)
0	0	1	11	Read control port (for *8254*)
1	X	X	XX	Not available

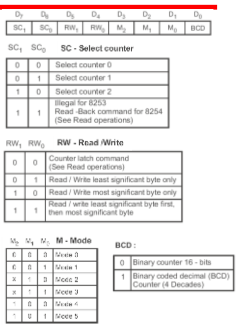
Internal Structure & Pins

Control Word Register:

- Selected when A_1=1, A_0=1
- Used to specify which counter to be used, its mode, and a read or write operation

Counters:

- Each consists of a single, 16-bit, pre-settable, down counter
- Can operate in either binary or BCD
- Input, gate and output are configured by the selection of modes
- Reading from a counter does not disturb the actual count in process



D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SC_1	SC_0	RW_1	RW_0	M_2	M_1	M_0	BCD

SC_1	SC_0	SC - Select counter
0	0	Select counter 0
0	1	Select counter 1
1	0	Select counter 2
1	1	Illegal for 8253 Read -Back command for 8254 (See Read operations)

RW_1	RW_0	RW - Read /Write
0	0	Counter latch command (See Read operations)
0	1	Read / Write least significant byte only
1	0	Read / Write most significant byte only
1	1	Read / write least significant byte first, then most significant byte

M_2	M_1	M_0	M - Mode
0	0	0	Mode 0
0	0	1	Mode 1
x	1	0	Mode 2
x	1	1	Mode 3
1	0	0	Mode 4
1	0	1	Mode 5

BCD :	
0	Binary counter 16 - bits
1	Binary coded decimal (BCD) Counter (4 Decades)

Write/Read Operations

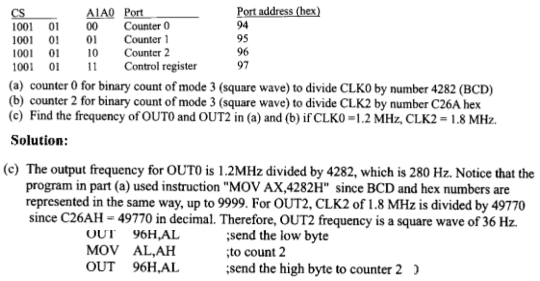
WRITE:

- Write a control word into control register
- Load the low-order byte of a count in the counter register
- Load the high-order byte of a count in the counter register

READ:

- **Simple Read**: two I/O read operations, first one for low-order byte and last one for the high order byte
- **Counter Latch Command**: one I/O write operation used to write a control word to the control register to latch a count in the output latch, then two I/O read operations are used to read the latched count as in Simple Read.
- **Read-Back Command**: for 8254 only

Example: Setting Up a Counter



CS		A1A0	Port	Port address (hex)
1001	01	00	Counter 0	94
1001	01	01	Counter 1	95
1001	01	10	Counter 2	96
1001	01	11	Control register	97

(a) counter 0 for binary count of mode 3 (square wave) to divide CLK0 by number 4282 (BCD)
(b) counter 2 for binary count of mode 3 (square wave) to divide CLK2 by number C26A hex
(c) Find the frequency of OUT0 and OUT2 in (a) and (b) if CLK0 =1.2 MHz, CLK2 = 1.8 MHz.

Solution:

(c) The output frequency for OUT0 is 1.2MHz divided by 4282, which is 280 Hz. Notice that the program in part (a) used instruction "MOV AX,4282H" since BCD and hex numbers are represented in the same way, up to 9999. For OUT2, CLK2 of 1.8 MHz is divided by 49770 since C26AH = 49770 in decimal. Therefore, OUT2 frequency is a square wave of 36 Hz.

```
OUT    96H,AL          ;send the low byte
MOV    AL,AH           ;to count 2
OUT    96H,AL          ;send the high byte to counter 2   )
```

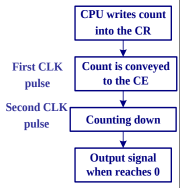
Features of 8253

- 8253 takes one CLK pulse to convey the count from CR to CE
- CE will start to count only when GATE = 1
  - *When to check the GATE?*
- On every CLK pulse's rising (0-to-1) edge
  - *When to count down?*
- On every CLK pulse's falling (1-to-0) edge

Mode 0 : Interrupt on Terminal Count (1)

Normal Operation:

- The output will be initially **low** after the mode set operation;
- After the count is loaded into the selected CR the output will remain **low**
- When the terminal count is reached, the output will go **high** and remain high until the selected counter is reloaded
- Output: N clock pulses low and high afterwards after writing a count

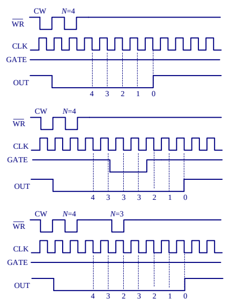

CPU writes count into the CR
First CLK pulse
Count is conveyed to the CE
Second CLK pulse
Counting down
Output signal when reaches 0

Mode 0 : Interrupt on Terminal Count (2)

Gate disable:
- Gate = 1 enables counting
- Gate = 0 disables counting

New count:
- If a new count is written to the counter, it will be loaded on the next CLK pulse and counting will continue from the new count
- In case of two byte count:
  - Writing the first byte disables the current counting
  - Writing the second byte loads the new count on the next CLK pulse and counting will continue from the new count

Mode 0: Interrupt on Terminal Count (3)



When loading a new count *N*, the actual number of CLK pulses in OUT is *N*+1

Does not automatically repeat

Mode 1: Hardware Retriggerable One-shot (1)

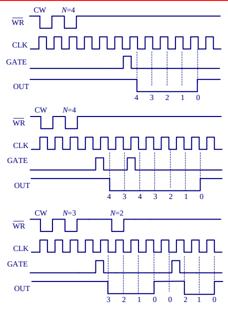
Normal Operation:
- The output will be initially **high** after the mode set operation;
- The output will go **low** on the CLK pulse following the rising (0-to-1) edge of the gate input;
- The output will go **high** on the terminal count and remain high until the next rising edge of the gate input.
- **Output: one-shot of N clock pulses on every trigger**

Mode 1 : Hardware Retriggerable One-shot (2)

Retriggering:
- retriggerable, hence the output will remain low for the full count after any rising edge of the gate input

New count:
- If the counter is loaded during one shot pulse, the current one shot is not affected unless the counter is retriggered
- If retriggered, the counter is loaded with the new count and the one-shot pulse continues until the new count expires

Mode 1 : Hardware Retriggerable One-shot (3)



When loading a new count *N*, the current counting will not be affected

Does not automatically repeat

Mode 2: Rate Generator (1)

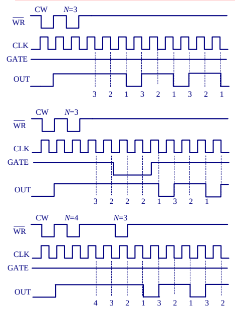
Normal Operation:
- The output will be initially **high**;
- The output will go **low** for one clock pulse before the terminal count;
- The output then goes **high**, the counter reloads the initial count and the process is repeated
- **Output: periodical signal with a period of N-1 clock pulses high and 1 clock pulse low**

Mode 2: Rate Generator (2)

Gate disable:
- If Gate=1 it enables a counting otherwise it disables counting (Gate=0)
- If Gate goes low during an low output pulse, output is set immediately high

New count:
- The current counting sequence is not affected when the new count is written
- If a trigger (a rising edge of GATE) is received after writing a new count but before the end of the current period, the new count will be loaded with the new count on the next CLK pulse and counting will continue from the new count
- Otherwise, the new count will be loaded at the end of the current counting cycle
- **Note : In mode 2, a count of 1 is illegal.** ***Why?***

Mode 2: Rate Generator (3)

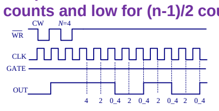


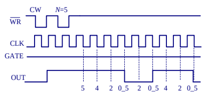
When loading a new count *N*, the current counting will not be affected

Automatically repeat on terminal count

Mode 3: Square Wave Rate Generator (1)

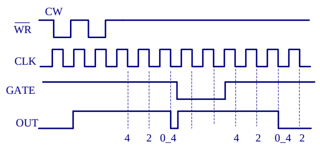
Normal Operation:
- The output will be initially high;
- For even count, counter is decremented by 2 on the falling edge of each clock pulse; when reaches terminal count, the state of the output is changed and the counter is reloaded with the full count and the whole process is repeated
- For odd count, the first clock pulse decrements the count by 1. Subsequent clock pulses decrement the clock by 2. After timeout, the output goes low and the full count is reloaded. The first clock pulse (following the reload) decrements the count by 3 and subsequent clock pulse decrement the count by two. Then the whole process is repeated.
- **Output: if the count is odd, the output will be high for (n+1)/2 counts and low for (n-1)/2 counts.**





Mode 3: Square Wave Rate Generator (2)

Gate disable:
- If Gate is 1 counting is enabled otherwise it is disabled.
- If Gate goes low while output is low, output is set high immediately. After this, When Gate goes high, the counter is loaded with the initial count on the next clock pulse and the sequence is repeated.



Mode 3: Square Wave Rate Generator (3)

New count:
- The current counting sequence does not affect when the new count is written.
- If a trigger is received after writing a new count but before the end of the current half-cycle of the square wave, the counter will be loaded with the new count on the next CLK pulse and counting will continue from the new count.
- Otherwise, the new count will be loaded at end of the current half-cycle.

When loading a new count *N*, the current half will not be affected

Automatically repeat on terminal count

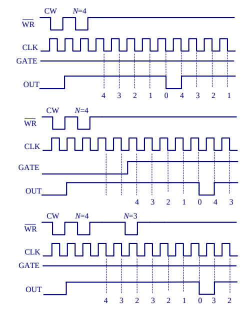
Mode 4: Software Triggered Strobe (1)

Normal Operation:
- The output will be initially **high**;
- The output will go **low** for one CLK pulse after the terminal count

Gate disable:
- If Gate is one, the counting is enabled; otherwise, it is disabled

New count:
- If a new count is written during counting, it will be loaded on the next CLK pulse and counting will continue from the new count. If the count is two byte then:
  - Writing the first byte has no effect on counting
  - Writing the second byte allows the new count to be loaded on the next CLK pulse

Mode 4: Software Triggered Strobe (2)



When loading a new count *N*, the actual number of CLK pulses in OUT is *N*+1

Automatically repeat

Mode 5: Hardware Triggered Strobe (Retriggerable) (1)

Normal Operation:

The output will be initially **high**;

The counting is triggered by the rising edge of the Gate

The output will go **low** for one CLK pulse after the terminal count

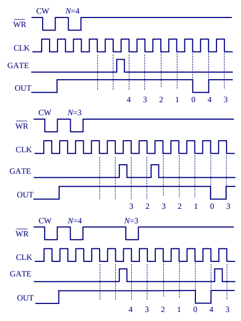
Retriggering:

If the triggering occurs during the counting, the initial count is loaded on the next CLK pulse and the counting will be continued until the terminal count is reached

New count:

the current counting sequence will not be affected. If the trigger occurs after the new count but before the terminal count, the counter will be loaded with the new count on the next CLK pulse and counting will continue from there

Mode 5: Hardware Triggered Strobe (Retriggerable) (2)



When loading a new count *N*, the current counting will not be affected

Automatically repeat on terminal count

Programming Example

Example 1: *Write a program to initialize counter 2 in mode 0 with a count of C030H. Assume address for control register = 0BH, counter 0 = 08H, counter 1 = 09H and counter 2 = 0AH.*

Sol. : Control word

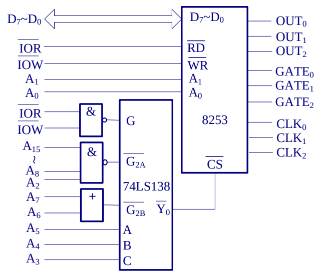
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0		
SC_1	**SC_2**	**RW_1**	**RW_0**	**M_2**	**M_1**	**M_0**	**BCD**		
1	0	1	1	0	0	0	0	=	B0H

Source Program

```
MOV  AL,B0H
OUT  0BH,AL         ;  Loads  control  word  (B0H)  in  the  control
                    ;  register.
MOV  AL,30H
OUT  0AH,AL         ;  Loads  lower  byte  of  (30H)the  count.
MOV  AL,0C0H
OUT  0AH,AL         ;  Loads  higher  byte  (C0H)  of  the  count.
```

Example & Quiz

The frequency of CLK is 2MHz, write initiation program to let counter 0 generate an interruption request after 100μs, let counter 1 generate 50% duty cycle square wave with a period of 10μs, and let counter 2 generate a negative pulse every 1ms.



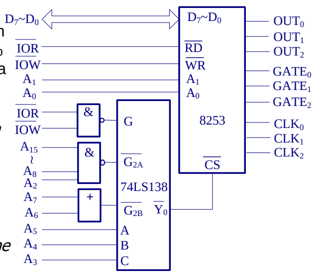
```
MOV DX, 0FF07H
MOV AL, 00010000B          ;counter 0, write LSB only, mode 0, binary
OUT DX, AL
MOV AL, 01010110B          ;counter 1, write LSB only, mode 3, binary
OUT DX, AL

MOV DX, 0FF04H
MOV AL, 200                ; initial count for counter 0
OUT DX, AL
MOV DX, 0FF05H
MOV AL, 20                 ;initial count for counter 1
OUT DX, AL

MOV DX, 0FF07H
MOV AL, 10110100B          ;counter 2, write LSB and MSB, mode 2
OUT DX, AL

MOV DX, 0FF06H
MOV AX, 2000               ; initial count for counter 2
OUT DX, AL
MOV AL, AH
OUT DX, AL
```

Quiz

The frequency of CLK is 2MHz, write initiation program to let counter 1 generate 50% duty cycle square wave with a period of 1s.

Use counter 0 to work as a rate generator (mode 2) with an initial count 0 in BCD (the max count), which will generate a clock with the frequency of 200HZ.

Then use OUT0 as the input clock of counter 1, and set the counter 1 to work in mode 3 with the initial count 200.



Lecture 09: Interrupts & 8259

Interrupts in 8086/8088

- 256 **interrupt types** in total
  - INT 00 ~ INT 0FFh
- Type * 4 = PA of **interrupt vector**
  - The first 1KB is used to store interrupt vectors, called **Interrupt Vector Table** (IVT)
- Interrupt vector points the entrance address of the corresponding **interrupt service routine (ISR)**

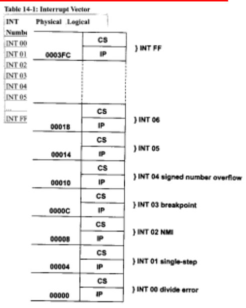
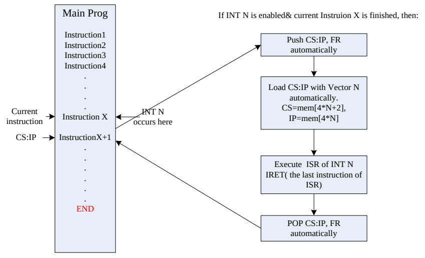


Table 14-1: Interrupt Vector

INT Number	Physical	Logical	
INT 00		CS	} INT FF
INT 01	0003FC	IP	
INT 02			
INT 03			
INT 04			
INT 05			
...		CS	} INT 06
INT FF	00018	IP	
		CS	} INT 05
	00014	IP	
		CS	} INT 04 signed number overflow
	00010	IP	
		CS	} INT 03 breakpoint
	0000C	IP	
		CS	} INT 02 NMI
	00008	IP	
		CS	} INT 01 single-step
	00004	IP	
		CS	} INT 00 divide error
	00000	IP	

Main Program and ISR



Main Prog
Instruction1
Instruction2
Instruction3
Instruction4
Current instruction
Instruction X
CS:IP
InstructionX+1
END
INT N occurs here
If INT N is enabled& current Instruion X is finished, then:
Push CS:IP, FR automatically
Load CS:IP with Vector N automatically. CS=mem[4*N+2], IP=mem[4*N]
Execute ISR of INT N IRET(the last instruction of ISR)
POP CS:IP, FR automatically

An ISR is launched by an interrupt event (internal 'int xx', external NMI and INTR) . So ISR is 'separated' from main.

Categories of Interrupts

- Hardware (external) interrupts
  - Maskable (from INTR)
  - Non-maskable (from NMI)
- Software (internal) interrupts
  - Using INT instructions
  - Predefined *conditional* (*exception*) interrupts

Hardware Interrupts

- Non-maskable interrupt
  - Trigger: NMI pin, input-signal, rising edge and two-cycle high activate
  - TYPE: INT 02
  - Not affected by the IF
  - Reasons:
    - E.g., RAM parity check error, interrupt request from co-CPU 8087

Hardware Interrupts

- Maskable interrupt
  - Trigger: INTR pin, input-signal, high active
  - TYPE: No predefined type
  - IF = 1, enable; IF = 0, disable
    - **STI** sets IF, **CLI** clears IF
  - Reasons:
    - Interrupt requests of external I/O devices

Procedure for Processing Maskable Interrupts

- CPU responds to INTR interrupt requests
  - External I/O devices send interrupt requests to CPU
  - CPU will check INTR pin on the last cycle of an instruction: if the INTR is high and IF = 1, CPU responds to the interrupt request
  - CPU sends two ~INTA signals to the I/O device
  - After receiving the second ~INTA, I/O device sends the interrupt type *N* on the data bus

Procedure for Processing Maskable Interrupts

- CPU executes the ISR of INT *N*
  - CPU reads the *N* from data bus
  - Push the **FR** in stack
  - Clear **IF** and **TF**
  - Push the CS and IP of the next instruction in stack
  - Load the ISR entrance address and moves to the ISR
  - At the end of the ISR, **IRET** will pop IP, CS and FR in turn, CPU returns to previous program and proceeds

Software Interrupts

- INT xx instruction
  - An ISR is called upon instruction such as "INT xx"
    - E.g., int 21h ; Dos service
  - CPU always responds and goes execute the corresponding ISR
    - Not affected by the IF
  - You can "CALL" any ISR by using the INT instruction

Difference between INT & CALL

- CALL FAR can jump anywhere within 1MB vs. INT jumps to a fix location (finding the corresponding ISR)
- CALL FAR is in the sequence of instructions vs. an external interrupt can come in at any time
- CALL FAR cannot be masked (disabled) vs. an external interrupt can be masked
- CALL FAR saves CS:IP of next instruction vs. INT saves FR + CS:IP of next instruction
- last instruction: RETF vs. IRET

Software Interrupts

- Predefined conditional interrupts
  - "INT 00" (divide error)
    - Reason: dividing a number by zero, or quotient is too large
  - "INT 01" (single step)
    - If TF = 1, CPU will generates an INT 1 interrupt after executing each instruction for debugging
  - "INT 03" (breakpoint)
    - When CPU hits the breakpoint set in the program, CPU generates INT 3 interrupt for debugging
  - "INT 04" (signed number overflow)
    - **INTO** instruction
    - Check the OF after an arithmetic instruction

```
;How to clear TF?
    PUSHF
    POP AX
    AND AX 0FEFFH
    PUSH AX
    POPF
```

```
MOV  AX,0009H
ADD AX,0080H
INTO
```

Procedure for Processing Non-Maskable & Software Interrupts

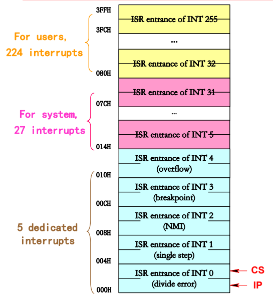
For NMI

CPU checks NMI, generates INT 02 interrupt automatically regardless of **IF** and executes to the ISR

For software (internal) interrupts

CPU generates INT *N* interrupt automatically and executes to the corresponding ISR

Interrupt Vector Table of 8086/8088


3FFH
3FCH
080H
07CH
014H
010H
00CH
008H
004H
000H
For users, 224 interrupts
For system, 27 interrupts
5 dedicated interrupts
ISR entrance of INT 255
...
ISR entrance of INT 32
ISR entrance of INT 31
...
ISR entrance of INT 5
ISR entrance of INT 4 (overflow)
ISR entrance of INT 3 (breakpoint)
ISR entrance of INT 2 (NMI)
ISR entrance of INT 1 (single step)
ISR entrance of INT 0 (divide error)
CS
IP

- 256 interrupts
  - 0～4 dedicated
  - 5～31reserved for system use
    - 08H～0FH：8259A
    - 10H～1FH：BIOS
  - 32～255 reserved for users
    - 20H～3FH：DOS
    - 40H～FFH：open

Interrupt Priority

INT instruction has higher priority than INTR and NMI

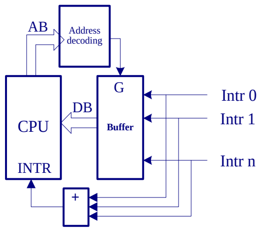
NMI has higher priority than INTR

For different external interrupt requests, different strategies can be used to determine their priorities.

Priority of INTR Interrupts

Software polling

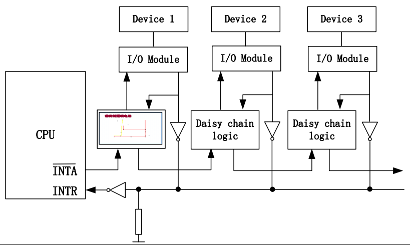
The sequence of checking determines the priority



Priority of INTR Interrupts

Hardware checking

The location in the daisy chain counts



Vectored interrupt controller 8259

8259 is Programmable Interrupt Controller (PIC)

It is a tool for managing the interrupt requests.

8259 is a very flexible peripheral controller chip:

PIC can deal with up to 64 interrupt inputs

interrupts can be masked

various priority schemes can also programmed.

originally (in PC XT) it is available as a separate IC

Later the functionality of *(two PICs)* is in the motherboards chipset.

In some of the modern processors, the functionality of the *PIC* is built in.

FIGURE 1 Block diagram and pin definitions for the 8259A Programmable Interrupt Controller (PIC). (Courtesy of Intel Corporation.)

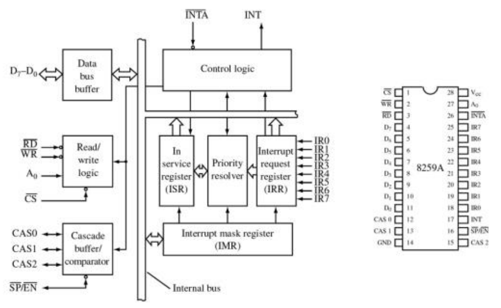
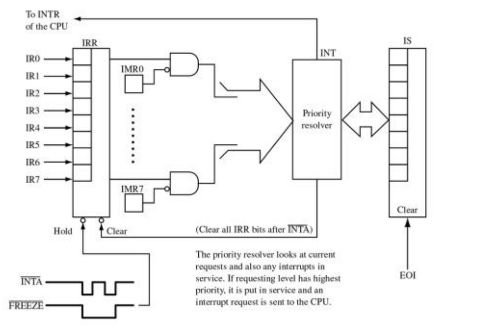


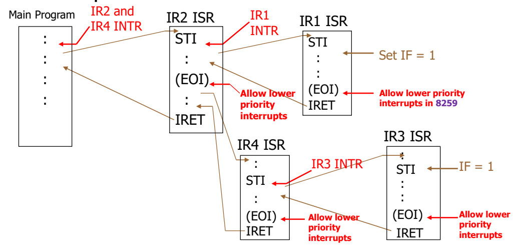
FIGURE 2 All interrupt requests must pass through the PIC's interrupt request register (IRR) and interrupt mask register (IMR). If put in service, the appropriate bit of the in-service (IS) register is set.



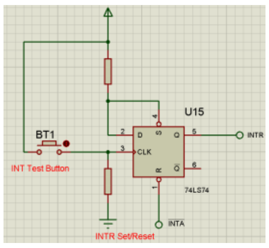
The priority resolver looks at current requests and also any interrupts in service. If requesting level has highest priority, it is put in service and an interrupt request is sent to the CPU.

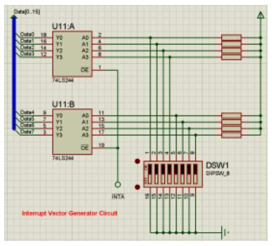
Interrupt Nesting

Higher priority interrupts can interrupt lower interrupts



Exp. 3: Interrupts





Interrupt Example

```
INT_INIT PROC FAR
   CLI           ; Disable interrupt
   MOV AX, 0
   MOV ES, AX    ; Set up IVT
   MOV BH, 0
   MOV CL, 2
   SHL BX, CL
   MOV WORD PTR ES:[BX], OFFSET MYIRQ
   MOV WORD PTR ES:[BX + 2], SEG MYIRQ
   STI

   RET
INT_INIT ENDP
```

```
MYIRQ   PROC FAR
   MOV AL, DS:[0]
   XOR AL, 80H
   MOV DS:[0], AL

   MOV AL, 10000000B
   MOV DX, L8255CS
   OUT DX, AL

   MOV DX, L8255PC
   MOV AL, DS:[0]
   OUT DX, AL
   IRET
MYIRQ   ENDP
```

Lecture 10: BIOS and DOS Programming

BIOS and DOS Interrupts

You can use those useful subroutines within BIOS and DOS to implement your applications

You can "CALL" those subroutines by explicitly embedding BIOS and DOS interrupt instructions in your program

We study: INT 10h (BIOS interrupt) and INT 21h (DOS interrupt)

- Each one can perform many functions
- See Appendices D and E for the complete function lists

BIOS INT 10H Programming

INT 10H subroutines are burned into the ROM BIOS (in 80x86-based IBM PCs)

Used to communicate with the computer's screen video

- E.g., changing the color of characters or background color, cleaning the screen, changing the location of the cursor, drawing lines on the screen
- By setting AH with different values, you can "call" those functions

Scrolling Window

Scroll window up

AH=06h

Additional Call Registers
AL = number of lines to scroll
BH = display attribute
CH = y coordinate of top left
CL = x coordinate of top left
DH = y coordinate of lower right
DL = x coordinate of lower right

Scroll window down

AH=07h

Additional Call Registers
AL = number of lines to scroll
BH = display attribute
CH = y coordinate of top left
CL = x coordinate of top left
DH = y coordinate of lower right
DL = x coordinate of lower right

Note: If AL = 0, the entire window is blank.

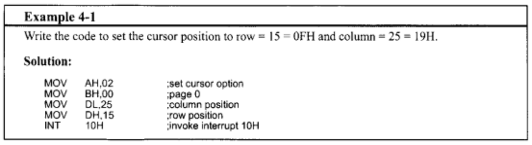
Code example: clear the screen

```
MOV     AX,0600H    ;scroll entire screen
MOV     BH,07       ;normal attribute
MOV     CX,0000     ;start at 00,00
MOV     DX,184FH    ;end at 24,79 (hex = 18,4F)
INT     10H         ;invoke the interrupt
```

Set cursor position

AH=02h

Additional Call Registers
BH = page number
DH = row
DL = column



Example 4-1

Write the code to set the cursor position to row = 15 = 0FH and column = 25 = 19H.

Solution:

```
MOV     AH,02       ;set cursor option
MOV     BH,00       ;page 0
MOV     DL,25       ;column position
MOV     DH,15       ;row position
INT     10H         ;invoke interrupt 10H
```

Set Video Mode

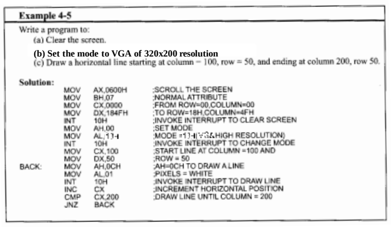
AH=00h

- In text mode, the screen is viewed as a matrix of rows and columns of characters
  - E.g., AL = 03h: VGA 80x25 chars, 16 colors
- In graphics mode, the screen is viewed as a matrix of horizontal and vertical pixels
  - Each pixel can have different color
  - The size of video memory decides the number of pixels and colors
  - E.g., AL = 13H: VGA 320x200 pixels, 256 colors

Draw Pixel

AH=0CH

Additional Call Registers
BH = page number
DH = row
DL = column



Example 4-5

Write a program to:
(a) Clear the screen.
(b) Set the mode to VGA of 320x200 resolution
(c) Draw a horizontal line starting at column = 100, row = 50, and ending at column 200, row 50.

Solution:

```
        MOV     AX,0600H    ;SCROLL THE SCREEN
        MOV     BH,07       ;NORMAL ATTRIBUTE
        MOV     CX,0000     ;FROM ROW=00,COLUMN=00
        MOV     DX,184FH    ;TO ROW=18H,COLUMN=4FH
        INT     10H         ;INVOKE INTERRUPT TO CLEAR SCREEN
        MOV     AH,00       ;SET MODE
        MOV     AL,13H      ;MODE =13H(VGA HIGH RESOLUTION)
        INT     10H         ;INVOKE INTERRUPT TO CHANGE MODE
        MOV     CX,100      ;START LINE AT COLUMN =100 AND
        MOV     DX,50       ;ROW = 50
BACK:   MOV     AH,0CH      ;AH=0CH TO DRAW A LINE
        MOV     AL,01       ;PIXELS = WHITE
        INT     10H         ;INVOKE INTERRUPT TO DRAW LINE
        INC     CX          ;INCREMENT HORIZONTAL POSITION
        CMP     CX,200      ;DRAW LINE UNTIL COLUMN = 200
        JNZ     BACK
```

DOS Interrupt 21H

Provided by MS-DOS
- Based on BIOS-ROM
- After the DOS is loaded into the memory, you can invoke INT 21H to perform some extremely useful functions
- E.g., input from keyboard, display results on screen
- By setting AH with different values, you can invoke those functions

Output String on Screen

AH=09H
- Can be used to send a set of ASCII data to the monitor
- DX is set to the offset address of the ASCII string to be displayed (DS is assumed to be the data segment)
- All characters will be displayed until it encounters the dollar sign "$"

```
DATA_ASC DB  'The earth is but one country','$'

          MOV       AH,09                ;Option 09 to display string of data
          MOV       DX,OFFSET DATA_ASC   ;DX= offset address of data
          INT       21H                  ;invoke the interrupt
```

Exit to DOS

AH=4CH
- AL=00H

```
MOV AX, 4C00H
INT 21H
```

Lecture 11: Serial Data Communication & 8251

Data Communication

Data transmission is the transfer of data from point to point often represented as an electromagnetic signal over a physical communication channel

A **communication channel** refers to the medium used to convey information from a sender (or transmitter) to a receiver.
- Examples: copper wires, optical fibbers or wireless communication channels.

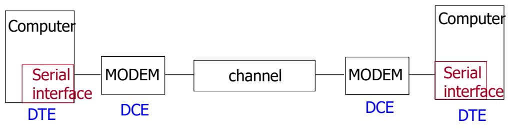
Two Ways: Parallel & Serial

Parallel data transfers:
- Each bit uses a separate line (wire)
- Often 8 or more lines are used
- Control signals in addition
- Fast & expensive & for short-distance communication

Serial data transfers:
- One single data line
- Bits are sent over the line one by one
- No dedicated lines for control signals
- Cheap & slow & for long-distance communication

The Whole Picture of Serial Communication

The sender and receiver need a **protocol** to make sense of data:
❖e.g., how the data is packed, how many bits constitute a character, when the data begin and end



DTE- Data Terminal Equipment, usually a computer.

DCE- Data Communication Equipment, usually a *modem*.

Serial interface –ICs such as 8251A,16550, and 8250, connecting DTE and DCE.

Serial Communication

Data transfer rate
Synchronization methods
Communication modes
Error detection
Modulation and Demodulation

Data Transfer Rate

Symbol rate, a misnomer is *baud rate*

The number of distinct symbol or pulse changes (signaling events) made to the transmission medium per second in a digitally modulated signal or a line code, quantified using the baud unit

Each symbol can represent one (binary encoded signal) or several bits of data

Bit rate

the number of bits that are conveyed or processed per unit of time, quantified using the bits per second (**bit/s** or **bps**) unit

Synchronization Methods

Asynchronous serial communication:

Transfer one byte at a time

The starting of each byte is asynchronous, and therefore each byte needs synchronization between the sender and the receiver using start bit.

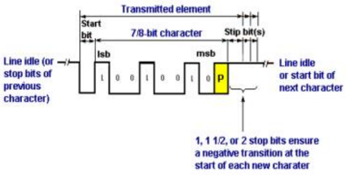
Synchronous serial communication:

Transfer a block of data at a time

The sender and the receiver are synchronized at the beginning of data transfer using synch characters.

Asynchronous Transmission

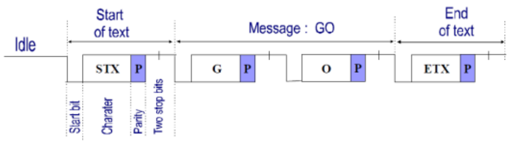
- In *asynchronous transmission*, the receiver clock (R×C) runs *unsynchronized* with respect to the incoming signal (R×D).
- Each character (byte) is encapsulated between an additional **start bit** and one or more **stop bits**.
- The state of the signal on the transmission line between characters is **idle** state.


Transmitted element
Start bit
7/8-bit character
Stip bit(s)
lsb
msb
Line idle (or stop bits of previous character)
1 0 0 1 0 0 1 0 P
Line idle or start bit of next character
1, 1 1/2, or 2 stop bits ensure a negative transition at the start of each new charater

Asynchronous Transmission

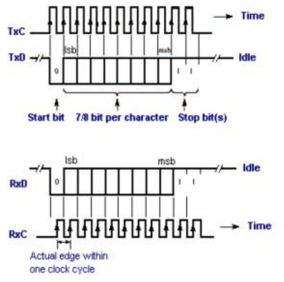
Example:

Construct the transmitted frame using a*synchronous transmission mode* which contains the following data: **GO**. Assume that the number of stop bits is 2 and parity bit is used.



Asynchronous Transmission

Principle of Operation and Timing:



Amplitude/Phase Distortions

Figure 1 below (from Notes 99-2, page 7) shows the reconstruction of the pulse train using the a_0 + first 9 coefficients a_n of the Fourier series, that is, using up to the term a_9. (These coefficients are: a_0= 0.25 and a_1 through a_9 respectively 0.4502 0.3183 0.1501 0.0000 -0.0900 -0.1061 -0.0643 0.0000 0.0500)

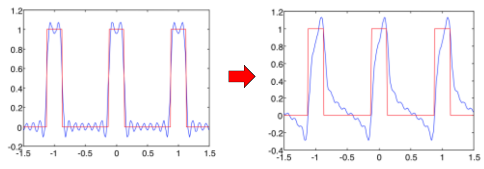


Figure 1

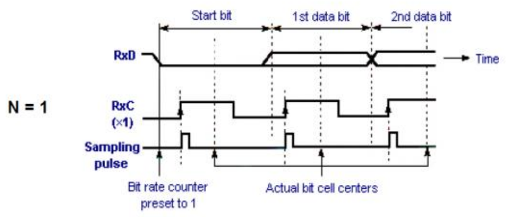
Asynchronous Transmission

Bit Synchronization in Asynchronous Transmission:

- The local receiver clock is *N* times the transmitted bit rate (*N*=16 is common).
- The first 1→0 transition is associated with the start bit.
- Each bit is sampled at the center to avoid delay distortion problem.
- After the first transition is detected, the signal is sampled after *N*/2 clock cycles and then subsequently after *N* clock cycles for each bit in the character.

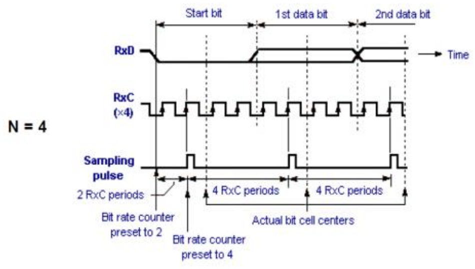
Asynchronous Transmission

Bit Synchronization in Asynchronous Transmission:



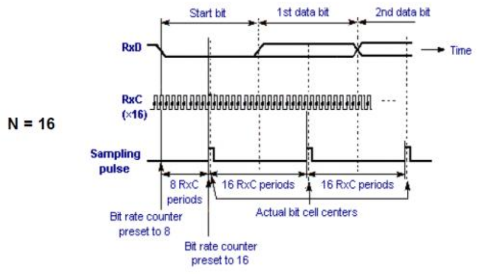
Asynchronous Transmission

Bit Synchronization in Asynchronous Transmission:



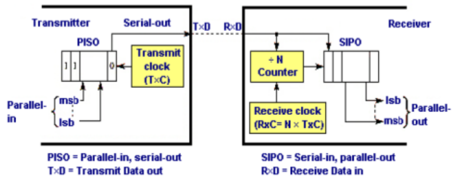
Asynchronous Transmission

Bit Synchronization in Asynchronous Transmission:



Asynchronous Transmission

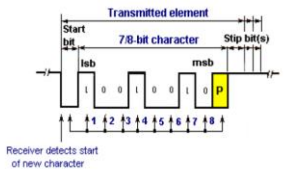
Principle of operation and Timing:



Asynchronous Transmission

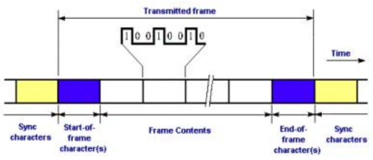
Character Synchronization in Asynchronous Transmission:

- After the start bit is detected, the receiver achieves character synchronization simply by counting the programmed number of bits.



Synchronous Transmission

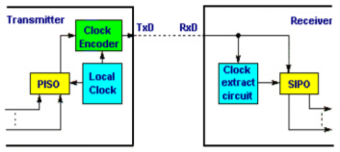
- The complete block or frame of data is transmitted as a contiguous stream with no delay between each 8-bit element.



Synchronous Transmission

Bit Synchronization using Synchronous Transmission:

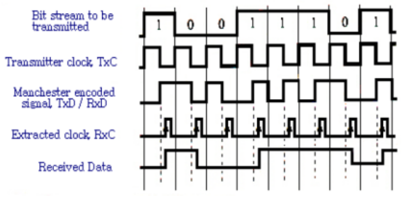
- With synchronous transmission, the receiver clock (R×C) operates in synchronism with the received data signal (R×D).
- Clock Encoding and Extraction: The clock information is embedded into the transmitted signal and subsequently extracted by the receiver.



Synchronous Transmission

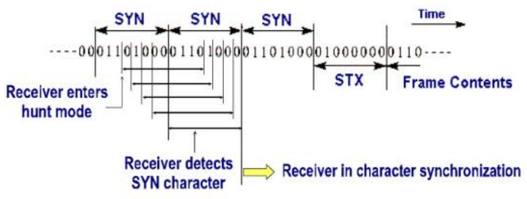
Bit Synchronization using Synchronous Transmission:

Clock Encoding and Extraction:



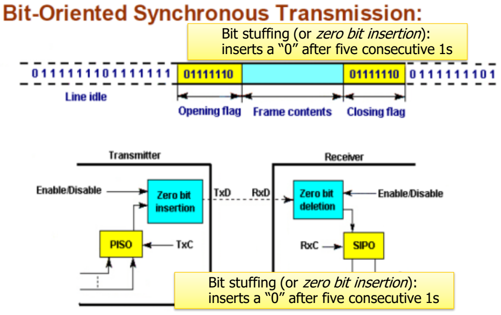
Synchronous Transmission

1. Character-Oriented Synchronous Transmission:


SYN
SYN
SYN
Time
----0001101000011010000110100001000000110----
STX
Frame Contents
Receiver enters hunt mode
Receiver detects SYN character
Receiver in character synchronization

Synchronous Transmission

2. Bit-Oriented Synchronous Transmission:

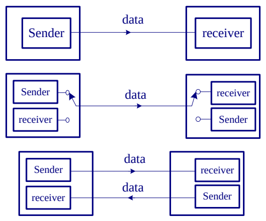


Communication Modes

Simplex:
- Only one way
- E.g., printer

Half-duplex:
- Data is transmitted one way at a time
- E.g., walky-talky

Full-duplex
- Data can go both ways at the same time
- E.g., telephone



Error Detection

Parity bit
- Used in asynchronous serial communication
- Put an extra parity bit at the end of each character
  - Even-parity: the data and the parity bit has an even number of 1s; odd-parity

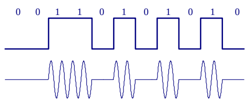
CRC Calculation
- *k*-bit data, *n*-bit CRC: $\frac{M(X) \times X^n}{G(X)}$
- Example:
  - Given G(x)= $x^3 + x^2 + 1$ ->1101, (take the coefficients of the polynomial, n=3)
  - If data is 1010110, M(x) * X^n -> 1010110000
  - CRC = 1010110000%1101 (the remainder of binary division, using XOR operation)

Modulation and Demodulation

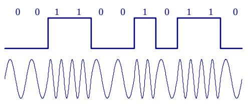
It is not suitable to transmit digital signals directly on a channel for a long distance
- signal distortion
- Need to modulate digital signals and get analog signals at the sender, and demodulate the analog signals and get the original digital signals

Three parameters (Amplitude, frequency, phase) of the carrier can be used for the modulation and demodulation purpose
- Amplitude-Modulating (AM)
- Frequency-Shift Keying (FSK)
- Phase-Shift Keying (PSK)

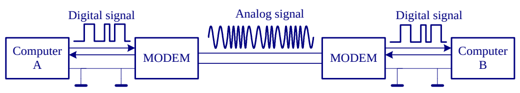
Modulation and Demodulation



1 and 0 are represented using different amplitude



1 and 0 are represented using different frequency



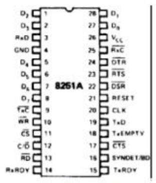
8251 USART Chip

Capable of doing both asynchronous and synchronous data communication
- synchronous: baud rate 0-64K, characters can be 5, 6, 7, or 8 bits, automatically detect or insert sync characters
- Asynchronous: baud rate 0-19.2K, characters can be 5, 6, 7, or 8 bits, automatically insert start, stop and parity bits, TxC and RxC clocks can be 1, 16, or 64 times of the baud rate

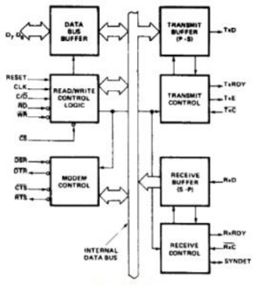
Full duplex, double-buffered

Error checking circuit

8251

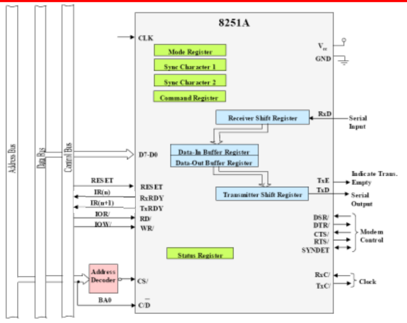


Pin Configuration

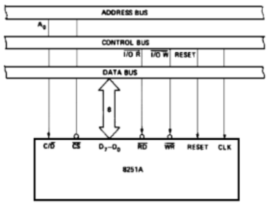


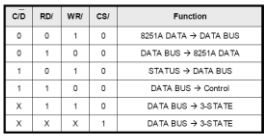
Block Diagram

8251 Communication Interface



Interfacing to 8088





C/D	RD/	WR/	CS/	Function
0	0	1	0	8251A DATA → DATA BUS
0	1	0	0	DATA BUS → 8251A DATA
1	0	1	0	STATUS → DATA BUS
1	1	0	0	DATA BUS → Control
X	1	1	0	DATA BUS → 3-STATE
X	X	X	1	DATA BUS → 3-STATE

8251 Signals

The 8251A is **doubled-buffered**. This means that one character can be loaded into a **data-out buffer register** while another character is being shifted out of the actual **transmit shift register**.

The **TxRDY** output of the 8251A will go high when:

- The **data-out buffer register** is Empty for another character from the CPU.
- The **CTS/** input has been asserted low.
- The **transmit-enable** (**TxEN**) bit of the 8251A's command word is set.

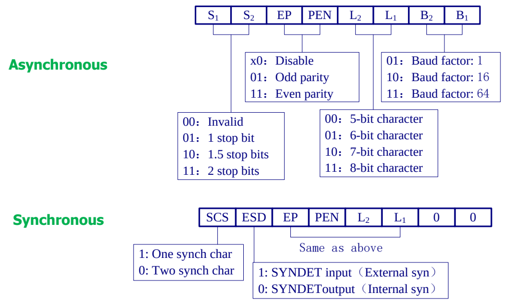
The **TxEMPTY** output of the 8251A will go high when both the **data-out buffer register** and the **transmit shift register** are empty.

The **RxRDY** output of the 8251A will go high when:

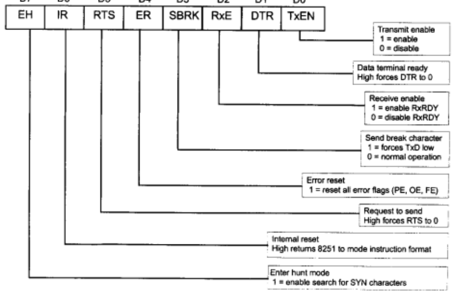
- The **data-in buffer register** is full and is ready to be read by the CPU.
- The **receive-enable** (**RxE**) bit of the 8251A's command word is set.

If the CPU does not read a character from the **data-in buffer register** before another character is shifted in, the first character will be overwritten and lost.

8251 Mode Word



8251 Command Word



8251 Command Word

Initializing the **TxEN** bit to 1 will enable the transmitter section of the 8251A and the **TxRDY** output.

Initializing **DTR/** bit to 1 will cause the DTR/ output of the 8251A to be asserted low. This signal is used to tell a modem that a PC or terminal is operational.

Initializing **RxE** bit to 1 will enable the RxRDY output of the 8251A.

Initializing **SBRK** bit to 1 will cause the 8251A to output characters of 0's including start bits, data bits, and parity bits (**break character**). A break character is used to indicate the end of block of transmitted data.

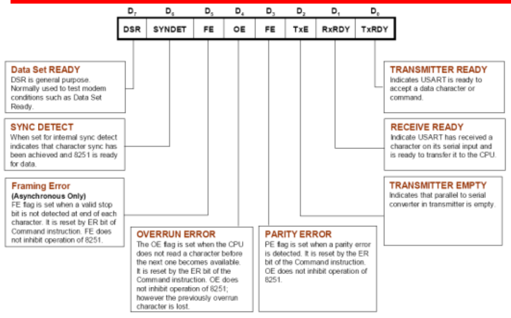
Initializing **ER** bit to 1 will cause the 8251A to reset the **parity**, **overrun**, and **framing** error flags in the 8251A status register.

Initializing **RTS** bit to 1 will cause the 8251A to assert its **request-to-send** (**RTS/**) output low. This signal is sent to a modem to ask whether a modem and the receiving system are ready for a data character to be sent.

Initializing **IR** bit to 1 will cause 8251A to be internally reset. After the software- reset command, a new mode word must be sent.

Initializing **EH** bit to 1 will cause 8251A to enter hunt mode (search for **SYN** characters, and is used only in synchronous mode.

8251 Status Word



Data Set READY
DSR is general purpose. Normally used to test modem conditions such as Data Set Ready.

SYNC DETECT
When set for internal sync detect indicates that character sync has been achieved and 8251 is ready for data.

Framing Error
(Asynchronous Only)
FE flag is set when a valid stop bit is not detected at end of each character. It is reset by ER bit of Command instruction. FE does not inhibit operation of 8251.

OVERRUN ERROR
The OE flag is set when the CPU does not read a character before the next one becomes available. It is reset by the ER bit of the Command instruction. OE does not inhibit operation of 8251; however the previously overrun character is lost.

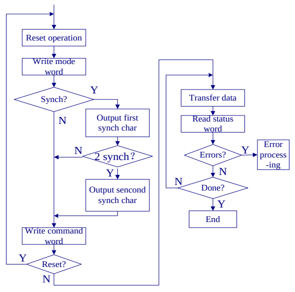
PARITY ERROR
PE flag is set when a parity error is detected. It is reset by the ER bit of the Command instruction. OE does not inhibit operation of 8251.

TRANSMITTER READY
Indicates USART is ready to accept a data character or command.

RECEIVE READY
Indicate USART has received a character on its serial input and is ready to transfer it to the CPU.

TRANSMITTER EMPTY
Indicates that parallel to serial converter in transmitter is empty.

Initializing 8251A



8251A Internal Reset on Power-Up

When power is first applied, the 8251A may come up in the mode, SYN character or command format.

It is safest to execute the worst-case initialization sequence (**SYNC** mode with two **SYN** characters). Loading three 00H consecutively into the device with C/#D = 1.

An **internal reset command** (40H) may then be issued to return the device to **mode word**.

The **mode word** must then be issued, and followed by the **command word**.

8251 Programming Example

Use 8251 to transfer 256 characters in asynchronous mode, assuming that the port addresses are 208H and 209H, the baud factor is 16, and 1 stop bit, 1 start bit, no parity bit, and 8-bit character are used.

Solution: Sender side: data is stored in Buf1

```
LEA DI, Buf1
MOV DX,209H
MOV AL,00H        ;worse-case init.
OUT DX, AL
CALL DELAY
MOV AL,00H        ;
OUT DX, AL
CALL DELAY
MOV AL,00H        ;
OUT DX, AL
CALL DELAY
MOV AL,40H        ;reset command
OUT DX, AL
```

```
MOV AL, 01001110B    ; mode word
OUT DX, AL
MOV AL, 00110111B    ; command word
OUT DX, AL
MOV CX, 256          ; to send 256 char.
NEXT: MOV DX, 209H
IN AL, DX            ; status word
AND AL, 01H          ; TxRDY?
JZ NEXT
MOV AL, [DI]
MOV DX, 208H         ; data register 208H
OUT DX, AL           ; send the char.
INC DI
LOOP NEXT
```

8251 Programming Example

Receiver side: data will be stored in Buf2

```
Data segment
buf2 DB 256 dup(?)
Data ends
    ¦
    MOV DX,209H
    MOV AL,00H
    OUT DX, AL
    CALL DELAY
    MOV AL,00H
    OUT DX, AL
    CALL DELAY
    MOV AL,00H
    OUT DX, AL
    CALL DELAY
    MOV AL,40H       ; reset
    OUT DX, AL
```

```
      MOV AL, 01001110B    ; mode word
      OUT DX, AL
      MOV AL, 00110111B    ; command word
      OUT DX, AL
      MOV CX, 256      ; to receive 256 char.
      MOV SI, 0
NEXT: MOV DX, 209H
      IN AL, DX            ; status word
      AND AL, 02H          ; RxRDY?
      JZ NEXT
      MOV DX, 208H
      IN AL, DX            ; receive a char
      MOV buf2[SI], AL
      INC SI
      LOOP NEXT
```