CS231 ระบบคอมพิวเตอร์และภาษาแอสแซมบลี

บทที่ 7: การขัดจังหวะการทำงาน

ผู้ช่วยศาสตราจารย์ ดร. ปวีณ เขื่อนแก้ว

สาขาวิชาวิทยาการคอมพิวเตอร์

คณะวิทยาศาสตร์ มหาวิทยาลัยแม่โจ้

เมื่อโปรแกรมเมอร์ต้องการตรวจสอบการทำงานของโปรแกรม มักนิยมใช้วิธีการ**หยุดการทำงานของโปรแกรม**
เพื่อตรวจสอบค่า รีจิสเตอร์และ Flag

เมื่อผู้ใช้กดปุ่มบนแป้นพิมพ์ โปรแกรม โปรแกรมที่ทำงานค้างอยู่ ให้**หยุดการทำงาน**
แล้วไปตรวจสอบข้อมูลที่ป้อนก่อน แล้วค่อยกลับมาทำงานต่อ

เมื่อโปรแกรมทำงานผิดพลาด หรือเมื่อมีคำสั่งไม่สามารถทำงานได้อย่างถูกต้อง ให้โปรแกรมย่อย**หยุดการทำงาน**
แล้วกลับมาที่โปรแกรมหลัก

หากผู้ใช้กดปุ่ม power ค้างไว้ ไม่ว่าขณะนั้นจะทำงานโปรแกรมอะไรอยู่ หรือโปรแกรมหยุดการทำงานไปแล้ว
ให้**หยุดทำทุกอย่าง** แล้วทำการรันโปรแกรมสำหรับ shutdown โดยไม่สามารถยกเลิกโปรแกรมนี้ได้

การขัดจังหวะการทำงาน (Interrupt) คือวิธีการหยุดการทำงานของโปรแกรมชั่วคราว เพื่อให้อุปกรณ์ภายนอกเข้ามาใช้งานไมโครโพรเซสเซอร์ โดยไมโครโพรเซสเซอร์จะตอบสนองโดยทำการรันโปรแกรมย่อยเพื่อประมวลผล ตามความต้องการของอุปกรณ์ภายนอก

โปรแกรมย่อยที่ทำงานเมื่อเกิดการ Interrupt เรียกว่า ISR (Interrupt Service Routine)

การ Interrupt สามารถเกิดจากสัญญาณไฟฟ้าภายนอก (Hardware Interrupt)
หรือเกิดจากโปรแกรมก็ได้ (Software Interrupt)

ไมโครโพรเซสเซอร์แต่ละรุ่นจะมีโครงสร้างและการทำงาน Interrupt แตกต่างกัน
บทนี้จะศึกษาการ Interrupt ของ 8086

Interrupt ของ 8086 มีอยู่ 256 แบบ (รูปแบบ / ชนิด / Type / เบอร์)

เมื่อเกิด Interrupt ขึ้น โปรแกรมจะกระโดดในลักษณะ CALL แบบ FAR ไปยังตำแหน่งที่แตกต่างกัน ขึ้นอยู่กับชนิดของ Interrupt

โดยไมโครโพรเซสเซอร์เมื่อได้รับสัญญาณ Interrupt จะทำงานดังนี้

1. ทำคำสั่งที่กำลังประมวลผลอยู่ให้เสร็จ
2. Push ค่า Flag , CS และ IP สำหรับ Return กลับ
3. เปิดดูตาราง Interrupt Vector ว่า Interrupt เบอร์ที่ถูกเรียก จะต้อง CALL ไปยัง Address ไหน
4. ตัดสัญญาณ Interrupt แล้วทำทำการ CALL ไปยัง Address นั้น

ระบบ Interrupt ของ 8086

- Hardware Interrupt
  - Maskable (INTR) รองรับ 256 แบบ
  - Non-maskable (NMI) รองรับ 2 เท่านั้น
- Software Interrupt รองรับ 256 แบบ

ชนิดของ Interrupt รวมแล้วมี 256 แบบ

หรือ

00H - FFH

ตารางที่ใช้กำหนด ตำแหน่งของโปรแกรมที่ใช้บริการ หรือทำงานเมื่อเกิด Interrupt แบบต่าง ๆ เรียกว่า Interrupt Vector

F0000	BIOS
E0000	BIOS
D0000	VRAM การ์ดจอ
C0000	VRAM การ์ดจอ
A0000	VRAM การ์ดจอ
90000	
.....	.....
.....	.....
10000	
00000	00000-003FF Interrupt Vector Table

เก็บ Segment (CS) ใช้ 2 ไบต์
เก็บ Offset (IP) ใช้ 2 ไบต์
ดังนั้น Interrupt 1 แบบจะใช้ 4 ไบต์

8086 รองรับการ Interrupt ทั้งหมด 256 แบบ
จึงใช้หน่วยความจำทั้งหมด 4 x 256 = 1024 ไบต์ หรือ 1 KiB

เริ่มจากแหน่งที่ 0 ถึง 1023 หรือ **0000 – 003FF** นั่นเอง

ตารางที่ใช้กำหนด ตำแหน่งของโปรแกรมที่ใช้บริการ หรือทำงานเมื่อเกิด Interrupt แบบต่าง ๆ เรียกว่า Interrupt Vector

F0000	BIOS
E0000	BIOS
D0000	VRAM การ์ดจอ
C0000	VRAM การ์ดจอ
A0000	VRAM การ์ดจอ
90000	
.....	.....
.....	.....
10000	
00000	00000-003FF Interrupt Vector Table

.....	.....	.....
0000C	IP	INT 3
0000A	CS	INT 2
00008	IP	INT 2
00006	CS	INT 1
00004	IP	INT 1
00002	CS	INT 0
00000	IP	INT 0

4 ไบต์

ตารางที่ใช้กำหนด ตำแหน่งของโปรแกรมที่ใช้บริการ หรือทำงานเมื่อเกิด Interrupt แบบต่าง ๆ เรียกว่า Interrupt Vector

.....	.....	.....
0000C	IP	INT 3
0000A	CS	INT 2
00008	IP	INT 2
00006	CS	INT 1
00004	IP	INT 1
00002	CS	INT 0
00000	IP	INT 0

เราสามารถ โปรแกรมไมโครโพรเซสเซอร์ ให้เรียกโปรแกรมย่อยที่ต้องการ เมื่อมีการเกิด Interrupt ได้การนำเอา Segment และ Offset ของตำแหน่งโปรแกรมย่อย มาเติมใน Interrupt Vector

ตัวอย่างเช่น ให้เรียกโปรแกรมย่อยที่ตำแหน่ง 0100:0400 เมื่อเกิด Interrupt แบบที่ 0

CS : IP

ให้นำค่า 0100 เขียนลงที่ตำแหน่ง 00002
และเขียนค่า 0400 ลงในตำแหน่ง 00000

หากต้องการ ให้เรียกโปรแกรมย่อยที่ตำแหน่ง 0100:0400 เมื่อเกิด Interrupt แบบที่ 1 ก็ให้เขียนข้อมูลลงตำแหน่ง INT 1 ซึ่งอยู่ตำแหน่งถัดไป

ให้นำค่า 0100 เขียนลงที่ตำแหน่ง 00006
และเขียนค่า 0400 ลงในตำแหน่ง 00004

ต้องการ Interrupt แบบที่ 7A ต้องเขียน CS:IP ลงในหน่วยความจำในตำแหน่งใด ?

.....	.....	.....
0000C	IP	INT 3
0000A	CS	INT 2
00008	IP	INT 2
00006	CS	INT 1
00004	IP	INT 1
00002	CS	INT 0
00000	IP	INT 0

ตำแหน่ง IP = n x 4

ตำแหน่ง CS = IP + 2

เมื่อ n คือ ชนิดหรือหมายเลข Interrupt

n=7A

ตำแหน่ง IP = 7A x 4 = 01E8

ตำแหน่ง CS = 1E8 + 2 = 1EA

8086 สามารถสร้าง Interrupt ได้ 256 แบบ (00 –FF) แต่มีบางตัวถูกใช้งานไปแล้ว

00 – 04 เป็น Default Interrupt ถูกต่อตรงกับสัญญาณดังต่อไปนี้

00 Divide by Zero เกิดเมื่อ มีการหารแล้วตัวหารมีค่าเป็น 0

01 Single Step ใช้สำหรับการรันทีละคำสั่ง (debug mode)

02 Non-maskable Interrupt เกิดเมื่อได้รับสัญญาณที่ขา INTR ของไมโครโพรเซสเซอร์

03 Break Point ใช้สำหรับสร้างจุดหยุดการทำงานของโปรแกรม (debug mode)

04 Overflow เกิดเมื่อการคำนวณทำให้เครื่องหมาย +/ - เปลี่ยนไป (บิตซ้ายสุดเปลี่ยน)

8086 สามารถสร้าง Interrupt ได้ 256 แบบ (00 –FF) แต่มีบางตัวถูกใช้งานไปแล้ว

หากพัฒนาโปรแกรมบน IBM PC จะมี Interrupt ที่ถูกใช้ไปแล้ว โดย BIOS

Interrupts	เกี่ยวข้องกับ
10H	จอภาพและการแสดงผล
11H	รายละเอียดของอุกรณ์เชื่อมต่อต่าง ๆ การ์ดต่าง ๆ ที่ใส่เพิ่มเข้าไป
12H	ตรวจสอบขนาดของหน่วยความจำทั้งหมด
13H	อ่าน/เขียน ข้อมูลลงดิสก์
15H	ใช้หน่วงเวลา (Delay)
16H	แป้นพิมพ์
19H	Reboot
1AH	นาฬิกา

8086 สามารถสร้าง Interrupt ได้ 256 แบบ (00 –FF) แต่มีบางตัวถูกใช้งานไปแล้ว

หาก Boot เครื่องด้วยระบบปฏิบัติการ MS-DOS จะมี Interrupt ที่ถูกใช้ไปโดยระบบปฏิบัติการ

Interrupts	เกี่ยวข้องกับ
20H	จบโปรแกรม และกลับสู่ shell
21H	เรียกใช้ API ของระบบปฏิบัติการ
33H	เรียกใช้ mouse driver

8086 สามารถสร้าง Interrupt ได้ 256 แบบ (00 – FF) แต่มีบางตัวถูกใช้งานไปแล้ว

Intel สงวนไว้ใช้เองในอนาคต สำหรับไมโครโพรเซสเซอร์รุ่นที่ยังไม่ออก

13H – 1FH (12 แบบ)

แต่ IBM ใช้ 13H, 15H, 16H, 19H, 1AH สำหรับ BIOS

ดังนั้นจึงมี Interrupt ให้เราใช้งานได้ในช่วง

20H - FFH (223 แบบ)

หากพัฒนาโปรแกรมบน MS-DOS ให้งดใช้งาน 20H,21H,33H

ทุกวันนี้ Intel ยังไม่ใช้งาน Interrupt ที่สงวนไว้

แต่ไปใช้ 05H – 12H แทน

MS-DOS ถูกยกเลิกไปตั้งแต่ WindowsXP และ IBM BIOS ก็ถูกใช้งานถึงแค่ขั้นตอนการบู๊ตเครื่องเท่านั้น

Windows11 ยกเลิกการสนับสนุน BIOS และเปลี่ยนไปใช้ UEFI

ระบบ Interrupt ของ 8086

- Hardware Interrupt
  - Maskable (INTR) รองรับ 256 แบบ
  - Non-maskable (NMI) รองรับ 2 เท่านั้น
- Software Interrupt รองรับ 256 แบบ

การสร้างสัญญาณ Interrupt ด้วย software ในโปรแกรม ใช้คำสั่ง

INT operand

Operand ที่ใช้ได้ : Immediate 8bit เท่านั้น

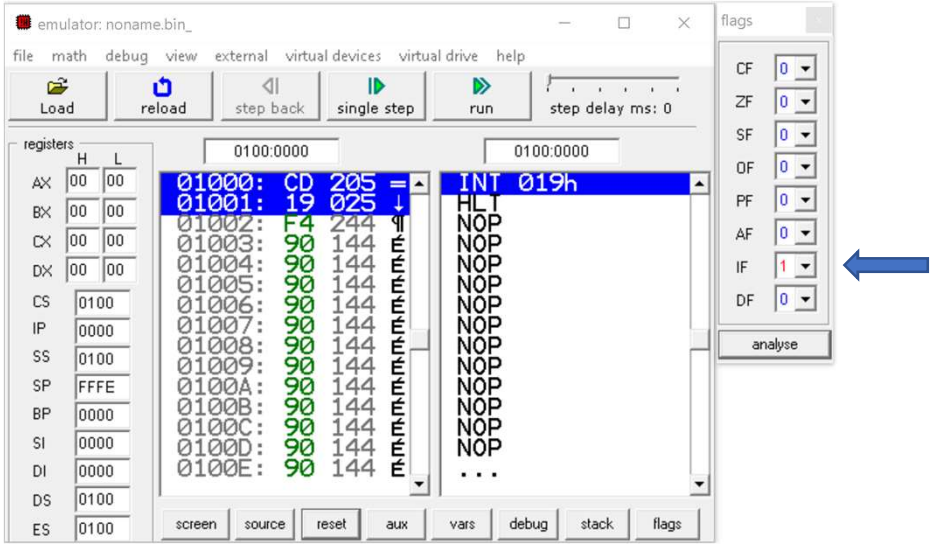
BIOS และ DOS มี API ให้เรียกใช้ผ่าน Interrupt โดยส่งค่าพารามิเตอร์ผ่านทางรีจิสเตอร์หรือ stack
ตัวอย่างเช่น

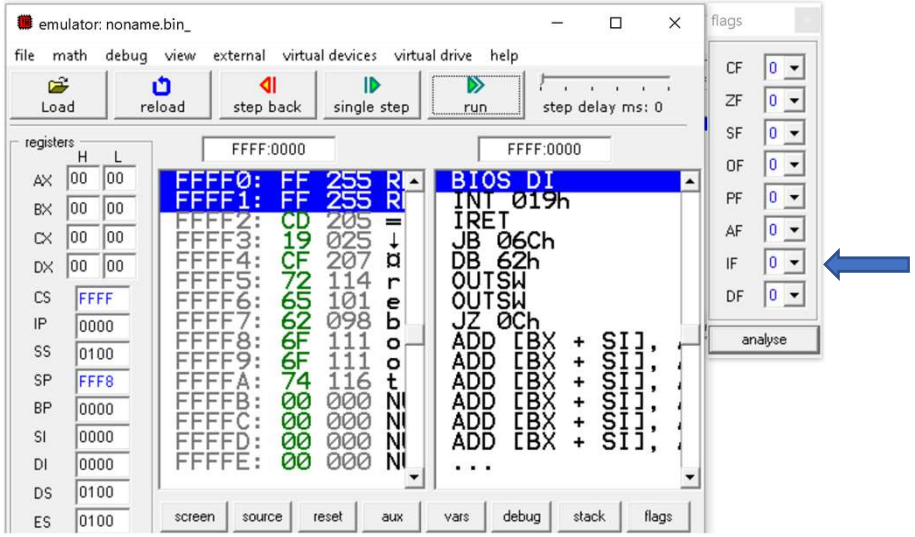
Interrupt หมายเลข 19H เป็น ISR ของ BIOS ใช้ในการ Reboot เครื่อง

Interrupt หมายเลข 19H เป็น ISR ของ BIOS ใช้ในการ Reboot เครื่อง

INT 19H

HLT

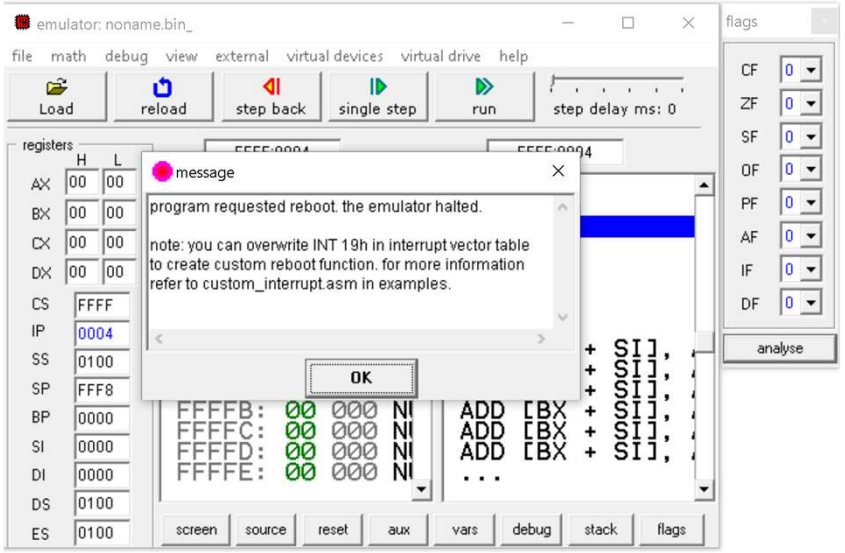




Interrupt หมายเลข 19H เป็น ISR ของ BIOS ใช้ในการ Reboot เครื่อง

INT 19H

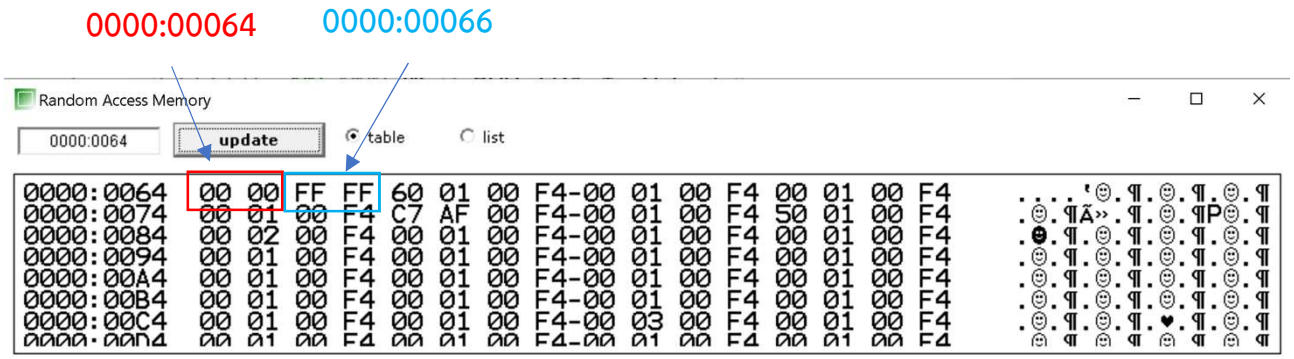
HLT



Interrupt 19H จะเก็บ Address ไว้ที่ตำแหน่ง

IP= 19H x 4 = **0064H**

CS= 64H +2 = **0066H**



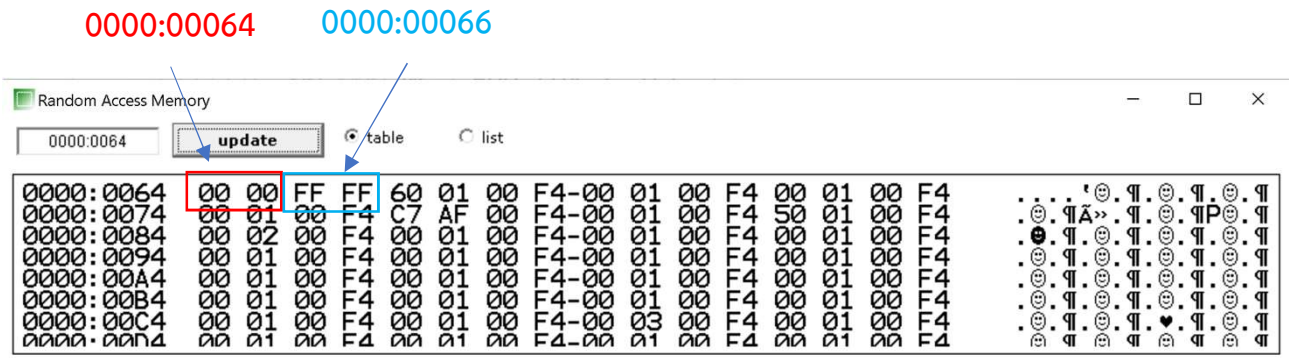
ข้อมูลใน Interrupt Vector บอกให้เรียกใช้โปรแกรมที่ตำแหน่ง CS : IP ที่ FFFF : 0000

Interrupt หมายเลข 19H เป็น ISR ของ BIOS ใช้ในการ Reboot เครื่อง

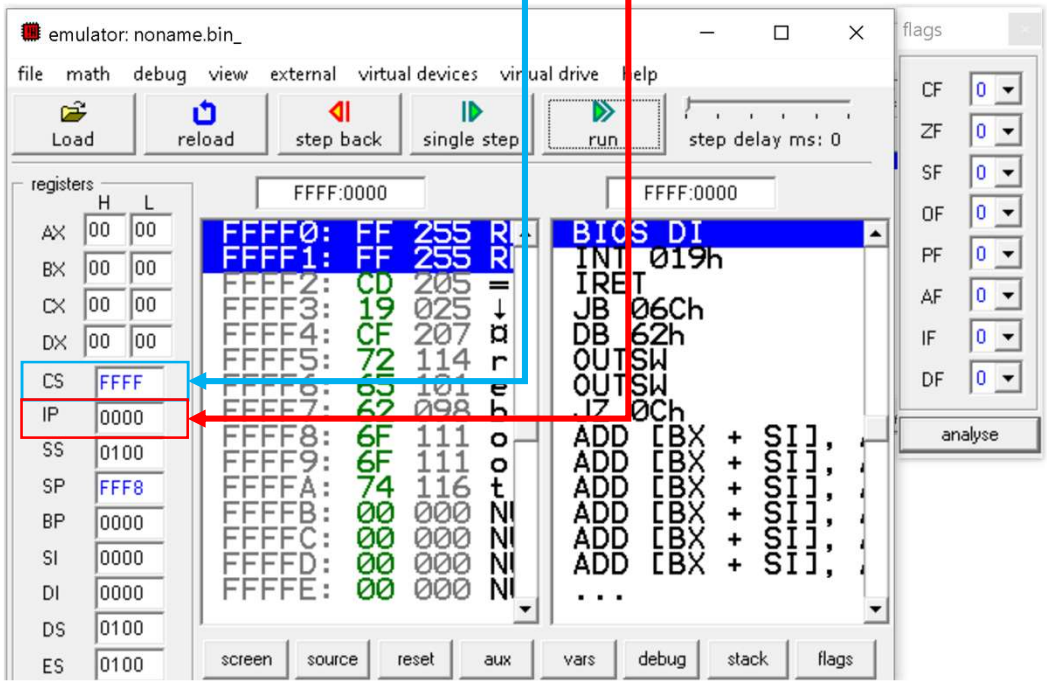
Interrupt 19H จะเก็บ Address ไว้ที่ตำแหน่ง

IP= 19H x 4 = **0064H**

CS= 64H +2 = **0066H**



ข้อมูลใน Interrupt Vector บอกให้เรียกใช้โปรแกรมที่ตำแหน่ง CS : IP ที่ FFFF : 0000



Interrupt Services ของ IBM 8086 PC (ที่ Emulator รองรับ)

http://www.ablmcc.edu.hk/~scy/CIT/8086_bios_and_dos_interrupts.htm

INT / AX

INT 10h/00h
INT 10h/01h
INT 10h/02h
INT 10h/03h
INT 10h/05h
INT 10h/06h
INT 10h/07h
INT 10h/08h
INT 10h/09h
INT 10h/0Ah
INT 10h/0Ch
INT 10h/0Dh
INT 10h/0Eh
INT 10h/13h

INT 10h/1003h
INT 11h
INT 12h
INT 13h/00h
INT 13h/02h
INT 13h/03h
INT 15h/86h
INT 16h/00h
INT 16h/01h
INT 19h
INT 1Ah/00h
INT 20h

INT 21h
INT 21h/01h
INT 21h/02h
INT 21h/05h
INT 21h/06h
INT 21h/07h
INT 21h/09h
INT 21h/0Ah
INT 21h/0Bh
INT 21h/0Ch
INT 21h/0Eh
INT 21h/19h
INT 21h/25h
INT 21h/2Ah
INT 21h/2Ch

INT 21h/35h
INT 21h/39h
INT 21h/3Ah
INT 21h/3Bh
INT 21h/3Ch
INT 21h/3Dh
INT 21h/3Eh
INT 21h/3Fh
INT 21h/40h
INT 21h/41h
INT 21h/42h
INT 21h/47h
INT 21h/4Ch
INT 21h/56h

INT 33h/0000h
INT 33h/0001h
INT 33h/0002h
INT 33h/0003h

Interrupt Services ของ IBM 8086 PC (ที่ Emulator รองรับ)

INT 10h / AH = 0Eh - teletype output.

input:

AL = character to write.

this functions displays a character on the screen, advancing the cursor and scrolling the screen as necessary. the printing is always done to current active page.

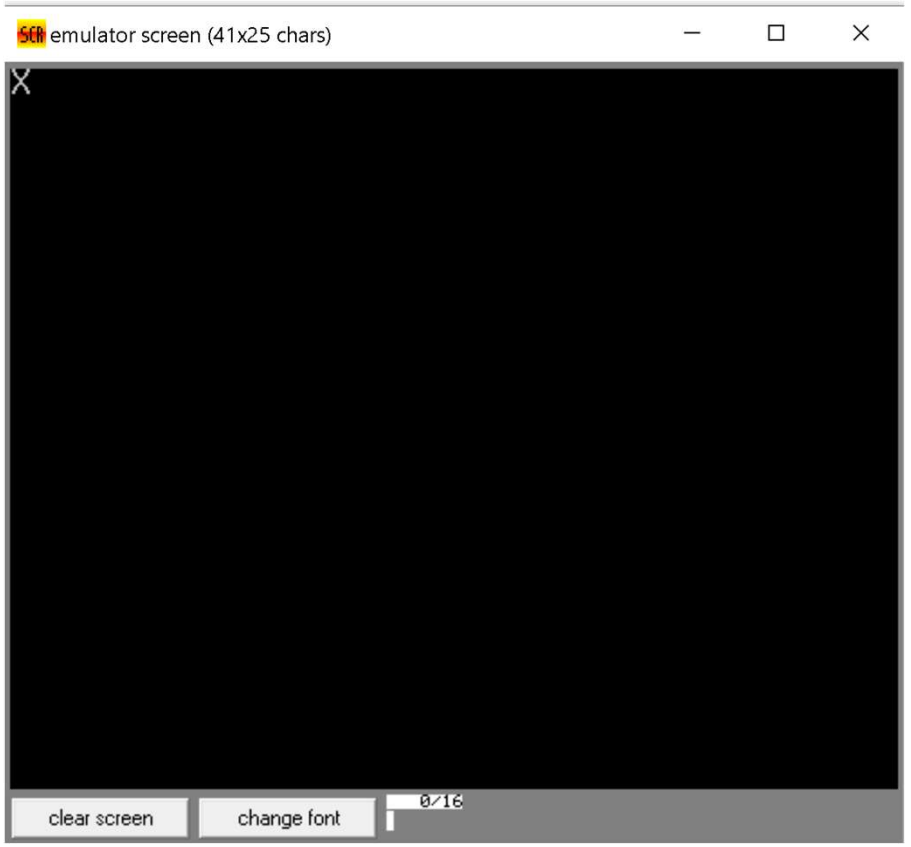
example:

```
mov al, 'a'
mov ah, 0eh
int 10h
; note: on specific systems this
; function may not be supported in graphics mode.
```

Teletypewriter (TTY)

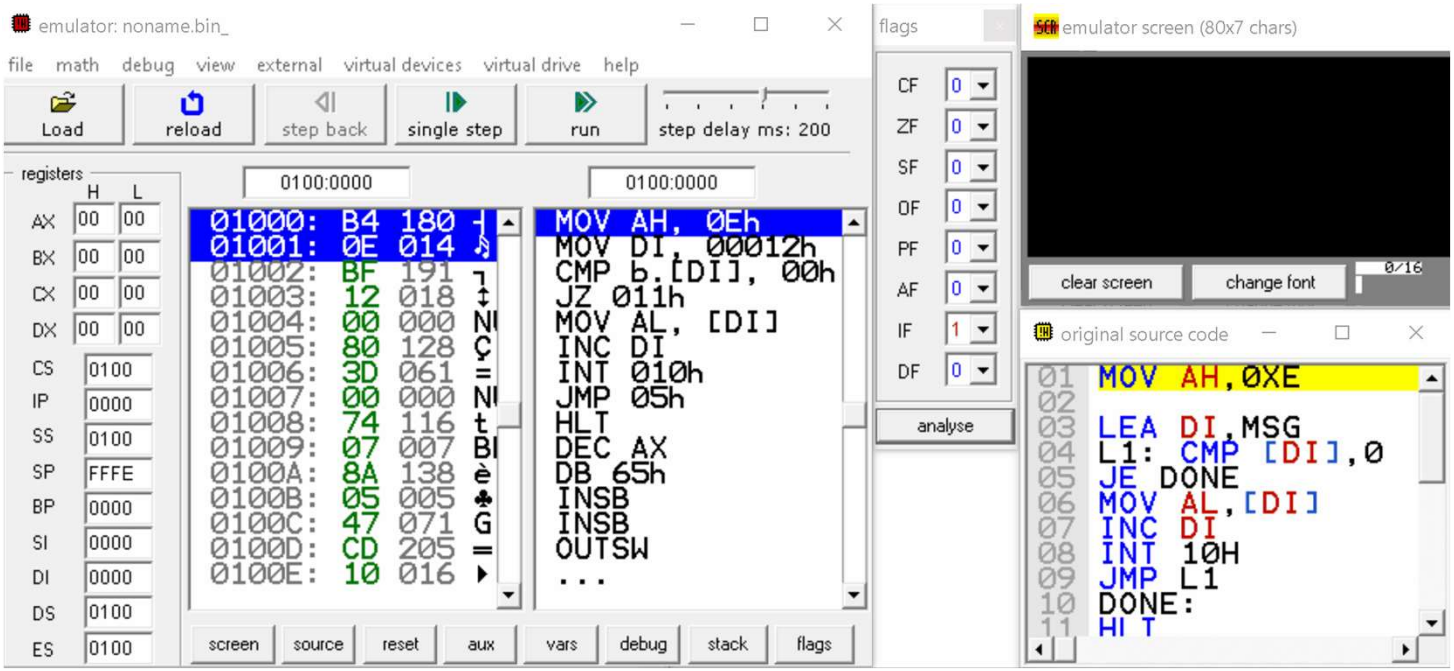
Interrupt Services ของ IBM 8086 PC (ที่ Emulator รองรับ)

```
MOV AH,0XE
MOV AL,'X'
INT 10H
HLT
```



Interrupt Services ของ IBM 8086 PC (ที่ Emulator รองรับ)

```
MOV AH,0XE

LEA DI,MSG
L1: CMP [DI],0
    JE DONE
    MOV AL,[DI]
    INC DI
    INT 10H
    JMP L1
DONE:
HLT

MSG DB "Hello World!",0
```



PC BIOS

INT 10h / AH = 0Eh - teletype output.

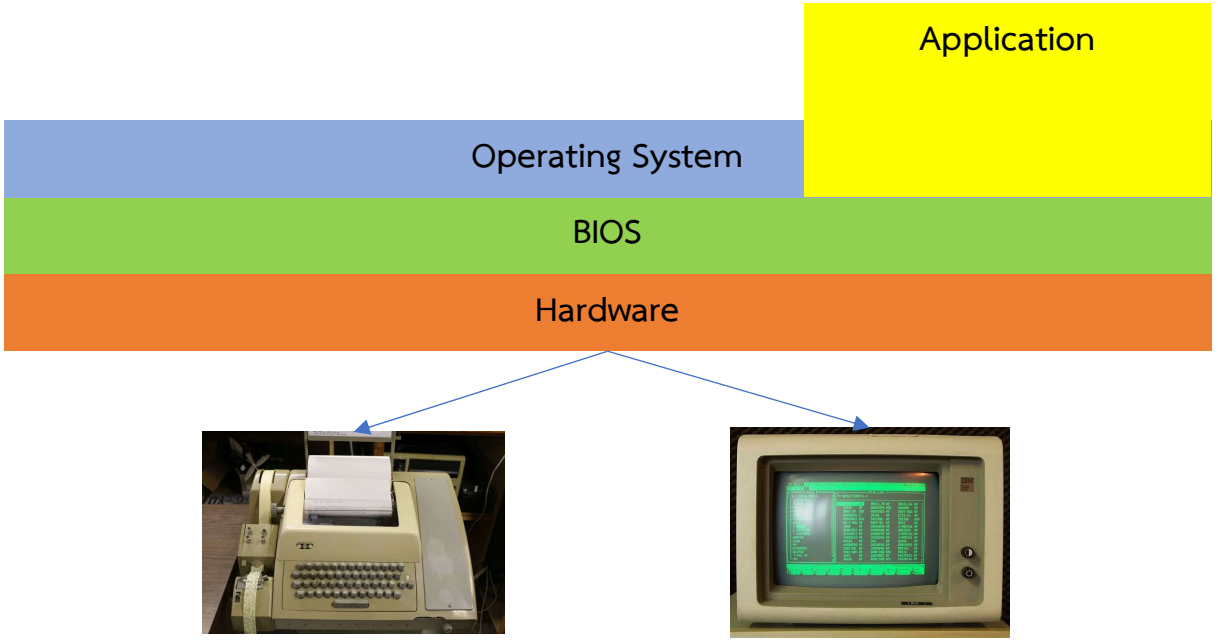
บริการนี้เป็นการพิมพ์ตัวอักษร 1 ตัวออกทาง**เครื่องพิมพ์ Teletype** แต่ทำไมโปรแกรมถึงแสดงออกทางจอภาพ ?

Teletypewriter (TTY)

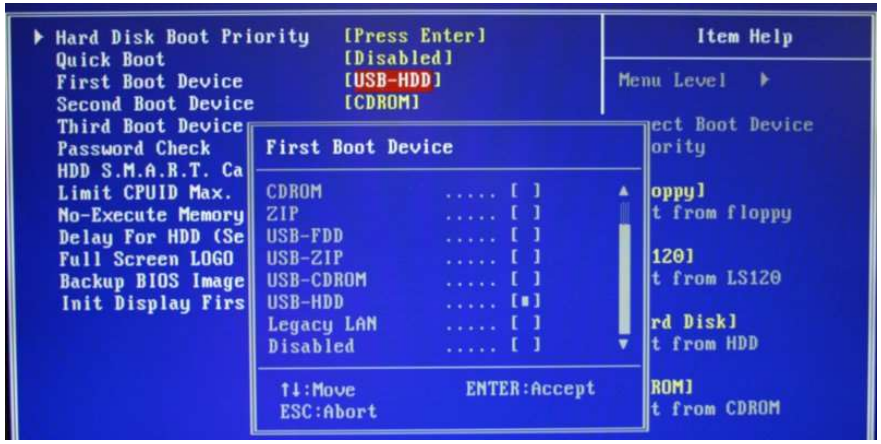
Display Monitor

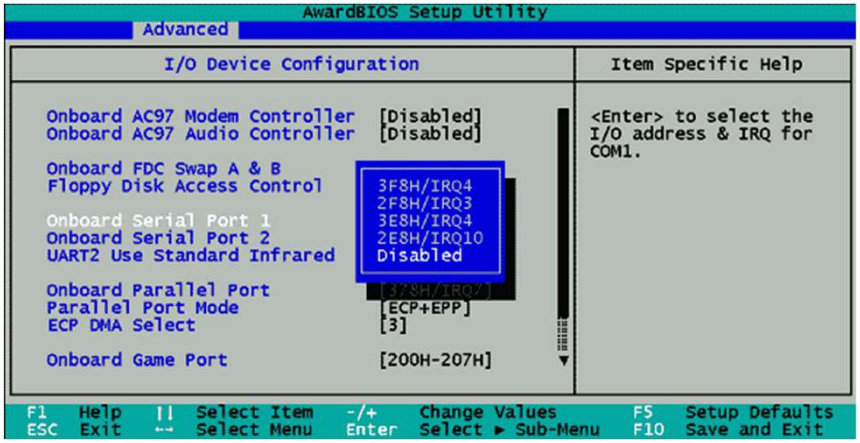
PC BIOS

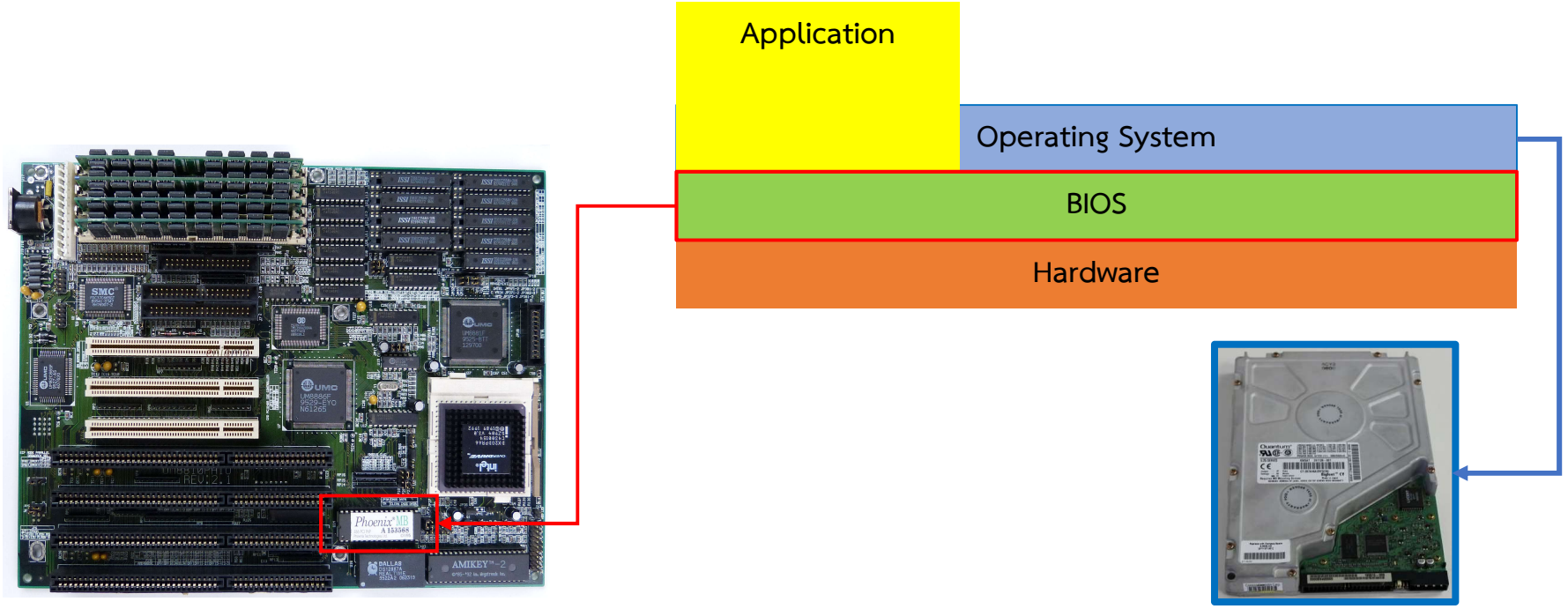
หน้าที่อีกอย่างของ BIOS คือเป็น Hardware Abstraction Layer เพื่อให้ Application สื่อสารกับ Hardware ช่วยให้ Application สามารถทำงานได้บน Hardware ที่แตกต่างกันได้



BIOS จะตรวจสอบว่า Terminal เป็น Teletype หรือ จอภาพ และจะส่งข้อความไปแสดงผล Hardware บางส่วนผู้ใช้สามารถปรับแต่งค่าได้ด้วยผ่านโปรแกรมของ BIOS

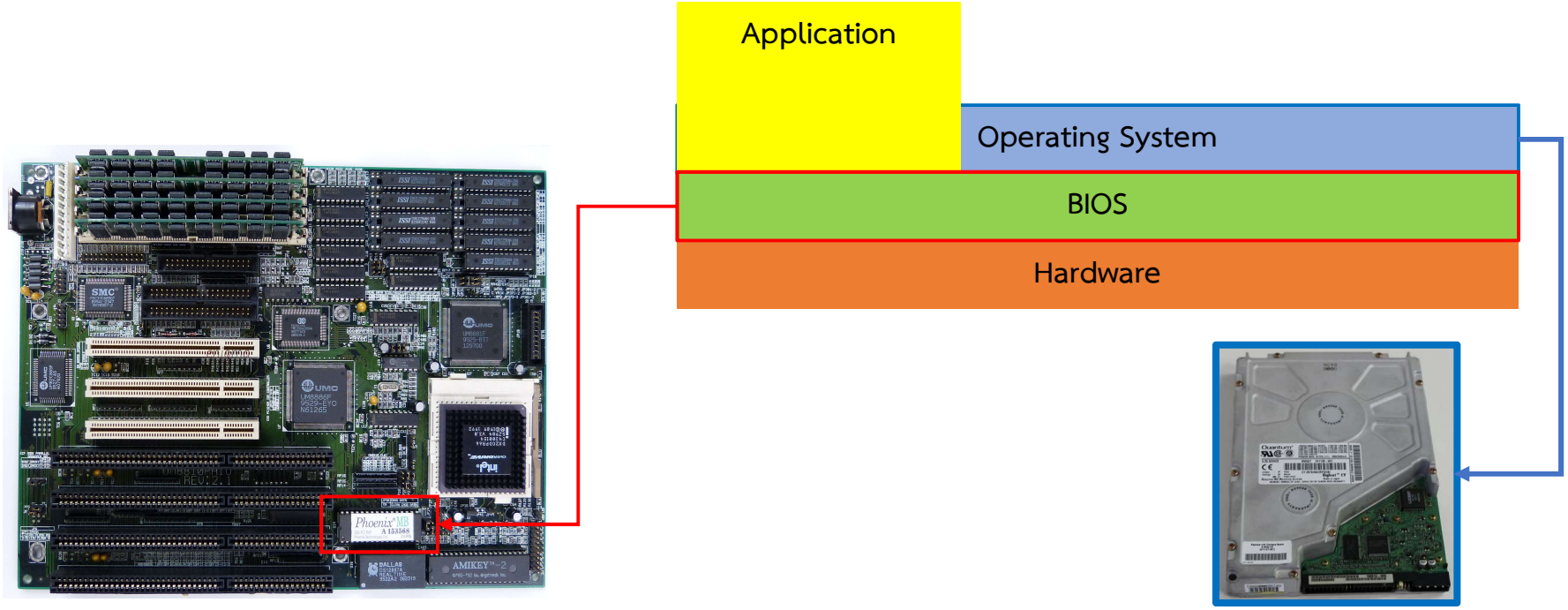

Hard Disk Boot Priority [Press Enter]
Quick Boot [Disabled]
First Boot Device [USB-HDD]
Second Boot Device [CDROM]
Third Boot Device
Password Check
HDD S.M.A.R.T. Ca
Limit CPUID Max.
No-Execute Memory
Delay For HDD (Se
Full Screen LOGO
Backup BIOS Image
Init Display Firs
Item Help
Menu Level
First Boot Device
CDROM
ZIP
USB-FDD
USB-ZIP
USB-CDROM
USB-HDD
Legacy LAN
Disabled
↑↓:Move
ESC:Abort
ENTER:Accept


AwardBIOS Setup Utility
Advanced
I/O Device Configuration
Item Specific Help
Onboard AC97 Modem Controller [Disabled]
Onboard AC97 Audio Controller [Disabled]
Onboard FDC Swap A & B
Floppy Disk Access Control
Onboard Serial Port 1
Onboard Serial Port 2
UART2 Use Standard Infrared
3F8H/IRQ4
2F8H/IRQ3
3E8H/IRQ4
2E8H/IRQ10
Disabled
Onboard Parallel Port
Parallel Port Mode [ECP+EPP]
ECP DMA Select [3]
Onboard Game Port [200H-207H]
<Enter> to select the I/O address & IRQ for COM1.
F1 Help
ESC Exit
Select Item
Select Menu
-/+ Change Values
Enter Select ▸ Sub-Menu
F5 Setup Defaults
F10 Save and Exit


Application
Operating System
BIOS
Hardware

จริง ๆแล้ว BIOS ช่วยในการทำ Hardware Abstraction ได้ในระดับหนึ่ง
แต่หากต้องการเขียน Application ให้รันบน CPU หรือแพลทฟอร์มที่แตกต่างกัน
เช่น IBM/PC XT หรือ IBM/PC AT
การเขียน Application ให้ทำงานในระดับ Operating System จะมีความยืดหยุ่นและโปรแกรมได้ง่ายกว่า

แต่ IBM PC Compatible ทุกเครื่องมี BIOS ที่ Clone หรือ Reverse Engineering มาจากต้นฉบับของ IBM
Application จึงสามารถใช้งาน BIOS ได้โดยไม่ต้องกังวลว่าโปรแกรมจะสามารถรันบนเครื่องอื่นไม่ได้



DOS API

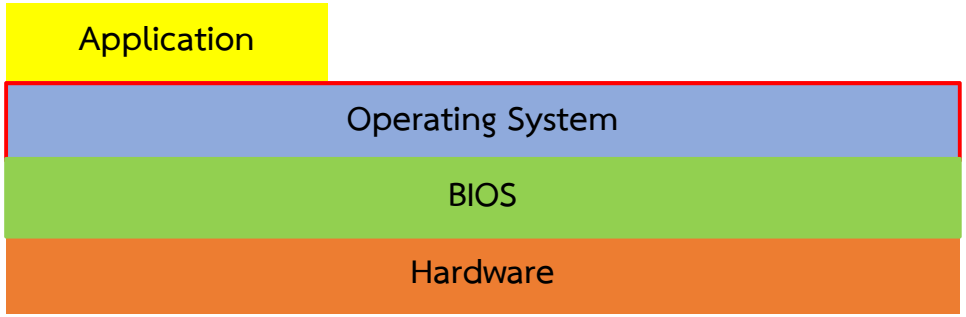
INT 21h / AH=2 - write character to standard output.

entry: DL = character to write, after execution AL = DL.

example:

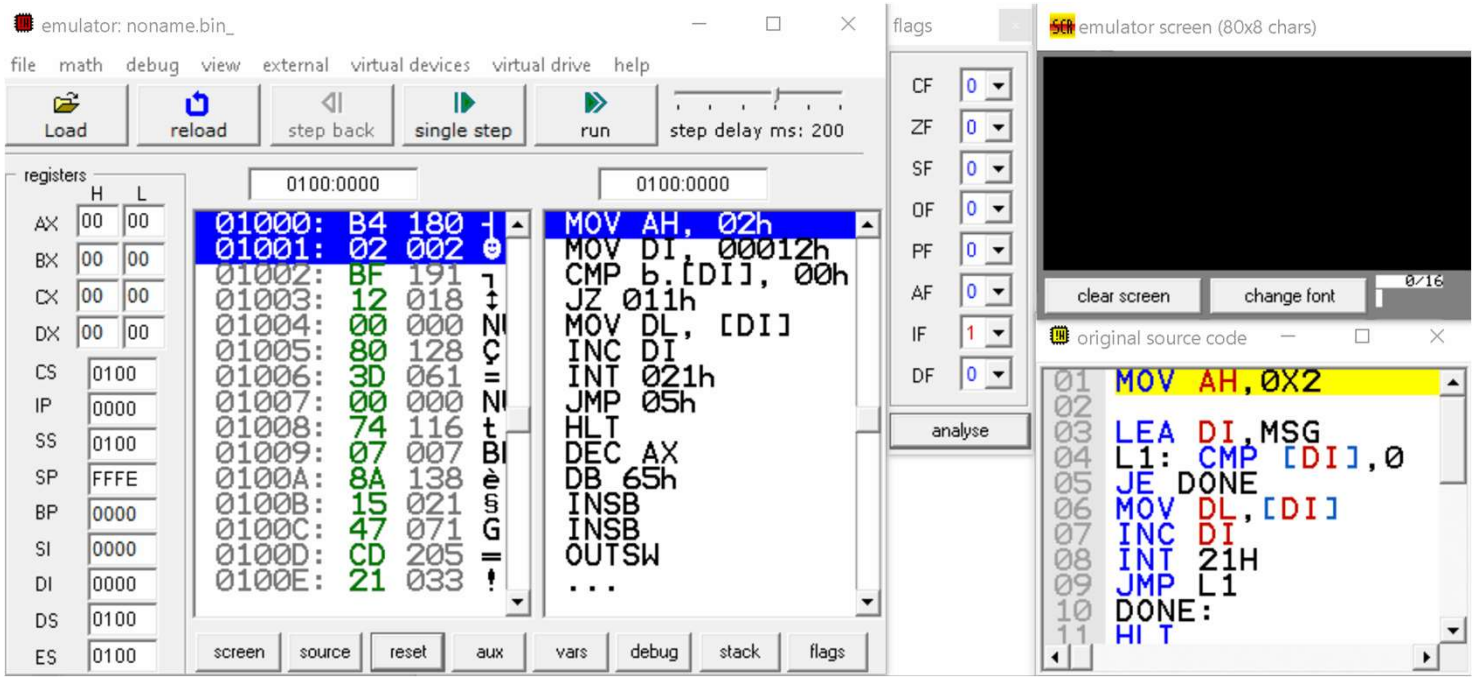
```
mov ah, 2
mov dl, 'a'
int 21h
```

ระบบปฏิบัติการจะช่วยทำ Hardware Abstraction ในระดับที่สูงขึ้น ตัวอย่างเช่น ที่ Layer นี้ จะไม่ใช้คำว่า Teletype แล้ว แต่จะเรียกรวมๆว่า standard output



DOS API

```
MOV AH,0X2

LEA DI,MSG
L1: CMP [DI],0
    JE DONE
    MOV DL,[DI]
    INC DI
    INT 21H
    JMP L1
DONE:
HLT

MSG DB "Hello World!",0
```



DOS API

INT 21h / AH=9 - output of a string at DS:DX. String must be terminated by '$'.

example:

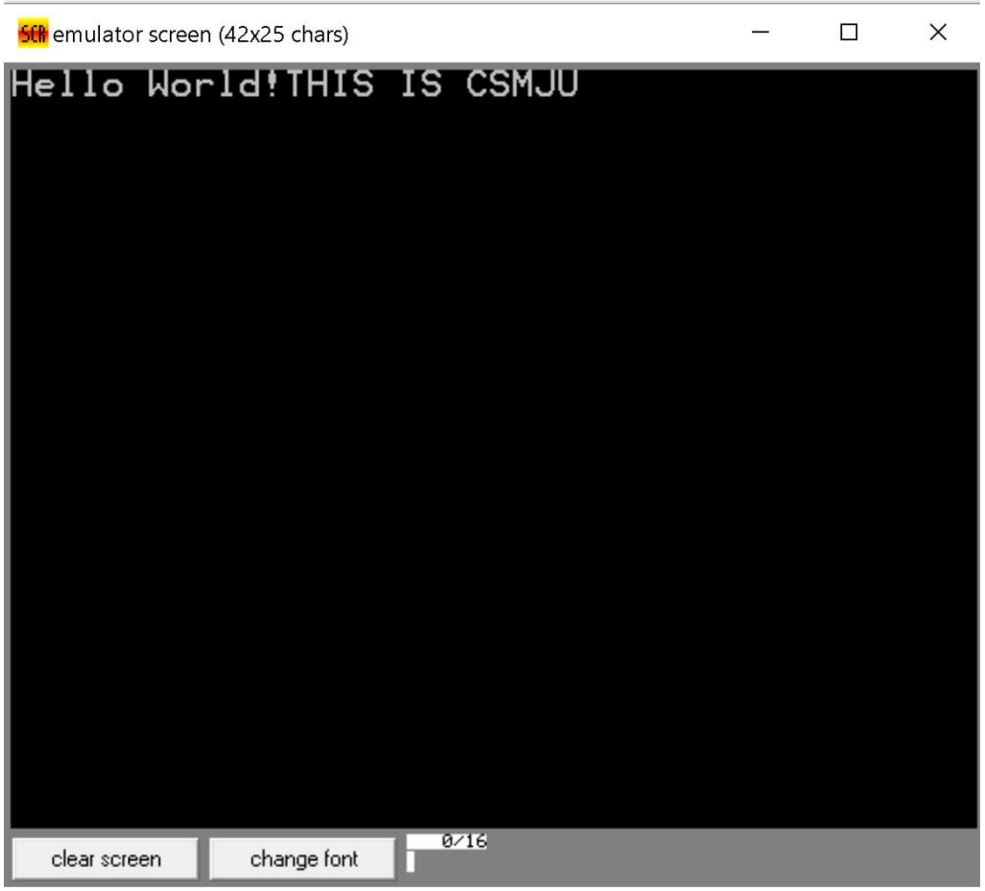
```
org 100h
mov dx, offset msg
mov ah, 9
int 21h
ret
msg db "hello world $"
```

DOS API

```
MOV AX,CS     ;
MOV DS,AX     ; DS=CS


MOV AH,0x09
LEA DX,MSG
INT 21h

HLT

MSG DB "Hello World!$"
```

DOS API

```
MOV AX,CS     ;
MOV DS,AX     ; DS=CS

MOV AH,0x09

LEA DX,MSG
INT 21h

LEA DX,MSG2
INT 21h

HLT

MSG DB "Hello World!$"
MSG2 DB "THIS IS CSMJU$ ABCD"
```



Standard Output Abstraction

INT 21H / 09H

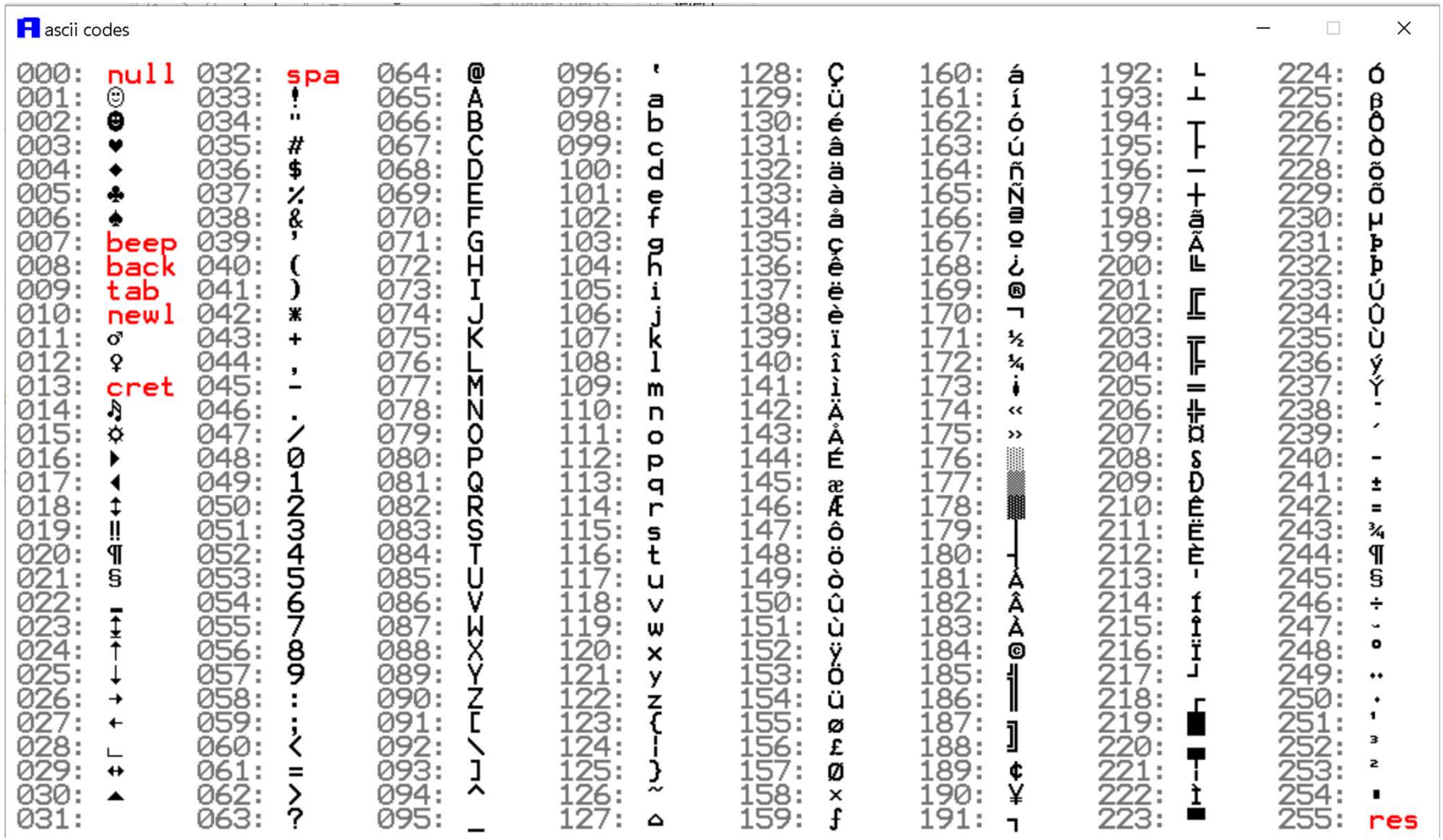
Standard Output Hardware Abstraction

Display Monitor

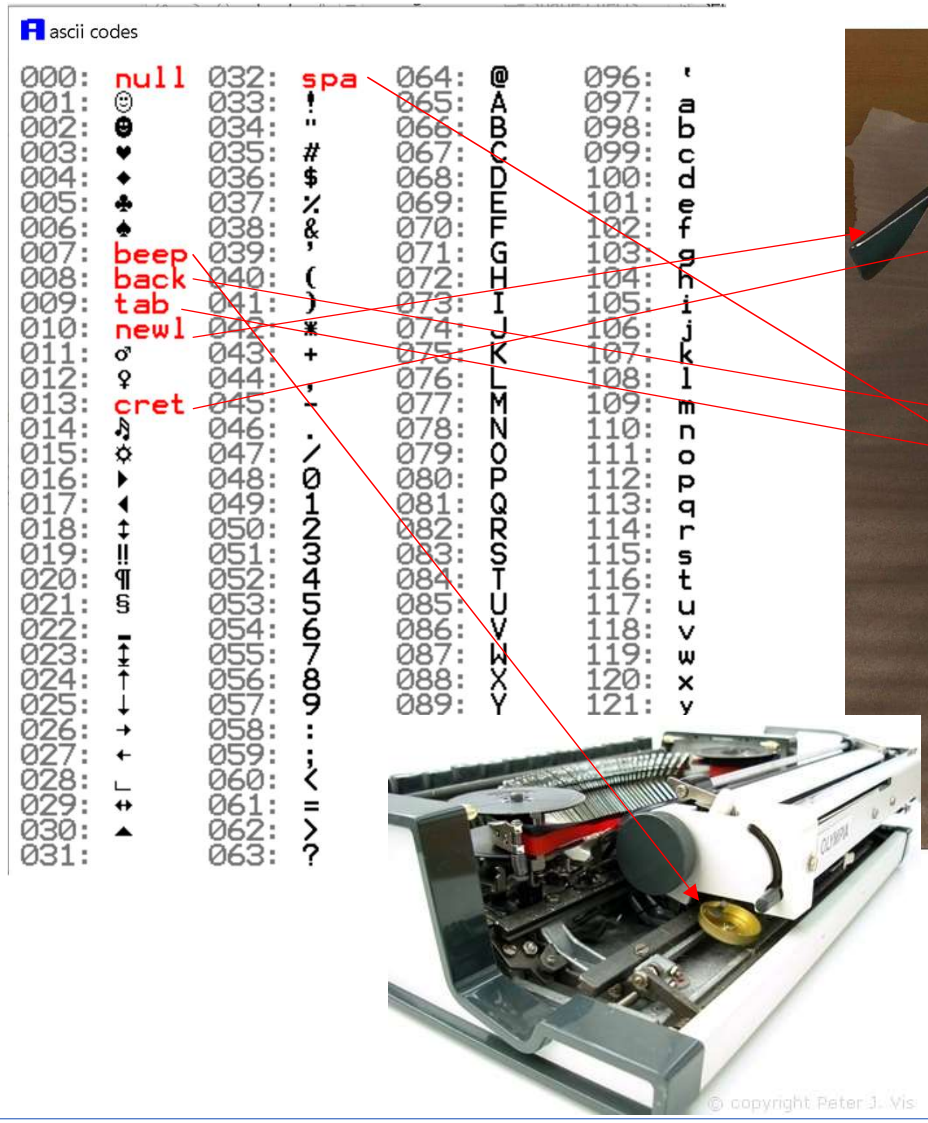
Teletypewriter

เครื่องพิมพ์ดีด

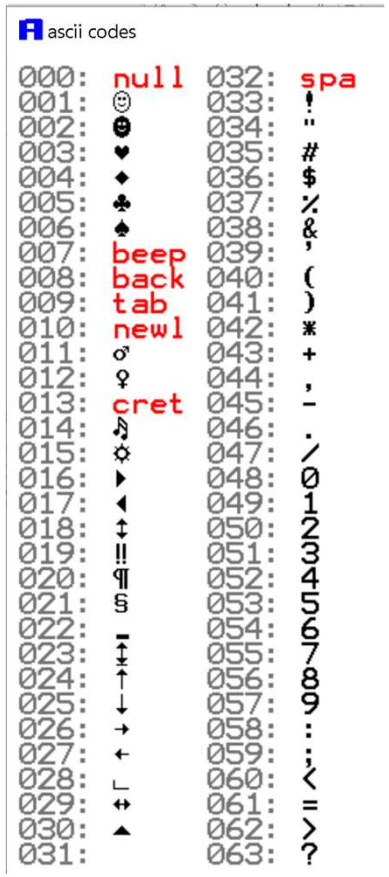
Standard Output Abstraction



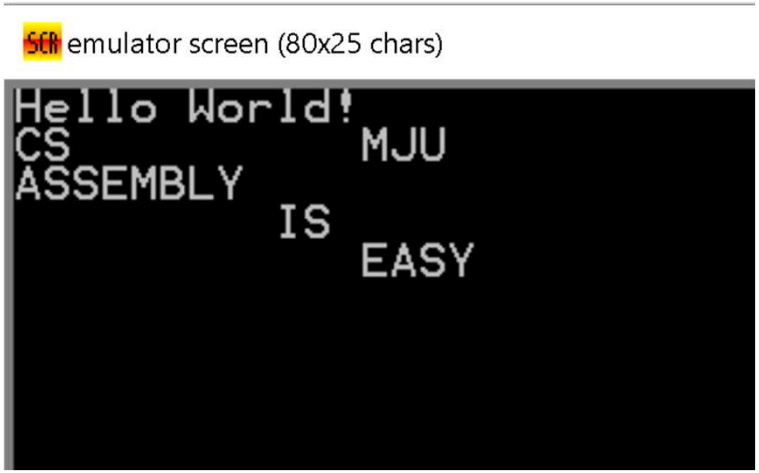
Standard Output Abstraction



Standard Output Abstraction




```
MOV AX,CS     ;
MOV DS,AX     ; DS=CS

MOV AH,0x09
LEA DX,MSG
INT 21h
HLT
MSG DB "Hello",32,"World!"
    DB 10,"MJU",13,"CS"
    DB 10,13,"ASSEMBLY",10," IS "
    DB 10,"EASY",07,"$"
```



การสร้าง ISR (Interrupt Service Routine)

ลองสร้าง ISR ขึ้นมาเองที่ทำงานเหมือน DOS Standard Output Print String แต่เปลี่ยนรหัส Terminate จาก $ เป็น NULL

1) สร้างโปรแกรมย่อย และทดสอบ
2) เลือกชนิด Interrupt
3) คำนวณ Address สำหรับ Interrupt Vector
4) เพิ่มข้อมูลลง Interrupt Vector

การสร้าง ISR (Interrupt Service Routine)

1) สร้างโปรแกรมย่อย และทดสอบ

Input : DS:DX ตำแหน่งหน่วยความจำที่ต้องการแสดงสายอักขระบนจอภาพ

การสร้าง ISR หากต้องการเปิดให้โปรแกรมอื่นมาใช้งานร่วมได้ โปรแกรมย่อยที่สร้างควรมีการรักษาค่าของรีจิสเตอร์และ Flag ไว้

เมื่อมีการเรียกใช้งานโปรแกรมย่อยด้วยวิธี INT จะมีการเก็บค่า Flag ลงใน Stack ด้วย ต้องระวัง!

ต้องเป็นโปรแกรมย่อยที่มีการเรียกใช้แบบ FAR เพราะ Interrupt Vector มีการเก็บทั้ง Segment และ Offset

การสร้าง ISR (Interrupt Service Routine)

1) สร้างโปรแกรมย่อย และทดสอบ

```
MOV AX,CS
MOV DS,AX
LEA DX,MSG
PUSHF
MOV off_printf, Printf
MOV seg_printf, seg Printf
CALL FAR off_printf
HLT
MSG DB "Hello World!",0
off_printf dw ?
seg_printf dw ?
```

Code ทดสอบโปรแกรมย่อย

```
Printf PROC FAR
    POP CX ; preserving IP
    POP BX ; preserving CS
    PUSHA  ; preserving Registers
    MOV AH,0XE
    MOV DI,DX
L1: CMP [DI],0
    JE DONE
    MOV AL,[DI]
    INC DI
    INT 10H
    JMP L1
DONE:
   POPA   ; Restoring Registers
   POPF   ; Restoring Flags
   PUSH BX; restirong CS
   PUSH CX; restoring IP
   RET
Printf ENDP
```

บันทึกค่ารีจิสเตอร์ และตำแหน่งกลับ

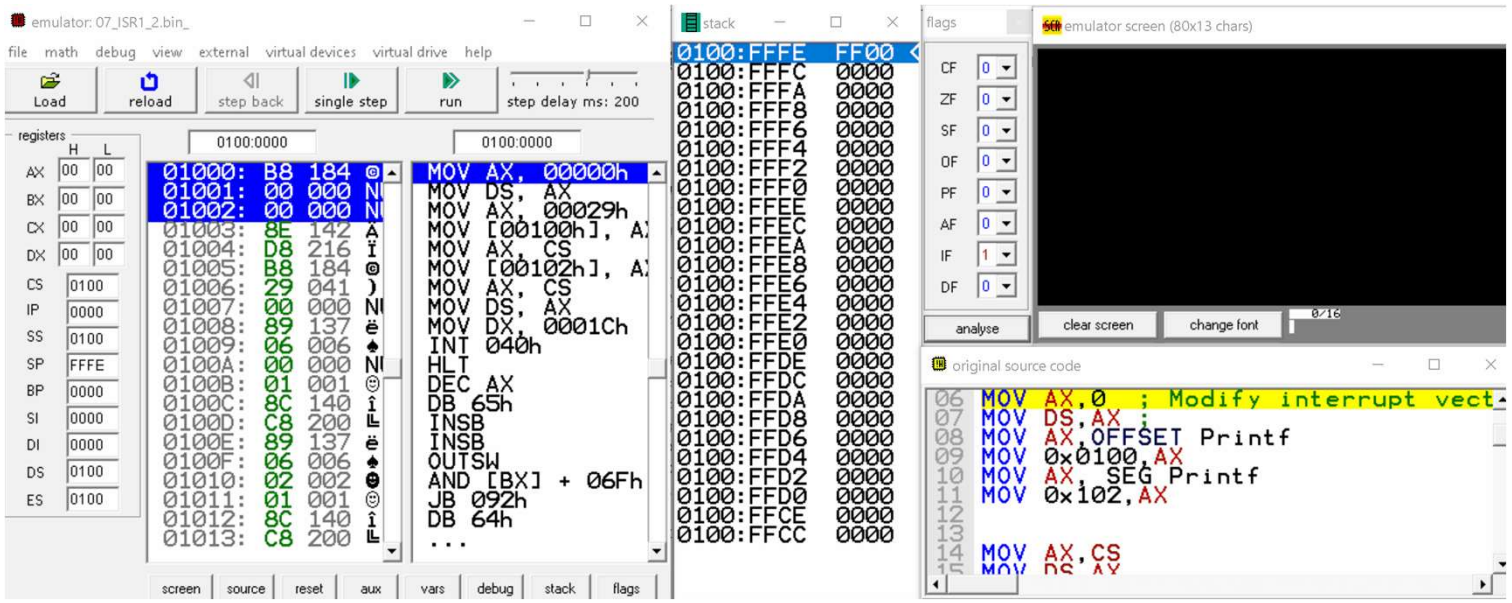
คืนค่ารีจิสเตอร์ , Flag และตำแหน่งกลับ

Code โปรแกรมย่อย

การสร้าง ISR (Interrupt Service Routine)

1) สร้างโปรแกรมย่อย และทดสอบ

```
MOV AX,CS
MOV DS,AX
LEA DX,MSG
PUSHF
MOV off_printf, Printf
MOV seg_printf, seg Printf
CALL FAR off_printf
HLT
MSG DB "Hello World!",0
off_printf dw ?
seg_printf dw ?

Printf PROC FAR
    POP CX ; preserving IP
    POP BX ; preserving CS
    PUSHA  ; preserving Registers
    MOV AH,0XE
    MOV DI,DX
L1: CMP [DI],0
    JE DONE
    MOV AL,[DI]
    INC DI
    INT 10H
    JMP L1
DONE:
   POPA   ; Restoring Registers
   POPF   ; Restoring Flags
   PUSH BX; restirong CS
   PUSH CX; restoring IP
   RET
Printf ENDP
```



การสร้าง ISR (Interrupt Service Routine)

2) เลือกชนิด Interrupt

8086 สามารถสร้าง Interrupt ได้ 256 แบบ (00 – FF) แต่มีบางตัวถูกใช้งานไปแล้ว

ดังนั้นจึงมี Interrupt ให้เราใช้งานได้ในช่วง

20H - FFH (223 แบบ)

หากพัฒนาโปรแกรมบน MS-DOS ให้งดใช้งาน 20H,21H,33H

ตัวอย่างนี้จะเลือกใช้เบอร์ 40H เพราะยังไม่ได้ถูกใช้งาน

การสร้าง ISR (Interrupt Service Routine)

3) คำนวณ Address สำหรับ Interrupt Vector

ตัวอย่างนี้จะเลือกใช้เบอร์ 40H

n= 40H

IP = 4 x n = 4 x 40H = 100H

CS= IP + 2 = 100H +2 = 102H

การสร้าง ISR (Interrupt Service Routine)

4) เพิ่มข้อมูลลง Interrupt Vector

เลือกหน่วยความจำที่ Segment 0

```
MOV AX,0  ; Modify interrupt vector
MOV DS,AX ;
```

บันทึกตำแหน่งโปรแกรมย่อย ลงใน Interrupt Vector

```
MOV AX,OFFSET Printf
MOV 0x0100,AX
MOV AX, SEG Printf
MOV 0x102,AX
```

ตัวอย่างนี้จะเลือกใช้เบอร์ 40H

n= 40H

IP = 4 x n = 4 x 40H = 100H

CS= IP + 2 = 100H +2 = 102H

ทดสอบใช้งาน Interrupt 40H

```
MOV AX,CS
MOV DS,AX
LEA DX,MSG
INT 40h
HLT
MSG DB "Hello World!",0
```

โปรแกรมย่อยที่ผ่านการทดสอบแล้ว

```
Printf PROC FAR
   POP CX ; preserving IP
   POP BX ; preserving CS
   PUSHA  ; preserving Registers
   MOV AH,0XE
   MOV DI,DX
L1: CMP [DI],0
   JE DONE
   MOV AL,[DI]
   INC DI
   INT 10H
   JMP L1
DONE:
   POPA   ; Restoring Registers
   POPF   ; Restoring Flags
   PUSH BX; restirong CS
   PUSH CX; restoring IP
   RET
Printf ENDP
```

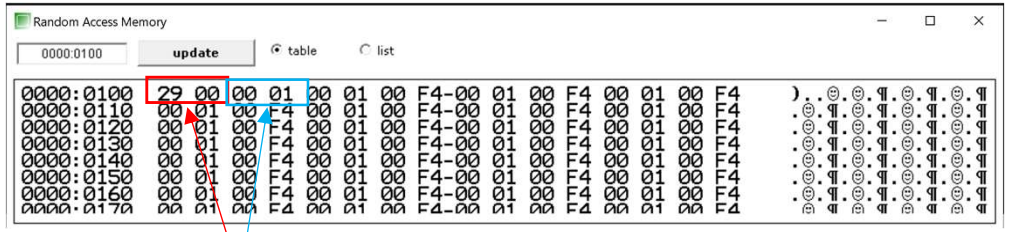
การสร้าง ISR (Interrupt Service Routine)

เลือกหน่วยความจำที่ Segment 0

```
MOV AX,0  ; Modify interrupt vector
MOV DS,AX ;
```

บันทึกตำแหน่งโปรแกรมย่อย ลงใน Interrupt Vector

```
MOV AX,OFFSET Printf
MOV 0x0100,AX
MOV AX, SEG Printf
MOV 0x102,AX
```



ทดสอบใช้งาน Interrupt 40H

```
MOV AX,CS
MOV DS,AX
LEA DX,MSG
INT 40h
HLT
MSG DB "Hello World!",0
```

CS:IP ของโปรแกรมย่อย Printf

โปรแกรมย่อยที่ผ่านการทดสอบแล้ว

```
Printf PROC FAR
   POP CX ; preserving IP
   POP BX ; preserving CS
   PUSHA  ; preserving Registers
   MOV AH,0XE
   MOV DI,DX
L1: CMP [DI],0
   JE DONE
   MOV AL,[DI]
   INC DI
   INT 10H
   JMP L1
DONE:
  POPA   ; Restoring Registers
  POPF   ; Restoring Flags
  PUSH BX; restirong CS
  PUSH CX; restoring IP
  RET
Printf ENDP
```

การสร้าง ISR (Interrupt Service Routine)

เลือกหน่วยความจำที่ Segment 0

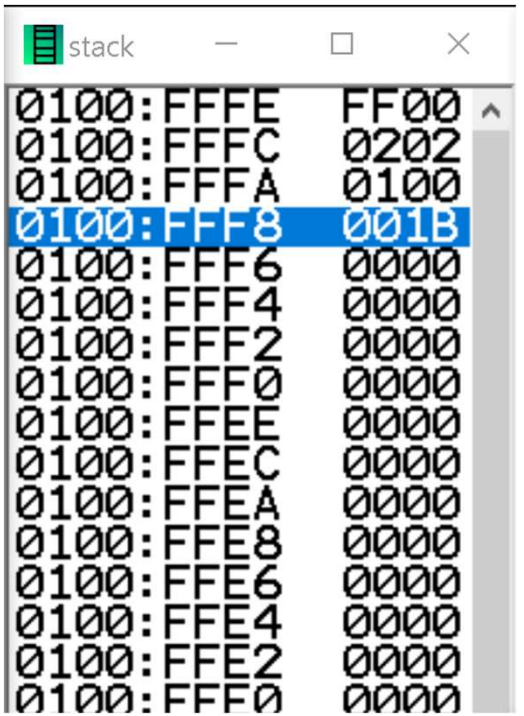
```
MOV AX,0  ; Modify interrupt vector
MOV DS,AX ;
```

บันทึกตำแหน่งโปรแกรมย่อย ลงใน Interrupt Vector

```
MOV AX,OFFSET Printf
MOV 0x0100,AX
MOV AX, SEG Printf
MOV 0x102,AX
```

ทดสอบใช้งาน Interrupt 40H

```
MOV AX,CS
MOV DS,AX
LEA DX,MSG
INT 40h
HLT
MSG DB "Hello World!",0
```



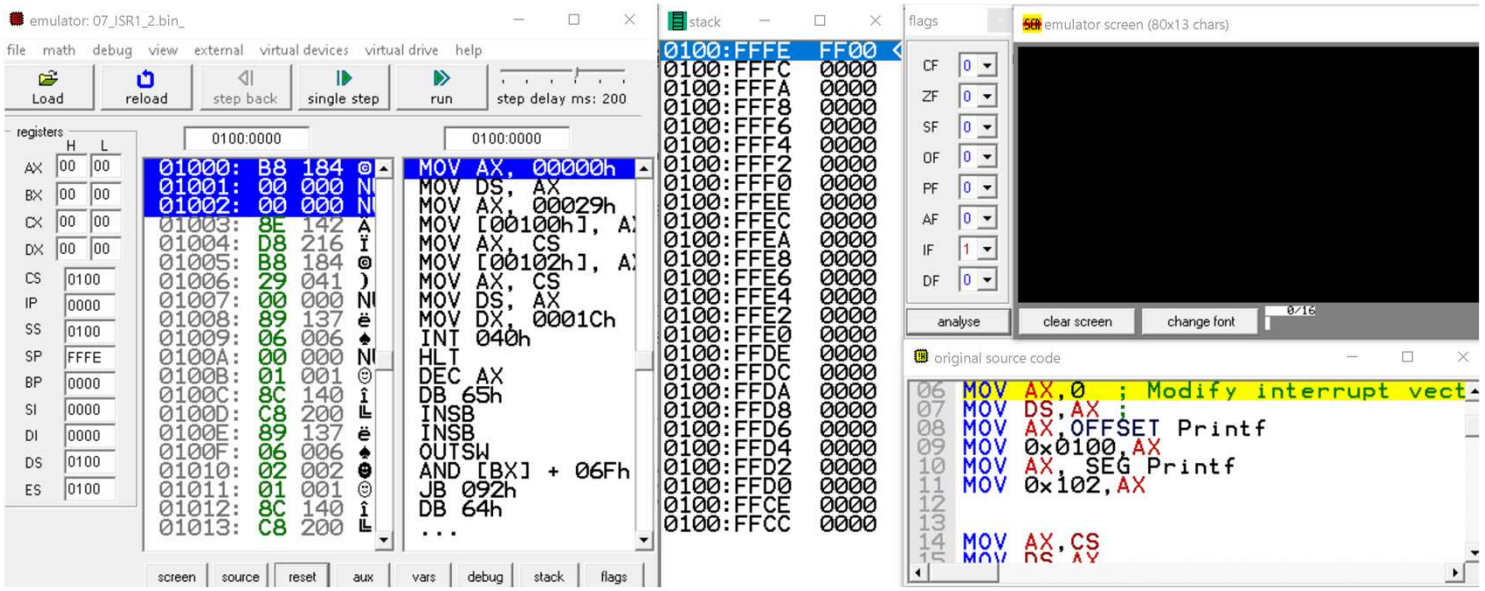
Flag
CS กลับ
IP กลับ

โปรแกรมย่อยที่ผ่านการทดสอบแล้ว

```
Printf PROC FAR
   POP CX ; preserving IP
   POP BX ; preserving CS
   PUSHA  ; preserving Registers
   MOV AH,0XE
   MOV DI,DX
L1: CMP [DI],0
   JE DONE
   MOV AL,[DI]
   INC DI
   INT 10H
   JMP L1
DONE:
  POPA   ; Restoring Registers
  POPF   ; Restoring Flags
  PUSH BX; restirong CS
  PUSH CX; restoring IP
  RET
Printf ENDP
```

การสร้าง ISR (Interrupt Service Routine)

```
MOV AX,0  ; Modify interrupt vector
MOV DS,AX ;
MOV AX,OFFSET Printf
MOV 0x0100,AX
MOV AX, SEG Printf
MOV 0x102,AX

MOV AX,CS
MOV DS,AX
LEA DX,MSG
INT 40h
HLT
MSG DB "Hello World!",0
```




```
Printf PROC FAR
   POP CX ; preserving IP
   POP BX ; preserving CS
   PUSHA  ; preserving Registers
   MOV AH,0XE
   MOV DI,DX
L1: CMP [DI],0
   JE DONE
   MOV AL,[DI]
   INC DI
   INT 10H
   JMP L1
DONE:
   POPA   ; Restoring Registers
   POPF   ; Restoring Flags
   PUSH BX; restirong CS
   PUSH CX; restoring IP
   RET
Printf ENDP
```

เพื่อให้ง่าย โปรแกรมนี้เก็บ CS:IP ไว้ในรีจิสเตอร์ วิธีที่ดีควรเก็บในแรม

การสร้าง ISR (Interrupt Service Routine)

```
MOV AX,0  ; Modify interrupt vector
MOV DS,AX ;
MOV AX,OFFSET Printf
MOV 0x0100,AX
MOV AX, SEG Printf
MOV 0x102,AX

MOV AX,CS
MOV DS,AX
LEA DX,MSG
INT 40h
HLT
MSG DB "Hello World!",0
```

```
Printf PROC FAR
    POP CX ; preserving IP
    POP BX ; preserving CS
    PUSHA  ; preserving Registers
    MOV AH,0XE
    MOV DI,DX
L1: CMP [DI],0
    JE DONE
    MOV AL,[DI]
    INC DI
    INT 10H
    JMP L1
DONE:
   POPA   ; Restoring Registers
   POPF   ; Restoring Flags
   PUSH BX; restirong CS
   PUSH CX; restoring IP
   RET
Printf ENDP
```

```
Printf PROC FAR
    POP CX ; preserving IP
    POP BX ; preserving CS
    PUSHA  ; preserving Registers
    MOV AH,0XE
    MOV DI,DX
L1: CMP [DI],0
    JE DONE
    MOV AL,[DI]
    INC DI
    INT 10H
    JMP L1
DONE:
   POPA   ; Restoring Registers
   POPF   ; Restoring Flags
   PUSH BX; restirong CS
   PUSH CX; restoring IP
   RET
Printf ENDP
```

การคืนค่า Flag และกลับสู่โปรแกรมหลัก
สามารถทำได้ในคำสั่งเดียวคือ Interrupt Return
IRET
หากจบโปรแกรมย่อยด้วย IRET
ให้ตัด POPF ออกจากโปรแกรมได้เลย

การสร้าง ISR (Interrupt Service Routine)

ระบบปฏิบัติการ MS-DOS และโปรแกรมประยุกต์บนระบบปฏิบัติการ MS-DOS ใช้วิธีนี้ในการสร้าง Device Driver

Driver บางชนิดระบบปฏิบัติการเป็นผู้โหลดเข้าสู่หน่วยความจำเอง ตัวอย่างเช่น

- **CDROM**
- **Memory**
- **Display**

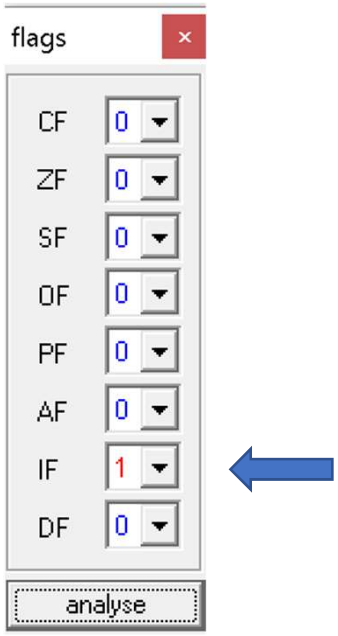
Driver บางชนิดจะโหลดเข้าสู่หน่วยความจำได้ด้วยตัวเอง (TSR) ตัวอย่างเช่น

- **Mouse**
- **Audio player**
- **Thai Font**

Interrupt ถูกควบคุมการทำงานด้วย IF

IF = 1 ; Enable

IF = 0 ; Disable



หากไม่ต้องการให้มีการตรวจจับสัญญาณ Interrupt ทำได้ด้วยการตั้ง IF เป็น 0 ด้วยคำส่ง CLI : Clear Interrupt Enable Flag

หากต้องการให้มีการตรวจจับสัญญาณ Interrupt ทำได้ด้วยการตั้ง IF เป็น 1 ด้วยคำส่ง STI : Set Interrupt Enable Flag

ระบบ Interrupt ของ 8086

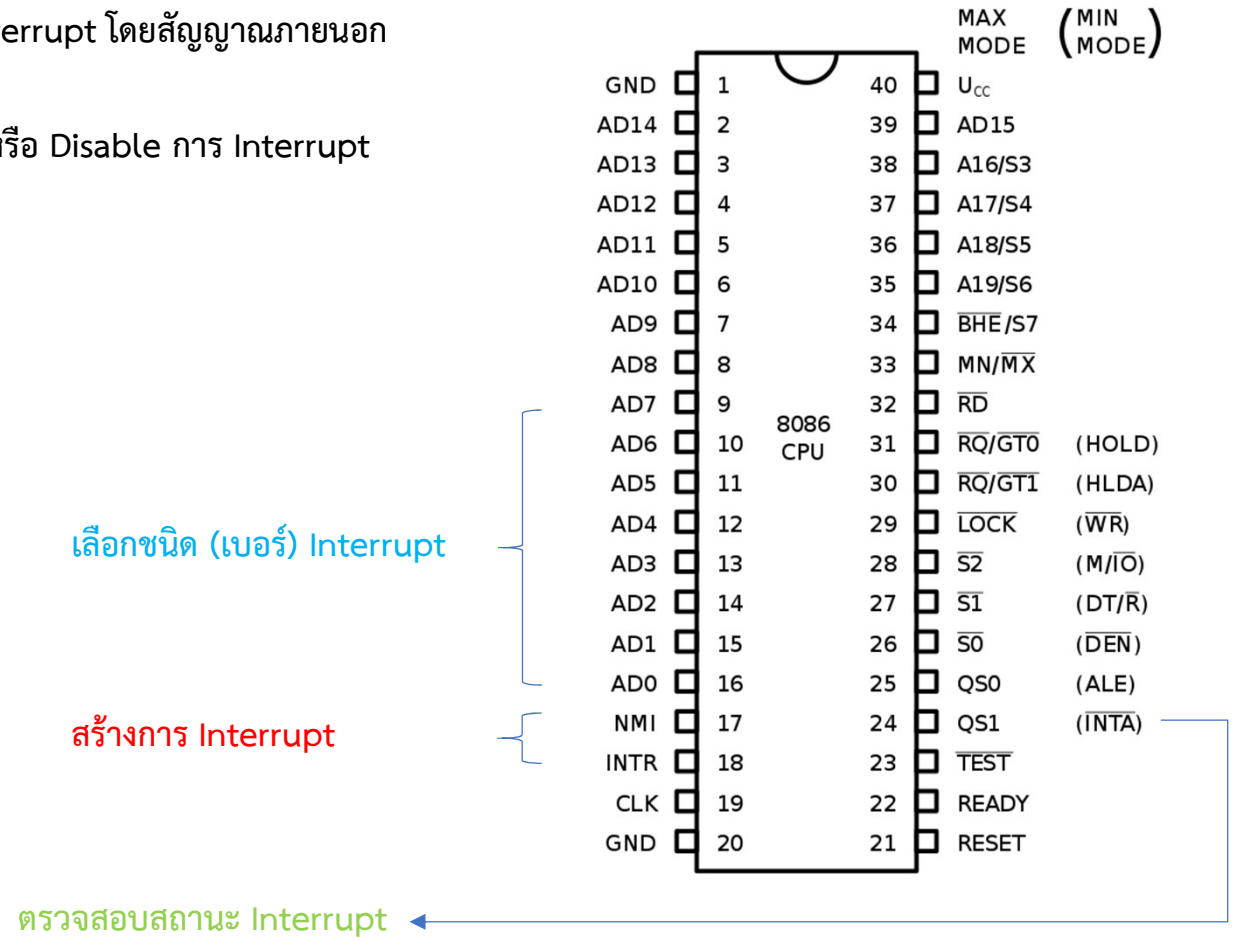
- Hardware Interrupt
  - Maskable (INTR) รองรับ 256 แบบ
  - Non-maskable (NMI) รองรับแบบที่ 2 เท่านั้น
- Software Interrupt รองรับ 256 แบบ

Hardware Interrupt

Hardware Interrupt จะเป็นการสร้างการ Interrupt โดยสัญญาณภายนอก

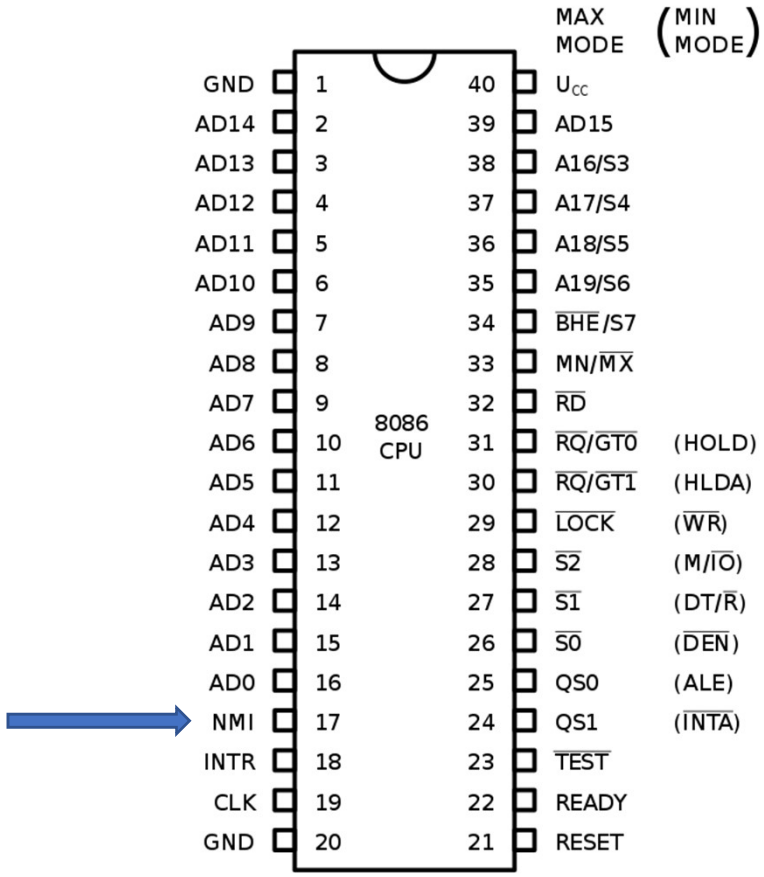
Pin D0 – D7 ใช้เลือกหมายเลข

NMI / INTR / INTA ตรวจสอบการ Enable หรือ Disable การ Interrupt และใช้สร้างการ Interrupt



Hardware Interrupt

การสร้าง Non-maskable interrupt ทำได้โดยการ เปลี่ยนระดับสัญญาณ
จาก Low เป็น High ที่ NMI
จะเกิด Interrupt แบบที่ 2 ขึ้นทันทีหลังเสร็จสิ้น operation ที่กำลังทำอยู่



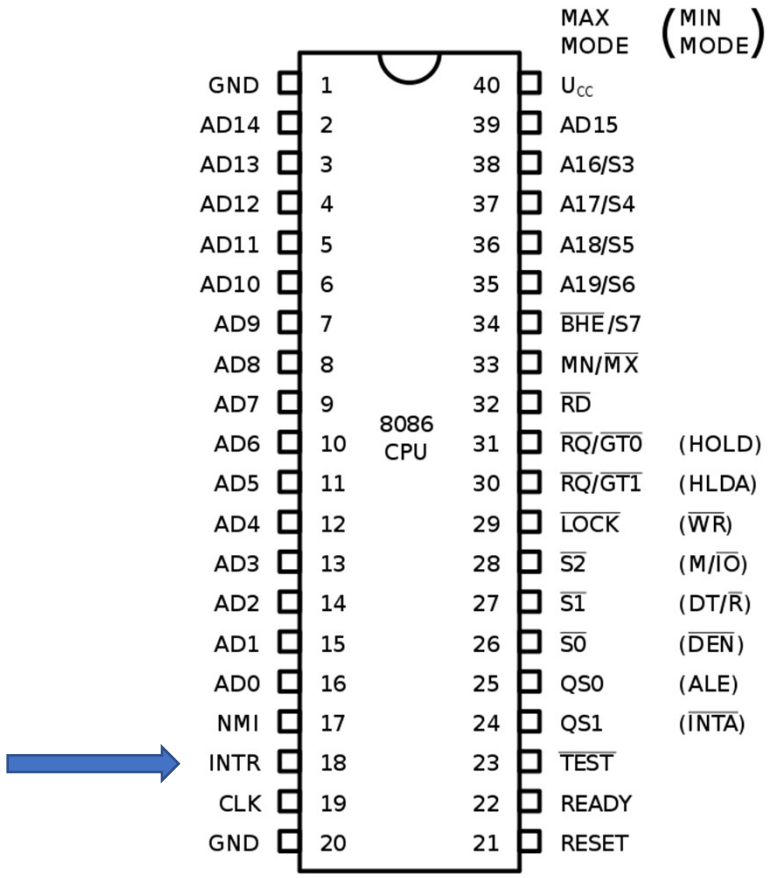
Hardware Interrupt

การสร้าง Maskable interrupt จะมีขั้นตอนเพิ่มขึ้นมาเพราะว่า

- Disable จากโปรแกรมได้
- เลือกชนิดได้

ขั้นตอนการ Interrupt แบบ Maskable

1) ส่งสัญญาณ High เข้า INTR (Interrupt Request)



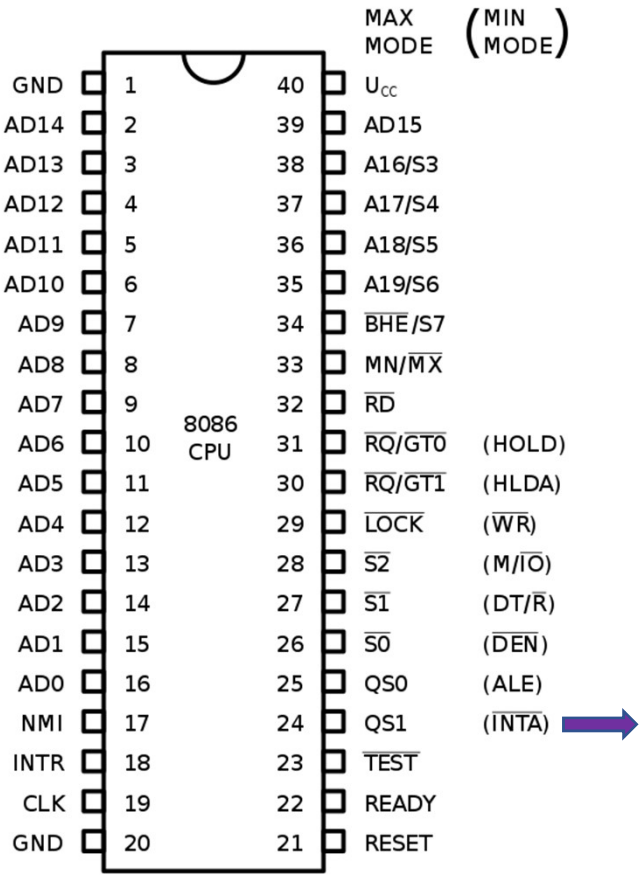
Hardware Interrupt

การสร้าง Maskable interrupt จะมีขั้นตอนเพิ่มขึ้นมาเพราะว่า

- Disable จากโปรแกรมได้
- เลือกชนิดได้

ขั้นตอนการ Interrupt แบบ Maskable

2) รออ่านสัญญาณที่ INTA (Interrupt Acknowledge)
หากการ Interrupt ไม่ได้ถูกDisable โดยโปรแกรมและ
ไมโครโพรเซสเซอร์พร้อมที่จะถูกขัดจังหวะ
สัญญาณที่ INTA จะเปลี่ยนดเป็น LOW



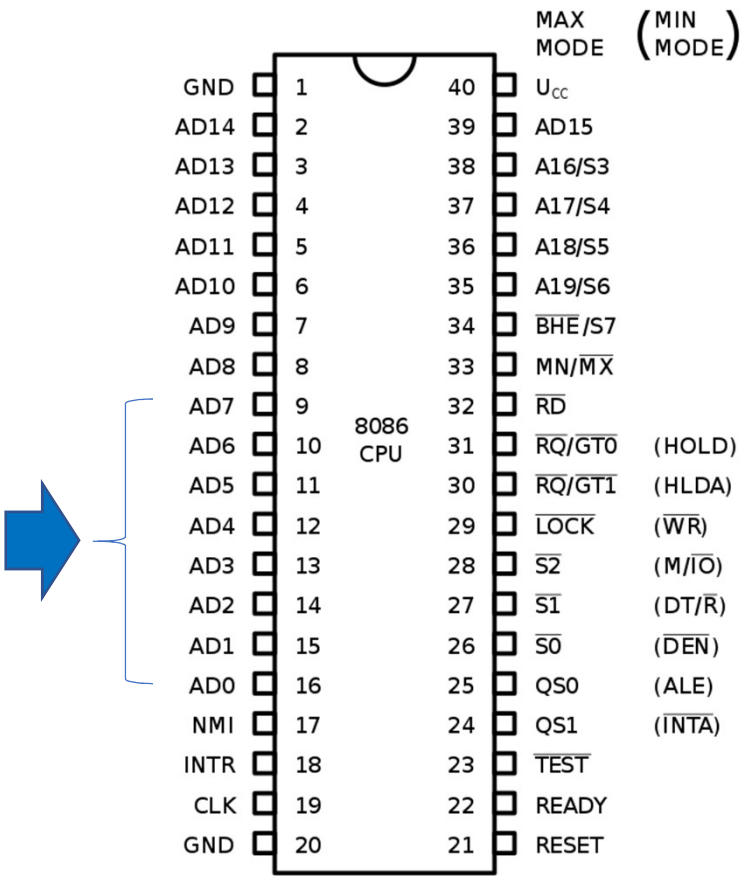
Hardware Interrupt

การสร้าง Maskable interrupt จะมีขั้นตอนเพิ่มขึ้นมาเพราะว่า

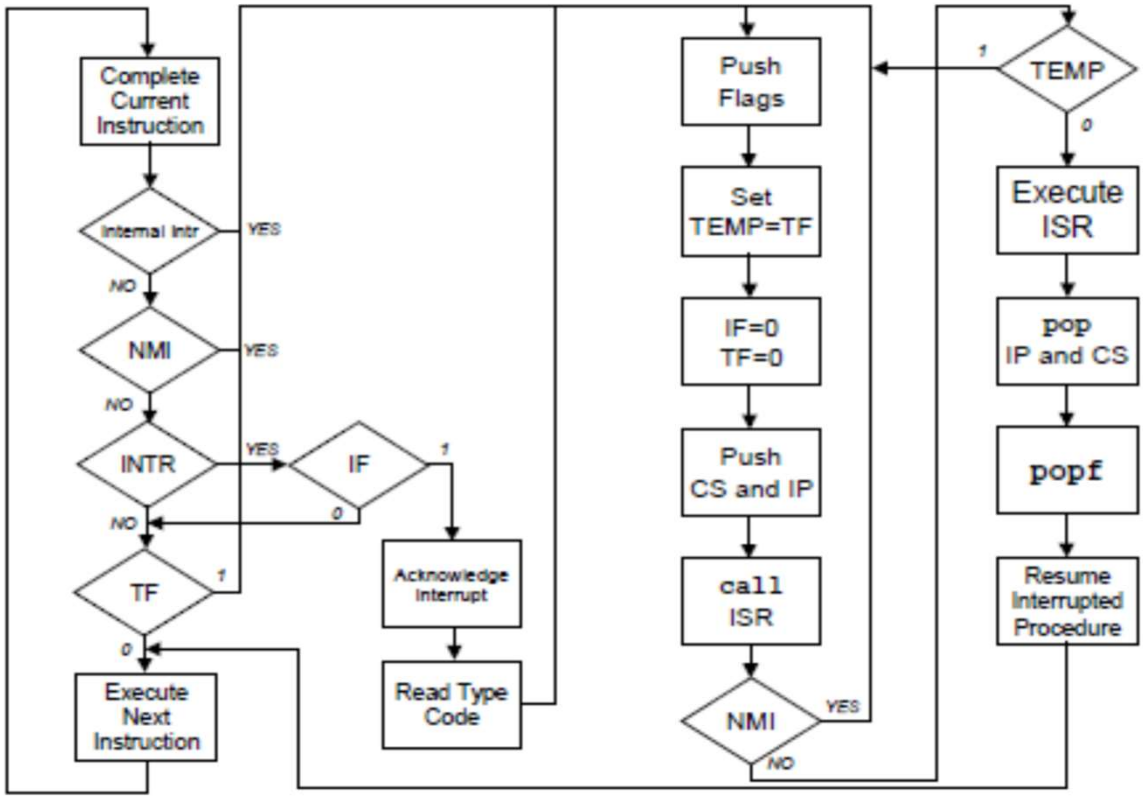
- Disable จากโปรแกรมได้
- เลือกชนิดได้

ขั้นตอนการ Interrupt แบบ Maskable

3) ป้อนชนิด หรือ เบอร์ของ Interrupt ทาง D0 – D7

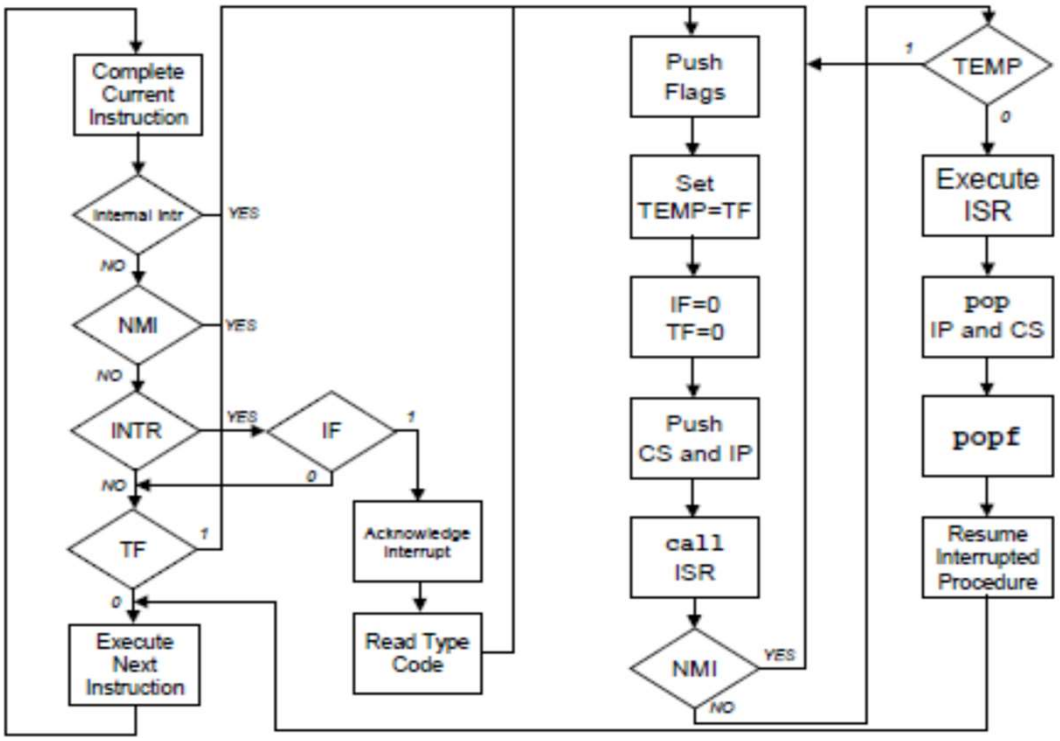


8086 Interrupt handle



http://www.uobabylon.edu.iq/eprints/publication_12_20160_1426.pdf

8086 Interrupt handle



ลำดับความสำคัญ
เรียงจากสูงสุดไปต่ำสุด

1) Software Interrupts , default
2) Non-maskable (NMI)
3) Maskable (INTR)
4) Single Step (TF)

Hardware Interrupt

INTR ให้เลือก 256 แบบ แต่มี INTR อยู่แค่เส้นเดียว

หากต้องการให้

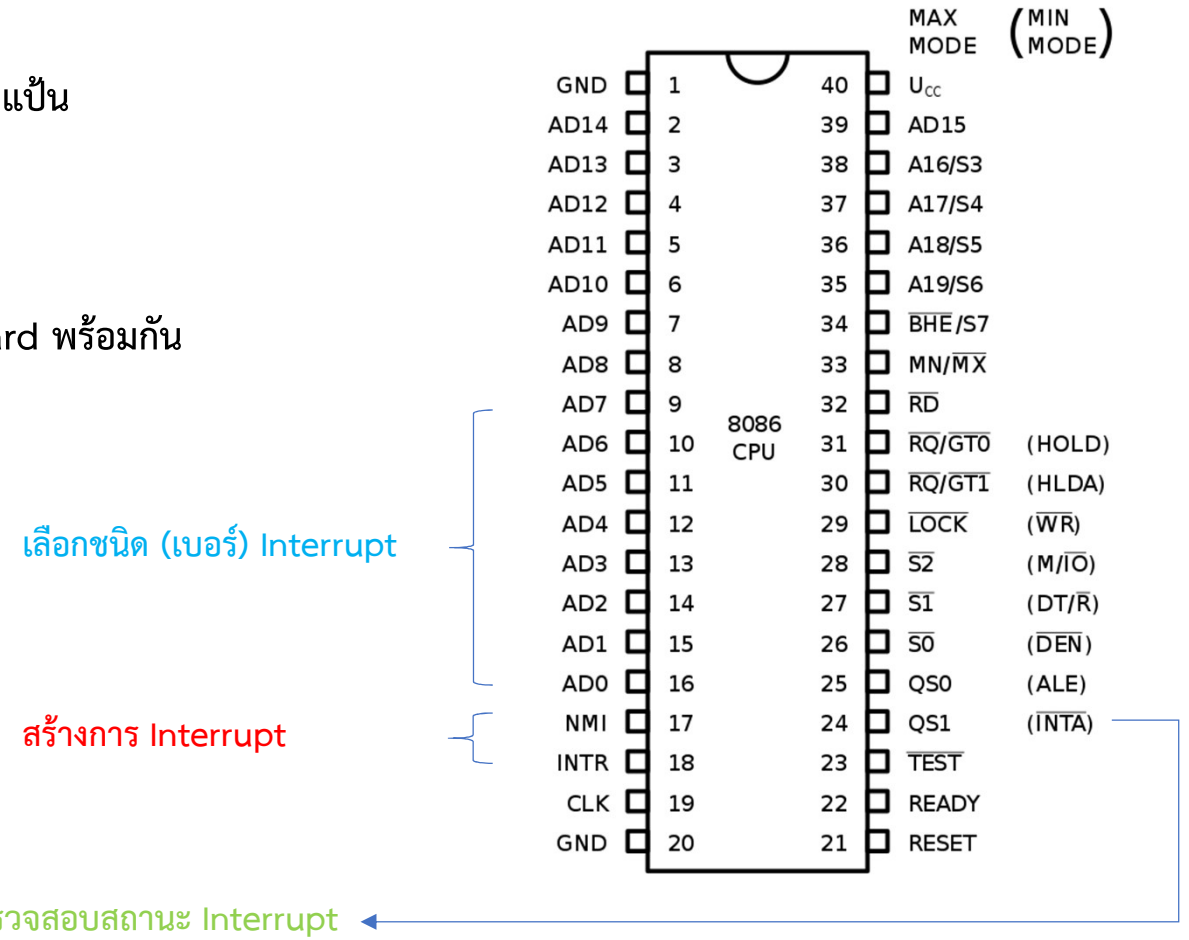
Keyboard เกิด INT 50H เมื่อมีการกดแป้น

Mouse เกิด INT 51H เมื่อมีการขยับ

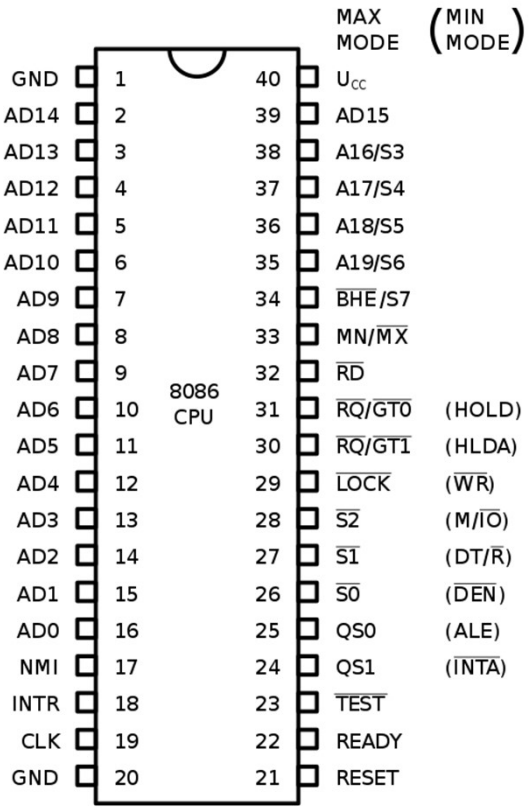
จะต่อสายไฟอย่างไร ?

หาก คลิ๊ก Mouse และ เคาะ Keyboard พร้อมกัน

จะเกิด Interrupt ตัวไหน ?



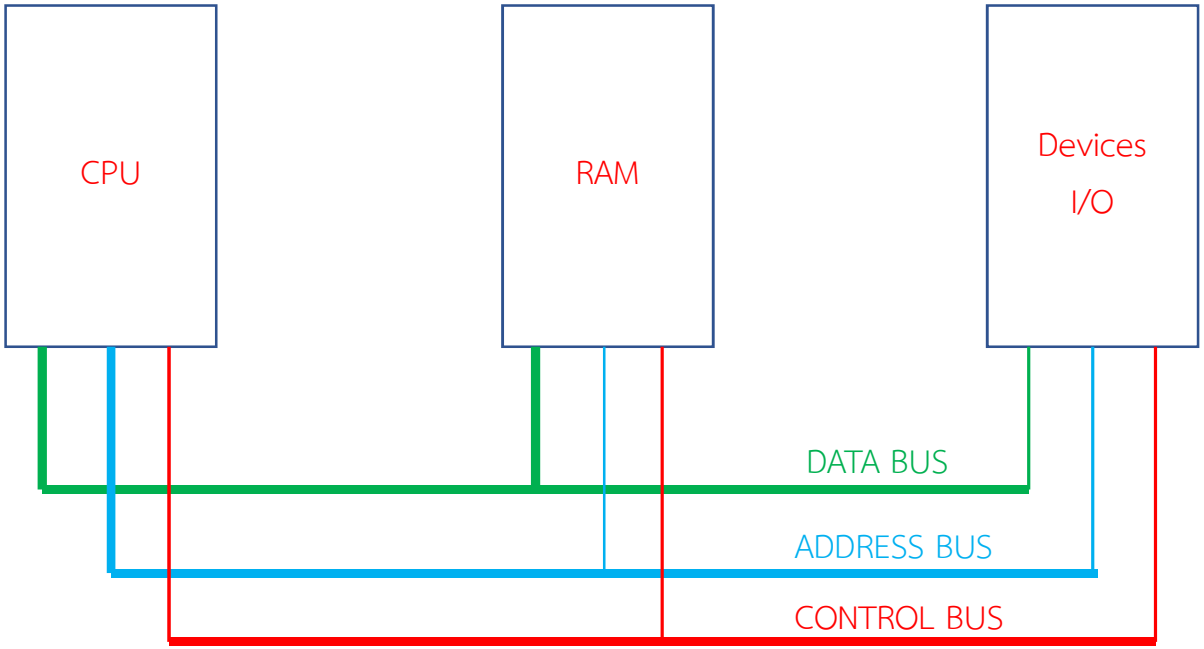
The Hardware



Intel 8086 ไม่ได้ออกแบบมาให้ทำงานได้ด้วยชิพตัวเดียว
ตัวอย่างเช่น Address pin และ Data pin ใช้งานร่วมกัน
ข้อมูลจะจ่ายออกมาสลับกัน (Multiplexing) ไปมา
จำเป็นต้องมีชิพอีกตัวเข้ามาช่วยในการแยกสัญญาณทั้ง 2 ออกจากกัน

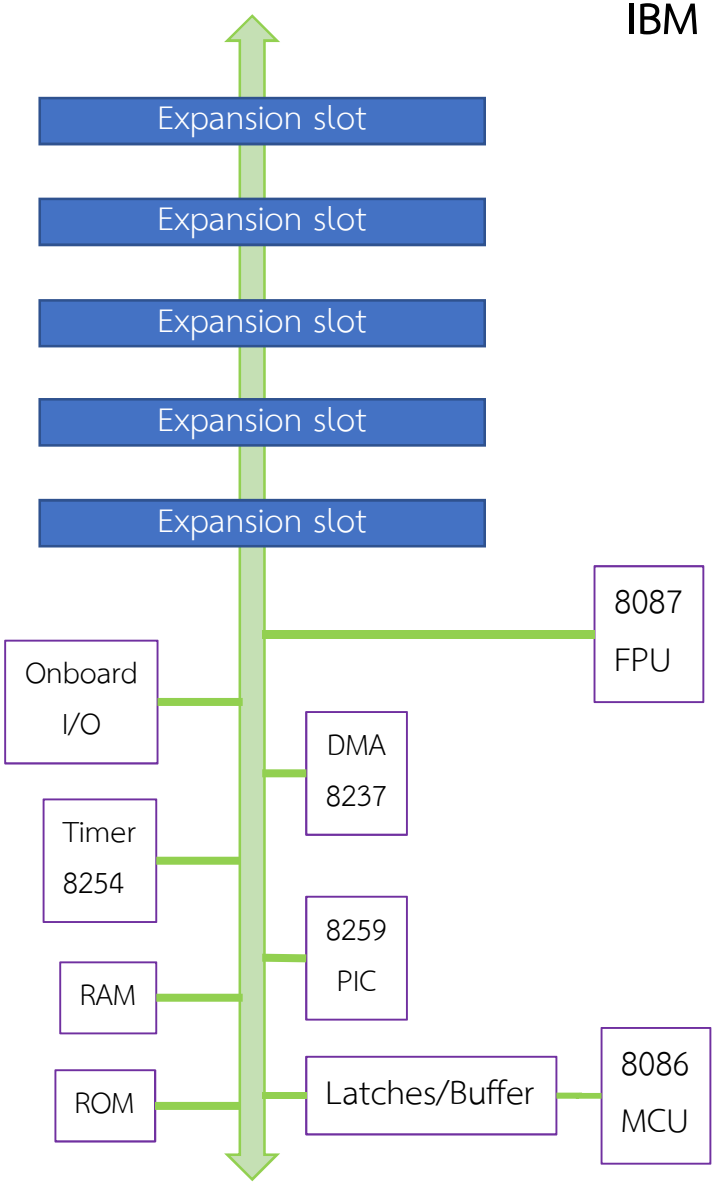
การ Interrupt จากภายนอกก็เช่นกัน

ระบบคอมพิวเตอร์ (Computer System)

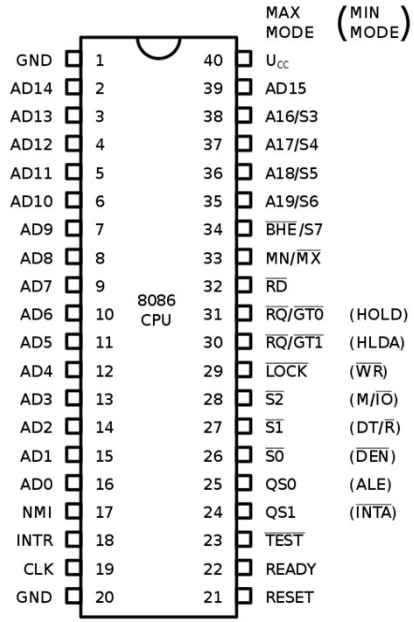


ระบบคอมพิวเตอร์ในบทที่ 1 เป็นแบบย่อ
เป็นภาพรวม

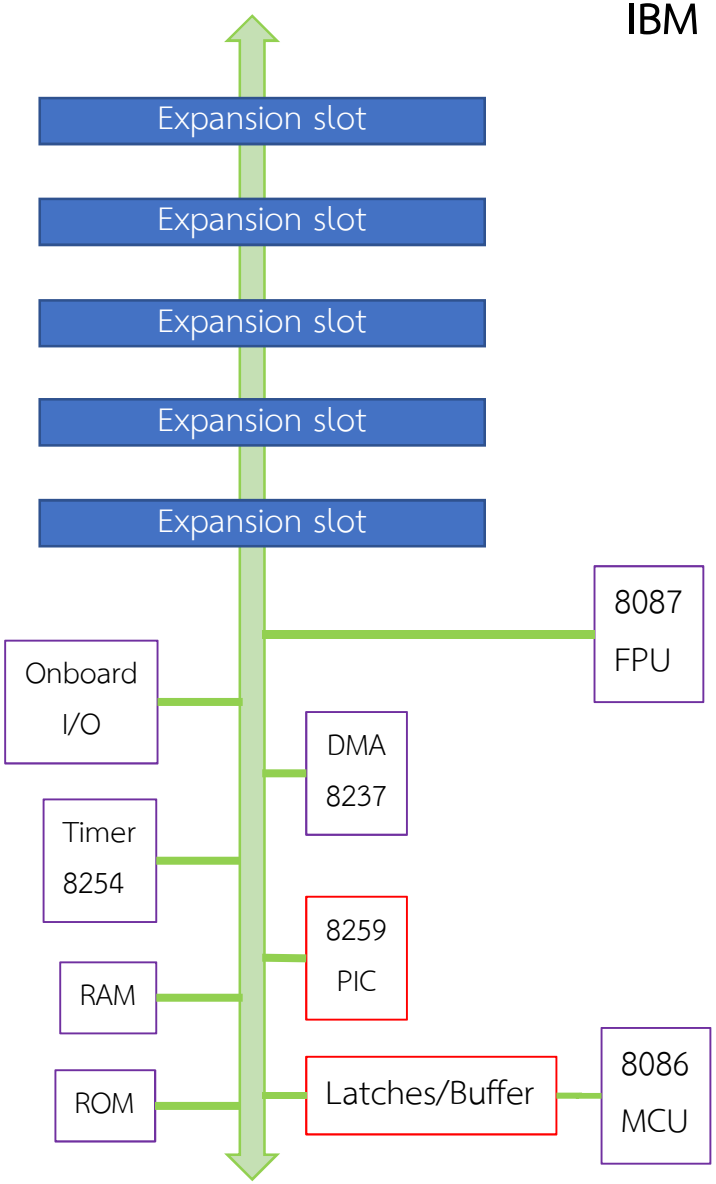
IBM PC Hardware

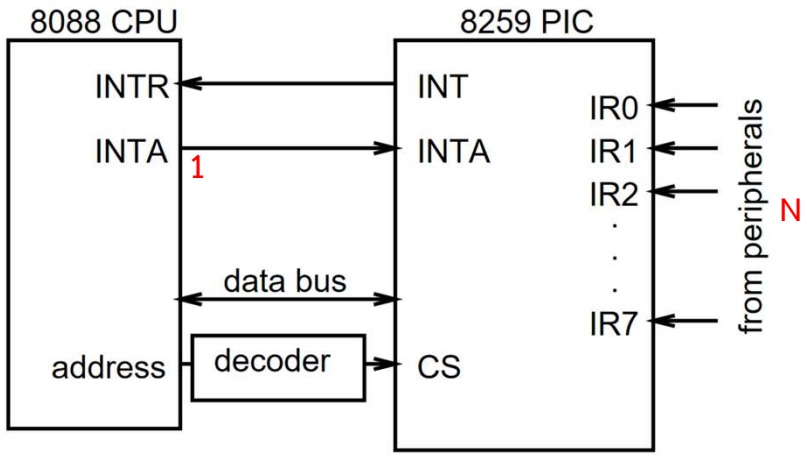


IBM 8088 PC/XT



IBM PC Hardware

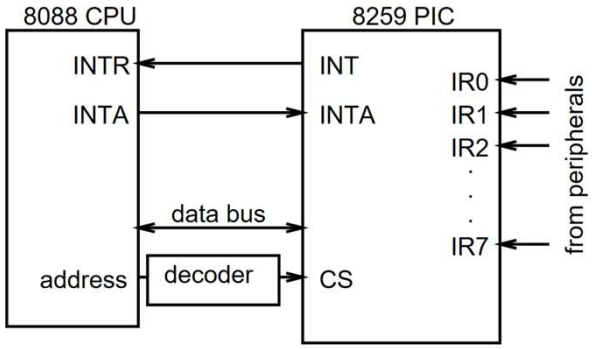




สัญญาณ Interrupt INTR ถูกขยายออกเป็นหลายเส้นโดยใช้ชิพ 8259 PIC (PROGRAMMABLE INTERRUPT CONTROLLER)

- ใช้ขยายจำนวน INTR โดยสามารถ Enable / Disable แต่ละตัวได้
- จัดลำดับความสำคัญ จัดการคิวของ Interrupt แต่ละตัวได้
- กำหนดชนิด Interrupt ให้กับ INTR ที่ถูกขยายไปแต่ละตัวด้วย

IBM PC Hardware

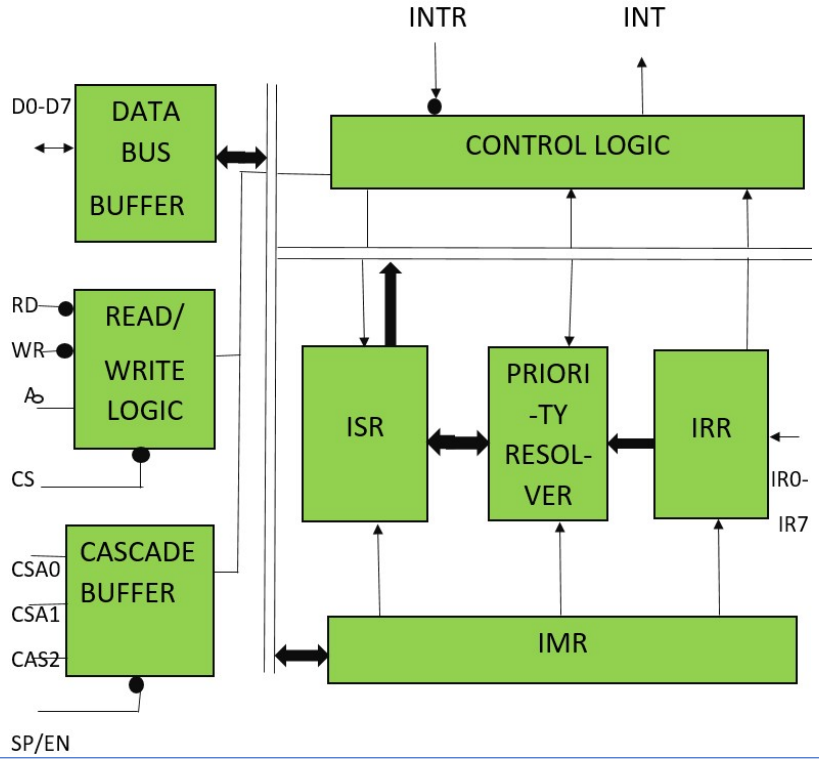


8259 สามารถขยาย Interrupt จาก 1 ไปเป็น 8 ได้

สามารถต่อพ่วงได้ถึง 8 ตัว

IBM PC / XT จะใช้ชิพนี้ 1 ตัว

IBM PC / AT จะมีชิพตัวนี้ต่อพ่วงกัน 2 ตัว



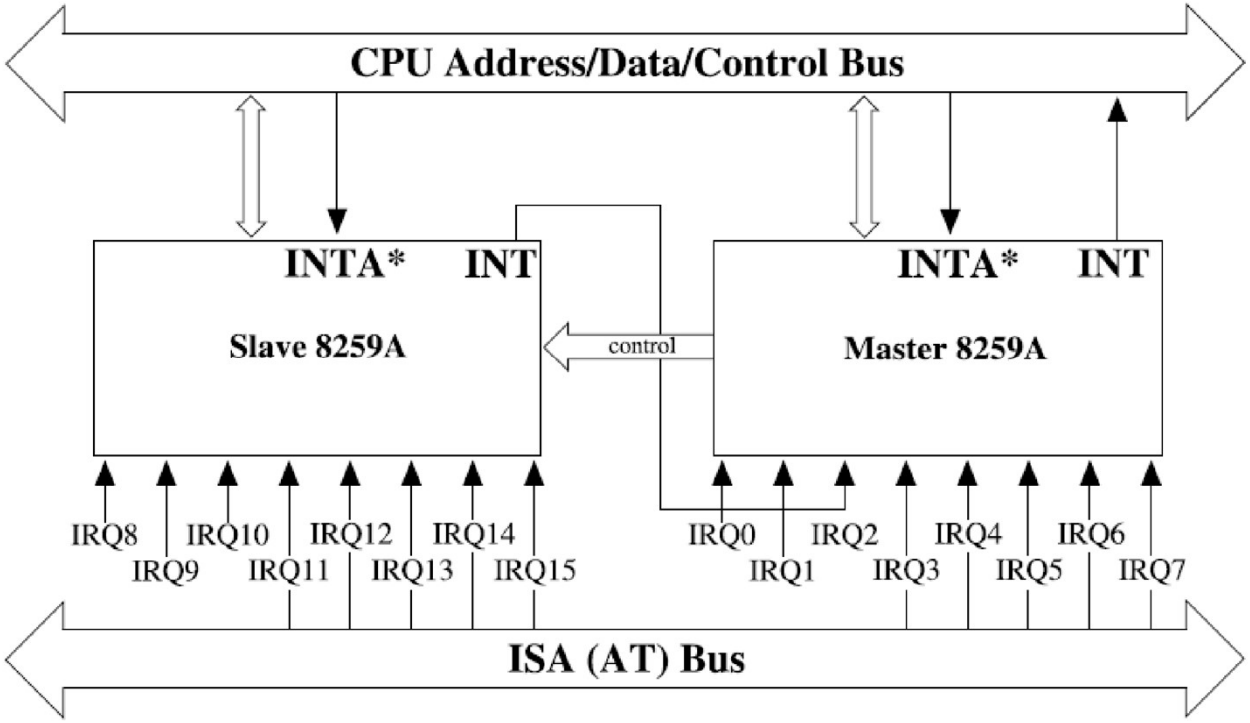
IRQ0 (ความสำคัญสูงสุด)

IRQ1

.....

IRQ7 (ความสำคัญต่ำสุด)

IBM PC Hardware

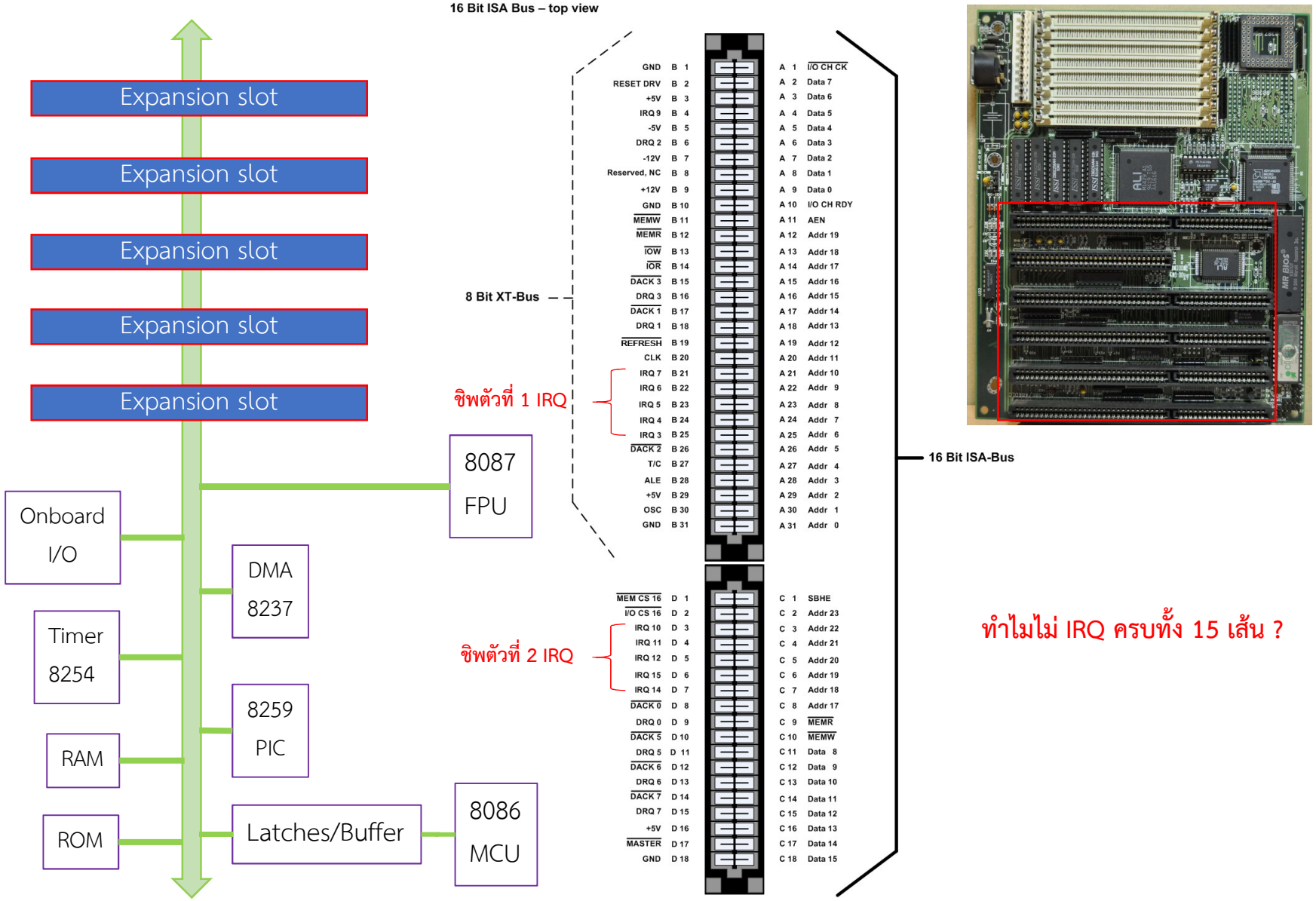


ชิพตัวแรก(ขวามือ) ขยายเป็น IRQ 0 – IRQ 7 IRQ 2 ใช้พ่วงกับชิพตัวที่ 2 จึงเสียไป 1 ตัว

ชิพตัวที่ 2 ขยายเป็น IRQ 8 – IRQ 15 ใช้งานได้ทุกตัว

รวมมี IRQ 0 – IRQ 15 แต่ IRQ 2 ใช้งานไม่ได้ รวมเป็น 15 ตัว

0,1,3,4,5,6,7,8,9,10,11,12,13,14,15


16 Bit ISA Bus – top view
Expansion slot
Expansion slot
Expansion slot
Expansion slot
Expansion slot
Onboard I/O
Timer 8254
RAM
ROM
DMA 8237
8259 PIC
Latches/Buffer
8087 FPU
8086 MCU
8 Bit XT-Bus
ชิพตัวที่ 1 IRQ
ชิพตัวที่ 2 IRQ
16 Bit ISA-Bus
GND B 1
RESET DRV B 2
+5V B 3
IRQ 9 B 4
-5V B 5
DRQ 2 B 6
-12V B 7
Reserved, NC B 8
+12V B 9
GND B 10
MEMW B 11
MEMR B 12
IOW B 13
IOR B 14
DACK 3 B 15
DRQ 3 B 16
DACK 1 B 17
DRQ 1 B 18
REFRESH B 19
CLK B 20
IRQ 7 B 21
IRQ 6 B 22
IRQ 5 B 23
IRQ 4 B 24
IRQ 3 B 25
DACK 2 B 26
T/C B 27
ALE B 28
+5V B 29
OSC B 30
GND B 31
A 1 I/O CH CK
A 2 Data 7
A 3 Data 6
A 4 Data 5
A 5 Data 4
A 6 Data 3
A 7 Data 2
A 8 Data 1
A 9 Data 0
A 10 I/O CH RDY
A 11 AEN
A 12 Addr 19
A 13 Addr 18
A 14 Addr 17
A 15 Addr 16
A 16 Addr 15
A 17 Addr 14
A 18 Addr 13
A 19 Addr 12
A 20 Addr 11
A 21 Addr 10
A 22 Addr 9
A 23 Addr 8
A 24 Addr 7
A 25 Addr 6
A 26 Addr 5
A 27 Addr 4
A 28 Addr 3
A 29 Addr 2
A 30 Addr 1
A 31 Addr 0
MEM CS 16 D 1
I/O CS 16 D 2
IRQ 10 D 3
IRQ 11 D 4
IRQ 12 D 5
IRQ 15 D 6
IRQ 14 D 7
DACK 0 D 8
DRQ 0 D 9
DACK 5 D 10
DRQ 5 D 11
DACK 6 D 12
DRQ 6 D 13
DACK 7 D 14
DRQ 7 D 15
+5V D 16
MASTER D 17
GND D 18
C 1 SBHE
C 2 Addr 23
C 3 Addr 22
C 4 Addr 21
C 5 Addr 20
C 6 Addr 19
C 7 Addr 18
C 8 Addr 17
C 9 MEMR
C 10 MEMW
C 11 Data 8
C 12 Data 9
C 13 Data 10
C 14 Data 11
C 15 Data 12
C 16 Data 13
C 17 Data 14
C 18 Data 15
ทำไมไม่ IRQ ครบทั้ง 15 เส้น ?

IBM PC Interrupts

IRQ	ใช้งาน(มาตรฐาน)	ใช้งาน(ทางเลือกอื่น)
0	System timer	
1	Keyboard Interface	
2	Cascade for second IRQ controller	
3	Serial port, COM2	
4	Serial Port COM1	
5	Parallel Port LPT2 (Parallel port ในโหมด Normal ไม่ใช้ IRQ)	Sound card
6	Floppy Disk Drive Interface	
7	Parallel Port LPT1 (Parallel port ในโหมด Normal ไม่ใช้ IRQ)	
8	Real Time Clock	
9	ว่าง	Video Display Interface cards
10	ว่าง	
11	ว่าง	
12	PS/2 - type mouse port	
13	Coprocessor	
14	Primary IDE/ATA adaptor	
15	Secondary IDE/ATA adaptor	

IBM PC Interrupts

IRQ ไม่ใช่ Interrupt Type!

IRQ 0 – 15

เมื่อถูก Hardware ภายนอก Interrupt

หากเราเราจะเขียน ISR ไปบริการ Hardware พวกนี้

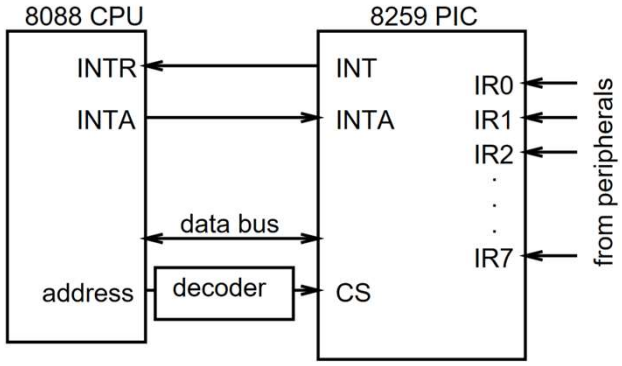
จะต้องเลือก INT ชนิดไหน เบอร์อะไร

IRQ เชื่อมโยงกับ Interrupt Vector Table อย่างไร ?

INT 00H – INT 255H

IRQ	ใช้งาน(มาตรฐาน)	ใช้งาน(ทางเลือกอื่น)
0	System timer	
1	Keyboard Interface	
2	Cascade for second IRQ controller	
3	Serial port, COM2	
4	Serial Port COM1	
5	Parallel Port LPT2 (Parallel port ในโหมด Normal ไม่ใช้ IRQ)	Sound card
6	Floppy Disk Drive Interface	
7	Parallel Port LPT1 (Parallel port ในโหมด Normal ไม่ใช้ IRQ)	
8	Real Time Clock	
9	ว่าง	Video Display Interface cards
10	ว่าง	
11	ว่าง	
12	PS/2 - type mouse port	
13	Coprocessor	
14	Primary IDE/ATA adaptor	
15	Secondary IDE/ATA adaptor	

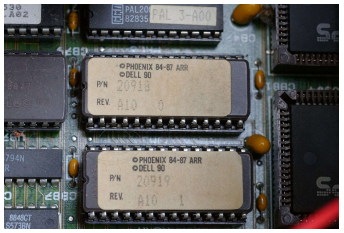
IBM PC Interrupts



ชิพ 8259 สามารถโปรแกรมให้ IRQ แต่ละตัว เกิด INT ชนิดไหนก็ได้
แต่ต้องไม่ชนกับ INT ตัวที่ถูกใช้ไปแล้ว
BIOS จะโปรแกรมชิพนี้ตอนบู๊ตเครื่อง

IBM PC Interrupts

Int. Num.	Description
0	CPU divide by zero
1	Debug single step
2	Non Maskable Interrupt (NMI input on processor)
3	Debug breakpoints
4	Arithmetic overflow
5	BIOS provided Print Screen routine
6	Reserved
7	Reserved
8	IRQ0, Time of day hardware services
9	IRQ1, Keyboard Interface
A	IRQ2, ISA Bus cascade services for second 8259
B	IRQ3, Com 2 hardware
C	IRQ4, Com1 hardware
D	IRQ5, LPT2, Parallel port hardware (Hard Disk on XT)
E	IRQ6, Floppy Disk adaptor
F	IRQ7, LPT1, Parallel port hardware
10	Video services, see note 1
11	Equipment check
12	Memory size determination
13	Floppy I/O routines
14	Serial port I/O routines
15	PC used for Cassette tape services
16	Keyboard I/O routines
17	Printer I/O routines
18	Points to basic interpreter in a "real" IBM PC
19	Bootstrap loader
1A	Time of day services
1B	Services Ctrl-Break service
1C	Timer tick (provides 18.2 ticks per second)
1D	Video parameters
1E	Disk parameters
1F	Video graphics
20	Program termination (obsolete)
21	All DOS services available through this Interrupt
22	Terminate address
23	Ctrl-Break exit address
24	Critical error handler
25	Read logical sectors
26	Write logical sectors
27	Terminate and stay resident routines (obsolete)
28 to 3F	Reserved for DOS
40 to 4F	Reserved for BIOS
50	Reserved for BIOS
51	Mouse functions
52 to 59	Reserved for BIOS
5A	Reserved for BIOS
5B	Reserved for BIOS
5D	Reserved for BIOS
5E	Reserved for BIOS
5F	Reserved for BIOS
60 to 66	Reserved for User programs
67	Used for EMS functions
68 to 6F	Unused
70	IRQ8, ISA bus Real time clock
71	IRQ9, takes the place of IRQ2
72	IRQ10 (available hardware interrupt)
73	IRQ11 (available hardware interrupt)
74	IRQ12 (available hardware interrupt)
75	IRQ13, maths co-processor
76	IRQ14, ISA bus hard disk controller
77	IRQ15, (available hardware interrupt)
78 to 7F	Unused
80 to 85	Reserved for basic
86 to F0	Used by basic
F1 to FF	Unused



http://philipstorr.id.au/pcbook/book2/intlist.htm

IBM PC Interrupts

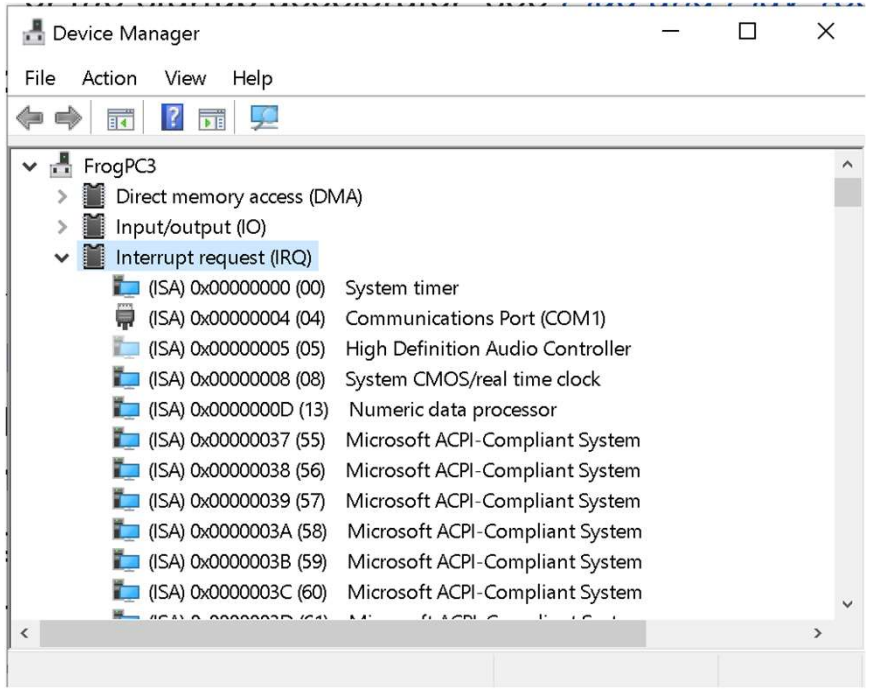
การ์ดส่วนขยายที่ไม่ได้เป็นมาตรฐานของ IBM PC ผู้ใช้สามารถเลือก IRQ ได้ได้เอง

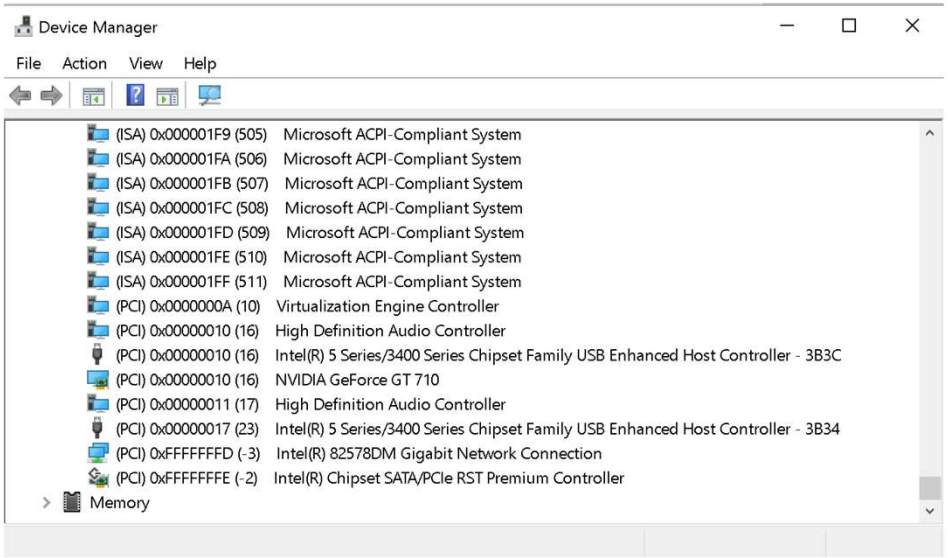
Jumper ตัวนี้จะต่อตรงกับ IRQ ของ System Bus

ผู้ใช้ต้องแน่ใจว่า จะไม่ชนกับอุปกรณ์ตัวอื่นในระบบ

IBM PC Interrupts

ปัจจุบันนี้ อุปกรณ์ต่อพ่วงมีความซับซ้อนและมีจำนวนมากขึ้น
การเลือก IRQ ด้วยมือ จึงมีโอกาสชน กันได้ง่าย
จึงมีการแทนที่ด้วยเทคโนโลยี Plug and Play โดยให้ BIOS เป็นผู้เลือก IRQ ให้ในขั้นตอนของการบู๊ตเครื่อง
AT BUS ถูกแทนที่ด้วย PCI BUS โดย CPU ไม่ได้ควบคุมสัญญาณต่าง ๆใน System Bus โดยตรงแล้ว
แต่อย่างไรก็ตามวิธีการ Interrupt ยังเหมือนเดิม





IBM PC Interrupts

บทนี้เราได้ศึกษาวิธีการเขียนโปรแกรมเพื่อ ตรวจจับการเรียกใช้บริการจาก Hardware
โดยอาศัยการนำเอา Address ของโปรแกรมย่อยไปวางในหน่วยความจำ Interrupt Vector

ขั้นตอนนี้เป็นเพียงครึ่งทางของการทำงานกับ hardware หรือการสร้าง Driver
เมื่อ Hardware ต้องการส่งข้อมูลเข้าสู่ Microprocessor จะทำอย่างไร ?
หรือเมื่อ Microprocessor ต้องการส่งข้อมูลไปยัง Hardware จะทำอย่างไร ?

Hardware มาตรฐานอย่างเช่น Mouse, Keyboard, Display, Disk Drive
เราเรียกใช้ API ของ BIOS หรือ Operating System ได้
แต่ถ้าเราสร้าง Hardware ขึ้นมาเอง เราจะสื่อสารกับมันได้อย่างไร

เราจะมาศึกษาเรื่องนี้ในบทถัดไป