INTERRUPT

November 15, 2009 in Komputer | Tags: contoh perintah interupt, interupt, interupt pada AMD, interupt pada intel, interupt pada komputer, tabel BIOS interupt, tabel DOS interupt

Interupsi atau bisa disebut **Interrupt** memiliki pengertian suatu permintaan khusus kepada mikroprosessor untuk melakukan sesuatu. Bila terjadi interupsi, mikroprosesor akan menghentikan dahulu apa yang sedang dikerjakannya dan mengerjakan permintaan khusus tersebut.

Jenis-jenis interrupt:
a.Software, interrupt jenis ini juga disebut System call. Misalnya, suatu program ingin mencetak hasil dengan printer
b.Hardware, terjadi karena adanya aksi pada perangkat keras, seperti penekanan tombol keyboard atau menggerakkan mouse. Interrupt ini terbagi lagi menjadi dua,yaitu: Maskable Interrupt(terjadi karena aksi luar) dan Non Maskable Interrupt(terjadi karena memori atau kesalahan parity pada program)

Penyebab terjadinya Interrupt:
a.Program, terjadi akibat eksekusi suatu instruksi
b.Timmer, disebabkan oleh timmer prosessor
c.I/O, disebabkan oleh I/O controller baik sebagai tanda bahwa operasi telah selesai maupun memberi tanda eror.
d.Kegagalan hardware, disebabkan oleh kesalahan hardware seperti power failure dan memori parity eror.

Ada dua aksi yang diberikan saat terjadi interrupt:
a.Syncronous I/O. I/O dijalankan, I/O selesai digunakan, kontrol menginformasikan kembali ke user proses. Untuk menunggu selesai digunakannya I/O, digunakan perintah wait.
b.Asyncronous I/O. Kembali ke user program tanpa harus menunggu I/O.

Vektor Interupsi
Vector interupsi merupakan 4 byte data yang disimpan pada 1024 byte pertama memori (000000h-0003FFFh) jika mikroprosesor dijalankan dalam real mode. Setiap vector interupsi ini berisi alamat procedure layanan interupsi, yaitu suatu procedure khusus yang dipanggil oleh vector interupsi. Dua byte pertama dari vector tersebut berisi alamat IP dan 2 byte terakhir berisi alamat CS dari procedure layanan interupsi tersebut.

ADa 256 vektor interupsi yang dimiliki mikroprosesor intel. Intel menyediakan 32 vektor interupsi untuk 8086-80486 dan kebutuhan-kebutuhan pengembangan di masa mendatang, sedangkan sisanya disediakan untuk dimanipulasi untuk digunakan untuk keperluan pengguna.

Berikut ini adalah table vector interupsi pada mikroprosesor keluarga intel:

No	Alamat	Mikroprosesor	Fungsi
0	0000h-0003h	8086-80486	Divide error
1	0004h-0007h	8086-80486	Single step
2	0008h-000Bh	8086-80486	NMI (Not Maskable Int.)
3	000Ch-000Fh	8086-80486	Breakpoint
4	0010h-0013h	8086-80486	Interrupt on overflow
5	0014h-0017h	80286-80486	Bound interrupt
6	0018h-001Bh	80286-80486	Invalid opcode
7	001Ch-001Fh	80286-80486	Coprosesor emulation intr.
8	0020h-0023h	80286-80486	Double fault
9	0024h-0027h	80386	Coprosesor segment overrun
10	0028h-002Bh	80386-80486	Invalid task state segmen
11	002Ch-002Fh	80386-80486	Segment not present
12	0030h-0033h	80386-80486	Stack Faulth
13	0034h-0037h	80386-80486	General protection fault
14	0038h-003Bh	80386-80486	Page Fault
15	003Ch-003Fh		Reserved*
16	0040h-0043h	80286-80486	Floating check error
17	0044h-0047h	80486SX	Alignment check interrupt
18-31	0048h-007Fh	80286-80486	Reserved*
32-255	0080h-03FFh	80286-80486	User Interrupt

Keterangan:
a.* = dicadangkan untuk pengembangan di masa mendatang
b.Vektor no. 1-7,9,16, dan 17 dapat digunakan untuk pemrograman real mode dan protected mode, sedangkan yang lain hanya untuk protected mode.
c.INT 100 berarti memanggil procedure layanan nomor 100 yang alamatnya pada 190h-193h.

Instruksi Interrupt pada PC (Personal Computer)
Instruksi interrupt pada PC(personal computer) berbeda dengan interupsi pada table interupai diatas, sebab PC pada awalnya dikembangkan berbasis (compatible dengan) system 8086-8088. Jadi interupsi yang sama di setiap PC adalah interupsi no 0-4. Berikut ini adalah table interupsi yang terdapat pada PC:

No	Fungsi	No	Fungsi
0	Divide error	18	ROM Basic
1	Single step (debug)	19	Reboot
2	NMI (Non Maskable Int.)	1A	Clock service
3	Breakpoint	1B	Control-break handler
4	Arithmatic overflow	1C	User timer service
5	Print Screen	1D	Video parameter
6	Illegal instruction error	1E	Disk drive parameter
7	Coprosesor not present int.	1F	Graphics character
8	Timer tick (hardware)	20	Terminate program
9	Keyboard (hardware)	21	DOS service
A	Hardware interrupt 2	22	Program termination handler
B-F	Hardware interrupt 3-7	23	Control-C handler
10	Video BIOS	24	Critical error handler
11	Equipment environment	25	Read disk
12	Conventional memory size	26	Write disk
13	Direct disk service	27	Terminate and stay resident
14	Serial COM port service	28	DOS idle
15	Miscellaneous service	2F	Multiplex handler
16	Keyboard service	70-77	Hardware interrupts 8-15
17	Parallel port LPT service		

Ada interupsi yang hanya trdiri dari 1 fungi layanan,misalnya INT20h (untuk menghentikan program), tetapi ada pula yang lebih, misalnya INT 21h, INT10h, dll. Untuk memanggil nomor layanan tertentu, nomor layanan tersebut harus dimasukkan dulu ke register AH sebelum INT bersangkutan dieksekusi.

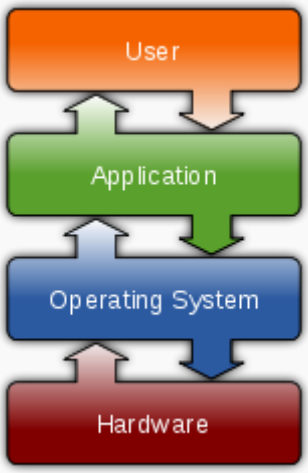
sumber : http://rani-amalia-elins1.blogspot.com/

Interrupt

From Wikipedia, the free encyclopedia

Jump to: navigation, search

Operating systems



Common features

- Process management
- **Interrupts**
- Memory management
- Virtual file system
- Device drivers
- Networking
- Security
- Graphical user interfaces

v • d • e

In computing, an **interrupt** is an asynchronous signal indicating the need for attention or a synchronous event in software indicating the need for a change in execution.

A *hardware interrupt* causes the processor to save its state of execution and begin execution of an interrupt handler.

Software interrupts are usually implemented as instructions in the instruction set, which cause a context switch to an interrupt handler similar to a hardware interrupt.

Interrupts are a commonly used technique for computer multitasking, especially in real-time computing. Such a system is said to be interrupt-driven.[1]

An act of *interrupting* is referred to as an interrupt request (IRQ).

Contents

Overview

Hardware interrupts were introduced as a way to avoid wasting the processor's valuable time in polling loops, waiting for external events. They may be implemented in hardware as a distinct system with control lines, or they may be integrated into the memory subsystem.

If implemented in hardware, an interrupt controller circuit such as the IBM PC's Programmable Interrupt Controller (PIC) may be connected between the interrupting device and the processor's interrupt pin to multiplex several sources of interrupt onto the one or two CPU lines typically available. If implemented as part of the memory controller, interrupts are mapped into the system's memory address space.

Interrupts can be categorized into: **maskable interrupt**, **non-maskable interrupt (NMI)**, **interprocessor interrupt (IPI)**, **software interrupt**, and **spurious interrupt**.

- **Maskable interrupt** (IRQ) is a hardware interrupt that may be ignored by setting a bit in an interrupt mask register's (IMR) bit-mask.
- **Non-maskable interrupt** (NMI) is a hardware interrupt that lacks an associated bit-mask, so that it can never be ignored. NMIs are often used for timers, especially watchdog timers.
- **Interprocessor interrupt** is a special case of interrupt that is generated by one processor to interrupt another processor in a multiprocessor system.
- **Software interrupt** is an interrupt generated within a processor by executing an instruction. Software interrupts are often used to implement system calls because they implement a subroutine call with a CPU ring level change.
- **Spurious interrupt** is a hardware interrupt that is unwanted. They are typically generated by system conditions such as electrical interference on an interrupt line or through incorrectly designed hardware.

Processors typically have an internal *interrupt mask* which allows software to ignore all external hardware interrupts while it is set. This mask may offer faster access than accessing an interrupt mask register (IMR) in a PIC, or disabling interrupts in the device itself. In some cases, such as the x86 architecture, disabling and enabling interrupts on the processor itself act as a memory barrier, however it may actually be slower.

An interrupt that leaves the machine in a well-defined state is called a **precise interrupt**. Such an interrupt has four properties:

- The Program Counter (PC) is saved in a known place.
- All instructions before the one pointed to by the PC have fully executed.
- No instruction beyond the one pointed to by the PC has been executed (that is no prohibition on instruction beyond that in PC, it is just that any changes they make to registers or memory must be undone before the interrupt happens).
- The execution state of the instruction pointed to by the PC is known.

An interrupt that does not meet these requirements is called an **imprecise interrupt**.

The phenomenon where the overall system performance is severely hindered by excessive amounts of processing time spent handling interrupts is called an interrupt storm.

Types of Interrupts

Level-triggered

A **level-triggered interrupt** is a class of interrupts where the presence of an unserviced interrupt is indicated by a high level (1), or low level (0), of the interrupt request line. A device wishing to signal an interrupt drives line to its active level, and then holds it at that level until serviced. It ceases asserting the line when the CPU commands it to or otherwise handles the condition that caused it to signal the interrupt.

Typically, the processor samples the interrupt input at predefined times during each bus cycle such as state T2 for the Z80 microprocessor. If the interrupt isn't active when the processor samples it, the CPU doesn't see it. One possible use for this type of interrupt is to minimize spurious signals from a noisy interrupt line: a spurious pulse will often be so short that it is not noticed.

Multiple devices may share a level-triggered interrupt line if they are designed to. The interrupt line must have a pull-down or pull-up resistor so that when not actively driven it settles to its inactive state. Devices actively assert the line to indicate an outstanding interrupt, but let the line float (do not actively drive it) when not signalling an interrupt. The line is then in its asserted state when any (one or more than one) of the sharing devices is signalling an outstanding interrupt.

This class of interrupts is favored by some because of a convenient behavior when the line is shared. Upon detecting assertion of the interrupt line, the CPU must search through the devices

sharing it until one requiring service is detected. After servicing this device, the CPU may recheck the interrupt line status to determine whether any other devices also need service. If the line is now de-asserted, the CPU avoids checking the remaining devices on the line. Since some devices interrupt more frequently than others, and other device interrupts are particularly expensive, a careful ordering of device checks is employed to increase efficiency.

There are also serious problems with sharing level-triggered interrupts. As long as any device on the line has an outstanding request for service the line remains asserted, so it is not possible to detect a change in the status of any other device. Deferring servicing a low-priority device is not an option, because this would prevent detection of service requests from higher-priority devices. If there is a device on the line that the CPU does not know how to service, then any interrupt from that device permanently blocks all interrupts from the other devices.

The original PCI standard mandated shareable level-triggered interrupts. The rationale for this was the efficiency gain discussed above. (Newer versions of PCI allow, and PCI Express requires the use of message-signalled interrupts.)

Edge-triggered

An **edge-triggered interrupt** is a class of interrupts that are signalled by a level transition on the interrupt line, either a falling edge (1 to 0) or a rising edge (0 to 1). A device wishing to signal an interrupt drives a pulse onto the line and then releases the line to its quiescent state. If the pulse is too short to be detected by polled I/O then special hardware may be required to detect the edge.

Multiple devices may share an edge-triggered interrupt line if they are designed to. The interrupt line must have a pull-down or pull-up resistor so that when not actively driven it settles to one particular state. Devices signal an interrupt by briefly driving the line to its non-default state, and let the line float (do not actively drive it) when not signalling an interrupt. This type of connection is also referred to as open collector. The line then carries all the pulses generated by all the devices. (This is analogous to the pull cord on some buses and trolleys that any passenger can pull to signal the driver that they are requesting a stop.) However, interrupt pulses from different devices may merge if they occur close in time. To avoid losing interrupts the CPU must trigger on the trailing edge of the pulse (e.g. the rising edge if the line is pulled up and driven low). After detecting an interrupt the CPU must check all the devices for service requirements.

Edge-triggered interrupts do not suffer the problems that level-triggered interrupts have with sharing. Service of a low-priority device can be postponed arbitrarily, and interrupts will continue to be received from the high-priority devices that are being serviced. If there is a device that the CPU does not know how to service, it may cause a spurious interrupt, or even periodic spurious interrupts, but it does not interfere with the interrupt signalling of the other devices. However, it is fairly easy for an edge triggered interrupt to be missed - for example if interrupts have to be masked for a period - and unless there is some type of hardware latch that records the event it is impossible to recover. Such problems caused many "lockups" in early computer hardware because the processor did not know it was expected to do something. More modern hardware often has one or more interrupt status registers that latch the interrupt requests; well

written edge-driven interrupt software often checks such registers to ensure events are not missed.

The elderly Industry Standard Architecture (ISA) bus uses edge-triggered interrupts, but does not mandate that devices be able to share them. The parallel port also uses edge-triggered interrupts. Many older devices assume that they have exclusive use of their interrupt line, making it electrically unsafe to share them. However, ISA motherboards include pull-up resistors on the IRQ lines, so well-behaved devices share ISA interrupts just fine.

Hybrid

Some systems use a hybrid of level-triggered and edge-triggered signalling. The hardware not only looks for an edge, but it also verifies that the interrupt signal stays active for a certain period of time.

A common use of a hybrid interrupt is for the NMI (non-maskable interrupt) input. Because NMIs generally signal major – or even catastrophic – system events, a good implementation of this signal tries to ensure that the interrupt is valid by verifying that it remains active for a period of time. This 2-step approach helps to eliminate false interrupts from affecting the system.

Message-signalled

Main article: Message Signaled Interrupts

A **message-signalled interrupt** does not use a physical interrupt line. Instead, a device signals its request for service by sending a short message over some communications medium, typically a computer bus. The message might be of a type reserved for interrupts, or it might be of some pre-existing type such as a memory write.

Message-signalled interrupts behave very much like edge-triggered interrupts, in that the interrupt is a momentary signal rather than a continuous condition. Interrupt-handling software treats the two in much the same manner. Typically, multiple pending message-signalled interrupts with the same message (the same virtual interrupt line) are allowed to merge, just as closely-spaced edge-triggered interrupts can merge.

Message-signalled interrupt vectors can be shared, to the extent that the underlying communication medium can be shared. No additional effort is required.

Because the identity of the interrupt is indicated by a pattern of data bits, not requiring a separate physical conductor, many more distinct interrupts can be efficiently handled. This reduces the need for sharing. Interrupt messages can also be passed over a serial bus, not requiring any additional lines.

PCI Express, a serial computer bus, uses message-signalled interrupts exclusively.

Doorbell

In a push button analogy applied to computer systems, the term **doorbell** or **doorbell interrupt** is often used to describe a mechanism whereby a software system can signal or notify a hardware device that there is some work to be done. Typically, the software system will place data in some well known and mutually agreed upon memory location(s), and "ring the doorbell" by writing to a different memory location. This different memory location is often called the doorbell region, and there may even be multiple doorbells serving different purposes in this region. It's this act of writing to the doorbell region of memory that "rings the bell" and notifies the hardware device that the data is ready and waiting. The hardware device would now know that the data is valid and can be acted upon. It would typically write the data to a hard disk drive, or send it over a network, or encrypt it, etc.

The term **doorbell interrupt** is usually a misnomer. It's similar to an interrupt because it causes some work to be done by the device, however the doorbell region is sometimes implemented as a polled region, sometimes the doorbell region writes through to physical device registers, and sometimes the doorbell region is hardwired directly to physical device registers. When either writing through or directly to physical device registers, this may, but not necessarily, cause a real interrupt to occur at the device's central processor unit (CPU), if it has one.

Doorbell interrupts can be compared to Message Signaled Interrupts, as they have some similarities.

Difficulty with sharing interrupt lines

Multiple devices sharing an interrupt line (of any triggering style) all act as spurious interrupt sources with respect to each other. With many devices on one line the workload in servicing interrupts grows in proportion to the square of the number of devices. It is therefore preferred to spread devices evenly across the available interrupt lines. Shortage of interrupt lines is a problem in older system designs where the interrupt lines are distinct physical conductors. Message-signalled interrupts, where the interrupt line is virtual, are favoured in new system architectures (such as PCI Express) and relieve this problem to a considerable extent.

Some devices with a badly-designed programming interface provide no way to determine whether they have requested service. They may lock up or otherwise misbehave if serviced when they do not want it. Such devices cannot tolerate spurious interrupts, and so also cannot tolerate sharing an interrupt line. ISA cards, due to often cheap design and construction, are notorious for this problem. Such devices are becoming much rarer, as hardware logic becomes cheaper and new system architectures mandate shareable interrupts.

Performance issues

Interrupts provide low overhead and good latency at low offered load, but degrade significantly at high interrupt rate unless care is taken to prevent several pathologies. These are various forms of livelocks, when the system spends all of its time processing interrupts, to the exclusion of other required tasks. Under extreme conditions, a large number of interrupts (like very high

network traffic) may completely stall the system. To avoid such problems, an operating system must schedule network interrupt handling as carefully as it schedules process execution.[2]

Typical uses

Typical uses of interrupts include the following: system timers, disks I/O, power-off signals, and traps. Other interrupts exist to transfer data bytes using UARTs or Ethernet; sense key-presses; control motors; or anything else the equipment must do.

A classic system timer interrupt interrupts periodically from a counter or the power-line. The interrupt handler counts the interrupts to keep time. The timer interrupt may also be used by the OS's task scheduler to reschedule the priorities of running processes. Counters are popular, but some older computers used the power line frequency instead, because power companies in most Western countries control the power-line frequency with a very accurate atomic clock.

A disk interrupt signals the completion of a data transfer from or to the disk peripheral. A process waiting to read or write a file starts up again.

A power-off interrupt predicts or requests a loss of power. It allows the computer equipment to perform an orderly shutdown.

Interrupts are also used in typeahead features for buffering events like keystrokes.

[edit] Interrupt routine example

Microchip PIC 18Fxxxx Microcontroller, compiled under MPLAB C18 compiler. The routine displays the number of interrupts occurred on port A, while it keeps toggling LEDs on port D:

```
 //************************************************************
 /*
 * Name : Interrupt.c
 *
 * Compiler : C18
 *
 * PIC : 18F Familly
 *
 * AUTHOR : BENKORICH Abdellah
 *
 * University : Boumerdes, ALGERIA
 */
 //************************************************************/

 #include <p18f4550.h> /* for the special function register declarations ,
you may include yours here  */
 #include <portb.h> /* for the RB0/INT0 interrupt */

 /* Set configuration bits for use with ICD2 / PICDEM2 PLUS Demo Board:
 * - set HS oscillator
 * - disable watchdog timer
 * - disable low voltage programming
```

```
 * - enable background debugging
 */

 #pragma config WDT = OFF
 #pragma config LVP = OFF
 #pragma config DEBUG = ON
 #define FOSC 40e6

 void delay(long time) ;
 int state=1; //The state of A0.

 /*
 * For high interrupts, control is transferred to address 0x8.
 */
 void update_cnt (void); /* prototype needed for 'goto' below */
 #pragma code HIGH_INTERRUPT_VECTOR = 0x8

void high_ISR (void)
 {
 _asm
  goto update_cnt
 _endasm
 }

 #pragma code /* allow the linker to locate the remaining code */
 #pragma interrupt update_cnt

 void update_cnt (void)
 {
  PORTA =state;
  state ++ ;
  INTCONbits.INT0IF = 0; /* clear flag to avoid another interrupt */
 }

 void EnableHighInterrupts (void)
 {
  RCONbits.IPEN = 1; /* enable interrupt priority levels */
  INTCONbits.GIEH = 1; /* enable all high priority interrupts */
 }

 // delay Function
 void delay (long time){
        long i;
        for (i=0; i<time; i++){
        }
 }

 // main function
 void main (void)
 {
  TRISA = 0x00; //Set port A as output
  LATA=0; //Initialize Port A.
  ADCON1 = 0b00001111; //All ADC disabled
  TRISD = 0x00; //Set port D as output
  LATD=0; //Initialize Port D.
  EnableHighInterrupts ( );
  OpenRB0INT (PORTB_CHANGE_INT_ON & /* enable the RB0/INT0 interrupt */
```

```
              PORTB_PULLUPS_ON &     /* configure the RB0 pin for input */
              FALLING_EDGE_INT);     /* trigger interrupt upon S3 button
depression */

  while (1) {
   PORTD = 0x00;
   delay(FOSC/5000);
   PORTD = 0xFF;
   delay(FOSC/5000);
  }
 }
```

Penjabaran

Arti istilah **interrupt** dianggap berkaitan erat dengan pengertian berikut

Kejadian-kejadian sinkron yang merupakan tanggapan pemroses terhadap kondisi-kondisi tertentu yang memerlukan perhatian. Sebuah setting hardware yang menjalankan perintah-perintah dalam sistem komputer.

Interrupt secara harfiah dalam bahasa Indonesianya diartikan sebagai selaan, menyela, atau menjegal, atau istilah kerennya disebut dengan interupsi.

Interrupt bisa diibaratkan dalam kehidupan sehari-hari sebagai suatu proses berjalan, namun belum selesai proses tersebut melakukan tugasnya, sudah dilaksanakan lagi proses lainnya.

Ibaratnya begini, ketika anda sedang melakukan suatu pekerjaan, katakanlah membaca sebuah buku, belum selesai buku tersebut anda tamatkan, lalu telepon anda berbunyi, sehingga anda melakukan percakapan terlebih dahulu melalui telepon tersebut. Setelah pembicaraan selesai, anda melanjutkan membaca buku tadi. menerima telepon di dalam kejadian tersebut disebut dengan menyela.

Begitu juga dengan proses yang terjadi pada komputer. Apabila sebuah komputer melakukan prosesnya tanda ada gangguan, tentu komputer tersebut dapat menyelesaikan pekerjaannya dengan serius khusus untuk satu pekerjaan yang sedang dikerjakannya. Dalam kondisi demikian, komputer anda melakukan tugasnya yang disebut dengan primitive batch processing. Pekerjaan seperti ini digunakan oleh komputer pada komputer zaman awal-awal ditemukannya. Dimana komputer tidak bisa mengerjakan beberapa program sekaligus dalam waktu bersamaan, sampai satu pekerjaan selesai dikerjakan, maka baru dia bisa berpindah ke pekerjaan lainnya.

Komputer terkini, memiliki kemampuan interrupt ini. Penulis melakukan pengetikkan naskah ini sambil mendengarkan musik yang terpasang pada notebook yang digunakan, tidak jarang komputer ini juga sambil terhubung dengan internet untuk membuka halaman web atau mengambil beberapa

data yang ada di internet.

Anda tentu juga tidak jarang mengalami hal dengan interrupt ini, katakanlah, ketika mengetikkan SMS ternyata ada telpon yang masuk, anda terima dulu telpon tersebut, lalu setelahnya anda lanjutkan pengetikan SMS tadi.

Untuk memungkinkan terjadinya interrupt ini pada sistem komputer, CPU memiliki suatu jalur khusus terhadap suatu chip pengatur interrupt eksternal (bagian dari chipset), yang berisi database sederhana yang dikenal dengan interrupt vectors.

Ketika sebuah interrupt terjadi pada chip, maka CPU menyimpan informasi terakhir yang dia kerjakan, berulah dia mengerjakan sesuai dengan informasi yang ada pada interrupt vector tesebut. Interrupt vector ini sebenarnya hanya sebuah nama pemanis yang berisi informasi tentang selaan yang terjadi, kalau dibelah lebih dalam lagi, isinya adalah berupa tabel yang berisi angka-angka). Pada interrupt vector inilah ditemukan kemana dan apa proses berikutnya yang harus dilaksanakan oleh komputer. Ketika pekerjaan interrupt tadi selesai dilaksanakan, maka komputer melakukan pelacakan kembali apa pekerjaan sebelumnya yang sedang dilaksanakannya.

Prioritas dalam interrupt

Dalam penerimaan suatu interrupt ini, komputer membagi interrupt tersebut dalam berbagai level, tergantung dari CPU yang digunakan. Misalnya pada komputer yang digunakan untuk pekerjaan yang cukup membutuhkan konsentrasi dari CPU, maka CPU tersebut memungkinkan untuk mengabaikan interrupt yang prioritasnya rendah, katakanlah pengetikkan yang dilakukan oleh seorang user melalui keyboard, namun komputer tersebut akan memberikan respon yang sangat cepat apabila terjadi gangguan pada memori yang digunakannya.