**Interupsi**

Interupsi terjadi bila suatu perangkat M/K ingin memberitahu prosesor bahwa ia siap menerima perintah, *output* sudah dihasilkan, atau terjadi *error*.

**Penanganan Interupsi**

Ada beberapa tahapan dalam penanganan interupsi:

1. *Controller* mengirimkan sinyal interupsi melalui *interrupt-request-line*
2. Sinyal dideteksi oleh prosesor
3. Prosesor akan terlebih dahulu menyimpan informasi tentang keadaan *state*-nya (informasi tentang proses yang sedang dikerjakan)
4. Prosesor mengidentifikasi penyebab interupsi dan mengakses tabel vektor interupsi untuk menentukan *interrupt handler*
5. Transfer kontrol ke *interrupt handler*
6. Setelah interupsi berhasil diatasi, prosesor akan kembali ke keadaan seperti sebelum terjadinya interupsi dan melanjutkan pekerjaan yang tadi sempat tertunda.

**Gambar 10.2. Siklus penanganan interupsi**

|  |
| --- |
| Siklus penanganan interupsi |

**Interrupt Request Line**

Pada kebanyakan CPU, ada dua *interrupt request line*. Pertama, interupsi *nonmaskable*, interupsi ini biasanya berasal dari perangkat keras dan harus segera dilaksanakan, seperti terjadinya *error* pada memori.

Kedua, interupsi *maskable*, jenis interupsi ini bisa dilayani oleh prosesor atau bisa tidak dilayani. Kalaupun dilayani, harus dilihat keadaan prosesor saat itu. Ada kemungkinan prosesor langsung menangani bila saat itu, prosesor *preemptive*, bila *nonpreemptive*, maka harus menunggu proses yang sedang dikerjakan selesai.

**Vektor Interupsi dan *Vector Chaining***

Bila ada sebuah sinyal interupsi pada *interrupt request line*, bagaimana sebuah *interrupt handler* mengetahui sumber dari interupsi itu? Apakah harus menelusuri semua sumber interupsi satu-persatu? Tidak perlu, karena setiap *interrupt handler* mempunyai alamat memori masing-masing. Alamat ini adalah *offset* pada sebuah tabel yang disebut dengan vektor interupsi.

**Tabel 10.1. Tabel Vector-Even pada Intel Pentium**

| **Vector number** | **Description** |
| --- | --- |
| 0 | Divide error |
| 1 | Debug Exception |
| 2 | Null Interrupt |
| 3 | Breakpoint |
| 4 | INTO-detected overflow |
| 5 | Bound range exception |
| 6 | Invalid opcode |
| 7 | Device not available |
| 8 | Double fault |
| 9 | Compressor segment overrun (reserved) |
| 10 | Invalid task state segment |
| 11 | Segment not present |
| 12 | Stack fault |
| 13 | General protection |
| 14 | Page fault |
| 15 | (Intel reserved, do not use) |
| 16 | Floating point error |
| 17 | Alignment check |
| 18 | Machine check |
| 19-31 | (Intel reserved, do not use) |
| 32-255 | Maskable interrupt |

Sesuai dengan perkembangan zaman, komputer mempunyai lebih banyak perangkat (dan lebih banyak *interrupt handlers*) daripada elemen alamat di vektor. Hal ini bisa diatasi dengan teknik *vector chaining*. Setiap elemen di vektor interupsi menunjuk ke kepala dari sebuah daftar *interrupt handlers*, sehingga bila ada interupsi, *handler* yang terdapat pada daftar yang ditunjuk akan dipanggil satu persatu sampai didapatkan *handler* yang bisa menangani interupsi yang bersangkutan.

**Prioritas Interupsi**

Mekanisme interupsi juga menerapkan sistem level prioritas interupsi. Sistem ini memungkinkan interupsi berprioritas tinggi menyela eksekusi interupsi berprioritas rendah. Sistem ini juga memungkinkan perangkat M/K yang membutuhkan pelayanan secepatnya didahulukan daripada perangkat lainnya yang prioritasnya lebih rendah. Pengaturan prioritas dan penanganan perangkat berdasarkan prioritasnya diatur oleh prosesor dan *controller* interupsi.

**Penyebab Interupsi**

Mekanisme interupsi tidak hanya digunakan untuk menangani operasi yang berhubungan dengan perangkat M/K. Sistem operasi menggunakan mekanisme interupsi untuk beberapa hal, di antaranya:

1. Menangani *exception*

*Exception* adalah suatu kondisi dimana terjadi sesuatu, atau dari sebuah operasi didapatkan hasil tertentu yang dianggap khusus sehingga harus mendapat perhatian lebih, contohnya, pembagian dengan nol, pengaksesan alamat memori yang *restricted* atau tidak valid, dll.

1. Mengatur *virtual memory paging*.
2. Menangani perangkat lunak interupsi.
3. Menangani alur kontrol kernel.

Jika interupsi yang terjadi merupakan permintaan untuk transfer data yang besar, maka penggunaan interupsi menjadi tidak efisien, untuk mengatasinya digunakanlah DMA.