

Input/Output



Tim Dosen COA

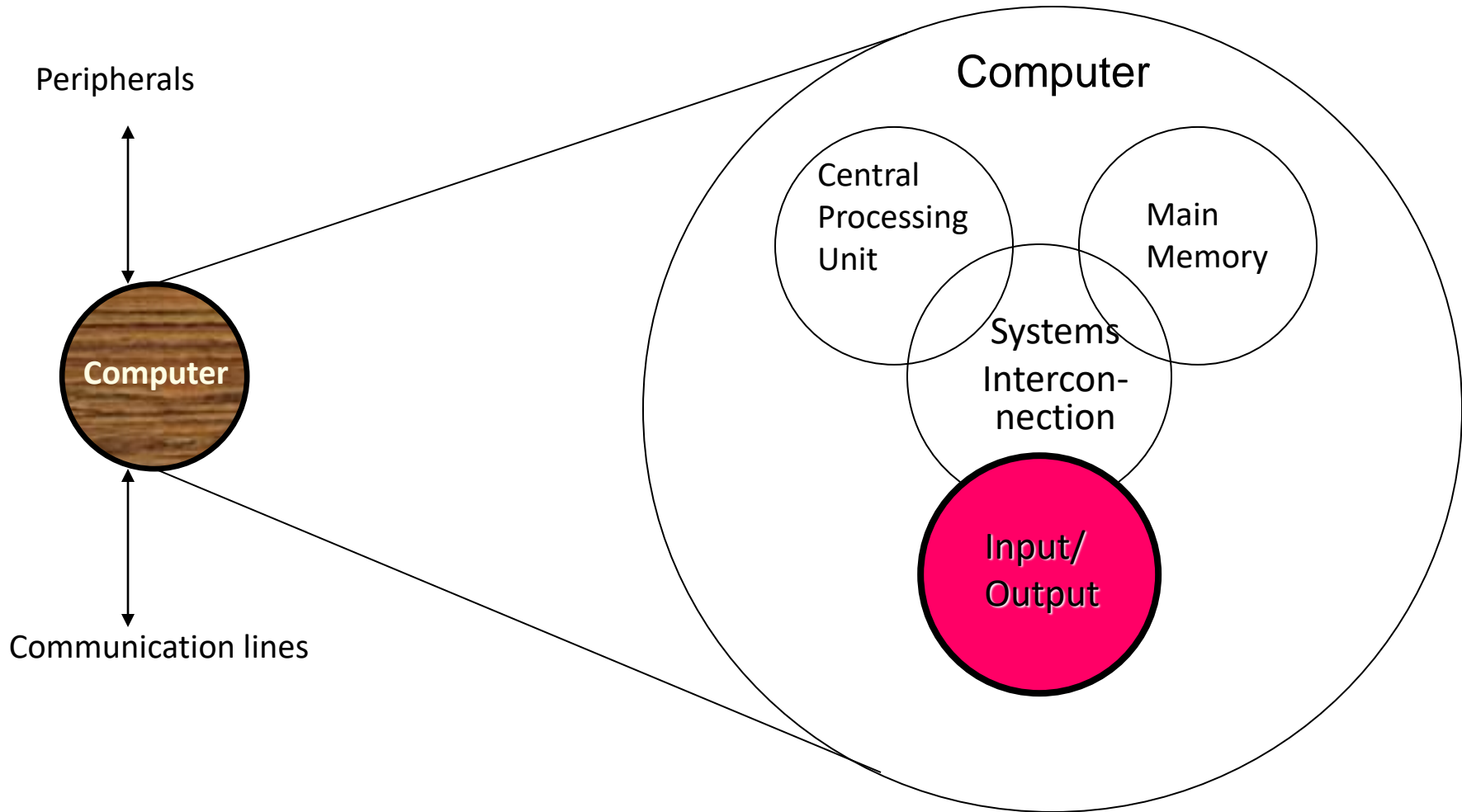
Fakultas Informatika
Universitas Telkom

Rencana Studi

Rencana Studi		Rincian Nilai Kuis																Rincian Nilai Tugas									Rincian Nilai Hasil Proyek												Bobot Tiap CLO (%)			
Pertemuan Ke-	Materi	CLO 1				CLO 2				CLO 3				CLO 4				CLO 1			CLO 2			CLO			CLO 3				CLO 4				1	2	3	4				
		Kuis (Kognitif) (20%)																Tugas Partisipatif (35%)									Hasil Proyek (45%)															
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	1	1	1	2	2	2	2	3	3	1	1	2	1	2	2	3	3					4	3	4	3
1	Sistem komputer	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	14	-	-	-	
2	Input/Output	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
3	Sistem Bus	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
4	Organisasi memori	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
5	Cara kerja memori utama (RAM)	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
6	Memori sekunder	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
7	Cara kerja cache memory (bag-1)	-	-	-	-	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
8	Cara kerja cache memory (bag-2)	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
9	Arsitektur SAP-1	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
10	Arsitektur SAP-2	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
11	Arsitektur SAP-3	-	-	-	-	-	-	-	-	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
12	Instruksi <i>Extended</i> dan <i>Indirect</i>	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
13	Arsitektur MIPS	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
14	Instruksi MIPS	-	-	-	-	-	-	-	-	-	-	-	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
15	Assembly MIPS (bag-1)	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
16	Assembly MIPS (bag-2)	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	

Part 1: Modul I/O dan Device External

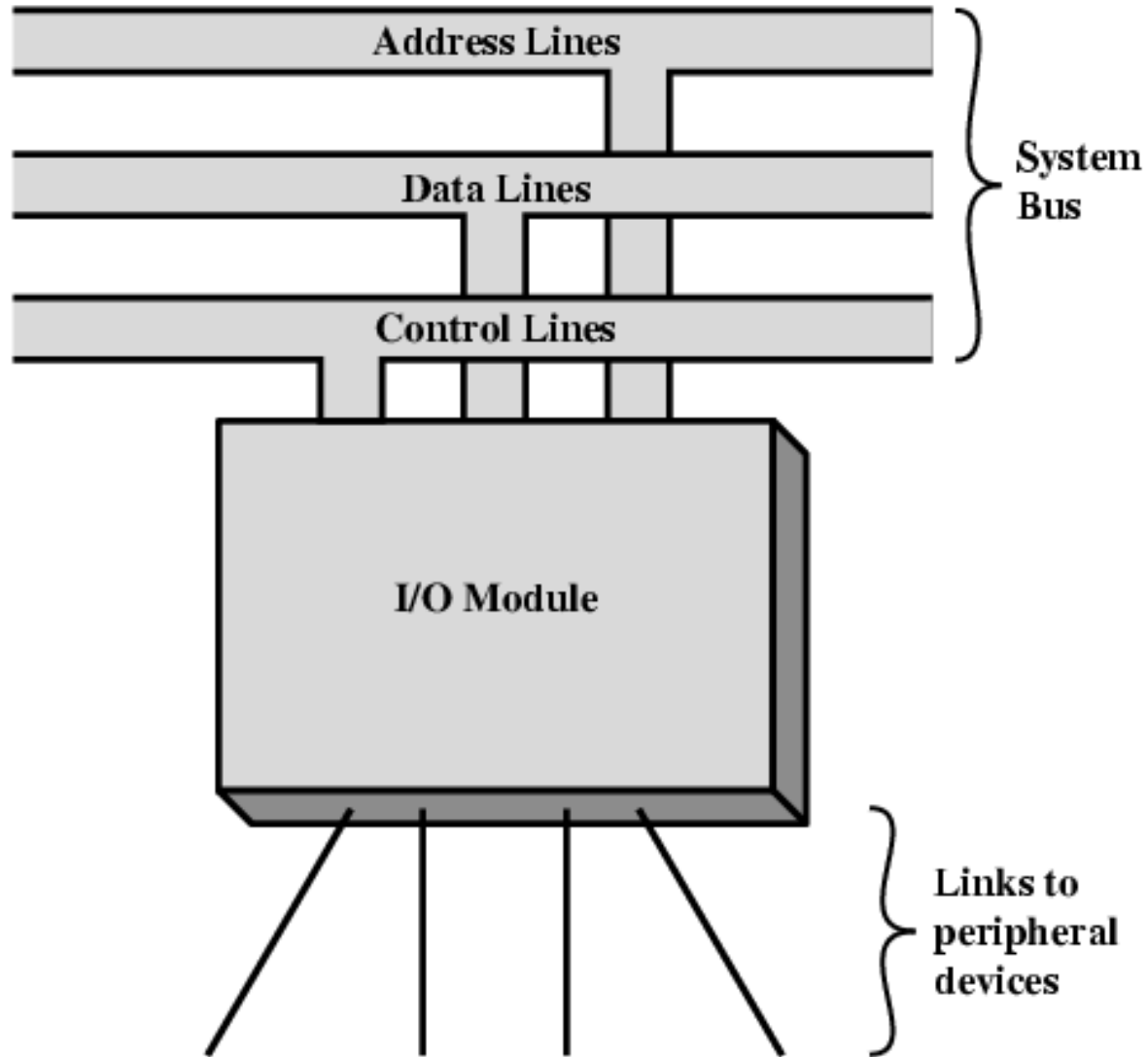
Struktur Komputer – *Input/Output*



- Mengapa perlu modul *Input/Output*?
 - Jenis *peripheral** (*device*) sangat bermacam-macam:
 - Ukuran data yang dapat ditransfer dalam satu saat berbeda
 - Kecepatan berbeda
 - Format data berbeda
 - Dll
 - Kecepatan semua *peripheral* jauh lebih lambat dibanding CPU dan RAM
- Apa fungsi utama modul I/O?
 - Untuk menjembatani *peripheral* dengan CPU dan memori
 - Untuk menjembatani CPU dan memori dengan satu atau beberapa *peripheral*

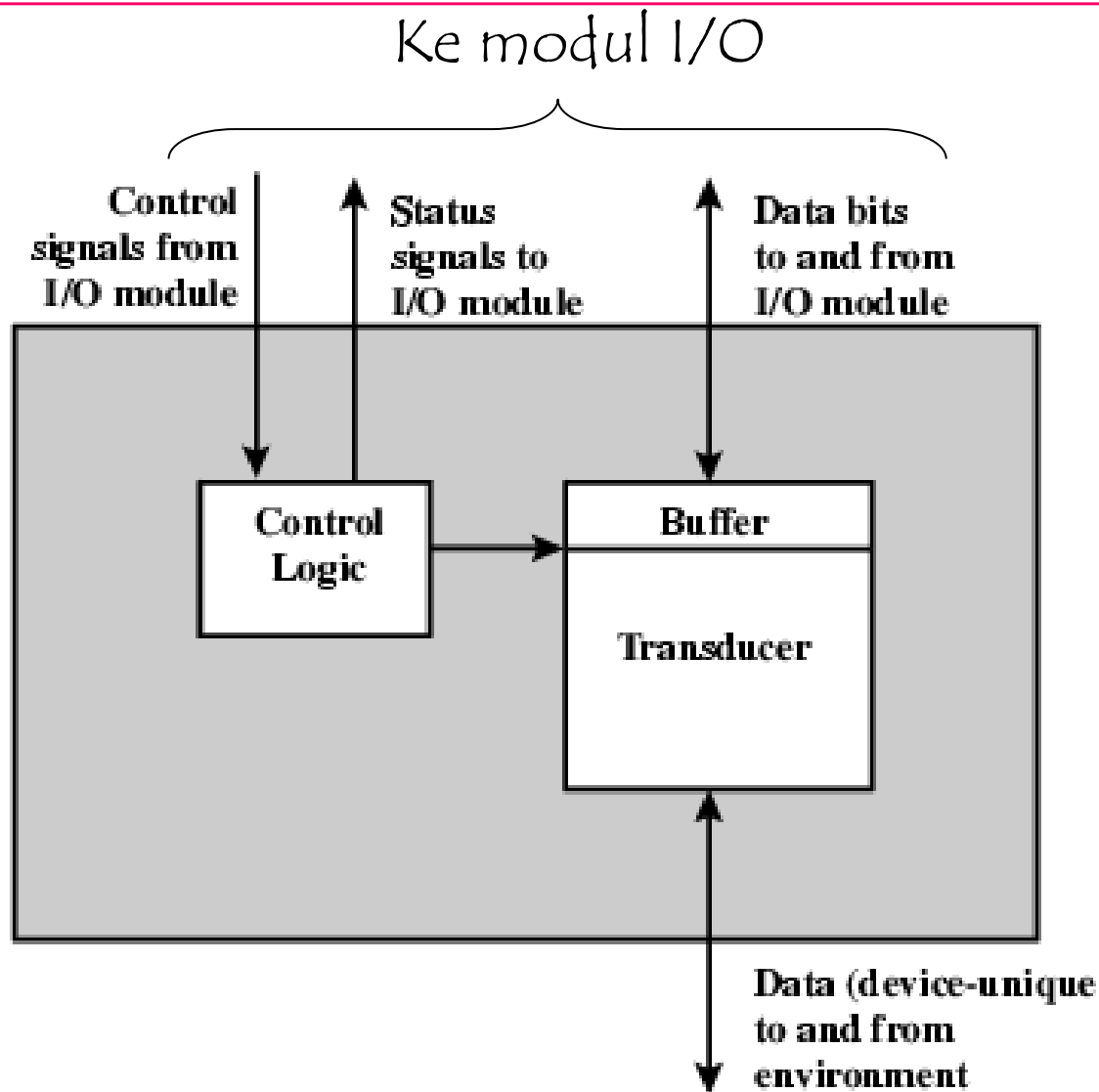
**Peripheral* = *device* eksternal yang terhubung ke modul I/O

Letak Modul I/O



- Jenis-jenis *device* eksternal:
 - *Human readable* → sarana komunikasi manusia dengan mesin (komputer)
 - *Screen, printer, keyboard*
 - *Machine readable* → sarana komunikasi antara komputer dengan *device* lain
 - *Harddisk, sensor, aktuator, dll*
 - *Communication* → sarana komunikasi komputer dengan komputer lain
 - Modem
 - *Network Interface Card (NIC)*

Blok Diagram *Device* Eksternal (1)



Blok Diagram *Device* Eksternal (2)

- **Signal kontrol:**
 - Menentukan apa yang harus dilakukan oleh *device*
 - Misal: INPUT atau READ untuk menerima/membaca data dan OUTPUT atau WRITE untuk mengirimkan data ke *device* lain
- **Signal status:**
 - Untuk mengirimkan status dari *device* (*ready* atau *error*)
- **Jalur data:**
 - Saluran untuk mengirimkan/menerima deretan bit-bit ke/dari modul I/O
- **Control logic:**
 - Menentukan aktifitas dan status *device* eksternal
- **Buffer:**
 - Untuk menampung data dari/ke modul I/O sementara waktu, biasanya berukuran 8 hingga 16 bit
- **Transducer:**
 - Mengubah bentuk data dari signal elektrik, mekanik, temperatur, tekanan, dll menjadi data digital dan sebaliknya

Data Rate Beberapa *Device*

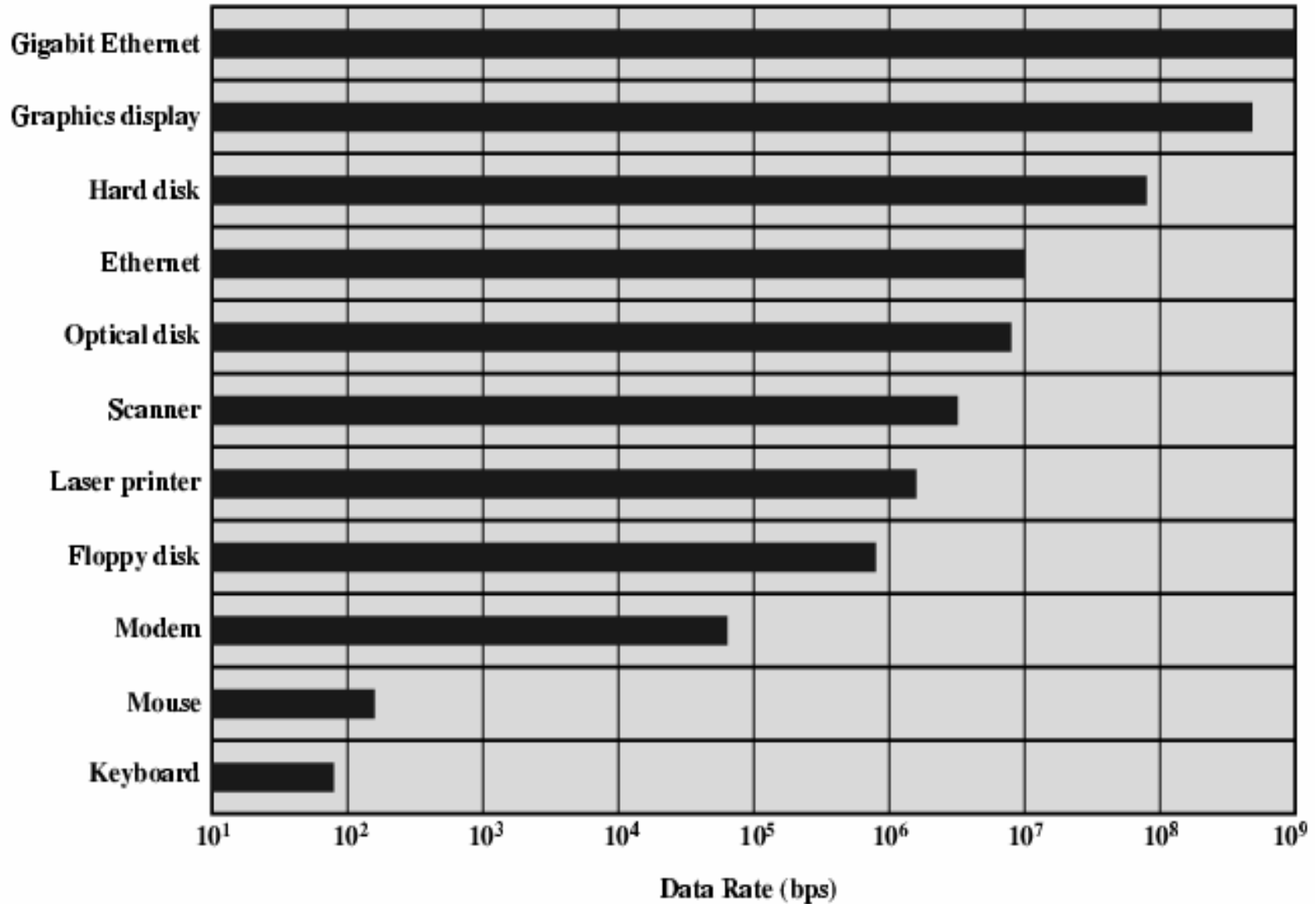
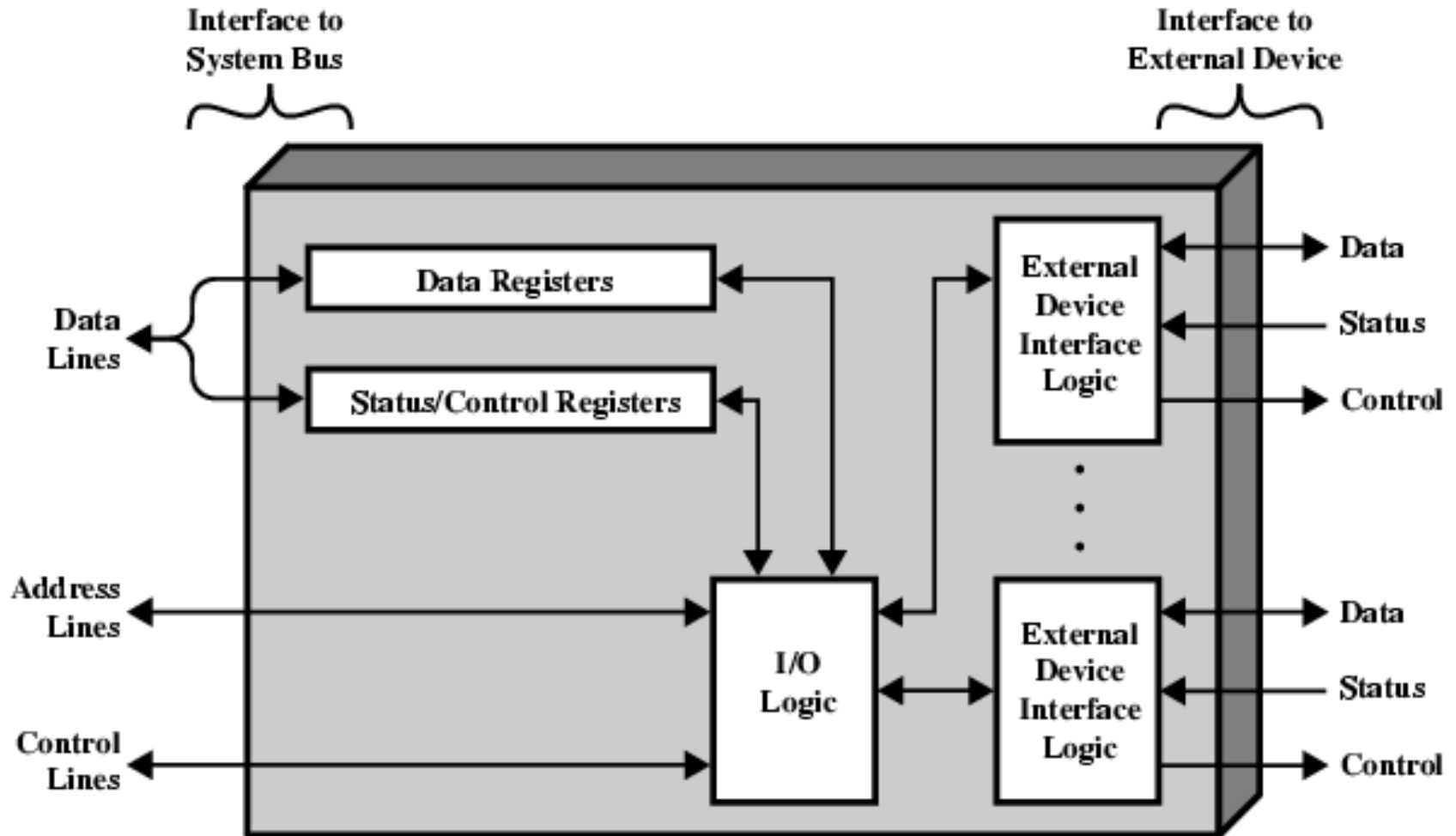


Diagram Modul I/O



Part 2: Apa saja fungsi Modul I/O?

(1) Control & Timing:

- Modul I/O berfungsi sebagai pengatur aliran data antara *resource* internal (CPU, memori) dengan *device* eksternal
- Contoh prosedur transfer data dari *device* → CPU:
 - CPU **memeriksa** status *device* melalui modul I/O
 - Device **memberikan** statusnya melalui modul I/O
 - Jika *ready* → CPU **minta** agar *device* mengirimkan data
 - Modul I/O **menerima** data dari *device*
 - Modul I/O **mengirimkan** data ke CPU

(2) CPU Communication:

- Modul I/O berfungsi sebagai media komunikasi dari CPU menuju *device* eksternal
- **Apa yang dilakukan modul I/O?**
 - Men-*decode* perintah/*command* dari CPU
 - Contoh perintah untuk *harddisk*: READ SECTOR, WRITE SECTOR, SEEK track number, dan SCAN record ID
 - Menjadi media untuk pertukaran data
 - Melaporkan status *device* (*status reporting*)
 - Misal: BUSY atau READY
 - Memeriksa/men-*decode* alamat yang dikirimkan oleh CPU (*address recognition*)

(3) *Device Communication:*

- Modul I/O berfungsi sebagai media komunikasi dari *device* eksternal menuju CPU
- **Apa yang dilakukan modul I/O?**
 - Meneruskan perintah/*command* dari CPU ke *device*
 - **Meneruskan status dari *device* ke CPU**
 - Meneruskan data dari *device* ke CPU

(4) *Data Buffering*

- Modul I/O berfungsi sebagai penampung data sementara baik dari CPU/memori maupun dari *peripheral*
- Mengapa data perlu di-*buffer*?
 - Kecepatan *device* sangat beragam
 - Kecepatan *device* \lll kecepatan CPU
- Contoh:
 - Data dari CPU:
 - Langsung ditaruh di *buffer*
 - Diberikan ke *device* sesuai dengan kecepatan (“daya serap”) *device*
 - Data dari *device*:
 - Dikumpulkan dulu di *buffer*
 - Setelah periode tertentu baru dikirimkan ke CPU → lebih efektif

(5) *Error Detection*

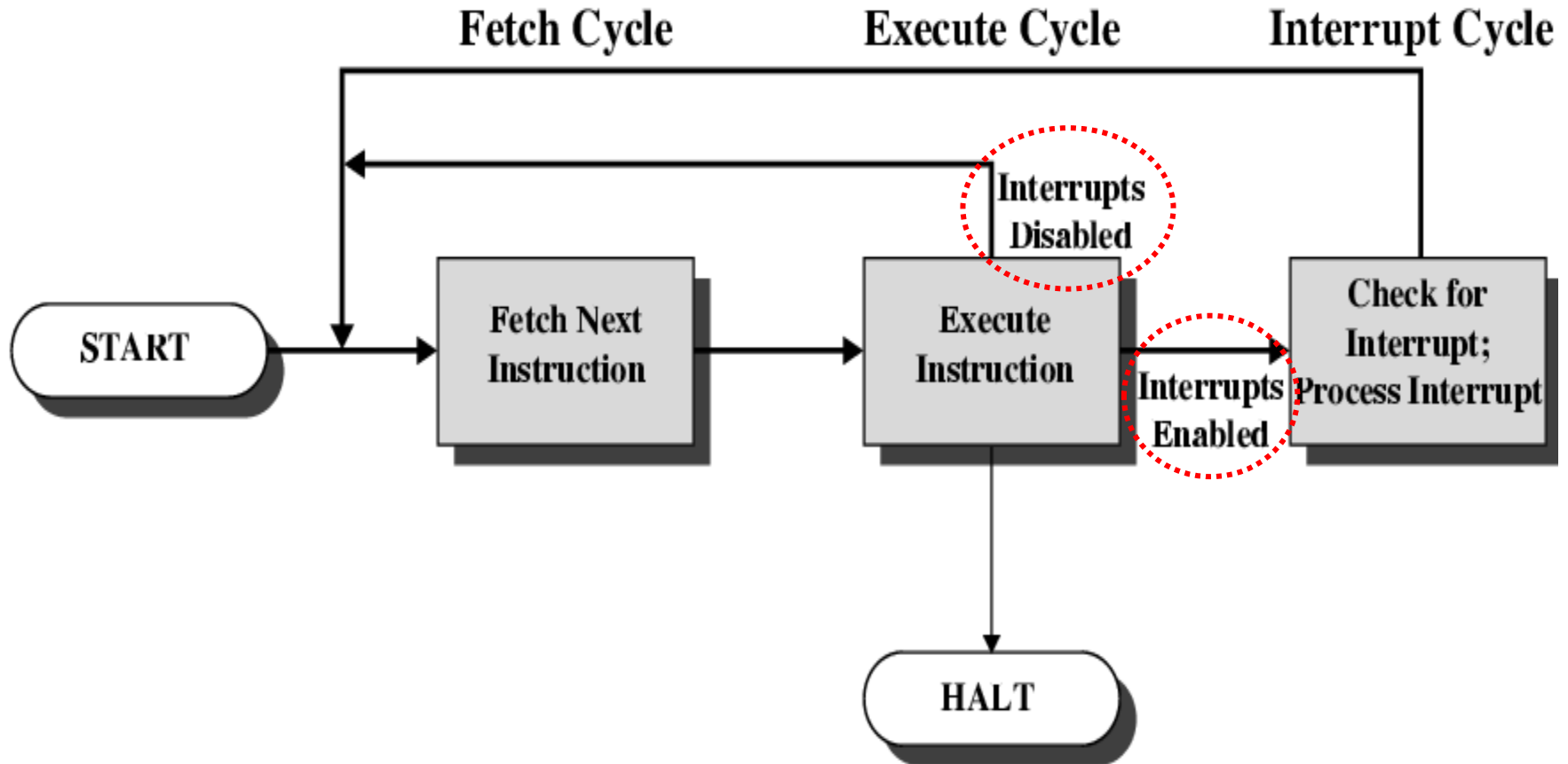
- Modul I/O berfungsi sebagai pendeteksi kesalahan yang ditimbulkan oleh *device*
- Contoh kesalahan:
 - *Paper jam*
 - *Bad sector*
 - Kertas habis
 - Terjadi perubahan bit-bit data
 - Dll
- Contoh metode deteksi:
 - Bit parity

Part 3: Bagaimana cara menangani *interrupt*?

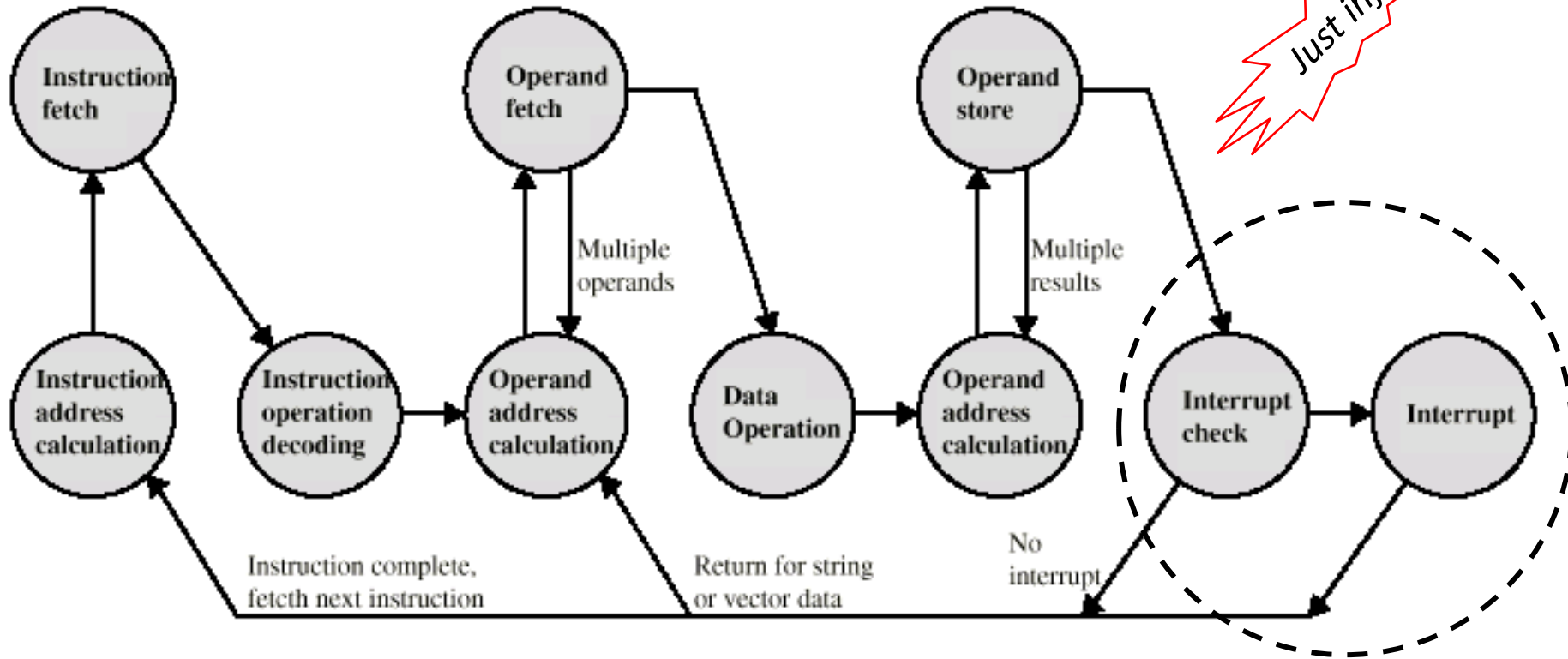
Apakah *Interrupt* itu?

- Adalah mekanisme untuk **menghentikan** sementara waktu urutan eksekusi program yang normal (*sequence*) jika:
 - kondisi tertentu telah terjadi
 - ada program lain yang lebih mendesak untuk dieksekusi
- Apa penyebab *interrupt*?
 - Program
 - Misal: *overflow*, *division by zero*, akses ke illegal memori, dll
 - *Timer*
 - Dihasilkan oleh timer prosesor internal
 - Digunakan pada *pre-emptive multi-tasking*
 - I/O
 - Dihasilkan oleh I/O *controller* (eksekusi telah selesai atau ada kesalahan)
 - *Hardware failure*
 - Misal: *memory parity error*, *power failure*, dll

Siklus interrupt (1)



Siklus *interrupt* - State Diagram

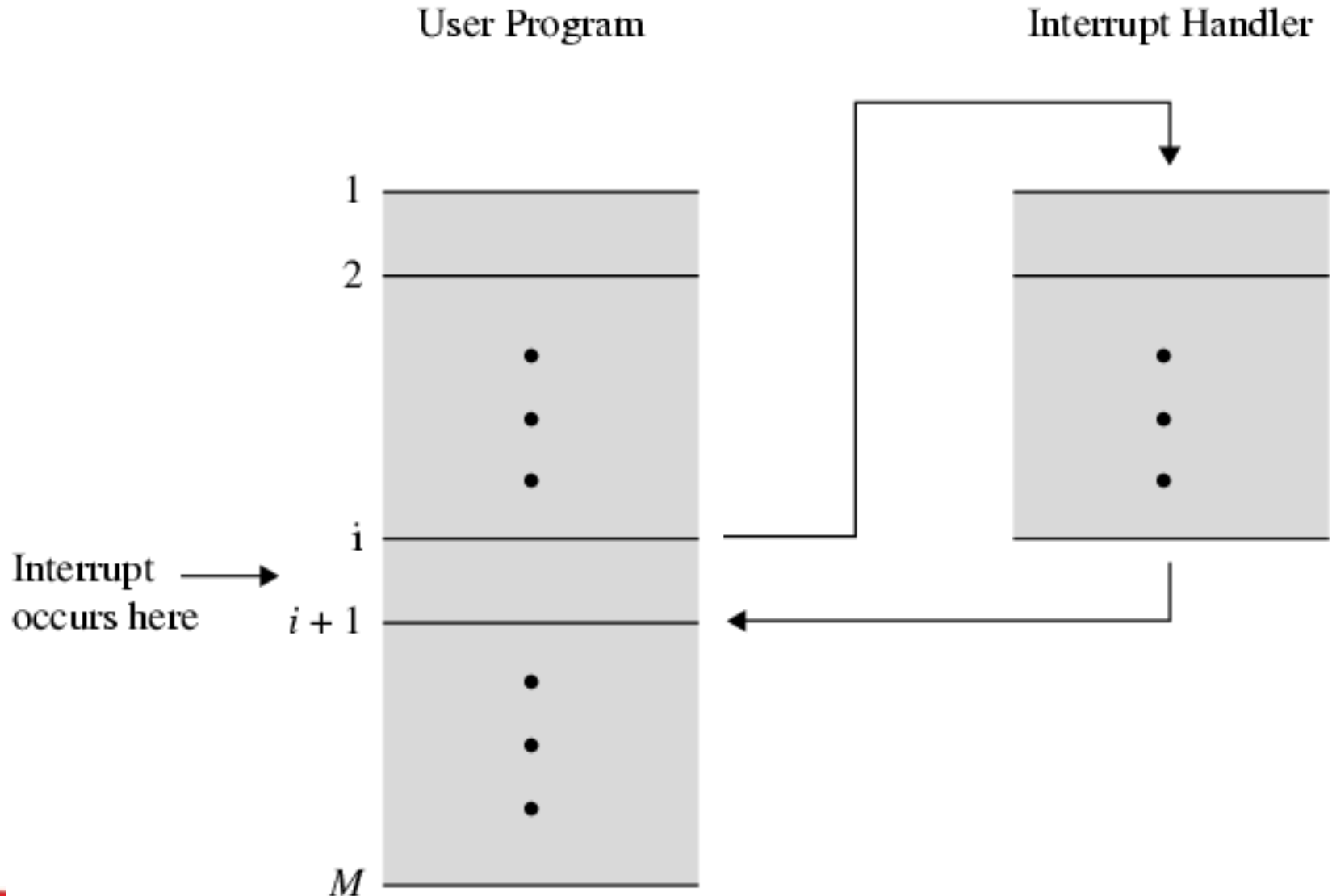


Just info

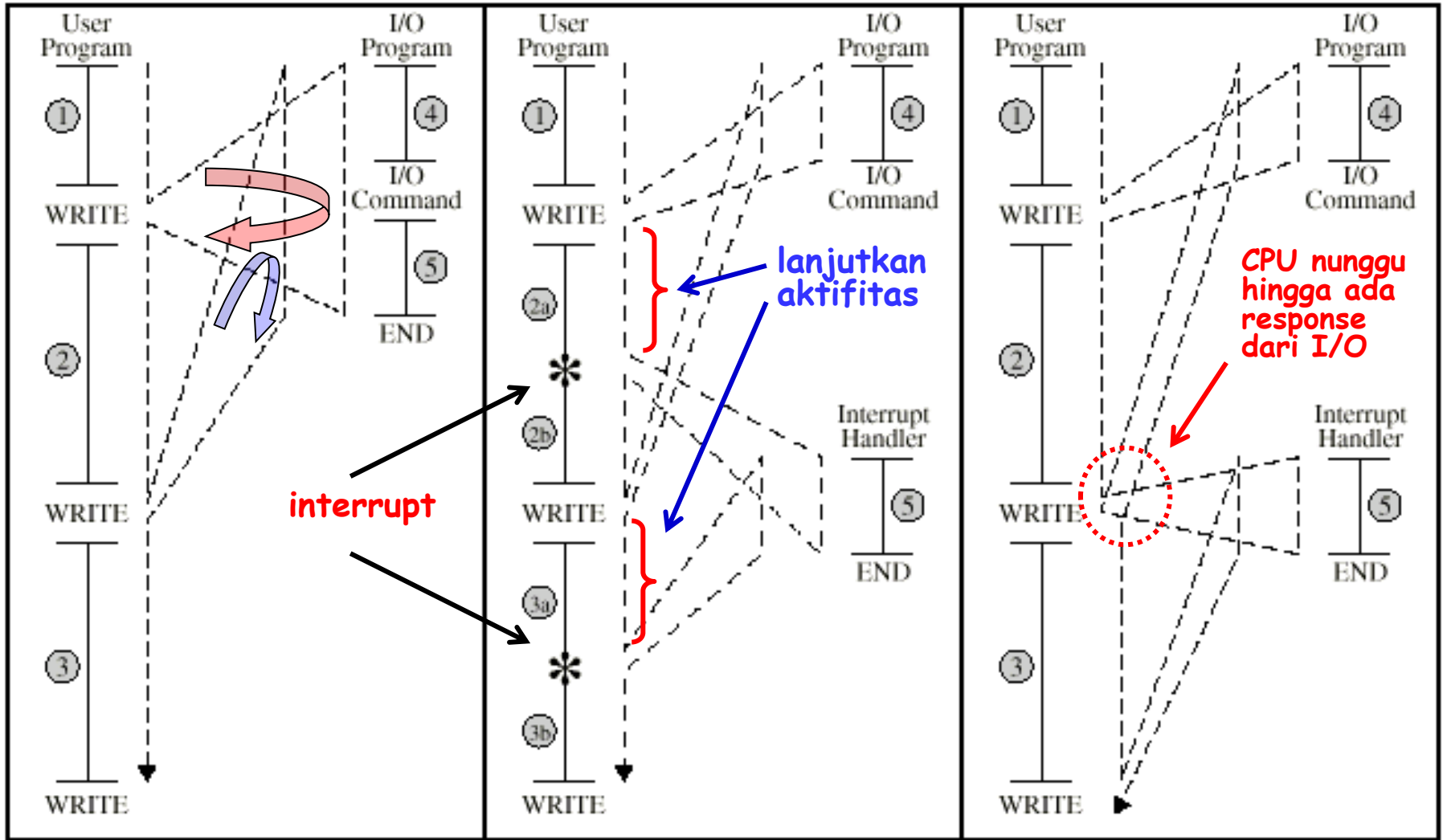
siklus interrupt

- Pengecekan *interrupt* ditambahkan pada siklus instruksi
- Prosesor memeriksa apakah terjadi *interrupt*
- Jika tidak ada *interrupt* => kerjakan instruksi berikutnya
- Jika ada *interrupt* (ditandai adanya *signal interrupt*)
 - Tunda program yang sedang dieksekusi
 - Simpan *context* (alamat instruksi, data, dll)
 - PC = alamat awal routine *interrupt handler*
 - Kerjakan *interrupt* sampai selesai
 - Kembalikan *context* (PC=alamat saat terjadi *interrupt*)
 - Lanjutkan program yang tertunda

Contoh *Interrupt* (1)



Contoh *Interrupt* (2)



(a) No interrupts

(b) Interrupts; short I/O wait

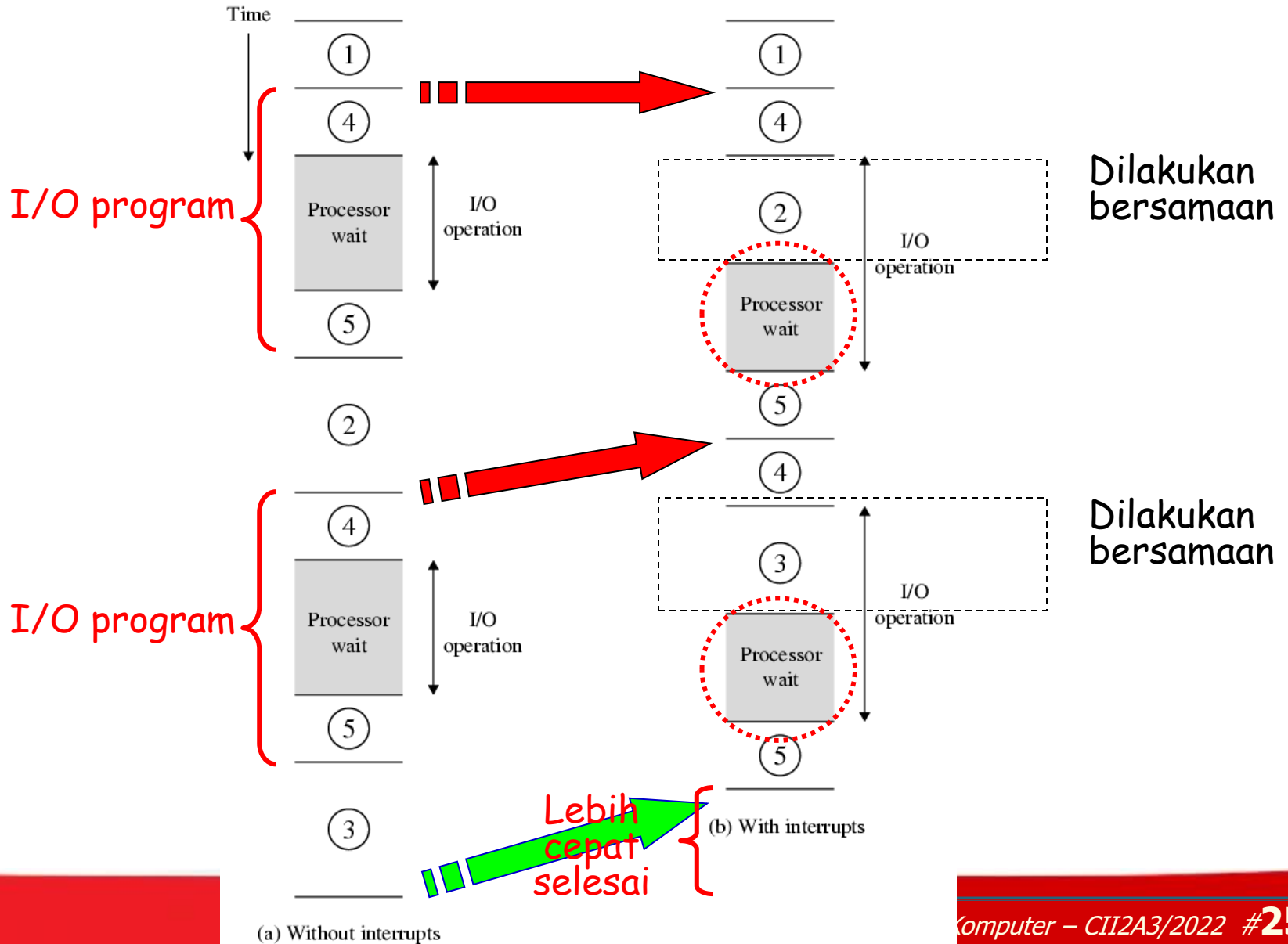
(c) Interrupts; long I/O wait

1, 2, 3: Program internal (tanpa I/O)

4: Persiapan I/O

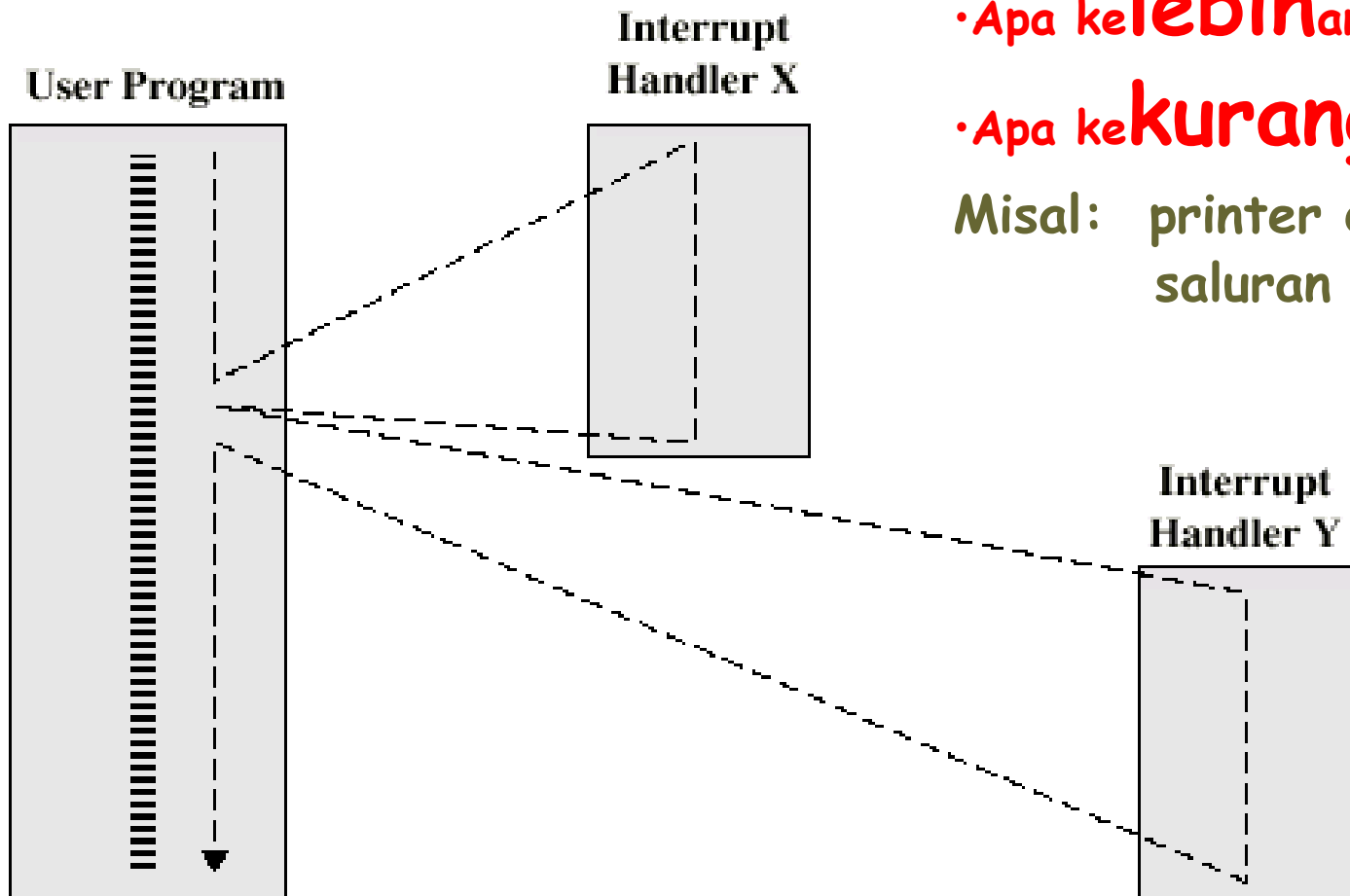
5: Tanda akhir I/O

Efisiensi waktu pada *Long I/O* Wait



- Ada 2 metode yang dapat digunakan:
 - *Disable interrupts*
 - Bila prosesor sedang menangani sebuah *interrupt*, maka *interrupt* berikutnya **diabaikan** (*disable*)
 - Bila *interrupt* pertama telah selesai, prosesor memeriksa apakah masih ada *interrupt* yang lain
 - Eksekusi *interrupt* dilakukan secara berurutan (*sekuensial*)
 - *Interrupt* bertingkat (ada prioritas)
 - *Interrupt* berprioritas rendah dapat *diinterrupt* oleh *interrupt* berprioritas tinggi
 - *Interrupt* yang ter-*interrupt* dieksekusi lagi bila *interrupt* yang lebih tinggi telah selesai ditangani

Multiple Interrupts - Sequential

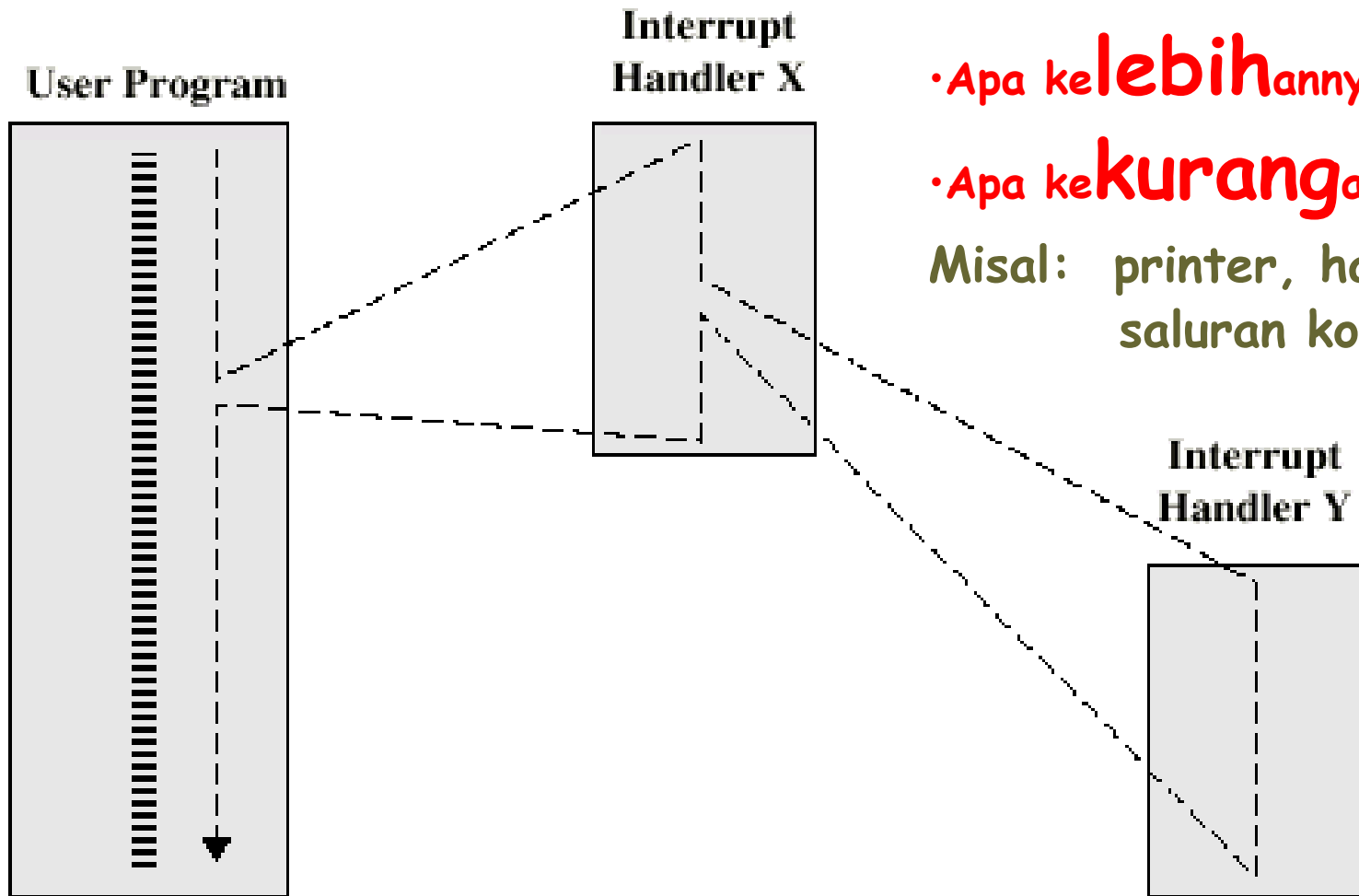


• Apa **kelebihannya**?

• Apa **kekurangannya**?

Misal: printer dengan saluran komunikasi

Multiple Interrupts – Bertingkat (1)

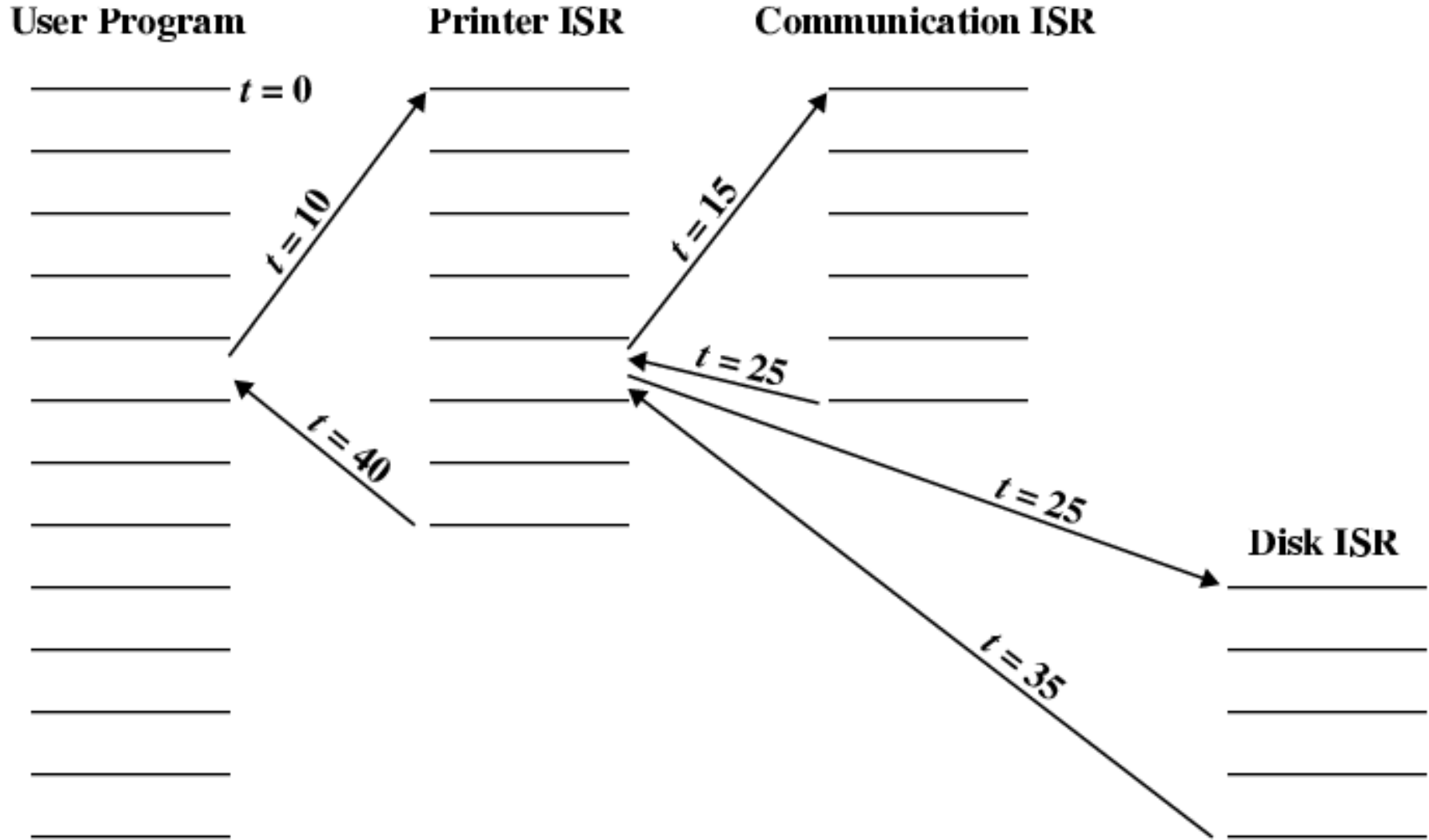


• Apa **kelebihannya** ?

• Apa **kekurangannya** ?

Misal: printer, harddisk, dan saluran komunikasi

Multiple Interrupts – Bertingkat (2)



Part 4: Bagaimana teknik *Programmed I/O* bekerja?

(1) *Programmed I/O*

- I/O terjadi pada saat program yang di dalamnya terdapat perintah I/O dieksekusi
- Eksekusi I/O terus menerus melibatkan prosesor

(2) *Interrupt driven I/O*

- I/O terjadi pada saat perintah I/O dieksekusi
- Sesudah perintah I/O dieksekusi → CPU melanjutkan eksekusi perintah lainnya → tidak terlibat terus menerus
- CPU berperan lagi jika sudah ada *interrupt* dari *device* (modul I/O)

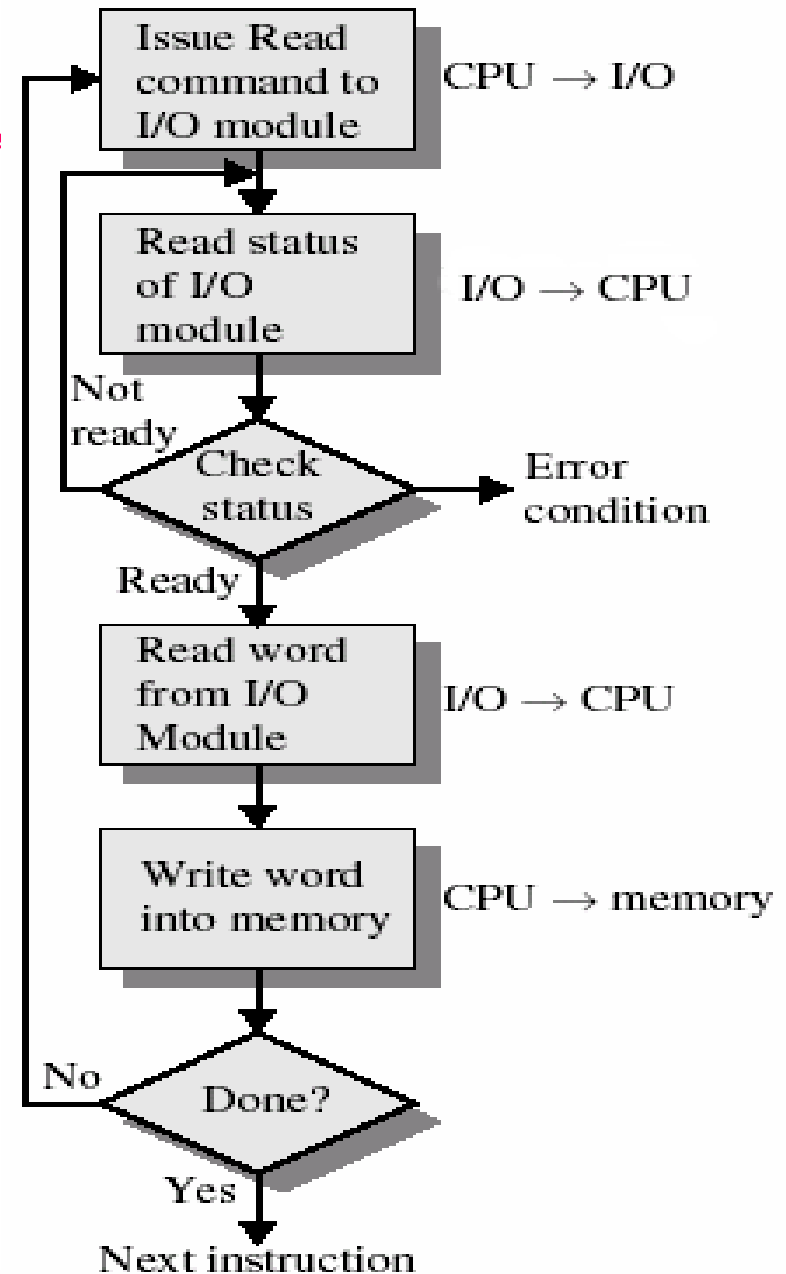
(3) *Direct Memory Access (DMA)*

- Transfer data ditangani oleh sebuah prosesor I/O khusus

Programmed I/O (1)

- CPU terlibat langsung dalam mengendalikan I/O:
 - Membaca status
 - Menerima/mengirim perintah
 - Transfer data
- (-) Selama menunggu operasi I/O selesai → CPU tidak melakukan apa-apa → Waktu CPU terbuang percuma
- (-) CPU harus selalu memeriksa status modul I/O
- (-) Performansi sistem menurun

Contoh prosedur READ data



• CPU:

- Mengirim perintah ke I/O
- Menunggu hingga aktifitas I/O selesai
- CPU memeriksa bit-bit status secara periodik

➤ Modul I/O:

- ❖ Melaksanakan permintaan CPU
- ❖ Men-set bit-bit status
- ❖ **Tidak berinisiatif** memberi laporan status
- ❖ Tidak *meng-interrupt* CPU

- Apa yang dilakukan CPU?
 - Mengirimkan **alamat** modul I/O (dan alamat *device* jika dalam modul tersebut terpasang lebih dari satu *device*)
 - Mengirimkan perintah (*command*):
 - Control:
 - Untuk mengaktifkan *peripheral*
 - Untuk menyuruh *peripheral* melakukan sesuatu
 - » Misal: disk berputar, head bergerak, dsb.
 - Test:
 - Untuk memeriksa status *device*
 - » Apakah ada tegangan?, apakah terjadi gangguan?, dsb
 - Read:
 - Untuk meminta data dari *peripheral* melalui modul I/O (misal baca data dari *harddisk*)
 - Write:
 - Untuk memberikan data ke *peripheral* (misal menulis data ke *harddisk*)

- Jenis pengalamatan I/O:

- (a) *Memory mapped I/O*

- Ruang alamat digunakan oleh memori dan I/O bersama-sama
 - Jumlah alamat untuk memori berkurang
 - CPU memperlakukan I/O seperti memori biasa
 - Tidak ada perintah khusus bagi I/O (perintah yang digunakan mirip dengan perintah untuk memori)



Just info

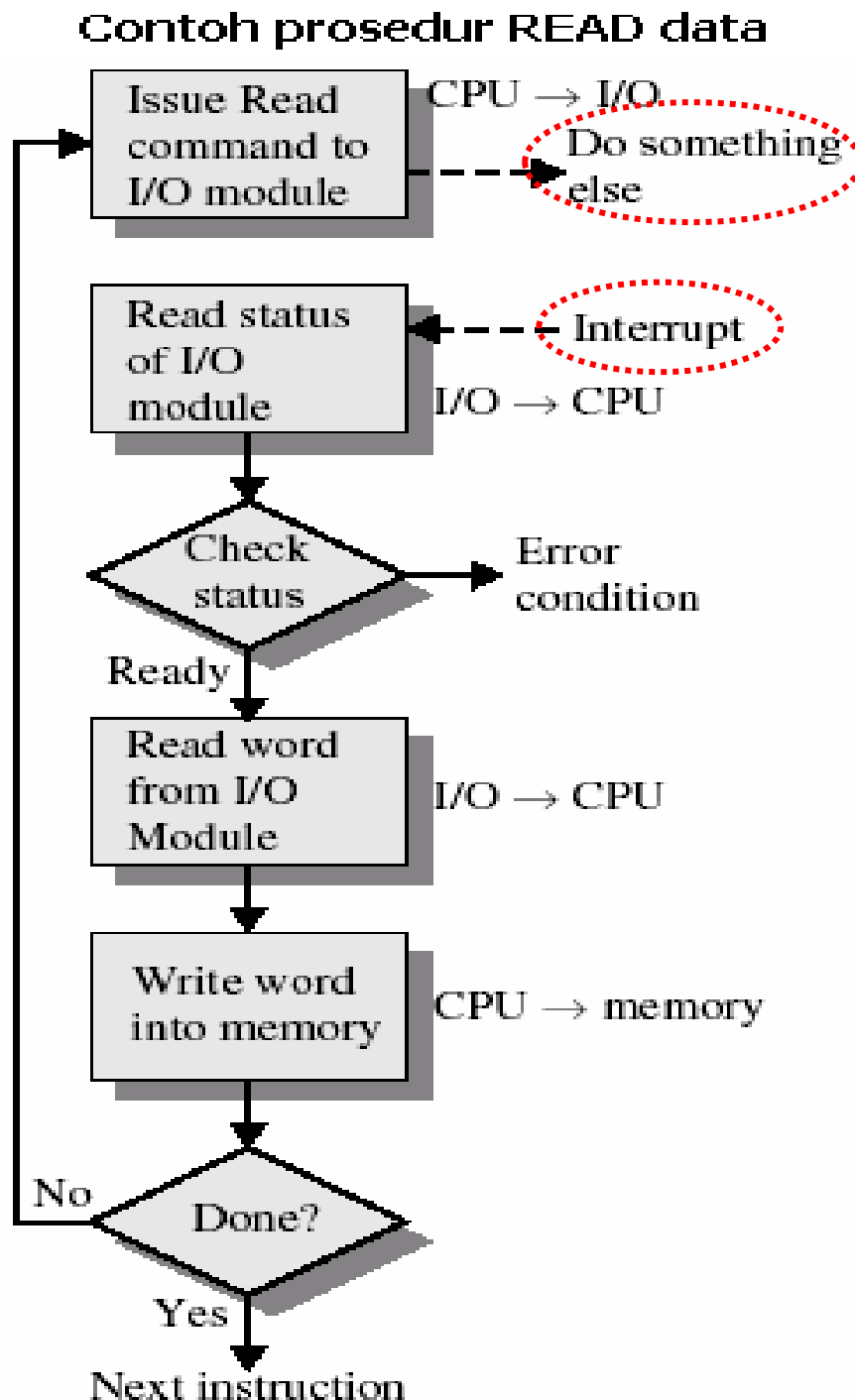
- (b) *Isolated I/O*

- Memori dan I/O menggunakan ruang alamat yang sama secara bergantian
 - Jumlah alamat untuk I/O sama banyak dengan alamat untuk memori
 - Diperlukan *select line* untuk membedakan antara memori dengan I/O
 - Diperlukan perintah khusus untuk I/O

Part 5: Bagaimana teknik *Interrupt Driven I/O* bekerja?

Interrupt Driven I/O (1)

- Modul I/O meng-interrupt CPU jika device telah menyelesaikan pekerjaannya
- (+) CPU tidak perlu menunggu operasi I/O selesai → CPU dapat mengerjakan program lain
- (+) CPU tidak perlu memeriksa status device terus menerus



• CPU:

- Mengirim perintah ke I/O
- Mengerjakan program berikutnya
- CPU memeriksa status *interrupt*
- Baca data
- Simpan data ke memori

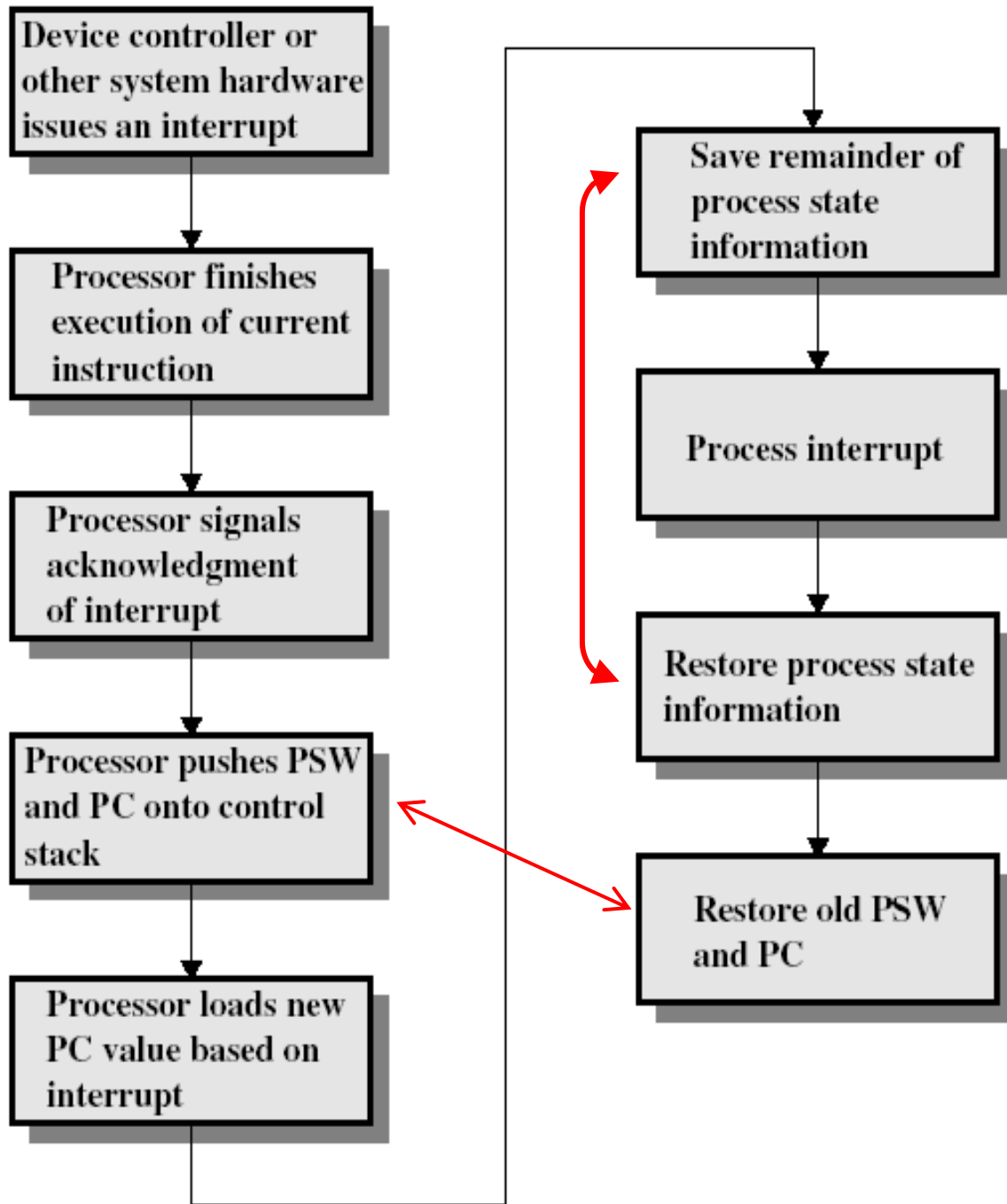
➤ Modul I/O:

- ❖ Melaksanakan permintaan CPU
- ❖ Bila telah selesai/ada masalah → kirim interrupt

- Apa yang dilakukan CPU?
 - Mengirimkan perintah baca
 - Kerjakan program lain
 - Periksa keberadaan *interrupt* setiap akhir siklus instruksi
 - Jika terjadi *interrupt*:
 - Simpan *context* (data di register, PC, dll)
 - Tangani *interrupt*
 - Ambil data dan simpan ke memori

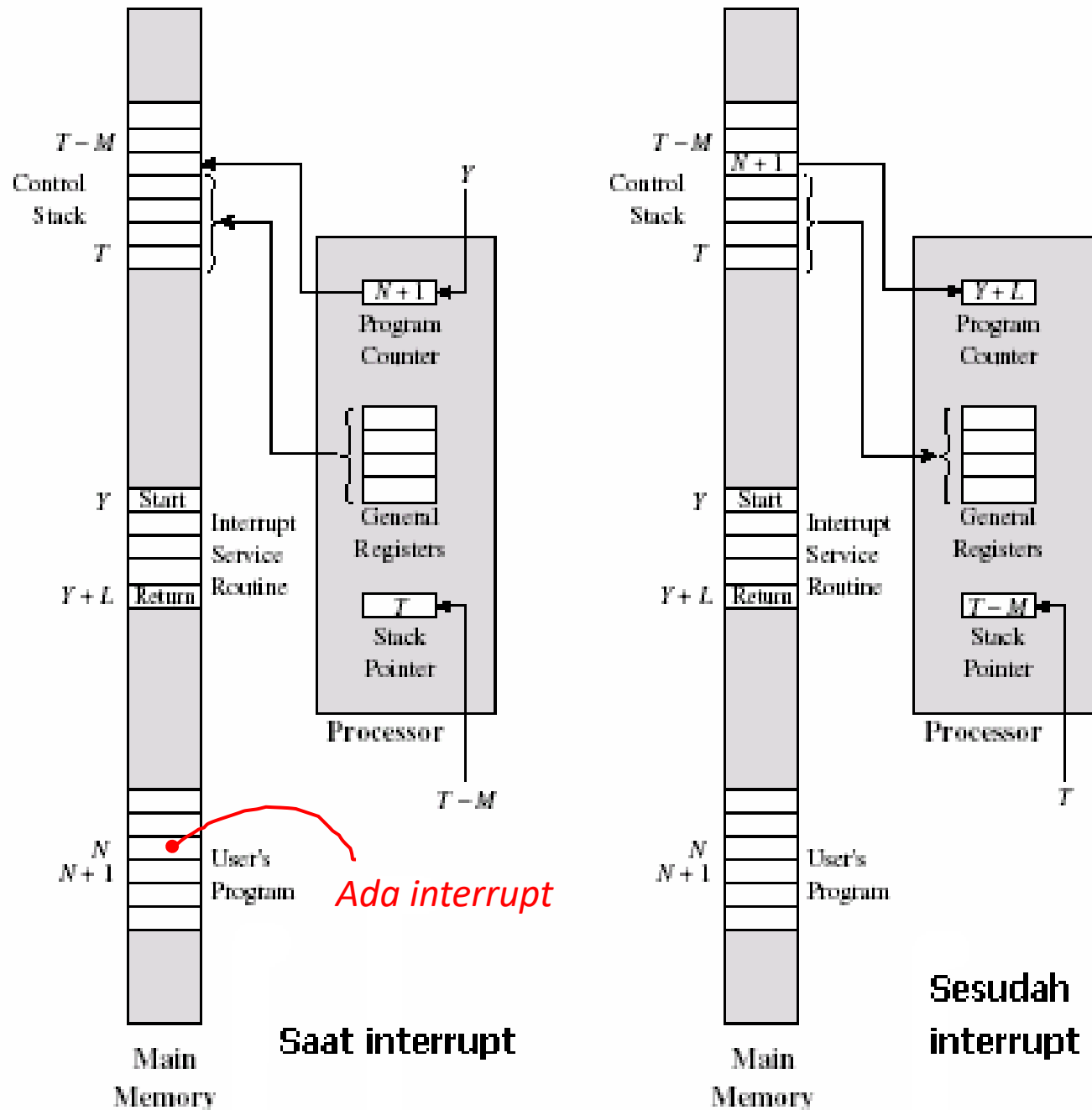
Interrupt Driven I/O (4)

- *Proses interrupt*
- *PSW = Program Status Word (isi register)*



Interrupt Driven I/O (5)

Contoh interrupt



Part 6: Bagaimana cara mengetahui device yang mengeluarkan *interrupt*?

- Bagaimana cara menentukan/mengetahui modul/*device* yang mengirimkan *interrupt*?
- Bagaimana cara menangani *multiple interrupt*?
 - Misal: program *interrupt handler* ter-*interrupt* lagi

- Cara menentukan asal *interrupt*:

(1) Digunakan banyak jalur *interrupt* → Satu modul satu jalur

(-) Tidak praktis karena harus menyediakan sejumlah jalur khusus untuk *interrupt*

(-) Jumlah *device* yang dapat dipasang terbatas

(2) Software poll (*polling*)

(-) Lambat, karena harus memeriksa modul I/O satu persatu

Mekanisme:

- Jika CPU mendeteksi adanya *interrupt*
- CPU menanyakan ke setiap modul I/O untuk mengetahui asal *interrupt* (*polling*)
 - Misal dengan mengirimkan signal TEST I/O
 - Modul pengirim *interrupt* akan menjawab signal tersebut

- Cara menentukan asal *interrupt*: (cont'd)

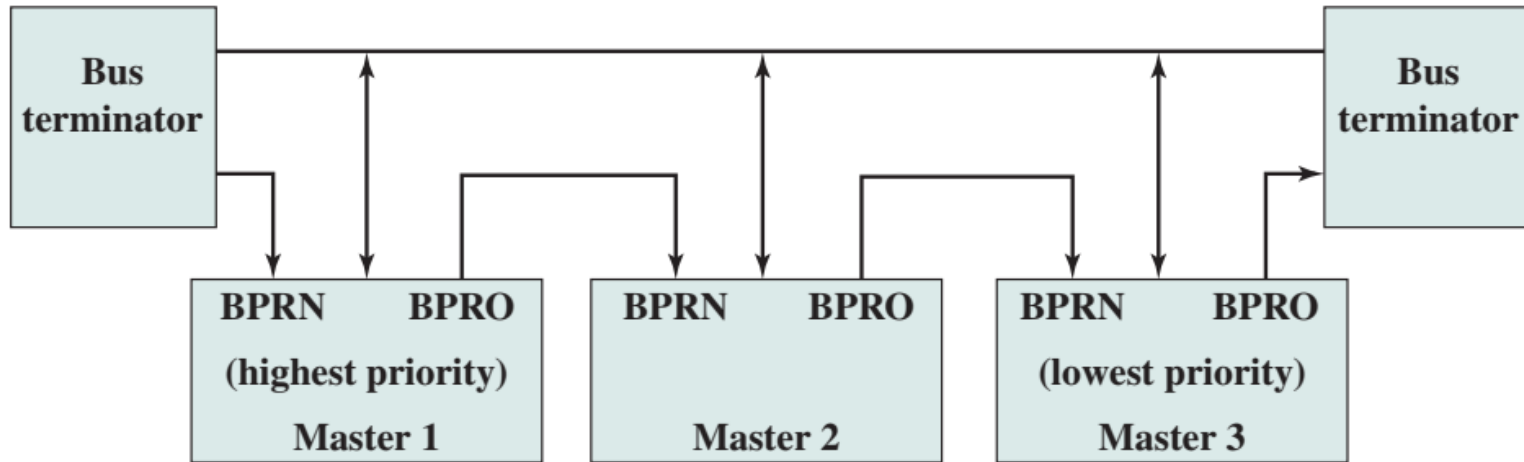
- (3) *Daisy chain* atau *hardware poll*

- Digunakan satu jalur *interrupt* yang menghubungkan setiap modul I/O
 - Modul satu dengan modul lainnya terhubung secara serial

- Mekanisme:

- Modul I/O mengirimkan *interrupt* melalui jalur *interrupt* bersama
 - CPU mendeteksi adanya *interrupt* → mengirimkan signal ke sebuah modul I/O
 - Signal akan diestafetkan dari satu modul ke modul lainnya
 - Hanya modul yang mengirimkan *interrupt* yang akan memberi jawaban
 - Jawaban berupa word (*vector*) yang ditaruh di jalur data. Isi word = alamat modul I/O atau identitas unik lainnya
 - *Vector* digunakan sebagai *pointer* untuk menunjuk langsung ke *device* sumber *interrupt* sebelum menjalankan *interrupt handler* → disebut ***vectored interrupt***

Teknik Identifikasi *Interrupt* (4)



Contoh Daisy Chain

- Cara menentukan asal interrupt: (cont'd)

(4) *Bus master*

- Pengiriman *interrupt* dilakukan bila modul I/O telah 'menguasai' (master) bus

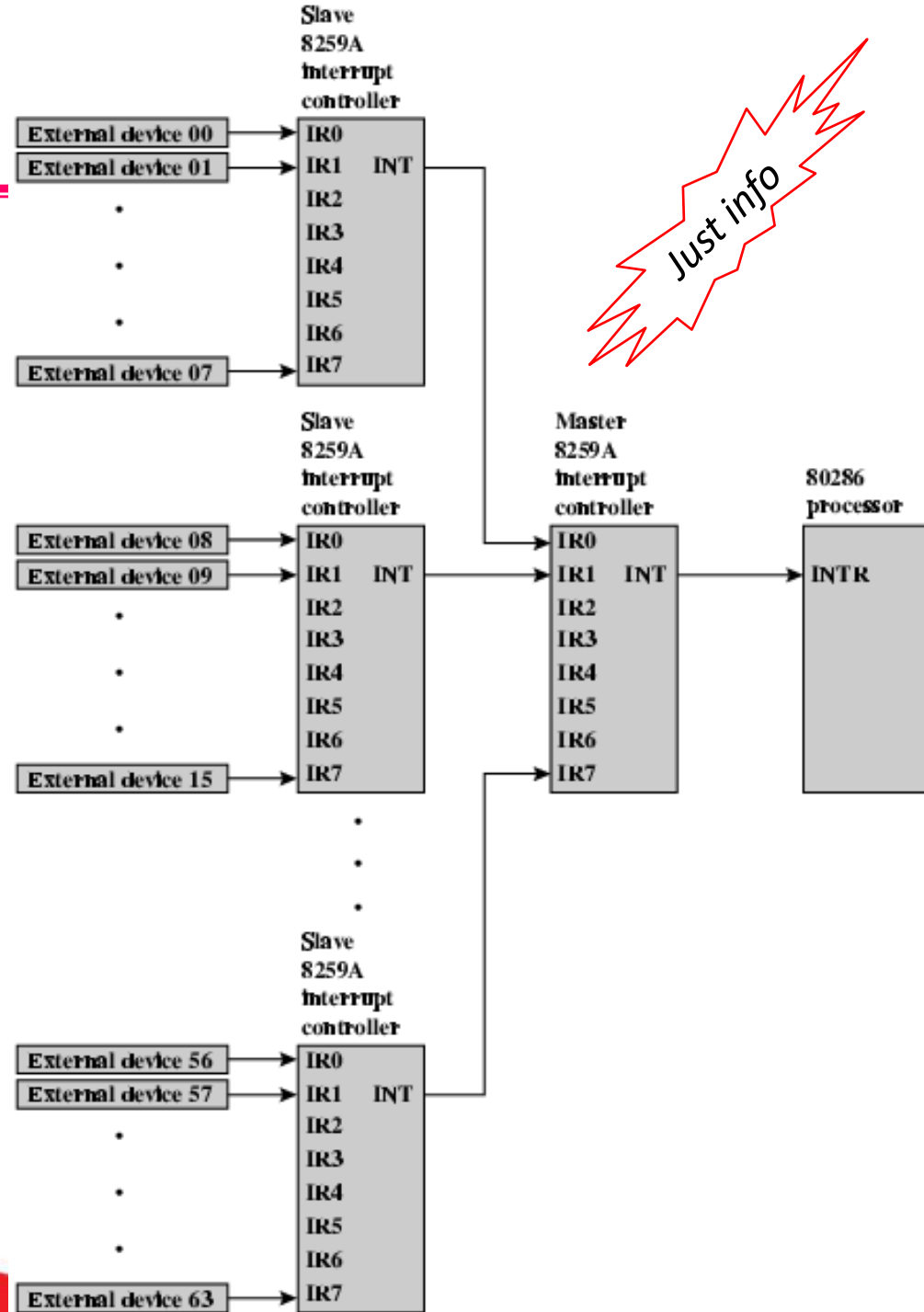
Mekanisme:

- Modul I/O mengirimkan permintaan untuk menggunakan bus
- Arbiter memberi kesempatan kepada I/O modul → hanya satu modul dalam satu saat
- Modul I/O mengirimkan interrupt
- CPU mendeteksi adanya *interrupt* dan memberi respons melalui jalur *acknowledge*
- Modul I/O mengirimkan *word* (*vector*) ke jalur data

End of Part 6

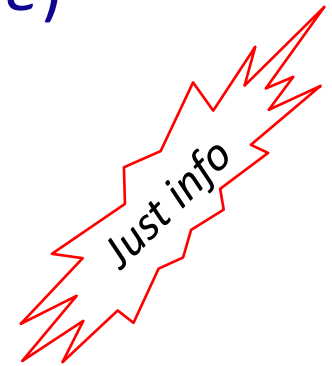
Contoh 82C59A (1)

- Contoh *interrupt controller*: 82C59A
- Setiap controller 82C59A dapat menangani 8 *interrupt*
- Bila jumlah *device* lebih banyak → 82C59A disusun secara bertingkat



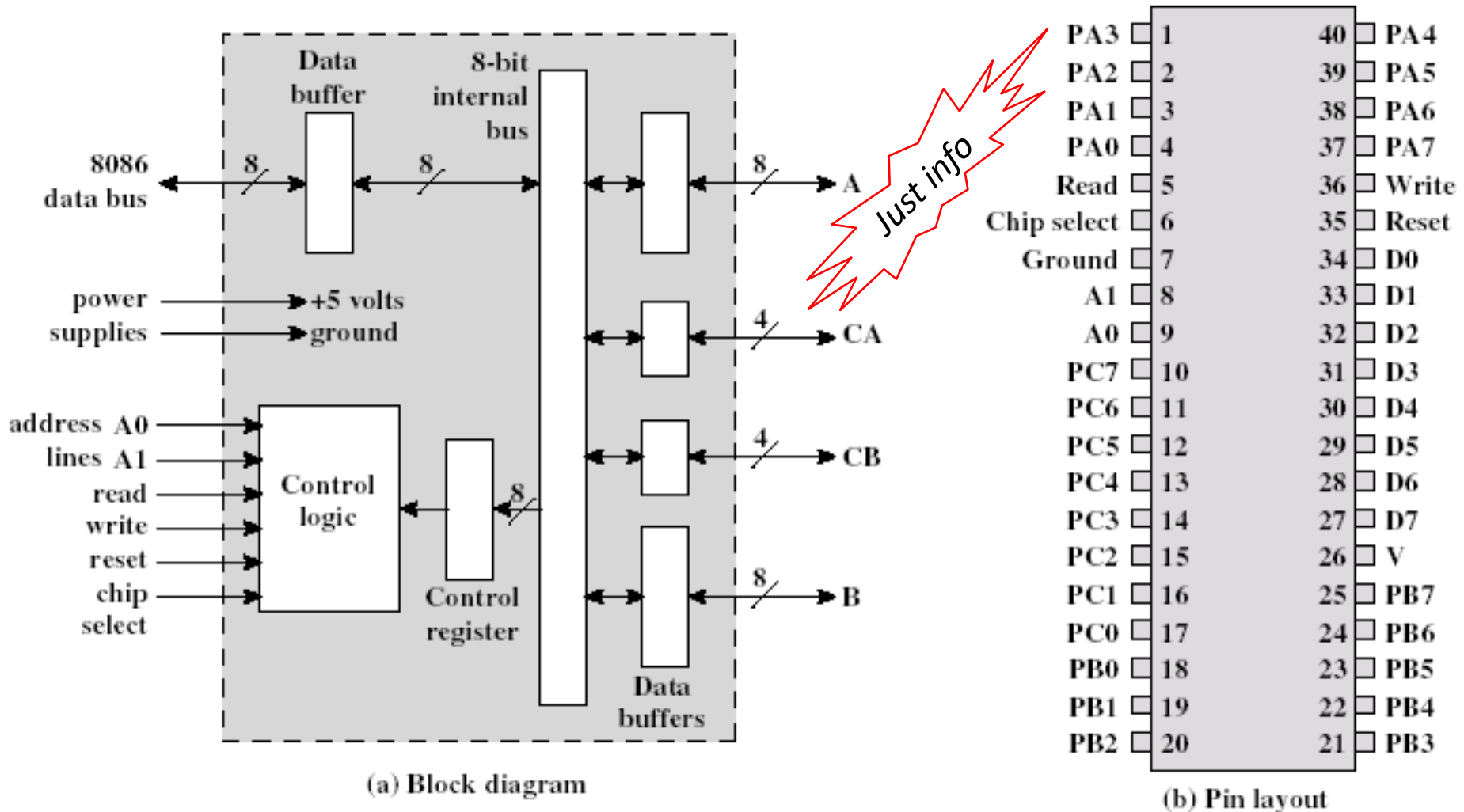
Contoh 82C59A (2)

- Cara kerja:
 - 8259A menerima *interrupt* dari *device*
 - 8259A menentukan prioritas (bila lebih dari satu *device* yang *meng-interrupt*)
 - 8259A mengirimkan signal ke CPU 8086 (melalui jalur INTR)
 - CPU mengirimkan respons (*acknowledge*)
 - 8259A menaruh vector ke bus data
 - CPU memproses *interrupt*



Contoh Intel 82C55A

Programmable Peripheral Interface (PPI) (1)

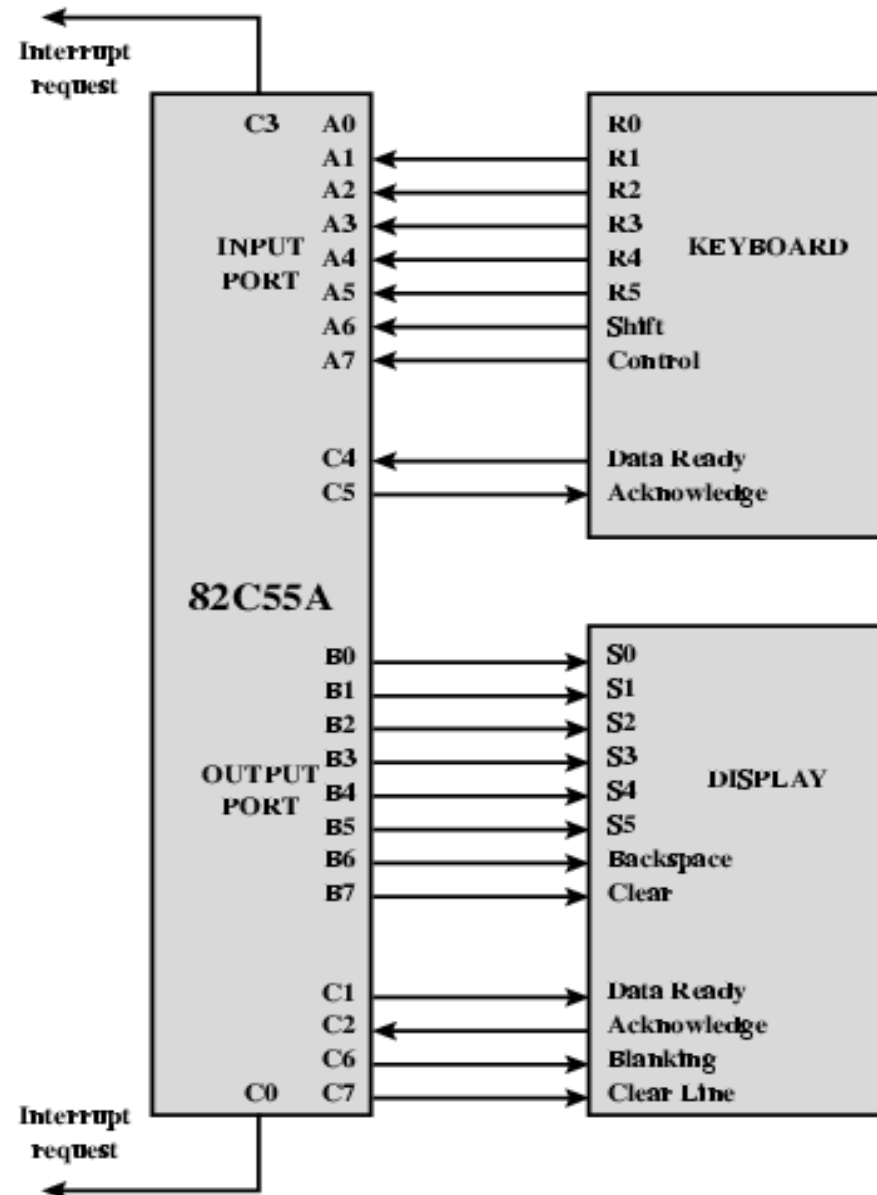




- Dapat digunakan untuk *programmed I/O* maupun *interrupt driven I/O*
- Terdiri dari single chip
- Termasuk modul I/O serbaguna yang dirancang untuk CPU 80386
- Terdapat 24 jalur I/O (PA0-PA7 + PB0-PB7 + PC0-PC7) yang dapat diprogram dengan 80386 sesuai dengan kebutuhan
- PC0-PC7 digunakan untuk jalur kontrol dan signal



Contoh 82C55A Untuk Mengontrol *Keyboard/Display*

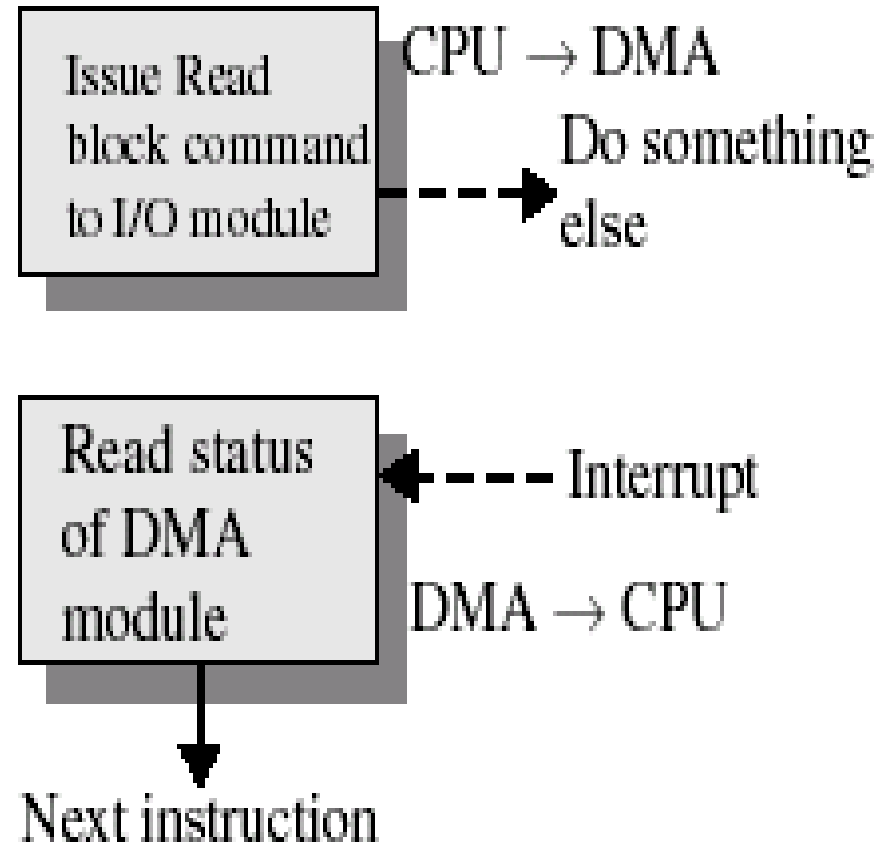


Just info

Part 7: Bagaimana teknik *DMA* bekerja?

Direct Memory Access (DMA)

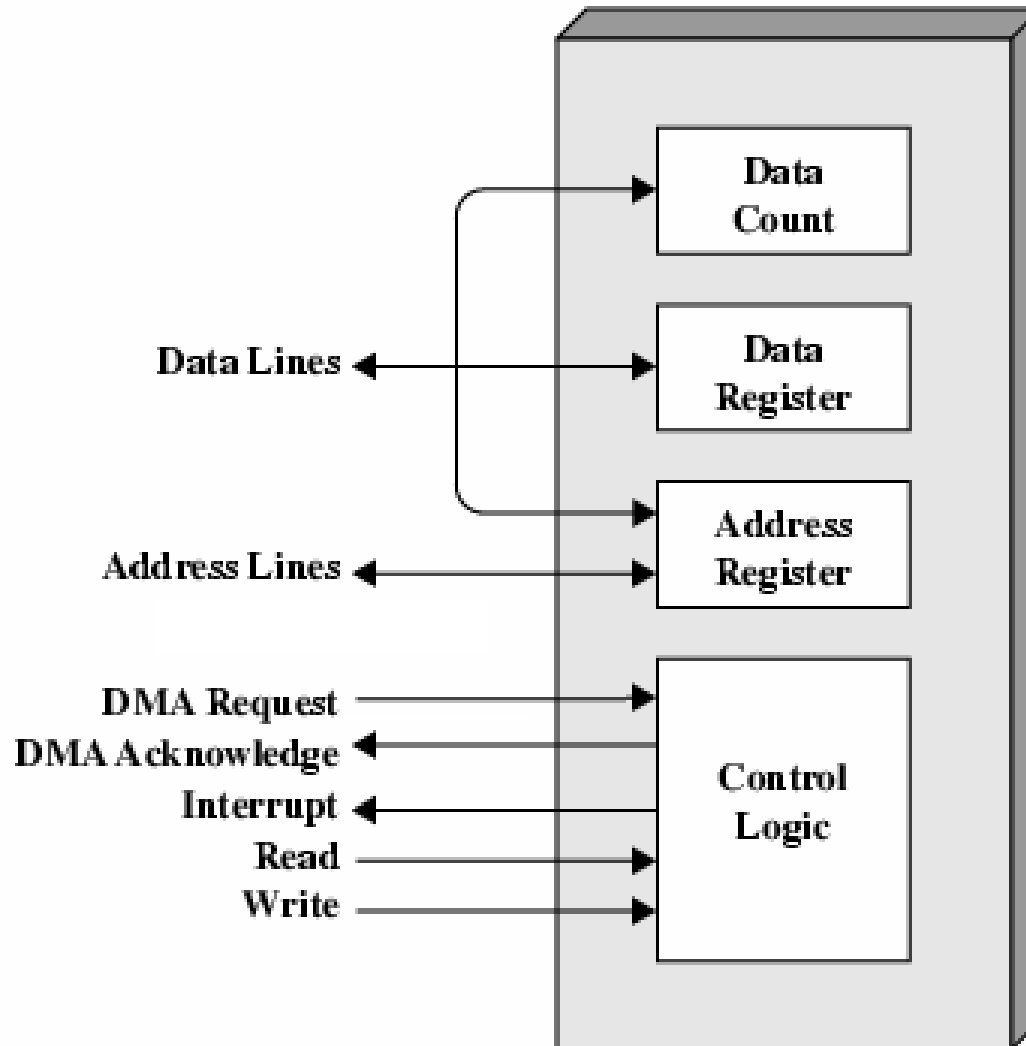
- Mengapa DMA diperlukan?
 - Karena *programmed I/O* dan *interrupt driven I/O*:
 - Masih memerlukan keterlibatan CPU → CPU menjadi sibuk
 - Transfer rate data terbatas
- DMA solusinya



(c) Direct memory access

- Digunakan modul khusus (*hardware*) yang terhubung ke sistem bus
- Fungsi modul DMA:
 - Dapat menirukan sebagian fungsi prosesor
 - Dapat mengambil alih fungsi prosesor yang berhubungan dengan transfer data
- Kapan DMA bekerja?
 - Saat prosesor sedang tidak menggunakan bus
 - Saat prosesor dipaksa berhenti sesaat (*suspend*) → siklusnya “dicuri” oleh DMA → disebut *cycle stealing*

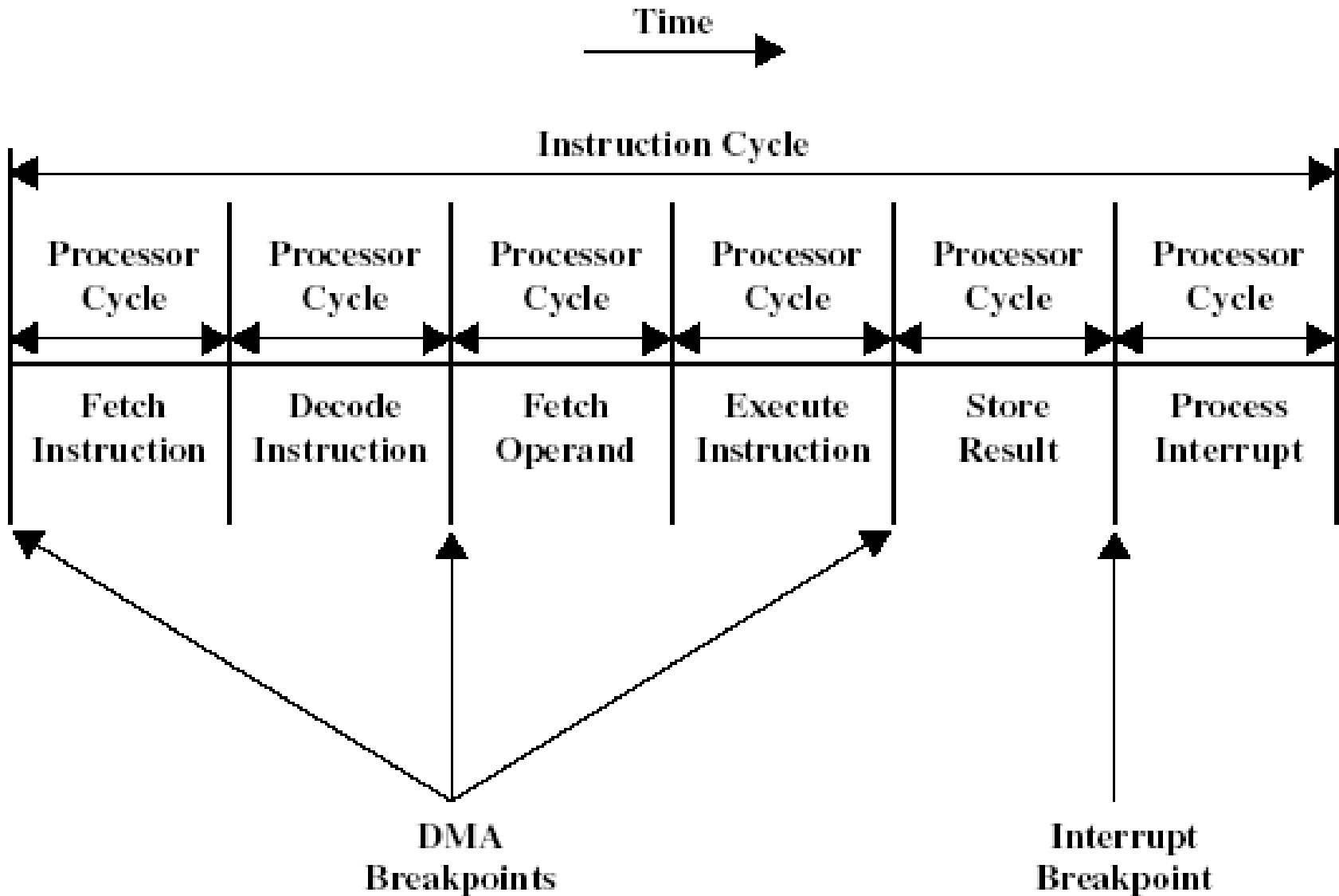
Diagram Modul DMA



- CPU mengirimkan data-data berikut ini ke DMA *controller*:
 - Perintah *Read/Write*
 - Alamat *device* yang akan diakses
 - Alamat awal blok memori yang akan dibaca/ditulis
 - Jumlah blok data yang akan ditransfer
- CPU mengeksekusi program lain
- DMA *controller* mengirimkan seluruh blok data (per satu *word*) langsung ke memori (tanpa melibatkan CPU)
- DMA *controller* mengirim *interrupt* ke CPU jika telah selesai

- DMA *controller* mengambil alih bus sebanyak satu siklus
- DMA men-transfer satu *word* data
- Pengambil alihan bus oleh DMA bukan *interrupt*
 - CPU tidak perlu menyimpan context
- CPU hanya tertunda (*suspend*) sesaat (satu siklus) sebelum mengakses bus
 - Yaitu sebelum *operand* atau data diambil atau data ditulis
- *Apa pengaruhnya terhadap CPU?*
 - Memperlambat CPU, tetapi masih lebih baik daripada CPU terlibat langsung pada transfer data

Cycle Stealing Pada DMA Transfer (2)



[STA19] Stalling, William. 2019. “*Computer Organization and Architecture: Designing for Performance*”. 11th edition. Prentice Hall.