# STRUKTUR SISTEM KOMPUTER

OLEH:
BAGUS SUDIRMAN., S.KOM M.KOM.

# STRUKTUR SISTEM KOMPUTER

- Operasi Sistem Komputer
- Struktur I/O
- Struktur Storage
- Proteksi Hardware

# OPERASI SISTEM KOMPUTER [1]

- *CPU devices* dan *I/O* dapat beroperasi secara serentak (*concurrent*)
  - Efisiensi pemakaian CPU
- Semua *request* ke *I/O* dikendalikan oleh *I/O systems*:
  - Setiap *device* terdapat *controller* yang mengendalikan *device* tertentu, misalkan *video display => video card*, *disk => disk controller*.
  - Setiap *device controller* mempunyai local buffer.
- CPU memindahkan data dari/ke *memory* ke/dari *local buffer*.
  - Setelah itu *controller* akan mengirimkan data dari *buffer* ke *device*.
- Bagaimana mekanisme *I/O* supaya *CPU* dapat melakukan *switch* dari satu *job* ke *job* lain?

# OPERASI SISTEM KOMPUTER [2]

- Ilustrasi:
  - Instruksi *CPU* dalam *orde*: beberapa mikro-detik
  - Operasi *read/write* dari *disk*: 10 – 15 mili-detik
  - Ratio: CPU ribuan kali lebih cepat dari operasi I/O
    - Jika *CPU* harus menunggu (*idle*) sampai data transfer selesai, maka utilisasi *CPU* sangat rendah (lebih kecil 1%).
- Solusi: operasi *CPU* dan *I/O* harus *overlap*
  - *Concurrent*: *CPU* dapat menjalankan beberapa *I/O device* sekaligus
  - *CPU* tidak menunggu sampai operasi *I/O* selesai tapi melanjutkan tugas yang lain
  - Bagaimana *CPU* mengetahui *I/O* telah selesai?

# PROGRAMMED INPUT/OUTPUT [1]

- Programmed I/O
  - Mekanisme CPU yang bertanggung jawab memindahkan data dari/ke memori ke/dari *controller*
- CPU bertanggung jawab untuk jenis operasi *I/O*
  - *Transfer* data dari/ke *buffer*
- *Controller* melakukan detil operasi *I/O*
  - Jika telah selesai memberikan informasi ke *CPU* => *flag*
- Bagaimana *CPU* mengetahui operasi telah selesai?
  - Apakah menguji *flag*? Seberapa sering?

# PROGRAMMED INPUT/OUTPUT [2]

- *CPU* harus mengetahui jika *I/O* telah selesai =>*hardware flag* (*controller*)
- *Polling*: *CPU* secara periodik menguji *flag* (*true or false*)
  - Menggunakan instruksi khusus untuk menguji *flag*
  - Masalah: seberapa sering? "*wasted CPU time* !"? Antar *I/O device* berbeda "*speed*"!
- Interrupt:
  - Bantuan hardware – melakukan interupsi pada CPU jika *flag* tersebut telah di-set (operasi *I/O* telah selesai)

# INTERRUPT

- Interrupt:
  - *CPU* transfer control ke "*interrupt service routine*", => address dari *service routine* yang diperlukan untuk *device* tsb.
  - ***Interrupt handler*: menentukan aksi/*service* yang diperlukan**
- Struktur *interrupt* harus menyimpan *address* dari instruksi yang sedang dikerjakan oleh *CPU* (*interrupted*).
  - *CPU* dapat *resume* ke lokasi tersebut jika *service routine* telah selesai dikerjakan
- Selama *CPU* melakukan *service interrupt*, maka *interrupt* selanjutnya tidak akan dilayani "*disabled*", karena *CPU* tidak dapat melayani *interrupt* (*lost*).
- Pengoperasian sistem tersebut menggunakan *interrupt driven*.

# INTERRUP HANDLING

- Hardware dapat membedakan devices mana yang melakukan interupsi.
  - Jenis interupsi :
    - *polling*
    - *vectored interrupt system*
- Tugas sistem operasi menyimpan status *CPU* (program *counter*, *register* dll)
  - Jika *service routine* telah selesai => *CPU* dapat melanjutkan instruksi terakhir yang dikerjakan
  - Sistem operasi akan "*load*" kembali status *CPU* tersebut.

# STRUKTUR INPUT/OUTPUT

- *User request I/O*:
  - *CPU*: *load* instruksi ke *register controller*
  - *Controller*: menjalankan instruksi
- Setelah *I/O* mulai, *control* kembali ke *user program* jika operasi *I/O* telah selesai
  - Instruksi khusus: *wait* => *CPU* menunggu sampai ada *interrupt* berikutnya dari *I/O* tersebut.
  - Paling banyak hanya mempunyai satu *I/O request*.
  - Keuntungan: *CPU* mengetahui secara pasti *device* mana yang melakukan *interrupt* (operasi *I/O* selesai).
  - Kerugian: operasi *I/O* tidak dapat serentak untuk semua *device*

# INPUT / OUTPUT INTERRUPT

- Pilihan lebih baik: *asynchronous I/O*
- Setelah *I/O* mulai, kendali langsung kembali ke *user program* tanpa menunggu *I/O* selesai
  - *CPU* dapat melanjutkan operasi *I/O* untuk *device* yang lain
  - *User* program dapat menjalankan program tanpa menunggu atau harus menunggu sampai *I/O* selesai.
  - *System call – request* ke OS untuk operasi *I/O* dan **menunggu** sampai *I/O* selesai.
- Potensi lebih dari satu device
  - *User* hanya dapat menggunakan *I/O* melalui *system call*
  - *Device-status table* memuat informasi untuk setiap *I/O device:* tipe, alamat, status dll
  - OS mengatur tabel ini dan mengubah isinya sesuai dengan status *device* (*interrupt*)

# DIRECT MEMORRY ACCESS (DMA)

- Jika I/O devices sangat cepat (“high-speed”),beban CPU menjadi besar harus mengawasi transfer data dari controller ke memory dan sebaliknya.
- Hardware tambahan => DMA controller dapat memindahkan blok data dari buffer langsung ke memory tanpa menggangu CPU.
  - CPU menentukan lokasi memory dan jika DMA controller telah selesai => interrupt ke CPU
  - Hanya satu interrupt ke CPU untuk sekumpulan data (blok).

# STRUKTUR STORAGE

- Main memory
  - Media penyimpanan, dimana CPU dapat melakukan akses secara langsung
- Secondary storage
  - Tambahan dari main memory yang memiliki kapasitas besar dan bersifat nonvolatile
- Magnetic disks
  - Metal keras atau piringan yang terbungkus material magnetik
  - Permukaan disk terbagi secara logikal dalam *track*, yang masing-masing terbagi lagi dalam *sector*
  - Disk controller menentukan interaksi logikal antara device dan komputer

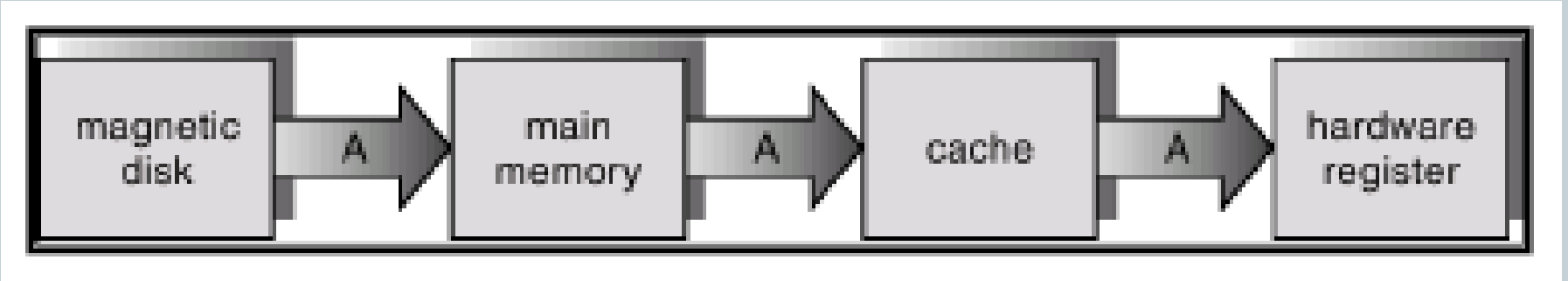
# HIRARKI STORAGE

- Hirarki sistem storage, diorganisasikan dalam bentuk :
  - Kecepatan
  - Biaya
  - Volatilitas
- *Caching*
  - Penduplikasian informasi ke dalam sistem storage yang cepat dapat dilakukan melalui cache pada secondary storage

# CACHING

- Menggunakan memori berkecepatan tinggi untuk menangani akses data saat itu juga (yang terbaru)
- Membutuhkan manajemen cache.
- Caching mengenalkan tingkatan lain dalam hirarki storage, dimana data secara serentak disimpan pada lebih dari satu tingkatan secara konsisten

# MIGRASI DARI DISK KE REGISTER



# TERIMAKASIH