IF3055 - Manajemen Memori

Henny Y. Zubir
STEI - ITB

Ikhtisar

- Binding
- Swapping
- Alokasi Kontigu
- Paging
- Segmentasi
- Dukungan Hardware

Latar Belakang

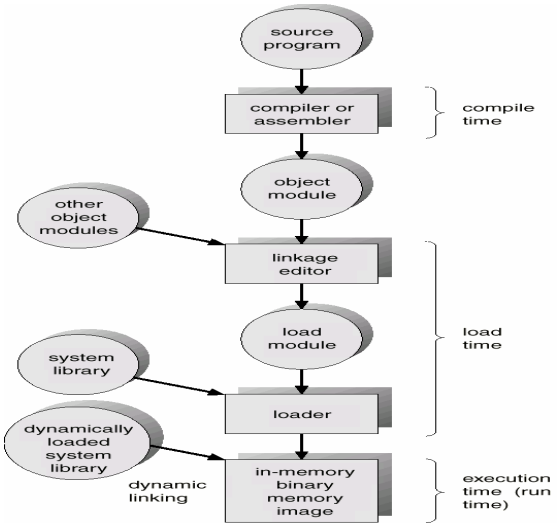
- Untuk dpt dieksekusi, program harus dibawa ke memori dan menjadi suatu proses
- *Input queue* – kumpulan proses pada disk yang menunggu dibawa ke memori untuk dieksekusi
- User program mengalami beberapa tahap sebelum dieksekusi
- Kebutuhan:
  - Meminimalkan waktu akses memori utama
  - Memaksimalkan ukuran memori utama
  - Hemat biaya

Binding Instruksi & Data ke Memori (1)

Binding alamat instruksi dan data ke alamat memori dapat terjadi dalam 3 tahap yg berbeda:

- **Compile time**: jika lokasi memori tlh diketahui sebelumnya, kode absolut dpt dibangkitkan; kompilasi ulang kode jika lokasi awal berubah
- **Load time**: harus membangkitkan kode *relocatable* jika lokasi memori tidak diketahui pada saat kompilasi
- **Execution time**: Binding ditunda hingga run time jika proses dapat dipindahkan dari satu segmen memori ke segmen lainnya selama eksekusi berlangsung; butuh dukungan H/W utk pemetaan alamat

Binding Instruksi & Data ke Memori (2)



Dynamic Loading

- Rutin tdk di-load hingga dipanggil
- Rutin yang tidak digunakan tidak di-load → penggunaan ruang memori lebih baik
- Bermanfaat jika banyak kode yang diperlukan untuk menangani kasus yg jarang terjadi
- Tidak ada dukungan khusus dari OS yg diperlukan; diimplementasikan melalui rancangan program

Dynamic Linking

- Linking ditunda hingga waktu eksekusi
- Sebagian kecil dari kode dan stub digunakan utk menempatkan rutin memory-resident library
- Stub menempatkan dirinya pd alamat rutin dan mengeksekusi rutin tsb
- OS diperlukan utk memeriksa jika rutin berada pd alamat memori proses

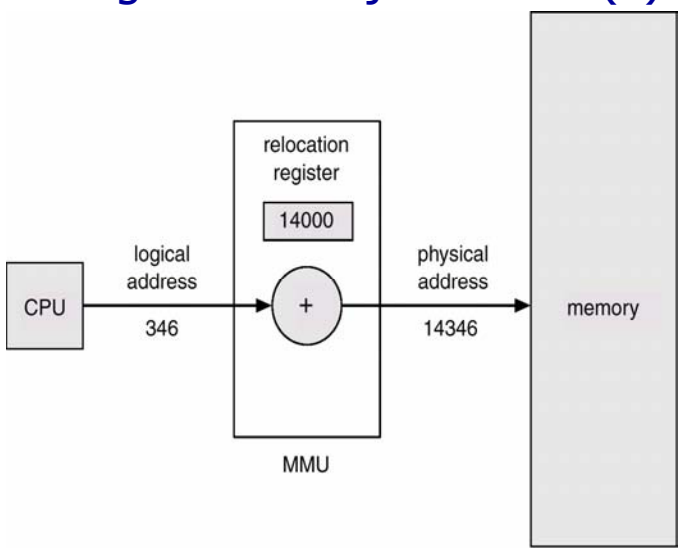
Overlays

- Hanya instruksi dan data yg diperlukan pd suatu waktu yang disimpan di memori
- Diperlukan jika ukuran proses lebih besar dari memori yg dialokasikan untuknya
- Diimplementasikan oleh user, tidak ada dukungan khusus dari OS
- Rancangan program struktur overlay sangat kompleks

Ruang Alamat Lojik vs Fisik (1)

- Konsep ruang alamat lojik yg di-bind ke ruang alamat fisik sangat penting dlm manajemen memori:
  - **Alamat lojik**: dibangkitkan oleh CPU; disebut juga **alamat virtual**
  - **Alamat fisik**: alamat yg diketahui oleh unit memori
- Alamat lojik dan alamat fisik sama pada skema binding compile-time maupun load-time;
- Alamat lojik (virtual) dan fisik berbeda pada skema binding execution-time

Ruang Alamat Lojik vs Fisik (2)



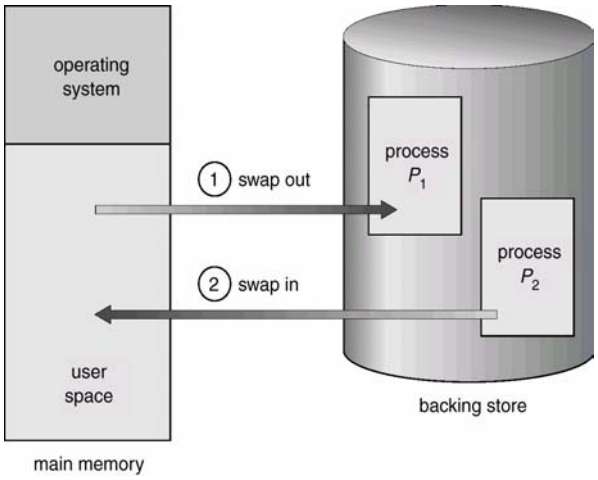
Memory Management Unit

- MMU: perangkat hardware yg memetakan alamat virtual ke alamat fisik
- Pada MMU, nilai di register relokasi ditambahkan ke setiap alamat yg dibangkitkan oleh proses user pd saat dikirim ke memori
- Program user hanya berurusan dgn alamat lojik, tidak pernah dgn alamat fisik yg sebenarnya

Swapping

- Proses bisa di-swap sementara dari memori ke backing store, dan nantinya dibawa kembali untuk meneruskan eksekusi
- Backing store – fast disk yg cukup besar utk meng-akomodasi salinan semua image memori utk semua user; harus menyediakan akses langsung ke memori image ini
- *Roll out, roll in* – varian swapping yg digunakan utk algoritma penjadwalan berbasis prioritas; proses dgn prioritas lebih rendah di-swap out sehingga proses prioritas lebih tinggi dpt di-load dan dieksekusi
- Kebanyakan waktu swap terpakai utk melakukan transfer; waktu transfer total terkait langsung dgn banyaknya memori yg di-swap
- Versi modifikasi dari swapping digunakan pd berbagai sistem, seperti UNIX dan Microsoft Windows

Skema Swapping

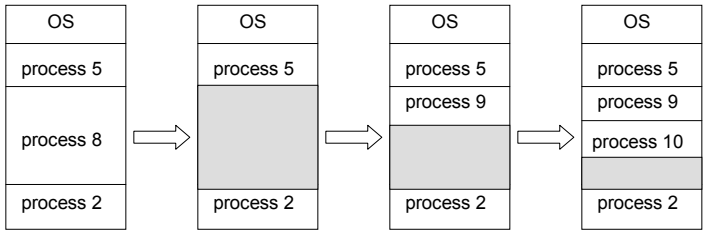


Alokasi Kontigu (1)

- Memori utama dibagi menjadi 2 partisi:
  - Resident OS, biasanya disimpan di memori rendah bersama dgn interrupt vector
  - Proses user disimpan di memori tinggi
- Alokasi partisi tunggal
  - Skema register relokasi digunakan utk memproteksi proses dari lainnya, dari mengubaha kode OS dan data
  - Register relokasi berisi nilai alamat fisik terkecil; register limit berisi rentang alamat lojik – tiap alamat lojik harus lebih kecil dari register limit

Alokasi Kontigu (2)

- Alokasi partisi-banyak:
  - Ruang kosong – blok memori yg tersedia; ruang kosong dgn berbagai ukuran tersebar pd memori
  - Proses baru akan dialokasikan memori pd ruang kosong yg cukup besar utk ditempatinya
  - OS mengelola informasi mengenai: (a) partisi yg dialokasikan, dan (b) partisi bebas (ruang kosong)



Masalah Alokasi Partisi Dinamis

- Bagaimana memenuhi permintaan memori dgn ukuran *n* dari list ruang kosong
  - **First-fit**: alokasikan ruang kosong pertama pada list yg muat
  - **Best-fit**: alokasikan ruang kosong terkecil yg muat; harus mencari keseluruhan list, kecuali jika list diurutkan sesuai ukuran Produces the smallest leftover hole.
  - **Worst-fit**: Allocate the *largest* hole; must also search entier list. Produces the largest leftover hole.

Fragmentasi

- **Fragmentasi eksternal** – total ruang memori tersedia utk memenuhi suatu permintaan, namun tidak kontigu
- **Fragmentasi internal** – memori yg dialokasikan sedikit lebih besar dari memori yg diminta; kelebihan bersifat internal, namun tidak digunakan
- Mengurangi fragmentasi eksternal melalui kompaksi
  - Menata ulang isi memori untuk menyatukan semua ruang kosong dalam satu blok yang besar
  - Kompaksi hanya mungkin jika relokasi bersifat dinamis dan dilakukan pada saat eksekusi
  - Masalah I/O
    - Kunci job di memori sementara terlibat dalam I/O
    - Lakukan I/O hanya ke buffer O/S

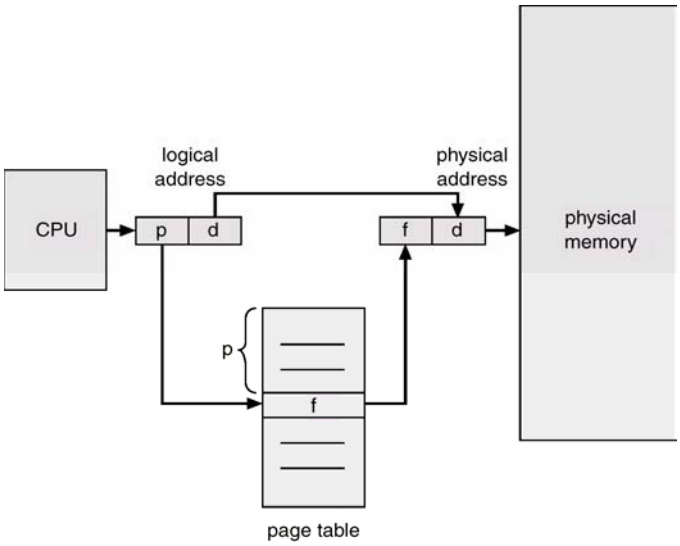
Paging

- Ruang alamat lojik dari proses dapat bersifat non-kontigu; proses dialokasikan memori fisik ketika memori tersedia
- Memori fisik dibagi menjadi blok-blok berukuran tetap yg disebut **frame** (berukuran kelipatan 2, antara 512 – 8192 bytes)
- Memori lojik dibagi menjadi blok-blok berukuran sama yg disebut **page**
- Mengelola semua frame kosong
- Untuk menjalankan program berukuran n page, harus dicari frame kosong sebanyak n untuk meload program
- Page table digunakan untuk translasikan alamat lojik ke alamat fisik
- Terjadi fragmentasi internal

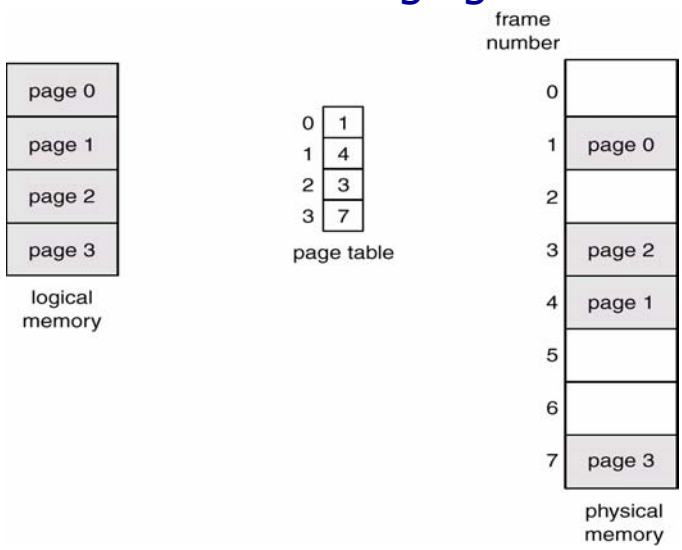
Skema Translasi Alamat (1)

- Alamat yang dibangkitkan oleh CPU dibagi menjadi:
  - **Page#** (p) – digunakan sbg index ke page table yg berisi alamat basis tiap page di memori fisik
  - **Page Offset** (d) – dikombinasikan dgn alamat basis utk mendefinisikan alamat memori fisik yg dikirim ke unit memori

Skema Translasi Alamat (2)



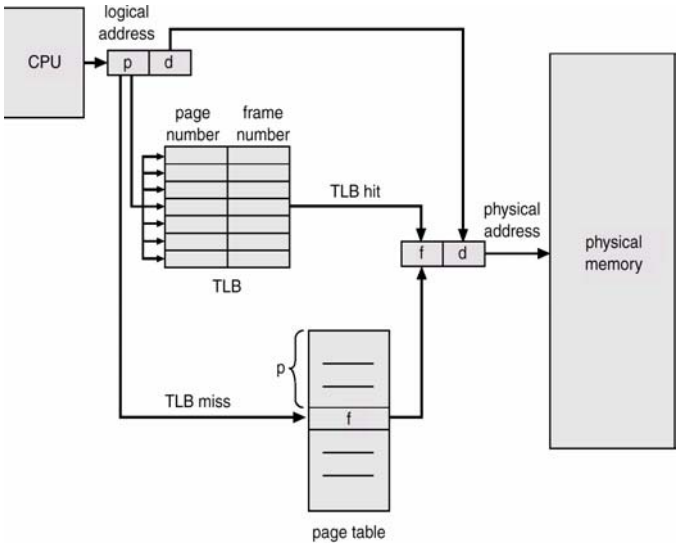
Contoh Paging



Implementasi Page Table (1)

- Page table disimpan di memori utama
- **Page-table base register** (PTBR) menunjuk ke page table
- **Page-table length register** (PRLR) berisi ukuran page table
- Setiap akses terhadap data/instruksi memerlukan 2x akses memori: (1) page table, dan (2) data/instruksi
- **Translation look-aside buffers** (TLBs) merupakan fast-lookup hardware cache khusus untuk mempercepat akses data/instruksi

Implementasi Page Table (2)



Effective Access Time

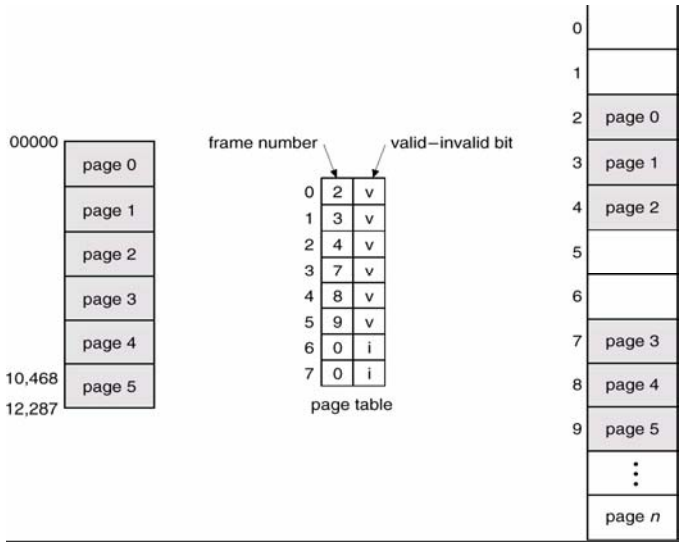
- Associative Lookup = ε unit waktu
- Asumsi waktu siklus memori 1 ms
- Hit ratio – % banyaknya page yg ditemukan di TLB; rasio terkait dgn banyaknya TLB
- Hit ratio = α
- Effective Access Time (EAT)

$$EAT = (1 + \varepsilon)\,\alpha + (2 + \varepsilon)(1 - \alpha)$$
$$= 2 + \varepsilon - \alpha$$

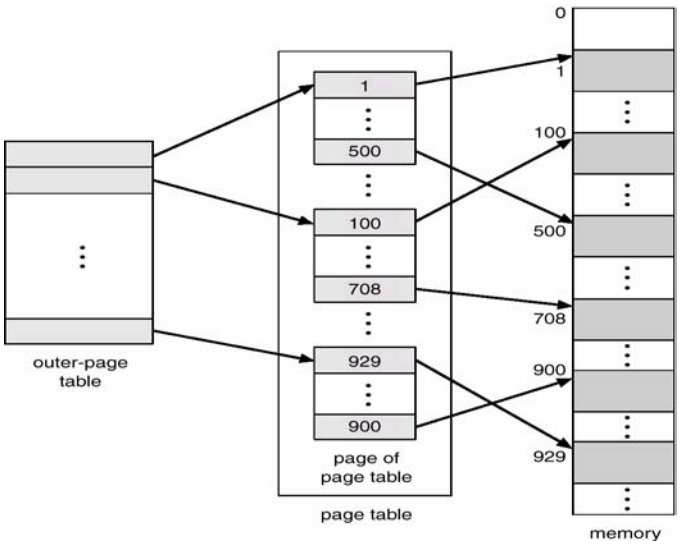
Proteksi Memori (1)

- Tiap frame memiliki **bit proteksi**
- Bit **valid-invalid** utk tiap entri pd page table:
  - "valid" : page terkait berada di ruang alamat lojik proses → legal page
  - "invalid" : page tidak berada di ruang alamat lojik proses

Proteksi Memori (2)



Page Table Dua Tingkat: Skema



Page Table Dua Tingkat: Contoh

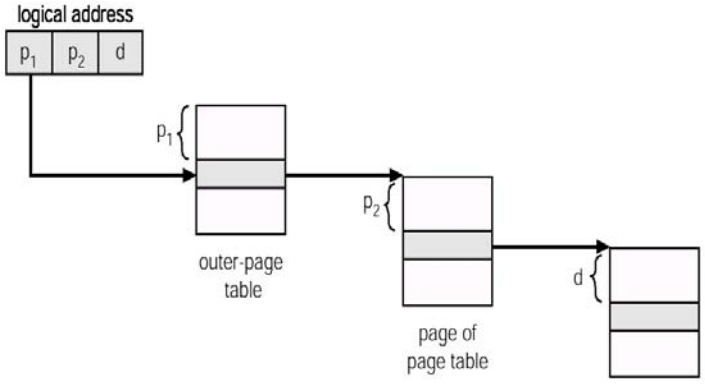
- Alamat lojik (pd mesin 32-bit dgn ukuran page 4K) dibagi menjadi:
  - Page# terdiri dari 20 bit
  - Page offset terdiri dari 12 bit
- Karena page table di-paged, page# dibagi lagi menjadi:
  - 10 bit page#
  - 10 bit page offset
- Dengan demikian, alamat lojik sbb:

page number		page offset
p_i	p_2	d
10	10	12

dimana p_i adalah index ke page table luar, dan p_2 adalah displacement pada page di page table luar

Skema Translasi Alamat

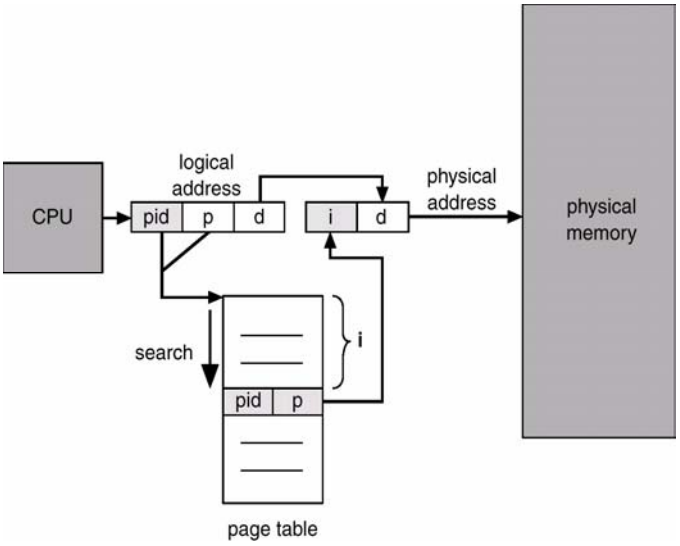
- Skema translasi alamat untuk arsitektur paging dua level 32-bit



Inverted Page Table

- Tiap entri untuk tiap page riil di memori
- Entri terdiri dari alamat virtual dari page yg disimpan lokasi riil memori tsb, dgn info mengenai proses yg memiliki page tsb
- Mengurangi memori yg dibutuhkan utk menyimpan page table, namun meningkatkan waktu yg diperlukan utk mencari di tabel jika page dirujuk
- Menggunakan hash table utk membatasi pencarian ke satu – atau, paling banyak beberapa – entri page table

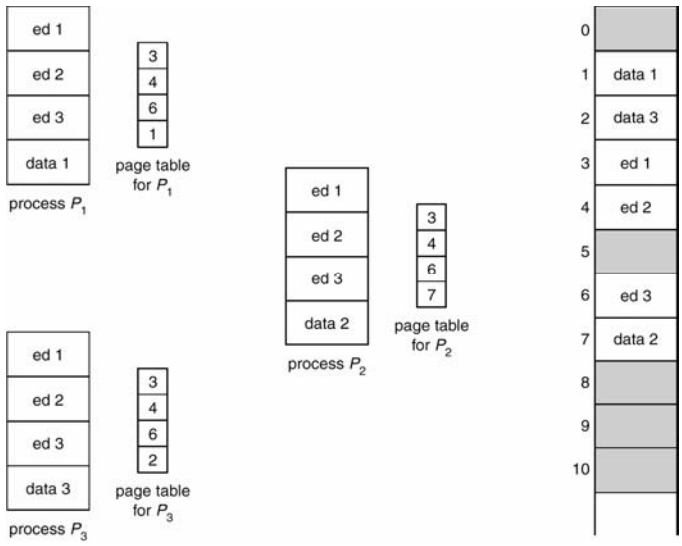
Inverted Page Table: Arsitektur



Shared Pages

- Shared code
  - Satu salinan kode read-only (reentrant) yg digunakan bersama oleh proses (i.e., text editors, compilers, sistem window)
  - Shared code harus berada pd lokasi yg sama di alamat lojik semua proses
- Kode privat dan data
  - Tiap proses menyimpan salinan terpisah untuk kode dan data
  - Page utk kode dan data privat bisa berada dimana saja di ruang alamat lojik

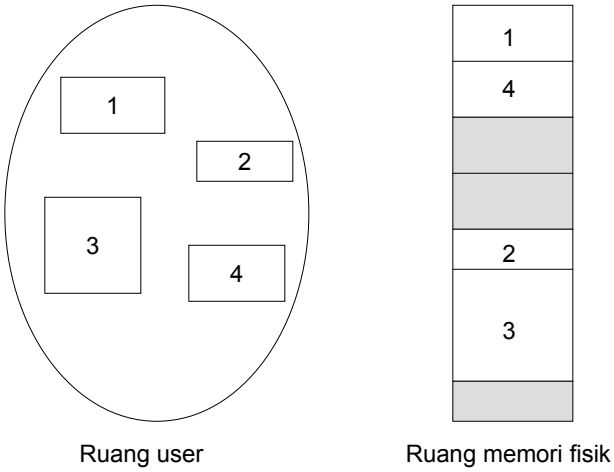
Shared Pages: Contoh



Segmentasi

- Skema memory-management yg mendukung user view dari memori
- Program merupakan kumpulan segment.
- Segment merupakan unit lojik seperti:
  - program utama,
  - prosedur,
  - fungsi,
  - variabel lokal, variabel global,
  - block bersama0,
  - stack,
  - tabel simbol, arrays

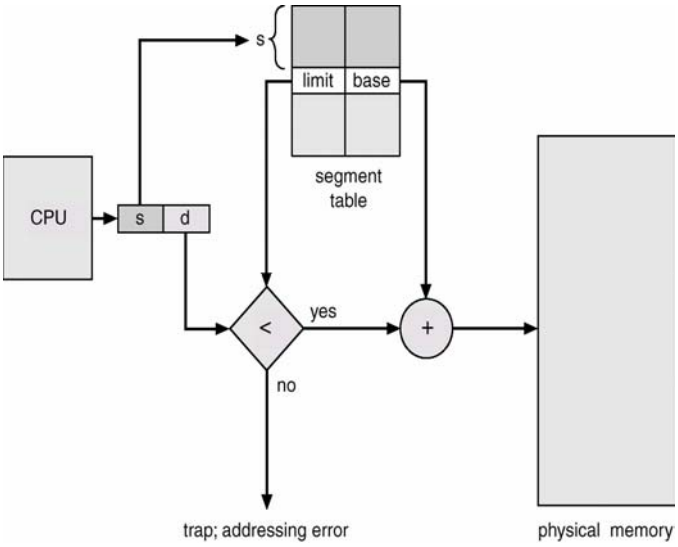
Segmentasi: View Lojik



Segmentasi: Arsitektur (1)

- Alamat lojik: <segment-number, offset>,
- **Segment table** – memetakan alamat 2-dimensi; tiap entri tabel memiliki:
  - **Basis** – berisi alamat fisik awal dimana segment berada di memori
  - **limit** – berisi ukuran segment
- **Segment-table base register** (STBR) menunjuk ke lokasi segment table di memori
- **Segment-table length register** (STLR) berisi banyaknya segment yg digunakan suatu program
  - Segment# *s* sah jika *s* < STLR

Segmentasi: Arsitektur (2)



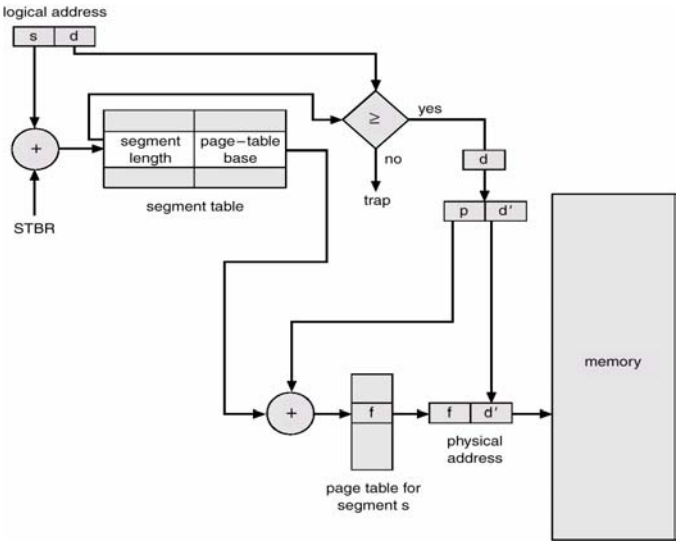
Segmentasi: Arsitektur (3)

- Relocation
  - Dinamis
  - Oleh segment table
- Sharing
  - shared segments
  - Segment# yg sama
- Allocation
  - first fit/best fit
  - Fragmentasi eksternal

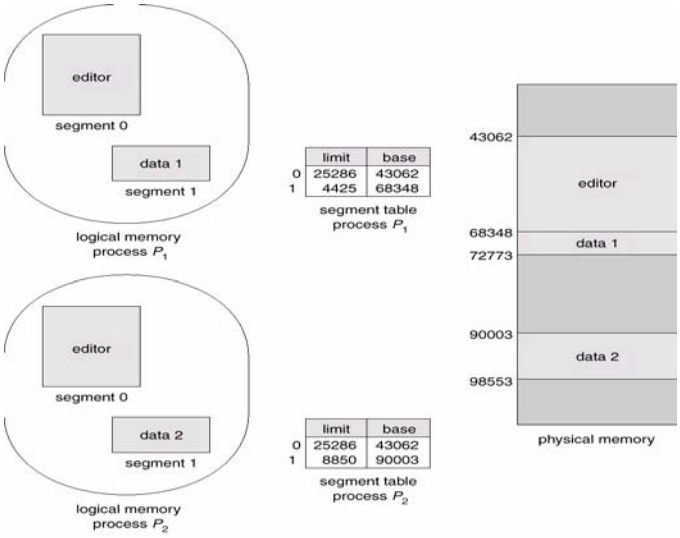
Segmentasi: Arsitektur (4)

- Tiap entri pd segment table memiliki:
  - bit validasi = 0 $\Rightarrow$ segment ilegal
  - Hak read/write/execute
- Bit proteksi diasosiasikan dgn segment; sharing kode terjadi pd level segment
- Karena ukuran segment bervariasi, alokasi memori memiliki masalah alokasi penyimpanan dinamis

Segmentasi: Arsitektur (5)

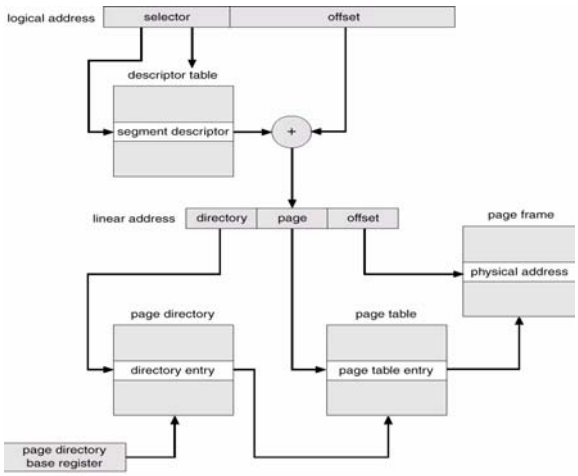


Segmentasi: Contoh



Dukungan Hardware: Intel 386

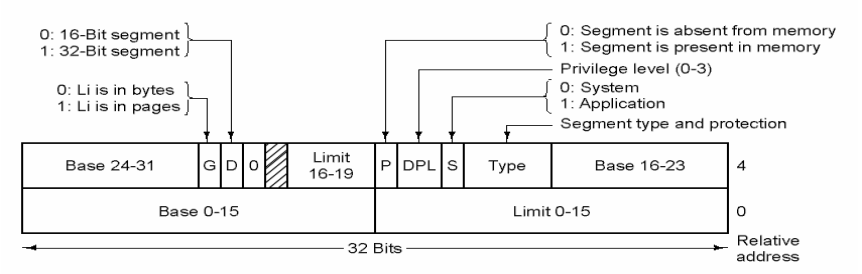
- Menggunakan segmentasi dgn paging utk manajemen memori dgn skema paging 2 tingkat



Dukungan Hardware: Pentium (1)

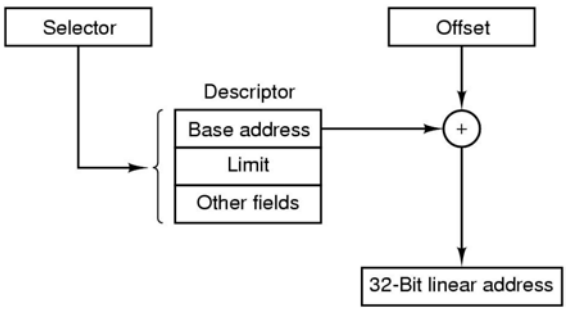
- Segmentasi dgn Paging
  - Segment selector

  - Code segment descriptor
    (data segment sedikit berbeda)



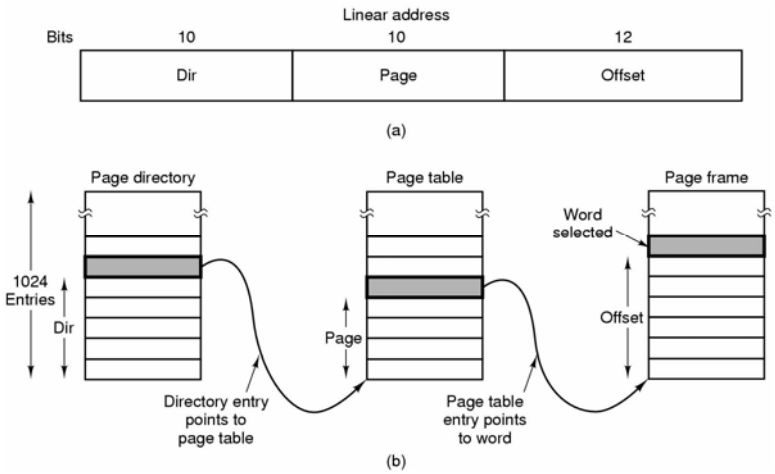
Dukungan Hardware: Pentium (2)

- Konversi dari (selector, offset) ke alamat linier



Dukungan Hardware: Pentium (3)

- Pemetaan alamat linier ke alamat fisik



Dukungan Hardware: Pentium (4)

- Proteksi