PENGANTAR

I.1. Apa itu Pemrograman Sistem ?

Sistem Software adalah kumpulan *system program* yang menyediakan beragam fungsi seperti file editing, resource accounting, IO management, storage management dsb. Lalu apa yang dimaksud dengan system program ?

> ***System program*** *adalah suatu program yang membantu* ***general user*** *menjalankan atau* ***mengeksekusi komputasi secara efektif*** *yang dibutuhkan oleh system komputer*.

"General user" yang dimaksud pada definisi di atas, tidak termasuk program khusus yang digunakan oleh user, seperti software application. Sedangkan eksekusi disini meliputi semua aktivitas diawali dari input teks program dan beragam tahapan pemrosesan dalam system komputer seperti : penamaan, editing, storage, relocation, linking dan pada akhirnya eksekusi itu sendiri.

I.1.1. Sistem Program dan Pemrograman Sistem

Pemrograman Sistem adalah aktivitas perancangan dan pengimplementasian Sistem Program. Pertanyaan yang sering muncul adalah apakah perbedaan antara Pemrograman Sistem dan aktivitas pemrogaraman yang lain ? Pertanyaan lain yang relevan adalah apakah perbedaan Sisetm Program dan jenis program yang lain, seperti scientific program, data processing program atau application program.

Sistem program membantu general user menjalankan suatu eksekusi dengn effektif pada system komputer. Apa yang dimaksud dengan "efektif" disini ? Apakah berkaitan dengan computer time atau programmer time ? Pada dasarnya yang dimaksud "efektif" dalam hal ini adalah keseluruhan proses pengembangan program dan eksekusi. Dengan kata lain "efektif " adalah keseimbangan antara aspek (i) keefektifan penggunaan system komputer, (ii) keefektifan dari sumber daya manusianya yang terlibat dalam pengembangan program. Secara keseluruhan efektifitas terebut akan terkait dengan optimalisasi pembiayaan.

Keseimbangan pembiayaan untuk komputer dan sumberdaya manusia tergantung dari situasi yang mempengaruhinya, yaitu (i) pengaruh factor lingkungan yang ada pada system komputer seperti main storage, auxiliary devices, dsb, (ii) pengaruh komputasi seperti pengembangan program, data processing, real time application, dsb. Karena tujuan utama perancang system program adalah merancang dan mengkode program agar tugas yang dijalankannya tidak hanya berjalan dengan benar tetapi juga effektif, maka berkaitan dengan keseimbangan biaya seperti yang telah dijelaskan di atas, efisiensi algoritma dan keserasian struktur data menjadi hal yang penting.

Sebagai contoh, compiler yang digunakan untuk bahasa pemrograman FORTRAN, yang banyak digunakan pada komputasi scientific. Apabila perancang compiler-dalam hal ini compiler merupakan system program- menekankan pada optimalisasi sumberdaya komputer, misalnya CPU, untuk menjalankan pekerjaannya,maka perancang tersebut akan berharap bahwa compiler tersebut (i) dapat menjalankan proses kompilasi FORTRAN secara cepat, atau (ii) menjalankan program secara efisien (*efficient execution*) setelah program tersebut ditranslasikan ke dalam bentuk bahasa mesin. Jika semua pekerjaan dalam instalasi berjalan hanya satu atau dua kali dan eksekusinya tidak berlangsung dalam waktu yang lama, maka kompilasi yang cepat menjadi pilihan yang lebih baik dibandingkan dengan *efficient execution*. Di lain sisi, jika pekerjaan cenderung dieksekusi oleh komputer dalam waktu bermenit-menit hingga berjam-jam, *efficient execution* menjadi prioritas utama. Karenanya compiler dirancang tidak menekankan pada optimalisasi manusia dan sumberdaya komputer yang dicurahkan pada eksekusi pekerjaan. Sebagai contoh, compiler mempunyai pekerjaan untuk mengindikasikan semua kesalahan yang ada pada suatu pekerjaan. Sebagai konsekuensinya, user akan menghabiskan jumlah waktu untuk mencoba menemukan *bugs* dalam program.

Permasalahannya penelusuran *bugs* mempertimbangkan berbagai kemungkinan dan membutuhkan ekstra pengerjaan program dan penggunaan sumber daya komputer secara baik.

Suatu komputer dirancang untuk memberikan nilai tambah bagi sumberdaya manusia yang dapat melakukan pengecekan secara mendalam terhadap program untuk mendeteksi semua kemungkinan kesalahan (*error*) yang mungkin terjadi. Karenanya dukungan terhadap pengindikasian kesalahan (*diagnostic support*) akan memperlambat compiler karena membutuhkan banyak waktu untuk memproses setiap statement. Dalam situasi seperti ini, *programmer time* menjadi lebih mahal dibandingkan dengan *computer time,* karenanya compiler menjadi lambat namun diimbangi dengan kemampuan diagnostik yang baik sehingga secara keseluruhan menjadi lebih efektif dibandingkan dengan *fast compiler* tanpa diagnostik yang memadai.

Suatu aspek yang membedakan Sistem Program dengan berbagai jenis program lain adalah adanya kaitan yang penting dengan factor lingkungannya, termasuk keseimbangan biaya antara sumberdaya manusia dan komputer. Hal yang perlu digarisbawahi adalah keseimbangan biaya tersebut menghasilkan suatu fakta penggunaan system komputer terkait erat dengan waktu.

I.2. Komponen Sistem Software

Seorang programmer memecahkan permasalahannya melalui system komputer. Untuk memecahkan permasalahnnya programmer dapat menggunakan berbagai bahasa pemrograman (*programming language*) dalam berhubungan dengan system komputer. Seperti kita ketahui CPU membutuhkan informasi yang sifatnya spesifik dan disajikan dalam format baku, dimana CPU hanya mengerti bahasa mesin (*machine language*). Karena itu programmer membutuhkan bahasa penterjemah (*language translation*) atau compiler yang akan menterjemahkan kumpulan instruksi dalam *programming language* ke dalam *machine languange*. Agar diperoleh intisari pekerjaan yang dihasilkan dari CPU, *machine languange* menyerahkannya pada *operating system* untuk menjadual pekerjaan yang dilakukan CPU dari waktu ke waktu. Hal ini membuat system komputer berjalan optimal.

Language translation dan *operating system* dapat dikategorikan sebagai *system program*.

Dengan menulis program menggunakan bahasa tingkat tinggi (*high level language*), seorang programmer memperoleh kebutuhannya tanpa perlu mengetahui lebih mendalam bagaimana program tersebut dimengerti oleh CPU. Compiler-lah yang akan bertugas agar program yang dibuat oleh programmer dimengerti oleh CPU. Agar kegiatan tadi berjalan secara optimal dalam system komputer maka *operating system* yang bertanggung jawab melakukannya. Dua aspek mendasar dari tugas *system software* adalah (i) membuat fasilitas yang ada menjadi lebih baik (ii) mencapai pekerjaan yang efisien.

1.3. Evoluai Sistem Software

Seperti telah dijelaskan di atas, system program yang merupakan komponen standar dalam system komputer yang dibangun secara bertahap. Dalam pembangunan system program tersebut aspek sumberdaya manusia dan sumberdaya komputer saling berkaitan satu sama lain agar dicapai efektivitas komputasi yang optimal. Pada masa sekarang biaya yang dikeluarkan untuk sumberdaya manusia lebih mahal dibandingkan dengan biaya komputer. Hal ini sangat berbeda dengan keadaan sekitar 30 tahun yang lalu dimana biaya untuk komputer lebih mahal dibandingkan dengan sumberdaya manusia. Oleh karena itu sejarah dari system software dapat dilihat dengan prioritas pada dua aspek yaitu pengenalan fasilitas yang lebih baik dan efektivitas penggunaan system.

I.3.1. Languange Translator

Tahapan awal yang cukup penting dalam sejarah system software adalah pengembangan bahasa penterjemah (*language translator*). Pada awalnya program ditulis dalam bahasa mesin

(*machine languange*). Hal ini sangat tidak praktis bila dipandang dari sisi programmer. Pengembangan languange translator membantu programmer untuk membuat kode program dalam bahasa yang mudah dimengerti oleh mereka untuk kemudian diubah ke dalam bahasa mesin. Translator ke dalam bahasa pemrogram tingkat rendah (*low level languange*) dikenal dengan istilah *assembler* atau bahasa assembly (*assembly languange*). *Assembly languange* merupakan bahasa yang mendekati bahasa mesin akan tetapi masih lebih mudah dipelajari oleh manusia. Bahasa ini menggunakan kode operasi mnemonik (*mnemonic operaration code*) seperti LOAD, ADD dan *symbolic operand* seperti VALUE, RESULT dalam merepresentasikan numeric dari kode instruksi mesin dan pengalamatanya. *Assembly languange* lebih mudah untuk menulis atau memodifikasi program, namun tetap mempunyai ketergantungan yang cukup tinggi pada mesin.

Perkembangan selanjutnya adalah bahasa pemrograman yang berdasar pada *machine independent*. Bahasa ini dikenal dengan istilah bahasa tingkat tinggi (*high level languange*) yang dibutuhkan programmer untuk merinci logic dalam penyelesaian masalah dalam bentuk algoritma, yaitu tahapan prosedur untuk mencapai solusi suatu masalah. Setiap tahapan prosedur direpresentasikan dalam program logic yang signifikan, seperti komputasi, *decision*, input nilai dan sebagainya serta tidak tergantung pada komputer dimana program tersebut dieksekusi. Digunakannya *high level languange* membebaskan programmer untuk mengetahui kerumitan detail pekerjaan dari komputer. H*igh level languange* akan ditranslasikan ke dalam bahasa mesin sebelum akhirnya dieksekusi. Kegiatan translasi ini lebih mahal bila dibandingkan translasi dari *assembly languange program*, karena adanya tambahan biaya dalam hal mereduksi rancangan, coding dan debugging program. Untuk memudahkan perancangan program itu sendiri saat ini banyak digunakan konsep struktur data dan penggunaan prosedur. *High level languange translator* membantu programmer mencari indikasi kesalahan selama proses proses translasi dijalankan.

I.3.2. Batch Monitor

Pada awal system komputer digunakan, sebuah program dijalankan pada satu waktu mode operasi (*one program at a time operating mode*). Operator komputer akan melakukan sedikit tindakan untuk men-set up dan menandai proses suatu pekerjaan. Tindakan tersebut sangat sederhana, hanya memutar/switches console, tetapi sering dalam pemberian instruksi dalam memori komputer, yang pada saat dieksekusi akan memulai pengoperasian translator. Setelah inisialisasi ini proses suatu pekerjaan akan dimulai. Pada akhir eksekusi pekerjaan, operator akan mengulangi tahapan yang sama untuk menginisialisasi proses pekerjaan akhir.

Mode operasi ini tidak efisien digunakan bagi komputer, karena banyak waktu yang terbuang dari kegiatan yang dilakukan oleh operator. Ketergantungan interaksi manusia dalam kegiatan ini-pun harus dikurangi. Dalam perancangan Sistem Program kemudian dikenal *Batch Monitor* yang merealisasikan proses dari sekumpulan pekerjaan user tanpa membutuhkan interaksi operator. *Batch monitor* mengambil alih kontrol operasi komputer, dimana dia akan menginisialisasi proses pada setiap pekerjaan secara *batch* dengan suatu cara yang pada akhirnya diproses, kendali selanjutnya akan dikembalikan ke *batch monitor*. Pada akhir pekerjaan dalam *batch* diproses, batch monitor akan dihentikan operasinya dan kendali akan dikembalikan ke operator komputer untuk inisialisasi tindakan berikutnya.

Otomatisasi kendali eksekusi pekerjaan *batch* dengan *batch monitor* meningkatkan efisiensi penggunaan komputer system. Jika system komputer dibagi (share) dalam suatu kelompok user akan lebih efisien dan diharapkan semua user akan memperoleh manfaatnya. Namun, ketika efisiensi coba untuk ditingkatkan, user secara umum akan mengalami *turn along time* yang cukup lama. T*urn along time* didefinisikan sebagai waktu yang terlewatkan sejak pengiriman pada pusat komputer hingga waktu pada saat hasil diperoleh. Pada "*one program at a time environment*" pengiriman pekerjaan dan pelepasan hasil umumnya dilakukan dalam basis informal, sering kali *turn around time* untuk suatu pekerjaan hanya berbeda tipis dengan waktu proses pekerjaan itu sendiri. Dengan dikenalkannya *batch processing*, prosedur formal mengharuskan pengenalan format fasilitas secara batch sejumlah pekerjaan.Untuk menghasilkan efisiensi yang lebih tinggi, pekerjaan dalam disimpan dalam media *input output* (IO) seperti *magnetic tape* atau *disk*.

Turn around time pada *batch processing* tergantung pada : (i) total waktu proses dari seluruh pekerjaan dalam batch (ii) waktu *batch formation* yang baik pada saat pencetakan output dan release time.

1.3.3. Multiprogramming Operating System

Pada arsitektur komputer klasik, instruksi input output dieksekusi bersamaan dengan instruksi lainnya (arithmetic, logical, dsb) oleh CPU. Ketika instruksi IO diterjemahkan, CPU akan membangkitkan signal kendali kepada *IO device*. Sekarang IO device sibuk dengan operasinya dan di akhir akan mengirimkan signal akhir operasi ke CPU. CPU akan mengalami waktu sia-sia ketika inisialisai IO hingga IO selesai. Konsep saluran (channel concept) akan membebaskan CPU dari waktu sia-sia yang tidak perlu ketika operasi IO sedang berjalan. IO dijalankan sebagai berikut : CPU mengeksekusi instruksi Start Input Output dengan alamat IO device sebagai operand. Pada saat instruksi dijalankan, alamat device akan dilewatkan melalui saluran/channel. Channel memeriksa device untuk melihat apakah device tersebut tersedia dan mengirimkan sebuah signal akhir operasi dengan suatu kode kondisi (ketersediaan device untuk beroprasi, device sibuk, *device non-exsistent*, dsb) ke CPU.

Penggunaan CPU dan IO channel secara bersamaan membutuhkan *data independence*, dimana 2 atau lebih *independence program* di panggil ke dalam memori. Ketika IO menjalankan sebuah program, CPU menjalankan komputasi untuk program yang lain. Dua atau lebih program yang dapat dijalankan secara bersamaan dalam sebuah interleave antara CPU dan IO subsystem disebut *multiprogramming*.

Suatu *multiprogramming operating system* adalah kumpulan system program termasuk *multiprogramming supervisor* (*control program*) dan beberapa program lain yang membutuhkan supervisor dari waktu ke waktu. *Multiprogramming* cukup efisien, namun masih mempunyai *turn around time* yang cukup lama.

1.3.4. Time Sharing Operating System

Dari sisi pandangan user, *turn around time* yang singkat sangat diharapkan, terutama kecepatan pendeteksian kesalahan program dan pengiriman ulang program untuk menjalankan test. Kebutuhan akan *turn around time* yang singkat dapat dipecahkan dengan konsep dari *interactive computing*.

Dalam *interactive computing*, user duduk di suatu terminal dan memasukan input di komputer dengan memasukan sedikit karakter atau suatu statement. Apa yang diketikkannya tersebut akan ditampilkan pada display yang kemudian ditransmisikan ke komputer dan komputer akan meresponnya. Dengan system penterjemah yang terinstall pada komputer, user harus memasukkan programnya sebelum dilakukan proses translasi atau dalam kasus ini komunikasi dengan translator dilakukan statement demi statement. Pada kasus lani translator akan memproses statement segera setelah dikirimkan dan melakukan pendeteksian kesalahan. Pemasukan input sesuai permintaan proses diikuti dengan penyelesaian proses oleh system komputer disebut dengan *interaction*. Waktu yang dibutuhkan komputer untuk merespon proses disebut dengan *response time*.

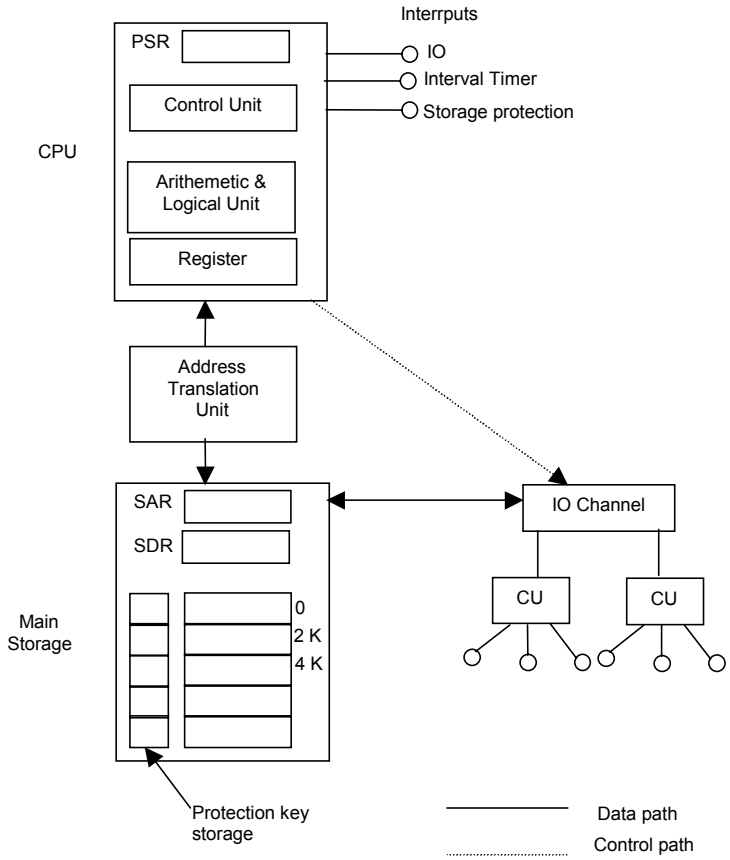
Dalam lingkungan *interactive*, user mengevaluasi *respon time* yang dikerjakan system. Salah satu cara untuk melihat *respon time* tersebut adalah dengan menghubungkan sejumlah terminal pada system dan menjalankan permintaan proses dari user dengan semua cara yang mungkin untuk mendapatkan *respon time* yang cukup baik. Sistem operasi yang demikian dikenal dengan *Time Sharing Operating Systems*. Penjadualan *Round Robin* adalah contoh dari penggunaan teknik *time sharing.*

I.4. Model Komputer Sistem

Pada bagian ini akan dibahas mengenai model system komputer yang menunjukkan cirri-ciri arsitektur yang ditemui pada kebanyakan system komputer. Secara umum model tersebut dibagi ke dalam dua tingkatan, yaitu (i) model mesin, yang menunjukkan cirri dari perangkat keras, dan (ii) model system operasi yang menggambarkan cirri arsitektur system operasi yang digunakan.

I.4.1. Mesin (Perangkat Keras)

Gambar I.1. mengilustrasikan organisasi dasar dari suatu mesin komputer dengan berbagai komponen dan keterhubungannya satu sama lain. Komponen utama dari model ini adalah storage unit, CPU dan subsistem IO yang meliputi IO processor atau IO channels dan IO device, dll.



Gambar I.1. Model mesin pada system komputer

A. Storage Unit

Main storage dari kumputer terdiri dari sekumpulan register yang masing-masing memilki alamat yang unik. Unit dasar pengaksesan biasanya disebut dengan *word*, dimana setiap unit penyimpananannya disebut dengan *byte*. Setiap byte terdiri dari 8 *bits* yang direpresentasikan dengan digit 0 atau 1. Sebuah byte disimpan dalam karakter alphanumeric (ASCII, EBCDIC, BCD, dll). Gabungan byte dikelompokkan ke dalam unit yang lebih besar seperti half words, full words, dsb.

Storage unit terdiri dari dua register yang penting yaitu :**Storage Address Register** (SAR) dan **Storage Data Register** (SDR). Komunikasi storage unit dengan lingkungan luar hanya dilakukan melalui register. Untuk membaca data pada suatu lokasi, dibutuhkan SAR, begitupula pada saat akan menulis, pengalamatannya membutuhkan SAR. Sedangkan data itu sendiri disimpan dalam SDR.

CPU dan IO Channel dihubungkan secara langsung melaui jalur data (data path) dengan storage unit. **Address Translation Unit** (ATU) ditempatkan pada CPU dengan storage path yang diimplementasikan sebagai *virtual storage*.

Mekanisme *Virtual storage* dikatakan tersedia jika alamat yang ditentukan oleh CPU untuk tujuan pengaksesan ternyata tidak sama dengan alamat sebenarnya yang ada pada storage unit yang diperuntukan untuk mengimplementasikan pengakasesan. Jika *virtual storage* mengimplementasikannya melalui *paging*, maka address translation unit akan mengenalnya sebagai *paging hardware* pada mesin.

Ketika komputer system dioperasikan secara multiprogramming, instruksi dan data dari program akan berada pada storage secara bersamaan. Dalam lingkungan demikian, jika alamat yang dihasilkan oleh program berada di luar jangkauan storage area, maka akan menyebabkan kerusakan pada instruksi atau data pada program yang lain. Untuk menghindari hal tersebut terjadi, skema *storage protection* digunakan untuk menghalangi sebuah program dari pengakasesan lokasi *storage* yang tidak dialokasikan untuknya.

B. Central Processing Unit (CPU)

CPU memili dua unit fungsi utama, yaitu **Control Unit** (CU) dan **Arithmetic Logical Unit** (ALU). Sekumpulan register pengendali yang dikenal dengan **Program Status Word** (PSW) atau **Program Status Register** (PSR) berisi semua informasi yang berkaitan dengan status CPU setiap waktunya.

Salah satu bagian dalam PSR adalah *Instruction Address Register* (IAR) yang berisi alamat berikutnya yang akan dieksekusi oleh CPU. Melalui alamat inilah *Control unit* pertama kali akan mendapatkan instruksi untuk dieksekusi. Hal ini dikenal dengan istilah *instruction fetch*. Instruksi ini kemudian akan disandikan dan operand akan diambil dari *main storage*. Instruksi ini kemudian akan melalui ALU untuk dieksekusi secara tepat. Suatu instruksi bisa jadi merupakan sekumpulan *condition code* dalam PSR yang mengindikasikan suatu hasil dari eksekusi, seperti hasil yang diperoleh dari perhitungan pembagian dengan nol, atau kondisi abnormal seperti kejadian *overflow*. Bagian dari PSR yang menyimpan condition code tersebut disebut *Condition Code Register* (CCR). ALU berisi sekumpulan register yang popular dengan sebutan *General Purpose Register* (GPR). Register ini mencakup pengalamatan serta operasi aritmetik dan logika.

Jenis dan format instruksi dari kebanyakan system, secara umum terbagi ke dalam tiga kelompok, yaitu :

(i) *Register to Register* (RR). Dalam instruksi in,i operasi dijalankan pada dua buah register, dan hasilnya akan disimpan pada salah satu register tersebut.

(ii) *Register to Storage* (RS). Dalam instruksi ini, suatu operasi yang melibatkan sebuah register operand dan sebuah operand yang berada di *main storage*. Hasil yang diperoleh akan ditinggalkan di dalam register, kecuali pada kasus operasi penyimpanan ketika hasil tersebut juga ditulis di lokasi penyimpanan khusus. Operasi itu menghasilkan *fixed /floating point arithmetic* dan *logical operation*.

(iii) *Storage to Storage (SS)*. Dalam Instruksi SS kedua operand berada di *main store*. Hasilnya akan ditinggalkan pada salah satu *storage* tersebut. Operasi ini umumnya menggunakan karakter dengan panjang yang yang bervariasi dan oriented *move operation*, operasi logical dan beberapa operasi khusus yang didukung oleh mesin.

Instruksi RR, RS dan SS yang digambarkan di atas ditulis dalam *two address instruction format*. Terdapat dua bagian untuk alamat operand yang terdapat pada instruksi, dimana setiap operand memiliki alamat masing-masing.

Alamat pada register ditunjukkan dengan penomoran register. Untuk itu, dikenal alamat absolut atau spesifikasi alamat secara langsung. Pengalamatan untuk operand dapat dilakukan dengan berbagai cara, diantaranya :

1. *Direct Addressing :* Alamat absolut dari lokasi penyimpanan ditetapkan dalam instruksi. Suatu index register dapat pula direpresentasikan sebagai optional. Alamat penyimpanan yang efektif dari operand kemudian akan dikomputasi sebagai **<index register> + absolute address**.
2. *Base displacement addressing* : Alamat penyimpanan operand ditunjukan dengan dua komponen. Komponen pertama adalah nomor register yang disebut *base register*. *Base register* akan berisi alamat penyimpanan absolut. Komponen kedua adalah nomor absolut yang berperan sebagai *displacement* (pemindahan) dari alamat register semula. Alamat penyimpanan yang efektif dari dikomputasi dengan cara **<base register> + displacement**. Alamat ini dapat dimodifikasi menggunakan *optional index register*.

Adanya *addressing mode* tidak akan menyebabkan kekeliruan pada mesin, karena adanya *addressing structure*. *Addressing structure* merupakan metode dasar dalam menentukan pengalamantan operand secara effektif. Di lain sisi, *addressing mode* adalah variasi yang digunakan dalam sturktur pengalamatan dasar atau gabungan beberapa tindakan yang berhubungan dengan pengalamatan.

Pada saat melakukan eksekusi, CPU secara terus menerus memonitor situasi khusus di dalam atau di luar lingkungannya. Situasi khusus yang timbul dari dalam CPU itu sendiri dapat berupa kondisi *arithmetic overflow/underflow*, penggunaan *invalid operation code*, dsb. Ketika situasi itu terjadi perangkat keras mesin akan menghidupkan *trap.* Tujuan dari *trap* adalah menarik perhatian CPU untuk kejadian tersebut. Jika perlu, CPU dapat mengalihkan eksekusi dari instruksi yang sedang berjalan dan menampilkan suatu aksi untuk memperoleh kembali situasi tersebut.

Suatu *interrupt* timbul sebagai suatu situasi khusus yang terjadi di luar CPU, seperti berakhirnya operasi IO, pengetikan kunci di terminal,dsb. Seperti halnya trap, tujuan dari interrupt adalah juga menarik perhatian CPU sehingga dapat melakukan aksi secara tepat. Sebagai contoh, berakhirnya operasi IO merupakan awal dari operasi IO yang lain yang membutuhkan device yang sama, penekanan kunci pada terminal membutuhkan system operasi meresponnya melalui prompt, dsb.

Suatu *interval timer* adalah unit perangkat keras yang mampu untuk membangkitkan *time-out interrupt* ketika awal penetapan interval waktu berakhir. Di sini akan ditemui register yang akan memanggil CPU yang membutuhkan interval waktu. Register akan melakukan pengurangan secara periodical setiap microsecond dan interrupt-pu akan terjadi pada saat register mencapai nilai nol.

C. IO Channel

Tujuan dari *IO channel* adalah membebaskan CPU selama operasi IO. Hal ini tercapai dengan adanya *data path* antara main storage dan IO device, serta monitoring kemajuan eksekusi operasi IO. Seperti pada penjelasan sebelumnya, multiprogramming terjadi pada saat CPU berada dalam keadaan bebas utnuk mengeksekusi instruksi, dimana pada waktu yang sama IO sedang bekerja. Tahapan eksekusi dari operasi IO adalah sebagai berikut :

(i) CPU mengeksekusi instruksi “Start IO”, penamaan IO device dilakukan untuk menunjukkan operasi tersebut.

(ii) IO channel akan menentukan IO device mana yang paling tersedia untuk operasi tersebut. Sekumpulan kode kondisi dalam PSR akan merefleksikan penemuan itu. Instruksi "Start IO" akan berhenti. CPU dapat menganalisa kode kondisi itu untuk menentukan aksi selanjutnya, sebagai contoh: mengulangi operasi (retry operation), meng-cancel program yang diinginkan IO atau men-switch eksekusi ke program lain.

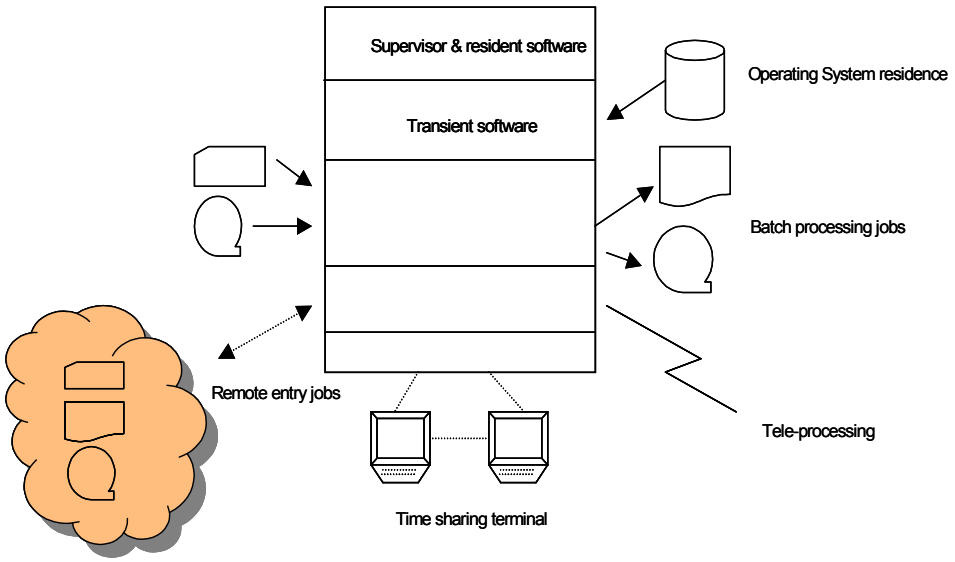
(iii) Jika IO device tersedia, IO channel mengambil informasi yang terkait dengan IO operation yang dilakukan nya dari sebuah area pada main storage. Kemudian dilakukan set up eksekusi IO operation pada device.

(iv) Pada IO operation,akan timbul interrupt pada device. Hal ini akan menyebabkan IO channel dipindahkan ke CPU.

Ada dua jenis IO channel : *Multiplexor channel* dan *selector channel*. Setiap channel memiliki sejumlah peralatan yang dapat dihubungkan. Peralatan yang berjalan perlahan seperti terminal dan printer terkoneksi ke *multiplexor channel*. Sedangkan peralatan yang berjalan lebih cepat seperti, magnetic tapes atau disk yang memiliki tingkat transfer tinggi terkoneksi dengan selector channel.

I.4.2 Operating System

Gambar I.2 menunjukkan gambaran system komputer dengan lingkungan kerjanya yang didukung oleh system operasi. *Batch processing, time-sharing* dan *remote jobs* ditunjukkan melalui proses yang berkesinambungan. Komponen system software terletak pada *operating system residence disk pack*. Hanya beberapa komponen OS yang tinggal pada main store secara permanen , sedangkan yang lain akan di-*load* ke dalam *transient area* di *main store* pada saat dibutuhkan.



Gambar I.2. Lingkungan Job processing pada sistem operasi

Tabel I.1 menunjukkan daftar komponen system software. Komponen-komponen tersebut dikelompokkan ke dalam 3 kelompok besar, yaitu *operating system software, standard system software* dan *aplicaton software*. Perbedaan antara system dan application software diantaranya adalah *system software* selalu dibutuhkan setiap kali user program dijalankan, sedangkan *application software* lebih pada software yang dirancang untuk memenuhi kebutuhan khusus dari user. Selanjutnya perbedaan antara *operating system software* dan *standard system software* adalah pada dasarnya apakah suatu komponen software dibutuhkan untuk pengelolaan sumberdaya system komputer agar lebih effektif atau hanya dibutuhkan untuk mendukung pemasukan kunci (*key in*),

editing dan pemrosesan user program. *Languange processor, editor, loader* dan utility manipulasi file termasuk ke dalam *standard system software*. Perbedaan nyatanya adalah languange translator melakukan diagnostik yang cukup baik untuk meningkatkan pemanfaatan sumberdaya komputer, karenanya penanganan user program, IO device dsb menjadi tanggungjawab system operasi.

Tabel I.1. Komponen Sistem Software

1. ***Operating System Software***	
Storage Manager	- Mengatur strategi alokasi/dealokasi storage - Mengatur strategi swapping dan demand paging
Processor Manager (Process Manager)	- mengatur strategi alokasi/dealokasi processor
File System Manager	- Mengelola akses file oleh user - Mengelola directory (proteksi, dsb) - Mengelola alokasi/dealokasi file
Input-Output Control System	- Metode pengaksesan (LIOCS) - Penjadualan IO dan pengelolaannya (PIOCS)
Communication Manager	
Operator Console Manager	
2. ***Standard System Software***	
Language Processor	- Assember, Compiler, Interpreter
Loader	- Linking Loader, Absolute Loaders
Software Tools	- Debugging Aids, Text Editor, Utilities
3. ***Applications Software***	- Sort/Merge Package - Payroll/Accounting Package - Data Base Management System