10. Konsep Operasional Prosessor dan Memori

10.1. Hubungan antara prosessor dan memori

Pada modul ini, akan dipelajari prinsip dasar dari hubungan antara prosessor dan memori. Dimulai dengan lokasi memori dan alamatnya, pada pembahasan ini dibuat suatu model abstrak dari memori yang terdiri dari kumpulan *cell* yang tiap *cell*nya dapat menyimpan *n* bit. Memori kemudian dialamati untuk dapat ditulis dan dibaca ke *cell* tertentu (tiap *cell* memiliki alamat yang berbeda). Beberapa cara untuk mengalamati lokasi memori (Mode Pengalamatan) akan dibahas lebih lanjut pada modul ini.

Suatu karakteristik unik dari memori adalah bahwa memori tersebut harus diorganisasikan dalam suatu *hierarchy*. Pada *hierarchy* tersebut, memori yang berukuran lebih besar dan kecepatannya lebih lambat digunakan untuk mendukung memori yang berukuran kecil tetapi memliki kecepatan tinggi. *Hierarchy* awal dari memori diawali dari memori yang berukuran kecil, mahal, tetapi kecepatan aksesnya tinggi, dinamakan dengan *cache memory*. Diikuti *hierarchy* berikutnya adalah memori yang berukuran lebih besar, harga lebih murah, tetapi kecepatan aksesnya lebih lambat dari *cache.*

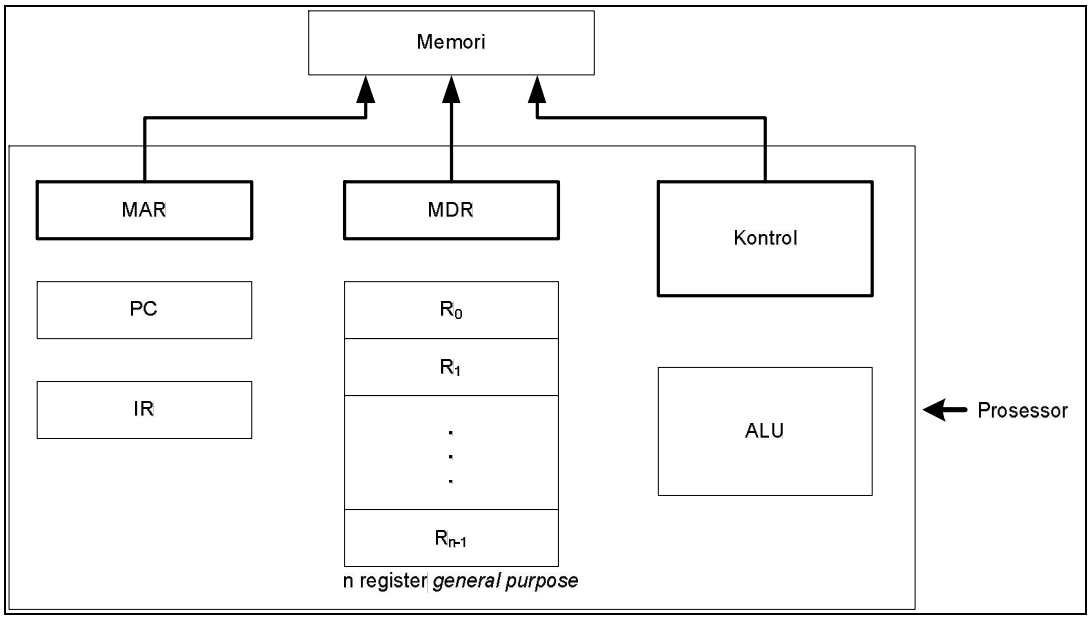
Aktivitas dalam komputer diatur oleh instruksi. Untuk melakukan suatu tugas tertentu, suatu program yang berisi daftar instruksi disimpan dalam memori. Instruksi individu dibawa dari memori ke prosessor, yang mengeksekusi operasi tertentu. Data yang digunakan sebagai operand juga disimpan dalam memori. Berikut ini adalah contoh dari suatu instruksi :

LOAD LOCA, R1
ADD R1,R0

Baris pertama dari instruksi diatas digunakan untuk mentransfer isi lokasi memori *LOCA*, ke register prosessor *R1,* dan instruksi kedua menambahkan isi register *R1* dan *R0* dan menyimpan hasil penjumlahan pada register *R0.* Dua instruksi diatas menghasilkan suatu operasi yang menghancurkan isi register *R1* dan *R0* sebelum instruksi dieksekusi, sedangkan isi memori lokasi *LOCA* tetap dipertahankan.

Transfer antara memori dan prosessor dimulai dengan mengirim alamat lokasi memori yang akan diakses ke unit memori dan menyampaikan sinyal kontrol yang sesuai. Data tersebut kemudian ditransfer ke atau dari memori.

Gambar berikut menunjukkan bagaimana memori dan prosessor dapat dihubungkan. Gambar tersebut juga meunjukkan beberapa detil operasional penting pada prosessor yang belum dibahas. Pola interkoneksi untuk komponen ini tidak ditunjukkan secara detil.



Gambar 10.1. Hubungan antara prosessor dan memori

Selain ALU dan unit kontrol, prosessor berisi sejumlah register yang digunakan untuk beberapa tujuan yang berbeda. *Instruction Register (IR)* menyimpan instruksi yang sedang dieksekusi. Outputnya dipersiapkan untuk unit kontrol, yang membangkitkan sinyal timing yang mengendalikan berbagai elemen pengolahan yang terlibat dalam eksekusi tersebut. *Program Counter* adalah register khusus yang lain. PC mencatat eksekusi suatu program. PC berisi alamat memori dari instruksi selanjutnya akan diambil dan dieksekusi. Selama eksekusi suatu instruksi, isi PC akan di-*update* untuk menyesuaikan dengan alamat instruksi berikutnya yang akan dieksekusi. Secara umum dapat dikatakan bahwa PC menunjuk pada instruksi berikutnya yang akan diambil dari memori. Selain IR dan PC, gambar berikut menunjukkan register *general purpose*, R_0 hingga R_{n-1}.

Dua register yang berfungsi sebagai fasilitator komunikasi dengan memori. Register tersebut adalah *Memory Address Register (MAR)* dan *Memory Data Register (MDR)*. MAR menyimpan alamat lokasi memori yang diakses. MDR berisi data yang akan ditulis atau dibaca dari memori.

Berikut adalah langkah operasional umum. Program terletak di memori dan biasanya berada disana melalui unit input. Eksekusi program dimulai saat PC di-set untuk menunjuk ke instruksi pertama dari program tersebut. Isi PC ditransfer ke MAR dan sinyal kontrol READ dikirim ke memori. Setelah waktu yang diperlukan untuk mengakses memori habis, *word* yang dimaksud (dalam hal ini, instruksi pertama dari program tersebut) dibaca dari memori dan diload ke MDR. Selanjutnya, isi dari MDR ditransfer ke IR. Pada titik ini, instruksi siap di-*decode* dan dieksekusi.

Jika instruksi tersebut melibatkan operasi yang harus dilakukan oleh ALU, maka perlu untuk mendapatkan operand yang diperlukan. Jika operand tersebut terletak dalam memori (dapat juga berada di register *general purpose*), maka harus diambil dengan mengirimkan alamatnya ke MAR dan menginisiasi siklus Read. Setelah operand dibaca dari memori ke MDR, kemudian ditransfer dari MDR ke ALU, Setelah satu atau lebih operand diambil dengan cara tersebut, maka hasilnya dikirim ke MDR. Alamat lokasi tempat hasil tersebut disimpan dikirimkan ke MAR, dan siklus Write diinisiasi. Jadi segera setelah instruksi terakhir diselesaikan, pengambilan instruksi yang baru dapat dialamati.

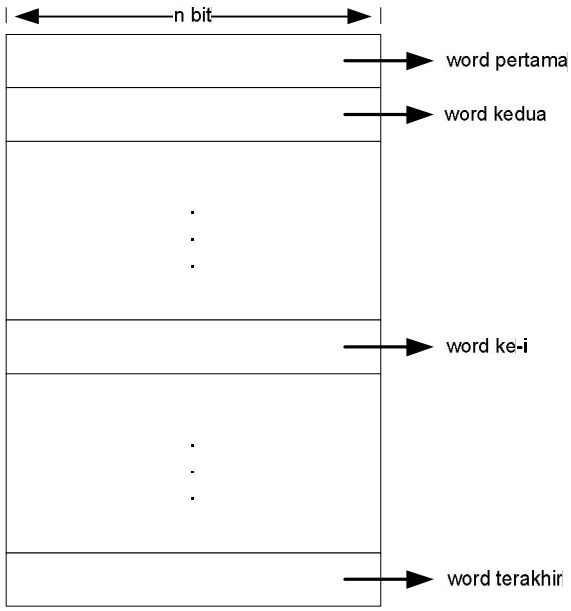
Selain mentransfer data antara memori dan prosessor, komputer menerima data dari peralatan input dan mengirimkan data ke peralatan output. Jadi tersedia beberapa instruksi mesin dengan kemampuan untuk menangani transfer I/O.

Eksekusi normal program dapat digantikan jika beberapa peralatan memerlukan pelayanan darurat. Misalnya, peralatan monitoring dalam proses industri yang dikendalikan dengan komputer dapat mendeteksi kondisi bahaya. Dalam rangka menangani situasi tersebut dengan segera, maka eksekusi normal dari program yang sedang berjalan harus diinterupsi. Untuk melakukan hal ini, peralatan tersebut menyampaikan sinyal *interrupt*. *Interrupt* adalah permintaan layanan dari peralatan I/O terhadap prosessor. Prosessor menyediakan layanan yang diminta dengan mengeksekusi *interrupt service routine* yang sesuai. Karena diversi semacam itu dapat mempengaruhi keadaan internal prosessor, maka keadaan tersebut harus disimpan kedalam lokasi memori sebelum melayani interupsi. Biasanya, isi PC,

register umum, dan beberapa informasi kontrol disimpan dimemori. Pada saat *interrupt service routine* telah selesai, keadaan prosessor dipulihkan sehingga program yang diinterupsi dapat dilanjutkan.

10.2. Lokasi Memori dan Alamat

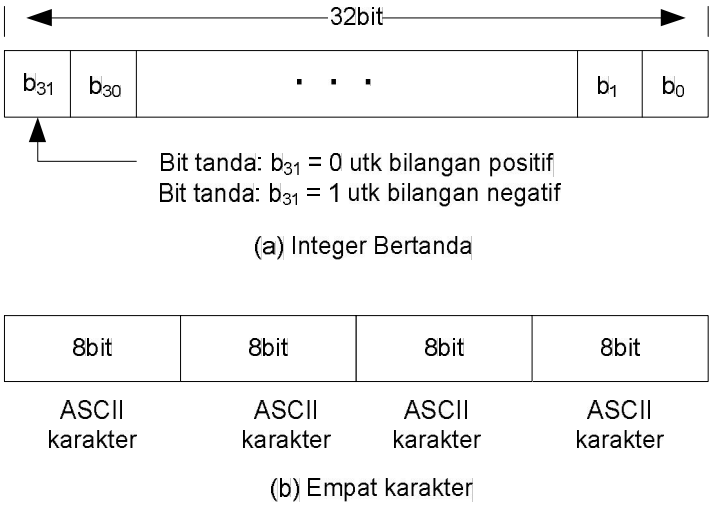
Operand bilangan dan karakter, seperti halnya instruksi, disimpan dalam memori komputer. Sekarang kita akan membahas bagaimana memori diatur. Memori terdiri dari jutaan sel penyimpanan, dimana tiap sel tersebut menyimpan suatu bit informasi yang berupa nilai 0 dan 1. Karena bit tunggal mewakili jumlah informasi yang sangat sedikit, maka bit jarang ditangani secara individu. Pendekatan yang umum adalah menanganinya dalam kelompok dengan ukuran tertentu. Untuk tujuan ini, memori tersebut diatur sehingga kelompok n bit disebut word informasi (beberapa referensi menyebut juga dengan nama register), dan n disebut *word length*. Memori suatu komputer dapat digambarkan secara skematis sebagai kumpulan word seperti pada gambar berikut.



Gambar 10.2.

Komputer modern memiliki *word length* yang biasanya berkisar antara 16 – 64 bit. Jika *word length* suatu komputer adalah 32 bit, maka word tunggal dapat menyimpan 32-bit bilangan komplemen atau empat karakter ASCII, masing-masing memiliki 8bit, sebagaimana ditunjukkan pada gambar berikut. Suatu unit 8bit disebut *byte*. Instruksi mesin mungkin memerlukan satu atau lebih word untuk mewakilinya.

Berikut akan dibahas bagaimana instruksi di-*encode* menjadi word memori pada bagian selanjutnya.



Gambar 10.3.

Mengakses memori untuk menyimpan atau mengambil suatu item informasi, baik berupa word atau byte, memerlukan nama yang berbeda atau alamat tiap lokasi item. Merupakan hal yang biasa menggunakan bilangan dari 0 hingga 2^k-1, untuk beberapa nilai k yang sesuai, sebagai alamat yang berurutan dalam memori. Alamat 2^k meliputi ruang alamat komputer tersebut, dan memori tersebut dapat memiliki lokasi *addressable* hingga 2^k. Misalnya alamat 24bit menghasilkan ruang alamat 2^{24} (16.777.216) lokasi. Bilangan ini biasanya ditulis 16M(16Mega), dimana 1M adalah bilangan 2^{20}(1.048.576). Alamat 32bit menghasilkan ruang alamat 2^{32} atau 4G(4Giga) lokasi, dimana 1G adalah 2^{30}. Konvensi yang biasa digunakan adalah K(kilo) untuk bilangan 2^{10}(1024), dan T(tera) untuk bilangan 2^{40}.

10.3. Byte Addressibility

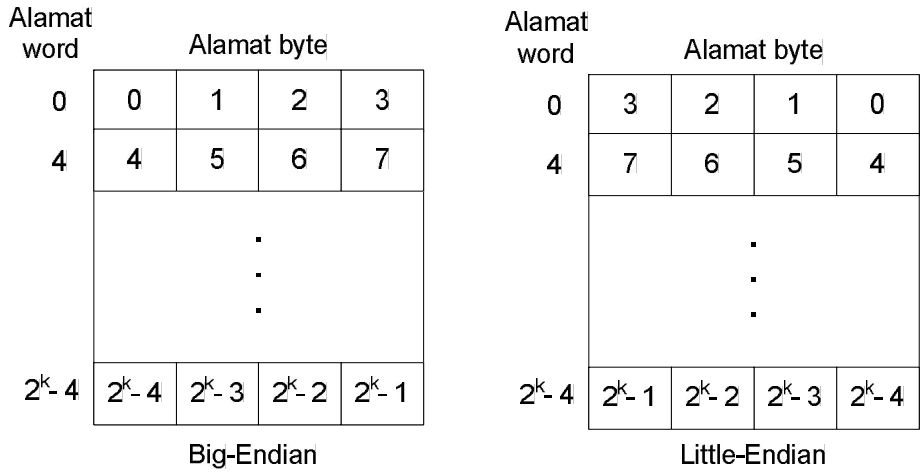
Sekarang terdapat tiga kuantitas informasi dasar yang telah dijelaskan, yaitu bit, byte, dan word. Byte selalu 8bit, tetapi word length biasanya berada pada rentang 16 hingga 64 bit. Sangat tidak praktis untuk menetapkan alamat yang berbeda bagi lokasi bit individu dalam memori. Penetapan paling praktis adalah dengan alamat yang berurutan mengacu pada lokasi byte yang berurutan dalam memori. Ini

merupakan penetapan yang digunakan pada sebagian besar komputer modern. Istilah yang digunakan adalah *byte addressable momory*. Lokasi byte memiliki alamat 0,1,2,… sehingga, jika word length suatu mesin adalah 32bit, maka word yang berurutan berada pada alamat 0,4,8, …, dengan tiap word terdiri dari empat byte.

10.3.1. Penetapan *Big-Endian* dan *Little-Endian*

Terdapat dua cara penetapan alamat byte pada word, sebagaimana ditampilkan pada gambar berikut. Nama *big-endian* dipakai jika alamat byte rendah untuk *Most Significant Byte* (byte paling kiri) dari word tersebut. Nama *little-endian* digunakan untuk pengaturan sebaliknya, yaitu alamat byte rendah dipakai untuk *less significant byte*(byte paling kanan) dari word tesebut. Kata "*most significant"* dan "*less significant"* digunakan dalam kaitannya dengan weight (pangkat 2) yang ditetapkan pada bit pada saat word tersebut menyatakan suatu bilangan. Penetapan *Big-Endian* dan *Little-Endian* digunakan dalam mesin komersial. Pada kedua kasus tersebut, alamat byte 0,4,8,… digunakan sebagai alamat word yang berurutan dalam memori dan merupakan alamat yang digunakan pada saat menetapkan operasi baca tulis memori untuk word.

Selain menentukan urutan alamat byte dalam word, juga perlu menentukan label bit atau word. Konvensi yang paling umum, ditampilkan pada gambar diatas. Konvensi tersebut merupakan penyusunan paling alami untuk data numerik. Penyusunan yang sama juga digunakan untuk menetapkan label bit dalam byte, yaitu $b_7,b_6,\ldots,b_0$, dari kiri ke kanan. Namun ada pula komputer yang menggunakan penyusunan sebaliknya.



Gambar 10.4.

10.3.2. *Word Allignment*

Dalam 32-bit *word length*, batasan word alami terjadi pada alamat 0,4,8,…, sebagaimana ditunjukkan pada gambar 10.4. diatas. Dikatakan bahwa lokasi word tersebut memiliki alamat *aligned address.* Secara umum, word disebut *aligned* dalam memori jika word tersebut mulai pada alamat byte yang merupakan kelipatan jumlah byte didalam word. Untuk alasan praktis yang dihubungkan dengan manipulasi alamat *binary code*, jumlah byte dalam word adalah pangkat 2. Karena itu jika *word length* adalah 16(2 byte), maka aligned word mulai pada alamat byte 0,2,4,… dan untuk *word length* 64(2^3 byte), maka *aligned word* mulai pada alamat byte 0,8,16,…

10.4. Mengakses Bilangan, Karakter, dan String Karakter

Sebuah bilangan biasanya memiliki satu word. Bilangan tersebut dapat diakses dalam memori menetapkan alamat wordnya. Seperti halnya karakter individu dapat diakses melalui alamat byte-nya.

Pada banyak aplikasi, diperlukan penanganan string karakter variable length. Awal string diindikasikan dengan menyatakan byte yang berisi karakter pertama pada alamat tersebut. Lokasi byte yang berurutan berisi karakter string yang berurutan. Terdapat dua cara untuk mengindikasikan panjang string tersebut. Suatu karakter kontrol khusus yang berarti “end of the string” dapat digunakan sebagai karakter terakhir dalam string tersebut, atau lokasi word memori atau register prosessor terpisah dapat berisi suatu bilangan yang mengindikasikan panjang string di dalam byte.

10.5. Mode Pengalamatan

Seluruh Informasi yang diperlukan oleh operasi apapun yang dilakukan oleh CPU harus dialamati. Dalam Ilmu Komputer, informasi tersebut dinamakan *operand.* Seluruh operasi yang dipakai pada prosessor sedikitnya memiliki 2 tipe informasi. Instruksi yang dipakai, di-*encode* dan dinamakan *op-code*, dan informasi alamat di-*encode* dan dinamakan *address*.

Instruksi dapat diklasifikasikan berdasarkan jumlah *operand* yaitu : *three-address (*tiga alamat*), two address(*dua alamat*), one-and-half-address(*satu-setengah alamat*), one address(*satu alamat*), zero address(*nol alamat*)*. Pada contoh-contoh berikut, instruksi yang digunakan menggunakan format *operasi*, *sumber*, *tujuan* untuk

mewakili instruksi apapun. '*Operasi'* mewakili operasi-operasi yang digunakan, contohnya adalah *add, substract, write,* atau *read. 'Sumber'* mewakili *operand sumber. Operand sumber* dapat berupa konstanta, nilai yang disimpan pada register, atau nilai yang disimpan pada memori. '*Tujuan'* mewakili tempat dimana hasil operasi disimpan, bisa di register atau di memori.

Instruksi tiga alamat memiliki bentuk *operasi add-1, add-2, add-3*. Pada bentuk ini, tiap *add-1, add-2, add-3* menunjuk ke register atau ke memori tertentu. Sebagai contoh, instruksi *ADD R1,R2,R3.* Instruksi ini mengindikasikan bahwa operasi yang dilakukan adalah *Addition* (Penjumlahan). Instruksi ini juga mengindikasikan bahwa data yang dijumlahkan adalah data yang tersimpan di register *R1* dan *R2,* dan hasil penjumlahan disimpan pada register *R3*. Contoh dari Instruksi tiga alamat yang menggunakan lokasi memori berbentuk *ADD A,B,C.* Instruksi tersebut menambahkan data yang ada pada memori lokasi *A* dan *B*, dan menyimpan hasilnya pada memori lokasi *C*.

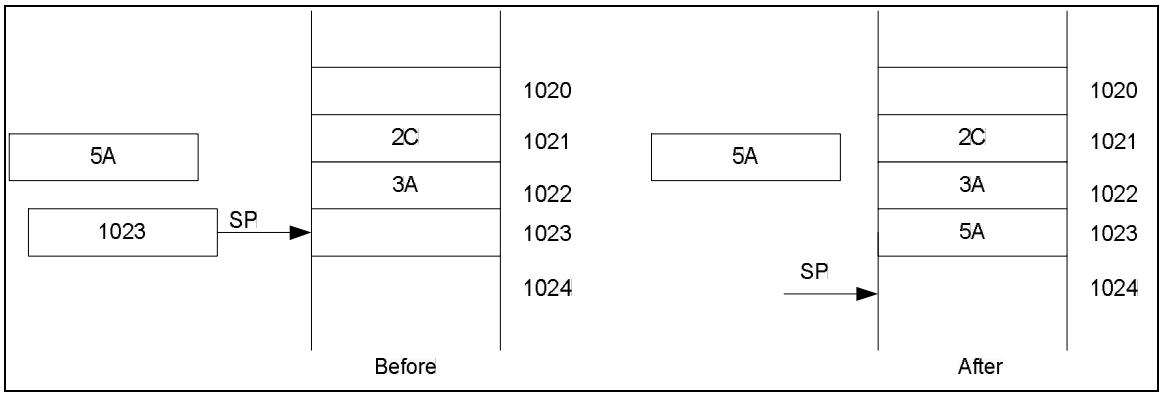
Instruksi dua alamat memiliki bentuk *operasi add-1, add-2*. Pada bentuk ini, tiap *add-1, add-2,* menunjuk ke register atau ke memori tertentu. Sebagai contoh, instruksi *ADD R1,R2.* Instruksi ini mengindikasikan bahwa data yang dijumlahkan adalah data yang tersimpan di register *R1* dan *R2,* dan hasil penjumlahan disimpan pada register *R2.* Contoh dari Instruksi dua alamat yang menggunakan lokasi memori berbentuk *ADD A,B.* Instruksi tersebut menambahkan data yang ada pada memori lokasi *A* dan *B*, dan menyimpan hasilnya pada memori lokasi *B*.

Instruksi satu alamat memiliki bentuk *ADD R1*. Pada kasus berikut, instruksinya menunjuk ke register, dinamakan *Accumulator* R_{acc}. Data dari *Accumulator* ditambahkan dengan data dari register *R1,* kemudian hasilnya disimpan pada *Accumulator.* Jika data di memori yang digunakan, maka bentuk instruksinya adalah *ADD B.* Pada bentuk ini, operasinya digunakan untuk menjumlahkan data dari Accumulator dengan data dari memori lokasi B, kemudian hasil penjumlahan disimpan pada *Accumulator.* Instruksi *ADD R1* ekivalen dengan instruksi tiga alamat *ADD R1,*R_{acc}*,*R_{acc} atau instruksi dua alamat *ADD R1,*R_{acc}.

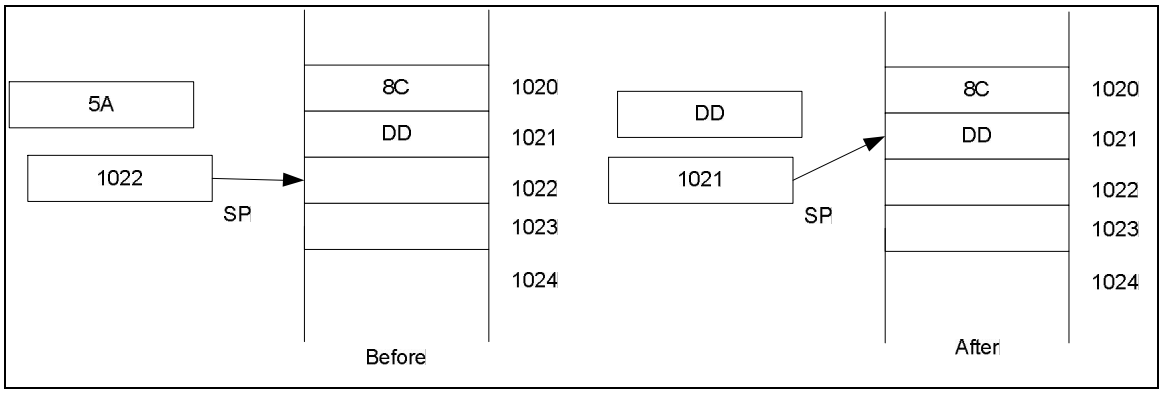
Diantara instruksi dua dan satu alamat, terdapat instruksi satu-setengah alamat. *ADD B,R1* adalah termasuk instruksi satu-setengah alamat. Pada bentuk ini, operasinya adalah menambah data Register *R1* dengan data dari memori lokasi *B*, kemudian menyimpan hasil penjumlahan di register *R1.* Faktanya, instruksi tersebut

menggunakan dua tipe pengalamatan, yaitu register dan lokasi memori, hal ini disebut dengan instruksi satu-setengah alamat, karena pengalamatan register membutuhkan jumlah bit yang lebih sedikit dibandingkan dengan bit yang dibutuhkan untuk pengalamatan memori.

Selain instruksi-instruksi tersebut diatas, ada pula instruksi yang dinamakan instruksi nol alamat. Instruksi-instruksi ini biasana menggunakan operasi *stack*. *Stack* adalah mekanisme organisasi data dimana data yang paling akhir tersimpan, adalah data pertama yang disimpan atau dieluarkan. Dua operasi khusus dari *stack* adalah operasi *push* dan *pop*. Gambar berikut menunjukkan operasi-operasi tersebut.



Gambar 10.5. Operasi *Stack Push*

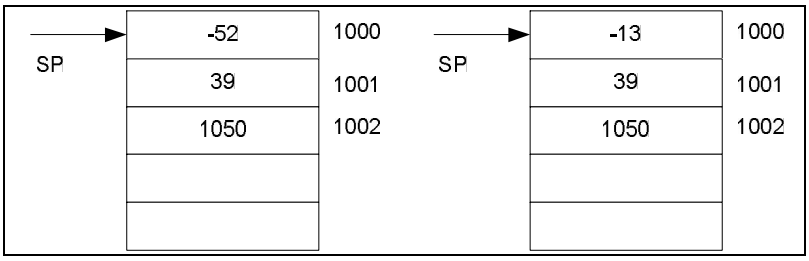


Gambar 10.6. Operasi *Stack Pop*

Seperti tampak pada gambar diatas, Register khusus, dinamakan *Stack Pointer* (SP), digunakan untuk mengindikasikan lokasi stack yang dapat dialamati. Pada operasi *Stack Push*, nilai dari SP digunakan untuk mengindikasikan lokasi (dinamakan stack teratas) dimana nilai (5A) akan disimpan (pada contoh ini lokasinya adalah 1023). Setelah menyimpan (*pushing*) nilai ini, maka SP-nya ditambahkan untuk menunjuk ke lokasi 1024. Pada operasi *Stack Pop,* SP dikurangkan menjadi

1021. Data yang tersimpan pada lokasi ini (DD dalam hal ini) dikeluarkan (*popped out*) dan disimpan pada register yang tampak.

Beberapa operasi dapat dipakai menggunakan struktur *stack. ADD (SP)+, (SP)* adalah salah satu bentuk instruksi yang menggunakan struktur *stack.* Instruksi ini menambahkan data yang ditunjuk oleh SP dan SP+1, dan menyimpan hasilnya dilokasi yang ditunjuk oleh SP. Gambar berikut menunjukkan operasi penjumlahan menggunakan *stack.*



Gambar 10.7. Penjumlahan menggunakan *stack*

Cara lain dimana operand dapat dialamati dinamakan *Mode Pengalamatan* (*Addressing Mode*). Mode pengalamatan berbeda dari pengalamatan operand yang telah dijelaskan diatas. Mode pengalamatan paling sederhana adalah dengan memasukkan operand itu sendiri pada instruksi, hanya itu, tidak ada informasi alamat yang ditambahkan. Mode ini dinamakan dengan *immediate addressing.* Operasi yang termasuk juga dalam mode pengalamatan adalah menghitung alamat *operand* dengan menambahkan nilai konstanta dengan isi dari register. Operasi ini dinamakan dengan *indexed addressing*.Selain dua mode pengalamatan tersebut, terdapat beberapa mode pengalamatan lain, diantaranya *absolute addressing*, *direct addressing*, dan *indirect addressing*.

Klasifikasi instruksi	Contoh
Tiga alamat	*ADD R1,R2,R3*
	ADD A,B,C
Dua alamat	*ADD R1,R2*
	ADD A,B
Satu-setengah alamat	*ADD B,R1*
Satu alamat	*ADD R1*

Nol alamat	*ADD (SP)+,(SP)*

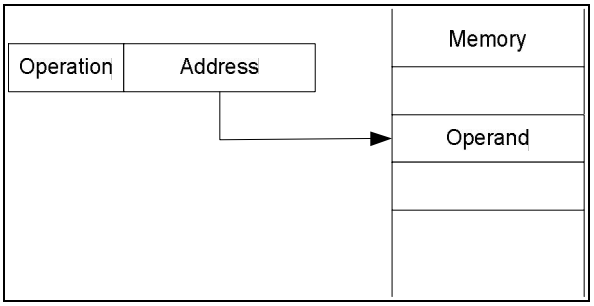
Tabel 10.1.

1. *Immediate Addressing*

   Pada mode pengalamatan ini, nilai dari operand adalah (segera) terdapat pada instruksi itu sendiri. Sebagai contoh, instruksi yang digunakan untuk menyimpan nilai desimal kedalam register Ri. Instruksinya adalah : *LOAD #1000, Ri*. Pada instruksi tersebut, operasi yang dilakukan adalah menyimpan nilai ke dalam register. *Operand* sumber adalah (segera) diberikan nilai 1000, dan tujuannya adalah register *Ri.* Nilai 1000 pada contoh tersebut adalah operand itu sendiri dan bukan alamatnya (*immediate mode*), ditandai dengan awalan '#' yang menunjukkan bahwa nilai tersebut adalah data dan bukan alamat. Seperti terlihat pada contoh, bahwa penggunaan *immediate addressing* sangat sederhana. Akan tetapi mode ini jarang dipakai pada pemrograman umum, hal ini dikarenakan data yang ada bersifat statis, membutuhkan penggantian nilai untuk tiap instruksi yang menggunakan nilai segera. Mode pengalamatan berikut adalah mode yang lebih fleksibel daripada *Immediate Addressing.*

2. *Direct (Absolute) Addressing*

   Pada mode pengalamatan ini, alamat dari lokasi memori berada didalam instruksi. Sebagai contoh, data yang berada pada memori lokasi 1000 kedalam register *Ri.* Operasi ini dapat menggunakan instruksi *LOAD 1000,Ri.* Pada instruksi ini, *operand* sumber adalah nilai yang disimpan pada memori dengan alamat 1000, dan tujuannya adalah disimpan pada register *Ri.* Perhatikan bahwa nilai 1000 tanpa awalan apapun, menunjukkan bahwa nilai tersebut adalah alamat dari *operand* sumber. Gambar berikut menunjukkan mode *Direct Addressing*.

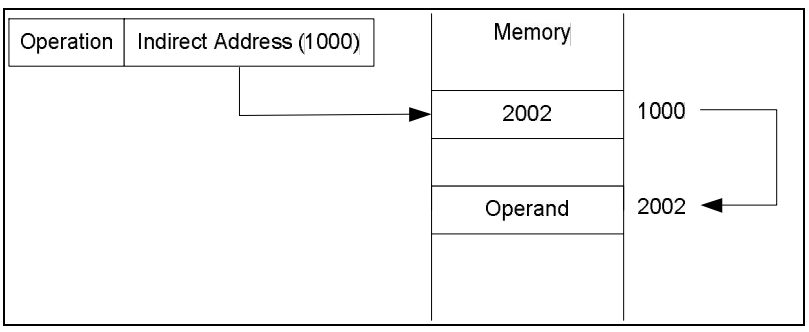


Gambar 10.8. Mode *Direct Addressing*

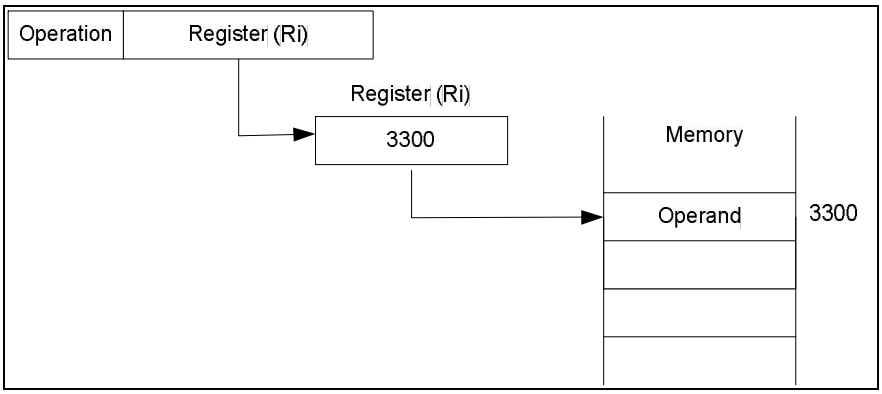
Jika dimisalkan bahwa memori dengan alamat 1000 memiliki nilai -345 dan ketika instruksi *LOAD 1000,Ri* dieksekusi, maka nilai -345 disimpan kedalam register *Ri*.

3. *Indirect Addressing*

Pada mode *indirect*, yang berada didalam instruksi bukan merupakan alamat dari operand, tetapi menunjuk ke memori atau register tertentu yang nilainya disimpan oleh register atau memori dengan alamat operand, biasanya ditandai dengan 'tanda kurung'. Perhatikan intruksi *LOAD (1000), Ri.* Instruksi ini menggunakan alamat memori 1000 dengan tanda kurung, yang menunjukkan ketidaklangsungan (*Indirection*). Operasi yang dilakukan oleh instruksi ini adalah mengisikan register *Ri* dengan data yang berasal dari memori yang alamatnya ditunjukkan oleh data dari memori alamat 1000. Jika data dari alamat memori 1000 adalah 2002, maka data yang diisikan ke register *Ri* adalah data dari memori alamat 2002. Mode *indirect* ini dinamakan dengan M*emory Indirect Addressing.* Jika register yang digunakan sebagai acuan alamat bagi register atau memori lain, maka mode ini dinamakan dengan *Register Indirect Addressing.* Perhatikan gambar berikut :



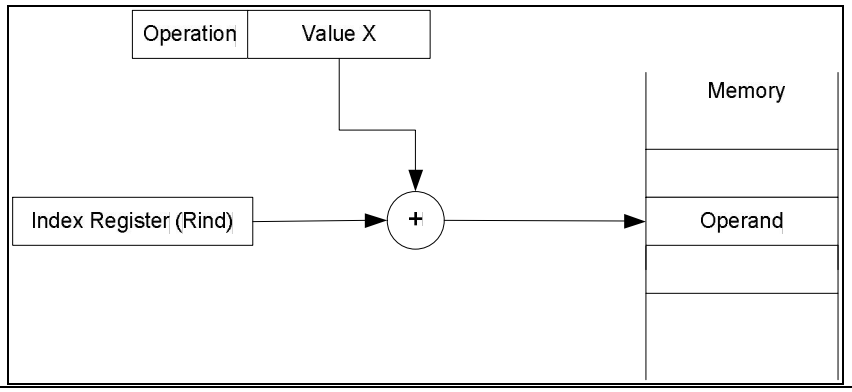
Gambar 10.9. Memory Indirect Addressing



Gambar 10.10. Register Indirect Addressing

4. *Indexed Addressing*

Pada mode *Indexed Addressing,* alamat dari *operand* didapatkan dengan menambahkan suatu konstanta dengan isi dari suatu register, yang dinamakan dengan *index register*. Salah satu contoh instruksi yang menggunakan *indexed addressing* adalah : *LOAD X(R_{ind}), Ri*. Operasi yang dilakukan oleh instruksi ini adalah mengisi register *Ri* dengan data dari memori dengan alamat nilai X + nilai dari R_{ind}. Alamat index ditunjukkan dengan memberikan tanda kurung pada register index dan menggunakan simbol X untuk menunjukkan konstanta yang ditambahkan. Gambar berikut menunjukkan ilustrasi dari i*ndexed addressing mode.*



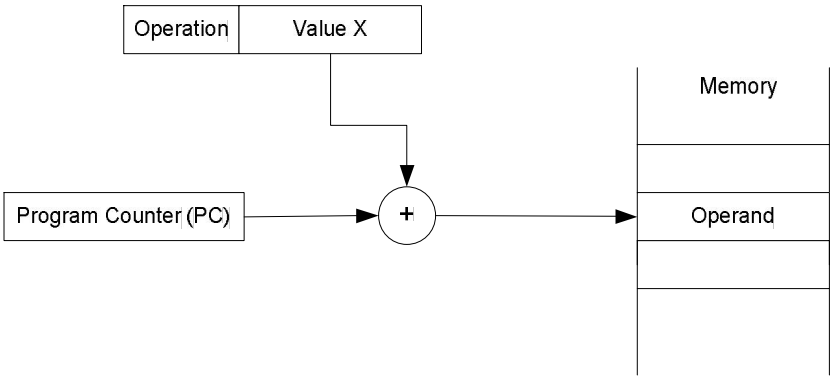
Gambar 10.11. Indexed Addressing Mode

5. Mode Pengalamatan lain

Mode-mode pengalamatan yang telah ditunjukkan diatas, adalah mode-mode yang paling sering dipakai oleh kebanyakan prosessor. Selain mode-mode pengalamatan tersebut diatas, ada beberapa mode-mode tambahan yang dikenal dengan nama mode *relative, autoincrement,* dan *autodecrement*.

a. Relative Mode

Relative mode mirip dengan *Indexed mode*. Pada *Indexed mode*, yang digunakan sebagai index adalah register (*index register*), sedangkan pada *Relative mode* tidak menggunaka *index register* tetapi menggunakan *Program Counter* (PC). Sebagai contoh, *LOAD X(PC), Ri* akan mengisi register *Ri* dengan data dari memori yang alamatnya adalah jumlah dari nilai *Program Counter* dengan nilai X. Gambar berikut mengilustrasikan *relative mode addressing.*

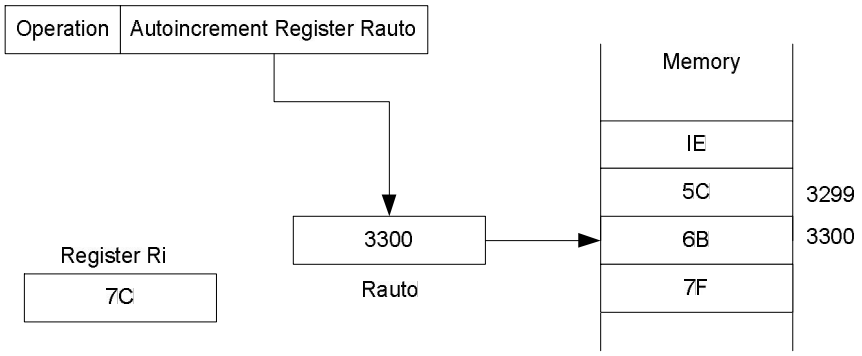


Gambar 10.12. Relative Mode Addressing

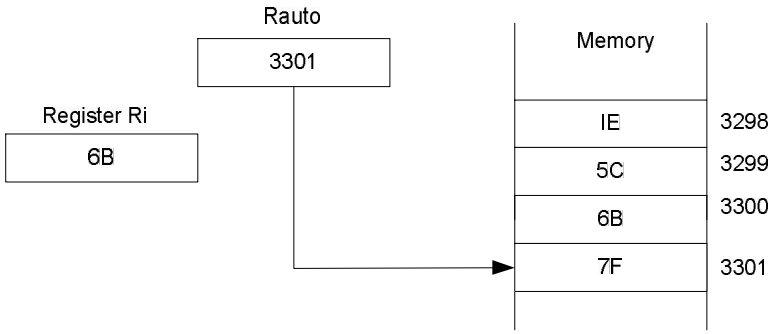
b. *AutoIncrement Mode*

AutoIncrement Mode mirip dengan *register indirect addressing mode,* hanya saja registernya dapat bertambah 1 setelah operasi dari instruksi dijalankan. Register ini dinamakan dengan *autoincrement register*. *Autoincrement register* diletakkan pada *source* dan diberi tanda kurung, dan ditambahkan simbol '+' untuk menunjukkan bahwa mode yang dipakai adalah *autoincrement mode*. Sebagai contoh, instruksi *LOAD* (R_{auto})*+,Ri*. Operasi dari instruksi ini akan mengisi register *Ri* dengan data dari *operand* yang

alamatnya ditunjukkan oleh register R_{auto}. Setelah mengisikan data ke register *Ri,* nilai dari register R_{auto} bertambah satu. Gambar berikut mengilustrasikan *autoincrement addressing mode.*



Sebelum Eksekusi

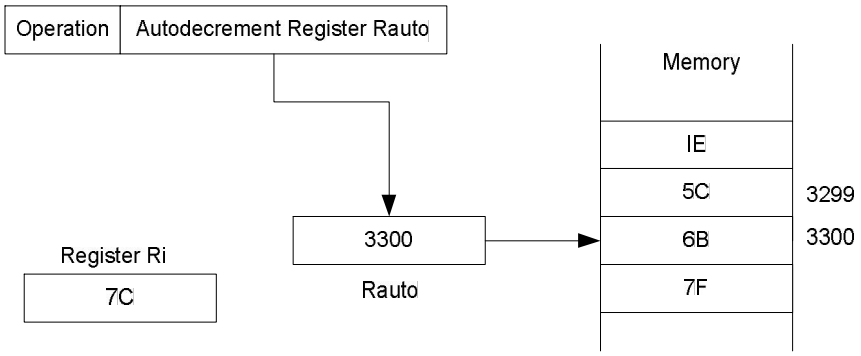


Sesudah Eksekusi

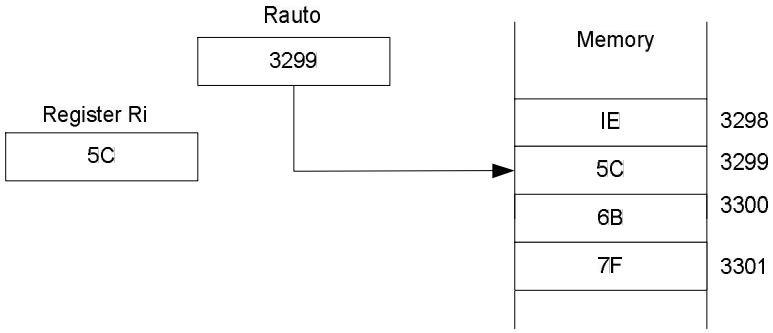
Gambar 10.13. Autoincrement Mode

c. *AutoDecrement Mode*

Mirip dengan *autoincrement*, *autodecrement mode* menggunakan register untuk menunjuk alamat operand. Akan tetapi pada mode ini, *autodecrement register* dikurangkan 1 terlebih dahulu, dan data baru yang digunakan sebagai alamat efektif dari operand. Untuk menunjukkan bahwa *autodecrement register* dikurangi terlebih dahulu sebelum mengakses *operand*, simbol '-' diletakkan sebelum *indirection operand.* Sebagai contoh, *LOAD –(R_{auto}),Ri.* Instruksi ini akan mengurangkan dengan 1 data yang berada pada R_{auto}, dan data alamat R_{auto} yang akan diisikan pada *Ri.*



Sebelum dieksekusi



Sesudah dieksekusi

Gambar 10.14. Autodecrement Mode

Rangkuman mode pengalamatan

Addressing Mode	Definisi	Contoh	Operasi
Immediate	Nilai dari operand langsung masuk kedalam instruksi	*load #1000, Ri*	*Ri ← 1000*
Direct (Absolute)	Alamat dari operand langsung masuk kedalam instruksi	*load 1000, Ri*	*Ri ← M[1000]*
Register Indirect	Operand berada dalam memori yang alamatnya berada pada register yang disebutkan pada instruksi	*load (Rj), Ri*	*Ri ← M[Rj]*
Memory Indirect	Operand berada dalam memori yang alamatnya berada pada memori yang disebutkan pada instruksi	*load (1000), Ri*	*Ri ← M[1000]*

Indexed	Alamat operand adalah jumlah dari nilai index dan data dari index register	*load X(Rind), Ri*	*Ri ← M[Rind+X]*
Relative	Alamat operand adalah jumlah dari nilai index dan data dari Program Counter	*load X(PC), Ri*	*Ri ← M[PC+X]*
Autoincrement	Alamat dari operand berada pada register yang nilainya ditambah satu setelah instruksi dijalankan	*load (Rauto)+,Ri*	*Ri ← M[Rauto]* *Rauto ← Rauto+1*
Autodecrement	Alamat dari operand berada pada register yang nilainya dikurang satu sebelum instruksi dijalankan	*load –(Rauto),Ri*	*Rauto ← Rauto-1* *Ri ← M[Rauto]*

Tabel 10.2. Rangkuman mode pengalamatan