2. TEORI PENUNJANG

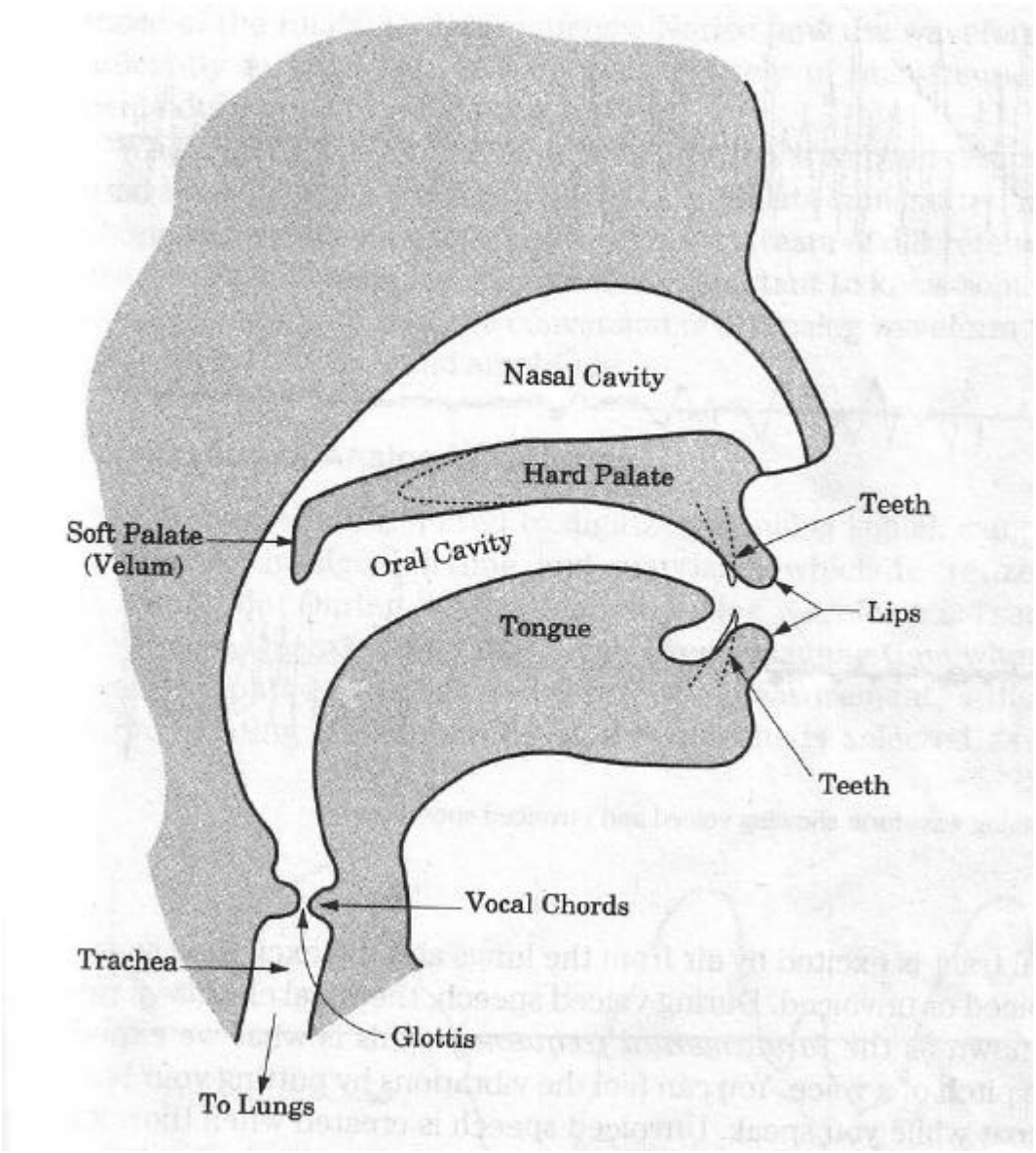
2.1. Sinyal Suara Manusia

Ketika seseorang berbicara, udara yang terkumpul di dalam paru-paru didorong keluar melalui organ-organ suara ke mulut sebagai sebuah gelombang suara. Variasi yang terjadi dari jumlah udara yang dikeluarkan, dan bentuk-bentuk konfigurasi dari organ-organ suara membentuk suatu kata tertentu. Gelombang ini disebut sebagai gelombang akustik. Untuk merekam suara seseorang maka diperlukan sebuah konverter untuk mengubah sinyal suara ini menjadi sinyal listrik analog, konverter itu disebut sebagai mikrofon. Sinyal listrik yang dihasilkan oleh sebuah mikrofon mempunyai variasi amplitudo sepanjang waktu dan inilah yang disebut sebagai sinyal analog atau gelombang analog.

Gelombang analog dari suara manusia terdiri dari banyak informasi yang membuat telinga dan otak dapat membedakan karakteristik tiap-tiap suara yang dikeluarkan. Banyak karakteristik ini dapat dideteksi dan digunakan oleh mesin selama proses analisa dan sintesa suara manusia. Karena hal ini sangat membantu dalam mengenali beberapa hal yang penting mengenai bagaimana sinyal tersebut dihasilkan, dan apa artinya.

Gambar 2.1 menunjukkan organ-organ suara pada manusia. Proses berbicara dimulai dari bawah, sebagaimana udara yang bertekanan tinggi keluar dari paru-paru menyebabkan pita suara bergetar. Efek dari getaran ini akan membebaskan semburan udara secara berulang-ulang ke lubang mulut atau lubang hidung atau keduanya dengan frekuensi tertentu tergantung dari getaran yang dihasilkan oleh pita suara. Kemudian semburan udara tersebut diteruskan sehingga keluar melalui bibir. Pada saluran antara lubang mulut dan lubang hidung terdapat katup yang disebut *velum*. *Velum* inilah yang mengatur jumlah udara yang akan dikeluarkan melalui masing-masing lubang. Kemudian lidah, rahang, gigi dan bibir digunakan untuk mengatur bentuk lubang pada mulut sehingga dapat menimbulkan suara yang diinginkan. Kemudian suara yang dihasilkan melalui mulut ini keluar sebagai partikel yang berjalan di udara dengan

kecepatan suara. Amplitudo dari sinyal ini pada sebuah tempat tertentu diukur melalui kepadatan molekul udara dan menjadi semakin lemah pada jarak yang semakin jauh. Ketika gelombang ini menyentuh telinga, gelombang ini akan diterjemahkan sebagai rangkaian warna nada, dan keras tidaknya suara.



Sumber: Pelton, Gordon E. Voice Processing. Singapore: McGraw-Hill, Inc., 1993., hal. 30.

Gambar 2.1. Organ Suara Manusia

Udara yang bertekanan tinggi bergerak dari paru-paru menuju ke organ vokal dan merinteraksi dengan berbagai macam halangan baik itu lidah, gigi, bibir, dan langit-langit mulut. Sebagian kecil dari energi yang dipancarkan ini diserap oleh halangan-halangan ini, sisanya dipantulkan. Pemantulan ini terjadi di

segala arah sehingga beberapa bagian gelombang sempat dipantulkan beberapa kali di dalam lubang mulut, bertabrakan dengan bagian gelombang yang lain, dan akhirnya keluar melalui mulut. Beberapa bagian lainnya akan menimbulkan resonansi di dalam organ vokal tergantung dari frekuensi dan bentuk lubang mulut pada saat itu. Oleh karena beberapa alasan inilah maka berarti gelombang suara yang keluar dari pita suara diperlemah oleh hal-hal tersebut bukan diperkuat.

Organ vokal diaktifkan oleh udara dari dalam paru-paru dan sumbernya bisa jadi suara *voiced* atau *unvoiced*. Suara *voiced* adalah suara yang terdengar saat huruf vokal (a, i, u, e, o) diucapkan, sedangkan *unvoiced* adalah suara yang terdengar saat huruf konsonan diucapkan. Pada saat suara *voiced* diucapkan maka pita suara bergetar dengan frekuensi tertentu yang biasa disebut sebagai frekuensi fundamental. Frekuensi inilah yang menentukan tinggi rendahnya suara yang terdengar. Getaran yang ditimbulkan oleh pita suara dapat dirasakan dengan cara meletakkan tangan di leher tempat pita suara. Pada saat suara *unvoiced* diucapkan, pita suara sama sekali tidak bergetar. Yang keluar dari mulut hanyalah udara yang telah dimodifikasi oleh organ vokal sehingga timbul aksen tertentu. Jadi jika sebah kata diucapkan maka suara yang keluar dari mulut merupakan gabungan dari suara *voiced* dan suara *unvoiced* dengan berbagai macam frekuensi dan intensitasnya. Sinyal suara yang kompleks ini kemudian diambil olah mikrofon untuk diubah menjadi sinyal listrik analog.

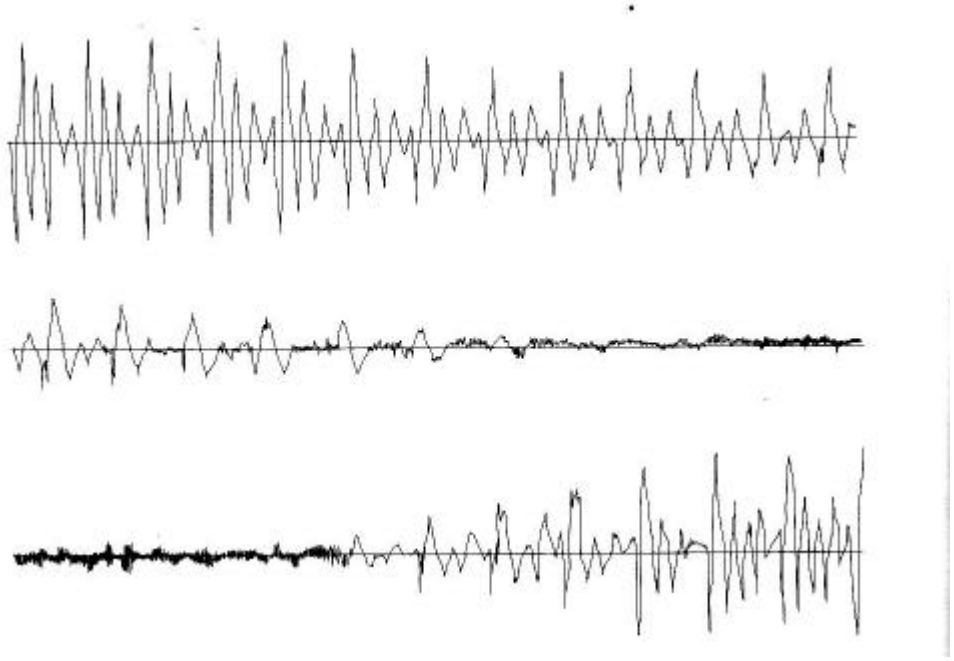
Gambar 2.2 menggambarkan gelombang suara analog dari sebuah kata yang mengandung *voiced* dan *unvoiced*. Bagian *voiced* terlihat berupa sinyal yang berulang namun tidak sama persis. Pada gambar 2.2 tersebut suara *voiced* ditunjukkan pada bagian awal dan bagian akhir. Selain itu dapat dilihat bagaimana sinyal *voiced* berkurang secara drastis pada saat mendekati sinyal *unvoiced* di mana sinyal *unvoiced* lebih banyak terlihat seperti sinyal gangguan atau *noise*.

2.2. Pengenalan Suara Manusia dengan Menggunakan Mesin

Dalam mengenali suara manusia, sebuah alat atau mesin akan berusaha merubah sinyal suara tersebut menjadi sebuah pola yang sekuensial atau beraturan, hal ini tergantung dari bagaimana sinyal tersebut dibentuk. Secara

umum, pengenalan suara manusia dengan menggunakan mesin dibagi menjadi tiga yaitu:

1. Pendekatan akustik dan fonem.
2. Pendekatan dengan pengenalan pola.
3. Pendekatan dengan kecerdasan buatan.



Sumber: Pelton, Gordon E. Voice Processing. Singapore: McGraw-Hill, Inc., 1993., hal. 33.

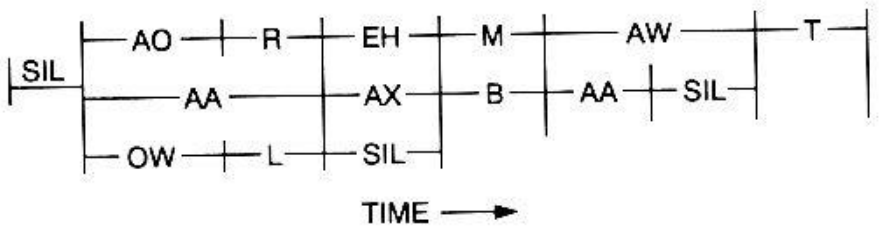
Gambar 2.2. Sinyal Analog dari Sebuah Kata

2.2.1. Pendekatan akustik dan fonem

Pendekatan akustik dan fonem ini berdasarkan pada sebuah teori yang mengatakan bahwa sebuah fonem dapat direpresentasikan menjadi sebuah spktrum yang memiliki ciri-ciri tertentu. Meskipun sebenarnya properti akustik dari sebuah fonem merupakan sebuah variabel yang sangat besar perbedaannya namun dapat diasumsikan bahwa dengan beberapa aturan yang memimpin tingkat variabel dari hal itu maka properti akustik tersebut dapat dipelajari dan diaplikasikan pada situasi yang sebenarnya. Langkah pertama dalam pendekatan akustik dan fonem ini adalah pembagian menjadi segmen-segmen tertentu dan memberi label untuk tiap segmen tersebut. Hal ini perlu dilakukan karena melibatkan pembagian sinyal suara menjadi daerah-daerah diskrit (dalam domain

waktu) di mana setiap segmen terdapat properti akustik yang dapat mewakili sebuah unit fonem atau lebih.

Untuk menggambarkan langkah-langkah dalam pendekatan akustik dan fonem maka perlu dilihat pada gambar 2.3 yang menunjukkan penggunaan fonem *lattice* dalam pembagian dan pemberian label pada sebuah kata. Yang menjadi masalah adalah bagaimana caranya merubah fonem *lattice* tersebut menjadi sebuah kata. Pada gambar 2.3 tersebut lambang SIL artinya keadaan sunyi. Dengan pencarian yang cukup tinggi maka ditemukan bahwa SIL-AO-L-AX-B-AW-T adalah sebuah kalimat "all about" dengan kemungkinan fonem yang lain seperti L, AX, dan B merupakan pilihan kedua dan ketiga. Inilah yang menjadi kesulitan utama dalam menggunakan metode ini biasanya diambil pilihan pertama.



Sumber: Rabiner, Lawrence, and Biing-Hwang Juang. Fundamentals of Speech Recognition. New Jersey: Prentice Hall, 1993., hal 43.

Gambar 2.3. Fonem *Lattice* dari sebuah kata

2.2.2 Pendekatan dengan pengenalan pola

Pada cara ini maka sebuah sistem pengenalan suara manusia pada dasarnya membandingkan secara langsung sebuah pola suara dengan suara yang baru tanpa perlu mengubah atau mengatur sinyal yang masuk. Metode ini pada umumnya terdiri dari dua tahap yaitu pelatihan pola suara dan pengenalan suara melalui pola yang telah dibentuk melalui pelatihan tersebut. Pengetahuan mesin mengenai jenis-jenis pola yang dipakai, semuanya dimasukkan dalam algoritma pelatihannya. Yang paling penting adalah prosedur pelatihannya, prosedur ini harus bisa mengkategorikan pola-pola suara tersebut. Kemudian setelah pola-pola tersebut terbentuk maka kemudian tahap pengenalan dari sebuah kata akan

dilakukan dengan cara membandingkan sinyal yang baru dengan pola yang telah dibentuk lewat pelatihan kemudian keputusan akan diambil berdasarkan kemiripan pola sinyal yang baru tersebut.

Pengenalan kata dengan menggunakan metode pendekatan dengan pengenalan pola adalah metode yang paling banyak dipakai. Hal ini terjadi karena adanya beberapa alasan yaitu:

1. Penggunaannya yang sederhana. Metode ini sangat mudah untuk dimengerti karena mengandung banyak teori matematika untuk setiap prosedur/bagian yang digunakan baik dalam tahap pelatihan maupun tahap pengenalan.
2. Ketahanan dan sedikit perubahan yang terjadi terhadap kosakata baru, pengguna baru, algoritma perbandingan dan pengambilan keputusan. Properti inilah yang menjadi pertimbangan utama banyak perancang sistem pengenalan suara.

2.2.3. Pendekatan dengan kecerdasan buatan

Metode ini merupakan pengembangan dari kedua metode sebelumnya yaitu pendekatan akustik dan fonem, dan pendekatan dengan pengenalan pola. Metode ini berusaha mengaplikasikan cara manusia mengenali sebuah kata atau kalimat menjadi sebuah mesin. Namun implementasi dari metode ini sangat sulit karena biasanya digunakan dalam sebuah sistem yang kompleks. Selain itu informasi yang dibutuhkan dalam prosesnya jauh lebih banyak dibandingkan dengan metode akustik dan fonem sehingga dibutuhkan waktu yang lebih lama terutama dalam hal pelatihannya. Biasanya penggunaan kecerdasan buatan ini bukan hanya untuk sekedar mengenali suara atau kata atau kalimat melainkan juga untuk mengenali intonasi serta susunan kata dalam sebuah kalimat. Sebenarnya cara ini merupakan cara yang paling baru oleh karena itu masih banyak masalah yang timbul namun mempunyai prospek yang bagus.

2.3. *Fast Fourier Transform*

Tranformasi Fourier berguna untuk mentranformasikan informasi dari domain waktu ke domain frekuensi dalam sistem waktu kontinyu. Tranformasi Fourier ini merupakan alat yang sangat berguna di berbagai bidang seperti optik,

seismologi, telekomunikasi, pemrosesan sinyal, dan pemrosesan gambar. Dalam sistem waktu diskrit, tranformasi fourier ini biasa disebut sebagai *Discrete Fourier Transform* (DFT). Karena perhitungan dari DFT ini sangat kompleks dan lama maka sangat sedikit aplikasi yang dibuat dengan menggunakan DFT ini meskipun pada jaman sekarang ini. Kemudian dari DFT ini dikembangkan lebih lanjut menjadi *Fast Fourier Transform* (FFT) yang memiliki algoritma untuk mengubah perhitungan DFT menjadi sangat efisien dan FFT inilah yang sekarang sering digunakan dalam pemrosesan sinyal digital saat ini.

2.3.1. Pengembangan dari algoritma *Discrete Fourier Tranform* (DFT)

DFT adalah bentuk waktu diskrit dari Tranformasi Fourier. Tranformasi Fourier pada sistem waktu kontinyu untuk sinyal analog x(t) dapat ditulis sebagai:

$$X(w) = \int_{-\infty}^{\infty} x(t).e^{-j.w.t}dt \qquad (2.1)$$

di mana, pada umumnya, kedua fungsi x(t) dan X(ω) merupakan fungsi kompleks dari variabel t dan variabel ω. Sinyal kontinyu x(t) diubah menjadi sinyal diskrit x(nT) dengan cara mengambil sampling setiap T detik. Jika T tidak dituliskan maka sinyal diskrit dapat ditulis dengan notasi x(n) maka persamaan tranformasi fourier untuk sinyal diskrit dapat ditulis sebagai:

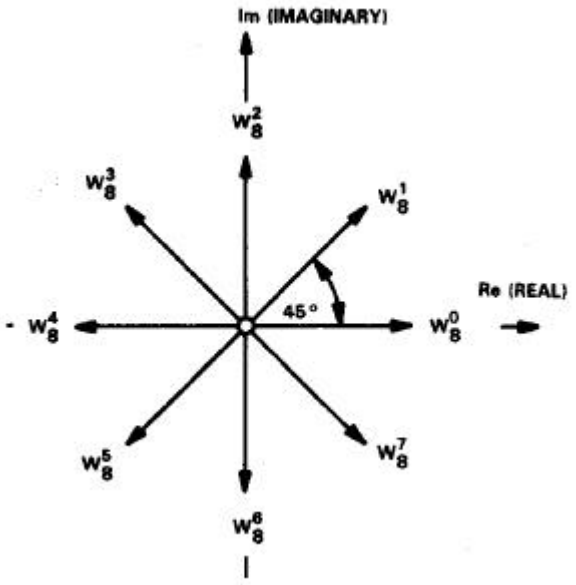
$$X(w) = \sum_{-\infty}^{\infty} x(n).e^{-j.w.n} \qquad (2.2)$$

dimana ω melambangkan frekuensi yang telah dinormalisasi antara 0 dan 2π. X(ω) periodik dengan periode 2π sehingga nilainya hanya berkisar antara 0 dan 2π. X(ω) menjadi periodik merupakan akibat langsung dari x(n) yang disampling secara periodik pula.

Asumsikan bahwa sebuah sinyal x(n) terdiri dari N sampel, karena tidak ada batasan interval dari N maka diasumsikan sinyal tersebut merupakan sinyal yang periodik (berulang setiap periode waktu tertentu). Karena asumsi inilah maka transformasi fourier dalam sisterm waktu diskrit dapat ditulis sebagai:

$$X(k) = \sum_{n=0}^{N-1} x(n).W_N^{nk} \qquad \text{k=0,1,2,…,N-1} \qquad (2.3)$$

dimana $W_N = e^{-j2\pi / N}$, dan W_N biasa disebut sebagai *twiddle factor*. Persamaan (2.3) disebut sebagai DFT sebanyak N buah sampel. Karena perhitungannya menggunakan bilangan kompleks maka operasi penjumlahan maupun perkalian membutuhkan kira-kira N^2 untuk N poin maka total operasi aritmatik yang dibutuhkan untuk nilai N tertentu semakin lama semakin besar seiring bertambahnya nilai N. Oleh karena itu dibutuhkan metode alternatif untuk dapat menghitung DFT secara efisien. Metode untuk mengefisienkan perhitungan DFT ini biasanya berdasarkan sifat simetris dan periodik dari *twiddle factor*. Hal ini dapat dilihat pada gambar 2.4 yang menunjukkan sifat simetris dan periodik dari *twiddle factor* untuk N=8.



Sumber: Papamichalis, Panos. Implementation of Fast Fourier Transform Algorithms with the TMS32020. Texas Instruments, Inc., 1989., hal 8.

Gambar 2.4. *Twiddle Factor* untuk N=8

Dari gambar 2.4 dapat dilihat bahwa:

$$W_N^k = -W_N^{k+(N/2)} \tag{2.4}$$

$$W_N^k = W_N^{N+k} \tag{2.5}$$

persamaan (2.4) menunjukkan sifat simetris dari *twiddle* factor sedangkan persamaan (2.5) menunjukkan sifat periodiknya.

2.3.2. Algoritma *Fast Fourier Transform* FFT

Metode yang lebih efisien untuk menghitung DFT biasa disebut algoritma *decimation in time* (DIT) FFT. Metode ini dapat mengurangi jumlah operasi aritmetik secara signifikan. Dengan menggunakan FFT, maka nilai N harus merupakan nilai tertentu. Ukuran terkecil dari sebuah perhitungan FFT biasanya disebut dengan *radix*. Pada sebuh FFT *radix*-2 maka nilai N terkecil yang dapat dihitung adalah 2. Karena menggunakan *radix*-2 maka nilai N harus merupakan bilangan pangkat dari 2 ($N=2^a$).

Jumlah operasi aritmatik yang dibutuhkan dapat dikurangi dengan mengubah N poin DFT menjadi dua buah N/2 poin DFT. Ini berarti mengubah sinyal input x(n) menjadi dua buah N/2 sinyal (yang diubah menjadi 2 bagian adalah x(n) bukan X(k) karena metode yang akan digunakan adalah DIT FFT). Maka persamaan (3) dapat diubah menjadi:

$$X(k) = \sum_{n=0}^{N/2-1} x(2n).W_N^{2nk} + \sum_{n=0}^{N/2-1} x(2n+1).W_N^{(2n+1)k} \qquad (2.6)$$

$$= \sum_{n=0}^{N/2-1} x(2n).W_N^{2nk} + W_N^k . \sum_{n=0}^{N/2-1} x(2n+1).W_N^{2nk}$$

dimana:

$$W_N^2 = W_{N/2}$$

maka persamaan (6) dapat ditulis sebagai:

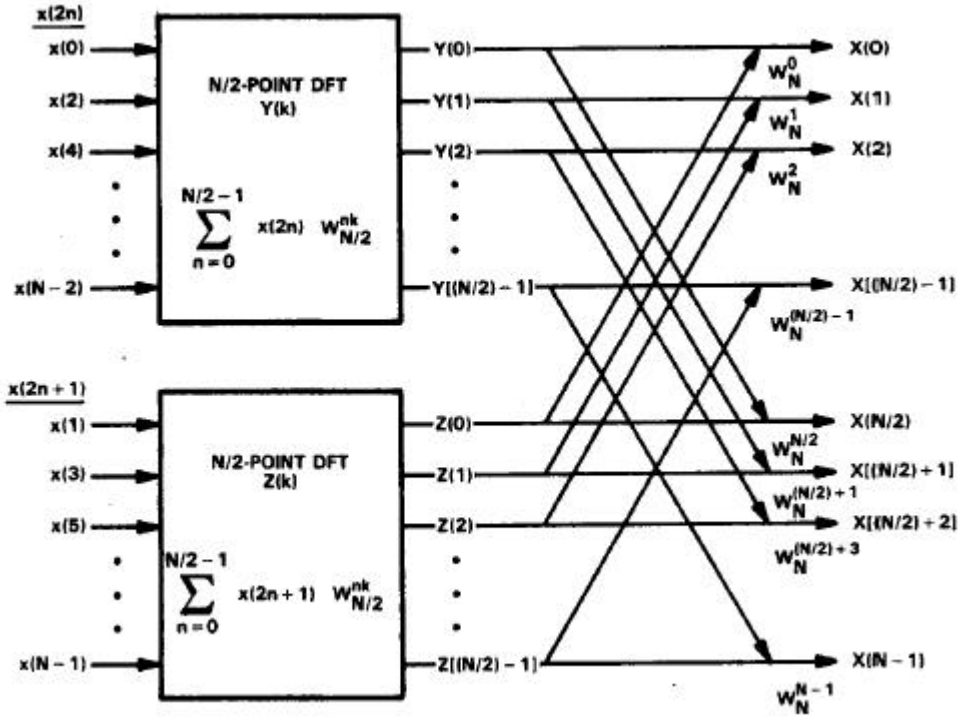
$$X(k) = \sum_{n=0}^{N/2-1} x(2n).W_{N/2}^{nk} + W_N^k . \sum_{n=0}^{N/2-1} x(2n+1).W_{N/2}^{nk}$$

$$= Y(k) + W_N^k .Z(k) \qquad \text{k=0,1,…,N-1} \qquad (2.7)$$

Dimana Y(k) adalah penjumlahan yang pertama dan Z(k) adalah penjumlahan yang kedua.

Y(k) dan Z(k) selanjutnya dilihat sebagai N/2 poin DFT dari nomor ganjil dan nomor genap. Dengan demikian maka jumlah operasi aritmatik yang dilakukan kira-kira $N+2(N/2)^2$ karena menurut persamaan (2.7), N poin DFT telah dipisah menjadi N/2 poin DFT , yang kemudian digabungkan oleh sejumlah N penjumlahan dan perkalian bilangan kompleks. Reduksi operasi aritmatika ini dapat dilihat pada gambar 2.5.

Melihat persamaan di atas bahwa sinyal X(k), Y(k), dan Z(k) merupakan sinyal yang periodik dalam k dengan periode sebesar N, dimana Y(k) dan Z(k) mempunyai periode sebesar N/2 dengan nilai k antara 0 dan N/2-1. Sifat periodik dari Y(k) dan Z(k) juga dapat dilihat pada gambar 2.5.



Sumber: Papamichalis, Panos. Implementation of Fast Fourier Transform Algorithms with the TMS32020. Texas Instruments, Inc., 1989., hal 8.

Gambar 2.5. Dekomposisi DIT Dari Sebuah N Poin DFT

Meskipun persamaan (2.7) dapat digunakan untuk menghitung X(k) untuk $0 \le k \le N-1$, reduksi operasi aritmatika lebih lanjut masih dapat dilakukan jika menggunakan sifat simetris dan sifat periodik dari *twiddle factor* seperti pada persamaan (2.4) dan (2.5) dalam perhitungan X(k) secara terpisah antara $0 \le k \le (N/2)-1$ dan $N/2 \le k \le N-1$. Persamaan (2.7) untuk $N/2 \le k \le N-1$ dapat ditulis sebagai:

$$X(k+N/2) = \sum_{n=0}^{N/2-1} x(2n).W_{N/2}^{n.(k+n/2)} + W_N^{n.(k+N/2)}.\sum_{n=0}^{N/2-1} x(2n+1).W_{N/2}^{n.(k+N/2)} \qquad (2.8)$$

dimana:

$$W_{N/2}^{n(k+N/2)} = W_{N/2}^{nk}$$

$$W_N^{k+N/2} = -W_N^k$$

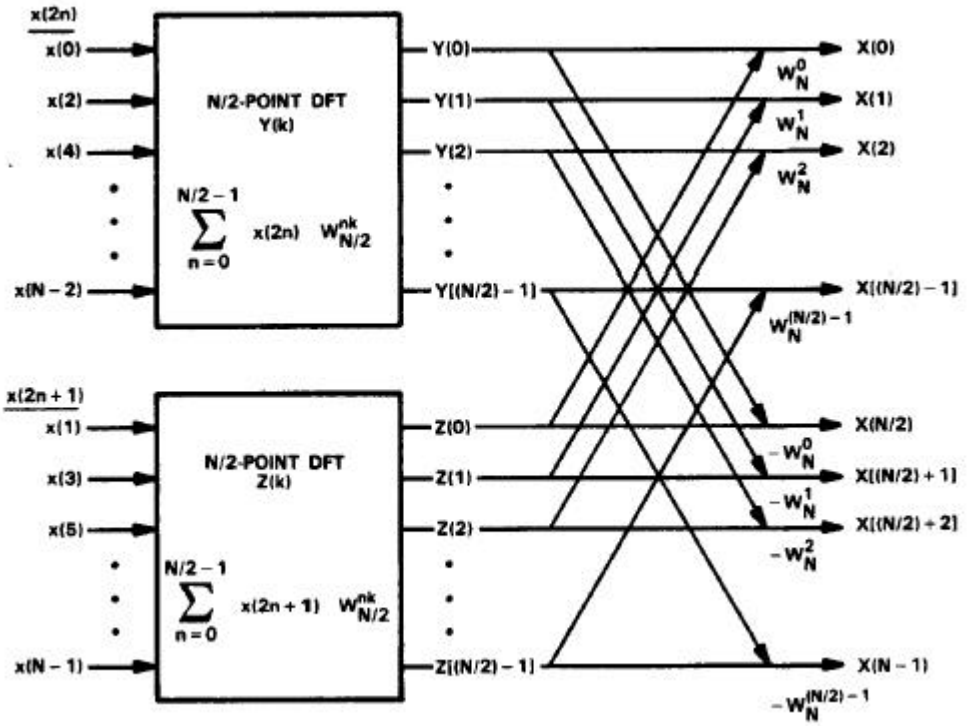
Maka persamaan (2.8) dapat ditulis sebagai:

$$X(k+N/2) = \sum_{n=0}^{N/2-1} x(2n).W_{N/2}^{nk} - W_N^k . \sum_{n=0}^{N/2-1} x(2n+1).W_{N/2}^{nk} \qquad (2.9)$$

$$= Y(k) - W_N^k . Z(k) \qquad \text{k=0,1,…,(N/2)-1}$$

Jadi dengan demikian, persamaan (2.7) dapat digunakan untuk menghitung setengah spektrum frekuensi pertama dari X(k) untuk $0 \le k \le (N/2)-1$, sedangkan persamaan (2.9) digunakan untuk menghitung setengah spektrum frekuensi kedua dari X(k+N/2).

Gambar 2.6 menunjukkan keadaan ketika sifat simetris dari *twiddle factor* digunakan dalam perhitungan X(k). Dengan menggunakan proses di atas maka perhitungan DFT dapat direduksi lebih banyak lagi. Dalam perhitungan lebih lanjut maka perhitungan FFT ini akan dibagi lagi menjadi 4 buah perhitungan N/4 dan kemudian 8 buah N/8 dan seterusnya hingga pada akhirnya tersisa hanya 2 buah poin DFT atau yang biasa disebut dengan *butterflies*.



Sumber: Papamichalis, Panos. Implementation of Fast Fourier Transform Algorithms with the TMS32020. Texas Instruments, Inc., 1989., hal 8.

Gambar 2.6. Dekomposisi DFT Menggunakan Sifat Simetris

2.4. *Linear Predictive Coding* (LPC)

Teori mengenai *Linear Predictive Coding* (LPC) yang diaplikasikan dalam *speech* (suara manusia), telah diteliti dengan baik selama bertahun-tahun. Dalam bagian ini hanya akan dijelaskan mengenai aplikasi LPC dalam sistem pengenalan kata saja. Sebelum menjelaskan mengenai LPC lebih jauh, ada baiknya diketahui keunggulan dari penggunaan LPC, yaitu:

- LPC memberikan model yang baik dari suatu sinyal suara manusia. Hal ini disebabkan karena model LPC pada semua kutub (*all-pole*) memberikan perkiraaan yang baik terhadap amplop spektral. Pada daerah suara *unvoiced* dan peralihan, LPC kurang efektif dibandingkan dengan daerah suara *voiced* tetapi LPC tetap memberikan hasil yang cukup baik jika digunakan untuk proses pengenalan suara manusia.
- Aplikasi LPC pada analisis sinyal suara manusia, memberikan hasil yang cukup baik. Sebagai hasilnya, representasi dari karakteristik vokal menjadi bisa direalisasikan.
- LPC adalah sebuah model yang bisa dikendalikan. Metode LPC adalah sebuah metode yang secara matematika akurat dan sederhana untuk diimplementasikan baik itu secara perangkat lunak maupun perangkat keras. Selain itu perhitungan yang digunakan dalam LPC tidaklah serumit metode *filter-bank*.
- Model LPC bekerja dengan baik dalam berbagai aplikasi pengenalan. Beberapa percobaan telah menunjukkan bahwa pengenal suara manusia dengan menggunakan metode ini memberikan hasil yang sama baiknya dengan model *filter-bank* atau bahkan lebih baik.

Berdasarkan pertimbangan-pertimbangan di atas maka LPC sebagai proses *front-end* telah digunakan pada banyak alat pengenalan suara manusia.

2.4.1. Model LPC

Ide dasar dari penggunaan LPC adalah bahwa sebuah sinyal suara manusia pada waktu *n*, *s(n)*, dapat diperkirakan sebagai kombinasi linier dari *p* sinyal suara sebelumnya, yaitu:

$$s(n) \approx a_1.s(n-1) + a_2.s(n-2) + ... + a_p.s(n-p) \qquad (2.10)$$

dimana koefisien $a_1, a_2, ..., a_p$ merupakan asumsi konstanta dari sebuah bagian dari analisa suara manusia. Persamaan (2.10) dapat diubah dengan menambahkan sebuah exitasi *G.u(n)*:

$$s(n) = \sum_{i=1}^{p} a_i .s(n-i) + G.u(n) \tag{2.11}$$

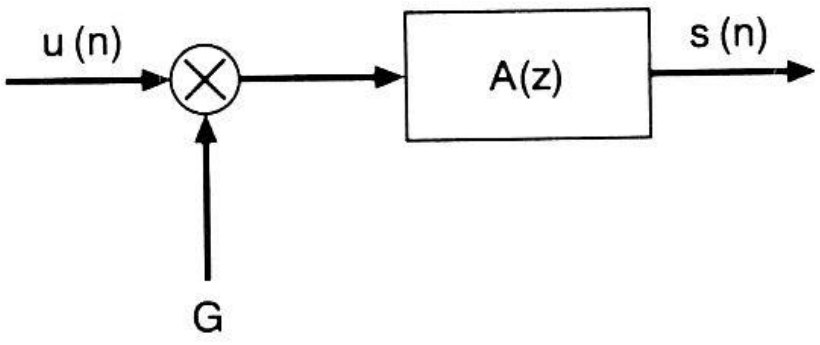
dimana *u(n)* adalah eksitasi yang dinormalisasi dan *G* adalah penguatan dari aksitasi tersebut. Jika persamaan (2.11) ini diekspresikan dalam domain z maka persamaan tersebut dapat diubah menjadi:

$$S(z) = \sum_{i=1}^{p} a_i .z^{-i} .S(z) + G.U(z) \tag{2.12}$$

Persamaan (2.12) dapat diubah menjadi fungsi transfer:

$$H(z) = \frac{S((z)}{G.U(z)} = \frac{1}{1 - \sum_{i=1}^{p} a_i .z^{-i}} = \frac{1}{A(z)} \tag{2.13}$$

Interpretasi dari persamaan (2.13) dapat dilihat pada gambar 2.7 yang menunjukkan sinyal eksitasi yang telah dinormalisasi *u(n)* dengan penguatan sebesar *G* yang berfungsi sebagai input dari sistem *H(z)* yang akan menghasilkan sinyal suara manusia *s(n)*.

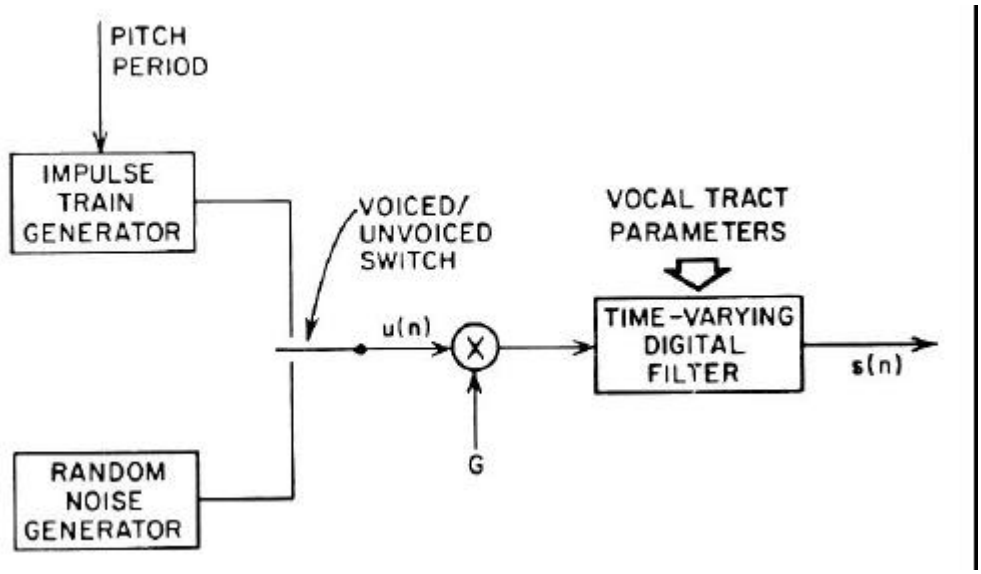


Sumber: Rabiner, Lawrence, and Biing-Hwang Juang. Fundamentals of Speech Recognition. New Jersey: Prentice Hall, 1993., hal. 100.

Gambar 2.7. Model LPC Untuk Sinyal Suara

Berdasarkan hasil percobaan diketahui bahwa fungsi eksitasi untuk suara sebenarnya merupakan barisan pulsa (untuk suara *voiced*) atau *noise* yang acak (untuk suara *unvoiced*). Jadi sebenarnya model untuk sintesa sinyal suara manusia dapat dilihat pada gambar 2.8. Pada gambar 2.8, sumber eksitasinya dipilih oleh

sebuah saklar yang posisinya diatur oleh karakter *voiced/unvoiced* dari suara manusia yang akan memilih antara barisan pulsa (*impulse train*) atau *noise* yang acak (*random noise*).



Sumber: Rabiner, Lawrence, and Biing-Hwang Juang. Fundamentals of Speech Recognition. New Jersey: Prentice Hall, 1993., hal. 101.

Gambar 2.8. Model Sintesa Suara Manusia Berdasarkan Model LPC

2.4.2. Analisa Persamaan LPC

Berdasarkan model pada gambar 1, hubungan yang tepat antara *s(n)* dan *u(n)* adalah:

$$s(n) = \sum_{k=1}^{p} a_k . s(n-k) + G.u(n) \tag{2.14}$$

Kombinasi linier dari sampel sinyal sebelumnya dapat didefinisikan sebagai:

$$\tilde{s}(n) = \sum_{k=1}^{p} a_k . s(n-k) \tag{2.15}$$

Sehingga prediksi kesalahan, *e(n)*, dapat ditulis sebagai:

$$e(n) = s(n) - \tilde{s}(n) = s(n) - \sum_{k=1}^{p} a_k . s(n-k) \tag{2.16}$$

dengan fungsi transfer kesalahan:

$$A(z) = \frac{E(z)}{S(z)} = 1 - \sum_{k=1}^{p} a_k . z^{-k} \tag{2.17}$$

Dapat dilihat dengan jelas ketika *s(n)* dihasilkan dari sistem linier yang ditunjukkan dari gambar 2.7, maka prediksi kesalahan, *e(n),* akan sama dengan *G.u(n)*.

Masalah dasar dari analisa LPC ini adalah untuk menentukan koefisien prediksi a_k secara langsung dari sinyal suara manusia yang masuk sehingga properti *spectral* dari filter digital pada gambar 2.8 cocok dengan bentuk gelombang dari sinyal suara manusia dalam jendela analisanya. Karena karakteristik *spectral* dari sebuah sinyal selalu berubah dalam jangka waktu tertentu, maka koefisien prediksi a_k pada waktu tertentu, *n*, harus ditentukan dari bagian kecil dari sinyal suara manusia yang masuk. Biasanya sinyal suara yang masuk dibagi menjadi sekitar 10 mili detik untuk bisa mendapatkan analisa *spectral* yang baik.

Untuk menentukan persamaan yang harus diselesaikan dalam menentukan koefisien prediksi, maka ditentukan bahwa bagian kecil dari sinyal suara dan sinyal kesalahan pada waktu *n* adalah:

$$s_n(m)=s(n+m) \tag{2.18a}$$

$$e_n(m)=e(n+m) \tag{2.18b}$$

dan untuk meminimalkan sinyal *mean squared error* pada waktu *n* maka:

$$E_n = \sum_m e_n^2(m) \tag{2.19}$$

dengan menggunakan definisi dari $e_n(m)$ dalam fungsi $s_n(m)$ maka:

$$E_n = \sum_m \left[s_n(m) - \sum_{k=1}^{p} a_k . s_n(m-k) \right]^2 \tag{2.20}$$

Untuk menyelesaikan persamaan (2.20) untuk koefisien prediksi maka E_n didiferensialkan untuk tiap a_k dan membuat hasilnya menjadi nol.

$$\frac{\partial E_n}{\partial a_k} = 0 \qquad k=1,2,\ldots,p \tag{2.21}$$

dengan memberikan:

$$\sum_m s_n(m-i).s_n(m) = \sum_{k=1}^{p} a_k . \sum_m s_n(m-i).s_n(m-k) \tag{2.22}$$

Dengan mengetahui bahwa bentuk dari $\sum s_n(m-i).s_n(m-k)$ adalah bentuk pendek dari $s_n(m)$ dan dengan mengumpamakan:

$$f_n(i,k) = \sum_m s_n(m-i).s_n(m-k) \tag{2.23}$$

Maka persamaan (2.22) dapat diekspresikan dengan bentuk:

$$f_n(i,0) = \sum_{k=1}^{p} a_k . f_n(i,k) \tag{2.24}$$

yang menggambarkan beberapa persamaan dalam fungsi *p*. Sehingga dapat dilihat bahwa nilai minimum dari *mean squared error*, E_n, dapat ditulis sebagai:

$$E_n = \sum_m s_n^2(m) - \sum_{k=1}^{p} a_k . \sum_m s_n(m).s_n(m-k) \tag{2.25}$$

$$= f_n(0,0) - \sum_{k=1}^{p} a_k . f_n(0,k) \tag{2.26}$$

Dapat dilihat bahwa nilai minimum dari *mean squared error* terdiri dari sebuah nilai konstan dan beberapa persamaan a_k.

Untuk menyelesaikan persamaan (2.24) untuk nilai koefisien prediksi yang optimal maka harus dihitung untuk nilai *i* antara 1 dan *p*, dan nilai *k* antara 0 dan *p*. Kemudian menyelesaikan persamaan tersebut dalam fungsi *p*. Untuk menghitungnya ada 2 metode standard yang bisa digunakan.

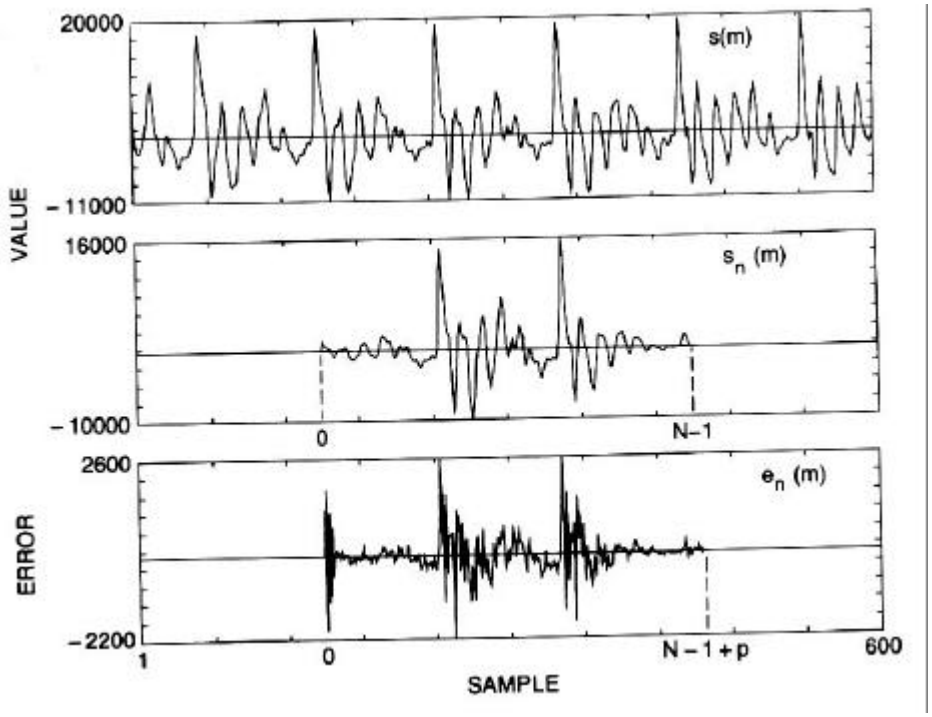
2.4.3. Metode *Autocorrelation*

Cara yang paling sederhana untuk menentukan limit dari *m* adalah dengan mengasumsikan bahwa potongan sinyal $s_n(m)$ bernilai 0 untuk interval $0 \le m \le N-1$. Hal ini berarti sama dengan mengasumsikan sebuah sinyal s(m+n) dikalikan dengan sebuah *window*, *w(m)* yang terbatas, yang memiliki nilai 0 di luar jangkauan $0 \le m \le N-1$. Sehingga persamaan sinyal suara manusia dalam jangka pendek dapat ditulis sebagai:

$$s_n(m) = \begin{cases} s(m+n).w(m), & 0 \le m \le N-1 \\ 0, & \text{lainnya} \end{cases} \tag{2.27}$$

efek dari pembebanan sinyal suara manusia oleh sebuah sinyal *window* dapat dilihat pada gambar 2.9, 2.10, dan 2.11. Pada tiap gambar, yang paling atas adalah bentuk sinyal aslinya, yang tengah merupakan gambar sinyal yang telah dibebani (dalam hal ini menggunakan fungsi pembebanan *Hamming Window*), dan yang

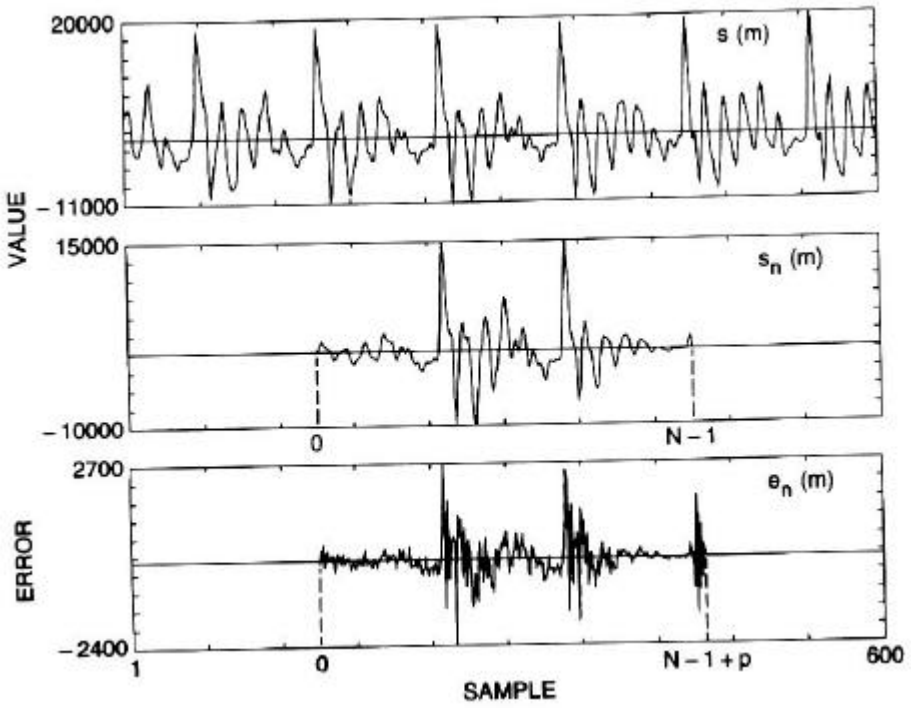
paling bawah adalah sinyal error yang dihasilkan dari parameter prediksi yang telah dihitung secara optimal.



Sumber: Rabiner, Lawrence, and Biing-Hwang Juang. Fundamentals of Speech Recognition. New Jersey: Prentice Hall, 1993., hal. 104.

Gambar 2.9. Contoh Efek Pembebanan yang Menyebabkan Prediksi Kesalahan yang Besar pada Awal Sinyal

Berdasarkan persamaan (2.27) maka untuk nilai *m* lebih kecil dari 0 maka sinyal kesalahan $e_n(m)$ bernilai nol karena sinyalnya juga nol, begitu juga untuk m lebih besar dari *N-1*. Tetapi pada daerah antara $m=0$ sampai $m=N-1$ sinyal $s_n(m)$ diprediksikan dari sampel sebelumnya di mana beberapa sampel pertama bernilai nol. Hal inilah yang menyebabkan kesalahan prediksi yang cukup besar pada awal sinyal (dapat dilihat pada gambar 2.9 bawah). Lebih jauh lagi, pada daerah $m=N-1$ (dari $m=N-1$ sampai $m=N-1+p$) juga berpotensi menghasilkan error yang cukup besar karena adanya beberapa sinyal yang bernilai nol (dapat dilihat pada gambar 2.10 bawah). Kedua prediksi kesalahan ini hanya berlaku untuk sinyal suara *voiced*. Untuk sinyal suara *unvoiced* dimana sinyal ini tidak mempunyai sinyal yang sensitif karena hanya berupa *noise* maka kedua kesalahan prediksi yang terjadi pada sinyal suara *voiced* tidak terjadi (dapat dilihat pada gambar 2.11 bawah).



Sumber: Rabiner, Lawrence, and Biing-Hwang Juang. Fundamentals of Speech Recognition. New Jersey: Prentice Hall, 1993., hal. 104.

Gambar 2.10. Contoh Efek Pembebanan yang Menyebabkan Prediksi Kesalahan yang Besar pada Akhir Sinyal

Berdasarkan sinyal yang telah dibebani pada persamaan (2.27) maka persamaan *mean squared error* menjadi:

$$E_n = \sum_{m=0}^{N-1+p} e_n^2(m) \tag{2.28}$$

dan $f_n(i,k)$ dapat ditulis menjadi:

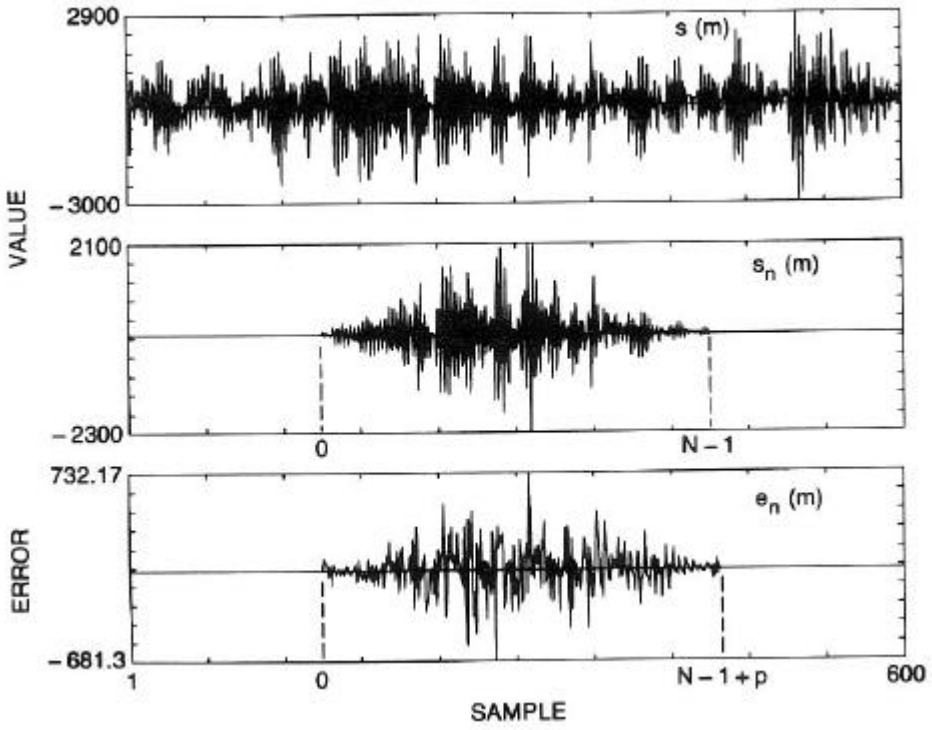
$$f_n(i,k) = \sum_{m=0}^{N-1+p} s_n(m-i).s_n(m-k) \tag{2.29}$$

atau

$$f_n(i,k) = \sum_{m=0}^{N-1-(i-k)} s_n(m).s_n(m+i-k) \tag{2.30}$$

Karena persamaan (2.30) hanya berupa fungsi (i-k) bukan fungsi I dan k secara terpisah maka fungsi *covariance,* $f_n(i,k)$, direduksi menjadi:

$$f_n(i,k) = r_n(i-k) = \sum_{m=0}^{N-1-(i-k)} s_n(m).s_n(m+i-k) \tag{2.31}$$



Sumber: Rabiner, Lawrence, and Biing-Hwang Juang. Fundamentals of Speech Recognition. New Jersey: Prentice Hall, 1993., hal. 105.

Gambar 2.11. Contoh Efek Pembebanan yang Tidak Menimbulkan Prediksi Kesalahan yang Besar pada Kedua Ujung Sinyal

Karena fungsi *autocorrelation* adalah fungsi yang simetris maka $r_n(-k)=r_n(k)$, maka persamaan LPC dapat ditulis sebagai:

$$\sum_{k=1}^{p} r_n(|i-k|).a_k = r_n(i) \qquad (2.32)$$

dan dapat digambarkan menjadi sebuah matriks:

$$\begin{bmatrix} r_n(0) & r_n(1) & r_n(2) & \Lambda & r_n(p-1) \\ r_n(1) & r_n(0) & r_n(1) & \Lambda & r_n(p-2) \\ r_n(2) & r_n(1) & r_n(0) & \Lambda & r_n(p-3) \\ \text{M} & \text{M} & \text{M} & & \text{M} \\ r_n(p-1) & r_n(p-2) & r_n(p-3) & \Lambda & r_n(0) \end{bmatrix} \bullet \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \text{M} \\ a_p \end{bmatrix} = \begin{bmatrix} r_n(0) \\ r_n(1) \\ r_n(2) \\ \\ r_n(p) \end{bmatrix} \qquad (2.33)$$

Matriks *autocorrelation* di atas biasa disebut dengan sebut matriks Toeplitz dan salah satu cara untuk menyelesaikan persamaan tersebut secara efisien adalah algoritma Durbin.

2.4.4. Metode *Covariance*

Salah satu alternatif untuk menggunakan sebuah fungsi pembebanan atau *window* untuk mendefinisikan $s_n(m)$ adalah dengan memperbaiki interval di mana *mean squared error* akan dihitung dan menggunakan sinyal aslinya secara langsung sehingga:

$$E_n = \sum_{m=0}^{N-1} e_n^2(m) \tag{2.34}$$

dengan $f_n(i,k)$ didefinisikan sebagai:

$$f_n(i,k) = \sum_{m=0}^{N-1} s_n(m-i).s_n(m-k) \tag{2.35}$$

atau dengan mengganti variabelnya:

$$f_n(i,k) = \sum_{m=0}^{N-i-1} s_n(m).s_n(m+i-k) \tag{2.36}$$

Jika diasumsikan $i=p$ maka perhitungan dari persamaan (2.36) melibatkan sinyal $s_n(m)$ dari $m=-p$ sampai $m=N-1-p$. Sedangkan untuk $k=0$ maka $s(m+I-k)$ membutuhkan sinyal dari 0 sampai $N-1$.

Dengan menggunakan interval yang berlebih dari $-p$ sampai $N-1$ maka bentuk matriks dari persamaan LPC menjadi:

$$\begin{bmatrix} f_n(1,1) & f_n(1,2) & f_n(1,3) & \Lambda & f_n(1,p) \\ f_n(2,1) & f_n(2,2) & f_n(2,3) & \Lambda & f_n(2,p) \\ f_n(3,1) & f_n(3,2) & f_n(3,3) & \Lambda & f_n(3,p) \\ \text{M} & \text{M} & \text{M} & & \text{M} \\ f_n(p,1) & f_n(p,2) & f_n(p,3) & \Lambda & f_n(p,p) \end{bmatrix} \bullet \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \text{M} \\ a_{p1} \end{bmatrix} = \begin{bmatrix} f_n(1,0) \\ f_n(2,0) \\ f_n(3,0) \\ \text{M} \\ f_n(p,0) \end{bmatrix} \tag{2.37}$$

Hasil dari matriks *covariance* ini juga simetris karena $f_n(i,k) = f_n(k,i)$ tapi bukan sebuah matriks Toeplitz, dan matriks ini dapat diselesaikan dengan menggunakan sebuah teknik yang disebut *Cholesky decomposition*. Karena bentuk ini jarang digunakan dalam aplikasi pengenalan suara manusia maka metode ini tidak akan dibahas lebih lanjut.

2.4.5. Prosesor LPC Untuk Pengenalan Suara Manusia

Untuk mengetahui lebih lanjut mengenai metode LPC maka akan dijelaskan mengenai prosesor *front-end* LPC yang biasa digunakan dalam aplikasi

pengenalan suara manusia. Gambar 2.12 menunjukkan blok diagram dari prosesor LPC. Langkah dasar yang harus dilakukan mengenai blok diagram tersebut adalah:

1. *Preemphasis*

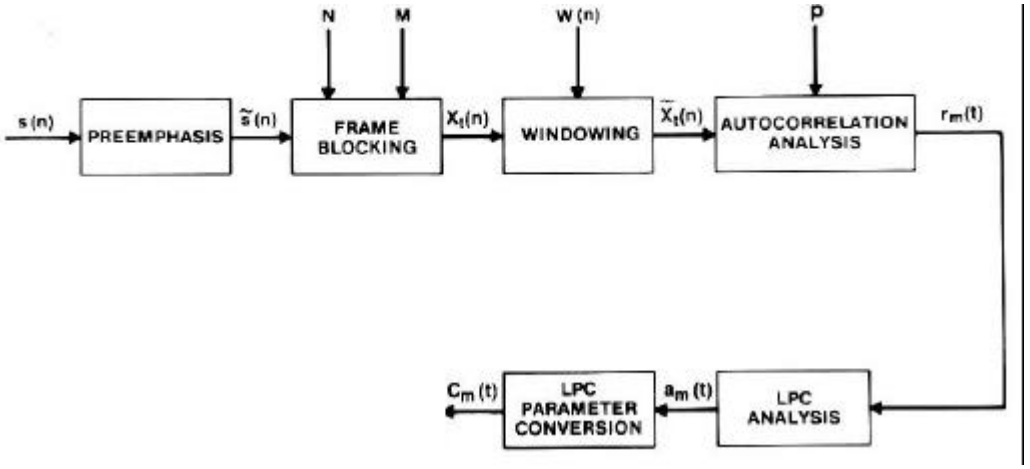
   Sinyal suara yang telah diubah menjadi sinyal digital, *s(n)*, dilewatkan pada sebuah filter yang berorde rendah (biasanya menggunakan filter FIR orde satu) untuk meratakan sinyal dan membuatnya lebih tahan terhadap akibat yang ditimbulkan pada pemrosesan sinyal tingkat berikutnya. Rangkaian preemphasis yang paling sering digunakan adalah sebuah sistem orde satu:

   $$H(z) = 1 - \tilde{a}.z^{-1}, \qquad 0.9 \le a \le 1 \tag{2.38}$$

   Dengan demikian maka output dari rangkaian filter tersebut, $\tilde{s}(n)$, dalam fungsi input, *s(n)*, adalah:

   $$\tilde{s}(n) = s(n) - \tilde{a}.s(n-1) \tag{2.39}$$

   Nilai yang paling sering digunakan untuk $\tilde{a}$ adalah 0.95. Namun untuk implementasi dalam *fixed point* biasanya nilai 15/16 juga sering digunakan. Gambar 2.13 menunjukkan renpon frekuensi untuk *H(z)* dengan nilai *a* sebesar 0.95.



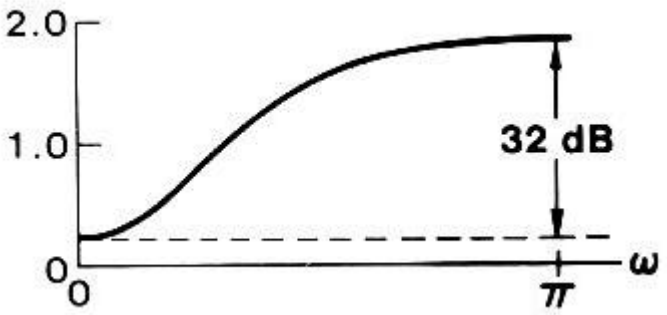
Sumber: Rabiner, Lawrence, and Biing-Hwang Juang. Fundamentals of Speech Recognition. New Jersey: Prentice Hall, 1993., hal. 113. (telah diolah kembali)

Gambar 2.12. Blok Diagram Prosesor LPC

2. *Frame Blocking*

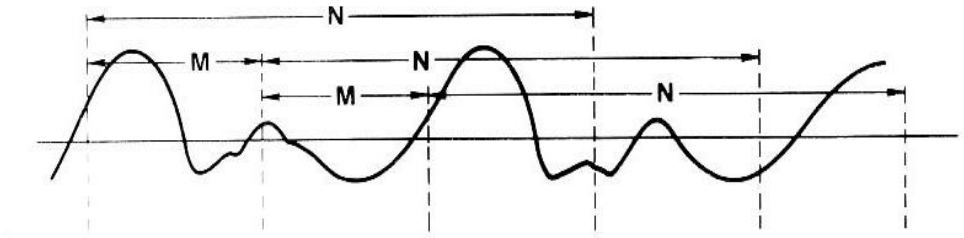
   Pada tahap ini, sinyal yang telah di-*preemphasis*, diblok menjadi beberapa bagian dengan jumlah sampel *N*, dan tiap bagian dipisahkan dengan sejumlah

M sampel. Gambar 2.14 menunjukkan penggambaran sinyal yang telah diblok dengan nilai *M*=(1/3)*N*. Bagian pertama terdiri dari sejumlah N sampel, kemudian bagian kedua dimulai dari sampel ke M juga sejumlah N sampel dan begitu seterusnya. Dengan demikian ada sinyal yang *overlap* dari setiap bagian sinyalnya. Hal ini memberikan hasil spektrum LPC yang akan berkorelasi tiap bagiannya.



Sumber: Rabiner, Lawrence, and Biing-Hwang Juang. Fundamentals of Speech Recognition. New Jersey: Prentice Hall, 1993., hal. 113.

Gambar 2.13. Respon Frekuensi dari Rangkaian Preemphasis untuk *a*=0.95



Sumber: Rabiner, Lawrence, and Biing-Hwang Juang. Fundamentals of Speech Recognition. New Jersey: Prentice Hall, 1993., hal. 114. (telah diolah kembali)

Gambar 2.14. Pembentukan Sinyal Menjadi Bagian-bagian yang O*verlap*

3. *Windowing*

Langkah berikutnya adalah melakukan proses *window* pada tiap bagian sinyal yang telah dibuat dari tingkat sebelumnya. Hal ini dilakukan untuk meminimalkan diskontinyuitas pada bagian awal sinyal dan bagian akhir sinyal. Jika didefinisikan sebuah *window w(n)* dan sinyal tiap bagian adalah *x(n)* maka sinyal hasil dari proses *windowing* ini adalah:

$$\tilde{x}(n) = x(n).w(n), \qquad 0 \le n \le N-1 \tag{2.40}$$

Model *window* yang paling sering digunakan untuk model LPC dengan metode autokorelasi adalah *Hamming Window* (Sumber: Rabiner, Lawrence, and Biing-Hwang Juang. Fundamentals of Speech Recognition. New Jersey: Prentice Hall, 1993., hal. 114) yang mempunyai bentuk:

$$w(n) = 0.54 - 0.46 \cdot \cos\left(\frac{2 \cdot \pi \cdot n}{N-1}\right) \tag{2.41}$$

4. Analisa autokorelasi

   Tiap bagian sinyal yang telah diberi *window* kemudian akan dibentuk autokorelasinya:

$$r(m) = \sum_{n=0}^{N-1-m} \tilde{x}(n) \cdot \tilde{x}(n+m), \qquad m=0, 1, \ldots, p \tag{2.42}$$

   dimana nilai tertinggi dari autokorelasi tersebut, *p*, adalah orde dari analisa LPC yang akan dilakukan. Nilai yang umum untuk orde analisa LPC adalah antara 8 sampai 16. Keuntungan dari penggunaan metode autokorelasi adalah bahwa nilai ke-nol, $r(0)$ adalah energi dari sinyal yang dibuat autokorelasinya.

5. Analisa LPC

   Langkah berikutnya adalah analisa LPC dimana semua nilai autokorelasi yang telah dihitung pada tahap sebelumnya akan diubah menjadi sebuah parameter LPC. Parameter ini bermacam-macam, ada disebut dengan nama koefisien LPC, koefisien pantulan (PARCOR), koefisien *cepstral*, atau transformasi lain yang diinginkan. Metode yang umum untuk menyelesaikan koefisien autokorelasi di atas menjadi koefisien LPC adalah dengan menggunakan metode Durbin yang algoritmanya adalah sebagai berikut:

$$E^{(0)} = r(0) \tag{2.43}$$

$$k_i = \frac{\left\{ r(i) - \sum_{j=1}^{L-1} \alpha_j^{(i-1)} \cdot r(|1-j|) \right\}}{E^{(i-1)}}, \qquad 1 \le i \le p \tag{2.44}$$

$$\alpha_i^{(i)} = k_i \tag{2.45}$$

$$\alpha_j^{(i)} = \alpha_j^{(i-1)} - k_i \cdot \alpha_{i-j}^{(i-1)} \tag{2.46}$$

$$E^{(i)} = (1 - k_i^2) \cdot E^{(i-1)} \tag{2.47}$$

Kemudian koefisien LPC yang dihasilkan adalah:

$$a_m = \alpha_m^{(p)}, \qquad 1 \le m \le p \tag{2.48}$$

6. Pengubahan parameter LPC menjadi koefisien *cepstral*.

   Parameter LPC yang sangat penting yang bisa diturunkan dari koefisien LPC adalah koefisien *cepstral* LPC, *c(m).* Persamaan yang digunakan untuk menghitungnya adalah:

$$c_m = a_m + \sum_{k=1}^{m-1} \left(\frac{k}{m} \right) \cdot c_k \cdot a_{m-k}, \qquad 1 \le m \le p \tag{2.49}$$

$$c_m = \sum_{k=1}^{m-1} \left(\frac{k}{m} \right) \cdot c_k \cdot a_{m-k}, \qquad m > p \tag{2.50}$$

   Koefisien *cepstral* ini adalah koefisien dari representasi transformasi Fourier pada spektrum logaritmis dan telah terbukti lebih dapat diandalkan untuk pengenalan suara manusia dibandingkan dengan koefisien LPC, atau koefisien PARCOR. Biasanya sebuah representasi *cepstral* menggunakan koefisien sebanyak *Q* dimana $Q > p$.

Perhitungan-perhitungan LPC di atas menggunakan pemilihan N=240, M=80, p=10, dan Q=12 (Sumber: Rabiner, Lawrence, and Biing-Hwang Juang. Fundamentals of Speech Recognition. New Jersey: Prentice Hall, 1993., hal. 122) karena frekuensi sampling yang akan digunakan adalah 8 KHz.

2.5. Pengukuran Distorsi

Hal yang paling penting dalam penggunaan algoritma perbandingan pola adalah cara mengukur kemiripan antara dua buah vektor suara. Pengukuran kemiripan ini dapat dilakukan dengan perhitungan matematika. Salah satu cara untuk mengukur kemiripan antara dua buah vektor suara adalah dengan cara mengukur distorsi spektral karena banyak perbedaan ciri-ciri suara yang dapat diinterpretasikan dalam hal spektral ini. Banyak cara yang dapat dilakukan untuk mengimplementasikan pengukuran distorsi spektral. Salah satu cara pengukuran distorsi spektral adalah dengan menggunakan pengukuran jarak *cepstral*.

Bentuk *cepstrum* kompleks dari sebuah sinyal dapat didefinisekan sebagai transformasi fourier dalam logaritma sebuah spektrum sinyal. Untuk sebuah

spektrum daya (*Magnitude Squared Fourier Transform*) ***S(w)*** yang simetris dengan pusat $w = 0$ dan periodik sepanjang datanya, representasi fouriernya dapat ditulis sebagai:

$$\log[S(w)] = \sum_{n=-\infty}^{\infty} c_n . e^{-jnw} \tag{2.51}$$

dimana $c_n = c_{-n}$ adalah real dan sering digambarkan sebagai koefisien *cepstral* dengan

$$c_0 = \int_{-p}^{p} \log[S(w)] \frac{dw}{2p} \tag{2.52}$$

Untuk sepasang spektrum ***S(w)*** dan ***S'(w)*** dengan mengaplikasikan teorema parseval maka jarak *cepstral* (d_c) dari kedua spektrum dalam rms adalah:

$$d_c^2 = \int_{-p}^{p} \left| \log[S(w)] - \log[S'(w)] \right|^2 \frac{dw}{2p}$$

$$d_c^2 = \sum_{n=-\infty}^{\infty} (c_n - c'_n)^2 \tag{2.53}$$

dimana c_n dan c'_n adalah koefisien *cepstral* dari ***S(w)*** dan ***S'(w)***.

Karena *cepstrum* merupakan urutan yang makin lama makin kecil maka penjumlahan pada persamaan (2.53) tidak memerlukan perhitungan sampai tak hingga. Untuk model LPC yang melambangkan *envelope* yang diperhalus dari sebuah spektrum suara manusia maka persamaan (2.53) biasanya diubah dalam jumlah kecil yaitu sekitar 10 sampai dengan 30. Karena koefisien *cepstral* sejumlah *p* yang pertama merupakan angka yang unik yang menentukan fase minimum dari filter LPC maka jumlah *L* dalam perhitungan jarak *cepstral* tidak boleh lebih kecil dari *p*. Maka persamaan (2.53) dapat diubah menjadi:

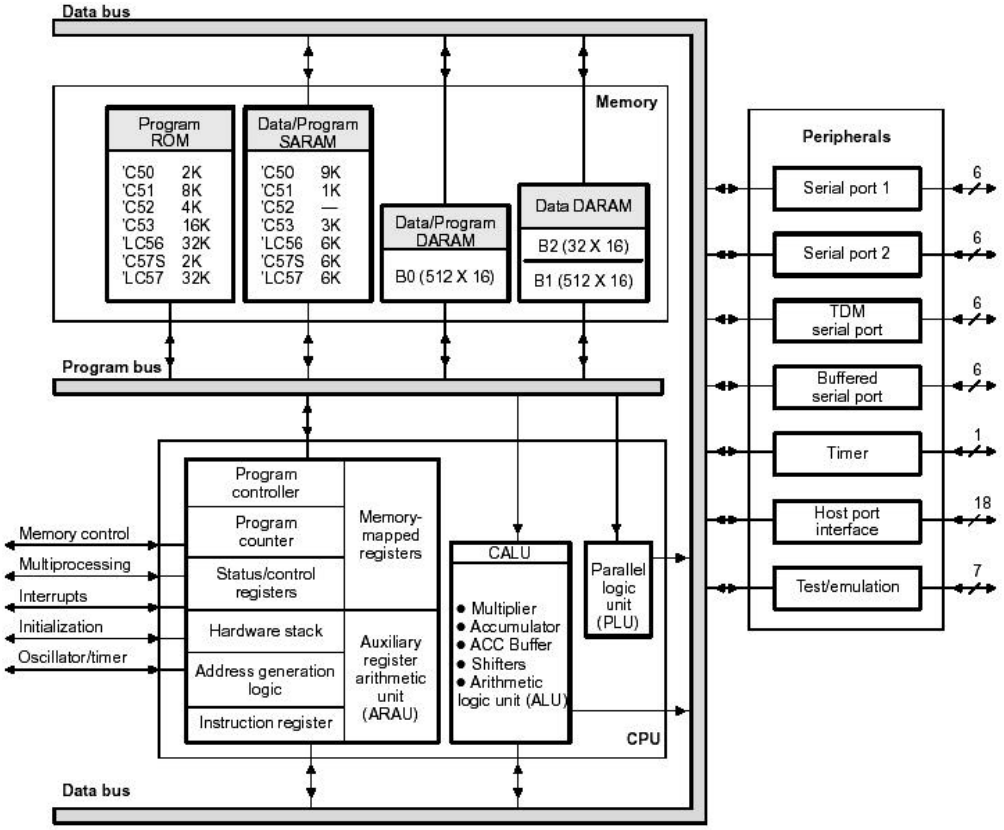
$$d_c^2 = \sum_{n=1}^{L} (c_n - c'_n)^2 \tag{2.54}$$

2.6. Sekilas Mengenai TMS320C5x

TMS320C5x merupakan salah satu prosesor sinyal digital yang dikeluarkan oleh pihak Texas Instrument dengan menggunakan teknologi prosesor *fixed-point*. Prosesor ini merupakan pengembangan dari prosesor TMS320C2x .

2.6.1. Arsitektur TMS320C5x

Generasi C5x terdiri dari prosesor sinyal digital C50, C51, C52, C53, C53S, C56, C57, dan C57S, yang dibuat dengan menggunakan teknologi rangkaian terpadu (*integrated circuit*) CMOS. Rancangan arsitekturnya didasarkan pada arsitektur C25. Fleksibilitas dan kecapan dari C5x merupakan hasil dari menggabungkan arsitektur Harvard (yang memisahkan jalur antara memori program dan memori data), CPU dengan hardware yang spesifik, peralatan *on-chip*, memory *on-chip*, dan instruksi-instruksi yang telah dikelompokkan secara khusus. C5x dirancang untuk dapat menjalankan perintah hingga 50 juta instruksi per detik.



Sumber: TMS320C5x User's Guide. Missouri: Texas Instruments, Inc., 1997., hal. 2-2.

Gambar 2.15. Blok Diagram Prosesor C5x

Keuntungan dari prosesor C5x:

- Arsitektur TMS320 yang canggih untuk meningkatkan performa.
- Teknologi pemrosesan rangkaian terpadu yang canggih untuk meningkatkan performa serta memperkecil konsumsi daya.

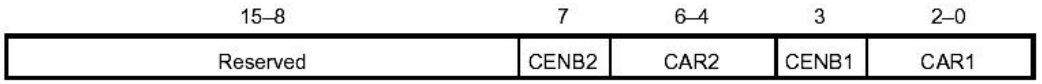
- Perintah yang kompatibel dengan C1x, C2x, dan C2xx.
- Kumpulan instruksi yang canggih untuk algoritma yang lebih cepat.
- Pengurangan konsumsi daya dan peningkatan ketahanan terhadap listrik statis.

Untuk mengetahui lebih jelas mengenai konstruksi dari prosesor sinyal digital C5x, dapat dilihat pada gambar 2.15. Gambar tersebut menunjukkan blok diagram dari C5x.

2.6.2. Register Status dan Kontrol

Prosesor C5x memiliki empat buah register status dan kontrol yaitu:

- *Circular Buffer Control Register* (CRBR) dan *Processor Mode Status Register* (PMST) berisi informasi kontrol dan status. Karena register ini dapat dibaca langsung oleh prosesor (*memory-mapped*) maka kedua register ini terletak pada memori data.
- Register status ST0 dan ST1 yang berisi status dari berbagai keadaan dan mode kompatibilitas dengan prosesor C2x.



Sumber: TMS320C5x User's Guide. Missouri: Texas Instruments, Inc., 1997., hal. 4-7.

Gambar 2.16. Diagram CRBR

Tabel 2.1. Fungsi Tiap Bit CRBR

Nama	Nilai Reset	Fungsi
CENB2	0	Bit ini berfungsi untuk mengaktifkan atau mematikan *circular buffer* 2. CENB2=0 Non-aktif CENB2=1 Aktif
CAR2	-	Bit ini berfungsi untuk memilih register *auxiliary* yang digunakan oleh *circular buffer* 2.
CENB1	0	Bit ini berfungsi untuk mengaktifkan atau mematikan *circular buffer* 1. CENB1=0 Non-aktif CENB1=1 Aktif

Tabel 2.1. (Sambungan)

CAR1	-	Bit ini berfungsi untuk memilih register *auxiliary* yang digunakan oleh *circular buffer* 1.

2.6.2.1. *Circular Buffer Control Register* (CRBR)

CRBR berada di dalam memori data halaman ke nol dan dapat disimpan dengan cara yang sama dengan semua data yang ada di dalam memori data. CRBR juga dapat diakses secara langsung oleh CALU (*Central Arithmatic Logic Unit*) dan PLU (*Parallel Logic Unit*). Bit-bit dari CRBR serta kegunaannya dapat dilihat pada gambar 2.16 dan tabel 2.1.

2.6.2.2. *Processor Mode Status Register* (PMST)

PMST berada di dalam memori data halaman ke 0 dan dapat disimpan dengan cara yang sama dengan semua data yang ada di dalam memori data. PMST juga dapat diubah secara langsung oleh CALU (*Central Arithmatic Logic Unit*) dan PLU (*Parallel Logic Unit*).

PMST juga mempunyai sebuah *stack* khusus yang berfungsi untuk menyimpan nilai PMST saat terjadi interupsi. Kemudian jika interupsi selesai dengan adanya perintah RETI atau RETE maka nilai PMST akan dikembalikan seperti sebelum terjadi interupsi. Bit-bit dari PMST serta kegunaannya dapat dilihat pada gambar 2.17 dan tabel 2.2.

15–11	10	9	8	7	6	5	4	3	2	1	0
IPTR	0	0	0	AVIS	0	OVLY	RAM	MP/$\overline{MC}$	NDX	TRM	BRAF

Sumber: TMS320C5x User's Guide. Missouri: Texas Instruments, Inc., 1997., hal. 4-8.

Gambar 2.17. Diagram PMST

Tabel 2.2. Fungsi Tiap Bit PMST

Nama	Nilai Reset	Fungsi
IPTR	00000	Bit ini mengatur letak vektor interupsi pada 2K word memori program yang pertama.
AVIS	0	Bit ini berfungsi menampilkan alamat program ketika sedang menjalankan program dalam memori internal. AVIS=0 Alamat program ditampilkan. AVIS=1 Alamat program tidak ditampilkan, alamat program menampilkan alamat terakhir saat mengakses memori eksternal.
OVLY	0	Bit ini berfungsi untuk menggunakan RAM internal sebagai memori data. OVLY=0 RAM internal tidak dapat digunakan sebagai memori data. OVLY=1 RAM internal dapat digunakan sebagai memori data.
RAM	0	Bit ini berfungsi untuk menggunakan RAM internal sebagai memori program. RAM=0 RAM internal tidak dapat digunakan sebagai memori program. RAM=1 RAM internal dapat digunakan sebagai memori program.
MP/MC	-	Bit ini berfungsi untuk menggunakan ROM internal sebagai memori program. MP/MC=0 ROM internal tidak dapat digunakan sebagai memori program. MP/MC=1 ROM internal dapat digunakan sebagai memori program.
NDX	0	Bit ini berfungsi untuk mengatur instruksi yang kompatibel dengan prosesor C2x, yang merubah nilai register AR0, juga merubah register INDX, serta merubah register ARCR. NDX=0 Semua instruksi yang kompatibel dengan C2x, yang merubah nilai AR0, juga merubah INDX dan ARCR NDX=1 Semua instruksi yang kompatibel dengan C2x, yang merubah nilai AR0, tidak ikut merubah nilai INDX dan ARCR.
TRM	0	Bit ini berfungsi untuk mengatur semua instruksi yang kompatibel dengan C2x, yang mengisi TREG0, juga ikut mengisi TREG1 dan TREG2. TRM=0 Semua instruksi yang kompatibel dengan C2x,

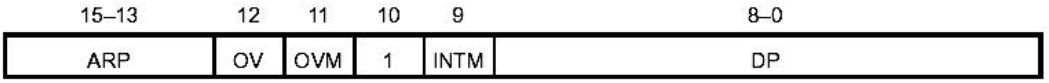
Tabel 2.2. (Sambungan)

		yang mengisi TREG0, juga ikut mengisi TREG1 dan TREG2. TRM=0 Semua instruksi yang kompatibel dengan C2x, yang mengsi TREG0, tidak diisikan ke TREG1 dan TREG2.
BRAF	0	Bit ini menunjukkan bahwa sebuah pengulangan blok sedang terjadi atau tidak. BRAF=0 Pengulangan blok tidak aktif. BRAF=1 Pengulangan blok sedang aktif.

2.6.2.3. Register status ST0 dan ST1

Register status ini dapat disimpan ke dalam memori data dan juga dapat dipanggil dari memori data. Perintah LST digunakan untuk menulis langsung ke dalam register ST0 dan ST1, dan perintah SST digunakan untuk membaca langsung isi dari register ST0 dan ST1. Register ini tidak seperti CRBR dan PMST yang terdapat langsung di dalam memori data. Register ini memiliki tempat khusus sehingga tidak dapat diakses langsung oleh PLU.

ST0 dan ST1 ini masing-masing memiliki sebuah *stack* khusus yang berfungsi untuk menyimpan nilai-nilai register itu saat terjadi interupsi. Kemudian jika interupsi selesai dengan adanya perintah RETI atau RETE maka nilai ST0 dan ST1 akan dikembalikan seperti sebelum terjadi interupsi kecuali bit INTM pada ST0 dan bit XF pada ST1. Kedua bit ini sama sekali tidak disimpan di dalam *stack* khusus tersebut. Ini berarti nilai kedua bit ini dapat digunakan dalam rutin interupsi yang sedang dijalankan. Bit-bit dari ST0 serta kegunaannya dapat dilihat pada gambar 2.18 dan tabel 2.3 sedangkan ST1 pada gambar 2.19 dan tabel 2.4.

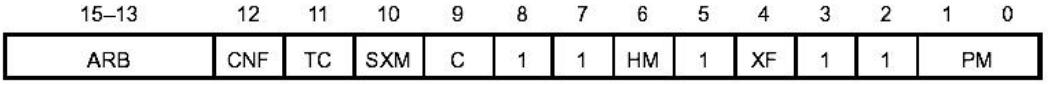


Sumber: TMS320C5x User's Guide. Missouri: Texas Instruments, Inc., 1997., hal. 4-11.

Gambar 2.18. Diagram ST0

Tabel 2.3. Fungsi Tiap Bit ST0

Nama	Nilai Reset	Fungsi
ARP	000	Bit ini berfungsi untuk memilih register *auxiliary* yang akan digunakan dalam pengalamatan tidak langsung (*indirect addressing*). Jika ARP diisi maka nilai ARP sebelumnya akan disimpan pada ARB yang terletak pada register ST1.
OV	0	Bit ini menunjukkan hasil dari ALU (*Arithmetic Logic Unit*) melebihi batas atau tidak. OV=0 Hasil dari ALU tidak melebihi batas. OV=1 Hasil dari ALU melebihi batas.
OVM	0	Bit ini mengatur mode *overflow* yang menentukan isi akumulator akan saturasi atau tidak. Isi akumulator tersebut tergantung hasil dari ALU. OVM=0 Jika hasil dari ALU melebihi batas, isi dari akumulator tetap. OVM=1 Jika hasil dari ALU melebihi batas, isi dari akumulator akan diisi dengan 7FFF FFFFh jika positif atau 8000 0000h jika negatif.
INTM	1	Bit ini berfungsi untuk mengatur interupsi akan diaktifkan secara global atau tidak. INTM=0 Semua interupsi diaktifkan. INTM=1 Semua interupsi tidak diaktifkan.
DP	0	Bit ini berfungsi untuk menyimpan 9 bit MSB (*most significant bit*) dari pointer memori data yang akan digunakan untuk pengalamatan secara langsung (*direct addressing*). Sedangkan 7 bit sisanya akan dimasukkan ke dalam opcodenya.



Sumber: TMS320C5x User's Guide. Missouri: Texas Instruments, Inc., 1997., hal. 4-13.

Gambar 2.19. Diagram ST1

Tabel 2.4. Fungsi Tiap Bit ST1

Nama	Nilai Reset	Fungsi
ARB	000	Bit ini berfungsi untuk menyimpan nilai sebelumnya dari ARP. Jika ARP diubah maka nilai sebelumnya akan disimpan dalam bit-bit ini.
CNF	0	Bit ini digunakan untuk mengaktifkan DARAM (*dual access* RAM) blok 0 agar dapat diakses melalui memori data atau memori program. CNF=0 DARAM dapat diakses oleh memori data. CNF=1 DARAM dapat diakses oleh memori program.
TC	0	Bit ini merupakan *flag* yang menyimpan hasil dari ALU atau PLU. Bit ini dipengaruhi oleh instruksi APL, BIT, BITT, CMPR, CPL, NORM, OPL, dan XPL.
SXM	1	Bit ini berguna untuk mengaktifkan atau mematikan fungsi tanda positif atau negatif pada operasi matematika. SXM=0 Tanda positif atau negatif tidak aktif (*unsigned*). SXM=1 Tanda positif atau negatif aktif (*signed*).
C	1	Bit ini disebut bit *carry*, bit ini menandakan adanya kelebihan perhitungan pada operasi penjumlahan atau pengurangan. C=0 Sebuah operasi penjumlahan tidak menghasilkan *carry* atau sebuah operasi pengurangan menghasilkan *borrow*. C=1 Sebuah operasi penjumlahan menghasilkan *carry* atau sebuah operasi pengurangan tidak menghasilkan *borrow*.
HM	1	Bit ini berfungsi untuk menentukan apakan CPU perlu memberhentikan atau meneruskan eksekusi instruksi pada saat sinyal HOLD aktif. HM=0 CPU meneruskan eksekusi instruksi dari memori program internal tapi tidak untuk memori program eksternal. HM=1 CPU menghentikan semua eksekusi instruksi.
XF	1	Bit ini berfungsi untuk menentukan level dari pin XF. XF=0 Pin XF bernilai logika nol. XF=1 Pin XF bernilai logika satu.
PM	00	Bit ini menentukan berapa banyak bit yang akan digeser pada hasil dari suatu operasi perkalian.

2.6.3. *Pipeline*

Dalam operasi *pipeline*, pengambilan dan penerjemahan perintah, pembacaan *operand*, dan menjalankan operasi tiap perintah merupakan hal yang terpisah-pisah sehingga dapat mengakibatkan beberapa perintah menjadi saling bertumpuk (*overlap*).

2.6.3.1. Struktur *Pipeline*

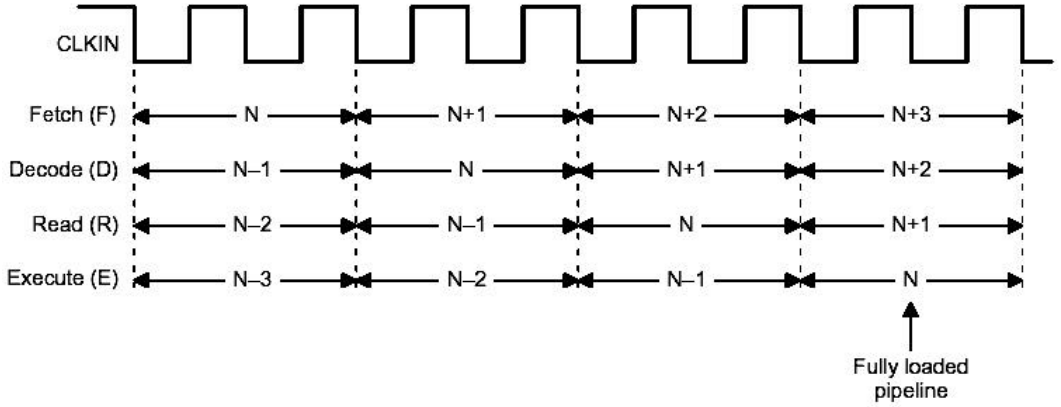
Ada empat fase dalam struktur *pipeline* C5x dan fungsi dari masing-masing fase adalah sebagai berikut:

1. Pengambilan (*Fetch*): Fase ini mengambil instruksi dari memori dan merubah *program counter* (PC).
2. Penerjemahan (*Decode*): Fase ini menerjemahkan instruksi yang diambil dari fase sebelumnya dan menghasilkan alamat serta merubah register *auxiliary*.
3. Pembacaan (*Read*): Fase ini membaca *operand* dari memori jika dibutuhkan.
4. Pengeksekusian (*Execute*): Fase ini menjalankan instruksi yang dimaksud pada fase sebelumnya dan kalau perlu maka akan diadakan operasi penulisan pada memori.

Gambar 2.20 menunjukkan operasi *pipeline* untuk perintah satu *word* dan satu siklus waktu yang dijalankan tanpa ada fase tunggu (*wait state*). Dapat dilihat pada gambar bahwa keempat fase tersebut sama sekali tidak berhubungan maka bisa saja jika terjadi konflik akan terjadi hasil yang tidak diinginkan. Oleh karena itu harus dipikirkan lebih jauh jika menginginkan hasil yang benar.

2.6.3.2. Operasi *Pipeline*

Operasi *pipeline* sebenarnya tidak dapat terlihat oleh pengguna kecuali pada beberapa kasus seperti perubahan nilai register *auxiliary*, instruksi NORM dan perintah-perintah untuk merubah isi memori. Untuk diketahui lebih lanjut bahwa operasi ini sama sekali tidak ada perlindungannya sehingga para pengguna prosesor C5x harus benar-benar mengerti tentang proses ini. Sebab jika tidak akan terjadi hal-hal yang tidak sesuai dengan yang diharapkan karena terjadinya konflik pada operasi *pipeline* ini.



Sumber: TMS320C5x User's Guide. Missouri: Texas Instruments, Inc., 1997., hal. 7-3.

Gambar 2.20. Operasi *Pipeline* Empat Tingkat dari Prosesor C5x

2.6.4. Interupsi

CPU C5x memiliki 16 interupsi yang bisa diatur oleh pemakai (INT1-INT16), namun tidak setiap prosesor C5x menggunakan semua interupsi ini. Sebagai contoh, semua prosesor C5x menggunakan 9 interupsi saja kecuali C57 yang menggunakan 10. Interupsi eksternal dibangkitkan oleh perangkat keras eksternal dengan menggunakan pin INT1-INT4. Interupsi internal yang dibangkitkan oleh sinyal-sinyal perangkat *on-chip* adalah:

- Timer (TINT)
- Kanal serial (RINT, XINT, TRNT, TXNT, BRNT, dan BXNT)
- Penggunaan kanal *Host* (HINT)

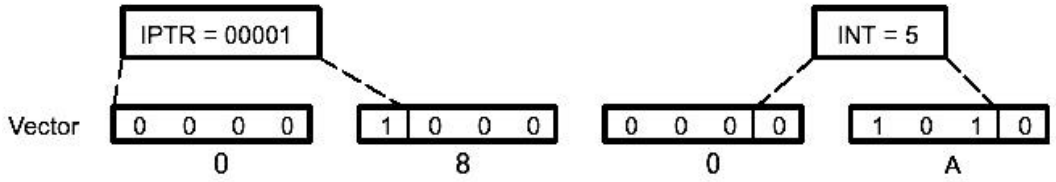
Sebagai tambahan, C5x juga mempunyai tiga perintah interupsi yang diatur secara perangkat lunak yaitu dengan menggunakan instruksi INTR, NMI, dan TRAP. Selain itu C5x juga mempunyai dua buah interupsi external yang tidak dapat diatur penggunaannya yaitu RS dan NMI. RS mempunyai prioritas interupsi yang tertinggi sedangkan INT16 adalah yang terendah. INT1-INT4 dan NMI baru aktif jika sinyal-sinyal tersebut mempunyai logika satu selama minimal dua siklus mesin dan nol selama minimal 3 siklus mesin. Kecuali jika prosesor dalam keadaan IDLE2 maka interupsi yang terjadi harus mempunyai logika satu minimal empat siklus mesin dan logika nol minimal 5 siklus mesin.

2.6.4.1. Lokasi Vektor Interupsi

Tabel 2.5 menunjukkan lokasi dari masing-masing vektor interupsi serta prioritas dari masing-masing interupsi. Vektor-vektor tersebut dapat diatur pada alamat 0 sampai 2048 pada memori program. Pengaturan alamat ini terletak pada register PMST yang telah dijelaskan di atas yaitu pada bit-bit IPTR. IPTR akan berubah kembali menjadi nol jika terjadi operasi reset. Sebagai contoh jika diinginkan vektor INT 5 berada pada alamat 080Ah maka IPTR cukup diisi dengan nilai 1. Untuk lebih jelasnya dapat dilihat pada gambar 2.21.

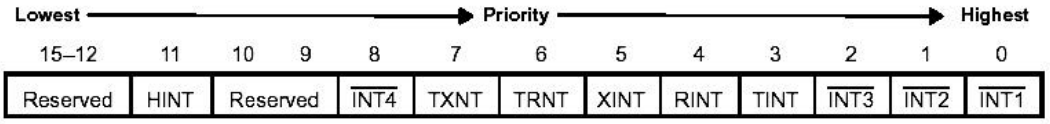
2.6.4.2. Operasi Interupsi

Ketika interupsi terjadi maka sebuah *flag* pada IFR (*Interrupt Flag* Register) akan diaktifkan. Susunan *flag* interupsi ini dapat dilihat pada gambar 2.22. *Flag* interupsi ini diaktifkan tidak peduli apakah interupsi diaktifkan atau tidak. Sebuah *flag* interupsi akan secara otomatis dikembalikan menjadi logika nol kecuali untuk *flag* yang berasal dari komunikasi serial.



Sumber: TMS320C5x User's Guide. Missouri: Texas Instruments, Inc., 1997., hal. 4-38.

Gambar 2.21. Mengubah Vektor Interupsi



Sumber: TMS320C5x User's Guide. Missouri: Texas Instruments, Inc., 1997., hal. 4-39.

Gambar 2.22. Diagram IFR

Tabel 2.5. Lokasi dan Prioritas Vektor Interupsi.

Name	Location Dec	Location Hex	Priority	Function
$\overline{RS}$	0	0	1 (highest)	External nonmaskable reset signal
$\overline{INT1}$	2	2	3	External user interrupt #1
$\overline{INT2}$	4	4	4	External user interrupt #2
$\overline{INT3}$	6	6	5	External user interrupt #3
TINT	8	8	6	Internal timer interrupt
RINT	10	A	7	Serial port receive interrupt
XINT	12	C	8	Serial port transmit interrupt
TRNT†	14	E	9	TDM port receive interrupt
TXNT‡	16	10	10	TDM port transmit interrupt
$\overline{INT4}$	18	12	11	External user interrupt #4
—	20–23	14–17	N/A	Reserved
HINT	24	18	—	HINT ('C57 only)
—	26–33	1A–21	N/A	Reserved
TRAP	34	22	N/A	Software trap instruction
$\overline{NMI}$	36	24	2	Nonmaskable interrupt
—	38–39	26–27	N/A	Reserved for emulation and test
—	40–63	28–3F	N/A	Software interrupts

† RINT2 on 'C52; BRNT on 'C56/C57
‡ XINT2 on 'C52; BXNT on 'C56/C57

Sumber: TMS320C5x User's Guide. Missouri: Texas Instruments, Inc., 1997., hal. 4-37.

Jika sebuah interupsi terjadi maka akan terjadi beberapa hal berikut ini:

- CPU akan menyelesaikan instruksi saat ini.
- Interupsi secara global akan di non aktifkan (INTM=1).
- PC akan didorong ke dalam *stack* yang paling atas.
- PC akan diisi dengan vektor interupsi yang bersangkutan.
- Register-register penting akan disimpan didalam *stack* bayangan.
- *Flag* interupsi yang bersangkutan akan dibuat menjadi logika nol.

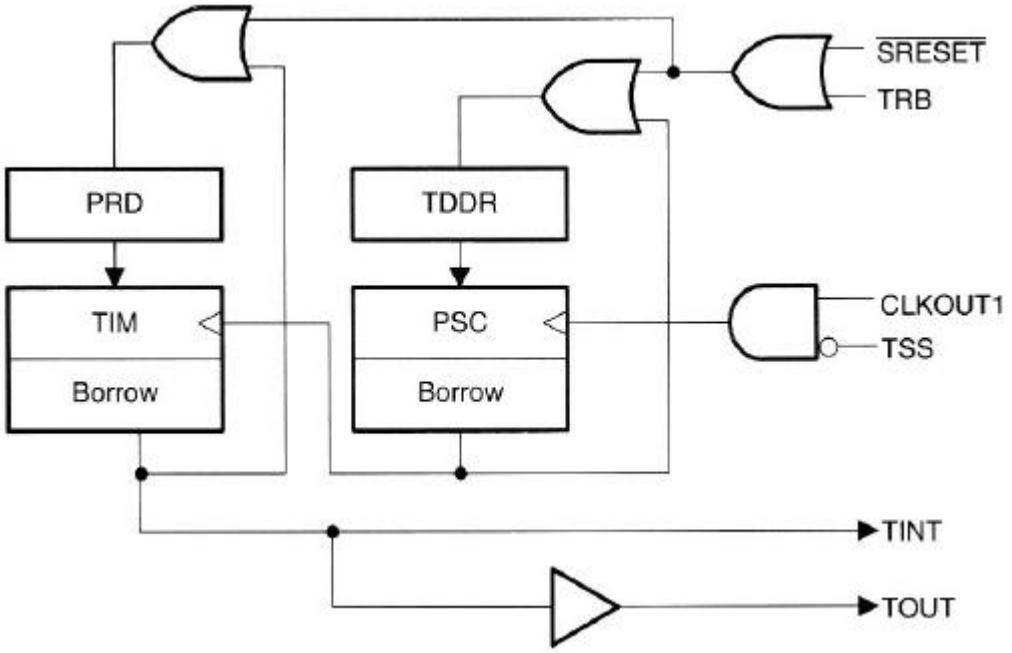
2.6.5. *Peripheral*

Peripheral on-chip pada prosesor C5x terdiri dari timer, generator waktu tunggu yang dapat diatur dengan perangkat lunak (*software programmable wait state generators*), kanal input/output paralel, dan kanal serial. Semua *peripheral*

ini dapat diatur melalui register-register tertentu yang terdapat dalam memori data. Kanal serial dan timer merupakan peripheral yang telah disinkronisasi dengan menggunakan interupsi.

2.6.5.1. Timer

Timer ini merupakan penghitung mundur yang dapat digunakan untuk menghasilkan sinyal interupsi secara periodik. Gambar 2.23 menunjukkan blok diagram dari timer pada C5x. Timer ini dikendalikan oleh sebuah *prescaler* yang akan dikurangi satu setiap ada sinyal CLKOUT1. Sebuah interupsi TINT akan diaktifkan setiap kali nilai timer ini mencapai nilai nol. Timer ini dapat digunakan untuk mengaktifkan sebuah input/output secara periodik. Pada saat TSS bernilai satu maka timer ini akan mati.

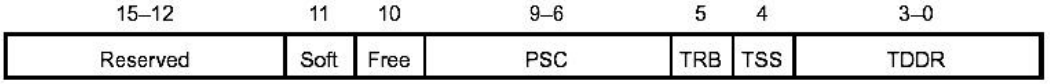


Sumber: TMS320C5x User's Guide. Missouri: Texas Instruments, Inc., 1997., hal. 9-9.

Gambar 2.23. Blok Diagram Timer

Operasi timer ini dapat diatur melalui register TCR, TIM, dan PRD. Susunan dari register TCR dapat dilihat pada gambar 2.24 dan kegunaan dari masing-masing bit dapat dilihat pada tabel 2.6. Register TIM berfungsi untuk menyimpan nilai dari timer sedangkan TCR dan PRD dapat digunakan untuk mengatur frekuensi TINT yang dihasilkan. Frekuensi TINT dapat ditulis sebagai:

$$TINT = \frac{1}{CLKOUT1 \times (TDDR + 1) \times (PRD + 1)} \quad (2.55)$$



Sumber: TMS320C5x User's Guide. Missouri: Texas Instruments, Inc., 1997., hal. 9-10.

Gambar 2.24. Diagram TCR

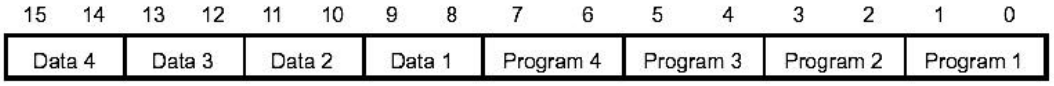
Tabel 2.6. Kegunaan bit-bit TCR

Nama	Nilai Reset	Fungsi
Soft	0	Bit ini bekerja sama dengan bit Free untuk menentukan keadaan timer saat terjadi sebuah peringatan. Jika bit Free bernilai nol maka bit ini baru bisa berfungsi. Soft=0 Timer berhenti seketika dan membatalkan segala macam transmisi. Soft=1 Timer berhenti setelah menyelesaikan transmisi.
Free	0	Bit ini bekerja sama dengan bit Soft untuk menentukan keadaan timer saat terjadi sebuah peringatan. Free=0 Mode timer ditentukan oleh bit Soft. Free=1 Timer terus berjalan tanpa mempedulikan bit Soft.
PSC	-	Bit ini menyimpan nilai dari perhitungan timer. Jika PSC mencapai nilai nol atau timer di-reset maka PSC akan diisikan ke TDDR dan TIM dikurangi satu.
TRB	-	Bit in berfungsi untuk me-reset timer. Jika TRB=1 maka TIM akan diisikan ke PRD dan PSC diisi dengan TDDR.
TSS	0	Bit ini berfungsi untuk mematikan atau menyalakan timer. TSS=0 Timer aktif. TSS=1 Timer tidak aktif.
TDDR	0000	Bit ini berisi nilai pembagian dari timer. Jika PSC bernilai nol maka PSC akan diisi dengan nilai dari TDDR.

2.6.5.2. *Software Programmable Wait State Generators*

Software Programmable Wait State Generators ini dapat digunakan untuk memperlambat akses eksternal. Hal ini kadang-kadang diperlukan karena tidak

semua peralatan eksternal memiliki kecepatan yang sama cepat dengan prosesor C5x. Waktu tunggu yang dihasilkan ini dapat digunakan untuk mengakses memori atau peralatan input/output. Untuk mengatur waktu tunggu akses memori digunakan register PDWSR dan untuk mengatur input/output digunakan registr IOWSR. Susunan PDWSR dan IOWSR dapat dilihat pada gambar 2.25 dan 2.26, dan jangkauan alamat untuk masing-masing register dapat dilihat pada tabel 2.7 dan 2.8. Waktu tunggu yang dihasilkan dapat dilihat pada tabel 2.9.



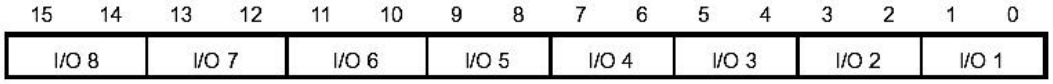
Sumber: TMS320C5x User's Guide. Missouri: Texas Instruments, Inc., 1997., hal. 9-13.

Gambar 2.25. Diagram PDWSR

Tabel 2.7. Alamat PDWSR

PDWSR Bits	Memory Space	Hex Address Range
15–14	Data 4	C000–FFFF
13–12	Data 3	8000–BFFF
11–10	Data 2	4000–7FFF
9–8	Data 1	0000–3FFF
7–6	Program 4	C000–FFFF
5–4	Program 3	8000–BFFF
3–2	Program 2	4000–7FFF
1–0	Program 1	0000–3FFF

Sumber: TMS320C5x User's Guide. Missouri: Texas Instruments, Inc., 1997., hal. 9-14.



Sumber: TMS320C5x User's Guide. Missouri: Texas Instruments, Inc., 1997., hal. 9-16.

Gambar 2.26. Diagram IOWSR

Tabel 2.8. Alamat IOWSR

Wait-State Field Bits	I/O Space	Ports/Hex Address Range	
		BIG = 0	BIG = 1
0–1	I/O 1	Port 0/1, port 10/11, etc.	0000–1FFF
2–3	I/O 2	Port 2/3, port 12/13, etc.	2000–3FFF
4–5	I/O 3	Port 4/5, port 14/15, etc.	4000–5FFF
6–7	I/O 4	Port 6/7, port 16/17, etc.	6000–7FFF
8–9	I/O 5	Port 8/9, port 18/19, etc.	8000–9FFF
10–11	I/O 6	Port 0A/0B, port 1A/1B, etc.	A000–BFFF
12–13	I/O 7	Port 0C/0D, Port 1C/1D, etc.	C000–DFFF
14–15	I/O 8	Port 0E/0F, Port 1E/1F, etc.	E000–FFFF

Sumber: TMS320C5x User's Guide. Missouri: Texas Instruments, Inc., 1997., hal. 9-16.

2.6.5.3. Kanal Input/Output Paralel

C5x memiliki 64K kanal input/output paralel. Semua kanal ini dapat diakses dengan menggunakan perintah IN dan OUT. Penggunaan kanal-kanal ini akan mengaktifkan sinyal IS sendangkan sinyal DS tidak terpengaruh sama sekali. Jadi untuk menggunakan peralatan input/output cukup diperlukan sinyal IS, WE untuk output, dan RD untuk input.

Tabel 2.9. Waktu Tunggu yang Dihasilkan.

	Number of CLKOUT1 Cycles†			
	Hardware Wait State		Software Wait State	
Number of Wait States	Read	Write	Read	Write
0	1	2n + 1	1	2n + 1
1	2	3n + 1	2	2n + 1
2	3	4n + 1	3	3n + 1
3	4	5n + 1	4	4n + 1

†Where n is the number of consecutive write cycles.

Sumber: TMS320C5x User's Guide. Missouri: Texas Instruments, Inc., 1997., hal. 9-15.

2.6.5.4. Kanal Serial

Kanal serial pada prosesor C5x beroperasi melalui tiga buah register yaitu SPC (Serial Port Control Register), DRR (Data Receive Register), dan DXR (Data Transmit Register). Jadi untuk menerima data digunakan DRR dan untuk mengirim data digunakan DXR. Sedangkan untuk mengatur operasi kanal serialnya digunakan register SPC. Untuk melihat susunan register SPC dapat dilihat pada gambar 2.27. Dalam operasi serial ini yang digunakan hanya bit RRST untuk me-reset penerima, bit XRST untuk me-reset pengirim, dan bit FSM yang mengatur agar komunikasi serial berjalan sesuai dengan sinyal FSX dan FSR untuk itu maka FSM diberi nilai satu.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Free	Soft	RSRFULL	$\overline{\text{XSREMPTY}}$	XRDY	RRDY	IN1	IN0	$\overline{\text{RRST}}$	$\overline{\text{XRST}}$	TXM	MCM	FSM	FO	DLB	Res
R/W	R/W	R	R	R	R	R	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R

Note: R = Read, W = Write

Sumber: TMS320C5x User's Guide. Missouri: Texas Instruments, Inc., 1997., hal. 9-28.

Gambar 2.27. Diagram SPC

2.6.6. Memori

Rancangan C5x didasarkan pada arsitektur Harvard yang mempunyai lebih dari satu jalur memori. Hal ini membuat prosesor ini mampu mengakses memori program dan memori data secara simultan. Dua jalur memori ini adalah jalur

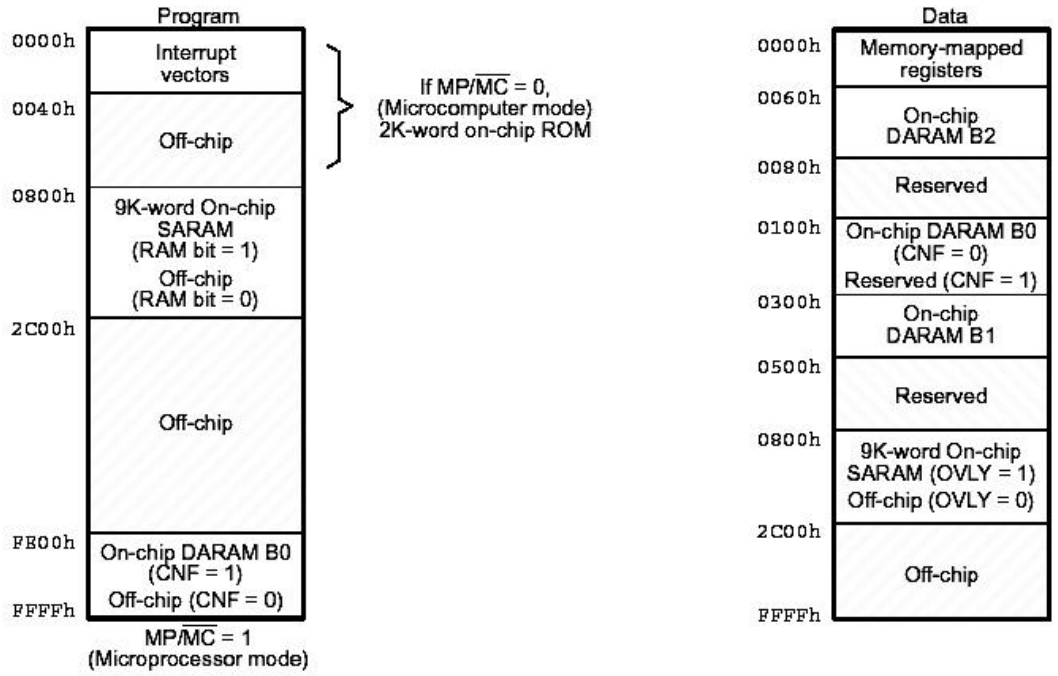
program (PB) dan jalur data (DB). Setiap jalur mengakses alamat memori yang berbeda tergantung operasi yang dilakukan. Memori C5x memiliki empat daerah yang dapat dipilih yaitu: memori program (64K *words*), memori data lokal (64K *words*), memori data global (32K *words*), dan kanal input/output (64K *words*). Daerah memori ini membentuk total alamat memori sebesar 224K *words* yang di dalamnya bisa terdiri dari RAM, ROM, EPROM, atau EEPROM.

Setiap prosesor C5x memiliki memori *on-chip* yang cukup besar yang terdiri RAM dan ROM. Dalam hal ini tipe yang digunakan adalah TMS320C50. RAM internal dari C50 berukuran 8K *words* dan ROM internalnya berukuran 2K *words* yang tersusun menjadi beberapa bagian. Untuk melihat lebih jelas pemetaan memori pada TMS320C50 dapat dilihat pada gambar 2.28. Dari gambar 2.28 tersebut terlihat bahwa memori data maupun memori program dari prosesor TMS320C50 dapat diatur sesuai dengan kebutuhan tergantung dari bit-bit yang mengontrol pembagian memori tersebut. Untuk melihat lebih jelas mengenai bit-bit serta alamat memori yang digunakan pada memori program dapat dilihat pada tabel 2.10 sedangkan pengaturan untuk memori data dapat dilihat pada tabel 2.11.

Tabel 2.10. Konfigurasi Memori Program C50

Bit values			ROM	SARAM	DARAM B0	
CNF	RAM	MP/$\overline{MC}$	(2K-words)	(9K-words)	(512-words)	Off-Chip
0	0	0	0000–07FF	Off-chip	Off-chip	0800–FFFF
0	0	1	Off-chip	Off-chip	Off-chip	0000–FFFF
0	1	0	0000–07FF	0800–2BFF	Off-chip	2C00–FFFF
0	1	1	Off-chip	0800–2BFF	Off-chip	0000–07FF, 2C00–FFFF
1	0	0	0000–07FF	Off-chip	FE00–FFFF	0800–FDFF
1	0	1	Off-chip	Off-chip	FE00–FFFF	0000–FDFF
1	1	0	0000–07FF	0800–2BFF	FE00–FFFF	2C00–FDFF
1	1	1	Off-chip	0800–2BFF	FE00–FFFF	0000–07FF, 2C00–FDFF

Sumber: TMS320C5x User's Guide. Missouri: Texas Instruments, Inc., 1997., hal. 8-8.



Sumber: TMS320C5x User's Guide. Missouri: Texas Instruments, Inc., 1997., hal. 8-4.

Gambar 2.28. Pemetaan Memori TMS320C50

Tabel 2.11. Konfigurasi Memori Data C50

Bit values		Registers	DARAM B2	DARAM B0	DARAM B1	SARAM	
CNF	OVLY	(96-words)	(32-words)	(512-words)	(512-words)	(9K-words)	Off-Chip
0	0	0000–005F	0060–007F	0100–02FF	0300–04FF	Off-chip	0800–FFFF
0	1	0000–005F	0060–007F	0100–02FF	0300–04FF	0800–2BFF	2C00–FFFF
1	0	0000–005F	0060–007F	Reserved	0300–04FF	Off-chip	0800–FFFF
1	1	0000–005F	0060–007F	Reserved	0300–04FF	0800–2BFF	2C00–FFFF

Sumber: TMS320C5x User's Guide. Missouri: Texas Instruments, Inc., 1997., hal. 8-16.

2.7. Algoritma *Floating Point*

Prosesor sinyal digital TMS320C50 adalah sebuah prosesor *fixed point* yang memiliki kemampuan 16/32 bit. Sehingga jika ingin dilakukan perhitungan *floating point* diperlukan tambahan modul perangkat lunak untuk melakukan perhitungan tersebut atau yang biasa disebut sebagai mengemulasikan perhitungan *floating point*. Format floating point adalah sebagai berikut:

$$f = m \times b^e \qquad (2.56)$$

dimana:

m = mantissa

b = basis

e = eksponen

sebagai contoh misalkan sebuah bilangan 145.000 dapat ditulis sebagai:

0.145×10^6

ini artinya m=0.145, b=10, dan e=6. Misalkan dua buah bilangan floating point ditulis sebagai:

$f_1 = m_1 \times b^{e1}$ (2.57)

$f_2 = m_2 \times b^{e2}$ (2.58)

maka penjumlahan atau pengurangan, perkalian, dan pembagian *floating point* adalah sebagai berikut:

$$f_1 \pm f_2 = \left\{m_1 \pm m_2 \times b^{-(e1-e2)}\right\} \times b^{e1}, \text{ jika } e1 \geq e2 \quad (2.59)$$

$$f_1 \pm f_2 = \left\{m_1 \times b^{-(e2-e1)} \pm m_2\right\} \times b^{e2}, \text{ jika } e1 < e2 \quad (2.60)$$

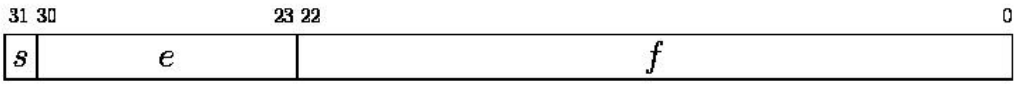
$$f_1 \times f_2 = m_1 \times m_2 \times b^{(e1+e2)} \quad (2.61)$$

$$\frac{f_1}{f_2} = \frac{m_1}{m_2} \times b^{(e1-e2)} \quad (2.62)$$

Persamaan (2.59) hingga (2.62) merupakan cara perhitungan *floating point*. Pada perhitungan sistem pengenalan kata ini digunakan format floating point yang telah dibuat standard oleh IEEE yaitu format *single-precision* yang memiliki format sebagai berikut:

$$X = (-1)^s \times 2^{(e-127)} \times 1.f \quad (2.63)$$

yang tersusun menjadi 32 bit seperti yang terlihat pada gambar 2.29.



Sumber: Simar, Ray, Jr. Floating Point Arithmetic with TMS32010. Texas Instruments, Inc., 1989., hal. 9.

Gambar 2.29. Format S*ingle-Precision* dari IEEE

Jadi nanti dalam implementasinya, bit-bit tersebut harus dipisah-pisah terlebih dahulu untuk diambil s (tanda), e (eksponen), dan f (*mantissa*). Kemudian dilakukan perhitungan sesuai dengan yang tertulis pada persamaan (2.59) hingga (2.62). Langkah-langkah untuk melakukan perkalian *floating point* dapat dilihat

pada gambar 2.30 sedangkan untuk melakukan penjumlahan atau pengurangan *floating point* dapat pada gambar 2.31.

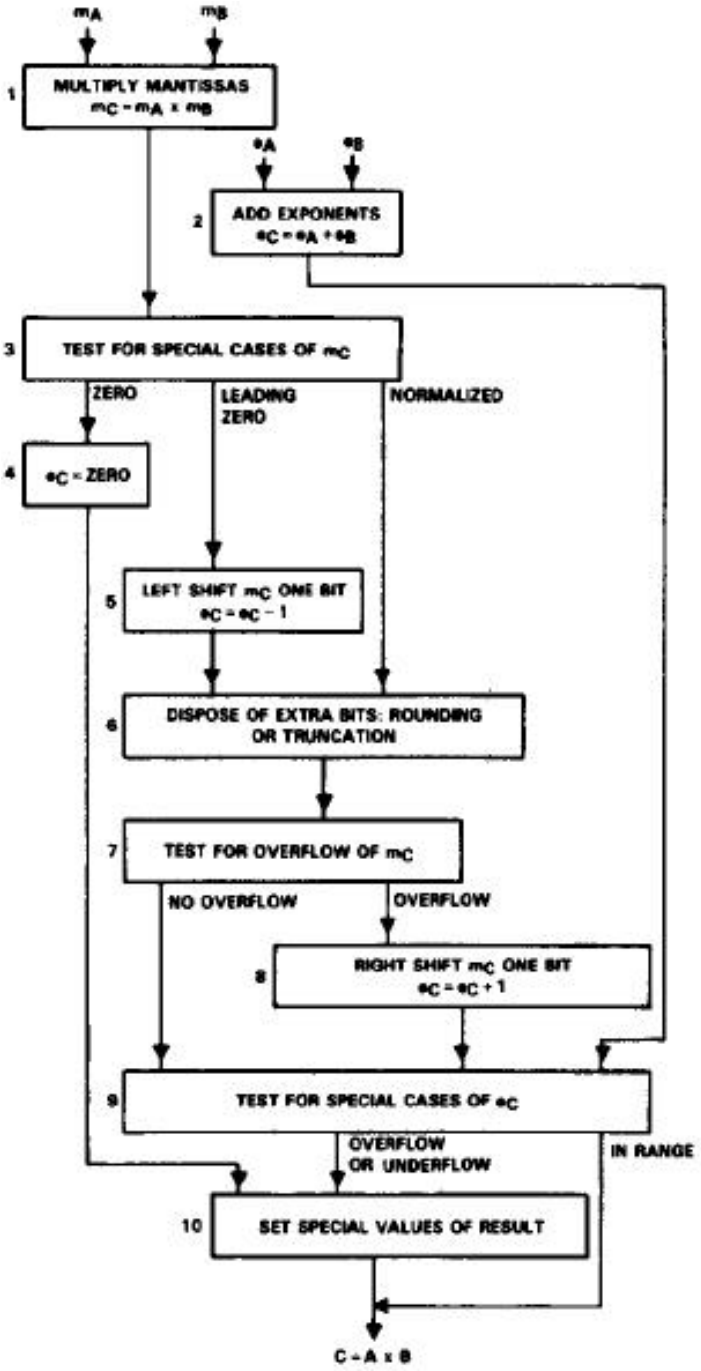
Langkah dasar dari penjumlahan atau pengurangan floating point adalah sebagai berikut:

1. Bandingkan kedua bilangan eksponennya yaitu e1 dan e2. Karena proses penjumlahan dan pengurangan ini ditentukan oleh bilangan eksponen yang lebih besar.
2. *Mantissa* dari bilangan eksponen yang lebih rendah, digeser ke kanan sejumlah selisih dari kedua bilangan eksponen.
3. *Mantissa* yang telah digeser dijumlahkan atau dikurangkan dengan *mantissa* yang lainnya.
4. Lakukan normalisasi pada mantissa hasil penjumlahan tadi. Artinya adalah hasil dari penjumlahan mantissa tadi harus dibuat dalam bentuk 1.f.

Langkah dasar dari perkalian floating point adalah sebagai berikut:

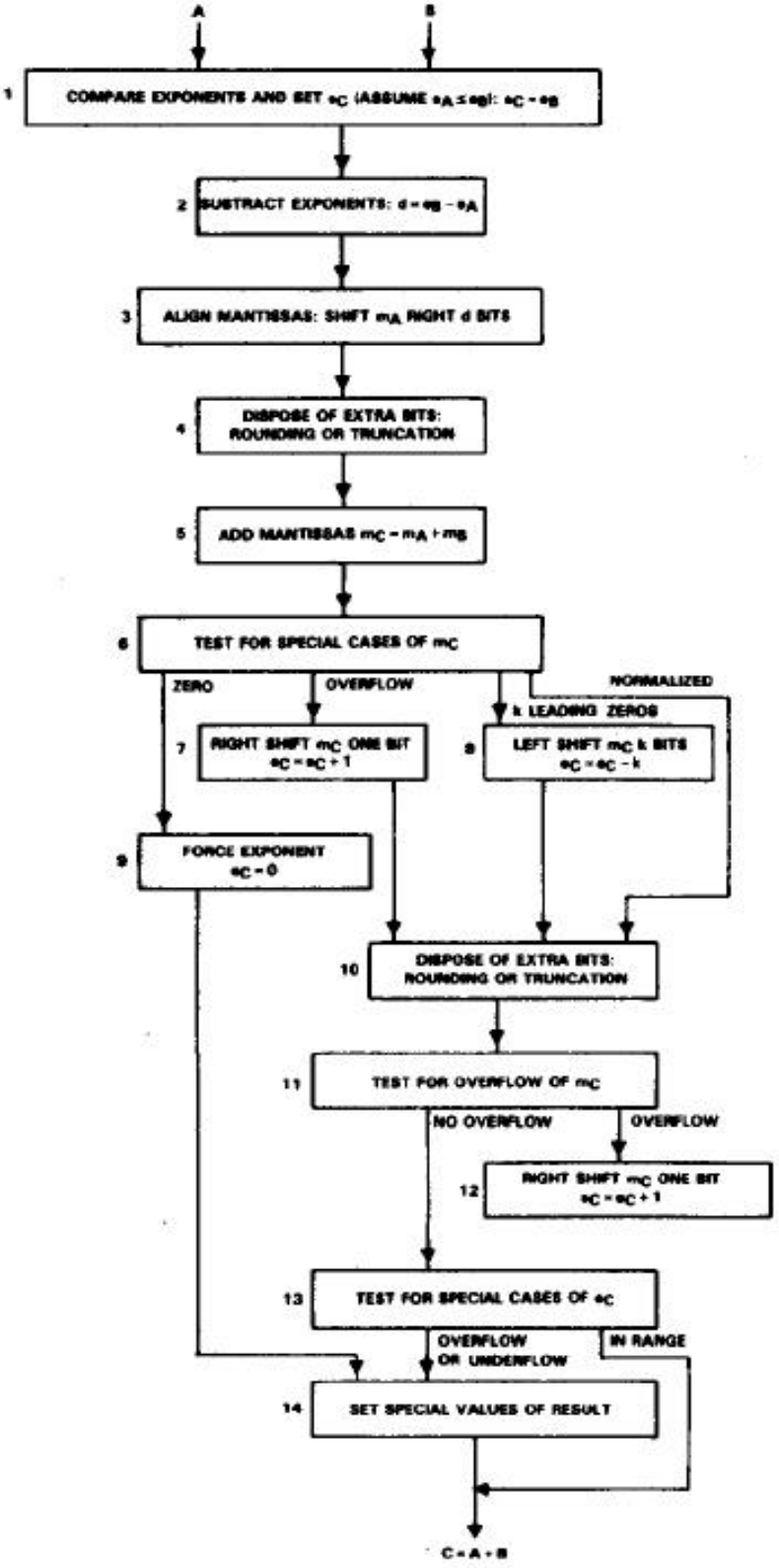
1. Lakukan perkalian antara *mantissa* dari bilangan yang pertama dengan *matissa* bilangan yang kedua.
2. Lakukan penjumlahan pada bilangan eksponennya yaitu e1+e2.
3. Lakukan normalisasi terhadap hasil perkalian *mantissa*.

Sedangkan untuk pembagian *floating point* pada dasarnya sama dengan perkalian *floating point* bedanya hanya terletak pada langkah nomor 1 dan 2. Pada pembagian *floating point*, langkah nomor 1 diganti dengan proses pembagian antara m_1 dan m_2 sedangkan untuk langkah nomor 2 diganti dengan proses pengurangan eksponen.



Sumber: Simar, Ray, Jr. Floating Point Arithmetic with TMS32010. Texas Instruments, Inc., 1989., hal. 8.

Gambar 2.30. Perkalian *Floating Point*



Sumber: Simar, Ray, Jr. Floating Point Arithmetic with TMS32010. Texas Instruments, Inc., 1989., hal. 9.

Gambar 2.31. Penjumlahan atau Pengurangan *Floating Point*