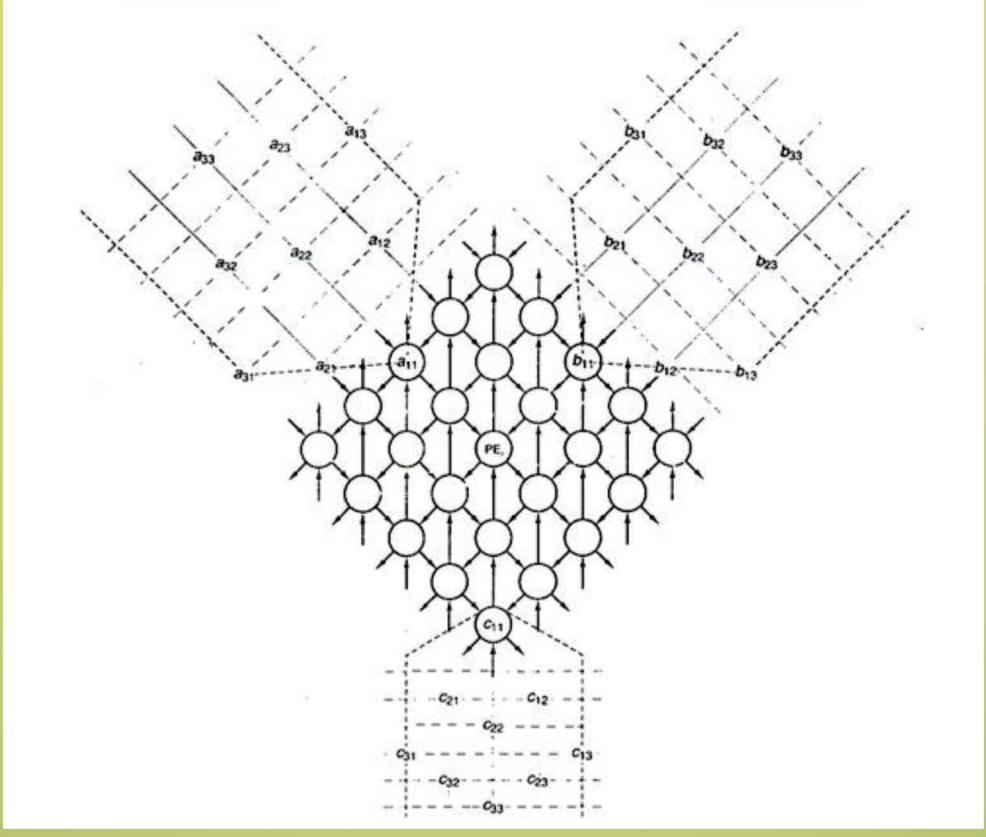
Dahlan Abdullah

PEMROSESAN PARALEL



UNIMAL PRESS

PEMROSESAN PARALEL
(PARALLEL PROCESSING)

PEMROSESAN PARALEL

Dahlan Abdullah, ST, MT

UNIMAL PRESS

Judul Buku: **Pemrosesan Paralel (**Parallel Processing)

Cetakan Pertama: Januari, 2016

Penulis:
Dahlan Abdullah, ST, MT
Editor:
Perancang Sampul:
Penata Letak:
Pracetak dan Produksi: **Unimal Press**

Penerbit:
UNIMAL PRESS

Unimal Press
Jl. Sulawesi No.1-2
Kampus Bukit Indah Lhokseumawe 24351
PO.Box. 141. Telp. 0645-41373. Fax. 0645-44450
Laman: www.unimal.ac.id/unimalpress.
Email: unimalpress@gmail.com

ISBN:
xvii + 200 hal., 14,8 cm x 21 cm

Kata Pengantar

Teknologi pemrosesan paralel telah menjadi minat dan perhatian yang tidak baru dalam dunia informatika, tetapi perkembangan penerapan aplikasinya sendiri baru dapat dirasakan beberapa tahun belakangan ini. Berbagai arsitektur yang menggunakan teknologi tersebut telah ditawarkan meluas dan mudah ditemukan di pasar komputer, dan sementara teori atau pondasi yang melatarbelakangi arsitektur ini sangat sedikit dan sulit kita temukan dengan mudah. Di berbagai Universitas, mata kuliah pemrosesan paralel menjadi bagian penting dan terintegrasi di dalam kurikulum program strudi Teknik Informatika atau Ilmu Komputer. Untuk itu, buku pemrosesan paralel yang pembaca pegang ini hadir menjadi referensi alternatif untuk mendukung penambahan khazanah di bidang komputer khususnya pemrosesan paralel.

Adapun materi yang hadir di dalam buku ini meliputi : Klasifikasi Perancangan, Arsitektur Von Neumann, Pipeline, Arsitektur RISC vs CISC, dan Arsitektur Systolic Array & Arsitektur Data Flow.

Daftar Isi

Bab 1
KLASIFIKASI PERANCANGAN

TAKSONOMI ARSITEKTUR KOMPUTER

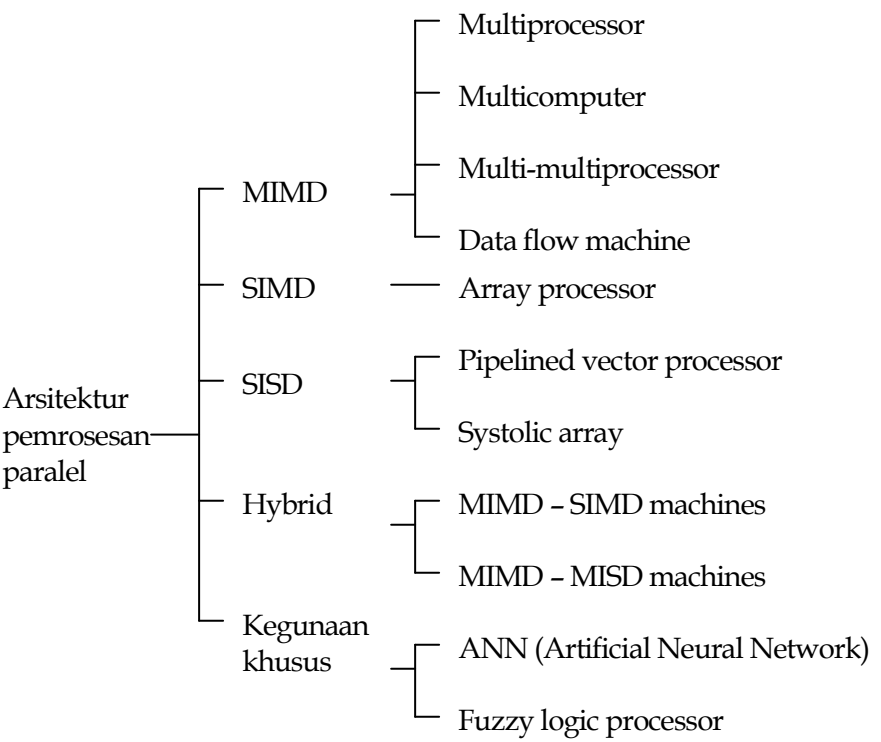
Taksonomi arsitektur komputer yang telah dikenal baik diklasifikasikan oleh Flynn. Michael Flynn mengklasifikasikan arsitektur ke dalam empat kategori berdasarkan hadir atau tidaknya berbagai aliran instruksi dan aliran data. Keempat kategori tersebut adalah :

1. SISD (single instruction stream, single data stream). Jenis kategori ini merupakan manifestasi dari konsep arsitektur Von Neumann terhadap perancangan komputer serial yang mana hanya satu instruksi yang dieksekusi pada sebarang waktu. SISD sering juga diistilahkan sebagai komputer skalar serial. Keseluruhan mesin SISD memanfaatkan sebuah register tunggal, yang disebut sebagai *program counter*, yang melakukan sederetan eksekusi instruksi secara serial. Setiap masing-masing instruksi diambil dari memori, program counter akan selalu diupdate untuk mengisi alamat instruksi berikut yang akan diambil dan dieksekusikan secara berurutan.

2. MISD (multiple instruction stream, single data stream). Kategori ini bermaksud bahwa beberapa instruksi melakukan operasi pada suatu data tunggal. Ada dua cara dalam menginterpretasikan organisasi mesin jenis MISD. Cara pertama adalah dengan mempertimbangkan kelas mesin yang membutuhkan berbagai unit pemrosesan berbeda yang menerima operasi instruksi berbeda pada data yang sama. Kelas mesin ini tidak begitu praktis diterapkan oleh para arsitek komputer pada umumnya. Dan sampai saat ini tidak ada satupun komputer yang mengadopsi definisi dari jenis mesin ini. Cara kedua adalah dengan mempertimbangkan kelas mesin yang memiliki definisi : data mengalir melalui sederetan unit pemrosesan. Arsitektur pipeline yang lebih tinggi : seperti array systolic dan prosesor vektor, dapat diklasifikasikan sebagai mesin jenis definisi ini. Arsitektur pipeline melakukan pemrosesan vektor melalui sederetan stadium, yang masing-masing memiliki fungsi tertentu dan menghasilkan output yang diharapkan dengan segera. Alasan kenapa arsitektur tersebut dikategorikan sebagai sistem MISD adalah bahwa elemen-elemen suatu vektor dapat dikelompokkan sebagai bagian dari data yang sama, dan seluruh stadium pipeline merepresentasikan multiple instruksi yang diaplikasikan terhadap vektor tersebut.
3. SIMD (single instruction stream, multiple data stream). Kategori ini bermaksud bahwa satu instruksi tunggal diaplikasikan terhadap berbagai data yang berbeda secara simultan. Dalam mesin jenis ini, berbagai unit pemrosesan yang terpisah melakukan kerja atas instruksi dari unit kontrol tunggal. Sebagaimana mesin MISD, mesin SIMD juga mendukung pemrosesan vektor. Ini dapat diselesaikan dengan menugaskan elemen vektor ke unit pemrosesan individual untuk komputasi yang bersamaan.
4. MIMD (multiple instruction stream, multiple data stream). Kategori ini meliputi mesin dan beberapa unit pengolahan di mana berbagai instruksi diaplikasi secara serempak ke berbagai data berbeda. Mesin MIMD dirancang dengan

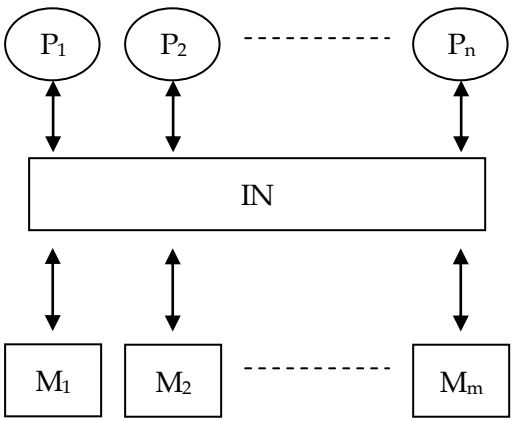
proses yang begitu kompleks, tetapi memiliki keunggulan dalam prihal efisiensi pada setiap pengolahan data yang dilakukan.

Klasifikasi Flynn telah terbukti menjadi metode klasifikasi terbaik selama tiga dekade ini, dan telah digunakan oleh para perancang komputer dalam mengarsitek komputer rancangannya. Tetapi perkembangan teknologi komputer telah memunculkan suatu ide baru dalam pengklasifikasian komputer era berikutnya, untuk itu diperkenalkanlah suatu klasifikasi arsitektur pemrosesan paralel, klasifikasi jenis ini diperlihatkan pada gambar 1.1.



Gambar 1.1 Klasifikasi arsitektur pemrosesan paralel

Multiprocessor dapat dipandang sebagai sebuah komputer paralel yang terdiri dari beberapa prosesor terinterkoneksi yang dapat menggunakan sistem memori secara bersama-sama. Masing-masing prosesor diatur untuk melakukan berbagai tugas yang berbeda dari sebuah program atau juga melakukan berbagai program yang berbeda secara serempak. Gambar 1.2 menunjukkan blok diagram dari sebuah multiprosesor.

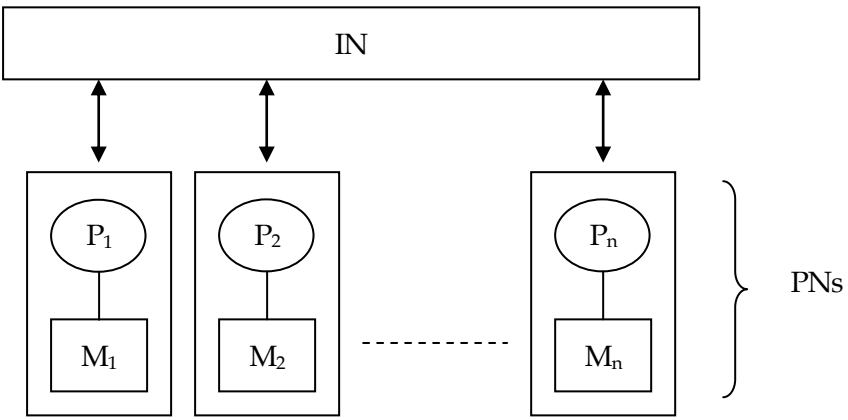


Gambar 1.2 Blok diagram multiprosesor

Keterangan : masing-masing prosesor didenotasikan sebagai P_1, P_2, …, dan P_n, dan modul memori didenotasikan sebagai M_1, M_2, …, dan M_m. Jaringan interkoneksi (IN) menghubungkan berbagai prosesor ke beberapa modul memori.

Multicomputer dapat dipandang sebagai sebuah komputer paralel yang memiliki ciri pada masing-masing prosesor memiliki memori lokalnya sendiri. Dalam sistem multikomputer memori utama didistribusikan secara private pada masing-masing prosesor, atau dengan kata lain prosesor hanya dapat mengalamatkan informasi secara langsung pada memori lokalnya saja, dan tidak dapat

mengalamatkan pada memori lokal lainnya. Gambar 1.3 menunjukkan blok diagram multikomputer.



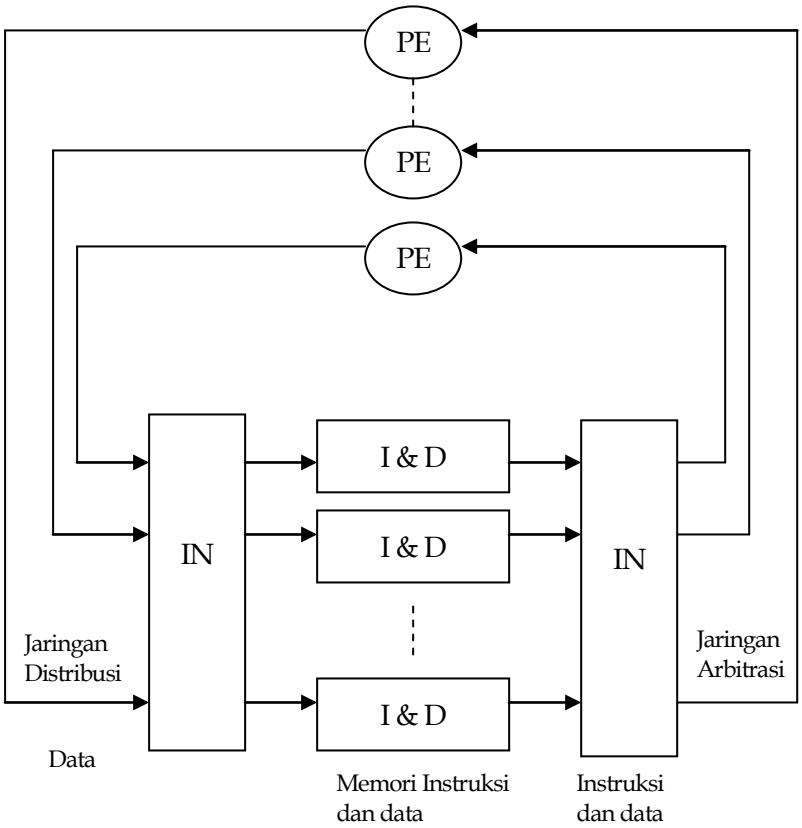
Gambar 1.3 Blok diagram multikomputer

Keterangan : pada gambar terdapat *n* node pemrosesan (processing nodes/PNs). Masing-masing PNs terdiri dari satu prosesor dan satu modul memori.

Multi-multiprocessor mengkombinasikan berbagai fitur multiprosesor tertentu dan multikomputer. Dapat juga dikategorikan sebagai multikomputer jika masing-masing PNs merupakan multiprosesor.

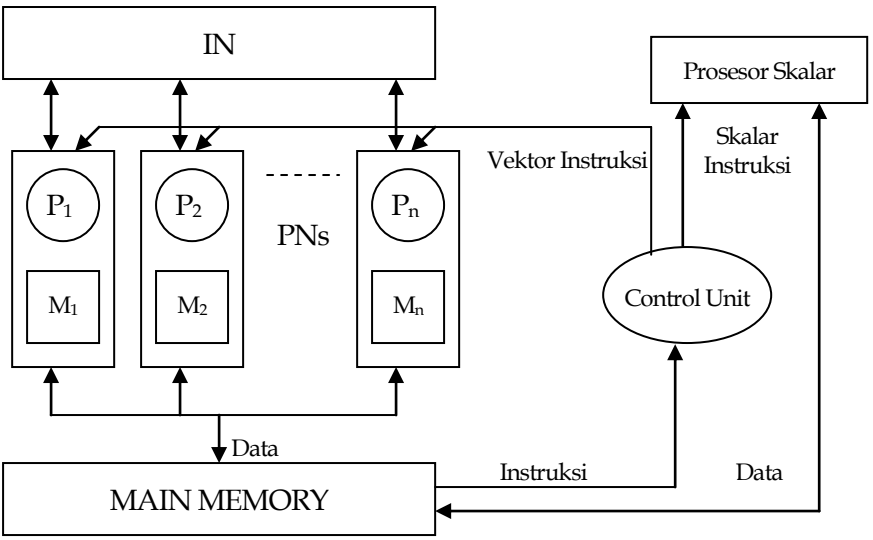
Data flow machine : pada arsitektur aliran data (data flow) sebuah instruksi dikatakan siap untuk pengeksekusian ketika data dengan operand-nya juga telah tersedia. Ketersedian data dapat dicapai dengan penyaluran berbagai hasil dari pengeksekusian berbagai instruksi sebelumnya ke dalam berbagai operand dari berbagai instruksi yang masih sedang menunggu. Bentuk penyaluran suatu aliran data, bertugas memicu pengeksekusian berbagai instruksi. Oleh karena itu, pengeksekusian instruksi selalu menghindari tipe aliran counter program terkontrol yang ditemukan di dalam mesin Von Neumann.

Instruksi data flow secara murni merupakan "self-contained." Artinya mereka tidak mengalamatkan berbagai variabel dalam memori global yang dipakai secara bersama-sama. Di dalam mesin data flow, eksekusi dari sebuah instruksi tidak mempengaruhi instruksi lain yang siap untuk dieksekusi. Tetapi beberapa instruksi yang telah siap dieksekusi akan secara serempak dieksekusikan. Gambar 1.4 mengilustrasikan blok diagram suatu mesin data flow.



Gambar 1.4 Diagram blok sebuah mesin data flow

Gambar 1.5 merepresentasikan struktur generik dari sebuah prosesor array (array processor).

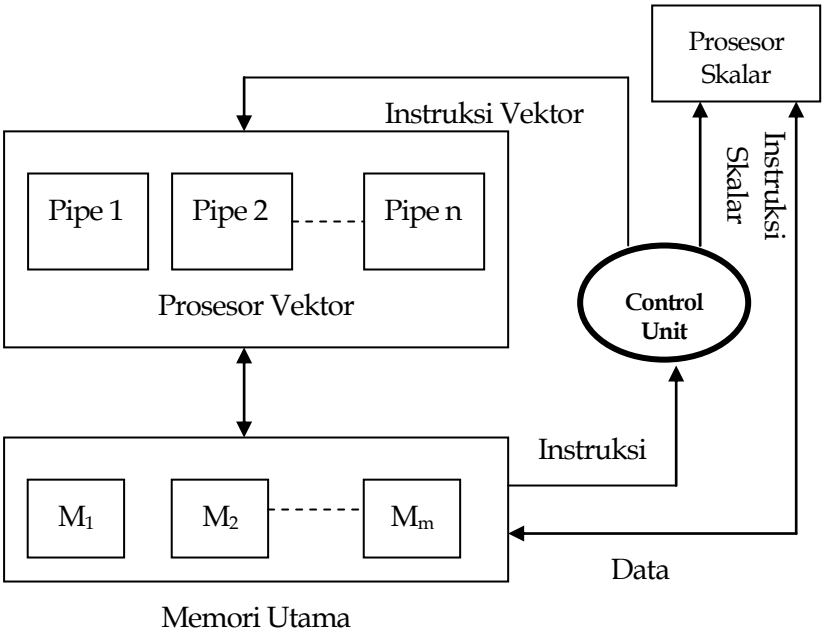


Gambar 1.5 Diagram blok sebuah prosesor array

Sebuah prosesor array terdiri dari sekumpulan PNs (Processing Nodes) dan sebuah prosesor skalar yang beroperasi di bawah suatu unit kontrol tersentralisasi. Unit kontrol mengambil dan mendekode berbagai instruksi dari memori utama dan selanjutnya instruksi dikirim ke salah satu prosesor skalar atau PNs, tergantung dari tipe instruksi tersebut. Jika instruksi yang diambil merupakan instruksi skalar maka instruksi tersebut akan dikirim ke prosesor skalar, sebalinya, instruksi akan dimuat ke berbagai PNs. Keseluruhan PNs mengeksekusikan instruksi yang sama secara serempak pada berbagai data yang disimpan dalam lokal memori-nya. Oleh karena itu, sebuah prosesor array membutuhkan cukup satu program untuk mengkontrol seluruh PNs dalam sistem, dan memastikan untuk tidak menduplikasi kode program pada masing-masing PN.

Sebuah prosesor vektor pipeline dapat memproses operand vektor (berbagai aliran data yang kontinu) secara efektif. Hal ini

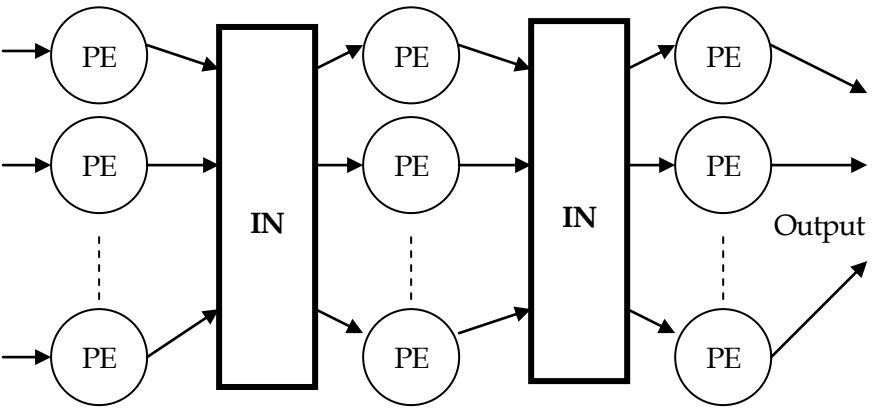
merupakan perbedaan mendasar antra prosesor vektor atau prosesor array dengan prosesor vektor pipeline. Prosesor array merupakan pengendalian oleh instruksi, sedangkan prosesor vektor pipeline dikendalikan oleh sekumpulan aliran data yang kontinu.



Gambar 1.6 Diagram blok prosesor vektor pipeline

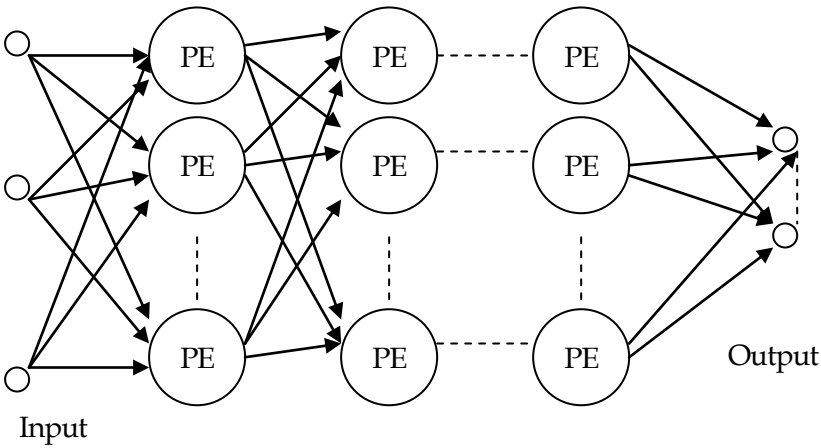
Ada dua prosesor utama : sebuah prosesor skalar dan lainnya prosesor vektor, kedua prosesor tersebut menyandarkan pada control unit secara terpisah untuk memberikan berbagai instruksi yang akan dieksekusi. Prosesor vektor menangani eksekusi berbagai instruksi vektor dengan menggunakan pipeline, dan prosesor skalar berurusan dengan eksekusi dari instruksi skalar. Bagian control unit mengambil dan mendekodekan berbagai instruksi dari memori utama dan selanjutnya mengirim instruksi-instruksi tersebut ke salah satu proseor skalar atau prosesor vektor, tergantung tipe data yang dieksekusi. Prosesor vektor pipeline

menggunakan beberapa modul memori untuk menyediakan sekumpulan aliran data pada pipeline.
Gambar 1.7 merepresntasikan struktur generik dari suatu array systolic.



Gambar 1.7 Diagram blok array systolic

Di dalam array sistolic terdapat sejumlah besar elemen-elemen pemrosesan (PEs) identik. Masing-masing PE memiliki penyimpanan lokal terbatas, dan untuk tidak membatasi jumlah PEs yang ditempatkan di dalam array, maka masing-masing PE hanya dimungkinkan terhubung ke PEs tetangganya melalui jaringan interkoneksi. Oleh karena itu, seluruh PEs disusun di dalam bentuk pipeline yang teroganisir, seperti array dua dimensi atau array linier. Di dalam array sistolic item-item data dan/atau aliran hasil partial yang melalui PEs selama waktu eksekusi akan memuat beberapa siklus pemrosesan. Pada masing-masing siklus, PEs melakukan operasi sederhana relatif yang sama pada item-item data dan mengirim item-item tersebut dan/atau aliran hasil partial ke PEs terdekatnya.
Gambar 1.8 merepresentasikan struktur generik dari artificial neural network (jaringan syaraf tiruan)/(ANN).



Gambar 1.8 Diagram blok jaringan syaraf tiruan (ANN)

Arsitektur alternatif lainnya sebagai arsitektur model masa depan adalah jaringan syaraf tiruan (ANN). ANN terdiri dari sejumlah PEs yang berkerja secara kompleks dan paralel. Arsitektur ini meniru cara kerja syaraf manusia yang kompleks. Ciri utamanya adalah kemampuan melakukan learning, mengadaptasikan terhadap berbagai perubahan pada lingkungan, dan mampu mengatasi berbagai gangguan yang kompleks.

Jenis arsitektur special-purpose lainnya adalah prosesor berbasis logika fuzzy. Logika fuzzy dikaitkan terhadap prinsip formal penalaran aproksimasi, sementara logika klasik bertumpu pada dua nilai true dan false. Logika fuzzy mencoba untuk mengimbangi kemampuan proses kognitif manusia, dan mengatasi beberapa kendala yang tidak dapat dilakukan oleh logika klasik. Aplikasi logika fuzzy sebagian besar diterapkan pada bidang sistem kontrol, dan lain-lain.

UNJUK KERJA DAN PENGUKURAN KUALITAS

Unjuk kerja atau performance dari suatu komputer berkaitan erat dengan kecepatan efisien yang dilakukannya dan kehandalan software dan hardware yang dimilikinya. Secara umum, kita tidak beralasan untuk berharap suatu modul tunggal dapat mengkarakteristik unjuk kerja yang dilakukan. Performance suatu komputer sangat tergantung dari interaksi berbagai komponennya dan kenyataan pada tingkat ketertarikan pengguna komputer menghadapi berbagai aspek kemampuan komputer yang berbeda.
Salah satu standar ukuran yang sering digunakan untuk merepresentasikan performance komputer adalah MIPS (million instructions per second). MIPS merepresentasikan kecepatan sebuah komputer dengan mengindikasikan sejumlah instruksi rata-rata yang dapat dieksekusi per detik. Untuk memahami arti dari instruksi rata-rata, maka perhatikan kebalikan dari ukuran MIPS, sebagai, waktu eksekusi dari sebuah instruksi rata-rata. Waktu eksekusi dari sebuah instruksi rata-rata dapat dikalkulasikan dengan menggunakan frekuensi dan waktu eksekusi untuk masing-masing kelas instruksi. Dengan menelusuri eksekusi dari sejumlah program benchmark, memungkinkan untuk menentukan berapa seringkah sebuah instruksi yang mungkin dapat digunakan di dalam suatu program. Sebagai contoh, perhatikan tabel berikut :

Kelas Instruksi	**Frekuensi Instruksi %(IF)**	**Siklus per Instruksi (CPI)**	**CPI yang diberi bobot (IF*CPI)**
Load dan Store	30,4	1,5	0,456
+ dan – (integer)	10,0	1	0,1
* dan / (integer)	3,8	10	0,38
+ dan – (Floating-point)	9,5	7	0,665
* dan / (Floating-point)	6,5	15	0,975
Logikal	3,0	1	0,03
Branch	20,0	1,5	0,3
Compare, shift	16,8	2	0,336

Siklus per instruksi rata-rata	3,242
Waktu eksekusi dari sebuah instruksi rata-rata Pada $\tau = 10 = 3{,}242 * 10 = 32{,}42$ nanodetik	
MIPS =(1/32,42) * 1000 =30,845	

MIPS setara dengan $\frac{1}{\sum(IF_i * CPI_i * \tau)} \times 1000$

Pada berbagai komputer dengan kegunaan khusus seperti komputer yang melakukan komputasi saintifik dan rekayasa (ex : prosesor vektor), sering digunakan notasi FLOPS untuk merepresentasikan jumlah operasi floating-point yang dapat dieksekusi per detik. Selanjutnya MFLOPS, untuk million, dan GFLOPS untuk giga.

Keterangan :

FLOPS = Floating-point operations per second

MFLOPS = Mega of floating-point operations per second

GFLOPS = Giga of floating-point operations per second

Selain MIPS dan FLOPS, ukuran lainnya yang sering muncul untuk merepresentasikan gambaran dari suatu sistem adalah :

1. **Throughput** dari suatu prosesor adalah ukuran yang mengindikasikan jumlah program (tugas atau request) yang dapat dieksekusi oleh prosesor per unit waktu.
2. **Utilization** dari suatu prosesor merupakan rasio waktu sibuk dan waktu yang telah dilalui pada suatu periode tertentu.
3. **Response time** adalah interval waktu antara waktu saat permintaan pertama dilakukan dan waktu pada saat pelayanan telah selesai dilakukan.
4. **Memory Bandwidth** mengindikasikan jumlah memori word yang dapat diakses dalam satu waktu.
5. **Memory Access Time** merupakan rata-rata waktu yang digunakan oleh prosesor untuk mengakses memori, sering diekspresikan dengan nanoseconds (ns).
6. **Memory size** mengindikasikan kapasitas memori, sering diekspresikan dengan istilah megabytes (Mbytes)

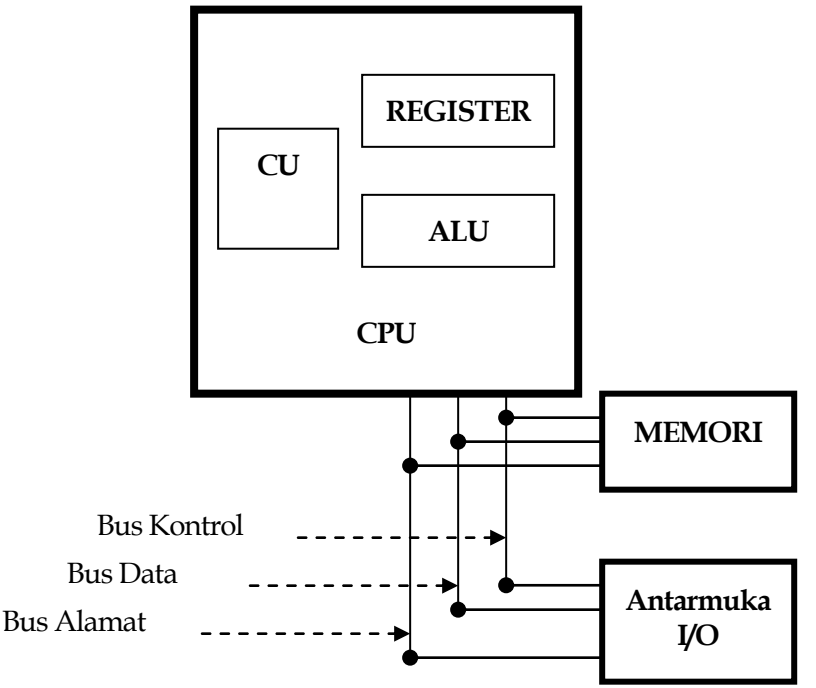
Bab 2
ARSITEKTUR VON NEUMANN

PENDAHULUAN

Konsep dasar yang melandasi arsitektur Von Neumann adalah kemampuan menyimpan instruksi di dalam memori bersamaan dengan data yang dioperasikan oleh instruksi-instruksi tersebut. Sebelum Von Neumann mengajukan jenis arsitektur ini, mesin-mesin komputasi dirancang dan dikembangkan hanya untuk tugas-tugas tunggal tertentu saja dan semua pemrograman dilakukan dengan mengatur ulang kabel-kabel sirkuit, sehingga perkerjaan pemrograman begitu sangat membosankan, terlebih-lebih jika terjadi suatu kesalahan, akan sulit untuk mendeteksi letak-letak kesalahannya dan terlebih lagi memperbaiki kesalahan-kesalahan tersebut.

Arsitektur Von Neumann disusun berdasarkan tiga komponen yang berbeda : satu unit CPU, memori, dan antarmuka I/O. gambar 2.1 mengilustrasikan salah satu dari berbagai cara yang mungkin menginterkoneksi berbagai komponen tersebut.

1. CPU, perannya sebagai otak sistem komputasi, meliputi tiga komponen utama : control unit (CU), satu atau lebih ALU, dan berbagai register. CU menentukan perintah instruksi yang harus dieksekusikan dan mengontrol pencarian berbagai operand yang tepat.



Gambar 2.1 Komponen dasar komputer

CU juga bertugas menginterpretasi berbagai instruksi mesin. Eksekusi dari masing-masing instruksi ditentukan oleh sebarisan sinyal kontrol yang dihasilkan oleh CU. Dengan kata lain, CU memerintahkan aliran informasi melalui sistem dengan memberikan sinyal-sinyal kontrol tertentu ke berbagai komponen yang berbeda. Masing-masing operasi yang disebabkan oleh sinyal kontrol sering diistilahkan sebagai mikrooperasi (MO). ALU bertugas

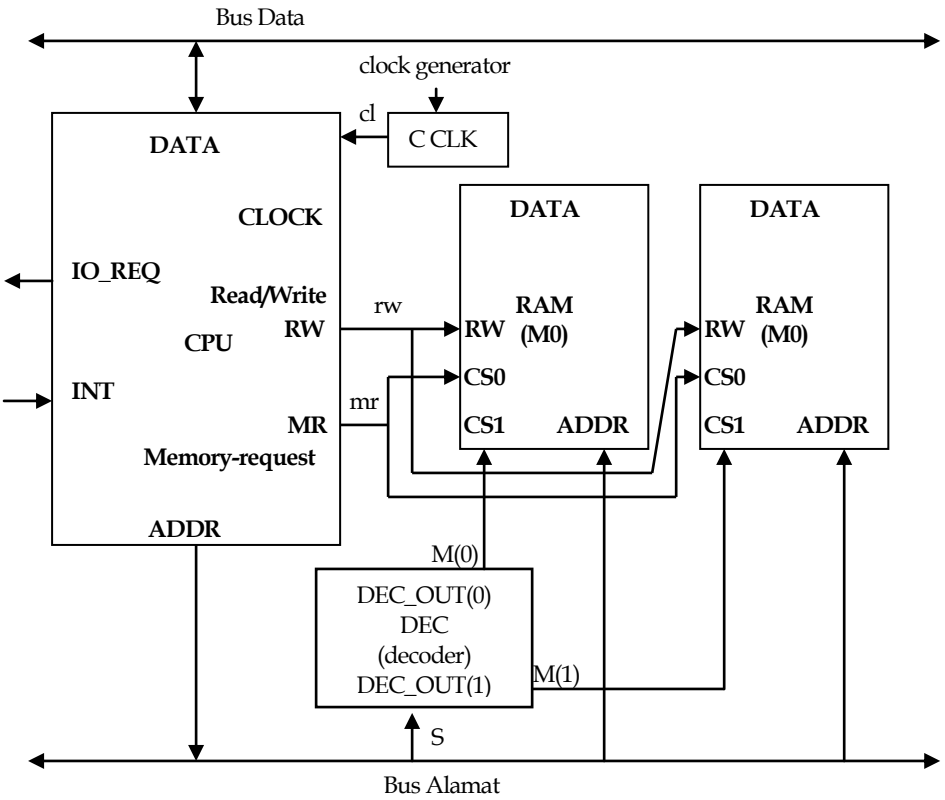
melakukan berbagai operasi matematika dan operasi Boolean. Register merupakan media penyimpanan atau lokasi penyimpanan yang bersifat sementara. Pada umumnya register-register sering dikemas dalam satu chip yang sama, dan secara langsung dikoneksikan ke CU, sehingga register memiliki waktu akses yang lebih cepat dibandingkan memori lainnya (lihat 2.1). CPU yang dimplemetasikan pada chip tunggal diistilahkan sebagai mikroprosesor.

2. Memori digunakan untuk menyimpan instruksi program dan data. Dua jenis umum memori yang sering digunakan adalah RAM dan ROM. RAM bertugas melakukan penyimpanan data dan berbagai program yang dieksekusi oleh mesin secara sementara, atau dengan kata lain isi dari memori RAM dapat diubah pada sebarang waktu dan terhapus ketika aliran listrik pada komputer terputus. ROM bersifat permanen, bertugas menyimpan berbagai instruksi mesin awal.
3. Antarmuka I/O memungkinkan memori komputer berinteraksi dengan perangkat output, mengirim dan menerima data. Juga memungkinkan komputer berkomunikasi ke user dan perangkat penyimpanan secondary seperti disk dan drive tape.

Selanjutnya komponen-komponen tersebut dikoneksikan melalui sekumpulan jalur-jalur sinyal yang disebut juga sebagai bus. Seperti pada gambar 2.1 bus-bus utama yang bertugas mengangkut informasi adalah bus kontrol, bus data, dan bus alamat. Masing-masing bus berisikan beberapa wire yang memungkinkan transmisi paralel suatu informasi di antara berbagai komponen hardware. Bus alamat bertugas mengidentifikasikan salah satu dari lokasi memori atau perangkat I/O. Bus data bersifat dua arah, mengirim data dari atau ke suatu komponen. Bus kontrol memuat sinyal yang memungkinkan CPU untuk berkomunikasi dengan memori atau perangkat I/O.

PERANCANGAN MIKROKOMPUTER SEDERHANA MENGGUNAKAN VHDL

Komputer yang memiliki CPU sebagai mikroprosesor-nya disebut mikrokomputer. Mikrokomputer memiliki fisik yang kecil dan harga yang tidak begitu mahal. Personal Computer yang sering kita gunakan sehari-hari termasuk kategori mikrokomputer. Gambar 2.2 merepresentasikan berbagai komponen utama pada sebuah komputer yang sederhana.



Gambar 2.2 Sistem komputer sederhana

Mikrokomputer pada gambar 2.2 terdiri dari satu unit CPU, satu unit clock generator, satu dekoder, dan dua modul memori. Masing-masing modul memori terdiri dari 8 word, di mana masing-masing word berukuran 8 bit. Nilai word inilah yang mengindikasikan berapa banyak data yang dapat diproses pada sebarang waktu oleh komputer. Mikrokomputer yang memiliki dua modul memori mempunyai total 16 memori word 8 bit. Bus alamat memiliki kapasitas 4 bit untuk mengalamatkan data 16 word. Tiga lsb (least significant bits) pada bus alamat dikoneksikan secara langsung ke berbagai modul memori, dan msb (most significant bits) atau satu bit yang berposisi di paling kiri dikoneksikan ke *select line* pada dekoder (S). Jika nilai bit msb setara dengan 0, maka M0 akan dipilih, sebaliknya Jika nilai bit msb setara dengan 1, maka M1 akan dipilih. Alamat 0 hingga 7 (0000 hingga 0111) berhubungan ke nilai-nilai word di dalam modul memori M0, dan alamat 8 hingga 15 (1000 hingga 1111) berkaitan dengan modul memori M1. Gambar 2.3 merepresentasikan pandangan struktur mikrokomputer VHDL.

```
architecture structure_view of microprocessor is
component CPU
   port (DATA: inout tri_vector (0 to 7);
          ADDR: out bit vector (3 downto 0);
          CLOCK, INT: in bit;
          MR, RW, IO_REQ: out bit);
   end component;
   component RAM
   port (DATA: inout tri_vector(0 to 7);
          ADDR: in bit_vector(2 downto O);
          CSO, CS1, RW: in bit);
   end component;
   component DEC
   port(DEC_IN: in bit: DEC_OUT: out bit_vector(0 to 1):
   end component;
   component CLK
   port (C: out bit);
   end component;
   signal M: bit_vector (0 to 1):
   signal cl, mr, rw: bit;
 begin
   PROCESSOR: CPU portmap (DATA, ADDR, cl, INT,mr,rw,
   IO_REQ):
   M0: RAM portmap (DATA, ADDR(2 downto 0),mr,M(0), rw);
   M1: RAM portmap (DATA, ADDR(2 downto 0),mr,M(1), rw);
   DECODER: DEC portmap ADDR(3), M);
   CLOCK: CLK portmap (cl);
 end struccure_view;
```

Gambar 2.3 Representasi struktur mikrokomputer sederhana

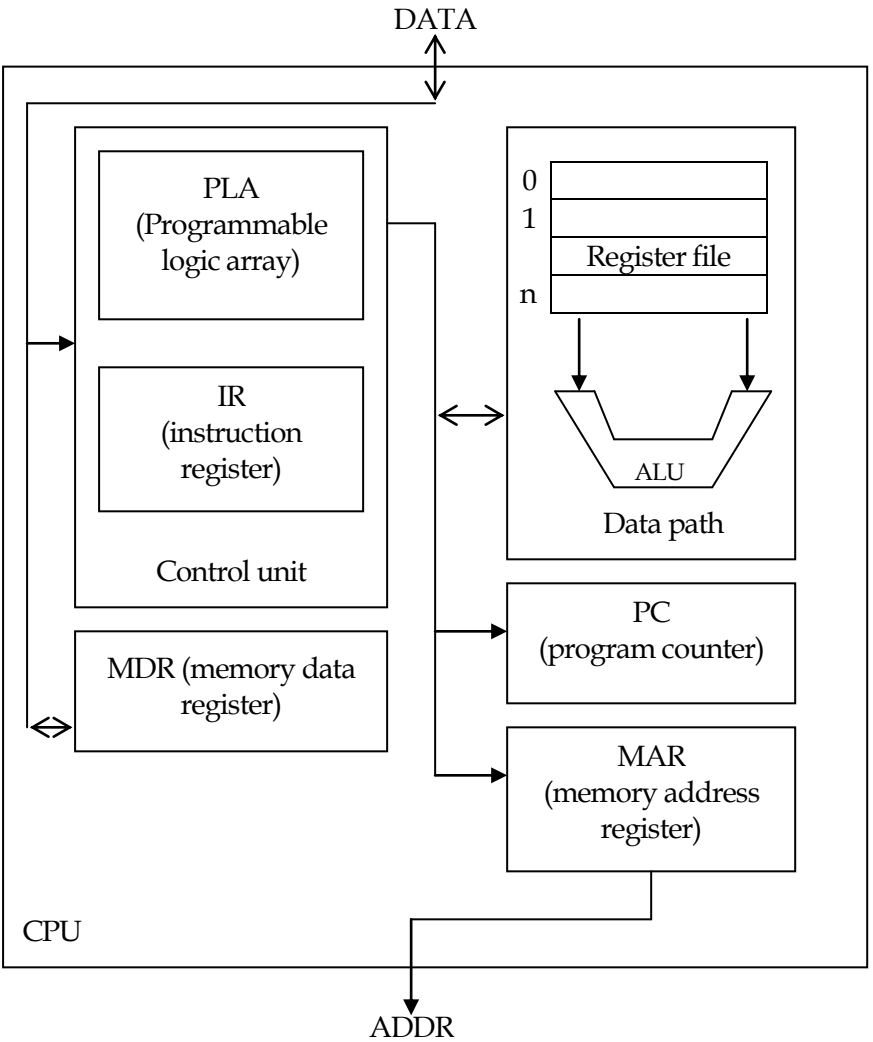
Gambar 2.4 merepresentasikan fungsi atau pandangan prilaku sebuah modul memori yang juga sering diistilahkan sebagai random-access memory (RAM).

```
architecture behavioral_view of RAM In
begin
  process
    type memory_unit is array(0 to 7) of bit_vector(0 to 7);
    variable memory: memory_unit;
    begin
    while (CS0='1' and CS1='1') loop
    case RW is --RW = 0 means read operation
               --RW = 1 means write operation
    when '0' => DATA <=memory (intval(ADDR)) after 50 ns;
    when '1'=> <=memory (intval(ADDR)) after 60 ns;
    end case;
    wait on CSO, CS1, DATA, ADDR, RW;
    end loop;
    wait on CSO, CSI;
    end process;
    end behavioral_view;
```

Gambar 2.4 Representasi prilaku RAM 8 by 8 (8 word dengan masing-masing word memiliki kapasitas 8 bit)

Pandangan struktur mikrokomputer mendeskripsikan sistem dengan mendeklarasikan berbagai komponen utamanya dan mengkoneksikan keseluruhan komponen ke sekumpulan sinyal-sinyal tertentu. Pandangan struktur mikrokomputer dibagi ke dalam dua bagian : bagian deklarasi, yang hadir pada gambar 2.3 sebelum keyword begin, dan bagian rancangan yang hadir pada gambar 2.3 sesudah keyword begin. Bagian deklarasi memuat empat statemen komponen dan dua statemen sinyal. Masing-masing statemen komponen mendefinisikan berbagai port input/ouput untuk masing-masing komponen mikrokomputer. Statemen sinyal mendefinisikan sekumpulan sinyal yang digunakan untuk menginterkoneksikan berbagai komponen.

Gambar 2.5 merepresentasikan komponen-komponen utama dari suatu unit CPU.



Gambar 2.5 Model CPU yang sederhana

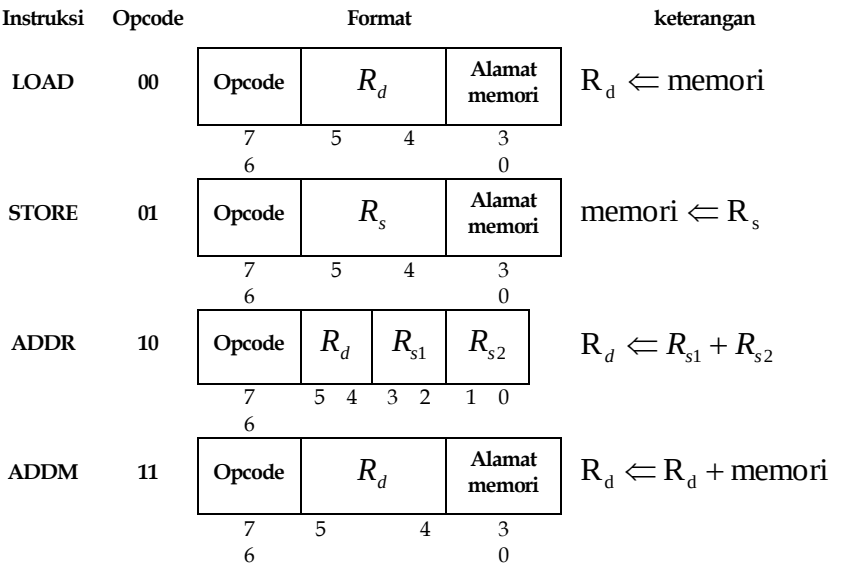
Komponen-komponen yang terdapat pada CPU dalam gambar 2.5 terdiri dari data path, CU, dan berbagai register yang dihubungkan ke register file. Data path terdiri dari ALU (arithmetic logic unit),

dan berbagai register. CPU berkomunikasi dengan berbagai modul memori melalui MDR (memory data register) dan MAR (memory address register). Program counter (PC) digunakan untuk menyimpan alamat instruksi berikutnya. Register instruksi (IR) digunakan untuk menyimpan sementara sebuah instruksi yang sedang didekode dan dieksekusikan.

Untuk mengekspresikan fungsi kontrol unit, asumsikan bahwa mikrokomputer memiliki empat instruksi yang berbeda. Masing-masing instruksi memiliki format berikut :

Opcode	Operand	…	Operand

Opcode (operation code) bertugas menentukan fungsi instruksi, dan operand memberikan alamat berbagai item data. Gambar 2.6 merepresentasikan opcode dan jenis operand untuk masing-masing instruksi.



Gambar 2.6 Format instruksi CPU

Instruksi LOAD bertugas memuat memori word ke dalam register. Instruksi STORE menyimpan isi register ke dalam memori word.

Instruksi ADDR menambah isi dua register dan menyimpan hasil pertambahan tersebut ke register ke tiga. Instruksi ADDM menambah isi register dengan memori word dan menyimpan hasil pertambahannya di dalam register.

Untuk memahami peran dari control unit, mari kita uji pengeksekusian sebuah penggal program berikut :

LOAD 1, 13

ADDM 1,14

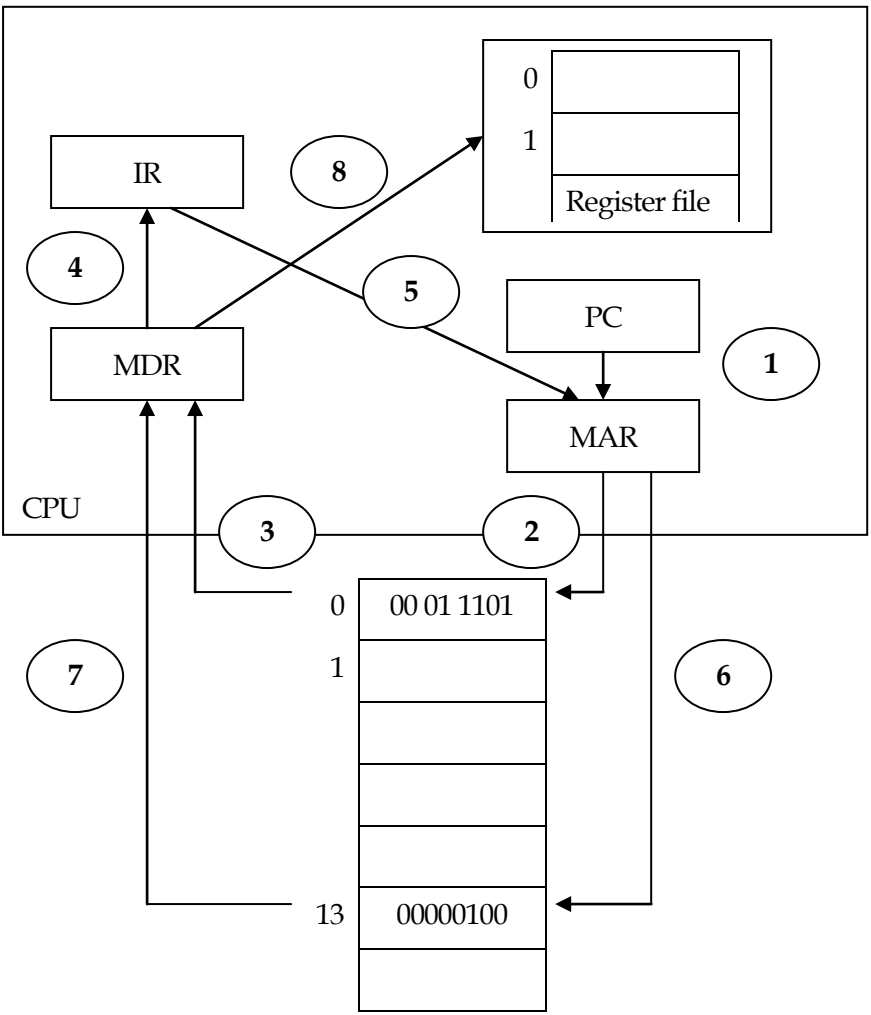
STORE 1,15

Aumsikan lokasi memori 13 dan 14 memuat nilai 4 dan 2. dan asumsikan pula bahwa kode biner listing program disimpan ke dalam 3 word memori pertama. Maka, isi memori akan berupa :

0	00 01 1101
1	11 01 1110
2	01 01 1111
13	00000100
14	00000010
15	

Memori

Aliran pengalamatan dan data untuk eksekusi instruksi LOAD diilustrasikan sesuai gambar 2.7.



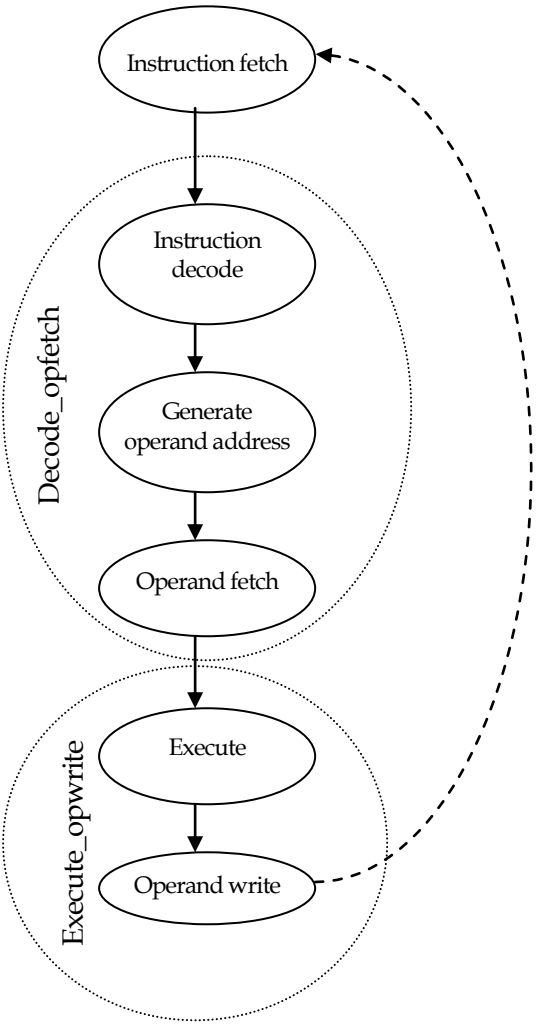
Gambar 2.7 Aliran berbagai alamat dan data untuk eksekusi instruksi LOAD

Proses awal, alamat dari instruksi pertama dimuat ke dalam program counter (PC), dan selanjutnya isi PC disalin ke dalam MAR, dan dari MAR ke bus alamat. Control unit meminta suatu operasi read dari unit memori. Pada saat yang bersamaan, isi PC

ditambah 1 untuk menunjukkan ke instruksi berikut yang akan dieksekusi. Unit memori mengambil alamat lokasi memori yang diminta dari bus alamat, setelah waktu tunggu tertentu, unit memori mentransfer isi lokasi memori yang diminta (00011101) ke MDR melalui bus data. Kemudian isi MDR disalin ke register instruksi (IR). Register IR berfungsi mendekodekan instruksi. Control unit menguji dua bit terkiri dari IR dan menentukan bahwa instruksi tersebut merupakan suatu operasi load. Control unit selanjutnya menyalin 4 bit terkanan IR ke dalam MAR, sehingga IR sekarang bernilai 1101, yang merepresentasikan alamat ke-13 (dalam bentuk desimal). Isi lokasi memori 13 merupakan instruksi LOAD dari memori. Selanjutnya isi MDR disalin ke dalam register R_1. Dalam waktu ini, eksekusi instruksi LOAD telah lengkap.

Proses selanjutnya berlanjut hingga keseluruhan instruksi telah dieksekusi. Pada akhir eksekusi, nilai 6 disimpan ke dalam lokasi memori 15.

Secara umum, proses eksekusi dari sebuah instruksi dapat dibagi ke dalam tiga fase utama, seperti ditunjukkan pada gambar 2.8. Fase-fase tersebut terbagi ke dalam : fase *instruction fetch*, *decode_opfetch*, dan *execute_opfetch*. Pada fase *instruction fetch*, sebuah instruksi diambil dari memori dan disimpan di dalam register instruksi. Barisan aksi yang dibutuhkan untuk melakukan proses tersebut dapat dikelompokkan ke dalam tiga langkah utama. Pada fase *decode_opfetch*, instruksi di dalam register instruksi didekodekan, dan jika instruksi membutuhkan sebuah operand, instruksi diambil dan ditempatkan ke dalam suatu lokasi yang ditentukan. Pada fase *execute_opfetch*, melakukan operasi yang sesuai dan selanjutnya disimpan di dalam lokasi yang dispesifikasikan.



Gambar 2.8 Fase-fase utama proses eksekusi sebuah instruksi

4.1 CONTROL UNIT

Secara umum, ada dua pendekatan utama untuk merealisasikan suatu control unit : sirkuit *hardwired* dan perancangan mikroprogram.

Hardwired control unit : pendekatan hardwired dalam mengimplementasikan suatu control unit biasanya

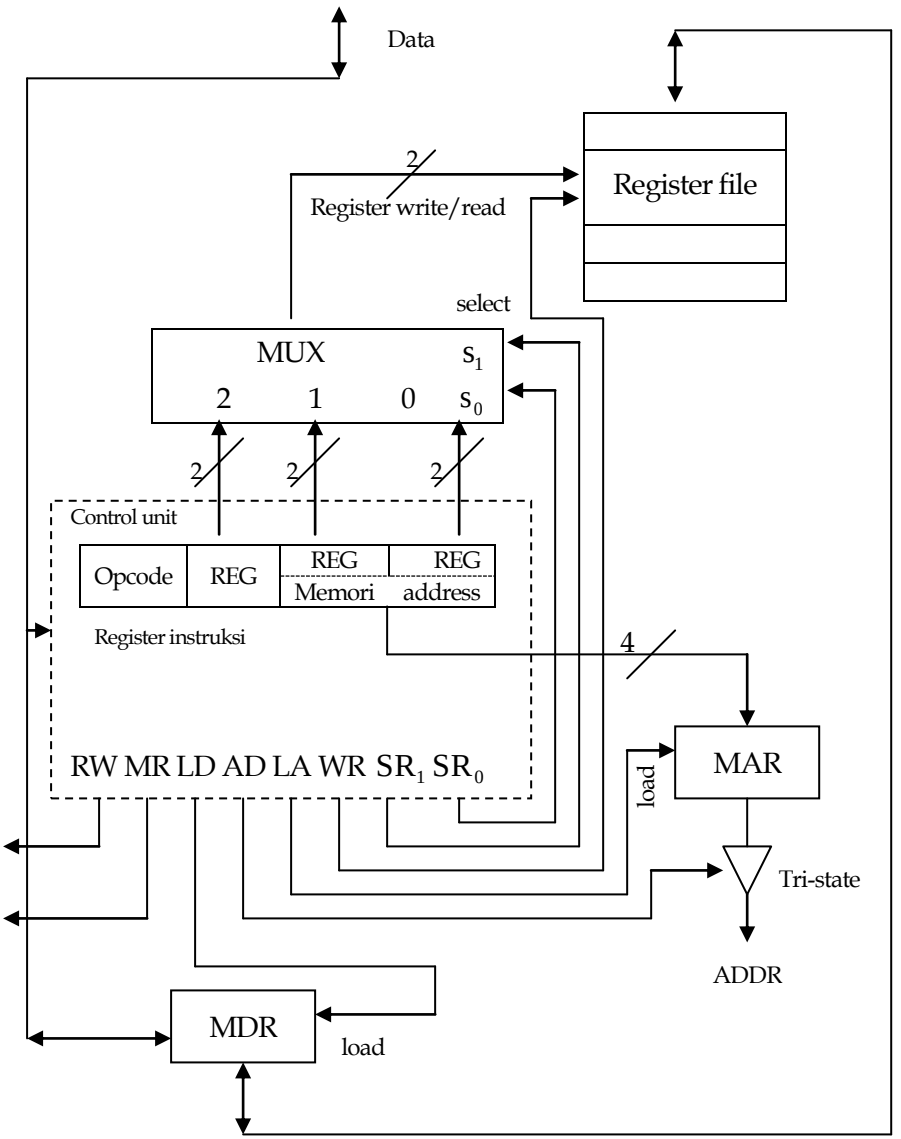
direpresentasikan sebagai sirkuit sekuensial berbasis state-state yang berbeda di dalam suatu mesin, dengan memberikan sebarisan sinyal kontrol pada masing-masing state untuk mengendalikan suatu operasi komputer. Untuk contoh, perhatikan perancangan sirkuit hardwired untuk instruksi LOAD pada satu unit mikrokomputer yang sederhana. Instruksi memiliki format berikut :

$$\text{LOAD } R_d, \text{Address}$$

di mana instruksi akan memuat isi memori word ke dalam register R_d.

Gambar 2.13 merepresentasikan berbagai register utama dan sinyal kontrol $\text{RW}, \text{MR}, \text{LD}, \text{AD}, \text{LA}, \text{WR}, \text{SR}_0$ dan SR_1. Fungsi dari masing-masing sinyal kontrol didefinisikan sebagai berikut :

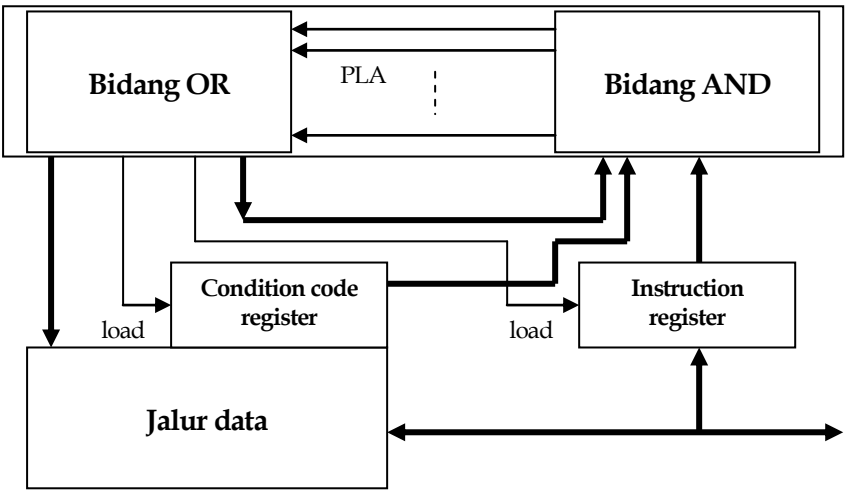
RW	:	Melakukan operasi read/write dari/ke memori (RW=0 berarti operasi yang dipilih adalah read, dan RW=1 berarti write)
MR	:	Meng-enable-kan terminal *chip select* (CS0) dari memori
LD	:	Memuat data dari bus data ke MDR
AD	:	Memuat alamat dari MAR ke bus alamat
LA	:	Memuat 4 bit terkanan dari IR ke MAR
WR	:	Melakukan operasi read/write dari/ke register file (WR=0 berarti operasi yang dipilih adalah read, dan WR=1 berarti write)
SR_0 dan SR_1	:	Memilih 2 bit dari IR sebagai alamat register file



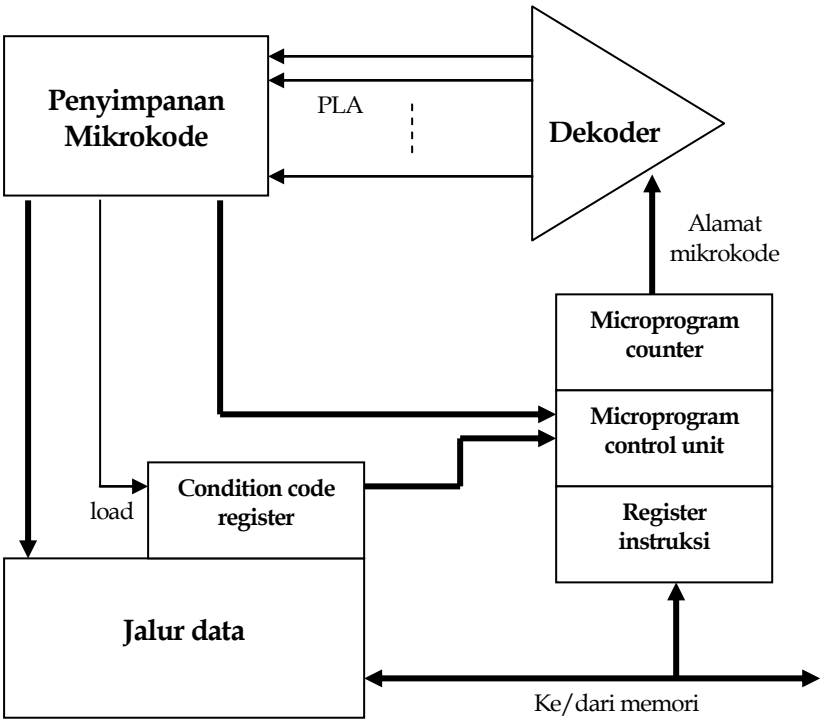
Gambar 2.13 Sinyal kontrol untuk beberapa porsi model CPU sederhana (yang diilustrasikan pada gambar 2.5)

Operasi load pada awalnya melakukan loading 4 lsb (least significant bits) dari register instruksi (IR) ke dalam MAR. Selanjutnya data yang sesuai diambil dari memori dan dimuat ke MDR. Kemudian register tujuan R_d ditentukan dengan mengatur s_0 dan s_1 bernilai 0 dan 1, berturut-turut. Operasi dilengkapi dengan memindahkan isi MDR ke register yang ditentukan oleh SR_0 dan SR_1.

Microprogrammed control unit. Untuk memecahkan ketidakfleksibelan pendekatan hardwired, pada tahun 1951 Wilkes mengajukan suatu teknik yang dikenal sebagai mikropemrograman. Saat ini mikropemrograman memainkan peranan penting sebagai metode perancangan yang digunakan luas, dengan konsep yang begitu fleksibel. Di dalam mikropemrograman (microprogramming), masing-masing instruksi mesin diterjemahkan ke sebarisan mikro-instruksi yang memicu sinyal kontrol pada *resource* mesin. Sinyal menginitialisasi operasi dasar komputer. Mikro-instruksi merupakan berbagai pola bit yang disimpan pada suatu memori terpisah yang dikenal sebagai media penyimpanan mikrokode.



Gambar 2.15 Struktur CPU (Hardwired control unit)



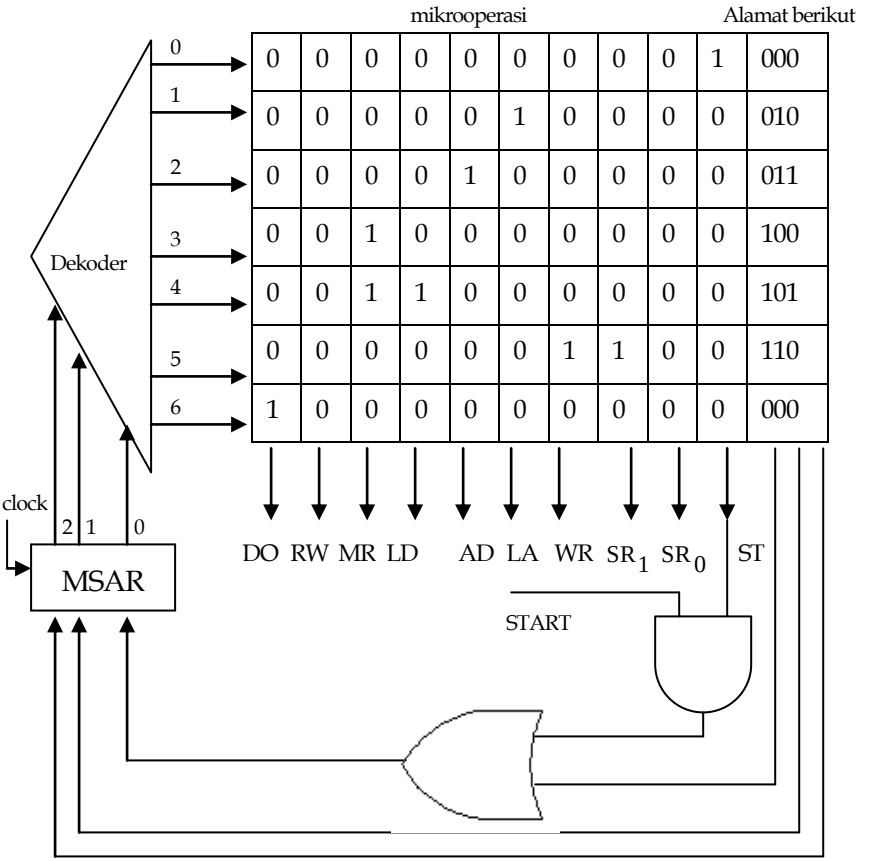
Gambar 2.16 Struktur CPU (microprogrammed control unit)

Perancangan word mikro-instruksi, ada beberapa kriteria yang harus diperhatikan dalam merancang format mikro-instruksi :

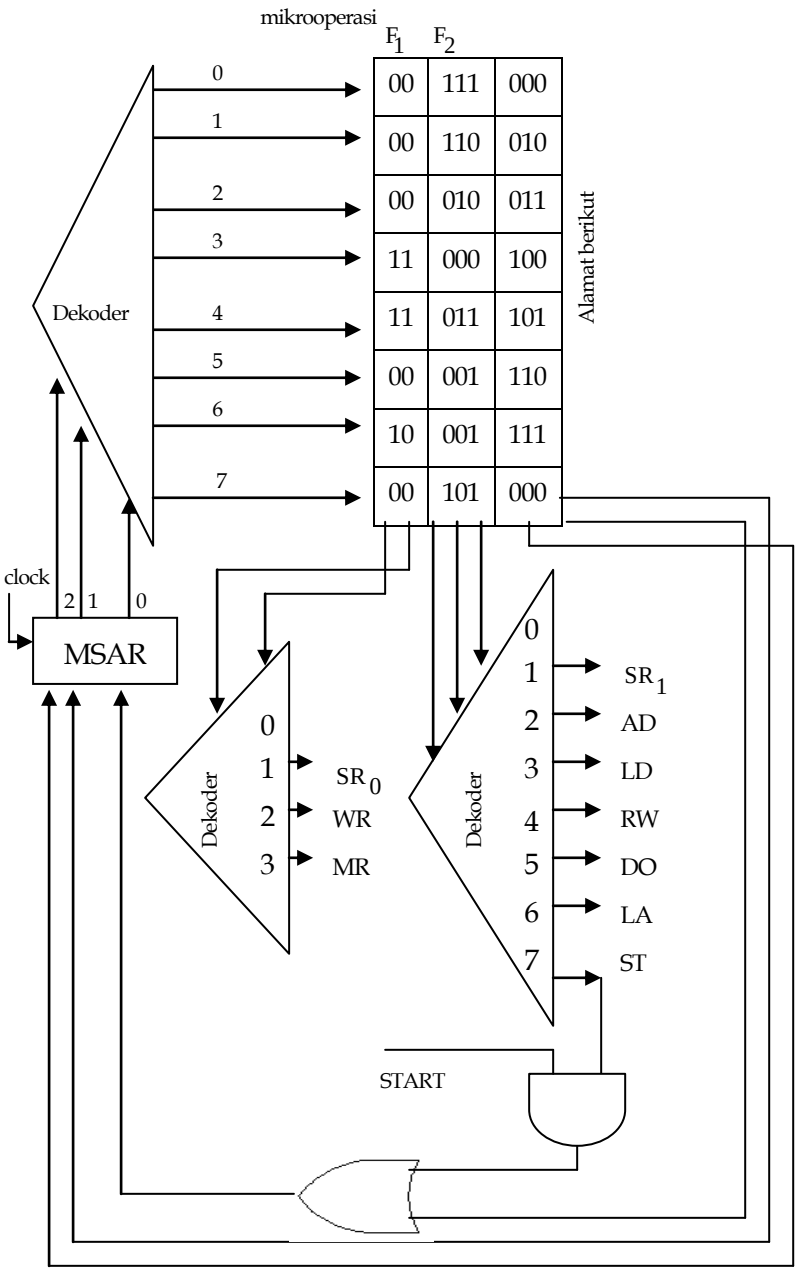
1. Minimasi ukuran word media penyimpanan mikrokode.
2. Minimasi ukuran mikroprogram.
3. Maksimalisasi tingkat fleksibilitas penambahan atau perubahan mikro-instruksi.
4. Maksimalisasi konkurensi mikrooperasi.

Secara umum, sebuah mikroword memuat dua field, field mikrooperasi dan field alamat selanjutnya. Salah satu rancangan ekstrim untuk mikrooperasi adalah dengan memberikan suatu nilai

bit ke masing-masing sinyal kontrol, perancangan ini diistilahkan sebagai perancangan horisontal, karena banyak bit-bit yang dibutuhkan, sehingga menghasilkan field mikrooperasi yang lebar atau horisontal. Rancangan jenis lain adalah perancangan vertikal. Dikatakan vertikal karena relatif sedikit bit yang dibutuhkan, sehingga hasil yang diperoleh terbatas. Di dalam rancangan vertikal terdapat berbagai subfield yang dienkodekan, di mana masing-masing subfield dapat menerima hanya satu sinyal kontrol di dalam sekelompok sinyal kontrol tertentu.



Gambar 2.17 Mikroprogram horisontal untuk operasi LOAD

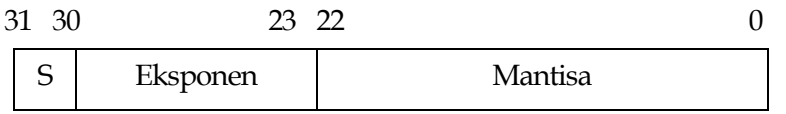


Gambar 2.18 Mikroprogram vertikal untuk operasi LOAD

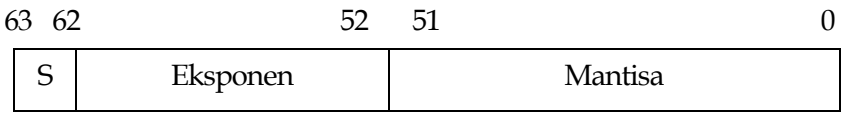
Keterangan : MSAR singkatan dari Microcode Storage Address Register.

ALU : REPRESENTASI FLOATING-POINT

Bilangan floating-point, berisikan 2 bagian, yaitu mantisa (signifikan atau pecahan), dan eksponen. Gambar 2.19 memperlihatkan bentuk 4 byte dan 8 byte dari bilangan floating-point yang disimpan di dalam sistem Intel. Bilangan floating-point atau bilangan real 4 byte disebut *single-precision*, dan Bilangan floating-point atau bilangan real 8 byte disebut *double-precision*.



(a)



(b)

Gambar 2.19 (a) Presisi tunggal menggunakan bias 7FH, dan (b) Presisi Ganda menggunakan bias 3FFH

Mantisa 24-bit memiliki bit satu yang tersembunyi (implied), yang memungkinkan mantisa untuk mewakili 24-bit walau hanya disimpan 23-bit. Bit yang disembunyikan adalah bit pertama dari bilangan yang dinormalisasi. Pada saat menormalisasi bilangan, bit ini diatur sehingga nilainya sekurang-kurangnya 1, tetapi kurang dari 2. sebagai contoh, jika 12 diubah ke biner (1102_2), maka

dinormalisasi dan hasilnya $1{,}1 \times 2^3$. Bit satu yang terdapat di depan tanda koma akan disembunyikan, atau tidak disimpan di dalam bagian mantisa.

Eksponen disimpan dalam eksponen terbias (*biased exponent*). Untuk presisi tunggal, Bias 7FH (127) akan dijumlahkan ke eksponen sebelum disimpan ke dalam tempat eksponen, dan untuk presisi ganda sebesar 3FFH (1023).

Ada 2 pengecualian mengenai aturan-aturan yang diterapkan mengenai bilangan floating-point. Angka 0,0 disimpan semuanya sebagai nol. Bilangan tak hingga disimpan dalam eksponen sebagai satu, dan dalam mantisa semuanya sebagai nol. Bit tanda menunjukkan bilangan tak hingga tersebut bernilai positif atau negatif.

Kasus 1

Bagaimana bilangan +12 disimpan dalam bentuk presisi tunggal.

Penyelesaian :

Desimal	Biner	Normal	Tanda
+12	1100	$1{,}1 \times 2^3$	0

s	Eksponen							
0	1	0	0	0	0	0	1	0
31	30	29	28	27	26	25	24	23

Mantisa																							
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	

Penjelasan :

+ = 0, pangkat pada normalisasi adalah 3 atau dalam bentuk biner sama dengan 11_2. Selanjutnya untuk presisi tunggal eksponen dapat diperoleh dengan menambahkan 11_2 +7FH = $11_2 + 1111111_2 = 10000010_2$.

Kasus 2

Bagaimana bilangan -12 disimpan dalam bentuk presisi tunggal.

Penyelesaian :

Desimal	Biner	Normal	Tanda
-12	1100	$-1{,}1 \times 2^3$	1

s	Eksponen							
1	1	0	0	0	0	0	1	0
31	30	29	28	27	26	25	24	23

Mantisa																							
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	

Kasus 3

Bagaimana bilangan +100 disimpan dalam bentuk presisi tunggal.

Penyelesaian :

Desimal	Biner	Normal	Tanda

+100	1100100	$1{,}1001 \times 2^6$	0

s	Eksponen							
0	1	0	0	0	0	1	0	1
31	30	29	28	27	26	25	24	23

Mantisa																							
1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	

Penjelasan :

Konversi bilangan 100 menjadi bilangan biner adalah sebagai berikut :

100/2	Sisa 0	3/2	Sisa 1
50/2	Sisa 0	1/2	Sisa 1
25/2	Sisa 1	0	
12/2	Sisa 0		
6/2	Sisa 0		1100100

Kasus 4

Bagaimana bilangan -1,75 disimpan dalam bentuk presisi tunggal.

Penyelesaian :

Desimal	Biner	Normal	Tanda

-1,75	1,11	$-1{,}11 \times 2^0$	1

s	Eksponen							
1	0	1	1	1	1	1	1	1
31	30	29	28	27	26	25	24	23

Mantisa																							
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	

Penjelasan :

Konversi bilangan -1,75 menjadi bilangan biner adalah sebagai berikut: (kita ambil 0,75, 1 desimal = 1 biner), sampai di sini kita sudah dapat menyatakan sebagai 1,…

0,75 x 2	
0,5 x 2	Digit 1
1,0	Digit 1

1,11

Kasus 5

Bagaimana bilangan +0,25 disimpan dalam bentuk presisi tunggal.

Penyelesaian :

Desimal	Biner	Normal	Tanda

+0,25	0,01	$1,\times 2^{-2}$	0

s	Eksponen							
0	0	1	1	1	1	1	0	1
31	30	29	28	27	26	25	24	23

Mantisa																							
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	

Penjelasan :

Konversi bilangan +0,25 menjadi bilangan biner adalah sebagai berikut: (kita ambil 0,25, 0 desimal = 0 biner), sampai di sini kita sudah dapat menyatakan sebagai 0,…

0,25 x 2	
0,5 x 2	Digit 0
1,0	Digit 1

0,01

Eksponen terbias kita peroleh dengan menjumlahkan :

7FH	1111111
-2	-0000010
	1111101

Berbagai kasus yang lain, ditinggalkan untuk ujicoba para mahasiswa,

Andai diketahui :

Ujicoba 1

s	Eksponen							
0	1	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23

Mantisa																							
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	

Maka ubahlah bilangan floating-point di atas ke dalam bilangan desimal.

Ujicoba 2

s	Eksponen							
1	0	1	1	1	1	1	1	1
31	30	29	28	27	26	25	24	23

Mantisa																							
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	

Maka ubahlah bilangan floating-point di atas ke dalam bilangan desimal.

Ujicoba 3

s	Eksponen							
0	1	0	0	0	0	0	1	0
31	30	29	28	27	26	25	24	23

Mantisa																							
1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	

Maka ubahlah bilangan floating-point di atas ke dalam bilangan desimal.

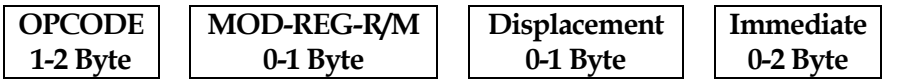
BAHASA MESIN

Bahasa mesin adalah kode biner asli (native) yang dapat dimengerti oleh mikroprosesor dan digunakan sebagai instruksi-instruksi kontrol dalam operasinya. Instruksi bahasa mesin untuk 8086 hingga Pentium sangat beraneka ragam panjangnya, mulai dari satu hingga tiga belas byte. Meski bahasa mesin terlihat sulit, bahasa inilah yang digunakan dalam aplikasi mikroprosesor. Ada lebih dari 100.000 variasi dari instruksi bahasa mesin, dan tidak ada daftar lengkap dari semua variasi yang ada. Karenanya beberapa bit biner dalam instruksi bahasa mesin diberikan di sini, dan yang lainnya ditentukan dari setiap variasi instruksi.

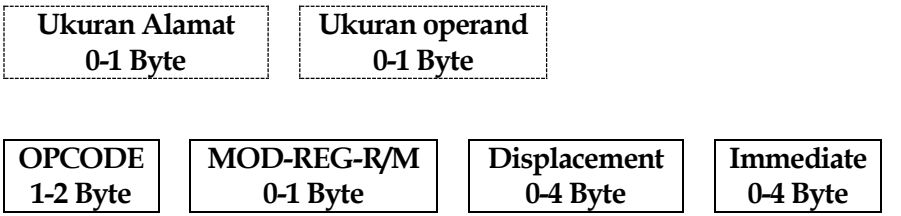
Instruksi dalam 8086 hingga 80286 adalah mode insiruksi 16-bit yang bentuknya dapat dilihat dalam Gambar 2.20(a). Mode instruksi 16-bit akan kompatibel dengan 80386 dan di atasnya bila mikroprosesor itu diprogram untuk beroperasi dalam mode instruksi 16-bit, tetapi juga diprefikskan seperti pada Gambar 2.20(b). Dengan 80386 ke atas, diasumsikan bahwa semua mode instruksi tipe 16-bit-lah yang dieksekusi ketika mikroprosesor beroperasi dalam mode real. Dalam mode protected, byte teratas dari deskriptor (pendeskripsi) berisi D-bit yang akan memilih mode instruksi 16-bit atau 32 bit. Pada saat ini hanya Windows NT, Windows 95, Windows 98 OS/2, dan Versi Windows di atasnya-lah yang beroperasi dalam mode instruksi 32-bit. Mode instruksi 32-bit diperlihatkan dalam Gambar 3.1(b). Instruksi ini sesungguhnya terjadi dalam mode instruksi 16-bit yang mempergunakan prefiks.

Dua byte pertama dari format mode insiruksi 32-bit disebut override prefix, hal ini disebabkan karena 2 byte ini tidak selalu ada. Byte yang pertama memodifikasi ukuran alamat operand yang digunakan oleh instruksi, dan byte kedua memodifikasi ukuran register. Apabila 80386 hingga Pentium 4 beroperasi dalam mode mesin instruksi 16-bit (mode real atau mode protected) dan register 32-bit digunakan, maka ditambahkanlah register-size prefix (66H) di depan baris instruksi. Bila beroperasi dalam mode instruksi 32-bit (mode protected saja), dan register 32-bit yang digunakan, maka. register-size prefix absen. Jika register 16-bit muncul dalam sebuah instruksi dalam mode instruksi 32-bit, maka register-size prefix muncul untuk memilih register 16-bit. Address-size prefix (67H) digunakan dalam bentuk yang sama, dan hal ini akan dijelaskan pada akhir bab ini. Prefiks mengubah ukuran register dan alamat operand dari 16-bit hingga 32-bit atau dari 32-bit ke 16-bit untuk instruksi prefiks.

(a) Mode Instruksi 16-bit



(b) Mode Instruksi 32-bit (80386 – Pentium 4)



Gambar 2.20 Format instruksi pada 8086-Pentium

Opcode : memilih operasi.

						D	**W**

Keterangan :

D=1	R/M ke Register
D=0	Register ke R/M
W=1	Data berukuran word atau doubleword
W=0	Data berukuran byte

MOD-REG-R/M

MOD		**REG**			**R/M**		

Keterangan :

MOD : memilih tipe pengalamatan, dan menentukan kehadiran dan ketidakhadiran displacement. Tabel berikut berlaku untuk operasi 16 bit.

00	Tidak displacement
01	Displacement extended-sign 8-bit
10	Displacement 16-bit
11	R/M adalah register

Untuk 32 bit, berlaku tabel MOD berikut :

00	Tidak displacement
01	Displacement extended-sign 8-bit
10	Displacement 32-bit
11	R/M adalah register

REG dan **R/M**

Kode	**W=0 (byte)**	**W=1 (Word)**	**W=1 (DoubleWord)**
000	AL	AX	EAX
001	CL	CX	ECX
010	DL	DX	EDX
011	BL	BX	EBX
100	AH	SP	ESP
101	CH	BP	EBP
110	DH	SI	ESI

111	BH	DI	EDI

Kasus : tentukan bahasa mesin untuk instruksi MOV BP,SP ?

Solusi :

Opcode : MOV = 100010

1	0	0	0	1	0	**D**	**W**

Pemindahan ke register BP, D=1

W=1, BP dan SP adalah word.

1	0	0	0	1	0	**1**	**1**

R/M = SP, register, maka MOD=11

MOD		**REG**			**R/M**		
1	1						

REG BP=101, dan R/M SP=100, sehingga

MOD		**REG**			**R/M**		
1	1	1	0	1	1	0	0

Maka bahasa mesin MOV BP, SP adalah :

1	0	0	0	1	0	**1**	**1**

MOD		**REG**			**R/M**		
1	1	1	0	1	1	0	0

Mulai dari atas : 1000 = 8, 1011 =B, 1110 =E, dan 1100=C, atau 8BECH.

Apabila field MOD berisi 00, 01, 10, maka field R/M akan mempunyai arti yang baru. Tabel berikut memuat daftar mode pengalamatan memori untuk field R/M ketika MOD berisi 00, 01, 10 untuk instruksi 16 bit.

Kode R/M	**Mode Pengalamatan**
000	DS:[BX+SI]
001	DS:[BX+DI]
010	SS:[BP+SI]
011	SS:[BP+DI]
100	DS:[SI]
101	DS:[DI]
110	SS:[BP]
111	DS:[BX]

Kasus : MOV DL, [DI]

Solusi :

Opcode = MOV

D = Pemindahan ke REG

W = Byte

MOD = tidak ada displacement

REG = DL

R/M = DS:[DI]

1	0	0	0	1	0	**1**	**0**

MOD		REG			R/M		
0	0	0	1	0	1	0	1

Mulai dari atas : 1000 = 8, 1010 =A, 0001 = 1, dan 0101 = 5, atau 8A15H.

Jika instruksi kita rubah sedikit menjadi MOV DL,[DI+1], maka bahasa mesinnya,

1	0	0	0	1	0	**1**	**0**

MOD		REG			R/M		
0	1	0	1	0	1	0	1

Displacement							
0	0	0	0	0	0	0	1

Atau setara dengan 8A5501H

Untuk MOV DL,[DI+1000], maka bahasa mesinnya adalah :

1	0	0	0	1	0	**1**	**0**

MOD		REG			R/M		
1	0	0	1	0	1	0	1

Displacement rendah							
0	0	0	0	0	0	0	0

Displacement tinggi							
0	0	0	1	0	0	0	0

1000 = 8, 1010 = A, 1001 = 9, 0101 = 5, 0010 atau 8A950010H.

Mode Pengalamatan Khusus. Ada mode pengalamatan yang tidak muncul dalam berbagai tabel sebelumnya. Hal ini terjadi ketika data memori direferensikan hanya dengan mode displacement dari pengalamatan untuk instruksi 16-bit. Contohnya adalah instruksi MOV [1000H],DL dan MOV NUMB,DL. Instruksi pertama memindahkan isi register DL ke lokasi memori segmen data 1000H. Instruksi kedua memindahkan register DL ke lokasi memori segmen data simbolis NUMB.

Ketika instruksi hanya mempunyai satu displacement, maka field MOD selalu 00 dan field R/M selalu 110. Seperti diperlihatkan dalam tabel, instruksi tidak berisi displacement dan menggunakan mode pengalamatan [BP]. Anda sebenarnya tidak dapat menggunakan mode pengalamatan [BP] tanpa displacement dalam bahasa mesin. Assembler sangat menjaga hal ini dengan menggunakan displacement 8-bit (MOD = 01) dari 00H ketika [BP] mode pengalamatan digunakan dalam suatu instruksi. Ini artinya mode pengalamatan [BP] terakit sebagai [BP+0], meskipun [BP] digunakan dalam instruksi. Mode pengalamatan yang khusus juga dapat digunakan dalam mode 32-bit.

Mode pengalamatan 32-bit dalam 80386 sampai versi di atasnya diperoleh dengan menjaiankan mesin ini dalam mode instruksi 32-bit atau dalam mode insiruksi 16-bit dengan mempergunakan prefiks ukuran-alamat 67H. Tabel berikut memperlihatkan pengkodean untuk R/M digunakan untuk mode pengalamatan 32 bit secara khusus. Perhatikan bahwa pada saat R/M = 100, maka byte tambahan akan muncul dalam instruksi dan disebut byte indeks-berskala. Byte indeks-berskala menandakan bahwa bentuk pengalamatan indeks-berskala tidak ada dalam Tabel berikut :

Mode pengalamatan 32 bit yang dipilih R/M

Kode R/M	Mode Pengalamatan
000	DS:[EAX]
001	DS:[ECX]
010	DS:[EDX]
011	DS:[EBX]
100	Memakai byte indeks-berskala
101	SS:[EBP]
110	DS:[ESI]
111	DS:[EDI]

Jika kode R/M = 100, maka Byte indeks-skala adalah :

s	**s**	**Indeks**			**Base**		

Keterangan :

s	**s**	
0	0	$\times 1$
0	1	$\times 2$
1	0	$\times 4$
1	1	$\times 8$

Byte indeks-berskala terutama digunakan ketika dua register ditambahkan ke dalam alamat memori dalam suatu instruksi. Karena byte indeks-berskala ditambahkan dalam instruksi, maka ada tujuh bit dalam opcode yang dibuat dan delapan bit dalam byte indeks-berskala. Ini artinya instruksi indeks-berskala mempunyai 2^{15} (32K) kemungkinan kombinasi. Ada lebih dari 32.000 variasi

yang berbeda dari instruksi MOV sendiri dalam mikroprosesor 80386 sampai Pentium 4.

Untuk field indeks dan basis, keduanya berisi nomor register seperti terlihat pada tabel berikut : (32 bit)

Kode	W=0 (byte)	W=1 (Word)	W=1 (DoubleWord)
000	AL	AX	EAX
001	CL	CX	ECX
010	DL	DX	EDX
011	BL	BX	EBX
100	AH	SP	ESP
101	CH	BP	EBP
110	DH	SI	ESI
111	BH	DI	EDI

Kasus : konversikan instruksi MOV [1000H], DL ke bahasa mesin ?

Solusi :

1	0	0	0	1	0	**0**	**0**
MOD			**REG**			**R/M**	
0	0	0	1	0	1	1	0

Displacement rendah							
0	0	0	0	0	0	0	0
Displacement tinggi							
0	0	0	1	0	0	0	0

Keterangan :

D = 0, transfer dari register

W = 0, byte

R/M = SS : [BP]

MOD : karena R/M sama dengan BP (pengalamatan khusus)

Displacement = 1000H

Reg = DL

Kasus : konversikan instruksi MOV [BP], DL ke bahasa mesin ?

Solusi :

1	0	0	0	1	0	**0**	**0**
MOD			**REG**			**R/M**	
0	1	0	1	0	1	1	0

Displacement 8 bit							
0	0	0	0	0	0	0	0

Keterangan :

D = transfer dari register

W = byte

R/M = SS : [BP]

MOD : karena R/M sama dengan BP (pengalamatan khusus)

Displacement = 00H

Reg = DL

Instruksi MOV EAX,[EBX+4*ECXJ dikodekan sebagai 67668B048BH. Perhatikan bahwa ukuran alamat (67H) dan ukuran register (66H) muncul pada instruksi. Kode ini (67668B048BH) digunakan ketika mikroprosesor 80386 hingga versi di atasnya beroperasi dalam mode instruksi 16-bit untuk instruksi ini. Apabila mikroprosesor beroperasi dalam mode instruksi 32-bit, kedua prefiks akan hilang dan instruksi akan menjadi instruksi 8B048BH. Penggunaan prefiks tergantung pada mode operasi dari mikroprosesor. Pengalamatan indeks-berskala dapat juga menggunakan register tunggal dikalikan dengan faktor skala. Sebagai contoh adalah instruksi MOV AL, [2*ECXJ. Isi dari lokasi segmen data dialamatkan oleh dua kali ECX yang disalin ke dalam AL.

Instruksi Segera. Diasumsikan bahwa instruksi MOV WORD PTR [BX+1000H], 1234H dipilih sebagai contoh instruksi 16-bit menggunakan pengalamatan *segera.* Instruksi ini memindahkan 1234H ke lokasi memori berukuran-word yang dialamatkan oleh penjumlahan dari 1000H, BX dan DS x I0H. Instruksi enam-byte menggunakan 2 byte untuk field opcode, W, MOD dan R/M. Dua dari enam byte adalah data 1234H. Dua dari enam byte adalah displacement dari I000H.

Instruksi ⇒ MOV WORD PTR [BX+1000H],1234H

1	1	0	0	0	1	**1**	**1**
MOD			**REG**			**R/M**	
1	0	0	0	0	1	1	1

Displacement rendah							
0	0	0	0	0	0	0	0

Displacement tinggi

0	0	0	1	0	0	0	0

Data rendah							
0	0	1	1	0	1	0	0

Data tinggi							
0	0	0	1	0	0	1	0

Instruksi MOV segmen.

Kode REG segmen yang digunakan adalah :

Kode	**Register Segmen**
000	ES
001	CS
010	SS
011	DS
100	FS
101	GS

Kasus : MOV BX, CS

Solusi :

1	0	0	0	1	1	**0**	**0**
MOD			**REG**			**R/M**	
1	1	0	0	1	0	1	1

Bab 3
PIPELINING

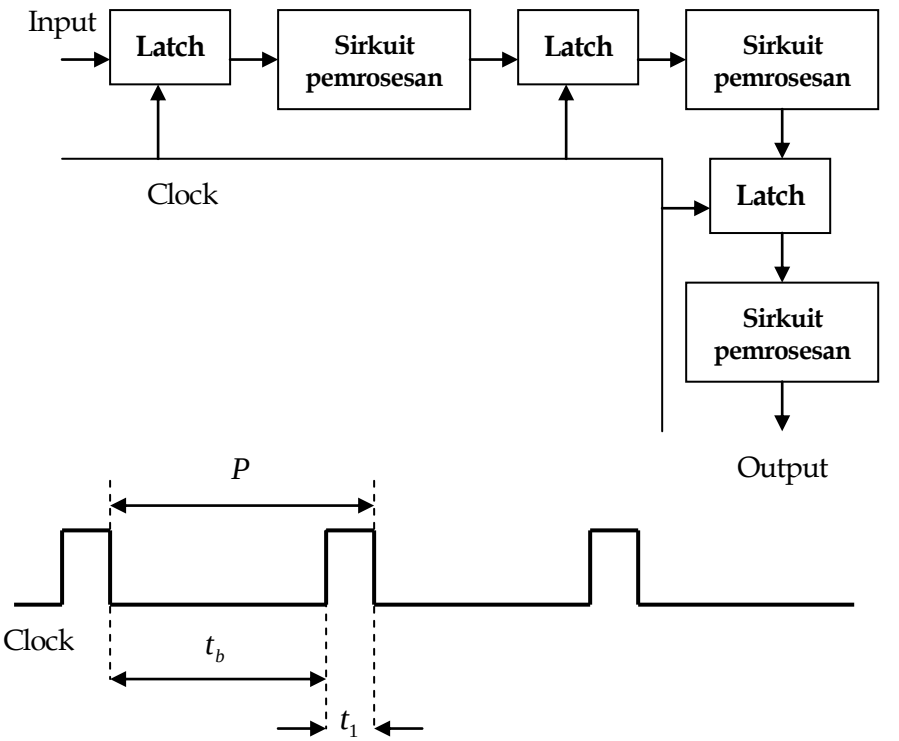
PENDAHULUAN

Pipelining adalah suatu cara meningkatkan keseluruhan *performance* proses dari sebuah prosesor. Pendekatan arsitektur tersebut memungkinkan eksekusi serempak beberapa instruksi. Pada bab ini, pembahasan akan meliputi : struktur pipeline, pengukuran performansi pipeline, jenis-jenis pipeline, instruksi pipeline, aritmatika pipeline, dan lain-lain.

STRUKTUR PIPELINE

Teknik perancangan pipeline mendekomposisikan sebarisan proses ke dalam sub-sub proses atau segmen/stadium. Setiap stadium melakukan suatu fungsi khusus dan menghasilkan suatu output yang dikehendaki dengan segera. Setiap stadium akan memuat satu input latch, atau disebut juga register atau buffer, yang diikuti oleh satu sirkuit pemrosesan. Sirkuit pemrosesan dari stadium yang diberikan dihubungkan dengan input latch dari stadium berikutnya. Sinyal clock dikoneksikan ke masing-masing input latch. Pada masing-masing pulsa clock, seluruh stadium akan

mentransfer hasil dengan segera ke input latch dari stadium berikutnya.



P = Periode clock

t_b = Waktu maksimum untuk satu stadium melakukan suatu fungsi

t_1 = Waktu bagi latch menerima data input

Gambar 3.1 Struktur dasar sebuah pipeline

Hasil akhir diperoleh setelah data input melalui keseluruhan pipeline, melengkapi satu stadium per pulsa clock. Periode pulsa clock harus cukup besar dalam memberikan waktu yang cukup ke

satu sinyal melintasi stadium-stadium yang sangat lambat, atau disebut juga *bottleneck*. Penambahan, latch harus memiliki cukup waktu untuk menyimpan sinyal inputnya. Jika periode clock, P, diekspresikan sebagai $P = t_b + t_1$, maka t_b harus lebih besar dari waktu tungu maksimum dari stadium bottleneck, dan t_1 harus cukup untuk menyimpan data ke latch.

PENGUKURAN PERFORMANSI PIPELINE

Kemampuan meng-overlap stadium sebarisan proses untuk berbagai hasil input-task yang berbeda dalam waktu penyelesaian teoretis secara keseluruhan diekspresikan sebagai :

$$T_{pipe} = m * P + (n-1) * P$$

di mana n merupakan jumah input-task, m merupakan jumlah stadium dalam pipeline, dan P adalah periode clock. Term $m * P$ merupakan waktu yang dibutuhkan input-task pertama melalui pipeline, dan term $(n-1) * P$ adalah waktu sisa yang dibutuhkan untuk berbagai task. Setelah pipeline terpenuhi, maka output diberikan pada masing-masing siklus clock, dengan kata lain, pipeline akan menghasilkan output secepat stadium yang paling lambat saja. Bahkan dengan keterbatasan tersebut, pipeline akan melakukan apa-apa yang mampu dilakukan teknik-teknik non-pipeline, di mana eksekusi task berikut dapat dilakukan jika hanya task yang sedang diproses telah lengkap atau sempurna tereksekusi. Untuk lebih spesifik, jika n lebih besar, maka output yang dihasilkan adalah m kali lebih cepat daripada output yang dihasilkan oleh prosesor non-pipeline. Waktu lengkap melakukan proses yang diperlukan oleh prosesor non-pipeline adalah :

$$T_{seq} = n * \sum_{i=1}^{m} \tau_i$$

di mana τ_i adalah waktu tunggu (delay time) untuk masing-masing stadium. Untuk kasus ideal, di mana semua stadium

memiliki waktu tunggu yang setara, $\tau_i = \tau$ untuk $i = 1$ hingga m, T_{seq} dapat dinyatakan sebagai :

$$T_{seq} = n * m * \tau$$

Jika kita mengabaikan waktu yang dibutuhkan latch untuk melakukan penyimpanan (storing) t_1, maka

$$T_{seq} = n * m * P$$

Speedup atau dinotasikan sebagai S direpresentasikan sebagai :

$$S = T_{seq} \Big/ T_{pipe} = n * m / (m + n - 1)$$

nilai S akan mendekati m ketika $n \to \infty$ [1]. Hubungan formulasi di atas merupakan speedup maksimum, atau speedup ideal dari sebuah prosesor pipeline dengan m stadium terhadap sebuah prosesor non-pipeline yang ekuivalen adalah m. Dengan kata lain, speedup ideal setara dengan jumlah stadium pipeline. Jika nilai n sangat besar, maka sebuah prosesor pipeline dapat menghasilkan output mendekati m kali lebih cepat dari sebuah prosesor non-pipeline. Sementara jika n bernilai lebih kecil, maka speedup akan menurun; untuk $n = 1$ pipeline akan memiliki speedup minimum yaitu bernilai 1.

Dua faktor yang tidak kalah penting yang sering digunakan untuk menentukan performansi sebuah pipeline adalah *efficiency* dan *throughput*. Efficiency E sebuah pipeline dengan m stadium didefinisikan sebagai :

$$E = S/m = \left[n * m/m + n - 1\right]/m = n/(m + n - 1)$$

Efficiency E merepresentasikan speedup per stadium, ketika $n \to \infty$ maka E mendekati nilai maksimum 1, dan jika $n = 1$, E akan bernilai $1/m$, atau nilai minimum yang mungkin diperoleh.

---

[1] $n \to \infty$ baca n menuju tidak hingga.

Throughput H, atau disebut juga bandwidth sebuah pipeline dapat didefinisikan sebagai jumlah task-task input yang dapat diproses per unit waktu. Ketika pipeline memiliki m stadium, H didefinisikan sebagai :

$$H = n/T_{pipe} = n/\left[m * P + (n-1) * P\right] = E/P = S/(mP)$$

Jika $n \to \infty$; throughput H mendekati nilai maksimum untuk satu task per siklus clock.

Jumlah stadium di dalam sebuah pipeline sering tergantung pada tradeoff antara performansi dan biaya. Pilihan optimal untuk suatu nilai tersebut dapat ditentukan dengan mencari nilai puncak dari suatu performansi per rasio biaya (PCR). Larson mendefinisikan PCR sebagai :

$$PCR = \frac{\text{maximum throughput}}{\text{pipeline cost}}$$

Untuk mengilustrasikan, asumsikan sebuah prosesor nonpipeline membutuhkan suatu waktu penyelesaian t_{seq} untuk pengolahan satu tugas input. Untuk sebuah pipeline dengan m stadium, memproses tugas yang sama akan memerlukan satu periode clock $P = \left(t_{seq}/m\right) + t_l$ (t_l adalah waktu tunda latch). Oleh karena itu, throughput maksimum yang dapat diperoleh dengan pipeline tersebut adalah,

$$1/P = 1/\left(t_{seq}/m\right) + t_l$$

Throughput maksimum $1/P$ sering juga diistilahkan sebagai frekuensi pipeline. Nilai throughput aktual biasanya lebih kecil dari $1/P$ atau tergantung pada rate berbagai task yang berhubungan yang sedang memasuki pipeline.

Nilai cost pipeline c_p dapat diekspresikan sebagai total cost gerbang logika dan berbagai latch yang digunakan pada masing-masing stadium. Atau dinotasikan dalam bentuk hubungan, $c_p = c_g + mc_l$, di mana c_g merupakan cost keseluruhan stadium

logika dan c_l merupakan cost untuk masing-masing latch. Dengan mensubstitusikan berbagai nilai untuk maksimum throughput dan cost pipeline dalam persamaan PCR, maka kita akan memperoleh formulasi berikut :

$$PCR = 1/\{[(t_{seq}/m)+t_l][c_g + mc_l]\}$$

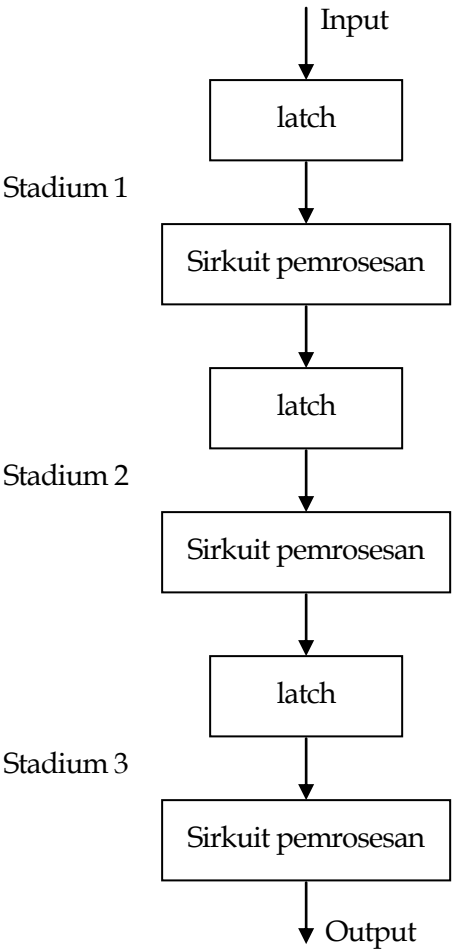
Persamaan ini akan memiliki suatu nilai maksimum m_0, di mana

$$m_0 = \sqrt{\frac{(t_{seq} * c_g)}{(t_l * c_l)}}$$

Karena nilai m_0 merupakan nilai maksimal PCR, maka nilai tersebut dapat digunakan sebagai pilihan optimal untuk jumlah berbagai stadium.

BERBAGAI JENIS PIPELINE

Pipeline biasanya dibagi menjadi dua kelas : pipeline instruksi dan pipeline aritmetik. Dalam setiap kelas pipeline dapat didesain dengan dua cara : statis atau dinamis. Sebuah pipeline statis hanya dapat melakukan satu operasi (seperti operasi penambahan atau operasi perkalian) pada satu waktu. Operasi sebuah pipeline statis hanya dapat dirubah setelah pipeline berkaitan telah dikosongkan (drained), maksudnya data input terakhir telah meninggalkan pipeline. Sementara sebuah pipeline dinamis dapat melakukan lebih dari satu operasi pada satu waktu. Untuk melakukan operasi tertentu terhadap data input, data harus terlebih dahulu melalui sebarisan stadium tertentu. Gambar 3.2 mengilustrasikan sebuah pipeline dinamis yang terdiri dari tiga stadium melakukan operasi penambahan dan perkalian terhadap berbagai data yang berbeda pada waktu yang sama.



Gambar 3.2 Sebuah pipeline dinamis yang terdiri dari tiga stadium

Untuk melakukan proses perkalian, input data harus melalui stadium 1, 2, dan 3; Untuk melakukan penambahan, data hanya cukup melalui stadium 1 dan 3 saja. Oleh karena itu, stadium pertama dari proses penambahan dapat dilakukan pada sebuah data input D_1 di stadium 1, sementara pada waktu yang bersamaan, stadium akhir dari proses perkalian dilakukan di

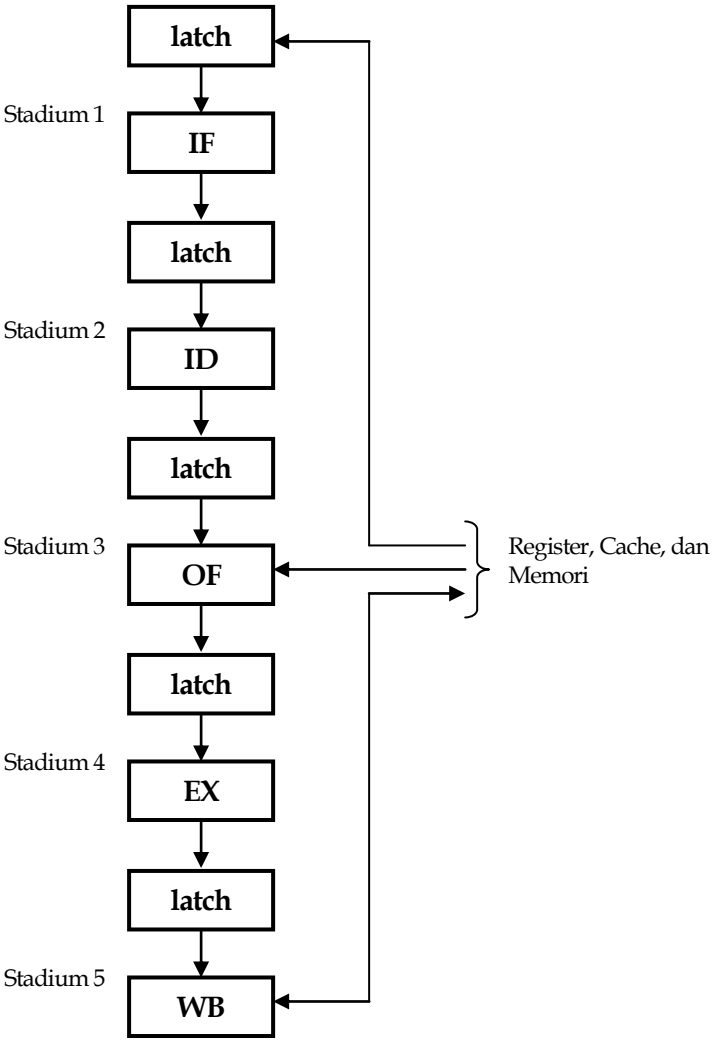
stadium 3 pada sebuah input data yang berbeda D_2. Perhatikan bahwa, interval waktu antara initiasi input D_1 dan D_2 ke pipeline harus diupayakan agar mereka tidak meraih stadium 3 secara bertepatan atau pada waktu yang sama, untuk menghindari terjadinya *collision*.

PIPELINE INSTRUKSI

Pada arsitektur Von Neumann, proses eksekusi sebuah instruksi melalui beberapa langkah. Pertama, control unit dalam prosesor mengambil instruksi dari cache atau dari memori, kemudian control unit mendekodekan intruksi untuk menentukan jenis operasi yang akan dilakukan. Ketika operasi membutuhkan operand, control unit akan menentukan alamat masing-masing operand dan mengambilnya dari cache atau memori. Selanjutnya operasi dilakukan pada operand dan, terakhir, hasil operasi disimpan pada lokasi tertentu.

Sebuah pipeline instruksi meningkatkan performansi prosesor dengan mengoverlapping pemrosesan berbagai instruksi yang berbeda. Hal ini sering dilakukan dengan membagi proses eksekusi instruksi ke dalam berbagai stadium. Sebagaimana ditunjukkan pada gambar 3.3, sebuah pipeline intstruksi sering terdiri dari lima stadium, sebagai berikut :

1. Instruction Fetch (IF). Mengemas instruksi dari cache atau memori.
2. Instruction Decode (ID). Mengidentifikasi operasi yang akan dilakukan.
3. Operand Fetch (OF). Mendekodekan dan menemas sebarang operand yang diperlukan.
4. Execution (EX). Melakukan operasi pada operand.
5. Write-Back (WB). Memperbarui berbagai operand tujuan.



Gambar 3.3 Berbagai stadium sebuah pipeline instruksi

Sebuah pipeline instruksi mengoverlap proses stadium sebelumnya untuk berbagai instruksi yang berbeda dengan maksud untuk mencapai waktu komplesi total yang minimum. Sebagai contoh, perhatikan gambar 3.4, gambar menunjukkan eksekusi empat

instruksi dalam sebuah pipeline intruksi. Sepanjang siklus pertama, atau denyut clock, instruksi i_1 diambil dari memori. Pada siklus kedua, instruksi i_1 didekodekan sementara itu instruksi i_2 diambil dari memori. Proses ini berlangsung kontinu sampai keseluruhan instruksi telah dieksekusi. Intruksi terakhir akan menyelesaikan stadium Write-Back setelah siklus clock ke delepan. Oleh karena itu, proses keseluruhan akan memakan waktu 80 nanodetik (ns) untuk melengkapi eksekui ke empat instruksi, di mana sebelumnya kita asumsikan periode clock adalah 10 ns. Waktu komplesi total dapat juga diperoleh menggunakan persamaan :

$$T_{pipe} = m * P + (n-1) * P$$

$$T_{pipe} = 5 * 10 + (4-1) * 10 = 80 \text{ ns}$$

Bandingkan dengan waktu yang diperlukan oleh rancangan non-pipeline,

$$T_{seq} = n * m * P = 4 * 5 * 10 = 200 \text{ ns}$$

	Siklus							
	1	2	3	4	5	6	7	8
Instruksi								
i_1	IF	ID	OF	EX	WB			
i_2		IF	ID	OF	EX	WB		
i_3			IF	ID	OF	EX	WB	
i_4				IF	ID	OF	EX	WB

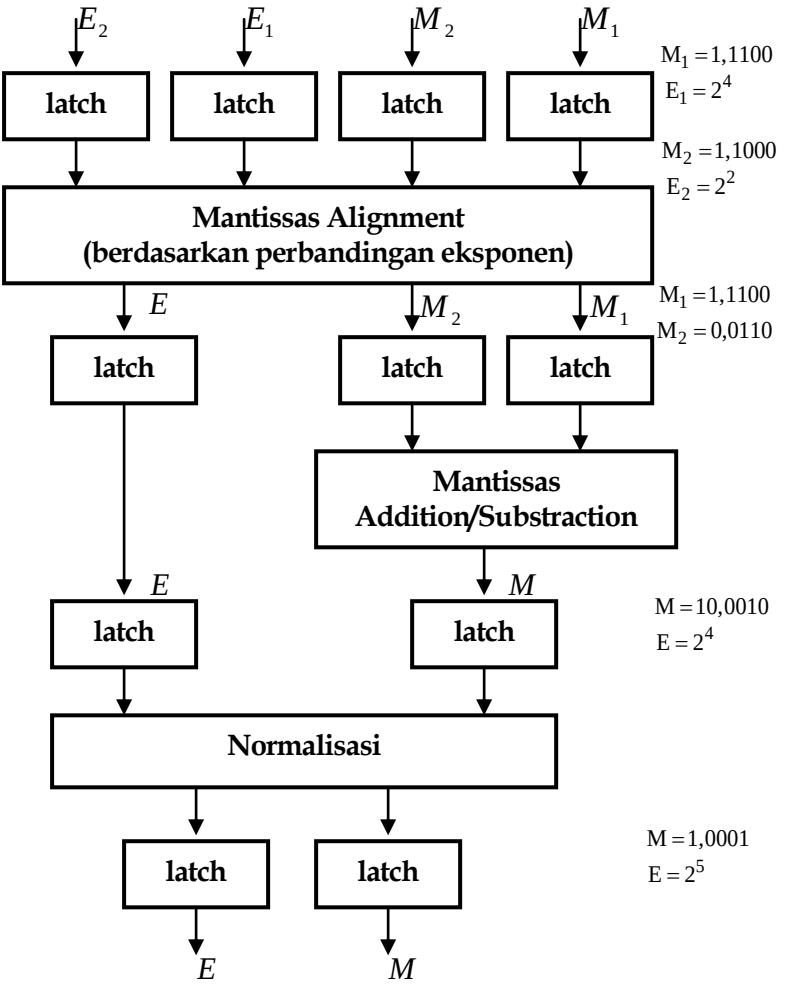
Gambar 3.4 Siklus eksekusi empat instruksi yang saling berhubungan di dalam sebuah pipeline instruksi

PIPELINE ARITMETIK

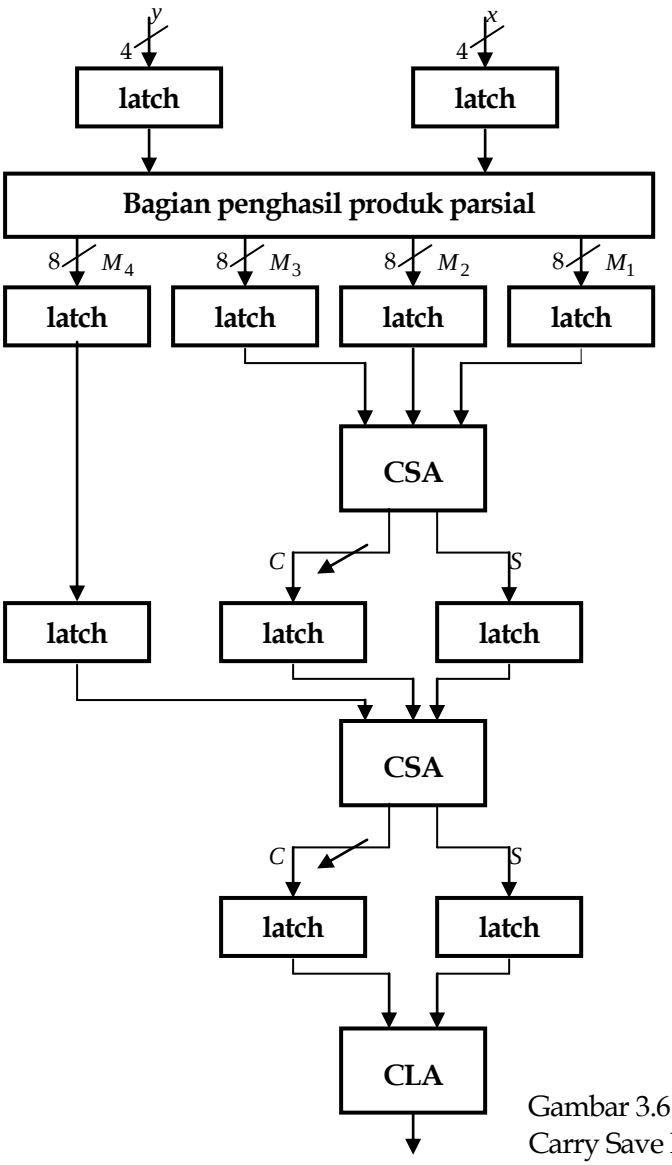
Teknologi pipeline dapat diterapkan ke berbagai fungsi unit logikal aritmetik, dengan maksud untuk memaksimalkan performansi. Pipeline aritmetik digunakan untuk pengimplementasian fungsi aritmetik kompleks seperti penambahan, perkalian, dan pembagian bilangan floating-point. Fungsi-fungsi tersebut dapat diatur ke berbagai sub-fungsi yang berkaitan. Gambar 3.5 merepresentasikan sebuah arsitektur pipeline untuk penambahan dua bilangan floating-point. Penambahan floating-point dapat dibagi ke dalam tiga stadium : mantissas alignment (penjajaran mantisa), penambahan mantisa (mantissas addition), dan normalisasi.

Pada stadium pertama, mantisa M_1 dan M_2 dijajarkan berdasarkan selisih eksponen E_1 dan E_2. Jika $|E_1 - E_2| = k > 0$, maka mantisa dengan eksponen terkecil digeser ke kanan sebanyak k digit. Pada stadium kedua mantisa selanjutnya akan ditambah (atau dikurangi). Pada stadium ketiga, hasil akan dinormalisasi agar mantisa akhir memiliki digit yang tidak nol sesudah nilai fraksi.

Contoh pipeline aritmetik lainnya diperlihatkan pada gambar 3.6. Gambar merepresentasikan arsitektur pipeline untuk perkalian dua bilangan 4-bit bertipe unsigned menggunakan CSA (Carry Save Adders). Stadium pertama menghasilkan produk parsial $M_1, M_2, M_3,$ dan M_4. Gambar 3.6 merepresentasikan bagaimana M_1 dihasilkan, sisa produk parsial juga dihasilkan dengan cara yang sama. $M_1, M_2, M_3,$ dan M_4 ditambahkan secara bersamaan melalui dua stadium CSA (Carry Save Adders) dan stadium CLA (Carry Lookahead Adder).



Gambar 3.5 Sebuah pipeline fungsi penambahan floating-point



Gambar 3.6 Sebuah pipeline Carry Save Multiplier

Bab 4
ARSITEKTUR RISC vs CISC

PENDAHULUAN

Arsitektur komputer, secara umum, telah mengalami perkembangan pesat dan lebih kompleks, kekompleksitasan dalam berbagai fitur seperti set instruksi yang lebih besar, lebih banyak mode pengalamatan, unjuk kerja instruksi yang lebih komputasional, terdapat lebih banyak register-register dengan fungsi khusus, dan sebagainya. Berbagai mesin yang memiliki berbagai peningkatan kekompleksitasan di atas sering diistilahkan sebagai ***complex instruction set computers*** (CISCs). Penambahan sebuah fungsi kompleks ke dalam set instruksi sangat mempengaruhi efisiensi dan harga prosesor. Pengaruh-pengaruh tersebut seharusnya telah dievaluasi sebelum instruksi tersebut ditambahkan ke dalam set instruksi. Beberapa instruksi yang disediakan oleh prosesor CISC bersifat esoteris dan secara umum berbagai kompiler tidak menyediakan fasilitas untuk memanfaatkannya, sehingga instruksi-instruksi tersebut hanya

dapat difungsikan dengan memanfaatkan fasilitas penulisan program menggunakan tool assembler. Walaupun terdapat sebagian kecil kompiler yang menyediakan fasilitas pemanfaatan berbagai instruksi tersebut, penggunaan fasilitas pemanfaatan instruksi tersebut tidak terjamin akan digunakan secara kontinu atau sering. Sehingga intruksi tersebut secara idealnya tidak perlu disediakan. Berdasarkan konsep peniadaan berbagai instruksi tersebut ke dalam set instruksi, maka munculah gagasan inovatif untuk mendesain suatu arsitektur komputer alternatif yang sering diistilahkan ***reduced instruction set computer*** (RISC). Filosofi dibalik perancangan arsitektur RISC adalah keuntungan performansi dengan hanya instruksi-instruksi tertentu saja yang ditambahkan ke set instruksi. Mesin arsitektur RISC pertama kali dibangun pada tahun 1982 oleh perusahaan IBM, minicomputer 801. Karakteristik umum pada perancangannya adalah kepemilikan set instruksi yang sederhana dan terbatas, memori cache yang bersifat on-chip, memiliki kompiler yang bertugas memaksimalkan penggunaan register sehingga meminimalkan akses memori, dan penekanan kepada optimisasi pipeline instruksi.

BERBAGAI SEBAB MENINGKATNYA KEKOMPLEKSITASAN ARSITEKTUR

Ada beberapa alasan meningkatnya kekompleksitasan suatu arsitektur komputer :

1. Mendukung high-level languages, perubahan lingkungan pemrograman dari penggunaan bahasa assembly ke lingkungan pemrograman high-level languages, sehingga pabrik-pabrik arsitektur telah memulai mengintegrasikan berbagai instruksi yang bersifat powerful untuk mendukung pengimplementasian program high-level languages secara efisien.
2. Migrasi berbagai fungsi dari software ke hardware, sebuah instruksi tunggal yang diterapkan pada hardware akan

melakukan unjuk kerja yang lebih baik dibandingkan unjuk kerja yang dilakukan sebarisan dari beberapa instruksi yang sederhana, terkait kepada jumlah akses memori yang lebih besar dan disparitas yang terbentuk antara kecepatan CPU dengan memori. Untuk meningkatkan kecepatan pengolahan komputer, fenomena perpindahan fungsi dari software ke firmware dan dari firmware ke hardware harus diperhatikan. Firmware merupakan sebarisan mikroinstruksi. Migrasi fungsi dari domain software ke domain hardware secara natural akan meningkatkan ukuran set instruksi, sehingga secara keseluruhan mempengaruhi terhadap kekompleksitasan komputer.

3. Kompatibel ke depan, kekompatibelan ke depan sering digunakan oleh pabrik sebagai strategi pemasaran produksi, di dalam tatanan untuk mempromosikan berbagai komputer produksi pabrik tersebut sebagai model yang terbaik dari berbagai model yang telah hadir sebelumnya. Konsekuensi dari penerapan strategi tersebut, berbagai pabrik harus meningkatkan jumlah instruksi dan power arsitektur yang dikeluarkannya. Kekompatibelan ke depan merupakan suatu cara meningkatkan suatu desain dengan menambahkan fitur-fitur yang lebih baru dan biasanya lebih kompleks.

KENAPA RISC

Terdapat dua kriteria utama yang menjadi tujuan dalam perancangan suatu sistem secara universal :

1. Memaksimalkan kecepatan operasi atau meminimalkan waktu eksekusi.
2. Meminimalkan biaya pengembangan dan harga jual.

Satu cara untuk mencapai tujuan pertama adalah dengan meningkatkan teknologi berbagai komponen pendukung, sehingga mampu beroperasi pada berbagai frekuensi yang lebih tinggi.

Peningkatan kecepatan juga dapat dicapai dengan meminimalkan jumlah rata-rata siklus clock per instruksi dan/atau pengeksekusian beberapa instruksi secara serempak. Untuk memenuhi kedua kriteria di atas, para desainer RISC harus memfokuskan kepada berbagai aspek penerapan VLSI. Sebagai hasilnya arsitektur akan memiliki sedikit instruksi, berbagai mode pengalamatan, dan berbagai format instruksi, sebuah sirkuit sederhana dan kecil untuk unit kontrol (CU) akan diperoleh. Pengurangan relatif tersebut, baik ukuran maupun kekompleksitasan dengan memanfaatkan chip VLSI menghasilkan beberapa tujuan yang sesuai pada CISC.

Manfaat utama teknologi VLSI adalah perealisasian berbagai prosesor pada sebuah chip tunggal, sehingga terjadinya pengurangan waktu tunggu yang diperlukan untuk pengiriman suatu sinyal dari satu chip ke chip lainnya, secara signifikan. Arsitektur yang kompleks (memiliki set instruksi, terdapat banyak mode pengalamatan, format instruksi yang bervariasi, dan sebagainya) akan membutuhkan fetch instruksi, dekode, dan logika eksekusi yang kompleks. Jika prosesor dimikroprogramkan, logika tersebut diletakkan ke dalam mikroprogram yang kompleks, dan hasilnya dalam storage mikrokode. Jika CISC dibangun dengan memanfaatkan teknologi VLSI, bagian substansial area chip dapat saja tergunakan untuk pemanfaatan storage mikrokode. Arsitektur CISC akan memanfaatkan area chip berkisar 40% hingga 60%, sementara arsitektur RISC menggunakan area ±10%. Area tersisa dalam RISC dapat dimanfaatkan untuk peletakkan berbagai komponen seperti : cache on-chip, dan berbagai arsip register yang besar yang berfungsi untuk meningkatkan performansi prosesor. Seiring dengan meningkatnya kemajuan teknologi VLSI, penerapan RISC juga selalu lebih maju ke depan dibandingkan arsitektur CISC. Jika CISC diterapkan pada sebuah chip tunggal, maka RISC dapat memiliki lebih banyak register, dan cache on-chip, dan ketika CISC memiliki banyak register dan cache on-chip, RISC memiliki lebih dari satu unit pemrosesan, dan seterusnya.

Beberapa faktor yang terkait ketika kita mendiskusikan berbagai keuntungan RISC : kecepatan komputasi, realisasi VLSI,

kehandalan biaya *design-time*, dan mendukung bahasa tingkat tinggi. Untuk kecepatan komputasi, desain RISC disesuaikan lebih elegan terhadap pendekatan pipeline instruksi. Pipeline instruksi memungkinkan beberapa instruksi diproses pada satu waktu. Proses sebuah instruksi dipisahkan ke sederetan fase, seperti yang didiskusikan pada bab pipeline. Ketika sebuah instruksi berada di dalam fase fetch, instruksi lainnya berada di dalam fase dekoding, dan seterusnya. Arsitektur RISC memaksimalkan ukuran throughput pipeline dengan memiliki ukuran instruksi yang seragam dan durasi eksekusi yang seragam untuk sebagian besar instruksi. Ukuran instruksi dan durasi eksekusi yang seragam mampu mengurangi periode *idle* di dalam pipeline.

Realisasi VLSI berhubungan kepada kenyataan atau fakta bahwa unit kontrol RISC diimplementasikan di dalam hardware atau perangkat keras. Sistem *hardwired-controlled* secara umum akan lebih cepat dibandingkan bentuk mikroprogram lainnya. Terlebih lagi bila arsitektur melibatkan file register yang besar dan cache on-chip dalam desainnya, yang dapat mengurangi memori akses secara signifikan. Kebanyakan berbagai item data yang digunakan secara rutin dapat disimpan di dalam berbagai register. Register dapat juga berungsi untuk penyimpanan berbagai parameter yang akan dikirimkan ke prosedur-prosedur tertentu. Akibat dari kemajuan teknologi VLSI, banyak mikroprosesor- mikroprosesor komersiil memiliki cache on-chip. Cache on-chip berukuran lebih kecil bila dibandingkan dengan cache onboard, dan berfungsi sebagai cache level satu. Cache onboard yang berdekatan dengan chip prosesor berperan sebagai cache level dua. Pemanfaatan ke dua cache tersebut pada arsitektur sangat berperan dalam meningkatkan performance komputer bila dibandingkan dengan arsitektur yang menggunakan hanya satu cache. Lebih jauh, cache pada masing-masing level secara aktual sering diorganisasikan sebagai hierarki cache, sebagai contoh, prosesor Intel P6 memuat dua level cache on-chip. Terkadang, masing-masing cache pada level yang paling tinggi dipilah menjadi dua cache : cache instruksi dan cache data, prosesor yang memiliki kategori tersebut sering diistilahkan dengan arsitektur berbasis *Harvard*. Pemanfaatan ke

dua cache, satu untuk instruksi dan yang lain untuk data, ke sebuah cache tunggal secara signifikan dapat meningkatkan waktu akses dan secara konsekuen meningkatkan performance prosesor, terlebih lagi bila diterapkan ke berbagai arsitektur RISC.

Periode desain yang diperlukan untuk merancang arsitektur RISC adalah lebih singkat. Waktu yang dihabiskan untuk merancang sebuah arsitektur baru adalah sangat tergantung kepada kekompleksitasan arsitektur. Secara praktis, waktu yang diperlukan untuk merancang arsitektur CISC adalah lebih lama, karena CISC membutuhkan proses *debugging* desain dan pembuangan berbagai error dari control unit yang dimikroprogram secara kompleks. Sebaliknya pada arsitektur RISC, waktu yang diperlukan untuk menguji dan men-*debug* hardware yang dihasilkan adalah lebih sedikit dikarenakan tidak diperlukannya mikroprogram dan ukuran control unit adalah kecil.

Secara umum, sebuah arsitektur RISC memiliki berbagai karakteristik berikut :

1. Sebagian besar instruksi mengakses berbagai operand dari register kecuali intruksi-instruksi tertentu, seperti LOAD/STORE, yang mengakses memori, dengan kata lain arsitektur RISC merupakan mesin load-store.
2. Eksekusi sebagian besar instruksi hanya membutuhkan satu siklus prosesor tunggal, kecuali intruksi-instruksi tertentu, seperti LOAD/STORE. Tetapi dengan kehadiran cache on-chip, instruksi LOAD/STORE dapat dilakukan pada satu siklus atau periode.
3. Instruksi memiliki format tetap dan tidak melewati batas word memori utama.
4. Control unit telah terintegrasi menyatu, sehingga arsitektur RISC tidak dimikroprogramkan.
5. Memiliki sejumlah kecil format instruksi.
6. CPU memiliki arsip register yang besar, alternatif yang lain adalah memori cache on-chip.

7. Kekompleksitasan pada kompiler
8. Relatif memiliki instruksi yang lebih sedikit (sering < 150) dan memiliki mode pengalamatan yang sangat sedikit.
9. Mendukung operasi HLL (High-Level Language) dengan pemilihan instruksi yang tepat dan dengan memanfaatkan kompiler yang optimal.
10. Memanfaatkan penggunaan pipeline instruksi dan berbagai pendekatan untuk bertransaksi dengan percabangan (branch), seperti multipel prefetch dan berbagai teknik prediksi percabangan.

STUDI KASUS

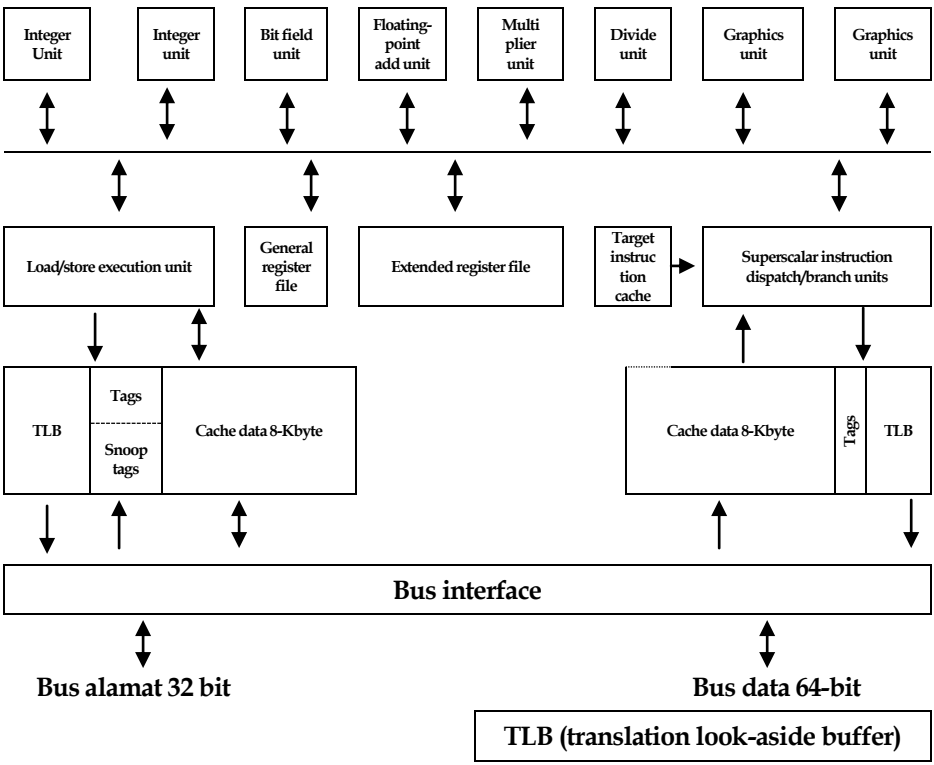
Ada sejumlah prosesor RISC dan CISC dipasaran. Prosesor yang termasuk kategori RISC meliputi : MIPS R4000, IBM RISC System/6000, Alpha AXP, PowerPC, dan Motorola 88000. Jenis-jenis tersebut memberikan berbagai variasi perancangan dengan derajat dan interpretasi beragam dalam pemanfaatan RISC. MIPS R4000 berlandaskan pada teknik super-pipeline, dan IBM RISC System/6000, Alpha AXP, PowerPC, dan Motorola 88000 berlandaskan pada teknik superscalar. Untuk jenis CISC kita akan mengambil sampel prosesor Intel Pentium yang juga memanfaatkan teknik superscalar.

MIKROPROSESOR MOTOROLA 88110

Motorola 88110 merupakan generasi ke dua kerabat arsitektur 88000. Generasi pertama, 88100/200, memuat tiga chip, satu chip CPU (88100) dan dua chip cache (88200). Ide dibalik perancangan 88110 adalah untuk menghasilkan sebuah mikroprosesor *general-purpose* untuk PC dan workstation yang murah biaya. Unjuk kerja (performance) yang ditawarkan adalah pengimplementasian antarmuka perangkat lunak *user-oriented* interaktif, pengolahan

suara dan citra, dan teknik grafik lanjutan untuk PC. 88110 merupakan mikroprosesor superscalar chip tunggal yang diimplementasikan di dalam teknologi CMOS, dan mampu mengeksekusi dua instruksi per siklus clock.

Gambar 4.1 merepresentasikan komponen-komponen utama 88110, terdiri dari tiga cache, dua file register, dan 10 unit eksekusi.

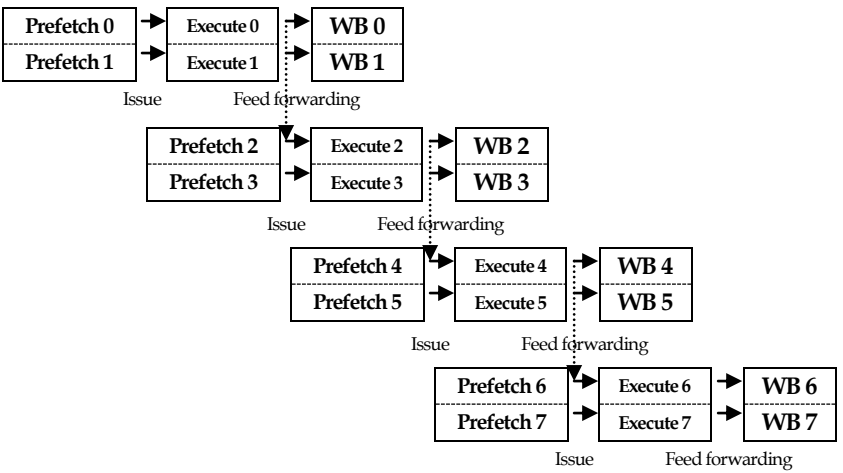


Gambar 4.1 Berbagai komponen utama 88110

Satu cache digunakan untuk menyimpan intruksi target percabangan, dan cache yang lain digunakan untuk menyimpan data dan instruksi. 88110 menyimpan instruksi di dalam salah satu cache, dan data di dalam cache yang lain. Dengan kata lain, 88110

mampu mengambil instruksi dan data secara serempak. 10 unit eksekusi yang terdapat pada 88110 adalah unit intruksi, load/store, floating-point adder, multiplier, divider, dua integer, dua grafik, dan unit bit-field.

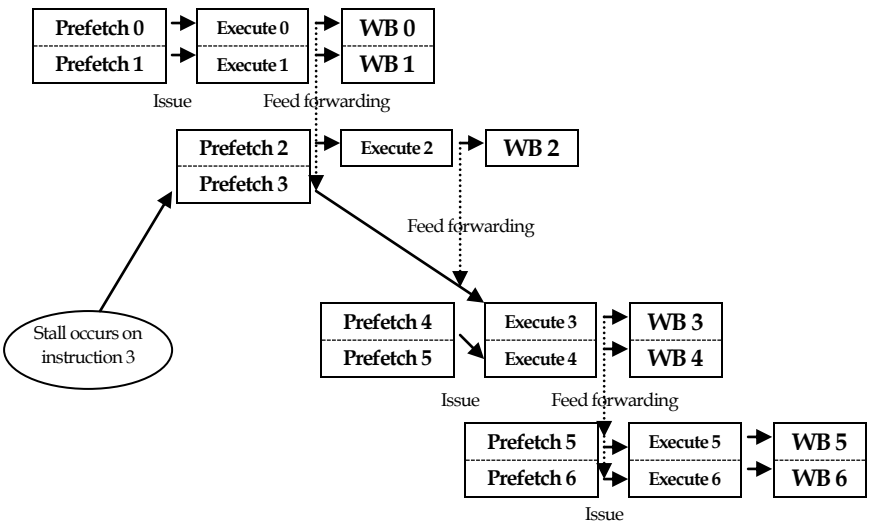
Unit instruksi. 88110 dapat mengeksekusi lebih dari satu instruksi per siklus clock. Hal ini dicapai dengan melibatkan berbagai fitur seperti pipelining, *feed forwarding*, prediksi percabangan, dan unit eksekusi multipel, di mana fitur-fitur tersebut beroperasi dengan tidak tergantung ke fitur lainnya dan secara paralel.



Gambar 4.2 Pewaktuan instruksi prefetch dan eksekusi

Seperti yang ditunujukkan pada gambar 4.2 pipeline intruksi memiliki tiga stadium. Dalam stadium pertama, stadium prefetch dan dekode, sepasang instruksi diambil dari cache instruksi dan didekodekan, operand diambil dari register, dan selanjutnya diputuskan apakah intruksi tersebut dieksekusi atau tidak. Jika resource yang diperlukan seperti operand, register tujuan, atau unit eksekusi tidak tersedia, maka instruksi tidak akan dikirim ke unit eksekusi untuk diproses. Stadium ke dua, stadium eksekusi, dalam

stadium ini instruksi telah dieksekusi. Stadium ini akan memerlukan satu siklus clock untuk mengeksekusi instruksi, tetapi ada beberapa instruksi tertentu memerlukan lebih dari satu siklus clock untuk eksekusi. Stadium ke tiga, stadium write-back, dalam stadium ini hasil eksekusi intruksi akan dituliskan ke dalam register.



Gambar 4.3 Urutan eksekusi instruksi

Unit instruksi selalu mengisukan instruksi ke unit eksekusi, menghasilkan sebuah alamat tunggal untuk masing-masing operasi prefetch, dan menerima dua instruksi dari cache instruksi. Unit instruksi selalu mencoba mengisukan ke dua instruksi pada satu waktu ke unit instruksi. Jika instruksi pertama tidak dapat diisukan dikarenakan berbagai hal seperti : ketidaktersedian resource, atau dependency data, maka tidak satupun dari ke dua instruksi akan diisukan dan ke dua instruksi dipetikan (stalled). Jika instruksi pertama diisukan dan yang ke dua tidak, maka instruksi ke dua akan dipetikan dan dipasangkan dengan instruksi yang baru untuk dieksekusikan. Gambar 4.3 merepresentasikan sebuah contoh untuk kasus di atas. Pada gambar, instruksi ke tiga dipetikan dan

diisukan sebagai instruksi pertama dalam siklus selanjutnya. Instruksi 4 diisukan sebagai instruksi ke dua. Perhatikan, ketika instruksi 5 tidak diisukan dengan instruksi 4, maka instruksi 5 kembali diprefetch di dalam siklus berikutnya.

Untuk menghindari bahaya *dependency* data, maka digunakan teknik *scoreboard*. Teknik scoreboard diimplementasikan sebagai sebuah vektor bit. Masing-masing bit berkorespondensi ke satu register di dalam arsip register. Pada saat sebuah instruksi diisukan untuk dieksekusi , maka register tujuan (destination register) diberi tanda "busy" dengan mengatur bit *scoreboard* terkait, dan bit *scoreboard* tersebut akan terus tetap dipertahankan nilainya hingga instruksi yang dimaksud selesai dieksekusi, serta hasil eksekusi ditulis kembali ke dalam register tujuan, dan bit *scoreboard* diatur "clear." Catatan, ketika sebuah instruksi (kecuali instruksi *store* dan percabangan) akan diisukan untuk pengeksekusian, maka kondisi bit-bit *scoreboard* untuk keseluruhan register tujuan dan register sumber harus "clear." Sebalinya jika bit-bit *scoreboard* telah diset sebelumnya, maka pengisuan sebuah instruksi akan dipetikan (stalled) sampai bit-bit *scoreboard* dalam kondisi "clear."

Jenis operand. 88110 mendukung delapan jenis operand yang berbeda; byte (8 bit), half-word (16 bit), word (32 bit), double word (64 bit), bilangan floating-point presisi tunggal (32 bit), bilangan floating-point presisi ganda (64 bit). bilangan floating-point presisi ganda extended (80 bit), dan bit field (1 hingga 32 bit dalam register 32 bit).

Set instruksi. 88110 merupakan jenis dari arsitektur RISC, yang memiliki berbagai instruksi yang sederhana, pendek, dan berukuran tetap. Keseluruhan instruksi memiliki lebar 32 bit. Berbagai instruksi yang umum yang digunakan di dalam 88110 adalah :

Transfer data

ld d, s1, s2	Load register dari memori.
ld d, s1, lmm 16	Load isi dari lokasi memori yang ditunjuk ke dalam register d. s1

	memuat alamat basis. Instruksi ld akan menambahkan indeks yang terisi di salah satu register s2 atau lmm 16 dengan s1.
st s, d1, d2	Store register ke memori.
st s, d1, lmm 16	Store isi register ke dalam memori. Register d1 memuat alamat basis. Instruksi st akan menambahkan indeks yang terisi di salah satu register d2 atau lmm 16 dengan d1.
xmem d, s1, s2	Menukar register dengan memori.
xmem d, s1, lmm 16	Menukar (load dan store) isi register d dengan suatu lokasi memori.
Aritmetika dan logika	
add, addu, fadd	Penambahan bertanda, penambahan tidak bertanda, penambahan floating-point.
add d, s1, s2	Tambah s1 ke s2
add d, s1, lmm 16	
sub, subu, fsub	Pengurangan bertanda, Pengurangan tidak bertanda, Pengurangan floating-point.
sub d, s1, s2	Kurangi s1 dari s2.
sub d, s1, lmm 16	
mul, fmul	Perkalian bilangan bulat, perkalian bilangan floating-point.
mul d, s1, s2	Kalikan s1 dengan s2.

mul d, s1, lmm 16	
div, divu, fdiv	Pembagian bertanda, pembagian tidak bertanda, pembagian floating-point.
div d, s1, s2	Bagi s1 dengan s2.
div d, s1, lmm 16	
cmp, fcmp	Perbandingan bilangan bulat, perbandingan bilangan floating-point.
cmp d, s1, s2	Bandingkan s1 dengan s2
cmp d, s1, lmm 16	Instruksi mengembalikan kondisi yang dievaluasi sebagai sederetan bit ke dalam register d.
and d, s1, s2	s1 **AND** s2.
and d, s1, lmm 16	
or d, s1, s2	s1 **OR** s2.
or d, s1, lmm 16	
xor d, s1, s2	s1 **XOR** s2.
xor d, s1, lmm 16	
Kontrol	
bb0, bb1	Branch pada bit 0, branch pada bit 1.
bb0 Bp5, s1, Disp16	Periksa Bp bit ke-5 pada register s1, jika bit bernilai 0, instruksi akan menuju ke alamat yang terbentuk oleh penambahan displacement Disp16 ke alamat instruksi bb0.
bcnd cc, s1, Disp 16	Percabangan kondisional. Instruksi akan membandingkan isi register s1

	dengan 0, jika isi s1 sesuai terhadap kode kondisi cc, maka lakukan percabangan.
br Disp26	Percabangan tidak kondisional
bsr Disp26	Percabangan ke subrutin
jmp s2	Jump tidak kondisional
jsr s2	Jump ke subrutin
Grafik	
padd d, s1, s2	Tambah field
psub d, s1, s2	Kurangi field
ppack d, s1, s2	Pack pixel
punpk d, s1, s2	Unpack pixel
Pmul d, s1, s2	Perkalian
Prot d, s1, s2	Rotasi
Pcmp d, s1, s2	Perbandingan Z
Sistem	
Ldcr d, CR	Load dari register kontrol.
rte	Kembali dari eksepsi

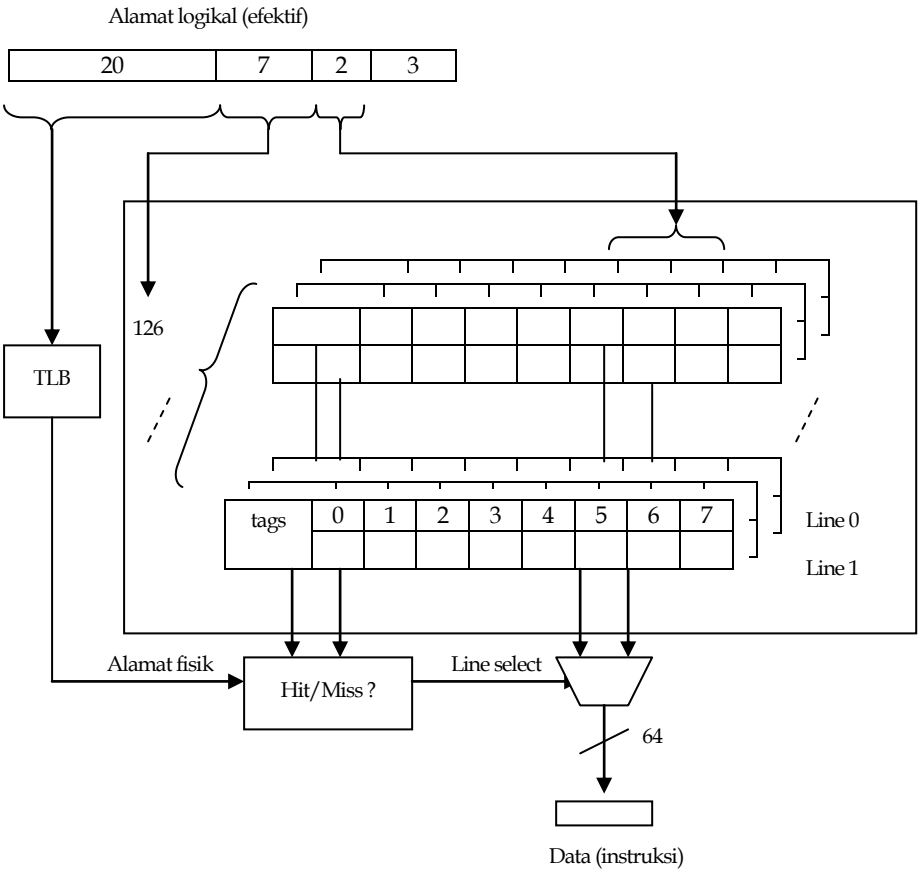
Ket : imm singkatan dari immediate, s (source), d (destination).

Cache instruksi dan cache data. Mikroprosesor 88110 memiliki cache on-chip terpisah untuk instruksi dan data. Cache instruksi menyediakan unti instruksi, dan cache data menyediakan unit eksekusi load-store. Masing-masing cache tersebut adalah berukuran 8-kbyte, dua-arah, cache set-asosiatif. Masing-masing cache memori yang berukuran 8-kbyte secara logikal

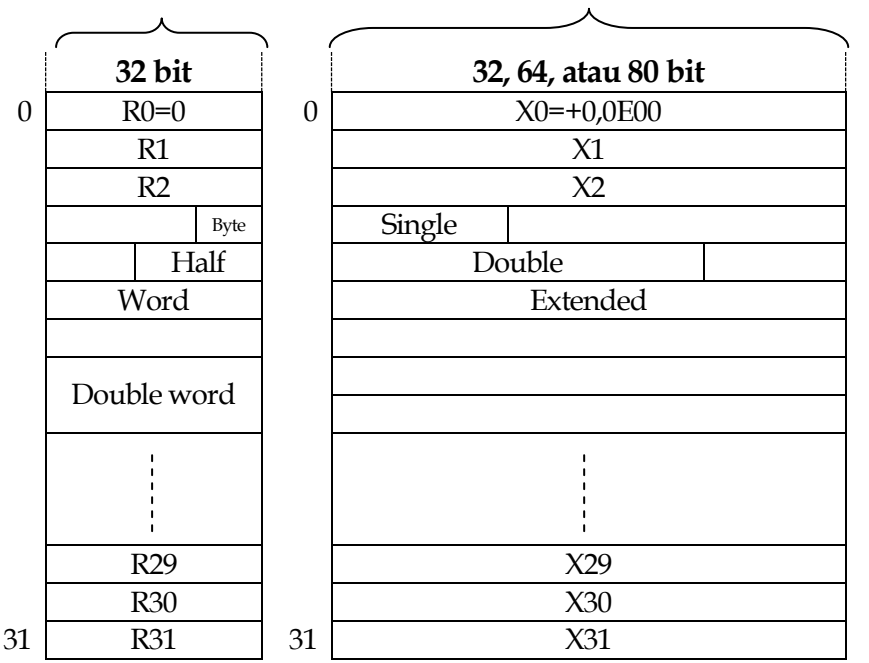
diorganisasikan sebagai dua blok memori, masing-masing memuat dua buah 128 line. Masing-masing line pada cache data memuat 32 byte (8 buah word 32 bit), tag alamat 20 bit, dan 3 bit state. Masing-masing line pada cache instruksi memuat 32 byte (8 buah word 32 bit), tag alamat 20 bit, dan 1 bit valid. Kedua cache menggunakan strategi *replacement* acak untuk menggantikan sebuah line cache pada saat ketidaktersediannya line cache yang kosong.

3 bit state pada cache data berfungsi menentukan status line cache, valid atau tidak valid, termodifikasi atau tidak, dan *shared* atau eksklusif. State termodifikasi bermaksud bahwa line telah termodifikasi akibat operasi penyalinan memori. State eksklusi bermaksud bahwa hanya line cache ini yang memenuhi nilai valid yang identik ke penyalinan memori. State *shared* mengindikasikan bahwa terdapat sedikitnya satu perangkat cache lainnya yang menyimpan atau menggunakan data yang sama yang identik ke penyalinan memori.

Pada masing-masing cache data dan cache instruksi terdapat alamat TLB atau Translation Lookaside Buffer. TLB merupakan cache asosiatif penuh dengan 40 entri dan mendukung lingkungan memori virtual. Masing-masing entri TLB memuat alamat translasi, kontrol, dan informasi proteksi untuk translasi alamat logikal-ke-fisik. Gambar 4.4 menunjukkan, cache secara logikal diberi indeks dan secara fisik diberi tag. 12 bit yang paling rendah pada alamat logikal menentukan lokasi yang mungkin dari sebuah instruksi atau data di dalam cache. 20 atau 13 bit teratas pada alamat logikal dimaksudkan ke TLB, yang mana selanjutnya bit-bit tersebut ditranslasikan ke satu alamat fisik. Entri TLB tersebut yang bertugas menyesuaikan alamat halaman logikal yang digunakan untuk memberikan alamat fisik terkait. Alamat fisik yang diperoleh selanjutnya dibandingkan dengan dua tag cache untuk menentukan kondisi "hit" atau "miss."



Gambar 4.4 Organisasi cache data dan cache instruksi (Computer Architecture : Single and Parallel System, Mehdi R. Zargham, PHIE)



Gambar 4.5 File register (a) general dan (b) extended atau floating-point (Computer Architecture : Single and Parallel System, Mehdi R. Zargham, PHIE)

88110 memiliki hardware yang mampu mendukung tiga mode update memori; write-back, write-through, dan cache-inhibit. Mode write-back digunakan sebagai mode default, perubahan pada data "cacheable" tidak menyebabkan siklus bus eksternal meng-update memori utama. Dalam mode write-through, memori utama akan selalu diupdate kapan saja line cache dimodifikasi. Di dalam mode cache-inhibit, operasi read/write hanya mengakses memori utamanya saja. Jika suatu lokasi memori didentifikasikan sebagai cache-inhibit, data dari lokasi ini tidak pernah disimpan di dalam cache data. Sebuah bit di dalam entri TLB sangat menentukan apakah suatu lokasi memori dikategorikan cache-inhibit.

Bus internal. 88110 berisikan 6 bus 80 bit : dua bus s1, dua bus s2, dan dua bus d (destination). Bus s(s1 dan s2) bertugas memindahkan berbagai operand sumber dari register (atau dari nilai Imm 16 yang tersimpan dalam instruksi) ke unit eksekusi. Bus d bertugas memindahkan hasil dari unit eksekusi ke file register.

File register. 88110 memiliki dua file register, satu untuk integer dan alamat pointer, dan yang lain untuk bilangan floating-point. Perhatikan gambar 4.5, File register integer terdiri dari 32 register 32 bit dan file register floating-point memiliki 32 register 80 bit. Gambar 4.6 memperlihatkan, kedua register memiliki 8 port : 6 port output dan dua port input. Empat output port digunakan untuk menempatkan berbagai operand sumber ke berbagai bus sumber secara serempak. Fenomena tersebut memungkinkan dua instruksi dieksekusi per siklus clock. Dua output port yang lain digunakan untuk menyalin isi register d untuk instruksi yang sedang dieksekusi ke dalam buffer history. Port input digunakan untuk mentransfer hasil pada dua bus d ke register d.

Unit load/store. Unit load/store memberikan akses cepat ke cache data. Unit load/store mengeksekusi keseluruhan instruksi yang mentransfer data antara file register dan cache data atau memori. Ketika sebuah instruksi diterima oleh Unit load/store, maka instruksi tersebut akan menunggu pengaksesannya ke cache data dalam salah satu antrian load atau antrian store. Antrian tersebut bersifat FIFO dan memungkinkan eksekusi normal berlanjut pada saat beberapa instruksi lainnya sedang menunggu pelayanan.

Unit penambahan floating-point. (jelas)

Unit multiplier. (jelas)

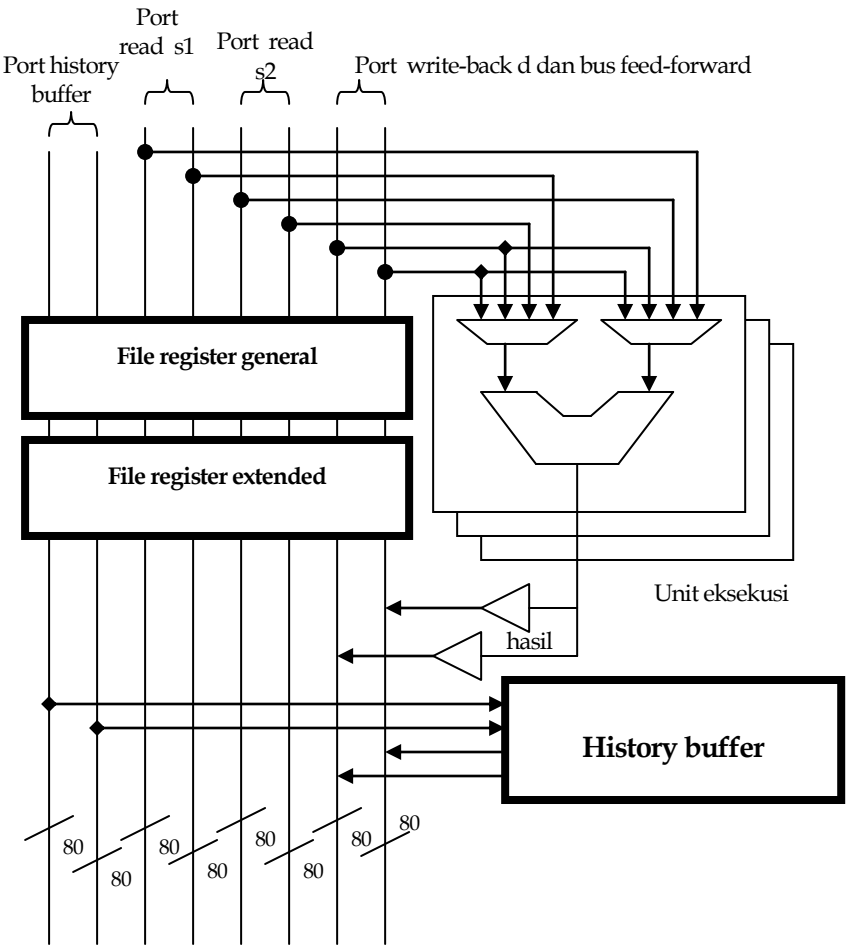
Unit divider. (jelas)

Unit integer. (jelas)

Unit graphic. (jelas)

Unit bit-field. Unit eksekusi bit-field 32 bit memuat sebuah sirkuit shifter atau sirkuit masker yang bertugas menangani instruksi manipulasi bit-field. Unit eksekusi bit-field 32 bit memiliki latency

eksekusi *one-clock-cycle* dan mampu menerima sebuah instruksi baru setiap siklus clock.



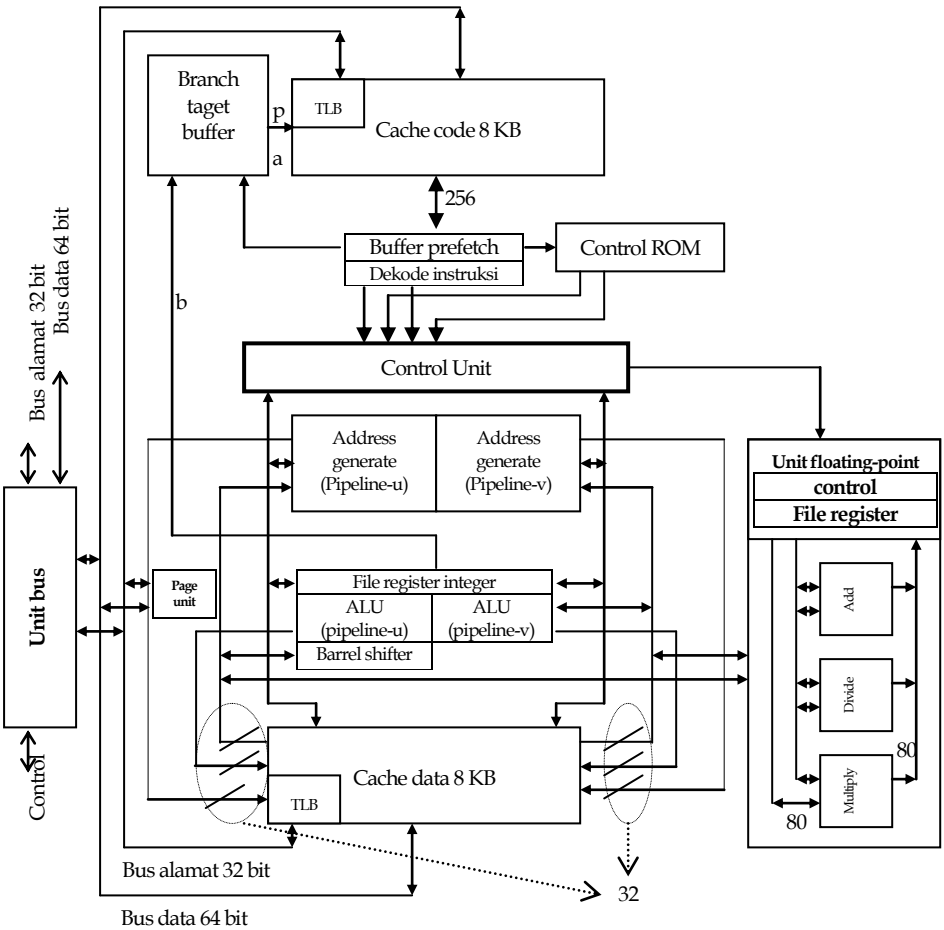
Gambar 4.6 Jalur/path operand data (Computer Architecture : Single and Parallel System, Mehdi R. Zargham, PHIE)

MIKROPROSESOR INTEL PENTIUM

Mikroprosesor Intel Pentium (66 MHZ, 112 MIPS, prosesor 32 bit) merupakan mikroprosesor superscalar *single-chip* yang diterapkan di dalam teknologi BiCMOS (bipolar complementary metal-oxide semiconductor). Teknologi BiCMOS menggunakan berbagai fitur bipolar dan fitur CMOS terbaik, yang mampu memberikan kecepatan yang tinggi, pengendalian yang tinggi, dan konsumsi power yang rendah.

Pentium dapat mengeksekusi dua instruksi per siklus clock, memiliki sifat CISC dan RISC, tetapi memiliki kecenderungan berkarakteristik CISC daripada RISC, oleh karena itu mikroprosesor Intel Pentium kita ambil sebagai kasus yang merepresentasikan CISC. Beberapa instruksi di-*hardwire* seluruhnya dan dieksekusi dalam satu siklus clock (sifat RISC), sementara instruksi lainnya memanfaatkan mikroinstruksi untuk pengeksekusiannya dan membutuhkan waktu eksekusi lebih dari satu silus clock (sifat CISC). Pentium memiliki beberapa mode pengalamatan, beberapa format instruksi, dan sedikit register (sifat CISC).

Gambar 4.7 menunjukkan berbagai komponen utama prosesor Pentium. Pentium memiliki dua pipeline instruksi, pipeline-u dan pipeline-v. pipeline-u dapat mengeksekusi semua instruksi integer dan instruksi floating-point, sebaliknya pipeline-v dapat mengeksekusi instruksi integer yang sederhana dan beberapa instruksi floating-point, atau berbagai instruksi sederhana yang tidak membutuhkan sebarang mikroinstruksi untuk pengeksekusiannya.



keterangan (a) = address, (p) = prefetch, dan (b) = branch verify and target address.

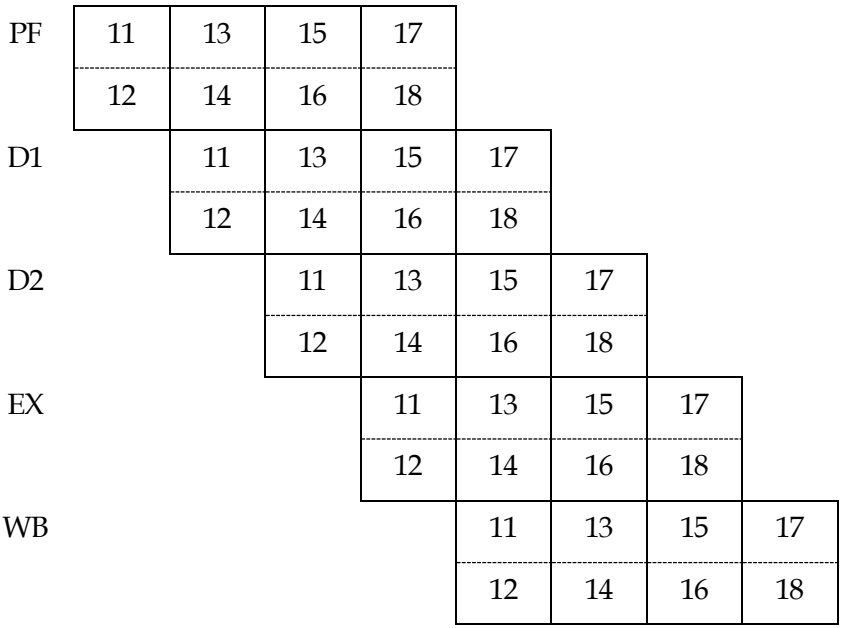
Gambar 4.7 Blok diagram prosesor Pentium (Computer Architecture : Single and Parallel System, Mehdi R. Zargham, PHIE),

Pentium memiliki dua cache 8-Kbyte, cache data dan cache instruksi, memiliki satu buffer target percabangan, dua buffer

prefetch, dan satu ROM kontrol. Cache instruksi, buffer target percabangan, dan buffer *prefetch* bertanggung jawab dalam menyediakan instruksi ke unit eksekusi. ROM kontrol berisikan sekumpulan mikroinstruksi untuk pengontrolan sebarisan operasi yang terlibat di dalam eksekusi instruksi.

Pentium memiliki dua unit, satu unit integer, dan yang lain unit floating-point. Bus-bus yang digunakan di dalam pentium adalah bus data 32 bit untuk unit integer, bus data internal 80 bit untuk unit floating-point, dan bus data internal untuk cache kode berukuran 256 bit.

Pipeline instruksi. Pentium dapat mengeksekusi dua intruksi integer per siklus clock melalui pipeline-u dan pipeline-v. Gambar 4.8 menunjukkan lima stadium yang dimiliki oleh masing-masing pipeline.



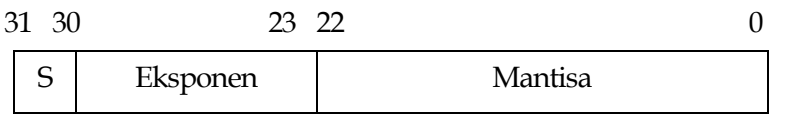
Gambar 4.8 Eksekusi pipeline Pentium

Kelima stadium tersebut adalah prefetch (PF), instruction decode (D1), address generate (D2), execute (EX), dan write-back (WB).

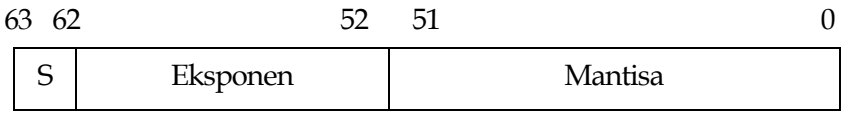
Jenis operand. Pentium mendukung beberapa jenis operand yang meliputi : byte (8 bit), word (16 bit), double word (32 bit), dan quad word (64 bit). Jenis-jenis operand khusus yang dimiliki Pentium meliputi : integer (32 bit, 16 bit, 8 bit), integer BCD, near pointer (32 bit), bit-field (1 hingga 32 di dalam register), bit string (dari sebarang posisi hingga 232-1 bit), byte string (string dapat berisi 0 hingga 232-1 byte), floating-point presisi tunggal (32 bit), floating-point presisi ganda (64 bit), dan floating-point presisi extended (80 bit).

Sama seperti Motorolla 88110, Pentium juga menggunakan format floating-point standard IEEE untuk merepresentasikan bilangan riil.

Bilangan floating-point, berisikan 2 bagian, yaitu mantisa (signifikan atau pecahan), dan eksponen. Gambar 4.9 memperlihatkan bentuk 4 byte dan 8 byte dari bilangan floating-point yang disimpan di dalam sistem Intel. Bilangan floating-point atau bilangan real 4 byte disebut *single-precision*, dan Bilangan floating-point atau bilangan real 8 byte disebut *double-precision*.



(a)



(b)

Gambar 4.9 (a) Presisi tunggal menggunakan bias 7FH, dan (b) Presisi Ganda menggunakan bias 3FFH

Mantisa 24-bit memiliki bit satu yang tersembunyi (implied), yang memungkinkan mantisa untuk mewakili 24-bit walau hanya disimpan 23-bit. Bit yang disembunyikan adalah bit pertama dari

bilangan yang dinormalisasi. Pada saat menormalisasi bilangan, bit ini diatur sehingga nilainya sekurang-kurangnya 1, tetapi kurang dari 2. sebagai contoh, jika 12 diubah ke biner (1102_2), maka dinormalisasi dan hasilnya $1{,}1 \times 2^3$. Bit satu yang terdapat di depan tanda koma akan disembunyikan, atau tidak disimpan di dalam bagian mantisa.

Eksponen disimpan dalam eksponen terbias (*biased exponent*). Untuk presisi tunggal, Bias 7FH (127) akan dijumlahkan ke eksponen sebelum disimpan ke dalam tempat eksponen, dan untuk presisi ganda sebesar 3FFH (1023).

Ada 2 pengecualian mengenai aturan-aturan yang diterapkan mengenai bilangan floating-point. Angka 0,0 disimpan semuanya sebagai nol. Bilangan tak hingga disimpan dalam eksponen sebagai satu, dan dalam mantisa semuanya sebagai nol. Bit tanda menunjukkan bilangan tak hingga tersebut bernilai positif atau negatif.

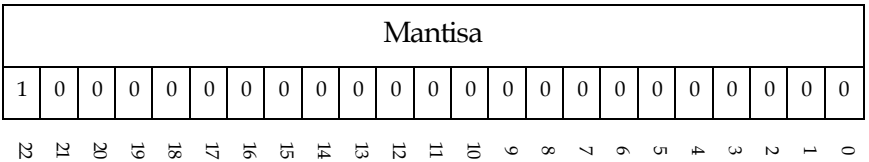
Kasus 1

Bagaimana bilangan +12 disimpan dalam bentuk presisi tunggal.

Penyelesaian :

Desimal	Biner	Normal	Tanda
+12	1100	$1{,}1 \times 2^3$	0

s	Eksponen							
0	1	0	0	0	0	0	1	0
31	30	29	28	27	26	25	24	23



Penjelasan :

+ = 0, pangkat pada normalisasi adalah 3 atau dalam bentuk biner sama dengan 11_2. Selanjutnya untuk presisi tunggal eksponen dapat diperoleh dengan menambahkan 11_2 +7FH = $11_2 + 1111111_2 = 10000010_2$.

Set instruksi. Berdasarkan karakteristik CISC, Pentium memiliki ukuran dan kombinasi yang bervariasi dari berbagai instruksi yang sederhana, kompleks, pendek, dan panjang.

Bab 5

ARSITEKTUR SYSTOLIC ARRAY dan ARSITEKTUR DATA FLOW

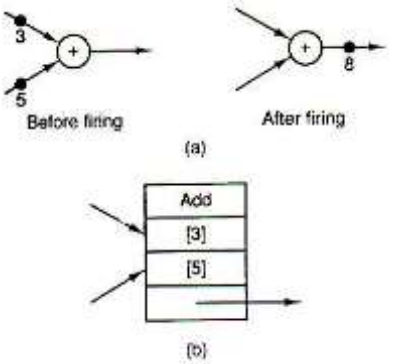
PENDAHULUAN

Di dalam arsitektur *data flow* sebuah instruksi siap untuk dieksekusi ketika data untuk operandnya juga telah tersedia. Ketersedian data diperoleh dengan menyalurkan berbagai hasil dari eksekusi instruksi sebelumnya ke dalam berbagai operand instruksi tunggu. Penyaluran (channeling) ini membentuk sebuah aliran data (data flow), yang memicu berbagai instruksi untuk dieksekusi. Maksud dari desain tersebut adalah pengeksekusian berbagai instruksi secara serempak, dengan resiko kemungkinannya adalah komputasi yang sangat tinggi.

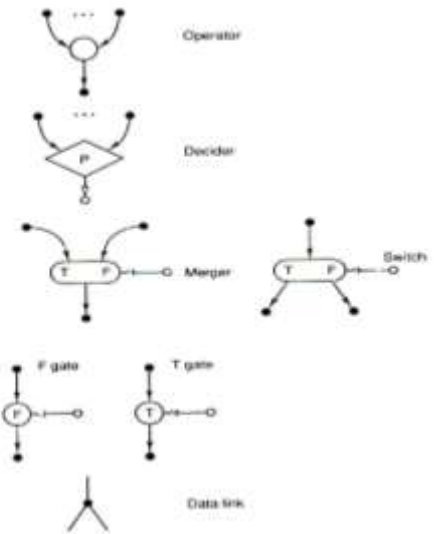
Arsitektur *systolic array* memiliki sejumlah prosesor sederhana yang identik, atau elemen pemrosesan (PEs). PEs disusun ke dalam bentuk yang terorganisasi baik, seperti array dua dimensi atau array linier. Masing-masing PEs memiliki *storage* privat yang terbatas dan dihubungkan ke PEs terdekatnya.

ARSITEKTUR DATA FLOW

Untuk mendemonstrasikan prilaku mesin *data flow*, kita manfaatkan graf. Graf *data flow* merepresentasikan *dependency* data di antara berbagai instruksi individual.



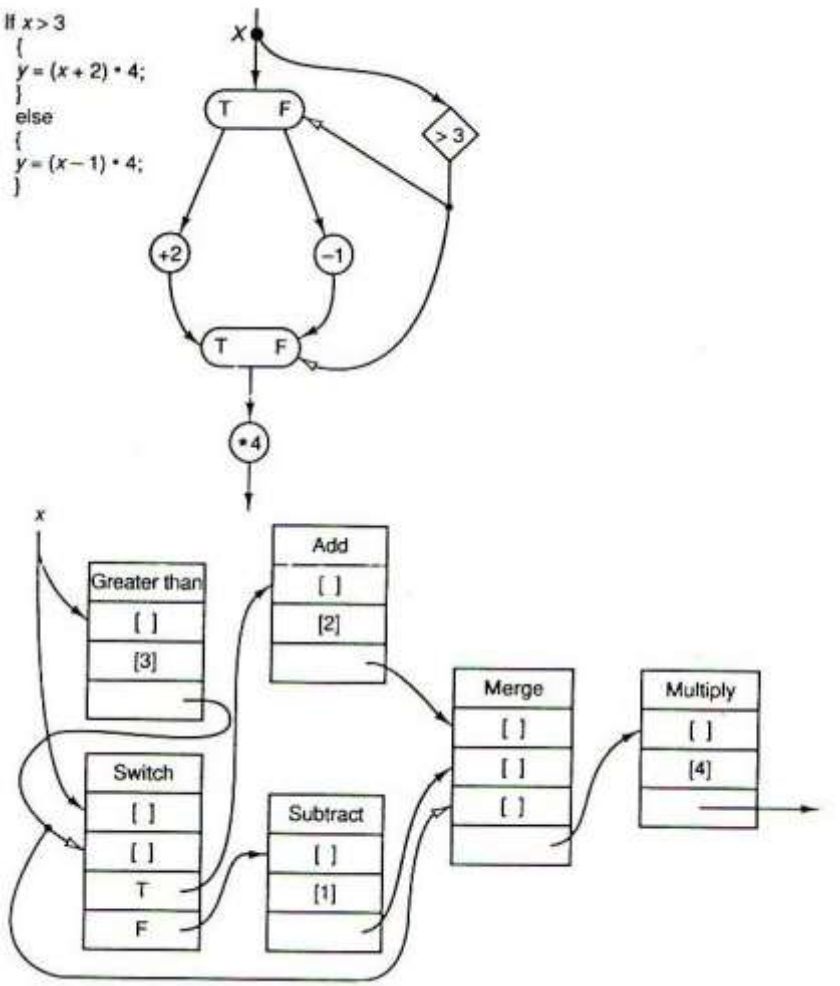
Gambar 5.1 (a) Graf *data flow* dan (b) template aktivitas terkait.



Gambar 5.2 Sekumpulan aktor konstruksi graf *data flow*.

Graf *data flow* merepresentasikan langkah-langkah sebuah program dan memberikan pelayanan sebagai interface antara sistem arsitektur dan bahasa pemrograman yang digunakan oleh user. Berbagai node di dalam graf *data flow* diistilahkan sebagai aktor, yang merepresentasikan berbagai operator dan dikoneksikan oleh berbagai busur panah input dan output yang membawa berbagai token yang menghasilkan nilai-nilai tertentu. Token ditempatkan pada dan dihilangkan dari busur panah didasari oleh aturan-aturan pengeksekusian pada saat itu. Masing-masing aktor membutuhkan berbagai busur input untuk mendapatkan token-token sebelum dia dieksekusi (fired). Ketika token hadir di seluruh busur input yang diperlukan oleh sebuah aktor, maka aktor di-*enable*-kan dan dieksekusi. Selama proses pengeksekusian, aktor akan menghilangkan satu token dari busur input, lalu melakukan suatu fungsi khusus ke berbagai nilai yang berkaitan dengan token, dan menempatkan berbagai token hasil pada busur output (gambar 5.1a). Masing-masing node pada graf *data flow* dapat diilustrasikan sebagai template aktivitas (gambar 5.1b). sebuah template aktivitas terdiri dari berbagai field untuk jenis operasi, token input, dan untuk berbagai alamat tujuan. Gambar 5.2 merepresentasikan sekumpulan aktor yang digunakan di dalam graf *data flow*. Ada dua jenis token : token data dan token Boolean. Untuk membedakan kedua token, dapat dilihat pada gambar, token data menggunakan panah solid, dan token Boolean menggunakan panah *outline*. Aktor switch menempatkan token input pada satu busur output berdasarkan token Boolean yang tiba pada inputnya. Aktor merge menempatkan satu token input pada busur output berdasarkan token Boolean yang tiba pada inputnya. Gerbang T menempatkan token input pada busur output jika token Boolean yang tiba pada inputnya bernilai true. Gerbang F menempatkan token input pada busur output jika token Boolean yang tiba pada inputnya bernilai false. Gambar 5.3 merepresentasikan graf *data flow* dan template terkait untuk pernyataan *if* berikut :

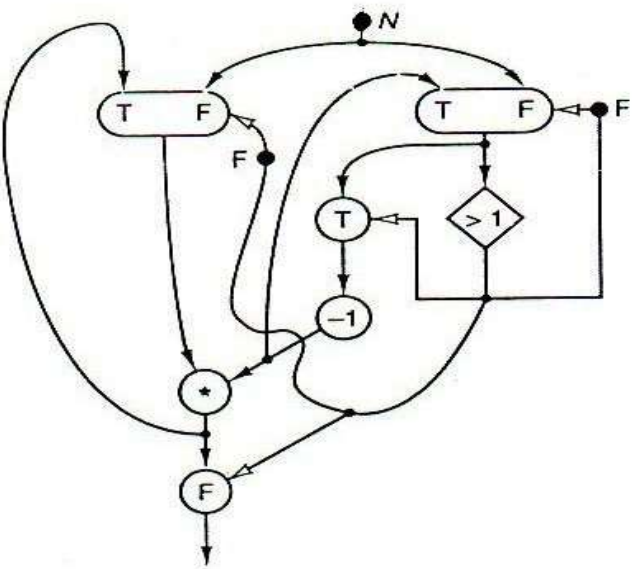
$$\text{if } x > 3 \{y = (x+2) * 4;\} \text{else } \{y = (x-1) * 4;\}.$$



Gambar 5.3 Graf *data flow* dan template aktivitas untuk pernyataan *if* if $x > 3\{y = (x+2)*4;\}$ else $\{y = (x-1)*4;\}$.

Contoh yang lain ditunjukkan pada gambar 5.4, graf *data flow* untuk merepresentasikan perhitungan faktorial (N !).

```
Input N
    i = N;
    While i > 1
        {
        N = N * (i - 1);
        i = i - 1;
        }
Output N
```



Gambar 5.4 Graf *data flow* untuk menghitung $N!$.

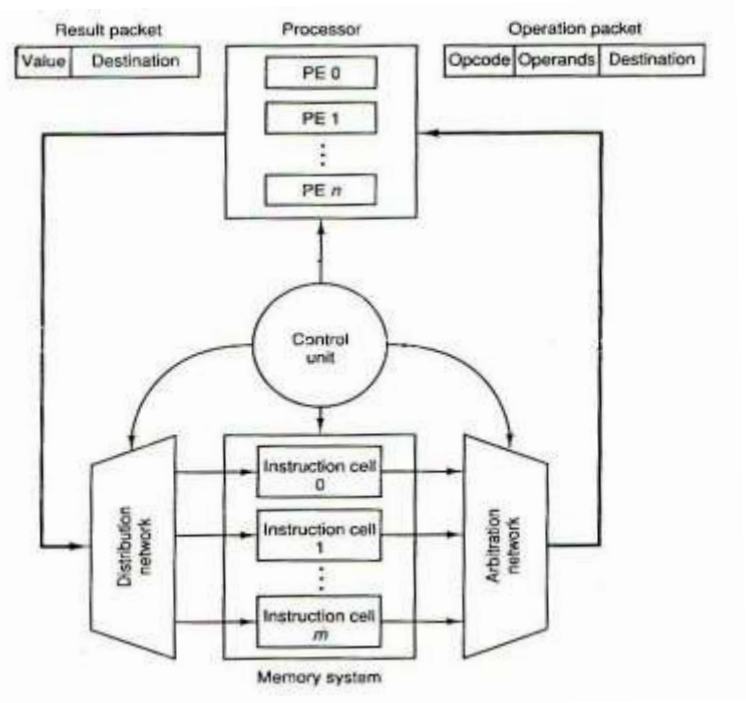
STRUKTUR DASAR KOMPUTER DATA FLOW

Gambar 5.5 merepresentasikan berbagai elemen utama dari sebuah mesin *data flow*, sebagai sampel di ambil komputer *data flow* MIT.

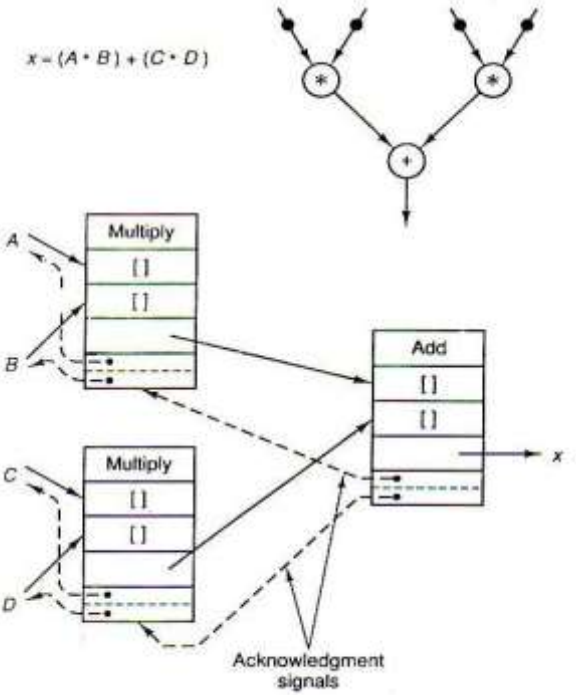
Komputer MIT terdiri dari lima unit utama : (1) unit pengolahan, terdiri dari berbagai elemen pemrosesan khusus, (2) unit memori, terdiri dari sel instruksi yang menyimpan instruksi dan berbagai operand, (3) jaringan arbitrasi, yang mengirim instruksi ke elemen pemrosesan untuk perlakuan eksekusi, (4) jaringan distribusi, untuk memindahkan data hasil dari elemen pemrosesan ke memori, dan (5) unit kontrol, yang mengatur keseluruhan unit.

Satu sel intruksi menyimpan sebuah instruksi yang terdiri dari opcode, operand, dan satu alamat tujuan. Instruksi di-*enable*-kan ketika semua operand dan sinyal kontrol yang dibutuhkan telah diterima. Jaringan arbitrasi mengirim intruksi yang telah di- *enable*-kan sebagai suatu paket operasi ke elemen pemrosesan yang tepat. Setelah instruksi dieksekusi, hasilnya dikirim kembali melalui jaringan distribusi ke lokasi tujuan di dalam memori. Masing-masing hasil dikrim sebagai sebuah paket, yang terdiri dari sebuah nilai dan sebuah alamat tujuan.

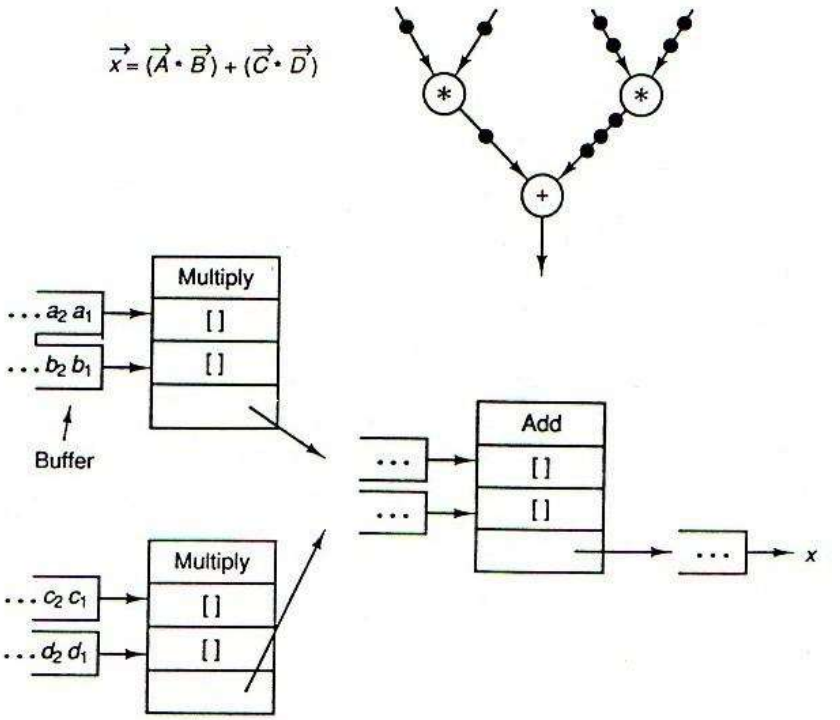
Mesin *data flow* dapat diklasifikasikan menjadi dua kelompok : statis dan dinamis. **Mesin *data flow* statis**, sebuah intruksi di-*enable*-kan ketika seluruh operand yang diperlukan telah diterima dan terdapat sebuah intruksi lainnya yang menunggu hasil dari instruksi tersebut. Gambar 5.6 merepresentasikan sampel mesin *data flow* statis. **Mesin *data flow* dinamis**, sebuah intruksi di-*enable*-kan ketika seluruh operand yang diperlukan telah diterima. Dalam kasus mesin *data flow* dinamis, berbagai operand bisa saja dipergunakan oleh intruksi dalam satu waktu, sehingga satu busur graf *data flow* dinamis dapat memuat lebih dari satu token. Dibandingkan dengan mesin *data flow* statis, pendekatan dinamis lebih memungkinkan pengeksekusian secara paralel, karena sebuah instruksi tidak perlu menunggu lokasi yang kosong dalam instruksi lainnya untuk pengeksekusian sebelum menempatkan berbagai hasil.



Gambar 5.5 Blok diagram *data flow* MIT sederhana.



Gambar 5.6 Sampel graf *data flow* untuk mesin *data flow* statis. Masing-masing busur di dalam graf *data flow* hanya dapat membawa paling banyak satu token pada sebarang pemisalan.



Gambar 5.7 Sampel graf *data flow* untuk mesin *data flow* dinamis. Masing-masing busur di dalam graf *data flow* dapat membawa lebih dari satu token pada sebarang pemisalan.

ARSITEKTUR SYSTOLIC ARRAY

Pada bidang komputasi sains, kita sering dihadapi pada berbagai kasus persamaan linier secara serempak, atau secara khusus, persamaan sistem linier skala besar. Berbagai kajian diperlukan untuk menemukan suatu metode yang cepat, akurat, dan dengan biaya yang efektif untuk penyelesaian berbagai kasus di atas. Menghadapi berbagai komputasi yang besar seperti perkalian, inversi pada matriks, dikembangkanlah sebuah software komputasi berbasis komputer digital *high-speed*. Pengolahan aljabar

matriks menggunakan software berbasis komputer *general-purpose* membutuhkan waktu komputasi yang lama, selain keterbatasan memori untuk mengakomodasi berbagai matriks berskala cukup besar.

Oleh karena itu, untuk mengurangi permasalah di atas, maka dikenalkanlah suatu pendekatan paralel yang diterapkan pada sebuah komputer, yang dikombinasikan dengan berbagai prilaku mesin *special-purpose*. Salah satu solusi terhadap kebutuhan komputasional paralel yang cukup tinggi adalah dengan mengkoneksikan sejumlah besar prosesor identik sederhana atau pengkoneksian berbagai emen pemroses (PEs). Masing-masing PE memiliki *storage* privat yang terbatas, dan setiap PE hanya dikoneksikan ke PE tetangganya yang terdekat. Oleh karena itu PE disusun ke dalam suatu struktur yang terorganisasi secara baik seperti array dua dimensi atau array linier. Karakteristik struktur inilah yang dapat diistilahkan sebagai *systolic array*. *Systolic array* memberikan suatu tatanan ideal untuk pengimplementasian VLSI, ditambah penerapan memori *interleaved* yang lebih difungsikan untuk mensuplai data ke berbagai array tersebut.

Pada umumnya, bentuk geometris s*ystolic array* adalah *hexagonal* dan *rectangular*, dan beberapa berbentuk sebarang geometris lainnya. Berdasarkan skema aliran data untuk perkalian matriks, maka s*ystolic array* dapat dibagi menjadi : *array hexagonal, array pipeline, array semibroadcast, array wavefront,* dan *array broadcast.*

TERMINOLOGI DASAR DAN PROSESOR ARRAY YANG DIAJUKAN

Sebelum memaparkan berbagai varian s*ystolic array*, beberapa terminologi yang sering terlibat di dalam perancangan akan dibahas terlebih dahulu. Pertama, elemen pemrosesan selalu digunakan pada masing-masing desain secara dasar sebagai sebuah prosesor *inner-product step*, yang terdiri dari tiga register : R_a, R_b

dan R_c. Register tersebut digunakan untuk melakukan perkalian dan pertambahan di dalam satu unit waktu komputasional :

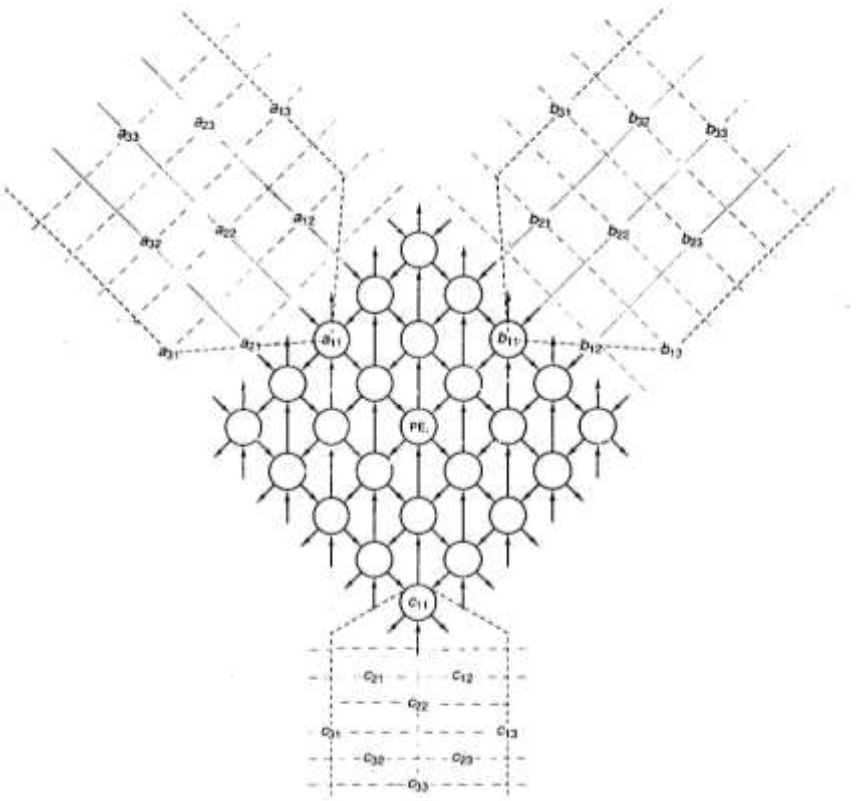
$$R_c = R_c + R_a * R_b$$

Unit waktu komputasional didefinisikan sebagai $t_a + t_m$, di mana t_a dan t_m merupakan waktu yang diperlukan untuk melakukan satu penambahan dan satu perkalian, berturut-turut. Untuk membedakan berbagai prosesor array yang diajukan, maka ada dua faktor yang harus dipertimbangkan : jumlah PE yang diperlukan dan waktu *turnaround*. P mendenotasikan jumlah PE yang diperlukan, dan T atau waktu *turnaround*, didefinisikan sebagai total waktu di dalam unit $t_a + t_m$ yang dibutuhkan untuk melengkapi suatu komputasi yang sedang berlangsung.

Pada paragraf berikut, kita akan membahas beberapa arsitektur s*ystolic array* yang diajukan. Struktur masing-masing desain diilustrasikan oleh sebuah contoh yang melakukan komputasi $C = A * B$, di mana :

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \quad B = \begin{bmatrix} b_{11} & b_{12} & c_{13} \\ b_{21} & b_{22} & c_{23} \\ b_{31} & b_{32} & c_{33} \end{bmatrix} \quad C = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix}$$

Array hexagonal. Array hexagonal, diajukan oleh Kung dan Leiserson, merupakan contoh efektif dari penggunaan sejumlah besar PE yang terstruktur dengan pengorganisasian yang baik.



Gambar 5.8 Blok diagram *array hexagonal*.

Dalam array hexagonal, masing-masing PE memiliki sebuah fungsi yang sederhana dan PE saling berkomunikasi dengan PE terdekatnya menggunakan pemodelan pipeline. PE yang terletak di sekitar garis batas berkomunikasi dengan dunia di luarnya. Gambar 5.8 mengilustrasikan model array hexagonal untuk kegunaan multiplikasi matriks 3x3, A dan B. Masing-masing PE yang digambarkan dengan lingkaran memiliki tiga input dan tiga ouput. Berbagai elemen matriks A dan B memasuki array dari barat dan timur sepanjang aliran-aliran data kedua diagonal. Berbagai entri matriks C dengan nilai awal nol, berpindah dari selatan ke

utara array, dan nilai input dan output berpindah melalui PE pada setiap pulsa clock. Sebagai contoh, ambil situasi pada gambar 5.8, nilai input a_{11} dan b_{11}, dan nilai output c_{11} tiba pada PE_i yang sama setelah dua pulsa clock. Pada saat keseluruhan nilai tersebut tiba, PE melakukan perhitungan nilai c_{11} yang baru dengan operasi :

$$c_{11} = c_{11} + a_{11} + b_{11}$$

Pada gambar 5.8 terdapat 25 PE, dan hanya 19 PE saja yang berkontribusi terhadap komputasi di atas. Asumsikan bahwa elemen pertama data masukan memasuki array setelah satu unit waktu, dan waktu *turnaround* untuk array tersebut adalah 10 unit. Maka untuk array hexagonal dengan matriks $n \times n$ akan diperoleh :

$P = (2n-1)^2$

$T = 4n-2$ ketika sebagian hasil tiba di array,

$T = 5n-3$ ketika hasil telah ditransfer keluar array.

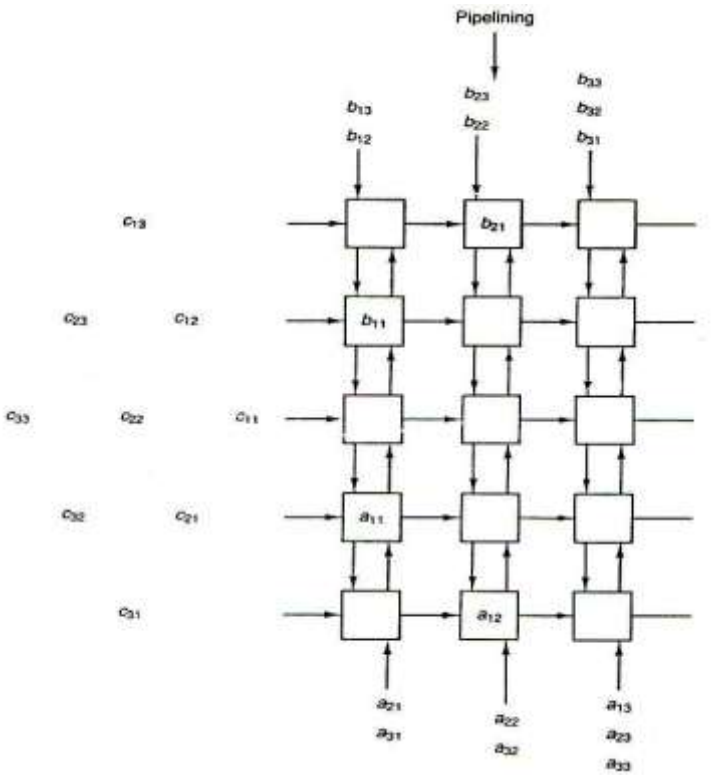
Catatan, hanya $3n^2 - 3n + 1$ di luar dari $(2n-1)^2$ PE berkontribusi terhadap pengkomputasian di atas. Kekurangan utama dari array hexagonal adalah bahwa mereka tidak mengunjukkerjakan "pipe" berbagai elemen data ke dalam setiap PE dalam setiap unit waktu. array hexagonal membutuhkan jumlah PE yang besar, untuk itu berbagai model alternatif lainnya telah diajukan.

Array pipeline. Array pipeline diajukan oleh Hwang dan Cheng, dilustrasikan pada gambar 5.9. Arsitektur ini memiliki bentuk desain *rectangular*. Elemen matriks A dan B dimasukan dari baris input teratas dan terendah dengan meniru prilaku pipeline. Maka untuk array pipeline dengan matriks $n \times n$ akan diperoleh :

$$P = n(2n-1)$$

$$T = 4n-2$$

Array pipeline secara relatif memiliki desain yang sederhana. Aliran data yang digunakan Array pipeline dapat dikatakan sama dengan aliran data yang digunakan pada Array hexagonal, tetapi memiliki sedikit PE. Sama seperti array hexagonal, Array pipeline tidak mengunjukkerjakan "pipe" berbagai elemen data ke dalam setiap PE dalam setiap unit waktu. Waktu tunggu yang terjadi sedikitnya 50% dari PE dalam sebarang waktu.



Gambar 5.9 Blok diagram *array pipeline*.

Array semibroadcast. Array semibroadcast diajukan oleh Huang dan Abraham. Gambar 5.10 merepresentasikan struktur array

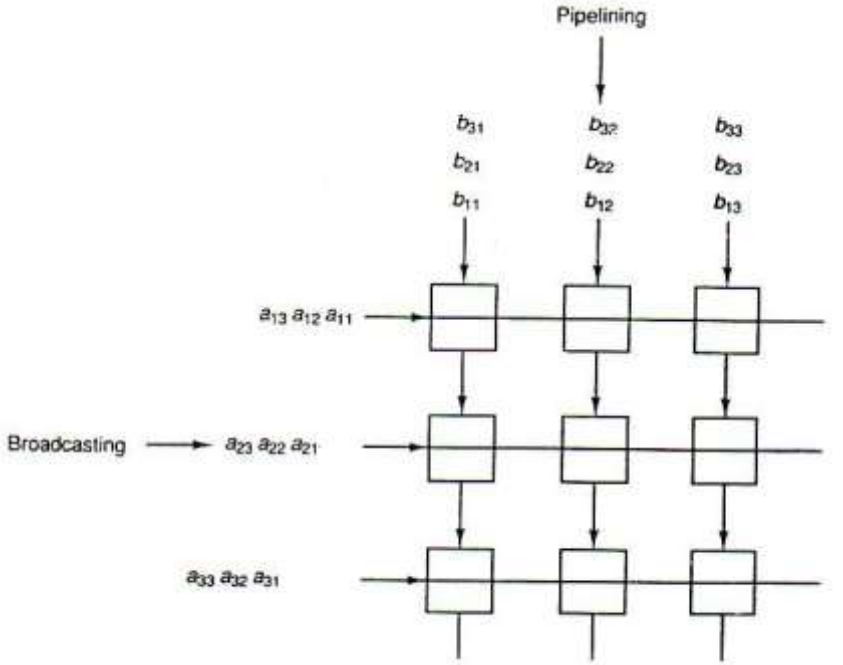
semibroadcast dengan berbagai aliran data. Semibroadcast di sini dimaksudkan untuk broadcast data satu dimensi. Untuk contoh, perhatikan gambar 5.10, hanya matriks A yang di-broadcast dari sisi kiri pada array, sementara matriks B di-pipeline-kan ke dalam array dari sisi pucak. Dan matriks C sebagai hasil dapat berada di kiri dalam array atau ditransfer ke luar dari array. Maka untuk array semibroadcast dengan matriks $n \times n$ diperoleh :

$P = n^2$

$T = 2n$ ketika sebagian hasil tiba di array,

$T = 3n$ ketika hasil telah ditransfer keluar array.

Bandingkan dengan desain sebelumnya, array semibroadcast menggunakan sedikit PE dan memerlukan sedikit waktu *turnaround*. Kekurangan yang dimiliki jenis array semibroadcast adalah waktu tunggu perambatan yang diperlukan data broadcast untuk melakukan transfer sedikit lebih lama bila dibandingkan dengan array nonbroadcast.



ambar 5.10 Blok diagram *array semibroadcast*.

Array wavefront. Array wavefront diajukan oleh Kung dan Arun. Struktur array wavefront dengan sebuah aliran data ditunjukkan pada gambar 5.11. Diberikan dua matriks A dan B $n \times n$, matriks A didekomposisikan ke dalam kolom A_i dan matriks B ke dalam baris B_j, oleh karena itu :

$$C = A_1 * B_1 + A_2 * B_2 + \ldots + A_n * B_n$$

Perkalian matriks kemudian dapat diselesaikan dengan iterasi n kali :

$$C^{(k)} = C^{(k-1)} + A_k * B_k \quad \forall k = 1,2,\ldots,n.$$

Iterasi dapat dilakukan dengan menerapkan konsep wavefront. Komputasi untuk keseluruhan iterasi diselesaikan dengan melakukan "pipelining" pada wavefront. Sebagai propagasi awal, kita dapat eksekusikan iterasi kedua secara paralel dengan melakukan pipeline pada wavefront segera setelah iterasi pertama.

Pada gambar 5.11 proses dimulai dengan $PE_{1,1}$ atau $C_{11}^{(1)} = C_{11}^{(0)} + a_{11} * b_{11}$, aktifitas komputasi selanjutnya merambat ke tetangga berikutnya, $PE_{1,2}$ dan $PE_{2,1}$ atau $C_{12}^{(1)} = C_{12}^{(0)} + a_{11} * b_{12}$ dan $C_{21}^{(1)} = C_{21}^{(0)} + a_{21} * b_{11}$, lalu $PE_{1,1}$ dapat mengeksekusikan $C_{11}^{(2)} = C_{11}^{(1)} + a_{12} * b_{21}$. Secara umum, untuk array wavefront dengan matriks $n \times n$ diperoleh :

$P = n^2$

$T = 3n - 2$ ketika sebagian hasil tiba di array,

$T = 4n - 2$ ketika hasil telah ditransfer keluar array.

Array wavefront merepresentasikan struktur yang tersinkroniasi dengan baik. Aliran data yang diterapkan setara atau sama dengan konsep aliran data pada array hexagonal kecuali wavefront tidak menyalurkan berbagai elemen matriks C ke dalam array. Ini berimplikasi bahwa wavefront menyalurkan berbagai data ke dalam PE sepanjang proses komputasi untuk beberapa saat saja. Kekurangan array wavefront adalah pada perfomansi *utilization* hingga maksimum 50%.

Array broadcast. Array broadcast diajukan oleh Chern dan Murata. Struktur array broadcast dengan berbagai aliran data diilustrasikan pada gambar 5.12. Desain tersebut lebih menekankan kepada skema broadcast data dua dimensi. Oleh karena itu, data dapat di-*broadcast* dalam dua arah, berdasarkan baris dan berdasarkan kolom, sepanjang array. Secara umum, untuk array broadcast dengan matriks $n \times n$ diperoleh :

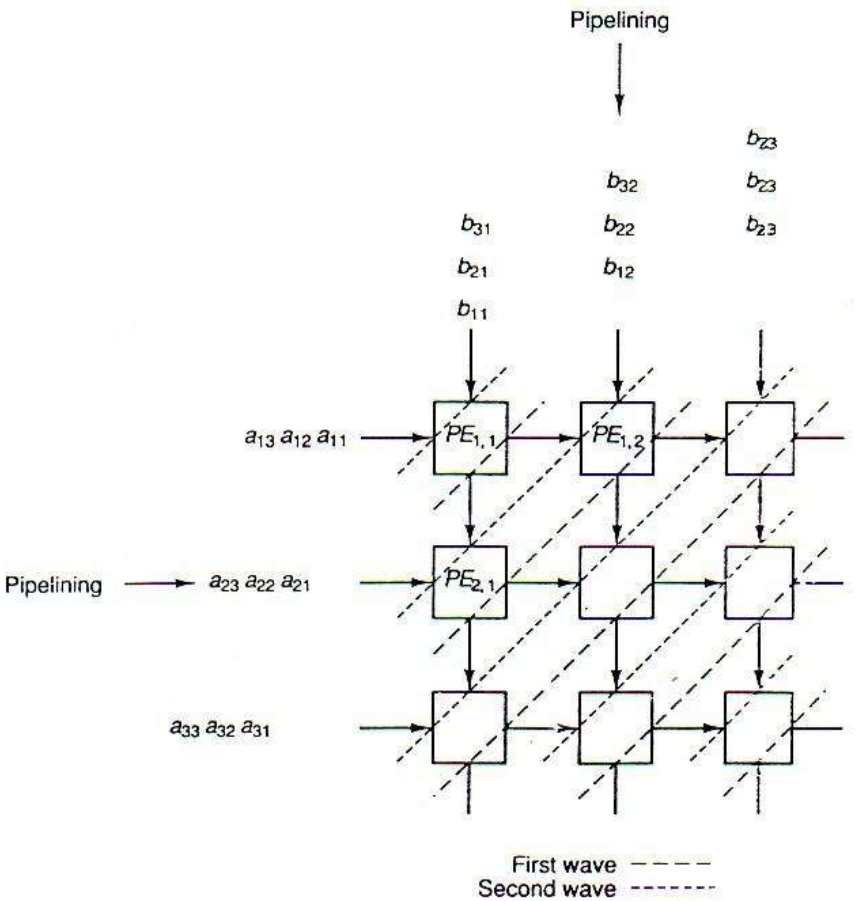
$P = n^2$

$T = n$ ketika sebagian hasil tiba di array,

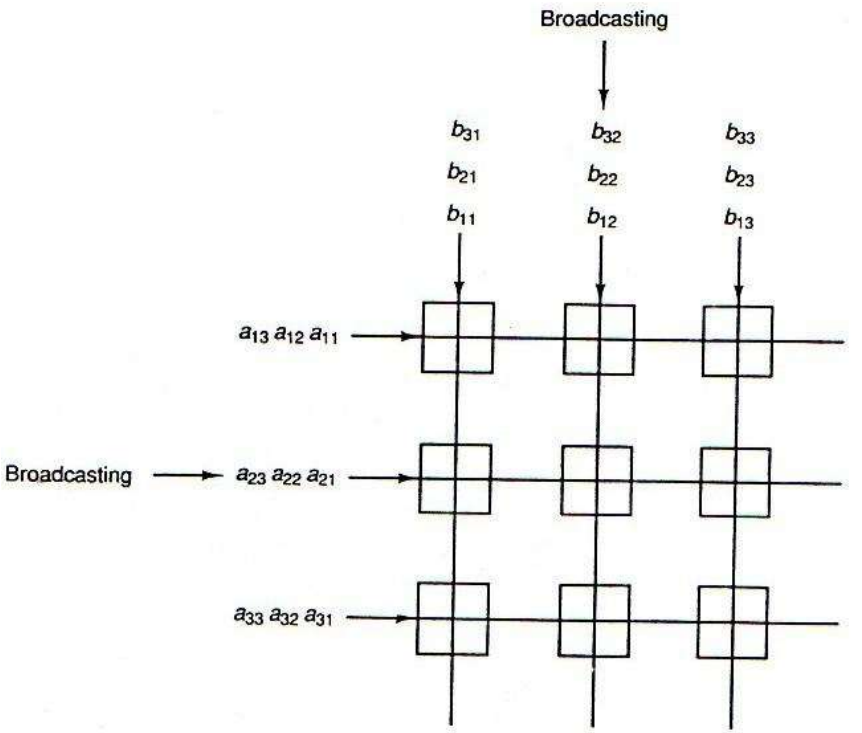
$T = 2n$ ketika hasil telah ditransfer keluar array.

Array broadcast memiliki waktu *turnaround* yang kecil dibandingkan desain sebelumnya, dan memiliki perfomansi *utilization* yang tinggi. Penerapan multiplikasi matriks yang

berukuran besar, tidak memerlukan penataan susunan matriks secara khusus. Sama seperti array semibroadcast, derajat kekompleksitasan untuk mengkontrol array broadcast masih sangat tinggi dibandingkan array pipeline. Baris bus data yang diperlukan oleh array broadcast adalah dua kali atau sebanyak yang diperlukan oleh array semibroadcast, oleh karena itu, desain array broadcast lebih banyak menggunakan daerah *layout*.



Gambar 5.11 Blok diagram *array wavefront*.



Gambar 5.12 Blok diagram *array broadcast*.

ALGORITMA PEMETAAN PADA ARSITEKTUR SYSTOLIC

Kuhn dan Moldovan telah mengajukan berbagai prosedur untuk pemetaan algoritma dengan berbagai loop ke dalam array systolic. Prosedur pemetaan didahului dengan melakukan transformasi berbagai indeks loop ke berbagai indeks loop yang baru. Sebelum membahas berbagai algoritma pemetaan, ada baiknya kita dahului dengan beberapa definisi berikut :

Definisi 1. Algoritma A dapat didefinisikan sebagai lima tuple $A = (I^n, C, D, X, Y)$, di mana,

I^n merupakan indeks berhingga himpunan A.
C merupakan himpunan komputasi A.
D merupakan himpunan dependency data.
X merupakan himpunan variabel input A.
Y merupakan himpunan variabel output A.

Definisi 2.

Dua algoritma $A = (I^n, C, D, X, Y)$ dan $A' = (I'^n, C', D', X', Y')$ dikatakan ekuivalen jika dan hanya jika,

1. Algoritma A adalah ekuivalen input/output terhadap A'; sehingga $X = X'$ dan $Y = Y'$.
2. Himpunan indeks A' merupakan himpunan indeks yang ditransformasikan dari A. $I'^n = T(I^n)$, di mana T merupakan fungsi monoton dan bijektif.
3. Sebarang operasi pada A berkorespondensi terhadap suatu operasi identik dalam A', dan sebaliknya; sehingga $C = C'$.
4. Dependency data pada A' merupakan dependency data yang ditransformasikan pada A; $D' = T(D)$.

Matriks T dipartisi menjadi dua fungsi sebagai berikut :

$$T = \begin{bmatrix} \prod \\ S \end{bmatrix}$$

Di mana $\prod : I^n \rightarrow I'^k \ (n > k)$, dan $S : I^n \rightarrow I'^{n-k}$.

k koordinat pertama dari elemen I'^n dikaitkan terhadap waktu. $n-k$ koordinat kedua dikaitkan ke sifat-sifat geometris algoritma. Lalu, perhatikan fungsi yang ditransformasikan berikut :

$$\prod : I^n \rightarrow I'^1$$

$$S : I^n \rightarrow I'^{n-1}$$

Pemetaan $\prod$ dipilih supaya matriks dependency data yang ditransformasikan, D', memiliki entri positif pada baris pertama. Hal ini memastikan sebuah susunan yang solid, sehingga, $\prod d_i > 0$ untuk sebarang $d_i \in D$. Oleh karena itu indeks komputasi $I \in I^n$ akan diproses pada waktu $\prod * I$.

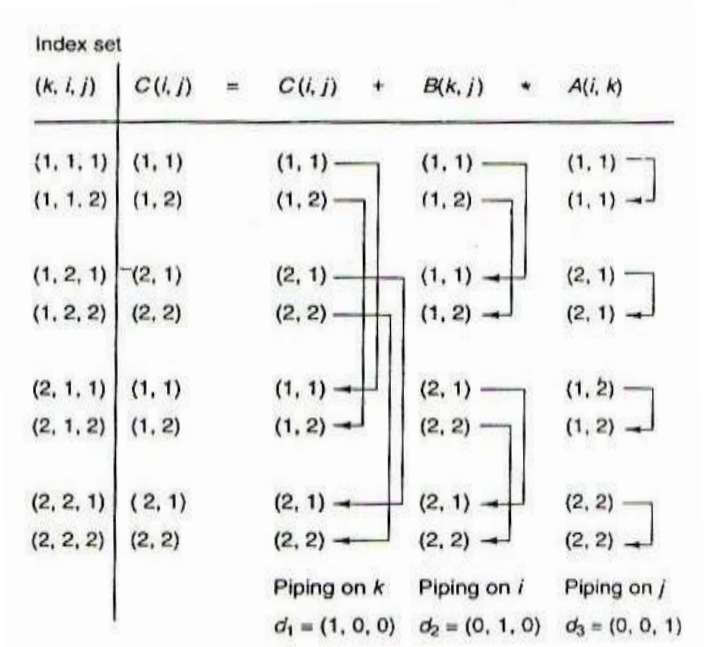
Beberapa tahapan prosedur pemetaan. Berikut adalah langkah-langkah yang ditempuh di dalam prosedur pemetaan,

1. Buffer seluruh variabel.
2. Tentukan fungsi PE dengan mengumpulkan berbagai pernyataan *assigment* pada tubuh loop ke dalam fungsi m input dan n output.
3. Aplikasikan transformasi indeks linier T.
4. Temukan koneksi antara prosesor dan arah aliran data.

Sebagai contoh perhatikan algoritma berikut yang merepresentasikan perkalian dua matriks A dan B, 2×2.

```
for (k=1; k<=2; k++)
 for (i=1; i<=2; i++)
  for (j=1; j<=2; j++)
    C(i,j)= C(i,j)+B(k,j)*A(i,k);.
```

Gambar 5.13 merepresentasikan himpunan indeks untuk algoritma di atas. Dalam gambar, masing-masing elemen indeks ditampilan



sebagai tuple (k,i,j). Catatan, kedua indeks $(k,i,1)$ dan $(k,i,2)$, dan beberapa nilai yang sama pada $A(i,k)$ tetap digunakan, agar, nilai $A(i,k)$ dapat disalurkan pada arah j. Sebenarnya, secara sederhana nilai $B(k,j)$ dan $C(i,j)$ dapat disalurkan pada arah i dan k berturut-turut. Berdasarkan kenyataan tersebut, algoritma di atas dapat dinyatakan kembali dengan memasukan berbagai variabel buffer A^{j+1}, B^{i+1}, dan C^{k+1}, sebagai berikut :

Gambar 5.13 Himpunan indeks

```
for (k=1; k<=2; k++)
 for (i=1; i<=2; i++)
  for (j=1; j<=2; j++)
    {
```

```
        A^{j+1}(i,k)=A^j(i,k);
        B^{i+1}(k,j)=B^i(k,j);
        C^{k+i}(i,j)=
C^k(i,j)+B^i(k,j)*A^j(i,k);
     }.
```

Sekumpulan vektor *dependency* data dapat ditentukan dengan mempersamakan berbagai indeks dari seluruh pasangan yang mungkin berbagai variabel yang dihasilkan dan berbagai variabel yang digunakan. Di dalam kode sebelumnya, variabel yang dihasilkan $C^{k+1}(i,j)$ dan variabel yang digunakan $C^k(i,j)$ memberikan $d_1 = (k+1-k, i-i, j-j) = (1,0,0)$. Secara sederhana, $< B^{i+1}(k,j)$ dan $B^i(k,j) >$ dan $< A^{j+1}(i,k)$ dan $A^j(i,k) >$ memberikan $d_2 = (0,1,0)$ dan $d_3 = (0,0,1)$. Matriks *dependency* $D = [d_1 \mid d_2 \mid d_3]$ dapat diekspresikan sebagai :

$$D = \begin{bmatrix} d_1 & d_2 & d_3 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Lalu kita cari transformasi T yang merupakan peningkatan bijektif dan monoton dari bentuk :

$$T = \begin{bmatrix} \Pi \\ S \end{bmatrix}$$

di mana $\prod d_i > 0$, andaikan,

$$T = \begin{bmatrix} t_{11} & t_{12} & t_{13} \\ t_{21} & t_{22} & t_{23} \\ t_{31} & t_{32} & t_{33} \end{bmatrix}$$

Kondisi $\prod d_i > 0$ (i=1, 2, 3) berakibat,

$$(t_{11}t_{12}t_{13})\begin{bmatrix}1\\0\\0\end{bmatrix} > 0 \rightarrow t_{11} > 0,$$

$$(t_{11}t_{12}t_{13})\begin{bmatrix}0\\1\\0\end{bmatrix} > 0 \rightarrow t_{12} > 0,$$

$$(t_{11}t_{12}t_{13})\begin{bmatrix}0\\0\\1\end{bmatrix} > 0 \rightarrow t_{13} > 0.$$

Untuk mengurangi waktu *turnaround*, kita ambil nilai terkecil untuk t_{11}, t_{12}, dan t_{13} sehingga,

$$t_{11} = t_{12} = t_{13} = 1; \text{ dan } \prod = (1,1,1).$$

Sebagai contoh, kita ambil satu indeks dari sekumpulan indeks pada gambar 5.13, misal (1, 1, 2), maka,

$$\prod \begin{bmatrix}1\\1\\2\end{bmatrix} = (1,1,1)\begin{bmatrix}1\\1\\2\end{bmatrix} = 4.$$

Nilai 4 mengindikasikan waktu referensi pada komputasi yang berkorespondensi terhadap indeks (1, 1, 2) yang telah dilakukan.

Pilihan kita selanjutnya adalah, bahwa S memainkan peran terhadap interkoneksi berbagai prosesor. Dalam penseleksian pemetaan S, kita dihadapkan pada fakta bahwa T harus bijektif dan memuat sejumlah bilangan integer. Sejumlah kemungkinan yang ada akan mengarahkan ke geometris jaringan yang berbeda. Dua pilihan yang dihadapi, misalkan,

$$S_1 = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{dan } S_2 = \begin{bmatrix} -1 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}.$$

Andai S_1 yang kita pilih, maka

$$T = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Secara umum, untuk perkalian dua matriks $\mathrm{n} \times \mathrm{n}$ akan diperlukan 2^n PE. Berdasarkan hal tersebut, sebagai sampel kita ambil empat PE. Maka interkoneksi antar prosesor didefinisikan sebagai,

$$S_1 d_i = \begin{bmatrix} x \\ y \end{bmatrix},$$

di mana x dan y berkaitan terhadap perpindahan variabel sepanjang arah i dan j, secara berturut-turut. Oleh karena itu,

$$S_1 d_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

mengindikasikan bahwa variabel c tidak mengalami perpindahan di segala arah dan diperbaharui di dalam waktu tersebut.

$$S_1 d_2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix},$$

mengindikasikan bahwa variabel b berpindah sepanjang arah i dengan kecepatan satu grid per unit waktu. Dan

$$S_1 d_3 = \begin{bmatrix} 0 \\ 1 \end{bmatrix},$$

mengindikasikan bahwa variabel a berpindah sepanjang arah j dengan kecepatan satu grid per unit waktu.

Gambar 5.14 merepresentasikan interkoneksi antar PE. Pada waktu 1, b_{11} dan a_{11} memasuki $PE_{1,1}$ yang memuat variabel c_{11}. Kemudian masing-masing PE melakukan operasi perkalian dan pertambahan, sehingga $c_{11} = c_{11} + a_{11} * b_{11}$. Pada waktu 2, a_{11} meninggalkan $PE_{1,1}$ dan memasuki $PE_{1,2}$. Pada waktu yang sama, a_{12} memasuki $PE_{1,1}$, b_{12} memasuki $PE_{1,2}$, b_{11} memasuki $PE_{2,1}$, dan b_{21} memasuki $PE_{1,1}$, dan kembali, masing-masing PE melakukan operasi operasi perkalian dan pertambahan. Oleh karena itu,

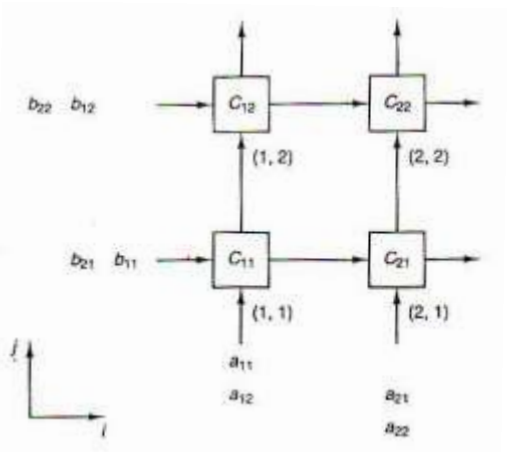
$$c_{11} = c_{11} + a_{12} + b_{21}$$

$$c_{12} = c_{12} + a_{11} + b_{12}$$

$$c_{21} = c_{211} + a_{21} + b_{11}$$

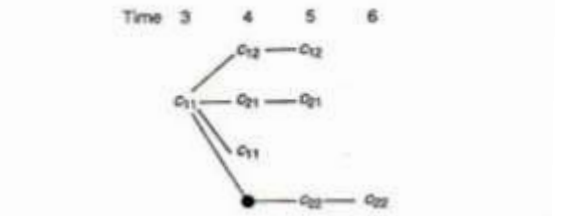
Proses ini berlanjut terus sampai keseluruhan nilai dihitung.

Pemetaan indeks awal menjadi indeks yang ditransformasikan diilustrasikan pada gambar 5.15 dan 5.16.

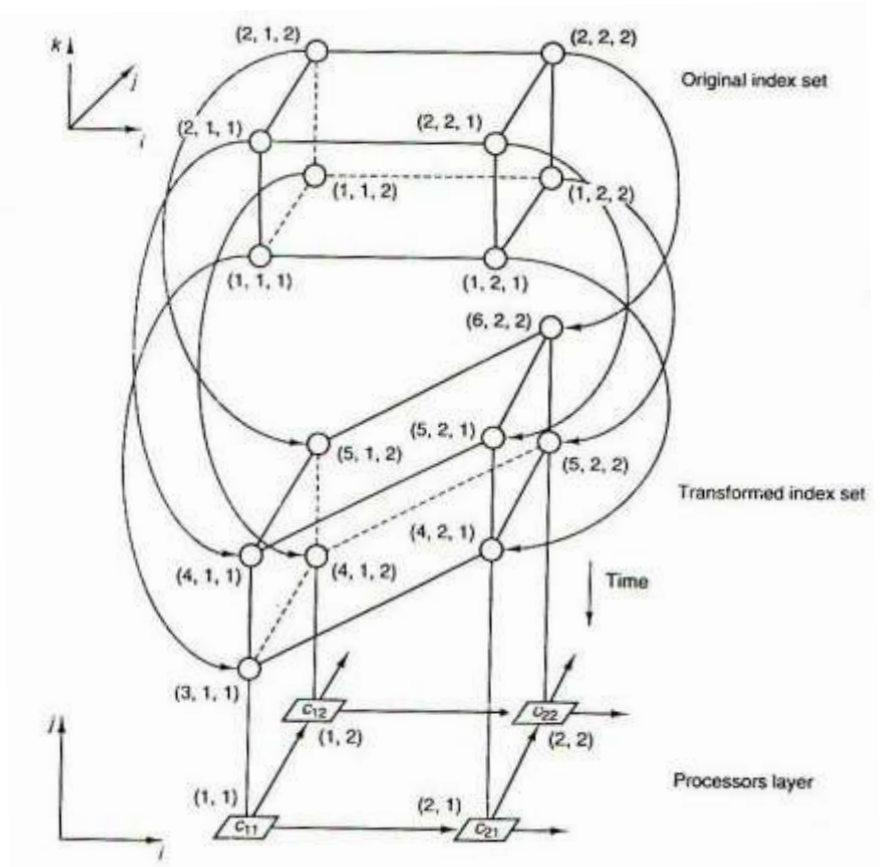


Gambar 5.14 Interkoneksi yang dibutuhkan antara PE.

Transfer matrix		Original index		Transformed index	C's	A's	B's	Time	PE element
T	•	$[1, 1, 1]^t$	=	$[3, 1, 1]^t$	c_{11}	a_{11}	b_{11}	3	(1, 1)
T	•	$[1, 1, 2]^t$	=	$[4, 1, 2]^t$	c_{12}	a_{11}	b_{12}	4	(1, 2)
T	•	$[1, 2, 1]^t$	=	$[4, 2, 1]^t$	c_{21}	a_{21}	b_{11}	4	(2, 1)
T	•	$[1, 2, 2]^t$	=	$[5, 2, 2]^t$	c_{22}	a_{21}	b_{12}	5	(2, 2)
T	•	$[2, 1, 1]^t$	=	$[4, 1, 1]^t$	c_{11}	a_{12}	b_{21}	4	(1, 1)
T	•	$[2, 1, 2]^t$	=	$[5, 1, 2]^t$	c_{12}	a_{12}	b_{22}	5	(1, 2)
T	•	$[2, 2, 1]^t$	=	$[5, 2, 1]^t$	c_{21}	a_{22}	b_{21}	5	(2, 1)
T	•	$[2, 2, 2]^t$	=	$[6, 2, 2]^t$	c_{22}	a_{22}	b_{22}	6	(2, 2)



Gambar 5.15 Pemetaan indeks awal menjadi indeks yang ditransformasikan



Gambar 5.16 Susunan indeks yang telah ditransformasi, dan dikomputasikan oleh PE.

DAFTAR PUSTAKA

Brey, Barry B. 2003. *The Intel Mikroprosesor 8086/8088, 80186/80188, 80286, 80386, 80486, Pentium, Pentium-Pro Prosessor, Pentium II, Pentium III, Pentium 4 : Architecture, Programming, and Interfacing (6th Edition)*, Prentice Hall.

Fadlisyah. 2005. *Pemrosesan Paralel : Seri Diktat Kuliah Universitas Malikussaleh*. -

Fadlisyah, dkk., 2008. *Robotika : Reasoning, Planning, Learning.*, Penerbit Graha Ilmu.

Mano, M. Morris. 1993. *Computer System Architecture*, Prentice-Hall International.

Stalling, William. 1998. *Organisasi dan Arsitektur Komputer : Perancangan Kinerja (jilid 1)*, Prentice Hall yang diterjemahkan oleh PT Prenhallindo, Jakarta.

Zargham, Mehdi R., 1996. *Computer Architecture : Single and Parallel Systems*, Prentice-Hall International.

Teknologi pemrosesan paralel telah menjadi minat dan perhatian yang tidak baru dalam dunia informatika, tetapi perkembangan penerapan aplikasinya sendiri baru dapat dirasakan beberapa tahun belakangan ini. Berbagai arsitektur yang menggunakan teknologi tersebut telah ditawarkan meluas dan mudah ditemukan di pasar komputer, dan sementara teori atau pondasi yang melatarbelakangi arsitektur ini sangat sedikit dan sulit kita temukan dengan mudah. Di berbagai Universitas, mata kuliah pemrosesan paralel menjadi bagian penting dan terintegrasi di dalam kurikulum program strudi Teknik Informatika atau Ilmu Komputer. Untuk itu, buku pemrosesan paralel yang pembaca pegang ini hadir menjadi referensi alternatif untuk mendukung penambahan khazanah di bidang komputer khususnya pemrosesan paralel. Adapun materi yang hadir di dalam buku ini meliputi : Klasifikasi Perancangan, Arsitektur Von Neumann, Pipeline, Arsitektur RISC vs CISC, dan Arsitektur Systolic Array & Arsitektur Data Flow.

Dahlan Abdullah, Menyelesaikan Pendidikan Strata Satu (S1) Program Studi Teknik Informatika Fakultas Teknologi Industri pada Universitas Islam Indonesia (UII) Yogyakarta dan mendapat Gelar Sarjana Teknik (ST) pada tahun 1999. Menyelesaikan Program Pasca Sarjana (S2) Magister Komputer di STMIK Eresha Jakarta pada tahun 2014 serta saat ini tahun 2015 sedang menjalani studi di Program Doktor (S3) Ilmu Komputer Universitas Sumatera Utara Medan, Aktif melakukan Penelitian dibidang Jaringan Komputer, Database, Radio Net, Komputer Aplikasi, Robotika, Web Based Application, Sistem Informasi Manajemen dan Infrastruktur Jaringan Komputer. Selain menjadi Dosen di Jurusan Teknik Informatika Fakultas Teknik Universitas Malikussaleh, pernah menduduki Jabatan sebagai Kepala UPT. Pusat Komputer Universitas Malikussaleh dari tahun 2005 sampai dengan 2011 dan Jabatan Saat ini sebagai Kepala UPT. Perpustakaan Universitas Malikussaleh Aceh - Indonesia. Aktif juga sebagai anggota dari INHERENT (Indonesian Higher Education Network), JARDIKNAS (National Education Network), dan Sekretaris Jenderal APTIKOM (Association of Computer and Information University) Provinsi Aceh, sebagai Koordinator ICTPura Provinsi Aceh dari Tahun 2010 - 2013, Koordinator Relawan TIK Provinsi Aceh, Koordinator E-KTP Provinsi Aceh, pernah menjabat sebagai Wakil Ketua Palang Merah Indonesia (PMI) Cabang Kota Lhokseumawe, pernah menjabat sebagai Ketua Radio Antar Penduduk Indonesia (RAPI) Wilayah Kota Lhokseumawe selama 2 periode dari tahun 2005 sampai dengan 2011, Kepala Bidang HUMAS Mania Golf Community (mGc) Kota Lhokseumawe, Anggota LMR-RI, Anggota Brata Airsoft Gun – PERBAKIN Kota Lhokseumawe, Koordinator atau Perwakilan Genta Foundation untuk SMK Provinsi Aceh tahun 2015, Wakil Sekretaris Bhayangkara Golf Club (BGC) Provinsi Aceh, Tim Ahli di POLRES Lhokseumawe dan POLRES Kabupaten Bireun dan beberapa Jabatan lainnya.

ISBN 602137345-6

9 786021 373453

Dahlan Abdullah, ST, M.Kom
Email : dahlan.unimal@gmail.com
Website : http://www.dahlan.web.id

UNIMAL PRESS