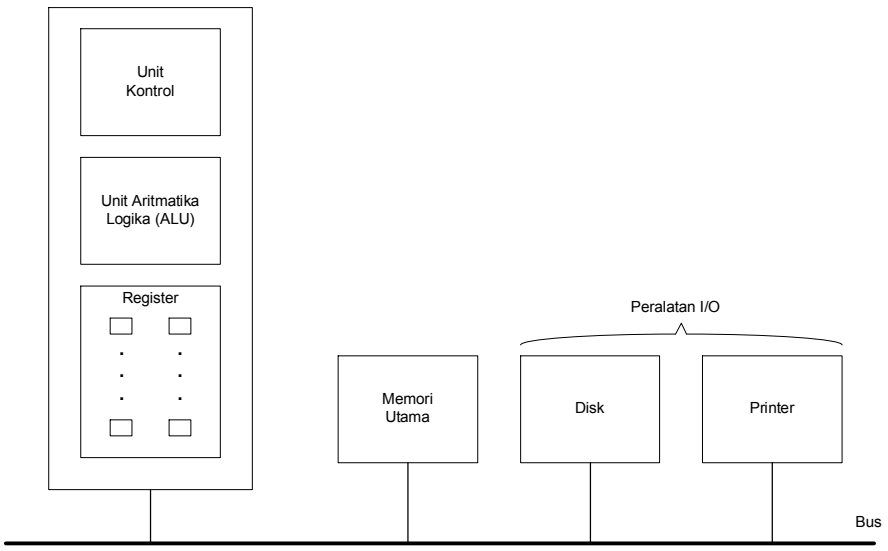
7. PROSESOR

Organisasi sebuah komputer sederhana yang berorientasi pada bus ditampilkan pada Gambar 7.1. *Central Processing Unit (CPU)* adalah "otak" dari sebuah komputer. Fungsi CPU adalah menjalankan program-program yang disimpan dalam memori utama dengan cara mengambil instruksi-instruksi, menguji instruksi tersebut, dan kemudian menjalankannya satu demi satu. Komponen-komponen itu dihubungkan oleh sebuah bus, yaitu sekumpulan kabel-kabel pararel untuk mentransmisikan alamat *(address),* data, dan sinyal-sinyal kontrol. Bus dapat berada di luar CPU, yang menghubungkan CPU dengan memori dan peralatan I/O (Input/Output), tapi juga ada di dalam CPU, seperti yang akan kita lihat sebentar lagi.

CPU terdiri dari beberapa bagian berbeda. Unit kontrol bertanggung jawab mengambil instruksi-instruksi dari memori utama dan menentukan jenis instruksi instruksi tersebut. Unit Logika Aritmetik (ALU) menjalankan operasi-operasi seperti penjumlahan dan Boolen AND.



Gambar 7.1. Organisasi komputer sederhana dengan CPU dan dua peralatan I/O

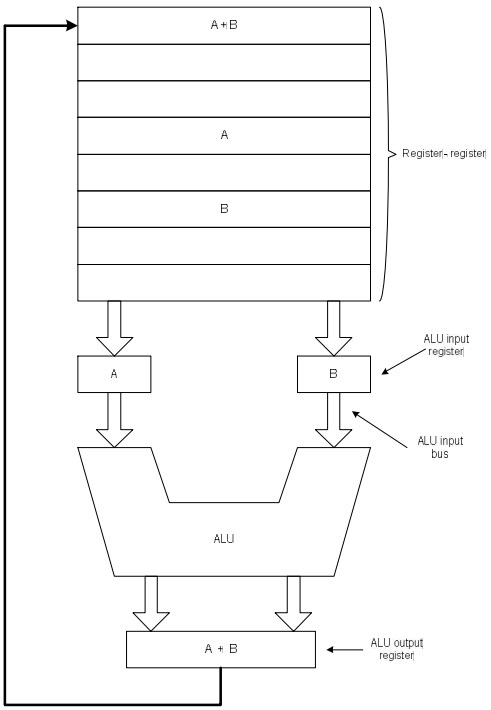
CPU juga berisi sebuah memori kecil berkecepatan tinggi yang digunakan untuk menyimpan hasil-hasil sementara clan informasi kontrol tertentu. Memori ini terdiri dari sejumlah register, yang masing-masing memiliki ukuran dan fungsi tersendiri. Biasanya, seluruh register itu memiliki ukuran yang sama. Setiap register dapat menyimpan satu bilangan, hingga mencapai jumlah maksimum tertentu tergantung pada ukuran register tersebut. Register-register

dapat dibaca dan ditulis dengan kecepatan tinggi karena mereka berada dalam CPU.

Register yang paling penting adalah ***Program Counter (PC)***, yang menunjuk ke instruksi berikutnya yang harus diambil untuk dijalankan. Nama "program counter" agak kurang tepat karena istilah ini tidak ada hubungannya sama sekali dengan *counting* (perhitungan), namun istilah ini telah digunakan secara luas. Selain itu, fungsi penting lainnya dari register adalah ***Instruction Register (IR)***, yang menyimpan instruksi yang sedang dijalankan. Sebagian besar komputer juga memiliki beberapa register lain, sebagian diantaranya digunakan untuk tujuan umum dan sebagiannya lagi untuk tujuan-tujuan khusus.

7.1. Organisasi CPU

Organisasi bagian internal dari sebuah CPU von Neumann khusus ditunjukkan secara lebih lengkap pada Gambar 7.2. Bagian ini disebut jalur data dan berisi register register (terutama 1 sampai 32), ALU (Arithmetic Logic Unit), dan beberapa bus yang menghubungkan bagian-bagian tersebut. Register-register tersebut melengkapi dua register untuk input ALU, yang dalam gambar tersebut diberi label A dan B. Register-register ini menyimpan input ALU sementara ALU menjalankan fungsi perhitungan.



Gambar 7.2. Jalur data untuk mesin von Neumann

ALU sendiri menjalankan operasi penambahan, pengurangan, dan operasi-operasi sederhana lainnya pada input-inputnya, dengan demikian memberikan suatu hasil pada register output. Register output ini dapat disimpan kembali ke dalam sebuah register. Kemudian, register tersebut dapat ditulis (yakni, disimpan) ke dalam memori, jika memang dikehendaki. Tidak semua rancangan memiliki register A, B, dan register-register lain. Dalam contoh pada gambar tersebut, operasi penambahan juga dijelaskan.

Sebagian besar instruksi dapat dibagi ke dalam dua kategori: register-memori atau register-register. Contoh, instruksi register-memori memungkinkan word dimunculkan dalam register-register, di mana word memori tersebut dapat digunakan sebagai input-input ALU dalam instruksi-instruksi berikutnya. (Word adalah satuan-satuan data yang bergerak antara memori dan register-register. Sebuah word dapat jadi adalah sebuah integer). Instruksi register-memori lainnya memungkinkan register-register disimpan kembali ke dalam memori.

Jenis instruksi lainnya adalah register-register. Sebuah instruksi register-register khusus mengambil dua operand dari register-register, membawa mereka ke register-register input ALU, menjalankan operasi tertentu atas kedua operand tersebut, misalnya, fungsi penambahan atau Boolean AND, dan menyimpan kembali hasil-hasilnya dalam salah satu dari register-register tersebut. Proses untuk menjalankan kedua operand tersebut melalui ALU dan menyimpan hasil-hasilnya disebut siklus jalur data dan merupakan inti dari sebagian besar CPU. Terutama, proses tersebut menentukan apa yang dapat dilakukan mesin. Semakin cepat siklus jalur data tersebut, semakin cepat pula mesin tersebut beroperasi.

7.2. Menjalankan Instruksi

CPU menjalankan setiap instruksi dalam beberapa langkah kecil. Jelasnya, langkah-langkah tersebut adalah sebagai berikut:

1. Mengambil instruksi berikutnya dari memori dan membawanya ke dalam register instruksi.
2. Mengubah PC (pencacah program) agar menunjuk ke instruksi selanjutnya.
3. Menentukan jenis instruksi yang baru saja diambil.

4. Jika instruksi tersebut menggunakan sebuah word dalam memori, ditentukan di mana instruksi tersebut berada.
5. Mengambil word tersebut, jika diperlukan, dan membawanya ke dalam sebuah register CPU.
6. Menjalankan instruksi.
7. Kembali ke langkah 1 untuk memulai menjalankan instruksi berikutnya.

Rangkaian langkah-langkah ini sering disebut sebagai siklus baca-decode execute. Langkah-langkah ini sangat penting bagi pengoperasian semua komputer.

Deskripsi ini tentang bagaimana sebuah CPU bekerja hampir menyerupai sebuah program yang ditulis dalam bahasa Inggris. Mesin yang diinterpretasi memiliki dua register yang kelihatan bagi program-program pemakai: *Program Counter (PC),* untuk mencari alamat instruksi berikutnya yang akan diambil, dan *akumulator (AC),* untuk mengumpulkan hasil-hasil aritmetik. Mesin tersebut juga memiliki register-register internal untuk menyimpan instruksi terkini selama pelaksanaan instruksi tersebut (instr), tipe instruksi terkini (instr type), alamat operand instruksi (data log), dan operand terkini (data). Instruksi-instruksi diasumsikan memiliki satu alamat memori saja. Di mana memori tersebut berisikan operand, contoh, item data untuk ditambahkan pada akumulator.

Kenyataan bahwa menulis sebuah program yang dapat meniru fungsi sebuah CPU dapat dilakukan menunjukkan bahwa sebuah program tidak perlu dijalankan oleh sebuah CPU "hardware" yang berisi sekotak penuh peralatan elekronik. Justru, sebuah program dapat dijalankan dengan meminta program lain mengambil, memeriksa, dan menjalankan instruksi-instruksinya. Sebuah program yang mengambil, memeriksa, dan menjalankan instruksi-instruksi dari program lain disebut interpreter.

Kesamaan fungsi ini antara prosesor hardware dan interpreter memiliki implikasi penting bagi organisasi komputer dan desain sistem komputer. Setelah menspesifikasikan bahasa mesin, L, untuk sebuah komputer baru, tim desain dapat memutuskan apakah mereka ingin membuat sebuah prosesor hardware untuk menjalankan program-program dalam L secara langsung atau apakah

mereka ingin menulis sebuah interpreter untuk menginterpretasikan program-program dalam L. Jika pilihannya adalah menulis sebuah interpreter, mereka juga harus menyediakan mesin hardware tertentu untuk menjalankan interpreter tersebut. Konstruksi-konstruksi tiruan tertentu juga memungkinkan, dengan menjalankan hardware tertentu dan juga interpretasi software tertentu.

Sebuah interpreter membagi instruksi-instruksi dari mesin targetnya ke dalam langkah-langkah kecil. Karena itu, mesin tempat interpreter tersebut beroperasi mungkin jauh lebih sederhana dan murah dibanding sebuah prosesor hardware untuk mesin target tersebut. Penghematan ini sangat penting artinya jika mesin target tersebut memiliki sejumlah besar instruksi dan instruksi-instruksi tersebut rumit dan memiliki banyak pilihan. Penghematan tersebut menjadi begitu penting bila: memperhatikan fakta bahwa hardware telah digantikan oleh software (interpreter).

Komputer-komputer zaman dahulu memiliki instruksi yang sederhana dan dalam jumlah yang kecil. Adanya permintaan untuk komputer-komputer yang lebih unggul telah menghasilkan instruksi-instruksi individu yang lebih unggul pula. Sejak awal, telah diketahui bahwa instruksi-instruksi yang lebih rumit sering membuat program-program dijalankan dengan lebih cepat meskipun masing-masing instruksi mungkin membutuhkan waktu lebih lama untuk dijalankan. Instruksi *Floating-point (Titik mengambang)* adalah sebuah contoh instruksi yang lebih kompleks. Contoh lain adalah: dukungan langsung untuk mengakses elemen larik. Kadang-kadang bila diamati dua instruksi yang sama sering muncul secara berurutan, sehingga satu instruksi saja dapat menjalankan pekerjaan kedua instruksi tersebut.

Instruksi-instruksi yang lebih kompleks masih dapat dijalankan dengan baik karena pelaksanaan operasi-operasi individu kadang-kadang dapat ditumpangtindihkan atau dapat juga dijalankan secara paralel dengan menggunakan hardware berbeda. Untuk komputer-komputer mahal berkinerja tinggi, biaya hardware tambahan ini tidak menjadi masalah. Jadi, komputer-komputer mahal dan berkinerja tinggi tampaknya memiliki lebih banyak instruksi daripada komputer-komputer murah. Adanya perkembangan software memerlukan persyaratan-persyaratan kompatibilitas instruksi dalam mengimplementasikan instruksi-instruksi kompleks. Tuntutan ini berlaku juga

untuk komputer-komputer murah pada satu sisi, di sisi lain adanya pertimbangan biaya yang cukup tinggi.

Menjelang akhir 1950-an, IBM (yang kini telah menjadi sebuah perusahaan komputer terkemuka) telah mengakui bahwa mengembangkan sekelompok mesin, yang semuanya menjalankan instruksi-instruksi yang sama, memiliki banyak keuntungan, baik bagi IBM maupun bagi para konsumennya. IBM memperkenalkan istilah arsitektur untuk menjelaskan persamaan tingkat kompatibilitas ini. Sejumlah komputer baru mungkin memiliki satu arsitektur tapi mempunyai banyak implementasi berbeda sehingga semua komputer tersebut dapat menjalankan program yang sama, dengan perbedaan mungkin hanya pada harga dan kecepatan. Tapi, bagaimana cara mengembangkan sebuah komputer murah yang dapat menjalankan semua instruksi kompleks dari mesin-mesin mahal yang berkecepatan tinggi.

Jawabannya terletak pada interpretasi. Teknik ini, yang pertama kali diperkenalkan oleh Wilkes (1951), memungkinkan desain komputer-komputer murah dan sederhana yang justru dapat menjalankan sejumlah besar instruksi. Hasilnya adalah bahwa arsitektur IBM System/362, dan komputer kompatible lainnya akan mampu bersaing baik dalam harga maupun kecepatan. Implementasi hardware langsung (yaitu, bukan yang diinterpretasikan) digunakan hanya pada tipe komputer yang sangat mahal.

Komputer-komputer sederhana dengan instruksi-instruksi yang diinterpretasikan juga memiliki beberapa keuntungan lain. Dan yang paling utama adalah:

1. Kemampuan untuk memperbaiki instruksi-instruksi yang dijalankan secara tidak tepat dalam bidang tersebut, atau membetulkan kesalahan-kesalahan desain dalam hardware utama.
2. Kesempatan untuk menambahkan instruksi-instruksi baru dengan biaya rendah, bahkan setelah penjualan komputer tersebut.
3. Desain terstruktur yang memungkinkan pengembangan, pengujian, clan pendokumentasian instruksi-instruksi kompleks secara efisien.

Pada saat pasar komputer begitu marak pada 1970-an dan kemampuan--kemampuan komputerisasi berkembang dengan pesat, permintaan untuk komputer-komputer murah menguntungkan desain komputer-komputer yang

menggunakan interpreter-interpreter. Kemampuan untuk menyesuaikan hardware dan interpreter untuk sejumlah instruksi tertentu berkembang seperti desain yang sangat hemat biaya untuk prosesor-prosesor. Ketika teknologi semikonduktor utama berkembang dengan pesat, keuntungan-keuntungan dari segi biaya mengungguli peluang untuk kinerja yang lebih tinggi, dan arsitektur-arsitektur yang berbasiskan interpreter menjadi cara konvensional untuk mendesain komputer-komputer. Hampir semua komputer baru yang didesain pada 1970-an, mulai dari minikomputer hingga mainframe, didasarkan pada interpretasi.

Menjelang akhir 1970-an, penggunaan prosesor-prosesor sederhana yang mengoperasikan interpreter-interpreter semakin meluas kecuali diantara model-model yang sangat mahal dengan kinerja yang sangat tinggi, seperti Cray-1 dan seri-seri Control Data Cyber. Penggunaan interpreter telah menghilangkan kendala keterbatasan biaya dalam pembuatan instruksi-instruksi yang kompleks, dan arsitektur-arsitektur mulai merambah instruksi-instruksi yang lebih komplek, terutama cara-cara untuk menspesifikasikan operand-operand yang akan digunakan.

Trend ini mencapai puncaknya dengan lahirnya komputer VAX buatan Digital Equipment Corporation, yang memiliki beberapa ratus instruksi, dan lebih dari 200 cara berbeda untuk menspesifikasikan operand-operand yang akan digunakan dalam setiap instruksi. Komputer VAX sejak awal dianggap akan dijalankan dengan menggunakan sebuah interpreter, tapi tidak didukung teknologi berkinerja tinggi. Hal ini menyebabkan dimasukkannya sejumlah instruksi yang sangat banyak yang kurang begitu bermanfaat yang pada akhirnya sulit untuk dijalankan secara langsung. Penghapusan ini terbukti sangat fatal bagi VAX, dan juga akhirnya bagi DEC (Compaq membeli DEC pada 1998).

Meskipun mikro-prosesor 8 bit generasi pertama adalah mesin-mesin yang sangat sederhana dengan perangkat-perangkat instruksi yang sangat sederhana, menjelang akhir 1970-an, mikroprosesor telah beralih ke desain-desain yang menggunakan interpreter. Selama periode ini, salah satu tantangan utama yang dihadapi para perancang mikroprosesor adalah mengatasi kompleksitas yang makin tinggi melalui sirkuit-sirkuit terpadu. Keuntungan utama dari pendekatan

yang berbasiskan interpreter adalah kemampuan untuk mendesain sebuah prosesor sederhana, dengan kompleksitas hanya pada memori yang menyimpan interpreter tersebut. Jadi desain hardware yang kompleks dapat dialihkan ke desain software yang kompleks.

Keberhasilan Motorola 68000, yang memiliki perangkat instruksi interpretasi yang besar, dan kegagalan Zilog Z8000 (yang juga memiliki perangkat instruksi yang sama besarnya, tapi tidak memiliki interpreter) menunjukkan keunggulan-keunggulan dari sebuah interpreter untuk memasarkan sebuah mikroprosesor baru dengan cepat. Keberhasilan ini sangat mengagumkan mengingat yang pertama kali diperkenalkan adalah Zilog (generasi Z8000 sebelumnya, yaitu Z80, jauh lebih populer daripada generasi pendahulu 68000, yaitu 6800). Tentu, faktor-faktor lain juga turut berperan di sini, paling tidak adalah sejarah panjang Motorola sebagai produsen pembuat chip dan sejarah panjang Exxon (pemilik Zilog) sebagai sebuah perusahaan minyak, bukan sebagai perusahaan pembuat chip.

Faktor lain yang juga menguntungkan perkembangan interpretasi selama era tersebut adalah keberadaan memori-memori cepat yang khusus untuk membaca *(fast, read-only memories),* yang disebut *Kontrol Store,* untuk menyimpan interpreter-interpreter. Misalkan bahwa sebuah instruksi 68000 yang diinterpretasikan secara khusus membutuhkan interpreter 10 instruksi, yang disebut mikroinstruksi, dengan masing-masing 100 nsec, dan dua referensi ke memori utama, dengan masing-masing 500 nsec. Jadi jumlah waktu total untuk menjalankan instruksi-instruksi adalah 2000 nsec, hanya merupakan salah satu faktor dari dua faktor terburuk dibanding yang terbaik yang dapat dicapai bila instruksi-instruksi dijalankan secara langsung. Jika *kontrol store* belum tersedia, instruksi tersebut akan membutuhkan waktu 6000 nsec. Suatu faktor dengan enam pinalti sedikit lebih sulit untuk digunakan daripada satu faktor dengan dua pinalti.

7.3. RISC versus CISC

Selama akhir 1970-an, karena keberadaan interpreter, telah dilakukan banyak eksperimen dengan instruksi-instruksi yang sangat kompleks. Para perancang mencoba untuk menutupi "perbedaan semantik" antara mesin-mesin

apa yang dapat menjalankan instruksi-instruksi dan bahasa-bahasa pemrograman tingkat tinggi apa yang dibutuhkan. Hampir tidak seorangpun berpikir tentang upaya untuk mendesain mesin-mesin yang lebih sederhana, sama seperti sekarang tidak banyak penelitian yang dilakukan untuk mendesain sistem-sistem pengoperasian, jaringan-jaringan, dan prosesor-prosesor word, dan lain-lain yang kurang berkinerja tinggi.

Satu kelompok yang menentang trend-tersebut dan mencoba memadukan sebagian dari ide-ide Seymour Cray dalam sebuah minikomputer berkinerja tinggi, dipimpin oleh John Cocke di IBM. Upaya ini menghantar ke upaya pengembangan sebuah minikomputer eksperimental, yang dinamakan 801. Meskipun IBM tidak pernah memasarkan mesin ini dan hasil-hasilnya tidak dipublikasikan hingga beberapa tahun kemudian, komentar-komentar bermunculan dan orang lain mulai meneliti arsitektur-arsitektur yang sama.

Pada tahun 1980, suatu kelompok di Barkeley yang dipimpin oleh David Patterson da Carlo Sequin mulai merancang keping VLSI CPU yang tidak menggunakan interpretasi (Patterson, 1985; Patterson dan Sequin, 1982). Mereka menggunakan istilah RISC untuk konsep ini dan menamakan chip CPU mereka RISC 1 yang langsung disusul oleh RISC II. Setahun kemudian, pada 1981, di Standford John Hennessy mendesain dan membuat sebuah chip yang agak berbeda yang dinamakannya MIPS (Hennessy, 1984). Chip-chip ini berkembang menjadi dua produk penting dari segi komersial, yakni SPARC dan MIPS.

Prosesor-prosesor baru ini sangat berbeda dibanding prosesor-prosesor komersial dewasa ini. Karena CPU-CPU baru ini tidak harus disesuaikan kembali dengan produk-produk yang telah ada, para perancang mereka bebas untuk memilih perangkat-perangkat instruksi baru yang akan memaksimalkan kinerja sistem seluruhnya. Pada awalnya perhatian utama diberikan pada seberapa cepat suatu instruksi menyelesaikan proses implementasi. Tapi kemudian disadari bahwa mendesain instruksi-instruksi dengan kinerja yang baik harus didasarkan pada seberapa cepat suatu instruksi dapat dimulai. Berapa lama waktu yang diperlukan instruksi kurang menjadi masalah dibanding berapa banyak instruksi yang dapat dimulai per detik.

Pada saat prosesor-prosesor sederhana ini pertama kali dirancang, karakteristik-karakteristik yang menarik perhatian banyak kalangan adalah jumlah instruksi yang tersedia relatif sedikit, kira-kira sekitar 50. Jumlah ini jauh lebih kecil dibanding jumlah 200 hingga 300 pada komputer-komputer tertentu seperti DEC VAX dan mainframe-mainframe IBM yang berukuran besar. Sebenarnya, kepanjangan dari RISC adalah Reduced Instruction Set Computer, yang dilawankan dengan CISC, yang merupakan kepanjangan dari Complex Instruction Set Computer (sebuah sebutan tidak langsung untuk VAX, yang mendominasi Jurusan Ilmu Komputer universitas pada saat itu). Kini, segelintir orang berpikiran bahwa ukuran perangkat instruksi merupakan suatu isu utama, meskipun namanya tetap tidak berubah.

Singkatnya, perang pendapat pun timbul, dengan para pendukung RISC menyerang tatanan yang telah mapan (VAX, Intel, dan mainframe-mainframe IBM yang berukuran besar). Mereka mengklaim bahwa cara paling tepat untuk mendesain sebuah komputer adalah menyediakan sejumlah kecil instruksi sederhana yang beroperasi dalam satu siklus jalur data seperti pada Gambar 7.2, yakni, mengambil dua register, menggabungkan kedua register tersebut (misalnya, dengan menambahkan atau mem-Boolean AND keduanya), dan menyimpan kembali hasilnya dalam sebuah register. Pendapat para pendukung RISC ini adalah bahwa meskipun sebuah mesin RISC membutuhkan empat atau lima instruksi untuk menjalankan apa yang dilakukan sebuah mesin CISC hanya dengan satu instruksi saja. Jika instruksi-instruksi RISC 10 kali lebih cepat (karena instruksi-instruksi tersebut tidak diinterpretasikan), maka RISC jauh lebih unggul. Penting juga untuk diperhatikan bahwa dewasa ini kecepatan memori-memori utama telah melampaui kecepatan kontrol store yang dikhususkan hanya untuk membaca, jadi pinalti interpretasi telah meningkat dengan cepat, sehingga sangat menguntungkan mesin-mesin RISC.

Seseorang mungkin beranggapan bahwa dengan mempertimbangkan keunggulan-keunggulan kinerja dari teknologi RISC, mesin-mesin RISC (seperti DEC Alpha) akan mengungguli mesin-mesin CISC (seperti Intel Pentium) di pasaran. Hal ini belum pernah terjadi. Mengapa tidak?

Pertama, terdapat isu mengenai kompatibilitas mundur dan dana software miliaran dolar yang telah diinvestasikan perusahaan-perusahaan untuk jalur

produksi Intel. Kedua, yang mengejutkan, Intel telah mampu memanfaatkan ide-ide yang sama bahkan dalam suatu arsitektur CISC. Dimulai dengan 486, CPU-CPU Intel berisi sebuah inti RISC yang menjalankan instruksi-instruksi paling sederhana (dan sangat umum) dalam satu siklus jalur data saja, sekaligus menginterpretasikan instruksi-instruksi yang lebih kompleks dengan cara CISC biasa. Hasil nyatanya adalah bahwa instruksi-instruksi biasa menjadi cepat dan instruksi-instruksi yang kurang biasa menjadi lamban. Meskipun pendekatan tiruan ini tidak secepat seperti sebuah desain RISC murni, pendekatan ini menghasilkan kinerja keseluruhan yang kompetitif asalkan software yang lama tetap dapat diaplikasikan dan tidak berubah.

7.4. Prinsip-prinsip Desain untuk Komputer-komputer Modern

Kini setelah lebih dari satu dekade sejak mesin-mesin RISC pertama diperkenalkan, prinsip-prinsip desain tertentu telah diakui sebagai cara yang tepat untuk mendesain komputer-komputer sesuai dengan situasi teknologi hardware dewasa ini. Tapi dimungkinkan juga munculnya prinsip-prinsip desain baru yang berbeda dengan prinsip desain yang telah diakui tersebut. Jadi para perancang komputer harus selalu mencermati kemungkinan adanya perubahan-perubahan teknologi yang dapat mempengaruhi keseimbangan antara komponen-komponen.

Dikatakan bahwa, terdapat sejumlah prinsip desain, yang kadang-kadang disebut Prinsip-prinsip Desain RISC, yang harus benar-benar dipatuhi oleh perancang CPU yang memiliki tujuan umum. Keterbatasan-keterbatasan eksternal, seperti ketentuan mengenai kompatibilitas dengan arsitektur tertentu yang sudah ada, sering menuntut adanya penyesuaian dari waktu ke waktu. Dan para perancang berusaha memenuhi tuntutan tersebut. Beberapa prinsip utama akan diuraikan berikut ini.

7.4.1. Semua Instruksi Secara Langsung Dijalankan oleh Hardware

Supaya mempunyai kecepatan tinggi maka semua instruksi umum dijalankan secara langsung oleh hardware. Instruksi-instruksi tersebut tidak diinterpretasikan oleh mikroinstruksi-mikroinstruksi. Menghilangkan suatu level interpretasi akan menghasilkan kecepatan tinggi bagi sebagian besar instruksi.

Untuk komputer-komputer yang menjalankan perangkat-perangkat instruksi CISC, instruksi-instruksi yang lebih kompleks sebaiknya dibagi menjadi bagian-bagian yang terpisah, yang kemudian dapat dijalankan sebagai suatu rangkaian mikroinstruksi-mikroinstruksi. Langkah tambahan ini akan memperlambat mesin, tapi untuk instruksi-instruksi yang jarang muncul langkah ini mungkin dapat dilakukan.

7.4.2. Memaksimalkan Kecepatan di mana Instruksi-instruksi Dikeluarkan

Komputer-komputer modern menggunakan banyak trik untuk meningkatkan kinerjanya, terutama dengan mencoba mengeksekusikan sebanyak mungkin instruksi per detik. Bagaimanapun juga, jika Anda dapat mengeksekusikan 500 juta instruksi/ detik, Anda telah menciptakan sebuah prosesor 500-MIPS, terlepas dari lamanya waktu untuk menyelesaikan instruksi-instruksi tersebut. (MIPS adalah singkatan dari *Millions of Instructions Per Second;* prosesor MIPS diberi nama demikian dengan maksud menunjukkan kepada ikhtisar ini.) Prinsip ini menunjukkan bahwa paralelisme dapat memainkan peranan utama dalam meningkatkan kinerja, karena mengeksekusikan begitu banyak instruksi dalam waktu singkat (walaupun berjalannya proses setiap instruksi lambat). Hanya mungkin terjadi jika berbagai macam instruksi dapat dijalankan sekaligus.

Meskipun instruksi-instruksi selalu ditemui dalam perintah progam, instruksi-instruksi tersebut tidak selalu dieksekusikan dalam perintah program (karena sebagian sumber daya yang dibutuhkan mungkin sibuk) dan instruksi-instruksi tersebut tidak harus selesai dalam perintah program. Tentu, jika instruksi 1 menetapkan sebuah register dan instruksi 2 menggunakan register tersebut, supaya yakin harus diperhatikan secermat mungkin bahwa instruksi 2 tidak membaca register tersebut sebelum register itu mengandung nilai yang benar. Untuk melakukan hal ini dengan benar diperlukan banyak panduan namun dengan menjalankan banyak instruksi sekaligus berpeluang meningkatkan kinerja.

7.4.3. Instruksi-instruksi Harus Mudah untuk Didekodekan

Batas kritis pada tingkat kecepatan mengeluarkan instruksi-instruksi adalah dengan mendekodekan masing-masing instruksi untuk mengetahui sumber daya-sumber daya apa yang mereka butuhkan. Apa saja yang dapat membantu proses ini akan sangat berguna. Termasuk di dalamnya adalah membuat instruksi-instruksi secara teratur, dengan panjang yang tetap, dengan sejumlah kecil format. Semakin sedikit format untuk instruksi-instruksi, akan semakin baik.

7.4.4. Hanya Instruksi-instruksi *Load* and Store yang Diakses ke Memory

Salah satu cara paling sederhana untuk membagi operasi-operasi ke dalam langkahlangkah terpisah adalah menetapkan supaya operand-operand untuk sebagian besar instruksi harus berawal dari-dan kembali ke-register-register. Operasi untuk memindahkan operand-operand dari memori ke dalam register-register dapat dijalankan dalam instruksi-instruksi yang terpisah. Karena akses ke memori membutuhkan waktu yang lama, dan penundaan sulit diprediksi, instruksi-instruksi ini sebaiknya ditumpang tindihkan dengan instruksi-instruksi lain jika instruksi-instruksi tersebut tidak melakukan apa-apa selain memindahkan operand-operand antara register-register dan memori. Observasi ini berarti bahwa hanya instruksi-instruksi LOAD dan STORE yang harus diakses ke memori.

7.4.5. Menyiapkan Banyak Register

Karena akses ke memori agak lambat, maka perlu disiapkan banyak register (paling tidak 32 register). Karena itu setelah sebuah instruksi diambil, instruksi tersebut tetap tersimpan dalam sebuah register hingga instruksi itu tidak diperlukan lagi. Menjalankan instruksi di luar register dan memindahkan kembali instruksi tersebut ke memori, kemudian hanya mereload instruksi-instruksi itu; bukanlah suatu cara yang tepat dan sebaiknya tidak boleh dilakukan. Cara terbaik untuk melakukan ini adalah menyediakan cukup register.

7.5. Paralelisme instruksi-level

Para perancang komputer selalu berusaha untuk meningkatkan kinerja mesin-mesin yang dirancangnya. Salah satu cara yang dilakukan adalah membuat chip-chip agar bekerja lebih cepat dengan menambah kecepatan detaknya, namun

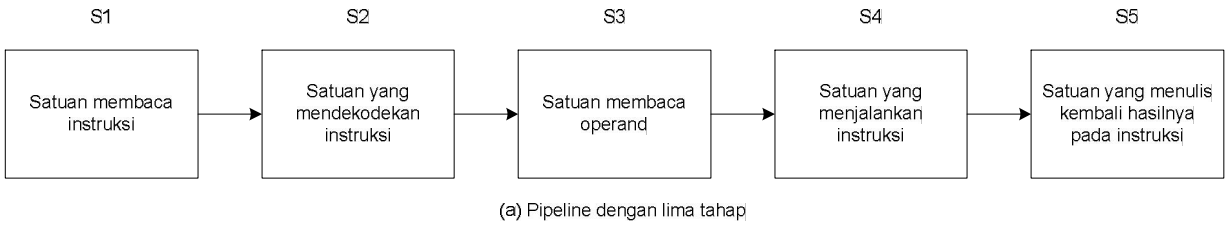
untuk semua desain baru, cara tersebut masih sulit untuk dilakukan. Oleh karena itu, sebagian besar arsitek komputer beralih ke paralelisme (melakukan dua hal atau lebih secara sekaligus) sebagai suatu cara untuk mencapai kinerja yang lebih tinggi pada suatu kecepatan detak tertentu.

Ada dua bentuk umum paralelisme: paralelisme instruksi-level dan paralelisme prosessor-level. Pada bentuk pertama, paralelisme dimanfaatkan dalam instruksi-instruksi individu agar dapat mengeksekusikan lebih banyak instruksi/detik mesin tersebut. Pada bentuk kedua, berbagai macam CPU bersama-sama menangani masalah yang sama. Setiap pendekatan memiliki keunggulannya masing-masing. Dalam bagian ini kita akan memberi perhatian khusus pada paralelisme instruksi level; pada bagian selanjutnya perhatian kita dialihkan ke paralelisme prosesor-level.

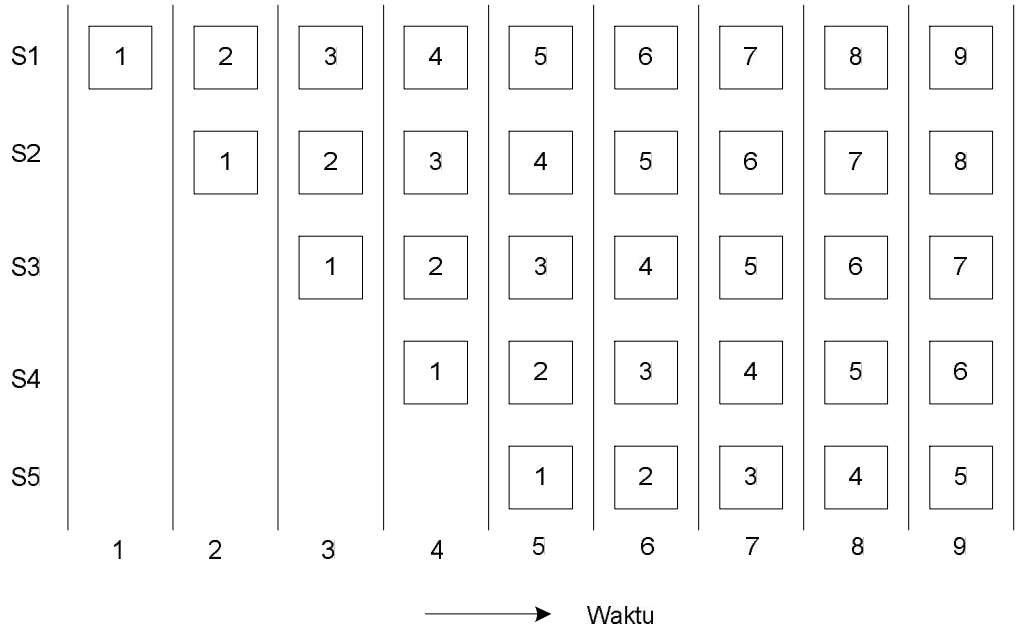
7.6. Pipelining

Telah lama diketahui bahwa membaca instruksi dari memori merupakan hambatan utama dalam hal kecepatan untuk menjalankan suatu instruksi. Untuk mengatasi masalah ini, komputer-komputer generasi IBM Stretch (1959) telah memiliki kemampuan untuk mengambil terlebih dahulu instruksi-instruksi dari memori sehingga instruksi-instruksi tersebut akan selalu siap ketika mereka dibutuhkan. Instruksi-instruksi ini disimpan dalam sekumpulan register yang disebut **penyangga prabaca**. Dengan cara ini, ketika sebuah instruksi dibutuhkan, instruksi tersebut biasanya dapat segera diambil dari penyangga prabaca daripada menunggu sebuah memori membaca hingga selesai.

Oleh karena itu, sistem prabaca membagi pelaksanaan instruksi menjadi bagian: membaca dan pelaksanaan aktual. Konsep **pipeline** menjelaskan strategi lebih jauh. Pelaksanaan instruksi sering dibagi ke dalam banyak bagian dan bukan hanya ke dalam dua bagian saja, di mana masing-masing bagian ditangani oleh seperangkat hardware khusus, dan keseluruhan bagian tersebut dapat beroperasi secara paralel.



(a) Pipeline dengan lima tahap



(b) Situasi masing-masing tahap sebagai suatu fungsi waktu. Sembilan siklus waktu diilustrasikan

Gambar 7.3.

Gambar 7.3(a) mengilustrasikan sebuah pipeline dengan lima unit, atau lima stage (5 tahap). Tahap 1 mengambil instruksi dari memoridan menempatkan instruksi tersebut dalam sebuah penyangga sampai instruksi itu dibutuhkan. Tahap 2 mendekodekan instruksi tersebut, menentukan jenisnya dan operand apa yang dibutuhkan instruksi tersebut. Tahap 3 melokasi dan mengambil operand-operand, baik itu dari register-register ataupun dari memori. Tahap 4 sebenarnya melaksanakan pekerjaan menjalankan instruksi tersebut, terutama dengan menjalankan operand-operand melalui jalur data pada Gambar 7.2. Terakhir, tahap 5 menulis hasilnya kembali ke register yang sesuai.

Dalam Gambar 7.3.(b) kita melihat bagaimana pipeline tersebut beroperasi sebagai suatu fungsi waktu. Selama siklus jam (waktu) 1, Sl sedang menangani instruksi 1, dengan mengambilnya dari memori. Selama siklus 2, tahap S2 mendekodekan instruksi 1, sedangkan tahap Sl mengambil instruksi 2. Selama

siklus 3, tahap S3 mengambil operand-operand dari instxuksi 1, tahap S2 mendekodekan instruksi 2, dan tahap Sl mengambil instruksi ketiga. Selama siklus 4, tahap S4 menjalankan instruksi 1, S3 mengambil operand-operand untuk instruksi 2, S2 mendekodekan instruksi 3, dan Sl mengambil instruksi 4. Terakhir, selama siklus 5, S5 menulis kembali hasil instruksi 1, sementara tahap-tahap lainnya menangani instruksi-instruksi berikutnya.

Mari kita lihat sebuah contoh untuk lebih memperjelas konsep pipeline. Bayangkan sebuah pabrik kue di mana proses pembakaran dan pengemasan kue-kue untuk pengiriman dilakukan secara terpisah. Misalkan bahwa departemen pengiriman memiliki sebuah ban berjalan pembawa panjang dengan lima pekerja (satuan-satuan pemrosesan) yang berdiri berjejer sepanjang ban berjalan tersebut. Setiap 10 detik (siklus jam), pekerja 1 menempatkan sebuah kotak kue kosong pada ban tersebut. Kotak tersebut dibawa ke pekerja 2, yang memasukkan sebuah kue ke dalam kotak itu. Sesaat kemudian, kotak tersebut sampai di pos kerja pekerja 3, yang kemudian menutup dan menyegel kotak tersebut. Selanjutnya kotak tersebut diteruskan ke pekerja 4, yang memasang sebuah label pada kotak kue itu. Terakhir, pekerja 5 memindahkan kotak tersebut dari ban dan memasukkannya dalam sebuah kontainer besar untuk kemudian dikirim ke sebtiah supermarket. Pada dasarnya, cara kerja seperti ini juga berlaku pada pipelining komputer: setiap instruksi (kue) melalui beberapa langkah pemrosesan sebelum mencapai hasil sempurna pada akhir proses.

Kembali ke pipeline pada Gambar 7.3., misalkan bahwa masing-masing tahapan siklus waktu mesin ini adalah 2 nsec. Maka sebuah instruksi membutuhkan siklus waktu 10 nsec untuk menempuh lima tahap pipeline. Sepintas, dengan dibutuhkannya waktu 10 nsec untuk sebuah instruksi, kelihatan bahwa mesin tersebut dapat menjalankan 100 MIPS. Namun sebenarnya mesin tersebut dapat menjalankan instruksi yang lebih besar dari jumlah ini. Pada setiap tahap siklus waktu (2 nsec), satu instruksi baru diselesaikan, sehingga jumlah pemrosesan instruksi yang sebenarnya adalah 500 MIPS, bukan 100 MIPS.

Pipelining memungkinkan terjadinya perimbangan antara latensi (berapa lama waktu yang dibutuhkan untuk menjalankan sebuah instruksi), dan lebar pita processor (berapa banyak MIPS yang dimiliki CPU). Dengan siklus waktu T

nsec, dan tahap-tahap n dalam pipeline, maka latensinya adalah nT nsec dan lebar pitanya adalah 1000/T MIPS (logikanya, karena kita sedang mengukur jumlah waktu dalam nano-detik, maka seharusnya kita mengukur lebar pita CPU dalam BIPS atau GIPS, tapi hal ini tidak dilakukan, jadi kita tidak memilih salah satu dari keduanya).

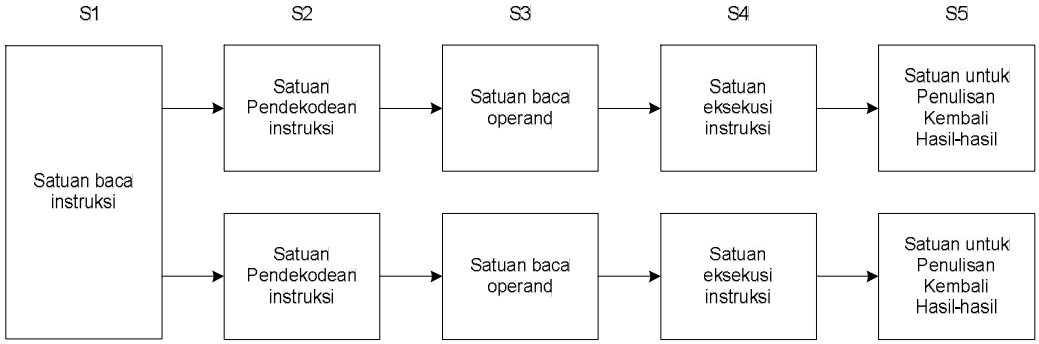
7.6.1. Arsitektur Superskalar

Jika satu pipeline bagus, maka tentu dua pipeline akan lebih bagus. Satu desain yang mungkin bagi sebuah CPU dengan pipeline ganda, didasarkan pada Gambar 7.3 ditunjukkan pada Gambar 7.4. Di sini suatu satuan membaca instruksi tunggal mengambil pasangan-pasangan dari instruksi-instruksi secara bersama dan memasukkan masing-masing pasangan ke dalam pipelinenya sendiri, lengkap dengan ALU-nya sendiri bagi operasi paralel. Agar dapat beroperasi secara paralel, kedua instruksi tersebut tidak boleh berebutan dalam menggunakan sumber daya (contoh, register-register), dan salah satu instruksi tidak boleh bergantung pada hasil dari instruksi yang lain. Seperti halnya dengan sebuah pipeline tunggal, begitu pul kompiler harus menjamin situasi ini tetap terjaga (yaitu, hardware tidak memeriksa dan memberikan hasil-hasil yang salah jika instruksi-instruksi tidak sebanding), atau konflik-konflik dideteksi dan dihilangkan selama pelaksanaan dengan menggunakan hardware tambahan.

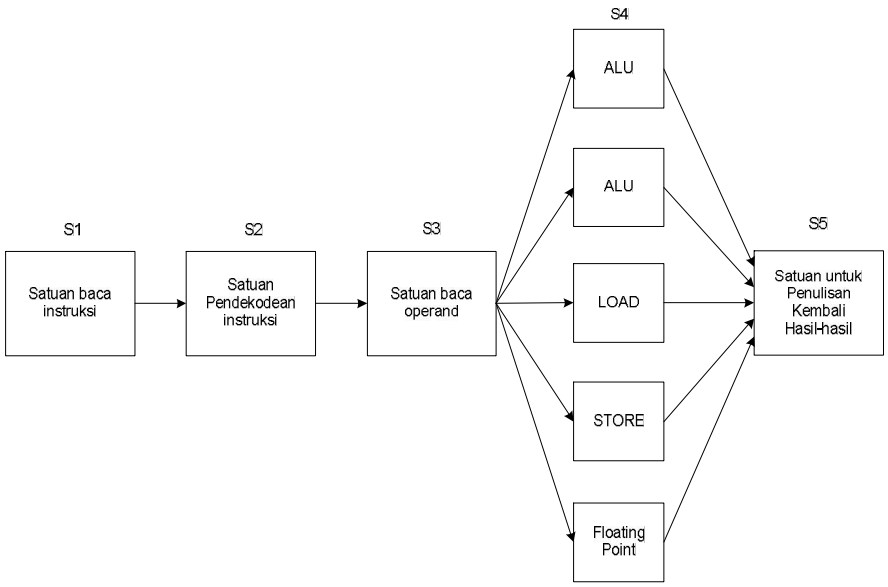
Meskipun pipeline-pipeline, tunggal atau ganda, sebagian besar digunakan pada mesin-mesin RISC (komputer 386 dan generasi-generasi pendahulunya tidak memiliki satupun), Intel 486 adalah yang pertama kali mulai memperkenalkan pipeline-pipeline ke dalam CPU-CPU-nya. Intel 486 memiliki satu pipeline dan Pentium memiliki dua *pipeline* lima tahap seperti pada Gambar 7.4., meskipun pembagian tugas sebenarnya antara tahap 2 dan tahap 3 (yang disebut decode-1 dan decode-2) sedikit berbeda dibanding dalam contoh kita. Pipeline utama, yang disebut pipeline u, dapat menjalankan sebuah instruksi Pentium yang selalu berubah-ubah. Pipeline kedua, yang disebut pipeline v, dapat menjalankan hanya instruksi-instruksi integer sederhana (dan juga satu instruksi titik mengambang sederhana-FXCH).

Peraturan-peraturan yang rumit menentukan apakah sepasang instruksi sebanding sehingga mereka dapat dijalankan secara paralel. Jika instruksi-

instruksi yang berpasangan tidak cukup sederhana atau tidak sebanding, hanya pasangan pertama yang dijalankan (dalam pipeline u). Pasangan kedua kemudian disimpan dan dipasangkan dengan instruksi berikutnya. Instruksi-instruksi selalu dijalankan secara berurutan. Jadi kompiler-kompiler khusus Pentium yang memproduksi pasangan-pasangan instruksi yang sebanding dapat memproduksi program-program yang beroperasi lebih cepat dibanding kampiler-kompiler lama. Pengukuran-pengukuran menunjukkan bahwa sebuah Pentium yang mengoperasikan kode yang dioptimalkan untuk Pentium tersebut memiliki kinerja dua kali lebih cepat dibandingkan dengan program-program integer seperti sebuah komputer 486 yang beroperasi pada laju kecepatan detak yang sama (Pountain, 1993). Hasil ini dapat dikaitkan seluruhnya dengan pipeline kedua.



Gambar 7.4. Pipeline lima tahap ganda dengan satuan membaca instruksi biasa



Gambar 7.5. Prosessor superskalar dengan lima satuan fungsional

Beralih ke empat pipeline dapat dilakukan, namun bila hal ini dilakukan akan menduplikat terlalu banyak hardware. Bahkan, suatu pendekatan berbeda digunakan pada *high-end CPU*. Ide dasarnya adalah untuk memiliki hanya satu pipeline tunggal namun pipeline tersebut memiliki berbagai macam satuan fungsi, seperti ditunjukkan pada Gambar 7.5. Contoh, Pentium III memiliki suatu struktur yang mirip dengan gambar. Istilah arsitektur superskalar ditetapkan bagi pendekatan ini pada 1987 (Agerwala dan Cocke, 1987). Namun sebenarnya pendekatan ini telah digunakan pada komputer CDC 6600 30 tahun sebelumnya. Komputer 6600 ini mengambil sebuah instruksi setiap 100 nsec dan membawa instruksi tersebut ke salah satu dari 10 satuan fungsional untuk dijalankan secara paialel sementara CPU beroperasi untuk untuk mendapatkan instruksi baru.

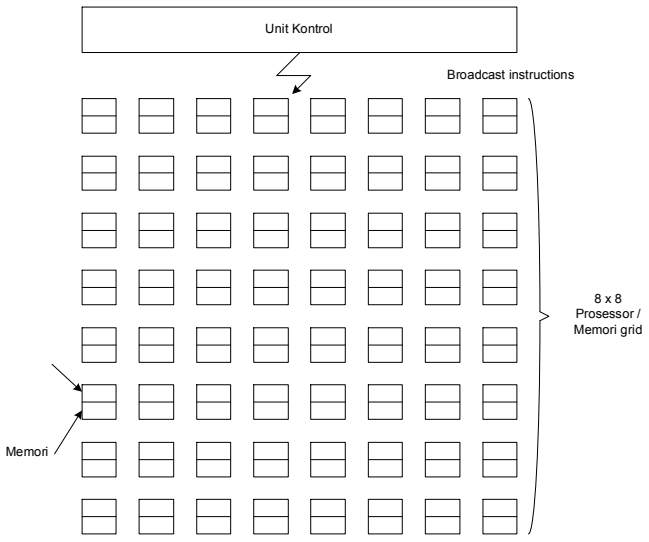
Yang tersirat dalam ide mengenai prosesor superskalar adalah bahwa tahap S3 dapat mengeluarkan instruksi-instruksi lebih cepat daripada tahap S4 dalam menjalankan instruksi-instruksi tersebut. Jika tahap S3 mengeluarkan sebuah instruksi setiap 10 nsec dan seluruh satuan fungsional dapat melaksanakan tugas mereka dalam 10 nsec, maka tidak lebih dari satu satuan yang akan benar-benar sibuk, terlepas dari ide keseluruhan. Dalam kenyataanya, sebagian besar satuan Fungsional dalam tahap S4 membutuhkan kira-kira lebih dari satu siklus detak untuk menjalankan instruksi-instruksi, dan tentu saja satuan-satuan tersebut adalah satuan-satuan yang dapat mengakses memori atau mengoperasikan aritmetik titik mengambang. Seperti dapat dilihat dari gambar tersebut, ada kemungkinan untuk memiliki berbagai macam ALU pada tahap S4.

Permintaan untuk komputer-komputer dengan kecepatan yang lebih tinggi tampaknya sulit dipenuhi. Para astronom ingin mensimulasi apa yang terjadi pada mikro-detik pertama setelah terjadi *big bang* (dentuman besar), para ahli ekonomi ingin memodelkan perekonomian dunia, dan para remaja ingin memainkan game game multimedia interaktif 3D melalui internet dengan teman-teman virtualnya. Meskipun CPU-CPU semakin cepat, pada akhirnya mereka akan menemui masalah berkaitan dengan kecepatan cahaya, yang mungkin tetap pada 20 cm/nanodetik dalam kabel tembaga atau serat optik, terlepas dari seberapa pintarnya para insinyur Intel. Demikian pula halnya dengan chip-chip yang berkecepatan tinggi, akan menghasilkan lebih banyak panas; yang penyebaran panas itu sendiri justru merupakan suatu masalah.

Paralelisme instruksi-level sedikit membantu, tapi pipeline dan operasi superskala jarang memperoleh hasil lebih dari suatu faktor lima atau sepuluh. Untuk memperoleh hasil 50, 100, atau lebih, satu-satunya cara adalah mendesain komputer dengan berbagai macam CPU, untuk itu sekarang kita akan melihat bagaimana sebagian dari CPU-CPU ini diorganisasikan.

7.6.2. Komputer-komputer Larik

Banyak masalah dalam ilmu fisika dan teknik yang melibatkan larik-larik atau sebaliknya memiliki struktur yang sangat teratur. Sering kali kalkulasi yang sama dilakukan pada berbagai kumpulan data yang berbeda dilakukan pada saat yang bersamaan. Regularitas dan struktur dari program-program ini menjadikan program program ini sebagai target-target yang sangat mudah untuk dijalankan secara paralel dengan kecepatan yang tinggi. Ada dua metode yang telah digunakan untuk menjalankan program-program ilmiah besar dengan cepat. Meskipun kedua metode ini sangat mirip dalam banyak hal, ironisnya, salah satu dari keduanya dianggap sebagai perluasan dari suatu prosesor tunggal, sedangkan yang lain dianggap sebagai sebuah komputer paralel.



Gambar 7.6. Sebuah prosessor larik tipe ILLIAC IV

Sebuah prosesor larik terdiri dari sejumlah besar prosesor yang sama, yang menjalankan rangkaian instruksi-instruksi yang sama pada kumpulan-kumpulan data berbeda. Prosesor larik pertama di dunia adalah komputer ILLIAC 1V di Universitas Illinois, yang diilustrasikan pada Gambar 7.6. (Bouknight dkk.,

1972). Rencana awalnya adalah membuat sebuah mesin yang terdiri dari empat kuadran, dan masing masing kuadran memiliki titik elemen-elemen prosesor/memori Sx8 kuadrat. Suatu satuan kontrol tunggal per kuadran menyebarkan instruksi-instruksi yang dijalankan oleh seluruh prosesor, dan setiap prosesor menggunakan datanya masing-masing dari memorinya sendiri (yang diload selama fase awal). Karena besarnya biaya untuk membuat empat kuadran tersebut, maka hanya satu kuadran yang dibuat, tetapi satu kuadran ini memiliki performans sebesar 50 megaflop (jutaan operasi titik mengambang per detik). Dikatakan bahwa jika mesin tersebut seluruhnya telah selesai dibuat dan jika mesin itu telah mencapai sasaran performans aslinya (1 gigaflop), mesin bersangkutan akan mampu melipatgandakan daya sistem komputer di seluruh dunia.

Bagi programer sebuah prosesor vektor sangat mirip dengan sebuah prosesor larik. Seperti halnya sebuah prosesor larik, prosesor vektor sangat efisien menjalankan serangkaian operasi-operasi pada pasangan elemen-elemen data. Namun perbedaannya adalah bahwa seluruh operasi tambahan pada prosesor vektor dijalankan dalam sebuah elemen penambah tunggal yang memiliki banyak pipeline. Cray Research yang didirikan oleh perusahaan Seymour Cray menghasilkan banyak prosesor vektor, yang dimulai dengan Cray-1 yang meniru model tahun 1974 dan terus berlanjut hingga model-model saat ini (Cray Research kini adalah bagian dari SGI).

Baik prosesor larik maupun prosesor vektor bekerja pada larik data. Kedua prosesor ini menjalankan instruksi-instruksi tunggal; misalnya, menambahkan elemen-elemen secara berpasangan untuk dua vektor. Tetapi sementara processor larik menjalankan itu dengan menggunakan banyak elemen penambah sebagai elemen elemen dalam vektor, prosesor vektor memiliki konsep berupa sebuah register vektor, yang terdiri dari sekumpulan register konvensional yang dapat diload dari memori dalam suatu instruksi tunggal, yang sebenarnya meload register-register tersebut dari memori secara serial. Kemudian sebuah instruksi penambahan vektor menambahkan elemen-elemen yang berpasangan dari dua vektor semacam itu dengan memasukkan elemen-elemen berpasangan tersebut ke sebuah adder yang memiliki pipeline dari register-register kedua vektor tersebut. Hasil dari adder tersebut adalah sebuah vektor barn, yang dapat disimpan ke

dalam sebuah register vektor, atau digunakan secara langsung sebagai sebuah operand untuk operasi vektor lairinya.

Prosesor-prosesor larik masih dalam tahap pembuatan, namun prosesor - prosesor ini memiliki pangsa pasar yang makin lesu, karena mereka hanya dapat bekerja dengan baik pada masalah-masalah yang mengharuskan digunakannya cara perhitungan yang sama pada banyak kumpulan data secara bersamaan. Prosesor prosesor larik dapat menjalankan beberapa operasi data secara lebih efisien dibanding komputer-komputer vektor, tapi mereka membutuhkan lebih banyak hardware dan sangat sulit untuk diprogramkan. Di sisi lain, prosesor-prosesor vektor dapat ditambahkan pada sebuah prosesor konvensional. Hasilnya adalah bahwa bagian-bagian program tersebut yang dapat divektorkan dapat dijalankan dengan cepat dengan menarik keuntungan dari unit vektor, sedangkan bagian program lainnya dapat dijalankan pada sebuah prosesor tunggal yang konvensional.

7.6.3. Multiprosesor

Elemen pengolahan dalam sebuah prosesor larik bukan hanya CPU-CPU yang berdiri sendiri; karena ada satu unit kontrol yang dimiliki bersama oleh semua CPU tersebut. Sistem paralel pertama kita dengan CPU multiple berkembang sepenuhnya adalah multiprosesor, suatu sistem dengan lebih dari satu CPU yang memiliki sebuah memori bersama, seperti sekelompok orang dalam suatu ruangan yang memili papan tulis bersama. Karena setiap CPU dapat membaca atau menulis bagian apa saja dari memori, mereka harus berkoordinasi (dalam software) agar tidak saling berebut jalurnya masing-masing.

Berbagai skema implementasi dapat dilaksanakan. Salah satu skema paling sederhana adalah memiliki sebuah bus tunggal dengan banyak CPU dan satu memo yang seluruhnya dipasang ke dalam bus tersebut. Sebuah diagram dari multiproses berbasiskan bus seperti itu ditunjukkan pada Gambar 7.7.(a). Banyak perusahaan membuat sistem-sistem semacam itu.

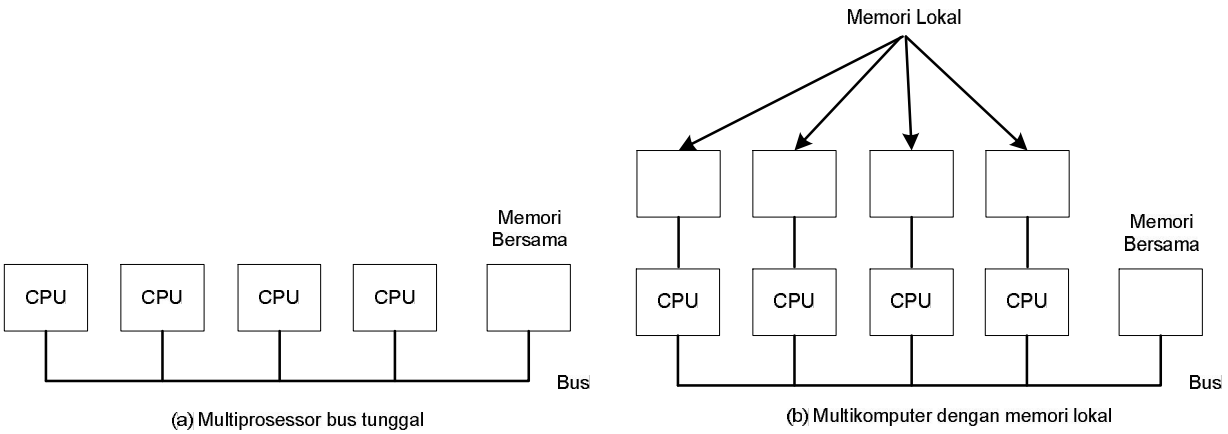
Tidak diperlukan banyak imajinasi untuk menyadari bahwa dengan sejumlah besar prosesor cepat yang selalu berusaha mengakses memori melalui bus yang sama, konflik-konflik akan terjadi. Para perancang multiprosesor telah mengajukan berbagai macam skema untuk mengurangi konflik ini dan

meningkatkan kinerja. Salah satu desain, yang ditunjukkan pada Gambar 7.7.(b), memberikan setiap prosesor memori lokalnya sendiri-sendiri, yang tidak dapat diakses ke memori-memori yang lain: Memori ini dapat digunakan untuk kode program dan item-item data tidak perlu dibagi-bagi. Akses ke memori pribadi ini tidak menggunakan bus utama, sehingga sangat mengurangi lalulintas bus. Skema-skema lain (contoh, caching) juga dapat dijalankan.

Multiprosesor-multiprosesor memiliki keunggulan dibanding jenis-jenis komputer paralel lainnya sehingga model ,pemrograman terhadap satu memori bersama mudah untuk ditangani. Contoh, bayangkan sebuah program untuk mencari cell-cell kanker dalam sebuah foto jaringan tertentu yang diambil dengan menggunakan sebuah mikroskop. Foto tersebut dapat disimpan dalam memori bersama, dan masing-masing prosesor ditunjuk untuk menelusuri daerah tertentu dari foto itu. Karena setiap prosesor memilik akses ke memori seluruhnya, mempelajari sebuah cell yang dimulai di daerah yang telah ditunjuk tapi kemudian melewati batas hingga masuk ke dalam daerah sekitarnya bukan merupakan suatu masalah.

7.6.4. Multikomputer

Meskipun multiprosesor dengan sejumlah kecil prosesor relatif mudah untuk dibuat, prosesor-prosesor besar sangat sulit untuk dibuat. Kesulitan tersebut terutama dalam menghubungkan semua prosesor ke memori. Untuk mengatasi masalah ini, banyak perancang telah melupakan ide untuk memiliki memori bersama dan justru membuat sistem-sistem yang terdiri dari banyak komputer yang saling terhubung, yang masing-masing memiliki memorinya sendiri-sendiri, tetapi tidak ada memori bersama. Sistem-sistem ini disebut sistem multikomputer.



Gambar 7.7.

Dalam sebuah multikomputer CPU-CPU berkomunikasi dengan saling mengirim pesan-pesan, seperti e-mail, tapi dengan kecepatan lebih tinggi. Untuk sistem-sistem besar, menghubungkan setiap komputer ke setiap komputer lainnya tidaklah praktis, jadi topologi-topologi seperti titik 2D dan 3D, pohon, dan cincin sudah biasa digunakan. Oleh karena itu, pesan-pesan dari satu komputer ke komputer lainnya harus melewati satu komputer perantara atau lebih untuk berjalan dari sumber ke tujuan. Bagaimanapun juga, waktu untuk menyampaikan pesan berdasarkan urutan beberapa mikrodetik dapat dicapai tanpa banyak kesulitan. Multikomputer-multikomputer dengan hampir 10.000 CPU telah dibuat dan dioperasikan.

Karena multiprosesor-multiprosesor lebih mudah untuk diprogramkan dan multikomputer-multikomputer lebih mudah untuk dibuat, telah dilakukan banyak penelitian untuk merancang sistem-sistem campuran yang memadukan keunggulankeunggulan masing-masing. Komputer-komputer semacam itu mencoba untuk mewujudkan harapan mengenai memori bersama, tanpa mempertimbangkan biaya yang dibutuhkan untuk membuat memori tersebut.