BAB 9

Komputer Pipeline

9.1 KONSEP PIPELINE

Pada umumnya, efisiensi sebuah sistem komputer dinilai berdasarkan kecepatan perangkat keras dan fasilitas-fasilitas perangkat lunak. Penilaian ini, disebut sebagai **throughput**, didefinisikan sebagai jumlah pemrosesan yang dapat dikerjakan dalam suatu interval waktu tertentu. Salah satu teknik yang mendorong peningkatan suatu sistem *throughput* yang cukup hebat disebut sebagai **pemrosesan pipeline**. Konsep pemrosesan pipeline dalam suatu komputer mirip dengan suatu baris perakitan dalam suatu pabrik industri. Ambil contoh, suatu proses pembuatan mobil. Ketika sebuah mobil dibuat, mobil tersebut berpindah sepanjang ban berjalan dengan berurutan, melewati beberapa stasiun. Pada setiap stasiun, dikerjakan sebagian proses konstruksi pada mobil itu, kemudian berpindah lagi ke stasiun berikutnya. Perpindahan mobil itu dari satu stasiun ke stasiun lainnya, memungkinkan beberapa mobil berada pada baris perakitan pada waktu yang bersamaan, masing-masing pada stasiun yang terpisah. Dengan demikian, hal ini mengakibatkan kita menghasilkan mobil dari baris perakitan satu per satu

secara berurutan. Tanpa teknik baris perakitan seperti ini, pengerjaan suatu mobil tidak dapat dimulai sampai mobil yang sebelumnya benar-benar selesai.

Pemrosesan pipiline dalam suatu komputer diperoleh dengan membagi suatu fungsi yang akan dijalankan menjadi beberapa subfungsi yang lebih kecil dan merancang perangkat keras yang terpisah, disebut sebagai **tingkatan** (*stage*), untuk setiap subfungsi. Stage-stage itu kemudian dihubungkan bersama-sama dan membentuk sebuah **pipeline** tunggal (atau **pipe**) untuk menjalankan fungsi asli tersebut.

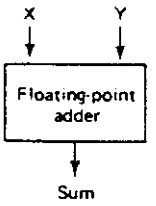
Contoh 9.1

Perhatikan fungsi penambahan dua angka floating-point (lihat Bagian 2.4) yang perlihatkan pada Gambar 9-1. Fungsi tersebut dapat kita bagi menjadi tiga subfungsi terpisah berikut ini:

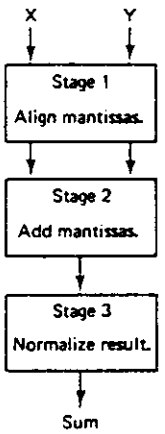
1. Sejajarkan mantissa-mantissa yang ada.
2. Tambahan mantissa-mantissa tersebut.
3. Normalisasikan hasilnya.

Jika kita merancang sebuah stage perangkat keras untuk setiap subfungsi ini, kita dapat menyusun fungsi tersebut sebagai suatu pipiline tiga-stage seperti diperlihatkan pada Gambar 9-2.

Keuntungan proses penambahan secara pipiline ini adalah bahwa dua input yang baru dapat dimulai melalui pipa tersebut segera sesudah dua input sebelumnya melewati stage 2. Hal ini berarti bahwa jumlah penambahan akan tersedia dengan kecepatan yang sama dengan kecepatan input. Gambar 9-3 memperlihatkan secara sistimatis bagaimana sekumpulan angka floating-point akan bergerak melalui penambah (adder) pipeline yang sederhana pada Gambar 9-2. Pada saat pasangan pertama angka-angka itu dihasilkan oleh stage 3 [bagian (d) pada



Gambar 9-1 Adder floating-point



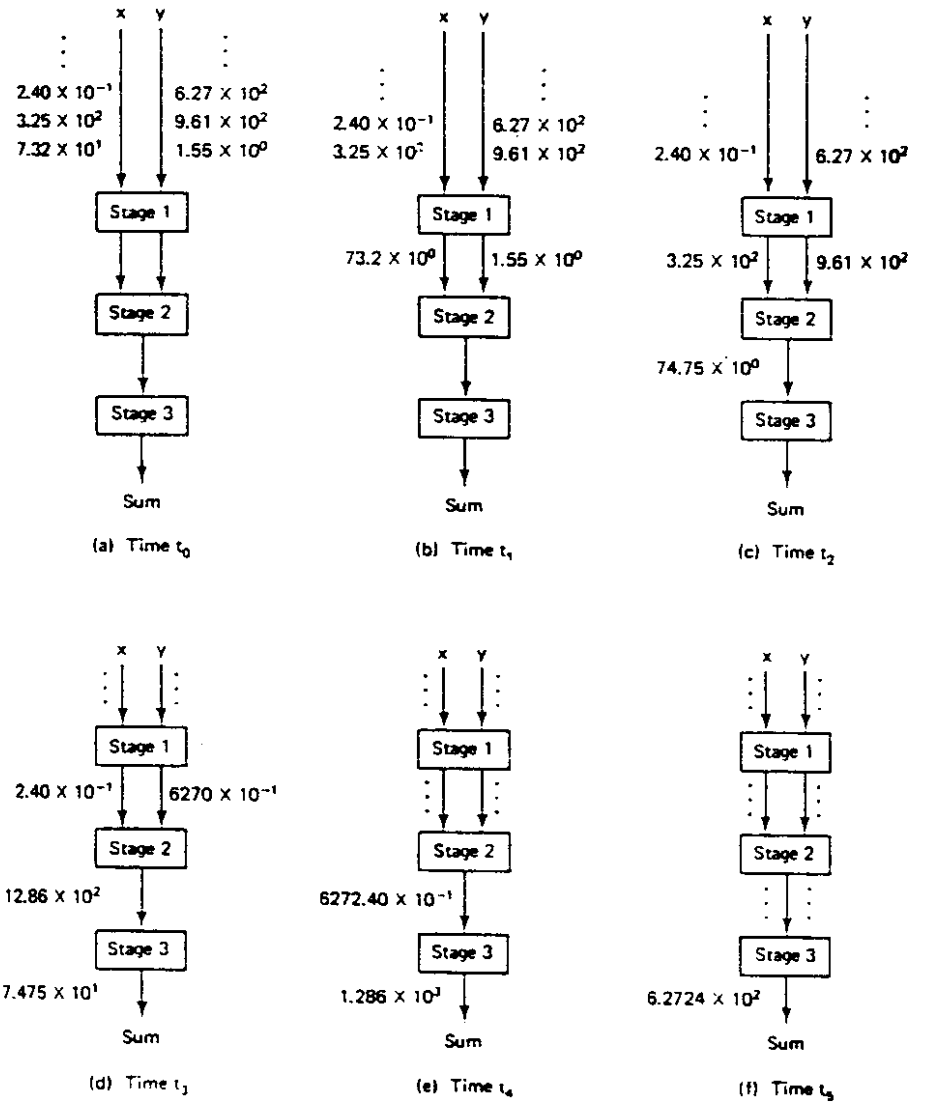
Gambar 9-2 Pemrosesan pipeline pada sebuah adder floating-point

gambar] maka pasangan kedua telah disejajarkan dan ditambahkan dan hanya perlu dinormalisir pada stage 3 [bagian (e)]. Dengan menggunakan pipeline, jumlah selisih waktu antara hasil pertama dan kedua merupakan jumlah waktu yang diperlukan untuk menormalisir sebuah angka. Tanpa suatu pipeline, waktu antara hasil-hasil tersebut merupakan waktu kumulatif yang diperlukan untuk semua ketiga subfungsi tersebut.

Kita mendefinisikan suatu **komputer pipeline** sebagai suatu komputer dengan komponen perangkat keras pipeline. Definisi ini mencakup kebanyakan komputer dewasa ini. Namun, mereka berbeda pada tingkatan pipelining mereka. Pada bab ini, kita membahas beberapa masalah-masalah pokok dalam perancangan dan penerapan pipelining dalam suatu komputer.

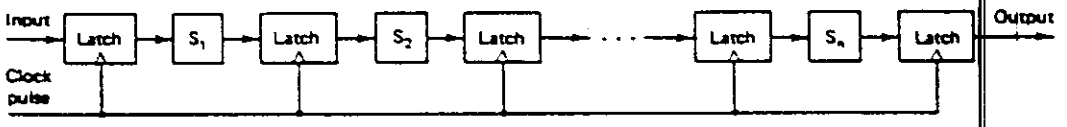
9.2 SINKRONISASI PADA PIPELINE

Pada semua baris perakitan industri, efisiensi suatu pipeline dapat berkurang jauh akibat suatu **bottleneck**. Suatu *bottleneck* terjadi sewaktu pemrosesan pada suatu stasiun, atau stage, menghabiskan waktu lebih lama daripada stage lainnya. Karena itu, idealnya, kita menginginkan agar setiap stage dalam suatu pipe menghabiskan jumlah waktu proses yang *sama*. Meskipun kita dapat memisahkan suatu

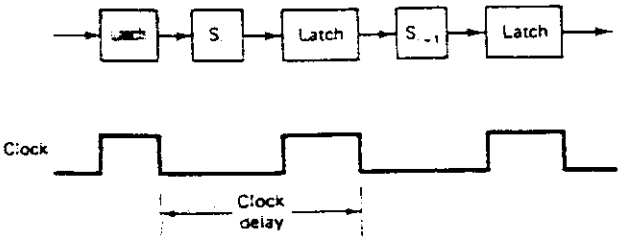


Gambar 9-3 Kemajuan angka-angka melalui suatu adder pipeline

fungsi menjadi beberapa subfungsi dengan waktu proses yang relatif sama, perbedaan logika dari setiap stage akan menyukarkan kita untuk menghasilkan waktu



Gambar 9-4 Pipeline linier dasar (S = stage)



Gambar 9-5 Periode clock pada sebuah pipeline

proses yang sama pada setiap stage. Untuk menyamakan waktu yang diperlukan pada setiap stage maka stage-stage tersebut harus **disinkronisasikan**. Hal ini biasa dilakukan dengan menyisipkan **kunci-kunci** (*latch*) sederhana (register cepat, lihat Bagian 3.7), antara stage-stage tersebut. Gambar 9-4 memperlihatkan sebuah pipeline **linier** dasar dimana setiap stage benar-benar sirkuit kombinasional yang terpisah dari stage berikutnya dalam pipeline oleh adanya latch. Ada juga latch, masing-masing pada bagian pipe paling awal dan satu lagi pada bagian paling akhir untuk memaksa input yang sinkron dan memastikan output yang sinkron.

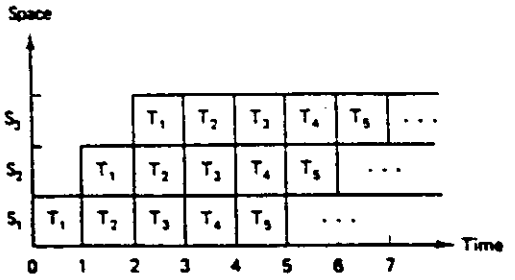
Waktu yang diperlukan untuk lewat dari suatu latch melalui stage ke latch berikutnya disebut sebagai **penangguhan clock** (*clock delay*) dan diperlihatkan pada Gambar 9-5. Karena hanya ada satu keseragaman penangguhan clock untuk seluruh pipeline maka latch disinkronkan sesuai dengan waktu proses maksimum pada masing-masing stage individual dalam pipeline tersebut. Bahkan jika hanya ada satu stage tersebut yang mempunyai waktu proses terpanjang maka penangguhan clock di-set sebesar waktu proses yang paling lama. Dengan demikian, suatu prinsip perancangan yang berhubungan dengan efisiensi adalah dengan

membagi fungsi yang sedang di-pipeline menjadi beberapa subfungsi yang memiliki implementasi perangkat keras dengan waktu proses yang relatif sama.

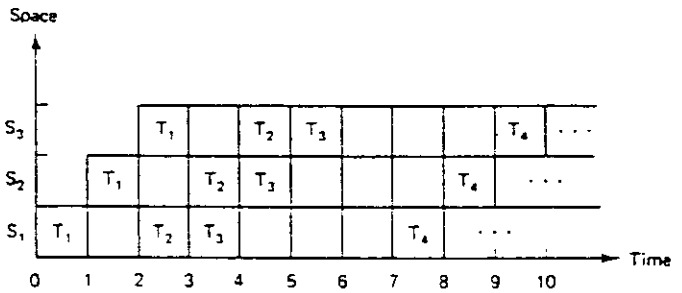
9.3 EFISIENSI PIPELINE

Untuk mengilustrasikan operasi-operasi yang saling tumpang tindih dalam pipeline linier digunakan suatu **diagram ruang-waktu** (*space-time diagram*). Gambar 9-6 memperlihatkan diagram semacam itu untuk sebuah pipeline tiga-stage. Tiap stage diwakili oleh sumbu ruang. Tugas-tugas yang diperlihatkan dalam diagram itu berhubungan dengan sekelompok input yang bergerak sepanjang pipeline. Diagram tersebut menunjukkan kelompok input mana, atau tugas, pada stage yang mana dalam pipe tersebut dan pada penangguhan clock yang mana. Dalam gambar tersebut, inputnya terus berlanjut, di mana hal tersebut sangat efisien tetapi tidak selalu terjadi demikian. Terkadang kelompok input yang baru tidak langsung tersedia atau, seperti yang akan kita lihat nanti, terlambat. Dalam hal ini, kita mungkin akan mempunyai diagram ruang-waktu seperti pada Gambar 9-7. Di sini, kelompok input kedua, T_2, menunggu satu penangguhan clock lebih lama sebelum masuk ke dalam pipe; input T_3 segera menyusul setelah input T_2; dan input T_4 menunggu tiga penangguhan clock tambahan sebelum masuk ke dalam pipe.

Dengan melihat pada diagram ruang-waktu sebuah pipeline, kita dapat menentukan *throughput* pipeline tersebut, yang merupakan jumlah hasil yang dapat diselesaikan per satuan waktu. Tingkat kecepatan ini merefleksikan kecepatan hitung pipeline tersebut. Untuk menghitung *throughput* itu, pada umumnya kita mengabaikan **waktu startup awal** (*initial startup time*) yang diperlukan untuk mengisi pipe, yaitu waktu yang diperlukan agar kelompok input pertama meme-



Gambar 9-6 Diagram ruang-waktu untuk sebuah pipeline



Gambar 9-7 Diagram ruang-waktu dengan input yang terlambat

nuhi seluruh pipe tersebut. Sebagai contoh, mari kita perhatikan pipeline yang ditunjukkan pada Gambar 9-6. Waktu startup awal pipeline ini adalah tiga penangguhan clock. Pada penangguhan clock yang kedelapan, hasil kelima sudah tersedia. Karena itu *throughput*-nya adalah 5/(8 - 3), yaitu sama dengan 1. Hal ini berarti bahwa setelah pengisian pipeline, akan tersedia satu hasil pada setiap penangguhan clock. Demikian juga, *throughput* untuk pipeline yang diperlihatkan pada Gambar 9-7 adalah 4/(11 - 3) = 0,5. Ini berarti bahwa secara rata-rata 0,5 hasil akan tersedia pada setiap penangguhan clock, atau satu hasil setiap dua penangguhan clock. Jadi secara umum, semakin tinggi *throughput*-nya semakin tinggi pula efisiensi pipeline tersebut.

Dalam teori, suatu fungsi pipeline hampir selalu lebih baik daripada fungsi non-pipeline. Satu-satunya hal yang tidak lebih baik adalah bahwa suatu fungsi tidak dapat dibagi-bagi menjadi beberapa sub-fungsi yang berbeda. Namun dalam kenyataannya, harga pipeline tersebut sangat mempengaruhi kapan dan apakah fungsi-fungsi yang di-pipeline-kan lebih baik atau tidak. Harga ini dapat disebabkan, tetapi tidak hanya terbatas, oleh faktor-faktor berikut ini:

1. Tambahan kunci (*latch*) perangkat keras yang diperlukan.
2. Kendali yang diperlukan untuk penjadwalan input tersebut (akan kita lihat nanti dalam bab ini bahwa input ke sebuah pipeline tidak selalu dapat terjadi pada setiap penangguhan clock).
3. Waktu yang dihabiskan oleh data dalam latch untuk menyesuaikan diri dengan suatu penangguhan clock yang seragam.

4. Jumlah rata-rata input yang akan tersedia agar penggunaan pipeline menjadi efisien.

Faktor terakhir, labih daripada yang lainnya, menentukan titik potong (*cutoff point*) apakah proses pipeline lebih baik atau tidak. Tidak perduli seberapa efisien suatu pipeline, jika ia akan digunakan untuk satu kelompok input pada suatu waktu, semua overhead yang diperlukan untuk mem-pipeline fungsi tersebut akan membuatnya lebih lambat dan lebih mahal dibandingkan dengan suatu fungsi yang tidak di-pipeline-kan. Namun jika jumlah kelompok input yang akan diproses pada basis regular sangat besar maka overhead pipeline tersebut akan lebih terjangkau. (Gambar 9-11 memperlihatkan suatu graph yang membandingkan waktu kerja (*run time*) untuk sebuah contoh pipeline tertentu).

9.4 KLASIFIKASI PIPELINE

Pipeline dapat kita klasifikasikan menurut fungsi dan konfigurasinya. Secara fungsional, mereka diklasifikasikan menjadi tiga kelompok pokok yaitu: pipelining aritmatika, instruksi dan prosesor. Ramamoorthy dan Li (1977) mengajukan tiga skema untuk mengklasifikasikan pipeline menurut konfigurasi dan strategi kendalinya: unifungsi atau multifungsi; statis atau dinamis; dan skalar atau vektor.

Klasifikasi Berdasarkan Fungsi

Pipelining aritmatika. Proses segmentasi fungsi dari ALU dari sistem yang muncul dalam katagori ini. Suatu contoh dari fungsi pipeline aritmatika diberikan dalam Bagian 9.7.

Pipelining instruksi. Dalam suatu komputer nonpipeline, CPU bekerja melalui suatu siklus yang berkesinambungan dari fetch-decode-eksekusi untuk semua instruksinya. Proses fetch suatu instruksi tidak akan dimulai sampai eksekusi instruksi sebelumnya selesai. Untuk mem-pipeline fungsi ini, instruksi-instruksi yang berdampingan di-fetch dari memori ketika instruksi yang sebelumnya di-decode dan dijalankan. Proses pipelining instruksi, disebut juga **instruction lihat-ke-muka** (*look-ahead*), mem-fetch instruksi secara berurutan. Dengan demikian, jika suatu instruksi menyebabkan percabangan keluar dari urutan itu maka pipe akan dikosongkan dari seluruh instruksi yang telah di-fetch sebelumnya dan instruksi percabangan (*branched-to instruction*) tersebut di-fetch. Proses pipeli-

ning instruksi dikerjakan pada hampir semua komputer berkemampuan tinggi (*high-perfomance*) dewasa ini.

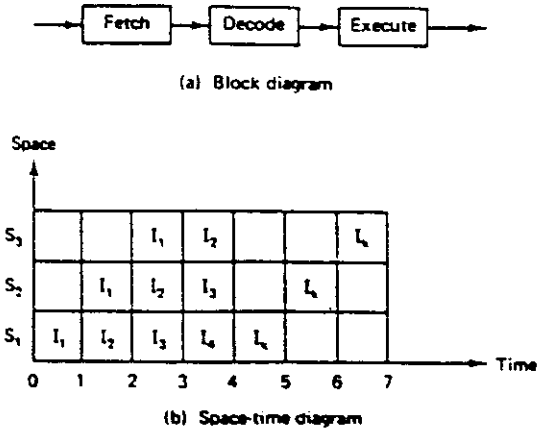
Contoh 9.2

Perhatikan pipeline *look-ahead* tiga-stage pada Gambar 9-8(a). Anggap bahwa instruksi I_2 merupakan cabang ke instruksi I_k. Karena itu pipeline harus dikosongkan dari semua instruksi yang telah di-fetch sebelumnya, dalam kasus ini I_3 dan I_4. Hal ini diperlihatkan pada diagram ruang-waktu pada Gambar 9-8(b).

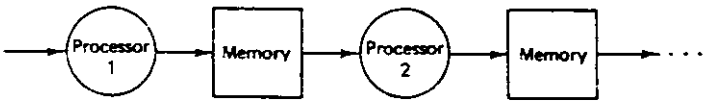
Pipelining prosesor. Sewaktu stage dari suatu pipeline merupakan prosesor aktual dan latch-latch saling berbagi memori antara prosesor-prosesor tersebut maka pipeline itu disebut sebagai **pipeline prosesor**. Dalam pipeline ini, setiap prosesor mempunyai suatu tugas tertentu yang akan dijalankan pada aliran data, seperti diperlihatkan pada Gambar 9-9. Pipelining banyak prosesor (multiple processor) merupakan konsep yang relatif baru dan belum umum.

Klasifikasi Berdasarkan Konfigurasi

Unifungsi versus multifungsi. Kemampuan suatu pipeline menjalankan hanya satu jenis pokok operasi disebut sebagai **pipeline unifungsi**. Misalnya, perkalian floating-point mensyaratkan pipeline agar juga menjalankan operasi yang sama pada setiap kelompok input. Jika pipeline dapat menjalankan fungsi-fungsi yang berbeda maka disebut sebagai **pipeline multifungsi**. Fungsi-fungsi



Gambar 9-8 Pipeline instruksi look-ahead

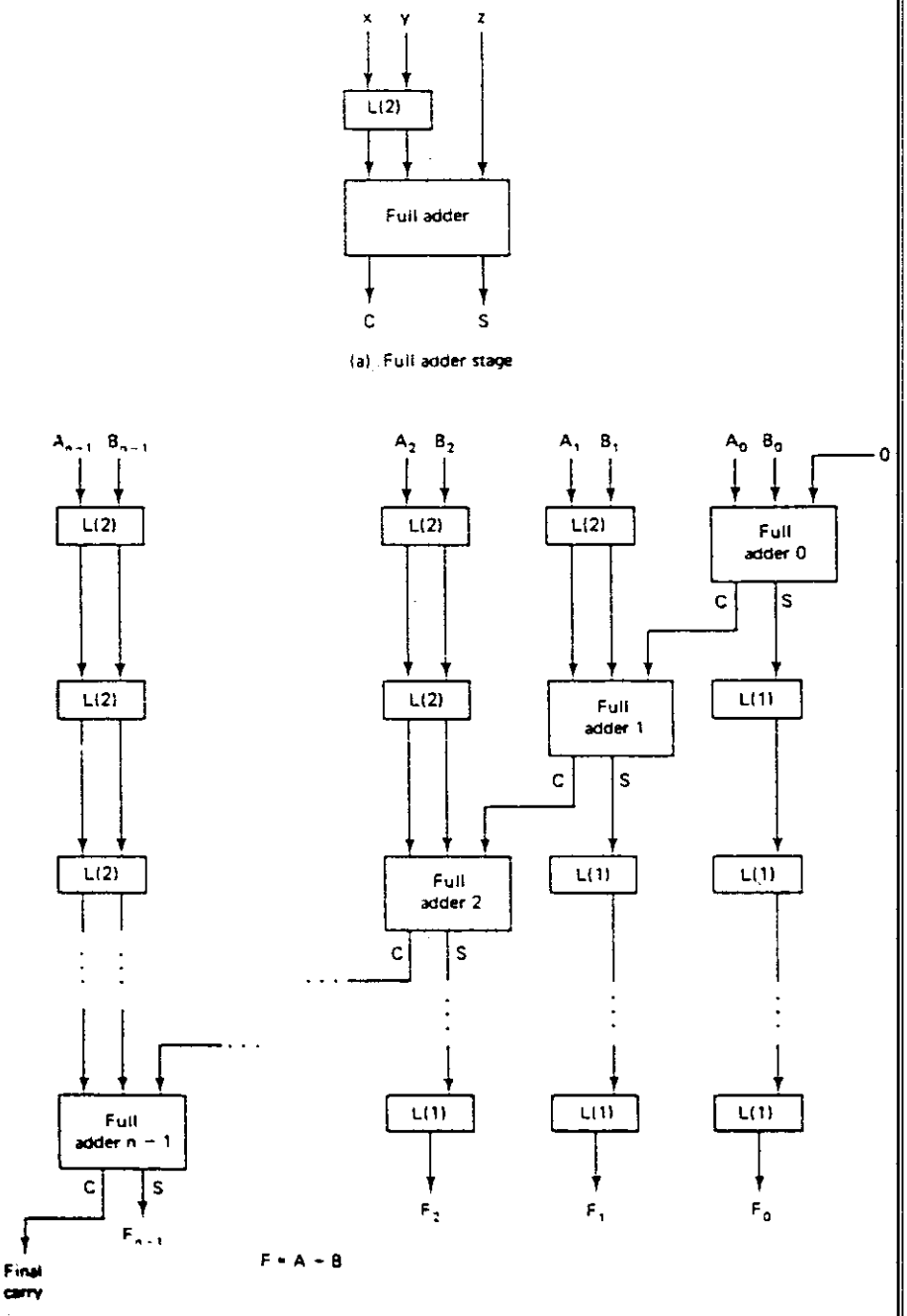


Gambar 9-9 Pipeline multiprosesor

yang berbeda itu bisa dijalankan baik pada waktu yang bersaman ataupun berbeda, dengan menghubungkan subkelompok-subkelompok stage yang berbeda dalam pipeline. Pipeline disusun seperlunya sesuai dengan nilai input kendali tambahan.

Statis versus dinamis. Ketika instruksi-instruksi yang berjenis sama akan dijalankan secara bersamaan waktunya maka digunakan **pipeline statis**. Pipeline ini dapat berupa pipeline fungsional maupun multifungsional tetapi mungkin mengasumsikan hanya satu konfigurasi fungsional pada suatu waktu. Suatu pipeline multifungsi statis dapat bekerja paling baik jika fungsi yang akan dijalankan tidak sering berubah. Antara perubahan, pipeline terlihat sebagai pipeline unifungsi dan mengulangi operasi yang sama berulang-ulang. Sebelum mengganti fungsi tersebut, kelompok input terakhir dari fungsi sebelumnya harus benar-benar telah melewati pipeline dan proses ini disebut sebagai *mengairi pipa* (*draining the pipe*). Kemudian pipeline dikonfigurasikan untuk fungsi yang baru dan input yang baru boleh masuk ke dalam pipe.

Dengan **pipeline dinamis**, beberapa konfigurasi fungsional dapat muncul sekaligus. Hal ini berarti harus digunakan pipeline multifungsional. Dalam kasus ini, konfigurasi pipe berubah-ubah secara konstan, tergantung pada data mana untuk fungsi yang mana pada stage yang mana untuk setiap penangguhan clock. Pipeline dinamis memerlukan kendali yang sangat kompleks dan mekanisme perangkai untuk mengkonfigurasikan pipe bagi input-input tertentu. Untuk alasan ini, pipelining aktual tidak berada di bawah kendali programer melainkan dibangun ke dalam arsitektur mesin tersebut. Jika suatu pipeline dalam sebuah sistem perlu mengubah fungsinya secara berkala maka harga implementasi pipeline dinamis melebihi pertimbangan harga dari waktu menganggur (*idle time*) yang dihasilkan dari proses draining pipeline statis secara terus menerus.



Gambar 9-10 Adder paralel n-bit pipeline [$L(i)$ = latch i-bit; clock delay = waktu dari satu adder]

Skalar versus vektor. Suatu **pipeline skalar** memproses serangkaian operasi skalar pada operand skalar (yang berhubungan dengan masing-masing angka bukannya vektor) seperti ditunjukkan oleh program. Salah satu contoh berupa operasi ADD dalam loop FOR. **Pipeline vektor** dirancang untuk memproses instruksi vektor dengan menggunakan operand vektor. Komputer yang mempunyai instruksi-instruksi vektor disebut sebagai **prosesor vektor** dan akan dibahas pada Bab 10. Contoh pipeline di bab ini dapat digunakan sebagai pipeline skalar atau sebagai bagian dari pipeline vektor.

9.5 CASCADING ATAS STAGE

Meskipun kebanyakan pipeline dirancang dengan membagi-bagi suatu fungsi menjadi beberapa stage, namun kita dapat membuat sebuah pipeline yang kompleks tanpa salah satu stage yang penting. Proses ini dikenal dengan **cascading atas stage**. Iteratife cascading atas stage dapat digunakan untuk membuat unit pipeline unifungsi maupun multifungsi. Kita akan menjelaskan konsep cascading dari stage sesuai cara dalam contoh pipeline unifungsi berikut ini.

Perhatikan full adder yang diberikan pada Bagian 3.6. Ia terdiri atas tiga input dan dua output. Inputnya merupakan dua bit yang akan ditambahkan, x dan y, dan carry sebelumnya, z. Outputnya merupakan bit penjumlahan, S, dan carry baru yang dihasilkan, C. Untuk membuat full adder menjadi sebuah stage, kita menambahkan sebuah latch 2-bit pada input x dan y seperti diperlihatkan pada Gambar 9-10(a). Dengan menghubungkan sejumlah stage ini dan latch tambahan bersama-sama, seperti diperlihatkan pada Gambar 9-10(b), kita akan menghasilkan adder paralel n-bit yang bekerja seperti berikut:

1. Pada penangguhan clock yang pertama, A_0 dan B_0 akan lewat melalui adder 0, sementara A_1 dan B_1 melalui A_{n-1} dan B_{n-1} di-load ke dalam latch tingkat pertama.
2. Pada penangguhan clock kedua, output C dari adder 0 telah tersedia dan bersama-sama dengan A_1 dan B_1, lewat melalui adder 1. Pada waktu yang sama, A_2 dan B_2 melalui A_{n-1} dan B_{n-1} menuju ke latch tingkat kedua dan output S dari adder 0 berpindah ke suatu latch.
3. Pola yang sama berlanjut untuk n penangguhan clock, dimana waktu bagi semua output F tersedia.

Suatu hal yang khusus tentang adder ini adalah bahwa dalam langkah 2 sebuah pasangan input baru A dan B dapat mulai berpindah melalui pipe tersebut. Sepanjang pasangan angka yang baru sebagai input tetap berlanjut maka adder akan mengeluarkan sebuah hasil akhir pada setiap penangguhan clock (setelah sejumlah n penangguhan clock awal). Inilah alasan mengapa setup ini dianggap sebagai adder paralel n-bit pipeline.

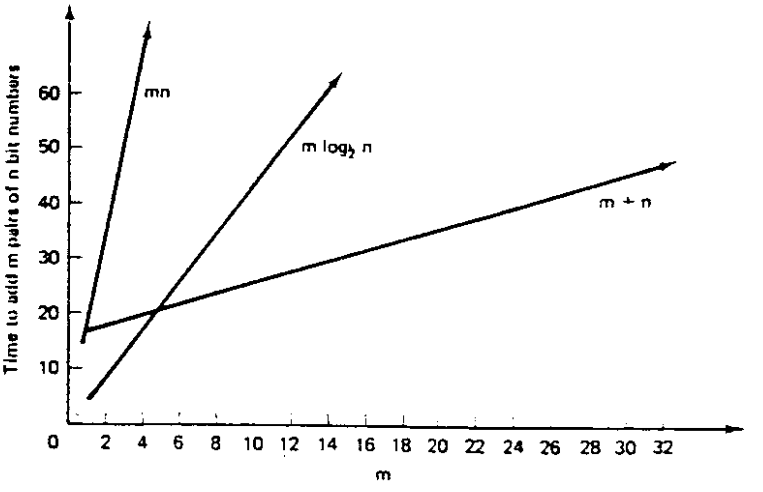
Seperti telah kita ketahui dari Bagian 3.6, kita dapat membuat sebuah adder paralel multibit dari sebuah full adder. Pendekatan yang digunakan dalam bagian itu menghasilkan sebuah *adder ripple*. Karena carry yang paling tidak signifikan harus disebarkan ke seluruh stage adder, waktu yang diperlukan untuk menambah dua angka n-bit secara kasar merupakan n dikali waktu yang diperlukan untuk melalui satu stage full adder. Oleh karena itu, waktu yang diperlukan untuk menambah sejumlah m pasangan angka n-bit adalah sejumlah mn. Sebaliknya, adder paralel pipeline kita memerlukan n dikali waktu yang diperlukan untuk melalui suatu stage full adder untuk menghasilkan hasil penjumlahan dari pasangan angka yang pertama. Tetapi setelah itu, akan tersedia satu hasil penjumlahan pada setiap penangguhan clock, sepanjang input tetap masuk ke dalam pipe. Ini berarti bahwa waktu yang diperlukan untuk menambahkan sejumlah m pasangan angka n-bit adalah sejumlah $m + n$.

Gambar 9-11 memperlihatkan suatu perbandingan waktu yang berbeda untuk macam-macam address yang disebutkan di atas. Keuntungan adder pipeline hanya kita sadari sewaktu penghematan waktu melebihi pertimbangan overhead pipeline tersebut, termasuk latch tambahan yang diperlukan. Untuk menambahkan satu pasangan angka n-bit pada suatu waktu akan lebih cepat dan murah jika kita menggunakan sebuah adder cepat (menghabiskan waktu sejumlah $m \log_2 n$) ataupun *adder ripple* daripada kita menggunakan adder pipeline.

9.6 PRINSIP PIPELINING SECARA UMUM

Tabel Reservasi

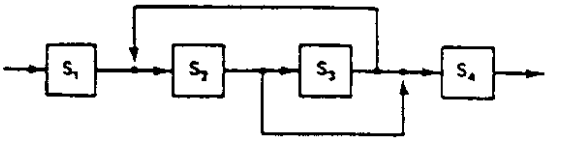
Sejauh ini kita hanya memperhatikan pipeline linier yang sederhana seperti pada Gambar 9-4. Namun demikian, sangat sering pipeline mempunyai hubungan **umpan balik** (*feedback*) dan **umpan maju** (*feedforward*), seperti diperlihatkan pada Gambar 9-12. Agar sederhana, pipeline digambarkan tanpa latch, meskipun



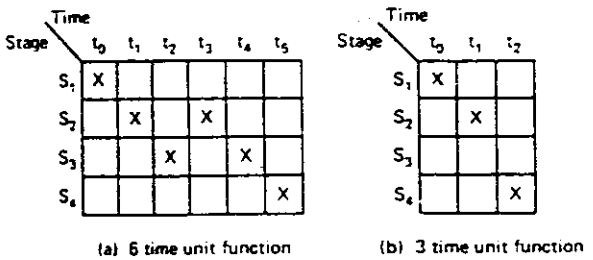
Gambar 9-11 Perbandingan run time antara macam-macam adder

mereka dianggap ada di antara semua stage. Hubungan dari suatu stage dengan stage sebelumnya disebut sebagai hubungan umpan balik (*feedback*), sedangkan hubungan dari suatu stage dengan stage berikutnya disebut sebagai hubungan umpan maju (*feedforward*).

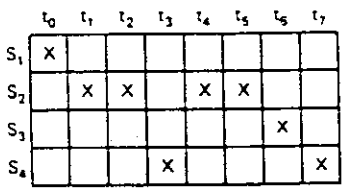
Ketika berhubungan dengan suatu pipeline umum, harus ditentukan urutan proses setiap stage. Untuk memberi karakteristik pada struktur antarhubungan (*interconnection*) dan arus data flow pada suatu pipeline umum digunakan **tabel reservasi**. Tabel tersebut menunjukkan bagaimana stage-stage tersebut diproses secara berurutan untuk suatu evaluasi fungsi tertentu. Baris-barisnya mewakili stage yang ada pada pipeline dan kolomnya menunjukkan penangguhan clock yang berurutan. Gambar 9-13 memperlihatkan contoh dari dua tabel reservasi yang menunjukkan dua fungsi berbeda yang dapat dievaluasi dengan pipeline



Gambar 9-12 Pipeline umum dengan umpan balik dan umpan maju



Gambar 9-13 Tabel reservasi untuk pipeline pada Gambar 9-12



Gambar 9-14 Tabel reservasi dengan stage penangguhan clock multiple

pada Gambar 9-12. Tabel reservasi pada Gambar 9-13(a) memperlihatkan bahwa evaluasi suatu fungsi mensyaratkan agar data dikerjakan melalui stage-stage tersebut dalam pipeline dengan urutan S_1, S_2, S_3, S_2, S_3, S_4. **Waktu evaluasi** (*evaluation time*) fungsi ini adalah 6 penangguhan clock, yaitu total jumlah unit waktu dalam tabel reservasi. Urutan stage pada evaluasi fungsi sesuai Gambar 9-13(b) adalah S_1, S_2, S_4. Meskipun fungsi ini menggunakan pipeline yang sama seperti fungsi pada bagian (a), waktu evaluasinya hanya 3 penangguhan clock.

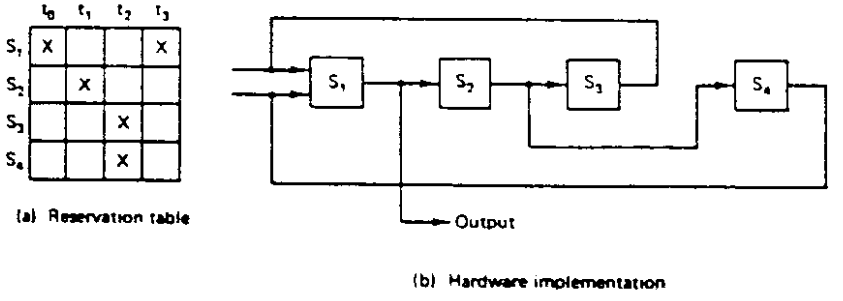
Jika sebuah tabel reservasi mempunyai lebih dari satu tanda dalam baris tertentu, seperti diperlihatkan pada Gambar 9-14, hal ini berarti bahwa stage yang diwakili oleh baris tersebut digunakan oleh lebih dari satu kali untuk evaluasi fungsi. Jika dua kolom yang berdampingan mempunyai tanda dalam baris yang sama, hal ini berarti bahwa stage yang dipertanyakan memerlukan penangguhan clock yang lebih banyak daripada yang lainnya untuk diproses. Jumlah penangguhan clock yang diperlukan untuk stage itu adalah jumlah tanda yang berdampingan dalam baris tersebut. Kapanpun stage ini masuk ke dalam evaluasi fungsi

maka diperlukan jumlah penangguhan clock yana sama. Stage S_2 pada Gambar 9-14 merupakan contoh dari stage dengan penangguhan clock multiple.

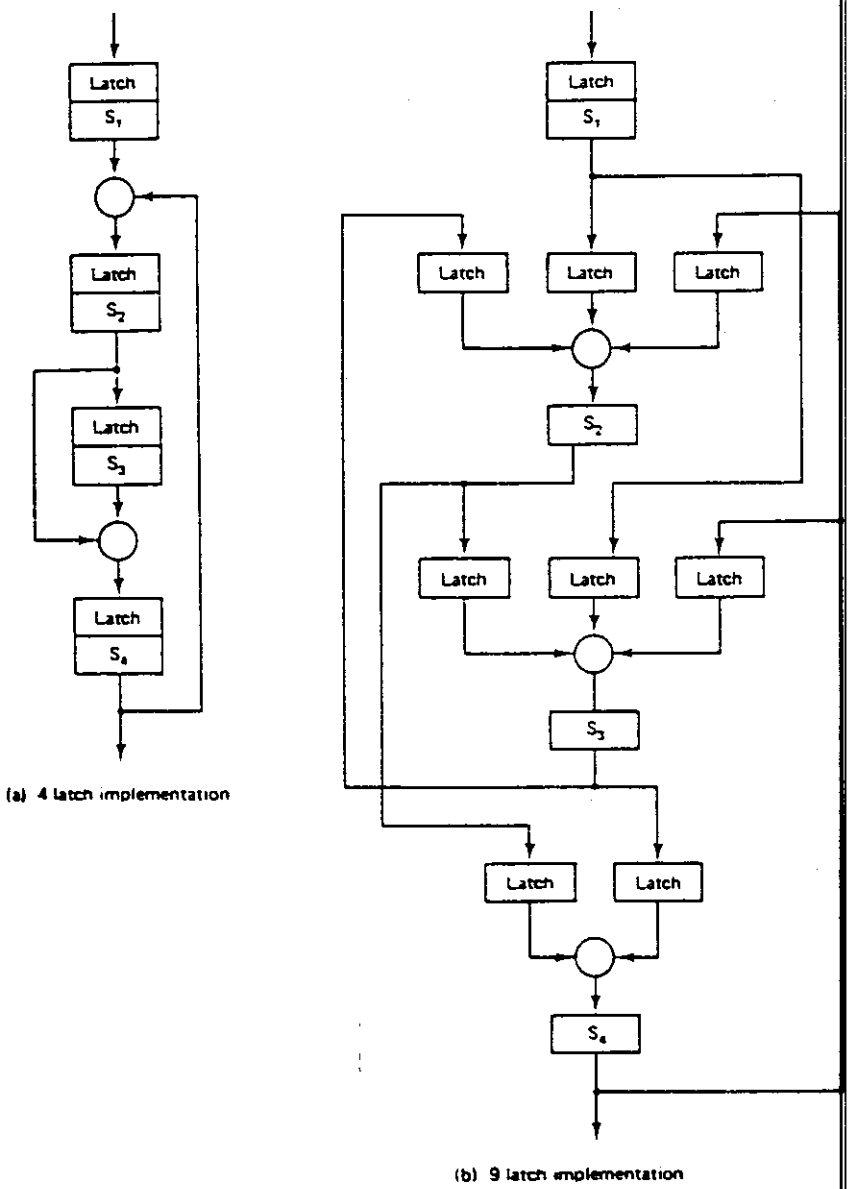
Jika tabel reservasi memiliki lebih dari satu tanda dalam sebuah kolom, hal ini berarti bahwa dua stage atau lebih secara bersamaan waktunya memproses data untuk evaluasi fungsi yang sama. Bagian (a) pada Gambar 9-15 memberikan sebuah contoh tabel semacam itu dan bagian (b) memperlihatkan suatu kemungkinan implementasi perangkat keras. Perhatikan bahwa hal ini berbeda dengan penggunaan stage yang umum secara serentak oleh kelompok input yang berbeda-beda. Tanda di dalam kolom ketiga pada Gambar 9-15(a) menunjukkan pemrosesan output yang beriringan pada stage S_2, sehingga satu kelompok input akan diproses dengan dua stage yang berbeda pada waktu yang sama.

Implementasi perangkat keras bagi suatu tabel reservasi tidak harus unik. Sebagai contoh, tabel reservasi pada Gambar 9-14 dapat diimplementasikan oleh kedua pipeline pada Gambar 9-16. Lingkaran pada gambar tersebut menunjukkan multiplexer. Implementasi yang benar-benar digunakan tergantung pada pertimbangan-pertimbangan tertentu seperti harga, ketersediaan pipe-pipe yang ada untuk melayani suatu fungsi yang baru dan kebutuhan untuk menghasilkan sistem tersebut.

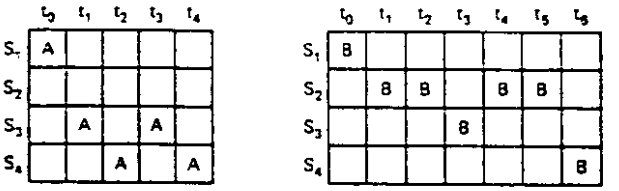
Suatu implementasi yang dapat dikonfigurasikan untuk menerapkan lebih dari satu fungsi disebut sebagai suatu **pipeline multifungsi**. Dengan pipeline multifungsi, kita perlu menyertakan perangkat kendali dan perangkai yang dapat diatur untuk berhubungan dengan tabel reservasi bagi masing-masing fungsi. Sebagai contoh, Gambar 9-17 memperlihatkan tabel reservasi untuk dua evaluasi fungsi yang berbeda untuk pipeline pada Gambar 9-16(b). Jika suatu tabel reservasi menunjukkan fungsi dari suatu pipeline multifungsi maka nama fungsi tersebut



Gambar 9-15 Pipeline dengan pemrosesan state yang beriringan



Gambar 9-16 Implementasi perangkat keras yang mungkin untuk suatu pipeline yang berhubungan dengan tabel reservasi pada Gambar 9-14



(a) 5 time unit function A (b) 7 time unit function B

Gambar 9-17 Multifungsi untuk sebuah pipeline

digunakan sebagai tanda dalam tabel, sebagai ganti hanya sekadar *X*. Untuk fungsi A, output dari S_1 harus diarahkan ke S_3, dimana untuk fungsi B, output dari S_1 harus ke S_2. Multiplexer yang di-set tergantung pada fungsi yang sedang dievaluasi.

Penjadwalan dan Pencegahan Adanya Tubrukan

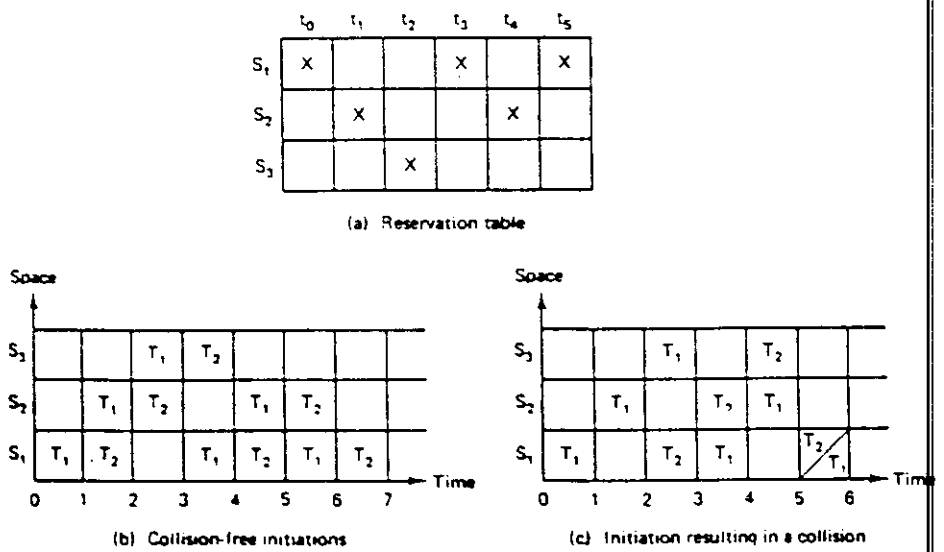
Inisiasi dari suatu tabel reservasi berhubungan dengan permulaan suatu evaluasi (tugas) fungsi tunggal yang akan mengikuti path yang dinamai oleh tabel. Jika suatu inisiasi dibuat maka **pengendali pipeline** (*pipeline controller*) harus mencadangkan stage pada pipeline yang tepat bagi data inisiasi tersebut dengan waktu yang relatif yang ditentukan oleh tabel reservasi. Jika data dari dua inisiasi yang berbeda akan masuk ke dalam stage yang sama pada waktu yang sama maka akan terjadi **tubrukan** (*collision*). Karena suatu stage tidak dapat menghitung dua hasil yang berbeda secara bersamaan waktunya maka tubrukan dapat dicegah oleh pengendali pipeline.

Contoh 9.3

Perhatikan tabel reservasi yang diperlihatkan pada Gambar 9-18(a). Diagram ruang-waktu (space-time diagram) pada Gambar 9-18(b) menunjukkan dua inisiasi yang bebas dari tubrukan, satu pada t_0 dan satu lagi pada t_1. Diagram ruang-waktu pada Gambar 9-18(c) menunjukkan dua inisiasi, pada t_0 dan t_2, yang akan menghasilkan sebuah tubrukan pada t_5.

Kita akan menghadirkan suatu metode untuk menentukan **algoritma penjadwalan** (*scheduling algorithm*) yang efisien yang dapat diimplementasikan oleh pengendali pipeline untuk mencegah adanya tubrukan-tubrukan.

Jumlah unit waktu antara dua inisiasi dalam suatu pipeline disebut sebagai **latensi** (*latency*). Serangkaian latensi di antara inisiasi yang berdampingan dise-



Gambar 9-18 Inisiasi untuk sebuah pipeline

but sebagai **rangkaian latensi** (*latency sequence*). Jika rangkaian latensi mengulangi dirinya sendiri, hal ini disebut sebagai **siklus latensi** (*latency cycle*). Dalam pipeline linier yang statis tanpa hubungan *feedback* atau *feedforward*, seperti pada Gambar 9-4, suatu siklus latensi yang sederhana {1} akan menyediakan *throughput* yang optimal bagi pipeline tersebut. Hal ini berarti bahwa inisiasi yang baru (selama masih ada input) dimulai setiap penangguhan clock. Untuk pipeline yang ditunjukkan pada Gambar 9-18 a), bagaimanapun juga, siklus latensi {1} tidak dapat digunakan. Jika digunakan maka inisiasi pertama dan kedua akan dimulai tanpa adanya masalah, namun inisiasi yang ketiga akan bertubrukan pada saat t_5, seperti diperlihatkan pada Gambar 9-19(a). Namun suatu siklus latensi {1, 6} akan bekerja dengan baik untuk pipeline ini seperti diperlihatkan pada Gambar 9-9(b). Dalam kasus ini, inisiasi kedua dimulai 1 penangguhan clock setelah inisiasi pertama, namun inisiasi ketiga tidak akan dimulai sampai 6 penangguhan clock setelah inisiasi kedua. Siklus ini mengulangi dirinya sendiri sehingga inisiasi keempat dimulai 1 penangguhan clock setelah inisiasi yang ketiga, sedangkan inisiasi kelima akan dimulai 6 penangguhan clock setelah inisiasi keempat dan seterusnya.

Pada suatu algoritma penjadwalan pipeline, kita tertarik pada siklus latensi yang bebas dari tubrukan untuk suatu alur inisiasi yang terus-menerus. Artinya, kita menginginkan suatu siklus latensi yang tidak pernah menghasilkan tubrukan antara inisiasi-inisiasi yang ada di dalam pipe. Salah satu kemungkinan siklus latensi yang akan selalu bekerja adalah siklus sederhana $\{n\}$, dimana n adalah total jumlah unit waktu di dalam tabel reservasi. Tentu saja hal ini mengurangi efisiensi pipeline sampai ke tingkat nol bagi penggunaan suatu implementasi nonpipeline dan sangat tidak diinginkan.

Untuk dapat menentukan algoritma penjadwalan, kita mulai dengan menghasilkan semua kemungkinan siklus latensi yang bebas tubrukan. Kumpulan siklus ini didasarkan pada **kumpulan latensi yang terlarang** (*forbidden set of latencies*), **F**, untuk pipeline yang mengandung semua kemungkinan latensi yang menyebabkan tubrukan antara dua inisiasi *yang berurutan*. Untuk dua inisiasi yang berurutan akan terjadi tubrukan jika latensi di antara mereka sama dengan jarak kolom antara sembarang X pada baris yang sama di tabel reservasi.

Contoh 9.4

Lihat kembali Gambar 9-18(a), X pada baris 1, kolom t_0 dan t_3 mempunyai jarak kolom sebesar 3 di antara mereka; X pada baris 1, kolom t_3 dan t_5 mempunyai jarak kolom sebesar 2 di antara mereka; X pada baris 1, kolom t_0 dan t_5 mempunyai jarak kolom sebesar 5 di antara mereka; dan X pada baris 2, kolom t_1 dan t_4 mempunyai jarak kolom sebesar 3 di antara mereka. Oleh karena itu, kumpulan latensi yang terlarang, F, adalah {2, 3, 5}. Dapat kita lihat pada Gambar 9-18(c), latensi antara inisiasi pertama dan kedua adalah 2. Karena 2 ada di dalam F maka terjadi suatu tubrukan.

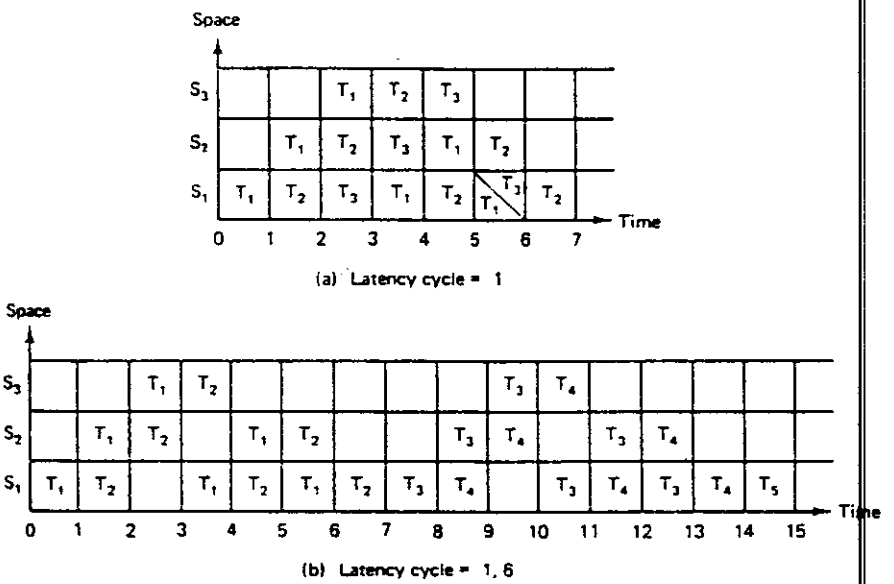
Vektor tubrukan (*collision vector*), **C**, merupakan suatu representasi vektor biner dari kumpulan latensi yang terlarang. Ia memiliki sejumlah n bit yang dimulai dari bit yang paling tidak signifikan, bit ke-i bernilai 1 jika i merupakan latensi terlarang dan jika tidak bernilai 0; sehingga kita tulis:

$$C = (c_n \ldots c_2 c_1) \tag{9.1}$$

dimana $c_i = 1$ jika i ada di dalam F dan $c_i = 0$ jika sebaliknya.

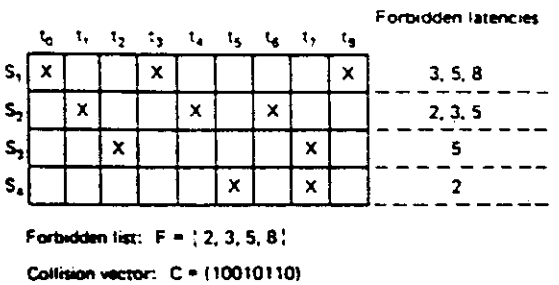
Contoh 9.5

Untuk tabel reservasi pada Gambar 9-20, $F = \{2, 3, 5, 8\}$ dan $C = (10010110)$. Dimulai dari bit yang paling tidak signifikan pada C, hal ini berarti bahwa dua inisiasi yang berurutan tidak akan bertubrukan satu sama lainnya jika dan hanya jika latensi di antara mereka adalah 1, 4, 6, 7 atau lebih besar dari atau sama dengan 9 unit waktu.



Gambar 9-19 Siklus latensi yang berbeda untuk sebuah pipeline

Vektor tubrukan, *C*, (dan kumpulan latensi yang terlarang, *F*), mewakili latensi-latensi yang akan menyebabkan tubrukan di antara dua inisiasi yang berdampingan. Namun, yang sebenarnya ingin kita ketahui adalah latensi yang menyebabkan tubrukan di antara dua inisiasi manapun. Untuk menentukan latensi semacam ini, perlu observasi atas hal-hal khusus yang dimiliki vektor tubrukan: Bit yang paling tidak signifikan pada *C* menunjukkan apakah akan terjadi (atau tidak) suatu tubrukan



Gambar 9-20 *F* dan *C* pada sebuah tabel reservasi

di antara dua inisiasi pertama yang berurutan, yaitu yang satu dimulai tepat setelah yang lainnya. Karena itu jika ada selang waktu di antara dua inisiasi tersebut (dalam unit waktu) maka vektor tubrukan dapat digeser ke sebelah kanan sejumlah unit waktu ini dengan maksud untuk tetap menunjukkan kumpulan latensi yang terlarang itu. Vektor yang digeser dengan demikian berhubungan dengan posisi inisiasi yang pertama saat ini di dalam pipe dengan memperhatikan inisiasi kedua yang tertunda.

Contoh 9.6

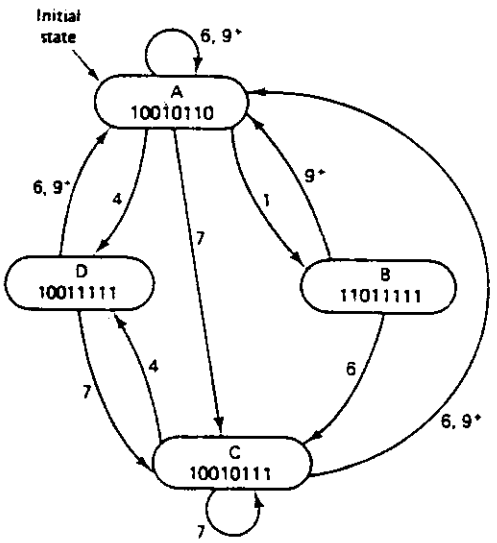
Jika pada sat t_{12}, C = (1100101) dan tidak terjadi inisiasi untuk dua unit waktu maka saat t_{14}, C = (0011001).

Berdasarkan ha-hal yang dimiliki ini, digunakan prosedur berikut ini untuk menentukan latensi yang terlarang untuk seluruh inisiasi baru, dengan memperhatikan seluruh inisiasi sebelumnya.

Anggaplah S adalah register geser-kanan, load paralel (lihat Bagian 3.7) dengan input serial (pada stage flip-flop paling kiri) dan output serial (dari stage flip-flop paling kanan). Input serial secara permanen dihubungkan ke 0; setiap saat sewaktu S bergeser ke kanan maka akan disebarkan 0 ke stage berikutnya. Asumsikan bahwa S menyimpan vektor tubrukan suatu pipeline. Kamudian dengan nilai tertentu dalam S, suatu inisiasi dari sebuah tugas yang baru dapat terjadi pada saat $t + k$ hanya jika, setelah menggeser register S sejumlah k kali, outputnya bernilai 0. Jika outputnya bernilai 1 maka akan terjadi sebuah tubrukan dan tugas yang baru tidak boleh ber-inisiasi. Pada input pertama ke dalam pipe, vektor tubrukan awal disimpan di S. Swaktu sebuah tubrukan baru terjadi saat $t + k$, vektor tubrukan yang asli masih menunjukkan kemungkinan tubrukan antara inisiasi ini dan inisiasi yang sebelumnya, namun ia tidak menunjukkan kemungkinan-kemungkinan tubrukan antara inisiasi yang telah ada di dalam pipe dengan inisiasi yang sebelumnya. Untuk menentukan semua latensi yang terlarang untuk saat ini, vektor tubrukan yang asli harus di di-OR-kan dengan vektor tubrukan yang telah digeser sebelumnya, S. Vektor yang dihasilkan sekarang akan menunjukkan latensi untuk inisiasi yang berdampingan yang akan (atau tidak) menghasilkan tubrukan. Penyimpanan vektor ini dalam register S berhubungan dengan penyimpanan state sistem saat ini.

Contoh 9.7

Anggaplah C_1 = (10010110) merupakan vektor tubrukan awal pada pipeline yang diperlihatkan oleh Gambar 9-20. Karena itu, S saat ini menyimpan C_1. Asumsikan



Gambar 9-21 Diagram state untuk pipeline pada Gambar 9-20

bahwa terjadi sebuah inisiasi baru setelah 4 unit waktu. Hal ini mungkin terjadi karena setelah empat kali penggeseran S menjadi (00001001) dan output terakhirnya bernilai 0. Pengoperasian OR terhadap S dengan C_1 akan menghasilkan (10011111), yang merupakan nilai S yang baru. Berdasarkan nilai S ini, suatu inisiasi baru tidak dapat terjadi pada unit waktu berikutnya. Jika terjadi maka inisiasi itu tidak akan bertubrukan dengan pendahulunya, namun akan bertubrukan dengan inisiasi sebelum pendahulunya. Inilah kasus yang ditunjukkan oleh Gambar 9-19(a).

Suatu **diagram keadaan** (*state diagram*) (lihat Bagian 3.5), dapat diturunkan untuk menandai inisiasi tugas yang berdampingan dalam sebuah pipeline. Gambar 9-21 memperlihatkan diagram keadaan untuk pipeline pada Gambar 9-20. Keadaan awal berhubungan dengan vektor tubrukan awal. Keadaan lainnya menunjukkan vektor tubrukan setelah beberapa rangkaian latensi untuk inisiasi. Keadaan itu dihasilkan oleh prosedur yang baru saja diberikan. Setiap anak panah transisi menunjukkan latensi-latensi yang diperkenankan untuk inisiasi berikutnya dari satu state ke state lainnya. Notasi 9^+ pada diagram state menggambarkan suatu latensi yang lebih besar atau sama dengan 9. Contoh 9.7 menerangkan transisi nomor 4 dari state awal tersebut.

Diagram state menunjukkan, dalam bentuk kompak, semua kemungkinan rangkaian inisiasi untuk suatu pipeline. Tujuan utama algoritma penjadwalan adalah untuk menggunakan rangkaian yang memberikan *throughput tertinggi*. Kunci algoritma penjadwalan yang efisien adalah berdasarkan pada analisa siklus pada diagram state yang berhubungan dengan siklus latensi yang telah disebutkan sebelumnya. Suatu **siklus** terdiri atas sekelompok state yang berurutan dan latensi transisinya yang berasal dari sembarang state pada siklus, rangkaian transisi siklis menuntun kita melalui state lainnya dalam siklus tersebut dan kembali ke state awal. Notasi untuk siklus tersebut dapat berupa label-label state yang berurutan atau transisi-transisi yang berurutan. Jika suatu siklus terdiri atas state-state yang tidak berulang (*nonrepeating*) maka disebut sebagai suatu **siklus sederhana** (*simple cycle*).

Contoh 9.8

Dalam diagram state pada Gambar 9-21, suatu kumpulan yang berurutan {*A*, *C*, *D*} membentuk suatu siklus. Dimulai dari state *A*, transisi 7 dapat diikutkan ke state *C*, transisi 4 dapat diikutkan ke state *D* dan kemudian transisi 6 atau 9^+ dapat diikutkan kembali ke state *A*. Siklus ini sederhana dan juga dapat diwakilkan sebagai {7, 4, 6}.

Perhatikan, karena state dan transisinya dalam suatu urutan tertentu membentuk suatu siklus, state-state yang sama dapat membentuk suatu siklus yang berbeda dengan urutan yang berbeda. Untuk seluruh siklus, latensi rata-rata dihitung sebagai total latensi di dalam siklus dibagi jumlah state dalam siklus. Misalnya, latensi rata-rata dari siklus state *A*, *D* dan *C* pada Gambar 9-21 adalah (4 + 7 + 6)/3 = 5,667. Perhatikan bahwa suatu siklus tidak harus dimulai dari state awal. Pada Gambar 9-21, ada suatu siklus antara state *D* dan *C* yang latensi rata-ratanya sebesar (7 + 4)/2 = 5,5. Untuk mengoptimalkan strategi pengendalian pada pipeline tersebut, siklus latensi terbaik yang akan digunakan adalah siklus latensi yang mempunyai **latensi rata-rata paling rendah** (*MAL* atau *minimum average latency*). Tabel 9.1 memperlihatkan semua siklus sederhana untuk diagram state pada Gambar 9-21. MAL untuk pipeline ini adalah sebesar 4,25 unit waktu.

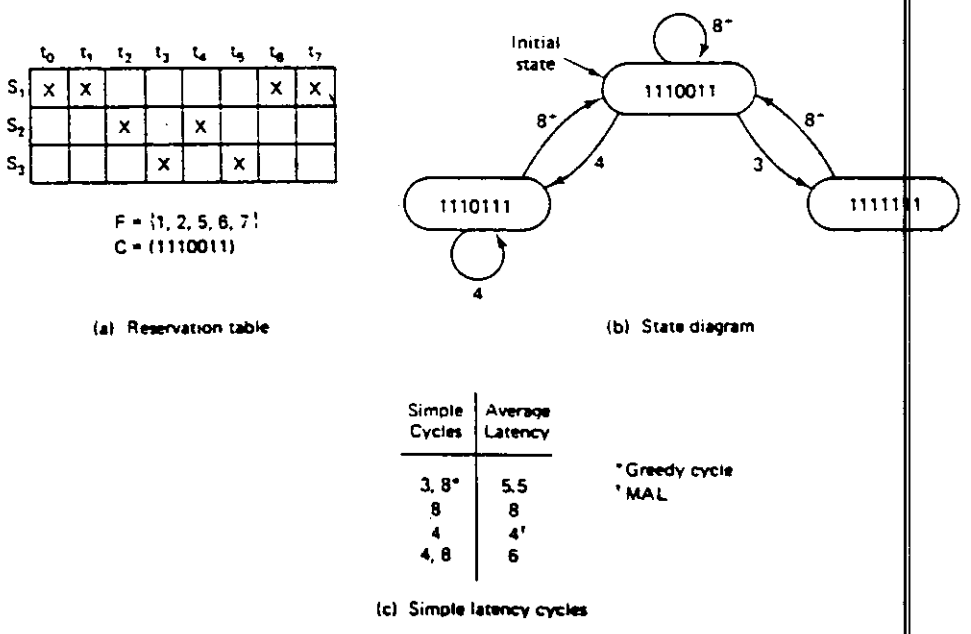
Meskipun strategi itu benar untuk diagram state pada Gambar 9-21, pemilihan path latensi terpendek dari setiap state, dikenal sebagai **siklus serakah** (*greedy cycle*), tidak selalu merupakan siklus yang optimal. Sebagai contoh, perhatikan tabel reservasi pada Gambar 9-22(a). Dapat kita lihat dalam diagram state pada Gambar 9-22(b) dan tabel yang berhubungan dengan Gambar 9-22(c), siklus {4} mempunyai latensi rata-rata yang lebih rendah daripada siklus serakah {3, 8}. Karena itu, dalam jangka panjang lebih efisien bagi kita untuk menunggu latensi keempat sebelum membuat inisiasi kedua daripada memulainya pada latensi ketiga.

TABEL 9.1 SIKLUS LATENSI SEDERHANA UNTUK PIPELINE PADA GAMBAR 9-21

Siklus Sederhana	Latensi Rata-rata
A	6,0
A-B	5,0
A-B-C	4,33
*A-B-C-D**	4,25+
A-C	6,5
A-C-D	5,667
A-D	5,0
A-D-C	5,667
C	7,0
C-D	5,5

* Siklus serakah
\+ Latensi rata-rata minimum

Metode penjadwalan yang dihadirkan untuk pipeline unifungsi dapat digeneralisasikan untuk merancang algoritma penjadwalan bagi pipeline multi-



Gambar 9-22 Pipeline di mana siklus serakah tidak mempunyai MAL

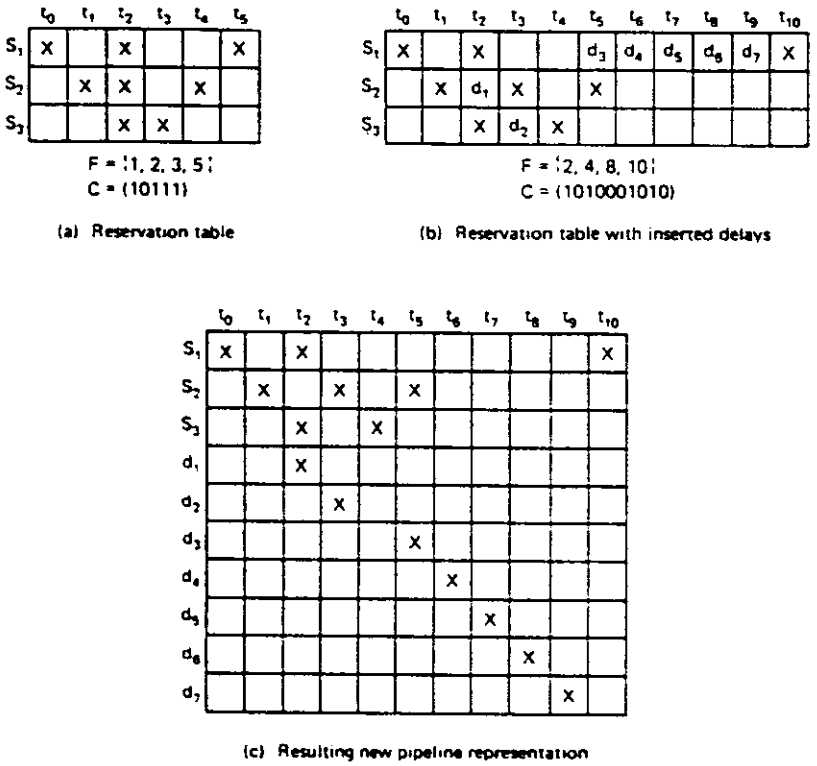
fungsi. Dalam kasus ini, inisiasi-inisiasi yang berdampingan pada fungsi-fungsi yang berbeda, juga untuk fungsi-fungsi yang sama, harus disertakan dalam diagram state. Sebagai contoh, pada pipeline multifungsi untuk dua fungsi, *A* dan *B*, rangkaian latensi harus disatukan dengan probabilita bahwa suatu inisiasi bagi fungsi *A* diikuti oleh inisiasi fungsi *A*, suatu inisiasi *A* akan diikuti oleh suatu inisiasi *B*, suatu inisiasi *B* diikuti oleh suatu inisiasi *B* dan suatu inisiasi *B* diikuti oleh suatu inisiasi *A*. Namun, sekali diagram state diturunkan maka ia akan diteruskan ke pengendali pipeline untuk meyakinkan siklus latensi yang diinginkan.

Penyisipan Penangguhan (Delay) untuk Throughput yang Optimal

Jumlah stage dalam sebuah pipeline tidak mempengaruhi *throughput* pipeline tersebut kecuali waktu startup awal. Jika pipeline itu merupakan suatu pipeline linier dasar dan inputnya berlanjut, maka setelah startup awal, hasil *throughput* akan dihasilkan satu per satu setiap penangguhan clock, tidak perduli dengan jumlah stage-nya. Dalam hal ini, siklus latensi yang sedang digunakan adalah {1}. Secara umum, kita dapat menggambarkan kemungkinan *throughput* suatu pipeline yang maksimum (setelah startup) sebagai kebalikan dari waktu latensi rata-rata yang minimum:

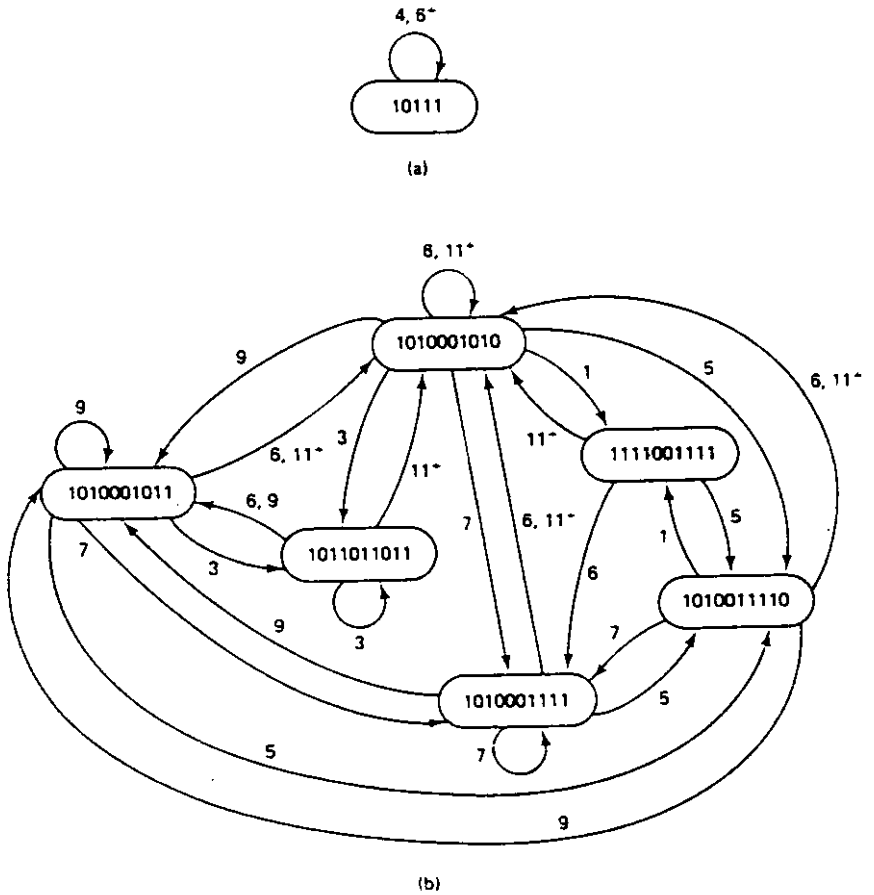
$$\textit{throughput} \text{ maksimal} = 1/\text{MAL} \tag{9.2}$$

Pada beberapa kasus, kita dapat menurunkan MAL dengan penambahan stage-stage **penangguhan tak-terhitung** (*noncompute delay*) pada pipeline itu. Meskipun hal ini meningkatkan jumlah stage dalam pipeline tersebut namun *throughput* aktualnya menjadi lebih baik. Suatu stage penangguhan tak-terhitung tidak mengubah nilai inputnya, namun semata-mata hanya menyimpan input untuk satu penangguhan clock. Gambar 9-23 menunjukkan suatu contoh, yang pertama kali diperkenalkan oleh Patel dan Davidson (1976), tabel reservasi yang MAL-nya berkurang dengan penyisipan stage penangguhan tak-terhitung. Setelah menentukan tempat untuk menyisipkan penangguhan tersebut, seperti pada Gambar 9-23(b), stage di-setup untuk setiap delay yang diperlukan. Karena digunakan stage yang terpisah untuk setiap penangguhan maka tidak ada latensi-latensi terlarang baru yang diperkenalkan karena adanya stage-stage baru. Pipeline baru yang dihasilkan ditunjukkan oleh Gambar 9-23(c). Diagram state pada Gambar



Gambar 9-23 MAL yang ditingkatkan dengan penyisipan penangguhan tak-terhitung

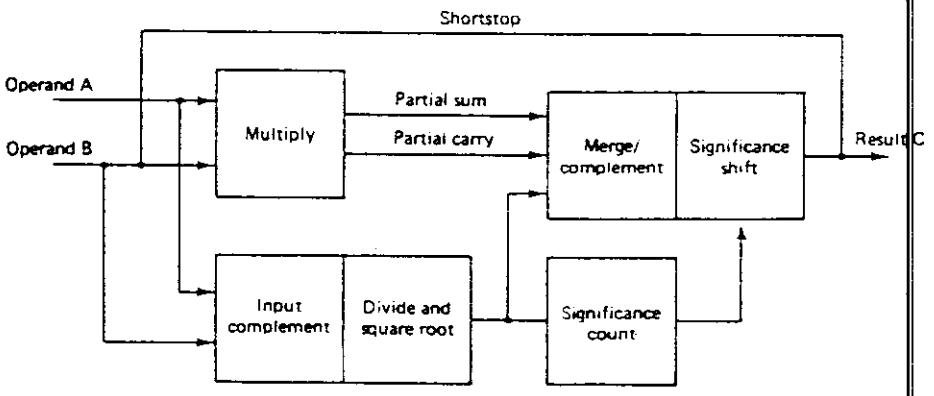
9-24 memperlihatkan bahwa MAL untuk pipeline tersebut telah berkurang dari 4 penangguhan clock seperti pada Gambar 9-24(a), yang berhubungan dengan Gambar 9-23(a), menjadi 3 penangguhan clock seperti Gambar 9-24(b), yang berhubungan dengan Gambar 9-23(c). Setiap inisiasi akan memakan waktu yang lebih lama di dalam pipe pada Gambar 9-23(c), tetapi inisiasi dapat dimulai lebih sering dan dengan demikian *throughput* maksimum dapat ditingkatkan dari 0,25 menjadi 0,33. Prosedur yang digunakan untuk menentukan tempat untuk meletakkan stage-stage penyisipan tidak akan dibahas dalam buku ini tetapi dapat kita temukan dalam Kogge (1981).



Gambar 9-24 Diagram state untuk pipeline pada Gambar 9-23

9.7 CONTOH PIPELINE MULTIFUNGSI

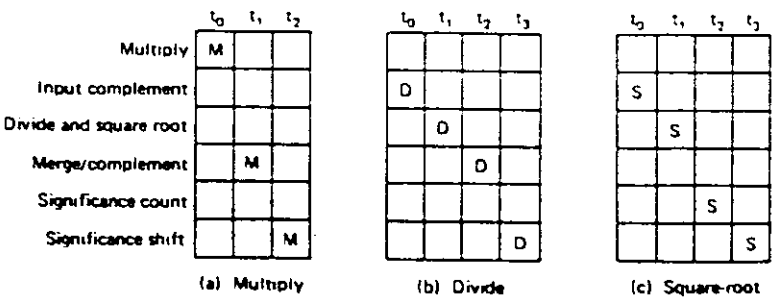
Contol Data Corporation (CDC) CYBER 205 merupakan komputer yang banyak memakai pipeline. Di antara pipeline-pipeline aritmatikanya terdapat pipeline perkalian floating-point. Ia merupakan pipeline multifungsi yang menjalankan operasi perkalian, pembagian dan akar pangkat dua. Konfigurasi pipeline ini diperlihatkan pada Gambar 9-25.



Gambar 9-25 Pipeline perkalian floating-point pada CYBER 205 (Courtesy of Control Data Corporation)

Jika operand A dan B akan dikalikan, pertama-tama kedua operand tersebut melewati stage perkalian, yang menghasilkan produk mantissa. Kemudian stage *penggabungan/komplemen* (*merge/complement*) menambahkan eksponen-eksponen mereka. Terakhir, angka floating point yang dihasilkan kita normalisir di dalam stage *pergeseran signifikan* (*significant shift*).

Baik operasi pembagian ataupun akar pangkat dua memerlukan pengurangan mantissa suatu operand dari operand lainnya (lihat algorithma pembagian pada Bagian 2.3). Untuk memungkinkan hal tersebut, operand yang sesuai dikomple-



Gambar 9-26 Tabel reservasi untuk pipeline pada Gambar 9-25 (M = *multiply* (perkalian), D = *divide* (pembagian), S = *square root* (akar kuadrat)

mentasikan dalam stage *pengkomplementasian input* (*input complement stage*) sebelum mantissa dibagi atau akar kuadrat dikerjakan pada stage *pembagian dan akar kuadrat* (*devide and square root stage*). Pada point ini, jika operasi tersebut merupakan sebuah divisi maka eksponen akan dikurangi dalam stage *penggabungan/komplemen* (*merge/complement stage*) dan hasilnya dinormalisir dalam stage *pergeseran signifikan* (*significant shift*). Tetapi jika operasi tersebut merupakan operasi akar kuadrat maka yang terlibat hanya sebuah eksponen tunggal. Sebagai ganti melewati stage *penggabungan/komplemen*, akar kuadrat mantissa akan melewati stage *perhitungan signifikan*. Hasilnya kemudian dinormalisir, seperti pada operasi lainnya, dalam stage *pergeseran signifikan*. Gambar 9-26 memperlihatkan tabel reservasi untuk ketiga fungsi ini.

CYBER 205 mempunyai sebuah hubungan umpanbalik *Shortstop* khusus, juga diperlihatkan pada Gambar 9-25. Hal ini memungkinkan hasil dari semua fungsi diarahkan rutenya langsung ke input fungsi lainnya tanpa harus pertama-tama disimpan di dalam register penengah. Hal ini sangat meningkatkan kecepatan komputer sewaktu mengerjakan instruksi vektor seperti pada penemuan produk dari seluruh elemen sebuah vektor.

9.8 MASALAH PIPELINE YANG BERULANG

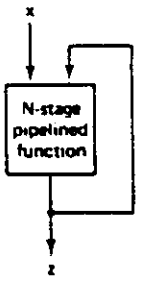
Diagram blok pada Gambar 9-27 memperlihatkan sebuah jenis khusus *pipeline dengan umpan balik* (*feedback*). Dalam kasus ini, output ke-i, z_i, merupakan sebuah fungsi input ke-i, x_i, dan sejumlah m output sebelumnya $z_{i-1}, z_{i-2}, \ldots, z_{i-m}$; yaitu:

$$z_i = f(x_i, z_{i-1}, z_{i-2}, \ldots, z_{i-m}) \tag{9.3}$$

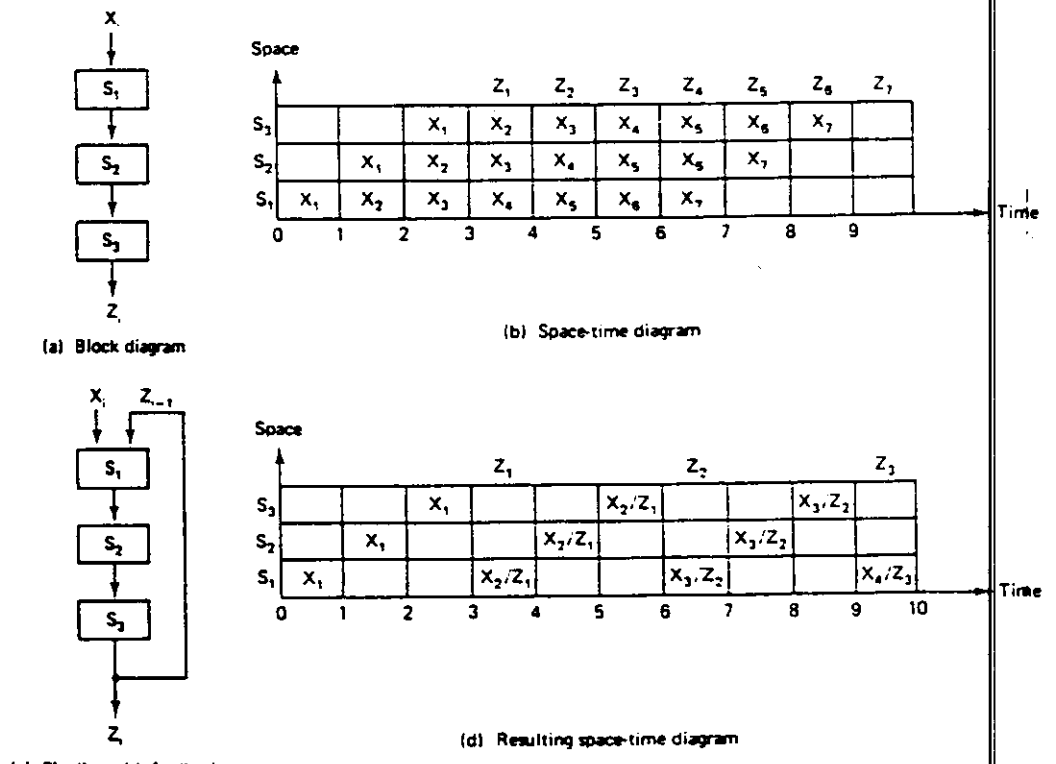
Persamaan ini disebut sebagai **persamaan perulangan** (*recurrence equation*). Masalah-masalah yang berhubungan dengan pipeline yang digambarkan oleh Persamaan (9.3) disebut sebagai **perulangan** (*recurrences*). Dalam usaha membuat suatu pipeline dengan perulangan, potensinya untuk merusak efisiensi pipeline sangat besar.

Contoh 9.9

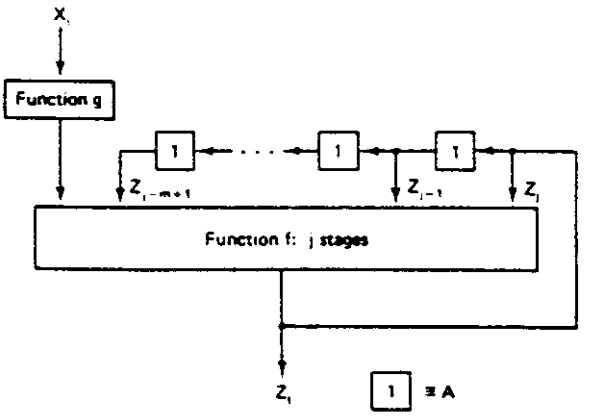
Perhatikan pipeline pada Gambar 9-28(a), yang terdiri atas tiga stage. Setelah waktu startup awal 3 penangguhan clock, suatu output z_i pada setiap penangguhan clock selama x_i terus-menerus dimasukkan ke dalam pipeline, seperti pada Gambar 9-28(b).



Gambar 9-27 Masalah perulangan



Gambar 9-28 Pipeline berulang dengan 3 stage



Gambar 9-29 Pipeline bagi masalah perulangan

Jika pipeline itu dimodifikasi seperti pada Gambar 9-28(c) maka hanya satu output z_i yang dikeluarkan setiap 3 penangguhan clock, seperti pada Gambar 9-28(d).

Seperti diperlihatkan pada Contoh 9.9, penyertaan output sebelumnya ke dalam input pipeline telah menghilangkan semua keuntungan pipeline. Meskipun x_2 tersedia pada saat t_2, tetapi ia tidak dapat diteruskan melalui pipe sampai z_1 tersedia pada saat t_4. Namun, saat hal ini terjadi, beberapa macam masalah perulangan dapat ditulis sebagai berikut:

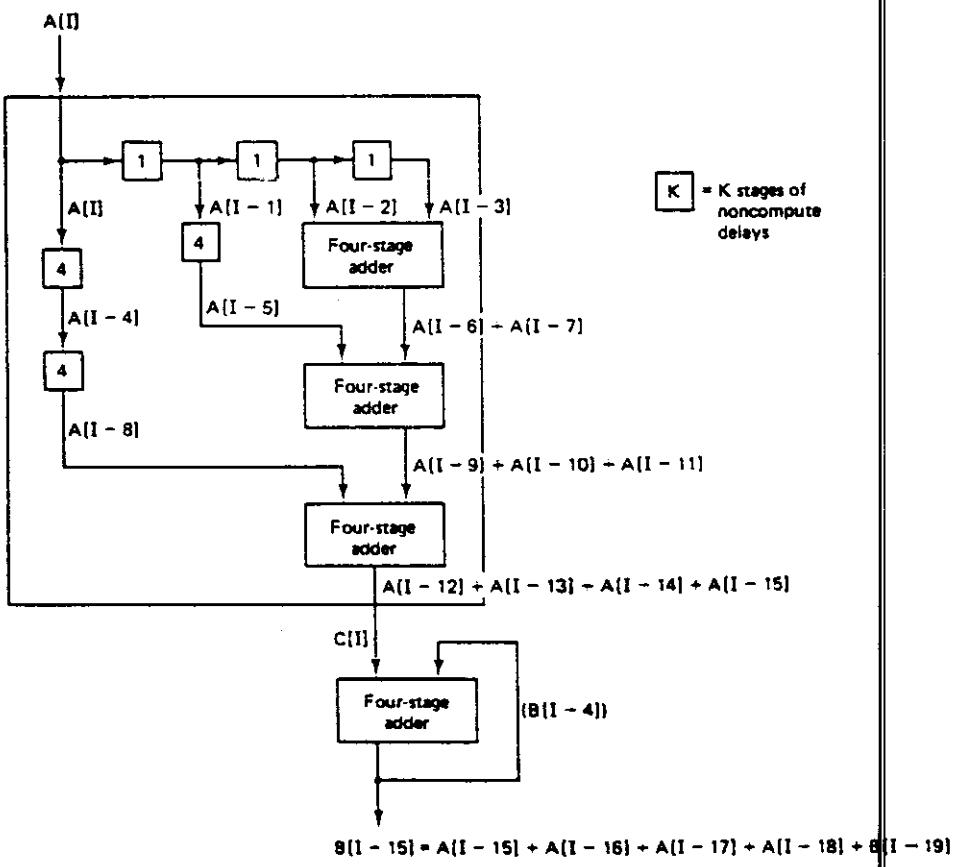
$$z_i = f(g, z_j, z_{j-1}, \ldots, z_{j-m+1}) \tag{9.4}$$

dimana j = (i - jumlah penangguhan clock di dalam pipe) dan g adalah fungsi hanya dari input j sebelumnya. Dalam hal ini, umpan balik dapat digabungkan ke dalam pipeline seperti pada Gambar 9-29.

Mari kita perhatikan suatu masalah perulangan tertentu untuk menjelaskan konsep ini. Asumsikanlah bahwa kita ingin merancang sebuah pipeline untuk menghitung:

$$B[I] = A[I] + B[I - 1] \tag{9.5}$$

untuk dua buah array A dan B, dimana $B(0)$ merupakan sembarang nilai awal. Jika setiap elemen A dan B merupakan angka 4-bit, maka kita dapat menggunakan sebuah



Gambar 9-30 Pipeline untuk fungsi perulangan yang baru

adder pipeline 4-stage ADD sebagai fungsi *f* pada Persamaan (9.4). Juga, karena ADD merupakan fungsi input *A* ke-*i* dan satu output *B* sebelumnya, maka $m = 1$.

Berdasarkan notasi Persamaan (9.3), anggaplah $x_i = A[I]$ dan $z_i = B[I]$. Dengan demikian Persamaan (9.5) dapat ditulis kembali dalam bentuk Persamaan (9.3) sebagai:

$$B[I] = \text{ADD}(A[I], B[I - 1]) \tag{9.6}$$

Karena kita sedang menggunakan adder empat-stage untuk ADD, $j = i - 4$. Maka berdasarkan Persamaan (9.4), kita dapat menulis kembali Persamaan (9.6) sebagai:

$$B[I] = \text{ADD}(C[I],B[I-4]) \tag{9.7}$$

TABEL 9.2 KOMPLEKSITAS SECARA RELATIF PADA KATEGORI-KATEGORI PIPELINE

Kompleksitas (dalam urutan naik)	Kategori Pipeline
1	Unifungsi statis skalar
2	Unifungsi statis vektor
3	Multifungsi statis skalar
4	Multifungsi statis vektor
5	Multifungsi dinamis skalar
6	Multifungsi dinamis vektor

dimana $C[I] = g(A[I],A[I-1],A[I-2],A[I-3])$ dan $g(w,x,y,z) = w+x+y+z$. Sekarang $B[I]$ sama dengan sebuah fungsi dengan bentuk empat A dan satu B $[I-4]$.

Dengan menggunakan adder empat-stage memungkinkan pengulangan baru fungsi (9.7) di-pipeline seperti diperlihatkan pada Gambar 9-30. Hubungan-hubungan dalam pipeline diberi nama untuk menunjukkan posisi input A yang relatif terhadap input ke-i. Karena tidak ada elemen ke-i yang negatif maka output tidak mempunyai arti sampai $i = 16$. Dengan demikian output pertama, $B[1]$, tidak akan tersedia sampai $A[16]$ dimasukkan, yang menghasilkan waktu startup awal 16 untuk pipe tersebut. Setelah pipe diisi, bagaimanapun juga, *throughput* akan menjadi 1.

9.9 RINGKASAN

Konsep pemrosesan pipeline dapat digunakan dalam sebuah komputer untuk memperbaiki *throughput* sistem tersebut dalam berbagai variasi cara. Tiga jenis pokok pipelining adalah pipelining **aritmatika**, **instruksi** dan **prosesor**. Peningkatan *throughput* sistem dengan satu atau lebih jenis pipelining ini tergantung pada fungsi dan harga pipelining. Harga pipeline termasuk tambahan perangkat keras yang diperlukan untuk mekanisme latch dan kendali, serta waktu yang tidak produktif bagi pengisian pipeline dan memaksa latensi untuk menghindari terjadinya tubrukan.

Pipeline dapat dikategorikan ke dalam basis **unifungsional** atau **multifungsional**, **statis** atau **dinamis**, dan **skalar** atau **vektor**. Dalam pengimplementasian dan penjadwalannya, pipeline bervariasi dari yang sangat sederhana sampai yang

sangat kompleks. Tabel 9.2 memberikan daftar kategori pipeline sesuai tingkat kompleksitas yang semakin menaik. (Perhatikan bahwa pipeline unifungsi tidak pernah berupa pipeline yang dinamis).

SOAL

1. Berikan definisi istilah-istilah berikut ini yang berhubungan dengan pipeline:
   - **(a)** penangguhan clock
   - **(b)** waktu startup awal
   - **(c)** instruksi lihat-ke-muka (*look-ahead*)
   - **(d)** men-*draining* pipe
   - **(e)** pipeline multifungsi yang statis
   - **(f)** peng-*cascade*-an stage
   - **(g)** hubungan umpanbalik (*feedback*)
   - **(h)** tabel reservasi
   - **(i)** tubrukan (*collision*)
   - **(j)** latensi
   - **(k)** MAL
   - **(l)** siklus greedy
2. Mengapa latch-latch disisipkan di antara stage-stage sebuah pipeline?
3. Perhatikan pipeline yang ditunjukkan oleh tabel reservasi berikut ini:

	t_0	t_1	t_2	t_3
S_1	X			
S_2			X	
S_3				X
S_4		X		

   - **(a)** Berapa waktu startup awal dari pipeline tersebut?
   - **(b)** Berapa maksimum *throughput* dari pipeline tersebut?
   - **(c)** Berikan diagram ruang-waktu dari keempat input pada pipeline sedemikian sehingga dicapai *throughput* yang maksimal.
4. Ulangi soal nomor 3 untuk pipeline berikut ini:

	t_0	t_1	t_2	t_3	t_4
S_1	X	X			
S_2					X
S_3				X	
S_4			X		

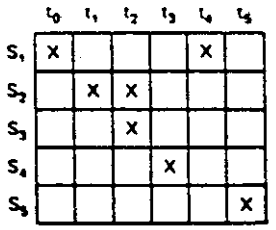
5. Mengapa pipeline yang dinamis selalu berupa pipeline multifungsi?

6. Perhatikan adder paralel 4-bit secara pipeline yang dibangun dengan meng-*cascade* stage-stage full adder.
   (a) Gambarkan konfigurasi adder pipeline tersebut.
   (b) Tunjukkan kemajuan aliran input berikut ini saat melalui pipeline tersebut.

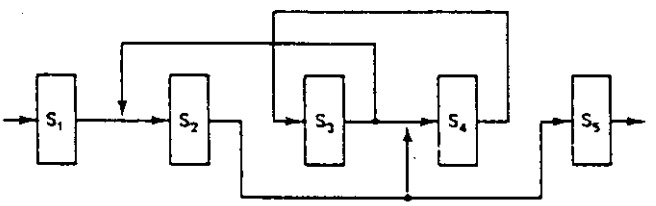
A	*B*
1101	1011
0001	1100
0111	0101
1111	0001
0010	0110

   (c) Gambarkan diagram ruang-waktu untuk pasangan angka A-B.

7. Buat diagram dua pipeline yang berbeda, yang dapat ditentukan oleh tabel reservasi berikut ini.



8. Perhatikan pipeline berikut ini.



   (a) Berikan dua tabel reservasi yang berbeda, yang dapat dibentuk dari gambar di atas.
   (b) Hitung waktu evaluasi untuk masing-masing tabel pada bagian (a).

9. Dengan menggunakan suatu diagram ruang-waktu untuk pipeline berikut ini, tunjukkan dua inisiasi berurutan yang latensinya < 5, dan (a) berpindah melalui pipe tanpa bertubrukan dan (b) yang bertubrukan.

	t_0	t_1	t_2	t_3	t_4
S_1	X		X		
S_2		X			X
S_3				X	
S_4					X

10. Berdasarkan pipeline pada Gambar 9-18(a), beri tanda apakah siklus latensi berikut ini akan bebas tubrukan (*collision-free*) atau tidak. Berikan diagram ruang-waktu untuk mendukung jawaban anda.

(a) {6} **(d)** {1, 3}
(b) {4} **(e)** {6, 1}
(c) {1, 6, 4} **(f)** {0}

11. Perhatikan pipeline yang ditunjukkan oleh tabel reservasi pada soal nomor 7.

(a) Mana kumpulan latensi yang terlarang (*forbidden set of latencies*) untuk pipeline tersebut?
(b) Apa vektor tubrukan (*collision vector*) pipeline tersebut?
(c) Berikan diagram keadaan (*state*) yang memberi karakteristik bagi inisiasi yang berdampingan pada pipeline tersebut.
(d) Berikan daftar siklus sederhana dan rata-rata latensi yang berhubungan untuk pipeline tersebut.
(e) Siklus mana yang menghasilkan MAL untuk pipeline tersebut?
(f) Berapa maksimum *throughput* untuk pipeline itu?

12. Ulangi soal nomor 11 untuk pipeline berikut ini:

	t_0	t_1	t_2	t_3	t_4
S_1			X	X	
S_2		X		X	X
S_3	X				
S_4		X			