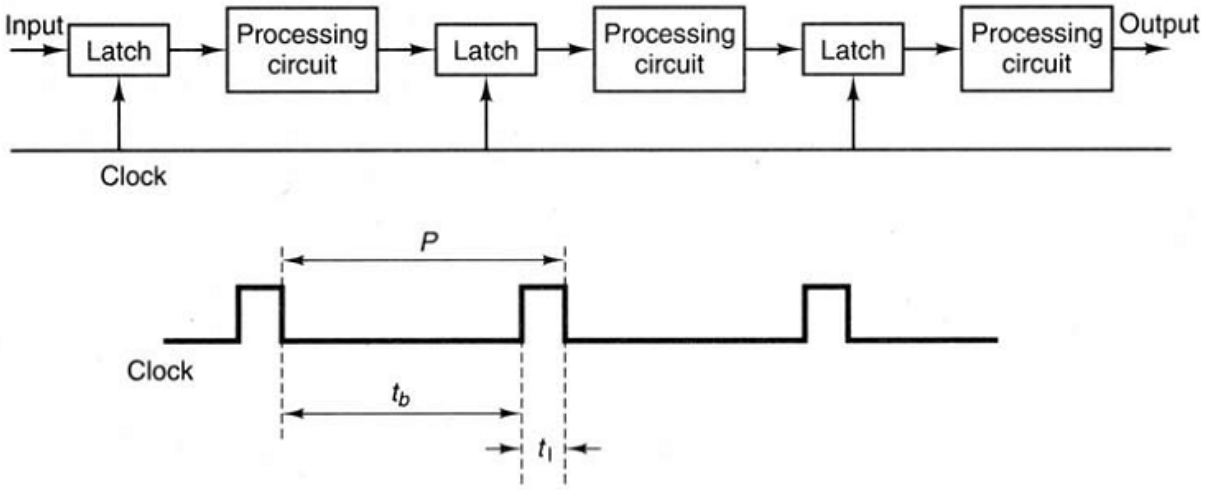
PIPELINE HAZARD

From M.R Zargham's book (Chapter 3.1)

- Pada Bab ini pembahasan akan meliputi:
  - Struktur Pipeline
  - Pengukuran Performance
  - Jenis-jenis Pipeline
  - Instruksi Pipeline
  - Aritmatika Pipeline

1. Struktur Pipeline (Pipeline Structure)

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



P: Clock period.
t_b: Maximum time for a stage to perform a function.
t_l: Time for latch to accept input data.

2. Pengukuran Performansi (Pipeline Performance Measurements)

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

$$T_{pipe} = m * P + (n-1) * P, \qquad (3.1)$$

  - n= jumlah input task
  - m= jumlah stadium pipeline
  - P= Perioda Clock

- Waktu lengkap melakukan proses yang diperlukan untuk non-pipeline adalah:

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

  - τi adalah waktu tunggu (delay time) untuk masing-masing stadium.
- Untuk kasus ideal $\tau_i = \tau$ untuk i=1 hingga m, sehingga τseq dapat ditulis:

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

- Jika kita mengabaikan waktu yang dibutuhkan untuk latch untuk melakukan penyimpanan (cth: $t_1 = 0$), maka:

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

KINERJA PIPELINE

- Jika prosesor adalah non-pipeline, maka waktu yang digunakan untuk m instruksi adalam nmtc dengan menganggap waktu siklus instruksi sama dengan ntc
- Gain Kinerja (Speed-up) pipeline adalah waktu yang digunakan dalam mode non-pipeline dibagi waktu yang digunakan dalam mode pipeline, dapat dituliskan:

$$\text{Speed - up} = \frac{\text{waktu mode non - pipeline}}{\text{waktu mode pipeline}} = \frac{nmt_c}{(n+m-1)t_c} = \frac{nm}{n+m-1}$$

Untuk nilai $m >> (n - 1)$, maka $(n + m - 1)$ mendekati m. Karena itu speed-up = $nm/m = n$

Jadi, speed-up maksimum secara teoritis adalah sama dengan jumlah tingkat dalam pipeline .

KINERJA PIPELINE

- Dua faktor yang tidak kalah penting yang sering digunakan untuk menentukan performansi sebuah pipeline adalah efficiency dan throughput

$$\textit{Throughput} = \textit{jumlah tugas yang dieksekusi per unit waktu} = \frac{m}{(n+m-1)t_c}$$

$$\textit{Efisiensi} = \textit{rasio antara speed - up aktual dan speed - up maks.} = \frac{\text{speed} - \text{up}}{n} = \frac{m}{n+m-1}$$

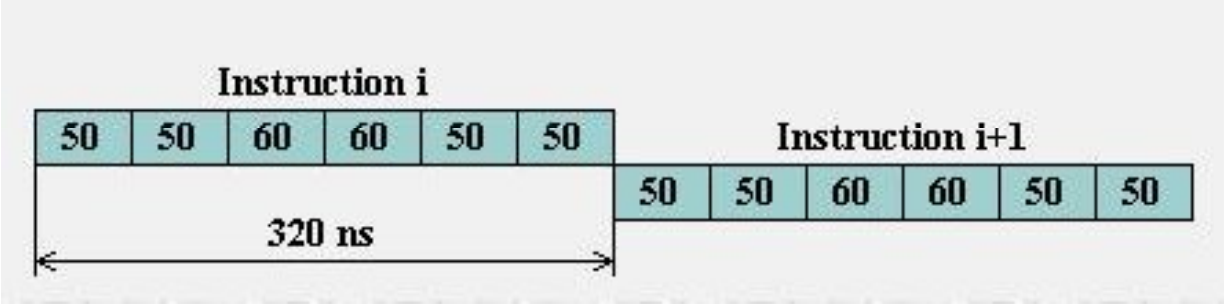
1. Consider a non-pipelined machine with 6 execution stages of lengths 50 ns, 50 ns, 60 ns, 60 ns, 50 ns, and 50 ns.
   - Find the instruction latency on this machine.
   - How much time does it take to execute 100 instructions?

   Solution:

2. Suppose we introduce pipelining on this machine. Assume that when introducing pipelining, the clock skew adds 5ns of overhead to each execution stage.
   - What is the instruction latency on the pipelined machine?
   - How much time does it take to execute 100 instructions?

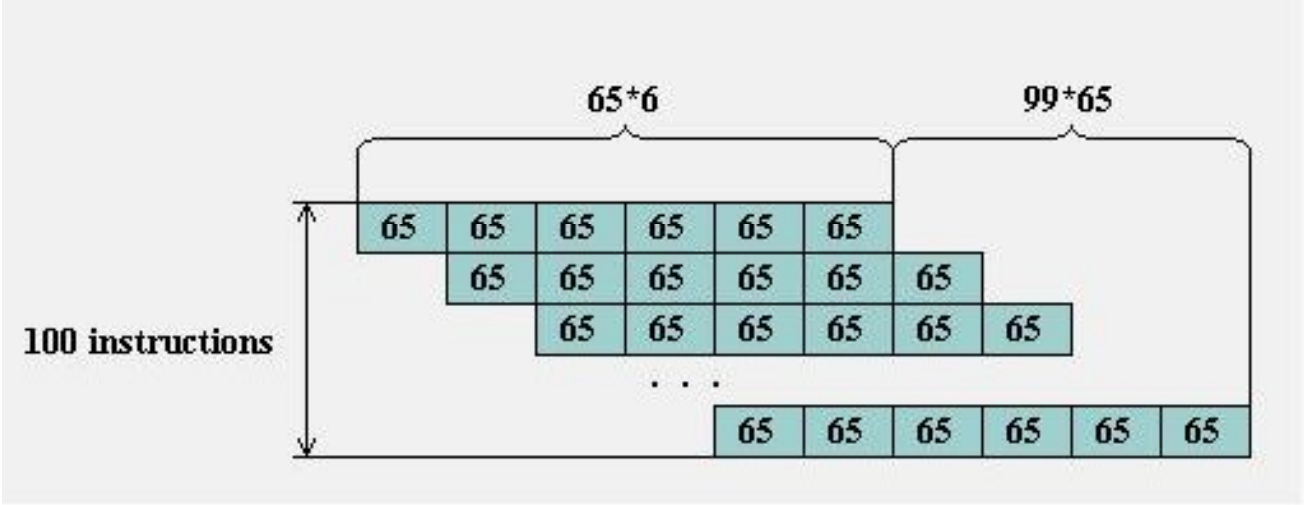
   Solution:

- *Instruction latency = 50+50+60+60+50+50= 320 ns*
  *Time to execute 100 instructions = 100*320 = 32000 ns*



-

- *Remember that in the pipelined implementation, the length of the pipe stages must all be the same, i.e., the speed of the slowest stage plus overhead. With 5ns overhead it comes to:*
- *The length of pipelined stage = MAX (lengths of unpipelined stages) + overhead = 60 + 5 = 65 ns*
  Instruction latency = 65 ns
  *Time to execute 100 instructions = 65*6*1 + 65*1*99 = 390 + 6435 = 6825 ns*


65*6
99*65
100 instructions
65 65 65 65 65 65
65 65 65 65 65 65
65 65 65 65 65 65
. . .
65 65 65 65 65 65

3. Jenis-jenis Pipeline

- Pipeline biasanya dibagi dua kelas:
  - Pipeline Instruksi (Instruction Pipeline)
  - Pipeline Aritmatika (Arithmetic Pipeline)
- Dalam setiap kelas pipeline dapat didesain dengan 2 cara:
  - Statis atau Dinamis
    - Sebuah pipeline statis hanya melakukan satu operasi (seperti operasi penjumlahan atau operasi perkalian) pada satu waktu.
    - Sebuah pipeline dinamis dapat melakukan lebih dari satu operasi pada satu waktu.

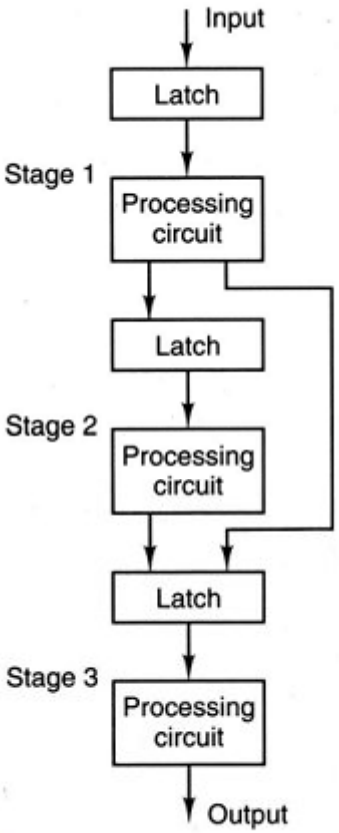


Figure 3.2 A three-stage dynamic pipeline.

- Untuk melakukan proses perkalian, input data harus melalui stadium 1, 2 dan 3; Untuk melakukan penambahan, data hanya cukup melalui stadium 1 dan 3 saja. Oleh karena itu, stadium pertama dari proses penambahan dapat dilakukan pada sebuah data input D1 di stadium 1, sementara pada waktu yang bersamaan, stadium akhir dari proses perkalian dilakukan di stadium 3 pada sebuah input data yang berbeda D2. Perhatikan bahwa, interval waktu antara inisiasi input D1 dan D2 ke pipeline harus diupayakan agar merka tidak meraih stadium 3 secara bertepatan atau pada waktu yag sama, untuk menghindari terjadinya *collision*.

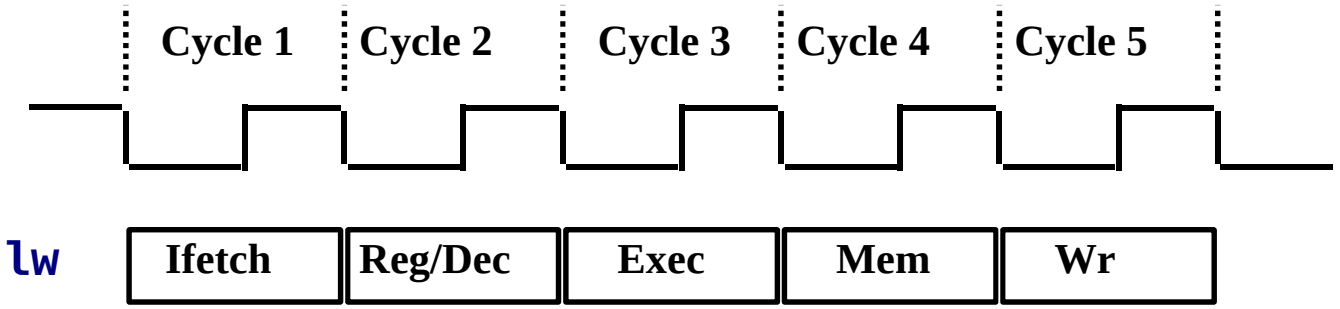
3.1. Instruksi Pipeline (From M.R Zargham's book Chapter 3.2))

In a von Neumann architecture, the process of executing an instruction involves several steps. First, the control unit of a processor fetches the instruction from the cache (or from memory). Then the control unit decodes the instruction to determine the type of operation to be performed. When the operation requires operands, the control unit also determines the address of each operand and fetches them from cache (or memory). Next, the operation is performed on the operands and, finally, the result is stored in the specified location.

An instruction pipeline increases the performance of a processor by overlapping the processing of several different instructions. Often, this is done by dividing the instruction execution process into several stages. As shown in Figure 3.3, an instruction pipeline often consists of five stages, as follows:

1. **Instruction fetch (IF)**. Retrieval of instructions from cache (or main memory).
2. **Instruction decoding (ID)**. Identification of the operation to be performed.
3. **Operand fetch (OF)**. Decoding and retrieval of any required operands.
4. **Execution (EX)**. Performing the operation on the operands.
5. **Write-back (WB)**. Updating the destination operands.

The Five Stages of An Instruction



- **Ifetch: Instruction Fetch**
  - **Fetch the instruction from the Instruction Memory**
- **Reg/Dec: Registers Fetch and Instruction Decode**
- **Exec: Calculate the memory address**
- **Mem: Read the data from the Data Memory**
- **Wr: Write the data back to the register file**

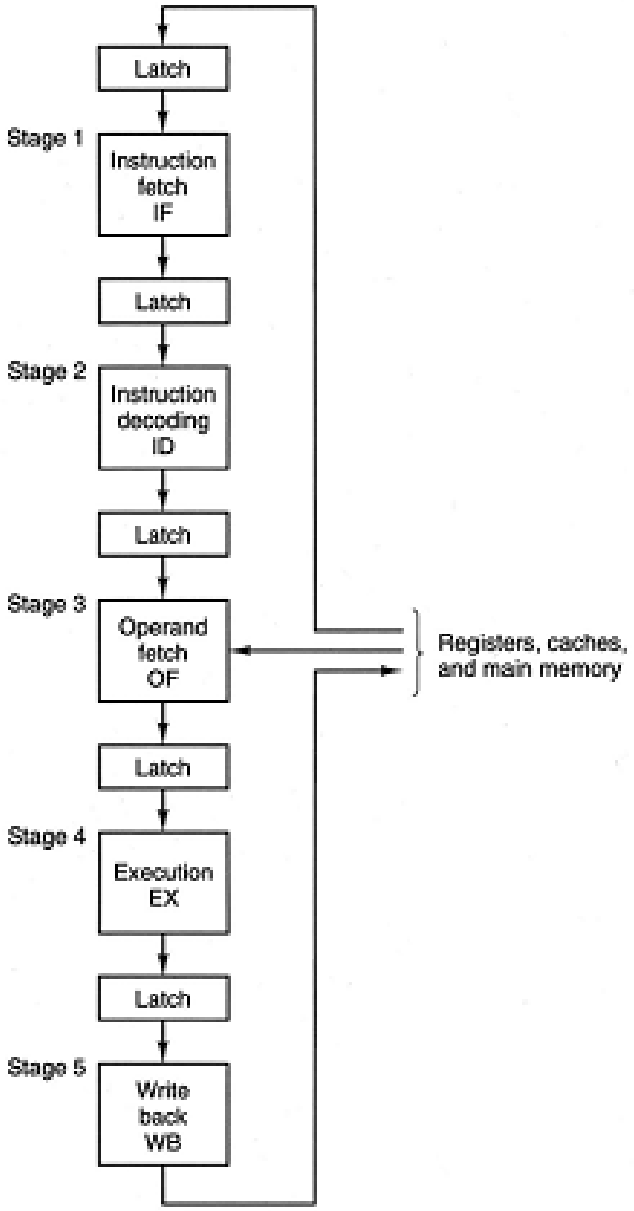


Figure 3.3 Stages of an instruction pipeline.

An instruction pipeline overlaps the process of the preceding stages for different instructions to achieve a much lower total completion time, on average, for a series of instructions. As an example, consider Figure 3.4, which shows the execution of four instructions in an instruction pipeline. During the first cycle, or clock pulse, instruction i_1 is fetched from memory. Within the second cycle, instruction i_1 is decoded while instruction i_2 is fetched. This process continues until all the instructions are executed. The last instruction finishes the write-back stage after the eighth clock cycle. Therefore, it takes 80 nanoseconds (ns) to complete execution of all the four instructions when assuming the clock period to be 10 ns. The total completion time can also be obtained using equation (3.1); that is,

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

Note that in a nonpipelined design the completion time will be much higher. Using equation (3.2),

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

- Dengan penerapan pipeline ini pada microprocessor akan didapatkan peningkatan dalam unjuk kerja microprocessor. Hal ini terjadi karena beberapa instruksi dapat dilakukan secara parallel dalam waktu yang bersamaan. Secara kasarnya diharapkan akan didapatkan peningkatan sebesar K kali dibandingkan dengan microprocessor yang tidak menggunakan pipeline, apabila tahapan yang ada dalam satu kali pemrosesan instruksi adalah K tahap.

- Teknologi pipeline yang digunakan pada komputer bertujuan untuk meningkatkan kinerja dari komputer. Secara sederhana, pipeline adalah suatu cara yang digunakan untuk melakukan sejumlah kerja secara bersamaan tetapi dalam tahap yang berbeda yang dialirkan secara kontiniu pada unit pemrosesan. Dengan cara ini, maka unit pemroses selalu bekerja.

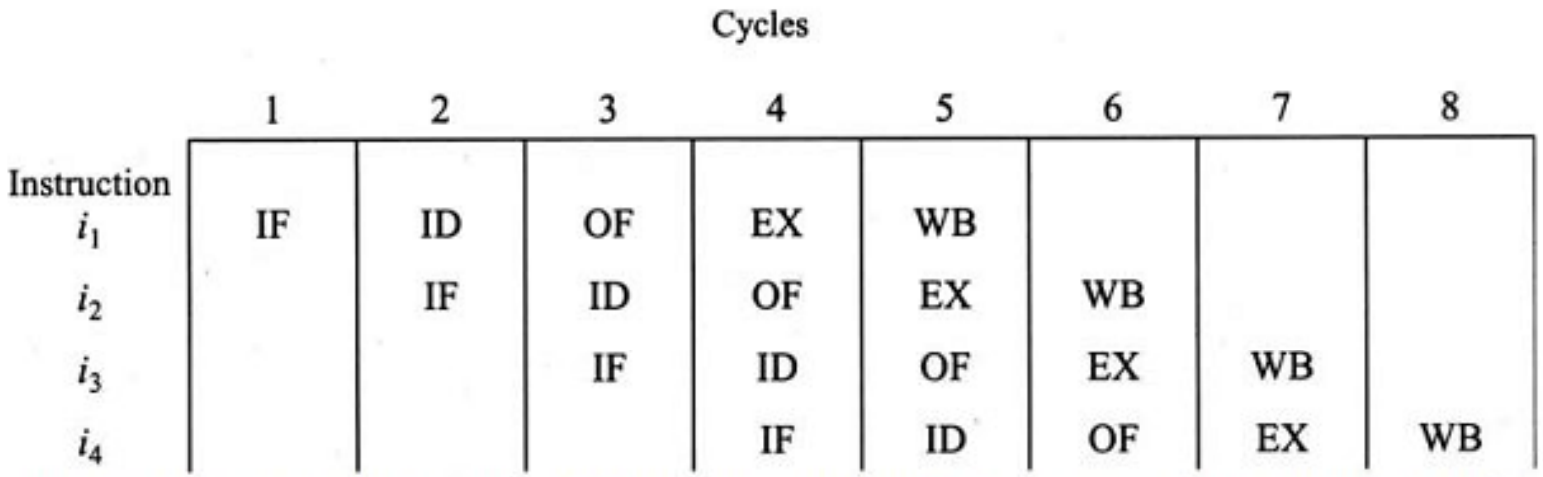


Figure 3.4 Execution cycles of four consecutive instructions in an instruction pipeline.

- Teknik pipeline ini dapat diterapkan pada berbagai tingkatan dalam sistem komputer. Bisa pada level yang tinggi, misalnya program aplikasi, sampai pada tingkat yang rendah, seperti pada instruksi yang dijalankan oleh microprocessor.
- Teknik pipeline yang diterapkan pada microprocessor, dapat dikatakan sebuah arsitektur khusus. Ada perbedaan khusus antara model microprocessor yang tidak menggunakan arsitektur pipeline dengan microprocessor yang menerapkan teknik ini.

- Teknik pipeline ini menyebabkan ada sejumlah hal yang harus diperhatikan sehingga ketika diterapkan dapat berjalan dengan baik.
- Tiga kesulitan yang sering dihadapi ketika menggunakan teknik pipeline ini adalah : Terjadinya penggunaan resource yang bersamaan, Ketergantungan terhadap data, Pengaturan Jump ke suatu lokasi memori.

- Karena beberapa instruksi diproses secara bersamaan ada kemungkinan instruksi tersebut sama-sama memerlukan resource yang sama, sehingga diperlukan adanya pengaturan yang tepat agar proses tetap berjalan dengan benar. (Structural Hazard)
- Sedangkan ketergantungan terhadap data, bisa muncul, misalnya instruksi yang berurutan memerlukan data dari instruksi yang sebelumnya. (Data Hazard)
- Kasus Jump, juga perlu perhatian, karena ketika sebuah instruksi meminta untuk melompat ke suatu lokasi memori tertentu, akan terjadi perubahan program counter, sedangkan instruksi yang sedang berada dalam salah satu tahap proses yang berikutnya mungkin tidak mengharapkan terjadinya perubahan program counter. (Control Hazard)

From Zargham's Book

The issuing problem. If an instruction is available, but cannot be executed for some reason, a hazard exists for that instruction. These hazards create issuing problems; they prevent issuing an instruction for execution. Three types of hazard are discussed here. They are called *structural hazard, data hazard,* and *control hazard*. A structural hazard refers to a situation in which a required resource is not available (or is busy) for executing an instruction. A data hazard refers to a situation in which there exists a data dependency (operand conflict) with a prior instruction. A control hazard refers to a situation in which an instruction, such as branch, causes a change in the program flow. Each of these hazards is explained next.

1. Structural Hazard

Structural Hazard. A structural hazard occurs as a result of resource conflicts between instructions. One type of structural hazard that may occur is due to the design of execution units. If an execution unit that requires more than one clock cycle (such as multiply) is not fully pipelined or is not replicated, then a sequence of instructions that uses the unit cannot be subsequently (one per clock cycle) issued for execution. Replicating and/or pipelining execution units increases the number of instructions that can be issued simultaneously. Another type of structural hazard that may occur is due to the design of register files. If a register file does not have multiple write (read) ports, multiple writes (reads) to (from) registers cannot be performed simultaneously. For example, under certain situations the instruction pipeline might want to perform two register writes in a clock cycle. This may not be possible when the register file has only one write port.

The effect of a structural hazard can be reduced fairly simply by implementing multiple execution units and using register files with multiple input/output ports.

2. Data Hazard

Data Hazard. In a nonpipelined processor, the instructions are executed one by one, and the execution of an instruction is completed before the next instruction is started. In this way, the instructions are executed in the same order as the program. However, this may not be true in a pipelined processor, where instruction executions are overlapped. An instruction may be started and completed before the previous instruction is completed. The data hazard, which is also referred to as the data dependency problem, comes about as a result of overlapping (or changing the order of) the execution of data-dependent instructions. For example, in Figure 3.5 instruction i_2 has a data dependency on i_1 because it uses the result of i_1 (i.e., the contents of register R_2) as input data. If the instructions were sent to a pipeline in the normal manner, i_2 would be in the OF stage before i_1 passed through the WB stage. This would result in using the old contents of R_2 for computing a new value for R_5, leading to an invalid result. To have a valid result, i_2 must not enter the OF stage until i_1 has passed through the WB stage. In this way, as is shown in Figure 3.6, the execution of i_2 will be delayed for two clock cycles. In other words, instruction i_2 is said to be *stalled* for two clock cycles. Often, when an instruction is stalled, the instructions that are positioned after the stalled instruction will also be stalled. However, the instructions before the stalled instruction can continue execution.

$i1$	Add	R_2,	R_3,	R_4	-- $R_2=R_3+R_4$
i_2	Add	R_5,	R_2,	R_1	-- $R_5=R_2+R_1$

Data Hazard

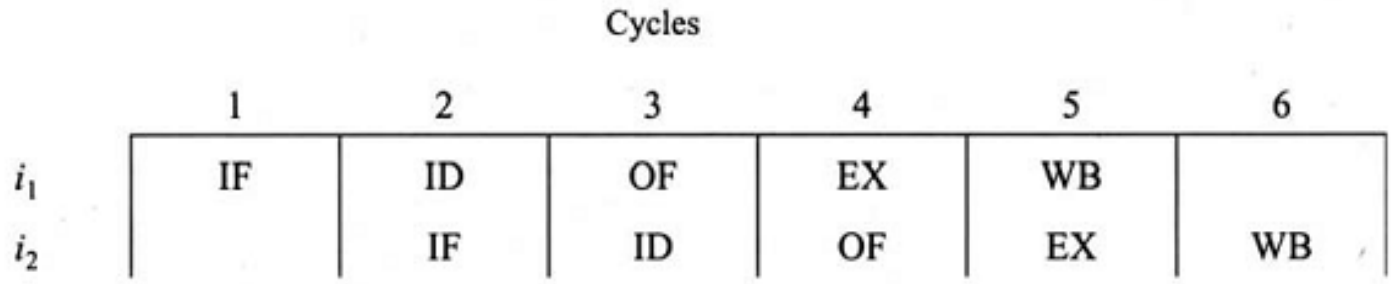


Figure 3.5 Instruction i_2 has data dependency on i_1.

Tipe Data Hazard

- Read After Write (RAW)
- Write After Read (WAR)
- Write After Write (WAW)

Read After Write (RAW)

RAW: This type of data hazard was discussed previously; it refers to the situation in which i_2 reads a data source before i_1 writes to it. This may produce an invalid result since the read must be performed after the write in order to obtain a valid result. For example, in the sequence

i_1:	Add	R_2,	R_3,	R_4	--R_2=R_3+R_4
i_2:	Add	R_5,	R_2,	R_1	--R_5=R_2+R_1

an invalid result may be produced if i_2 reads R_2 before i_1 writes to it.

- i_2 mencoba untuk membaca sumber sebelum i_1 menulisnya, sehingga i_2 mendapat nilai lama yang tidak benar
- Ini adalah jenis hazard data yang paling umum dan kita menggunakan teknik forwarding untuk mengatasinya.
- Teknik forwarding adalah dengan menambahkan jalur balik data dari hasil operasi ALU langsung kepada sumber daya yang dibutuhkan

Write After Read (WAR)

WAR: This refers to the situation in which i_2 writes to a location before i_1 reads it. For example, in the sequence

i_1:	Add	R_2,	R_3,	R_4	--R_2=R_3+R_4
i_2:	Add	R_4,	R_5,	R_6	--R_4=R_5+R_6

an invalid result may be produced if i_2 writes to R_4 before i_1 reads it; that is, the instruction i_1 might use the wrong value of R_4.

- i_2 mencoba menulis hasil tujuan sebelum dibaca oleh i_1, sehingga i_1 memperoleh nilai baru dari R_4 yang salah
- Ini tidak akan terjadi pada contoh pipeline karena semua pembacaan adalah pada tahap awal (ID) dan semua penulisan pada tahap akhir (WB)
- Hazard ini terjasi ketika terdapat instruksi-instruksi yang menulis pada tahap awal pipeline, dan instruksi lain yang membaca sumber pada bagian akhir pipeline.

Write After Write (WAW)

WAW: This refers to the situation in which i_2 writes to a location before i_1 writes to it. For example, in the sequence

i_1:	Add	R_2,	R_3,	R_4	--R_2=R_3+R_4
i_2:	Add	R_2,	R_5,	R_6	--R_2=R_5+R_6

the value of R_2 is recomputed by i_2. If the order of execution were reversed, that is, i_2 writes to R_2 before i_1 writes to it, an invalid value for R_2 might be produced.

- i_2 mencoba untuk menulis operand sebelum ditulis oleh i_1. penulisan berakhir dilaksanakan dengan urutan yang salah, meninggalkan nilai yang ditulis oleh i_1 bukan nilai yang ditulis oleh i_2.
- Hazard ini hanya terjadi dalam pipeline yang menulis pada lebih satu tahap pipeline.

PIPELINE STALL

- Resiko dalam *pipeline* mengakibatkan *pipeline* harus melakukan langkah lain agar tetap berjalan. Ini diistilahkan sebagai *pipeline stall*.
- Dalam mengatasi hazard seringkali memerlukan beberapa instruksi dalam *pipeline* diizinkan untuk diproses sementara instruksi lain menunggu. Pada saat sebuah instruksi dihentikan dulu pemrosesannya maka semua instruksi yang keluar sesudah instruksi tersebut dihentikan sementara juga. Instruksi yang muncul sebelum instruksi tersebut harus terus di- *pipeline*. Selama kondisi ini tidak ada instruksi baru yang diambil dan dieksekusi menunggu sistem normal kembali.

Menangani Data Hazard

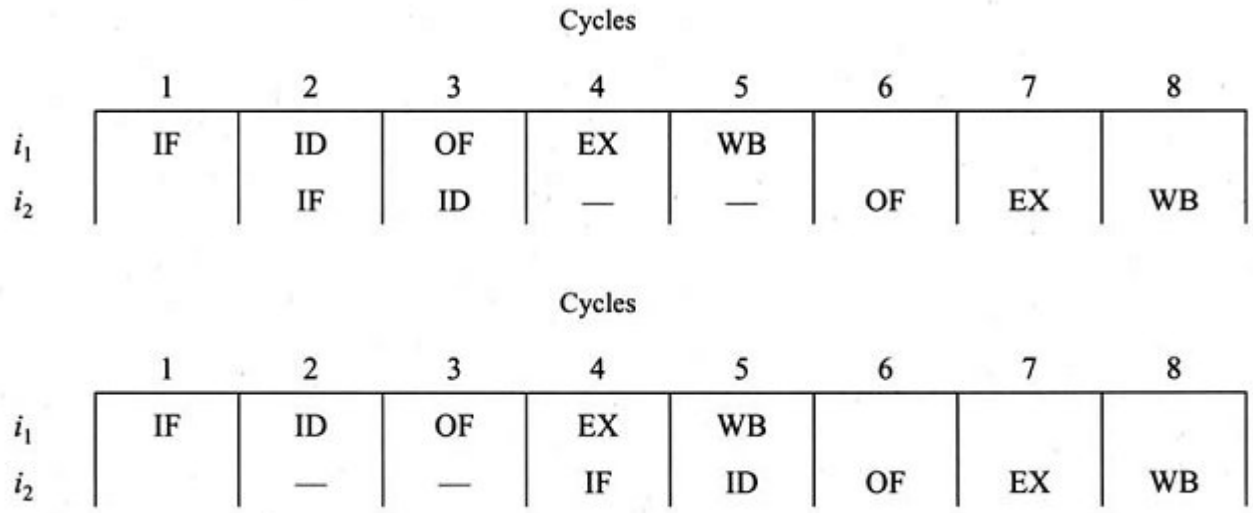


Figure 3.6 Two ways of executing data dependent instructions.

- Kinerja *pipeline* dengan adanya *pipeline stall*. *Pipeline stall* menyebabkan kinerja *pipeline* menurun dari kinerja idealnya.
- Perlu diingat bahwa *pipeline* dapat berupa penurunan CPI atau waktu *clock cycle*

Penjadwalan Instruksi -Menangani Data Hazard

- Daripada melakukan pipeline Stall, compiler mencoba untuk menjadwal urutan instruksi dengan menyusun ulang urutan kode untuk menghilangkan hazard
- Contoh, kompilator dapat mencoba untuk mencegah adanya kode *load* yang diikuti oleh penggunaan langsung register tujuan *load*.

i_1	Add	R_2,	R_3,	R_4	-- $R_2=R_3+R_4$
i_2	Add	R_5,	R_2,	R_1	-- $R_5=R_2+R_1$
i_3	Add	R_6,	R_6,	R_7	-- $R_6=R_6+R_7$
i_4	Add	R_8,	R_8,	R_7	-- $R_8=R_8+R_7$

Cycles

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

Latihan

- Buatlah kode yang mencegah pipeline stall untuk urutan perintah berikut:

  a = b+c;

  d = e-b;

3. Control Hazard

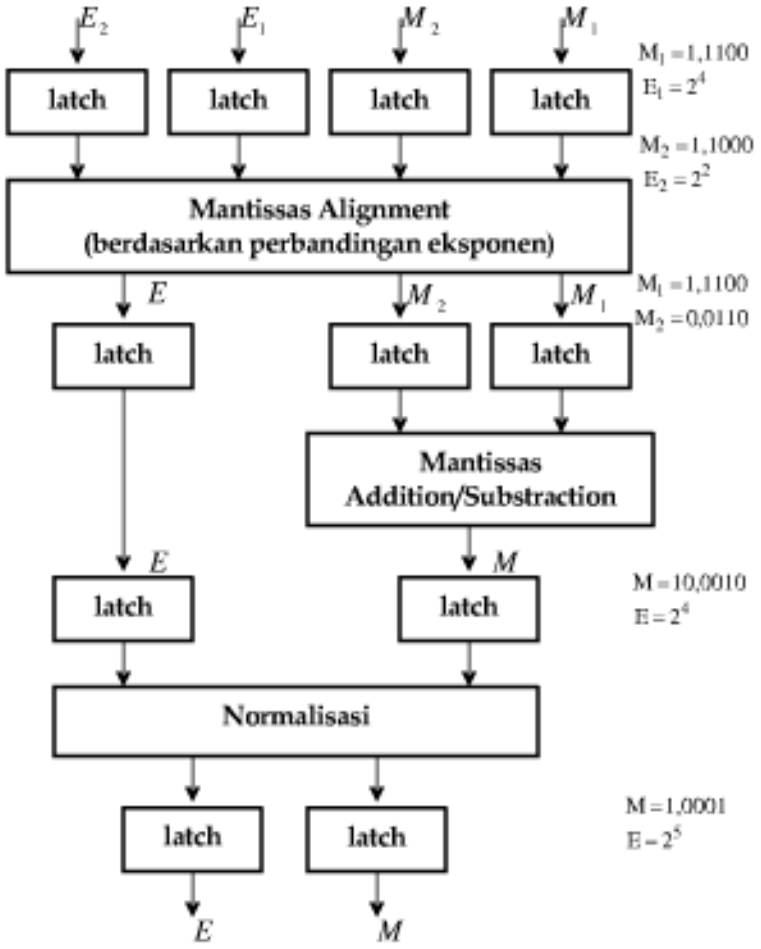
- Control Hazard dapat menyebabkan penurunan kinerja yang lebih besar daripada Data Hazard.
- Pada saat operasi pencabangan dieksekusi, ada kemungkinan PC diubah ke suatu nilai lain alamat instruksi yang dituju.

3.2. Aritmatika Pipeline

- Teknologi pipeline dapat diterapkan ke berbagai fungsi unit logikal aritmetik, dengan maksud untuk memaksimalkan performansi.
- Pipeline aritmetik digunakan untuk pengimplementasian fungsi aritmetik kompleks seperti penambahan, perkalian, dan pembagian bilangan floating point. Fungsi-fungsi tersebut dapat diatur ke berbagai sub fungsi yang berkaitan.

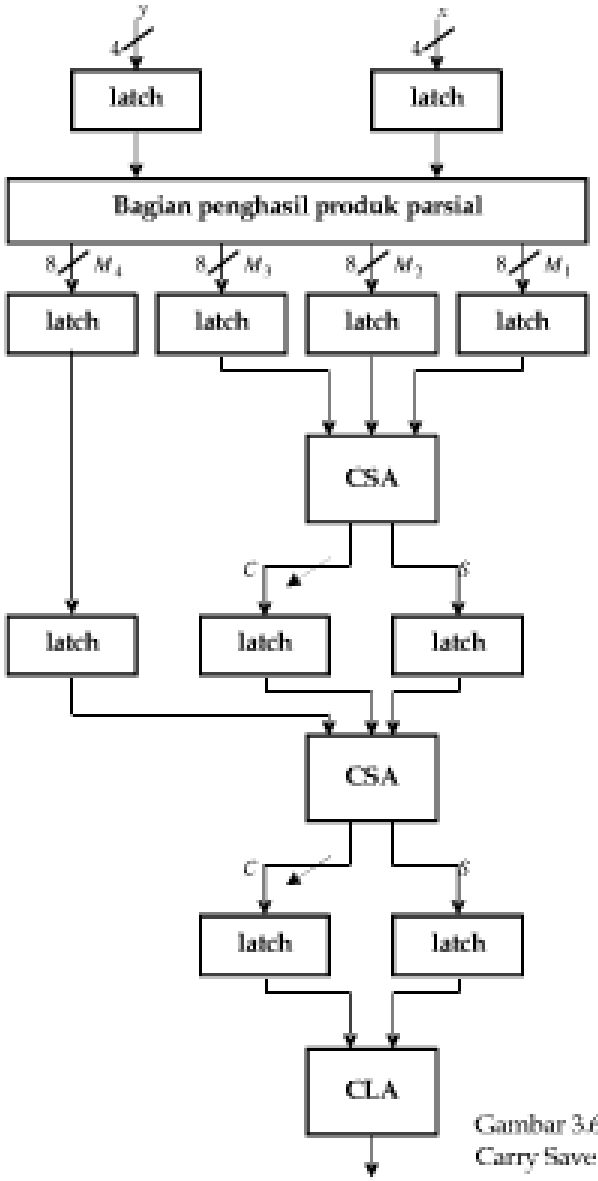
- Pada stadium pertama, mantisa M1 dan M2 dijajarkan berdasarkan selisih eksponen E1 dan E2. Jika |E1-E2|=k>0, maka mantisa dengan eksponen terkecil digeser ke kanan sebanyak *k* digit.
- Pada stadium kedua mantisa selanjutnya ditambah (atau dikurangi).
- Pada stadium ketiga, hasil akan dinormalisasi agar mantisa akhir memiliki digit yang tidak nol sesudah nilai fraksi.

Gambar berikut merepresentasikan sebuah arsitektur pipeline untuk penambahan dua bilangan floating point. Penambahan floating point dapat dibagi kedalam tiga stadium: mantissas alignment (penjajaran mantisa), penambahan mantisa (mantissas addition) dan normalisasi.



Gambar 3.5 Sebuah pipeline fungsi penambahan floating-point

- Pada stadium pertama, mantisa M1 dan M2 dijajarkan berdasarkan selisih eksponen E1 dan E2. Jika |E1-E2|=k>0, maka mantisa dengan eksponen terkecil digeser ke kanan sebanyak *k* digit.
- Pada stadium kedua mantisa selanjutnya ditambah (atau dikurangi).
- Pada stadium ketiga, hasil akan dinormalisasi agar mantisa akhir memiliki digit yang tidak nol sesudah nilai fraksi.



Gambar 3.6 Sebuah pipeline Carry Save Multiplier

- Contoh pipeline aritmetik lainnya diperlihatkan pada gambar disamping.
- Gambar merepresentasikan arsitektur pipeline untuk perkalian dua bilangan 4-bit bertipe unsigned menggunakan CSA (Carry Save Address). Stadium pertama menghasilkan produk parsial M1, M2, M3 dan M4.
- Gambar disamping merepresentasikan bagaimana M1 dihasilkan , sisa produk parsial juga dihasilkan dengan cara yang sama. M1, M2, M3, dan M4 ditambahkan secara bersamaan melalui dua stadium CSA (Carry Save Address) dan stadium CLA (Carry Lookahead Adder)

- Dengan menerapkan teknik pipeline ini, akan ditemukan sejumlah perhatian yang khusus terhadap beberapa hal di atas, tetapi tetap akan menghasilkan peningkatan yang berarti dalam kinerja microprocessor. Ada kasus tertentu yang memang sangat tepat bila memanfaatkan pipeline ini, dan juga ada kasus lain yang mungkin tidak tepat bila menggunakan teknologi pipeline.
- Example implementations
  - Pentium
  - PowerPC
  - SPARC
  - MIPS