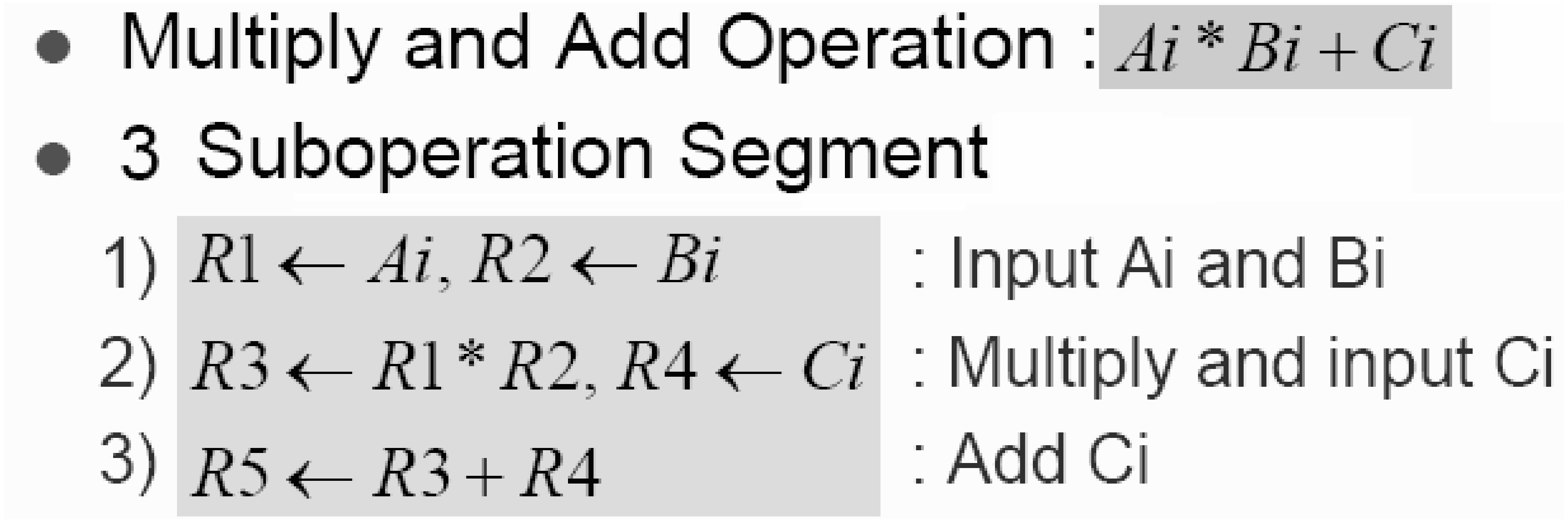
Advanced Computer Architecture
(0630561)

Lecture 4

Pipelining Processing

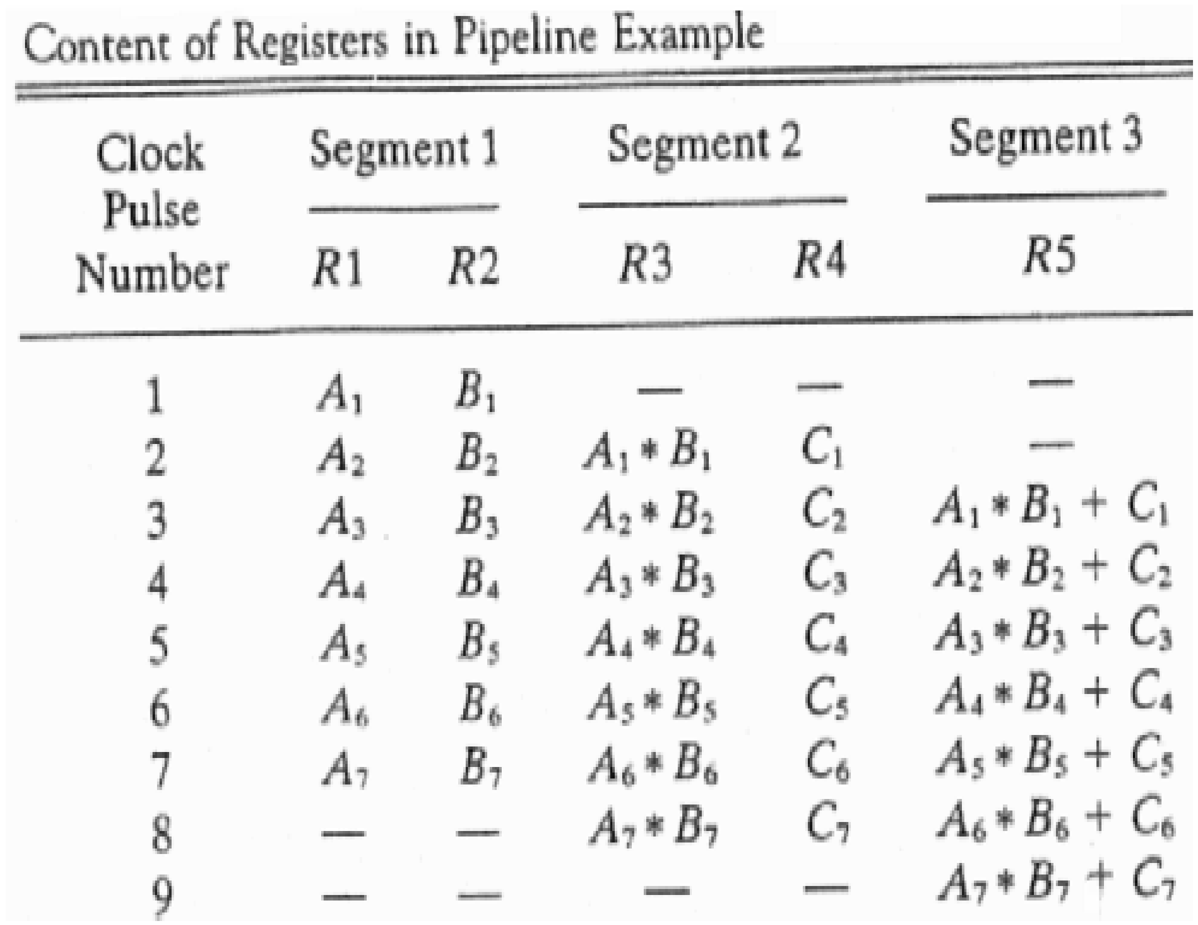
Prof. Kasim M. Al-Aubidy
Computer Eng. Dept.

- A **pipeline** is a set of data processing elements connected in series, so that the output of one element is the input of the next one.
- The pipeline organization can be demonstrated by this simple example:



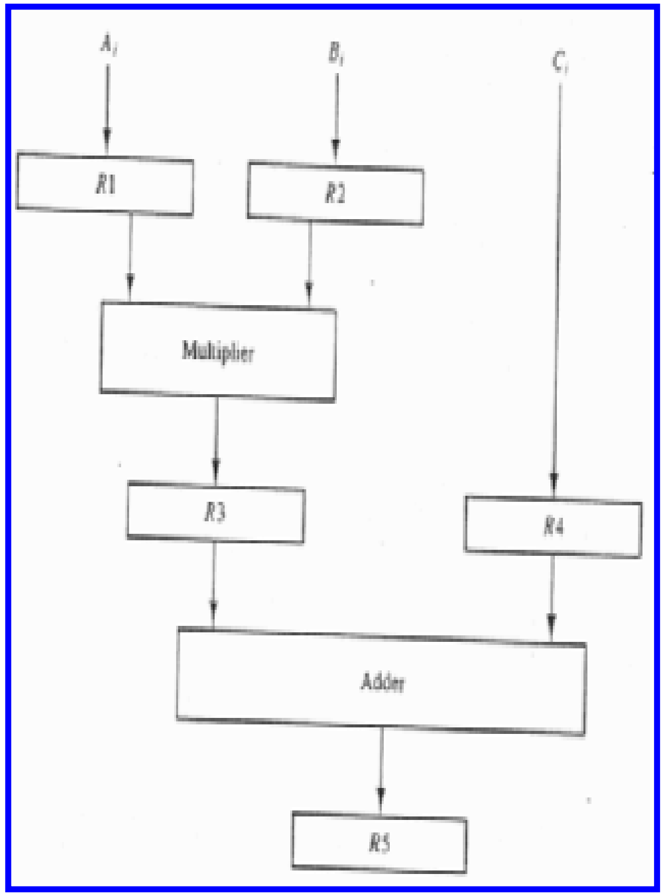
- Multiply and Add Operation : $Ai * Bi + Ci$
- 3 Suboperation Segment

1) $R1 \leftarrow Ai, R2 \leftarrow Bi$: Input Ai and Bi
2) $R3 \leftarrow R1 * R2, R4 \leftarrow Ci$: Multiply and input Ci
3) $R5 \leftarrow R3 + R4$: Add Ci



Content of Registers in Pipeline Example

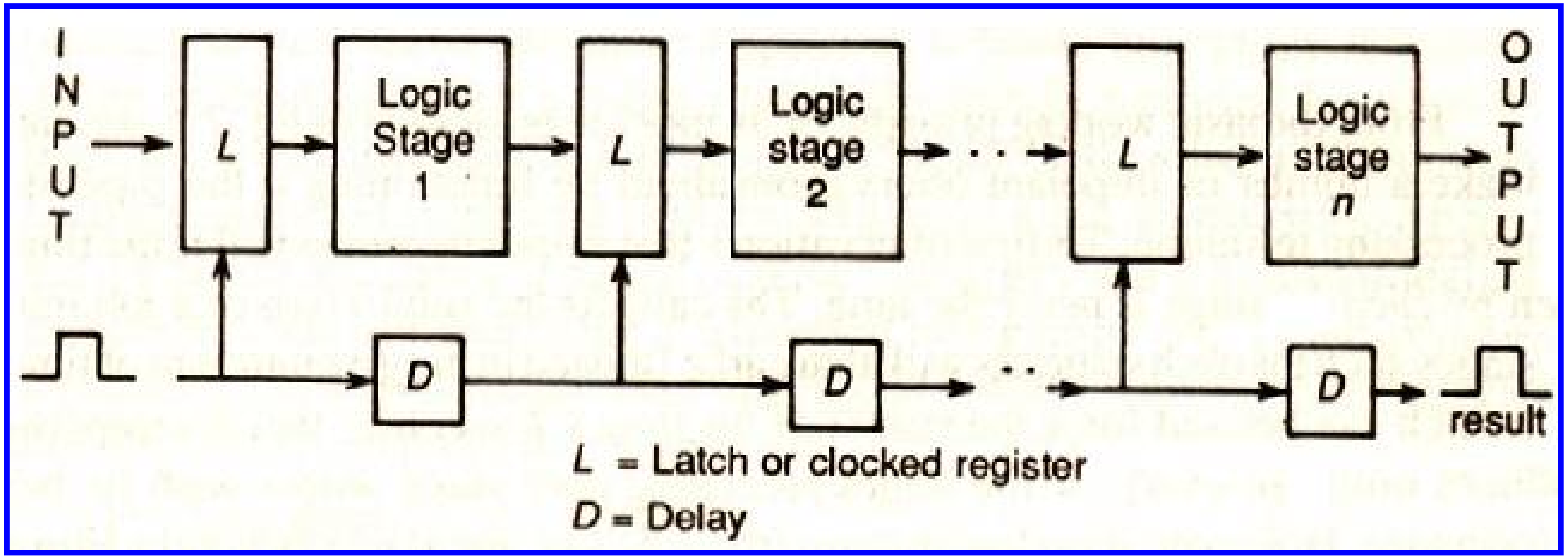
Clock Pulse Number	Segment 1		Segment 2		Segment 3
	$R1$	$R2$	$R3$	$R4$	$R5$
1	A_1	B_1	—	—	—
2	A_2	B_2	$A_1 * B_1$	C_1	—
3	A_3	B_3	$A_2 * B_2$	C_2	$A_1 * B_1 + C_1$
4	A_4	B_4	$A_3 * B_3$	C_3	$A_2 * B_2 + C_2$
5	A_5	B_5	$A_4 * B_4$	C_4	$A_3 * B_3 + C_3$
6	A_6	B_6	$A_5 * B_5$	C_5	$A_4 * B_4 + C_4$
7	A_7	B_7	$A_6 * B_6$	C_6	$A_5 * B_5 + C_5$
8	—	—	$A_7 * B_7$	C_7	$A_6 * B_6 + C_6$
9	—	—	—	—	$A_7 * B_7 + C_7$

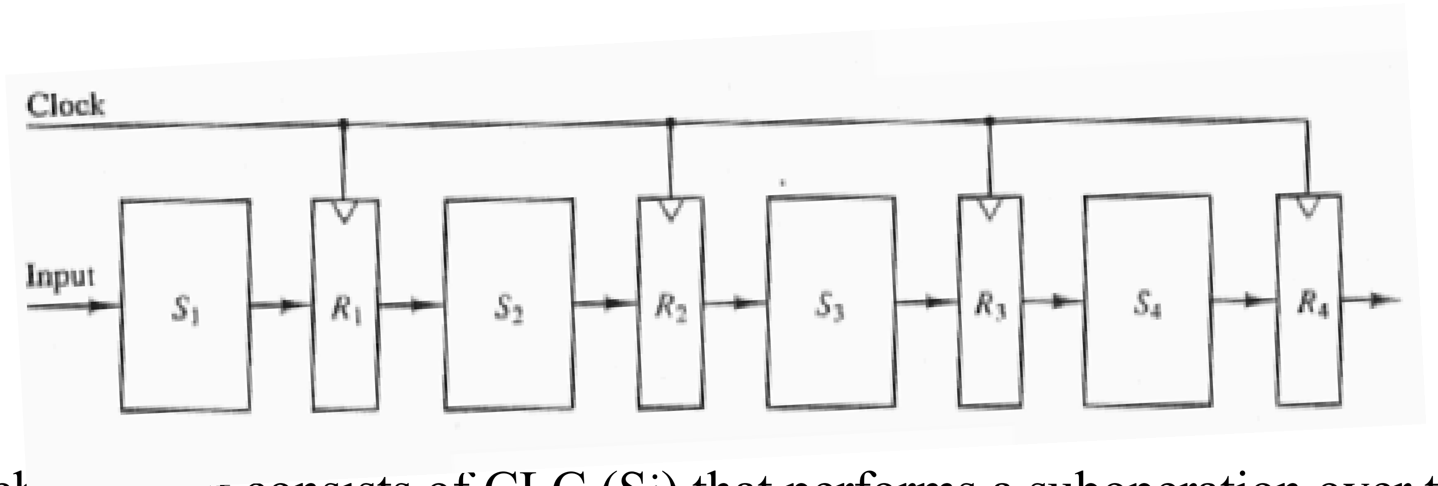


Each clock produces new output and moves the data one step down the pipeline. When no more input data are available, the clock must continue until the last output emerges out of the pipeline.

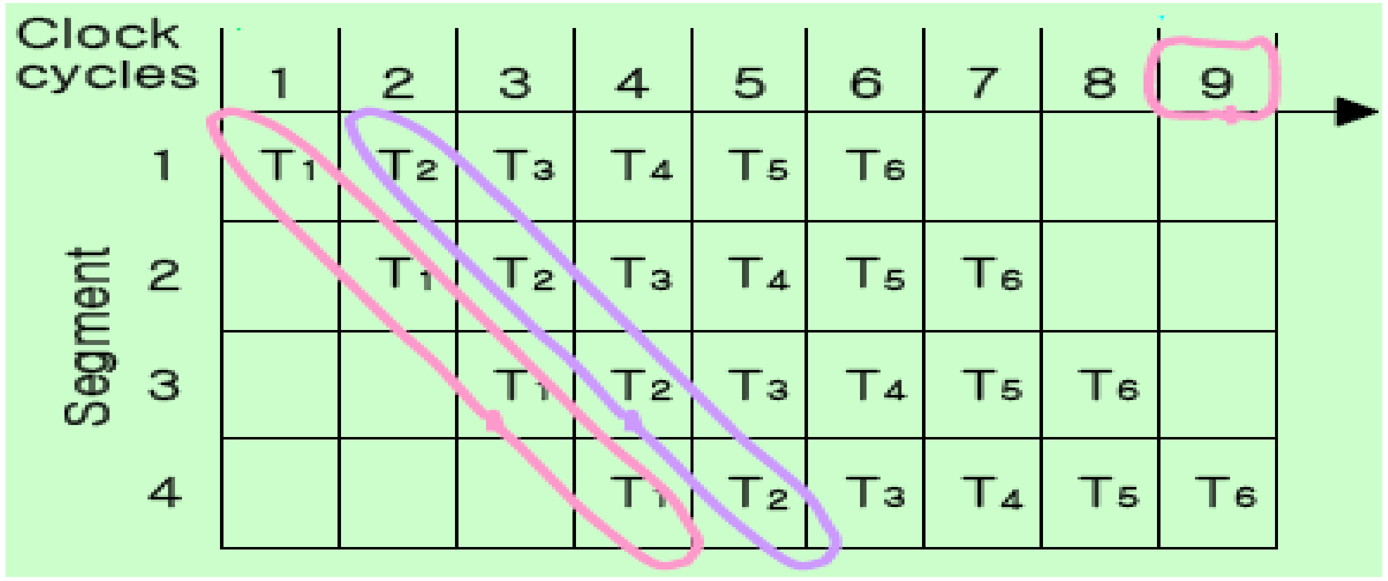
Pipeline Logic:

- A clock drives all the registers in the pipeline. This clock causes the CLC output to be latched in the register which provides input to the next stage, and thus making a start of new computation possible for next stage.
- The maximum clock rate is decided by the time delay of the CLC in the stage and the delay of the staging latch.





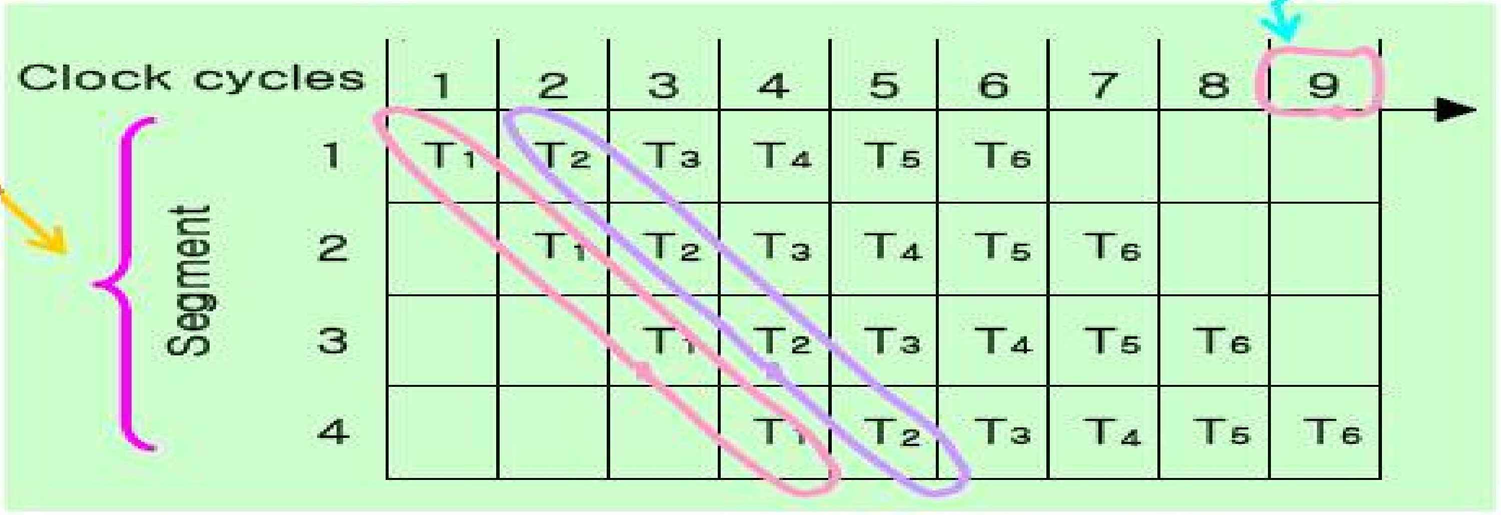
- Each segment consists of CLC (Si) that performs a suboperation over the data stream flowing through the pipe.
- The segments are separated by registers (Ri) that hold the intermediate results between stages.



Segment \ Clock cycles	1	2	3	4	5	6	7	8	9
1	T_1	T_2	T_3	T_4	T_5	T_6			
2		T_1	T_2	T_3	T_4	T_5	T_6		
3			T_1	T_2	T_3	T_4	T_5	T_6	
4				T_1	T_2	T_3	T_4	T_5	T_6

Speedup S : Nonpipeline / Pipeline

- $S = n \cdot t_n / (\boxed{k + n - 1}) \cdot t_p$
  - » n : *task number* (6)
  - » t_n : *time to complete each task in nonpipeline*
  - » t_p : *clock cycle time* (1 clock cycle)
  - » k : *segment number* (4)



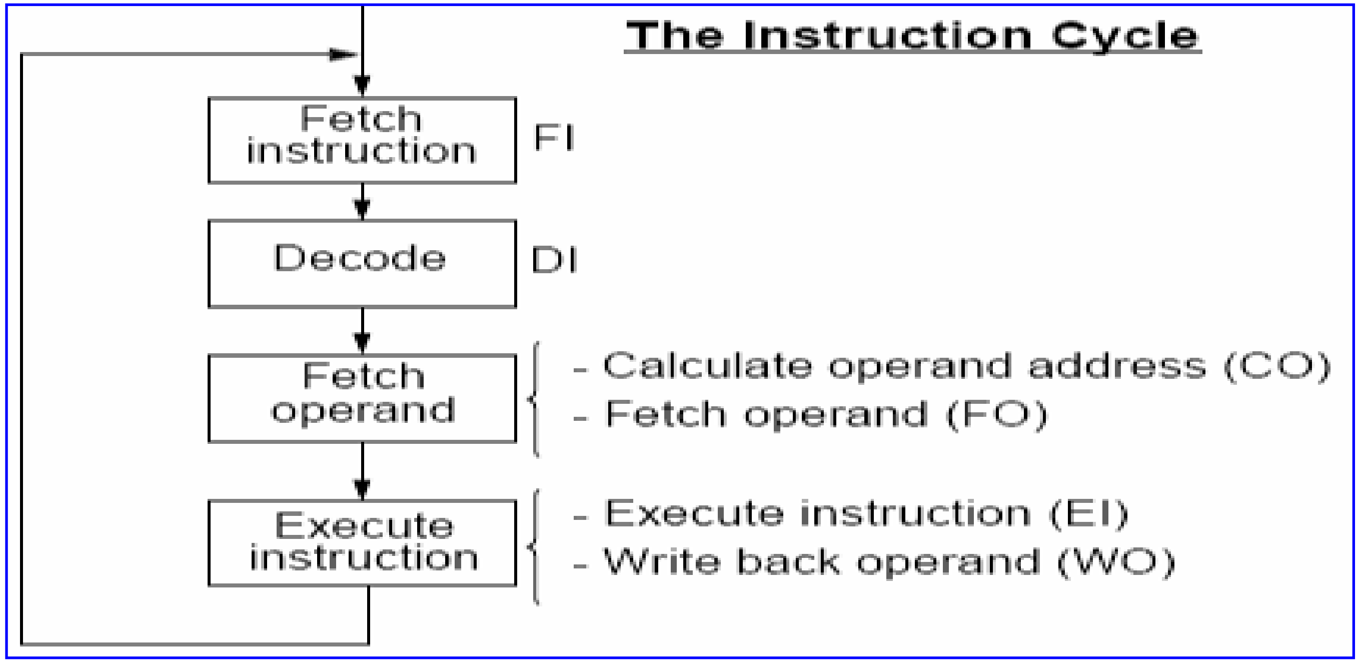
Clock cycles	1	2	3	4	5	6	7	8	9
Segment 1	T_1	T_2	T_3	T_4	T_5	T_6			
Segment 2		T_1	T_2	T_3	T_4	T_5	T_6		
Segment 3			T_1	T_2	T_3	T_4	T_5	T_6	
Segment 4				T_1	T_2	T_3	T_4	T_5	T_6

- If $n \to \infty$, $S = t_n / t_p$ $k + n - 1 \approx n$
- ***nonpipeline*** (t_n) **=** ***pipeline*** ($k \cdot t_p$)

$$S = t_n / t_p = k \cdot t_p / t_p = k$$

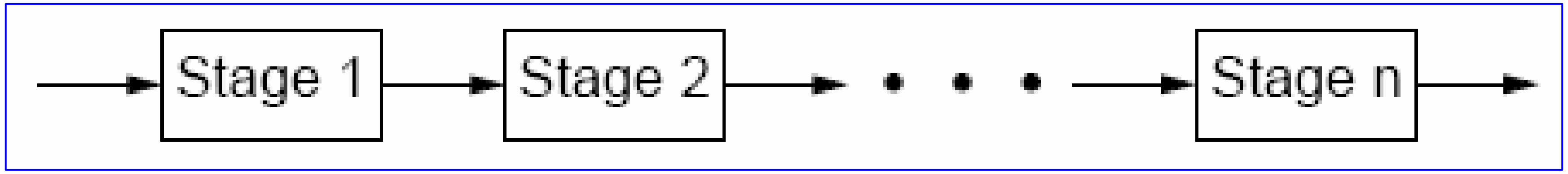
Instruction Pipelining

- Instruction execution is extremely complex and involves several operations which are executed *successively*

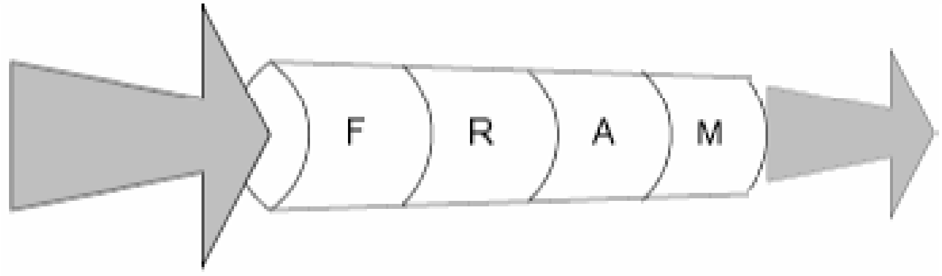


- Pipelining is an implementation technique whereby multiple instructions are overlapped in execution. This is solved without additional hardware but only by letting different parts of the hardware work for different instructions at the same time.

- The pipeline organization of a CPU is similar to an assembly line: the work to be done in an instruction is broken into smaller steps (pieces), each of which takes a fraction of the time needed to complete the entire instruction. Each of these steps is a *pipe stage* (or a *pipe segment*).
- Pipe stages are connected to form a pipe:



- The time required for moving an instruction from one stage to the next: a *machine cycle* (often this is one clock cycle). The execution of one instruction takes several machine cycles as it passes through the pipeline.

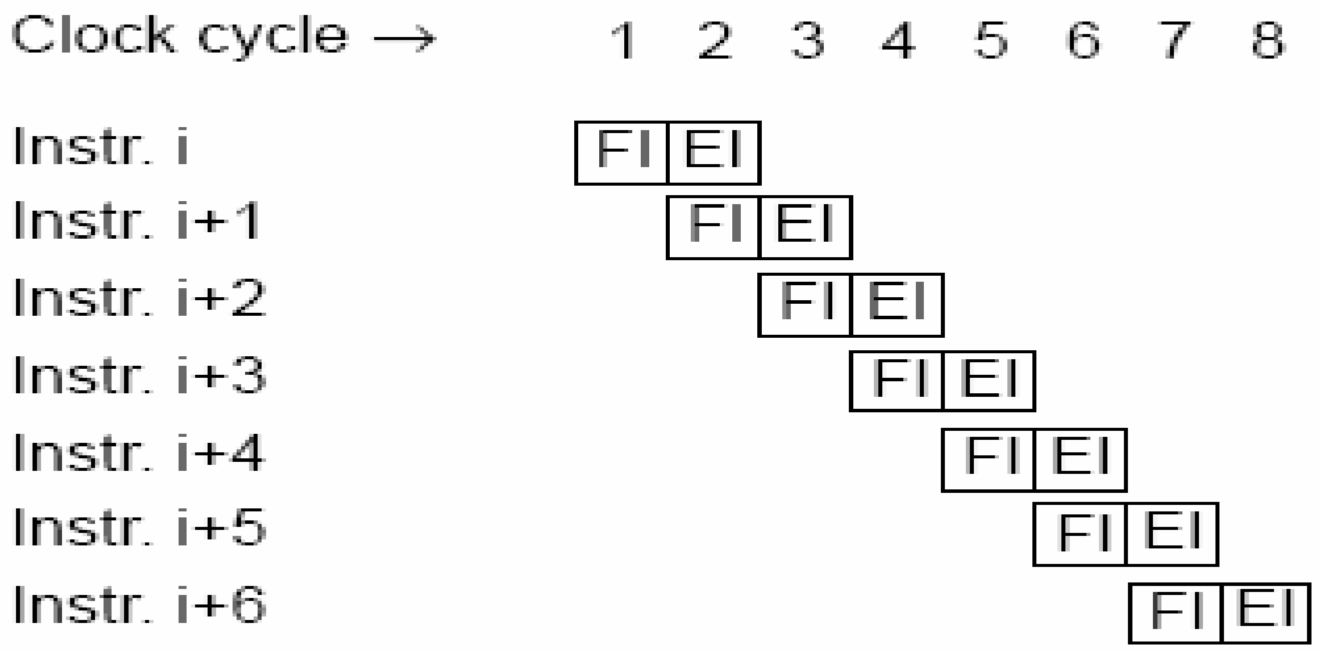


F	- Fetch ML Instruction	A	- ALU Operation
R	- Read Source Registers	M	- Memory Access

Acceleration by Pipelining

Two stage pipeline: FI: fetch instruction

EI: execute instruction



We consider that each instruction takes execution time T_{ex}.

Execution time for the 7 instructions, with pipelining:

$$(T_{ex}/2)*8 = 4*T_{ex}$$

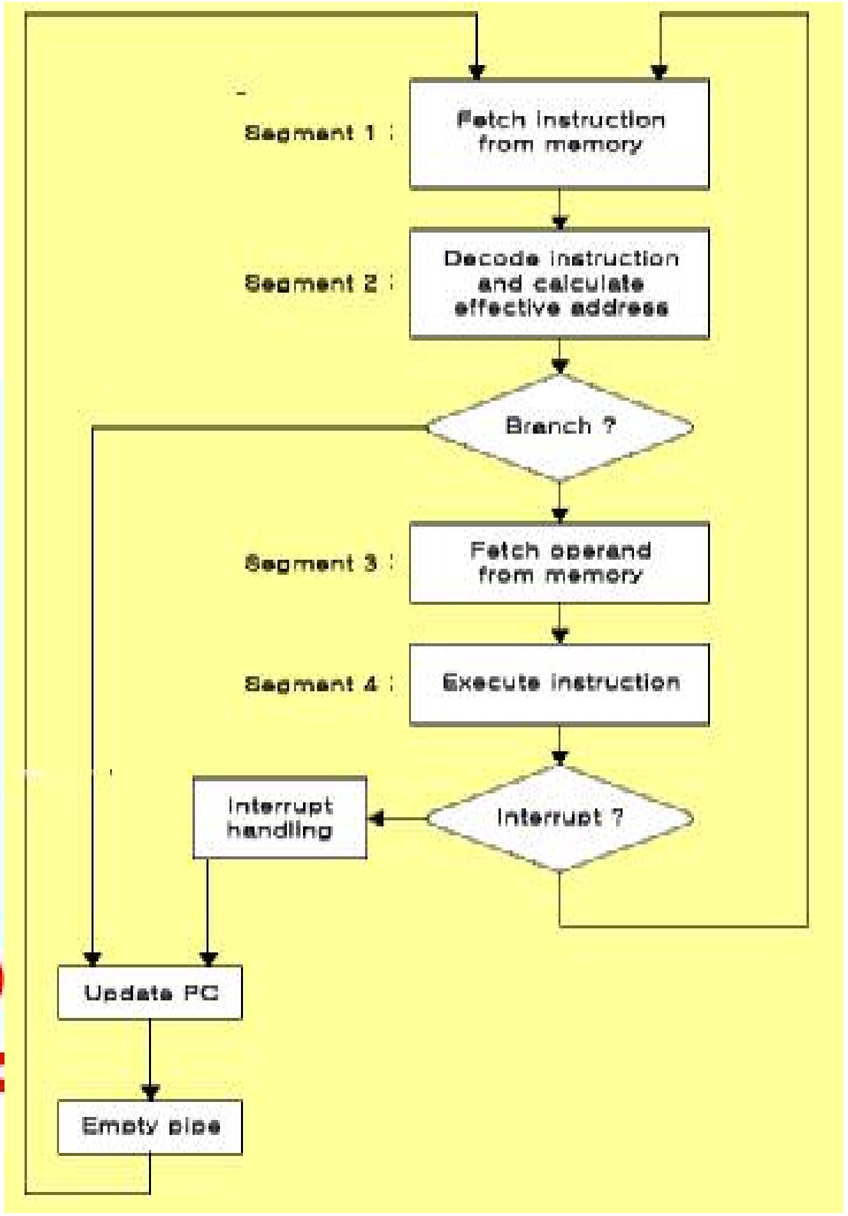
Four-segment CPU pipeline :

» 1) **FI** : Instruction Fetch
» 2) **DA** : Decode Instruction & calculate EA
» 3) **FO** : Operand Fetch
» 4) **EX** : Execution

Step :		1	2	3	4	5	6	7	8	9	10	11	12	13
Instruction :	1	FI	DA	FO	EX									
	2		FI	DA	FO	EX								
(Branch)	3			FI	DA	FO	EX							
	4				FI	—	—	FI	DA	FO	EX			
	5					—	—	—	FI	DA	FO	EX		
	6									FI	DA	FO	EX	
	7										FI	DA	FO	EX

No Branch

Branch



Six stage pipeline

FI:	fetch instruction	FO:	fetch operand
DI:	decode instruction	EI:	execute instruction
CO:	calculate operand address	WO:	write operand

Clock cycle →	1	2	3	4	5	6	7	8	9	10	11	12
Instr. i	FI	DI	CO	FO	EI	WO						
Instr. i+1		FI	DI	CO	FO	EI	WO					
Instr. i+2			FI	DI	CO	FO	EI	WO				
Instr. i+3				FI	DI	CO	FO	EI	WO			
Instr. i+4					FI	DI	CO	FO	EI	WO		
Instr. i+5						FI	DI	CO	FO	EI	WO	
Instr. i+6							FI	DI	CO	FO	EI	WO

Execution time for the 7 instructions, with pipelining:

$$(T_{ex}/6)*12 = 2*T_{ex}$$

- After a certain time (N-1 cycles) all the N stages of the pipeline are working: the pipeline is filled. Now, *theoretically*, the pipeline works providing maximal parallelism (N instructions are active simultaneously).
- Apparently a greater number of stages always provides better performance. However:
  - a greater number of stages increases the over-head in moving information between stages and synchronization between stages.
  - with the number of stages the complexity of the CPU grows.
  - it is difficult to keep a large pipeline at maximum rate because of *pipeline hazards*.

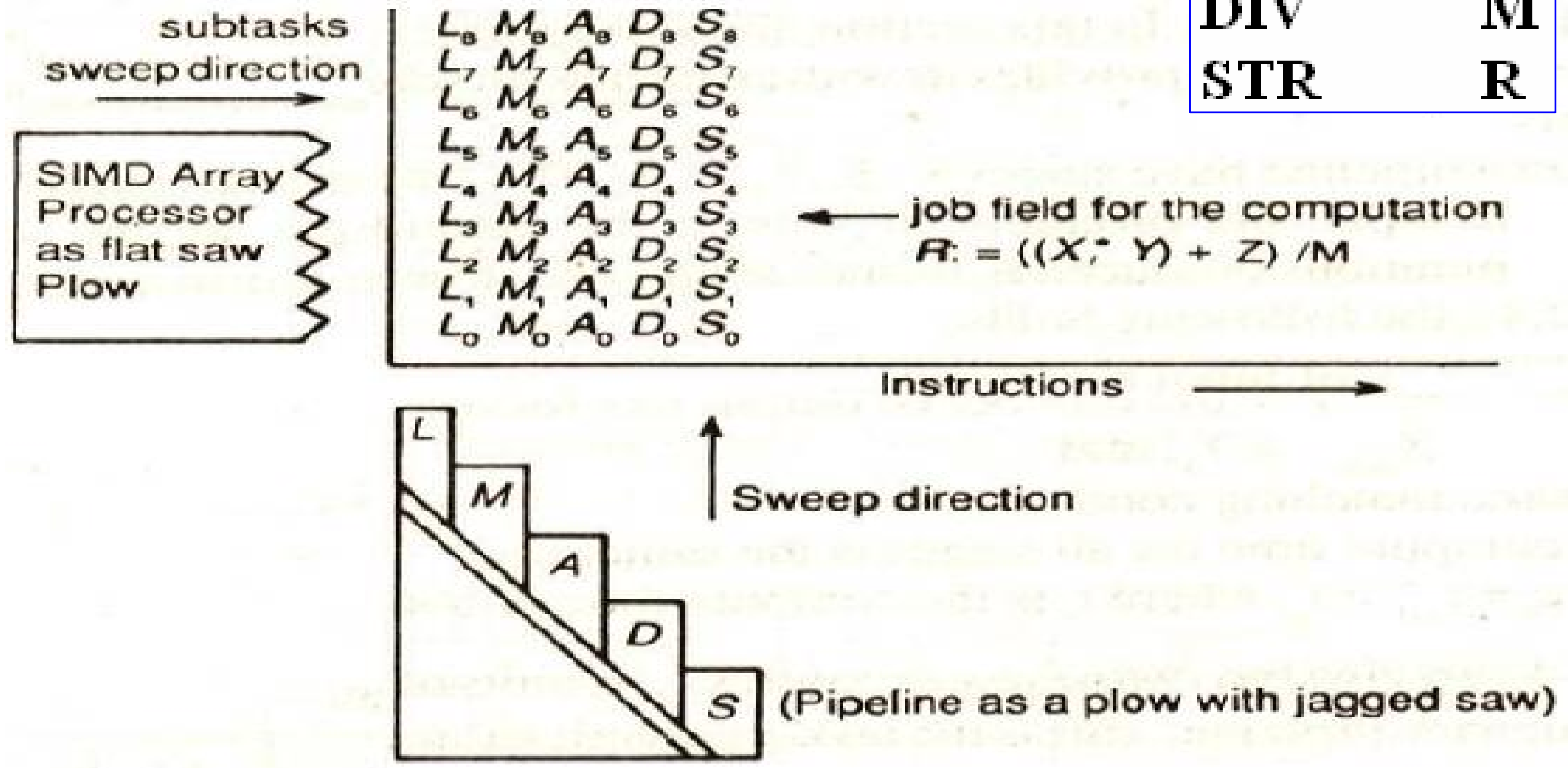
80486 and Pentium:	five-stage pipeline for integer instr.
	eight-stage pipeline for FP instr.
PowerPC:	four-stage pipeline for integer instr.
	six-stage pipeline for FP instr.

Pipeline and Parallel Processors:

Example: $R = ((X*Y)+Z)/M$

A vector machine program would be like that: →

LOAD	X
MULT	Y
ADD	Z
DIV	M
STR	R



subtasks sweep direction
SIMD Array Processor as flat saw Plow
$L_8\ M_8\ A_8\ D_8\ S_8$
$L_7\ M_7\ A_7\ D_7\ S_7$
$L_6\ M_6\ A_6\ D_6\ S_6$
$L_5\ M_5\ A_5\ D_5\ S_5$
$L_4\ M_4\ A_4\ D_4\ S_4$
$L_3\ M_3\ A_3\ D_3\ S_3$
$L_2\ M_2\ A_2\ D_2\ S_2$
$L_1\ M_1\ A_1\ D_1\ S_1$
$L_0\ M_0\ A_0\ D_0\ S_0$
job field for the computation $R := ((X * Y) + Z) / M$
Instructions
Sweep direction
L
M
A
D
S
(Pipeline as a plow with jagged saw)

Steady-state Analysis of Pipelines:

Let n= # of stages.

m= # of tasks run on the pipeline

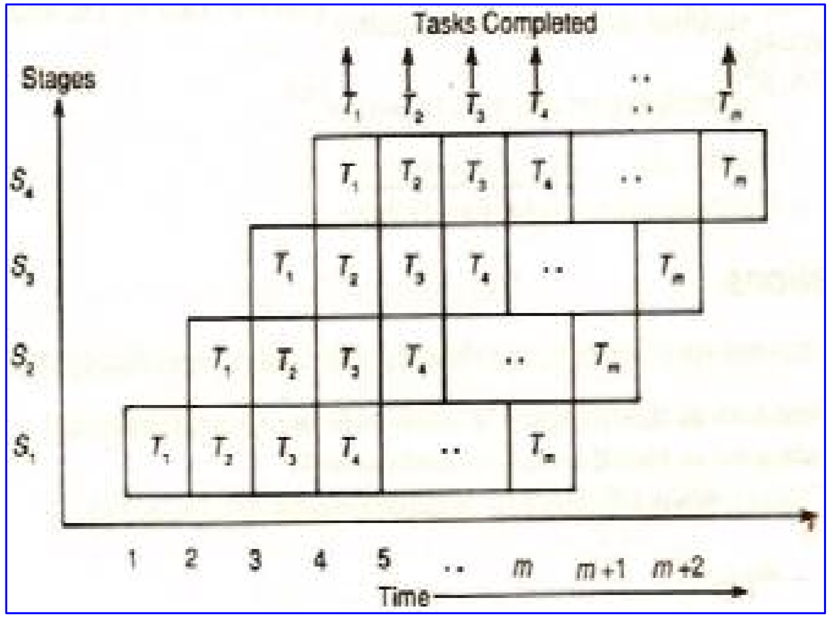
- **Efficiency (e):** is the ratio of the energy used in doing the work and the total energy supplied.

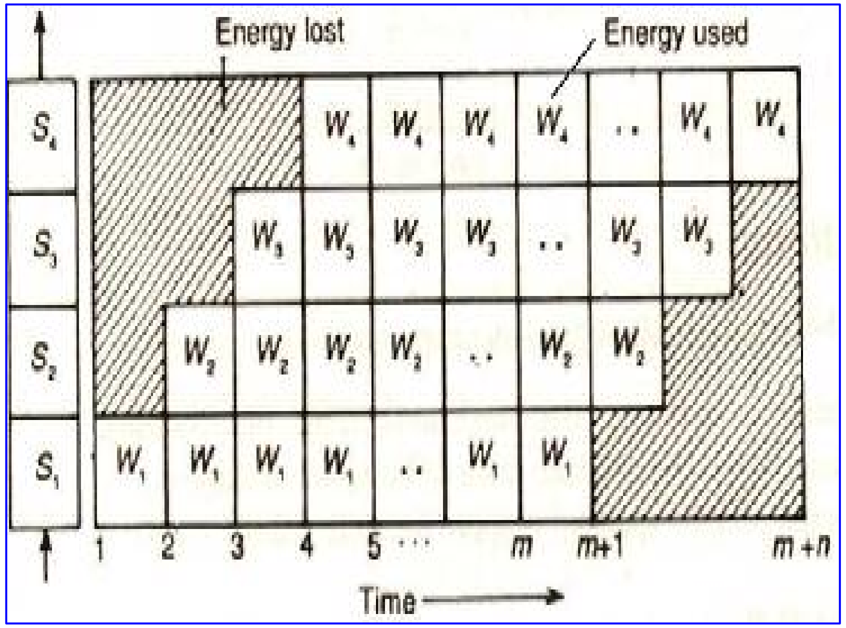
$$e = \frac{m \cdot n}{(m+n-1).n} = \frac{m}{(m+n-1)}$$

When n>>m, then e tends to be m/n.

When m>>n, then e tends to be1.

When n=m, then e is 0.5





Steady-state Analysis of Pipelines:

Let n= # of stages.

m= # of tasks run on the pipeline

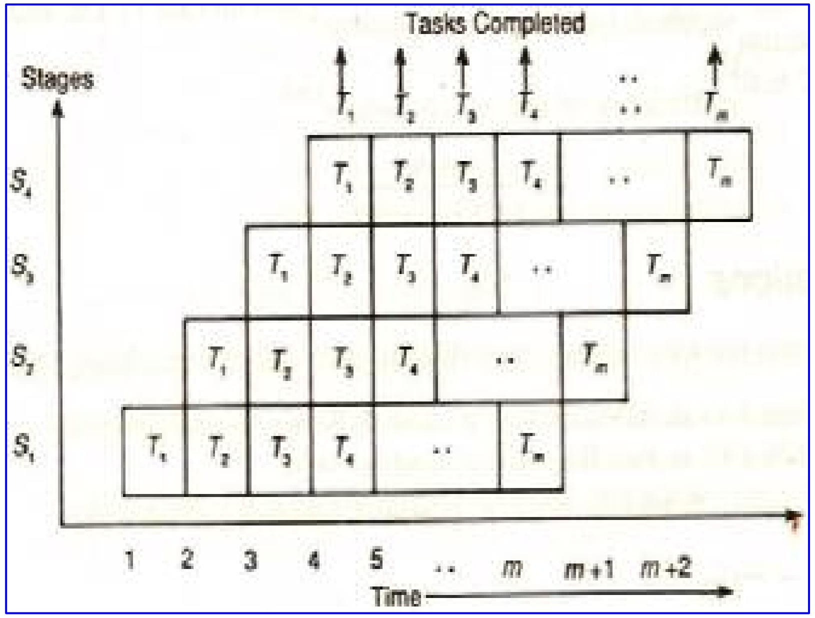
Speedup (S): is the ratio of the time taken by the nonpipelined architecture to the time taken by the pipeline.

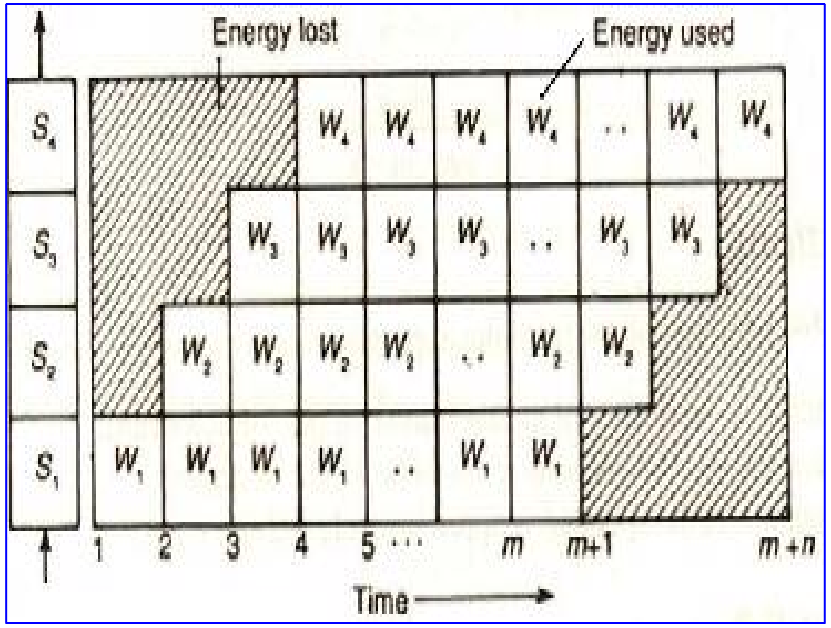
$$S = \frac{(n.t).m}{(m+n-1).t} = \frac{m.n}{(m+n-1)}$$

When n>>m, then S tends to be m.

When m>>n, then S tends to be n.

When n=m, then S is n/m





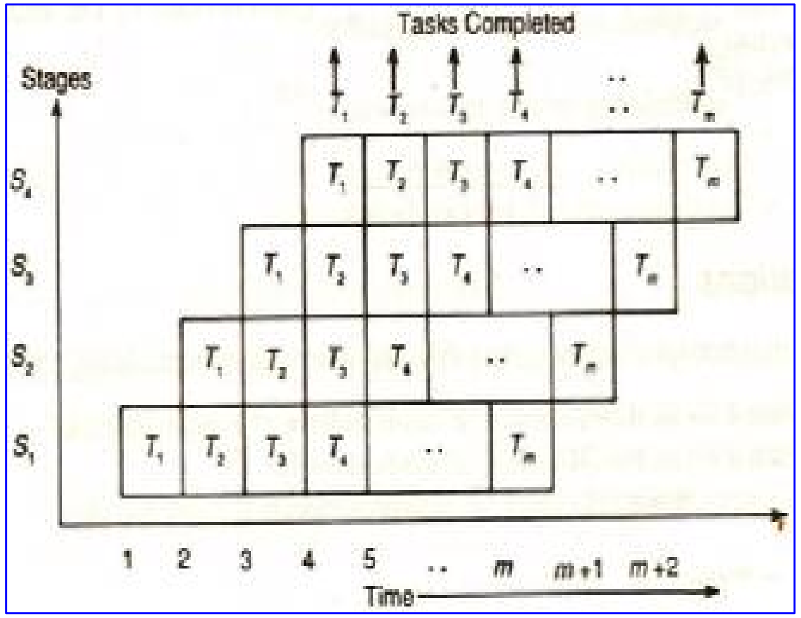
Steady-state Analysis of Pipelines:

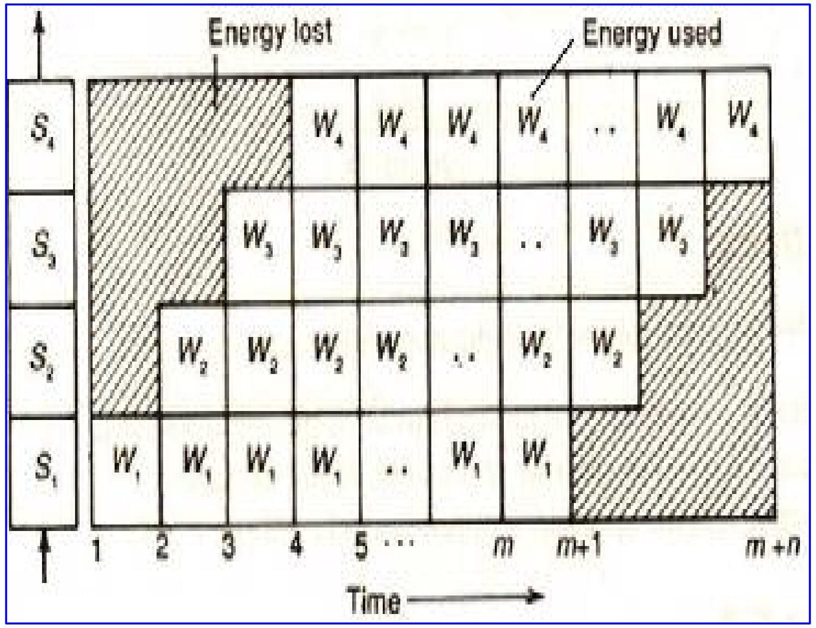
Let n= # of stages.

m= # of tasks run on the pipeline

Throughput (Q): is defined as the number of tasks completed per unit amount of time.

$$Q = \frac{m}{(m+n-1).t} = \frac{e}{t}$$



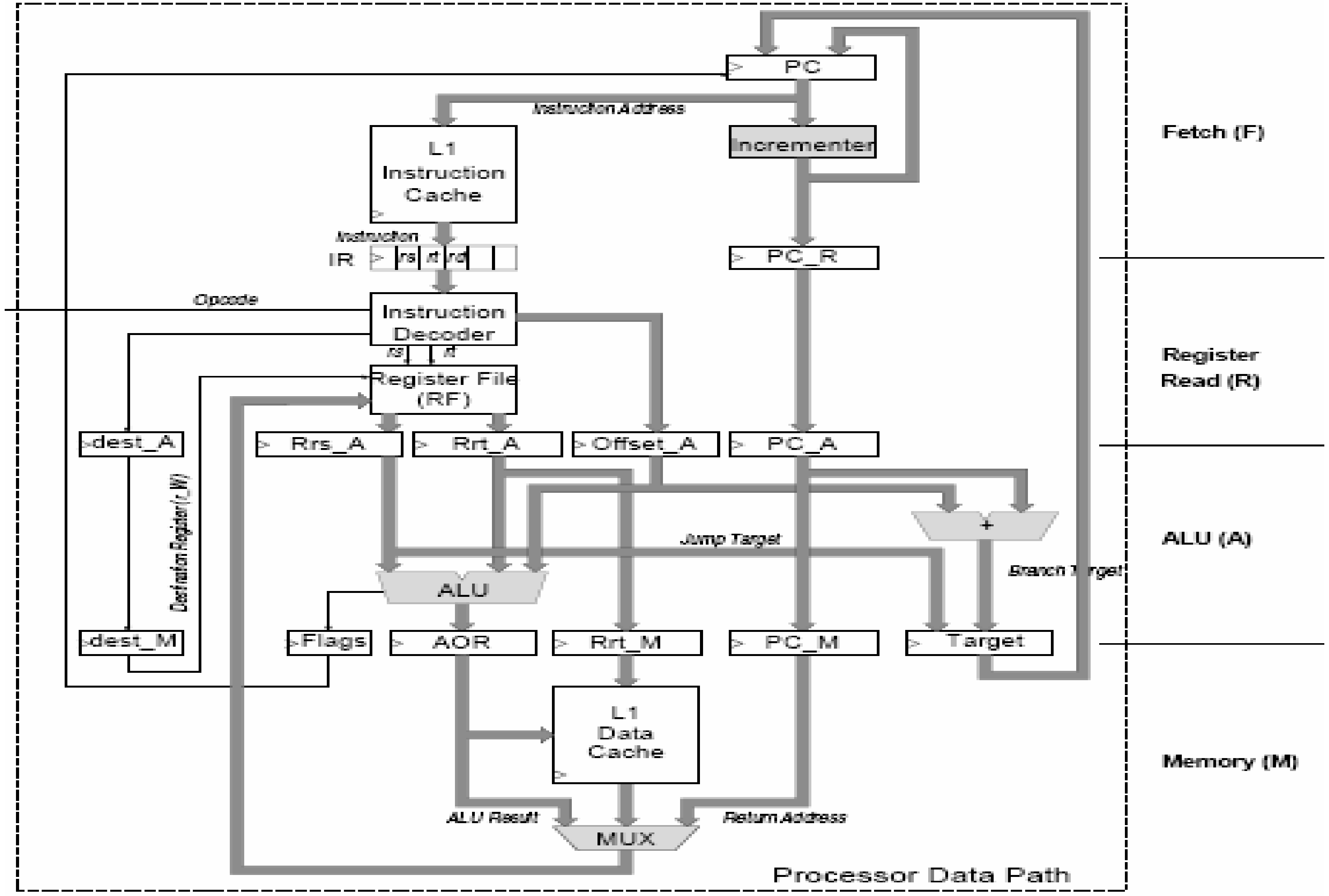


When n>>m, then Q tends to be m/(n.t)

When m>>n, then Q tends to be1/t.

When n=m, then Q is 0.5/t

Processor Design: Pipeline



Summary

- Instructions are executed by the CPU as a sequence of steps. Instruction execution can be substantially accelerated by *instruction pipelining*.
- A pipeline is organized as a succession of N stages. At a certain moment N instructions can be active inside the pipeline.
- Keeping a pipeline at its maximal rate is prevented by *pipeline hazards*. *Structural hazards* are due to resource conflicts. *Data hazards* are produced by data dependencies between instructions. *Control hazards* are produced as consequence of branch instructions
- With conditional branch we have a penalty even if the branch has *not* been taken. This is because we have to wait until the branch condition is available.
- Branch instructions represent a major problem in assuring an optimal flow through the pipeline. Several approaches have been taken for reducing branch penalties