Pipelining

↳ Doing multiple instruction simultaneously by dividing instruction into stages

↳
- Fetch instruction (FI)
  - Read the next expected instruction into a buffer
- Decode instruction (DI)
  - Determine the opcode and the operand specifiers
- Calculate operands (CO)
  - Calculate the effective address of each source operand
  - This may involve displacement, register indirect, indirect, or other forms of address calculation
- Fetch operands (FO)
  - Fetch each operand from memory
  - Operands in registers need not be fetched
- Execute instruction (EI)
  - Perform the indicated operation and store the result, if any, in the specified destination operand location
- Write operand (WO)
  - Store the result in memory

↳

Time →

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Instruction 1	FI	DI	CO	FO	EI	WO	} 6 stage pipeline							
Instruction 2		FI	DI	CO	FO	EI	WO							
Instruction 3			FI	DI	CO	FO	EI	WO						
Instruction 4				FI	DI	CO	FO	EI	WO					
Instruction 5					FI	DI	CO	FO	EI	WO				
Instruction 6						FI	DI	CO	FO	EI	WO			
Instruction 7							FI	DI	CO	FO	EI	WO		
Instruction 8								FI	DI	CO	FO	EI	WO	
Instruction 9							input instr →		FI	DI	CO	FO	EI	WO

Pipelining:- doing thing simultaneously

pipeline latency — output instr

↳

Time → ←Branch Penalty→

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Instruction 1	FI	DI	CO	FO	EI	WO								
Instruction 2		FI	DI	CO	FO	EI	WO							
Instruction 3			FI	DI	CO	FO	EI	WO	→ fetch ins. 15					
Instruction 4				FI	DI	CO	FO							
Instruction 5					FI	DI	CO							
Instruction 6						FI	DI							
Instruction 7							FI							
Instruction 15								FI	DI	CO	FO	EI	WO	
Instruction 16									FI	DI	CO	FO	EI	WO

cannot straight jump / must empty pipe

↳ maximum of r, latch delay between stage

$r = \max[r_i] + d$; $1 \le i \le k$

$r = r_m + d$

$r_m \gg d$

$r = r_m$

$r = \max[r_i] + d$

- r : cycle time of an instruction pipeline
- r_m : maximum stage delay/latency
- d : time delay/ latency of a latch (time needed to advance singals and data from one stage to the next stage)
- k : number of stages in the instruction pipeline
- Total time (T_k) required to execute all n instructions (with no branch)

$$T_k = [k + (n-1)]r \leftarrow (1 \times k + (n-1) \times 1)r$$

(latch delay between stages; time for 1 stage to complete)

$$S_k = \frac{Sequential\ (Time)}{Pipeline\ (Time)}$$

} compare pipeline w/ sequential — * speedup

$$= \frac{nkr}{[k + (n-1)]r}$$

$$= \frac{nk}{k + (n-1)}$$

Time →

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Instruction 1	FI	DI	CO	FO	EI	WO								
Instruction 2		FI	DI	CO	FO	EI	WO							
Instruction 3			FI	DI	CO	FO	EI	WO						
Instruction 4				FI	DI	CO	FO	EI	WO					
Instruction 5					FI	DI	CO	FO	EI	WO				
Instruction 6						FI	DI	CO	FO	EI	WO			
Instruction 7							FI	DI	CO	FO	EI	WO		
Instruction 8								FI	DI	CO	FO	EI	WO	
Instruction 9									FI	DI	CO	FO	EI	WO

(num of ins., num of stage, cycle time)

$$S_k = \frac{nkr}{[k + (n-1)]r} = \frac{9 * 6 * r}{[6 + (9-1)]r} = \frac{54}{14} = 3.86$$

pipeline cycle time, r

60, 50, 80, 90 ns   latch = 10ns

r = max{60, 50, 80, 90} + 10ns

= 90 + 10

= 100 ns

Non-pipeline exec time → sequentially

60 + 50 + 90 + 80

= 280 ns

Pipeline time 1000 task

= 1 × 4 clock cycles + 999 × 1 clock cycle

= 4 × cycle time + 999 × cycle time

= 4 × 100ns + 999 × 100ns

= 400 ns + 99900 ns

= 100300 ns

Non pipeline time 1000 task

= 1000 × time taken one task

= 1000 × 280 ns

= 280000 ns

Throughput → number of task that can be done in pipeline time

$T = \frac{1000}{100300\ ns} = 0.00099$

↳ Pipeline Hazard

↳ • Resource Hazard

hardware cannot support all possible comb. of instruction

Clock cycle

Instruction	1	2	3	4	5	6	7	8	9
I1	FI	DI	FO	EI	WO				
I2		FI	DI	FO	EI	WO			
I3			Idle	FI	DI	FO	EI	WO	
I4	in used				FI	DI	FO	EI	WO

• Data Hazard

instruction depends on result of previous instruction

Clock cycle

	1	2	3	4	5	6	7	8	9	10
ADD EAX, EBX	FI	DI	FO	EI	WO					
SUB ECX, EAX		FI	DI	Idle		FO	EI	WO		
I3			FI	RAW		DI	FO	EI	WO	
I4						FI	DI	FO	EI	WO

↳ Type:-

↳ RAW (read after write) : read happen before write completed

WAR (write after read) : write complete before read start

WAW (write after write) : take place in reverse wanted order

• Control Hazard (branch)

pipeline make wrong decision on branch prediction

↳ Dealing w/ Branch:

↳ • Multiple streams

allow pipeline to fetch both instructions, using 2 stream

• Prefetch Branch Target

target of the branch is prefetched & saved for execution

• Loop Buffer

buffer containing n most recently fetched instruction

• Branch prediction

using various technique to predict whether a branch will be taken