

CS8625-June-2-09

Pipelines

**CS8625 High Performance and
Parallel Computing
Dr. Ken Hoganson**

Class

Will

Start

Momentarily...

- We observed that we can obtain better performance in executing instructions, if a single cycle accomplishes multiple operations:
- We added a shift register after the ALU to improve the performance on the multiplication and division algorithms.
- Not all operations will need both the ALU and the SHIFT register
- But all instructions do have some things in common:
 - Instruction fetch
 - Decode
 - Operand fetch (if operands)

- Pipelines are an architectural enhancement that improves the performance of a computer system without increasing the clock speed.
- Pipelines take advantage of **parallelism** inherent in the instruction execution process and instructions.
- Based on the observation that all instructions go through the same set of phases (stages) in the process of begin executed: *(one possible example:)*
 - INSTRUCTION FETCH
 - INSTRUCTION DECODE
 - OPERAND FETCH
 - EXECUTION
 - RESULT STORE (not all architectures)

- Do each stage in parallel, with different instructions in each stage
- Instructions "flow" from one stage to the next

Stages

time	IF	DE	OF	EX	RS	DONE	
1	1						
2	2	1	Instructions				} Load Time
3	3	2	1				
4	4	3	2	1			
5	5	4	3	2	1		
6	6	5	4	3	2	1	
7	7	6	5	4	3	2	
etc.							

CPU cycle time subdivided into stage-times (5 in this example) .One Instructions completes each stage-time after the pipeline is loaded.

- k : stages in pipeline
- t : time to do each stage (actually slowest stage constrains performance)
- n : number of instructions to be processed

Time required to complete the processing on a non-pipelined processor: (CPU cycle = kt time)
 $= n(kt)$

Time required for pipelined processor:

1st instruction: $= kt$ (pipeline load)

All after 1st ($n-1$ more after 1st): $= (n-1)t$

$= kt + (n-1)t$

SPEEDUP is measured as SERIAL TIME divided by PARALLEL TIME

$$\frac{\text{SERIAL TIME}}{\text{PARALLEL TIME}} = \frac{nkt}{kt + (n-1)t} = \frac{nk}{k + n - 1}$$

Or, for large n , $\rightarrow \frac{nk}{n} \rightarrow k$

- EFFICIENCY is measured as the ratio of **SPEEDUP** over NUMBER of PROCESSING ELEMENTS (in this case stages)

$$EFFICIENCY = \frac{SPEEDUP}{STAGES} = \frac{nk}{k + n - 1} = \frac{n}{k + n - 1}$$

Example:

100 instructions

4 stages

$$Speedup = \frac{100 \times 4}{4 + 100 - 1} = \frac{400}{103} \approx 4$$

$$Efficiency = \frac{100}{4 + 100 - 1} = \frac{100}{103} \leq 1$$

- Up until now, we have assumed an **idealized** pipeline.
- Problems that can limit pipeline performance:
 - **STRUCTURAL HAZARDS**
 - von Neuman characteristics, multiple memory accesses for each instruction, but only one bus to memory
 - **DATA DEPENDENCY HAZARDS**
 - Some instructions may need the results that are yet to be produced from the instruction(s) ahead of it in the pipeline, but not yet complete
 - **CONTROL HAZARDS**
 - Jumps and branches
 - interrupt the sequential flow of instructions
 - cause pipeline to be “flushed” and reloaded
 - reported to be about 30% of all instructions!

- In a 5-stage pipeline, three of the five stages may require access to memory over the bus (or access to cache)

IF - memory access

DE - no access

OF - memory access

EX - no access

RS - memory access

Causes the pipeline to STALL – only one access at a time.

Reduces realizable performance improvements.

CYCLES (subdivisions of system clock). **BLUE** is bus/memory access.

Instr.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	IF	DE	OF	EX	RS										
2		IF	DE	OF	EX	RS		(S=Stall)							
3			S	S	S	S	IF	DE	OF	EX	RS				
4								IF	DE	OF	EX	RS			
5									S	S	S	S	IF	DE	OF

Note the pattern - 2 instruction complete every six (subdivided) cycles.

SPEEDUP = SERIAL over PARALLEL =

$$\text{Serial/Par} = (2\text{inst} * 5\text{cycle}) / 6 = 10/6 = 1.667$$

Reduce the number & frequency of access to memory over the bus:

1. CACHE
2. SEPARATE INSTRUCTION AND DATA CACHE
3. WIDE WORD MEMORY
4. MULTI-PORT MEMORY

- Allows fetch of two consecutive instructions at a time.
- Frees the bus for operand fetches or result stores
- Implemented either as a wide-word interface to the memory, where the bus is twice as wide as most instructions and perhaps larger than the processor "word size"
- Or, implemented as a pipelined-memory scheme, where one address sent from CPU to memory results in two consecutive instructions being returned, one-at-a-time but in quick sequence over the bus (saves significant time compared to two separate instruction fetches)

CYCLES (subdivisions of system clock)

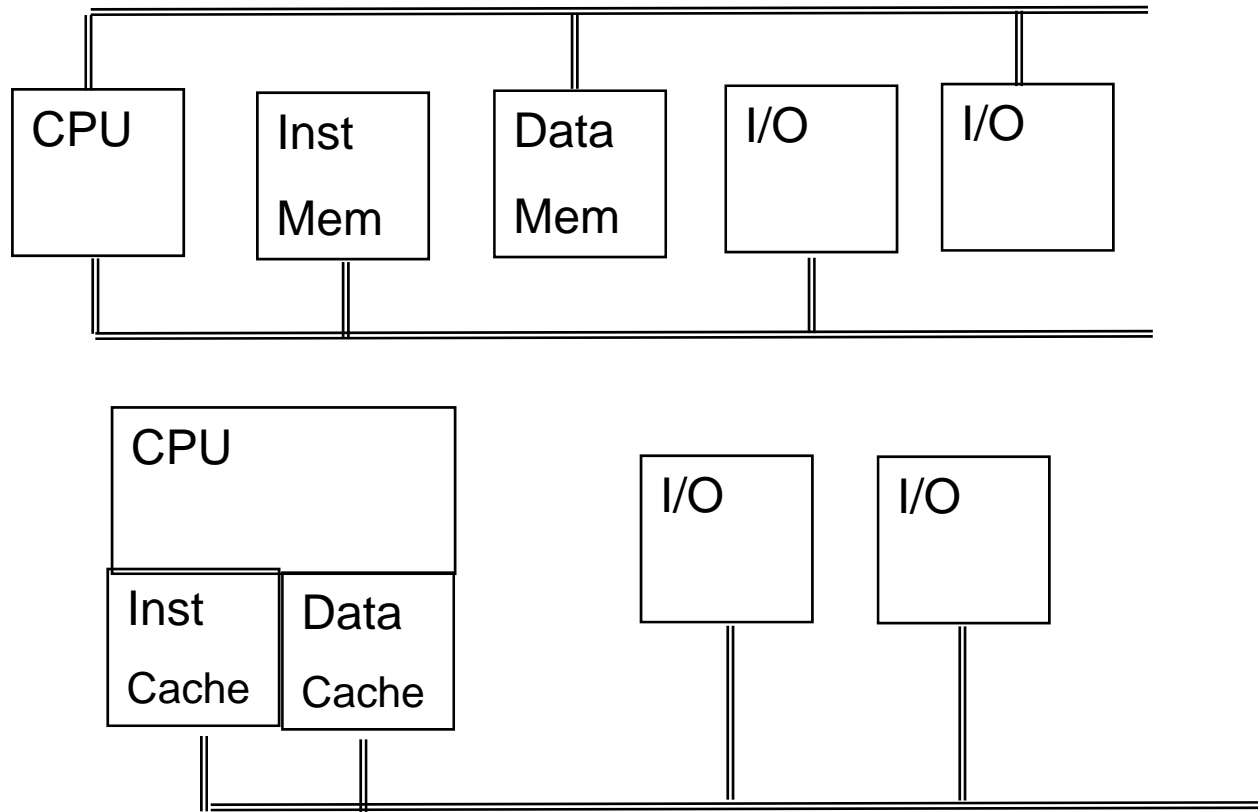
Instr.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	IF	DE	OF	EX	RS										
2	IF	S	DE	OF	EX	RS		(S=Stall)							
3		IF	S	DE	S	S	OF	EX	RS						
4		IF	S	S	DE	S	S	OF	EX	RS					
5											IF	DE	OF	EX	RS
6											IF	S	DE	OF	EX
7												IF	S	DE	S
8												IF	S	S	DE

^Pattern Repeats

Note the pattern - 4 instruction complete every 10 (subdivided) cycles.

$$\text{SPEEDUP} = \text{SERIAL over PARALLEL} = (4 \times 5) / 10 = 20 / 10 = 2$$

- Allows fetch of instructions and data simultaneously
- Separate caches, one for data, one for instructions



CYCLES (subdivisions of system clock)

Instr.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	IF	DE	OF	EX	RS										
2		IF	DE	OF	EX	RS						(S=Stall)			
3			IF	DE	S	S	OF	EX	RS						
4				IF	DE	S	S	OF	EX	RS					
5									IF	DE	OF	EX	RS		
6										IF	DE	OF	EX	RS	
7											IF	DE	S	S	OF
8												IF	DE	S	S

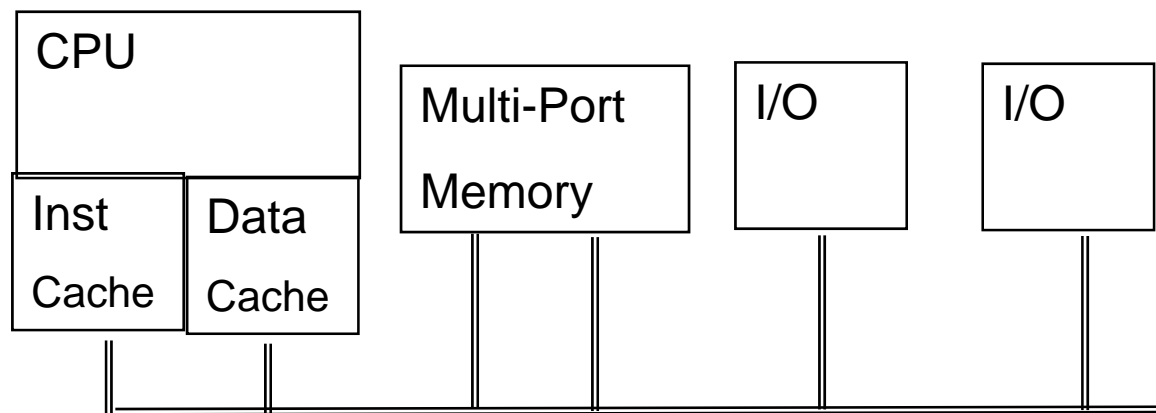
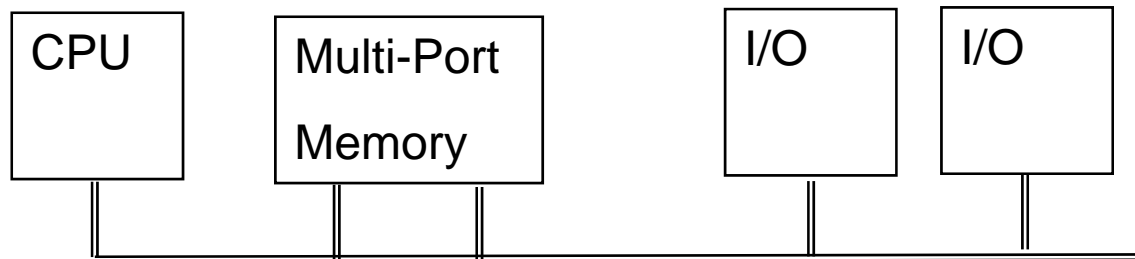
Pattern Repeats every 10, with overlap of 2

== pattern of 8 with +2 startup

Note the pattern: 4 instructions complete every 8 (subdivided) cycles.

$$\text{SPEEDUP} = \text{SERIAL over PARALLEL} = (4 \cdot 5) / 8 = 20 / 8 = 2.5$$

- Memory can work on two separate addresses at the same time
- Allows two different operands to be fetched at (almost) the same time
- More expensive memory, but in use in various systems
- Can be combined with Harvard architecture (dual cache)



CYCLES (subdivisions of system clock)

Instr.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	IF	DE	OF	EX	RS										
2		IF	DE	OF	EX	RS									
3			IF	DE	OF	EX	RS								
4				IF	DE	OF	EX	RS							
5					IF	DE	OF	EX	RS						
6						IF	DE	OF	EX	RS					
7							IF	DE	OF	EX	RS				
8								IF	DE	OF	EX	RS			

(S=Stall)

Supports 3 fetches per subcycle - perfect speedup after pipeline load time

One instruction per subcycle. Pipeline load time of 4 (k-1)

SPEEDUP = SERIAL over PARALLEL = 5/1 = 5

**End
Of
Today's
Lecture.**

