Some notes on Pipelining

Yogesh Virkar

Department of Computer Science,

University of Colorado, Boulder.

Pipeline stages

Each instruction can be subdivided into smaller stages such as:

1. Fetch
2. Decode
3. Execute
4. Memory
5. Write

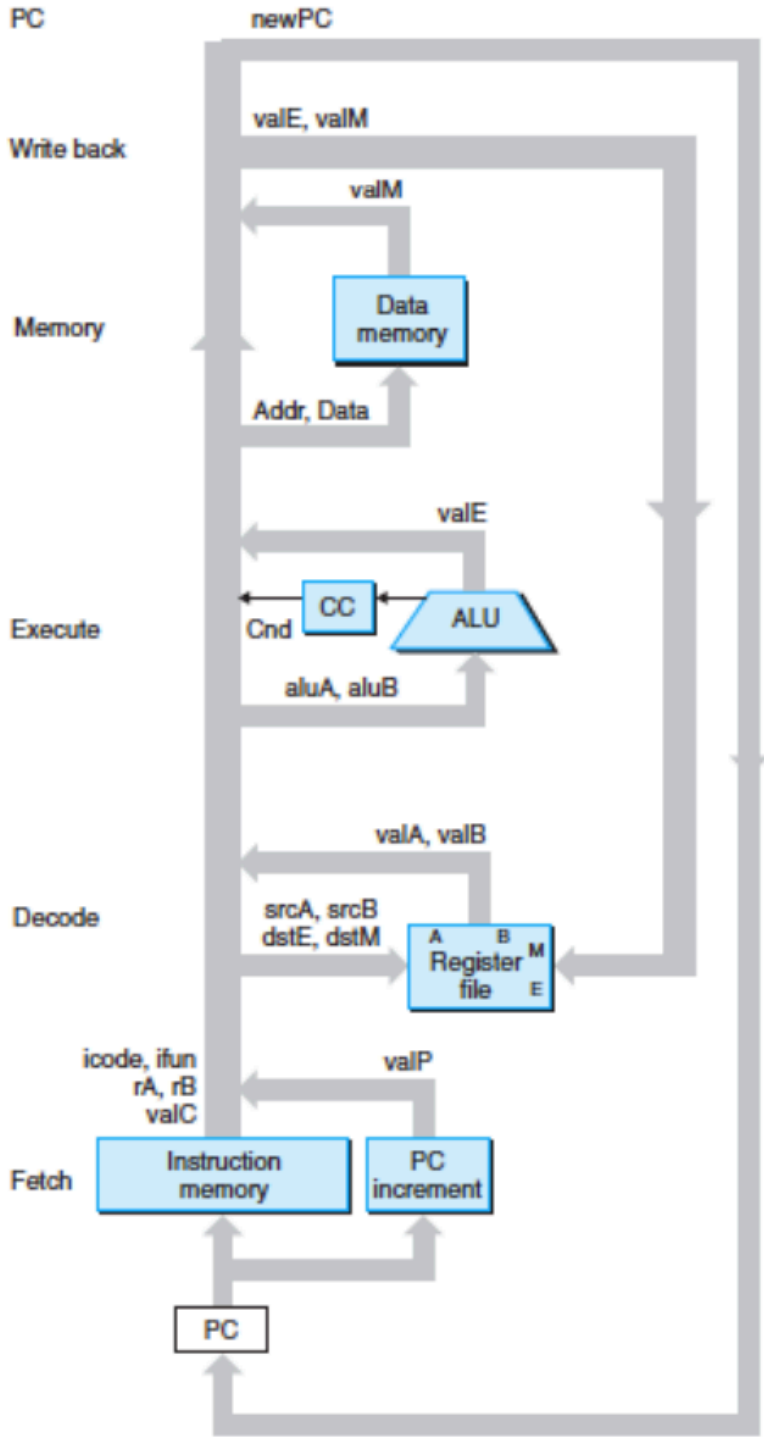
E.g. addq %rdx, %rcx

Pipeline stages

Each instruction can be subdivided into smaller stages such as:

1. Fetch
2. Decode
3. Execute
4. Memory
5. Write

E.g. addq %rdx, %rcx

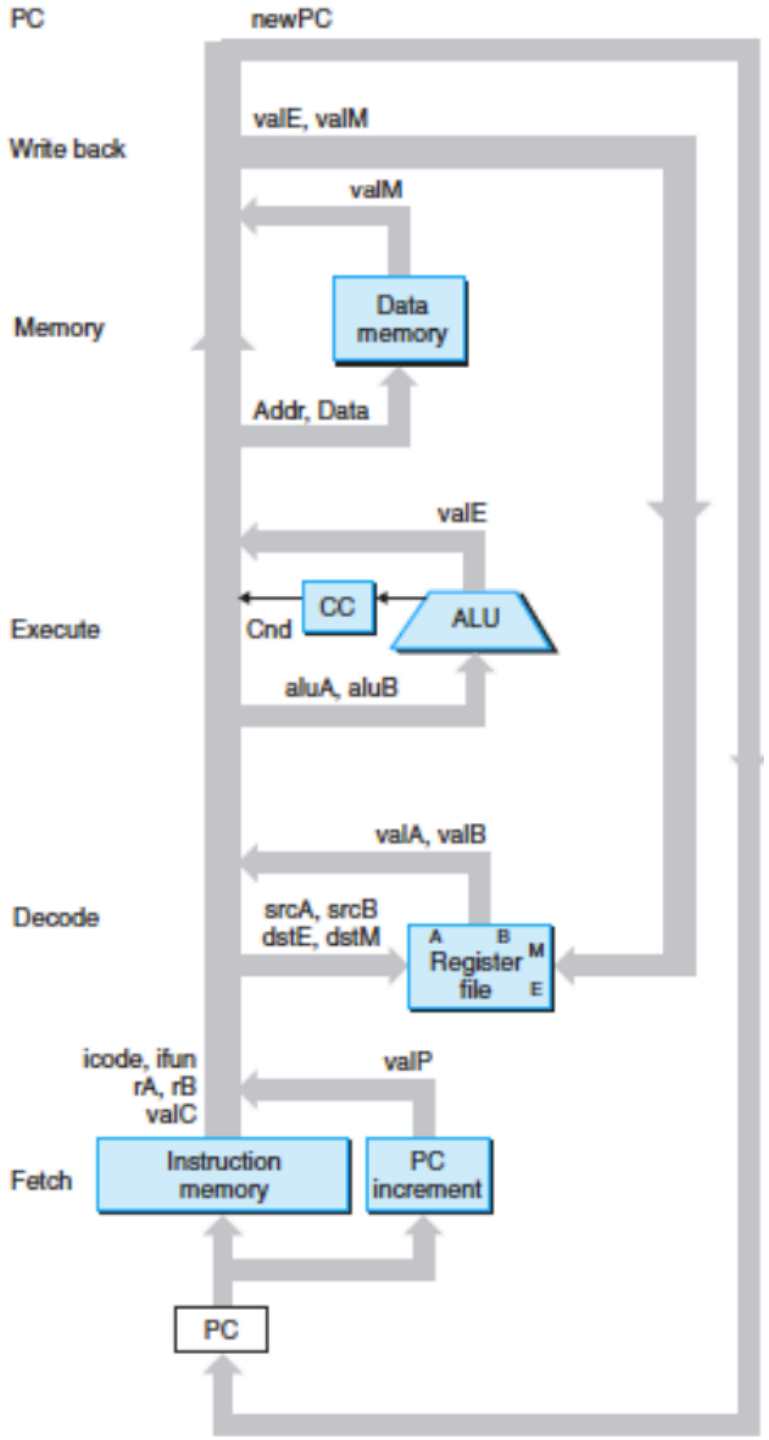


Pipeline stages

Each instruction can be subdivided into smaller stages such as:

1. Fetch
2. Decode
3. Execute
4. Memory
5. Write

E.g. addq %rdx, %rcx



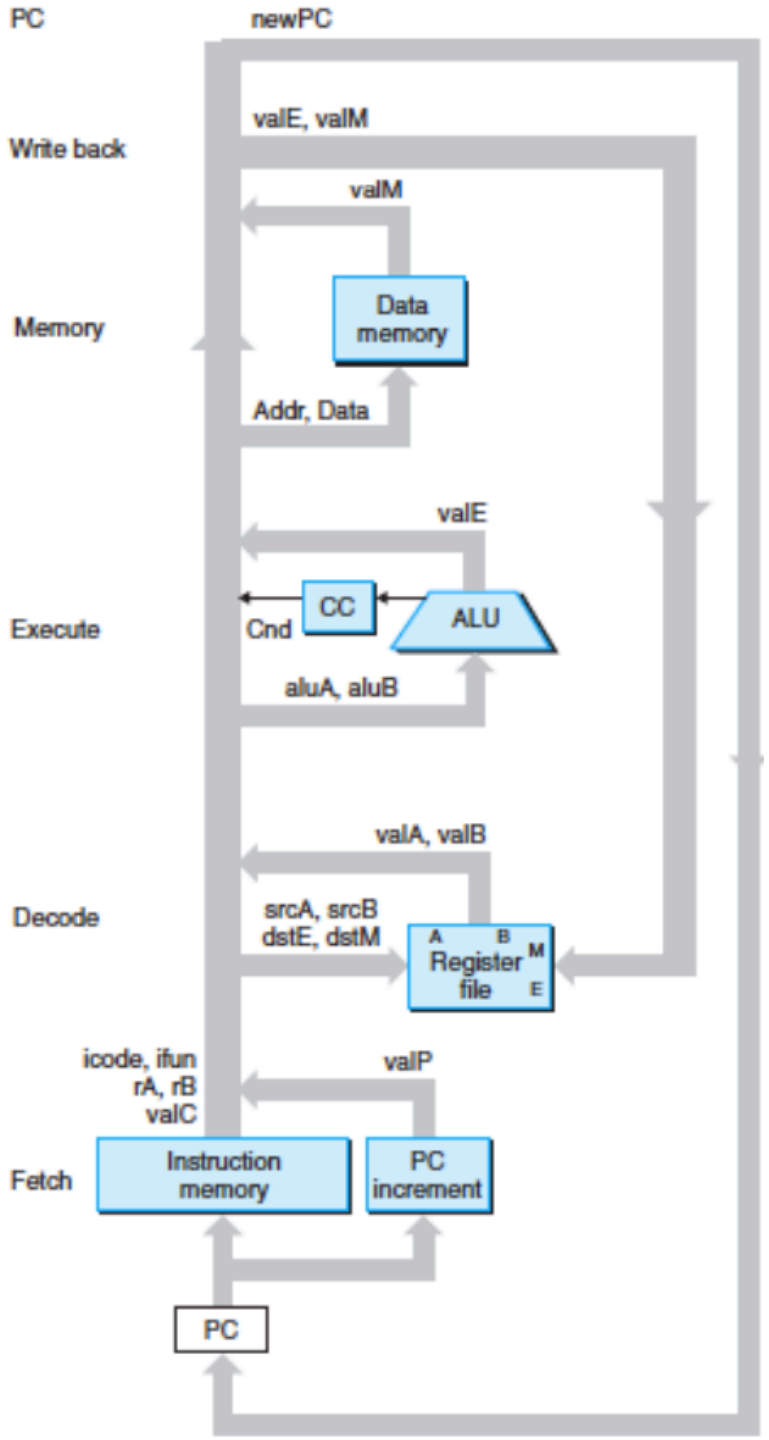
For the addq example we do not need a memory stage since there are no memory accesses. Memory stage is thus unused.

Pipeline stages

Each instruction can be subdivided into smaller stages such as:

1. Fetch
2. Decode
3. Execute
4. Memory
5. Write

E.g. addq %rdx, %rcx



For the addq example we do not need a memory stage since there are no memory accesses. Memory stage is thus unused.

Note: There is a bank of registers between each stage such that result of each stage can be saved.
This allows for pipelining.

NOP question example

- movq $10, %rdx
- movq $3, %rax
- addq %rdx, %rax

NOP question example (3)

- Clock Cyle
- movq $10, %rdx
- movq $3, %rax
- addq %rdx, %rax

1	2	3	4	5	6	7	8	9	10	11
F	D	E	M	***W***						
	F	D	E	M	***W***					
		F	***D***	E	M	W				

- Issue: We have already decoded the operands (i.e., %rax, %rdx) before they have been correctly assigned values by their respective write (W) stages.
- Thus addq instruction will yield incorrect result!

NOP question example (4)

- Clock Cyle
- movq $10, %rdx
- movq $3, %rax
- nop
- nop
- nop
- addq %rdx, %rax

1	2	3	4	5	6	7	8	9	10	11
F	D	E	M	W						
	F	D	E	M	W					
		F	D	E	M	W				
			F	D	E	M	W			
				F	D	E	M	W		
					F	D	E	M	W	

NOP question example (5)

- Clock Cyle
- movq $10, %rdx
- **movq** $3, %rax
- nop
- nop
- nop
- **addq** %rdx, %rax

1	2	3	4	5	6	7	8	9	10	11
F	D	E	M	W						
	F	D	E	M	***W***					
		F	D	E	M	W				
			F	D	E	M	W			
				F	D	E	M	W		
					F	***D***	E	M	W	

- 3 nops solves the issue: The write (W) of **movq** instruction occurs before the decode (D) of the **addq** instruction.

NOP question summary

- Given a list of instructions, you may need to insert NOPs.
- Steps to solve:

1. Outline all the stages for all instructions.
2. Figure out the data dependencies. Does a given instruction need to wait for a prior instruction to finish its write stage?
3. If the answer to the above question is yes, you would need to delay that instruction using some number of nop instructions.

CPE question

Assume there is one of each functional unit, data1 and data2 have the correct types, e.g. int or floating point. Assume acc1, acc2, out1, and out2 can be stored in registers.

(a) **[5 Points]**

```
int acc1, acc2;
    for (i=0; i< length; i++){
        acc1 = acc1 + data1[i];
        acc2 = acc2 * data2[i];
    }
```

What is the CPE of this loop?

(b) **[5 Points]**

```
float out1, out2, acc1, acc2;
    for (i=0; i< length; i++){
        out1 = acc1 + data1[i];
        out2 = acc2 * data2[i];
    }
```

What is the CPE of this loop?

CPE question (Assumptions)

Operation	Latency	Issue Time/Rate
Integer Add	1	1
Integer Multiply	4	1
Floating Point Add	3	1
Floating Point Multiply	5	2
Load or Store (Cache Hit)	1	1

CPE question: Latency and Issue rate

- Latency is the number of cycles the instruction takes to finish once it starts
- Issue time is the minimum number of cycles required before we can reissue the same instruction.

CPE question (a): Data dependecies across iterations

Cycles	Iteration 1			Iteration 2		
1	load		incl			
2	add (int)	load	cmpl			
3		mul (int)	jmp	load		incl
4		mul (int)		add (int)		cmpl
5		mul (int)				jmp
6		mul (int)			load	
7					mul (int)	
8					mul (int)	
9					mul (int)	
10					mul (int)	

CPE question (a): Data dependecies across iterations

Cycles	Iteration 1			Iteration 2		
1	**add (int)**	**mul (int)**				
2		**mul (int)**		**add (int)**		
3		**mul (int)**				
4		**mul (int)**				
5					**mul (int)**	
6					**mul (int)**	
7					**mul (int)**	
8					**mul (int)**	

Note: If we ignore load and inc, cmp, jmp instructions. Typically inc, cmp, jmp can be ignored. If latency, issue rate are mentioned for load, DO NOT ignore loads.

[CPE = number of these blocks] Thus in this case, **CPE = 4**

CPE question (b): No data dependencies

Cycles	Iteration 1			Iteration 2		
1	load		incl			
2	add (float)	load	cmpl			
3	add (float)	mul (float)	jmp	load		incl
4	add (float)	mul (float)		add (float)	load	cmpl
5		mul (float)		add (float)	mul (float)	jmp
6		mul (float)		add (float)	mul (float)	
7		mul (float)			mul (float)	
8					mul (float)	
9					mul (float)	

[CPE = number of these blocks] Thus in this case, **CPE = 2**

CPE question (b): No data dependencies

Cycles	Iteration 1			Iteration 2		
1	**add (float)**	**mul (float)**				
2	**add (float)**	**mul (float)**		**add (float)**		
3	**add (float)**	**mul (float)**		**add (float)**	**mul (float)**	
4		**mul (float)**		**add (float)**	**mul (float)**	
5		**mul (float)**			**mul (float)**	
6					**mul (float)**	
7					**mul (float)**	

Note: If we ignore load and inc, cmp, jmp instructions. Typically inc, cmp, jmp can be ignored. If latency, issue rate are mentioned for load, DO NOT ignore loads.

[CPE = number of these blocks] Thus in this case, **CPE = 2**

CPE question: Some thoughts

- Though we have shown just 2 iterations--- in theory we should be writing out these diagrams for a few more number of iterations

- **For instance--- think about what happens if the floating add operation in part (b) had a issue time of 3 cycles as opposed to 1.**

- **CPE might increase? Let me know ☺**