Unit 17

Improving Performance
Caching and Pipelining

Improving Performance

- We want to improve the performance of our computation
- Question: What are we referring to when we say "performance"?
  - Speed
  - Energy consumption
  - Cost?
- We will primarily consider **speed** in this discussion

How Do We Measure Speed

- **Fundamental Measurement**: TIME
  - Absolute time from start to finish
  - To compare two alternative systems (HW + SW) and their performance, start a timer when you begin a task and stop it when the task ends
  - Do this for both systems and compare the resulting times
- We call this the **latency** of the system and it works great from the perspective of the single, overall task
  - If system A completes the task in 2 seconds and system B requires 3 seconds, then system A is clearly superior
- But when we dig deeper and realize that the single, overall task is likely made of many small tasks, we can consider more than just latency

Performance Depends on View Point?!

- What's faster to get from point A to point B?
  - A 747 Jumbo Airliner
  - An F-22 supersonic, fighter jet
- If only 1 person needs to get from point A to point B, then the F-22
  - This is known as latency [units of seconds]
  - Time from the start of an operation until it completes
- If 200 people need to get from point A to point B, the 747 looks much better
  - This is known as throughput [jobs/second]
- The ***overall*** execution time (latency) may best be improved by increasing throughput and not the latency of individual tasks

Improving Latency and Throughput

CACHING AND PIPELINING

Hardware Techniques

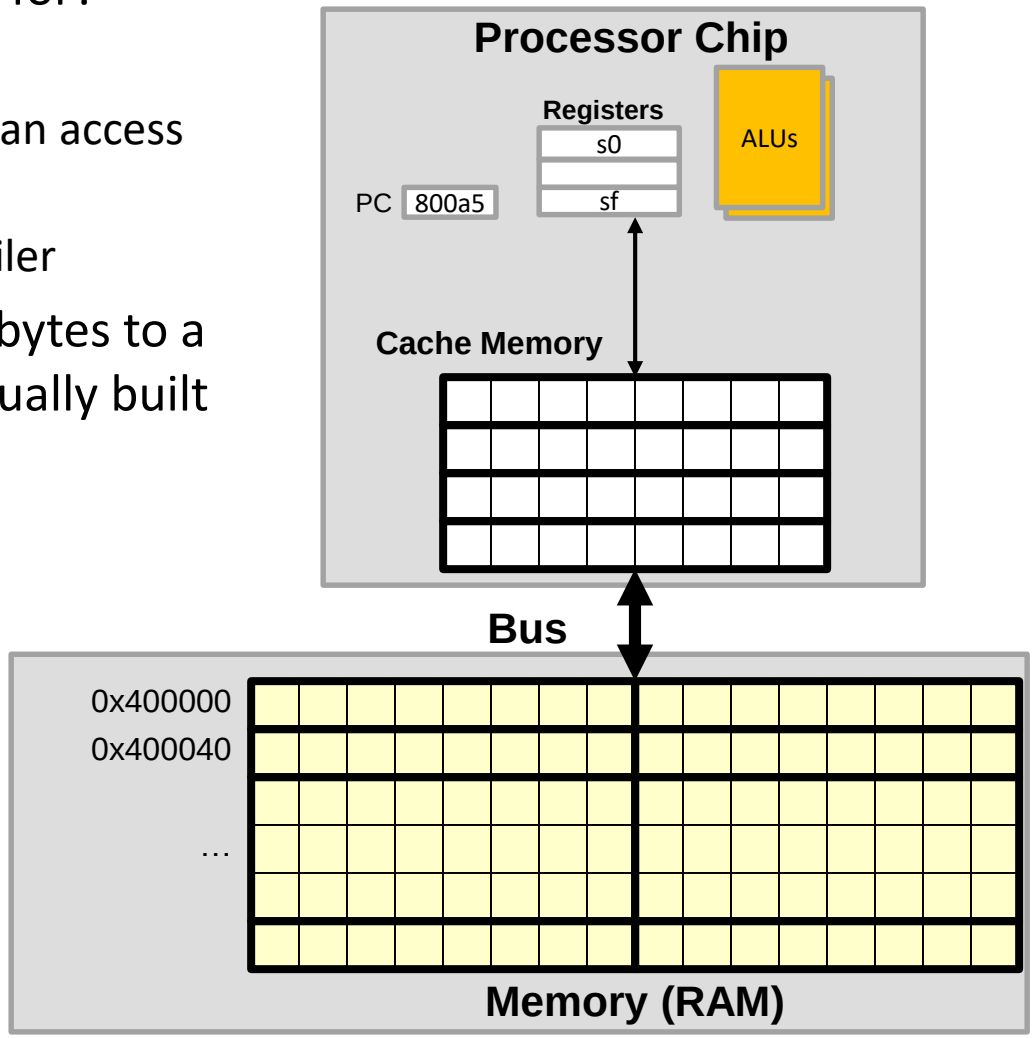
- We can add hardware or reorganize our hardware to improve throughput and latency of individual tasks in an effort to reduce the total latency (time) to finish the overall task
- We will look at two examples:
  - Caching: Improves latency
  - Pipelining: Improves throughput

Caching

- **Cache (def.)** – "*to store away in hiding or for future use*"
- Primary idea
  - The first time you access or use something you expend the full amount of time to get it
  - However, store it someplace (i.e. in a cache) you can get it more quickly the next time you need it
  - The next time you need something check if it is in the cache first
  - If it is in the cache, you can get it quickly; else go get it expending the full amount of time (but then save it in the cache)
- Examples:
  - Web-browser
  - Checking out a book from the library
  - Your refrigerator

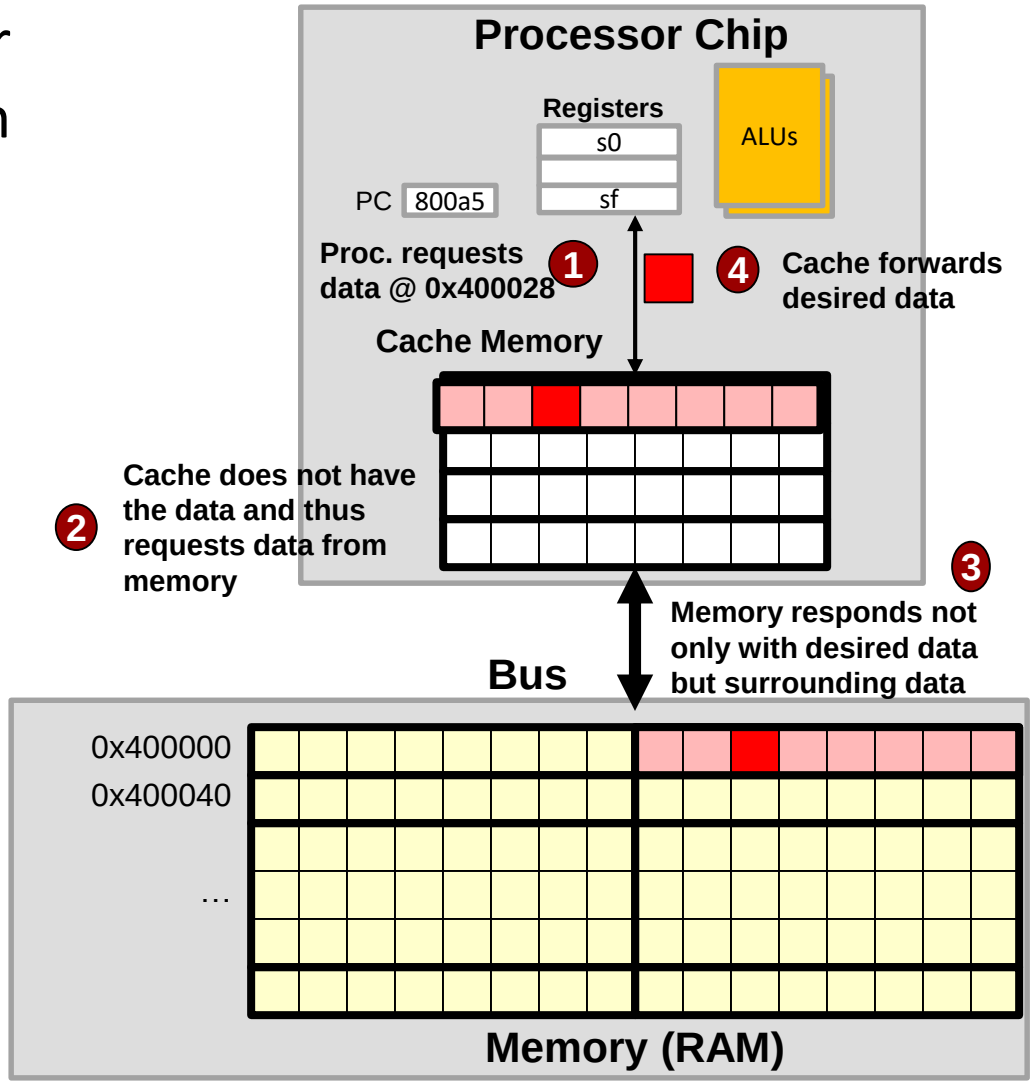
Cache Overview

- Remember what register are used for?
  - Quick access to copies of data
  - Only a few (32 or 64) so that we can access really quickly
  - Controlled by the software/compiler
- Cache memory is a small-ish, (kilobytes to a few megabytes) "fast" memory usually built onto the processor chip
- Will hold copies of the latest data & instructions accessed by the processor
- Managed by the HW
  - Transparent to the software



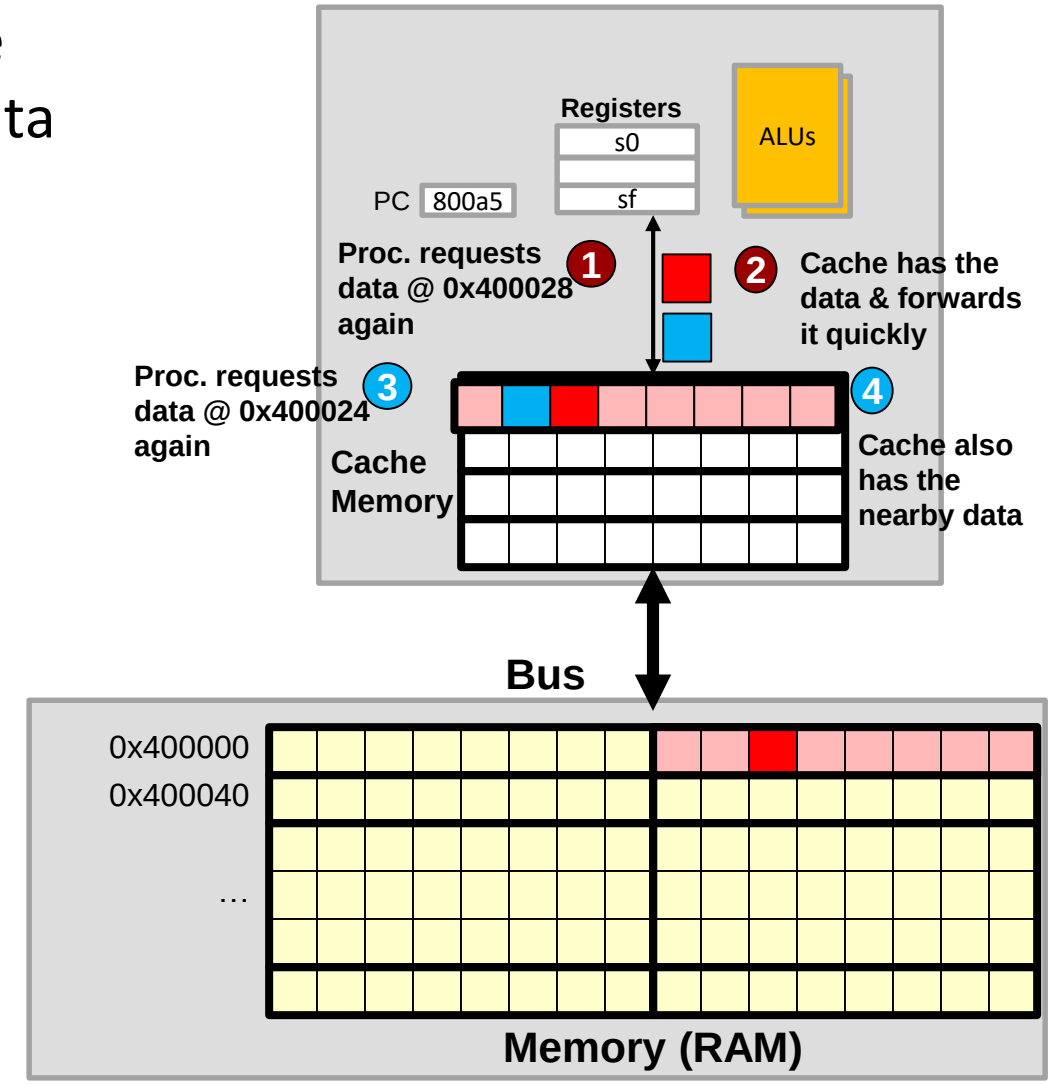
Cache Operation (1)

- When processor wants data or instructions it always checks in the cache first
- If it is there, fast access
- If not, get it from memory
- Memory will also supply surrounding data since it is likely to be needed soon
  - Why?
  - Things like arrays & code (instructions) are commonly accessed sequentially



Cache Operation (2)

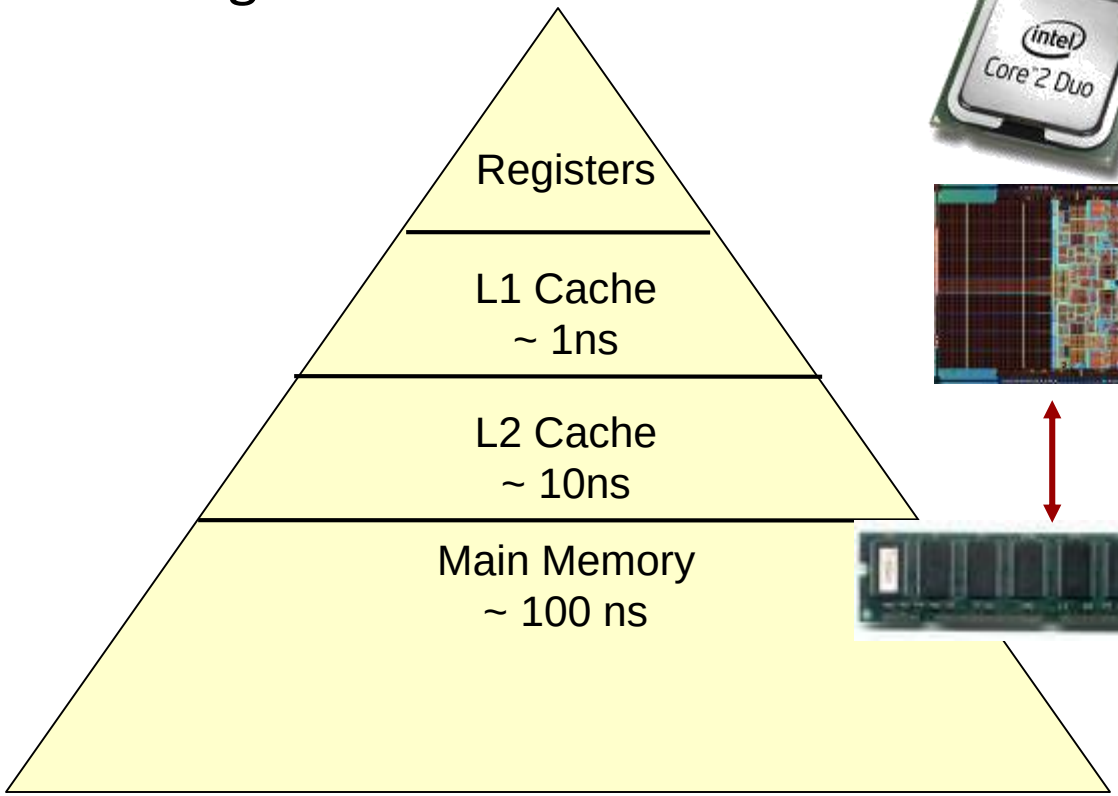
- When processor asks for the data again or for the next data value in the array (or instruction of the code) the cache will likely have it
- Questions?



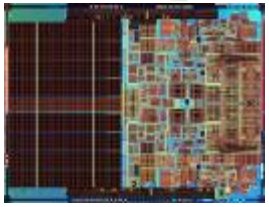
Main point: Caching reduces the latency of memory accesses which improves overall program performance.

Memory Hierarchy & Caching

- Use several levels of faster and faster memory to hide latency of larger levels



Unit of Transfer:
8- to 64- bits



Unit of Transfer:
8-64 bytes

Smaller — More Expensive — Faster

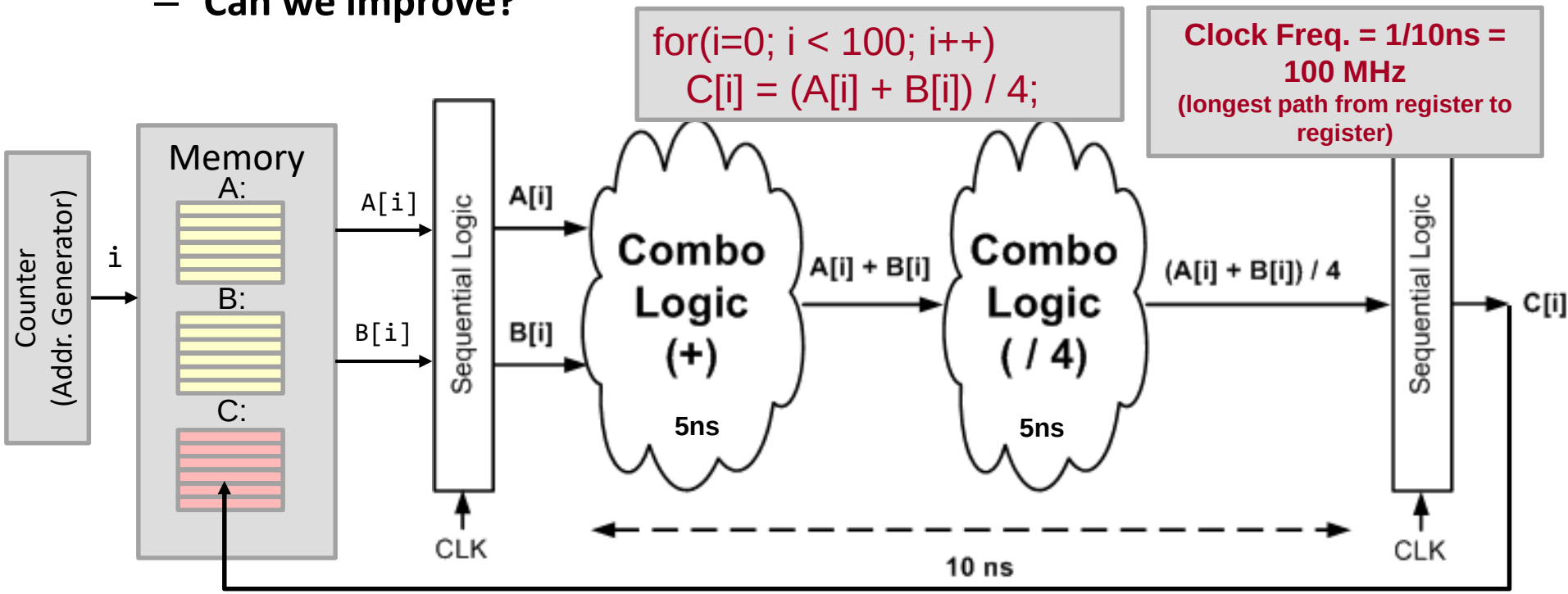
Larger — Less Expensive — Slower

Pipelining

- We'll now look at a hardware technique called **pipelining** to improve throughput
- The key idea is to **overlap** the processing of multiple "items" (either data or instructions)

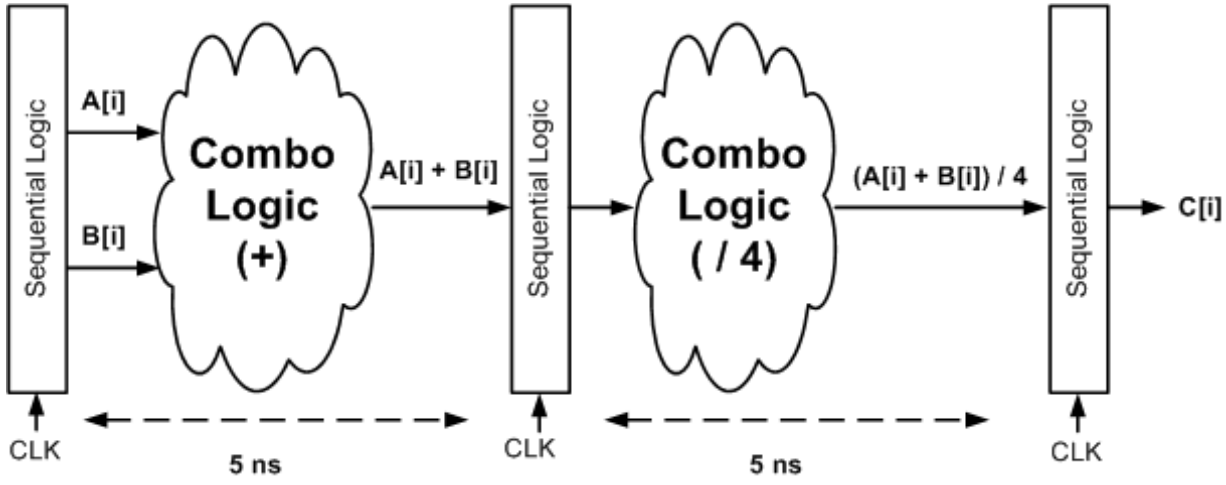
Example

- Suppose you are asked to build dedicated hardware to perform some operation on all 100 elements of some arrays
- Suppose the operation (A[i]+B[i])/4 takes 10 ns to perform
- How long would it take to process the entire arrays: **1000 ns**
  - **Can we improve?**



Pipelining Example

- Pipelining refers to insertion of registers to split combinational logic into smaller stages that can be overlapped in time (i.e. create an assembly line)



Stage 1 Stage 2

	Stage 1	Stage 2
Clock Cycle 0	A[0] + B[0]	
Clock Cycle 1	A[1] + B[1]	(A[0] + B[0]) / 4
Clock Cycle 2	A[2] + B[2]	(A[1] + B[1]) / 4

```
for(i=0; i < 100; i++)
   C[i] = (A[i] + B[i]) / 4;
```

Time for 0th elements to complete: 10ns

Time between each of the remaining 99 element completing: 5ns

Total: 10 + 99*5 = 505ns

Define:

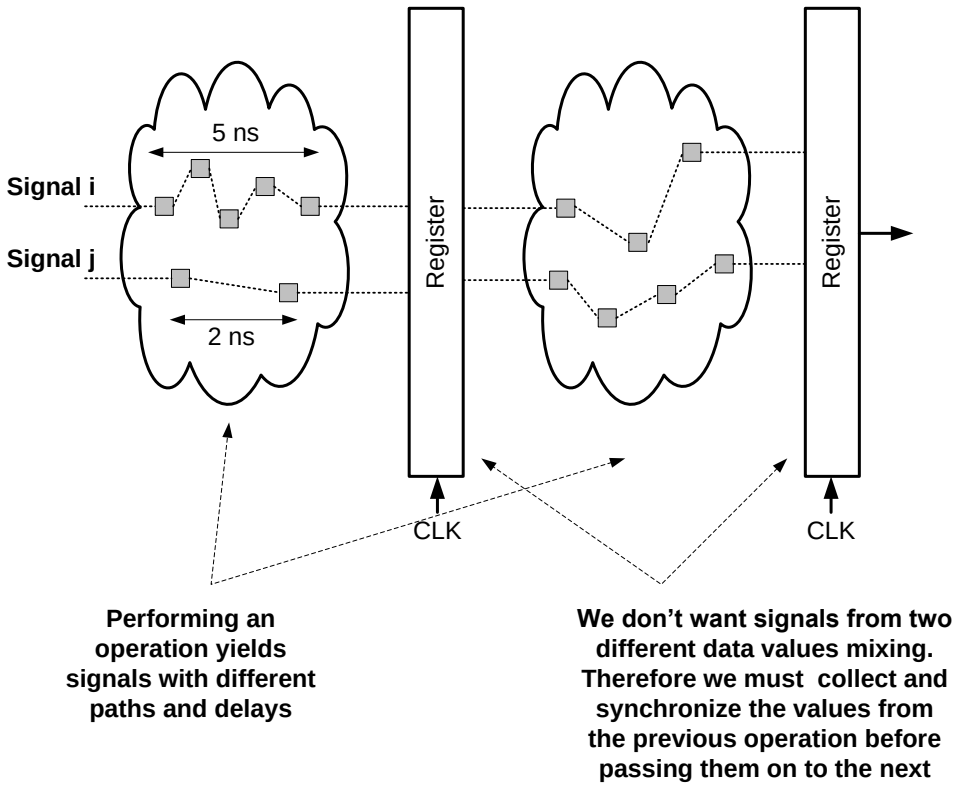
$$speedup = \frac{Original\ time}{Improved\ Time}$$

$$speedup = \frac{1000ns}{505ns} \approx 2x$$

Clock freq: = 200MHz = 1/5ns

Need for Registers

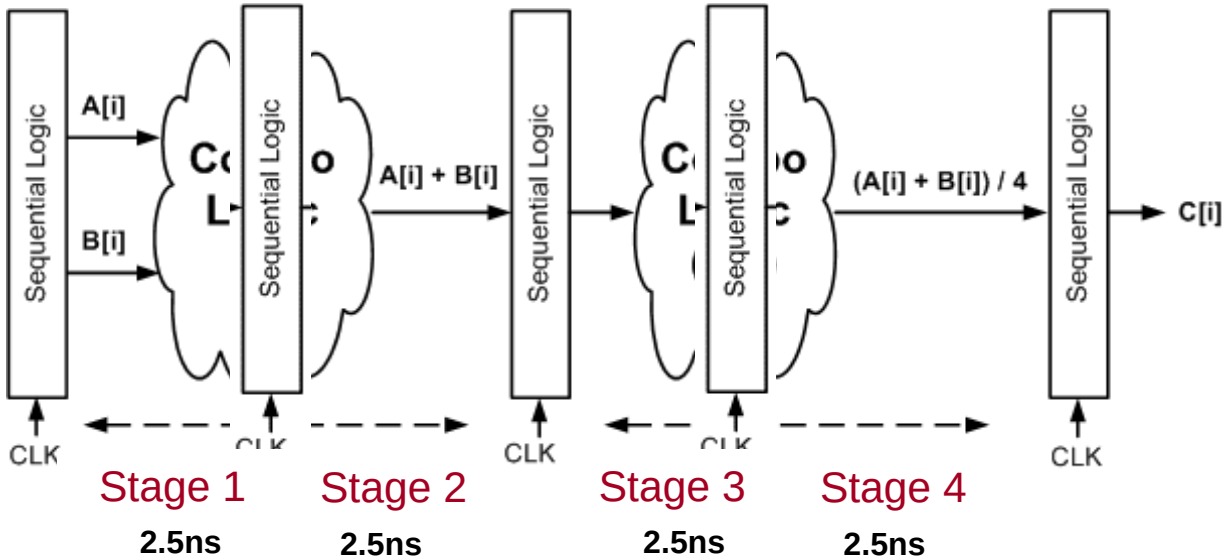
- Provides separation between combinational functions
  - Without registers, fast signals could "catch-up" to data values in the next operation stage



Pipelining Example

- By adding more pipelined stages we can improve throughput
- Have we affected the latency of processing individual elements? No!!!
- Questions/Issues?
  - Balancing stage delays
  - Overhead of registers (Not free to split stages)
    - This limits how much we can split our logic

```
for(i=0; i < 100; i++)
   C[i] = (A[i] + B[i]) / 4;
```



Time for 0th elements to complete: 10 ns

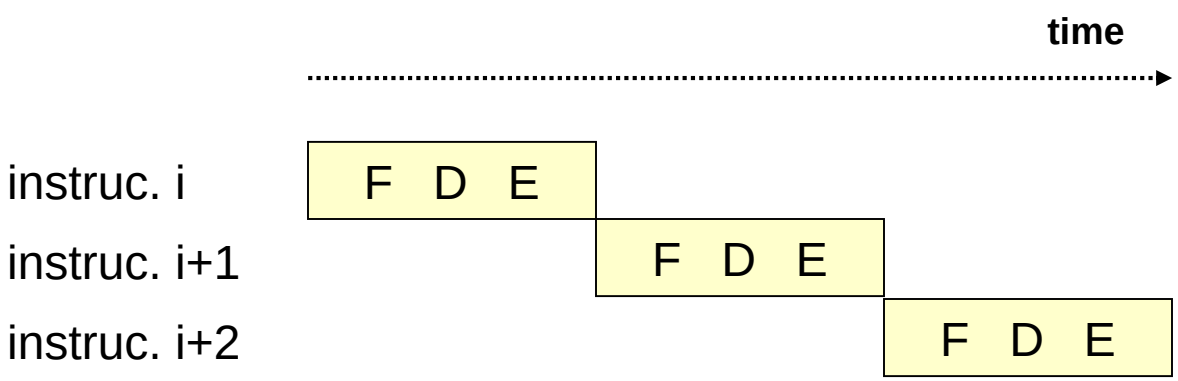
Time between each of the remaining 99 element completing: 2.5 ns

Total: 257.5 ns

$$speedup = \frac{1000ns}{257.5ns} \approx 4x$$

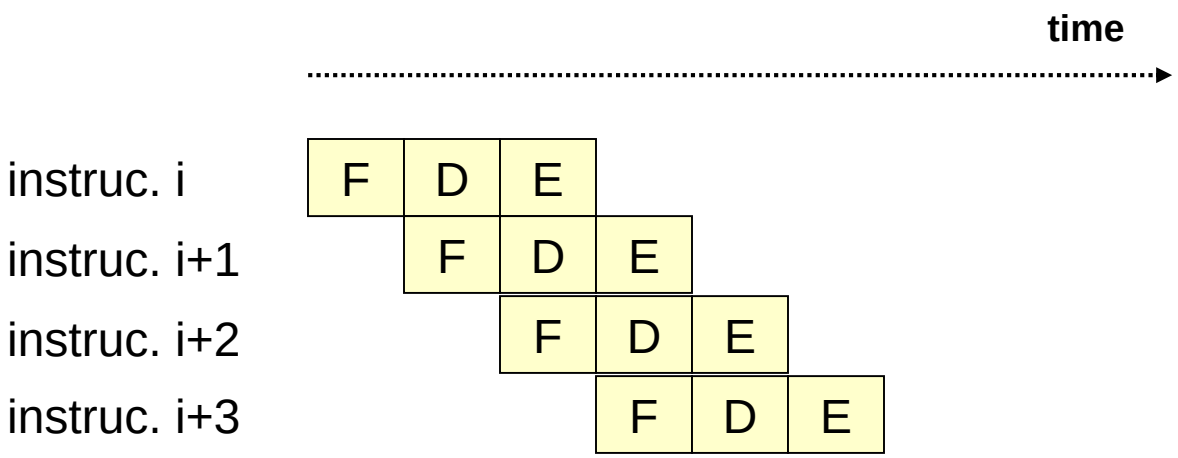
Non-Pipelined Processors

- Currently we know our processors execute software 1 instruction at a time
- 3 steps/stages of work for each instruction are:
  - Fetch
  - Decode
  - Execute



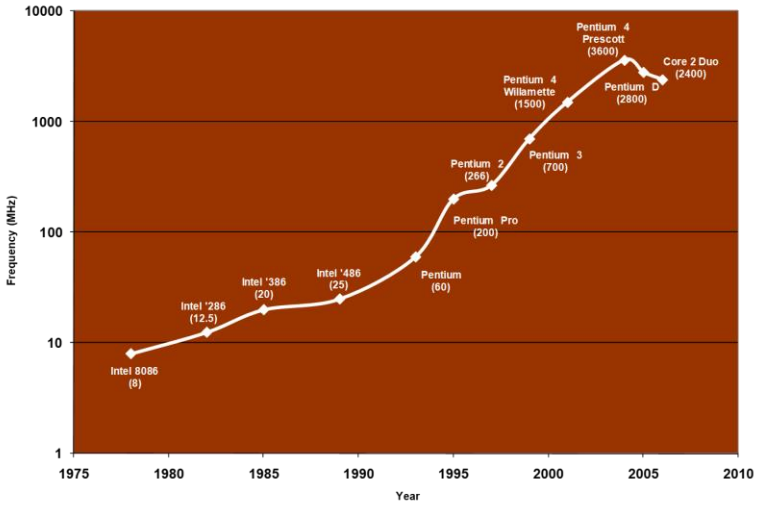
Pipelined Processors

- By breaking our processor hardware for instruction execution into stages we can overlap these stages of work
- Latency for a single instruction is the same
- Overall throughput, and thus total latency, are greatly improved

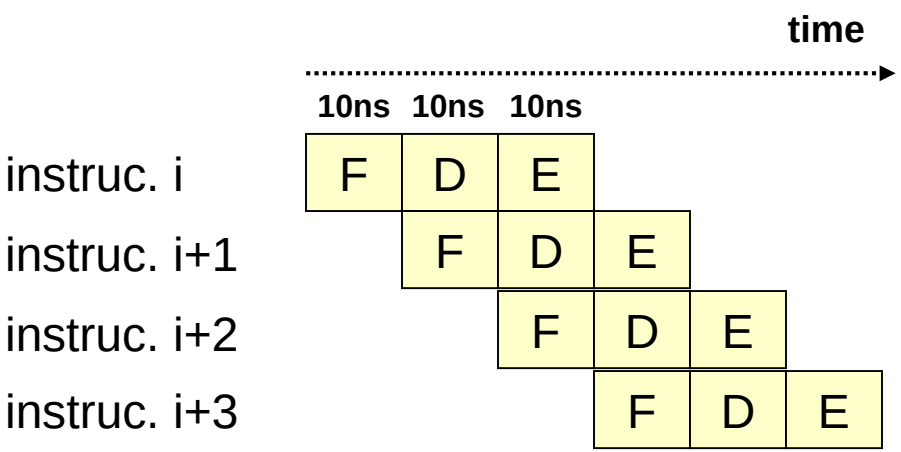


More and More Stages

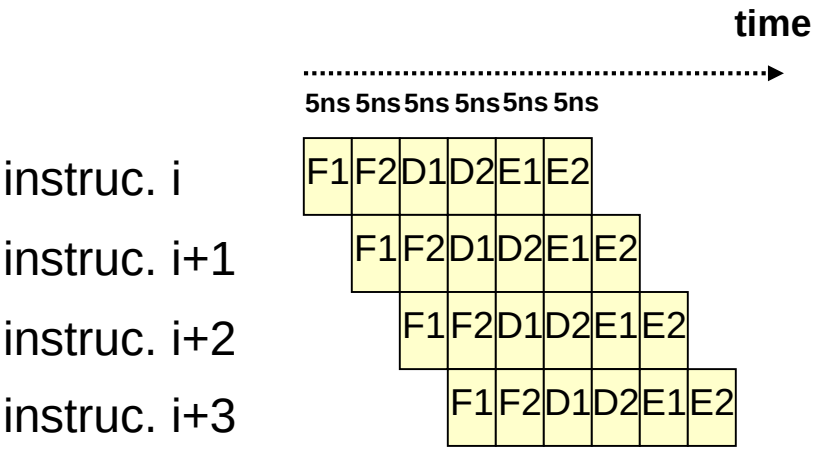
- We can break the basic stages of work into substages to get better performance
- **In doing so our clock period goes down; frequency goes up**
- All kinds of interesting issues come up though when we overlap instructions and are discussed in future CENG courses



Clock freq. = 1/10ns = 100MHz



Clock freq. = 1/5ns = 200MHz



Summary

- By investing extra hardware we can improve the overall latency of computation
- Measures of performance:
  - Latency is start to finish time
  - Throughput is tasks completed per unit time (measure of parallelism)
- Caching reduces latency by holding data we will use in the future in quickly accessible memory
- Pipelining improves throughput by overlapping processing of multiple items (i.e. an assembly line)