Week 6th

Prepared by Dr Syed Khaldoon Khurshid

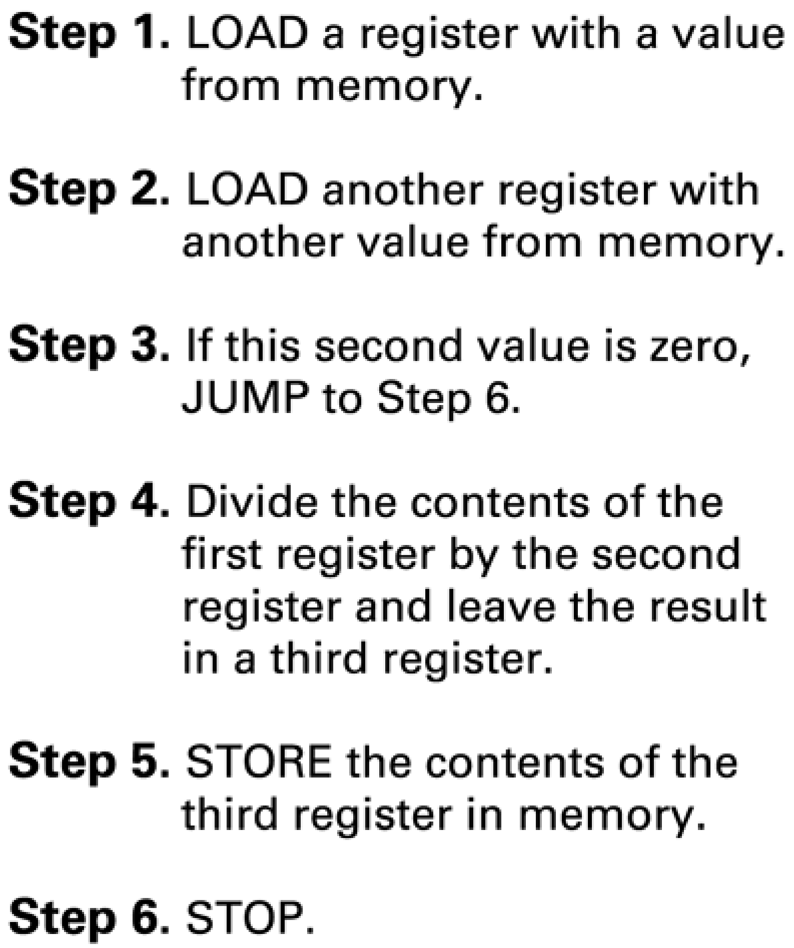
2.2: Machine Language

- To apply the stored-program concept:
  - CPUs must recognize encoded instructions
- Collection of instructions known as:
  - *Instruction set*, or
  - *Machine language*
- Essential set of instructions quite small
  - Additional instructions do not increase capabilities

2.2: Instruction categories

- Machine instructions classified into 3 categories:
  - (1) Data Transfer
    - move/copy data from one location to another
    - typical examples: LOAD, STORE
  - (2) Arithmetic/Logic
    - perform actual operations on data
    - typical examples: AND, OR, XOR, SHIFT, ROTATE
  - (3) Control
    - manipulate the execution of a program
    - typical examples: JUMP, HALT

2.2: Example: dividing 2 values stored in memory



Step 1. LOAD a register with a value from memory.

Step 2. LOAD another register with another value from memory.

Step 3. If this second value is zero, JUMP to Step 6.

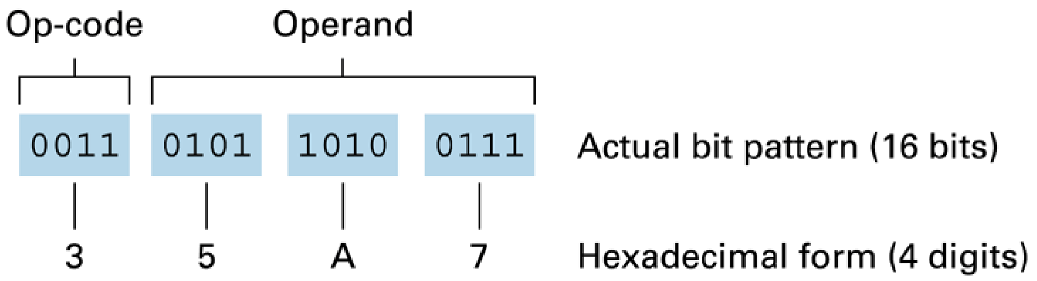
Step 4. Divide the contents of the first register by the second register and leave the result in a third register.

Step 5. STORE the contents of the third register in memory.

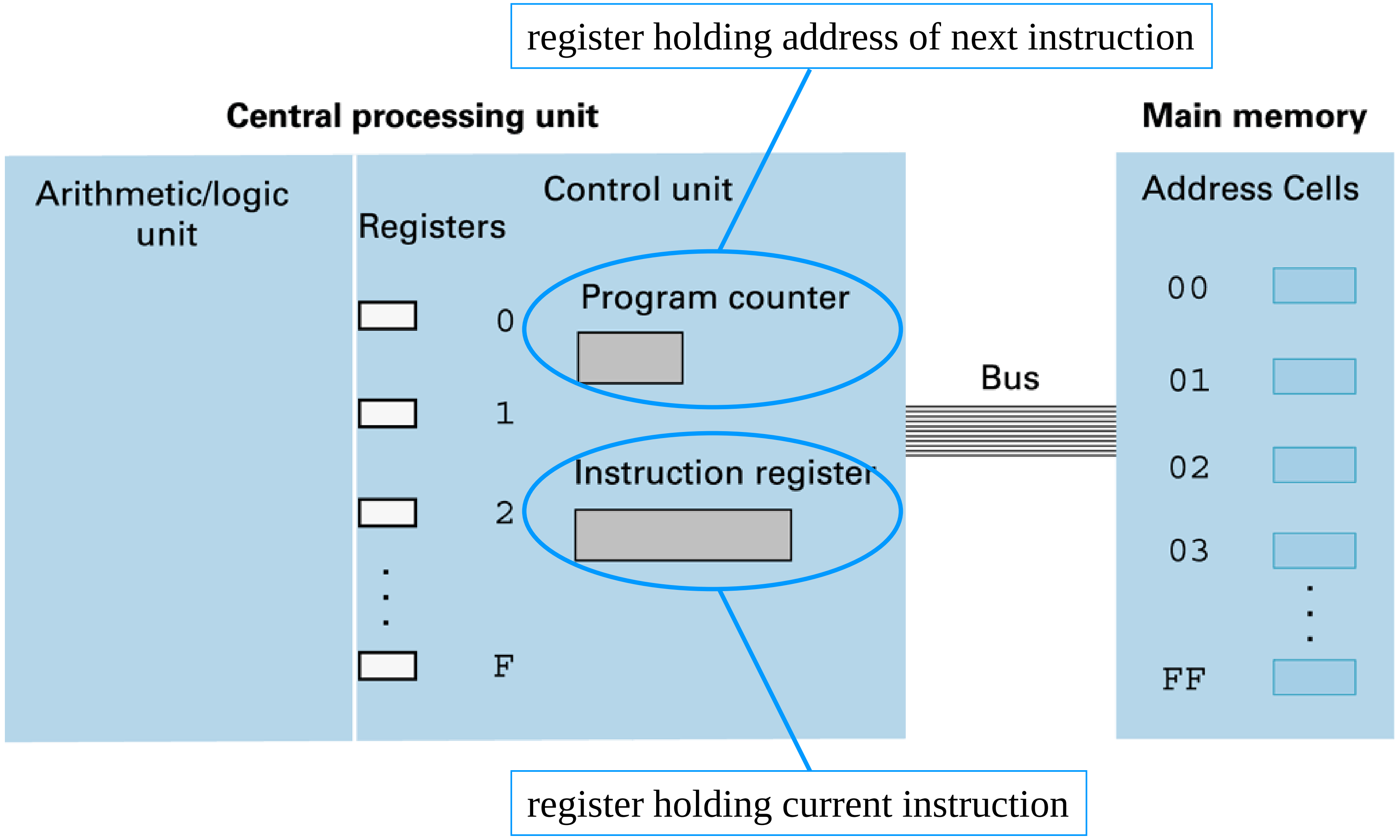
Step 6. STOP.

2.2: Composition of Machine Instructions

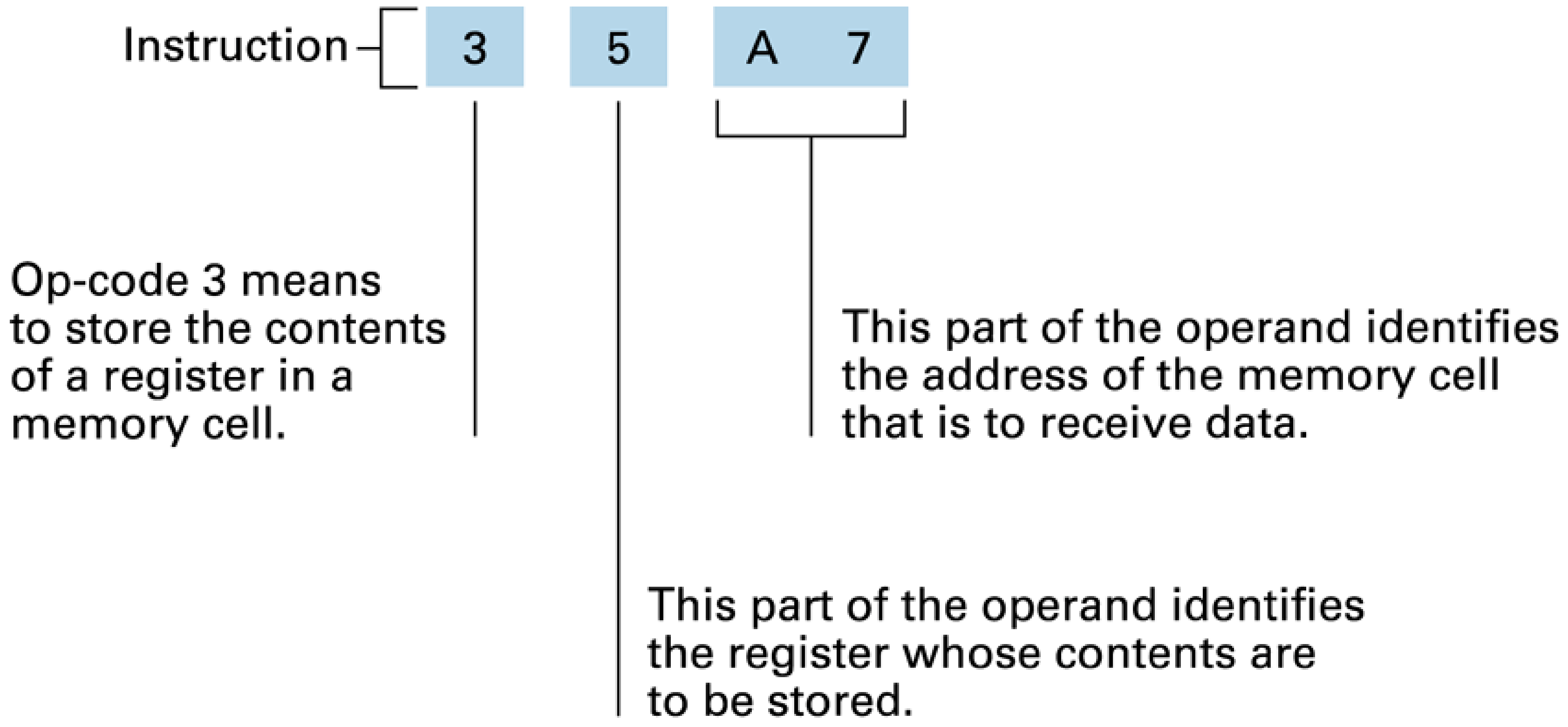
- Machine instructions typically consist of 2 parts:
  - (1) Op-code (operation code)
    - indicates which operation (such as STORE, SHIFT, XOR) is requested by the instruction
  - (2) Operand
    - provides more information about the operation specified by the op-code (e.g. registers/memory cells to be used)



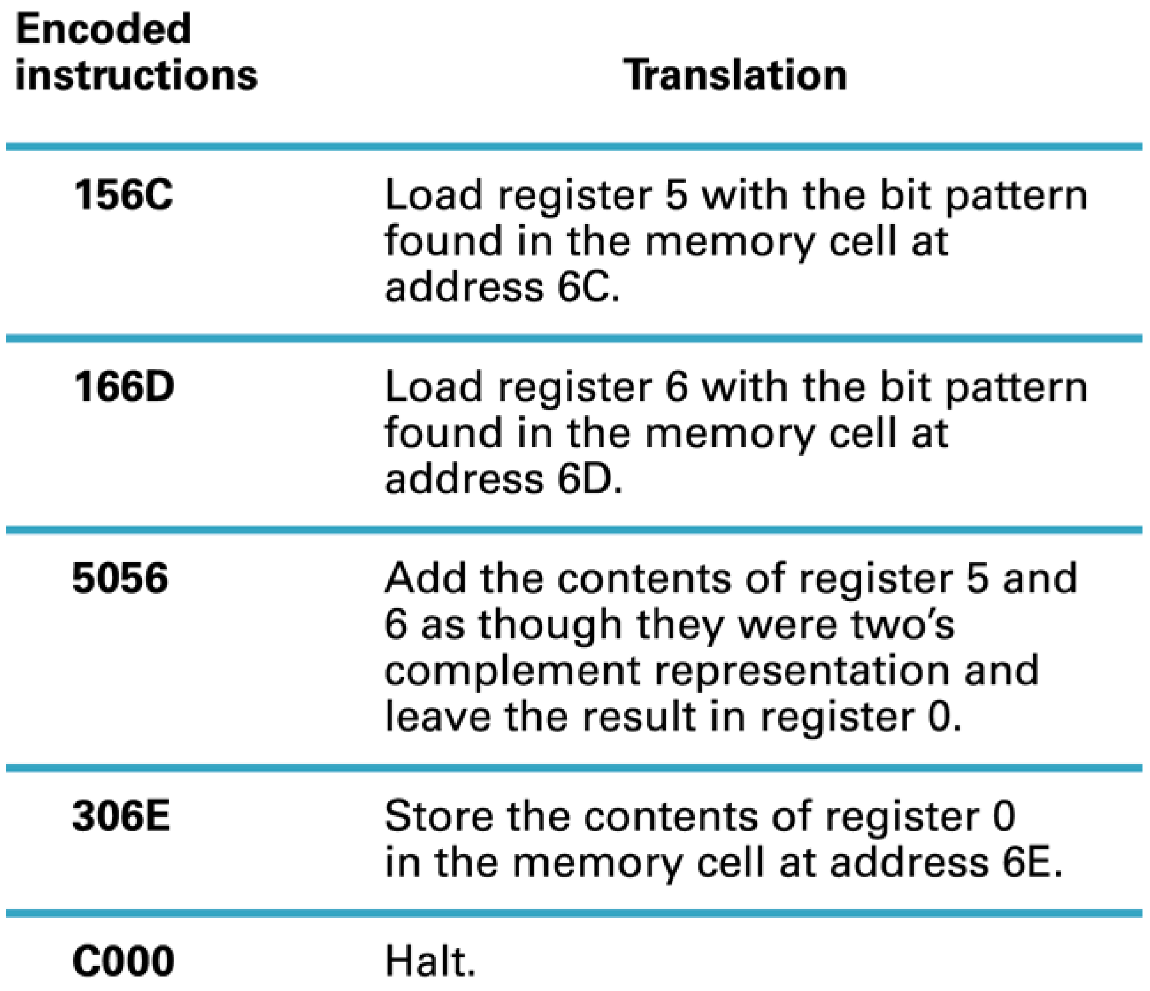
2.2: Simple Machine Architecture (Appendix C)



2.2: Example: decoding instruction 35A7 (App. C)

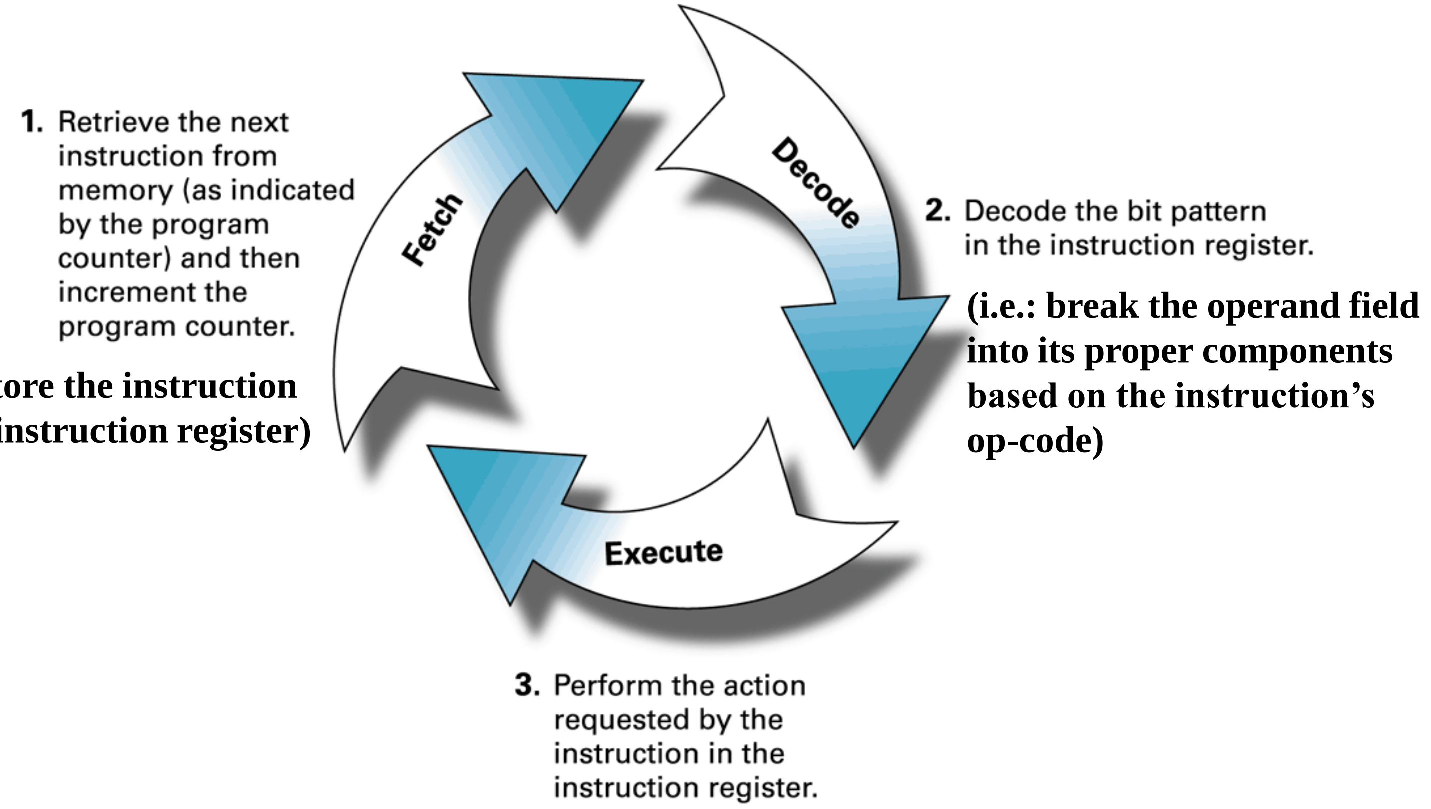

Instruction
3
5
A
7
Op-code 3 means to store the contents of a register in a memory cell.
This part of the operand identifies the address of the memory cell that is to receive data.
This part of the operand identifies the register whose contents are to be stored.

2.2: Adding two values stored in main memory



Encoded instructions	Translation
156C	Load register 5 with the bit pattern found in the memory cell at address 6C.
166D	Load register 6 with the bit pattern found in the memory cell at address 6D.
5056	Add the contents of register 5 and 6 as though they were two's complement representation and leave the result in register 0.
306E	Store the contents of register 0 in the memory cell at address 6E.
C000	Halt.

2.3: The Machine Cycle

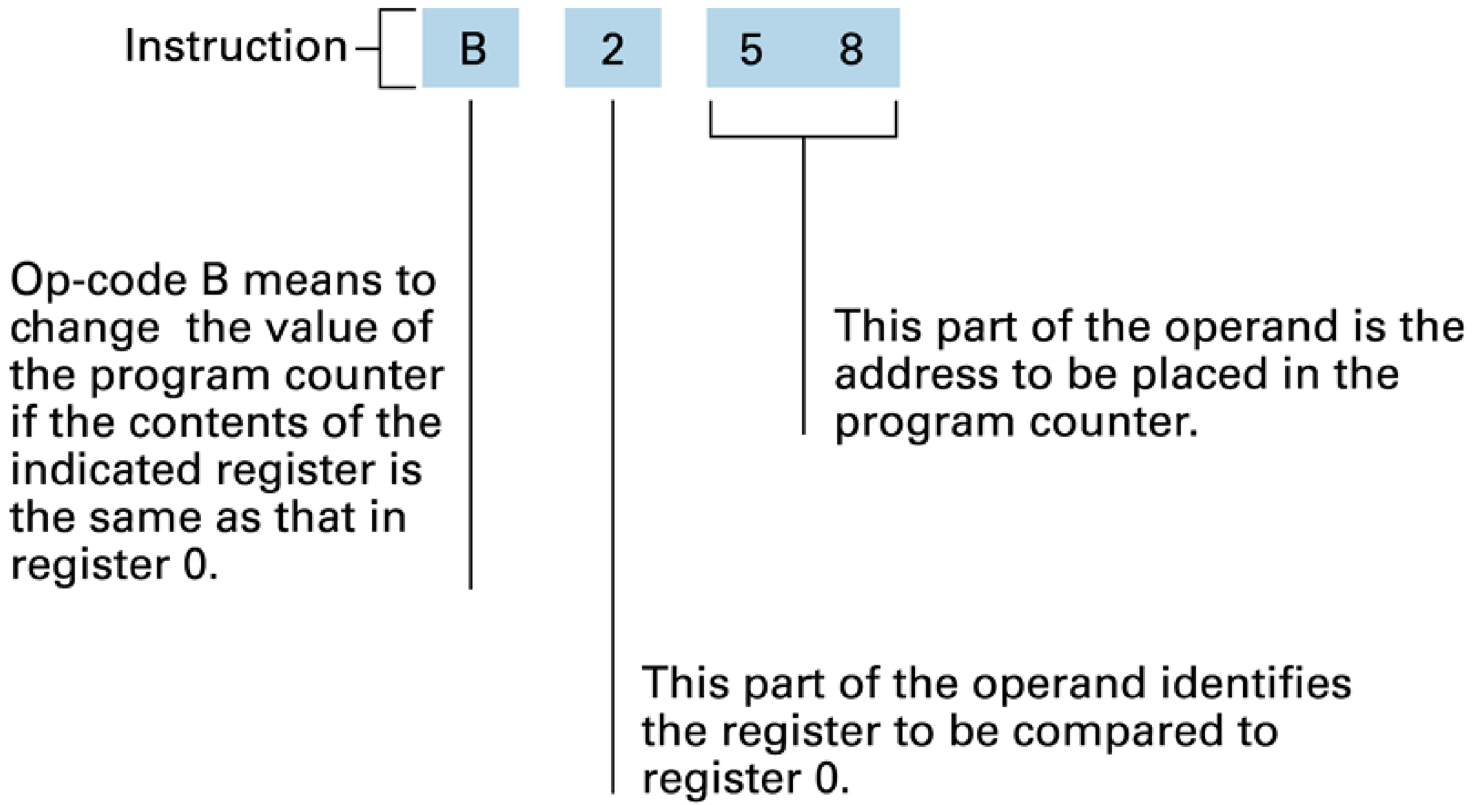

Fetch
Decode
Execute
1. Retrieve the next instruction from memory (as indicated by the program counter) and then increment the program counter.
2. Decode the bit pattern in the instruction register.
3. Perform the action requested by the instruction in the instruction register.

(and store the instruction in the instruction register)

(i.e.: break the operand field into its proper components based on the instruction's op-code)

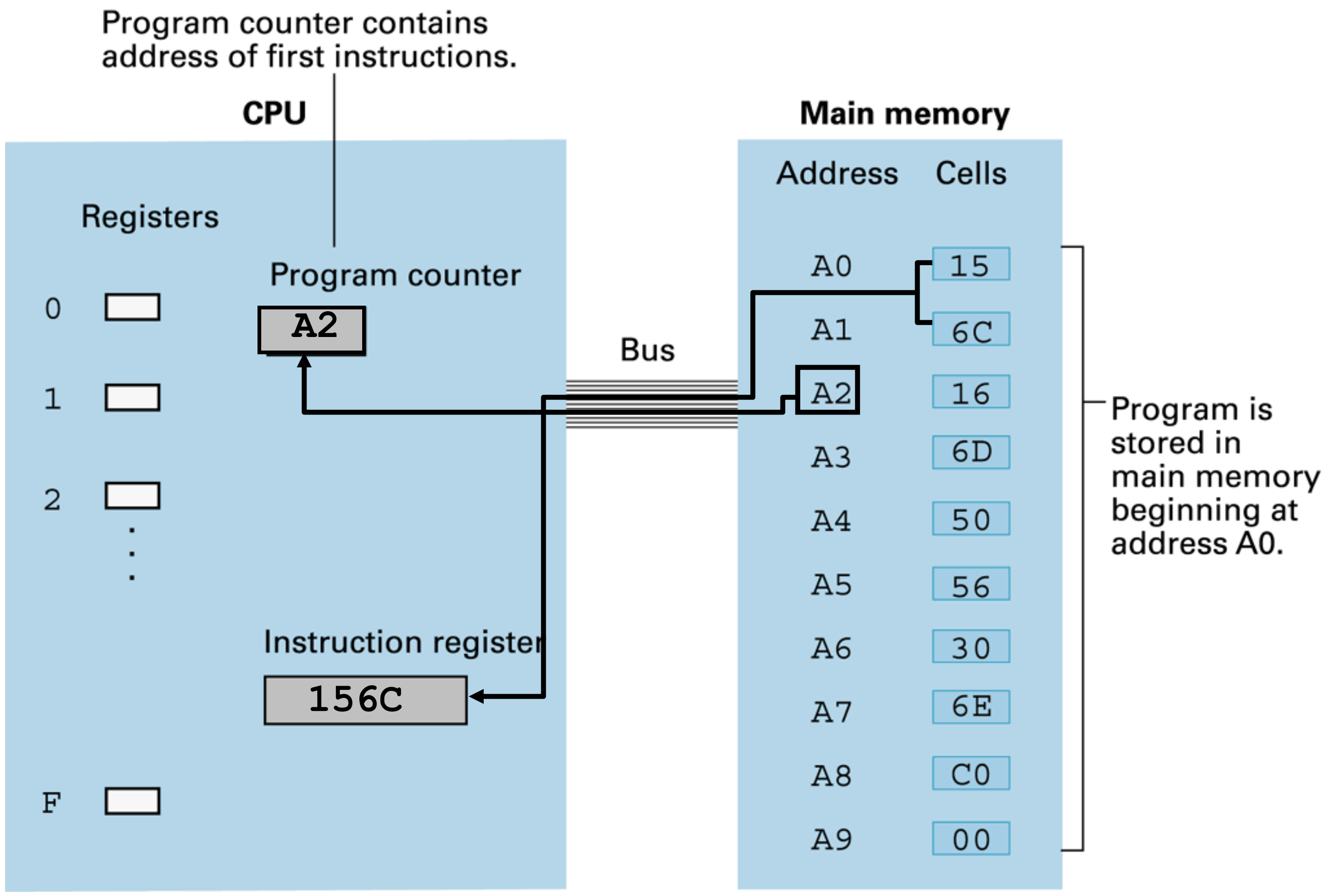
- Continually repeated by Control Unit until HALT
- Special case: JUMP (i.e: may change program counter)

2.3: Decoding JUMP instruction B258


Instruction
B
2
5 8
Op-code B means to change the value of the program counter if the contents of the indicated register is the same as that in register 0.
This part of the operand is the address to be placed in the program counter.
This part of the operand identifies the register to be compared to register 0.

- Translates as follows:
  - If contents registers 0 & 2 equal: place 58 in program counter
  - Otherwise: do nothing

2.3: Example of Program Execution (Fetch)



2.4: Arithmetic/Logic Instructions

- Boolean operations:

```
     10011010          10011010          10011010
AND  11001001     OR   11001001     XOR  11001001
-------------     -------------     -------------
     10001000          11011011          01010011
```

- AND used to force 0's into certain positions:

```
     00001111
AND  10101010
-------------
     00001010
```

- OR used to force 1's into certain positions:

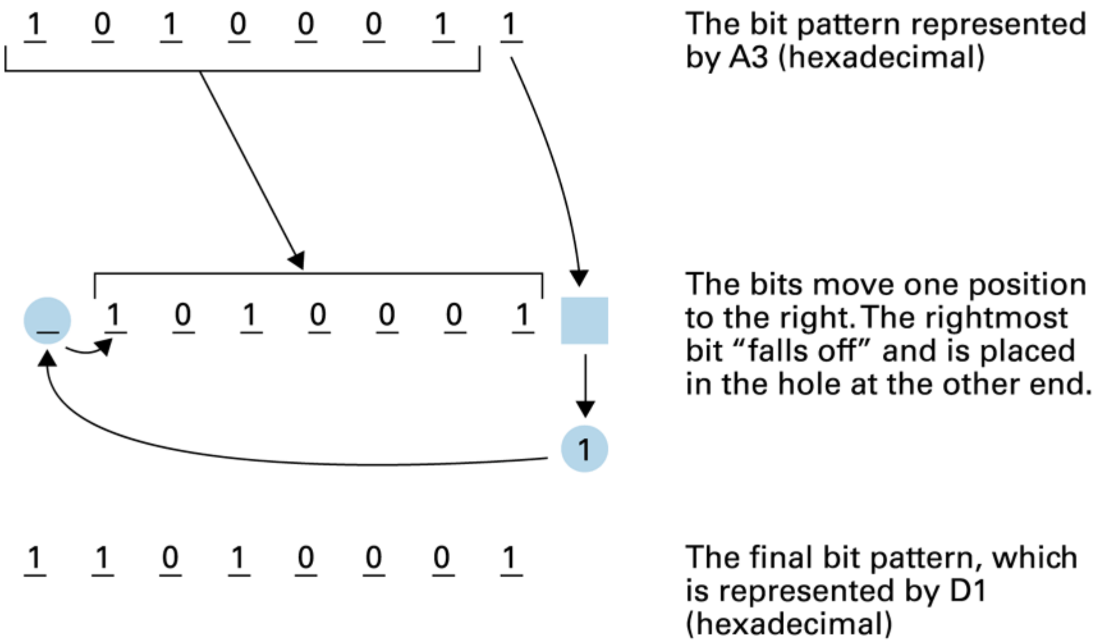
```
     00001111
OR   10101010
-------------
     10101111
```

- XOR used to form bit-string complement:

```
     11111111
XOR  10101010
-------------
     01010101
```

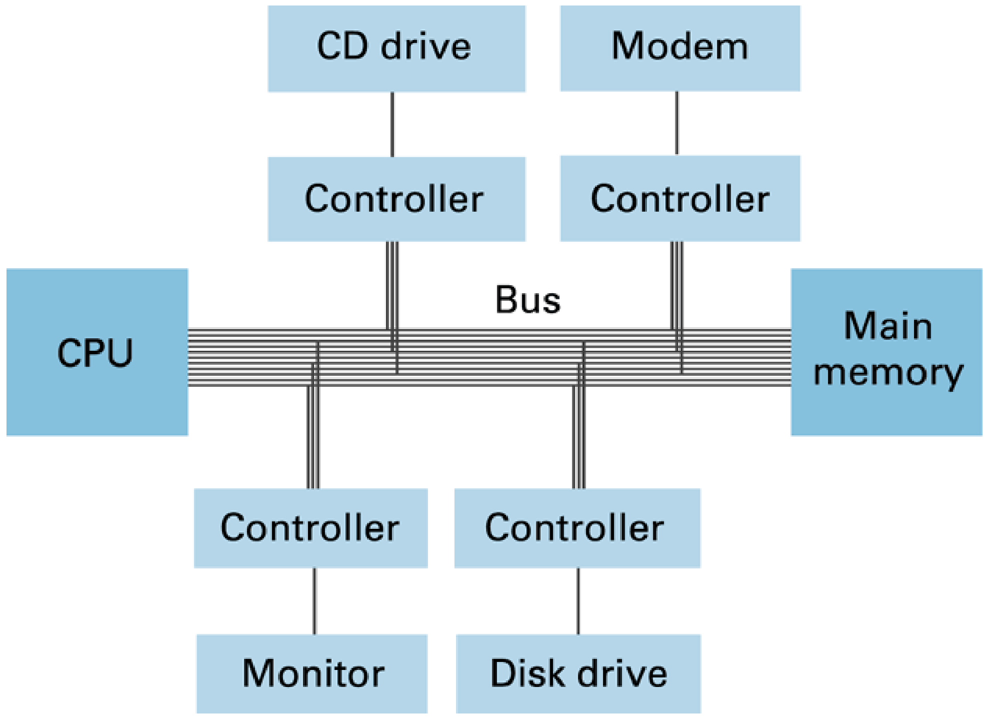
2.4: Arithmetic/Logic Instructions (cont'd)

- Rotation and Shift:
  - Move bits within a register to left or right
  - e.g. for obtaining the mantissa in floating-point values
- Example:


1 0 1 0 0 0 1 1
The bit pattern represented by A3 (hexadecimal)
1 0 1 0 0 0 1
The bits move one position to the right. The rightmost bit "falls off" and is placed in the hole at the other end.
1
1 1 0 1 0 0 0 1
The final bit pattern, which is represented by D1 (hexadecimal)

2.5: Communication with other devices

- Handled through intermediary device: ‘*controller*’
- CPU communicates with controllers in the same way that it communicates with main memory



2.6: ‘Von Neumann Architecture’- Problem

- A problem of speed……!
  - electric pulses can go no faster than speed of light (approx. 30 cm per nanosecond)
- For average machine:
  - fetch-decode-execute cycle takes several nano secs.
- So: increasing speed becomes problem of scale
  - small, smaller, … stuck
- In other words: there appears to be a speed limit!

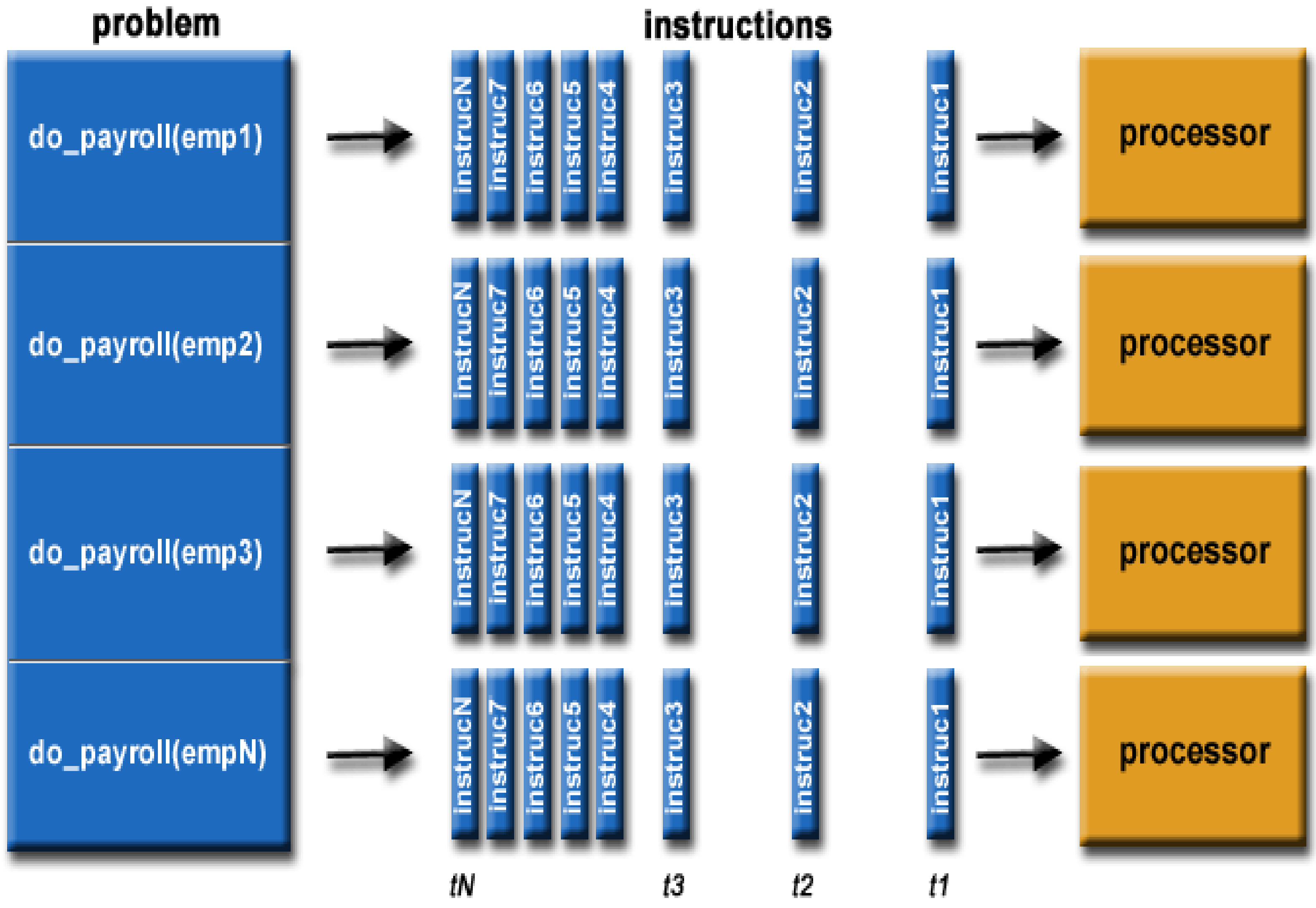
2.6: ‘Von Neumann Architecture’- Alternatives (1)

- One solution: Pipelining
  - allows steps in the machine cycle to overlap
  - during execution of one instruction, next is fetched
  - so: more than one instruction is ‘in the pipe’
- Result: same speed - but higher *throughput*
- Problems with JUMP instructions, of course…
  - pre-fetching becomes useless
- Modern CPUs:
  - fetch & execute multiple instructions at a time

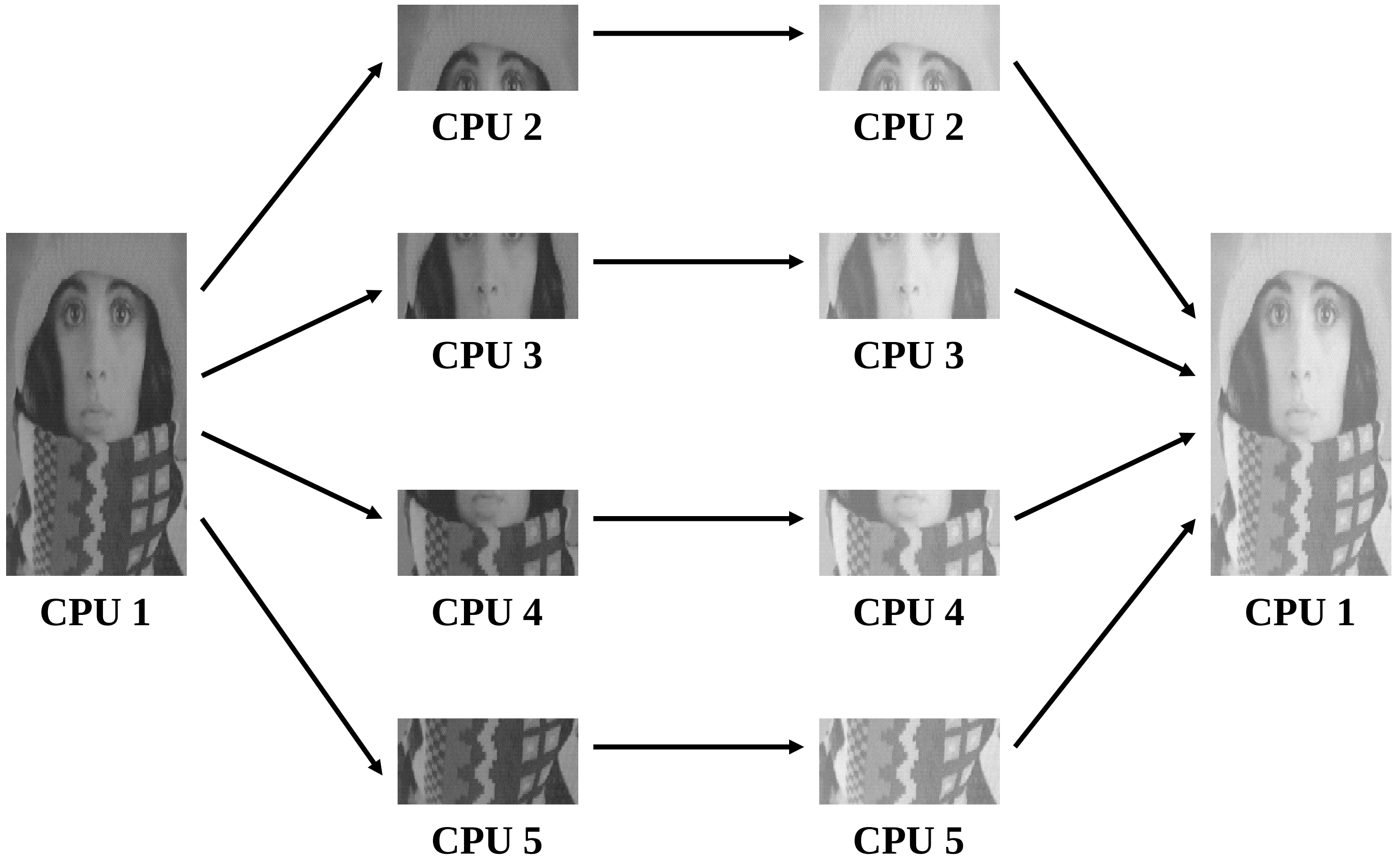
2.6: ‘Von Neumann Architecture’- Alternatives (2)

- Other solution: (true) Parallel Processing
  - performing several tasks at the same time
  - needs multiple processing units
- For example:
  - several complete PCs connected via fast network
    - each PC capable of independent execution
  - …

2.6: Parallel Processing: Example



- **Parallel Image Processing:**



Chapter 2 - Data Manipulation: Conclusions

- Today’s computers based on ‘Von Neumann’ Architecture
  - CPU connected to main memory via bus
- Both program & data stored in main memory
- Program execution based on Machine Cycle
  - fetch - decode - execute
- Peripheral devices connected via controllers
- Alternatives to ‘Von Neumann’ Architecture:
  - based on pipelining / true parallel processing
  - Concurrency

PARALLELISM

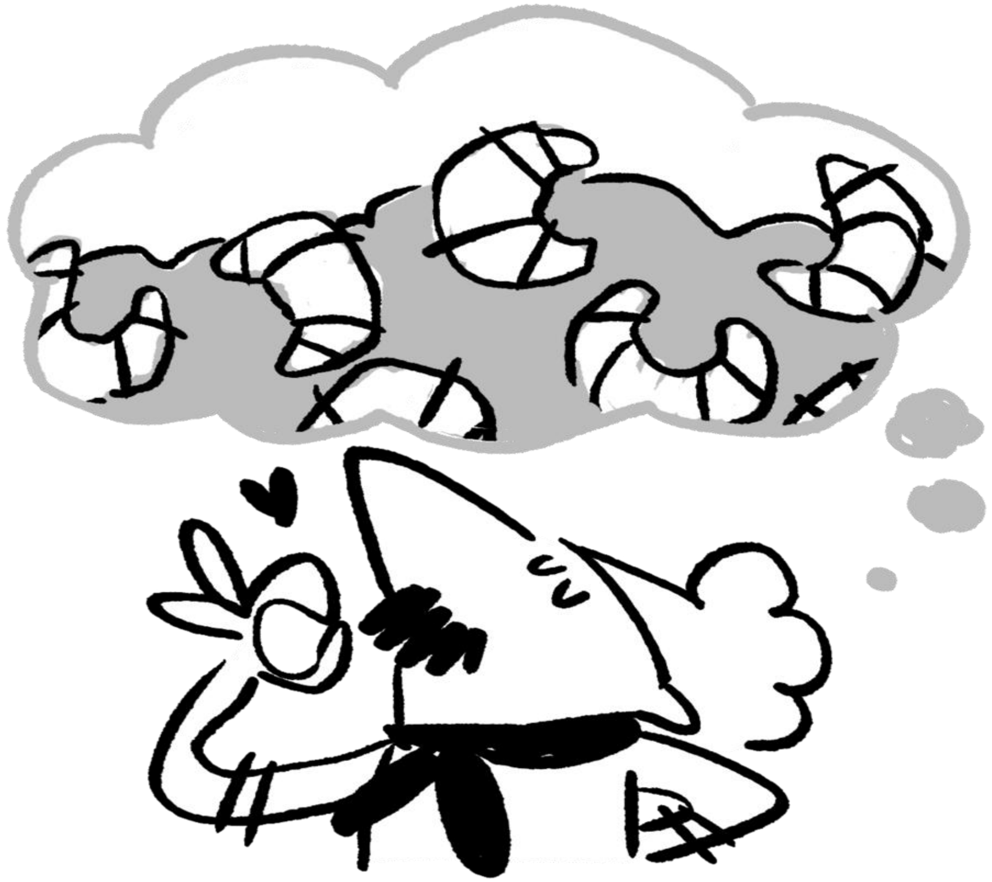
Prepared by Dr Syed Khaldoon Khurshid
Powered by Brilliant Application

- ***Taking advantage of parallelism*** is a good computer science problem-solving technique to explore, because it is both fundamentally simple and extremely powerful.
- Simply put, parallelism is just trying to solve problems faster by doing multiple things at the same time.

- You'll explore parallelism in the context of Pierre and his patisserie.

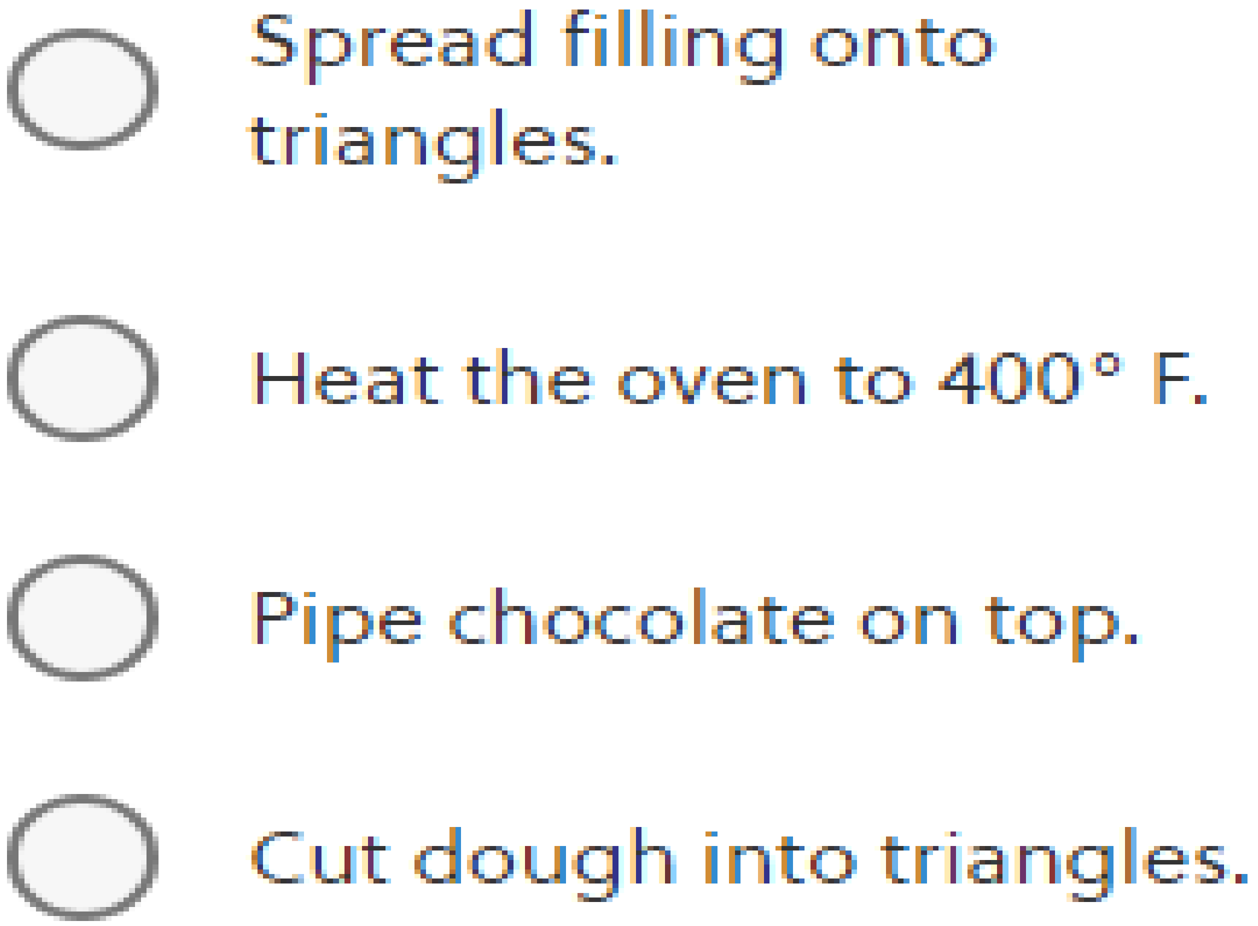
- There's **lots of parallelism in a bakery**. Multiple tasks can happen at once, and multiple people can combine forces to make single tasks faster.
- By the end of this exploration, you'll understand how the **underlying concept of parallelism applies to computer systems in the real world**. You'll learn about:
- ***Amdahl's law,*** an important piece of common-sense mathematics for computer scientists,
- ***Pipelining,*** a form of parallelism that enables advanced computer graphics, and
- ***Concurrency,*** and how it's managed to let the many apps on your phone run at the same time.

- Pierre arrives at his bakery in the morning, with the goal of preparing 200 fresh pastries as quickly as possible.



- He takes the dough he prepared the night before out of the refrigerator, and looks at his recipe:
- Cut the dough into triangles (2 minutes for all 200).
- Spread filling onto triangles (4 minutes for all 200).
- Roll filled triangles into crescents (10 minutes for all 200).
- Heat the oven to 400° F (oven takes 1 minute to turn on and then 16 minutes to heat up).
- Place pastries in the fully heated oven for 20 minutes.
- Remove the pastries from the oven and pipe chocolate on top (4 minutes for all 200).
- After piping, let the pastries cool for 10 minutes before serving.

- What should Pierre's first step be?



○ Spread filling onto triangles.

○ Heat the oven to 400° F.

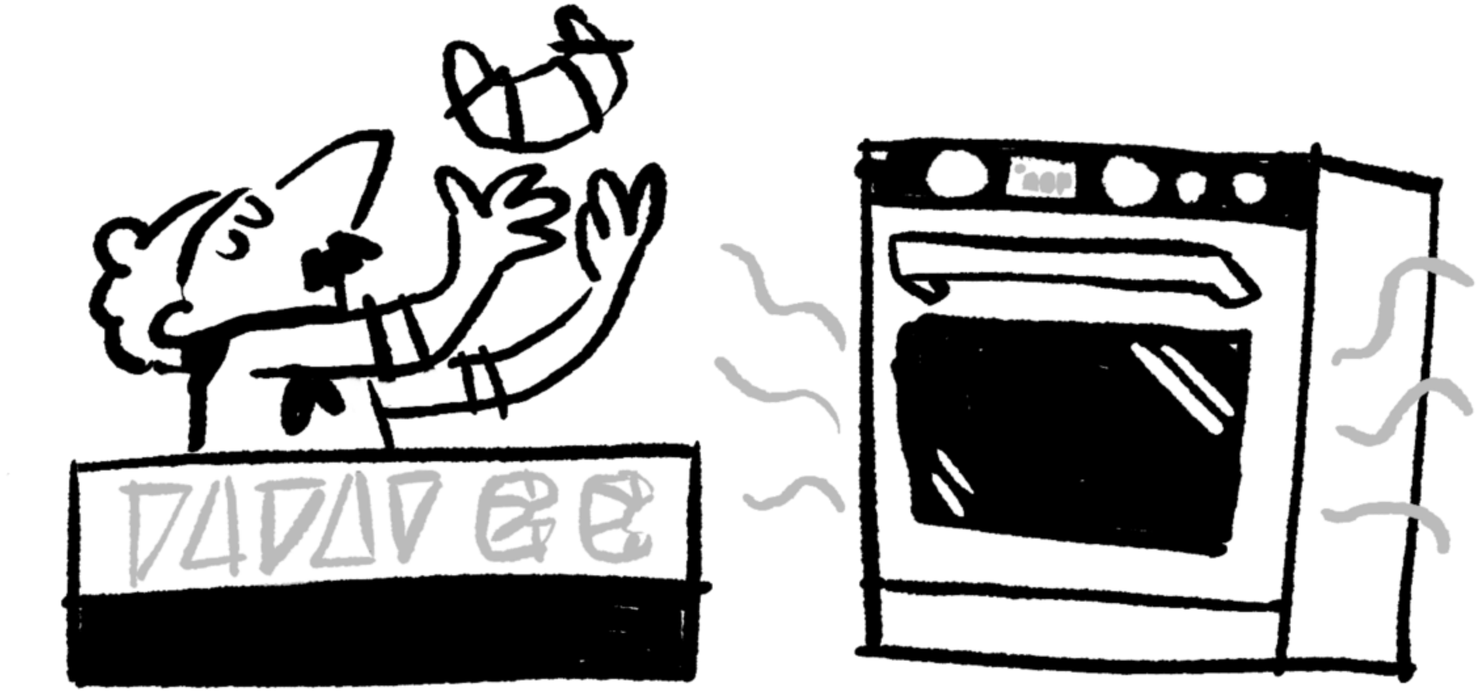
○ Pipe chocolate on top.

○ Cut dough into triangles.

Correct answer: **Heat the oven to 400° F.**

- Only two of the options make much sense as a first step: cutting the dough into triangles and heating the oven. You can't pipe chocolate or spread filling onto cut triangles when the pastry dough hasn't been cut!
- To see why he should heat the oven first, notice that Pierre will need 2 minutes to cut the dough into triangles, 4 minutes to spread filling onto the triangles, and 10 minutes to roll the filled triangles into crescents. **This is a total of 16 minutes, the same amount of time it takes for the oven to heat up.**
- Therefore, the first thing Pierre should do is turn the oven on. Then, he'll have pastries in the oven 17 minutes after he starts. Any delay in heating the oven means that it'll take longer to get the pastries finished.

- Pierre's kitchen can do two things at once: the oven can warm up while Pierre prepares his pastries.



- This phenomenon comes up frequently in our daily lives, and it also comes up in computer science—sometimes in surprising ways!
- Every modern computer is incredibly fast at performing calculations. It's *so* fast that reading a file from a hard drive takes ages by comparison: a computer can perform *tens of thousands* of calculations **in the time it takes to request information from its own hard drive and get that information back.** Depending on the computer, it might be able to perform *millions* of calculations.
- Computer systems are designed to do useful work while they're waiting for the hard drive to access a file, just as Pierre works to prep his pastries while the oven warms up.

- When Pierre brings in an assistant, he's practicing ***another form of parallelism***. One person can cut a sheet of pastry dough in 30 seconds, and **two** people can split up the work and cut **two** sheets of pastry dough in 30 seconds, getting twice as much done!

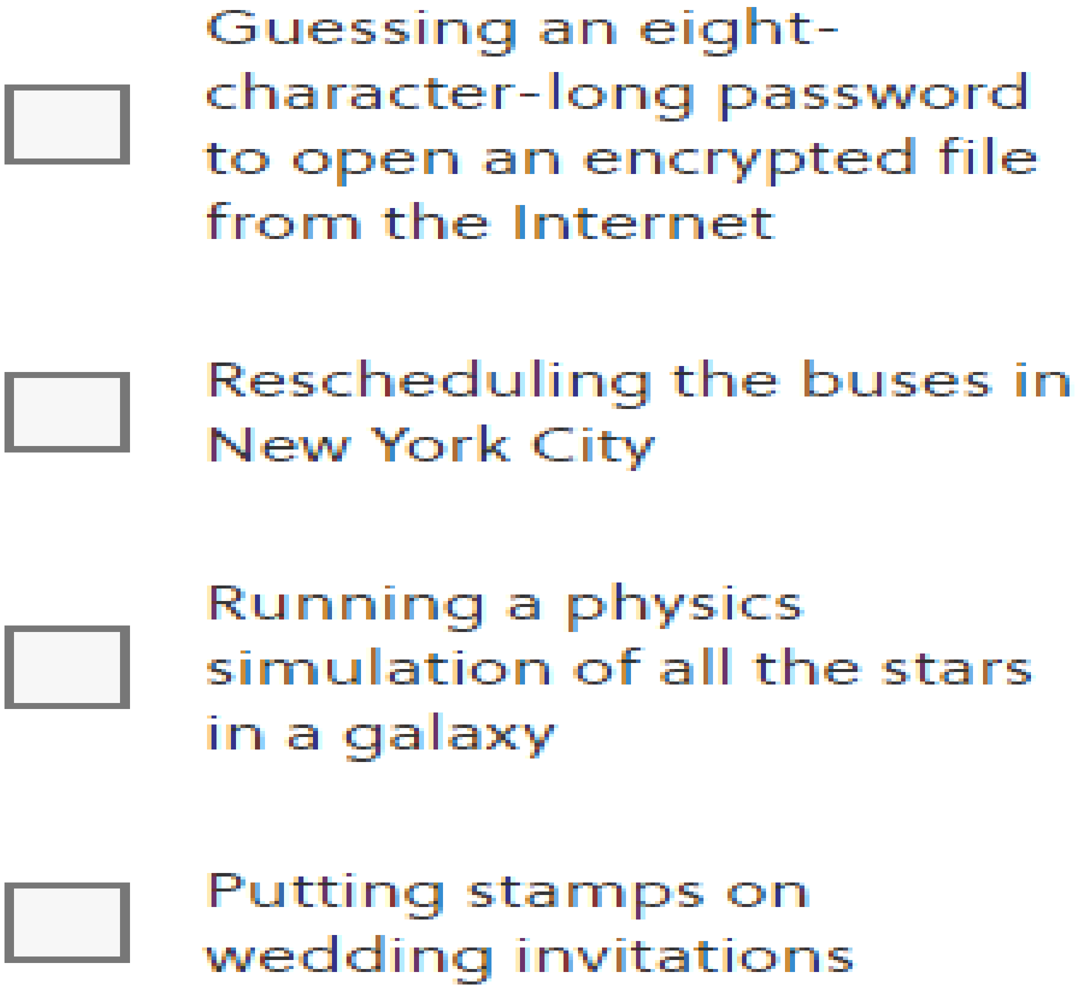
- **When multiple workers can split up a problem without adding in much extra coordination overhead, computer scientists sometimes call the problem**

***embarrassingly* parallel.**

An embarrassingly parallel task can be considered a trivial case - little or no manipulation is needed to separate the problem into a number of parallel tasks.
This is often the case where there is little or no dependency or need for communication between those parallel tasks, or for results between them.

https://en.wikipedia.org/wiki/Embarrassingly_parallel#:~:text=From%20Wikipedia%2C%20the%20free%20encyclopedia,delightfully%20parallel%20or%20pleasingly%20parallel).

- Which of these problems are embarrassingly parallel? (Select all that apply.)



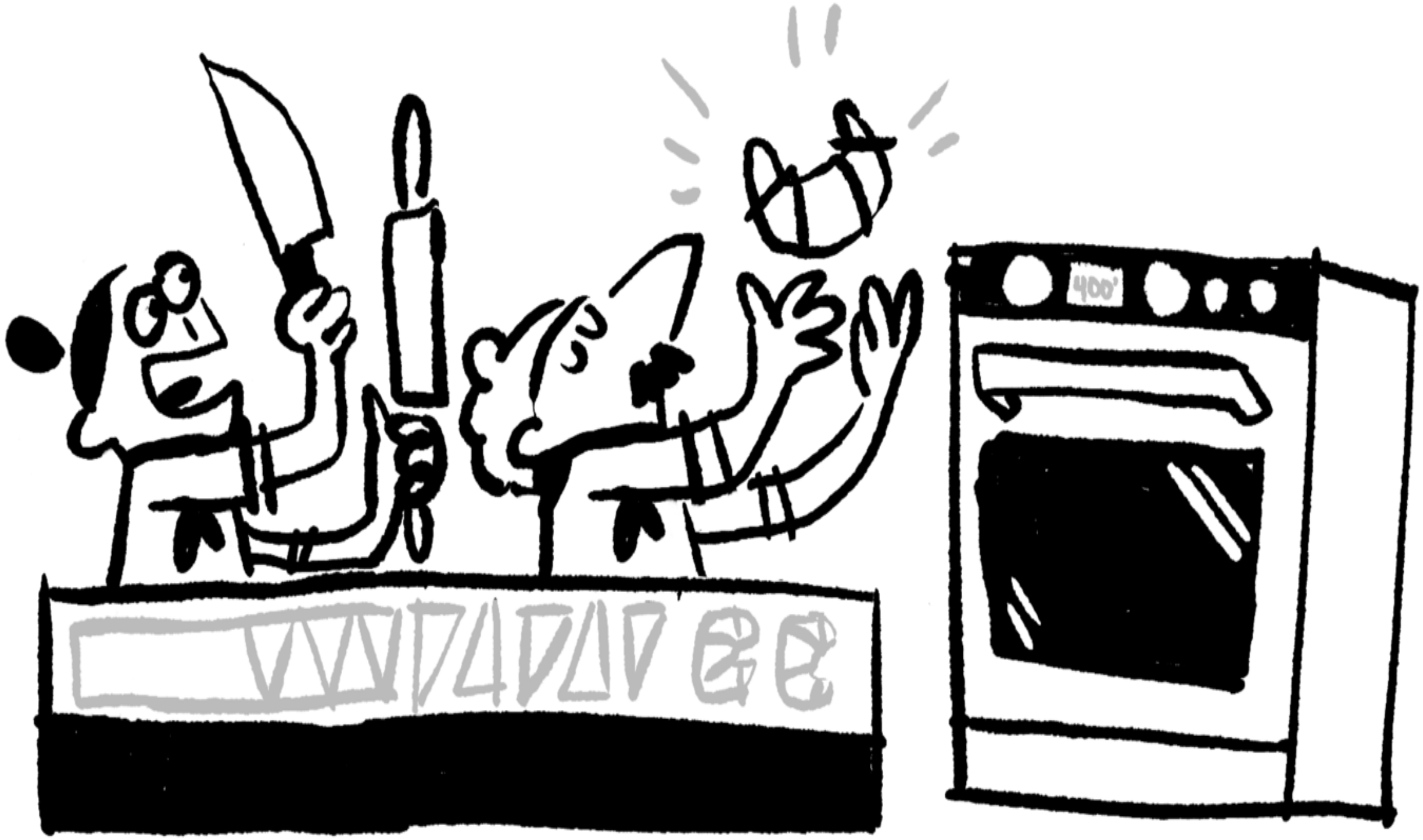
- ☐ Guessing an eight-character-long password to open an encrypted file from the Internet
- ☐ Rescheduling the buses in New York City
- ☐ Running a physics simulation of all the stars in a galaxy
- ☐ Putting stamps on wedding invitations

Correct answer: **Guessing an eight-character-long password to open an encrypted file from the Internet, and Putting stamps on wedding invitations**

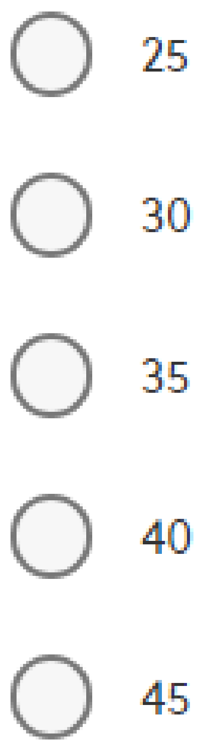
- It's easy to split up the invitations so that multiple people can share the work of adding stamps.
- If you **share the password-protected file, it's easy to divide up the list of all possible eight-character-long passwords so that two computers, a thousand computers, or a million computers can share the work.**
- If you've ever had your bank account locked because there were too many incorrect guesses, that was the bank trying to prevent multiple people (or computers) from working together to guess your password!

- Bus schedules are all dependent on one another for fast transfers, and many other entities, including schools, trains, and hospitals, make decisions based on bus schedules. Therefore, **bus scheduling is a problem with an enormous and necessary amount of coordination overhead.** It is not embarrassingly parallel at all.
- Physics simulations are interesting. **Because every star in the galaxy has a gravitational effect on every other star in the galaxy, this is not considered an embarrassingly parallel problem.** However, using multiple computers in parallel to perform physics simulations is an incredibly important part of modern cosmology. Physicists and computer scientists have put decades of work into figuring out how to **minimize coordination overhead** to make physics simulations as fast and parallel as possible.

- Once Pierre's oven is up and running, he doesn't have to worry about reheating it anymore. **His oven can fit 200 pastries at once.** Here's his recipe again:
- Cut the dough into triangles (**2 minutes** for all 200).
- Spread filling onto triangles (**4 minutes** for all 200).
- Roll filled triangles into crescents (**10 minutes** for all 200).
- Place pastries in the oven for **20 minutes**.
- Remove the pastries from the oven and pipe chocolate on top (**4 minutes** for all 200).
- After piping, let the pastries cool for **10 minutes** before serving.
- If Pierre is working alone with a preheated oven, it takes him **50 minutes to completely bake 200 pastries**.


400°

- If Pierre is working with one assistant who can help with the cutting, spreading, rolling, and piping steps at the same speed as Pierre, and the oven is preheated, **how many minutes will it take before he has ready-to-serve pastries?**



○ 25

○ 30

○ 35

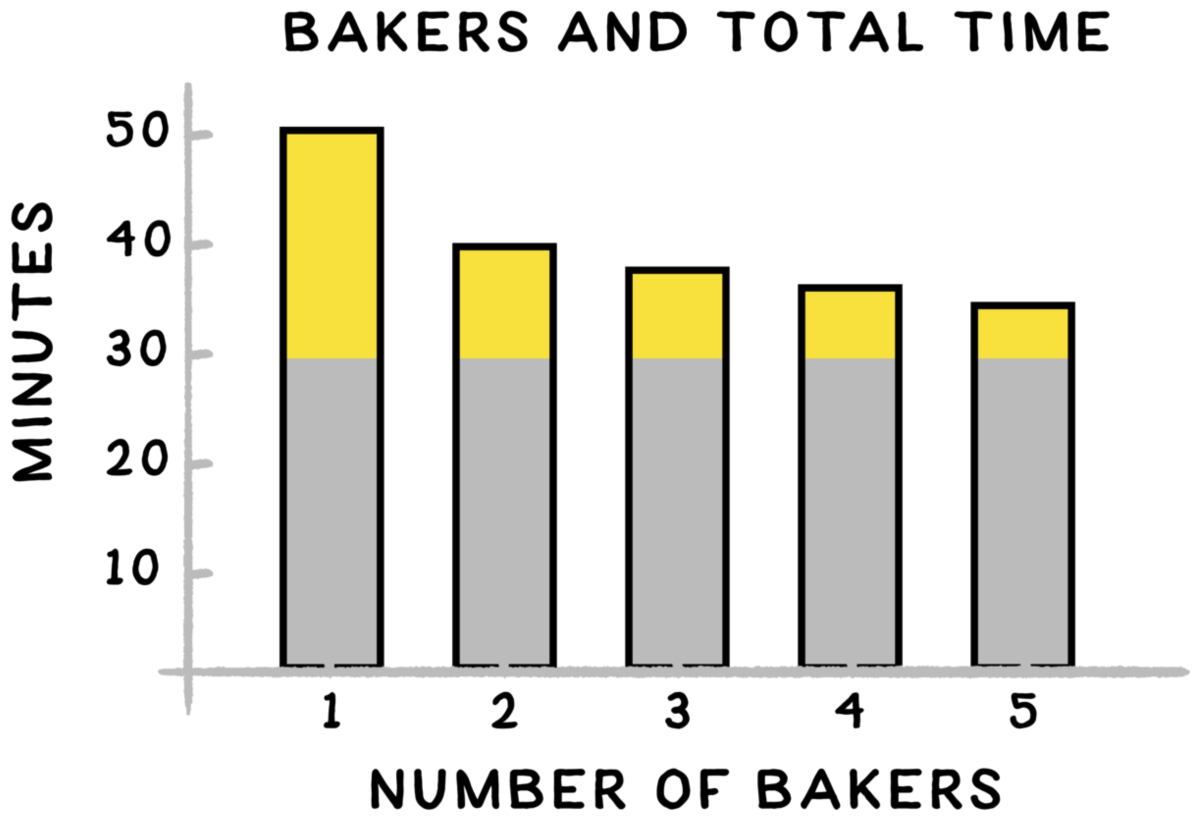
○ 40

○ 45

Correct answer: 40

- There's no way to make the 20 minutes of cooking or 10 minutes of cooling go faster. However, the remaining 20 minutes of work—cutting, spreading, rolling, and piping—can be done in parallel by two bakers, so a single assistant would help Pierre get that work done in **10 minutes**.

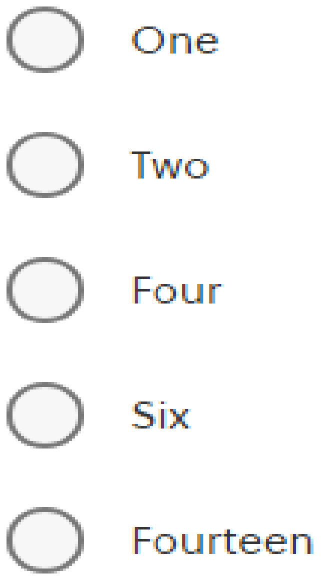
- Most of the time Pierre needs to make his pastries is spent in the baking and cooling phases. The chart below shows how more assistants don't end up helping him make pastries much faster. **Only 20 minutes of work can be split up—that's the yellow part of the bars.** The other 30 minutes of work can't be done in parallel—that's the gray part.



- Everyone from computer scientists to bakers and architects can fall into the trap of thinking that they can speed something up by throwing more workers into the mix. **But that isn't true. When you think about parallel and non-parallel tasks, it should be obvious that adding more workers will only make you faster at the parts of a task that can be done in parallel.**
- This common-sense observation is important enough in computer science that it has a name, **Amdahl's law**, after Gene Amdahl who wrote a paper about it in 1967.

- Pierre makes pastries **10 minutes** faster by hiring one assistant. You can see this on the graph above, where **50 minutes** required for Pierre alone is reduced to **40 minutes** when Pierre is working with one assistant.
- In a different bakery, Hans and his one assistant work together, so there are two bakers and baking takes **40 minutes**.

- Hans is thinking about buying a fan to decrease the non-parallelizable cooling time from **10 minutes to 5 minutes**. If Pierre and his one assistant wanted to achieve the same 5-minute gain by hiring more people, **how many additional assistants would they need to hire?**



○ One

○ Two

○ Four

○ Six

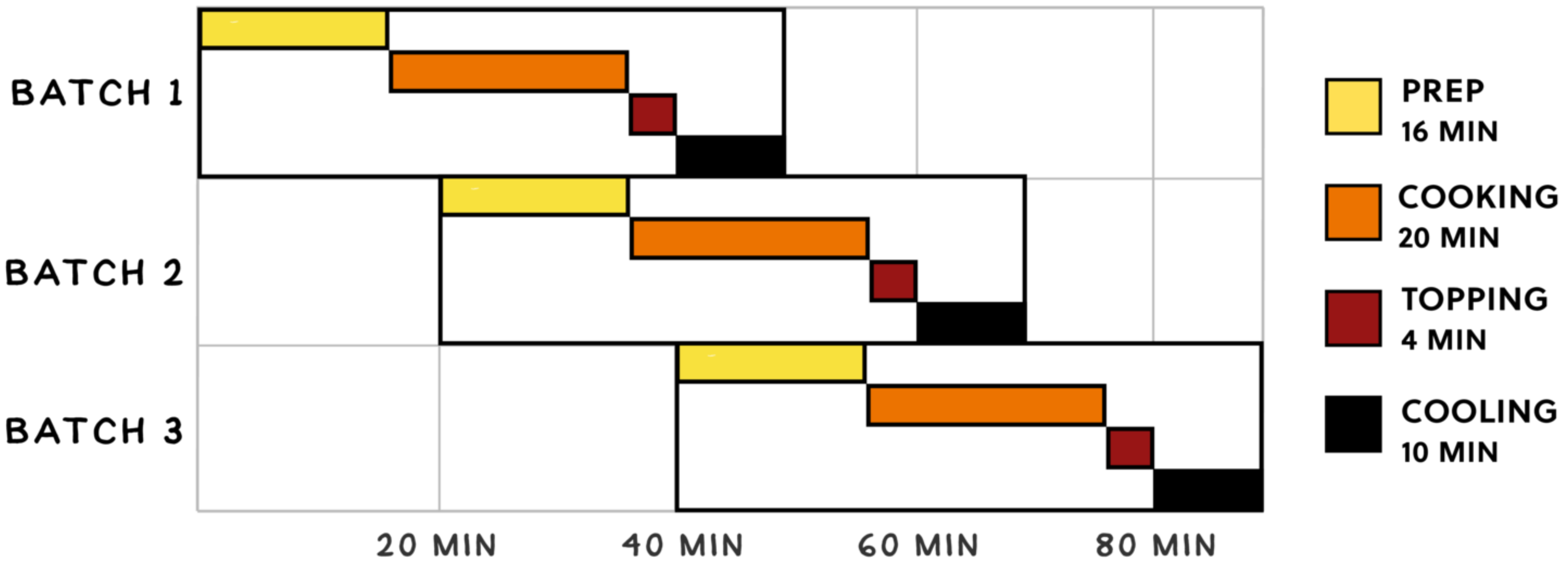
○ Fourteen

Correct answer: **Two**

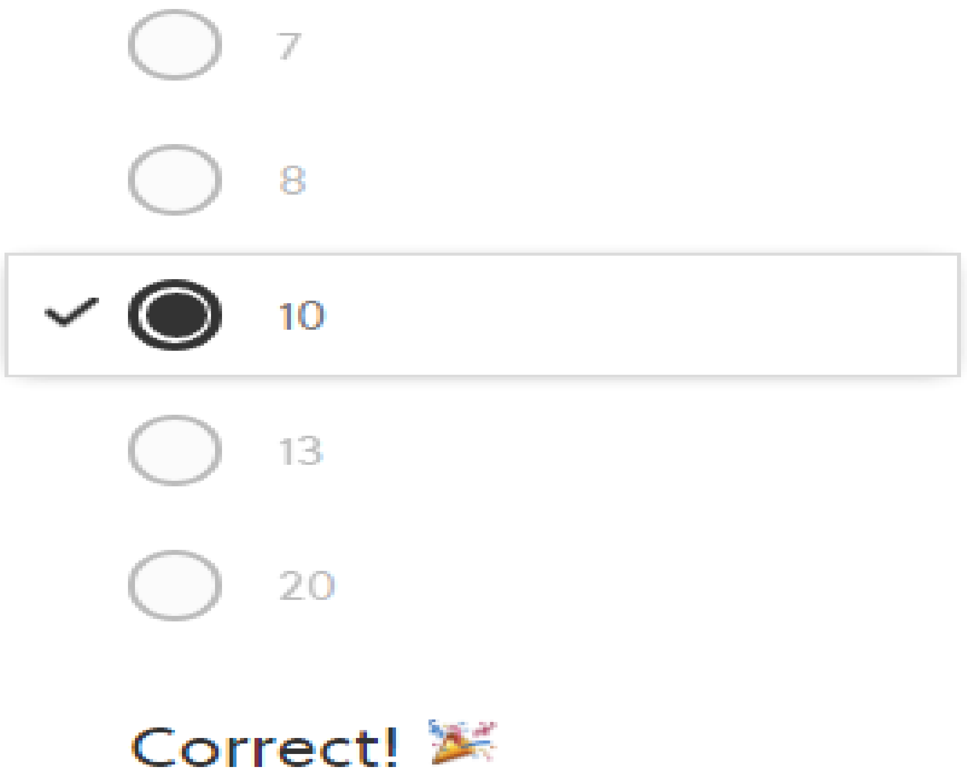
- Hiring two new assistants doubles the total number of bakers and doubles their working speed. Doubling their working speed only saves them five minutes.
- Even though the fan only reduces the non-parallel food preparation time from 30 minutes to 25 minutes, a roughly 17% reduction, it gives the same overall speedup.

- You saw in the previous example that, even with a preheated oven, **it took Pierre 50 minutes to make pastries, and 30 minutes of that time can't be parallelized.** Does this mean that Pierre's patisserie is limited to producing 200 pastries every 50 minutes (4 pastries per minute) if Pierre works alone, or at most 200 pastries every 30 minutes (~6.7 pastries per minute) no matter how many assistants he hires.
- **No! If Pierre is concerned with his overall production rate, he can do better than 6.7 pastries per minute, and he can do it without any assistant.**

- The solution may be familiar to anyone who has ever made lots of cookies at once. **Pierre needs to prep his current batch of 200 pastries while the previous batch of 200 is cooking.** If he times it so that his prep is done when the **previous batch** finishes cooking, he can put the **current batch** in the oven, pipe chocolate on the **previous batch** and leave it to cool, and then immediately start prepping the **next batch** of pastries.


BATCH 1
BATCH 2
BATCH 3
20 MIN
40 MIN
60 MIN
80 MIN
PREP 16 MIN
COOKING 20 MIN
TOPPING 4 MIN
COOLING 10 MIN

- **This is another form of parallelism, called *pipelining*.** By working on different batches of pastries in parallel, the plan above ensures that both the oven and Pierre are always working: either on the previous, current, or next batch of pastries.
- **It takes an hour for Pierre's operation to get up and running at full speed, but after that, the schedule above shows that Pierre is able to begin a new batch of 200 pastries every 20 minutes. What is his average rate of pastry production per minute?**



7

8

[✓ 10]

13

20

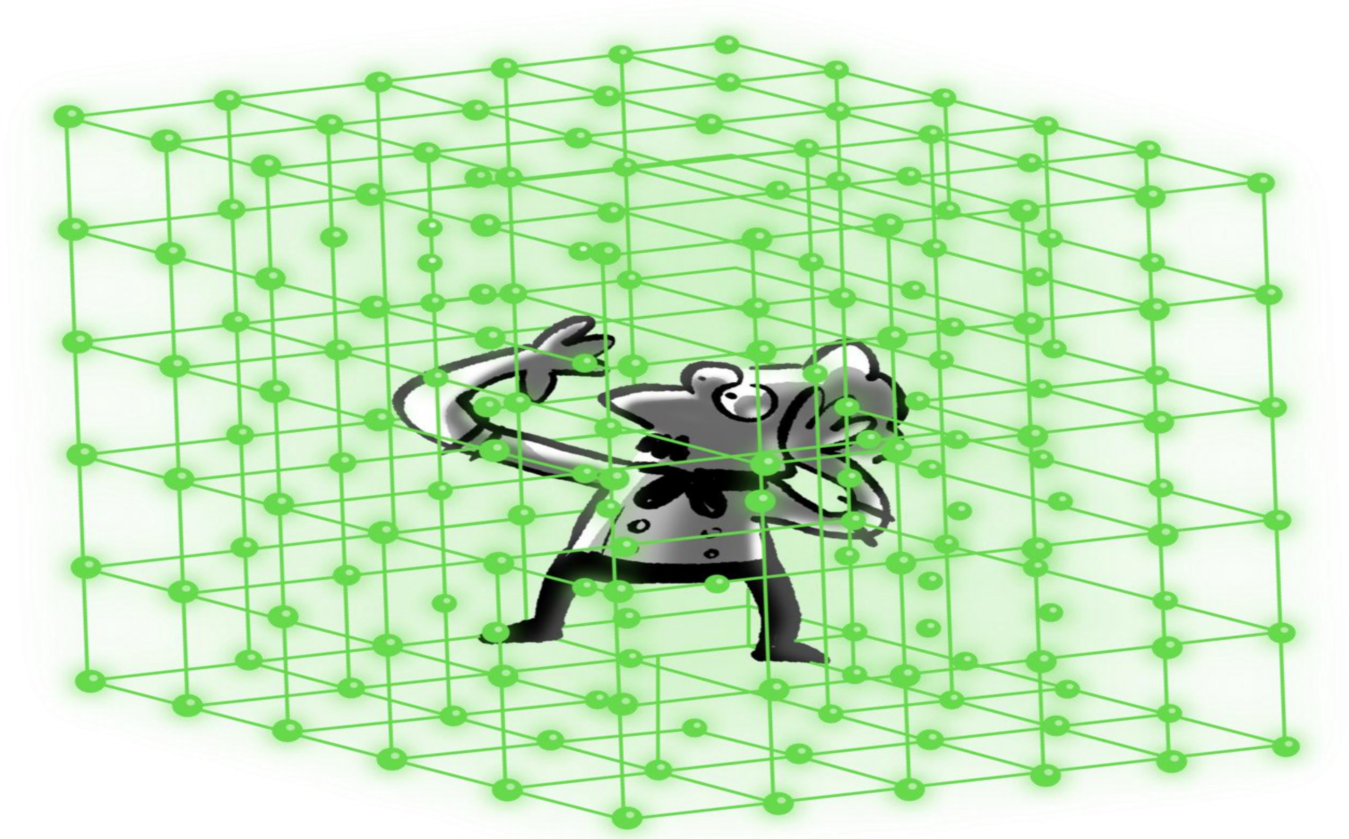
Correct! 🎉

Once Pierre has his operation running, it's clear to see on the chart he is *starting* a batch of pastries every 20 minutes: on the 20 minute mark, the 60 minute mark, the 80 minute mark, and so on.

The schedule doesn't have any bottlenecks, so **Pierre will also *complete* 200 pastries every 20 minutes.**

Therefore, Pierre will be producing 10 pastries per minute with this approach.

- Pipelining is a critical form of parallelism in the design of computer systems. **3D computer graphics are a place where pipelining has been important for a long time.**



- A computer has to do a **predictable amount of work to display a three-dimensional scene in a computer game.** The computer has to determine which parts of which objects are visible, figure out how those objects are lit, and translate the visible objects into individual pixels to be displayed. It's possible for the computer to anticipate how long each step will take ahead of time.
- This predictable flow makes computer graphics good for pipelining: **while your computer is figuring out how to display pixels for the current scene, it is also figuring out how to light a scene it will show in the next fraction of a second, and is also figuring out which objects are visible in a scene that is a few fractions of a second further away from being shown**.

- Pipelining is so important that a computer's hardware will make guesses about what it will be asked to do next in order to keep all the parts of a pipeline working. This is called ***speculative execution***.
- Imagine Pierre having a spare moment in his kitchen and prepping bread dough or pastry dough before knowing whether he will get an order for bread or pastries. It won't slow him down if he guesses wrong (he might just waste some dough), **but guessing right will help him finish the next order faster.**

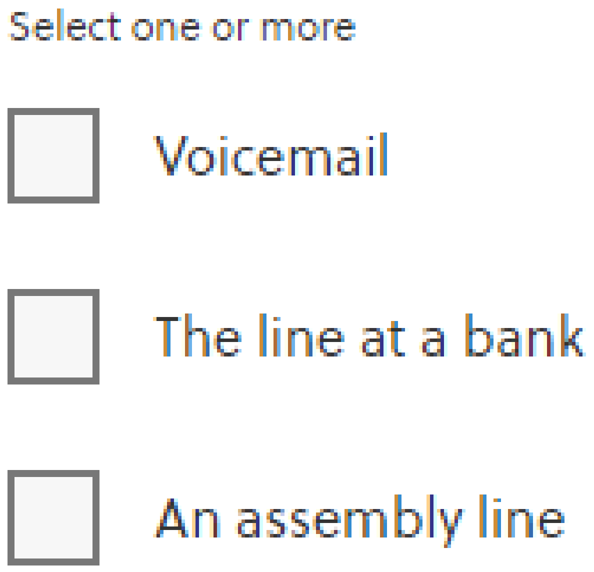
- If you leave the kitchen and walk into Pierre's **storefront**, you'll see a concept closely related to parallelism:

- From a certain perspective, there is parallelism at work in the busy crowd. **All the individuals who want to buy Pierre's pastries are independently working on accomplishing their goals for the day, in parallel.**
- However, this situation introduces a new wrinkle: there is **conflict**, because **each person in the bakery needs brief but exclusive access to the cashier in order to meet their goals**.

- ***Concurrency*** is the name for situations where multiple agents each want a turn at having exclusive access to the same resource (like the cashier). **It sometimes shows up in similar places as parallelism, but concurrency occurs when there's no way to plan in advance for the order in which things will happen.**
- Pierre can plan a schedule for himself and his assistants. However, **he and his customers don't have any way of anticipating who else will show up wanting pastries**.

- In your computer or phone right now, multiple independent programs are trying to **access the processor** to do their computations, trying to access the networking hardware that communicates with the Internet, and trying to display content on the screen. **It is the task of your computer's operating system to help these multiple programs manage concurrent access to the resources they all need.**

- Which of these are systems that help manage concurrency in the real world? (Select all that apply.)



Select one or more

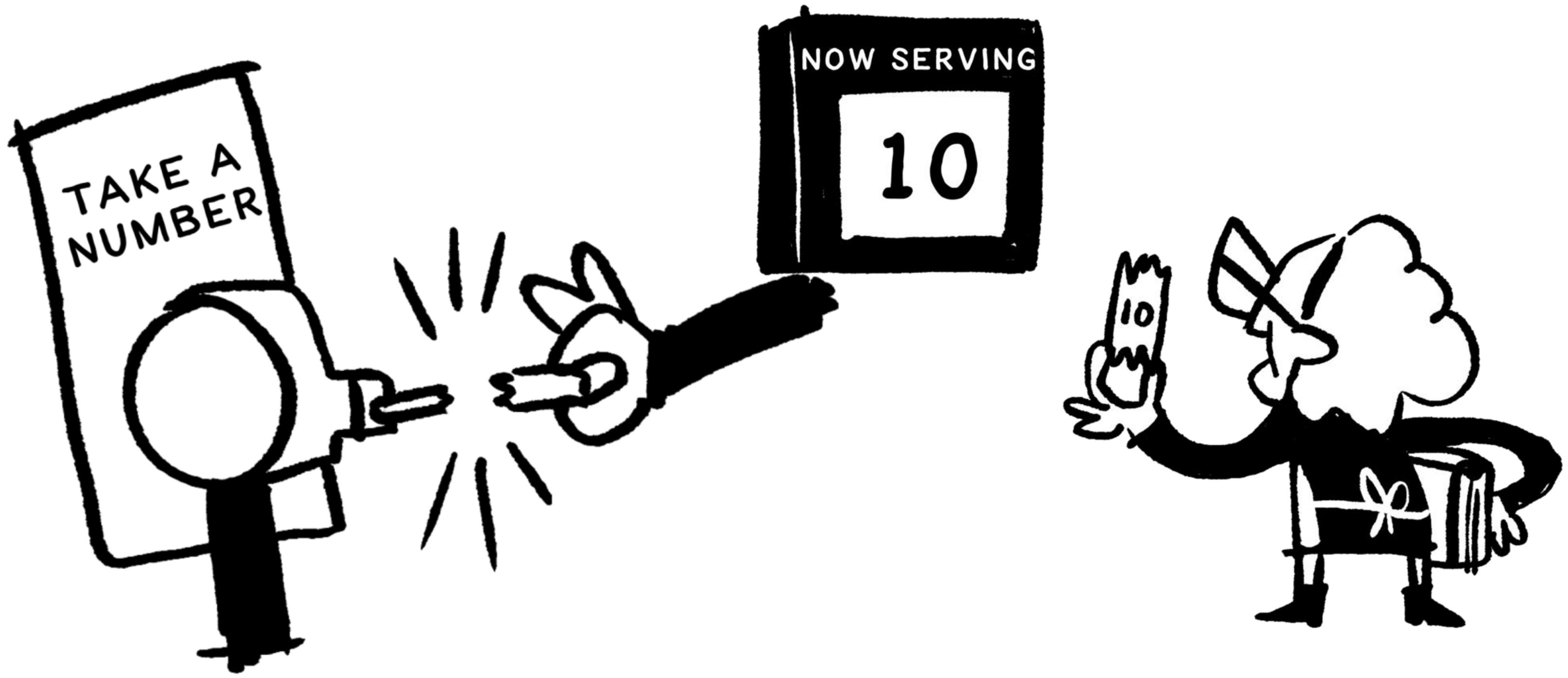
- ☐ Voicemail
- ☐ The line at a bank
- ☐ An assembly line

Correct answer: **Voicemail and The line at a bank**

- A traffic light and the line at a bank are both straightforward tools for managing concurrency in the real world: when many cars or customers want to cross an intersection or talk to a bank teller, **the system determines who gets to go and who has to wait.**
- **Voicemail is less obviously a tool for managing concurrency.** If the person you're trying to contact is already on the phone, you will be sent to voicemail so that they can respond later. **If you have ever been put on hold until a customer service representative is able to take your call, you've encountered a different tool for dealing with the same concurrency problem**.

- **Assembly lines are excellent real-world examples of pipeline parallelism. Multiple parts of an object are built at once, in a well-defined order, and the assembly line keeps everyone busy by working on different parts of different tasks at the same time.** That makes an assembly line a good example of parallelism, but not of concurrency.

- One way of dealing with concurrency that is commonly used in bakeries and food joints is **to have a machine that distributes numbers.**



- The cashier can then control which customer is **currently able to access their limited resource.**
- **This is fairly straightforward and, in fact, computer systems use the same intuition for managing concurrency between different programs that need to coordinate access to some resource.**
- **One method** for doing so, popularized by Leslie Lamport, is even called the **bakery algorithm!**

Conclusion

- The most common kind of parallelism is splitting up a problem and assigning it to different workers.
- If you can't use parallelism on every part of a problem, adding more and more workers may not help you very much.
- When you need to work on different problems at the same time, it can be useful to use pipeline parallelism.
- When different tasks need exclusive access to a single shared resource in an unpredictable order, you are dealing with concurrency.
- Concurrency and parallelism show up in many of the same places, but they are different ideas.

Any
Questions
?