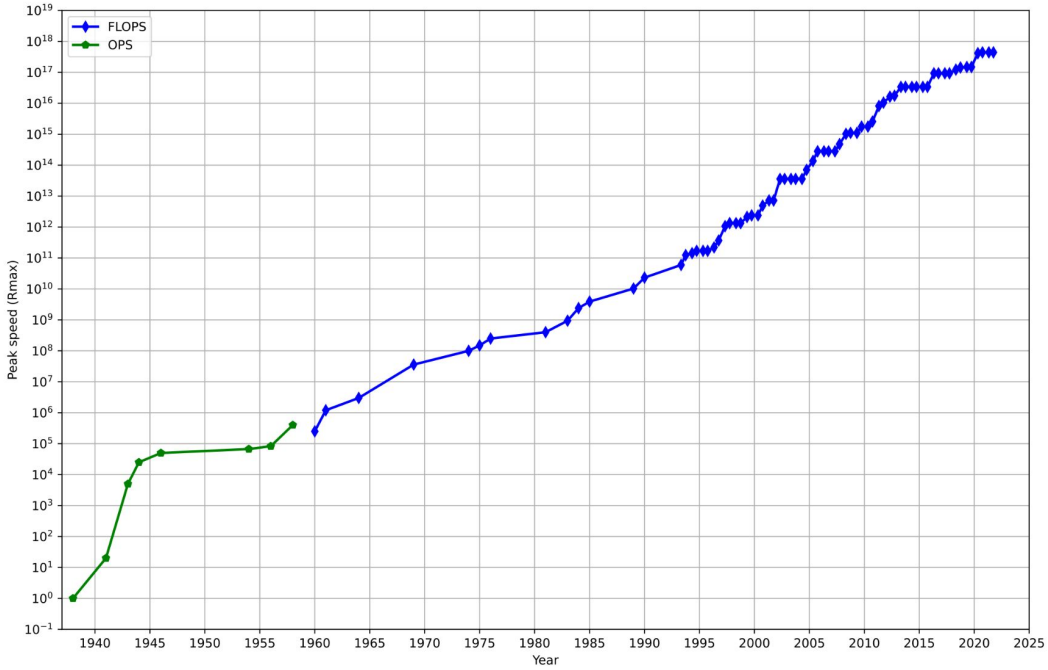
Computers are fast. Why?

https://tinyurl.com/CPUsFast

Computers have grown exponentially fast. How?



Source: Top500

Presentation outline

- What a computer does, actually.
- Boring, widely accessible things that make computers faster
- Obscure, nerdy things that make computers* faster

Presentation outline

- What a computer does, actually.
- Boring, widely accessible things that make computers faster
- Obscure, nerdy things that make computers* faster

Computers can’t read code.

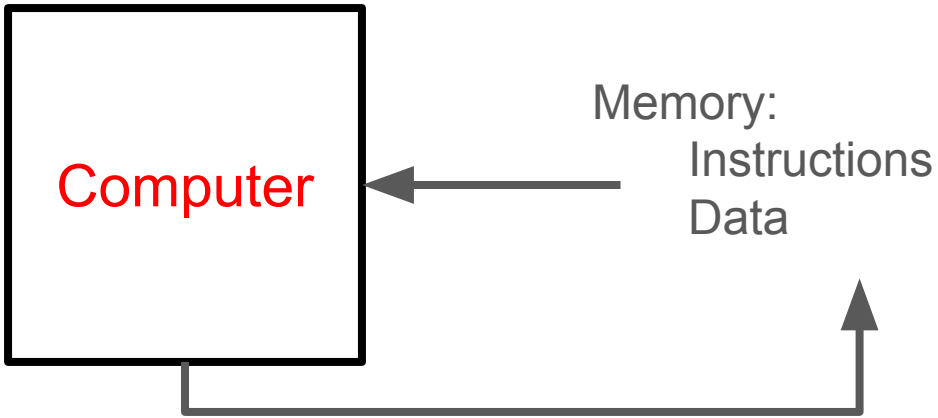
```
import serial
arduino = serial.Serial(port='COM6',   baudrate=200
x = []
y = []
cont = 0
while cont < 12000:
    content = arduino.readline().rstrip()
    if(b", " in content and len(content) < 1000):
        data = str(content.decode()).split(", ")
        x.append(int(data[0]))
        y.append(float(data[1])/(20*int(data[0])))
        print(f"{len(x)}: ({data[0]}, {data[1]})")
        cont += 1
```

Compile

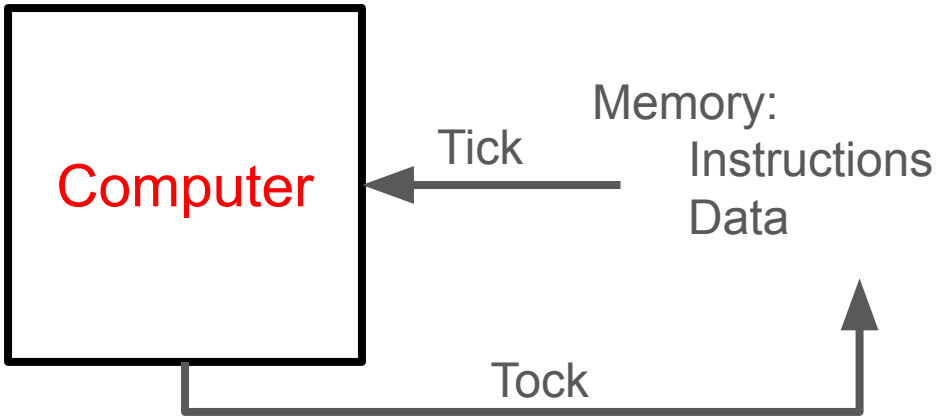
```
_start:
    MOV R0, #42
    ADD R1, R0, #100
    SUB R2, R1, #0xBAD
    MOV R3, R2
    LDR R4, =foo
    STR R3, [R4]
```

```
10101110001010101101011
10101010100111010101001
01001010101110101011010
10101001010111010100101
01010011010101101101010
01001010101010111010010
```

Computers execute binary instructions from memory, one thing at a time



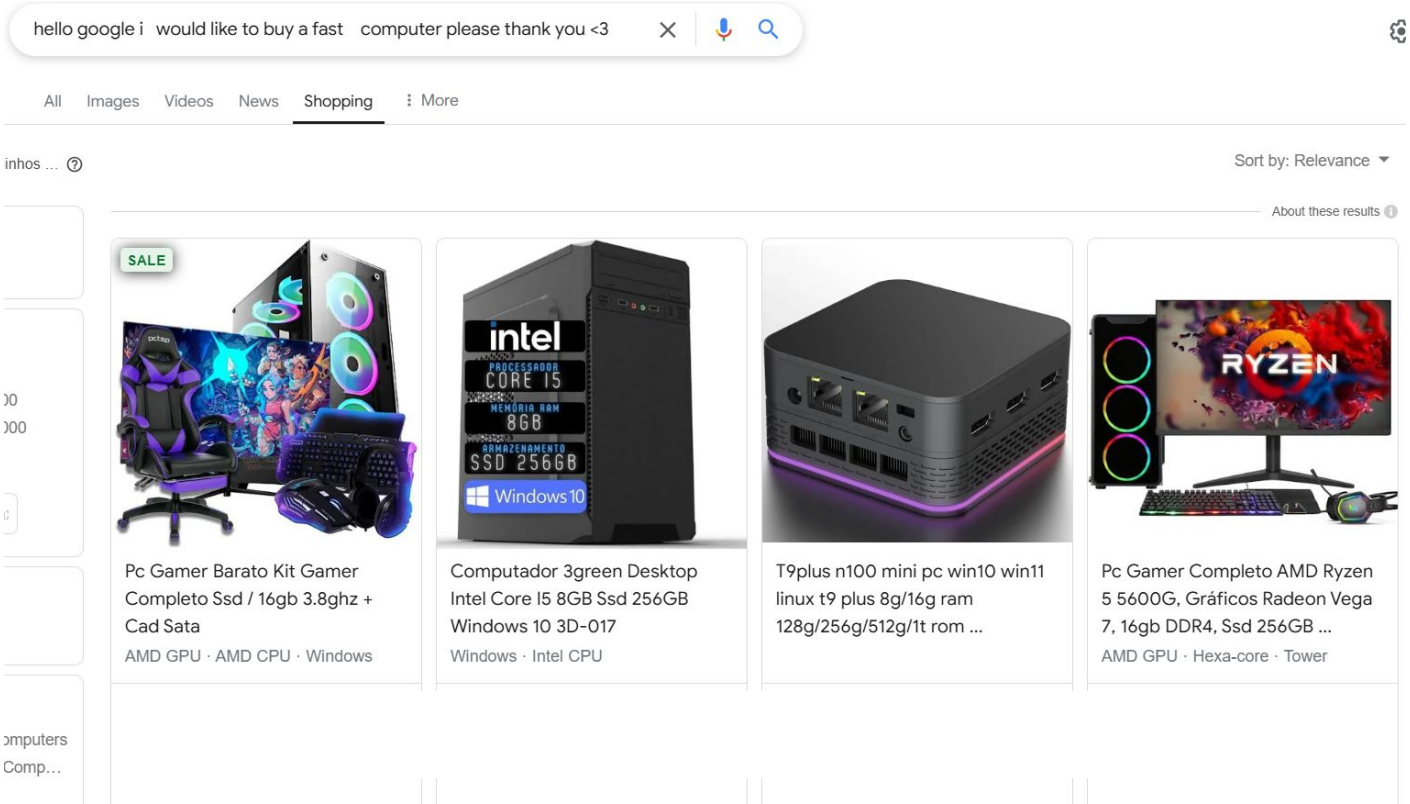
Computers work based on a clock



Presentation outline

- What a computer does, actually.
- Boring, widely accessible things that make computers faster
- Obscure, nerdy things that make computers* faster

Bright colours don’t actually make computers faster



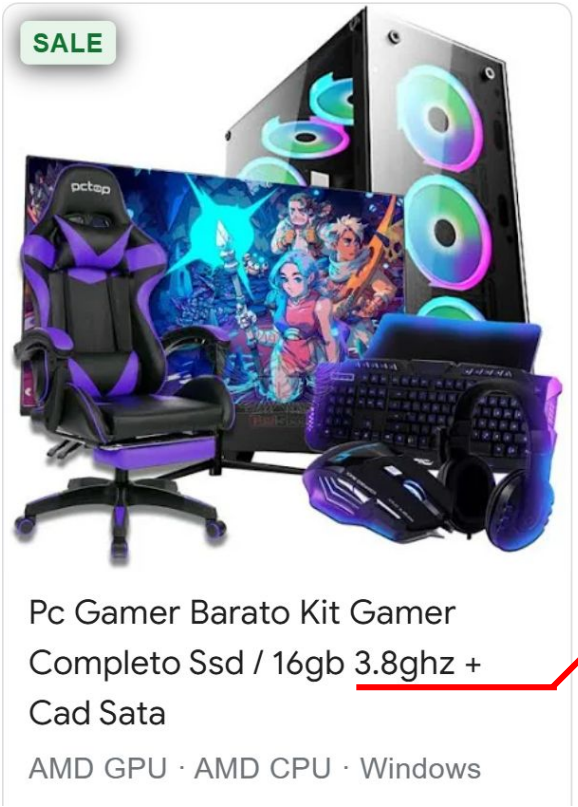
1st trick: have specialized components inside the computer

Pc Gamer Completo AMD Ryzen 5 5600G, Gráficos Radeon Vega 7, 16gb DDR4, Ssd 256GB ...

AMD GPU · Hexa-core · Tower

Graphics card (GPU) is a special type of component that accomplishes certain tasks much quicker than the main processor (CPU)

2nd trick: make the clock tick faster

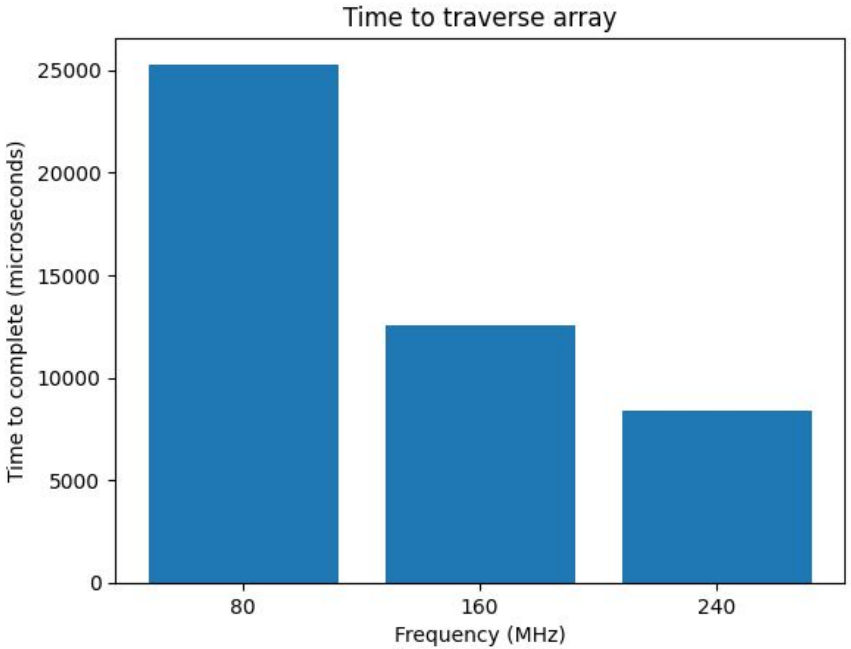


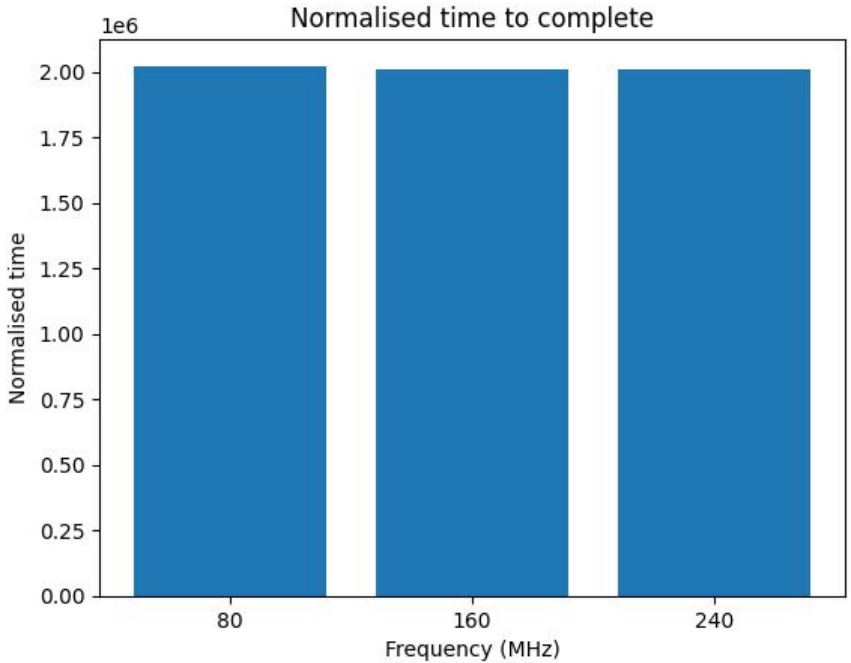
This means the CPU has a clock that ticks 3.8 billion times a second, which is a million times faster than some 1980s cpus

The code for testing the frequency of ESP32

```
for(byte s = 0; s < 3; ++s){
  setCpuFrequencyMhz(speeds[s]);
  long long ini = micros();
  int chcksum = 1;
  for(byte i = 0; i < 20; ++i){
    for(int j = 0; j < MAXN; ++j){
      chcksum ^= arr[j];
    }
    //Serial.println("summing...");
  }
```

The results are linear, as expected





3rd trick: have several computers inside one

Pc Gamer Completo AMD Ryzen 5 5600G, Gráficos Radeon Vega 7, 16gb DDR4, Ssd 256GB ...

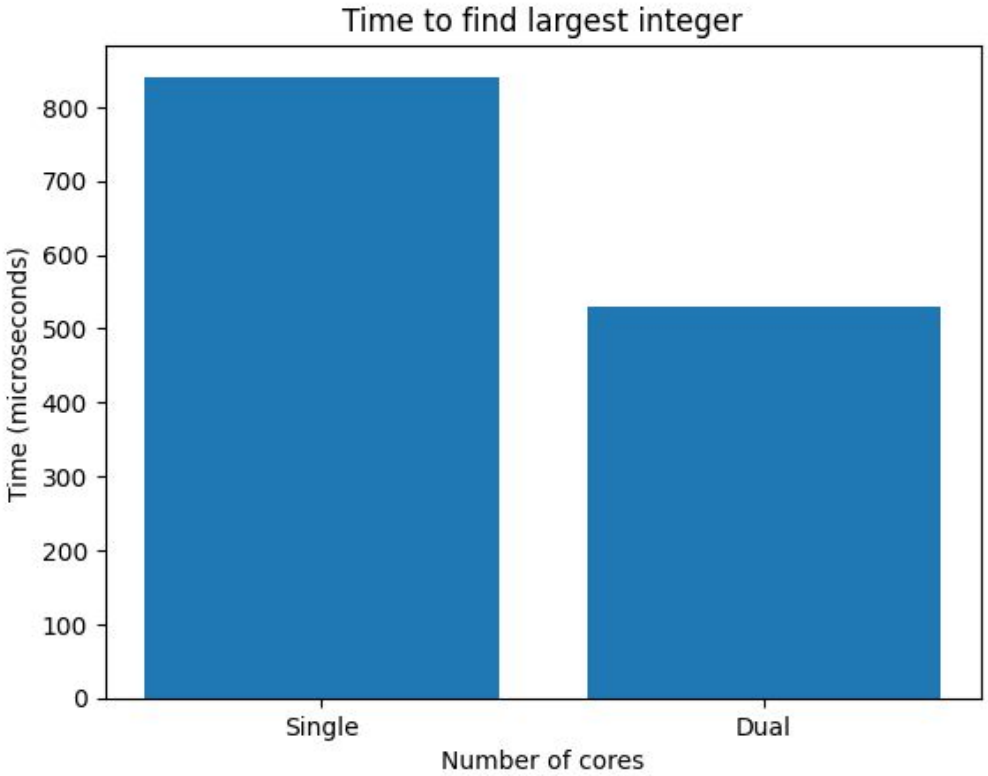
AMD GPU · Hexa-core · Tower

Actually, this CPU is composed of several “CPUs” (cores) eating instructions in parallel

ESP32 (!) code for testing multiple cores (choose max)

```
void code0(void* parameter) {
  for (int i = 0; i < MAXN / 2; ++i)
    if (arr[i] > arr[max0])
      max0 = i;
  xQueueSend(q, &max0, 0);
  while(1){}
}
void code1(void* parameter) {
  for (int i = MAXN / 2; i < MAXN; ++i)
    if (arr[i] > arr[max1])
      max1 = i;
  int buff;
  xQueueSend(q, &buff, portMAX_DELAY);
  vTaskDelete(NULL);
}
```

Results are around linear, as expected

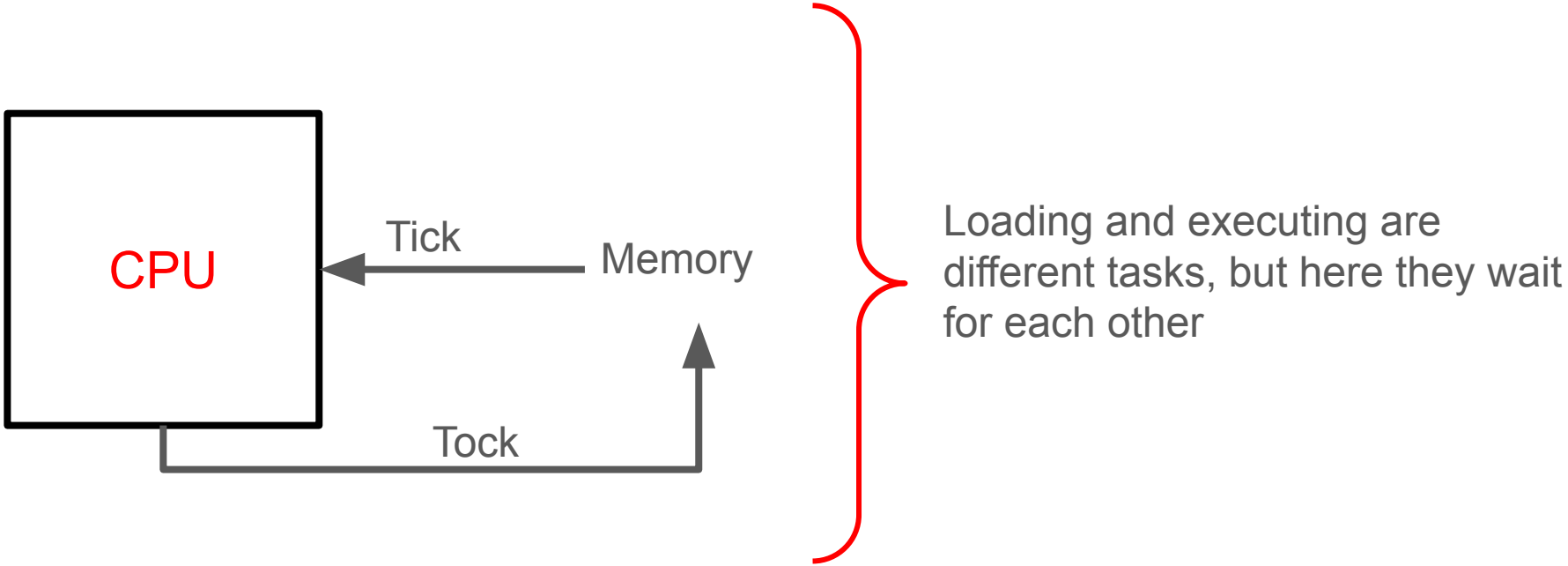


NB! Not all tasks can be parallelized

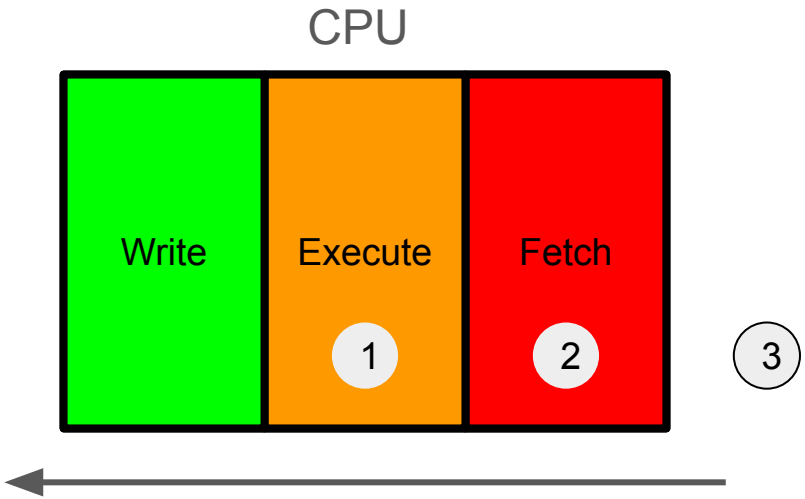
Presentation outline

- What a computer does, actually.
- Boring, widely accessible things that make computers faster
- Obscure, nerdy things that make computers* faster
  - *From now on, we will focus only on the CPU

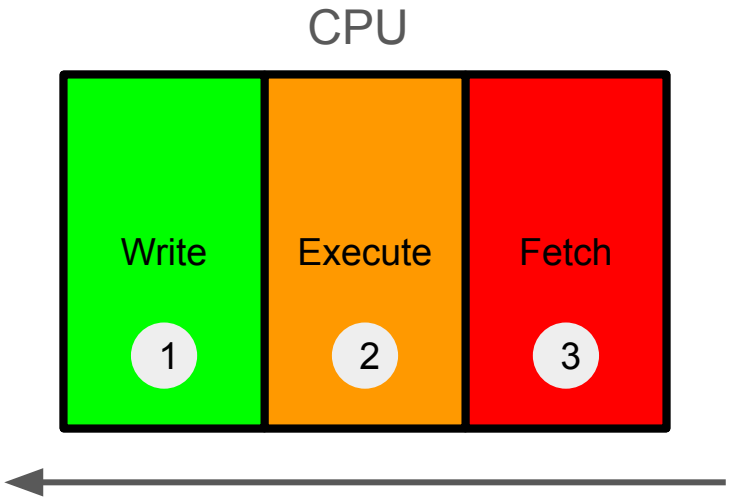
4th trick: pipelines let you fully use your CPU



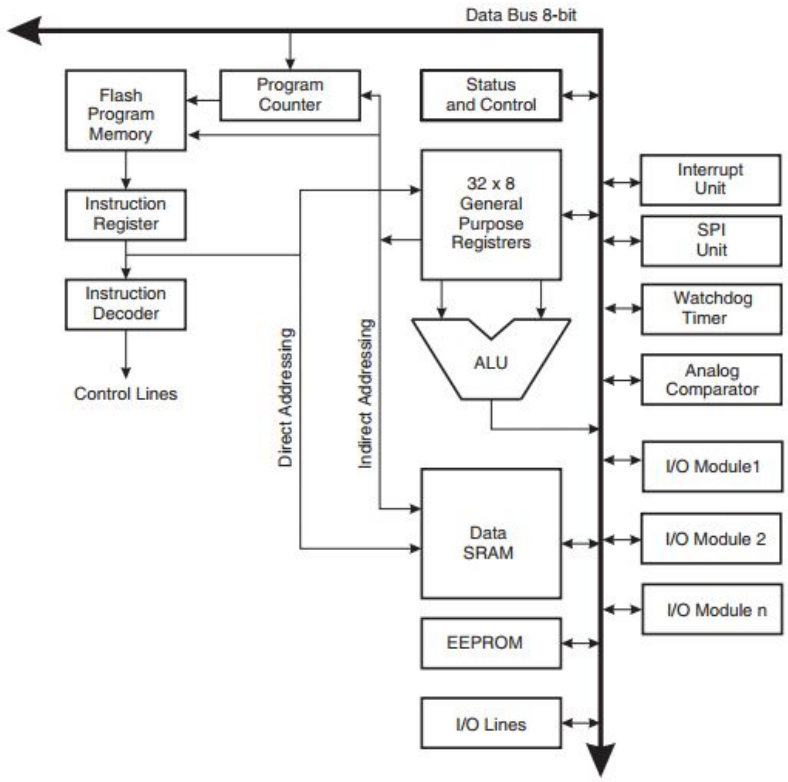
4^{th} trick: pipelines separate stages, which work together



4^{th} trick: pipelines separate stages, which work together

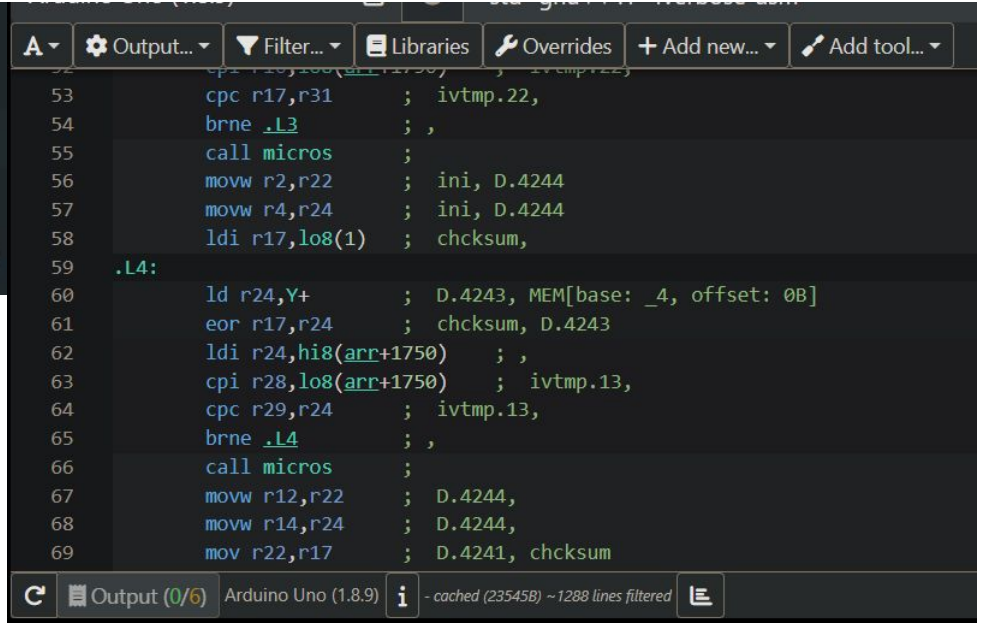


Arduino UNO has a two-stage pipeline

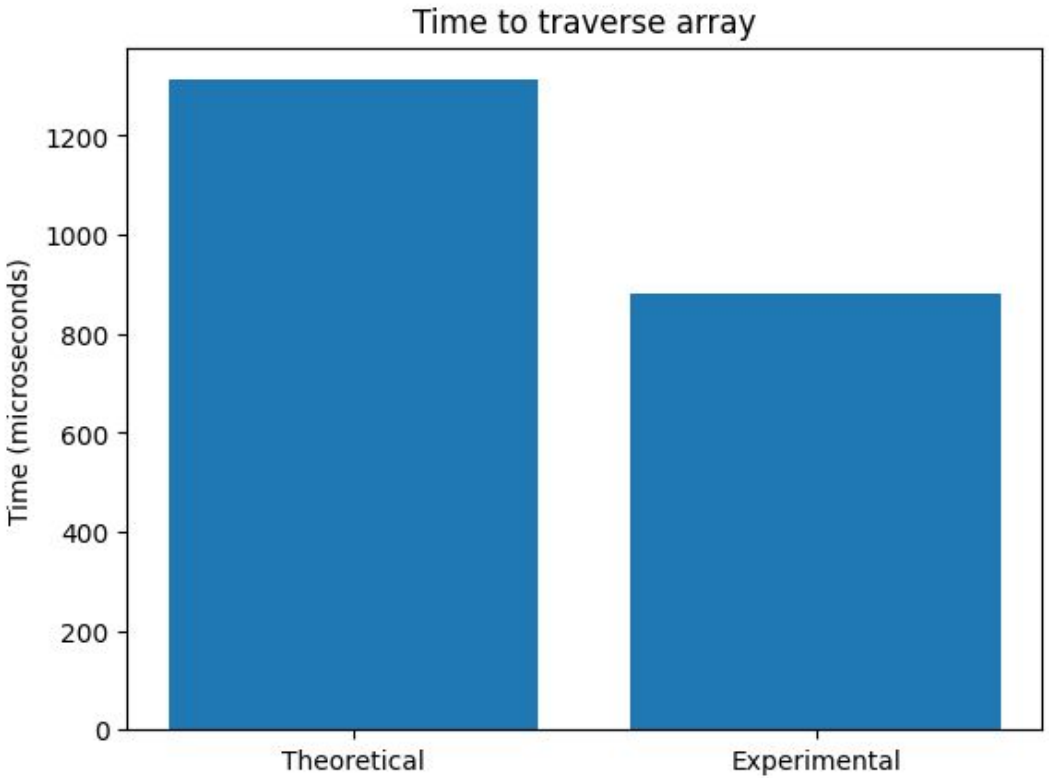


Code to test Arduino fetch/execute pipeline

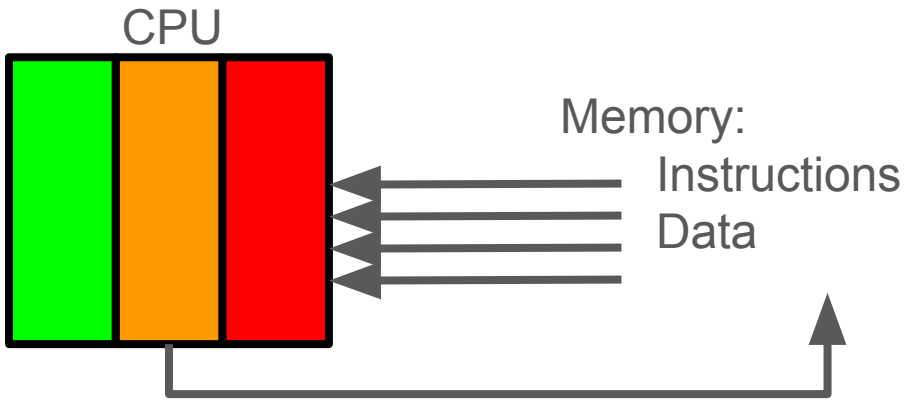
```
for(int i = 0; i < MAXN; ++i)
    chcksum ^= arr[i];
long long end = micros();
Serial.println((int)chcksum);
Serial.println((double)(end - ini));
```



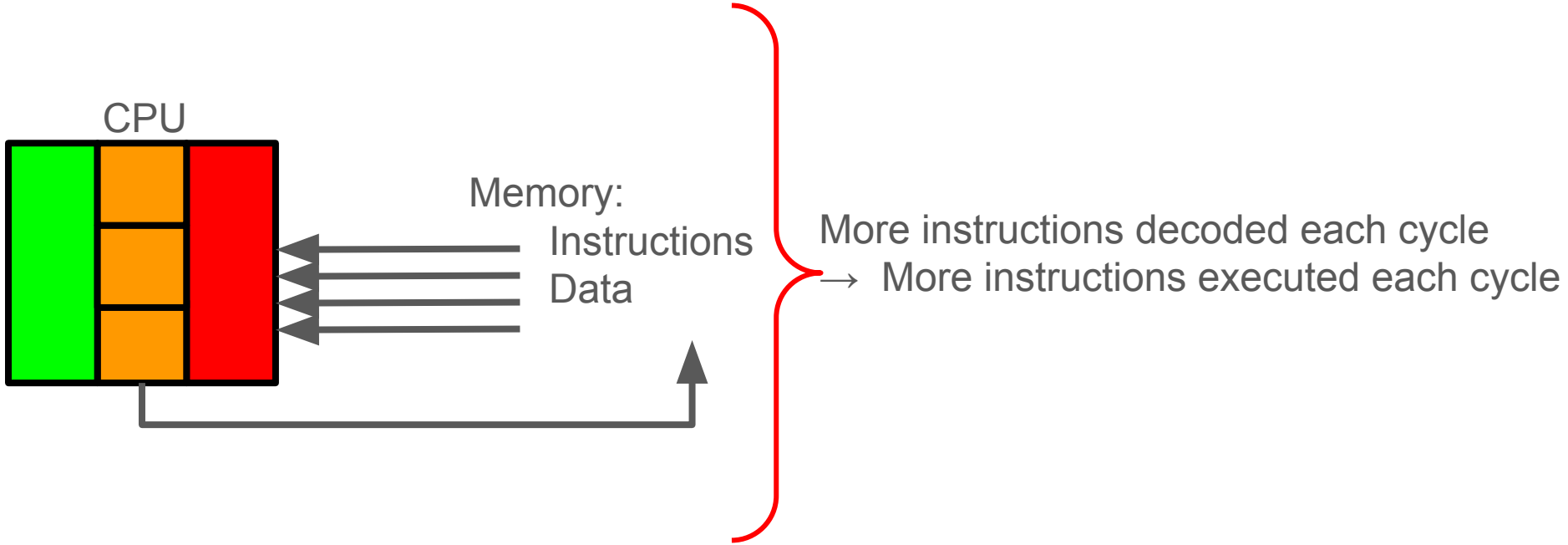
Result of Arduino code shows the pipeline



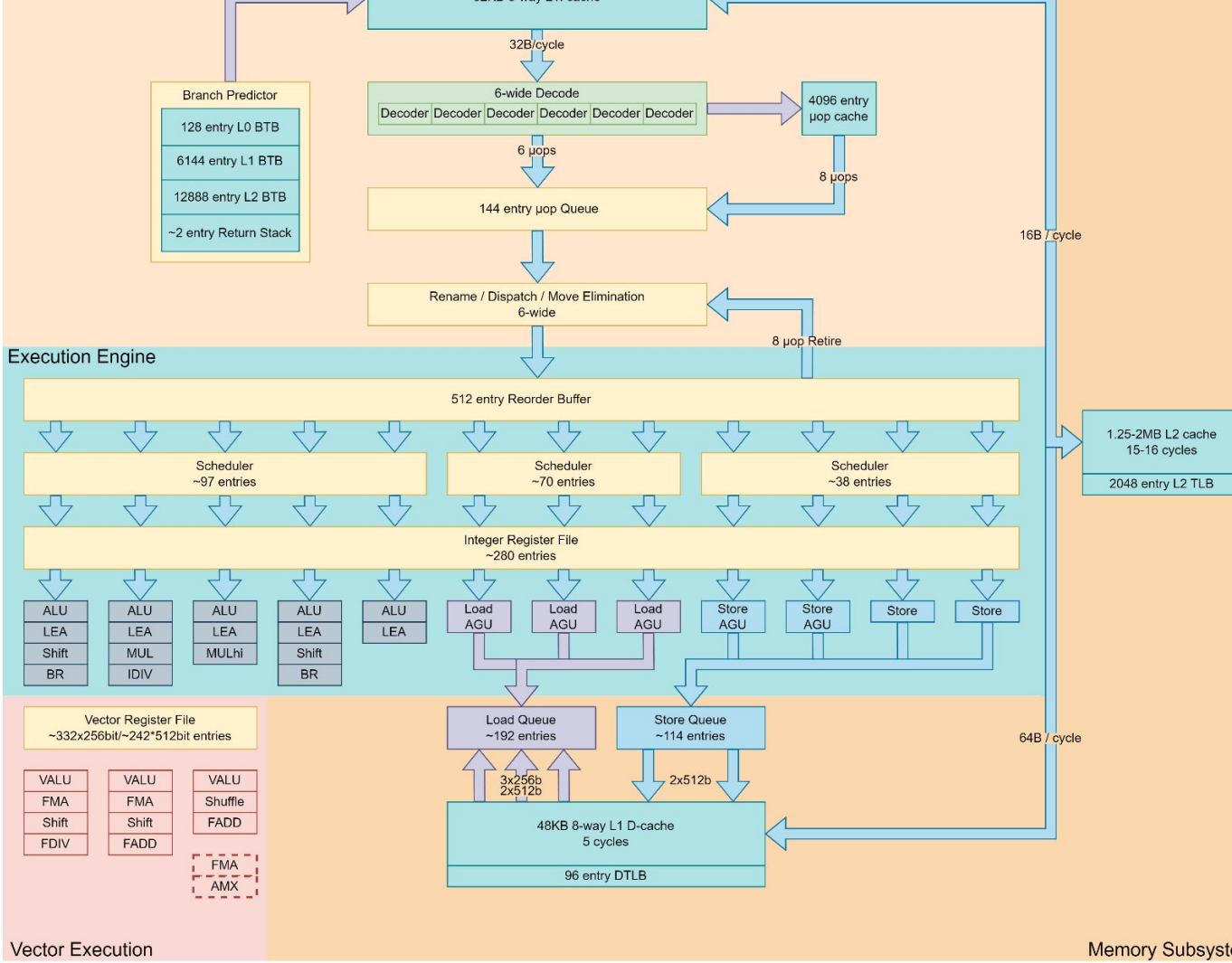
Cool idea: decode more than one instruction at a time



5th trick: superscalar execution is similar to multiprocessing



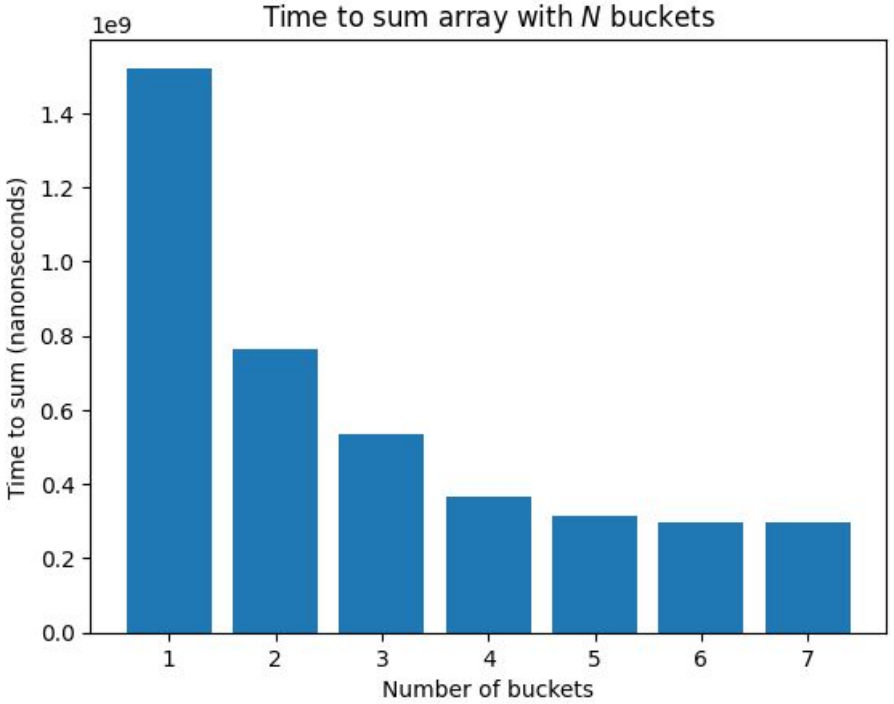
My laptop core design is superscalar

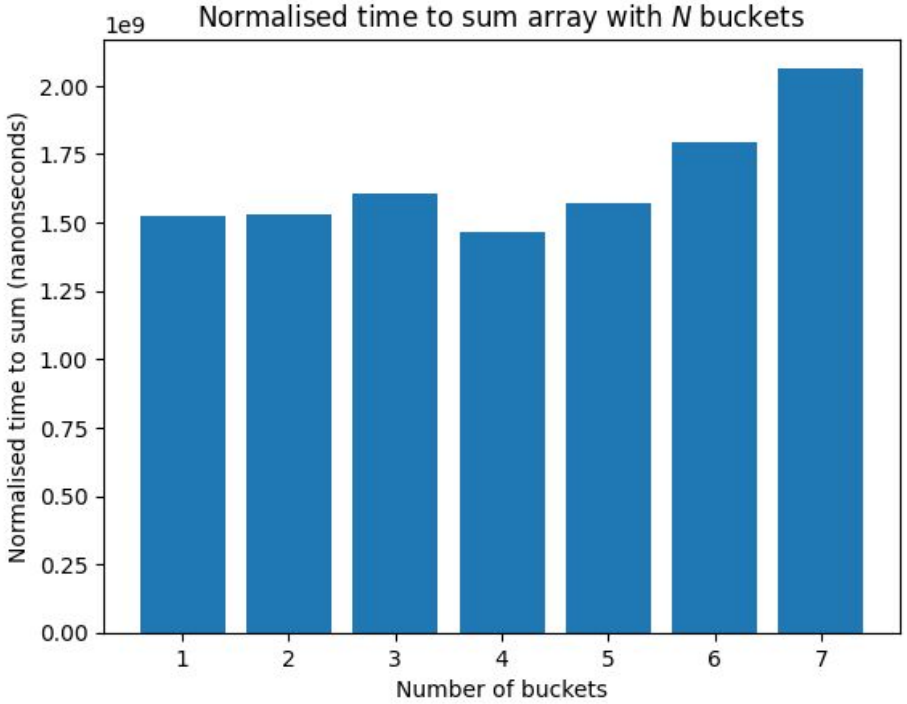


Testing x86 superscalar processing! The code:

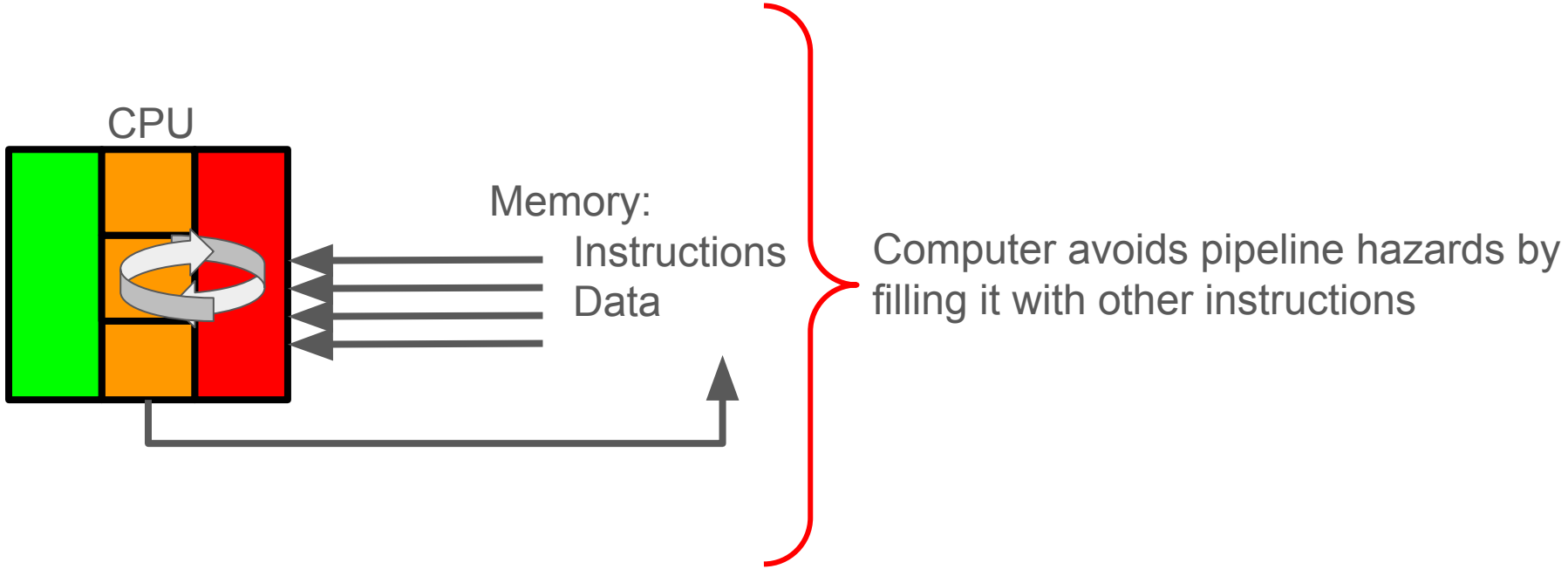
```
int s1 = 0, s2 = 0, s3 = 0, s4 = 0;
auto ini = chrono::high_resolution_clock::now();
for (int i = 0; i < mult; ++i)
{
    for (int j = 0; j < MAXN - 3; j += 4)
    {
        s1 += arr[j];
        s2 += arr[j + 1];
        s3 += arr[j + 2];
        s4 += arr[j + 3];
    }
}
s1 += s2 + s3 + s4;
```

The results show the power of superscalar architectures





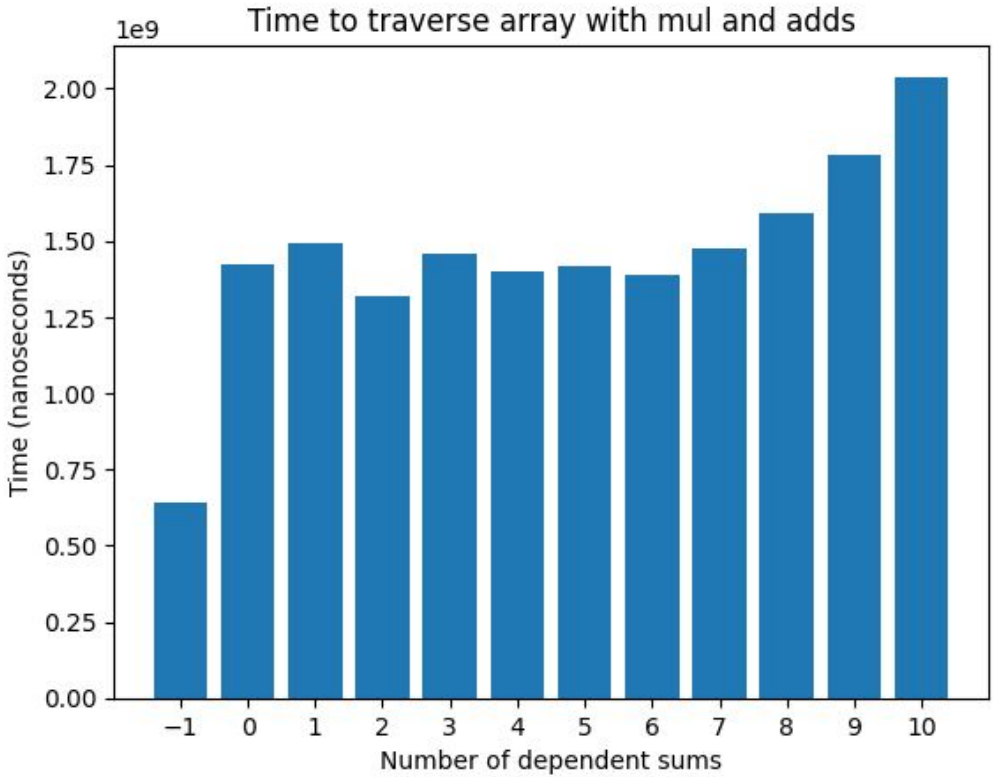
6th trick: executing things out of order is smart!



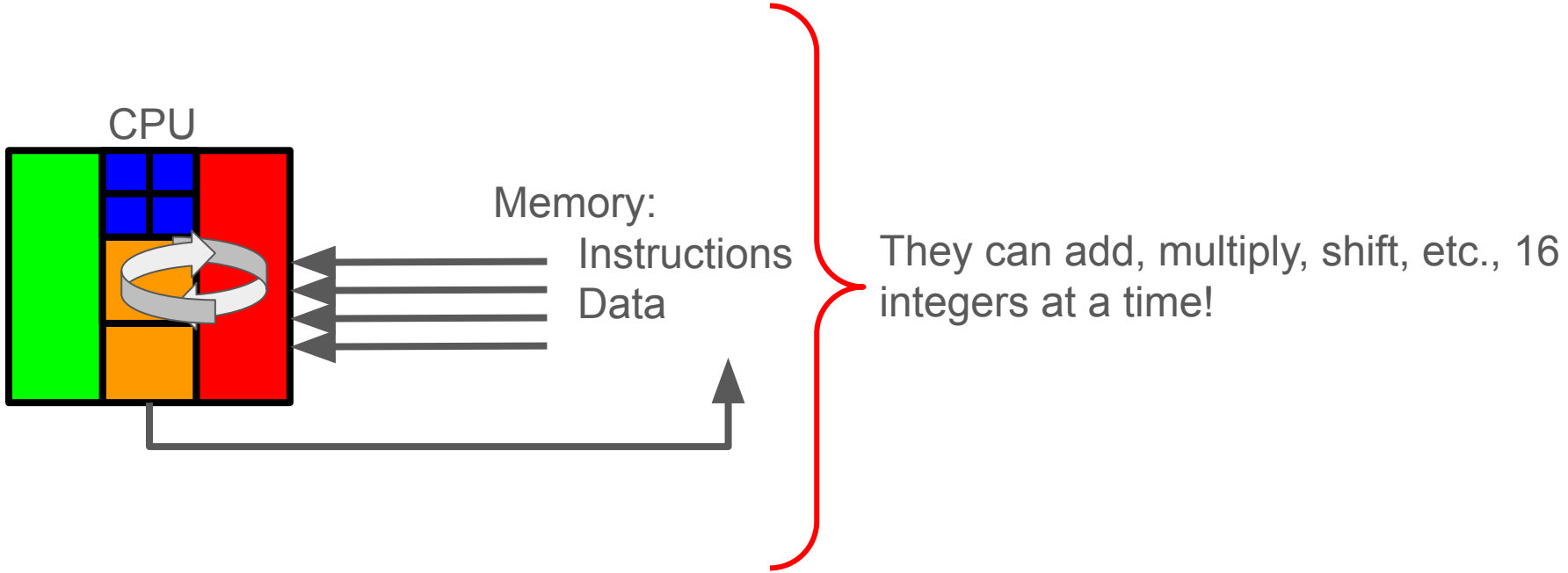
Testing x86 oooe is difficult, but here is the code:

```
for (int i = 0; i < mult; ++i)
{
    for (int j = 0; j < MAXN; ++j)
    {
        mul *= arr[j];
        //mul *= mul;
        //s5 += mul;
        //++sum;
        //++sum;
        //++sum;
        //++sum;
        //++sum;
        //++sum;
        //++sum;
        //++sum;
        //++sum;
        //++sum; //comment the sums to see the effect of oooe
    }
}
```

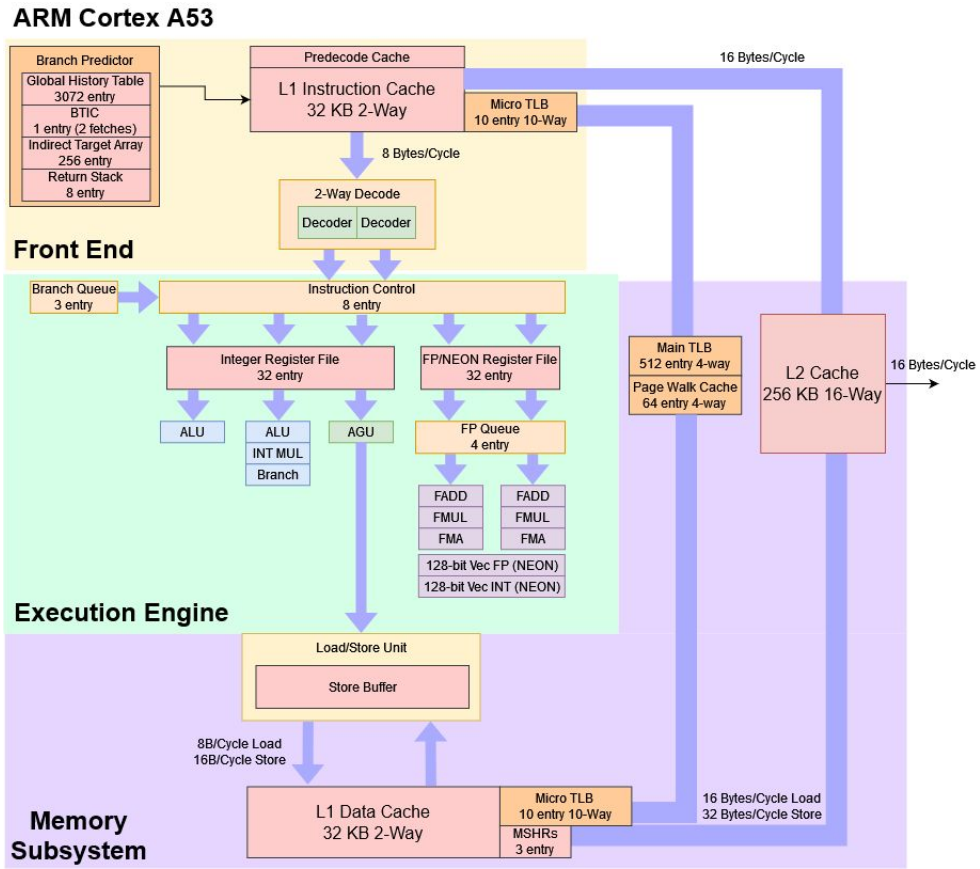
The results show latency hiding



Why act on 1 data per instruction if you can act on multiple?



Raspberry pi core has SIMD

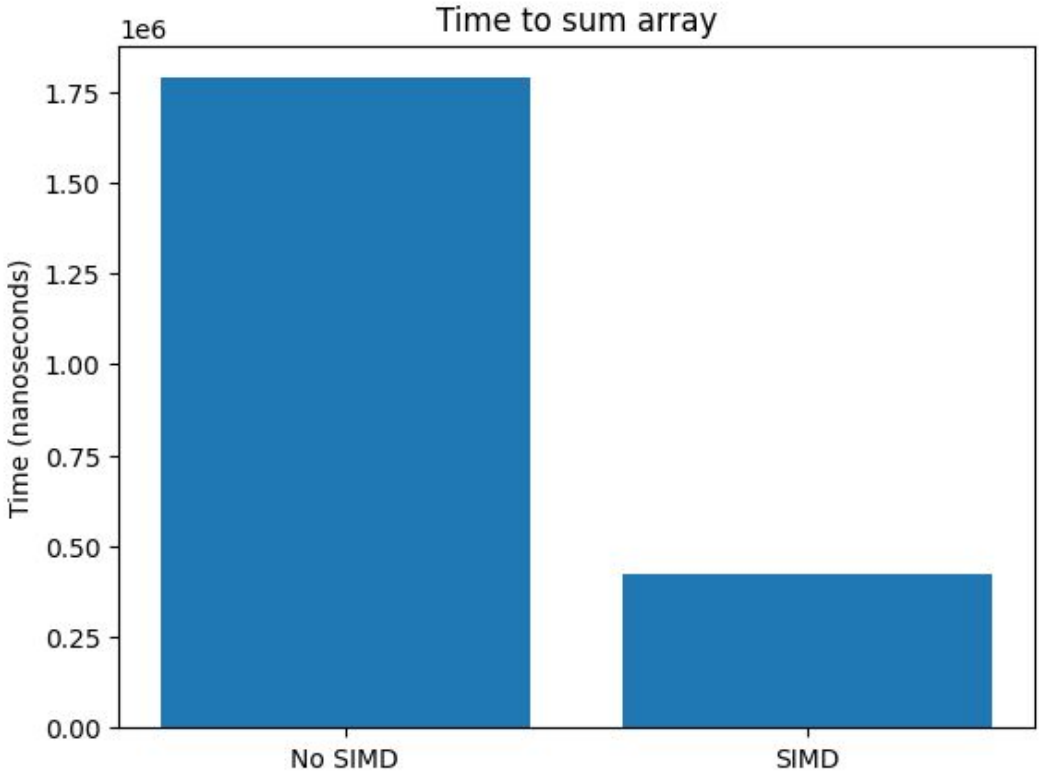


Raspberry pi has (somewhat narrow) SIMD; code:

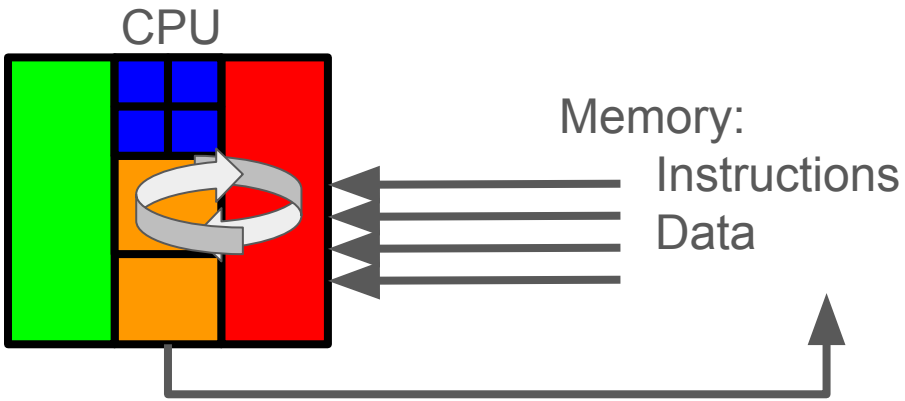
```
for(int j = 0; j < 20; ++j)
    for(int i = 0; i < MAXN; ++i)
        sum += arr[i];
```

(Compiler optimization)

The result is much faster, as expected

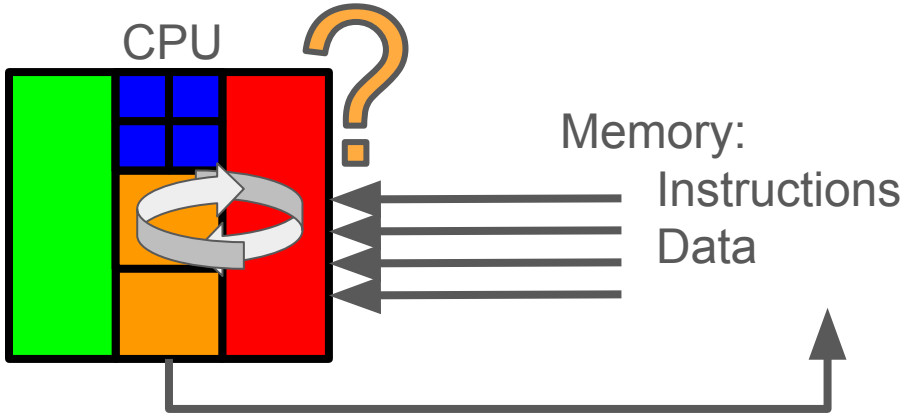


If the data is still in pipeline or underway, what to do? :/




```
if(arr[5] > 10){
    //blabla
}
```

Computers just predict where to go next

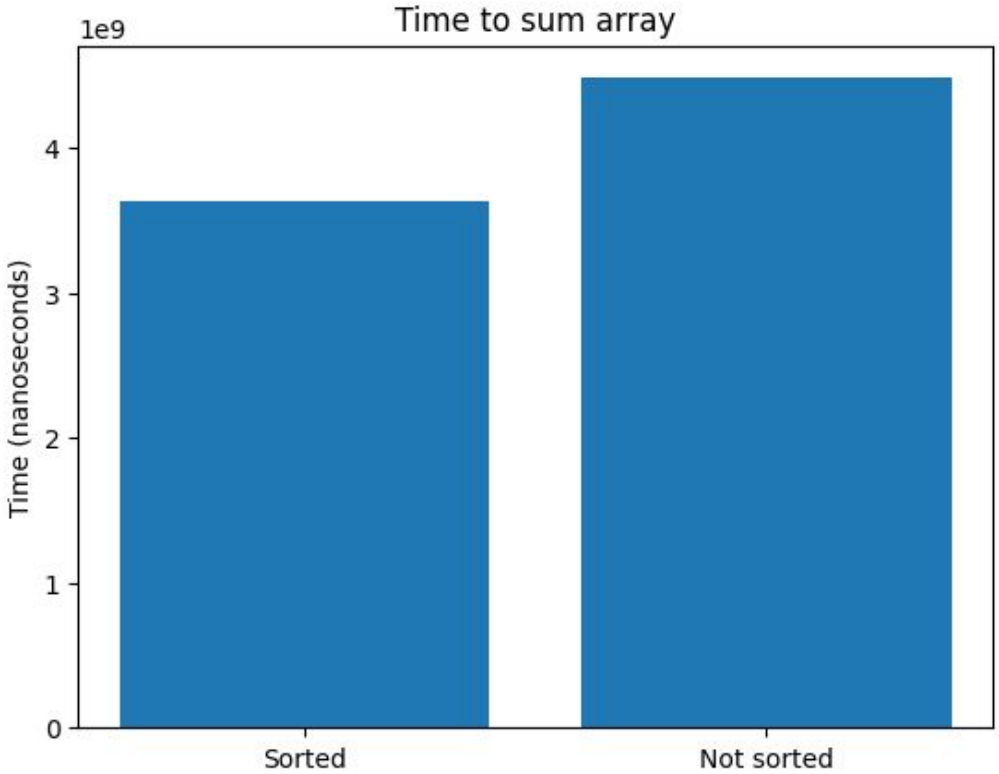


Raspberry pis have very simple branch prediction we test

```
sort(arr, arr + MAXN);
auto ini = chrono::high_resolution_clock::now();
for(int j = 0; j < 5; ++j)
    for(int i = 0; i < MAXN; ++i)
        if(arr[i] > INT_MAX/2)
            ++sum;
auto end = chrono::high_resolution_clock::now();
```

```
auto ini = chrono::high_resolution_clock::now();
for(int j = 0; j < 5; ++j)
    for(int i = 0; i < MAXN; ++i)
        if(arr[i] > INT_MAX/2)
            ++sum;
auto end = chrono::high_resolution_clock::now();
```

The results are better, but not by much



Conclusions

- Engineering computers is very hard
- Understanding how computers are engineered is important to performance*
- *A lot of optimizations are designed to work seamlessly or are well integrated with compilers
- Be aware of them, but trust the compiler!

Your turn!

- You are each given an ordered set of computer instructions
  - A X <-- Y + Z = Add Y and Z and store in X
  - L X <-- Q = Load computer memory from Q and store in X
  - S X = Store the value of X somewhere in memory
- The computer has
  - Decode width 2
  - 3 Pipeline stages: fetch, decode, execute
  - 2 ALUs
  - 2 Load execution units
  - 2 Store execution units
- Find the most efficient way to reorder the commands, without changing the result. Equal registers cannot be used simultaneously by different instructions