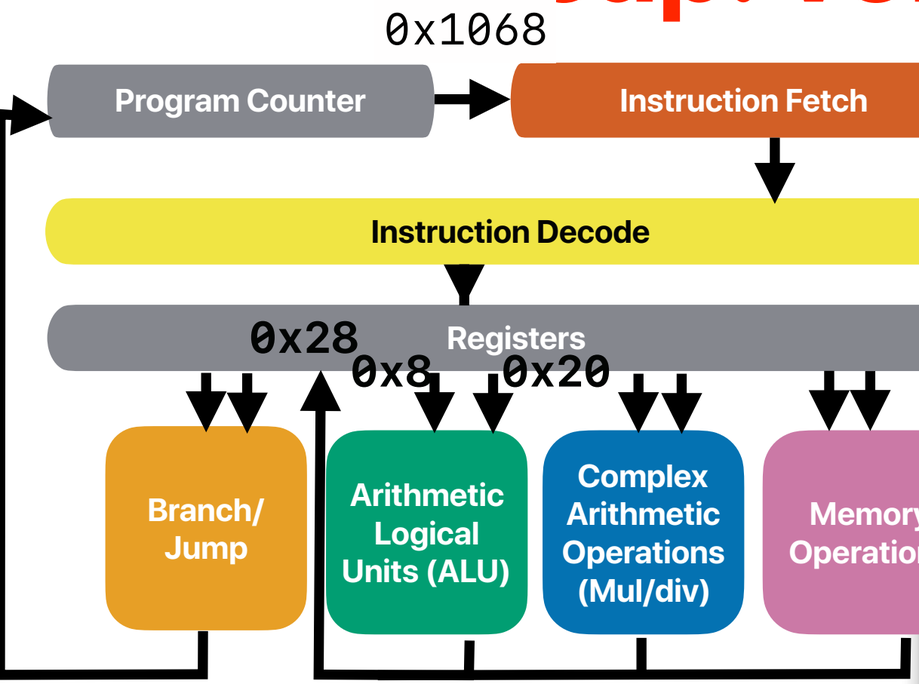
Modern Processor Design (I): in the pipeline

Hung-Wei Tseng

Recap: von Neumann architecture

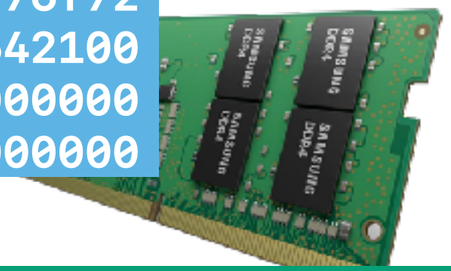


Processor

4883ec08

sub $0x8,%rsp

Instructions	Data
f30f1efa	08400000
4883ec08	00000100
488d3d95	02004865
0f0000e8	6c6c6f2c
dcffffff	20776f72
31c04883	6c642100
c408c30f	00000000
1f440000	00000000



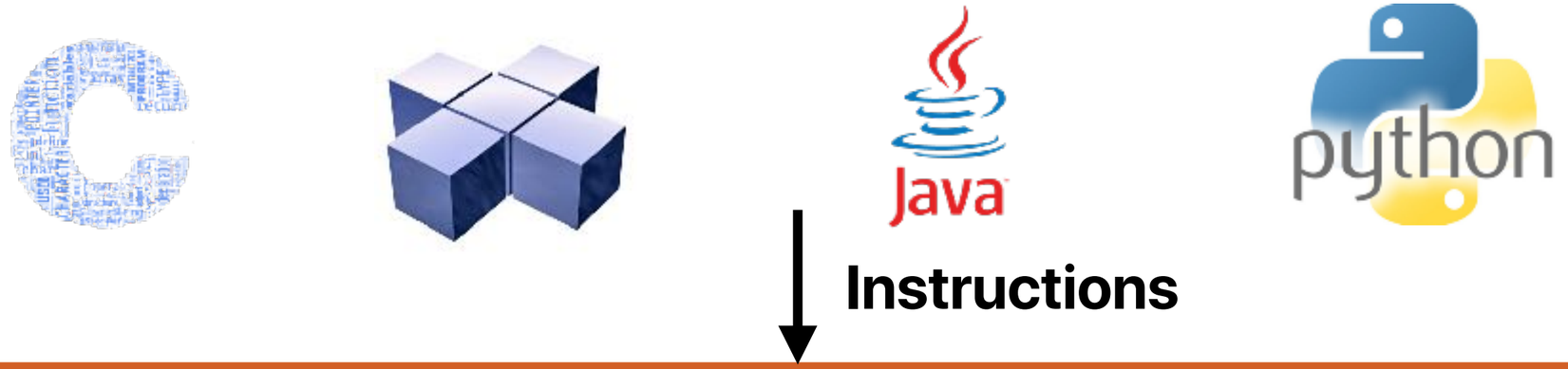
Memory

```
int main(){
    pr
}
```

By loading different programs into memory, your computer can perform different functions

Storage

Recap: Microprocessor — a collection of functional units



3:00

Where will you go for lunch on campus if you're short on time? Why?

Tricky C/C++ programming questions?

- Give a fastest way to multiply any number by 9
- How to measure the size of any variable without "sizeof" operator?.
- How to measure the size of any variable without using "sizeof" operator?
- Write code snippets to swap two variables in five different ways
- How to swap between first & 2nd byte of an integer in one line statement?
- What is the efficient way to divide a no. by 4?
- Suggest an efficient method to count the no. of 1's in a 32 bit no. Remember without using loop & testing each bit.
- Test whether a no. is power of 2 or not.
- How to check endianness of the computer.
- Write a C-program which does the addition of two integers without using '+' operator.
- Write a C-program to find the smallest of three integers without using any of the comparision operators.
- Find the maximum & minimum of two numbers in a single line without using any condition & loop.
- What "condition" expression can be used so that the following code snippet will print Hello world.
- How to print number from 1 to 100 without using conditional operators.
- WAP to print 100 times "Hello" without using loop & goto statement.
- Write the equivalent expression for x%8.

https://www.emblogic.com/blog/12/tricky-c-interview-questions/

https://www.pollev.com/hungweitseng close in 1:30

Which one is faster?

- Considering the following two implementations that deliver the same result (all numbers are integers). Without turning on compiler optimizations, which one is faster and why?

A

```
for(i=0;i<1000000000;i++) {
    sum+=data[index];
    index =
(data[index]*15)%131072;
}
```

B

```
for(i=0;i<1000000000;i++) {
    sum+=data[index];
    index =
((data[index]<<4)-data[index])%131072;
}
```

A. Version A is faster as the CPI is lower
B. Version A is faster as the IC is lower
C. Version B is faster as the CPI is lower
D. Version B is faster as the IC is lower
E. They're about the same — sometimes A is faster, sometimes B is faster

Which one is faster?

- Considering the following two implementations that deliver the same result (all numbers are integers). Without turning on compiler optimizations, which one is faster and why?

A

```
for(i=0;i<1000000000;i++) {
    sum+=data[index];
    index =
(data[index]*15)%131072;
}
```

B

```
for(i=0;i<1000000000;i++) {
    sum+=data[index];
    index =
((data[index]<<4)-data[index])%131072;
}
```

A. Version A is faster as the CPI is lower
B. Version A is faster as the IC is lower
C. Version B is faster as the CPI is lower
D. Version B is faster as the IC is lower
E. They're about the same — sometimes A is faster, sometimes B is faster

Which one is faster?

- Considering the following two implementations that deliver the same result (all numbers are integers). Without turning on compiler optimizations, which one is faster and why?

A

```
for(i=0;i<1000000000;i++) {
    sum+=data[index];
    index =
(data[index]*15)%131072;
}
```

B

```
for(i=0;i<1000000000;i++) {
    sum+=data[index];
    index =
((data[index]<<4)-data[index])%131072;
}
```

A. Version A is faster as the CPI is lower
B. Version A is faster as the IC is lower
C. Version B is faster as the CPI is lower
D. Version B is faster as the IC is lower
E. They're about the same — sometimes A is faster, sometimes B is faster

Recap: Demo (3) — Bitwise operations?

d. /* one line statement using bit-wise operators */ (most efficient)
a^=b^=a^=b;

The order of evaluation is from right to left. This is same as in approach (c) but the three statements are compounded into one statement.

A

```
void regswap(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}
```

B

```
void xorswap(int* a, int* b) {
    *a ^= *b ^= *a = *b;
}
```

Recap: Why adding a sort makes it faster

- Why the sorting the array speed up the code despite the increased instruction count?

```
if(option)
    std::sort(data, data + arraySize);

for (unsigned i = 0; i < 100000; ++i) {
    int threshold = std::rand();
    for (unsigned i = 0; i < arraySize; ++i) {
        if (data[i] >= threshold)
            sum ++;
    }
}
```

Outline

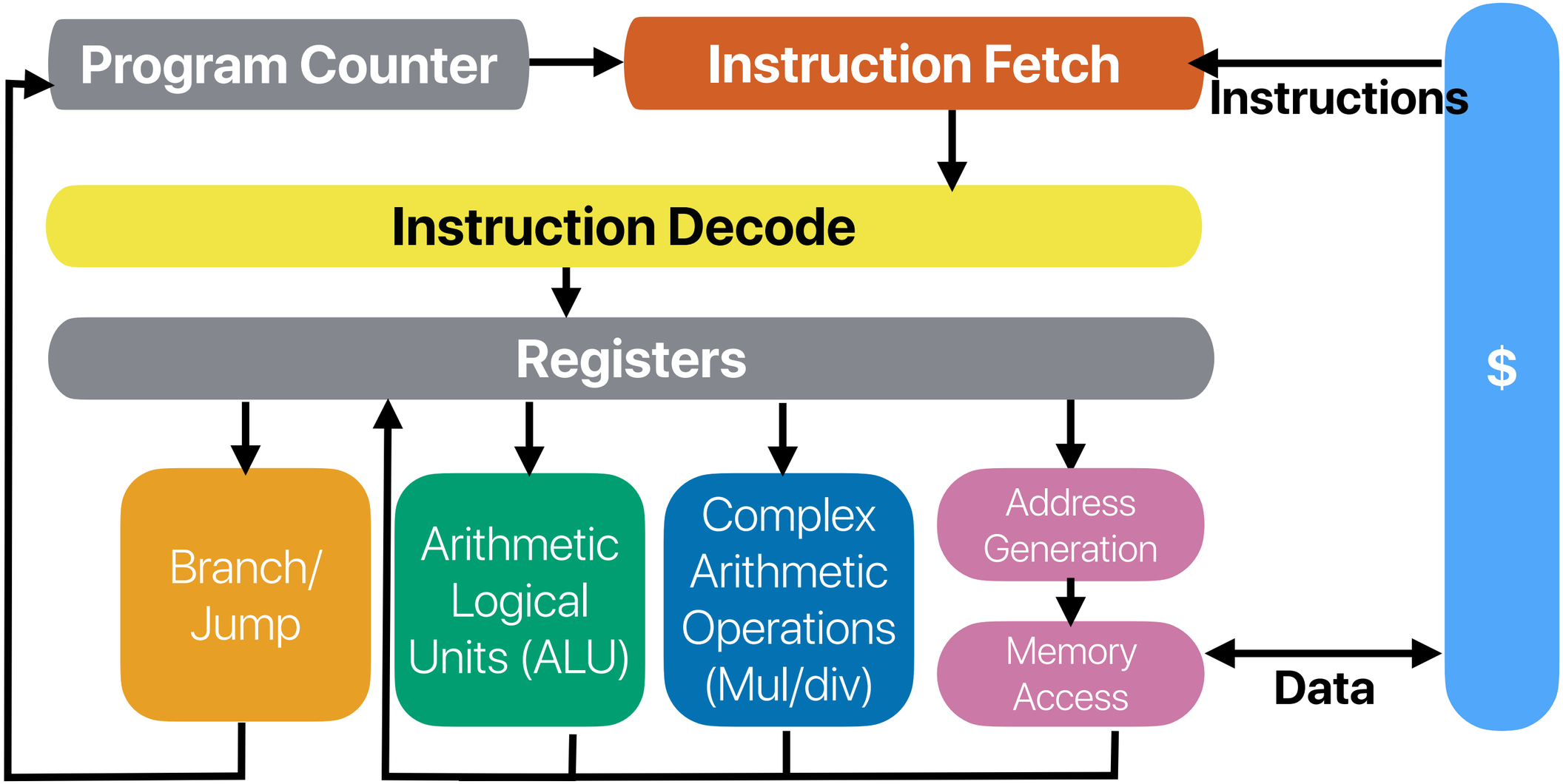
- Recap: the concept of a processor
- Pipelined Processor
- Pipeline Hazards
  - Structural Hazards
  - Control Hazards
  - Data Hazards
- Solutions to structural hazards

Basic Processor Design

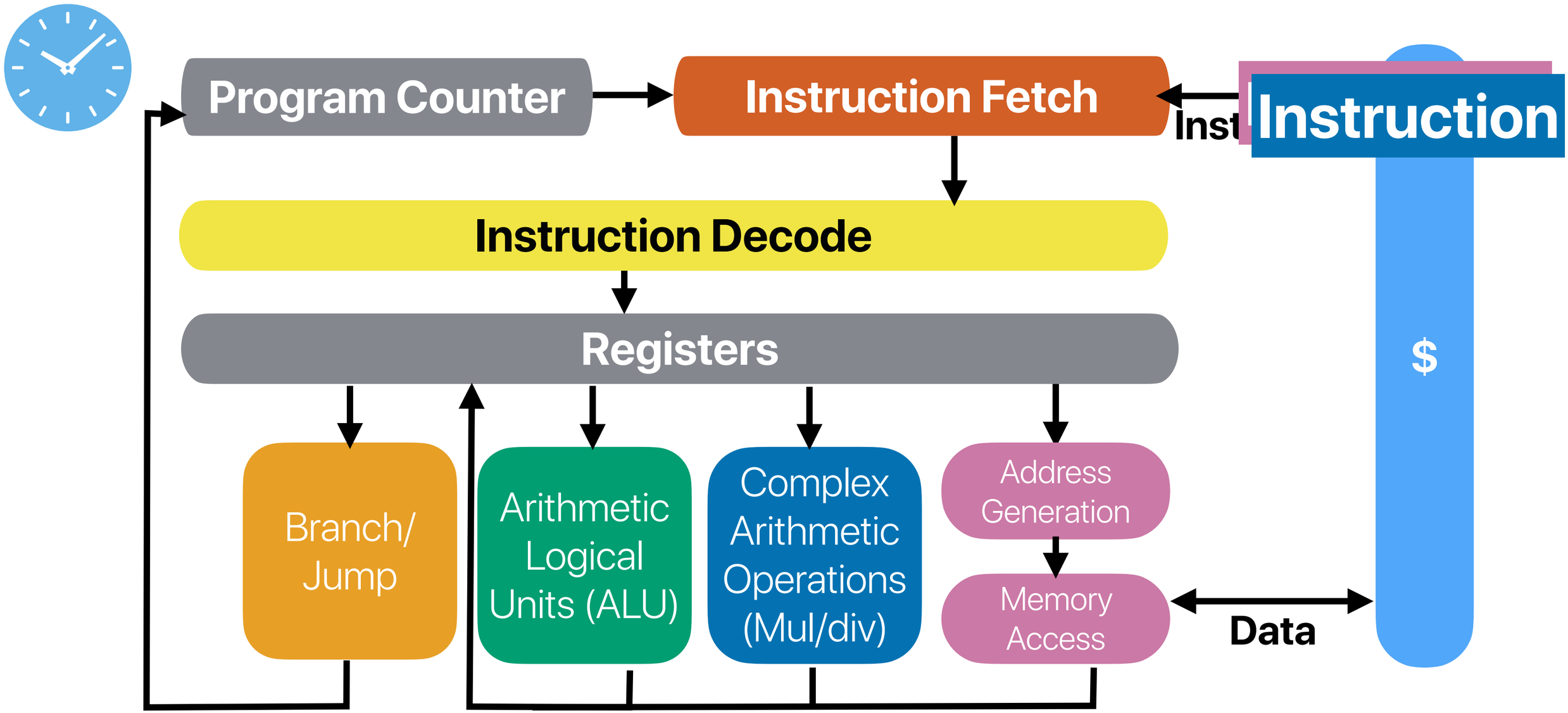
The “life” of an instruction

- Instruction Fetch (**IF**) — fetch the instruction from memory
- Instruction Decode (**ID**)
  - Decode the instruction for the desired operation and operands
  - Reading source register values
- Execution (**EX**)
  - ALU instructions: Perform ALU operations
  - Conditional Branch: Determine the branch outcome (taken/not taken)
  - Memory instructions: Determine the effective address for data memory access
- Data Memory Access (**MEM**) — Read/write memory
- Write Back (**WB**) — Present ALU result/read value in the target register
- Update PC
  - If the branch is taken — set to the branch target address
  - Otherwise — advance to the next instruction — current PC + 4

Functional Units of a Microprocessor

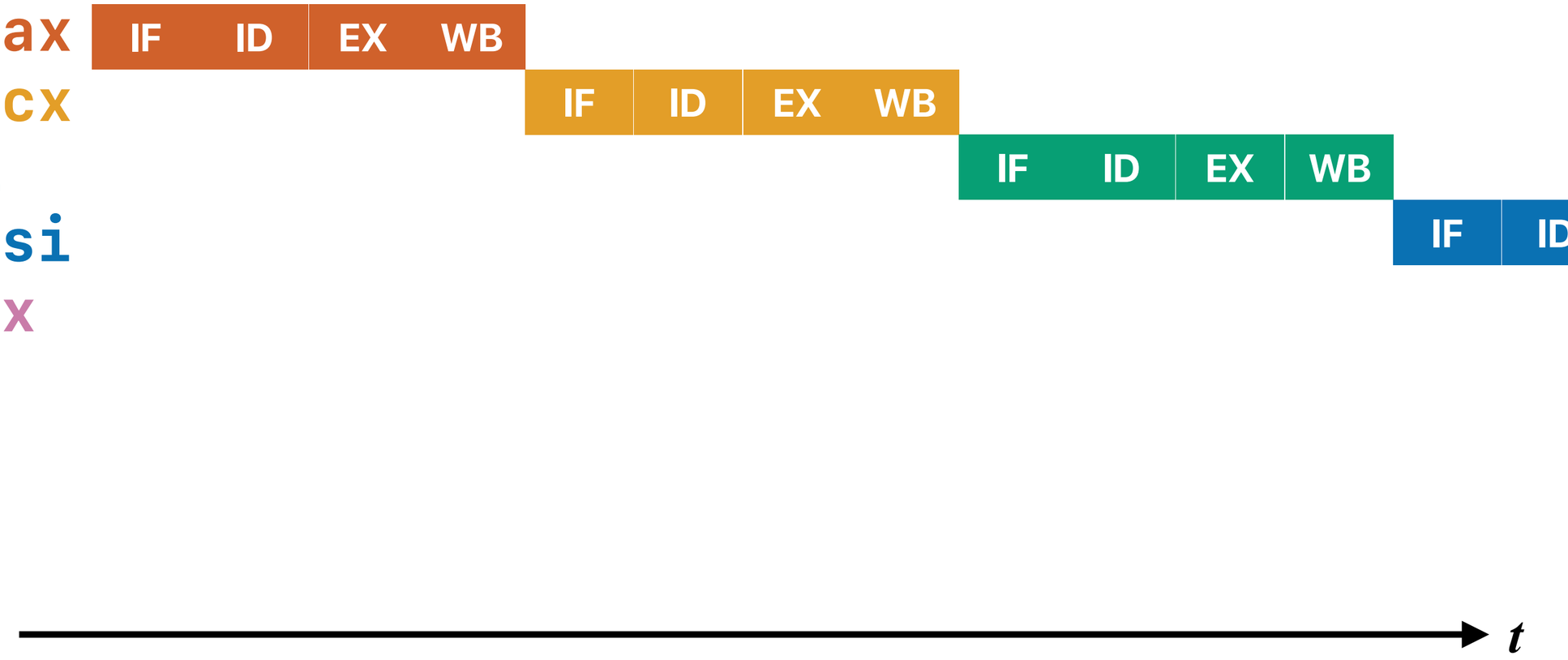


If we want to perform one instruction each cycle…



Simple implementation w/o branch

```
addl   %eax, %eax
addl   %rdi, %ecx
addq   $4, %r11
testl %esi, %esi
movl   $10, %edx
```

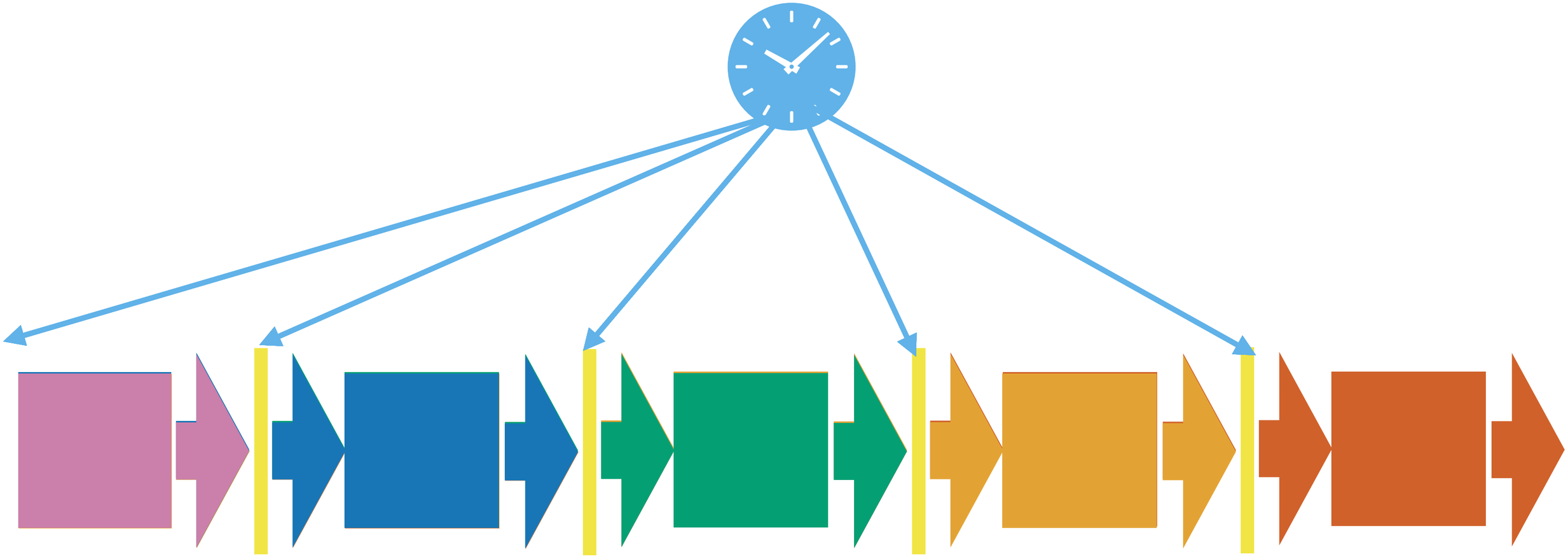


Pipelining

Pipelining

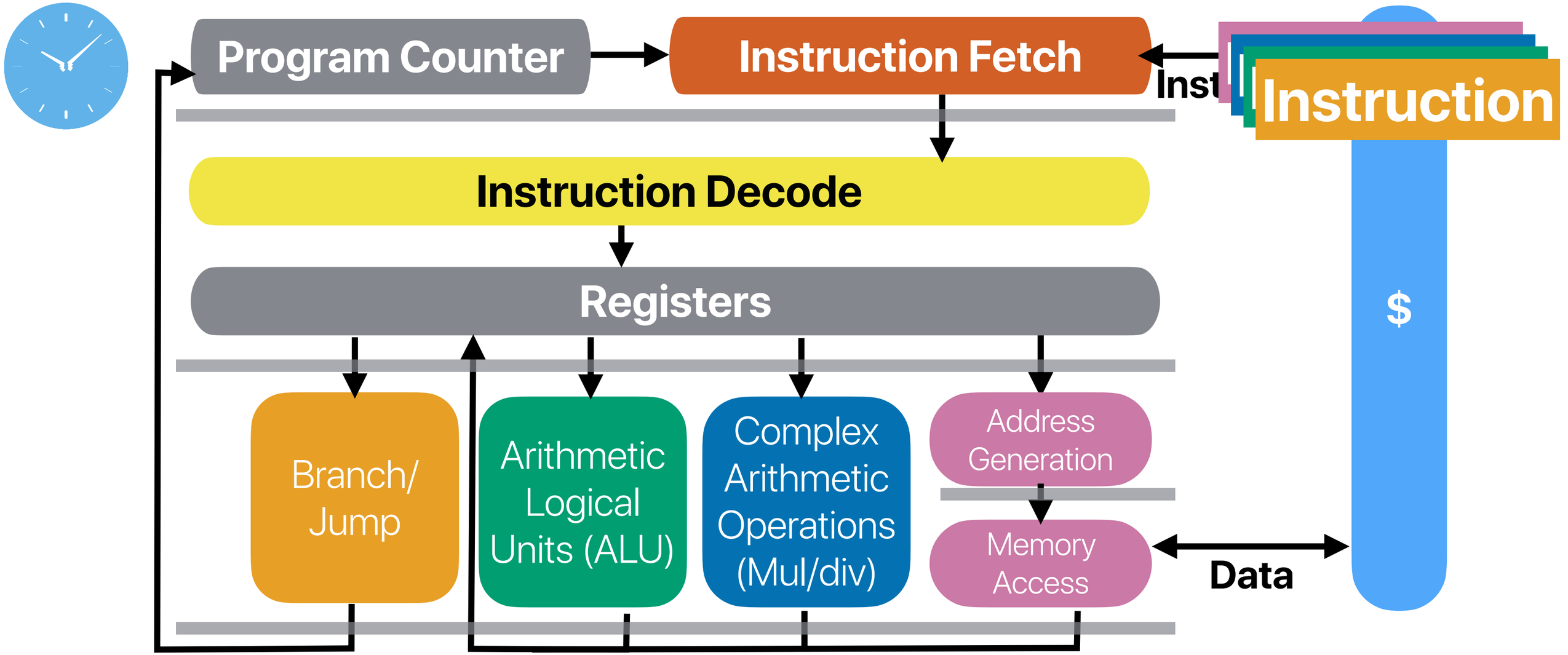
- Different parts of the processor works on different instructions simultaneously
- A processor is now working on multiple instructions from the same program (though on different stages) simultaneously.
  - ILP: **Instruction-level parallelism**
- A **clock** signal controls and synchronize the beginning and the end of each part of the work
- A **pipeline register** between different parts of the processor to keep intermediate results necessary for the upcoming work

Pipelining



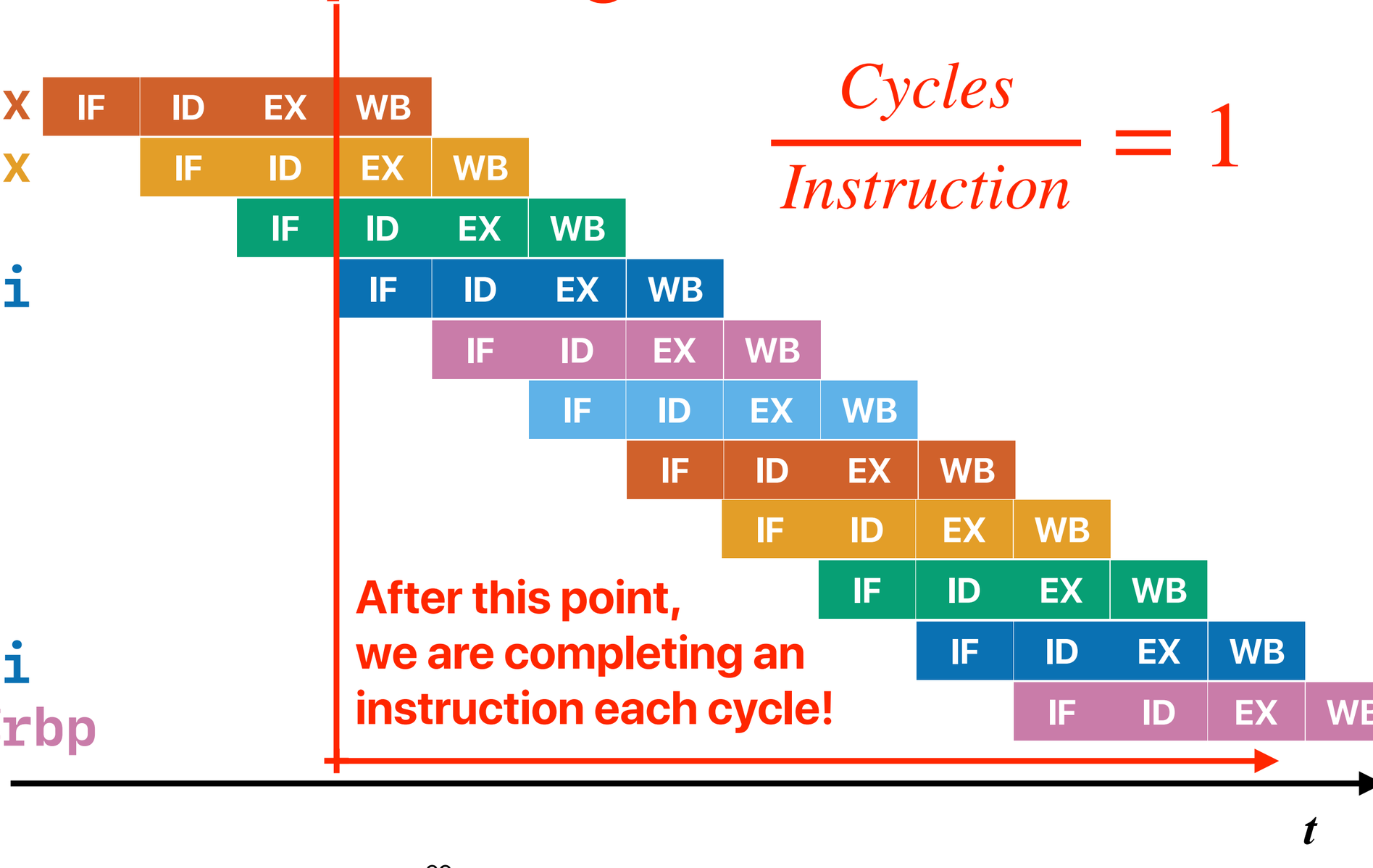
Pipelining

Pipelined execution



Pipelining

```
addl   %eax, %eax
addl   %rdi, %ecx
addq   $4, %r11
testl %esi, %esi
movl   $10, %edx
pushq %r12
pushq %rbp
pushq %rbx
subq   $8, %rsp
addl   %rsi, %rdi
movslq  %eax, %rbp
```



$$\frac{Cycles}{Instruction} = 1$$

https://www.pollev.com/hungweitseng close in 1:30

How well can we pipeline?

- With a pipelined design, the processor is supposed to deliver the outcome of an instruction each cycle. For the following code snippet, how many pairs of instructions are preventing the pipeline from generating results in back-to-back cycles?

```
①       xorl    %eax, %eax
② L3:   movl    (%rdi), %ecx
③       addl    %ecx, %eax
④       addq    $4, %rdi
⑤       cmpq    %rdx, %rdi
⑥       jne     .L3
⑦       ret
```

```
for(i = 0; i < count; i++) {
    s += a[i];
}
return s;
```

A. 1
B. 2
C. 3
D. 4
E. 5

How well can we pipeline?

- With a pipelined design, the processor is supposed to deliver the outcome of an instruction each cycle. For the following code snippet, how many pairs of instructions are preventing the pipeline from generating results in back-to-back cycles?

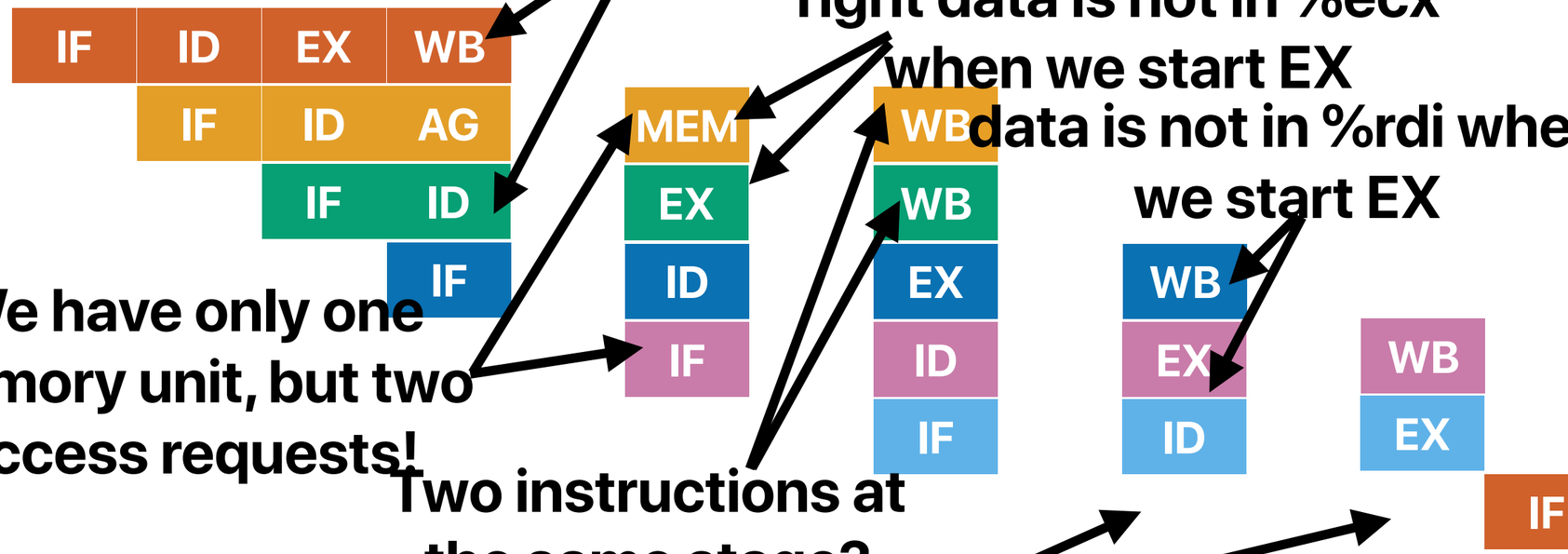
```
①       xorl    %eax, %eax
②  L3:  movl    (%rdi), %ecx
③       addl    %ecx, %eax
④       addq    $4, %rdi
⑤       cmpq    %rdx, %rdi
⑥       jne     .L3
⑦       ret
```

```
for(i = 0; i < count; i++) {
    s += a[i];
}
return s;
```

A. 1
B. 2
C. 3
D. 4
E. 5

Pipelining

```
① xorl %eax, %eax
② movl (%rdi), %ecx
③ addl %ecx, %eax
④ addq $4, %rdi
⑤ cmpq %rdx, %rdi
⑥ jne  .L3
⑦ ret
```



Both (1) and (3) are attempting to access %eax

right data is not in %ecx when we start EX

data is not in %rdi whe we start EX

We have only one memory unit, but two access requests!

Two instructions at the same stage?

We cannot know if we should fetch (7) or (2) before the EX is done

How well can we pipeline?

- With a pipelined design, the processor is supposed to deliver the outcome of an instruction each cycle. For the following code snippet, how many pairs of instructions are preventing the pipeline from generating results in back-to-back cycles?

```
①       xorl    %eax, %eax
② L3:   movl    (%rdi), %ecx
③       addl    %ecx, %eax
④       addq    $4, %rdi
⑤       cmpq    %rdx, %rdi
⑥       jne     .L3
⑦       ret
```

```
for(i = 0; i < count; i++) {
    s += a[i];
}
return s;
```

A. 1
B. 2
C. 3
D. 4
E. 5

Takeaways: pipeline processors

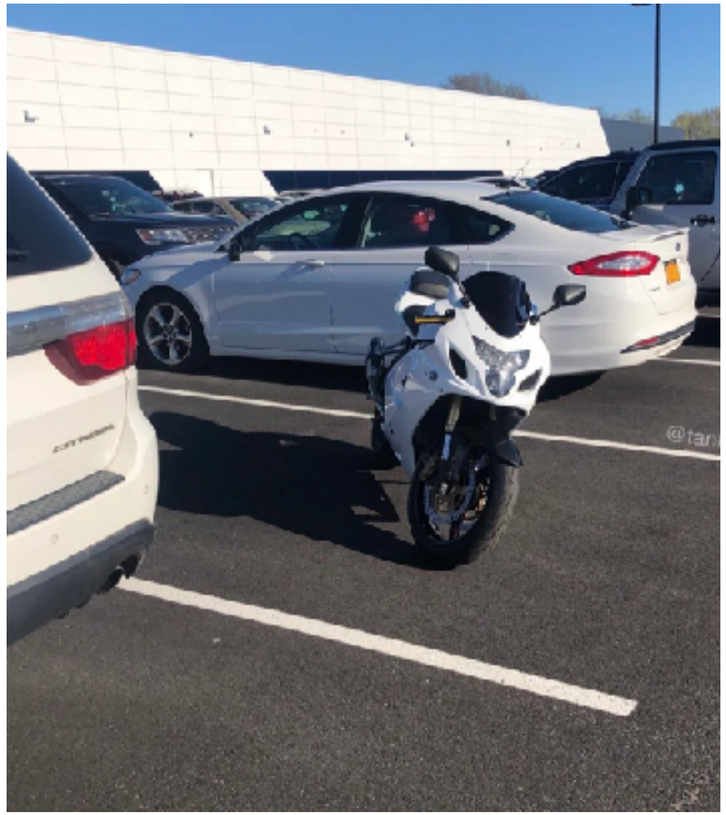
- Pipelining helps to improve the throughput of processors
  - Allowing shorter cycle time as each cycle only make progress for part of an instruction
  - Different pipeline stages work on different instructions concurrently
  - Theoretical CPI remains the same as single-cycle design and the throughput/speedup is in proportion to the speedup of cycle time

Pipeline hazards

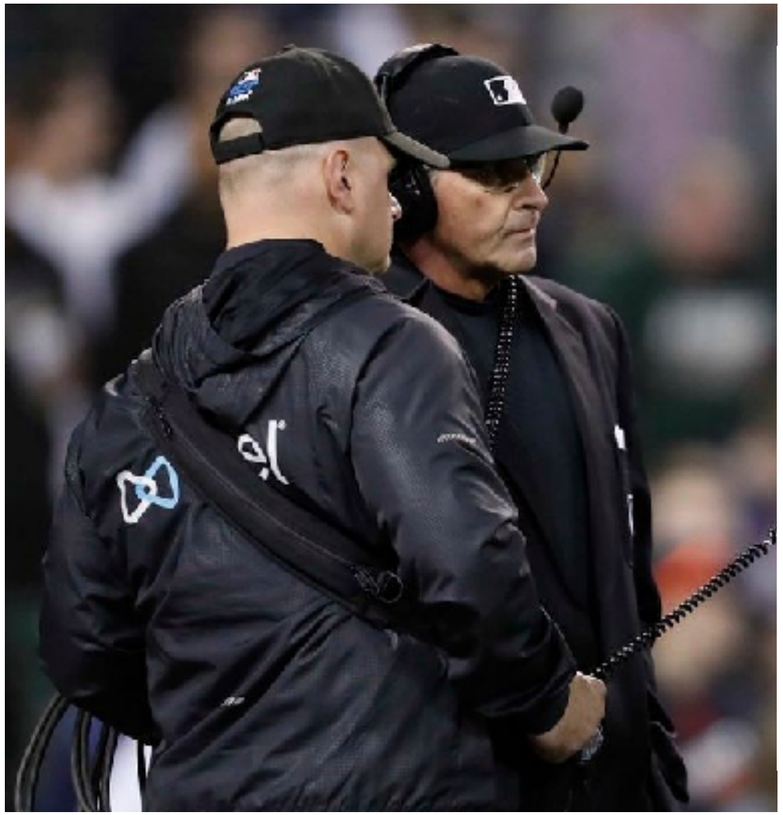
Three types of pipeline hazards

- Structural hazards — resource conflicts cannot support simultaneous execution of instructions in the pipeline
- Control hazards — the PC can be changed by an instruction in the pipeline
- Data hazards — an instruction depending on a the result that's not yet generated or propagated when the instruction needs that

Pipeline Hazards



Structural Hazards



Control Hazards

Data Hazards

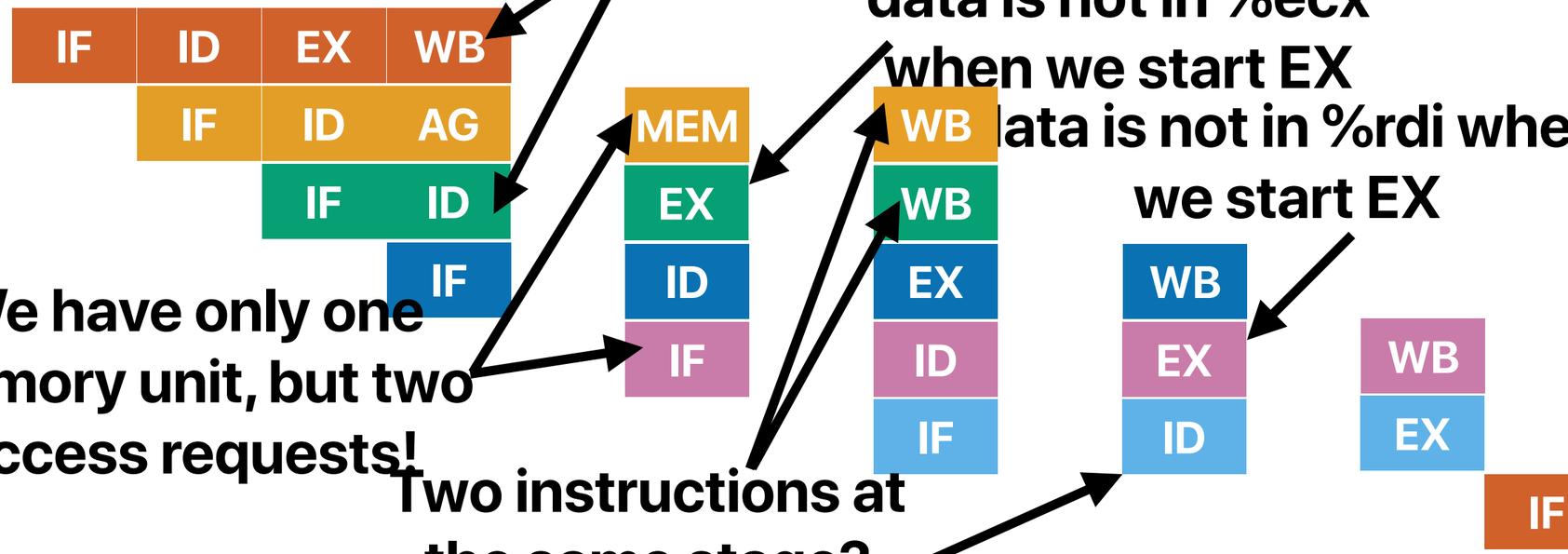
https://www.pollev.com/hungweitseng close in 1:30

Pipelining

```
① xorl %eax, %eax
② movl (%rdi), %ecx
③ addl %ecx, %eax
④ addq $4, %rdi
⑤ cmpq %rdx, %rdi
⑥ jne  .L3
⑦ ret
```

- How many of the "hazards" are data hazards?

A. 0
B. 1
C. 2
D. 3
E. 4



Both (1) and (3) are attempting to access %eax

data is not in %ecx when we start EX

ata is not in %rdi whe we start EX

We have only one memory unit, but two access requests!

Two instructions at the same stage?

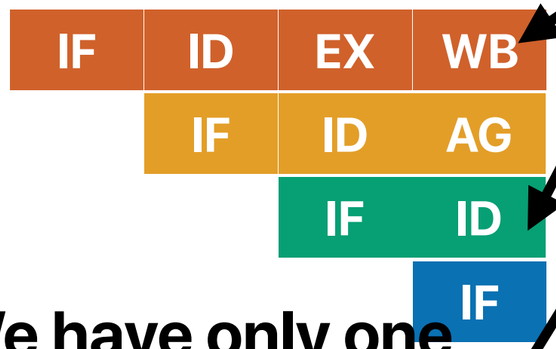
We cannot know if we should fetch (7) or (2) before the EX is done

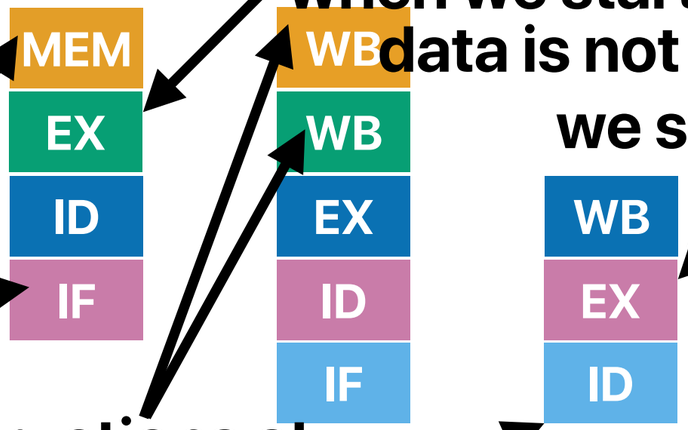
Pipelining

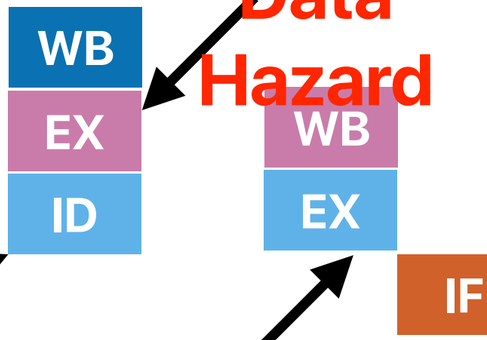
```
① xorl %eax, %eax
② movl (%rdi), %ecx
③ addl %ecx, %eax
④ addq $4, %rdi
⑤ cmpq %rdx, %rdi
⑥ jne  .L3
⑦ ret
```

- How many of the "hazards" are data hazards?

A. 0
B. 1
C. 2
D. 3
E. 4







Both (1) and (3) are attempting to access %eax

Structural Hazard

Data Hazard

data is not in %ecx when we start EX

data is not in %rdi when we start EX

Data Hazard

We have only one memory unit, but two access requests!

Structural Hazard

Structural Hazard

Two instructions at the same stage?

We cannot know if we should fetch (7) or (2) before the EX is done

(6) may not have the outcome from (5)

Control Hazard

https://www.pollev.com/hungweitseng close in 1:30

Why is A is faster?

A

```
void regswap(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}
```

B

```
void xorswap(int* a, int* b) {
    *a ^= *b;
    *b ^= *a;
    *a ^= *b;
}
```

- What's the main reason why version B cannot outperform version A on modern processors?
  A. Control hazards
  B. Data hazards
  C. Structural hazards

Why is A is faster?

A

```
void regswap(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}
```

B

```
void xorswap(int* a, int* b) {
    *a ^= *b;
    *b ^= *a;
    *a ^= *b;
}
```

- What's the main reason why version B cannot outperform version A on modern processors?
  A. Control hazards
  B. Data hazards
  C. Structural hazards

Takeaways: pipeline processors

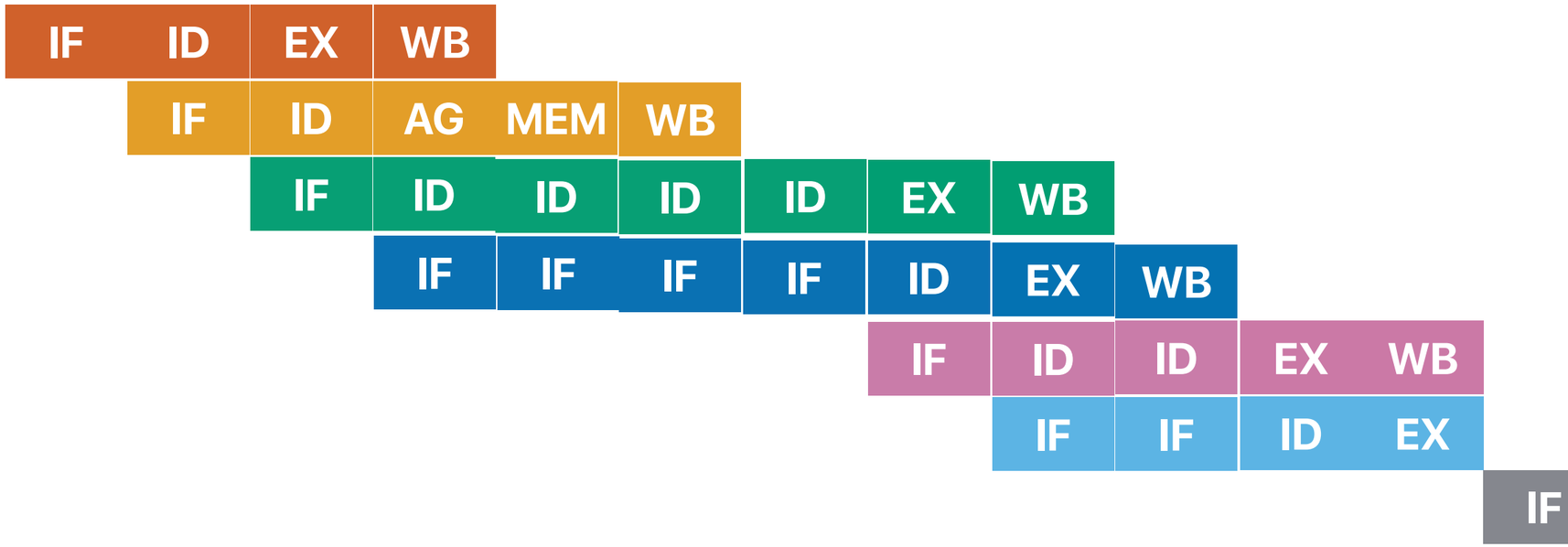
- Pipelining helps to improve the throughput of processors
  - Allowing shorter cycle time as each cycle only make progress for part of an instruction
  - Different pipeline stages work on different instructions concurrently
  - Theoretical CPI remains the same as single-cycle design and the throughput/speedup is in proportion to the speedup of cycle time
- Pipeline hazards prevent us from reaching the theoretical CPI
  - Structural hazards
  - Control hazards
  - Data hazards

Stall — the universal solution to pipeline hazards

Stall whenever we have a hazard

- Stall: the hardware allows the earlier instruction to proceed, all later instructions stay at the same stage
- Disable the pipeline register update for later instructions
- The stalled instructions still have the same input from the pipeline registers

```
① xorl %eax, %eax
② movl (%rdi), %ecx
③ addl %ecx, %eax
④ addq $4, %rdi
⑤ cmpq %rdx, %rdi
⑥ jne  .L3
⑦ ret
```



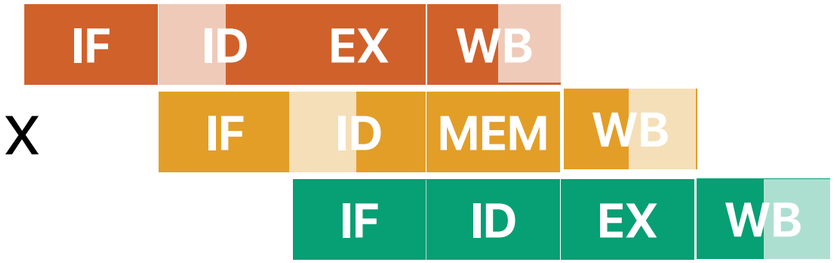
Slow! — 4 additional cycles

Structural Hazards

Dealing with the conflicts between ID/WB

- The same register cannot be read/written at the same cycle
- Better solution: write early, read late
  - Writes occur at the clock edge and complete long enough before the end of the clock cycle.
  - This leaves enough time for outputs to settle for reads
  - The revised register file is the default one from now!

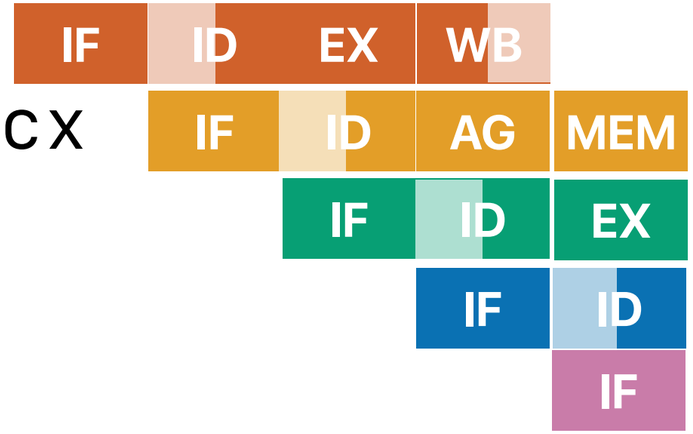
```
① xorl %eax, %eax
② movl (%rdi), %ecx
③ addl %ecx, %eax
```

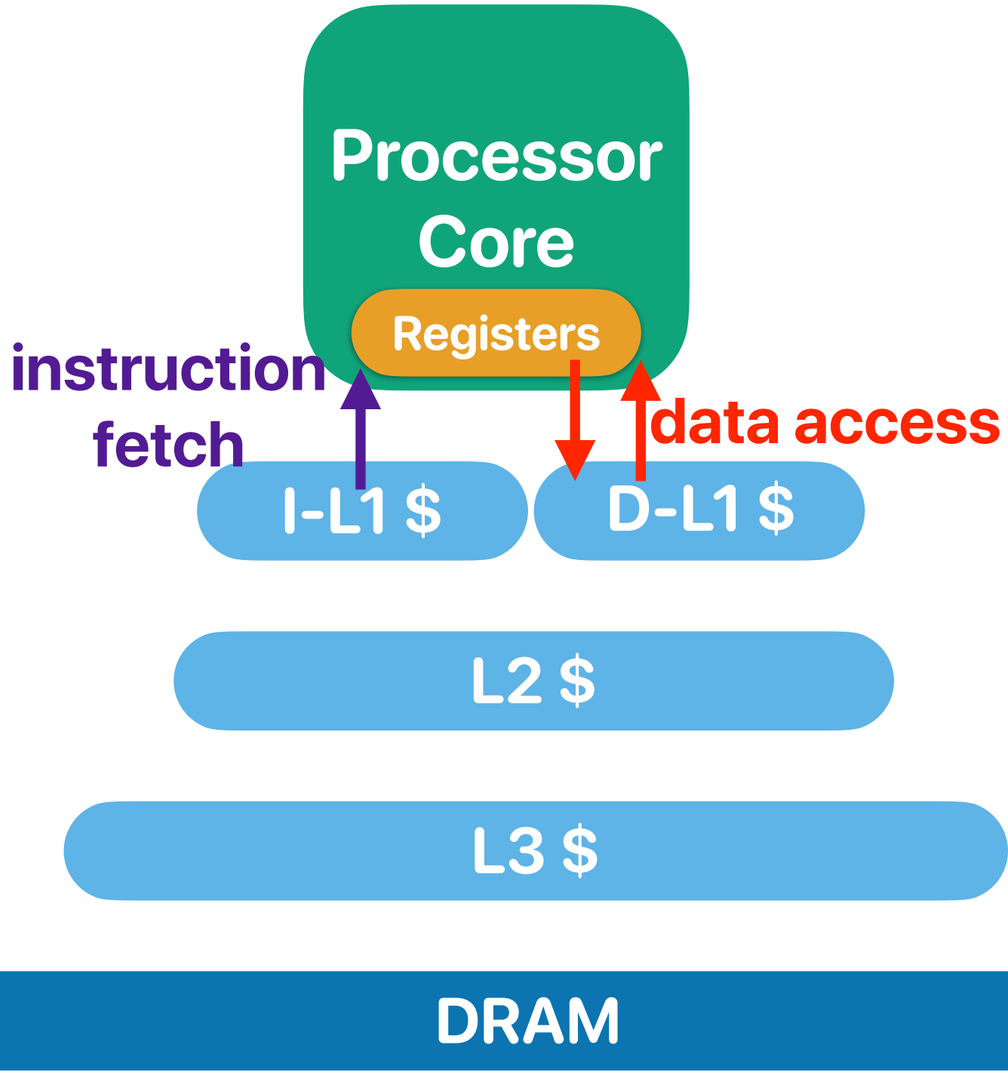


How to with the conflicts between MEM and IF?

- The memory unit can only accept/perform one request each cycle

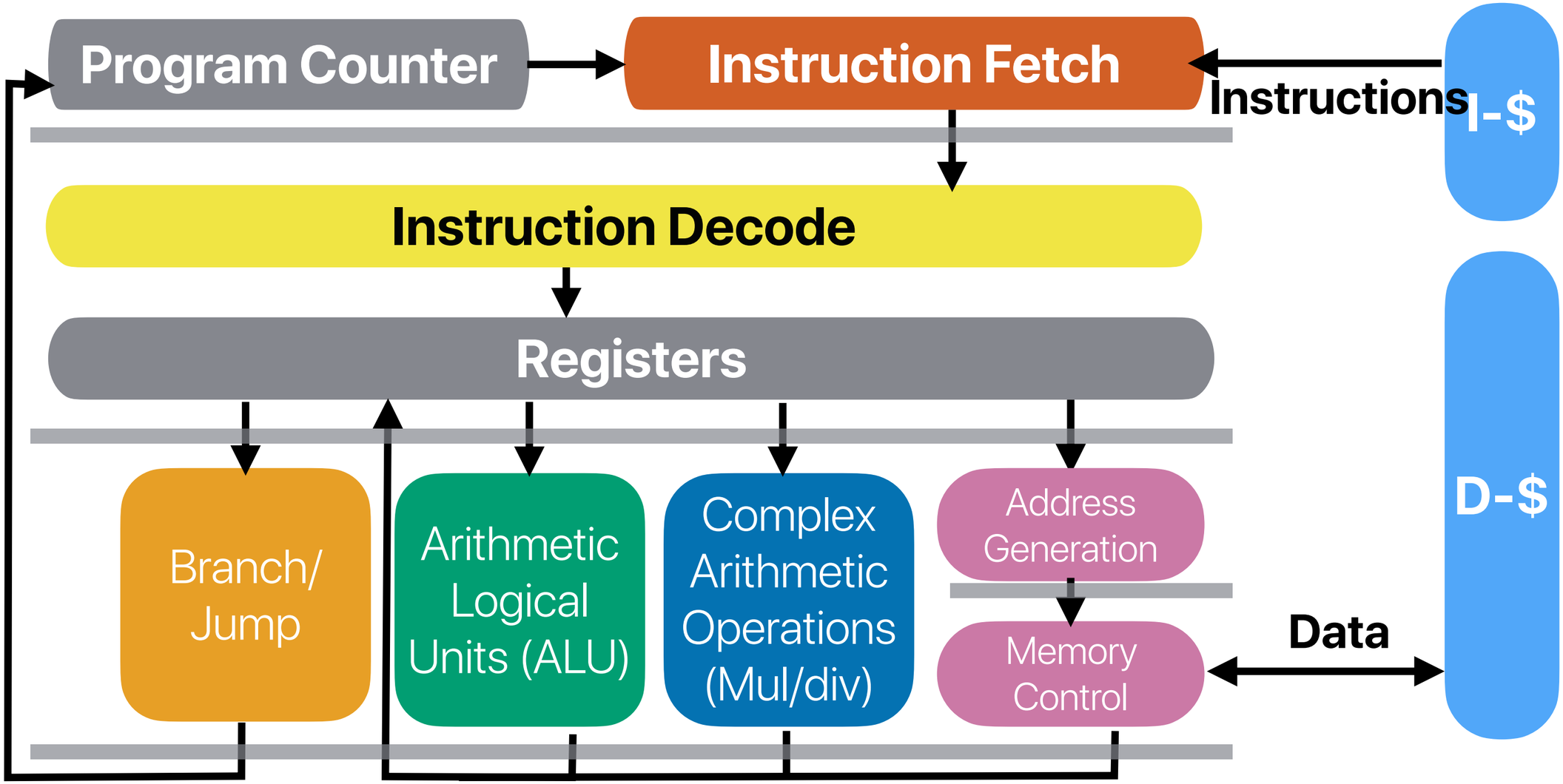
```
① xorl %eax, %eax
② movl (%rdi), %ecx
③ addl %ecx, %eax
④ addq $4, %rdi
⑤ cmpq %rdx, %rdi
```





"Split L1" cache!

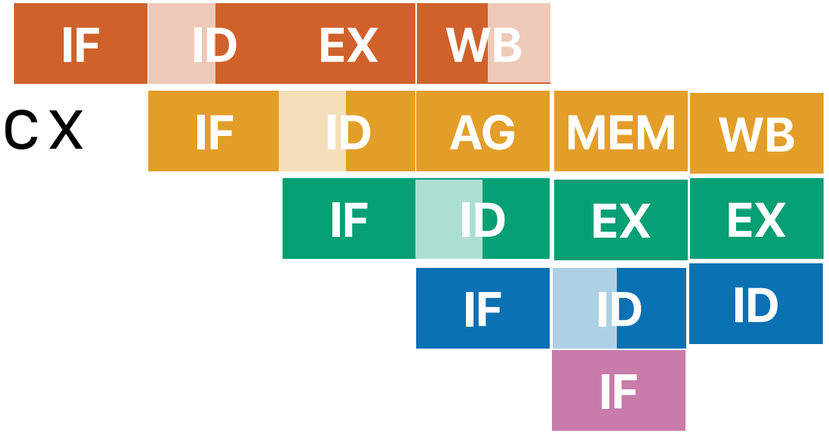
Split L1-$


Program Counter
Instruction Fetch
Instructions
I-$
Instruction Decode
Registers
Branch/ Jump
Arithmetic Logical Units (ALU)
Complex Arithmetic Operations (Mul/div)
Address Generation
Memory Control
Data
D-$

Both (2) and (3) want to “WB”

- The memory unit can only accept/perform one request each cycle

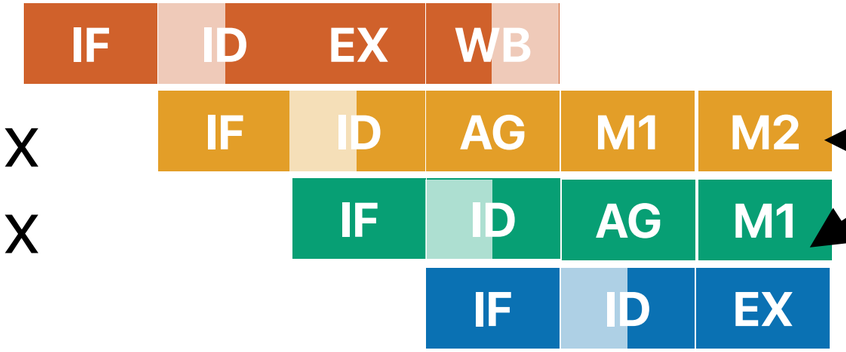
```
① xorl %eax, %eax
② movl (%rdi), %ecx
③ addl %ecx, %eax
④ addq $4, %rdi
⑤ cmpq %rdx, %rdi
```



(3) has to stall

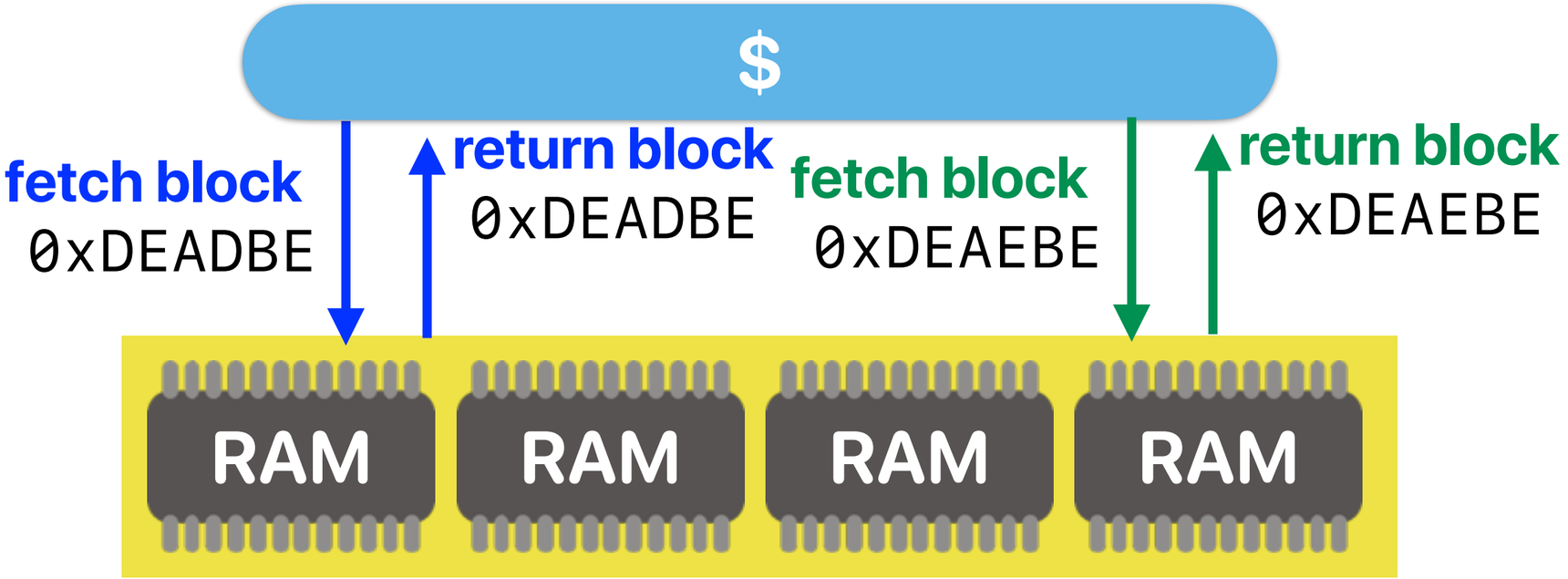
What if we've back-to-back memory instructions?

① `xorl %eax, %eax`
② `movl (%rdi), %rcx`
③ `movl (%rcx), %rax`
④ `addq $8, %rdi`

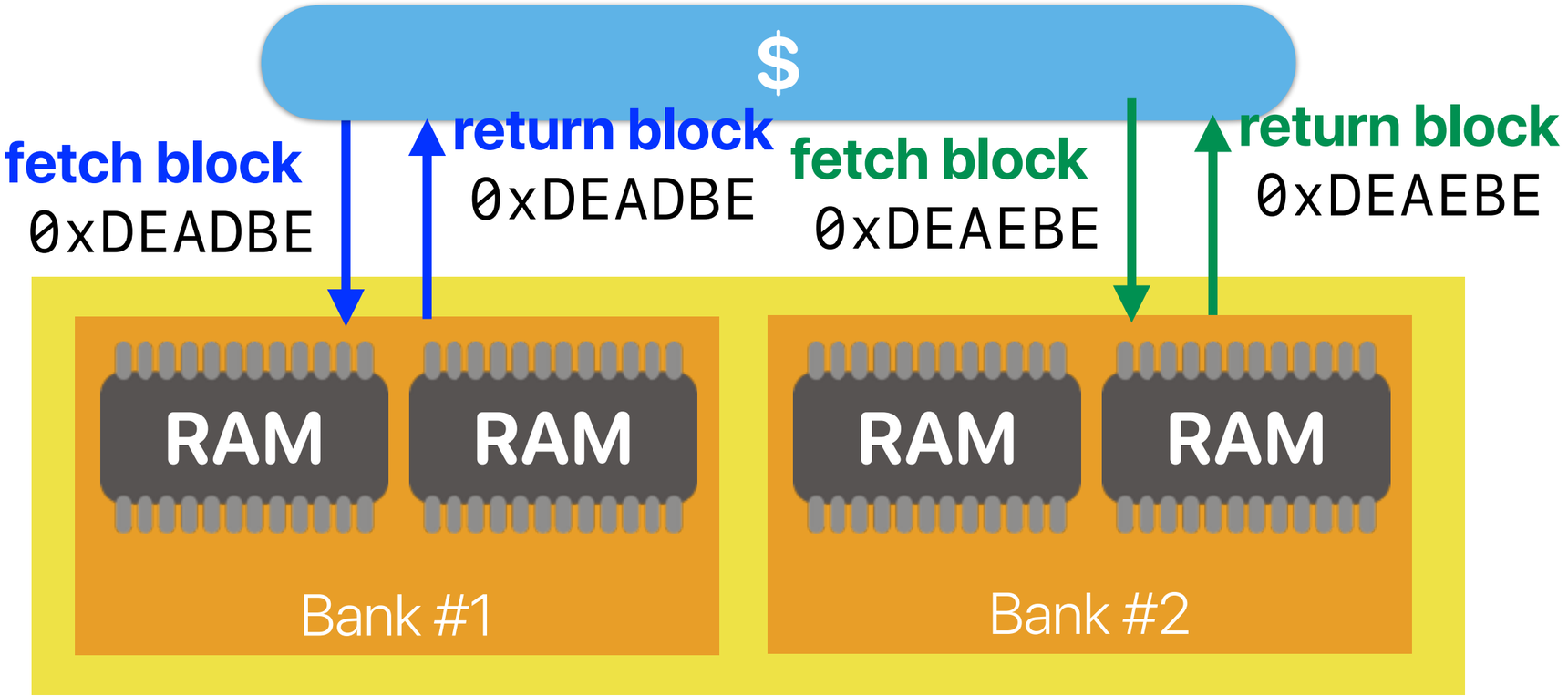


Both (2) and (3) are attempting use data memory unit and it's occupied by (2)

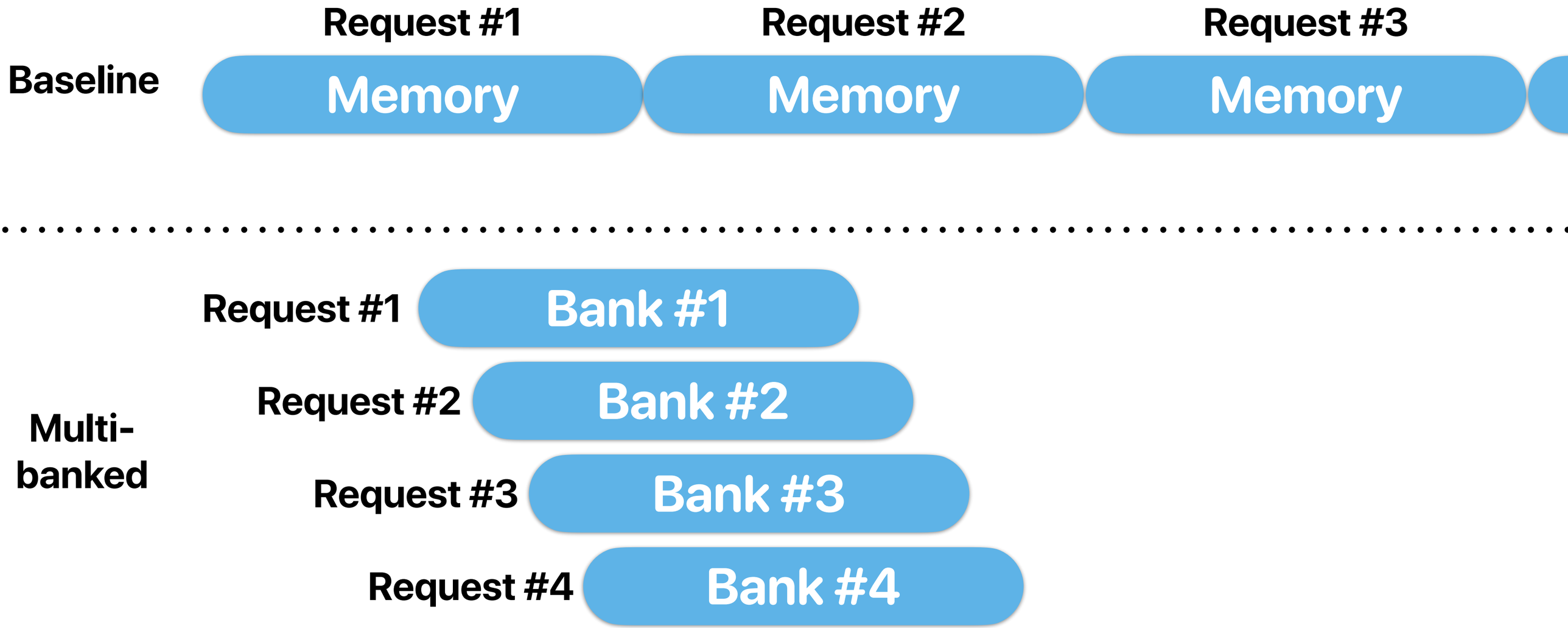
Blocking cache



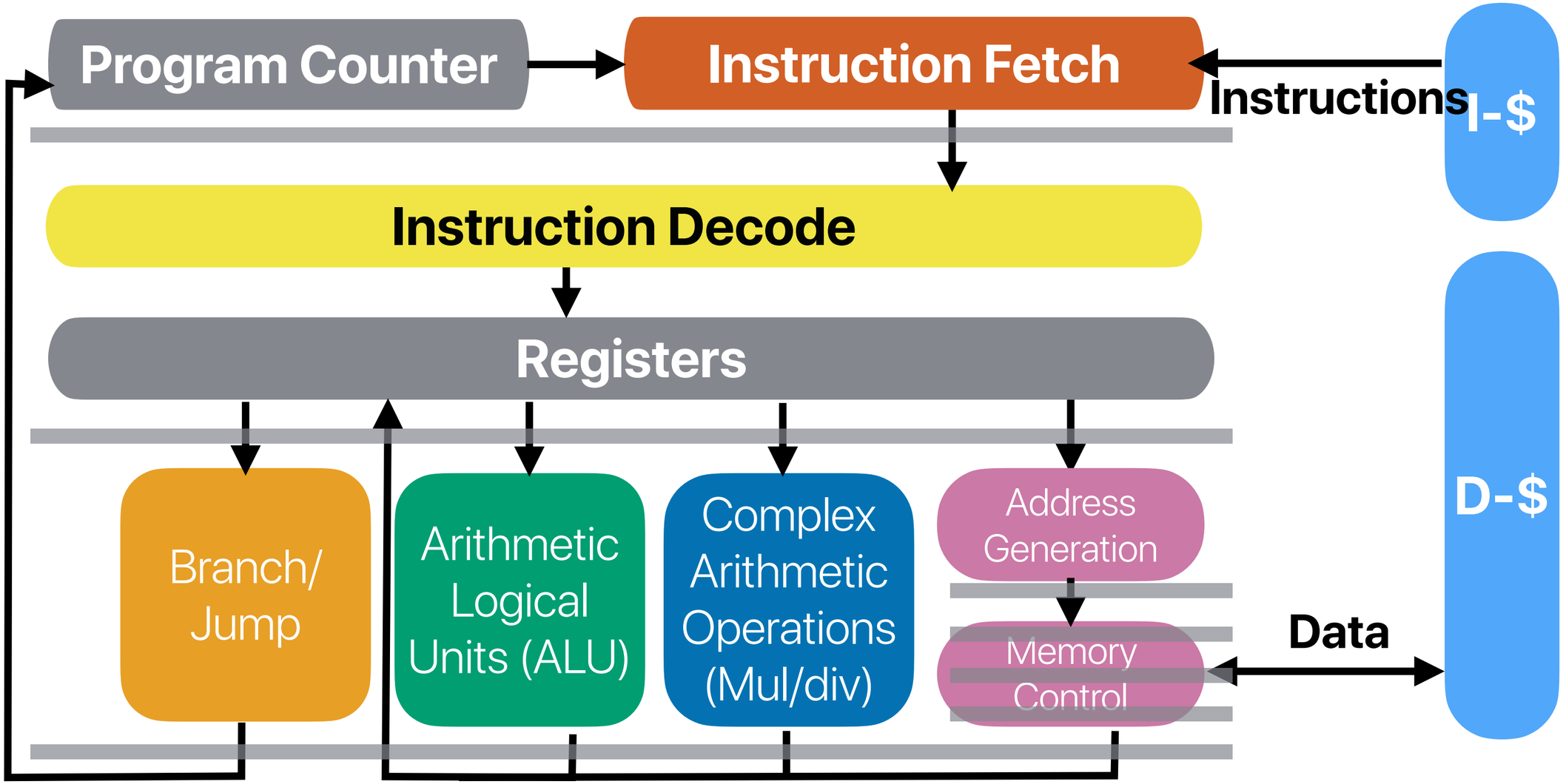
Multibanks & non-blocking caches



Pipelined access and multi-banked caches



Split L1-$ + Multibanked, non-blocking cache

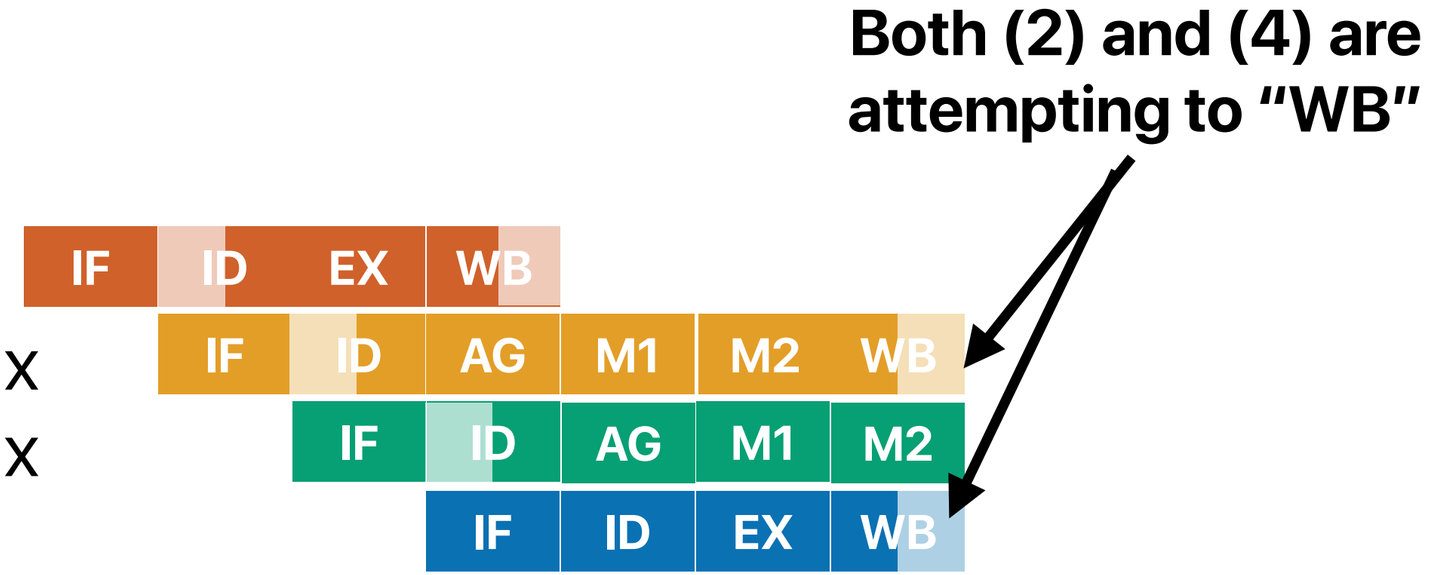


Takeaways: pipeline processors

- Pipelining helps to improve the throughput of processors
  - Allowing shorter cycle time as each cycle only make progress for part of an instruction
  - Different pipeline stages work on different instructions concurrently
  - Theoretical CPI remains the same as single-cycle design and the throughput/speedup is in proportion to the speedup of cycle time
- Pipeline hazards prevent us from reaching the theoretical CPI
  - Structural hazards
  - Control hazards
  - Data hazards
- The most efficient approach to address structural hazards is to make the hardware available to support concurrent execution
  - Register file
  - Split caches
  - Multibanked, non-blocking cache

Now, the memory is pipeline but takes more stages

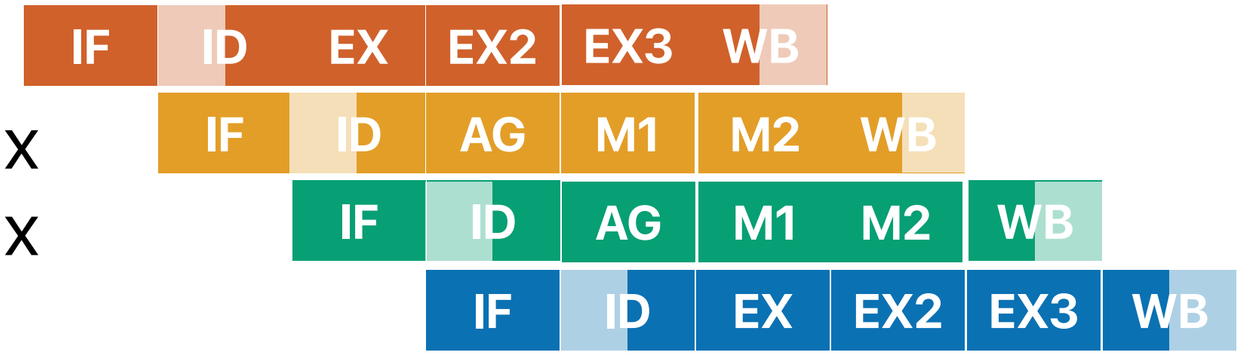
① `xorl %eax, %eax`
② `movl (%rdi), %ecx`
③ `movl (%rcx), %rax`
④ `addq $4, %rdi`



What if the memory instruction needs more time?

- Every instruction needs to go through exactly the same number of stages

① `xorl %eax, %eax`
② `movl (%rdi), %ecx`
③ `movl (%rcx), %rax`
④ `addq $4, %rdi`



Takeaways: pipeline processors

- Pipelining helps to improve the throughput of processors
  - Allowing shorter cycle time as each cycle only make progress for part of an instruction
  - Different pipeline stages work on different instructions concurrently
  - Theoretical CPI remains the same as single-cycle design and the throughput/speedup is in proportion to the speedup of cycle time
- Pipeline hazards prevent us from reaching the theoretical CPI
  - Structural hazards
  - Control hazards
  - Data hazards
- The most efficient approach to address structural hazards is to make the hardware available to support concurrent execution
  - Register file
  - Split caches
  - Multibanked, non-blocking cache
  - Force all instructions go through exactly the same number of cycles/stages before the last stage

Can we answer why only IC matters here?

- Considering the following two implementations that deliver the same result (all numbers are integers). Without turning on compiler optimizations, which one is faster and why?

A

```
for(i=0;i<1000000000;i++) {
    sum+=data[index];
    index =
(data[index]*15)%131072;
}
```

B

```
for(i=0;i<1000000000;i++) {
    sum+=data[index];
    index =
((data[index]<<4)-data[index])%131072;
}
```

A. Version A is faster as the CPI is lower
B. Version A is faster as the IC is lower
C. Version B is faster as the CPI is lower
D. Version B is faster as the IC is lower
E. They're about the same — sometimes A is faster, sometimes B is faster

Sample Midterm

Format

- 15 Multiple Choice Questions — 30%
- 3 Assignment Style Free Answer Questions — 70%
- Closed note, closed book, no outside materials

Programmer's impact

- By adding the "sort" in the following code snippet, what the programmer changes in the performance equation to achieve **better** performance?

```
std::sort(data, data + arraySize);

for (unsigned c = 0; c < arraySize*1000; ++c) {
        if (data[c%arraySize] >= INT_MAX/2)
            sum ++;
    }
}
```

A. CPI
B. IC
C. CT
D. IC & CPI
E. CPI & CT

How programming languages affect performance

- Performance equation consists of the following three factors
  ① IC
  ② CPI
  ③ CT

  How many can the **programming language** affect?

  A. 0
  B. 1
  C. 2
  D. 3

Demo — programmer & performance

A

```
for(i = 0; i < ARRAY_SIZE; i++)
{
  for(j = 0; j < ARRAY_SIZE; j++)
  {
    c[i][j] = a[i][j]+b[i][j];
  }
}
```

B

```
for(j = 0; j < ARRAY_SIZE; j++)
{
  for(i = 0; i < ARRAY_SIZE; i++)
  {
    c[i][j] = a[i][j]+b[i][j];
  }
}
```

- How many of the following make(s) the performance of A better than B?
  - ① IC
  - ② CPI
  - ③ CT
  - A. 0
  - B. 1
  - C. 2
  - D. 3

How compilers affect performance

- If we apply compiler optimizations for both code snippets **A** and **B**, how many of the following can we expect?
  - ① Compiler optimizations can reduce IC for both
  - ② Compiler optimizations can make the CPI lower for both
  - ③ Compiler optimizations can make the ET lower for both
  - ④ Compiler optimizations can transform code B into code A
  - A. 0
  - B. 1
  - C. 2
  - D. 3
  - E. 4

A

```
for(i = 0; i < ARRAY_SIZE; i++)
{
  for(j = 0; j < ARRAY_SIZE; j++)
  {
    c[i][j] = a[i][j]+b[i][j];
  }
}
```

B

```
for(j = 0; j < ARRAY_SIZE; j++)
{
  for(i = 0; i < ARRAY_SIZE; i++)
  {
    c[i][j] = a[i][j]+b[i][j];
  }
}
```

Amdahl's Law on Multicore Architectures

- Regarding Amdahl's Law on multicore architectures, how many of the following statements is/are correct?
  - ① If we have unlimited parallelism, the performance of each parallel piece does not matter as long as the performance slowdown in each piece is bounded
  - ② With unlimited amount of parallel hardware units, single-core performance does not matter anymore
  - ③ With unlimited amount of parallel hardware units, the maximum speedup will be bounded by the fraction of parallel parts
  - ④ With unlimited amount of parallel hardware units, the effect of scheduling and data exchange overhead is minor

  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

How reflective is FLOPS?

- If you're given the FLOPS of an underlying GPU, how many situations below can the FLOPS be representative to the real performance?
  - ① The FLOPS remains the same on the same GPU even if we change the data size
  - ② The FLOPS remains the same on the same GPU even if we change the data type to double
  - ③ The FLOPS remains the same on the same GPU if we change the algorithm implementation
  - ④ The ratio of FLOPS on two different GPUs reflects the ratio execution on these two GPUs when executing floating point applications

  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

Data locality

- Which description about locality of arrays `matrix` and `vector` in the following code is the **most accurate**?

```
for(uint64_t i = 0; i < m; i++) {
    result = 0;
    for(uint64_t j = 0; j < n; j++) {
        result += matrix[i][j]*vector[j];
    }
    output[i] = result;
}
```

A. Access of `matrix` has temporal locality, `vector` has spatial locality
B. Both `matrix` and `vector` have temporal locality, and `vector` also has spatial locality
C. Access of `matrix` has spatial locality, `vector` has temporal locality
D. Both `matrix` and `vector` have spatial locality and temporal locality
E. Both `matrix` and `vector` have spatial locality, and `vector` also has temporal locality

intel Core i7

- L1 data (D-L1) cache configuration of Core i7
  - Size 48KB, 12-way set associativity, 64B block
  - Assume 64-bit memory address
  - Which of the following is NOT correct?

    A. Tag is 52 bits

    B. Index is 6 bits

    C. Offset is 6 bits

    D. The cache has 128 sets

intel Core i7

- D-L1 Cache configuration of intel Core i7
  - Size 48KB, 12-way set associativity, 64B block, LRU policy, write-allocate, write-back, and assuming 64-bit address.

```
double a[16384], b[16384], c[16384], d[16384], e[16384];
/* a = 0x10000, b = 0x20000, c = 0x30000, d = 0x40000, e = 0x50000 */
for(i = 0; i < 512; i++) {
    e[i] = (a[i] * b[i] + c[i])/d[i];
    //load a[i], b[i], c[i], d[i] and then store to e[i]
}
```

How many of the cache misses are **compulsory** misses?

A. 12.5%
B. 66.67%
C. 68.75%
D. 87.5%
E. 100%

What if we change the processor?

- If we have an intel processor with a 32KB, 8-way, 64B-blocked L1 cache, which version of code performs better?
  A. Version A, because the code incurs fewer cache misses
  B. Version B, because the code incurs fewer cache misses
  C. Version A, because the code incurs fewer memory references
  D. Version B, because the code incurs fewer memory references
  E. They are about the same

A

```
double a[8192], b[8192], c[8192], \
       d[8192], e[8192];
for(i = 0; i < 512; i++)
    e[i] = a[i] * b[i] + c[i];
for(i = 0; i < 512; i++)
    e[i] /= d[i];
```

B

```
double a[8192], b[8192], c[8192], \
       d[8192], e[8192];
for(i = 0; i < 512; i++) {
    e[i] = (a[i] * b[i] + c[i])/d[i];
}
```

What kind(s) of misses can tiling algorithm remove?

- Comparing the naive algorithm and tiling algorithm on matrix multiplication, what kind of misses does tiling algorithm help to remove? (assuming an intel Core i7)

Naive

```
for(i = 0; i < ARRAY_SIZE; i++) {
  for(j = 0; j < ARRAY_SIZE; j++) {
    for(k = 0; k < ARRAY_SIZE; k++) {
      c[i][j] += a[i][k]*b[k][j];
    }
  }
}
```

Block

```
for(i = 0; i < ARRAY_SIZE; i+=(ARRAY_SIZE/n)) {
  for(j = 0; j < ARRAY_SIZE; j+=(ARRAY_SIZE/n)) {
    for(k = 0; k < ARRAY_SIZE; k+=(ARRAY_SIZE/n)) {
      for(ii = i; ii < i+(ARRAY_SIZE/n); ii++)
        for(jj = j; jj < j+(ARRAY_SIZE/n); jj++)
          for(kk = k; kk < k+(ARRAY_SIZE/n); kk++)
            c[ii][jj] += a[ii][kk]*b[kk][jj];
    }
  }
}
```

A. Compulsory miss
B. Capacity miss
C. Conflict miss
D. Capacity & conflict miss
E. Compulsory & conflict miss

What kind(s) of misses can matrix transpose remove?

- By transposing a matrix, the performance of matrix multiplication can be further improved. What kind(s) of cache misses does matrix transpose help to remove?

Block

```
for(i = 0; i < ARRAY_SIZE; i+=(ARRAY_SIZE/n)) {
  for(j = 0; j < ARRAY_SIZE; j+=(ARRAY_SIZE/n)) {
    for(k = 0; k < ARRAY_SIZE; k+=(ARRAY_SIZE/n)) {
      for(ii = i; ii < i+(ARRAY_SIZE/n); ii++)
        for(jj = j; jj < j+(ARRAY_SIZE/n); jj++)
          for(kk = k; kk < k+(ARRAY_SIZE/n); kk++)
            c[ii][jj] += a[ii][kk]*b[kk][jj];
    }
  }
}
```

A. Compulsory miss
B. Capacity miss
C. Conflict miss
D. Capacity & conflict miss
E. Compulsory & conflict miss

Block + Transpose

```
// Transpose matrix b into b_t
for(i = 0; i < ARRAY_SIZE; i+=(ARRAY_SIZE/n)) {
  for(j = 0; j < ARRAY_SIZE; j+=(ARRAY_SIZE/n)) {
      b_t[i][j] += b[j][i];
  }
}


for(i = 0; i < ARRAY_SIZE; i+=(ARRAY_SIZE/n)) {
  for(j = 0; j < ARRAY_SIZE; j+=(ARRAY_SIZE/n)) {
    for(k = 0; k < ARRAY_SIZE; k+=(ARRAY_SIZE/n)) {
        for(ii = i; ii < i+(ARRAY_SIZE/n); ii++)
          for(jj = j; jj < j+(ARRAY_SIZE/n); jj++)
            for(kk = k; kk < k+(ARRAY_SIZE/n); kk++
              // Compute on b_t
              c[ii][jj] += a[ii][kk]*b_t[jj][kk];
    }
  }
}
```

When we have virtual memory…

- If an x86 processor supports virtual memory through the basic format of the page table as shown in the previous slide, how many memory accesses can a **`mov`** instruction that access data memory once incur?
  A. 2
  B. 4
  C. 6
  D. 8
  E. 10

Why is A is faster?

A

```
void regswap(int* a, int* b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}
```

B

```
void xorswap(int* a, int* b) {
    *a ^= *b;
    *b ^= *a;
    *a ^= *b;
}
```

- What's the main reason why version B cannot outperform version A on modern processors?
  A. Control hazards
  B. Data hazards
  C. Structural hazards

Which of the following schemes can help Tegra X1?

- How many of the following schemes mentioned in “improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers” would help NVIDIA Tegra X1’s cache performance for the code below?
  - ① Missing cache
  - ② Victim cache
  - ③ Prefetch
  - ④ Stream buffer
  - A. 0
  - B. 1
  - C. 2
  - D. 3
  - E. 4

```
double a[8192], b[8192], c[8192], d[8192], e[8192];
/* a = 0x10000, b = 0x20000, c = 0x30000, d = 0x40000, e = 0x50000 */
for(i = 0; i < 8192; i++) {
    e[i] = (a[i] * b[i] + c[i])/d[i];
    //load a[i], b[i], c[i], d[i] and then store to e[i]
```

Remind yourself

- What are the limitations of compiler optimizations? Can you list two?
- Please define Amdahl's Law and explain each term in it
- Please define the CPU performance equation and explain each term.
- Can you list two things affecting each term in the performance equation?
- What's the difference between latency and throughput? When should you use latency or throughput to judge performance?
- What's "benchmark" suite? Why is it important?
- Why TFLOPS or inferences per second is not a good metrics?

Performance equation

- Consider the following c code snippet and x86 instructions implement the code snippet

C	x86

`for(i = 0; i < count; i++) {`
`  s += a[i];`
`}`
	`.L3:`
	`movslq (%rdi), %rdx`
	`addq $4, %rdi`
	`addq %rdx, %tax`
	`cmpq %rcx, %rdi`
	`jne .L3`

If (1) count is set to 1,000,000,000, (2) a memory instruction takes 4 cycles, (3) a branch/jump instruction takes 3 cycles, (4) other instructions takes 1 cycle on average, and (5) the processor runs at 4 GHz, how much time is it take to finish executing the code snippet?

Speedup of Y over X

- Consider the same program on the following two machines, X and Y. By how much Y is faster than X?

	Clock Rate	Instructions	Percentage of Type-A	CPI of Type-A	Percentage of Type-B	CPI of Type-B	Percentage of Type-C	CPI of Type-C
Machine X	3 GHz	5000000000	20%	8	20%	4	60%	1
Machine Y	5 GHz	5000000000	20%	13	20%	4	60%	1

Cache performance on the following code

- D-L1 Cache configuration of intel Core i7 processor
  - Size 48KB, 12-way set associativity, 64B block, LRU policy, write-allocate, write-back, and assuming 64-bit address.

```
double data[16384]; /* &data[0] = 0x20000 */
uint64_t sum = 0;
for(uint i = 0; i < arg1; i++) {
    for(uint x = 0; x < size; x+=arg1) {
      sum += data[x];
    }
}
return sum;
```

What's the data cache miss rate for this code when **arg1=1** and **size = 16384**?

Cache performance on the following code

- D-L1 Cache configuration of intel Core i7 processor
  - Size 48KB, 12-way set associativity, 64B block, LRU policy, write-allocate, write-back, and assuming 64-bit address.

```
double data[16384]; /* &data[0] = 0x20000 */
uint64_t sum = 0;
for(uint i = 0; i < arg1; i++) {
    for(uint x = 0; x < size; x+=arg1) {
      sum += data[x];
    }
}
return sum;
```

What's the data cache miss rate for this code when **arg1=16** and **size = 16384**?

Practicing Amdahl's Law

- Final Fantasy XV spends lots of time loading a map — within which period that 95% of the time on the accessing the H.D.D., the rest in the operating system, file system and the I/O protocol. If we replace the H.D.D. with a flash drive, which provides 100x faster access time. By how much can we speed up the map loading process?

Practicing Amdahl's Law (2)

- Final Fantasy XV spends lots of time loading a map — within which period that 95% of the time on the accessing the H.D.D., the rest in the operating system, file system and the I/O protocol. If we replace the H.D.D. with a flash drive, which provides 100x faster access time and a better processor to accelerate the software overhead by 2x. By how much can we speed up the map loading process?

CEN TECH TALK

Join Us to Learn How to Shape Tech Careers:

Insights and Guidance from Industry Experts

Ethan Law

Senior Firmware Engineer at Foresight Sports

Anthony Mucciolo

Systems Architect at Foresight Sports

FORESIGHT SPORTS

Friday, November 1, 2024
1:00 pm - 2:00 pm

Bourns A265

Food will be provided!

Announcement

- Assignment #3 due **this evening**
- Programming Assignment #2 **due 11/7**
- Reading Quiz #6 due **next Thursday**
- **All questions in the sample midterm are from the slides & your assignments — please don't ask for solutions on those**
- Midterm **next Tuesday**
  - 80 minutes, in-person only
  - Closed book, closed note, no laptop, no mobile phones (including the calculator app)
  - You may use a calculator


Computer
Science &
Engineering
203
つづく