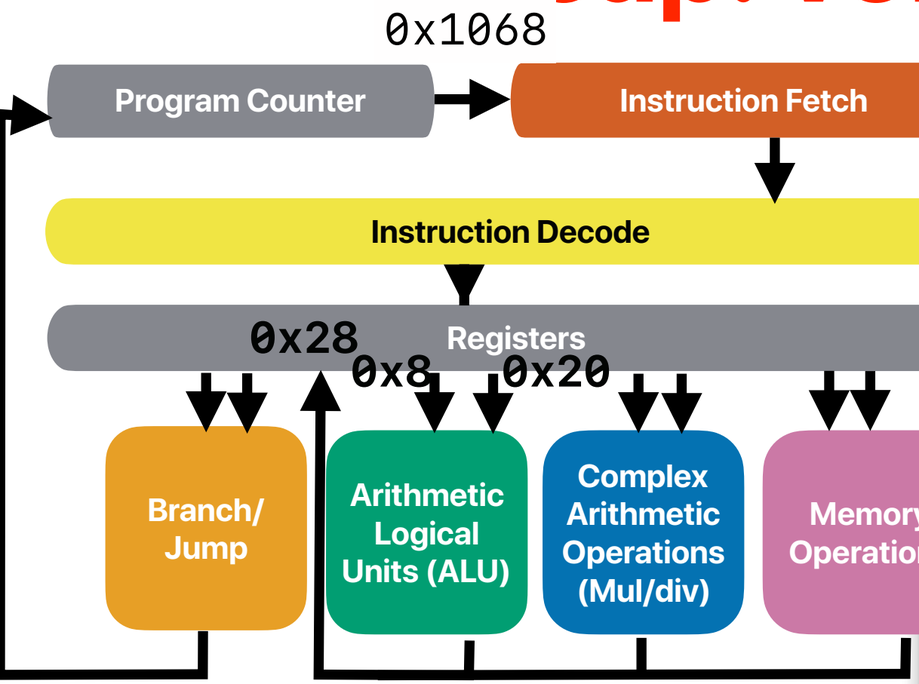
Modern Processor Design (I): in the pipeline

Hung-Wei Tseng

Recap: von Neumann architecture

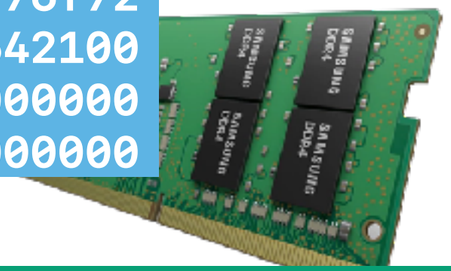


Processor

4883ec08

sub $0x8,%rsp

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



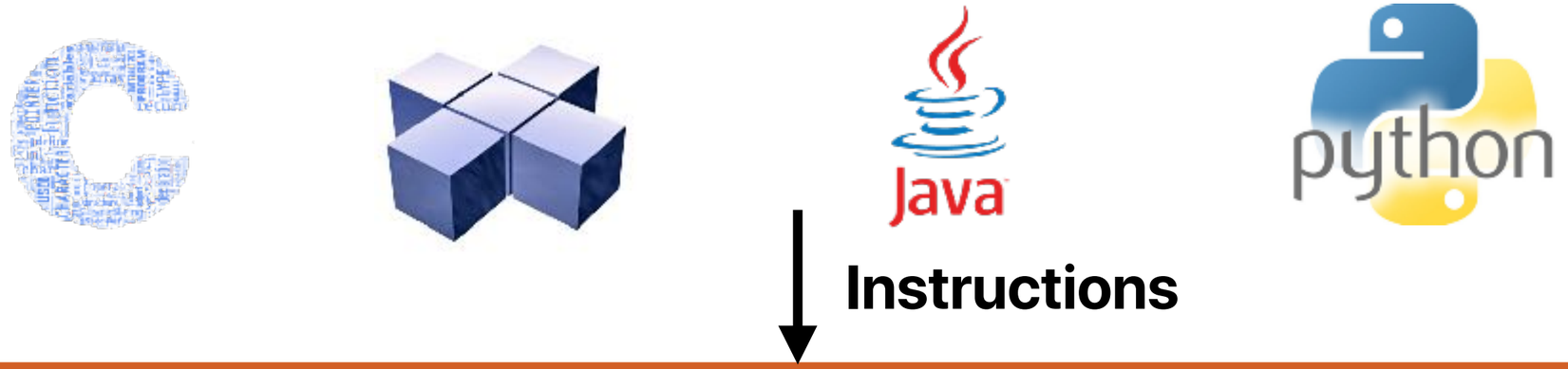
Memory

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

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

Storage

Recap: Microprocessor — a collection of functional units



Broadcom

Engaged Employer

Follow | Add an interview

Overview | 6.6K Reviews | 213 Jobs | 15K Salaries | 44 @Broadcom | 862 Interviews

Interview Question

Firmware Engineer Interview

How to swap the values in 2 variables without using a temporary variable.

Answer

SAP

Engag

Overviev

Interview Question

Support Engineer Interview

How can you swap two variables without using a third temporal variable?

Answer

Qualcomm

Engaged Employer

Add an interview

Overview | 12K Reviews | 584 Jobs | 245K Salaries | 112 @Qualcomm | 3.7K Interviews

Got a burning question about Qualcomm? Just ask!

On Glassdoor, you can share insights and advice anonymously with Qualcomm employees and get real answers from people on the inside.

Ask a question

DSP Firmware Engineer (Wireless Modem) Interview

Jan 6, 2010

Anonymous Interview Candidate

Santa Clara, CA

✗ No offer ✓ Positive experience ✓ Easy interview

Application

I applied online. The process took 1 day.I interviewed at Qualcomm (Santa Clara, CA) in Nov 2009

Interview

The HR took 2 weeks to schedule the first phone interview. The interviewer asked about my background, academic projects and past internships. Then asked basic technical questions about computer architecture, basic DSP, and few C questions. I fumbled and did not get any further interviews.

Interview questions [1]

Question 1

Swap two numbers without using a temporary variable

3 Answers →

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/

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: Leveraging more “bit-wise” operations in C code will make the program significantly faster

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

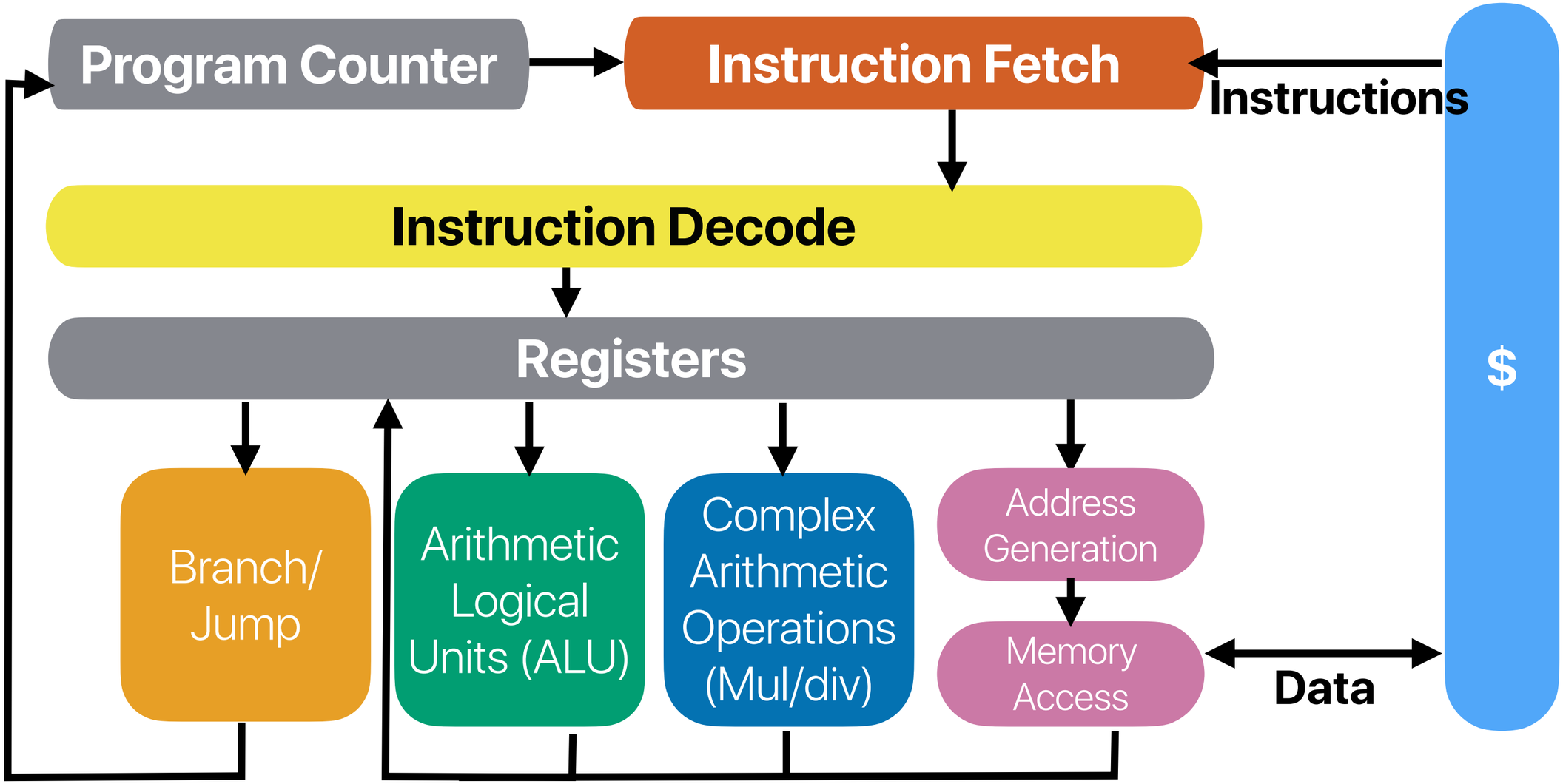
- Recap: the concept of a processor
- Pipelined Processor
- Pipeline Hazards
  - Structural Hazards
  - Control Hazards
  - Data Hazards

Basic Processor Design

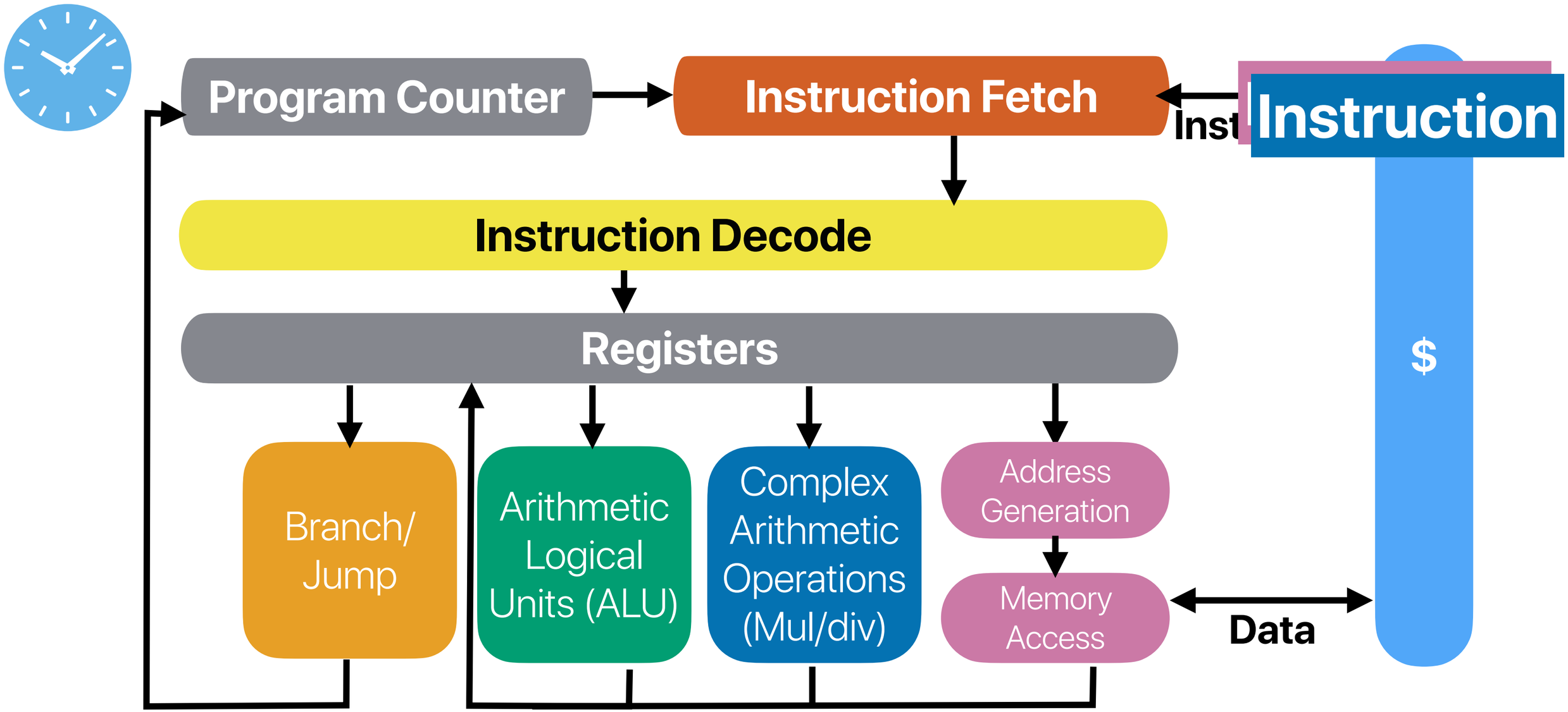
The “life” of an instruction

- 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

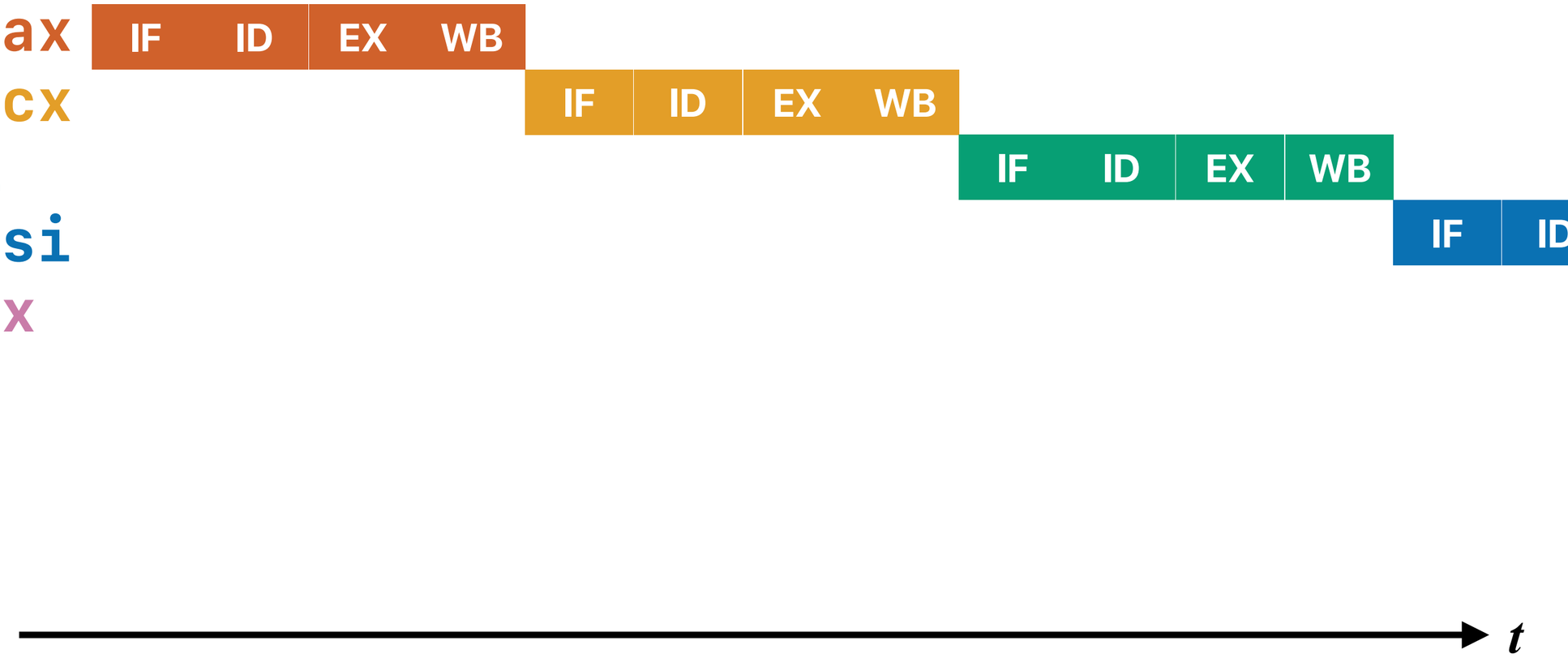


If we want to perform one instruction each cycle…



Simple implementation w/o branch

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

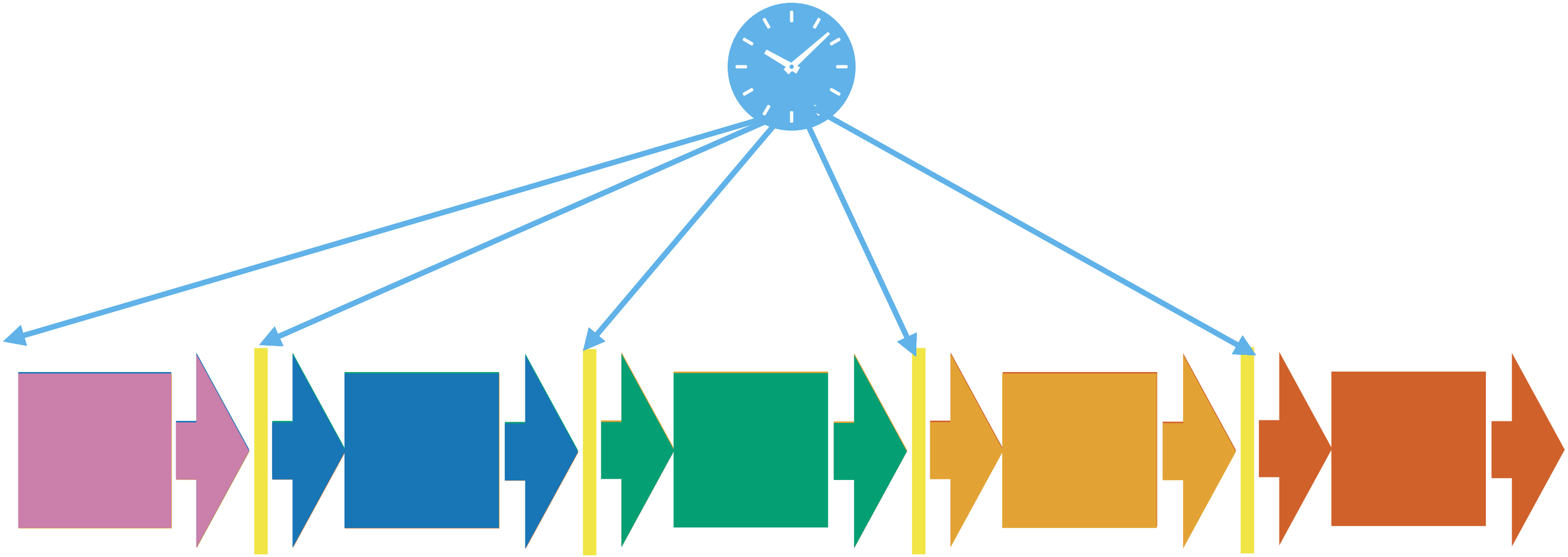


Pipelining

Pipelining

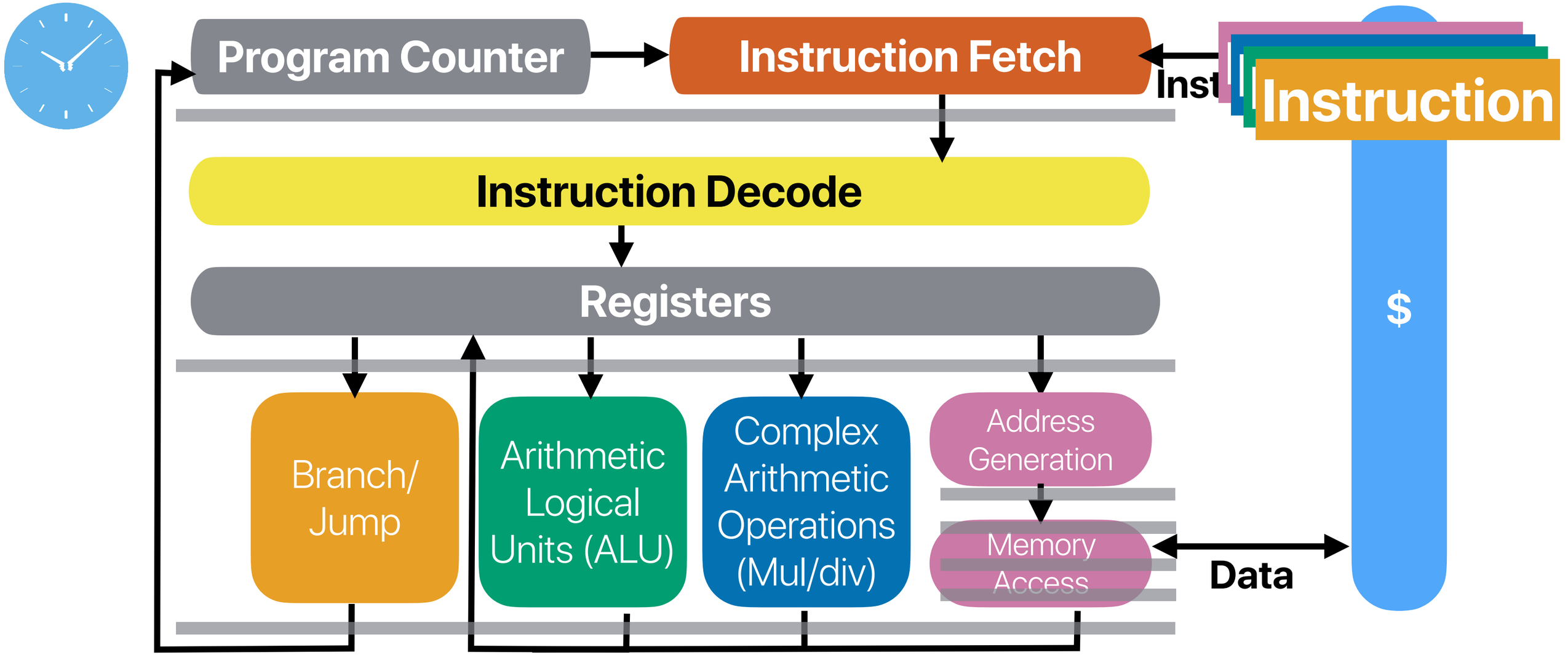
- Different parts of the processor works on different instructions simultaneously
- A processor is now working on multiple instructions from the same program (though on different stages) simultaneously.
  - ILP: **Instruction-level parallelism**
- A **clock** signal controls and synchronize the beginning 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



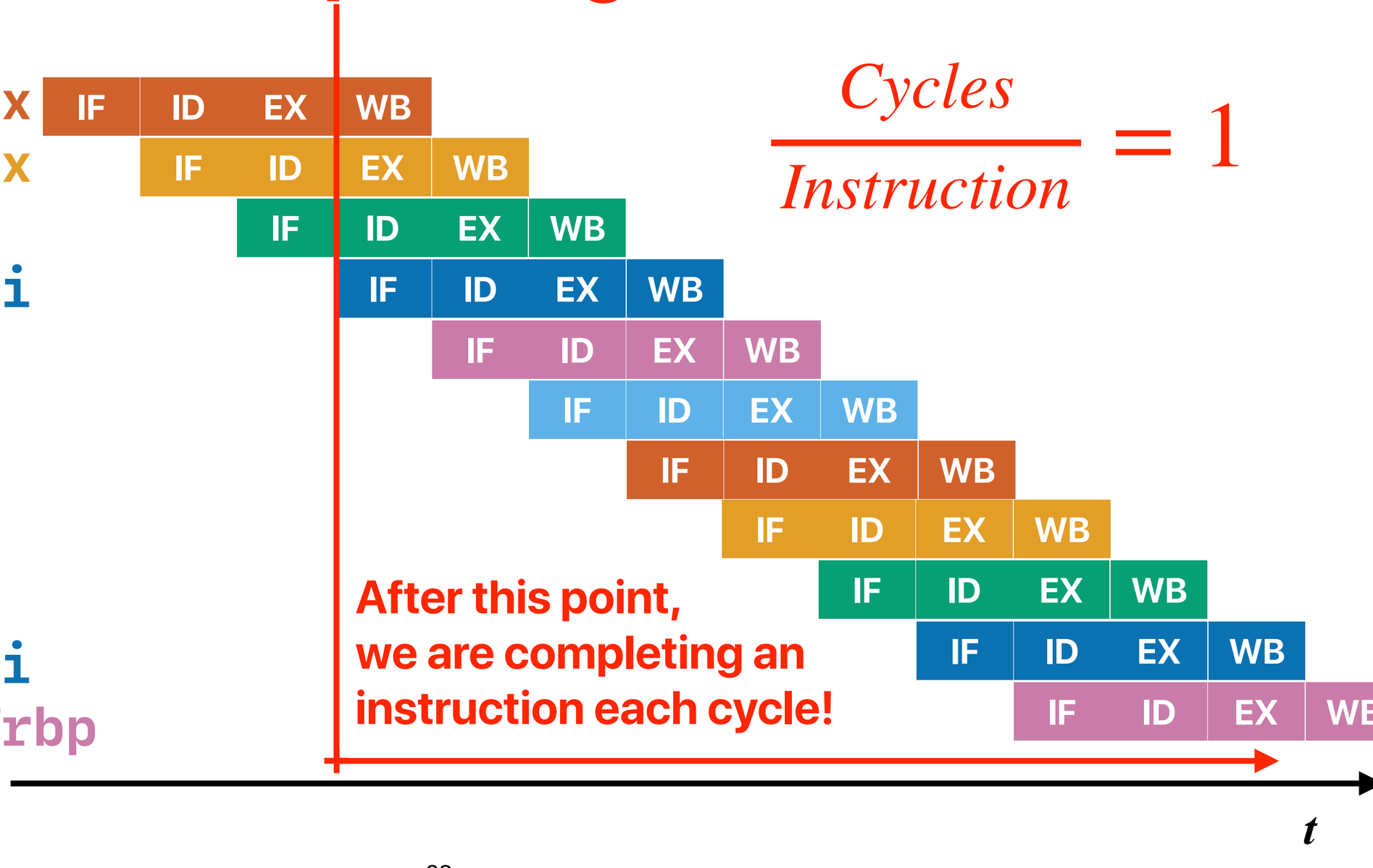
Pipelining

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
```



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

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?

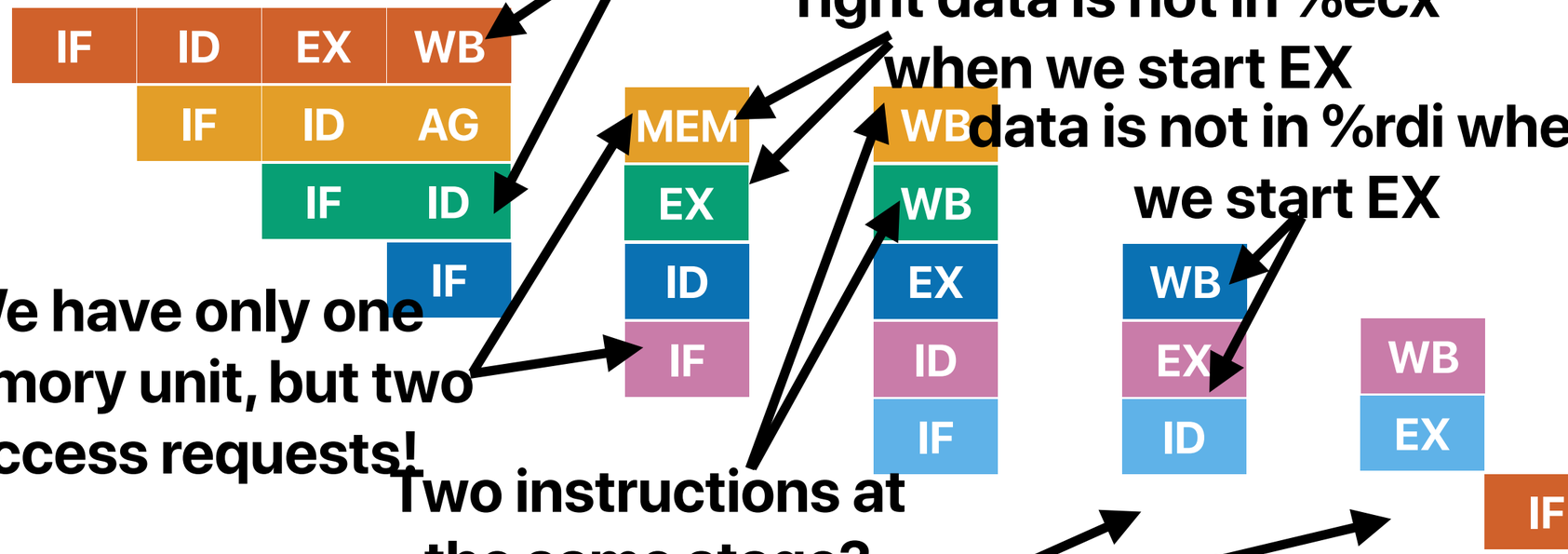
```
①       xorl    %eax, %eax
② L3:   movl    (%rdi), %ecx
③       addl    %ecx, %eax
④       addq    $4, %rdi
⑤       cmpq    %rdx, %rdi
⑥       jne     .L3
⑦       ret
```

```
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

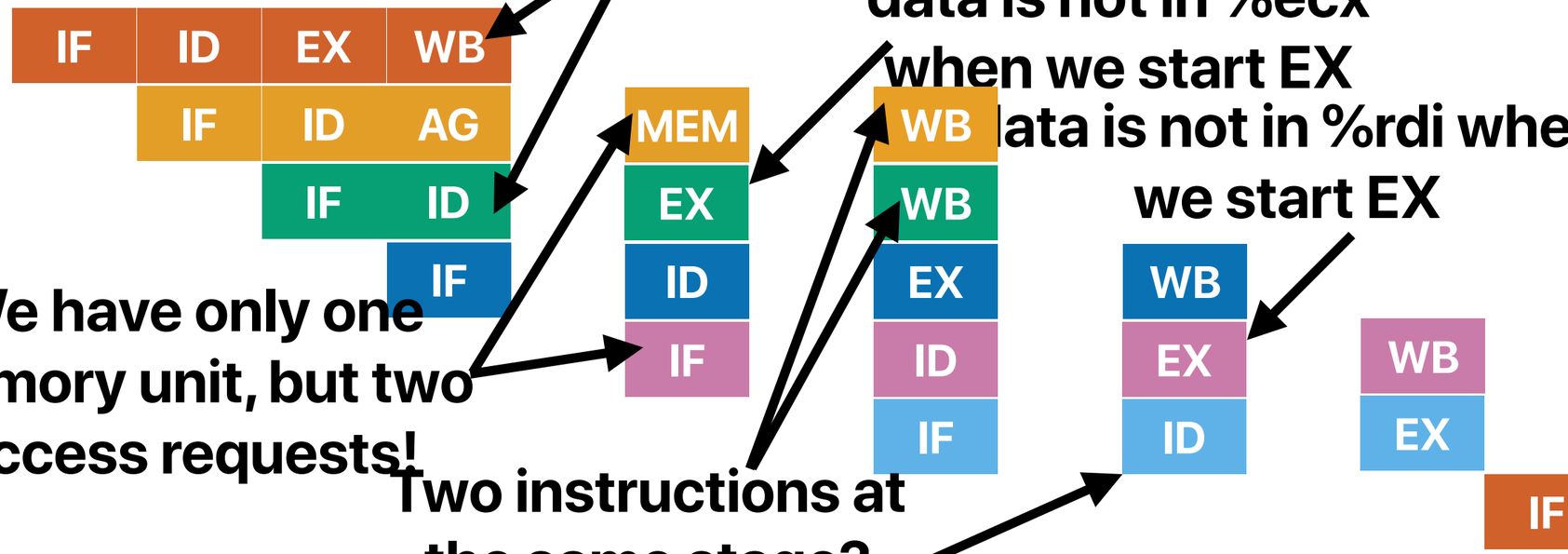
- Structural hazards — resource conflicts cannot support simultaneous execution of instructions in the pipeline
- Control hazards — the PC can be changed by an instruction in the pipeline
- Data hazards — an instruction depending on a the result that's not yet generated or propagated when the instruction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

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

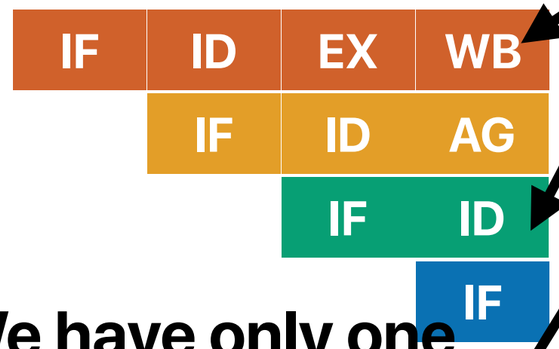


Pipelining

```
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



Structural Hazard

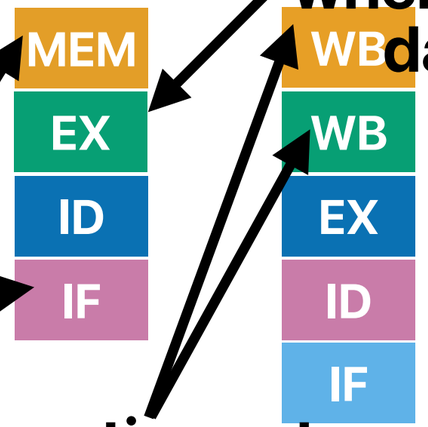
We have only one memory unit, but two access requests!

Structural Hazard

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

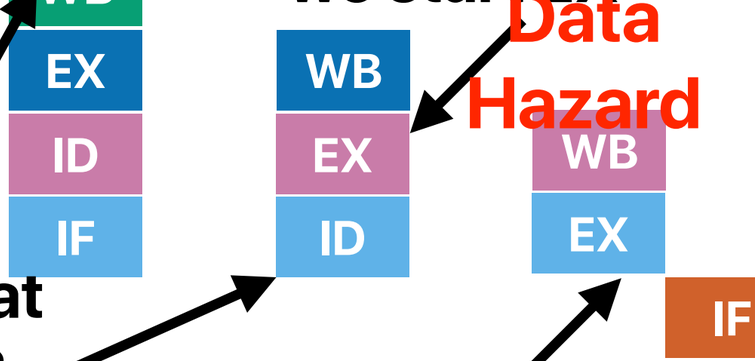
Data Hazard

data is not in %ecx when we start EX



data is not in %rdi when we start EX

Data Hazard



Two instructions at the same stage?

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

Control Hazard

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

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

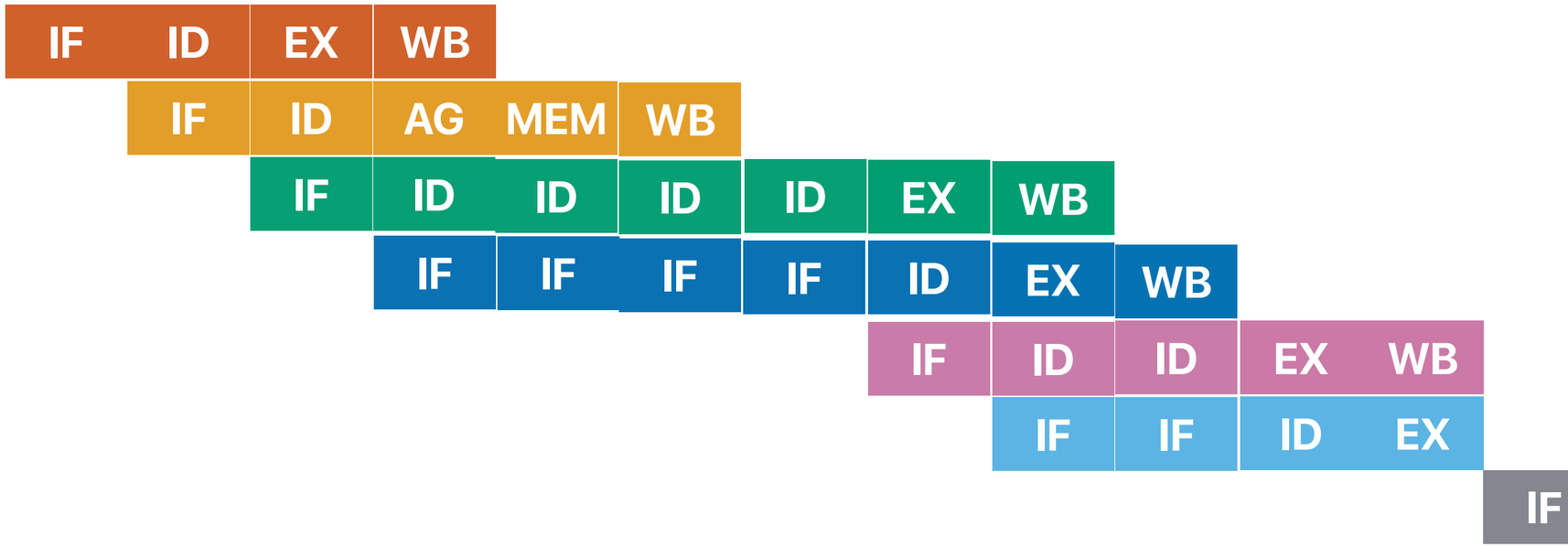
- Pipelining helps to improve the throughput of processors
  - Allowing shorter cycle time as each cycle only make progress for part of an instruction
  - Different pipeline stages work on different instructions concurrently
  - Theoretical CPI remains the same as single-cycle design and the throughput/speedup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



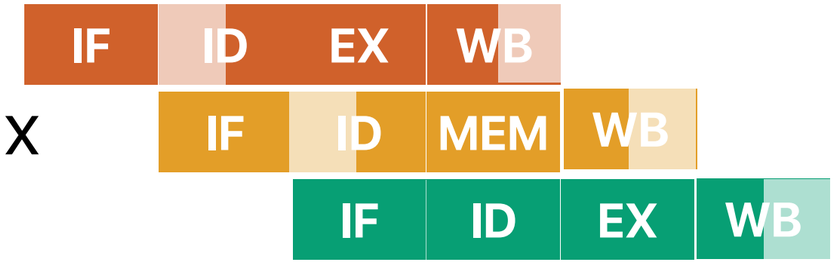
Slow! — 4 additional cycles

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!

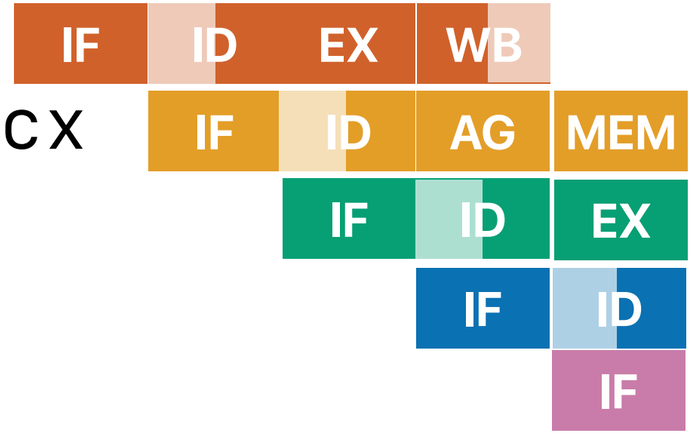
```
① xorl %eax, %eax
② movl (%rdi), %ecx
③ addl %ecx, %eax
```

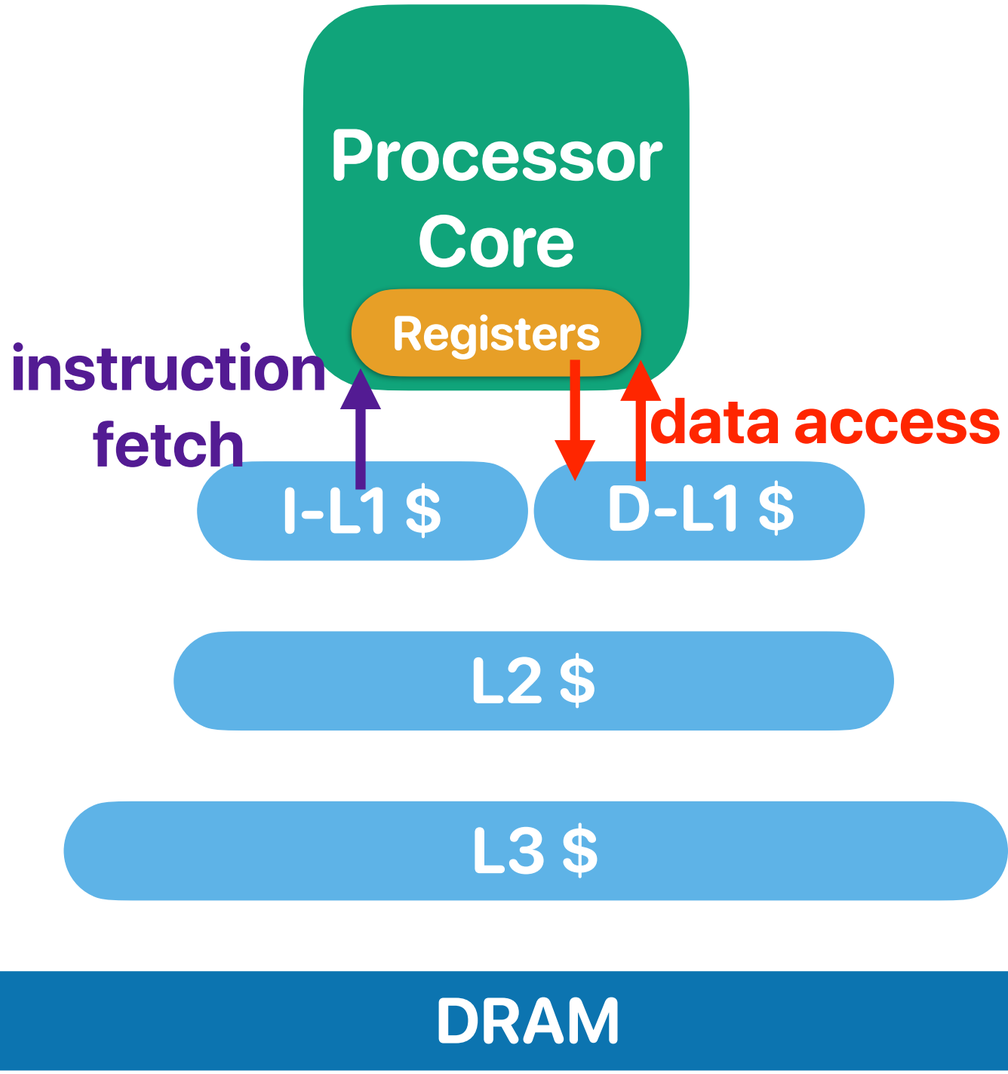


How to with the conflicts between MEM and IF?

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

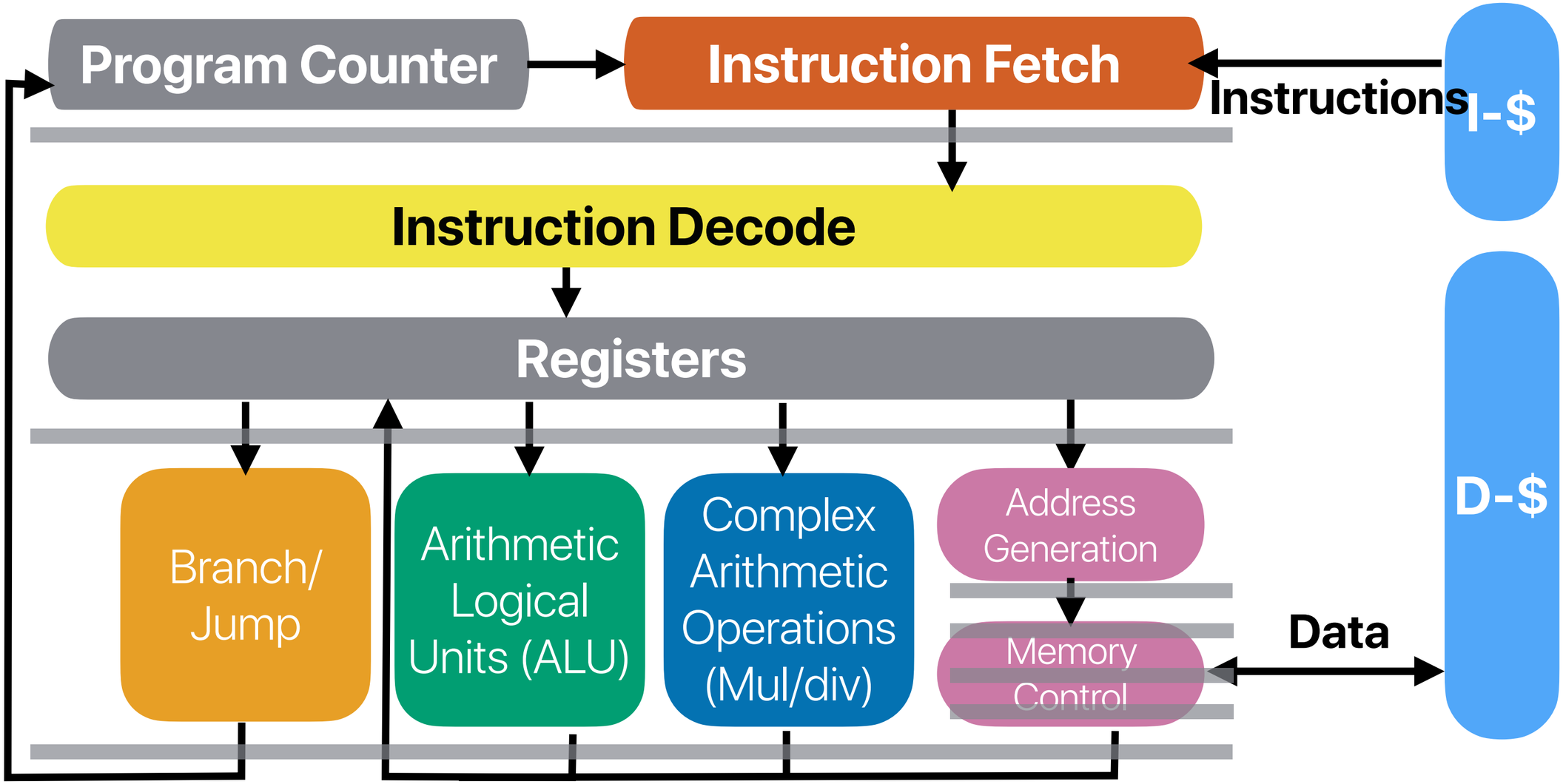
```
① xorl %eax, %eax
② movl (%rdi), %ecx
③ addl %ecx, %eax
④ addq $4, %rdi
⑤ cmpq %rdx, %rdi
```





"Split L1" cache!

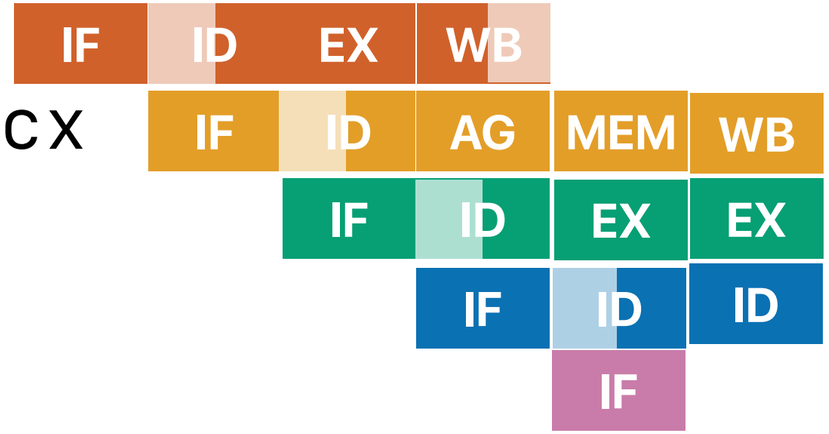
Split L1-$


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

Structural Hazards

- Force later instructions to stall
- Improve the pipeline unit design to allow parallel execution
  - Write-first, read later register files
  - Split L1-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

Control Hazards

Outline

- Dynamic Branch Predictions

How does the code look like?

```
for (unsigned i = 0; i < size; ++i) {
    if (data[i] < threshold)
        call_when_true(&a[i]);
    else
        call_when_false(&a[i]);
}
```

// taken when true

// taken when false

Branch taken simply means we are using branch target address as the next PC

```
.L12:
    movl  -32(%rbp), %eax
    cltq
    leaq  0(,%rax,8), %rdx
    movq  -8(%rbp), %rax
    addq  %rdx, %rax
    movq  (%rax), %rax
    andl  $1, %eax
    testq %rax, %rax
    je .L10
    movl  -32(%rbp), %eax
    cltq
    leaq  0(,%rax,8), %rdx
    movq  -8(%rbp), %rax
    addq  %rdx, %rax
    movq  %rax, %rdi
    call  call_when_true
```

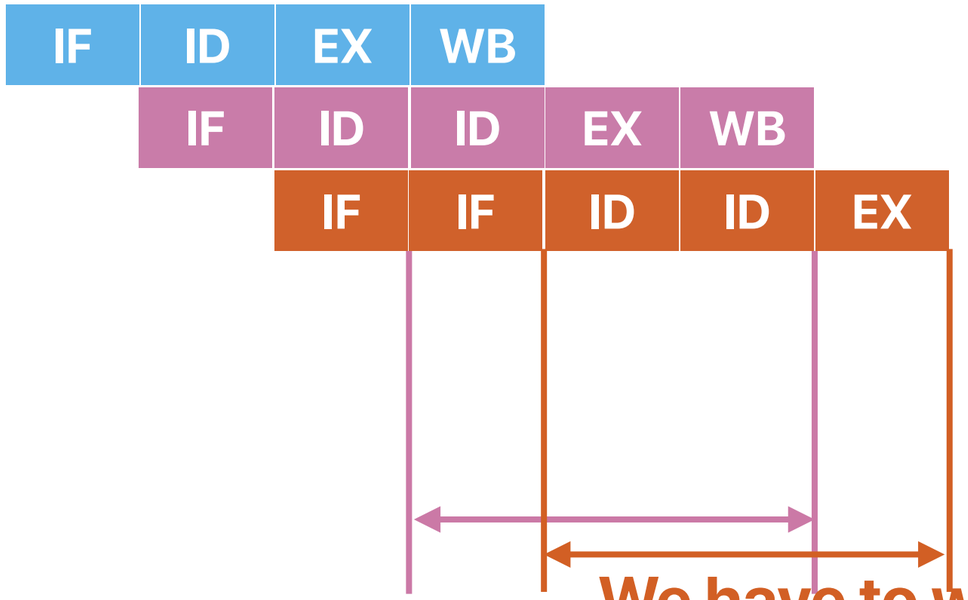
Branch taken

```
    jmp   .L11
.L10:
    movl  -32(%rbp), %eax
    cltq
    leaq  0(,%rax,8), %rdx
    movq  -8(%rbp), %rax
    addq  %rdx, %rax
    movq  %rax, %rdi
    call  call_when_false
.L11:
    addl  $1, -32(%rbp)
.L9:
    movl  -32(%rbp), %eax
    cmpl  -28(%rbp), %eax
    jl .L12
```

Branch taken

Why is “branch” problematic in performance?

```
① addq $8, %rbx
② cmpq %r12, %rbx
③ jne  .L5
```

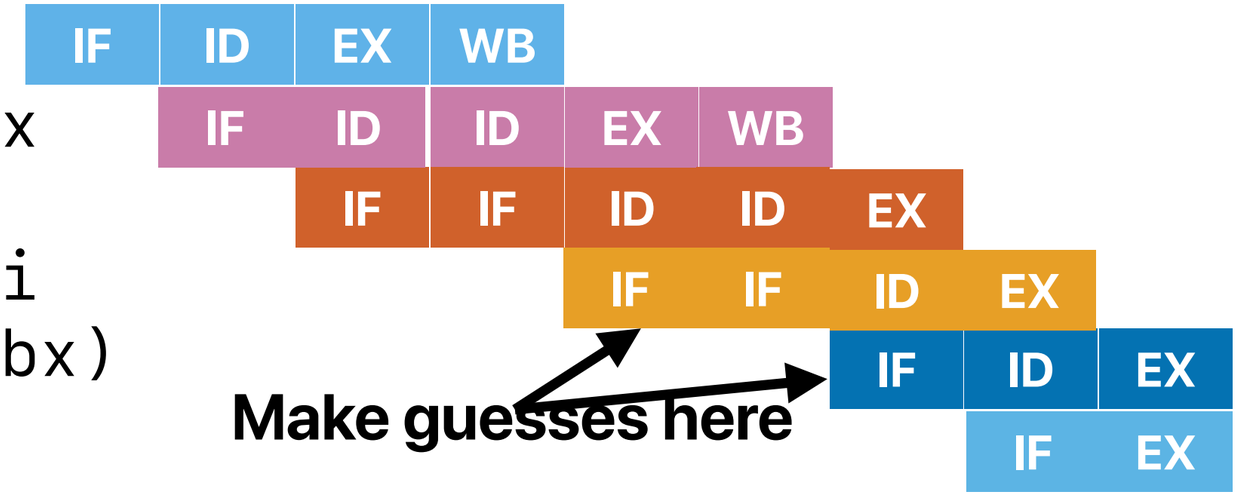


The latency of executing the cmpq instruction

We have to wait almost as long as the latency of the previous instruction to make a decision — we cannot fetch anything before that

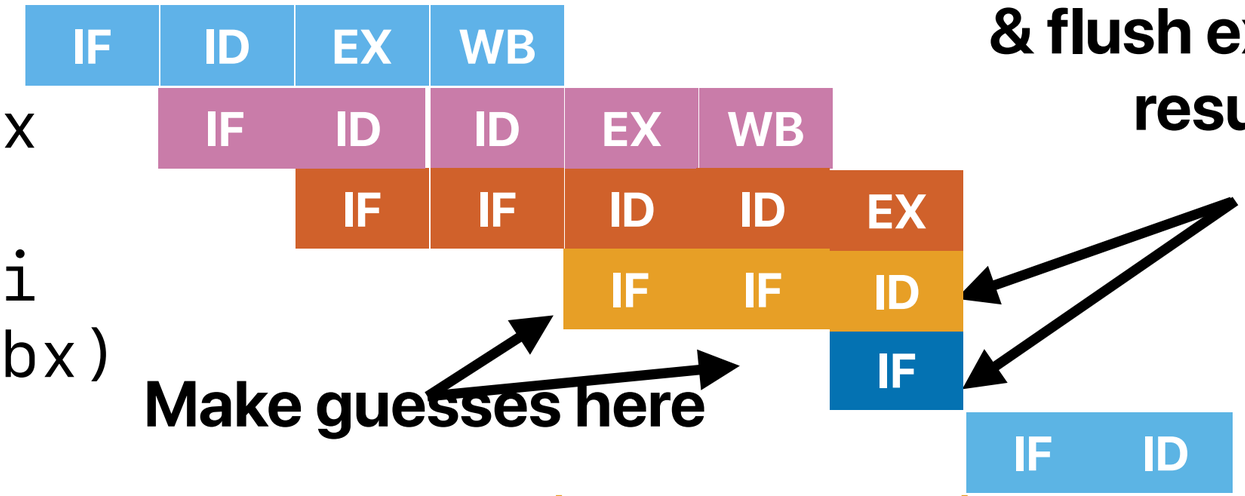
Prediction: What if we guessed right?

```
①         addq  $8, %rbx
②         cmpq  %r12, %rbx
③         jne   .L5
④ .L5:    movq  %rbx, %rdi
⑤         cmpq  %rbp, (%rbx)
```



Prediction: What if we are wrong?

```
①        addq  $8, %rbx
②        cmpq  %r12, %rbx
③        jne   .L5
④ .L5:   movq  %rbx, %rdi
⑤        cmpq  %rbp, (%rbx)
⑥ .L14:  popq  %rbx
```

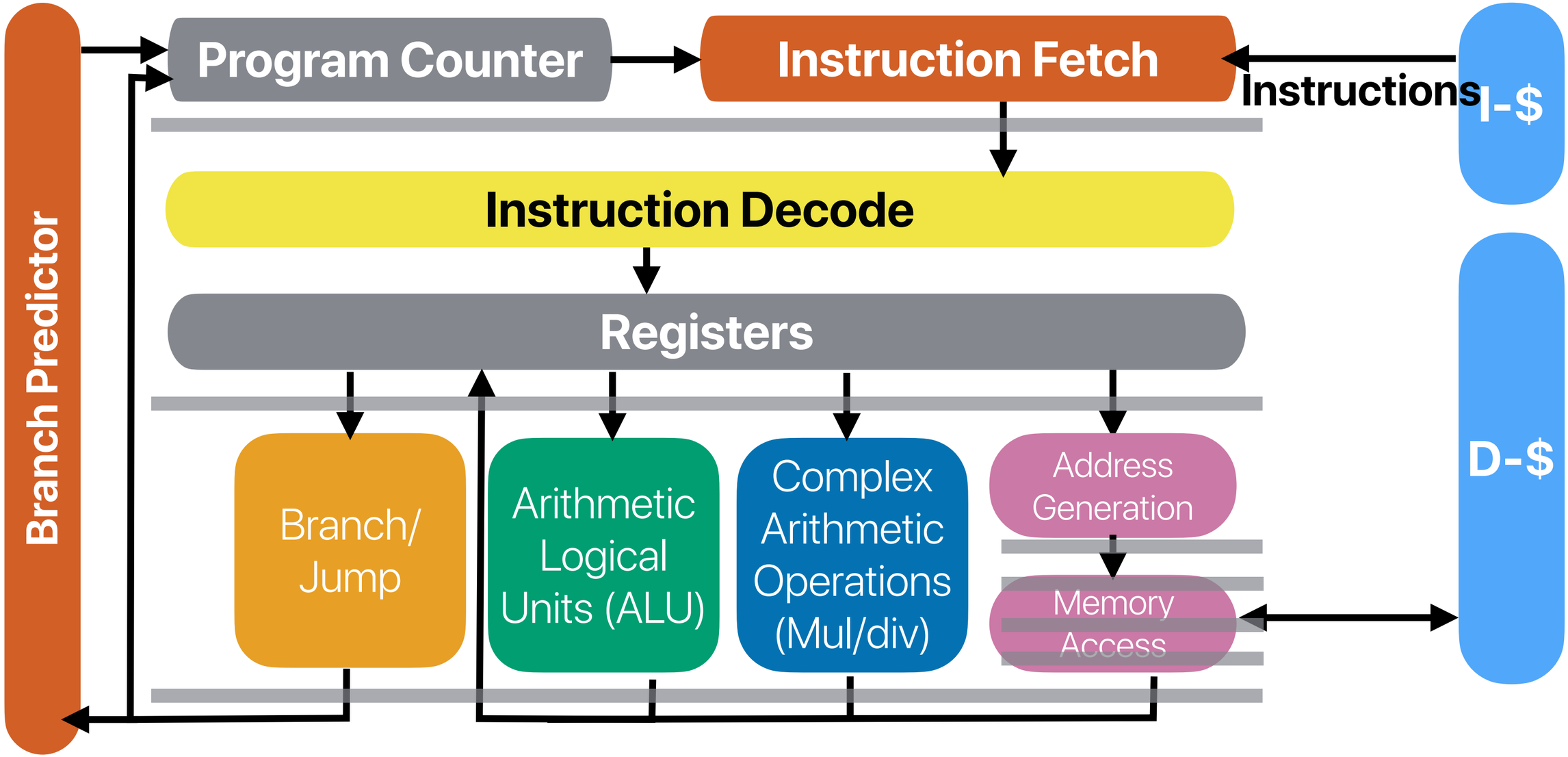


Bubble the rest stages & flush executed results

Make guesses here

We still only wasted three cycles — same as not doing anything

Microprocessor with a “branch predictor”



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

What should branch prediction "predict"

- How many of the following statements are true regarding the why is branch can lead to serious performance issues
  - ① The result value of the previous instruction generating the input to the branch
  - ② The direction of the branch (i.e., taken or not-taken)
  - ③ The target address of the branch
  - ④ The forth-through address of the branch

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

What should branch prediction "predict"

- How many of the following statements are true regarding the why is branch can lead to serious performance issues
  - ① The result value of the previous instruction generating the input to the branch
  - ② The direction of the branch (i.e., taken or not-taken)
  - ③ The target address of the branch
  - ④ The forth-through address of the branch

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

What are the "outcome" of the branch?

- **Taken, not-take You need to predict that — history/states**
- **Target address, if taken**

You need a cheatsheet for that — branch target buffer

Announcement

- Assignment #3 due tonight
- Reading quiz #6 due next Tuesday
- Programming assignment due next Thursday
  - Only 3 submissions for now
  - Gradescope is based on AWS — no guarantee if their server can schedule your workloads as soon as you want
- Plan your time carefully
  - Time management is an important skill
  - You get what you paid for

Computer
Science &
Engineering

203

つづく