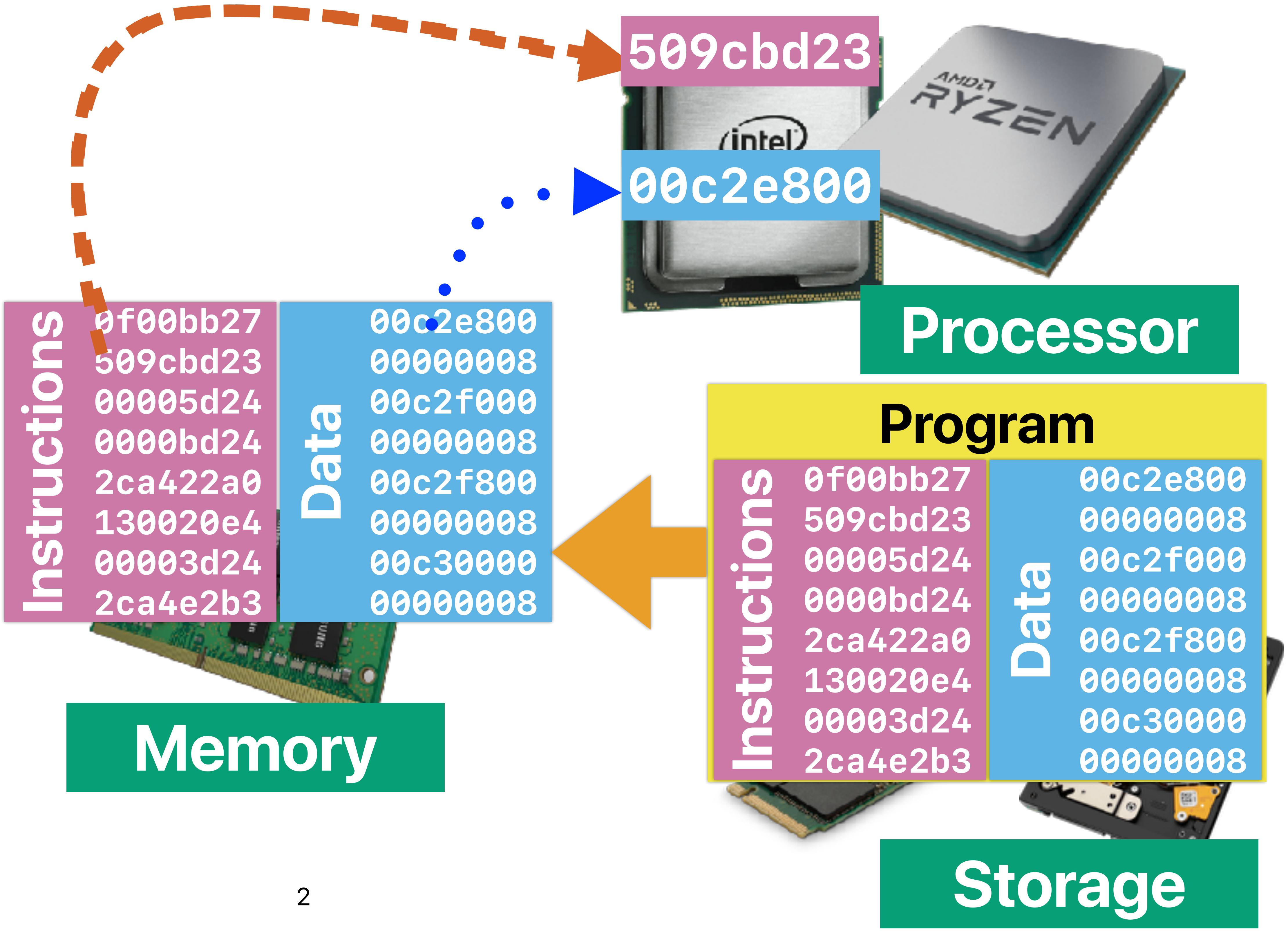
Modern Processor Design (II): What’s Next?

Hung-Wei Tseng

von Neumman Architecture



Recap: Microprocessor — a collection of functional units

Instructions

Instruction Set Architecture

Logical operations

Simple Arithmetic Operations (Add/Sub)

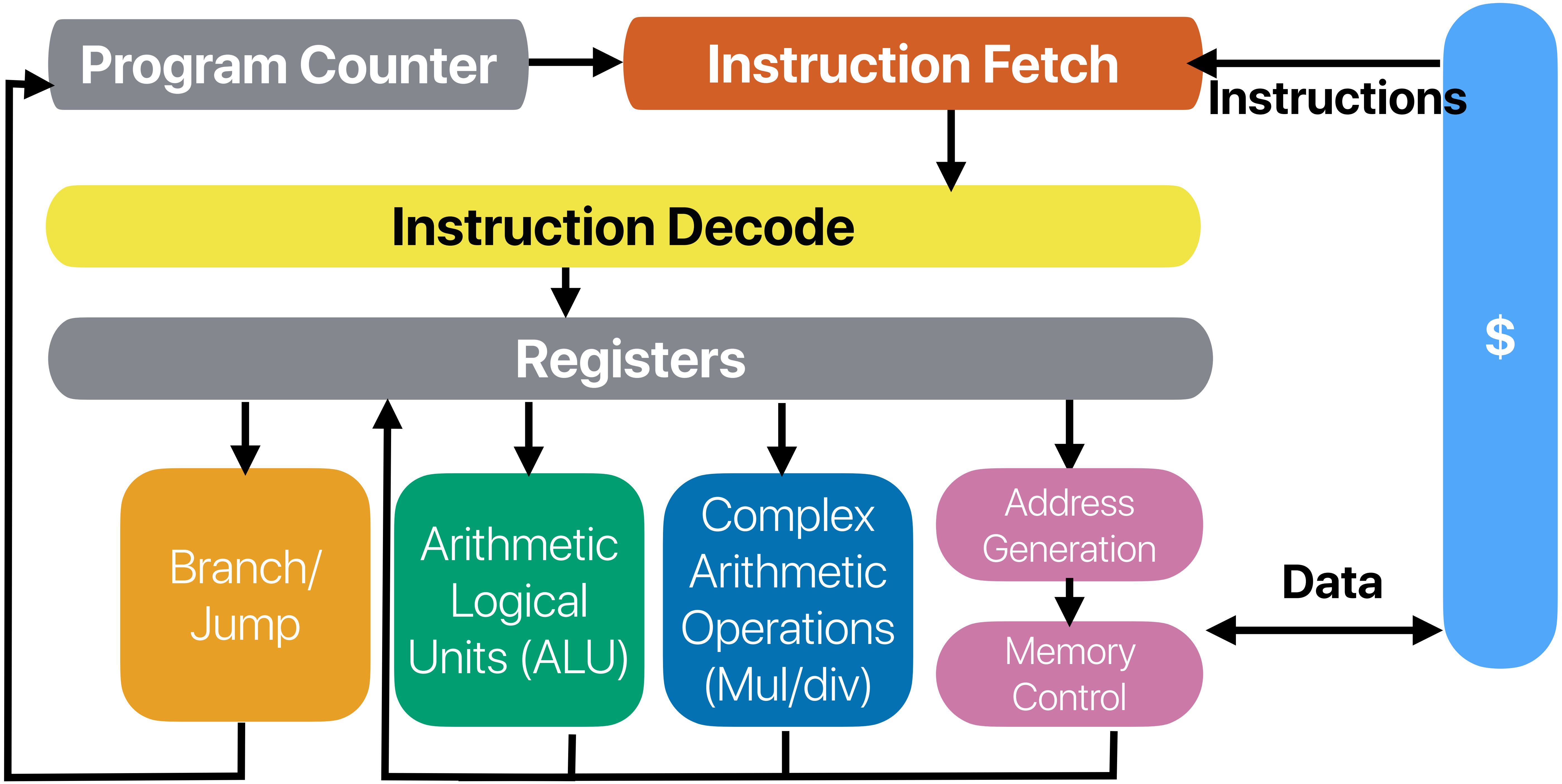
Complex Arithmetic Operations (Mul/div)

Branch/ Jump

Memory Operations

Processor

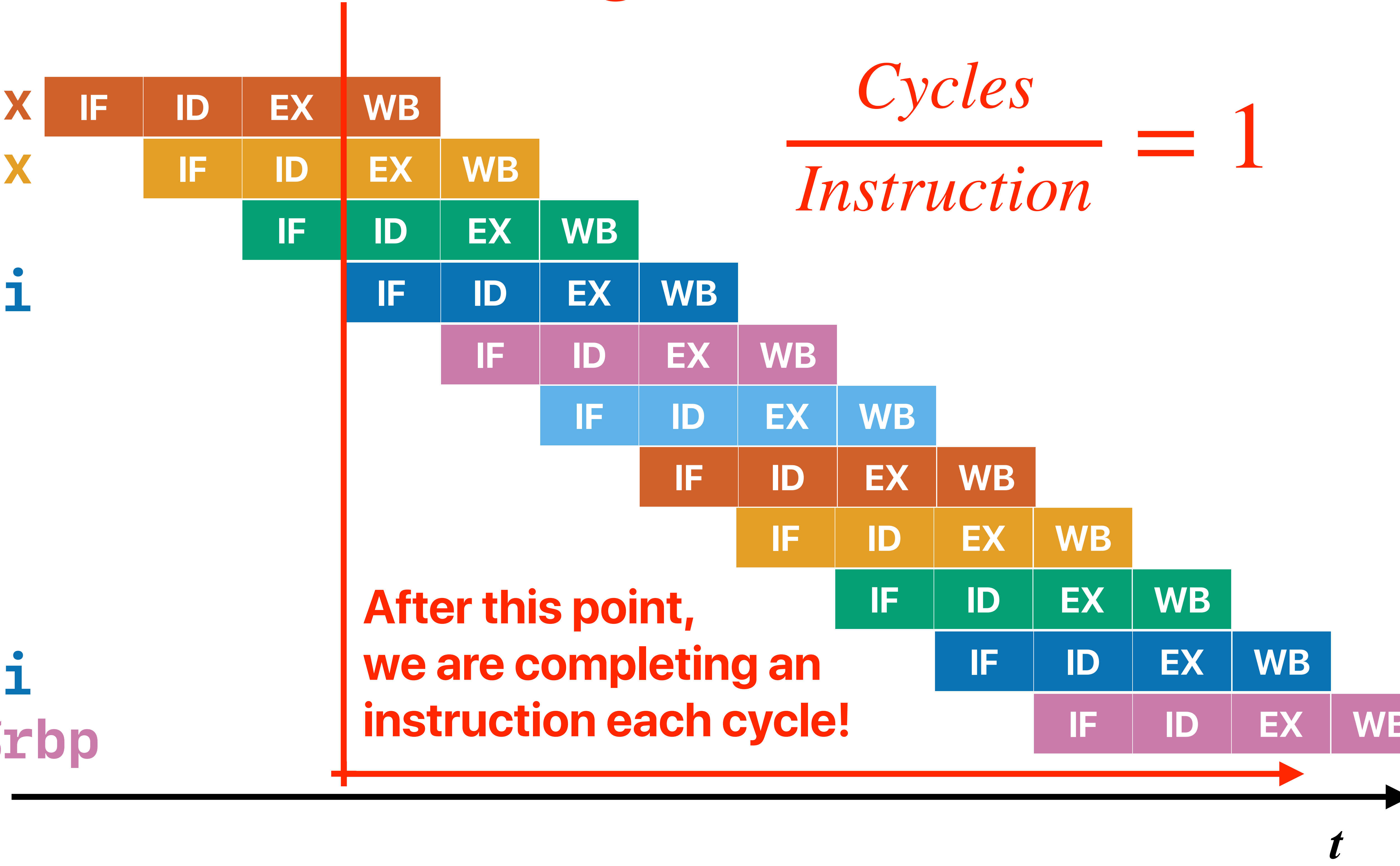
Functional Units of a Microprocessor



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
```

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



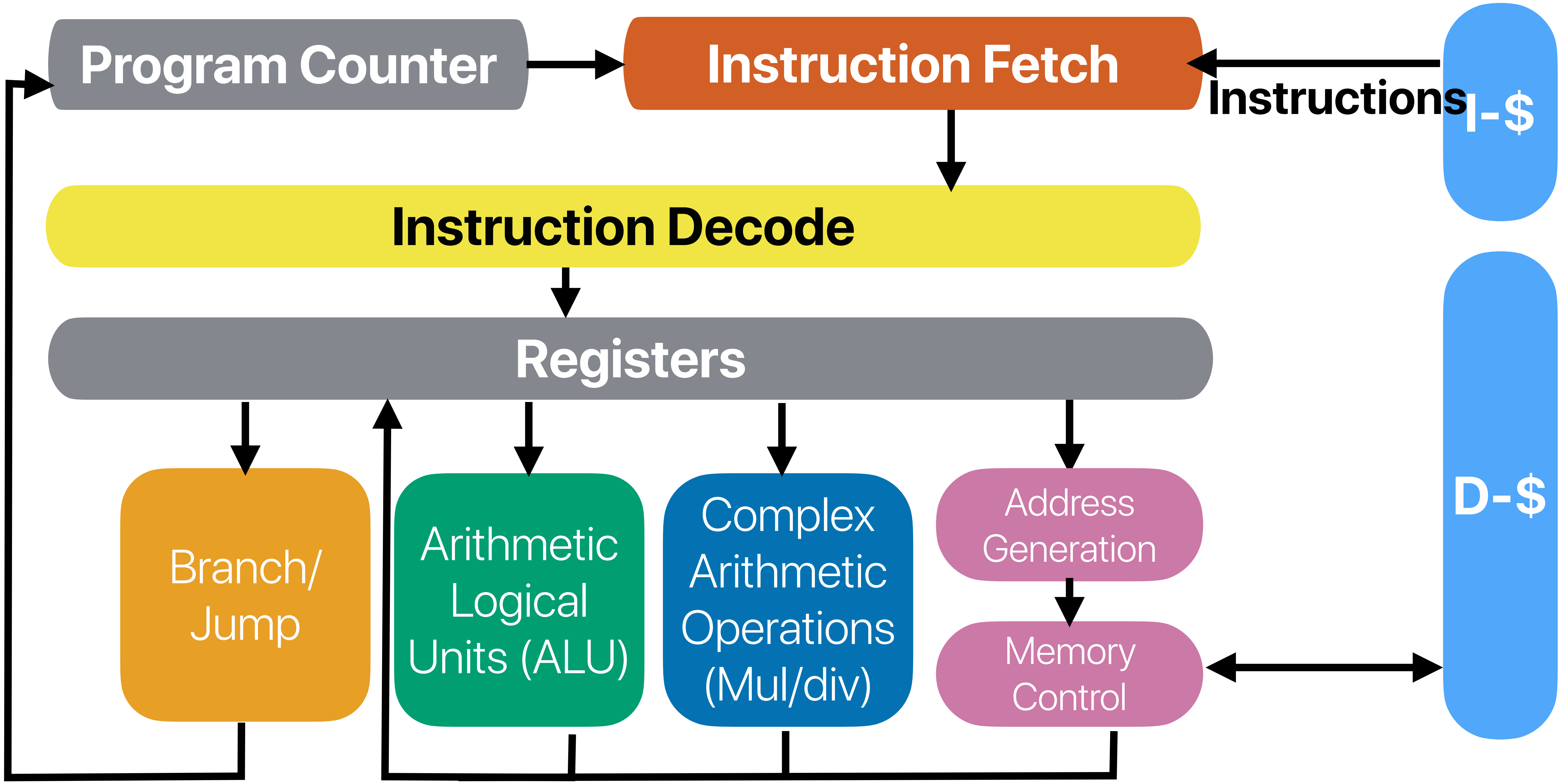
Three types of pipeline hazards

- 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

Takeaways: Pipeline

- Modern processors are pipelined to improve the throughput and hardware utilization
- We cannot achieve the optimal throughput as we have pipeline hazards
  - Structural hazards — pipeline elements do not allow two instructions to perform their tasks at the same cycle
  - Control hazards — the pipeline cannot continuously fetch/feed instructions due to the uncertainty of the upcoming instruction
  - Data hazards — the correct input of an instruction is not generated yet
- Stall can address the issue — but slow
- Improve the pipeline unit design to allow parallel execution
  - Write-first, read later register files
  - Split L1-Cache
  - Force all instructions go through exactly the same number of stages
  - Non-blocking, multiple-banked cache/memory

Recap: Microprocessor



Outline

- Control Hazards
- Branch prediction
  - Basic two-bit local predictor
  - Two-bit global predictor
  - Perceptron
  - Hybrid predictors
    - TAGE
    - Tournament

Control Hazards

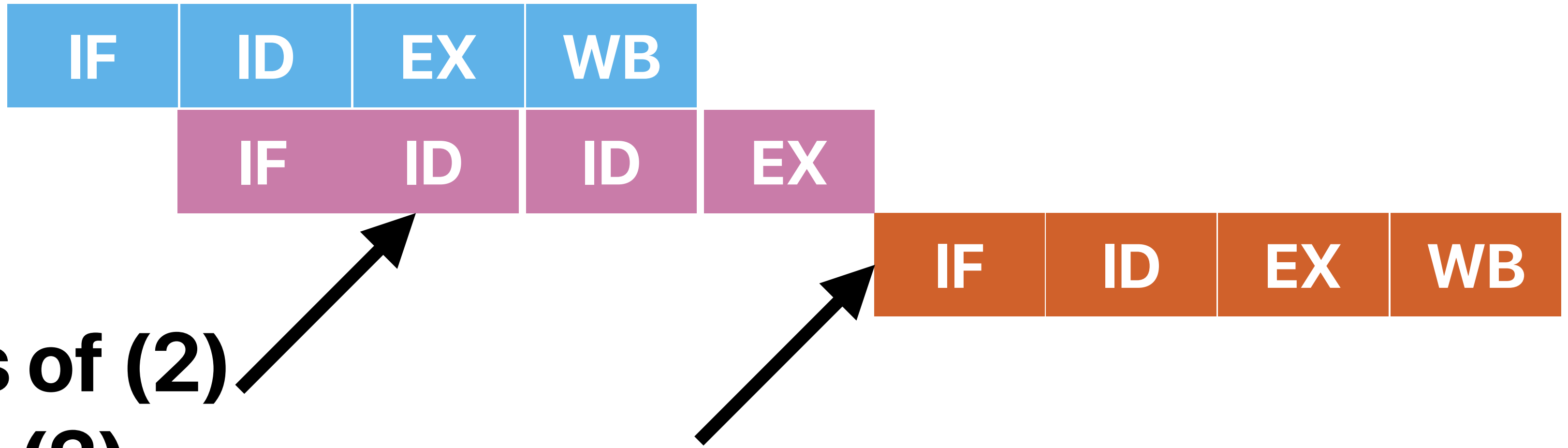
https://www.pollev.com/hungweitseng close in 1:30

Why can't we proceed without stalls/no-ops?

- How many of the following statements are true regarding why we have to stall for each branch in the current pipeline processor
  ① The target address when branch is taken is not available for instruction fetch stage of the next cycle
  ② The target address when branch is not-taken is not available for instruction fetch stage of the next cycle
  ③ The branch outcome cannot be decided until the comparison result of ALU is not out
  ④ The next instruction needs the branch instruction to write back its result
  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

Control Hazard

```
⑤ cmpq %rdx, %rdi
⑥ jne  .L3
⑦ ret
```



We don’t the address of (2) if we want to go to (2)

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

Why can't we proceed without stalls/no-ops?

- How many of the following statements are true regarding why we have to stall for each branch in the current pipeline processor
  ① The target address when branch is taken is not available for instruction fetch stage of the next cycle
  ② The target address when branch is not-taken is not available for instruction fetch stage of the next cycle
  ③ The branch outcome cannot be decided until the comparison result of ALU is not out
  ④ The next instruction needs the branch instruction to write back its result
  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

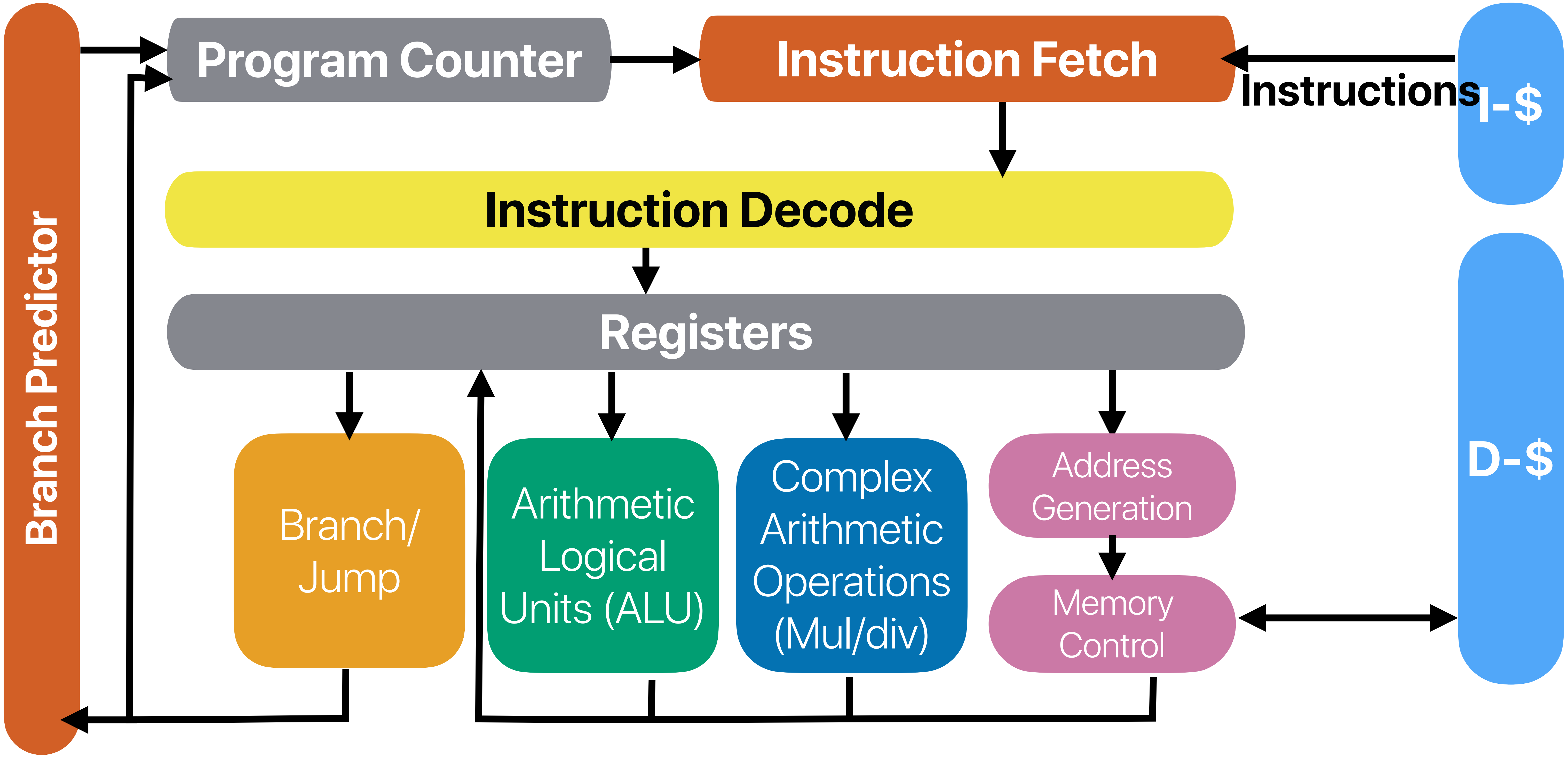
Why can't we proceed without stalls/no-ops?

- How many of the following statements are true regarding why we have to stall for each branch in the current pipeline processor
  1. The target address when branch is taken is not available for instruction fetch stage of the next cycle **You need a cheatsheet for that — branch target buffer**
  2. The target address when branch is not-taken is not available for instruction fetch stage of the next cycle **You need to predict that — history/states**
  3. The branch outcome cannot be decided until the comparison result of ALU is not out
  4. The next instruction needs the branch instruction to write back its result

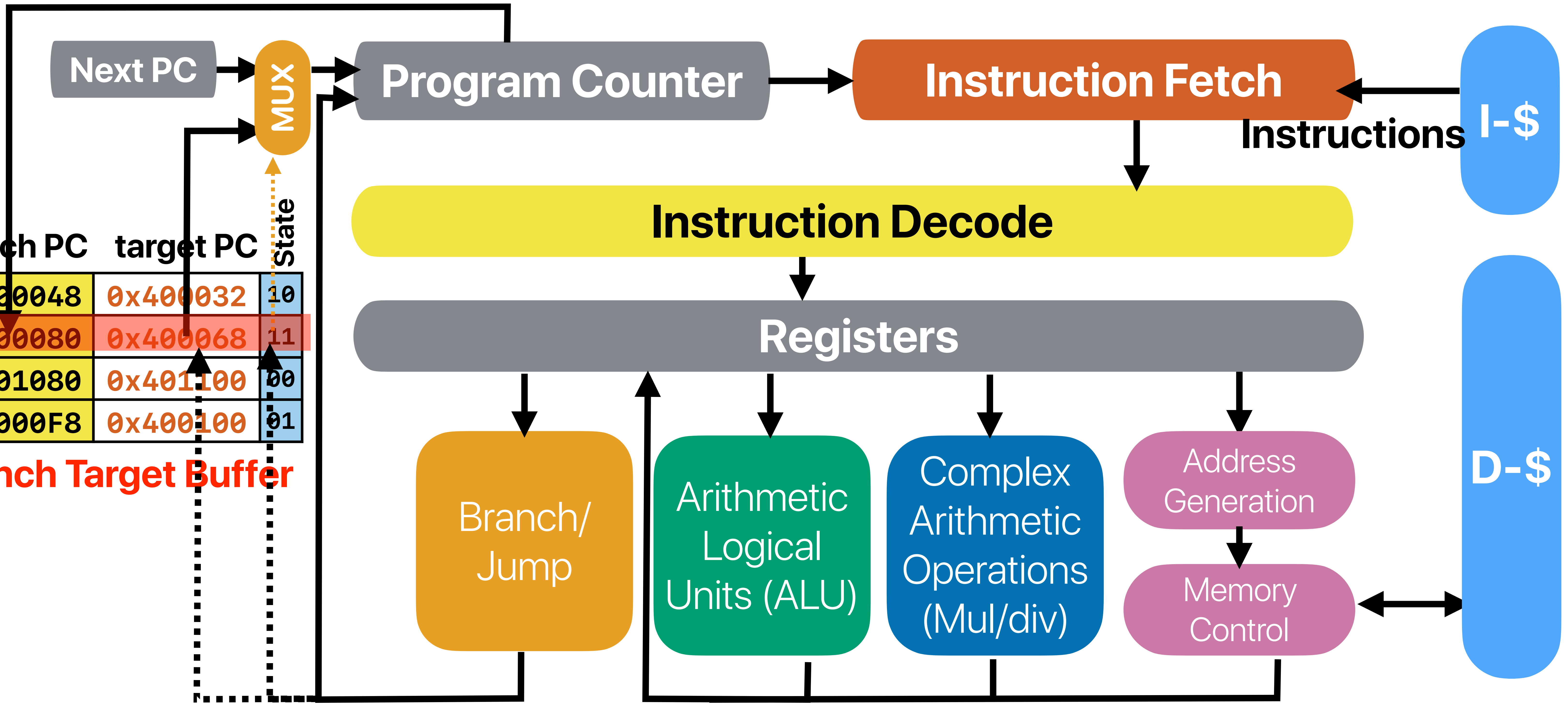
  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

Dynamic Branch Prediction

Microprocessor with a “branch predictor”



Detail of a basic dynamic branch predictor

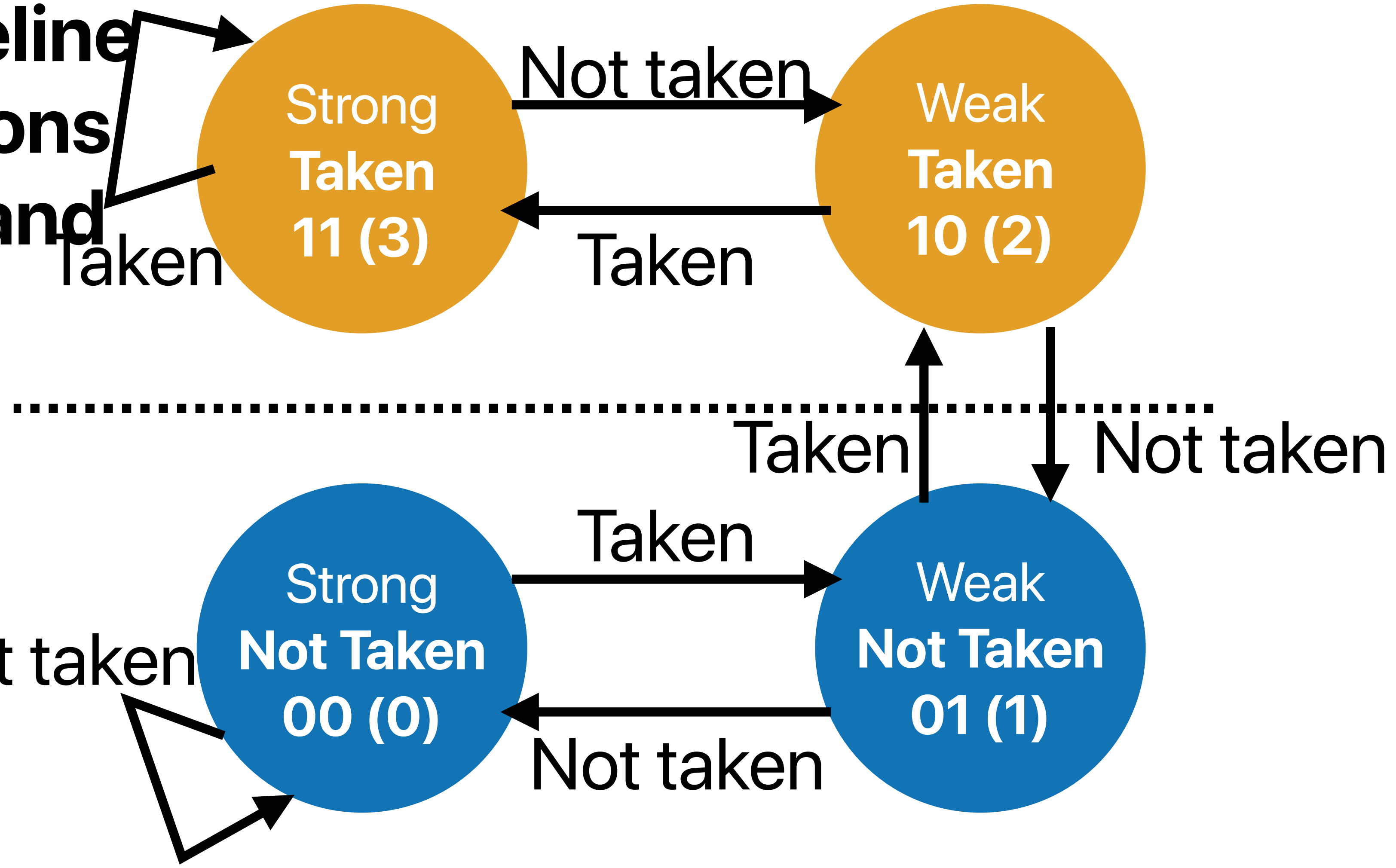


branch PC	target PC	State
0x400048	0x400032	10
0x400080	0x400068	11
0x401080	0x401100	00
0x4000F8	0x400100	01

Branch Target Buffer

2-bit/Bimodal local predictor

- Local predictor — every branch instruction has its own state
- 2-bit — each state is described using 2 bits
- Change the state based on **actual** outcome
- If we guess right — no penalty
- **If we guess wrong — flush (clear pipeline registers) for mis-predicted instructions that are currently in IF and ID stages and reset the PC**

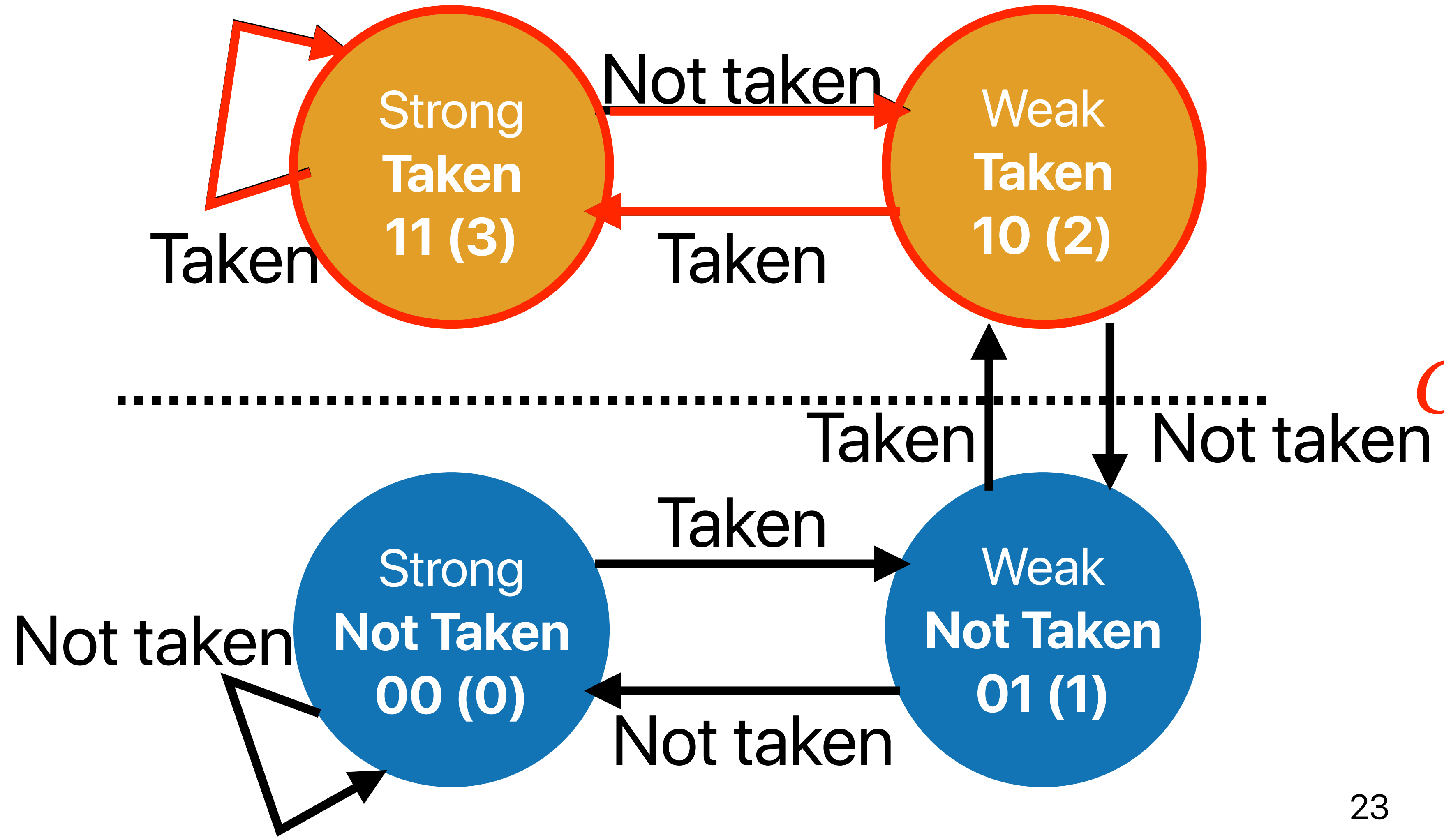


Predict Taken

branch PC	target PC	State
0x400048	0x400032	10
0x400080	0x400068	11
0x401080	0x401100	00
0x4000F8	0x400100	01

2-bit local predictor

```
i = 0;
do {
    sum += a[i];
} while(++i < 10);
```



i	state	predict	actual
1	10	T	T
2	11	T	T
3	11	T	T
4-9	11	T	T
10	11	T	NT

90% accuracy!

$$CPI_{average} = 1 + 20\% \times 10\% \times 2 = 1.04$$

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

2-bit local predictor

- What's the overall branch prediction (include both branches) accuracy for this nested for loop?

```
i = 0;
do {
    if( i % 2 != 0) // Branch X, taken if i % 2 == 0
       a[i] *= 2;
    a[i] += i;
} while ( ++i < 100)// Branch Y
```

(assume all states started with 00)

A. ~25%
B. ~33%
C. ~50%
D. ~67%
E. ~75%

2-bit local predictor

- What's the overall branch prediction (include both branches) accuracy for this nested for loop?

```
i = 0;
do {
    if( i % 2 != 0) // Branch X, taken if i % 2 == 0
       a[i] *= 2;
    a[i] += i;
} while ( ++i < 100)// Branch Y
```

(assume all states started with 00)

A. ~25%
B. ~33%
C. ~50%
D. ~67%
E. ~75%

For branch Y, almost 100%,
For branch X, only 50%

i	branch?	state	prediction	actual
0	X	00	NT	T
1	Y	00	NT	T
1	X	01	NT	NT
2	Y	01	NT	T
2	X	00	NT	T
3	Y	10	T	T
3	X	01	NT	NT
4	Y	11	T	T
4	X	00	NT	T
5	Y	11	T	T
5	X	01	NT	NT
6	Y	11	T	T
6	X	00	NT	T
7	Y	11	T	T

2-bit local predictor

- What's the overall branch prediction (include both branches) accuracy for this nested for loop?

```
i = 0;
do {
    if( i % 2 != 0) // Branch X, taken if i % 2 == 0
        a[i] *= 2;
    a[i] += i;
} while ( ++i < 100)// Branch Y
```

Can we do a better job?

(assume all states started with 00)

A. ~25%
B. ~33%
C. ~50%
D. ~67%
E. ~75%

For branch Y, almost 100%,
For branch X, only 50%

i	branch?	state	prediction	actual
0	X	00	NT	T
1	Y	00	NT	T
1	X	01	NT	NT
2	Y	01	NT	T
2	X	00	NT	T
3	Y	10	T	T
3	X	01	NT	NT
4	Y	11	T	T
4	X	00	NT	T
5	Y	11	T	T
5	X	01	NT	NT
6	Y	11	T	T
6	X	00	NT	T
7	Y	11	T	T

Two-level global predictor

Marius Evers, Sanjay J. Patel, Robert S. Chappell, and Yale N. Patt. 1998. An analysis of correlation and predictability: what makes two-level branch predictors work. In Proceedings of the 25th annual international symposium on Computer architecture (ISCA '98).

2-bit local predictor

- What's the overall branch prediction (include both branches) accuracy for this nested for loop?

```
i = 0;
do {
    if( i % 2 != 0) // Branch X, taken if i % 2 == 0
        a[i] *= 2;
    a[i] += i;
} while ( ++i < 100)// Branch Y
```

(assume all states started with 00)

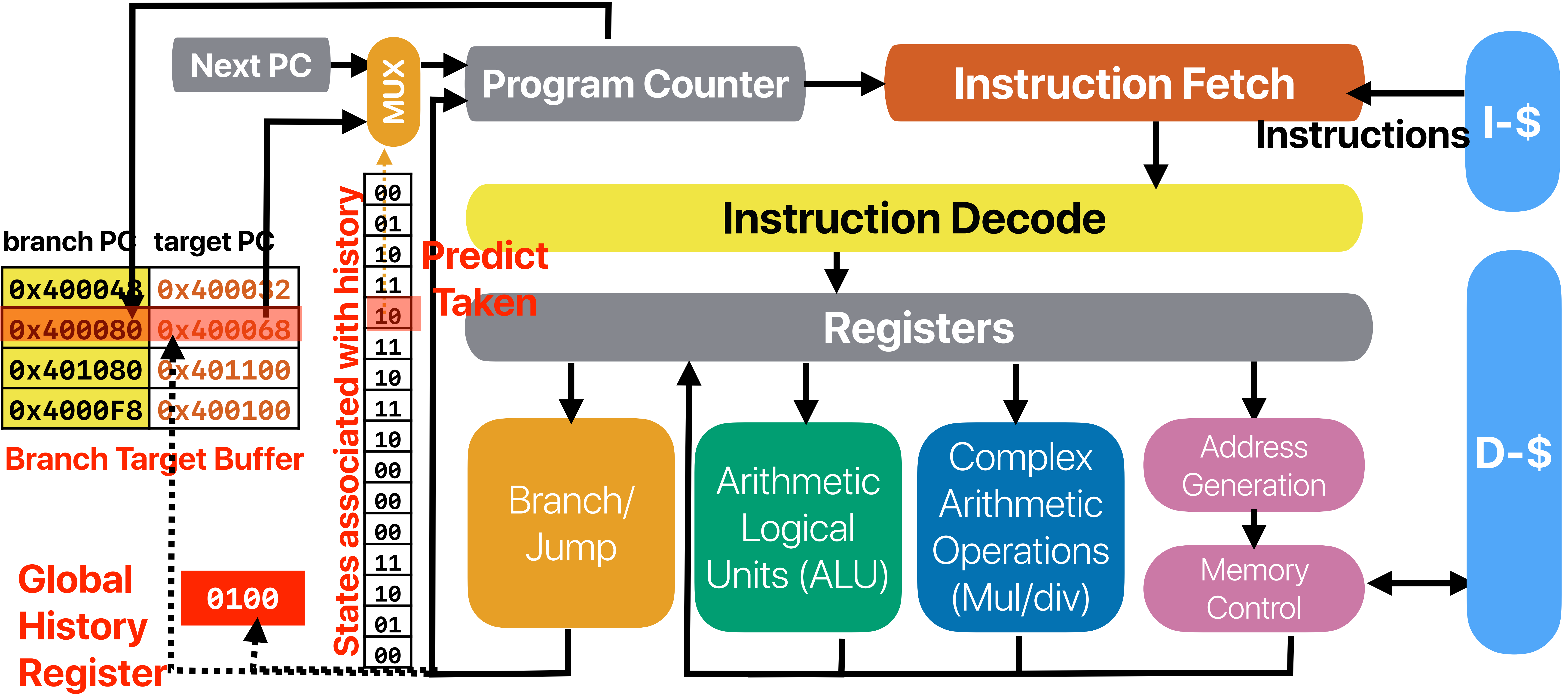
A. ~25%
B. ~33%
C. ~50%
D. ~67%
E. ~75%

This pattern repeats all the time!

For branch Y, almost 100%,
For branch X, only 50%

i	branch?	state	prediction	actual
0	X	00	NT	T
0	Y	00	NT	T
1	X	01	NT	NT
1	Y	01	NT	T
2	X	00	NT	T
2	Y	10	T	T
3	X	01	NT	NT
3	Y	11	T	T
4	X	00	NT	T
4	Y	11	T	T
5	X	01	NT	NT
5	Y	11	T	T
6	X	00	NT	T
6	Y	11	T	T

Detail of a basic dynamic branch predictor



Performance of GH predictor

```
i = 0;
do {
    if( i % 2 != 0) // Branch X, taken if i % 2 == 0
       a[i] *= 2;
    a[i] += i;
} while ( ++i < 100)// Branch Y
```

Near perfect after this

i	branch?	GHR	state	prediction	actual
0	X	000	00	NT	T
0	Y	001	00	NT	T
1	X	011	00	NT	NT
1	Y	110	00	NT	T
2	X	101	00	NT	T
2	Y	011	00	NT	T
3	X	111	00	NT	NT
3	Y	110	01	NT	T
4	X	101	01	NT	T
4	Y	011	01	NT	T
5	X	111	00	NT	NT
5	Y	110	10	T	T
6	X	101	10	T	T
6	Y	011	10	T	T
7	X	111	00	NT	NT
7	Y	110	11	T	T
8	X	101	11	T	T
8	Y	011	11	T	T
9	X	111	00	NT	NT
9	Y	110	11	T	T
10	X	101	11	T	T
10	Y	011	11	T	T

Better predictor?

- Consider two predictors — (L) 2-bit local predictor with unlimited BTB entries and (G) 4-bit global history with 2-bit predictors. How many of the following code snippet would allow (G) to outperform (L)?

I
```
i = 0;
do {
    if( i % 10 != 0)
       a[i] *= 2;
    a[i] += i;
} while ( ++i < 100);
```

II
```
i = 0;
do {
    a[i] += i;
} while ( ++i < 100);
```

III
```
i = 0;
do {
    j = 0;
    do {
      sum += A[i*2+j];
    }
    while( ++j < 2);
} while ( ++i < 100);
```

IV
```
i = 0;
do {
    if( rand() %2 == 0)
       a[i] *= 2;
    a[i] += i;
} while ( ++i < 100)
```

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

Better predictor?

- Consider two predictors — (L) 2-bit local predictor with unlimited BTB entries and (G) 4-bit global history with 2-bit predictors. How many of the following code snippet would allow (G) to outperform (L)?

I — about the same

```
i = 0;
do {
    if( i % 10 != 0)
       a[i] *= 2;
    a[i] += i;
} while ( ++i < 100);
```

II — about the same

```
i = 0;
do {
    a[i] += i;
} while ( ++i < 100);
```

III

```
i = 0;
do {
    j = 0;
    do {
      sum += A[i*2+j];
    }
    while( ++j < 2);
} while ( ++i < 100);
```

IV — L could be better

```
i = 0;
do {
    if( rand() %2 == 0)
       a[i] *= 2;
    a[i] += i;
} while ( ++i < 100)
```

A. 0

B. 1

C. 2

D. 3

E. 4

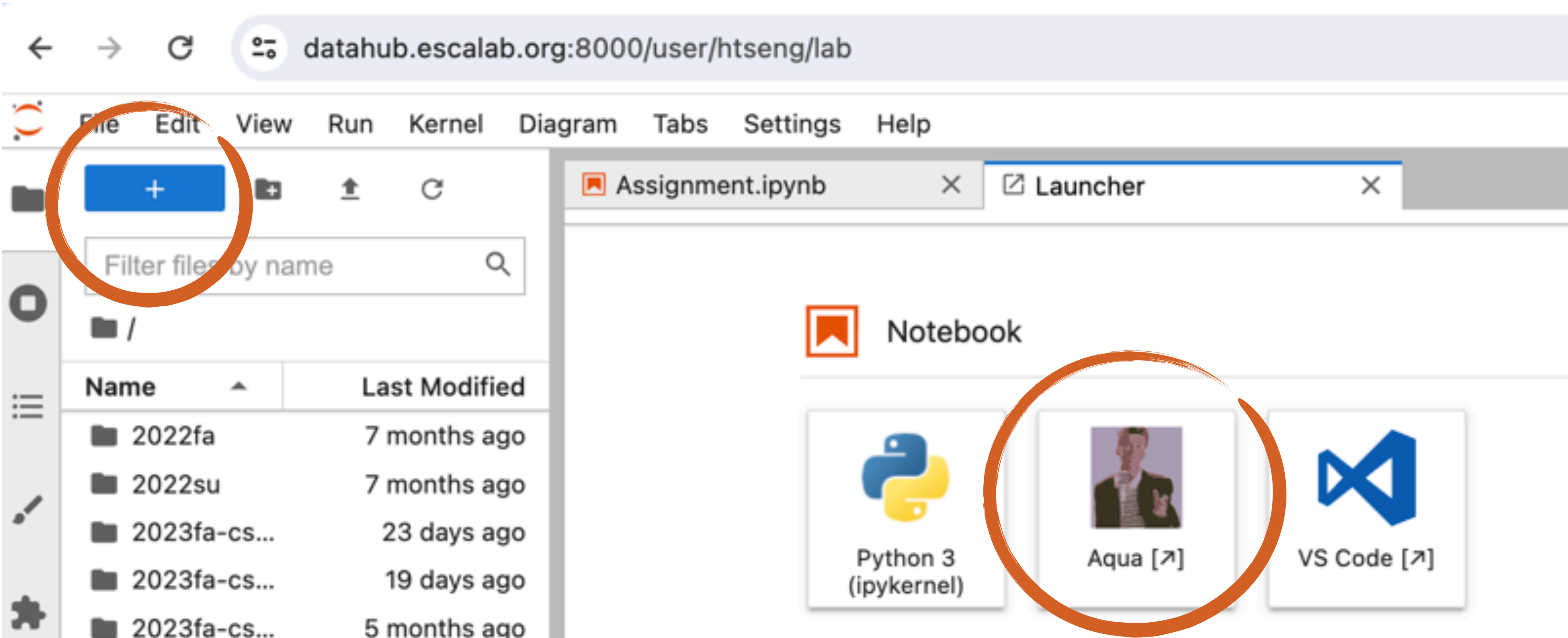
Demo revisited: evaluating the cost of mis-predicted branches

- Compare the number of mis-predictions
- Calculate the difference of cycles
- We can get the “average CPI” of a mis-prediction!

34 cycles!!!

Announcement

- Reading quiz due next Tuesday
- Assignment #4 due next Thursday
  - Please start early
- Please help us test "Aqua"



Computer Science & Engineering

203

つづく