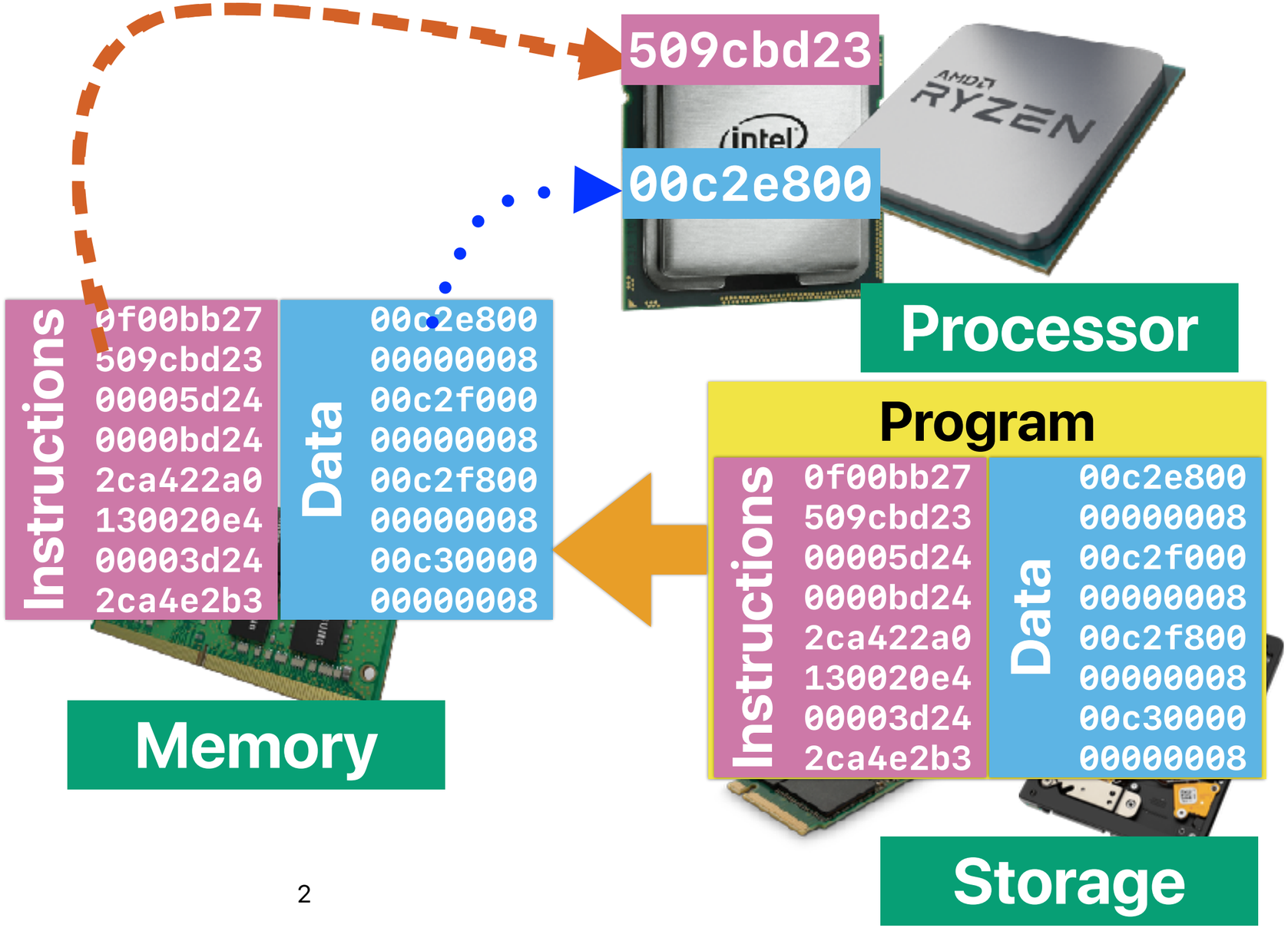
Modern Processor Design (II): I Guess I Just Feel Like

Hung-Wei Tseng

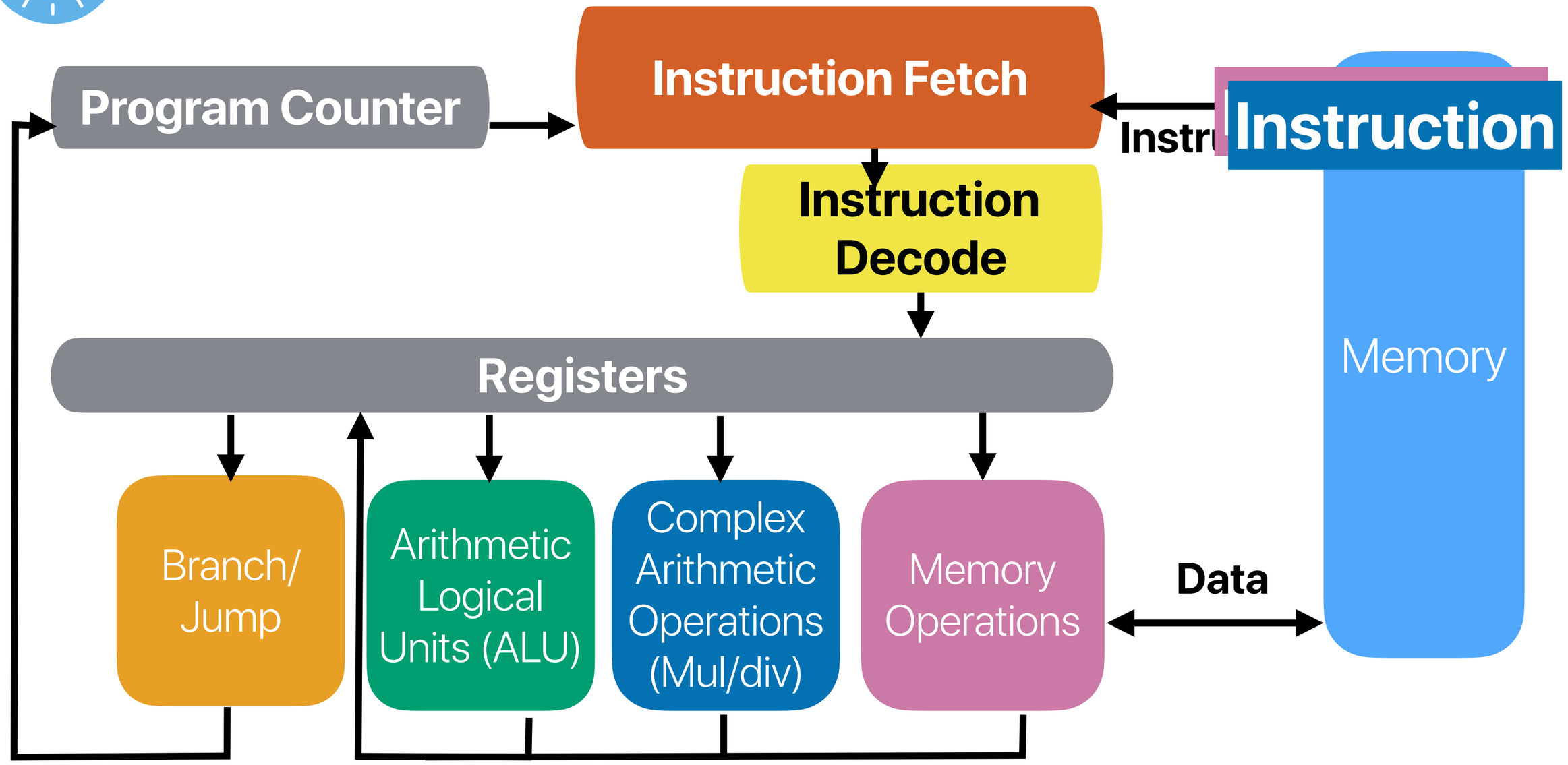
Recap: von Neumman Architecture



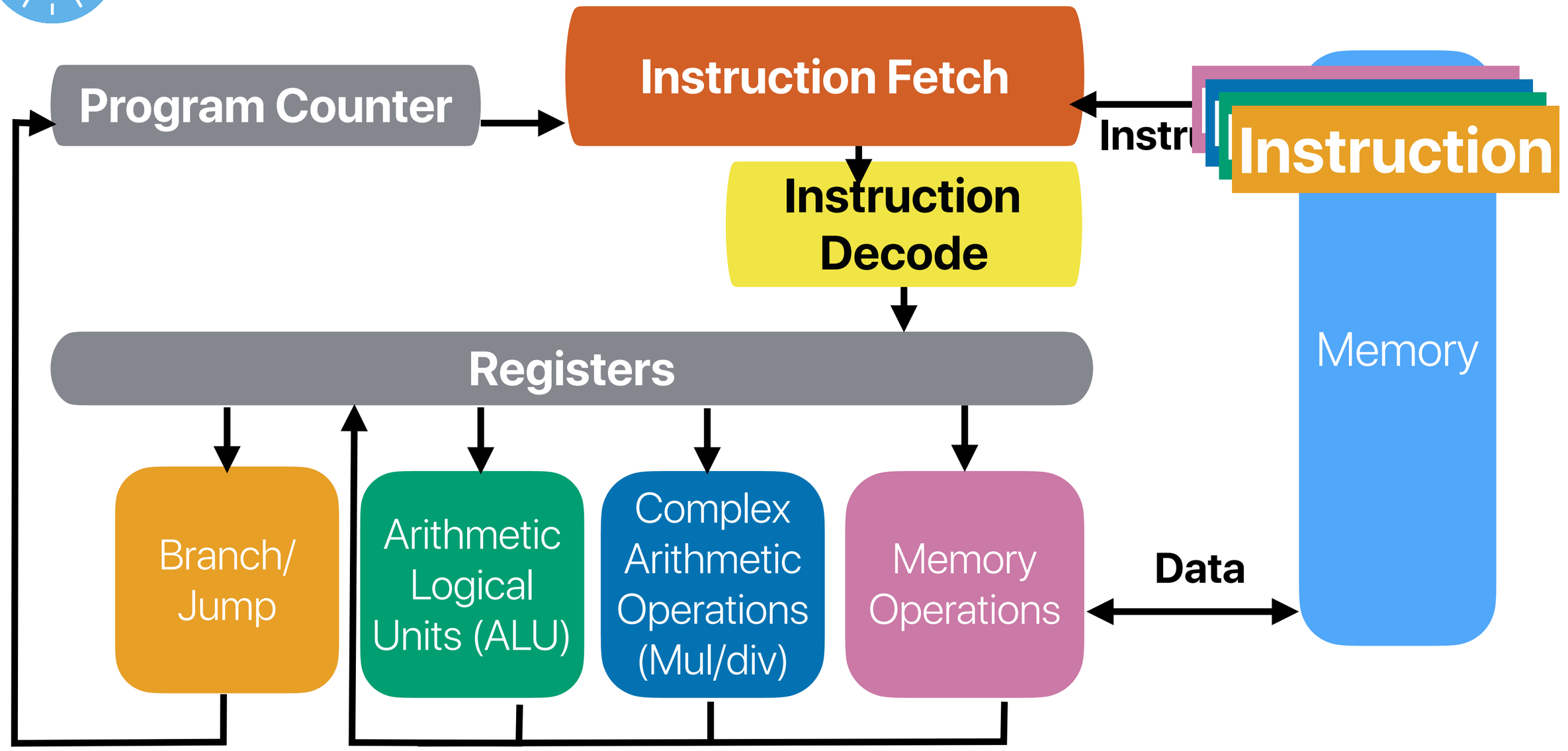
Recap: The “life” of an instruction

- 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

Recap: Within a cycle…

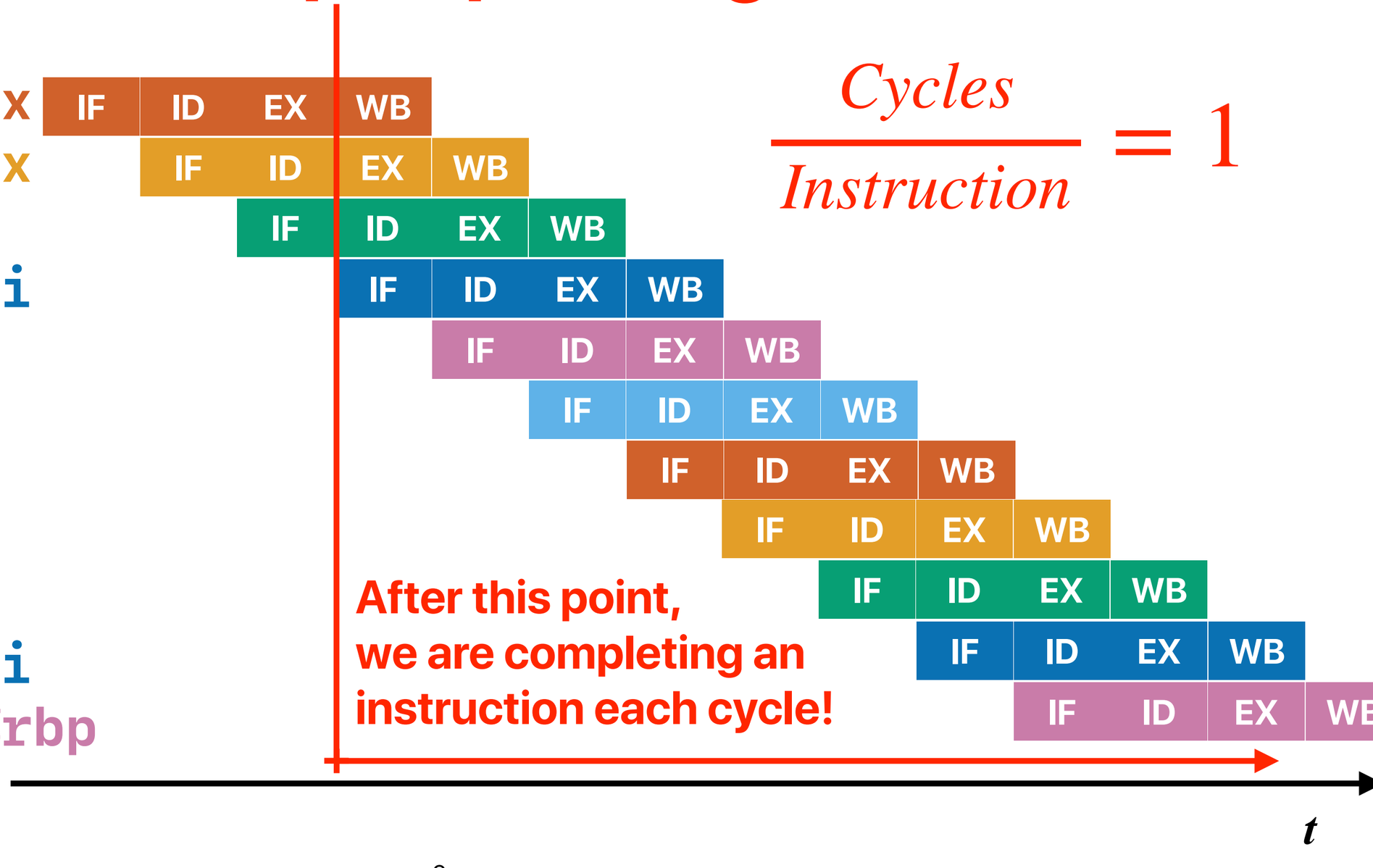


Recap: "Pipeline" the processor!



Recap: 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
```

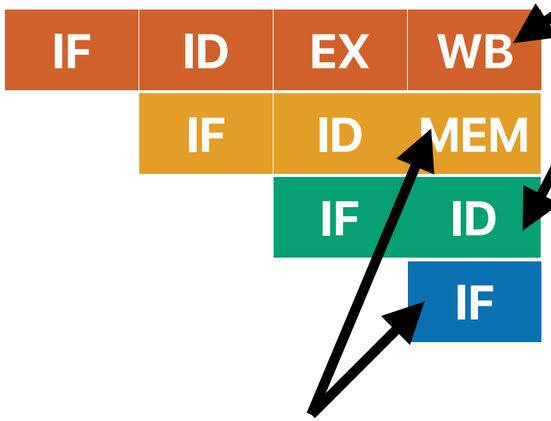


$$\frac{Cycles}{Instruction} = 1$$

Recap: hazards

```
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
```

Structural Hazard



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

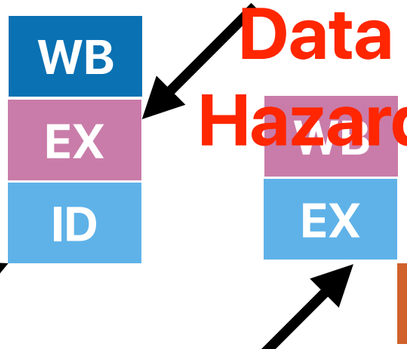
Both (1) and (3) are attempting to access %eax

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

Control Hazard

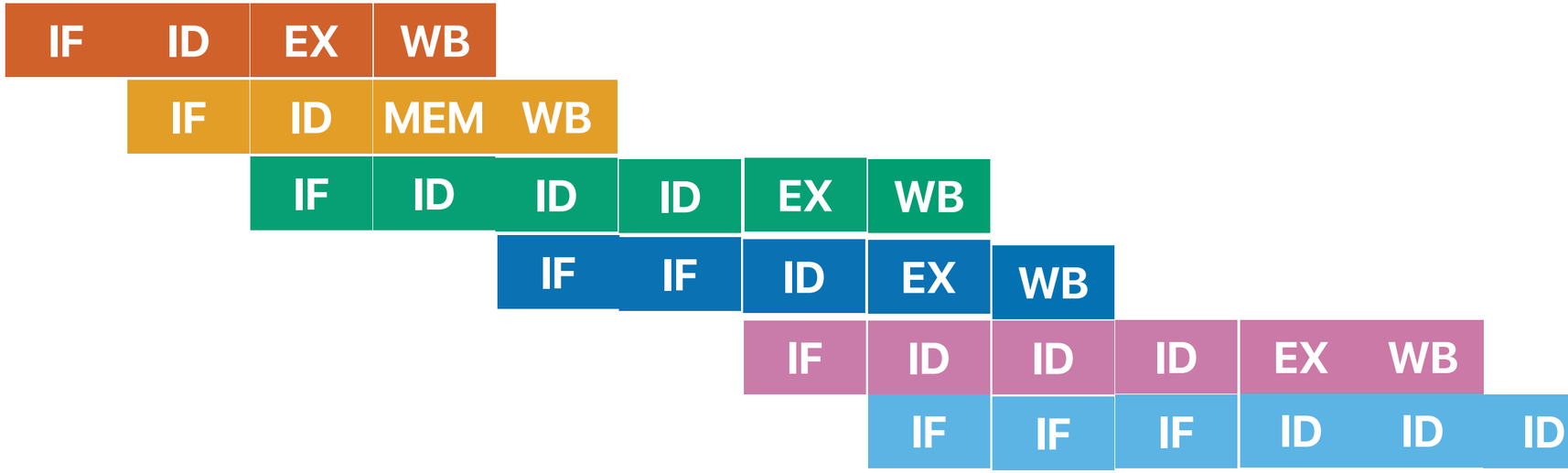
(6) may not have the outcome from (5)

Data Hazard

Recap: Stall whenever we have a hazard

- Stall: the hardware allows the earlier instruction to proceed, all later instructions stay at the same stage

```
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
```

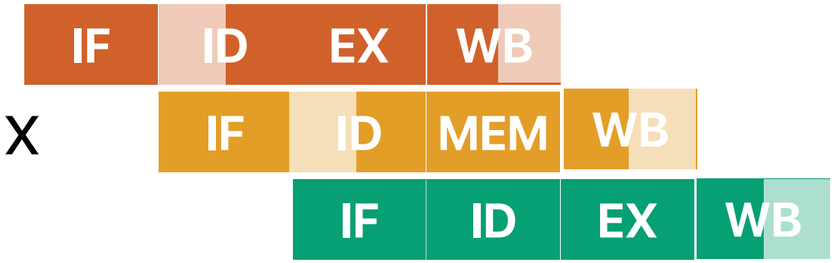


Slow! — 5 additional cycles

Recap: 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!

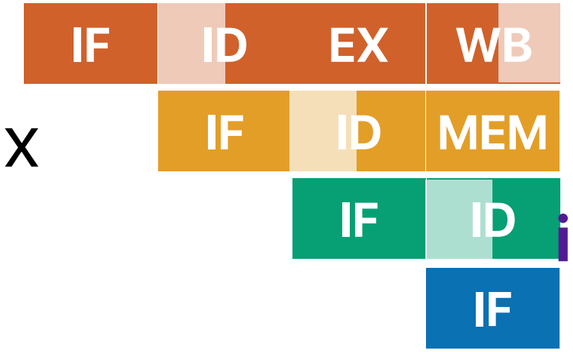
```
① xorl %eax, %eax
② movl (%rdi), %ecx
③ addl %ecx, %eax
```

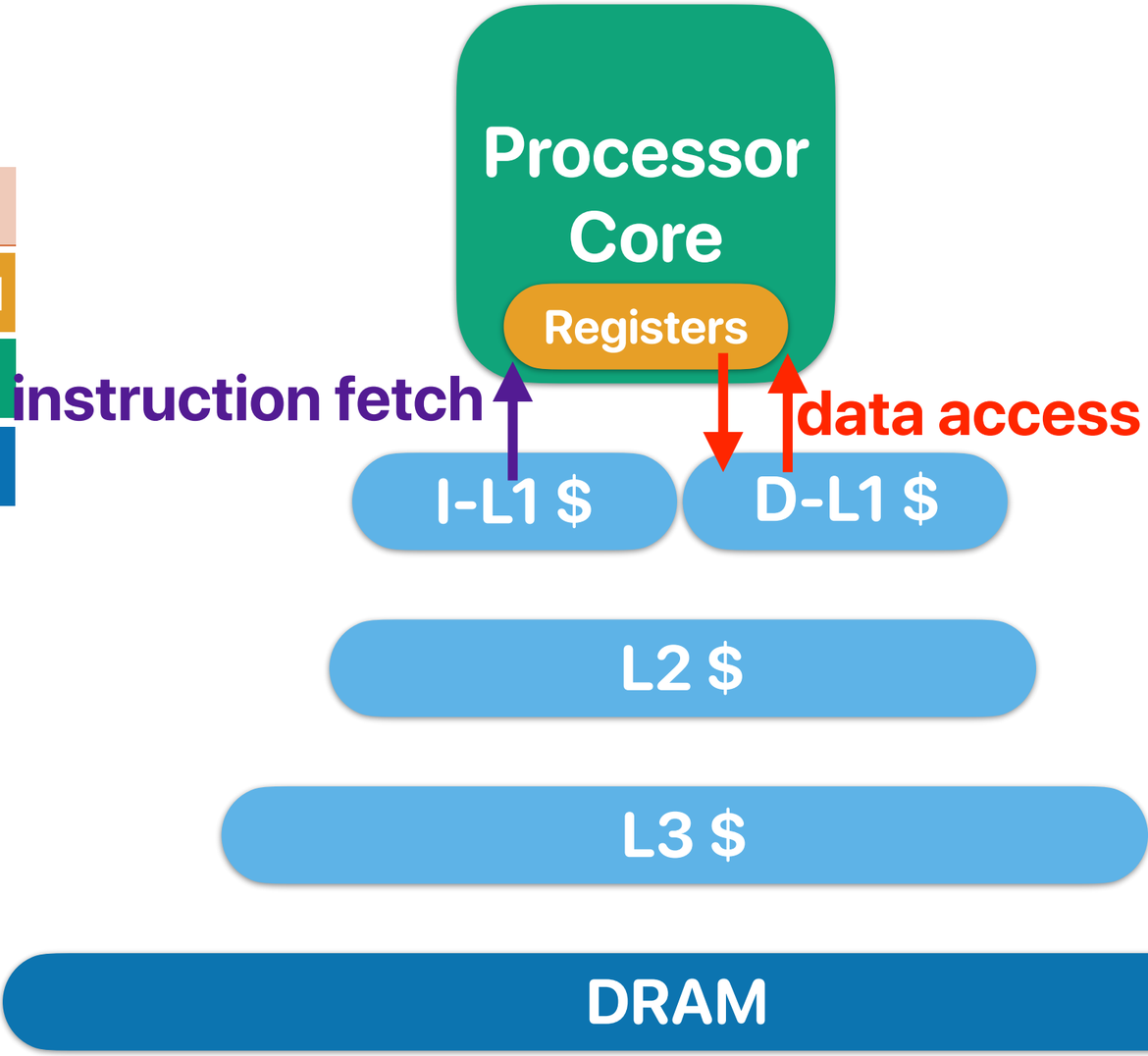


Recap: How to with the conflicts between MEM and IF?

- 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
```





"Split L1" cache!

3:00

What will do you if you are not sure about an answer in exams?

What will you do?

- Guess!
- How to guess?
  - Random?
  - Based on the occurrence of answers?

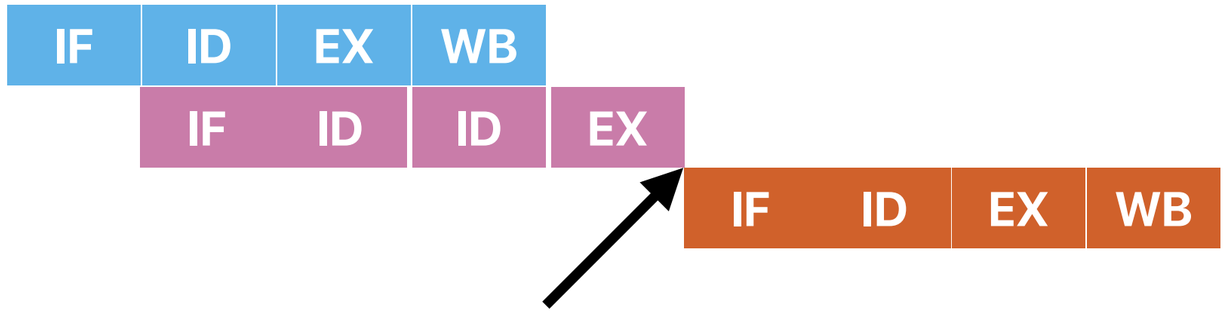
Outline

- Pipeline Hazards — Control Hazards
- Dynamic branch prediction

Control Hazards

Control Hazard

```
① cmpq %rdx, %rdi
② jne  .L3
③ ret
```



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

If the branch instruction “jne” is taken — we have to change the PC to L3

How does the code look like?

```
for (j = 0; j < reps; ++j) {
    for (unsigned i = 0; i < size; ++i) {
        if (data[i] >= threshold)
            sum ++;
    }
}
```

We skip the following code block if data[i]< threshold

We use “backward” branches (taking if going back) to implement loops

```
      loop0:
      .LFB0:
         .cfi_startproc
         endbr64
         pushq %rbp
         .cfi_def_cfa_offset 16
         .cfi_offset 6, -16
         movq  %rsp, %rbp
         .cfi_def_cfa_register 6
         movq  %rdi, -24(%rbp)
         movl  %esi, -28(%rbp)
         movl  %edx, -32(%rbp)
         movl  %ecx, -36(%rbp)
         movl  $0, -8(%rbp)
         movl  $0, -12(%rbp)
         jmp   .L2
.L6:
         movl  $0, -4(%rbp)
         jmp   .L3
.L5:
         movl  -4(%rbp), %eax
         leaq  0(,%rax,4), %rdx
         movq  -24(%rbp), %rax
         addq  %rdx, %rax
         movl  (%rax), %eax
         cmpl  %eax, -32(%rbp)
         jg .L4
         addl  $1, -8(%rbp)
.L4:
         addl  $1, -4(%rbp)
.L3:
         movl  -28(%rbp), %eax
         cmpl  %eax, -4(%rbp)
         jb .L5
         addl  $1, -12(%rbp)
.L2:
         movl  -12(%rbp), %eax
         cmpl  -36(%rbp), %eax
         jl .L6
         movl  -8(%rbp), %eax
         popq  %rbp
         .cfi_def_cfa 7, 8
         ret
```

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
  - 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
  - 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

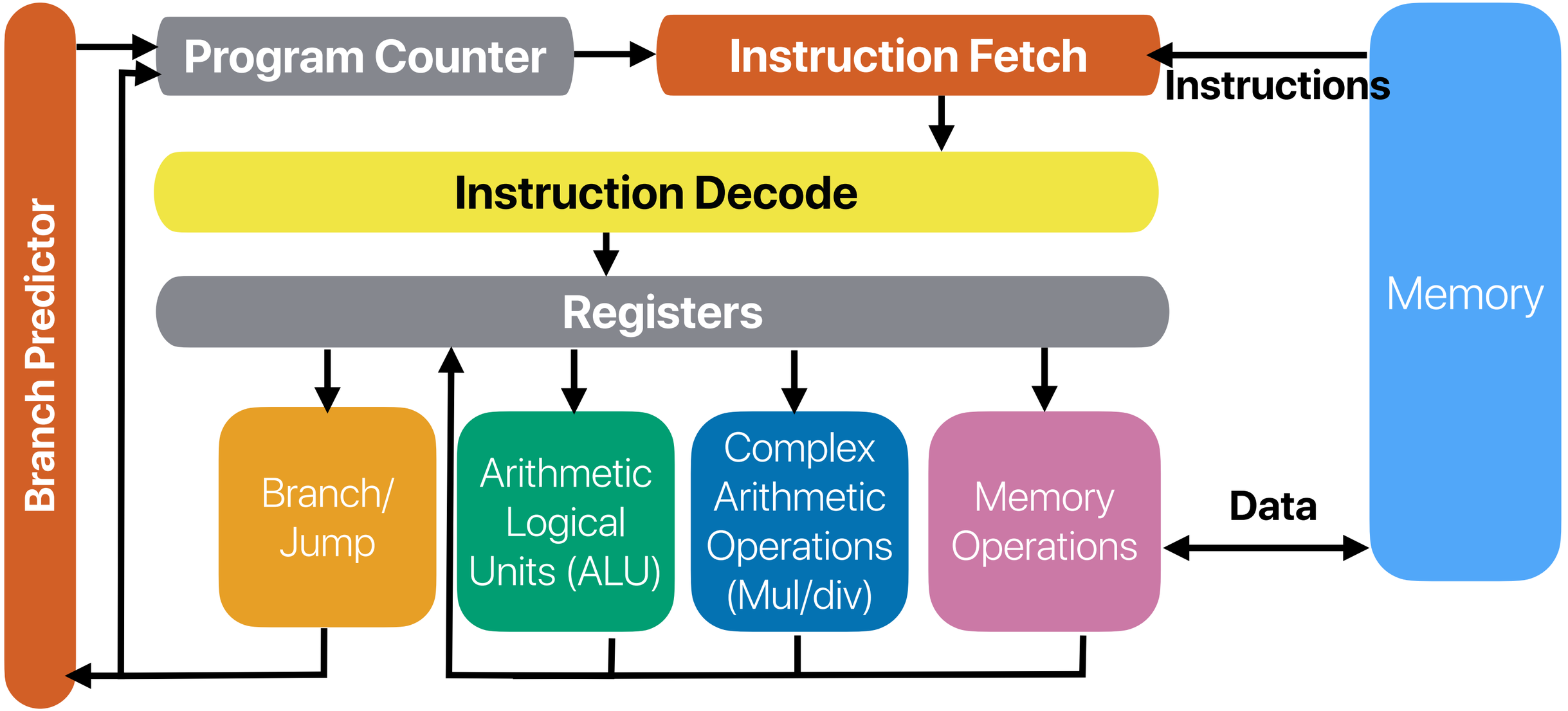
  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

You need a cheatsheet for that — branch target buffer

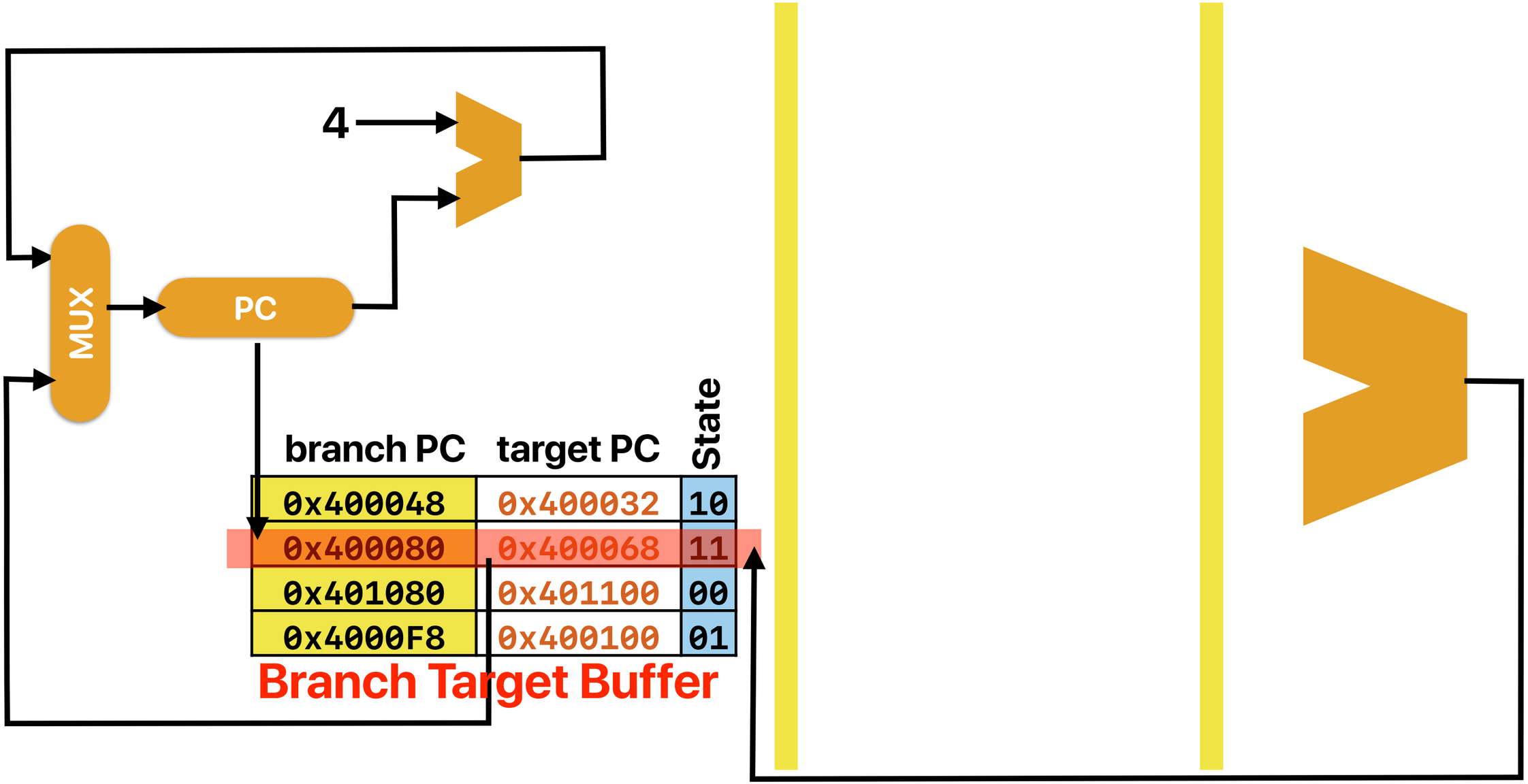
You need to predict that — history/states

Dynamic Branch Prediction

Microprocessor with a “branch predictor”



Detail of a basic dynamic branch predictor

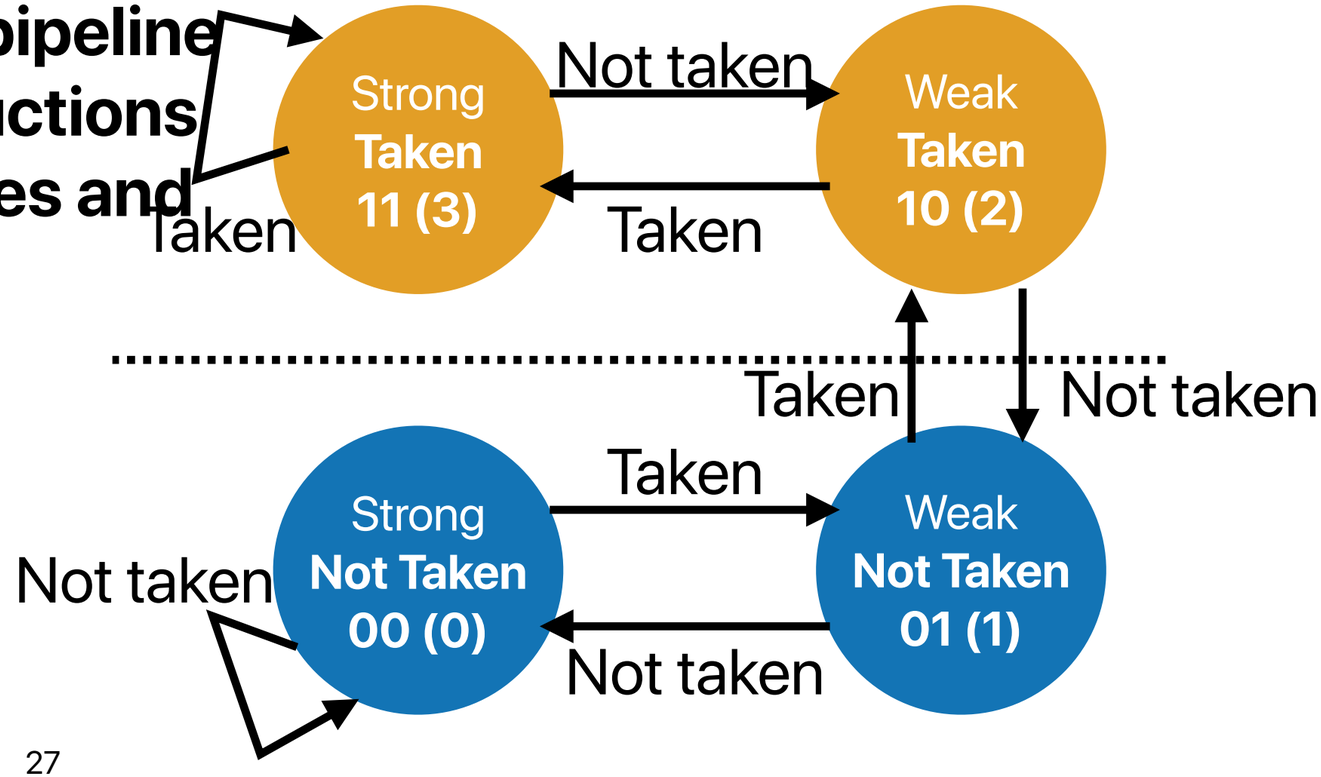


2-bit/Bimodal local predictor

- 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**

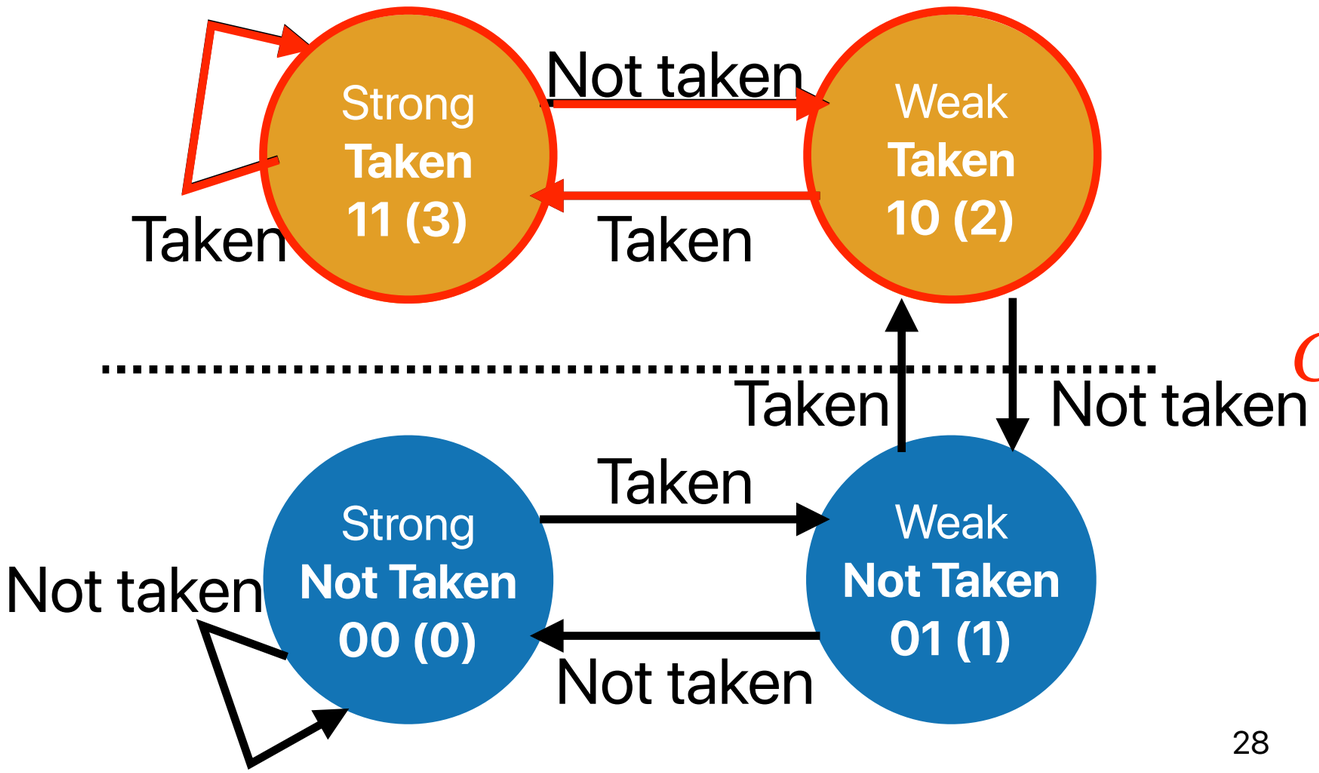
branch PC	target PC	State
0x400048	0x400032	10
0x400080	0x400068	11
0x401080	0x401100	00
0x4000F8	0x400100	01

Predict Taken



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!

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

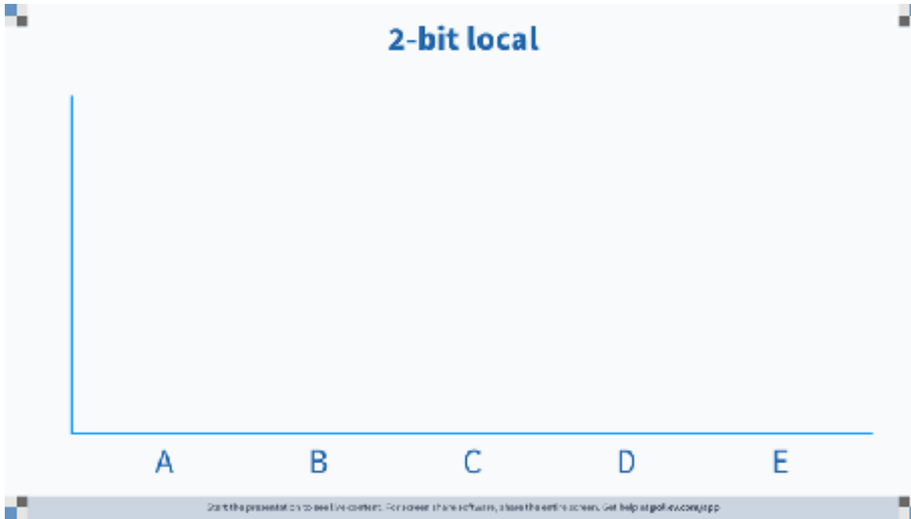
2-bit local predictor

- 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%

Can we do a better job?

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)

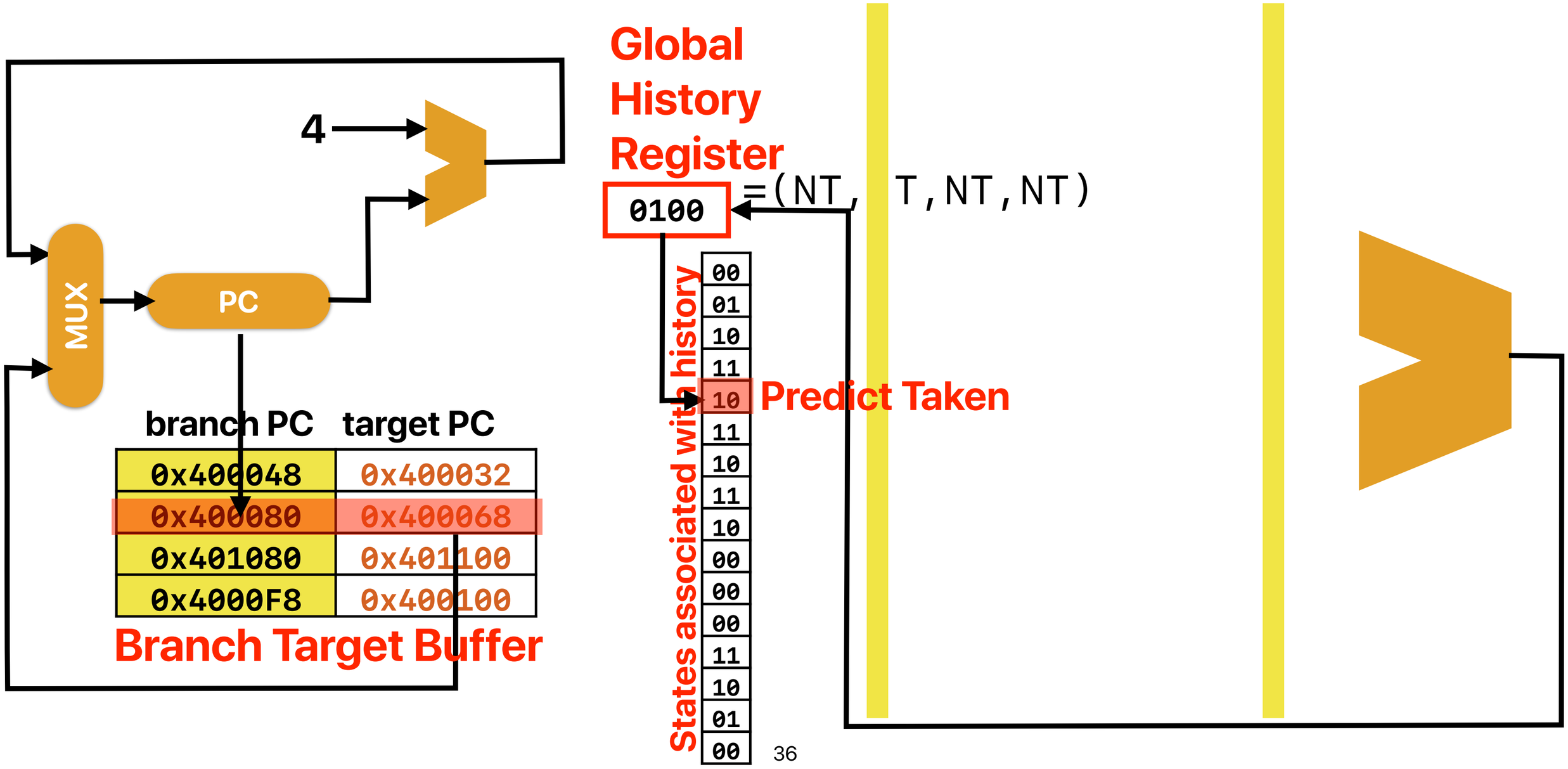
A. ~25%
B. ~33%
C. ~50%
D. ~67%
E. ~75%

This pattern repeats all the time!

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

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

Global history (GH) 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
```

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

Near perfect after this

Announcement

- Assignment 3 is already up
  - A total of 18 questions to answer
  - Also a programming assignment
  - Please do not expect any last minute help
- Pick up your midterm
  - Outside of WCH 406
  - Please respect others when you pick up
    - Don’t mess up the order
    - Don’t disturb people who are working in this area

Computer
Science &
Engineering

203

つづく