Dynamic Branch Prediction

Hung-Wei Tseng

Outline

- The cost of control hazards
- Dynamic branch predictions
  - Local predictor — 2 bit
  - Global predictor — 2-level
  - Hybrid predictors
    - Tournament
    - Perceptron

Control Hazards

How does the code look like?

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

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

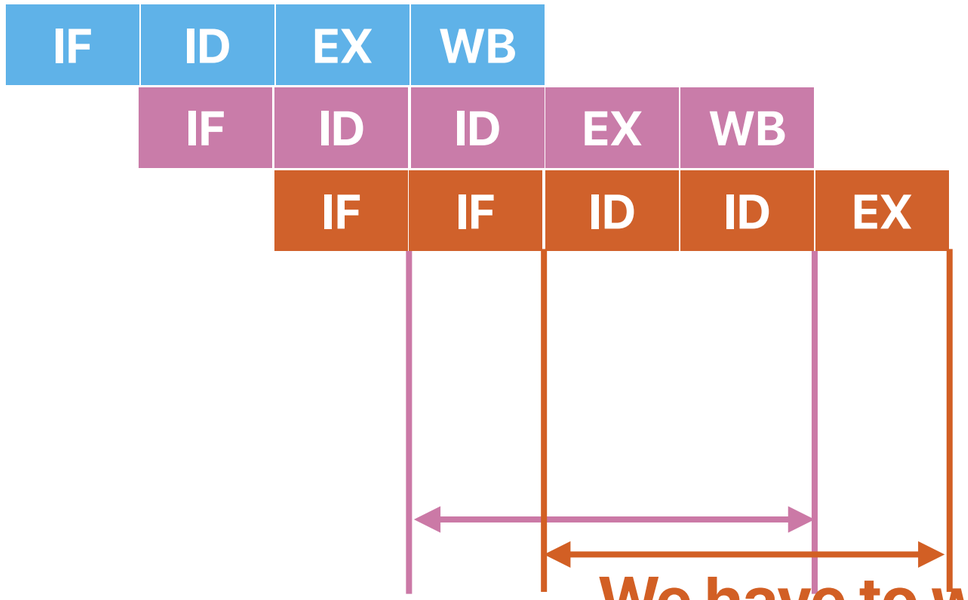
```
.LFB16:
    endbr64
    testl %esi, %esi
    jle   .L10
    movslq   %esi, %rsi
    pushq %r12
    leaq  (%rdi,%rsi,8), %r12
    pushq %rbp
    movslq   %edx, %rbp
    pushq %rbx
    movq  %rdi, %rbx
    jmp   .L5
    .p2align 4,,10
    .p2align 3
.L15:
    call  call_when_true@PLT
    addq  $8, %rbx
    cmpq  %r12, %rbx
    je .L14
.L5:
    movq  %rbx, %rdi
    cmpq  %rbp, (%rbx)
    jl .L15
    call  call_when_false@PLT
    addq  $8, %rbx
    cmpq  %r12, %rbx
    jne   .L5
.L14:
    popq  %rbx
    xorl  %eax, %eax
    popq  %rbp
    popq  %r12
    ret
```

Branch taken

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

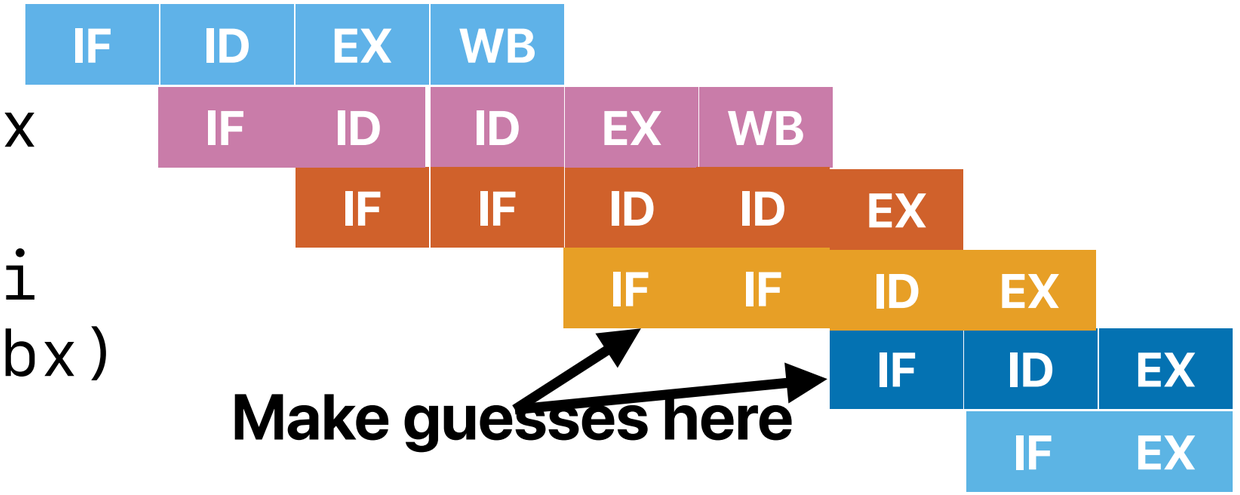
We have to wait almost as long as the latency of the previous instruction to make a decision — we cannot fetch anything before that

Takeaways: branch predictions

- The cost of not to predict a branch is to stall until the data dependency is resolved

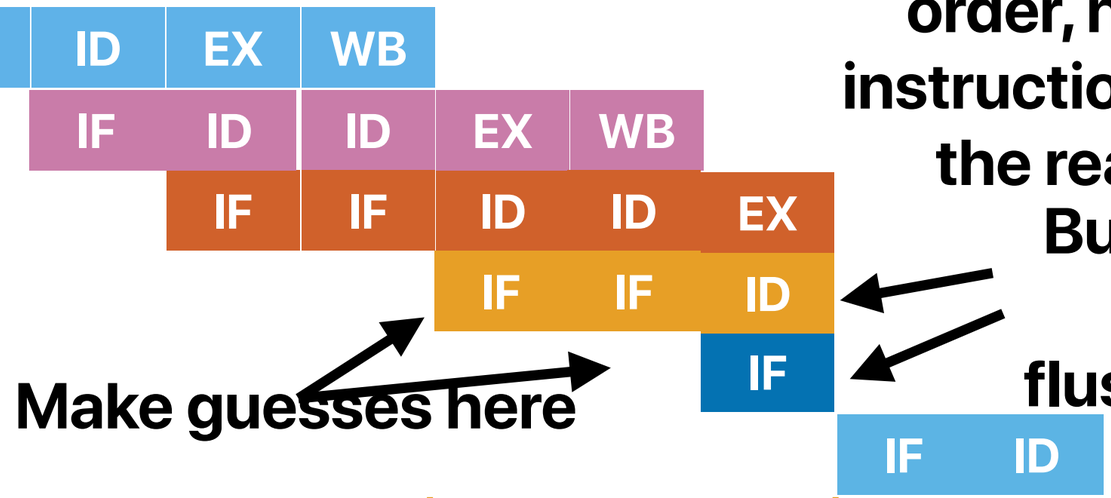
Prediction: What if we guessed right?

```
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
```



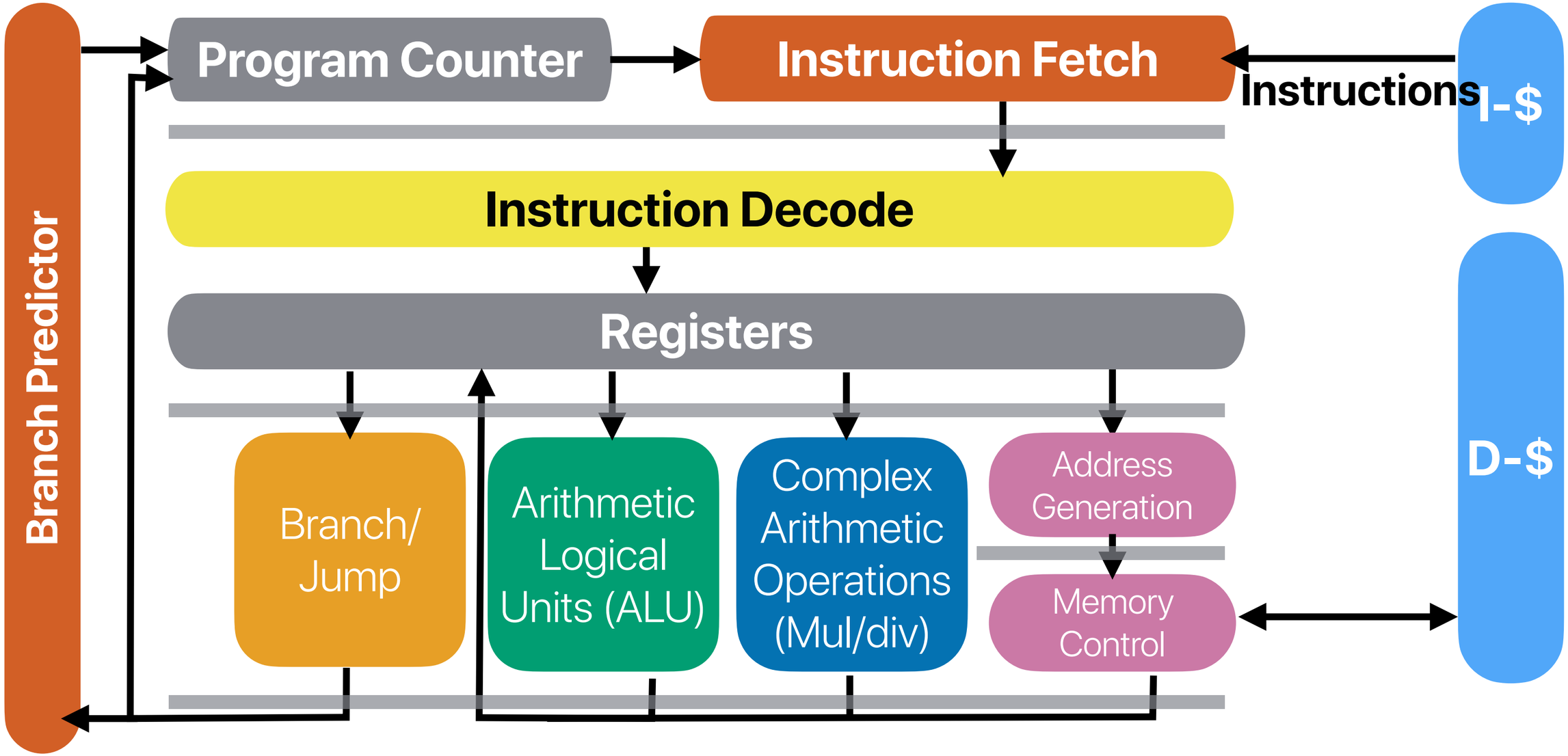
Make guesses here

Since we execute in-order, no predicted instructions will modify the real registers

Bubble the rest stages by flushing pipeline registers

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

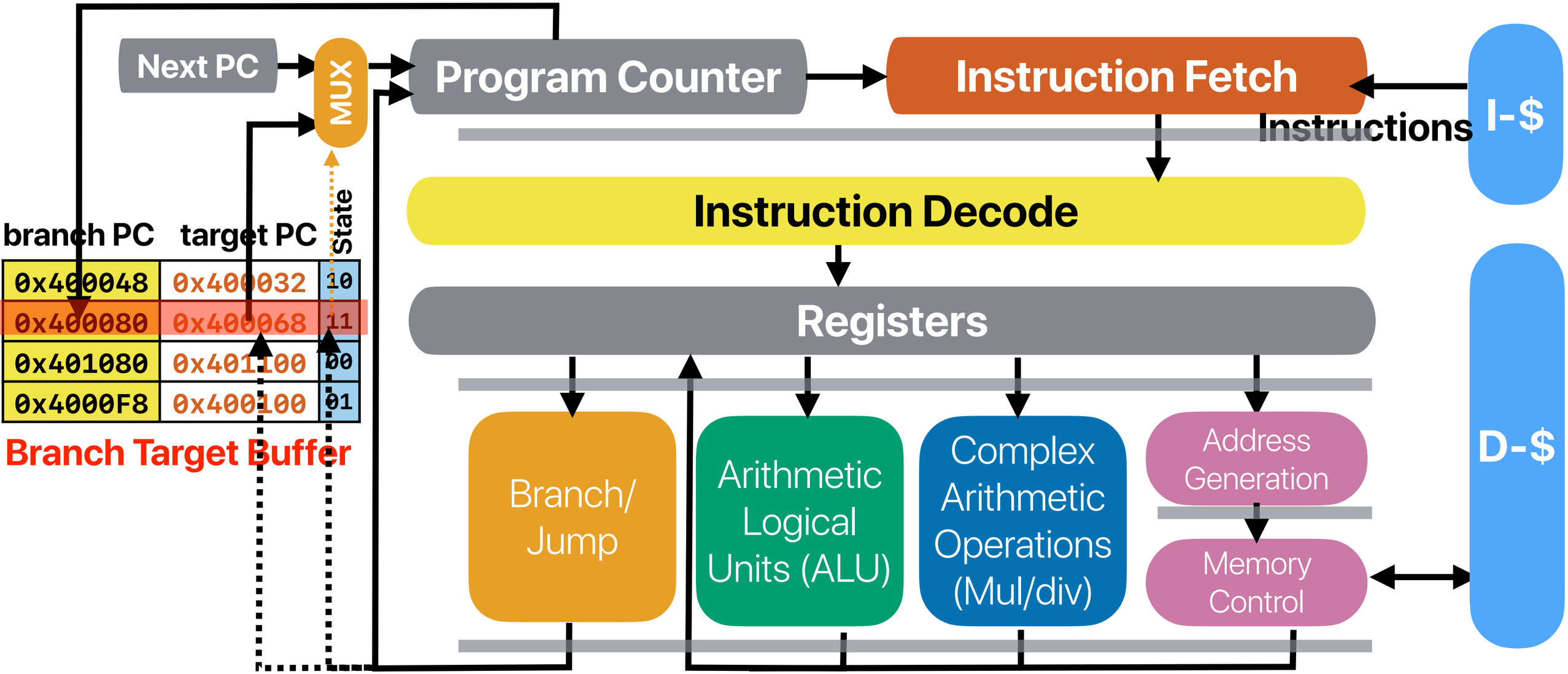
Microprocessor with a “branch predictor”



Takeaways: branch predictions

- The cost of not to predict a branch is to stall until the data dependency is resolved
- Branch predictions allow the processor to at least make some progress and hide the stalls if we guessed correctly!

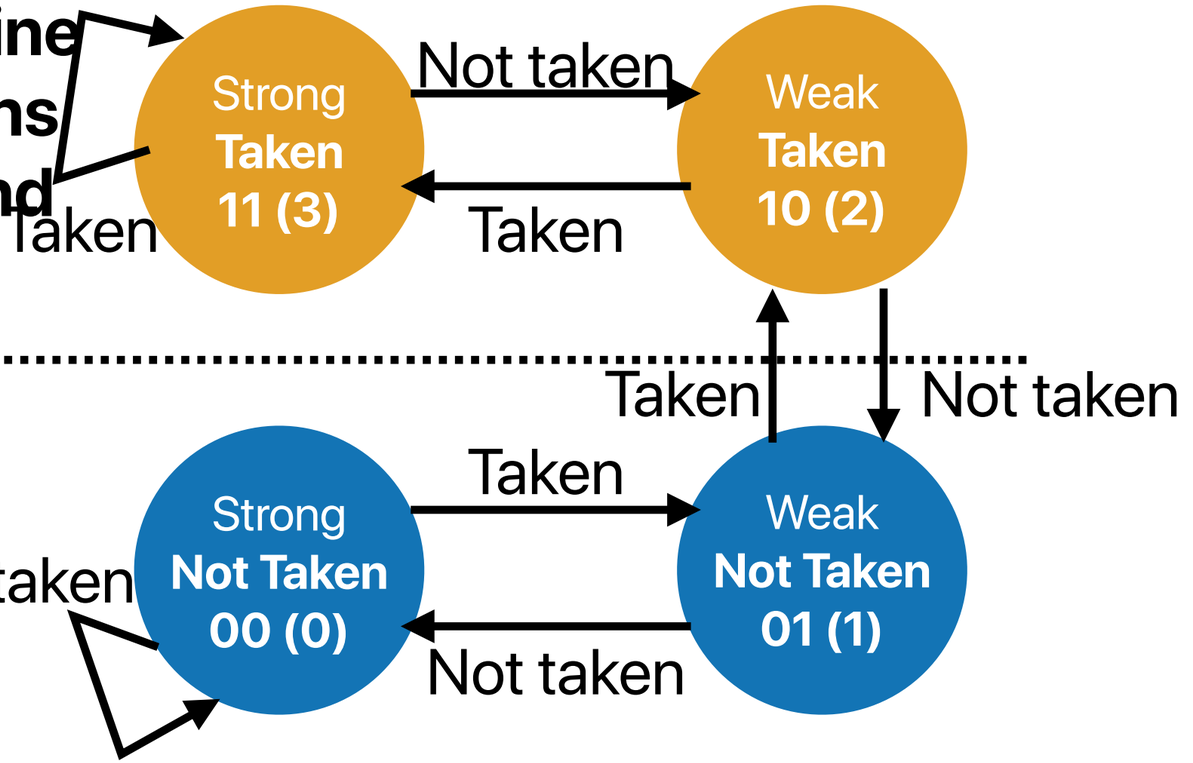
Detail of a basic dynamic branch predictor



2-bit/Bimodal local predictor

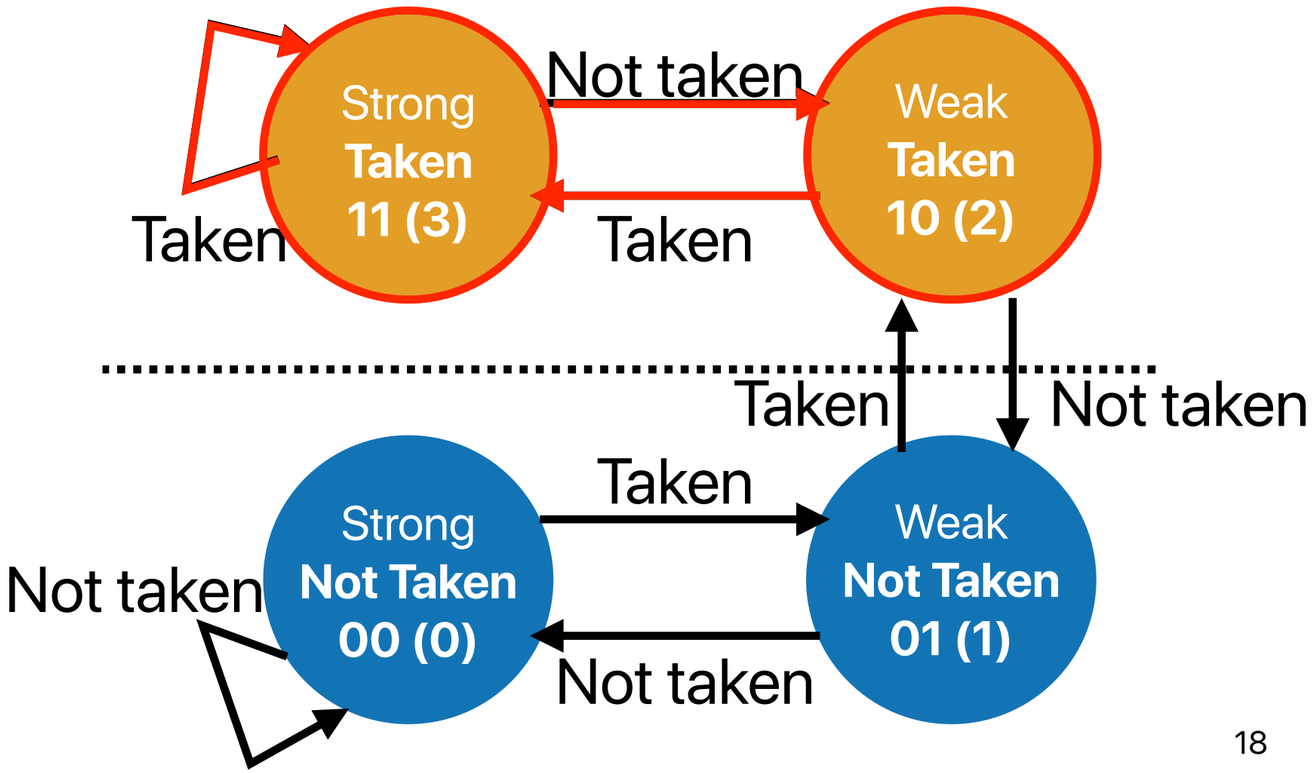
- Local predictor — every branch instruction has its own state
- 2-bit — each state is described using 2 bits
- Change the state based on **actual** outcome
- If we guess right — no penalty
- **If we guess wrong — flush (clear pipeline registers) for mis-predicted instructions that are currently in IF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

	branch PC	target PC	State
	0x400048	0x400032	10
Predict Taken	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!

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!!!

Takeaways: branch predictions

- The cost of not to predict a branch is to stall until the data dependency is resolved
  - 34 cycles on modern intel processors!
- Branch predictions allow the processor to at least make some progress and hide the stalls if we guessed correctly!
- Dynamic branch prediction — predict based on prior history
  - Local predictor — make prediction based on the state of each branch instruction

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)

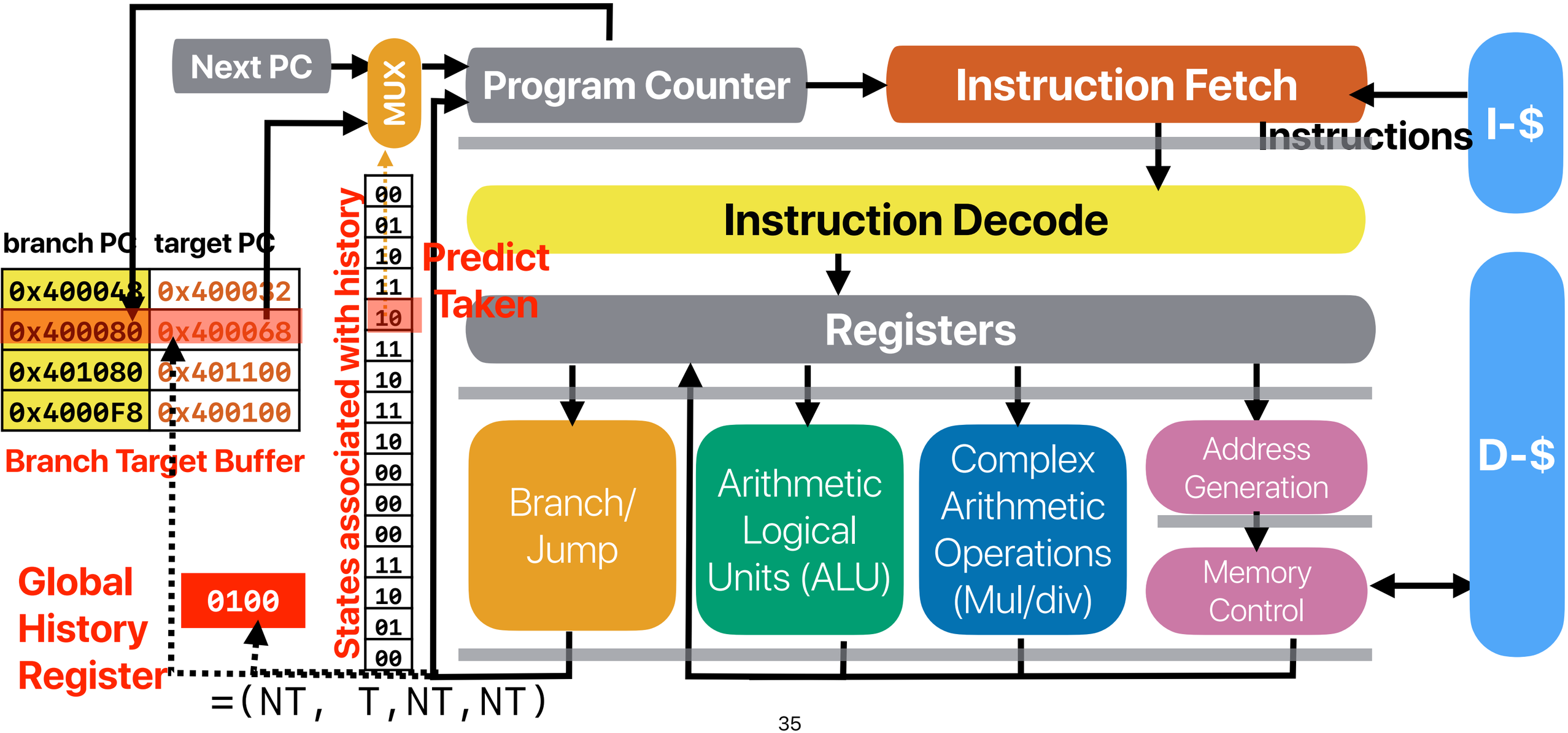
A. ~25%
B. ~33%
C. ~50%
D. ~67%
E. ~75%

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
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

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
```

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

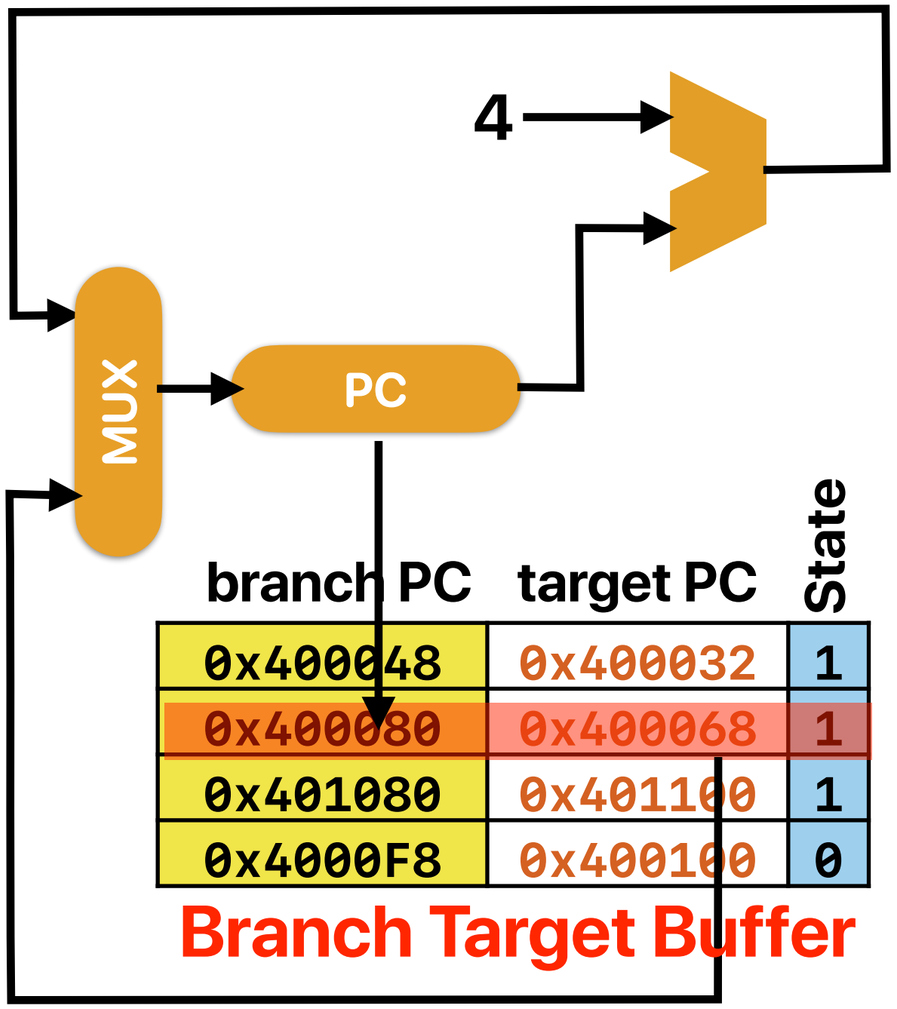
Near perfect after this

Takeaways: branch predictions

- The cost of not to predict a branch is to stall until the data dependency is resolved
  - 34 cycles on modern intel processors!
- Branch predictions allow the processor to at least make some progress and hide the stalls if we guessed correctly!
- Dynamic branch prediction — predict based on prior history
  - Local predictor — make predictions based on the state of each branch instruction
  - Global predictor — make predictions based on the state from all branches
  - Both are not perfect

Hybrid predictors

Tournament Predictor



Global History Register

0100

States associated with history

00
01
10
11
10
11
10
11
10
00
00
00
11
10
01
00

Local History Predictor

branch PC	local history
0x400048	1000
0x400080	0110
0x401080	1010
0x4000F8	0110

Predict Taken

States associated with history

00
01
10
11
10
11
10
11
10
00
00
00
11
10
01
00

Tournament Predictor

- The state predicts “which predictor is better”
  - Local history
  - Global history
- The predicted predictor makes the prediction
- Tournament predictor is a “hybrid predictor” as it takes both local & global information into account

Perceptron

Jiménez, Daniel, and Calvin Lin. "Dynamic branch prediction with perceptrons." Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture. IEEE, 2001.

The following slides are excerpted from https://www.jilp.org/cbp/Daniel-slides.PDF by Daniel Jiménez

Branch Prediction is Essentially an ML Problem

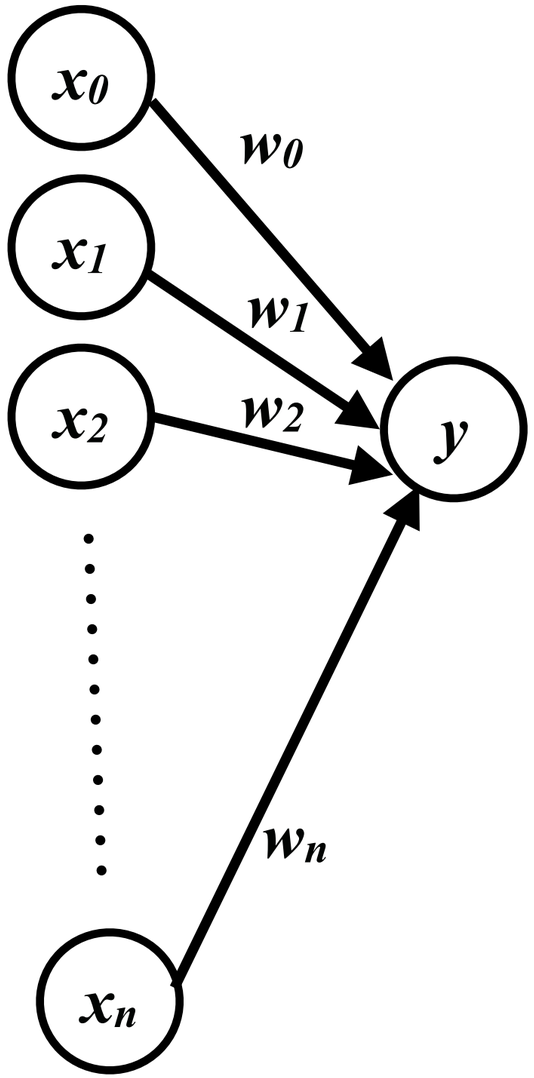
- The machine learns to predict conditional branches
- Artificial neural networks
  - Simple model of neural networks in brain cells
  - Learn to recognize and classify patterns

Mapping Branch Prediction to NN

- The inputs to the perceptron are branch outcome histories
  - Just like in 2-level adaptive branch prediction
  - Can be global or local (per-branch) or both (alloyed)
  - Conceptually, branch outcomes are represented as
    - +1, for taken
    - -1, for not taken
- The output of the perceptron is
  - Non-negative, if the branch is predicted taken
  - Negative, if the branch is predicted not taken
- Ideally, each static branch is allocated its own perceptron

Mapping Branch Prediction to NN (cont.)

- Inputs (x's) are from branch history and are -1 or +1
- n + 1 small integer weights (w's) learned by on-line training
- Output (y) is dot product of x's and w's; predict taken if y = 0
- Training finds correlations between history and outcome



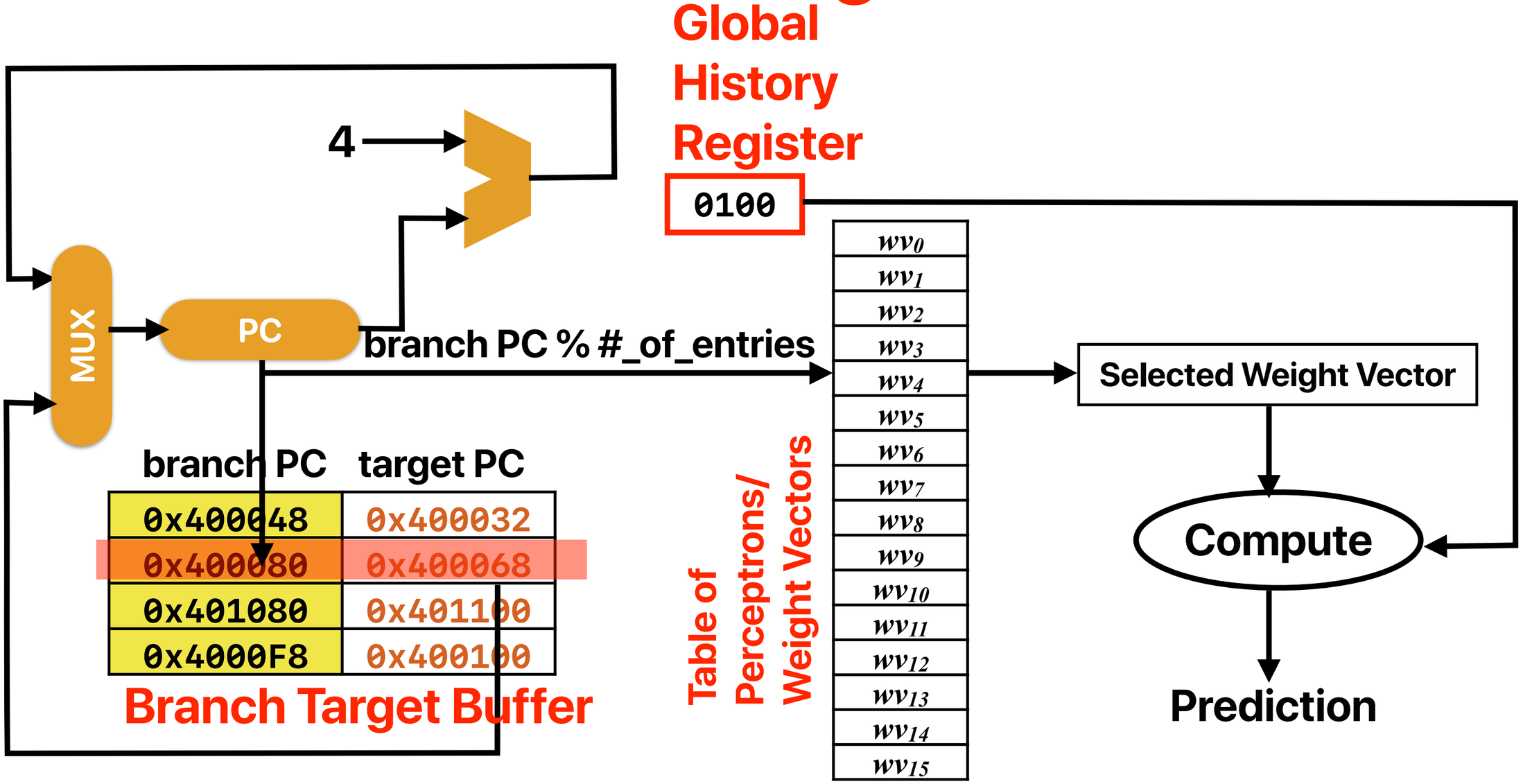
$$y = w_0 + \sum_{i=1}^{n} x_i w_i$$

Training Algorithm

$x_{1..n}$ is the n-bit history register, x_0 is 1.
$w_{0..n}$ is the weights vector.
t is the Boolean branch outcome.
θ is the training threshold.

```
if |y| ≤ θ or ((y ≥ 0) ≠ t) then
    for each 0 ≤ i ≤ n in parallel
        if t = x_i then
            w_i := w_i + 1
        else
            w_i := w_i − 1
        end if
    end for
end if
```

Predictor Organization



Branch predictors in processors

- The Intel Pentium MMX, Pentium II, and Pentium III have local branch predictors with a local 4-bit history and a local pattern history table with 16 entries for each conditional jump.
- Global branch prediction is used in Intel Pentium M, Core, Core 2, and Silvermont-based Atom processors.
- Tournament predictor is used in DEC Alpha, AMD Athlon processors
- The AMD Ryzen multi-core processor's Infinity Fabric and the Samsung Exynos processor include a perceptron based neural branch predictor.

Takeaways: branch predictions

- The cost of not to predict a branch is to stall until the data dependency is resolved
  - 34 cycles on modern intel processors!
- Branch predictions allow the processor to at least make some progress and hide the stalls if we guessed correctly!
- Dynamic branch prediction — predict based on prior history
  - Local predictor — make predictions based on the state of each branch instruction
  - Global predictor — make predictions based on the state from all branches
  - Both are not perfect — hybrid predictors
    - Tournament
    - Perceptron
  - All modern processors have branch predictors!