CPSC 313: Computer Hardware and Operating Systems

Unit 2: Pipelining

Pipelining: Branch Prediction

Administration

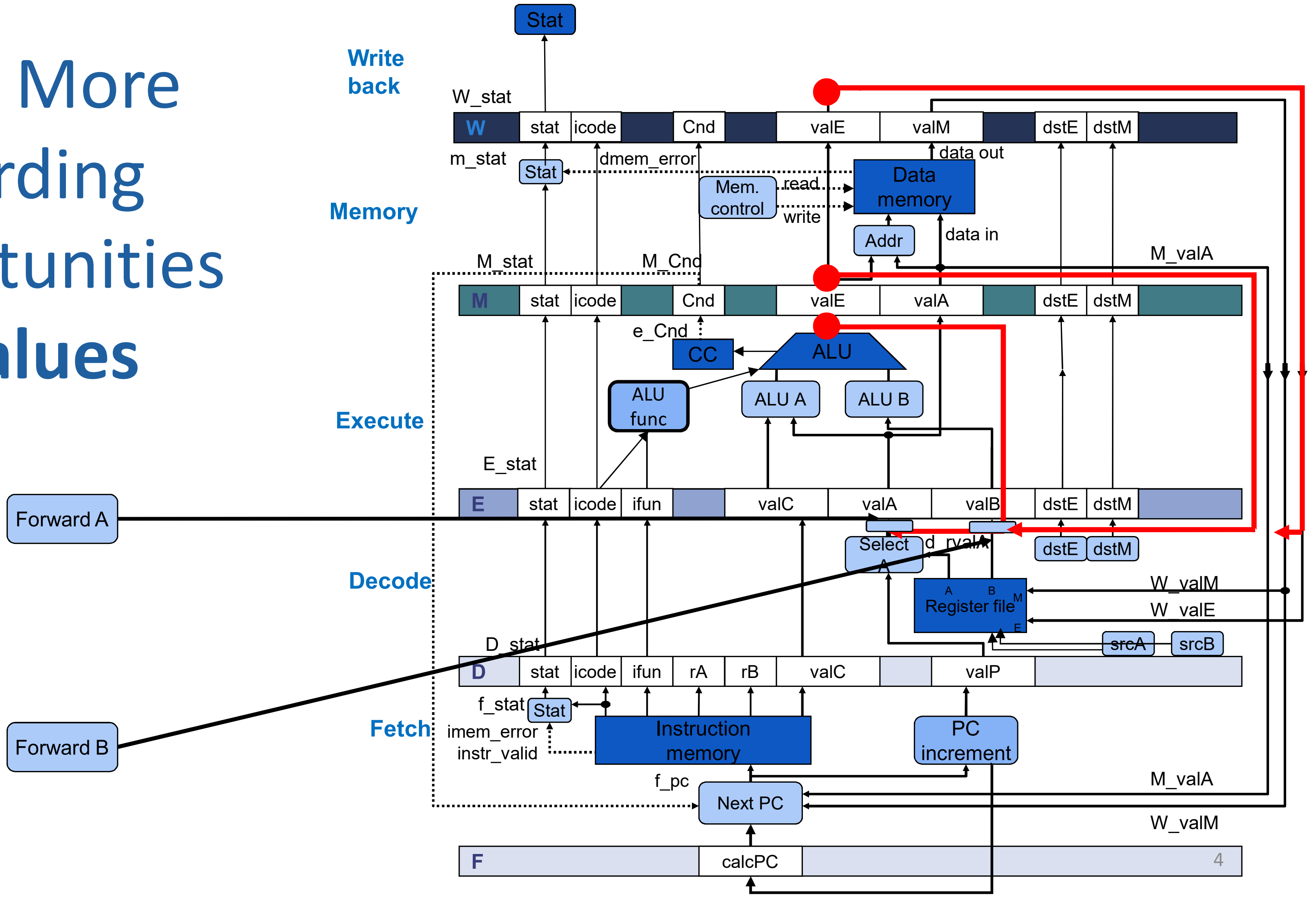
- Quiz 2: Register & check out Info/Practice
- Lab 4: Is out!

> As always: See the Syllabus on Canvas for deadlines.

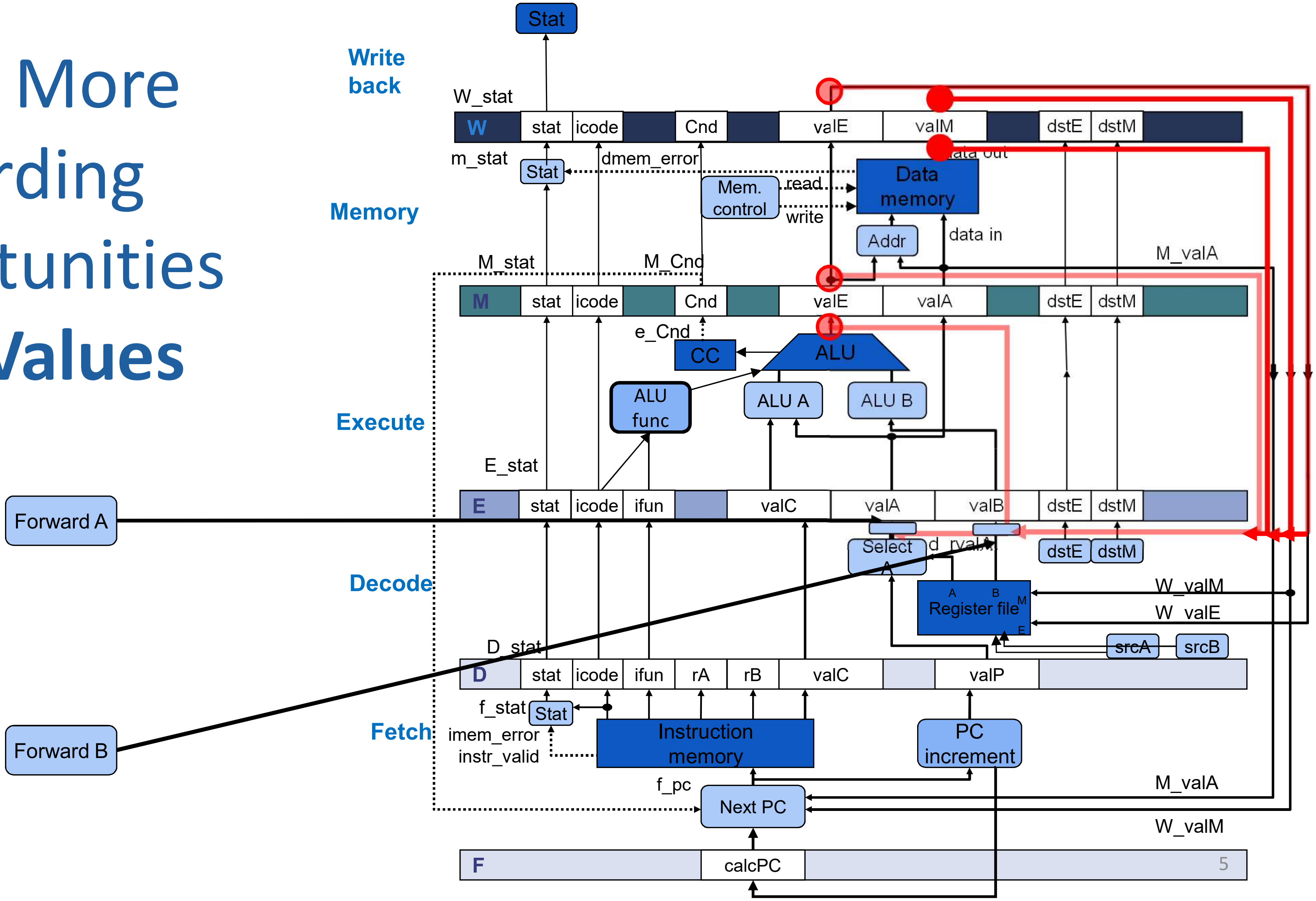
Today

- Learning outcomes
  - Describe what branch prediction is
  - Enumerate different branch prediction algorithms
  - Exploit branch prediction to improve CPI

Recall: More Forwarding Opportunities **ALU values**



Recall: More Forwarding Opportunities **Mem Values**



Dealing with Control Hazards

- `jmp` and `call`: we know where to go in Fetch!
- `ret`: we have to stall for three cycles ☹
- `jxx`: we… guess!

Dealing with Control Hazards

- `jmp` and `call`: we know where to go in Fetch!
- `ret`: we have to stall for three cycles ☹
- `jxx`: use branch prediction (guess) for conditional jumps:
  - Always taken (assume that the nextPC is going to be valC)
  - Never taken (assume that the nextPC is going to be valP)
  - Or fancier things, but how should we decide which is correct? Let’s try some examples.

Branch Prediction

- Let’s convert these two code snippets into y86

```
j = 1;               if (p == NULL) {
do {                     //handle error
    j++;                 return -1;
} while j < 10;      }
```

Branch Prediction

- Let’s convert these two code snippets into y86

```
j = 1;
do {
    j++;
} while j < 10;
```

```
if (p == NULL) {
    //handle error
    return -1;
}
```

```
# Let’s assume j is in %rsi
        irmovq 1, %rax          # incr = 1
        irmovq 1, %rsi          # j = 1

Loop:   addq %rax, %rsi
        irmovq 10, %rdi         # rdi = 10
        subq %rsi, %rdi         # set CC
        jg Loop
```

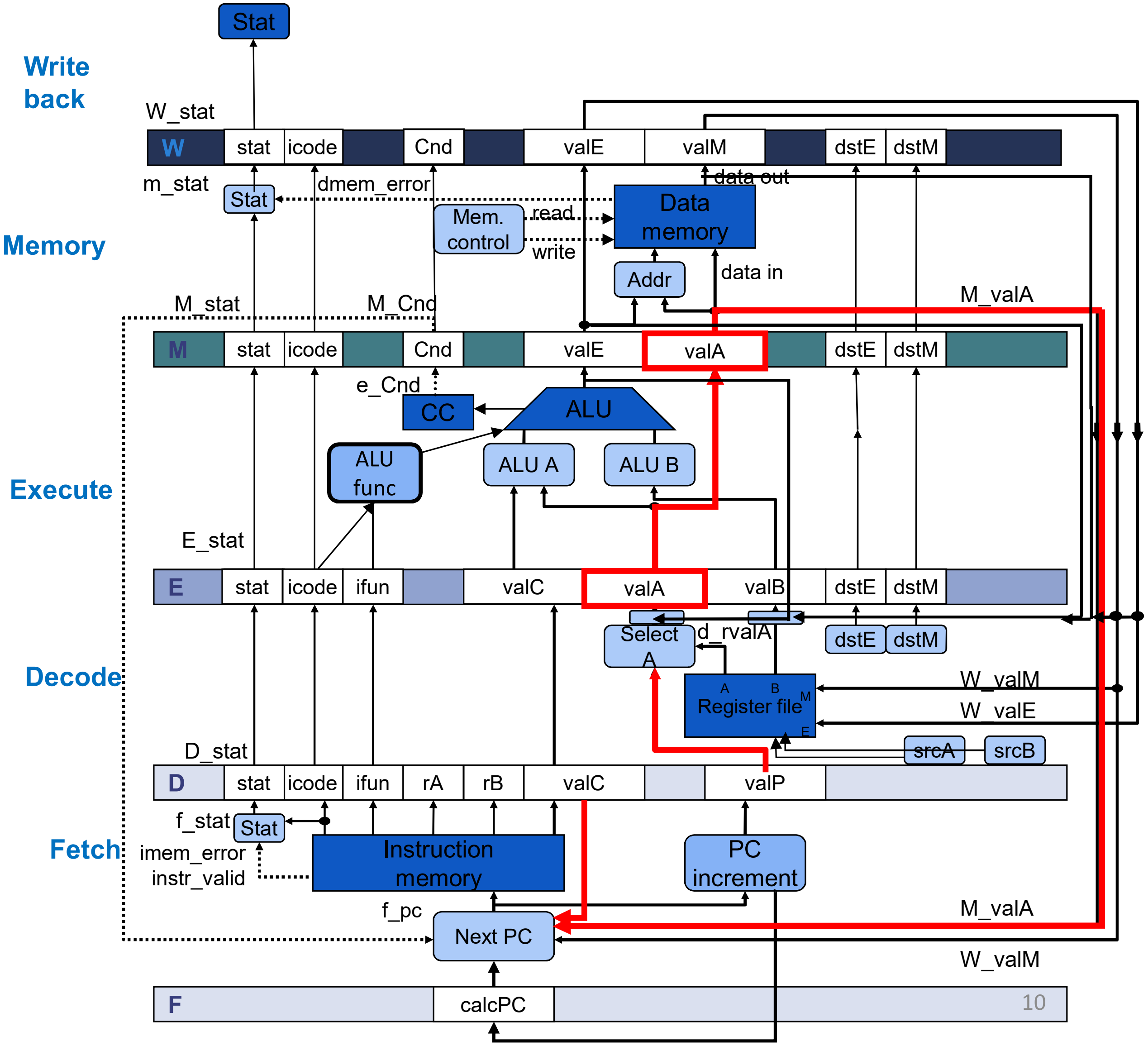
```
# Let’s assume p is in %rbp
# and is already assigned a value
        andq %rbp, %rbp
        je error
        # The rest of the function
        # goes here

error:  # Handle the 'if' clause
        irmovq  0xFFFFFFFFFFFFFFFF, %rax
        ret
```

Branch Prediction: Always Taken

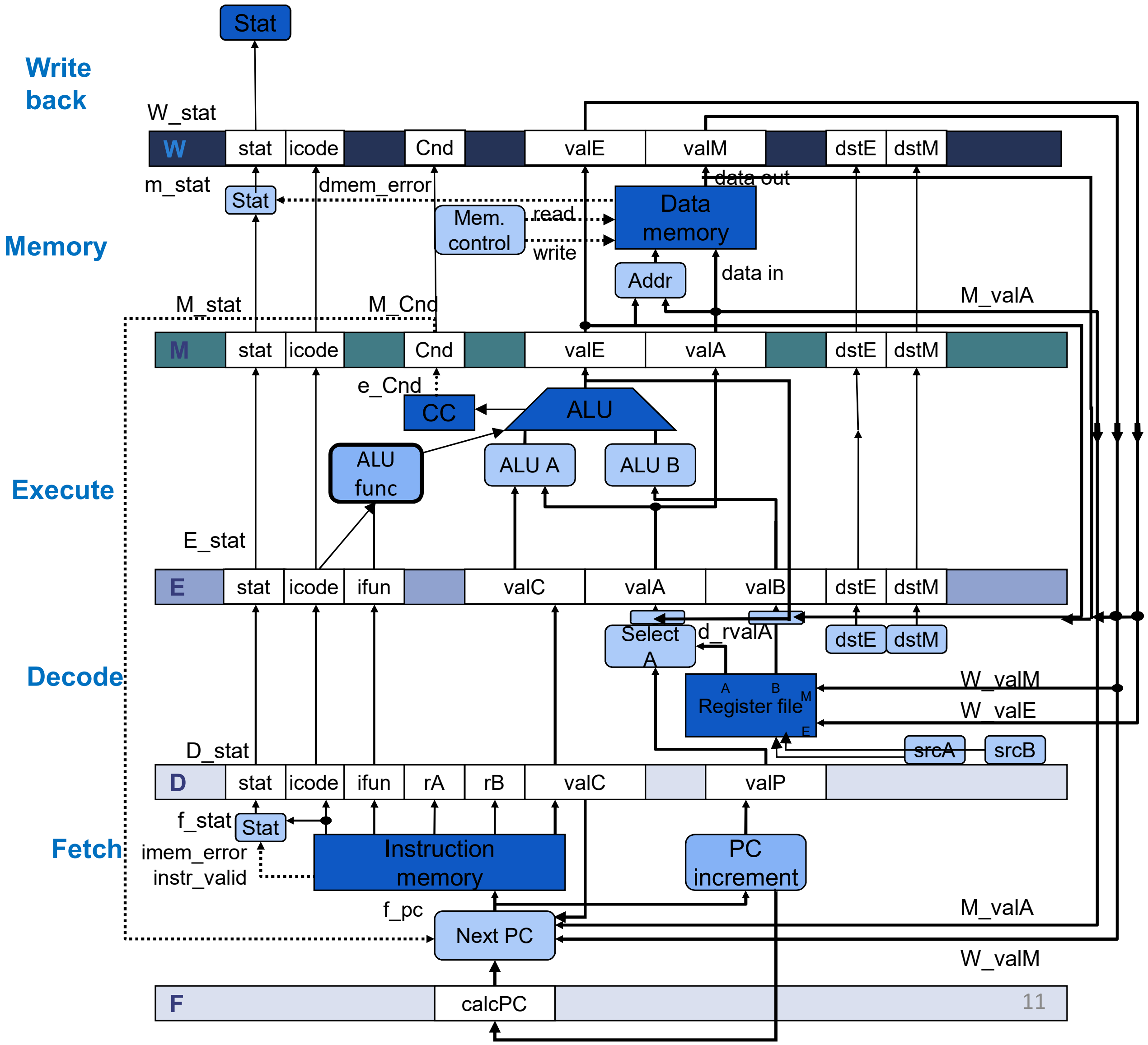
Do we have what we need where we need it…

To predict always taken?

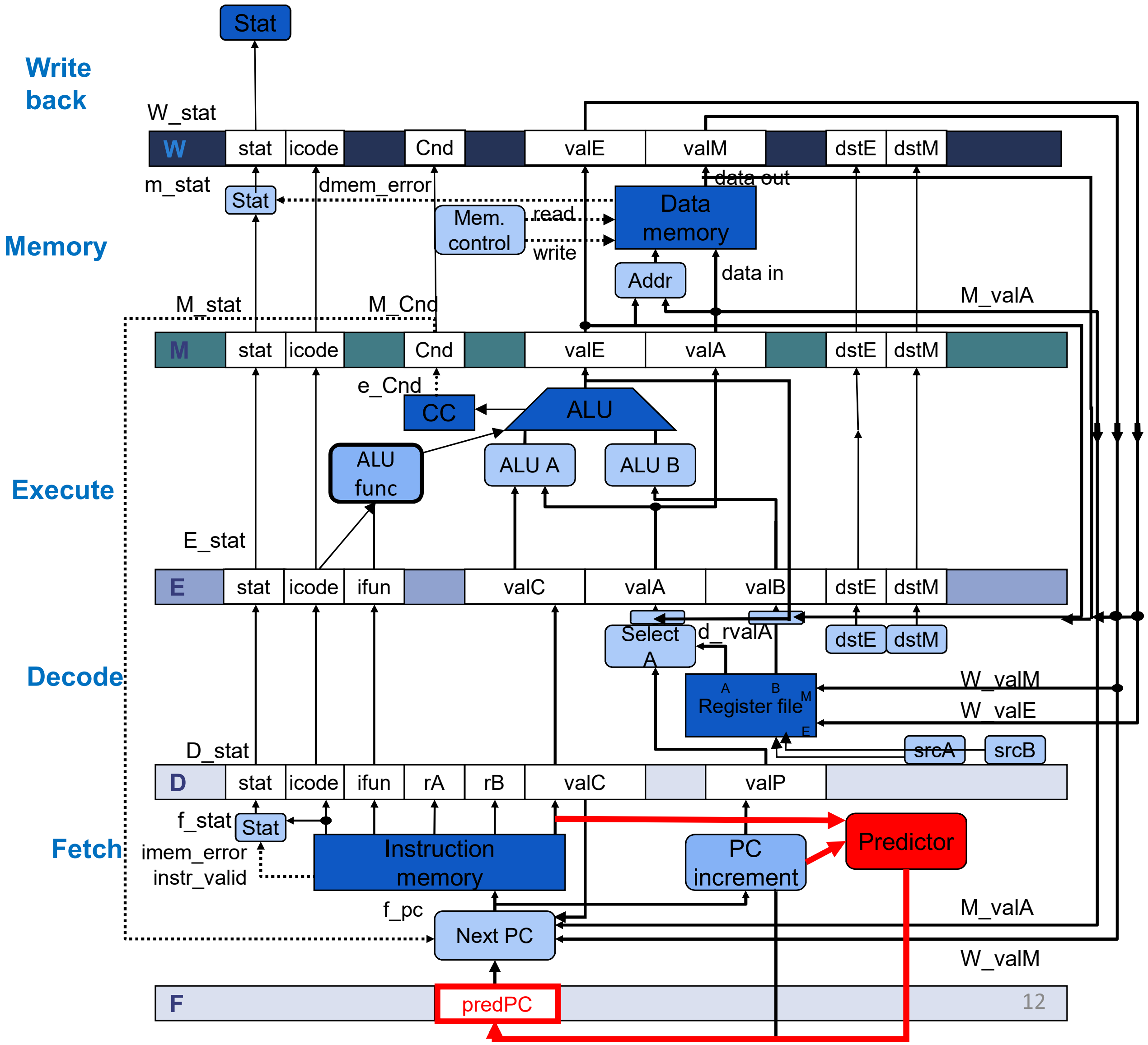
If we mispredict?



Branch Prediction: More Complex Strategies

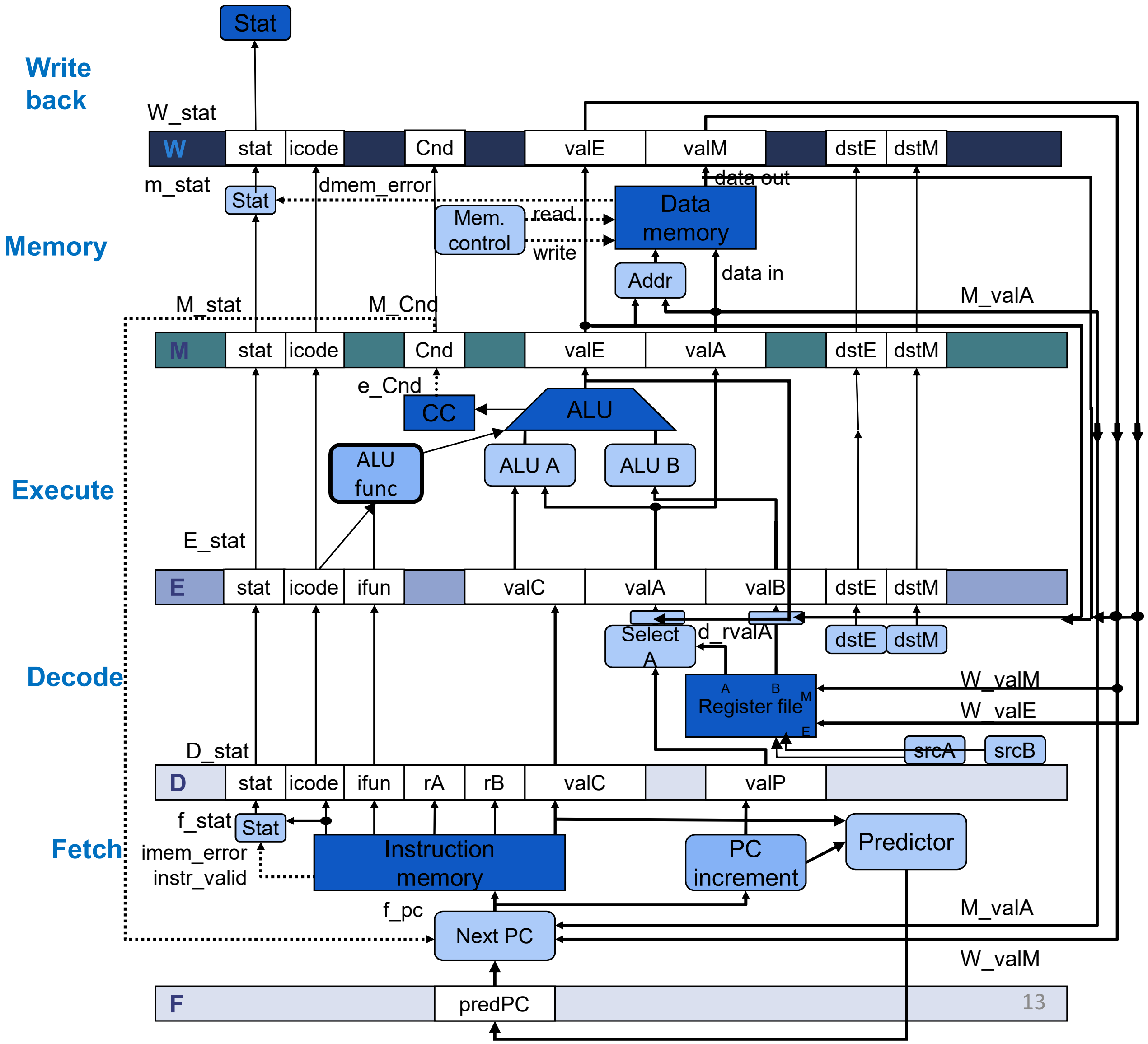


Branch Prediction: More Complex Strategies



Branch Prediction: More Complex Strategies

Do we have what we need on a mispredict?



Fun Facts about Branch Prediction

- Always taken typically has about a 60% success rate
- Never taken typically has about a 40% success rate
- Some processors do:
  - Backward taken; forward not taken (65% success rate)
- Some processors devote lots of fancy hardware to branch prediction
  - 1 bit: do whatever you did last time
  - More bits for more complicated schemes
  - Entries for branches at different values of the PC
- Other processors let you execute instructions while you're waiting to determine what's going to happen with your jump (these instructions occupy what are known as *branch delay slots*).

Some Recent Research on Branch Prediction

Half&Half: Demystifying Intel’s Directional Branch Predictors for Fast, Secure Partitioned Execution
by Hosein Yavarzadeh et al.

> https://ieeexplore.ieee.org/document/10179415/

"This work presents the first exhaustive analysis of modern conditional branch prediction structures"

Steering Branch Prediction

- You can give *hints*
- Linux has two macros:

```
  #define likely(x)       __builtin_expect(!!(x), 1)
• #define unlikely(x)     __builtin_expect(!!(x), 0)

 For example:

 if (unlikely(x))
    foo ();

 would indicate that we do not expect to call
 `foo', since we expect `x' to be zero.
```

Results may wary...

The Problem with Branch Prediction

- What happens if you are wrong?
- Consider:

```
  irmovq 0x1, %rax
  irmovq 0x1, %rbx
  subq %rax, %rbx
  je skip
  irmovq 0x5000, %rcx
  irmovq 0x5000, %rdx
  # more instructions here
skip:
  addq %rcx, %rbx
  mulq %rbx, %rdx
```

Let’s assume that our prediction algorithm is “never taken”

Mis-Prediction (1)

- What happens if you are wrong?
- Consider:

```
    irmovq 0x1, %rax
    irmovq 0x1, %rbx
    subq %rax, %rbx
    je skip
    irmovq 0x5000, %rcx
    irmovq 0x5000, %rdx
    # more instructions here
skip:
    addq %rcx, %rbx
    mulq %rbx, %rdx
```

Let’s assume that our prediction algorithm is “never taken”

You have two bad instructions in the pipeline!

- That is, instructions you should not have executed!
- So what do you do???

Mis-Prediction (2)

- What happens if you are wrong?
- Consider:

```
    irmovq 0x1, %rax
    irmovq 0x1, %rbx
    subq %rax, %rbx
    je skip
    irmovq 0x5000, %rcx
    irmovq 0x5000, %rdx
    # more instructions here
skip:
    addq %rcx, %rbx
    mulq %rbx, %rdx
```

Let’s assume that our prediction algorithm is “never taken”

You have two bad instructions in the pipeline!

- That is, instructions you should not have executed!
- So what do you do???

You cancel or squash or quash them!

So:

- In the best case, conditional branches incur no penalty
- In the worst case (where you mispredict), it’s no worse than if you’d done no prediction

Branch Prediction and CPI

- A good branch predictor can dramatically improve your CPI (cycles per instruction)
- We’ll give you lots of practice with that

In-class Exercise

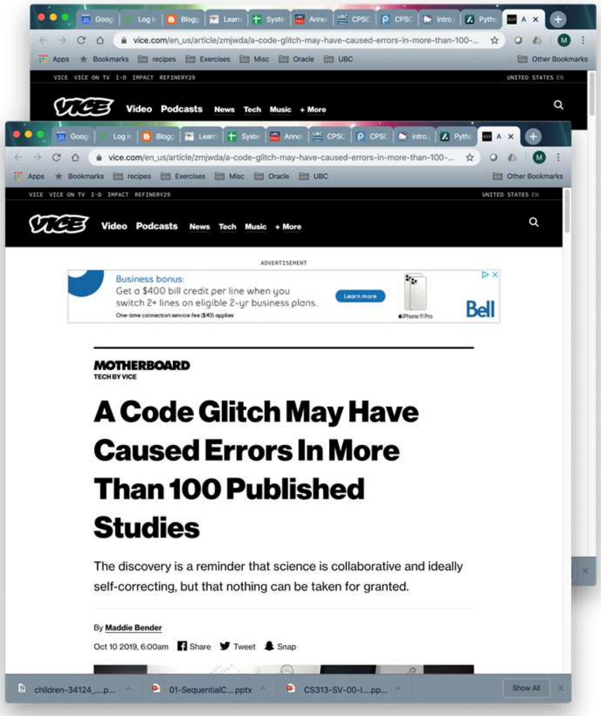
- Remember to do multiple instances of a problem if there is only one problem listed!
  - We guessed maybe 5, but it’s good to practice as many as you have time for!

Wrapping Up

- Branch prediction (tries to) avoid losing cycles on every conditional jump.
- A good predictor will reduce a program’s CPI.
- You should be able to:
  - Compute CPI for a simple program
  - Given an unoptimized program, rearrange it to improve CPI

- If you have not yet done the pre-class for next class, give yourself some time to write a fun problem! (It’s a great way to start to study for the midterm.)

Why We Love Systems



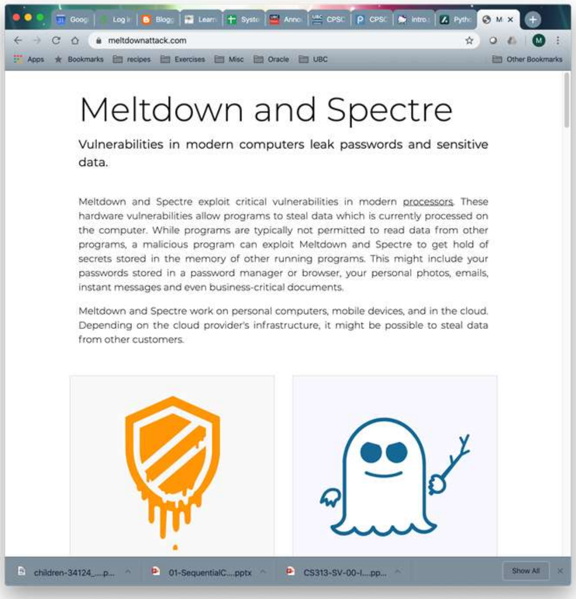
MOTHERBOARD
TECH BY VICE

A Code Glitch May Have Caused Errors In More Than 100 Published Studies

The discovery is a reminder that science is collaborative and ideally self-correcting, but that nothing can be taken for granted.

By Maddie Bender

Oct 10 2019, 6:00am Share Tweet Snap

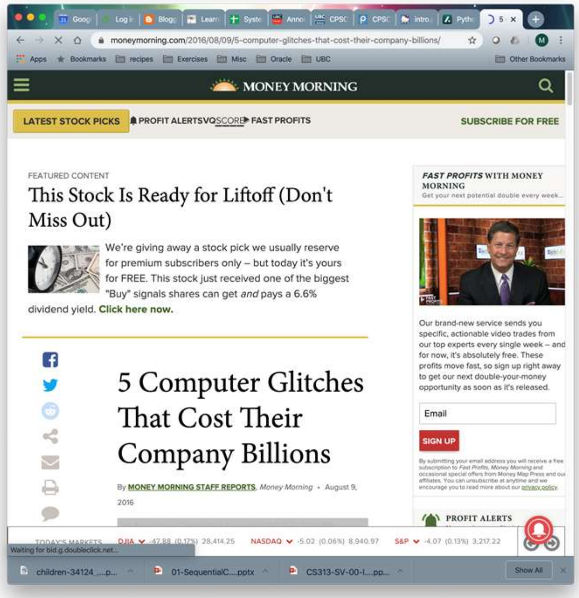


Meltdown and Spectre

Vulnerabilities in modern computers leak passwords and sensitive data.

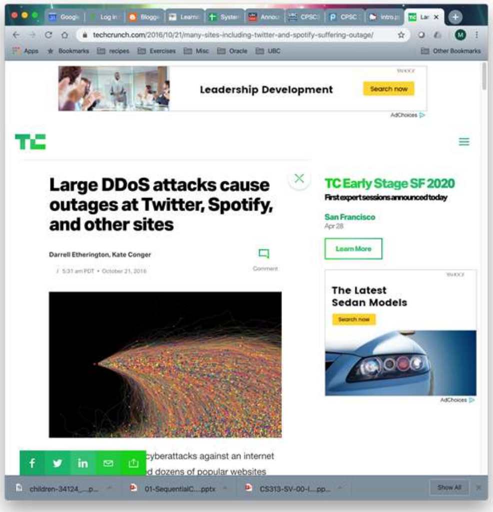
Meltdown and Spectre exploit critical vulnerabilities in modern processors. These hardware vulnerabilities allow programs to steal data which is currently processed on the computer. While programs are typically not permitted to read data from other programs, a malicious program can exploit Meltdown and Spectre to get hold of secrets stored in the memory of other running programs. This might include your passwords stored in a password manager or browser, your personal photos, emails, instant messages and even business-critical documents.

Meltdown and Spectre work on personal computers, mobile devices, and in the cloud. Depending on the cloud provider's infrastructure, it might be possible to steal data from other customers.



5 Computer Glitches That Cost Their Company Billions

By MONEY MORNING STAFF REPORTS, *Money Morning* • August 9, 2016



Large DDoS attacks cause outages at Twitter, Spotify, and other sites

Darrell Etherington, Kate Conger

5:31 am PDT • October 21, 2016

Comment

cyberattacks against an internet

d dozens of popular websites