Handling Control Hazards in Pipelined Alpha Processor

March 31, 1998

Topics

- **Branch & Jump Instructions**
  - Added data hazards
- **Handling Control Hazards**
  - Stall until resolve
  - Predict not taken

Branch Instructions

Cond. Branch: PC <-- Cond(Ra) ? PC + 4 + disp*4 : PC + 4

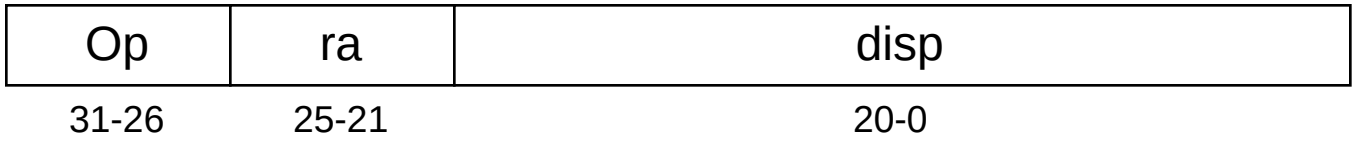
Sources

- **PC, Ra**

Destinations

- **PC**

Branch [Subroutine] (br, bsr): Ra <-- PC + 4; PC <-- PC + 4 + disp*4



Sources

- **PC**

Destinations

- **PC, Ra**

New Data Hazards

Branch Uses Register Data

- **Generated by ALU instruction**
- **Read from register in ID**

Handling

- **Same as other instructions with register data source**
- **Bypass**
  - EX-EX
  - MEM-EX

ALU-Branch

```
addq $2, $3, $1
beq $1, targ
```

Distant ALU-Branch

```
addq $2, $3, $1
bis $31, $31, $31
beq $1, targ
```

Load-Branch

```
lw $1, 8($2)
beq $1, targ
```

Jump Instructions

jmp, jsr, ret: Ra <-- PC+4; PC <-- Rb

0x1A	ra	rb	Hint
31-26	25-21	20-16	15-0

Sources

- **PC, Rb**

Destinations

- **PC, Ra**

Still More Data Hazards

Jump Uses Register Data

- **Generated by ALU instruction**
- **Read from register in ID**

Handling

- **Same as other instructions with register data source**
- **Bypass**
  - EX-EX
  - MEM-EX

ALU-Jump

```
addq $2, $3, $1
jsr $26 ($1), 1
```

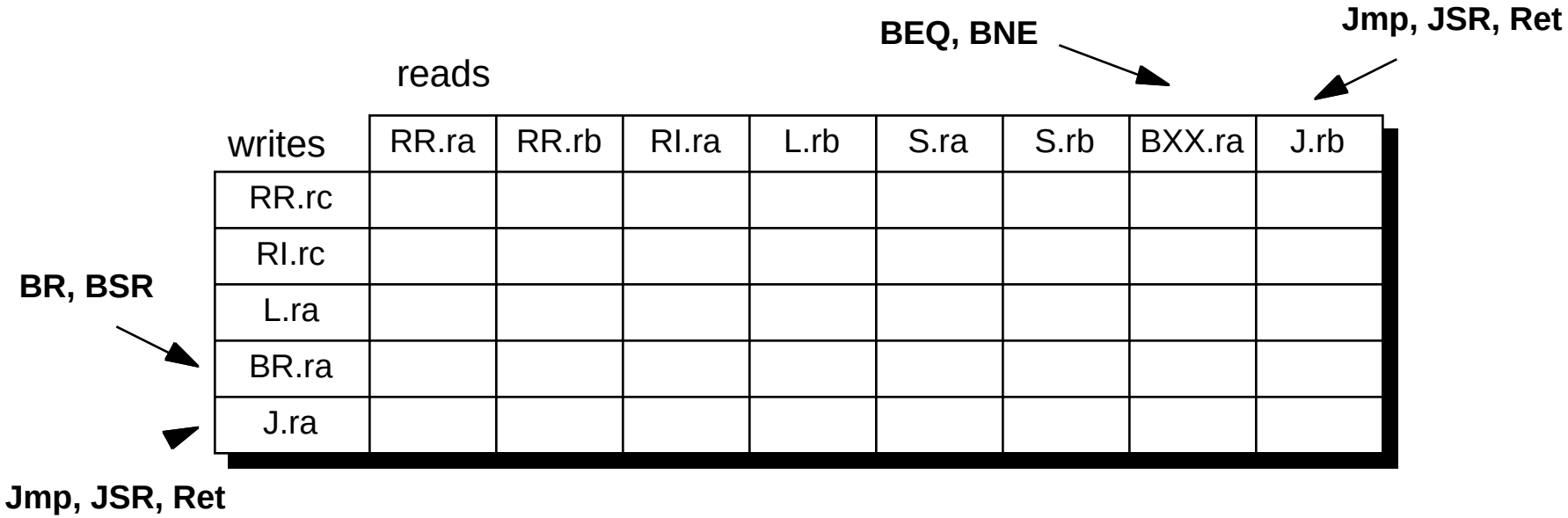
Distant ALU-Jump

```
addq $2, $3, $1
bis $31, $31, $31
jmp $31 ($1), 1
```

Load-Jump

```
lw $26, 8($sp)
ret $31 ($26), 1
```

Enumerating data hazards



Cases

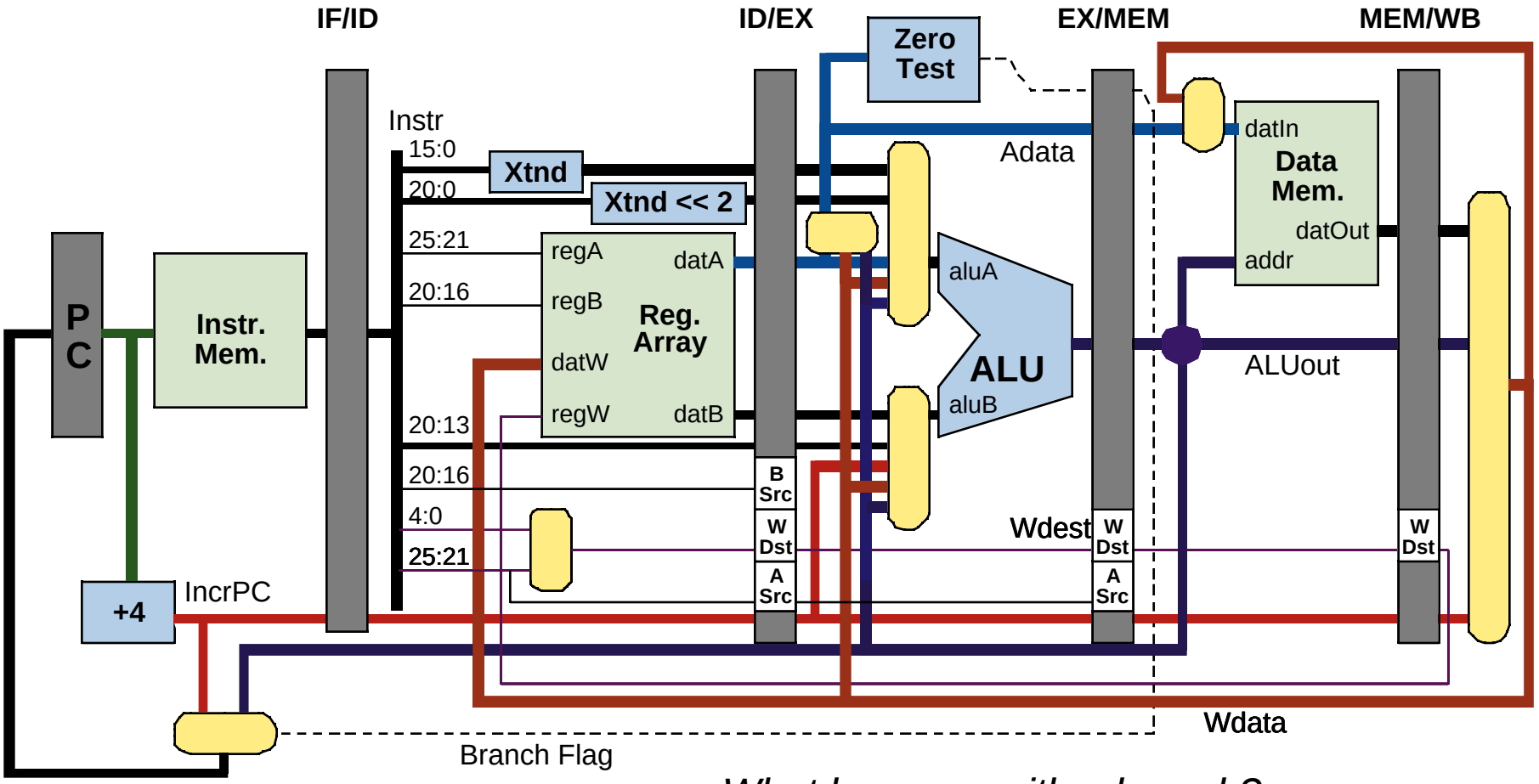
- **2 distances (either 1 or 2)**
- **5 classes of writer**
- **8 classes of readers**

Testing Methodology

- **80 cases to cover all interactions between supported instruction types**

Pipelined datapath

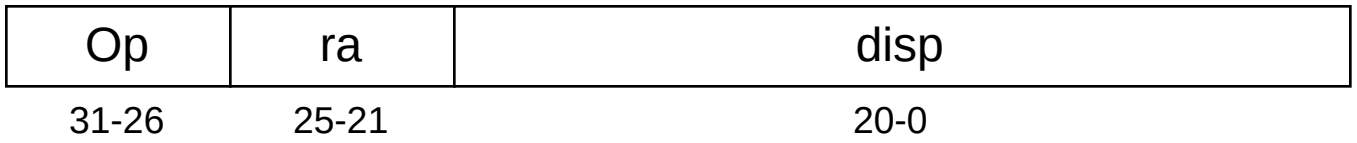
IF	ID	EX	MEM	WB
instruction fetch	instruction decode/ register fetch	execute/ address calc	memory access	write back

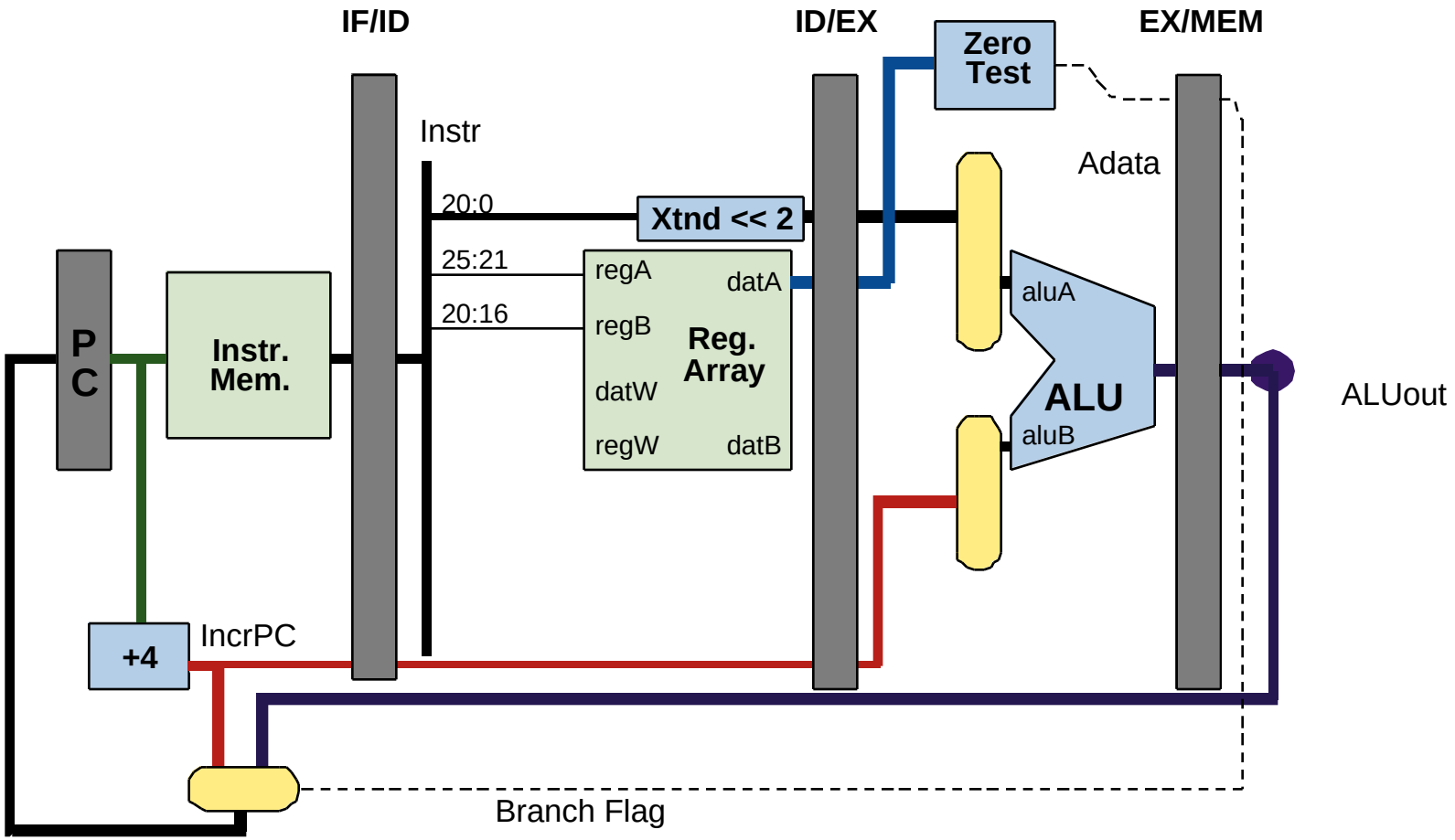


What happens with a branch?

Conditional Branch Instruction Handling

beq: PC <-- Ra == 0 ? PC + 4 + disp*4 : PC + 4





Branch on equal

beq: PC <-- Ra == 0 ? PC + 4 + disp*4 : PC + 4

IF: Instruction fetch

- **IR <-- IMemory[PC]**
- **incrPC <-- PC + 4**

ID: Instruction decode/register fetch

- **A <-- Register[IR[25:21]]**

Ex: Execute

- **Target <-- incrPC + SignExtend(IR[20:0]) << 2**
- **Z <-- (A == 0)**

MEM: Memory

- **PC <-- Z ? Target : incrPC**

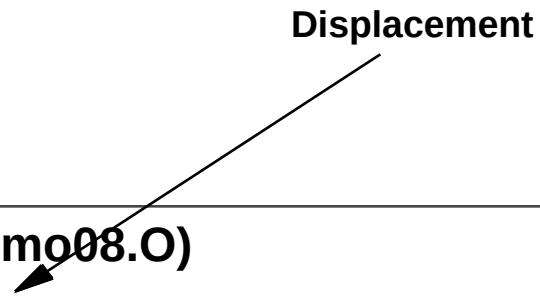
WB: Write back

- **nop**

Branch Example

Desired Behavior

- **Take branch at 0x00**
- **Execute target 0x18**
  - PC + 4 + disp << 2
  - PC = 0x00
  - disp = 5



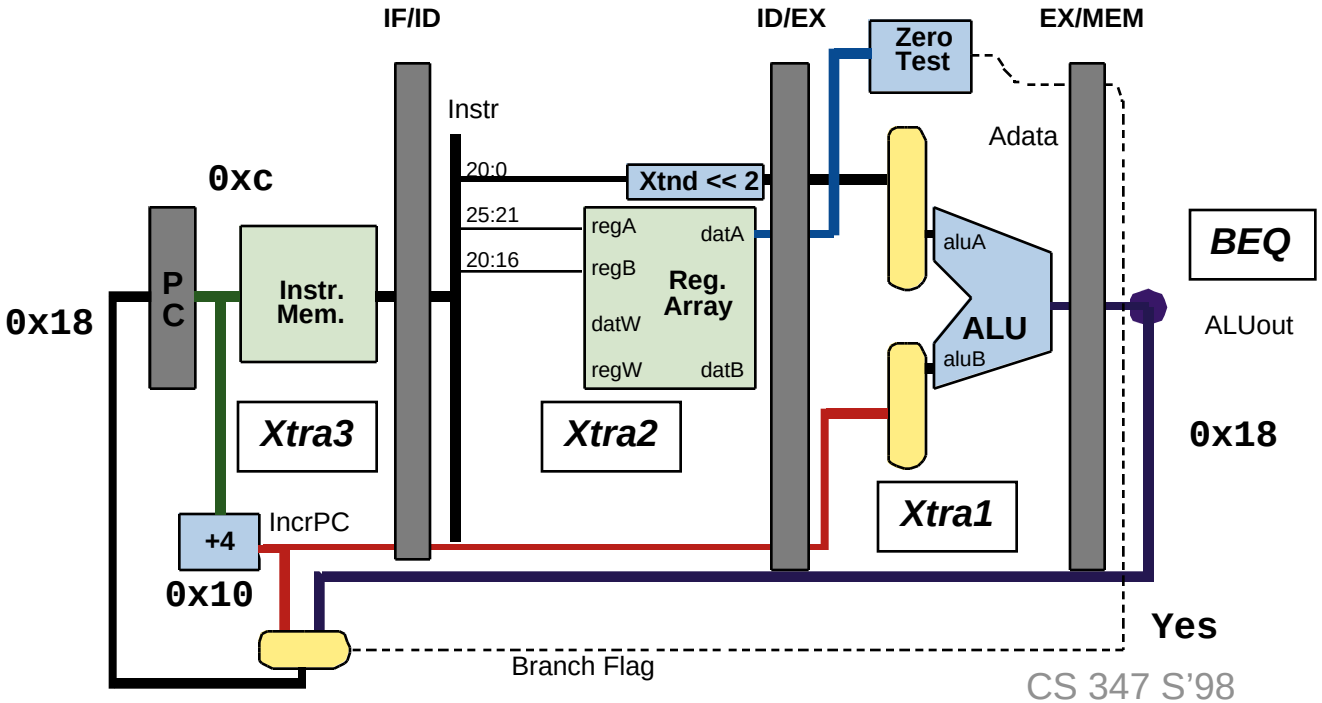
Branch Code (demo08.O)

```
0x0:  e7e00005 beq   r31, 0x18       # Take
0x4:  43e7f401 addq  r31, 0x3f, r1   # (Skip)
0x8:  43e7f402 addq  r31, 0x3f, r2   # (Skip)
0xc:  43e7f403 addq  r31, 0x3f, r3   # (Skip)
0x10: 43e7f404 addq  r31, 0x3f, r4   # (Skip)
0x14: 47ff041f bis   r31, r31, r31
0x18: 43e7f405 addq  r31, 0x3f, r5   # (Target)
0x1c: 47ff041f bis   r31, r31, r31
0x20: 00000000 call_pal              halt
```

Branch Hazard Example

```
0x0:  beq     r31, 0x18        # Take
0x4:  addq    r31, 0x3f, r1    # Xtra1
0x8:  addq    r31, 0x3f, r2    # Xtra2
0xc:  addq    r31, 0x3f, r3    # Xtra3
0x10: addq    r31, 0x3f, r4    # Xtra4

0x18: addq    r31, 0x3f, r5    # Target
```

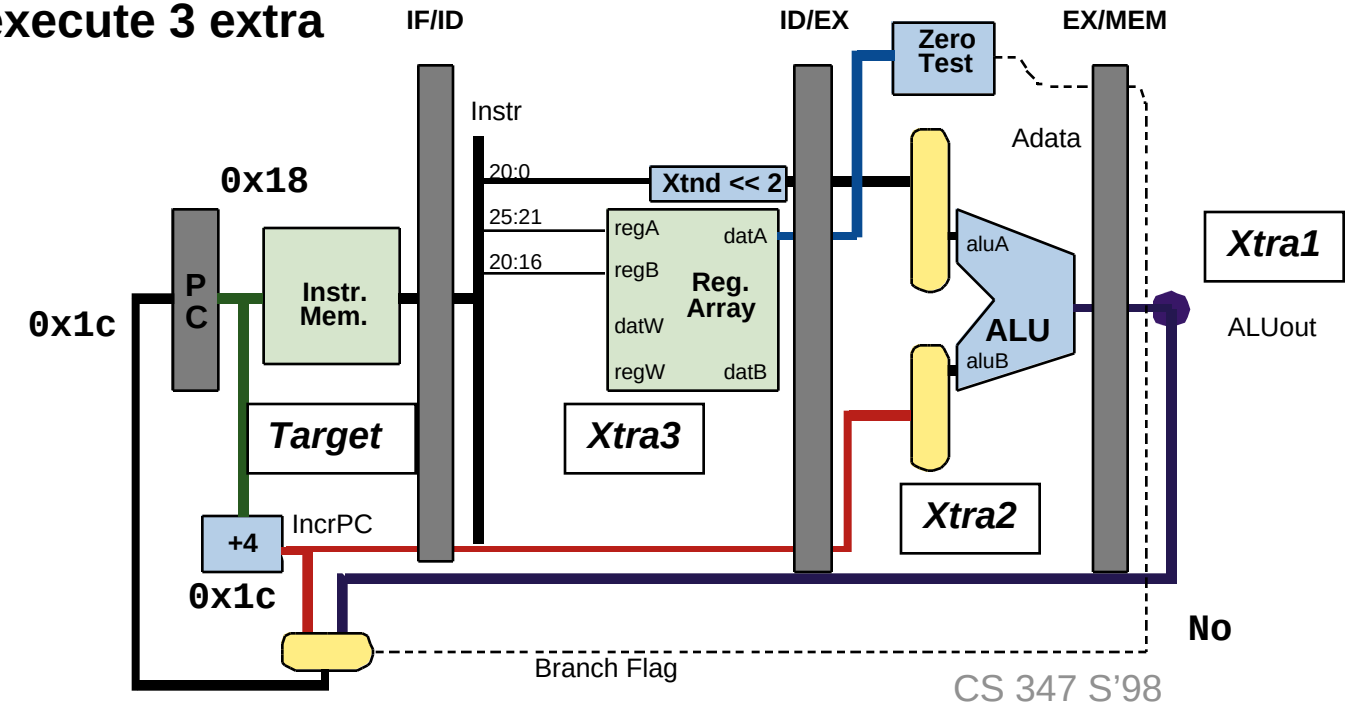
- **With BEQ in Mem stage**



Branch Hazard Example (cont.)

```
0x0:  beq     r31, 0x18       # Take
0x4:  addq    r31, 0x3f, r1   # Xtra1
0x8:  addq    r31, 0x3f, r2   # Xtra2
0xc:  addq    r31, 0x3f, r3   # Xtra3
0x10: addq    r31, 0x3f, r4   # Xtra4

0x18: addq    r31, 0x3f, r5   # Target
```

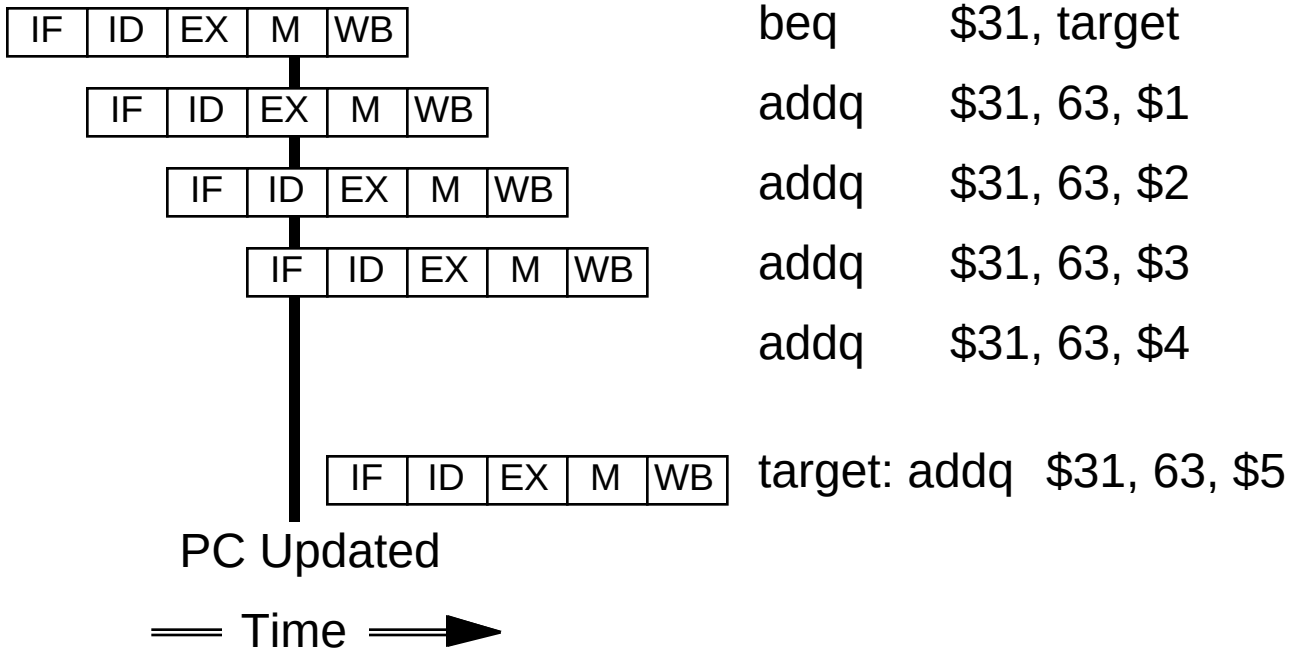
- **One cycle later**
- **Problem: Will execute 3 extra instructions!**



Branch Hazard Pipeline Diagram

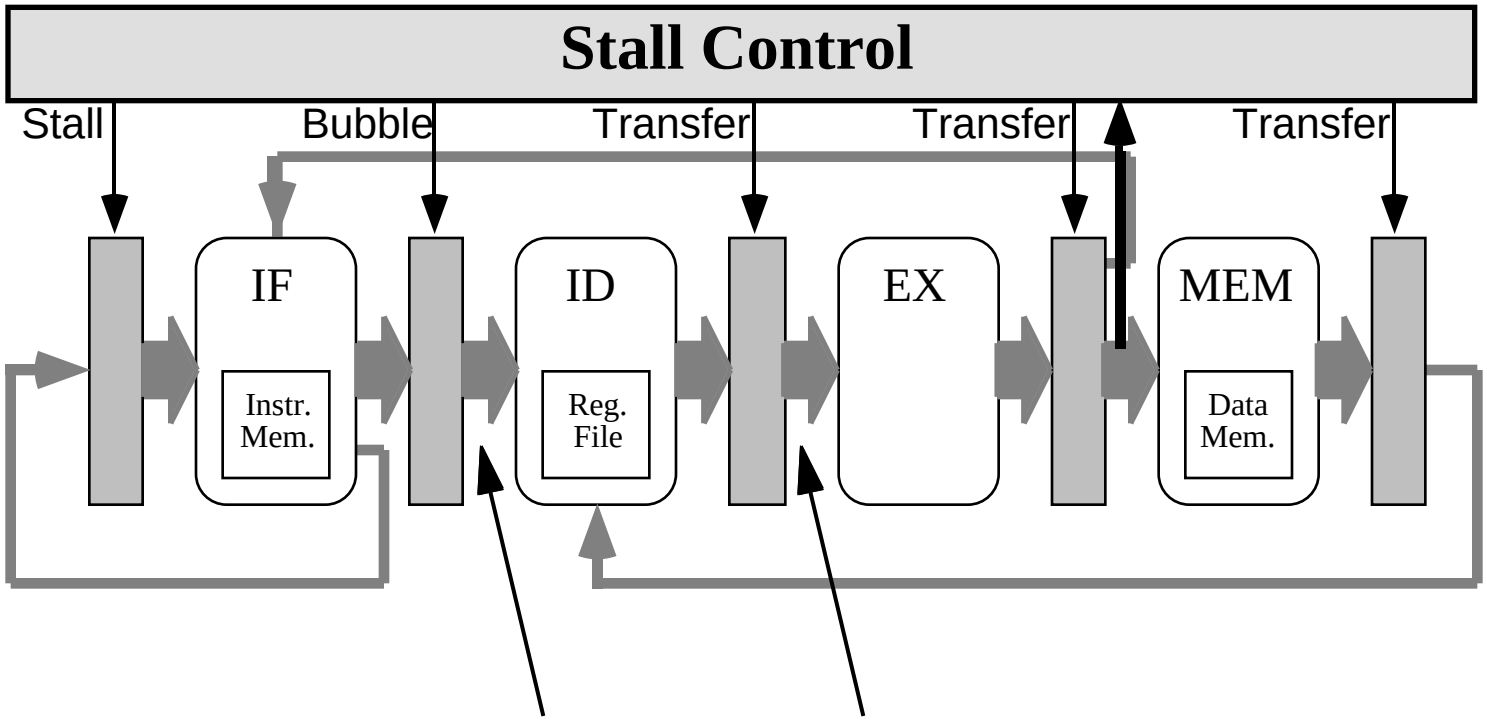
Problem

- **Instruction fetched in IF, branch condition set in MEM**



Stall Until Resolve Branch

- **Detect when branch in stages ID or EX**
- **Stop fetching until resolve**
  - Stall IF. Inject bubble into ID

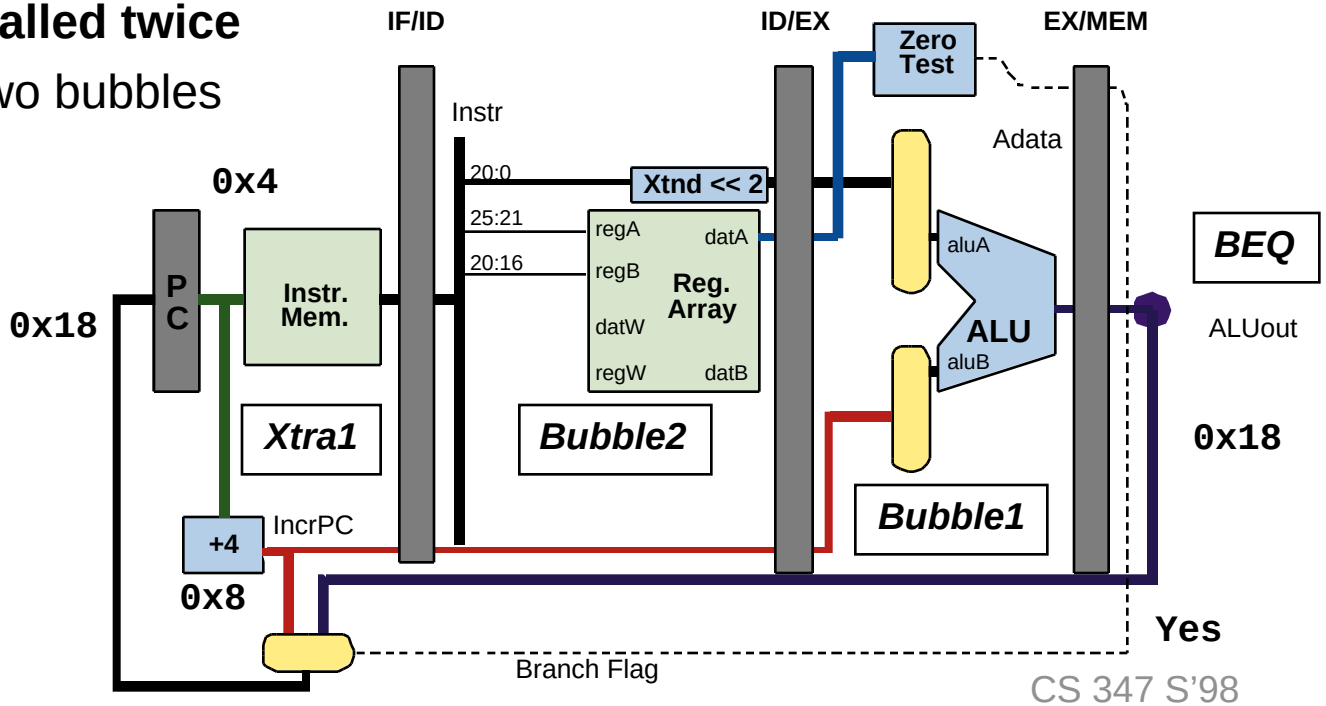


Perform when branch in either stage

Stalling Branch Example

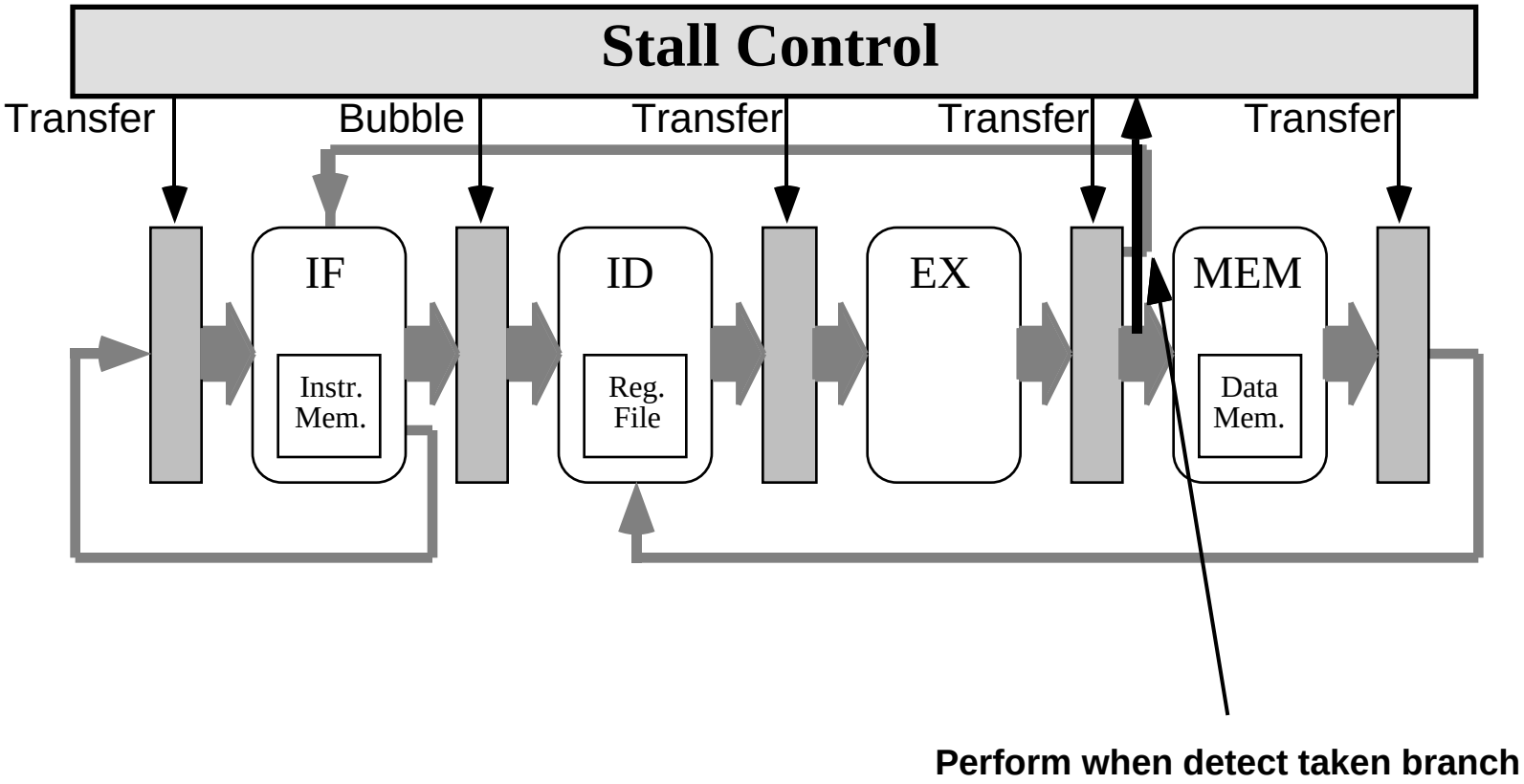
```
0x0:  beq     r31, 0x18       # Take
0x4:  addq    r31, 0x3f, r1   # Xtra1
0x8:  addq    r31, 0x3f, r2   # Xtra2
0xc:  addq    r31, 0x3f, r3   # Xtra3
0x10: addq    r31, 0x3f, r4   # Xtra4

0x18: addq    r31, 0x3f, r5   # Target
```

- **With BEQ in Mem stage**
- **Will have stalled twice**
  - Injects two bubbles



Taken Branch Resolution

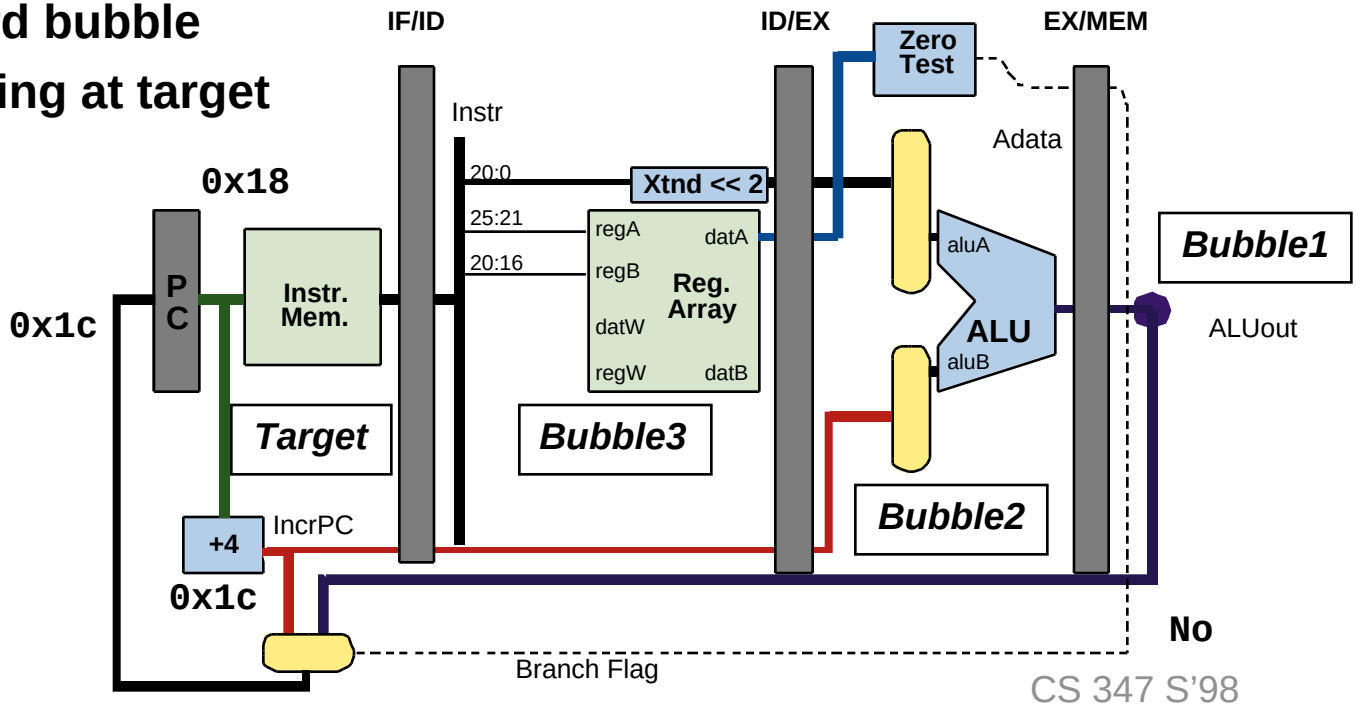
- **When branch taken, still have instruction Xtra1 in pipe**
- **Need to flush it when detect taken branch in Mem**
  - Convert it to bubble



Taken Branch Resolution Example

```
0x0:  beq     r31, 0x18        # Take
0x4:  addq    r31, 0x3f, r1    # Xtra1
0x8:  addq    r31, 0x3f, r2    # Xtra2
0xc:  addq    r31, 0x3f, r3    # Xtra3
0x10: addq    r31, 0x3f, r4    # Xtra4

0x18: addq    r31, 0x3f, r5    # Target
```

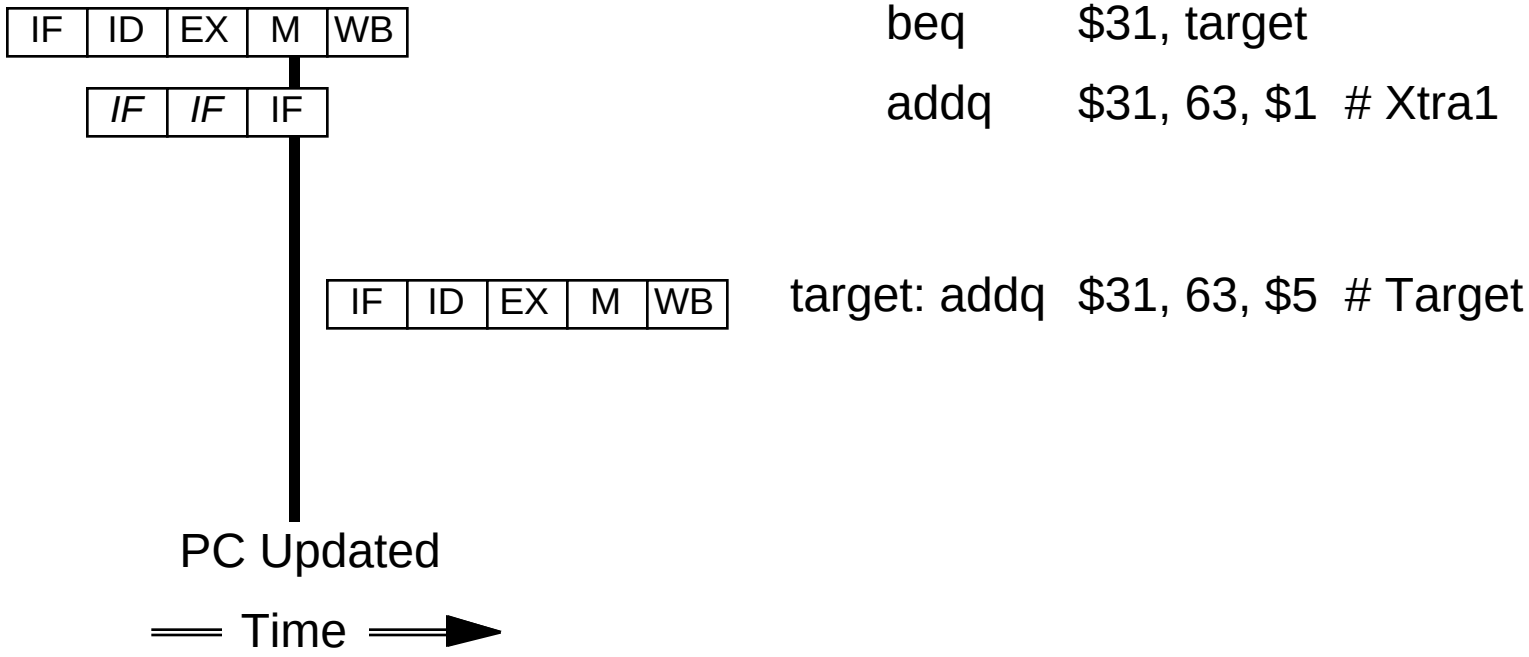
- When branch taken
- Generate 3rd bubble
- Begin fetching at target



Taken Branch Pipeline Diagram

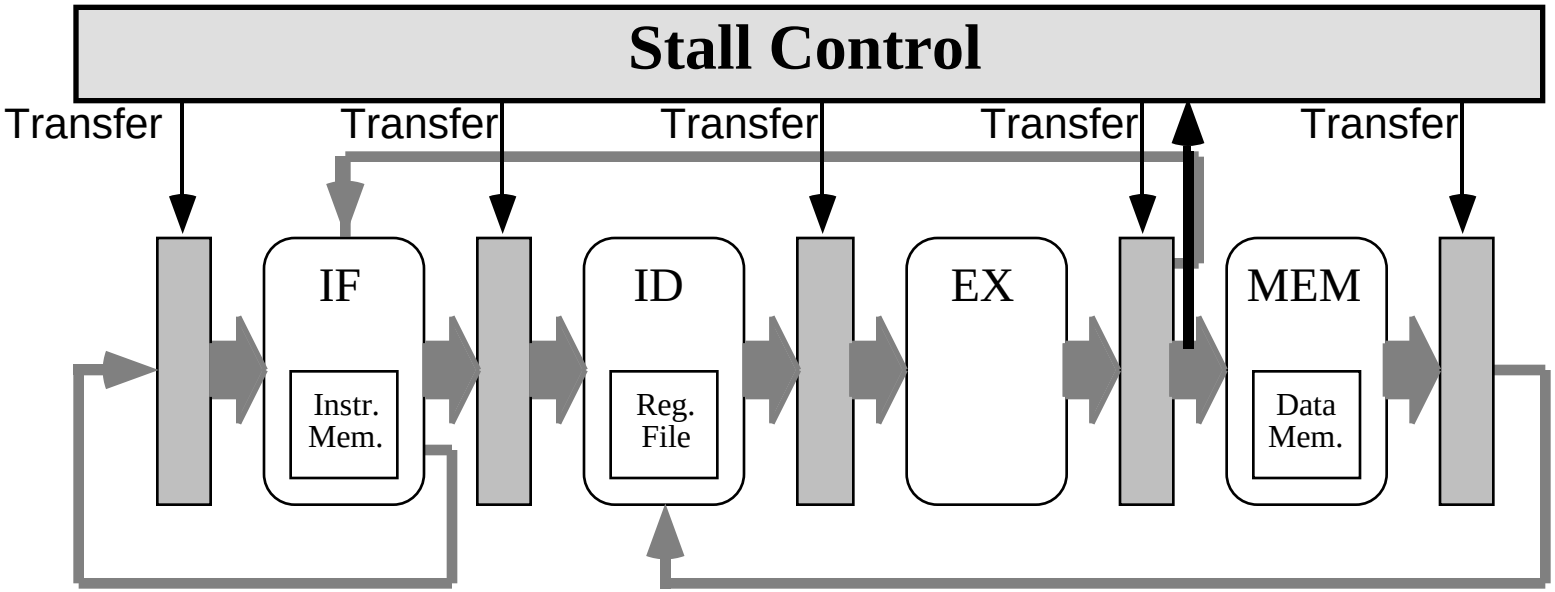
Behavior

- **Instruction Xtra1 held in IF for two extra cycles**
- **Then turn into bubble as enters ID**



Not Taken Branch Resolution

- **[Stall two cycles with not-taken branches as well]**
- **When branch not taken, already have instruction Xtra1 in pipe**
- **Let it proceed as usual**

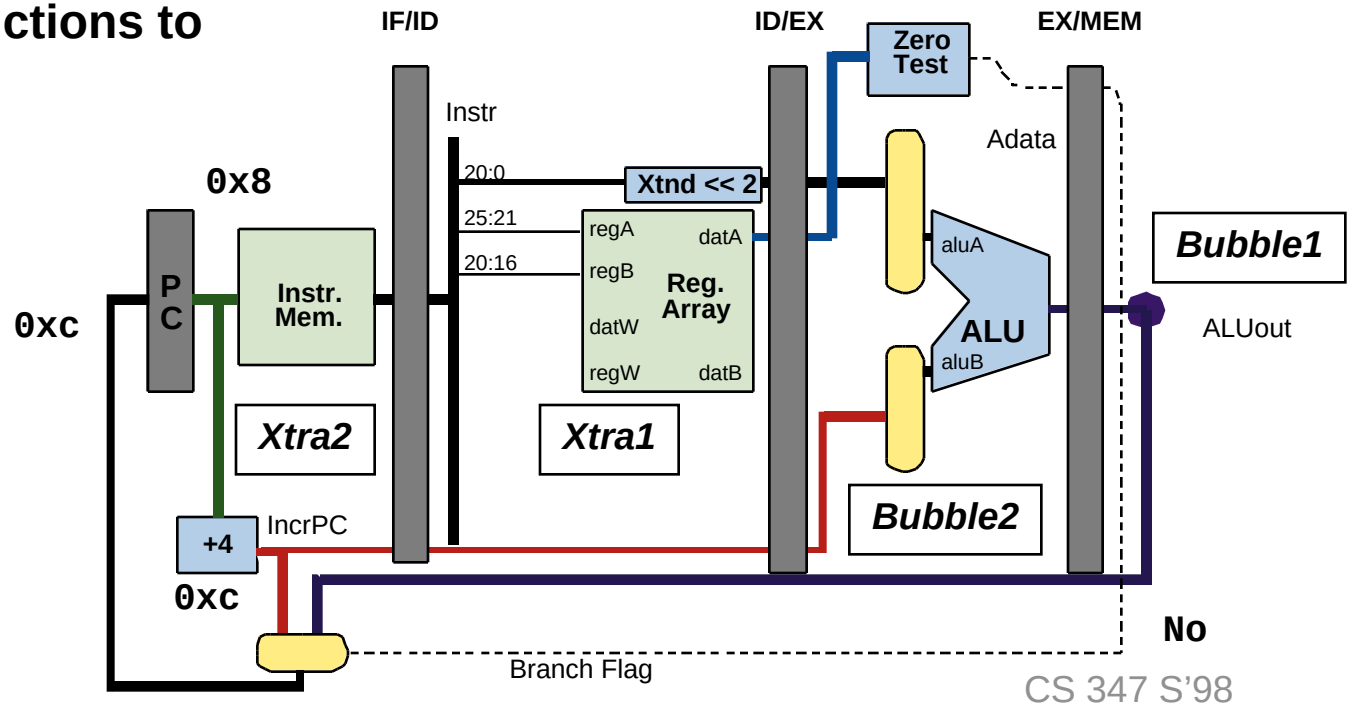


Not Taken Branch Resolution Example

demo09.O

```
0x0:  bne    r31, 0x18      # Don't Take
0x4:  addq   r31, 0x3f, r1  # Xtra1
0x8:  addq   r31, 0x3f, r2  # Xtra2
0xc:  addq   r31, 0x3f, r3  # Xtra3
0x10: addq   r31, 0x3f, r4  # Xtra4
```

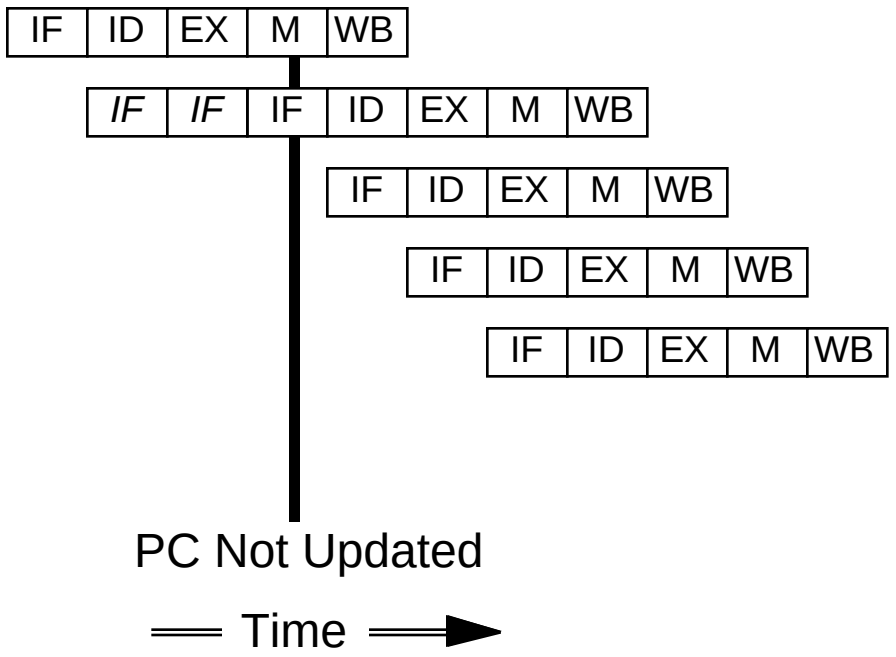
- **Branch not taken**
- **Allow instructions to proceed**



Not Taken Branch Pipeline Diagram

Behavior

- **Instruction Xtra1 held in IF for two extra cycles**
- **Then allowed to proceed**




```
beq     $31, target
addq    $31, 63, $1  # Xtra1
addq    $31, 63, $2  # Xtra2
addq    $31, 63, $3  # Xtra3
addq    $31, 63, $4  # Xtra4
```

Analysis of Stalling

Branch Instruction Timing

- **1 instruction cycle**
- **3 extra cycles when taken**
- **2 extra cycles when not taken**

Performance Impact

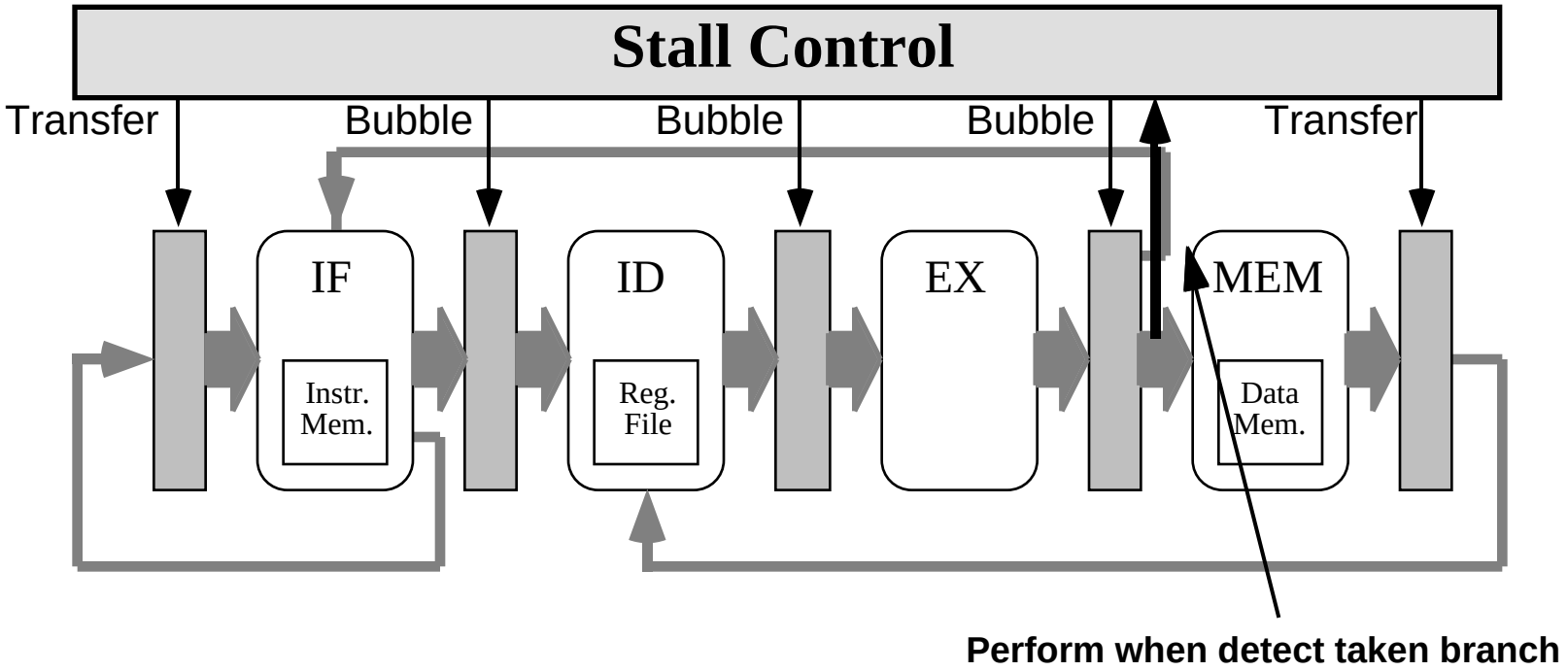
- **Branchs 16% of instructions in SpecInt92 benchmarks**
- **67% branches are taken**
- **Adds 0.16 * (0.67 * 3 + 0.33 * 2) == 0.43 cycles to CPI**
  - Average number of cycles per instruction
  - Serious performance impact

Fetch & Cancel When Taken

- **Instruction does not cause any updates until MEM or WB stages**
- **Instruction can be "cancelled" from pipe up through EX stage**
  - Replace with bubble

Strategy

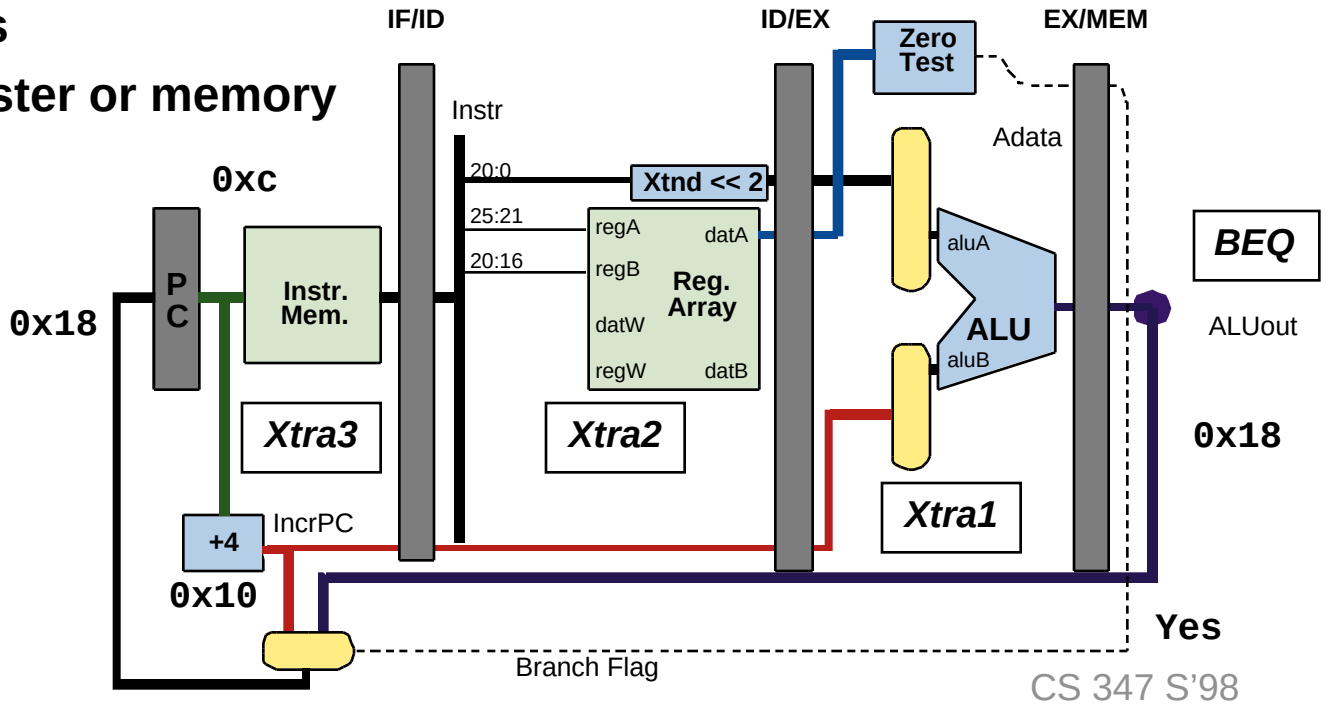
- **Continue fetching under assumption that branch not taken**
- **If decide to take branch, cancel undesired ones**



Canceling Branch Example

```
0x0:  beq     r31, 0x18       # Take
0x4:  addq    r31, 0x3f, r1   # Xtra1
0x8:  addq    r31, 0x3f, r2   # Xtra2
0xc:  addq    r31, 0x3f, r3   # Xtra3
0x10: addq    r31, 0x3f, r4   # Xtra4

0x18: addq    r31, 0x3f, r5   # Target
```

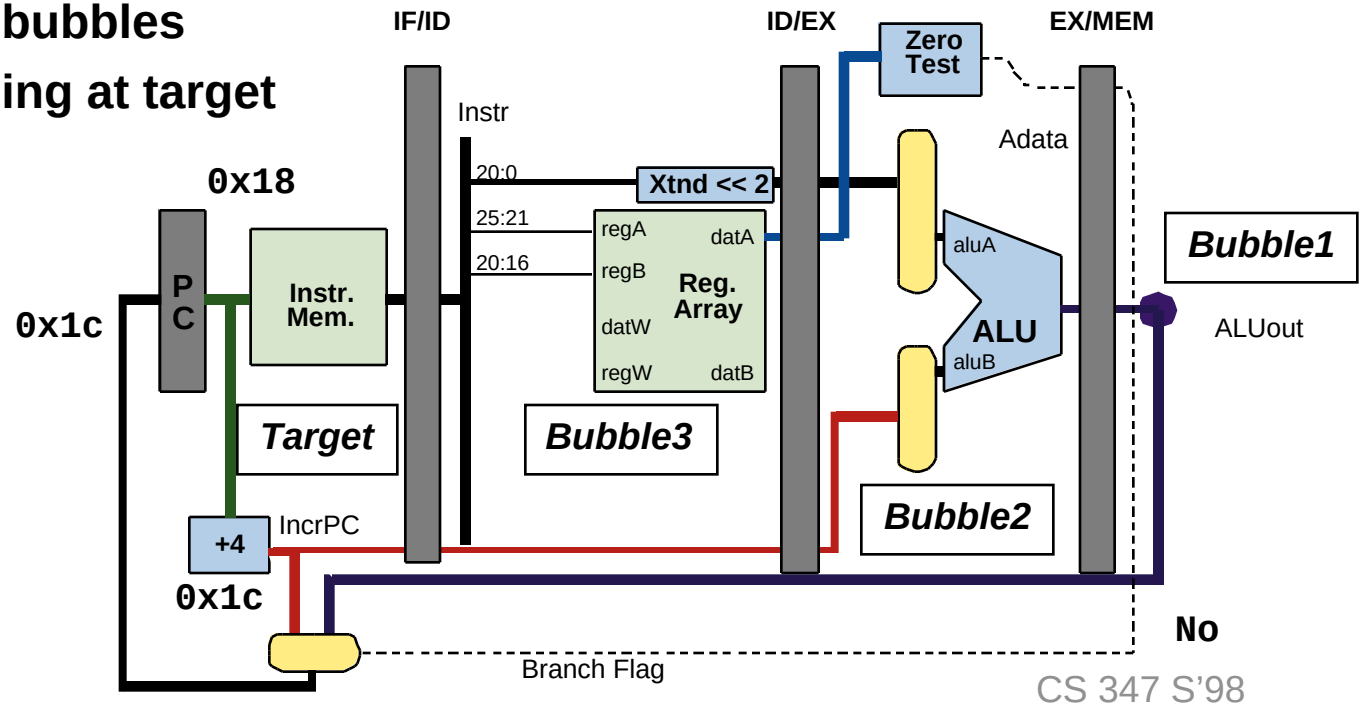
- **With BEQ in Mem stage**
- **Will have fetched 3 extra instructions**
- **But no register or memory updates**



Canceling Branch Resolution Example

```
0x0:  beq     r31, 0x18       # Take
0x4:  addq    r31, 0x3f, r1   # Xtra1
0x8:  addq    r31, 0x3f, r2   # Xtra2
0xc:  addq    r31, 0x3f, r3   # Xtra3
0x10: addq    r31, 0x3f, r4   # Xtra4

0x18: addq    r31, 0x3f, r5   # Target
```

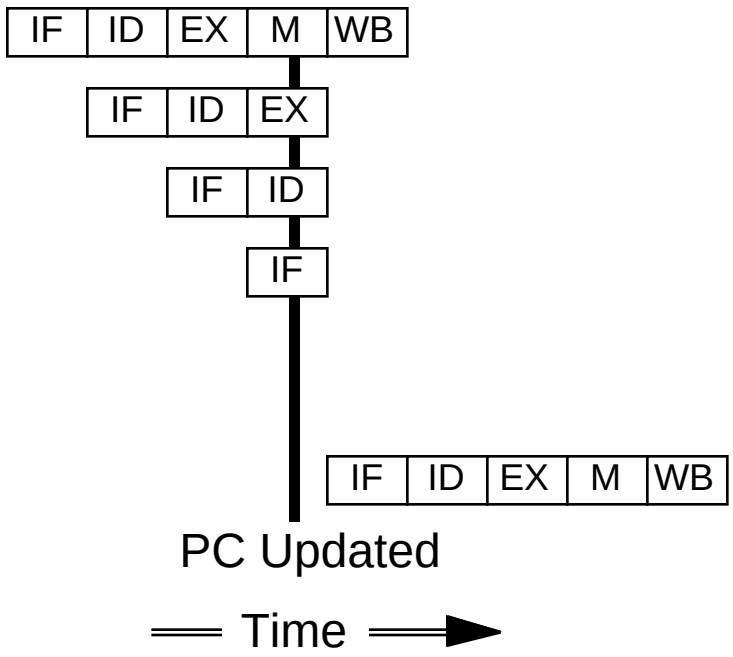
- When branch taken
- Generate 3 bubbles
- Begin fetching at target



Canceling Branch Pipeline Diagram

Operation

- **Process instructions assuming branch will not be taken**
- **When *is* taken, cancel 3 following instructions**

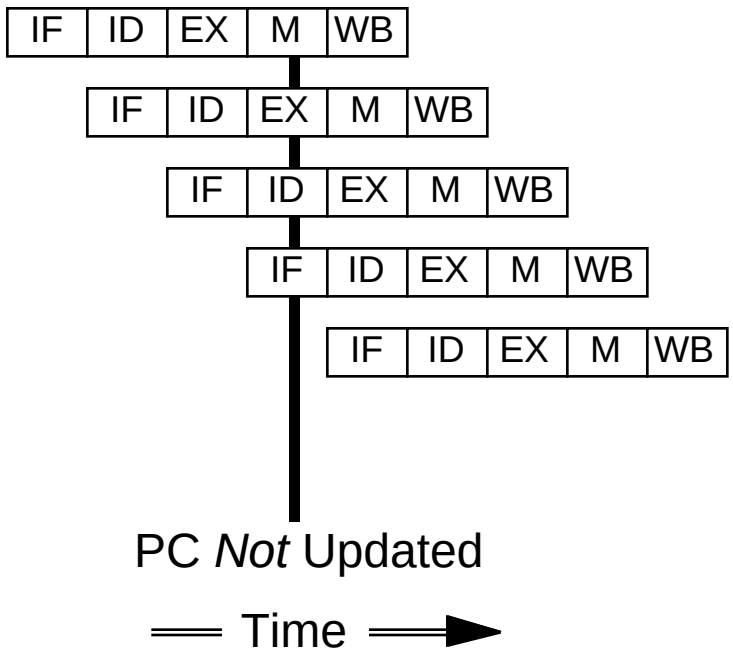



```
beq     $31, target
addq    $31, 63, $1
addq    $31, 63, $2
addq    $31, 63, $3
addq    $31, 63, $4

target: addq  $31, 63, $5
```

Noncanceling Branch Pipeline Diagram

Operation

- **Process instructions assuming branch will not be taken**
- **If really isn't taken, then instructions flow unimpeded**




```
bne     $31, target
addq    $31, 63, $1
addq    $31, 63, $2
addq    $31, 63, $3
addq    $31, 63, $4

target: addq  $31, 63, $5
```

Branch Prediction Analysis

Our Scheme Implements "Predict Not Taken"

- **But 67% of branches are taken**
- **Impact on CPI: 0.16 * 0.67 * 3.0 = 0.32**
  - Still not very good

Alternative Schemes

- **Predict taken**
  - Would be hard to squeeze into our pipeline
    - » Can't compute target until ID
- **Backwards taken, forwards not taken**
  - Predict based on sign of displacement
  - Exploits fact that loops usually closed with backward branches