Handling Control Hazards in Pipelined MIPS Processor

March 5, 1996

Topics

- **Branch Instructions**
  - Added data hazards
- **Pipeline for Instruction Fetch & Control**
- **Handling Control Hazards**
  - Stall until resolve
  - Predict not taken
- **More advanced techniques**

MIPS architecture subset

R-type instructions (add, sub, and, or, slt): rd <-- rs funct rt

0	rs	rt	rd	shamt	funct
31-26	25-21	20-16	15-11	10-6	5-0

RI-type instructions (addiu): rt <-- rs funct I16

0	rs	rt	I16
31-26	25-21	20-16	15-0

Load: rt <-- Mem[rs + I16] **Store:** Mem[rs + I16] <-- rt

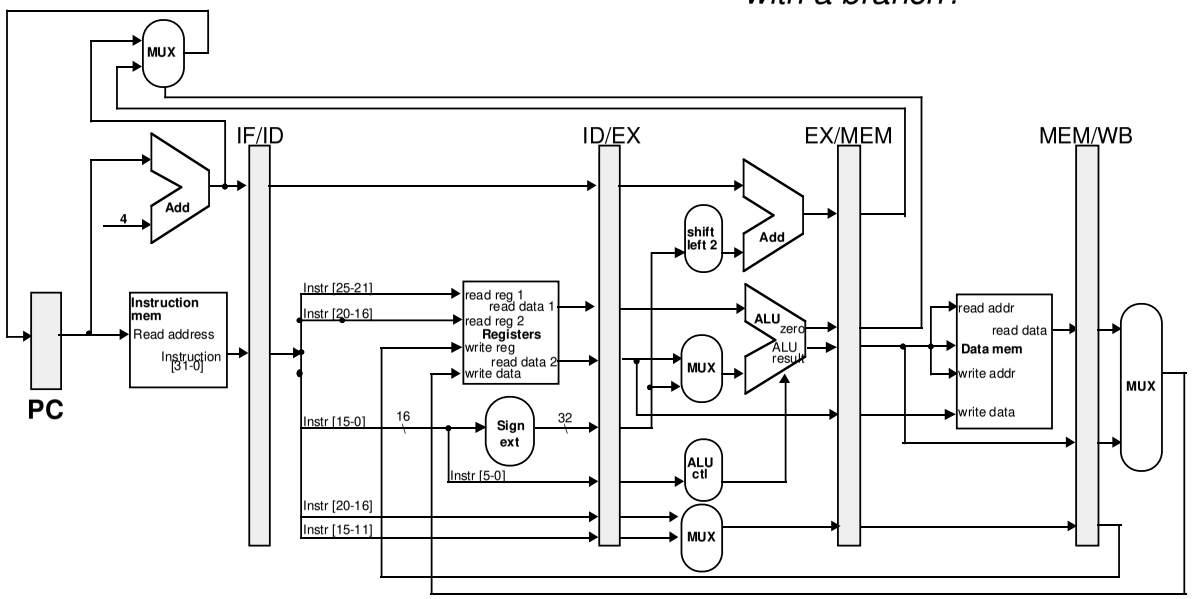
35 or 43	rs	rt	I16
31-26	25-21	20-16	15-0

Branch equal: PC <-- (rs == rt) ? PC + 4 + I16 <<2 : PC + 4

4	rs	rt	I16
31-26	25-21	20-16	15-0

Pipelined datapath

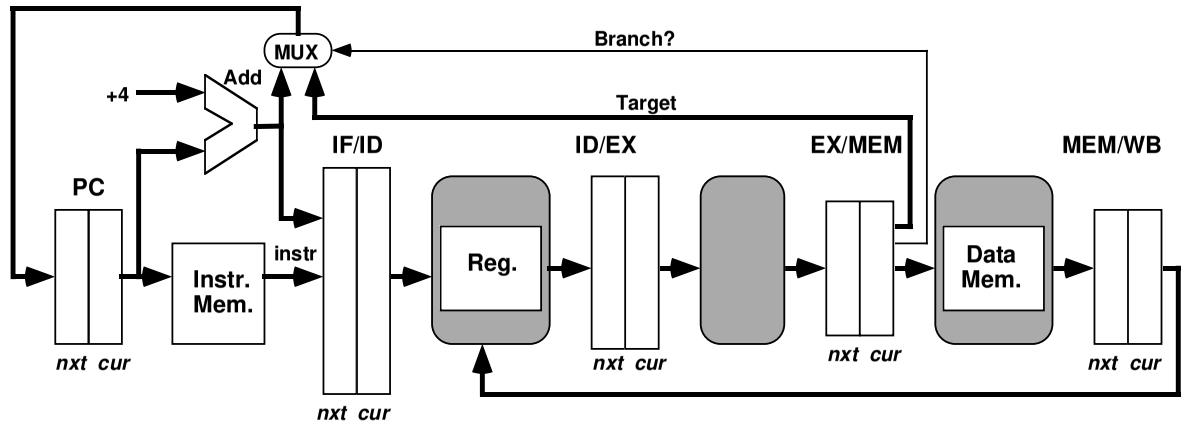
What happens with a branch?



Branch Instructions

Branch equal: PC <-- (rs == rt) ? PC + 4 + I16 <<2 : PC + 4

4	rs	rt	I16
31-26	25-21	20-16	15-0



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**
- **Don't allow dependent branch in load delay slot**
- **Bypass**
  - EX-EX
  - MEM-EX

ALU-Branch

```
add $1, $2, $3
beq $4, $1, targ
```

Distant ALU-Branch

```
add $1, $2, $3
add $0, $0, $0
beq $4, $1, targ
```

Load-Branch

```
lw $1, 8($2)
beq $1, $0, targ
```


Branch Example

Assume

- **All registers initially 0**

Desired Behavior

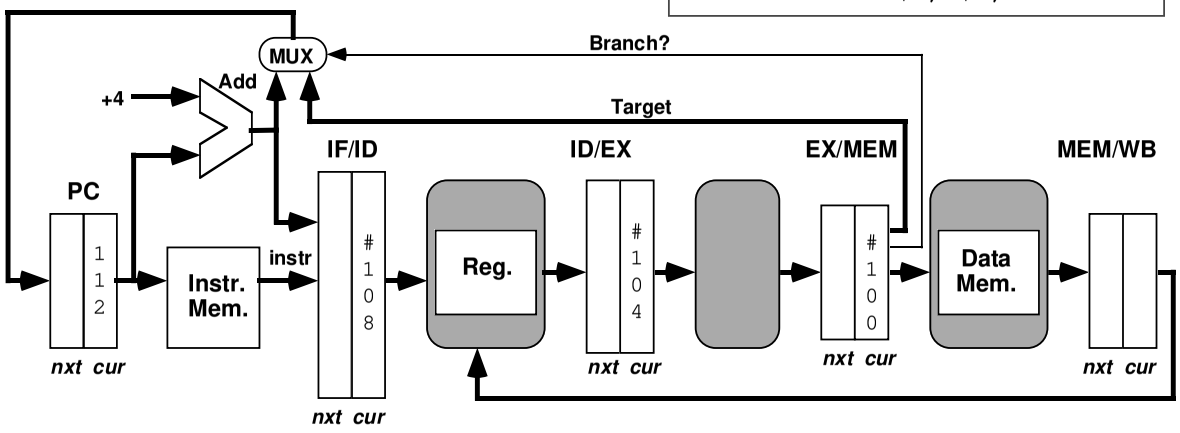
- **Take branch 100**
- **Execute delay slot 104**
  - H&P don't do this
- **Execute target 120**
  - PC + 4 + I16<<2
  - PC = 100
  - I16 = 4

Branch Code

```
100:   beq $0, $0, 4
104:   addi $1, $0, 1
108:   addi $2, $0, 2
112:   addi $3, $0, 3
116:   addi $4, $0, 4
120:   addi $5, $0, 5
```

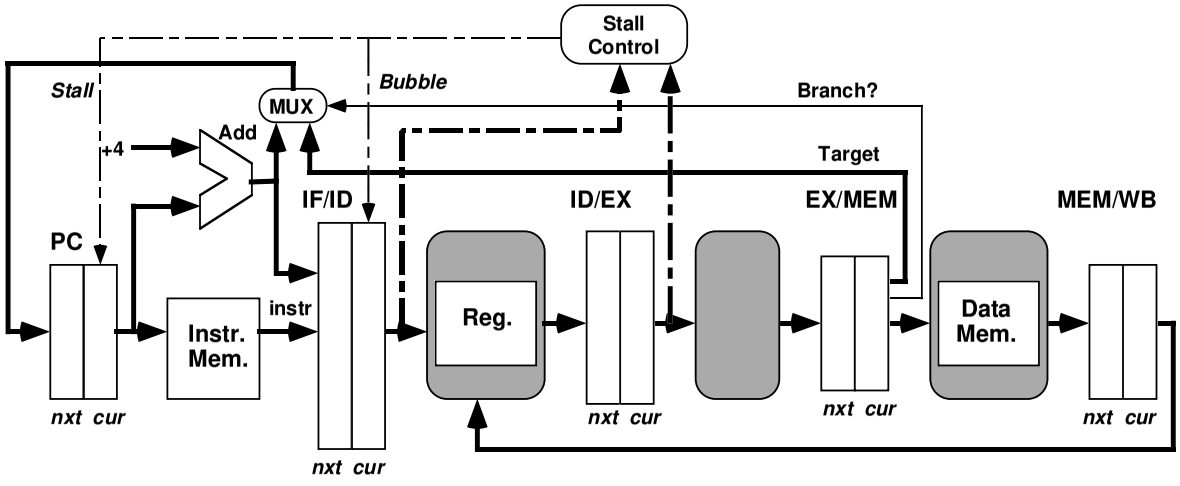
Branch Example

```
100:    beq $0, $0, 4
104:    addi $1, $0, 1
108:    addi $2, $0, 2
112:    addi $3, $0, 3
116:    addi $4, $0, 4
120:    addi $5, $0, 5
```



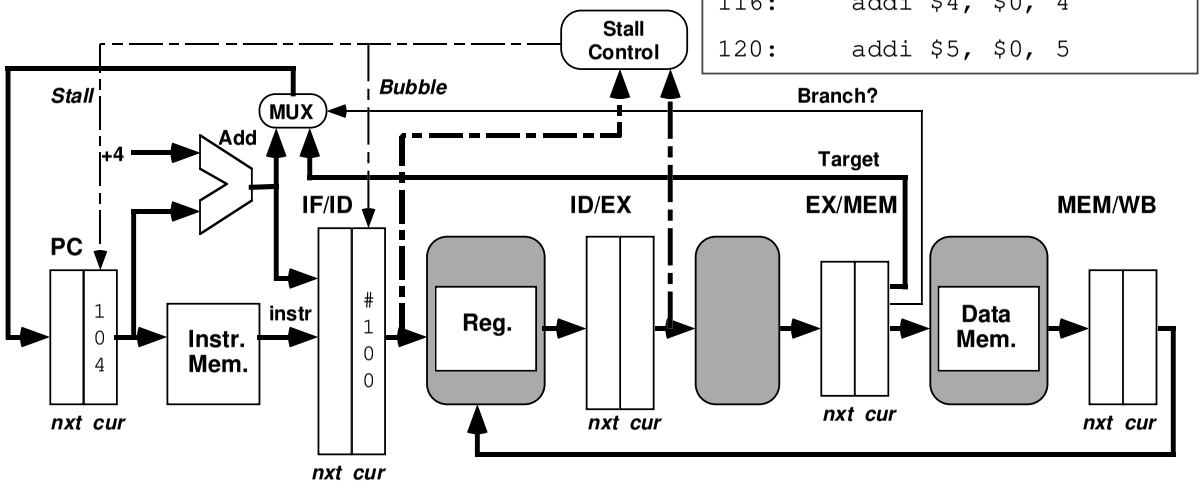
Stall Until Resolve Branch

- Detect when branch in stages ID or EX
- Stop fetching until resolve
- Delay slot instruction just before target



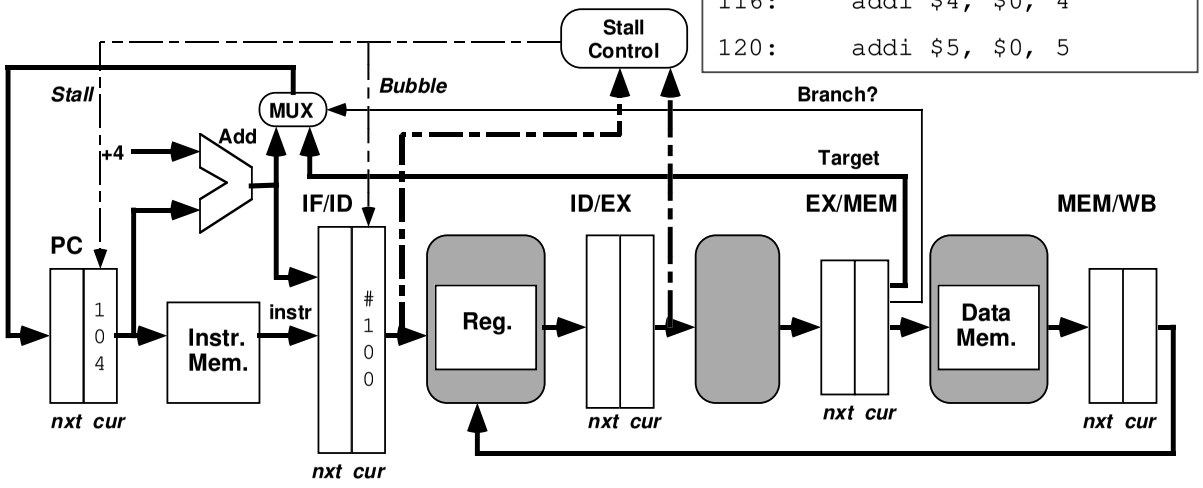
Stall Until Branch Taken Example

```
100:    beq $0, $0, 4
104:    addi $1, $0, 1
108:    addi $2, $0, 2
112:    addi $3, $0, 3
116:    addi $4, $0, 4
120:    addi $5, $0, 5
```



Stall Until Branch NOT Taken Example

```
100:    BNE $0, $0, 4
104:    addi $1, $0, 1
108:    addi $2, $0, 2
112:    addi $3, $0, 3
116:    addi $4, $0, 4
120:    addi $5, $0, 5
```



Analysis of Stalling

Branch Instruction Timing

- 1 instruction cycle + 2 stall cycles
- Whether or not branch taken

Performance Impact

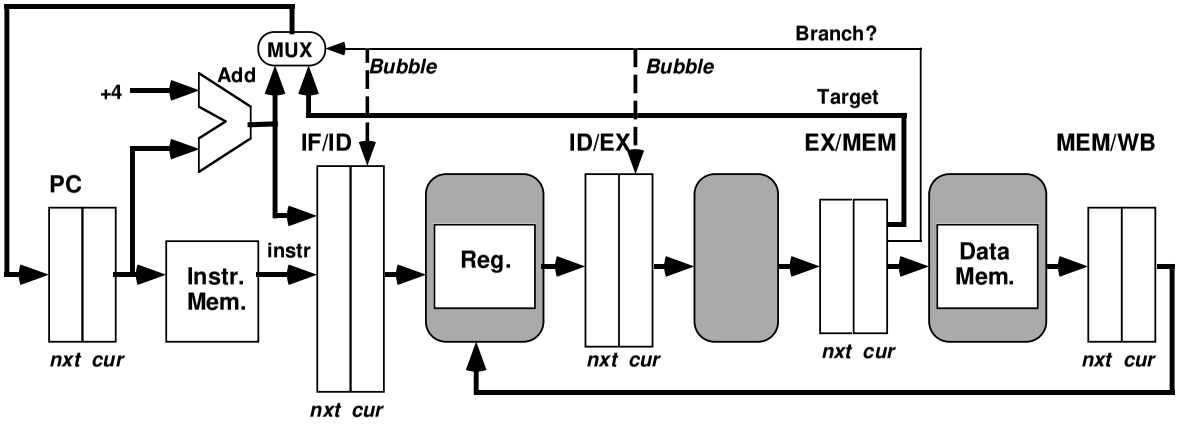
- Branchs 16% of instructions in SpecInt92 benchmarks
- Would give CPI $\geq$ 1.32
  - Lower bound achieved only when all other instructions single cycle

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

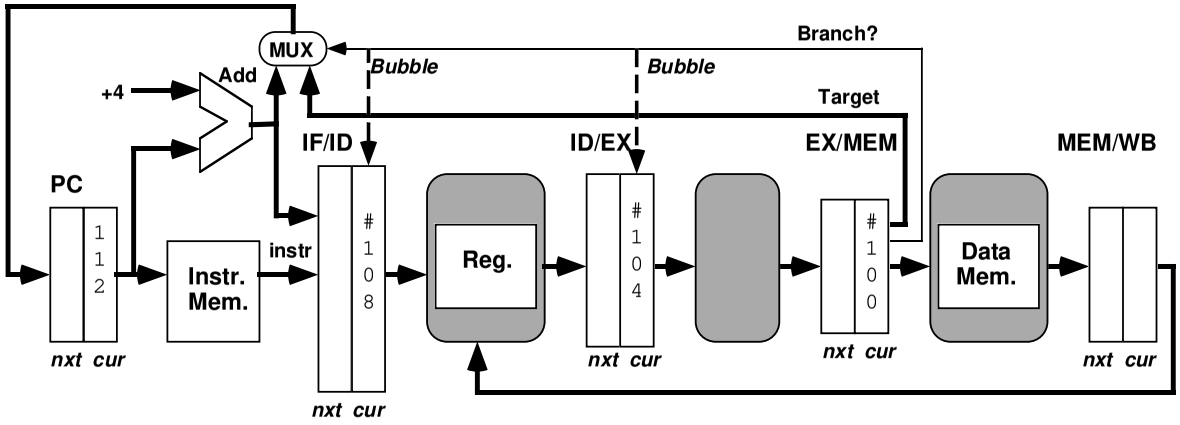
Strategy

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



Branch Taken Example

```
100:    beq $0, $0, 4
104:    addi $1, $0, 1
108:    addi $2, $0, 2
112:    addi $3, $0, 3
116:    addi $4, $0, 4
120:    addi $5, $0, 5
```



Branch Prediction Analysis

Our Scheme Implements "Predict Not Taken"

- **But 67% of branches are taken**
- **Would give CPI ≥ 1.0 + 0.16 * 0.67 * 2.0 = 1.22**

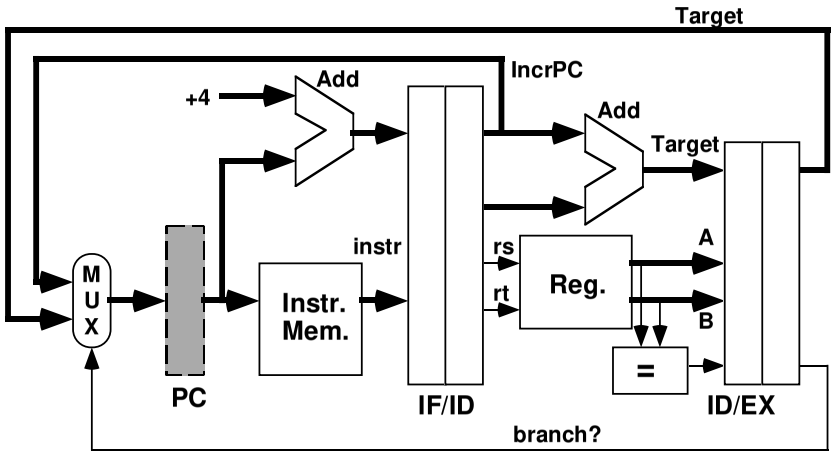
Alternative Schemes

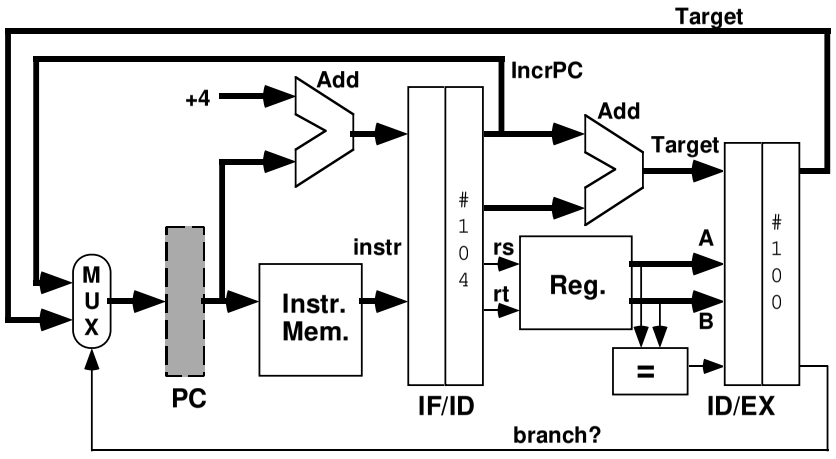
- **Predict taken**
  - Would be hard to squeeze into our pipeline
    - » Can't compute target until ID
  - In principle, would give CPI ≥ 1.11
- **Backwards taken, forwards not taken**
  - Predict based on sign of I16
  - Exploits fact that loops usually closed with backward branches

How Does MIPS Handle Branches?

Avoids any Branch Penalty!

- **Do not treat PC as pipe register**
  - Loaded at beginning of IF
- **Compute branch target & branch condition in ID**
  - Can fetch at target in following cycle



MIPS Branch Example

```
100:    beq $0, $0, 4
104:    addi $1, $0, 1
108:    addi $2, $0, 2
112:    addi $3, $0, 3
116:    addi $4, $0, 4
120:    addi $5, $0, 5
```