Microprocessor & Computer Architecture (µpCA)

UE19CS252

Dr. D. C. Kiran

Department of
Computer Science and Engineering

Microprocessor & Computer Architecture (µpCA)

Pipeline Processor: Control Hazard

Dr. D. C. Kiran
Department of Computer Science and Engineering

Syllabus

~~Unit 1: Basic Processor Architecture and Design~~

Unit 2: Pipelined Processor and Design

- ~~3-Stage ARM Processor~~
- ~~5-Stage Pipeline Processor~~
- ~~Introduction to Pipeline Processor~~
- ~~What May Go Wrong?~~
- ~~Introduction to Hazards, Stalls,~~
- ~~Structural Hazards~~
- ~~Data Hazard~~
  - ~~RAW, WAR, WAW Hazards~~
- ~~Attacking Data Hazard~~
  - ~~Software Approach~~
  - ~~Hardware Approach~~
- Control Hazards

Microprocessor & Computer Architecture (µpCA)

Text 1: "Computer Organization and Design", Patterson, Hennessey, 5th Edition, Morgan Kaufmann, 2014.

Reference 1:"Computer Architecture: A Quantitative Approach", Hennessey, Patterson, 5th Edition, Morgan Kaufmann, 2011.

Appendix C **Pipelining: Basic and Intermediate Concepts**

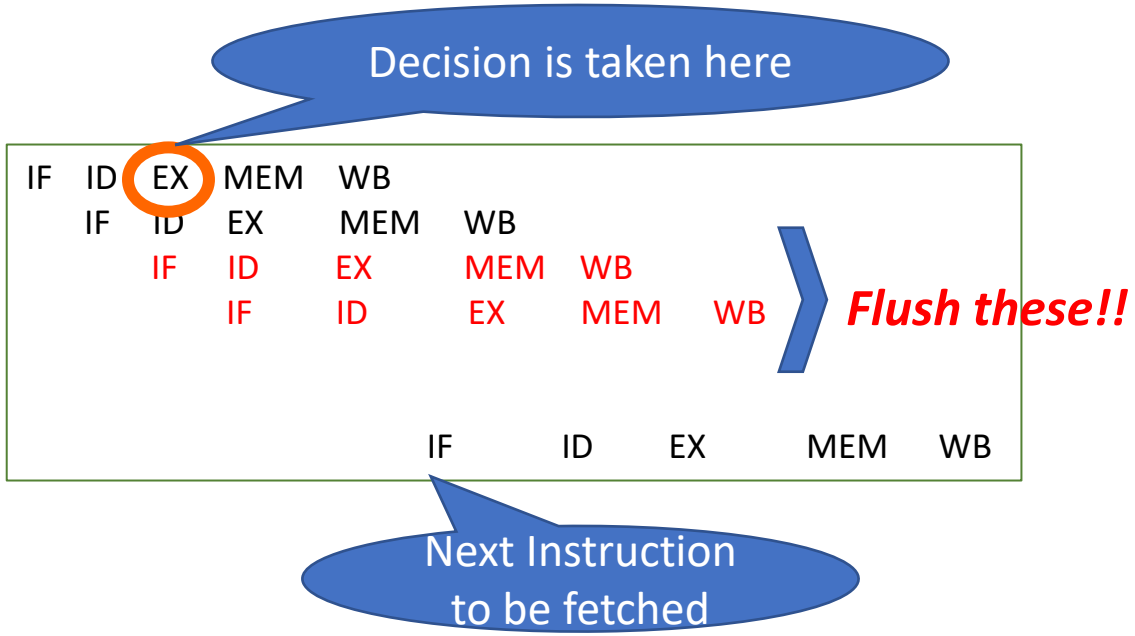
Control Hazards

- When the flow of instruction addresses is not sequential (i.e., PC = PC + 4); incurred by change of flow instructions
  - Conditional branches (**beq, bne**)
  - Unconditional branches (**b, bal**)
  - Exceptions
- Undesirable instructions get into the pipeline even before branch instruction is fetched, if branch is taken (Compare and Change PC in EX Stage)

Ex:

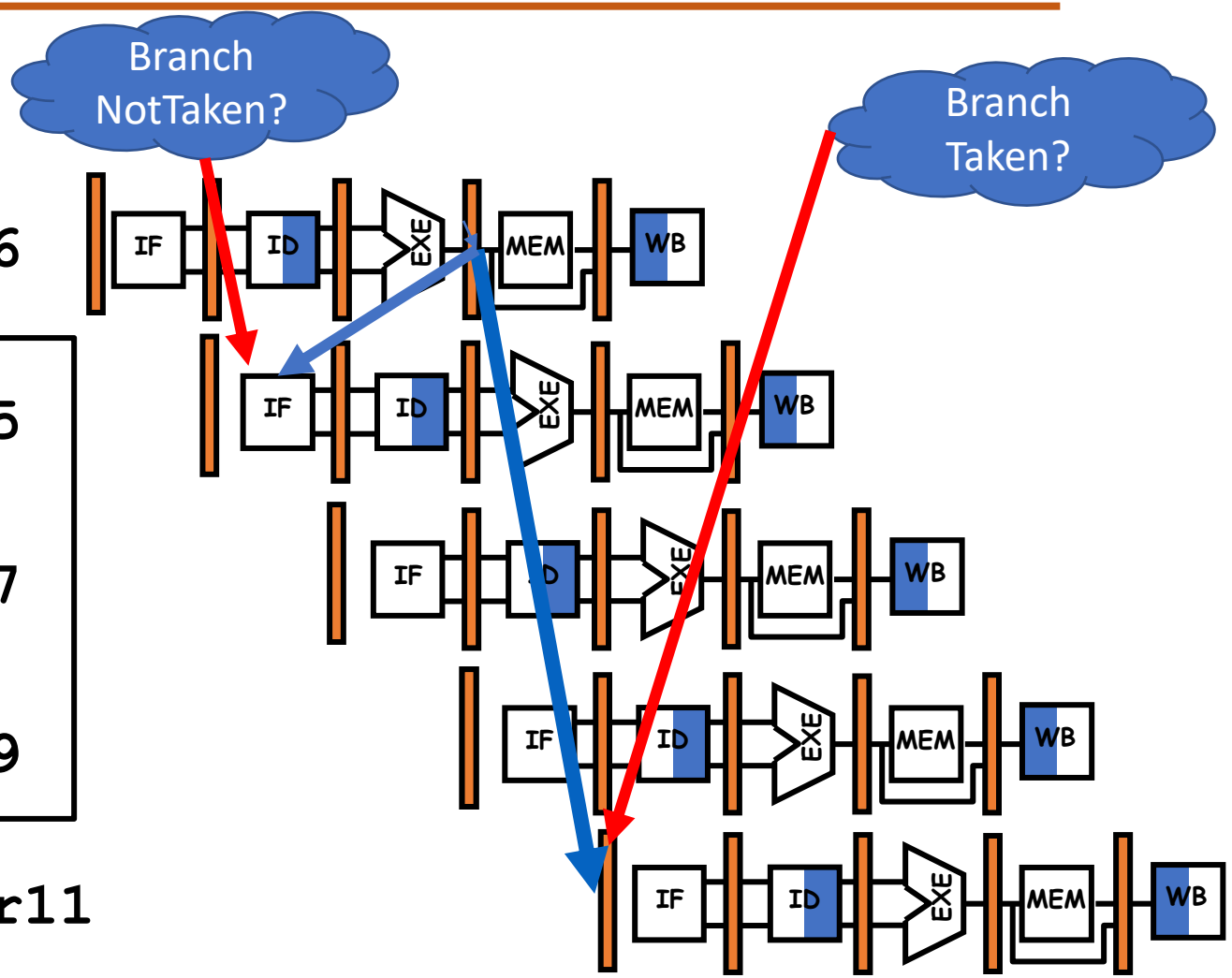
```
add   R1, R2, R3
beq   R1, R2, target
sub   R1, R4, R5
orr   R2, R7, R8

                 ....
target:          add      R1, R4, R5
```



Control Hazard on Branches: Three Stage Stall

```
10: beq r1,r3,36
14: and r2,r3,r5
18: or  r6,r1,r7
22: add r8,r1,r9
36: xor r10,r1,r11
```



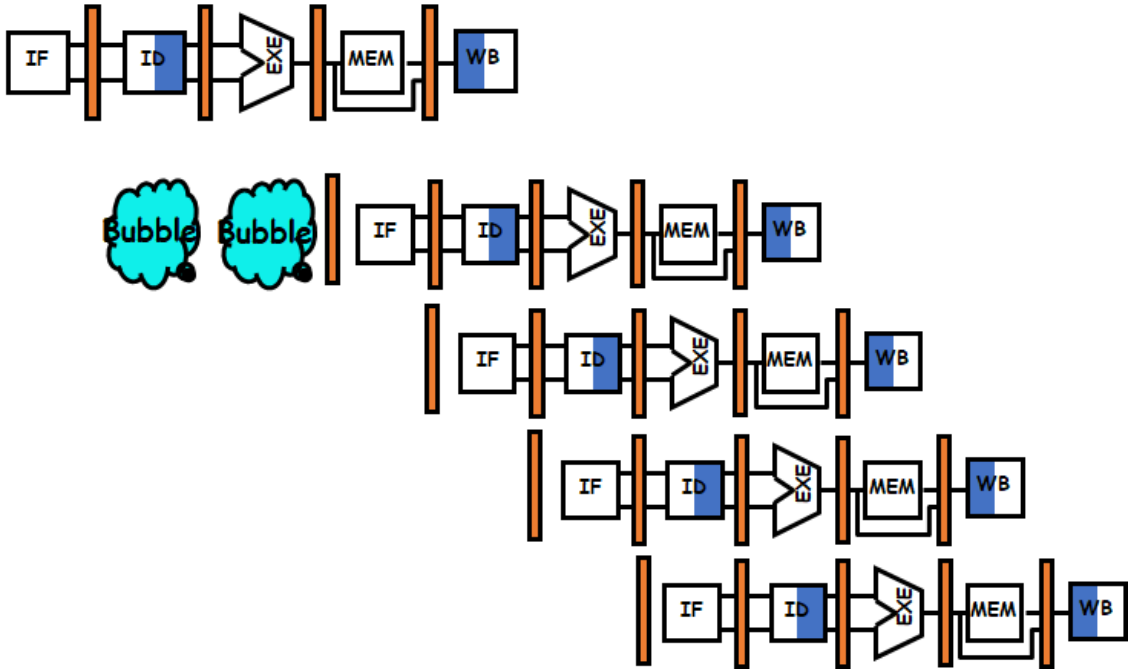
All branches waste 2 cycles irrespective of whether branch is taken or not.

Wasteful..

Better way???

- Reduce stall cycles
- Guess or Predict

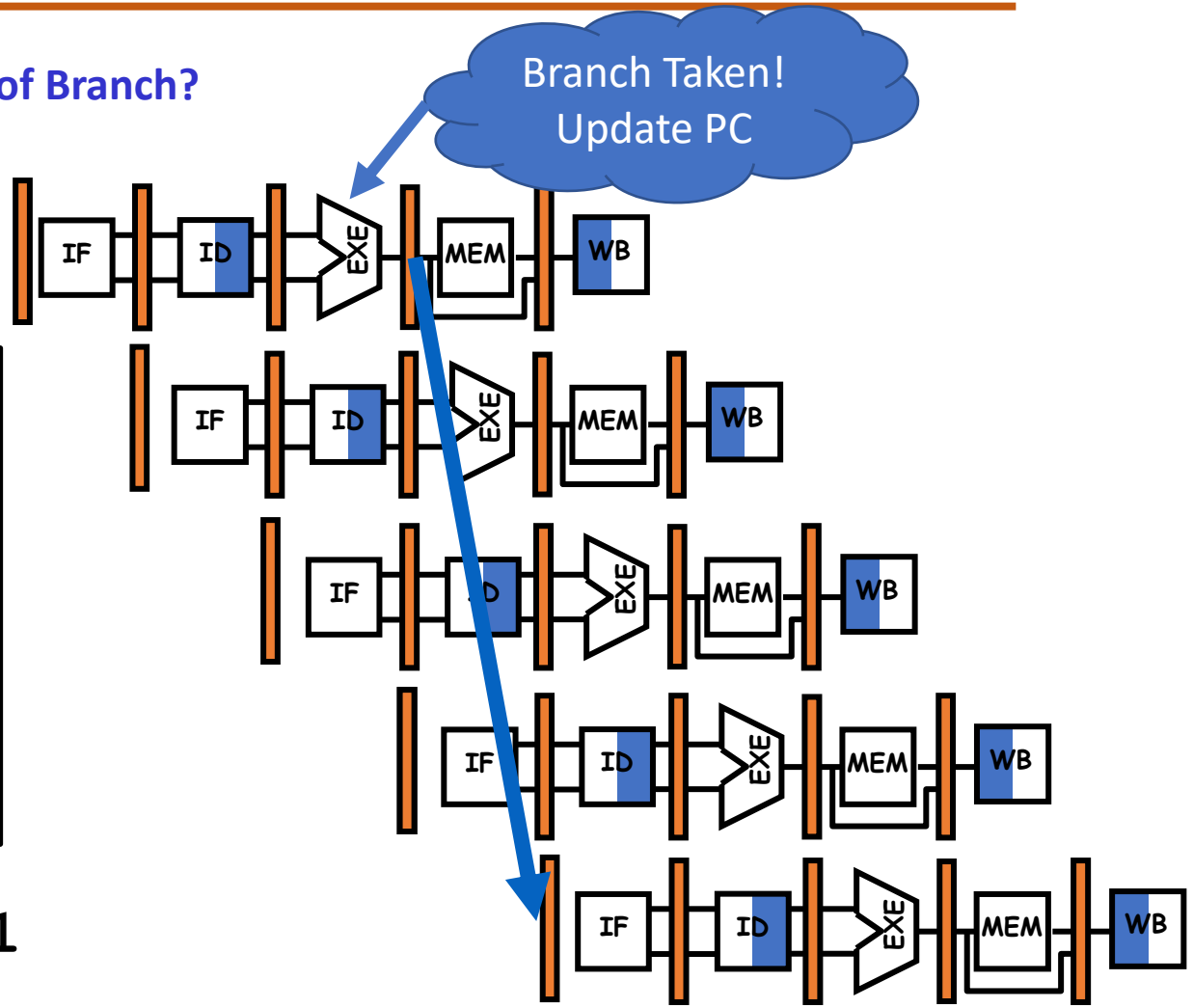
```
10: beq r1,r3,36
14: and r2,r3,r5
18: or  r6,r1,r7
22: add r8,r1,r9
36: xor r10,r1,r11
```



Control Hazard on Branches

When Do I know the result of Branch?
When Do I update PC?

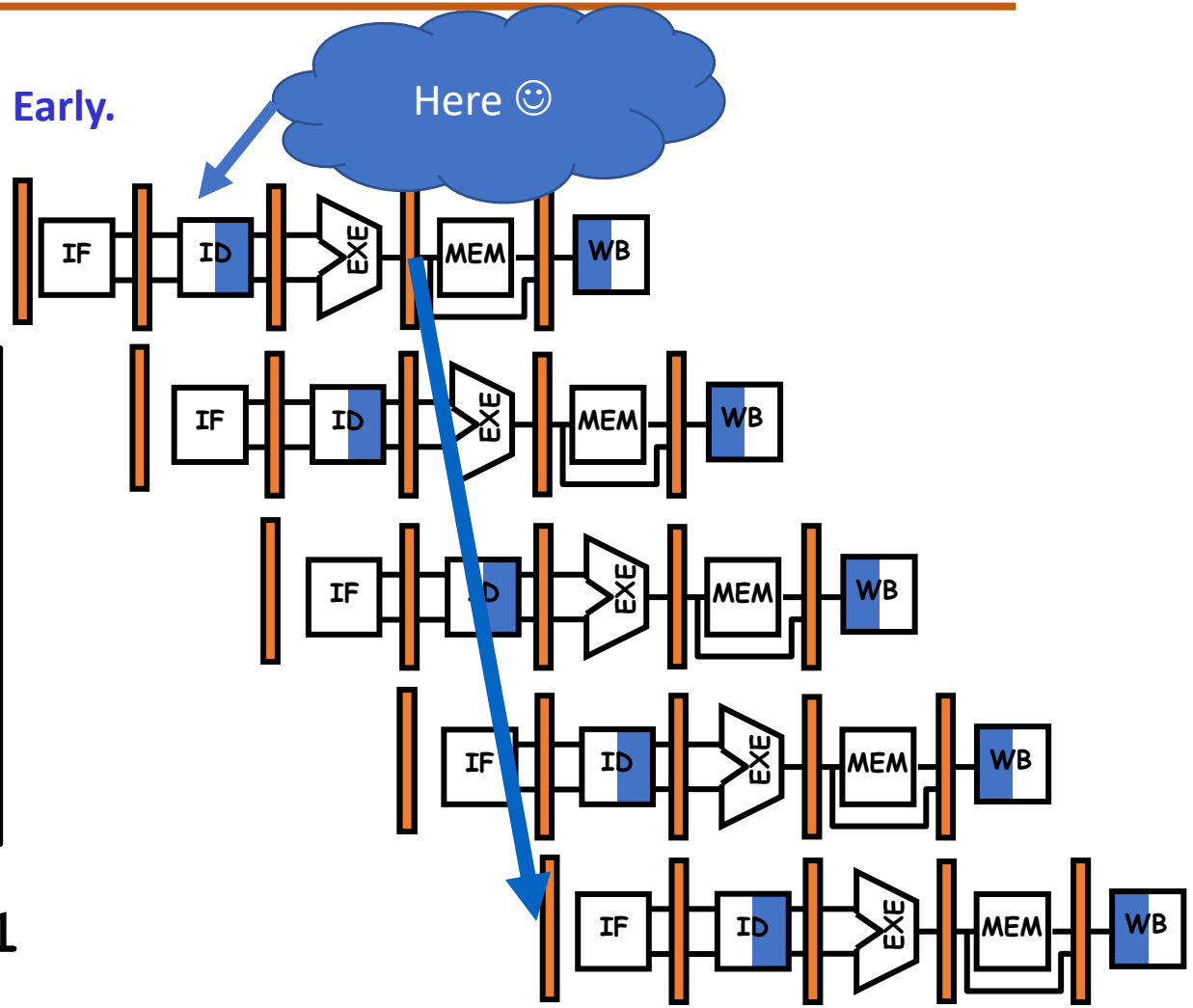
```
10: beq r1,r3,36
14: and r2,r3,r5
18: or  r6,r1,r7
22: add r8,r1,r9
36: xor r10,r1,r11
```



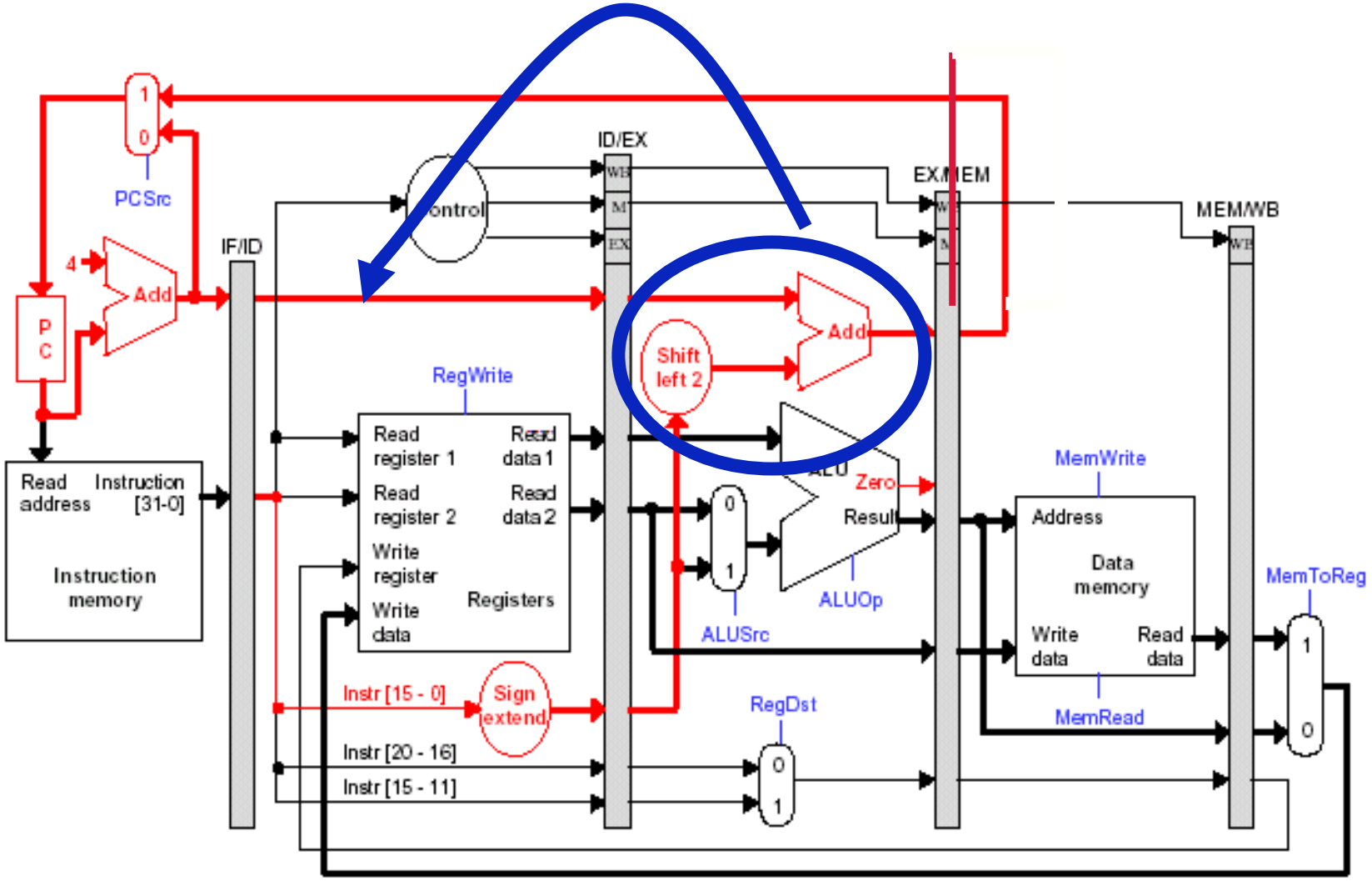
Control Hazard on Branches

Can I Reduce the Stalls by
Computing Result of Branch Early.
Update the PC Early.

```
10: beq r1,r3,36
14: and r2,r3,r5
18: or  r6,r1,r7
22: add r8,r1,r9
36: xor r10,r1,r11
```

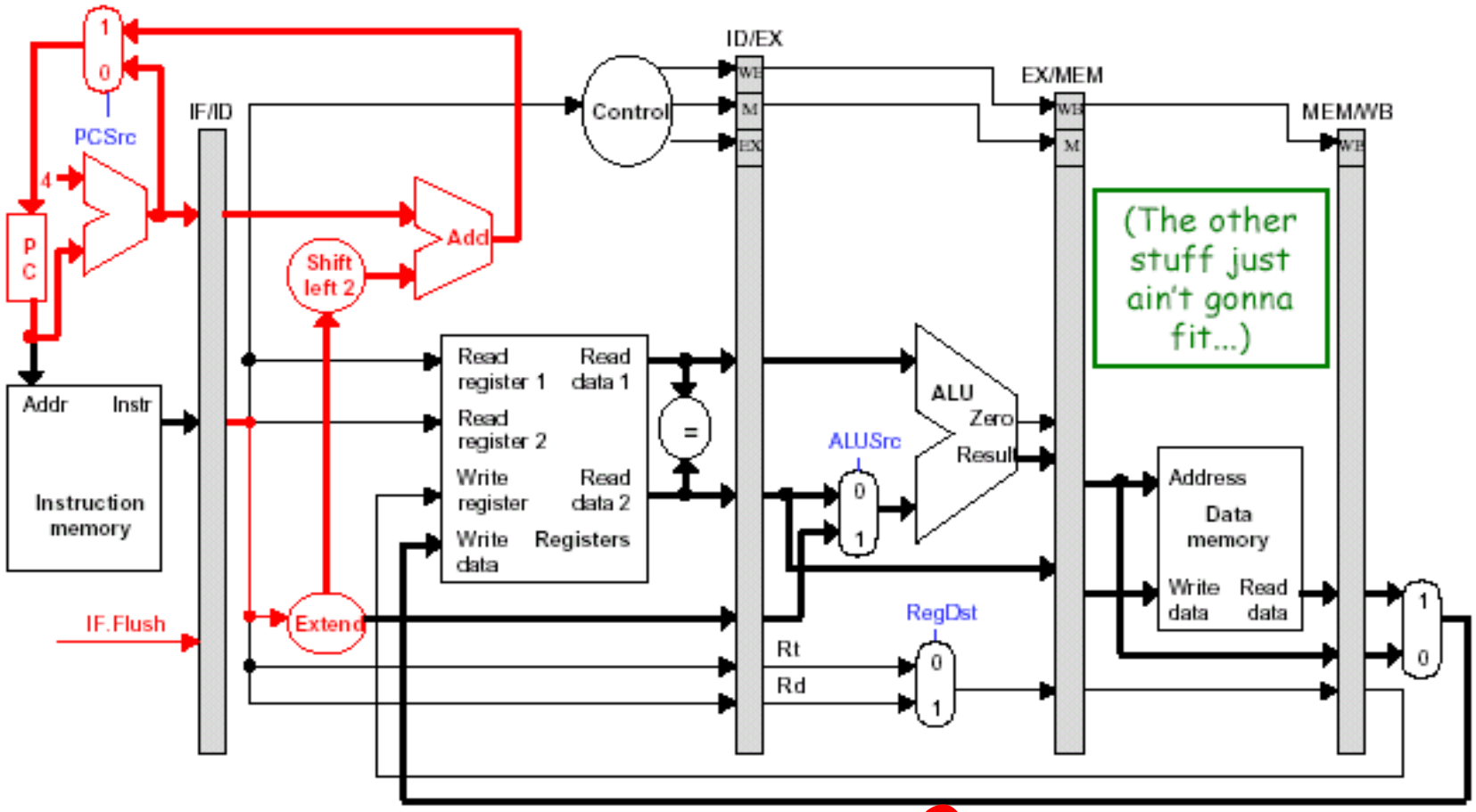


Reduce Stall Cycles?



- By moving updating of PC from EX to ID???
- Branch Delay = 1 cycle

Reduce Stall Cycles



- By moving updating of PC from EX to ID???
- Branch Delay = 1 cycle

- **Can we reduce it further by moving the updating of PC from ID to IF???** NO

Example

- Assume the following instruction mix:

Type	Frequency	
Arith/Logic	40%	
Load	30%	of which 25% are followed immediately by an instruction using the loaded value
Store	10%	
branch	20%	of which 45% are taken

- What is the resulting CPI for the pipelined processor with forwarding and branch address calculation in ID stage?

- CPI = Ideal CPI + Pipeline stall clock cycles per instruction

  = 1 + stalls by loads + stalls by branches

  = 1 + .3x.25x1 + .2 x .45x1

  = 1 + .075 + .09

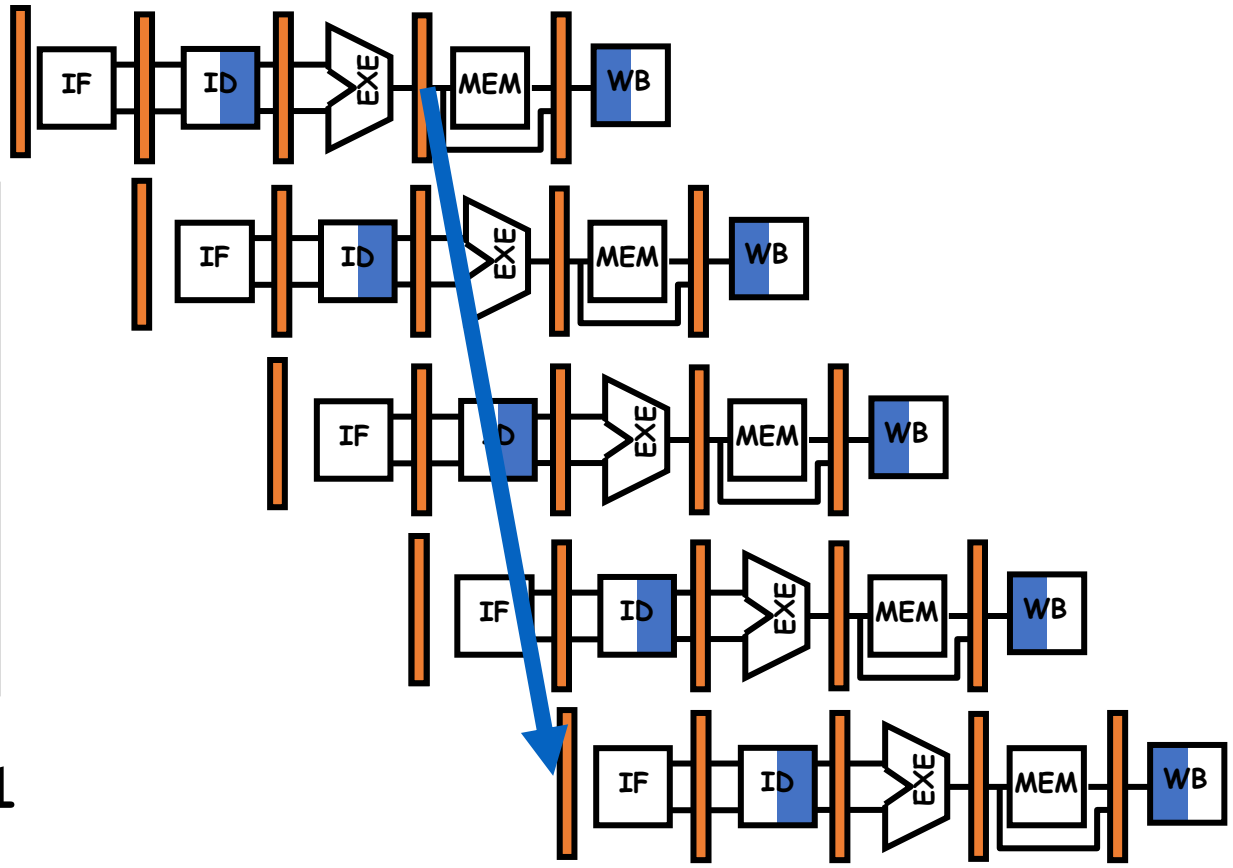
  = 1.165

Delayed Branching

- Similar to *insert no-ops*
- Compiler inserts….??
- Used to eliminate branch stalls
- Instruction after branch is known as delay slot
- Instruction in delay slot is always executed
- Fill the slots with useful instructions

Control Hazard on Branches: Delay Slot

```
10: beq r1,r3,36
14: and r2,r3,r5
18: or  r6,r1,r7
22: add r8,r1,r9
36: xor r10,r1,r11
```

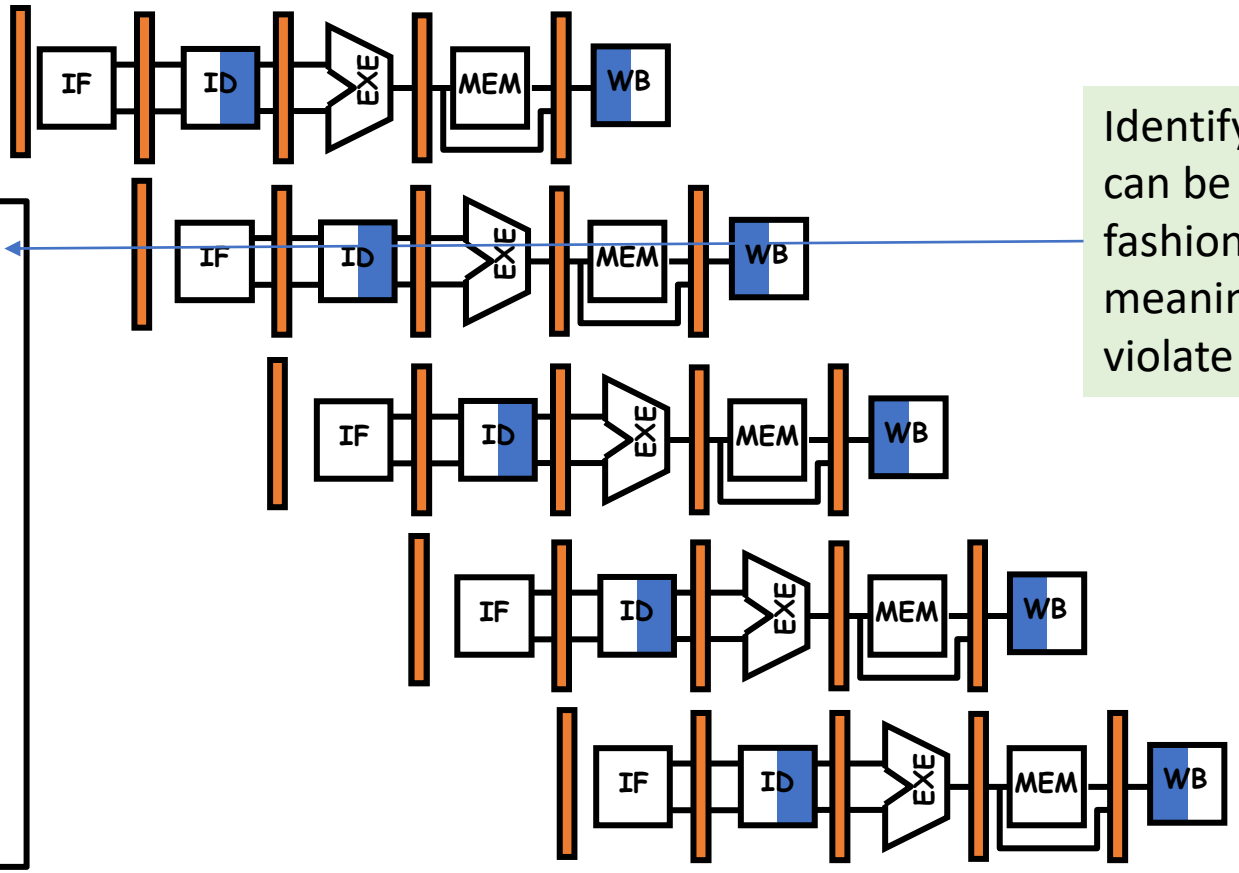



```
48: and r3,r1,r4
```

Identify the Instruction which can be executed out of order fashion i.e will not change the meaning of the program or not violate register write

Control Hazard on Branches: Delay Slot

```
10: beq r1,r3,36
48: sub r3,r1,r4
14: and r2,r3,r5
18: or  r6,r1,r7
22: add r8,r1,r9
36: xor r10,r1,r11
```



Identify the Instruction which can be executed out of order fashion i.e will not change the meaning of the program or not violate register write

A: Filling the delay slot Before Branch

- MUL R3, R4, R5

SUB R2, R1, R0

ADD R1, R2, R2

BEQZ R1, R7, there

ADD R1, R4, R7

there:

Delay Slot

```
# SUB R2, R1, R0
  ADD R1, R2, R2
  BEQZ R1, R7, there
  MUL R3, R4, R5
  ADD R1, R4, R7

there:
```

```
# In case, the compiler is
  not able to find suitable
  instruction, then
# MUL R3, R4, R5
  SUB R2, R1, R0
  ADD R1, R2, R2
  BEQZ R1, R7, there
  NOP
  ADD R1, R4, R7

there:
```

B: Filling the delay slot From Branch Target

```
SUB     R2,  R1, R0
ADD     R1, R2, R2
BEQZ    R1, R7, there
```

Delay Slot

```
ADD     R1, R4,  R7
```

Branch Not Taken Part

```
there: MUL R3, R4, R5
```

Branch Taken Part

```
SUB R2, R1, R0
ADD R1, R2, R2
BEQZ R1, R7,  there
MUL  R3, R4, R5
ADD R1, R4,  R7

there:
```

- Useful when it is predicted, 80 % Branch Taken & 20% Not Taken
- It should ensure no Register write violation in the Not Taken part of the instructions
- If Branch is not taken, then FLUSH i.e Waste of Cycle time

Microprocessor & Computer Architecture (µpCA)

C: Filling the delay slot From Fall through

```
SUB     R2,  R1, R0
ADD     R1, R2, R2
BEQZ    R1, R7, there
MUL R3, R4, R5
ADD     R1, R4,  R7
```

Branch Not Taken Part

```
there:
```

Branch Taken Part

Delay Slot

```
#   SUB R2, R1, R0
    ADD R1, R2, R2
    BEQZ R1, R7,  there
    MUL  R3, R4, R5
    ADD R1, R4,  R7

there:
```

- Useful when it is predicted, 80 % Branch Not Taken & 20% Taken
- It should ensure no Register write violation in the Not Taken part of the instructions
- If Branch is not taken, then FLUSH i.e Waste of Cycle time

Next Session

Branch Prediction

THANK YOU

Dr. D. C. Kiran
Department of Computer Science and Engineering

dckiran@pes.edu

9829935135