# Computer Architecture
# (Pipelined Processor Design)

विद्याधनं सर्वधन प्रधानम्

**Subhasis Bhattacharjee**
Department of Computer Science and Engineering,
Indian Institute of Technology, Jammu

September 18, 2023

## Outline

## Overview of Pipelining

## Up till now

- We have designed a processor that can execute all the SimpleRisc Instructions
- We have look at two styles :
- With a hardwired control unit
- Microprogrammed control unit
- Microprogrammed data path
- Microassembly Language
- Microinstructions4

## Designing Efficient Processors

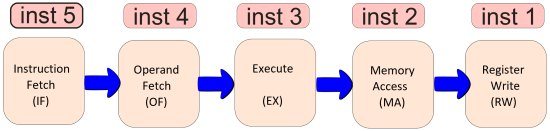
- Microprogrammed processors are much slower that hardwired processors
- Even hardwired processors
- Have a lot of waste !!!
- We have 5 stages.
- What is the IF stage doing, when the MA stage is active ?
- ANSWER : It is idling5

## The Notion of Pipelining

- Let us go back to the car assembly line
- Is the engine shop idle, when the paint shop is painting a car ?
- NO : It is building the engine of another car
- When this engine goes to the body shop, it builds the engine of another car, and so on . . . .
- Insight :
- Multiple cars are built at the same time.
- A car proceeds from one stage to the next6

## Pipelined Processors

- The IF, ID, EX, MA, and RW stages process 5 instructions simultaneously
- Each instruction proceeds from one stage to the next
- This is known as pipelining



## Advantages of Pipelining

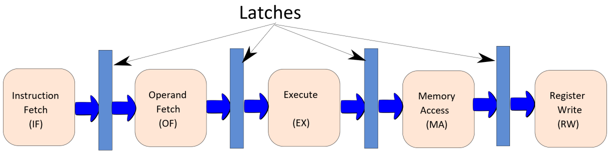
- We keep all parts of the data path, busy all the time
- Let us assume that all the 5 stages do the same amount of work
- Without pipelining, every T seconds, an instruction completes its execution
- With pipelining, every T/5 seconds, a new instruction completes its execution8

## Design of a Pipeline

- Splitting the Data Path
- We divide the data path into 5 parts : IF, OF, EX, MA, and RW
- Timing
- We insert latches (registers) between consecutive stages
- 4 Latches → IF-OF, OF-EX, EX-MA, and MA-RW
- At the negative edge of a clock, an instruction moves from one stage to the next9

## Pipelined Data Path with Latches

- Add a latch between subsequent stages.
- Triggered by a negative clock edge

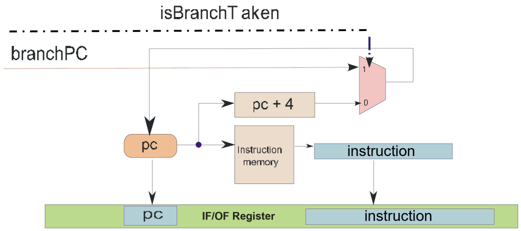


## The Instruction Packet

- What travels between stages ?
- ANSWER : the instruction packet
- Instruction Packet
- Instruction contents
- Program counter
- All intermediate results
- Control signals
- Every instruction moves with its entire state, no interference between instructions11

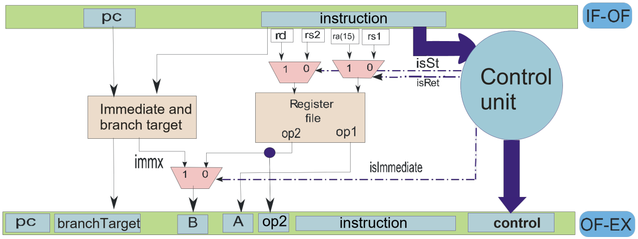
## A Pipelined Data Path

## IF Stage



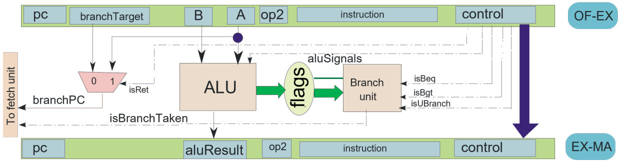
- Instruction contents saved in the instruction field IF/OF Register instruction

## OF Stage



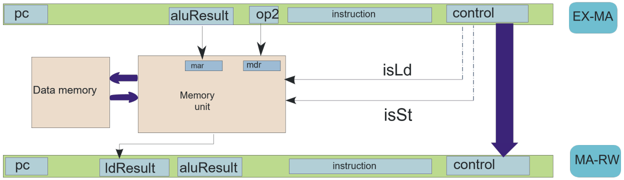
- A, B →ALU Operands, op2 (store operand), control (set of all control signals)

## EX Stage



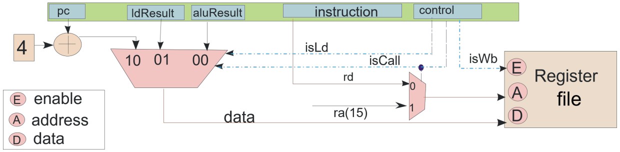
- aluResult $\rightarrow$result of the ALU Operation
- op2, control, pc, instruction (passed from OF-EX)

# MA Stage



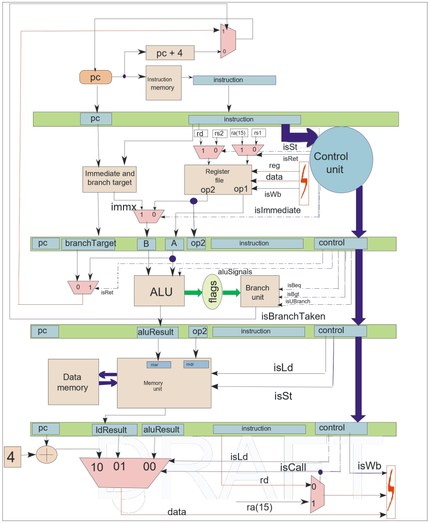
- ldResult →result of the load operation
- aluResult, control, pc, instruction (passed from EX-MA)

## RW Stage

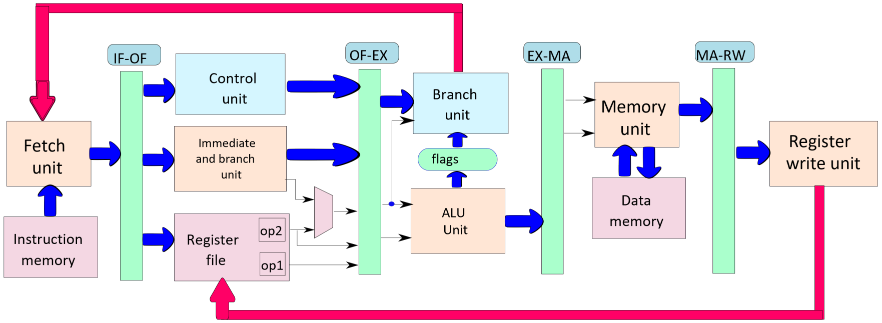

pc
ldResult
aluResult
instruction
control
4
10
01
00
isLd
isCall
isWb
rd
ra(15)
0
1
data
Register file
E enable
A address
D data
E
A
D

●

# Internal Details of All 5 Stages With Latches



# Abridged Diagram



•

## Pipeline Hazards

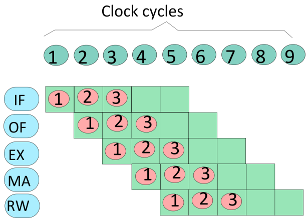
## Pipeline Hazards

- Now, let us consider correctness
- Let us introduce a new tool →Pipeline

[1]: add r1, r2, r3

[2]: sub r4, r5, r6

[3]: mul r8, r9, r10



## Rules for Constructing a Pipeline Diagram

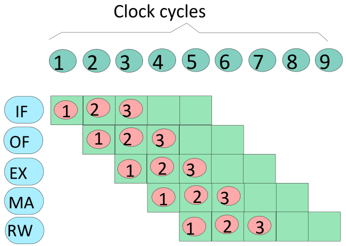
- It has 5 rows
- One per each stage
- The rows are named : IF, OF, EX, MA, and RW
- Each column represents a clock cycle
- Each cell represents the execution of an instruction in a stage
- It is annotated with the name(label) of the instruction
- Instructions proceed from one stage to the next across clock cycles

## Example

[1]: add r1, r2, r3

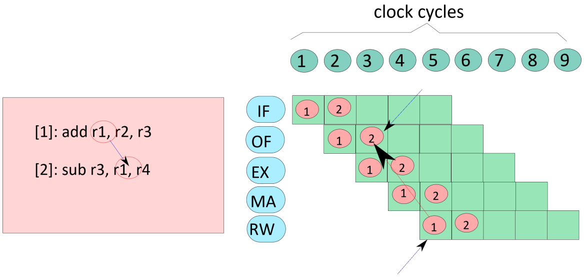
[2]: sub r4, r2, r5

[3]: mul r5, r8, r9



-

# Data Hazards



[1]: add r1, r2, r3

[2]: sub r3, r1, r4

- Instruction 2 will read incorrect values !!!

## Data Hazard

- Definition: A hazard is defined as the possibility of erroneous execution of an instruction in a pipeline. A data hazard represents the possibility of erroneous execution because of the unavailability of data, or the availability of incorrect data.
- This situation represents a data hazard
- In specific,
- it is a RAW (read after write) hazard
- The earliest we can dispatch instruction 2, is cycle 525

## Other Types of Data Hazards

- Our pipeline is in-order
- We will only have RAW hazards in our pipeline.
- Out-of-order pipelines can have WAR and

## WAW hazards

- Definition: In an in-order pipeline (such as ours), a preceding instruction is always ahead of a succeeding instruction in the pipeline. Modern processors however use out-of-order pipelines that break this rule. It is possible for later instructions to execute before earlier instructions.26

## WAW Hazards

- Instruction [2] cannot write the value of r1, before instruction [1] writes to it, will lead to a WAW hazard [1]: add r1, r2, r3 [2]: sub r1, r4, r327

## WAR Hazards

- Instruction [2] cannot write the value of r2, before instruction [1] reads it $\rightarrow$will lead to a WAR hazard [1]: add r1, r2, r3 [2]: add r2, r5, r628

## Control Hazards

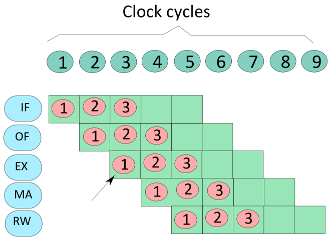
- If the branch is taken, instructions [2] and [3], might get fetched, incorrectly [1]: beq .foo [2]: mov r1, 4 [3]: add r2, r4, r3 ... ... .foo: [100]: add r4, r1, r229

## Control Hazard – Pipeline Diagram

```
[1]: beq .foo

[2]: mov r1, 4

[3]: add r2, r4, r3
```



- The two instructions fetched immediately after a branch instruction might have been fetched incorrectly.

## Control Hazards

- The two instructions fetched immediately after a branch instruction might have been fetched incorrectly.
- These instructions are said to be on the wrong path
- A control hazard represents the possibility of erroneous execution in a pipeline because instructions in the wrong path of a branch can possibly get executed and save their results in memory, or in the register file

## Structural Hazards

- A structural hazard may occur when two instructions have a conflict on the same set of resources in a cycle
- Example :
- Assume that we have an add instruction that can read one operand from memory
- add r1, r2, 10[r3]32

## Structural Hazards - II

- This code will have a structural hazard

3 tries to read 10[r3] (MA unit) in cycle 4

1 tries to write to 20[r5] (MA unit) in cycle 4

- Does not happen in our pipeline [1]: st r4, 20[r5] [2]: sub r8, r9, r10 [3]: add r1, r2, 10[r3]33

## Solutions in Software

- Data hazards
- Insert nop instructions, reorder code [1]: add r1, r2, r3 [2]: sub r3, r1, r4 [1]: add r1, r2, r3 [2]: nop [3]: nop [4]: nop [5]: sub r3, r1, r434

## Code Reordering

- add r1, r2, r3 add r4, r1, 3 add r8, r5, r6 add r9, r8, r5 add r10, r11, r12 add r13, r10, 2 add r1, r2, r3 add r8, r5, r6 add r10, r11, r12 nop add r4, r1, 3 add r9, r8, r5 add r13, r10, 235

## Control Hazards

- Trivial Solution : Add two nop instructions after every branch
- Better solution :
- Assume that the two instructions fetched after a branch are valid instructions
- These instructions are said to be in the delay slots
- Such a branch is known as a delayed branch36

## Example with 2 Delay Slots

- The compiler transfers instructions before the branchto the delay slots.
- If it cannot find 2 valid instructions, it inserts nops. add r1, r2, r3 add r4, r5, r6 b .foo add r8, r9, r10 b .foo add r1, r2, r3 add r4, r5, r6 add r8, r9, r1037

## Pipeline with Interlocks

## Why interlocks ?

- We cannot always trust the compiler to do a good job, or even introduce nop instructions correctly.
- Compilers now need to be tailored to specific hardware.
- We should ideally not expose the details of the pipeline to the compiler (might be confidential also)
- Hardware mechanism to enforce correctness $\rightarrow$interlock

## Two kinds of Interlocks

- Data-Lock
- Do not allow a consumer instruction to move beyond the OF stage till it has read the correct values. Implication : Stall the IF and OF stages.
- Branch-Lock
- We never execute instructions in the wrong path.
- The hardware needs to ensure both these conditions.40

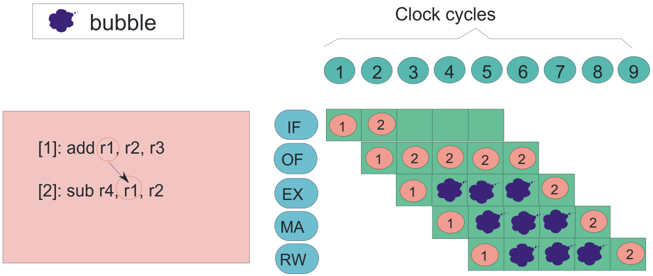
## Comparison between Software and Hardware

- TABLE Attribute Software Hardware(withinterlocks) Portability Limited to a specific processor Programs can be run on any processor irrespective of the nature of the pipeline Branches Possible to have no performance penalty, by using delay slots RAW hazards Possible to eliminate them through code scheduling Need to stall the pipeline Performance Highly dependent on the nature of the program The basic version of a pipeline with interlocks is expected to be slower than the version that relies on software Need to stall the pipeline for 2 cycles in our design41

## Conceptual Look at Pipeline with Interlocks

- We have a RAW hazard
- We need to stall, instruction [2] at the OF stage for 3 cycles.
- We need to keep sending nop instructions to the EX stage during these 3 cycles [1]: add r1, r2, r3 [2]: sub r4, r1, r242

## Example



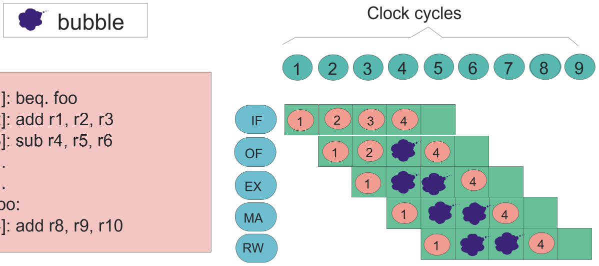
bubble
Clock cycles
1
2
3
4
5
6
7
8
9
[1]: add r1, r2, r3
[2]: sub r4, r1, r2
IF
OF
EX
MA
RW

-

## A Pipeline Bubble

- A pipeline bubble is inserted into a stage, when the previous stage needs to be stalled
- It is a nop instruction
- To insert a bubble
- Create a nop instruction packet
- OR, Mark a designated bubble bit to 144

# Bubbles in the Case of a Branch Instruction



```
[1]: beq. foo
[2]: add r1, r2, r3
[3]: sub r4, r5, r6
....
....
.foo:
[4]: add r8, r9, r10
```

-

## Control Hazards and Bubbles

- We know that an instruction is a branch in the OF stage
- When it reaches the EX stage and the branch is taken, let us convert the instructions in the IF, and OF stages to bubbles
- Ensures the branch-lock condition46

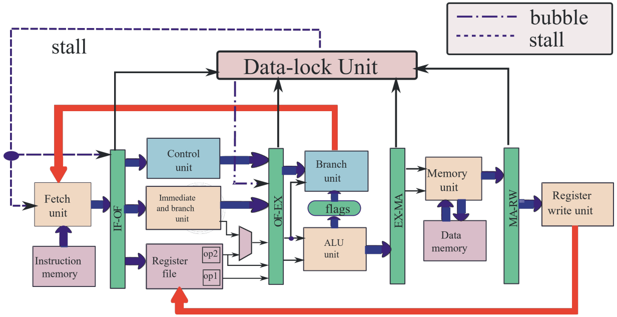
## Ensuring the Data-Lock Condition

- When an instruction reaches the OF stage, check if it has a conflict with any of the instructions in the EX, MA, and RW stages
- If there is no conflict, nothing needs to be done
- Otherwise, stall the pipeline (IF and OF stages only)47

## How to Stall a Pipeline ?

- Disable the write functionality of :
- The IF-OF register
- and the Program Counter (PC)
- To insert a bubble
- Write a bubble (nop instruction) into the OF-EX register50

# Data Path with Interlocks (DataLock)



-

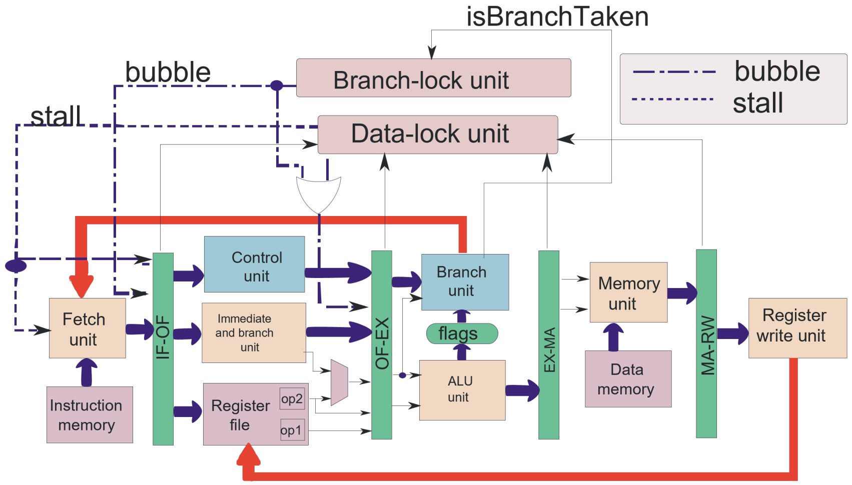
## Ensuring the Branch-Lock Condition

- Option 1 :
- Use delay slots (interlocks not required)
- Option 2 :
- Convert the instructions in the IF, and OF stages, to bubbles once a branch instruction reaches the EX stage.
- Start fetching from the next PC (not taken) or the branch target (taken)52

## Ensuring the Branch-Lock Condition - II

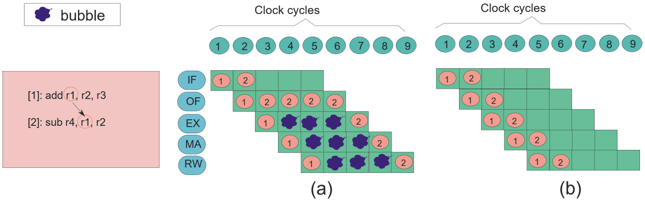
- Option 3
- If the branch instruction in the EX stage is taken, then invalidate the instructions in the IF and OF stages. Start fetching from the branch target.
- Otherwise, do not take any special action
- This method is also called predict not-taken (it is more efficient that option 2)53

# Data Path with Interlocks



## Forwarding

# Relook at the Pipeline Diagram

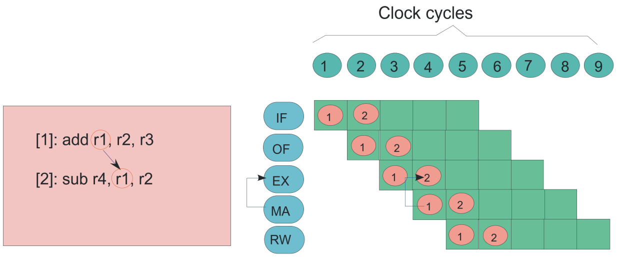


- (a) $\rightarrow$with bubbles
- (b) $\rightarrow$no bubbles (may lead to wrong results)

## Crucial Insight (Figure (b))

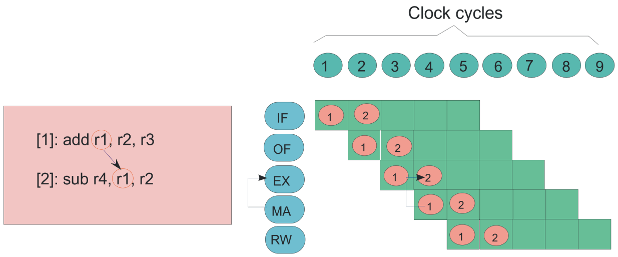
- When does instruction 2 need the value of r1 ?
- ANSWER : Cycle 3, OF Stage (wrong!!!)
- CORRECT ANSWER : Cycle 4, EX Stage
- When does instruction 1 produce the value of r1 ?
- ANSWER : Cycle 5, RW Stage (wrong!!!)
- CORRECT ANSWER : End of Cycle 3, EX Stage57

## Forwarding



- If the correct value is already there in another stage, we can forward it.

## Forwarding from MA to EX



- Fowarding in cycle 4 from instruction [1]

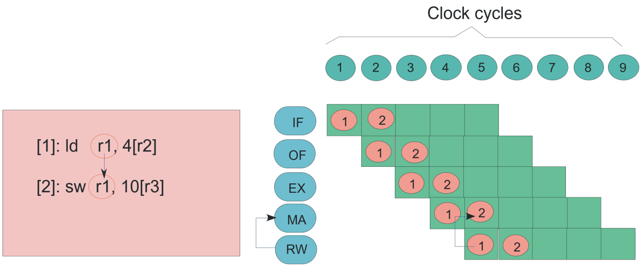
## Different Forwarding Paths

- We need to add a multitude of forwarding paths
- Rules for creating forwarding paths
- Add a path from a later stage to an earlier stage
- Try to add a forwarding path as late as possible. For, example, we avoid the EX $\rightarrow$OF forwarding path, since we have the MA $\rightarrow$EX forwarding path
- The IF stage is not a part of any forwarding path.60

## Forwarding Path

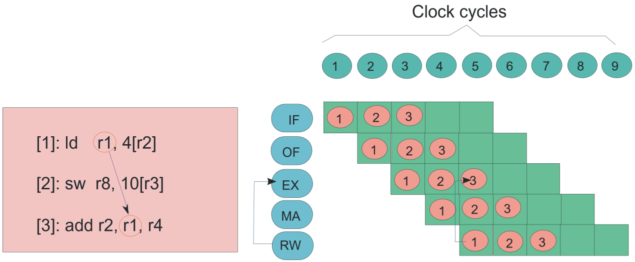
- 3 Stage Paths
- RW $\rightarrow$OF
- 2 Stage Paths
- RW $\rightarrow$EX
- MA $\rightarrow$OF (X Not Required)
- 1 Stage Paths
- RW $\rightarrow$MA (load to store)
- MA $\rightarrow$EX (ALU Instructions, load, store)
- EX $\rightarrow$OF (X Not Required)61

# Forwarding Paths : RW →MA



-

# Forwarding Paths : RW →EX



[1]: ld r1, 4[r2]

[2]: sw r8, 10[r3]

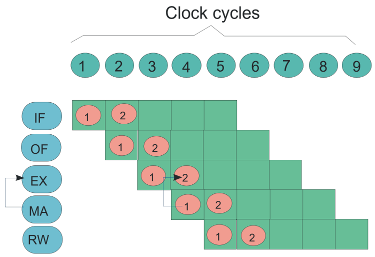
[3]: add r2, r1, r4

-

# Forwarding Path : MA →EX

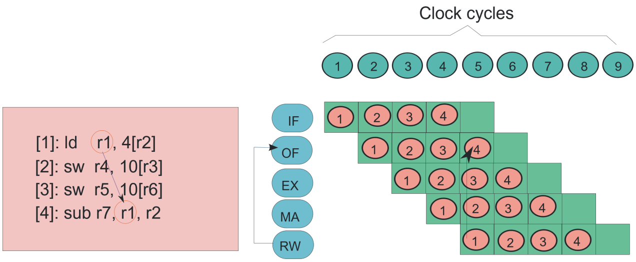
[1]: add r1, r2, r3

[2]: sub r4, r1, r2



-

## Forwarding Path : RW $\rightarrow$OF

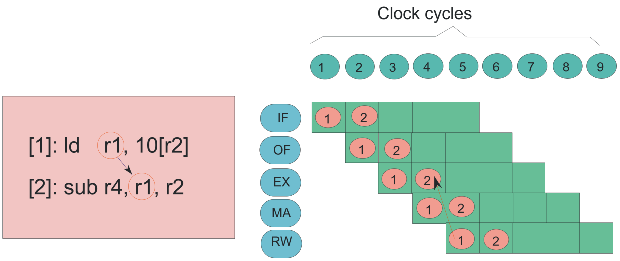


-

## Data Hazards with Forwarding

- Forwarding has unfortunately not eliminated all data hazards
- We are left with one special case.
- Load-use hazard
- The instruction immediately after a load instruction has a RAW dependence with it.66

# Load-Use Hazard



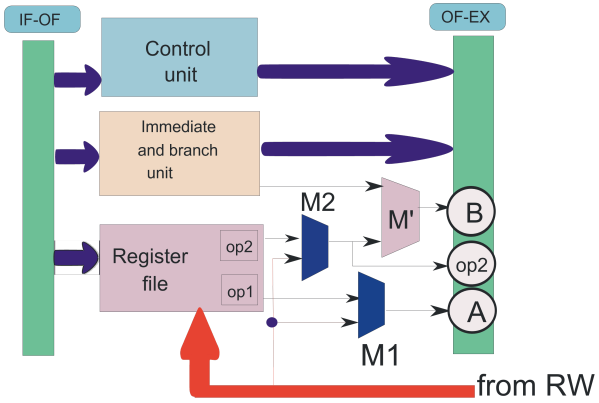
- Cannot forward (arrow goes backwards in time)
- Need to add a bubble (then use RW →EX forwarding)

## Implementation of Forwarding

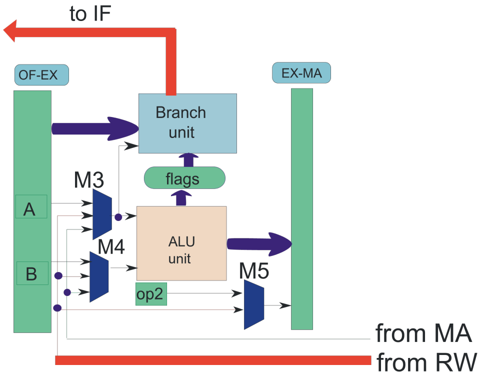
## Implementation of Forwarding

- At every stage there is a choice of inputs for each functional unit
- Use the default inputs from the previous stage
- OR, use one of the forwarded inputs
- Use a multiplexer to choose between the inputs
- A dedicated forwarding unit generates the signals for the forwarding multiplexers68

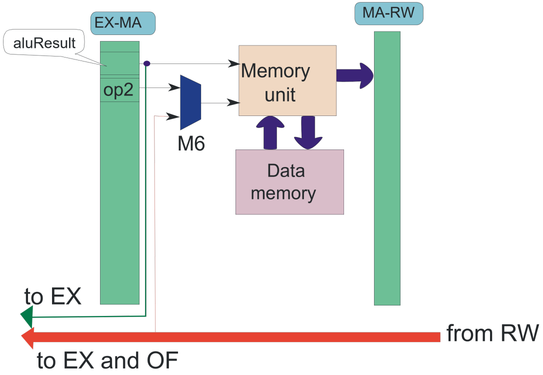
## OF Stage with Forwarding



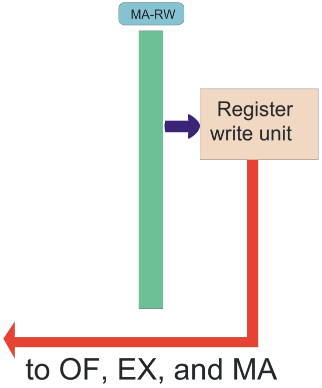
# EX Stage with Forwarding



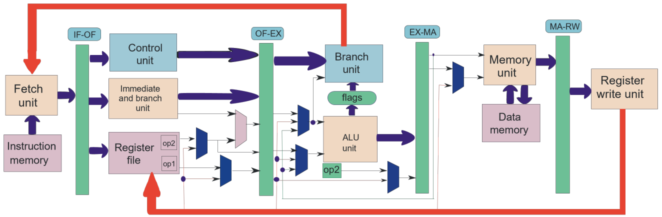
# MA Stage with Forwarding



## RW Stage with Forwarding



# Data Path with Forwarding



## Forwarding Conditions

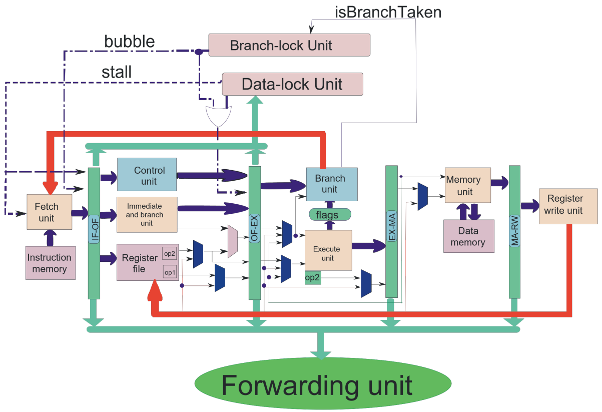
- Determine if there is a conflict between two instructions in different stages
- Find if there is a conflict for the first operand (rs1/ ra)
- Find if there is a conflict for the second operand (rs2/rd)
- Always forward from the latest instruction (that is earlier than the current instruction)

## Interlocks and Forwarding

## Interlocks with Forwarding

- Data-Lock
- We need to only check for the load-use hazard
- If the instruction in the EX stage is a load, and the instruction in the OF stage uses its loaded value, then stall for 1 cycle
- Branch-Lock
- Remains the same as before.78

# Complete Data Path



## Performance Metrics