Performance

- What does it mean when you say “system A is of a higher performance as compared to system B”?

Performance

- What does it mean when you say “system A is of a higher performance as compared to system B”?
- “Time taken to perform a certain task T on system A is less than that taken to perform the same task T on system B.”
  - Note that this statement is specific to the task T. For some other task T’, it may be possible that system B is the better performer.

Performance

```
int f = 1;
for(int i=2; i<=n; i++)
{
    f = f * i;
}
cout << f;
```

Program A

```
int f = 1;
for(int i=2; i<=2*n; i++)
{
    if (i <= n)
        f = f * i;
}
cout << f;
```

Program B

- System A: Program A is run on processor type P
- System B: Program B is run on processor type P
- Which system displays higher performance?

Performance

```
.main:
[0]     mov r2, 3
[4]     mov r1, 1
.loop:
[8]     mul r1, r1, r2
[12]    sub r2, r2, 1
[16]    cmp r2, 1
[20]    bgt .loop
[24]    st r1, 4[r0]
[28]    end
```

How many instructions are executed?

Performance

```
.main:
[0]     mov r2, 3
[4]     mov r1, 1
.loop:
[8]     mul r1, r1, r2
[12]    sub r2, r2, 1
[16]    cmp r2, 1
[20]    bgt .loop
[24]    st r1, 4[r0]
[28]    end
```

- Number of static instructions = 8
- Number of dynamic instructions = 12

Performance

```
int f = 1;
for(int i=2; i<=n; i++)
{
    f = f * i;
}
cout << f;
```

Program A

```
int f = 1;
for(int i=2; i<=2*n; i++)
{
    if (i <= n)
        f = f * i;
}
cout << f;
```

Program B

- System A: Program A is run on processor type P
- System B: Program B is run on processor type P
- System A displays higher performance because program A has fewer dynamic instructions (assuming every instruction takes equal amount of time)

SimpleRISC Processor Frequency

- How long does it take to execute a single instruction on our SimpleRISC processor?

SimpleRISC Processor Frequency

- How long does it take to execute a single instruction on our SimpleRISC processor?
- Work out the longest path in the combinational circuitry. Processor clock period should be greater than the time taken to execute this longest path. Let this clock period be t seconds. Processor frequency f = 1/t .
- Every instruction is executed in t s.
- Every instruction is executed in 1 cycle in our SimpleRISC processor.
  - Cycles per instruction, CPI = 1
  - Instructions per cycle, IPC = 1

Computing the Time a Program Takes

$$\tau = \#seconds$$
$$= \frac{\#seconds}{\#cycles} * \frac{\#cycles}{\#instructions} * (\#instructions)$$
$$= \underbrace{\frac{\#seconds}{\#cycles}}_{1/f} + \underbrace{\frac{\#cycles}{\#instructions}}_{CPI} * (\#instructions)$$
$$= \frac{CPI * \#insts}{f}$$

* CPI ⟶ Cycles per instruction
* f ⟶ frequency (cycles per second)

The Performance Equation

$$P \propto \frac{IPC * f}{\#insts}$$

* IPC $\longrightarrow$ 1/CPI (Instructions per Cycle)
* What are the units of performance ?
  * **ANSWER** : arbitrary

Number of Instructions (#insts)

Static Instruction: The binary or executable of a program, contains a list of ***static instructions***.

Dynamic Instruction: A **dynamic instruction** is a running instance of a static instruction, which is created by the processor when an instruction enters the pipeline.

* Note that these are dynamic instructions
  * **NOT** static instructions
* A smart compiler can reduce the number of executed instructions

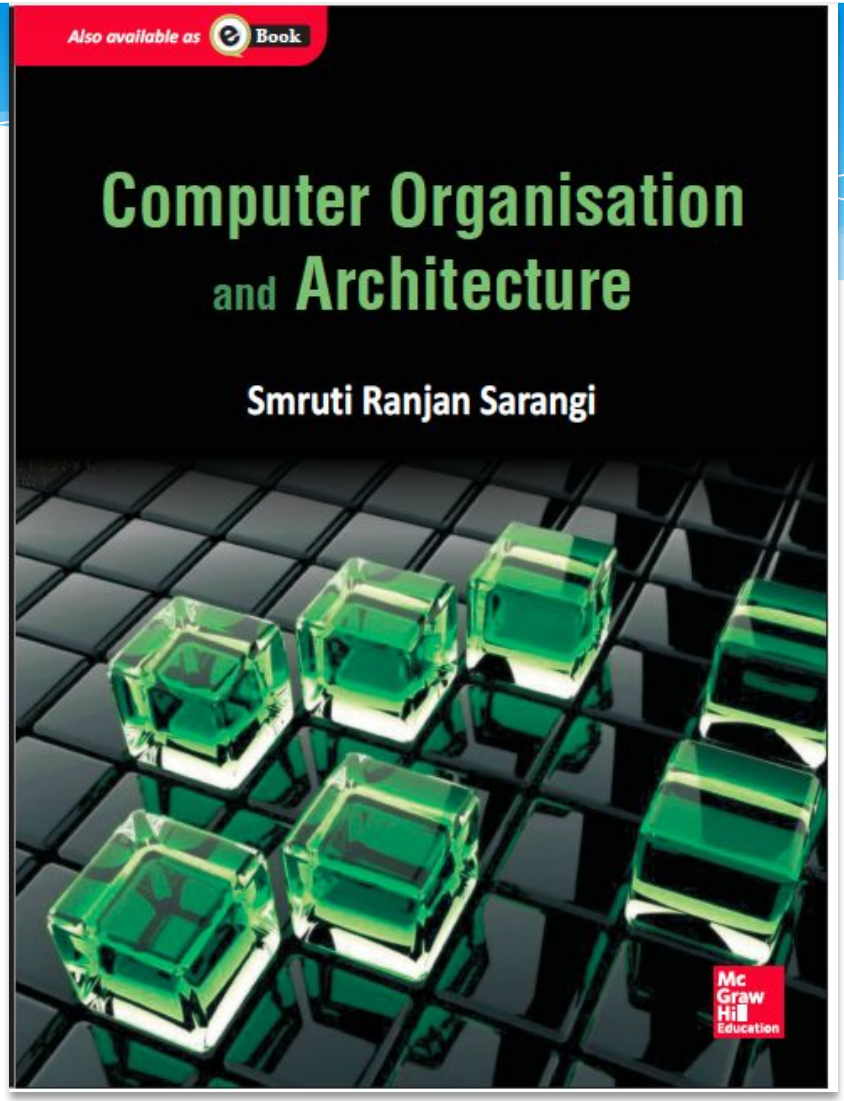
Number of Instructions(#insts) – 2

* Dead code removal
  * Often programmers write code that does not determine the final output
  * This code is redundant
  * It can be identified and removed by the **compiler**
* Function inlining
  * Very small functions have a lot of overhead → call, ret instructions, register spilling, and restoring
  * Paste the code of the callee in the code of the caller (known as **inlining**)

Computer Organisation and Architecture

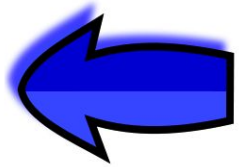
Smruti Ranjan Sarangi,
IIT Delhi

Chapter 9 Principles of Pipelining



These slides are meant to be used along with the book: Computer Organisation and Architecture, Smruti Ranjan Sarangi, McGrawHill 2015
Visit: http://www.cse.iitd.ernet.in/~srsarangi/archbooksoft.html

Outline



* Overview of Pipelining
* A Pipelined Data Path
* Pipeline Hazards
* Pipeline with Interlocks
* Forwarding
* Performance Metrics
* Interrupts/ Exceptions

Mc Graw Hill Education

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
    * Microinstructions

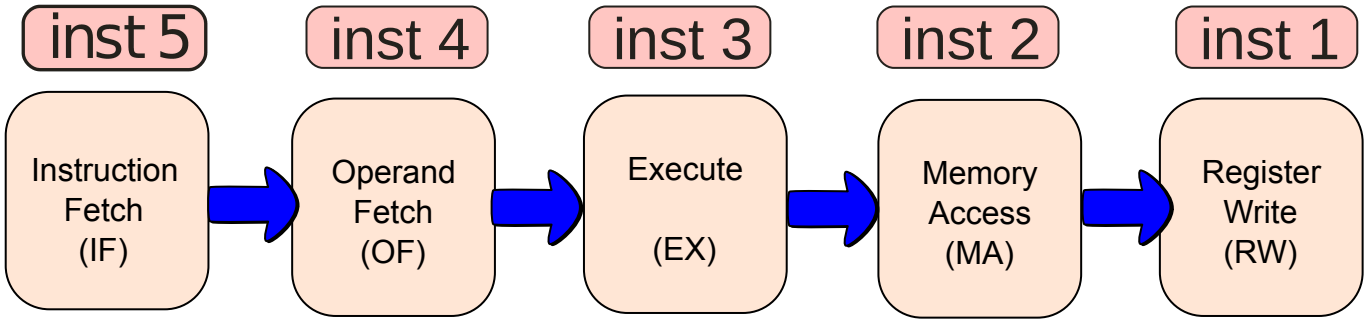
Designing Efficient Processors

* 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 **idling**

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 next

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

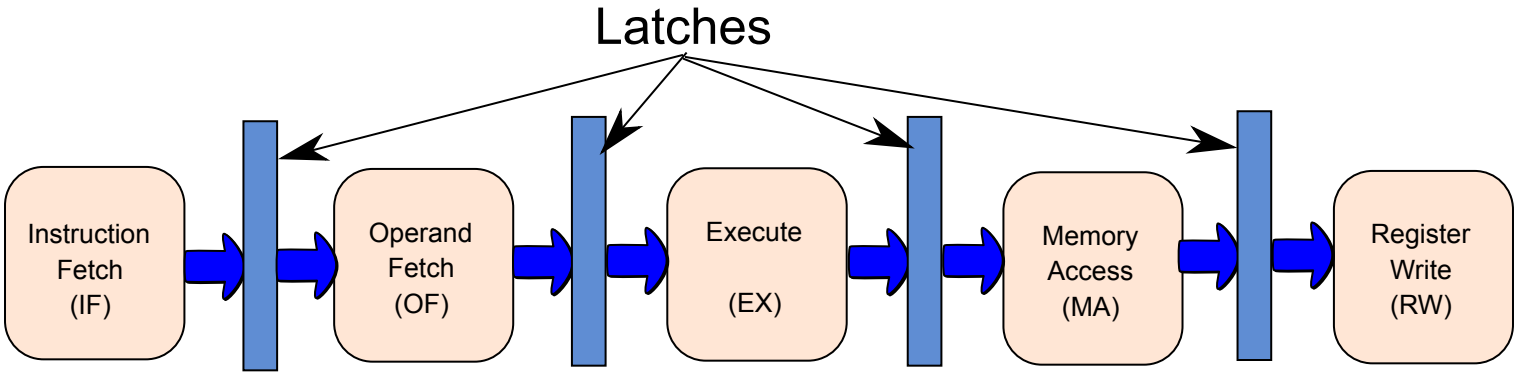
Advantages of Pipelining

* 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 execution

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 next

Pipelined Data Path with Latches



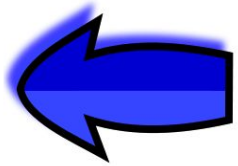
* Add a latch between subsequent stages.
  * Triggered by a negative clock edge

The Instruction Packet

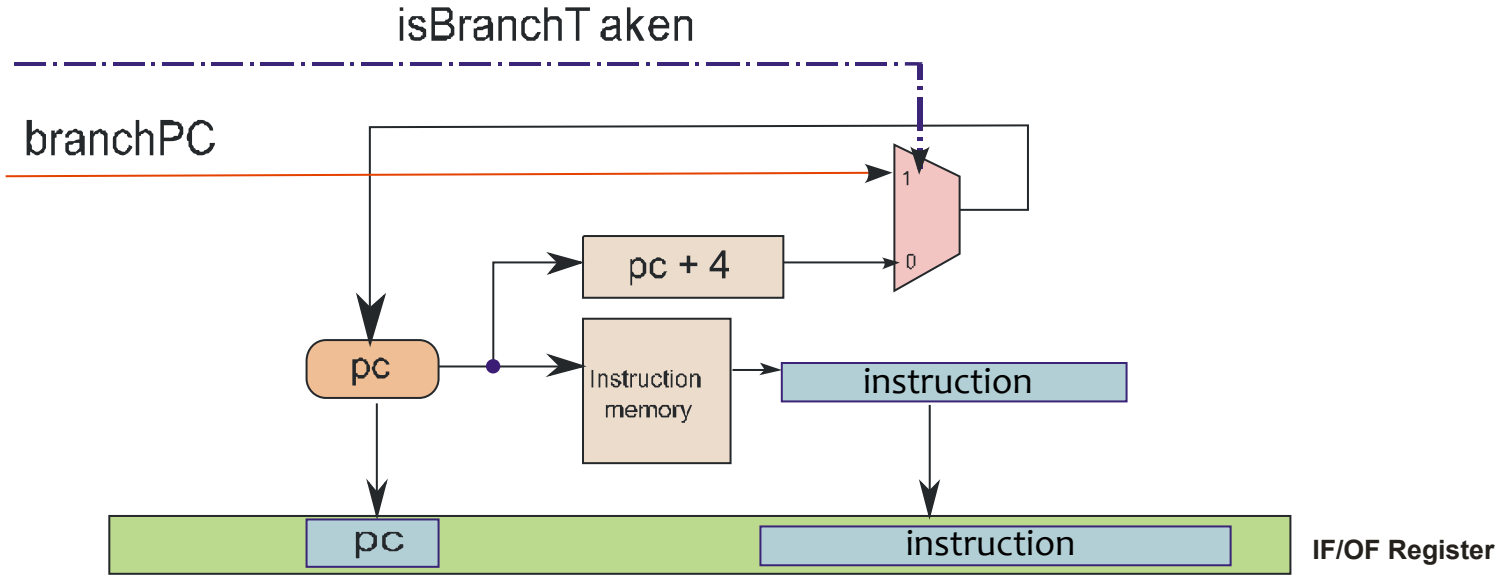
* What travels between stages ?
  * **ANSWER** : the instruction packet
* Instruction Packet
  * Instruction contents
  * Program counter
  * All intermediate results
  * Control signals
* Every instruction moves with its entire state, no interference between instructions

Outline

* Overview of Pipelining
* A Pipelined Data Path
* Pipeline Hazards
* Pipeline with Interlocks
* Forwarding
* Performance Metrics
* Interrupts/ Exceptions

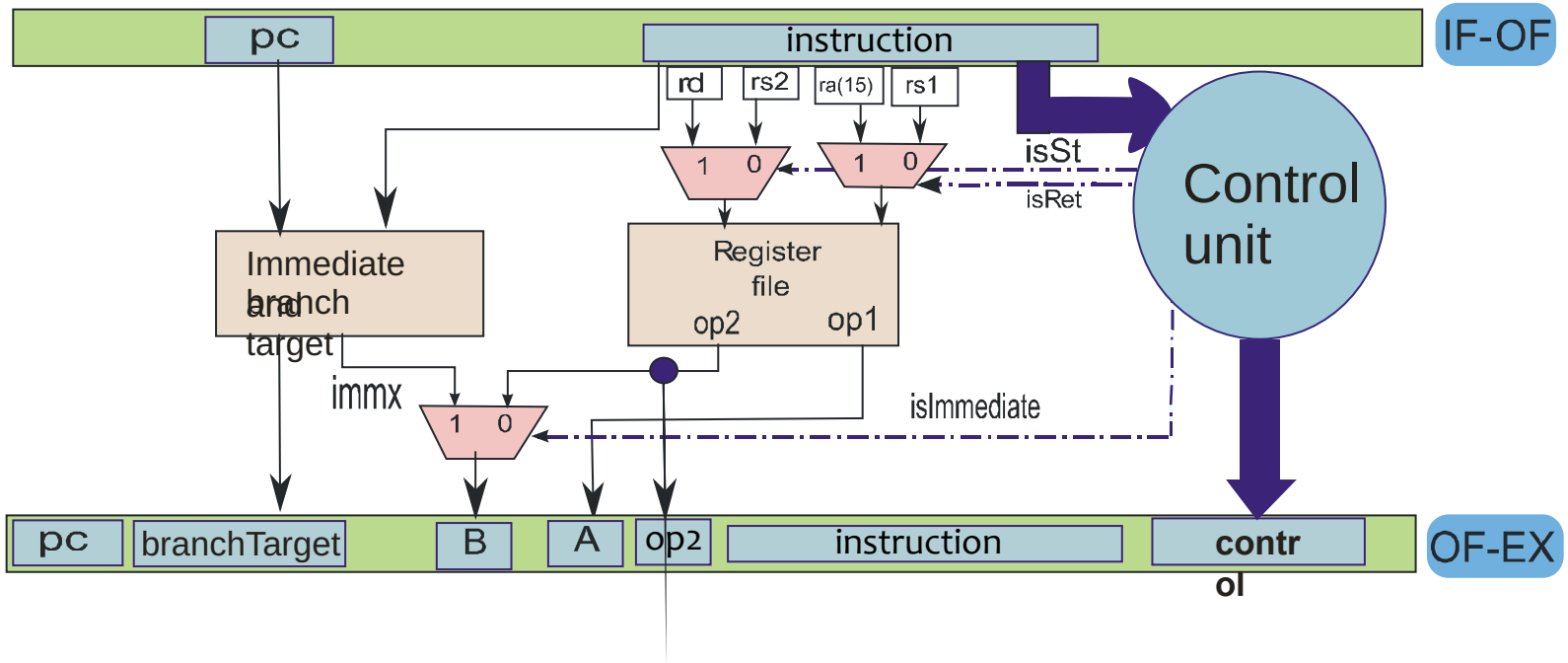


IF Stage



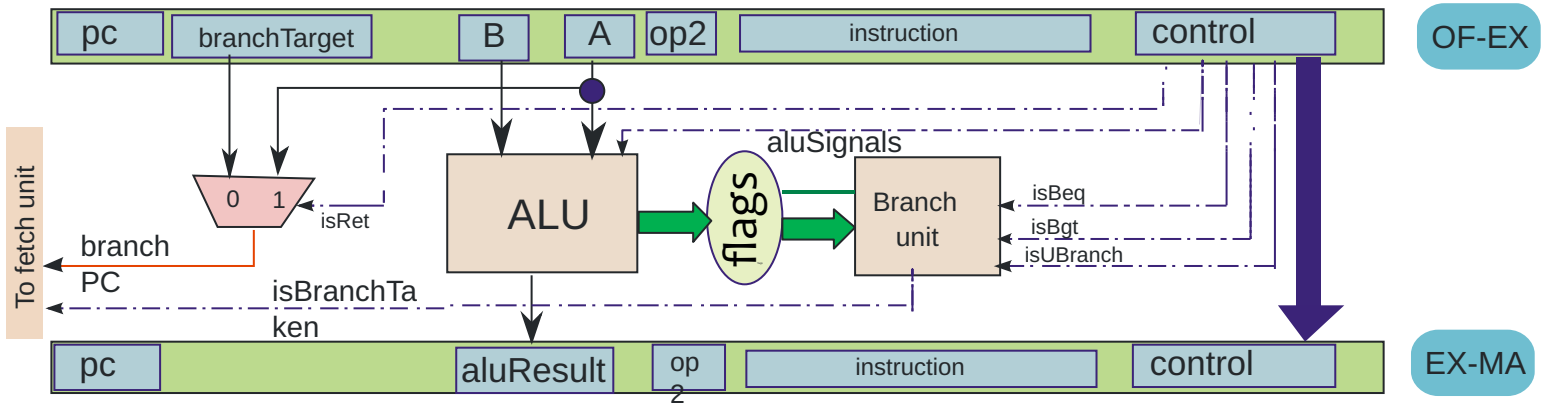
* Instruction contents saved in the instruction field

OF Stage



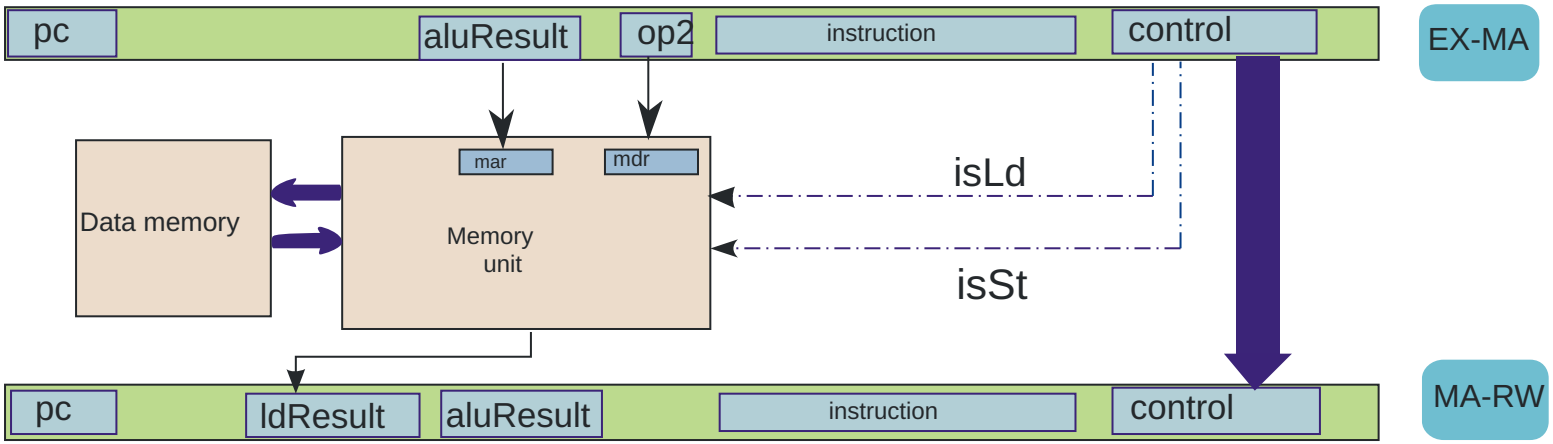
* A, B → ALU Operands, op2 (store operand), control (set of all control signals)

EX Stage



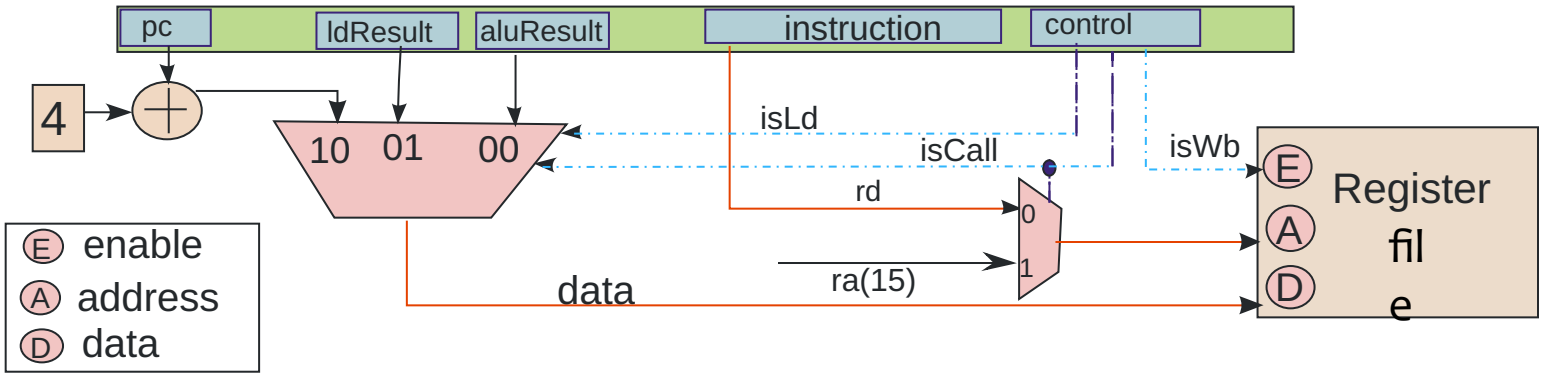
* aluResult → result of the ALU Operation
* op2, control, pc, instruction (passed from OF-EX)

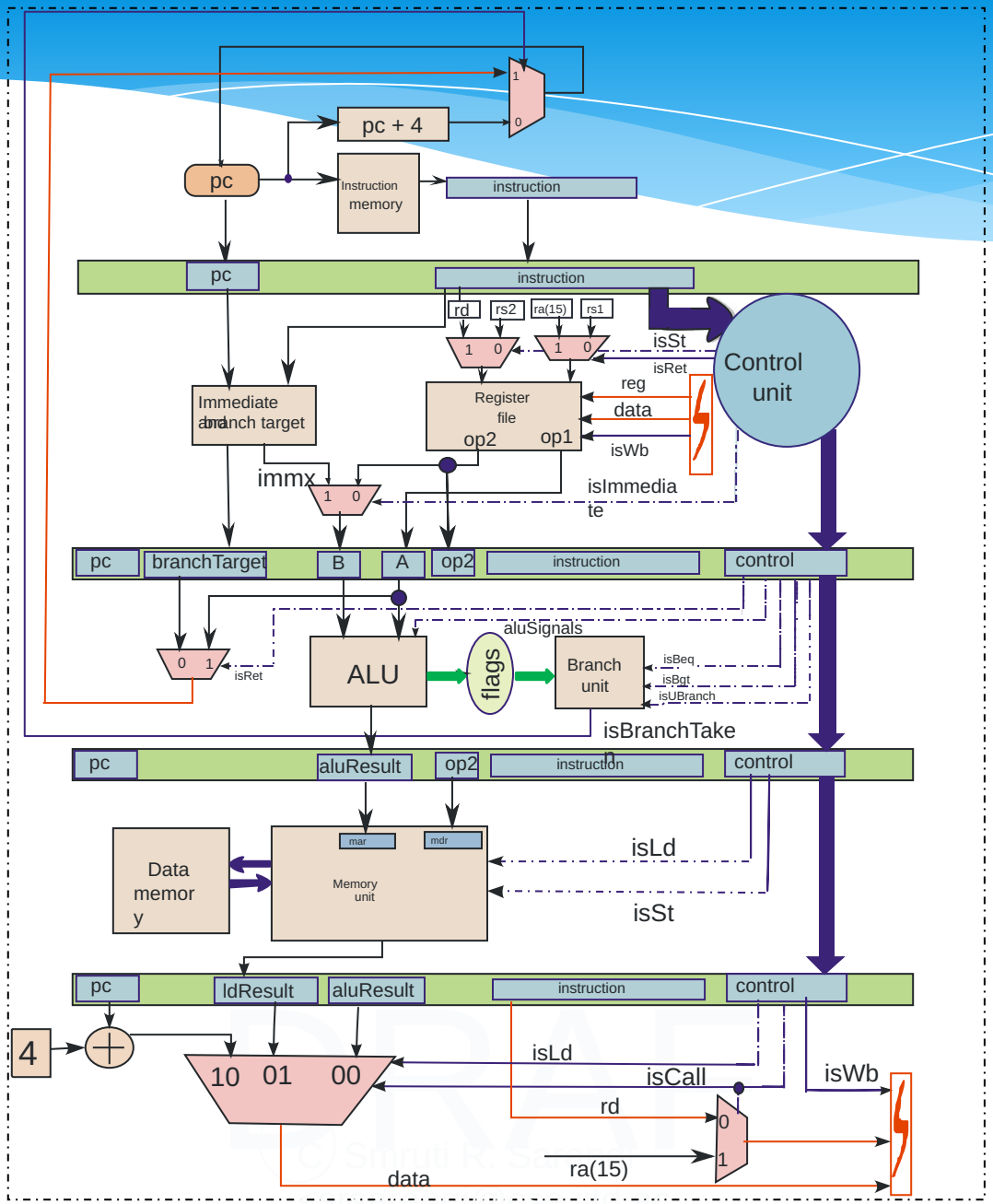
MA Stage



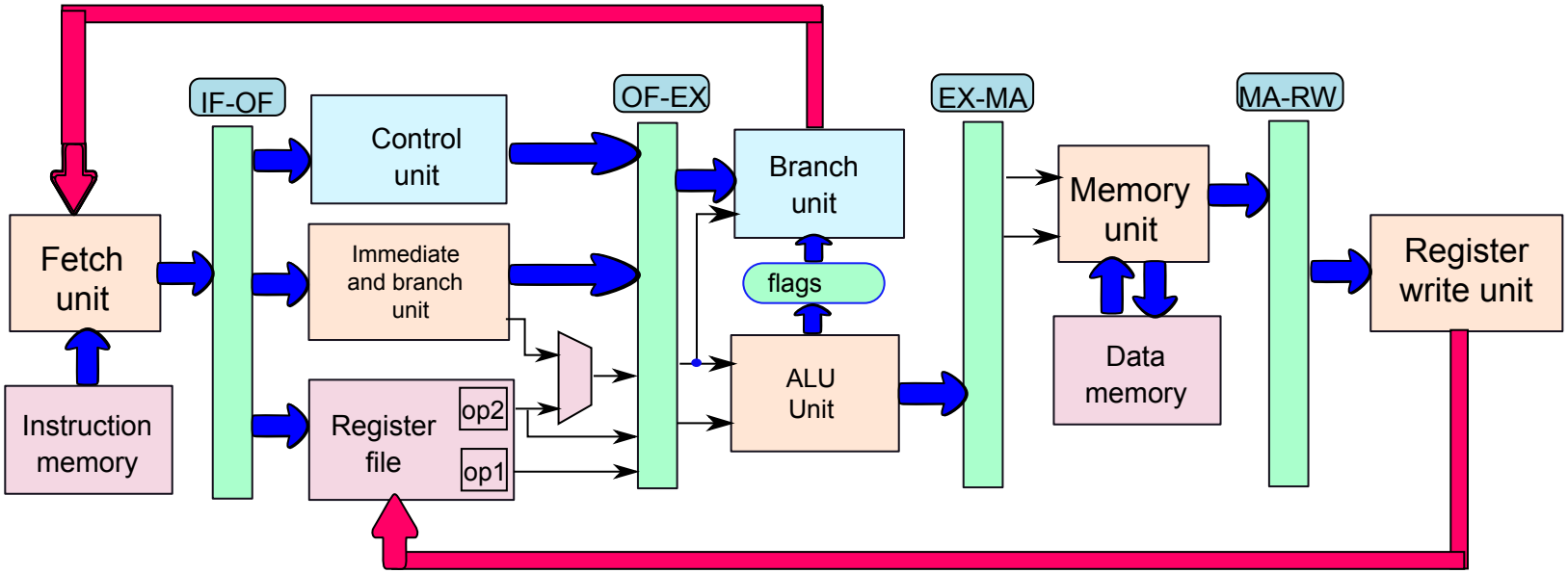
* ldResult → result of the load operation
* aluResult, control, pc, instruction (passed from EX-MA)

RW Stage


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

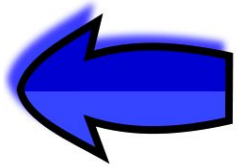


Abridged Diagram



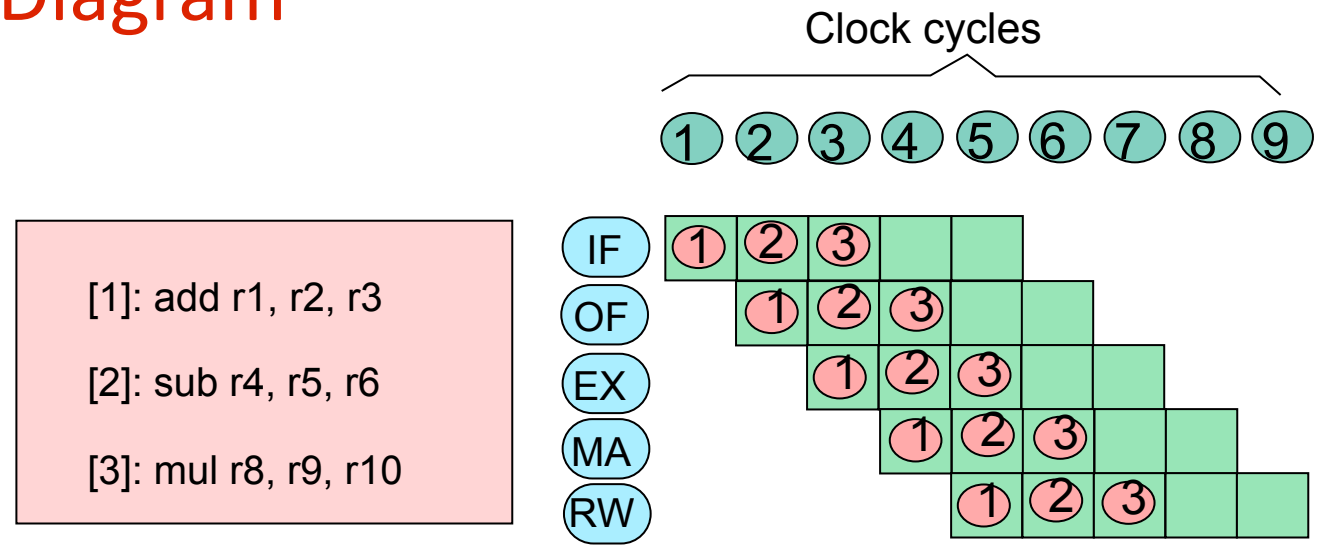
Outline

* Overview of Pipelining
* A Pipelined Data Path
* Pipeline Hazards
* Pipeline with Interlocks
* Forwarding
* Performance Metrics
* Interrupts/ Exceptions



Pipeline Hazards

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



Rules for Constructing a Pipeline Diagram

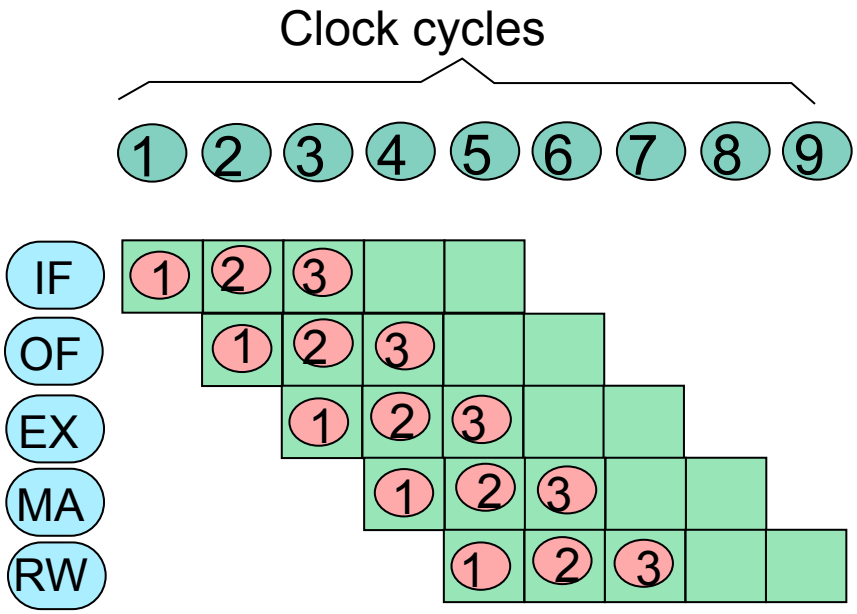
* It has 5 rows
  * One per each stage
  * The rows are named : IF, OF, EX, MA, and RW
* Each column represents a clock cycle
* Each cell represents the execution of an instruction in a stage
  * It is annotated with the name(label) of the instruction
* Instructions proceed from one stage to the next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

Example

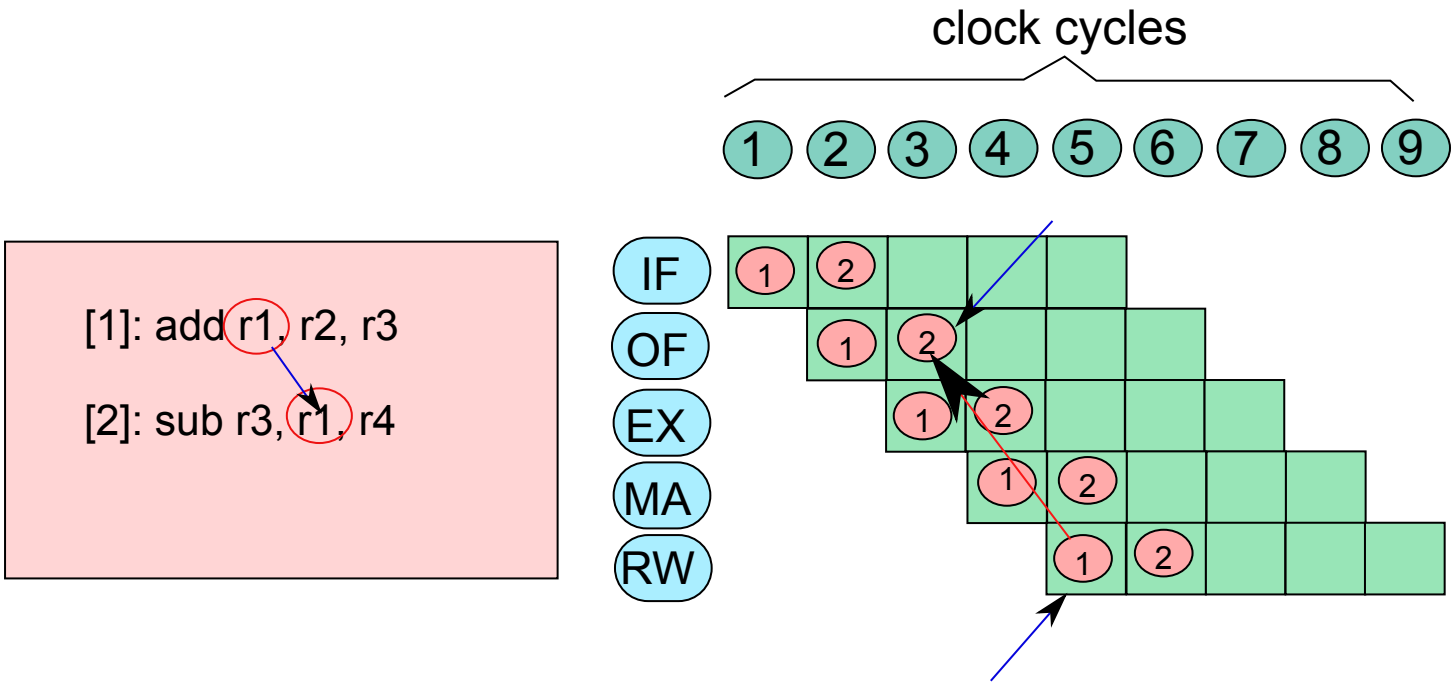
[1]: add r1, r2, r3

[2]: sub r4, r2, r5

[3]: mul r5, r8, r9



Data Hazards



* 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 5