# Instruction Level Parallelism

## Pipeline Architecture

Indian Institute of Technology Tirupati

Jaynarayan T Tudu
[jtt@iittp.ac.in]

Computer System Architecture (CS5202)
19th March, 2020

# Pipeline CPU

- Pipeline architecture and design

- Performance measurement

# CPU Design

**Objective:** To execute the ISA

CPU design begins from ISA

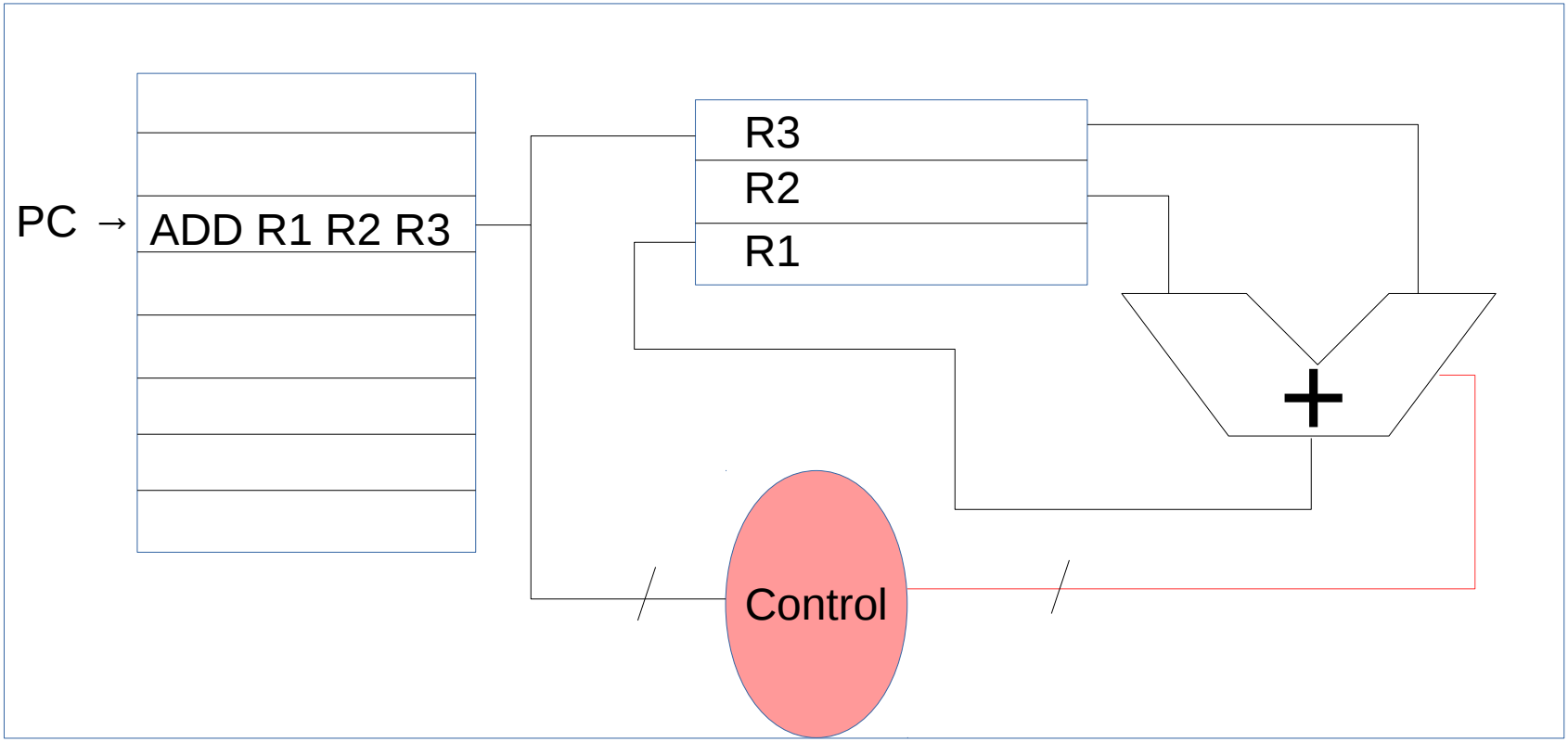
| ISA | Register | ALU Type | Year |
|---|---|---|---|
| IBM 701 | 1 | Accumulator | 1953 |
| CDC6600 | 8 | Load-store | 1963 |
| IBM360 | 18 | Reg-Mem | 1964 |
| DEC PDP-8 | 1 | Accumulator | 1965 |
| DEC PDP-11 | 8 | Reg-Mem | 1970 |
| Intel 8008 | 1 | Accumulator | 1974 |
| DEC VAX | 16 | Reg-Mem Mem-Mem | 1977 |
| Motorola | 16 | Reg-Mem | 1980 |
| Intel 80386 | 8 | Reg-Mem | 1985 |
| ARM | 16 | Load-store | 1985 |
| MIPS | 32 | Load-store | 1985 |
| HP PARISC | 32 | Load-store | 1986 |
| SPARC | 32 | Load-store | 1987 |
| Power PC | 32 | Load-store | 1992 |
| IA-64 | 128 | Load-store | 2001 |
| AMD64 | 16 | Reg-Mem | 2003 |
| x86-64 | 16 | Reg-Mem | 2003 |
| RISC-V | 32 | Load-store | 2010 |

# CPU Design

**Evolution of CPU Design:**

**Example:** ADD R1 R2 R3 | R1 ← R2 + R3

We will design a processor that executes the above ADD instruction!




Single cycle CPU design: All the micro-operations of a given instruction need to be carried out in just one cycle.

# CPU Design

## Single cycle CPU design

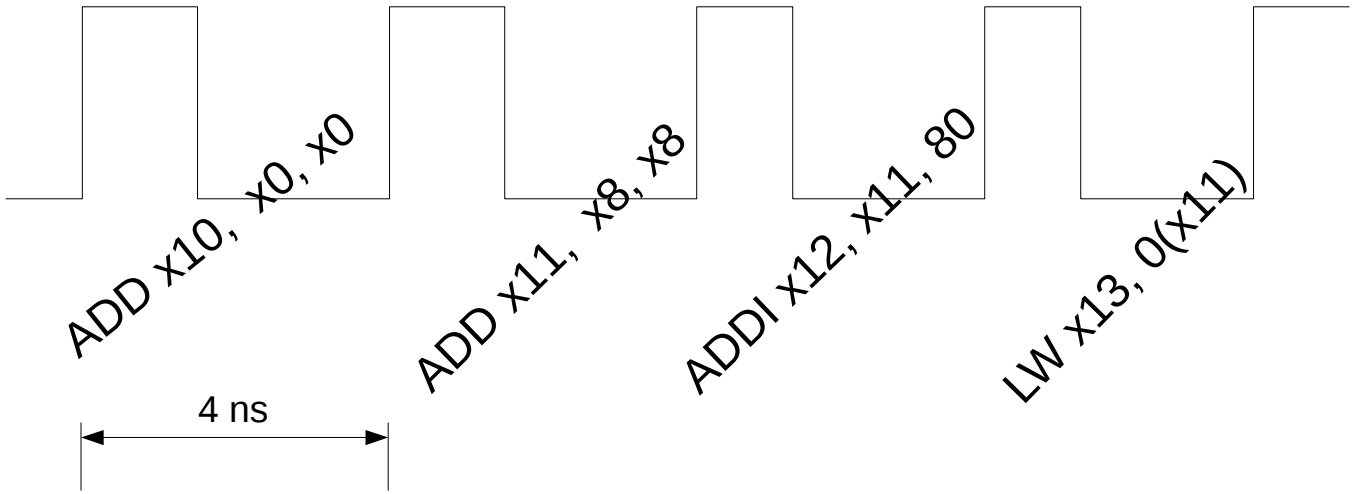
| Instructions | Actual Time |
|---|---|
| ADD R10, R0, R0 | 2 ns |
| ADD R11, R8, R8 | 2 ns |
| ADDI R12, R11, 80 | 2 ns |
| Loop: | |
| LW R13, 0(R11) | 4 ns |
| ADD R10, R10, R13 | 2 ns |
| ADDI R11, R11, 4 | 2 ns |
| BLT R11, R12, Loop | 2 ns |

Imagine a processor for large ISA!

Total time to execute the program:

= instruction count x cycle time
= 7 x 4 = 28 ns

Where cycle time is determined by the maximum time of any instruction.



# CPU Design

## Single cycle CPU design:

Important points to note with respect to single cycle design:

1) Each instruction take only one cycle to complete execution.
2) Every instruction has to be go through the four important phases:
   - Fetch the instruction from memory
   - Decode the instruction to identify the the operands and generate control signal
   - Fetch the necessary operand either from Registers, immediate, or data memory according to the addressing mode.
   - Perform the necessary operation such as ALU, Load, Store, Branching etc
   -Write the results back to register or data memory

3) All these micro-operations are to be performed in just one cycle.
4) Since the single cycle processor uses only one clock of fixed period, it implies that all the instruction would require same cycle time.
5) One things to observe for a processor is: it has two different set of paths:
   - control path
   - data path

(in the later designs these paths will be isolated systematically to create multi-cycle and pipeline)

# CPU Design

**Multi-cycle CPU design:**

The first question we ask is what is the problem with single cycle design?

We need to think in terms of performance gain and loss!
We also need to think in terms of hardware area overhead!
We also need to think in terms power consumption!

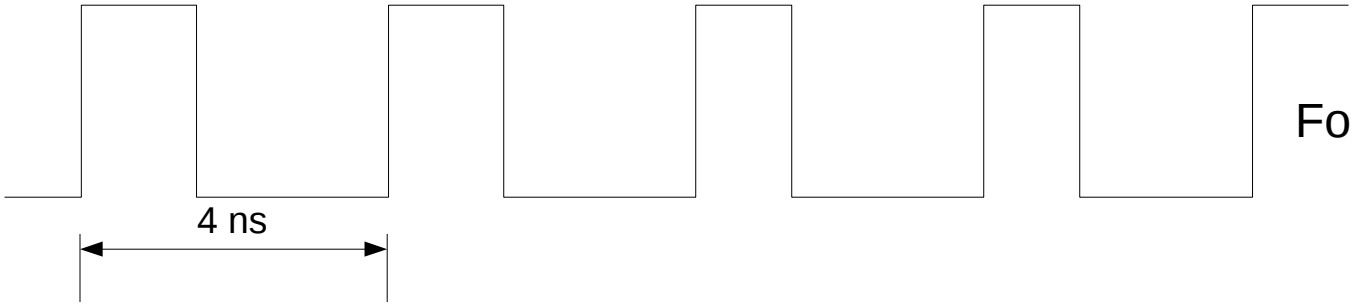
| Instructions | Actual Time |
|---|---|
| ADD R10, R0, R0 | 2 ns |
| ADD R11, R8, R8 | 2 ns |
| ADDI R12, R11, 80 | 2 ns |
| Loop: | |
| LW R13, 0(R11) | 4 ns |
| ADD R10, R10, x13 | 2 ns |
| ADDI R11, R11, 4 | 2 ns |
| BLT R11, R12, Loop | 2 ns |

- What if we design a processor with clock cycle time of 2 ns?

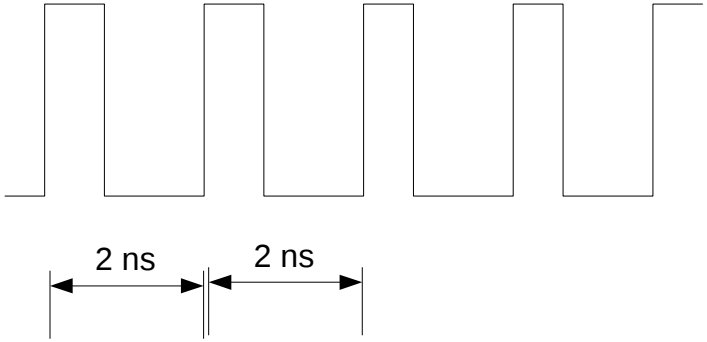
- LW would take two cycles to complete
- Rest all instruction would take one cycle

# CPU Design

**Multi-cycle CPU design:**




For single cycle CPU




For multi-cycle CPU

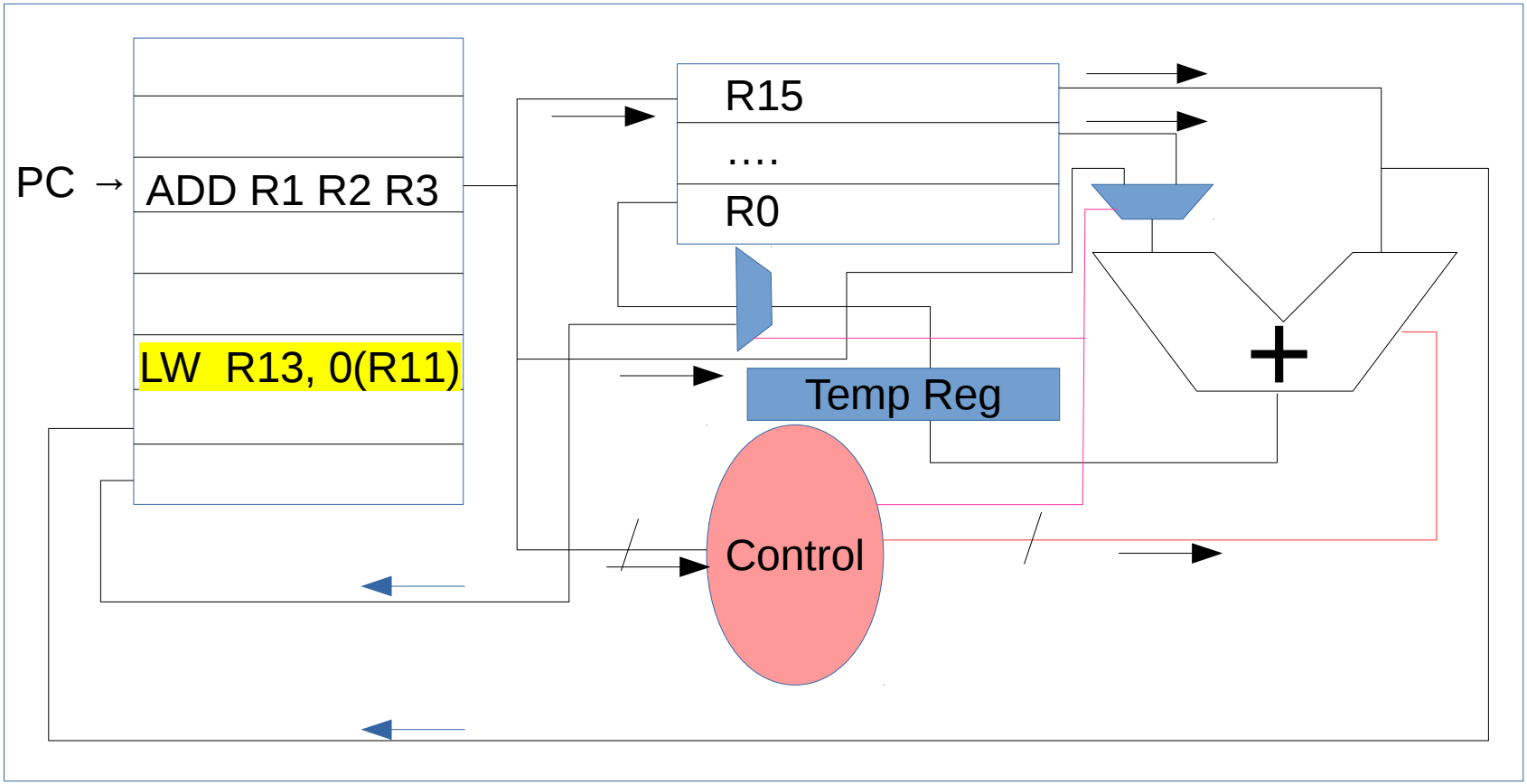
| | Instructions | Actual Time |
|---|---|---|
| CC1 | ADD R10, R0, R0 | 2 ns |
| CC2 | ADD R11, R8, R8 | 2 ns |
| CC3 | ADDI R12, R11, 80 | 2 ns |
| | Loop: | |
| CC4 CC5 | LW R13, 0(R11) | 4 ns |
| | ADD R10, R10, R13 | 2 ns |
| | ADDI R11, R11, 4 | 2 ns |
| | BLT R11, R12, Loop | 2 ns |

# CPU Design

**Multi-cycle CPU design:**

The idea is to partition the data path and buses

The path being broken by introducing Temp Reg.



# CPU Design

**Multi-cycle CPU design:** Performance analysis

| Instructions | Actual Time |
|---|---|
| ADD R10, R0, R0 | 2 ns |
| ADD R11, R8, R8 | 2 ns |
| ADDI R12, R11, 80 | 2 ns |
| Loop: | |
| LW R13, 0(R11) | 4 ns |
| ADD R10, R10, R13 | 2 ns |
| ADDI R11, R11, 4 | 2 ns |
| BLT R11, R12, Loop | 2 ns |

Total execution time =

2 + 2 + 2 + 4 + 2 + 2 + 2
= 16 ns

Speed up = Performance of new/Performance of old

= Time of old (single cycle) / Time of new (multi cycle)

= 28 ns / 16 ns = 1.7 time

How did you get this performance?
At the cost of additional registers and nets

# CPU Design

## Multi-cycle CPU design:

Important points to note:

1 – The idea is to partition the data path in such a way that the cycle time would be as minimum as possible (the smallest execution time of any instruction)

2 – The design would require more hardware resources than the corresponding single cycle design.

3 – Each instruction would require multiple cycles to complete their execution.

4 – The multiple cycle design would support more diverse set of instruction in efficiently. Where as in single cycle the diverse set of instruction would lead to performance loss.

5 – If the instruction set is uniform in terms of execution time it is certainly wise to implement In single cycle. However, we will see next that this argument is not always true.

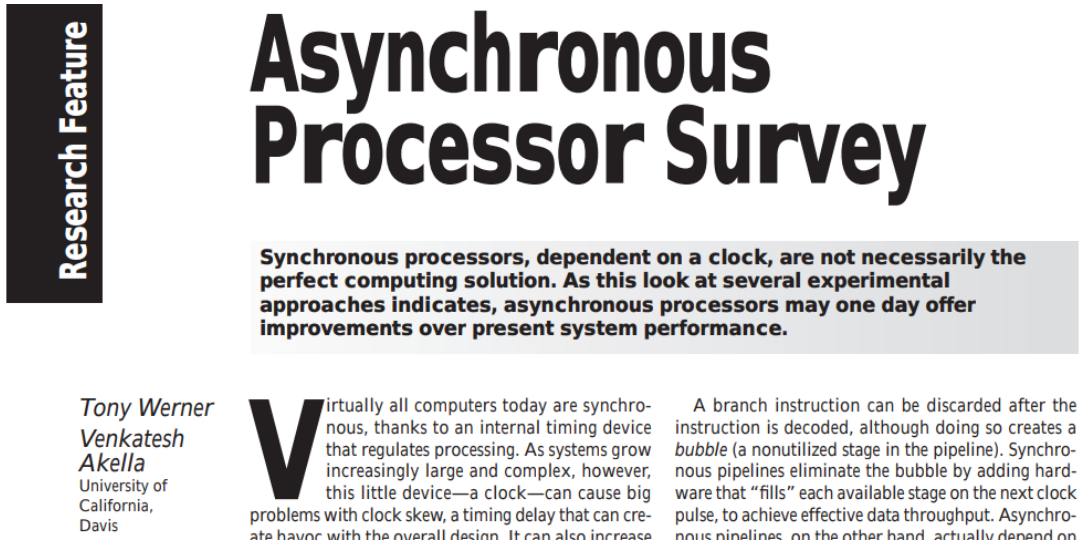
6 – Multi-cycle is the beginning of pipeline architecture.

7 – Partition design is one of the challenging problems since it require that each data path be uniform in terms of path length (or propagation time delay).

# CPU Design

Single cycle CPU design

Multi-cycle CPU design



Research Feature

## Asynchronous Processor Survey

**Synchronous processors, dependent on a clock, are not necessarily the perfect computing solution. As this look at several experimental approaches indicates, asynchronous processors may one day offer improvements over present system performance.**

*Tony Werner*
*Venkatesh Akella*
University of California, Davis

Virtually all computers today are synchronous, thanks to an internal timing device that regulates processing. As systems grow increasingly large and complex, however, this little device—a clock—can cause big problems with clock skew, a timing delay that can create havoc with the overall design. It can also increase

A branch instruction can be discarded after the instruction is decoded, although doing so creates a *bubble* (a nonutilized stage in the pipeline). Synchronous pipelines eliminate the bubble by adding hardware that "fills" each available stage on the next clock pulse, to achieve effective data throughput. Asynchronous pipelines, on the other hand, actually depend on

How can we do better than the multi-cycle?

# Pipeline Architecture

The basic idea is:

## Parallelism

This is one of the architectural principle

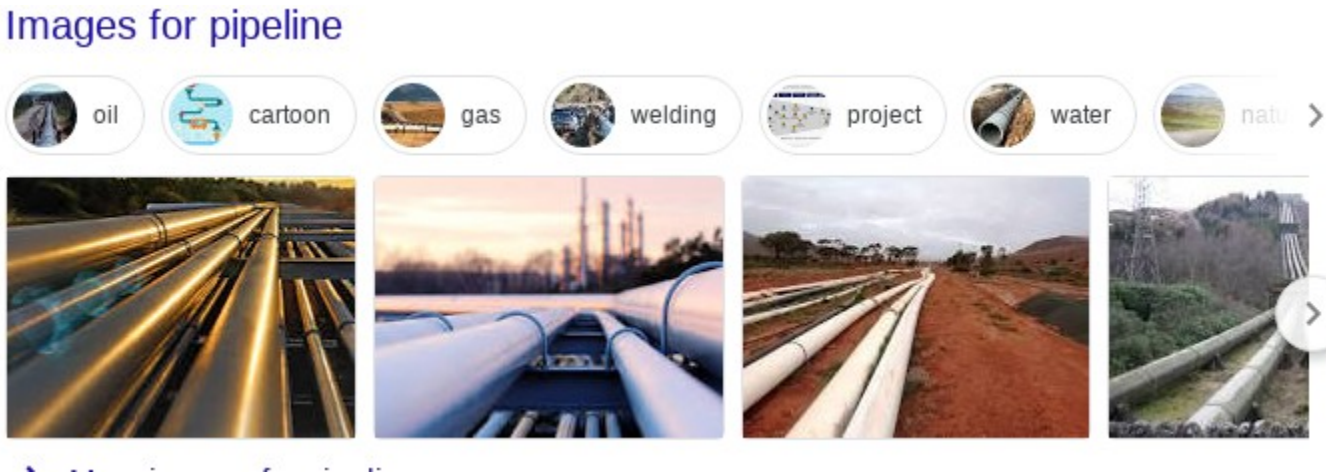
Pipeline

Computing

In computing, a pipeline, also known as a data pipeline, is a set of data processing elements connected in series, where the output of one element is the input of the next one. The elements of a pipeline are often executed in parallel or in time-sliced fashion.
Wikipedia

When you do google search

There are many real life example of pipeline



# Pipeline Architecture

The basic idea is: Parallelism

| Instructions | Actual Time |
|---|---|
| ADD R10, R0, R0 | 2 ns |
| ADD R11, R8, R8 | 2 ns |
| ADDI R12, R11, 80 | 2 ns |
| Loop: | |
| LW R13, 0(R11) | 4 ns |
| ADD R10, R10, R13 | 2 ns |
| ADDI R11, R11, 4 | 2 ns |
| BLT R11, R12, Loop | 2 ns |

How do you execute these instruction in parallel?

From starting to end an instruction has to travel through:

1 – Fetch from the instruction memory
2 – Decode to generate control signal And read operand
3 – Execute (ALU operations)
4 – Memory operations if data to be read from or to be written into
5 – Writing back the results from Stage 3

Above micro operations are necessary.

# Pipeline Architecture

The basic idea is: Parallelism

How do you execute these instruction parallely?

Example of very bad design:

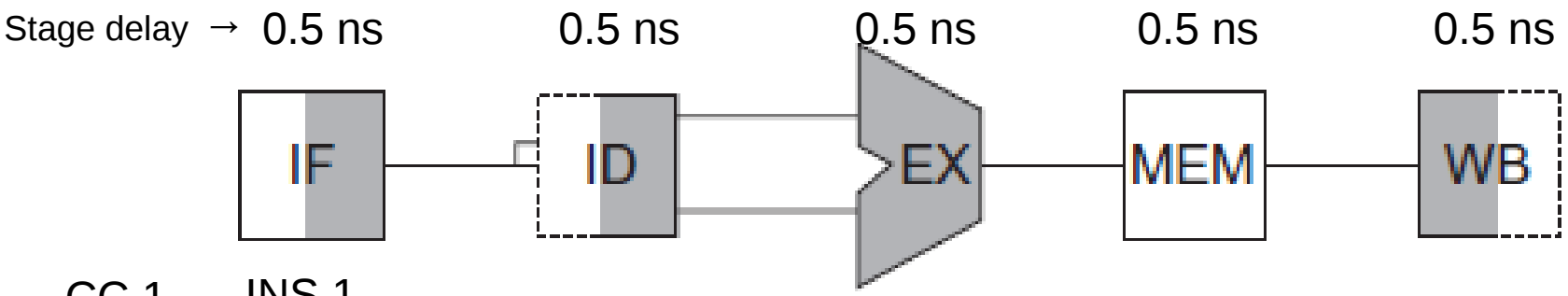
1 – You can have multiple single cycle processor operating in parallel
2 – You can have multiple multi-cycle design in parallel
3 – You can have multi-clock multiple single cycle design

## The proven idea of pipeline architecture:

The parallelism can be achieved by executing the micro operations of more than one instructions in parallel.
This is one kind of instruction level parallelism (ILP)

# Pipeline Architecture

## The pipeline execution flow:

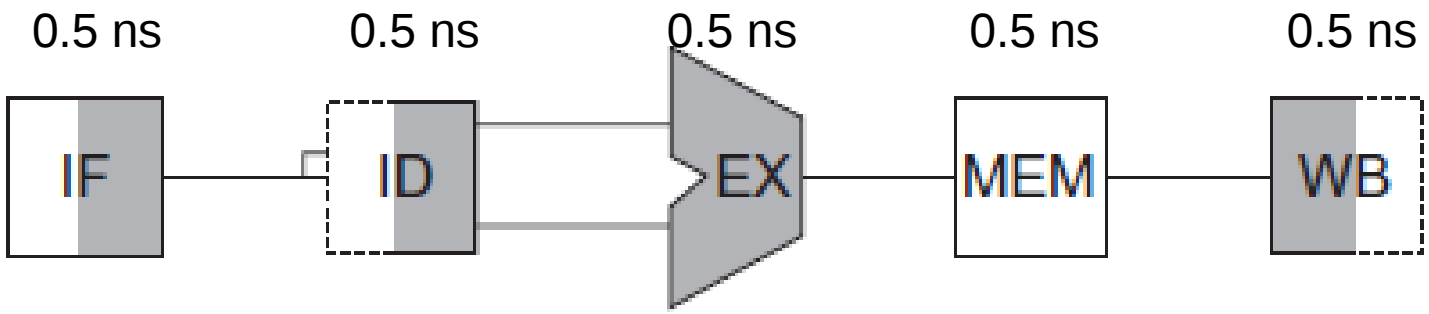



| | | | | | |
|---|---|---|---|---|---|
| CC 1 | INS 1 | | | | |
| CC 2 | INS 2 | INS 1 | | | |
| CC 3 | INS 3 | INS 2 | INS 1 | | |
| CC 4 | INS 4 | INS 3 | INS 2 | INS 1 | |
| CC 5 | INS 5 | INS 4 | INS 3 | INS 2 | INS 1 |
| CC 6 | | INS 5 | INS 4 | INS 3 | INS 2 |
| CC 7 | | | INS 5 | INS 4 | INS 3 |
| CC 8 | | | | INS 5 | INS 4 |
| CC 9 | | | | | INS 5 |

CC – clock cycle, INS1 – Instruction 1;
IF – instruction fetch, ID – instruction decode, EX – instruction execution,
MEM – Memory operation (load/store), WB – write back to register

# Pipeline Architecture

## Performance measurement:




Assuming an ideal pipeline, what would be the performance improvement?

Ideal pipeline:

- No halt in any stage at any point of time
- All instructions get executed in free flow.

**Example:** n instructions, k stages, and stage delay of t ns

Total execution time = $k + (n - 1) \times t$

*k* is due to the fact that the first instruction requires *k* cycle to complete, remaining $n - 1$ instructions would be completed in followed by cycles.

# Pipeline Architecture

## Exercise on Performance Comparison:

Exercise 1:

1) What is the performance improvement due to ideal pipeline over the single cycle and multi-cycle design?

2) What is the difference between multi-cycle design and pipeline architecture?

3) Imagine an ISA where all the instructions take exactly same time, let say *t* ns, then what would be the performance comparision of single, multi and pipeline processor?

# Pipeline Architecture

## Pipeline Processor Design:

Recall the multi-cycle design:

- It partitioned the data path by introducing additional registers, multiplexers and interconnects (nets or wires).

On this design, if the control path is also partitioned, you will get a pipeline design.

Let us start from the micro-operations at each stages:

IF – instruction fetch:

IR ← Mem[PC] where IR is isntruction register, PC is program counter
NPC ← PC + 4 and NPC is next program counter temporary register.

# Pipeline Architecture

## Pipeline Processor Design:

Let us start from the micro-operations at each stages:

IF – instruction fetch:

IR ← Mem[PC] where IR is isntruction register, PC is program counter
NPC ← PC + 4 and NPC is next program counter temporary register.

ID – instruction decode

A ← Regs[rs] where rs is the source 1 and source 2 register
B ← Regs[rt] A and B are temporary registers
Imm ← Sign extended immediate field of IR, IR is instruction register

EX – Execute the ALU instruction or computer effective address

ALUout ← A + Imm where Imm is immediate value in IR
ALUout ← A func B func is the funuction fied in ALU instruction
ALUout ← A op Imm op is operation field in immediate instruction
ALUout ← NPC + (Imm << 2) << is the left shift

# Pipeline Architecture

## Pipeline Processor Design:

Let us start from the micro-operations at each stages:

MEM – Memory Access and branch complete

PC ← NPC or
PC ← ALUout if condition is satified for branch instruction
LMD ← Mem[ALUout] or  this is for load instruction
Mem[ALUout] ← B  this is for store instruction

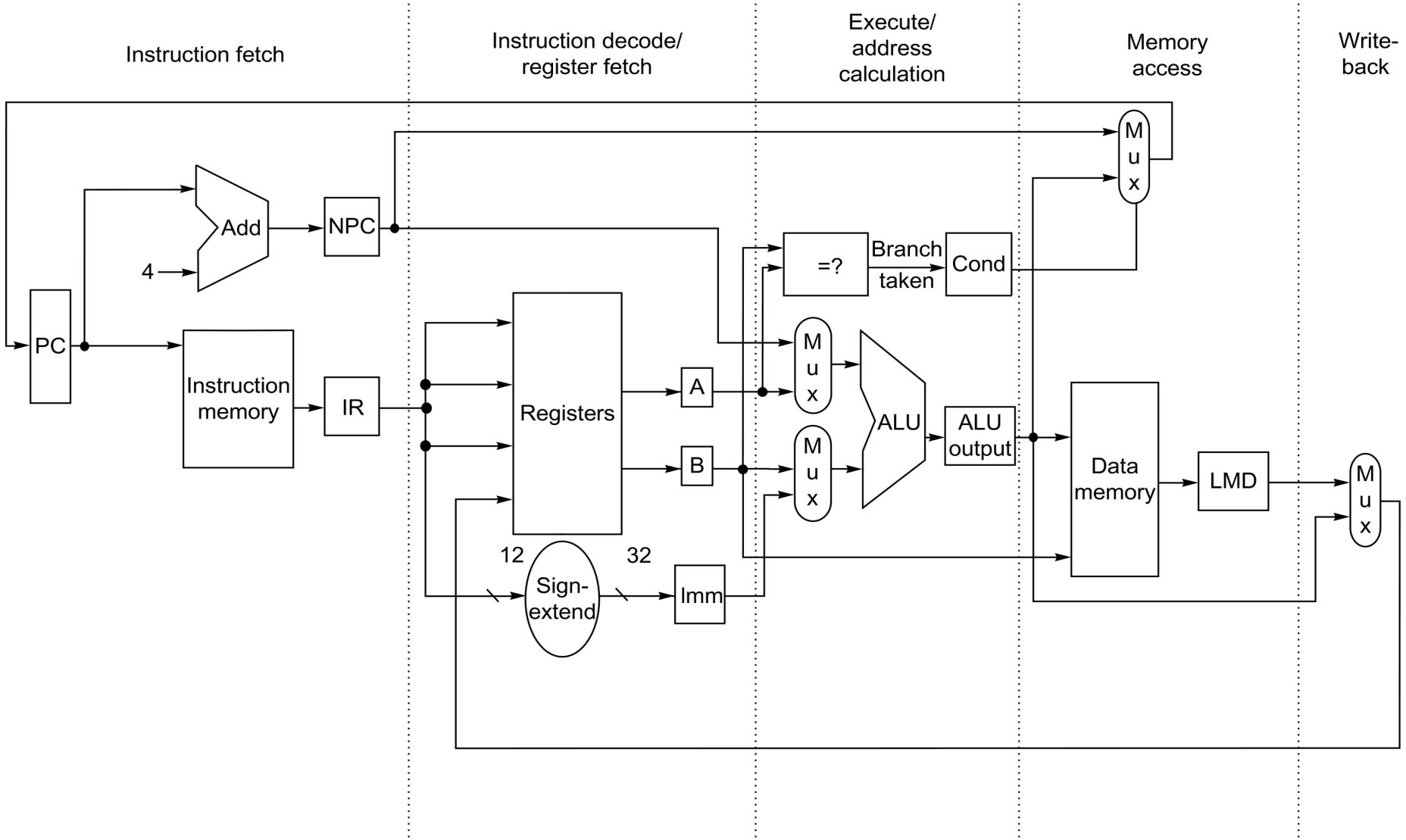
WB – Write back to registers

Reg[rs] ← ALUout  for register type instructions
Reg[rt] ← ALUout
Reg[rt] ← LMD  this is for laod instruction

Next, the processor is designed according to these operations!

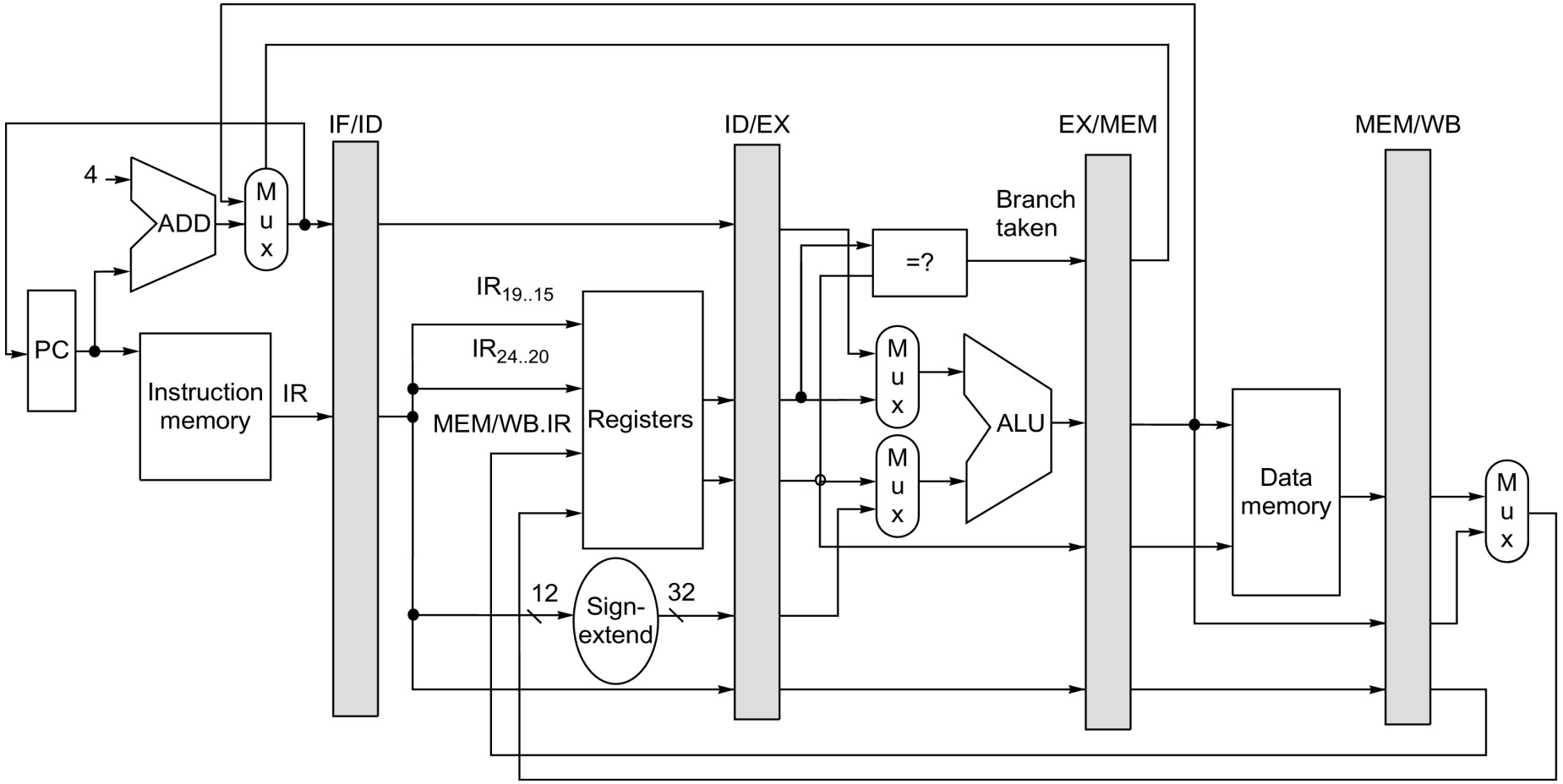
# Pipeline Architecture

## Multi-cycle processor data path:



# Pipeline Architecture

## Pipeline view of the data path:




Note: The figure above does not show control path, the control signals would go to the select line of muxes and to the ALU control. These signals are also pipelined.

# Pipeline Architecture

Exercise:

1 – Complete the pipeline design of the MIPS architecture with control signals

2 - Complete the pipeline design of RISC-V architecture with control signals

3 - Find out the hardware overhead in comparison to single cycle and multi-cycle design for MIPS architecture and RISC-V architecture.

4 – Calculate the exact stage delay for each stage

Assume that mux takes 0.2 ns, registers read or write takes 0.5 ns, ALU or any other arithmetic operations take 0.4 ns, reading/writing to data instruction memory takes 1 ns.

## Reference:

1 – Chapter 4, Patterson and Henessy, Computer Organisation and Design, RISC-V edition or MIPS edition.

2 – Appendix C, Henessy and Patterson, Computer Architecture Quantitative Approach, 5th Edition or 6th Edition

# Next Lecture

Pipeline to continue: Hazards

Long Latency pipeline