Computer Organization and Architecture – Let’s understand Pipelining.

Shriram K Vasudevan

Session 1

Introduction

- Well, this is not easy. Will not be a cake walk folks!
- Let's put more effort and focus to understand pipelining, better.
- Remember and recollect the manufacturing of bikes/cars etc. It will be easier to understand pipelining with that example.
- Let us define what pipelining is all about!
  - It is a technique for making the processors work faster. Here, with pipelining multiple instructions are handled at a time. Overlapping of instructions during execution can be termed as pipelining.
  - Today, all the architectures are pipelined. Key would be, if it is three stage pipeline or 5 stage pipelining or 7 stage pipelining. (Don't worry, I shall explain)

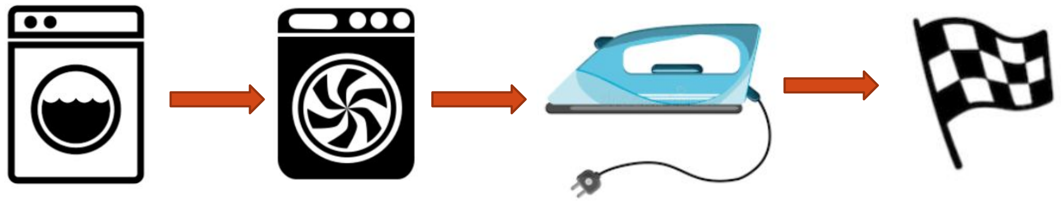
Let’s see an example

Lets us take the cloth washing scenario to understand the pipelining.

1. Collect the clothes to be washed. Stack them inside the door of the washing machine. (Step – 1)
2. After the washing is done, the clothes should be dried and should use a dryer now. (Step – 2)
3. Now, you got to iron the clothes in a table with iron. (Step – 3)

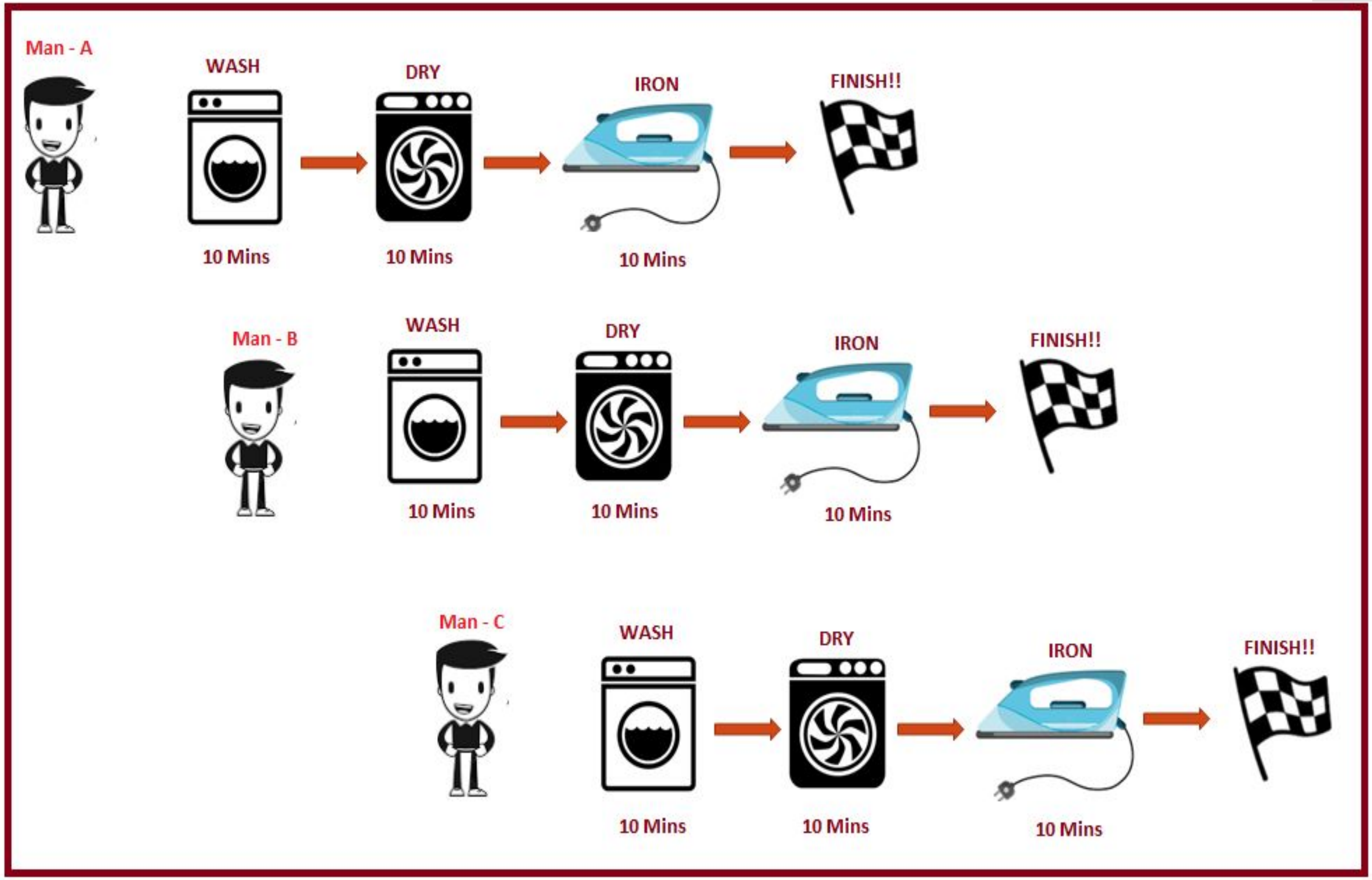
That’s it. You can let the next person wash the clothes. 😊

(Disclaimer: All these really matters for the one who washes clothes)



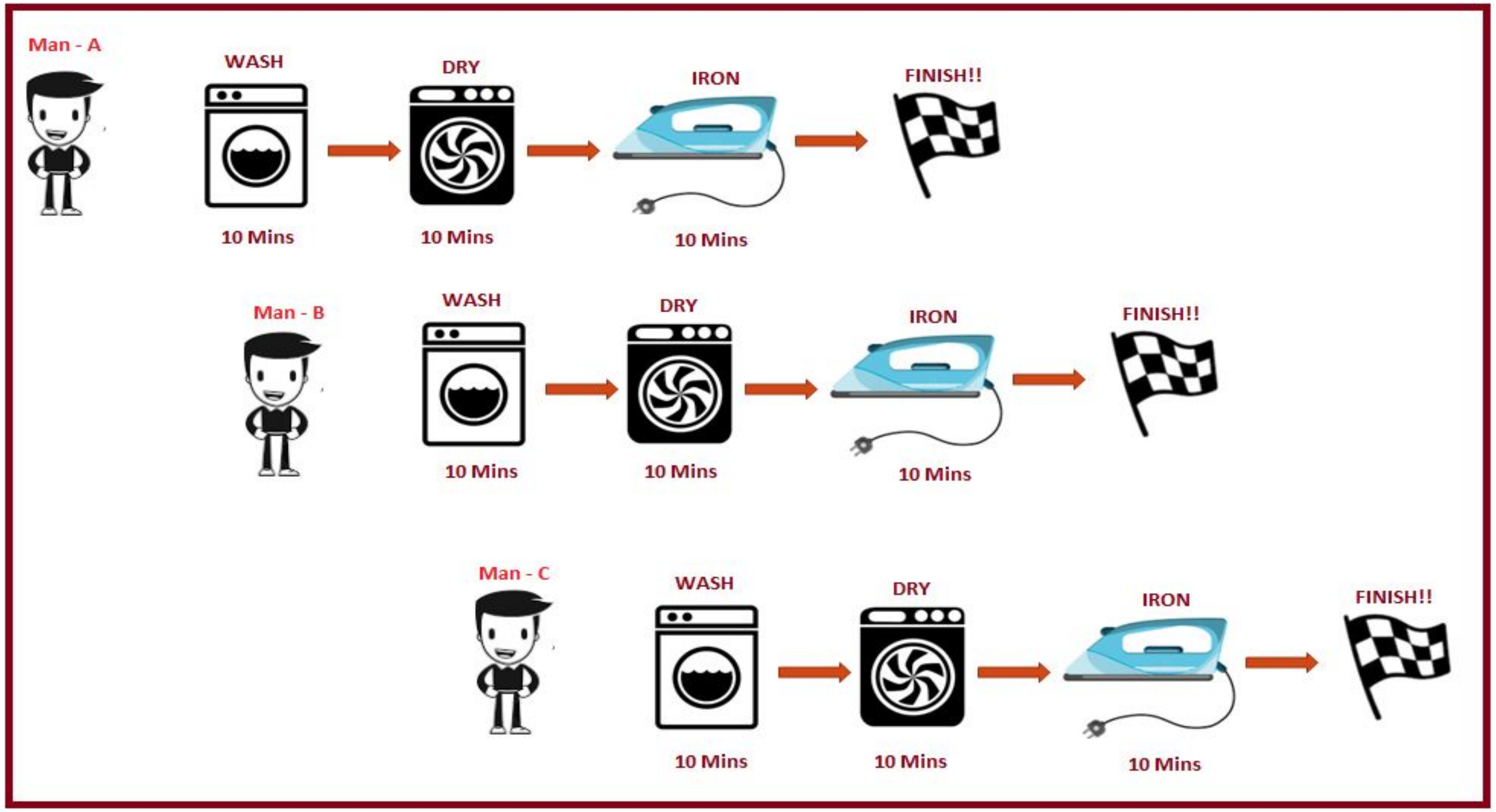
Let's get better …

- Here is where pipelining could come into picture.
- Let us see it diagrammatic.



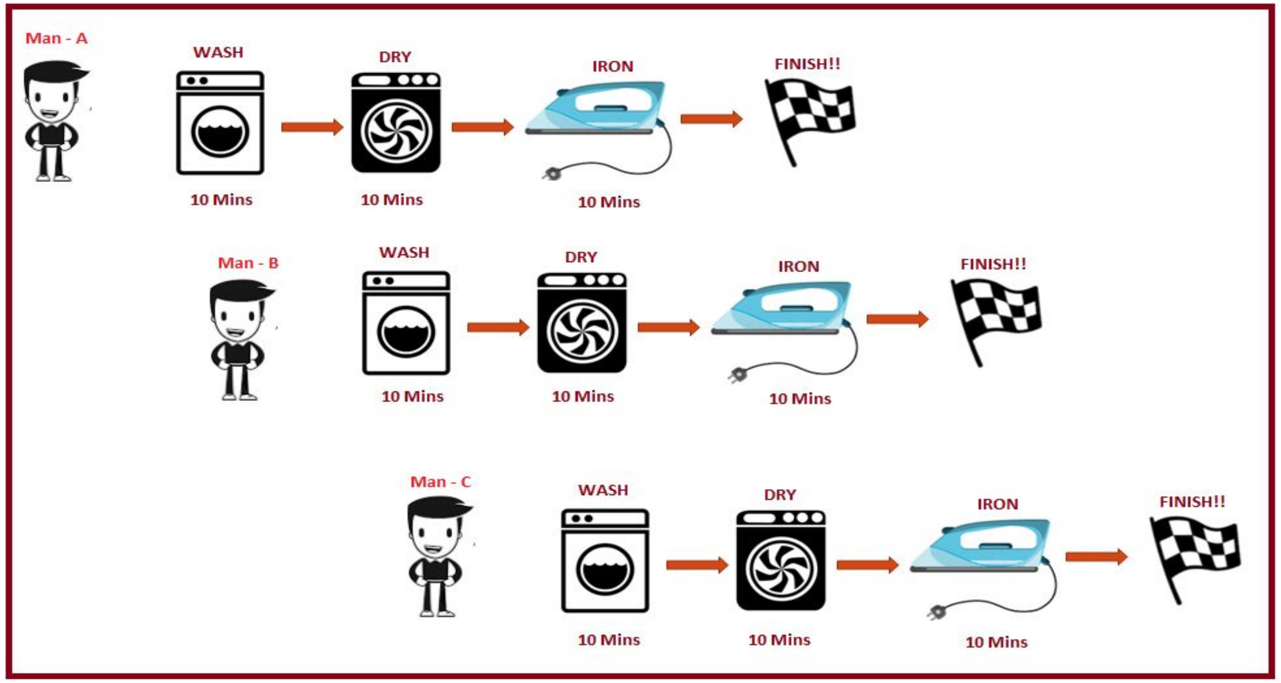
Let's get better …

- If all the three persons do this one after another (i.e. no pipelining) time required totally for all the three is 90 Mins.
- Considering, we have enabled pipelining, the total timeframe required is 50 mins. Awesome right?



Well, lets get better.

- When things go in parallel, we call it pipelining and the washing/drying etc. are all referred as stages in pipelining.
- Listen to this point. Pipelining might not be possible all time. We need to have separate (distinct) resources for accomplishing pipelining. (Means, you cannot have washing unit, dryer and iron all in one equipment and pipelining becomes impossible there.)



Contd.,

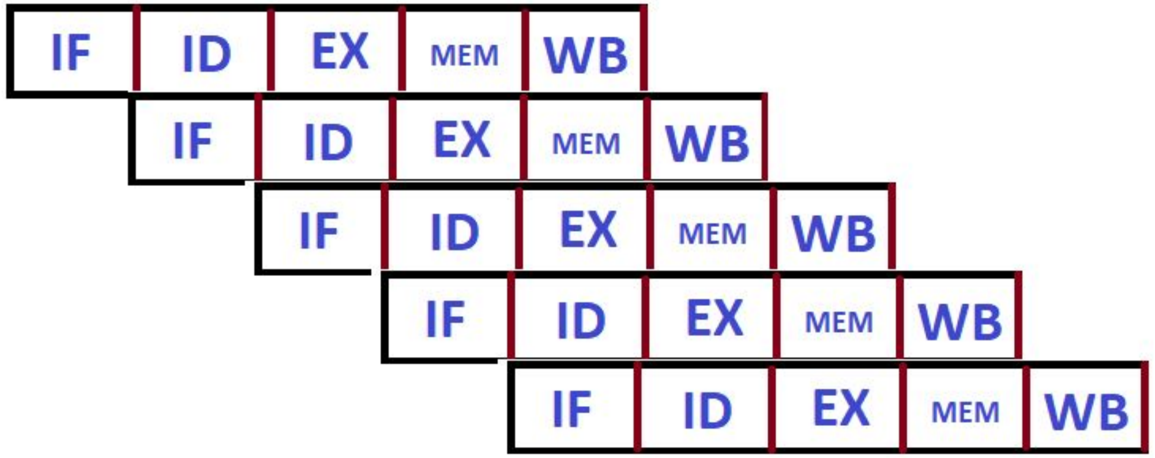
- Well, let us answer some questions.
  - Will pipelining increase the throughput? - Yes.
    - Throughput - the amount of material or items passing through a system or process. (Means, the overall result)
  - Will pipelining increase efficiency (decrease in time of execution) for single load?
    - Nope. It is not the case. It won't.
- Hence, it would be important to understand that pipelining is meant to increase the efficiency on the whole and not for any single instruction. i.e. load.

Session – 2

Shriram K Vasudevan

Stages of MIPS pipelining

- MIPS has five stages in its architecture for pipelining.
- Let us read the steps through:
  - Fetch the instruction from memory. (People call it fetch, simply)
  - Since instruction is available, it can be decoded and registers are to be read. (MIPS permits you to decode and read registers at the same time)
  - Now, comes execution. (It could be an operation to perform addition or to compute an address, I mean, it can be ADD or LW)
  - Access the data from the memory location.
  - Result is available in hand and it has to be written back to the registers.



IF - Instruction Fetch

ID - Instruction Decode

EX - Instruction Execution

MEM - Memory Access

WB - Write Back

Let us make it technical. Single Cycle Vs. Pipelining

Instruction	Fetch	Read / Decode	ALU (Execute)	Memory Access	Write Back	Time Consumed (Total)
LW (Load Word)	200 PS	100 PS	200 PS	200 PS	100 PS	800 PS
SW (Store Word)	200 PS	100 PS	200 PS	200 PS		700 PS
ADD/SUB/AND	200 PS	100 PS	200 PS		200 PS	700 PS
BEQ/BNE	200 PS	100 PS	200 PS			500 PS

Note: We have to assume that the intermediary components have no delay and Mux has no delay as well. (Delay would further complicate the learning at this point, hence, let us ignore the same)

Contd.,

Instruction	Fetch	Read / Decode	ALU (Execute)	Memory Access	Write Back	Time Consumed (Total)
LW (Load Word)	200 PS	100 PS	200 PS	200 PS	100 PS	800 PS
SW (Store Word)	200 PS	100 PS	200 PS	200 PS		700 PS
ADD/SUB/AND	200 PS	100 PS	200 PS		200 PS	700 PS
BEQ/BNE	200 PS	100 PS	200 PS			500 PS

Observations:

1. We have considered no delay for the Mux and intermediary components. (Remember this, don't forget)
2. In single cycle model, each instruction needs 1 clock cycle. So, the point is, the clock cycle should be stretched so as to also accommodate the slowest instruction.
3. **In the above listing, which is the slowest instruction? Undoubtedly, it is LW with 900 PS. – Understand this point.**

Contd.,

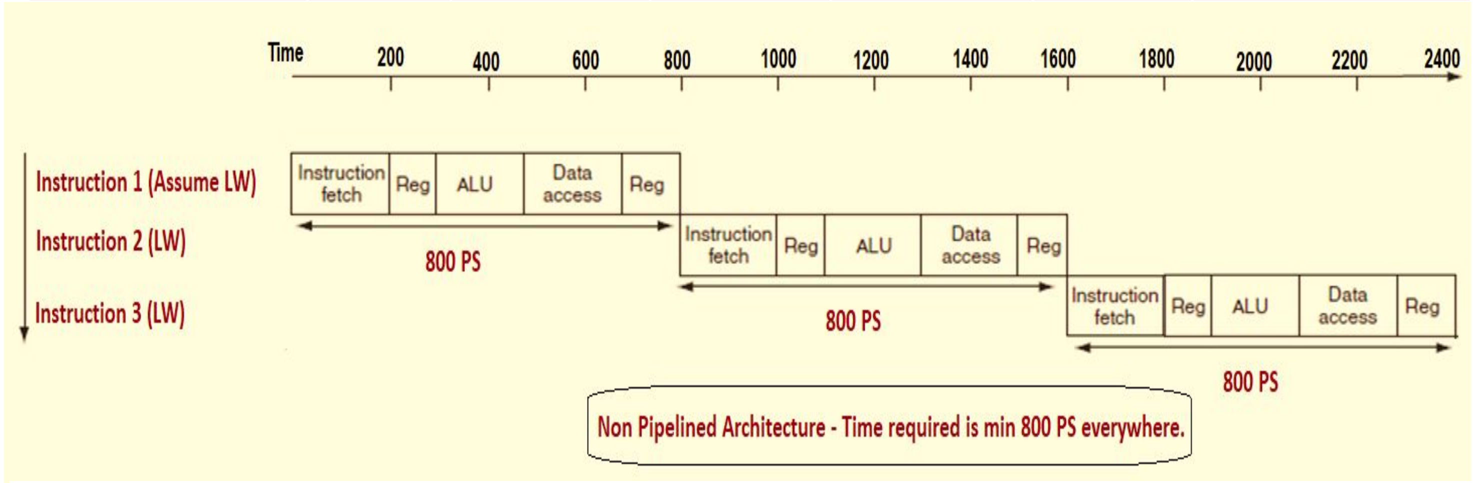
Instruction	Fetch	Read / Decode	ALU (Execute)	Memory Access	Write Back	Time Consumed (Total)
LW (Load Word)	200 PS	100 PS	200 PS	200 PS	100 PS	800 PS
SW (Store Word)	200 PS	100 PS	200 PS	200 PS		700 PS
ADD/SUB/AND	200 PS	100 PS	200 PS		200 PS	700 PS
BEQ/BNE	200 PS	100 PS	200 PS			500 PS

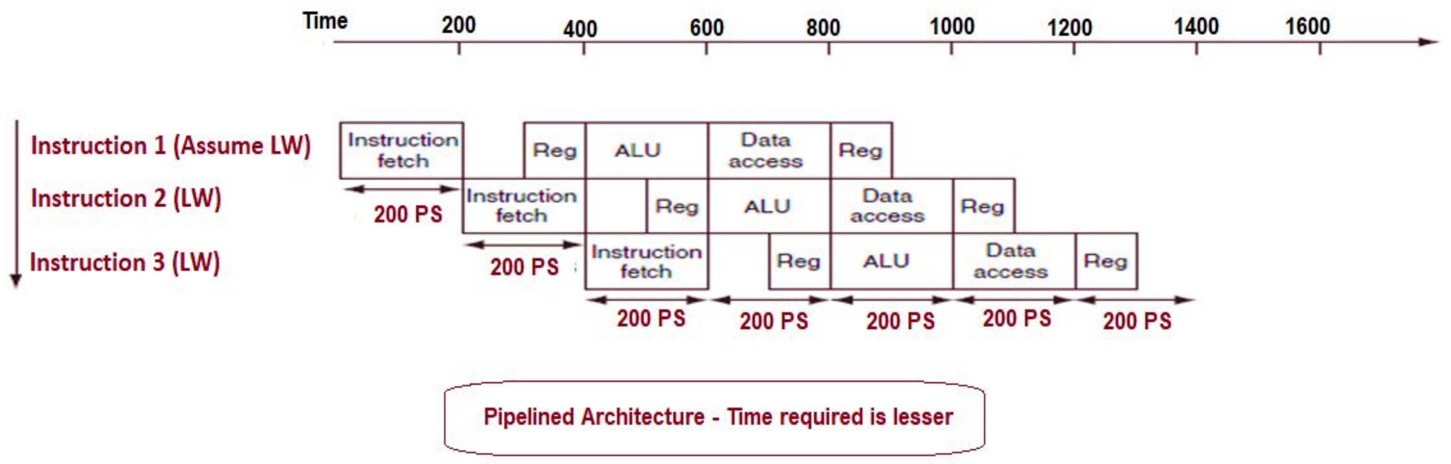
Observations:

- Since, we should also accommodate the slowest instruction without partiality, the time required for LW (Slowest Instruction) – 800 PS becomes the time required for all the instructions in the list above.
- Hence the total time required would 4 * 800 = 3200 PS. Or one can say this way, time between the first and fourth instruction would be equal to 3 * 800 or 2400 PS.

Contd.,
A Diagrammatic View shall be handy!!

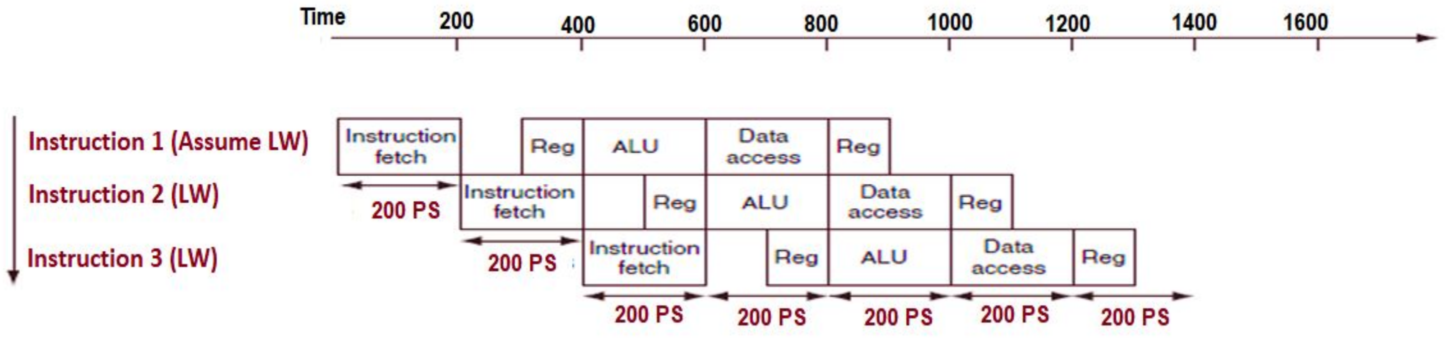
Instruction	Fetch	Read / Decode	ALU (Execute)	Memory Access	Write Back	Time Consumed (Total)
LW (Load Word)	200 PS	100 PS	200 PS	200 PS	100 PS	800 PS
SW (Store Word)	200 PS	100 PS	200 PS	200 PS		700 PS
ADD/SUB/AND	200 PS	100 PS	200 PS		100 PS	600 PS
BEQ/BNE	200 PS	100 PS	200 PS			500 PS





Contd.,

- **Now comes the very important point!** The pipelined execution clock cycle must have the worst-case clock cycle of 200 PS even though some stages take only 100 ps. (See the figure below)
- **Pipelining is still better with respect to performance. Means it would need not as much time as non pipelined architecture needs.**



Pipelined Architecture - Time required is lesser

Can we bring Math!??

- Let us bring in a formula to make it more mathematical.
- Assuming that stages are perfectly balanced, One can find the time between instructions in MIPS (Pipelined) is presented by

$$\text{Time between instructions}_{\text{pipelined}} = \frac{\text{Time between instructions}_{\text{nonpipelined}}}{\text{Number of pipe stages}}$$

* Conditions Apply – Under Ideal Conditions.

Let’s get better. Learn the hazards now! Session 3

Shriram K Vasudevan

Pipeline Hazards

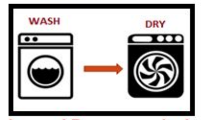
- Can we define a hazard?
  - When you are not in a position to go to a step next, one can term it a hazard.
- Can we define hazard with respect to Computer Architecture pipelining?
  - Yes, Very much important it is.
  - When in pipelining, if the next (successive) instruction cannot execute in the ensuing clock cycle, the condition can be termed a hazard.
  - Let us learn more about this.

Pipeline Hazards – Structural Hazards

- There are 4 type of hazards. Let us start with a easier one – Structural hazard.
- When one hears the term structure, it can be realized that it is connected to hardware. When the hardware is incapable to support the combination of the instructions (which pipelining is all about) in a same clock cycle, **we cite this an issue.**
- Let us take the same example of "Washing". If the equipment washer and dryer are separate and available as two different units, pipelining would not be a problem. But, if in case, it is all available as one unit instead of being two different, independent units, problem starts. (Until both the stages are over, the next instruction cannot use it)
- Secondly, if the first person who has finished the washing has forgotten to take the clothes away or involved in something else, then the next person should wait until the first one lets the equipment free.
- **This is a hazard! You can't get the pipelining done!**

Ideal for pipelining

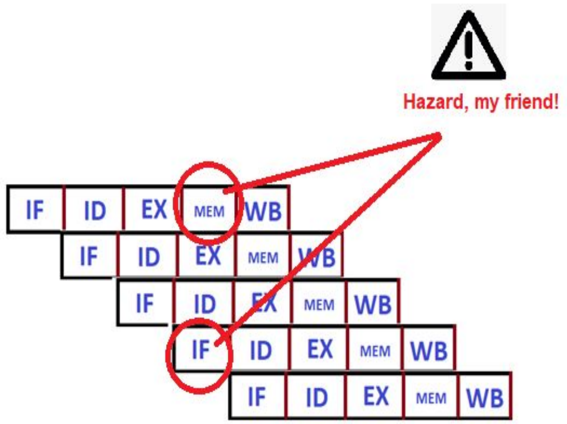
Structural Hazard



Washer and Dryer as a single unit.

Contd.,

- As we noticed in the RISC Vs. CISC difference, MIPS supports pipelining and the instruction set is constructed which could enable pipelining naturally. Also, the structural hazards are avoided from design perspective.
- There is a challenge:
- Assume we have single memory unit (Not dual).
- So what is the challenge?
  - Simple. When the first instruction is accessing the data from the memory (MEM stage), the fourth instruction would try to fetch the instruction from the same memory. **(One memory – Cant be accessed twice at the same point in time)**
  - So, what is the remedy?
    - Without two memories, pipelining may meet with the structural hazard.



Moral of the story?

Though expensive, for pipelining to be done, we need sufficient components to stay away from structural hazard.

Session – 4
Data Hazard

Let’s understand the next hazard – Data Hazard.

- **This is very interesting “Hazard”. Well, data hazard occur when the pipeline is stopped (halted, technically stalled) because one stage must wait for another one to get completed.**
- **Okay, let us take an example. You have a dress pair – A matching Shirt and Trouser. You wanted to get both (the pair) to be washed and ironed.**
- **Assuming you washed both, while folding the clothes after wash, you found the trouser missing!**
- **Now, what would you do?**
- **Simple. Go find the trouser in the room as you can’t miss it.**
- Until, the match is found (i.e. the time you spend in the search) rest of the clothes which actually could have been folded and ironed, will be forced to wait.
- **What is this scenario? This is a hazard!**
  - **Yes, this has to be given care.**

Contd., Let's get this more technical.

- Well, shall we make it technical? I.e. we shall move to pipelining related view of this hazard.
- When would data hazard arise in pipeline?
  - Simple. When there is a dependency **for one instruction with another earlier instruction which is still in pipeline. (I need the result which is yet to be given by the previous one, without that result, I cant move forward)**
- Let us take this up with some instructions.
- We have the first instruction as **ADD. (MIPS)**
- The subsequent instruction is **ADDI**, which rely on the result of the previous one. (I mean, B needs A to give the result, **ADDI is dependent on ADD**)

```
ADD $s1, $t1, $t2     #Result of this operation stored in S1
ADDi $s1, $s1, 5      #s1 is used here.
```

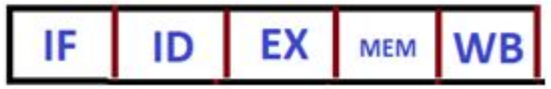
Contd.,

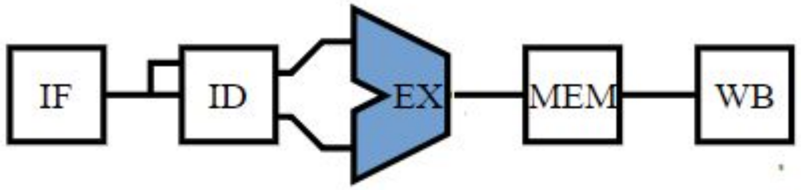
- Now, do we have a solution?
- Yes, There is a solution.
- For the sequence presented below (ADD followed by ADDI) , ALU shall compute the SUM first and as soon as it is available it can be supplied to ADDI.
- Bypassing AKA forwarding is the remedy!
- Some extra hardware (*at extra cost, but fine as we avoid the hazard) towards retrieving the missing item from the internal components is called BYPASSING AKA FORWARDING.

```
ADD $s1, $t1, $t2     #Result of this operation stored in S1
ADDi $s1, $s1, 5      #s1 is used here.
```

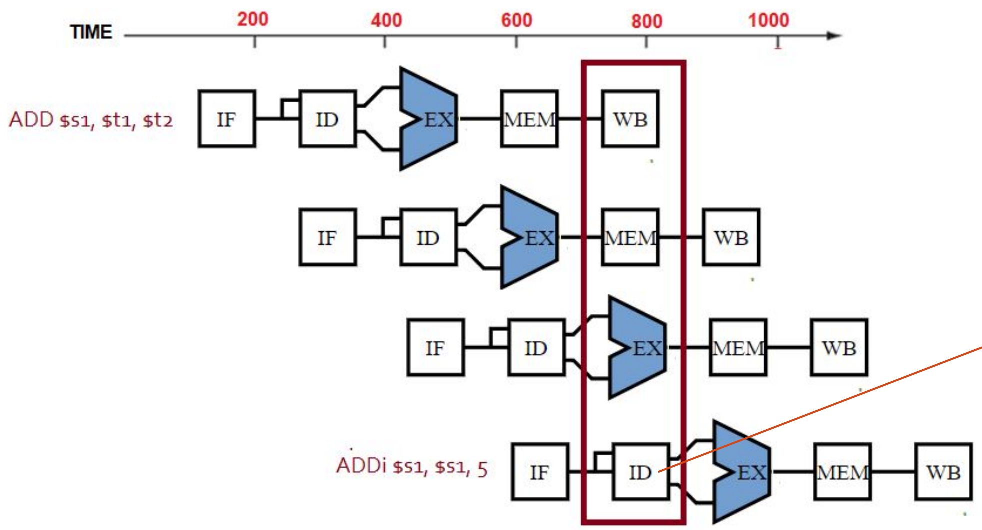
Can we realize this through a Diagram…

```
ADD $s1, $t1, $t2     #Result of this operation stored in S1
ADDi $s1, $s1, 5      #s1 is used here.
```

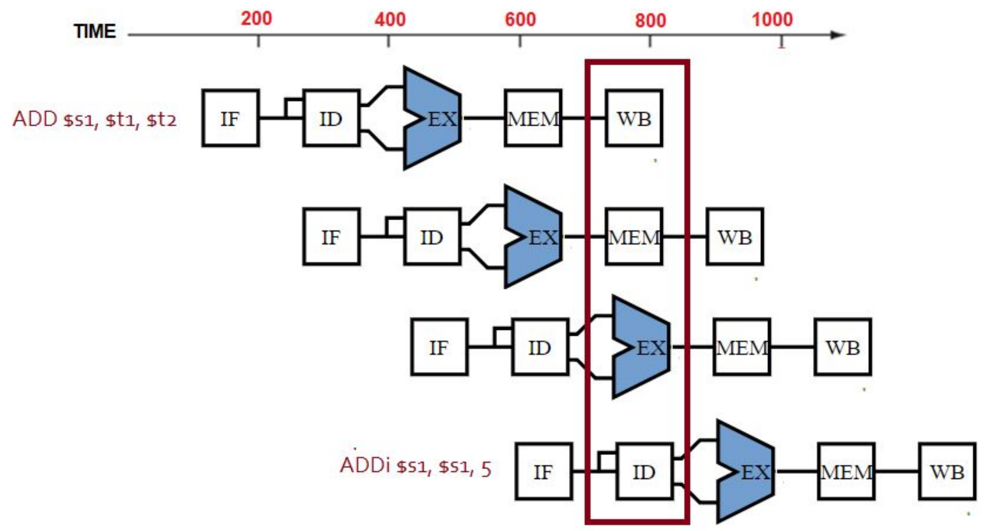




Please see the symbols for the each stage, So, for the above two instructions, how do you represent?

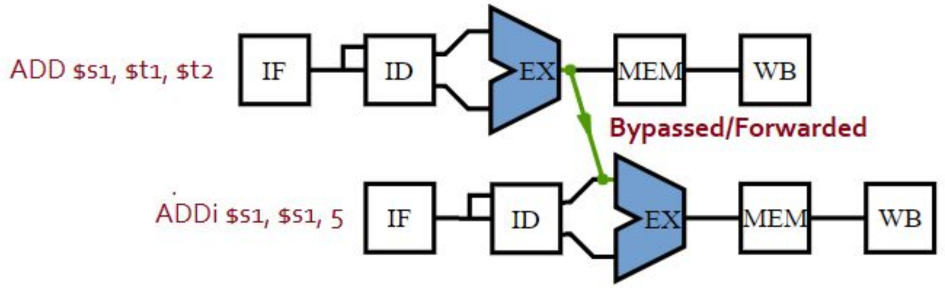


This is a hazard!



So, what is the solution.

- Now, let us think. The result is available for the first instruction right after the ALU (EX). So, why to wait till last stage (Means, can we avoid hazard??)
- Yes, see the below solution.
- **Lets forward the value in $s1 after the execution stage of the add instruction as input to the execution stage of the ADDi instruction.**

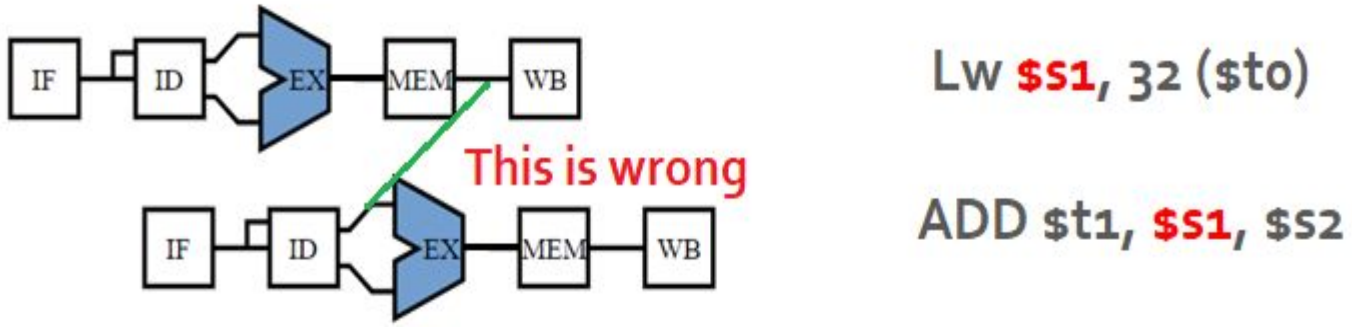


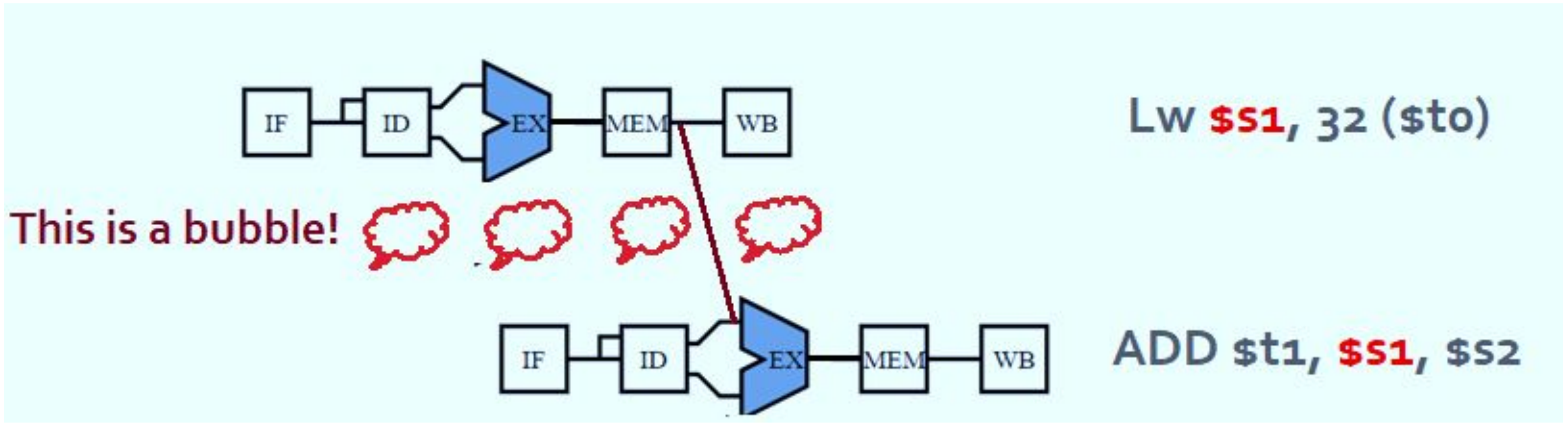
Let’s handle a situation.

- When an R Type instruction follows the Load, condition is different and difficult to handle.
- Let us handle that too.
- Assume that the first instruction is a Load. And, it loads Register $S1. **Lw $s1, 32 ($to) - is the assumption.**
- **The next instruction – ADD $t1, $s1, $s2.**
- Can we make it a diagram?

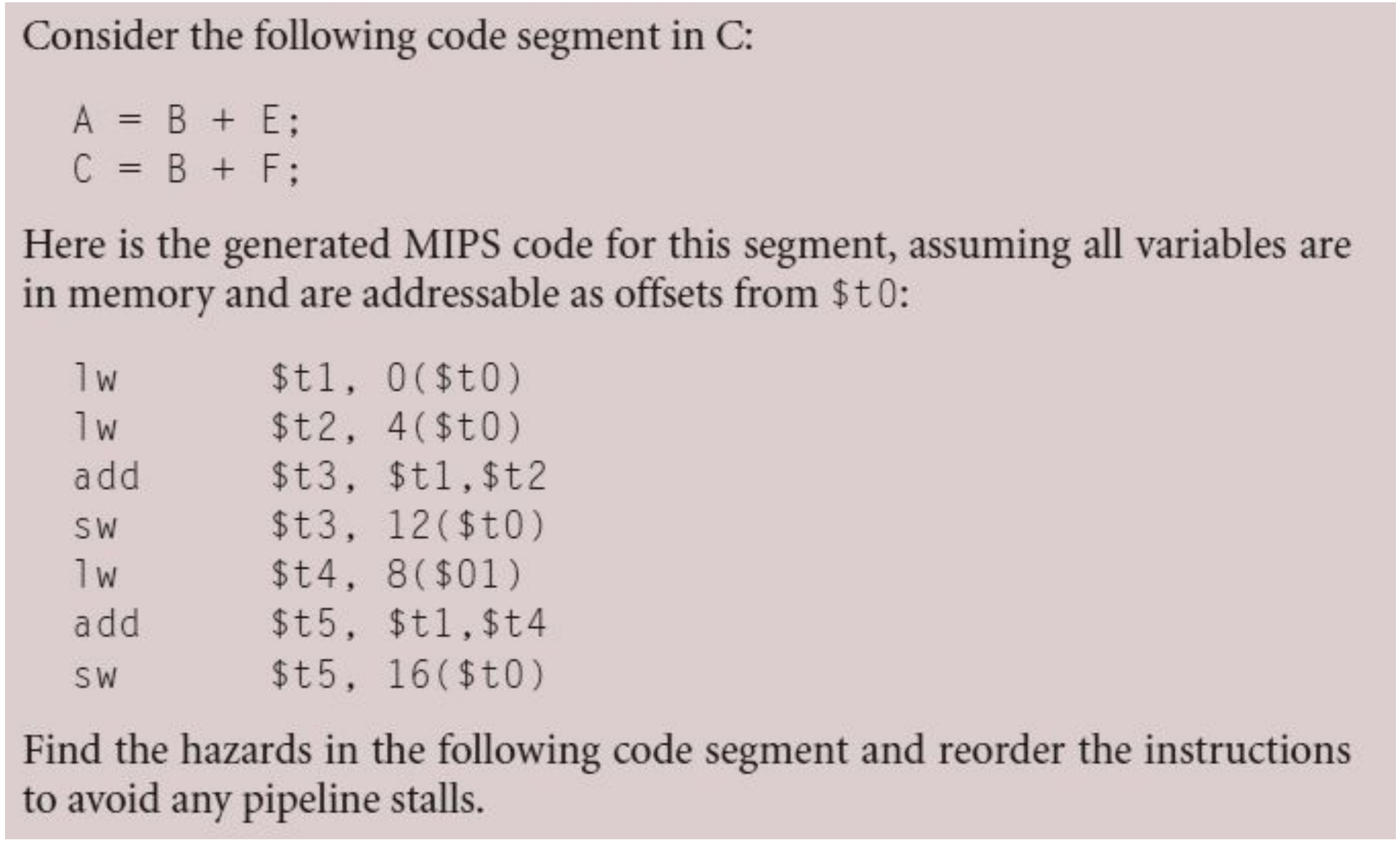
Contd.,

- **The data which has to be fed to the instruction add is available at the fourth stage of the first instruction.**
- **Is it not late? Yes, it is very late and forwarding is not even going to work. So, what works? Let's stall one stage! This is called DATA HAZARD and Stalling is the solution.**
- The stall is referred as bubble as well!





Can we have some math?



Consider the following code segment in C:

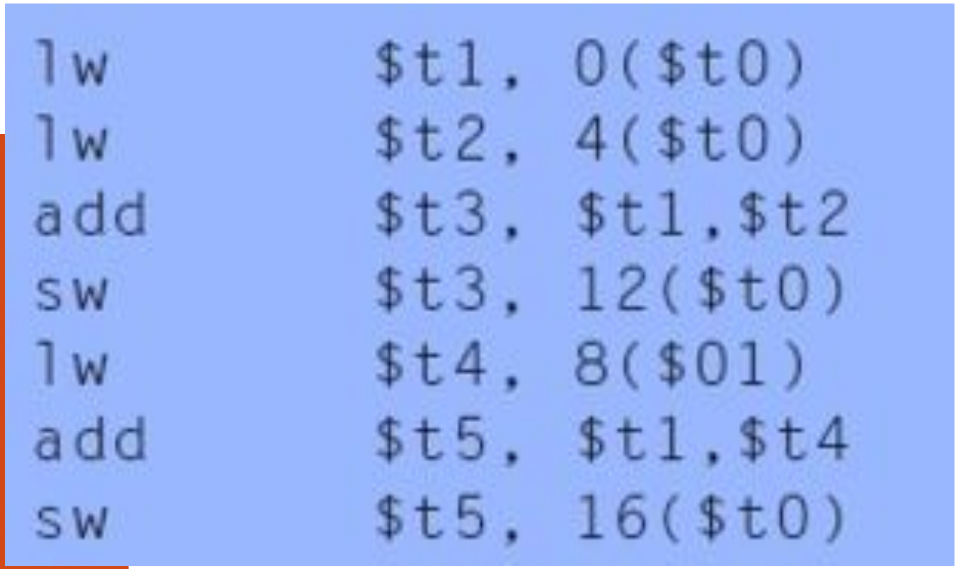
```
A = B + E;
C = B + F;
```

Here is the generated MIPS code for this segment, assuming all variables are in memory and are addressable as offsets from `$t0`:

```
lw      $t1, 0($t0)
lw      $t2, 4($t0)
add     $t3, $t1,$t2
sw      $t3, 12($t0)
lw      $t4, 8($01)
add     $t5, $t1,$t4
sw      $t5, 16($t0)
```

Find the hazards in the following code segment and reorder the instructions to avoid any pipeline stalls.

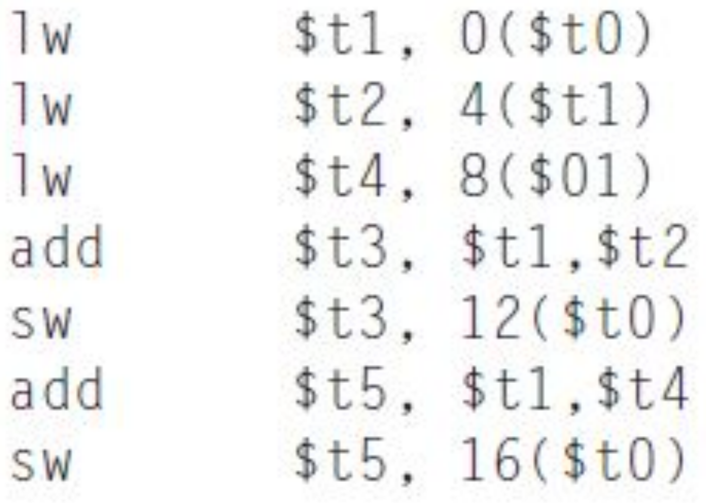
Contd.,



```
lw      $t1, 0($t0)
lw      $t2, 4($t0)
add     $t3, $t1,$t2
sw      $t3, 12($t0)
lw      $t4, 8($01)
add     $t5, $t1,$t4
sw      $t5, 16($t0)
```

- Can you notice? ADD after Load. This is a hazard. Both ADDs have hazard.
- Let us modify this to get things into a better shape.
- Moving up the third lw instruction eliminates both hazards:

BETTER



```
lw      $t1, 0($t0)
lw      $t2, 4($t1)
lw      $t4, 8($01)
add     $t3, $t1,$t2
sw      $t3, 12($t0)
add     $t5, $t1,$t4
sw      $t5, 16($t0)
```

Control Hazards

Session 5

What is control hazard?

- Well, this is an interesting hazard! Control hazard is all about "Need to make decision".
- Let us take an example.
- We have to wash Uniforms of school children.
  - We cannot just like that wash the clothes with random choice of the detergent quantity and sanitizer liquid.
  - **You should make the combo powerful to clean the dresses. At the same time, it should not damage the clothes.**
  - So, what do we do? We shall probably wash one set of cloth and see the result if the combination is powerful and at the same time, it should not damage the clothes. (Clothes should get dry after the wash and only then you can determine if the combination is right)
  - Example of someone going to a store to buy components where he is being commanded by someone else can be taken as reference. (I shall explain this)

Contd., What can be the solution?

- *Stalling – What do we do? We wait and see if the combination is correct. Else, we shall try the same with different combination. Until then, you have to wait. This is a nice approach providing result. But, most importantly, it is slow. (We can't afford this being slow)*
- Now, Sir, where do we connect this with computer architecture?
- **Can you recollect the way BEQ/BNE worked? I mean, Branching. We should decide an action based on what is the result. Correct??**
- The challenges is, pipeline is not powered with the knowledge of what next instruction could be for a branch, well in advance.
- One default solution immediately to be thought of is "**when there is a branch, wait until the outcome is available. It will give the pipeline knowledge of which instruction to be fetched after the execution of a BRANCH.**"

Contd.,

- What can be the solution? Well, we can try the following.
  - Add some extra hardware which would enable you to test the register and through which branch address can be computed and PC can be updated during the second stage of pipeline.
- *The next option is to go for prediction! Well, lets predict.*
- *Let us take the same washing scenario.*
- *We said that* "**You should make the combo powerful to clean the dresses. At the same time, it should not damage the clothes".**
- **Now, I say that " If you are very sure about the combination of the washing powder, sanitizer liquid and water, go ahead with full belief to the second set of dress while the first one is being dried" Means, have confidence, predict!**
- *Well, now few points are to be asked!*
  - *If everything is correct, I mean the prediction (the combination) is correct, pipelining will not be affected and in fact, it wont slow down at all. (Since, the result would favor the prediction)*
  - *When luck does not favor, things go different. What's the solution? Redo with the load which had been taken next with prediction.*
  - *Well, how do computers do this? Computers can as well predict when the branching is handled. So? How?*
    - *Simple, Handle it as always branch would not be taken and proceed. I.e. Branch Untaken!* ☹
- **So, what is the moral of the story?** IF EVERYTHING IS RIGHT, PIPELINING COULD PROCEED AT HIGH SPEED AS EXPECTED! IF BRANCHING IS TAKEN, STALLING IS UNAVOIDABLE! ☹

So ?

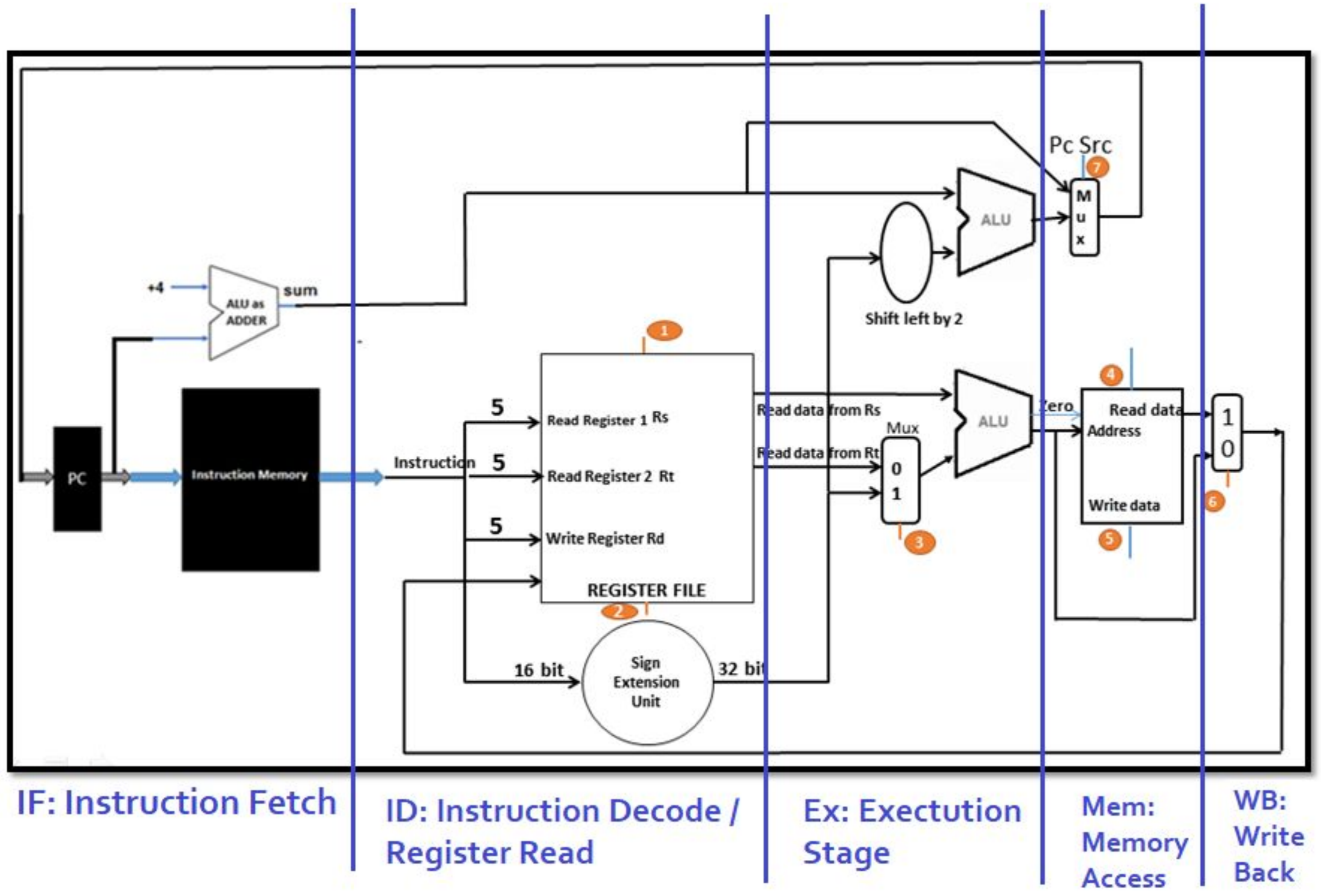
branch prediction A scheme of resolving a branch hazard that assumes a given outcome for the branch and proceeds from that assumption rather than waiting to ascertain the actual outcome.

Pipelining and Datapath

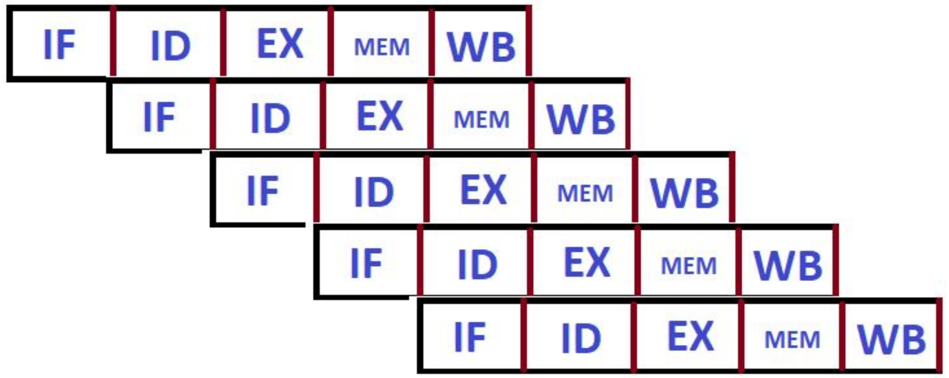
Session - 6

Pipelining with Datapath – A real interesting stuff!

- The learning gets better and in fact best, here.
- **We talk about the five stage pipeline and it actually relates to the point that – 5 instructions will be in execution at any point in time. (Means, it is really pipelined).**
- **Pipelining steps should be visualized from left to right in the picture. Lets us have the diagram as reference.**



Contd.,



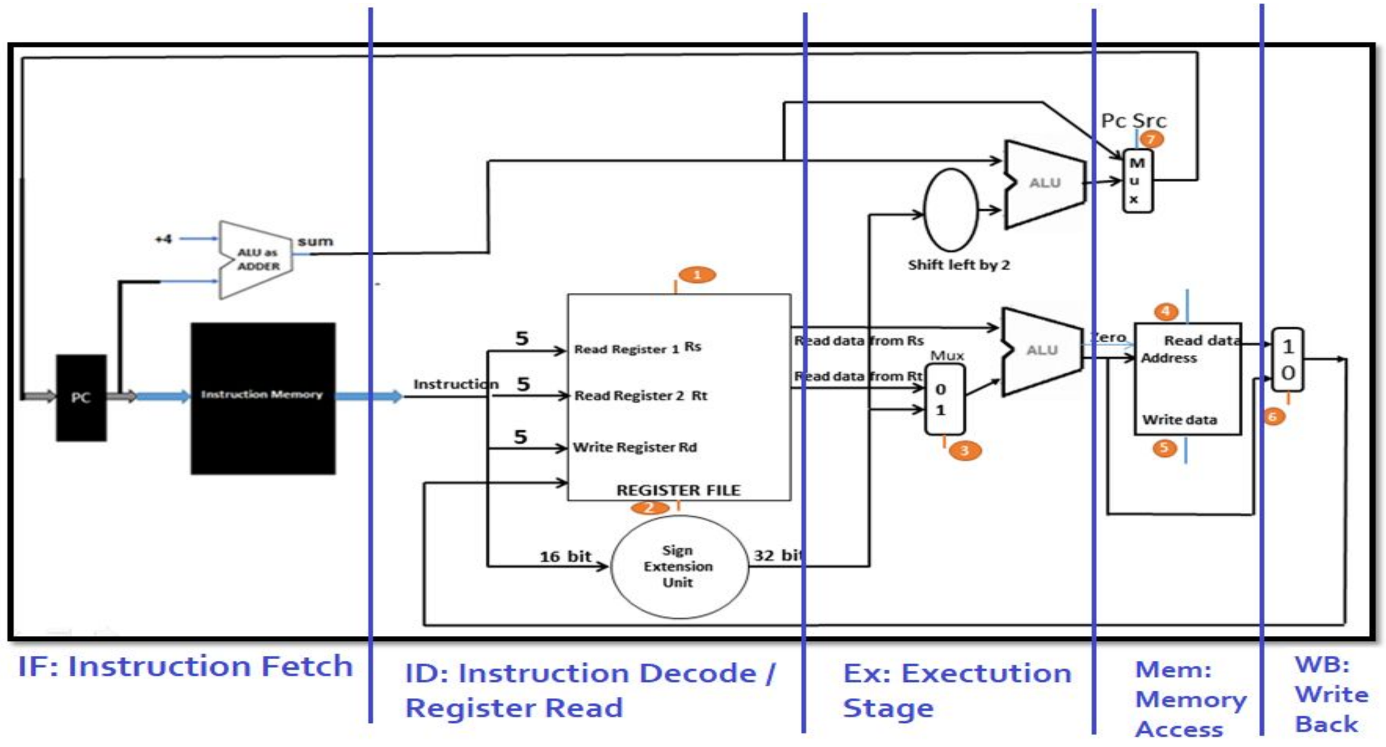
IF - Instruction Fetch

ID - Instruction Decode

EX - Instruction Execution

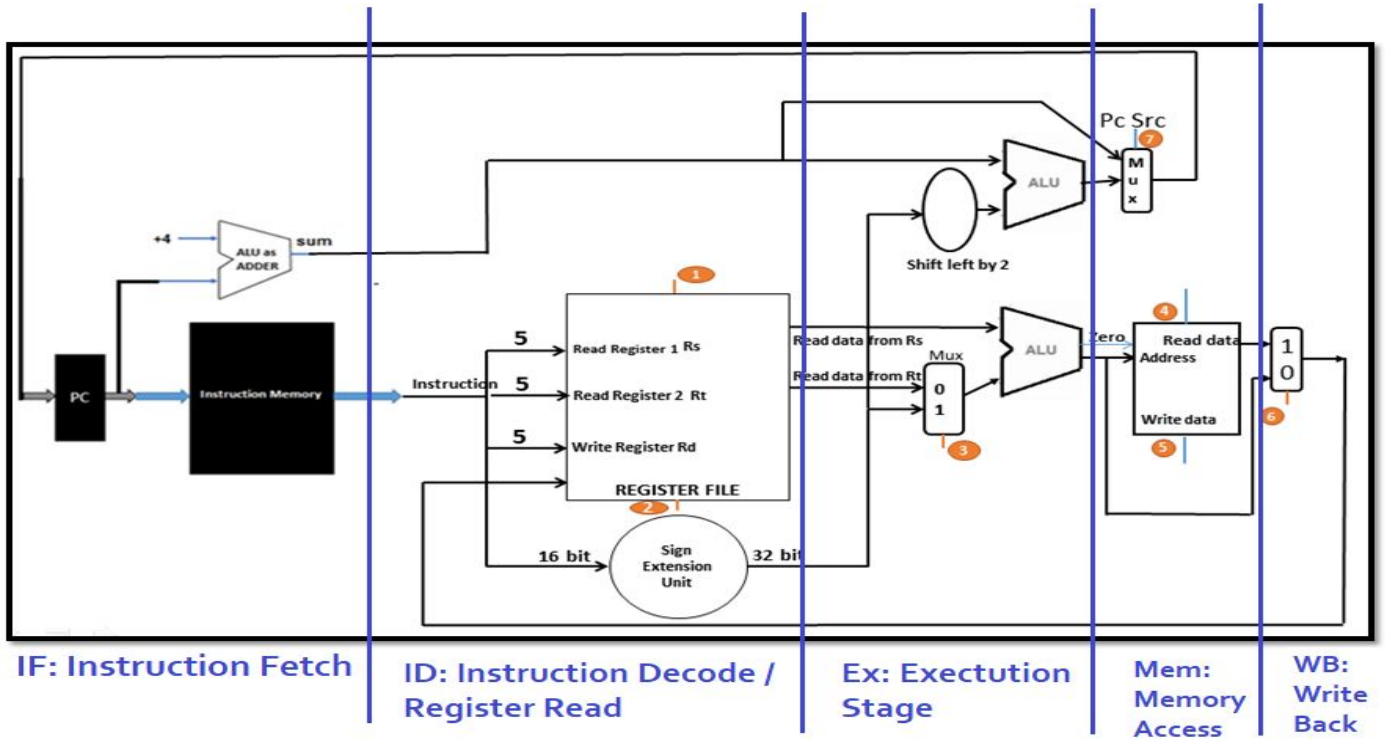
MEM - Memory Access

WB - Write Back



Contd.,

- As we highlighted, the instruction starts from left to right.
- **How do we wash clothes? Remember it? You don't fold the clothes and wash it! ;)**

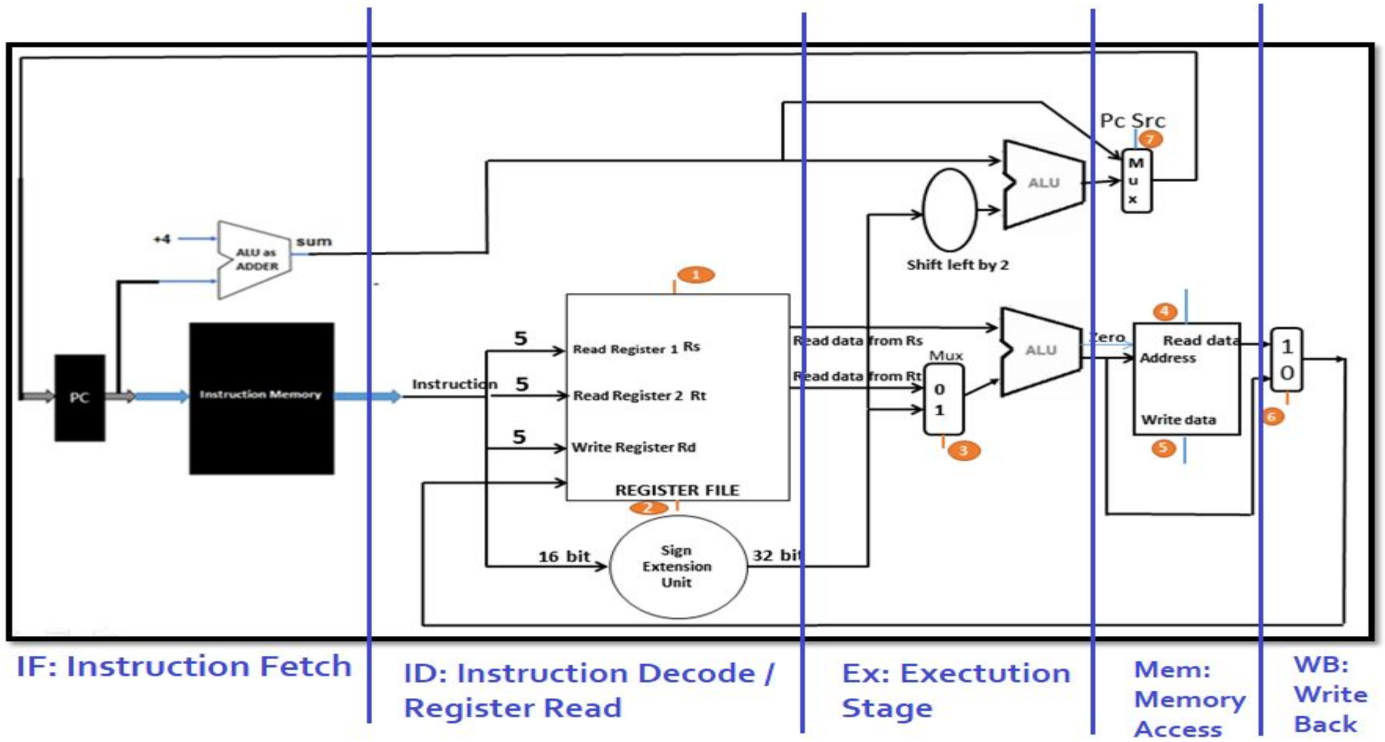


Contd.,

Well, can the left to right rule hold good for everything?

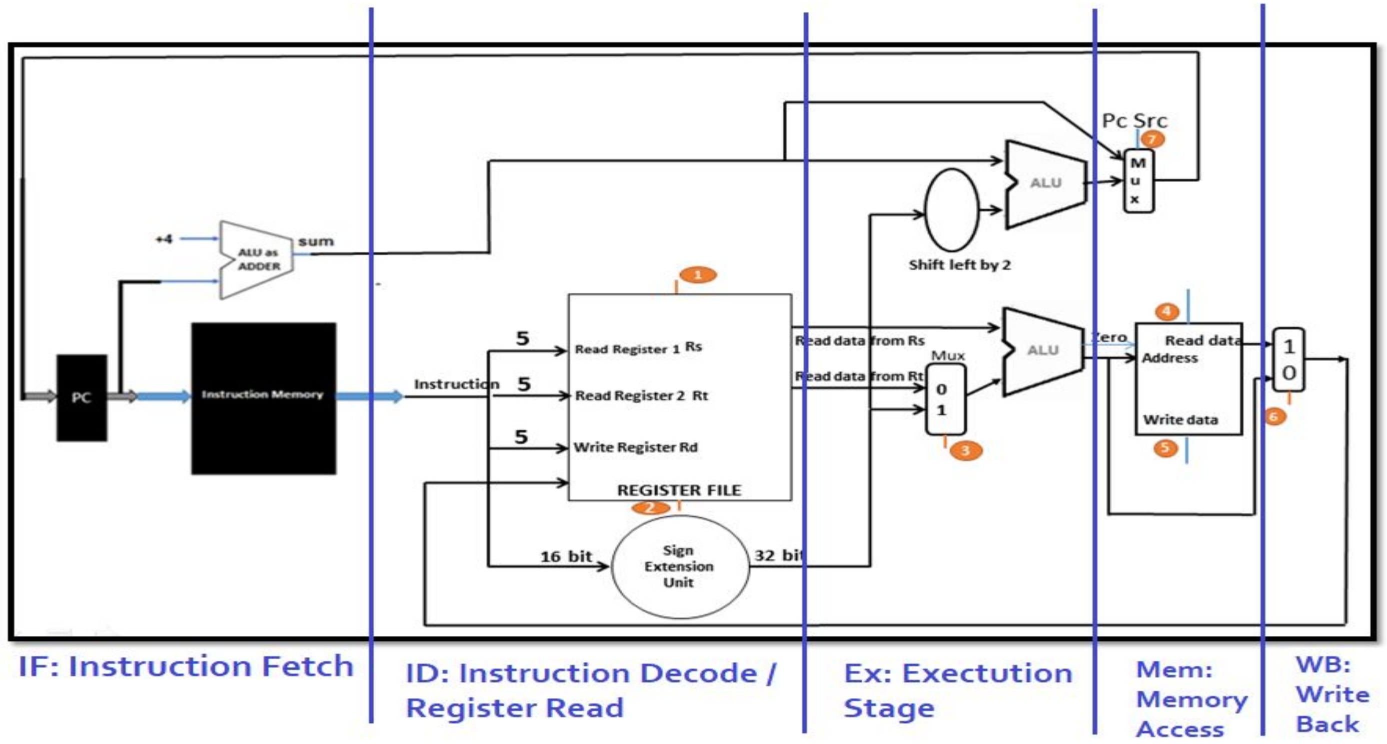
There are some exceptions which we need to understand.

1. **Write back stage shall write the result back to the registers (This is not Left to Right, it is Right to Left)**
2. **PC Update also happens based on the instruction if being Branch.**



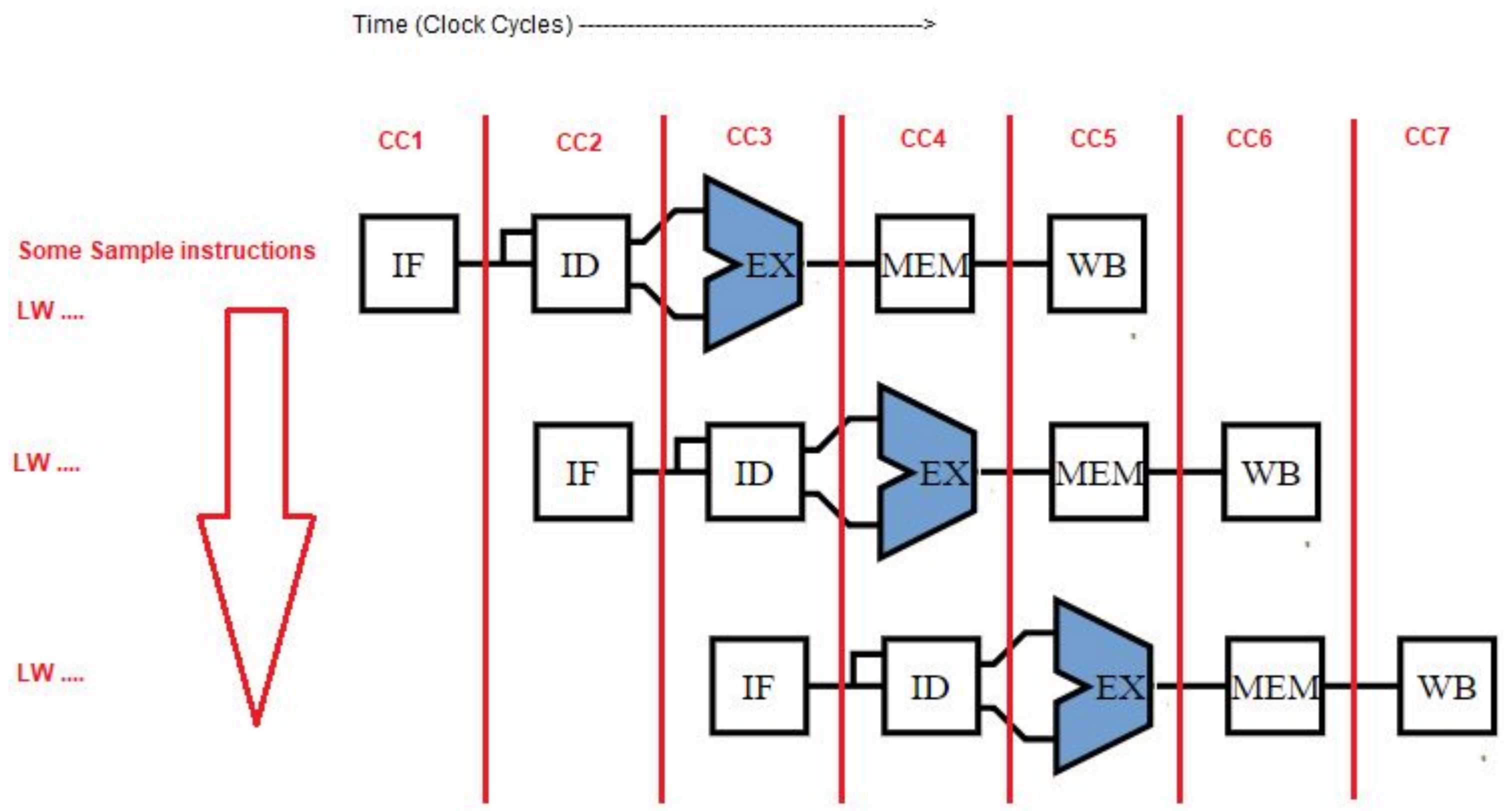
Contd.,

- Ground Rule to be understood:
- Data flow from Left to Right is normal. Nothing will be influenced.
- Data flow from the Right to Left is also fine and shall not influence the currently executing instruction.
    - But, impact would be there on the next pipelined instruction.



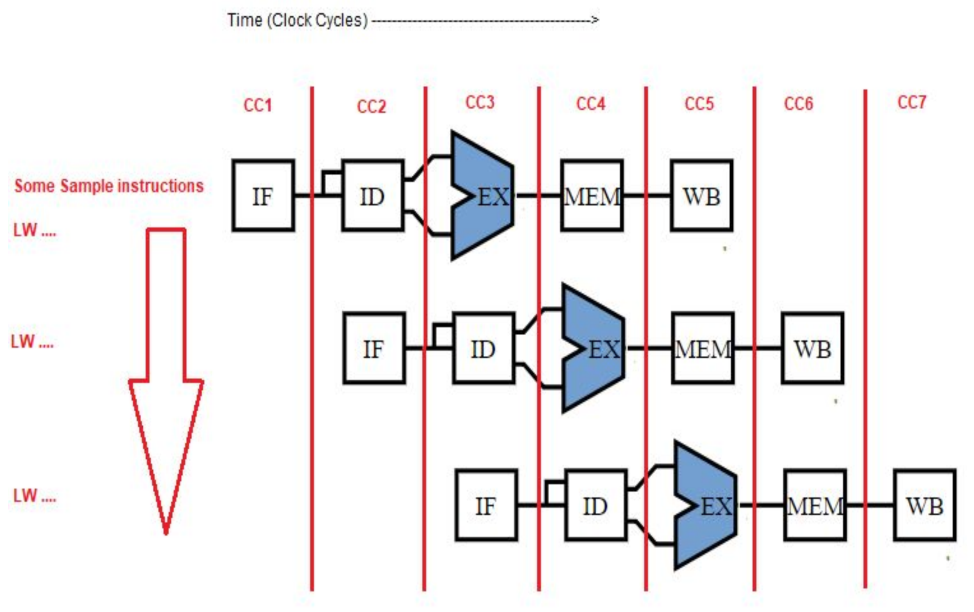
Contd.,

- Understand this point clearly.
  - Pipelining has to be seen this way.
  - Each instruction should be seen as if it has its own datapath.
  - One should see the below diagram to understand this aspect.



Contd.,

- Now, comes the important thing to look at from the RHS figure.
- Three different instructions are available and all of them are to be seen as three different datapath. (Remember?, pipeline improves throughput, but not a single instruction's execution time).
- Remember? We had registers added in the datapath to hold data. This enabled the datapath sharing smooth during the execution.
- See here, from the figure. IM is used by one stage of any instruction and it can be used by other instructions during the other 4 stages.

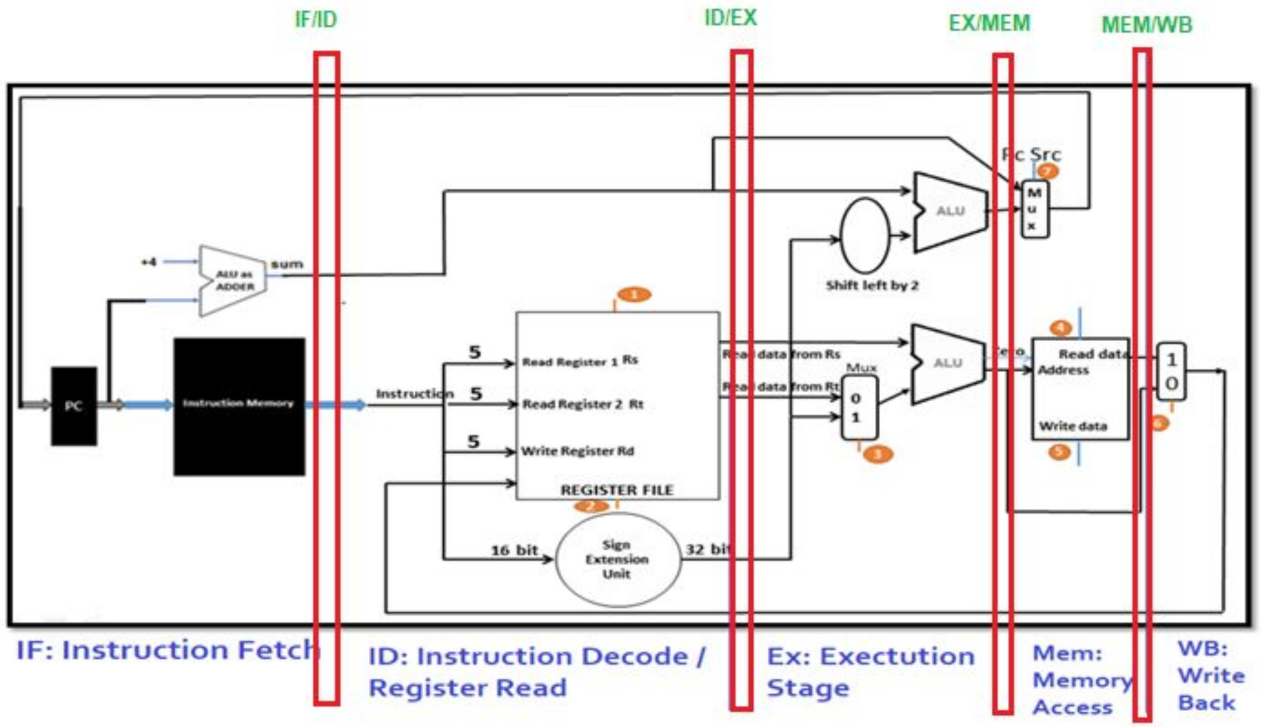


Contd.,

- Now, comes the challenge. We have to retain the values if we need to handover the memory to other instructions in the pipeline. (Means, if it has to be shared, we need to make sure that, the data is not lost.)
- So, what has to be done? Simple. Use registers. Registers are to be used in between to store the values. (**Define register??)**
- **Place the registers at whatever place you need to save the values intermediary!** 😊
- **Come to the washing machine story again, we need to have a basket between each stage to get the clothes stored for the next step to work fine.**

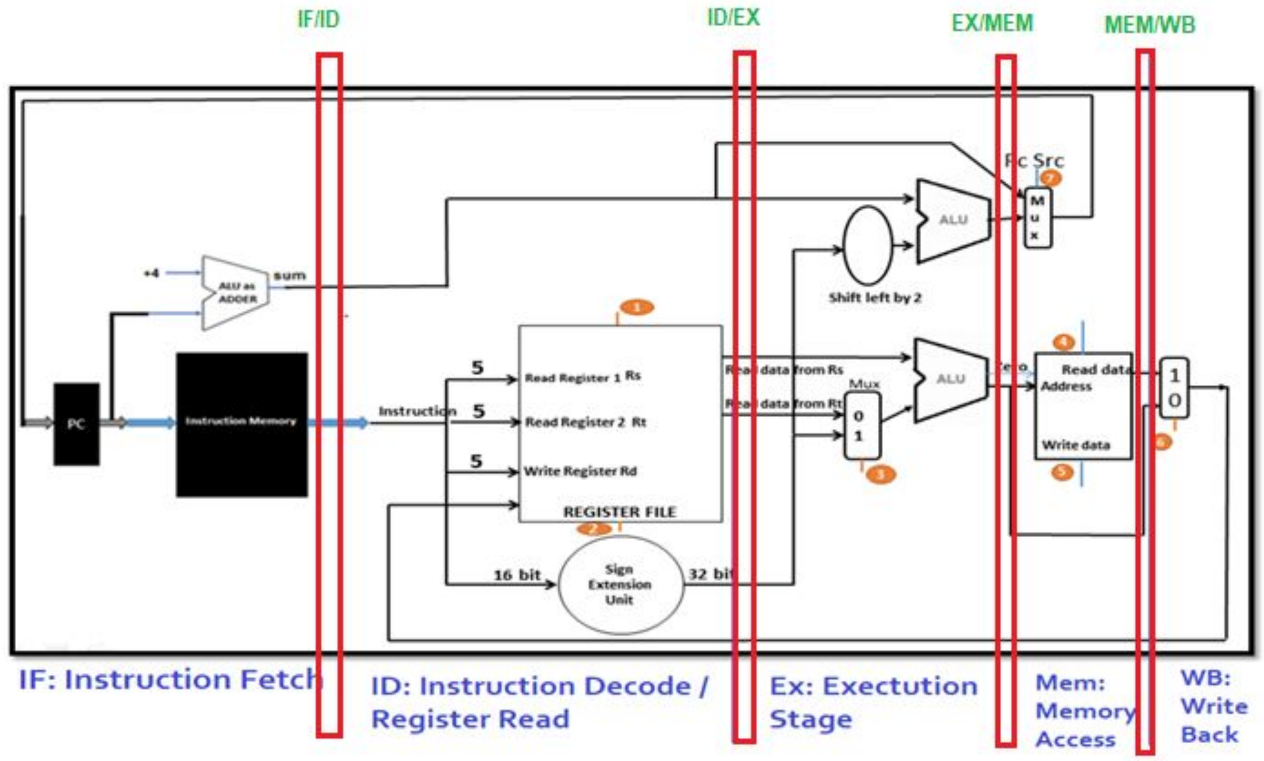
Contd.,

- Now, in the below figure, one can see that registers are included (red in color).
- The naming convention followed here is very meaningful and you could certainly appreciate it.
- Registers between two stages will have the influence of both the stages when it comes to naming.
  - IF/ID is a register between IF and ID.
  - ID/EX is a register between ID and EX.



Contd.,

- Now comes the core point to understand. These registers actually could be seen as a separator of the pipelining stages.
- IF/ID separates the IF and ID, ID/EX separates ID and EX.
- The next question or challenge is here. What would be the size of the register? 32 bit? No. it should be 64 bits. IF/ID has to be 64 bits as it may have to hold the instruction from memory and incremented 32 bit address (PC). Size gets expanded as 64, 128, 97, 64.

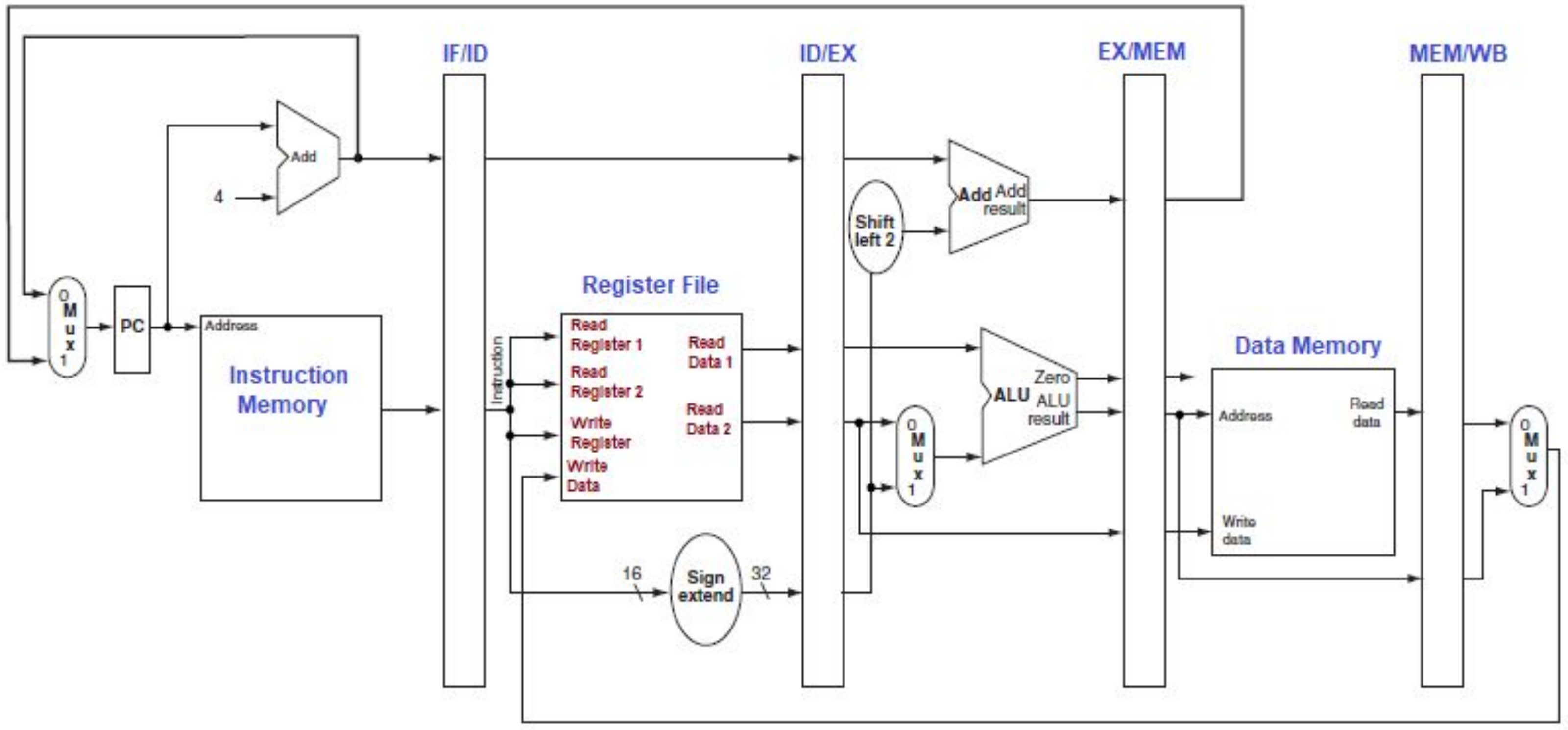


Instructions and Pipelining

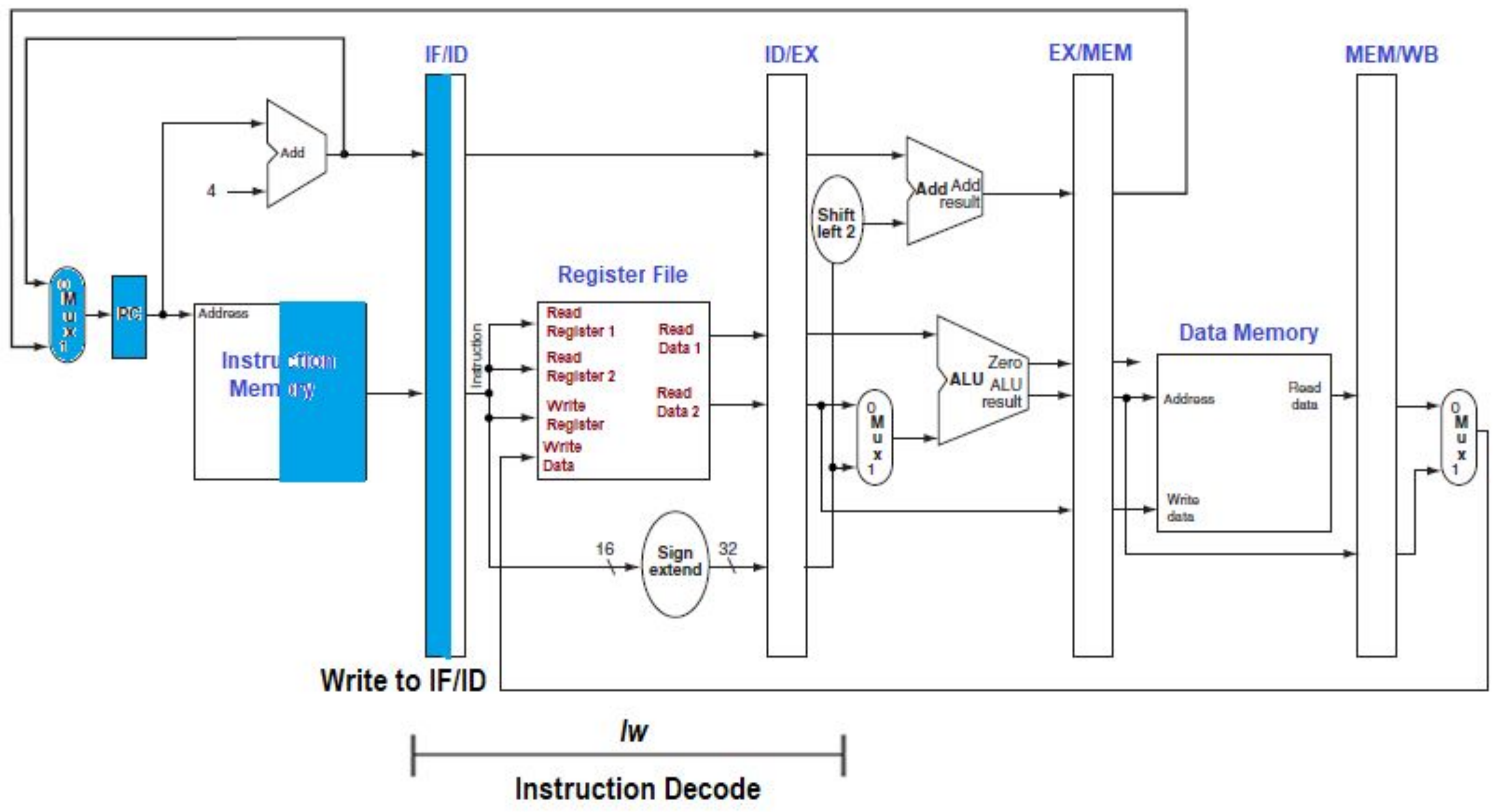
Session 7

Let us learn an instruction with Pipelining

- Let's trace how LW works with the pipelined datapath. We shall trace the steps one after another.



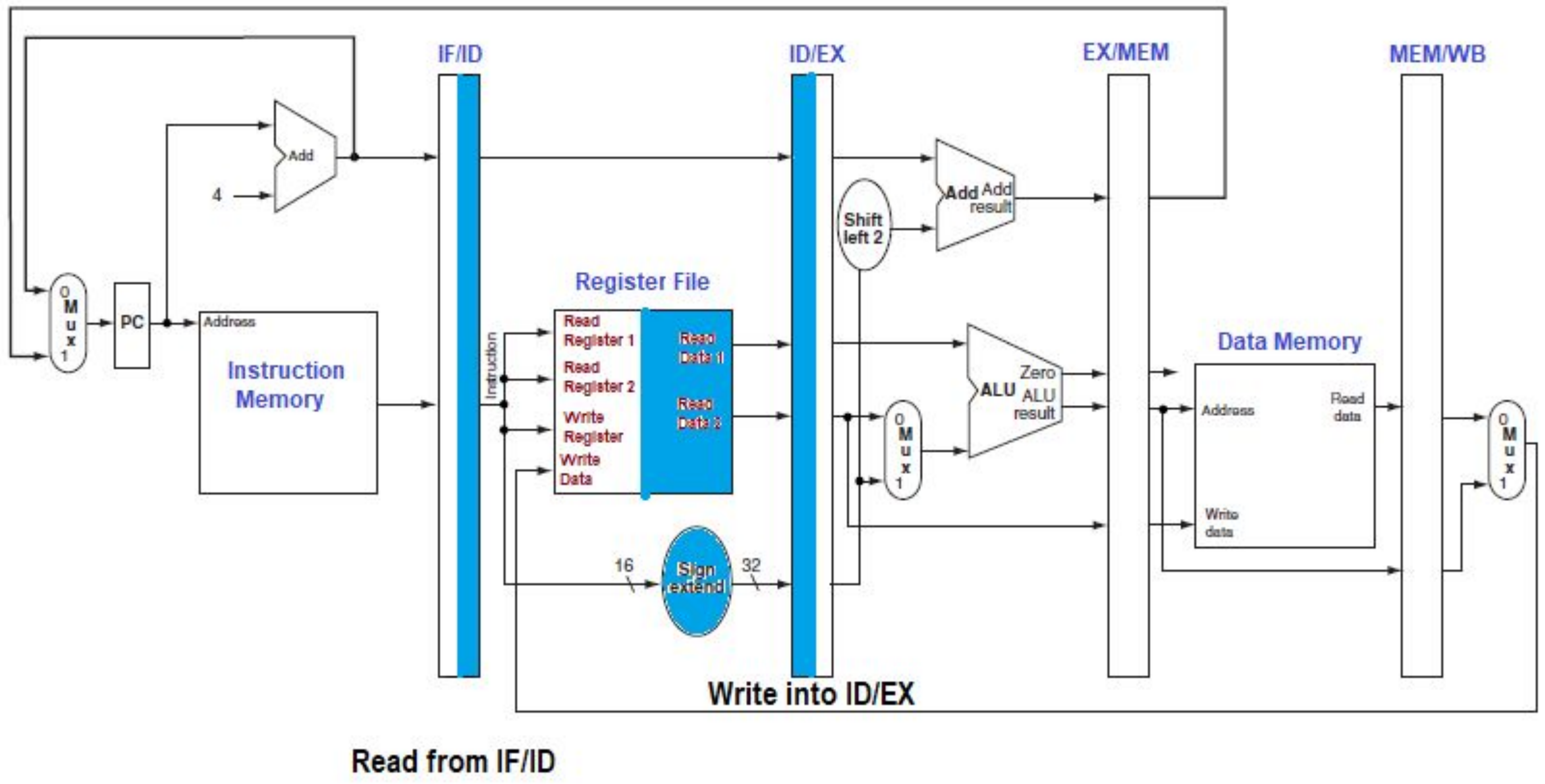
Let’s go one step at a time. – Step 1



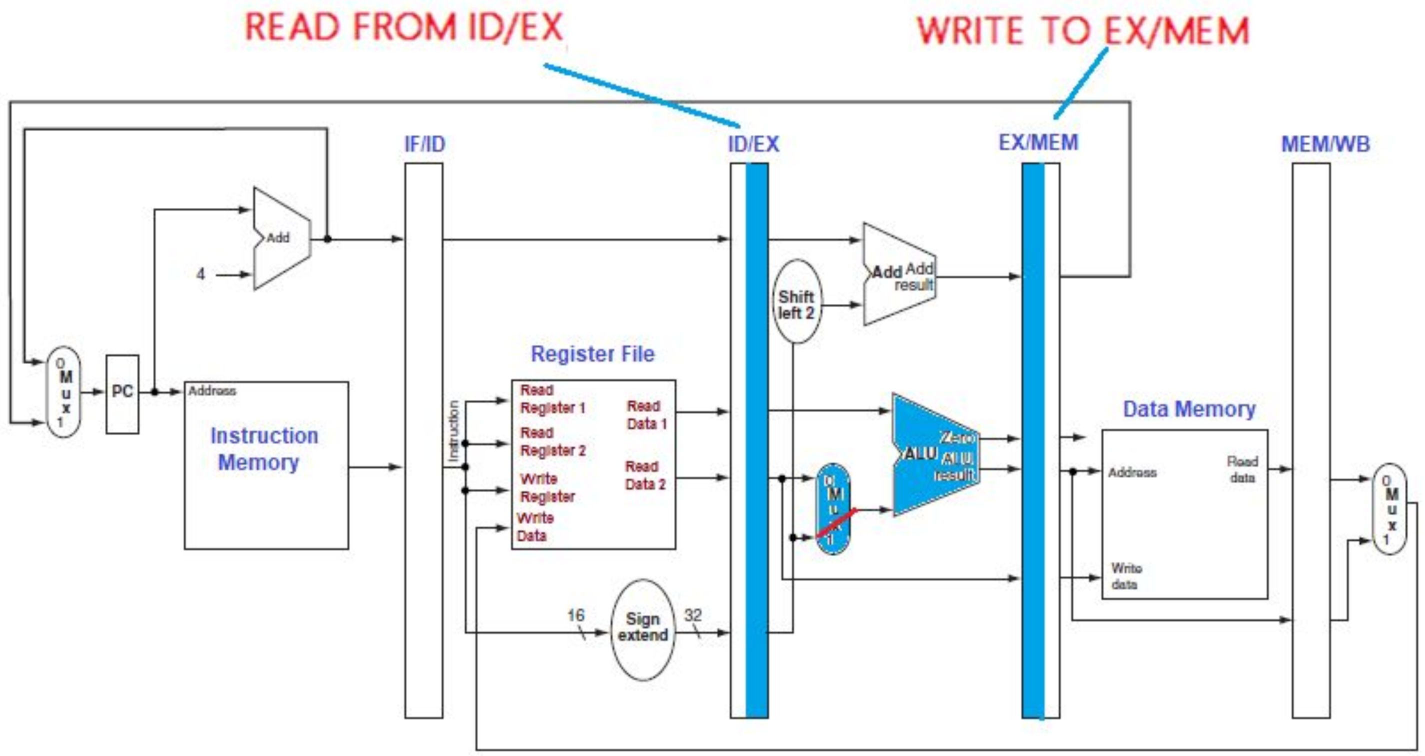
Instruction fetch: The instruction is read from instruction memory (Read to the right, Write to the left- this is the mantra) and write it to the register – IF/ID.

Now, see that the data goes to the register! I read the instruction from the memory and move it to the register.

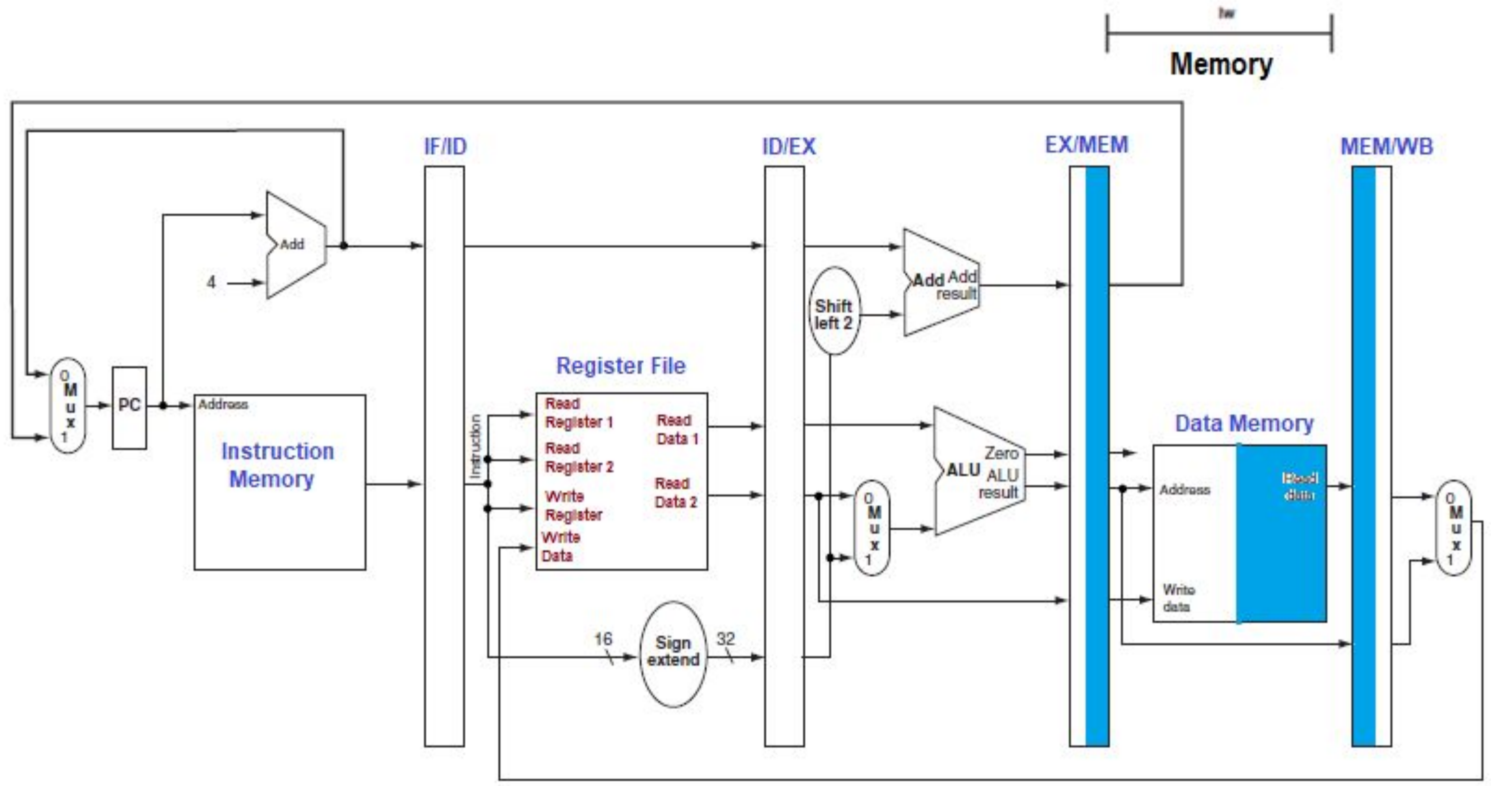
Let’s move to the step 2



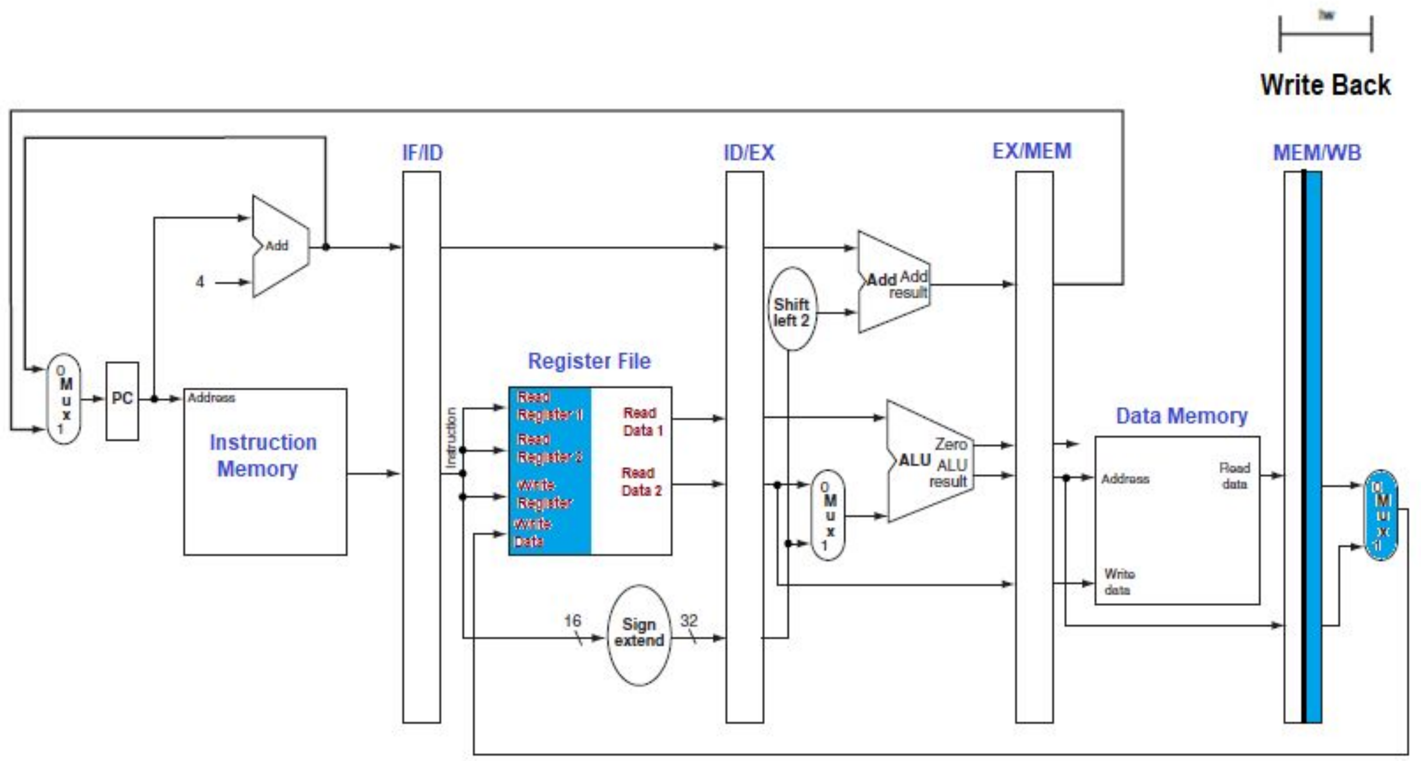
Let us go to step 3



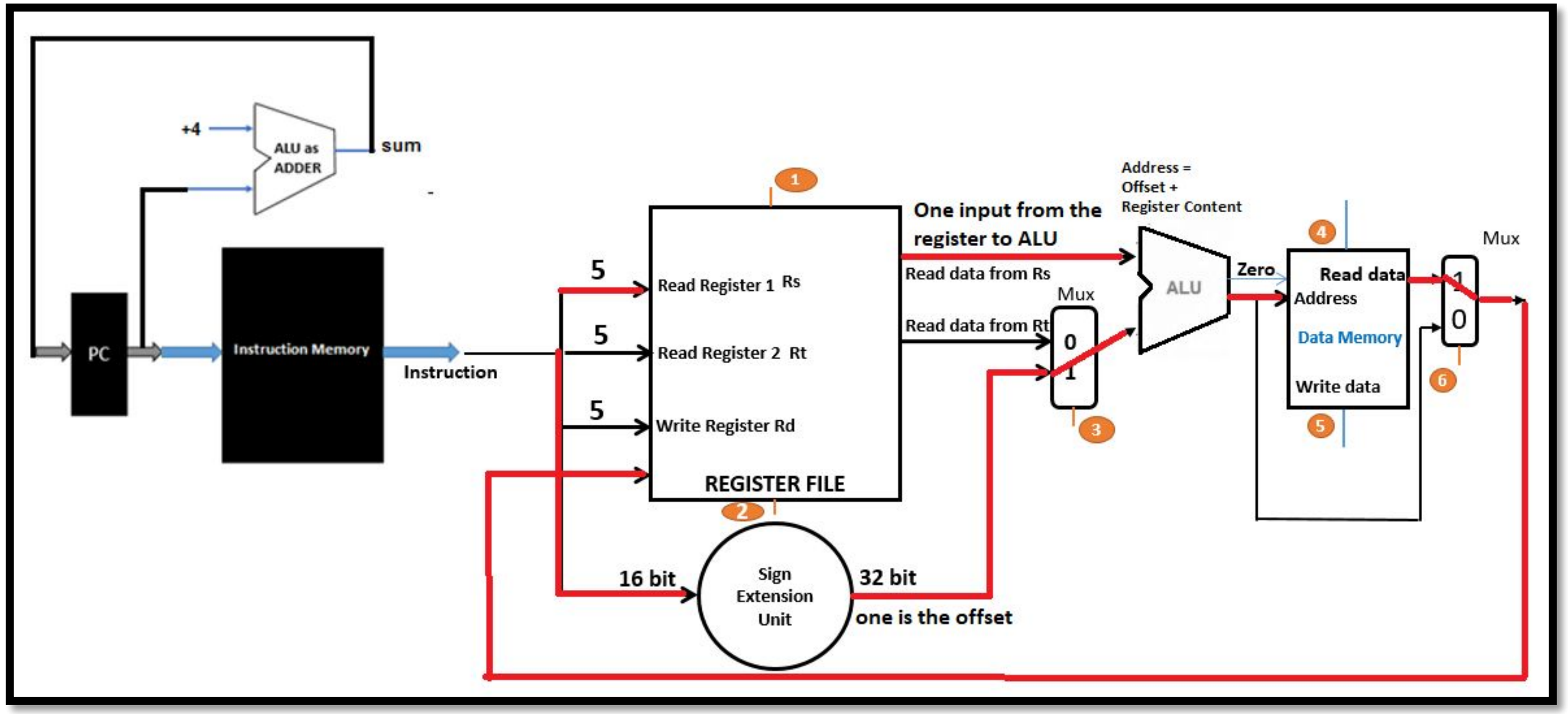
Let us go to step 4



Let us go to step 5 – Yes, this is final.



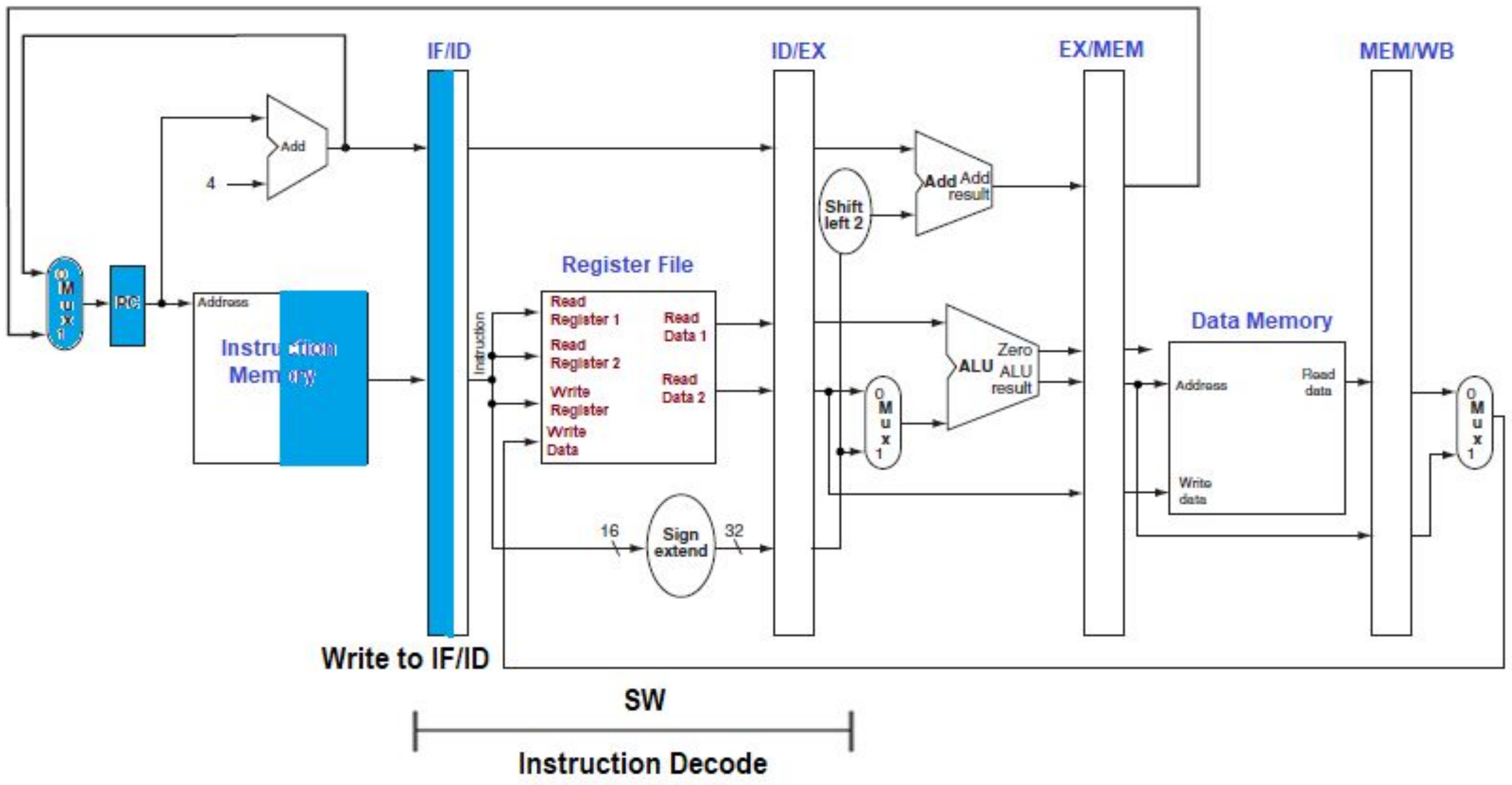
And yes, you got this!



For SW operation. Let’s trace the path for SW.

- Remember? How SW works?
- Let us do the same here.
- First three steps remain the same.
- Hence, the first three diagrams are to be retained.

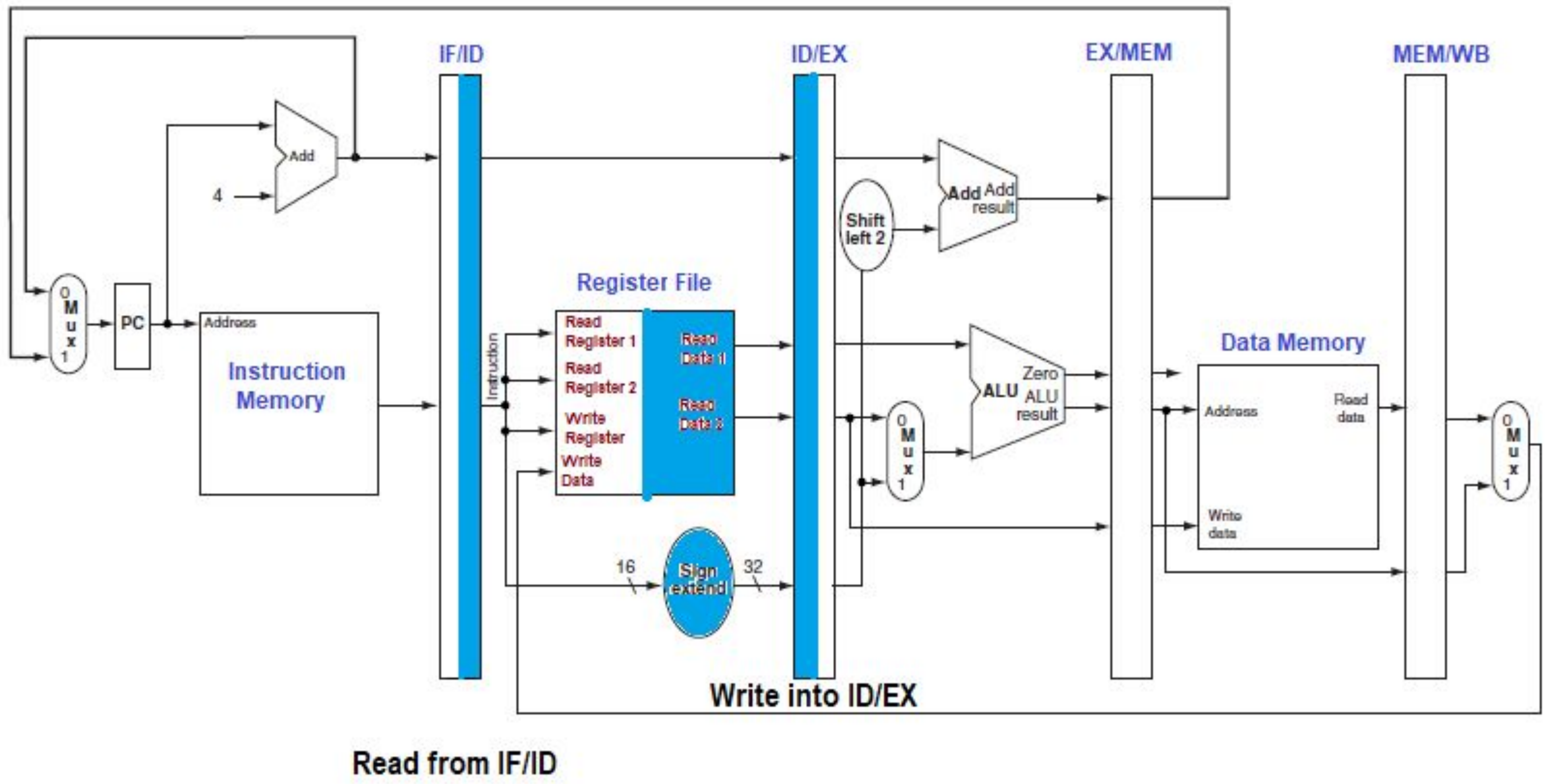
Let's go one step at a time. – Step 1



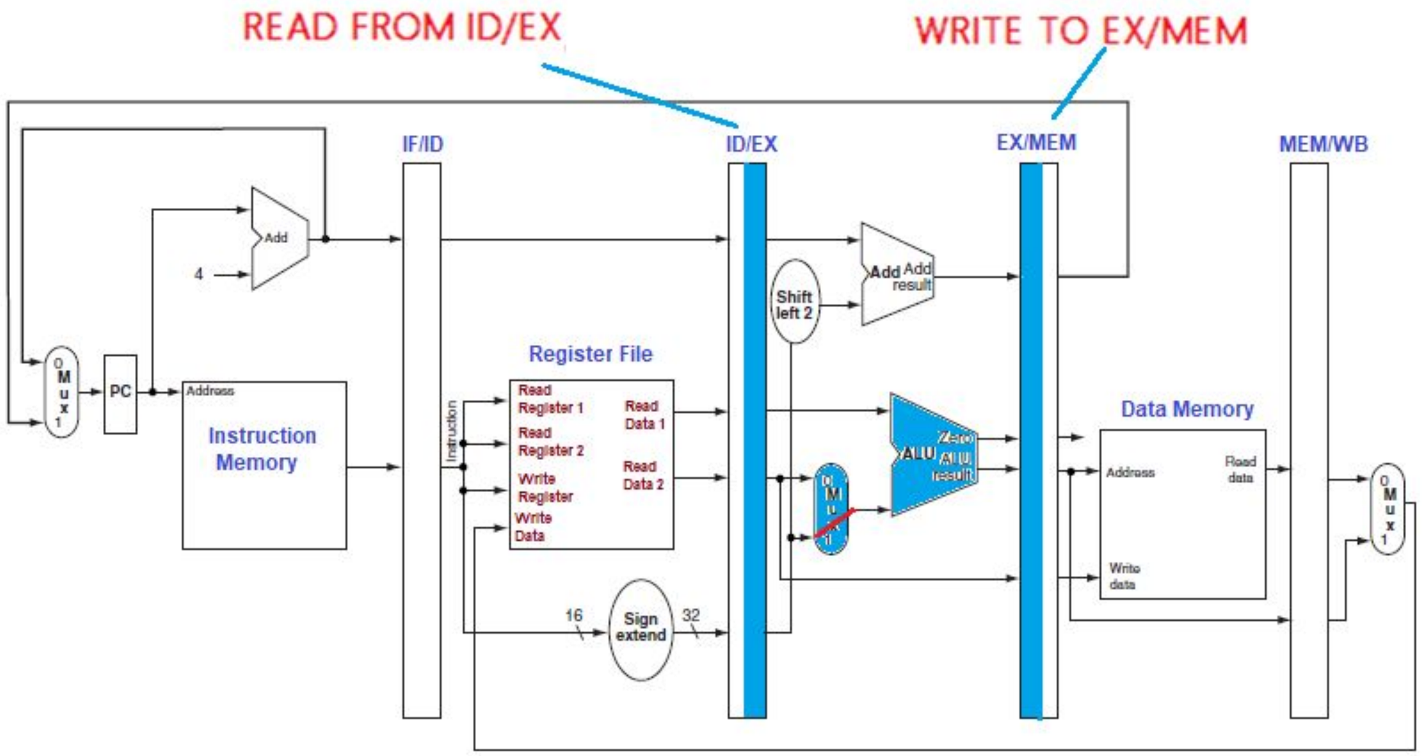
Instruction fetch: The instruction is read from instruction memory (Read to the right, Write to the left- this is the mantra) and write it to the register – IF/ID.

Now, see that the data goes to the register! I read the instruction from the memory and move it to the register.

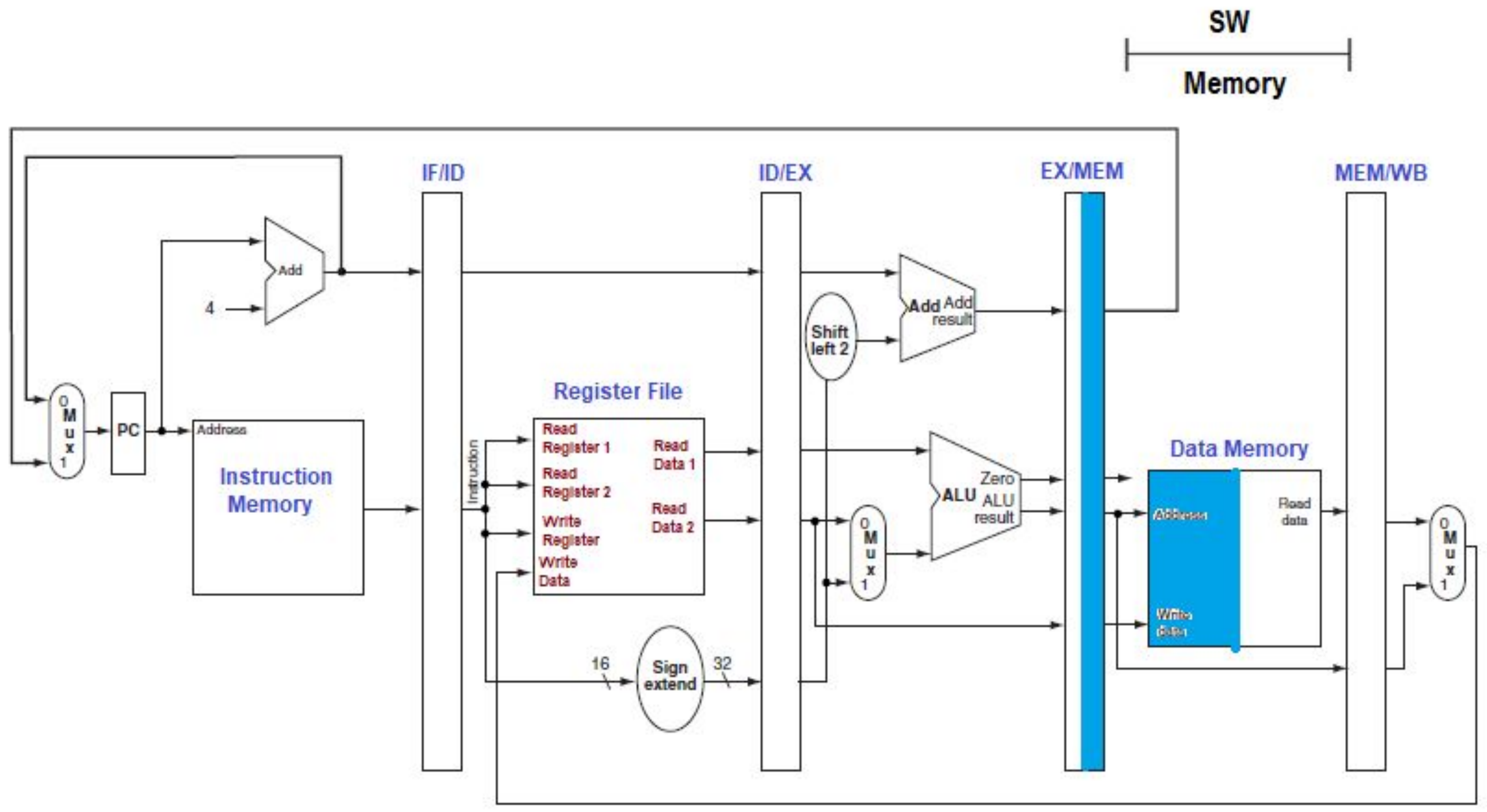
Let’s move to the step 2



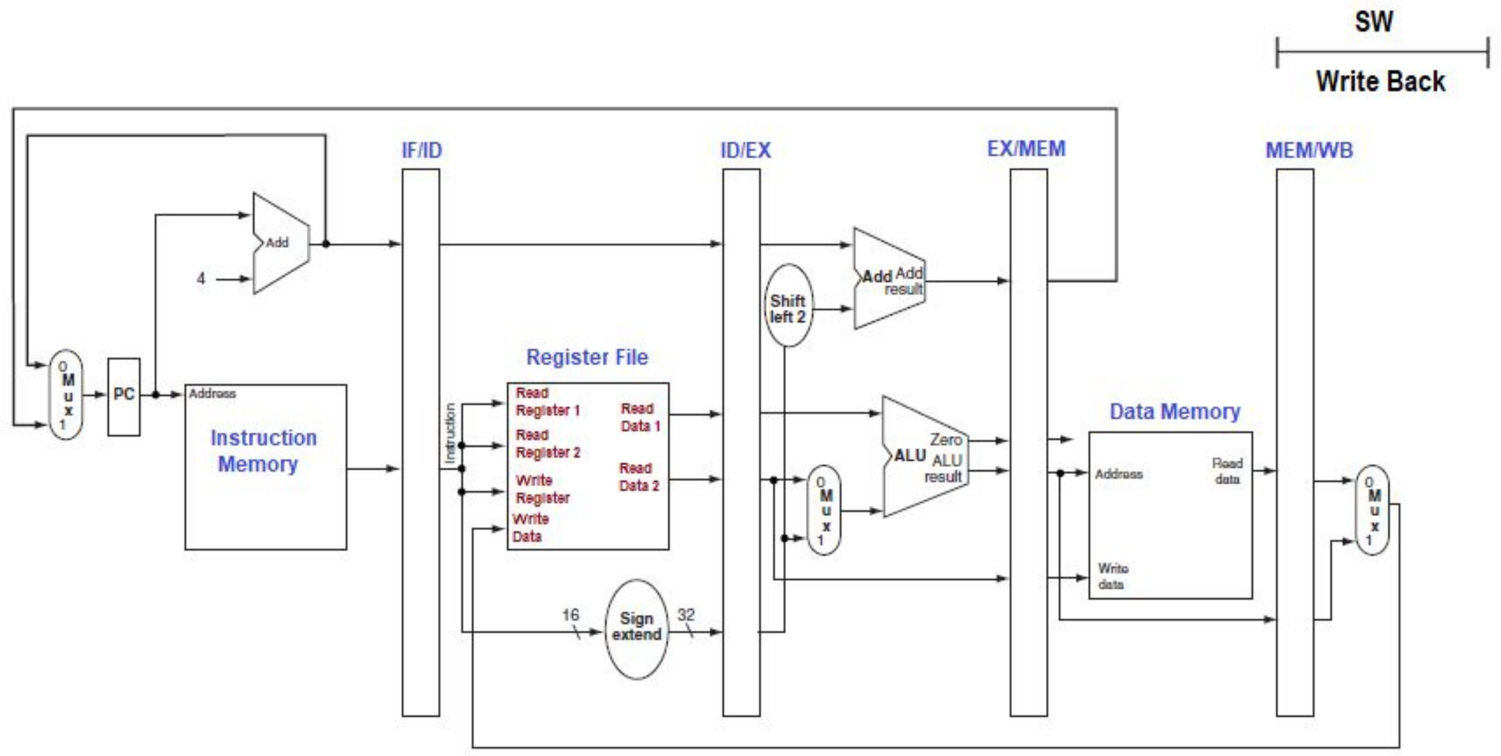
Let us go to step 3



Let us go to step 4



Final Step (See this, nothing is there as there is WB in STORE WORD!)



Up till this is your second periodical portion.

All the best!

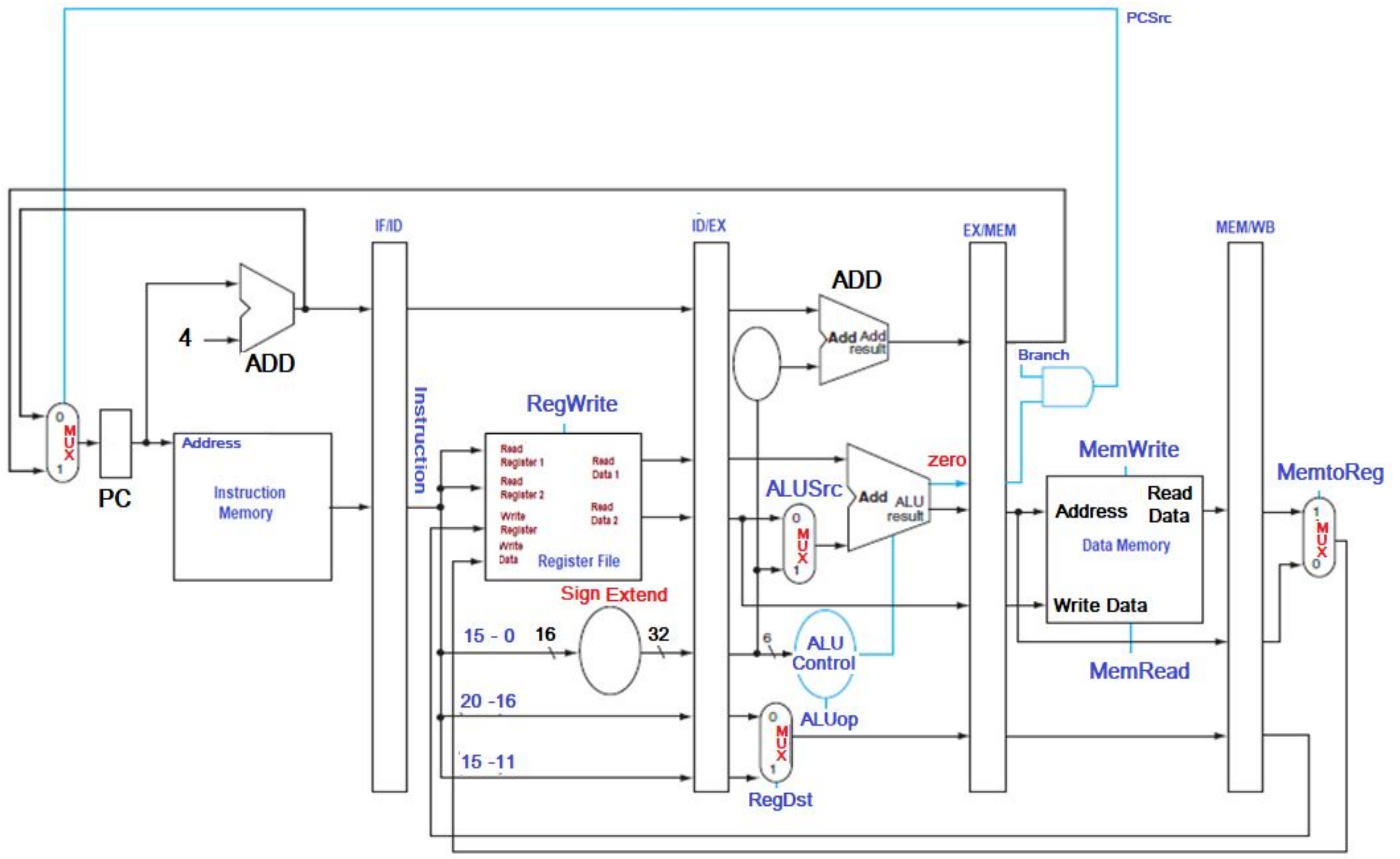
Pipelined Control. (With Control Signals)

Session - 8

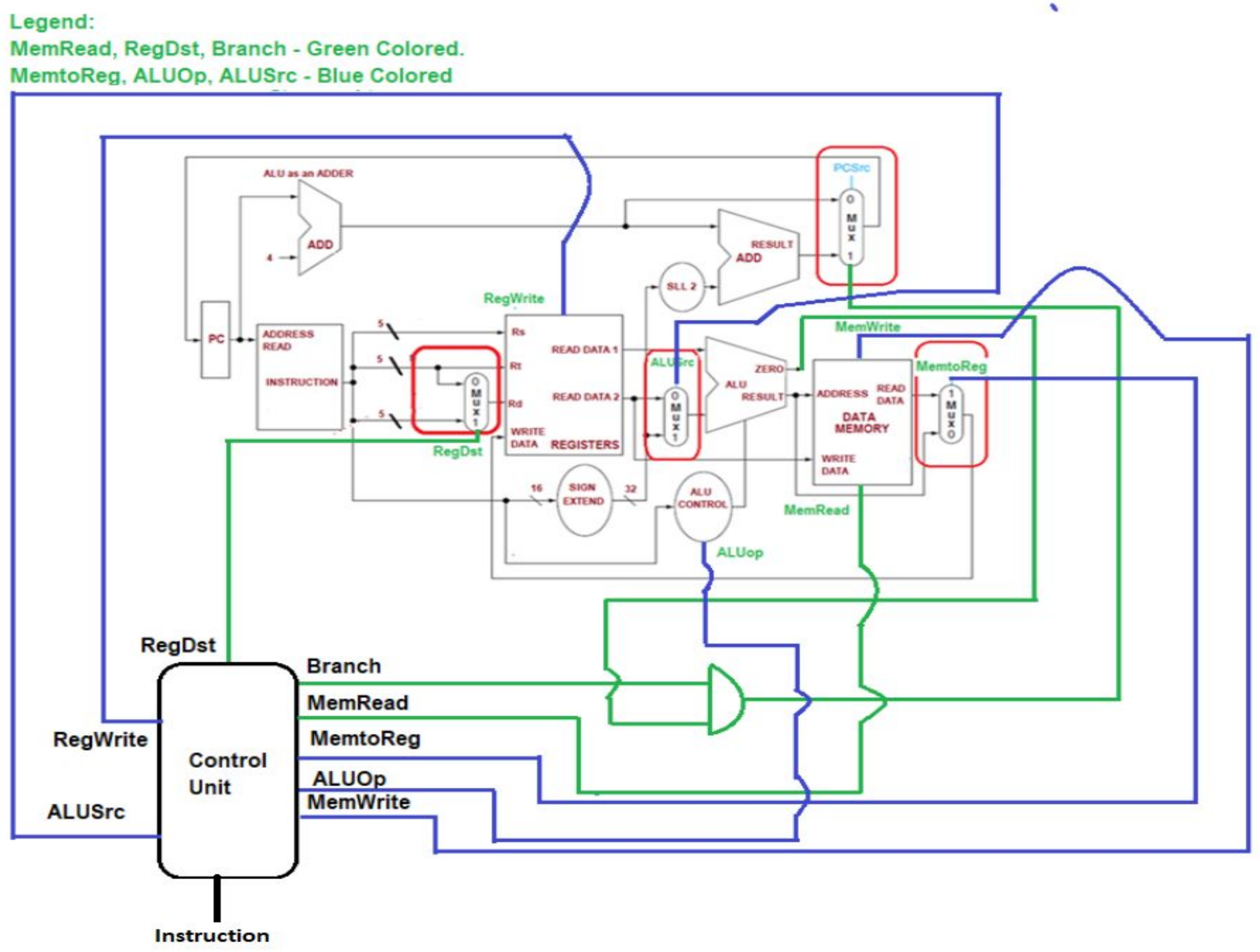
Understand this..

- Can you recollect the way we added the control signals to the data path? – Yes. It helps here.
- The pipelined datapath is presented sometime back to you folks. It is time to add the control signals to the pipleline to make it complete!
- Pipelined datapath + Control Signals = Complete Pipeline.

See this!


PCSrc
IF/ID
ID/EX
EX/MEM
MEM/WB
4
ADD
ADD
Add
Add result
Branch
0
M
U
X
1
PC
Address
Instruction Memory
Instruction
RegWrite
Read Register 1
Read Register 2
Write Register
Write Data
Read Data 1
Read Data 2
Register File
zero
ALUSrc
Add
ALU result
MemWrite
Address
Read Data
Data Memory
Write Data
MemtoReg
Sign Extend
15 - 0
16
32
6
ALU Control
MemRead
20 -16
ALUop
15 -11
RegDst

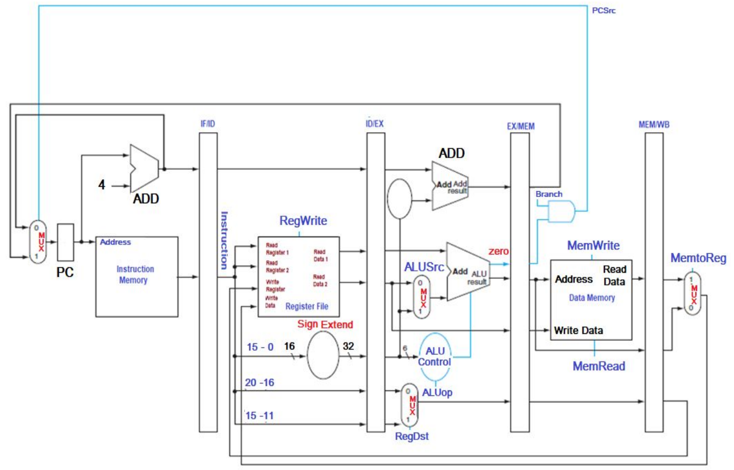
Can we connect this with the previous pipelined control signal diagram? Yes.



Let's make it better.

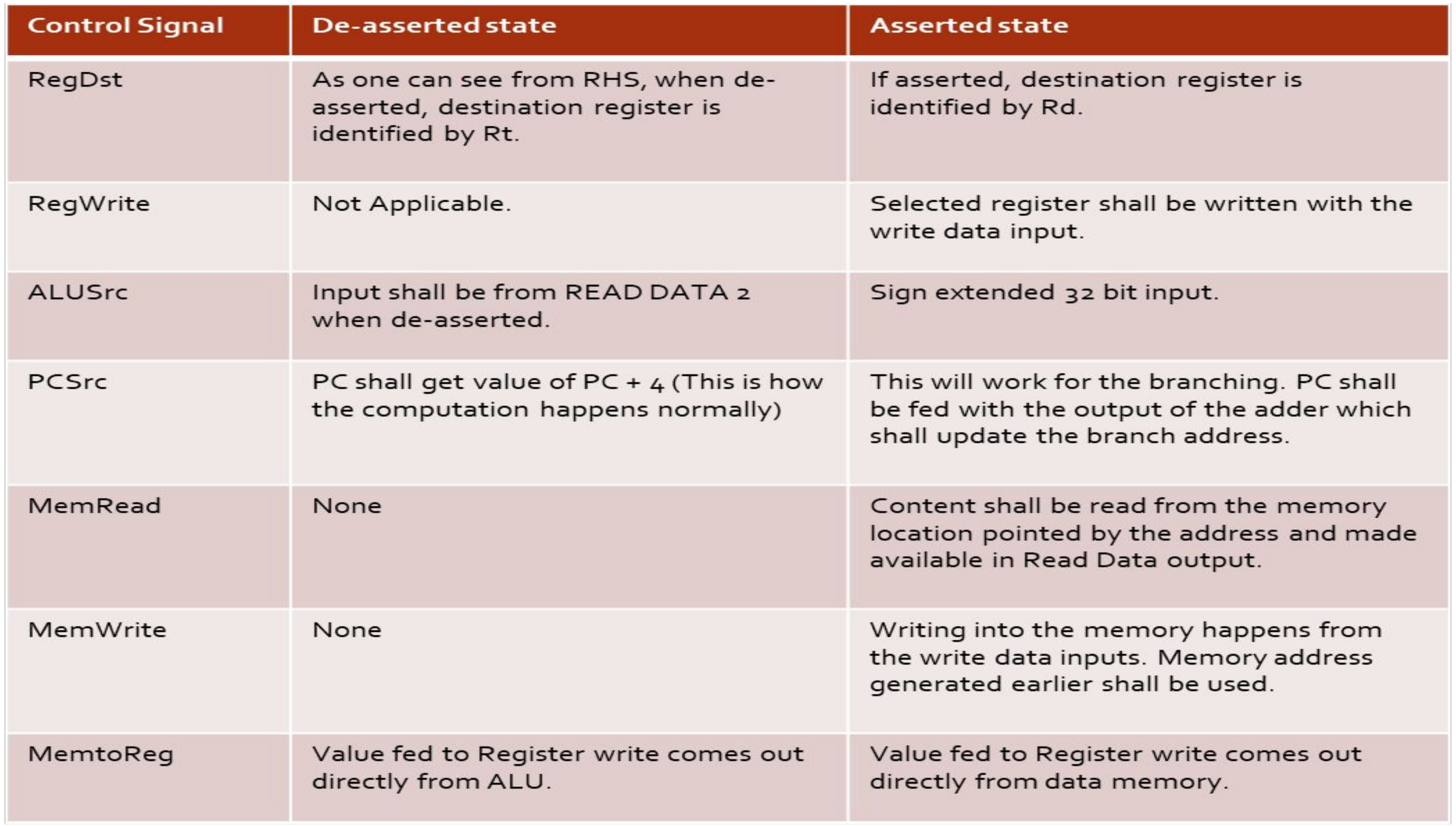
- Most of the control signals we used in the previous comprehensive datapath are similar here. Also, ALU Control, the branching operation, Mux etc. are the one and the same.
- Can we re-iterate something clearer?
- Have we ever used a signal by name "PCWrite" ?? No need as it happens naturally on every cycle.
- Similarly, no write signals are required for **IF/ID, ID/EX, EX/MEM, and MEM/WB pipelined registers. They are also written during each cycle!** 😊

Contd.,



- How many stages are there in the pipeline?
  - Obvious. 5 stages.
  - So, control signals shall be seen with respect to 5 stages.
- **Instruction Fetch:**
  - Related to fetching the instruction.
  - Related to reading the instruction memory and to write to the PC.
  - So, the signals related to this phases are asserted.
- **Instruction Decode/Register File Read:**
  - This is a traditional sequence and No optional control lines to be set.
- **Execution and Address Calculation:**
  - Signals connected in this stage are RegDst, ALUop, ALUSrc.
  - Register Selection, ALU operation shall be taken care here.
- **Memory Access:**
  - Branch, MemRead and MemWrite are the signals connected to this stage.
  - If it is branching instruction, signals shall be pulled out accordingly.
- **Write Back:**
  - MemtoReg and RegWrite are connected.
  - You know the purpose!

Contd.,



Control Signal	De-asserted state	Asserted state
RegDst	As one can see from RHS, when de-asserted, destination register is identified by Rt.	If asserted, destination register is identified by Rd.
RegWrite	Not Applicable.	Selected register shall be written with the write data input.
ALUSrc	Input shall be from READ DATA 2 when de-asserted.	Sign extended 32 bit input.
PCSrc	PC shall get value of PC + 4 (This is how the computation happens normally)	This will work for the branching. PC shall be fed with the output of the adder which shall update the branch address.
MemRead	None	Content shall be read from the memory location pointed by the address and made available in Read Data output.
MemWrite	None	Writing into the memory happens from the write data inputs. Memory address generated earlier shall be used.
MemtoReg	Value fed to Register write comes out directly from ALU.	Value fed to Register write comes out directly from data memory.

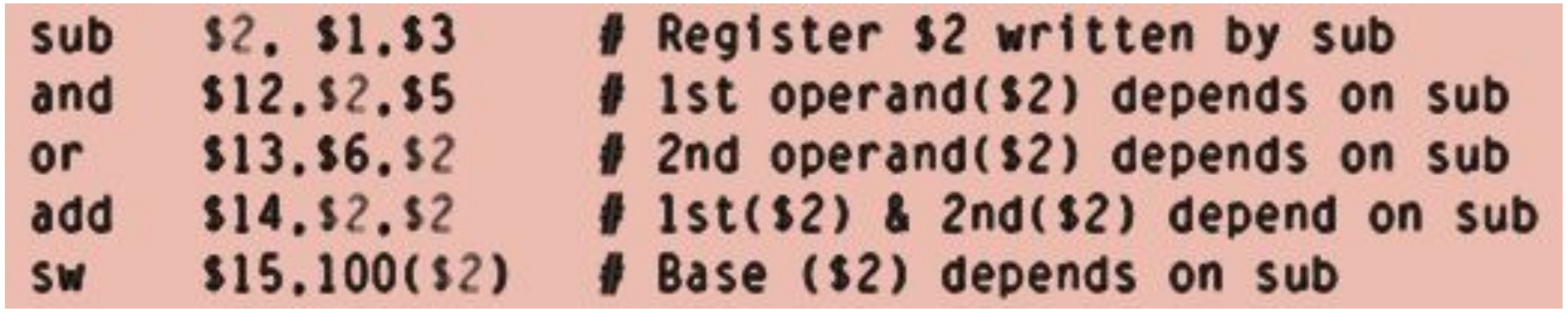
Let us learn better.

Shriram K Vasudevan

Let us start it again! 😊 Session – 9

Data Hazards and Forwarding – We got to learn this!

A Scenario

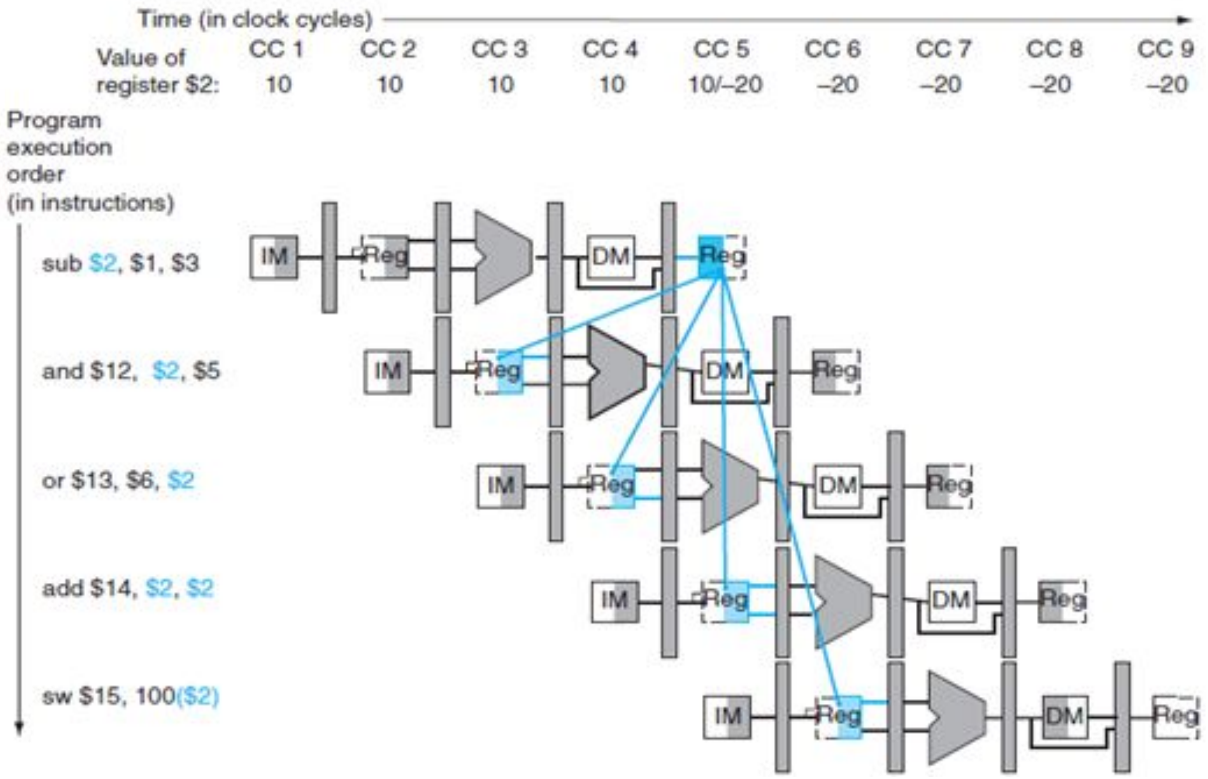


```
sub    $2, $1,$3      # Register $2 written by sub
and    $12,$2,$5      # 1st operand($2) depends on sub
or     $13,$6,$2      # 2nd operand($2) depends on sub
add    $14,$2,$2      # 1st($2) & 2nd($2) depend on sub
sw     $15,100($2)    # Base ($2) depends on sub
```

- **Now, let us see the above instruction sequence. First instruction is the boss and rest of the instructions are dependent on the first.**
  - **See $2 is there all the instructions.**
  - **There is the catch, $2 should have the correct value for the rest of the operations to be carried out correct. ☺ Makes sense, huh?**
  - **If $2 has 5 initially and after sub, if it gets 2 it should be reflected properly. (Your programing logic would depend on the 2 which is the correct result)**

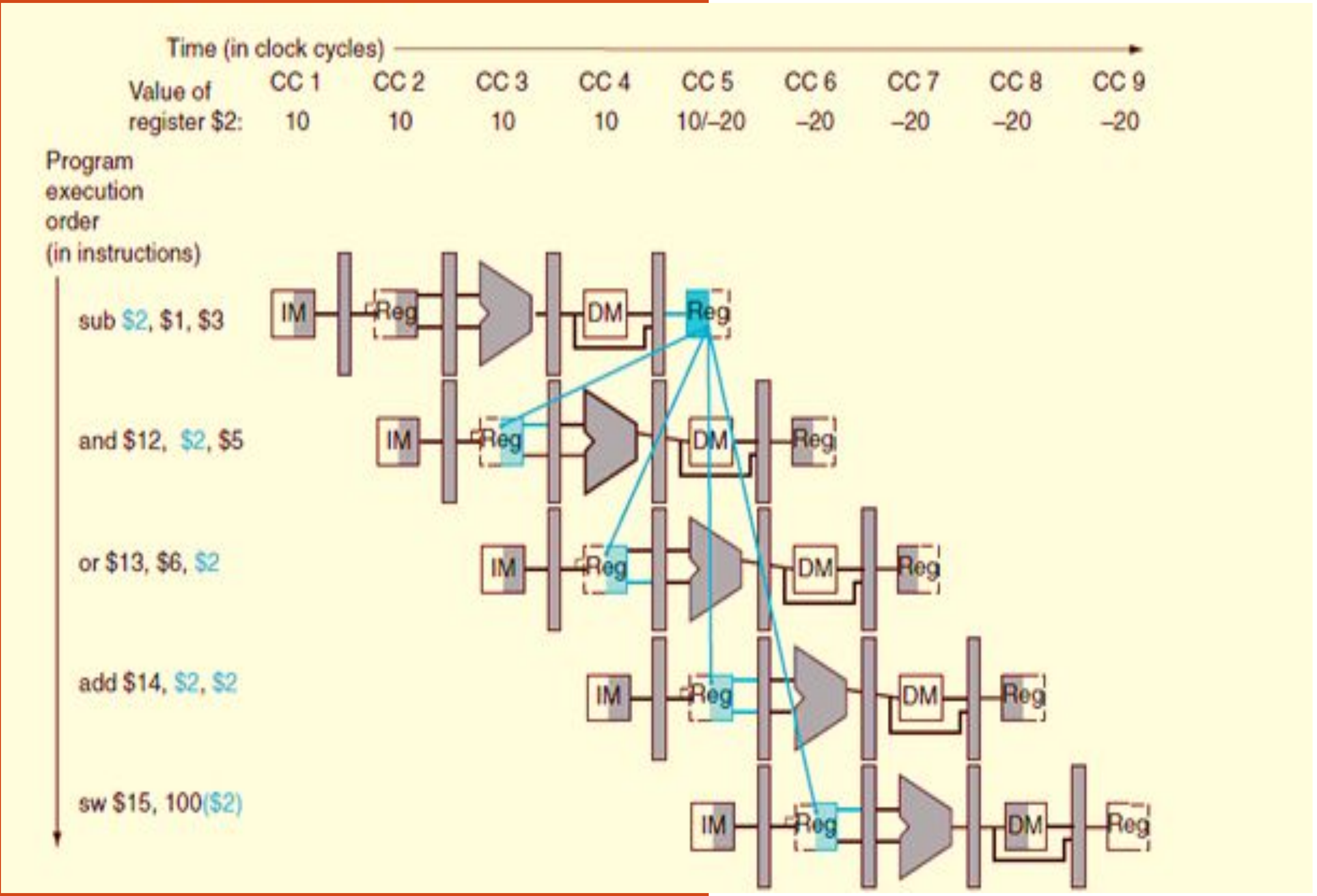
With pipelining one should see the result.

- **How would this sequence perform with our pipeline?**
- One can visualize the example. $2 has some value, say 10.
- This value changes only at the CC 5 , that too in the middle of the cycle. The result will be written.



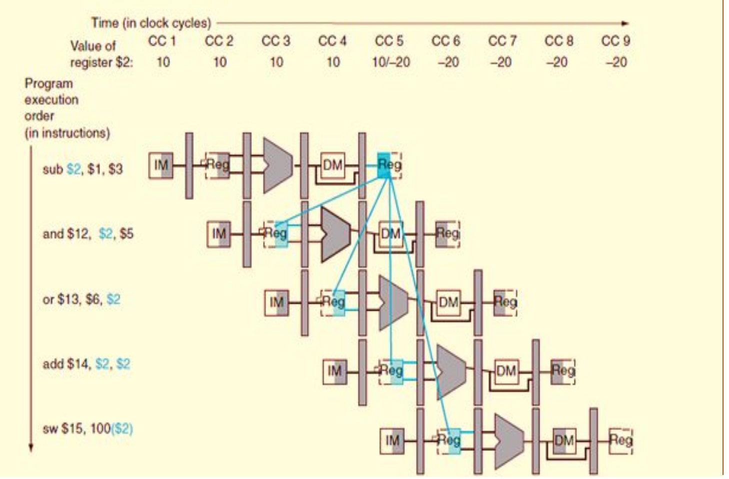
Contd.,

- So, the updated value may not be available at the right time for the right instruction which is seen as a hazard. (Remember the stories)
- Let us ask a question now:
- Will there be a problem if a register is read & written in the same clock cycle?
  - Not a tough question. Very trivial, indeed.
- This is why we say, Write in the first half of the clock cycle (Write in the left) and read happens in the right (Read in the right). I mean, write to read!
- This approach has eliminated most of the hazards in the pipelining, Now you understand why do we tell write to the left and read from the right???? 😊
  - Million dollar question answered.



- Well, is everything solved? Nope!
- We have more challenges on the way!
- Where do we get the correct result? It is during the 5th cycle (midway). So, correct answer shall be available during that time.
- This implies a straight forward fact that only the 4th and 5th instructions can get the correct value (Remember, we can't travel backward in time)
- The precise value of -20 shall be made available to add and sw.
- *This directly conveys the fact that "2nd and 3rd instruction would still receive the old -10 as the result.*
- *Sir, why do you say this? Simple. When you draw this way, this problems are obvious and remember the time travel related hint I gave you.*

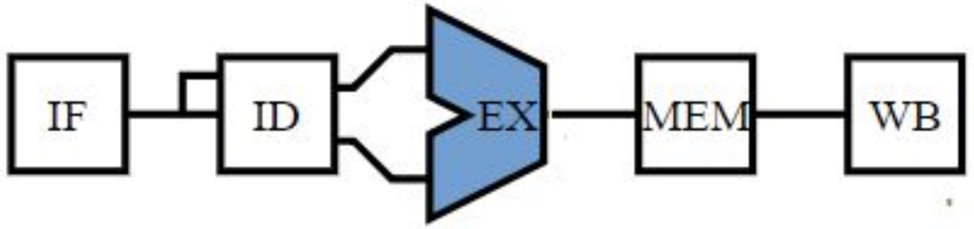
So, what is the solution?



- Can you recollect the whole old stories, I conveyed?
- The first instruction has the result by end of the EX stage (I Mean, ALU shall give what you ask for). This really happens in the stage i.e. CC3.
- When the data is needed by 2nd and 3rd instruction?
  - You can see that the requirement is visible in CC4 for AND and CC5 for OR. Both are at the EX stage.
  - **Hence, no special effort required here.**
  - **No stalling is required.**
  - **Forward the data as soon as it is available to be sent**
  - **This would help resolving the challenging.**

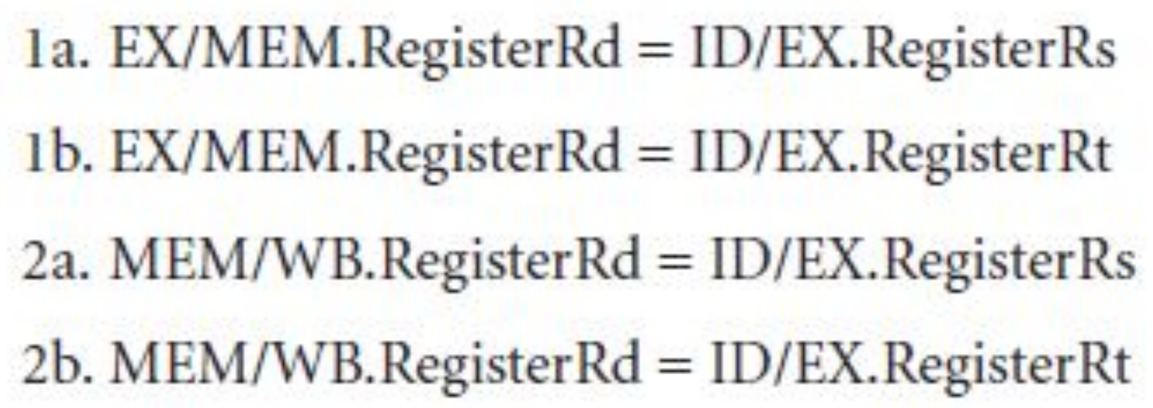
Let us get deeper with forwarding . . .

- Do we have any notation with forwarding?
- Simple examples shall help the understanding clearer.
- ID/EX.RegisterRs - Reveals the fact that "The value which Rs needs is available at the pipeline register ID/EX"



Contd.,

- We shall make it better here!
- ID/EX.RegisterRs
  - To the left of the DOT (.) – Name of the pipeline register is present.
  - To the right, the register is represented.
- Shall we make the design rules for the hazard conditions now?
- Using this notation, the two pairs of hazard conditions are to be framed.

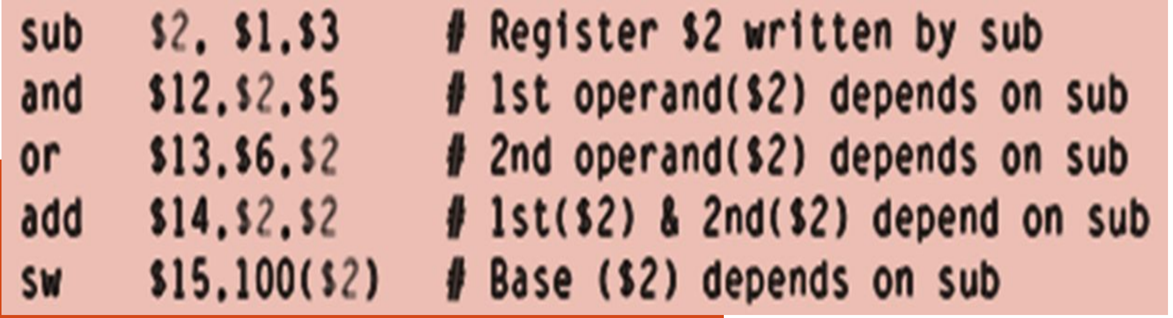


1a. EX/MEM.RegisterRd = ID/EX.RegisterRs

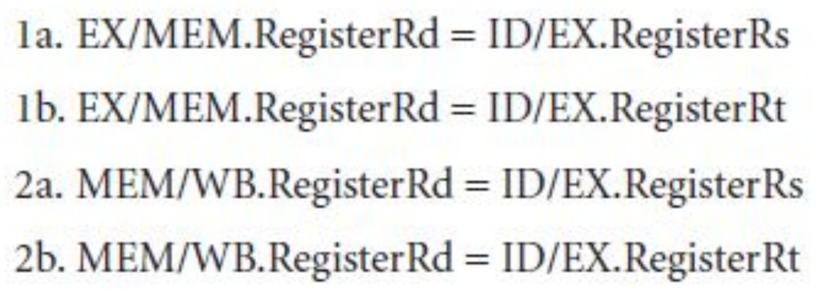
1b. EX/MEM.RegisterRd = ID/EX.RegisterRt

2a. MEM/WB.RegisterRd = ID/EX.RegisterRs

2b. MEM/WB.RegisterRd = ID/EX.RegisterRt



```
sub   $2, $1,$3      # Register $2 written by sub
and   $12,$2,$5      # 1st operand($2) depends on sub
or    $13,$6,$2      # 2nd operand($2) depends on sub
add   $14,$2,$2      # 1st($2) & 2nd($2) depend on sub
sw    $15,100($2)    # Base ($2) depends on sub
```



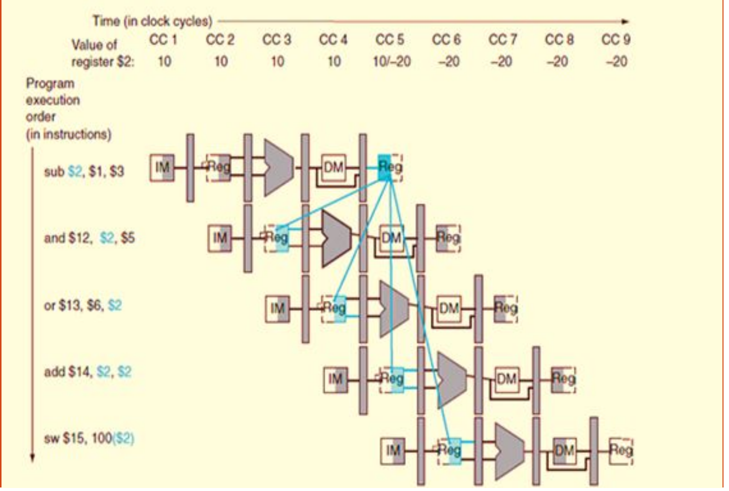
1a. EX/MEM.RegisterRd = ID/EX.RegisterRs

1b. EX/MEM.RegisterRd = ID/EX.RegisterRt

2a. MEM/WB.RegisterRd = ID/EX.RegisterRs

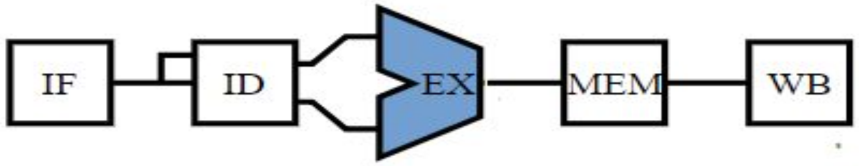
2b. MEM/WB.RegisterRd = ID/EX.RegisterRt

Contd.,



- Let us take the sequence shown above and classify the hazard type.
- **SUB $2, $1, $3 --- Result is written in the $2. (Remember this)**
- **AND $12, $2, $5 --- The previous line's result goes here. Hence, this hazard is to be classified as one of the above 4.**
- **The hazard will be found wen the AND is in the EXECUTION STAGE whereas by then, the previous instruction SUB would have gone to MEM stage (Logical – Right??)**
- **So we can classify the hazard as**

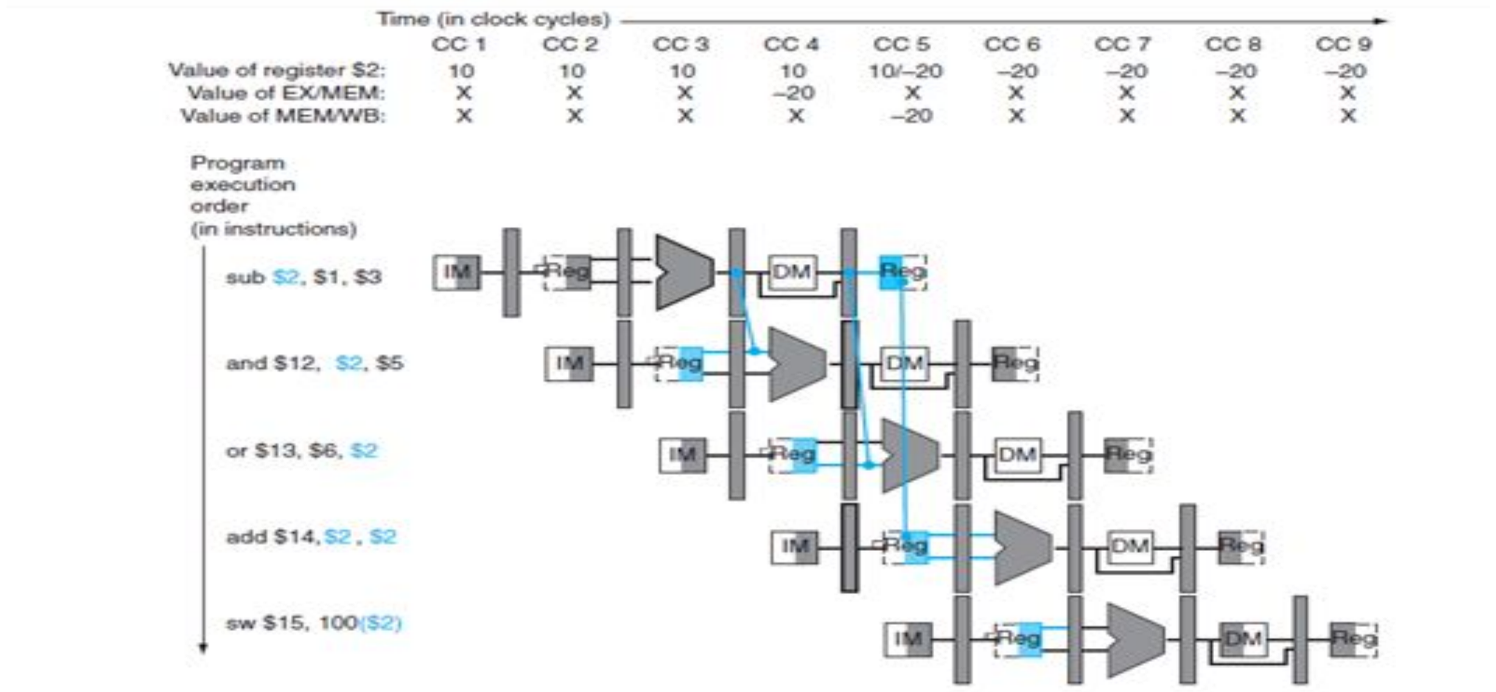
EX/MEM.RegisterRd = ID/EX.RegisterRs = $2 (1a)



ID/EX.RegisterRs - Reveals the fact that "The value which Rs needs is available at the pipeline register ID/EX"

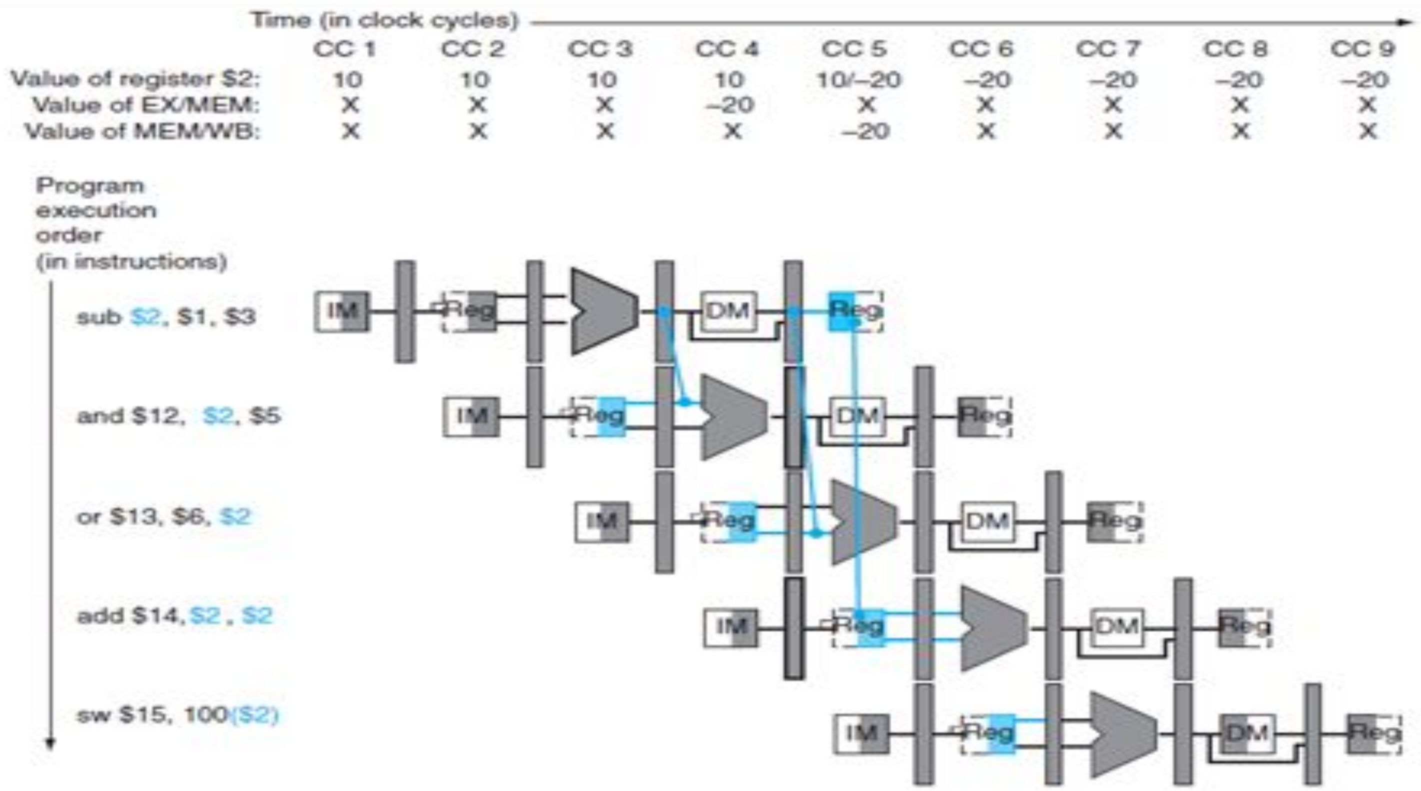
Contd.,

- We have detected the hazard! But, It is job half done. We still have to forward the correct data! Here we go about it!
- We have a point. See the below figure, carefully. It highlights the dependencies between the Pipeline registers and the ALU
- Observe that the dependence starts with the pipeline register and not the WB stage.
- Thus the required data exists in time for later instructions, with the pipeline registers holding the data to be forwarded



Contd.,

- So,
- We shall think this way! Can we feed the ALU from any pipeline register? Why should it be from ID/EX? It will be more appropriate and proper when it is done that way.
- Wait. Is this straight forward? No. Add multiplexors to the ALU inputs. Of course, the control signals are to be embedded. Pipeline shall be at its full speed if this can be done!



Time (in clock cycles)

	CC 1	CC 2	CC 3	CC 4	CC 5	CC 6	CC 7	CC 8	CC 9
Value of register $2:	10	10	10	10	10/–20	–20	–20	–20	–20
Value of EX/MEM:	X	X	X	–20	X	X	X	X	X
Value of MEM/WB:	X	X	X	X	–20	X	X	X	X

1a. EX/MEM.RegisterRd = ID/EX.RegisterRs

1b. EX/MEM.RegisterRd = ID/EX.RegisterRt

2a. MEM/WB.RegisterRd = ID/EX.RegisterRs

2b. MEM/WB.RegisterRd = ID/EX.RegisterRt

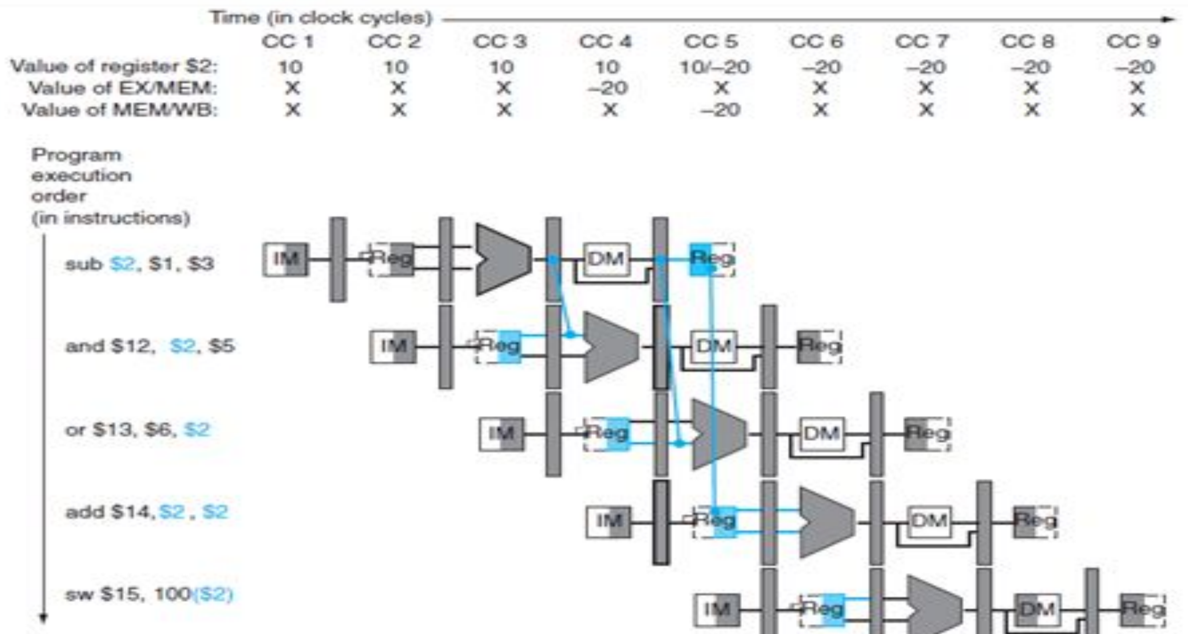
Contd.,

```
sub $2, $1, $3        # Register $2 is updated by sub.
and $12, $2, $5       # 1st operand - $2 is the dependency
or $13, $6, $2        # 2nd operand - $2 is the dependency
add $14, $2, $2       #  Both the operands are having dependency
sw $15, 100 ($2)      #  Here, the index has the dependency
```

1. The sub and hazard is discussed. EX/MEM.RegisterRd = ID/EX.RegisterRs = $2 (1a) (Remember the rules)

2. sub - or hazard now would be classified as - MEM/WB.RegisterRd = ID/EX.RegisterRt = $2 (2b)

Rest all are not hazards! We can understand that through the diagram.



Time (in clock cycles)
CC 1 CC 2 CC 3 CC 4 CC 5 CC 6 CC 7 CC 8 CC 9
Value of register $2: 10 10 10 10 10/–20 –20 –20 –20 –20
Value of EX/MEM: X X X –20 X X X X X
Value of MEM/WB: X X X X –20 X X X X
Program execution order (in instructions)
sub $2, $1, $3
and $12, $2, $5
or $13, $6, $2
add $14, $2 , $2
sw $15, 100($2)
IM
Reg
DM

We need to see more. Session – 10

Shriram K Vasudevan