**ILP (Instruction Level Parallelism):**

ILP <=> IPC when

1)processor does entire instruction in 1 cycle

2)processor can do any number of instructions in same cycle (has to obey true dependencies)

**Steps to get ILP:**

1)register renaming

2) ”Execute”

***🡪ILP is a property of a program NOT of a processor.***

All instructions go through all the pipeline stages in the same cycle such that if we have a 5-stage pipeline, all the instructions will complete & exit the pipeline in 5 cycles.

CPI = 5 cycles/infinite instr = 0 (ideal value)

While this sounds ideal & perfect, but the subtle problem here is if every instruction in the pipeline starts executing in the same cycle, dependent instructions will start executing will stale result values resulting in incorrect execution of dependent instructions.

To work around that problem, we have identified that the problem occurs when multiple instructions start executing in the same cycle. So, in this case, we could either stall & forward the result to the dependent instruction in the next cycle which hurts the CPI a little bit but not significantly. But when this process has to happen for multiple dependent instructions in the pipeline, the CPI takes a significant hit.

WAR & WAW are false/name dependencies.

To remove false dependencies, duplicating register values could be used (not scalable).

**One more method to remove false dependencies is register renaming.**

**Register Renaming:**

**1)Archiectural registers:-** registers which the programmer or compiler use (mostly probably ISA defines them and they become visible for assembly lang programmer)

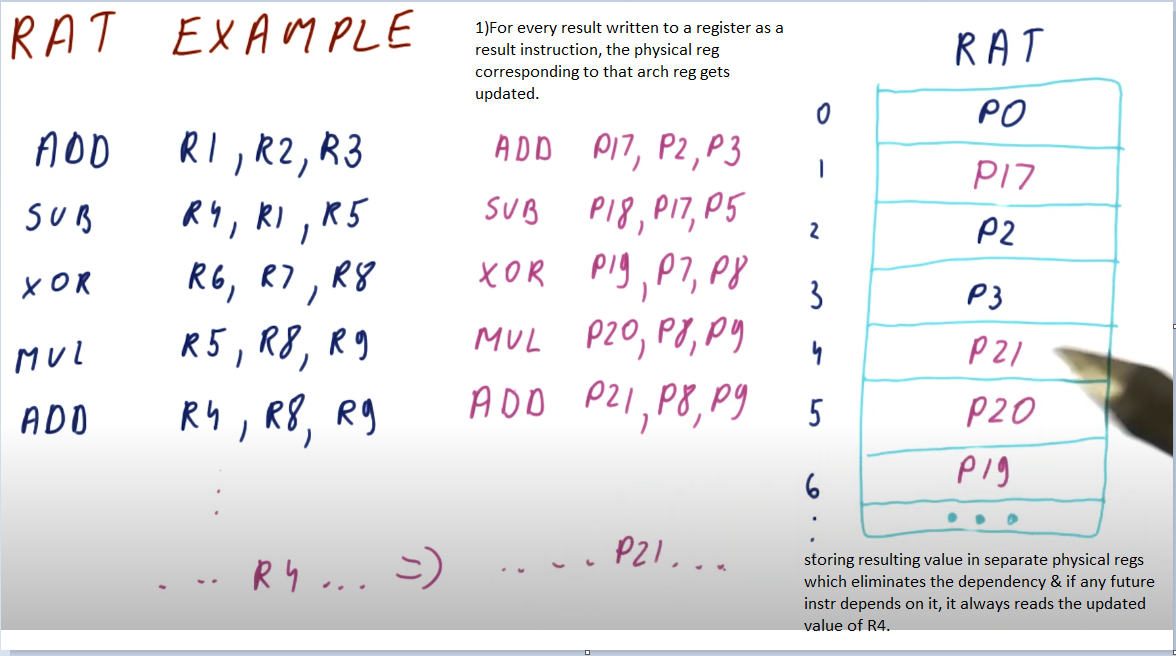
ADD R1,R2,R3 (Here R1,R2 & R3 are arch regs defined by ISA & visible to assembly lang programmer)

**2)Physical registers:-** All the places where one can put value in the processor.

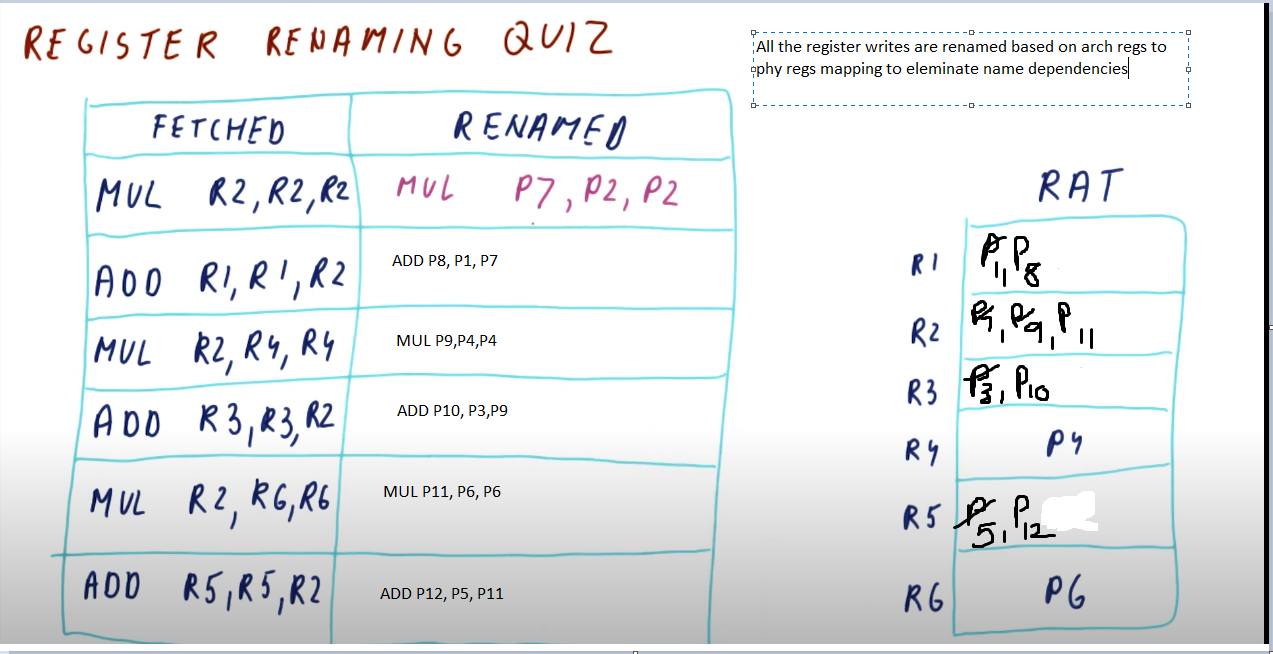
🡪Register renaming means to rewrite the program to use physical registers.

🡪Register Allocation Table(RAT)=> table that says which physical register has value for which arch register.

Ex.: PTO:



**Register Renaming Quiz-1:**



**Question-2:**

**2)** Assume that the processor has 6 logical registers (R1, R2, …, R6) and 12 physical registers (P1, P2, ….., P12). Before the above program fragment is executed, the state of the renaming table is as follows:

|  |  |
| --- | --- |
| Logical Register | Physical Register |
| R6 | P6 |
| R4 | P5 |
| R5 | P8 |
| R2 | P11 |
| R3 | P9 |
| R1 | P2 |

Note that physical registers P1, P3, P4, P7, P10 and P12 are free before the code sequence is executed. Rewrite the above code fragment using register renaming to remove output and anti-dependences. Your new code sequence should substitute all the logical register renames with the appropriately renamed physical register names.

Notes: the key is when doing a register write, if that register was used previously, in order to avoid RAW, WAR, WAW hazards, need to rename that register and replace it with a register from the free physical register list. When doing a register read, just use the Logical to Physical Register Table provided to do the conversion. This is to avoid data hazards.

Load 🡪register write

Store 🡪register read

Program sequence: **Renamed Sequence:**

I1: STORE R6, #20(R4) STORE P6, #20(P5)

I2: ADD R4, R4, R5 ADD P1, P5, P8 (register write R4=>P1)

I3: LOAD R2, #20(R4) LOAD P3, #20(P1) (register write R2=>P3)

I4: SUB R3, R6, R2 SUB P4, P6, P3 (register write R3 => P4)

I5: BEQZ R2, I7 BEQZ P3, I7

I6: LOAD R2, #10(R4) LOAD P7, #10(P1) (register write R2 => P7)

I7: STORE R1, #10(R4) STORE P2, #10(P1)