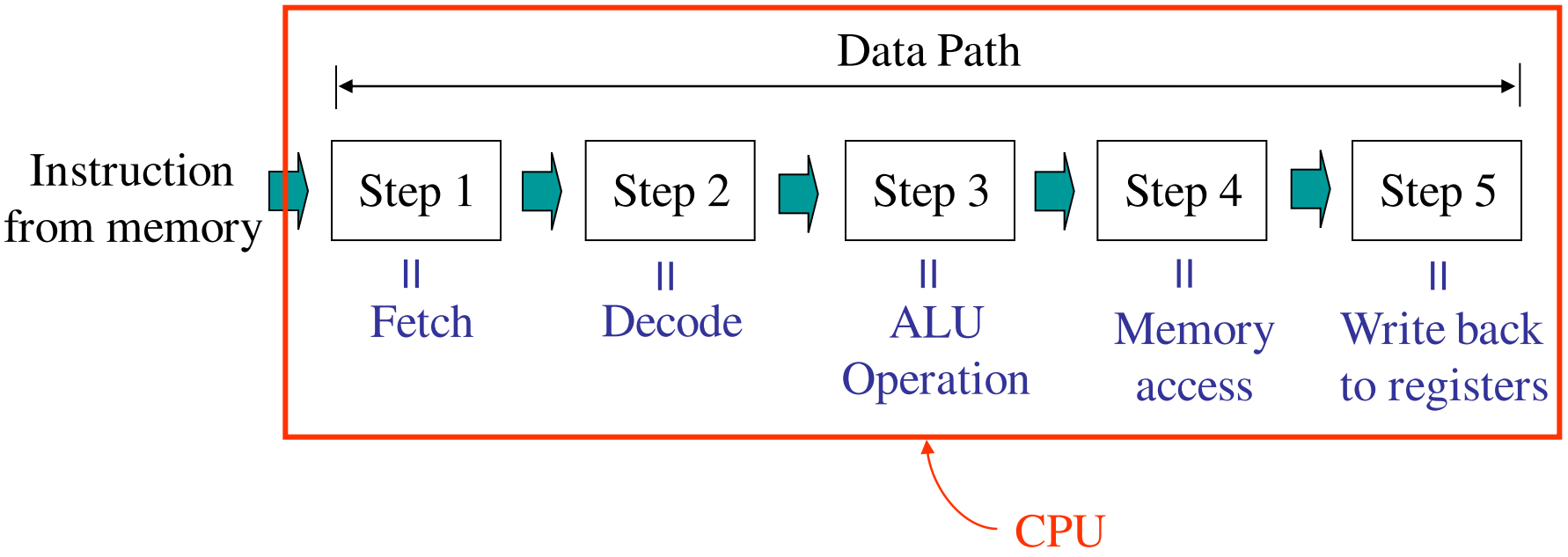
# The way instructions are being executed inside a processor

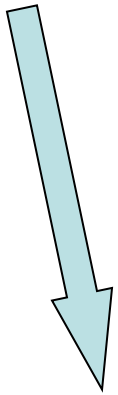
Question: how machine instructions are executed in a processor?

Answer: There are five steps to execute a machine instruction.



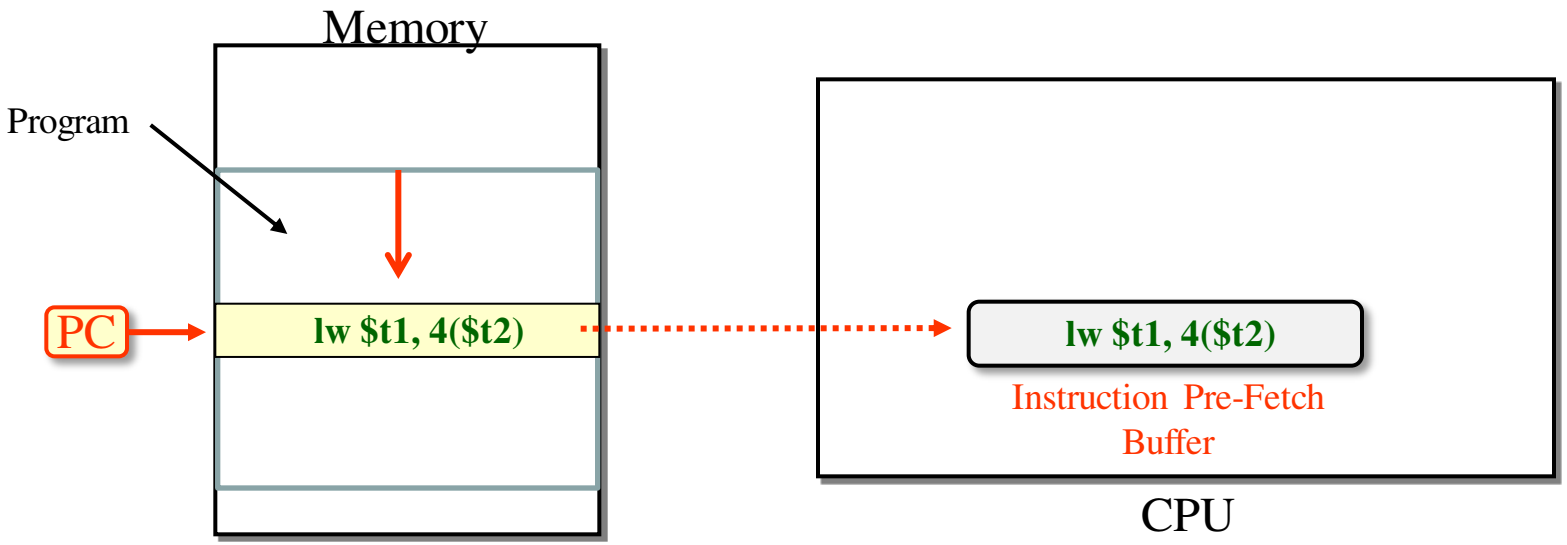

Datapath: a sequence of CPU circuits that execute CPU instructions step by step

Example: Data path for the “load” instruction



```
lb  $t1, 0($t2)      # Load one byte from the memory pointed by t2 register
```

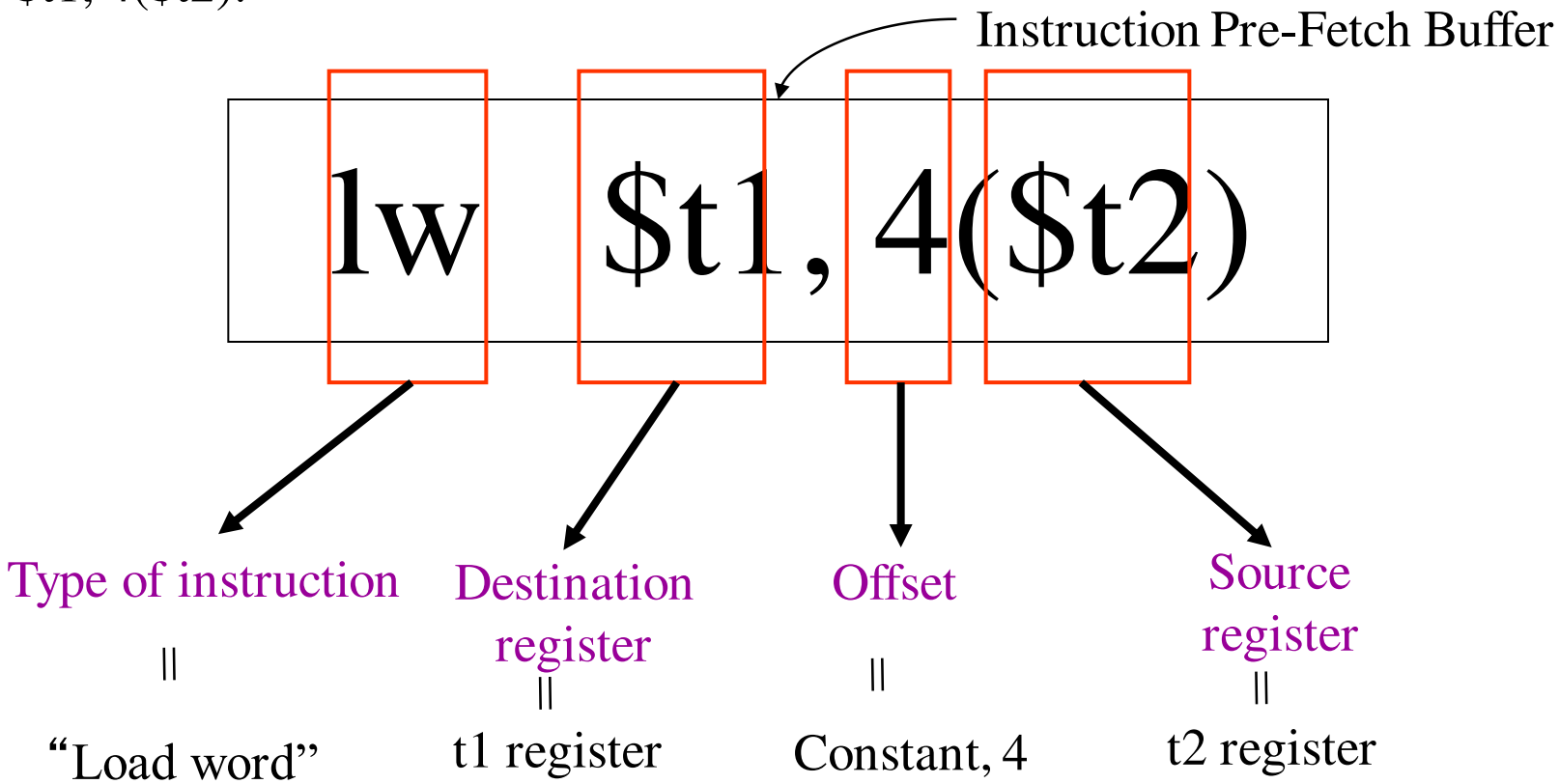
## Step 1: Instruction Fetch (IF)




**Instruction Fetch (IF):**
**Send out the PC and fetch the instruction from memory into the instruction register (IR); increment the PC (by 4) to address the next sequential instruction.**
**IR holds the instruction that will be used in the next stage.**

## Step 2: Instruction Decoding (ID)

lw $t1, 4($t2):



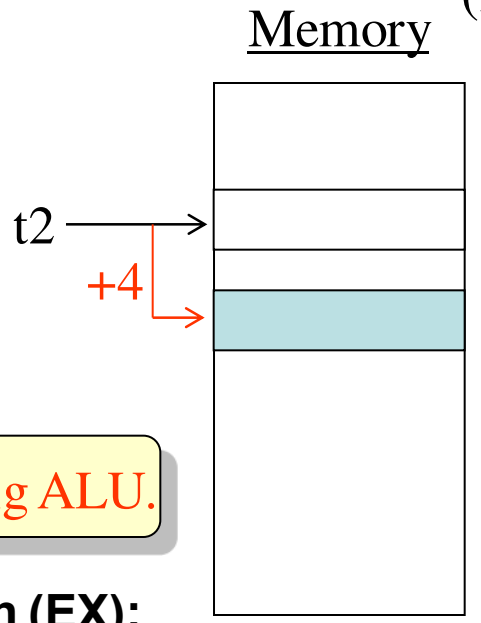

**Instruction Decode/Register Fetch Cycle (ID):**
**Decode the instruction and access the register file to read the registers.**
**The outputs of the general purpose registers are read into two temporary registers (A & B) for use in later clock cycles.**
**We extend the sign of the lower 16 bits of the Instruction Register.**

## Step 3: ALU Operation (EX)

lw $t1, 4($t2):

OPERATIONS

(1) Memory pointed by ***t2***
(2) +4 bytes offset

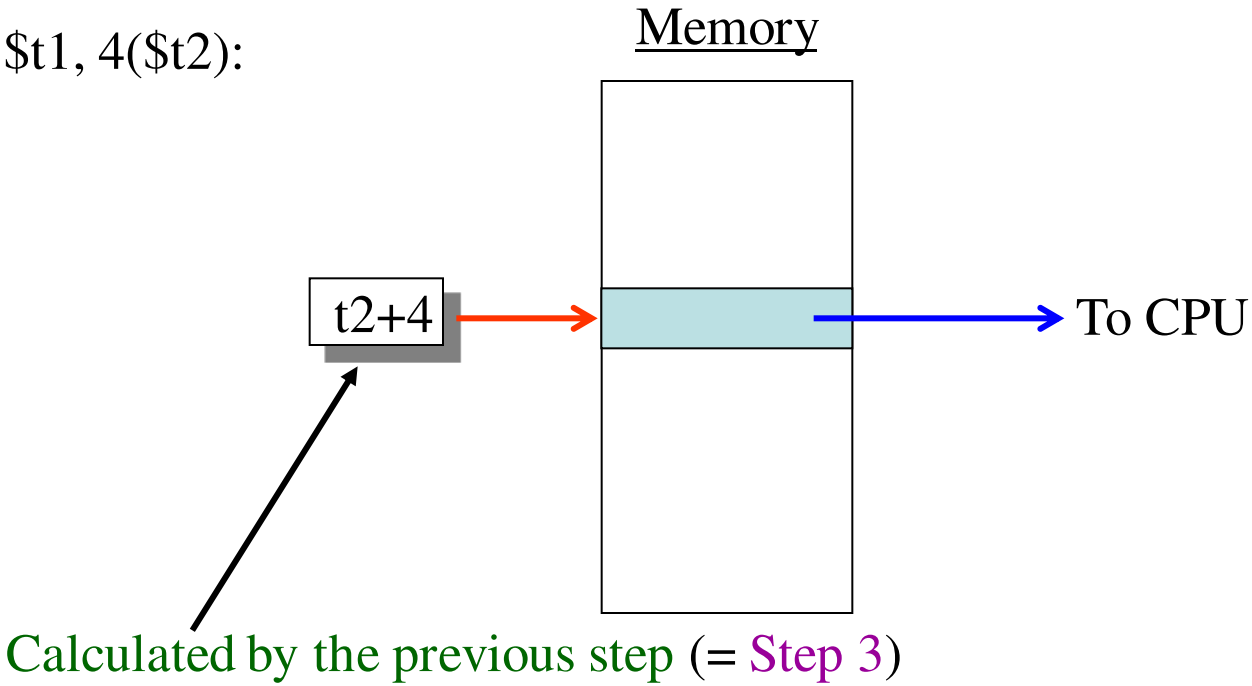



We need to calculate t2 + 4 using ALU.

**Execute Address Calculation (EX):**
**We perform an operation (for an ALU) or an address calculation (if it's a load or a Branch).**
**If an ALU, actually do the operation. If an address calculation, figure out how to obtain the address and stash away the location of that address for the next cycle.**

## Step 4: Memory Access (ME)

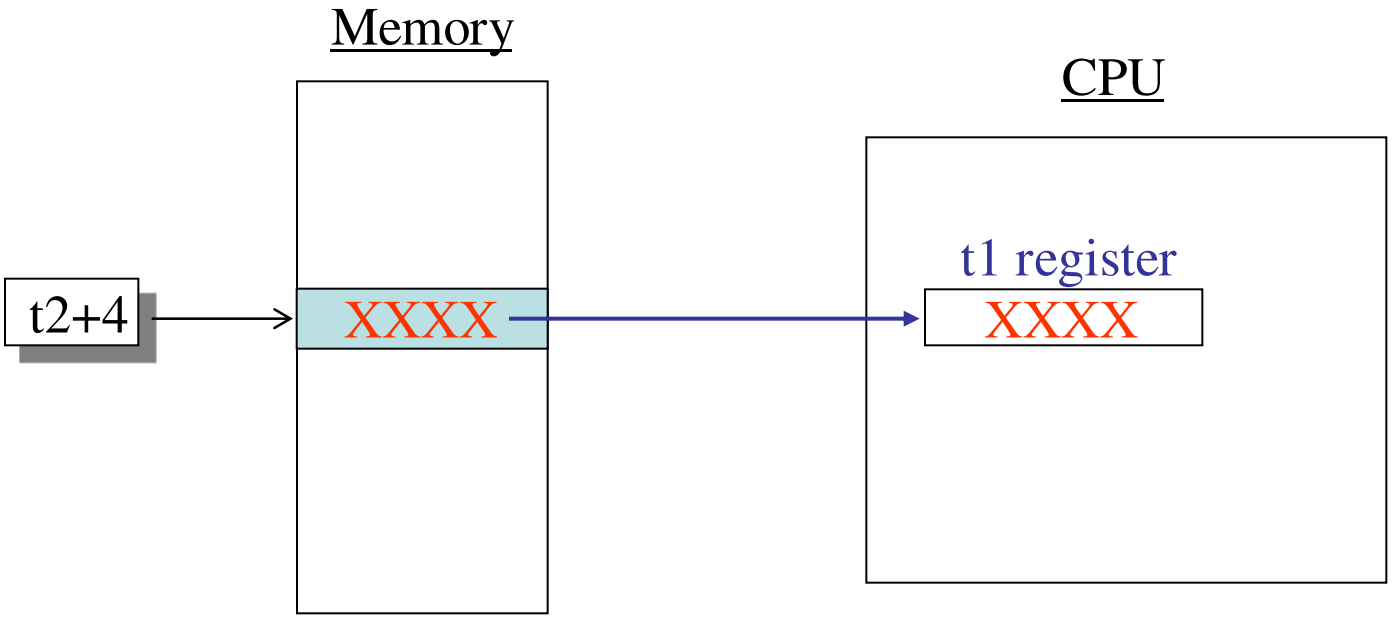
lw $t1, 4($t2):




**MEMORY ACCESS (MEM):**
**If this is an ALU, do nothing.**
**If a load or store, then access memory.**

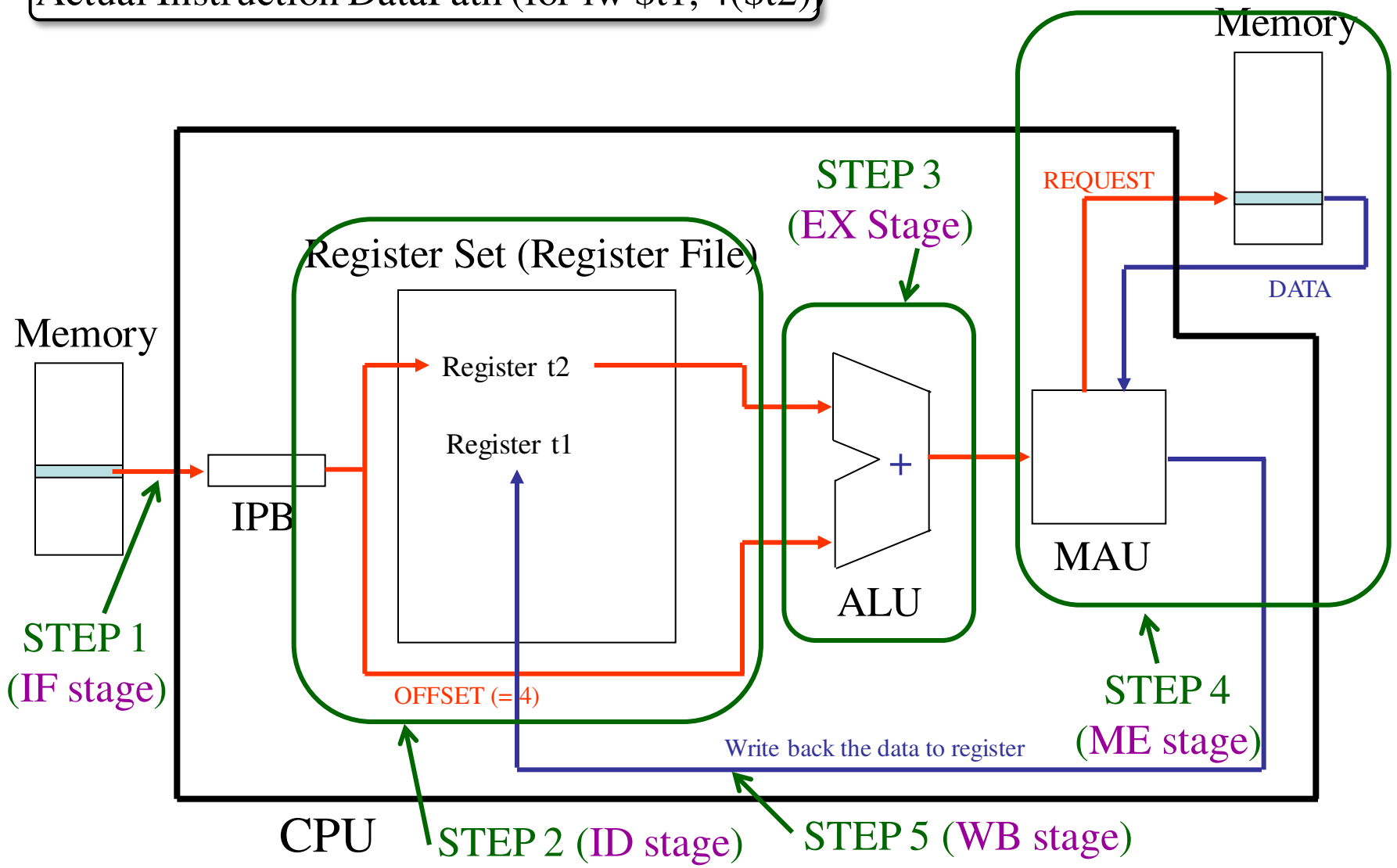
## Step 5: Output to the register (WB)

lw $t1, 4($t2):

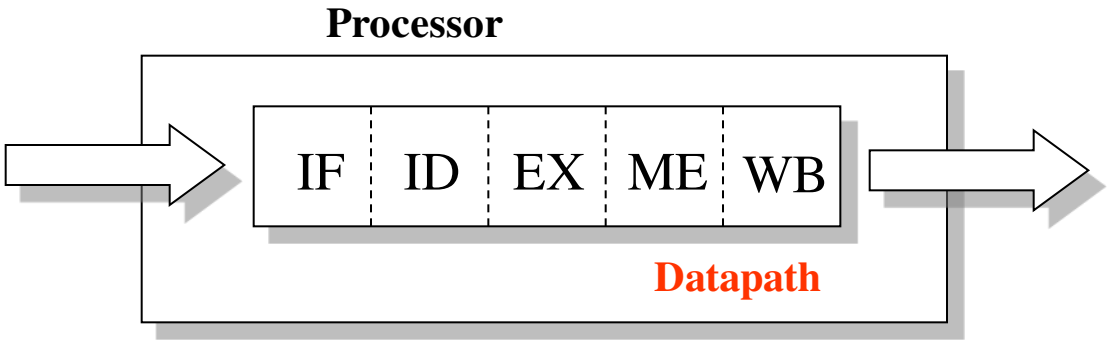



**WRITE BACK (WB):**
**Update the registers from either the ALU or from the data loaded.**

# Actual Instruction DataPath (for lw $t1, 4($t2))



- Datapath executes processor instruction
- The essential datapath phases are: IF, ID, EX, ME, and WB
- Each datapath phase requires a processor one processor clock cycle

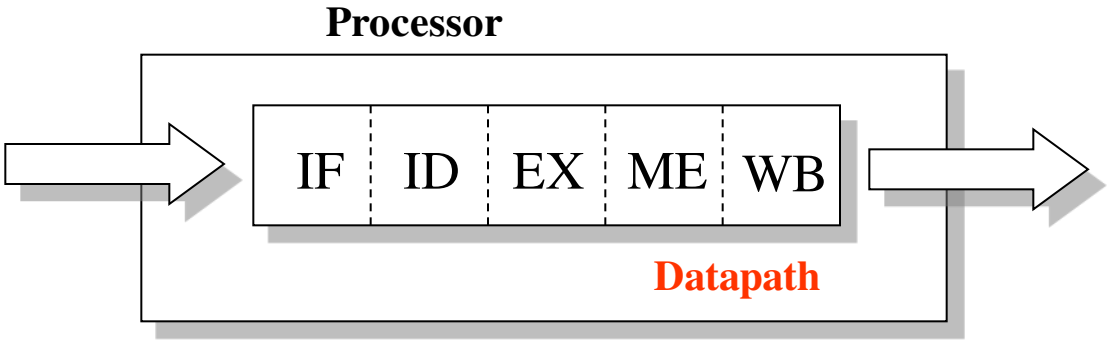



IF: Instruction Fetch ID: Instruction Decode EX: Execution

ME: Memory access WB: Write Back to registers

## 1. Scalar Datapath Processors

- The datapath includes the five circuit units
- All five units are implemented as single monolithic unit
- When an instruction is being executed, no other instruction can enter the datapath



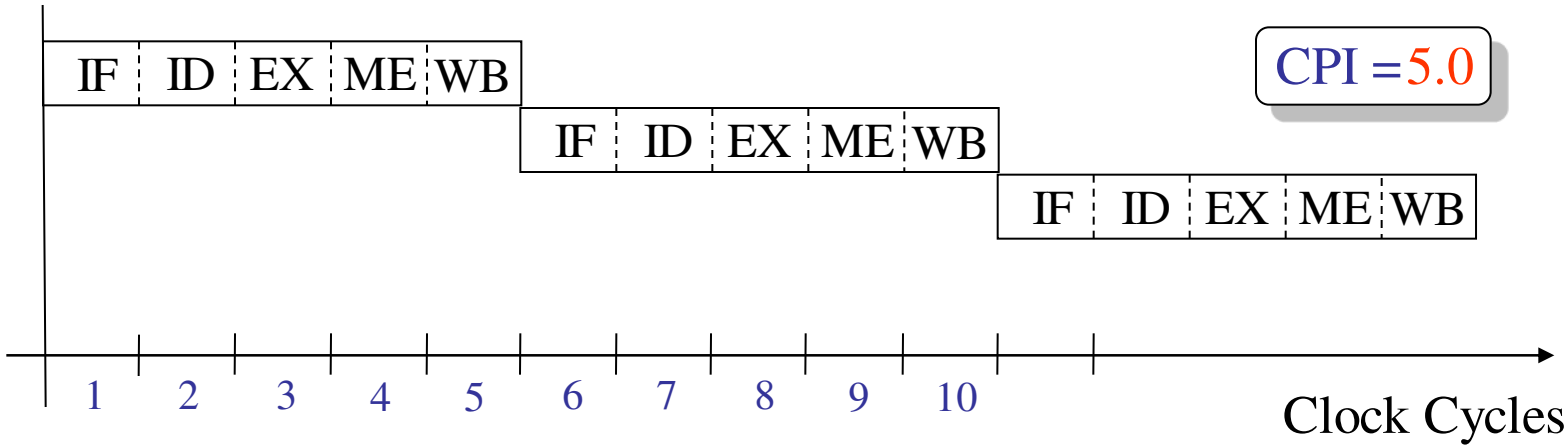

IF: Instruction Fetch ID: Instruction Decode EX: Execution

ME: Memory access WB: Write Back to registers

## 1. Scalar Datapath Processors

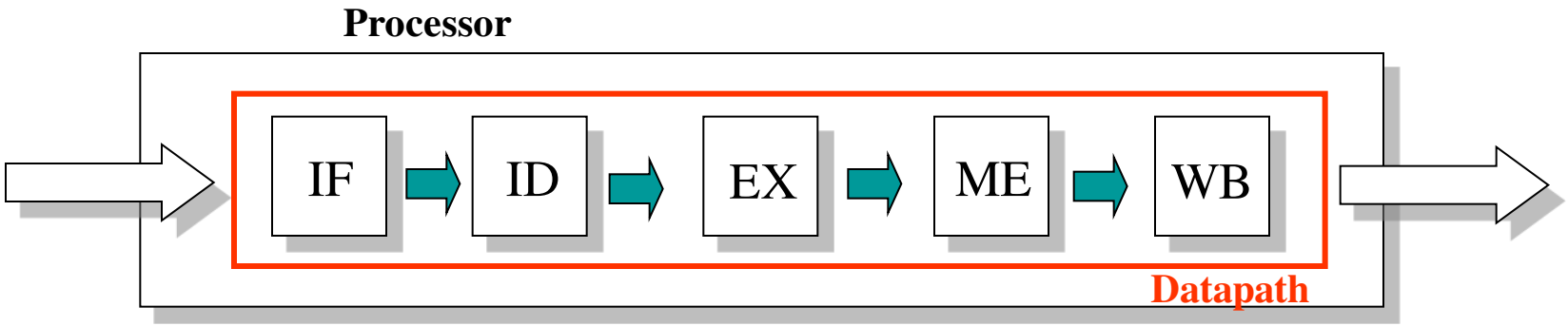
Representative processors in this generation

i4004, i8080, i8086, i80816, Z80, MC68000



## 2. Pipeline Datapath Processors

- All five units are implemented as independent units
- When an instruction is completed in a unit, the instruction can be forwarded to the next unit
- All five units can be occupied by different instructions



## 2. Pipeline Datapath Processors (continued)

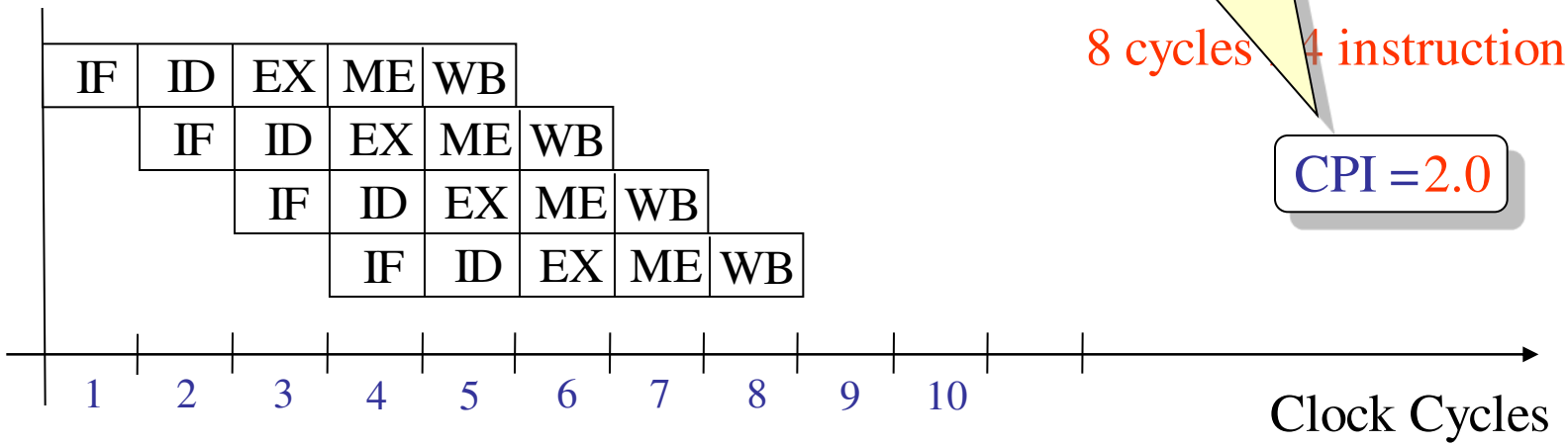
Representative processors in this generation

i80386, i40846, MC68040,

CPI for these only for these four instructions

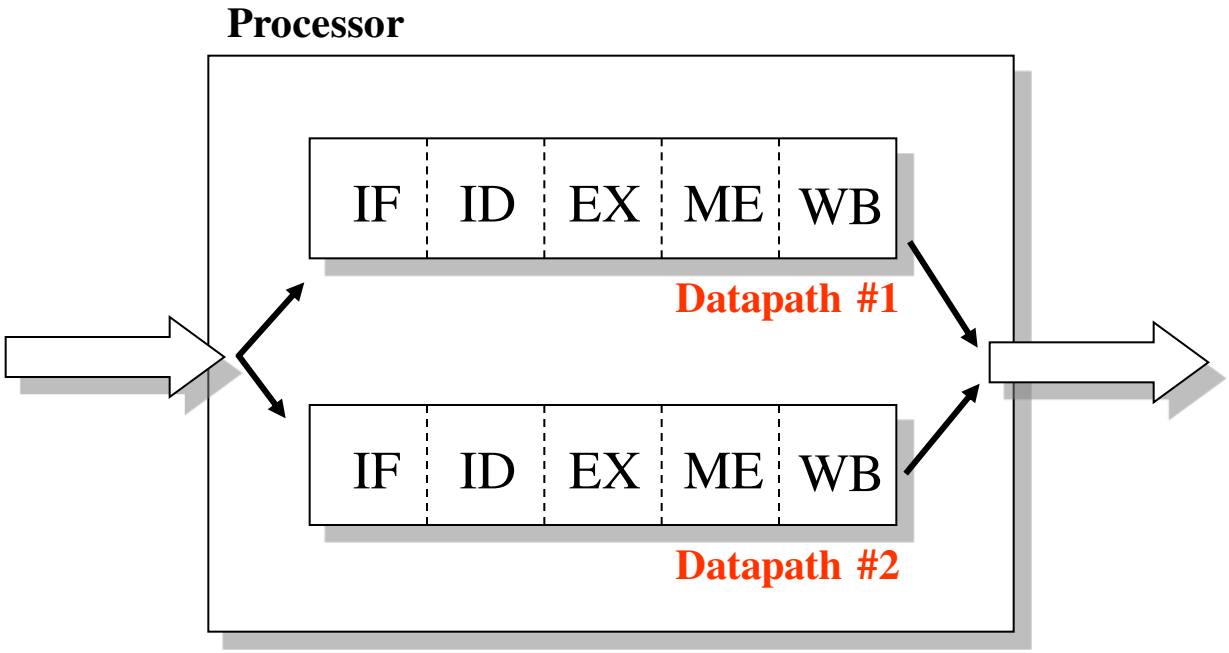
8 cycles 4 instructions

CPI =2.0

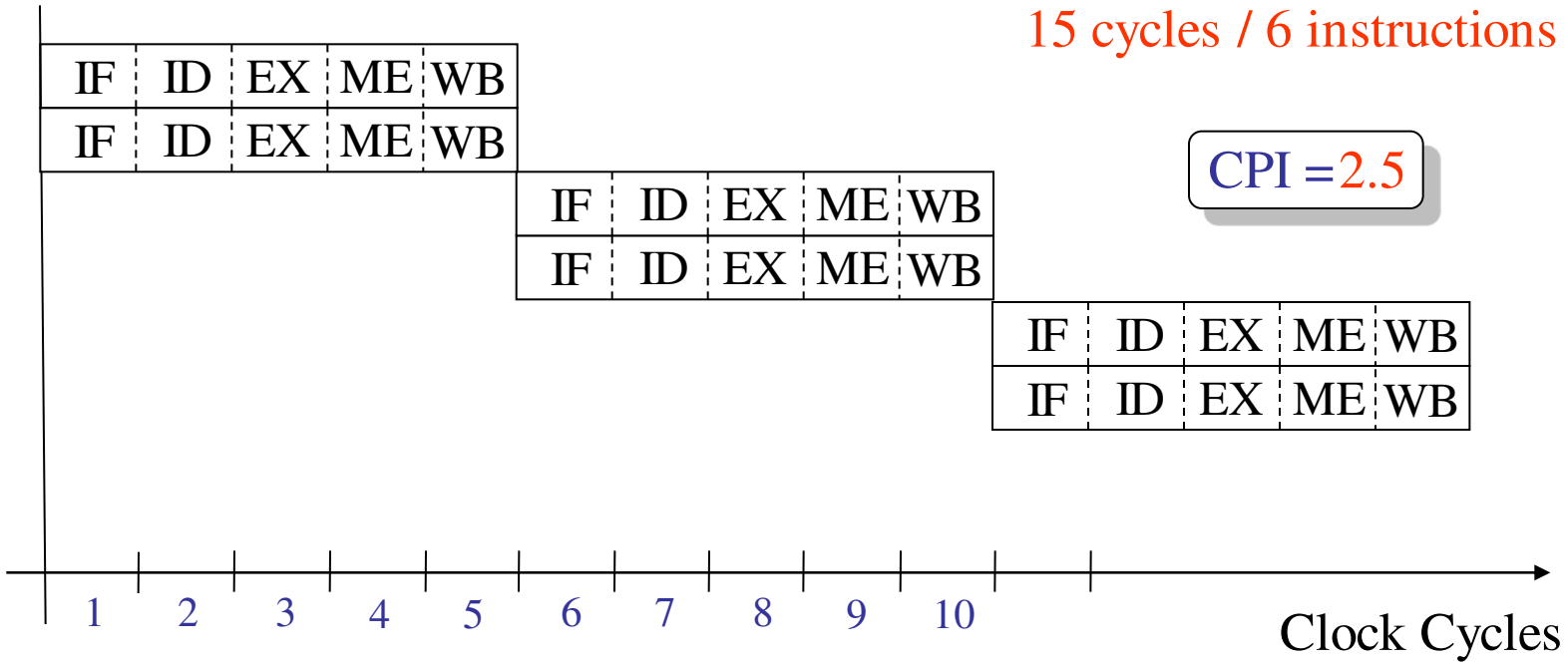


## 3. Super-Scalar Datapath Processors

- Multiple Scalar Datapath

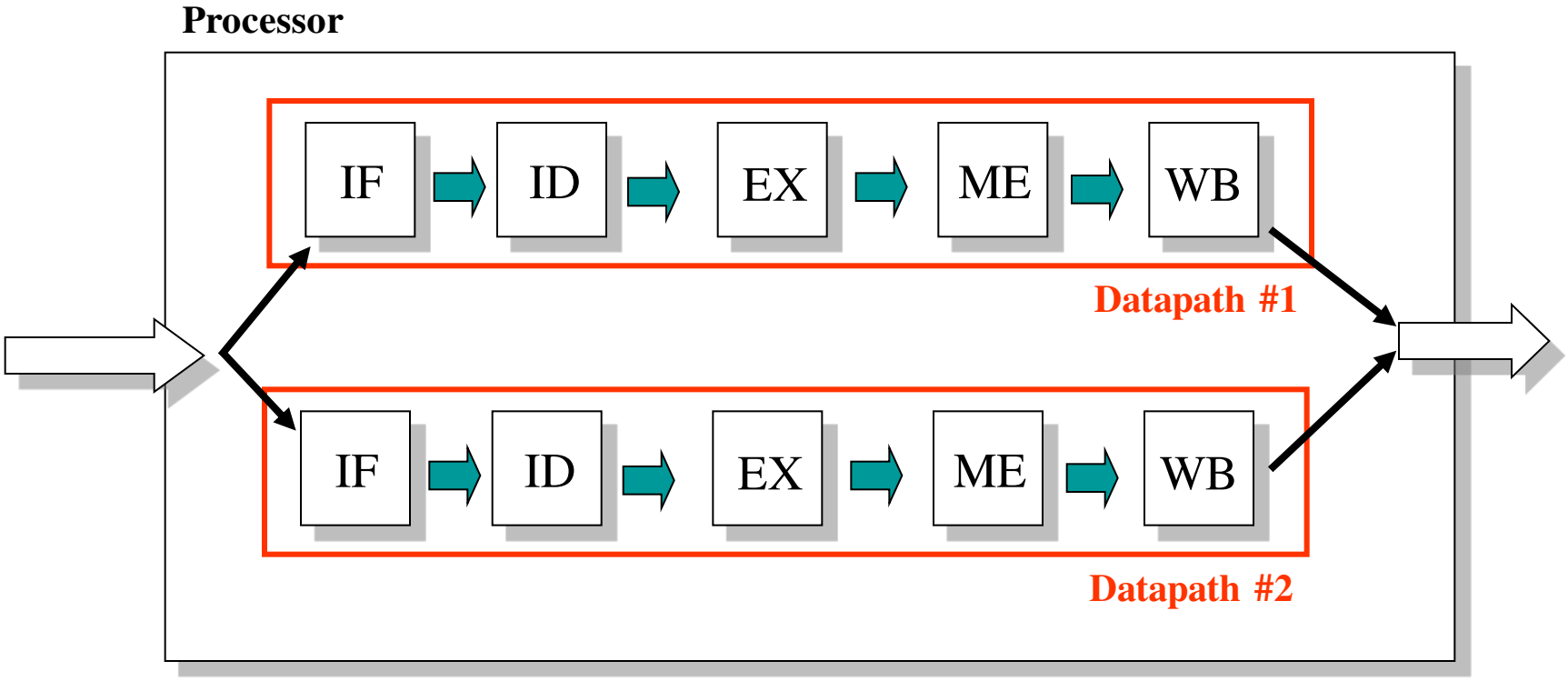


## 3. Super-Scalar Datapath Processors (continued)



## 4. Super-Pipeline Datapath Processors

- A combination of super-scalar and pipeline



## 4. Super-Pipeline Datapath Processors (continued)

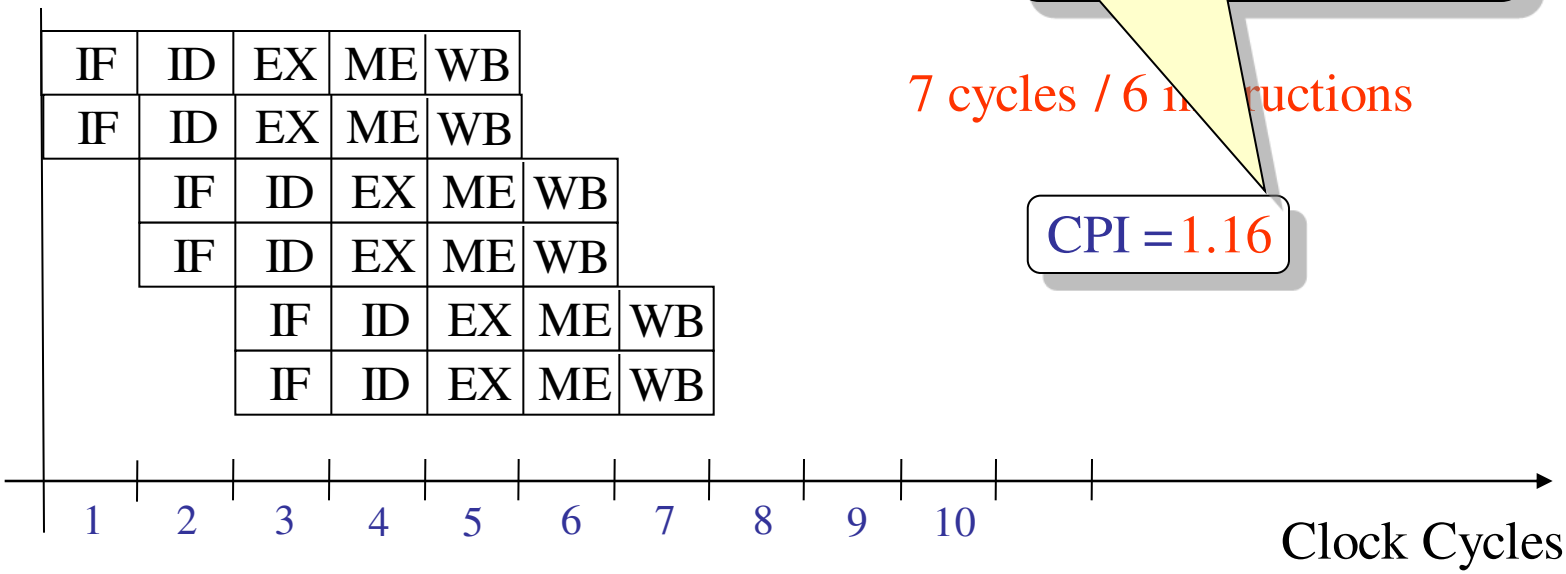
Representative processors in this generation

Pentiums (54, P55C), MIPS

CPI for these only for these six instructions

7 cycles / 6 instructions

CPI = 1.16



# SPEED UP IN PIPELINE

- **For a pipeline processor:**
  - *k-stage* pipeline processes with a clock cycle time $t_p$ is used to execute *n* tasks.
  - The time required for the first task $T_1$ to complete the operation = $k*t_p$ (if k segments in the pipe)
  - The time required to complete (n-1) tasks = $(n-1)*t_p$
  - Therefore to complete n tasks using a k-segment pipeline requires =k + (n-1) clock cycles.
- **For the non-pipelined processor :**
  - Time to complete each task = $t_n$
  - Total time required to complete n tasks=$n*t_n$
  - Speed up = non pipelining processing/pipelining processing

$$S = \frac{T_1}{T_K} = \frac{nt_n}{(k + (n-1))t_p}$$

- As the number of tasks increases, n becomes much larger than k-1, and approaches the value of n.
- Under this condition, speed up becomes
- $S = t_n / t_p$
- Assume the time taken for pipeline and non pipeline circuits are same then $t_n = k * t_p$
- Speed up reduces to= $S = (k * t_p)/t_p = k$
- This shows that the theoretical maximum speedup that a pipeline can provide is k, where k is the number of segments in the pipeline.