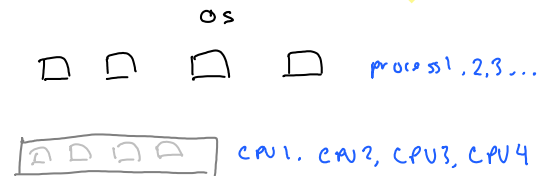


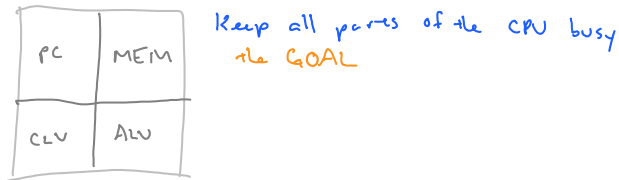
# 19 Instruction Parallelism

Monday, October 19, 2015 10:02 AM

## Instruction Level Parallelism



ScrAM



Pipelining

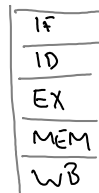


Instructions in Stages  
like an assembly line, building a car  
Keep hardware busy in chunks

Pipeline Stages of the MIPS CPU

- IF (instruction fetch)
- ID (instruction decode)
- EX (execute)
- MEM (read/write memory)
- WB (writes to register)

} Each needs  
to take  
the same  
amount of  
time



In essence,  
doing 5 pieces  
of instructions  
at the same  
time.

- JUMP is a problem
- Dependencies, (concurrent ADDs)

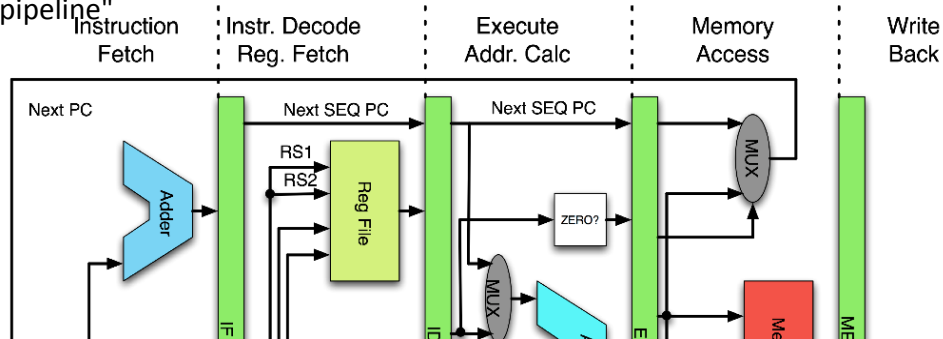
fetches from memory,  
writes to memory

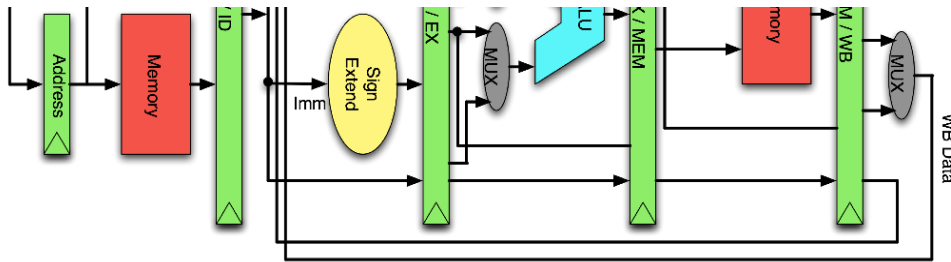
LIE, there are 2 memories,  
so no problem

"instruction" memory  
"data" memory

The diagram shows a curved arrow indicating the flow from memory access to the next stage. A note explains that there are two memories (instruction and data), so there is no problem with concurrent accesses.

"Google image search MIPS  
pipeline"





## HAZARDS

- A pipeline problem

### Structural Hazards

- When you want to do something, but you don't have enough hardware to do it
- "out of hardware"
- Ex: IF and MEM stages, both need memory
- If MIPS has one memory
- "never shows up on an exam"

### Data Hazard

- Dependency problems, like concurrent adds
- Add \$s0, \$t0, \$t1
- Add \$t2, \$s0, \$s7
- Every time you start an add instruction, need to wait 5 stages before you can use the answer to the add
- Fill in instructions that do nothing
  - "stalling the pipeline"
- Forwarding
  - The result from 1 stage directly goes into another stage
  - Sometimes there is a mandatory stall!

lw \$s0, 12(\$t1)  
 add \$s1, \$s0, \$t7

Pipeline diagram showing stages: IF, ID, EX, MEM, WB. The second instruction (add) starts at the EX stage, and the first instruction (lw) starts at the IF stage. The second instruction's EX stage is highlighted in red.

*need to stall  
 by 1!!  
 even w/ forwarding*

lw \$t1, 0(\$t0)  
 lw \$t2, 4(\$t0)  
 add \$t3, \$t1, \$t2  
 sw \$t3, 12(\$t0)  
 lw \$t4, 8(\$t0)  
 add \$t3, \$t1, \$t4

*← stalls for 1st inst. but this inst. still needs stall*  
*← if moved up, no stalls necessary!*

This is called "INSTRUCTION SCHEDULING"  
 compilers do a pretty good job at this