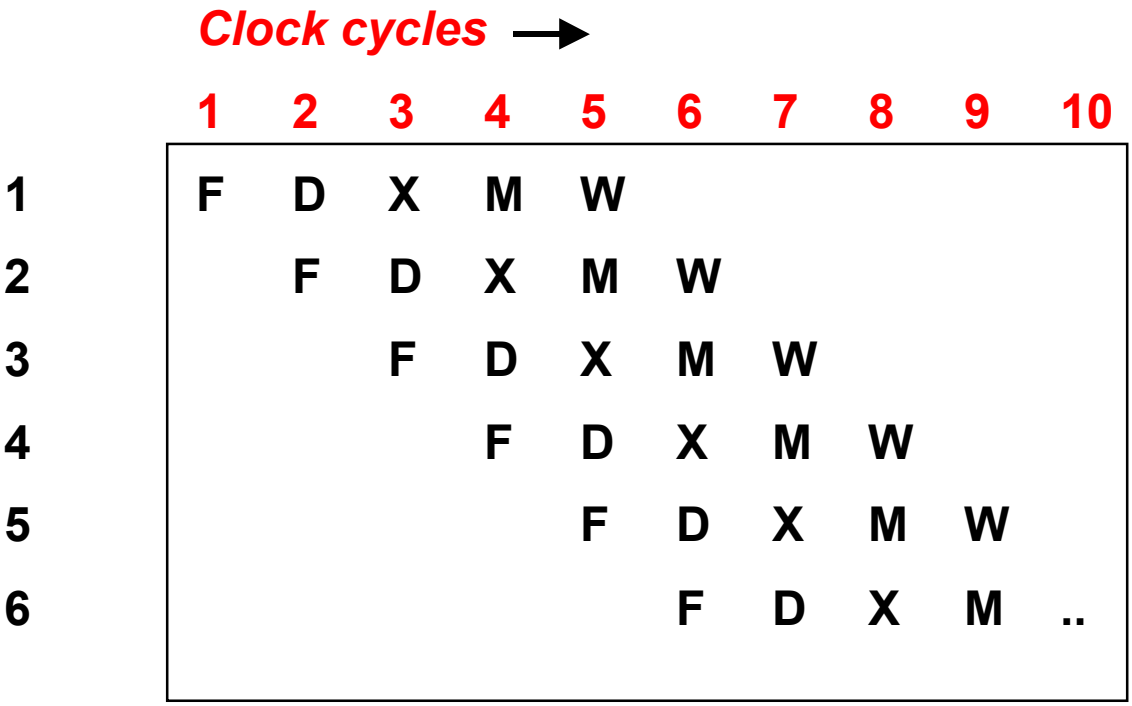
Pipelining

Five stages in the datapath of MIPS:

1. **Fetch**
2. **Decode**
3. **EXecute (or address calculation)**
4. **Memory access**
5. **Result Write back**

Instructions take 3-5 cycles to complete execution. How to overlap the operation of the different stages and maximize the througput?

Ideal Instruction Pipeline

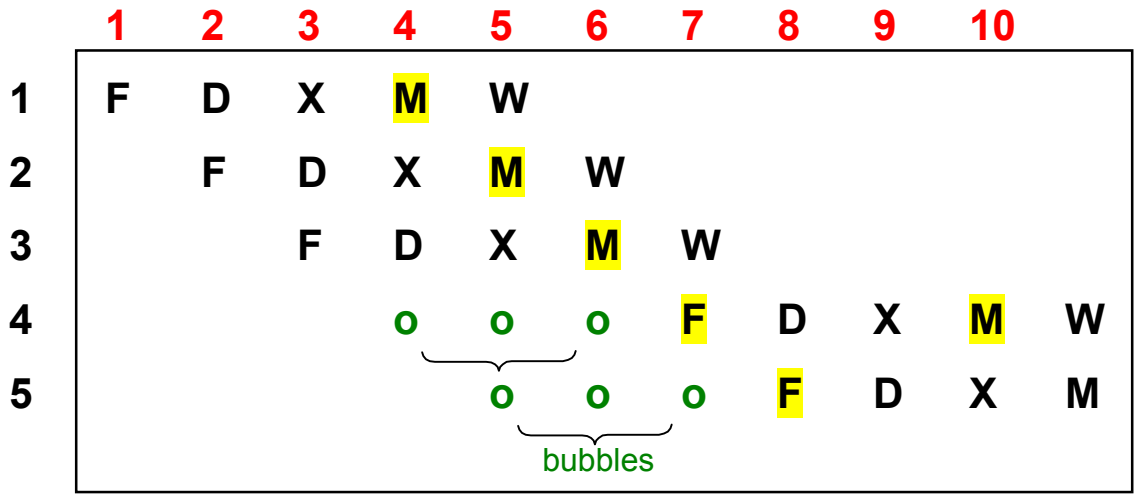


The average throughput for the ideal pipeline is 1 CPI, an improvement of nearly 400% over the previous implementation. Real speedup will however be **somewhat less** due to various types of hazards.

1. **Structural hazards**
2. **Data hazards**
3. **Control hazards**

Strcutural Hazard

Can we fetch an instruction (F) and read data from the memory at the same time? For a single port memory, the answer is NO.



These bubbles (stall cycles) will appear if we use a single-port memory to store both instructions and data.

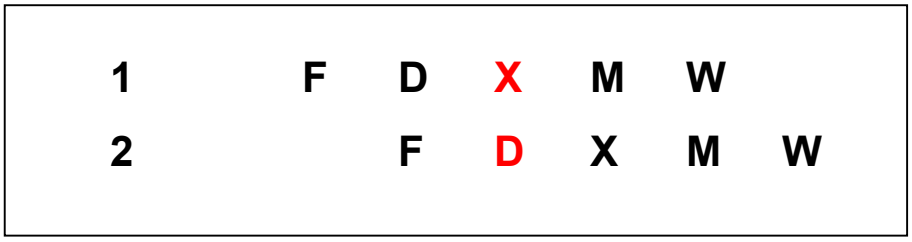
What is the average throughput (i.e. CPI) now?

Structural Hazards

Concurrent operations fighting for the same resource leads to structural hazards. Examples are

1. Two simultaneous memory access operations on a single-port memory. Note that separation of the **instruction memory** from **data memory** resolves this problem.

2.

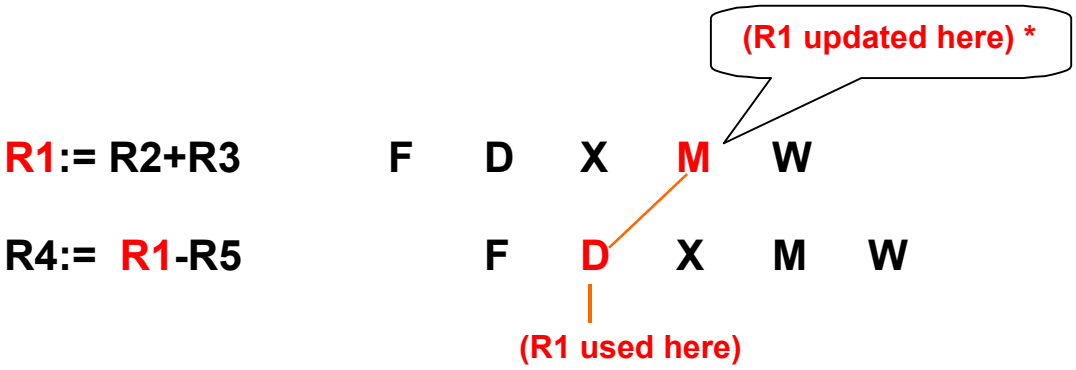


What if both X and D require an adder, and there is only one adder in the processor? Note that two separate adders will resolve this problem.

3. What if more than one write operations are to be performed on a register file that has only a single write port? Insert bubbles.

Data Hazards

Example of Read-After-Write (RAW) Hazard



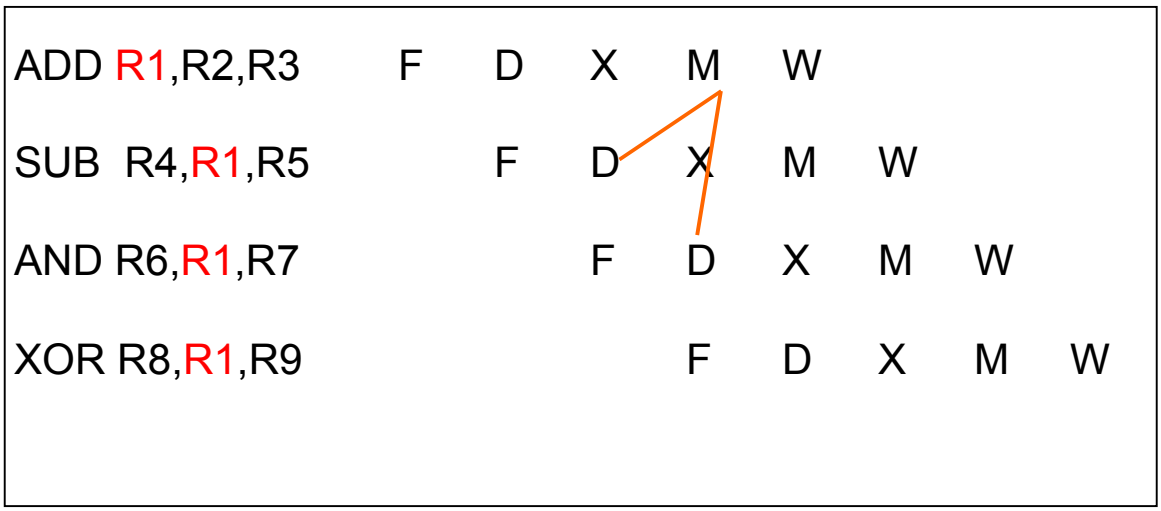
The second instruction will read an *old value* of R1.

Solution?

1. Insert bubbles.
2. Use internal data forwarding
3. Instruction reorganization by the compiler.

More about other types of data hazards later…

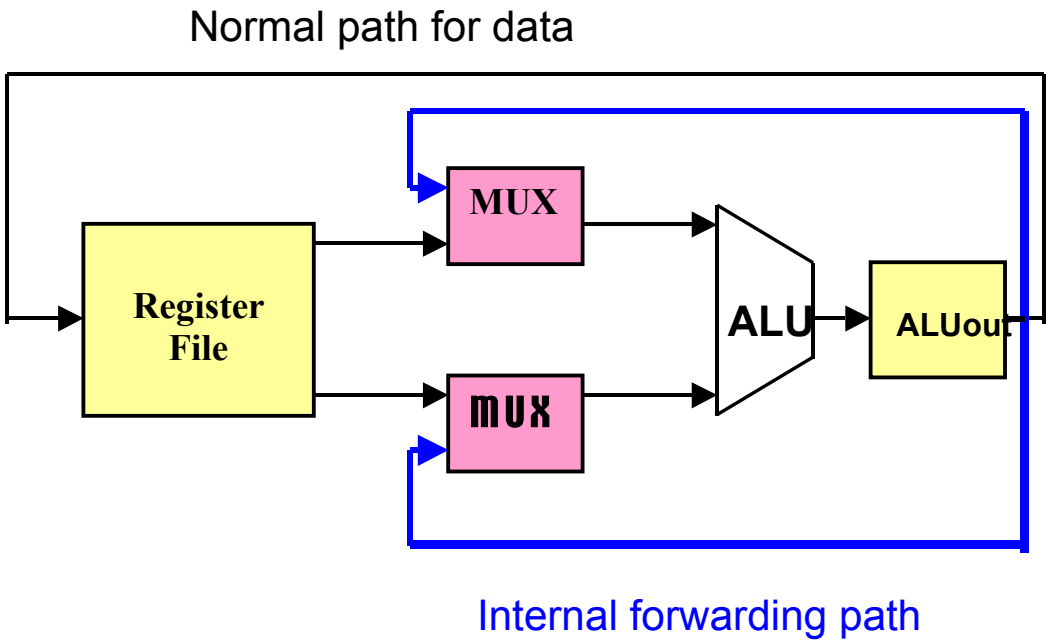
More Examples of RAW



Internal Data Forwarding

The updated value of the variable is made available very early using special internal data paths. This will help reduce the number of bubbles in the pipeline, and speed up the processor.

Internal Data Forwarding



Note that data is available in ALUout buffer just after the “X” phase, but it is written into the register file in the **M** phase (or in the **W** phase for the LW instruction) only.

Example of speedup using internal data forwarding

Data forwarding does not fully resolve the problem …

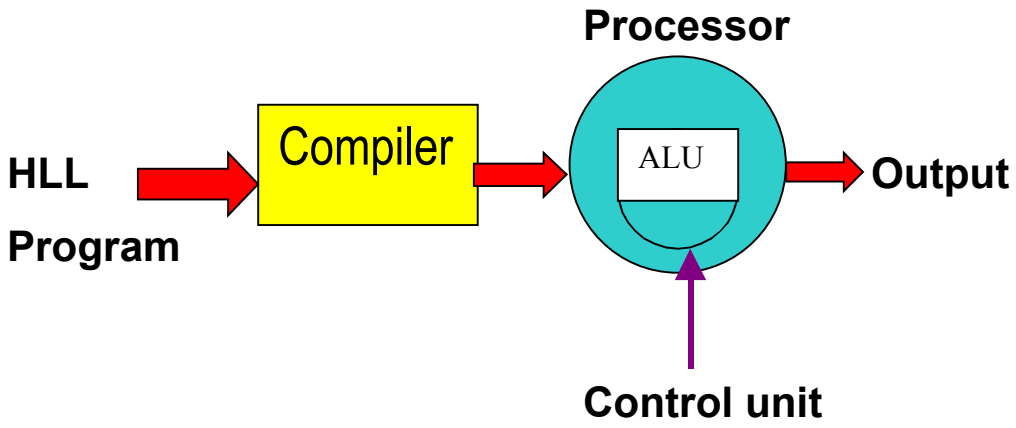
LW R1, 0(R2)	F	D	X	M	**W**			
SUB R4,R1,R5		F	D	**X**	M	W		
AND R6,R1,R7				F	D	X	M	W
OR R6,R1,R9				F	D	X	M	W

And the solution involves using bubbles too.

LW R1, 0(R2)	F	D	X	**M**	W			
SUB R4,R1,R5		F	D	o	**X**	M	W	
AND R6,R1,R7			F	o	D	X	M	W
OR R6,R1,R9				o	F	D	X	M

Only 1 stall cycle is required. However, without internal data forwarding, 2 stall cycles are necessary.

Two Design Choices



1. Either the control unit can be smart, i,e. it can delay instruction phases to avoid hazards. Processor cost increases.

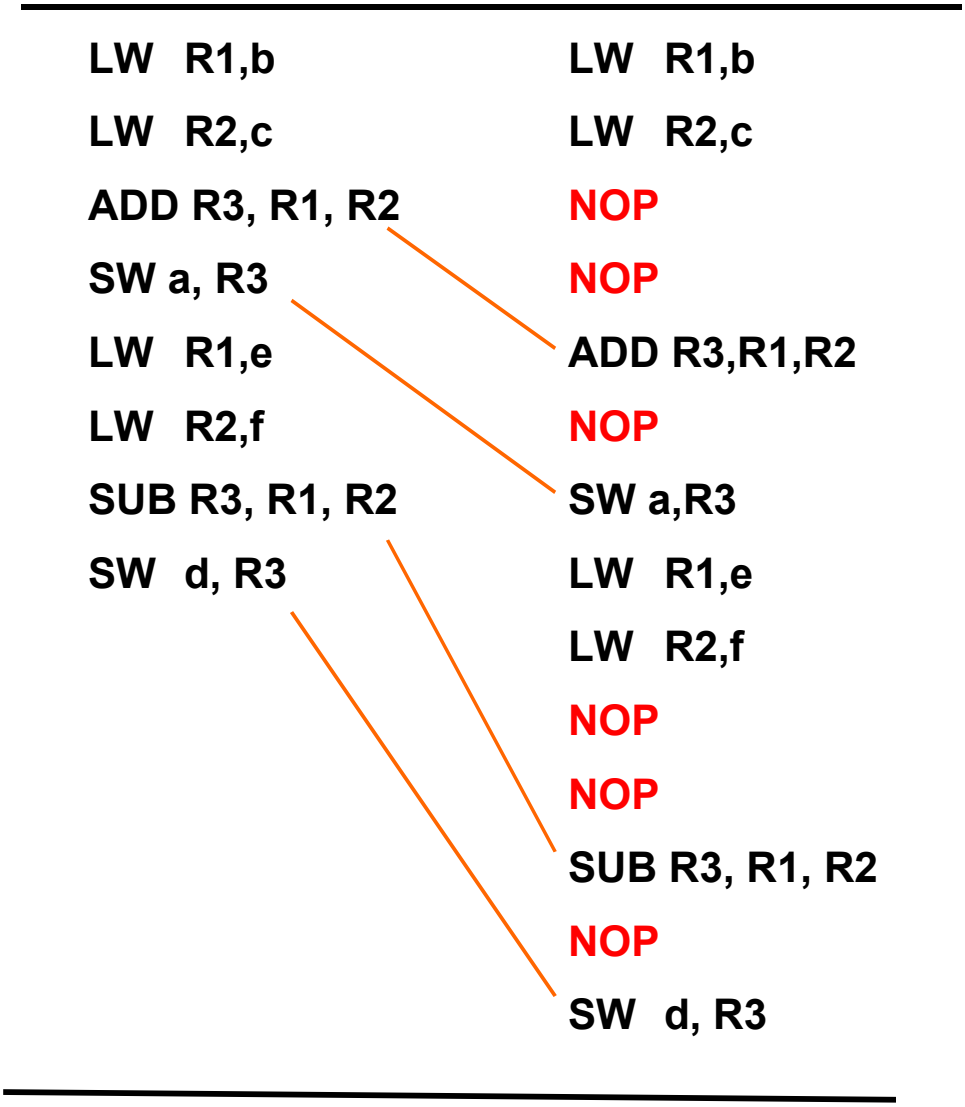
2. The compiler can be smart, i.e. produce optimized codes either by inserting NOPs or by rearranging instructions. The cost of the compiler goes up.

Instruction Reorganization by Compiler

To avoid data hazards, the control unit can insert bubbles.

As an alternative, the compiler can use **NOP** instructions.

Example: **Compute** **a:= b + c; d:= e + f**



Original code *Code generated by a smart compiler*

(Control unit remains unchanged)

Instruction Reorganization by Compiler

The compiler can further speedup by reorganizing the instruction stream and minimizing the no of NOP's.

Example: **Compute** **a:= b + c; d:= e + f**

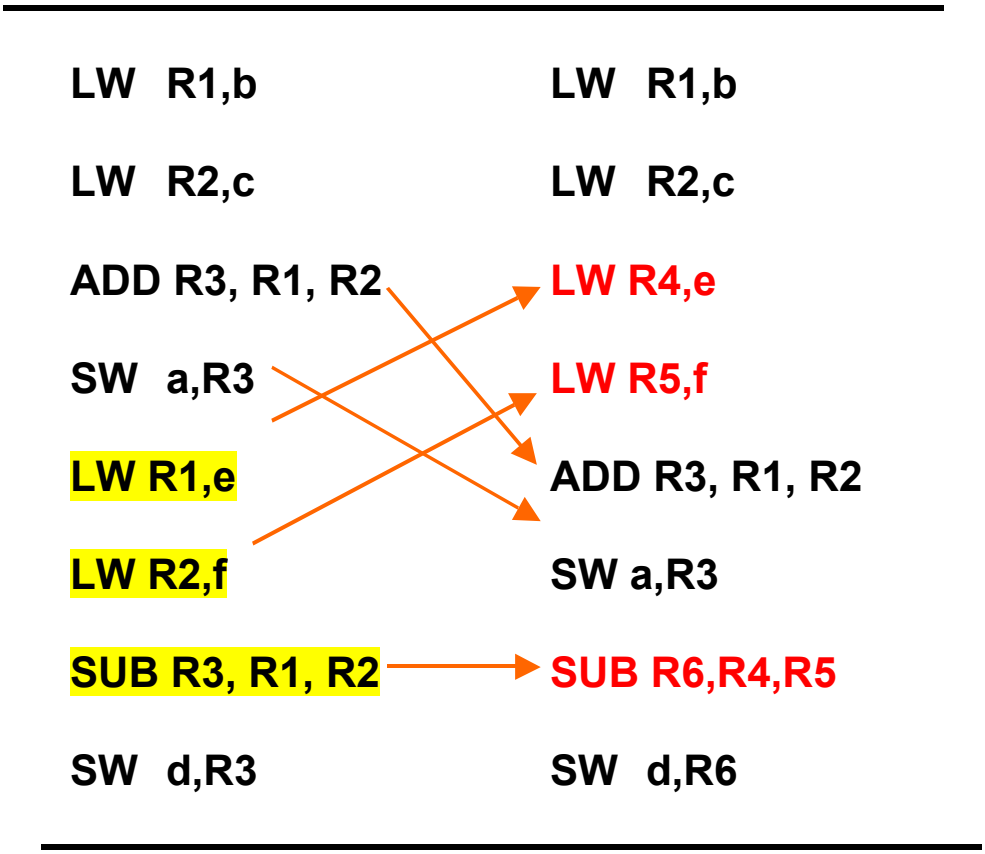
LW R1,b	**LW R1,b**
LW R2,c	**LW R2,c**
ADD R3, R1, R2	**LW R4,e**
SW a, R3	**LW R5,f**
LW R1,e	**ADD R3,R1,R2**
LW R2,f	**NOP**
SUB R3, R1, R2	**SW a,R3**
SW d, R3	**SUB R6, R5, R4**
	NOP
	SW d, R6
	NOP

Original code *Code reorganized by a smart compiler*

(Control unit remains unchanged)

Note the reassignment of registers

A More Refined Solution



original code *after reorganization*

This reorganization assumes that internal data forwarding paths have been added. Data hazard avoided without sacrificing speed.