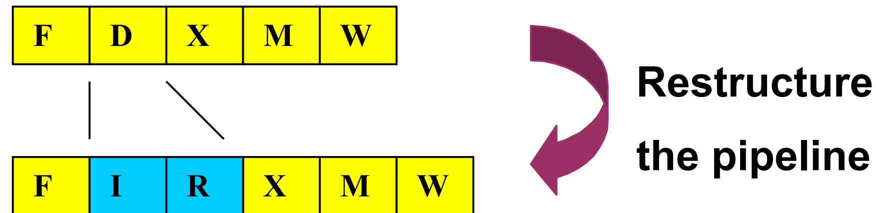
Advanced Pipelining Techniques

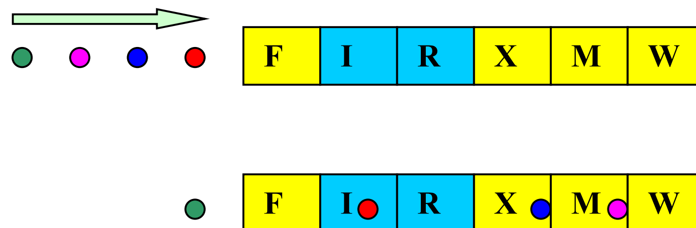
1. Dynamic Scheduling
2. Loop Unrolling
3. Software Pipelining
4. Dynamic Branch Prediction Units
5. Register Renaming
6. Superscalar Processors
7. VLIW (Very Large Instruction Word) Processors
8. EPIC (Explicitly Parallel Instruction Computers)
9. IA-64 Features

Dynamic Scheduling: CDC 6600 Style



Issue (I) **=** decode & wait for all structural hazards to clear

Read (R) = read operands



Instructions can be **issued out-of-order** and they can **complete out-of-order. First used in CDC 6600.**

Dynamic pipelining needs **additional buffer space between stages**, but will also **speedup computation.**

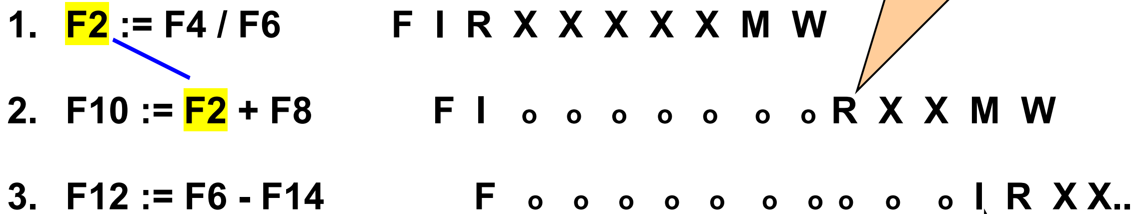
The Impact of Dynamic Scheduling

Assume that the processor has one add/subtract unit (2 cycles), one multiplier (3 cycles) and a division unit (5 cycles).

Example 1 (speedup)

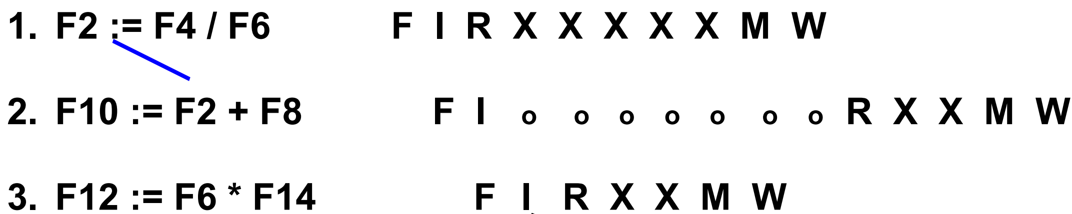
Read delayed to avoid RAW hazard (with no data forwarding

With static scheduling



Issue delayed to avoid structural hazard

With dynamic scheduling



No need to delay this one

Example 2 (Possibility of new hazards)

1. F2 := F4 * F6 **F I R X X X M W**

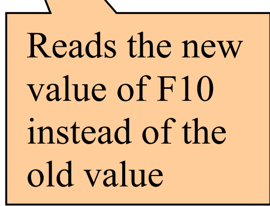
2. F8 := F10 * F12 **F o o o o I R X X X M W**

3. F8 := F14 / F6 **F I R X X X X X M W***

Note. This WAW hazard was *not possible* with static scheduling.

Example 3 (Write After Read, or WAR Hazard)

1. F12 := F4 / F6 F I R X X X X X M W
2. F8 := F10 / F12 F o o o o o o I o R* X X X X M
3. F10 := F14 + F6 F I R X X M W



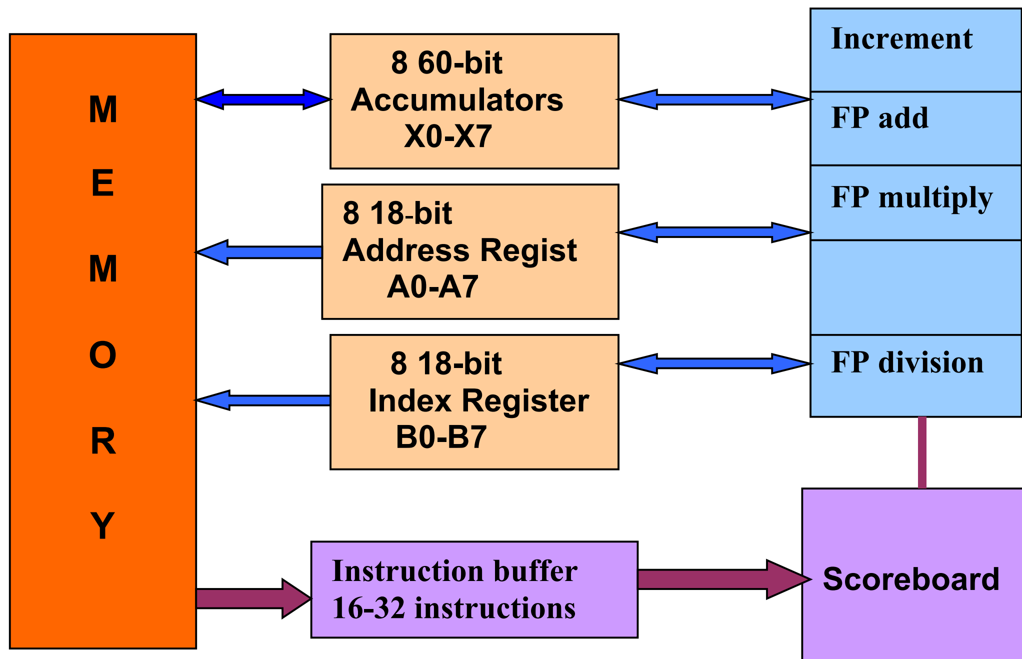
This is a new problem. Instruction 2 now reads the new value of F10 already modified by instruction 3. Did the programmer intend this? No!

WAR hazard is also known as anti-dependence. **One solution is to *delay the write (W) operation when a read (R) is pending from an earlier instruction.***

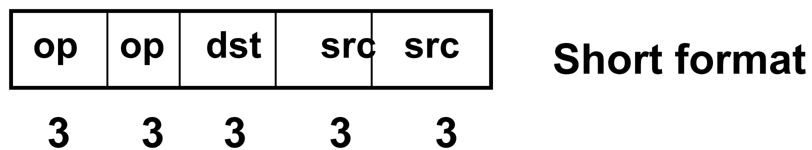
Question

Is WAR hazard possible without dynamic scheduling?

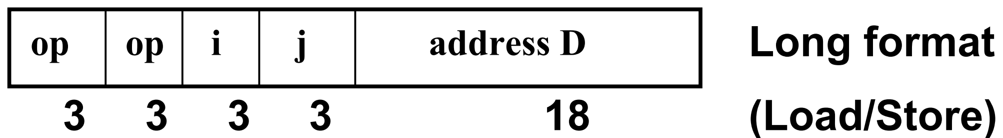
The CDC 6600 Processor (a quick look)



Instruction Formats



Short format



Long format
(Load/Store)

CDC 6600 dynamic scheduling (Scoreboarding)

Phase 1

ISSUE (Fetch & decode instruction)

Make sure that there is no structural hazard, or if there is no pending instruction with the same destination register. This avoids WAW hazard.

Phase 2

READ OPERANDS (Delay reading to avoid RAW)

Phase 3

EXECUTE

Phase 4

MEMORY OPERATION

Phase 5

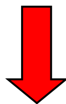
WRITE (Delay the write (W) operation if a read (R) is pending from an earlier instruction. This avoids WAR)

Tomasulo Approach

It is an alternative method of dynamic scheduling that was first used in IBM 360/91. Historically compilers resolved name dependencies by allocating new registers to existing variables (Register Renaming), Tomasulo used this idea to avoid WAR and WAW.

What is Register Renaming?

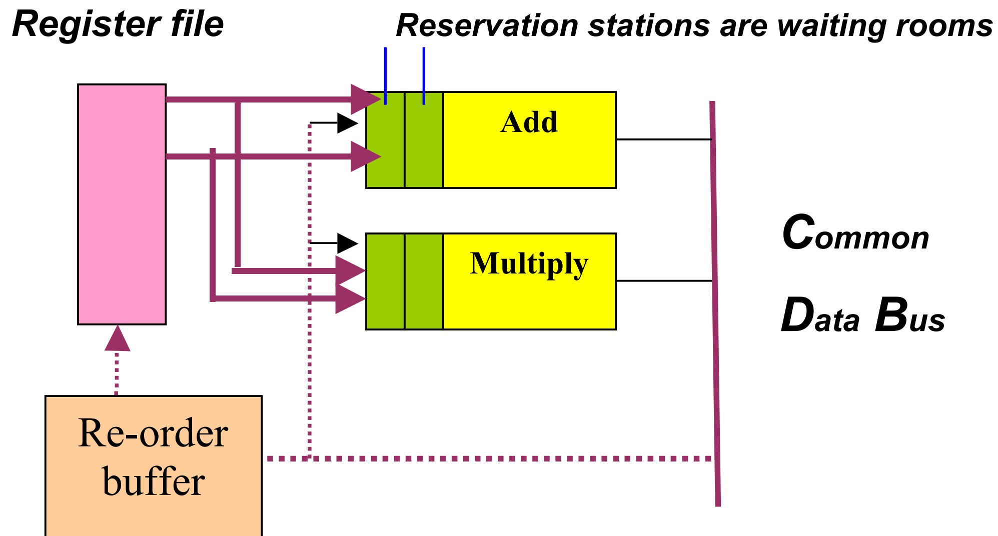
1	R1 := R2 + R3	
2	R2 := R3 – R4	(WAR with 1)
3	R5 := R1 & R2	(RAW with 2, RAW with 1)
4	R1 := R3 or R4	(WAW with 1, WAR with 3)
5	R9 := R1 – R12	(RAW with 4)



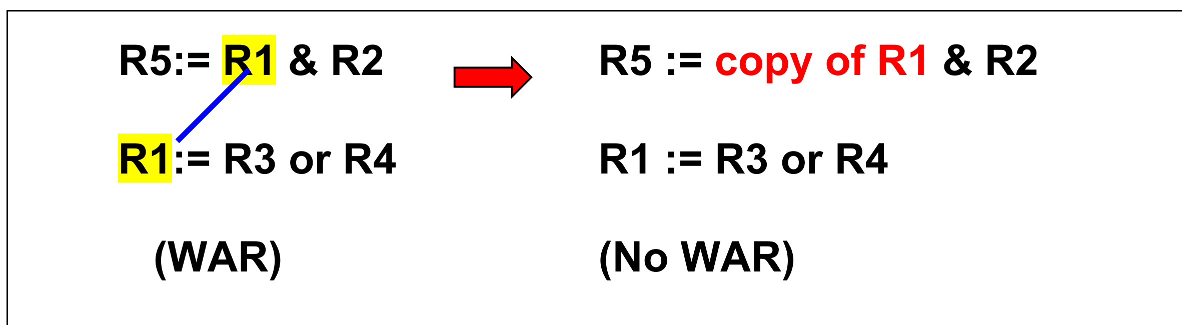
1	R1 := R2 + R3	
2	R6 := R3 – R4	WAR is gone!
3	R5 := R1 & R6	
4	R7 := R3 or R4	Both WAW and WAR are gone!
5	R9 := R7 – R12	

Tomasulo Scheme

Run-time Renaming using Reservation Stations.



(Issue) If the latest copy is available then send a copy of the operands from the register file to a reservation station. Otherwise, wait to receive the result from another functional unit.



(Execute) When both operands are available, and the functional unit is available, execute (avoids RAW)

(Write) Write Result to a Common Data Bus. This helps forward data to multiple reservation stations. At the same time, the result is sent to a commit unit called the re-order buffer.

Re-order buffer

Results are written into a pool of temporary buffers. They are written into the actual registers when there is no risk of a WAW hazard. Data may be forwarded from here too (much like internal data forwarding). It is easy to undo the execution of speculative instructions.

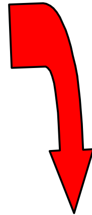
Avoiding data hazards in Tomasulo Scheme

More Examples

1 F6 := M[R2+34]

2 F10 := F10/F6

3 F6 := F8+F2



If instruction 1 has not computed the new value of F6 when instruction 2 has been issued, then instruction 2 waits at the reservation station of the divider. F6 is forwarded from the output of the Load Buffers.

To avoid WAW between 1 and 3, the values of F6 are committed in instruction order from the re-order buffer.

Dynamic Branch Prediction

In the simple 5-stage MIPS pipeline, predict-not-taken is simple prediction strategy. This is ok since the penalty for misprediction is not much.

If the penalty is large (as in many deeply pipelined machines or superscalar processors), we cannot afford make frequent incorrect predictions. The predictions have to be more sophisticated. Some popular schemes are:

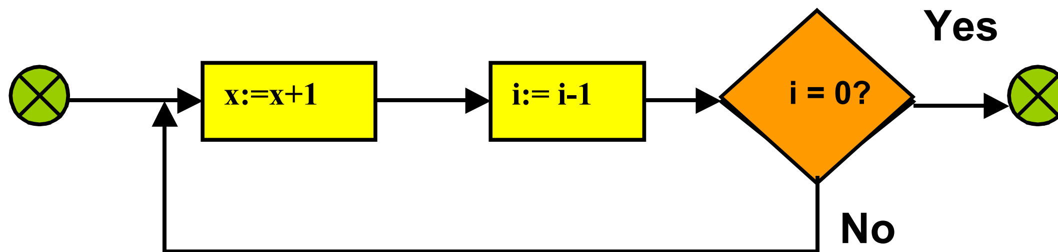
1-bit/2-bit prediction using Branch Prediction Buffers

Branch target buffer

A 1-bit scheme for dynamic branch prediction

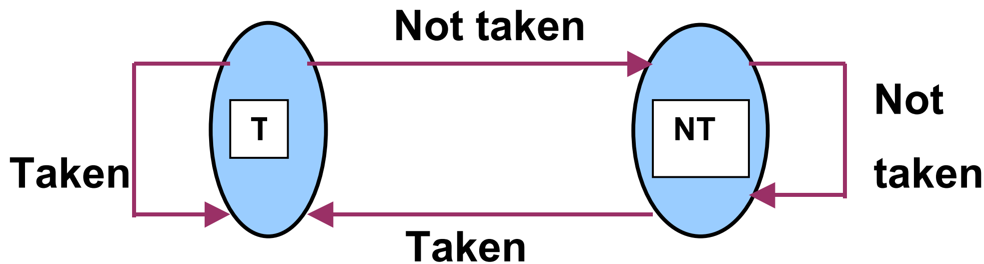
for (i =10, i > 0, i =i - 1)

x := x+1



With the branch instruction, a history bit is associated.

The bit is changed as follows:

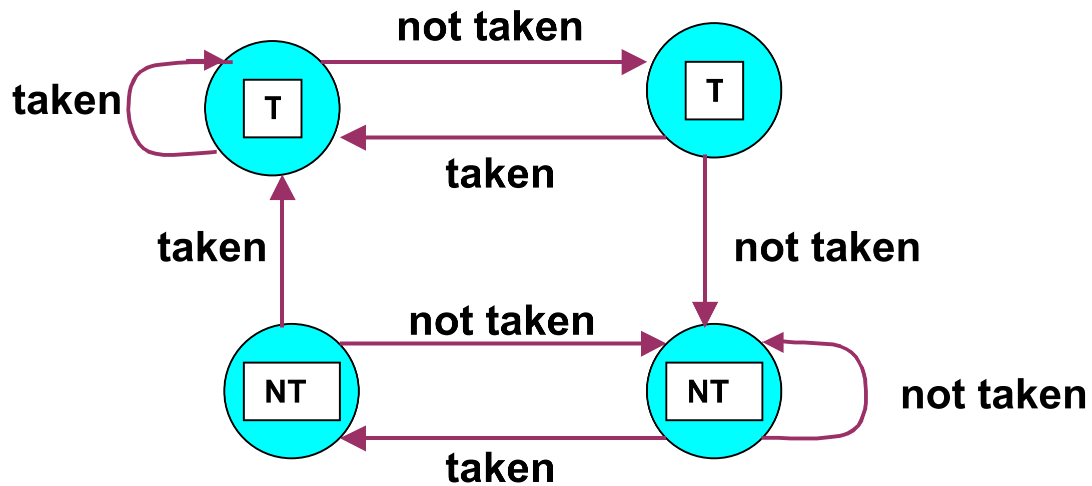


(T= taken, NT= not taken)

The success rate of branch prediction is 80%.

F, T, T, T, T, T, T, T, T, F, F, T, T, T, T, T, T, T, T, F, F

A 2-bit scheme

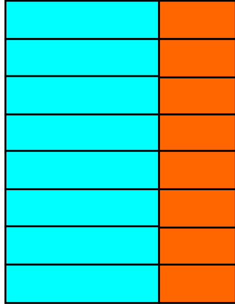


For the same program, the success profile is:

F, F, T, T, T, T, T, T, T, F, T, T, T, T, T, T, T, T, T, F, T

Predictions do not change here

Branch prediction buffer



The buffer is indexed by the last few bits of the address of the branch instructions.

Buffer read in the “D” phase. Penalty for wrong prediction depends on when the PC is calculated.