More pipeline facts

Where to spend your $

The steady state throughput is determined by the slowest stage of the pipeline. There is no point in speeding up the ALU if the memory units are slow.

Performance Enhancement

$$\text{Speedup} = \frac{\text{Execution time on non-pipelined CPU}}{\text{Execution time on pipelined CPU}}$$

Control Hazard

lw r1, 32(r2)	F	D	X	M	W			
L1: beq r1, r3, L2		F	D	X	M	W		
addi r1, r1, 1			F	D	X	M	W	
addi r4, r4, -1				F	D	X	M	
L2:								

Equality detected here

If the condition of beq is true, then two wrong instructions will enter the pipe, and need to be flushed out.

Solution 1: Insert bubble

	lw r1, 32(r2)	F	D	X	M	W		
L1:	beq r1, r3, L2		F	D	X	M	W	
	addi r1, r1, 1			o	o	o	o	o
	addi r4, r4, -1				o	o	o	o
L2:						F	D	X

Branch penalty = 2 cycles

But how will the control unit know that the conditional branch will be taken? Use prediction. Since most forward branches are not taken, a common prediction by the control unit is that the branch will not be taken.

Solution 2: Predict not taken

Case 1. Branch not taken

lw r1, 32(r2)	F	D	X	M	W		
L1: beq r1, r3, L2		F	D	X	M	W	
addi r1, r1, 1			F	D	X	M	W
addi r4, r4, -1				F	D	X	M
L2:					F	D	X

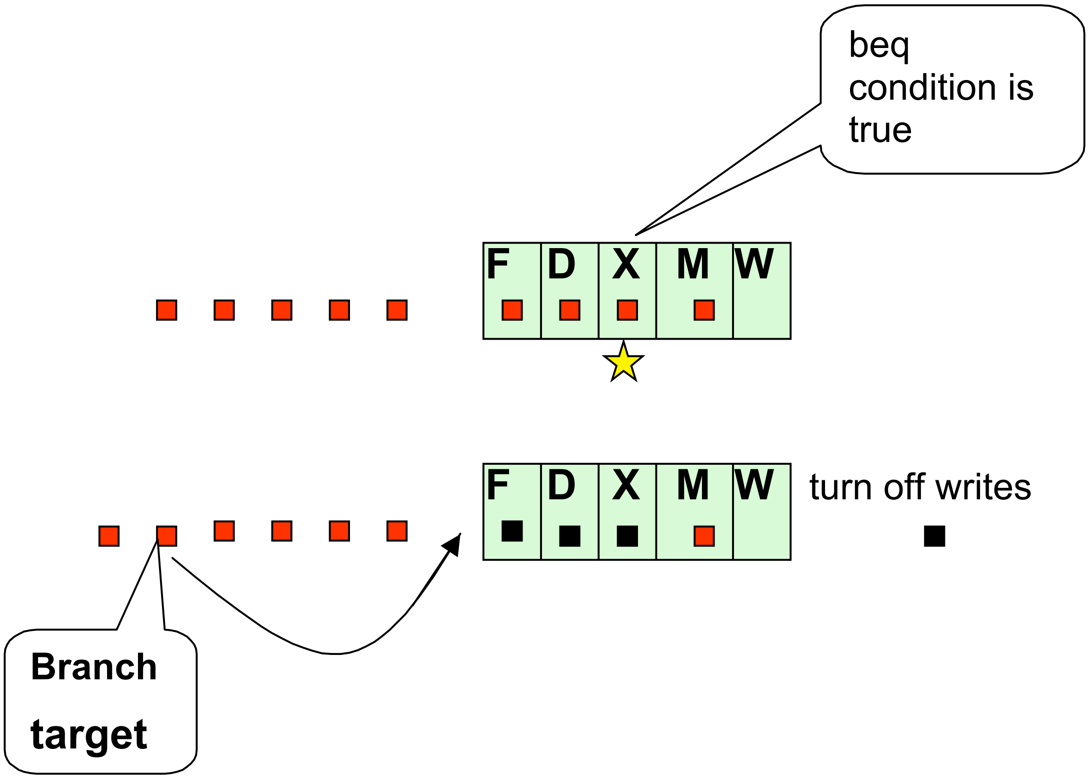
Case 2. Branch taken

lw r1, 32(r2)	F	D	X	M	W		
L1: beq r1, r3, L2		F	D	X	M	W	
addi r1, r1, 1			F	D	o	o	o
addi r4, r4, -1				F	o	o	o
L2:					F	D	X

2 wrong instructions

Flushing the pipe

The wrong instructions need to be flushed out from the pipeline. One mechanism is to disable the writes (by generating proper control signals), so that the wrong instructions are reduced to NOP.



Speedup

Estimate the slowdown now (assume that 15% instructions are conditional branch).

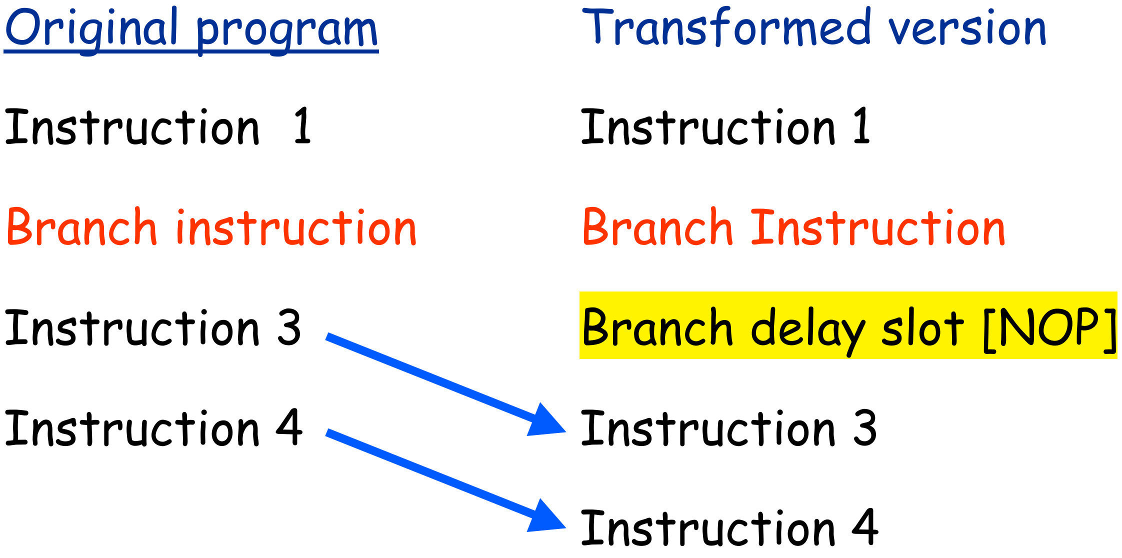
$$\text{Speedup} = \frac{\overset{\text{Ideal speedup}}{5}}{1 + \text{branch frequency} \times \text{branch penalty}}$$

Reducing Branch Penalty

Early detection of *branch condition* and computation of the *branch target address* help reduce branch penalty. One can modify the datapath to detect the branch condition in the D-stage and reduce the branch penalty to one cycle.

Compiler Scheduling of Branch Delay Slots

(Assume that branch penalty = 1 cycle)



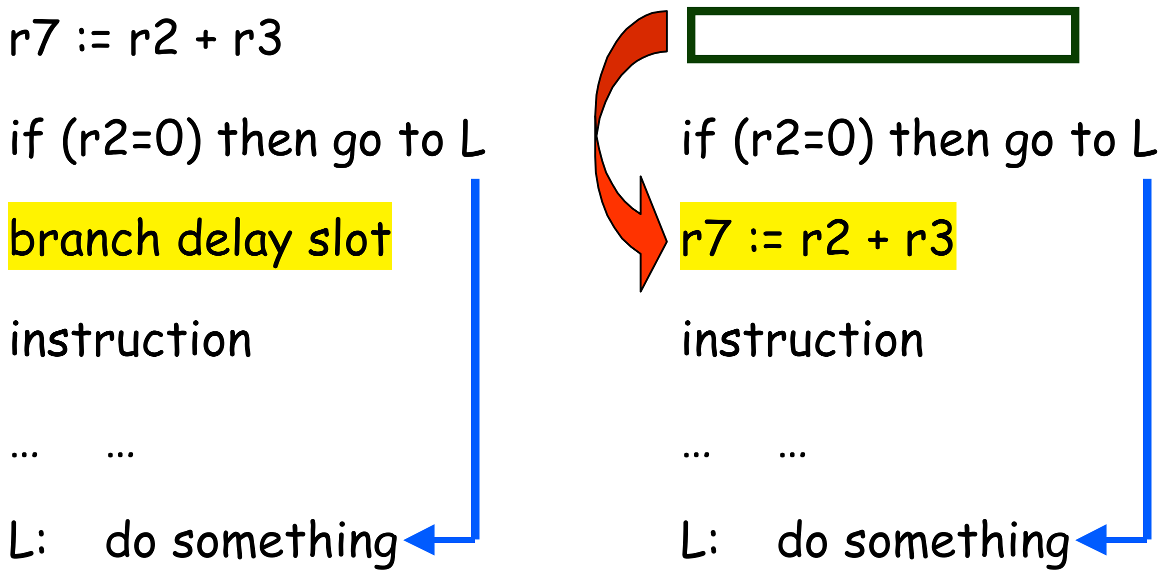
Original program	Transformed version
Instruction 1	Instruction 1
Branch instruction	Branch Instruction
Instruction 3	Branch delay slot [NOP]
Instruction 4	Instruction 3
	Instruction 4

Scheduling a NOP after a conditional branch is equivalent to “fetching and flushing an instruction” by the control unit.

Al alternative is to schedule some meaningful instruction in the branch delay slot.

Instruction reordering by the compiler

Example 1

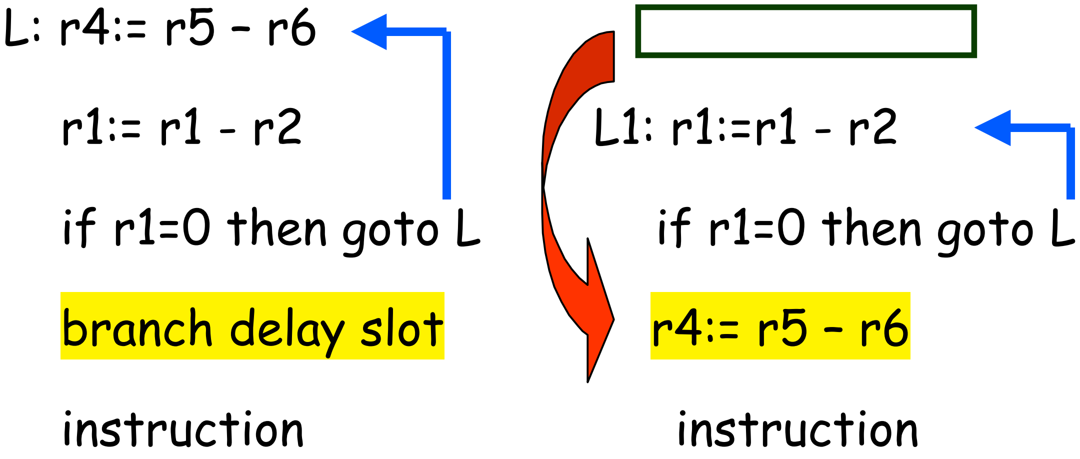


```
r7 := r2 + r3
if (r2=0) then go to L
branch delay slot
instruction
...     ...
L:    do something
```

```
if (r2=0) then go to L
r7 := r2 + r3
instruction
...     ...
L:    do something
```

NOTE: Always improves performance if such an unrelated instruction is available.

Example 2 (Backward branch)



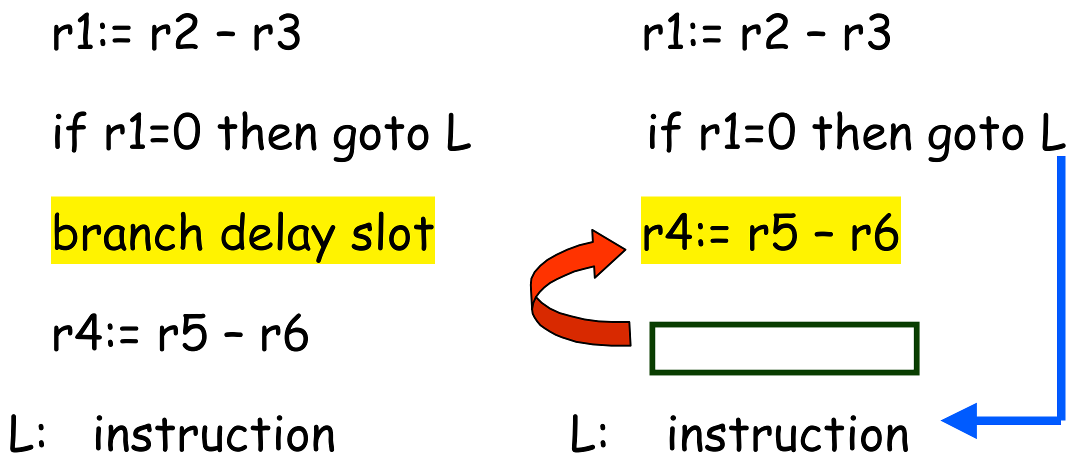
```
L: r4:= r5 – r6
   r1:= r1 - r2
   if r1=0 then goto L
   branch delay slot
   instruction
```

```
L1: r1:=r1 - r2
    if r1=0 then goto L
    r4:= r5 – r6
     instruction
```

Performance improves when the branch is taken.

However, it must be OK to execute r4:= r5-r6 when the branch is not taken (or else the instruction has to be canceled)

Example 3 (Forward branch)



r1:= r2 - r3

if r1=0 then goto L

branch delay slot

r4:= r5 - r6

L: instruction

r1:= r2 - r3

if r1=0 then goto L

r4:= r5 - r6

L: instruction

Performance improves when the branch is NOT taken. However, it must be OK to execute r4:= r5-r6 when the branch is taken (or else the instruction has to be canceled)

Brach prediction

Static prediction vs. dynamic prediction

Static prediction is done during compile time. It is a rule of thumb that forward branches are usually not taken, and backward branches are usually taken.

Dynamic branches are predicted by profiling the run-time behavior of programs.

Ideally we want to have a "crystal ball", so that we can see ahead of time whether a branch will be taken or not.