

EE 457 Unit 6c

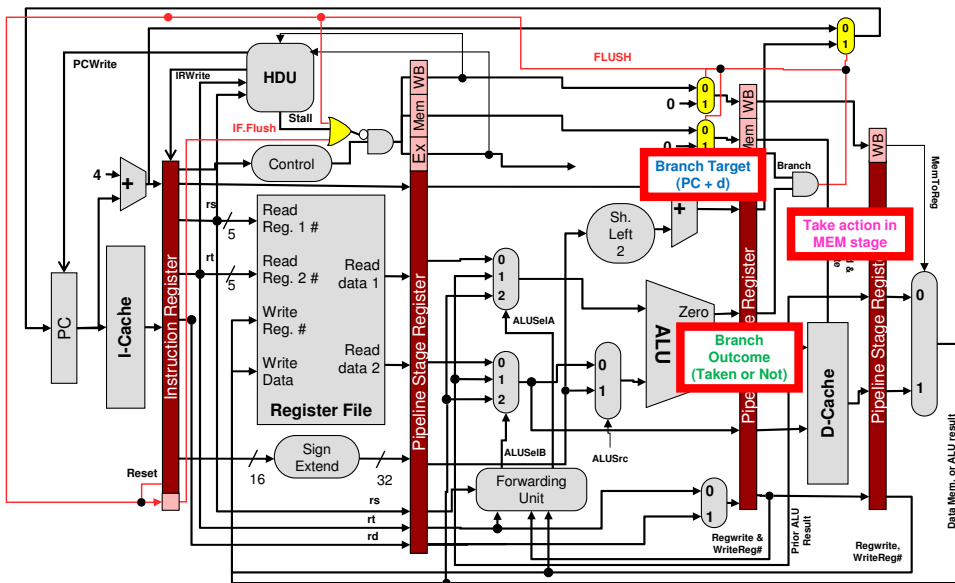
Control Hazards

Overview

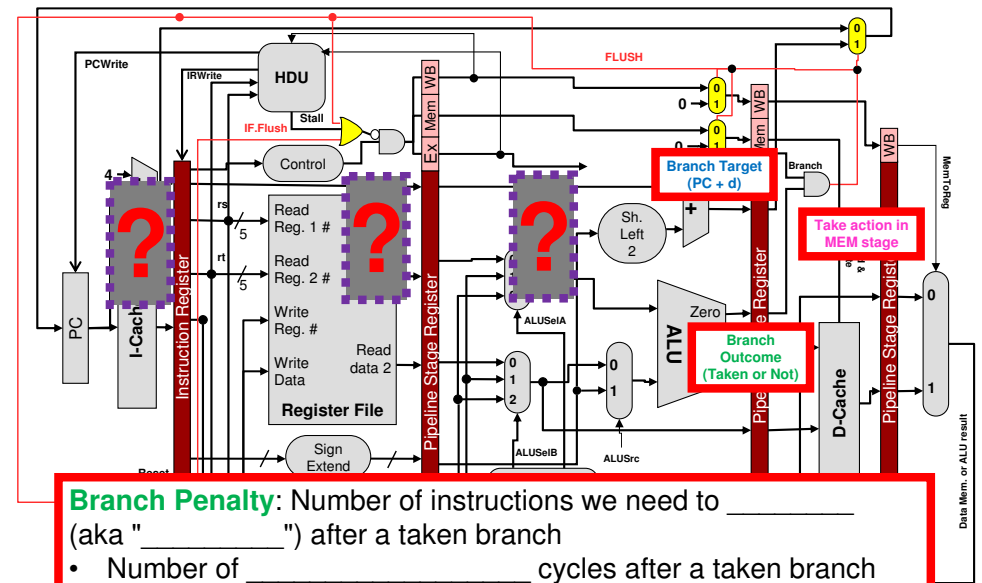
- Branching requires knowing 2 values:
 - Branch : Should I branch or not (i.e. is \$1 == \$3)?
 - Only 2 option (yes or no)
 - Use T = Taken and NT = Not taken to describe these 2 outcomes
 - Branch target: Where should I branch?
 - Requires computation of new PC value (i.e. PC =)
- Where in the pipeline do I know these values?
 - Branch outcome: End of EX stage (_____ bit from ALU)
 - Branch target: End of EX stage (_____)
 - End of EX stage...Too _____ to do anything with it (wait until MEM stage)

40:	BEQ	\$1, \$3, 28
44:	AND	\$12, \$2, \$5
48:	OR	\$13, \$6, \$2
52:	ADD	\$14, \$2, \$2
...		
72:	LW	\$4, 50 (\$7)

Branch Outcome and Target



Branch Penalty



Branch Penalty: Number of instructions we need to _____ (aka "_____") after a taken branch

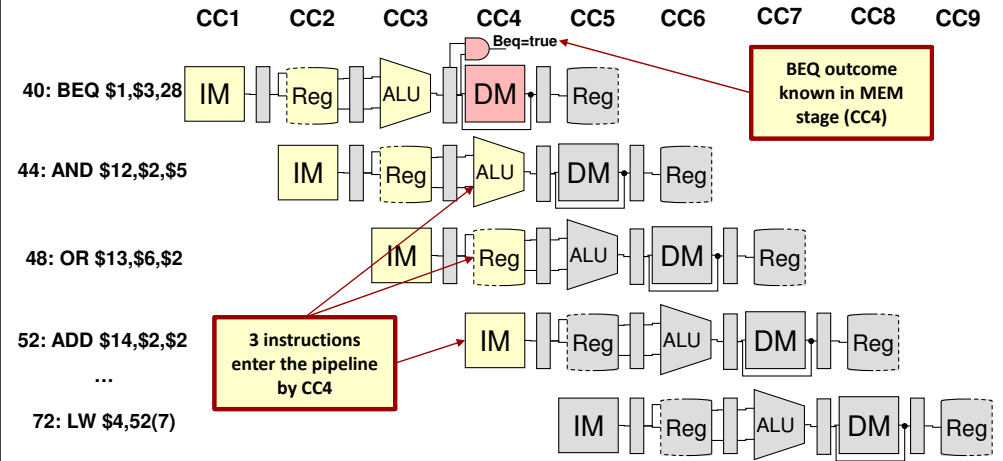
- Number of _____ cycles after a taken branch

Control Hazards

- Control (branch) hazards are named such because they deal with issues related to program control instructions (branch, jump, subroutine call, etc.)
- There is some delay in determining a branch or jump instruction and thus incorrect instructions may already be in the pipeline

40: BEQ	\$1, \$3, 28
44: AND	\$12, \$2, \$5
48: OR	\$13, \$6, \$2
52: ADD	\$14, \$2, \$2
...	
72: LW	\$4, 50 (\$7)

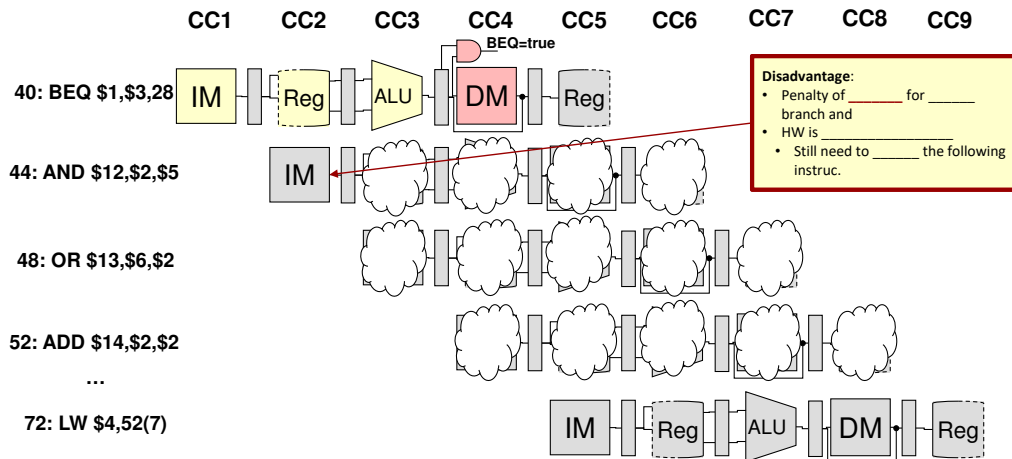
An Opening Example



- How can we solve this problem?

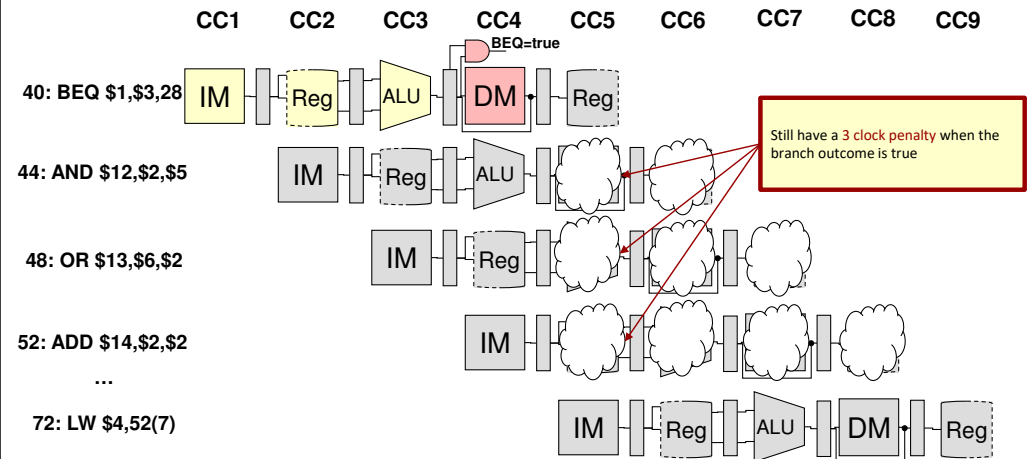
Option 1: Stalling

- Option 1:** Start stalling the pipeline as soon as you detect that it is a branch and keep stalling until you know the outcome



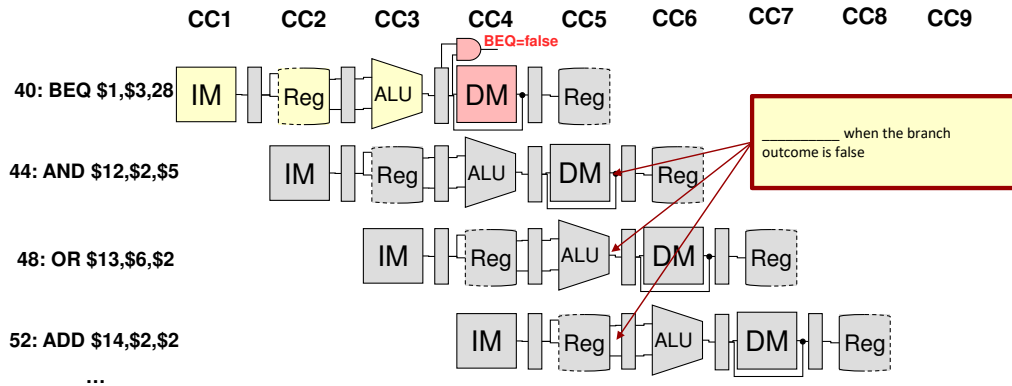
Option 2: Flushing

- Option 2:** Pipeline assumes sequential execution by default. Optimistically assume sequential execution. Since the incorrectly fetched instructions are still in stages [IF, ID, EX] that do not _____ (write a register or memory) they can be safely flushed. Let us add support for this flushing...

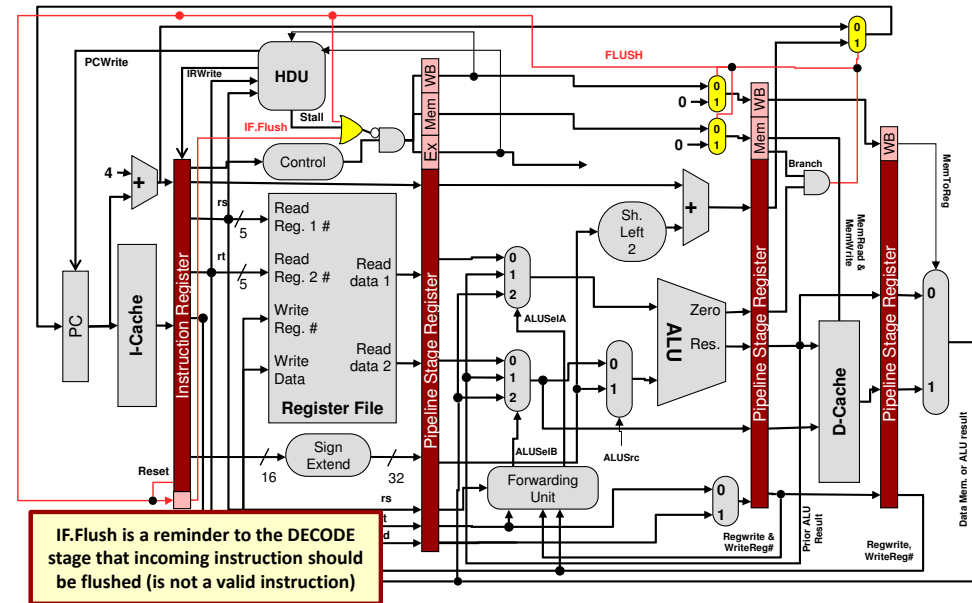


Option 2: Flushing

- Option 2:** Pipeline assumes sequential execution by default. Optimistically assume sequential execution. Since the incorrectly fetched instructions are still in stages [IF, ID, EX] that do not alter processor state (write a register or memory) they can be safely flushed. Let us add support for this flushing...



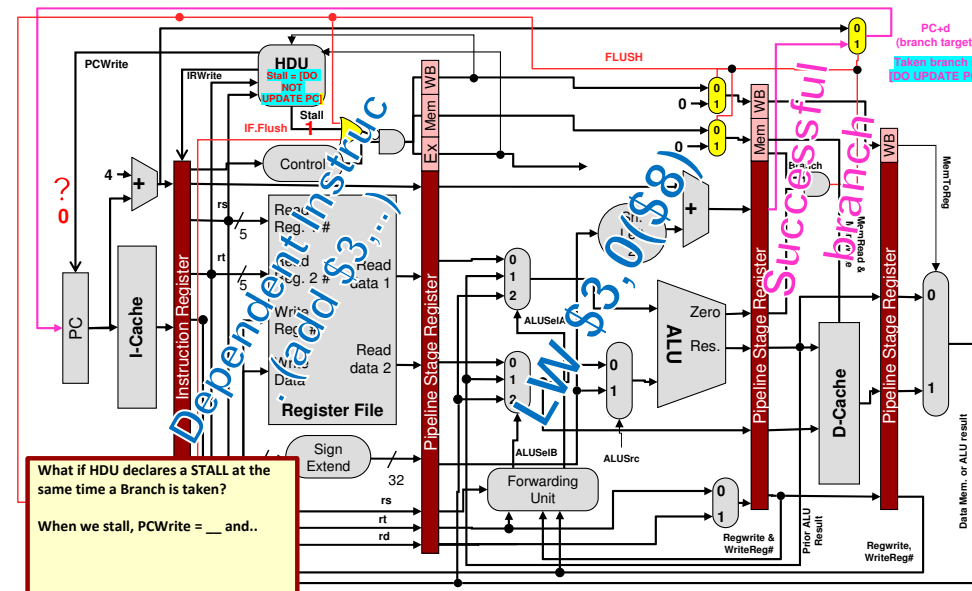
Late Branch Determination



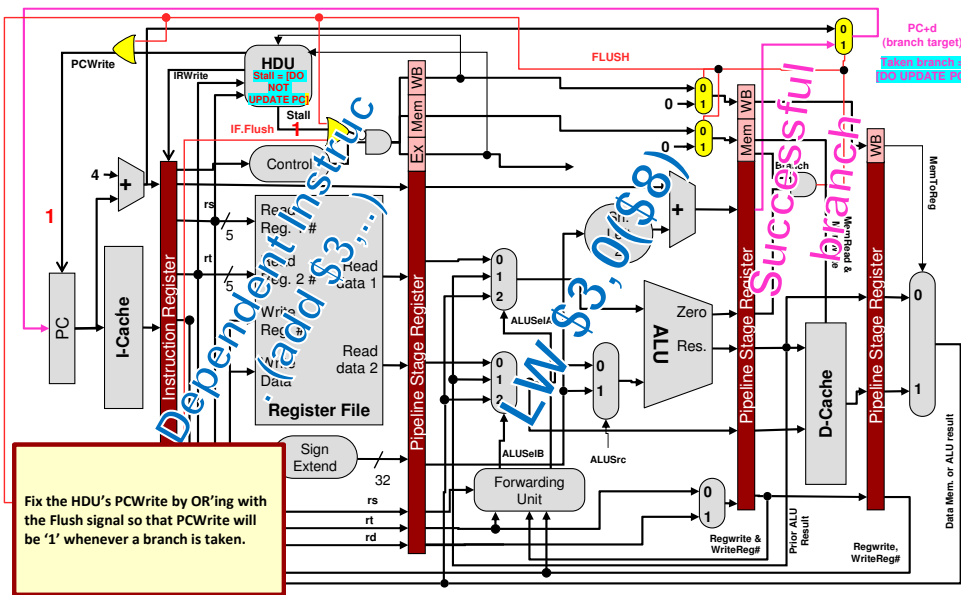
Flushing Strategy

- To flush we merely override the pipeline control signals to _____ similar to the stall logic
 - _____ can be re-used and triggered by a successful branch (Branch AND ALUZero = 1)
 - Stalling only dealt with ID and subsequent stages, not _____ stage
 - Successful branch requires that the instruction in IF be _____, but on the next cycle how will the _____ stage know that the bits in the _____ register are not a _____ instruction but a _____ instruction
- When a branch outcome is true we will...
 - Zero out the control signals in the ID,EX,MEM stages
 - Set a control bit in the _____ register that will tell the _____ stage on the _____ cycle that the instruction is INVALID

Late Branch Determination



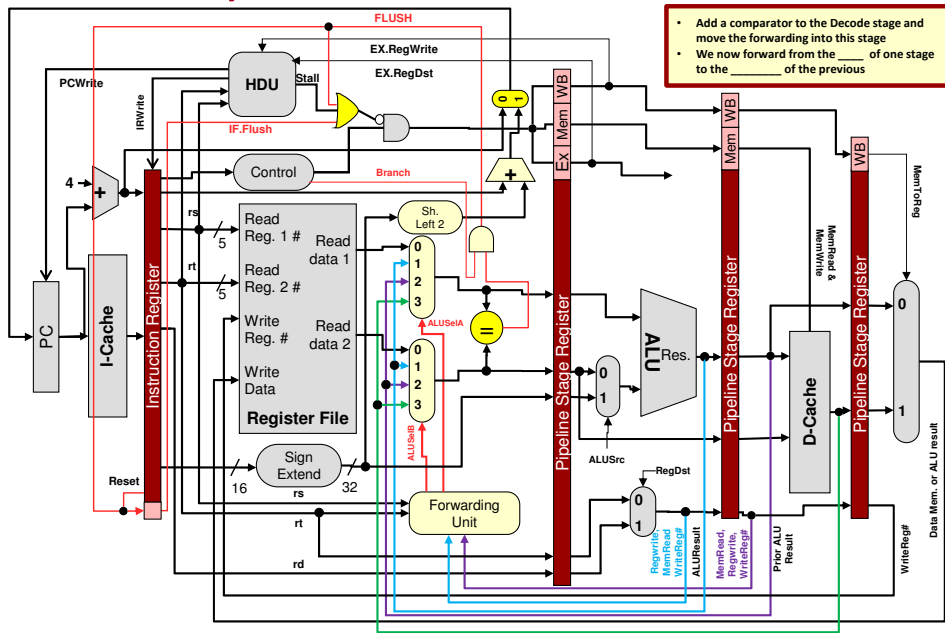
Late Branch Determination



Early Branch Determination

- The stage distance between and and determines how many instructions are flushed (i.e. the branch **penalty**)
 - Again, branch penalty is the number of instructions/clock cycles that are wasted when a branch is taken
- If we can determine the branch **outcome** and **target** computation earlier, we can this penalty
- Observation: All necessary information for both branch outcome and target computation are available (late) in the stage
 - Move comparison and PC+disp. operations to the stage
 - Requires moving since branch instructions may need data from later in the pipe.

Early Branch Determination



Early Determination w/ Predict NT

```

BEQ $a0,$a1,L1 (NT)
L2: ADD $s1,$t1,$t2
    SUB $t3,$t0,$s0
    OR  $s0,$t6,$t7
    BNE $s0,$s1,L2 (T)
L1: AND $t3,$t6,$t7
    SW  $t5,0($s1)
    LW  $s2,0($s5)
    
```

	Fetch (IF)	Decode (ID)	Exec. (EX)	Mem. (ME)	WB
C1	BEQ				
C2	ADD	BEQ			
C3	SUB	ADD	BEQ		
C4	OR	SUB	ADD	BEQ	
C5	BNE	OR	SUB	ADD	BEQ
C6					
C7					
C8					
C9					
C10					

Using early determination & predict NT keeps the pipeline full when we are correct and has a instruction penalty for our 5-stage pipeline

Branch Delay Slots

- Problem: After a branch we fetch instructions that we are not sure should be executed
- Idea: Find an instruction(s) that should _____ be executed (independent of whether branch is ___ or ___), move them to directly after the branch, and have HW just let them be _____ no matter what the branch outcome is
- **Branch delay slot(s)** = _____ that the HW will execute after a branch and not flush
 - Assuming early branch determination (i.e. in decode), only need _____ delay slot

```
x = x + y;
y--;
if(x < 5) {
    dat[i] = a;
    a = 0;
}
else {
    dat[i] = b;
    b = 0;
}
i++;
```

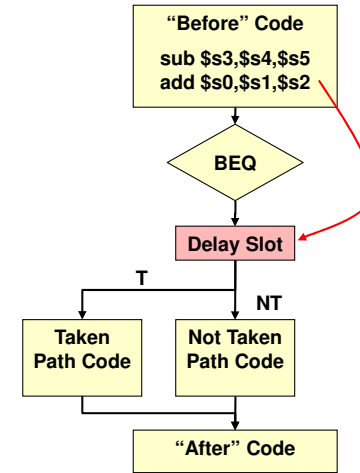
Consider the code above. What lines of code are guaranteed to execute regardless of whether the if or else execute?

Branch Delay Slot Example

```
sub $s3,$s4,$s5
add $s0,$s1,$s2
beq $s3,$t8, ELSE
delay slot instruc.

// if code
b END
// else code
END:
// after code
```

Assume a single instruction delay slot (as with our updated early determination pipeline)



Flowchart perspective of the delay slot

```
sub $s3,$s4,$s5
beq $s3,$t8, ELSE
add $s0,$s1,$s2
...
```

Move an ALWAYS executed instruction (the "add" from above) down into the delay slot and let it execute no matter what

What if sub was instead: lw \$s3,0(\$s4)? Would that change the performance? Yes

Implementing Branch Delay Slots

- HW will define the number of branch delay slots (usually a small number...1 or 2)
- Compiler will be responsible for arranging instructions to fill the delay slots
 - Must find instructions that the branch does NOT DEPEND on
 - If no instructions can be rearranged, can always insert _____ instructions and just waste those cycles

```
sub $s3,$s4,$s5
add $s0,$s1,$s2
beq $s3,$t8, NEXT
delay slot instruc.
...
```

Cannot move 'sub' into delay slot because beq needs the \$s3 value generated by it

```
sub $s3,$s4,$s5
add $t8,$s1,$s2
beq $s3,$t8, NEXT
_____
...
```

If no instruction can be found a '_____' can be inserted by the compiler

Early Determination w/ Delay Slot

```
XOR $s1,$s1,$s1
L2: ADD $s1,$t1,$t2
SUB $t3,$t0,$s6
OR $s0,$t6,$t7
BNE $s0,$s1,L2 (T,NT)
L1: AND $t3,$t6,$t7
SW $t5,0($s1)
LW $s2,0($s5)
```

	Fetch (IF)	Decode (ID)	Exec. (EX)	Mem. (ME)	WB
C1	XOR				
C2					
C3					
C4					
C5					
C6					
C7					
C8					
C9					
C10					

By scheduling the delay slot with an earlier instruction we incur no stalls/bubbles and don't have to "predict" the branch

How Good is the Compiler?

- Source: Hennessey and Patterson, "Computer Architecture – A Quantitative Approach", 2nd Ed. Pg. 169
- How many delay slots should be use?
 - While delay slots seem to improve performance, the benefit depends on the compiler's ability to fill them with useful instructions
 - One of more NOP's in the delay slots but increase the instruction count

# of Delay Slots	Compiler Fills #Useful + #NOPs	Loss of Cycles if taken	Loss of Cycles if not taken	Assume 60%Taken + 40% Not Taken Loss of Cycles	Compiler filling prob.	Loss of cycles (Expectation)	Instruction increasing factor
0		3	0	$3 * 0.6 + 0 * 0.4 = 1.8$	100%	1.8	1
1	1 Use + 0 NOP				65%	1.55	1.35
	0 Use + 1 NOP				35%		
2	2 Use + 0 NOP				40%	1.55	1.95
	1 Use + 1 NOP				25%		
	0 Use + 2 NOP				35%		
3	3 Use + 0 NOP				12%	1.83	2.83
	2 Use + 1 NOP				28%		
	1 Use + 2 NOP				25%		
	0 Use + 3 NOP				35%		

Other Delay Slots?

- Recall that a LW followed by a dependent instruction requires our HDU logic to insert 1 bubble (stall for 1 cycle)
- The MIPS ISA could "declare" a delay slot...
- ...This means the compiler _____ schedule a dependent instruction into the delay slot after a LW
 - If necessary compiler can follow the LW with a 'nop'
- If the ISA declares a LW delay slot do we need the HDU?

Example

- Compile the following code snippet on CompilerExplorer (<https://godbolt.org/>) with:
 - MIPS gcc 13.1.0 or higher
 - **-O0** flag then **-O1**, then **-O3**
 - Then try to increase N to **100**

```
#include <iostream>
#include <string>

using namespace std;
#define N 100
int dat[N];
int main()
{
    for(int i=0; i < N; i++){
        dat[i] = i;
    }
}
```

BACKUP

Late Branch Determination w/ HDU fix

