Speedup = exectime_old / exectime_new = 1 / (1 – frac_en) + (frac_en / speedup_en)
exectime_new = exectime_old * [(1 – frac_en) + frac_en / speedup_en]
CPU time = cpu clock cycles for program / clock rate
CPU time = cpu clock cycles for program * clock cycle time
inst / prog * clock cycles / inst * seconds / clock cycle = seconds / program = CPU time
CPI stuff with different instructions is just a weighted average
Speedup = CPU time_orig / CPU time_new
antidepedence (war) so that i reads. output dependence (waw). raw (true data dependence)
2-bit prediction scheme needs to be wrong twice to change its mind
reorder-buffer holds result until instruction commits**
branch-target buffers rule. Branch penalty = penalty * (probably in buffer, but not taken + branch not in buffer, but taken)
pipeline cpi = ideal + structural stalls + data stalls + control stalls
pipeline speedup = avg inst time without / avg inst time with
avg inst time = clock cycle * avg CPI
clock cycle pipelined = clock cycle normal / pipe depth.
Pipe depth = clock cycle unpipe / clock cycle piped
speedup from pipeline = [1 / (1 + pipeline stall cycles per inst)] * pipe depth
if 2 two instructions need to use memory with 1 bus you'll have to stall
pipeline speedup = depth / (1 + branch_freq * branch penalty /* using branch delay slot */
pipeline stall cycles from branches = branch_freq * branch penalty
pipeline speedup = depth / 1 + pipeline stall cycles per instruction
branch penalties: CPI = (1 – branch %) * non-branch CPI + branch% * branch CPI
CPI = (1-branch%) * 1 + branch% * (1 + penalty)
CPI = 1 + (branch% * penalty)
penalty = (not taken% * not taken cost) + (taken% * taken cost)
remember there is no branch penalty on the times you can usefully use delay slot

Register	Add R3, R4	Regs[R3] = Regs[R3] + Regs[R4]
Immediate	Add R4, #3	Regs[R4] = Regs[R4] + 3
Displacement	Add R4, 100(R1)	Regs[R4] = Regs[R4] + Mem[100+Regs[R1]]
Register Indirect	Add R4, (R1)	Regs[R4] = Regs[R4] + Mem[Regs[R1]]

Chip Die Equations: