Empirical Summary on Optimizations

Cost-benefit analysis on implementing different optimizations

Note: empirical data only, may not be accurate

Optimization	Explain	Performance Speedup	Implementation Difficulty
CSE Common Subexpr Elim	See lecture notes	Negative	Medium
CP Copy Propagation	See lecture notes	Low	Easy
CSE + CP	See lecture notes	Low	Trivial
Constant Folding	Compute any expressions with known values at compile time	Low	Easy
DSE Dead Store Elimination	Remove variables whose values are never used	Low	Easy
DCE Dead Code Elimination	Remove instructions that can never be executed	Low	Easy
CF + DSE + DCE	Combining these optimizations can reveal more instructions that ultimately have no effect on the output	Low	Trivial
Loop Invariant Extraction	See lecture notes	Medium	Medium
Register Allocation (Graph coloring)	See lecture notes	High	Hard
Register Allocation (Heuristic + Greedy + brute force)	For each variable try to assign it to a register and test for conflicts	High	Medium

Optimization	Explain	Performance Speedup	Implementation Difficulty
Stack Allocation	Similar to Register Allocation, use the least amount of stack space to fit the rest of variables by analyzing their lifetime	Medium	Medium
Stack Coalescing	Combining all push/pop’s into one single stack pointer adjustment	Medium	Easy
Omit Frame (Base) Pointer	Use rbp as another scratch register (You can just use rsp to manage the stack)	Low	Easy
Bounds Check Elim	Derive the possible range of variables, and eliminate bound checks if the variable cannot be out of bounds in theory	Medium	Hard
Instruction Scheduling	See lecture notes	Medium	Hard
Loop Unrolling	Unroll a few iteration of the loop body to maximize the utilization of the out-of-order execution engine	Medium-High?	Very Hard
Function Inline	Substitute the function call using the arguments	High	Very Hard
Vectorization	Utilize the Single Instruction Multiple Data capability of the CPU	High	Extremely Hard
Parallelization	Utilize multiple cores on the CPU for parallelizable tasks	High	Hard

Optimization	Explain
Smaller optimizations	
Instruction selection	Select faster instructions (sequence) to do the same operations.
Conditional optimizations	Conditional instructions are status-flag based, some test/cmp instruction may be unnecessary
Tail call optimization	If the last statement in a function is a function call, the callee can reuse the same stack frame of the caller.
μop fusion	Use unsigned comparison in simple loops such as `for (i = 0; i < 100; i += 1)`
Branch prediction	Put basic block of the unusual path away from the usual path.
Branch/Jump elimination	Use conditional move whenever possible (e.g. a branch that only sets one variable), merge two basic blocks connected by unconditional jump
Alignment	Data, Functions, and any Jump targets should be aligned to cache line (16-byte)