CSci 5980/8980
Manual and Automated Binary Reverse Engineering
Day 3: x86 Conditions and Control

Stephen McCamant
University of Minnesota

Outline

x86 conditions

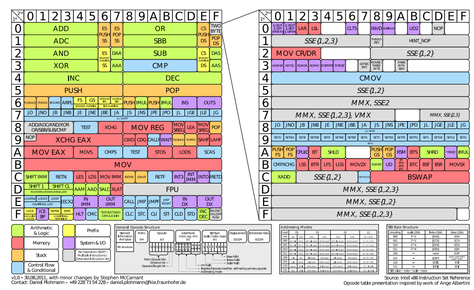
Machine code branching

Machine code loops

Conditional branches

- Basic control-flow instruction: go to another address if a condition is true
- Implements `if`, `while`, and more, structuring code as a graph
- x86 supports several conditions that are part of the "jCC" instruction, or a few other families of instructions

x86 opcode map



Conditions used with comparison

- Signed comparison: `l` $<$, `le` $\leq$, `g` $>$, `ge` $\geq$
- Unsigned comparison: `b` $<$, `be` $\leq$, `a` $>$, `ae` $\geq$ ("below" and "above")
- Equalty `e`
- Every condition can be negated with `n`

Conditions from single flags

- `z`, `s`, `c`, `o`, and `p` for ZF, SF, CF OF, and PF
- `s` means negative, `ns` non-negative
- `p` and `np` are also `pe` "parity even" and `po` "parity odd"

Condition synonyms

- Some conditions are synonyms, only 16 in total
- 8 pairs of opposites, negated by the low encoding bit
- Remember comparison is like subtraction, so ordering is related to sign of difference
  - Unsigned below `b` is the same as unsigned carry `c`
  - Signed less is (`s` XOR `o`)
  - `e` $=$ `z`
- `nl` $=$ `ge`, etc.

Test conditions

- `and`, `or`, and `xor` clear OF and CF, and set ZF, SF, and PF based on the result
- `test` is like `and` but only sets the flags discarding the result
- Checking `nz` after `test` is like `if (x & mask)` in C
- `test` a register against itself is the fastest way to check for zero or negative

Classifying jumps

- Direct jump: target(s) specified in code
- Indirect jump: target selected from runtime data like register or memory contents
- Conditional jump: target differs based on a condition
- The plain words "jump" and "branch" are similar, but usage differs as to which they cover

Short jumps

- `0xeb` plus a 1-byte offset is an unconditional jump
- `0x7[0-f]` plus a 1-byte offset is a conditional jump
- Offset is signed, and interpreted relative to the location of the next instruction
  - `eb fe` is an infinite loop
- Commonly used for `if` (usually positive offsets) and loops (usually negative offsets)

32-bit jumps

- `0xe9` plus a 4-byte offset is an unconditional jump
- `0x0f 0x8[0-f]` plus a 4-byte offset is a conditional jump
- Offset is interpreted relative to the location of the next instruction
  - `e9 fb ff ff ff` is an infinite loop
- Offset is still 4 bytes in 64-bit mode, sign extended
  - Code bigger than 2GB would need other tricks

Conditional moves

- `cmovCC` (`0x0f 0x4[0-f]`) is a 32/16/64-bit move from register or memory into a register
- But, the move only happens if the condition is true; otherwise nothing happens
- Useful for making decisions without changing control-flow

setCC

- `setCC` (`0x0f 0x9[0-f]`) sets a byte to 1 if a condition is true, 0 otherwise
- Like the behavior of C comparisons in the rare case of storing them to a variable
- But, the lack of zero-extension is somewhat inconvenient

Indirect jumps

- `0xff`/4 is a jump instruction where the target comes from a register or memory
- In AT&T syntax, operand prefixed with `*`, like `jmp *%eax`
- Most commonly used for jump tables (q.v.)

Calls and returns

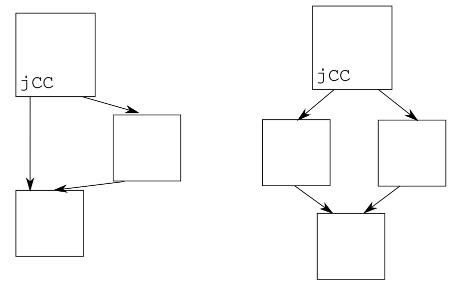
- A call is like a jump, but also pushes the address of the next instruction on the stack
  - `0xe8` with a 4-byte offset is a direct call
  - `0xff`/2 is an indirect call, commonly used for C function pointers
- Return `ret` (`0xc3`) is an indirect jump that pops its address from the stack

Outline

CFGs, basic blocks

- It is useful to think of machine code in a graph structure, called a control-flow graph
- A node in a CFG is a group of adjacent instructions called a basic block:
  - The only jumps into a basic block are to the first insn
  - The only jumps out of a basic block are from the last insn
  - I.e., a basic block always executes as a unit
- Edges between blocks represent possible jumps

CFGs for simple branches



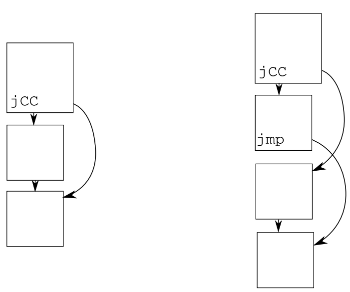
Domination relations

- Basic block a dominates basic block b if every path to b passes through a first; strictly dominates if $a \neq b$
- The immediate dominator a of b is the unique "closest" dominator
  - a strictly dominates b, but there is no a' where a strictly dominates a' and a' strictly dominates b

Post-dominators

- Basic block b post-dominates a if every path through a also passes through b later
- Strict and immediate versions are analogous
- The immediate post-dominator of a branch is the block where execution "reconverges"

Linearizing a CFG to code



Branches with jCC

- The most general way to compile an `if` or `if-else` statement is with a conditional jump
- Note that the condition is inverted compared to the way it's written in source code
  - In simple `if`, condition to skip to the end
  - In `if-else`, condition to skip to else block
- Also need an unconditional jump to skip the else block

Compound conditions

- Logical `&&` and `||` usually compile to more conditional jumps
- `if (A && B) S` $\longrightarrow$ `if (A) { if (B) S }`
- `if (A || B) S` $\longrightarrow$ `if (A) S else if (B) S`
  - But only one copy of S needed

Branches with conditional moves

- If the branch sides are simple and have limited side effects, straight-line code with a conditional move may be faster
- Intuitively, though not strictly, like `if` versus `?:` in C
  - Actually C's `?:` short-circuits too
- Benefits on modern CPU architectures:
  - Low cost to execute both sides (e.g. in parallel)
  - High cost of branch misprediction

Many-way branching

- How about choosing between among many options, like a C `switch`?
- One option is to use a lot of 2-way branches
  - For a switch, a balanced binary-search-like tree is better than a long if-else-if chain
- For a sufficiently large and dense choice, using an indirect jump is usually faster

Computed jump

- Potentially, could space code equally and directly compute a jump target
- But this is rare, including because it would need special assembler support

Jump table

- More common approach is an array of jump targets, indexed like an array
- Usually also has a bounds check
- Jump tables are a common kind of data to be intermixed with code, which can be a challenge for disassembly

Outline

x86 conditions

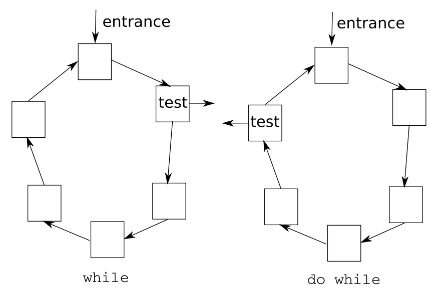
Machine code branching

Machine code loops

Loops

- The source-code concept of a loop corresponds to a cycle in the CFG
- In C, `while` and `do while` loops differ in whether the check comes before or after the body
- `for` loops are syntactic sugar for `while` loops

CFG loop patterns



Rotating loop intuition

- Since a loop is a cycle, it stays basically the same when the parts are rotated
  - But must still keep entrances and exits at the right places
- In source, `break` can exit anywhere
- Source loops usually enter at the start
  - Inelegant alternatives: `goto`, code duplication, first iteration flag

Loop optimizations

- Some of the most interesting compiler optimizations transform loops
  - Sweet spot of valuable but not too hard
- Undo these optimizations in reverse engineering when it makes the code more natural

Induction variables

- An induction variable has a value that is a linear function of the loop iteration count
- Inefficient: counter and multiplication
- Efficient: add constant on each iteration
- E.g., equivalence of array indexing and pointer traversal

Tail-call elimination

- A tail call is a recursive call that is the last operation on an execution path
- The call and return can be replaced with a jump back to the function beginning
- Considered critical for functional languages, not as important for C

More loop optimizations

- Count up $\rightarrow$ count down
- Merge adjacent loops
- Unroll groups of iterations, or all of them