

Check-in #33

Give an example of a snippet of x64 code that benefits from two peephole optimizations

je LBL-1
~~addq \$0, %rax~~
LBL-1: retq

pushq \$4, %rdi
popq %rdi

⇒ LBL-1: retq

pushq \$1, %rax
popq %rax
pushq \$4, %rdi
popq %rdi
⇒ movq \$4, %rdi

LBL-1

Flipped Wednesday



Quiz 3

Quiz 3 – Question 1

Many compilers, including lexic, uses multiple intermediate representations. Describe why each of these IRs are used.

Quiz 3 – Question 2

Write x64 code to compute the sum of an array NUMS and exit the program with the sum as the exit value. You may assume that the length of the array is stored in LEN. That is, complete the following assembly program:

```
.globl _start
.data
NUMS: .quad 6, 3, 2, ...
LEN: .quad ?
.text
_start :
( your code here)
```

Handwritten assembly code and annotations:

```
movq $0, %rdi
movq $NUMS, %rcx
movq (LEN), %rdx

LOOP: cmpq $0, %rdx
     je AFTER
     addq $8, %rcx
     jmp Loop

AFTER: subq $1, %rdx
      movq 0(%rcx), %rax
      addq %rax, %rdi
      movq $60, %rax
      syscall
```

Annotations:

- `LOOP:` is written in green.
- `addq $8, %rcx` is written in purple.
- `jmp Loop` is written in purple.
- `AFTER:` is written in green.
- A purple arrow points from the `jmp Loop` instruction to the `AFTER:` label.

Quiz 3 – Question 3

```
wacky : (arg1 : int , arg2 : bool) int {  
  a : int = arg1 - 1;  
  if (arg2) {  
    while (arg1 < 2) {  
      arg1 = wacky(a, arg2);  
    }  
    L3: return arg1;  
  }  
  return a - 1;  
}
```

Convert the above code to 3AC in the manner that a non-optimizing compiler would do, in the fashion discussed in class.

fn-wacky: enter wacky
getarg 1, [arg 1]
getarg 2, [arg 2]
[a] := SUB64 [arg 1], 1
ifz [arg 2] goto L2
[tmp 1] := [arg 1] L764 2
ifz [tmp 1] goto L3
getarg 1, [a]
getarg 2, [arg 2]
call wacky
getret [arg 1]
goto Loop
Loop:
[tmp 2] := [a] sub:64 1
L2: setret [tmp 2]
goto fn-end-wacky
fn-end-wacky: leave wacky
L3: setret [arg 1]
goto fn-end-wacky
Loop:

L2: ~~setret [tmp 2]~~
goto fn-end-wacky

fn-end-wacky: leave wacky

Quiz 3 – Question 4

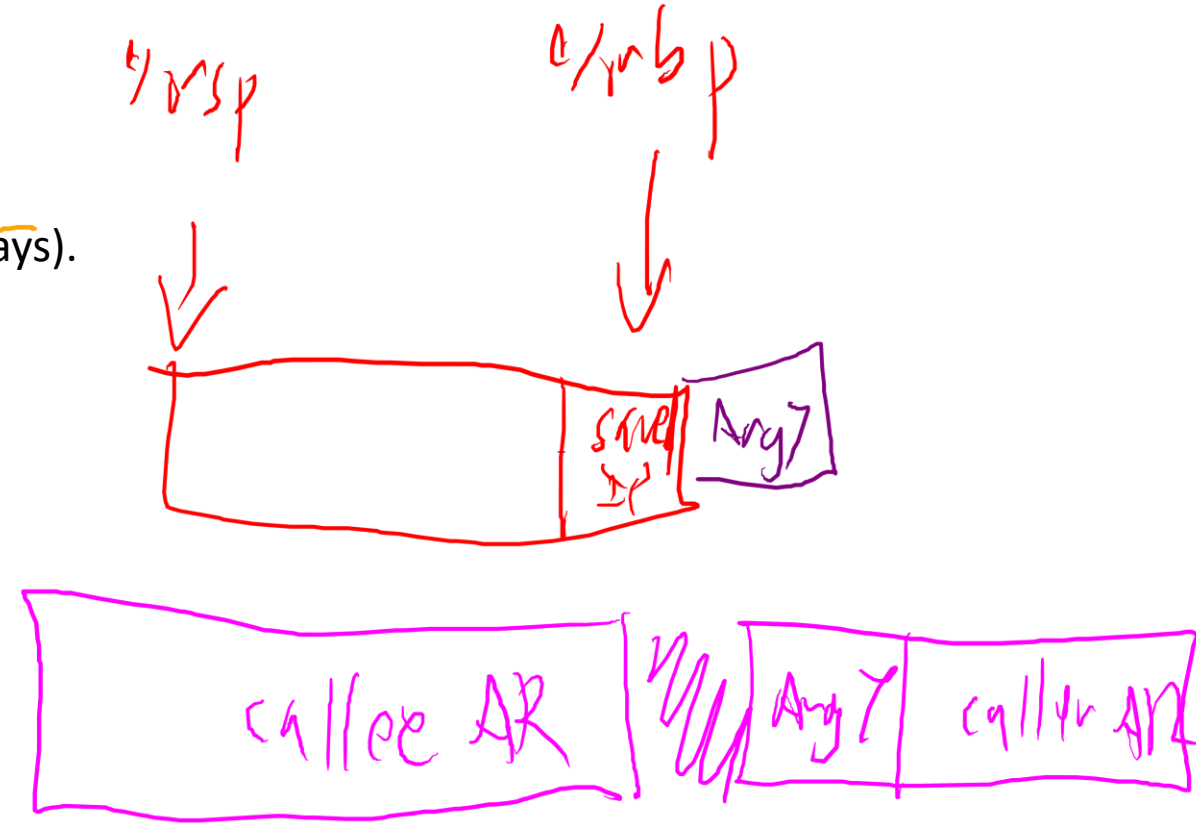
What is a runtime environment?

What is the runtime environment for the Levi language?

Quiz 3 – Question 5

Imagine the following code is intended to represent an x64 function call. It violates the System V ABI in (at least two ways).
Point out two violations

```
movq $20, %rdi      # set argument 1
movq $30, %rsi      # set argument 2
movq $40, %rdx      # set argument 3
movq $50, %rcx      # set argument 4
movq $90, %r8       # set argument 5
movq $12, %r9       # set argument 6
pushq $276         # set argument 7
callq my_callee     # make the call
movq %rbx, -32(%rbp) # retrieve the return
nop                 #(end of the call)
```



W7 – Question 2

Convert the following function into 3AC

```
int v(int a){
    while (a < 2){
        while (a < 3){
            a++;
        }
        a++;
    }
    return a;
}
```

```
fn_v: enter v
      getarg 1, [a]
LBL_1: [tmp1] := [a] LT64 2
      ifz [tmp1] goto LBL_2
LBL_3: [tmp2] := [a] LT64 3
      ifz [tmp2] goto LBL_4
      [a] := [a] ADD64 1
      goto LBL_3
LBL_4: nop
      [a] := [a] ADD64 1
LBL_2: nop
      setret [a]
      goto end_fn_v
end_fn_a: leave v
```

W7 – Question 3

Convert the 3AC procedure into source code

```
fn_k: enter k
      getarg 1, [b]
      [i] = [b]
lbl_1: [t] = [i] LT64 10
      ifz [t] goto lbl_2
      [i] = [i] ADD64 1
      WRITE [i]
      goto lbl_1
      lbl_2: nop
fn_end_k: leave k
```

```
k : (b : int) void {
    i : int;
    i = b;
    while (i < 10){
        i++;
        out << I;
    }
}
```

W7 – Question 4

Assume a language that allows for pass-by-reference or pass-by-value parameters. What would the 3AC code look like for a pass-by-reference call? Illustrate with an example.

Don't use the brackets around the variable (which indicate a memory lookup) in the generated setarg / getarg

```
void foo(int& arg){          fn_foo: enter foo
    arg = 3;                  getarg 1, arg
}                             end_fn_foo: leave foo

int main(){                 fn_main: enter main
    int p;                   setarg 1, p
    foo(p);                  end_fn_main: leave main
}
```