ILOC Reference
CS 432 – Fall 2022

Form	Op1	Op2	Op3	Comment
Integer Arithmetic				
add op1, op2 => op3	reg	reg	reg	addition
sub op1, op2 => op3	reg	reg	reg	subtraction
mult op1, op2 => op3	reg	reg	reg	multiplication
div op1, op2 => op3	reg	reg	reg	division
addI op1, op2 => op3	reg	imm	reg	addition w/ constant
multI op1, op2 => op3	reg	imm	reg	multiplication w/ constant
neg op1 => op2	reg	reg		arithmetic negation
Boolean Arithmetic				
and op1, op2 => op3	reg	reg	reg	boolean AND
or op1, op2 => op3	reg	reg	reg	boolean OR
not op1 => op2	reg	reg		boolean NOT
Data Movement				
i2i op1 => op2	reg	reg		register copy
loadI op1 => op2	imm	reg		load integer constant
load [op1] => op2	reg	reg		load from address
loadAI [op1+op2] => op3	reg	imm	reg	load from base + immediate
loadAO [op1+op2] => op3	reg	reg	reg	load from base + offset
store op1 => [op2]	reg	reg		store to address
storeAI op1 => [op2+op3]	reg	reg	imm	store to base + immediate
storeAO op1 => [op2+op3]	reg	reg	reg	store to base + offset
push op1	reg			push onto stack
pop op1	reg			pop from stack
Comparison				
cmp_LT op1, op2 => op3	reg	reg	reg	less-than comparison
cmp_LE op1, op2 => op3	reg	reg	reg	less-than-or-equal-to comparison
cmp_EQ op1, op2 => op3	reg	reg	reg	equality comparison
cmp_GE op1, op2 => op3	reg	reg	reg	greater-than-or-equal-to comparison
cmp_GT op1, op2 => op3	reg	reg	reg	greater-than comparison
cmp_NE op1, op2 => op3	reg	reg	reg	inequality comparison
Control Flow				
label (“op1:”)	lbl			control flow label
jump op1	lbl			unconditional branch
cbr op1 => op2, op3	reg	lbl	lbl	conditional branch
call	fun			call function
return				return to caller
Miscellaneous				
print	reg			print integer to standard out
print	str			print string to standard out
nop				no-op (do nothing)
phi	reg	reg	reg	φ-function (for SSA only)

Op	Meaning
reg	register (int or bool)
imm	immediate (int constant)
lbl	jump label
fun	call label
str	string

ILOC	**Y86-64**	**similar for:**
	.pos 0 code	
	_start:	
	irmovq 0xf00, %rsp	
	call main	
BOILERPLATE	halt	
	rrmovq r1, r3	
add r1, r2 => r3	addq r2, r3	sub
add r1, r2 => r1	addq r2, r1	sub
add r1, r2 => r2	addq r1, r2	sub
	rrmovq r1, t1	
	irmovq $1, t2	
	irmovq $0, r3	
	loop:	
	andq t1, t1	
	je done	
	addq r2, r3	
	subq t2, t1	
	jmp loop	
mult r1, r2 => r3	done:	
div r1, r2 => r3		
	rrmovq r1, r2	
	irmovq imm1, t1	
addI r1, imm1 => r2	addq t1, r2	multI
	irmovq imm1, t1	
addI r1, imm1 => r1	addq t1, r1	multI
	irmovq $0, r2	
neg r1 => r2	subq r1, r2	
	rrmovq r1, r3	
and r1, r2 => r3	andq r2, r3	
and r1, r2 => r1	andq r2, r1	
and r1, r2 => r2	andq r1, r2	
	rrmovq r1, r3	
	xorq r2, r3	
	rrmovq r1, t1	
	andq r2, t1	
or r1, r2 => r3	addq t1, r3	
	rrmovq r1, r2	
	irmovq $0xFFFFFFFFFFFFFFFF, t1	
not r1 => r2	xorq t1, r2	
i2i r1 => r2	rrmovq r1, r2	
loadI imm1 => r1	irmovq imm1, r1	
loadS sym1 => r1	irmovq sym1, r1	
load [r1] => r2	mrmovq (r1), r2	store
loadAI [r1+imm1] => r2	mrmovq imm1(r1), r2	storeAI
	rrmovq r1, t1	
	addq r2, t1	
loadAO [r1+r2] => r3	mrmovq (t1), r3	storeAO

```
                        rrmovq r2, t1
                        subq r1, t1
                        jl true
                        irmovq $0, r3
                        jmp done
                      true:
                        irmovq $1, r3
                      done:

                      – OR (w/ cmov) –

                        rrmovq r2, t1                   cmp_LE
                        irmovq $0, r3                   cmp_EQ
                        irmovq $1, t2                   cmp_GE
                        subq r1, t1                     cmp_GT
cmp_LT r1, r2 => r3     cmovl t2, r3                    cmp_NE
                                                        cmp_LE, cbr
                        rrmovq r2, t1                   cmp_EQ, cbr
                        subq r1, t1                     cmp_GE, cbr
cmp_LT r1, r2 => r3     jl l1                           cmp_GT, cbr
cbr r3 => l1, l2        jmp l2                          cmp_NE, cbr
label l1                l1:
jump l1                   jmp l1
                          andq r1, r1
                          jne l2
cbr r1 => l1, l2          jmp l1
param r1                  pushq r1
                          call fun
                          pushq %rbp
call fun                  rrmovq %rsp, %rbp
                          rrmovq %rbp, %rsp
                          popq %rbp
return                    ret
print r1                (no equivalent—use iotrap?)
print str               (no equivalent)
nop                       nop
phi                     (no equivalent needed)
```