X64 Reference

- movq ⟨opd$_1$⟩ ⟨opd$_2$⟩ - *Copy the 8-byte value of opd$_1$ into opd$_2$*
- addq ⟨opd$_1$⟩ ⟨opd$_2$⟩ - *Put the result of opd$_2$ + opd$_1$ into opd$_2$*
- subq ⟨opd$_1$⟩ ⟨opd$_2$⟩ - *Put the result of opd$_2$ − opd$_1$ into opd$_2$*
- imulq ⟨opd$_1$⟩ - *Put the result of %rax * opd$_1$ into the octoword %rdx:%rax*
- callq ⟨lbl⟩ - *Stack (push) the address of the next instruction, move %rip to the address ⟨lbl⟩*
- retq - *Unstack (pop) into %rip*
- xorq ⟨opd$_1$⟩ ⟨opd$_2$⟩ - *Put the result of ⟨opd$_2$⟩ XOR ⟨opd$_1$⟩ into opd$_2$*
- negq ⟨opd⟩ - *Put the 2's complement negation of ⟨opd⟩ into ⟨opd⟩*
- notq ⟨opd⟩ - *Flip all bits of ⟨opd⟩*
- jmp ⟨lbl⟩ - *jump to ⟨lbl⟩*
- cmpq ⟨opd$_1$⟩ ⟨opd$_2$⟩ - *Set rflags according to ⟨opd$_2$⟩ - ⟨opd$_1$⟩*
- je ⟨lbl⟩ - *jump to ⟨lbl⟩ if rflags indicates a = relation on prior operands*
- jne ⟨lbl⟩ - *jump to ⟨lbl⟩ if rflags indicates a ≠ relation on prior operands*
- jge ⟨lbl⟩ - *jump to ⟨lbl⟩ if rflags indicates a ≥ relation on prior operands*
- jl ⟨lbl⟩ - *jump to ⟨lbl⟩ if rflags indicates a < relation on prior operands*
- jg ⟨lbl⟩ - *jump to ⟨lbl⟩ if rflags indicates a > relation on prior operands*
- jle ⟨lbl⟩ - *jump to ⟨lbl⟩ if rflags indicates a ≤ relation on prior operands*
- sete ⟨opd⟩ - *Set opd to be 1 if rflags indicates that the last compare operation had equal operands, 0 otherwise. ⟨opd⟩ must be a 1-byte register.*
- setg ⟨opd⟩ - *Set opd to be 0 if rflags indicates that the last compare operation had an opd$_2$ less than or equal to its opd$_1$, 1 otherwise. ⟨opd⟩ must be a 1-byte register.*
- setle ⟨opd⟩ - *Set opd to be 0 if rflags indicates that the last compare operation had an opd$_2$ greater than its opd$_1$, 0 otherwise. ⟨opd⟩ must be a 1-byte register.*

Registers

General-purpose registers

- `%rax - %rdx` (lowest 1 byte is %al - %dl)
- `%r8 - %r15` (lowest 1 byte is %r8b - %r15b)
- `%rsi` (lowest 1 byte is %sil)
- `%rdi` (lowest 1 byte is %dil)
- `%rsp` - stack pointer
- `%rbp` - base pointer

Select special-purpose registers

- `%rflags` status flags, stores comparison results
- `%rip` instruction pointer, next address to execute

3AC Reference

List of Pseudoinstructions operating over pseudovariables. It's ok to fudge this a little bit, as long as you don't nest expressions or instructions.

`x := y op z`
Perform a logical, relational, or mathematical operation on y and z, then assign the result to x. You may assume relational and logical operators represent true as 1, false as 0.

`x := y`
Assign the value of pseudovariable y to pseudovariable x

`iftrue x goto L`
If psuedovariable x has the value 1, jump to the program location with label L.

`iffalse x goto L`
If psuedovariable x has the value 0, jump to the program location with label L.

`goto L`
Jump to the program location with label L.

`call p`
Transfer control to the body of function p with any arguments set via the set_arg pseudoinstruction.

`setarg k, x`
Set the kth argument value in caller to x.

`setret x`
Set the return value to x.

`getarg k, x`
Set the kth argument value in callee to x.

`getret x`
Set x to the return value from the last call.

enter ⟨proc⟩
Begin procedure ⟨proc⟩.

leave ⟨proc⟩
End procedure ⟨proc⟩.

`label L` Mark the next instruction as being at label L.

`WRITE x, y` Output the value of x to filesystem handle x.

`READ x, y` Get the value of x from filesystem handle y.