Instrucciones básicas

Creación de variables globales (crea punteros al tipo definido)

@nombre = global tipo[float, i32 , i1] valor o variable

Creación de variables locales (crea punteros al tipo definido)

%nombre = alloca tipo[float, i32 , i1]

Lectura de valores

%nombre = load tipo[float, i32 , i1], tipo[float, i32 , i1]* @variable

Escritura de valores

store tipoOrigen [valorOrigen o variableOrigen], tipoDestino* @variableDestino

Convertir variables

Integer to Float

%vdestino = sitofp i32 %vorigen to float

Float to Integer

% vdestino = fptosi float %vorigen to i32

Operaciones aritmeticas

Integer

%dest = mul tipo[i32] %op1, %op2

%dest = add tipo[i32] %op1, %op2

%dest = sub tipo[i32] %op1, %op2

%dest = sdiv tipo[i32] %op1, %op2

Float

%dest = fmul tipo[i32,float] %op1, %op2

%dest = fadd tipo[i32,float] %op1, %op2

%dest = fsub tipo[i32,float] %op1, %op2

%dest = fdiv tipo[i32,float] %op1, %op2

Comparaciones con salto

Integer

%res = icmp [comparador] [tipo_operandos] %op1, %op2

br i1 %res, label %labelTrue, label %labelFalse

Comparadores

1. `eq`: equal
2. `ne`: not equal
3. `ugt`: unsigned greater than
4. `uge`: unsigned greater or equal
5. `ult`: unsigned less than
6. `ule`: unsigned less or equal
7. `sgt`: signed greater than
8. `sge`: signed greater or equal
9. `slt`: signed less than
10. `sle`: signed less or equal

Float

%res =fcmp [comparador] [tipo_operandos] %op1, %op2

br i1 %res, label %labelTrue, label %labelFalse

Comparadores

1. `false`: no comparison, always returns false
2. `oeq`: ordered and equal
3. `ogt`: ordered and greater than
4. `oge`: ordered and greater than or equal
5. `olt`: ordered and less than
6. `ole`: ordered and less than or equal
7. `one`: ordered and not equal
8. `ord`: ordered (no nans)
9. `ueq`: unordered or equal
10. `ugt`: unordered or greater than
11. `uge`: unordered or greater than or equal
12. `ult`: unordered or less than
13. `ule`: unordered or less than or equal
14. `une`: unordered or not equal
15. `uno`: unordered (either nans)
16. `true`: no comparison, always returns true

Operadores lógicos:

%temp = operador i1 %op1, %op2

operador: and, or, xor

Salto incondicional

br label %labelSalto

Constantes string

Debe indicarse su longitud en la definición y todas deben terminar con \00 (caracter nulo), ejemplo:

@str = private constant [**11** x i8] c"Hola mundo**\00**"

Prints (interface de C)

Al inicio del programa declarar las siguientes funciones y variables globales (\0A es salto de línea)

declare i32 @puts(i8*)

declare i32 @printf(i8*, ...)

@.integer = private constant [4 x i8] c"%d\0A\00"

@.float = private constant [4 x i8] c"%f\0A\00"

Para imprimir un string por ejemplo la variable @str:

%temp = call i32 @puts(i8* getelementptr ([11 x i8], [11 x i8] * @str, i32 0, i32 0))

(el 11 es por la longitud de @str)

Para imprimir enteros o float (ejemplo la variable %mi_num)

%temp= call i32 (i8*, ...) @printf(i8* getelementptr([4 x i8], [4 x i8]* @.float, i32 0, i32 0), float %mi_num)

%temp= call i32 (i8*, ...) @printf(i8* getelementptr([4 x i8], [4 x i8]* @.integer, i32 0, i32 0), i32 %mi_num)

(el 4 es por las longitudes de @integer y @float)

Read

Nota importante, quienes desarrollen el compilador en Windows van a tener que linkear la biblioteca scanf.o provista junto con su programa. De esta manera para generar su ejecutable van a tener que correr los siguientes comandos (program.ll contiene el código generado por su compilador):

clang -c -o program.o program.ll

clang -o program.exe program.o scanf.o

Quienes utilicen linux o mac, no deberían necesitar linkear la biblioteca scanf.o.

Al inicio del programa declarar las siguientes funciones y variables globales (\0A es salto de línea)

declare i32 @scanf(i8* %0, ...)

@int_read_format = unnamed_addr constant [3 x i8] c"%d\00"

@double_read_format = unnamed_addr constant [4 x i8] c"%lf\00"

Para leer enteros:

%dest = alloca i32

%temp = call i32 (i8*, ...) @scanf(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @ int_read_format, i64 0, i64 0), i32* %dest)

Para leer float (se debe leer como double y truncar porque algunos números no se pueden representar como float):

%dest = alloca float

%destaux = alloca double

%temp = call i32 (i8*, ...) @scanf(i8* getelementptr inbounds ([4 x i8], [4 x i8]* @double_read_format, i64 0, i64 0), double* %dest_aux)

%temp_double = load double, double* %dest_aux

%temp_float = fptrunc double %temp_double to float :trucamos double a float

%dest = store float %temp_float, float* %dest

Funciones

define tipo_retorno[i1, i32, float] @nombre_funcion (tipo_par1[i1, i32, float] %nombre_par1, ...){

cuerpo función

}