ejecutar [[Programa: Programa -> Definicion*]]() =	for(Definicion d: Definicion*)
	if(d instanceof DefVariable)
	Ejecutar [[d]]();
	for(Definicion d: Definicion*)
	if(d instanceof DefFuncion)
	Ejecutar [[d]]();

ejecutar[[Asignacion: sentencia -> expresion1 expresion2]]() =	direccion[[expresion1]](); valor[[expresion2]](); GC.convertirA(expresion2.tipo, expresion1.tipo); <STORE> expresion1.tipo.sufijo();

ejecutar [[Escritura: sentencia -> Expresion*]]() =	for(Expresion exp : Expresion*){
	valor [[exp]];
	<OUT> exp.tipo.sufijo;
	}

valor[[Invocacion: expresion -> variable Expresion*]]() =	for(Expresion exp : Expresion*){
	contador = 0;
	valor[[expresion]]();
	GC.convertir(expresion.tipo, (TipoFuncion)
	variable.tipo.parametro(contador).tipo);
	contador++;
	}

ejecutar[[Invocacion: sentencia -> variable Expresion*]]() =	valor[[(Expresion) sentencia]](); if(!(Expresion)sentencia.tipo instanceof TipoVoid) <POP>((Expresion)sentencia).tipo.sufijo;

ejecutar [[Lectura: sentencia -> Expresion*]]() =	for(Expresion exp : Expresion*){
	Direccion[[exp]]();
	<IN> exp.tipo.sufijo;
	<STORE> exp.tipo.sufijo;
	}

ejecutar[[return: sentencia -> expresion]](df: DefFuncion) =	valor[[expresion]]; GC.convertir(expresión.tipo, df.tipo.tipoRetorno); <RET> df.tipo.tipoRetorno.numBytes <,> df.numBytesLocal <,> df.tipo.paramereos.numBytesParam;

ejecutar[[SentenciaIf: sentencia -> expresion sentencias1*
sentencias2*]] =
	int count = GC.getLabels(2);
	valor[[expresion]]();
	GC.convertirA(expresion.tipo, TipoEntero);
	<JZ> <label> count;
	for(Sentencia s: sentencias1*)
	ejecutar[[s]];
	<JMP> <label> count + 1;
	<label> count <:>;
	for(Sentencia s: sentencias1*)
	ejecutar[[s]];
	<label> count + 1 <:>;

ejecutar[[SentenciaWhile: sentencia -> expresion
Sentencia*]]() =
	GC.getLavels(2);
	int count = 0;
	<label> count <:>;
	valor[[sentencia]]();
	GC.convertirA(expresion.tipo, TipoEntero);
	<JZ> <label> count + 1;
	for(Sentencia s : Sentencia*)
	ejecutar[[s]];
	<JMP> <label> count;
	<label> count + 1 <:>

valor[[AccesoArray: expresion1 -> expresion2 expresion3]]() =	direccion[[expresion1]]; <LOAD> expresion1.tipo.sufijo;

direccion [[AccesoArray: expresion1 -> expresion2 expresion3]]() =	direccion[[expresion2]]; <PUSH> expresion1.tipo.nBytes(); valor[[expresion3]]; GC.convertirA(expresion3.tipo, TipoEntero); <MUL> <ADD>
valor[[AccesoCampo: expresion1 -> expresion2 ID]]() =	direccion[[expresion1]](); <LOAD> expresion1.tipo.sufijo;
direccion[[AccesoCampo: expresion1 -> expresion2 ID]]() =	direccion[[expresion2]]; <PUSH> expresion1.tipo.Campo(ID).offset; <ADD>
Valor[[Aritmetica: expresion1 -> expresion2 expresion3]]() =	valor[[expresion2]]; GC.convertirA(expresion2.tipo, expresion1.tipo); valor[[expresion3]]; GC.convertirA(expresion3.tipo, expresion1.tipo); GC.Aritmetrica(expresion1.operador, expresion1.tipo);
Valor[[Cast: expresion1 -> tipo expresion2]]() =	valor[[expresion2]]; GC.convertirA(expresion2.tipo, expresion1.tipo);
Valor[[Compracacion: expresion1 -> expresion2 expresion3]]() =	Tipo mayor = expresion2.tipo.mayor(expresion3.tipo); valor[[expresion2]]; GC.convertirA(expresion2.tipo, mayor); valor[[expresion3]]; GC.convertirA(expresion3.tipo, mayor); GC.Aritmetrica(expresion1.operador, expresion1.tipo);
valor[[LiteralCaracter: expresion -> Cte_Caracter]]() =	<PUSH> expresión.valor.sufijo;
valor[[LiteralEntero: expresion -> Cte_Entera]]() =	<PUSH> expresión.valor.sufijo;
valor[[LiteralReal: expresion -> Cte_Real]]() =	<PUSH> expresión.valor.sufijo;
valor[[Logica: expresion1 -> expresion2 expresion3]]() =	valor[[expresion2]]; valor[[expresion3]]; GC.Logica(expresion1.operador);
valor[[MenosUnario: expresion1 -> expresion2]]() =	valor[[expresion2]]; GC.Logica(expresion1.operador);
valor[[Negacion: expresion1 -> expresion2]]() =	valor[[expresion2]]; <PUSH> -1; GC.convertirA(TipoEntero, expresion1.tipo); <MUL>
valor[[Variable: expresion -> ID]]() =	direccion[expresión]; <LOAD> expresión.tipo.sufijo;

direccion[[Variable: expresión -> ID]]() =	if(expresión.def.ambito == 0)
	  <PUSHA> expresión.def.offset;
	else {
	  <PUSH BP>
	  <PUSH> expresión.def.offset;
	}