Código intermedio – Código final (subprogramas)

En este punto el alumno debe de haber:

- Concluido el análisis semántico
  - Creación de los diferentes ámbitos
  - Declaración de tipos y símbolos en las tablas de sus correspondientes ámbitos
  - Comprobación de tipos:
    - Referencias y expresiones
    - Entre expresiones
    - Invocación de subprogramas
- Generar el código intermedio para las diferentes sentencias y expresiones.
  - Sentencias de asignación
  - Sentencias de control de flujo (IF-THEN-ELSE/WHILE/FOR)
  - Sentencias de entrada/salida (WRITESTRING/WRITEINT/WRITELN)
- **Generar el código intermedio para la invocación de subprogramas**

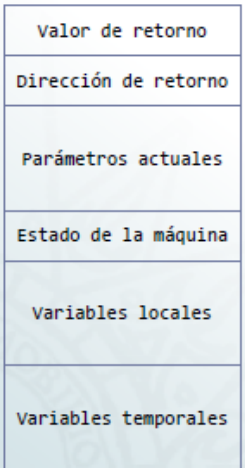
Registro de Activación (RA) es un espacio de memoria reservado, donde cada subprograma tendrá los datos relativos a su valor de retorno, dirección de retorno, parámetros actuales, variables, temporales, etc.

El lenguaje PL1UnedES se basa en un entorno de **ejecución basado en pila** para poder permitir la recursividad en los subprogramas, en concreto solo se dará soporte a la recursividad directa.

Entorno de ejecución estático

En los entornos de ejecución estáticos todas las activaciones de un mismo subprograma coinciden en memoria. Es decir, existe un único registro de activación por cada subprograma, incluido el programa principal (si es que éste no se incluye en la zona de datos estáticos). Esto permite conocer en tiempo de compilación cada direccionamiento pero imposibilita las activaciones recursivas

Diseño del registro de activación



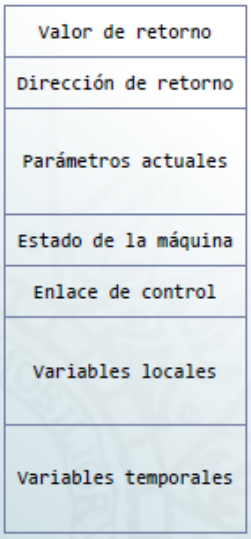
Nombre del campo	Escritor	Lector	Descripción
Valor de retorno	Llamado	Llamante	En funciones, almacena el valor devuelto por la función tras la invocación
Dirección de retorno	llamante	llamado	Se establece la dirección de código a la que debe saltar el llamado tras su invocación
Parámetros actuales	llamante	llamado	Proporciona los parámetros actuales
Estado de la máquina	Llamante	Llamado	Contiene el estado de la máquina (copia de los registros) antes de la llamada para restaurarlos tras su invocación
Variables locales	llamado	llamado	Contiene el valor de las variables locales
Variables temporales	llamado	llamado	Contiene el valor de las variables temporales

Referencia: Javier Vélez Reyes (Generación de código intermedio. Activación de subprogramas)

Entorno de ejecución basado en pila

En los entornos de ejecución dinámicos, cada activación distinta dispone de un registro de activación propio, gestionado a través de una pila. Esto complica la gestión de memoria ya que no todos los enlaces de datos son conocidos en tiempo de compilación pero permite la articulación de activaciones recursivas

Diseño del registro de activación



Nombre del campo	Escritor	Lector	Descripción
Valor de retorno	Llamado	Llamante	En funciones, almacena el valor devuelto por la función tras la invocación
Dirección de retorno	llamante	llamado	Se establece la dirección de código a la que debe saltar el llamado tras su invocación
Parámetros actuales	llamante	llamado	Proporciona los parámetros actuales
Estado de la máquina	Llamante	Llamado	Contiene el estado de la máquina (copia de los registros) antes de la llamada para restaurarlos tras su invocación
Enlace de control	Llamante	Llamado	Contiene un puntero al Registro de activación del subprograma llamante
Variables locales	llamado	llamado	Contiene el valor de las variables locales
Variables temporales	llamado	llamado	Contiene el valor de las variables temporales

Referencia: Javier Vélez Reyes (Generación de código intermedio. Activación de subprogramas)

Referencias no locales

En programas con estructura de bloques anidados (Pascal, Modula-2, Ada), las referencias no locales se refieren a variables locales a algún ámbito accesible desde los RA de los hijos según la regla de anidamiento del ámbito más cercano. En A11, la referencia n es no local ya que está definida localmente en el ámbito A, accesible desde éste

Diseño del registro de activación

Nombre del campo	Escritor	Lector	Descripción
Valor de retorno	Llamado	Llamante	En funciones, almacena el valor devuelto por la función tras la invocación
Dirección de retorno	llamante	llamado	Se establece la dirección de código a la que debe saltar el llamado tras su invocación
Parámetros actuales	llamante	llamado	Proporciona los parámetros actuales
Estado de la máquina	Llamante	Llamado	Contiene el estado de la máquina (copia de los registros) antes de la llamada para restaurarlos tras su invocación
Enlace de control	Llamante	Llamado	Contiene un puntero al Registro de activación del subprograma llamante
Enlace de acceso	Llamante	Llamado	Contiene un puntero al Registro de activación del subprograma anidante (padre)
Variables locales	llamado	llamado	Contiene el valor de las variables locales
Variables temporales	llamado	llamado	Contiene el valor de las variables temporales

Referencia: Javier Vélez Reyes (Generación de código intermedio. Activación de subprogramas)

Registro de Activación

El registro índice **IX** lo usaremos como puntero de marco, y la pila de la memoria empezara en las posiciones superiores e ira decrementándose.

#0[.IX]	Valor de Retorno
#-1[.IX]	Enlace de Control
#-2[.IX]	Estado Máquina
#-3[.IX]	Enlace de Acceso
#-4[.IX]	Parámetro Actuales
#-4 - nP[.IX]	Dirección de Retorno
#-5 - nP[.IX]	Variables locales
#-5 - np - nV[.IX]	Temporales

nP = número de parámetros
nV = número de variables locales

ENS2001

La Máquina Virtual que simula la aplicación posee las siguientes características:

- **Procesador** con ancho de palabra de 16 bits.
- **Memoria** de 64 K palabras (de 16 bits cada una). Por tanto, el direccionamiento es de 16 bits, coincidiendo con el ancho de palabra, desde la dirección 0 a la 65535 (FFFFh).
- **Banco de Registros**. Todos ellos son de 16 bits.
- **PC** (Contador de Programa): Indica la posición en memoria de la siguiente instrucción que se va a ejecutar.
- **SP** (Puntero de Pila): Indica la posición de memoria donde se encuentra la cima libre de la pila.
- **SR** (Registro de Estado): Almacena el conjunto de los biestables de estado.
- **IX**, **IY** (Registros Índices): Se emplean para efectuar direccionamientos relativos.
- **A** (Acumulador): Almacena el resultado de las operaciones aritméticas y lógicas de dos operandos.
- **R0**..**R9** (Registros de Propósito General): Son registros cuyo uso decidirá el programador en cada momento.

Modos de direccionamiento

- Direccionamiento inmediato
  - MOVE #67, /1000 MOVE #67,.R1
- Direccionamiento directo a registro
  - SUB .IX, #8 SUB .R1,#8
- Direccionamiento directo a memoria
  - MOVE #67, /1000 INC /1000
- Direccionamiento indirecto
  - MOVE #-8[.IX], [.R1] DEC [.R1]
- Direccionamiento Relativo a registro índice
  - MOVE #-8[.IX], [.R1] DEC #-8[.IX]
- Direccionamiento Relativo a contador del programa
  - BR $3 BR /bucle

CONTROL DE SUBRUTINAS.

Instrucción	CALL
Descripción	Llamada a subrutina.
Formato	CALL op1
Código de Operación	29
Número de Operandos	1
Modos de Direccionamiento	op1: memoria, relativo a contador de programa, indirecto
Comportamiento	Almacena en la pila el valor del contador de programa y salta a la dirección de destino indicada por el operando 1.

Instrucción	RET
Descripción	Retorno de subrutina.
Formato	RET
Código de Operación	30
Número de Operandos	0
Modos de Direccionamiento	N/A
Comportamiento	Rescata de la pila el contenido del contador de programa.

Ejemplo invocación de un subprograma en el lenguaje PL1UnedES.

```
programa test:
 variables z : entero;
 subprogramas
 procedimiento incrementar (x:entero):  # declaración del subprograma
        variables a: entero;
        comienzo
           a = x + 1;
           escribir("a(4)= ");
           escribir(a);
           escribir();
        fin;
 comienzo
        escribir("*** INVOCACION DE SUBPROGRAMAS ***");
        escribir();
        z = 3;
        incrementar(z);  # sentencia de invocación del subprograma
        escribir("FIN");
        escribir();
fin.
```

Código intermedio generado:

Quadruple - [STARTGLOBAL null, null, null]
Quadruple - [VARGLOBAL Z, 0, null]
Quadruple - [PUNTEROGLOBAL T_6, 14, null]
Quadruple - [WRITESTRING T_0, L_3, null]
Quadruple - [WRITELN null, null, null]
Quadruple - [MV T_2, 3, null]
Quadruple - [MVA T_1, Z, null]
Quadruple - [STP T_1, T_2, null]
Quadruple - [STARTSUBPROGRAMA null, null, null] Prepara o inicializa el R.A. del subprograma
Quadruple - [MVP T_4, Z, null]
Quadruple - [PARAM T_4, null, null] Introduce en el R.A. del subprograma el valor del parámetro “z”
Quadruple - [CALL L_INCREMENTAR, null, null] actualizar el registro índice IX y salta a la etiqueta de comienzo del código del subprograma
Quadruple - [WRITESTRING T_5, L_5, null]
Quadruple - [WRITELN null, null, null]
Quadruple - [HALT null, null, null]
Quadruple - [ETIQUETA L_INCREMENTAR, null, null] etiqueta de comienzo del código del subprograma
Quadruple - [VARSUBPROGRAMA A, 0, null] inicializa la variable local “y” sino esta inicializada la inicializa a 0
Quadruple - [PUNTEROSUBPROGRAMA T_7, 15, null] posiciona .SP según el tamaño del R.A. del subprograma
Quadruple - [MVP T_2, X, null]
Quadruple - [MV T_1, 1, null]
Quadruple - [ADD T_3, T_2, T_1]
Quadruple - [MVA T_0, A, null]
Quadruple - [STP T_0, T_3, null]
Quadruple - [WRITESTRING T_5, L_0, null]
Quadruple - [MVP T_6, A, null]
Quadruple - [WRITEINT T_6, null, null]
Quadruple - [WRITELN null, null, null]
Quadruple - [FINSUBPROGRAMA L_FIN_INCREMENTAR, 6, null] posicionar .SP para rescatar la dirección de retorno al programa principal
Quadruple - [CADENA "A(4)= ", L_0, null]
Quadruple - [CADENA "*** INVOCACION DE SUBPROGRAMAS ***", L_3, null]
Quadruple - [CADENA "FIN", L_5, null]

Al invocar al subprograma deberemos de generar el código intermedio para pasar a la ejecución de subprograma invocado.

```
//Invocar un subprograma(función/procedimiento)
invSub ::= IDENTIFICADOR:id PARIZD parametrosActuales:pa PARDRCH PUNTOYCOMA
{:	InvSub iSp = new InvSub();
	//comprobaciones semánticas
	……..
	//código intermedio
	TemporalFactory tF = new TemporalFactory(scope);
	IntermediateCodeBuilder cb = new IntermediateCodeBuilder(scope);
	TemporalIF temp = tF.create();
	cb.addQuadruple("STARTSUBPROGRAMA");   //llamada a la función o procedimiento activación del R.A.
	cb.addQuadruples(pa.getIntermediateCode());  //añadir el código intermedio de los parámetros
	LabelFactory lF = new LabelFactory();
	LabelIF l1 = lF.create(id.getLexema());  //etiqueta con el identificador del subprograma
	cb.addQuadruple("CALL", l1);   //llamada a la función o procedimiento
	SymbolIF simbolo = scopeManager.searchSymbol(id.getLexema());
	if(simbolo instanceof SymbolFunction) {
		cb.addQuadruple("VALORRETORNO", temp);   //recuperar el valor de retorno si es función
	}
	iSp.setIntermediateCode(cb.create());
	iSp.setTemporal(temp);  //este temporal contendrá el valor retornado por la función
	RESULT = iSp;
:};
```

```
//Parámetros actuales
parametrosActuales ::= expresion:e COMA parametrosActuales:pa
{:
	ScopeIF scope = scopeManager.getCurrentScope()
	Parametros p = new Parametros();
	//comprobaciones semánticas
	……..
	//Generacion de Codigo Intermedio
	TemporalFactory tF = new TemporalFactory(scope);
	IntermediateCodeBuilder cb = new IntermediateCodeBuilder(scope);
	TemporalIF temp = e.getTemporal();
	cb.addQuadruples(pa.getIntermediateCode());  //añadir el código de los parámetros ya generados
	cb.addQuadruples(e.getIntermediateCode());  //añadir el código de la expresión para el nuevo parámetro
	cb.addQuadruple("PARAM", temp);  //añadir el nuevo parámetro
	p.setIntermediateCode(cb.create());

	RESULT = p;
:};
```

Si para invocar el subprograma hemos utilizado una etiqueta deberemos de insertar ésta al declarar dicho subprograma

```
//Declaración del subprograma
subprograma ::= VOID IDENTIFICADOR:id PARIZD parámetros:param PARDRCH LLVIZD sentencias:senten LLVDRCH
{:
Subprograma sP= new Subprograma();
//comprobaciones semánticas
……..
//código intermedio
TemporalFactory tF = new TemporalFactory(scope);
TemporalIF temp = tF.create();
LabelFactory lF = new LabelFactory();
LabelIF l1 = lF.create(scope.getName());   //Etiqueta inicio subprograma
LabelIF l2 = lF.create('F'+scope.getName());  //Etiqueta fin subprograma
IntermediateCodeBuilder cb= new IntermediateCodeBuilder(scope);
cb.addQuadruple("ETIQUETA", l1);  //Prepara el R.A. subprograma
List<SymbolIF> symbols = scope.getSymbolTable ().getSymbols();
for (SymbolIF s: symbols) {
        if (s instanceof SymbolVariable) {  //Comprobamos que se trata de una variable y se inicializa
                int valor = 0;
                if (((SymbolVariable)s).getValor() != null)  valor = Integer.parseInt(((SymbolVariable)s).getValor());
                // introduce la variable en el R.A. del subprograma inicializada
                cb.addQuadruple("VARSUBPROGRAMA", temp, valor);
        }
}

//la variable tamanoVariables será igual al tamaño de las variables de la tabla de símbolos (los símbolos de tipo primitivo su tamaño será 1 y los símbolos declarados por el usuario como los vectores su tamaño ira en función del número de elementos que contengan el vector)
//la variable tamano será igual al valor al tamanoVariable  más el tamaño de la tabla de temporal (cada temporal ocupa una posición de memoria por lo que su tamaño será 1 para cada temporal) y más los valores fijos del Registro de Activación del subprograma (5 = valor de retorno, enlace de control, estado de la máquina, enlace de acceso y dirección de retorno)
int tamanoVariables = obtener(de la tabla de símbolos el tamaño de variables locales del subprograma
“scope.getSymbolTable().getSize()”);
int tamano = tamanoVariables + scope.getTemporalTable().getSize() + 5;
cb.addQuadruple("PUNTEROSUBPROGRAMA", temp, tamano);  // posiciona el .SP según el tamaño del R.A. del subprograma
cb.addQuadruples(senten.getIntermediateCode ());  //añadir el código intermedio de las sentencias
cb.addQuadruple("FINSUBPROGRAMA", l2, param.getSize()* + 5);
 //param.getSize() = número de parámetros formales + valores fijos del R.A.
sP.setIntermediateCode(cb.create());

RESULT = sP;
:};
```

```
sentencia ::= RETURN expresion:e SEMICOLON;
{:      Sentencia sent = new Sentencia();
        //comprobaciones semánticas
        ……..
        //código intermedio
        IntermediateCodeBuilder cb= new IntermediateCodeBuilder(scope);
        TemporalIF temp = e.getTemporal();
        LabelFactory lF = new LabelFactory();
        LabelIF l1 = lF.create('F'+scope.getName()); //Ojo el nombre del ámbito debe de ser el del ámbito del subprograma
        cb.addQuadruples(e.getIntermediateCode ());  //añadimos el código de la expresión
        cb.addQuadruple("RETURN", l1, temp);
        sent.setIntermediateCode(cb.create());
        RESULT = sent;
 :}
```

Asignación de posición dentro del R.A. de los parámetros, variables y temporales de cada ámbito (subprograma y programa principal).

- En el Axioma de lenguaje vamos a recuperar **todos los ámbitos** que se han creado e iremos asignado las posiciones de memoria que corresponde a las variables, parámetros y temporales dentro de cada ámbito.
- La variable **direccionRA** la inicializaremos a 4 que incluye las posiciones reservadas para el Valor de Retorno, Enlace de Control, Estado de la Máquina y Enlace de Acceso del Registro de Activación propuesto.
- Asignaremos las posiciones de memoria para los parámetros del subprograma (el programa principal no tiene parámetros).
- Ajustaremos la variable **direccionRA** con una posición más correspondiente a la dirección de retorno, excepto para el programa principal que no la necesita.
- Asignaremos las posiciones de memoria para las variables locales de cada ámbito.
- Asignaremos las posiciones de memoria para los temporales de cada ámbito.
- Generamos el código intermedio para el programa principal creando las cuádruplas
    - STARTGLOBAL (inicializa el R.A. del programa principal)
    - VARGLOBAL var, 0 (inicializar las variables a cero)
    - PUNTEROGLOBAL temp, tamaño (posiciona el .SP según el tamaño del R.A. global)
- Finalmente, todo el código intermedio generado se lo pasamos al código final.

```
program ::=
{: syntaxErrorManager.syntaxInfo ("Starting parsing...");
:}


 axiom:ax
{:
//Asignación de posiciones de memoria dentro del R.A. para las variables globales y los temporales
List<ScopeIF> scopes = scopeManager.getAllScopes();
for (ScopeIF scope: scopes) {
	//posiciones reservadas dentro del R.A.(Valor de Retorno + Enlace de Control + Estado Máquina + Enlace de Acceso)
	int direccionRA = 4;
	List<SymbolIF> simbolos = scope.getSymbolTable().getSymbols();
	for (SymbolIF simbolo: simbolos) {
		//Comprobar si el simbolo es un parámetro y lo guardamos en el R.A. de la función o procedimiento correspondiente
		if (simbolo instanceof SymbolParameter) {
			//Guardamos la dirección del parámetro en SymbolParameter
			//C:\..\ArquitecturaPLII-cursoXX\src\compiler\semantic\symbol\SymbolParameter.java
			((SymbolParameter)simbolo).setAddress(direccionRA);
			//Actualizamos el valor de la dirección del RA para el próximo símbolo
			direccionRA = direccionRA + simbolo.getType().getSize();
		}
	}
//Se ajusta el valor de la variable direccionRA para los subprogramas excepto para el programa principal al no usar Dirección de Retorno
	if (scope.getLevel() != 0) {
		direccionRA = direccionRA + 1;  //se añade uno más por la posición la Dirección de Retorno PC
	}
	for (SymbolIF simbolo: simbolos) {
		if (simbolo instanceof SymbolVariable) {  //Comprobar si el simbolo es una variable
			//Guardamos la dirección del variable en SymbolVariable
			//C:\..\ArquitecturaPLII-cursoXX\src\compiler\semantic\symbol\SymbolVariable.java
			((SymbolVariable)simbolo).setAddress(direccionRA);
			//Actualizamos el valor de la dirección estatica para el próximo símbolo
			direccionRA = direccionRA + simbolo.getType().getSize();
		}
	}
	//al igual que se hicimos con las variables lo hacermos con los temporales
	List<TemporalIF> temporales = scope.getTemporalTable ().getTemporals();
	for (TemporalIF t: temporales) {
		if (t instanceof Temporal) {
			// C:\..\ArquitecturaPLII-cursoXX\src\compiler\intermediate\Temporal.java
			((Temporal)t).setAddress(direccionRA);
			direccionRA = direccionRA + ((Temporal)t).getSize();
		}
	}
}
```

```
for (ScopeIF scope: scopes) {
	//Asignación de dirección y generación de código intermedio
	IntermediateCodeBuilder cb = new IntermediateCodeBuilder(scope);
	TemporalFactory tF = new TemporalFactory(scope);
	TemporalIF temp = tF.create();
	int tamanoRA;
	tamanoRA = scope.getSymbolTable().getSize() + scope.getTemporalTable().getSize() + 4;
	if (scope.getLevel () == 0) {
		cb.addQuadruple("STARTGLOBAL");  //Prepara el R.A. global
		List<SymbolIF> simbolos = scope.getSymbolTable ().getSymbols();
		for (SymbolIF simbolo: simbolos) {
			if (simbolo instanceof SymbolVariable) {  //Comprobar si el simbolo es una variable
				int direccion = ((SymbolVariable)simbolo).getAddress();
				Variable var = new Variable(simbolo.getName(), simbolo.getScope());
				// introduce la variable en el R.A. GLOBAL inicializadas a 0
				cb.addQuadruple("VARGLOBAL", var, 0);
			}
		}
		// posiciona el .SP según el tamaño del R.A. del programa principal
		cb.addQuadruple("PUNTEROGLOBAL", temp, tamanoRA);
	}
	cb.addQuadruples(ax.getIntermediateCode());
	ax.setIntermediateCode(cb.create());
}

// No modificar esta estructura, aunque se pueden añadir más acciones semánticas
 //printamos el código intermedio generado
System.out.println("Codigo intermedio en el AXIOMA: " + ax.getIntermediateCode());
List intermediateCode = ax.getIntermediateCode ();
finalCodeFactory.setEnvironment(CompilerContext.getExecutionEnvironment());
finalCodeFactory.create (intermediateCode);
// En caso de no comentarse las sentencias anteriores puede generar una excepcion
// en las llamadas a cupTest si el compilador no está completo. Esto es debido a que
// aún no se tendrá implementada la generación de código intermedio ni final.
// Para la entrega final deberán descomentarse y usarse.

syntaxErrorManager.syntaxInfo ("Parsing process ended.");
:};
```

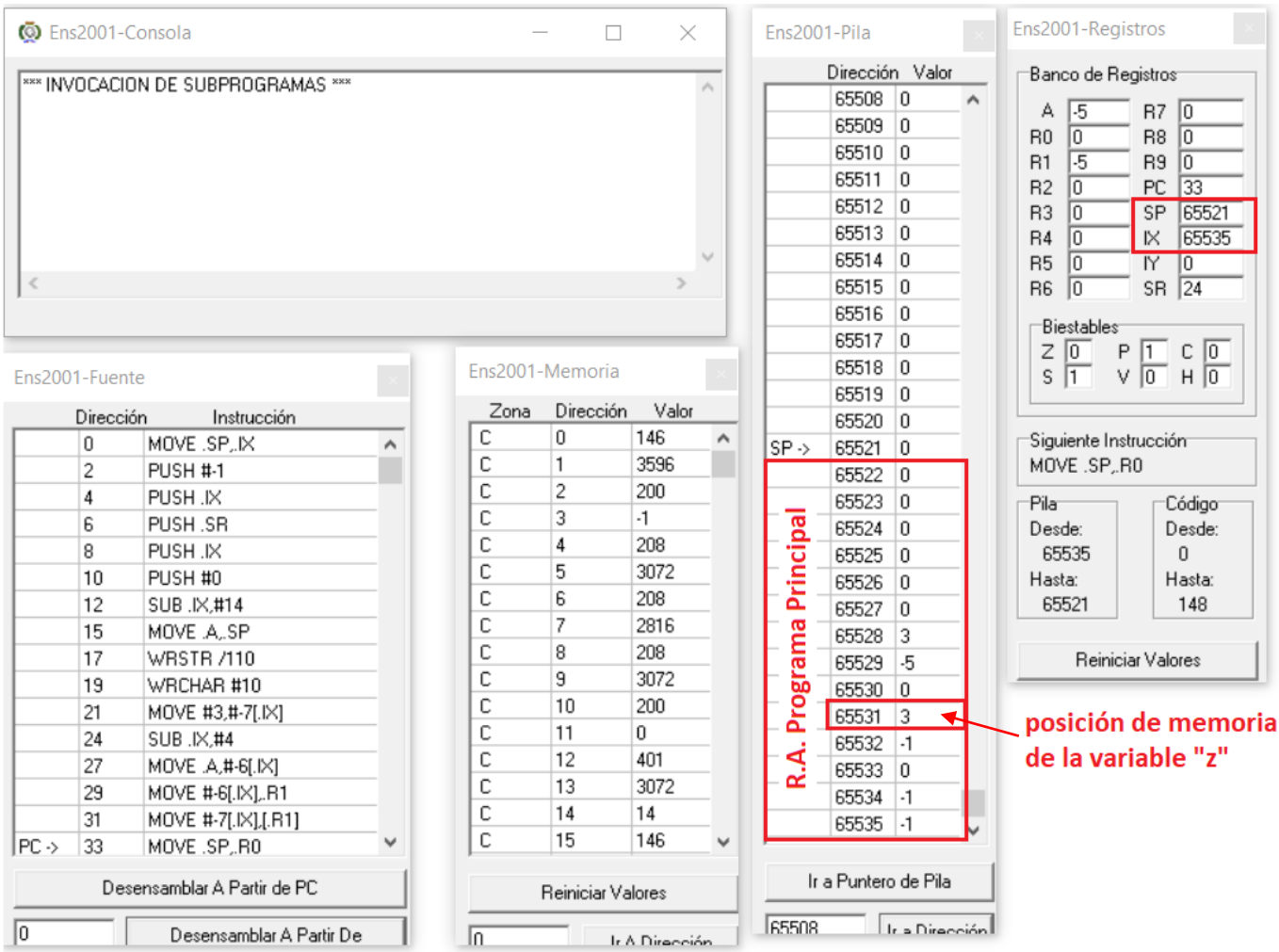
Código ensamblador (ens2001) generado por el ejemplo:

```
;Quadruple - [STARTGLOBAL null, null, null]
MOVE .SP, .IX           ;guarda el valor de SP en IX
PUSH #-1                ;valor de retorno
PUSH .IX                ;enlace de control
PUSH .SR                ;estado máquina
PUSH .IX                ;enlace de acceso
;Quadruple - [VARGLOBAL Z, 0, null]
PUSH #0
;Quadruple - [PUNTEROGLOBAL T_6, 14, null]
SUB .IX, #14
MOVE .A, .SP
;Quadruple - [WRITESTRING T_0, L_3, null]
WRSTR /L_3
;Quadruple - [WRITELN null, null, null]
WRCHAR #10
;Quadruple - [MV T_2, 3, null]
MOVE #3, #-7[.IX]
;Quadruple - [MVA T_1, Z, null]
SUB .IX, #4
MOVE .A, #-6[.IX]
;Quadruple - [STP T_1, T_2, null]
MOVE #-6[.IX], .R1
MOVE #-7[.IX], [.R1]
;Quadruple - [STARTSUBPROGRAMA null, null, null]
MOVE .SP, .R0           ;guarda el valor de SP en R0
PUSH #-1                ;valor de retorno
PUSH .R0                ;enlace de control
PUSH .SR                ;estado máquina
PUSH .IX                ;enlace de acceso
;Quadruple - [MVP T_4, Z, null]
MOVE #-4[.IX], #-9[.IX]
;Quadruple - [PARAM T_4, null, null]
PUSH #-9[.IX]
;Quadruple - [CALL L_INCREMENTAR, null, null]
MOVE .R0, .IX           ;guarda el valor de R0 en IX
CALL /L_INCREMENTAR     ;guarda en la pila el valor del contado de programa y salta a L/decrementa
MOVE .IX, .SP           ;restaura el valor de puntero de pila al retornar del subprograma
MOVE #0[.IX], .R9       ;guarda en el valor del retorno en el registro R9
MOVE #-3[.IX], .R0      ;recupera el valor del enlace de acceso para restaurar el registro indice
MOVE .R0, .IX           ;actualiza el registro índice IX para el programa principal
;Quadruple - [WRITESTRING T_5, L_5, null]
WRSTR /L_5
;Quadruple - [WRITELN null, null, null]
WRCHAR #10
;Quadruple - [HALT null, null, null]
HALT
;Quadruple - [ETIQUETA L_INCREMENTAR, null, null]
```

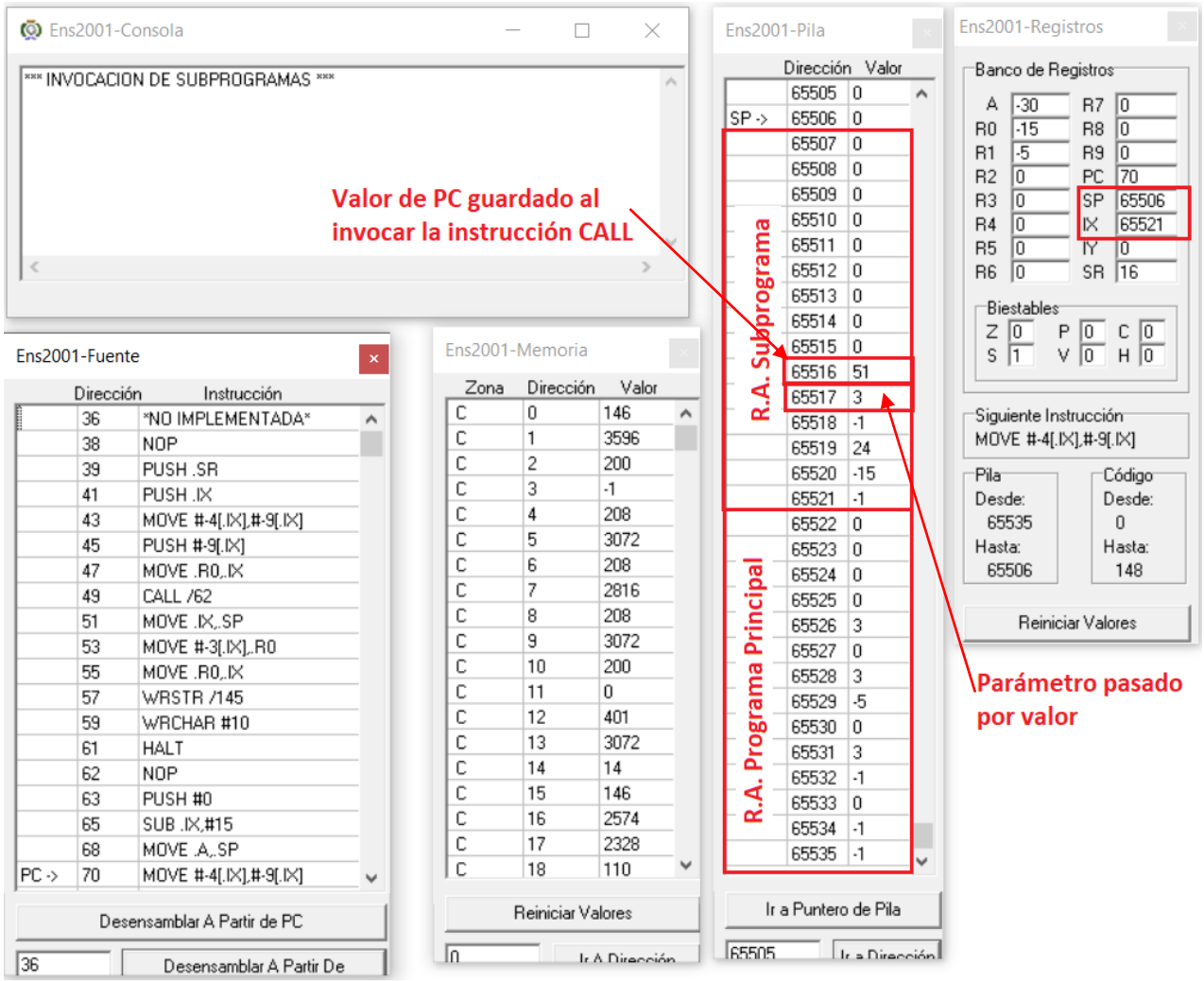
```
L_INCREMENTAR : NOP
;Quadruple - [VARSUBPROGRAMA A, 0, null]
PUSH #0
;Quadruple - [PUNTEROSUBPROGRAMA T_7, 15, null]
SUB .IX, #15
MOVE .A, .SP
;Quadruple - [MVP T_2, X, null]
MOVE #-4[.IX], #-9[.IX]
;Quadruple - [MV T_1, 1, null]
MOVE #1, #-8[.IX]
;Quadruple - [ADD T_3, T_2, T_1]
ADD #-9[.IX], #-8[.IX]
MOVE .A , #-10[.IX]
;Quadruple - [MVA T_0, A, null]
SUB .IX, #6
MOVE .A, #-7[.IX]
;Quadruple - [STP T_0, T_3, null]
MOVE #-7[.IX], .R1
MOVE #-10[.IX], [.R1]
;Quadruple - [WRITESTRING T_5, L_0, null]
WRSTR /L_0
;Quadruple - [MVP T_6, A, null]
MOVE #-6[.IX], #-13[.IX]
;Quadruple - [WRITEINT T_6, null, null]
WRINT #-13[.IX]
;Quadruple - [WRITELN null, null, null]
WRCHAR #10
;Quadruple - [FINSUBPROGRAMA L_FIN_INCREMENTAR, 6, null]
L_FIN_INCREMENTAR : NOP
SUB .IX, #6                    ;posiciona el puntero de pila SP para recuperar el contador del programa
MOVE .A, .SP
RET                            ;rescata de la pila el contenido del contador de programa
;Quadruple - [CADENA "A(4)= ", L_0, null]
L_0 : DATA "A(4)= "
;Quadruple - [CADENA "*** INVOCACION DE SUBPROGRAMAS ***", L_3, null]
L_3 : DATA "*** INVOCACION DE SUBPROGRAMAS ***"
;Quadruple - [CADENA "FIN", L_5, null]
L_5 : DATA "FIN"
```

Evolución de la pila y los registros ens2001

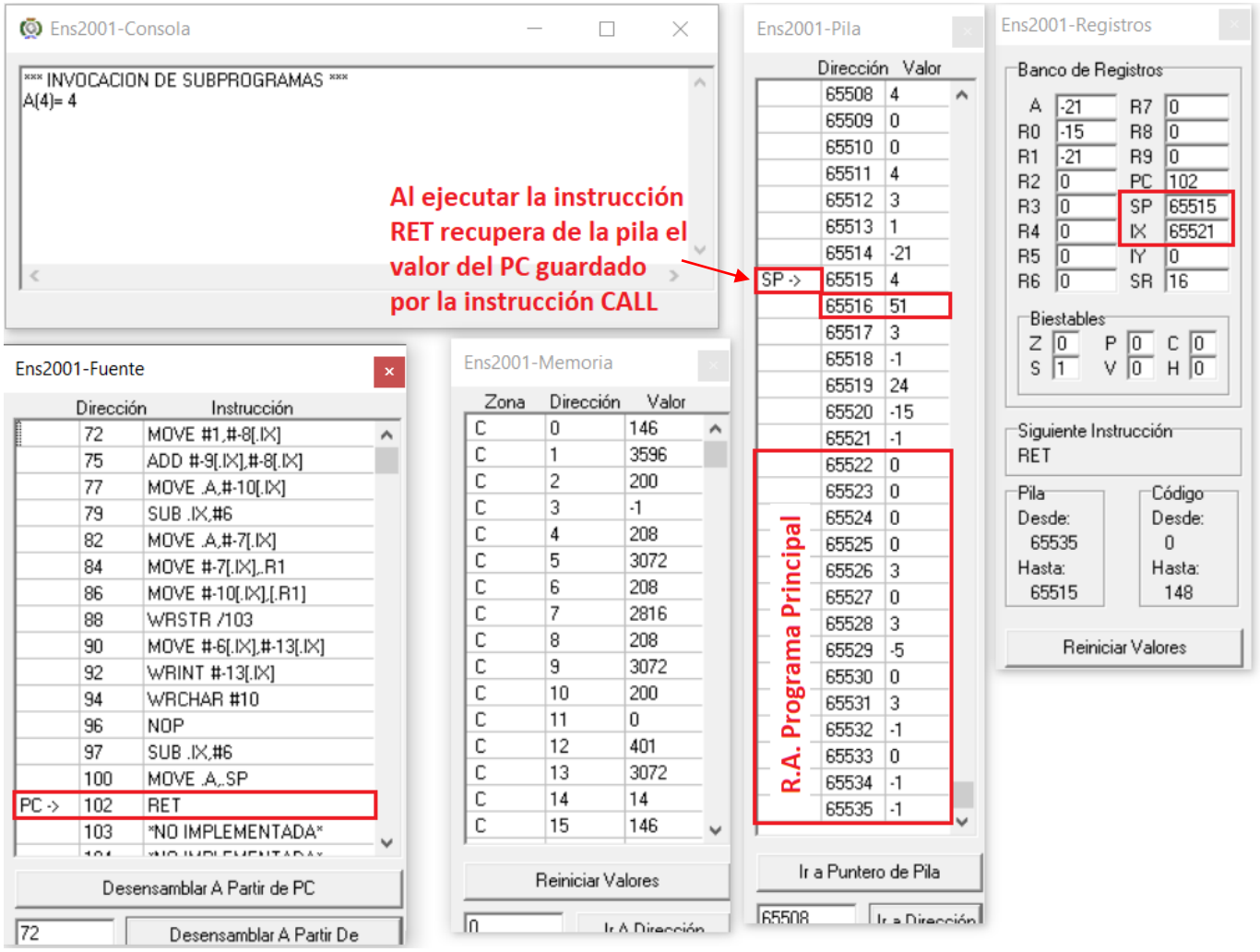
```
;Quadruple - [STARTGLOBAL null, null, null]
MOVE .SP, .IX
PUSH #-1
PUSH .IX
PUSH .SR
PUSH .IX
;Quadruple - [VARGLOBAL Z, 0, null]
PUSH #0
;Quadruple - [PUNTEROGLOBAL T_6, 14, null]
SUB .IX, #14
MOVE .A, .SP
;Quadruple - [WRITESTRING T_0, L_3, null]
WRSTR /L_3
;Quadruple - [WRITELN null, null, null]
WRCHAR #10
;Quadruple - [MV T_2, 3, null]
MOVE #3, #-7[.IX]
;Quadruple - [MVA T_1, Z, null]
SUB .IX, #4
MOVE .A, #-6[.IX]
;Quadruple - [STP T_1, T_2, null]
MOVE #-6[.IX], .R1
MOVE #-7[.IX], [.R1]
```



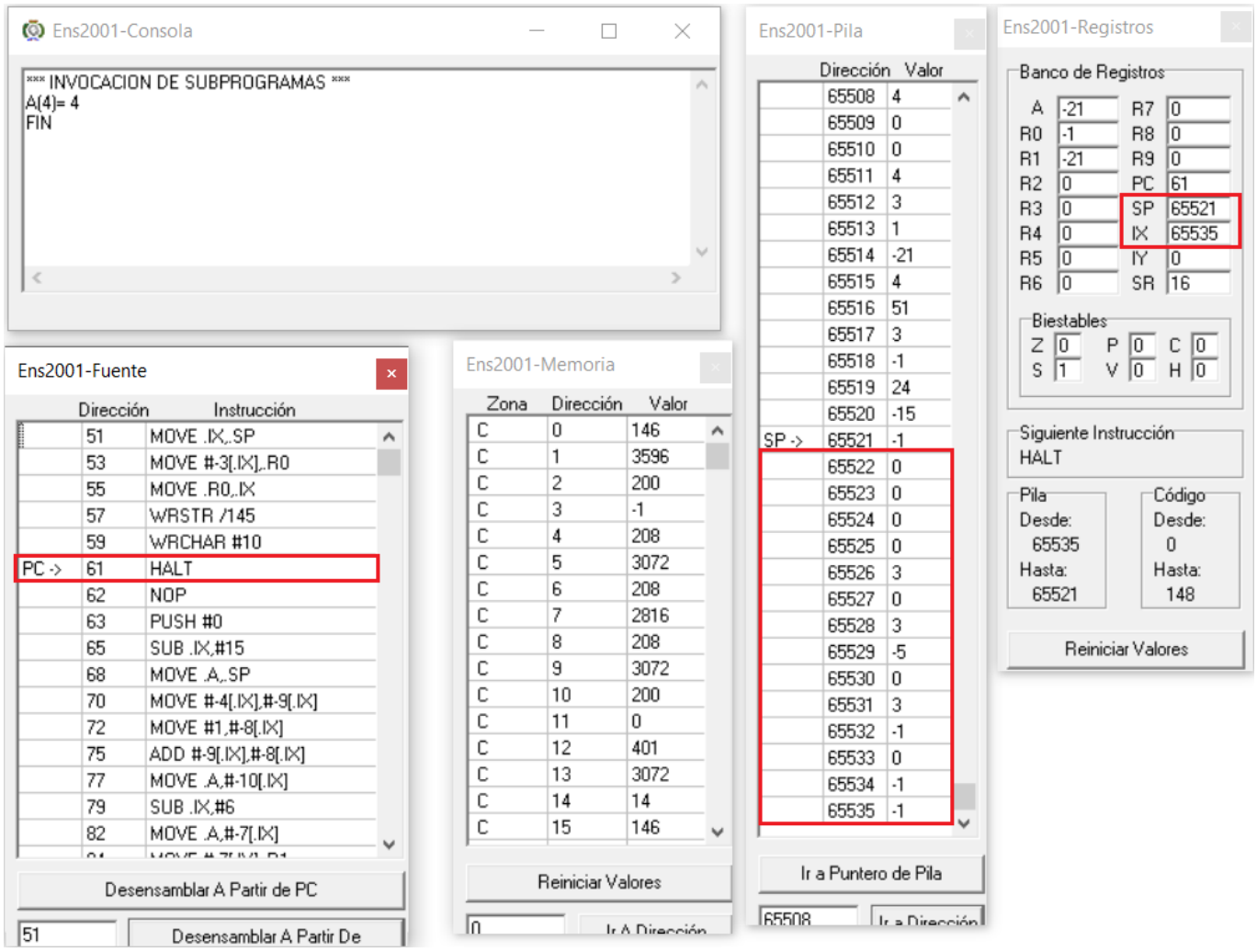
```
;Quadruple - [STARTSUBPROGRAMA null, null, null]
MOVE .SP, .R0
PUSH #-1
PUSH .R0
PUSH .SR
PUSH .IX
;Quadruple - [MVP T_4, Z, null]
MOVE #-4[.IX], #-9[.IX]
;Quadruple - [PARAM T_4, null, null]
PUSH #-9[.IX]
;Quadruple - [CALL L_INCREMENTAR, null, null]
MOVE .R0, .IX
CALL /L_INCREMENTAR
MOVE .IX, .SP
MOVE #-3[.IX], .R0
MOVE .R0, .IX
……
……
;Quadruple - [ETIQUETA L_INCREMENTAR, null, null]
L_INCREMENTAR : NOP
;Quadruple - [VARSUBPROGRAMA A, 0, null]
PUSH #0
;Quadruple - [PUNTEROSUBPROGRAMA T_7, 15, null]
SUB .IX, #15
MOVE .A, .SP
```



```
;Quadruple - [MVP T_2, X, null]
MOVE #-4[.IX], #-9[.IX]
;Quadruple - [MV T_1, 1, null]
MOVE #1, #-8[.IX]
;Quadruple - [ADD T_3, T_2, T_1]
ADD #-9[.IX], #-8[.IX]
MOVE .A , #-10[.IX]
;Quadruple - [MVA T_0, A, null]
SUB .IX, #6
MOVE .A, #-7[.IX]
;Quadruple - [STP T_0, T_3, null]
MOVE #-7[.IX], .R1
MOVE #-10[.IX], [.R1]
;Quadruple - [WRITESTRING T_5, L_0, null]
WRSTR /L_0
;Quadruple - [MVP T_6, A, null]
MOVE #-6[.IX], #-13[.IX]
;Quadruple - [WRITEINT T_6, null, null]
WRINT #-13[.IX]
;Quadruple - [WRITELN null, null, null]
WRCHAR #10
;Quadruple - [FINSUBPROGRAMA L_FIN_INCREMENTAR, 6, null]
L_FIN_INCREMENTAR : NOP
SUB .IX, #6
MOVE .A, .SP
RET
```



```
;Quadruple - [CALL L_INCREMENTAR, null, null]
MOVE .R0, .IX
CALL /L_INCREMENTAR
MOVE .IX, .SP
MOVE #-3[.IX], .R0
MOVE .R0, .IX
;Quadruple - [WRITESTRING T_5, L_5, null]
WRSTR /L_5
;Quadruple - [WRITELN null, null, null]
WRCHAR #10
;Quadruple - [HALT null, null, null]
HALT
```



Ejemplo programa en modulUned

```
MODULE test;
 VAR z : INTEGER;
 PROCEDURE incrementar (VAR x:INTEGER);
        VAR y: INTEGER;
        BEGIN
                y := x + 1;
                x := y;
        END incrementar;
 BEGIN
        z:= 999;
        incrementar(z);
        WRITESTRING("z (1000) = ");
        WRITEINT(z);
        WRITELN;
 END test;
```

Parámetros pasados por valor y por referencia

Paso del parámetro al invocar el subprograma:

Paso por referencia

```
Quadruple - [MVA T_2, Z, null]
SUB .IX, #4
MOVE .A, #-9[.IX];
;Quadruple - [PARAM T_2, null, null]
PUSH #-9[.IX]
```

Paso por valor

```
;Quadruple - [MVP T_2, Z, null]
MOVE #-4[.IX], #-9[.IX]

;Quadruple - [PARAM T_2, null, null]
PUSH #-9[.IX]
```

Instanciar un parámetro dentro del subprograma:

Paso por referencia

```
;Quadruple - [MVP T_2, X, null]
MOVE #-4[.IX], .R1
MOVE [.R1], #-9[.IX]

;Quadruple - [MVA T_10, X, null]
MOVE #-4[.IX], #-17[.IX]
```

Paso por valor

```
;Quadruple - [MVP T_1, X, null]
MOVE #-4[.IX], #-8[.IX]

;Quadruple - [MVA T_6, X, null]
SUB .IX, #4
MOVE .A, #-13[.IX]
```

Dentro del Registro de Activación vamos asignando posiciones de memoria a los valores fijos del registro (Valor de retorno, enlace de control,…) variables y temporales. Para acceder a las posiciones de memoria que ocupa el registro de Activación lo haremos a través de direccionamiento relativo al registro índice IX

R.A. test

0	Valor de Retorno
1	Enlace de Control
2	Estado maquina
3	Enlace de Acceso
4	z
5	T_0
6	T_1
7	T_2
8	T_3
9	T_4
10	T_5

R.A. incrementar

0	Valor de Retorno
1	Enlace de Control
2	Estado maquina
3	Enlace de Acceso
4	x
5	Dirección de Retorno
6	y
7	T_0
8	T_1
9	T_2
10	T_3
11	T_4
12	T_5
13	T_6

Pila de la memoria

Valor de Retorno	65535	#0[.IX]
Enlace de Control	65534	#-1[.IX]
Estado maquina	65533	#-2[.IX]
Enlace de Acceso	65532	#-3[.IX]
z	65531	#-4[.IX]
T_0	65530	#-5[.IX]
T_1	65529	#-6[.IX]
T_2	65528	#-7[.IX]
T_3	65527	#-8[.IX]
T_4	65526	#-9[.IX]
T_5	65525	#-10[.IX]
Valor de Retorno	65524	#0[.IX]
Enlace de Control	65523	#-1[.IX]
Estado maquina	65522	#-2[.IX]
Enlace de Acceso	65521	#-3[.IX]
x	65520	#-4[.IX]
Dirección de retorno	65519	#-5[.IX]
y	65518	#-6[.IX]
T_0	65517	#-7[.IX]
T_1	65516	#-8[.IX]
T_2	65515	#-9[.IX]
T_3	65514	#-10[.IX]
T_4	65513	#-11[.IX]
T_5	65512	#-12[.IX]
T_6	65511	#-13[.IX]
.SP	65510	
	65509	
	65508	

Código intermedio generado:

Quadruple - [STARTGLOBAL null, null, null],
Quadruple - [VARGLOBAL Z, 0, null],
Quadruple - [PUNTEROGLOBAL T_5, 11, null],
Quadruple - [MV T_0, 999, null],
Quadruple - [MVA T_1, Z, null],
Quadruple - [STP T_1, T_0, null],
Quadruple - [STARTSUBPROGRAMA null, null, null],
Quadruple - [MVA T_2, Z, null],
Quadruple - [PARAM T_2, null, null],
Quadruple - [CALL L_INCREMENTAR, null, null],
Quadruple - [WRITESTRING T_3, L_3, null],
Quadruple - [MVP T_4, Z, null],
Quadruple - [WRITEINT T_4, null, null],
Quadruple - [WRITELN null, null, null],
Quadruple - [HALT null, null, null],
Quadruple - [ETIQUETA L_INCREMENTAR, null, null],
Quadruple - [VARSUBPROGRAMA Y, 0, null],
Quadruple - [PUNTEROSUBPROGRAMA T_6, 14, null],
Quadruple - [MVP T_0, X, null],
Quadruple - [MV T_1, 1, null],
Quadruple - [ADD T_2, T_0, T_1],
Quadruple - [MVA T_3, Y, null],
Quadruple - [STP T_3, T_2, null],
Quadruple - [MVP T_4, Y, null],
Quadruple - [MVA T_5, X, null],
Quadruple - [STP T_5, T_4, null],
Quadruple - [FINSUBPROGRAMA L_FIN_INCREMENTAR, 6, null],
Quadruple - [CADENA "Z (1000) = ", L_3, null]

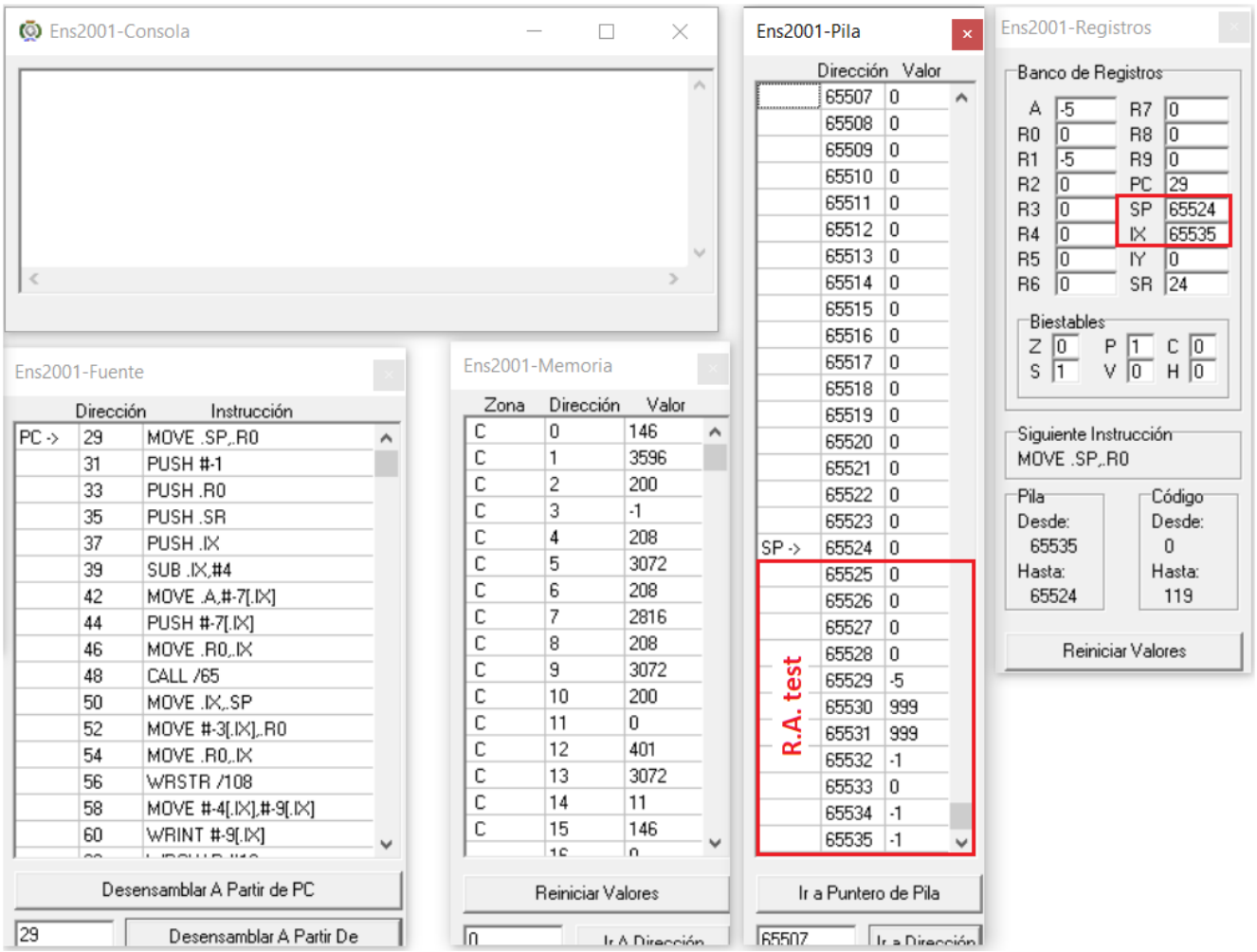
Código objeto test.ens

```
;Quadruple - [STARTGLOBAL null, null, null]
MOVE .SP, .IX
PUSH #-1
PUSH .IX
PUSH .SR
PUSH .IX
;Quadruple - [VARGLOBAL Z, 0, null]
PUSH #0
;Quadruple - [PUNTEROGLOBAL T_5, 11, null]
SUB .IX, #11
MOVE .A, .SP
;Quadruple - [MV T_0, 999, null]
MOVE #999, #-5[.IX]
;Quadruple - [MVA T_1, Z, null]
SUB .IX, #4
MOVE .A, #-6[.IX]
;Quadruple - [STP T_1, T_0, null]
MOVE #-6[.IX], .R1
MOVE #-5[.IX], [.R1]
;Quadruple - [STARTSUBPROGRAMA null, null, null]
MOVE .SP, .R0
PUSH #-1
PUSH .R0
PUSH .SR
PUSH .IX
;Quadruple - [MVA T_2, Z, null]
SUB .IX, #4
MOVE .A, #-7[.IX]
;Quadruple - [PARAM T_2, null, null]
PUSH #-7[.IX]
;Quadruple - [CALL L_INCREMENTAR, null, null]
MOVE .R0, .IX
CALL /L_INCREMENTAR
MOVE .IX, .SP
MOVE #0[.IX], .R9
MOVE #-3[.IX], .R0
MOVE .R0, .IX
;Quadruple - [WRITESTRING T_3, L_3, null]
WRSTR /L_3
;Quadruple - [MVP T_4, Z, null]
MOVE #-4[.IX], #-9[.IX]
;Quadruple - [WRITEINT T_4, null, null]
WRINT #-9[.IX]
;Quadruple - [WRITELN null, null, null]
WRCHAR #10
;Quadruple - [HALT null, null, null]
HALT
;Quadruple - [ETIQUETA L_INCREMENTAR, null, null]
```

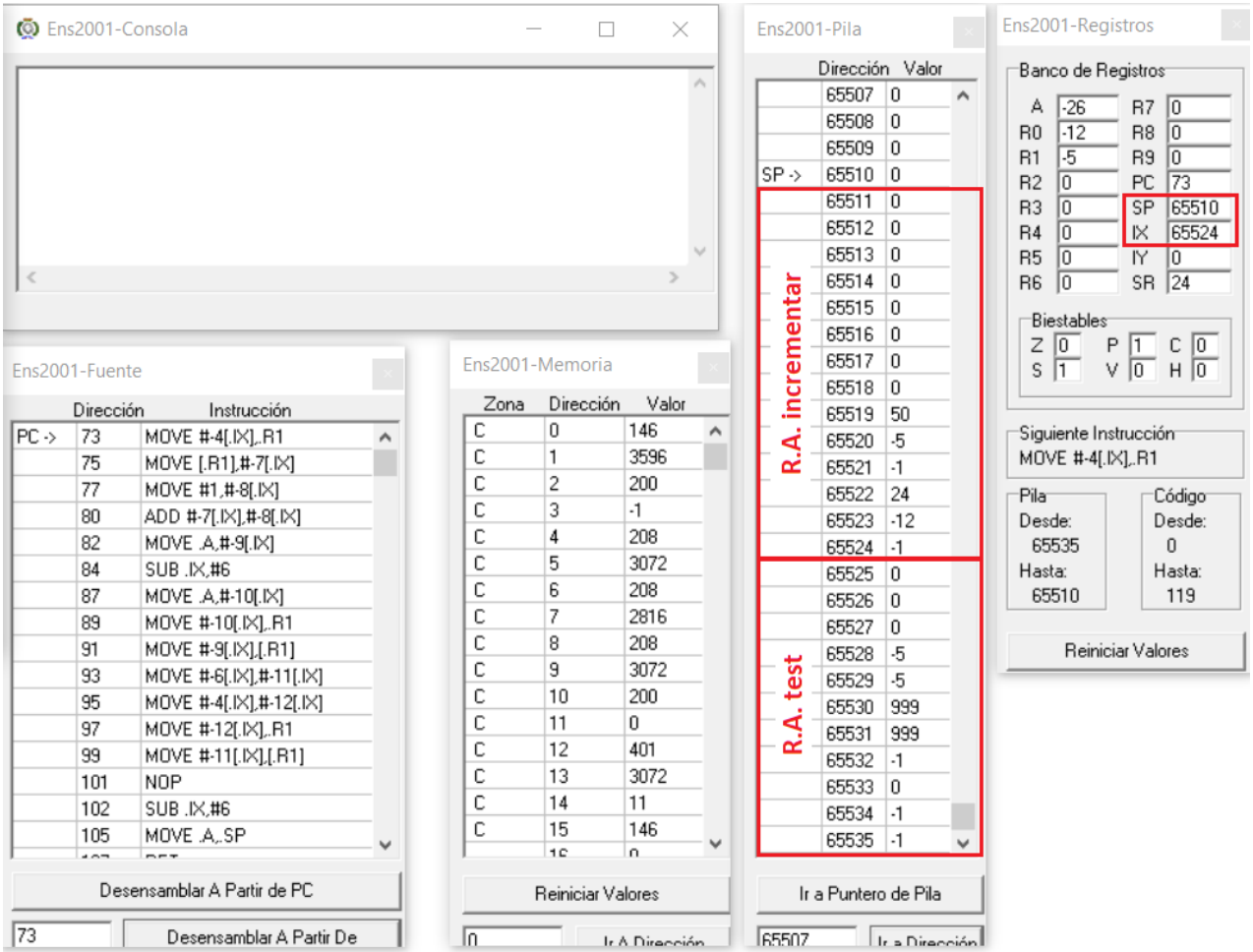
```
L_INCREMENTAR : NOP
;Quadruple - [VARSUBPROGRAMA Y, 0, null]
PUSH #0
;Quadruple - [PUNTEROSUBPROGRAMA T_6, 14, null]
SUB .IX, #14
MOVE .A, .SP
;Quadruple - [MVP T_0, X, null]
MOVE #-4[.IX], .R1
MOVE [.R1], #-7[.IX]
;Quadruple - [MV T_1, 1, null]
MOVE #1, #-8[.IX]
;Quadruple - [ADD T_2, T_0, T_1]
ADD #-7[.IX], #-8[.IX]
MOVE .A , #-9[.IX]
;Quadruple - [MVA T_3, Y, null]
SUB .IX, #6
MOVE .A, #-10[.IX]
;Quadruple - [STP T_3, T_2, null]
MOVE #-10[.IX], .R1
MOVE #-9[.IX], [.R1]
;Quadruple - [MVP T_4, Y, null]
MOVE #-6[.IX], #-11[.IX]
;Quadruple - [MVA T_5, X, null]
MOVE #-4[.IX], #-12[.IX]
;Quadruple - [STP T_5, T_4, null]
MOVE #-12[.IX], .R1
MOVE #-11[.IX], [.R1]
;Quadruple - [FINSUBPROGRAMA L_FIN_INCREMENTAR, 6, null]
L_FIN_INCREMENTAR : NOP
SUB .IX, #6
MOVE .A, .SP
RET
;Quadruple - [CADENA "Z (1000) = ", L_3, null]
L_3 : DATA "Z (1000) = "
```

Evolución de la pila y los registros ens2001

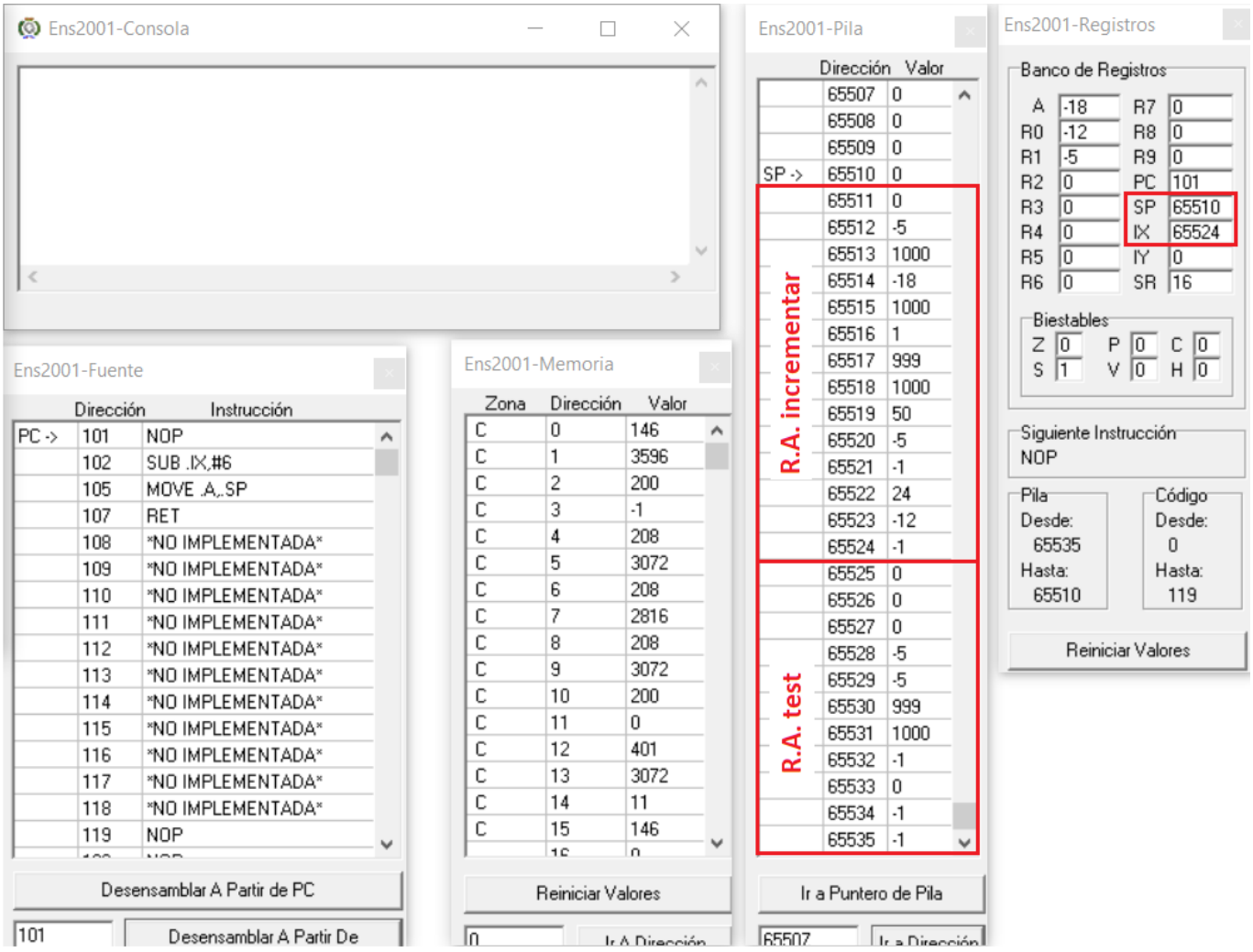
```
;Quadruple - [STARTGLOBAL null, null, null]
MOVE .SP, .IX
PUSH #-1
PUSH .IX
PUSH .SR
PUSH .IX
;Quadruple - [VARGLOBAL Z, 0, null]
PUSH #0
;Quadruple - [PUNTEROGLOBAL T_5, 11, null]
SUB .IX, #11
MOVE .A, .SP
;Quadruple - [MV T_0, 999, null]
MOVE #999, #-5[.IX]
;Quadruple - [MVA T_1, Z, null]
SUB .IX, #4
MOVE .A, #-6[.IX]
;Quadruple - [STP T_1, T_0, null]
MOVE #-6[.IX], .R1
MOVE #-5[.IX], [.R1]
```



```
;Quadruple - [STARTSUBPROGRAMA null, null, null]
MOVE .SP, .R0
PUSH #-1
PUSH .R0
PUSH .SR
PUSH .IX
;Quadruple - [MVA T_2, Z, null]
SUB .IX, #4
MOVE .A, #-7[.IX]
;Quadruple - [PARAM T_2, null, null]
PUSH #-7[.IX]
;Quadruple - [CALL L_INCREMENTAR, null, null]
MOVE .R0, .IX
CALL /L_INCREMENTAR
MOVE .IX, .SP
MOVE #-3[.IX], .R0
MOVE .R0, .IX
.....
.....
;Quadruple - [ETIQUETA L_INCREMENTAR, null, null]
L_INCREMENTAR : NOP
;Quadruple - [VARSUBPROGRAMA Y, 0, null]
PUSH #0
;Quadruple - [PUNTEROSUBPROGRAMA T_6, 14, null]
SUB .IX, #14
MOVE .A, .SP
```



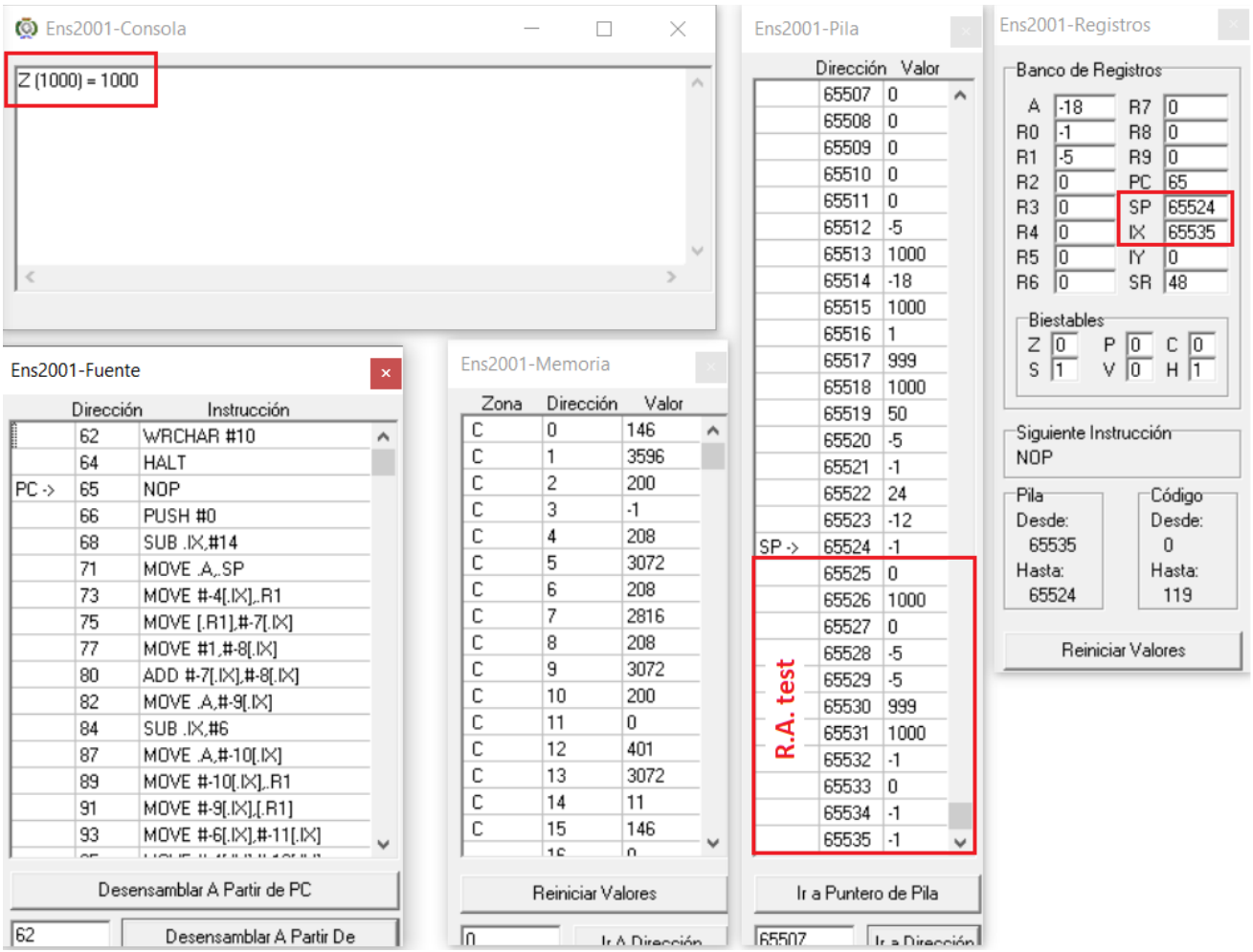
```
;Quadruple - [MVP T_0, X, null]
MOVE #-4[.IX], .R1
MOVE [.R1], #-7[.IX]
;Quadruple - [MV T_1, 1, null]
MOVE #1, #-8[.IX]
;Quadruple - [ADD T_2, T_0, T_1]
ADD #-7[.IX], #-8[.IX]
MOVE .A , #-9[.IX]
;Quadruple - [MVA T_3, Y, null]
SUB .IX, #6
MOVE .A, #-10[.IX]
;Quadruple - [STP T_3, T_2, null]
MOVE #-10[.IX], .R1
MOVE #-9[.IX], [.R1]
;Quadruple - [MVP T_4, Y, null]
MOVE #-6[.IX], #-11[.IX]
;Quadruple - [MVA T_5, X, null]
MOVE #-4[.IX], #-12[.IX]
;Quadruple - [STP T_5, T_4, null]
MOVE #-12[.IX], .R1
MOVE #-11[.IX], [.R1]
;Quadruple - [FINSUBPROGRAMA L_FIN_INCREMENTAR, 6, null]
L_FIN_INCREMENTAR : NOP
SUB .IX, #6
MOVE .A, .SP
RET
```



```
;Quadruple - [CALL L_INCREMENTAR, null, null]
MOVE .R0, .IX
CALL /L_INCREMENTAR
MOVE .IX, .SP
MOVE #-3[.IX], .R0
MOVE .R0, .IX
;Quadruple - [WRITESTRING T_3, L_3, null]
WRSTR /L_3
;Quadruple - [MVP T_4, Z, null]
MOVE #-4[.IX], #-9[.IX]
;Quadruple - [WRITEINT T_4, null, null]
WRINT #-9[.IX]
;Quadruple - [WRITELN null, null, null]
WRCHAR #10
;Quadruple - [HALT null, null, null]
HALT

.....
.....

;Quadruple - [CADENA "Z (1000) = ", L_3, null]
L_3 : DATA "Z (1000) = "
```



Ejemplo invocación de un subprograma (función) en cUned

```
/* INVOCACION SUBPROGRAMA */
int z;

int decrementa (int x) {
        int y;
        y = x -1;

        return y;
}

void main () { /* Función principal */
        /* variables locales*/
        int w;

        printc("*** INVOCACION DE SUBPROGRAMAS***");
        z = 3;
        w = decrementa(z);    /* expresión de invocación de un subprograma */
        printc("w (2) = ");
        printi(w);
        printc("FIN");
}
```

Código intermedio generado:

Quadruple - [VARGLOBAL z, 0, null],
Quadruple - [STARTMAIN null, null, null],
Quadruple - [VARMAIN null, 0, null],
Quadruple - [PUNTEROMAIN null, 14, null],
Quadruple - [PRINTC T_0, L_3, null],
Quadruple - [MV T_1, 3, null],
Quadruple - [MVA T_2, z, null],
Quadruple - [STP T_2, T_1, null],
Quadruple - [STARTSUBPROGRAMA null, null, null],
Quadruple - [MVP T_3, z, null],
Quadruple - [PARAM T_3, null, null],
Quadruple - [CALL L_decrementa, null, null],
Quadruple - [**VALORRETORNO** T_4, null, null],
Quadruple - [MVA T_5, w, null],
Quadruple - [STP T_5, T_4, null],
Quadruple - [PRINTC T_6, L_5, null],
Quadruple - [MVP T_7, w, null],
Quadruple - [PRINTI T_7, null, null],
Quadruple - [PRINTC T_8, L_6, null],
Quadruple - [HALT null, null, null],
Quadruple - [ETIQUETA L_decrementa, null, null],
Quadruple - [VARSUBPROGRAMA T_5, 0, null],
Quadruple - [PUNTEROSUBPROGRAMA T_5, 13, null],
Quadruple - [MVP T_0, x, null],
Quadruple - [MV T_1, 1, null],
Quadruple - [SUB T_2, T_0, T_1],
Quadruple - [MVA T_3, y, null],
Quadruple - [STP T_3, T_2, null],
Quadruple - [MVP T_4, y, null],
Quadruple - [**RETURN** L_Fdecrementa, T_4, null],
Quadruple - [FINSUBPROGRAMA L_Fdecrementa, 6, null],
Quadruple - [CADENA "*** INVOCACION DE SUBPROGRAMAS***", L_3, null],
Quadruple - [CADENA "w (2) = ", L_5, null],
Quadruple - [CADENA "FIN", L_6, null]

Código ensamblador (ens2001) generado por el ejemplo:

Código ens	**Quadrupla**
MOVE #0, /1000	VARGLOBAL z, 0, null
MOVE .SP, .IX	STARTMAIN null, null, null
PUSH #-1	- valor de retorno
PUSH .IX	- enlace de control
PUSH .SR	- estado de la maquina
PUSH .IX	- enlace de acceso
PUSH #0	VARMAIN null, 0, null
SUB .IX, #14	PUNTEROMAIN null, 14, null
MOVE .A, .SP	
WRSTR /L_3	PRINTC T_0, L_3, null
WRCHAR #10	
MOVE #3, #-6[.IX]	MV T_1, 3, null
ADD #0, #1000	MVA T_2, z, null
MOVE .A, #-7[.IX]	
MOVE #-7[.IX], .R1	STP T_2, T_1, null
MOVE #-6[.IX], [.R1]	
MOVE .SP, .R0	STARTSUBPROGRAMA null, null, null -guarda el valor de SP en el R0
PUSH #-1	- valor de retorno
PUSH .R0	- enlace de control
PUSH .SR	- estado de la maquina
PUSH .IX	- enlace de acceso
MOVE /1000, #-8[.IX]	MVP T_3, z, null
PUSH #-8[.IX]	PARAM T_3, null, null
MOVE .R0, .IX	CALL L_decrementa, null, null - guarda el valor de R0 en IX
CALL /L_decrementa	- guarda en la pila el valor del contado de programa y salta a L/decrementa
MOVE .IX, .SP	- restaura el valor de puntero de pila al retornar del subprograma
MOVE #0[.IX], .R9	- recupera el valor de retorno en un registro de propósito general (R9)
MOVE #-3[.IX], .R0	- recupera el valor del enlace de acceso para restaurar el registro indice
MOVE .R0, .IX	- actualiza el registro índice IX para la función main
MOVE .R9, #-9[.IX]	VALORRETORNO T_4, null, null
SUB .IX, #4	MVA T_5, w, null
MOVE .A, #-10[.IX]	
MOVE #-10[.IX], .R1	STP T_5, T_4, null
MOVE #-9[.IX], [.R1]	
WRSTR /L_5	PRINTC T_6, L_5, null
WRCHAR #10	
MOVE #-4[.IX], #-12[.IX]	MVP T_7, w, null
WRINT #-12[.IX]	PRINTI T_7, null, null
WRCHAR #10	
WRSTR /L_6	PRINTC T_8, L_6, null
WRCHAR #10	
HALT	HALT null, null, null
L_decrementa : NOP	ETIQUETA L_decrementa, null, null
PUSH #0	VARSUBPROGRAMA T_5, 0, null
SUB .IX, #13	PUNTEROSUBPROGRAMA T_5, 13, null
MOVE .A, .SP	
MOVE #-4[.IX], #-7[.IX]	MVP T_0, x, null
MOVE #1, #-8[.IX]	MV T_1, 1, null
SUB #-7[.IX], #-8[.IX]	SUB T_2, T_0, T_1
MOVE .A , #-9[.IX]	
SUB .IX, #6	MVA T_3, y, null

```
MOVE .A, #-10[.IX]
MOVE #-10[.IX], .R1                 STP T_3, T_2, null
MOVE #-9[.IX], [.R1]
MOVE #-6[.IX], #-11[.IX]            MVP T_4, y, null
MOVE #-11[.IX], #0[.IX]             RETURN L_Fdecrementa, T_4, null
BR /L_Fdecrementa
L_Fdecrementa : NOP                 FINSUBPROGRAMA L_Fdecrementa, 6, null
SUB .IX, #6                             - posiciona el puntero de pila SP para recuperar el contador del programa
MOVE .A, .SP
RET                                     - rescata de la pila el contenido del contador de programa

L_3 : DATA "*** INVOCACION DE SUBPROGRAMAS***"      CADENA "*** INVOCACION DE SUBPROGRAMAS***", L_3, null
L_5 : DATA "w (2) = "               CADENA "w (2) = ", L_5, null
L_6 : DATA "FIN"                    CADENA "FIN", L_6, null
```

Ejemplo invocación subprograma (función) en modulUned

```
MODULE test;

 VAR    z:INTEGER;
        r:INTEGER;

PROCEDURE decrementar (x: INTEGER) : INTEGER;
        VAR y: INTEGER;

        BEGIN
                y := x-1;
                RETURN y;
        END decrementar;

 BEGIN
        z:=10;
        r := decrementar(z);
        WRITESTRING("r (9) = ");
        WRITEINT(r);
 END test;
```

Código intermedio generado:

Quadruple - [STARTGLOBAL null, null, null]
Quadruple - [VARGLOBAL Z, 0, null]
Quadruple - [VARGLOBAL R, 0, null]
Quadruple - [PUNTEROGLOBAL T_7, 13, null]
Quadruple - [MV T_0, 10, null]
Quadruple - [MVA T_1, Z, null]
Quadruple - [STP T_1, T_0, null]
Quadruple - [STARTSUBPROGRAMA null, null, null]
Quadruple - [MVP T_2, Z, null]
Quadruple - [PARAM T_2, null, null]
Quadruple - [CALL L_DECREMENTAR, null, null]
Quadruple - [VALORRETORNO T_3, null, null]
Quadruple - [MVA T_4, R, null]
Quadruple - [STP T_4, T_3, null]
Quadruple - [WRITESTRING T_5, L_1, null]
Quadruple - [MVP T_6, R, null]
Quadruple - [WRITEINT T_6, null, null]
Quadruple - [WRITELN null, null, null]
Quadruple - [HALT null, null, null]
Quadruple - [ETIQUETA L_DECREMENTAR, null, null]
Quadruple - [VARSUBPROGRAMA Y, 0, null]
Quadruple - [PUNTEROSUBPROGRAMA T_5, 13, null]
Quadruple - [MV T_0, 1, null]
Quadruple - [MVP T_1, X, null]
Quadruple - [SUB T_2, T_1, T_0]
Quadruple - [MVA T_3, Y, null]
Quadruple - [STP T_3, T_2, null]
Quadruple - [MVP T_4, Y, null]
Quadruple - [RETURN L_FIN_DECREMENTAR, T_4 null]
Quadruple - [FINSUBPROGRAMA L_FIN_DECREMENTAR, 6, null]
Quadruple - [CADENA "r (9) = ", L_1, null]

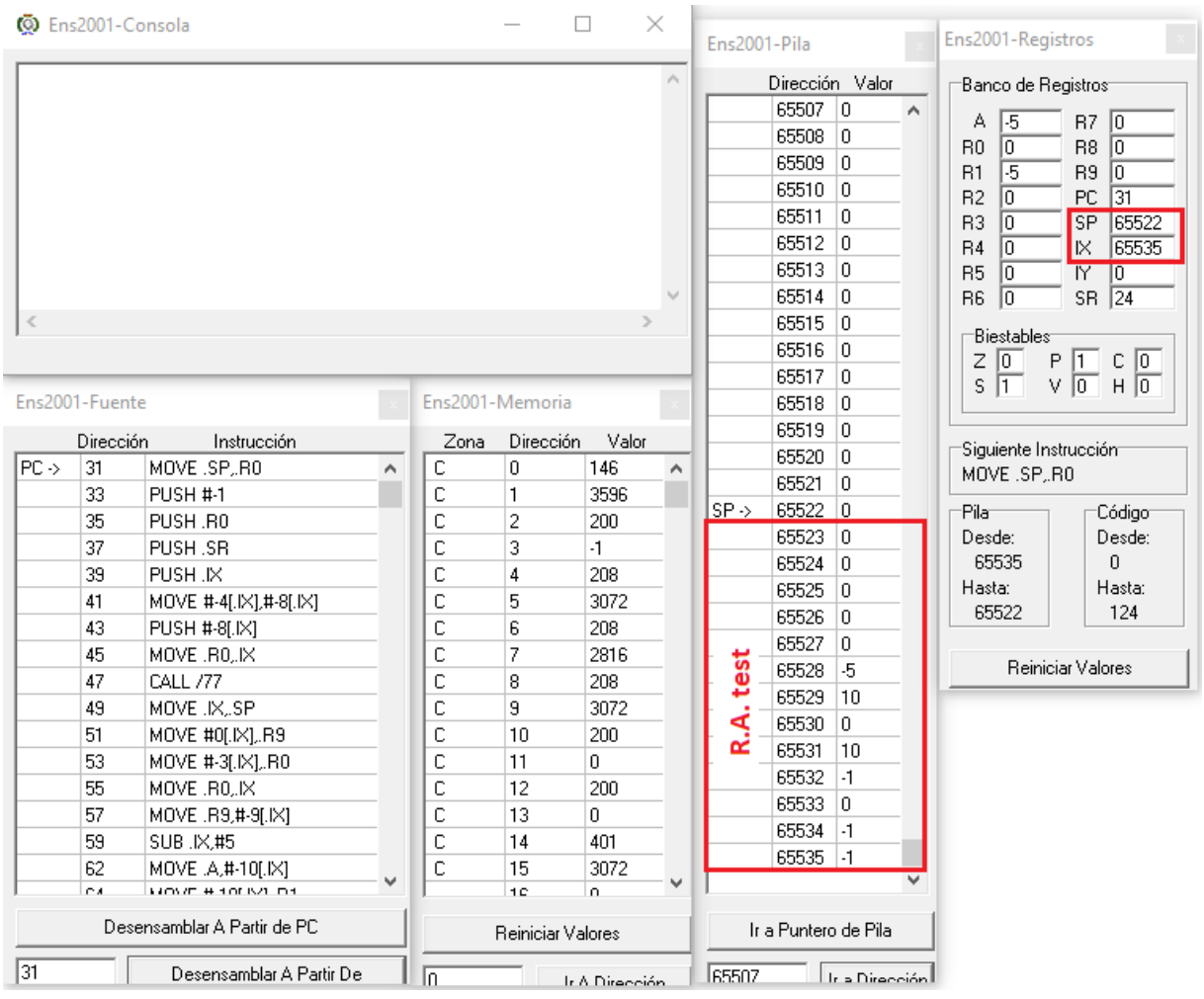
Código objeto test.ens:

```
;Quadruple - [STARTGLOBAL null, null, null]
MOVE .SP, .IX
PUSH #-1
PUSH .IX
PUSH .SR
PUSH .IX
;Quadruple - [VARGLOBAL Z, 0, null]
PUSH #0
;Quadruple - [VARGLOBAL R, 0, null]
PUSH #0
;Quadruple - [PUNTEROGLOBAL T_7, 13, null]
SUB .IX, #13
MOVE .A, .SP
;Quadruple - [MV T_0, 10, null]
MOVE #10, #-6[.IX]
;Quadruple - [MVA T_1, Z, null]
SUB .IX, #4
MOVE .A, #-7[.IX]
;Quadruple - [STP T_1, T_0, null]
MOVE #-7[.IX], .R1
MOVE #-6[.IX], [.R1]
;Quadruple - [STARTSUBPROGRAMA null, null, null]
MOVE .SP, .R0
PUSH #-1
PUSH .R0
PUSH .SR
PUSH .IX
;Quadruple - [MVP T_2, Z, null]
MOVE #-4[.IX], #-8[.IX]
;Quadruple - [PARAM T_2, null, null]
PUSH #-8[.IX]
;Quadruple - [CALL L_DECREMENTAR, null, null]
MOVE .R0, .IX
CALL /L_DECREMENTAR
MOVE .IX, .SP
MOVE #-0[.IX], .R9
MOVE #-3[.IX], .R0
MOVE .R0, .IX
;Quadruple - [VALORRETORNO T_3, null, null]
MOVE .R9, #-9[.IX]
;Quadruple - [MVA T_4, R, null]
SUB .IX, #5
MOVE .A, #-10[.IX]
;Quadruple - [STP T_4, T_3, null]
MOVE #-10[.IX], .R1
MOVE #-9[.IX], [.R1]
```

```
;Quadruple - [WRITESTRING T_5, L_1, null]
WRSTR /L_1
;Quadruple - [MVP T_6, R, null]
MOVE #-5[.IX], #-11[.IX]
;Quadruple - [WRITEINT T_6, null, null]
WRINT #-11[.IX]
;Quadruple - [WRITELN null, null, null]
WRCHAR #10
;Quadruple - [HALT null, null, null]
HALT
;Quadruple - [ETIQUETA L_INCREMENTAR, null, null]
L_DECREMENTAR : NOP
;Quadruple - [VARSUBPROGRAMA Y, 0, null]
PUSH #0
;Quadruple - [PUNTEROSUBPROGRAMA T_5, 13, null]
SUB .IX, #13
MOVE .A, .SP
;Quadruple - [MV T_0, 1, null]
MOVE #1, #-7[.IX]
;Quadruple - [MVP T_1, X, null]
MOVE #-4[.IX], #-8[.IX]
;Quadruple - [SUB T_2, T_1, T_0]
SUB #-8[.IX], #-7[.IX]
MOVE .A, #-9[.IX]
;Quadruple - [MVA T_3, Y, null]
SUB .IX, #6
MOVE .A, #-10[.IX]
;Quadruple - [STP T_3, T_2, null]
MOVE #-10[.IX], .R1
MOVE #-9[.IX], [.R1]
;Quadruple - [MVP T_4, Y, null]
MOVE #-6[.IX], #-11[.IX]
;Quadruple - [RETURN L_FIN_DECREMENTAR, T_4 null]
MOVE #-11[.IX], #0[.IX]
BR /L_FIN_DECREMENTAR
;Quadruple - [FINSUBPROGRAMA L_FIN_INCREMENTAR, 6, null]
L_FIN_DECREMENTAR : NOP
SUB .IX, #6
MOVE .A, .SP
RET
; Quadruple - [CADENA "r (9) = ", L_1, null]
L_1 : DATA "R (9) = "
```

Evolución de la pila y los registros ens2001

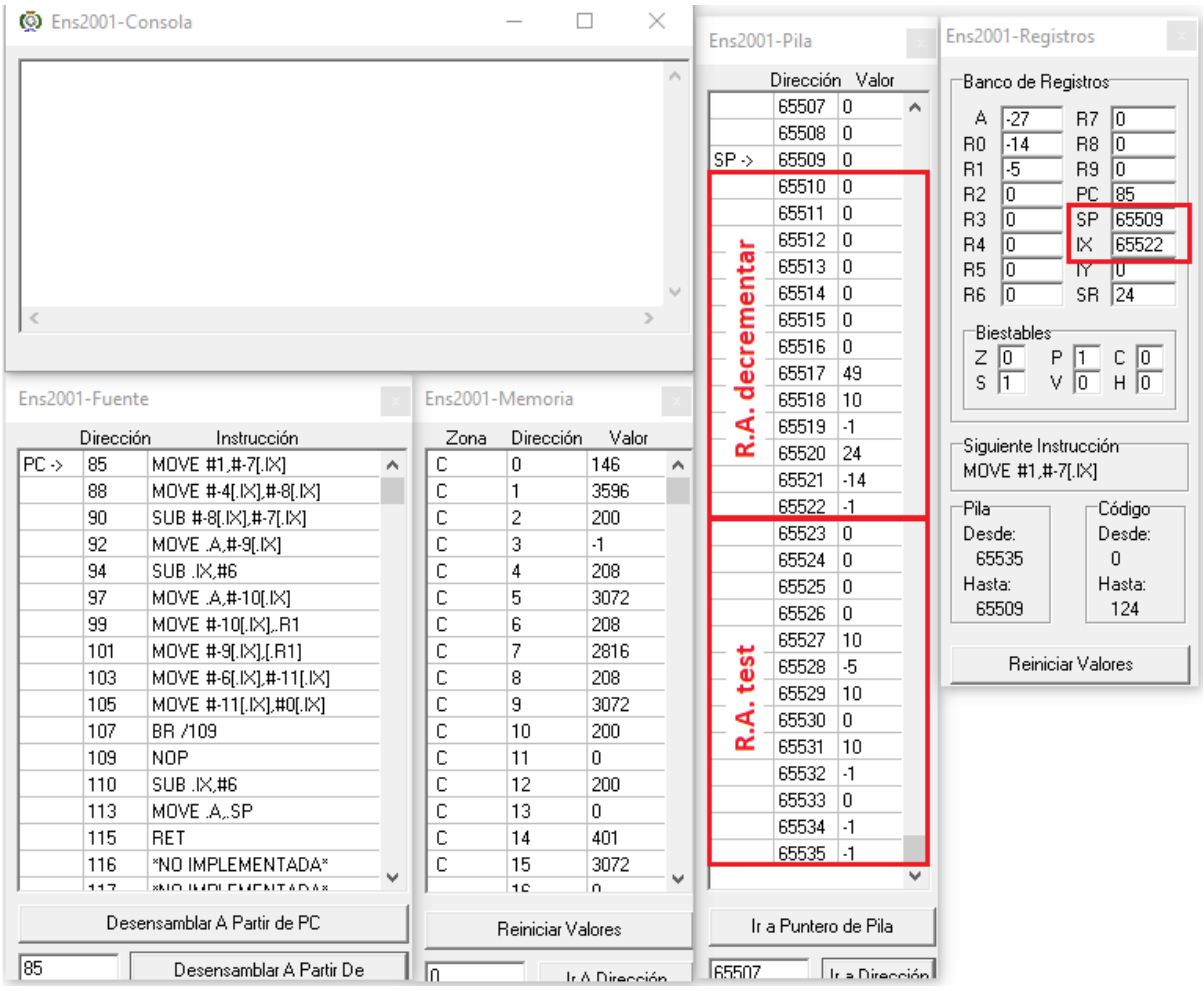
;Quadruple - [STARTGLOBAL null, null, null]
MOVE .SP, .IX
PUSH #-1
PUSH .IX
PUSH .SR
PUSH .IX
;Quadruple - [VARGLOBAL Z, 0, null]
PUSH #0
;Quadruple - [VARGLOBAL R, 0, null]
PUSH #0
;Quadruple – [PUNTEROGLOBAL T_7, 13, null]
SUB .IX, #13
MOVE .A, .SP
;Quadruple - [MV T_0, 10, null]
MOVE #10, #-6[.IX]
;Quadruple - [MVA T_1, Z, null]
SUB .IX, #4
MOVE .A, #-7[.IX]
;Quadruple - [STP T_1, T_0, null]
MOVE #-7[.IX], .R1
MOVE #-6[.IX], [.R1]



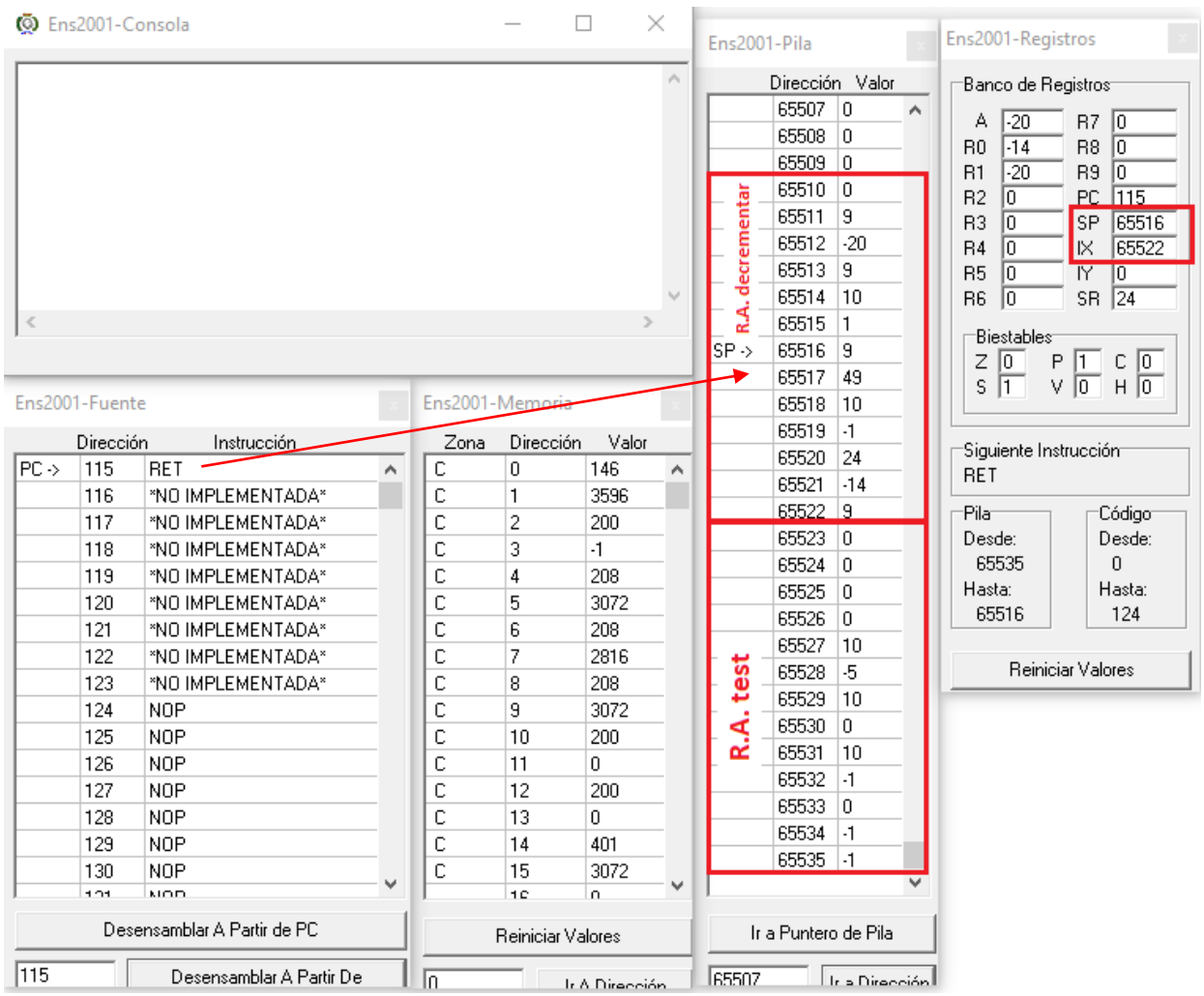
;Quadruple - [STARTSUBPROGRAMA null, null, null]
MOVE .SP, .R0
PUSH #-1
PUSH .R0
PUSH .SR
PUSH .IX
;Quadruple - [MVP T_2, Z, null]
MOVE #-4[.IX], #-8[.IX]
;Quadruple - [PARAM T_2, null, null]
PUSH #-8[.IX]
;Quadruple - [CALL L_DECREMENTAR, null, null]
MOVE .R0, .IX
CALL /L_DECREMENTAR
MOVE .IX, .SP
MOVE #-0[.IX], .R9
MOVE #-3[.IX], .R0
MOVE .R0, .IX

.....
.....

;Quadruple - [ETIQUETA L_INCREMENTAR, null, null]
L_DECREMENTAR : NOP
;Quadruple - [VARSUBPROGRAMA Y, 0, null]
PUSH #0
;Quadruple - [PUNTEROSUBPROGRAMA T_5, 13, null]
SUB .IX, #13
MOVE .A, .SP



```
;Quadruple - [MV T_0, 1, null]
MOVE #1, #-7[.IX]
;Quadruple - [MVP T_1, X, null]
MOVE #-4[.IX], #-8[.IX]
;Quadruple - [SUB T_2, T_1, T_0]
SUB #-8[.IX], #-7[.IX]
MOVE .A, #-9[.IX]
;Quadruple - [MVA T_3, Y, null]
SUB .IX, #6
MOVE .A, #-10[.IX]
;Quadruple - [STP T_3, T_2, null]
MOVE #-10[.IX], .R1
MOVE #-9[.IX], [.R1]
;Quadruple - [MVP T_4, Y, null]
MOVE #-6[.IX], #-11[.IX]
;Quadruple - [RETURN L_FIN_DECREMENTAR, T_4 null]
MOVE #-11[.IX], #0[.IX]
BR /L_FIN_DECREMENTAR
;Quadruple - [FINSUBPROGRAMA L_FIN_INCREMENTAR, 6, null]
L_FIN_DECREMENTAR : NOP
SUB .IX, #6
MOVE .A, .SP
RET
```



```
;Quadruple - [CALL L_DECREMENTAR, null, null]
MOVE .R0, .IX
CALL /L_DECREMENTAR
MOVE .IX, .SP
MOVE #-0[.IX], .R9
MOVE #-3[.IX], .R0
MOVE .R0, .IX
;Quadruple - [VALORRETORNO T_3, null, null]
MOVE .R9, #-9[.IX]
;Quadruple - [MVA T_4, R, null]
SUB .IX, #5
MOVE .A, #-10[.IX]
;Quadruple - [STP T_4, T_3, null]
MOVE #-10[.IX], .R1
MOVE #-9[.IX], [.R1]
;Quadruple - [WRITESTRING T_5, L_1, null]
WRSTR /L_1
;Quadruple - [MVP T_6, R, null]
MOVE #-5[.IX], #-11[.IX]
;Quadruple - [WRITEINT T_6, null, null]
WRINT #-11[.IX]
;Quadruple - [WRITELN null, null, null]
WRCHAR #10
;Quadruple - [HALT null, null, null]
HALT
.....
.....

; Quadruple - [CADENA "r (9) = ", L_1, null]
L_1 : DATA "R (9) = "
```