Máquina virtual PL/0

Implementación microprogramada sobre COPRO-II

Curso 2007 - 2008
Asignatura: Laboratorio de Computadores
3º Ingeniería en Informática
Departamento de Informática e Ingeniería de Sistemas
Centro Politécnico Superior

Material adicional

- Descripción de la arquitectura de COPRO II
- Manual del microensamblador de COPRO II
- Ejemplo de implementación de Máquina Sencilla sobre COPRO II
- Programas fuente y binario de Máquina Sencilla sobre COPRO II
- Manual de usuario del simulador de COPRO II
- Plantilla para la codificación de instrucciones en código máquina de COPRO II

1 DISEÑO DE UN MICROPROCESADOR PL/0

1.1 El problema

En la empresa "Polipastos y ferrallas S.A." tienen un problema. Hace años compraron un montón de microprocesadores PL/0 para controlar los distintos dispositivos y automatismos de su cadena de producción. Invirtieron mucho tiempo en desarrollar y depurar todo el software que, corriendo sobre dichas máquinas, ha permitido funcionar a la empresa durante años, sin apenas un fallo.

Pero los microprocesadores PL/0 hace tiempo que dejaron de fabricarse, y no existe hoy en día en el mercado ningún otro microprocesador con compatibilidad binaria, que pueda ejecutar directamente todo el código que "Polipastos y ferrallas S.A." desarrolló en su día para ellos. Como los microprocesadores PL/0, poco a poco, se han ido estropeando, la empresa se ha encontrado con un serio problema a la hora de renovar su tecnología.

Se ha estudiado a fondo el problema, y se han barajado varias alternativas:

1.- Solución *Software*.
Se trataría de comprar unos nuevos microprocesadores, pertenecientes a una familia distinta, y rehacer todo el código de control de la planta.

La solución puede ser interesante a medio o largo plazo, pero rehacer todo llevaría bastante tiempo, y la empresa necesita seguir funcionando mientras tanto, y no le quedan más procesadores PL/0, por lo que si se estropea uno más, algún componente de la fábrica dejará de funcionar.

2.- Solución *Hardware*
La alternativa hardware consistiría en encargar a una empresa de microelectrónica la fabricación de procesadores compatibles con PL/0.

Lo cierto es que no se dispone de los esquemas de diseño electrónico de los procesadores originales, por lo que el diseño habría que realizarlo desde cero, y la solución se descarta por cara y complicada.

3.- Solución *Microprogramada*
En esta opción, lo que se propone es comprar procesadores de arquitectura microprogramable, para los que se pueda diseñar un microcódigo (firmware) que los haga compatibles binarios con el código PL/0.

Estos procesadores se pueden encontrar en el mercado a un precio muy razonable, su microprogramación no parece especialmente complicada ni costosa, y eso permitiría reutilizar todo el código PL/0 existente sin ninguna modificación. Parece que es la solución óptima, desde el punto de vista económico y de tiempo, por lo que es la solución adoptada.

Así pues, "Polipastos y ferrallas S.A." ha decidido contrataros para que realicéis la microprogramación de este procesador, basado en la arquitectura COPRO II, de forma que sea totalmente compatible con el código binario PL/0; es decir, el procesador COPRO II, una vez microprogramado, ejecutará código binario PL/0. Dentro del contrato, como no podía ser menos, os exige una garantía de que el microprograma que tenéis que hacer y entregar esté totalmente verificado, funcione sin un solo bug, y tendréis que demostrar que eso es así mediante un conjunto de programas de prueba adecuado a éste requisito.

1.2 Objetivo de la práctica

El objetivo de esta práctica es:

- Microprogramar el procesador COPRO II de forma que pueda ejecutar código máquina PL/0
- Diseñar una estrategia de pruebas que permita verificar sin ninguna duda la validez estabilidad de máquina virtual PL/0 que se ha microprogramado. Deberá verificarse escrupulosamente el correcto funcionamiento de todos los elementos propios de la arquitectura PL/0:
    - Instruction Set (conjunto de instrucciones)
    - Modos de direccionamiento
    - Modos de paso de parámetros
    - Funcionalidades específicas (anidamiento de procedimientos, recursividad...)

Objetivo opcional:

- Aprovechar al máximo las posibilidades de la arquitectura COPRO II para optimizar todo lo posible la máquina virtual PL/0, tanto en tamaño de microcódigo como en tiempo de ejecución.

1.3 Herramientas y material

Para realizar el trabajo encargado, "Polipastos y ferrallas S.A." pone a nuestra disposición las siguientes herramientas:

- Ensamblador de COPRO II
- Desensamblador de COPRO II
- Simulador software de COPRO II
- Procesador COPRO II montado en placa de pruebas para verificación física.
- Documentación sobre los procesadores COPRO II
- Ejemplo de implementación de "Máquina Sencilla" sobre COPRO II

1.4 Conceptos a refrescar

- Máquinas virtuales
- Máquinas microprogramadas (firmware)
- Paso de parámetros por valor y por referencia
- Procesadores tipo "pila"
- En general, todos los conceptos de "Arquitectura de computadores".

1.5 Material a entregar para la evaluación de la práctica:

Documentación presentada correctamente (nombres, asignatura, práctica, año…) encuadernada (no hojas sueltas), que contenga los siguientes apartados:

a. Introducción y planteamiento general de la práctica, objetivos y metodología seguida.
b. Diseño de una estrategia de pruebas para la implementación de PL/0. Se realizará mediante una serie de programas que permitan determinar la robustez de la implementación, y depurar hipotéticos errores en la implementación de la máquina virtual, así como poner a prueba los elementos más complejos de PL/0 (bloques de activación, paso de parámetros por valor y referencia, recursividad, devolución de valores…). Se deberá entregar una explicación de la estrategia de pruebas desarrollada, justificando la elección de los programas presentados, así como el código de dichos programas en alto nivel (PASCAL o similar) y ensamblador PL/0.

Para cada programa, se incluirá su código o pseudocódigo en alto nivel, su codificación binaria, y una representación de la pila en su máxima expresión.

c. Comentarios sobre las dificultades técnicas encontradas a la hora de realizar la práctica
d. Decisiones de diseño tomadas en la implementación del microcódigo.
e. Listado del microcódigo bien comentado.
f. Fichero informático comprimido (zip) con el microprograma fuente, el programa objeto (*.mic) y el programa desensamblado (*.dis), así como el código binario de los programas de prueba.

2 Descripción de la arquitectura PL/0

2.1 Introducción

La máquina PL/0 fue descrita originalmente por Nicklaus Wirth, el creador de lenguajes de programación como Pascal o Modula-2. Wirth pretendía, introduciendo esta arquitectura, demostrar cómo, con elementos sencillos de una arquitectura, se podía conseguir una máquina (virtual o real) capaz de ejecutar sin problemas programas escritos en un subconjunto de Pascal (sPascal). La máquina PL/0 sería capaz de ejecutar las estructuras básicas del Pascal, tales como procedimientos y funciones, cláusulas condicionales, bucles, operaciones aritméticas básicas y recursividad. Por simplicidad imponía, eso sí, algunas limitaciones, como el hecho de que la máquina debería trabajar exclusivamente con números enteros, pero no podría operar, a nivel de Lenguaje Máquina, ni con caracteres o cadenas, ni con números no enteros. Cualquier ampliación en este sentido, debería venir resuelta por funciones de alto nivel, o por dispositivos de entrada/salida, capaces de representar como números enteros otro tipo de variables. No habría problema, por ejemplo, en adaptar una pantalla a la máquina PL/0 que presentase los valores numéricos proporcionados como caracteres ASCII.

2.2 Descripción básica de la máquina PL/0

La máquina PL/0 propuesta por Wirth consta de dos memorias, un registro de instrucción, y tres registros de direcciones. La *memoria de programa*, llamada *código*, permanece inalterada durante la ejecución del programa. Puede considerarse, por tanto, como una memoria de sólo lectura. La *memoria de datos* está organizada como una pila, y todos los operadores aritméticos y lógicos trabajan con los elementos de la parte superior de la pila, sustituyendo los operandos por el resultado (procesador tipo pila). El elemento superior se direcciona (indexa) con el registro de *tope de pila (Stack Pointer)*. El *registro de instrucciones* (RI) contiene la instrucción que se está ejecutando en cada momento. El *registro de dirección* (PC) designa la siguiente instrucción a interpretar. Nos queda un último registro, el *puntero de base* (Frame Pointer) del que hablaremos más adelante.

Resumiendo los elementos básicos de la máquina:

- Memoria de programa o código
- Memoria de datos (pila)
- Registro de instrucciones (RI)
- Registro de dirección (PC)
- Registro de tope de pila (SP)
- Registro de dirección de base (FP)

2.3 Procedimientos

En PL/0, los procedimientos pueden tener variables locales. Como los procedimientos pueden ser activados recursivamente, no puede asignarse memoria a estas variables antes de que se produzca la llamada a los mismos. Por ello, los segmentos de datos de cada uno de los procedimientos se apilan consecutivamente en la memoria tipo pila P. Como las activaciones de los procedimientos siguen de forma estricta el esquema "primero en entrar, último en salir" (LIFO), la pila es la estrategia de asignación de memoria apropiada. Cada procedimiento posee como propia cierta información interna, es decir, la dirección de programa desde donde fue llamado (llamada *dirección de retorno*), y la dirección del segmento de datos del módulo que le llamó. Estas dos direcciones son necesarias para proseguir adecuadamente la ejecución del programa, cuando se acabe el procedimiento en ejecución, devolviendo el control al procedimiento desde el que este último fue llamado. Estas direcciones pueden entenderse como variables internas o locales, situadas en el segmento de datos del procedimiento. Se llaman *dirección de retorno DR, y enlace dinámico ED*.

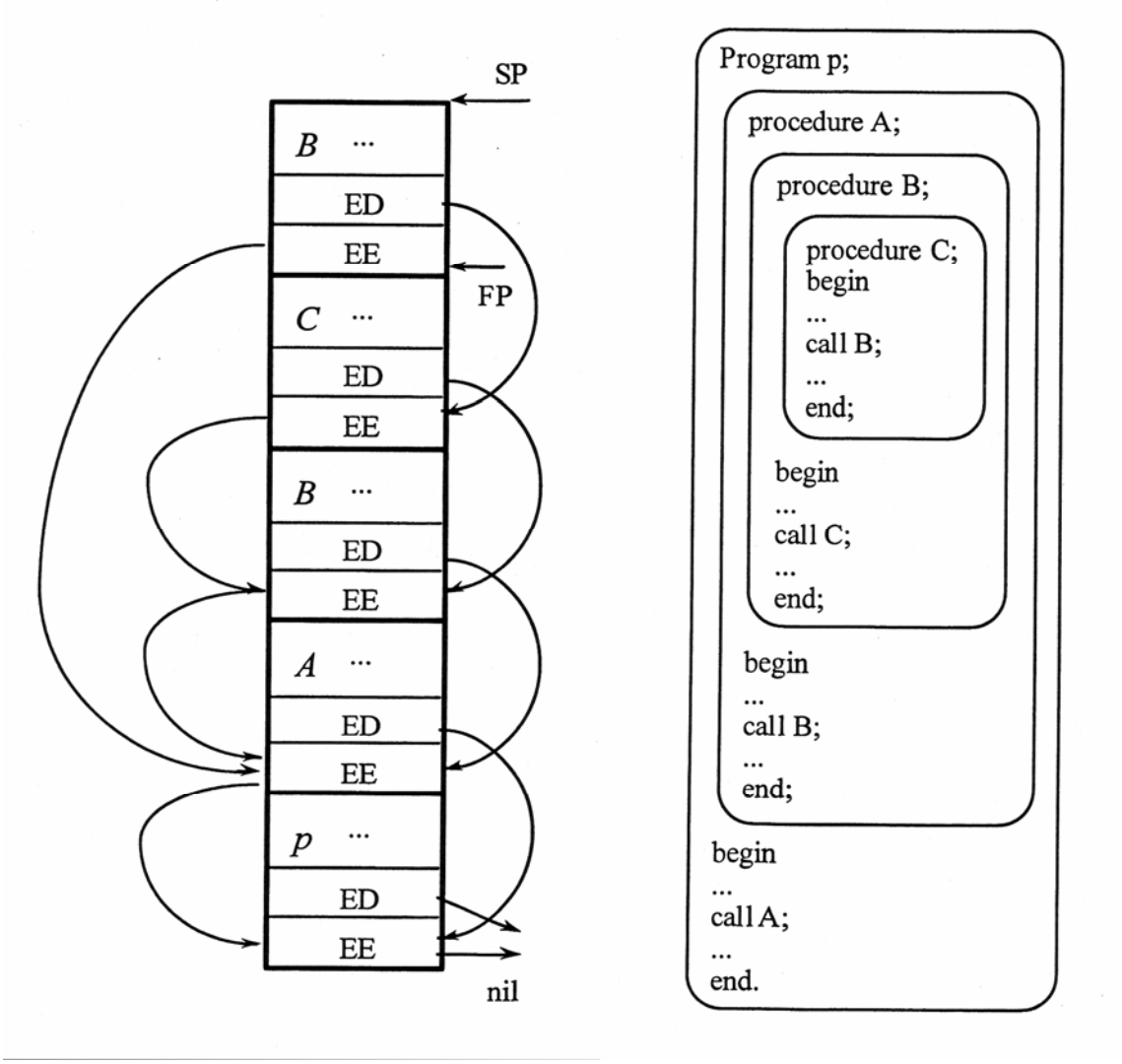
De esta forma, cuando se llama a un procedimiento (función o subrutina), lo primero que se hace es crear en la pila un segmento de datos o *bloque de activación* asociado a esa instancia del procedimiento. Este bloque de activación será el espacio de la memoria de pila donde se guardan todas las variables locales del procedimiento. La dirección del bloque de activación recién creado se guarda en el registro de base (FP), y en dicho bloque de activación deberemos guardar el enlace dinámico (ED) de dicho procedimiento, es decir, la dirección del bloque de activación del procedimiento que lo llamó, para restaurar el contexto una vez terminado el procedimiento en ejecución.

Un aspecto a tener en cuenta, como ya se ha indicado más arriba, es que llamadas sucesivas (recursivas) a un mismo procedimiento generan distintos bloques de activación, con distintas variables locales (y distintos enlaces dinámicos).

Como la asignación de memoria tiene lugar en el momento de la ejecución, el compilador que traduce desde un lenguaje de alto nivel al lenguaje máquina PL/0 no puede utilizar direcciones absolutas en el código que genera. Al no poder determinar más que la posición relativa de las variables dentro de un segmento de datos, sólo es capaz de generar *direcciones relativas*. La máquina virtual tiene que sumar, en tiempo de ejecución, a este desplazamiento la dirección de base del segmento de datos apropiado. Si una variable es local al procedimiento que se está ejecutando, la dirección de base correspondiente se encuentra en el registro FP. Sin embargo, si la variable no es local al procedimiento, sino que pertenece a un procedimiento de nivel más alto (recordemos que Pascal permite anidamiento de procedimientos, y los subprocedimientos tienen visibilidad sobre las variables de los procedimientos que lo engloban), la dirección de base adecuada debe encontrarse descendiendo a lo largo de la cadena de bloques de activación.

Sin embargo, en tiempo de compilación, podemos conocer sólo la profundidad "estática" de un camino de acceso (sabemos si la variable es local, o de su procedimiento padre, o abuelo…), mientras la cadena de enlaces dinámicos mantiene la "historia dinámica" de ejecución de los procedimientos. Desgraciadamente, estos dos caminos "estático" y "dinámico" no son generalmente iguales.

Por ejemplo, supongamos que un procedimiento A llama a un procedimiento B, declarado local en A. B llama a su vez a C, local en B, y C llama a B (recursivamente). Decimos que A está declarado en el nivel 1, B en nivel 2 y C en nivel 3.



Para acceder desde B a una variable v declarada en A, el compilador sabe que existe una diferencia de nivel de 1 entre A y B. Sin embargo, al descender un paso en la cadena dinámica, excepto en la primera instancia de B, se accedería a una variable de C, y no de A.

Por ello resulta evidente que se necesita una segunda cadena de enlaces, que relacione los segmentos de datos en la forma que el compilador ve de la estructura de los procedimientos y variables. El valor que permite realizar esta cadena "estática" se denomina Enlace Estático (EE), y se guarda en el bloque de activación, junto con el enlace dinámico y la dirección de retorno.

De acuerdo con todo esto, las direcciones son generadas por el compilador como pares de números. El primer número indica la diferencia estática de nivel (0 para variables locales, 1 para indicar el procedimiento padre, 2 el abuelo, etc…). Se supone que la memoria de datos P tiene anchura suficiente como para contener una dirección física, o un número entero.

El formato de cada bloque de activación es el siguiente:

	Variables locales
	Parámetros (opcional)
	Valor de retorno en funciones (opcional)
	DR
	ED
FP ->	EE

2.4 Repertorio de instrucciones de la máquina PL/0

Vamos a describir aquí el repertorio de instrucciones de la arquitectura PL/0, junto con una ampliación y simplificación de los mnemónicos originales, que es la representación que vamos a utilizar nosotros para nuestra implementación.

El PL/0 original consta de las siguientes instrucciones:

- LIT: Instrucción para cargar números (literales) en la pila
- LOD: Carga una variable determinada en el tope de la pila
- STO: Asigna el valor del tope de la pila a una variable.
- CAL: Realiza una llamada a una subrutina.
- INT: Reserva espacio en la pila, incrementando el valor de SP
- JMP, JPC: Instrucciones de salto incondicional y condicional.
- OPR: Conjunto de operadores aritméticos y lógicos.

A este código, nosotros le añadiremos unas cuantas instrucciones más:

- IN: Lee un dato del exterior (entrada de datos).
- OUT: Escribe un dato en una salida determinada.
- DIR: Coloca en el tope de la pila la dirección física de una variable (puntero de dato para el paso por referencia).
- LODA: Coloca en el tope de la pila el valor obtenido por referencia desde la variable indicada (la variable es un puntero al valor buscado).
- STOA: Almacenamos en la dirección indicada por referencia el valor del tope de la pila (el valor pasado como parámetro es un puntero a la variable de destino).

En el PL/0 original, todas las instrucciones tienen un formato uniforme en su expresión en ensamblador. Este formato era:

opcode	link	parameter

Donde:

- opcode: código de la operación a realizar
- link: En aquellas instrucciones que lo necesiten, indica el número de enlaces estáticos que hay que recorrer para acceder a la posición real del parámetro.
- Parámetro: Puede ser un valor inmediato, un desplazamiento dentro de un segmento de datos, o un código ampliado.

El repertorio de instrucciones de nuestra máquina virtual queda definido como sigue:

PL/0 original	PL/0 adaptado	Cód. op.	Formato 15 8 7 4 3 0 15 0			
INT 0,Y	INT Y	01	01		Y	
LIT 0, V	LIT V	02	02			V
LOD X,Y	LOD X,Y	03	03	X	Y	
STO X,Y	STO X,Y	04	04	X	Y	
CAL X,D	CAL X,D	05	05	X		D
OPR 0,0	RET	06	06			
JMP 0,D	JMP D	07	07			D
JPC 0,D	JPC D	08	08			D
OPR 0,1	NEG	09	09			
OPR 0,3	SUB	0A	0A			
OPR 0,2	ADD	0B	0B			
	IN P	0C	0C		P	
	OUT P	0D	0D	P		
OPR 0,6	ODD	0E	0E			
OPR 0,7	EQU	0F	0F			
OPR 0,8	NEQ	10	10			
OPR 0,9	LES	11	11			
OPR 0,12	LEQ	12	12			
OPR 0,11	GRT	13	13			
OPR 0,10	GEQ	14	14			
	DIR X,Y	15	15	X	Y	
	LODA X,Y	16	16	X	Y	
	STOA X,Y	17	17	X	Y	

La columna PL/0 original se incluye como curiosidad histórica, aunque nosotros usaremos la representación adaptada.
X indica un enlace, es decir, número de saltos estáticos para acceder a una variable.
Y indica un desplazamiento dentro de un segmento de datos (bloque de activación).
V indica un valor inmediato
D indica una dirección física dentro de la memoria de programa.
Casillas en negro: Ese campo no se usa.

2.5 Implementación de la pila

Habitualmente se implementan las pilas creciendo hacia direcciones inferiores de memoria. La posición inicial de la pila coincide con la máxima dirección de memoria física reservada para la misma (realmente se inicializa con ese valor + 1), y a medida que vamos introduciendo elementos, el puntero de tope de pila va decrementándose, apuntando a direcciones sucesivamente inferiores de la memoria. Tal es el caso de la arquitectura de COPRO-II, en la que tenemos una única memoria disponible para código y datos, y optimizamos el uso de la misma haciendo que el código comience en la primera posición de la memoria, y la pila empiece en la última, creciendo hacia posiciones inferiores.

2.6 Entrada / Salida sobre COPRO II

En la versión ampliada de PL/0, hemos añadido dos instrucciones para efectuar las entradas y salidas.

En la arquitectura de COPRO-II, la posibilidad de realizar entradas y salidas es limitada. Sin embargo, se ha coseguido un módulo electrónico adicional que es capaz de conectar directamente los dispositivos de entrada / salida al banco de registros de propósito general, reservando para esta función los registros 0 (00h) a 31 (1Fh), de la siguiente manera:

- Tendremos 8 puertos de entrada (00h a 07h) y ocho puertos de salida (08h a 0Fh)
- Los registros de datos para cada puerto de entrada o salida serán los correspondientes registros generales (R0 a R7 para entrada, R8 a R15 para salida).
- Cada puerto llevará asociado un registro de estado/control, que estará mapeado en el registro P+16, donde P es el número del puerto asociado. Así, los registros R16 a R23 serán registros de control para los puertos de entrada, y R24 a R31 serán los registros de control asociados a los puertos de salida.

PUERTOS	REG. DATOS	REG. CONTROL
IN 0	R0	R16
IN 1	R1	R17
IN 2	R2	R18
IN 3	R3	R19
IN 4	R4	R20
IN 5	R5	R21
IN 6	R6	R22
IN 7	R7	R23
OUT 0	R8	R24
OUT 1	R9	R25
OUT 2	R10	R26
OUT 3	R11	R27
OUT 4	R12	R28
OUT 5	R13	R29
OUT 6	R14	R30
OUT 7	R15	R31

En esta primera versión de dispositivos de entrada / salida, sólo se utiliza el bit 0 de cada registro de control. Un valor 1 de ese bit significa que hay un dato válido en el registro de datos, y por tanto habrá que leerlo, pero no se podrá sobreescribir. Un valor 0 indica que no hay dato válido en el registro de datos, y por tanto se puede escribir sin problemas.

Protocolo de entrada:

- Comprobar el bit 0 del registro de estado. Si es 0, esperar.
- Si el bit 0 del registro de estado es 1, leer el dato, y poner el bit a 0.

Protocolo de salida

- Comprobar el bit 0 del registro de estado. Si es 1, esperar
- Si el bit 0 del registro de estado es 0, escribir dato, y poner el bit a 1.

2.7 Descripción de las instrucciones a nivel RTL de PL/0

A continuación, se describen de forma abreviada los movimientos de datos entre los registros y memorias definidos en PL/0

INT Y

Crea un bloque Y de palabras en la pila
SP ← SP –Y

LIT V

Coloca la constante V en el tope de la pila
SP ← SP –1
Mem[SP] ← V

LOD X, Y

Coloca en el tope de la pila la variable situada en la dirección X, Y. base(X) será una subrutina en la que se obtendrá la dirección inicial del bloque de activación de la variable.
SP ← SP – 1
Mem[SP] ← Mem[base(X)-Y]

STO X,Y

La variable situada en la dirección X,Y recibe el valor colocado en el tope de la pila. El valor se elimina del tope de la pila.
Mem[base(X)-Y] ← Mem[SP]
SP ← SP + 1

CAL X,D

Llama al procedimiento cuya dirección de inicio es D. Actualiza los parámetros siguiente en el nuevo bloque de activación creado para el procedimiento llamado: enlace estático, enlace dinámico, y dirección de retorno. Modificará también FP y PC. X permite obtener la dirección del enlace estático.
Mem[SP-1] ← base(X) ; enlace estático
Mem[SP-2] ← FP ; enlace dinámico
Mem[SP-3] ← PC siguiente instrucción; dirección de retorno
FP ← SP-1
PC ← D

RET

Efectúa el retorno del procedimiento, actualizando SP y FP
SP ← FP + 1
PC ← Mem [SP - 3]
FP ← Mem [SP – 2]

JMP D

Salto incondicional a la dirección D
PC ← D

JPC D

Salto condicional a D en función del valor de tope de pila. Elimina el valor de tope de pila.

Si Mem[SP] = 0 entonces PC ← D
Si_no PC ← PC siguiente instrucción
SP ← SP + 1

NEG

Efectúa la negación del valor de tope de pila
Mem[SP] ← -Mem[SP]

SUB

Efectúa la substracción de los dos valores superiores de la pila, los elimina de la pila, y coloca el resultado en el tope de pila.
SP ← SP + 1
Mem[SP] ← Mem [SP] – Mem[SP-1]

ADD

Efectúa la adición de los dos valores superiores de la pila, los elimina de la pila, y coloca el resultado en el tope de pila.
SP ← SP + 1
Mem[SP] ← Mem [SP] + Mem[SP-1]

IN P

Toma un dato del exterior y lo coloca en el tope de pila
SP ← SP – 1
Mem[SP] ← valor de entrada del puerto P
(Ver protocolo de entrada salida, para determinar los valores de los registros de estado)

OUT P

Efectúa la escritura del dato en el tope de la pila sobre el puerto de salida P
Puerto de salida P ← Mem[SP]
SP ← SP + 1
(Ver protocolo de entrada salida, para determinar los valores de los registros de estado)

ODD

Testea si el valor colocado en el tope de pila es impar
Mem[SP] ← impar(Mem[SP])

Las instrucciones siguientes colocan en el tope de pila el valor de una condición resultado de una operación de comparación realizado entre los valores que están al tope de la pila (eliminando previamente dichos valores). Es importante fijarse en el orden en el que se comparan los operandos. El valor colocado como resultado de la comparación será: 0 = FALSO; distinto de 0 = VERDADERO.

EQU

Igual
SP ← SP + 1
Mem[SP] ← Mem[SP] = Mem[SP-1]

NEQ

Distinto
SP ← SP + 1
Mem[SP] ← Mem[SP] <> Mem[SP-1]

LES

Menor que
SP ← SP + 1
Mem[SP] ← Mem[SP] < Mem[SP-1]

LEQ

Menor o igual que
SP ← SP + 1
Mem[SP] ← Mem[SP] <= Mem[SP-1]

GRT

Mayor que
SP ← SP + 1
Mem[SP] ← Mem[SP] > Mem[SP-1]

GEQ

Mayor o igual que
SP ← SP + 1
Mem[SP] ← Mem[SP] <= Mem[SP-1]

DIR X,Y

Coloca en el tope de la pila la **dirección** de la variable que pasamos por referencia.
SP ← SP – 1
Mem[SP] ← base(X) – Y

LODA X, Y

Carga en el tope de la pila el valor cuya dirección es X,Y. En general X será 0, ya que al pasar a una subrutina un parámetro por referecia, la dirección del parámetro se encontrará en el bloque de activación activo.
SP ← SP – 1
Mem[SP] ← Mem[Mem[base(X) – Y]]

STOA X, Y

Asigna a la variable pasada por referencia en X,Y el valor situado en el tope de la pila
Mem[Mem(base(X) – Y]] ← Mem[SP]
SP ← SP + 1

A estas instrucciones, hay que agregar un elemento, que no es una instrucción PL/0 propiamente dicha, pero que debe repetirse con cada instrucción. Es el *fetch*, la operación de leer la siguiente instrucción de la memoria de código, y cargarla en el registro IR, e incrementar el contador de programa para dejarlo preparado para la siguiente instrucción.

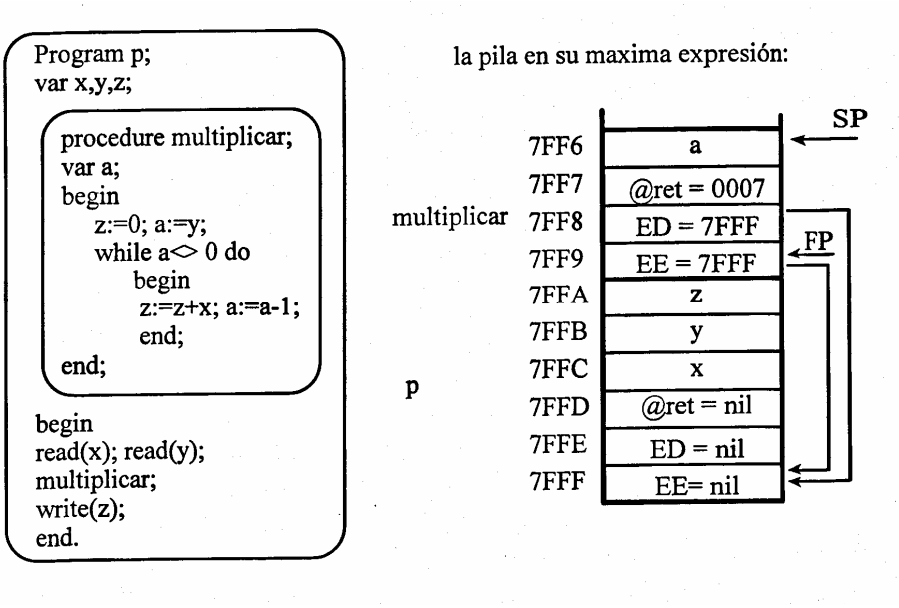
FECTH

IR ← MemCodigo[PC]
PC ← PC + 1

2.8 Ejemplo de traducción sPascal -> PL/0

Este ejemplo es un programa en PASCAL con sólo una subrutina y sin paso de parámetros; no usará ni mostrará todo el bloque de activación, por ejemplo, ni todas las instucciones de PL/0. Lo traduciremos al ensamblador del PL/0 y a hexadecimal; este último sería el código a cargar en la macromemoria como programa prueba que se ejecuta sobre la máquina virtual de PL/0.

Serán necesarios otros ejemplos con varias procedures y funciones, imbricadas en diferentes niveles, pasándose parámetos por valor y referencia entre subrutinas del mismo nivel o de diferente, etc.



	Ensamblador PL/0		Alto Nivel	@	L.M.
0)	p:	INT 6	var x,y,z	0000	0106
1)		IN 0	read (x), del registro rO	0001	OCOO
2)		STO 0,3		0002	0403
3)		IN 1	read (y) del puerto 1	0003	OCO1
4)		STO 0,4		0004	0404
5)		CAL 0,mul	call	0005	0500
				0006	0010
7)		LOD 0,5		0007	0305
8)		OUT 8	write (z) en puerto 8	0008	OD88
9)	fin:	JMP fin	bucle final	0009	0700
				OOOA	0009
10)	mul:	INT 4	var a	0010	0104
11)		LIT 0	z:=0	0011	0200
				0012	0000
13)		STO 1,5		0013	0415
14)		LOD 1,4	a:=y	0014	0314
15)		STO 0,3	a<>0	0015	0403
16)	whi:	LOD 0,3		0016	0303
17)		JPC lo		0017	0800
				0018	0024
19)		LOD 1,5	z:=z+x	0019	0315
1A)		LOD 1,3		001A	0313
1B)		ADD		001B	OBOO
1C)		STO 1,5		001C	0415
1D)		LOD 0,3	a:=a-1	001D	0303
1E)		LIT 1		001E	0200
				001F	0001
20)		SUB		0020	OAOO
21)		STO 0,3		0021	0403
22)		JMP whi		0022	0700
				0023	0016
24)	lo:	RET		0024	0600

2.9 Especificación de los binarios en código máquina PL/0

El formato de dichos programas será el siguiente:
Columnas 1-4 : Dirección de la memoria en hexadecimal (0000 a 0FFF)
Columna 5: espacio
Columnas 6-9: Instrucción en L.M. de PL/0. Dos bytes en hexadecimal.

Ejemplo:

```
0000 0103
0001 0200
0002 000A
0003 0200
0004 000A
0005 1400
0006 0D00
0007 0200
0008 00FF
0009 0200
000A 00FE
000B 1400
000C 0D10
000D 0200
000E 00FE
000F 0200
0010 00FF
0011 1400
0012 0D20
0013 0700
0014 0013
```