Paso de parámetros y gestión del marco de pila

Estructura de Computadores

1. Introducción

En clases previas se ha presentado el modelo de programación del 88110 y se ha explicado cómo se ubican las distintas estructuras que se utilizan comúnmente en un programa (matrices, listas, etc.). El manejo de estas estructuras de datos se ha realizado utilizando ejemplos prácticos para que el alumno tenga puntos de partida en la realización de programas en ensamblador. Por otra parte, en la clase anterior se ha presentado el concepto de subrutina, subrutinas anidadas y se han expuesto los pasos necesarios para llamar a una subrutina.

En esta clase se presentarán los distintos métodos que se utilizan en los procesadores y compiladores actuales para realizar el paso de parámetros a una subrutina. Por otra parte, se mostrará la gestión de recursos privados de una subrutina que realiza un compilador de un lenguaje de alto nivel y las facilidades que proporciona el juego de instrucciones de algunos computadores para realizar esta gestión. Para asimilar correctamente estos conceptos se propone el siguiente índice:

- Paso de parámetros en registros
- Paso de parámetros en variables globales
- Paso de parámetros en la pila:
  - Caso general
  - Particularización para el procesador 88110
- Activación de subrutinas desde lenguajes de alto nivel:
  - Ubicación de las variables locales
  - Creación del marco de pila
  - Destrucción del marco de pila
  - Acceso a las variables locales
  - Orden en el paso de parámetros

2. Paso de parámetros

Para que una subrutina sea útil es necesario proporcionarle los datos sobre los que tiene que operar y cómo y dónde debe almacenar los resultados de su proceso. Se denomina parámetro a cada uno de los datos de entrada que el programa llamante pasa a la subrutina o a cada uno de los resultados que la subrutina devuelve al programa llamante. Hay distintos procedimientos para llevar a cabo el paso de parámetros entre el programa llamante y la subrutina. Los más utilizados pasaremos a estudiarlos a continuación.

2.1. Paso de parámetros en registros

Una de las formas más sencillas de transferir los parámetros entre el programa llamante y la subrutina es utilizar los registros de propósito general del computador. Para ello el programa que realiza la llamada debe cargar en un conjunto de registros previamente "acordados" los datos sobre los que va a procesar la subrutina y posteriormente hacer la llamada a la subrutina. La subrutina que se ha llamado realizará los cálculos utilizando los valores que el llamante ha cargado en los registros de entrada y almacenará los resultados de la subrutina en el banco de registros del computador. A modo de ejemplo, en la figura 1 se muestra una subrutina que realiza la suma de dos números que se pasan en los registros `r2` (parámetro de entrada–salida) y `r3` (parámetro de entrada) respectivamente. El resultado de la subrutina `suma` se devuelve en el primero de ellos:

```
Programa Llamante

ld r2, r20, 0 ;Paso de parámetros
ld r3, r20, 4 ;en r2 y r3
bsr suma
```

```
      Subrutina

suma: add r2, r2, r3 ;Operación
      jmp (r1)       ;Retorno
```

Figura 1: Ejemplo de paso de parámetros en registros

La principal ventaja de este procedimiento de paso de parámetros es su rapidez, puesto que si el programa llamante tiene los parámetros disponibles en alguno de los registros del banco no es necesario acceder a memoria para pasarlos. La subrutina llamada puede operar directamente sobre los datos que se han pasado en registros sin necesidad de realizar salvaguardas de datos en memoria.

Su principal desventaja es su limitación en el número de parámetros que permite pasar, así como el tipo de los mismos. El número de parámetros que se permiten pasar en registros está limitado por el tamaño del banco de registros del computador. Por otra parte, el tipo de los parámetros queda limitado por la utilización de unos pocos registros para cada parámetro. Por ejemplo, si se desea pasar una cadena de caracteres se hace difícil ubicar subcadenas en distintos registros.

2.2. Paso de parámetros en variables globales

Como ya se ha expuesto en clases anteriores las variables globales son visibles desde todas las subrutinas de un programa, bien sea por el nombre de la variable o por la dirección que ocupa en la memoria asignada al programa. Esta característica se puede aprovechar para realizar el intercambio de parámetros de un programa llamante con una subrutina. En el momento de realizar la llamada, el programa llamante carga las variables globales con los parámetros que va a utilizar la subrutina. Una vez realizada la llamada, la subrutina accede a los argumentos de la llamada, realiza la operación y puede almacenar los resultados en variables globales.

A continuación, en la figura 2, se muestra el ejemplo de la suma de dos números que se pasan en dos variables globales (`num1` y `num2`). La subrutina invocada carga los parámetros de las variables globales en las que se pasan los parámetros y el resultado de la suma se devuelve en la variable `num1`.

```
 Programa Llamante

num1: res 4   ;Variable global
      ....
      ....
num2: res 4   ;Variable global
      bsr suma
```

```
       Subrutina

suma: or r20, r0, low(num2)
      or.u r20, r20, high(num2)
         ;Carga el primer parámetro
      ld r5, r20, 0
      or r20, r0, low(num1)
      or.u r20, r20, high(num1)
         ;Carga el segundo parámetro
      ld r6, r20, 0
      add r5, r5, r6 ;Operación
      st r5, r20, r0 ;Guarda resultado
      jmp (r1)       ;Retorno
```

Figura 2: Ejemplo de paso de parámetros en variables globales

La principal ventaja de este método es su sencillez, puesto que no necesita asignar espacio en registros o posiciones adicionales de memoria para realizar el paso de parámetros, basta con utilizar las variables que ya están definidas. Su excesiva sencillez no posibilita su utilización de forma generalizada en subrutinas que formen parte de bibliotecas. Por otra parte, esta técnica se debe evitar en la medida de lo posible, puesto que obliga a exportar un gran número de variables que pueden ocasionar problemas de reentrancia como se verá en la siguiente clase.

2.3. Paso de parámetros en la pila

Un procedimiento que permite evitar los inconvenientes del paso de parámetros descrito en las secciones anteriores es utilizar la pila del computador para realizar el intercambio de información entre el programa llamante y la subrutina. El programa llamante introduce

los parámetros en la pila mediante instrucciones *PUSH* (se han explicado en el tema de *Instrucciones y direccionamientos* de la asignatura *Estructura de Computadores*) y ejecuta la llamada a subrutina. La subrutina accede a la pila y lee los parámetros.

El principal problema que aparece en la utilización de este procedimiento se presenta cuando se almacena la dirección de retorno de la subrutina y se emplea una única pila para salvaguardar la dirección de retorno y los parámetros de la subrutina. En este caso la subrutina debe garantizar que la cima de la pila contiene la dirección de retorno en el momento de realizar el retorno de la subrutina.

La principal ventaja que aporta este procedimiento es su flexibilidad, puesto que no impone ninguna restricción en cuanto al número de parámetros a pasar ni al tamaño de cada uno de ellos. Por otra parte permite asegurar que los parámetros tienen un ámbito de validez acotado al periodo de tiempo en que la subrutina está activada, puesto que al final de la ejecución de la misma los parámetros ya no son válidos.

A modo de ejemplo, en la figura 3, se presenta un programa (en ensamblador del estándar IEEE–694) que realiza la suma de dos números pasados por valor y devuelve el resultado de la operación en un parámetro pasado por dirección.

Este programa carga en la pila los registros `.R1` y `.R2` en los que están almacenados los dos operandos. El último parámetro que se pasa es el resultado. Puesto que se pasa por dirección se introduce en la pila la dirección donde se desea que la subrutina almacene el resultado (variable `RESULT`).

```
 Programa Llamante

RESULT: RES 4
        ...
        ...
       PUSH .R1  ;Carga los dos
       PUSH .R2  ;operandos
       LD .R1, #RESULT
       PUSH .R1  ;Carga la dirección
                 ;del resultado
       CALL /SUMA
```

```
        Subrutina

SUMA: LD .R5, #8[.SP]
        ;Carga Operando 2
      LD .R6, #12[.SP]
        ;Carga Operando 1
      ADD .R5, .R6 ;Operación
      LD .R6, #4[.SP]
      ST .R5, [.R6] ;Guarda resultado
      RET           ;Retorno
```

Figura 3: Ejemplo de paso de parámetros en la pila (I)

En este ejemplo se ha supuesto que la subrutina crece hacia direcciones de memoria decrecientes y el puntero de pila apunta a la cima de la pila. En la figura 4 se muestra el estado de la pila antes de ejecutar la instrucción `CALL` e inmediatamente después. Nótese que al ejecutar la instrucción `CALL`, la dirección de retorno se almacena en la pila. Esta situación deberá tenerse en cuenta en el momento de acceder a los parámetros.

La primera instrucción de la subrutina carga el operando que se introdujo en segundo lugar en la pila en el registro `R5`. Esto se realiza utilizando direccionamiento relativo al puntero de pila `SP`. El desplazamiento utilizado en esta instrucción tiene en cuenta que la pila crece hacia direcciones de memoria decrecientes (desplazamiento positivo). Hay que

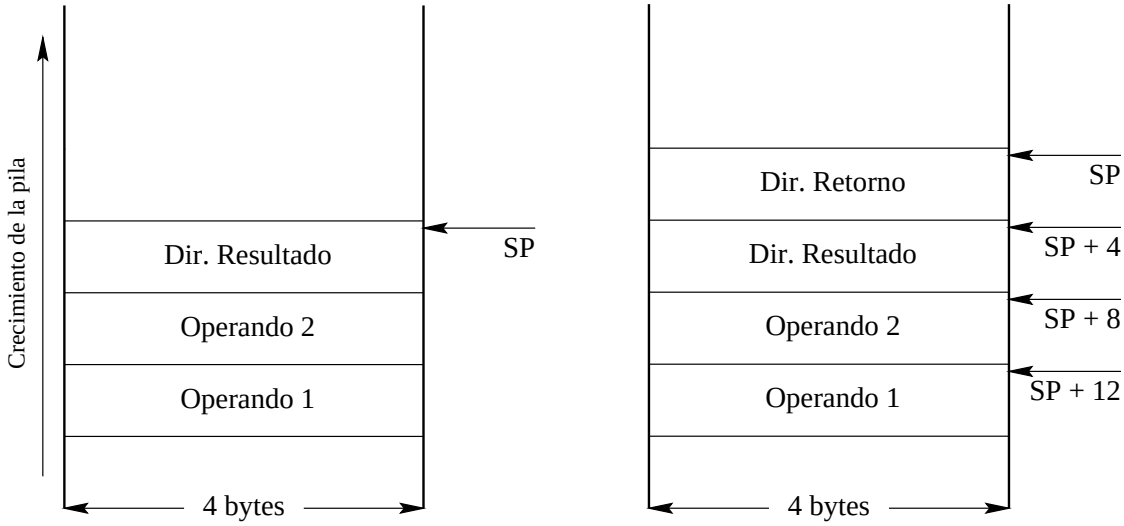

Figura 4: Estado de la pila antes y después de ejecutar la instrucción `CALL`

tener en cuenta que en el momento de acceder a dicho parámetro hay que "atravesar" dos palabras: la dirección de retorno y el parámetro `RESULT`. Cada uno de estos parámetros es de 32 bits y el computador genera direcciones a nivel de byte, por eso el desplazamiento es 2 palabras * 4 bytes/palabra = 8 bytes. El mismo razonamiento es válido para la carga del `Operando 1`.

Una vez realizada la suma de los dos operandos, es necesario almacenar el resultado en la dirección que se ha pasado como último parámetro. Para ello se carga en el registro `R6` la dirección donde se debe guardar el resultado. Esta dirección se encuentra "debajo" de la dirección de retorno, y por tanto el desplazamiento a aplicar en el direccionamiento relativo a `SP` es 4. Por último se realiza la salvaguarda del resultado en la dirección especificada por `R6`.

2.4. Gestión de la pila en el 88110

Para la realización de la práctica es válido el esquema visto en el punto anterior, pero con las siguientes salvedades:

- El procesador 88110 no tiene puntero de pila. Esto se resuelve asignando uno de los registros de propósito general del computador como puntero de pila (`r30`).
- El procesador 88110 almacena la dirección de retorno en el registro `r1`. Si la rutina que estamos realizando no es el final de una cadena de anidamientos será necesario almacenarla en la pila para permitir posteriores llamadas a subrutina.

Teniendo en cuenta estas dos premisas, el programa que se expone en la figura 3 se resolvería tal y como se expone en la figura 5. Puesto que este procesador no dispone de pila y se ha creado por el usuario, el procesador no dispone de instrucciones de gestión directa de la pila. Para facilitar dicha gestión se han realizado dos macros equivalentes a las instrucciones `PUSH` y `POP`.

```
 Programa Llamante

PUSH: MACRO(ra)
        sub   r30,r30,4
        st    ra,r30,0
      ENDMACRO
POP:  MACRO(ra)
        ld    ra,r30,0
        add   r30,r30,4
      ENDMACRO

result: res 4
       ...
       ...
      PUSH(r1)  ;Carga los dos
      PUSH(r2)  ;operandos
      or r1, r0, low(result)
      or.u r1, r1, high(result)
      PUSH(r1)  ;Carga la dirección
                ;del resultado
      bsr suma
```

```
      Subrutina

suma: PUSH(r1)
        ;Guarda la dirección
        ;de retorno
      ld r5, r30, 8
      ld r6, r30, 12
        ;Carga Operando 1
      add r5, r5, r6 ;Operación
      ld r6, r30, 4
      st r5, r6, 0;Guarda resultado
      POP(r1)
      jmp (r1)    ;Retorno
```

Figura 5: Ejemplo de paso de parámetros en la pila (y II)

La diferencia fundamental estriba en que la subrutina realiza la salvaguarda de la dirección de retorno en la pila en vez de realizarla la instrucción de bifurcación. Por otra parte, el retorno de la subrutina se realiza cargando la dirección que se almacenó al entrar en la subrutina en un registro y saltando incondicionalmente a la dirección contenida en dicho registro.

Este protocolo de entrada a una subrutina es obligatorio si ésta ejecuta a su vez otra llamada a subrutina. En el ejemplo, esta operación no es necesaria, puesto que se podría mantener la dirección de retorno en el registro `r1` y utilizar otros registros para almacenar resultados temporales de la subrutina. Por tanto bastaría con eliminar el primer `PUSH` y el `POP` y decrementar en una palabra (4 bytes) los desplazamientos que se aplican al puntero de pila (`r30`).

3. Activación de subrutinas en lenguajes de alto nivel: Marco de pila

Hoy en día está descartada la realización de aplicaciones completas en ensamblador, pero la utilización de este lenguaje es interesante cuando se desea acceder a recursos del computador que no son accesibles desde un lenguaje de alto nivel. Por ejemplo cuando se

desea acceder a los registros y dispositivos de un computador. Cuando se presenta esta situación es necesario conocer la forma de generar código maquina por el compilador, puesto que la rutina escrita en ensamblador deberá utilizar el mismo protocolo de activación. En esta sección se expone el mecanismo utilizado por los compiladores de lenguajes de alto nivel para gestionar los datos asociados a una activación de subrutina.

El principal problema que se plantea en los ejemplos expuestos en las figuras 3 y 5 es que el puntero de pila puede cambiar a lo largo de la subrutina. Esto ocurre si se desea almacenar en la pila temporalmente una variable (mediante una instrucción `PUSH`) y después se accede a un parámetro. Si los accesos a estos parámetros se realizan utilizando como base el puntero de pila, el desplazamiento a aplicar para acceder a este parámetro, antes y después de realizar el `PUSH`, es distinto.

Para resolver este problema, se establece una variable (puntero) que apunta a una zona que contiene los datos privados de una subrutina (**marco de pila**): parámetros, dirección de retorno y variables locales. Esta variable se almacenará en un registro de propósito general o específico que denominaremos **puntero de marco de pila**. En el caso particular del 88110 no existe un puntero de marco de pila y, al igual que se ha realizado con el puntero de pila, se asignará esta función a uno de los registros del computador (`r31`).

El puntero de marco de pila se deberá actualizar cuando se realice una llamada a una subrutina o cuando finalice una activación de la subrutina. En la figura 6 se muestran dos fragmentos de código que permiten crear y destruir el marco de pila de una subrutina en el 88110 para el ejemplo que se muestra a continuación. Esta subrutina opera sobre dos vectores (`a` y `b`) pasados por dirección y utiliza variables locales.

```
void vector (int a[ ], int b[ ])
{
   int vector_local[2]; /* Vector de enteros de 1 palabra (4 bytes) */
   short entero_16bits = 0; /* Entero de 2 bytes*/
   char un_byte;        /* Variable de 1 byte */
     ....
     ....
```

Los pasos que se siguen en el fragmento de código mostrado en la figura 6 para crear un nuevo marco de pila son los siguientes:

- Almacenar la dirección de retorno. Puesto que la dirección de retorno de la subrutina se ha almacenado en el registro `r1` almacenaremos esta dirección en la pila puesto que esta subrutina puede realizar nuevas llamadas.

- Esta rutina habrá sido llamada por un programa `llamante` que puede ser a su vez otra subrutina. El llamante tendrá su propio marco de pila que, cuando la subrutina `vector` finalice, tendremos que volver a activar para que `llamante` pueda seguir accediendo a sus datos privados. Por tanto, deberemos salvaguardar el puntero de marco de pila de `llamante` en la pila.

- Una vez salvaguardado el puntero de marco de pila, hay que crear el nuevo marco de pila para la subrutina que se está activando. Para ello utilizamos el puntero de

```
  Creación del marco de pila

vector: PUSH(r1)  ;Guarda
          ;la dirección de retorno
        PUSH(r31) ;Guarda
          ;el puntero al marco de
          ;pila del llamante
        or r31, r30, r0 ;Crea el
          ;nuevo marco de pila a
          ;partir del SP (r30)
        sub r30, r30, 12
           ;Reserva de espacio para
           ;variables locales
        st.h r0, r31, -10
           ;Incicialización del
           ;entero de 16 bits
           ;Código de la
           ;subrutina
        ld r6, r31, 12
           ;Acceso a la dirección
           ;de comienzo del vector
           ;b
           ...
```

```
Destrucción del marco de pila

          ...
          ...
     or r30, r31, r0
      ;Restaura el puntero
      ;de pila al valor
      ;del puntero de
      ;marco
     POP(r31)
     POP(r1)
     jmp(r1)
```

Figura 6: Creación y destrucción del marco de pila en el 88110

pila (`r30`) como valor de partida para crear nuevo puntero de marco, copiando el valor que tiene `r30` en `r31`.

- La subrutina tiene un conjunto de variables locales que se crean cuando se activa y se destruyen cuando la subrutina finaliza. Estas variables se ubican en la pila y el procedimiento de creación del marco de pila deberá reservar espacio para ellas. Las variables locales se almacenarán a continuación del puntero de marco de `llamante`. El espacio que hay que reservar se compone por un vector de 2 números enteros (8 bytes), un entero de 16 bits (2 bytes) y una variable de tipo carácter (1 byte). En total habría que reservar 11 bytes para todas estas variables. Si únicamente se reserva esta cantidad, cuando se desee volver a escribir una palabra en la pila para salvaguardar la dirección de vuelta, se producirá un error de alineamiento. Por esta razón se reservan 12 bytes para variables locales. Esta reserva se realiza restando 12 al puntero de pila. En la figura 7 se muestra la asignación a cada una de esas variables.

- Si alguna de esas variables tiene un valor inicial, a continuación se carga el valor especificado. En el ejemplo la única variable que hay que inicializar es `entero_16bits`. El acceso a las variables locales se realiza utilizando un direccionamiento relativo

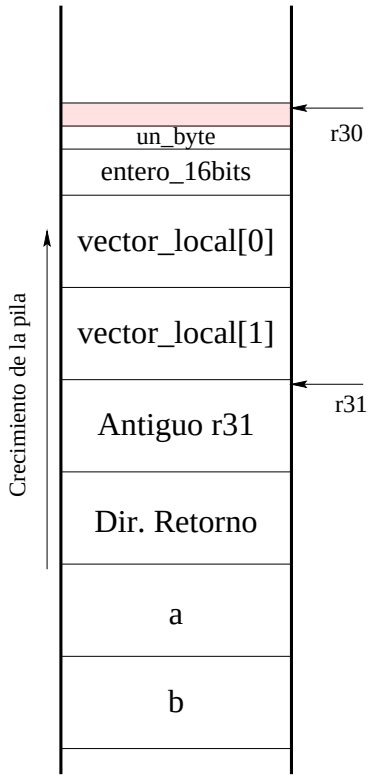

Figura 7: Marco de pila para la subrutina `vector`

al registro `r31`. Para almacenar el valor inicial basta con calcular el desplazamiento de este direccionamiento. En la figura 7 se aprecia que entre la dirección ocupada por esa variable y el registro de marco de pila hay 10 bytes (8 correspondientes al vector y los dos bytes del entero de 16 bits).

- Finalizada la creación del nuevo marco de pila, a continuación aparecería el código propio de la subrutina `vector`. Este código tendrá que acceder a los parámetros que se le han pasado. En el ejemplo se ha supuesto que se accede al parámetro `b`. Al igual que en el acceso a las variables locales, se utiliza el puntero de marco para acceder al parámetro. En este caso, el desplazamiento que hay que aplicar es positivo y hay que tener en cuenta que por encima de los parámetros están almacenados la dirección de retorno de la subrutina y el puntero de marco de `llamante`. Por tanto el desplazamiento a aplicar es 12.

La destrucción del marco de pila de `vector`, que aparece en la figura 6, es más sencilla que la activación y se compone de los siguientes pasos:

- Cuando la subrutina `vector` ha finalizado su ejecución, las variables locales dejan de tener validez y, por tanto, se eliminan de la pila. Esto se consigue moviendo el puntero de pila al lugar donde apunta el puntero de marco, es decir, se mueve el contenido del registro `r31` al `r30`.

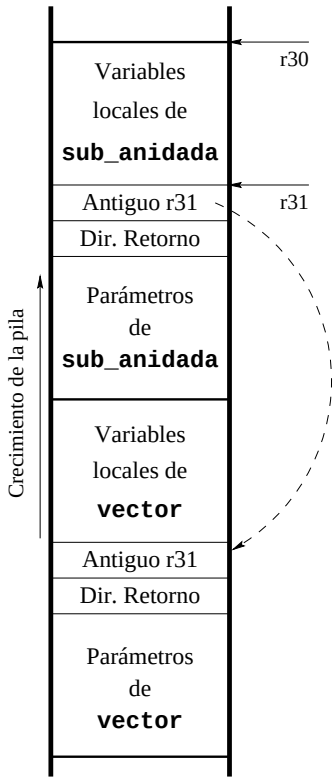

Figura 8: Dos marcos de pila anidados

- Puesto que la ejecución de la subrutina vector ha finalizado, hay que volver a activar el marco de pila de `llamante`. Esto se consigue sacando de la pila el puntero de marco que se salvaguardó al activar el marco de `vector`.
- Por último hay que retornar a `llamante` para lo cual se saca de la pila la dirección de retorno.

Si la subrutina `vector` realiza una llamada a otra subrutina (llamada anidada), se crearía un nuevo marco de pila por encima de la llamada a `vector` tal y como aparece en la figura 8.

3.1. Ordenación de los parámetros y funciones de lenguajes de alto nivel

Tal y como se ha explicado en la sección 2, el paso de parámetros es un protocolo acordado entre el programa llamante y la subrutina. Cuando se escribe una subrutina que va a ser llamada desde un lenguaje de alto nivel, se deberá seguir el protocolo impuesto por el compilador del lenguaje que se esté utilizando. Si se utiliza paso de parámetros en la pila del computador queda claro dónde se pasan los parámetros, pero en el caso de existir varios parámetros se debe establecer el orden en que se pasan.

En el ejemplo de la subrutina `vector`, se pasan dos vectores por dirección `a` y `b`. El orden en que se pasan es el inverso al que aparece en la especificación de la subrutina en

lenguaje de alto nivel, de tal forma que el primero que aparece en la especificación es el que se almacena en la cima de la pila. Los compiladores de lenguajes de alto nivel suelen pasar los parámetros de esta forma. Esto es debido a que en algunas ocasiones se desean escribir subrutinas con un número variable de parámetros (p.e. la impresión de variables con formato). En este caso es necesario que el parámetro que queda en la cima de la pila pueda especificar, de alguna forma, el número de parámetros que le siguen.

En la especificación de la subrutina en lenguaje de alto nivel, el parámetro que va a especificar el número y tamaño de cada uno de los que le siguen es el primero de la lista de argumentos. Esta es la razón por la que éste queda en la cima de la pila tal y como queda representado en las figuras 6 y 7.

Por otra parte, los lenguajes de alto nivel disponen de subrutinas que devuelven un valor de retorno (funciones). Este valor de retorno suele ser una palabra del computador que:

- Indica el estado de ejecución de la subrutina: código de error.
- Se utiliza como una función real que se va a introducir en expresiones aritméticas. Por ejemplo las funciones seno, logaritmo, etc.

En cualquiera de estos dos casos el valor de retorno se va a utilizar inmediatamente después de que la subrutina retorne al punto desde donde se llamó, bien para comprobar el estado de ejecución o bien para seguir calculando la expresión. Por esta razón los compiladores suelen manejar este valor de retorno como un parámetro que se devuelve en los registros del computador.

4. Conclusiones

En esta clase se han expuesto los métodos para el paso de parámetros entre subrutinas más utilizados: en registros, en variables globales y en la pila.

El primero de ellos presenta el inconveniente de no ser general, es decir si se desea pasar un objeto de más de una palabra se deben utilizar varios registros. Esto puede provocar que no existan suficientes registros disponibles para ello.

El segundo método no presenta los problemas anteriores, pero no es una técnica de programación aceptable puesto obliga a que estén visibles un gran número de variables.

El tercer método de paso de parámetros es el más general y el que no genera problemas en cuanto al número de parámetros a pasar a la subrutina y el tamaño de los mismos. Además los parámetros únicamente son válidos durante la ejecución de la subrutina.

Por último se ha expuesto una introducción en cuanto a la forma en que generan código los compiladores de lenguajes de alto nivel: cómo acceden a los parámetros, en qué orden se pasan los parámetros, dónde se ubican las variables locales y cómo se activan los datos privados de una subrutina.

En la próxima clase se expondrán los conceptos de subrutinas reentrantes y recursivas, basándose en ejemplos (para las recursivas) lo que permitirá asentar los conocimientos expuestos en esta lección.