Índice



# Procedimientos y funciones

Área de Arquitectura y Tecnología de Computadores – Versión 1.0.899, 18/02/2022

## Objetivos

Esta sesión prosigue el objetivo de comprender la traducción de programas escritos en lenguaje de alto nivel (C++) al lenguaje del Computador Teórico. Se estudiará en esta sesión la traducción de los procedimientos y sus llamadas. Además, se trabajará con variables locales y con la pila del programa.

## Conocimientos y materiales necesarios

Para aprovechar adecuadamente esta sesión de prácticas, el alumno necesita:

- Conocer los sistemas de codificación binario natural y complemento a 2, así como tener soltura en la conversión a hexadecimal.
- Conocer el lenguaje de programación C++.
- Conocer el lenguaje ensamblador del Computador Teórico.
- Conocer conceptos básicos de traducción de sentencias de alto nivel, tales como sentencias de asignación, aritmético-lógicas, condicionales, bucles y procedimientos, a instrucciones de la máquina.

## 1. Procedimientos

- **Introducción.** Para los ejercicios relacionados con procedimientos, a fin de poder probar si funcionan correctamente mediante el simulador web, será necesario no sólo codificar el procedimiento, sino también una llamada al mismo y verificar si a su retorno los registros y posiciones de memoria tienen los valores esperados. Comenzaremos por tanto por un ejemplo guiado que puede servirte de modelo para los siguientes.

  Se tiene el siguiente código C++ que implementa una función trivial que no recibe parámetros y siempre retorna cero. A la vez, el programa principal `main()` llama a esta función y deja el resultado en una variable llamada `a`.

```
int Cero(void)
{
  return 0;
}
void main(void)
{
  int a;
  a=Cero();
}
```

  A la hora de convertirlo en ensamblador, la función `Cero` será una etiqueta a la que se "saltará" mediante la instrucción `call` desde el programa principal, y que terminará con una instrucción `ret` para volver al punto desde el que se saltó. El `0` que la función debe retornar lo guardará en `r0` antes del `ret`, siguiendo el convenio de que siempre retornaremos el resultado de la función en el registro `r0`. Según esto, la traducción de esta función es muy sencilla:

```
Cero:         ; Etiqueta "nombre" de la función
  movh r0, 0  ; preparamos un 0 en r0 para retornarlo
  movl r0, 0  ;
  ret         ; y saltamos al punto desde donde la función fue llamada
```

  En cuanto al programa principal, en el lenguaje C++ se trata de otra función llamada `main()` y en realidad, como función que es, contiene un retorno (al sistema operativo). Sin embargo, cuando hagamos la conversión a código ensamblador no la trataremos como una función, sino que escribiremos "directamente" su código sin que sea llamado desde ningún sitio ni tenga que retornar a otro.

  En el ejemplo anterior, ya que no consideraremos a `main()` como una función, la variable local `a` tampoco la consideraremos variable local, y no se guardará en la pila. Supondremos que está en una posición de memoria prefijada, igual a `0400h`. Por tanto, el código que llama a la función `Cero()` y guarda en `a` el resultado de esta función sería:

```
principal:    ; Esta etiqueta no es necesaria, es para mayor claridad
  movh r1, 04h
  movl r1, 00h  ; r1 apunta a 0400 que es donde está la variable a
  call Cero     ; Llamamos a la función. El "resultado" viene en r0
  mov  [r1], r0 ; y lo guardamos en a
```

  A fin de probar este código, es necesario juntarlo en un solo listado. Dado que el simulador web siempre comienza ejecutando las instrucciones por la primera que escribamos, la función `Cero()` no puede ir al principio del listado. Si así lo hiciéramos, la ejecución empezaría por esa función, en lugar de hacerlo por el "programa principal" y, cuando alcanzara la instrucción `ret`, ¿a dónde retornaría?

  Por tanto al principio del código ha de ir el programa "principal", como muestra el siguiente listado:

```
principal:    ; Esta etiqueta no es necesaria, es para mayor claridad
  movh r1, 04h
  movl r1, 00h  ; r1 apunta a 0400 que es donde está la variable a
  call Cero     ; Llamamos a la función. El "resultado" viene en r0
  mov  [r1], r0 ; y lo guardamos en a

Cero:         ; Etiqueta "nombre" de la función
  movh r0, 0  ; preparamos un 0 en r0 para retornarlo
  movl r0, 0  ;
  ret         ; y saltamos al punto desde donde la función fue llamada
```

  Sin embargo ténganse en cuenta otro detalle. Cuando el programa ejecute `call Cero`, se producirá un salto hasta la etiqueta `Cero:`, se ejecutará esa función y cuando se encuentre la instrucción `ret` se volverá a la instrucción que hay tras el `call`, es decir, tras el `ret` se ejecutará `mov [r1], r0`. Pero ¿qué ocurrirá después?

  El procesador "no sabe" que el programa ha terminado ahí. Continuará ejecutando por la instrucción siguiente a esa, y resulta que la siguiente instrucción será la primera de la función `Cero`. Es decir, ¡se ejecutará otra vez esta función! Y lo que es peor, cuando se alcance por segunda vez el `ret` no hay a dónde retornar, ya que esta segunda vez no ha sido llamada realmente mediante `call`.

  Es necesario indicar de alguna forma al procesador que el programa principal ha finalizado. En un procesador real, con un sistema operativo, se colocaría ahí una instrucción que retornara al sistema operativo (por eso en el C++, `main()` es otra función, que retorna al operativo cuando termina). En nuestro simulador del Computador Teórico no hay operativo, por lo que no hay a dónde retornar.

  Por suerte el simulador tiene implementado que, cuando la instrucción a ejecutar sea `nop` (de *No operation*, una operación que no hace nada), el simulador debe detenerse y dar el programa por finalizado. Esta es la razón por la que todos los ejercicios que hemos hecho hasta ahora se detenían al terminar, y es que todas las posiciones de la memoria en las que no hayamos puesto algo, están por defecto rellenas con `0000h`, que es el código máquina de la instrucción `nop`. Por eso, al final de cada uno de los ejercicios anteriores había instrucciones `nop` aunque no las hubiéramos escrito.

  En este caso, para forzar a que el simulador se detenga una vez haya finalizado el programa principal, escribiremos explícitamente la instrucción `nop` al final del mismo. El código completo queda por tanto así:

```
principal:    ; Esta etiqueta no es necesaria, es para mayor claridad
  movh r1, 04h
  movl r1, 00h  ; r1 apunta a 0400 que es donde está la variable a
  call Cero     ; Llamamos a la función. El "resultado" viene en r0
  mov  [r1], r0 ; y lo guardamos en a
  nop           ; FIN DEL PROGRAMA

Cero:         ; Etiqueta "nombre" de la función
  movh r0, 0  ; preparamos un 0 en r0 para retornarlo
  movl r0, 0  ;
  ret         ; y saltamos al punto desde donde la función fue llamada
```

  Puedes probar a ejecutar este código en el simulador web. Activa el modo traza para ver cómo se produce la llamada y el retorno. Inicializa la dirección `0400h` (variable `a`) con un valor distinto de cero y activa la casilla "Mirar", para comprobar cómo al final de la ejecución esa posición ha cambiado de valor y es cero.

  En los ejercicios que siguen deberás ceñirte al esquema anterior, es decir, programa principal al principio del listado, que se ocupe de ajustar los parámetros si los hubiere (en el ejemplo anterior no los hay), llamar a la función, eliminar los parámetros incrementando `r7`, guardar el resultado en la variable apropiada y terminar con `nop`, y tras éste, vendría el código de la función a implementar.

  **Importante.** Como sabes, la pila utiliza *implícitamente* el registro r7, por lo que éste irá cambiando de valor a lo largo del programa (con cada `push`, `pop`, `call` y `ret`). Sin embargo, al finalizar la ejecución, el valor final de `r7` ha de ser igual al que tenía al principio, pues de no ser así ello significaría que la pila no ha sido usada de forma equilibrada. Asegúrate de que así es en todos los casos (lo cual es sencillo porque si `r7` cambia de valor aparecerá en rojo).

- **Ejercicio 15.** Implementa la función `Max()` que recibe dos números naturales y retorna el valor del mayor de ellos. Implementa también el programa principal que llama a esa función, como se muestra en el siguiente código C++. Las variables deben asignarse a los registros que se indican en los comentarios.

```
unsigned int Max(unsigned int a, unsigned int b)  // a y b son parámetros en la pila,
                                                  // apilados de dcha. a izda.
{
  if (a>b)
    return a;          // retornado en r0
  else
    return b;
}
void main(void)
{
    unsigned int x=5, y=9;      // x en r1, y en r2
    unsigned int resultado;     // resultado en memoria, dirección 0500h

    resultado=Max(x, y);
}
```

  Comprueba en el simulador, trazando el contenido de la dirección `0500h` y comprobando que al final contiene `9`, ya que éste es el mayor entre `x` e `y`.

- **Ejercicio 16.** Implementa la función `MaxMin()` que recibe dos números naturales y retorna por separado el valor del mayor y del menor de ellos. Para poder retornar dos resultados, recibirá dos parámetros por referencia. Ya que los resultados los "devuelve" en esos parámetros, no necesita retornar nada a través del `return`, por lo que la función se declara de tipo `void` (lo cual en ensamblador implica que no va a devolver nada en `r0`).

  Traduce al ensamblador del Computador Teórico el siguiente programa en C++, siguiendo las indicaciones de los comentarios para mapear las variables en registros o memoria.

```
void MaxMin(unsigned int a, unsigned int b, unsigned int& max, unsigned int& min)
{  // a, b, max y min son parámetros en la pila, apilados de dcha. a izda.
  if (a>b) {
    max=a;      // Observar que max y min se han pasado por referencia
    min=b;      // En ensamblador se usa la dirección a que apuntan estos parámetros
  }
  else {
    max=b;
    min=a;
  }
}
void main(void)
{
  unsigned int x=5, y=9;        // x en r1, y en r2
  unsigned int mayor, menor;    // ambas en memoria, mayor en 0500h, menor en 0501h
  MaxMin(x, y, mayor, menor);
}
```

  Una vez traducido a ensamblador, ejecutarlo en el simulador web, "mirando" las direcciones `0500h` y `0501h` y comprobando que una vez finalizado el programa cada una contiene, respectivamente, el mayor y el menor de los datos.

## 2. Ejercicios adicionales

Intenta realizar ahora los siguientes ejercicios que tienen algo más de dificultad:

- **Ejercicio 17.** El siguiente código en C++ implementa una función llamada `Maximo()` que recibe dos parámetros. El primero es un array pasado por referencia. El segundo es un entero que indica el tamaño del array (pasado por copia). La función recorre el array para encontrar el elemento de mayor valor, y retorna el valor encontrado.

  Para probar la función, se escribe un programa principal que tiene un array con 5 elementos, inicializados como se indica en la declaración del array.

```
unsigned int Maximo(unsigned int datos[], unsigned int cuantos)
{
   unsigned int i;   // i en registro r5
   unsigned int mayor;  // mayor en registro a elegir por el alumno

   mayor=0;
   for (i=0; i<cuantos; i++)
     if (datos[i]>mayor)     // datos[i] en r4
       mayor=datos[i];
   return mayor;
}

void main()
{
   unsigned int d[5]={3, 2, 9, 1, 4};
   unsigned int n=5;                  // n en r1
   unsigned int resultado;            // resultado en r2

   resultado=Maximo(d, n);
}
```

  En tu traducción a ensamblador, el array estará almacenado a partir de la dirección `0600h`. Por tanto, usando el simulador web, debes inicializar las posiciones de memoria `0600h`, `0601h`, `0602h`, `0603h` y `0604h` con los valores `3`, `2`, `9`, `1` y `4`, respectivamente. No es necesario "mirar" estas posiciones de memoria, ya que en teoría no cambian de valor durante la ejecución del programa. No obstante, si algo no funciona bien puedes activar la casilla "mirar" para comprobar que efectivamente no cambien.

  Tras ejecutar tu programa, `r2` (que contiene el resultado) ha de tomar el valor `9`, ya que este es el máximo entre los datos. Puedes probar a ejecutar varias veces tu código, cambiando los valores iniciales que hay en memoria, o incluso el valor inicial de `r1` (que es el que almacena la variable `n` que dice cuántos elementos tiene el array), reduciéndolo a valores inferiores a `5`, para comprobar que la función busca el máximo sólo entre los primeros `n` elementos. Por ejemplo, si con los datos anteriores `n` fuera 2, el resultado sería `3`, ya que el máximo entre los dos primeros elementos es `3`.

- **Ejercicio 18.** Se tiene un array conteniendo varios números naturales, y se quiere buscar dentro del array la primera aparición de un dato dado. Para ello se implementa una función `Busca()` que recibe el array (por referencia), un número natural indicando cuántos elementos tiene el array, y otro número natural que es el dato que se busca. La función retorna, en un número entero, el índice del primer elemento cuyo valor coincida con el dato buscado. Si el dato buscado no aparece, la función retorna `-1` (`FFFFh`).

  El siguiente ejemplo muestra cómo se llamaría a la función, que en este caso retornaría 3, ya que el dato buscado aparece en el elemento 3 del array (recuerda que los índices de los arrays comienzan en `0`, por lo que el elemento de índice `3` es en realidad el cuarto elemento del array).

```
void main()
{
   unsigned int datos[]={1, 7, 2, 4, 5};
   unsigned int n = 5;              // número de elementos del array, r1
   unsigned int busco = 4;          // dato buscado, r2
   int encontrado;                  // resultado, r3

   encontrado=Buscar(datos, n, busco);
}
```

  La implementación de la función en C++ sería como sigue:

```
int Buscar(unsigned int dat[], unsigned int cuantos, unsigned int cual)
{
   unsigned int i;              // Índice de bucle, r1
   for (i=0; i<cuantos; i++)
     if (dat[i]==cual)      // Si se encuentra
       return i;            // retornamos la posición
   // Si hemos llegado aquí sin encontrarlo, retornamos -1
   return -1;
}
```

  Traduce al ensamblador del Computador Teórico la función anterior y el programa principal de prueba. Ejecútalo en el simulador web con los datos suministrados (que guardarás en memoria a partir de la posición `0500h`) y pruébalo también con otros datos, con datos repetidos, buscando otros números, etc. para asegurarte de que funciona en todos los casos.