

ESTRUCTURA DE COMPUTADORES

GRADO EN INGENIERÍA INFORMÁTICA



UNIVERSIDAD CARLOS III DE MADRID

Práctica 3

Ensamblador de MIPS: Paso de parámetros entre subrutinas

Noviembre 2013



1. Objetivos de la práctica

El objetivo de esta práctica es que el alumno se familiarice con la programación en ensamblador y el convenio de paso de parámetros a subrutinas en el ensamblador del MIPS32.

Para alcanzar este objetivo el estudiante realizará la codificación en lenguaje ensamblador de una serie de funciones de las que se proporciona su código fuente en lenguaje C.

Otro objetivo de la práctica es el trabajo en grupo, para ello, los alumnos deben repartirse las tareas y las funciones a desarrollar y posteriormente describir este reparto en el Ejercicio 4

NOTA: Las subrutinas pedidas deben seguir OBLIGATORIAMENTE el nombre indicado en las siguientes secciones. Se ha de tener en cuenta que un nombre con mayúsculas es diferente de otro con minúsculas. Por ejemplo, los siguientes nombres corresponden a subrutinas diferentes:

imprimir_entero, Imprimir_Entero, imprimir_ENTERO, etc.

Observe que los nombres de función que se piden en esta práctica tienen todas sus letras en MINÚSCULAS.

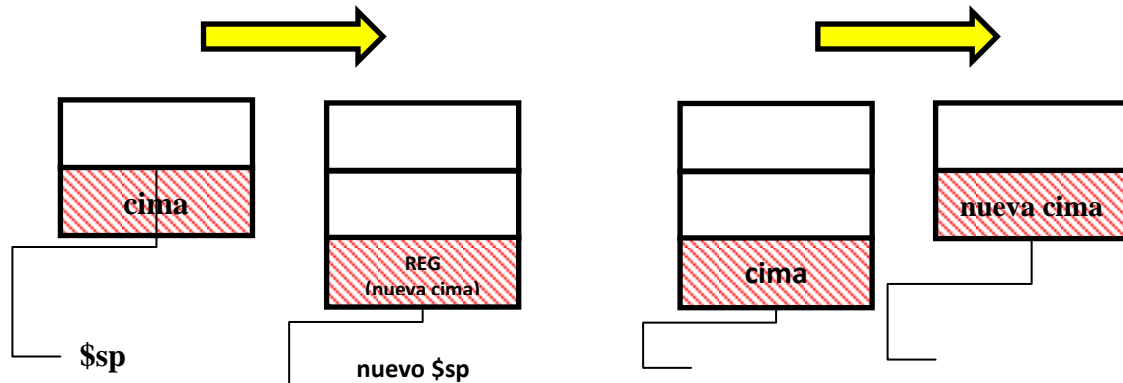


2. Convenio de paso de parámetros entre rutinas

Operaciones básicas: apilar y desapilar

Apilar
sub \$sp, \$sp, 4
sw \$REG, (\$sp)

Desapilar
lw \$REG, (\$sp)
add \$sp, \$sp, 4



La pila crece hacia direcciones de memoria más bajas.

\$p apunta al último dato en la pila. (OJO con esto: al último dato, NO al primer hueco libre).

Convenio de uso de registros

- **Parámetros no flotantes:** \$a0, \$a1, \$a2, \$a3, ---- si hay más de 4, el resto en la pila ----
- **Parámetros flotantes:** \$f12, \$f13, \$f14, \$f15, ---- si hay más de 4, el resto en la pila ----
- **Parámetros flotantes doble precisión:** \$f12, \$f14, ---- si hay más de 2, el resto en la pila ----
- **Resultados no flotantes:** \$v0, \$v1, ---- si hay más de 2, el resto en la pila ----
- **Resultados flotantes:** \$f0, ---- si hay más de 1, el resto en la pila ----

Siempre que se pueda se deben usar registros, ya que es más rápido que usar la pila.

Paso de parámetros con tipos de datos no básicos

- **Cadenas de caracteres:** se pasa la dirección de comienzo de la cadena.
- **Arrays unidimensionales:** se pasa la dirección de comienzo del array y el número de elementos
- **Arrays bidimensionales (solo en la práctica):** se pasa la dirección de comienzo del array, el número de filas, y el número de columnas.

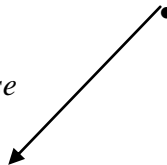
Rutina A que llama a rutina B (B, terminal)

Rutina A

- Antes de llamar a B



- Apilar $\$a$ que vaya a utilizar
- Apilar $\$t$ que quiera conservar
- Apilar $\$ra$
- Pasar parámetros en $\$a$
- Apilar (si no caben en regs) resto de parámetros de B
- **Llamar a B**
 - *jal B* <Pasa a ejecutarse B>
- **Después de llamar a B**
 - Desapilar parámetros de B (si hay)
 - Desapilar $\$ra$
 - Desapilar $\$t$ (si se apilaron)
 - Desapilar $\$a$
- Apilar registros (diferentes de $\$t$ y $\$v$) que se vayan a modificar (*p.ej. $\$s$*)
- Acceder a los parámetros de la pila (sin desapilar)
- <Ejecutar su código>
- Desapilar registros que se hayan utilizado al principio (*p. ej $\$s$*)
- Retornar (*jr $\$ra$*)



Rutina B



Rutina A que llama a rutina C que a su vez llama a rutina B

A y B igual que antes, pero A invocando a C en lugar de a B ($A \rightarrow C \rightarrow B$)

C debe realizar los pasos del convenio de A y de B.

Rutina C

- Apilar registros (diferentes de $\$t$ y $\$v$) que se vayan a modificar (*p.ej. $\$s$*)
- Acceder a los parámetros de la pila (sin desapilar)
- *<Ejecutar su código>*
- Apilar $\$a$ que vaya a utilizar
- Apilar $\$t$ que quiera conservar
- Apilar $\$ra$
- Pasar parámetros en $\$a$
- Apilar (si no caben en regs) resto de parámetros de B
- **Llamar a B (*jal B*)**
- Desapilar parámetros de B (si hay)
- Desapilar $\$ra$
- Desapilar $\$t$
- Desapilar $\$a$
- *<Seguir ejecutando su código>*
- Desapilar registros que se hayan utilizado al principio (*p. ej $\$s$*)
- Retornar a A (*jr $\$ra$*)

3. Ejercicios a desarrollar

Recuerde que las rutinas implementadas tienen que seguir el convenio de paso de parámetros que se ha indicado anteriormente. En caso contrario, la práctica tendrá la calificación de 0.

Ejercicio 1

Implementar un programa **matrix.s** que contenga las siguientes subrutinas. En el material de apoyo de la práctica 3, que puede descargar en Aula Global 2, se dispone del fichero *matrix.s*, del que debe partir el alumno.

```
int set(int[][] matrix, int n_rows, int n_cols, int row, int col, int value) {
    if(row < n_rows && col < n_cols) {
        matrix[row][col] = value;
        return 0;
    } else {
        return -1;
    }
}

int* get(int[][] matrix, int n_rows, int n_cols, int row, int col) {
    int* res = (int*)malloc(2*sizeof(int));
    if(row < n_rows && col < n_cols) {
        res[0] = matrix[row][col];
        res[1] = 0;
    } else {
        res[0] = 0;
        res[1] = -1;
    }

    return res;
}
```

El código que se muestra presenta dos operaciones de acceso a una matriz, get y set. En el caso de la función set, se devolverá un 0 si el cambio ha podido realizarse y un -1 en caso contrario. La función get devolverá dos enteros. El primero es el valor al que se quería acceder, y el segundo un código de finalización. Si la ejecución es correcta, se devolverá el valor solicitado y un 0. Si la ejecución es incorrecta, se devolverá un 0 y un -1 para indicar que el resultado no es válido.

NOTA: el programa debe funcionar independientemente de los datos declarados en la zona de `.data`.

Ejercicio 2

Implementar un programa **operations.s** que contenga las siguientes subrutinas. En el material de apoyo de la práctica 3 se dispone del fichero *operations.s*, del que debe partir el alumno.

```
int[][] sumar(int[][] matrixA, int[][] matrixB, int n_rows, int n_cols) {
    int i, j, tmp;
    int *aux1, *aux2;
    int[][] res = (int[][])malloc(n_rows * sizeof(int*));
    for(i = 0; i < n_rows; i++) {
        res[i] = (int*)malloc(n_cols * sizeof(int));
    }

    for(i = 0; i < n_rows; i++) {
        for(j = 0; j < n_cols; j++) {
            aux1 = get(matrixA, n_rows, n_cols, i, j);
            aux2 = get(matrixB, n_rows, n_cols, i, j);
            if(aux1[1] == 0 && aux2[1] == 0) {
                tmp = aux1[0] + aux2[0];
                set(res, n_rows, n_cols, i, j, tmp);
            }
        }
    }

    return res;
}

int[][] multiplicar(int[][] matrixA, int[][] matrixB, int n_rowsA, int n_colsA, int n_rowsB, int n_colsB) {
    int i, j, k, tmp = 0;
    int *aux1, *aux2;
    int[][] res = (int[][])malloc(n_rowsA * sizeof(int*));
    for(i = 0; i < n_rowsA; i++) {
        res[i] = (int*)malloc(n_colsB * sizeof(int));
    }

    for(i = 0; i < n_rowsA; i++) {
        for(j = 0; j < n_colsB; j++) {
            for(k = 0; k < n_colsA; k++) {
                aux1 = get(matrixA, n_rowsA, n_colsA, i, k);
                aux2 = get(matrixB, n_rowsB, n_colsB, k, j);
                if(aux1[1] == 0 && aux2[1] == 0) {
                    tmp = tmp + aux1[0] * aux2[0];
                }
            }
            set(res, n_rowsA, n_colsB, i, j, tmp);
            tmp = 0;
        }
    }

    return res;
}
```

Las subrutinas a desarrollar son sumar y multiplicar, que deberán realizar la suma y multiplicación de dos matrices respectivamente, dadas sus dimensiones. Los parámetros y valores de retorno deben seguir el orden que se muestra en la imagen anterior.

Para desarrollar este programa deberá utilizar reserva dinámica de memoria, que se corresponde con la llamada al sistema 9. El resultado equivalente en ensamblador deberá ser la dirección de comienzo de la matriz resultado.

También deberá emplear las subrutinas get y set del ejercicio anterior en el acceso a las matrices.

NOTA: el programa debe funcionar independientemente de los datos declarados en la zona de `.data`.

Ejercicio 3

Implementar un programa **axpy.s** que contenga la subrutina descrita a continuación. En el material de apoyo de la práctica 3 se dispone del fichero *axpy.s*, del que debe partir el alumno.

Se desea desarrollar una subrutina que realice la operación $A * X + Y$, donde A, X e Y son matrices. Se deberán utilizar las funciones desarrolladas previamente para la realización del ejercicio. Un ejemplo de código es el siguiente:

```
int[][] matrixA = {{1,1},{2,2},{3,3},{4,4}};
int[][] matrixX = {{1,1,1},{2,2,3}};
int[][] matrixY = {{1,1,1},{2,2,2},{3,3,3},{4,4,4}};
int n_rowsA = 4;
int n_rowsB = 2;
int n_colsA = 2;
int n_colsB = 3;

int[][] axpy(int[][] matrixA, int[][] matrixX, int[][] matrixY, int n_rowsA, int n_colsA, int n_rowsB, int n_colsB) {
    int[][] mul = multiplicar(matrixA, matrixX, n_rowsA, n_colsA, n_rowsB, n_colsB);
    int[][] res = sumar(mul, matrixY, n_rowsA, n_colsB);

    return res;
}
```

La salida en este caso será:

```
11 11 11 11
22 22 22 22
33 33 33 33
44 44 44 44
```

NOTA: el programa debe funcionar independientemente de los datos declarados en la zona de `.data`.

Ejercicio 4

En este ejercicio se solicita la creación de un plan de trabajo detallado. Los alumnos deberán detallar como mínimo los siguientes puntos:

- Reparto de tareas entre los miembros del grupo, tanto si se ha realizado a nivel de ejercicio o por funciones.
- Planificación del tiempo de trabajo.
- Comunicación entre los miembros del grupo (reuniones semanales, trabajo en mismo lugar...)
- Herramientas utilizadas para la comunicación entre los miembros y envío del trabajo.
- Método de unión de trabajo.
- Plan de pruebas.
- Evaluación del resultado de la planificación.

NOTA: es obligatorio la realización de este ejercicio para aprobar la práctica, será evaluado como APTO o NO APTO.

Ejercicio 5

Implementar un programa **axpy_flat.s** que desarrolle un comportamiento equivalente al de *axpy.s* (con todas las subrutinas) SIN utilizar la pila para guardar ningún dato. Puede reservar espacio en la zona de datos o en memoria dinámica. En el material de apoyo de la práctica 3 se dispone del fichero *axpy_flat.s*, del que debe partir el alumno.

Adicionalmente, deberá incluir en la memoria una comparación entre ambas versiones, detallando el número de registros utilizados, el número de líneas totales de la aplicación y el tamaño total utilizado en la zona de datos (si la cantidad varía en función del tamaño de las matrices, indíquelo de forma genérica).

NOTA: el programa debe funcionar independientemente de los datos declarados en la zona de `.data`.

Procedimiento de evaluación de la práctica

La evaluación de la práctica se va a dividir en tres partes.

- **Parte Obligatoria (8 puntos)**

En esta parte se incluyen los ejercicios 2 y 3 de la práctica, así como la memoria de la misma. El reparto de puntuación dentro de la entrega obligatoria será el siguiente:

- Ejercicio 1 (2 puntos)
- Ejercicio 2 (3 *puntos*)
- Ejercicio 3 (2 *puntos*)
- Ejercicio 4 (APTO o NO APTO)

No descuide la calidad de este ejercicio, un NO APTO en él supone un suspenso en la práctica

- Memoria (1 *punto*)

- **Parte Opcional (2 puntos)**

El ejercicio 5 es de realización opcional. El reparto de puntuación dentro de esta entrega opcional será el siguiente:

- Ejercicio 5 (1.5 *puntos*)
- Memoria (0.5 puntos)

Procedimiento de entrega

La entrega de la práctica 3 se realizará de la siguiente manera:

- La fecha de entrega de estos ejercicios podrá consultarse en Aula Global, en los entregadores habilitados a los efectos.
- **Entregador 1:** Se deberá entregar un único archivo comprimido en formato zip con el nombre:

ec_p3_AAAAAAAAAA_BBBBBBBBBB.zip

donde A...A y B...B son los NIAs de los integrantes del grupo. El archivo zip debe contener:

- **matrix.s**
- **operations.s**
- **axpi.s**
- **axpi_flat.s**

- **Entregador 2:** Se deberá entregar un único archivo pdf con el nombre:

ec_p3_AAAAAAAAAA_BBBBBBBBBB.pdf

donde A...A y B...B son los NIAs de los integrantes del grupo. La memoria debe contener al menos los siguientes apartados:

- Portada donde figuren los autores (incluyendo nombre completo, NIA y dirección de correo electrónico)
- Índice de contenidos.
- Descripción de los programas solicitados.
- Respuesta al ejercicio 4.
- Batería de pruebas utilizadas y resultados obtenidos.
- Conclusiones y problemas encontrados.

NOTA: NO DESCUIDE LA CALIDAD DE LA MEMORIA DE SU PRÁCTICA.

Aprobar la memoria es tan imprescindible para aprobar la práctica, como el correcto funcionamiento de la misma. Si al evaluarse la memoria de su práctica, se considera que no alcanza el mínimo admisible, su práctica estará suspensa.

La longitud de la memoria no deberá superar las 20 páginas (portada e índice incluidos).

- La entrega de las prácticas ha de realizarse de forma electrónica. En AULA GLOBAL se habilitarán unos enlaces a través de los cuales podrá realizar la entrega de las prácticas.
- La única versión registrada de su práctica es la última entregada. La valoración de esta es la única válida y definitiva.

Normas

1. Las prácticas que no compilen o que no se ajusten a la funcionalidad y requisitos planteados, obtendrán una calificación de 0.
2. Un programa no comentado, obtendrá una calificación de 0.
3. La entrega de la práctica se realizará a través de los entregadores habilitados. No se permite la entrega a través de correo electrónico sin autorización previa.
4. Se prestará especial atención a detectar funcionalidades copiadas entre dos prácticas. En caso de encontrar implementaciones comunes en dos prácticas, ambas obtendrán una calificación de 0.