ESTRUCTURA DE COMPUTADORES

GRADO EN INGENIERÍA INFORMÁTICA

UNIVERSIDAD CARLOS III DE MADRID

Grupo de Arquitectura de Computadores

Práctica 2
Programación en ensamblador

Curso 2015/2016

Contenido

Objetivos de la práctica

El objetivo de la práctica es que el alumno se familiarice con la programación en ensamblador, de forma específica con el ensamblador del MIPS32. Para el desarrollo de esta práctica se **usará** el emulador QTSPIM disponible en:

http://spimsimulator.sourceforge.net/

QTSPIM es un simulador auto-contenido que permite ejecutar programas escritos en el lenguaje ensamblador del MIPS32. QTSPIM proporciona además un depurador y un conjunto mínimo de servicios del sistema operativo.

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.

Descripción del simulador QTSPIM

La versión QTSPIM (en su versión para Windows) posee una interfaz similar a la que aparece en la siguiente figura. Hay dos paneles horizontales. El panel de la izquierda contiene los valores de los registros, y el panel de la derecha contiene el código y los datos declarados.

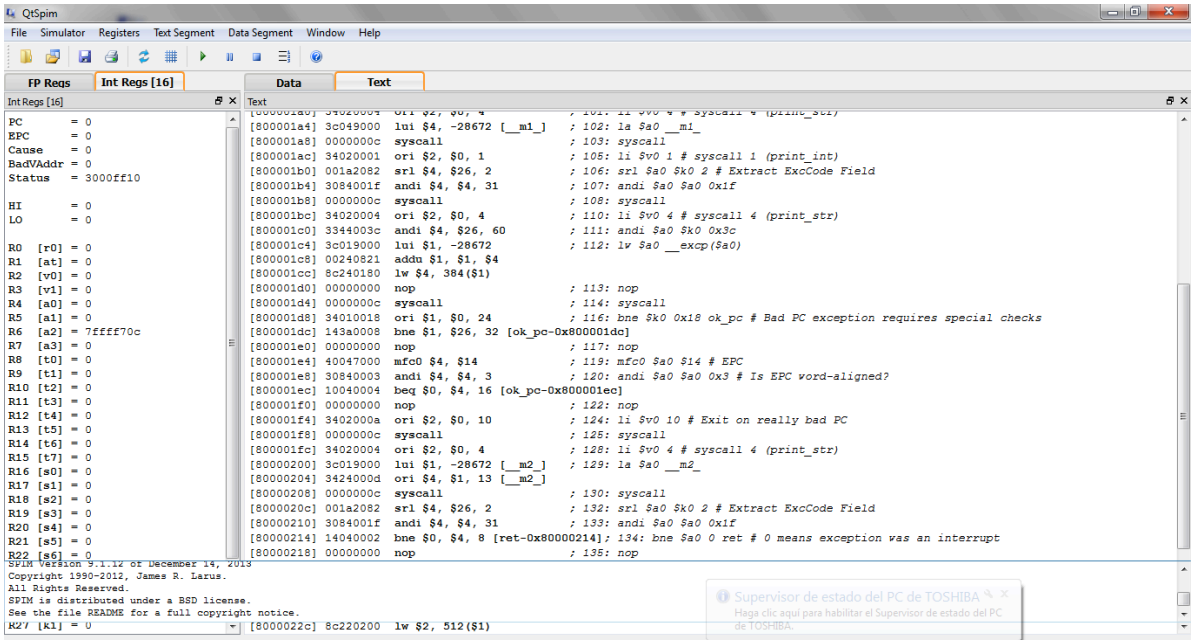
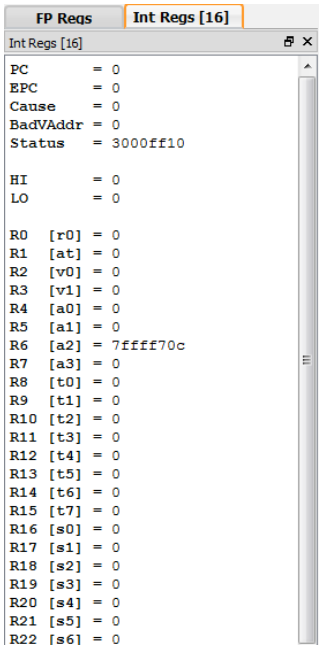


Figura 1. Interfaz del simulador QTSPIM

PANEL 1. Registros de la máquina MIPS.

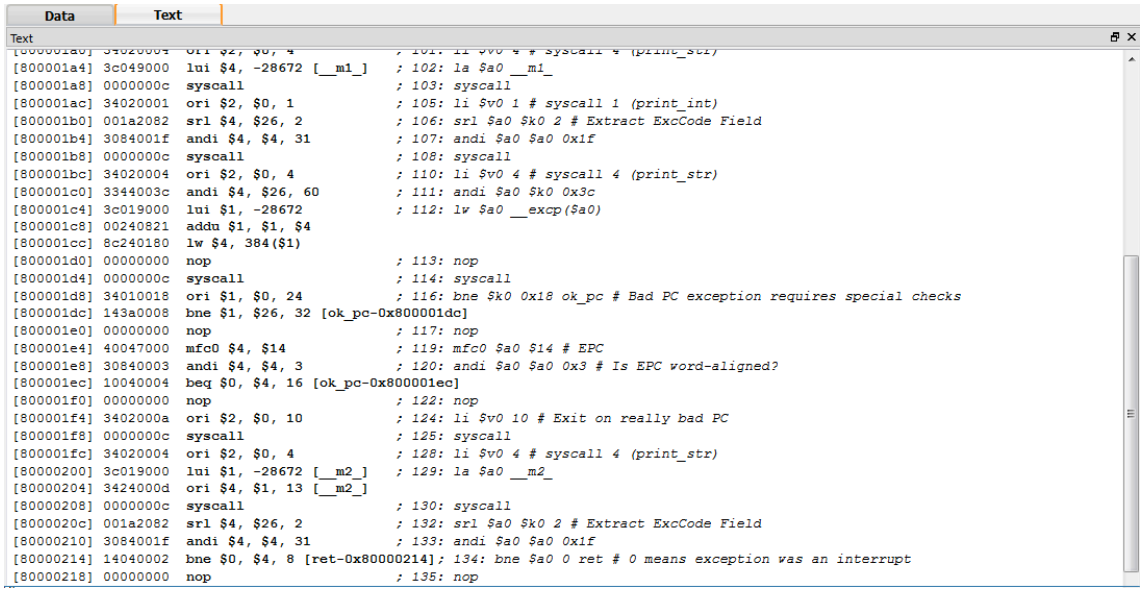
Incluye el banco de registros del procesador MIPS. En primer lugar aparece el contador de programa (**PC**) y otros registros especiales como **HI** y **LO** utilizados para almacenar los resultados de la multiplicación y división.
En las siguientes líneas aparecerán los 32 registros enteros de propósito general (**R0** a **R31**), junto a su correspondiente etiqueta simbólica.



En la pestaña FP Regs aparecen los 32 registros usados para representar números en coma flotante, tanto en precisión simple, como en precisión doble (**FP0** a **FP31**).

PANEL 2. Segmento de Texto.

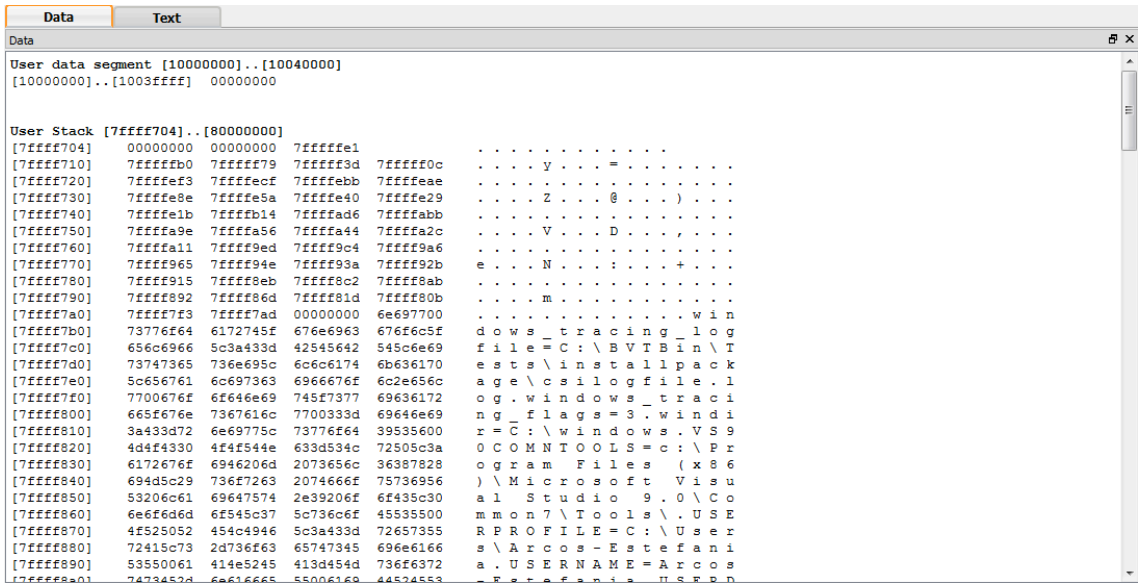
Incluye el código del programa ensamblador a ejecutar, identificado mediante la directiva *.text*. Cada instrucción ocupa una única línea, identificada por una dirección de memoria que aparece en la parte izquierda de la línea.



La primera dirección de memoria utilizada para almacenar el texto de un programa es 0x00400000.

PANEL 3. Segmento de Datos y pila.

Incluye el conjunto de datos definidos en el segmento de datos *.data* del programa a ejecutar. Estos datos se almacenan de manera secuencial en memoria, a partir de la dirección **0x10000000.** A continuación se muestra el contenido de la pila (*stack*) que comienza en la dirección **0x7FFFEFFF**.



PANEL 4. Pantalla para depuración de programas

El último panel es el depurador (situado en la parte inferior de la Figura 1), destinado a visualizar mensajes de control o error del simulador QTSPIM.

Ejercicio 1

El objetivo de este ejercicio es desarrollar una pequeña biblioteca de funciones escritas en ensamblador que permiten trabajar con números representados en coma flotante según el estándar IEEE 754 de 32 bits (variables de tipo `float` en C). Las funciones a implementar son las siguientes. Tenga en cuenta que la correspondiente rutina en ensamblador debe tener el mismo nombre.

- `int` **`iszero`**`(float x)`: devuelve 1 si el número `x` es cero y 0 en caso contrario. La implementación realizada NO puede comparar x con el valor 0.0.
- `int` **`isinfpos`**`(float x)`: devuelve 1 si el número `x` es más infinito y 0 en caso contrario. La implementación realizada NO puede comparar x con el valor Inf.
- `int` **`isinfneg`**`(float x)`: devuelve 1 si el número `x` es menos infinito y 0 en caso contrario. La implementación realizada NO puede comparar x con el valor –Inf.
- `int` **`isnan`**`(float x)`: devuelve 1 si el número `x` es `NaN` y 0 en caso contrario. La implementación realizada NO puede comparar x con el valor NaN.
- `int` **`isnormal`**`(float x)`: devuelve 1 si el número `x` se corresponde con un número normalizado y 0 en caso contrario.
- `int` **`isunnormal`**`(float x)`: devuelve 1 si el número `x` se corresponde con un número no normalizado y 0 en caso contrario.
- `int` **`split`**`(float x, int *s, int *e, int *m)`: la función descompone los 32 bits del número `x`, devolviendo tres valores: el signo en `s`, el exponente almacenado en el número `e` y la mantisa almacenada en `m` . El valor del signo se almacenará en el bit menos significativo de `s`; el valor del exponente en los 8 bits menos significativos de `e` y el valor de la mantisa en los 23 bits menos significativos de `m`. En la implementación a realizar tenga en cuenta que esta función recibe un único parámetro de entrada y devuelve tres valores (el signo, el exponente y la mantisas).
- `int` **`join`**`(int s, int e, int m, float *x)`: la función construye un número en coma flotante y lo devuelve en `x`, a partir del bit de signo `s`, del exponente a almacenar `e` y de la mantisa a almacenar `m`. El bit de signo se encuentra en el bit menos significativo de `s`; el exponente en los 8 bits menos significativos de `e`; y la mantisa en los 23 bits menos significativos de `m`. En la implementación a realizar tenga en cuenta que esta función recibe tres parámetro de entrada y devuelve un único valor de tipo float.

Para el desarrollo de esta biblioteca de funciones ha de seguirse estrictamente el paso de parámetros visto en la asignatura. Tenga en cuenta que las variables de tipo entero se almacenan en el banco de registros del procesador de propósito general y que las variables de tipo `float` se almacenan en los registros del banco de registro de coma flotante. Por tanto los argumentos de tipo entero a pasar a las funciones en los registros `$a` correspondiente y las variables de tipo `float` en los registros `$f` correspondientes del banco de registro del coprocesador en coma flotante.

Código a entregar:

- Archivo `p2-e1.s` con el código solicitado.
- Archivo `p2-e1-pruebas.s`. Este archivo incluirá un programa principal (función `main`) que incluirá el código de todas las pruebas utilizadas para probar las funciones desarrolladas en el otro archivo.

Documentación a entregar del ejercicio **en la memoria** final:

- Descripción de la implementación realizada para cada función, incluyendo el pseudocógido utilizado, así como descripción de las pruebas realizadas cuyo código aparece en `p2-e1-pruebas.s`.

El archivo p2-e1.s a entregar tendrá el siguiente formato, de forma que cada grupo de prácticas deberá desarrollar el código para cada función.

```
iszero:     # código a incluir
            jr    $ra

isinfpos:   # código a incluir
            jr    $ra

isinfneg:   # código a incluir
            jr    $ra

isnan:      # código a incluir
            jr    $ra

isnormal:   # código a incluir
            jr    $ra

isunnormal: # código a incluir
            jr    $ra

split:      # código a incluir
            jr    $ra

join:       # código a incluir
            jr    $ra
```

El archivo `p2-e1-pruebas.s` a entregar tendrá el siguiente formato, de forma que cada grupo de prácticas deberá desarrollar el código que estime oportuno para probar las funciones anteriormente desarrolladas.

```
.data:
             # segmento de datos a utilizar para las pruebas

.text:

      main:  # código utilizado para probar las funciones anteriores
```

Ejercicio 2

El objetivo de este segundo ejercicio es desarrollar dos funciones que trabajan con matrices y hacen uso de las funciones desarrolladas en el ejercicio 1. Se trata de implementar el código para las dos siguientes funciones, cuyo prototipo en C es el siguiente:

- `int computenormalized(float **matrix, int N, int M);`
- Esta función recibe 3 argumentos de entrada: una matriz de números en coma flotante de tipo `float` de dimensión `NxM.` La función devuelve el número de elementos de la matriz que se corresponden con números normalizados. Tenga en cuenta que en C, para pasar una matriz a una función, lo que se pasa es su dirección de inicio.
- `int buildmatrix(int **s, int **e, int **m, int N, int M, float **matrix);`

  Esta función recibe 5 argumentos de entrada: tres matrices de tipo entero (`s`, `e` y `m`) y dos valores que representan las dimensiones de estas tres matrices (N y M), de forma que las dimensión de cada matriz es `NxM` (filas x columnas). La función devuelve como único valor de salida la dirección donde se almacena una matriz de elementos de tipo `float` de dimensión `NxM.` Tenga en cuenta que en C, para pasar una matriz a una función, lo que se pasa es su dirección de inicio.

  La función construye cada elemento de la matriz `matrix[i][j]` a partir del valor de signo (`s[i][j]`), exponente (`e[i][j]`) y mantisa (`m[i][j]`) que se encuentran en las matrices pasadas en los tres primeros argumentos. El valor del exponente es el exponente a almacenar directamente y el valor de la mantisa correspondiente es el valor a almacenar directamente (no se necesita ningún procesamiento adicional, es decir, no hay que añadir bit implícito, etc.).

  El elemento `s[i][j]` almacena el bit de signo del número a almacenar en `matrix[i][j]`. El bit de signo se almacena en el bit menos significativo de elemento `s[i][j]`. El elemento `e[i][j]` almacena los 8 bits del exponente a almacenar en `matrix[i][j]`. El exponente se encuentra almacenado en los 8 bits menos significativos del elementos `e[i][j]`. Por último, el elemento `m[i][j]` almacena los 23 bits de la mantisa a almacenar en `matrix[i][j]`. El valor de la mantisa se encuentra almacenado en los 23 bits menos significativos del elemento `m[i][j]`.

Código a entregar:

- Archivo `p2-e2.s` con el código pedido.
- Archivo `p2-e2-pruebas.s`. Este archivo incluirá un programa principal (función `main`) que incluirá el código de todas las pruebas utilizadas para probar las funciones desarrolladas en el otro archivo.

Documentación a entregar del ejercicio **en la memoria** final:

- Descripción de la implementación realizada para cada función, incluyendo el pseudocódigo utilizado, así como descripción de las pruebas realizadas cuyo código aparece en `p2-e2-pruebas.s`.

El archivo `p2-e2.s` a entregar tendrá el siguiente formato, de forma que cada grupo de prácticas deberá desarrollar el código para cada función.

```
computenormalized:      # código a incluir
                        jr    $ra

buildmatrix:            # código a incluir
                        jr    $ra
```

El archivo `p2-e2-pruebas.s` a entregar tendrá el siguiente formato, de forma que cada grupo de prácticas deberá desarrollar el código que estime oportuno para probar las funciones anteriormente desarrolladas.

```
.data:
            # segmento de datos a utilizar para las pruebas

.text:

      main:  # código utilizado para probar las funciones anteriores
```

Aspectos importantes a tener en cuenta

Normas generales

1) La entrega de la práctica se realizará a través de los entregadores habilitados. No se permite la entrega a través de correo electrónico.
2) La entrega se realizará en el plazo dado por los entregadores. Es posible que para un entregador de Aula Global el fin del plazo para una entrega a las 24:00 termine 10 minutos antes. Revise el soporte de Aula Global.
3) Se prestará especial atención a detectar funcionalidades copiadas entre dos prácticas. En caso de encontrar implementaciones comunes en dos prácticas (o contenidos similares en la memoria), ambas obtendrán una calificación de 0.

Códigos de la práctica

1) Todos los ejercicios deben utilizar los nombres de función descritos en el enunciado y seguir el convenio de paso de parámetros visto en la asignatura.
2) Los ejercicios que no compilen o que no se ajusten a la funcionalidad y requisitos planteados, obtendrán una calificación de 0.
    - Esto incluye no usar los nombres de variables, funciones o ficheros pedidos.
3) Un programa no comentado, obtendrá una calificación de 0.

Memoria de la práctica

1) La memoria (un único documento) tendrá que contener al menos los siguientes apartados:
    - Portada donde figuren los autores (incluyendo nombre completo, NIA y dirección de correo electrónico).
    - Índice de contenidos.
    - Contenidos pedidos en los distintos ejercicios (una sección por ejercicio).
    - Conclusiones y problemas encontrados.
2) **La longitud de la memoria no deberá superar las 20 páginas** (portada e índice incluidos).
3) Al respecto de la posible descripción de los programas pedidos:
    - Se ha de detallar las principales funciones implementadas.
    - Se ha de incluir la batería de pruebas utilizadas y resultados obtenidos. Se dará mayor puntuación a pruebas avanzadas, casos extremos, y en general a aquellas pruebas que garanticen el correcto funcionamiento de la práctica en todos los casos.
        - Evite pruebas duplicadas que evalúan los mismos flujos de programa. La puntuación de este apartado no se mide en función del número de pruebas,

sino del grado de cobertura de las mismas. Es mejor pocas pruebas que evalúan diferentes casos a muchas pruebas que evalúan siempre el mismo caso.

<u>NOTA:</u> 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.

Procedimiento de entrega de la práctica

La entrega de la práctica 2 se realizará de forma electrónica a través de Aula Global

Se habilitarán dos entregadores distintos, uno para el código de los **ejercicios** de este cuaderno y otro para la **memoria** completa. Para ello se habilitará dos enlaces, uno por cada entregador.

La fecha límite de entrega para ambos es el día **15 de noviembre de 2015 a las 23:55 horas.**

Es posible entregar tantas veces como quiera dentro del plazo dado, la única versión registrada de su práctica es la última entregada. La valoración de la práctica es la valoración del contenido de esta última entrega. Revise siempre lo que entregue.

Entregador: Se deberá entregar un único archivo comprimido en formato **zip** con el nombre ec_p2_AAAAAAAAA_BBBBBBBBB.zip donde A…A y B…B son los NIA de los integrantes del grupo.

El archivo **zip** debe contener solo los siguientes archivos:

- **p2-e1.s**
- **p2-e1-pruebas.s**
- **p2-e2.s**
- **p2-e2-pruebas.s**
- **memoria-p2.pdf**

Evaluación de la práctica

La evaluación de la práctica se va a dividir en dos partes:

- **Código (6 puntos)**
- **Memoria (4 puntos)**

La puntuación de cada ejercicio es proporcional a su dificultad:

- Ejercicio 1 (3 *puntos sobre 6*)
- Ejercicio 2 (3 *puntos* sobre 6)

Si un ejercicio no se entrega su puntuación será 0. Tenga en cuenta que para seguir el proceso de evaluación continua, la nota mínima obtenida en cada práctica debe ser de 2 y la media de las tres prácticas 4.

NOTAS:

1. **Si se detecta un error de concepto grave en la práctica (en cualquier apartado de cualquier ejercicio), la valoración global de toda la práctica será de cero puntos (0 puntos).**
2. **En caso de encontrar implementaciones comunes en dos prácticas (o contenidos similares en la memoria), ambas obtendrán una calificación de 0.**