ESTRUCTURA Y TECNOLOGÍA DE COMPUTADORES

1º Ingeniero en Informática

PRÁCTICA 1: REPRESENTACIÓN DE LA INFORMACIÓN (TIPOS DE DATOS)

Octubre de 2008

ÍNDICE

1. CRITERIOS DE EVALUACIÓN Y RECURSOS OFRECIDOS

1.1 - EVALUACIÓN DE LA PARTE PRÁCTICA DE LA ASIGNATURA:

- 40% de la nota total.

- Puntuación máxima en la parte práctica de Estructura y Tecnología de Computadores ("ETC", en lo sucesivo): 10 puntos. Se exigen 5 puntos para aprobar.

- Reparto de puntos:

  - 1 punto en un examen voluntario que se convocará para finales de Noviembre. Este examen versará sobre la práctica nº 1.
  - 1 punto en un examen voluntario que se convocará para finales de Febrero. Este examen versará sobre la práctica nº 3.
  - 8 puntos en la realización, en grupos de 2 personas, de un programa obligatorio en el ensamblador del MIPS, con entrevista individual con el profesor que se concertará a finales de curso. PARA QUE ESTE PROGRAMA SEA TENIDO EN CUENTA, ES IMPRESCINDIBLE OBTENER, AL MENOS, 4 PUNTOS.
  - 1 punto en un examen voluntario que se convocará para mediados de Junio. Este examen versará sobre la práctica nº 5.
  - La puntuación total máxima (11 puntos, teóricamente) no podrá superar los 10 puntos.

1. CRITERIOS DE EVALUACIÓN Y RECURSOS OFRECIDOS

1.1 - EVALUACIÓN DE LA PARTE PRÁCTICA DE LA ASIGNATURA (continuación):

- Otras consideraciones:

  - Durante el curso académico 2008-2009 se considerará una Convocatoria de Exámenes Ordinaria, en Junio de 2008, y dos Convocatorias Extraordinarias, en Septiembre de 2008 y Febrero de 2009, respectivamente. DE NO HABER APROBADO LAS PRÁCTICAS DE ETC EN FEBRERO DE 2009, EL ALUMNO DEBERÁ "PARTIR DE 0" Y VOLVER A REALIZAR LAS PRÁCTICAS QUE, EN SU MOMENTO, SE ESTABLEZCAN PARA EL CURSO 2009-2010.

  - En las dos convocatorias extraordinarias sólo se tendrá en cuenta el proyecto en el ensamblador del MIPS, que ahora ya no se valorará con 8 puntos, sino con 10. Además, en estas dos convocatorias extraordinarias podrán exigirse algunas modificaciones y/o mejoras adicionales sobre la versión del programa que se pida en Junio.

1. CRITERIOS DE EVALUACIÓN Y RECURSOS OFRECIDOS

1.2 - INFORMACIÓN ACADÉMICA:

- Convocatorias de Examen: De forma obligatoria, en el Tablón de Anuncios que cada titulación tiene asignado en la Planta Baja de la Facultad. Opcionalmente, el aviso puede llegar a través de SUMA: https://suma.um.es.

- Avisos: Por lo común, mediante SUMA (https://suma.um.es).

- Calificaciones: En SUMA (https://suma.um.es), en el Tablón de Anuncios de la Planta Baja de la Facultad, o en la página web de la asignatura (http://www.ditec.um.es/etc).

1.3 - RECURSOS DIDÁCTICOS:

- En la página web de la asignatura, http://www.ditec.um.es/etc:

  - Apuntes, ejercicios y prácticas de la asignatura, actualizados en la web algunos días antes de que sean tratados en las clases presenciales. SE RECOMIENDA UNA LECTURA PREVIA DE LOS APUNTES ANTES DE SU TRATAMIENTO EN CLASE. Para acceder a todos estos recursos se necesita la siguiente identificación:

    ➔Nombre de usuario: **etc-08-09**
    ➔Contraseña: **pdp11**

1. CRITERIOS DE EVALUACIÓN Y RECURSOS OFRECIDOS

1.3 - RECURSOS DIDÁCTICOS (continuación):

- Software necesario para la realización de las prácticas (con licencia libre).
- Exámenes resueltos de la asignatura, pertenecientes a convocatorias anteriores.

- En la página web del profesor: http://webs.um.es/einiesta/miwiki/doku.php?id=docencia

- Bibliografía recomendada en los apuntes de teoría y prácticas. Para la primera práctica de ETC se recomienda especialmente el Apéndice A del libro "Estructura y diseño de computadores: Interficie circuitería/programación", de David A. Patterson y Jhon L. Hennessy.

1. CRITERIOS DE EVALUACIÓN Y RECURSOS OFRECIDOS

1.4 - RECURSOS MATERIALES:

- Laboratorios de la Facultad, en aquellos horarios en los que no se esté dando clase (ver horario en el tablón que hay frente a cada laboratorio).

- Redes inalámbricas "Eduroam" e "Icarus", con cobertura en todo el campus. ESTÁ PERMITIDA LA CONEXIÓN A ESTAS REDES DESDE PORTÁTILES PARTICULARES, PERO SÓLO A TRAVÉS DE UNA DE ESTAS DOS REDES WIFI, NO MEDIANTE CABLE UTP.

  Para saber cómo configurar el acceso inalámbrico en un ordenador particular, consultar la dirección http://www.um.es/atica/red-inalambrica.

1. CRITERIOS DE EVALUACIÓN Y RECURSOS OFRECIDOS

1.5 - DOCENCIA:

- Profesor: Eduardo Iniesta Soto, sólo para el primer trimestre de la parte práctica de la asignatura. Más información en http://ditec.um.es/personal/48.

- Tutorías:

  - Presenciales: En el despacho 3.02 de la Facultad (3ª planta), durante el horario expuesto en http://ditec.um.es/personal/48 (actualmente los lunes, de 17:00 a 21:00, y los jueves, de 17:00 a 19:00). Teléfono: 968398573 (sólo se asegura respuesta en horario de tutorías).

  - Telemáticas: A través de SUMA (https://suma.um.es) o, alternativamente, enviando un correo electrónico a einiesta@ditec.um.es.

2 - INTRODUCCIÓN

2.1 - OBJETIVO DE LA PRÁCTICA 1:

- Conocer el modo en que se almacenan en un computador los distintos datos que éste maneja (caracteres, números enteros, números reales, etc). Además de datos, un computador también almacena y procesa las instrucciones de que se compone todo programa. Estas instrucciones actúan sobre los datos de entrada, los transforman y, por último, producen otros datos de salida. ESTE ES EL PROPÓSITO GENERAL DE TODO PROGRAMA INFORMÁTICO. El modo en que se almacenan en un computador las instrucciones de los programas no es objeto de la práctica 1 (se tratará en la práctica 3).

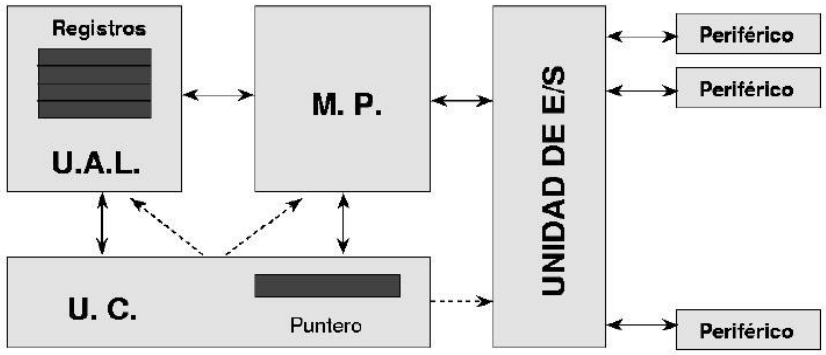
2.2 - CONTEXTO BÁSICO:

- Computador = Ordenador = Máquina de Von Neumann = Máquina MULTIFUNCIONAL de PROGRAMA ALMACENADO. Inventor del Computador: El matemático e ingeniero húngaro Jhon Von Neumann (1903-1957). Mérito: Idear un modelo técnico (es decir, una "ARQUITECTURA", o "HARDWARE") que convertía una máquina real en un número potencialmente infinito de máquinas virtuales, distinguiéndose cada posible máquina virtual en función del programa respectivo ("SOFTWARE") que, en cada caso, se le colocara en memoria.

2 - INTRODUCCIÓN

2.2 - CONTEXTO BÁSICO (continuación):

- Arquitectura de Von Neumann:

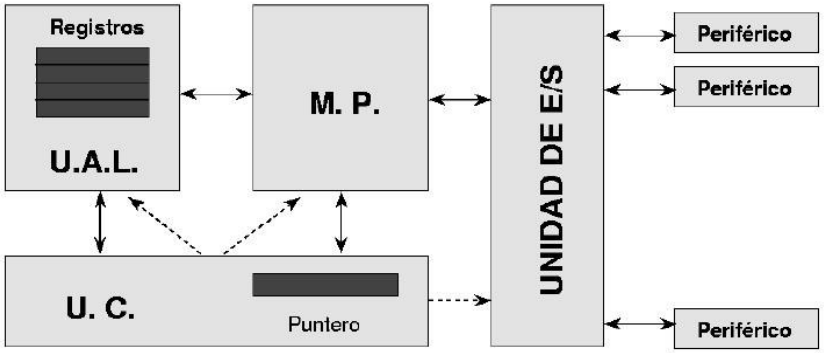


- El programa que ahora mismo esté ejecutándose debe residir en MEMORIA PRINCIPAL (= RAM). Está compuesto por INSTRUCCIONES MÁQUINA en lenguaje binario (secuencias de ceros y unos -bits-).

- Los datos que necesita el programa son recabados por los DISPOSITIVOS DE ENTRADA (como el teclado, por ejemplo). Los resultados son enviados a DISPOSITIVOS DE SALIDA (pantalla, por ejemplo).

2 - INTRODUCCIÓN

2.2 - CONTEXTO BÁSICO (continuación):



- CPU (Unidad Central de Proceso) = UC (Unidad de Control) + ALU (Unidad Aritmético-Lógica). A la CPU también se la denomina MICROPROCESADOR (o, simplemente, PROCESADOR). La UC va procesando en secuencia las instrucciones máquina del programa que esté en marcha, y envía órdenes al resto de unidades funcionales (ALU, Memoria, dispositivos de entrada/salida) en función de qué tipo de instrucción se esté procesando en cada momento. Cada instrucción cubre las etapas del denominado CICLO DE INSTRUCCIÓN, que en su versión más básica se compone de las fases de CAPTURA, DECODIFICACIÓN y EJECUCIÓN.

  En el microprocesador MIPS R2000, por ejemplo, todas las instrucciones máquina son de 32 bits, siendo los 6 primeros bits el código de operación (OBTENIDO EN LA FASE DE DECODIFICACIÓN) y el resto los distintos operandos que en cada caso sean necesarios.

2 - INTRODUCCIÓN

2.2 - CONTEXTO BÁSICO (continuación):

- Ejemplos de microprocesadores:

  - Intel y compatibles: Computadores PC.

*Intel 8088 (año 1974, microprocesador de 16 bits, *)*

Intel Core Duo (año 2007, microprocesador de 64 bits)

(*) Definición "informal" de REGISTRO: Pequeño dispositivo electrónico de alta velocidad que se utiliza como "comodín" para guardar cualquier información (dato, instrucción, constante, etc). Se dice que un microprocesador es de "x" bits cuando ése es el tamaño de todos sus registros (tanto los de la UC como los de la ALU), o al menos es el tamaño predominante. Al valor "x" también se le denomina "ANCHURA DE PALABRA".

2 - INTRODUCCIÓN

2.2 - CONTEXTO BÁSICO (continuación):

- PowerPC: Computadores Macintosh.

PowerPC 601 (año 1993, microprocesador de 32 bits)

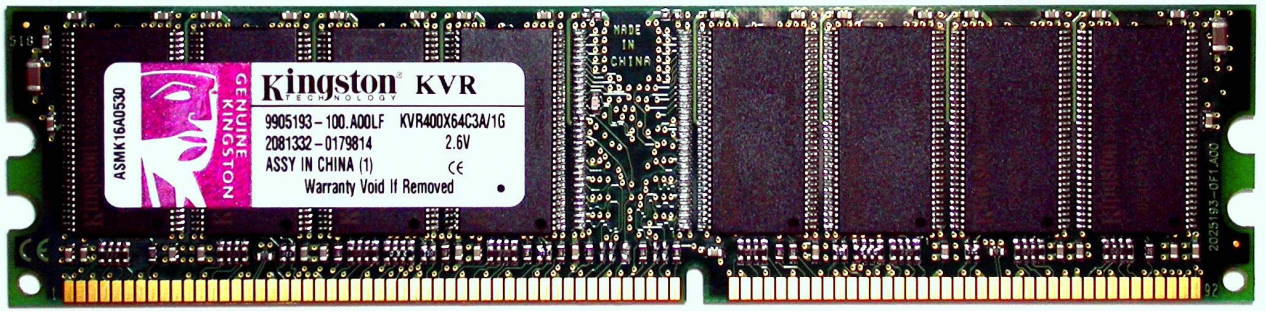
- MIPS: PDA's, videoconsolas (Nintendo, PlayStation), routers, etc.

MIPS R4400 (año 1993, microprocesador de 64 bits)

2 - INTRODUCCIÓN

2.2 - CONTEXTO BÁSICO (continuación):

- Ejemplo de memoria:



Módulo DDR-SDRAM de 1 Gbyte (*)

(*) El tamaño de un programa o de cualquier archivo de otro tipo siempre es un múltiplo de 8 bits. Las unidades de medida más utilizadas son:

byte (sin abreviatura) = 8 bits
kilobyte (KB) = 1024 bytes
megabyte (MB) = 1024 KB
gigabyte (GB) = 1024 MB
terabyte (TB) = 1024 GB

2 - INTRODUCCIÓN

2.3 - ARQUITECTURA DE REFERENCIA:

- Tanto la práctica 1 como el resto de prácticas se llevarán a cabo trabajando VIRTUALMENTE sobre un computador con microprocesador MIPS R2000.

  Decimos que este trabajo será virtual porque, en realidad, los computadores del laboratorio son PCs con microprocesador Intel (o compatible). Sin embargo, utilizaremos herramientas de simulación, como el software PCSPIM, con las que conseguiremos un entorno de trabajo similar al que tendríamos si programáramos realmente sobre un MIPS R2000.

- MIPS R2000 tiene un ancho de palabra de 32 bits. Todas sus instrucciones máquina son de 32 bits, correspondiendo los 6 primeros bits al código de instrucción, y el resto a los operandos necesarios.

2 - INTRODUCCIÓN

2.3 - ARQUITECTURA DE REFERENCIA (continuación):

- El banco de registros del MIPS R2000 cuenta con 32 elementos:

Nombre	Número de registro	Uso	Preservado en llamadas
`$zero`	`$0`	Valor constante 0	Sí, puesto que es de sólo lectura
`$at`	`$1`	Reservado por el ensamblador	No, puesto que es utilizado por el ensamblador
`$v0-$v1`	`$2-$3`	Valores para resultados y evaluación de expresiones	No
`$a0-$a3`	`$4-$7`	Argumentos	Sí
`$t0-$t7`	`$8-$15`	Temporales	No
`$s0-$s7`	`$16-$23`	Salvados (estáticos)	Sí
`$t8-$t9`	`$24-$25`	Más temporales	No
`$k0-$k1`	`$26-$27`	Reservados por el sistema operativo	n.a.
`$gp`	`$28`	Puntero global	Sí
`$sp`	`$29`	Puntero de pila	Sí
`$fp`	`$30`	Puntero de bloque de activación	Sí
`$ra`	`$31`	Dirección de retorno	Sí

2 - INTRODUCCIÓN

2.4 - LENGUAJE ENSAMBLADOR:

- Los programas que implementemos en las prácticas estarán escritos en LENGUAJE ENSAMBLADOR.

- El ENSAMBLADOR es un LENGUAJE DE PROGRAMACIÓN más (como también lo son otros lenguajes de programación menos "complejos", como, por ejemplo, JAVA, C o PASCAL).

- Un lenguaje de programación es una forma normalizada de escribir programas de un modo fácilmente comprensible para los humanos.

- Los lenguajes de programación siempre están asociados a un COMPILADOR. Un compilador es un software que considera como entrada el programa escrito en el lenguaje de programación, o CÓDIGO FUENTE del programa, y genera como salida otro archivo que es el mismo programa pero ya en CÓDIGO MÁQUINA (*).

(*) No es precísamente el caso de los ensambladores, pero en general los compiladores casi siempre están integrados en lo que actualmente se conoce como "HERRAMIENTAS RAD" (Microsoft Visual Studio, Borland Delphi, Eclipse, etc).

2 - INTRODUCCIÓN

2.4 - LENGUAJE ENSAMBLADOR (continuación):

- Todo PROGRAMA (o "APLICACIÓN") es un ALGORITMO (*) expresado en un lenguaje que el computador sabe interpretar (el LENGUAJE MÁQUINA, compuesto por INSTRUCCIONES MÁQUINA, que son, según lo visto, códigos binarios que el hardware de la máquina conoce y sabe cómo procesar).

- Por extensión, a un algoritmo que, en lugar de adoptar la forma de una secuencia de instrucciones máquina, viene expresado en un lenguaje de programación (y, por tanto, fácilmente legible), también se le denomina "programa".

(*) ALGORITMO: Conjunto claro y preciso de pasos que tienen por objeto resolver algún problema. La realización secuencial de todos los pasos de que consta un algoritmo permite obtener un conjunto de resultados a partir de un conjunto de datos de entrada. Un algoritmo no tiene por qué expresarse en ningún lenguaje de programación en particular; de hecho, puede ser descrito en lenguaje natural, pero siempre de un modo claro y sin ambigüedades.

2 - INTRODUCCIÓN

2.4 - LENGUAJE ENSAMBLADOR (continuación):

- Ejemplos de un mismo programa expresado en 1) el lenguaje máquina del MIPS, 2) en su lenguaje ensamblador y 3) en un lenguaje de alto nivel (C).

```
00100111101111011111111111100000
10101111101111110000000000010100
10101111101001000000000000100000
10101111101001010000000000100100
10101111101000000000000000011000
10101111101000000000000000011100
10001111101011100000000000011100
10001111101110000000000000011000
00000001110011100000000000011001
00100101110010000000000000000001
00101001000000010000000001100101
10101111101010000000000000011100
00000000000000000111100000010010
00000011000011111100100000100001
00010100001000001111111111110111
10101111101110010000000000011000
00111100000001000001000000000000
10001111101001010000000000011000
00001100000100000000000011101100
00100100100001000000010000110000
10001111101111110000000000010100
00100111101111010000000000100000
00000011111000000000000000001000
00000000000000000001000000100001
```

FIGURA A.2 Código de la rutina que calcula e imprime la suma de los cuadrados de los números enteros del 0 al 100 en lenguaje máquina del MIPS.

2 - INTRODUCCIÓN

2.4 - LENGUAJE ENSAMBLADOR (continuación):

```
addiu   $29,   $29, 32
sw      $31,   20($29)
sw      $4,    32($29)
sw      $5     36($29)
sw      $0,    24($29)
sw      $0,    28($29)
lw      $14,   28($29)
lw      $24,   24($29)
multu   $14,   $14
addiu   $8,    $14, 1
slti    $1,    $8, 101
sw      $8,    28($29)
mflo    $15
addu    $25,   $24, $15
bne     $1,    $0, -9
sw      $25,   24($29)
lui     $4,    4096
lw      $5,    24($29)
jal     1048   812
addiu   $4     $4, 1072
lw      $31,   20($29)
addiu   $29,   $29, 32
jr      $31,
move    $2,    $0
```

FIGURA A.3 La misma rutina escrita en lenguaje ensamblador. De todos modos, en el código no se han etiquetado los registros ni las direcciones de memoria, y tampoco se incluyen comentarios.

2 - INTRODUCCIÓN

2.4 - LENGUAJE ENSAMBLADOR (continuación):

```
  #include <stdio.h>
int
main (int argc, char *argv[])
{
      int i;
      int sum = 0;

      for (i = 0; i <=100; i = i + 1) sum = sum + i * i;
      printf ("La suma desde 0 … 100 es %d\n", sum);
}
```

FIGURA A.5 La misma rutina escrita en el lenguaje de programación C.

2 - INTRODUCCIÓN

2.4 - LENGUAJE ENSAMBLADOR (continuación):

- Los ensambladores son lenguajes que se caracterizan por su cercanía al código máquina. Cada instrucción del ensamblador generalmente se corresponde con una determinada instrucción máquina (al contrario que otros lenguajes de programación de más alto nivel, en los que cada instrucción habitualmente el compilador respectivo la transforma en un conjunto de varias instrucciones máquina).

- Hay, por tanto, una clara UTILIDAD DIDÁCTICA en los ensambladores, pero NORMALMENTE NO SE UTILIZAN PARA PROGRAMAR, salvo que la tarea que haya que implementar requiera unos niveles de eficiencia y seguridad que no puedan conseguirse con el código máquina generado por los compiladores de los lenguajes de programación de alto nivel (cita de un ejemplo del apéndice A de libro de Patterson y Hennessy -pág. 7- *).

(*)*"La razón principal para decidir programar en lenguaje ensamblador, y no en un lenguaje de alto nivel, es que la velocidad y el tamaño del programa sean críticos. Por ejemplo, considere un computador que controla una pieza de un motor, como pueden ser los frenos de un coche. Un computador que se incorpora a otro dispositivo, como puede ser un coche, se denomina "computador empotrado". Este tipo de computador ha de RESPONDER RÁPIDO Y DE FORMA PREDECIBLE a situaciones externas. Los compiladores introducen incertidumbre sobre el coste en tiempo de las operaciones de los lenguajes de alto nivel. Por tanto, no ofrecen a los programadores la seguridad de que un programa escrito en alto nivel responda en el tiempo deseado (por ejemplo, 1 milisegundo después de que un sensor detecta que un neumático patina). En cambio, un programador de lenguaje ensamblador controla exactamente qué instrucciones se van a ejecutar".*

2 - INTRODUCCIÓN

2.4 - LENGUAJE ENSAMBLADOR (continuación):

- Debido a su proximidad a la máquina, cada ensamblador vale sólo para su arquitectura respectiva. De este modo, cabe hablar de un ensamblador para máquinas Intel, otro para máquinas MIPS, otro para PowerPC, etc. Sin embargo, los lenguajes de programación de alto nivel permiten que el código fuente escrito en esos lenguajes sea válido para diferentes arquitecturas. Por ejemplo, un programa escrito en el lenguaje C puede valer perfectamente para un PC (con microprocesador Intel, por ejemplo) y para un Macintosh (con microprocesador PowerPC, por ejemplo). LO QUE SÍ QUE CAMBIARÍA EN ESTOS DOS EJEMPLOS ES EL COMPILADOR, YA QUE EL CÓDIGO MÁQUINA GENERADO SERÁ, EN EL PRIMER CASO, EL DEL INTEL, Y EN EL SEGUNDO CASO EL DEL POWERPC.

- Por su diseño claro y eficiente, hemos elegido el microprocesador MIPS R2000 como arquitectura de referencia para las prácticas. No obstante, nuestros laboratorios carecen de este procesador. El software PCSPIM es un simulador que permite ejecutar virtualmente programas escritos en el ensamblador del MIPS, PERO NO ES UN COMPILADOR (sino más bien un "INTÉRPRETE"). Otros ensambladores, como, por ejemplo el ensamblador del Intel, sí disponen de compiladores que generan código máquina real ("MASM" -macroassembler-, "TASM" -turboassembler-, etc).

3 - DIRECTIVAS MIPS PARA EL SEGMENTO DE DATOS

3.1 - PARTES DE UN PROGRAMA EN CÓDIGO MÁQUINA:

- En todas las arquitecturas un programa en ejecución se compone de cuatro partes o "SEGMENTOS":

  - SEGMENTO DE CÓDIGO: Parte del programa correspondiente a sus instrucciones máquina. Su tamaño, para cada programa, es fijo durante todo el tiempo que dura la ejecución de éste.

  - SEGMENTO DE DATOS: Parte del programa correspondiente a aquellos datos cuyo valor puede cambiar, pero que en ningún caso cambian de ubicación en la memoria o son desalojados de ésta. El tamaño del segmento de datos, por tanto, también permanece fijo durante toda la ejecución del programa.

  - SEGMENTO DE PILA: Su tamaño aumenta y decrece durante la ejecución del programa. La pila se suele utilizar para guardar temporalmente el valor de algunos registros de la CPU cuyo contenido corre el riesgo de ser sobreescrito tras la invocación a un PROCEDIMIENTO (*), que quizá necesite esos registros para alojar sus propios datos. Cuando el procedimiento termina, el contenido anterior de los registros, que está salvado en la pila, es restaurado en sus respectivos orígenes, liberándose de la pila esa zona de la memoria y disminuyendo, de este modo, su tamaño.

(*) PROCEDIMIENTO: Trozo de código perfectamente acotado que cumple un cometido concreto y que puede ser RELLAMADO (REUTILIZADO) cuantas veces se quiera y desde cualquier punto del programa, sin tener que reescribirlo cada vez que se necesite.

3 - DIRECTIVAS MIPS PARA EL SEGMENTO DE DATOS

3.1 - PARTES DE UN PROGRAMA EN CÓDIGO MÁQUINA (continuación):

- SEGMENTO "HEAP": Aloja datos que son "creados" y en otro momento posterior (cuando ya no hacen falta) son "destruidos" dinámicamente, es decir, durante la misma ejecución del programa. El tamaño del heap, por tanto, también es, como el de la pila, variable mientras dure la ejecución del programa.

- En las prácticas trabajaremos con los segmentos de código, datos y pila, aunque para la práctica 1 sólo nos interesan el segmento de código y el segmento de datos.

3.2 - DIRECTIVAS MIPS PARA DELIMITAR SEGMENTOS:

- En el ensamblador del MIPS, una DIRECTIVA es una palabra reservada del lenguaje QUE NO ES UNA INSTRUCCIÓN.

- Las directivas del ensamblador MIPS que sirven para delimitar, dentro del código fuente de un programa, sus segmentos de datos y de código (este último también denominado "SEGMENTO DE TEXTO") son, respectivamente, **".data"** y **".text"**.

3 - DIRECTIVAS MIPS PARA EL SEGMENTO DE DATOS

3.3 - ETIQUETAS:

- Una ETIQUETA es toda CADENA DE CARACTERES que acabe con dos puntos (":"). Las etiquetas pueden aplicarse tanto a instrucciones como a datos, pero en cualquier caso SIEMPRE HACEN REFERENCIA A LA DIRECCIÓN DE MEMORIA (o "POSICIÓN") EN LA QUE **COMIENZA** DICHA INSTRUCCIÓN O DATO.

- Ejemplo de uso de directivas y etiquetas:

```
# SEGMENTO DE DATOS
          .data
var_a:    .word 255
var_b:    .word 0xFF
cadena:   .asciiz "El resultado es "

# SEGMENTO DE CÓDIGO
          .text
main:     jail print_cadena
          lw $8,var_a
          lw $t1,var_b
...
```

3 - DIRECTIVAS MIPS PARA EL SEGMENTO DE DATOS

3.3 - ETIQUETAS (continuación):

- Una etiqueta es una dirección de memoria, pero no toma un valor (no se "INSTANCIA") hasta que el SISTEMA OPERATIVO (Windows, Linux…) carga el programa en memoria. Con cada ejecución (carga) distinta, es posible que las etiquetas cambien de valor, si bien sus posiciones relativas en memoria seguirán siendo las mismas.

- Las direcciones de memoria SIEMPRE SON "NÚMEROS DE BYTE", suponiendo que el primer byte de la memoria es el 0, el segundo es el 1, el tercero es el 2 y así sucesivamente.

- MIPS admite direcciones de 32 bits, lo que quiere decir que su dirección menor es la 0 (byte 0), y que su dirección mayor es la 2^{32} -1, es decir la 4.294.967.296 (byte 4.294.967.296). La capacidad máxima teórica de la memoria RAM de un MIPS es, pues, 4 GB.

3 - DIRECTIVAS MIPS PARA EL SEGMENTO DE DATOS

3.4 - DIRECTIVAS MIPS PARA LA DECLARACIÓN DE DATOS:

- Directiva **".ascii str"**:

  - Propósito: Sirve para indicar que lo que sigue es un dato de tipo cadena.

  - Ejemplo: `mi_cadena: .ascii "El resultado es "`

    Se almacena, a partir de la dirección "cadena", el código ascii de la letra "E" (byte 0x45 -69 en decimal-), el código ascii de la letra "l" (byte 0x6C -108 en decimal-), y así sucesivamente (EL CÓDIGO ASCII ES LA REPRESENTACIÓN INTERNA DE LOS CARACTERES DE UNA CADENA EN LOS MICROS MIPS E INTEL, entre otros).

- Directiva **".asciiz str"**: Similar a la directiva "ascii", pero con la particularidad de que el lenguaje inserta automáticamente el byte 0x00 al final de la cadena, lo que es interpretado como su terminación.

3 - DIRECTIVAS MIPS PARA EL SEGMENTO DE DATOS

3.4 - DIRECTIVAS MIPS PARA LA DECLARACIÓN DE DATOS (continuación):

- Directiva **".byte"**:

  - Propósito: Indica que lo siguiente es un conjunto de uno o más números enteros de tipo "byte". Estos enteros SON REPRESENTADOS INTERNAMENTE EN COMPLEMENTO A 2, Y CONSUMEN 1 BYTE, siendo, por tanto, su rango de representación el intervalo $[-2^{(8-1)} , 2^{(8-1)}-1]$ (o sea, de -128 a +127).

  - Ejemplo: `enteros: .byte 0,2,8,-5,199`

    Si el sistema operativo, en el momento de cargar en memoria el programa para comenzar su ejecución, asignara a la etiqueta "enteros" la dirección 0x10010000 (en decimal sería el byte nº 268.500.992), entonces a partir de esa posición el contenido de la memoria sería: (LEER TABLA DE ABAJO A ARRIBA)

Posición de memoria	Contenido
0x10010004	0xC7
0x10010003	0xFB
0x10010002	0x08
0x10010001	0x02
0x10010000	0x00

3 - DIRECTIVAS MIPS PARA EL SEGMENTO DE DATOS

3.4 - DIRECTIVAS MIPS PARA LA DECLARACIÓN DE DATOS (continuación):

Representación de -5:

Complemento a 2 del valor absoluto:

$|-5_{10}| = 5_{10} = 00000101_2$. $C_2(00000101)$ -cambiando 0's por 1's y después sumando 1- = 11111011 = 0xFB

Nótese, además, que la resta de dos números enteros se consigue sumándole al minuendo el complemento a 2 del valor absoluto del sustraendo. Por ejemplo 8-5 sería: 00001000 + 11111011 = 00000011 (3_{10}). Gracias a esta característica del complemento a 2, es posible lograr restas a partir de sumas, con lo que se simplifica el hardware de la ALU (sólo serían necesarios circuitos sumadores), siendo este UNO DE LOS MOTIVOS DE UTILIZAR EL COMPLEMENTO A 2 COMO FORMA DE REPRESENTACIÓN DE LOS NÚMEROS ENTEROS.

Representación de 199:

0xC7 = 11000111_2. Sin embargo, esto corresponde a un número negativo (bit de signo -el más significativo- igual a 1), lo cuál hace que esta representación sea INCORRECTA. ¿POR QUÉ? Pues porque hemos excedido el rango de representación de los datos de tipo "byte", que va del -127 al +127. Por tanto, UN BUEN COMPILADOR NOS DEBERÍA INFORMAR DE ESTE ERROR DE "OVERFLOW" (cosa que, por cierto, no hace PCSPIM).

3 - DIRECTIVAS MIPS PARA EL SEGMENTO DE DATOS

3.4 - DIRECTIVAS MIPS PARA LA DECLARACIÓN DE DATOS (continuación):

- Directiva **".half"**:

  - Propósito: Se utiliza para declarar valores enteros de 2 BYTES, REPRESENTADOS EN COMPLEMENTO A 2. Rango de representación: $[-2^{(16-1)}, 2^{(16-1)}-1]$ (o lo que es lo mismo, de -32768 a +32767).

  - Peculiaridades:

    ➔ Cada valor "half" ocupa 2 bytes, y debe comenzar OBLIGATORIAMENTE en una dirección (número de byte) que sea múltiplo de 2. Si ello fuere preciso, se incluiría el byte 0x00 como relleno previo antes del dato. Esta característica recibe el nombre de ALINEAMIENTO.

    ➔ En las CPU's MIPS cada dato que ocupe más de 1 byte se almacena según el esquema "LITTLE ENDIAN". Esto significa que si el dato debe guardarse a partir de la dirección "d", entonces el byte menos significativo del dato siempre se alojará en la dirección "d", el segundo byte menos significativo en la dirección "d+1", el tercero en "d+2", y así sucesivamente (la otra alternativa, el esquema "BIG ENDIAN" -típico de otros microprocesadores- opta por el orden inverso de almacenamiento, esto es, guarda en la posición más baja de la memoria el byte más significativo).

3 - DIRECTIVAS MIPS PARA EL SEGMENTO DE DATOS

3.4 - DIRECTIVAS MIPS PARA LA DECLARACIÓN DE DATOS (continuación):

➔Cuando el número a representar es un valor negativo, el bit de signo que hay que insertar al principio (en la parte izquierda) debe ser replicado tantas posiciones como sea preciso hasta alcanzar la longitud propia de este tipo de datos (16 bits, en este caso).

- Ejemplo: `enteros: .half 2,-5,961,-961`

Suponiendo que, tras la carga del programa por parte del sistema operativo, la etiqueta "enteros" es instanciada con el valor 0x10010001 (en decimal sería el byte nº 268.500.993), el contenido de la memoria debería ser el siguiente:

Posición de Memoria	Contenido
0x10010009	0xFC
0x10010008	0x3F
0x10010007	0x03
0x10010006	0xC1
0x10010005	0xFF **(3)**
0x10010004	0xFB **(3)**
0x10010003	0x00 **(2)**
0x10010002	0x02 **(2)**
0x10010001	0x00 **(1)**

3 - DIRECTIVAS MIPS PARA EL SEGMENTO DE DATOS

3.4 - DIRECTIVAS MIPS PARA LA DECLARACIÓN DE DATOS (continuación):

(notas de la página anterior)

(1) Por el necesario alineamiento de los datos "half" a posiciones que sean múltiplos de 2, hay que rellenar con 0x00 el byte impar 0x10010001 (268.500.993).

(2) El almacenamiento "little endian" obliga a que, siendo la representación del 2 la ristra "0000000000000010" (o 0x0002), el byte menos significativo (0x02) ocupe la dirección 268.500.994, y el byte más significativo (0x00) la dirección 268.500.995.

(3) Si el -5 hubiese sido un "byte", su representación habría sido 0xFB (véase ejemplo anterior). Siendo ahora el -5 un "half", hay que extender el bit de signo a las 8 posiciones más significativas, quedando como 0xFFFB.

3 - DIRECTIVAS MIPS PARA EL SEGMENTO DE DATOS

3.4 - DIRECTIVAS MIPS PARA LA DECLARACIÓN DE DATOS (continuación):

- Directiva **".word"**:

  - Propósito: Declarar valores enteros de 4 BYTES (es decir, 1 PALABRA EN EL MIPS R2000), REPRESENTADOS EN COMPLEMENTO A 2. Rango de representación: $[-2^{(32-1)}, 2^{(32-1)}-1]$ (de -2.147.483.648 a +2.147.483.647).

  - Peculiaridades:

    ➔ Cada valor "word" ocupa 4 bytes, y debe comenzar OBLIGATORIAMENTE en una dirección (número de byte) que sea múltiplo de 4. Si ello fuere preciso, se incluirían como prefijo de 1 a 3 bytes 0x00, hasta lograr comenzar en una dirección que fuese múltiplo de 4 (ALINEAMIENTO).

    ➔ El esquema de almacenamiento es "LITTLE ENDIAN".

    ➔ Cuando el número a representar es un valor negativo, el bit de signo que hay que insertar al principio debe ser replicado tantas posiciones como sea preciso hasta alcanzar la longitud propia de este tipo de datos (32 bits).

3 - DIRECTIVAS MIPS PARA EL SEGMENTO DE DATOS

3.4 - DIRECTIVAS MIPS PARA LA DECLARACIÓN DE DATOS (continuación):

- Ejemplo: `enteros: .word 396010`

  Suponiendo que, tras la carga del programa por parte del sistema operativo, la etiqueta "enteros" es instanciada con el valor 0x10010001 (en decimal sería el byte nº 268.500.993), el contenido de la memoria debería ser el siguiente:

Posición de Memoria	Contenido
0x10010007	0x00 **(2)**
0x10010006	0x06 **(2)**
0x10010005	0x0A **(2)**
0x10010004	0xEA **(2)**
0x10010003	0x00 **(1)**
0x10010002	0x00 **(1)**
0x10010001	0x00 **(1)**

(1) Alineamiento.

(2) Almacenamiento "little endian" del entero 396.010 (0x00060AEA).

3 - DIRECTIVAS MIPS PARA EL SEGMENTO DE DATOS

3.4 - DIRECTIVAS MIPS PARA LA DECLARACIÓN DE DATOS (continuación):

- Directiva **".space"**:

  - Propósito: Establece como espacio de memoria reservado un número de bytes igual al valor que se indique a continuación de ".space".

  - Ejemplo: `espacio_para_mi_variable: .space 4`

    En este caso, a partir de la dirección de memoria correspondiente a la etiqueta "espacio_para_mi_variable" se dejan reservados 4 bytes, se supone que para poder dejar ahí posteriormente un dato que no ocupe más de 4 bytes (un "word", por ejemplo).

3 - DIRECTIVAS MIPS PARA EL SEGMENTO DE DATOS

3.4 - DIRECTIVAS MIPS PARA LA DECLARACIÓN DE DATOS (continuación):

- Directiva **".float"**:

  - Propósito: Sirve para declarar un conjunto de uno o más números reales en simple precisión. Estos valores SON REPRESENTADOS INTERNAMENTE EN EL FORMATO IEEE-754 DE SIMPLE PRECISIÓN, Y CONSUMEN 4 BYTES.

  - Peculiaridades:

    ➔ Como es habitual, el ALINEAMIENTO exige que cada "float" comience en una posición de memoria que sea múltiplo de 4 (su tamaño). El esquema de almacenamiento es, como siempre, "LITTLE ENDIAN".

    ➔ Toda representación en punto flotante, incluida la IEEE-754 de simple precisión, trae consigo la existencia de ERRORES DE REDONDEO, de modo que CASI NUNCA el número real a representar coincidirá exactamente con el número real representado internamente en los registros o en la memoria de una máquina MIPS (AUNQUE ESTA IDEA TAMBIÉN ES APLICABLE A CUALQUIER OTRA CPU).

3 - DIRECTIVAS MIPS PARA EL SEGMENTO DE DATOS

3.4 - DIRECTIVAS MIPS PARA LA DECLARACIÓN DE DATOS (continuación):

- Ejemplo: `mifloat: .float 786.8020305`

  Suponiendo que, tras la carga del programa por parte del sistema operativo, la etiqueta "mifloat" es instanciada con el valor 0x10010060 (en decimal sería el byte nº 268.501.088, que es múltiplo de 4), el contenido de la memoria debería ser el siguiente:

Posición de Memoria	Contenido
0x10010063	0x44
0x10010062	0x44
0x10010061	0xB3
0x10010060	0x54

JUSTIFICACIÓN: ¿Es realmente 0x4444B354 la representación interna del número 786.8020305?

3 - DIRECTIVAS MIPS PARA EL SEGMENTO DE DATOS

3.4 - DIRECTIVAS MIPS PARA LA DECLARACIÓN DE DATOS (continuación):

Valor a codificar en IEEE 754 de simple precisión: 786,8020305. Pasamos a binario,

- Parte entera (786):

786 = 512 + 256 + 16 + 2, luego $786_{10} = 1100010010$

- Parte decimal (0,8020305):

Hemos de obtener, entre la parte entera (que después normalizaremos multiplicando y dividiendo por 2^9*) y la parte decimal, un total de 23 dígitos para la mantisa. La parte entera aporta 9 dígitos (el 1 más significativo no cuenta, pues se considera implícito en las mantisas de IEEE 754). En consecuencia, hemos de llegar a 23 – 9 = 14 bits decimales en el proceso de cambio de base de la parte decimal.*

0,8020305 · 2 = 1,604061
0,416244 · 2 = 0,832488
0,329952 · 2 = 0,659904
0,639616 · 2 = 1,279232
0,116928 · 2 = 0,233856

0,604061 · 2 = 1,208122
0,832488 · 2 = 1,664976
0,659904 · 2 = 1,319808
0,279232 · 2 = 0,558464
0,233856 · 2 = 0,467712

0,208122 · 2 = 0,416244
0,664976 · 2 = 1,329952
0,319808 · 2 = 0,639616
0,558464 · 2 = 1,116928

3 - DIRECTIVAS MIPS PARA EL SEGMENTO DE DATOS

3.4 - DIRECTIVAS MIPS PARA LA DECLARACIÓN DE DATOS (continuación):

*Así pues, la parte decimal es: 0,11001101010100 **(1)**, de todo lo cuál se infiere que:*

$786{,}8020305_{10} = 1100010010{,}11001101010100 = 1{,}10001001011001101010100 \cdot 2^{9}$.

***(1)** Realmente deberíamos utilizar la regla del "redondeo al par" que sigue IEEE 754 para saber si el último 0 se queda en 0 o si se cambia por 1. Por lo tanto, obtenemos los dos siguientes bits decimales:*

$0{,}467712 \cdot 2 = \underline{0},935424$ $\qquad$ $0{,}935424 \cdot 2 = \underline{1},870848$

Bits obtenidos para decidir el redondeo: 01. Criterio de redondeo:

00 (fracción 0,00):	*El último bit se deja con el valor que tenga.*
01 (fracción 0,25):	*El último bit se deja con el valor que tenga.*
11 (fracción 0,75):	*Al último bit se le suma un 1.*
10 (fracción 0,50):	*Si el último bit es un 0, lo dejamos a 0; si es un 1, le sumamos un 1 (forzando así que el valor representado siempre sea par; de ahí el nombre de la regla).*

Los bits 01 representan la fracción 0,25, luego dejamos el último bit ("0") como está.

3 - DIRECTIVAS MIPS PARA EL SEGMENTO DE DATOS

3.4 - DIRECTIVAS MIPS PARA LA DECLARACIÓN DE DATOS (continuación):

Ahora el paso a IEEE 754 en simple precisión es casi inmediato:

- *Bit de Signo: 0, pues el valor a representar es positivo.*
- *8 bits del exponente: Es 9 más el sesgo (127), o sea, 136, que en binario es 10001000.*
- *23 bits de la mantisa: Los ya calculados (omitiendo el "1,"), es decir, 10001001011001101010100.*

Reuniendo todos estos bits, obtenemos la representación del 786,8020305 en IEEE 754 de simple precisión:

0100 0100 0100 0100 1011 0011 0101 0100 = ***0x4444B354***

3 - DIRECTIVAS MIPS PARA EL SEGMENTO DE DATOS

3.4 - DIRECTIVAS MIPS PARA LA DECLARACIÓN DE DATOS (continuación):

Sin embargo, la representación interna 0x4444B354 no es exactamente 786.8020305, sino otro valor muy próximo. Concretamente es 786.80200195 (diferencia: 0.00002855). Este ERROR DE REDONDEO se debe al tamaño limitado que tenemos para la mantisa (23 dígitos). Estudiar y ejecutar el siguiente programa:

```
# REPRESENTACIÓN DEL VALOR 786.8020305 EN SIMPLE PRECISIÓN.

        .data
cad:    .asciiz "Representación de 786.8020305: "
minum:  .float 786.8020305

        .text
main:   li $v0,4
        la $a0,cad
        syscall

        li $v0,2
        la $t0,minum
        l.s $f12,0($t0)
        syscall

        li $v0,10
        syscall
```

3 - DIRECTIVAS MIPS PARA EL SEGMENTO DE DATOS

3.4 - DIRECTIVAS MIPS PARA LA DECLARACIÓN DE DATOS (continuación):

- Directiva **".double"**:

  - Propósito: Sirve para declarar un conjunto de uno o más números reales en doble precisión. Estos valores SON REPRESENTADOS INTERNAMENTE EN EL FORMATO IEEE-754 DE DOBLE PRECISIÓN, Y CONSUMEN 8 BYTES.

  - Peculiaridades:

    ➔Como es habitual, el ALINEAMIENTO exige que cada "double" comience en una posición de memoria que sea múltiplo de 8 (su tamaño). El esquema de almacenamiento es, como siempre, "LITTLE ENDIAN".

    ➔Toda representación en punto flotante, incluida la IEEE-754 de doble precisión, trae consigo la existencia de ERRORES DE REDONDEO.

4 - UBICACIÓN DE LOS PROGRAMAS EN MEMORIA

4.1 - UBICACIÓN DE LOS DATOS:

- El simulador del ensamblador del MIPS R2000 (PCSpim) asume que la dirección inicial del segmento de datos en la memoria principal es la **0x10010000**.

  Esto es sólo una HIPÓTESIS DE PCSPIM, pero de ninguna manera supone que, en un caso real, los segmentos de datos empiecen siempre en esa posición (ESO DEPENDERÁ DE QUÉ DECIDA EL SISTEMA OPERATIVO EN EL MOMENTO DE EMPEZAR A EJECUTAR EL PROGRAMA).

- Tal y como podrá comprobarse con PCSpim, LOS DATOS SIEMPRE SE ALOJAN DE FORMA CONTIGUA a partir de la citada posición 0x10010000, insertando, cuando ello proceda, los bytes nulos que aseguren el ALINEAMIENTO de aquellos datos que ocupen más de 1 byte (“half”, “word”, “float” y “double”).

  El esquema de almacenamiento, como ya se apuntó en su momento, es LITTLE ENDIAN.

4 - UBICACIÓN DE LOS PROGRAMAS EN MEMORIA

4.2 - UBICACIÓN DE LAS INSTRUCCIONES:

- El simulador del ensamblador del MIPS R2000 (PCSpim) asume que la dirección inicial del segmento de código en la memoria principal es la **0x00400000**.

  Al igual que sucedía en el caso del segmento de datos, esto es otra HIPÓTESIS DE PCSPIM, pero no es lo que sucede en realidad (ES EL SISTEMA OPERATIVO QUIEN DECIDE, EN CADA CASO, LA DIRECCIÓN INICIAL DEL SEGMENTO DE CÓDIGO).

- Las instrucciones también se almacenan en la memoria de forma contigua, y siguiendo el esquema LITTLE ENDIAN.

  TODAS LAS INSTRUCCIONES SON DE 4 BYTES.

4 - UBICACIÓN DE LOS PROGRAMAS EN MEMORIA

4.3 - EJEMPLO DE UBICACIÓN EN MEMORIA DE UN PROGRAMA MIPS:

- Código fuente (ejemplo anterior de representación de un "float"):

```
# REPRESENTACIÓN DEL VALOR 786.8020305 EN SIMPLE PRECISIÓN.

        .data
cad:    .asciiz "Representación de 786.8020305: "
minum:  .float 786.8020305

        .text
main:   li $v0,4
        la $a0,cad
        syscall

        li $v0,2
        la $t0,minum
        l.s $f12,0($t0)
        syscall

        li $v0,10
        syscall
```

4 - UBICACIÓN DE LOS PROGRAMAS EN MEMORIA

4.3 - EJEMPLO DE UBICACIÓN EN MEMORIA DE UN PROGRAMA MIPS (continuación):

- Contenido del segmento de datos:

```
        DATA
[0x10000000]...[0x10010000]     0x00000000
[0x10010000]                    0x72706552  0x6e657365  0x69636174  0x64206ef3
[0x10010010]                    0x38372065  0x30382e36  0x30333032  0x203a2e35
[0x10010020]                    0x00000000  0x4444b354  0x00000000  0x00000000
[0x10010030]...[0x10040000]     0x00000000
```

Nótese cómo el primer byte que hay en la primera dirección del segmento de datos (0x10010000) es el código ascii de la primera letra del primer dato que hemos declarado en el segmento de datos de nuestro fichero fuente (ascii 0x52, esto es, el 82, o sea, la letra "R" de "Representación de 786.8020305: ").

Nótese también que 31 bytes después ("Representación de 786.8020305: " tiene 31 caracteres) aparece el carácter fin de línea, 0x00, típico de toda cadena declarada como "asciiz". Dicho carácter 0x00 está en 0x10010020. Como el siguiente dato es un float (0x4444B354, representación interna de 786.8020305), PCSpim se encarga de insertar automáticamente los tres bytes nulos que son necesarios para que ese float comience en una dirección que sea múltiplo de 4, respetando así su alineamiento.

4 - UBICACIÓN DE LOS PROGRAMAS EN MEMORIA

4.3 - EJEMPLO DE UBICACIÓN EN MEMORIA DE UN PROGRAMA MIPS (continuación):

- Contenido del segmento de código:

```
[0x00400000]    0x8fa40000  lw $4, 0($29)                ; 174: lw $a0 0($sp)
[0x00400004]    0x27a50004  addiu $5, $29, 4             ; 175: addiu $a1 $sp 4
[0x00400008]    0x24a60004  addiu $6, $5, 4              ; 176: addiu $a2 $a1 4
[0x0040000c]    0x00041080  sll $2, $4, 2                ; 177: sll $v0 $a0 2
[0x00400010]    0x00c23021  addu $6, $6, $2              ; 178: addu $a2 $a2 $v0
[0x00400014]    0x0c100009  jal 0x00400024 [main]        ; 179: jal main
[0x00400018]    0x00000000  nop                          ; 180: nop
[0x0040001c]    0x3402000a  ori $2, $0, 10               ; 182: li $v0 10
[0x00400020]    0x0000000c  syscall                      ; 183: syscall
[0x00400024]    0x34020004  ori $2, $0, 4                ; 8: li $v0,4
```

La primera instrucción comienza en la posición 0x00400000, y forma parte de un grupo inicial de 9 instrucciones de inicialización que siempre son insertadas automáticamente por PCSpim. A partir de la décima instrucción (posición 0x00400024) pueden empezar a verse las instrucciones de nuestro programa (“li $v0,4”).

ANEXO: LLAMADAS AL SISTEMA

- Un SISTEMA OPERATIVO es un "macro-programa" que hace de intermediario entre los programas "normales" y el hardware de la máquina. Nada más arrancar un computador, el primer programa que de modo automático se carga en memoria y comienza inmediatamente a ejecutarse es siempre un sistema operativo, y no deja de ejecutarse hasta que se apaga la máquina.

- Ejemplos de sistemas operativos: WINDOWS (en sus distintas versiones: 3.1, 95, 98, NT, 2000, XP, Vista…), LINUX (en sus distintas distribuciones: Red Hat, Debian, Suse, Fedora, Mandriva, etc), SOLARIS (como Linux, otro sistema operativo derivado del Unix original), MacOS (típico de los Macintosh), etc.

- Desde el ensamblador del MIPS es posible llamar a determinados procedimientos del SISTEMA OPERATIVO. Estos procedimientos se denominan "LLAMADAS AL SISTEMA" y son invocables desde el ensamblador y también desde otros lenguajes de alto nivel. Las llamadas al sistema, al igual que el resto del sistema operativo, están permanentemente en memoria.

- La finalidad de las llamadas al sistema es la de proporcionar a los programadores una LIBRERÍA de procedimientos que lleve a cabo tareas básicas y muy frecuentes en casi todas las aplicaciones, como pueden ser, por ejemplo, la entrada por teclado, la salida por pantalla, el manejo de ficheros, la ejecución de otros programas, etc.

ANEXO: LLAMADAS AL SISTEMA

- Tabla de llamadas al sistema invocables desde el ensamblador del MIPS:

Servicio	Código de la llamada a sistema	Argumentos	Resultado
print_int	1	$a0 = entero	
print_float	2	$f12 = coma flotante	
print_double	3	$f12 = doble precisión	
print_string	4	$a0 = cadena	
read_int	5		entero (en $v0)
read_float	6		coma flotante (en $f0)
read_double	7		doble precisión (en $f0)
read_string	8	$a0 = buffer, $a1 = longitud	
sbrk	9	$a0 = cantidad	longitud (en $v0)
exit	10		

FIGURA A.17 Servicios del sistema.