EXAMPLES SHOWING HOW TO READ A SPIM MEMORY DUMP

BITS AND BYTES:

1 byte = 8 bits = 2 hexadecimal digits
32 bits = 4 bytes = 8 hexadecimal digits

HEXADECIMAL NOTATION

In the standard C notation 0x12abcdef, the symbol "0x" means "treat what follows as an unsigned integer in the hexadecimal representation". For example, 0x12abcdef denotes the unsigned 32-bit integer that one writes as 313,249,263 in the unsigned decimal representation. 0x12abcdef also denotes 313,249,263 (base 10) in two's complement notation.

BYTE-ADDRESSED MEMORY

Modern microprocessors can address each byte of main memory from address 0x00000000 up to and including the maximum address allowed in the instruction set architecture.
The maximum address in a 32-bit architecture is 0xffffffff.

WORD ADDRESSES

In a 32-bit architecture such as the MIPS R2000, words are 32 bits (4 bytes) in length. Addresses in main memory, registers and instructions are all 32 bits in length. Valid word addresses are 0x00000000, 0x00000004, 0x00000008, 0x0000000c (12 in decimal), 0x00000010 (16 in decimal), 0x00000014 (20 in decimal), 0x00000018 (24 in decimal), 0x0000002c (28 in decimal), and every other multiple of 4 up to and including 0xfffffffc.

The address of a 4-byte word is always the address of the byte with the lowest address.

BYTE ORDERING

Because memory is byte-addressed, the order in which the four bytes of a word in memory are assembled to form a numerical constant or an instruction is a matter of convention. Let's consider the following example of addresses and the contents (bytes) stored at each address:

Byte addresses	Contents		Word addresses
[0x00000007]	0xce	>	
[0x00000006]	0x8a	>	
[0x00000005]	0x46	>	
[0x00000004]	0x02	>	0x00000004
[0x00000003]	0xef	}	
[0x00000002]	0xcd	}	
[0x00000001]	0xab	}	
[0x00000000]	0x12	}	0x00000000

Two conventions are common:

BIG-ENDIAN byte ordering
(used in most architectures other than VAX or 80x86):
The word at address 0x00000000 in the above example is read as 0x12abcdef. The word at address 0x00000004 is read as 0x02468ace. Note that the most significant

("biggest") byte has the lowest address, and therefore is the byte whose address is the address of the whole word. (Hence the word address is the address of the "big end".)

In this example, the word at address 0x00000000 would be interpreted as representing 313,249,263 (decimal) in the 32-bit two's complement representation. The word at address 0x00000004 would be interpreted as 38,177,486 (decimal) in the 32-bit two's complement representation.

LITTLE-ENDIAN byte ordering
(used in the 80x86 architecture, therefore in all PCs):
The word at address 0x00000000 in the above example is read as 0xefcdab12. The word at address 0x00000004 is read as 0xce8a4602. Note that the least significant ("littlest") byte has the lowest address, and therefore is the byte whose address is the address of the whole word. (Hence the word address is the address of the "little end".)

In this example, the word at address 0x00000000 would be interpreted as representing -271,733,998 (decimal) in the 32-bit two's complement representation. The word at address 0x00000004 would be interpreted as -829,798,910 (decimal) in the 32-bit two's complement representation.

MEMORY DUMPS

Since there are versions of SPIM on both big-endian and little-endian architectures, we give examples for all of the architectures on which students are likely to run SPIM.

BIG-ENDIAN (Motorola 680x0, IBM PowerPC, Sun SPARC, DEC Alpha, HP PA-RISC, MIPS)

Here's the first part of a SPIM display of the data segment of the program display.sal (listed below), with extra annotations to explain what's going on:

```
DATA
[0x10000000]...[0x1000fffc] 0x00000000
<----- range of word ----->   ^^^^^^^^
      addresses; each           |
       word contains            |
         0x00000000 -------------
```

Word addresses:

[0x10010000]	0x43000000	0x00007fff	0x46fffe00	0x48656c6c
^^^^^^^^^^	^^^^^^^^^^	^^^^^^^^^^	^^^^^^^^^^	^^^^^^^^^^
address of first word on this line is in brackets	address is 0x10010000	address is 0x10010004 = add. of 1st word + 4	address is 0x10010008 = add. of 1st word + 8	address is 0x1001000c = add. of 1st word + c
[0x10010010]	0x6f2c2077	0x6f726c64	0x21004865	0x6c6c6f2c
^^^^^^^^^^	^^^^^^^^^^	^^^^^^^^^^	^^^^^^^^^^	^^^^^^^^^^
address of first word	address is 0x10010010	address is 0x10010014	address is 0x10010018	address is 0x1001001c

```
 on this                = add. of   = add. of   = add. of
 line is in             1st word    1st word    1st word
 brackets               + 4         + 8         + c
```

Byte addressing examples:

   The byte at address 0x1001000d is 0x65. The byte at
   address 0x1001000e is 0x6c.

Data storage order:

   In the source program declare.sal (listed below), the
   first data declaration reserves a byte (with the ASCII
   value 'C' and the numerical value 0x43). The rest of
   the word is 0's, because the next data item declared
   is an integer, and the next valid word address is
   0x10010004. The value of the integer is
   32,767 (base 10) = 0x00007fff.
   The third data item declared is a floating-point number
   with the value 3.2767 X 10^4 (its IEEE-754 floating-point
   representation is 0x46fffe00). The next data declaration
   is for a null-terminated string, "Hello, world!" Note
   that a null byte, 0x00 at address 0x10010019 immediately
   follows (i.e., terminates) the bytes of the string.
   (And so on...)

LITTLE-ENDIAN (Intel 80x86):

Here's the first part of a SPIM display of the data segment
of the program display.sal (listed below), with extra
annotations to explain what's going on:

```
DATA
[0x10000000]..[0x1000ffff] 0x00000000
<----- range of BYTE ---->   ^^^^^^^^
      addresses; each            |
       WORD contains             |
         0x00000000 -------------
```

Word addresses:

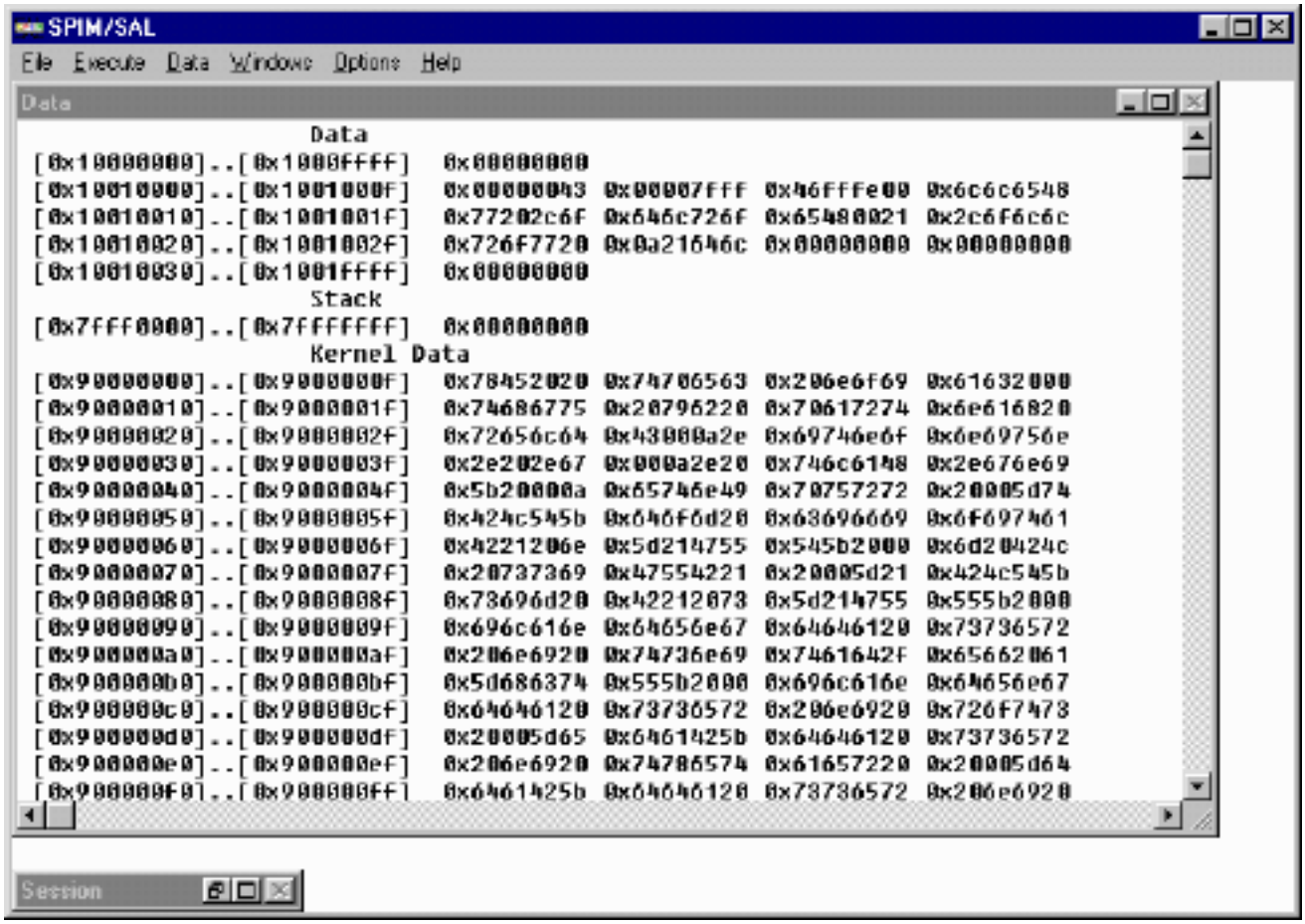
```
[0x10010000]..[0x1000000f] 0x00000043  0x00007fff  0x46fffe00  0x6c6c6548
<----- range of byte ----> ^^^^^^^^^^  ^^^^^^^^^^  ^^^^^^^^^^  ^^^^^^^^^^
         addresses           word        word        word        word
                           address is  address is  address is  address is
                           0x10010000  0x10010004  0x10010008  0x1001000c
                                       = add. of   = add. of   = add. of
                                       1st word    1st word    1st word
                                       + 4         + 8         + c

[0x10010010]..[0x1000001f] 0x77202c6f  0x646c726f  0x65480021  0x2c6f6c6c
<----- range of byte ----> ^^^^^^^^^^  ^^^^^^^^^^  ^^^^^^^^^^  ^^^^^^^^^^
         addresses           word        word        word        word
                           address is  address is  address is  address is
                           0x10010010  0x10010014  0x10010018  0x1001001c
                                       = add. of   = add. of   = add. of
                                       1st word    1st word    1st word
                                       + 4         + 8         + c
```

Byte addressing examples:

   The byte at address 0x1001000d is 0x6c. The byte at address
   0x1001000e is 0x65. Note that the byte ordering is DISPLAYED

as reversed only for character (single-byte) and string data. (The words at addresses 0x10010004 and 0x10010008 are numeric data, for which the creators of SPIM wisely decided not to reverse the displayed byte ordering). Similarly, the assembled MIPS instructions in the text segment are displayed with the same byte ordering on big-endian and little-endian machines.

```
# SAL program declare.sal
# Data segment
.data
c: .byte 'C'                    # C equivalent: char c;
#                                             c = 'C';
n: .word 32767                  # C equivalent: int n;
#                                             n = 32767;
f: .float 3.2767e4              # C equivalent: float f;
#                                             f = 3.2767e4;
greet0: .asciiz "Hello, world!"
greet: .ascii "Hello, world!"
newline: .byte '\n'
# Text segment (instructions)
.text
__start:     # Start of arithmetic & logical operations
put c        # C equivalent: printf("%c\n",c);
put newline  #
put n        # C equivalent: printf("%d\n",n);
put newline  #
put f        # C equivalent: printf("%6.6f\n",f);
put newline  #
puts greet0  # C equivalent: printf("Hello, world!");
put newline  #
done         # Return control to the OS
```


SPIM/SAL
File Execute Data Windows Options Help
Data
Data
[0x10000000]..[0x1000ffff] 0x00000000
[0x10010000]..[0x1001000f] 0x00000043 0x00007fff 0x46fffe00 0x6c6c6548
[0x10010010]..[0x1001001f] 0x77202c6f 0x646c726f 0x65480021 0x2c6f6c6c
[0x10010020]..[0x1001002f] 0x726f7720 0x0a21646c 0x00000000 0x00000000
[0x10010030]..[0x1001ffff] 0x00000000
Stack
[0x7fff0000]..[0x7fffffff] 0x00000000
Kernel Data
[0x90000000]..[0x9000000f] 0x78452020 0x74706563 0x206e6f69 0x61632000
[0x90000010]..[0x9000001f] 0x74686775 0x20796220 0x70617274 0x6e616820
[0x90000020]..[0x9000002f] 0x72656c64 0x43000a2e 0x69746e6f 0x6e69756e
[0x90000030]..[0x9000003f] 0x2e202e67 0x000a2e20 0x746c6148 0x2e676e69
[0x90000040]..[0x9000004f] 0x5b20000a 0x65746e49 0x70757272 0x20005d74
[0x90000050]..[0x9000005f] 0x424c545b 0x646f6d20 0x63696669 0x6f697461
[0x90000060]..[0x9000006f] 0x4221206e 0x5d214755 0x545b2000 0x6d20424c
[0x90000070]..[0x9000007f] 0x20737369 0x47554221 0x20005d21 0x424c545b
[0x90000080]..[0x9000008f] 0x73696d20 0x42212073 0x5d214755 0x555b2000
[0x90000090]..[0x9000009f] 0x696c616e 0x64656e67 0x64646120 0x73736572
[0x900000a0]..[0x900000af] 0x206e6920 0x74736e69 0x7461642f 0x65662061
[0x900000b0]..[0x900000bf] 0x5d686374 0x555b2000 0x696c616e 0x64656e67
[0x900000c0]..[0x900000cf] 0x64646120 0x73736572 0x206e6920 0x726f7473
[0x900000d0]..[0x900000df] 0x20005d65 0x6461425b 0x64646120 0x73736572
[0x900000e0]..[0x900000ef] 0x206e6920 0x74786574 0x61657220 0x20005d64
[0x900000f0]..[0x900000ff] 0x6461425b 0x64646120 0x73736572 0x206e6920
Session