Memory Addressing Modes

The MIPS architecture is a Load/Store architecture, which means the only instructions that access main memory are the load and store instructions. Only one addressing mode is implemented in the hardware. The addressing mode is referred to as base address plus displacement.

A load instruction accesses a value from memory and places a copy of the value found in memory in the register file. For example, the instruction:

```
lw    $s1, 8($a0)
```

computes the effective address of the memory location to be accessed by adding together the contents of register $a0 (the base address) and the constant value eight (the displacement). A copy of the value accessed from memory at the effective address is loaded into register $s1. The equivalent pseudocode statement would be:

```
$s1 = Mem[$a0 + 8]
```

The following is an example of a “Store Word” instruction:

```
sw    $s1, 12($a0)
```

When the hardware executes this instruction, it will compute the effective address of the destination memory location by adding together the contents of register $a0 and the constant value 12. A copy of the contents of register $s1 is stored in memory at the effective address. The equivalent pseudocode statement would be:

```
Mem[$a0 + 12] = $s1
```

From the point of view of an assembly language programmer, memory can be thought of as a very long linear array of locations where binary codes are stored. An effective address is a pointer to some location in this array.

Assembler Directives

To allocate space in memory for a one-dimensional array of 1024 integers, the following construct is used in C++:

```
int ARRAY[1024];
```
In MIPS, the corresponding construct is:

```assembly
.data
ARRAY: .space 4096
```

To *initialize a memory array* before program execution, say, with a set of 16 values corresponding to the powers of 2 \(2^N\) with N going from 0 to 15, the following construct is used in C++:

```c
int Pof2[16] = {1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768};
```

In MIPS, the corresponding construct is:

```assembly
.data
Pof2: .word 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768
```

Here is an example of how MIPS code can be written to access the eighth element in the array and place a copy of the value in register \(\$s0\). (The counts starts at zero.)

The *load address* `la` macro instruction is used to initialize a pointer in \(\$a0\) with the base address of the array labeled `Pof2`.

```assembly
la $a0, Pof2 # a0 = &Pof2
lw $s0, 8($a0) # s0 = MEM[a0 + 8]
```

The following program systematically echoes each element of an array on the terminal.
# PrintList.s
.data
  size:    .word  10
Array:   .word  1, 1, 2, 3, 5, 8, 13, 21, 34, 55
Bye:     .asciiz \nBye\n"
newline: .asciiz \n"

.text
main:
  lw  $s7, size       # get the size of the list in $s7
  li  $s1, 0         # set counter for the number of elements
  li  $s2, 0         # set offset from Array
print_loop:
  bge  $s1, $s7, print_loop_end  # stop after last elem is printed
  lw  $a0, Array($s2)  # print next value from the list
  li  $v0, 1
  syscall
  la  $a0, newline     # print a newline
  li  $v0, 4
  syscall
  addi $s1, $s1, 1    # increment the loop counter
  addi $s2, $s2, 4    # step to the next array element
  b print_loop        # repeat the loop
print_loop_end:
  li  $v0, 4
  la  $a0, Bye
  syscall
  li  $v0, 10         # exit
  syscall