Interacting with the OS

- We need the OS's help!!!
  - How to print a number? (output)
  - How to read a number? (input)
  - How to terminate (halt) a program?
  - How to open, close, read, write a file?
  - These are operating system "services"
- Special instruction: syscall
  - A "software interrupt" to invoke OS for an action (to do a *service*)
  - Need to **indicate the service to perform** (e.g., print vs. terminate)
  - May also need to **pass an argument value** (e.g., number to print)

A few useful syscalls

- syscall takes a service ID (number) in $v0
- Print integer
  - $v0=1, $a0=integer to print
- Read integer
  - $v0=5, after syscall, $v0 holds the integer read from keyboard
- Print string
  - $v0=4, $a0=memory address of string to print (null terminated)
- Exit (halt)
  - $v0=10, no argument
- See MARS docs for more!!! Also, attend recitation.

Example: Print an integer

```
# example as C program code
#   int a;
#   a = 10 + 8;
#   print("%d", a);
# code below carries out the above
li          $t0,10        # $t0 is a, $t0=10
addi        $t0,$t0,8     # $t0=10+8
li          $v0,1         # print service
add         $a0,$t0,$0    # pass value in $t0 to service
syscall                   # invoke OS to do service
li          $v0,10        # terminate program service
syscall                   # invoke OS to do service
```

Let's try it in MARS!!!! (mips1.asm)

Example: More useful output

```
   li          $t0,10              # put 10 into a
   addi        $t0,$t0,8           # do the add with 8
   li          $v0,4               # print string service
   la          $a0,msg             # load string
   syscall
   li          $v0,1               # print integer ervice
   add         $a0,$t0,$0
   li          $v0,10              # terminate program
   syscall
   # message to print before the number
.data
   msg:        .asciiz       "Sum of 10 + 8 is\n"
```

Let's try it in MARS!!!! (mips2.asm)

Example: Add 10 + x?

```
   li         $v0,4                # print prompt
   la         $a0,x_msg
   syscall
   li         $v0,5                # read integer service
   syscall
   move       $t0,$v0              # number read in $v0
   addi       $t0,$t0,10           # add number with 10
   li         $v0,4                # print result prompt
   la         $a0,msg
   syscall
   li         $v0,1                # print the sum
   move       $a0,$t0
   li         $v0,10               # terminate program
   syscall
.data
   x_msg:     .asciiz      "Number x to add?\n"
   msg:       .asciiz      "Sum of 10 + x is\n"
```

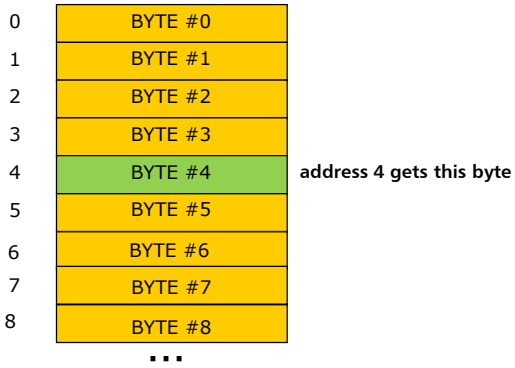

Let's try it in MARS!!!! (mips3.asm)

Memory transfer instructions

- Also called *memory access* instructions
- Only two types of instructions
  - Load: move data from memory to register
    - e.g., lw $s5, 4($t6) # $s5 ⇐ memory[$t6 + 4]
  - Store: move data from register to memory
    - e.g., sw $s7, 16($t3) # memory[$t3+16] ⇐ $s7
- In MIPS (32-bit architecture) there are memory transfer instructions for
  - 32-bit word: "int" type in C
  - 16-bit half-word: "short" type in C
  - 8-bit byte: "char" type in C

Memory view

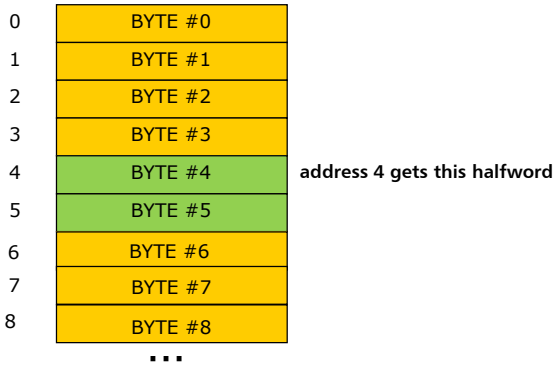
- Memory is a large, single-dimension 8-bit (byte) array with an address to each 8-bit item ("byte address")
- A ***memory address is just an index into the array***



- loads and stores give the index (address) to access

Memory view

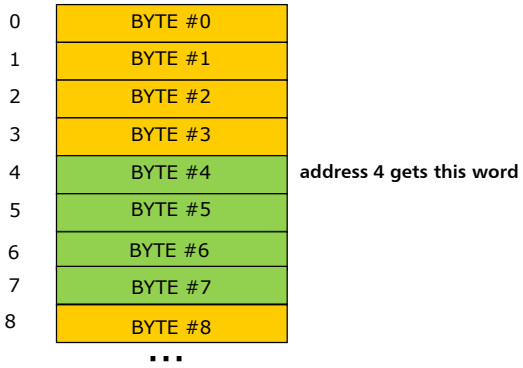
- Memory is a large, single-dimension 8-bit (byte) array with an address to each 8-bit item ("byte address")
- A ***memory address is just an index into the array***



- loads and stores give the index (address) to access

Memory view

- Memory is a large, single-dimension 8-bit (byte) array with an address to each 8-bit item ("byte address")
- A ***memory address is just an index into the array***



- loads and stores give the index (address) to access

Address calculation

- Memory address is specified with (**register, constant**) pair
  - Register to keep the ***base address***
  - Constant field to keep the ***offset*** from the base address
  - Address is, then (contents of register + offset)
  - The offset can be positive or negative
- Suppose base register $t0=64, then:

```
lw     $t0, 12($t1)        address = 64 + 12 = 76
lw     $t0, -12($t1)       address = 64 - 12 = 52
```

- MIPS uses this simple address calculation method; other architectures such as PowerPC and x86 support different methods

Machine code example

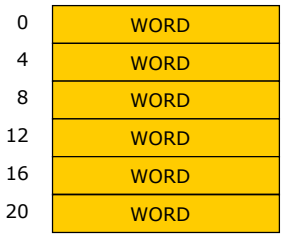
```
void swap(int v[], int k)
{
        int temp;
        temp = v[k];
        v[k] = v[k+1];
        v[k+1] = temp;
}
```

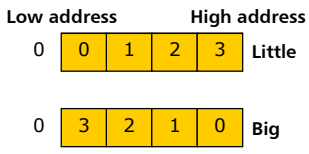
```
swap:
        sll     $t0, $a1, 2
        add     $t1, $a0, $t0
        lw      $t3, 0($t1)
        lw      $t4, 4($t1)
        sw      $t4, 0($t1)
        sw      $t3, 4($t1)
        jr      $ra
```

Let's try it in MARS!!!! (mips4.asm)

Memory organization

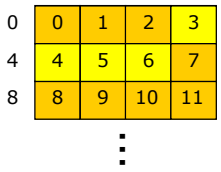
- 32-bit byte address
  - 2^{32} bytes with byte addresses from 0 to $2^{32} - 1$
  - 2^{30} words with byte addresses 0, 4, 8, …, $2^{32} - 4$
- Words are aligned
  - 2 least significant bits (LSBs) of an address are 0s
- Half words are aligned
  - LSB of an address is 0
- Addressing within a word
  - Which byte appears first and which byte the last?
  - Big-endian vs. little-endian
    - "Little end (LSB) comes first (at low address)"
    - "Big end (MSB) comes first (at low address)"





More on alignment

- A misaligned access
  - lw $s4, 3($t0)



- How do we define a word at address?
  - Data in byte 0, 1, 2, 3
    - If you meant this, use the address 0, not 3
  - Data in byte 3, 4, 5, 6
    - If you meant this, it is indeed misaligned!
    - Certain hardware implementation may support this; usually not
    - If you still want to obtain a word starting from the address 3 – get a byte from address 3, a word from address 4 and manipulate the two data to get what you want
- Alignment issue does not exist for byte access

Let's try it in MARS!!!! (mips5.asm)

Shift instructions

Name	Fields						Comments
R-format	op	NOT USED	rt	rd	shamt	funct	shamt is "shift amount"

- Bits change their positions inside a word
- <op> <r_{target}> <r_{source}> <shift amount>
- Examples
  - sll $s3, $s4, 4 # $s3 ⇐ $s4 << 4
  - srl $s6, $s5, 6 # $s6 ⇐ $s5 >> 6
- Shift amount can be in a register ("shamt" is not used)
- Shirt right arithmetic (sra) keeps the sign of a number
  - sra $s7, $s5, 4

Let's try it in MARS!!!! (mips6.asm)