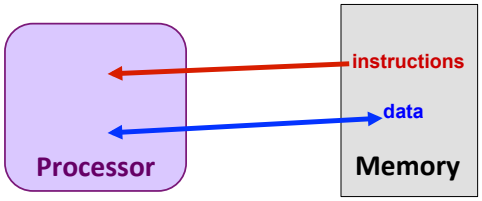
CS 240, Episode 2: Memory + MIPS

- Poll by email later today to set office hours.
- Reminder: PS1 due in one week.
- Today:
  - Review bytes, binary, hex
  - Words
  - Memory organization
  - Intro to MIPS architecture and assembly language

Note: today, we start mixing in existing 240 materials by Randy Shull.

Modern Digital Computer



How are data and instructions represented?

How does a program find its data in memory?

Review

byte = 8 bits

- **Conventional smallest unit of data**

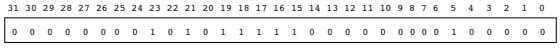
- **Binary** 00000000_2 -- 11111111_2
- **Decimal** 000_{10} -- 255_{10}
- **Hexadecimal** 00_{16} -- FF_{16}
  - Base 16
  - Practice
- Notation:
  - Binary: `0b1011`
  - Decimal: `11`
  - Hex: `0xB`

Hex	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

3

word

- **Normal/largest unit of data in machine**
- **ISA-dependent size -- MIPS: 4 bytes (32 bits)**
  - ISA = Instruction Set Architecture
- word size = register size = address size



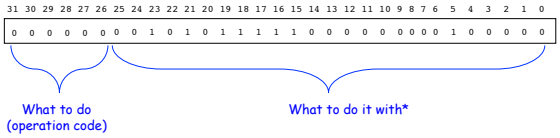
Java/C int = 4 bytes: 11,501,584

4

machine code

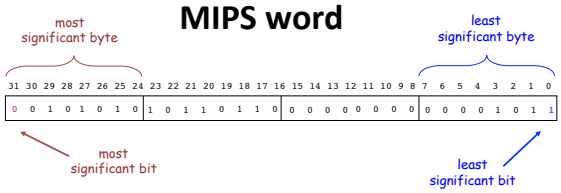
The binary representation of a computer's instruction set is known as its machine code

On MIPS, **1 machine code instruction = 1 word**

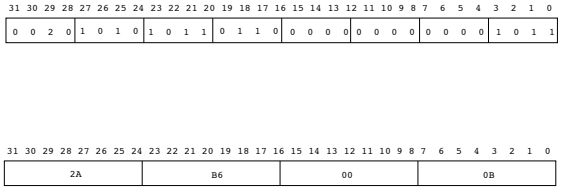


*Well, more or less.

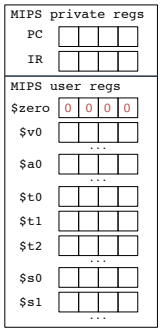
MIPS word



In hex



The MIPS processor

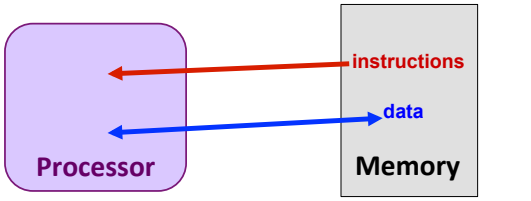


- **Register = word-sized storage**
- **32 registers**
  - **Named, like local variables.**
- [Invisible:]
  Hardware to implement operations using data in registers.

MIPS registers

Name	Register Number	Usage
$zero	0	the constant value 0
$v0-$v1	2-3	values for results and expression evaluation
$a0-$a3	4-7	arguments
$t0-$t7	8-15	temporaries
$s0-$s7	16-23	saved
$t8-$t9	24-25	more temporaries
$gp	28	global pointer
$sp	29	stack pointer
$fp	30	frame pointer
$ra	31	return address

Modern Digital Computer



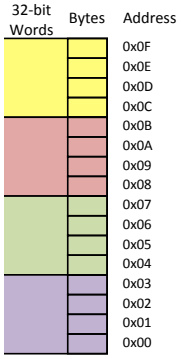
How does a program find its data in memory?

Byte-Addressable Memory

- **Memory = array of bytes (*locations*), unique *address* = index.**
- **Read/Write**
- Programs refer to bytes in memory by their *addresses.*
- Address = word
- Address space size?

Words in Memory

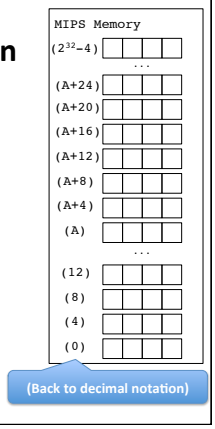
- **Address of word = address of 1st byte in word**
- ***Alignment***
  - Data of size *n bytes* stored at *a* only if *a* mod *n* = 0
    - *n* is a power of 2
    - Required (MIPS) or recommended (x86), depending on platform.
  - Why?



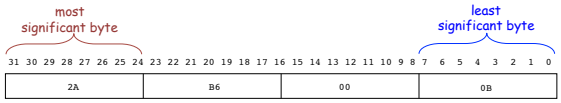
MIPS memory organization

- Up to 2^{32} bytes = 2^{30} words
- Word-aligned

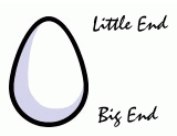
How are the bytes of a word ordered in memory?



Endianness: byte order *within memory words*

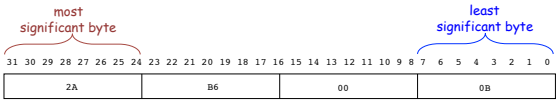


word in positional notation



Design 3-14

little endian (not us!)



word in positional notation

x86

Little-endian memory layout.

Address	Contents	
03	2A	MSB
02	B6	
01	00	
00	0B	LSB

Little end first:

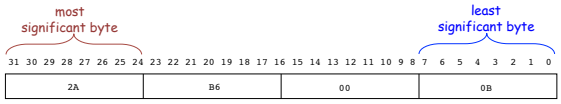
Least significant byte at lowest address.

Most significant byte at highest address.

Position increases as address increases.

Design 3-15

big endian (this one!)



word in positional notation

Big end first:

Least significant byte at highest address.

Most significant byte at lowest address.

Position decreases as address increases.

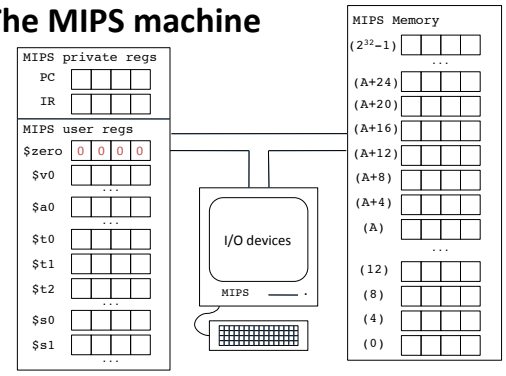
MIPS

Big-endian memory layout.

Address	Contents	
03	0B	LSB
02	00	
01	B6	
00	2A	MSB

Design 3-16

The MIPS machine



Hello, world!

MIPS assembly language

Let's write a program in assembly.

```
##############################################################
# helloWorld.asm                          #
# Our first assembly language program            #
# prints (you guessed it) "Hello world!" to screen       #
##############################################################
```


Assembler directives

```
##############################################################
# helloWorld.asm
# Our first assembly language program
# prints (you guessed it) "Hello world!" to screen
##############################################################
    .text               # program instructions
    .globl main         # where should execution start?
main:                   # label

    .data               # program data
```

Assembler directives start with .
Structure the program

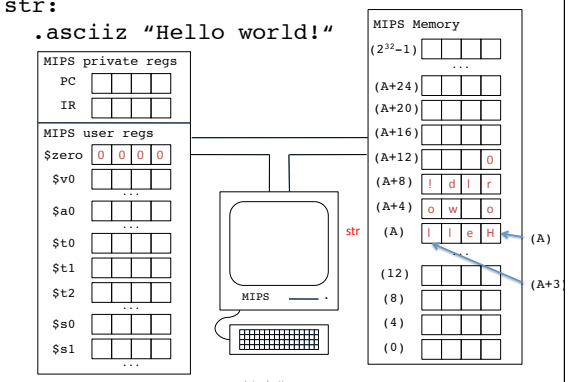
comment

First reserve room for our message

```
#############################################################
# helloWorld.asm                          #
# Our first assembly language program             #
# prints (you guessed it) "Hello world!" to screen       #
#############################################################
    .text               # program instructions
    .globl main
main:



    .data               # program data
str: .asciiz "Hello world!"
```

Another assembler directive


```
str:
  .asciiz "Hello world!"
```



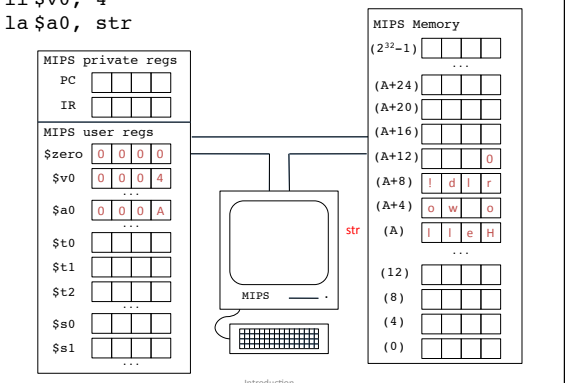
syscall does all the work (why?)

```
#############################################################
# helloWorld.asm                          #
# Our first assembly language program             #
# prints (you guessed it) "Hello world!" to screen       #
#############################################################
    .text               # program instructions
    .globl main
main:
    li  $v0, 4          # sys call code print_str
    la  $a0, str        # addr of string to print
    syscall             # print the prompt

    .data               # program data
str: .asciiz "Hello world!"
```

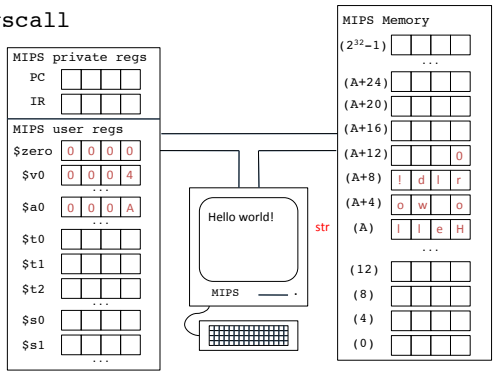
Pseudoinstructions

Okay OS do your thing


```
li $v0, 4
la $a0, str
```



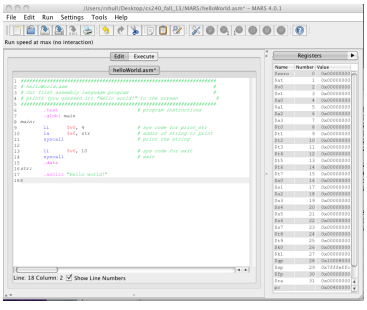
syscall



Finally we halt

```
#############################################################
# helloWorld.asm                          #
# Our first assembly language program             #
# prints (you guessed it) "Hello world!" to screen        #
#############################################################
    .text              # program instructions
    .globl main
main:
    li  $v0, 4         # sys call code print_str
    la  $a0, str       # addr of string to print
    syscall            # print the prompt
    li  $v0, 10        # sys code halt
    syscall            # halt

    .data              # program data
str: .asciiz "Hello world!"
```


Assembling and running our program*



*More in lab.

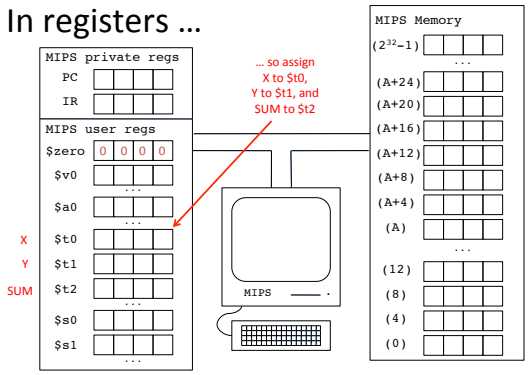
Our second assembly program

```
#############################################################
# addTwoNumbers.asm                          #
# This program computes the sum of two numbers        #
# X and Y that are input during program execution.        #
#############################################################
# program instructions
                # read 1st num into location X
                # read 2nd num into location Y

                # add Y to X and store in AC
                # put the sum into location SUM
                # write SUM to screen
                # and stop
# program variables
```

Where should X, Y and SUM go?

```
############################################################
# addTwoNumbers.asm                        #
# This program computes the sum of two numbers        #
# X and Y that are input during program execution.       #
############################################################
    .text              # program instructions
    .globl main
main:                  # input X
                       # input Y

                       # add X to Y and
                       # store SUM
                       # output SUM
                       # halt
    .data              # program data
```


In registers …



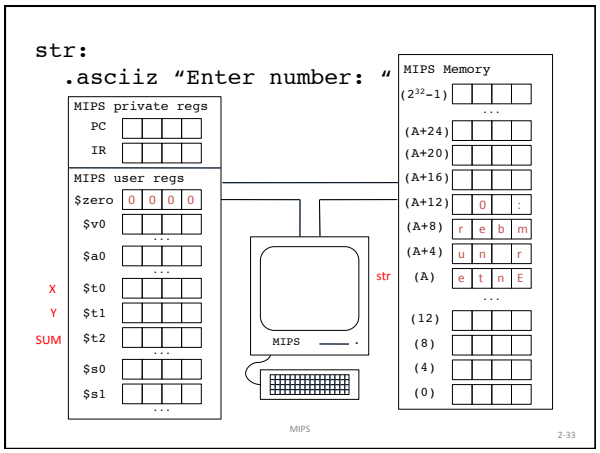
How do we input values to X and Y?

```
############################################################
# addTwoNumbers.asm                        #
# This program computes the sum of two numbers        #
# X and Y that are input during program execution.       #
############################################################
    .text              # program instructions
    .globl main
main:                  # input X
                       # input Y

                       # add X to Y and
                       # store SUM
                       # output SUM
                       # halt
    .data              # program data
```


First reserve room for prompt

```
############################################################
# addTwoNumbers.asm                        #
# This program computes the sum of two numbers        #
# X and Y that are input during program execution.       #
############################################################
    .text              # program instructions
    .globl main
main:                  # input X
                       # input Y

                       # add X to Y and
                       # store SUM
                       # output SUM
                       # halt
    .data              # program data
str: .asciiz "Enter number: "
```

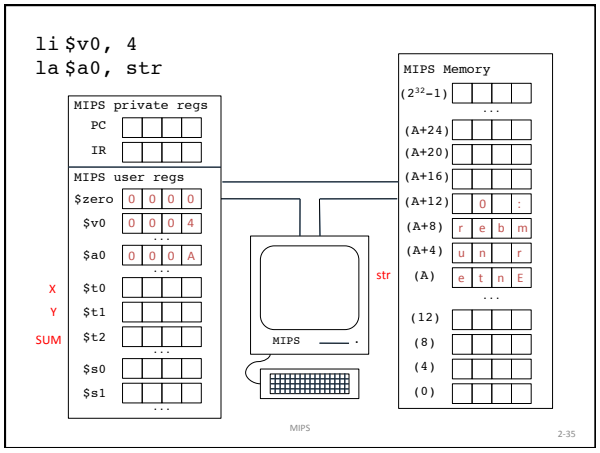
Another assembler directive

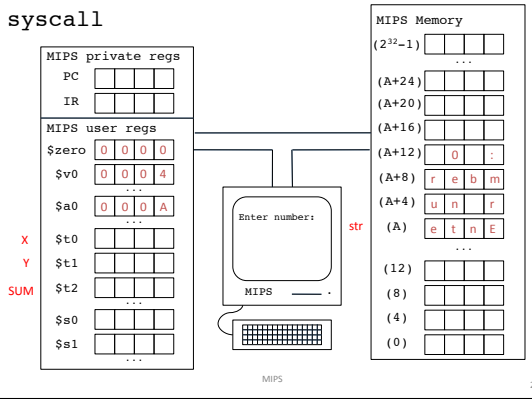


Reading in an integer

```
#############################################################
# addTwoNumbers.asm                                         #
# We're going need some more room up here real soon.  #
#############################################################
    .text               # program instructions
    .globl main     Pseudoinstructions
main:
    li  $v0, 4          # sys call code print_str
    la  $a0, str        # addr of string to print
    syscall             # print the prompt
                        # input x; input Y
                        # add X to Y and
                        # store SUM
                        # output SUM
                        # halt
    .data               # program data
str: .asciiz “Enter number: “
```

Okay OS do your thing

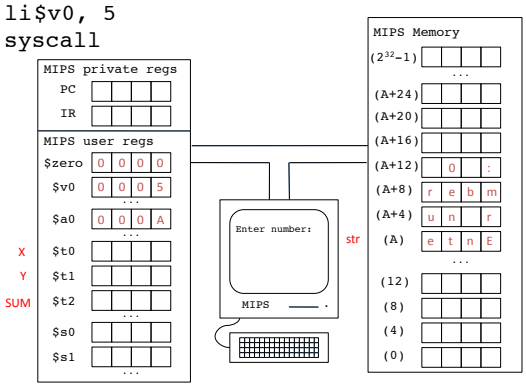




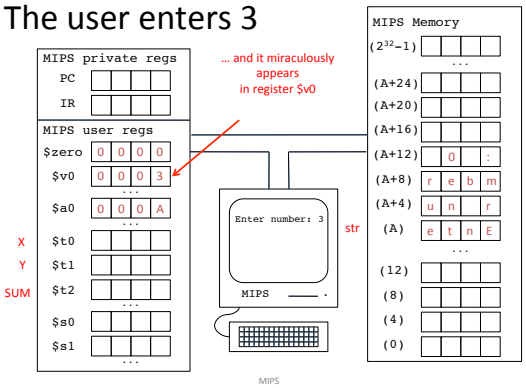
Reading in X

```
    .text               # program instructions
    .globl main
main:
    li  $v0, 4          # sys call code print_str
    la  $a0, str        # addr of string to print
    syscall             # print the prompt
    li  $v0, 5          # sys code read integer
    syscall
    move    $t0, $v0    # $t0 is the variable X
                        # input Y
                        # add X to Y and
                        # store SUM
                        # output SUM
                        # halt
    .data               # program data
str: .asciiz "Enter number: "
```

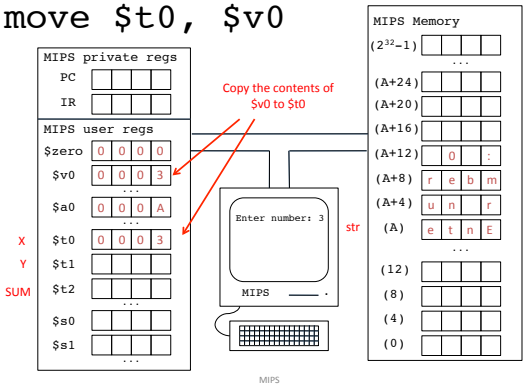

li $v0, 5
syscall



The user enters 3



move $t0, $v0



Second verse, same as the first*

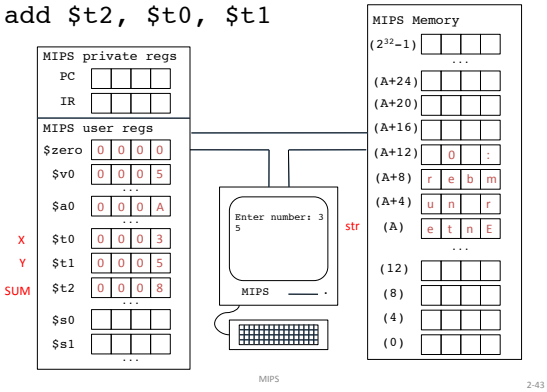
```
    .text               # program instructions
    .globl main
main:
    li  $v0, 4          # sys call code print_str
    la  $a0, str        # addr of string to print
    syscall             # print the prompt
    li  $v0, 5          # sys code read integer
    syscall
    move    $t0, $v0    # $t0 is the variable X
    li  $v0, 5          # sys code read integer
    syscall
    move    $t1, $v0    # $t1 is the variable Y
                        # add X to Y & store SUM
                        # output SUM
                        # halt
    .data               # program data
str: .asciiz "Enter number: "
```

*We should print another prompt here, but PowerPoint space is tight.

Adding is easy

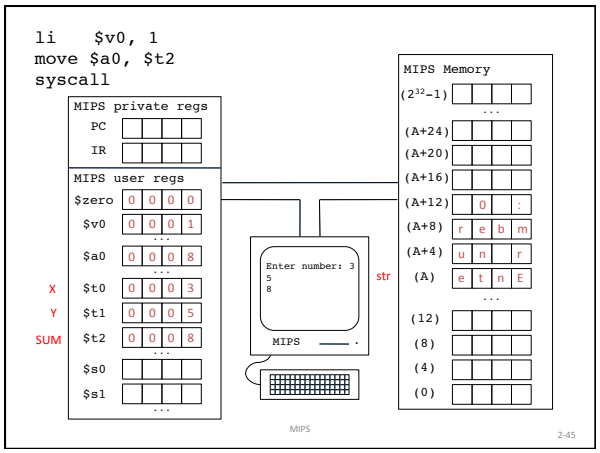
```
    .text               # program instructions
    .globl main
main:
    li  $v0, 4          # sys call code print_str
    la  $a0, str        # addr of string to print
    syscall             # print the prompt
    li  $v0, 5          # sys code read integer
    syscall
    move    $t0, $v0    # $t0 is the variable X
    li  $v0, 5          # sys code read integer
    syscall
    move    $t1, $v0    # $t1 is the variable Y
    add $t2, $t0, $t1   # SUM <- X + Y
                        # output SUM
                        # halt
    .data               # program data
str: .asciiz "Enter number: "
```


add $t2, $t0, $t1



Now all we need to do is print …

```
main:
    li  $v0, 4          # sys call code print_str
    la  $a0, str        # addr of string to print
    syscall             # print the prompt
    li  $v0, 5          # sys code read integer
    syscall
    move    $t0, $v0    # $t0 is the variable X
    li  $v0, 5          # sys code read integer
    syscall
    move    $t1, $v0    # $t1 is the variable Y
    add $t2, $t0, $t1   # SUM <- X + Y
    li  $v0, 1          # sys code print integer
    move    $a0, $t2    # integer to print
    syscall             # now print it
                        # halt
```



… and halt

```
main:
    li  $v0, 4          # sys code print_str
    la  $a0, str        # addr of string to print
    syscall             # print the prompt
    li  $v0, 5          # sys code read integer
    syscall
    move    $t0, $v0       # $t0 is the variable X
    li  $v0, 5          # sys code read integer
    syscall
    move    $t1, $v0       # $t1 is the variable Y
    add $t2, $t0, $t1      # SUM <- X + Y
    li  $v0, 1          # sys code print integer
    move    $a0, $t2       # integer to print
    syscall             # now print it
    li  $v0, 10         # sys code halt
    syscall             # that's all, folks.
```

MIPS 2-46

Service	Mode	Arguments	Result
print_int	1	$a0 = integer	
print_float	2	$f12 = float	
print_double	3	$f12 = double	
print_string	4	$a0 = string	
read_int	5		integer in $v0
read_float	6		float in $f0
read_double	7		double in $f0
read_string	8	$a0 = buffer, $a1 = length	
sbrk	9	$a0 = amount	address in $v0
exit	10		
print_char	11	$a0 = char	
read_char	12		char in $v0
open	13	$a0=filename(string), $a1=flags, $a2=mode	file descriptor in $a0
read	14	$a0=file descriptor, $a1=buffer, $a2=length	num chars read in $a0
write	15	$a0=file descriptor, $a1=buffer, $a2=length	num chars written in $a0
close	16	$a0 = file descriptor	
exit2	17	$a0 = result	

MIPS 2-47