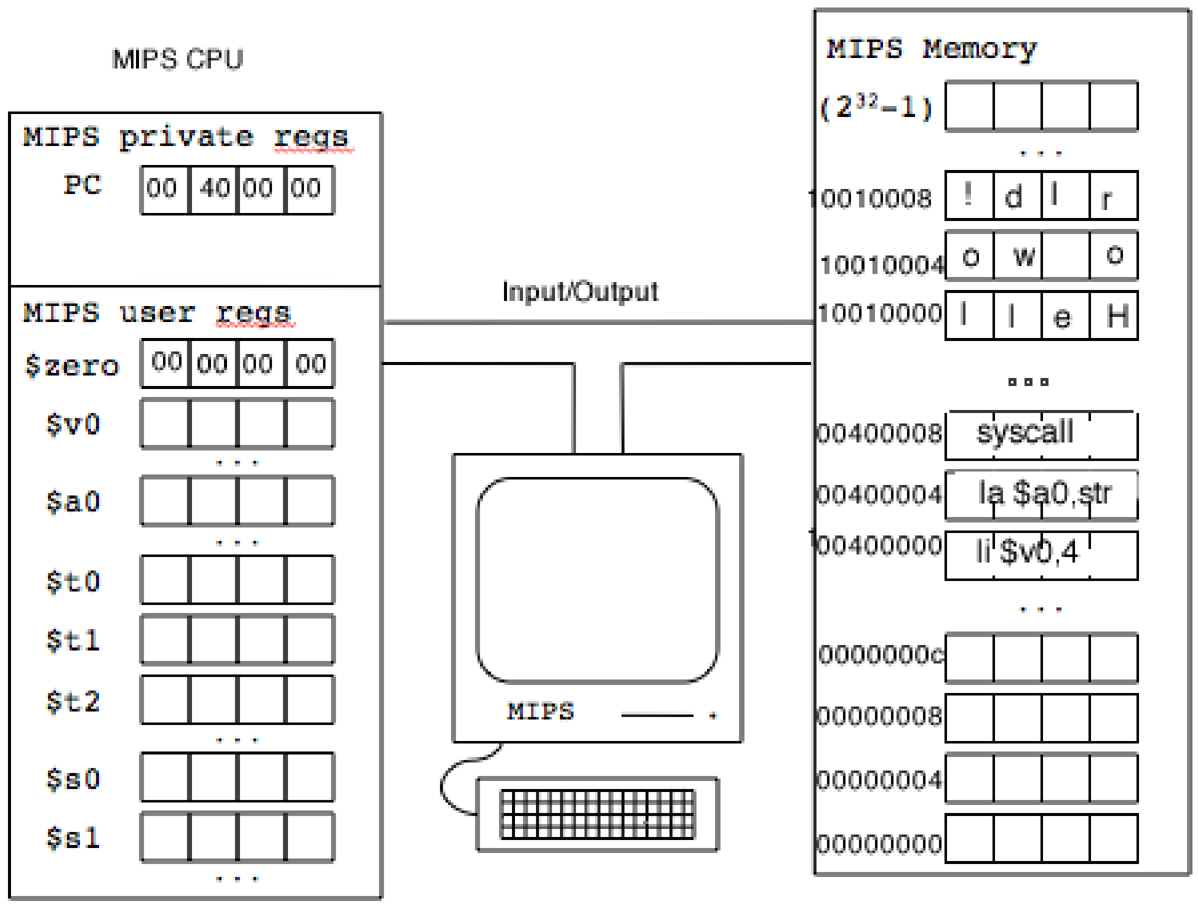
Introduction to MIPS/MARS

Computer = programmable information processing machine.

Major components = CPU, Memory, and Input/Output (often called I/O)



CPU: contains 32 user registers = storage for data within the CPU

`$zero`	- Contains a value of 0
`$t0 - $t9`	- Temporaries
`$s0 - $s7`	- Saved
`$a0 - $a3`	- Arguments
`$v0 and $v1`	- Return values

The contents of these registers are used as operands in the program instructions. Although all these registers are general purpose (can be used to store any kind of data), there are conventions for typical usage, as indicated.

There are also 7 other registers with specific purposes that you will soon learn more about.

- Each register contains a 32-bit value (32-digit binary/base 2 number)
- A **byte** is an 8-bit unit, a **word** is a 4-byte or 32-bit unit.
- We often represent the values in hexadecimal (base 16), to be concise

$zero always contains a value of 0, which is stored in 32 bits:

```
0000 0000  0000 0000  0000 0000 0000 0000₂ , is equivalent to:
  0    0     0    0     0    0    0    0₁₆
```

Groups of 4 binary digits can easily be converted to hexadecimal:

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

Another example of binary to hexadecimal conversion:

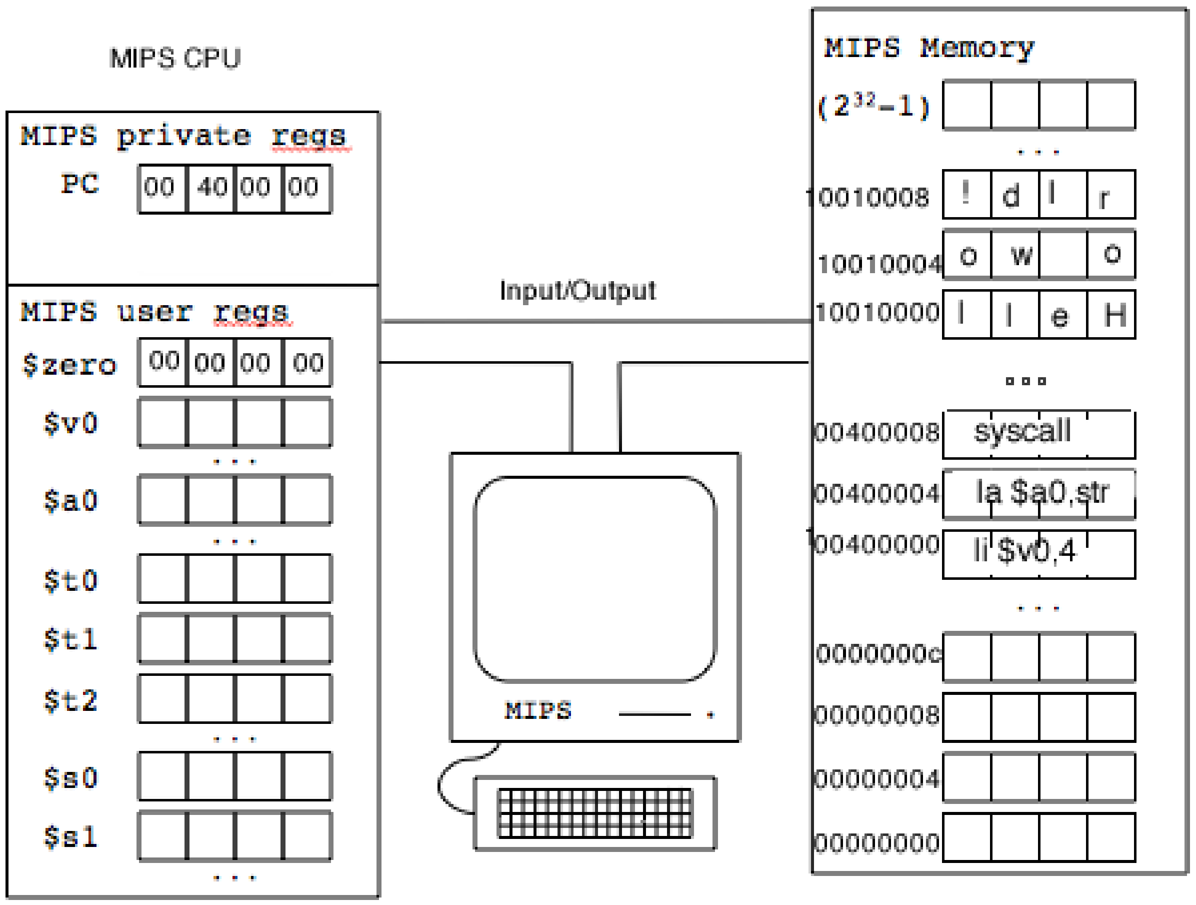
```
0010 0100  0011 0101  1000 1100  0001 1111₂ ,  is equivalent to:
  2    4     3    5     8    C     1    F₁₆
```

Memory:

There are $2^{32} - 1$ locations or **addresses** in memory in which a byte (8 bits) of information can be stored.

The locations can contain data or program instructions (although these are shown as mnemonics below, they are actually stored as numeric values).

Each instruction in MIPS is 32 bits (one **word**) long.



Address	**Memory Usage in MARS**
10040000_{16}	↑ Heap and Stack (learn more later)
10010000_{16}	↑ Data Segment (Data with fixed size)
00400000_{16}	↑ Text Segment (Program Instructions)
00000000_{16}	↑ Reserved

MIPS instructions for use in Lab #1

Arithmetic (R-type) Instructions

`add  $t3,$t1,$t2` #**add**, $t3 <- contents of $t1 + contents of $t2

`addi  $t3, $t1,5` #**add immediate**,$t3 <- contents of $t1 + 5

Memory Access Instructions

`lw   $t1,label` #**load word**, $t1 <- value of word stored at memory address/location specified by *label*

`lw  $t1,3($s0)` #**load word**, $t1 <= value of word stored at memory address (base address in $s0 + 3)

`sw  $t1,label` #**store word**, stores value of word in $t1 to address/location in memory specified by *label*
(**lw** and **sw** can also be byte or halfword, i.e. **lb**, **lh, sb,sh**)

Pseudo-instructions

Not part of the basic MIPS instruction set but used for programmer's convenience. These instructions translate to basic MIPS instructions when the program is assembled (but those basic instructions are not as intuitive from a programmer's perspective).

`li   $t1, 3` #**load immediate**, $t1 <- 3

`la  $s1, label` #**load address**, $s1 <- address corresponding to *label*

`move $t1,$t2` #**move**, move contents of $t2 to $t1

Directives – are not instructions, but you use them in your program to tell the assembler how to store your program in memory

.text
.globl main
Precedes your **text segment** (program instructions), and specifies **main** as a global symbol (recognized by other files in a multi-file project

.data
Precedes your **data segment** (data declarations)
(text segment can come before data segment, or vice versa)

.ascii "string"
Defines a string of characters (each character is stored as a 1-byte ascii value)

.asciiz "string"
Defines a null-terminated string (ends with a null byte)

.byte b0,b1,b2

Defines and initializes subsequent bytes in memory

.half h0,h1,h2

Defines and initializes subsequent half-words (16-bit values – alignment forced to next even address

.word w0,w1,w2

Defines and initializes subsequent words (32-bit values) – alignment forced to next word address (multiple of 4)

.space n

allocates n bytes of space, usually initialized to 0

SYSCALL functions overview

System services used for input/output (I/O)

How to use SYSCALL system services

1. Load the service number in register $v0.
2. Load argument values, if any, in $a0, $a1, or $a2
3. Issue the SYSCALL instruction.
4. Retrieve return values, if any, from result registers

Table of Commonly Used Services

Service	$v0	Arguments	Result
print integer	1	$a0 = integer to print	
print string	4	$a0 = address of null terminated string to print	
read integer	5		$v0 contains integer read
read string	8	$a0=address of input buffer $a1=max. # of chars. to read	
exit (stop execution)	10		
print character	11	$a0=character to print	
read character	12		$v0 contains character read
open file	13	$a0=address of null-terminated string containing filename $a1=flags $a2=mode	$a0 contains file descriptor (- if error)
read from file	14	$a0 = file descriptor $a1=address of output buffer $a2=max. # of chars to read	$a0 contains # of chars read (0=EOF,- if error)
write to file	15	$a0 = file descriptor $a1=address of output buffer $a2= # of chars to write	$a0 contains # chars written (- if error)
close file	16	$a0 = file descriptor	

Examples of Simple I/O for lab #1

#print an integer

```
li $v0,1   # load service number into $v0
li $a0,5   # load value to be printed into $a0
syscall
```

#print a null-terminated string

```
li $v0,4   #load service number in $v0
la $a0,prompt_string
           #load address of string to be printed into $a0
Syscall    #the null-terminated string must also be defined in the data segment!

.data
prompt_string:  .asciiz "Enter a value: "
```

#read in an integer

```
li $v0,5        #load service number in $v0
syscall         #the value entered by the user is returned in $v0
move $t0,$v0    #store value entered into another register
```

#read in a string

```
li $v0,8        #load service number in $v0
la $a0,answer   #put address of answer string in $a0
lw $a1,alength  #put length of string in $a1
syscall         #answer and alength must be defined in data segment!

.data
answer:  .space 50   #allocate space for string to be stored
alength: .word 50    #length of string to be entered
```

#terminate execution of program

```
           #should always be the final instructions executed in program
li $v0,10  #load service number in $v0
syscall
```
