Computer Architecture
Tutorial 07

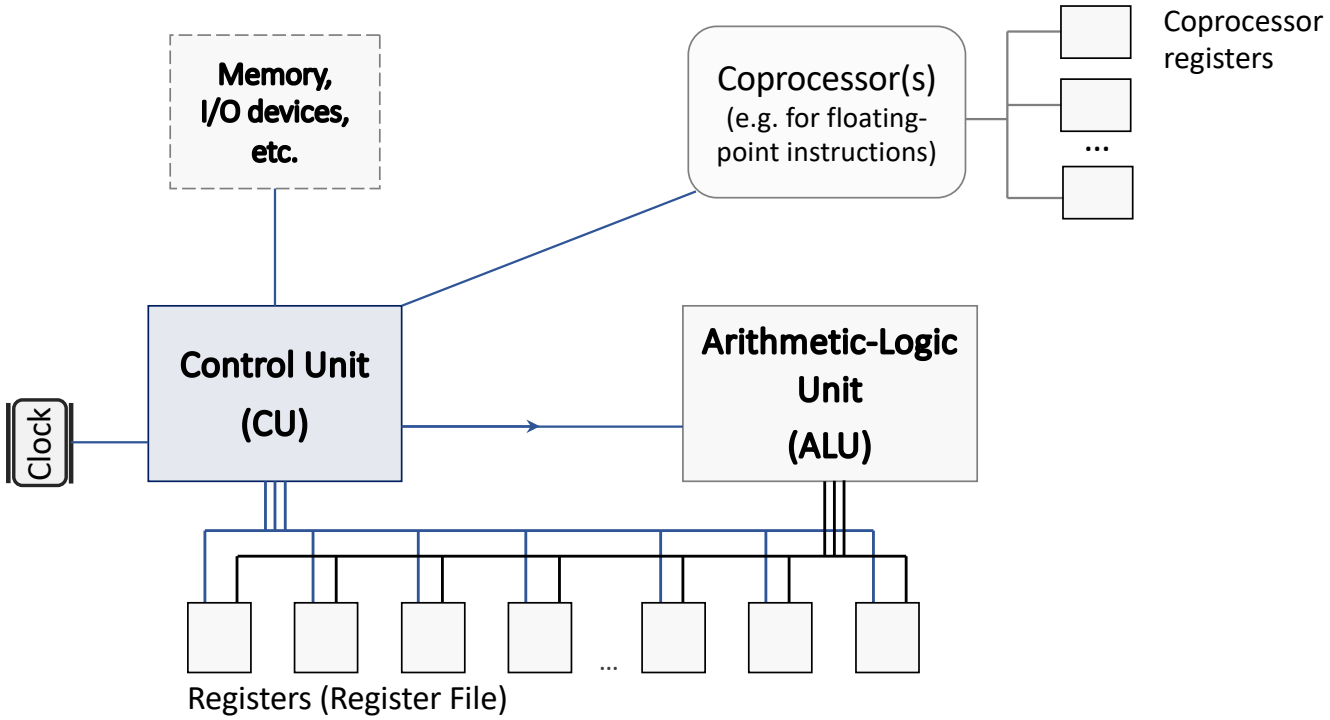
MIPS Instruction Set

Artem Burmyakov, Alexander Tormasov

October 07, 2021

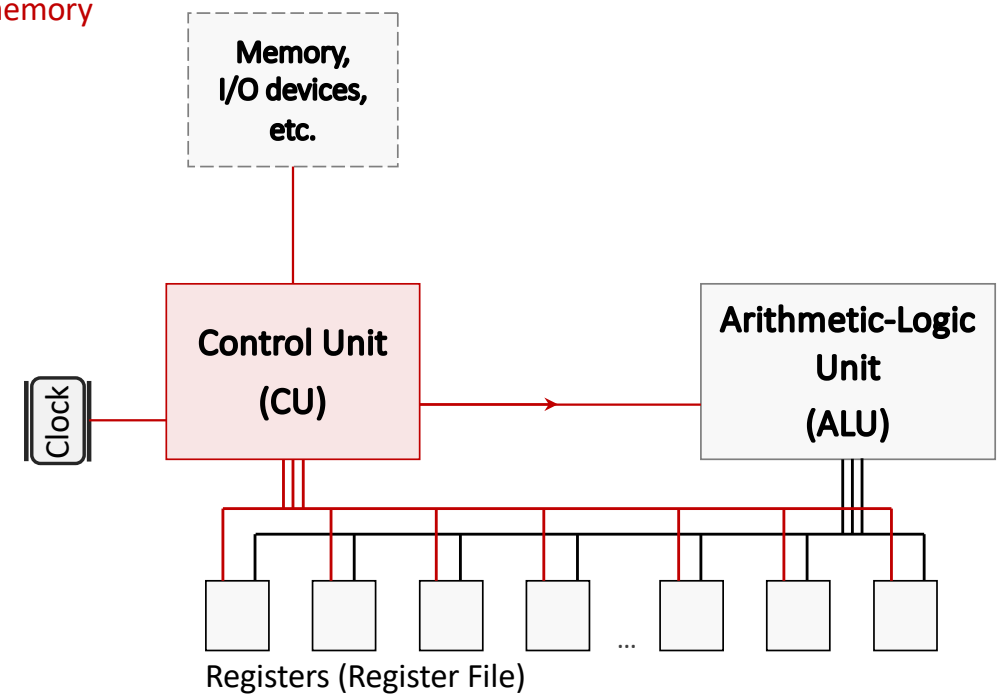
INNOPOLIS UNIVERSITY

MIPS Architecture Layout



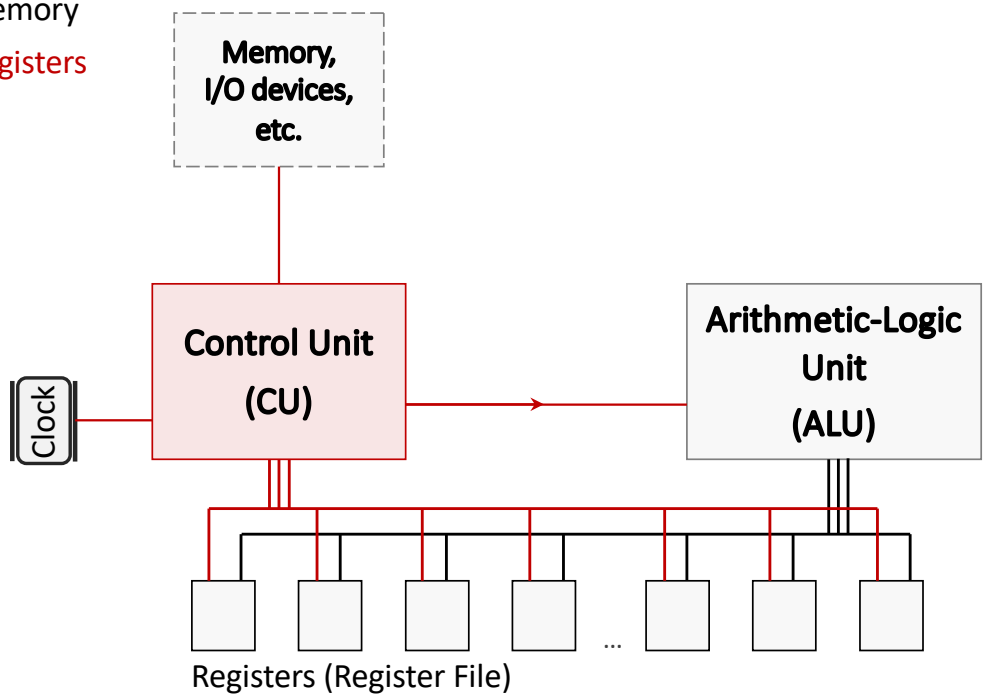
MIPS Architecture Layout

- Fetches instructions from memory



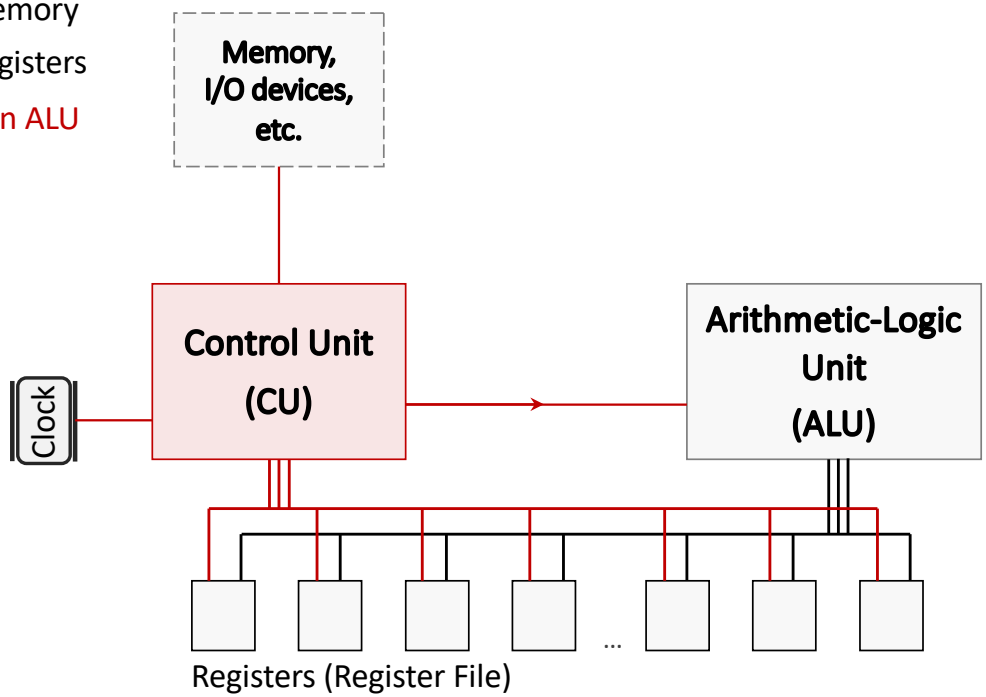
MIPS Architecture Layout

- Fetches instructions from memory
- Decodes instructions over registers



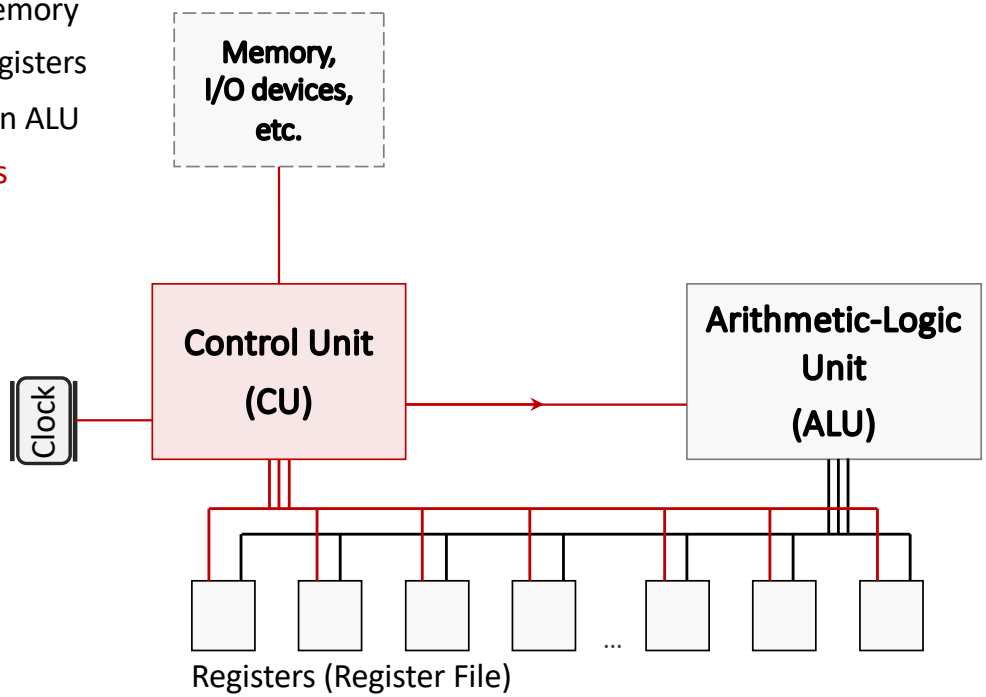
MIPS Architecture Layout

- Fetches instructions from memory
- Decodes instructions over registers
- Controls the execution flow in ALU



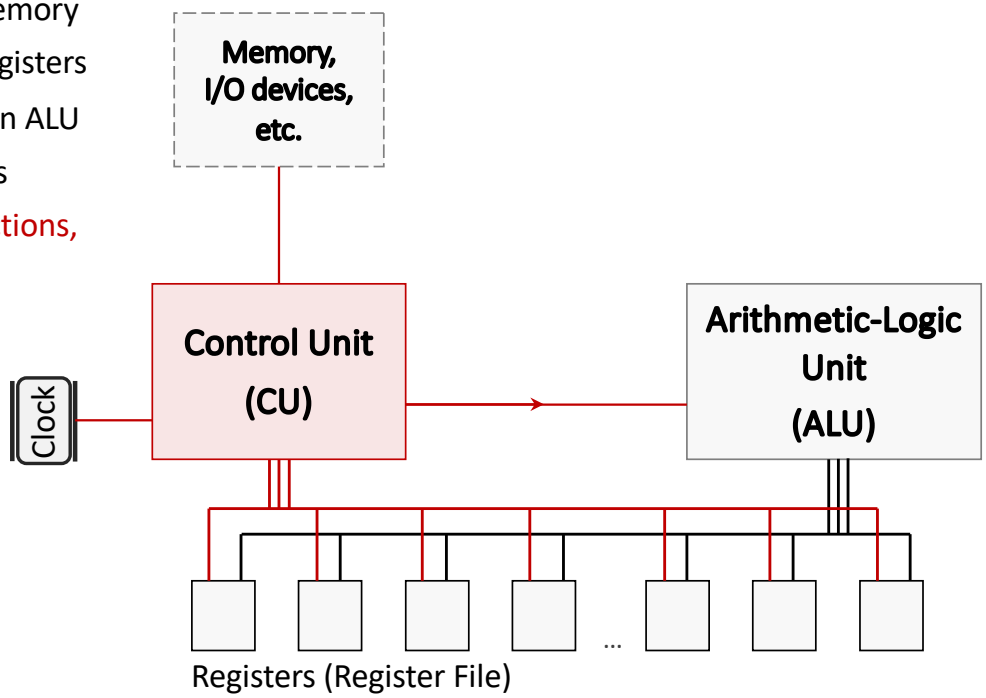
MIPS Architecture Layout

- Fetches instructions from memory
- Decodes instructions over registers
- Controls the execution flow in ALU
- Communicates to I/O devices



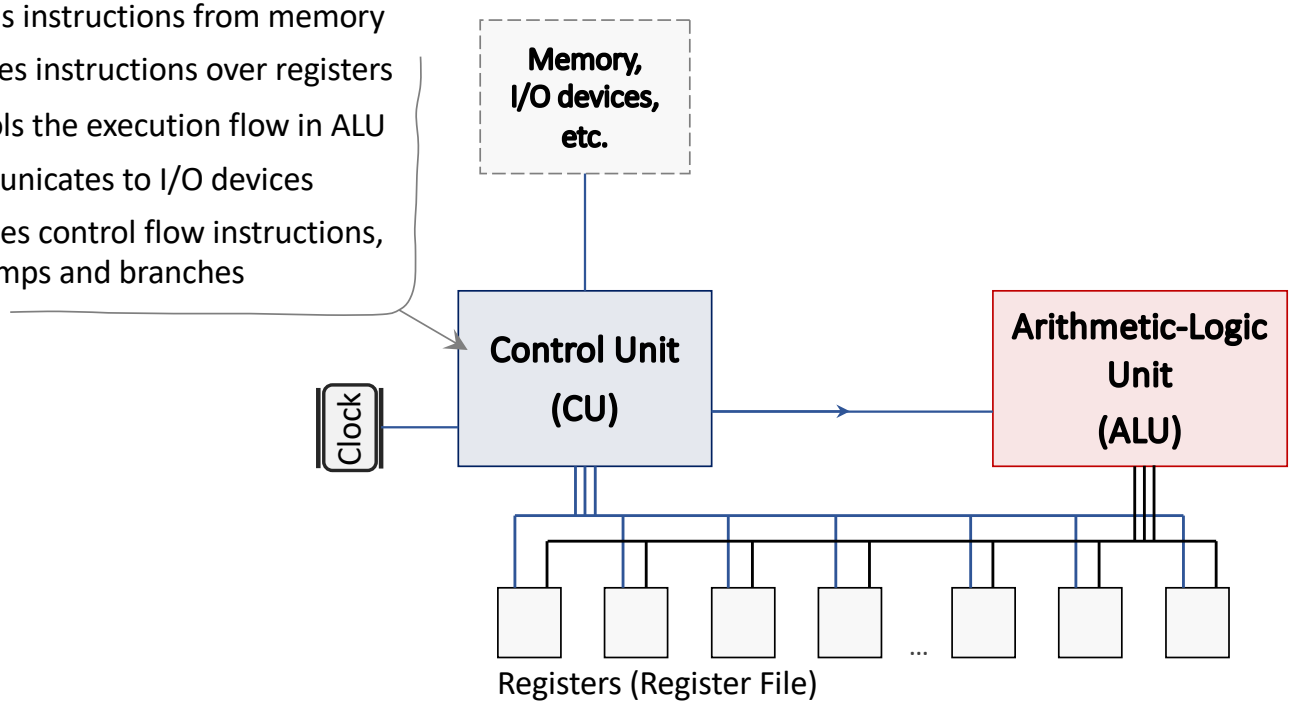
MIPS Architecture Layout

- Fetches instructions from memory
- Decodes instructions over registers
- Controls the execution flow in ALU
- Communicates to I/O devices
- Executes control flow instructions, e.g. jumps and branches



MIPS Architecture Layout

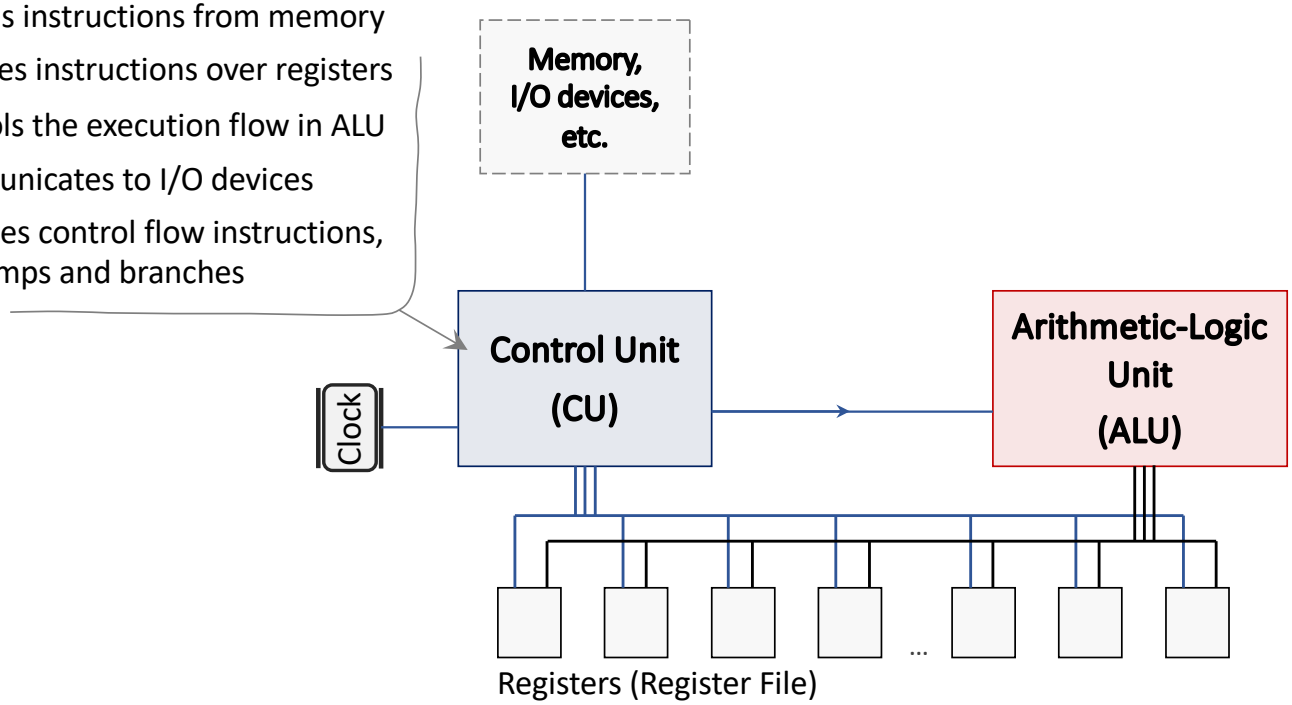
- Fetches instructions from memory
- Decodes instructions over registers
- Controls the execution flow in ALU
- Communicates to I/O devices
- Executes control flow instructions, e.g. jumps and branches



- Executes arithmetic and logic instructions

MIPS Architecture Layout

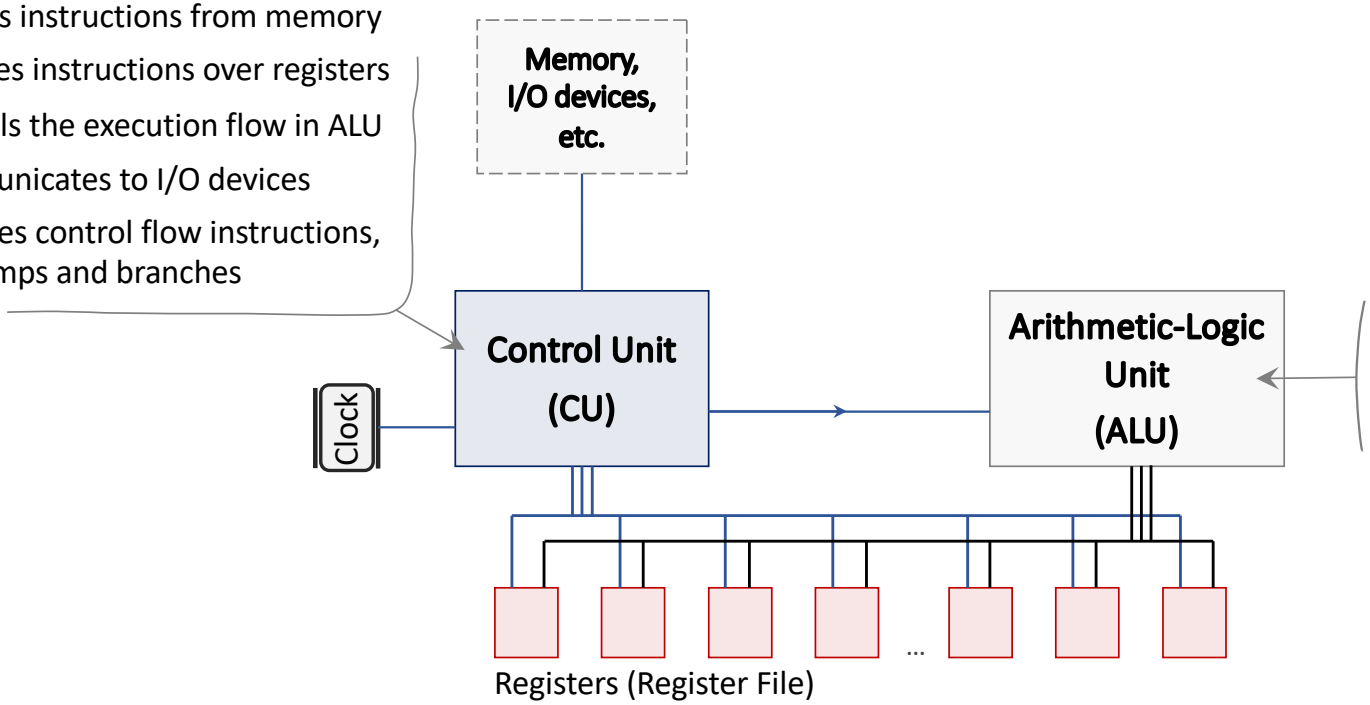
- Fetches instructions from memory
- Decodes instructions over registers
- Controls the execution flow in ALU
- Communicates to I/O devices
- Executes control flow instructions, e.g. jumps and branches



- Executes arithmetic and logic instructions
- Designed for the minimized propagation delay

MIPS Architecture Layout

- Fetches instructions from memory
- Decodes instructions over registers
- Controls the execution flow in ALU
- Communicates to I/O devices
- Executes control flow instructions, e.g. jumps and branches

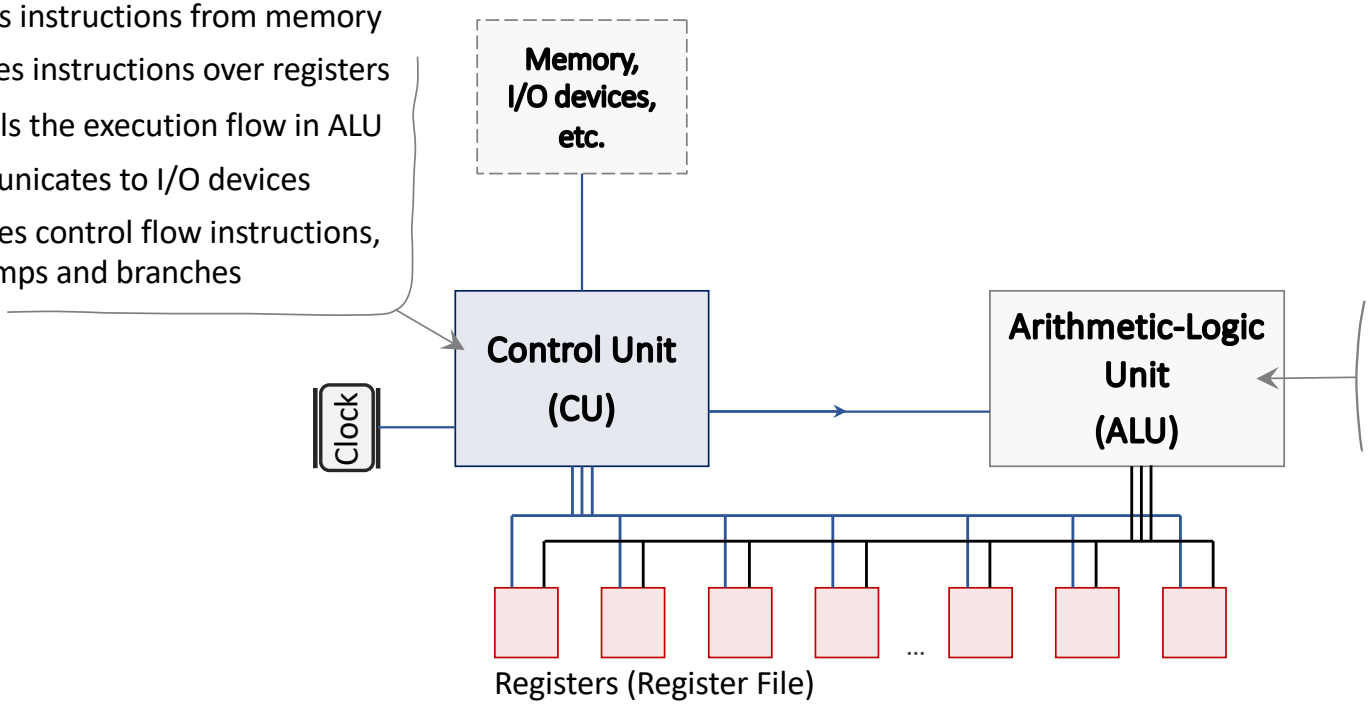


- Executes arithmetic and logic instructions
- Designed for the minimized propagation delay

- Fast memory elements, directly connected to CU and ALU

MIPS Architecture Layout

- Fetches instructions from memory
- Decodes instructions over registers
- Controls the execution flow in ALU
- Communicates to I/O devices
- Executes control flow instructions, e.g. jumps and branches

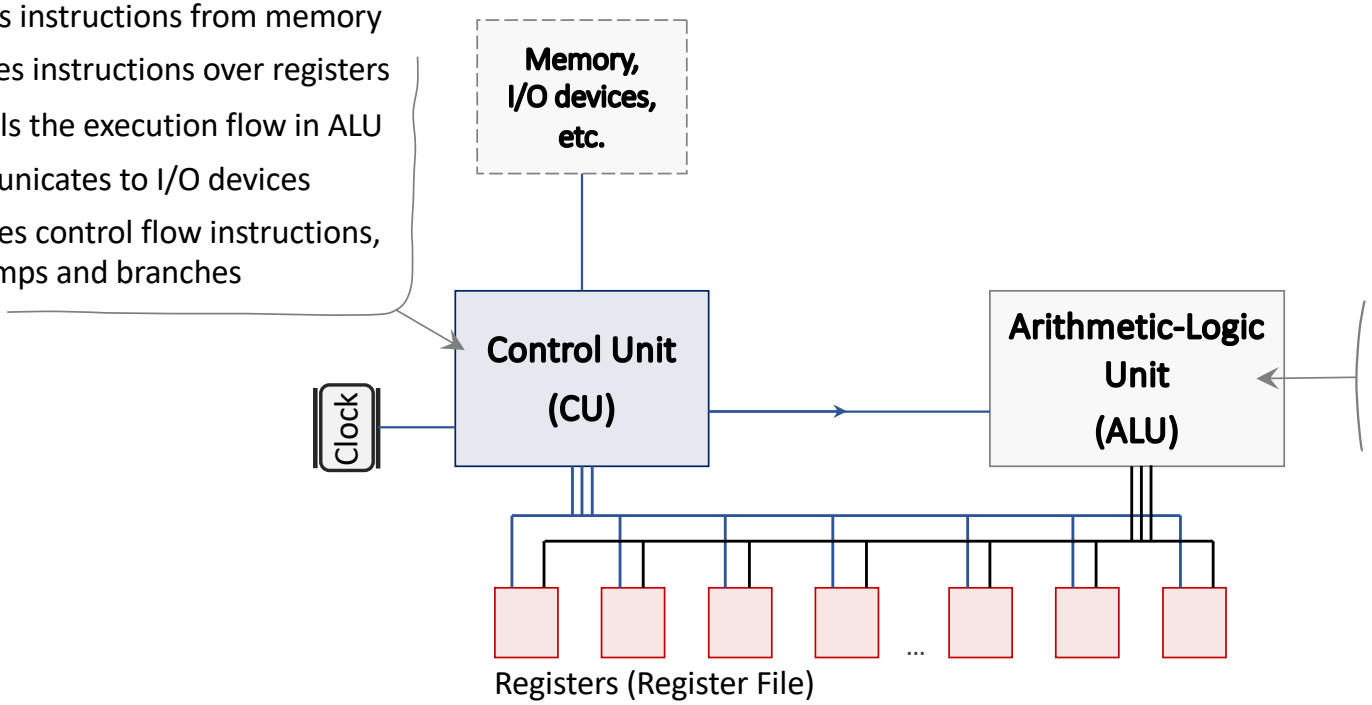


- Executes arithmetic and logic instructions
- Designed for the minimized propagation delay

- Fast memory elements, directly connected to CU and ALU
- Store data for instruction to execute (e.g. instruction code, its input arguments), as well as the result of its execution

MIPS Architecture Layout

- Fetches instructions from memory
- Decodes instructions over registers
- Controls the execution flow in ALU
- Communicates to I/O devices
- Executes control flow instructions, e.g. jumps and branches



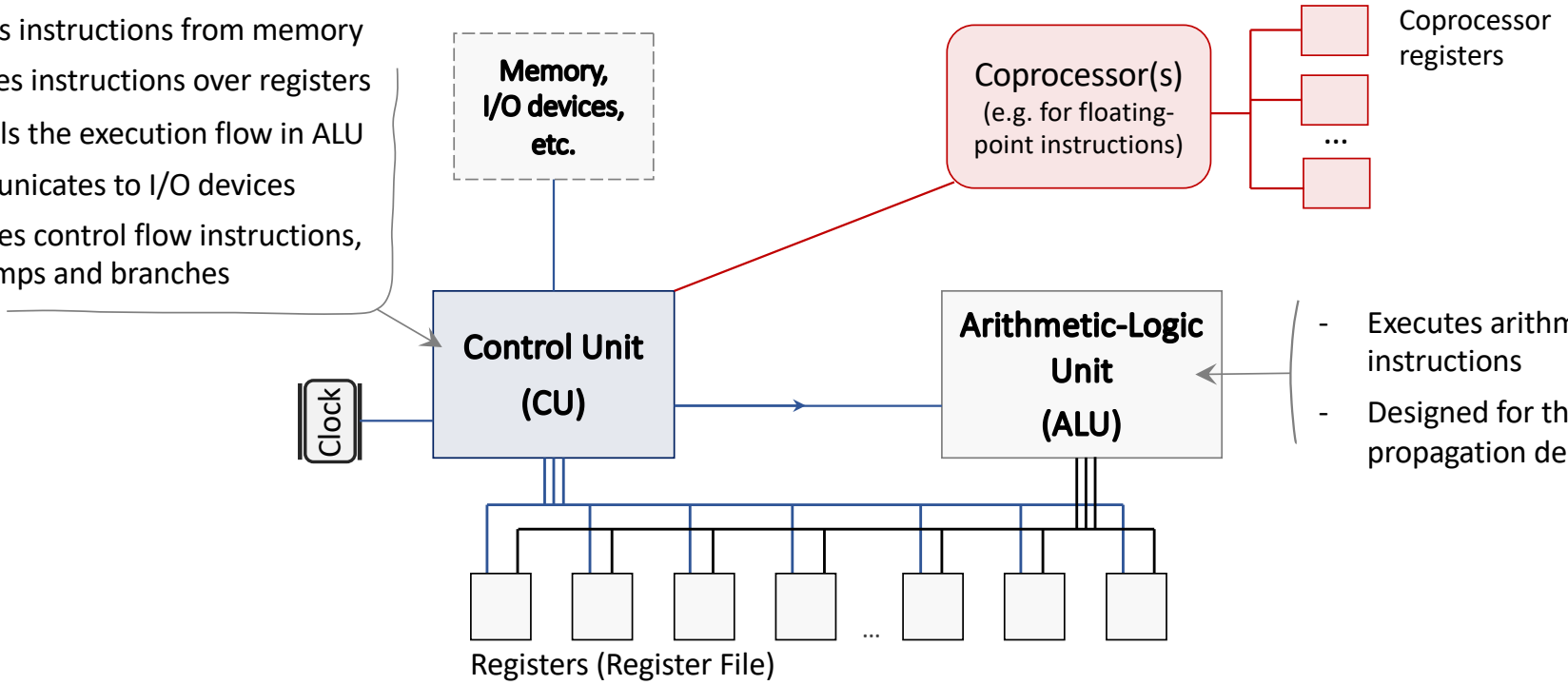
- Executes arithmetic and logic instructions
- Designed for the minimized propagation delay

- Fast memory elements, directly connected to CU and ALU
- Store data for instruction to execute (e.g. instruction code, its input arguments), as well as the result of its execution
- Every register is reserved for a specific purpose (e.g. to store input arguments, or the result of computation)

MIPS Architecture Layout

- Circuit optimized for a limited set of specific instructions (e.g. floating-point)

- Fetches instructions from memory
- Decodes instructions over registers
- Controls the execution flow in ALU
- Communicates to I/O devices
- Executes control flow instructions, e.g. jumps and branches



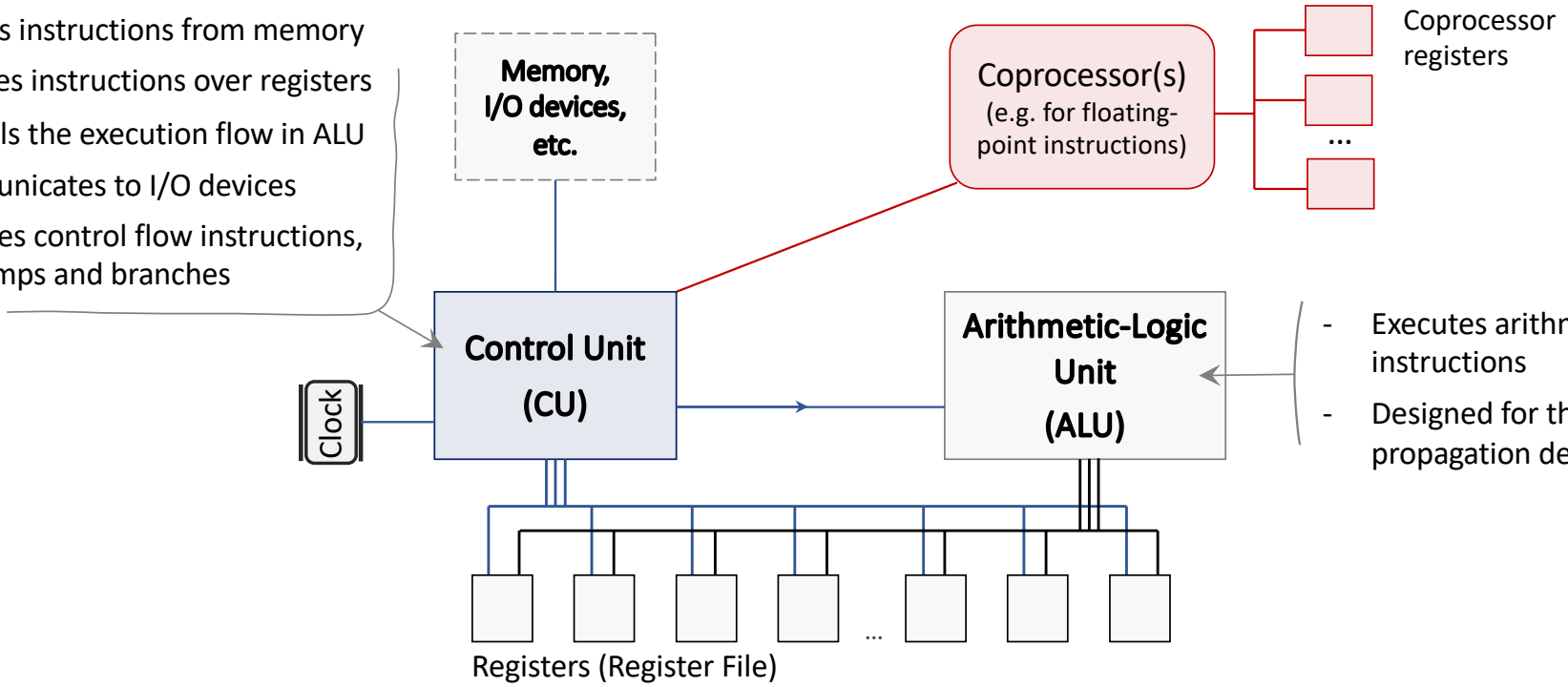
- Executes arithmetic and logic instructions
- Designed for the minimized propagation delay

- Fast memory elements, directly connected to CU and ALU
- Store data for instruction to execute (e.g. instruction code, its input arguments), as well as the result of its execution
- Every register is reserved for a specific purpose (e.g. to store input arguments, or the result of computation)

MIPS Architecture Layout

- Circuit optimized for a limited set of specific instructions (e.g. floating-point)
- Has its own registers

- Fetches instructions from memory
- Decodes instructions over registers
- Controls the execution flow in ALU
- Communicates to I/O devices
- Executes control flow instructions, e.g. jumps and branches



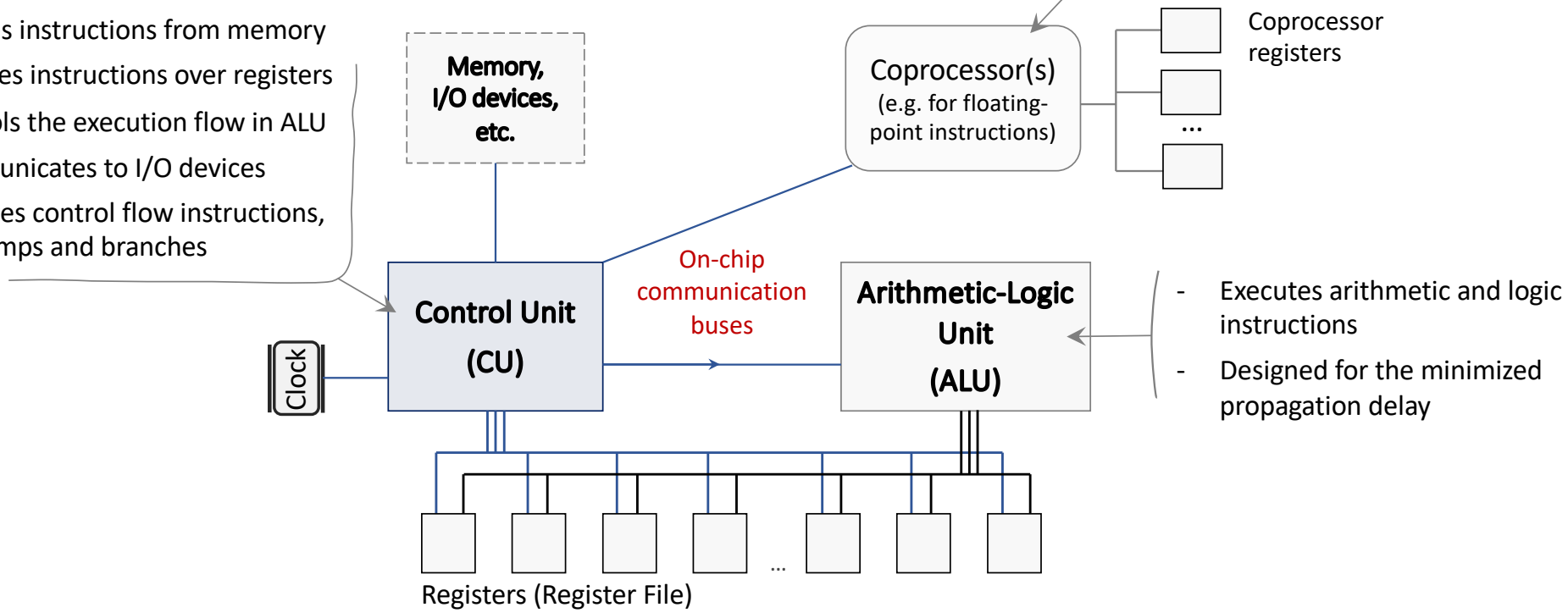
- Executes arithmetic and logic instructions
- Designed for the minimized propagation delay

- Fast memory elements, directly connected to CU and ALU
- Store data for instruction to execute (e.g. instruction code, its input arguments), as well as the result of its execution
- Every register is reserved for a specific purpose (e.g. to store input arguments, or the result of computation)

MIPS Architecture Layout

- Circuit optimized for a limited set of specific instructions (e.g. floating-point)
- Has its own registers

- Fetches instructions from memory
- Decodes instructions over registers
- Controls the execution flow in ALU
- Communicates to I/O devices
- Executes control flow instructions, e.g. jumps and branches

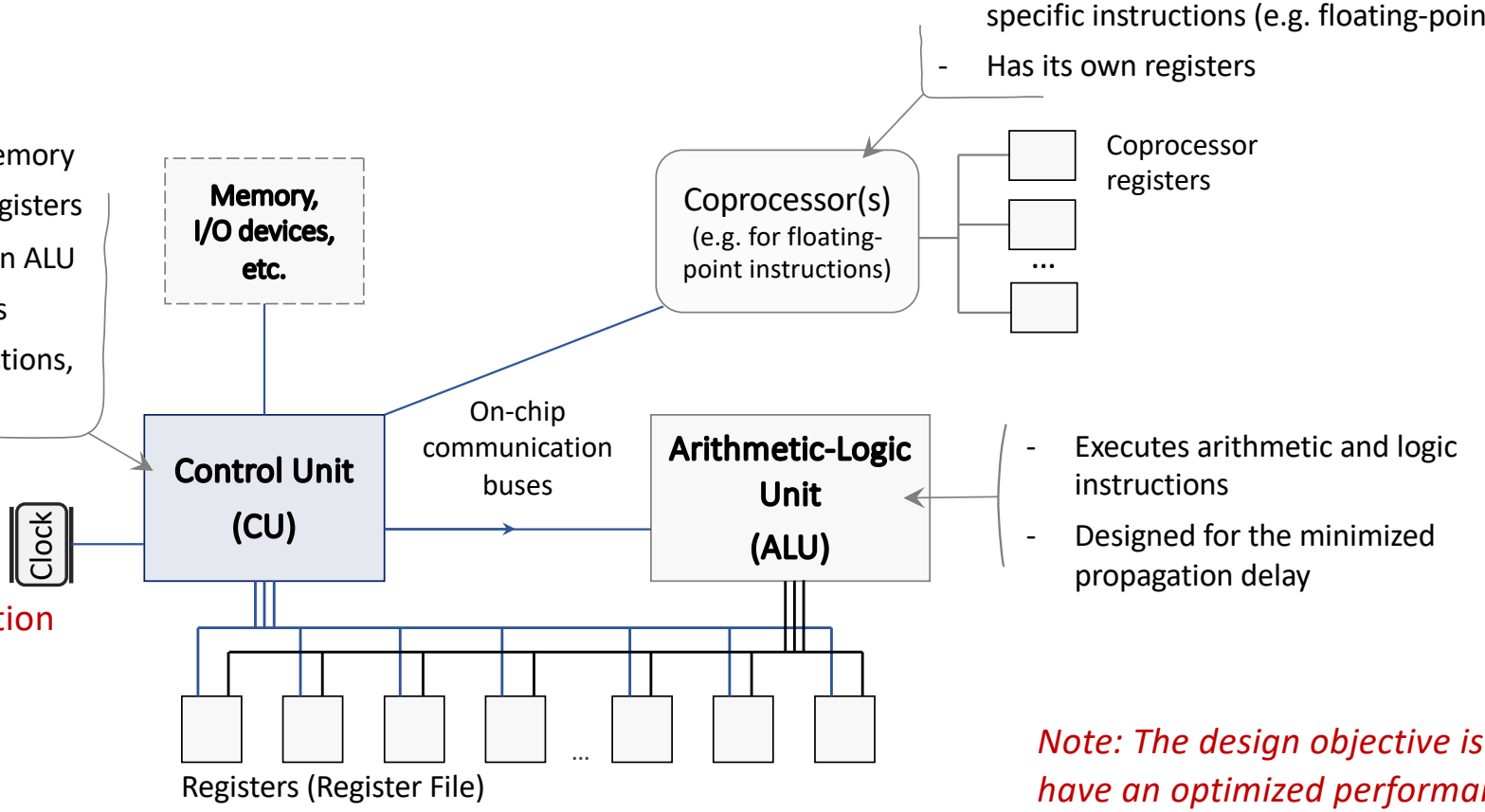


- Executes arithmetic and logic instructions
- Designed for the minimized propagation delay

- Fast memory elements, directly connected to CU and ALU
- Store data for instruction to execute (e.g. instruction code, its input arguments), as well as the result of its execution
- Every register is reserved for a specific purpose (e.g. to store input arguments, or the result of computation)

Some Aspects Affecting the CPU Clock Frequency

- Fetches instructions from memory
- Decodes instructions over registers
- Controls the execution flow in ALU
- Communicates to I/O devices
- Executes control flow instructions, e.g. jumps and branches



- Circuit optimized for a limited set of specific instructions (e.g. floating-point)
- Has its own registers

- Executes arithmetic and logic instructions
- Designed for the minimized propagation delay

The worst-case propagation delay of the slowest instruction drastically affects CPU clock frequency

The number of registers:

The worst-case propagation delay tends to increase for a larger number of registers

(due to longer wires, multiplexers with a larger number of inputs, etc.)

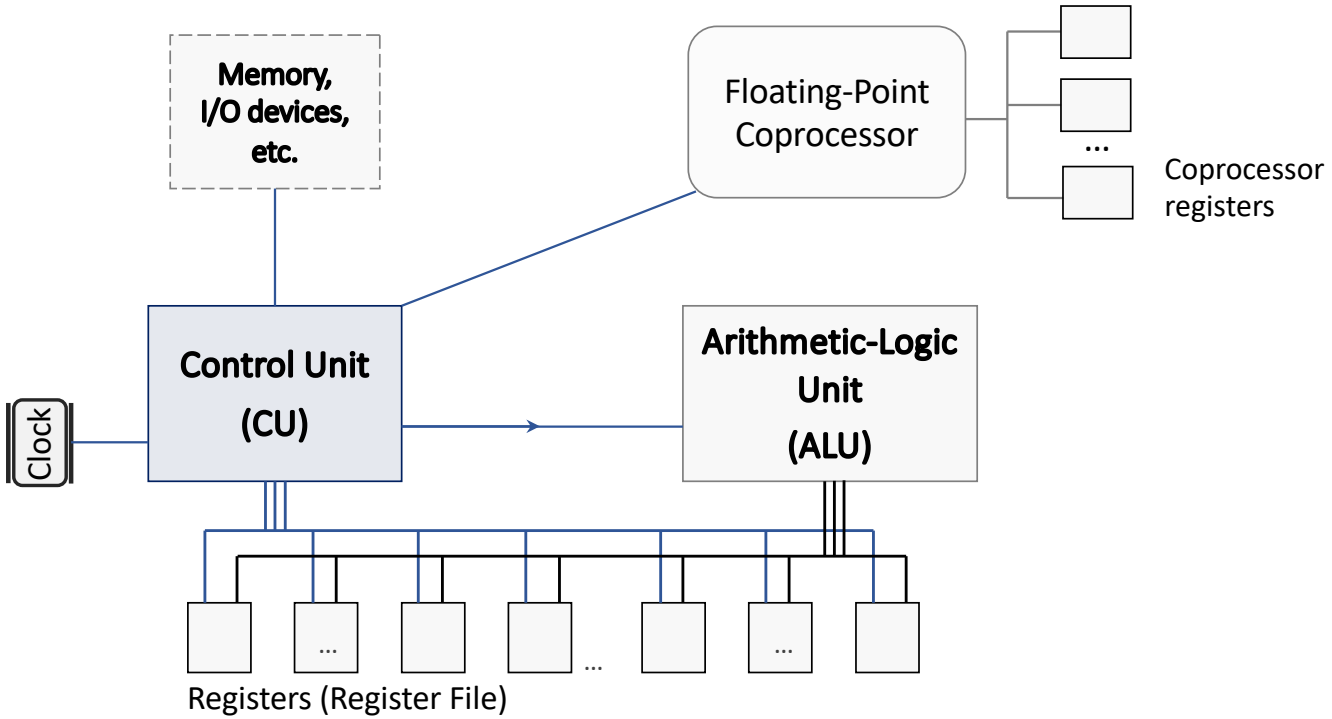
- Fast memory elements, directly connected to CU and ALU
- Store data for instruction to execute (e.g. instruction code, its input arguments), as well as the result of its execution
- Every register is reserved for a specific purpose (e.g. to store input arguments, or the result of computation)

Note: The design objective is to have an optimized performance in the average use case, and not to have a higher number of instructions supported

(recall the difference between RISC and CISC)

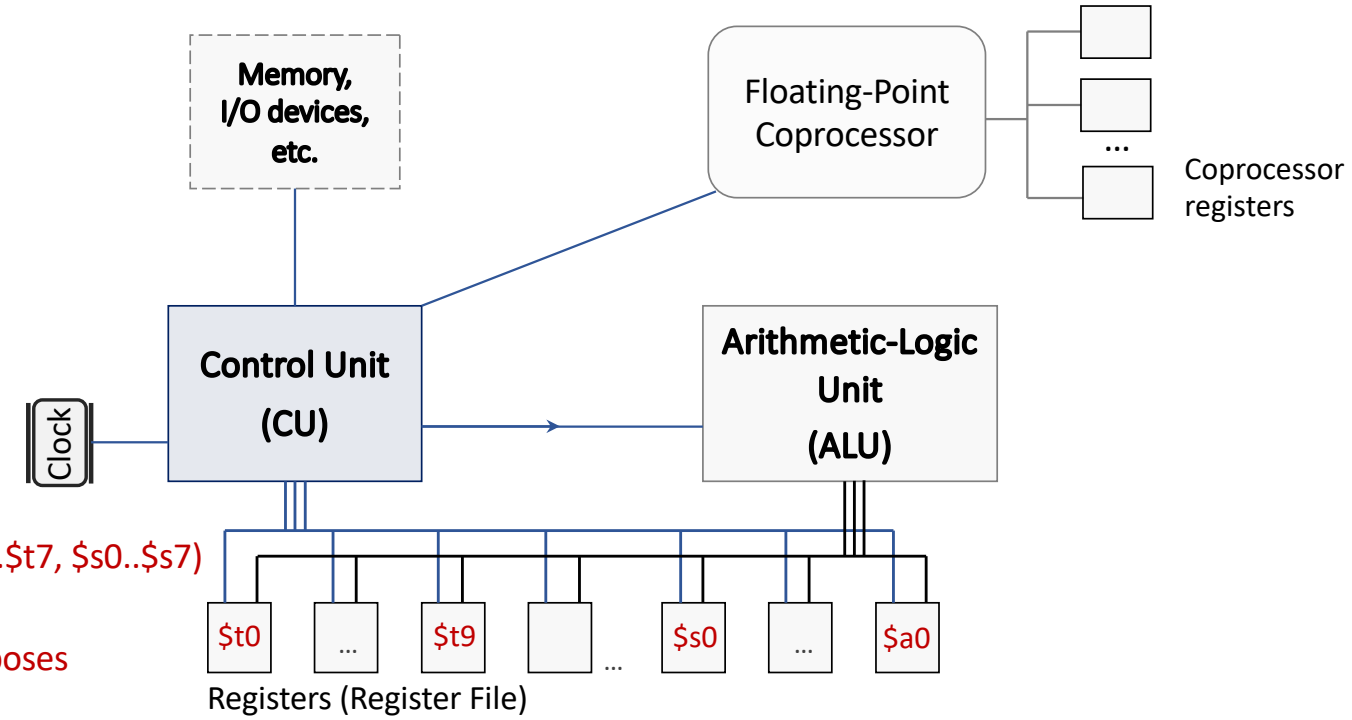
MIPS Processor Architecture

MIPS is a RISC architecture



MIPS Processor Architecture

MIPS is a RISC architecture

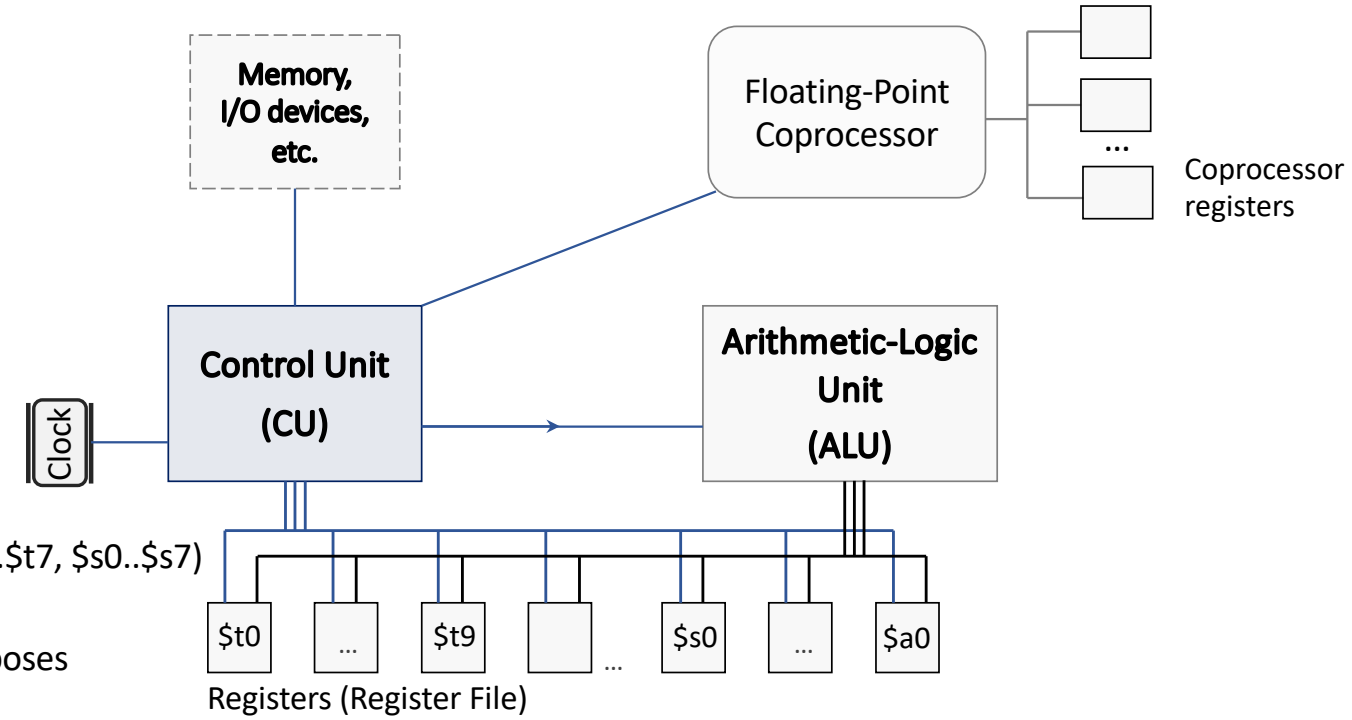


32 general-purpose registers:

- Have their names (like $t0..$t7, $s0..$s7)
- Directly addressable
- Reserved for different purposes

MIPS Processor Architecture

MIPS is a RISC architecture



32 general-purpose registers:

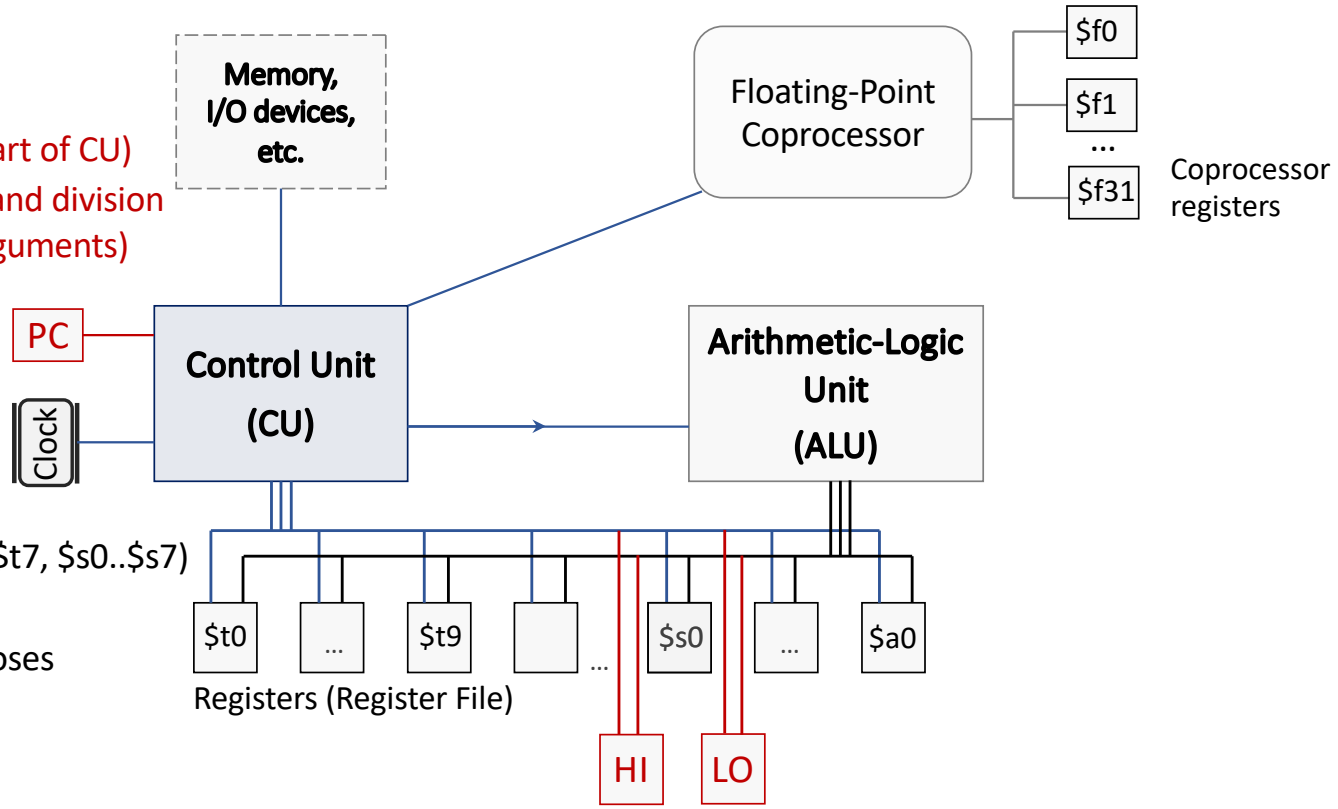
- Have their names (like $t0..$t7, $s0..$s7)
- Directly addressable
- Reserved for different purposes

"Register spilling":
If the number of live variables exceeds the number of available registers, then the compiler spills some variables from registers into memory

MIPS Processor Architecture

MIPS is a RISC architecture

3 special-purpose registers:
- PC – program counter (a part of CU)
- HI, LO – for multiplication and division instructions (for integer arguments)



32 general-purpose registers:
- Have their names (like $t0..$t7, $s0..$s7)
- Directly addressable
- Reserved for different purposes

Data is retrived from these registers by using special functions mfhi and mflo

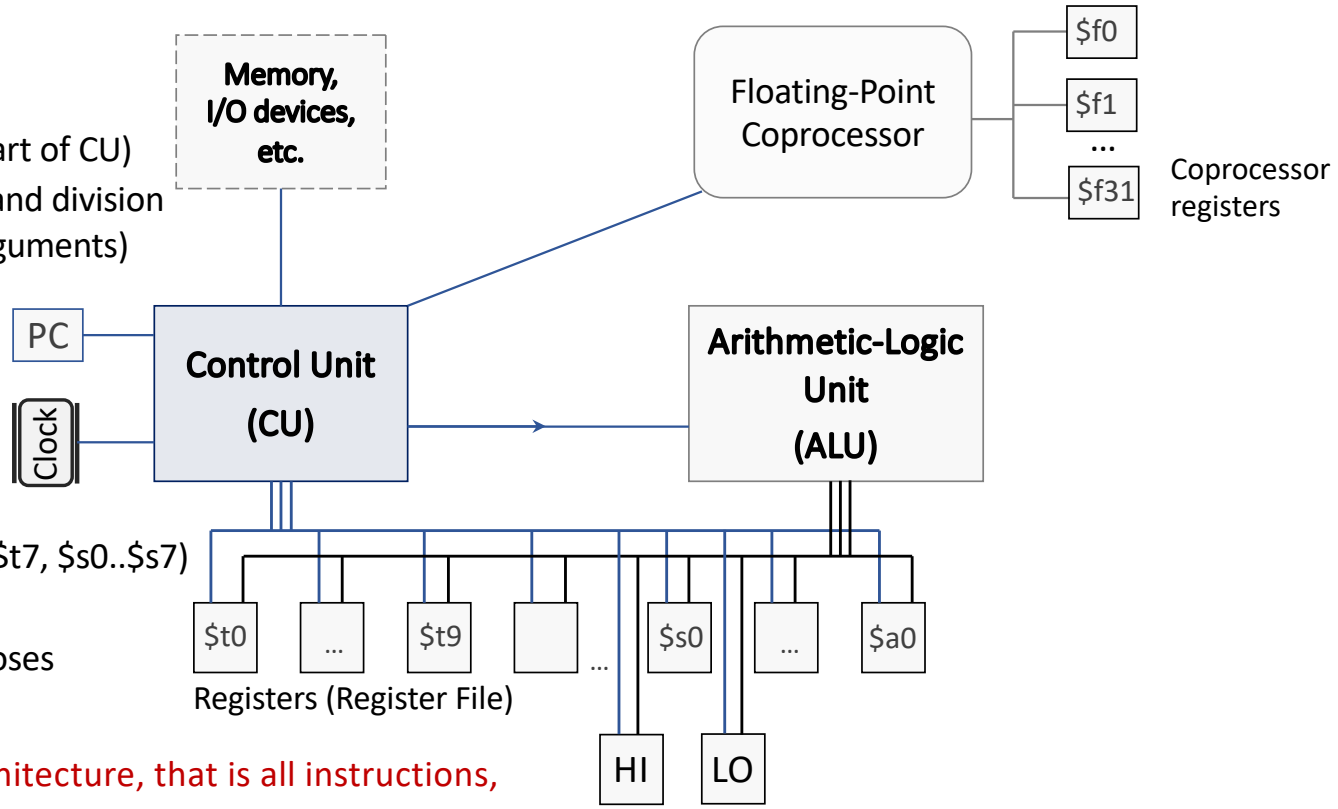
"Register spilling":
If the number of live variables exceeds the number of available registers, then the compiler spills some variables from registers into memory

MIPS Processor Architecture

MIPS is a RISC architecture

32 registers for floating-point instructions:
- Named by $f0..$f31
- Directly addressable
- Reserved for different purposes

3 special-purpose registers:
- PC – program counter (a part of CU)
- HI, LO – for multiplication and division instructions (for integer arguments)



32 general-purpose registers:
- Have their names (like $t0..$t7, $s0..$s7)
- Directly addressable
- Reserved for different purposes

MIPS is a load/store architecture, that is all instructions, except for memory access, operate on registers

Data is retrived from these registers by using special functions mfhi and mflo

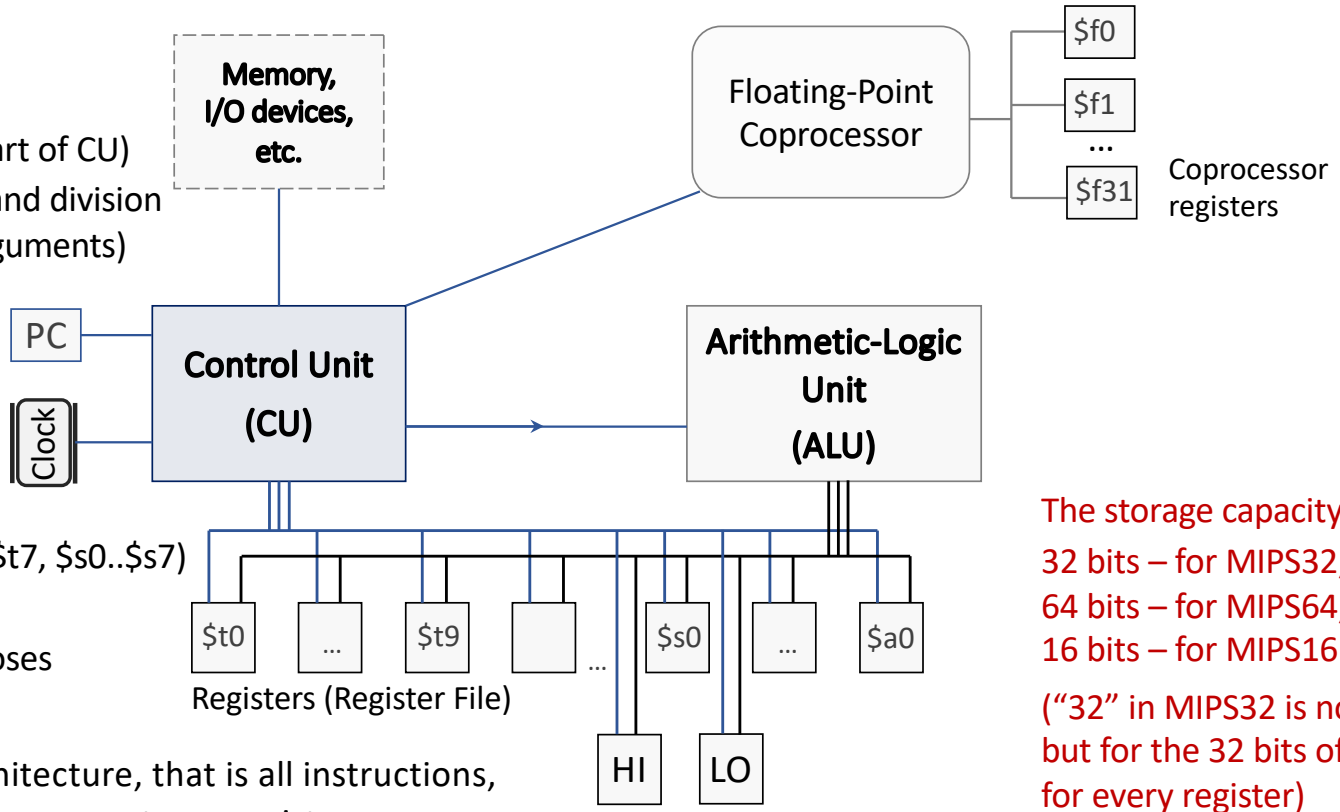
"Register spilling":
If the number of live variables exceeds the number of available registers, then the compiler spills some variables from registers into memory

MIPS Processor Architecture

MIPS is a RISC architecture

32 registers for floating-point instructions:
- Named by $f0..$f31
- Directly addressable
- Reserved for different purposes

3 special-purpose registers:
- PC – program counter (a part of CU)
- HI, LO – for multiplication and division instructions (for integer arguments)



32 general-purpose registers:
- Have their names (like $t0..$t7, $s0..$s7)
- Directly addressable
- Reserved for different purposes

The storage capacity of a register varies:
32 bits – for MIPS32;
64 bits – for MIPS64;
16 bits – for MIPS16

("32" in MIPS32 is not for 32 registers, but for the 32 bits of storage capacity for every register)

MIPS is a load/store architecture, that is all instructions, except for memory access, operate on registers

Data is retrived from these registers by using special functions mfhi and mflo

"Register spilling":
If the number of live variables exceeds the number of available registers, then the compiler spills some variables from registers into memory

32 Directly Addressable MIPS Registers

Reg. Num	Reg. Name	Reg. Purpose

32 Directly Addressable MIPS Registers

Reg. Num	Reg. Name	Reg. Purpose
$0	**$zero**	Hardwired zero (0x00000000)

32 Directly Addressable MIPS Registers

Reg. Num	Reg. Name	Reg. Purpose
$0	**$zero**	Hardwired zero (0x00000000)
$1	**$at**	Assembler temporary

32 Directly Addressable MIPS Registers

Reg. Num	Reg. Name	Reg. Purpose
$0	**$zero**	Hardwired zero (0x00000000)
$1	**$at**	Assembler temporary
$2-$3	**$v0-$v1**	Codes of system calls; return values of system calls

32 Directly Addressable MIPS Registers

Reg. Num	Reg. Name	Reg. Purpose
$0	**$zero**	Hardwired zero (0x00000000)
$1	**$at**	Assembler temporary
$2-$3	**$v0-$v1**	Codes of system calls; return values of system calls
$4-$7	**$a0-$a3**	Arguments for system calls

32 Directly Addressable MIPS Registers

Reg. Num	Reg. Name	Reg. Purpose
$0	**$zero**	Hardwired zero (0x00000000)
$1	**$at**	Assembler temporary
$2-$3	**$v0-$v1**	Codes of system calls; return values of system calls
$4-$7	**$a0-$a3**	Arguments for system calls
$8-$15	**$t0-$t7**	Registers for temporary values

32 Directly Addressable MIPS Registers

Reg. Num	Reg. Name	Reg. Purpose
$0	**$zero**	Hardwired zero (0x00000000)
$1	**$at**	Assembler temporary
$2-$3	**$v0-$v1**	Codes of system calls; return values of system calls
$4-$7	**$a0-$a3**	Arguments for system calls
$8-$15	**$t0-$t7**	Registers for temporary values
$16-$23	**$s0-$s7**	Registers for variables

32 Directly Addressable MIPS Registers

Reg. Num	Reg. Name	Reg. Purpose
$0	**$zero**	Hardwired zero (0x00000000)
$1	**$at**	Assembler temporary
$2-$3	**$v0-$v1**	Codes of system calls; return values of system calls
$4-$7	**$a0-$a3**	Arguments for system calls
$8-$15	**$t0-$t7**	Registers for temporary values
$16-$23	**$s0-$s7**	Registers for variables
$24-$25	**$t8-$t9**	Additional registers for temporary values

32 Directly Addressable MIPS Registers

Reg. Num	Reg. Name	Reg. Purpose
$0	**$zero**	Hardwired zero (0x00000000)
$1	**$at**	Assembler temporary
$2-$3	**$v0-$v1**	Codes of system calls; return values of system calls
$4-$7	**$a0-$a3**	Arguments for system calls
$8-$15	**$t0-$t7**	Registers for temporary values
$16-$23	**$s0-$s7**	Registers for variables
$24-$25	**$t8-$t9**	Additional registers for temporary values
$26-$27	**$k0-$k1**	Registers reserved for OS kernel

32 Directly Addressable MIPS Registers

Reg. Num	Reg. Name	Reg. Purpose
$0	**$zero**	Hardwired zero (0x00000000)
$1	**$at**	Assembler temporary
$2-$3	**$v0-$v1**	Codes of system calls; return values of system calls
$4-$7	**$a0-$a3**	Arguments for system calls
$8-$15	**$t0-$t7**	Registers for temporary values
$16-$23	**$s0-$s7**	Registers for variables
$24-$25	**$t8-$t9**	Additional registers for temporary values
$26-$27	**$k0-$k1**	Registers reserved for OS kernel
$28	**$gp**	Global pointer (to the next program instruction to be executed)

32 Directly Addressable MIPS Registers

Reg. Num	Reg. Name	Reg. Purpose
$0	**$zero**	Hardwired zero (0x00000000)
$1	**$at**	Assembler temporary
$2-$3	**$v0-$v1**	Codes of system calls; return values of system calls
$4-$7	**$a0-$a3**	Arguments for system calls
$8-$15	**$t0-$t7**	Registers for temporary values
$16-$23	**$s0-$s7**	Registers for variables
$24-$25	**$t8-$t9**	Additional registers for temporary values
$26-$27	**$k0-$k1**	Registers reserved for OS kernel
$28	**$gp**	Global pointer (to the next program instruction to be executed)
$29	**$sp**	Stack pointer

32 Directly Addressable MIPS Registers

Reg. Num	Reg. Name	Reg. Purpose
$0	**$zero**	Hardwired zero (0x00000000)
$1	**$at**	Assembler temporary
$2-$3	**$v0-$v1**	Codes of system calls; return values of system calls
$4-$7	**$a0-$a3**	Arguments for system calls
$8-$15	**$t0-$t7**	Registers for temporary values
$16-$23	**$s0-$s7**	Registers for variables
$24-$25	**$t8-$t9**	Additional registers for temporary values
$26-$27	**$k0-$k1**	Registers reserved for OS kernel
$28	**$gp**	Global pointer (to the next program instruction to be executed)
$29	**$sp**	Stack pointer
$30	**$fp**	Frame pointer

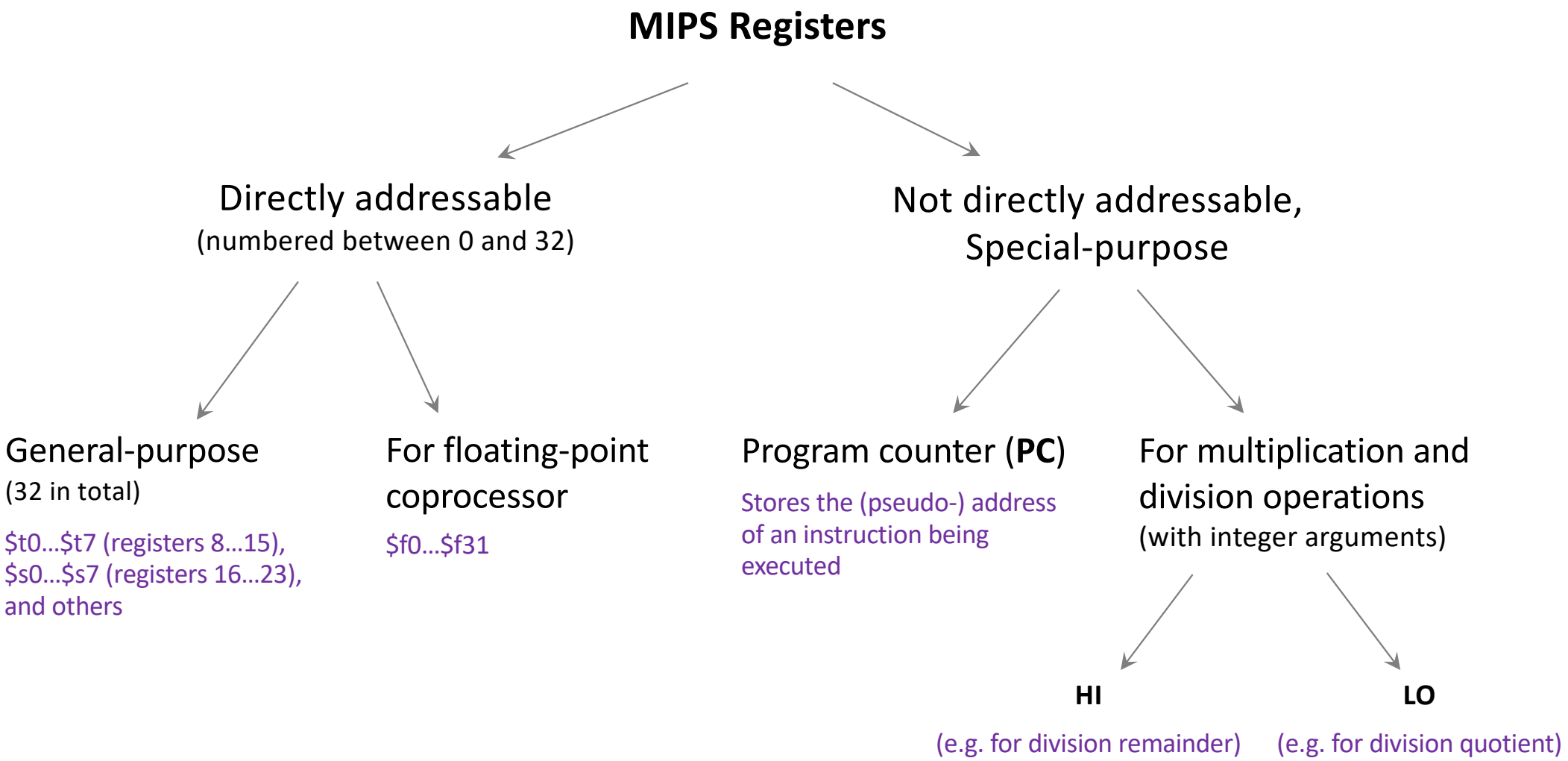
32 Directly Addressable MIPS Registers

Reg. Num	Reg. Name	Reg. Purpose
$0	**$zero**	Hardwired zero (0x00000000)
$1	**$at**	Assembler temporary
$2-$3	**$v0-$v1**	Codes of system calls; return values of system calls
$4-$7	**$a0-$a3**	Arguments for system calls
$8-$15	**$t0-$t7**	Registers for temporary values
$16-$23	**$s0-$s7**	Registers for variables
$24-$25	**$t8-$t9**	Additional registers for temporary values
$26-$27	**$k0-$k1**	Registers reserved for OS kernel
$28	**$gp**	Global pointer (to the next program instruction to be executed)
$29	**$sp**	Stack pointer
$30	**$fp**	Frame pointer
$31	**$ra**	Return address

32 Directly Addressable MIPS Registers

Reg. Num	Reg. Name	Reg. Purpose
$0	**$zero**	Hardwired zero (0x00000000)
$1	**$at**	Assembler temporary
$2-$3	**$v0-$v1**	Codes of system calls; return values of system calls
$4-$7	**$a0-$a3**	Arguments for system calls
$8-$15	**$t0-$t7**	Registers for temporary values
$16-$23	**$s0-$s7**	Registers for variables
$24-$25	**$t8-$t9**	Additional registers for temporary values
$26-$27	**$k0-$k1**	Registers reserved for OS kernel
$28	**$gp**	Global pointer (to the next program instruction to be executed)
$29	**$sp**	Stack pointer
$30	**$fp**	Frame pointer
$31	**$ra**	Return address

To provide flow control in a program


MIPS Registers
Directly addressable
(numbered between 0 and 32)
Not directly addressable,
Special-purpose
General-purpose
(32 in total)
$t0…$t7 (registers 8…15),
$s0…$s7 (registers 16…23),
and others
For floating-point
coprocessor
$f0…$f31
Program counter (PC)
Stores the (pseudo-) address
of an instruction being
executed
For multiplication and
division operations
(with integer arguments)
HI
(e.g. for division remainder)
LO
(e.g. for division quotient)

Sample MIPS Assembly Program: Addition of Two Numbers

Sample MIPS Assembly Program: Addition of Two Numbers

```
1   li  $t1, 5          # load value “5” into register $t1
```

li – “load immediate”, to load a constant value (number) into some register

Sample MIPS Assembly Program: Addition of Two Numbers

```
1   li  $t1, 5          # load value “5” into register $t1
```

li – “load immediate”, to load a constant value (number) into some register

la – “load argument”, to load a constant string into some register

Sample MIPS Assembly Program: Addition of Two Numbers

```
1   li  $t1, 5          # load value “5” into register $t1
2   li  $t2, 7          # load value “7” into register $t1
```

Sample MIPS Assembly Program: Addition of Two Numbers

```
1   li  $t1, 5      # load value “5” into register $t1
2   li  $t2, 7      # load value “7” into register $t1
```

Some available MIPS arithmetic and logic instructions:

Arithmetic	add	**add $t0, $t1, $t2**	$t0 = $t1 + $t2
	subtract	**sub $t0, $t1, $t2**	$t0 = $t1 - $t2
	add immediate	**addi $t0, $t1, 26**	$t0 = $t1 - 26
Logical (bitwise, that is bit by bit)	and	**and $t0, $t1, $t2**	$t0 = $t1 & $t2
	or	**or $t0, $t1, $t2**	$t0 = $t1 | $t2
	nor	**nor $t0, $t1, $t2**	$t0 = ~($t1 | $t2)
	and immediate	**andi $t0, $t1, 17**	$t0 = $t1 & 17
	or immediate	**ori $t0, $t1, 13**	$t0 = $t1 | 13
	shift left logical	**sll $t0, $t1, 4**	$t0 = $t1 << 4
	shift right logical	**srl $t0, $t1, 6**	$t0 = $t1 >> 6

Sample MIPS Assembly Program: Addition of Two Numbers

```
1   li  $t1, 5          # load value “5” into register $t1
2   li  $t2, 7          # load value “7” into register $t1
```

Some available MIPS arithmetic and logic instructions:

Arithmetic	add	**add $t0, $t1, $t2**	$t0 = $t1 + $t2
	subtract	**sub $t0, $t1, $t2**	$t0 = $t1 - $t2
	add immediate	**addi $t0, $t1, 26**	$t0 = $t1 - 26
Logical (bitwise, that is bit by bit)	and	**and $t0, $t1, $t2**	$t0 = $t1 & $t2
	or	**or $t0, $t1, $t2**	$t0 = $t1 | $t2
	nor	**nor $t0, $t1, $t2**	$t0 = ~($t1 | $t2)
	and immediate	**andi $t0, $t1, 17**	$t0 = $t1 & 17
	or immediate	**ori $t0, $t1, 13**	$t0 = $t1 | 13
	shift left logical	**sll $t0, $t1, 4**	$t0 = $t1 << 4
	shift right logical	**srl $t0, $t1, 6**	$t0 = $t1 >> 6

Sample MIPS Assembly Program: Addition of Two Numbers

```
1   li  $t1, 5          # load value "5" into register $t1
2   li  $t2, 7          # load value "7" into register $t1
3   add  $t0, $t1, $t2  # $t0 = $t1 + $t2
```

Some available MIPS arithmetic and logic instructions:

Arithmetic	add	**add $t0, $t1, $t2**	$t0 = $t1 + $t2
	subtract	**sub $t0, $t1, $t2**	$t0 = $t1 - $t2
	add immediate	**addi $t0, $t1, 26**	$t0 = $t1 - 26
Logical (bitwise, that is bit by bit)	and	**and $t0, $t1, $t2**	$t0 = $t1 & $t2
	or	**or $t0, $t1, $t2**	$t0 = $t1 | $t2
	nor	**nor $t0, $t1, $t2**	$t0 = ~($t1 | $t2)
	and immediate	**andi $t0, $t1, 17**	$t0 = $t1 & 17
	or immediate	**ori $t0, $t1, 13**	$t0 = $t1 | 13
	shift left logical	**sll $t0, $t1, 4**	$t0 = $t1 << 4
	shift right logical	**srl $t0, $t1, 6**	$t0 = $t1 >> 6

Sample MIPS Assembly Program: Addition of Two Numbers

1	**li** $t1, 5	# load value “5” into register $t1
2	**li** $t2, 7	# load value “7” into register $t1
3	**add** $t0, $t1, $t2	# $t0 = $t1 + $t2

Next we need to show the computed result to a user. How?

Sample MIPS Assembly Program: Addition of Two Numbers

```
1   li  $t1, 5          # load value “5” into register $t1
2   li  $t2, 7          # load value “7” into register $t1
3   add  $t0, $t1, $t2  # $t0 = $t1 + $t2
```

Next we need to show the computed result to a user. How?

System call – the mechanism to interact between the program and the Operating System (e.g. for I/O operations)

Sample MIPS Assembly Program: Addition of Two Numbers

```
1   li  $t1, 5          # load value “5” into register $t1
2   li  $t2, 7          # load value “7” into register $t1
3   add  $t0, $t1, $t2  # $t0 = $t1 + $t2
```

System call – the mechanism to interact between the program and the Operating System (e.g. for I/O operations)

Various types of system calls (or “syscalls”) are available

Sample MIPS Assembly Program: Addition of Two Numbers

```
1   li  $t1, 5          # load value "5" into register $t1
2   li  $t2, 7          # load value "7" into register $t1
3   add  $t0, $t1, $t2  # $t0 = $t1 + $t2
```

System call – the mechanism to interact between the program and the Operating System (e.g. for I/O operations)

Various types of system calls (or "syscalls") are available

Code in $v0		
1		
2		
3		
4		
5		
8		
10		

Sample MIPS Assembly Program: Addition of Two Numbers

```
1   li  $t1, 5          # load value "5" into register $t1
2   li  $t2, 7          # load value "7" into register $t1
3   add  $t0, $t1, $t2  # $t0 = $t1 + $t2
```

System call – the mechanism to interact between the program and the Operating System (e.g. for I/O operations)

Various types of system calls (or "syscalls") are available

Code in $v0	Purpose	Arguments/Return values
1	Print integer	$a0 contains integer value to be printed
2		
3		
4		
5		
8		
10		

Sample MIPS Assembly Program: Addition of Two Numbers

```
1  li  $t1, 5          # load value "5" into register $t1
2  li  $t2, 7          # load value "7" into register $t1
3  add  $t0, $t1, $t2  # $t0 = $t1 + $t2
```

System call – the mechanism to interact between the program and the Operating System (e.g. for I/O operations)

Various types of system calls (or "syscalls") are available

Code in $v0	Purpose	Arguments/Return values
1	Print integer	$a0 contains integer value to be printed
2	Print float	$f12 contains float value to be printed
3	Print double	$f12 contains double value to be printed
4	Print string	$a0 contains the memory address of a null terminated string, and $a1 – a string size
5		
8		
10		

Sample MIPS Assembly Program: Addition of Two Numbers

```
1  li  $t1, 5          # load value "5" into register $t1
2  li  $t2, 7          # load value "7" into register $t1
3  add  $t0, $t1, $t2  # $t0 = $t1 + $t2
```

System call – the mechanism to interact between the program and the Operating System (e.g. for I/O operations)

Various types of system calls (or "syscalls") are available

Code in $v0	Purpose	Arguments/Return values
1	Print integer	$a0 contains value to be printed
2	Print float	$f12 contains float value to be printed
3	Print double	$f12 contains double value to be printed
4	Print string	$a0 contains the memory address of a null terminated string, and $a1 – a string size
5	Read integer	Value that was read is stored into $v0 register
8		
10		

Sample MIPS Assembly Program: Addition of Two Numbers

```
1  li  $t1, 5          # load value "5" into register $t1
2  li  $t2, 7          # load value "7" into register $t1
3  add  $t0, $t1, $t2  # $t0 = $t1 + $t2
```

System call – the mechanism to interact between the program and the Operating System (e.g. for I/O operations)

Various types of system calls (or "syscalls") are available

Code in $v0	Purpose	Arguments/Return values
1	Print integer	$a0 contains value to be printed
2	Print float	$f12 contains float value to be printed
3	Print double	$f12 contains double value to be printed
4	Print string	$a0 contains the memory address of a null terminated string, and $a1 – a string size
5	Read integer	Value that was read is stored into $v0 register
8	Read string	$a0 contains the address of a string that was read, and $a1 – the num. of chars to read
10		

Sample MIPS Assembly Program: Addition of Two Numbers

```
1  li  $t1, 5          # load value "5" into register $t1
2  li  $t2, 7          # load value "7" into register $t1
3  add  $t0, $t1, $t2  # $t0 = $t1 + $t2
```

System call – the mechanism to interact between the program and the Operating System (e.g. for I/O operations)

Various types of system calls (or "syscalls") are available

Code in $v0	Purpose	Arguments/Return values
1	Print integer	$a0 contains value to be printed
2	Print float	$f12 contains float value to be printed
3	Print double	$f12 contains double value to be printed
4	Print string	$a0 contains the memory address of a null terminated string, and $a1 – a string size
5	Read integer	Value that was read is stored into $v0 register
8	Read string	$a0 contains the address of a string that was read, and $a1 – the num. of chars to read
10	Exit program/function	none

Sample MIPS Assembly Program: Addition of Two Numbers

```
1   li  $t1, 5          # load value "5" into register $t1
2   li  $t2, 7          # load value "7" into register $t1
3   add  $t0, $t1, $t2  # $t0 = $t1 + $t2
```

System call – the mechanism to interact between the program and the Operating System (e.g. for I/O operations)

Various types of system calls (or "syscalls") are available

Code in $v0	Purpose	Arguments/Return values
1	Print integer	$a0 contains value to be printed
2	Print float	$f12 contains float value to be printed
3	Print double	$f12 contains double value to be printed
4	Print string	$a0 contains the memory address of a null terminated string, and $a1 – a string size
5	Read integer	Value that was read is stored into $v0 register
8	Read string	$a0 contains the address of a string that was read, and $a1 – the num. of chars to read
10	Exit program/function	none

Sample MIPS Assembly Program: Addition of Two Numbers

```
1   li  $t1, 5          # load value “5” into register $t1
2   li  $t2, 7          # load value “7” into register $t1
3   add  $t0, $t1, $t2  # $t0 = $t1 + $t2
```

Code in $v0	Purpose	Arguments/Return values
1	Print integer	$a0 contains value to be printed

Sample MIPS Assembly Program: Addition of Two Numbers

```
1   li  $t1, 5          # load value "5" into register $t1
2   li  $t2, 7          # load value "7" into register $t1
3   add  $t0, $t1, $t2  # $t0 = $t1 + $t2
4   move  $a0, $t0      # move value from register $t0 to $a0
```

Code in $v0	Purpose	Arguments/Return values
1	Print integer	$a0 contains value to be printed

Sample MIPS Assembly Program: Addition of Two Numbers

```
1  li  $t1, 5          # load value "5" into register $t1
2  li  $t2, 7          # load value "7" into register $t1
3  add  $t0, $t1, $t2  # $t0 = $t1 + $t2
4  move  $a0, $t0      # move value from register $t0 to $a0
5  li  $v0, 1          # set code for syscall to "1" (to print)
```

Code in $v0	Purpose	Arguments/Return values
1	Print integer	$a0 contains value to be printed

Sample MIPS Assembly Program: Addition of Two Numbers

```
1  li  $t1, 5              # load value "5" into register $t1
2  li  $t2, 7              # load value "7" into register $t1
3  add  $t0, $t1, $t2      # $t0 = $t1 + $t2
4  move  $a0, $t0          # move value from register $t0 to $a0
5  li  $v0, 1              # set code for syscall to "1" (to print)
6  syscall                 # execute syscall with code in $v0 and argument in $a0
```

Code in $v0	Purpose	Arguments/Return values
1	Print integer	$a0 contains value to be printed

Sample MIPS Assembly Program: Addition of Two Numbers

```
1  li  $t1, 5              # load value “5” into register $t1
2  li  $t2, 7              # load value “7” into register $t1
3  add  $t0, $t1, $t2      # $t0 = $t1 + $t2
4  move  $a0, $t0          # move value from register $t0 to $a0
5  li  $v0, 1              # set code for syscall to “1” (to print)
6  syscall                 # execute syscall with code in $v0 and argument in $a0
```

We are ready to exit the program

Sample MIPS Assembly Program: Addition of Two Numbers

```
1  li  $t1, 5              # load value “5” into register $t1
2  li  $t2, 7              # load value “7” into register $t1
3  add  $t0, $t1, $t2      # $t0 = $t1 + $t2
4  move  $a0, $t0          # move value from register $t0 to $a0
5  li  $v0, 1              # set code for syscall to “1” (to print)
6  syscall                 # execute syscall with code in $v0 and argument in $a0
```

We are ready to exit the program

Code in $v0	Purpose	Arguments/Return values
10	Exit program	None

Sample MIPS Assembly Program: Addition of Two Numbers

```
1  li  $t1, 5              # load value "5" into register $t1
2  li  $t2, 7              # load value "7" into register $t1
3  add  $t0, $t1, $t2      # $t0 = $t1 + $t2
4  move  $a0, $t0          # move value from register $t0 to $a0
5  li  $v0, 1              # set code for syscall to "1" (to print)
6  syscall                 # execute syscall with code in $v0 and argument in $a0
7  li  $v0, 10             # set code for syscall to "10" (exit)
```

Code in $v0	Purpose	Arguments/Return values
10	Exit program	None

Sample MIPS Assembly Program: Addition of Two Numbers

```
1  li  $t1, 5              # load value “5” into register $t1
2  li  $t2, 7              # load value “7” into register $t1
3  add  $t0, $t1, $t2      # $t0 = $t1 + $t2
4  move  $a0, $t0          # move value from register $t0 to $a0
5  li  $v0, 1              # set code for syscall to “1” (to print)
6  syscall                 # execute syscall with code in $v0 and argument in $a0
7  li  $v0, 10             # set code for syscall to “10” (exit)
8  syscall                 # execute syscall with code “10”
```

Another Example: Reading and Printing a String

```
1   . data
```

Another Example: Reading and Printing a String

```
1   . data
```

Possible MIPS assembly directives:
.data
.text
.globl

Another Example: Reading and Printing a String

```
1   . data
2       msg:        .asciiz “Enter your string: ”        # message asking to input a string
```

Another Example: Reading and Printing a String

```
1   . data
2       msg:        .asciiz “Enter your string: ”       # message asking to input a string
4       inputStr:   .space 10                           # array of 10 bytes, to store input string
```

Another Example: Reading and Printing a String

```
1   . data
2       msg:        .asciiz “Enter your string: ”       # message asking to input a string
4       inputStr:   .space 10                           # array of 10 bytes, to store input string
5   main:
```

Another Example: Reading and Printing a String

```
1   . data
2       msg:        .asciiz “Enter your string: ”        # message asking to input a string
4       inputStr:   .space 10                            # array of 10 bytes, to store input string
5   main:
6       li $v0, 4                    # syscall code to print message asking to input a string
```

Codes of system calls:

Code in $v0	Purpose	Arguments/Return values
1	Print integer	$a0 contains value to be printed
2	Print float	$f12 contains float value to be printed
3	Print double	$f12 contains double value to be printed
4	Print string	$a0 contains the memory address of a null terminated string, and $a1 – a string size
5	Read integer	Value that was read is stored into $v0 register
8	Read string	$a0 contains the address of a string that was read, and $a1 – the num. of chars to read
10	Exit program/function	none

Another Example: Reading and Printing a String

```
1   . data
2       msg:        .asciiz “Enter your string: ”      # message asking to input a string
4       inputStr:   .space 10                          # array of 10 bytes, to store input string
5   main:
6       li $v0, 4                  # syscall code to print message asking to input a string
7       la $a0, msg1               # load memory address of the beginning of a string
8       syscall                    # syscall invocation, to print message
```

Codes of system calls:

Code in $v0	Purpose	Arguments/Return values
1	Print integer	$a0 contains value to be printed
2	Print float	$f12 contains float value to be printed
3	Print double	$f12 contains double value to be printed
4	Print string	$a0 contains the memory address of a null terminated string, and $a1 – a string size
5	Read integer	Value that was read is stored into $v0 register
8	Read string	$a0 contains the address of a string that was read, and $a1 – the num. of chars to read
10	Exit program/function	none

Another Example: Reading and Printing a String

```
1   . data
2       msg:        .asciiz “Enter your string: ”       # message asking to input a string
4       inputStr:   .space 10                           # array of 10 bytes, to store input string
5   main:
6       li $v0, 4                   # syscall code to print message asking to input a string
7       la $a0, msg1                # load memory address of the beginning of a string
8       syscall                     # syscall invocation, to print message
9       li $v0, 8                   # syscall code to read string from user
```

Code in $v0	Purpose	Arguments/Return values
1	Print integer	$a0 contains value to be printed
2	Print float	$f12 contains float value to be printed
3	Print double	$f12 contains double value to be printed
4	Print string	$a0 contains the memory address of a null terminated string, and $a1 – a string size
5	Read integer	Value that was read is stored into $v0 register
8	Read string	$a0 contains the address of a string that was read, and $a1 – the num. of chars to read
10	Exit program/function	none

Another Example: Reading and Printing a String

```
1   . data
2       msg:        .asciiz “Enter your string: ”        # message asking to input a string
4       inputStr:   .space 10                            # array of 10 bytes, to store input string
5   main:
6       li $v0, 4                   # syscall code to print message asking to input a string
7       la $a0, msg1                # load memory address of the beginning of a string
8       syscall                     # syscall invocation, to print message
9       li $v0, 8                   # syscall code to read string from user
10      la $a0, inputStr            # memory address, where to start writing an input string
```

Another Example: Reading and Printing a String

```
1   . data
2       msg:        .asciiz “Enter your string: ”        # message asking to input a string
4       inputStr:   .space 10                            # array of 10 bytes, to store input string
5   main:
6       li $v0, 4                   # syscall code to print message asking to input a string
7       la $a0, msg1                # load memory address of the beginning of a string
8       syscall                     # syscall invocation, to print message
9       li $v0, 8                   # syscall code to read string from user
10      la $a0, inputStr            # memory address, where to start writing an input string
11      li $a1, 10                  # the maximum size of an input string
12      syscall                     # syscall to read string; $v0 now “contains” a user string
```

Another Example: Reading and Printing a String

```
1   . data
2       msg:        .asciiz “Enter your string: ”       # message asking to input a string
4       inputStr:   .space 10                           # array of 10 bytes, to store input string
5   main:
6       li $v0, 4                   # syscall code to print message asking to input a string
7       la $a0, msg1                # load memory address of the beginning of a string
8       syscall                     # syscall invocation, to print message
9       li $v0, 8                   # syscall code to read string from user
10      la $a0, inputStr            # memory address, where to start writing an input string
11      li $a1, 10                  # the maximum size of an input string
12      syscall                     # syscall to read string; $v0 now “contains” a user string
13      move $a0, $v0               # move string address from $v0 to $a0
14      li $v0, 4                   # syscall code to print a string
15      li $a1, 10                  # the size of a string to be printed
16      syscall                     # printing string
```

Another Example: Reading and Printing a String

```
1   . data
2       msg:        .asciiz “Enter your string: ”       # message asking to input a string
4       inputStr:   .space 10                           # array of 10 bytes, to store input string
5   main:
6       li $v0, 4                   # syscall code to print message asking to input a string
7       la $a0, msg1                # load memory address of the beginning of a string
8       syscall                     # syscall invocation, to print message
9       li $v0, 8                   # syscall code to read string from user
10      la $a0, inputStr            # memory address, where to start writing an input string
11      li $a1, 10                  # the maximum size of an input string
12      syscall                     # syscall to read string; $v0 now “contains” a user string
13      move $a0, $v0               # move string address from $v0 to $a0
14      li $v0, 4                   # syscall code to print a string
15      li $a1, 10                  # the size of a string to be printed
16      syscall                     # printing string
17      li $v0, 10                  # syscall code to terminate the program
18      syscall                     # termination
```