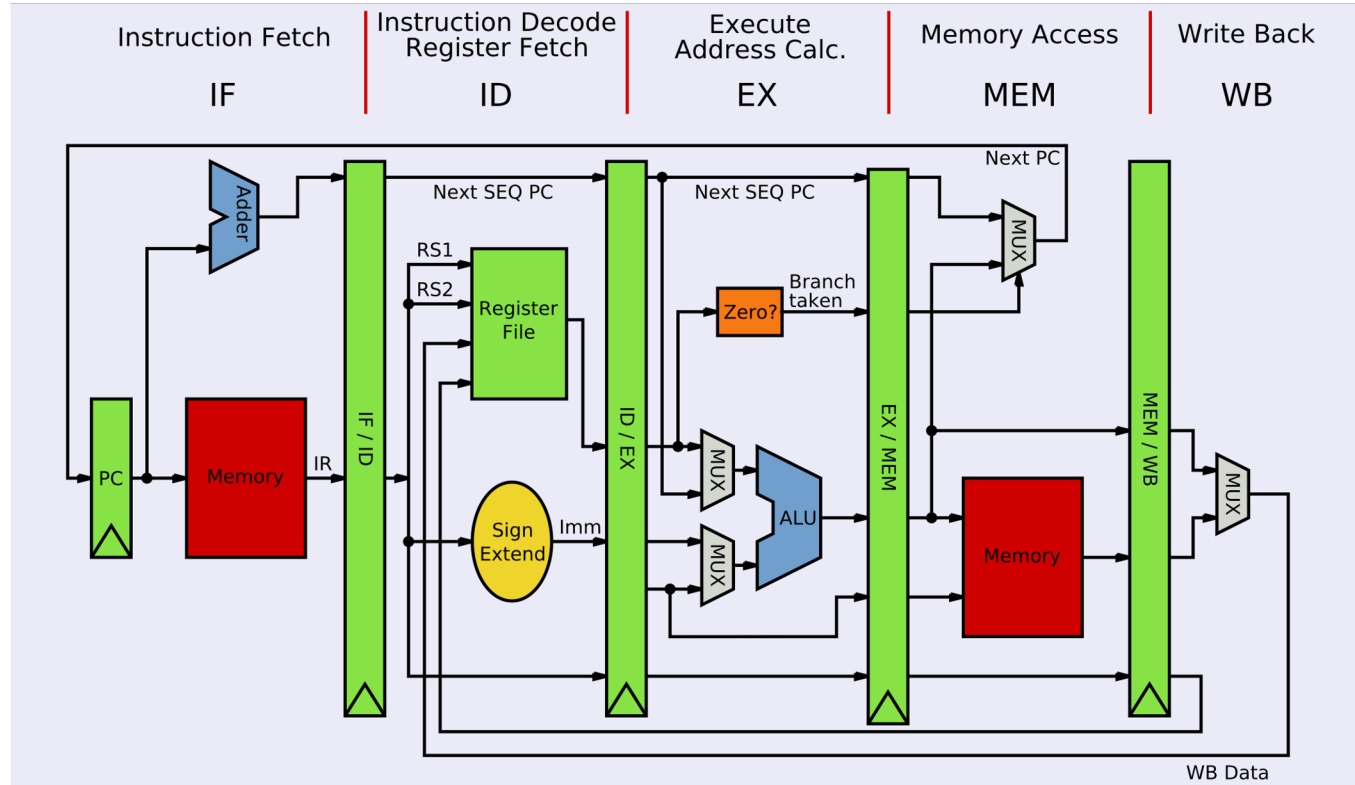


MIPS Microarchitecture



Instruction Set Architectures

- The ISA is one level above the physical architecture
- Defines the following:
 - Supported instruction and their semantics
 - Supported data types
 - Registers: size, number, and purpose
 - Memory: layout, addressing, alignment, endianness
 - Memory is an array of bytes
 - OS interface:
- Does not define the following:
 - technology used
 - chip layout
 - memory implementation
 - etc
- Very Similar to an API:
- RISC versus CISC
 - Reduced Instruction Set Computer (RISC)
 - Complex Instruction Set Computer (CISC)

endianness: the order of bytes within a word

- big: 1,2,3,4 (yy/mm/dd)
- little: 4,3,2,1 (dd/mm/yy)
- middle: 3,4,1,2 (mm/dd/yy)

RISC Examples

ARM

MIPS

CISC Examples

AMD

x86

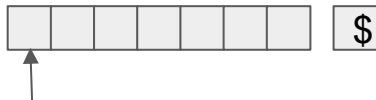
INTEL

x86

Universal Computer

- $TM(Q, \Sigma, \Gamma, \delta, q_0)$
- Tape: sufficiently large
- A specialized control unit (aka firmware)
- A specialized program placed on tape
- A generic program placed on tape
- Input coming from an I/O device

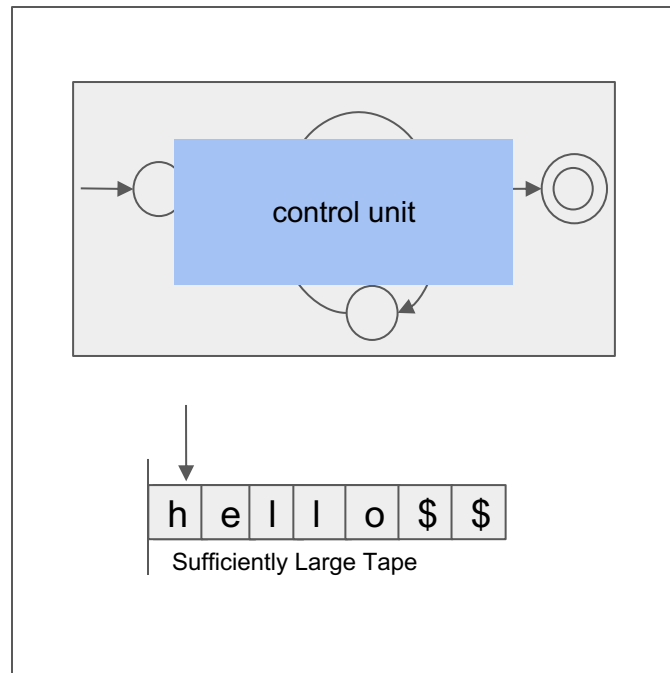
input string:



memory

The OS

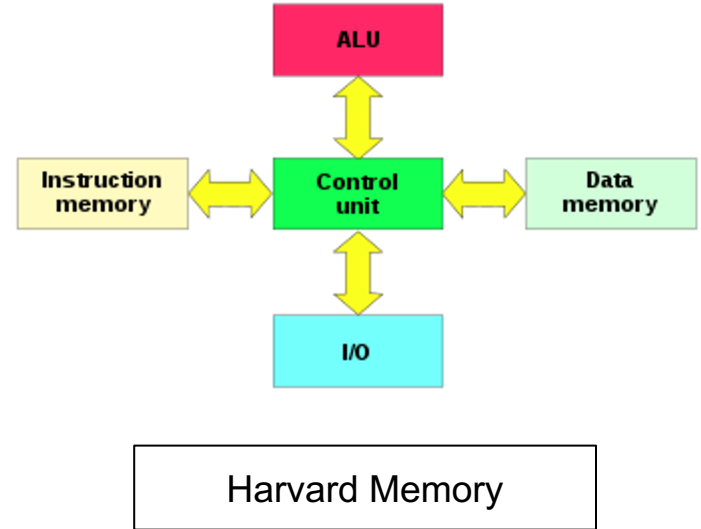
Your program



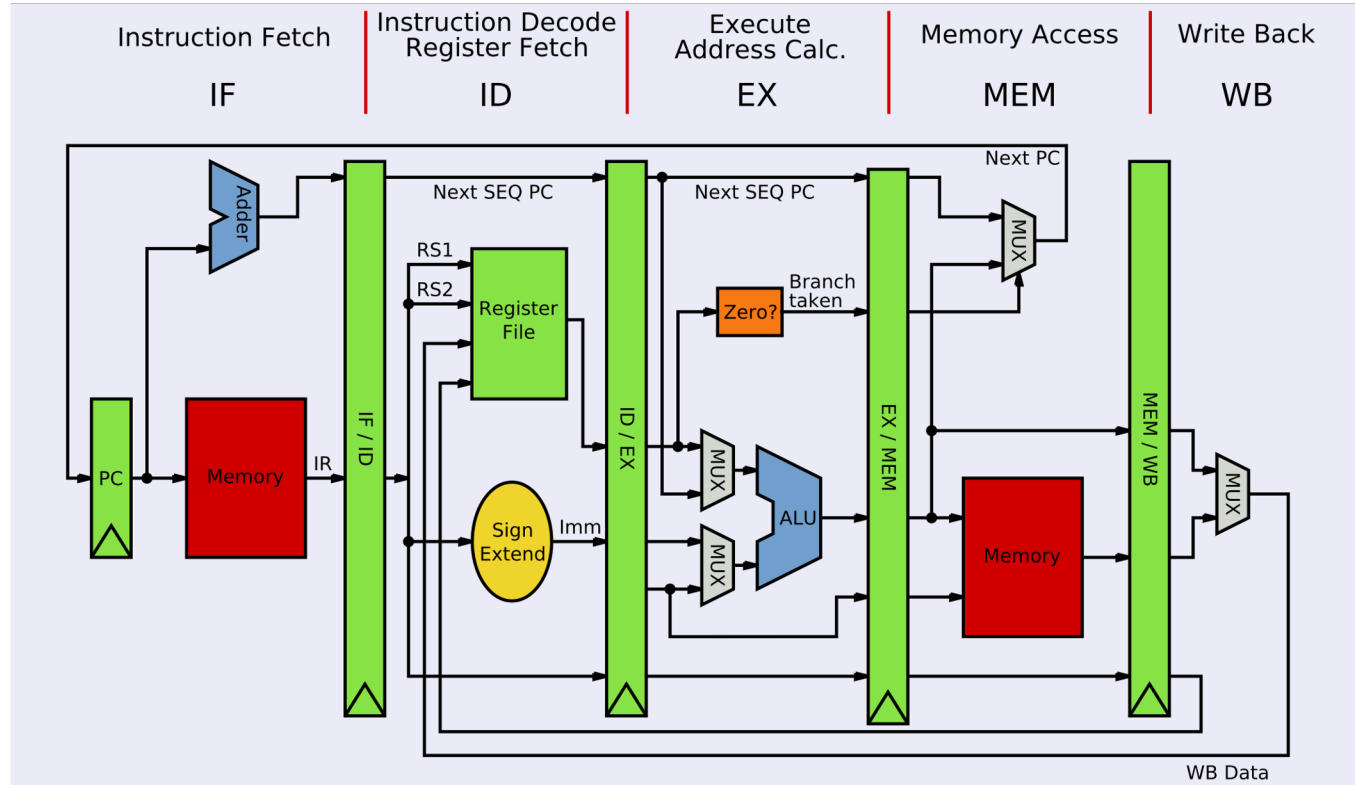
Universal Computer

Generalized Execution Cycle

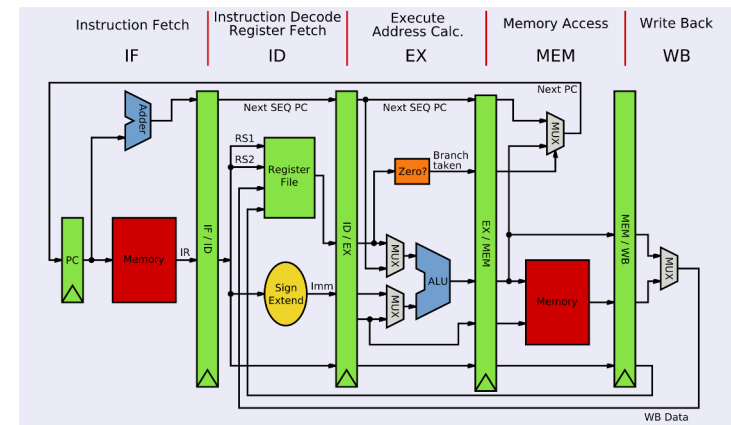
1. Fetch:
 - Move the instruction into the control unit
2. Decode
 - Set control lines to allow data to flow to ALU
3. Execute
 - Activate the ALU
4. Writeback
 - Write the data to a register or memory



MIPS Microarchitecture



MIPS Pipeline Execution



Instruction	Clock 1	Clock 2	Clock 3	Clock 4	Clock 5	Clock 6	Clock 7	Clock 8
#1	Fetch	Decode	Execute	Mem	WB	Fetch #6		
#2		Fetch	Decode	Execute	Mem	WB		
#3			Fetch	Decode	Execute	Mem	WB	
#4				Fetch	Decode	Execute	Mem	WB
#5					Fetch	Decode	Execute	Mem

Instruction Set Architectures

- The ISA is one level above the physical architecture
- Defines the following:
 - Supported instruction and their semantics
 - Supported data types
 - Registers: size, number, and purpose
 - Memory: layout, addressing, alignment, endianness
 - Memory is an array of bytes
 - OS interface:
- Does not define the following:
 - technology used
 - chip layout
 - memory implementation
 - etc
- Very Similar to an API:
- RISC versus CISC
 - Reduced Instruction Set Computer (RISC)
 - Complex Instruction Set Computer (CISC)

endianness: the order of bytes within a word

- big: 1,2,3,4 (yy/mm/dd)
- little: 4,3,2,1 (dd/mm/yy)
- middle: 3,4,1,2 (mm/dd/yy)

RISC Examples

ARM

MIPS

CISC Examples

AMD

x86

INTEL

x86

MIPS ISA Architecture: Instructions

- Three basic instruction types

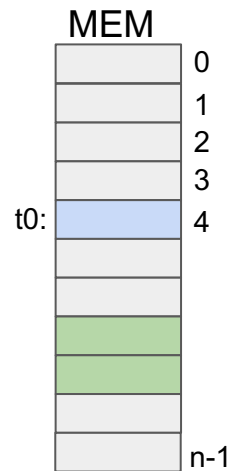
- Arithmetic, bitwise logic, etc.
- Data transfers
- Basic control flow

- Examples:

- `add $v0, $v0, $a0` # `$v0 = $v0 + $a0`
- `addi $v0, $v0, 2` # `$v0 = $v0 + 2`
- `srl $a0, $a1, 4` # `$a0 = $a1 >>> 4`
- `li $t0, 4` # `$t0 = 4`

- `move $t1, $t2` # `$t1 = $t2`
- `lb $s0, 0($t0)` # `$s0 = MEM[$t0]`
- `lh $s1, 3($t0)` # `$s1 = concat(MEM[$t0+3+1], MEM[$t0+3+0])`

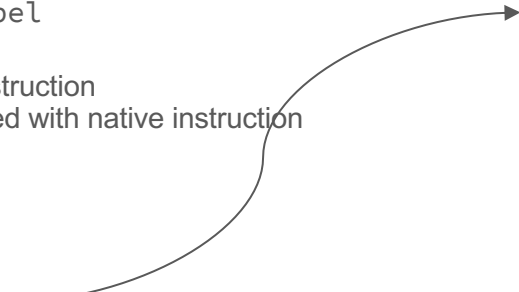
- `beq $t3, $t5, label` # `if ($t3 == $t5) goto label`
- `jal proc` # `method()`



`$t0 = 4;`
`0($t0) ⇔ $t0[0]`


Category of Instructions

- native instructions:
 - defined by the ISA, directly corresponds to a hardware operation
- idioms:
 - defined by the ISA, an alternate form of a hardware instruction
 - b label
 - beq \$zero, \$zero, label
- pseudo instructions:
 - an alternate syntactic form of an instruction
 - pseudo instruction is textual replaced with native instruction
 - lb \$t1, label+offset
 - lui \$at, &label
 - lb \$t1, offset(\$at)
- macros:
 - user-defined pseudo instruction
 - average \$v0, \$t2, \$t3
 - average(\$v0, \$t2, \$t3)
- subroutines
 - user defined abstraction, resulting
 - jal subroutine
 - nop
 - a change in control-flow
 - a ownership of registers



A curved arrow originates from the 'average' macro call in the 'macros' section of the list and points to this code block.

```
.macro average(%d, %s, %t)
    addu %d, %s, %t
    srl %d, 2
.end_macro
```



A curved arrow originates from the 'jal subroutine' instruction in the 'subroutines' section of the list and points to this code block. A second arrow originates from the 'nop' instruction in the same section and points to the 'nop' instruction in this code block.

```
subroutine:    .text
               nop
               jr $ra
```

MIPS ISA Architecture: Registers

- Data types:
 - byte, half, word
 - integer (signed/unsigned), binary32, binary64
- Registers:
 - 32: 32-bit integer registers
 - 32: 32-bit floating point registers
 - binary32: \$fp0 .. \$fp31
 - binary64: {\$fp0, \$fp1} .. {\$fp30, \$fp31}
 - 3: system registers: pc, hi, lo

Integer Registers

Name	Register Number	Usage
\$zero	0	the constant value 0
\$at	1	reserved for the assembler
\$v0-\$v1	2-3	value for results and expressions
\$a0-\$a3	4-7	arguments (procedures/functions)
\$t0-\$t7	8-15	temporaries
\$s0-\$s7	16-23	saved
\$t8-\$t9	24-25	more temporaries
\$k0-\$k1	26-27	reserved for the operating system
\$gp	28	global pointer
\$sp	29	stack pointer
\$fp	30	frame pointer
\$ra	31	return address

MIPS ISA (Architecture) Memory Layout

endian: the order of bytes within a word

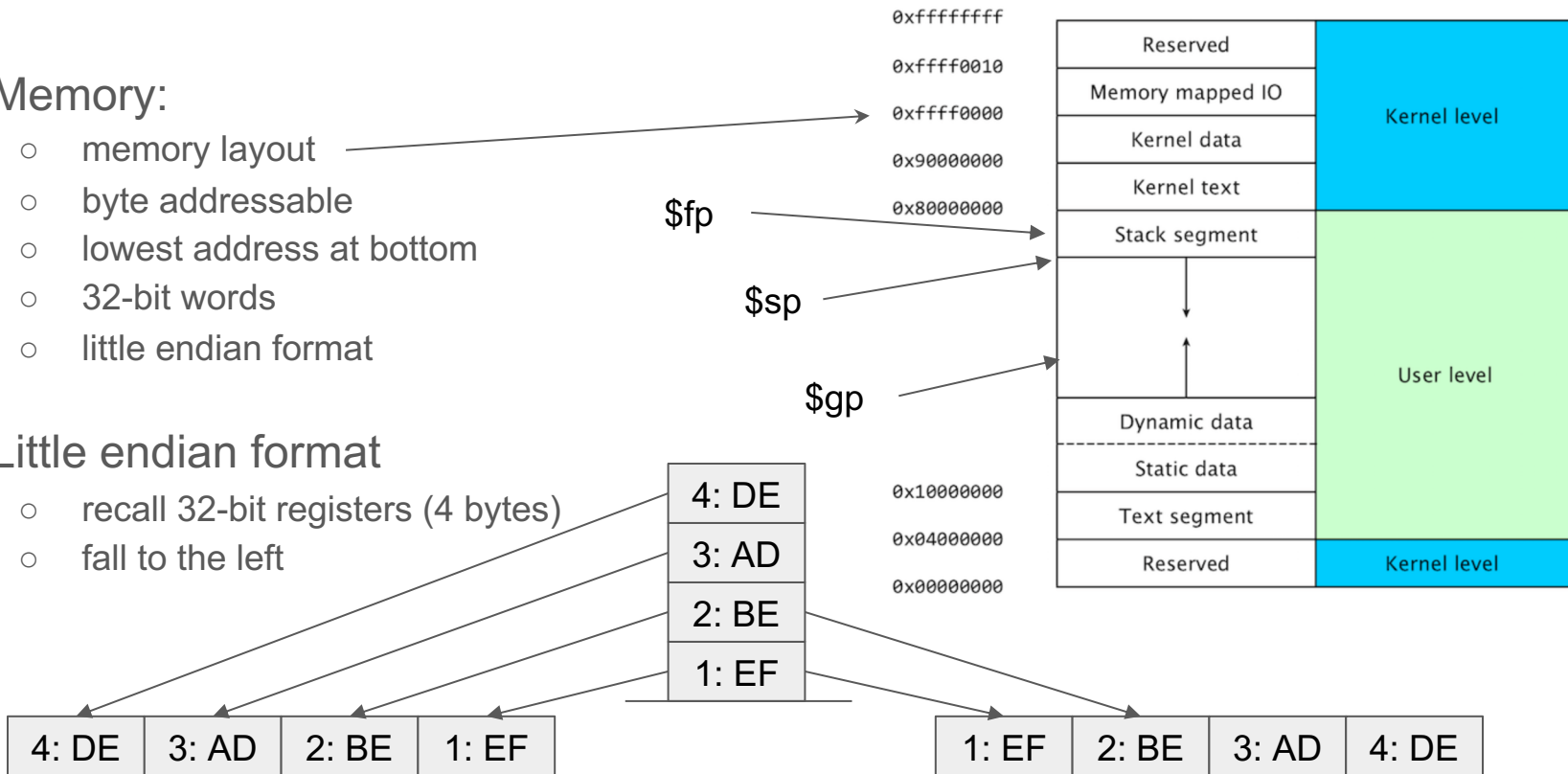
- big: 1,2,3,4 (yy/mm/dd)
- little: 4,3,2,1 (dd/mm/yy)
- middle: 3,4,1,2 (mm/dd/yy)

- **Memory:**

- memory layout
- byte addressable
- lowest address at bottom
- 32-bit words
- little endian format

- **Little endian format**

- recall 32-bit registers (4 bytes)
- fall to the left



MIPS ISA Architecture: OS interface

```
# Print the integer '1'
# Macro: print_di %imm
li $a0, %imm
li $v0, 1
syscall
```

- Service Requests to the operating system via the 'syscall' instruction

Service Name	\$v0	input: \$a0..\$a3	output: \$v0..\$v1
print integer	1	\$a0 = value	none
read integer	5	none	\$v0 = value
malloc	9	\$a0 = size	\$v0 = buffer address
exit	10	none	none
file read	14	\$a0 = fd, \$a1 = buffer address \$a2 = num bytes	\$v0 = bytes read -1 == error 0 == eof

