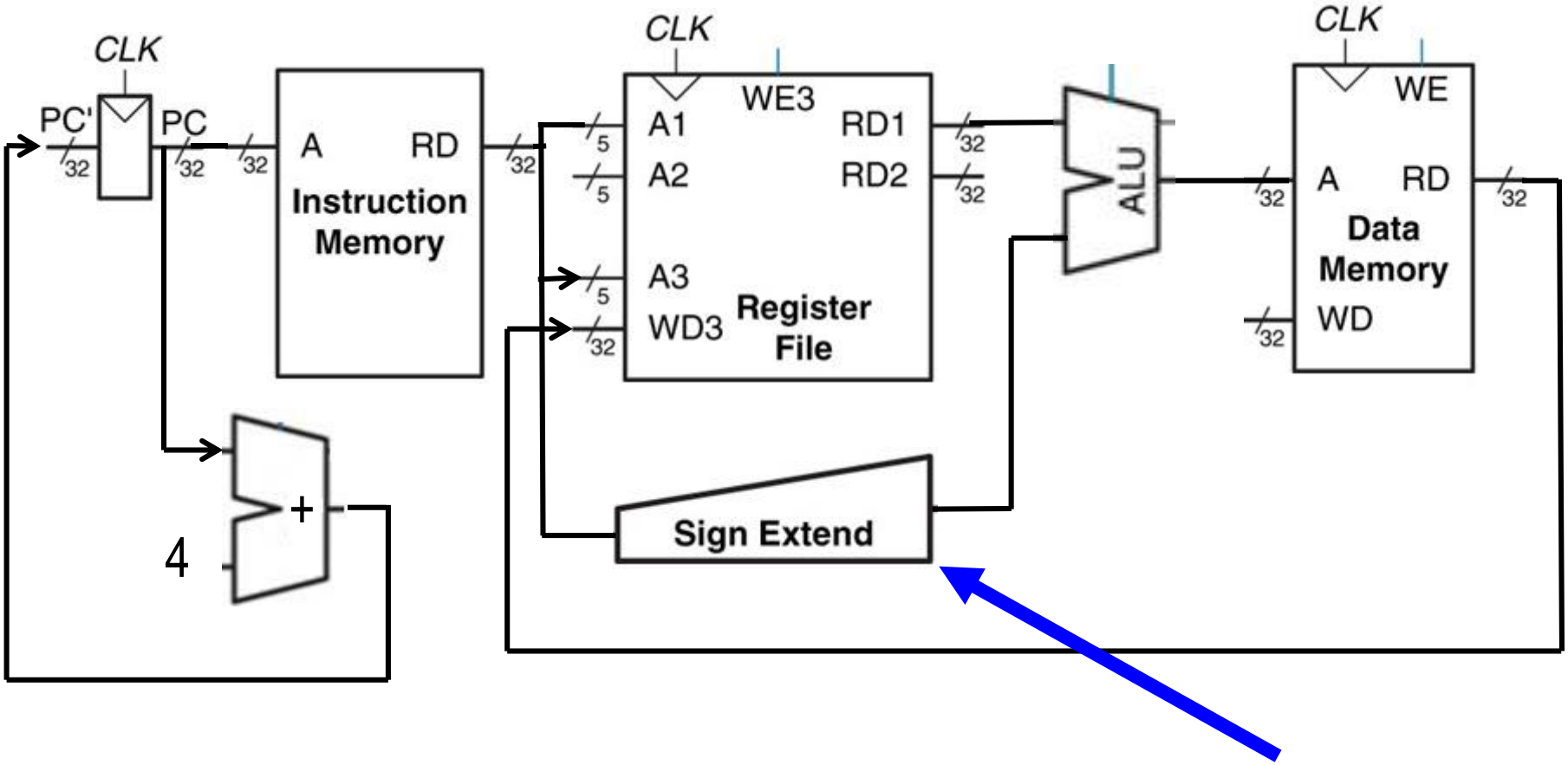
[LOAD / STORE]

MIPS Assembly Programming

[Load] / [Store]


Sign Extend

MIPS


CLK
PC'
PC
32
A
RD
Instruction Memory
5
A1
A2
A3
WD3
WE3
RD1
RD2
Register File
ALU
WE
Data Memory
WD
Sign Extend
+
4

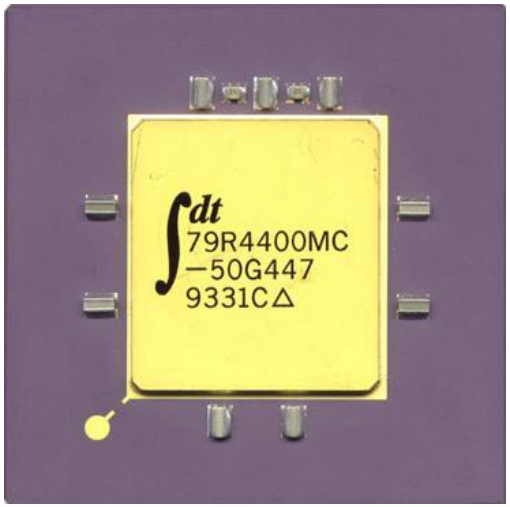
Sign Extend

- Use 0's or 1's to **extend the sign** of a number to 32-bits.
- **Assume we have a 32-bit CPU.**
- If a number is positive >> add 0's to left
  - 0101
  - **00000000000000000000000000000101**
- If a number is negative >> add 1's to left
  - 1010
  - **11111111111111111111111111111010**

MIPS and ARM based CPU's

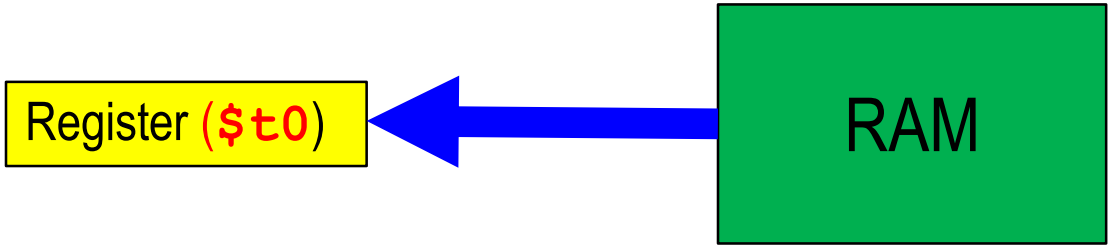
- Memory (RAM) access is allowed only with **Load** and **Store** instructions

It is a «**Load/Store**» Computer Architecture…



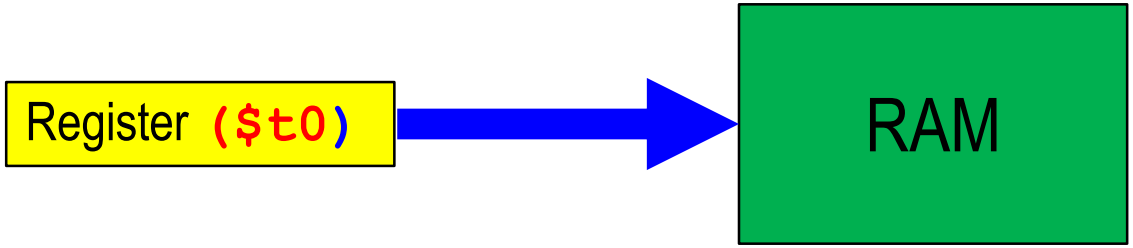
Accessing the RAM

- **Load instructions**: Read data from a RAM address and copy to a register. (`lw $t0`, [RAM])



- **Store instructions**: Write data from a register to a RAM address. (`sw $t0`, [RAM]).

.



RAM and Registers

- Our CPU has only 32 registers
- For large data structures (Arrays, Images, …)
- The 32 registers are not enough for storage
- We need more storage…
- Must use the System Memory (RAM)
  - Memory (RAM) is large
  - RAM to register and back, data transfer … is slow with respect to the speed of the register-to-registers transfer
- Commonly used variables are kept in registers.

Load instructions (`lb, lw`)

	Op	Operands	Description
○	`la`	*des, addr*	Load the address of a label.
	`lb(u)`	*des, addr*	Load the byte at *addr* into *des*.
	`lh(u)`	*des, addr*	Load the halfword at *addr* into *des*.
○	`li`	*des, const*	Load the constant *const* into *des*.
	`lui`	*des, const*	Load the constant *const* into the upper halfword of *des*, and set the lower halfword of *des* to 0.
	`lw`	*des, addr*	Load the word at *addr* into *des*.
	`lwl`	*des, addr*	
	`lwr`	*des, addr*	
○	`ulh(u)`	*des, addr*	Load the halfword starting at the (possibly unaligned) address *addr* into *des*.
○	`ulw`	*des, addr*	Load the word starting at the (possibly unaligned) address *addr* into *des*.

Store instructions (`sb, sw`)

	Op	Operands	Description
	`sb`	*src1, addr*	Store the lower byte of register *src1* to *addr*.
	`sh`	*src1, addr*	Store the lower halfword of register *src1* to *addr*.
	`sw`	*src1, addr*	Store the word in register *src1* to *addr*.
	`swl`	*src1, addr*	Store the upper halfword in *src* to the (possibly unaligned) address *addr*.
	`swr`	*src1, addr*	Store the lower halfword in *src* to the (possibly unaligned) address *addr*.
○	`ush`	*src1, addr*	Store the lower halfword in *src* to the (possibly unaligned) address *addr*.
○	`usw`	*src1, addr*	Store the word in *src* to the (possibly unaligned) address *addr*.

MIPS addressing

MIPS addressing modes

- Addressing; General methods to access the data in the registers or the memory (RAM):

- **Register addressing**
  - Register: `add $s0, $t2, $t3`
  - Immediate: `addi $s4, $t5, 53`

- **Memory addressing**
  - Direct memory
  - Indexed or Based memory

- MIPS uses Indexed or Based addressing to access (Load [`lw`] … Store [`sw`]) the RAM.

Direct and Indexed memory access

```
lw $rd, x
sw $rs, x
```

Direct

Address represented by label **x**

```
lw $rd, offset($rs)
sw $rs, offset($rd)
```

Indexed or Based

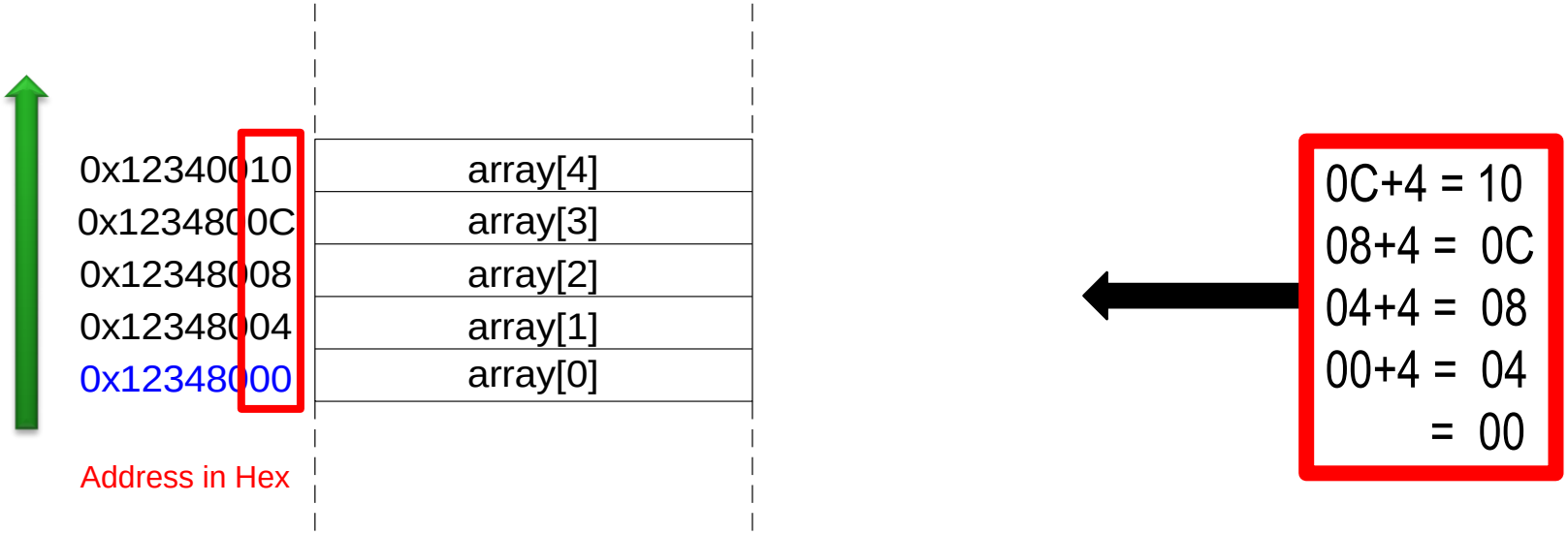
Uses a register as an offset. The offset is added to the address in the operand. This is the Effective Address (EA) of the data.

MIPS memory access instructions

- **WORD (Address in memory must be word-aligned)**
  - **`lw`; Loads a word from a location in memory to a register**
  - **`sw`; Stores a word from a register to a location in memory**
- **BYTE (Address not aligned-**Only one byte is loaded from memory)
  - **`lb`; Loads a byte from a location in memory to a register. Sign extends this result in the register.**
  - **`sb`; Store the least significant byte of a register to a location in memory.**

Memory word-alignment

MIPS requires that all words start at byte addresses and are multiples of 4 bytes (4 x 8 = 32-bits)



`.align` # directive the next datum on a 2^n byte boundary.

Instruction analysis and example

lw

loads a **w**ord from a location in memory to a register

lw

- The Memory Write operation is called: **load**
- **Mnemonic:** **l**oad **w**ord (**lw**)
- Instruction Format:

```
lw $t2,4($t0)
```

- Effective Address (EA): `$t2` ← `Mem[$t0+4]`.

Indexed or Based addressing

- The address operand specifies an Immediate (signed constant or Offset) and a `rs` (register source) that holds the based-address:

```
lw $t2, 4($t0)
```

- **The actual memory location** from where the operand is retrieved is **the Effective Address (EA).**

- The Effective Address (EA) is determined by **adding** the offset to the Register (based-address)

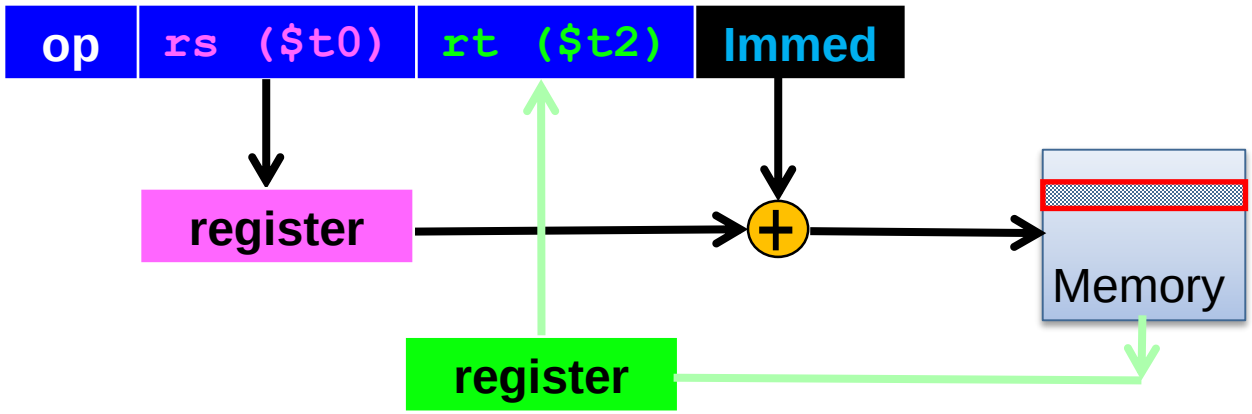
EA ← Mem{ $(register) + sign-ext$_{32}$(offset) }

```
$t2 ← Mem{$t0 + 00000000000000000000000000000100}
```

MIPS uses INDEXED ADDRESSING

Indexed or Based Addressing; **Load**

Load a word from a location in memory to a register



1. Go to the memory address [4+$t0]
2. Take the data from the memory location [4+$t0] and put them in register: ($t2)

```
lw $t2, 4($t0)
```

$t2 ← Mem[$t0+4]

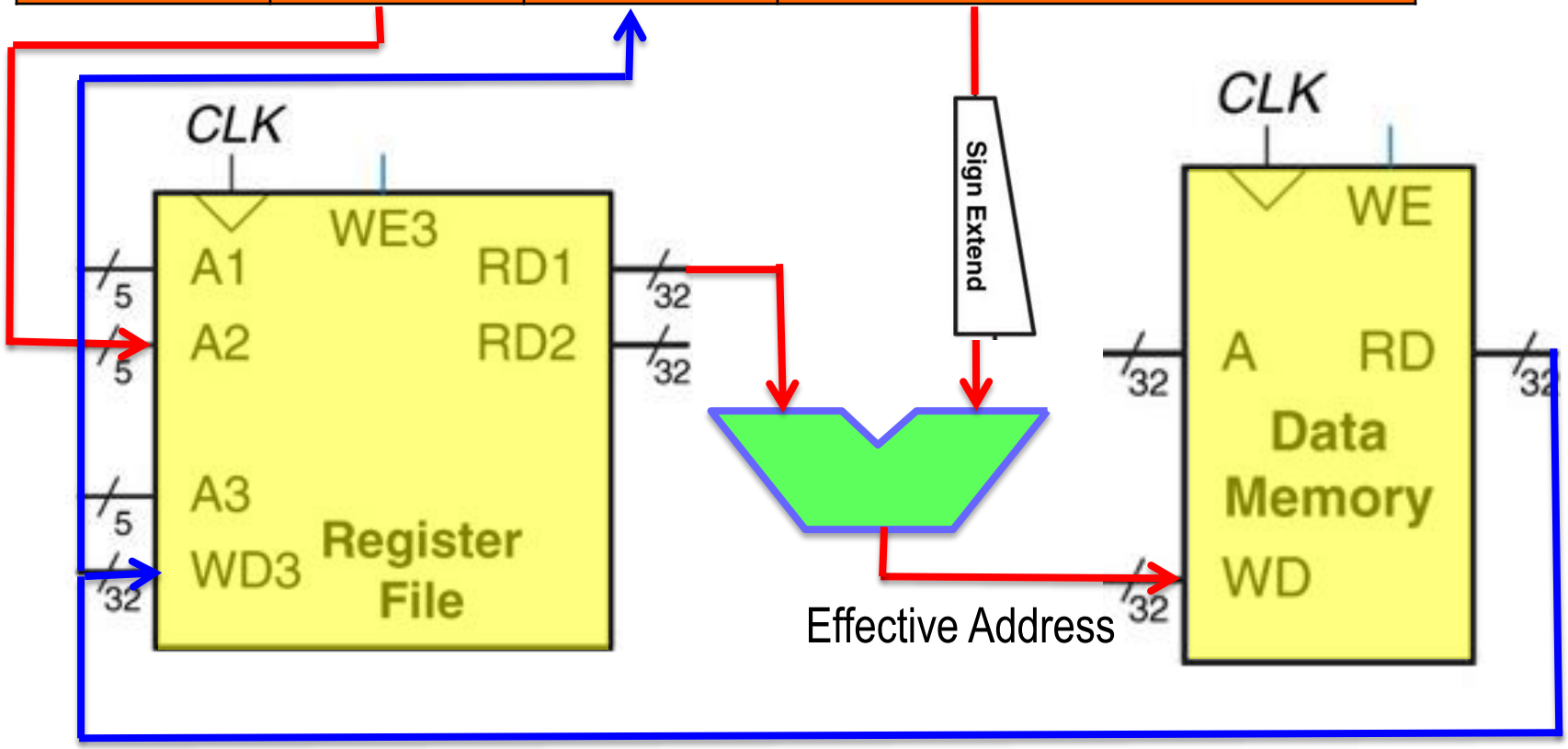
op code	rs ($t0)	rt($t2)	Address/Immediate
100011	01000	01010	0000 0000 0000 0100

Address	Code	Basic	
0x00400000	0x8d0a0004	lw $10,4($8)	1: lw $t2, 4($t0)

OR …

lw $t2, 4($t0)

op code	rs ($t0)	rt ($t2)	Address/Immediate
100011	01000	01010	0000 0000 0000 0100



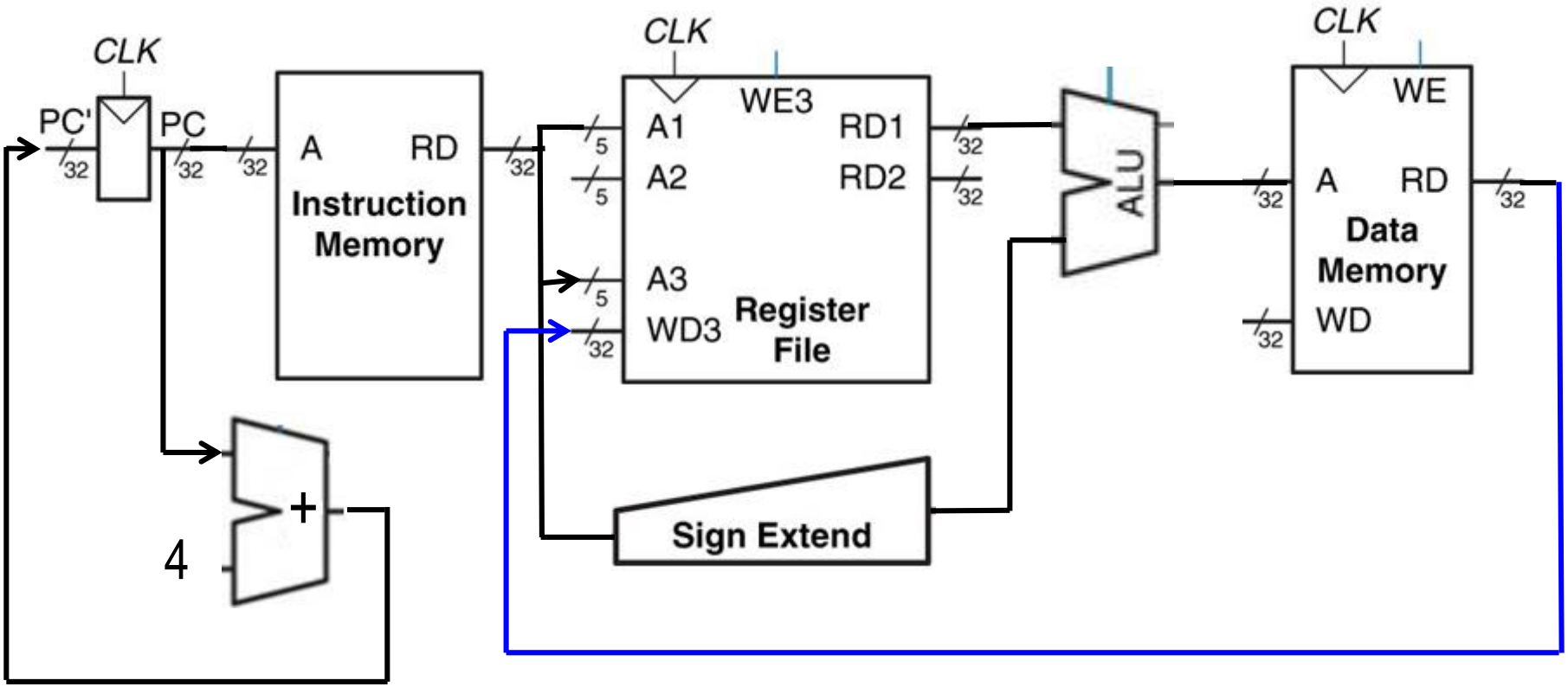
rs is the first source register

rt is the second source register

Address	Code	Basic	
0x00400000	0x8d0a0004	lw $10,4($8)	1: lw $t2, 4($t0)

OR …

lw $t2, 4($t0)


CLK
PC'
PC
32
A
RD
Instruction Memory
CLK
WE3
A1
A2
A3
WD3
RD1
RD2
Register File
5
ALU
CLK
WE
A
RD
Data Memory
WD
+
4
Sign Extend

Example

`lw`

Word addressable memory

- Each 32-bit data word has a unique address

Word Address	Data	
⋮	⋮	⋮
00000003	4 0 F 3 0 7 8 8	Word 3
00000002	0 1 E E 2 8 4 2	Word 2
00000001	F 2 F 1 A C 0 7	Word 1
. 00000000	A B C D E F 7 8	Word 0

Loading (reading) from Memory (`lw`)

- Memory read is called **load**
- **Mnemonic:** `l`oad `w`ord (`lw`)
- **Format:**

  `lw $t2, 1($t0)`
- Effective Address calculation: EA ← `Mem`{`$t0` + sign-ext$_{32}$(offset)}
  - add *base-address* (`$t0`) to the *offset* (`1`)
  - `$t2` ← `Mem`[`$t0+1`] (Effective-address)
- **Result:**
  - `$t2` holds the value (data) of the effective-address (`$t0+1`)

(Any register may be used as base address).

Example: `lw`

- Load (Read) the word of data at memory address: 0x00000001, into register: `$t2`
  - Effective Address:
    `$t2` ← `Mem[$t0+1]` = `0x00000001`
  - `$t2` holds the value: 0xF2F1AC07 after load

1. Go to the memory address [1+$t0]
2. Take the data and put them in the register ($t2).

Assembly code

`$t0 = 0x00000000 (base address)`

```
lw $t2, 1($t0)
```

Register File	
Reg	value
$t0	
… .	
$t2	0xF2F1AC07

Word Address	Data	
⋮	⋮	⋮
00000003	4 0 F 3 0 7 8 8	Word 3
00000002	0 1 E E 2 8 4 2	Word 2
00000001	F 2 F 1 A C 0 7	Word 1
00000000	A B C D E F 7 8	Word 0

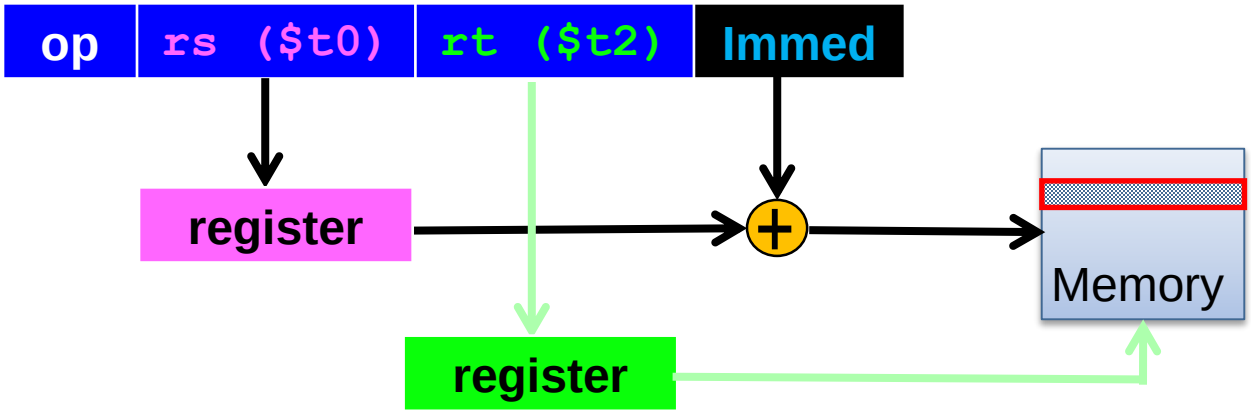
Instruction analysis and example

sw

stores (writes) a **w**ord from a register to a location in memory

Indexed or Based Addressing; **Store**

Store a word from a register to a location in memory.



1. Go to the memory address [4+$t0]
2. Put the data of the register ($t2), to the memory address [4+$t0]

```
sw $t2, 4($t0)
```

$t2 → Mem[$t0+4]

op code	rs ($t0)	rt($t2)	Address/Immediate
101011	01000	01010	0000 0000 0000 0100

Address	Code	Basic	
0x00400000	0xad0a0004	sw $10,4($8)	1: sw $t2, 4($t0)

Example

sw

`sw` (Storing to Memory)

- The Memory Write operation is called: **store**
- **Mnemonic:** **s**tore **w**ord (**sw**)
- Instruction Format:

```
sw $s0,3($t0)
```

- `$s0` → `Mem[$t0+3]` (Effective Address)
- **Result:** The value (data) of the register `$s0` is stored at Effective Address: `[$t0+3]` of the RAM.

Example: `sw`

- Example: Write (store) the value of the register `$s0` into memory address: 0x00000003

Example: **sw**

- Example: Write (store) the value of the register **$s0** into memory address: 0x00000003
  - Effective-Address:
    Mem[**$t0**+3] ➔ 0x0000000+3 = 0x00000003
  - To the above address load the word: **0x40F30788**

Assembly code

$t0 = 0x00000000 (base address)

sw $s0, 3($t0)

Register File	
Reg	value
$t0	
… .	
$s0	0x40F30788

Word Address	Data	
⋮	⋮	⋮
00000003	4 0 F 3 0 7 8 8	Word 3
00000002	0 1 E E 2 8 4 2	Word 2
00000001	F 2 F 1 A C 0 7	Word 1
00000000	A B C D E F 7 8	Word 0

`lw` … `sw`

Indirect memory access

Direct memory access

```
lw      $t0, x      #Load contents of RAM address x into $t0.

sw      $t0, x      #Store contents of $t0 into memory address x.

x:      (Memory Address, where a value is stored)
```

Example-1

Store-Load

```
#  Example-1: Store to x, then load to register: $t1

        .text
        .globl main
main:
        lw   $t0, x          # Load contents of  Memory address: x into register: $t0
        addi $t0, $t0, 3
        sw   $t0, x
        lw   $t1, x
        li   $v0, 10
        syscall

        .data
x:      .word 9              # Memory address: x = 00000000000000000000000000001001
```

$t0 = ?
$t1 = ?

Result

```
#  Example-1: Store to x, then load to $t1

        .text
        .globl main
main:
        lw   $t0, x
        addi $t0, $t0, 3
        sw   $t0, x
        lw   $t1, x
        li   $v0, 10
        syscall

        .data
x:      .word 9
```

$t0 = 12
$t1 = 12

Coproc 1 | Coproc 0

Registers

Name	Number	Value
$zero	0	0
$at	1	268500992
$v0	2	0
$v1	3	0
$a0	4	0
$a1	5	0
$a2	6	0
$a3	7	0
$t0	8	12
$t1	9	12
$t2	10	0
$t3	11	0
$t4	12	0
$t5	13	0
$t6	14	0
$t7	15	0
$s0	16	0
$s1	17	0
$s2	18	0
$s3	19	0
$s4	20	0
$s5	21	0
$s6	22	0
$s7	23	0
$t8	24	0
$t9	25	0
$k0	26	0
$k1	27	0
$gp	28	268468224
$sp	29	2147479548
$fp	30	0
$ra	31	0
pc		4194332
hi		0
lo		0

Example-2

Load-Store and **addi**

```
#  Example-2: load-store

        .text
        .globl main
main:
        lw   $t0, x                 # Load contents of Memory address x into register $t0
        addi $t0, $t0, 3
        sw   $t0, x
        lw   $t1, x
        addi $t2, $t1, 3
        li   $v0, 10
        syscall

        .data
x:      .word 9
```

```
$t0 = ?
$t1 = ?
$t2 = ?
```

Result

```
#   load-store example-2

        .text
        .globl main
main:
        lw   $t0, x
        addi $t0, $t0, 3
        sw   $t0, x
        lw   $t1, x
        addi $t2, $t1, 3
        li   $v0, 10
        syscall

        .data
x:      .word 9
```

.word: Data type

$t0 = 12
$t1 = 12
$t2 = 15

Coproc 1 | Coproc 0

Registers

Name	Number	Value
$zero	0	0
$at	1	268500992
$v0	2	0
$v1	3	0
$a0	4	0
$a1	5	0
$a2	6	0
$a3	7	0
$t0	8	12
$t1	9	12
$t2	10	15
$t3	11	0
$t4	12	0
$t5	13	0
$t6	14	0
$t7	15	0
$s0	16	0
$s1	17	0
$s2	18	0
$s3	19	0
$s4	20	0
$s5	21	0
$s6	22	0
$s7	23	0
$t8	24	0
$t9	25	0
$k0	26	0
$k1	27	0
$gp	28	268468224
$sp	29	2147479548
$fp	30	0
$ra	31	0
pc		4194336
hi		0
lo		0

Except … data type: `.word`

Other Data data types: [`.byte`] and [`.space`]

Data:
`.byte`

Data:
`.word`

Data:
`.space`

[.byte], [.word], [.space]

(Bytes):

```
array:  .byte 'x','y','z',…(1 byte = 8-bits)
```

(Word):

```
x:      .word 1, 2, 3,…     (1 word = 32-bits)
```

(Array-Space):

```
array:  .space x
```

Example .space

```
array:   .space 12
```

Allocate 12-consecutive bytes, with storage uninitialized
Create a 12-element character array
Equivalent to a 3-element integer array: (3 x 4 = 12)

Indexed or based addressing

REVIEW

- `lw $t2,4($t0)  # $t2 ← Mem[$t0 + 4]`
  - load word at RAM address `($t0+4)` into register `$t2`
  - `$t0` contains the base address
  - "**4**" gives offset from address in register `$t0`
- `sw $t2,4($t0)  # $t2 → Mem[$t0 + 4]`
  - store word in register `$t2` into RAM at address `($t0 + 4)`
  - `$t0` contains the base address
  - negative offsets are fine
- Note: based addressing is especially useful for:
  - arrays; access elements as offset from base address
  - stacks; easy to access elements at offset from stack pointer or frame pointer

Offset values for indexed or based …

```
lw   $t1,0($t0)
```
offset: **0** (base address)

```
lw   $t1,1($t0)
```
offset: **1** (for characters)

1-Byte/character

```
lw   $t1,4($t0)
```
offset: **4** (for integers)

4-Bytes (1-Word)

Indexed or Based addressing

(Examples)

1 memory location is used: `0($s0)`

Example-3

Store and Load

```
#  load-store WORD

      .text
      .globl main
main:
la    $s0, 0xFFFF0010  # Load FFFF0010 address to register: $s0
li    $t0, 123
sw    $t0, 0($s0)      # Store to memory location FFFF0010
lw    $t1, 0($s0)      # Load from memory location FFFF0010 to reg. $t1
li    $t2, 5
add   $t3, $t2, $t1

li    $v0, 10
syscall
```

```
$t0 = ?
$t1 = ?
$t2 = ?
$t3 = ?
```

Result

Load-Store.asm

```
#  load-store WORD example-1

      .text
      .globl main
main:

la    $s0, 0xFFFF0010 # Load FFFF0010 address to $s0
li    $t0  123
sw    $t0, 0($s0)     # Store to memory location  FFFF0010
lw    $t1, 0($s0)     # Load from memory location FFFF0010 to reg. $t1
li    $t2, 5
add   $t3, $t2, $t1

li $v0, 10
syscall
```

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0
$at	1	-65536
$v0	2	10
$v1	3	0
$a0	4	0
$a1	5	0
$a2	6	0
$a3	7	0
$t0	8	123
$t1	9	123
$t2	10	5
$t3	11	128
$t4	12	0
$t5	13	0
$t6	14	0
$t7	15	0
$s0	16	-65520
$s1	17	0
$s2	18	0
$s3	19	0
$s4	20	0
$s5	21	0
$s6	22	0
$s7	23	0
$t8	24	0
$t9	25	0
$k0	26	0
$k1	27	0
$gp	28	268468224
$sp	29	2147479548
$fp	30	0
$ra	31	0
pc		4194340
hi		0
lo		0

```
$t0 = 123
$t1 = 123
$t2 =   5
$t3 = 128
```

1 memory location is used: `0($s0)`

Example-4

Store-Load in the same Memory location

```
#  load-store WORD

      .text
      .globl main
main:

la    $s0, 0xFFFF0010  # Load FFFF0010 address to $s0
li    $t0, 15
sw    $t0, 0($s0)      # Store to memory location FFFF0010
lw    $t1, 0($s0)      # Load from memory location FFFF0010 to reg. $t1

li    $t2, 9
sw    $t2, 0($s0)      # Store to memory location FFFF0010
lw    $t3, 0($s0)      # Load from memory location FFFF0010 to reg. $t3
add   $t4, $t1, $t3

li    $v0, 10
syscall
```

```
$t0 = ?
$t1 = ?
$t2 = ?
$t3 = ?
$t4 = ?
```

Trace the program

```
3        .text
4        .globl main
5   main:
6
7   la    $s0, 0xFFFF0010
8   li    $t0  15
9   sw    $t0, 0($s0)
10  lw    $t1, 0($s0)
11
12  li    $t2  9
13  sw    $t2, 0($s0)
14  lw    $t3, 0($s0)
15
16  add   $t4, $t1, $t3
17
18  li $v0, 10
19  syscall
```

line	$t0	$t1	$t2	$t3	$t4
(8)	15				
(10)		15			
(12)			9		
(14)				9	
(16)					24

Result

```
#  load-store: Example-4

        .text
        .globl main
main:

la    $s0, 0xFFFF0010 # Load FFFF0010 address to $s0
li    $t0  15
sw    $t0, 0($s0)     # Store to memory location  FFFF0010
lw    $t1, 0($s0)     # Load from memory location FFFF0010 to reg. $t1

li    $t2  9
sw    $t2, 0($s0)     # Store to memory location  FFFF0010
lw    $t3, 0($s0)     # Load from memory location FFFF0010 to reg. $t3

add   $t4, $t1, $t3

li $v0, 10
syscall
```

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0
$at	1	-65536
$v0	2	10
$v1	3	0
$a0	4	0
$a1	5	0
$a2	6	0
$a3	7	0
$t0	8	15
$t1	9	15
$t2	10	9
$t3	11	9
$t4	12	24
$t5	13	0
$t6	14	0
$t7	15	0
$s0	16	-65520
$s1	17	0
$s2	18	0
$s3	19	0
$s4	20	0
$s5	21	0
$s6	22	0
$s7	23	0
$t8	24	0
$t9	25	0
$k0	26	0
$k1	27	0
$gp	28	268468224
$sp	29	2147479548
$fp	30	0
$ra	31	0
pc		4194348
hi		0
lo		0

```
$t0 = 15
$t1 = 15
$t2 =  9
$t3 =  9
$t4 = 24
```

2 memory locations are used:

```
0($s0)
4($s0)
```

Example-5

With 2 memory locations

```
#  load-store WORD

        .text
        .globl main
main:
        la    $s0, 0xFFFF0010
        li    $t0, 15
        sw    $t0, 0($s0)
        lw    $t1, 0($s0)

        li    $t2, 9
        sw    $t2, 4($s0)
        lw    $t3, 4($s0)
        add   $t4, $t1, $t3

        li    $v0, 10
        syscall
```

```
$t0 = ?
$t1 = ?
$t2 = ?
$t3 = ?
$t4 = ?
```

Result

```
#  load-store WORD

        .text
        .globl main
main:
        la    $s0, 0xFFFF0010
        li    $t0, 15
        sw    $t0, 0($s0)
        lw    $t1, 0($s0)

        li    $t2, 9
        sw    $t2, 4($s0)
        lw    $t3, 4($s0)
        add   $t4, $t1, $t3

        li    $v0, 10
        syscall
```

```
$t0 = 15
$t1 = 15
$t2 =  9
$t3 =  9
$t4 = 24
```

`S11`

3 memory locations are used:

```
0($t0)
4($t0)
8($t0)
```

Example-6

With sll

```
        .text
        .globl main
main:
        la   $t0, myData
        lw   $t1, 0($t0)
        lw   $t2, 4($t0)
        add  $t3, $t1, $t2
        sw   $t3, 8($t0)
        sll  $t4, $t3, 1
        lw   $t5, 8($t0)

        li    $v0, 10
        syscall

        .data
myData: .word 100, 200          # allocate 2-words (0 and 4)
```

In the memory

```
4($t0) ⇦ 200
0($t0) ⇦ 100
```

```
$t1 = ?
$t2 = ?
$t3 = ?
$t4 = ?
$t5 = ?
```

Trace the program

6-example*

```
1          .text
2          .globl main
3  main:
4          la   $t0, myData
5          lw   $t1, 0($t0)
6          lw   $t2, 4($t0)
7          add  $t3, $t1, $t2
8          sw   $t3, 8($t0)
9          sll  $t4, $t3, 1
10         lw   $t5, 8($t0)
11
12         li    $v0, 10
13         syscall
14
15         .data
16 myData:.word 100, 200
17
```

line	$t1	$t2	$t3	$t4	$t5
(5)					
(6)					
(7)					
(8)					
(9)					
(10)					

$t1 = ?
$t2 = ?
$t3 = ?
$t4 = ?
$t5 = ?

Trace the program

6-example*

```
1          .text
2          .globl main
3  main:
4          la   $t0, myData
5          lw   $t1, 0($t0)
6          lw   $t2, 4($t0)
7          add  $t3, $t1, $t2
8          sw   $t3, 8($t0)
9          sll  $t4, $t3, 1
10         lw   $t5, 8($t0)
11
12         li    $v0, 10
13         syscall
14
15         .data
16 myData:.word 100, 200
17
```

line	$t1	$t2	$t3	$t4	$t5
(5)	**100**				
(6)	100	**200**			
(7)	100	200	**300**		
(8)	100	200	300		
(9)	100	200	300	**600**	
(10)	**100**	**200**	**300**	**600**	**300**

```
$t1 = 100
$t2 = 200
$t3 = 300
$t4 = 600
$t5 = 300
```

Result

6-example*

```
        .text
        .globl main
main:
        la   $t0, myData
        lw   $t1, 0($t0)
        lw   $t2, 4($t0)
        add  $t3, $t1, $t2
        sw   $t3, 8($t0)
        sll  $t4, $t3, 1
        lw   $t5, 8($t0)

        li    $v0, 10
        syscall

        .data
myData:.word 100, 200
```

Registers | Coproc 1 | Coproc 0

Name	Number	Value
$zero	0	0
$at	1	268500992
$v0	2	10
$v1	3	0
$a0	4	0
$a1	5	0
$a2	6	0
$a3	7	0
$t0	8	268500992
$t1	9	100
$t2	10	200
$t3	11	300
$t4	12	600
$t5	13	300
$t6	14	0
$t7	15	0
$s0	16	0
$s1	17	0
$s2	18	0
$s3	19	0
$s4	20	0
$s5	21	0
$s6	22	0
$s7	23	0
$t8	24	0
$t9	25	0
$k0	26	0
$k1	27	0
$gp	28	268468224
$sp	29	2147479548
$fp	30	0
$ra	31	0
pc		4194344
hi		0
lo		0

$t1 = 100
$t2 = 200
$t3 = 300
$t4 = 600
$t5 = 300

Next Arrays