Arrays

Based on slides by Jared Moore

Arrays

Useful for accessing large amounts of data.

Organized by sequential data addresses in memory.

Each element identified by a number (index)

Number of elements is the size.

	0	1	2	3	4	5	6
A	73	95	112	-3	14	56	0

```
int A[7];
A[0] = 73
A[1] = 95;
…
```

How many bits does it take to store A[0]?

How many registers would it take to store all of A?

Arrays

No way to store all of array in registers

We have just 32 registers to work with, and each array element is as large as a register

Certainly want to be able to handle arrays of more than 32 elements!

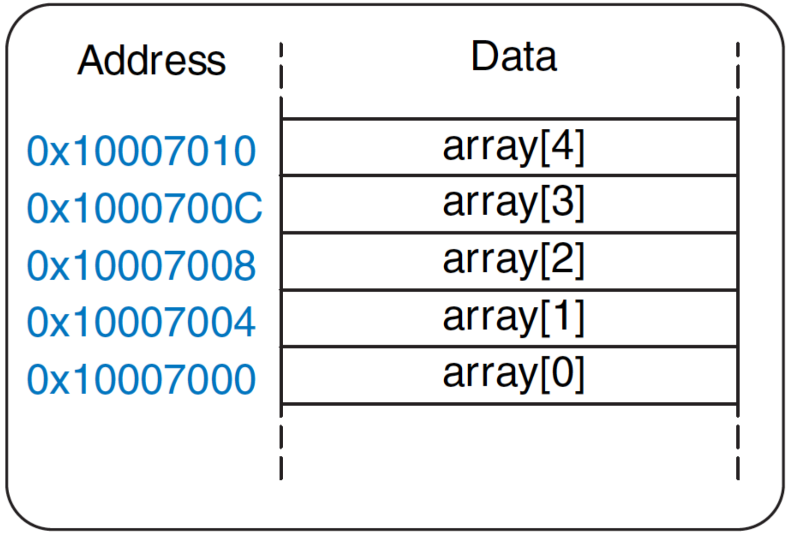
Arrays

Recall our three options for storage in MIPS

- Registers – small and fast
- Immediates – hard-coded constant (not exactly storage)
- Memory – large and slow

When we want to store something large, we must store it in memory

How an array is stored



Address	Data
0x10007010	array[4]
0x1000700C	array[3]
0x10007008	array[2]
0x10007004	array[1]
0x10007000	array[0]

Main Memory

How an array is stored

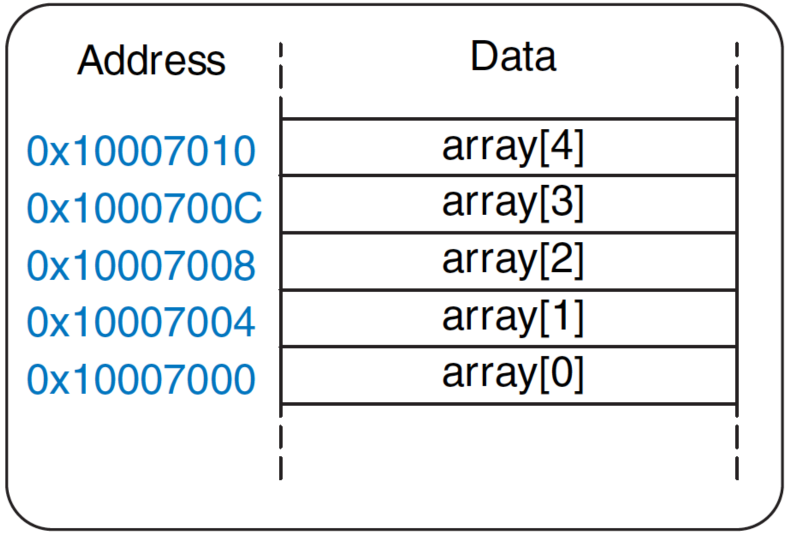
Key to accessing arrays:

If we know where first element is stored and how large each element is, we can determine location of any array element

How an array is stored

Assume register `$t1` holds address of 0th element of array

How would we place address of second element of array into `$t2`?



Address	Data
0x10007010	array[4]
0x1000700C	array[3]
0x10007008	array[2]
0x10007004	array[1]
0x10007000	array[0]

Main Memory

How an array is stored

Assume register `$t1` holds address of 0th element of array

Address of 0th element known as **base address**

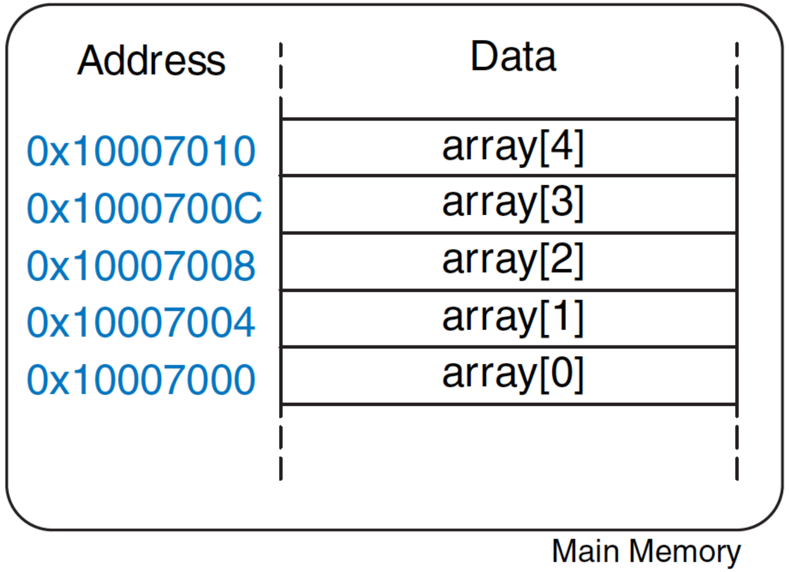
How would we place address of second element of array into `$t2`?

```
addi $t2, $t1, 4
```

How an array is stored

Critical to remember distinction between *location* and *data*

In previous example, `$t2` holds *address* of `array[1]`

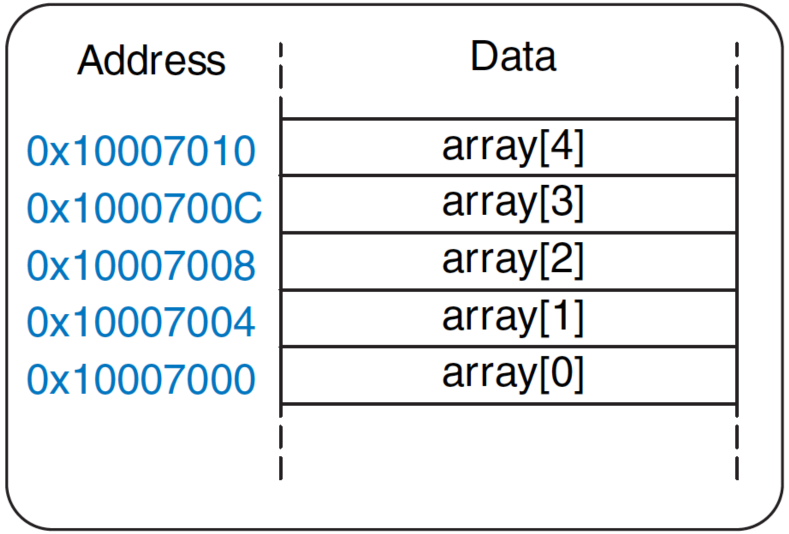


Address	Data
0x10007010	array[4]
0x1000700C	array[3]
0x10007008	array[2]
0x10007004	array[1]
0x10007000	array[0]

How an array is stored

To actually work with value `array[1]`, must get it into register

How to get *value* of `array[1]` into register `$t3`?



Address	Data
0x10007010	array[4]
0x1000700C	array[3]
0x10007008	array[2]
0x10007004	array[1]
0x10007000	array[0]

Main Memory

How an array is stored

Same way we get anything from memory:

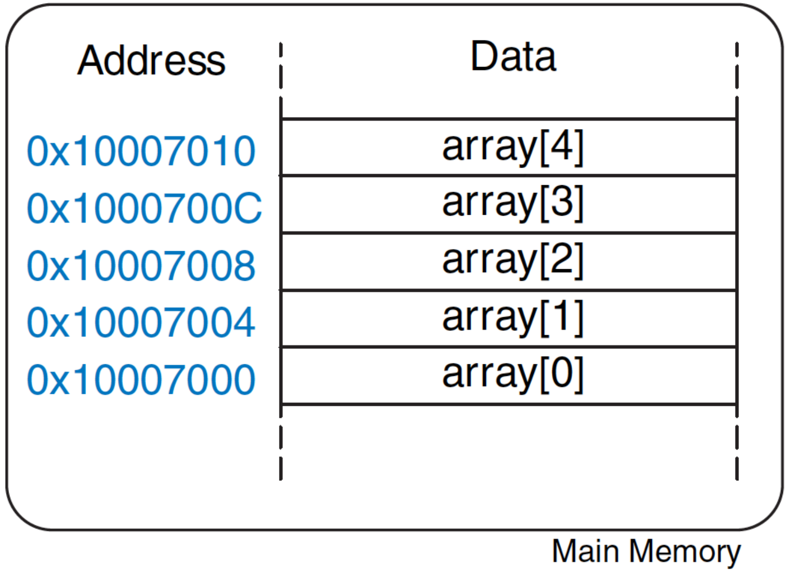
```
lw $t3, 0($t2)
```

Recall that `$t2` holds location of entry

How an array is stored

Since `$t1` holds base address, could also do

```
lw $t3, 4($t1)
```



Address	Data
0x10007010	array[4]
0x1000700C	array[3]
0x10007008	array[2]
0x10007004	array[1]
0x10007000	array[0]

Main Memory

High-Level Code

```
int array[5];



array[0] = array[0] * 8;



array[1] = array[1] * 8;
```

High-Level Code

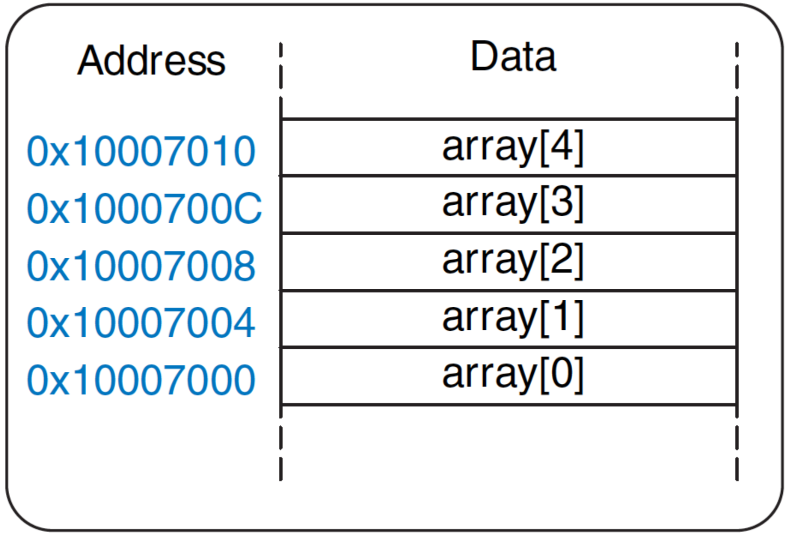
```
int array[5];



array[0] = array[0] * 8;



array[1] = array[1] * 8;
```



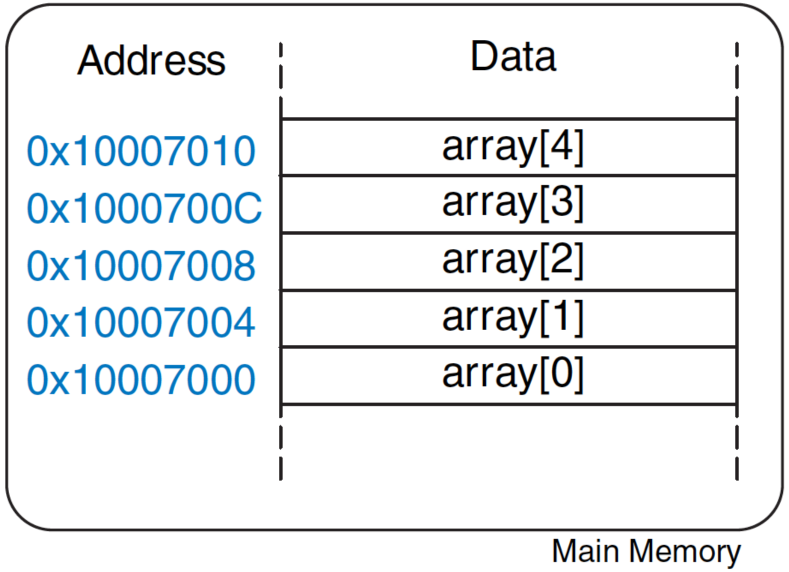
Address	Data
0x10007010	array[4]
0x1000700C	array[3]
0x10007008	array[2]
0x10007004	array[1]
0x10007000	array[0]

Main Memory

High-Level Code	MIPS Assembly Code

`int array[5];`	`# $s0 = base address of array`
	`  lui $s0, 0x1000          # $s0 = 0x10000000`
	`  ori $s0, $s0, 0x7000     # $s0 = 0x10007000`

`array[0] = array[0] * 8;`	
`array[1] = array[1] * 8;`	



Address	Data
0x10007010	array[4]
0x1000700C	array[3]
0x10007008	array[2]
0x10007004	array[1]
0x10007000	array[0]

Main Memory

Note about `lui` + `ori`

In code examples here, need some way of getting base address into a register

`lui` puts upper 16 bits, `ori` puts lower 16 bits

Try to convince yourself that we cannot use a single command to put a 32-bit value into a register

Note about `lui` + `ori`

Good to know that this is how base address actually input "behind the scenes"

You *should not* actually use this pattern in your code because you should not work directly with memory addresses – let MIPS handle it for you

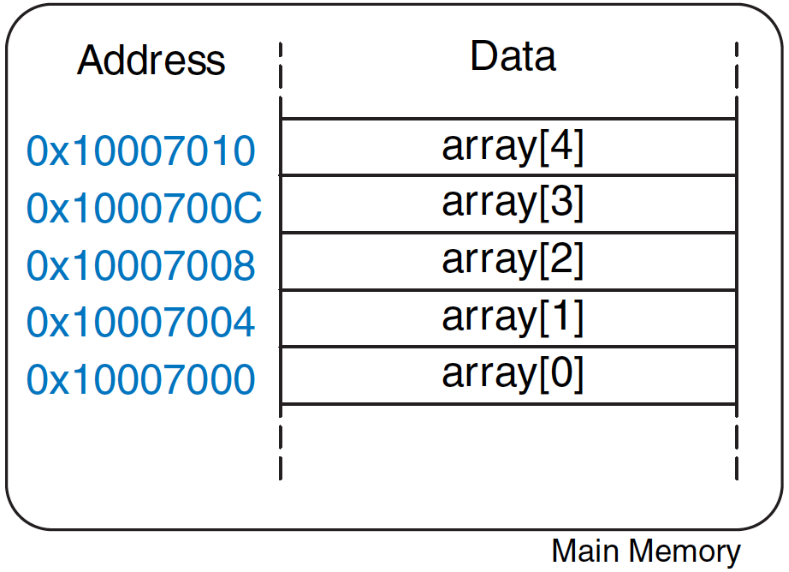
We will talk later about data section, which is where MIPS allows you to refer to memory addresses by name – just like a variable!

High-Level Code	**MIPS Assembly Code**

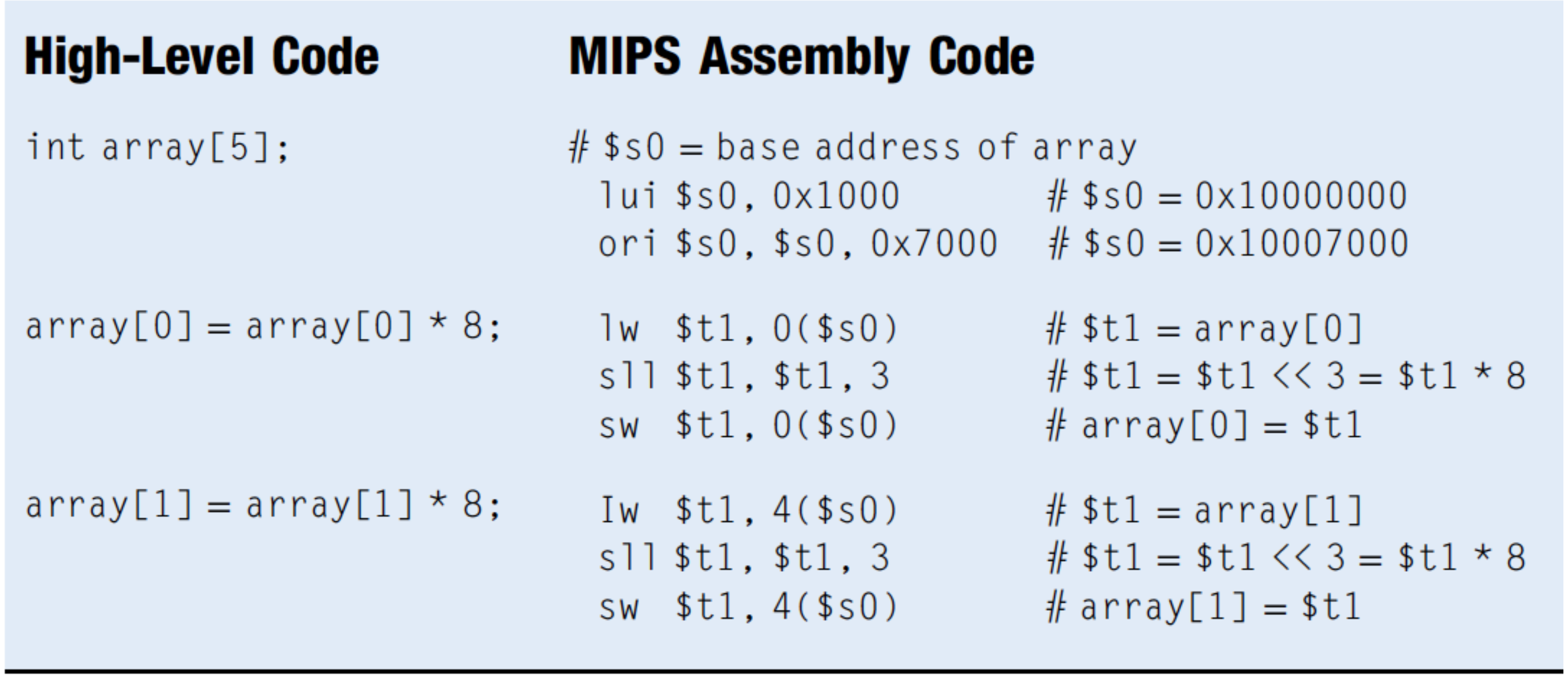
`int array[5];`	`# $s0 = base address of array`
	`  lui $s0, 0x1000            # $s0 = 0x10000000`
	`  ori $s0, $s0, 0x7000      # $s0 = 0x10007000`

`array[0] = array[0] * 8;`	`  lw  $t1, 0($s0)            # $t1 = array[0]`
	`  sll $t1, $t1, 3            # $t1 = $t1 << 3 = $t1 * 8`
	`  sw  $t1, 0($s0)            # array[0] = $t1`

`array[1] = array[1] * 8;`	



Address	Data
0x10007010	array[4]
0x1000700C	array[3]
0x10007008	array[2]
0x10007004	array[1]
0x10007000	array[0]

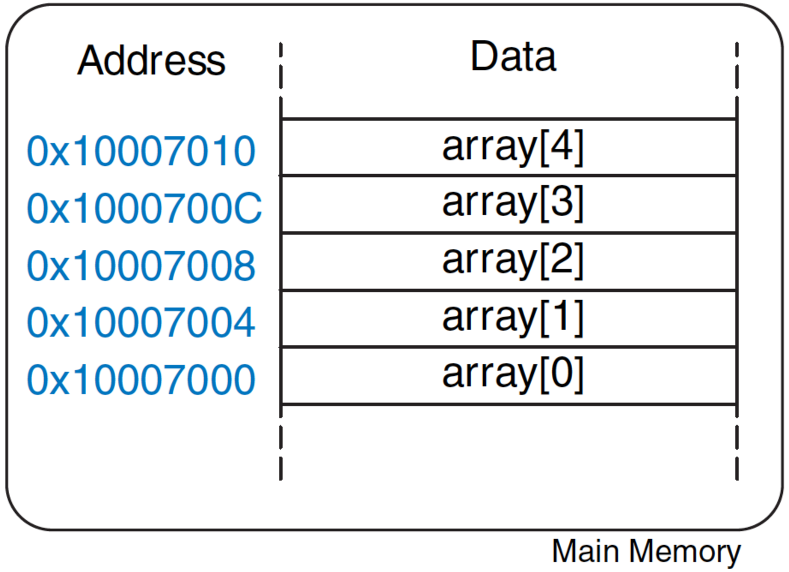


High-Level Code **MIPS Assembly Code**

```
int array[5];               # $s0 = base address of array
                              lui $s0, 0x1000           # $s0 = 0x10000000
                              ori $s0, $s0, 0x7000      # $s0 = 0x10007000

array[0] = array[0] * 8;      lw  $t1, 0($s0)           # $t1 = array[0]
                              sll $t1, $t1, 3           # $t1 = $t1 << 3 = $t1 * 8
                              sw  $t1, 0($s0)           # array[0] = $t1

array[1] = array[1] * 8;      Iw  $t1, 4($s0)           # $t1 = array[1]
                              sll $t1, $t1, 3           # $t1 = $t1 << 3 = $t1 * 8
                              sw  $t1, 4($s0)           # array[1] = $t1
```



Address	Data
0x10007010	array[4]
0x1000700C	array[3]
0x10007008	array[2]
0x10007004	array[1]
0x10007000	array[0]