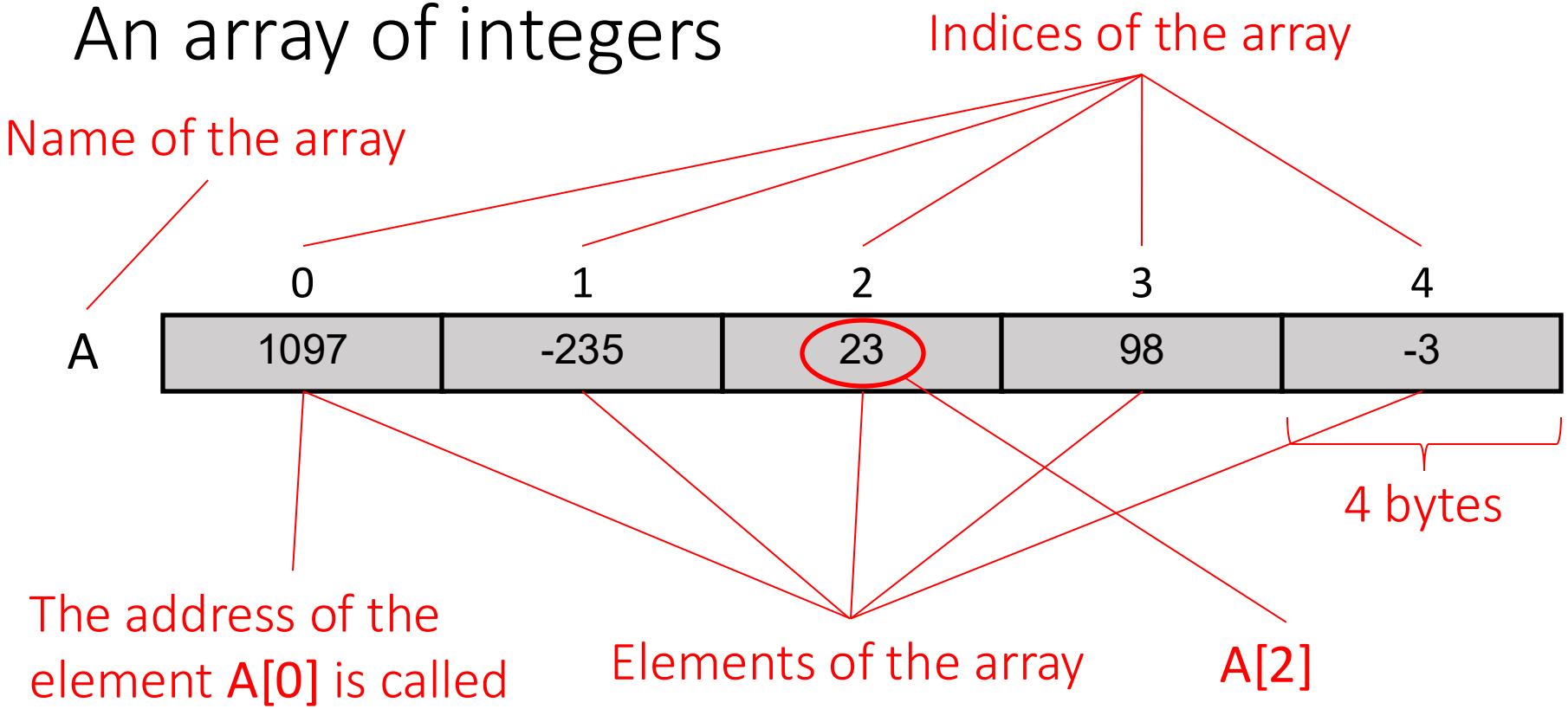
Topic V08

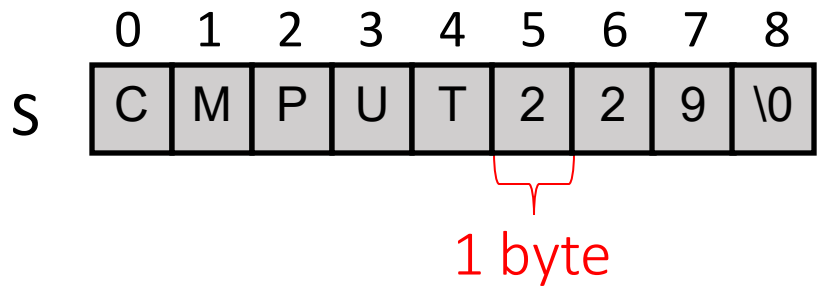
Storing Arrays in Memory

Reading: (Section 2.13)

An array of integers



An array of characters (strings)



C code:

```
x = A[i];
```

RISC-V code:

Assumption

x ↔ s3

i ↔ t0

base of A[] ↔ s6

A is an array of 32-bit integers

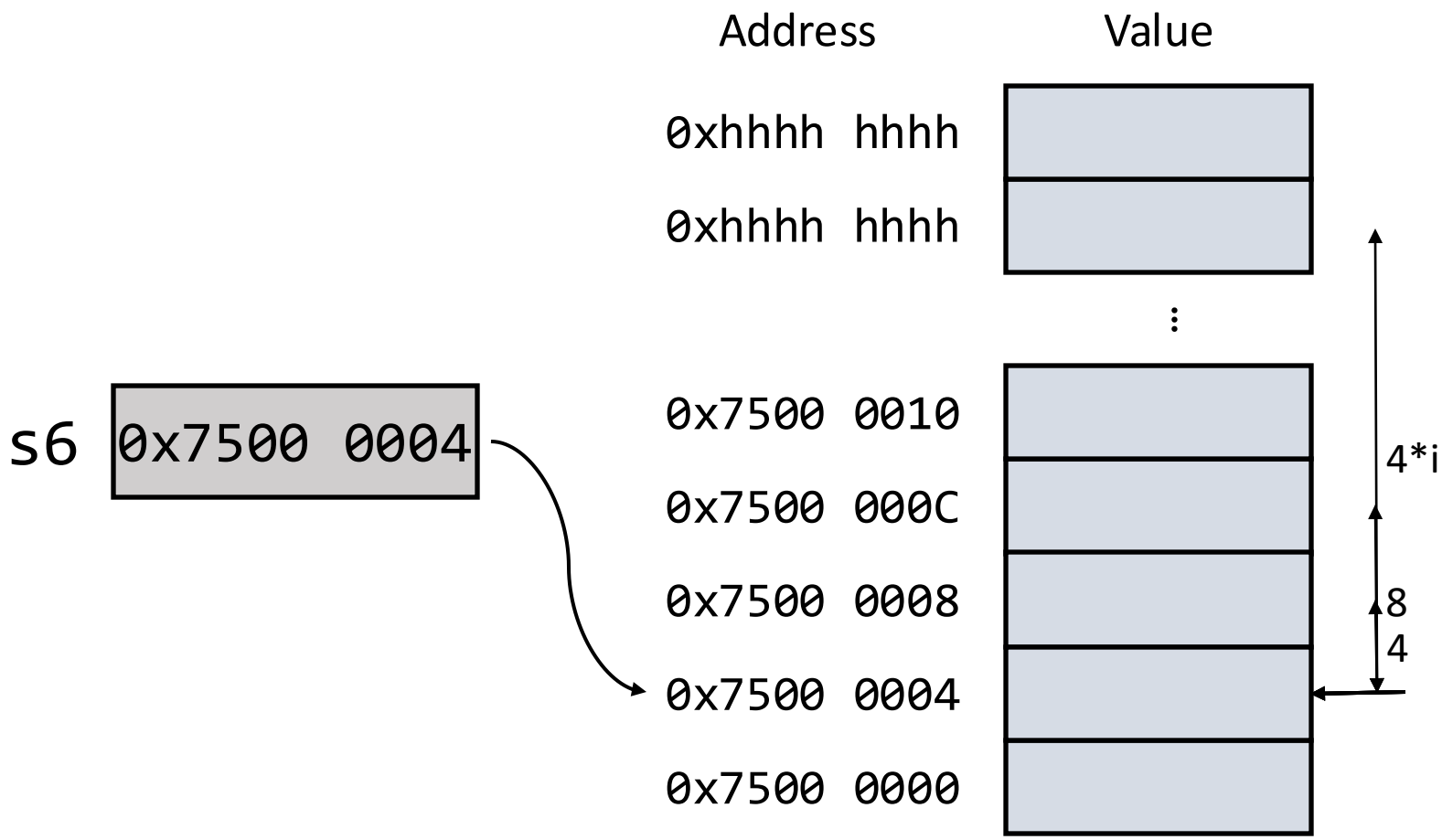
Assumption

x ↔ s3
i ↔ t0
base of A[] ↔ s6
A is an array of 32-bit integers

Where is A[0]?
At the address in s6
Where is A[1]?
At the address in s6 + 4
Where is A[2]?
At the address in s6 + 8
Where is A[i]?
At the address in s6 + 4 * i

How do we get the value of A[i] into s3?

```
tA ← 4 * i
tB ← s6 + tA
s3 ← Mem[tB]
```



C code:

```
x = A[i];
```

```
tA ← 4 * i
tB ← s6 + tA
x ← Mem[tB]
```

Assumption

x ↔ s3
i ↔ t0
base of A[] ↔ s6
A is an array of 32-bit integers

RISC-V code:

```
slli  t1, t0, 2        # t1 ← 4 * i
add   t2, s6, t1       # t2 ← A + 4 * i
lw    s3, 0(t2)        # x ← A[i]
```

C code:

```
A[i] = x;
```

```
tA ← 4 * i
tB ← s6 + tA
Mem[tB] ← x
```

Assumption

x ↔ s3
i ↔ t0
base of A[] ↔ s6
A is an array of 32-bit integers

RISC-V code:

```
slli  t1, t0, 2       # t1 ← 4 * i
add   t2, s6, t1      # t2 ← A + 4 * i
sw    s3, 0(t2)       # A[i] ← x
```

C code:

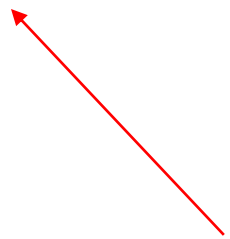
```
c = name[k];
```

Assumption

c ↔ s0
k ↔ t0
base of name[] ↔ s4
name is an array of 8-bit characters
8 bits = 1 byte

RISC-V code:

```
add   t1, s4, t0      # t1 ← name + k
lbu   s0, 0(t1)       # x ← name[k]
```



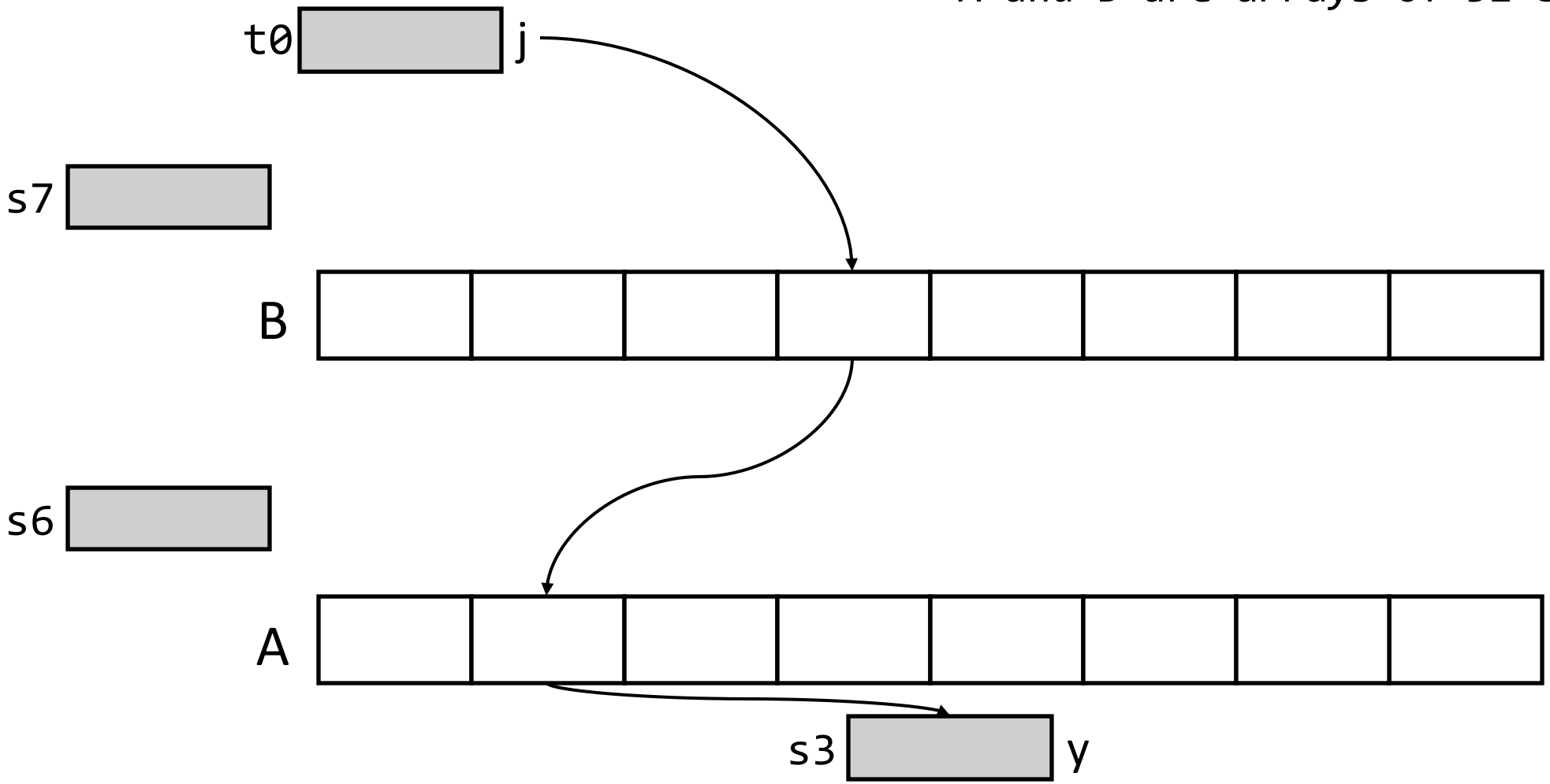
Load byte unsigned zero-extends the byte loaded from memory before writing into s0

C code:

```
y = A[B[j]];
```

Assumption

y ↔ s3
j ↔ t0
base of A[] ↔ s6
base of B[] ↔ s7
A and B are arrays of 32-bit integers

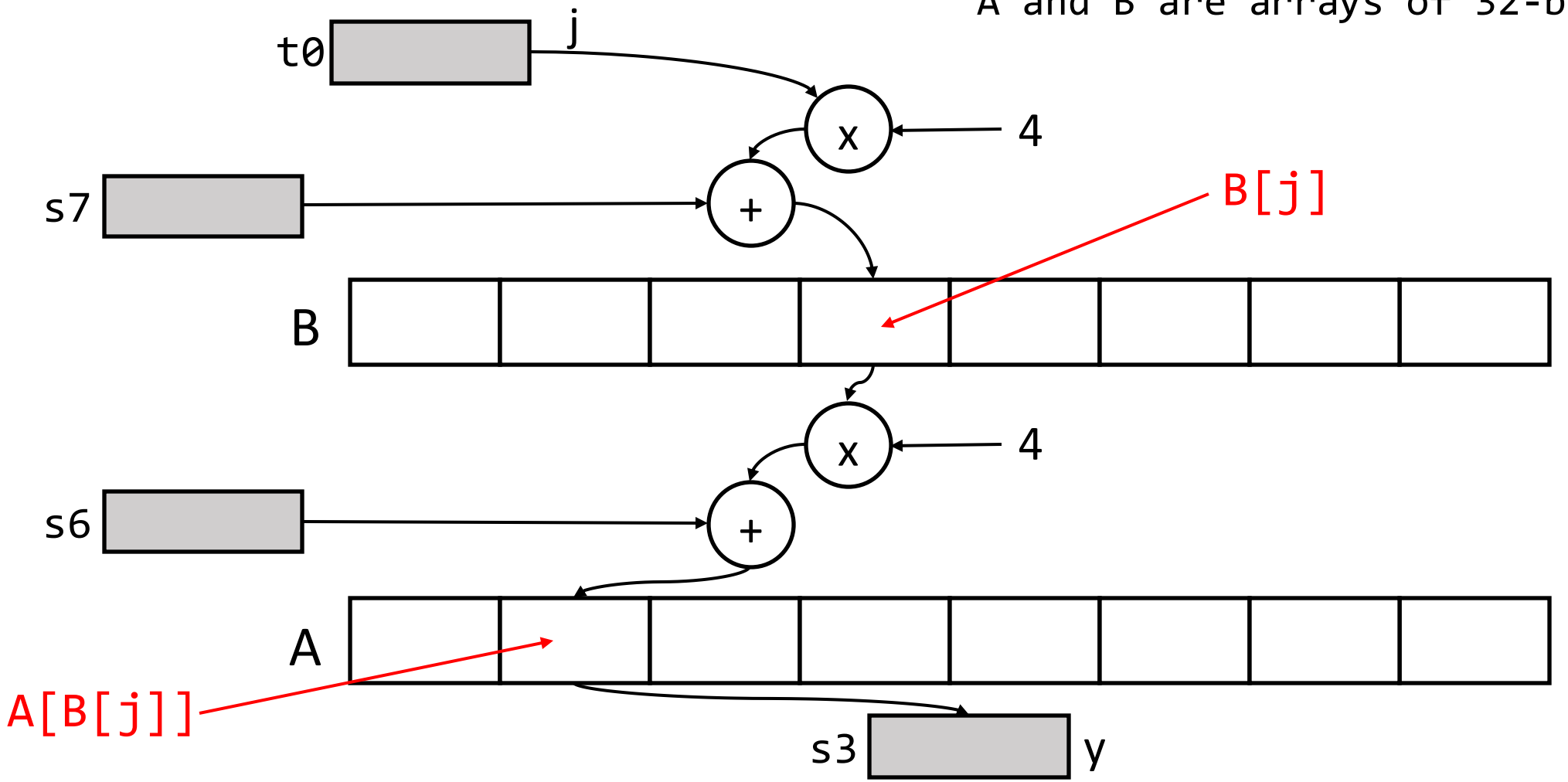


C code:

```
y = A[B[j]];
```

Assumption

y ↔ s3
j ↔ t0
base of A[] ↔ s6
base of B[] ↔ s7
A and B are arrays of 32-bit integers

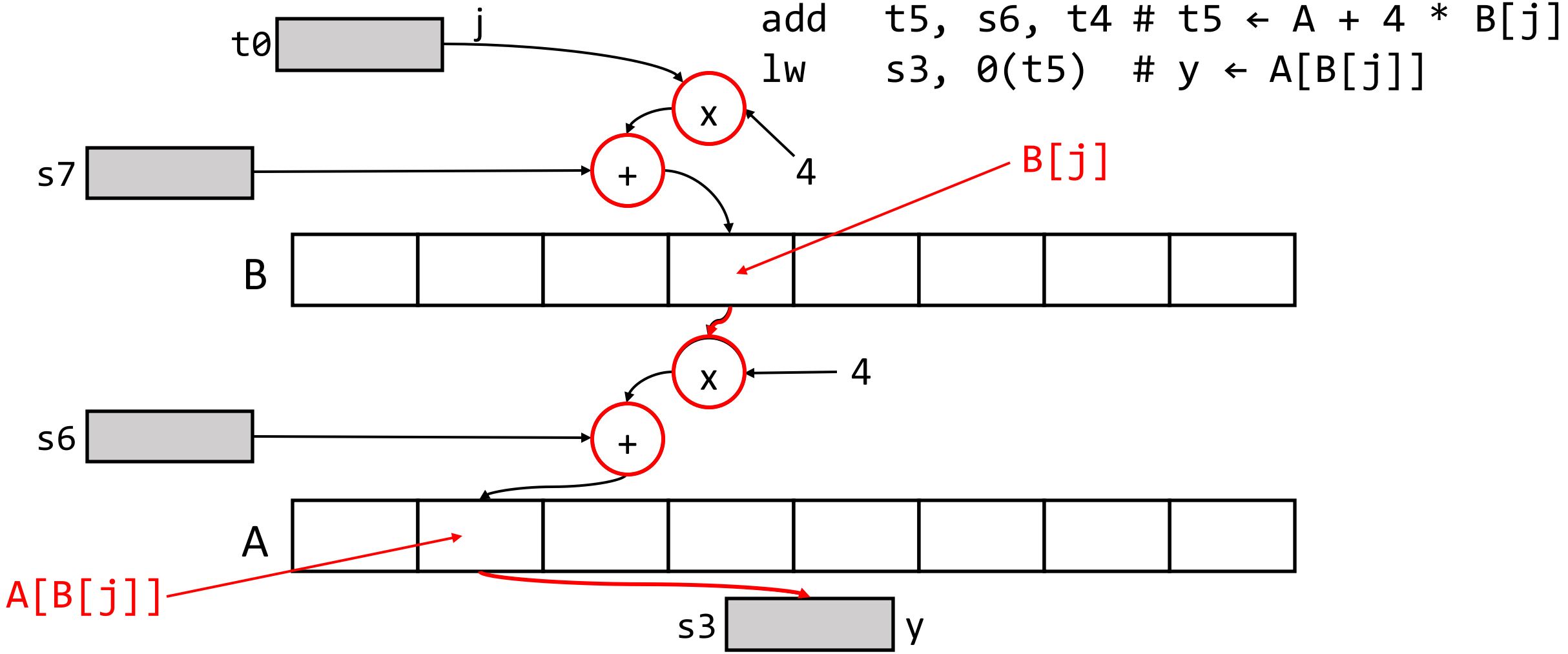


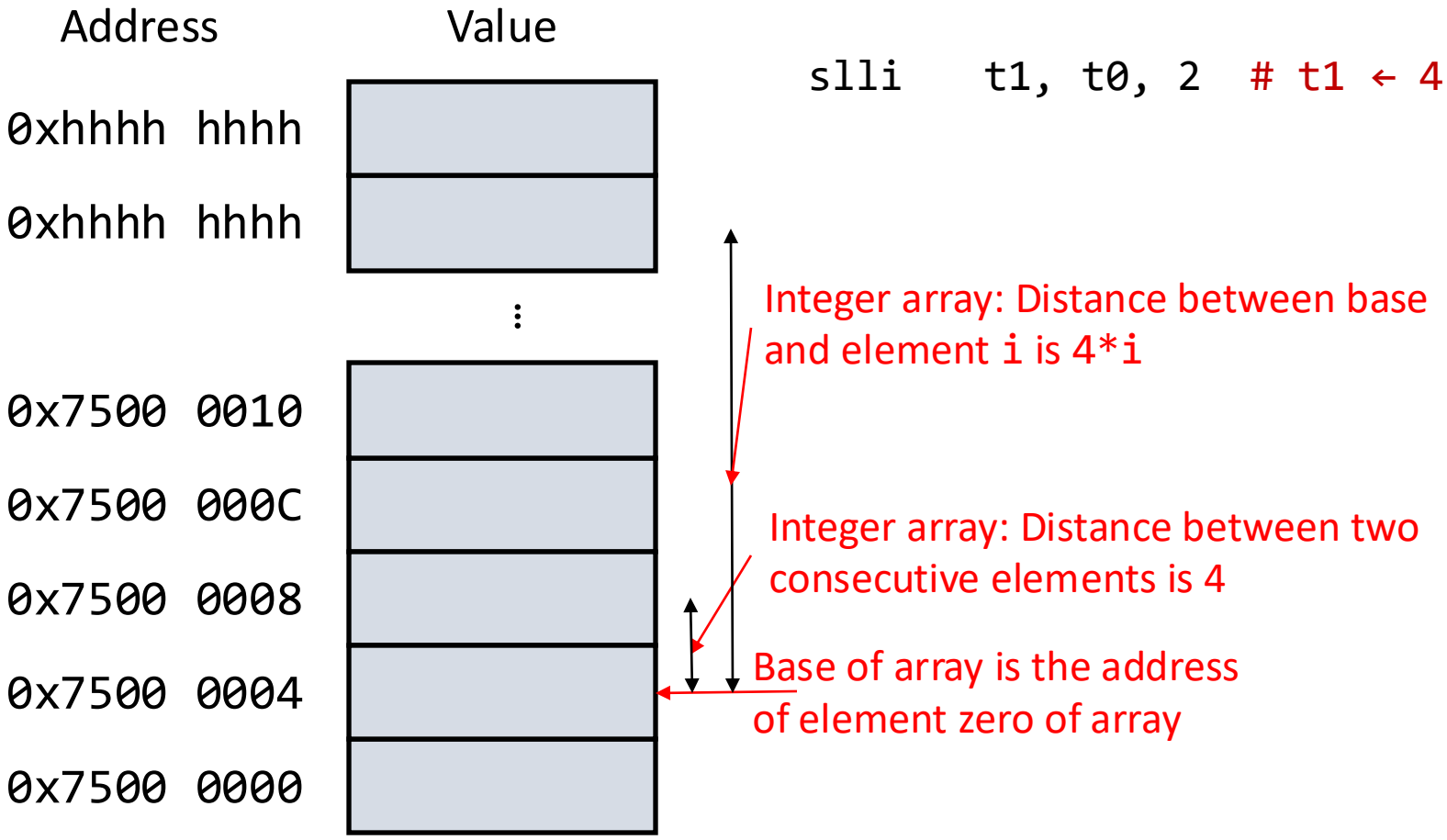
C code:

```
y = A[B[j]];
```

RISC-V code:

```
slli  t1, t0, 2  # t1 ← 4 * j
add   t2, s7, t1 # t2 ← B + 4 * j
lw    t3, 0(t2)  # t3 ← B[j]
slli  t4, t3, 2  # t4 ← 4 * B[j]
add   t5, s6, t4 # t5 ← A + 4 * B[j]
lw    s3, 0(t5)  # y ← A[B[j]]
```






```
slli    t1, t0, 2  # t1 ← 4 * i
```

Integer array: Distance between base and element `i` is `4*i`

Integer array: Distance between two consecutive elements is 4

Base of array is the address of element zero of array

Each integer is four bytes

Recap

Each character is one byte

C code:

```
char *name;
c = name[k];
```

No need to multiply by four to find element `name[k]`