

CS-200

Computer Architecture

Part 1d. Instruction Set Architecture

Arrays and Data Structures

Paolo Ienne

`<paolo.ienne@epfl.ch>`

Arrays in High-Level Languages

- Java:

```
short[] myData = {10407, -16533, -22715, 29796, 18956,...};
```

- Python:

```
myData = [10407, -16533, -22715, 29796, 18956,...]
```

or

```
import array
```

```
myData = array.array('h', [10407, -16533, -22715, 29796, 18956,...])
```

- Scala:

```
val myData: Array[Short] = Array(10407, -16533, -22715, 29796, 18956,...)
```

Different Ways to Store Arrays

A

myData →

10407
-16533
-22715
29796
18956
-882
-27277
16066
-30868
0

16-bit

B

myData →

9
10407
-16533
-22715
29796
18956
-882
-27277
16066
-30868

16-bit

C

myData →

10407
-16533
-22715
29796
18956
-882
-27277
16066
-30868

16-bit

Adding Positive Elements

- Add all positive elements in an array of signed 16-bit integers
 - At call time, `a0` points to the array (and, in type **C**, `a1` is the length)
 - At return time, `a0` contains the result
- Write it three times, for arrays of type **A**, **B**, and **C**

Type C

```
short add_positive(short myData[], int N) {
    short sum = 0;
    for (int i = 0; i < N; i++) {
        if (myData[i] > 0) {
            sum += myData[i];
        }
    }
    return sum;
}
```

A

myData →

10407
-16533
-22715
29796
18956
-882
-27277
16066
-30868
0

16-bit

Adding Positive Elements (Type A)

add_positive:

```
li    t0, 0                # t0 will hold the sum (initialized to 0)
```

next_short:

```
lh    t1, 0(a0)           # Load short (half-word) at address a0 into t1
beqz  t1, end              # If t1 is 0 (null short), we are done
bltz  t1, negative         # If t1 is negative, ignore
add   t0, t0, t1           # Add t1 to the sum (t0)
```

negative:

```
addi  a0, a0, 2           # Move array pointer (a0) by sizeof(short) to the next element
j     next_short           # Repeat the loop
```

end:

```
mv    a0, t0              # Move the sum (t0) into a0 as the return value
ret
```

A

myData →

10407
-16533
-22715
29796
18956
-882
-27277
16066
-30868
0

← 16-bit →

B

myData →

9
10407
-16533
-22715
29796
18956
-882
-27277
16066
-30868

16-bit

Adding Positive Elements (Type B)

add_positive:

```
li    t0, 0           # t0 will hold the sum (initialized to 0)
lh    t1, 0(a0)       # t1 will count the elements to process
add   a0, a0, 2       # Move array pointer (a0) to the first real element
```

next_short:

```
beqz  t1, end         # If t1 is 0 (no more elements), we are done
lh    t2, 0(a0)       # Load short (half-word) at address a0 into t2
bltz  t2, negative    # If t2 is negative, ignore
add   t0, t0, t2      # Add t2 to the sum (t0)
```

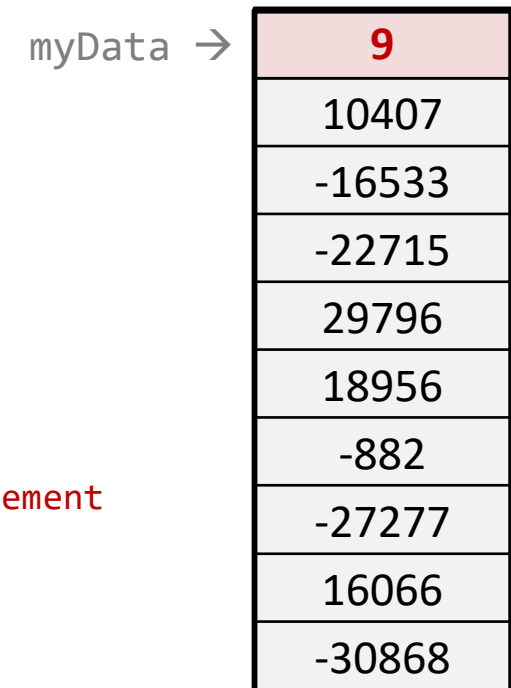
negative:

```
addi  a0, a0, 2       # Move array pointer (a0) by sizeof(short) to the next element
addi  t1, t1, -1      # Decrement the counter of elements to process
j     next_short      # Repeat the loop
```

end:

```
mv    a0, t0          # Move the sum (t0) into a0 as the return value
ret
```

B



← 16-bit →

Adding Positive Elements (Type C)

add_positive:

```
li    t0, 0           # t0 will hold the sum (initialized to 0)
mv    t1, a1         # t1 will count the elements to process (a1)
```

next_short:

```
beqz  t1, end        # If t1 is 0 (no more elements), we are done
lh    t2, 0(a0)      # Load short (half-word) at address a0 into t2
bltz  t2, negative   # If t2 is negative, ignore
add   t0, t0, t2     # Add t2 to the sum (t0)
```

negative:

```
addi  a0, a0, 2      # Move array pointer (a0) by sizeof(short) to the next element
addi  t1, t1, -1     # Decrement the counter of elements to process
j     next_short     # Repeat the loop
```

end:

```
mv    a0, t0         # Move the sum (t0) into a0 as the return value
ret
```

C

myData →

10407
-16533
-22715
29796
18956
-882
-27277
16066
-30868

← 16-bit →

Adding Positive Elements (Variation on C)

- Add all positive elements in an array of signed 16-bit integers
 - At call time, `a0` points to the array and `a1` is the length of the array
 - At return time, `a0` contains the result
- Write it by incrementing the index of the array

```
int i = 0;
while (i < N) {
    if (myData[i] > 0) {
        ...
    }
    i++;
}
```

=

```
int i;
for (i = 0; i < N; i++) {
    if (myData[i] > 0) {
        ...
    }
}
```

C

myData →

10407
-16533
-22715
29796
18956
-882
-27277
16066
-30868

↔
16-bit

Adding Positive Elements (Type C; index)

add_positive:

```
li    t0, 0           # t0 will hold the sum (initialized to 0)
li    t1, 0           # t1 will hold the array index (initialized to 0)
```

next_index:

```
bge   t1, a1, end     # If index (t1) ≥ # elements (a1), we are done
slli  t2, t1, 1        # t2 = offset of the element as index (t1) * sizeof(short)
add   t2, a0, t2       # Address of the element = myData (a0) + offset (t2)
lh    t3, 0(t2)       # Load short (half-word) at address a0 into t3
bltz  t3, negative     # If t3 is negative, ignore
add   t0, t0, t3       # Add t3 to the sum (t0)
```

negative:

```
addi  t1, t1, 1       # Decrement the counter of elements to process
j     next_index      # Repeat the loop
```

end:

```
mv    a0, t0          # Move the sum (t0) into a0 as the return value
ret                               # Return to caller
```

C

myData →

10407
-16533
-22715
29796
18956
-882
-27277
16066
-30868

← 16-bit →

Which is Better?

Pointer to memory

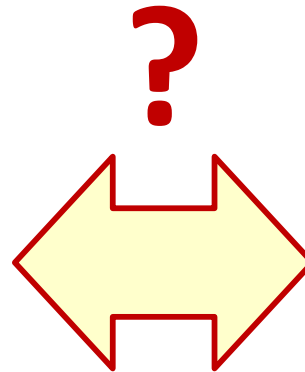
```
add_positive:
    li    t0, 0
    mv    t1, a1

next_short:
    beq   t1, zero, end
    lh    t2, 0(a0)
    bltz  t2, negative
    add   t0, t0, t2

negative:
    addi  a0, a0, 2
    addi  t1, t1, -1
    j     next_short

end:
    mv    a0, t0
    ret
```

Less instructions → Faster



Index in array

```
add_positive:
    li    t0, 0
    li    t1, 0

next_index:
    bge   t1, a1, end
    slli  t2, t1, 1
    add   t2, a0, t2
    lh    t3, 0(t2)
    bltz  t3, negative
    add   t0, t0, t3

negative:
    addi  t1, t1, 1
    j     next_index

end:
    mv    a0, t0
    ret
```

Which is Better?

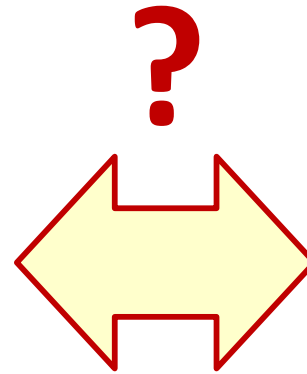
Pointer to memory

```
short sum = 0;
short *ptr = myData;
short *end = myData + N;

while (ptr < end) {
    if (*ptr > 0) {
        sum += *ptr;
    }
    ptr++;
}
```

A pointer to **short** integers
= address of **myData**

Advances the **pointer** instead of recomputing the element address



Index in array

```
short sum = 0;
int i;
for (i = 0; i < N; i++) {
    if (myData[i] > 0) {
        sum += myData[i];
    }
}
```

Better code

We Need Good Compilers!

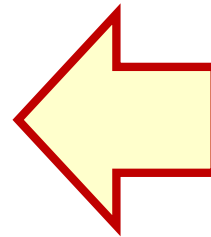
Pointer to memory

```
add_positive:
    li    t0, 0
    mv    t1, a1

next_short:
    beq   t1, zero, end
    lh    t2, 0(a0)
    bltz  t2, negative
    add   t0, t0, t2

negative:
    addi  a0, a0, 2
    addi  t1, t1, -1
    j     next_short

end:
    mv    a0, t0
    ret
```

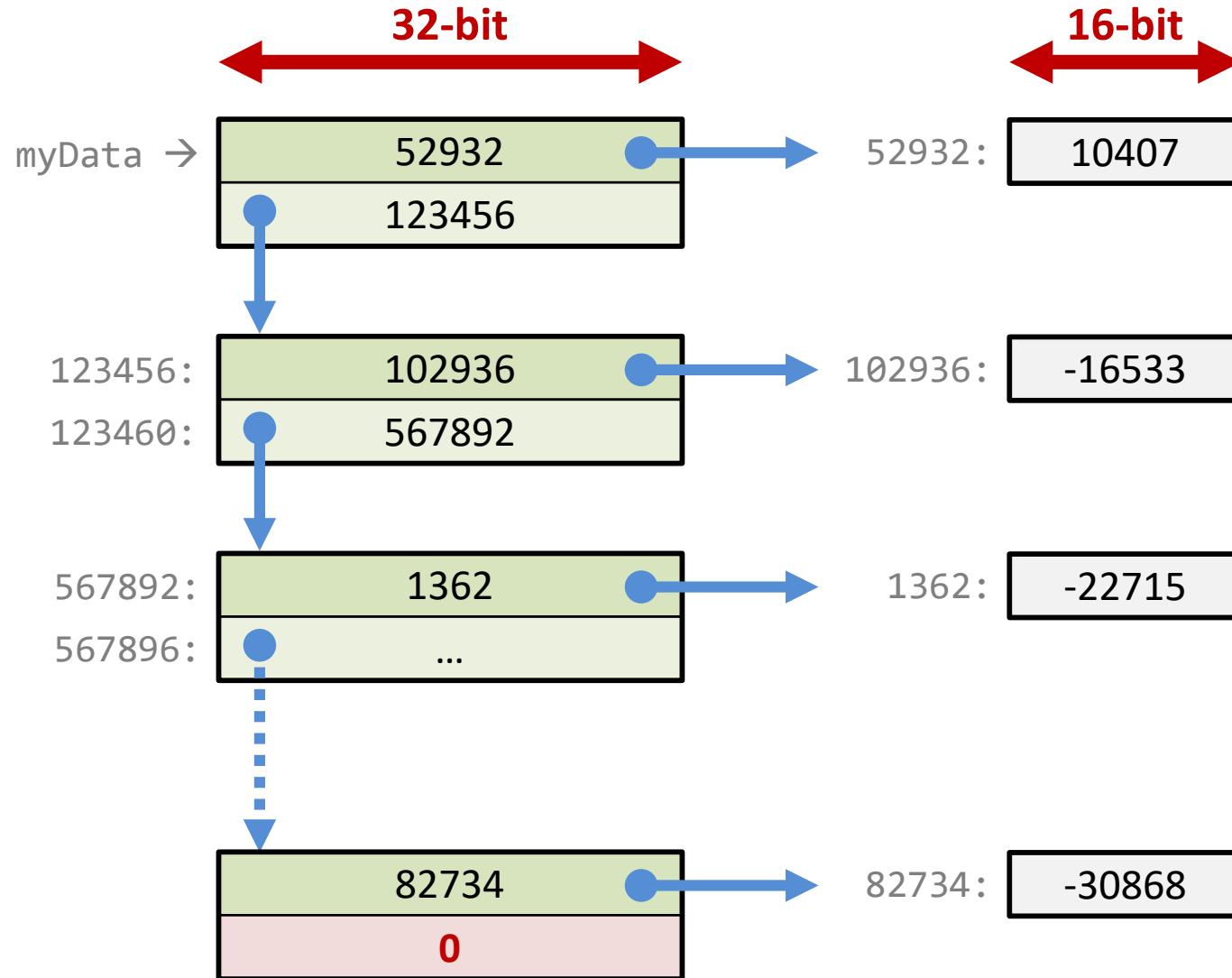


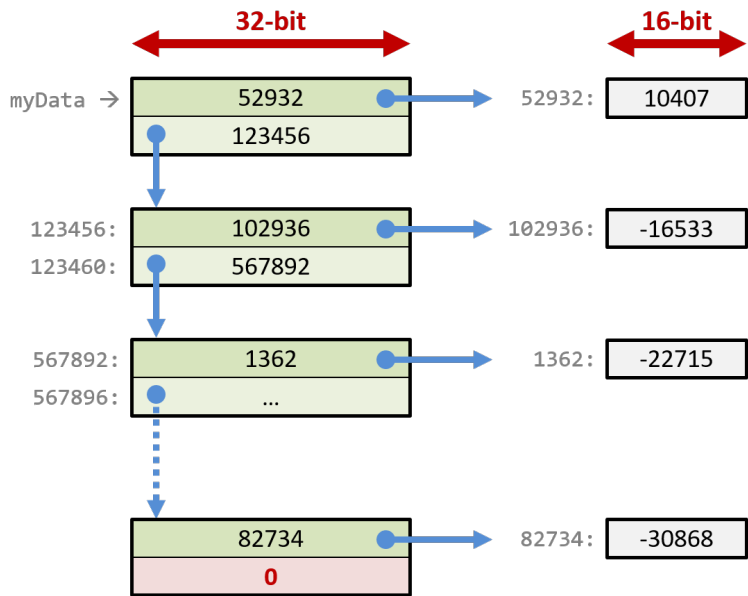
Index in array

```
short sum = 0;
int i;
for (i = 0; i < N; i++) {
    if (myData[i] > 0) {
        sum += myData[i];
    }
}
```

Write **better code** but get the **best performance!**

One More Way to Store Arrays





Adding Positive Elements (Linked List)

add_positive:

```
li    t0, 0           # t0 will hold the sum (initialized to 0)
```

next_element:

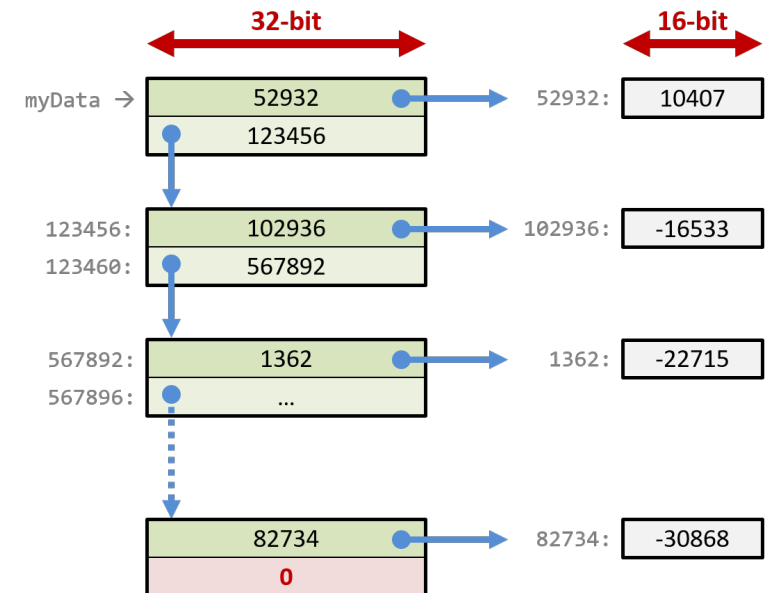
```
beqz  a0, end        # If address of next element (a0) is zero, we are done
lw    t1, 0(a0)      # Load address of actual data into t1
lh    t1, 0(t1)      # Load short (half-word) at address t1 into t1
bltz  t1, negative   # If t1 is negative, ignore
add   t0, t0, t1     # Add t1 to the sum (t0)
```

negative:

```
lw    a0, 4(a0)      # Load address of next element into a0
j     next_index     # Repeat the loop
```

end:

```
mv    a0, t0         # Move the sum (t0) into a0 as the return value
ret
```



Not Much More Complex...

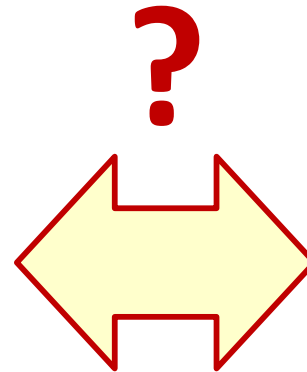
Pointer to memory

```
add_positive:
    li    t0, 0
    mv    t1, a1

next_short:
    beq   t1, zero, end
    lh    t2, 0(a0)
    bltz  t2, negative
    add   t0, t0, t2

negative:
    addi  a0, a0, 2
    addi  t1, t1, -1
    j     next_short

end:
    mv    a0, t0
    ret
```



Linked list

```
add_positive:
    li    t0, 0

next_element:
    beqz  a0, end
    lw    t1, 0(a0)
    lh    t1, 0(t1)
    bltz  t1, negative
    add   t0, t0, t1

negative:
    lw    a0, 4(a0)
    j     next_index

end:
    mv    a0, t0
    ret
```

Multiple noncontiguous loads

References

- Patterson & Hennessy, COD – RISC-V Edition
 - **Chapter 2** and, in particular, **Sections 2.9, 2.13, and 2.14**