Array layout

Elements of equal size sequentially in memory

Array addressing

Can arrays always be stored "packed"?

What happens if array bounds are violated?

And who is checking that?
Array addressing via element pointer

\[ \text{Rd} - \text{Destination} \]

\[ \text{Rp} - \text{Array pointer} \]

\[ \text{Element offset} \]

\[ \text{Write-back} \]

\[ \text{lset}<c><q> <Rd>, [<Rp>, #/-<offset>] \]

Array addressing via index register

\[ \text{Rb} - \text{Base address} \]

\[ \text{Ri} - \text{Index [shifted]} \]

\[ \text{Rd} - \text{Destination} \]

\[ \text{lset}<c><q> <Rd>, [<Rb>, <Ri> {, lsl <shift>}] \]
Arbitrary array indexing

```
Data Structures

Arbitrary array indexing
; r0 base address for array a
; r1 from array index
; r2 to array index

mov r3, #0
mov r4, #4
mov r5, #1

for_sum:
    cmp r1, r2
    bgt end_for_sum
    mla r7, [r0, r5, lsl #2]
    add r3, r7
    add r1, #1
    b for_sum

end_for_sum:
    mov r0, r3
    ; r0 sum over all a [from .. to]
```

Zero-based array indexing

```
Data Structures

Zero-based array indexing
; r0 base address for array a
; r1 from array index
; r2 to array index

mov r3, #0
mov r4, #4

for_sum:
    cmp r1, r2
    bgt end_for_sum
    mul r5, r1, r4
    add r3, r5
    mov r0, r3
    ; r0 sum over all a [from .. to]
```

Calculate $\sum x_i$

```
Data Structures

Calculate $\sum x_i$

```

```
; r0 base address for array a
; r1 from array index
; r2 to array index

mov r3, #0
mov r4, #4

for_sum:
    cmp r1, r2
    bgt end_for_sum
    mla r6, [r0, r1, lsl #2]
    add r3, r6
    add r1, #1
    b for_sum

end_for_sum:
    mov r0, r3
```

```
; r0 base address for array a
; r1 from array index
; r2 to array index

mov r3, #0
mov r4, #4

for_sum:
    cmp r1, r2
    bgt end_for_sum
    mul r5, r1, r4
    add r3, r5
    add r1, #1
    b for_sum

end_for_sum:
    mov r0, r3
```

Calculate $\sum x_i$

```
; r0 base address for array a
; r1 from array index
; r2 to array index

mov r3, #0
mov r4, #4

for_sum:
    cmp r1, r2
    bgt end_for_sum
    mla r6, [r0, r1, lsl #2]
    add r3, r6
    add r1, #1
    b for_sum

end_for_sum:
    mov r0, r3
```

```
; r0 base address for array a
; r1 from array index
; r2 to array index

mov r3, #0
mov r4, #4

for_sum:
    cmp r1, r2
    bgt end_for_sum
    mul r5, r1, r4
    add r3, r5
    add r1, #1
    b for_sum

end_for_sum:
    mov r0, r3
```

```
; r0 base address for array a
; r1 from array index
; r2 to array index

mov r3, #0
mov r4, #4

for_sum:
    cmp r1, r2
    bgt end_for_sum
    mul r5, r1, r4
    add r3, r5
    add r1, #1
    b for_sum

end_for_sum:
    mov r0, r3
```

```
; r0 base address for array a
; r1 from array index
; r2 to array index

mov r3, #0
mov r4, #4

for_sum:
    cmp r1, r2
    bgt end_for_sum
    mul r5, r1, r4
    add r3, r5
    add r1, #1
    b for_sum

end_for_sum:
    mov r0, r3
```

```
; r0 base address for array a
; r1 from array index
; r2 to array index

mov r3, #0
mov r4, #4

for_sum:
    cmp r1, r2
    bgt end_for_sum
    mul r5, r1, r4
    add r3, r5
    add r1, #1
    b for_sum

end_for_sum:
    mov r0, r3
```

```
; r0 base address for array a
; r1 from array index
; r2 to array index

mov r3, #0
mov r4, #4

for_sum:
    cmp r1, r2
    bgt end_for_sum
    mul r5, r1, r4
    add r3, r5
    add r1, #1
    b for_sum

end_for_sum:
    mov r0, r3
```

```
; r0 base address for array a
; r1 from array index
; r2 to array index

mov r3, #0
mov r4, #4

for_sum:
    cmp r1, r2
    bgt end_for_sum
    mul r5, r1, r4
    add r3, r5
    add r1, #1
    b for_sum

end_for_sum:
    mov r0, r3
```

```
; r0 base address for array a
; r1 from array index
; r2 to array index

mov r3, #0
mov r4, #4

for_sum:
    cmp r1, r2
    bgt end_for_sum
    mul r5, r1, r4
    add r3, r5
    add r1, #1
    b for_sum

end_for_sum:
    mov r0, r3
```

```
; r0 base address for array a
; r1 from array index
; r2 to array index

mov r3, #0
mov r4, #4

for_sum:
    cmp r1, r2
    bgt end_for_sum
    mul r5, r1, r4
    add r3, r5
    add r1, #1
    b for_sum

end_for_sum:
    mov r0, r3
```

```
; r0 base address for array a
; r1 from array index
; r2 to array index

mov r3, #0
mov r4, #4

for_sum:
    cmp r1, r2
    bgt end_for_sum
    mul r5, r1, r4
    add r3, r5
    add r1, #1
    b for_sum

end_for_sum:
    mov r0, r3
```
**Data Structures**

**Replacing multiplication with shifted index register**

; \( r0 \) base address for array \( a \)
; \( r1 \) from array index
; \( r2 \) to array index

\[
\text{mov } r3, w0 \quad ; \text{sum} := 0
\]

\[
\text{for_sum: }
\]

\[
\text{cmp } r1, r2 \quad ; i > to
\]

\[
\text{bgt } \text{end_for_sum}
\]

\[
\text{ldr } r4, [r0, r1, lsl \#2] \quad ; a[i] := [base \times \text{element offset}]
\]

\[
\text{add } r3, r4 \quad ; \text{sum} := \text{sum} + a[i]
\]

\[
\text{add } r1, \#1 \quad ; i := i + 1
\]

\[
\text{b } \text{for_sum}
\]

\[
\text{end_for_sum: }
\]

\[
\text{mov } r0, r3 \quad ; r0 \text{ sum over all } a[\text{from .. to}]
\]

**Replacing multiplication with shifted index register**

\[
\text{mov } r3, w0 \quad ; \text{sum} := 0
\]

\[
\text{for_sum: }
\]

\[
\text{cmp } r1, r2 \quad ; i > to
\]

\[
\text{bgt } \text{end_for_sum}
\]

\[
\text{ldr } r4, [r0, r1, lsl \#2] \quad ; a[i] := [base \times \text{element offset}]
\]

\[
\text{add } r3, r4 \quad ; \text{sum} := \text{sum} + a[i]
\]

\[
\text{add } r1, \#1 \quad ; i := i + 1
\]

\[
\text{b } \text{for_sum}
\]

\[
\text{end_for_sum: }
\]

\[
\text{mov } r0, r3 \quad ; r0 \text{ sum over all } a[\text{from .. to}]
\]

**Replacing offsets with addresses**

; \( r0 \) base address for array \( a \)
; \( r1 \) from array index
; \( r2 \) to array index

\[
\text{lsl } r1, r1, \#2 \quad ; \text{translate from index to offset}
\]

\[
\text{add } r3, r4 \quad ; \text{offset} := \text{offset} + 4
\]

\[
\text{mov } r0, w0 \quad ; \text{sum} := 0
\]

\[
\text{for_sum: }
\]

\[
\text{cmp } r1, r2 \quad ; i > to
\]

\[
\text{bgt } \text{end_for_sum}
\]

\[
\text{ldr } r4, [r0, r1] \quad ; a[i] := [base + \text{offset}]
\]

\[
\text{add } r3, r4 \quad ; \text{sum} := \text{sum} + a[i]
\]

\[
\text{add } r1, \#4 \quad ; \text{offset} := \text{offset} + 4
\]

\[
\text{b } \text{for_sum}
\]

\[
\text{end_for_sum: }
\]

\[
\text{mov } r0, r3 \quad ; r0 \text{ sum over all } a[\text{from .. to}]
\]

**Replacing offsets with addresses**

; \( r0 \) base address for array \( a \)
; \( r1 \) from array index
; \( r2 \) to array index

\[
\text{lsl } r1, r1, \#2 \quad ; \text{translate from index to offset}
\]

\[
\text{add } r1, \#4 \quad ; \text{offset} := \text{offset} + 4
\]

\[
\text{mov } r0, w0 \quad ; \text{sum} := 0
\]

\[
\text{for_sum: }
\]

\[
\text{cmp } r1, r2 \quad ; i > to
\]

\[
\text{bgt } \text{end_for_sum}
\]

\[
\text{ldr } r4, [r0, r1] \quad ; a[i] := [i_addr + \text{offset}]
\]

\[
\text{add } r3, r4 \quad ; \text{sum} := \text{sum} + a[i]
\]

\[
\text{add } r1, \#4 \quad ; \text{offset} := \text{offset} + 4
\]

\[
\text{b } \text{for_sum}
\]

\[
\text{end_for_sum: }
\]

\[
\text{mov } r0, r3 \quad ; r0 \text{ sum over all } a[\text{from .. to}]
\]
Array Slices

numbers  = [0, 1, 2, 3, 4, 5]
numbersSlice = numbers [1:3]
# numbersSlice equals [1, 2, 3]

numbers  := []int {0, 1, 2, 3, 4, 5}
numbersSlice := numbers [1:3]

type Naturals is array (Integer range <>) of Natural;
Numbers         : constant Naturals (-50 .. 50) := (others => Random (Generator));
Numbers_Slice_1 := Numbers (-10 .. 10);
Numbers_Slice_2 := Numbers (11 .. 20);
Numbers_Slice_3 := Numbers (-20 .. 50);

begin
  for n of Numbers_Slice_3 loop
    n := n + 1;
  end loop;
end;

Copy array slice

; r0 base address for array a
; r1 from array index
; r2 to array index >= from index
; r3 base address for array b

lsl r1, r1, #2 ; translate from index to offset
lsl r2, r2, #2 ; translate to index to offset
add r1, r0 ; translate from index to address -> i_addr
add r2, r0 ; translate to index to address -> to_addr
for_copy:
  ldr r4, [r1], #4
  str r4, [r3], #4
  cmp r1, r2
  bne for_copy
  ; a [i] := [i_addr]; i_addr += 4
  ; [j_addr] := a [i]; j_addr += 4
  ; i_addr <= to_addr
end_for_copy:

; b [i] := a [from .. to]

Are those copy or reference affairs?

Moving blocks of memory can be done even/much faster with special hardware.
DMA controllers
Summary

Arrays

• Structure
• Alignment
• Addressing
• Iterators
• Copy procedures