Lecture 8
Data structures

Computing platforms

Novosibirsk State University
University of Hertfordshire

D. Irtegov, A.Shafarenko

2018

Data structures in CdM-8 assembly

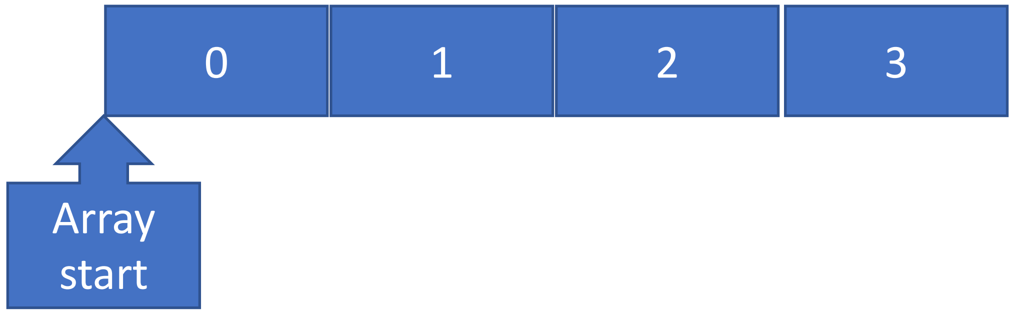
- Pointers
  - No such type
  - You can use any 8-bit numeric value as memory address
- Arrays
  - No such structure in the language
  - But some supporting constructs, like ds directive
  - You can use addresses (pointers) and address arithmetic to work with arrays
- Strings
  - Support for string literals in dc directive
  - No other support
  - You can operate strings like arrays of characters

More on data structures in CdM-8

- Structures
  - You can use tplate section to describe a structure layout
- Arrays of structures
  - Tplate section have _ symbol which designates size of the structure
    roughly equivalent to C sizeof operator
  - You must implement C-style pointer arithmetic manually
    but it is hard because you do not have multiplication
  - May be, it is good idea to pad structures to power of 2
    But in CdM-8 you must save memory
- Linked lists and other linked structures (trees, graphs)
  - No builtin support
  - Note that C also has no builtin linked lists
    it justs offer facilities to implement them manually (structures and pointers

Arrays

- Sequence of elements of same size



- Can be allocated by ds (Define Space) directive
- Or by dc directive with many operands (“initialized array”)

```
array: dc 1,2,3,4
array2: dc “Hello world”
```

Byte array (array of 8-bit ints or chars)

- Indexing with arbitrary index

```
ldi r0, array
ldi r1, index
add r0,r1  # r1 contains address of array[index]
ld r1,r1.    # r1 contains value of array[index]
```

- Scanning all elements

```
ldi r0, array
ldi r1, array.size+1
while
    dec r1
stays gt
    …
    inc r0
wend
```

How to determine array size?

- In most assemblers, you can assign arithmetic expressions to symbols
- Like:

```
array: dc 0,1,2,3,4
.set array_size=.-array # . (dot characters) means current position
```

- In CdM-8, no equivalent of .set directive, no . pseudo-symbol
- And limitations on symbol arithmetic
- I plan a feature request

Using tplate section to define constants

```
tplate array
        dc 0,2,5,3,4
        ds 1
size:
      asect 0
      br main
array: dc 0,2,5,3,4
main:
      ldi r1, array.size
```

- Ugly, because you need to duplicate dc statement
- You can use macros to avoid this
- We will discuss macros later
- Why all this?
  - Because tplate directive produces no code
  - And its labels are calculated in compile time, not in runtime

So, the array scanning routine (body)

```
main:
        ldi r0,array
        ld r0,r3
        ldi r1, array.size
        while
                dec r1
        stays gt
                ld r0,r2  # value of current element
                if
                        cmp r2,r3
                is gt
                        move r2,r3
                fi
                inc r0
        wend
        halt
```

Two-dimensional arrays

- Two possible implementations:
- Array of arrays
  - Indexing of [i1][i2] calculated as i1*row_size+i2
  - Not convenient on CdM-8 because you have no multiplication
  - Impossible if rows have different size (why not?)
- Array of pointers (takes extra memory)

```
row1: ds 5
row2: ds 6
row3: ds 4
array: dc row1, row2, row3 # Yes you can use labels as values in dc!
```

Arrays on stack

- Why not?
- Just allocate enough space on stack by using addsp instruction
- This way you can even allocate dynamic arrays
  (size defined at run time)
- Use ldsa instead of ldi to load array start pointer in r0..3
- BTW, do you know that C99 allows variable size arrays?
- Or, you can push the array element by element, and thus initialize it

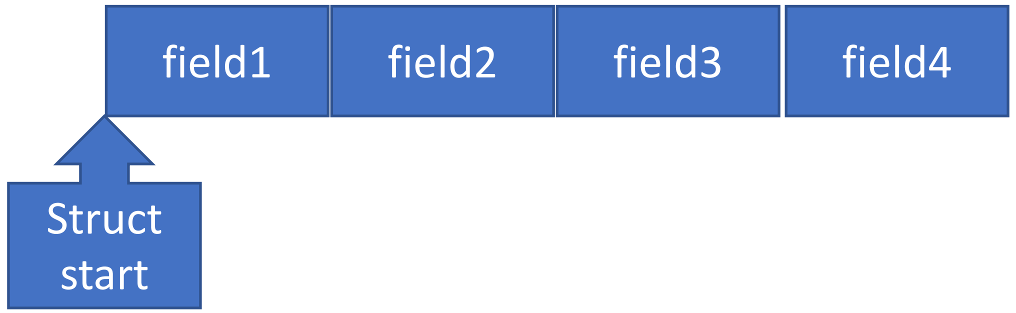
Copy array on stack and back (in reverse)

```
ldi r0, array
ldi r1, array.size
while
        dec r1
stays gt
        ld r0,r2
        push r2
        inc r0
wend
```

```
ldi r0, array
ldi r1, array.size
while
        dec r1
stays gt
        pop r2
        st  r0,r2
        inc r0
wend
```

Structures

- Structure is a collection of fields
- Fields are defined by offset from the beginning of the structure
- It can be seen as an array with predefined indices



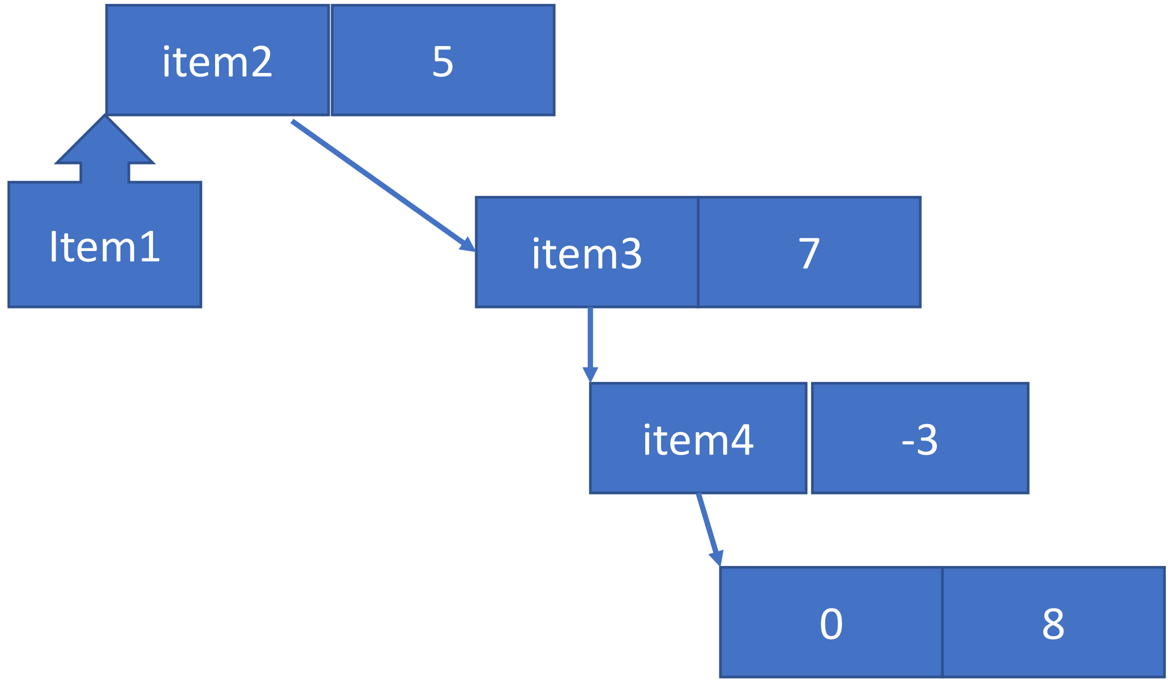
Tplate section can define structures

```
        tplate struct
field1: ds 1
field2: ds 1
field3: ds 1
field4: ds 1
        asect 0
struct: ds 4. # you can have tplate and label with same name!
main:
        ldi r0, struct+struct.field3 # unfortunately, impossible in CdM-8!
        ldi r0, struct
        ldi r1, struct.field3
        add r0, r1  # address of field3 is calculated at runtime
```

Linked lists

- But you can interpret some fields as addresses
- Below is valid CdM-8 data section

```
        asect 0x0D
item1:
        dc item2, 5
item2:
        dc item3, 7
item3:
        dc item4, -3
item4:
        dc 0x00, 8
```


item2
5
Item1
item3
7
item4
-3
0
8