Lecture 8
Data structures

Computing platforms

Novosibirsk State University
University of Hertfordshire

D. Irtegov, A.Shafarenko

2018

Структуры данных

- Структура данных — это адресная совокупность двух или более элементов данных, обеспечивающая механизм доступа к ее составным частям.
- Функция структуры данных состоит в объединении компонентов в коллекцию, которой можно манипулировать, как если бы она была неделимой, а также в поддержке извлечения, введения, модификации и реорганизации отдельных компонентов, составляющих часть коллекции.

Свойства структур данных

- Каждая структура данных имеет двойное назначение:

1. С одной стороны, это адресуемый контейнер для набора компонентов, объединяющий их одновременно.
   - Знание того, что определенная структура данных существует (хранится где-то в памяти компьютера), равносильно знанию того, что все ее компоненты существуют одновременно.
   - На всю структуру данных можно ссылаться и манипулировать ею как единым целым, и ее можно найти, указав ее начальный адрес в памяти.
   - Операция со структурой данных в целом может привести к изменению одной или нескольких ее составных частей или отношений между ними, но мы все равно получим структуру данных того же типа, идентифицированную таким же образом, над которым могут выполняться такие же операции.

- 2. С другой стороны, структура данных предоставляет механизм доступа, который позволяет программам находить и обрабатывать отдельные компоненты в определенном порядке и/или определенным образом.
  - Мы можем выполнять операции над отдельными компонентами, которые не влияют на структуру данных в целом, и в итоге получить результат, который не является структурой данных того же вида.

Структуры данных объясняют:

- 1. какие битовые строки компонентов она содержит;
- 2. как её составные битовые строки «склеиваются» в единую, более длинную битовую строку;
- 3. как найти начало и конец битовой строки каждого компонента.

Статические и динамические структуры данных

- Статическая структура данных (структуры данных предопределенного размера) предоставляет структуру, в которую вставляется информация (карта), описывающая расположение и размер каждого компонента в структуре, а также тип данных элемента компонента.
- Динамическая структура данных (структуры данных, размер которой может изменяться во время работы программы) по-прежнему дает типы данных элементов компонентов, но вместо полной структуры и карты описывает, как построить каждый из них по мере необходимости.

Структуры данных

- массивы и записи, имеют регулярную форму и представлены строками битов, которые хранятся в виде непрерывных блоков памяти. Структура данных такого типа, имеющая начальный адрес x и длину n байтов, будет занимать все ячейки памяти от адреса x до адреса x + n − 1 включительно.

Структуры данных

- ссылочные структуры данных (также известные как связанные структуры данных). Компоненты не располагаются в одной длинной последовательности в памяти
- В ссылочной структуре данных некоторые элементы данных являются ссылками (также известными как ссылки) на другие. Механизм ссылок может варьироваться от платформы к платформе. Ссылка может быть указателем на местоположение или ассоциативной ссылкой, которая предоставляет значение, связанное с местоположением (обычно называемое «ключом»).

Ссылочные структуры данных

- Указатель на место может быть разыменован немедленно, следуя за указателем на место и затем читая элемент данных, расположенный там;
- Для ассоциативной ссылки требуется механизм поиска (например, таблица, которая связывает ключи с местоположениями), который необходимо использовать для поиска элемента данных.

Ссылочные структуры данных

Указатели

- Недостатком указателей является отсутствие у них гибкости и необходимость поддерживать их согласованность.

Ассоциативная ссылка

- элемент данных, на который делается ссылка, может быть перемещен в любое время при условии, что таблица ассоциаций обновлена соответствующим образом. Все ассоциативные ссылки на элемент данных продолжают действовать, и нет необходимости знать, сколько их и где они хранятся в любой момент времени.

Представление структур данных

- В нашем представлении структура данных - единая непрерывная битовая строка, состоящая из последовательности более коротких битовых строк, объединенных вместе, каждая из которых представляет собой более простой элемент данных.
- Платформы обычно могут обрабатывать всю битовую строку сразу только в том случае, если она имеет заданную длину.
- Длина битовой строки, которая может быть извлечена, сохранена и обработана механизмом платформы целиком, называется длиной слова платформы, а битовая строка, которая имеет такую длину, может называться машинным словом.

Массивы (Arrays)

- это статическая структура данных (фиксированной длины), состоящая из определенного количества элементов данных одного типа, каждый из которых представлен битовой строкой одинаковой длины.
- массивы являются однородными структурами данных (каждый компонент массива имеет тот же тип, что и любой другой компонент).
- Массивы строк символов (Arrays of character strings)

Записи (Records)

- Запись — это статическая (т. е. фиксированной длины) структура данных, состоящая из двух или более элементов данных, которые могут быть разных типов, каждый из которых представлен битовой строкой фиксированной длины.
- записи представляют собой разнородные структуры данных (разные компоненты записи могут относиться к разным типам).
- Элементы данных компонента известны как поля записи.
- поле в записи может представлять данные, совершенно отличные от других, и каждое поле может быть представлено битовой строкой, длина которой отличается от остальных.

Строки с завершающим нулем и массивы символов

- любой индекс, который ссылается либо на символ NULL, либо на символ, который находитя дальше в строке, выходит за границы.
- Только те индексы, которые меньше позиции NULL, соответствуют допустимым элементам.
- Доступ полностью последовательный: чтобы добраться до k-го символа необходимо прочитать и проверить предыдущие k символов.

Адреса и указатели
Addresses and pointers

- В больших машинах обычно адреса представляют собой целые числа без знака, каждое из которых занимает ровно одно машинное слово.
- **Определение 12. Согласованность адресов** Платформа называется согласованной по адресам, если ее адресное пространство (количество различных областей памяти, к которым можно обращаться) равно числу различных битовых строк одинаковой длины, равной одному машинному слову.
- Если платформа имеет адресное пространство, которое больше или меньше, чем количество различных представлений, доступных в одном машинном слове, говорят, что адрес несовместим.

Указатели (Pointers)

- **Определение 13: Указатель (Pointers)**
- Указатель — это адрес, указывающий на элемент данных. Это может быть неделимый элемент, или структура данных, или элемент, являющийся компонентом структуры данных, но указатель всегда будет содержать начальный адрес рассматриваемого элемента.
- Указатель должен указывать на адрес памяти, занятый элементом данных, или его первый байт (в случае многобайтового элемента).

Связанный список (Linked list)

- Это структура данных, которая предлагает минимальные возможности для последовательного доступа к данным.
- С помощью указателей мы можем создать структуру данных — связанный список — который обеспечивает последовательный доступ и позволяет вставлять и удалять из последовательности:
-

Связанный список (Linked list)

field	type	offset (bytes)
next	list_of_X	0
payload	X	1

Где X — любой тип, неделимая структура или структура данных, включая строку с нулевым завершением.

1. указатель указывает на запись, состоящую из двух полей
2. первое поле (смещение 0) является указателем того же типа, что и
3. второе поле (смещение 1) это данные типа X

Связанный список (Linked list)

field	type	offset (bytes)
next	list_of_X	0
payload	X	1

- На первую запись указывает x напрямую, местоположение 13. Полезная нагрузка поля находится в местоположении со смещением 1, которое имеет адрес 14. Мы отмечаем первый элемент последовательности, 5, затем следуем указателю, содержащемуся в поле next. по смещению 0 (местоположение 13).

Последовательность : (5, 7, −3, 8).

address	content
...	...
10	15
11	7
12	...
13	10
14	5
15	20
16	-3
17	...
18	...
19	...
20	0
21	8
...	...

вставим 9 между элементами 7 и −3:

address	before	after
...	...	...
10	15	**18**
11	**7**	**7**
12	...	...
13	10	10
14	**5**	**5**
15	20	20
16	**-3**	**-3**
17	...	...
18	...	**15**
19	...	**9**
20	0	0
21	**8**	**8**
...	...	...

Удаление члена списка.

- Все, что нам нужно сделать, это переключить указатель предыдущего члена.

field	type	offset (bytes)
`next`	`double_list_of_X`	0
`prev`	`double_list_of_X`	1
`payload`	`X`	2

Data structures in CdM-8 assembly
Структуры данных в сборке CdM-8

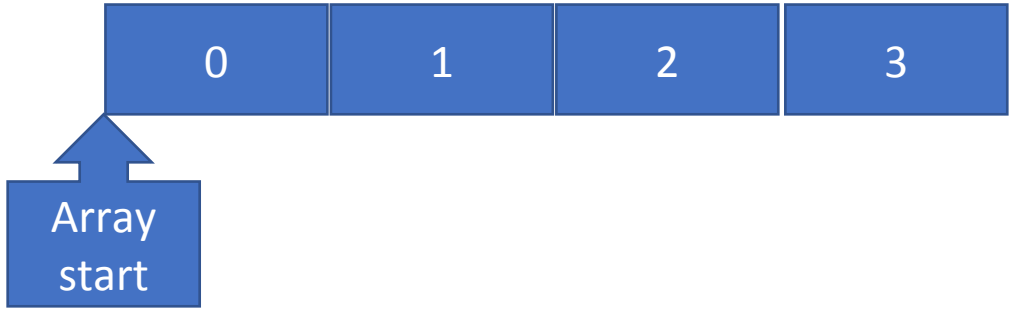
- Pointers (указатели)
  - No such type (Нет такого типа)
  - You can use any 8-bit numeric value as memory address (Вы можете использовать любое 8-битное числовое значение в качестве адреса памяти.)
- Arrays (Массивы)
  - No such structure in the language (Нет такой структуры в языке)
  - But some supporting constructs, like ds directive (Но некоторые вспомогательные конструкции, такие как директива ds)
  - You can use addresses (pointers) and address arithmetic to work with arrays (Вы можете использовать адреса (указатели) и адресную арифметику для работы с массивами)
- Strings (Строки)
  - Support for string literals in dc directive (Поддержка строковых литералов в директиве dc)
  - No other support (Никакой другой поддержки)
  - You can operate strings like arrays of characters (Вы можете оперировать строками как массивами символов)

More on data structures in CdM-8
Подробнее о структурах данных в CdM-8

- Структуры
  - Вы можете использовать раздел tplate для описания макета структуры
- Массивы структурp
  - Секция Tplate имеет символ _, который обозначает размер конструкции. roughly equivalent to C sizeof operator
  - Вы должны реализовать арифметику указателя в стиле C вручную. но это сложно, потому что у вас нет умножения
  - May be, it is good idea to pad structures to power of 2 But in CdM-8 you must save memory
- Связанные списки и другие связанные структуры (деревья, графы)
  - Нет встроенной поддержки
  - Обратите внимание, что C также не имеет встроенных связанных списков. он просто предлагает средства для их реализации вручную (структуры и указатели)

Arrays (Массивы)

- Последовательность элементов одинакового размера



- Может быть выделен директивой ds (Define Space)
- Или директивой dc со многими операндами («инициализированный массив»)

  array: dc 1,2,3,4

  array2: dc “Hello world”

Byte array (array of 8-bit ints or chars)
Массив байтов (массив 8-битных целых чисел или символов)

- Индексирование с произвольным индексом

```
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?
Как определить размер массива?

- В большинстве ассемблеров вы можете присваивать символам арифметические выражения.
- Like:

  array: dc 0,1,2,3,4

  .set array_size=.-array # . (dot characters) means current position
- В CdM-8 нет эквивалента директивы .set, нет .pseudo-symbol
- И ограничения на арифметику символов

Using tplate section to define constants
Использование раздела tplate для определения констант

```
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
```

- Некрасиво, потому что вам нужно дублировать оператор dc
- Вы можете использовать макросы, чтобы избежать этого
- Это обсудим позже
- Почему все это?
  - Потому что директива tplate не создает кода
  - И его метки вычисляются во время компиляции, а не во время выполнения.

So, the array scanning routine (body)
Итак, процедура сканирования массива (тело)

```
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
Двумерные массивы

- Две возможные реализации:
- 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

- Просто выделите достаточно места в стеке с помощью инструкции addsp
- Таким образом, вы даже можете выделить динамические массивы (размер определяется во время выполнения)
- Используйте ldsa вместо ldi для загрузки указателя начала массива в r0..3
- Или вы можете нажать элемент массива на элемент и, таким образом, инициализировать его.

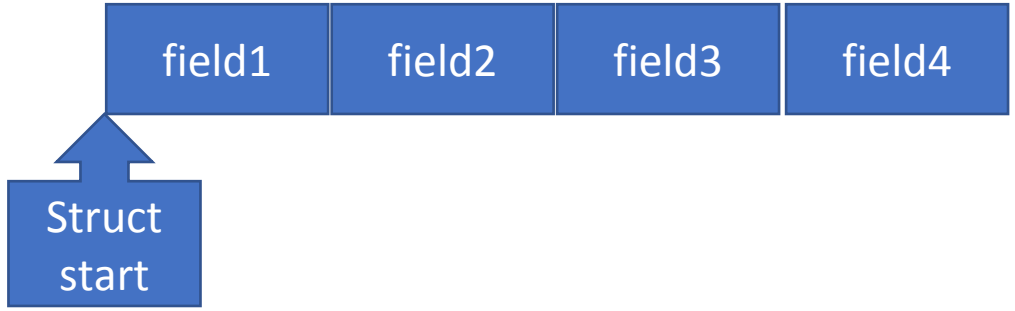
Copy array on stack and back (in reverse)

```
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

- Структура представляет собой набор полей
- Поля определяются смещением от начала структуры
- Его можно рассматривать как массив с предопределенными индексами.



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
```