**Лекция №8** Модель вычислений с косвенной адресацией (Равнодоступная адресная машина ,*РАМ – машина*). Машины с неограниченными регистрами (МНР). Машина с конечным числом регистров. Функциональные вычислительные модели.

**ЦЕЛЬ**: Ознакомить с понятиями: Модель вычислений с косвенной адресацией (Равнодоступная адресная машина, *РАМ – машина*). Машины с неограниченными регистрами (МНР). Функциональные вычислительные модели.

**ВОПРОСЫ**:

1. Модель вычислений с косвенной адресацией (Равнодоступная адресная машина ,*РАМ – машина*).
2. Машины с неограниченными регистрами (МНР).
3. Функциональные вычислительные модели.

**ОСНОВНЫЕ ПОНЯТИЯ:** Модель вычислений с косвенной адресацией (Равнодоступная адресная машина ,*РАМ – машина*). Машины с неограниченными регистрами (МНР). Функциональные вычислительные модели.

**РАВНОДОСТУПНАЯ АДРЕСНАЯ МАШИНА**

Равнодоступная адресная машина (РАМ) – машина это числовая модель вычислительного устройства. Эта модель является наиболее близкой из рассмотренных к реальным вычислительным машинам и позволяет наиболее реалистично применять теоретические оценки сложности алго-ритмов к реальным вычислениям.

***Память*** машины состоит из регистров (ячеек). Каждый регистр имеет адрес и может содержать произвольное число. Регистр с номером 0 называется сумматором.***Программа*** – последовательность пронумерованных команд. Команда имеет вид <код операции> <операнд>

Коды операций: Load, Store, Add, Sub, Mult, Div, Read, Write, Jump, JgtZ, Jzero, Halt.

***Операнд*** может быть одного из трех видов = *i*, *i*, \**i*, где *i* – натуральное число.

Содержимое регистра с номером *i* обозначим через *c* (*i*). Значение *v*(*a*) операнда *a* определяется в зависимости от его вида следующим образом

*v*(*a*) – число *i*, если *a* имеет вид =*i*,

*v*(*a*) – число *c*(*i*), если *a* имеет вид *i*,

*v*(*a*) – число *c*(*c*(*i*)), если *a* имеет вид \**i*.

При оценке сложности алгоритмов используют в зависимости от об-стоятельств разные веса команд. При работе с малыми числами и малым числом регистров реалистичным оказывается так называемый равномер-ный вес при котором каждая команда требует единицу времени. При ра-боте с большими числами и большим количеством регистров использует-ся так называемый логарифмический вес команды при котором время вы-полнения команды зависит как от значения операнда, так и от значения его адреса. При определении веса команды используется функция *L*: *Z* → *Z*, вы-ражающая длину записи числа.Основание логарифма при получении асимптотических оценок не имеет существенного значения.

Вес *t*(*a*) операнда *a* определяется в зависимости от его вида следующим образом

*t*(*a*) = *L*(*i*), если *a* имеет вид =*i*, *t*(*a*) = *L*(*i*) + *c*(*i*), если *a* имеет вид *i*,

|  |
| --- |
| *t*(*a*) = *L*(*i*) + *L*(*c*(*i*)) + *L*(*c*(*c*(*i*))), если *a* имеет вид \**i*. **Команда Действие Логарифмический** **вес**  |
| Load(*a*) *с*(0) := *v*(*a*) *t*(*a*)  |
| Store(*i*) *c*(*i*) := *c*(0) *L*(*c*(0)) + *L*(*i*)  |
| Store(\**i*) *c*(*c*(*i*)):=*c*(0) *L*(*c*(0)) + *L*(*i*) + *L*(*c*(*i*))  |
| Add(*a*) *с*(0) := *c*(0) + *v*(*a*) *L*(*c*(0)) + *t*(*a*)  |
| Sub(*a*) *с*(0) := *c*(0) − *v*(*a*) *L*(*c*(0)) + *t*(*a*)  |
| Mult(*a*) *с*(0) := *c*(0)·*v*(*a*) *L*(*c*(0)) + *t*(*a*)  |
| Div(*a*) *с*(0) := *c*(0) div *v*(*a*) *L*(*c*(0)) + *t*(*a*)  |
| Read(*i*) *c*(*i*) := очередное число *L*(*i*) + *L*(*c*(*i*))  |
| Read(\**i*) *c*(*c*(*i*)):= очередное число *L*(*i*) + *L*(*c*(*i*)) + *L*(*c*(*c*(*i*))  |
| Write(*a*) печать *v*(*a*) *t*(*a*)  |
| Jump(*a*) переход на команду с номером a 1  |
| JgtZ(*a*) переход на команду с номером *a*, если *c*(0)>0 *L*(*c*(0))  |
| Jzero(*a*) переход на команду с номером *a*, если *c*(0)=0 *L*(*c*(0))  |
| Halt(*a*) конец вычислений 1  |

**ФУНКЦИОНАЛЬНЫЕ МОДЕЛИ ВЫЧИСЛЕНИЯ**

**Функциональное программирование** — раздел [дискретной математики](http://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%81%D0%BA%D1%80%D0%B5%D1%82%D0%BD%D0%B0%D1%8F_%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0) и [парадигма программирования](http://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%80%D0%B0%D0%B4%D0%B8%D0%B3%D0%BC%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), в которой процесс [вычисления](http://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5) трактуется как вычисление значений [функций](http://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29) в математическом понимании последних (в отличие от функций как подпрограмм в [процедурном программировании](http://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D0%B4%D1%83%D1%80%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)). При необходимости, в функциональном программировании вся совокупность последовательных состояний вычислительного процесса представляется явным образом, например как [список](http://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_%28%D0%B8%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0%29).

**Функциональное программирование** предполагает обходиться вычислением результатов функций от исходных данных и результатов других функций, и не предполагает явного хранения состояния программы. Соответственно, не предполагает оно и изменяемость этого состояния (в отличие от[императивного](http://ru.wikipedia.org/wiki/%D0%98%D0%BC%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), где одной из базовых концепций является [переменная](http://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D0%B0%D1%8F_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29), хранящая своё значение и позволяющая менять его по мере выполнения[алгоритма](http://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC)).На практике отличие математической функции от понятия «функции» в императивном программировании заключается в том, что императивные функции могут опираться не только на аргументы, но и на состояние внешних по отношению к функции переменных, а также иметь [побочные эффекты](http://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B1%D0%BE%D1%87%D0%BD%D1%8B%D0%B9_%D1%8D%D1%84%D1%84%D0%B5%D0%BA%D1%82_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29)и менять состояние внешних переменных. Таким образом, в императивном программировании при вызове одной и той же функции с одинаковыми параметрами, но на разных этапах выполнения алгоритма, можно получить разные данные на выходе из-за влияния на функцию состояния переменных. А в функциональном языке при вызове функции с одними и теми же аргументами мы всегда получим одинаковый результат: выходные данные зависят только от входных. Это позволяет средам выполнения программ на функциональных языках [кешировать](http://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D0%BC%D0%BE%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F) результаты функций и вызывать их в порядке, не определяемом алгоритмом.

[**λ-исчисления**](http://ru.wikipedia.org/wiki/%D0%9B%D1%8F%D0%BC%D0%B1%D0%B4%D0%B0-%D0%B8%D1%81%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5) **являются основой для функционального программирования**, многие функциональные [языки](http://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) можно рассматривать как «надстройку» над ними

Наиболее известными [языками функционального программирования](http://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) являются:

* [LISP](http://ru.wikipedia.org/wiki/LISP) - ([Джон МакКарти](http://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D0%BA%D0%9A%D0%B0%D1%80%D1%82%D0%B8%2C_%D0%94%D0%B6%D0%BE%D0%BD), [1958](http://ru.wikipedia.org/wiki/1958)) и множество его диалектов, наиболее современные из которых:[Scheme](http://ru.wikipedia.org/wiki/Scheme), [Clojure](http://ru.wikipedia.org/wiki/Clojure%22%20%5Co%20%22Clojure), [Common Lisp](http://ru.wikipedia.org/wiki/Common_Lisp%22%20%5Co%20%22Common%20Lisp), [F#](http://ru.wikipedia.org/wiki/F_Sharp) — функциональный язык для платформы [.NET](http://ru.wikipedia.org/wiki/.NET_Framework)
* [Haskell](http://ru.wikipedia.org/wiki/Haskell) — [чистый](http://ru.wikipedia.org/wiki/%D0%A7%D0%B8%D1%81%D1%82%D0%BE%D1%82%D0%B0_%D1%8F%D0%B7%D1%8B%D0%BA%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) функциональный. Назван в честь [Хаскелла Карри](http://ru.wikipedia.org/wiki/%D0%9A%D0%B0%D1%80%D1%80%D0%B8%2C_%D0%A5%D0%B0%D1%81%D0%BA%D0%B5%D0%BB%D0%BB).
* [Erlang](http://ru.wikipedia.org/wiki/Erlang) — ([Joe Armstrong](http://ru.wikipedia.org/w/index.php?title=Joe_Armstrong&action=edit&redlink=1" \o "Joe Armstrong (страница отсутствует)), [1986](http://ru.wikipedia.org/wiki/1986)) функциональный язык с поддержкой процессов.
* [APL](http://ru.wikipedia.org/wiki/APL) — предшественник современных научных вычислительных сред, таких как [MATLAB](http://ru.wikipedia.org/wiki/MATLAB).
* [ML](http://ru.wikipedia.org/wiki/ML) ([Робин Милнер](http://ru.wikipedia.org/wiki/%D0%9C%D0%B8%D0%BB%D0%BD%D0%B5%D1%80%2C_%D0%A0%D0%BE%D0%B1%D0%B8%D0%BD), [1979](http://ru.wikipedia.org/wiki/1979), из ныне используемых диалектов известны [Standard ML](http://ru.wikipedia.org/wiki/SML) и [Objective CAML](http://ru.wikipedia.org/wiki/OCaml)).
* [Scala](http://ru.wikipedia.org/wiki/Scala_%28%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F%29)
* [Miranda](http://ru.wikipedia.org/wiki/%D0%9C%D0%B8%D1%80%D0%B0%D0%BD%D0%B4%D0%B0_%28%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F%29) ([Дэвид Тёрнер](http://ru.wikipedia.org/wiki/%D0%A2%D1%91%D1%80%D0%BD%D0%B5%D1%80%2C_%D0%94%D1%8D%D0%B2%D0%B8%D0%B4), [1985](http://ru.wikipedia.org/wiki/1985), который впоследствии дал развитие языку Haskell).
* [Nemerle](http://ru.wikipedia.org/wiki/Nemerle) — гибридный функционально/императивный язык.
* [XQuery](http://ru.wikipedia.org/wiki/XQuery)

*(Подробно истории и описания этих языков см. литературу).*Первым функциональным языком был [Lisp](http://ru.wikipedia.org/wiki/Lisp).

Некоторые **концепции и парадигмы специфичны для функционального программирования** и в основном чужды императивному программированию (включая [объектно-ориентированное программирование](http://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)). Тем не менее, языки программирования обычно представляют собой гибрид нескольких парадигм программирования, поэтому «большей частью императивные» языки программирования могут использовать какие-либо из этих концепций.

Хотя большинство компиляторов императивных языков программирования распознают чистые функции и удаляют общие подвыражения для вызовов чистых функций, они не могут делать это всегда для предварительно скомпилированных библиотек, которые, как правило, не предоставляют эту информацию. Некоторые компиляторы, такие как [gcc](http://ru.wikipedia.org/wiki/GNU_Compiler_Collection), в целях оптимизации предоставляют программисту ключевые слова для обозначения чистых функций. [Fortran 95](http://ru.wikipedia.org/wiki/%D0%A4%D0%BE%D1%80%D1%82%D1%80%D0%B0%D0%BD) позволяет обозначать функции как «pure» (чистые).

### Рекурсия. В функциональных языках [цикл](http://ru.wikipedia.org/wiki/%D0%A6%D0%B8%D0%BA%D0%BB_%28%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%29) обычно реализуется в виде рекурсии. Строго говоря, в функциональной парадигме программирования нет такого понятия, как цикл. Рекурсивные функции вызывают сами себя, позволяя операции выполняться снова и снова. Для использования рекурсии может потребоваться большой [стек](http://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B5%D0%BA), но этого можно избежать в случае [хвостовой рекурсии](http://ru.wikipedia.org/wiki/%D0%A5%D0%B2%D0%BE%D1%81%D1%82%D0%BE%D0%B2%D0%B0%D1%8F_%D1%80%D0%B5%D0%BA%D1%83%D1%80%D1%81%D0%B8%D1%8F). Хвостовая рекурсия может быть распознана и оптимизирована компилятором в код, получаемый после компиляции аналогичной итерации в императивном языке программирования. Стандарты языка Scheme требуют распознавать и оптимизировать хвостовую рекурсию. Оптимизировать хвостовую рекурсию можно путём преобразования программы в стиле использования продолжений при её компиляции, как один из способов. Рекурсивные функции можно обобщить с помощью функций высших порядков, используя, например, [катаморфизм](http://ru.wikipedia.org/w/index.php?title=%D0%9A%D0%B0%D1%82%D0%B0%D0%BC%D0%BE%D1%80%D1%84%D0%B8%D0%B7%D0%BC&action=edit&redlink=1) и [анаморфизм](http://ru.wikipedia.org/w/index.php?title=%D0%90%D0%BD%D0%B0%D0%BC%D0%BE%D1%80%D1%84%D0%B8%D0%B7%D0%BC&action=edit&redlink=1) (или «свертка» и «развертка»). Функции такого рода играют роль такого понятия как цикл в императивных языках программирования.

### Подход к вычислению аргументов.Функциональные языки можно классифицировать по тому, как обрабатываются аргументы функции в процессе её вычисления. Технически различие заключается в денотационной семантике выражения. К примеру, при строгом подходе к вычислению выражения print length([2+1, 3\*2, 1/0, 5-4]) на выходе будет ошибка, так как в третьем элементе списка присутствует деление на ноль. При нестрогом подходе значением выражения будет 4, поскольку для вычисления длины списка значения его элементов, строго говоря, не важны и могут вообще не вычисляться. При строгом (аппликативном) порядке вычисления заранее подсчитываются значения всех аргументов перед вычислением самой функции. При нестрогом подходе (нормальный порядок вычисления) значения аргументов не вычисляются до тех пор, пока их значение не понадобится при вычислении функции[5]

Как правило, нестрогий подход реализуется в виде редукции графа. Нестрогое вычисление используется по умолчанию в нескольких чисто функциональных языках, в том числе [Miranda](http://ru.wikipedia.org/wiki/%D0%9C%D0%B8%D1%80%D0%B0%D0%BD%D0%B4%D0%B0_%28%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F%29),[Clean](http://ru.wikipedia.org/wiki/Clean) и [Haskell](http://ru.wikipedia.org/wiki/Haskell). Функциональное программирование предполагает обходиться вычислением результатов функций от исходных данных и результатов других функций, и не предполагает явного хранения состояния программы

**МАШИНЫ С НЕОГРАНИЧЕННЫМИ РЕГИСТРАМИ (МНР)**

Вы­бран­ный на­ми ги­по­те­ти­че­с­кий ком­пь­ю­тер на­зы­ва­ет­ся ма­ши­ной с не­огра­ни­чен­ны­ми ре­ги­стра­ми (МНР) [Кат­ленд,1983,16-29]; он яв­ля­ет­ся лег­ким ви­до­из­ме­не­ни­ем ма­шин, впер­вые рас­смо­трен­ных Дж.Шепердсоном и Х.Стерджисом (1963) и по­лу­чив­ших на­зва­ние РАМ (рав­но­до­ступ­ная ад­рес­ная ма­ши­на) и РАСП (рав­но­до­ступ­ная ад­рес­ная ма­ши­на с хра­ни­мой про­грам­мой). Эти мо­де­ли в бо­ль­шей сте­пе­ни, чем ма­ши­на Тью­рин­га от­ра­жа­ют струк­ту­ру сов­ре­мен­ных вы­чис­ли­те­ль­ных устройств.

МНР со­дер­жит бес­ко­неч­ное чис­ло ре­ги­стров, обоз­на­ча­емых че­рез R1,R2,R3,..., каж­дый из ко­то­рых в лю­бой мо­мент вре­ме­ни со­дер­жит не­ко­то­рое на­ту­ра­ль­ное чис­ло, при­чём чис­ло, со­дер­жа­ще­еся в Rn, бу­дем обоз­на­чать че­рез rn. Это мож­но изоб­ра­зить сле­ду­ющим об­ра­зом:



МНР мо­жет из­ме­нять со­дер­жи­мое ре­ги­стров с по­мо­щью не­ко­то­рой ко­ман­ды. Каж­дая ко­ман­да мо­жет быть од­но­го из сле­ду­ющих че­ты­рех ви­дов: ко­ман­да об­ну­ле­ния, ко­ман­да при­бав­ле­ния еди­ни­цы, ко­ман­да пе­ре­ад­ре­са­ции и ко­ман­да усло­вно­го пе­ре­хо­да (см.таблицу 1).



 Ариф­ме­ти­че­с­ки­ми ко­ман­да­ми на­зо­вём ко­ман­ды об­ну­ле­ния, при­бав­ле­ния еди­ни­цы и пе­ре­ад­ре­са­ции, при­чём ко­ман­ды Z(n) и S(n) со­от­ве­т­ст­ву­ют про­стей­шим опе­ра­ци­ям над на­ту­ра­ль­ны­ми чис­ла­ми в фор­ма­ль­ной ариф­ме­ти­ке.Про­грам­ма МНР - это ко­неч­ная по­сле­до­ва­те­ль­ность ко­манд. На­ча­ль­ной кон­фи­гу­ра­ци­ей на­зы­ва­ет­ся по­сле­до­ва­те­ль­ность a1,a2,...на­ту­ра­ль­ных чи­сел, со­дер­жа­щи­х­ся в ре­ги­страх R1,R2,....

Что­бы МНР при­сту­пи­ла к вы­чис­ле­ни­ям (ра­бо­те), она до­лжна быть снаб­же­на про­грам­мой P и на­ча­ль­ной кон­фи­гу­ра­ци­ей.Пусть P«(I1,I2,...Is), где I1,I2,...Is - по­сле­до­ва­те­ль­ность ко­манд. Вна­ча­ле МНР вы­пол­ня­ет ко­ман­ду I 1.Опре­де­лим ин­ду­к­тив­но по­ня­тие "сле­ду­ющая ко­ман­да в вы­чис­ле­нии".

 **Определение**.(1) I1 - сле­ду­ющая ко­ман­да в вы­чис­ле­нии.(2) Ес­ли Ik (k=1,2,...,s-1) не яв­ля­ет­ся ко­ман­дой усло­вно­го пе­ре­хо­да, то ко­ман­да Ik+1 яв­ля­ет­ся сле­ду­ющей ко­ман­дой в вы­чис­ле­нии.(3) Ес­ли Ik (k=1,2,...,s-1) яв­ля­ет­ся ко­ман­дой J(m,n,q), а rm и rn - те­ку­щее со­дер­жи­мое ре­ги­стров Rm и Rn со­от­ве­т­ст­вен­но, то сле­ду­ющей ко­ман­дой в вы­чис­ле­нии яв­ля­ет­ся ко­ман­да Iq, q=1,2,...,s, ес­ли rm=rn, и ко­ман­да Ik+1 - в про­тив­ном слу­чае.(4) Ес­ли Is яв­ля­ет­ся ко­ман­дой J(m,n,q), а rm и rn - те­ку­щее со­дер­жи­мое ре­ги­стров Rm и Rn со­от­ве­т­ст­вен­но, то сле­ду­ющей ко­ман­дой в вы­чис­ле­нии яв­ля­ет­ся ко­ман­да Iq, ес­ли rm=rn и q£s.(5) Ни­ка­ких дру­гих сле­ду­ющих ко­манд в вы­чис­ле­нии нет. Определение.Бу­дем го­во­рить, что сле­ду­ющая ко­ман­да от­сут­ст­ву­ет, ес­ли:

(1) Is яв­ля­ет­ся ариф­ме­ти­че­с­кой ко­ман­дой;

(2) Is яв­ля­ет­ся ко­ман­дой J(m,n,q), при­чём rm=rn и q>s;

(3) Is яв­ля­ет­ся ко­ман­дой J(m,n,q), при­чём rm¹rn. В этом слу­чае го­во­рят, что вы­чис­ле­ние оста­но­ви­лось по­сле вы­пол­не­ния ко­ман­ды Is, а за­клю­чи­те­ль­ной кон­фи­гу­ра­ци­ей бу­дем на­зы­вать по­сле­до­ва­те­ль­ность r1,r2,r3,... со­дер­жи­мых ре­ги­стров на этом ша­ге.

 Определение.Бу­дем го­во­рить, что вы­чис­ле­ние оста­нав­ли­ва­ет­ся, ес­ли сле­ду­ющая ко­ман­да от­сут­ст­ву­ет.МНР ра­бо­та­ет, по­ка вы­чис­ле­ние не оста­но­ви­т­ся.

 Предложение.Для каж­дой ко­ман­ды пе­ре­ад­ре­са­ции T(m,n) су­ще­ст­ву­ет про­грам­ма, не со­дер­жа­щая ко­манд пе­ре­ад­ре­са­ции, ко­то­рая на вся­кой кон­фи­гу­ра­ции МНР да­ет тот же ре­зу­ль­тат, что и T(m,n). Та­ким об­ра­зом, ко­ман­да пе­ре­ад­ре­са­ции в опре­де­ле­нии МНР яв­ля­ет­ся из­бы­то­чной; тем не ме­нее, пред­став­ля­ет­ся ес­те­ст­вен­ным и удоб­ным иметь

Команды МНР и ихоперационнаяемантика та­кие ко­ман­ды, т.к. они об­лег­ча­ют по­стро­ение про­грамм.

**МАШИНА С КОНЕЧНЫМ ЧИСЛОМ РЕГИСТРОВ**

Рас­смо­трим дру­гую мо­дель, ко­то­рую мож­но на­звать *ма­ши­на­ми с ко­неч­ным чис­лом ре­ги­стров*[Ве­ре­ща­гин,Шень,1999,с.129]. Про­грам­ма для та­кой ма­ши­ны ис­по­ль­зу­ет ко­неч­ное чис­ло *пе­ре­мен­ных*, зна­че­ни­ями ко­то­рых яв­ля­ют­ся на­ту­ра­ль­ные чис­ла. Чис­ла мо­гут быть про­из­во­ль­но­го раз­ме­ра, так что ма­ши­на ре­аль­но име­ет па­мять не­огра­ни­чен­но­го объ­ё­ма. (*подробно см. литературу*)

*Про­грам­ма* со­сто­ит из ну­ме­ро­ван­ных по по­ряд­ку*ко­манд*. Каж­дая ко­ман­да име­ет один из сле­ду­ющих ви­дов [Ве­ре­ща­гин,Шень,1999,с.129]:

(1) a:=0 (ко­ман­да об­ну­ле­ния)

(2) a:=b (ко­ман­да ко­пи­ро­ва­ния)

(3) a:=b+1 (ко­ман­да при­бав­ле­ния еди­ни­цы)

(4) a:=b-1 (ко­ман­да вы­чи­та­ния еди­ни­цы)

(5) goto <Но­мер> (ко­ман­да бе­зус­ло­вно­го пе­ре­хо­да)

(6) if a=0 (ко­ман­да усло­вно­го пе­ре­хо­да) thengoto <Но­мер 1> else goto <Но­мер 2>

(7) stop (ко­ман­да оста­но­вки)

 ***Функция, вычислимая в этой модели.*** Опре­де­лим по­ня­тие "*вы­чис­ли­мая (в этой мо­де­ли) фун­кция*".Пусть есть про­грам­ма с дву­мя пе­ре­мен­ны­ми x иy (и воз­мож­но, дру­ги­ми). по­мес­тим в пе­ре­мен­ную x не­ко­то­рое чис­ло n, а в оста­ль­ные пе­ре­мен­ные по­мес­тим ну­ли. За­пус­тим про­грам­му. Ес­ли она не оста­но­ви­т­ся, то вы­чис­ля­емая еюфун­кция в точ­ке n не опре­де­ле­на. Ес­ли оста­но­ви­т­ся, то со­дер­жи­мое пе­ре­мен­ной y по­сле оста­но­вки и бу­дет зна­че­ни­ем фун­кции, вы­чис­ля­емой на­шей про­грам­мой (в точ­ке n).

 **Опре­де­ле­ние** [Ве­ре­ща­гин,Шень,1999,с.131]**.**  Фун­кция на­зы­ва­ет­ся *вы­чис­ли­мой* (*в этой мо­де­ли*), ес­ли су­ще­ст­ву­ет вы­чис­ля­ющая её про­грам­ма.При анализе задачи важно знать ∃ ли полиномиальный алг. ее решения. Частично на этот ? отвечает теория NP-полноты. Класс NP – это мн. ЗРС, у кот. проверка ответа «Да» для *заданного* решения осуществляется за полиномиальное время.ЗРС, соответствующие ЗР, КМ, ГЦ, … ∈ NP.

Трудоёмкость алгоритма Пусть *n* – входная длина. Алг. *A*1 решает задачу *P* с трудоемкостью *O*(*n5*).Алг. *A*2 имеет трудоемкость *O*(2*n*) решения задачи *P*. ЭВМ - 1 млн. опер./сек. Тогда при *n =* 60 алг. *A*1 будет работать около **13 минут**, а алг. *A*2 – более **300 столетий** Предположим, что *A*2 строит решение задачи размерности *D* на вышеупомянутом компьютере за 1 час. Если взять компьютер, выполняющий в 1000 раз больше операций в секунду, то размерность задачи, которая решается алг. *A*2 на таком компьютере в течение 1 часа, будет всего ***D* + 9.97**.

**ВЫВОДЫ:** Равнодоступная адресная машина (РАМ) – машина это числовая модель вычислительного устройства. Эта модель является наиболее близкой из рассмотренных к реальным вычислительным машинам и позволяет наиболее реалистично применять теоретические оценки сложности алгоритмов к реальным вычислениям. времени. При работе с большими числами и большим количеством регистров используется так называемый логарифмический вес команды при котором время вы-полнения команды зависит как от значения операнда, так и от значения его адреса. РАМ (рав­но­до­ступ­ная ад­рес­ная ма­ши­на) и РАСП (рав­но­до­ступ­ная ад­рес­ная ма­ши­на с хра­ни­мой про­грам­мой). Эти мо­де­ли в бо­ль­шей сте­пе­ни, чем ма­ши­на Тью­рин­га от­ра­жа­ют струк­ту­ру сов­ре­мен­ных вы­чис­ли­те­ль­ных устройств.По­стро­ен­ная вы­чис­ли­те­ль­ная мо­дель не сла­бее ма­шин Тью­рин­га в том смыс­ле, что лю­бую вы­чис­ли­мую на ма­ши­нах Тью­рин­га фун­кцию мож­но вы­чис­лить и про­грам­мой с ко­неч­ным чис­лом пе­ре­мен­ных.Про­грам­мы с ко­неч­ным чис­лом пе­ре­мен­ных на­по­ми­на­ют ас­се­мб­лер, а ре­кур­сив­ные фун­кции ско­рее на­по­ми­на­ют фун­кци­она­ль­ное про­грам­ми­ро­ва­ние.

**№ 8 практическая работа.** М Модель вычислений с косвенной адресацией (Равнодоступная адресная машина, РАМ – машина). Машина с неограниченными регистрами (МНР). Машина с конечным числом регистров Функциональные вычислительные модели.

**ЗАДАНИЯ: Опишите принцип работы и приведите примеры**

1. Модель вычислений с косвенной адресацией (Равнодоступная адресная машина, РАМ – машина)
2. Машина с неограниченными регистрами (МНР).
3. Машина с конечным числом регистров
4. Функциональные вычислительные модели.
5. Подготовить презентацию «Языки функционального программирования»