Обзорный курс по языкам программирования

Антон Москаль

17 января 2003 г.

Оглавление

Глава 1

Основные концепции языков программирования

1.1 Управляющие конструкции

1.1.1 переходы, хвостовая рекурсия

1.1.2 `if`, `match`

1.1.3 циклы, `while` через рекурсию

1.1.4 исключения

Глава 2

Языки программирования

2.1 Языки ассемблера

Языки ассемблера являются языками низкого уровня, предназначенными для программирования в терминах машинных команд. Наиболее существенными их особенностями являются:

- прямое соответствие операторов ассемблера и машинных команд
- средства, позволяющие детально управлять размещением программы и данных в памяти
- макросредства, позволяющие создавать псевдокоманды несколько более высокого уровня, нежели машинные операции

В свое время языки ассемблера использовались как языки для системного программирования: на них писались операционные системы, компиляторы и просто стандартные утилиты. Вызывалось это с одной стороны недостатком гибкости тогдашних языков высокого уровня, с другой стороны – крайней ограниченностью ресурсов. Ассемблер позволяет "выжать" из аппаратуры максимум возможного, особенно, пока объемы программ сравнительно невелики и проблемы, связанные со сложностью и управляемостью проекта не начинают становиться определяющими.

А на машинах, у которых объем памяти исчислялся тысячами, в лучшем случае – десятками тысяч слов, они не могли быть велики.

Со временем область применения ассемблера сужалась, и в настоящий момент она ограничена программированием критических по эффективности фрагментов кода (часто переписывание нескольких десятков строк критического участка кода на ассемблер способно дать выигрыш в скорости в несколько раз, иногда на порядки), особенно, если там возможно использование специализированных команд, наподобие команд сигнальных процессоров или `IA-32 SSE`.

Кроме этого, язык ассемблера часто используется как промежуточное представление программы в процессе компиляции: большинство компиляторов, работающих под `Unix`, порождают именно ассемблер, который потом уже компилируется в объектный код.

2.1.1 Система команд `PDP-11`

`PDP-11` – одна из вычислительных машин, разработанных фирмой `DEC` (*Digital Equipment Corporation*) в ??? году. Хотя машина полностью устарела, ее архитектура оказала большое влияние на последующую эволюцию вычислительной техники, а появившиеся на ней ОС `Unix` и язык программирования `C` до сих пор являются важнейшими инструментальными средствами.

`PDP-11` является 16-разрядной машиной с байтовой адресацией. Она имеет восемь регистров общего назначения, имеющих номера с 0 до 7. Регистры 6 и 7 выполняют специальные функции: Регистр 7 называется счетчиком команд и содержит адрес следующей, подлежащей выполнению, инструкции. Регистр счетчика команд обозначается `PC` (*program counter*). С ним можно выполнять все те же операции, которые

можно выполнять и с обычными регистрами, но следует иметь в виду, что изменение его содержимого приводит к изменению того, какая команда будет выполнена следующей. Скажем, засылка адреса в `PC` по действию эквивалентна операции безусловного перехода.

Регистр 6 называется указателем стека `SP` (*stack pointer*) и мало отличается от обычных регистров. Остальные регистры обычно обозначаются именами `R0`–`R5`.

Команды `PDP-11` состоят из одного слова, за которым, в зависимости от видов операндов, могут следовать одно или два слова данных. Команды `PDP-11` бывают двух-, одно-, и безадресные. У подавляющего большинства команд операнд кодируется унифицированным образом: каждый операнд занимает в коде команды 6 бит, первые три из которых задают способ адресации, а последние три – номер используемого для адресации операнда, регистр:

M_2	M_1	M_0	R_2	R_1	R_0

Основные виды адресации

$M = 0$: *Регистровая* адресация. Используется для непосредственного обращения к регистру, номер которого содержится в младших трех битах `R`. В языке ассемблера этот вид адресации обозначается просто именем или номером регистра (`R0` либо `%0`)

$M = 2$: *Косвенная пост-автоинкрементная* адресация: номер регистра используется в качестве адреса операнда. *После* использования производится увеличение его содержимого на 1 или 2 (в зависимости от того, работает ли данная команда с байтами или со словами) [1]. Записывается этот вид операндов на ассемблере так: `(R5)+`.

$M = 4$: *Косвенная пре-автодекрементная* адресация: аналогична предыдущему виду, но содержимое регистра, *уменьшается*, а не увеличивается, и производится это *перед* использованием содержимого регистра в качестве адреса. Записывается этот вид операндов на ассемблере так: `-(SP)`.

$M = 6$: *Относительная* адресация: при вычислении адреса выбирается значение слова, на которое в данный момент указывает `PC` (значение счетчика команд при этом увеличивается на 2) и в качестве адреса используется сумма этого слова с базовым регистром.

Виды адресации с нечётными номерами ($M_0 = 1$) являются "косвенными" модификациями соответствующих "чётных" видов адресации. Это означает, что значения операндов, получаемые в результате обработки соответствующего "четного" вида адресации не используются непосредственно, а рассматриваются в качестве адреса операнда. Этот вид адресации записывается на ассемблере путем приписывания в начале операнда значка |@|: `@R0`, `@(R5)+`, `@-(SP)`, `@10(R2)`.

Дополнительные виды адресации В случае, когда в качестве базового регистра используется `PC`, некоторые виды адресации приобретают особый смысл:

- *Непосредственная* адресация: рассмотрим команду `MOV (PC)+, R0`. В момент выборки первого операнда PC указывает на слово, следующее за этой командой. В результате оно выбирается в качестве операнда, а `PC` продвигается на следующий адрес. Таким образом фактически эта команда засылает литерал, следующий за ней в регистр `R0`.

  Поскольку необходимость в такой адресации встречается часто, то для нее в ассемблере `PDP-11` введено специальное обозначение: `#n`. То есть, если в нашем примере за командой следует число 1, что записывается на ассемблере следующим образом:

```
 mov   (pc)+, r0
.word     1
```

  то вместо этого можно написать просто `mov #1, r0`

[1] в случае использования в этом и следующем видах адресации регистров `SP` и `PC`, их содержимое всегда увеличивается на два, независимо от размера операнда. Это обеспечивает сохранение выравнивания адреса команды или верхушки стека

- *Абсолютная* адресация: если вместо автоинкрементной адресации относительно `PC` используется косвенная автоинкрментная адресация ($M = 3$), то слово, следующее за командой используется в качестве адреса операнда: таким образом команда `mov #1, @#17650` перешлет слово `1` по адресу `176560`.

- *Относительная* адресация: когда в относительной адресации ($M = 6$) в качестве базового регистра используется `PC`, то слово, следующее за командой фактически обозначает адрес операнда, отсчитываемый *относительно* адреса текущей команды.

  В принципе, относительная и абсолютная адресация взаимозаменяемы (за исключением того, что относительная адресация может использовать косвенным образом), но как правило абсолютной адресацией пользуются для обращения к адресам, имеющим постоянный физический адрес (векторам прерываний, портам внешних устройств и т.п.), тогда как относительную адресацию используют для обращения к переменным, лежащим вместе с исполняемым кодом. Это позволяет минимизировать зависимость кода программы от ее местоположения в памяти (в идеальном случае - свести ее к нулю).

  Относительная адресация от `PC` обозначается просто указанием адреса или метки: `mov #1, 176562` эквивалента по смыслу команде из предыдущего пункта.

2.1.2 Метки, локальные метки, операторы присваивания и секционирования

2.1.3 Макросредства

2.1.4 "Минимальная" архитектура: `PDP-8`

Литература: [3]

2.2 Язык `C`

2.2.1 `CPL`, `BCPL`, создание `B`

2.2.2 Описание `B`

Язык `B` был разработан для

2.2.3 `PDP-11`: Эволюция `B` в `C`

2.2.4 Дальнейшая эволюция `C`

- препроцессор `C`, *K&R* `C`
- `ANSI C`
- `C99`

Литература: [1], [2],

2.3 Функциональные языки

2.3.1 Основные понятия λ-исчисления

2.3.2 LISP, Scheme

2.3.3 ML

2.3.4 ленивые языки: Haskell

2.3.5 Ввод-вывод в чисто функциональных языках, линейные типы в Clean

2.4 Объектно-ориентированные языки

2.4.1 Simula-67

2.4.2 SmallTalk

2.4.3 Self: Бесклассовый ОО-язык

2.4.4 Различные виды наследования

Множественное наследование в C++

Интерфейсы в Java

Signatures в GNU C++, наследование в O'Caml

2.4.5 Мультиметоды: Язык Cecil

Литература: [4]

Литература

[1] Dennis Ritchie, *Evolution of C*

[2] Brian Kernighan, Dennis Ritchie, *C programming language*

[3] Брусенцов, Мини-ЭВМ

[4] Simula-67