III
ОРГАНИЗАЦИЯ АЛЬФА-ТРАНСЛЯТОРА*
1.ОБЩАЯ СТРУКТУРА

1.1. Требования и возможности. Трудности в принятии решений об организации АЛЬФА-транслятора типичны для любой задачи конструирования сложных систем: необходимо преодолевать всегда имеющееся противоречие между заданными эксплуатационными характеристиками системы и наличными средствами для ее реализации.

В случае АЛЬФА-транслятора проектное задание требовало достичь высокого качества программирования, приближающегося к качеству массовой продукции профессионального программиста при сохранении приемлемой скорости трансляции (не менее 100 команд рабочей программы на минуту работы транслятора). Необходимо было считаться также со сложностью АЛЬФА-языка и ограниченными возможностями машины: малая емкость оперативной памяти – 4096 слов, сравнительно небольшая емкость и недостаточная надежность магнитных барабанов – 12 288 слов, сравнительно малая скорость магнитной ленты (около 3000 слов в секунду). Единственным благоприятным фактором была высокая скорость машины при выполнении логических операций над оперативной памятью – 40 000 трехадресных операций в секунду.

С логической точки зрения процесс программирования – это процесс перевода с языка высшего уровня (АЛЬФА-языка) на язык низшего уровня (язык машины). Понятие уровня языка не точно определяемо, но содержательно включает в себя качества языка, характеризующие степень его сложности: богатство синтаксических конструкций, степень разнообразия форматов, количество типов операторов и т. п. Принципиально важно, что при переводе с языка высшего уровня на язык низшего уровня теряется некоторая информация, содержащаяся в программе высшего уровня. Действительно, даже досконально зная алгоритмы трансляции, в общем случае в принципе невозможно восстановить исходную АЛЬФА-программу по рабочей программе.

Основу процесса программирования составляют прямые универсальные алгоритмы трансляции, реализующие некоторую синтаксическую единицу высшего уровня в виде совокупности операторов низшего уровня. Однако универсальные алгоритмы, рассчитанные на общий случай, не учитывают реальной статистики употребления синтаксических конструкций высшего уровня и в большинстве случаев создают в программе низшего уровня избыточные конструкции, понижая качество программы. В АЛГОЛе 60, составляющем главную часть АЛЬФА-языка, это, в первую очередь, касается процедур и циклов.

Одним из средств повышения качества трансляции служит разработка алгоритмов программирования, учитывающих индивидуальные особенности транслируемой программы. Для этого универсальный алгоритм трансляции дополняется серией более простых "стратегий" программирования, рассчитанных на более компактные конструкции в рабочей программе. "Стратегия" выбирается на основе заранее выполняемого анализа, цель которого – установить реальную сложность употребления в программе синтаксических конструкций высшего уровня.

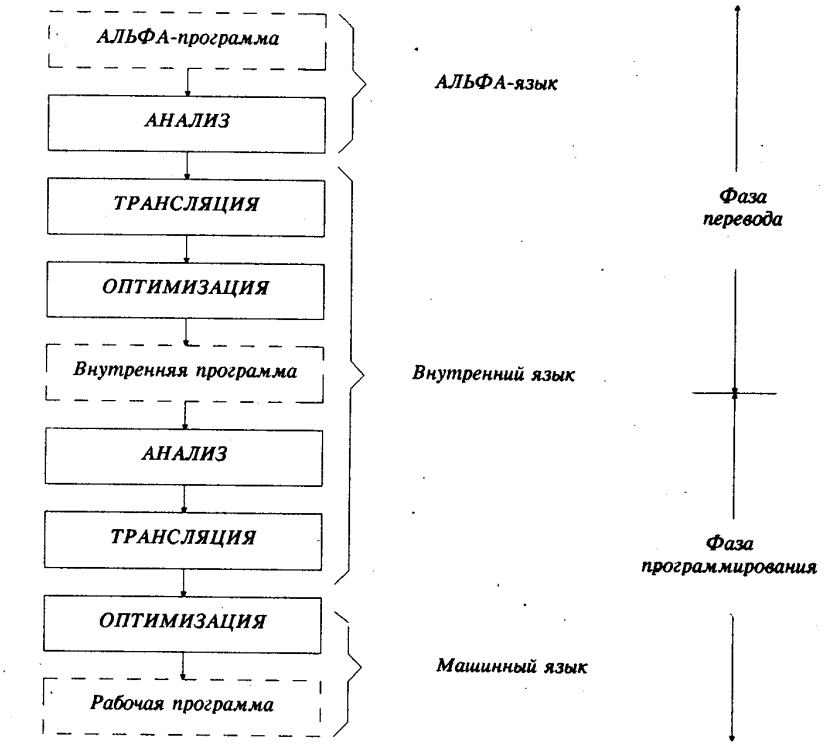
Другим средством улучшения качества программирования является выполнение оптимизирующих преобразований транслируемой программы, т. е. преобразований, сохраняющих уровень, в котором записана программа, но улучшающих те ее характеристики, которые связаны с объемом памяти или временем, требуемым для ее выполнения.

Анализ АЛЬФА-языка показывает, что он совершенно не приспособлен к выполнению формальных оптимизирующих преобразований из-за переменности формата идентификаторов и чисел, рекурсивного строения выражений и операторов, контекстного понимания многих символов и обилия подразумеваемых, но не указанных явно алгоритмических действий. Однако язык низшего уровня (язык машины), обладая необходимой простотой синтаксиса и семантики, состоит из слишком мелких единиц действия и – что самое главное – уже не содержит части той информации о программе, которая пропала при трансляции, но была бы полезна для большей эффективности оптимизирующих преобразований.

1.2. Многофазная трансляция. В связи с вышеизложенным при организации АЛЬФА-транслятора естественно возникла мысль о введении промежуточного языка, на уровне которого выполнялись бы те оптимизирующие преобразования, которые не могут быть выполнены на низшем уровне. Этот язык, получивший в АЛЬФА-трансляторе название внутреннего, является также барьером, который отделяет алгоритмы трансляции, ориентированные на АЛЬФА-язык и не зависящие от рабочей машины, от алгоритмов трансляции, ориентированных на машинный язык и в значительной степени зависящих от особенностей рабочей машины.

* Отдельная статья [18].

В результате работа АЛЬФА-транслятора приобретает характер многофазного процесса (схема 1).



Процесс трансляции с АЛЬФА-языка на внутренний язык получил название *фазы перевода*, а трансляции с внутреннего языка на машинный – *фазы программирования*. Это расчленение процесса трансляции служит не только средством повышения качества программирования, но и создает принципиальную возможность разбиения программы АЛЬФА-транслятора на последовательно работающие блоки, достаточно мелкие, чтобы каждый из них вместе с необходимой для его работы информацией помещался в оперативной памяти.

В результате анализа соотношения размеров и скоростей запоминающих устройств машин и их сопоставления с размерами АЛЬФА-транслятора и возможными размерами транслируемых программ были приняты следующие решения.

Все 24 блока АЛЬФА-транслятора размерами от 500 до 3000 слов хранятся на отдельных зонах магнитной ленты, работающей при отсутствии указания на необходимость двойного счета при трансляции в режиме только считывания и одностороннего движения. Вся информация о транслируемой программе разбивается на два типа. К первому типу относится собственно программа, более точно, ее операторная часть; ко второму – вся остальная информация о программе, хранящаяся в виде отдельных таблиц. До некоторой степени таблицы соответствуют описательной части АЛЬФА-программы. Программа целиком хранится на барабане и считывается в оперативную память для переработки кусками. Таблицы, которые нужны для работы очередного блока, хранятся в оперативной памяти, остальные – на барабане. Алгоритмы трансляции организованы таким образом, чтобы каждый блок осуществлял строго последовательную и однократную запись и чтение программы, а также входил в таблицы строго по адресу востребуемой информации. Таким образом сокращается количество обращений к барабану и полностью используется адресная структура оперативной памяти.

2. ВНУТРЕННИЙ ЯЗЫК

2.1. Требования. Выбор внутреннего языка накладывает существенный отпечаток на весь транслятор, поэтому структура и содержание внутреннего языка будут описаны достаточно подробно.

В соответствии с теми задачами, которые возложены на внутренний язык, он, по мнению автора, должен удовлетворять следующим требованиям:

1) отсутствие неявных или подразумеваемых алгоритмических действий;

2) фиксированный формат основных символов, идентификаторов и чисел;
3) отсутствие рекурсивности в строении операторов;
4) способность к сохранению информации, извлекаемой из информации АЛЬФА-программы;
5) приспособленность к выполнению оптимизирующих преобразований и к последующей трансляции на язык машины*.

Информация о транслируемой программе делится во внутреннем языке на программу и таблицы. Программа представляет собой последовательность основных символов.

2.2. Основные символы внутреннего языка имеют вид 15-разрядных двоичных кодов и делятся на идентификаторы, знаки и номера.

Особенность *идентификаторов* состоит в том, что для них транслятор накапливает некоторую информацию, хранящуюся в таблицах. Эта информация ставится в таблицы или извлекается из них при чтении идентификаторов в программе. В связи с этим во внутреннем языке принята признаково-адресная кодировка идентификаторов, при которой три или четыре разряда кода (признаковая часть) отводятся под признаки, отличающие идентификатор данного типа от остальных символов, а двенадцать или одиннадцать разрядов (*идентифицирующая часть*) содержат номер строки таблицы с информацией о данном идентификаторе.

Вся информация о знаках в них самих. *Номер* также изображает самого себя.

Идентификаторы делятся на идентификаторы *констант, меток, рабочих величин, скаляров* и *индексных величин.*

Знаки делятся на операторные знаки и ограничители.

Операторные знаки делятся на знаки: *коммутативных операций, двуместных операций, одноместных операций, условных переходов, переходов с возвратом, возврата, останова, обращения к стандартным подпрограммам, обращения к административной системе, операторов цикла.*

Ограничители делятся на парные скобки: циклов { }. *переключателей* † ‡, *внешних блоков* В В, *индексов* [ ], на *символ конца обращения к стандартной подпрограмме* (, *концевой символ* * и на *алголовские основные символы* **шаг, до, пока.**

2.3. Синтаксис. Синтаксис внутреннего языка дается следующими металингвистическими формулами:
<программа> :: = <нулевая метка> <строка операторов> *
<строка операторов> :: = <оператор> | <строка операторов> <оператор>
<оператор> :: = <непомеченный оператор> | <метка> <оператор>
<непомеченный оператор> :: = <оператор присваивания> | <логический оператор> | <оператор обращения к СП> | <оператор обращения к АС> | <оператор цикла> | <внешний блок>
<внешний блок> В :: = <метка с указанием вида внешней памяти> <строка операторов> В
<оператор цикла> :: = <знак оператора цикла> {<заголовок цикла>{<строка операторов>}
<оператор обращения к АС> :: = <знак обращения к административной системе>
<оператор обращения к СП> :: = <знак обращения к стандартной подпрограмме> <номер подпрограммы> <список параметров> ∇
<список параметров> :: = <пусто> | <список параметров> <метка> | <список параметров> <величина>
<логический оператор> :: = <условный переход> | <безусловный переход> | <переключатель> < | переход с возвратом> | <возврат> | <останов>
<останов> :: = <знак останова>
<возврат> :: = <метка возврата> <знак возврата> <номер возврата>
<переход с возвратом> :: = <знак перехода с возвратом> <метка, куда возвратиться> <метка перехода> <метка возврата>
<переключатель> :: = <метка переключателя> † <скаляр переключателя> <список меток> ‡
<список меток> :: = <метка> | <список меток> <метка>
<безусловный переход> :: = <знак безусловного перехода> <метка перехода>
<условный переход> :: = <переход по признаку> | <переход по сравнению>
<переход по признаку> :: = <знак условного перехода по признаку> <логическая величина> <метка перехода при истинности логической величины>

* Следует заметить, что эти требования, взятые сами по себе, весьма расплывчаты, поэтому читатель не должен воспринимать их буквально. Но он может использовать их как некую канву при анализе описания внутреннего языка.

<переход по сравнению> :: = <знак условного перехода по сравнению> <знак операции отношения> <первая сравниваемая величина> <вторая сравниваемая величина> <метка перехода при истинности отношения>
<оператор присваивания> :: = <коммутативная операция> | <двуместная операция> | <одноместная операция>
<одноместная операция> :: = <знак одноместной операции> <величина-аргумент> <величина-результат>
<двуместная операция> :: = <знак двуместной операции> <величина-аргумент> <величина-аргумент> <величина-результат>
<коммутативная операция> :: = <знак коммутативной операции> <список величин> <величина-результат>
<список величин> :: = <величина> | <список величин> <величина>
<величина> :: = <константа> | <рабочая величина> | <скаляр> | <индексная величина>
<константа> :: = <идентификатор константы>
<рабочая величина> :: = <идентификатор рабочей величины>
<скаляр> :: = <идентификатор скаляра>
<метка> :: = <идентификатор метки>
<индексная величина> :: = <идентификатор индексной величины> [<индекс>]
<индекс> :: = <зависимость от параметров циклов> <переменная составляющая> <постоянная составляющая>
<постоянная составляющая> :: = <константа>
<переменная составляющая> :: = <нулевая константа> | <скаляр> | <рабочая величина>
<зависимость от параметров циклов> :: = <пусто> | <зависимость от параметров циклов> <параметр цикла> <шаг по параметру цикла>
<параметр цикла> :: = <скаляр>
<шаг по параметру цикла> :: = <константа> | <скаляр>

Для краткости в этом синтаксическом описании опущены металингвистические формулы для тривиальных переобозначений типа

<величина-результат>< величина>

Синтаксис заголовков циклов практически совпадает с алголовским синтаксисом с заменой выражений на строки операторов, вычисляющих эти выражения, если только последние не являются простыми переменными.

2.4. Семантика. Подбор содержательных названий для металингвистических переменных делает ненужным объяснение семантики большинства формул, поэтому комментарии будут сделаны лишь к отдельным конструкциям.

2.4.1.Выделение промежуточных результатов, возникающих при трансляции выражений, в особый класс рабочих величин служит примером сохранения информации о программе высшего уровня, способствующей более эффективной экономии памяти. Тем же целям служит имеющаяся во внутреннем языке классификация меток – на внутренние и внешние. *Внутренние метки* используются в передачах управления, возникающих при программировании условных выражений и покомпонентных действий над многомерными массивами. Появляющиеся при этом разветвления и циклы (так называемые *внутренние циклы)* в отличие от общего случая имеют упрощенную структуру, и учет этого обстоятельства также позволяет улучшить экономию выражений и памяти. Все невнутренние метки являются *внешними.*

2.4.2 .Структура индексных величин полностью приспособлена к программированию циклов. В особый случай выделена линейная зависимость адреса индексной величины от параметров циклов, что позволяет применять для вычисления адреса индекс-регистр и переадресацию. Более точно, если обозначить через m_0 начальный адрес массива m, через v и c – переменную и постоянную составляющие, а через $i_1h_1, \ldots, i_nh_n$ –

$$A = \sum_{k=1}^{n} i_k h_k + v + c + m_0$$

параметры циклов и шаги, образующие зависимость от параметров циклов, адрес A индексной величины $m[i_1h_1 \ldots i_nh_nvc]$ будет вычисляться по формуле

Эта форма представления индексного выражения называется *канонической формой индекса*. Присутствие отличной от нуля переменной составляющей v означает, что вычисления адреса A производится, помимо вычисления по данной формуле, также и путем формирования, т. е. вычисленное отдельно значение целой величины v прибавляется к A специальным *оператором формирования*. Оператор формирования во внутреннем языке синтаксически оформлен в виде одноместной операции suv , где s – операторный знак; u – аргумент и v – результат. При его выполнении значение величины u прибавляется к адресу той индексной величины, переменная составляющая которой есть v.

2.4.3. По отношению к начальному адресу m_0 индексные величины по особому признаку делятся на два типа: *массивы* и *дыры*, которые, в свою очередь, делятся на *динамические массивы* и *формальные параметры.* Массивы соответствуют статическим массивам АЛГОЛа . Для них m_0 определяется в процессе трансляции и ставится в рабочую программу заранее. Для *дыр* m_0 включается в переменную составляющую *v* и, вычисляясь во время выполнения рабочей программы, ставится в адрес дыры оператором формирования *siv*.

Для динамических массивов *u* вычисляется операторами обращения к административной системе. Дыры – формальные параметры – происходят от формальных параметров, подставляемых именем..

Для них *u* вычисляется с помощью специальных операторов *засылки идентификатора*, синтаксически также оформленных в виде одноместной операции *ψIu*, где *ψ* – операторный знак, *I* – аргумент и *u* – результат. При его выполнении адрес величины *I* если *I* – массив, то берется начальный адрес) присваивается величине *u*. Операторы засылки идентификатора появляются в результате программирования операторов процедур и сообщают формальному параметру – дыре имя подставляемого на его место идентификатора (*затычки*).

2.4.4. Как видно из синтаксиса, во внутреннем языке полностью сохранена структура заголовков операторов циклов. Это также пример сохранения во внутреннем языке информации об АЛЬФА-программе. Программирование заголовков циклов сразу в терминах машинных команд с учетом информации, содержащейся в операторных знаках, позволяет существенно повысить качество программирования операторов циклов. Информация, содержащаяся в операторных знаках, накапливается в процессе анализа заголовков и тел операторов циклов из двоичных признаков, характеризующих следующие особенности циклов:

1) есть или нет параметр у цикла;
2) есть или нет выход из цикла по оператору перехода;
3) есть или нет присваивание параметру помимо заголовка цикла;.
4) вещественный или целый тип параметра;
5) возможно или нет вычисление индексов переадресацией;
6) обычный цикл или внутренний;
7) есть или нет использование параметра в цикле помимо вхождений в индексные выражения.

2.4.5. Операторы присваивания возникают при трансляции выражений АЛЬФА-языка. К одноместным операциям относится передача числа, преобразование типа, вычисление абсолютной величины, тригонометрические функции, отрицание и т. д. – всего 31 операция. К двуместным операциям относятся 12 некоммутативных бинарных операций: сдвиг, степень, арифметические отношения и т. д.; к коммутативным – целые сложение и умножение, дизъюнкция, конъюнкция и сложение по mod 2.

Особенность коммутативных операций, называемых также мультикомандами, в том, что они имеют произвольное число аргументов (от 2 до 8). Количество аргументов указывается в двоичной шкале, помещаемой в операторном знаке. Кроме того, для первых четырех коммутативных операций в шкале указывается, берется ли аргумент со знаком прямой (сложение, умножение) или обратной (вычитание, деление) операции. Произвольное количество аргументов объясняется тем, что в пределах одной мультикоманды объединяются все коммутативные операции одного уровня старшинства, содержащиеся в исходном выражении в АЛЬФА-языке. Так, оператор

$$r := x \uparrow y - a \times sin\ (b) / (a - b) + r - K$$

будет во внутреннем языке иметь вид

↑		x	y	t_1			
sin		b	t_2				
	$+_{01}$		a	b	t_3		
	$\times_{001}$		a	t_2	t_3	t_4	
	$+_{0101}$		t_1	t_4	r	K	r

В виде индекса у операторных знаков выписана шкала аргументов мультикоманды. Использование мультикоманд – еще один пример сохранения информации об АЛЬФА-программе. Оно создает некоторые преимущества при чистке циклов, обработке индексов и экономии выражений.

2.4.6. Переключатели во внутреннем языке соответствуют описаниям переключателя в АЛЬФА-языке. При переходе по переключателю (на *S[i]*) значение индекса *i* засылается в скаляр переключателя, а затем происходит переход по метке переключателя *S*. Последующий выбор метки перехода – из списка меток, заключенных в скобки переключателя, – производится по значению скаляра переключателя.

2.4.7. Скобками внешних блоков ограничены куски программы, которые в рабочей программе должны быть помещены во внешнюю память. Информация о виде внешней памяти при переводе на внутренний язык

переносится из описаний в основной символ, стоящий справа от открывающей скобки внешнего блока и синтаксически оформленный в виде метки.

2.4.8. Таблицы внутреннего языка делятся на таблицы констант, массивов, дыр и возвратов.

В *таблице констант* помещаются значения скалярных величин, введенных транслятором для обозначения констант. Специальной шкалой в таблице помещаются константы, являющиеся начальными значениями компонент массивов (так называемых *константных массивов*). В *таблице массивов* помещены длины статических массивов и ссылки на позицию в таблице констант начальной компоненты константных массивов. В *таблице дыр* для каждой дыры – формального параметра – приведен список затычек – имен подставляемых на его место величин. В *таблице возвратов* для каждого возврата содержится список всех меток, на которые можно вернуться по данному возврату.

2.5. Промежуточный язык. Надо сказать, что внутренний язык не остается неизменным во время фазы перевода. Мало того, если стать на формальную точку зрения, то придется сказать, что почти каждый из первых 14 блоков АЛЬФА-транслятора имеет свой собственный язык, на котором записывается транслируемая программа, и лишь язык блока 14 совпадает с вышеописанным внутренним языком. Для блоков, выполняющих главным образом оптимизирующие преобразования (8–13), эта разница между языками не настолько существенна, чтобы ее обсуждать, однако стоит упомянуть о некоторых существенных различиях между языком, здесь описанным, и внутренним языком на уровне первых семи блоков. Эти различия состоят не столько в синтаксисе операторов, сколько в кодировке основных символов.

Если сравнивать АЛЬФА-язык и внутренний язык, то видно, что для АЛЬФА-языка характерно контекстное понимание смысла операций и идентификаторов. Смысл идентификатора устанавливается из описания, а смысл операции определяется путем анализа ее операндов (например, знак × в АЛЬФА-языке может пониматься более чем тридцатью различными способами). В случае внутреннего языка каждый операторный знак и в значительной степени каждый идентификатор несут в себе всю информацию, необходимую для их однозначного понимания и последующих реализаций машинных командах. Таким образом, при переводе на внутренний язык происходит перекачка информации из описаний в величины и из величин в знаки операций. Для хранения перекачиваемой информации признаковая часть идентификаторов и основных символов на уровне первых семи блоков транслятора увеличена и составляет 10 – 11 разрядов вместо трех-четырех при общей длине основного символа в 22 разряда.

Чтобы отличать 15-разрядную кодировку от 22-разрядной, последня на уровне блока 7 называется промежуточным языком. Перекодировка промежуточного языка во внутренний производится блоком 8 АЛЬФА-транслятора. Из других отличий промежуточного языка от внутреннего стоит лишь упомянуть об изображении индексов и операторов формирования. В индексах вместо канонической формы стоит вся цепочка команд, вычисляющих индексное выражение. Операторы формирования менее детализированы и имеют синтаксическую *форму* $\chi(\alpha, M)$ и обозначают следующее действие: прибавить значение величины α ко всем вхождениям адреса массива M в программу, расположенным правее данного оператора, либо до конца программы, либо до следующего χ с тем же самым M.

3. СТРУКТУРА БЛОКОВ

3.1. Транзитные и местные массивы. Ячейки связи. При проектировании АЛЬФА-транслятора уделялось особое внимание обеспечению однотипности структуры блоков транслятора и установлению как можно более слабой связи между ними. Это позволило упорядочить передачу информации между блоками и организовать их параллельную разработку.

Блоки не имеют ведущей программы, так что каждый блок по окончании работы сам вызывает следующий. Для этого программы всех блоков начинаются с ячейки 0001. В первых двух ячейках стоят команды считывания следующего блока с ленты. Первая по выполнению команда каждого блока находится в ячейке 0003. По окончании работы очередного блока передается управление на 0001. Команды в ячейках 0001 и 0002 считывают в память следующий блок, который тут же начинает выполняться с команды в ячейке 0003.

Вся основная информация о транслируемой программе передается от блока к блоку в так называемых *транзитных массивах*. Были приняты меры к тому, чтобы количество транзитных массивов было бы по возможности меньшим. Основные транзитные массивы:

программа (или *основной массив*);
таблица констант со шкалой;
таблица массивов;
таблица возвратов;

таблица дыр;
общий список.

Кроме того, на каждой из фаз трансляции появляется ряд транзитных массивов, специфических для этих фаз. Однако сразу от одного блока к другому передается не более десяти транзитных массивов. Назначение общего списка будет объяснено ниже. Помимо транзитных массивов каждый блок организует в памяти *местные массивы*, т. е. массивы, используемые только в пределах одного блока.

Блоки АЛЬФА-транслятора составлены таким образом, чтобы не существовало никаких неявных предположений о фактическом размещении в памяти транзитных массивов. Вся информация о размещении в машине транслируемой программы указывается в явном виде в специальных *ячейках связи*, занимающих последние 64 ячейки оперативной памяти. Ячейки связи делятся на три категории: *фиксаторы, счетчики* и *мемориалы*. В фиксаторах хранятся координаты транзитных массивов, в счетчиках накапливается число использованных к данному моменту идентификаторов того ли иного вида. Имеется также особый счетчик, указывающий номер очередного работающего блока. Вывод этой ячейки на пульт позволяет следить за работой транслятора. В мемориалах хранится разнообразная транзитная информация, задающая режим работы тех или иных блоков (например, есть в задаче процедуры или нет, понадобится или нет административная система и т. п.) или способствующая более тщательному распределению оперативной памяти.

Как уже указывалось, машина имеет сравнительно высокую скорость выполнения логических операций над оперативной памятью. Чтобы заставить это качество машины работать на транслятор, необходимо как можно полнее использовать оперативную память. Для. этого в АЛЬФА-трансляторе было реализовано своего рода полудинамическое распределение оперативной памяти. Оно состоит в том, что если некоторый блок A должен заново образовать или пополнить некоторый массив M, то блок, предшествующий A, анализируя транслируемую программу, заранее или точно подсчитывает, или приближенно оценивает длину массива и сообщает ее блоку A в некотором мемориале. Эта информация позволяет блоку A перед началом своей работы аккуратно распределить оперативную память.

3.2. Общий список. Такой априорный подсчет длины новых массивов не всегда, однако, возможен, и транслятору приходится иметь дело с массивами, точная длина которых становится известной только после их заполнения. В таких случаях неизбежно приходится оставлять зазор в памяти "на рост". Положение затрудняется тем, что неопределенно наращиваемых массивов может быть несколько. Для накопления таких массивов в АЛЬФА-трансляторе применяется списочная структура их хранения в памяти. Механизм списочной структуры описывается ниже на абстрактном примере.

Пусть необходимо выписать из программы информацию n сортов $I_1, I_2, \ldots, I_n$. Эта информация будет накапливаться ячейка за ячейкой в некотором массиве G. Адрес очередной свободной ячейки массива G хранится в счетчике r. Сопоставим информациям $I_1, \ldots, I_n$ выходные ячейки $i_1, \ldots, i_n$, в которых будут храниться ссылки на начало размещения информации данного типа в массиве G.

Пусть при чтении программы встретилась порция информации $_{Ik}$. Если $i_k = 0$, то это значит, что информация типа I_k встретилась впервые. В этом случае r направляется в i_k, а порция информации переписывается в G с соответствующим изменением r. Будем для определенности считать, что порции информации делятся на кванты, скажем, по 15 разрядов каждый, так что в одной ячейки помещается три кванта. Если порция информации содержит число квантов, не кратное трем, то последняя ячейка массива G, содержащая порцию, будет не до конца заполнена; если порция содержит $3m$ квантов, то вслед за порцией искусственно резервируется еще одна пустая ячейка.

Рассмотрим теперь случай, когда $i_k \neq 0$. Это значит, что в G уже есть информация типа I_k. По содержимому i_k находим ссылку на начало информации I_k в G, которая просматривается о конца первой порции. Если эта порция кончается частично заполненной или полностью пустой ячейкой, то в эту пустую позицию ставится значение r в качестве *ссылки на продолжение*, а порция информации выписывается в G, начиная с места, указанного r. Если же порция заканчивается ранее поставленной ссылкой на продолжение, то поиск конца информации продолжается по этой ссылке. Считается, что ссылка синтаксически отличима от квантов информации. На схеме 2 изображен результат применения такой процедуры для простого примера.

Таким образом, хотя информации пишутся в G разрозненными кусками в порядке их обнаружения в программе, система ссылок позволяет свести воедино всю информацию одного типа. В АЛЬФА-трансляторе роль массива G, одного для всех целей, играет уже упоминавшийся общий список. По ходу работы транслятора общий список время от времени перетряхивается и находящаяся в нем информация либо выбрасывается за ненадобностью, либо упорядочивается.

Дополнительным средством экономии оперативной памяти является запись на барабан в конце работы блока всех транзитных массивов, которые не потребуются следующему блоку (основной массив помещается на барабан всегда).

3.3. Строение блоков. Таким образом, каждый блок транслятора обычно состоит из трех частей: формирующей, основной и заключительной. Формирующая часть распределяет память для местных и транзитных массивов. Местные массивы имеют фиксированную или заранее вычисляемую длину. Часть памяти в случае необходимости резервируется на рост общего списка. Вся оставшаяся часть оперативной памяти (иногда очень незначительная) отводится под *обменный массив*, предназначенный для связи с .основным массивом (программой) (рис. 1).

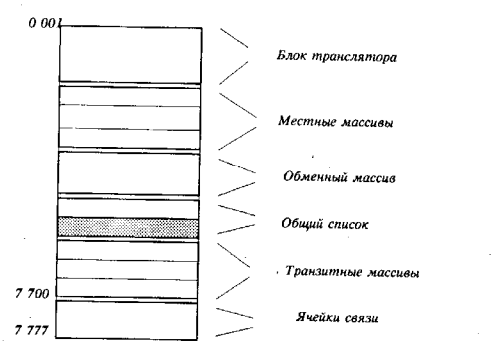


Рис. 1. Типичное распределение оперативной памяти.

После распределения памяти формирующая часть настраивает блок на выработанное распределение памяти. Как правило, место, занимаемое формирующей программой, стоящей обычно в конце блока, тоже распределяется для работы основной части. Общий список заполняется снизу вверх. Иногда наверху резервного пространства общего списка накапливается какой-нибудь массив информации, для которого почему-либо нецелесообразно организовывать списочную структуру.

После окончания работы основной части блока заключительная часть записывает ненужные транзитные массивы на барабан, перетряхивает, если необходимо, общий список и вызывает следующий блок.

3.4. Обменный массив. В заключение укажем на некоторые особенности организации обменного массива (рис. 2). Основной массив хранится на барабане, логически трактуемом как циклическая память. Старая программа возможно максимальными кусками последовательно читается с барабана в нижнюю часть обменного массива (начиная от точки чтения). Почти все блоки транслятора работают по принципу "чтение – обработка – запись".

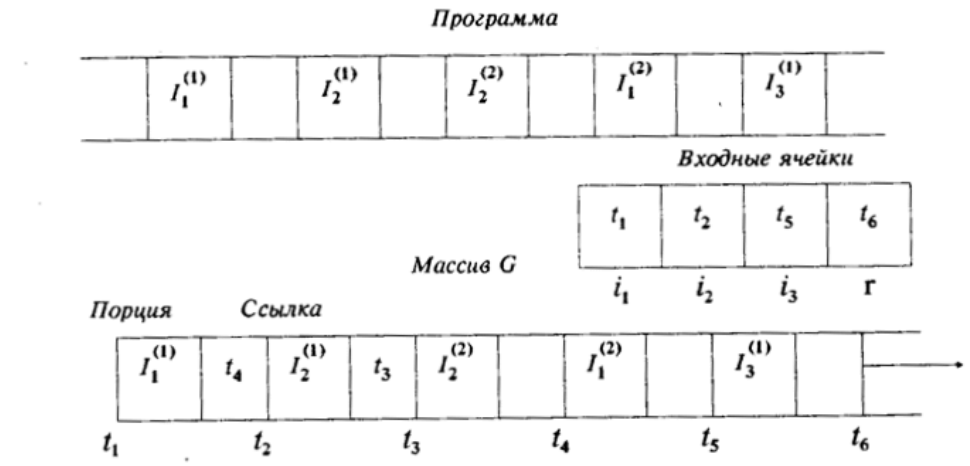


Схема 2. Использование общего списка.

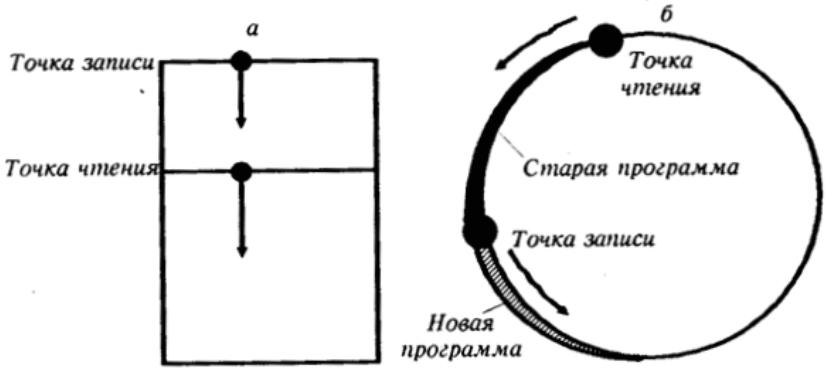


Рис. 2. Организация обмена массива:
а — обменный массив; *б*— основной массив.

Переработанная программа пишется в верхнюю часть обменного массива, начиная от точки записи. Получается так, что в процессе работы точка записи догоняет убегающую от нее точку чтения. Как только точка записи догонит точку чтения или точка чтения дойдет до конца обменного массива, порция новой программы пишется в основной массив, начиная от конца старой программы. Таким образом, в основном массиве точка чтения также догоняет точку записи. Начальное расстояние между точками записи и чтения в обменном массиве выбирается для каждого блока индивидуально в зависимости от степени разбухания или сжатия информации в транслируемой программе при работе блока.

4. РАЗНЫЕ ДЕТАЛИ

4.1. Функция расстановки. Как уже отмечалось, работа транслятора организована таким образом, чтобы обеспечить линейность просмотра и локальность переработки транслируемой программы. Большинство алгоритмов трансляции носит локальный характер, однако в некоторых алгоритмах перекодировки и оптимизации приходилось проявлять некоторую изобретательность, чтобы достичь линейности просмотра и локальности переработки. В том случае, когда такие алгоритмы сводились к задаче идентификации разных элементов в некотором наборе, применялся метод так называемых функций расстановки.

Задача идентификации разных элементов в наборе ставится следующим образом. Пусть дано некоторое множество элементов *M*. Элементами этого множества будем считать двоичные коды. Длина этих кодов может быть различна, но для простоты будем считать, что каждый код помещается в одной ячейке памяти. Пусть, наконец, имеется последовательность (возможно, с повторениями) $A = =N\{a_1DN, ...,a_2\}$ элементов из *M*, которую нужно без повторений расставить в ячейках расстановочного поля с адресами $L + 1,... ...,L + n$ $(n \geq k)$. Существенно здесь то, что *n* много меньше мощности *M*. Это не позволяет применить адресную кодировку элементов из *M*, при которой *A* в расстановочном поле размещалось бы тривиально. Очевидно, что переработка элементов из *A* носит, по существу, нелокальный характер: чтобы распорядиться очередным элементом a_s, нужно знать, содержится ли a_s среди элементов $a_1, ...,a_{s-1}$.

Простейшим подходом к решению задачи является расстановка элементов из A в расстановочном поле подряд. При этом каждый новый элемент сличается поочередно со всеми уже находящимися в расстановочном поле. При таком подходе общее количество действий, затрачиваемое на расстановку, пропорционально k^2 (если в качестве оценки числа действий брать случай, когда в A нет повторений).

Некоторое усовершенствование механизма сличения нового элемента с уже находящимися в расстановочном поле позволяет сделать количество действий пропорциональным $k \times ln\ (k)$ (см., например, [1]).

Оказывается, что в этой задаче количество действий можно сделать пропорциональным k, если организовать работу следующим образом. Предположим, что на элементах множества M определена некоторая целочисленная *функция расстановки f(a),* такая, *что* $L + 1 \le f(a) \le L + n$. Предположим далее, что перед началом работы расстановочное поле пусто. Процедура помещения очередного элемента a_s из A носит следующий характер. Вычисляется $f(a_s) = \mu$. Если ячейка с адресом пуста, то это значит, что a_s встретилось в A впервые, и тогда a_s направляется в ячейку μ. Если μ оказалась занятой элементом, тождественным с a_s, то это значит, что a_s встретилось в A повторно, и поэтому a_s игнорируется. Если μ оказалась занятой элементом, не совпадающим с a_s, то адрес увеличивается на единицу (если μ равнялась $L + n$, то вместо прибавления единицы берется μ, равное $L + 1$), после чего процесс сличения a_s с содержимым μ повторяется. Так как $n \ge k$, процесс сличения обязательно оборвется. Справедливость предположений, сделанных при описании механизма сличения, проверяется индукцией по s. Легко видеть, что если взять $f(a) + 1$, получится вышеописанный способ расстановки элементов A, дающий квадратичный перебор.

Очевидно, что эффективность применения функции расстановки зависит от того, насколько равномерно значения $f(a)$ "разбросаны" по интервалу $[L + 1, L + n]$ при вычислении на элементах $a_1, \ldots, a_k$, и от того, насколько велик запас n по отношению к k. Статистические эксперименты показали [2], если считать, что значения $f(a)$ равномерно распределены на случайных последовательностях $\{a_1, \ldots, a_k\}$, то достаточно взять $n \sim 1{,}5k$, чтобы получить не более двух сличений с содержимым ячеек расстановочного поля на один элемент a_s. Это дает общее время работы, пропорциональное k. При $n = k$ получится время работы, пропорциональное $k \times ln(k)$.

Наиболее простым способом вычисления функции расстановки $f(a)$ является, по-видимому, такой, когда n берется равным 2^m, а $f(a)$ вычисляется так: двоичный код a режется на куски длиной по m разрядов, которые складываются друг с другом по $mod\ 2^m$. При некоторых весьма общих предположениях о статистической независимости складываемых кусков этот способ оптимален с вероятностной точки зрения и дает распределение, близкое к равномерному даже при небольшом числе складываемых кусков.

В АЛЬФА-трансляторе функция расстановки применяется при переводе идентификаторов АЛЬФА-программы, при экономии совпадающих выражений и при экономии констант.

4.2. Режимы работы транслятора. Одним из выражений "смешанной стратегии" программирования является анализ транслируемой программы на полное отсутствие в ней тех или иных синтаксических единиц. В этом случае блоки транслятора, ответственные за программирование этих единиц, можно обойти, что дополнительно повысит скорость трансляции. В АЛЬФА-трансляторе таким образом могут обходиться блоки 6 и 7 (программирование процедур), основная часть блока 9 и блок10 (программирование операций над многомерными массивами).

Кроме того, транслятор может отключать некоторые алгоритмы оптимизации, когда в них нет необходимости или когда их полная реализация невозможна из-за переполнения некоторых служебных массивов, используемых при выполнении оптимизирующих преобразований, например глобальная экономия памяти [IX] не производится, если АЛЬФА-программа не содержит динамических массивов и влезает в оперативную память без экономии памяти или если таблицы, возникающие при глобальной экономии памяти, не помещаются в оперативной памяти машины. В этом случае отключается основная часть блока 21 и блок 22.

Блок экономии выражений (№ 14) искусственно сокращает размер участка, в пределах которого выполняется экономия, если этот участок слишком велик. Блок чистки циклов (№ 12) производит чистку только во внутренних циклах, если объем команд, выносимых за пределы циклов, превышает допустимые размеры.

Можно было бы назвать еще довольно много мелких деталей, рассеянных по всему транслятору и направленных на то, чтобы режим работы транслятора как можно теснее "прилегал" к особенностям – как приятным, так и неприятным – транслируемой программы.

4.3. Формальный контроль транслируемой программы. При построении АЛЬФА-транслятора были приняты специальные меры к тому, чтобы не допустить проникновения формальных ошибок из АЛЬФА-программы в рабочую программу. Под формальными ошибками понимаются ошибки в синтаксисе и такие семантические ошибки, наличие которых вызывает зацикливание транслятора или делает невозможным разумное продолжение трансляции или выполнение рабочей программы. Универсальным и единственным средством выравнивания формальных ошибок была явная проверка условий, при которых возникает ошибка, с последующей сигнализацией в виде передачи управления на команду СТОП (*контрольный останов*). Для лучшей идентификации ошибки всюду, где это имело смысл или было возможно, для каждого условия наличия ошибки отводился свой контрольный останов. Типичным приемом было замыкание контрольными остановами цепи проверки несовместных логических условий. Пусть, например, в некоторый момент анализируется идентификатор скаляра, рабочей ячейки, константы или массива. Для анализа ситуации строилась цепочка четырех сравнений, передающих управление на обработку величины данного вида или на продолжение анализа. Последнее же, четвертое, сравнение (строго говоря, уже ненужное) передавало управление либо на обработку массивов, либо на контрольный останов. Такие избыточные контрольные остановы сыграли неоценимую роль при отладке транслятора и оказались во многих случаях полезными при рабочей эксплуатации в качестве сигнализаторов сбоев машины. Действительно, транслятор, обильно оснащенный контрольными остановами (всего их около 200), приобрел своеобразные черты "абсолютной вычислительной неустойчивости". При программировании первых 500 задач произошел только один случай проникновения синтаксической ошибки типа "**если** a : = 1 **то** B **иначе** C" в рабочую программу и не более пяти случаев, когда сбой машины в процессе трансляции не фиксировался бы контрольными остановами.

В АЛЬФА-трансляторе отсутствует предварительный синтаксический контроль АЛЬФА-программы. Однако были предприняты меры для возможно более полной и ранней сигнализации о формальных ошибках во всей программе. Для этого 1-й блок, который вводит АЛЬФА-программу с перфокарт, подсчитывает баланс скобок, проверяет их взаимное расположение и производит частичный синтаксический контроль, сопоставляя АЛЬФА-программу с так называемой матрицей сочетаемости. Порядок этой логической матрицы $|.|\ m_{ij}\ ||$ равен числу основных символов АЛЬФА-языка (все буквы и все цифры считаются за один символ соответственно); $m_{ij} = 1$, если i-й основной символ может по синтаксису предшествовать j-му символу, и $v_{ij} = 0$ – в противном случае. При обнаружении в АЛЬФА-программе несочетаемой пары печатается контекст программы, содержащий эту пару (вправо и влево от пары по 10 символов), без немедленной остановки транслятора. Кроме того, блок 3 транслятора выдает в конце своей работы список всех неописанных идентификаторов.

Анализ ошибок, обнаруженных в первых 500 транслируемых программах, показал, что блоки 1 и 3 выявляют 93 % всех формальных ошибок. Остальные формальные ошибки выявляются транслятором индивидуально. Некоторые типы формальных семантических ошибок, правда, весьма редких (не более 2 %), обнаруживаются глубинными блоками – вплоть до 12-го.

4.4. Количественные ограничения. Под этим термином понимаются ограничения на класс транслируемых программ, вызываемых ограниченным объемом памяти машины. Проблема состоит в том, что при трансляции происходит разбухание информации: например, если АЛЬФА-транслятор строит программу длиной в 3000 машинных слов, то объем промежуточной информации может достигать 6000 – 7000 слов. Бесконтрольное использование внешней памяти для хранения промежуточной информации может приводить к неоправданному замедлению трансляции. Поэтому при организации АЛЬФА-транслятора, как об этом уже говорилось выше, был сделан упор на максимальное использование оперативной памяти. Этот подход имеет, однако, свою обратную сторону, состоящую в увеличении и разнотипности количественных ограничений. Полудинамическое распределение памяти не решает вопроса полностью, так как существует довольно большое количество мелких промежуточных таблиц или рабочих полей, для которых аккуратный подсчет их размера или в принципе невозможен, или излишне загромоздил бы транслятор. Фиксация размера таких таблиц и рабочих полей создает разнообразные количественные ограничения.

Основная проблема при формулировании количественных ограничений состоит в их сбалансированности, с тем, чтобы сократить число таких задач, которые проходили бы через транслятор по общим показателям, но застревали бы на каком-нибудь узком месте из-за непродуманного размера какого-либо рабочего поля. Кроме того, при анализе количественных ограничений надо было выделить такие ситуации (чтобы отдать им предпочтение в смысле увеличения объема отводимой памяти), к которым программист не может легко приспособиться путем редактирования АЛЬФА-программы.

Вся эта работа по анализу и балансировке количественных ограничений проводилась для АЛЬФА-транслятора эмпирически, в основном при опытной эксплуатации. Из 500 транслированных программ порядка 30 – 40 послужили причиной изменений транслятора, направленных на ослабление тех или иных количественных ограничений.

Сейчас можно, по-видимому, считать, что подавляющее большинство программ объемом до 3000 команд будет проходить через транслятор. Однако в отношении задач большого объема количественные ограничения АЛЬФА-транслятора будут, вероятно, сказываться довольно часто.

4.5. Некоторые дополнительные преимущества многоблочной структуры и наличия универсальных оптимизирующих алгоритмов. Многоблочная структура и обилие оптимизирующих алгоритмов имеют очевидный недостаток, состоящий в отяжелении транслятора и понижении его быстродействия. Однако, поскольку так или иначе транслятор должен обладать этими свойствами, они создают дополнительные преимущества для организации работы транслятора.

Наличие многократных просмотров транслируемых программ позволяет, во-первых, рассредоточить сложные алгоритмы трансляции по нескольким блокам, чтобы избежать излишней напряженности с оперативной памятью в основном блоке, ответственном за данный алгоритм. Типичный пример в этом отношении – это программирование циклов и индексных выражений [VII]. Основными блоками программирования циклов и вычисления переменных адресов являются блоки 16, 17 и 18. Однако значительную часть работы по подготовке циклов и индексных выражений к программированию удалось переложить на предыдущие блоки. Наиболее прямой эффект такого расчленения – повышение скорости работы транслятора, поскольку вспомогательные и подготовительные операции, выполняемые "мимоходом" предыдущими блоками без заметного увеличения времени их работы, позволяют в некоторых случаях упростить структуру основных блоков и повысить их быстродействие.
Другим полезным качеством многоблочной структуры оказалась ее эластичность, позволившая при отладке АЛЬФА-транслятора вносить довольно серьезные изменения в блоки без изменения общей структуры транслятора. Действительно, расчлененность блоков позволяла иногда полностью заменить старый вариант блока на новый, ничего не меняя в остальных и, наоборот, обнаруживавшиеся трудноустранимые недочеты работы одного блока удавалось в некоторых случаях компенсировать внесениями поправок в другие блоки. Надо, однако, добавить, что полностью устранить все недочеты алгоритмов трансляции не удалось, что потребовало [II] сформулировать дополнительные мелкие ограничения на входной язык.

Своеобразным преимуществом наличия мощных оптимизирующих алгоритмов явилась возможность в некоторых случаях применять нарочито упрощенные алгоритмы трансляции в расчете на будущую оптимизацию. Использование "запаса мощности" алгоритмов оптимизации – характерная особенность АЛЬФА-транслятора. Например, наличие алгоритмов анализа процедур позволяет сводить функции-выражения и функции-процедуры к процедурам общего типа, не снижая качества трансляции. Присутствие алгоритмов экономии памяти позволяет не заботиться об экономном введении обозначений для промежуточных величин, возникающих при трансляции. Ориентация на чистку циклов позволила строить команды вычисления индексных выражений и последующего формирования и изменения адресов простейшим способом, когда все эти управляющие команды выписываются в самом внутреннем цикле непосредственно перед командой, содержащей обрабатываемый переменный адрес. При чистке циклов управляющие команды автоматически выносятся настолько далеко из внутренних циклов, насколько это возможно.

5. ОРГАНИЗАЦИЯ РАЗРАБОТКИ АЛЬФА-ТРАНСЛЯТОРА

5.1. Замысел. При проектировании АЛЬФА-транслятора мыслилась некоторая идеальная организация работы, которая должна была бы послужить своевременному построению качественного транслятора. Предполагалось, что разработка будет состоять из следующих этапов.

1. Разработка проектного задания и проблемного плана. Уточняются технические условия на систему, находятся общие контуры системы, намечаются основные подходы к построению алгоритмов трансляции и оптимизации и формулируются задания разработчиком.

2. Расчлененная разработка алгоритмов трансляции и оптимизации. Разработчики, действующие в значительной степени независимо друг от друга, создают основные алгоритмы работы транслятора. Работа расчленения в большей степени "по вертикали", т. е. по типам транслируемых операторов входного языка и видам оптимизации, нежели "по горизонтали", т. е. о стадиям работы транслятора.

3. "Сбойка" разработанных алгоритмов и компоновка транслятора. Производится сопряжение и объединение разработанных алгоритмов, уточняется структура транслятора и расчленяется работа транслятора по буквам.

4. Черновая "прокрутка" транслятора. На нескольких простых примерах воспроизводится воображаемая работа транслятора в соответствии с запланированными функциями блоков. Во время черновой прокрутки вырабатывается по возможности единый стиль организации блоков, уточняется структура и содержание информации, передаваемой от блока к блоку.

5. Разработка логических схем блоков. Работа опять расчленяется, но уже не по вертикали, а по горизонтали – по отдельным блокам транслятора.

6. Тщательная прокрутка транслятора. На одном более-менее содержательном примере воспроизводится работа транслятора в соответствии с содержанием логических схем блоков. Цель прокрутки – обнаружить наиболее грубые ошибки в схемах и особенно ошибки в согласовании работы блоков.

7. Программирование блоков и черновая сепаратная отладка на простых тестах.

8. Отладка транслятора. Тщательно отлаживаются отдельные блоки и комплексно отлаживается весь транслятор на системе расширяющихся тестов, начиная с самого простого, проверяющего только механизм передачи основной информации от блока к блоку, и кончая первой содержательной задачей.

9. Опытная эксплуатация транслятора примерно на ста производственных задачах.

5.2. Реализация. При разработке АЛЬФА-транслятора эта организация была реализована далеко не в полной мере. Представляет некоторый интерес анализ причин, которые привели к разрыву между планируемым и фактическим ходом работ.

Первый этап прошел, по-видимому, достаточно успешно. Он был несколько затянут и длился с сентября 1960 г. по февраль 1961 г.. вместо возможных двух-трех месяцев. Это, однако, было вызвано посторонними причинами, не связанными с существом дела. Результаты первого этапа (проектное задание и проблемный план) были хорошо документированы и не содержали существенных ошибок. Наиболее серьезным просчетом была недооценка трудностей, стоявших перед разработчиками задач (были занижены в три раза длина транслятора и в два раза сроки его изготовления).

Второй этап (март–октябрь 1961 г.) был пройден в целом удачно. Большинство научных проблем, связанных с разработкой новых алгоритмов транслятора и оптимизации, были успешно разрешены именно в этот период. Недостаток второго этапа, однако, состоит в том, что решение некоторых проблем, казавшихся сначала чисто техническими (например, программирование выражений), было отложено "на потом". Это привело к тому, что, когда на более поздних этапах эти проблемы всплыли, их пришлось решать в жестких рамках уже принятых решений по другим вопросам.

Третий этап (ноябрь 1961 – январь 1962 гг.) был, вероятно, самым ответственным. Именно на этот период падает максимум объема информации, циркулировавшей между разработчиками. Главная проблема – правильно оценить алгоритмическую сложность реализации тех или иных функций, возлагаемых на отдельные блоки транслятора. Анализ дальнейшей работы над транслятором показал, что большинство решений, принятых на третьем этапе, было правильным, особенно в отношении окончательного отбора алгоритмов оптимизации и установления общей структуры транслятора. В то же время были допущены серьезные просчеты в оценке сложности реализации алгоритмов программирования выражений и операций над многомерными массивами и комплексными величинами.

Основным недостатком четвертого этапа (январь–февраль 1962 г.). была нечеткость и поверхностность его проведения. Это привело к тому, что были упрощены возможности более точно представить себе характер работы отдельных блоков транслятора и откорректировать, если нужно, алгоритмы трансляции еще до того, как они будут воплощены в логических схемах и программах. В то же время именно на этом этапе удалось продумать и упорядочить передачу информации при трансляции между блоками, определить распределение информации между запоминающими устройствами машины, найти и зафиксировать единый стиль конструирования блоков.

Самым крупным недостатком в работе над АЛЬФА-транслятором на последующих этапах была невозможность обеспечить равномерность в конструировании, программировании и отладке блоков. В значительной степени эта неравномерность объяснялась неопытностью многих разработчиков и разнобоем в степени их профессиональной подготовки[1] , а также различием в объеме работы, выпавшей на долю составителей блоков. Неравномерность разработки усугублялась атмосферой спешки и возникавшей время от времени необходимостью преодолевать непредвиденные ранее трудности реализации отдельных блоков. Наиболее трудными для реализации оказались блок программирования выражений, который уже после составления логических схем первого варианта, не поместившегося в оперативной памяти, пришлось искусственно разбивать на два блока, а также блок 10 программирования операций над массивами, составление и отладка которого были закончены только в 1966 г., уже через полтора года после запуска основной части АЛЬФА-транслятора, позволяющей транслировать алголовские программы.

* Одиннадцать авторов транслятора – все с университетским образованием – к концу 1962 г. имели следующий стаж работы в программировании: один – 8 лет, один – 6 лет, один – 5, пятеро – 3, двое – 2, один – меньше года.

Неравномерность разработки блоков не позволила осуществить прокрутку транслятора в полной мере. Однако на примере тех 18 блоков, для которых прокрутка была проведена, ее эффективность была несомненна, по крайней мере для тех условий, в которых создавался АЛЬФА-транслятор.

Сепаратная отладка блоков была проведена недостаточно тщательно, что прежде всего сказалось на ходе опытной эксплуатации. Главная причина этому – несовершенство тестов и большая трудоемкость их подготовки. Неравномерность разработки блоков не позволила применить идеальную систему отладки, когда тест для очередного блока автоматически готовится предыдущими блоками, уже отлаженными на данном тесте.

Комплексная отладка транслятора прошла сравнительно быстро (на тестах с мая по ноябрь 1963 г. за вычетом двухмесячного отпуска и на задачах в течение декабря 1963 – января 1964 гг.), особенно если учесть ненадежную работу машины в этот период. Система расширяющихся тестов была реализована не полностью, что прежде всего объясняется нетерпением авторов перейти к программированию реальных задач после получения первых свидетельств жизнеспособности транслятора.

Опытная эксплуатация АЛЬФА-транслятора началась в феврале 1964. г. и длилась до февраля 1965 г. Критерием окончания опытной эксплуатации явилось месячное интенсивное использование транслятора, не обнаружившее в нем новых ошибок. Поскольку опытная эксплуатация была последним рубежом в работе над транслятором, естественно, что во время ее пришлось ликвидировать все ранее сделанные упущения. Это привело к тому, что опытная эксплуатация затянулась более чем на год по сроку и на 500 задач вместо 100 по количеству программ. Работа носила в основном будничный характер по исправлению довольно многочисленных мелких ошибок, однако приходилось иногда принимать довольно кардинальные решения, связанные с усовершенствованием работы блоков (особенно блоков экономии памяти) и ликвидацией количественных ограничений. Зачастую в процессе опытной эксплуатации основная трудность состояла не в том, чтобы найти ошибку, а в том, чтобы правильно ее исправить. На первых стадиях опытной эксплуатации часто бывало так, что "исправление" мелкой ошибки на несколько дней выводило транслятор из строя. В дальнейшем была разработана целая система буферных лент с транслятором для того, чтобы не нарушать нормального прохождения транслируемых программ при внесении исправлений или изменений в транслятор.

5.3. Количественные показатели. В целом трудоемкость и ход разработки блоков АЛЬФА-транслятора характеризуются данными, представленными в приведенной выше таблице.

Некоторые суммарные данные, характеризующие АЛЬФА-транслятор в целом, приводятся ниже:

Общая длина АЛЬФА-транслятора	44680
В том числе констант и таблиц	5530
Трудоемкость, чел.-лет из расчета 300 чел.-дней на 1 чел.-год:	
начальный период разработки (1 – 4 этапы)	7,0
составление логических схем	5,8
программирование блоков	2,3
сепаратная отладка	3,4
комплексная отладка	6,2
опытная эксплуатация	7,3
.Суммарная трудоемкость	32,0
Количество ошибок, обнаруженных	
при сепаратной отладке	2000
при комплексной отладке	300
при опытной эксплуатации	220
Количество заходов на машину:	
при сепаратной отладке	1420
при комплексной отладке	105
при опытной эксплуатации	180
Расход машинного времени, ч:	
на сепаратную отладку	112
на комплексную отладку	130
на опытную эксплуатацию	280
Расход машинного времени (на одну ошибку, млн):	
при сепаратной отладке	3,4
при комплексной отладке	26,0
при опытной эксплуатации	76,4

Стоимость разработки АЛЬФА-транслятора (150 руб. за чел.-мес.

и 40 руб. за час машинного времени), руб. 78500

Приведенные цифры достаточно показательны и говорят сами за себя. Автору хотелось бы только обратить внимание на то, что в общей трудоемкости создания транслятора отладка превалирует над остальными этапами (16,9 чел.-лет против 15,1). Огорчает сравнительно высокий процент ошибок (см. таблицу). Поразительна разница в "цене" обнаружения ошибки на разных стадиях отладки – 3,4 мин на одну ошибку при сепаратной отладке против 76 мин – при опытной эксплуатации.

6. ОБЗОР ЛИТЕРАТУРЫ

В данной статье организация АЛЬФА-транслятора и его разработки описана post factum. Некоторые априорные рекомендации и суждения на этапе подготовки проекта транслятора содержатся в работе [3] и в отчетах [4, 5].

К моменту написания данной статьи известно несколько концепций внутреннего, или промежуточного, языка. Так, еще в одной из работ, посвященных ФОРТРАНу [6], было показано, что при программировании арифметических выражений их выгодно записывать сначала в некоторой промежуточной форме, напоминающей трехадресные команды, и лишь затем развертывать в машинный одноадресный код. Эта идея – отделить процесс синтаксического анализа и декомпозиции сложных конструкций входного языка от процесса построения машинных команд – стала, по-видимому, общепринятой. В частности, она была развита в трансляторе ТА-2 [7] в виде языка так называемых "основных понятий". Простота основных понятий позволяет организовать их перевод на язык машины "табличным методом", при котором правило перевода некоторого основного понятия дается постоянной табличкой, содержащей группу машинных команд с незаполненными внешними адресами, заменяющую при переводе данное понятие.

Существует и другой подход к выделению уровня промежуточного языка в виде машинно-ориентированного универсального языка типа УНКОЛ [8]. Этот подход, однако, в меньшей степени отражает внутренние потребности процесса трансляции, а определяется больше техническими или экономическими требованиями, например необходимостью совместной разработки трансляторов с одного или нескольких языков для целого семейства машин.

Подход к организации АЛЬФА-транслятора диктовался не только внутренней логикой проблемы, но до некоторой степени определялся и традициями советской школы автоматического программирования "доалголовского" периода [9–11], для которой были характерными забота о качестве программирования и необходимость рассчитывать на малый объем оперативной памяти машины.

Начатки смешанной стратегии программирования на основе анализа операторов входной программы можно найти в работе В. М. Курочкина [12], посвященной методике программирования циклов в программирующей программе для машины СТРЕЛА-3.

Из зарубежных работ наиболее близок к АЛЬФА-транслятору по общим подходам и даже по некоторым конкретным идеям, особенно в области анализа процедур и операторов циклов, проект транслятора для машины КДФ-9, описанный Хоукинсом и Хакстеблом [13].

Идея хранения и обработки информации на основе списочной структуры настолько общеизвестна в настоящее время, что просто невозможно указать конкретный источник, стимулировавший ее применение в АЛЬФА-трансляторе. Единственным, возможно, элементом новизны, во всяком случае для советских работ по программированию, является систематическое применение этой методики для системы программирования транслирующего типа.

Источниками для применения функции расстановки в АЛЬФА-трансляторе были работы [2, 14]. Однако нам недавно стали известны более ранние работы, посвященные этому методу, в частности работа Петерсона [15], в которой проведено исследование метода "открытой адресации", аналогичное исследованию, выполненному ранее [2]. В этой же работе указывается, что идея "открытой адресации", практически совпадающей с механизмом функции расстановки, восходит к Сэмюэлу, Эмделу и Боем (Boehme), которые применили этот механизм в 1954 г. для просмотра таблиц в составляющей программе для машины ИБМ-701

Номер блока	Наименование блока	Общая длина блока Машинных слов	Таблица***	Время Окончания Сепаратной отладки		Трудоемкость чел.	
						Логической схемы блока***	Программи-рования
1	Ввод и контроль	930	210	25/9	1962	25	6
2	Обработка описаний	1400	1600				
3	Замена идентификаторов	2560					
4	Выражения I	1730					
5	Выражения II	3400					
6	Анализ процедур (старый вариант)	800					
6	Анализ процедур (новый вариант)	2300					
7	Программирование процедур (старый вариант)	3130	120	28/4	1963	58	35
7	Программирование процедур (новый вариант)	2100	150	½	1965	50	10
8	Перевод в 15-разрядный код	2370	260	28/4	1963	30	35
9	Оюработка формирований и Компоновок	2120	150	30/11	1962	145	15
10	Многомерные массивы	5600	1600	15/5	1966	400	100
11	Анализ циклов	770	170	23/10	1962	13	3
12	Чистка циклов (начало)	2800	80	29/4	1963	50	20
13	Чистка циклов (заключение)	550	50	11/5	1963	1	2
14	Экономия выражений	2400	200	5/11	1962	80	100
15	Построение машинных команд	3000	800	9/11	1962	110	35
16	Программирование индекс-регистров	1140	60	9/11	1962	13	10
17	Циклы (начало)	1600	100	11/11	1962	30	39
18	Циклы (заключение)	600	100	11/11	1962	9	15
19	Экономия констант	620	70	22/11	1962	15	20
20	Операторная схема	2400	440	14/5	1963	70	40
21	Граф несовместимости	1130	40	17/6	1963	30	15
22	Распределение памяти	850	50	27/11	1962	50	17
23	Чистка программы	1010	100	26/10	1962	78	20
24	Компановка программы (новый вариант)	1300	120	10/12	1963	40	20
24	Компановка программы (старый вариант)	1000	100	25/12	1962	50	40

Примечание: * - округлено; ** - в том числе константы и таблицы; *** - считая с марта 1962 г.

Разработки дней	Количество ошибок	Количество	Расход машин	Количество ошибок	Номер

Сепаратной отладки	суммарная	Исправленных при прокрутке	Обнаруженных при сепаратной отладке			Обнаруженных при комплексной отладке	Обнаруженных в период опытной эксплуатации	Суммарное на 100 команд	
10	41	5	6	5	22	4	-	1,1	1
15	73	17	30						
150	270	8	135	116	425	22	11	6,6	3
20	140	-	75	41	254	12	6	5.,4	4
40	180	-	190	96	701	23	10	6,6	5
25	90	-	40	42	269	15	2	7,4	6
20	105	15	50	17	100	2	-	-	
40	133	-	150	106	534	21	7	5,6	7
25	85	10	40	20	200	15	-	-	7
15	80	63	25	17	194	12	8	1,9	8
25	195	5	100	64	234	4	-	-	9
100	600	15	50	60	360	50	10	4,5	10
4	20	4	23	12	30	12	6	5,3	11
100	170	30	188	263	1353	32	22	8,6	12
9	12	-	33	31	182	17	2	9,6	13
18	108	28	44	55	219	10	11	3,3	14
40	185	35	150	88	385	23	22	6,7	15
6	29	-	35	15	72	9	10	4,7	16
31	100	31	70	56	226	9	16	5,9	17
25	49	-	27	22	80	9	5	7,0	18
20	55	4	40	26	108	6	7	8,6	19
100	210	63	175	58	182	17	25	9,0	20
95	140	14	85	54	180	8	15	9,5	21
24	91	23	47	52	167	5	6	6,8	22
20	118	17	52	38	126	6	14	7,1	23
25	85	-	35	30	150	1	11	6,5	24
35	125	30	116	36	171	1	11	6,5	24

7. ЗАКЛЮЧЕНИЕ

7.1. Развитие техники программирования с языка АЛГОЛ 60 идет до последнего времени в основном под флагом создания универсальных, быстродействующих и элегантных алгоритмов трансляции, идущих, прежде всего, от природы языка или способа его описания. Идейной закваской для этих алгоритмов явились работы Замельзона и Бауэра [16] о последовательном программировании, Дейкстры [17] о рекурсивном программировании, Ингермана [18], Пауля [19] и Айронса [20] о синтаксически управляемых трансляторах..

На этом фоне подход к разработке АЛЬФА-транслятора выглядит несколько старомодным и громоздким. Однако можно изложить некоторую аргументацию в пользу или в необходимость этого подхода.

1°. При разработке АЛЬФА-транслятора был подчеркнуто применен метод инженерного конструирования системы, направленный, прежде всего, на удовлетворение заранее заданным эксплуатационным характеристикам. Эти рамки не позволили подчинить организацию транслятора какой-либо центральной идее, придавшей бы транслятору большее внутреннее единство и логическую стройность, а, наоборот, требовали внесения в транслятор самых разнообразных качеств, не связанных друг с другом логически, но служащих одной и той же цели – повышению качества программирования. В этом смысле

сложность АЛЬФА-транслятора и разнообразие его алгоритмов отражают сложность реального процесса программирования, выполняемого человеком.

2° Дополнительной причиной увеличения объема транслятора и усложнения его строения является недостаточная емкость оперативной памяти машины, для которой был создан транслятор. Хотя это звучит парадоксально, но транслятор получился слишком большим, потому что машина слишком мала! По-видимому, характеристики машины предельны для трансляторов подобного типа. Удлинение транслятора связано с искусственным увеличением числа блоков и наличием большого числа дублирующих действий и подпрограмм в отдельных блоках.

3°. Реализация АЛЬФА-транслятора на более современной машине среднего класса позволила бы значительно разгрузить транслятор и повысить его быстродействие. Можно сделать предположение, что транслятор с Выходного языка для машины типа ИБМ-7090 с тем же качеством программирования, что и АЛЬФА-транслятор, будет составлять программы не более чем за десять просмотров со скоростью не менее 1500 одноадресных команд готовой программы на минуту работы транслятора.

7.2. Разработка системы АЛЬФА> показывает, что задача создания сложного комплекса программ типа АЛЬФА-транслятора не предъявляет чрезмерных требований к профессиональному и научному уровню разработчиков. Любой математик или инженер средних способностей, но с выраженным "комбинаторным" складом мышления, имеющий хотя бы двухлетний стаж работы в качестве профессионального программиста в хорошо организованном вычислительном центре, может успешно работать в составе группы разработчиков. Однако в коллективе совершенно необходимы два-три программиста экстра-класса, играющих прежде всего роль генераторов конструкторских идей, координирующих отдельные участки разработки и умеющих быстро принимать правильные решения в ответственный период комплексной наладки системы.

Что касается требований, предъявляемых к руководителю разработки, то автору трудно обсуждать всесторонне этот вопрос, поскольку ему пришлось выступить именно в этой роли при создании системы АЛЬФА. Можно сделать, однако, три вывода, полученных на основе анализа ошибок, допущенных при организации работы над транслятором.

1°. Руководитель должен умело сочетать индивидуальный и коллективный характер работы сотрудников. Налаживание духа сотрудничества и взаимопонимания, чувства ответственности за общее дело и организация фактического взаимодействия – все это очевидные задачи, стоящие перед руководителем. Однако чрезмерное увлечение дискуссиями на семинарах группы, длительные консультации с коллегами иногда приводят к поверхностным решениям и к утрате вкуса к углубленному и индивидуальному исследованию.

2°. Задание оптимального темпа и ритма работы, по-видимому, –. одна из самых главных задач, стоящих перед руководителем. Он должен ощущать ту внутреннюю скорость, с которой движется дело и, поддерживая совершенно необходимую напряженность темпа работы, не давать этой напряженности перерастать в штурмовщину. Выбор правильного темпа работы особенно важен в период проектирования и программирования блоков системы, когда последствия той или иной ошибки еще не проявляют себя немедленно. В этом аспекте руководитель должен уметь преодолевать как собственное честолюбие, питающее его чрезмерный оптимизм, так и зачастую давление администрации, настаивающей на обязательном соблюдении срока поставки и других обязательств.

3°. Наконец, как первое, так и второе требования будут соблюдены лишь тогда, когда руководитель сможет полностью разгрузить себя от работы, связанной с деталями реализации проекта. Обладая запасом опыта и знаний, он должен бороться с желанием исправлять ошибки его менее опытных подчиненных своими собственными руками.

Автор выражает глубокую благодарность Г. И. Кожухину и И. В. Поттосину, которые, блестяще выполнив свои функции координаторов по фазам перевода и программирования соответственно, внесли большой вклад в принятие решений по организации АЛЬФА-транслятора..

Автор с признательностью отмечает деятельность Информационного центра по автоматическому программированию в Брайтоне (A. P. I. C.), чьи публикации по автоматизации программирования постоянно помогали нам быть в курсе последних достижений этой области.

Автор благодарит С. С. Лаврова, прочитавшего рукопись статьи и сделавшего полезные замечания.

СПИСОК ЛИТЕРАТУРЫ

1. Адельсон-Вельский Г. М., Ландис Е. М. Один алгоритм организации информации. – Докл. АН СССР, 1962, 146, № 2.

2. Ершов А. П.О программировании арифметических операторов. – Докл. АН СССР, 1961, 118, № 3.

3. Ершов А. П. Основные принципы построения программирующей программы Института математики СО АН СССР. – Сиб. матем. ж., 1961, 2, № 6.

4. Ершов А. П., Кожухин Г. И. Проект программирующей программы Института математики СО АН СССР. – Отчет Вычисл. центра СО АН СССР. Новосибирск, 1961 (приложение 1 в настоящем сборнике).

5. Ершов А. П. Основные проблемы построения программирующей программы Института математики СО АН СССР. – Отчет Вычисл. центра СО АН СССР. Новосибирск, 1961 (приложение 2 в настоящем сборнике).

6. Sheridan P. B.The arithmetic translator-compiler of the IBM Fortran automatic coding system. – C.A.C.M., 1959, 2, № 2.

7. Шура-Бура М.Р., Любимский Э.З. Транслятор АЛГОЛ 60. – Ж. вычисл. матем. и матем. физ., 1964, 4, № 1.

8. Steel T.B., Jr.UNCOL: the myth and the fact. – Ann. Review in autom. programming, v.2. Oxford, Pergamon Press, 1961.

9. Камынин С.С., Любимский Э.З. Автоматизация программирования. Конференция "Пути развития советского математического машиностроения и приборостроения", ч.III. М.,1956.

10. Ершов А. П. Программирующая программа для быстродействующей электронной счетной машины. М., Изд-во АН СССР, 1958.

11. Система автоматизации программирования. Под ред. Н.П. Трифонова и М.Р. Шура-Буры. М., Физматгиз, 1961.

12. Великанова Т. М., Ершов А. П., Ким К. В., Курочкин В. М., Олейник-Овод Ю. А., Поддерюгин В. Д. Программирующая программа для машины (ППС). – Тр. Всес. совещ. по вычисл. матем. и применению средств вычисл. техн. 3–8 февр. 1958. Баку, Изд-во АзербССР, Б.г.

13. Hawkins E. N., Huxtable D. H. R.A multi-pass translation scheme for ALGOL 60. – Ann. Review in autom. Programm, v.3. Oxford, Pergamon Press, 1963.

14. Иванова Э.К. О выборе функции расстановки для организации табличных просмотров. – Отчет Вычисл. Центра СО АН СССР, Новосибирск, 1961.