Глава 14

Программная реализация управляющих автоматов в базисе языков инструкций

Рассмотрим вопрос о программной реализации управляющих автоматов в базисе языков инструкций («Instruction List») на примере использования языка ALPro, применяемого в программируемых логических контроллерах «Autolog» [230] (разд. 14.1—14.6).

14.1. Реализация автоматов без памяти

Предположим, что требуется программно реализовать следующий алгоритм сигнализации: при отсутствии признака мигания ($p = 0$) выход z повторяет значения входного сигнала x\ при наличии признака мигания ($p=1$) выход z повторяет значения переменной g, вырабатываемой генератором мигания. Этот алгоритм может быть формализован с помощью таблицы истинности (табл. 14.1).

Таблица 14.1

p	x	g	z	p	x	g	z
0	0	0	0	1	0	0	0
0	0	1	0	1	0	1	1
0	1	0	1	1	1	0	0
0	1	1	1	1	1	1	1

14.1.1. Программирование граф-схем алгоритмов

Применяя канонический метод Блоха [18], построим ГСА (рис. 14.1), реализующую табл. 14.1. Оптимизируем эту граф-схему за счет выноса операторной вершины г = 0 вверх (рис. 14.2).

Особенность используемого языка состоит в том, что без применения шагового регистра он обеспечивает реализацию только двух управляющих

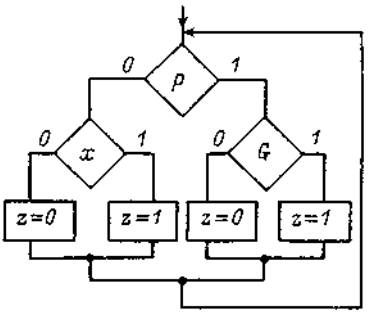

Рис. 14.1

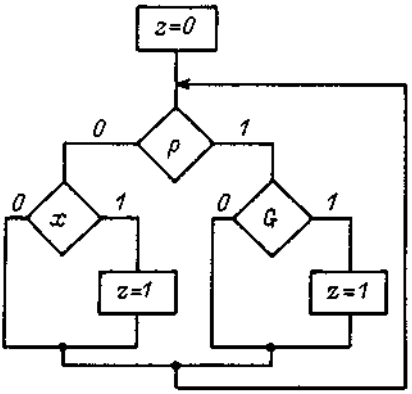

Рис. 14.2

конструкций, а именно: «последовательное соединение блоков» и «неполный выбор». Это приводит к необходимости дальнейшего преобразования граф-схемы — к построению линеаризованной ГСА (рис. 14.3), число структурированных блоков в которой, не включая блок начальной установки, равно числу путей в граф-схеме, завершающихся операторными вершинами. В силу того что число таких путей в граф-схеме достаточно велико, то и число команд в программе, созданной таким образом, также будет достаточно велико.

Для реализации ГСА (рис. 14.3) используем следующие команды:

`STR C K` — запись битовой константы С со значением К в битовый сумматор (БС) — БС: = К;

`EQ RO i` — сброс (R) единичного значения выходной (О) переменной i, если в БС единица (БС: = 1), и сохранение ее значения — в противном случае;

`IF NI j` — переход к следующей команде, если значение инверсии (N) входной (I) переменной j равно единице, и к метке CONT — в противном случае;

`IF I j` — переход к следующей команде, если значение входной переменной j равно единице, и к метке CONT — в противном случае;

`EQ SO i` — установка (S) единичного значения выходной переменной i, если БС: = 1, и сохранение ее значения — в противном случае;

CONT — метка, после которой осуществляется переход к следующей команде;

STOP — переход к первой команде программы.

При этом программа, записанная для экономии места в два столбца, имеет следующий вид:

```
STR  C  I    IF   I  p
EQ  RO  z    IF   I  g
IF  NI  p    EQ  SO  z
IF   I  x    CONT
EQ  SO  z    STOP
CONT
```

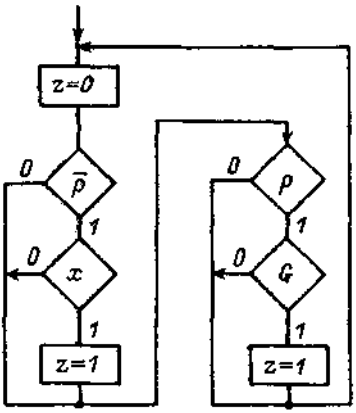

Рис. 14.3

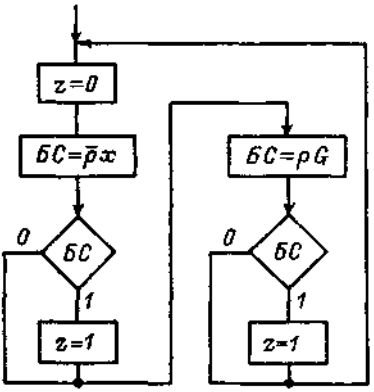

Рис. 14.4

Таким образом, непосредственная реализация последней граф-схемы требует 11 команд и необходимость использования битовых ячеек промежуточной памяти (ЯПП) отсутствует. Следовательно, сложность программы составляет 11; 0. При этом отметим, что и в общем случае бупеву формулу можно реализовать по бинарной ГСА без применения ЯПП [271].

Число команд в программе может быть сокращено, если линеаризованную ГСА (рис. 14.3) реализовать не непосредственно, как это было выполнено выше, а с помощью операторно-бинарной граф-схемы программы (рис. 14.4), отражающей специфику выполнения команд ПЛК. При этом применяются следующие новые команды:

`STR NI j` — ввод значения инверсии входной переменной j в БС **(БС:=1);**

`STR I j` — ввод значения входной переменной j в БС (БС: = j);

`AND I j` — конъюнкция содержимого БС и входной переменной j с записью результата в битовый сумматор (БС: = БС & j).

При этом программа, записанная в два столбца, приобретает следующий вид:

```
STR  C  I      STR  I  p
EQ  RO  z      AND  I  g
STR NI  p      EQ  SO  z
AND  I  x      STOP
EQ  SO  z
```

Сложность этой программы 9; 0.

По исходной граф-схеме (рис. 14.1) может быть построена и другая ГСА (рис. 14.5). Линеаризуем эту граф-схему (рис. 14.6). При написании программы по ней применяется команда:

`EQ O j` — запоминание содержимого битового сумматора в битовой ячейке памяти, соответствующей выходной переменной j (j: = БС).

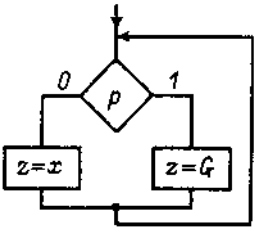

Рис. 14.5

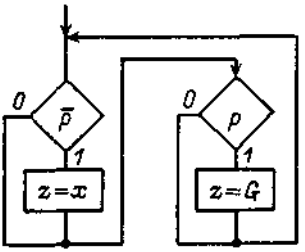

Рис. 14.6

Программа, записанная в два столбца, в этом случае имеет следующий вид:

```
IF  NI  p      IF  I  p
STR  I  x      STR I  g
EQ   O  z      EQ  O  z
CONT           CONT
               STOP
```

Таким образом, несмотря на то что последняя граф-схема проще предыдущих, сложность программы не изменилась и составляет 9; 0.

14.1.2. Реализация булевых формул операторными программами

Табл. 14.1 описывается булевой формулой $z = \bar{p} \,\&\, x \vee p \,\&\, g$, которая реализуется следующей программой, также записанной в два столбца:

```
STR NI  p      STR I  p
AND  I  x      AND I  g
EQ   M  y      OR  M  y
               EQ  O  z
               STOP
```

Сложность программы 8; 1. В этой программе используются следующие новые команды:

`EQ M j` — запоминание содержимого БС в битовой ячейке промежуточной памяти j (j: = БС);

`OR M j` — дизъюнкция содержимого БС и содержимого битовой ячейки промежуточной памяти j с записью результата в битовый сумматор (БС: = ВС V j).

При реализации булевой формулы операторными программами в общем случае требуется применять ячейки промежуточной памяти. Так как в программируемых логических контроллерах число битовых ЯПП весьма ограничено, то рассмотрим метод минимизации числа таких ячеек.

Метод базируется на использовании того факта, что бесповторные пороговые формулы (БПФ) при их записи в порядке увеличения порогов

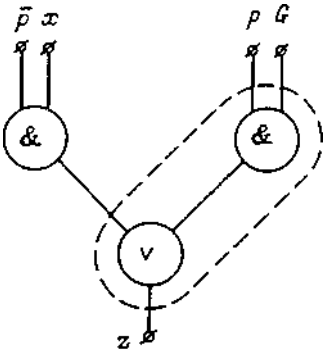

Рис. 14.7

могут быть реализованы операторными программами без применения ЯПП. Также могут быть реализованы формулы той же структуры, что и БПФ, в которых используется операция «неравнозначность».

Положительно монотонные формулы этих классов из h букв могут быть реализованы линейным (не древовидным) каскадом из h - 1 двухвходовых элементов: И, ИЛИ, НЕРАВНОЗНАЧНОСТЬ.

Покажем, что специфика работы ПЛК позволяет реализовать формулы и других классов без применения ячеек промежуточной памяти. Возможность такой реализации базируется на том, что в рассматриваемом контроллере в память выходного модуля считывается из модуля оперативной памяти лишь последнее значение каждой выходной переменной из множества ее значений, формируемых в течение программного цикла. Поэтому ячейки модуля памяти, предназначенные для фиксации значений выходных переменных, можно использовать также и для фиксации промежуточных результатов вычислений.

Метод состоит в построении по заданной булевой формуле схемы максимальной глубины из двухвходовых элементов указанных типов и выделении в ней минимально связанных между собой линейных каскадов. Если при этом формулу удается реализовать таким образом, что каждый каскад будет соединен не более чем с одним другим каскадом схемы, то такая формула может быть реализована без ячеек промежуточной памяти. Такой схеме из линейных каскадов соответствует СБФ, являющаяся суперпозицией формул, каждая из которых реализуется одним каскадом и содержит не более одной переменной z.

Для рассматриваемого примера схема с выделенными каскадами приведена на рис. 14.7. Этой схеме соответствует система булевых формул

$$z = \bar{p} \,\&\, x; \quad z = p \,\&\, g \vee z,$$

которая реализуется следующей программой, состоящей из двух столбцов:

```
STR NI  p      STR I  p
AND  I  x      AND I  g
EQ   O  z      OR  O  z
               EQ  O  z
               STOP
```

Сложность программы 8; 0. В этой программе используется новая команда:

`OR O i` — дизъюнкция содержимого БС и битовой ячейки памяти, соответствующей выходной переменной i, с записью результата в битовый сумматор (БС: = БС v i).

На рис. 14.8 в качестве более сложного примера приведена схема, реализующая формулу

$$z = (x_1 \,\&\, x_2 \vee x_3 \,\&\, x_4) \,\&\, x_5 \vee x_6 \,\&\, x_7 \,\&\, x_8.$$

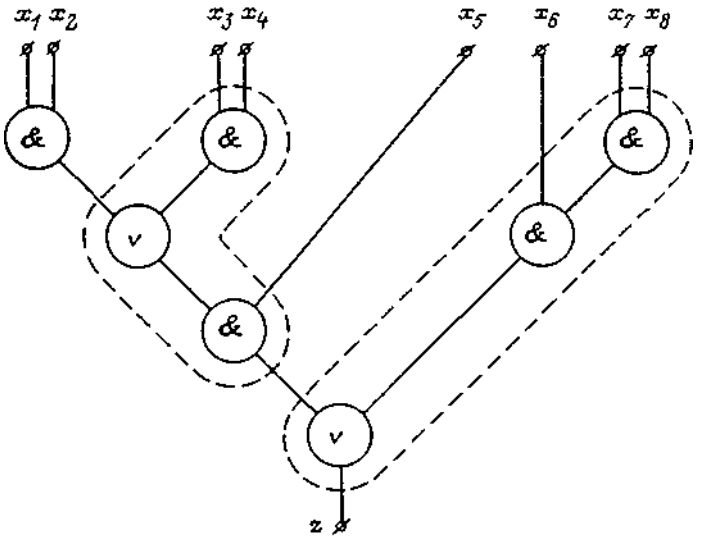

Рис. 14.8

В этой схеме выделены линейные каскады, каждый из которых связан не более чем с одним другим каскадом. Такому выделению каскадов соответствует СБФ:

$$z = x_1 \,\&\, x_2; \quad z = (x_3 \,\&\, x_4 \vee z) \,\&\, x_5; \quad z = x_6 \,\&\, x_7 \,\&\, x_8 \vee z,$$

которая реализуется программой, сложность которой 14; 0.

Естественно, что не любая булева формула может быть реализована операторной программой без ЯПП. Так, например, формула

$$z = (x_1 \,\&\, x_2 \vee x_3 \,\&\, x_4) \,\&\, (x_5 \,\&\, x_6 \vee x_7 \,\&\, x_8)$$

не может быть реализована по крайней мере без одной ячейки промежуточной памяти.

Однако исследование структур схем (с линейными минимально связанными каскадами), реализующих произвольные формулы в базисе И, ИЛИ, НЕ, НЕРАВНОЗНАЧНОСТЬ из семи и менее букв, показывает, что любая из таких формул может быть реализована операторной программой без ячеек указанного типа. Так, например, формула

$$z = (x_1 \,\&\, x_2 \vee x_3 \,\&\, x_4) \,\&\, (x_5 \oplus x_6 \,\&\, x_7)$$

реализуется без таких ячеек, так как она может быть представлена суперпозицией формул вида

$$z = x_1 \,\&\, x_2; \quad z = x_3 \,\&\, x_4 \vee z; \quad z = (x_6 \,\&\, x_7 \oplus x_5) \,\&\, z.$$

14.1.3. Реализация булевых формул операторно-бинарными программами

Выделим в заданной булевой формуле фрагменты, реализуемые линейными каскадами, и заменим их новыми буквами. Если в результате получается бесповторная пороговая формула, то, применяя метод, изложенный в разд. 4.3.1, построим структурированную ГСА. Реализуя эту граф-схему и выделенные формулы, помечающие ее условные вершины, получим операторно-бинарную программу, в которой не используются ячейки промежуточной памяти.

Так как в данном случае в отличие от предыдущего не применяются команды запоминания промежуточных результатов, то для рассматриваемого класса формул удается получать программы с числом команд, не большим, чем их количество в программах, построенных с помощью предыдущего метода.

Так, формула

$$z = x_1 \,\&\, x_2 \vee x_3 \,\&\, x_4 \vee x_5 \,\&\, x_6 \vee x_7 \,\&\, x_8$$

с помощью предыдущего метода реализуется программой, сложность которой 16; 0, в то время как, используя метод, излагаемый в настоящем разделе, получим программу, сложность которой 15; 0. Эта программа строится по операторно-бинарной граф-схеме (рис. 14.9).

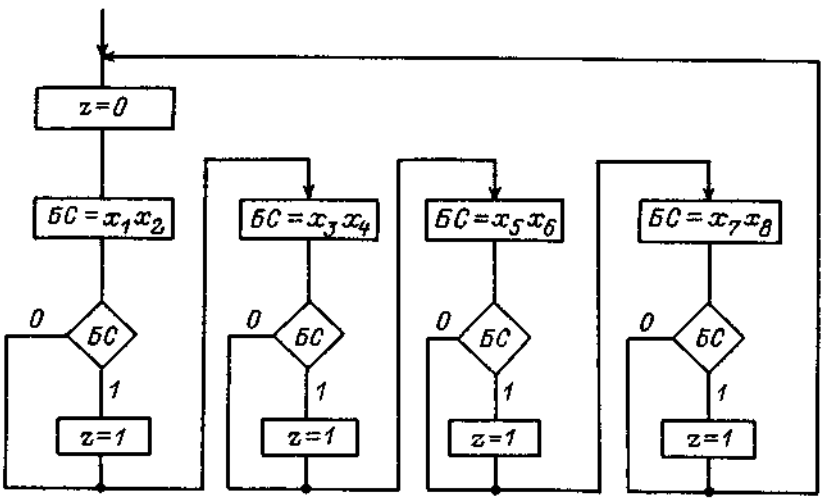

Рис. 14.9

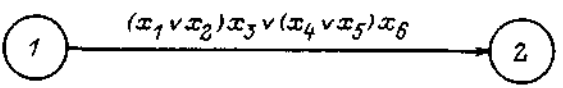

Рис. 14.10

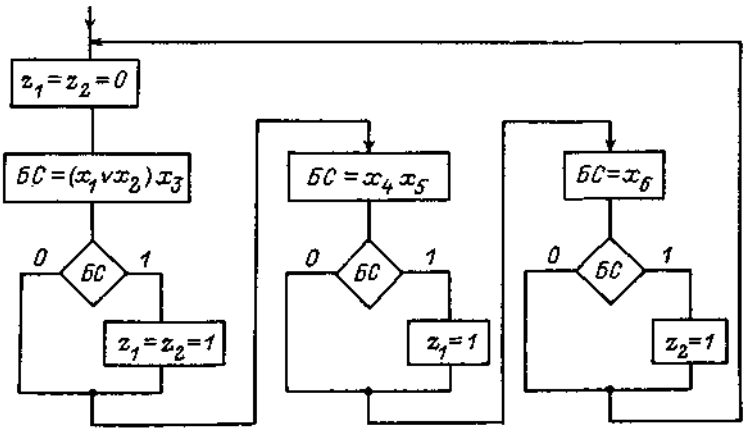

Рис. 14.11

Необходимо отметить, что за счет хотя бы частичного раскрытия скобок предлагаемый метод всегда может быть применен, обеспечивая построение программы с не минимальным числом команд, но без использования указанных ячеек. Например, булева формула

$$z = (x_1 \,\&\, x_2 \vee x_3 \,\&\, x_4) \,\&\, (x_5 \,\&\, x_6 \vee x_7 \,\&\, x_8)$$

может быть заменена формулой

$$z = x_1 \,\&\, x_2 \,\&\, x_5 \,\&\, x_6 \vee x_1 \,\&\, x_2 \,\&\, x_7 \,\&\, x_8 \vee x_3 \,\&\, x_4 \,\&\, x_5 \,\&\, x_6 \vee$$
$$\vee\, x_3 \,\&\, x_4 \,\&\, x_7 \,\&\, x_8,$$

которая реализуется без применения ячеек промежуточной памяти.

Рассмотренный метод может весьма эффективно применяться при реализации пометок дуг графов переходов. На рис. 14.10 приведен фрагмент графа переходов, реализованный ниже двумя программами, применяющими шаговые регистры (разд. 14.2.3), в первой из которых используется одна ячейка промежуточной памяти, а вторая построена с помощью модификации предлагаемого метода:

```
STR  I x1      STR  I x1
OR   I x2      OR   I x2
AND  I x3      AND  I x3
EQ   M y       AND  S  1
STR  I x4      STEP S  2
OR   I x5      STR  I x4
AND  I x6      OR   I x5
OR   M y       AND  I x6
AND  S  1      AND  S  1
STEP S  2      STEP S  2
```

Сложность первой программы 10; 1, а второй — 10; 0.

В этих программах применяется новая команда:

`OR I j` — дизъюнкция содержимого БС и входной переменной j с записью результата в битовый сумматор (БС: = БС v j).

Семантика команд `AND S G` и `STEP S G` описана в разд. 14.2.3.

При этом отметим, что в каждой из этих программ «стробирование» формулы перехода или ее фрагментов номером вершины, из которой осуществляется переход (применение команд AND S 1), выполняется после вычисления формулы или фрагмента, в то время как для конъюнкций «стробирование» может выполняться и до вычисления формулы, что ближе к структуре графа переходов.

Изложенный метод позволяет реализовывать не только описанный класс булевых формул, но и системы из них без применения ячеек промежуточной памяти. Так, например, СБФ:

$$z_1 = (x_1 \vee x_2)\ \&\ x_3 \vee x_4\ \&\ x_5;\quad z_2 = (x_1 \vee x_2)\ \&\ x_3 \vee x_6,$$

может быть представлена операторно-бинарной граф-схемой (рис. 14.11), сложность программной реализации которой 14; 0. При использовании факторизации эта система преобразуется:

$$y = (x_1 \vee x_2)\ \&\ x_3;\quad z_1 = x_4\ \&\ x_5 \vee y;\quad z_2 = y \vee x_6,$$

что позволяет построить программу, сложность которой 12; **1.**

14.2. Реализация автоматов с памятью

14.2.1. Использование для кодирования состояний двоичных переменных

Пусть требуется программно реализовать *R*-триггер. При этом отметим, что программы, приведенные ниже, в пунктах *1*—7, записаны в два столбца.

1. Выполним программирование по ГСА (рис. 13.38):

```
IF  NO z      IF   M z
IF  NI R      IF   I R
IF   I S      STR  C 1
STR  C 1      EQ  RO z
EQ  SO z      CONT
CONT          STOP
```

Сложность программы 12; 0. Фрагменты этой программы, завершающиеся меткой CONT, ввиду их ортогональности могут быть записаны в произвольном порядке.

В этой программе применяются новые команды:

`IF NO j` — переход к следующей команде, если значение инверсии выходной переменной j равно единице, и к метке CONT — в противном случае;

`IF M j` — переход к следующей команде, если значение переменной j в битовой ячейке промежуточной памяти равно единице, и к метке CONT — в противном случае.

2. Выполним программирование по упрощенной ГСА (рис. 13.40):

```
IF   NO  z     IF    M  z
IF    I  S     IF    I  R
STR   C  1     STR   C  1
EQ   SO  z     EQ   RO  z
CONT           CONT
               STOP
```

Сложность программы 11; 0. Порядок расположения фрагментов в этой программе не может быть изменен. При $R = S = 1$ за один программный цикл реализуются два фрагмента программы, однако выбранный порядок расположения фрагментов обеспечивает правильную реализацию заданного алгоритма.

3. Выполним программирование по еще более упрощенной ГСА (рис. 13.41):

```
IF   NI  R     IF    I  R
IF    I  S     STR   C  1
STR   C  1     EQ   RO  z
EQ   SO  z     CONT
CONT           STOP
```

```
IF    I  S     IF    I  R
STR   C  1     STR   C  1
EQ   SO  z     EQ   RO  z
CONT           CONT
               STOP
```

Сложность программы 9; 0.

5. Выполним программирование непосредственно по графу переходов (рис. 4.15):

```
STR NO z     STR  O z
AND NI R     AND  I R
AND  I S     EQ  RM z1
EQ  SM z1
             STR  M z1
             EQ   O z
             STOP
```

Сложность программы 10; 0. В данной программе используется новая команда:

`AND NI j` — конъюнкция содержимого БС и инверсии переменной j с записью результата в битовый сумматор (БС: = БС &j).

Команды `EQ SM i; EQ RM i; STR M i` отличаются от аналогичных команд, рассмотренных выше, типом операнда — М вместо О. В программе применяется переобозначение переменной z для реализации в течение одного программного цикла не более одного перехода в ГП.

6. Выполним программирование непосредственно по графу переходов с использованием других типов команд (рис. 4.15):

```
STR NO z     STR  O z
AND NI R     AND  I R
AND  I S     EQ  RO z
EQ  SO z     STOP
```

Сложность программы 8; 0.

Так как в рассматриваемом контроллере в течение одного программного цикла значения входных переменных измениться не могут, а фрагменты программы, соответствующие переходам, ортогональны по переменной *R,* то в каждом таком цикле реализуется только один фрагмент программы (один переход в графе переходов) и значения переменной *z* не могут быть отфильтрованы. В этой программе внутри каждого фрагмента содержится вся информация о предыдущем значении *z* (предыстория), и поэтому для определения этого значения нет необходимости его помнить внемодельными средствами (в голове), а достаточно подняться по программе вверх. Ввиду того что каждый фрагмент программы эквивалентен одному переходу (дуге с начальной и конечной вершинами) в графе переходов, то такая программа является наиболее читаемой: по ее тексту граф переходов восстанавливается «один в один», используя только информацию, имеющуюся в тексте программы.

В данном случае программа построена по графу переходов формально и изоморфно и носит декларативный характер, так как размещение фрагментов в ней может быть произвольным, что в некотором смысле является наилучшим способом ее организации. Естественно, что это свойство программы обеспечивается за счет избыточности, которая, как будет показано ниже, может быть уменьшена при увеличении степени процедурности — зависимости функционирования программы от порядка расположения ее фрагментов.

В этой программе применяются следующие новые команды:

`STR NO i` — ввод в БС значения инверсии выходной переменной i (БС: = !i);

`STR O i` — ввод в БС значения выходной переменной i (БС: = i).

При непосредственном программировании по графу переходов можно считать также, что программирование выполняется по структурированной и линеаризованной ГСА (рис. 13.38).

7. Выполним программирование по ГСА (рис. 13.40). При этом программа приобретает вид:

```
STR NO z    STR   O z
AND  I S    AND   I R
EQ  SO z    EQ   RO z
            STOP
```

Сложность программы 7; 0.

В этой программе фрагмент, соответствующий дуге графа переходов с большим приоритетом (рис. 13.39), расположен ниже фрагмента, соответствующего дуге с меньшим приоритетом, а порядок расположения фрагментов в программе не может быть изменен.

8. Выполним программирование по простейшей ГСА (рис. 4.79). При этом программа приобретает следующий вид:

```
STR   I  S
EQ   SO  z
STR   I  R
EQ   RO  z
STOP
```

Сложность программы 5; 0.

Несмотря на идеальную простоту этой программы, определяемую, в частности, отсутствием в ней проверок значений выходной и дополнительно вводимых переменных, ее сложно читать, так как по тексту программы в любой момент времени не определить значение z при $R = S = 0$, а для определения этого значения необходимо помнить предысторию внемодельными средствами.

Необходимо отметить, что в этом случае отсутствует изоморфизм между ГСА, по которой строится программа, и функционально эквивалентным ей графом переходов.

В рассмотренных программах, реализующих автомат с памятью, предполагалось, что начальная установка значения выходной переменной z выполняется по умолчанию: считается, что при включении питания $z = 0$. Если начальная установка должна выполняться явно, то в ПЛК должна быть выбрана такая программная компонента, которая точно сбрасывается при отключении питания и может служить по этой причине в качестве «инициатора» начальной установки других компонент. Используем для указанной цели тот шаговый регистр, например с номером «0», который при отключенном питании точно находится в нулевом состоянии (разд. 14.2.3). При этом блок начальной установки для рассмотренных в настоящем разделе автоматов реализуется следующим образом:

```
READ S  0
STR  S  0
EQ  RO  z
STEP S  1
```

Этот фрагмент программы обеспечивает однократное выполнение команды STR RO z, так как уже при втором проходе программы, ввиду того что значение 5 к этому моменту стало равным единице, вторая команда фрагмента не изменит значения переменной z.

9. Если программирование выполнить по формуле $z = (S \vee z) \& \overline{R}$, построенной по графу переходов (рис. 4.73), то получим следующую программу:

```
STR  I  S
OR   O  z
AND NI  R
EQ   O  z
STOP
```

Сложность программы 5; 0. При этом отметим, что переменные S, применяемые в приведенной программе и в предыдущем пункте, семан-

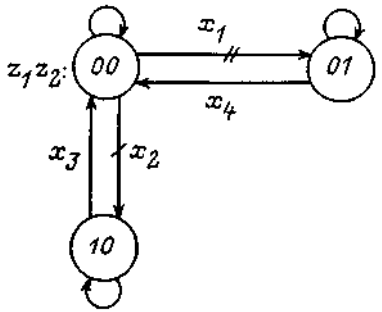

Рис. 14.12

тически различны. Несмотря на минимальную сложность, эта программа обладает тем недостатком, что ее весьма сложно читать, если в понятие «чтение» включать и процесс понимания текста. При этом термин «понимание» трактуется автором в том смысле, что непосредственно по тексту программы можно определить поведение того автомата, который она реализует. Если текст программы не изоморфен с исходным графом переходов, то восстановить последний (определить поведение автомата) без дополнительных математических преобразований не представляется возможным. В случае, если «чтение программы» трактовать более традиционно, как процесс выполнения команд в ней, то про рассмотренную программу можно сказать, что ее просто читать, но весьма сложно понять. Аналогичная ситуация возникает и при чтении текста на каком-либо естественном языке человеком, плохо знающим этот язык, когда он может правильно прочесть текст, но не понять его содержание.

Перейдем к вопросу о реализации более сложных графов переходов. Если в нем число дуг и число вершин не совпадают,то при программировании непосредственно по графу переходов возможны два подхода к его реализации:

— построение программы по дугам (переходам) графа;
— построение программы по вершинам графа.

Построим по графу переходов программу таким образом, что каждому переходу (кроме перехода по петле) в нем будет соответствовать фрагмент программы.

При этом если в графе противоречивость переходов устранена ортогонализацией, то фрагменты программы, каждый из которых соответствует одному переходу в графе, могут располагаться в ней произвольно. Таким образом, программа в этом случае носит декларативный характер. В этом случае фрагменты наиболее целесообразно размещать в порядке возрастания кодов состояний.

Если в графе переходов противоречивость переходов устранена расстановкой приоритетов, то фрагмент программы, соответствующий дуге с более высоким приоритетом, должен располагаться в ней выше фрагмента, соответствующего дуге с меньшим приоритетом.

Рассмотрим в качестве примера граф переходов (рис. 14.12), в котором использованы приоритеты, и построим для него три программы, в первой из которых применяются полные коды состояний, во второй — полные коды состояний и умолчания не изменяющихся при переходе значений выходных переменных, а в третьей — неполные коды и умолчания не изменяющихся при переходе значений выходных переменных:

```
STR NO z1    STR NO z1    STR NO z1
AND NO z2    AND NO z2    STR NO z2
```

```
AND  I x2
EQ  SO z1
EQ  RO z2

STR NO z1
AND NO z2
AND  I x1
EQ  RO z1
EQ  SO z2

STR NO z1
AND  O z2
AND  I x4
EQ  RO z1
EQ  RO z2

STR  O z1
AND NO z2
AND  I x3
EQ  RO z1
EQ  RO z2
STOP
```

```
AND  I x2
EQ  SO z1

STR NO z1
AND NO z2
AND  I x1
EQ  SO z2

STR NO z1
AND  O z2
AND  I x4
EQ  RO z2

STR  O z1
AND NO z2
AND  I x3
EQ  RO z1
STOP
```

```
AND  I x2
EQ  SO z1

STR NO z1
AND NO z2
AND  I x1
EQ  SO z2

STR  O z2
AND  I x4
EQ  RO z2

STR  O z1
AND  I x3
EQ  RO z1
STOP
```

В этих программах используются новые команды:

AND NO i — конъюнкция содержимого БС и инверсии выходной переменной i с записью результата в битовый сумматор (БС: = БС & !i).

AND O i — конъюнкция содержимого БС и выходной переменной i с записью результата в битовый сумматор (БС: = БС & i).

Сложность первой программы 21; 0, второй — 17; 0, а третьей — 15; 0. При этом необходимо отметить, что «понимаемость» программы уменьшается с сокращением числа команд.

Приведем также фрагменты программ для случая программирования «по вершинам», которые изменяются по сравнению с фрагментами программ, построенных «по переходам». При этом если умолчания не применяются, то программирование может выполняться как по ортогонализованным графам переходов, так и по графам переходов с приоритетами (первый фрагмент), а при использовании умолчаний — только по ортогонализованным графам (второй и третий фрагменты):

```
STR NO z1
AND NO z2
IF   T
STR  I x1
EQ  RO z1
EQ  SO z2
STR  I x2
EQ  SO z1
EQ  RO z2
CONT
```

```
STR NO z1
AND NO z2
IF   T
STR  I x1
AND NI x2
EQ  SO z2
STR  I x2
EQ  SO z1
CONT
```

```
STR NO z1
AND NO z2
IF   T
STR  I x1
AND NI x2
EQ  SO z2
STR  I x2
EQ  SO z1
CONT
```

В этих фрагментах используется новая команда IF T: если содержимое битового сумматора равно единице, то осуществляется переход к следующей команде, в противном случае — на метку CONT.

Особенность первого фрагмента программы состоит в том, что в нем в отличие от программ, построенных «по переходам», подфрагмент, соответствующий дуге с более высоким приоритетом, располагается в программе ниже фрагмента, соответствующего дуге с меньшим приоритетом. Для второго и третьего фрагментов порядок расположения подфрагментов произволен. Сами фрагменты во всех типах программ могут располагаться в любом порядке. Однако наиболее целесообразно располагать их в порядке возрастания кодов состояний.

Сложность программы в целом в первом случае составляет 21; 0, во втором — 18; 0, а в третьем — 16; 0. Несмотря на то что сложность реализации «по переходам» в рассмотренном примере ниже сложности реализации «по вершинам», в общем случае априори не известно, какой из подходов обеспечит более простую реализацию. При построении графа переходов (рис. 14.12) предполагалось, что переменные x_1, x_2 являются «короткоживущими» (соответствуют сигналам от кнопок без памяти), а переменные x_3, x_4 — длительными (соответствуют сигналам от сигнализаторов положения).

Если переменные x_1, x_2, так же как и переменные x_3, x_4 являются длительными (соответствуют, например, сигналам от тумблеров), то устранение генерирующих контуров в графе переходов и соответственно в программе может быть выполнено, например, за счет работы с «фронтами». При этом последний из приведенных фрагментов, записанный в два столбца, приобретает следующий вид:

```
STR NO z1      STR  I x2
AND NO z2      EQ   M M
IF   T         STR DP M
STR  I x1      EQ  SO z1
AND NI x2      EQ  RM M
EQ   M M       CONT
STR DP M
EQ  SO z2
```

В этом фрагменте новые команды:

`STR DR M` — запись единицы в битовый сумматор по переднему фронту дополнительно вводимой промежуточной переменной M: если значение этой переменной в начале программного цикла равно нулю, а затем — единице, то битовый сумматор устанавливается в единицу, иначе в него записывается ноль;

`EQ RM M` — сброс содержимого битовой ячейки памяти, если содержимое битового сумматора равно 1, и сохранение предыдущего значения — в противном случае.

Рассмотрим в качестве примера еще четыре программы, построенные на основе разных моделей, описывающих работу счетного триггера.

На рис. 14.13 приведен граф переходов с двумя вершинами, работающий по передним фронтам потенциальной входной переменной *x*. Этот

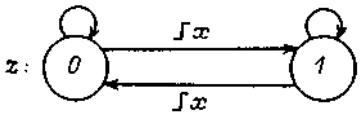

Рис. 14.13

граф реализуется первой программой, сложность которой 13; 2. Недостаток этой программы состоит в том, что ее текст не изоморфен графу переходов, по которому она строится.

Вторая программа построена наименее трудоемко — непосредственно по графу переходов с четырьмя вершинами, в котором используется потенциальная входная переменная *x* (рис. 13.22). Ее сложность 17; 1.

Третья программа, построенная по линеаризованной ГСА (рис. 13.25), является простейшей. Ее сложность 13; 1. Эту программу можно построить также и по графу переходов (рис. 13.22), применяя для кодирования вершин самые короткие коды (значение одной из двух переменных — выходной и промежуточной), так как выбранный код в сочетании со значением входной переменной может однозначно идентифицировать вершину.

Четвертая программа построена по СБФ (16.12), в которой выполнена факторизация. Сложность программы 14; 3.

Эти программы имеют следующий вид:

```
STR  I x     STR NO z    STR I  x     STR NI x
EQ   M M     AND NM y    AND NM y     AND  O z
STR DP M     AND  I x    EQ  SO z     EQ   M M
EQ  SM M1    EQ  SO z

STR NO z     STR  O z    STR NI x     STR  I x
AND  M M1    AND NM y    AND  O z     AND NM y
EQ  SO z     AND NI x    EQ  SM y     OR   M M
EQ  RM M1    EQ  SM y                 EQ   M z1

STR  O z     STR  O z    STR  I x     STR  I x
AND  M M1    AND  M y    AND  M y     AND  M y
EQ  RO z     AND  I x    EQ  RO z     OR   M M
EQ  RM M1    EQ  RO z                 EQ   M y
STOP

             STR NO z    STR NI x     STR  M z1
             AND  M y    AND NO z     EQ   O z
             AND NI x    EQ  RM y     STOP
             EQ  RM y    STOP
             STOP
```

В этих программах используются следующие команды:

AND NM j — конъюнкция содержимого БС и инверсии содержимого битовой ячейки промежуточной памяти j с записью результата вычисления в битовый сумматор;

AND M j — конъюнкция содержимого БС и содержимого битовой ячейки промежуточной памяти j с записью результата вычисления в битовый сумматор;

OR M j — дизъюнкция содержимого БС и содержимого битовой ячейки j с записью результата вычисления в битовый сумматор.

14.2.2. Использование двоичного кодирования состояний

Приведем в качестве примера две программы, реализующие графы переходов (рис. 14.14, 14.15) и содержащие в явном виде установку начального значения переменной Y_0 ($Y_0 = 1$). Первая из этих программ построена по переходам, а вторая — по вершинам:

```
READ  S  0      READ  S  0
STR   S  0      STR   S  0
EQ   SM Y0      EQ   SM Y0
STEP  S  1      STEP  S  1

STR   M Y0      IF    M Y0
AND   I x2      STR   I x1
EQ   RM Y0      AND  NI x2
EQ   SM Y2      EQ   RM Y0
                EQ   SM Y1
STR   M Y0      STR   I x2
AND   I x1      EQ   RM Y0
EQ   RM Y0      EQ   SM Y2
EQ   SM Y1      CONT

STR   M Y1      STR   M Y1
EQ    O z2      EQ    O z2
AND   I x4      AND   I x4
```

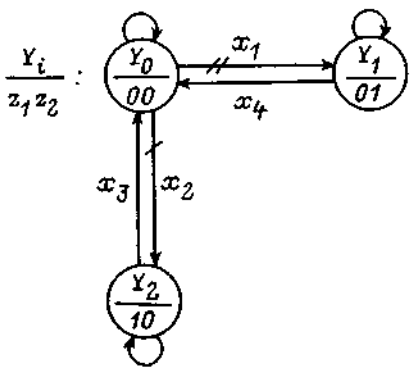

Рис. 14.14

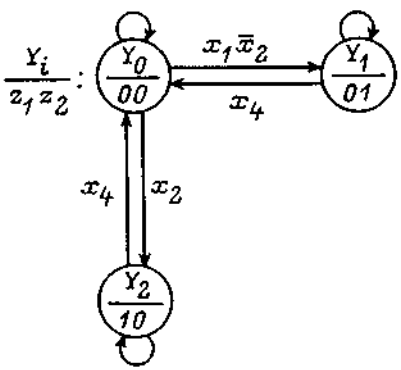

Рис. 14.15

```
EQ  RM Y1      EQ  RM Y1
EQ  SM Y0      EQ  SM Y0
STR  M Y2      STR  M Y2
EQ   O z1      EQ   O z1
AND  I x3      AND  I x3
EQ  RM Y2      EQ  RM Y2
EQ  SM Y0      EQ  SM Y0
STOP           STOP
```

Сложность первой программы 23; 3, а второй — 24; 3. Сложность программы при построении ее по СБФ — 40; 5. В этих программах не применяются команды принудительного сброса единичных значений выходных переменных (разд. 14.2.3).

14.2.3. Использование многозначного кодирования состояний

Применение конструкции IF ... CONT. При использовании этого вида кодирования применяются регистровые (R) ячейки промежуточной памяти. В этом случае целесообразно реализовывать графы переходов «по вершинам», так как построение «по переходам» в этом случае обычно более фомоздко. Ниже приведены четыре программы, реализующие в качестве примера ГП автомата Мура (рис. 14.16), в котором противоречивость переходов устранена расстановкой приоритетов и предполагается, что в начале $Y = 0$.

Первая из этих программ обладает важнейшим свойством: она позволяет реализовать в течение одного программного цикла не более одного перехода, что делает доступными для «окружения» программы значения всех выходных переменных и номера состояний и не позволяет «отфильтровать» их при любых значениях входных переменных. Это свойство обеспечивается за счет введения дополнительной регистровой промежуточной переменной Y_1 и применения переобозначения переменной Y, кодирующей номера состояний автомата ($Y = Y_1$). При этом отметим, что при реализации системы из N графов переходов можно использовать $N + 1$ регистровых промежуточных переменных. В рассматриваемой программе все фрагменты имеют одинаковую организацию и в ней отсутствуют умолчания. Сложность программы 33; 0; 2, где последняя цифра указывает число примененных в нем регистровых ячеек промежуточной памяти.

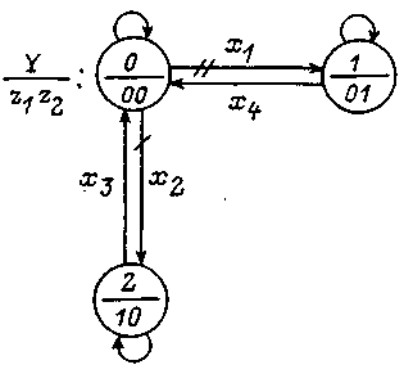

Рис. 14.16

Три другие программы указанным свойством не обладают. При этом во второй программе фрагменты разнотипны, а в третьей и четвертой — еще и умалчиваются не изменяющиеся значения выходных переменных.

При построении четвертой программы предполагается, что в начале значения выходов обнулены. Учитывается также тот факт, что во втором и третьем фрагментах не применяется конструкция IF ... CONT, а следовательно, в автомате Мура при использовании команд EQ O z (вместо команд EQ SO z) команды принудительного сброса единичного значения переменной z (EQ RO z) в первом фрагменте могут не применяться. Это объясняется тем, что в этом случае, пока автомат находится в состояниях «1» или «2», с помощью команд EQ O zi (i=1,2) формируется сигнал z, = 1, который с помощью той же команды автоматически сбрасывается при уходе автомата из состояния «i», так как рассматриваемые команды не «защищены» конструкциями IF ... CONT. Поэтому в этой программе команды, образующие второй и третий фрагменты, выполняются последовательно, формируя z = 0 в случае, когда автомат находится в нулевом состоянии. При этом отметим, что если указанная замена возможна, то она эквивалентна замене *RS*-триггеров на *D*-триггеры.

Отметим также, что в первой программе указанная замена не может быть выполнена, так как в случае такой замены при нахождении автомата в первом или втором состоянии $z_i = 1$, а при переходе из них в нулевое состояние переменная z_i не может быть сброшена, так как при этом условия в командах IF T не выполняются и осуществляется переход на метку CONT, обходя (не выполняя) команду EQ O zi и сохраняя тем самым предыдущее значение переменной r,-.

Изложенное, снижая универсальность и наглядность, приводит к упрощению остальных программ, сложность которых соответственно 25; 0; 1, 22; 0; 1, 20; 0; 1. Указанные программы имеют следующий вид:

```
STR R  C  0      STR R  C  0      STR R  C  0      STR R  C  0
EQU R  M Y       EQU R  M Y       EQU R  M Y       EQU R  M Y
IF     T         IF     T         IF     T         IF     T
EQ     RO z1     EQ     RO z1     EQ     RO z1     STR    I x1
EQ     RO z2     EQ     RO z2     EQ     RO z2     INC R  M Y
STR    I x1      STR    I x1      STR    I x1      STR    I x2
STR R  C  1      STR R  C  1      INC R  M Y       STR R  C  2
EQ  R SM Y1      EQ  R SM Y       STR    I x2      EQ  R SM Y
STR    I x2      STR    I x2      STR R  C  2      CONT
STR R  C  2      STR R  C  2      EQ  R SM Y
EQ  R SM Y1      EQ  R SM Y       CONT
CONT             CONT

STR R  C  1      STR R  C  1      STR R  C  1      STR R  C  1
EQU R  M Y       EQU R  M Y       EQU R  M Y       EQU R  M Y
IF     T         EQ     RO z1     EQ     SO z2     EQ     O z2
EQ     RO z1     EQ     SO z2     AND    I x4      AND    I x4
EQ     SO z2     AND    I x4      EQ  R RM Y       EQ  R RM Y
STR    I x4      EQ  R RM Y
STR R  C  0
EQ  R SM Y1      STR R  C  2      STR R  C  2      STR R  C  2
CONT             EQU R  M Y       EQU R  M Y       EQU R  M Y
                 EQ     SO z1     EQ     SO z1     EQ     O z1
```

```
STR R  C  2    EQ    RO z2    AND    I x3   AND    I x3
EQU R  M Y     AND    I x3    EQ  R RM Y    EQ  R RM Y
IF     T       EQ  R RM Y     STOP          STOP
EQ    SO z1    STOP
EQ    RO z2
STR    I x3
STR R  C  0
EQ  R SM Y1
CONT

STR R  M Y1
EQ  R  M Y
STOP
```

В этих программах применяются следующие новые команды:

STR R C K — запись константы со значением K в регистровый сумматор (РС);

EQU R M Y — запись в БС единицы, если содержимое регистровой ячейки промежуточной памяти Y равно содержимому регистрового сумматора, и нуля — в противном случае;

EQ R SM Y — запись содержимого РС в регистровую ячейку Y, если содержимое БС:=1, и сохранение значения Y — в противном случае;

EQ R RM Y — сброс содержимого регистровой ячейки Y, если содержимое БС:=1, и сохранение значения Y — в противном случае;

STR R M Y — ввод содержимого регистровой ячейки Y в регистровый сумматор;

EQ R M Y — запись содержимого РС в регистровую ячейку Y.

Важная особенность предлагаемого подхода состоит в том, что изменение типа графа переходов приводит к однозначным изменениям в тексте программы. Так, например, при задании рассматриваемого алгоритма в форме автомата Мили программа при начальном обнулении переменных z_1, z_2, Y и записи в виде двух столбцов приобретает следующий вид:

```
STR R  C  0      STR R  C  1
EQU R  M Y       EQU R  M Y
IF     T         AND    I x4
STR    I x1      EQ    RO z2
EQ    SO z2      DEC R  M Y
INC R  M Y
STR    I x2      STR R  C  2
EQ    SO z1      EQU R  M Y
STR R  C  2      AND    I x3
EQ  R SM Y       EQ    RO z1
CONT             EQ  R RM Y
                 STOP
```

Сложность программы 22; 0; 1. Новые команды в ней:

INC R M Y — содержимое регистровой внутренней переменной Y увеличивается на единицу и записывается, в том числе и в регистровый сумматор, если БС:=1; сохранение предыдущего значения Y — в противном случае;

DEC R M Y — содержимое регистровой внутренней переменной Y уменьшается на единицу и записывается, в том числе и в регистровый сумматор, если БС:=1; сохранение предыдущего значения Y — в противном случае.

В этой программе значения выходных переменных формируются после вычисления формул перехода, а не до их вычисления, как в автоматах Мура. Это, в частности, не позволяет отказаться в таких программах от использования команд сброса единичных значений выходных переменных.

Применение конструкции «шаговый регистр». Для реализации графов переходов с многозначным кодированием состояний наиболее адекватной является управляющая конструкция «шаговый регистр». Использование этой конструкции делает программирование весьма «высокоуровневым» и приближает язык инструкций к языкам высокого уровня. При этом отметим, что каждый шаговый регистр позволяет реализовать граф переходов, содержащий до 256 вершин.

Для работы с шаговыми регистрами существуют три основные команды.

Так как ПЛК содержит несколько таких регистров (до 32), то для выбора /-го из них применяется команда READ S I, после которой выполняются команды, относящиеся к *I*-му шаговому регистру. Каждый из таких регистров позволяет реализовать один граф переходов, и поэтому *N* шаговых регистров могут реализовать СВГП, состоящую из *N* графов переходов.

Команда STR S G проверяет, находится ли выбранный шаговый регистр на шаге (в состоянии) G, и записывает единицу в битовый сумматор, если указанный регистр находится на этом шаге, и ноль — в противном случае.

Команда STEP S G переводит шаговый регистр на шаг (в состояние) G, если содержимое битового сумматора равно единице, и не изменяет номера шага — в противном случае.

Для оптимизации длины программы существуют еще несколько команд.

Команда IF S G передает управление следующей команде, если шаговый регистр находится на шаге G, и на метку CONT — в противном случае.

Команда STEP T увеличивает содержимое шагового регистра на единицу, если БС:=1.

Команда AND S G реализует конъюнкцию содержимого БС и бита, соответствующего G-му шагу используемого шагового регистра, записывая результат в битовый сумматор.

Команда OR S G реализует дизъюнкцию содержимого БС и бита, соответствующего G-му шагу используемого шагового регистра, записывая результат в битовый регистр.

Команда XOR S G реализует неравнозначность содержимого БС и бита, соответствующего G-му шагу используемого шагового регистра, записывая результат в битовый регистр.

Покажем на примере (рис. 14.16), что использование шагового регистра позволяет существенно сократить число команд по сравнению с

применением конструкций I F . . . CONT без этого регистра и практически достичь результатов, полученных при принудительном кодировании состояний, которое не связано с введением в граф переходов и соответствующие программы избыточности, т. е. дополнительных переменных, кодирующих номера состояний автомата, так как в этом случае можно работать непосредственно с многозначными номерами состояний. Ниже приводятся две программы, каждая из которых содержит по 16 команд, причем первая из них построена по дугам графа перехода, а вторая — по вершинам:

```
READ S  0     READ S  0
STR  S  0     IF   S  0
AND  I x2     STR  I x1
STEP S  2     STEP S  1
                    STR  I x2
STR  S  0     STEP S  2
AND  I x1     CONT
STEP S  1

STR  S  1     STR  S  1
EQ   O z2     EQ   O z2
AND  I x4     AND  I x4
STEP S  0     STEP S  0

STR  S  2     STR  S  2
EQ   O z1     EQ   O z1
AND  I x3     AND  I x3
STEP S  0     STEP S  0
STOP          STOP
```

Несмотря на указанные достоинства шаговых регистров, они обладают и существенным недостатком: если исходный граф переходов не корректировать с помощью ортогонализации последовательно выполняемых переходов, то при использовании шагового регистра в программе за один программный цикл может быть реализовано более одного фрагмента (перехода в графе переходов) с фильтрацией как промежуточных значений шаговой переменной, так и значений выходных переменных. Корректировка графа в этом случае может состоять в ортогонализации входными переменными тех дуг, которые соответствуют одновременно реализуемым фрагментам программы. Для рассматриваемого примера такая корректировка совпадает с устранением генерирующих контуров в графе переходов: вместо x_4 на дуге следует записать $\bar{x}_1 \,\&\, x_4$, а вместо x_3 — $\bar{x}_2 \,\&\, x_3$

Приведем еще четыре программы, реализующие граф переходов (рис. 14.17). Первая из них содержит 21 команду и изоморфна графу по дугам (в графе шесть дуг, следовательно, в программе шесть фрагментов, каждый из которых соответствует определенной дуге). Вторая программа содержит столько же команд и изоморфна графу по вершинам (в графе четыре вершины, следовательно, в программе четыре фрагмента, каждый из которых соответствует определенной вершине). В третьей и четвертой

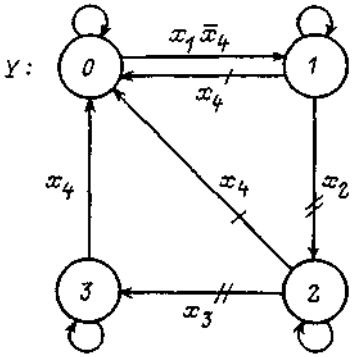

Рис. 14.17

программах с целью сокращения их длины изоморфизм нарушен. Это позволило сократить число команд в программе до 17 и 14 соответственно. Из приведенного примера следует, что, несмотря на возможность существенного сокращения длины программы, применять оптимизацию следует крайне осторожно, так как в ходе ее могут быть допущены ошибки, которые ввиду потери изоморфизма между графом переходов и текстом программы весьма трудно обнаружить. Кроме того, построение программ, неизоморфных графам переходов, весьма трудно автоматизировать. Тексты указанных программ имеют следующий вид:

```
READ  S  0      READ  S  0      READ  S  0      READ S  0
STR   S  0      STR   S  0      STR   S  0      STR  S  0
AND   I x1      AND   I x1      AND   I x1      AND  I x1
AND  NI x4      AND  NI x4      AND  NI x4      AND NI x4
STEP  S  1      STEP  S  1      STEP  S  1      STEP S  1

STR   S  1      IF    S  1      STR   S  1      STR  S  1
AND   I x4      STR   I x2      AND   I x2      AND  I x2
STEP  S  0      STEP  S  2      STEP  S  2      STEP S  2
                STR   I x4
STR   S  1      STEP  S  0      STR   S  2      STR  S  2
AND   I x2      CONT            AND   I x3      AND  I x3
STEP  S  2                      STEP  S  3      STEP S  3

STR   S  2      IF    S  2      STR   S  1      STR  I x4
AND   I x4      STR   I x3      OR    S  2      STEP S  0
STEP  S  0      STEP  S  3      OR    S  3      STOP
                STR   I x4      AND   I x4
STR   S  2      STEP  S  0      STEP  S  0
AND   I x3      CONT            STOP
STEP  S  3

STR   S  3      STR   S  3
AND   I x4      AND   I x4
STEP  S  0      STEP  S  0
STOP            STOP
```

Завершая рассмотрение вопроса о реализации автоматов, отметим, что при большом числе их выходов существенное сокращение команд в программах может быть достигнуто, если вместо вывода значений каждой отдельной переменной осуществлять параллельный вывод по восемь значений выходных переменных, что может быть обеспечено применени-

ем команды `STR R C K` («K» в данном случае является десятичным эквивалентом значений выходных переменных, формируемых в рассматриваемой вершине) и команды `BIT O 1`, осуществляющей пересылку содержимого регистрового сумматора на восемь битовых выходов, так что *i*-й бит регистрового сумматора передается на (*l* + *i*)-й выход.

14.3. Реализация управляющих автоматов

Управляющий автомат состоит из автомата и ФЭЗ. Реализация автоматов рассмотрена в предыдущих разделах настоящей главы. Система команд рассматриваемого ПЛК содержит следующие средства для программирования ФЭЗ: таймеры, импульсные переменные, команды NEXT. Эти команды могут применяться совместно только в таких графах переходов, в которых запуск ФЭЗ выполняется в вершинах графов, а не на их дугах.

14.3.1. Использование таймеров

Таймер с номером *n* реализует задержку на срабатывание длительностью в *D* секунд с помощью команды `LOAD T n D`. Отсчет времени происходит при условии, что содержимое битового сумматора равно нулю. После срабатывания таймера битовый сумматор устанавливается в единицу. Указанный принцип работы таймера приводит к тому, что при изоморфной реализации автомата, входящего в управляющий автомат, каждая единица, указанная в вершинах графа переходов на позициях, соответствующих запуску ФЭЗ, в тексте программы отражается парой команд INV (инверсия содержимого БС) и `LOAD T n D`. Достоинство таймеров состоит в том, что их сброс выполняется автоматически и поэтому нули на позициях графа переходов, соответствующих сбросу ФЭЗ, в тексте программы отражать не требуется.

Изложенный подход является первым шагом к реализации логико-вычислительных алгоритмов, при которой логика управления задается графом переходов, а вычисления выполняются, например, с помощью граф-схемы алгоритма.

При многократном вызове одного и того же таймера число команд в программе сокращается за счет уменьшения степени изоморфизма между графом переходов и текстом программы.

Приведем две программы, построенные по переходам ГП автоматов указанного класса (рис. 14.18, 14.19) с помощью изложенных подходов:

```
STR   NO  z1      STR  NO z1
AND   NO  z2      AND  NO z2
STR    I  x2      AND   I x2
EQ    SO  z1      EQ   SO z1

STR   NO  z1      STR  NO z1
AND   NO  z2      AND  NO z2
STR    I  x1      AND   I x1
EQ    SO  z2      EQ   SO z2
```

```
STR  O    z2      STR  O    z1
INV               OR   O    z2
LOAD T 10  5      INV
EQ   RO   z2      LOAD T 10  5
                  EQ   RO   z1
STR  O    z1      EQ   RO   z2
INV               STOP
LOAD T 11  5
EQ   RO   z1
STOP
```

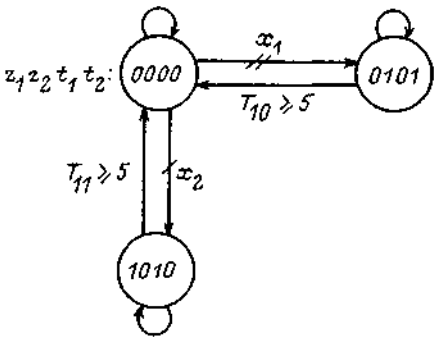

Рис. 14.18

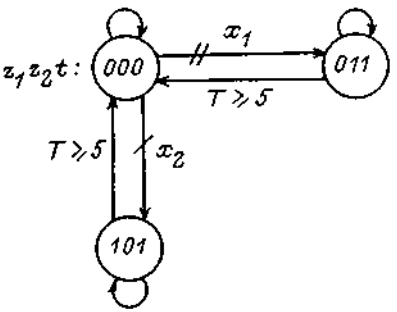

Рис. 14.19

Сложность первой программы 17; 0, а второй — 15; 0. Применение таймеров ограничено их малым количеством.

14.3.2. Использование импульсной переменной для реализации функциональных элементов задержки

Ограничение на количество ФЭЗ в программе снимается при их реализации на основе импульсной переменной P. При этом применяются следующие новые команды:

`AND PI` — конъюнкция содержимого БС и переменной P, вырабатываемой один раз в секунду, с записью результата в битовый сумматор;

`INC R M t` — увеличение регистровой переменной t на единицу и запись ее, в том числе и в регистровый сумматор, при наличии единицы в битовом сумматоре; сохранение значения t — в противном случае;

`LES R C D` — если значение D регистровой константы C меньше содержимого регистрового сумматора, то БС:=1, иначе БС:=0.

Приведем пять программ, реализованных по графу переходов на рис. 14.19, первая и вторая из которых построены по переходам, третья и четвертая — по вершинам, а пятая — по системе булевых формул, записанной по графу переходов:

```
STR  NO z1   STR  NO z1   STR  NO z1   STR  NO z1   STR  I x2
AND  NO z2   AND  NO z2   AND  NO z2   AND  NO z2   AND NO z1
EQ R RM t    EQ R RM t    EQ R RM t    EQ R RM t    AND NO z2
```

```
AND   I x2
EQ   SO z1

STR  NO z1
AND  NO z2
AND   I x1
EQ   SO z2

STR   O z2
AND   P  1
INC R M t
LES R C  5
EQ   RO z2

STR   O z1
AND   P  1
INC R M t
LES R C D
EQ   RO z1
STOP
```

```
AND   I x2
EQ   SO z1

STR  NO z1
AND  NO z2
AND   I x1
EQ   SO z2

STR   O z1
OR    O z2
AND   P  1
INC R M t
LES R C  5
EQ   RO z1
EQ   RO z2
STOP
```

```
IF    T
STR   I x1
AND  NI x2
EQ   SO z2
STR   I x2
EQ   SO z1
CONT

STR   O z2
AND   P  1
INC R M t
LES R C  5
EQ   RO z2

STR   O z1
AND   P  1
INC R M t
LES R C  5
EQ   RO z1
STOP
```

```
IF    T
STR   I x1
AND  NI x2
EQ   SO z2
STR   I x2
EQ   SO z1
CONT

STR   O z1
OR    O z2
AND   P  1
INC R M t
LES R C  5
EQ   RO z1
EQ   RO z2
STOP
```

```
EQ    M M1
STR  NM T
AND   O z1
OR    M M
EQ    M z11

STR   I x1
AND  NI x2
AND  NO z1
AND  NO z2
EQ    M M
STR  NM T
AND   O z2
OR    M M
EQ    O z2

STR   O z11
OR    O z2
AND   P  1
INC R  M t
LES R  C  5
EQ    M T

EQ  R RM t

STR   M z11
EQ    O z1
STOP
```

При этом необходимо отметить, что если первые четыре программы представляют собой одну компоненту, включающую управляющую и вычислительную части, то пятая реализована по классической автоматной модели, состоящей из двух компонент — автомата и ФЭЗ. Импульсная переменная может использоваться для формирования произвольного числа ФЭЗ. Регистровая переменная *t*, в которой «накапливается время», должна принудительно сбрасываться во фрагментах программы, соответствующих вершинам графа переходов, в которых на позициях временных переменных указаны нули.

14.3.3. Использование команд «NEXT»

При применении шаговых регистров для реализации ФЭЗ могут использоваться таймеры и (или) импульсная переменная. Однако наиболее ясная и компактная программа получается в том случае, когда без изменения структуры графа переходов удается применить команду NEXT S L D, имеющую следующую семантику: если выбранный шаговый регистр находится на шаге с номером *L* в течение *D* секунд, то он переходит на следующий по номеру шаг, в противном случае номер шага не изменяется.

Достоинство этой команды — отсутствие необходимости выполнять принудительный сброс ФЭЗ, так как он выполняется автоматически либо после его срабатывания в вершине с номером *L,* либо после ухода из этой вершины до срабатывания элемента задержки (для неполумодулярных алгоритмов [13]).

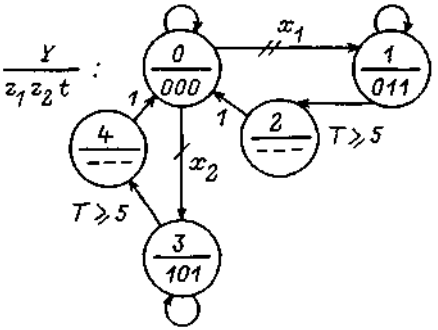

Рис. 14.20

Недостаток этой команды — необходимость перенумерации и (или) увеличения числа вершин в исходном графе переходов в случае, если после срабатывания ФЭЗ в вершине с номером L автомат должен перейти в вершину с номером, отличным от $L + 1$.

При реализации графа переходов (рис. 14.19) с помощью шагового регистра, а ФЭЗ — с помощью команд NEXT требуется заменить номер вершины «2» на «3» и ввести две новые вершины — вторую и четвертую (рис. 14.20). При этом программа, записанная в два столбца и содержащая 18 команд, имеет следующий вид:

```
READ   S   0        STR    S   2
STR    S   0        STEP   S   0
AND    I  x2
STEP   S   3        STR    S   3
                    EQ     O  z1
STR    S   0        NEXT  S 3  5
AND    I  x1
STEP   S   1        STR    S   4
                    STEP   S   0
STR    S   1        STOP
EQ     O  z2
NEXT  S 1  5
```

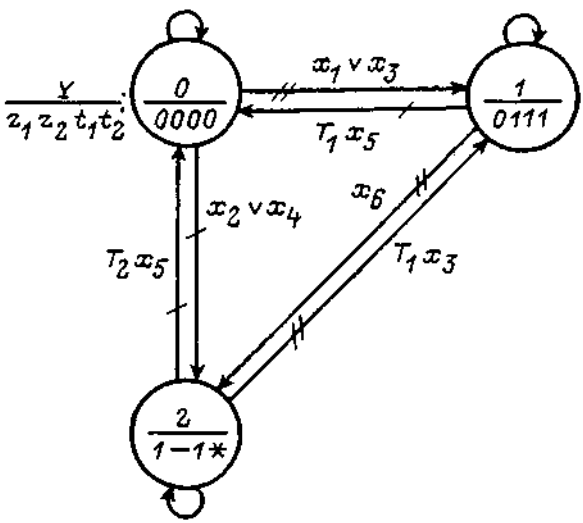

Рис. 14.21

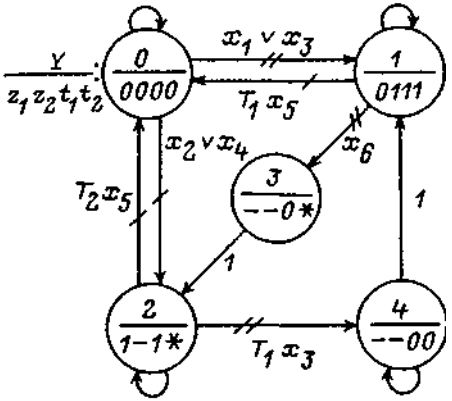

Рис. 14.22

Рассмотренные средства для программирования ФЭЗ могут применяться совместно при реализации одного графа переходов, выполняя, например, одну задержку на основе использования импульсной переменной, а другую — с помощью команды NEXT. Более того, если один и тот же ФЭЗ запускается в нескольких вершинах, то в программе он может выступать в роли различных элементов и реализовываться поэтому разными средствами. Это может позволить, не изменяя исходный граф переходов, оптимизировать длину текста программы.

Все рассмотренные графы переходов (рис. 14.18—14.20) обладали тем свойством, что из вершин, в которых запускался ФЭЗ, переход по дуге, помеченной условием срабатывания этого элемента, осуществлялся только в одну и только в смежную вершину. Это условие, записанное в виде неравенства $t >= D$, соответствовало условному выражению: «Если $t >= D$, то БС: = 1, иначе БС: = 0», и не требовало введения дополнительной битовой промежуточной переменной для запоминания факта срабатывания, так как эту функцию выполняет битовый сумматор.

Если из одной вершины указанного типа возможны переходы «по времени» в разные вершины, или из-за приоритетов переход по неравенству выполнить невозможно, или в алгоритме факт срабатывания ФЭЗ

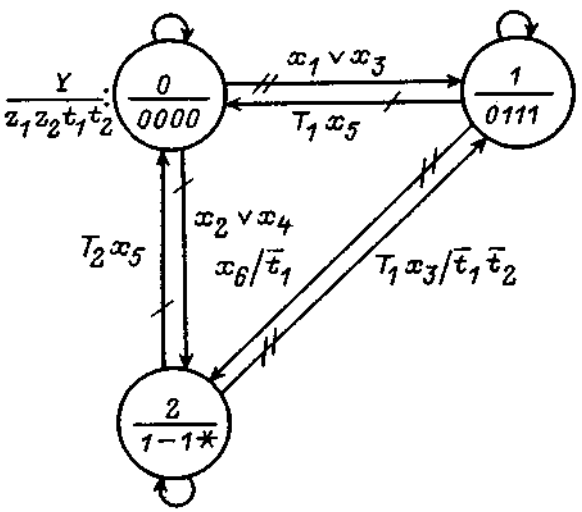

Рис. 14.23

используется для перехода не в смежную вершину, а дистанционно — для перехода в другую вершину, то факт срабатывания ФЭЗ должен запоминаться с помощью битовой промежуточной переменной *T,* которая и должна применяться в дальнейшем, возможно, наряду с приведенным выше неравенством.

Изобразительные средства графов переходов и система команд ПЛК рассматриваемого типа позволяют реализовать различные режимы работы ФЭЗ в управляющих автоматах.

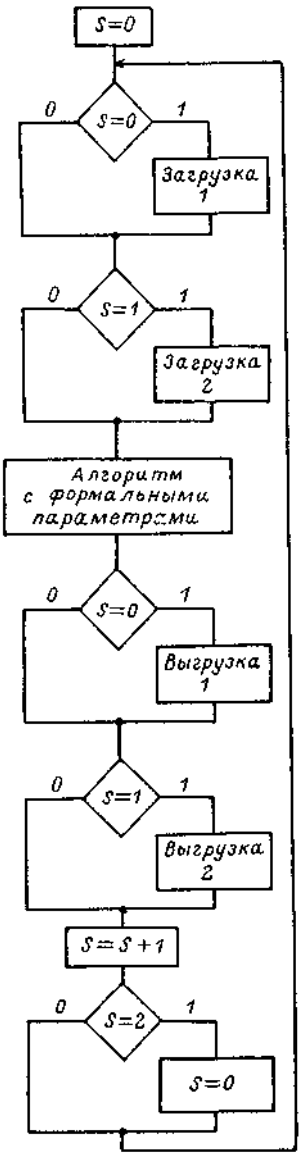

Рис. 14.24

Так, на рис. 14.21 приведен граф переходов, обеспечивающий в вершине с номером «1» параллельный запуск двух ФЭЗ ($D_1 < D_2$), что невозможно выполнить, используя только команды типа NEXT, а в вершине с номером «2» — продолжение счета времени первым ФЭЗ, если в первой вершине этот элемент еще не сработал, и приостановку счета времени вторым ФЭЗ с последующим сбросом каждого из этих элементов в нулевой вершине — при одном условии ($T_2x_5 = 1$) и продолжение счета времени этими элементами после перехода в первую вершину — при другом условии ($T_1x_3 = 1$). Прочерк в вершине «2» отражает сохранение предыдущего значения выходной переменной z_2, а звездочка — снятие (но не сброс) управляющего сигнала со второго ФЭЗ.

На рис. 14.22 и 14.23 алгоритм, реализуемый предыдущим графом переходов, откорректирован: обеспечивается двукратный запуск первого ФЭЗ и при всех переходах из второй вершины сброс второго ФЭЗ. Первое решение связано с увеличением числа вершин в ГП автомата Мура, а второе — с применением графа переходов С-автомата, в котором сброс временных переменных осуществляется не только в нулевой вершине, но и на дугах.

14.4. Реализация однотипных алгоритмов

Отсутствие команд «переход по программе назад» (за исключением перехода из конца программы в начало) не позволяет циклически реализовать за один проход программы *N* однотипных алгоритмов, отличающихся только индексацией фактических параметров. Эти алгоритмы могут быть реализованы либо за один проход программой, образованной последовательным соединением *N* фрагментов,

каждый из которых соответствует одному алгоритму, либо программой, которая в каждом из N проходов реализует один алгоритм. Во втором случае тело программы образует ее фрагмент, соответствующий заданному алгоритму (графу переходов) с формальными параметрами, в который в i-М программном цикле загружаются фактические входные параметры и из которого после их обработки выгружаются фактические выходные параметры i-го алгоритма. Такая реализация при увеличении времени выполнения для достаточно сложных алгоритмов позволяет уменьшить число команд по сравнению с однопроходной программой.

Пусть в качестве примера требуется реализовать два однотипных автомата Мура, первый из которых приведен на рис. 14.16 (с учетом замены переменной Y на Y_1), а второй может быть получен из первого после выполнения следующих замен: $x_1 \to x_5$; $x_2 \to x_6$; $x_3 \to x_7$; $x_4 \to x_8$; $z_1 \to z_3$; $z_2 \to z_4$; $Y_1 \to Y_2$

Тогда в качестве алгоритма с формальными параметрами также может быть использован ГП на рис. 14.16 после выполнения в нем следующих замен: $x_1 \to x_{10}$; $x_2 \to x_{20}$; $x_3 \to x_{30}$; $x_4 \to x_{40}$; $z_1 \to z_{10}$; $z_2 \to z_{20}$.

При этом формальными параметрами являются переменные x_{10}, x_{20}, x_{30}, x_{40}, z_{10}, z_{20}, Y, а фактическими: для первого автомата — переменные x_1, x_2, x_3, x_4, z_1, z_2, Y_1; Для второго автомата — переменные x_5, x_6, x_7, x_8, z_3, z_4, Y_2.

Граф-схема алгоритма двупроходной реализации заданных автоматов приведена на рис. 14.24, а собственно программа, построенная по этой граф-схеме и записанная в два столбца, имеет следующий вид:

```
;Начальное обнуление
READ    S  0
STR     S  0
EQ  R RM Y0
EQ  R RM Y1
STEP    S  1
;Назначение рабочего
;шагового регистра
READ    S  1
;Загрузка параметров
;первого графа переходов
IF      S  0
STR R   M Y0
EQ  R   M Y
STR     I X1
EQ      M X10
STR     I X2
EQ      M X20
STR     I X3
EQ      M X30
STR     I X4
EQ      M X40
CONT
```

```
STR R   C  1
EQU R   M Y
EQ      RM Z10
EQ      SM Z20
AND      M X40
EQ  R RM Y
STR R   C  2
EQU R   M Y
EQ      SM Z10
EQ      RM Z20
AND      M X30
EQ  R RM Y
;Выгрузка параметров
;первого графа
IF      S  0
STR R   M Y
EQ  R   M Y0
STR     M Z10
EQ      O Z1
STR     M Z20
EQ      O Z2
CONT
```

```
;Загрузка параметров
;второго графа
IF      S  1
STR R   M Y1
EQ  R   M Y
STR     I X5
EQ      M X10
STR     I X6
EQ      M X20
STR     I X7
EQ      M X30
STR     I X8
EQ      M X40
CONT
;Реализация графа
;после загрузки
;его параметров
STR R   C   0
EQU R   M Y
IF      T
EQ     RM Z10
EQ     RM Z20
STR     M X10
AND    NM X40
INC R   M Y
STR     M X20
AND    NM X30
STR R   C  2
EQ  R SM Y
CONT
```

```
;Выгрузка параметров
;второго графа
IF      S  1
STR R   M Y
EQ  R   M Y1
STR     M Z10
EQ      O Z3
STR     M Z20
EQ      O Z4
CONT
;Переход к реализации
;следующего графа
STR     C  1
STEP    T
;Завершение реализации
;всех графов
IF      S  2
STR     C  1
STEP    S  0
CONT

STOP
```

В контроллере, описанном в [232], реализация однотипных алгоритмов может быть выполнена более эффективно, так как его система команд расширена и позволяет организовывать подпрограммы. Обратим внимание также и на тот факт, что в эту систему команд кроме команд IF (начало условного выполнения) и CONT (конец всех предыдущих условных выполнений) входят также команда ELSE (начало альтернативного выполнения) и EOIF (конец последнего вложенного условного выполнения), что позволяет упростить структурирование ГСА, так как в данном случае отсутствует необходимость использования только линеаризованных и особым образом структурированных ГСА.

14.5. Реализация логико-вычислительных алгоритмов

В качестве примера приведем программу, записанную в два столбца, которая реализует алгоритм Евклида — нахождение наибольшего общего делителя двух целых положительных чисел M и N. Программирование выполняется по ГП автомата Мили с вычислениями (рис. 11.30), который

формально построен по ГСА (рис. 11.29). Приведенная ниже программа записана в два столбца:

```
READ   S  0      STR W M X
IF     S  0      LES W M Y
STR  W C M       AND   S  1
EQ   W M X       IF    T
STR  W C N       STR   C  0
EQ   W M Y       STR W M X
STR    C  1      MIN W M Y
STEP   S  1      EQ  W M X
CONT             CONT
STR  W M X       STR W M Y
EQU  W M Y       LES W M X
AND    S  1      AND   S  1
IF     T         IF    T
STR  W M X       STR   C  0
EQ   W O HOD     STR W M Y
STEP   S  0      MIN W M X
CONT             EQ  W M Y
                 CONT
                 STOP
```

Здесь W — обозначение «словных» переменных, а команда MIN W M n обеспечивает вычитание из содержимого сумматора слов переменной n и значения содержимого битового сумматора, записывая результат в сумматор слов.

14.6. Автоматизация программирования

Транслятор «Язык СИ—язык инструкций ALPro», разработанный Б. П. Кузнецовым совместно с автором, обеспечивает автоматический переход от системы взаимосвязанных ГП, изоморфно реализованной на языке СИ и промоделированной (в случае необходимости) с помощью программной оболочки (разд. 5.4.2), позволяющей наблюдать состояния каждой компоненты системы, в текст программы на рассматриваемом языке инструкций.

Произвольные графы переходов, описываемые, например, конструкцией `switch`, реализуются конструкцией `IF. . .CONT` с переобозначением переменной, кодирующей состояния. Это обеспечивает реализацию за один программный цикл не более одного перехода в графе переходов. Ниже в качестве примера приведена реализация ГП автомата Мура (рис. 14.16) с помощью конструкции `switch`, которая транслируется указанным выше образом:

```
        switch (Y)  {
case 0: z1=0; z2=0;
        if(x1)      Y=1;
        if(x2)      Y=2;
        break;
```

```
case 1: z1=0; z2=1;
        if(x4)      Y=0;
        break;
case 2: z1=1; z2=0;
        if(x3)      Y=0;
        break;}.
```

Если ограничение на число переходов за один программный цикл несущественно или возможность нескольких переходов за программный цикл отсутствует, то трансляция текста алгоритма, записанного на языке СИ, в текст программы на языке инструкций может выполняться с помощью управляющей конструкции «шаговый регистр». Так, например, ГП автомата Мили (рис. 13.8), описываемый на языке СИ следующим образом:

```
   Y=0;
M:if((Y==0)&x) {z=0; Y=1;}
   if((Y==1)&x̄) {z=1; Y=2;}
   if((Y==2)&x) {z=1; Y=3;}
   if((Y==3)&x̄) {z=0; Y=0;}
   goto M;
```

транслируется в следующую программу на языке инструкций, записанную в два столбца:

```
READ   S  0      STR   S  2
STR    S  0      AND   I x
EQ  R RM Y       EQ   SO z
STEP   S  1      STEP  S  3

READ   S  1      STR   S  3
STR    S  0      AND  NI x
AND    I x       EQ   RO z
EQ    RO z       STEP  S  0
STEP   S  1      STOP

STR    S  1
AND   NI x
EQ    SO z
STEP   S  2
```

14.7. Стековые реализации булевых формул на языке инструкций

Инструкции многих программируемых логических контроллеров ориентированы на стековые реализации булевых формул в базисе И, ИЛИ, НЕ, являющиеся разновидностями операторных реализаций. Такие реализации позволяют отказаться от использования команд записи в оперативную память промежуточных результатов. При этом программа состоит из команд ввода, битовых логических операций, в том числе и с содержимым верхушки стека, и одной команды вывода.

14.7.1. Лестничная стековая реализация

Эта реализация направлена на обеспечение взаимно однозначного соответствия между текстом программы на языке инструкций и лестничной схемой, построенной по заданной булевой формуле.

В качестве примера рассмотрим реализацию булевых формул на языке инструкций ПЛК фирмы «Omron» [236]. Отметим, что так же реализуются формулы при применении языка инструкций ПЛК и многих других фирм, например «Mitsubishi Electric» [237].

Пусть заданы бесповторные пороговые формулы [274]: $z = (x_3 \,\&\, x_4 \vee \vee x_2) \,\&\, x_1$, $z = x_1 \,\&\, (x_2 \vee x_3 \,\&\, x_4)$, $z = \bar{x}_1 \,\&\, (\bar{x}_2 \vee \bar{x}_3 \,\&\, \bar{x}_4)$. Приведем три программы, их реализующие:

```
LD   x3      LD   x1      LD   NOT x1
AND  x4      LD   x2      LD   NOT x2
OR   x2      LD   x3      LD   NOT x3
AND  x1      AND  x4      AND  NOT x4
OUT          OR   LD      OR        LD
             AND  LD      AND       LD
             OUT          OUT
```

Сложность программ 5; 7; 7 команд соответственно. Для первой из них имеет место изоморфизм между текстом программы и соответствующей лестничной (параллельно-последовательной) схемой. Для остальных программ соответствие между их текстами и схемами менее наглядно.

Важное достоинство рассматриваемого языка состоит в том, что он позволяет реализовать однотипные формулы с одинаковой сложностью вне зависимости от числа инверсий в них.

Из рассмотрения этих программ следует, что при использовании команды типа LD (ввод переменной или ее инверсии в аккумулятор) результат, вычисленный до ее применения, загружается в стек. Логические команды `OR LD` и `AND LD` выполняют операции, одним из операндов которых является верхушка стека.

Из изложенного следует, что число команд в программе, реализующей произвольную бесповторную пороговую формулу из h букв, удовлетворяет неравенству

$$h + 1 \leqslant K_{sl} \leqslant 2h - 1.$$

При этом верхняя оценка образуется из h - 1 команд типа LD, h - 1 логических команд и одной команды вывода.

При этом необходимо отметить, что за счет оптимизации порядка записи любая бесповторная пороговая формула может быть реализована с помощью $h + 1$ команд.

Сложность реализации непороговых формул может быть также оценена с помощью полученного неравенства. Приведем программы, реализующие две однотипные булевы формулы:

$$z = ((x_3 \vee x_4) \,\&\, (x_5 \vee x_6) \vee x_2) \,\&\, x_1 \quad \text{и} \quad z = x_1 \,\&\, (x_2 \vee (x_3 \vee x_4) \,\&\, (x_5 \vee x_6)),$$

```
LD   x3      LD   x1
OR   x4      LD   x2
LD   x5      LD   x3
OR   x6      OR   x4
AND  LD      LD   x5
OR   x2      OR   x6
AND  x1      AND  LD
OUT          OR   LD
             AND  LD
             OUT
```

Сложность этих программ 8 и 10 команд соответственно.

Наиболее сложными из непороговых являются инвариантные формулы, так как они обладают «пирамидальной» структурой и число команд при их реализации не может быть уменьшено за счет перестановки фрагментов. К этому классу относится, например, формула

$$z = (x_1 \vee x_2) \,\&\, (x_3 \vee x_4) \vee (x_5 \vee x_6) \,\&\, (x_7 \vee x_8),$$

которая реализуется следующей программой, записанной в два столбца:

```
LD   x1      OR   x6
OR   x2      LD   x7
LD   x3      OR   x8
OR   x4      AND  LD
AND  LD      OR   LD
LD   x5      OUT
```

Сложность программы 12 команд. В ней применяются h - 1 логических команд, [Л/2] команд ввода и одна команда вывода.

Таким образом, для инвариантных формул

$$K_{sli} = [3h/2],$$

где $[a]$ — символ округления числа *«a»* до ближайшего меньшего целого.

Необходимо отметить, что любая непороговая формула в результате перестановки фрагментов всегда может быть реализована не сложнее, чем с помощью K_{sli} команд.

Обобщая полученные результаты, можно утверждать, что рассматриваемый подход позволяет реализовать произвольную булеву формулу без оптимизации с числом команд

$$h + 1 \leqslant K_{sl} \leqslant 2h - 1,$$

а при оптимизации — с числом команд

$$h + 1 \leqslant K_{sl} \leqslant [3h/2].$$

Оптимизация порядка записи формулы в этом случае сводится к построению по ней схемы максимальной глубины из двухвходовых элементов И и ИЛИ и выделению в схеме минимально связанных между собой линейных каскадов.

14.7.2. Стековая реализация на основе обратной польской записи

Простой формальный переход от булевой формулы к тексту программы на языке инструкций [236] обеспечивается за счет использования обратной польской записи, применяемой в качестве промежуточной формы, по которой осуществляется изоморфный переход от заданной формулы к тексту программы.

Используем обратную польскую запись для следующих булевых формул:

$$z = (x_3 \,\&\, x_4 \vee x_2) \,\&\, x_1 = x_3 x_4 \,\&\, x_2 \vee x_1 \,\&\,;$$

$$z = x_1 \,\&\, (x_2 \vee x_3 \,\&\, x_4) = x_1 x_2 x_3 x_4 \,\&\, \vee \,\&;$$

$$z = \bar{x}_1 \,\&\, (\bar{x}_2 \vee \bar{x}_3 \,\&\, \bar{x}_4) = \bar{x}_1 \bar{x}_2 \bar{x}_3 \bar{x}_4 \,\&\, \vee \,\&.$$

Запишем три программы по этим записям:

```
LD   x3      LD   x1      LD   NOT x1
LD   x4      LD   x2      LD   NOT x2
AND  LD      LD   x3      LD   NOT x3
LD   x2      LD   x4      LD   NOT x4
OR   LD      AND  LD      AND       LD
LD   x1      OR   LD      OR        LD
AND  LD      AND  LD      AND       LD
OUT          OUT          OUT
```

Сложность приведенных программ одинакова и равна восьми. Из этих примеров следует, что при таком подходе сложность реализации зависит только от числа букв в формуле и не зависит от ее структуры и числа инверсий.

При этом требуется h команд ввода входных переменных, h - 1 логических команд работы с верхушкой стека и одна команда вывода. Таким образом,

$$K_{sp} = 2h.$$

При применении обратной польской записи отсутствует соответствие между текстом программы и лестничной схемой даже для бесповторных пороговых формул, записанных в порядке возрастания весов входных переменных.

В заключение раздела отметим, что при использовании обратной польской записи БФ реализуется по подформулам. Метод реализации БФ по фрагментам, которые могут и не быть подформулами, изложен в разд. 4.3.4.