Глава 15

Программная реализация автоматов в базисе языков ассемблер

Рассмотрим вопрос о программной реализации автоматов в базисе языков ассемблер на примере такого языка для микроконтроллера «Intel 8051» [136]. Особенность системы команд этого контроллера состоит в наличии команд работы с битами одного слова. При этом если для арифметических и логических команд, работающих с регистровыми переменными, один из операндов и результат должны находиться в аккумуляторе, то для битовых команд работа происходит через бит переноса C.

Рассматривается также эффективный метод реализации булевых формул на языке ассемблер микропроцессоров, в системе команд которых отсутствуют операции для работы с битами.

15.1. Реализация автоматов без памяти

15.1.1. Программирование ГСА

В качестве примера рассмотрим вопрос о программной реализации алгоритма сигнализации, заданного в табл. 14.1.

Построим, используя канонический метод Блоха [18], граф-схему алгоритма (рис. 14.1). Оптимизируем число вершин в ней за счет выноса «вверх» операторной вершины $z = 0$ (рис. 14.2). В силу того что в рассматриваемом языке условные переходы являются одноадресными, линеаризуем полученную граф-схему (рис. 14.3). Так как в системе команд отсутствует команда ввода инверсного значения входной переменной, изменим на противоположные все пометки первой условной вершины в построенной ГСА. Откорректированная граф-схема может быть реализована программой на языке ассемблер следующим образом:

```
M3: CLR     z; сброс переменной z
    MOV C,  p;  пересылка переменной p в бит переноса C
    JC      M1; переход, если бит переноса единица
    MOV C,  x
    JNC     M1; переход, если бит переноса ноль
    SETB    z;  установка переменной z
```

```
M1:   MOV C, p
      JNC    M2
      MOV C, g
      JNC    M2
      SETB   z
M2:   SJMP   M3; короткий безусловный переход
```

Сложность программы 12;0.

Ниже приводятся еще четыре программы, реализующие этот же алгоритм:

```
M3: CLR    z      M3: MOV C, p     M1: MOV C, x     M4: MOV C, p
    MOV C, x          JC     M1         ANL C,/p         JC     M1
    ANL C,/p          MOV C, x          MOV y, C         MOV C, x
    JNC    M1         MOV z, C          MOV C, p         JC     M2
    SETB   z      M1: MOV C, p          ANL C, g     M1: MOV C, p
M1: MOV C, p          JNC    M2         ORL C, y         JNC    M3
    ANL C, g          MOV C, g          MOV z, C         MOV C, g
    JNC    M2         MOV z, C          SJMP   M1        JNC    M3
    SETB   z          SJMP   M3                      M2: SETB   z
M2: SJMP   M3                                            SJMP   M4
                                                       M3: CLR    z
                                                           SJMP   M4
```

Сложность программ: 10;0, 9;0, 8;1, 12;0 соответственно.

В этих программах применяются новые команды:

`ORL C, y` — дизъюнкция бита переноса и внутренней битовой переменной с записью результата в бит переноса;

`ANL C, /p` — конъюнкция бита переноса и инверсии входной битовой переменной с записью результата в бит переноса;

`ANL C, p` — конъюнкция бита переноса и входной битовой переменной с записью результата в бит переноса.

Первая из этих программ отличается от предыдущей тем, что в ней последовательное соединение условных вершин заменено конъюнкцией (рис. 14.4).

Вторая программа построена по более компактной ГСА (рис. 14.6), в которой вместо присваивания констант выполняется присваивание переменных.

В этих программах, построенных по линеаризованным граф-схемам, порядок расположения фрагментов может быть произвольным, так как в исходной ГСА пути ортогональны. Число фрагментов в программе равно числу путей в исходной ГСА, содержащих операторные вершины.

15.1.2. Построение операторных программ

Третья программа реализована непосредственно по булевой формуле $z = \bar{p} \,\&\, x \vee p \,\&\, g$, используя выходную ячейку для запоминания промежуточных результатов. Программы этого класса называются операторными, так как в них используются логические битовые операции и не применяются условные переходы. Поэтому все команды в таких программах вне зависимости от входных наборов всегда выполняются последовательно, и в общем случае в них используются дополнительные ячейки памяти для

запоминания промежуточных результатов. Это приводит к увеличению числа команд по сравнению со случаем, когда запоминания не требуется. Без запоминания реализуются бесповторные пороговые формулы (БПФ) при их записи в порядке возрастания весов переменных. При этом число команд в программе, реализующей БПФ, которая содержит h букв в правой части, минимально и равно $h + 2$.

Верхняя оценка числа команд, равная $3h - 2$, для положительно монотонных бесповторных пороговых формул с максимальными весами и порогом достигается при их записи в порядке возрастания весов. Таким образом, для положительно монотонных бесповторных пороговых формул

$$h + 2 \leqslant K_0 \leqslant 3h - 2.$$

Приведем в качестве примера две программы, реализующие однотипные булевы формулы

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

на которых достигаются приведенные оценки:

```
M1: MOV C, x3        M1: MOV C, x1
    ANL C, x4            MOV z, C
    ORL C, x2            MOV C, x2
    ANL C, x1            MOV y, C
    MOV z, C             MOV C, x3
    SJMP   M1            ANL C, x4
                         ORL C, y
                         ANL C, z
                         MOV z, C
                         SJMP   M1
```

В первой программе применяется команда типа ORL C, x — дизъюнкция бита переноса и входной битовой переменной с записью результата в бит переноса.

Сложность программ 6;0, 10; 1 соответственно.

Первая программа реализуется по нижней оценке. Из рассмотрения второй программы следует, что она сначала с помощью $2(h - 2)$ команд вводит в память слева направо $h - 2$ буквы формулы, а затем реализует формулу (с учетом запомненных букв) в обратном порядке, используя минимальное число команд.

Для отрицательно монотонных БПФ оценка числа команд имеет вид

$$h + 3 \leqslant K_0 \leqslant 4h - 3.$$

Увеличение верхней оценки связано с необходимостью применения еще $h - 1$ команд, осуществляющих инверсии переменных. Ниже в качестве примера приведены две программы, реализующие однотипные отрицательно монотонные БПФ,

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

а третья программа построена по последней формуле, преобразованной с помощью правила де Моргана,

$$z = \overline{x_1 \vee x_2 \,\&\, (x_3 \vee x_4)}$$

и требует $3h - 1$ команд:

```
M1: CPL    x3
    MOV C, x3
    ANL C,/x4
    ORL C,/x2
    ANL C,/x1
    MOV z, C
    SJMP   M1
```

```
M1: CPL    x1;
    MOV C, x1
    MOV z, C
    CPL    x2
    MOV C, x2
    MOV y, C
    CPL    x3
    MOV C, x3
    ANL C,/x4
    ORL C, y
    ANL C, z
    MOV z, C
    SJMP   M1
```

```
M1: MOV C, x1
    MOV z, C
    MOV C, x2
    MOV y, C
    MOV C, x3
    ORL C, x4
    ANL C, y
    ORL C, z
    CPL C
    MOV z, C
    SJMP   M1
```

В этих программах используются следующие новые команды:

`ORL C, /xi` — дизъюнкция бита переноса и инверсии входной битовой переменной с записью результата в бит переноса;

`CPL xi` — инверсия бита xi;

`CPL C` — инверсия переноса.

Сложность программ 7;0, 13; 1, 11; 1 соответственно.

Из изложенного следует, что не существуют бесповторные пороговые формулы, которые для своей реализации требуют числа команд большего чем $4/h$ - 3. Таким образом, для БПФ без оптимизации порядка записи справедливо соотношение

$$h + 2 \leqslant K_0 \leqslant 4h - 3, \qquad (15.1)$$

а для БПФ с оптимизацией

$$h + 2 \leqslant K_0 \leqslant h + 3.$$

При этом отметим, что число команд в первой из трех последних программ равно $h + 3$.

Перейдем к рассмотрению класса бесповторных непороговых формул. Так как самые простые формулы этого класса реализуются сложнее простейших БПФ, а самые сложные формулы этого класса реализуются проще, чем самые сложные БПФ, то и для этого класса формул без оптимизации порядка их записи справедливо соотношение (15.1).

Наиболее сложными формулами с учетом оптимизации порядка их записи являются так называемые инвариантные булевы формулы, для которых число команд в программе не изменяется от порядка записи формул. К этому классу относятся, например, формулы

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

Ниже приведены две программы на языке ассемблер, реализующие эти формулы, а кроме того, третья программа на языке инструкций ALPro, реализующая последнюю формулу:

```
M1: MOV C, x3     M1: MOV C, x1      STR I x1
    ORL C, x4         ANL C, x2      AND I x2
    MOV z, C          MOV z, C       EQ  O z
    MOV C, x5         MOV C, x3      STR I x3
    ORL C, x6         ORL C, x4      OR  I x4
    ANL C, z          MOV y, C       EQ  M y
    MOV z, C          MOV C, x5      STR I x5
    MOV C, x1         ORL C, x6      OR  I x6
    ANL C, x2         ANL C, y       AND M y
    ORL C, z          ORL C, z       OR  O z
    MOV z, C          MOV z, C       EQ  O z
    SJMP   M1         SJMP   M1      STOP
```

Сложность программ 12;0, 12;1, 12;1 соответственно.

Число команд, требующихся для реализации положительно монотонных формул этого класса, равно *2h*. При этом число команд типа `MOV C, xi` равно *h*/2, число команд типа MOV p, C — *h*/2, число команд типа ANL и ORL — *h* - 1, число безусловных переходов — 1.

Таким образом, для положительно монотонных непороговых формул с оптимизацией порядка их записи

$$h + 2 \leqslant K_0 \leqslant 2h.$$

Так как положительно монотонные инвариантные непороговые формулы являются наиболее сложными из всех положительно монотонных непороговых формул с учетом оптимизации последних, то, увеличивая величину 2*h* на *h* - 1 (максимально возможное число инверсий), получим 3*h* - 1 — верхнюю оценку сложности для класса непороговых формул, которая, однако, является завышенной.

Объединяя полученные результаты, можно утверждать, что для положительно монотонных булевых формул в базисе И, ИЛИ с оптимизацией

$$h + 2 \leqslant K_0 \leqslant 2h,$$

без оптимизации —

$$h + 2 \leqslant K_0 \leqslant 3h - 2;$$

для класса всех булевых формул в базисе И, ИЛИ, НЕ с оптимизацией

$$h + 2 \leqslant K_0 \leqslant 3h - 1,$$

без оптимизации —

$$h + 2 \leqslant K_0 \leqslant 4h - 3.$$

Из рассмотрения класса инвариантных формул следует, что порядок их записи не влияет на число команд, но определяет число ячеек промежуточной памяти. Из приведенных примеров следует, что при реализации формул промежуточные ячейки памяти совместно с выходной ячейкой могут быть объединены в стек, в котором переменная, записанная последней, обрабатывается первой. При этом необходимое число ячеек удовлетворяет неравенству

$$0 \leqslant Я \leqslant [\log_2 h] - 2.$$

Из этого соотношения следует, что для формул из семи и менее букв всегда может быть выбран такой порядок их записи, при котором дополнительные ячейки не требуются. Порядок записи формулы для оптимизации числа команд и ячеек определяется в результате разбиения построенной по формуле схемы максимальной глубины из двухвходовых элементов на минимально связанные между собой линейные каскады. Это соответствует замене булевой формулы системой таких формул, в которой новые переменные не используются.

В заключение отметим, что результаты, полученные в настоящем разделе для положительно монотонных формул, могут применяться также для оценки сложности операторных программ, построенных на языке инструкций ALPro. Это объясняется тем, что в последнем случае командой STR может быть введена не только переменная, но и ее инверсия, и поэтому все однотипные формулы вне зависимости от расстановки и числа инверсий в них реализуются одинаковым числом команд.

В заключение раздела отметим, что если для работы с битами команда XRL («Неравнозначность») в рассматриваемом ассемблере отсутствует, то для работы с байтами она может использоваться, что позволяет в этом случае реализовать операторными программами произвольные формулы в базисе И, ИЛИ, НЕ, НЕРАВНОЗНАЧНОСТЬ.

15.1.3. Построение бинарных программ

Четвертая из программ, реализующих алгоритм сигнализации (разд. 15.1.1), относится к классу бинарных, так как при этом не применяются команды логических операций, а вместо них используются условные переходы [68]. Граф-схемы этих программ строятся непосредственно по формуле. Их сложность (но не быстродействие) не зависит от порядка записи формул и числа инверсий в них.

При этом число вершин в линейной неструктурированной планарной граф-схеме [88, 271], реализующей произвольную (в том числе любую скобочную) формулу в базисе И, ИЛИ, НЕ из h букв, определяется соотношением

$$B = h + 2.$$

Эта граф-схема содержит h условных вершин, расположенных последовательно, и две операторные: $z = 1$ и $z = 0$. Метод построения таких граф-схем, названный в [68] формульным методом, изложен в [88, 271]. На рис. 15.1 приведена граф-схема для булевой формулы $z = \bar{p}\,\&\,x \vee \vee p\,\&\,g$, реализованной формульным методом.

При изоморфном переходе от граф-схемы к бинарной программе число команд в ней удовлетворяет соотношению

$$K_b = 2(h + 2).$$

Эта оценка может быть уменьшена до $2/h + 3$ при написании программы по граф-схеме, в которой вынесена «вверх» вершина $z = 0$.

Комбинируя оценки, полученные для операторных и бинарных программ, можно утверждать, что для микроконтроллера «Intel 8051» слож-

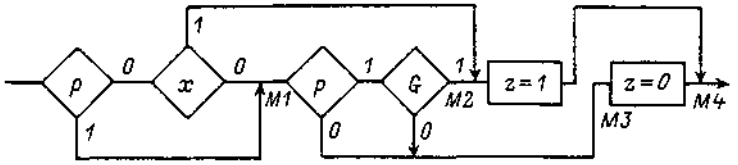

Рис. 15.1

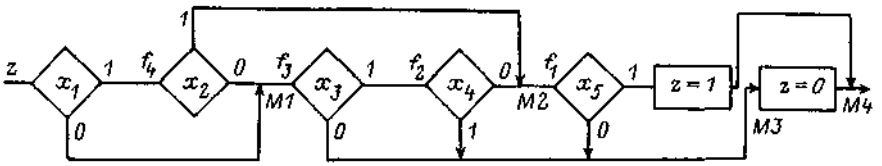

Рис. 15.2

ность реализации произвольной формулы в указанном базисе определяется неравенством

$$h + 1 \leqslant K \leqslant 2h + 3.$$

Из приведенных соотношений следует, что строить бинарные программы вместо операторных целесообразно для реализации инвариантных формул с числом инверсий больше четырех, так как для этого класса булевых формул такие программы более компактны.

Так как бинарные программы реализуются «на переходах», то в них в отличие от операторных программ запоминать промежуточные результаты не требуется при любых значениях h. Порядок записи формулы, не влияя на сложность реализации и объем памяти, определяет среднее быстродействие программы. Для БПФ максимальное быстродействие обеспечивается при их записи в порядке убывания весов. Так, например, программа, построенная по формуле $z = x_1 \vee x_2 \,\&\, x_3$, обладает более высоким средним быстродействием по сравнению с программой для формулы $z = x_2 \,\&\, x_3 \vee x_1$

Метод построения граф-схемы, обеспечивающий максимальное среднее быстродействие бинарной программы, реализующей произвольную булеву формулу в указанном базисе, изложен в [88, 273]. Метод состоит в выборе порядка записи формулы, при котором минимизируется число всех путей от входа граф-схемы до ее операторных вершин. Так, например, если граф-схема для булевой формулы

$$z = (x_1 \,\&\, x_2 \,\&\, x_3 \vee (x_4 \vee x_5) \,\&\, x_6) \,\&\, (x_7 \,\&\, x_8 \vee x_9 \,\&\, x_{10}) \vee x_{11} \,\&\, x_{12}$$

содержит 132 пути, то их минимальное число, равное 69, достигается для формулы [273]

$$z = x_{11} \,\&\, x_{12} \vee (x_6 \,\&\, (x_4 \vee x_5) \vee x_1 \,\&\, x_2 \,\&\, x_3) \,\&\, (x_7 \,\&\, x_8 \vee x_9 \,\&\, x_{10}).$$

Приведем оценки числа путей при реализации произвольной булевой формулы в базисе И, ИЛИ, НЕ из h букв линейной бинарной граф-схемой [272].

$$h + 1 \leqslant S(h) \leqslant F_{h+2},$$

$$h + 1 \leqslant S(h) \leqslant \begin{cases} 2^{1+(h/2)}, & \text{если } h = 2, 4, 6, \ldots, 16; \\ 0.5(3^{1+(h/3)} - 1), & \text{если } \mathrm{mod}_3\, h = 0 \text{ и } h \neq 6, 12; \\ 3^{(h+1)/3}, & \text{если } \mathrm{mod}_3\, h = 2 \text{ и } h \neq 2, 8, 14; \\ 2 \times 3^{(h-1)/3} + 1, & \text{если } \mathrm{mod}_3\, h = 1 \text{ и } h \neq 4, 10, 16. \end{cases}$$

Изложенный подход не может быть использован при программировании на языке инструкций ALPro, так как в нем в отличие от рассматриваемого языка ограничены возможности условных переходов, что не позволяет реализовывать неструктурированные граф-схемы непосредственно.

Рассмотренный метод, однако, является наиболее эффективным для микропроцессоров и ЦВМ, в системе команд которых отсутствуют операции для работы с битами, так как позволяет не перемещать биты внутри слова, а также между словами.

В качестве примера рассмотрим реализацию булевой формулы программой на языке ассемблер ЦВМ PDP-11 фирмы «DEC» (США). Пусть задана БФ

$$z = (x_1 \,\&\, x_2 \vee x_3 \,\&\, \bar{x}_4) \,\&\, x_5$$

а переменные x_1, x_2, x_3 размещены в 14-, 6- и 3-м разрядах первого слова, переменные x_4 и x_5 — в 5- и 2-м разрядах второго слова.Для написания программы непосредственно по формуле строится линейная граф-схема (рис. 15.2) и определяются маски, записанные в восьмеричной системе, для выделения отдельных битов из слов: 040000, 000100, 000010, 000040, 000004. Программа, реализующая полученную граф-схему, имеет следующий вид:

```
START: BIT #040000, POLE1
       BEQ M1
       BIT #000100, POLE1
       BNE M2
   M1: BIT #000010, POLE1
       BEQ M3
       BIT #000040, POLE2
       BNE M3
   M2: BIT #000004, POLE2
       BEQ M3
       MOV #1, z
       BR  M4
   M3: CLR z
   M4: END START
```

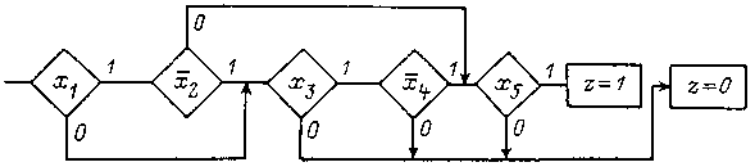

Рис. 15.3

Пусть, например

$$\text{POLE1} = 1\ \ 101\ \ 110\ \ 110\ \ 011\ \ 011 = 156633,$$

$$\text{POLE2} = 0\ \ 011\ \ 011\ \ 101\ \ 000\ \ 111 = 033507.$$

При этом $x_1 = 1$, $x_2 = 0$, $x_3 = 1$, $x_4 = 0$, $x_5 = 1$ и программа формирует $z = 1$.

Оценки сложности программы для этого языка совпадают с оценками, полученными выше для контроллера 8051.

В заключение раздела изложим метод построения по булевой формуле логической схемы алгоритма (ЛСА), являющейся строчной записью, состоящей из букв, соответствующих условиям и операторам, и исходящих и входящих стрелок, описывающих переходы.

Метод состоит из трех этапов. На первом из них по формуле строится линейная граф-схема (рис. 15.2). На втором — в случае необходимости осуществляется инвертирование пометок условных вершин с целью получения в остове этой граф-схемы только единичных пометок дуг, исходящих из условных вершин (рис. 15.3). Этот этап необходим в связи с тем, что в логической схеме алгоритмов при выполнении условия осуществляется переход к следующей букве и по стрелке — в противном случае. На третьем этапе по откорректированной линейной граф-схеме строится логическая схема алгоритма:

$$x_1 \overset{1}{\uparrow} \bar{x}_2 \overset{2}{\uparrow} \overset{1}{\downarrow} x_3 \overset{3}{\uparrow} \bar{x}_4 \overset{3}{\uparrow} \overset{2}{\downarrow} x_5 \overset{3}{\uparrow} z \overset{3}{\downarrow} \bar{z}.$$

15.1.4. Сравнение сложности реализации булевых формул программами в базисе алгоритмических языков низкого уровня

Из приведенных выше соотношений следует, что для реализации произвольных булевых формул в базисе И, ИЛИ, НЕ из h букв число команд не превышает следующих величин:

$2h + 3$ — для бинарных программ в рассмотренных языках ассемблер;

$2h$ — для операторных программ на языке инструкций ALPro и программ на языке инструкций ПЛК фирмы «Omron», использующих обратную польскую запись;

$[3h/2]$ — для лестничных программ на языке инструкций ПЛК фирмы «Omron».

Из приведенных соотношений следует, что изложенные подходы строят избыточные по отношению к величине h реализации.

Покажем, что может быть предложен такой подход, при котором произвольная булева формула в указанном базисе из h букв, содержащая

скобки произвольной глубины, реализуется с минимальной сложностью, с помощью h операторов.

Этот подход базируется на методе, впервые изложенном в [87]. Он состоит из двух этапов: построение по формуле линейной граф-схемы и запись по этой граф-схеме фрагментов формулы, а не подформул, как это делается обычно, например при использовании обратной польской записи.

Фрагменты формируются после условных вершин при просмотре граф-схемы справа налево. При этом считается, что j-я условная вершина с пометкой x_i реализует соотношение:

$$f_j = \bar{x}_i \,\&\, \varphi_0 \vee x_i \,\&\, \varphi_1,$$

где φ_0 — фрагмент формулы, «подаваемый» в вершину через дугу с пометкой «0»; φ_1 — фрагмент формулы, «подаваемый» в вершину через дугу с пометкой «1».

Приведем в качестве примера реализацию формулы

$$z = (x_1 \,\&\, x_2 \vee x_3 \,\&\, \bar{x}_4) \,\&\, x_5.$$

Линейная граф-схема, реализующая эту формулу, изображена на рис. 15.2. Читая эту граф-схему справа налево, получим следующую систему соотношений:

$$f_1 = \bar{x}_5 \,\&\, 1 \vee x_5 \,\&\, 0 = x_5;$$
$$f_2 = \bar{x}_4 \,\&\, f_1 \vee x_4 \,\&\, 0 = \bar{x}_4 \,\&\, x_5;$$
$$f_3 = \bar{x}_3 \,\&\, 0 \vee x_3 \,\&\, f_2 = x_3 \,\&\, \bar{x}_4 \,\&\, x_5;$$
$$f_4 = \bar{x}_2 \,\&\, f_3 \vee x_2 \,\&\, f_1 = (x_2 \vee x_3 \,\&\, \bar{x}_4) \,\&\, x_5;$$
$$z = \bar{x}_1 \,\&\, f_3 \vee x_1 \,\&\, f_4 = (x_1 \,\&\, x_2 \vee x_3 \,\&\, \bar{x}_4) \,\&\, x_5.$$

Из рассмотренного примера следует, что предлагаемый подход позволяет строить операторную программу, в которой на каждом шаге вводится и обрабатывается одна буква реализуемой формулы, несмотря на наличие в последней скобок произвольной глубины.

15.2. Программирование автоматов с памятью

15.2.1. Использование принудительного кодирования состояний

Рассмотрим в качестве примера реализацию R-триггера. Ниже приведены четыре программы, реализующие этот алгоритм, которые построены по граф-схемам, приведенным на рис. 13.38, 13.40, 13.41, 4.79 соответственно:

```
M3: MOV C, z     M3: MOV C, z     M3: MOV C, R     M3: MOV C, S
    JC     M1        JC     M1        JC     M1        JNC    M1
    MOV C, R         MOV C, S         MOV C, S         SETB   z
    JC     M1        JNC    M1        JNC    M1   M1: MOV C, R
    MOV C, S         SETB   z         SETB   z         JNC    M2
```

```
     JNC    M1      M1: MOV C, z       M1: MO  C, R          CLR  z
     SETB   z           JNC    M2          JNC    M2     M2: SJMP M3
M1:  MOV C, z           MOV C, R           CLR    z
     JNC    M2          JNC    M2      M2: SJMP   M3
     MOV C, R           CLR    z
     JNC    M2      M2: SJMP   M3
     CLR    z
M2:  SJMP   M3
```

Сложность программ 13;0, 11;0, 9;0, 7;0 соответственно.

Следующие четыре программы также реализуют *R*-триггер. Первая из них построена по графу переходов на рис. 4.15. В ней за счет применения универсального метода (переобозначение переменной *z*) обеспечена реализация не более одного перехода в этом графе за один программный цикл. Во второй программе переобозначение не применяется, так как в этом нет необходимости ввиду ортогональности пометок дуг в графе переходов. Эти программы во многом носят декларативный характер, так как в них фрагменты, соответствующие переходам, могут быть расположены в произвольном порядке. По этим программам граф переходов восстанавливается без дополнительных преобразований.

В третьей программе порядок расположения фрагментов не может быть изменен, так как фрагмент, расположенный ниже, обеспечивает приоритет сброса при $R = S = 1$. Изоморфный переход от текста программы к графу переходов приводит к некорректному графу, так как в нем по правилам его чтения при $R = S = 1$ на выходе z происходит генерация.

Четвертая программа построена по булевой формуле, и переход от нее к графу переходов связан с выполнением математических преобразований.

Эти программы имеют следующий вид:

```
M3: CPL    z       M3: CPL    z       M3: CLR    z       M1: MOV  C, S
    MOV C, z           MOV C, z           MOV C, z           ORL  C, z
    ANL C,/R           ANL C,/R           ANL C, S           ANL  C,/R
    ANL C, S           ANL C, S           JNC    M1          MOV  z, C
    JNC    M1          JNC    M1          SETB   z           SJMP    M1
    SETB   z1          SETB   z       M1: MOV C, z
M1: MOV C, z       M1: MOV C, z           ANL C, R
    ANL C, R           ANL C, z           JNC    M2
    JNC    M2          JNC    M2          CLR    z
    CLR    z1          CLR    z       M2: SJMP   M3
    MOV C, z1      M3: SJMP   M3
    MOV z, C
M2: SJMP   M3
```

Сложность программ 13;1, 11;0, 10;0, 5;0 соответственно.

Исходя из изложенного, в дальнейшем будем рассматривать только такие программы, которые строятся по графам переходов. Реализуем граф переходов (рис. 14.12) с помощью четырех программ:

```
M5: CPL    z1      M5: CPL    z1      M5: CPL    z1      M5: CPL    z1
    MOV C, z1          MOV C, z1          MOV C, z1          MOV C, z1
    ANL C,/z2          ANL C,/z2          ANL C,/z2          ANL C,/z2
    ANL C, x2          ANL C, x2          JNC    M1          JNC    M1
    JNC    M1          JNC    M1          MOV C, x1          MOV C, x1
    SETB   z1          SETB   z1          JNC    M2          ANL C,/x2
    CLR    z2      M1: CPL    z1          CPL    z1          JNC    M2
```

```
M1: CPL    z1
    MOV C, z1
    ANL C,/z2
    ANL C, x1
    JNC    M2
    CLR    z1
    SETB   z2
M2: MOV C, z2
    ANL C,/z1
    ANL C, x4
    JNC    M3
    CLR    z2
M3: MOV C, z1
    ANL C, x3
    JNC    M4
    CLR    z1
    CLR    z2
M4: SJMP   M5
```

```
    MOV C, z1
    ANL C,/z2
    ANL C, x1
    ANL C,/x2
    JNC    M2
    SETB   z2
M2: MOV C, z2
    ANL C, x4
    JNC    M3
    CLR    z2
M3: MOV C, z1
    ANL C, x3
    JNC    M4
    CLR    z1
M4: SJMP   M5
```

```
    SETB   z2
M2: MOV C, x2
    JNC    M1
    SETB   z1
    CPL    z2
M1: MOV C, z2
    ANL C,/z1
    ANL C, x4
    JNC    M3
    CLR    z1
    CLR    z2
M3: MOV C, z1
    ANL C,/z2
    ANL C, x3
    JNC    M4
    CLR    z1
    CLR    z2
M4: SJMP   M5
```

```
    SETB   z2
M2: MOV C, x2
    JNC    M1
    SETB   z1
M1: MOV C, z2
    ANL C, x4
    JNC    M3
    CLR    z2
M3: MOV C, z1
    ANL C, x3
    JNC    M4
    CLR    z1
M4: SJMP   M5
```

Сложность программ 27,0; 22,0; 25,0; 20,0 соответственно.

Первая и вторая программы реализованы по переходам, а третья и четвертая — по вершинам графа переходов. В первой и третьей программах используются полные коды и не применяются умолчания, а во второй и четвертой — неполные коды и умолчания. Наличие умолчаний потребовало ортогонализации пометок дуг, исходящих из нулевой вершины, в то время как при отсутствии умолчаний достаточно использовать приоритеты.

15.2.2. Применение принудительно-свободного кодирования

Ниже приводятся три программы, построенные по графу переходов (рис. 13.22) счетного триггера:

```
M5: CPL    z
    MOV C, z
    ANL C,/y
    ANL C, x
    JNC    M1
    SETB   z
M1: MOV C, z
    ANL C,/y
    ANL C,/x
    JNC    M2
    SETB   y
M2: MOV C, z
    ANL C, y
    ANL C, x
    JNC    M3
    CLR    z
M3: MOV C, y
    ANL C,/z
    ANL C,/x
    JNC    M4
```

```
M5: MOV C, x
    ANL C,/y
    JNC    M1
    SETB   z
M1: MOV C, z
    ANL C,/x
    JNC    M2
    SETB   y
M2: MOV C, y
    ANL C, x
    JNC    M3
    CLR    z
M3: CPL    z
    MOV C, z
    ANL C,/x
    JNC    M4
    CLR    y
M4: SJMP   M5
```

```
M1: MOV C, z
    ANL C,/x
    MOV y, C
    MOV C, x
    ANL C,/y
    ORL C, y
    MOV z1,C
    MOV C, x
    ANL C, y
    ORL C, y
    MOV y, C
    MOV C, z1
    MOV z, C
    SJMP   M1
```

```
      CLR     z
M4:   SJMP    M5
```

Сложность программ 22; 1; 18;1; 14;3 соответственно.

Первая из них построена по переходам графа переходов, во второй каждый переход определяется только одной переменной кода состояния и значением входной переменной, а третья построена по СБФ, найденной по ГП. Граф переходов легко восстанавливается только по тексту первой программы.

15.2.3. Применение двоичного и многозначного кодирования

Если все рассмотренные программы были построены в предположении, что в начале все переменные, кодирующие состояния, обнулены, то при двоичном кодировании в исходном состоянии $Y_0 = 1$.

Ниже приведены две программы, реализующие ГП автомата Мура (рис. 13.30, 4.124) при двоичном и многозначном кодировании состояний:

```
      SETB    Y0
M5:   MOV C,  Y0
      JNC     M1
      CLR     z1
      CLR     z2
      MOV C,  x1
      ANL C,/x2
      JNC     M2
      CLR     Y0
      SETB    Y1
M2:   MOV C,  x2
      JNC     M1
      CLR     Y0
      SETB    Y2
M1:   MOV C,  Y1
      JNC     M3
      SETB    z2
      MOV C,  x4
      JNC     M3
      CLR     Y1
      SETB    Y0
M3:   MOV C,  Y2
      JNC     M4
      SENB    z1
      MOV C,  x3
      JNC     M4
      CLR     Y2
      SETB    Y0
M4:   SJMP    M5
```

```
M5:   CJNE RO,O,  M1
      CLR         z1
      CLR         z2
      MOV       C, x1
      JNC         M2
      MOV       R1,#1
M2:   MOV       C, x2
      JNC         M1
      MOV       R1,#2
M1:   CJNE RO, 1,M3
      CLR         z1
      SETB        z2
      MOV       C, x4
      JNC         M3
      MOV       R1,#0
M3:   CJNE RO, 2,M4
      SETB        z1
      CLR         z2
      MOV       C, x3
      JNC         M4
      MOV       R1,#0
M4:   MOV       A, R1
      MOV       RO,A
      SJMP        M5
```

Сложность программ 29;3;0, 24;0;2 соответственно.

В последней программе используются новые команды, обеспечивающие работу с регистровыми переменными, причем команда `CJNE Ri, N, Mj` реализует переход на метку `Mj`, если содержимое регистра `Ri` не равно константе N, и переход к следующей команде — в противном

случае. В этой программе для кодирования состояний может применяться только одна регистровая переменная R0. При этом вторая переменная этого типа (R1) используется для обеспечения не более одного перехода за программный цикл.

15.3. Сравнение языков ALPro и ассемблер

В силу того что программируемый логический контроллер, реализующий программы, написанные на языке ALPro, построен на базе микропроцессора, программно совместимого с микроконтроллером «Intel 8051», то структуры аналогичных программ, реализованных на языках ALPro и ассемблер указанного типа, весьма близки.

В первом случае команды являются более «крупными», что обеспечивает построение более компактных текстов программ. Например, если ввод входной инверсной переменной *x* в этом ПЛК осуществляется одной командой `STR NI x`, то на рассматриваемом ассемблере для этой цели требуются две команды.

В команды ПЛК заложена семантика, позволяющая проще «читать» программу в традиционном понимании этого термина. Например, если в ПЛК ввод переменной обозначается STR, а вывод EQ, то в ассемблере оба этих действия реализуются командой MOV, а различие действий в этом случае обеспечивается порядком расположения операндов в команде.

В командах ПЛК в отличие от ассемблера в явном виде записывается только один операнд, что также увеличивает наглядность программ.

Структура условного перехода в ПЛК рассматриваемого типа, усложняя программирование, улучшает его дисциплину за счет исключения возвратов «назад» и ограничения по расстановке меток при движении по программе «вперед».

Из изложенного следует, что для целей управления применение языка ALPro более целесообразно, так как позволяет повысить надежность программирования.

Трудоемкость программирования на языке ассемблер (так же как и на языке ALPro) может быть снижена за счет использования транслятора «Язык СИ — ассемблер» и применения предлагаемой SWITCH-технологии для получения корректного и функционально проверенного текста программы на входе транслятора.