Министерство образования Российской Федерации
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет информатики
Кафедра теоретических основ информатики

УДК 811.93:681.5.01

ДОПУСТИТЬ К ЗАЩИТЕ В ГАК
Зав. кафедрой, проф., д.т.н.
_____ Ю.Л. Костюк
«___» _____ 2010 г.

Бобков Дмитрий Сергеевич

Разработка языка управления манипуляторами и компилятора для него

Дипломная работа

Научный руководитель,
Зам. начальника сектора ИО АСУ ТП МП и
ВДК ООО НПП «ТЭК» С.А. Черухин

Исполнитель,
студ. гр. 1461 Д.С. Бобков

Томск - 2010

Реферат

Дипломная работа 44 с., 11 рис., 2 табл., 7 источников, 4 прил.

ЯЗЫК УПРАВЛЕНИЯ, АВТОМАТИЗАЦИЯ, ПРОМЫШЛЕННЫЕ РОБОТЫ, МАНИПУЛЯТОР, SCADA СИСТЕМА, ПЛК, КОМПИЛЯТОР, ИНТЕРПРЕТАТОР.

Цель работы – разработать язык управления манипуляторами, компилятор для него и интерпретатор.

Метод исследования – теоретический и практический

Разработан язык управления манипуляторами, который предоставляет основные команды позиционирования и параметрирования устройства, а также дополнительные инструкции, которые позволяют формировать логику движения (управляющие инструкции). Также реализован компилятор языка, позволяющий транслировать код с языка MCL в промежуточный код, и интерпретатор промежуточного кода.

Проект готов к работе в тестовом режиме.

Оглавление

Введение

На данный момент не существует языка управления механизмами, который не зависел бы от специфики конкретной отрасли машиностроения и тех аппаратных решений, которые были реализованы для создания конкретного механизма или технологической линии. Так, каждое предприятие, занимающееся разработкой и внедрением роботизированных комплексов и технологических линий, предоставляет свою систему управления, алгоритм выполнения технологических тактов в которой в общем случае может задаваться не языком управления механизмами, а другими средствами, например, иметь табличное представление шагов технологического такта и параметров движения. Таким образом, нельзя взять готовый язык управления и реализовать его поддержку в своей системе.

Целью данной работы является:

- Разработать язык управления манипуляторами, который бы предоставил необходимые команды и инструкции для работы с манипуляторами и наряду с этим был бы прост в изучении и использовании. Использование такого языка предоставляло бы в целом больше возможностей, чем табличное представление алгоритма.
- Разработать компилятор для этого языка для трансляции программы в промежуточный код
- Реализовать интерпретатор для чтения промежуточного кода на конечном выполняющем устройстве, а именно на ПЛК семейства SIMATIC.

1 Предпосылки задачи

На данный момент существует большое количество роботизированных систем, производимых различными компаниями, такими как ABB, KUKA, Mitsubishi, Adept, FANUC. Эти компании также предоставляют для своих роботов и системы управления, которые в общем случае основаны на PC. Очевидно, что разрабатывая роботизированные механизмы (роботы), компания не ограничивается предоставлением поддержки сторонних языков программирования, как единственный путь программирования своих роботов, а предоставляет свои средства для этих целей. Необходимо заметить, что мощность и удобство этих средств напрямую зависит от того, к какому типу по характеру производства относится компания. Так, автором данной работы научно производственные компании абстрактно делятся на два типа

- Тип Б. Компании, решающие вопросы автоматизации производства напрямую. Это компании, которые занимаются разработкой и внедрением роботизированных комплексов и технологических линий. Таким образом, их производство несет «индивидуальный» характер. Поэтому в общем случае нет острой необходимости разрабатывать специализированные системы управления, которые обладали бы «избыточной» функциональностью и универсальностью. Так технологическая линия производства разрабатывается на выполнение конкретной задачи и, будучи запущенной в производство, не потребует кардинальных возможных изменений. Поэтому алгоритм работы линии «закладывается» во время ее проектирования, а для изменения параметров предоставляется необходимая система, которая хоть и обладает всеми необходимыми средствами задания алгоритма и параметров, в редких случаях может считаться мощным и универсальным средством программирования роботизированных систем.
- Тип Б. Компании, занимающиеся разработкой универсальных роботизированных систем. Проще говоря, занимаются непосредственно созданием роботов. Промышленные роботы, хотя и имеют некоторую спецификацию, все же универсальны по своему применению. Так, робот-тяжеловес, созданный для перемещения тяжелых объектов, может применяться во многих отраслях промышленности. Возможно, алгоритм для выполнения необходимой задачи будет время от времени меняться. Таким образом, компания-производитель, хотя и определяет области применения своих роботов, не может заведомо знать алгоритмы их работы. Следовательно, необходимо предоставить системы управления с необходимым программным обеспечением, которое предоставляло бы конечному пользователю мощный инструмент для программирования и параметрирования своих промышленных роботов.

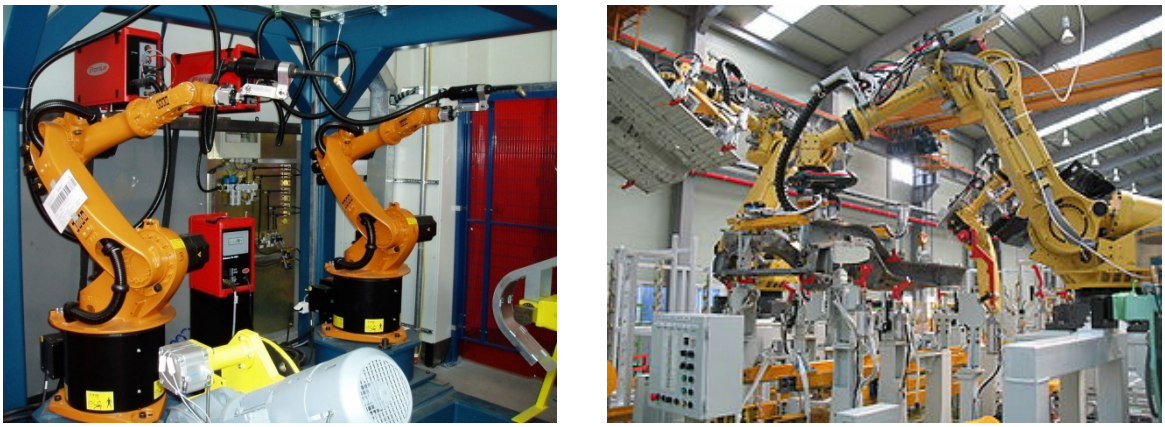
Так, компания Mitsubishi, абстрактно относящаяся к типу предприятий Б, предоставляет программное обеспечение COSIROP 2.0 для программирования своих роботов. COSIROP 2.0 поддерживает языки программирования роботов Melfa-Basic IV и Movemaster Command, которые также являются разработкой компании Mitsubishi. Это программное обеспечение позволяет создать проект для определенного типа робота, написать программу, задать позиционные точки, загрузить программу и список позиционных точек в контроллер робота, запустить программу на выполнение и т.д. Также компания Mitsubishi предоставляет программное обеспечение COSIMIR Educational, которое является средой виртуального обучения. COSIMIR Educational позволяет писать программы управления для роботов и тестировать их на виртуальных 3D моделях. Для этого могут быть использованы языки программирования роботов Melfa-Basic IV (MB4), Movemaster Command (MRL) или Industrial Robotic Language (IRL). Языки

MB4 и MRL предназначены только для программирования роботов Mitsubishi, а язык IRL подходит для программирования альтернативных серий роботов.

2 Роботы и манипуляторы

Робот (чеш. Robot - работа) – автоматическое устройство с антропоморфным действием, которое частично или полностью заменяет человека при выполнении работ в опасных для жизни условиях или при относительной недоступности объекта [1].

Роботы призваны решать задачи, которые бы освободили человека от выполнения многократно повторяющихся однотипных действий, а также выполнять те действия, которые человеку непосильны, например точные операции или передвижения тяжелых объектов. Широкое применение роботы нашли в промышленности, где они выполняют стадии технологических процессов, либо вспомогательные функции.



а) б)

Рисунок 2.1. Промышленные роботы, а – KUKA, б- FANUC

2.1 Промышленные роботы

Промышленный робот – автономное устройство, состоящее из механического манипулятора и перепрограммируемой системы управления, которое применяется для перемещения объектов в пространстве и для выполнения различных производственных процессов.

Промышленные роботы являются важными компонентами автоматизированных гибких производственных систем, которые позволяют увеличить производительность труда и повысить качество выпускаемой продукции. На рис 2.1 показаны промышленные роботы.

2.1.1 Функциональная схема промышленного робота

В составе робота есть механическая часть и система управления этой механической частью, которая в свою очередь получает сигналы от сенсорной части. Механическая часть робота делиться на манипуляционную систему и систему передвижения.

Манипулятор – это механизм для управления пространственным положением орудий и объектов труда

Манипуляторы включают в себя подвижные звенья двух типов:

- Звенья, обеспечивающие поступательные движения
- Звенья, обеспечивающие угловые перемещения

Сочетание и взаимное расположение звеньев определяет степень подвижности, а также область действия манипуляционной системы робота. Для обеспечения движения в звеньях могут использоваться электрические, гидравлические и пневматические приводы. В последнее время все больше используются сервоприводы.

Частью манипуляторов (хоть и необязательно) являются захватные устройства. Наиболее универсальные захватные устройства аналогичны руке человека – захват осуществляется с помощью механических пальцев. Для захвата плоских предметов используются захватные устройства с пневматической присоской. Для захвата же множества однотипных деталей, что обычно и имеет место быть в промышленности, применяют специализированные конструкции. Вместо захватных устройств манипулятор может быть оснащен рабочим инструментом. Это может быть пульверизатор, сварочные клещи, отвертка и т.д.

2.1.2 Программирование промышленных роботов

Практически все фирмы-производители робототехники разрабатывают собственные языки программирования и средства вспомогательного программного обеспечения. Большинство промышленных роботов имеют комплексную программную оболочку, в которую по необходимости можно интегрировать разнообразные дополнительные модули расширения. Так, например, существует возможность подключения модулей коммуникаций с внешними сенсорными устройствами: система видео наблюдения, система замера прилагаемой нагрузки, вращающего момента, что дает возможность робототехнической системе реагировать на изменение внешних условий.

Написание программ происходит в обычном текстовом редакторе, хотя некоторые роботы обладают собственными встроенными текстовыми редакторами. Программы одного производителя робота, как правило, не подходят для робота другого, по меньшей мере, без предварительной переработки. Программирование промышленных роботов делится на два вида:

- Online – программирование
- Offline – программирование

Как правило, для программирования робота могут использоваться оба вида. Существуют также различия относительно методов программирования, возможностей самих языков программирования и возможностей роботов.

Online – программирование

Online программированием называют программирование робота непосредственно на месте его установки, с помощью самого робота. К данному способу относятся два метода: Teach-In (обучение) и Playback (проигрывание).

- Метод Teach-In. При этом методе движение робота в пространстве к заданному участку производится управляющей консолью. В большинстве случаев в самом роботе (в первую ось) заложена система координат, связанная в свою очередь посредством кинематической цепи с самой удаленной точкой робота (например, 6-й осью у шестиосевого робота). Таким образом, местоположение и ориентация всех осей и предполагаемого инструмента робота в пространстве всегда известны. Достигнутое местоположение (пункт) запоминается котроллером робота и выполняется до тех пор, пока робот не выполнит все требуемые операции. Совокупность таких пунктов определяет траекторию самостоятельного движения робота. Каждый пункт имеет определенное количество изменяемых параметров таких как скорость движения и углового вращения, точность конфигурацию осей.

- Метод Playback. Робот посредством человека вручную обводится по траектории предполагаемого движения, которая в последствии в точности повторяется роботом. Этот метод часто применяется при программировании роботов, занимающихся лакированием и покраской.

К недостаткам Online – программирования относится то, что производственный процесс во время программирования приостанавливается. К тому же такое программирование не обеспечивает высокой точности обработки и конечно не очень удобно для каких-либо изменений.

Offline – программирование

Данный вид программирования производится на обыкновенном компьютере без непосредственного участия робота, что дает возможность программирования робота без остановки производственного процесса. Offline – программирование осуществляется следующими видами программирования:

- Текстовое программирование. Это обычный вид программирования, когда используется текстовой язык программирования. Программа описывает алгоритм работы всех частей робота, опрос сенсоров и т.д. Программа непосредственно или удаленно загружается в контроллер робота.
- Графическое программирование: (3D-модели). Программирование контуров обрабатываемых деталей посредством запоминания отдельных пунктов, достаточно кропотливая работа, занимающая зачастую много времени. С развитием компьютерной техники и конструкторских программ стало возможным применение CAD моделей для программирования траектории движения робота на графические модели деталей и затем интерпретировать их в язык программирования роботов. Данные программы позволяют также создавать модели и прототипы робототехнических комплексов с роботами и периферийным оборудованием, которые наглядно отображают технологический процесс. Конечно, такие программы не лишены недостатков и должны быть впоследствии адаптированы непосредственно на месте.

Преимущество таких программ в том, что они экономят массу времени и практически не останавливают производства, плюс к этому дают возможность работать с программами моделирования, которые позволяют увидеть работу робота прямо на экране монитора. Получаемое наглядное изображение дает возможность предварительной оценки многих параметров еще на стадии планирования и конструирования РТК.:

- Выбор типа робота
- Будет ли робот держать деталь или инструмент
- В состоянии ли робот достичь желаемой позиции в пространстве
- Позиционирование детали в пространстве, возможное столкновения робота и вспомогательного оборудования, время рабочего цикла и т.д.

Виртуальная оценка рабочего пространства робота со всех перспектив дает четкое представление о расположении узлов установки, что в реальности не всегда возможно.

Но есть и недостатки такого программирования. Offline-программирование предполагает наличие CAD-данных по возможности всех узлов робототехнической установки. Чем точнее и полнее данные, тем точнее осуществляется программирование робота. На практике не всегда возможно получение всех 3D-моделей и, как правило, точность 3D-моделей по сравнению с реальными оставляет желать лучшего. Также довольно трудно оценить расположение проводок водо-, газо-, и энергоснабжения монтированных на роботе и изменяющих свое положение в зависимости от конфигураций осей робота.

2.2 Манипулятор

Манипулятор можно рассматривать как прародителя современных промышленных роботов, хотя и сами манипуляторы широко используются в промышленности. В общих чертах манипулятор является тем же самым чем и является промышленный робот и выполняет схожую работу. Это подтверждает определение согласно [2] «*Манипулятор – механизм для управления пространственным положением орудий, объектов труда и конструкционных узлов и элементов*». Но это более современное определение, есть и другое, которое раскрывает первоначальный смысл слова «Манипулятор». «*Манипулятором называется техническое устройство, предназначенное для воспроизведения рабочих функций руки человека*»[3, глава 36, пар. 115]. То есть манипуляторы появлялись как механизмы, способные копировать и выполнять действия выполняемые рукой человека. Это так называемые копирующие манипуляторы. Копирующие манипуляторы состоят из управляющего и исполнительного механизмов. Вследствие механической, электрической, магнитной или какой-либо другой связи движения звеньев исполнительного механизма повторяют (копируют) движения звеньев управляющего механизма. Копирующие манипуляторы применяются во многих областях техники для выполнения операций в условиях, исключающих возможность присутствия человека возле обрабатываемого или перемещаемого изделия (радиоактивность, вакуум,

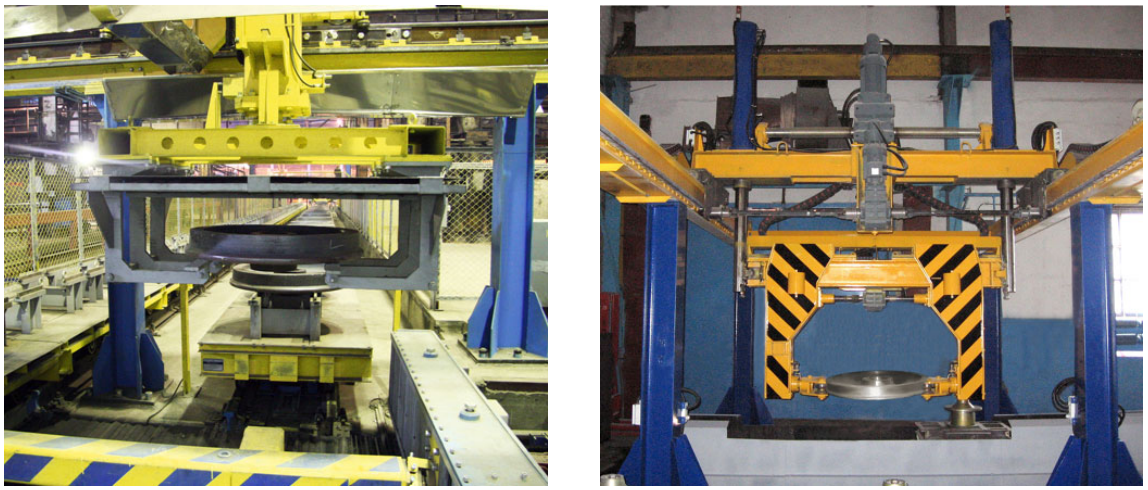


Рисунок 2.2. Промышленные манипуляторы (ТЭК)

высокая температура, повышенное давление, вредное химическое производство и т.п.). В настоящее время существуют и другие типы манипуляторов, которые могут выполнять какие-либо действия, не копируя действия управляющего механизма, а выполняя команды управляющей системы, то есть, выполняя действия по некоторому алгоритму. На рисунки 2.2 показанные портальные манипуляторы компании ТЭК.

В зависимости от системы управления различают манипуляторы с ручным управлением и манипуляторы с автоматическим управлением. В манипуляторах с ручным управлением оператор, воздействуя на звенья управляющего механизма, приводит в движение звенья исполнительного механизма. В простейших случаях передача движения может быть выполнена посредством механической связи, то есть через зубчатые колеса, тросы и рычаги. Однако в этом случае предельные усилия и перемещения исполнительного механизма ограничиваются возможностями оператора. От этого недостатка свободны манипуляторы с сервоприводами, то есть с вспомогательными приводами, которые приводят в движения отдельные звенья исполнительного механизма по сигналам, вырабатываемым при движении звеньев управляющего механизма. В таких манипуляторах легко реализуется дистанционное управление.

В манипуляторах с автоматическим управлением звенья исполнительного механизма получают движения от сервоприводов, работающих по заданной программе. Именно эти манипуляторы и являются «основой» для промышленных роботов. Автоматическое управление по-другому называется (современное название) программным управлением. В качестве аппаратного обеспечения обычно используются промышленные компьютеры в мобильном исполнении PC/104, реже MicroPC. Также такое управление может происходить с помощью ПК или программируемого логического контроллера (ПЛК). Последние «платформы» широко распространены в отечественной промышленности.

Другие типы управления, такие как адаптивное управление (сигналы, передаваемые датчиками, анализируются и, в зависимости от результатов, применяется решение о дальнейших действиях, переходе к следующей стадии действий и т.д.), управление, основанное на методах искусственного интеллекта, присущи роботизированным системам.

Технические показатели манипуляторов

Число степеней свободы манипуляторов. Манипулятор, как правило, предназначен для выполнения разнообразных движений, цель которых может изменяться не только при переходе к другому виду работ, но и при изменении внешних условий. Число степеней свободы манипулятора, как многоцелевой системы, должно выбираться в соответствии с той целью, которая требует максимальной подвижности захвата. Например, для воспроизведения пространственного движения захвата в общем случае манипулятор должен иметь шесть степеней свободы. Если же надо воспроизвести пространственную траекторию только одной точки захвата, то необходимо только три степени свободы. В промышленных роботах три степени свободы, необходимые для перемещения центра захвата в заданную точку пространства, называются переносными. Для ориентации рабочего органа (захвата) необходимые еще три степени свободы, называемые ориентирующими.

Маневренность манипуляторов. Маневренностью манипулятора называется число его степеней свободы при неподвижном захвате. Маневренность дает возможность звеньям манипулятора обходить препятствия или же располагаться в более удобной позиции при одном и том же положении захвата.

Рабочее пространство и зоны обслуживания манипуляторов. Рабочим пространством манипулятора называется пространство, в котором может находиться исполнительное устройство при функционировании манипулятора. Та часть рабочего пространства, в котором может находиться рабочий орган при функционировании манипулятора, называется рабочей зоной. Вид рабочей зоны определяется переносными степенями свободы и зависит от кинематических пар манипулятора и их взаимной ориентации. Наибольшее распространение имеют рабочие в виде плоскости, поверхности, параллелепипеда, цилиндра и шара. Видам рабочей зоны соответствуют системы координат, в которых определяются движения захвата: прямоугольная, цилиндрическая, сферическая. Однако не все части рабочей зоны одинаково удобны для выполнения заданных движений. Зоной обслуживания называется часть рабочей зоны, в которой рабочий орган выполняет свои функции в соответствии с назначением манипулятора и установленными значениями их характеристик.

3 Языки программирования роботов

Как уже было сказано, языков программирования роботизированных систем существует достаточно много. Вот только некоторые из них: AL, AML, MELFA Basic 4, Movemaster Command, IRL, KRL, RAPID, Vplus, FROB, RPL, RPS, RCCL, Saphira, Colbert. Конечно, все эти языки различаются тем, на каком уровне они предназначены работать. Так некоторые предназначены для «общения» с внешним миром, например, с оператором, другие языки предназначены для реализации логики алгоритма на машинном уровне (языки низкого уровня). Также есть отличие в задачах, которые призваны решать эти языки. Возможно, некоторые их них являются более «теоретическими» и являются результатом работ по разработке универсального способа общения человека и робота, в то время как другие, наоборот, призваны решать более прикладные задачи конкретных типов механизмов в конкретных сферах производственной деятельности человека. Также многообразие языков программирования определяется тем, что многие из них разрабатывались для конкретных платформ и сред исполнения конкретными производителями. Так, например, язык AML был разработан компанией IBM для своих роботов типа IBM RS-1, языки MB4 и MRL компанией Mitsubishi для своих систем.

Согласно уровням управления манипуляторами [3 глава 36, пар 118], можно выделить следующие уровни для языков программирования.

- Первый (низший) уровень. На этом уровне происходит программирование непосредственно узлов исполнения механизма (приводов). На этом уровне приходится брать во внимание структуру самого манипулятора. Например, чтобы позиционировать захват в необходимую точку пространства, необходимо отдать соответствующие отдельные команды позиционирования для каждой оси.
- Второй (средний или тактический) уровень. На этом уровне программирование идет командами типа ВЗЯТЬ, ПЕРЕНЕСТИ, ОТКРЫТЬ ДВЕРЦУ и т.д. Эти команды расшифровываются вычислительной машиной и переводятся на язык низшего уровня. На этом уровне, например, для позиционирования захвата необходимо только определить трехмерную точку назначения, а соответствующие команды для осей будут выработаны компилятором языка самостоятельно.
- Третий (высший или стратегический). Команды на этом уровне имеют более общую формулировку задания, например, СОБРАТЬ УЗЕЛ, РАЗГРУЗИТЬ КОНТЕЙНЕР и т.п. Эти обобщенные команды переводятся на язык низшего уровня, а возможно и тактического, с учетом информации о свойствах внешней среды, рабочих объектов, причем возможные варианты алгоритмов для достижения заданной цели определяются и сравниваются по критериям оптимизации. Например, на этом уровне для движения по заданным точкам может использоваться интерполяция. То есть в предыдущих двух уровнях позиционирование будет осуществляться к точно указным точкам, а на этом уровне позиционирование будет осуществляться по интерполяционной траектории, возможно с обработкой ситуации возникновения препятствия на пути.

Очевидно, что чем выше уровень языка, тем он сложнее в своей реализации, но более прост в своем использовании.

Основная часть перечисленных выше языков предназначена для управления универсальными роботизированными системами. Но есть и более узкие по своему применению системы, которые также призваны решать задачи автоматизации производства. Такие системы могут разрабатываться для конкретного предприятия. Примером такой системы может служить автоматизированная технологическая линия. Обычно для параметрирования и программирования такой системы могут использоваться не языки программирования, а табличное представление шагов, которые необходимо

выполнить механизму во время одного технологического такта. Таким же образом перечисляются точки назначения, из которых состоят эти шаги и параметры, с которым механизм должен выполнять тот или иной шаг. Тут необходимо сделать замечание, что не следует путать средство программирования самой системы и язык программирования аппаратной части исполнительного механизма этой системы. Так, например, если роботизированная система построена на основе программируемого логического контроллера семейства SIMATIC фирмы Siemens, то такой логический контроллер может быть запрограммирован на языке SCL (Structured Control Language). Другими словами «движок» этой системы может быть реализован средствами указанного языка. А вот средство программирования системы разрабатывается проектировщиком этой системы и предназначено для предоставления оператору технологической линии средства для формирования алгоритма работы этой системы, и может представлять собой язык программирования, табличное представления шагов, необходимых для выполнения технологических тактов или имеет иное представление.

Выбор того или иного средства формирования алгоритма зависит от конкретного проекта и, возможно, от его универсальности.

4 Автоматизация производства

Все большую роль и большее распространение приобретает автоматизация производства. Часто под этим понимают замену на некоторых этапах производства (а иногда и полностью), ручного труда на выполнение этой работы автоматизированными системами. Это приводит к улучшению качества выпускаемой продукции, увеличению скорости ее производства и, в конечном счете, к увеличению экономической выгоды. В последнее время все больше и больше предпосылок появляются для увеличения масштабов этого процесса, одними из которых являются:

- понимание самими производителями о необходимости автоматизации своего производства
- существенное снижение затрат на реализацию внедрения автоматизированных систем, благодаря развитию современной техники и науки.

Аппаратной основой для систем автоматизации могут служить промышленные компьютеры, ПК, ПЛК (программируемые логические контроллеры). Особый интерес представляют ПЛК, которые обладают достаточной вычислительной мощностью для решения большинства производственных задач,

4.1 Программируемый логический контроллер

Программируемый логический контроллер (ПЛК) (англ. Programmable Logic Controller, PLC) является основой для автоматизированных систем, выполняя задачи управляющего устройства. ПЛК является устройством реального времени. В качестве основного режима длительной работы ПЛК, зачастую в неблагоприятных условиях окружающей среды, выступает его автономное использование, без серьезного

Рисунок 4.1 ПЛК SIMATIC S7-300 (SIEMENS)

обслуживания и практически без вмешательства человека.

ПЛК обладает следующими особенностями

- Имеет развитые устройства ввода-вывода сигналов датчиков и исполнительных механизмов
- Устанавливается отдельно от управляемого при его помощи оборудования

Датчики и исполнительные устройства подключаются к ПЛК:

- Централизовано. В корзину ПЛК устанавливаются модули ввода-вывода. Датчики и исполнительные устройства подключаются непосредственно, либо при помощи согласовательных модулей, к входам/выходам сигнальных модулей

- По методу распределенной периферии, когда датчики и исполнительные устройства находиться удаленно от ПЛК и связаны с ним посредством каналов связи и, возможно, модулей расширения с использованием связи типа «ведущий-ведомый» (англ. Master-Slave).

ПЛК в своем составе не имеет развитых средств интерфейса, типа клавиатуры и дисплея. Программирование, диагностика и обслуживание ПЛК производиться, подключаемыми для этой цели, программаторами – специальными устройствами или устройствами на базе более современных технологий – персонального компьютера или ноутбука, со специальными интерфейсами и со специальными программным обеспечением (например, SIMATIC STEP 7 для ПЛК SIMATIC S7-300 или SIMATIC S7-400). В системах управления технологическими процессами ПЛК взаимодействует с различными компонентами систем человеко-машинного интерфейса (англ. human-machine interface, HMI), например, операторскими панелями, или рабочими местами операторов на базе ПК, часто промышленных, обычно через промышленную сеть.

В системах управления технологическими объектами логические команды преобладают над числовыми операциями, что позволяет при сравнительной простоте микроконтроллера (шина шириной 8 и 16 бит), получить мощные системы, действующие в режиме реального времени. В современных ПЛК числовые операции реализуются наравне с логическими. В тоже время, в отличие от большинства компьютерных процессоров, в ПЛК обеспечивается доступ к отдельным битам памяти.

4.2 SCADA системы

SCADA система (аббр. от англ. Supervisory Control And Date Acquisition, Диспетчерское управление и сбор данных) это система контроля и управления процессом с применением PC. Процесс может быть технологическим, инфраструктурным или обслуживающим:

- Технологические процессы включают — производство, выработку энергии, конструирование, переработка. Может протекать в непрерывном, пакетном, периодическом или дискретном режимах.
- Инфраструктурные процессы могут быть общественными либо частными, и включают: обработку и распределение воды, сбор и обработку сточных вод, нефте- и газо- проводы, передачу и распределение электроэнергии, системы оповещения для гражданской обороны, и большие системы связи.
- Процессы в сфере обслуживания имеют как частную, так и общественную стороны — здания, аэропорты, корабли и космические станции. Они контролируют и управляют HVAC (климат-контроль), доступом и потреблением энергии.

SCADA-система обычно содержит следующие подсистемы:

- Человеко-машинный интерфейс (**HMI**, англ. *Human Machine Interface*) — инструмент, который представляет данные о ходе процесса человеку оператору, что позволяет оператору контролировать процесс и управлять им.
- Диспетчерская система — собирает данные о процессе и отправляет команды процессору (управление).
- Абонентский оконечный блок, либо УСО (**RTU**, англ. *Remote Terminal Unit*), подсоединяемый к датчикам процесса, преобразует сигнал с датчика в цифровой код и отправляет данные в диспетчерскую систему.
- Программируемый Логический Контроллер
- Коммуникационная инфраструктура для реализации промышленной сети.

Термин SCADA обычно относится к централизованным системам контроля и управления всей системой, или комплексами систем, расположенных на больших областях (между промышленной установкой и потребителем). Большинство управляющих воздействий выполняется автоматически RTU или ПЛК. Первостепенные функции управления обычно ограничиваются по уровням отмены или контролирующему вмешательству. Например, PLC может управлять потоком охлаждающей воды внутри части производственного процесса, а SCADA система может позволить операторам изменять параметры для потока, и установить условия сигнализации, такие как — потеря потока и высокая температура, которые должны быть отображены и записаны. Цикл управления с обратной связью проходит через RTU или ПЛК, в то время как SCADA система контролирует полное выполнение цикла.

Сбор данных начинается в RTU или на уровне PLC и включает — показания измерительного прибора и отчеты об отказе оборудования (алармы или тревоги), соединенного со SCADA, по мере надобности. Далее данные собираются и форматируются таким способом, чтобы оператор диспетчерской, используя HMI мог принять контролирующие решения — корректировать или прервать стандартное управление средствами RTU/ ПЛК. Данные могут также быть помешены в Историю, часто основанную на СУБД, для построения трендов и другой аналитической обработки накопленных данных.

Системы SCADA обычно оснащаются распределенной базой данных, часто называемой базой данной тэгов. Эта база содержит элементы данных, названные тэгами или точками. Тег (точка) представляет собой единичный ввод или вывод, значения которого контролируют или регулируют в системе. Теги могут быть аппаратными (hard) (внешними) или программными (soft) (внутренними). Аппаратный тег представляет собой фактический ввод или вывод в пределах системы, в то время как тег программный - результат математических и логических операций со значениями других тегов. Большинство реализаций систем снимает концептуальное различие между «soft» и «hard» тегами, делая каждое свойство в выражении программным тегом, который может, в самом простом случае, равняться единичному аппаратному тегу. Теги обычно сохраняются как пары «значение – штамп времени»: значение, и штамп времени — то время, когда событие было зарегистрировано, или вычислено. Серия пар «значение – штамп времени» представляет собой хронологию данного тега. Также распространено сохранение дополнительных метаданных с тэгами, такими как путь до полевого устройства или регистра ПЛК, комментарии во время разработки, и сигнальная информация.

SCADA-системы решают ряд задач:

- Обмен данными с УСО (*устройства связи с объектом*, то есть с промышленными контроллерами и платами ввода/вывода) в реальном времени через драйверы.
- Обработка информации в реальном времени.
- Отображение информации на экране монитора в удобной и понятной для человека форме.
- Ведение базы данных реального времени с технологической информацией.
- Аварийная сигнализация и управление тревожными сообщениями.
- Подготовка и генерирование отчетов о ходе технологического процесса.

- Осуществление сетевого взаимодействия между SCADA ПК.
- Обеспечение связи с внешними приложениями (СУБД, электронные таблицы, текстовые процессоры и т. д.).

На рисунке 4.2 показан пример одного из окон процессов в SCADA системе (Screen).

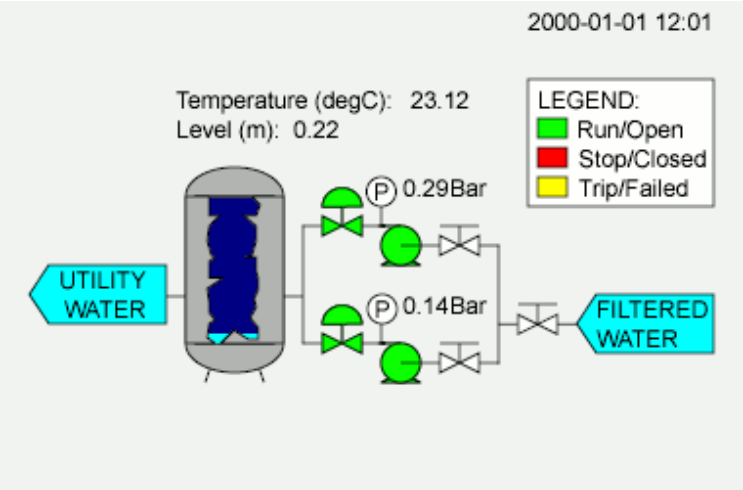


Рисунок 4.2. Пример одного из окон процесса в SCADA системе

5 Программирование ПЛК

Программирование ПЛК происходит с помощью необходимого программного обеспечения (обычно поставляется производителем ПЛК) через программаторы. В данном разделе будет рассмотрено программирование ПЛК SIMATIC S7-300 фирмы Siemens,так как это один из самых распространенных промышленных контроллеров и именно для этого ПЛК разрабатывалась практическая часть данной работы.

Программируемый логический контроллер SIMATIC S7-300/400 имеет модульную конструкцию и включает в себя следующие компоненты:

- Стойки (Rack): стойки используются для размещения в них модулей и для соединения последних друг с другом.
- Источник питания (PS . "power supply"): источник питания обеспечивает внутренние напряжения питания.
- Центральный процессор (CPU . "central processing unit"): центральный процессор используется для размещения и обработки программы пользователя.
- Интерфейсные модули (IM . "interface module"): интерфейсные модули используются для соединения стоек друг с другом.
- Сигнальные модули (SM . "signal module"): сигнальные модули используются для преобразования сигналов, поступающих от процесса, во внутренние сигналы для последующей обработки или в дискретные или аналоговые сигналы для управления приводами.
- Функциональные модули (FM . "function module"): функциональные модули не зависят от CPU, используются для выполнения сложных или зависящих от времени процессов.
- Коммуникационные процессоры (CP . " communication processor"): коммуникационные процессоры используются для связи с подсетями.
- Подсети: подсети используются для связи программируемых контроллеров друг с другом или с другими устройствами.

5.1 SIMATIC S7-300/400

Программирование ПЛК SIMATIC S7-300 (и SIMATIC S7-400) осуществляется с помощью программного обеспечения SIMATIC STEP 7 фирмы Siemens. Программирование в этой среде может производиться на следующих языках программирования[4]:

- LAD (Контактный план). Это графическое представление языка программирования Step 7, его синтаксис для команд похож на релейно-контактные схемы: такая схема дает возможность проследить поток энергии между шинами при его прохождении через различные контакты, составные элементы и выходные катушки.
- FBD (Функциональный план). это графическое представление языка программирования STEP 7, использующее для представления логики логические блоки подобно булевой алгебре. Сложные функции (например, математические функции) могут быть представлены непосредственно в соединении с логическими блоками.
- STL (Список команд). Текстовое представление языка программирования STEP 7, подобное машинному коду. Если программа написана в виде списка команд, то отдельные команды соответствуют шагам, с помощью которых CPU исполняет программу. Для облегчения программирования список команд расширен путем включения в него некоторых конструкций языков высокого уровня (таких как доступ к структурированным данным и параметры блоков).

- SCL. Текстовое представление языка программирования подобное PASCAL.[5, гл. 27]

Вне зависимости от того, какой язык используется, общий подход к программированию SIMATIC S7-300/400 следующий:

- Создается проект в Simatic Manger, в котором определяется конфигурация ПЛК, то есть при загрузке программы в ПЛК, загружается не только сама программа, но и конфигурация той аппаратной части, на которой и будет выполняться эта программа. Такая архитектура состоит из наименования стойки и модулей (блок питания, процессор, коммуникационный процессор, модули ввода/вывода) которые на ней находиться.
- Создаются исходные тексты программ. В одном проекте могут использоваться исподники, написанные на различных языках программирования из списка указанного выше.
- На основе исходных текстов средствами компиляции создаются блоки программы. Когда блоки созданы, в общем случае не важно, на каком языке были написаны их исходные тексты. Также скомпилированные блоки могут использоваться далее в программе через их вызовы (похоже на вызов обычных функций в языках, типа Pascal). Так же, блоки могут быть помещены в проект путем импортирования их из библиотеки блоков. Именно блоки загружаются в ПЛК и обрабатываются операционной системой контроллера.

Блоки бывают 3 типов:

- User block (пользовательские блоки). Это блоки, содержащие пользовательскую программу и пользовательские данные
- System block (системные блоки), блоки, содержащие системную программу и системные данные
- Standard block (стандартные блоки). Это готовые к использованию блоки, такие, например, как драйверы для функциональных блоков FM или коммуникационных процессоров CP

Пользовательские блоки:

- FB (Function Block) – функциональные блоки. Эти блоки являются частями программы, вызов которых может быть запрограммирован с помощью параметров блока. Они обладают областью памяти для переменных (variable memory), которая расположена в блоке данных. Этот блок данных постоянно назначен функциональному блоку, или, точнее, *вызову* функционального блока. Возможно даже назначение нескольких блоков данных (с одинаковой структурой данных, но содержащих разные значения) каждому вызову функционального блока. Такой постоянно назначенный блок данных называется *экземплярным блоком данных* (instance data block), а совокупность вызова функционального блока и экземплярного блока данных называется *экземпляром вызова* (call instance) или, для краткости, "экземпляром" ("instance"). Функциональные блоки могут также хранить свои переменные в экземплярном блоке данных вызывающего функционального блока; тогда такой экземплярный блок данных называется "локальным экземпляром" ("local instance").
- *OB (*Organization Block*)* – организационные блоки. Этот тип блоков служит своеобразным интерфейсом между операционной системой и пользовательской программой. Операционная система CPU вызывает организационные блоки при возникновении особого события, например, аппаратного прерывания или прерывания времени суток. Главная программа находится в организационном

блоке OB 1. Остальные организационные блоки имеют постоянные назначенные номера, основанные на событиях, для обработки которых они вызываются.

- FC (Functions) – функции. Функции используются для программирования часто повторяющихся или сложных функций автоматики. Функциям могут назначаться параметры. Функции могут возвращать значение (значение вызванной функции) в вызывающий блок. Причем значение функции - необязательный параметр. Кроме функционального значения функция может иметь другие выходные параметры. Функции не сохраняют информацию и не имеют назначенных блоков данных, другими словами переменные, объявленные в FC, своих значений не сохраняют.
- DB (Date Block) – блоки данных. Эти блоки содержат данные программы. Программируя блоки данных, мы определяем, в какой форме данные будут сохраняться (в котором блоке, в каком порядке и с каким типом данных). Существует два способа использования блоков данных: как блоки глобальных данных (global data blocks) и как экземплярные блоки данных (instance data blocks). Блоки глобальных данных в пользовательской программе являются, как говорится, "свободными" блоками данных и не назначаются кодовому блоку. Экземплярные блоки данных, однако, назначаются функциональному блоку и сохраняют часть локальных данных этого функционального блока.

Максимальное число для каждого типа блоков и размер этих блоков определяются типом CPU. Число организационных блоков и их номера фиксированы; они назначаются операционной системой CPU. Блокам других типов можно самостоятельно назначать номера внутри определенных пределов. Также можно выбрать для каждого блока имя (символ) в таблице символов, с тем, чтобы ссылаться на блок по символьному имени.

Системные блоки:

Системные блоки являются компонентами операционной системы. Они могут содержать программы (системные функции SFC или системные функциональные блоки SFB) или данные (системные блоки данных SDB). Системные блоки предоставляют множество важных системных функций, доступных пользователю, таких, например, как функции управления внутренними часами CPU или различные коммуникационные функции. Пользователь может вызывать SFC и SFB, но не может, ни изменять эти функции, ни запрограммировать их самостоятельно. Собственно системные блоки не занимают места в пользовательской памяти (user memory); только вызовы блоков и экземплярные блоки данных для SFB располагаются в пользовательской памяти.

5.2 Область памяти CPU

На рисунке 5.1 показаны области памяти CPU, имеющие значение для программы пользователя. Программа пользователя собственно располагается в двух областях, а именно в загрузочной памяти (load memory) и в рабочей памяти (work memory).

Загрузочная память (load memory) конструктивно может быть частью CPU или может быть в виде встраиваемого отдельного модуля памяти. Вводимая пользователем программа, включая данные конфигурирования, располагается в загрузочной памяти и в оперативной памяти.

Рабочая память (work memory) конструктивно является частью CPU и представляет собой быструю RAM-память. В оперативной памяти содержатся релевантные части программы пользователя: собственно код программы и данные пользователя. Здесь "релевантность" означает, что в эту память загружается код, описывающий существующие объекты, но это не предполагает обязательность вызова отдельных блоков этого кода для обработки.

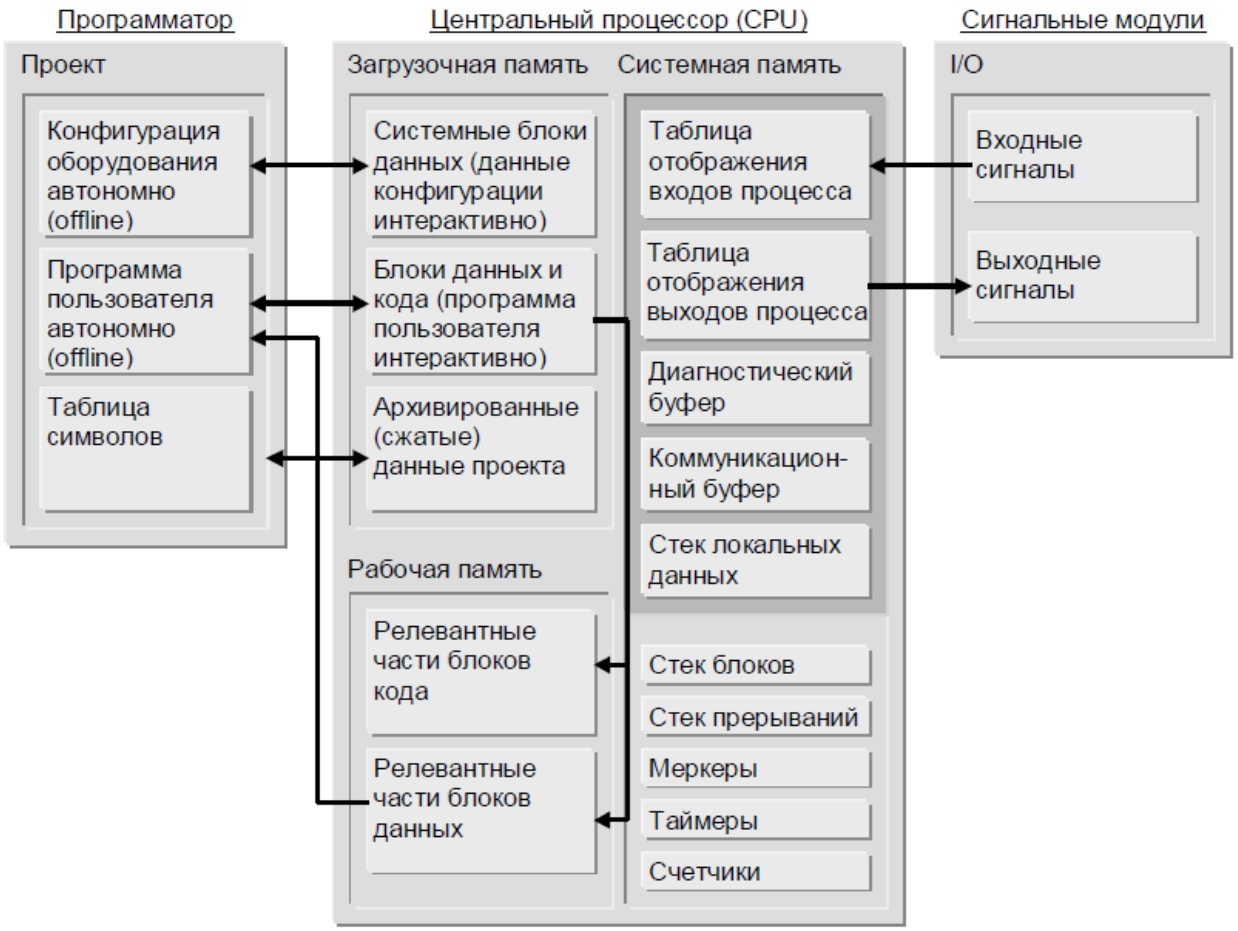


Рисунок 5.1 Области памяти CPU

5.3 Выполнение программы в CPU

В CPU всегда исполняются две программы:

- операционная система.
- программа пользователя.

Операционная система

Каждый CPU содержит операционную систему, которая организует все функции и последовательности в CPU, не связанные с конкретной задачей управления. Задачи операционной системы состоят в следующем:

- обработка "теплого" и "горячего" перезапуска
- обновление таблицы образа процесса для входов и вывод таблицы образа процесса для выходов
- вызов программы пользователя
- обнаружение прерываний и вызов OB прерываний
- обнаружение и обработка ошибок
- управление областями памяти

- обмен информацией с устройствами программирования и другими коммуникационными партнерами

Программа пользователя

Пользователь самостоятельно создает необходимую программу и загружает ее в CPU. Эта программа должна содержать все функции, необходимые для обработки конкретной задачи автоматизации. Задачи программы пользователя состоят в следующем:

- определение условий для "теплого" и "горячего" перезапуска в CPU (например, инициализация сигналов с определенным значением)
- обработка данных процесса (например, логическая комбинация двоичных сигналов, считывание и анализ аналоговых сигналов, задание двоичных сигналов для вывода, вывод аналоговых значений)
- определение реакции на прерывания
- обработка нарушений в нормальном исполнении программы.

Циклическая обработка программы – это "стандартный" способ исполнения программы в программируемых логических контроллерах, означающий, что операционная система работает в программном цикле и вызывает организационный блок OB1 один раз в каждом цикле главной программы. Поэтому программа пользователя в OB1 исполняется циклически (см. рис. 5.2).

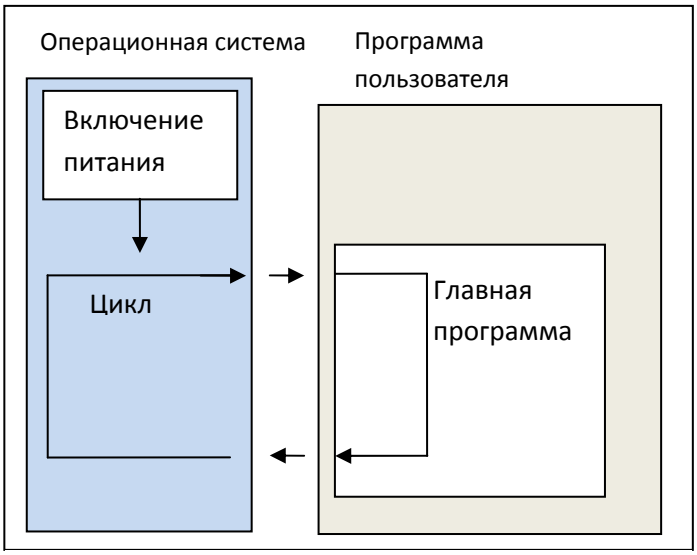


Рисунок 5.2. Циклическая обработка программы в ПЛК

Циклическая обработка программы может быть прервана определенными событиями (прерываниями). Если такое событие происходит, то блок, обрабатываемый в это момент времени, прерывается на границе команды, и вызывается другой организационный блок, назначенный соответствующему событию. Как только этот организационный блок завершает свою работу, циклическая программа возобновляется с точки, на которой она была прервана. Это значит, что имеется возможность обрабатывать части программы пользователя, которые не должны обрабатываться циклически, а только тогда, когда они необходимы. Программа пользователя может быть разделена на "подпрограммы" и распределена между различными организационными блоками. Если программа пользователя должна реагировать на важный сигнал, который появляется относительно редко (например, датчик граничного значения для измерения уровня в резервуаре сообщает, что достигнут максимальный уровень), то подпрограмма, которая

должна обрабатываться, когда выдается этот сигнал, может быть помещена в OB, обработка которого управляется событиями.

Иерархия вызовов в программе пользователя

Для функционирования программы пользователя, составляющие ее блоки должны вызываться. Это делается с помощью специальных команд STEP 7, вызовов блоков, которые могут быть запрограммированы и запущены только в логических блоках. Порядок и вложение вызовов блоков называется иерархией вызовов. Количество блоков, которые могут быть вложены друг в друга (глубина вложения), зависит от конкретного CPU. Следующий рисунок иллюстрирует порядок и глубину вложения вызовов блоков внутри цикла обработки программы.

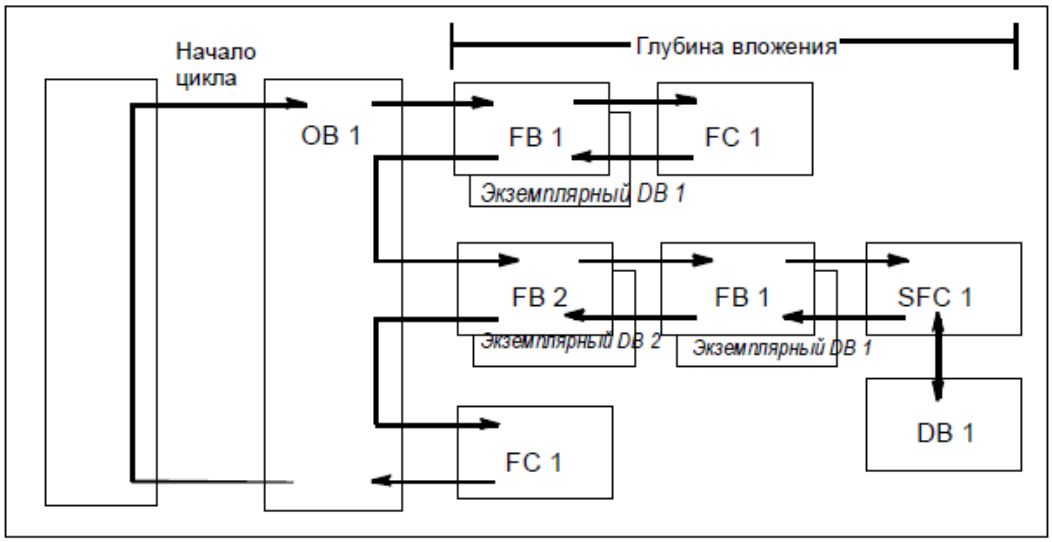


Рисунок 5.3. Порядок вызовов блоков

Существует установленный порядок создания блоков:

- Блоки создаются сверху вниз, то есть с верхнего ряда блоков.
- Каждый вызываемый блок уже должен существовать, т. е. внутри ряда блока они должны создаваться справа налево.
- Последним создается блок OB1.

Время выполнения цикла

Время выполнения цикла – это время, необходимое операционной системе для выполнения циклической программы и всех программных секций, прерывающих цикл (например, выполнение других организационных блоков), и системных операций (например, обновления образа процесса). Это время контролируется. Время выполнения цикла (TC) не одинаково в каждом цикле. На следующем рисунке показаны различные времена выполнения циклов (TC1 ≠ TC2) для CPU от 10/98:

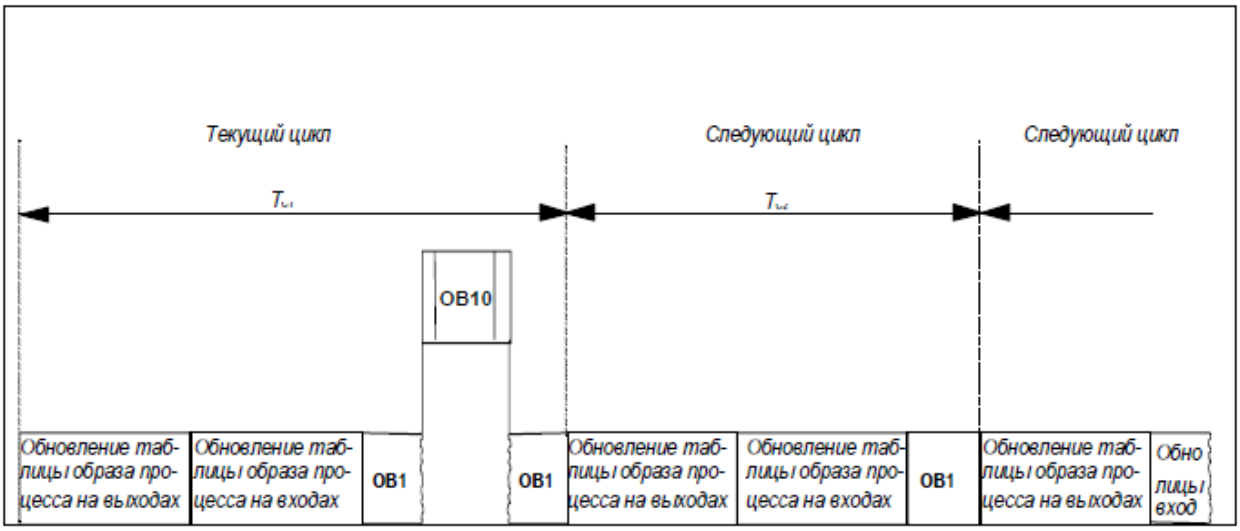


Рисунок 5.4. Время выполнения цикла

Сканирование программы в организационном блоке OB 1 отслеживается с помощью так называемого "монитора цикла сканирования" ("scan cycle monitor") или "таймера цикла сканирования" ("scan cycle watchdog). Значение, которое принимается по умолчанию для времени мониторинга цикла сканирования, равно 150 мс. Пользователь может изменять это значение в пределах от 1 мс до 6 с путем соответствующей параметризации CPU. Если сканирование основной программы выполняется за больший промежуток времени, чем установленное время мониторинга цикла сканирования, тогда CPU вызывает OB 80 ("Timeout" - "Превышение времени"), в котором пользователь может определить, как CPU должен реагировать на эту ошибку. Если организационный блок OB 80 не запрограммирован, то CPU переходит в состояние STOP.

6 Язык управления манипуляторами (Manipulators Control Language, MCL)

Целью данной работы является разработать и реализовать систему управления манипуляторами, которая предоставляла бы текстовый вариант формирования алгоритма движения механизмов (язык программирования (управления)). В предыдущих главах был представлен обзор понятий «манипулятор» и «робот». И хотя зачастую промышленные роботы и манипуляторы решают одинаковые задачи, все же эти термины не могут считаться синонимами. Так, термин «робот» несет оттенок «интеллектуальности» системы, под «манипулятором» - понимается механизм, выполняющий функции руки человека (такие, как поднятие, перетаскивание, установка объекта и т.д.). Манипуляторы, какие бы задачи они не решали и какое конструкторское решение бы не несли, имеют один и тот же принцип реализации движения. А именно, основой является понятие оси, как механической части, реализующей одну из степеней свободы данного механизма. Движение оси осуществляется с помощью электропривода (сервопривода). Следовательно, для управления манипулятором необходимо управлять каждым электроприводом. Каждый манипулятор имеет свое конструкторское решение и нельзя предугадать какое количество осей должно быть задействовано для того или иного движения, поэтому язык управления, в данном случае, для универсальности должен предоставлять команды управления каждым электроприводом отдельно. Следовательно, и параметрирование осей должно происходить раздельно.

Условия движения формируются на основе состояния датчиков. Эти условия определяют, когда механизм должен начинать движение, когда должен завершать движение. Каждый манипулятор «обладает» своим набором датчиков, датчики всей же производственной лини могут быть объединены в одну логическую группу, в которой доступ к конкретному датчику осуществляется через его номер. Номер конкретного датчика и его «роль» определяется конкретной реализацией производственной линии. Конечно, в данном случае идет речь только о дискретных датчиках. Таким образом, язык MCL предоставляет набор из 3000 булевских элементов. Элемент такого набора называется условием (condition). Язык MCL предоставляет команды чтения, установки и сброса состояния условия, а так же команду для задания символьного имени для конкретного условия, чтобы пользователь мог обращаться к данному условию не только по его номеру, но и по имени, что делает код более читабельным и понятным.

Формирование логики поведения механизма обеспечивают управляющие инструкции:

- For-инструкция. Циклически выполняет последовательность команд заданное количество раз.
- While-инструкция. Циклически выполняет последовательность команд, пока выполняется условие цикла.
- If-инструкция. Позволяет выбрать один из нескольких возможных вариантов развития программы. Выбор осуществляется в зависимости от выполнения условия.

Спецификация языка MCL

Набор знаков языка:

- Символы от a до z в нижнем и верхнем регистрах
- Арабские цифры от 0 до 9
- Другие знаки
    - Пробел

o + - * / = () : ; . {} < > “ ” ‘ _

В таблице 6.1 представлено описание использования знаков внутри синтаксических правил или для комментариев.

Таблица – 6.1

Символ	Описание
;	Конец инструкции
.	Разделитель в вещественном числе
( )	Операторные скобки для изменения приоритетов вычисления в выражении
{ }	Блок комментариев
“ ”	Определяют строковый литерал
‘ ’	Определяют символьный литерал
:=	Оператор присвоения
+ - / *	Арифметические операции
< <= > >= <> =	Операции сравнения

Идентификаторы

Идентификаторы переменных строятся по схеме:

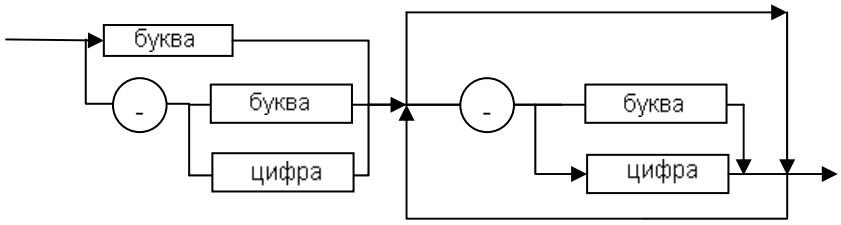


Рисунок 6.1. Схема построения идентификатора

Регистр символов в идентификаторе имеет значение.

Ключевые слова

Ключевые слова представлены в Приложении А.

Регистр символов в ключевых словах не имеет значение.

Комментарии

Комментарии к программе определяются обособлением необходимого текста фигурными скобками, то есть символами «{» и «}». Отдельного оператора, для определения однострочных комментариев, не существует.

Типы данных

Типы данных языка MCL состоят только из «простых» типов:

- Int – Целочисленное знаковое 32 битовое число
- Real – Вещественное знаковое 32 битовое число
- Bool –Булева переменная
- String - Строка символов
- Char – Символьный тип

Объявление переменных

Объявление переменных имеет следующий вид:

<тип данных> <идентификатор переменной > {, <идентификатор переменной>} ;

Где конструкция {…} определяет альтернативность данного выражения.

Операторы, операнды и выражения

Выражения состоят из операторов, операндов и круглых скобок. Цель выражения состоит в том, чтобы связать операторы, такие как -, +, / и т.д. с операндами (переменные, числа, строковые и символьные литералы), чтобы сформировать новое значение.

Выражения бывают двух типов:

- Арифметические
- Логические

Операторы:

- Оператор присвоения :=
- Арифметические операторы: -, +, /, *
- Операторы сравнения: <, <=, >, >=, <>, =
- Логические операторы: AND, OR, NOT

Результат арифметических операций имеет тот же тип данных, что и тип данных обоих операндов, исключении составляет операция арифметического деления /, которая возвращает результат типа REAL, даже если оба операнда имеют тип INT.

Управляющие инструкции

Язык MCL содержит 4 управляющих инструкций:

- IF…
- FOR…
- WHILE…
- GOTO…

Инструкции IF является условной инструкцией, так как с помощью нее осуществляется выбор из различных альтернатив в программе. К инструкциям цикла относятся инструкции FOR и WHILE, которые обеспечивают n-кратный повтор набора инструкций. Команда GOTO, являясь инструкцией безусловной передачи управления, прерывает последовательность выполнения и передает управление в другую точку программы.

Инструкция IF…

Имеет следующий синтаксис
IF <условие> THEN <инструкции>;
ELSEIF <условие> THEN <инструкции>;
…
ELSE <инструкции>;
END_IF
Команда IF выполнятся согласно следующим правилам:

1. Если значение первого выражения True, то выполняется секция команд, следующих после Then, иначе выполняются команды в ветви Elseif.
2. Если все выражения в ветвях Elseif равны False, то выполняются команды в ветви Else (или не выполняются вообще, если ветвь Else не представлена). Инструкция ELSEIF может присутствовать неоднократно, а может и вообще не использоваться.

Инструкция FOR…

Имеет следующий синтаксис

FOR < <управляющая переменная> := начальное значение для счетчика>
TO Выражение для конечного значения>
BY <Выражение для инкремента (шаг)>
DO <инструкции>;
END_FOR;
Инструкция FOR выполняется согласно следующим правилам:

1. В начале цикла управляющая переменная устанавливается на стартовое значение и после каждого цикла увеличивается (приращение >0) или уменьшается (приращение <0) на величину приращения, пока не будет достигнуто конечное значение.
2. После каждого цикла производится проверка, было ли выполнено условие окончания, которое заключается в том, чтобы значение управляющей переменной было строго больше конечного значения. Если это условие не выполнено, последовательность команд выполняется, иначе цикл заканчивается.

Объявление приращения может быть опущено, в этом случае оно будет равно +1

Инструкция WHILE…

Имеет следующий синтаксис

WHILE <Выражение> DO <Инструкции>;
END_WHILE
Инструкция WHILE выполняется согласно следующим правилам:

1. Условие выполнения проверяется перед каждым выполнением секции команд
2. Если значение TRUE, то секция команд выполняется
3. Если значение FALSE, то выполнение инструкции While заканчивается.

Синтаксис и назначение команд, представлено в Приложении Б.

7 Компилятор языка MCL

Компилятор языка MCL необходим для трансляции программы с языка MCL в промежуточный код. Промежуточный код передается в ПЛК, в котором находится интерпретатор промежуточного кода. Необходимо заметить, что реализованный промежуточный код в рамках данной работы является и промежуточным по своей реализации. Это значит, что хотя он и выполняет возложенные на него задачи, все же является не оптимальным по своей реализации. В дальнейшем его развитии объем результата трансляции (длина строки промежуточного кода) может быть уменьшен.

Компилятор написан в среде разработки Microsoft Visual Studio 2008 на языке C# и представляет собой динамически подключаемую библиотеку (DLL).

Компилятор логически состоит из двух частей: лексического и синтаксического анализаторов (семантический анализ выполняется на уровне синтаксического анализа).

7.1 Лексический анализ

Лексический анализатор (Scanner) сканирует входной поток символов, который представляет исходную программу и в соответствии с необходимыми правилами грамматики строит токены (token). Также происходит проверка, какие токены являются ключевыми словами. С помощью таблицы предшествований выполняется начальная проверка допустимости следования проверяемых токенов. Например, токен, определяющий ключевое слово INT (целочисленный тип данных), не может располагаться перед токеном, определяющим ключевое слово MOV (команда движения). То есть выполняется некоторая часть синтаксического анализа. Результатом лексического анализа является поток токенов, которые и передаются синтаксическому анализатору.

Синтаксический анализатор (Parsing) в данной реализации помимо своих задач решает параллельно задачи семантического анализа (проверка типов) и генератора кода (создание целевого кода, который является промежуточным).

7.2 Синтаксический анализ

На данном этапе происходит группирование токенов исходной программы в грамматические фразы. Это осуществляется предикативным разбором (анализом)[6]. Для каждого нетерминального символа существует процедура, которая реализует вывод конечной цепочки терминалов из нетерминала, реализуя необходимую продукцию (порождающая грамматика языка MCL, по которой происходит разбор, представлена в Приложении В, грамматика отображена в расширенной нотации Бэкуса-Наура [7, гл. 2]).То есть процедура использует продукцию, имитируя ее правую часть. Нетерминал приводит к вызову процедуры, соответствующей этому нетерминалу, а токен, совпадающий с текущим сканируемым символом (токеном из потока), к чтению следующего токена из входного потока. Если в какой-то момент токен продукции не совпадает со сканируемым токеном из потока, то производится запись об ошибке и, в общем случае, откат назад (возврат токенов в поток) на такое количество токенов, какое было прочитано в данной процедуре. Этот откат позволяет продолжить трансляцию кода с целью проверки его правильности, хотя, конечно, промежуточный код, как результат компиляции, уже не может быть получен. Совместно с этим происходит формирование выходной строки промежуточного кода. Так, если прочитанный токен является значимым (должен быть представлен в результате, например, аргумент команды, в то время как токен, обозначающий, например, запятую, значимым не является) то он, в общем случае, передается процедуре генерации промежуточного кода, которая помещает определитель этого токена в выходную строку.

7.3 Проверка типов (семантический анализ)

Проверка типов состоит из проверки типов выражения, которая является предпоследней стадией в трансляции выражения. Обработка (трансляция) выражения состоит из трех шагов:

- Перевод выражения в постфиксную форму. То есть выражения приводятся к польской инверсной записи. Также на этом этапе производится «разыменование» переменных и некоторых команд. Так, например, если для условия (CND) было определено имя, необходимо в выражение поставить вместо имени номер этого условия, то есть «подменить» необходимый токен новым.
- Проверка типов выражения. Проверка осуществляется «чтением» построенной постфиксной записи. Формирование типа выражения происходит по следующему общему правилу:

  $$E \rightarrow E1\ operation\ E2$$

  $$\{TYPE(E) := IF\ TYPE(E1) = type\ and\ TYPE(E2) = type\ THEN\ return\ type\ ELSE\ return\ type_{error}\}$$

  Где *operation* – операция (арифметическая, логическая), *E* – выражение или подвыражение (операнд), *TYPE* –функция возвращающая тип выражения или операнда.
- Если выражение имеет правильный тип (операнды в выражении указаны с правильными типами), то вызывается процедура, которая «читая» транслирует выражение в выходную строку. Под «чтением» понимается «раскрытие» польской инверсной строки. То есть выражение хоть и остается в постфиксной форме, но перестраивается таким образом, чтобы непосредственно в начало поставить части выражения, которые должны выполниться первыми. Например, для выражения $a*(b+c)$ постфиксная запись будет выглядеть как $abc+*$, а «раскрытая» постфиксная запись как $bc+a*$. Тем самым, при интерпретации (выполнении) этого выражения, интерпретатору понадобится стек вычислений гораздо меньшей глубины.

  Под трансляцией понимается перевод выражения согласно правилам построения промежуточного кода в выходную строку, то есть генерация промежуточного кода.

7.4 Генерация промежуточного кода.

Под промежуточным кодом (п.к.) понимается код, который представляет собой результат компиляции, и в общем случае может иметь различное представление, в зависимости от целевой системы исполнения. Так, в данном случае, промежуточный код представляет собой строку символов, которая передается выполняющему устройству, ПЛК, который с помощь интерпретатора читает и выполняет этот код. Данную реализацию нельзя считать оптимальной, так как ее доработка может дать более 50% уменьшения длины промежуточного кода. В то же время данная реализация, дает промежуточный код, который по объему, в общем случае, равен около 80-110% от объема исходного кода. Что при небольших программах допустимо. Больший коэффициент «сжатия» объема промежуточного кода по отношению к исходному достигается, когда в исходной программе большая часть переменных имеют длину имени больше 3 символов, и используется, по возможности, минимальное количество вещественных чисел. Это следует из того что имена переменных заменяются их трехзначным номером, а вещественные числа переводятся в формат, понятный для функции языка SCL (SIMATIC STEP 7),которая переводит строку символов в вещественное число типа REAL. Такой формат требует, что бы вещественное число было представлено строкой вида:

$$\pm v.mmmmmE \pm xx$$

Что дает 14 символов для каждого вещественного числа в выходной строке.

Данная реализация, как видно, не оптимальна, но позволяет сделать реализацию интерпретатора более простой, а его работу более наглядной, так как предполагается дальнейшая его доработка. Так, доработав интерпретатор до промышленного использования, можно решать вопрос с оптимизацией (возможно и с изменением) промежуточного кода.

Промежуточный код

Промежуточный код представляет собой строку символов. То есть каждая инструкция (команда) исходной программы заменяется на соответствующее обозначение из одного или нескольких символов в целевом коде. Такие символы будем называть командами. Список команд и управляющих символов, а также другие символы представлены в Приложении Г. Выбор в строну представления промежуточного кода, как строки символов, был определен тем что, в данном случае, промежуточный код передается в ПЛК через SCADA систему WINCC, используя строковые теги. Такое символьное представление команд дает возможность интерпретатору непосредственно реагировать на эти команды, без предварительной их обработки, что делает работу интерпретатора более «прозрачной» и дает возможность в режиме отладки непосредственно следить за процессом чтения входной строки символов (промежуточного кода).

7.5 Структура приложения

Приложение состоит из следующих классов:

- **CompileManager**. Реализует логику работы компилятора и предоставляет набор методов, с которыми должно работать внешнее приложение, чтобы воспользоваться сервисом компилятора. Таким образом, набор методов этого класса является программным интерфейсом для внешних приложений.
- **Scanner**. Лексический анализатор. Сканирует переданную ему входную строку символов и возвращает очередь токенов.
- **Parser**. Синтаксический и Семантический анализаторы. Разбирает переданную ему очередь токенов, возвращает, если трансляция прошла успешно, результат компиляции, выходной поток символов (промежуточный код). Также возвращает лог-информацию трансляции, в которой отображаются ошибки трансляции или другие возможные оповещения.
- **Logger**. Журнал трансляции. Реализует сбор информации о ходе компиляции.
- **TokenTable.** Реализует очередь токенов. Так же содержит все константы.
- **Token.** Реализует токен.
- **SemanticStructure.** Реализует семантическую структуру.

На рисунке 7.1 изображена диаграмма классов для приложения компилятора.

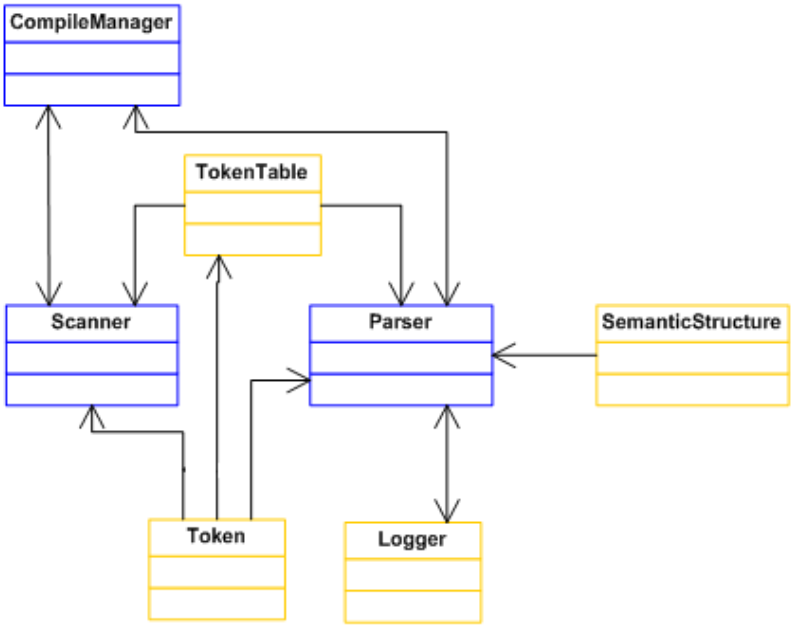


Рисунок 7.1. Диаграмма классов

8 Интерпретатор

Интерпретатор предназначен для выполнения на целевой машине и чтения промежуточного кода, предоставленного компилятором. Если говорить образно, то задача интерпретатора, это читать входную строку символов и параллельно «объяснять» смысл прочитанного движку, который представляет собой программную часть исполняющего устройства (исполняющим она является в паре «Система управления - ПЛК»).

При этом интерпретатор является частью программного обеспечения ПЛК (движка). Так в рамках данной работы интерпретатор разрабатывался для выполнения на программируемых логических контроллерах SIMATIC S7-300/400. Для этого он разрабатывался в системе SIMATIC STEP 7, на языке SCL,и представляет собой набор пользовательских блоков (FB, FC, DB), который может быть помещен в другой проект.

8.1 Основная логика

В основу реализации интерпретатора положен механизм конечного автомата и его переходов из одного состояния в другое посредством чтения входных символов. Таким образом, интерпретатор избавляется от излишней «интеллектуальности». Ему не нужно «думать» что делать, ему надо «знать» как делать, как реагировать на новый прочтенный символ.

В выделенную область памяти загружается посредством тегов и функции загрузки входная строка символов (промежуточный код). Такую строку назовем Кодовой строкой. Эта строка располагается в отдельном блоке данных. В этом же блоке находится переменная, хранящая длину кодовой строки. Максимальный объем блока данных зависит от CPU ПЛК, так в данном случае это 64Kb. Так как символ занимает 1 байт, то кодовая строка может иметь длину не больше 65524 символов. Что для большинства алгоритмов можно считать достаточным. Затем эта строка подвергается непосредственно интерпретации – интерпретатор читает строку и когда встречает команду, вызывает процедуру соответствующей этой команде. Далее эта процедура вычитывает необходимые ей параметры, выполняет заложенные в ней действия (установка флагов, вызов методов движка и т.д.) и возвращает управление главному циклу сканирования. Чтение строки может на некоторое время приостанавливаться. Например, после команды MOVN. В таком случае запоминается позиция, с которой необходимо будет продолжить чтение строки и происходит «пропуск» процедуры главного цикла сканирования.

8.2 Работа с переменными

Отдельный блок данных выделен для переменных, которые занимают:

- INT – 4 байта
- REAL – 4 байта
- BOOL – 1 байт
- STRING – 50 байт
- CHAR – 1 байт
- POINT – 8 байт

Переменные определяются, записываются и читаются в этой области данных посредством специальных функций.

Механизм определения переменных. После прочтения управляющим циклом команды объявления переменной, вызывается функция *DeclarVar,* которой передается тип объявляемой переменной. На основе переданного типа и согласно таблице 8.1 происходит резервирование места в области памяти (блок данных *GlobalVariables*) для созданной переменной. То есть в массив указателей *Pointers* записывается новый указатель, который содержит сам указатель на начало области памяти для новой переменой, ее тип, размер в

байтах (см рис. 8.1). Новой переменной “присваивается” ее порядковый номер, причем такой номер «синхронизирован» с номером этой переменной из кодовой строки. Это значит, что хотя в исходном коде обращение к переменным происходит через их имена, в промежуточном коде имена заменены на трехзначные числа (мах 999), представляющее номер переменной. Такой номер «вычитывает» из кодовой строки интерпретатор и непосредственно к переменной обращается по этому номеру. Таким образом, интерпретатору не нужно тратить ресурсы на имена переменных и поиск переменной по ее имени.

Таблица – 8.1. Типы данных, их размер и обозначения

символ	тип	размер/смещение (байт)	тип (обозначение)
l	Local variable	4	-
i	INT	4	1
r	REAL	4	2
s	STRING	50	4
c	CHAR	1	5
b	BOOL	1	3

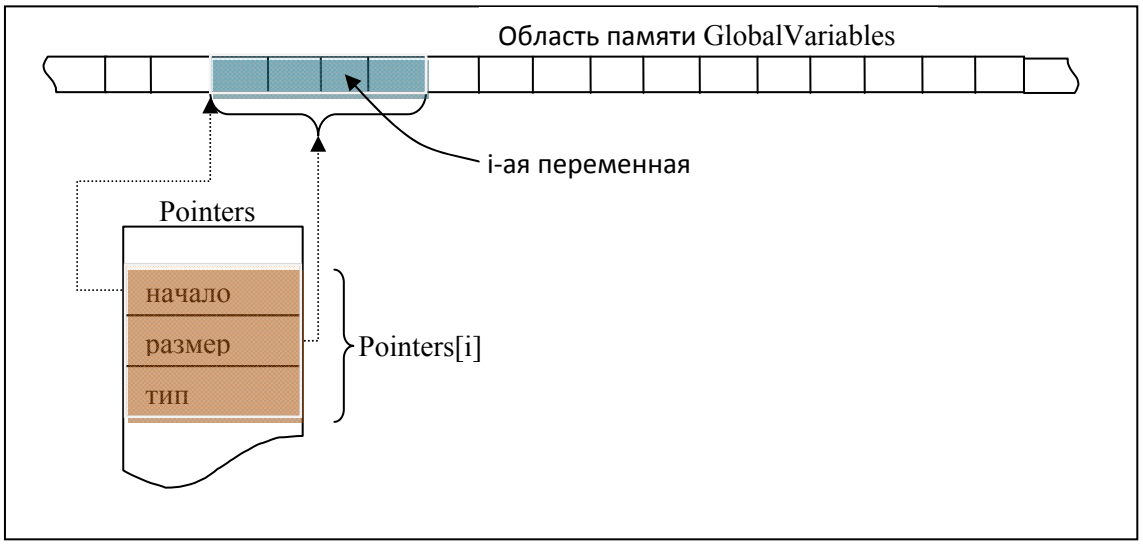


Рисунок 8.1. Механизм определения переменных

Механизм чтения (записи) значения переменной. После того как была прочитана команда чтения (присвоения) переменой, из кодовой строки читается номер переменной и этот номер передается функции чтения (записи) переменной. Функция *ReadVar* получает необходимую информацию («начало» переменной, ее тип, занимаемое количество байт) о требуемой переменной из массива *Pointers* и производит ее побайтовое чтение в специальную переменную *TVDWORD* которая имеет тип данных DWORD (4 байта). Если читается строка, то она читается в переменную *TVString2* (50 байт). Похожий механизм присвоения, только перед вызовом функции записи, необходимо присвоить нужное значение переменным *TVDWORD* или *TVString2*.

8.3 Реализация циклов FOR… и WHILE…

Реализация этих циклов конечно разная, но подход одинаковый и заключается в решении следующего нюанса. Как было уже сказано в главе «Программирование ПЛК», программа в ПЛК выполняется циклически и время на цикл ограничено. Это значит, что пользовательская программа, выполнившись, передает управление ОС ПЛК, которая выполняет необходимые действия, например, обновляет образ процесса и заново запускает пользовательскую программу. При этом данные пользовательской программы, находящиеся в блоках данных, не теряются. Из-за ограниченности времени цикла цикл WHILE не может выполняться сколь угодно долго в рамках одного цикла. Поэтому реализован следующий механизм выполнения циклов WHILE… (FOR…). Если интерпретатор прочел команду цикла, то обрабатывается выражение условия выполнения этого цикла. Если это условие оказывается истинным, что говорит о необходимости выполнения вложенных инструкций, то эти инструкции выполняются один раз, после чего происходит завершение пользовательской программы в ПЛК, а после возврата в нее снова происходит обработка выражения условия данного цикла и процесс повторяется. Если же условие оказалось ложным, вложенные инструкции пропускаются, и чтение кодовой строки продолжается. Такое решение оказывается не совсем оптимальным, когда вложенные инструкции требуют небольшого времени исполнения, по сравнению со временем выполнения цикла программы. Зато дает гарантию, что время выполнения программы не превысит максимального времени и CPU не перейдет в режим Stop.

Заключение

В рамках данной работы были выполнены следующие задачи:

1. Разработан язык управления манипуляторами (MCL). Дальнейшая доработка языка может предполагать добавление необходимых команд и инструкций, необходимость в которых может возникнуть в ходе дальнейшего использования языка.
2. Разработан компилятор для трансляции кода на языке MCL в промежуточный код.
3. Разработан интерпретатор промежуточного кода. Доработка интерпретатора предполагает изменение его для чтения более оптимального промежуточного кода.

Компилятор и интерпретатор готовы для работы в тестовом режиме. То есть они готовы к работе на реальном оборудовании, но требуют тщательного тестирования и доработки прежде, чем они смогут быть допущены для работы в реальных промышленных условиях.

Список использованной литературы

1. Wikipedia //робот. – 2010. Режим доступа: http://ru.wikipedia.org/wiki/Робот, свободный
2. Wikipedia // Манипулятор_(механизм). – 2010. Режим доступа: http://ru.wikipedia.org/wiki/Манипулятор_(механизм), свободный
3. Левитский Н.И. Теория механизмов и машин. – М. «Наука», 1990. – 592с.
4. Программирование с помощью STEP 7 v 5.3. Siemens AG, – 2004. –602с.
5. Г. Бергер. Автоматизация посредством STEP 7 с использованием STL и SCL и программируемых контроллеров SIMATIC S7-300/400. Siemens AG, 2001. – 776с.
6. Ахо А., Ульман Д., Сети Р. Компиляторы. Принципы, технологии, инструменты. – М. «Вильямс», - 2001. – 769с.
7. Карпов Ю.Г. Основы построения трансляторов. – СП. «БХВ – Петербург», 2005. - 273с.

Приложение А

Обзор ключевых слов языка MCL

ACCEL
AND
BOOL
BY
CHAR
CLR
CND
DECEL
DEFINE
DELAY
DO
ELSE
ELSEIF
END_FOR
END_IF
END_WHILE
FOR
GOTO
IF
INT
LABEL
MOV
MOVN
NOT
OR
PARAM
REAL
SET
SETAXISNAME
SPEED
STOP
STRING
THEN
TO
TORQUE
WHILE

Приложение Б

Команды и инструкции языка MCL

Типы данных:

Int – целочисленный тип
Real – вещественный тип
Bool – логический тип
Char – символьный тип
String – строковый тип

Команды:

MOV N1 P1 T1 G1, N2 P2 T2 G2,… осуществляет движение оси(осей) к указанной позиции с указанным типом позиционирования и окрестностью. После начала выполнения команды, интерпретатор переходит к следующей команде. Если в одной команде было перечислено несколько осей, то осуществляется совмещенное движение.
Аргументы:

- N – номер оси. Номер может быть указан непосредственно числом, либо переменной, установленной командой SetAxisName. Номер указывается в ограничениях от 0 до 99.
- P – позиция для позиционирования. Указывается либо числом, либо переменной типа INT
- T- тип Позиционирования. **P** – обычное позиционирование, **C** – позиционирование в проходную точку, **E** – позиционирование к жесткому упору, **B** – признак позиционирования с базой
- G – окрестность.

*Пример: **mov** Telega1 100 **p** 10, 3 Pos1 **p** 32;*

MOVN – команда аналогична команде ***MOV***, различается только тем, что выполнение следующих команд приостанавливается, пока все оси, перечисленные в команде ***MOVN***, не достигнут своей точки назначения.

PARAM N1 Arg1_1 Val1_1 Arg1_2 Val1_2…Arg1_4 Val1_4, N2 Arg2_1 Val2_1… Задание параметров для оси.
Аргументы:

- N- номер оси. Указывается либо числом, либо переменной, заданной командой ***SetAxisName***.
- Arg1-Arg4 – **S**- скорость, **A** – ускорение, **D** – замедление, **T**- момент.
- Val – значение задаваемого параметра. Указывается либо числом, либо переменной типа INT

*Пример: **param** 0 **s** 100 **d** 200, Axis1 **t** 320;*

SETAXISNAME *Name Number* – Устанавливает для оси с номером *Number* имя *Name*.
Параметр *Number* задается числом типа INT
*Пример: **SetAxisName** Axis1 31;*
Замечание: В одной команде *SetAxisName* устанавливается только одно имя только для одной оси.

DELAY P V – задержка.

Параметры

- P – Определяет единицу измерения времени задержки: **ml** (миллисекунды), **s** (секунды), **m** (минуты), **h** (часы)
- V – Значение задержки. Задается числом или переменной типа INT

STOP Axis – Остановить Ось Axis, где Axis это имя оси, заданное командой SetAxisName, либо число INT (не переменная)

DEFINE CondName1 CondNumber1, … CondNameN CondNumberN – Задать для Условия с номером *CondNumber* имя *CondName*. *CondNumber* может быть заданно только числом типа INT. В дальнейшем, к указанному условию можно обращаться по имени
*Пример: **Define** LoadState 0, OpenDoor 1;*

SET/CLR *Cond* – Установка/Сброс условия *Cond*. Условие *Cond* задается переменной, определенной командой ***DEFINE***, либо числом типа INT
*Пример: **Set** LoadState; **Clr** 3;*

CND CondNumber – чтение состояния условия с номером *CondNumber*. *CondNumber* задается числом типа INT.
*Пример: **bool** State; State := **CND** 31;*

LABEL Name – определение метки
GOTO labelName –безусловная передача управления. LabelName – имя ранее определенной метки
*Пример: **Label** Result1; **Goto** Result1;*

ACCEL Axis; **DECEL** Axis; **SPEED** Axis; **TORQUE** Axis; - Чтение у оси Axis параметров: Ускорение, замедление, скорость, момент на валу соответственно.

Примечание: Команды LABEL, GOTO, ACCEL, DECEL, SPEED, TORQUE не реализованы в данной версии компилятора

Управляющие инструкции

Инструкция FOR…
For i:=a **to** b **by** s **do**
Последовательность инструкций
End_for;
Аргументы:

- A – начальное значение счетчика цикла
- B – конечное значение счетчика цикла
- S – величина, на которую изменяется счетчик цикла на каждом шаге цикла

Инструкция WHILE…
While Expression **do**
Последовательность инструкций
End_while;
Аргументы:

4. Expression – Выражение которое вычисляется, перед тем как цикл может быть выполнен. Если выражение имеет значение Истина (True), то цикл выполняется.

Замечание: Expression не может содержать операцию присваивания или состоять из одиночного операнда.

Инструкция IF…

If Expr **then**
Последовательность инструкций
Elseif Expr **then**
Последовательность инструкций
…
Elseif Expr **then**
Последовательность инструкций
Else
Последовательность инструкций
End_if

Если какое-либо из выражений Expr выполняется, то выполняется последовательность инструкций в соответствующей ветви If или Elseif иначе выполняется ветвь Else. Ветви Elseif могут появляться в инструкции более одного раза или их может не быть вообще. Else может так же не использоваться.

Приложение В

Грамматика языка MCL

<Letter> ::= a | …| z | A | …| Z

<Digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<Id> ::= <Letter> | _ { <Letter> | <Digit> | _ }

<Literal > ::= “ {<Letter> | <Digit>| <another symbol>} ”

<Character> ::= ‘ [<Letter> | <Digit> | <another symbol>]’

<Integer> ::= [+|-] <Digit> {<Digit>}

<Real> ::= [+|-] <Digit> {<Digit>} . <Digit> {<Digit>}

<Bool Value> ::= true | false

<Bool Operation> ::= or | and | not

<Math Operation> ::= + | - | * | /

<Relation> ::= < | <= | = | <> | > | >=

<Comments> ::= ‘**{**’<Any Symbol> {<Any Symbol>} ‘**}**’

<Any Symbol> ::= $x, x \in ASCII$

<Date type> := Int | Real | String | Char | Bool

<Variable Definition> ::= <Date type > <Id> {, <Id> };

S: <Statement Sequence> ::= <Statement> {; <Statement>} ;

<Statement> ::= <ForStatement> | <WhileStatement> | <IfStatement> | <Assignment> | <Commands> | < Variable Definition > | <Comments>

<ForStatement> ::= for <Id> := <Id> | <Integer> to <Id> | <Integer> [by<Id> | <Integer>] do <Statement Sequence> end_for

<WhileStatement> ::= while <Condition> do < Statement Sequence> end_while

<IfStatement> ::= if <Condition> then < Statement Sequence > {elseif <Condition> then < Statement Sequence >} [else < Statement Sequence >] end_if

<Assignment> ::= <Id> := <Expression> | <Condition> | <Character> | <Literal> ;

<Condition> ::= <Expression> <Relation> <Expression> | <BoolValue> | not <Condition> | < Expression > <Bool Operation> < Expression >

<Expression> ::= <HPExpr> <SPExpr>

<SPExpr>::= + <HPExpr> <SPExpr> | - <HPExpr> <SPExpr> | ε

<HPExpr>::= < ExprOperand > <SubHPExpr>

<SubHPExpr>::=* < ExprOperand > <SubHPExpr> | / < ExprOperand > <SubHPExpr> | ε

<ExprOperand>::= (<Expression>) | <Id> | < Integer > | <Real>

<Commands> ::= <Mov statement> | <Param statement> | <Delay statement > | <Define statement > | <SetAxisName statement > | <Stop statement > | <Set_Clr statement > | <ADST statement > | <CND statement > | <Delay statement > | <GOTO statement > | <Label statement >

<argument> ::= <id> | <Integer>

<Mov statement> ::= mov | movn <argument> <argument> P | C | E | B <argument> {, <argument> <argument> P | C | E | B <argument>};

<Param statement> ::= param <argument> A | D | S | T <argument> { A | D | S | T <argument>} {,<argument> A | D | S | T <argument> { A | D | S | T <argument>} };

<SetAxisName statement > ::= <Id> <Integer>;

<Delay statement> ::=delay ml | s | m | h <argument>;

<Stop statement> ::= stop <argument> {, <argument>} ;

<Define statement> ::= define <Id> <Integer> {, <Id> <Integer> };

<Set_Clr statement> ::= set | clr <argument> {, <argument>} ;

<ADST statement> ::= accel | decel | speed | torque <argument> ;

<CND statement> ::= cnd <Integer>;

<Goto statement> ::= goto <Id>;

<Label statement> ::= label <Id>;

Приложение Г

Команды и управляющие символы Промежуточного языка

Управляющие символы

- команда

!- завершает число

%-завершает выражение

Определяющие символы (определяют, что будет читаться далее или определяют какой-то аргумент, операцию)

v – указывает, что читается переменная

i – определяемый тип INT (в команде #a – далее читается число int)

r – определяемый тип REAL (в команде #a – далее читается число real)

c - определяемый тип CHAR (в команде #a – далее читается символ)

b - определяемый тип BOOL (в команде #a – далее читается булева переменная)

s - определяемый тип STRING (в команде #a – далее читается символьная строка)

l – миллисекунды (команда #D) (в команде #f – объявить локальную переменную)

C – указывает, что читается состояние условия

W-определяет команду MOV

Q-определяет команду MOVN

s – секунды (команда #D)

m – минуты (команда #D)

h – часы(команда #D)

n – операция NOT

& - операция AND

| - операция OR

R - операция >=

E - операция <=

T - операция <>

Команды

#a – присвоение (:=)

#c – CLR

#D – DELAY

#d – объявить переменную

#e - ELSE

#F – END_FOR

#f – FOR…

#i – IF…

#k – END_IF

#l – ElSEIF

#m – MOVE

#p – PARAM

#s – SET

#S – STOP

#W – END_WHILE

#w – WHILE

Пример:

Кода на языке MCL:

```
int AxSpeed;
bool Error;
if AxSpeed > 100 then Error:=true;
else
     Error:=false;
     Mov 0 100 p 10;
End_if;
```

Промежуточный код:

```
#di#db#iv000i100!>%#a001b1%#e#a001b0%#m400i100!pi10!%W#k
```