ЧАСТЬ 1. АРХИТЕКТУРА ПРОЦЕССОРОВ INTEL x86

Практическое занятие № 1.1
ЯЗЫК АССЕМБЛЕРА. ОБЗОР АССЕМБЛЕРОВ

1. ЦЕЛИ РАБОТЫ

- Знакомство с понятием языка ассемблера и его возможностями.
- Обзор ассемблеров.

2. ТЕОРЕТИЧЕСКИЙ БЛОК

Понятие ассемблера

Ассемблер *(от англ. Assembler – сборщик) – компилятор исходного текста программы, написанной на языке ассемблера, в программу на машинном языке.*

Как и сам язык, ассемблеры, как правило, специфичны для конкретной архитектуры, операционной системы и варианта синтаксиса языка. Существуют также мультиплатформенные или ограниченно-универсальные ассемблеры. Среди последних можно также выделить группу кросс-ассемблеров, способных собирать машинный код и исполняемые модули (файлы) для других архитектур и операционных систем. Часто ассемблером называют не сам компилятор, а язык программирования.

Ассемблер – язык программирования низкого уровня, представляющий собой формат записи машинных команд, удобный для восприятия человеком. Команды языка соответствуют командам процессора и представляют собой удобную символьную форму записи (мнемокод) команд и их аргументов.

Помимо команд в ассемблере определены собственные директивы.

Директивы *– параметры (ключевые слова) в тексте программы на языке ассемблера, влияющие на процесс ассемблирования или свойства выходного файла.*

Директивы позволяют включать в программу блоки данных (описанные явно или считанные из файла); повторять определённый фрагмент указанное число раз; компилировать фрагмент по условию; задавать адрес исполнения фрагмента, менять значения меток в процессе компиляции; использовать макроопределения с параметрами и

другое, т. е. в целом упрощают работу программиста и улучшают читабельность кода. Как правило, каждая модель процессора имеет свой набор команд и соответствующий ему язык (диалект) ассемблера.

Достоинства ассемблера

- Доступ к регистрам процессора. Регистры представляют собой особые участки памяти процессора, превосходящие по скорости доступа оперативную память.
- Минимальная избыточность кода (использование меньшего количества команд и обращений в память). Как следствие – бóльшая скорость и меньший размер программы.
- Возможность метапрограммирования.
- Достижение максимальной совместимости для требуемой платформы.
- Непосредственный доступ к аппаратуре: портам ввода-вывода, особым регистрам процессора.

Недостатки ассемблера

- Зачастую большой объем кода (особенно для сложных и оконных приложений).
- Плохая читабельность кода, трудность поддержки (отладка, добавление возможностей).
- Трудность реализации различных парадигм программирования, сложность совместной разработки.
- Малое количество доступных библиотек.
- Ассемблер как таковой некросплатформенен.

В связи с вышесказанным можно сделать вывод, что ассемблер, как правило, имеет преимущество при решении узкоспециализированных задач, требующих жесткой экономии ресурса, в то время как языки высокого уровня удобны для больших проектов (задача предельной оптимизации не стоит).

Область применения и роль изучения

- На ассемблере пишутся драйверы и ядра операционных систем.
- Его используют для «прошивки» BIOS.
- На ассемблере пишут компиляторы и интерпретаторы языков высокого уровня.
- Он эффективен для написания вирусов и используется при взломе программ.

Изучение ассемблера полезно как начинающему, так и опытному программисту.

- Ассемблер способствует развитию навыков эффективного программирования, изучению приемов оптимизации и отладки кода.
- Позволяет лучше понять устройство процессора и архитектуры ПК в целом.
- Это язык процессоров и микроконтроллеров.
- Программирование на ассемблере формирует особую культуру мышления программиста.

Рекомендации

Изучение любого нового языка программирования требует сосредоточенности, терпения и постоянной практики. Ассемблер – не исключение.

- Эффективное изучение требует длительной подготовки, в частности, чтения дополнительной литературы.
- Ассемблер – низкоуровневый язык, требующий от программиста постоянного контроля действий.
- Отладчик – верный друг программиста на ассемблере. Он позволяет отслеживать ошибки, которые не всегда очевидны.
- Постоянная практика. Сложные на первый взгляд манипуляции со временем переходят в опыт и выполняются автоматически.

Обзор ассемблеров

Наличие различных ассемблерных языков говорит об отсутствии единого универсального ассемблера, способного одинаково эффективно решать различные задачи.

Выделим наиболее распространенные ассемблерные языки, предназначенные для различных процессоров и микроконтроллеров.

Turbo Assembler (TASM) – продукт компании Borland, предназначенный для разработки программ на языке ассемблера для архитектуры x86. Кроме того, TASM способен работать совместно с трансляторами с языков высокого уровня фирмы Borland, такими как Turbo C и Turbo Pascal.

TASM до сих пор используется для обучения программированию на ассемблере под MS-DOS. Многие находят его очень удобным и продолжают использовать, расширяя набором дополнительных макросов.

TASM способен работать в режиме совместимости с Macro Asembler (MASM). Кроме того, TASM имеет собственный режим IDEAL, улучшающий синтаксис языка и расширяющий его функциональные возможности.

Как и прочие программные пакеты серии Turbo, TASM официально больше не поддерживается, поэтому считается «умирающим» языком. Однако последние версии (TASM 5.0) позволяют писать 32-битные приложения.

Macro Assembler (MASM) – ассемблер для процессоров семейства x86. Первоначально был произведён компанией Microsoft для написания программ в операционной системе MS-DOS и был в течение некоторого времени самым популярным ассемблером, доступным для неё. Это поддерживало широкое разнообразие макросов и структурированность программных идиом, включая конструкции высокого уровня для повторов, вызовов процедур и чередований (поэтому MASM – ассемблер высокого уровня). Позднее была добавлена возможность написания программ для Windows.

В связи с широкой распространённостью MASM посвящено большое количество учебной литературы (в том числе и русскоязычной) и готовых библиотек. Это позволяет называть MASM самым популярным ассемблером из доступных.

Flat assembler (FASM) – свободно распространяемый многопроходной ассемблер, написанный Томашем Грыштаром (польск. Tomasz Grysztar). Fasm написан на самом себе, обладает небольшими размерами и высокой скоростью компиляции, мощным макросинтаксисом, позволяющим автоматизировать множество рутинных задач. Поддерживаются как объектные форматы, так и форматы исполняемых файлов.

FASM – многоплатформенный ассемблер: на нем можно программировать как в Windows, так и в Unix-системах.

NASM (Netwide Assembler) – свободный ассемблер для архитектуры Intel x86. Используется для написания 16-, 32- и 64-битных программ.

NASM компилирует программы под различные операционные системы в пределах x86-совместимых процессоров. Находясь в одной операционной системе, можно беспрепятственно откомпилировать исполняемый файл для другой.

YASM – название имеет несколько возможных истолкований: Yes, it's an ASseMbler, Yet Another aSseMbler, Your fAvorite aSseMbler и др. YASM – попытка полностью переписать ассемблер NASM. В настоящее время развивается Питером Джонсоном и Майклом Ерманом.

YASM предлагает поддержку x86-64. Кроме Intel-синтаксиса, применяемого в NASM, YASM также поддерживает AT&T-синтаксис, распространённый в Unix. YASM построен «модульно», что позволяет легко добавлять новые формы синтаксиса, препроцессоры и т. п.

GNU Assembler, или **GAS** – ассемблер проекта GNU; используется компилятором GCC. Входит в пакет GNU Binutils. Кроссплатформенная программа запускается и компилирует код для многочисленных процессорных архитектур. Распространяется на условиях свободной лицензии GPL 3.

GAS был разработан для поддержки компиляторов Unix, он использует стандартный синтаксис AT&T, который несколько отличается от большинства ассемблеров, основанных на синтаксисе Intel.

RosAsm – 32-битовый Win32 x86 ассемблер. Согласно своему имени, ассемблер поддерживает операционную систему ReactOS, хотя проекты RosAsm и ReactOS независимы. RosAsm – среда разработки с полной интеграцией ассемблера, встроенного линкера, редактора ресурсов, отладчика и дизассемблера. Синтаксис сделан как продолжение NASM'а.

3. ВОПРОСЫ ДЛЯ САМОПРОВЕРКИ

1. Что называют ассемблером?
2. Чем команды ассемблера отличаются от директив?
3. Каковы основные преимущества и недостатки ассемблерного кода?
4. Чем объясняется отсутствие единого ассемблерного языка?
5. Чем полезно изучение ассемблера начинающему и опытному программисту?