1 вариант

1. **Основные функции первого прохода в 2-х проходном ассемблере:**
   - **В первом проходе ассемблер выполняет следующие функции:**
     - **Построение таблицы меток (ярлыков) и определение адресов меток. Это необходимо для правильной обработки всех ссылок на метки в программе, поскольку вторая фаза будет иметь доступ к этим меткам.**
     - **Сборка информации о всех переменных и их размещении в памяти, включая определение адресов.**
     - **Преобразование исходного кода в промежуточный формат, в который включены только метки и адреса.**
     - **Первый проход не выполняет полную генерацию кода, так как это требует точных данных о размерах программных объектов и метках, которые можно определить только после первого прохода.**
2. **Структура перемещаемого объектного модуля:**
   - **Перемещаемый объектный модуль состоит из:**
     - **Машинных команд: непосредственно код программы.**
     - **Данных: включая константы, которые используются в программе.**
     - **Информации о настраиваемых элементах: таблица настраиваемых элементов (например, адреса), которая необходима для настройки объекта при его загрузке в память.**
     - **Таблицы символов и меток: включает информацию о метках, переменных и других элементах программы, требующих перенастройки. Эти части обеспечивают гибкость загрузки модуля в любую память, так как необходимая настройка выполняется с помощью информации о настраиваемых элементах.**
3. **Примеры постоянных таблиц ассемблера:**
   - **Таблица символов: хранит информацию о всех символах (например, метках и идентификаторах). Строка таблицы может включать поля для имени символа, типа (метка или переменная), адреса и других атрибутов.**
   - **Таблица операций: хранит операции и их коды для перевода команд в машинный код. Строка таблицы может содержать поля для кода операции, ее мнемоники и других характеристик.**
4. **Макроопределение в макроассемблере:**
   - **Макроопределение — это набор инструкций, который можно многократно использовать в программе. Он задает шаблон, который будет подставляться в код во время компиляции. Основное назначение макроопределений — автоматизация повторяющихся операций, упрощение и ускорение разработки программы.**
5. **Назначение и функции лексического анализа в компиляторе:**
   - **Лексический анализ отвечает за разбиение исходного кода на лексемы (ключевые слова, идентификаторы, операторы и т. д.). Этот процесс включает в себя классификацию элементов и упрощение представления программы, чтобы дальнейшие фазы компиляции могли эффективно работать с уже разделенными и нормализованными лексемами.**
6. **Транслирующая грамматика:**
   - **Транслирующая грамматика — это грамматика, которая используется для описания языков программирования, где элементы языка преобразуются в промежуточный или машинный код через трансляцию. Такие грамматики включают правила, по которым программа может быть переведена в код.**
7. **Атрибутная транслирующая грамматика:**
   - **Атрибутная транслирующая грамматика расширяет обычную транслирующую грамматику, добавляя атрибуты, которые могут быть ассоциированы с элементами грамматики. Эти атрибуты используются для вычисления значений, таких как типы данных или адреса в процессе трансляции.**
8. **Таблица отношений предшествования в грамматиках предшествования:**
   - **Таблица отношений предшествования отображает отношения между операторами в грамматике. В таблице определяется, какой оператор предшествует другому в выражении. Таблица должна быть однозначной и содержать информацию о том, какие операторы могут идти друг за другом. Условия для реализации грамматического разбора снизу-вверх: таблица должна однозначно определять отношения предшествования для каждой пары символов.**
9. **Типовые способы машинно-независимой оптимизации:**
   - **Приведение общих подвыражений: замена повторяющихся подвыражений на одно вычисленное ранее.**
   - **Вычисление литеральных подвыражений на стадии компиляции: предварительное вычисление значений выражений с постоянными литералами.**
   - **Оптимизация логических выражений: упрощение логических выражений для уменьшения вычислительных затрат.**
   - **Оптимизация циклов: преобразование циклов для уменьшения их времени выполнения.**
10. **Отличия автоматных и контекстно-независимых грамматик по представлению в синтаксических диаграммах:**
    - **Автоматные грамматики: представление линейное, без рекурсивных вызовов. Синтаксические диаграммы для автоматных грамматик развиваются по одному пути, без возвратов.**
    - **Контекстно-независимые грамматики: диаграммы включают рекурсивные вызовы, что позволяет обрабатывать более сложные структуры, например, вложенные выражения.**

2 вариант

---

1. **Типы команд, обрабатываемых ассемблером:**
   - **Команды данных: определяют размещение данных в памяти.**
   - **Команды инструкций: задают машинные команды, которые будут выполняться процессором.**
   - **Команды директивы: инструкции для ассемблера, управляющие процессом трансляции, но не преобразуемые в машинные коды.**
   - **Команды переходов: определяют метки и точки переходов в программе.**

---

2. **Основные функции второго прохода в 2-х проходном ассемблере:**

- **Замена меток и символов на их адреса: с использованием таблицы символов, построенной на первом проходе.**
- **Генерация машинного кода: перевод инструкций в бинарный код, который может быть исполнен процессором.**
- **Эти функции необходимы, так как первый проход собирает информацию, а второй непосредственно выполняет трансляцию, используя уже известные адреса и зависимости.**

---

3. **Примеры динамических таблиц ассемблера:**
   - **Таблица символов (меток): включает имя символа, его адрес и тип (метка или переменная).**
   - **Таблица адресов загрузки: содержит адреса, где будут размещаться инструкции и данные.**
   - **Таблица настраиваемых элементов: содержит поля для адресов, требующих настройки, и типов элементов (например, относительных или абсолютных).**

---

4. **Критерии сравнения одно-, двух- и трехпроходных ассемблеров:**
   - **Скорость работы: однопроходный быстрее, так как трансляция выполняется за один проход.**
   - **Объем памяти: двух- и трехпроходные требуют больше памяти для хранения промежуточных таблиц.**
   - **Гибкость: двух- и трехпроходные поддерживают более сложные конструкции, например, ссылки на метки, определяемые ниже в коде.**

---

5. **Определение обратной польской записи (ОПЗ):**
   - **ОПЗ — это форма записи выражений, в которой операции следуют за операндами.**
   - **В интерпретаторах используется для упрощения вычислений выражений, поскольку позволяет избегать скобок и легко обрабатывать выражения стековыми структурами.**

---

6. **Суть синтаксического и семантического анализа:**
   - **Синтаксический анализ: проверяет корректность структуры входного кода на основе грамматических правил.**
   - **Семантический анализ: проверяет смысловую корректность кода (например, типы данных и допустимость операций).**

---

7. **Определение эквивалентных грамматик:**
   - **Две грамматики называются эквивалентными, если они задают один и тот же язык, то есть генерируют одинаковое множество допустимых предложений.**

---

8. **Распознающий агрегат для контекстно-свободных языков:**
   - **Для контекстно-свободных языков требуется конечный автомат со стековой (магазинной) памятью, который может обрабатывать вложенные структуры.**

---

9. **Управляющие структуры для лексического анализа:**
   - **Необходимы следующие структуры:**
     - **Таблица лексем: для хранения идентификаторов, ключевых слов и других элементов.**
     - **Диаграмма переходов состояний: для реализации конечного автомата, распознающего лексемы.**

---

10. **Грамматический разбор снизу-вверх и порядок распознавания правил:**
    - **Разбор снизу-вверх начинается с распознавания терминальных символов и постепенно строит синтаксическое дерево, двигаясь к корню (начальному символу грамматики).**
    - **Порядок распознавания правил соответствует их использованию в правом выводе, но в обратной последовательности (правый вывод в обратном порядке).**

---

3 вариант

---

1. **Определение абсолютной и относительной адресации:**
   - **Абсолютная адресация: используется, когда адрес инструкции или данных фиксирован и жестко задан в памяти. Абсолютный адрес требует, чтобы объектный модуль загружался в конкретное место памяти.**
   - **Относительная адресация: используется, когда адрес инструкции или данных задается относительно базового адреса, что позволяет перемещать объектный модуль в память без необходимости изменения инструкций.**

---

2. **Формальные аргументы (параметры) макрокоманды:**
   - **Формальные аргументы макрокоманды — это именованные параметры, используемые внутри макроопределения, которые заменяются на фактические значения во время вызова макрокоманды. Они позволяют создавать универсальные макрокоманды для многократного использования с различными входными данными.**

---

3. **Обработка псевдокоманд в 2-х проходном ассемблере:**
   - **Псевдокоманды обрабатываются преимущественно на первом проходе. На этом этапе ассемблер анализирует их, чтобы выполнить задачи, такие как резервирование памяти, определение меток, настройка размещения данных или определение начальных значений. Эти команды не преобразуются в машинный код, но управляют трансляцией.**

---

4. **Типовые функции начального прохода в 3-х проходном ассемблере:**
   - **В начальном проходе 3-х проходного ассемблера выполняются следующие задачи:**
     - **Построение таблицы всех символов, включая метки и переменные.**
     - **Анализ структуры программы, включая формирование информации о сегментах.**
     - **Подготовка предварительных данных для последующих проходов, включая разбиение на блоки или секции.**

5. **Определение интерпретатора:**
   - **Интерпретатор — это программа, которая совмещает процессы трансляции и исполнения, поочередно анализируя, преобразуя и выполняя команды программы без создания отдельного объектного модуля.**

6. **Формы передачи информации внутри компилятора:**
   - **В компиляторе информация передается в следующих формах:**
     - **Таблицы символов: для хранения идентификаторов, их типов и атрибутов.**
     - **Промежуточные представления: например, в виде синтаксических деревьев, триад или матриц.**
     - **Обобщенные команды (внутренние коды): для передачи данных между фазами компиляции.**

7. **Типовые синтаксические классы элементов алгоритмических языков:**
   - **Идентификаторы (имена).**
   - **Литералы (константы).**
   - **Зарезервированные слова (ключевые слова языка).**
   - **Разделители (например, скобки, запятые).**
   - **Операторы (например, арифметические, логические).**

8. **Грамматический разбор сверху вниз и порядок распознавания правил:**
   - **Грамматический разбор сверху вниз предполагает восстановление синтаксического дерева от корня к листьям. Порядок распознавания правил соответствует левому выводу, при котором правила применяются последовательно от начального символа к терминальным.**

9. **Отличие функций компилятора при распределении памяти статического и динамического классов:**
   - **Статический класс памяти: компилятор выделяет фиксированную область памяти для переменных и данных во время компиляции. Эта память не изменяется во время выполнения программы.**
   - **Динамический класс памяти: компилятор готовит структуру для выделения памяти во время выполнения программы (например, через стек или кучу). Для этого взаимодействие с операционной системой или использование библиотек управления памятью может быть частью задачи.**

4 вариант

1. **Определения видов адресации, обрабатываемых ассемблером:**
   - **Символическая адресация: адрес данных или инструкций задается именем (символом), который ассемблер преобразует в конкретный числовой адрес на этапе трансляции.**
   - **Непосредственная адресация: данные указываются прямо в инструкции, и они не требуют обращения к памяти для получения значений.**

2. **Главные различия между постоянными и динамическими таблицами ассемблера:**
   - **Постоянные таблицы: заполняются на этапе инициализации ассемблера, содержат фиксированную информацию, которая не меняется в ходе трансляции (например, таблица операций, содержащая коды инструкций и их мнемоники).**
   - **Динамические таблицы: заполняются или модифицируются в процессе трансляции, отражают текущие данные о программе (например, таблица символов, содержащая имена меток и их адреса).**
   - **Примеры:**
     - **Постоянная таблица: таблица операций (поля: код операции, мнемоника, длина инструкции).**
     - **Динамическая таблица: таблица символов (поля: имя символа, тип, адрес).**

3. **Фактические аргументы макрокоманды:**
   - **Фактические аргументы макрокоманды — это реальные значения или выражения, которые подставляются вместо формальных аргументов в момент вызова макрокоманды. Они определяют, как макрокоманда должна быть развёрнута в конкретной ситуации.**

4. **Обоснование двухпроходной схемы работы ассемблера:**
   - **Двухпроходная схема позволяет эффективно обрабатывать метки, определенные позже их использования.**
     - **На первом проходе строится таблица символов (меток) и вычисляются адреса.**
     - **На втором проходе генерируется объектный код с использованием данных, собранных на первом.**
   - **Это делает двухпроходный ассемблер более гибким и универсальным, чем однопроходный, при меньшей сложности, чем у трехпроходного.**

5. **Факторы, определяющие минимальное теоретическое количество проходов в компиляторе:**
   - **Факторы:**
     - **Наличие контекстной зависимости в языке.**
     - **Структура грамматики входного языка.**
   - **Диапазон: от одного до трех проходов:**
     - **Один проход возможен для простых языков.**

- **Три прохода необходимы для языков с высокой степенью контекстной зависимости.**

---

6. **Понятие “однородное представление программы”:**
   - **Однородное представление программы — это формат, в котором элементы программы разделены на синтаксические классы и унифицированы, исключая несущественные детали (например, пробелы, комментарии).**
   - **Суть однородности: упрощение структуры программы для дальнейших этапов компиляции, например, представление в виде таблицы однородных символов.**

---

7. **Проблемы разбора сверху-вниз с возвратами:**
   - **Проблемы:**
     - **Порядок проверки правил: приходится перебирать все правила для нетерминалов, что увеличивает вычислительную сложность.**
     - **Вид рекурсии: прямолинейная рекурсия затрудняет обработку левой рекурсии, что требует модификации грамматики.**
     - **Возможность множества возвратов делает разбор медленным и ресурсоемким.**

---

8. **Матрица синтаксического дерева как внутренняя форма представления:**
   - **Информация в матрице:**
     - **Оператор.**
     - **Операнд 1.**
     - **Операнд 2.**
     - **Указатели на предыдущие и последующие строки (для оптимизации и анализа).**
   - **Пример строки:**
     - **Поля: оператор (+), операнд 1 (X), операнд 2 (Y), указатель вперед (2), указатель назад (0).**

---

9. **Отличие автоматных грамматик от контекстно-независимых в БНФ:**
   - **Автоматные грамматики:**
     - **Правила не содержат рекурсивных подстановок.**
     - **Все предложения формируются линейно.**
   - **Контекстно-независимые грамматики:**
     - **Могут включать рекурсию, что позволяет описывать вложенные структуры.**
     - **Используют нетерминальные символы для определения более сложных конструкций.**

---

10. **Определение и критерии машинно-зависимой оптимизации:**

- **Определение: машинно-зависимая оптимизация — это процесс улучшения программы с учетом архитектурных особенностей целевой машины (например, использование регистров, кэшей, специфичных команд процессора).**
- **Критерии:**
  - **Учет особенностей аппаратного обеспечения.**
  - **Сокращение времени выполнения программы.**
  - **Оптимизация использования памяти с учетом аппаратных ограничений.**
