Билет №1

1. ***Архітектура пристрою з плаваючою точкою.***

Архитектурой называют комплекс програмно достигнутых аппаратных средств. Архитектура устройства с плав.зап. включает 8 регистров , каждый и з которых включает 80 бит. В архитектуру входят 3 двухбайтных регистра , которые имеют название CW – управл., SW- состояния, TW – рег.тэгов . Слово состояния сохраняет 3-х битовый указатель на вершину вх. стека , который образовывается в этих 8 регистрах . В этом рег. записываются признаки результатов. Они похожи на признаки результатов флагов F центр. процессора .В мл. разрядах исп. 6 бит признаков исключительных ситуаций , кот. возникают при вычислении . СW включает управления режимами процессора . В мл. разрядах есть 6 бит маскирования исключительных ситуаций , кот. фиксируятся в SW . TW использует по 2 бита определения состояния для каждого из 8-ми регистров . Состояние может указывать на неопределенное значение регистра , на обычное или специальное значение регистра.

**Представление для dd**

**|7 mmmmmmmm 0|15 mmmmmmmm 8| 23 c.mmmmmmm 16|31 s ccccccc 24|**

**Представление для dq**

**|7 mmmmmmmm 0|15… 41| 55 cccc.mmmm 48|63 s ccccccc 56|**

**.data**

**X dd (dq) 23.23**

**X=41 b9 d7 0a (40 37 3a e1 47 ae 14 7b)**

Командой **обмена** является команда FXCH она осуществляет обмен двух регистров сопроцессора (лише в стеку, один з регістрів st[0]). Команда применяется в двух форматах: без операндов и с одним операндом. Команда без операндов осуществляет обмен st(0) и st(1), если задан операнд [FXCH ST(i)], обмен осуществляется между st(0) и st(i).

Для ***загрузки*** операндов в стек можно использовать следующие команды: ***FLD*** источник – загрузка в st(0) вещественного числа из области памяти; FLD1, FLDL2T, FLDL2E, FLDLG2, FLDLN2, FLDPI, FLDZ – загрузка в вершину стека констант: вещ. единица (1.0), log2(10), log2(e), log10(2), ln(2), π, нуль соответственно.

Для снятия результата используются команды ***FST (FSTP)*** периемник – команды **сохранения вещ. число** из st(0) **в память или др. регистр** стека. **Различие** команд только в том, что если есть P, то происходит выталкивание значения из вершины стека сопроцессора(+1 к вказівнику стеку).

Приклад:

.data

X dd 0.5

.code

FLD X ; ST[0] 🡨 X

FLDL2E ; ST[1] 🡨 ST[0]; ST[0] 🡨 LOG2(e)

FXCH ; обмін ST[0] і ST[1]

1. ***Машинно-незалежна оптимізація.***

Целью данной фазы обработки программы является уменьшение затрат программы по памяти и по времени.

* изъятие повторных вычислений, кот-е не использовались, фрагменты программ на которые не возможно попасть с помощью дополнительных управляющих операторов.
* преобразование блок-схемы программы, т.е. фрагменты программы , что повторяются в циклах, выходят за границы цикла, что бы избежать повторений.

Наиболее хорошо проработаны алгоритмы для некоторых частных случаев избыточности, однако в общем случае оптимизация связана с анализом смысла и поиском решения задачи. Фазы оптимизации не всегда присутствуют в составе компиляторов - все зависит от целей проектирования компилятора. Если целью проектирования является скорость работы компилятора, то фаза оптимизации не включается в его состав. Если же целью является минимизация затрат памяти и максимизация скорости работы программы, то модуль оптимизации включается в состав компилятора и время его работы может быть соизмеримо с суммарным временем работы остальных модулей компилятора. Ниже рассматриваются несколько простых алгоритмов оптимизации программы в промежуточной форме:

*1) исключение общих подвыpажений;*

*2) вычислений во вpемя тpансляции;*

*3) вынесение инвариантных выражений за цикл;*

*4) оптимизация булевых выpажений.*

Для виконання такого виду оптимізації важливо використовувати комбінації команд, яка використовують спеціальні особливості регістрів загального призначення. Машинно незалежна оптимізація передбачає вилучення окремих груп вузлів з внутрішнього подання і їх заміну більш ефективними елементами, або раніше виконуваними елементами.

Вона включає і інші задачі серед яких є усунення повторних обчислень та обчислень, результат яких не використовується. Щоб усунути повторні обчислення в програмі необхідно проаналізувати граф програми на наявність однакових під графів, для яких використовуються однакові значення змінних підграфа. Однакові підграфи можна замінити посиланнями на перше використання підграфа. Для цього треба вміти організовувати пошук чергового підграфа серед підграфів програми. Для того щоб реалізувати такий пошук відносно швидким доцільно побудувати індекс над вершинами підграфа за визначеним відношенням підгарафа. Однак аналіз областей існування значень змінних потребує додаткових інформаційних структур, які використовувались в тому чи іншому іншому піддереві.

1. ***Способи організації драйверів в ОС.***

*Драйвер периферийного устройства* служит логическим *интерфейсом между подсистемой ввода/вывода операционной системы* *и* обслуживаемыми ею *аппаратными средствами*. В некоторых случаях драйвер заменяет или расширяет определенные аспекты BIOS'а и таким образом несет ответственность за обработку характеристик обслуживаемых аппаратных средств. В других драйверах физический ввод/вывод осуществляется исключительно через BIOS. В ранних версиях операционной системы *MS-DOS* не была предусмотрена процедура установки драйверов. Однако, *начиная с версии 2.0*, программисты получили возможность дополнять операционную систему. Несмотря на отсутствие общих стандартов для резидентных программ, *интерфейс между операционной системой и драйвером периферийного устройства жестко определен*, поэтому большинство драйверов может работать друг с другом совместно.

Драйверы бывают двух типов: *ориентированные на символьный либо блоковый обмен данными*. Драйверы, обслуживающие хранение и/или доступ к данным на дисках либо других устройствах прямого доступа, обычно ориентированы на блоковый обмен данными. Блоковые драйверы передают данные фиксированными порциями - отсюда и происходит их название.

Драйвер состоит из трех основных частей: ***заголовка, программы стратегий и программы прерываний***. *Заголовок* описывает возможности и атрибуты драйвера, присваивает имя драйверу символьного устройства и содержит NEAR (только смещение, одно слово) указатели на программы стратегий и прерываний, а также FAR (смещение и сегмент, двойное слово) указатель на следующий драйвер в цепочке драйверов (цепочка является однонаправленной, что затрудняет поиск ее начала). Указатель на следующий драйвер устанавливает MS-DOS сразу после завершения процедуры инициализации, а в самом драйвере ему должно быть присвоено начальное значение -1 (FFFFFFFFH). *Драйвер должен помещаться в 64K памяти, т.к. программы стратегий и прерываний содержат лишь смещения внутри выделенного драйверу сегмента*.

**Программа стратегий.** Программа стратегий вызывается, когда драйвер устанавливается при загрузке операционной системы, а также при каждом сгенерированном операционной системой запросе на ввод/вывод. Единичный запрос на ввод/вывод от прикладной программы может породить несколько запросов к драйверу. *Задача данной* программы заключается лишь в том, чтобы *сохранить* где-нибудь некоторый *адрес для дальнейшей обработки*. Этот адрес, *передаваемый в паре регистров ES:BX*, *указывает на структуру, называемую* ***заголовком запроса****,* которая содержит информацию, сообщающую драйверу, какую операцию он должен выполнить.

rlength db 0 ; 0 - длина существенных данных в заголовке

unit db 0 ; 1 - номер элемента

command db 0 ; 2 - фактическая команда

status dw 0 ; 3 - состояние после возврата

reserve db 8 dup (0) ; 5 - зарезервировано для DOS

media db 0 ; 13 - дескриптор среды

address dd 0 ; 14 - длинный указатель для ввода/вывода

count dw 0 ; 18 - число символов для ввода/вывода (целое без знака)

sector dw 0 ; 20 - начальный сектор

Существенно то, что ***программа стратегий*** *не осуществляет никаких операций ввода/вывода, а лишь сохраняет адрес заголовка запроса для последующей обработки программой прерываний*. В мультизадачной системе этот адрес должен был бы храниться в некотором массиве, который при последующем вызове процедуры прерываний подвергался бы сортировке с целью оптимального использования устройства. Под управлением MS-DOS программа прерываний вызывается сразу после программы стратегий. Следует обратить внимание на то, что *между вызовами программ стратегий и прерываний допустимы прерывания*, а это может создать трудности, если драйвер написан в предположении, что между этими двумя вызовами проходит "нулевое время".

**Программа прерываний.** В программе прерываний *выполняется вся фактическая работа драйвера*, поэтому она является наиболее сложной его частью. *При вызове этой программы исследуется командный байт (третий байт) ранее сохраненного заголовка запроса* и в зависимости от его значения выполняются те или иные действия. Программа прерываний *обычно использует командный байт в качестве индекса для некоторой управляющей таблицы*, вызывая таким образом нужную процедуру для каждой команды. Конечно, при желании можно использовать таблицу переходов.

Если вместо цикла ожидания использовать систему апаратных прерываний по готовности, то нет необходимости менять структуру драйвера, но цикл ожидания надо заменить на обращения к системной программе, обеспечивающей взаимодействие с обработчиком прерываний. Для реализации системы прерываний(в частности апаратных прерываний по готовности внешних устройств). В интерфейсные схемы включен блок програмируемых прерываний. Этот блок, как и ВУ, подключается через порты ввода/вывода, т.е. в нем есть порты программирования и порты настройки. В машинах типа IBM PC настройка этих устройств стандартизована, т.е. она выполняется так, что обращение к аппаратным прерываниям осуществляется по адресам(номерам векторов прерываний с 08h по 0Fh и с 0C0h...0C7h). Таким образом, блок программируемого прерывания имеет 16 входов, к которым можно подключать сигналы готовности ВУ. При поступлении сигнала готовности блок ПП генерирует команду int<Nвектора>.

Структура підпрограм драйверів пристроїв включає наступні блоки:

1. видача команди на пристрій для його підготовки до обміну
2. очікування готовності пристрою до обміну
3. виконання власне обміну
4. видача на пристрій команди для закінчення операції обміну
5. організація обміну драйвера даними з програмою, яка його використовує.

При створенні мікропроцесорів фактично було повторено процес створення програм обміну для зовнішніх пристроїв, але вже з деякою систематизацією. В структурі процесорів Pentium використали системи обміну через порт введення/виведення.

Порт- фізична адреса пристрою, яка розпознається чеерз інтерфейсні схеми in al(ax,eax),[dx(edx)]

Підключення зовнішніх пристроїв до мікропроцесорів виконувалось шляхом визначення вихідного командного порту, вхідного порту стану, які мали однаковий номер, та вхідного та вихідного порту для введення/виведення даних, які також мали однакову адресу. Тому щоб написати узагальнений простий драйвер треба було визначити адреси портів

CMPRT EQU 41h

CTPRT EQU CMPRT

INPRT EQU 42h

OUTPRT EQU INPRT

drIn Proc

mov al,CmOn

out CMPRT,al

in al,STPRT

kew: test al,RdyBit

jnz kew

in al,INPRT

push ax

mov al,CmOff

out CMPRT,dl

pop ax

ret

drIn endp

Билет №2

1. ***Форми даних пристрою з плаваючою точкою.***

Устройство с плавающей точкой, а именно, микросхема 8087 работает с 7-ю типами данных, 6-ть из которых присущи только этой микросхеме. четыре формата представляют целые числа, а это слово – 16 бит (единственный формат данных общий для 8088 и 8087) – оператор dw, короткое целое – 32 бита – оператор dd и длинное целое – 64 бита (эти три формата представлены в двоичном дополнительном коде) – оператор dq и четвертый - упакованные десятичные числа (10 байт, 1 байт - знаковый) – оператор dt. три формата представляют вещественные числа или числа с плавающей точкой, а это - короткий формат (32 бита), длинный (64 бита), и специальный (80 бит) – формат промежуточного действительного числа для повышенной точности.

***Представление для dd***

***|7 mmmmmmmm 0|15 mmmmmmmm 8| 23 c.mmmmmmm 16|31 s ccccccc 24|***

***Представление для dq***

***|7 mmmmmmmm 0|15… 41| 55 cccc.mmmm 48|63 s ccccccc 56|***

.data

X1 dw 2

X2 dd 5

X3 dq 6

**Устройство с плавающей точкой**, а именно, микросхема 8087 работает с 7-ю **типами данных**, 6-ть из которых присущи только этой микросхеме. четыре формата представляют целые числа, а это слово – 16 бит (единственный формат данных общий для 8088 и 8087) – оператор dw, короткое целое – 32 бита – оператор dd и длинное целое – 64 бита (эти три формата представлены в двоичном дополнительном коде) – оператор dq и четвертый - упакованные десятичные числа (10 байт, 1 байт - знаковый) – оператор dt. три формата представляют вещественные числа или числа с плавающей точкой, а это - короткий формат (32 бита), длинный (64 бита), и специальный расширеный (80 бит) – формат промежуточного действительного числа для повышенной точности.

Итак, 0,625 = 0,101b. При записи вещественных чисел всегда выполняют нормализацию — умножают число на такую степень двойки, чтобы перед десятичной точкой стояла единица, в нашем случае 0,625 = 0,101b = 1,01b \* 2-1

Говорят, что число имеет мантиссу 1,01 и экспоненту -1. Как можно заметить, при использовании этого алгоритма первая цифра мантиссы всегда равна 1, так что ее можно не писать, увеличивая тем самым точность представления числа дополнительно на 1 бит. Значение экспоненты хранят не в виде целого со знаком, а в виде суммы с некоторым числом так, чтобы хранить всегда только положительное число и чтобы было легко сравнивать вещественные числа — в большинстве случаев достаточно сравнить экспоненту. c = 2^(n-1) --1 + p, где n- кол. разрядов для предоставления порядка, p – знач. двоичного порядка для представления мантиссы числа с единицей в целой части. Вещественные форматы, используемые в процессорах Intel:

*короткое вещественное*: бит 31 — знак мантиссы, биты 30 – 23 — 8-битная экспонента + 127, биты 22 – 0 — 23-битная мантисса без первой цифры;

*длинное вещественное*: бит 63 — знак мантиссы, биты 62 – 52 — 11-битная экспонента + 1024, биты 51 – 0 — 52-битная мантисса без первой цифры;

*расширенное вещественное*: бит 79 — знак мантиссы, биты 78 – 64 — 15-битная экспонента + 16 383, биты 63 – 0 — 64-битная мантисса с первой цифрой (то есть бит 63 равен 1).

FPU выполняет все вычисления в 80-битном расширенном формате, 32- и 64-битные числа используются для обмена данными

1. ***Методи висхідного синтаксичного аналізу.***

Алгоритм должен содержать в себе блоки обращения к лексическому анализу для получения лексем, и возвращения к семантической обработке.

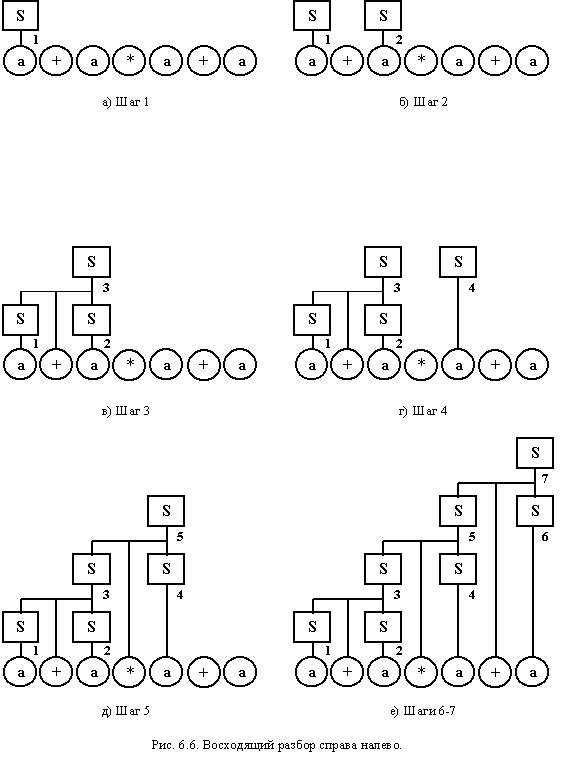
Для того чтобы алгоритм имел более общую форму, в начальных установках алгоритма целесообразно сформировать фиксированный элемент с низким приоритетом фиктивной операции, который будет служить признаком конца обработки выражения. Ображения к процедуре, в результате можно получить или одну лексему или пару (операнд и операция). Сравнение приоритетов текущей операции с предыдущей. Если новой выше, то ее рядом с операндом следует запомнить в стэке и снова обратится к *выражению,* если нет, то к программе семантической обработки с тем, чтобы получить со стэка предыдущую операцию и выполнить необходимые действия.

Программы с таким алгоритмом можно записать в трех вариантах :

1) программа которая использует специальный стэк в виде массива;

2) рекурсивная процедура на языке высшего уровня, которая использует системный стэк для сохранения лексем как аргументов рекурсивных вызовов и адресов возврата в программу;

3) циклическую программу на языке асм., которая будет использовать в середине процедуры только для сохранения промежуточных значений.

При **восходящем разборе** дерево начинает строиться от терминальных листьев путем подстановки правил, применимых к входной цепочке, опять таки, в общем случае, в произвольном порядке. На следующем шаге новые узлы полученных поддеревьев используются как листья во вновь применяемых правилах. Процесс построения дерева разбора завершается, когда все символы входной цепочки будут являться листьями дерева, корнем которого окажется начальный нетерминал. *Если, в результате полного перебора всех возможных правил, мы не сможем построить требуемое дерево разбора, то рассматриваемая входная цепочка не принадлежит данному языку*.

Восходящий разбор также непосредственно связан с любым возможным выводом цепочки из начального нетерминала. Однако, эта связь, по сравнению с нисходящим разбором, реализуется с точностью до "наоборот".

На рис. приведены примеры построения деревьев разбора для грамматики **G6** и процессов порождения цепочек, представленных выражениями.

В общем случае *для восходящего разбора* строится так называемые *грамматики предшествования*. В грамматике предшествования строится матрица попарных отношений всех терминальных и не терминальных символов. При этом определяется три вида отношений: R предшествует S (R<•S); S предшествует R (R•>S); и операция с одинаковым предшествованием (R•=S); четвертый вариант отношение предшествия отсутствует (это говорит о недопустимости использования R и S рядом в тексте записи на этом языке)

**(з конспекту)**

**Найпростіший алгоритм висхідного розбору для аналізу математичних виразів. На основі пріоритетів окремих операцій. Порівнюємо пріоритет наступної з пріоритетом попередньої. Незалежно від результатів порівняння цбого пріоритету треба обрати варіант подальшої роботи - якшо має низький пріоритет - відкласти обробку операції записюючи її в стек. Якщо пріоритет більш високий - виконати інтерпретацію чи саму операцію. Результат такого розбору буде дерево розбору (дерево підлеглості)**

1. ***Роль переривань в побудові драйверів.***

Драйвер состоит из трех основных частей: ***заголовка, программы стратегий и программы прерываний***.

**Программа прерываний.** В программе прерываний *выполняется вся фактическая работа драйвера*, поэтому она является наиболее сложной его частью. *При вызове этой программы исследуется командный байт (третий байт) ранее сохраненного заголовка запроса* и в зависимости от его значения выполняются те или иные действия. Программа прерываний *обычно использует командный байт в качестве индекса для некоторой управляющей таблицы*, вызывая таким образом нужную процедуру для каждой команды. Конечно, при желании можно использовать таблицу переходов.

*Заголовок запроса содержит всю необходимую информацию для корректной обработки каждой команды* и сообщает вызывающей программе (в большинстве случаев таковой является MS-DOS) о состоянии запроса после завершения соответствующей процедуры. *Слово, хранящее состояние после возврата, разбито на несколько полей. Оно содержит бит ошибки*, указывающий на то, что в оставшейся части содержится специфический код ошибки, *бит выполнения*, сигнализирующий о том, что требуемая операция была завершена, и *бит занятости*, призванный в первую очередь сигнализировать о текущем состоянии устройства.

Обязательно должны присутствовать три раздела драйвера: **ЗАГОЛОВОК ДРАЙВЕРА, ПРОГРАММА СТРАТЕГИЙ и ПРОГРАММА ПРЕРЫВАНИЙ.** Программа ПРЕРЫВАНИЙ это не тоже самое, что программа обработки прерываний, которая может присутствовать в качестве необязательной части работающего по прерываниям драйвера. На самом деле, программа ПРЕРЫВАНИЙ - это точка входа в драйвер для обработки получаемых от MS-DOS команд.

Выше представлен скелет драйвера устройства. Хотя структура драйвера похожа на структуру .COM программы, важно отметить следующие отличия:

|  |
| --- |
| Заголовок драйвера |
| Область данных драйвера |
| Программа СТРАТЕГИЙ |
| Вход в программу ПРЕРЫВАНИЙ |
| Обработчик команд |
| Программа обработки прерываний |
| Процедура инициализации |
|  |

* *Программа начинается с нулевого смещения, а не 100H*.
* Образ программы начинается с директив определения данных для заголовка драйвера.
* Программа не содержит директивы ASSUME для стекового сегмента.
* Программа не содержит директивы END START.

Заголовок драйвера, программы СТРАТЕГИЙ и ПРЕРЫВАНИЙ

DRIVER SEGMENT PARA

ASSUME CS:DRIVER,DS:NOTHING,ES:NOTHING

ORG 0

START EQU $ ; Начало драйвера

;\*\*\*\*\*\*\* ЗАГОЛОВОК ДРАЙВЕРА\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

dw -1,-1 ; Указатель на следующий драйвер

dw ATTRIBUTE ; Слово атрибутов

dw offset STRATEGY ; Точка входа в программу STRATEGY

dw offset INTERRUPT ; Точка входа в программу INTERRUPT

db 8 dup (?) ; Количество устройств/поле имени

;\*\*\*\*\*\*\* РЕЗИДЕНТНАЯ ЧАСТЬ ДРАЙВЕРА

req\_ptr dd ? ; Указатель на заголовок запроса

;\*\*\*\*\*\*\* ПРОГРАММА СТРАТЕГИИ

; Сохранить адрес заголовка запроса для программы СТРАТЕГИЙ

; в REQ\_PTR.

; На входе адрес заголовка запроса находится в регистрах ES:BX.

STRATEGY PROC FAR

mov cs:word ptr [req\_ptr],bx

mov cs:word ptr [req\_ptr + 2],bx

ret

STRATEGY ENDP

;\*\*\*\*\*\*\* ПРОГРАММА ПРЕРЫВАНИЙ

; Обработать команду, находящуюся в заголовке запроса.

; Адрес заголовка запроса содержится в REQ\_PTR в форме

; СМЕЩЕНИЕ:СЕГМЕНТ.

INTERRUPT PROC FAR

pusha ; Сохранить все регистры

lds bx,cs:[req\_ptr] ; Получить адрес заголовка запроса

…

INTERRUPT ENDP

…

DRIVER ENDS

END