Т
Министерство образования Российской Федерации
Мухосранский государственный университет им. Э. Мулдашева

Кафедра иного разума

Исследование влияния торсионного поля на моск землян

Реферат студента 1 курса какого-то факультета
Иванова Ивана

Научный руководитель:
к.т.н, доцент _____ Х.З. Мамаев

Заведующий кафедрой:
д.ф.-м.н., профессор _____ В.С. Луговской

Мухосранск, 2006

Содержание

1 13.09.2024 лекция 2

Специальным образом реализуется и используется сегмент стека

фото

Стек используется для временного хранения данных, для организации работы с подпрограммами

Для того, чтобы стек можно было использовать для хранения и фактических и локальных параметров, после перередачи фактических параметров значение указателя на вершину стека можно сохранить в регистре BP и тогда к глобальным параметрам можно обратиться, используя инструкцию BP + к, а к локальным -

Регистр флагов. Регистр FLAGS или EFLAGS определяет состояние процессора и программы в каждый текущий момент времени. В реальном и защищенном режиме CF — Перенос PF — четность AF — полуперенос ZA — флаг нуля SF — флаг знака TF — флаг трассировки IF — флаг прерывания DF — флаг направления OF — флаг переполнения

Только в защищенном режиме:

- AC — флаг выравнивания операндов
- VM — флаг виртуальных машин
- RF — флаг маскировки прерывания
- NT — флаг вложенной задачи
- IOPL — флаг

VIF — , VIP, ID

1.1 Регистр флагов

CF устанавливается в единицу, если в рез-те выполнения операции(например, в рез-те сложения перенос из старшего разряда, а при вычитании - заём) PF = 1, если в младшем байте результата содержится четное кол-во единиц AF = 1, если в рез-те выполнения команды сложения(вычитания) осуществляется перенос(заём) из 3 разряда байта в 4 (из 4 в 3) ZF = 1, если все разряды результат окажутся равны 0 SF всегда равен знаковому разряду результата TF = 1, прерывает работу процессора после каждой выполненной команды(пошаговая откладка программы) IF — если сбросить этот флга в 0, то процессор перестанет обрабатывать прерывания от внешних устройств. Обычно его сбрасывают на короткое время для выполнения критических участков программы DF определяет направление обработки строк данных, если DF = 0 — обработка строк идет в сторону увеличения адресов 1 — в сторону уменьшения(от старших к младшим). Автоматическое увеличение или уменьшение содержимого регистра указателей индексов SI и DI. OF = 1, если результат команды превышает саксимально допустимый для данной разрядной сетки IOPL = 1, если уровень привелегии текущей программы

меньше значенгия этого флажка, то выполнение команды ввод/вывод для этой программы запрещен NT - определяет режим работы вложенных задач Флаги в защищенном режиме мы использовать не будем, рассказала вскользь.

1.2 Регистры 64-х битного процессора

16 целочисленных 64 битных регистра общего назначения(RAX, RBX, RCX, RDX, RBP, RSI, RDI, RSP, R8 — R15) 8 80 битных

1.3 Оперативная память

Состоит из байтов, каждый байт состоит из 8 информационных битов 0 - 3 цифровая часть байт 4 - 7 зонная часть байта 32 - х разрядный процессор может работать с ОП до 4 ГБайт и, следовательно, адреса байтов изменяются от 0 до 2^{32} ст - 1 $00000000_{16} - FFFFFFFF_{16}$

Байты памяти могут объединяться в поля фиксированной и переменной длины

Фиксированная длина - слово(2 байта), двойное слово(4 байта). Поля переменной длины могут содержать

Адресом поля является адрес младшего входящего в поле байта. Адрес поля может быть любым. ОП может использоваться как непрерывная последовательность байтов, так и сегментированная.

физический адрес(ФА) записывается как: <сегмент>: <смещение>, т.е. он может быть получен по формуле ФА = АС + ИА, где АС - адрес сегмента, ИА - исполняемый адрес, т.е. ИА - смещение

В защищенном режиме программа может определить до 16383 сегментов размером до 4 ГБайт, и таким образом может работать с 64 терабайтами виртуальной памяти.

Для реального режима АС определяется сегментным регистром и для получения двадцатиразрядного двоичного адреса байта необходимо к содержанному сегментного регистра, смещенного на 4 разряда влево, прибавить шестнадцатиразрлное спещение ИА. Например, адрес следующей выполняемой команды:

ФА = (CS) + (IP)
(CS) = $7A15_{16}$ = 0111 1010 0001 0101 0000
(IP) = $C7D9_{16}$ = 1100 0111 1101 1001
ФА = 86929_{16} = 1000 0110 1001 0010 1001

1.4 Оперативная память

Процессор ix86 вместе с сопроцессором могут обрабатывать большой набор различных типов данных: целый числа без знака, целые числа со знаком, действительные числа с плавающей точкой, двоично-десятичные числа, символы, строки, указатели.

Целые числа без знака могут занимать байт, слово, двойное слово и принимать значени из диапазонов: 0 - 255, 0 - 65535, 0 - 4294967295

Дополнительный код положительного числа равен самому числу. Дополнительный код отрицательного числа в любой системе счисления может быть получен по формуле:

$= 10^n - [X]$, где

Числа с плавающей точкой могут занимать 32 бит, 64 бит, или 80 бит, и называется короткое вещественное, длиное вещественное, рабочее вещественное. Формат числа с плавающей точкой состоит из трёх полей:

<знак числа>, <машинный порядок>, <мантисса>.

- Короткое вещественное $1 + 8 + 23 - 10^{+-32} - +10^{+-32}$
- Длинное вещественное $1 + 11 + 52 - 10^{+-308} - +10^{+-308}$
- рабочее вещестыенное $1 + 15 + 64 - 10^{+-4932} - +10^{+-4932}$

Машинный порядок (Пм) включает в себя неявным образом знак порядка и связан с истинным порядом (Пи) формулой:

Пм = Пи + 127 10 (1023 10 16383 10)

Предполагается, что мантисса нормализована в старший и старший единичный разряд мантиссы не помещается в разрядную сетку.

Двоично - десятичные чисоа могут обрабатываться 8-ми разрядные в упакованном и неупакованном формате, и сопроцессором могут обрабатываться 80 разрядные данные в упакованном формате. Упакованный формат предполагает хранение двух цифр в байте, а неупакованнй - хранит одну цифру в цифровой части байта.

1.5 Форматы данных

Символьные данные - символы в виде ASCII. Для любого символа отводится один байт. Строковые данные - это последовательности байт, слов или двойных слов. Указатели - существуют два типа указателей: длинный, занимающий 48 бит - селектор(16) + смещение(32) и короткий указатель, занимающий 32 байта - только смещение

1.6 Форматы команд

Команда - это цифровой двоичный код, состоящий из двух подпоследовательностей двоичных цифр, одна из которых определяет код операции(сложить, умножить, переслать), вторая - определяет операнды, участвующие в операции и место хранения результата

Рассматриваемый процессор может работать с безадресными командами, одно-, двух-, и трехадресными командами. Команда в памяти может занимать от 1 до 15 байт и длина команлды зависит от кода операции, количества и места расположения операндов. Одноадресные команлы могут работать с операндами, расположенными в памяти и регистрах, для двухадресных команд существует много форматов, такие как:

- R-R
- M-M
- R-M
- M-R
- R-D
- M-D

где R - регистр, M - память, D - данные

Операнды могут находиться в регистрах, памяти и непосредственно в команде и размер операндов может быть - байт, слово или двойное слово. Исполняемый адрес операнда в общем случае может состоять из трех частей

Существуют различные способы адресации операндов, такие как:

1. регистровая
2. непосредственная
3. прямая
4. косве

2 26.09.2024 лекция 3

2.1 Примеры использования адресаций

фото1 Примеры команд с различной адресацией

1. Регистровая
2. Непосредственная

   MOV AX, 25 ; 25 -> AX CONST EQU 34h
3. Прямая

   Если известен адрес памяти, начиная с которого размещается операнд, то в команде можно непосредственно указать этот адрес.

   MOV AX, ES : 0001

   ES - регистр сегмента данных, 0001 - смещение внутри сегмента

   Содержимое двух байтов, начиная с адреса (ES) + 0001 пересылаются в AX

   Прямая адресация может быть записана с помощью символического имени, которое предварительно поставлено в соответствие некоторому адресу памяти, с помощью специальной директивы определения памяти.

   Например: DB — слово
4. Косвенно-регистровая Данный вид адресации отличается от регистровой адресации тем, что в регистре содержится не сам операнд, в адрес области памяти, в которой операнд содержится

   MOV AX, [SI]

   Могут использоваться регистры:

   SI, DI, BX, BP, EAX, EBX, ECX, EDX, EBP, ESI, EDL

   Не могут использоваться: AX, CX, DX, SP, ESP.
5. По базе со смещением

   MOV AX,[BX] + 2 = MOV AX,[BX + 2] = MOV AX,2[BX] MOV AX,[BX + 4]
6. Прямая с индексированием

   MOV AX, MAS[SI]

   MAS — адрес в области памяти. С помощью этой адресации можно работать с одномерными массивами. Символическое имя определяет начало массива, а переход от одного элемента к другому осуществляется с помощью содержимого индексного регистровая

7. По базе с индексированием

   MOV AX, Arr[BX][DI]

   Эта адресация используется для работы с двумерными массивами. Символически имя определяет начало массива, с помощью базового регистра осуществляется перехолд от одной строки матрицы к другой, а с помощью индексного регистра — переход от одного элемента к другому внутри строки.

2.2 Особенности использования команд пересылки

1. Нельзя пересылать информацию из одной области памяти в другую;
2. Нельзя пересылать информацию из одного сегментного регистра в другой;

   PUSH DS

   POP ES
3. Нельзя пересылать непосредственно операнд в сегментный регистр, но если такая необходимость возникает, то нужно использовать в качестве промежуточного одни из регистров общего назначения.
4. Нельзя изменять командой MOV содержимое регистра CS.
5. Данные в памяти хранятся в «Перевернутом» виде, а в регистрах в «нормальном» виде, и команда пересылки учитывает это, например,
6. Размер передаваемых данных определяется типом оперндов в команде

   ```
   X DB ?
   Y DW ?
   MOV X, 0
   MOV Y, 0
   MOV AX, 0
   MOV[SI], 0
   ```

   В последнем случае необходимо использовать ...
7. Если тип обоих операндов в команде определяется, то эти типы должны соответствовать друг другу

   ```
   MOV AH, 500
   ```

   К командам пересылки относят команду обмена значений операндов.

   ```
   XCHG OP1, OP 2
   ```

   Для перестановки значений байтов внутри регистра используют **BSWOP**

   ```
   (EAX) = 12345678h
   BSWOP EAX; (EAX) = 78563412h
   ```

2.3 К командам пересылки относят:

Команды конвертирования

```
CBW ; безадресная команда
CWD ;
CWE ;
CDF ;
```

Команды условной пересылки **CMOVxx**

```
CMOV AL, BL ; если
```

Загрузка адреса.

```
LEA OP1, OP2 ;вычисляет адрес OP2 и пересылает первому операнду, который может быть толь
LEA BX, M[BX][DI]
```

2.4 Команды и директивы в Ассемблере

Три этапа обработки программы

1. Получаем машинный код (исходных модулей может бвть один или несколько)
2. Модули объединяются в один исполняемый модуль .exe(для .com нужно выполнить ещё один этап обработки .exe -> .com с помощью системной программы или в среде разработки с помощью ключа)
3. Программы

Программма состоит из команд, директив и комментарив.

Команды в процессе трансляции(асемблирования) преобразуются в машинный формат, директивы определяют способы ассемблирования и редактирования, выделяют место под исходные, промежуточные и окончательные данные/результаты, а комментарии используются для пояснения выполняемых действий.

Команда ассемблера состоит из 4 полей:

[<имя>[:]]<код операции>[<операнды>][;комментарии]

Директива, как и команда, состоит из 4 полей:

[<имя>[:]]<код псевдооперации>[<операнды>][;комментарии]

Здесь <имя> — символическое имя Ассемблера,

Здесь <код псевдооперации> — определяет назначение директивы.

Операндов может быть различное количество и для одной директивы.

Например:

```
1
```

Исходный модуль на Ассемблере — последовательность строки

```
; сегмент стека
    Sseg Segment...



;



;
```

В кодовом сегменте специальная директива...

```
ASSUME SS:SSeg, DS::DSeg, CS::CSeg, ES::DSeg;
```

на DSeg отсылаются и DS, и ES
дальще по фоткам **Я УСТАЛЬ**

3 27.09.2024 лекция 4

3.1 Слова, константы, выражения, переменные

фото 1

Строковые данные — последовательность символов, заключенная в апострофы или кавычки

Также,как и в языках программирования высоко уровня, в Ассемблере могут использоваться именованные константы. Для этого существует специальная директива EQU.

Например,

M EQU 27 : директива EQU присваивает имени M значение 27.

Переменные в Ассемблере определются с помощью директив определения данных и памяти, например,

```
v1 DB
```

Арифметические операции: +, -, *, /, mod
Логические операции: NOT, AND, OR, XOR
Операции отношений: LT, LE, EQ, NE, GT, GE
Операции сдвига: сдвиг влево(SHL), сдвиг вправо(SHR)
Специальные операции: offset, PTR
Метка или переменная:
PTR BYTE

3.2 Директива определения

Общий вид директивы определения следующий

где X B, W, D, F, Q, T

В поле операндов может быть "?". одна или несколько констант, разделенных запятой. Имя, если оно есть, определяет адрес первого байта выделяемой области. Директивой выделяется указанное количество байтов ОП и указанные операнды пересылаются в эти поля памяти. Если операнд "? то в соответствующее поле ничего не заносится.

1. Если операндом является символическое имя IM1, которое соответсвует смещению в фрагменте 03AC1h, то после выполнения

   M DD IM1

   будет выделено 4 байта памяти. Адрес — M. Значение — 03AC1h.

2. если необходимо выделить 100 байтов памяти и заполнить 1, то это можно сделать с помощью специального повторителя DUP

   D DB 100 DUP(1)

3. Определение одномерного массива слов, адрес первого элемента массива — имя **MAS**, значение его 1.

4. Определение двумерного массива:

```
v1 DB
```

5. **const EQU 100**

   D DB Const DUP(?) ;выделить 100 байтов памяти. В директиве определения байта(слова) максимально допустимая константа 255(65535).

   С помощью директивы определения байта можно определить строковую константу длинной 255 символов, а с помощью определения

3.3 Команда прерывания Int, команды работы со стеком

С помощью этой команды приостанавливается работа процессора, управление передается DOC или BIOS и после выполнения какой-то системноц обрабатывающей программы, кправление передается команде6 следующей за командой INT.

Выполянемые действия будут зависеть от опернада, параметра команды INT и содержания некоторых регистров.

Например, чтобы вывести на экран "!"необходимо:

```
v1 DB
```

Сткек определяется с помощью регистров SS и SP(ESP).

Сегментный регистр SS содержит адрес начала сегмента стека.

ОС сама выбирает этот адрес и пересылает его в регистр SS.

Регистр SP указывает на вершину стека и при добавлении элемента стека содержимое этого регистра уменьшается на длину операнда.

Добавить элемент в стек можно с помощью команды

где операндом может быть как регистр, так и переменная.

Удалить элемент с вершины стека можно с помощью операции

Для i386 и > PUSH A/ POP A позволяют положить в стек, удалить содержимое всех регистров общего назначения в последовательности AX, BX, CX, DX, SP, BP, SI, DI

Пример программы, использующей комнады пересылки содержимого 4 байтов из одной области памяти в другую и вывод на экран.

```
TITLE Prim.asm
Page, 120
;описание сегмента стека
Sseg Segment Para stack 'stack'
    DB 100h DUP(?)
    Sseg ends
;описание данных
Dseg Segment Para Public 'Data'
    DAN DB '1', '3', '5', '7'
    REZ DB 4 DUP (?)
```

```
Dseg ends
;
;
Cseg Segment Para Public 'Code'
```

3.4 Директива сегмента

Общий вид

Любой из операторов может отсутствовать.

1. Если есть readonly, то будет выведено сообщение об ошибке при попытке записи в сегмент.

2. Операнд выравнивание определяет адрес начала сегмента. BYTE — WORD — DWORD — Para — Page —

3. тип определяет тип объединения сегментов. Значение stack указывается в сегменте стека, для остальных сегментов public. Если такой параметр присутствует, то все сегменты с одним именем и различными классами объединяются в один последовательно в порядке их записи. Значение 'Common' говорит, что сегменты с одним именем объединены, но нне последовательно, а с одного и того же адреса так, что общий размер сегмента будет равен не сумме, а максимуму из них. Значение

3.5 Точечные директивы

В программе на Ассемблере могут использоваться упрощенные(точечные) директивы. .MODEL — директива, определяющая модель выделяемой памяти для программы.

Модель памяти опредлеяется параметром:

tiny — под всю программу выделяется 1 сегмент памяти.

small — под данные и под программу выделяются по одному сегменту.

medium — под данные выделяется один сегмент, под программу выделяется несколько сегментов.

compact — под программу выделяется один сегмент, под данные выделяется несколько сегментов.

large — под данные и под программу выделябтся по n сегментов.

huge — позволяет использовать сегментов больше, чем позволяет ОП. (можно на внешний)

```
.MODEL small
.STACK 100h
.Data
St1 DB 'Line1, $'
St2 DB 'Line2, $'
St3 DB 'Line3, $'
```

```
.CODE
begin: MOV AX, @DATA
    MOV DS, AX
MOV AH, 9
MOV DX, offset St1
INT 21h
MOV DX, offset St2
INT 21h
MOV DX, offset St3
INT 21h
MOV AX, 4C00h
INT 21h
END begin
```

'$' — конец строки, которую необходимо вывести на экран
В результате выполнения программы:
Line1
Line2
Line3,

3.6 Com-файлы

После обработки компилятором и редактором связей получаем exe-файл, который содержит блок начально загрузки, размером не менее 512 байт, но существует возможность создания другого вида исполняемого файла, который может быть получен на основе exe-файла

4 04.10.2024 лекция 5

4.1 Арифметические операции

фото 1 Пример, работая в байтами, получим:

$$250 + 10 = (250 + 10) mod 2^s = 260 mod 256 = 4$$

Пример: в байте

$$1 - 2 = 1 + 2^8 - 2 = 257 - 2 = 255, CF = 1$$

Сложение(Вычитание) знаковых чисел сводится к сложению(вычитанию) с использованием дополнительного кода.

1

В байте:

$$-1 = 256 - 1 = 255 = 11111111,\ -3 = 256 - 3 = 25311111101,$$
$$3 + (-1) = (3 + (-1)) mod 256 = (3 + 255) mod 256 = 2,$$
$$1 + (-3) = (1 + (-3)) mod 256 = 254 = 11111110,$$

Ответ получили в дополнительном коде, следовательно результат получаем в байте по формуле $X = 10^n - |X|$, т.е. x = 256 - 254 = |2| и знак минус. Ответ -2.

Переполнение происходит, если есть перенос из старшего цифрового в знаковый, а из знакового нет и наоборот, тогда OF = 1. программист сам ре (фото)

4.2 Сложение и вычитание в Ассемблере

Арифметические операции изменяют значение флажков OF, CF, SF, ZF, AF, PF

В Ассемблере команда "+":

```
ADD OP1, OP2 ;(OP1) + (OP2) -> OP1
ADC OP1, OP2 ;(OP1) + (OP2) + (CF) -> OP1
XADD OP1, OP2 ;i486 u >
(OP1) <-> (OP2) (меняет местами), (OP1) - (OP2) -> OP1
```

В Ассемблере команда '-':
МНОГО ФОТОК!
последнее

4.3 Команды безусловной передачи управления

Команда вызова процедуры:

CALL <имя> ;

Адресация может быть использована как прямая, так и косвенная.

При обращении к процедуре типа NEAR в стеке сохраняется адрес возврата, адрес команды, следующей за CALL содержится в IP или EIP.

При обращении к процедуре типа FAR в стеке сохраняется полный адрес возврата CS:EIP.

Возврат из процедуры реализуется с помощью команды **RET**.

Она может иметь один из следующих видов:

```
RET [n]      ;возврат из процедуры типа NEAR и из процедуры типа FAR
RETN [n]     ;возврат только из процедуры типа NEAR
RETF [n]     ;возврат только из процедуры типа FAR
```

Параметр n является необязательным, он определяет какое количество байтов удаляется из стека после возврата из процедуры.

фото

4.4 Команды условной передачи управления

Команды условной передачи управления можно разделить на 3 группы:

команды, используемые после команд сравнения команды, используемые после команд, отличных от команд сравнения, но реагирующие на значения флагов

```
JZ/JNZ
JC/JNC
```

5 11.10.2024 лекция 6

5.1 Команды условной передачи управления

фото 1 Примеры:

```
JE M1 ;передача управления на команду с меткой M1, если ZF = 1
JNE M2 ;передача управления на команду с меткой M2, если ZF = 0
JC M3 ;передача управления на команду с меткой M3, если CF = 1
JNC M4 ;передача управления на команду с меткой M4, если CF = 0
    ADD AX, BX
    JC M
```

если в результате сложения CF = 1, то управление передается на команлду с меткой M, иначе — на команду, следующую за JNC

```
SUB AX, BX
JZ Met
```

фото2 фото3 таблица

5.2 Команды управления

Команды условной передачи управлвения могут осуществлять только короткий переход, а команды безусловной передачи упраавления могут реализовать как короткую передачу так и длинную. Если необходимо осуществить условный дальний переход, то можно использовать jx вместе jmp следующим образом:

```
if AX = BX goto m
if AX <> BX goto L
Goto m      ;
------------------
L: ---------------- ;
```

На Ассемблере это будет так:

```
cmp AX, BX
jne
```

С помощью команд jx jmp можно реализовать цикл с предусловием:

```
while x > 0 do S;
    beg:    cmp x, byte ptr 0
            jle fin
                S
            jmp beg
    fin:    ------------------
```

и постусловием:

```
do S while x > 0;
    beg:
                S
                cmp x, byte ptr 0
                jg beg
    fin:        ------------------
```

5.3 Команды для организации циклов

1. loop<метка>
2. loope<метка> loopz<метка>
3. loopne<метка> loopnz<метка>

По команде в форме

1. (CX) = (CX) - 1 и если (CX) <> 0, <метка>
2. (CX) = (CX) - 1 и если (CX) <> 0 и одновременно ZF = 1, <метка>

   Цикл завершается, если или (CX) = 0 или ZF = 0 или (CX) = (ZF) = 0
3. (CX) = (CX) - 1 и если (CX) <> 0 и одновременно ZF = 0, <метка>

   Выход из цикла осуществляется, если или (CX) = 0 или ZF = 1 или одновременно (CX) = 0 и (ZF) = 1

Примеры: (фото)

5.4 Примеры использования команд условного перехода, сравнения и циклов

Дана матрица целых байтовых величин, размером 4*5, необходимо подсчитать количество нулей в каждой строке и заменить их числом 0FFh. Под стек отведёс 256 байтов, программу оформим как две последовательные процедуры: внешняя (FAR) — это связь с ОС, внутренняя (NEAR) — решение поставленнной задачи

```
;prim.asm
title prim.asm
page , 132
Sseg segment para stack 'stack'
        db 256 dup(?)
Sseg ends
Dseg segment para public 'data'
Dan db 0,2,5,0,91 ;адрес первого элемента массива
```

```
        db 4,0,0,15,47 ; имя "--- Dan
        db 24,15,0,9,55
        db 1,7,12,0,4
Dseg ends
Cseg segment para public 'code'
        Assume cs:cseg, ds:dseg, ss:sseg
start proc far
        push DS     ;для связи
        push AX     ;с ОС
        mov BX, Dseg;загрузка адреса сегмента данных
        mov DS, BX  ;в регистр DS
        call main
        ret
start endp
main proc near
                mov BX, offset Dan
                mov CX, 4       ;количество повторений внешнего цикла
nz1: push CX
        mov DL, 0   ;счётчик нулей в строке матрицы
            mov SI, 0
            mov CX, 5   ;количесвто повторений внутреннего цикла
nz2: push CX
            cmp byte ptr [BX+SI], 0
            jne mz
            mov byte ptr [BX+SI], 0FFh
            ine DL
mz: ine SI
            pop CX
kz2: loop nz2
        add DL, '0' ;вывод на экран
        mov AH, 6   ;количество нулей
        int 21h
            add BX, 5   ;переход к следующей строке матрицы
            pop CX
kz1: loop nz1
            ret
main endp
Cseg ends
            end start
```

5.5 Массивы в Ассемблере

Массивы в языке Ассемблер описывается директивами определения данных, возможно с использованием конструкции повторения DUP.

Например, **x DW 30 dup(?)**

Так можно описать массив x, состоящий из 30 элементов длиной в слово,

но в этом описании не указано как нумеруются элементы массива, т.е. это может быть **x[0..29] и x[1..30] и x[k..29 + k].**

Если в задаче жестко не оговорена нумерация элементов, то в Ассемблере удобнее считать элементы от нуля, тогда адрес любого элемента будет записываться наиболее просто:

адрес (x[i]) = x + (type x) * i

фото

С учетом этих формул для записи адреса элемента массива можно использовать различные способы адресации.

Для описанного выше массива слов, адрес его i-го элемента равен:

x + 2*i = x + type(x) * i

Т.е. адрес состоит из двух частей: постоянной x и переменной 2 * i, зависящей от номера элемента массива. Логично использовать адресацию прямую с индексированием: x — смещение, а 2 * i — в регистре модификаторе Si или DI **x[i]**

Для двумерного массива, например:

я

фото

Фрагмент программы, в которой фото...

5.6 Команды побитовой обработки данных

К командам побитовой обработки данных относятся логические команды, команды сдвига, установки, сброса и инверсии битов.

Логические операнды: and, or, xor, not. Для всех логических команд, кроме not, операнды одновременно не могут находиться в памяти, OF = Cf = 0, AF — не определён, SF, ZF, PF определяются результатом команды.

and OP1, OP2 ; (OP1) логически умножается на (OP2), результат OP1

Пример: (AL) = 1011 0011, (DL) = 0000 1111, **and AL, DL** ;(AL) = 0000 0011

Второй опреанд называют маской. Основным назначением команды and является установка в ноль с поощью маски некоторых разрядов первого операнда. Нулевые разряды маски обнуляют соответствующие разряды первого операнда, а единичные оставляют соответствующие разряды первого операнда без изменения. Маску можно задавать непосредственно в команде и модно извлекать из регистра или памяти.

Например:

1. and CX, 0FFh ;маской является константа
2. and AX, CX ;маска содержится в регистре
3. and AX, TOT ;маска в ОП по адресу (DS) + TOT

4. and AX, TOT[BX+SI] ;. . . в ОП по адресу (DS)+(BX)+(SI)+(TOT)
5. and TOT[BX+SI], CX ; вноль устанавливаются некоторые разряды ОП
6. and CL, 0Fh ;в ноль устанавливаются старшие 4 разряда регистра CL

Команда — **or OP1, OP2** ;рез

6 11.10.2024 лекция 7

фото до 10:58

6.1 К

Имя первоц структуры dst, второй — dst+4, третьей — dst+8 и т.д.

Работать с полями структуры можно также, как с полями переменной комбинированного типа в языках высокого уровня:

<имя переменной> . <имя поля>

Например, dt1.y, dt2.m, dt3.d

Ассемблер (дальше по фото 10:57)

Точка, указанная при обращении к полю, это оператор Ассемблера, который вычисляет адрес по формуле:

<адресное выражение> + <смещение поля в структуре>

Тип полученного адреса совпадает с типом поля, т.е.

type(dt1.m) = type m = byte

адресное выражение может быть любой сложности, например:

1. mov AX, (dts+8).y
2. mov SI, inc (dts[SI]).m ; Аисп = (dts + (SI)).m = (dts + 8).m
3. lea BX, dt1 mov[BX].d, 10 ; Аисп = (BX) + d = dt1.d

Замечания:

1. 1
2.

Одно исключение: если поле описано как строка, то оно можеит иметь начальным значением строу той же длины или меньшей, в последнем случае строка дополняется справа пробелами. Например:

```
student struc
        f DB 10DUP(?)    ;
        i DB
```

Примеры программ с использванием данных типа структура.

```
; prim.asm - прямое обращение к полям структуры
    .model tiny
    .code
    org 100h    ;
Start: mov AH, 9
```

```
    mov DX, offset message
    int 21h

    lea DX, st1.s
    int 21h
    lea DX, st1.f
    int 21h
    lea DX, st1.i
        int 21h
        ret
    message DB "hello", 0dh, 0ah, "$"

    tst struc   ;
                s DB "student", "$"
                f DB "Ivanov", "$"
                i DB "Ivan", "$"
            tst ends
            st1 tst < > ;описание переменной типа tst
        end start


            ; prim.asm - обращение к полям структуры в цикле
Start: mov AH, 9
        mov DX, offset message
        int 21h
    mov SI, 0
    mov CX, 3
m1: lea DX, st1[SI]
    int 21h
    add SI, 9
    loop m1
    ret
message DB "hello", 0dh, 0ah, "$"
    tst struc   ;описание типа структуры
                s DB "student", "$"
                f DB "Ivanov", "$"
                i DB "Ivan", "$"
            tst ends
                st1 tst < >
                    end start
```

7 25.10.2024 лекция 8

7.1 К

```
; prim3.asm - обращение к полям структур: цикл в цикле для работы с 2-мя структурами
Start: mov AH, 9
        mov DX, offset message
        int 21h
    lea BX, st1     ;
    mov CX,2
m2: push CX
    mov SI, 0
    mov CX, 3
m1: push CX
        lea DX, [BX][SI]
            int 21h
            add SI, 9
            pop CX
            loop m1
add BX, type tst

pop CX
    loop m2
    ret
messsage DB "hello", 0dh, 0ah, "$"
    lst struc
            s DB ?
            f DB ?
            i DB ?
    tst ends
            st1 tst <"student$", "$",>
```

7.2 Записи в Ассемблере

Запись в Ассемблере, также как и структура состоит из различнх данных различной длины, но запись — упакованные данные, занимающие не полные ячейки памяти (байты, слова), а их части. Поля записи представляют собой последовательности битов(разрядов), прижатые друг к другу, между полями не может быть пробелов, размер полей мождет быть различным, но в сумме их размер не должен превышать 16 разрядов. Т.е. сумма размеров полей — это размер записи, а размер записи может быть 8 или 16 рарзрядов. Если сумма разрядов полей меньше 8 или 16, то поля должны быть прижаты к правой границе поля, а лишние левые разряды равны 0. К записи не имею отношения, не влияют на неё и не используются.

Чтобы использовать запись, нужно вначале описать её тип, а затем опи-

сать переменную такого типа. Описание типа записи может располагаться в любом месте, но до описания переменных. **ВАЖНО! Поля в записи имеют собственные имена, располагаются в памяти, как при описании, слева направо. Но обращаться к полям записи, как к полям структуры напрямую нельзя.**

Директива описания типа записи имеет вид:

<имя типа записи> record <поле> , <поле> <поле>::=<имя поля>:<размер>[=<выражение>]

Здесь <размер> и <выражение> — константные выражения.

<размер> определяет размер поля в битах, <выражажение> определяет значение поля по умолчанию. Знак ? не допускается.

Например: ==

Год, записанный двумя последними цифрами Имена полей, также как и в структурах, должны быть уникальными в рамках всей программы, в описании они перечисляются слева направо. <Выражение> может отсутствовать, если оно есть, то его значение должно умещаться в отведенный ему размер в битах. Если для некоторого поля выражажение отсутствует, то его значение по умолчанию равно нулю, неопределенных полей не может быть.

Определение директивой record имя типа(Trec, TData) используется далее как директива для описания переменных-записей такого типа.

имя записи имя типа записи <начальные значения>,

угловые скобки здесь не метасимволы, а символы языка, внутри которых через запятую указываются начальные значения полей.

Начальными значениями могут быть:

В отличии от структуры, знак ? определяет нулевой начальное значение, а «пусто», как и в структуре, определяет начальное значение равнм значению по умолчанию. Например:

Также, как и для структур:

1

Одной директивой можно описать массив записей, используя несколько параметров в поле операндрв или конструкцию повторения, например,

1

Описали 100 записей с начальными значениями, равными принятыми по умолчанию.

Со всей записью в целом можно работать как обычно с байтами или со словами, т.е. можно реализовать присваивание Rec1=Rec2:

```
mov AL, Rec2
mov Rec1, AL
```

Для работы с отлельными полями записи существует специальные операторы **1**

оператор mask имеет вид:

Mask <имя поля записи> Mask <имя записи или имя типа записи>

Значением этого оператора является «маска» — это байт или слово, в зависиморсти от размера записи, соедржащее единицы в тех разрядах, которые принадлежат полю или всей записи, указанных в качестве операнда, и нули в остальынх, не

Пример. Выявить родившихся 1-го числа, для этого придется выделять поле D и сравнивать его значение с 1-ей

```
m1:
    mov AX, Dat1
    and AX, mask D
    cmp AX, 1
    je yes
no:



    jmp
```

Обьединение (union) является тином данных, состоящим из нескольких переменных, которые хранятся, начиная с одного и того же адреса памяти (перекрывая друт друua). Этот тип данных есть во многих языках программирования, например, в языке C. Пример на Ассемблере:

```
pdate record d:5, m:4, y:7=4
udate strus
        d db ?; локальное имя!
        m db ?
        y dw 4
udate ends
date union
        Dp pdate <>;2 байта
        Du udate <>;4 байта
        7, dw 3 dup (?); 6 байт
date ends

MyDate date <>
mov ax,MyDate.Dp;2 байта
and ax,Mask d; d - глобальное имя из записи pdate
mov al, MyDate.Du.d
mov ax, MyDate.Z; ax := Z[0]
mov ax, sizeof MyDate; sizeof MyDate=6
```

В данном случае имена полей в объединении, как и имена полей в структурах, локализованы внутри обьединения. Объединения используются в основном для экономии места в памяти при хранении данных.

На рисунке показано расположение полей объединения с именем MyDate в памяти.

(фото 1)

Наложение полей объединения друг на друга

7.3 Работа с подпрограммами

Программа, оформленная как процедура, к которой обращение происходит из ОС, заканчивается командой возврата ret;

Подпрограмма, как вспомогательный алгоритм, к которому возможно многократное обращение с помощью команды call, тоже оформляется как процедура с помощью директив proc и endp. Структуру процедуры можно оформить так:

```
<имя процедуры> proc <параметр>

                <тело процедуры>

            ret

<имя процедуры> endp
```

В Ассемблере один тип подпрограмм — процедура. Размещать ее можно в любом месте программы, но так, чтобы управление на нее не попадало случайно, а только по команде call. Поэтому описание ПП принято располагать в конце программного сегмента (после последней исполняемой команды), или вначале его — перед первой псполняемой командой.

Графическое представление(фото)

7.3.1 Замечания

1. После имени в директивах proc и endp двоеточие не ставится, но имя считается меткой, адресом первой исполняемой команды процедуры.

2. Метки, описанные в ПП, не локализуются в ней, поэтому они должны быть уникальными в рамках всей программы.

3. Параметр в директиве начала процедуры один — FAR или NEAR

Основная проблема при работе с ПП в Ассемблере — это передача параметров и возврат результатов в вызывающую программу.

Существуют различные способы передачи параметров:

1. по значению

2. по ссылке

3. по возвращаемому значению
4. по результату
5. отложенным вычислением

Параметры можно передавать:

1. через регистры
2. в глобальных переменных
3. через стек **(наиболее универсальный, используется в языках высоко уровня)**
4. в потоке кода
5. в блоке параметро

Передача параметров через регистры — нанболее простой способ. Вызывающая программа записывает в некоторые регистры фактические параметры...

Процедура получает адрес начала этого блока при помощи любого из рассмотренных методов: в регистре, в переменной, в стеке, в коде пли даже в другом блоке параметров. Примеры использования этого способа многие функции QC и BIOS, например, поиск файла, использующий блок параметров DIA, или загрузка и исполнение программы, использующая блок параметров EPB

Передача параметров по значению и по ссылке.

При передаче параметров по значению процедуре передается значение фактического параметра, оно копируется в I , и I использует копию, поэтому изменение, модификация параметра оказывается невозможным. Этот механизм используется для передачи параметров небольшого размера.

Например, нужно вычислить c = tax (a, b) + max (7, a-1). Здесь все числа знаковые, размером в слово. Используем передачу параметров через регистры Процедура получает параметры через регистры AX ц BX, результат возвращает в регистре AX.

7.4 Передача параметров по ссылке

Оформим как процедуру вычисление x = x div 16

Процедура имеет один параметр-переменную x, которой в теле процедуры присваивается новое значение. T.c результат записывается в некоторую ячейку памяти. И чтобы обратиться к процедуре с различными параметрами, например, a ц b, ей нужно передавать адреса памяти, где. хранятся значения переменных з и в. Передавать адреса можно любым способом, в том числе и через регистры. Можно использовать различные регистры, но чаще используются BX, BP, SI, DI. Пусть адрес параметра передается через регистр BX, тогда фрагмент программы:

```
;основная программа
==============
lea BX, a
call Proc_dv
lea BX, b
call Proc_dv
==============
```

Процедура:

```
push SX mov CL. 4
shr word ptr [BX], CL; x = x div 16
pop CX
ret
```