Регистри на процесора.

Регистрите на процесора представляват памет в самия чип. Достъпа до тях е много по-

бърз, отколкото до RAM. Регистрите също са спцеиализирана памет. Те се използват от процесора при изпълнение на аритметични и логически операции.

Регистрите на процесора са важни за системното програмиране, защото функциите на

BIOS и DOS получават информация от тях.

От гледна точка на системното програмиране , нищо не се е променило в регистрите от процесора 8086 насам. Това е така, защото DOS и BIOS са се развили във връзка с него, така че те поддържат само 16-битовите регистри, включени в този процесор.
32-битовите регистри на intel 80386 и 80486 не могат да бъдат използвани в системното програмиране под DOS. Затова от значение са само 8088 регистрите, които се използват във всички следващи чипове.

Всеки регистър е с голе2222222222222222байта). Ако всичките му битове съдържат 1 се получава десетичното число 65 535. Това е най-голямото десетично число, което може да се представи в 16 бита. Така че един регистър може да съдържа всички стойности от 0 до 65535 (FFFF шестнадесетично, или 1111111111111111 двоично).

Регистрите се разделят на четири групи :

1. Общи регистри – това са AX (accumulator - акумулатор), BX (base - база), CX (counter - брояч), DX (data - данни), DI (destination index - индекс получател), SI (source index - индекс източник), SP (stack pointer - стеков указател) и BP (base pointer - базов указател).
2. Сегментни регистри – това са DS (data segment – сегмент за данни), CS (code segment – сегмент за програмата), ES (extra segment – допълнителен сегмент) и SS (stack segment – стеков сегмент).
3. Програмен брояч – това е IP (instruction pointer – указател към инструкция)
4. Флагов регистър.

Различните регистри спомагат за оптимизиране на начина, по който програмите обработват данните. Последното е основна задача на микропроцесора.

DOS и стандартните функции, съхранявани в ROM широко използват регистрите с общо предназначение, особено AX, BX, CX и DX. Тяхното съдържание указва на DOS какви задачи трябва да извърши и какви данни да използва за тяхното изпълнение.

Основно влияние на тези регистри оказват математическите операции (събиране, изваждане, умножение, деление и т. н.), както и входно-изходните инструкции. Регистрите,

завършващи с “X” заемат особено място в 8088, защото могат да бъдат разделени на две.

Всеки от тях съдържа един 16-битов и два по-малки 8-битови регистъра.

Общите регистри са важни за извикване на DOS и BIOS функции и се използват за предаване на параметри на конкретна функция, която се нуждае от тях. Те са под въздействието на математически операции, които са в центъра на вниманието на всички софтуерни дейности на ниво процесор. AX, BX, CX и DX заемат специално положение, защото могат да бъдат разделени на два 8-битови регистъра. Това означава, че всеки от тях се състои от 3 регистъра – един 16-битов и два 8-битови.

Регистрите имат H (high – висок) и L (low – нисък) означения. Така че 16-битовия регисър AX може да бъде разделен на 8-битовите AH и AL. Означенията H и L са въведени, защото L регистъра съдържа нисшите 8 бита (оъ 0 до 7) на X регистъра, а H регистъра – старшите 8 бита (от 8 до 15). И така, AH се състои от битовете 8 – 15, а AL от битовете 0 – 7 на регистъра AX.

Тези три регистъра не могат да бъдат разгледани независимо един от друг. Например ако третият бит на AH се промени, тогава бит 11 на AX се променя автоматично. Стойностите и на двата регистъра AH и AX се променят. AL в случая остава непроменен, тъй като той се състои от 0 до 7 бит на AX (т. е. 11 бит на AX не принадлежи на AL. Тази връзка между

AX, AH и AL важи също за всички други регистри с общо предназначение, завършващи на

“X” и може да бъде изразена математически.

Специфицирайки например CH или CL може да се чете или записва 8-битова данна от или във всеки от тях. Същото се отнася за регистъра CX и 16-битова данна.

Флаговият регистър осъществява връзката между последователните машинни инструкции, като запазва резултата от математическите и логическите операции. Например след използването на carry flag (флага за пренос) при събиране на два 16-битови регистъра, една програма определя дали резултата е по-голям от 65535 и ако е така го представя като 32-битово число. Знаковият, нулевият и препълващият флаг изпълняват подобни задачи и могат да бъдат използвани след като два регистъра бъдат сравнени, за да установят дали стойността в първия е по-голяма, по-малка или равна на стойността на втория.

Само carry flag и zero flag (нулев флаг) са важни за системното програмиране на езици от високо ниво. Повечето от DOS и BIOS функциите използват тези флагове, за да означат грешки. Например за недостатъчна памет или непознати имена на файлове.

За системното програмиране е особено важно как процесора образува адресите на паметта, защото трябва постоянно да се подават адреси на буфери на DOS или BIOS функциите. В тези случаи трябва да е известно какво прави процесора.

Едно от най-добрите постижения при процесора 8088 е създаването на пакет от инструкции, които са по-висши от тези на ранните 8-битови микропроцесори (6502, Z80 и т. н.). Друго сполучливо попадение е предоставянето на лесен достъп до повече от 64 килобайта памет. Това е важно, защото нарастването на възможностите на процесора позволява на програмиста да създава по-сложни приложения, които от своя страна изискват повече памет. Създателите на 8088 са повишили капацитета на паметта или адресното пространство на микропроцесора (повече от 16 пъти) до един мегабайт.

Броят на клетки от паметта, които един процесор може да адресира, зависи от адресния регистър. Тъй като всяка от тях е адресирана чрез определяне на уникален номер, максималната стойност, която може да се съдържа в адресния регистър определя големината на адресното пространство. Ранните микропроцесори използваха 16-битов адресен регистър, който даваше на потребителя достъп до адреси от 0 до 65535. Това съответства на капацитета от 64 килобайта на тезо процесори.

За адресирането на един мегабайт памет, адресния регистър трябва да има най-малко 20 бита. По времето, когато е разработван 8088, не е било възможно да се използва 20-битов регистър, така че конструкторите са използвали друг начин за достигане на 20-битова шина. Съдържанието на две 16-битови числа се използва за достигане на 20-битов адрес.

Едното от тези 16-битови числа се съдържа в сегментен регистър. 8088 има четири такива сегментни регистъра. Второто число се съдържа в друг регистър или в клетка от паметта. За да се образува 20-битово число, съдържанието на сегментния регистър се премества наляво с четири бита (или се умножава по 16) и второто число се прибавя към полученото.

Сегментният адрес, който се формира от сегментния регистър, показва началото на сегмента в паметта. Когато се създава адреса, отместването се добавя към сегментния адрес. Отместването показва позицията в сегмента, определен от сегментен регистър.

Отместването не може да бъде по-голямо от 16 бита, т. е. Стойност от 0 до 65535 за размер на сегмент.

В случая, когато отместването е 0 и началото също е 0, се получава адрес, отговарящ на клетка от паметта (или физически адрес) 0. Ако сегмента е 1, ще се получи физически адрес 16, а не 1. Това е така, застото сегментният адрес се умножава по 16 при образуването на адреса.

Ако се увеличава още сегмента, а отместването остава 0, се получават физически адреси 32, 48, 64 и т. н. Според този принцип максималния адрес е 1 мегабайт, когато сегмента достигне максималната си стойност 65535 (FFFFh). Обратно, ако сегмента остава постоянен, а се мени отместването, първия сочи сегмент от паметта, в който може да има достъп до 65535 различни клетки. Всеки сегмент съдържа 64K. Отместването показва разстоянието между желаната клетка и началото на сегмента.

Въпреки че сегментите във физическата памет са през 16 байта, всеки от тях съдържа по 64K. Така че те се припокриват. Например физическия адрес 130 може да се представи по различни начини. Така както адреса, образуван от сегмент 0 и отместване 130 сочи тази клетка, така също и сегмент 1 и отместване 114, сегмент 2 и отместване 98 и т. н. сочат в тази клетка.

Припокриването на сегменти е удобно за работа. При определен адрес може да се избере произволна комбинация от сегмент и отместване, която го сочи. Проверка се прави, като се

Умножи сегмента по 16 и бъде прибавен към отместването.

Начало на сегмент не може да бъде всяка клетка от паметта. Като се умножи сегментния регистър по 16, винаги се получава число, което е кратно на 16. Така например сегмент не може да започва от клетка с адрес 22.

Физическият адрес се получава от съчетанието на сегмента и отместването. Той

определя точния номер на клетката от паметта, която става досъпна. За разлика от него, сегментния адрес и отместването са относителни.

8088 съдържа четири сегментни регистъра, които имат важна роля при изпълнението на програма на машинен език. Те съдържат основната структура на всяка програма, която се състои от множество от инструкции (или код). Програмата също обработва променливи и данни. Една структурирана програма държи кода и данните отделно едно от друго, докато се намират в паметта. Един удобен начин за тяхното отделяне е предоставянето на собствени сегменти на всеки от тях. Сегментни регистри са следните:

CS (Code Segment – кодов сегмент) използва регистъра IP като отместване и заедно правят адрес от паметта. По този начин се определя адреса на поредната ин-

Струкция. IP се нарича и с името Program Counter (програмен брояч). Когато

Процесорът изпълнява текущата инструкция, IP регистъра автоматично се пре-

Насочва към следващата. Това осигурява изпъленеието на инструкцууте в пра-

вилен ред.

DS Както CS, регистъра DS (Data Segment – сегмент за данни) съдържа сегментните данни, до които програмата има достъп (записва или чете данни в или от памет-

та). Отместването на адреса се получава от друг регистър или от текущата инс-

трукция.

SS Регистъра SS (Stack Segment – стеков сегмент) определя сегмента на стека, както подсказва самото му име. Стекът действа като място за временно съхранение за

Някои програми на машинен език. Той позволява бързо запазване и възстановя-

ване на данни за различни инструкции. Например, когато се изпълнява инструк-

цията CALL, процесорът поставя адреса за сръщане в стека. SS заедно със SP

или BP образуват адреса, на върха на стека.

ES Последният сегментен регистър е ES (Extra Segment – допълнителен сегмент).

Той се използва от някои машинни инструкции за адресиране на повече от 64K

данни или за пренос на данни между два различни сегмента от паметта.

С помощта на ES е възможно да оставим DS да сочи в изходната област от паметта, а за обръщение към областта, в която се копира, се използва ES. 8088 и неговите наследници имат дори инструкции, които могат да копират цял буфер, като се предполага, че преди тяхното изпълнение сегмента на адреса на първата област е зареден в DS, а този на областта в която се копира – в ES.

За да се извърши копирането , инструкциите също трябва да знаят какви са отместванията на двете области в техните сегменти от паметта. Те очакват да получат началото на шървата област в SI, а на втората в DI регистъра. Според въведените преди това означения това може да се запише по следния начин: тези инструкции копират данни от DS:SI в DS:DI.

Два сегментни регистъра могат да определят области от паметта, които се покриват или са напълно различни една от друга. Обикновено една програма не може да заеме целия сегмент със своите код и данни. Затова може да се спести памет чрез припокриването на сегменти. Например може да се запазят данните веднага след програмния код като се определи по подходящ начин съдържанието на регистрите DS и CS.

Атхитектура на копроцесор

Копроцесорът адресира паметта по същия начин, както и процесора, но има и свои собствени регистри за данни и управляващи регистри – осем регистъра за данни, организирани като стек и седем управляващи регистъра, подобни на флаговия регистър. Наборът от инструкции на копроцесора осигурява директен достъп до тези регистри.

Осемте регистъра за данни са 80-битови и са организирани като стек, въпреки че не е задължително винаги да се използват така. Когато се запишат данни в регистъра на върха, предишните данни се преместват в регистрите с по-високи номера, които са по-долу в стека. Регистър 0 е на върха на регистровия стек, а регистъ 7 – на дъното му. Всички данни се съхраняват в регистрите на копроцесора в 10-байтов реален формат.

Всички действия в копроцесора се извършват над 10-байтови числа, защото това осигурява по-висока точност. Инструкциите, които пренасят данни между паметта и копроцесора, автоматично преобразуват данните от и в десетичен 10-байтов реален формат.

Чрез управляващите регистри, най-често в програмата се осъществява достъп до регистъра на състоянието (нарича се също дума на състоянието). Той отразява цялостната работа на копроцесора. Условните флагове C0 – C3 се променят от операциите за сравнение и тестване и се използват за управление на изпълнението на програмата. Флагът за заетост B отразява дали копроцесорът е зает с изпълнението на инструкция. Той се проверява чрез изследване на стойността на думата на състоянието. Последните модели копроцесори автоматично се синхронизират с процесора, така че състоянието на този флаг не трябва да се проверява. Указателят към върха на стека (STP) съдържа номера на регистъра, който се счита за връх на стека ST. Обикновено това е регистъра за данни с номер 0.

Управляващият регистър на копроцесора служи за избор на точност на операциите, на закръглението и на представянето на безкрайно голяма стойност (със или без знак). С него могат да се маскират битове 0 – 5 от думата на състоянието. За зареждане на стойност в управляващия регистър се използва инструкцията FLDCW.

Таг-регистърът (регистър с признаци) отразява какво е съдържанието на всеки от регистрите за данни. Всеки признак показва дали стойността на съответния регистър е валидна, нулева, невалидна или безкрайност, или регистърът е празен. За да може програмата да провеи стойността на таг-регистъра, трябва да се съхрани средата на регистъра посредством инструкциите FSTENV или FSAVE, които записват стйностите на всички регистри на копроцесора в паметта.

Асемблер

Асемблерът използва различни методи за обработка на числа с плаваща запетая (реални числа) и на целочислени данни. За работа с реални числа могат да се използват две възможности – аритметичен копроцесор или наричан за краткост FPU (Floating Point Unit) и с емулираща програма за централния процесор, наричан също за краткост CPU (Central Processor Unit).

Аритметичните копроцесори – чиповете 8087, 80287, 80387SX, 80387DX и 80487SX – работят заедно с централния процесор при обработката на числа с плаваща запетая. Процесорите 486DX, Pentium и тези, които са произведени след тях, съдържат копроцесора в самия чип и изпълняват действията с плаваща запетая директно. Наборът от инструкции и програмирането на всички копроцесори е почти еднакво, въпреки че са проектирани да работят с различни процесори.

Копроцесорите могат да изпълняват различни действия: събиране, изваждане, имножене, деление, коренуване, изчисляване на arctg, log и други трансцедентни функции. Данните могат да бъдат цели 16-, 32- и 64-битови числа със знак, 18-битови BCD числа и, разбира се, числя с плаваща запетая.

Данните с плаваща запетая са в пряк машинен код по стандарта IEEE-754 (IEEE – Institute of Electrical and Electronical Engineers). Те са с три дължини – единична (4 байта), двойна (8 байта) и разширена (10 байта). Развядността на мантисата и характеристиката са съответно 24 и 8, 53 и 11, 64 и 15, като мантисата е двоично нормализирана.При форматите с единична и двойна дължина на мантисата се обработва скрит бит, който не се обменя с паметта. Разширеният формат не оперира със скрит бит на номализираната мантиса.

Копроцесорът използва регистрите си и изпълнява обработката над реални числа, пакетирани BCD числа и дълги цели числа многократно по-бързо от еквивалентните операции, реализирани посредством най-ефективните емулиращи програми, които използват само основния набор от инструкции на централния процесор.

Ако системата не е с процесор 486DX, Pentium или някой от произведените след тях, може да се използва емулатор на 8087, предоставен от асемблера. Достъп до емулатора се получава чрез директивата OPTION EMULATOR, която се поставя пред директивата .MODEL в програмата.

Указателите биват два вида – близки и далечни:

Близкия указател е отместването на даден обект и има ширина само 16 бита. Но достъп до паметта не може да се получи без сегмента на адреса. Затова компилатора го

подготвя и го зарежда автоматично в определения за целта сегментен регистър на централния процесор при осъществяване на достъп до обекта. Поради тази причина използването на близки указатели е възможно само за променливи, намиращи се вътре в създадения от компилатора сегмент с големина 64 килобайта. Далечните указатели обаче, съдържат и сегмента и отместването на адреса, които се запазват като две думи. Младшата дума получава отместването, а старшата – сегмента на адреса.

Представяне на числа с

плаваща запетая в паметта.

Реалните числа, наричани също числа с плаваща запетая, се състоят от три части: знак, мантиса и характеристика (отместен порядък). Асемблерът съхранява тези числа според стандартния формат IEEE-754-85. Той има следния вид:

Четирибайтовото число се нарича число с единична точност (късо реално число, или число тип REAL4)., осембайтовото – число с двойна точност (дълго реално число, или число тип REAL8), а десетбайтовото – число с разширена точност (вътрешно представяне на реално число, или число тип REAL10)

Според стандарта късите реални числа се представят в пряк код с един бит за знака (0 за + и 1 за -), 8-битова характеристика и 24-битова мантиса. Това изисква общо 33 бита, докато за едно число се заделят 32. За да се заобиколи това несъответствие, мантисата използва т.нар. скрит бит, който представлява първият бит на нормализираното реално число (винаги със стойност 1) и затова тя се побира в 23 бита. Нормализираното реално числе се представя така, че да е по-голямо от 1 и по-малко от 2. Това е т. нар. IEEE-754 нормализация. Така цялата част от числото винаги е 1 и не се съхранява в паметта, поради което се нарича скрит бит. Скритият бит се използва и при дългите реални числа, но не и при числата с разширена точност, при които той се съхранява в паметта като единствен бит (No 63) на цялата част.

Повядъкът (експонентата) е множител от вида 2^n . За да могат да се представят отрицателни отрицателни експоненти, стойността в полето на експонентата е с отместване (bias), т. е. действителната експонента се получава чрез изваждане на отместваща стойност от стойността в полето (характеристиката). Отместването за късите реални числа е 127, за дългите реални числа е 1023, а за разширените – 16383.

Изключения от тези правила за представяне са стойностите нула (представя се с нули във всички битове) и безкрайност (представя се с нули във всички битове на мантисата и единици във всички битове на характеристиката).

За дефиниране на реални константи и променливи се използват директивите за дефиниране на данни REAL4, REAL8 и REAL10. С тях се заделя памет съответно за къси (с дължина 4 байта), дълги (с дължина 8 байта) и разпирени (с дължина 10 байта) реални числа. За съвместимост с по-ранните версии на асемблерния език се допуска и използването на директивите DD, DQ и DT. Асемблерът винаги счита числата за десетични и ги преобразува в двоични, като кодира знака, мантисата и характеристиката в съответните битове на числото. Броят на значещите цифри може да се променя при аритметичните операции, защото при закръгляване на числото може да се загуби най-младшата цифра.

Обхват на числата с плаваща запетая:

Тип на данните

Дължина, b

Брой значещи цифри

Обхват на числената стойност

REAL4

32

6-7

от 1,18.10 $^{-38}$ до 3,40.10 ^{38}

REAL8

64

15-16

от 23,23.10 $^{-308}$ до 1,79.10 ^{308}

REAL10

80

19

от 3,37.10 -4932 до 1,18.10 4932

Решение:

По надолу са представени два варианта на решение на поставеното задание:

Първият е по-прост и представлява извеждане на екрана на число тип REAL4, което е предварително дефинирано в изходния код на програмата като константа.

Второто решение, което е подобрен вариант на първото е доста по-гъвкаво: при него след стартиране на програмата, потребителят трябва да въведе числото от клавиатурата, след което на следващия ред се извежда вече въведеното число.

Изходният код на езика Асемблер на първото решение е следният:

```
.MODEL SMALL

.386

.387
```

```
.STACK

.DATA

numb REAL4 2224.5

.CODE

.STARTUP

push numb                ; Зареждане на числото в стека

call display             ; Извикване на процедурата

.EXIT

disp proc near

mov ah,6                 ; Зареждане на функция 6 в ah за изписване на символ на
екрана

mov dl,al                ; Вземане на стойността за извеждане от al

int 21h                  ; Извикване на прекъсване 21 на MS DOS
```

```
ret

disp endp

display proc c,num:real4

local temp:word,

whole:dword,

fract:dword

fstcw temp                    ; Съхране на командната дума в променлива temp

or temp,0c00h                 ; Промяна на командната дума в режим на закръгляване чрез
;изрязване

fldcw temp                    ; Зареждане на командната дума

fld num                       ; Зареждане на параметъра num в Floating point стека

ftst
```

```
fstsw ax              ; Зареждане на думата на състоянието в ax

and ax,4500h          ; Извличане на необходимите битове от думата на
състоянието

.IF ax==0100h         ; Проверка за отрицателност

mov al,'-'

call disp             ; Отпечатване на знака “-“ при изпълване на условието

fabs                  ; Превръщане на ST(0) в абсолютната му стойност

.ENDIF

fld st                ; Зареждане на ST(0) в ST(1)

frndint               ; Закръгляне на ST(0) чрез изрязване на дробната част

fist whole            ; Зареждане на ST(0) в whole като цяло число

fsubr                 ; Изваждане на ST(1) от ST(0)

fabs                  ; Превръщане на ST(0) в абсолютната му стойност
```

```
fstp fract                    ; Зареждане на ST(0) в fract и изтегляне от стека

mov eax,whole                 ; Зареждане на whole в eax

mov ebx,10                    ; Зареждане на константа 10 в ebx

mov cx,0                      ; Инициализиране на броячя cx

push bx                       ; Зареждане на 10 в стека

.WHILE 1

mov edx,0

div ebx                       ; Целочислено делене на eax на ebx резултата се записва в
eax , a ;остатъка в edx

add dl,30h                    ; Превръщане на получения остатък в ASCII кода на съотв
цифра

push dx                       ; и зареждане в стека

.BREAK .IF eax==0             ; прекъсване при остатък = 0
```

```
inc cx

.IF cx==3

push ','

mov cx,0

.ENDIF

.ENDW

.WHILE 1                        ; Цикъл за извеждане на цялата част

pop dx                          ; Зареждане на dx от стека

.BREAK.IF dx==bx                ; прекъсване при dx = 10

mov al,dl

call disp                ; извеждане на цифрата
```

```
.ENDW

mov al, '.'

call disp               ; извеждане на десеттична точка

mov eax, fract                  ; Зареждане на fract в eax

fstcw temp

xor temp,0c00h                  ; Промяна на командната дума

fldcw temp

fld fract                       ; Зареждане  на fract в FP стека

fxtract                          ; Извличане на мантисата в ST(0) и на експонентата в ST(1)
от ;ST(0)

fstp fract                      ; Зареждане на мантисата в fract

fabs                            ; и на абсолютната  стойност на

fistp whole                     ; експонентата в whole
```

```
mov ecx,whole                ; Зареждане на експонентата в ecx

mov eax,fract                ; Зареждане на мантисата в eax

shl eax,9            ; Изчистване на битовете за експонента и знак от eax

rcr eax,cl                   ; Ротиране надясно със стойността на експонентата в ecx

.REPEAT

mul ebx                      ; Умножение на eax с 10. Цялятя част в edx, резултата – в eax

push eax                     ; Зареждане на eax в стека

xchg eax,edx                 ; Размяна на eax и edx

add al,30h                   ; Получаване на ASCII кода на цифрата в eax

call disp            ; и отпечатване

pop eax                      ; Зареждане на eax от стека
```

```
.UNTIL eax==0                ; Прекъсване при нулева стойност в eax

ret

display endp

END
```

Изходният код на второто решение, също на езика Асемблер е следният:

```
.MODEL SMALL

.386

.387

.STACK

.DATA

crlf db 13,10,"$"

numb REAL4 ?
```

```
ten REAL4 10.0

.CODE

get proc

mov ah,1

int 21h

ret

get endp

read proc c,num:near ptr REAL4

LOCAL sign:BYTE,

temp1:word

fldz
```

```
call get

.IF al=='+'

mov sign,0

call get

.ENDIF

.IF al=='-'

mov sign,1

call get

.ENDIF

.REPEAT

fmul ten

mov ah,0
```

```
sub al,30h

mov temp1,ax

fiadd temp1

call get

.UNTIL al'9'

.IF al=='.'

fld1

.WHILE 1

fdiv ten

call get

.BREAK .IF al'9'
```

```
mov ah,0

sub al,30h

mov temp1,ax

fild temp1

fmul st,st(1)

fadd st(2),st

fcomp

.ENDW

fcomp

.ENDIF

.IF sign==1

fchs
```

```
.ENDIF

mov bx,num

fstp REAL4 ptr[bx]

ret

read endp

.STARTUP

invoke read,offset numb

call new_line

push numb

call display

.EXIT
```

```
disp proc near

mov ah,6

mov dl,al

int 21h

ret

disp endp

display proc c,num:real4

local temp:word,

whole:dword,

fract:dword

fstcw temp

or temp,0c00h
```

```
fldcw temp

fld num

ftst

fstsw ax

and ax,4500h

.IF ax==0100h

mov al,'-'

call disp

fabs

.ENDIF

fld st
```

frndint

fist whole

fsubr

fabs

fstp fract

mov eax,whole

mov ebx,10

mov cx,0

push bx

.WHILE 1

mov edx,0

div ebx

```
add dl,30h

push dx

.BREAK .IF eax==0

inc cx

.IF cx==3

push ','

mov cx,0

.ENDIF

.ENDW

.WHILE 1

pop dx
```

```
.BREAK.IF dx==bx

mov al,dl

call disp

.ENDW

mov al, '.'

call disp

mov eax, fract

fstcw temp

xor temp,0c00h

fldcw temp

fld fract

fxtract
```

```
fstp fract

fabs

fistp whole

mov ecx,whole

mov eax,fract

shl eax,9

rcr eax,cl

.REPEAT

mul ebx

push eax

xchg eax,edx
```

```
add al,30h

call disp

pop eax

.UNTIL eax==0

ret

display endp

new_line proc near

mov ah,9

lea dx,crlf

int 21h

ret

new_line endp
```

END

Резултати от трасирането на програмата

По-долу са представени резултатите от трасирането на второто, по-сложно решение на заданието. Трасирането е извършено при зададена тестова стойност 22.5.

AX = 1936

BX = 1938

CX = 0000

DX = 0000

SP = 0400

BP = 0000

SI = 0000

DI = 0000

DS = 1936

ES = 1910

SS = 1938

CS = 1920

IP = 0080

FL = 3202

NV UP EI PL

NZ NA PO NC

AX = 1936

BX = 0002

CX = 0000

DX = 0000

SP = 0400

BP = 0000

SI = 0000

DI = 0000

DS = 1936

ES = 1910

SS = 1938

CS = 1920

IP = 0082

FL = 3202

NV UP EI PL

NZ NA PO NC

AX = 1936

BX = 0020

CX = 0000

DX = 0000

SP = 0400

BP = 0000

SI = 0000

DI = 0000

DS = 1936

ES = 1910

SS = 1938

CS = 1920

IP = 0085

FL = 3212

NV UP EI PL

NZ AC PO NC

AX = 1936

BX = 0020

CX = 0000

DX = 0000

SP = 0420

BP = 0000

SI = 0000

DI = 0000

DS = 1936

ES = 1910

SS = 1936

CS = 1920

IP = 0089

FL = 3202

NV UP EI PL

NZ NA PO NC

AX = 1936

BX = 0020

CX = 0000

DX = 0000

SP = 041E

BP = 0000

SI = 0000

DI = 0000

DS = 1936

ES = 1910

SS = 1936

CS = 1920

IP = 008C

FL = 3202

NV UP EI PL

NZ NA PO NC

AX = 010D

BX = 000D

CX = 0000

DX = 0000

SP = 041E

BP = 0000

SI = 0000

DI = 0000

DS = 1936

ES = 1910

SS = 1936

CS = 1920

IP = 008F

FL = 3297

NV UP EI NG

NZ AC PE CY

AX = 010D

BX = 000D

CX = 0000

DX = 0000

SP = 0420

BP = 0000

SI = 0000

DI = 0000

DS = 1936

ES = 1910

SS = 1936

CS = 1920

IP = 0092

FL = 3212

NV UP EI PL

NZ AC PO NC

AX = 0924

BX = 000D

CX = 0000

DX = 000A

SP = 0420

BP = 0000

SI = 0000

DI = 0000

DS = 1936

ES = 1910

SS = 1936

CS = 1920

IP = 0095

FL = 3212

NV UP EI PL

NZ AC PO NC

ds:000d

41ac0000

AX = 0000

BX = 000A

CX = 0001

DX = 0035

SP = 041C

BP = 0000

SI = 0000

DI = 0000

DS = 1936

ES = 1910

SS = 1936

CS = 1920

IP = 009D

FL = 3246

NV UP EI PL

ZR NA PE NC

AX = 0136

BX = 0020

CX = 0000

DX = 0000

SP = 0414

BP = 041A

SI = 0000

DI = 0000

DS = 1936

ES = 1910

SS = 1936

CS = 1920

IP = 0002

FL = 3213

NV UP EI PL

NZ AC PO CY

AX = 1936

BX = 0020

CX = 0000

DX = 0000

SP = 0416

BP = 041A

SI = 0000

DI = 0000

DS = 1936

ES = 1910

SS = 1936

CS = 1920

IP = 000D

FL = 3213

NV UP EI PL

NZ AC PO CY

AX = 0032

BX = 0020

CX = 0000

DX = 0000

SP = 0416

BP = 041A

SI = 0000

DI = 0000

DS = 1936

ES = 1910

SS = 1936

CS = 1920

IP = 002C

FL = 3216

NV UP EI PL

NZ AC PE NC

AX = 0002

BX = 0020

CX = 0000

DX = 0000

SP = 0416

BP = 041A

SI = 0000

DI = 0000

DS = 1936

ES = 1910

SS = 1936

CS = 1920

IP = 002E

FL = 3202

NV UP EI PL

NZ NA PO NC

ss:0416

1920

AX = 0002

BX = 0020

CX = 0000

DX = 0000

SP = 0416

BP = 041A

SI = 0000

DI = 0000

DS = 1936

ES = 1910

SS = 1936

CS = 1920

IP = 0031

FL = 3202

NV UP EI PL

NZ NA PO NC

ss:0416

0002

AX = 0002

BX = 0020

CX = 0000

DX = 0000

SP = 0416

BP = 041A

SI = 0000

DI = 0000

DS = 1936

ES = 1910

SS = 1936

CS = 1920

IP = 0034

FL = 3202

NV UP EI PL

NZ NA PO NC