J.Nawrocki, M. Antczak, G. Palik, A. Widelska

Ćwiczenie nr 4

Język asemblera

Środowisko uruchomieniowe

1. Pobrać plik ***masm.zip*** (*Macro Assembler 6.15* & *Segmented Executable Linker 5.60)* (`http://www.cs.put.poznan.pl/mantczak/teaching/itc/masm.zip`).
2. Rozpakować powyższe archiwum w katalogu roboczym (na zajęciach najprawdopodobniej C:\Temp).
3. Ważne pliki wykorzystywane podczas kompilacji:
   a. *ml.exe, LINK.EXE* – wykonywalne pliki reprezentujące odpowiednio *Macro Assembler 6.15* i *Segmented Executable Linker 5.60*; są one wykorzystywane w procesie kompilacji plików źródłowych programów napisanych w języku asemblera,
   b. *clean.bat* – służy do usuwania następujących plików: **.bak*, *prog.exe, prog.obj, prog.map, prog.lst, prog.com* z katalogu, w którym się znajduje,
   c. *prog.asm* – kod źródłowy programu zapisany w języku asemblera (*blank.asm* – szkielet programu napisanego w języku asembler, który wystarczy uzupełnić odpowiednimi instrukcjami w celu reprezentowania przez niego określonej funkcjonalności; zawiera kod, który jest niezbędny i wymagany przez wszystkie programy pisane na ćwiczeniach),
   d. *make.bat* – zawiera składnię poleceń wykorzystywanych w celu kompilacji pliku zawierającego kod źródłowy programu napisanego w języku asemblera (np.: `make.bat prog`, w celu kompilacji pliku *prog.asm*).

```
ml   /Fe'kod_wynikowy'   /Fl'listing_kompilacji'   /Fm'mapa_kompilacji'
/Fo'kod_przejściowy' 'kod_zrodlowy'
link 'kod_przejściowy',,,,,
```

np.:

```
ml /Feprog.exe /Flprog.lst /Fmprog.map /Foprog.obj prog.asm
link prog.obj,,,,,
```

Wprowadzenie do asemblera

Podstawowa struktura programu

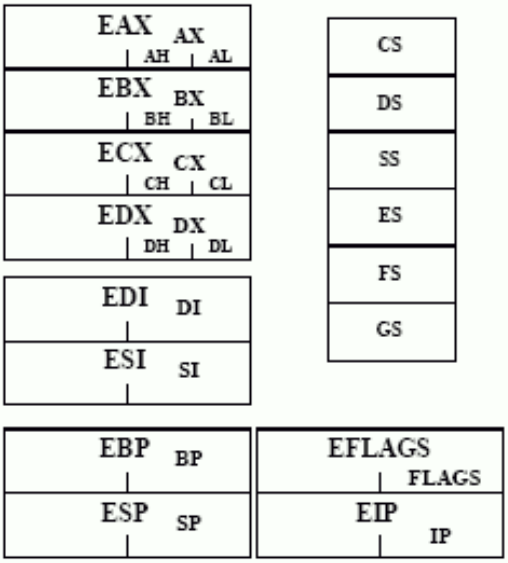
```
; prog.asm
;================================================
.model tiny
.code
;================================================
prog segment
assume cs: prog
start:
                  ; instrukcje
int 3
;================================================
.stack
;================================================
prog ends
end start
```

(*) gwiazdką oznaczone są zadania, które nie są realizowane na ćwiczeniach i są przeznaczone do wykonania jako zadania domowe.

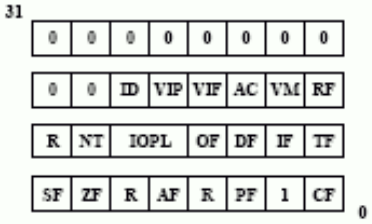
Podstawowe rozkazy procesorów 80x86

1. Struktura rejestrów
2. Arytmetyka binarna i heksadecymalna
3. Podstawowe rozkazy

1. Struktura rejestrów

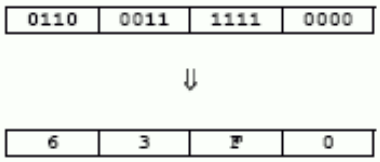


Rejestr znaczników (flagi)



ID	ID flag
VIP	Virtual Interrupt Pending
VIF	Virtual Interrupt Flag
AC	Alignment Check
VM	Virtual 8086 Mode
RF	Resume Flag
NT	Nested Task
IOPL	I/O Privelege Level
OF	Overflow Flag
DF	Direction Flag
IF	Interrupt Flag
TF	Trace Flag
SF	Sign Flag
ZF	Zero Flag
AF	Auxiliary Carry Flag
PF	Parity Flag
CF	Carry Flag

2. Arytmetyka binarna i heksadecymalna



Uzupełnienie do dwóch

```
 3 = 011
 2 = 010          3 - 2 = 3 + - 2
 1 = 001
 0 = 000             (3) 011
-1 = 111            (-2) 110
-2 = 110            ---------
-3 = 101                1 001
                        ↑ przeniesienie
```

3. Podstawowe rozkazy

mov dest,src				przesłanie (dest=src)				
O	D	I	T	S	Z	A	P	C
-	-	-	-	-	-	-	-	-

```
cs:0100 B406            mov ah,06
cs:0102 B80600          mov ax,0006
cs:0105 66B806000000    mov eax,00000006
cs:010B 8AE7            mov ah,bh
cs:010D 8BC3            mov ax,bx
cs:010F 668BC3          mov eax,ebx
```

cmp arg1,arg2				porównanie				
O	D	I	T	S	Z	A	P	C
*	-	-	-	*	*	*	*	*

```
cs:0100 80FC06          cmp ah,06
cs:0103 3D0600          cmp ax,0006
cs:0106 663D06000000    cmp eax,00000006
cs:010C 3AE7            cmp ah,bh
cs:010E 3BC3            cmp ax,bx
cs:0110 663BC3          cmp eax,ebx
```

JGE lab/JNL lab				skocz jeśli SF=OF				
O	D	I	T	S	Z	A	P	C
-	-	-	-	-	-	-	-	-

(*) gwiazdką oznaczone są zadania, które nie są realizowane na ćwiczeniach i są przeznaczone do wykonania jako zadania domowe.

Minimum dwóch liczb
(4 warianty)

program 16-bitowy w segmencie 16-bitowym
(ax = min(bx,cx))

```
cs:0100 8BC1            mov ax,cx
cs:0102 3BD9            cmp bx,cx
cs:0104 7D02            jnl 0108
cs:0106 8BC3            mov ax,bx
cs:0108
```

program 32-bitowy w segmencie 16-bitowym
(eax = min(ebx,ecx))

```
cs:0100 668BC1          mov eax,ecx
cs:0103 663BD9          cmp ebx,ecx
cs:0106 7D03            jnl 010B
cs:0108 668BC3          mov eax,ebx
cs:010B
```

program 32-bitowy w segmencie 32-bitowym
(eax = min(ebx,ecx))

```
cs:0100 8BC1            mov eax,ecx
cs:0102 3BD9            cmp ebx,ecx
cs:0104 7D02            jnl 0108
cs:0106 8BC3            mov eax,ebx
cs:0108
```

program 16-bitowy w segmencie 32-bitowym
(ax = min(bx,cx))

```
cs:0100 668BC1          mov ax,cx
cs:0103 663BD9          cmp bx,cx
cs:0106 7D03            jnl 010B
cs:0108 668BC3          mov ax,bx
cs:010B
```


Skoki: bezwarunkowy i warunkowe

Zasięg skoków warunkowych

w procesorach 8086÷80286

- **tylko 16-bitowe segmenty, przemieszczenie 1 bajt,**
- **rozmiar rozkazu - 2 bajty**

począwszy od 80386 dodatkowo:

- **w segmentach 16-bitowych,**
  - **przemieszczenie 2 bajty,**
  - **rozmiar rozkazu 4 bajty**
- **w segmentach 32-bitowych,**
  - **przemieszczenie 4 bajty,**
  - **rozmiar rozkazu 6 bajtów**

JMP lab				Skok bezwarunkowy				
O	D	I	T	S	Z	A	P	C
-	-	-	-	-	-	-	-	-

Skoki warunkowe (dla liczb ze znakiem):

jcc etykieta

cc = angielski skrót nazwy relacji

Relacja	Nazwa	Mnemonika
=	equal / zero	je / jz
<>	not equal / not zero	jne / jnz
<	less / not greater or equal	jl / jnge
<=	less or equal / not greater	jle / jng
>	greater / not less or equal	jg / jnle
>=	greater or equal / not less	jge / jnl

JE lab/JZ lab				skocs jeśli ZF=1				
O	D	I	T	S	Z	A	P	C
-	-	-	-	-	-	-	-	-

JNE lab/JNZ lab				skocs jeśli ZF=0				
O	D	I	T	S	Z	A	P	C
-	-	-	-	-	-	-	-	-

JLE lab/JNG lab				skocs jeśli ZF=1 lub SF<>OF				
O	D	I	T	S	Z	A	P	C
-	-	-	-	-	-	-	-	-

JG lab/JNLE lab				skocs jeśli ZF=0 i SF=OF				
O	D	I	T	S	Z	A	P	C
-	-	-	-	-	-	-	-	-

JL lab/JNGE lab				skocs jeśli SF<>OF				
O	D	I	T	S	Z	A	P	C
-	-	-	-	-	-	-	-	-

(*) gwiazdką oznaczone są zadania, które nie są realizowane na ćwiczeniach i są przeznaczone do wykonania jako zadania domowe.

ADD arg1,arg2				dodaj (arg1+=arg2)				
O	D	I	T	S	Z	A	P	C
*	-	-	-	*	*	*	*	*

```
cs:0100 80C406          add ah,06
cs:0103 050600          add ax,0006
cs:0106 660506000000    add eax,00000006
cs:010C 02E7            add ah,bh
cs:010E 03C3            add ax,bx
cs:0110 6603C3          add eax,ebx
```

SUB arg1,arg2				odejmij (arg1-=arg2)				
O	D	I	T	S	Z	A	P	C
*	-	-	-	*	*	*	*	*

```
cs:0100 80EC06          sub ah,06
cs:0103 2D0600          sub ax,0006
cs:0106 662D06000000    sub eax,00000006
cs:010C 2AE7            sub ah,bh
cs:010E 2BC3            sub ax,bx
cs:0110 662BC3          sub eax,ebx

cs:0100 662BC0          sub eax,eax
```


NEG arg				zaneguj (arg=-arg)				
O	D	I	T	S	Z	A	P	C
*	-	-	-	*	*	*	*	*

```
cs:0100 F6DC            neg ah
cs:0102 F7D8            neg ax
cs:0104 66F7D8          neg eax
```

DEC arg				zmniejsz o 1 (arg--)				
O	D	I	T	S	Z	A	P	C
*	-	-	-	*	*	*	*	-

```
cs:0100 FECC            dec ah
cs:0102 48              dec ax
cs:0103 6648            dec eax
```

INC arg				zwieksz o 1 (arg++)				
O	D	I	T	S	Z	A	P	C
*	-	-	-	*	*	*	*	-

```
cs:0100 FEC4            inc ah
cs:0102 40              inc ax
cs:0103 6640            inc eax
```


imul arg				mnożenie całkowite ze znakiem (format 1-operandowy)				
O	D	I	T	S	Z	A	P	C
*	-	-	-	?	?	?	?	*

Rozmiar operandu	Mnożna	Mnożnik	Wynik
16b/8b	AL	r/m8	AX
32b/16b	AX	r/m16	DX:AX
64b/32b	EAX	r/m32	EDX:EAX

```
cs:0100 66B805000000 mov eax,5    ; mnożna
cs:0106 66B902000000 mov ecx,2    ; mnożnik
cs:010C 66F7E9         imul ecx
                     ; rezultat: EDX = 0, EAX = A
```

Uwaga:
- dla rozkazu *imul* dostępne są jeszcze rozszerzone formaty 2- i 3-operandowe (nie ma ich dla mnożenia bez znaku - rozkaz *mul*)

idiv arg				dzielenie całkowite ze znakiem				
O	D	I	T	S	Z	A	P	C
?	-	-	-	?	?	?	?	?

Rozmiar operandu	Dzielna	Dzielnik	Wynik	Reszta
16b/8b	AX	r/m8	AL	AH
32b/16b	DX:AX	r/m16	AX	DX
64b/32b	EDX:EAX	r/m32	EAX	EDX

```
cs:0100 66B804000000 mov eax,5
                       ; młodsza część dzielnej
cs:0106 66BA00000000 mov edx,0
                       ; starsza część dzielnej
cs:010C 66B902000000 mov ecx,2 ; dzielnik
cs:0112 66F7F9         idiv ecx
          ; rezultat: część całkowita EAX = 2,
                       reszta          EDX = 1
```


Największy wspólny podzielnik (Euklides)

a(x) = b(x) = NWP(a(x),b(x))

C:
```
if(a<0)a=-a;
if(b<0)b=-b;
while(a!=b)
  if(a>b)a-=b;
  else b-=a;
```

ASM:
```
cs:0100 3D0000          cmp ax,0000
cs:0103 7D02            jnl 0107
cs:0105 F7D8            neg ax
cs:0107 83FB00          cmp bx,0000
cs:010A 7D02            jnl 010E
cs:010C F7DB            neg bx
cs:010E 3BC3            cmp ax,bx
cs:0110 740C            je  011E
cs:0112 3BC3            cmp ax,bx
cs:0114 7E04            jle 011A
cs:0116 2BC3            sub ax,bx
cs:0118 EB02            jmp 011C
cs:011A 2BD8            sub bx,ax
cs:011C EBF0            jmp 010E
cs:011E
```

Powyższe i dodatkowe materiały znajdują się na stronie dr. inż. W. Complaka

1. http://www.cs.put.poznan.pl/wcomplak/BFILES/PN_W_1.PDF
2. http://www.cs.put.poznan.pl/wcomplak/BFILES/PN_W_2.PDF
3. http://www.cs.put.poznan.pl/wcomplak/BFILES/PN_W_3.PDF
4. http://www.cs.put.poznan.pl/wcomplak/BFILES/PN_W_4.PDF
5. http://www.cs.put.poznan.pl/wcomplak/BFILES/PN_W_5.PDF

(*) gwiazdką oznaczone są zadania, które nie są realizowane na ćwiczeniach i są przeznaczone do wykonania jako zadania domowe.

6. http://www.cs.put.poznan.pl/wcomplak/BFILES/PN_W_6.PDF
7. http://www.cs.put.poznan.pl/wcomplak/BFILES/PN_W_7.PDF

Zadania

Zad. 1(*). Skompiluj i wykonaj programy prezentowane na wykładzie.

Zad. 2. Zaznacz rzeczywiste rejestry procesora x86:
a) AX, b) FX, c) BX, d) GX, e) CX

Zad. 3. Zaznacz poprawne odpowiedzi. Instrukcja `add p, z`:
a) pozwala wykonywać operację dodawania na liczbach znajdujących się w odpowiednich rejestrach,
b) po wykonaniu operacji wynik znajduje się w rejestrze p,
c) po wykonaniu operacji wynik znajduje się w rejestrze z,
d) pozwala wykonywać operację mnożenia na liczbach znajdujących się w odpowiednich rejestrach.

Zad. 4. Jaka instrukcja kończy działanie programu napisanego w języku asemblera?

Zad. 5. Zaznacz poprawne odpowiedzi:
a) narzędzia wykorzystywane w procesie kompilacji to tylko kompilator,
b) narzędzia wykorzystywane w procesie kompilacji to tylko linker,
c) narzędzia wykorzystywane w procesie kompilacji to kompilator i linker,
d) proces kompilacji przebiega w następujący sposób: program źródłowy jest linkowany, a następnie kompilowany,
e) proces kompilacji przebiega w następujący sposób: program źródłowy jest kompilowany, a następnie linkowany,
f) kompilator generuje kod wynikowy,
g) linker generuje kod wynikowy,
h) kompilator generuje kod przejściowy,
i) linker generuje kod przejściowy.

Zad. 6. Zaznacz poprawne odpowiedzi:
a) narzędzie do testowania kodu wynikowego to masm,
b) narzędzie do testowania kodu wynikowego to debug,
c) narzędzie do testowania kodu wynikowego to link,
d) liczby w rejestrach procesora są przechowywane w postaci binarnej,
e) liczby w rejestrach procesora są przechowywane w postaci dziesiętnej,
j) liczby w rejestrach procesora są przechowywane w postaci heksadecymalnej.

Zad. 7. Załóżmy, że kompilowany jest plik źródłowy zawierający program napisany w języku asemblera. Uzupełnij brakujące informacje:
a) w wyniku procesu kompilacji można uzyskać następujące pliki:
   - kod przejściowy przechowywany w pliku o rozszerzeniu ……
   - kod wynikowy przechowywany w pliku o rozszerzeniu ……...
   - listing kompilacji przechowywany w pliku o rozszerzeniu ……...
   - mapę kompilacji przechowywaną w pliku o rozszerzeniu ……...
b) w celu uzyskania informacji dotyczących możliwych opcji narzędzia *ml*, które mogą zostać wykorzystane podczas kompilacji kodu źródłowego napisanego w języku asemblera należy wykonać następujące polecenie: `ml` ……
c) w celu uzyskania informacji dotyczących możliwych opcji narzędzia *debug*, które mogą zostać wykorzystane podczas testowania poprawności kodu wynikowego uzyskanego w wyniku kompilacji należy wykonać następujące polecenie: `-`……
d) …….. – przykładowa polecenie uruchamia program *debug* w celu testowania poprawności kodu źródłowego *prog.exe*

(*) gwiazdką oznaczone są zadania, które nie są realizowane na ćwiczeniach i są przeznaczone do wykonania jako zadania domowe.

e) jakie będą rezultaty wykonania poniższych operacji:

- ```
  -rax
  AX 0000
  :1
  ```

.....

- ```
  -rax
  AX 0003
  :
  ```

.....

- ```
  -g
  ```

.....

- ```
  -q
  ```

.....

Zad. 8. Oblicz metodą **bezpośredniego dodawania** wartości następujących wyrażeń (liczby są podane w systemie **heksadecymalnym**, czyli szesnastkowym; wynik ma być również heksadecymalny)(*4-6):

524	735	345	2A8	E45	CCE
+ 214	+ 879	+ CBA	+ 34C	+ 28B	+ ABA

Zad. 9. Oblicz metodą **bezpośredniego odejmowania** wartości następujących wyrażeń (liczby są podane w systemie **heksadecymalnym**; wynik ma być również heksadecymalny) (*4-6):

548	EAB	189	437	1F2	E49
- 211	- 341	- 67	- 78	- BC	- AC8

Zad. 10. Do czego służy kod uzupełnień do dwóch (U2) oraz jaka jest jego charakterystyczna cecha?

Zad. 11. Podaj **zakres** liczb całkowitych, jakie można zakodować za pomocą **5 bitów** metodą uzupełnienia do 2.

Zad. 12. Podaj wartość bezwzględną (w systemie heksadecymalnym) następujących liczb całkowitych zakodowanych na 16 bitach metodą uzupełnienia do 2 (*4-6):

FFFE	FFF7	FFD3	FEF9	F30E	47EB

Zad. 13. Zaznacz poprawne odpowiedzi. Zmiana znaku podczas tworzenia liczby ujemnej, przy wykorzystaniu jej dodatniego odpowiednika, jest wykonywana poprzez:

a) zanegowanie bitów i dodanie 1,
b) dodanie jedynki i zanegowanie bitów.

Zad. 14. Przedstaw podane liczby ujemne w uzupełnieniu do 2 na szesnastu bitach i w kodzie heksadecymalnym (*4-6):

-4	-B	-E	-23	-C4	-A8

Zad. 15. Zaznacz poprawne odpowiedzi. Instrukcja `sub p, z`:

a) pozwala wykonywać operację dodawania na liczbach znajdujących się w odpowiednich rejestrach,
b) wynik po wykonaniu operacji znajduje się w rejestrze p,
c) wynik po wykonaniu operacji znajduje się w rejestrze z,
d) pozwala wykonywać operację odejmowania na liczbach znajdujących się w odpowiednich rejestrach.

Zad. 16. Zaznacz poprawne odpowiedzi. Po wykonaniu instrukcji `neg c`:

a) w rejestrze c znajduje się wartość taka sama jak przed wykonaniem operacji,
b) w rejestrze c znajduje się wartość powstała poprzez zanegowanie postaci binarnej liczby c.

(*) gwiazdką oznaczone są zadania, które nie są realizowane na ćwiczeniach i są przeznaczone do wykonania jako zadania domowe.

Zad. 17. Procesor wykonuje rozkazy zapisane w kodzie wynikowym programu w następujący sposób:
a) ……,
b) licznik rozkazów to …….., który wskazuje na …….

Zad. 18. Skoki warunkowe są realizowane za pomocą operacji porównania …….., i operacji skoku. Podaj instrukcje asemblerowe odpowiadające następującym operacjom skoku:
a) **j**ump if **e**qual - ……..
b) **j**ump if **n**ot **l**ess - ……..
c) **j**ump if **g**reater - ……
d) **j**ump if **n**ot **g**reater - …….
e) **j**ump if **l**ess - …….

Zad. 19. Zapisz w języku asemblera pętle **while** oraz instrukcję warunkową **if-else**.

Zad. 20. Napisz program przesyłający do rejestru AX **najmniejszą z liczb** znajdujących się w rejestrach BX, CX i DX. Oto przykładowe przypadki testowe:

Dane wejściowe (hex)			Wynik
BX	**CX**	**DX**	**AX**
1	2	3	1
5	FF	FFFF	FFFF

Zad. 21. Napisz program przesyłający do rejestru AX **resztę z dzielenia** liczby naturalnej znajdującej się w rejestrze BX przez liczbę dodatnią znajdującą się w rejestrze CX. Zastosuj **metodę wielokrotnego odejmowania**. Oto przykładowe przypadki testowe przedstawione w systemie dziesiętnym:

Dane wejściowe		Wynik
BX	**CX**	**AX**
6	5	1
16	5	2
4	5	4
5	5	0
F	4	3

Zad. 22. Napisz program obliczający wartość funkcji n!. Przyjmij, że *n* jest w rejestrze BX, a wynik ma być w rejestrze AX. Do mnożenia wykorzystaj instrukcję

```
imul s
```

która przesyła do pary rejestrów DX:AX wynik mnożenia liczby znajdującej się w rejestrze AX przez liczbę znajdującą się w rejestrze (lub innym miejscu) określonym przez operand *s*. Jeżeli wynik mieści się na 16 bitach i jest **nieujemny**, to wartość rejestru DX jest równa 0 (jeśli wynik mieści się na 16 bitach i jest **ujemny**, to wartością rejestru jest FFFF). Na przykład instrukcja

```
imul bx
```

może być opisana następującą instrukcją w języku C:

```
ax = ax * bx;
```

Oto przykładowe dane testowe:

Wejście: n	Wynik: n!
BX	**AX**
0	1
1	1
2	2
3	6
4	24

- *Jaka jest największa wartość n, dla której n! może być poprawnie przedstawiona w uzupełnieniu do 2 na 16 bitach?*

(*) gwiazdką oznaczone są zadania, które nie są realizowane na ćwiczeniach i są przeznaczone do wykonania jako zadania domowe.

- *Jak należałoby zmodyfikować ten program, gdyby osoby korzystające z tego programu doszły do wniosku, że będzie im wygodniej, gdy n będzie podane w rejestrze ax, a wynik będzie w rejestrze bx?*

Zad. 23. Napisz program obliczania **liczby cyfr** dziesiętnych podanej liczby *n*. Skorzystaj z instrukcji

```
idiv s
```

która dzieli zawartość pary rejestrów DX:AX przez zawartość *s* i iloraz przesyła do rejestru AX, zaś resztę do rejestru DX. W przypadku, gdy dzielna jest 16-bitowa należy pamiętać o wyzerowaniu rejestru DX. Na przykład instrukcja języka C

```
ax = ax / bx;
```

mogłaby być przetłumaczona w następujący sposób:

```
mov dx, 0
idiv bx
```

Oto przykładowe dane testowe:

Wejście: n	Wynik:
DX	CX
D	2

Zad. 24(*). Napisz program obliczania **wartości a^n**, gdzie *a* i *n* są liczbami naturalnymi (a>0, *n* >= *0*). Skorzystaj z instrukcji

```
imul s
```

która przesyła do pary rejestrów DX:AX wynik mnożenia liczby znajdującej się w rejestrze AX przez liczbę znajdującą się w rejestrze (lub innym miejscu) określonym przez operand *s*. Jeżeli wynik mieści się na 16 bitach i jest **nieujemny**, to wartość rejestru DX jest równa 0 (jeśli wynik mieści się na 16 bitach i jest **ujemny**, to wartością rejestru jest FFFF). Na przykład instrukcja

```
imul bx
```

może być opisana następującą instrukcją w języku C:

```
ax = ax * bx;
```

Oto przykładowe dane testowe:

Wejście: n	Wynik: a^n	Wejście: a
BX	AX	CX
3	8	2

Zad. 25(*). Napisz program obliczania **sumy cyfr** dziesiętnych podanej liczby *n*. Skorzystaj z instrukcji

```
idiv s
```

która dzieli zawartość pary rejestrów DX:AX przez zawartość *s* i iloraz przesyła do rejestru AX, zaś resztę do rejestru DX. W przypadku, gdy dzielna jest 16-bitowa należy pamiętać o wyzerowaniu rejestru DX. Na przykład instrukcja języka C

```
ax = ax / bx;
```

mogłaby być przetłumaczona w następujący sposób:

```
mov  dx, 0
idiv bx
```

Oto przykładowe dane testowe:

Wejście: n	Wynik:
DX	CX
D	4

(*) gwiazdką oznaczone są zadania, które nie są realizowane na ćwiczeniach i są przeznaczone do wykonania jako zadania domowe.