Rejestry

8-bit	AL/AH	CL/CH	DL/DH	BL/BH	SPL	BPL	SIL	DIL	R8B-R15B
16-bit	AX	CX	DX	BX	SP	BP	SI	DI	R8W-R15W
32-bit	EAX	ECX	EDX	EBX	ESP	EBP	ESI	EDI	R8D-R15D
64-bit	RAX	RCX	RDX	RBX	RSP	RBP	RSI	RDI	R8-R15

Funkcje – rejestry całkowite

	32-bit	64-bit
argumenty	stos	RDI, RSI, RDX, RCX, R8, R9, stos
wynik	EAX	RAX, RDX
trzeba zachować	EBX, ESI, EDI, EBP, ESP	RBX, RBP, R12–R15, RSP
można zmienić	EAX, ECX, EDX	RAX, RCX, RDX, RSI, RDI, R8–R11

Funkcje – rejestry zmiennoprzecinkowe

	32-bit	64-bit
argumenty	XMM0/YMM0	float: XMM0–XMM7/YMM0–YMM7 long double: stos
wynik	XMM0/YMM0, ST(0) (?)	float: XMM0–XMM1/YMM0–YMM1 long double: ST(0)–ST(1)
trzeba zachować		
można zmienić	ST(0)–ST(7), XMM0–XMM7/YMM0–YMM7	ST(0)–ST(7), XMM0–XMM15/YMM0–YMM15

Funkcje – prolog-epilog

```
1  global  _myfunc
2
3  _myfunc:
4          push    ebp
5          mov     ebp, esp
6
7          sub     esp, 0x40          ; 64 bytes of local stack space
8          mov     ebx, [ebp+8]       ; first parameter to function
9
10         ; some more code
11
12         leave                      ; mov esp, ebp / pop ebp
13         ret
```

Funkcje – układ stosu

```
:
| [ebp + 16] (3rd function argument)
| [ebp + 12] (2nd argument)
| [ebp + 8]  (1st argument)
| [ebp + 4]  (return address)
| [ebp]      (old ebp value)
| [ebp - 4]  (1st local variable)
:

| [ebp - X]  (== esp - the current stack pointer. The use of push / pop is
    valid now)
```

Skoki warunkowe

	bez znaku	ze znakiem
$=$	JE/JZ	JE/JZ
$\neq$	JNE/JNZ	JNE/JNZ
$\geqslant$	JAE/JNB	JGE/JNL
$>$	JA/JNBE	JG/JNLE
$\leqslant$	JBE/JNA	JLE/JNG
$<$	JB/JNAE	JL/JNGE

Rozmiary

	rozmiar	zainicjalizowane	niezainicjalizowane
8 bit	byte	db	resb
16 bit	word	dw	resw
32 bit	dword	dd	resd
64 bit	qword	dq	resq
80 bit	tword	dt	rest
128 bit	oword	do	reso
256 bit	yword	dy	resy
512 bit	zword	dz	resz

Dane

```
1 buf:    times 64  db 0
2 buflen: equ  $-buf
```

Wybrane instrukcje

SAL, SAR arithmetic shift

SHL, SHR logical shift

CWD, CDQ, CQO sign extension: AX/EAX/RAX → *DX:*AX

MOVSX, MOVZX move data with sign / zero extend

CMOVcc conditional move

IDIV EDX:EAX is divided by the given operand; result: EAX – quotient, EDX – remainder

LOOP decrements *CX, and if the counter does not become zero as a result of this operation, it jumps to the given label.