**REJESTRY:**

Ogólnego przeznaczenia:

- **AX** (ang. Accumulator) - jest wykorzystywany głównie do operacji arytmetycznych i logicznych
- **BX** (ang. Base Registers) - rejestr bazowy, głównie wykorzystywany przy adresowaniu pamięci
- **CX** (ang. Counter Registers) – rejestr często wykorzystywany jako licznik, np. przy instrukcji LOOP.
- **DX** (ang. Data Register) - rejestr danych, wykorzystywany przy operacjach mnożenia i dzielenia, a także do wysyłania i odbierania danych z portów.

Indeksowe:

- **SI** (ang. Source Index) - rejestr indeksujący pamięć, wskazuje obszar z którego przesyłane są dane. W połączeniu z DS tworzy adres logiczny DS:SI
- **DI** (ang. Destination Index) - rejestr indeksujący pamięć, wskazuje obszar, do którego przesyłane są dane. W połączeniu z ES, tworzy adres logiczny ES:DI

Wskaźnikowe:

- **BP** (ang. Base Pointer) - rejestr stosowany do adresowania pamięci.
- **SP** (ang. Stack Pointer) - wskaźnik stosu.

Instrukcji:

- **IP** (ang. Instruction Pointer) – zawiera adres aktualnie wykonywanej instrukcji, może być modyfikowany przez rozkazy sterujące pracą programu

- **FLAGS** – rejestr znaczników (flagi)

Segmentowe:

- **CS** (ang. Code Segment) - rejestr informujący o segmencie aktualnie wykonywanego rozkazu. Razem z IP tworzy adres logiczny CS:IP kolejnej instrukcji.
- **DS** (ang. Data Segment) - rejestr informujący o segmencie z danymi.
- **ES** (ang. Extra Segment) - rejestr informujący o segmencie dodatkowym np. przy operacjach przesyłania łańcuchów.
- **SS** (ang. Stack Segment) - rejestr informujący o segmencie stosu.
- **FS**
- **GS**

**Rejestry MMX -** Działają na nich instrukcje całkowitoliczbowe SIMD Wykorzystują rejestry koprocesora

**Rejestry XMM** - Działają na nich instrukcje zmiennoprzecinkowe SIMD

**AVX - Advanced Vector eXtensions - Rejestry ymm** - działają na nich instrukcje zmiennoprzecinkowe SIMD - AVX

**AVX512 - Advanced Vector eXtensions - Rejestry zmm** – tylko w wybranych procesorach

**bajt** – 8b
**słowo** – 16b
**podwójne słowo** – 32b
**poczwórne słowo** – 64b

## TRYBY ADRESOWANIA:

**rejestrowy** - Argumentem instrukcji jest rejestr
**prosty - natychmiastowy** - Argumentem instrukcji jest wartość
**bezpośredni** - Argumentem instrukcji jest adres w pamięci (wskaźnik)
**pośredni - rejestrowy** - Argumentem instrukcji jest rejestr – wskaźnik (mov al, [rcx] )
**pośredni - bazowy** - Argumentem instrukcji jest wskaźnik
**pośredni - indeksowy** - Argumentem instrukcji jest rejestr – wskaźnik
**pośredni – bazowo - indeksowy** - Argumentem instrukcji jest wskaźnik

## INSTRUKCJE:

### Przesyłania:

**XCHG** - Zamienia zawartość źródła i celu.
**BSWAP** - Zamienia bajty w argumencie
**XADD** - Zamienia zawartość źródła i celu (8/16/32/64 bity), a ich sumę umieszcza w miejscu przeznaczenia (cel).
**CMPXCHG** - służy do porównania wartości dwóch operandów oraz wykonania operacji wymiany (exchange) na podstawie wyniku porównania.
**CMPXCHG8(16)B** - jest specyficzna dla architektury x86-64 i jest używana do porównywania i wymiany wartości 64-bitowych (8 bajtów) w atomowy sposób.
**PUSH** - Przesyła zawartość argumentu na stos.
**POP** - Przesyła zawartość stosu do celu.
**PUSHF/PUSHFD/PUSHFQ** - Przesyła zawartość Flag/Eflag/Rflag na stos
**POPF**- analogicznie
**PUSHA/PUSHAD** - Przesyła zawartość di, si, bp, bx, dx, cx, ax / edi, esi, ebp, ebx, edx, ecx, eax na stos.
**POPA** - analogicznie
**CWD/CDQ/CQO** - Konwertuje z zachowaniem znaku word na doubleword / doubleword na quadword / quadword na octaword (ax na dx:ax, eax na edx:eax, rax na rdx:rax).
**CBW/CWDE/CDQE** - Konwertuje byte (AL) na word(AX) / word(AX) na doubleword (EAX) / doubleword (EAX) na quadword (RAX) z uwzględnieniem znaku.
**MOVSX/MOVSXD** - Przesyła zawartość źródła do rejestru celu z uwzględnieniem znaku. Cel posiada 2/4/8 razy więcej bitów.
**MOVZX** - Przesyła zawartość źródła do rejestru celu z dopisaniem na starszych bitach zer. Cel posiada 2/4/8 razy więcej bitów. Źródło 8/16 bitów.

### Arytmetyczne:

**ADD** - dodawanie całkowitoliczbowe
**ADC** - dodawanie z przeniesieniem
**ADCX** - dodawanie z przeniesieniem bez znaku
**ADOX** - dodawanie z przeniesieniem bez znaku
**SUB** - odejmowanie
**SBB** - odejmowanie z pożyczką
**MUL** - mnożenie bez znaku
**MULX** - mnożenie bez znaku
**IMUL** - mnożenie ze znakiem
**DIV** - dzielenie bez znaku
**IDIV** - dzielenie ze znakiem
**INC** - inkrementacja (zwiększenie)
**DEC** - dekrementacja (zmniejszenie)
**NEG** - zmiana znaku
**CMP** - porównanie

### Arytmetyczne BCD:

**DAA -** Korekta upakowanego kodu BCD (w AL) po dodawaniu. Polega na dodaniu 6 najpierw do młodszego półbajta, a potem do starszego, jeśli ich zawartości były większe od 9 lub wystąpiło przeniesienie AF (CF)
**DAS -** Korekta upakowanego kodu BCD (w AL) po odejmowaniu. Polega na odjęciu 6 najpierw od młodszego półbajta, a potem od starszego, jeśli ich zawartości były większe od 9 lub wystąpiło przeniesienie AF (CF).

**AAA -** Korekta nieupakowanego kodu BCD (w AL) po dodawaniu. Polega na dodaniu 6 do młodszego półbajta i 1 do AH, jeśli zawartość AL była większa od 9 lub wystąpiło przeniesienie AF.
**AAS -** Korekta nieupakowanego kodu BCD (w AL) po odejmowaniu. Polega na odjęciu 6 od młodszego półbajta i 1 od AH, jeśli zawartość AL. była większa od 9 lub wystąpiło przeniesienie AF.
**AAM -** Korekta nieupakowanego kodu BCD (w AX) po mnożeniu. Polega na jednoczesnym wykonaniu:
AH=AL div 10
AL:=AL Mod 10
**AAD -** Korekta nieupakowanego kodu BCD (w AX) przed dzieleniem. Polega na jednoczesnym wykonaniu:
AL:=AH*10+AL
AH=o

**Logiczne:**

**AND** bitowa funkcja **AND** (bit po bicie)
**ANDN** bitowa funkcja **AND** z negacją (bit po bicie)
**OR** bitowa funkcja **OR** (bit po bicie)
**XOR** bitowa funkcja **XOR** (bit po bicie)
**NOT** bitowa funkcja **NOT** (bit po bicie)

**Przesunięcia i rotacji:**

**SAR -** Przesunięcie arytmetyczne celu w prawo o ile bitów
**SHR -** Przesunięcie logiczne w prawo o ile bitów
**SAL -** Przesunięcie arytmetyczne celu w lewo o ile bitów
**SHL -** Przesunięcie logiczne celu w lewo o ile bitów
**SARX -** Przesunięcie arytmetyczne źródła w prawo o ile bitów i zapisanie w celu. Cel i ile są rejestrami 32|64 bitowymi
**SHRX -** Przesunięcie logiczne źródła w prawo o ile bitów i zapisanie w celu. Cel i ile są rejestrami 32|64 bitowymi
**SHLX -** Przesunięcie logiczne źródła w lewo ile bitów i zapisanie w celu. Cel i ile są rejestrami 32| 64 bitowymi
**SHRD -** Przesunięcie źródła:celu w prawo o ile bitów. Ile=cl lub wartość 0-31|64. Rejestr źródła (16,32,64) pozostaje bez zmian
**SHLD -**Przesunięcie źródła:celu w lewo o ile bitów. Ile=cl lub wartość 0-31|63. Rejestr źródła (16,32,64) pozostaje bez zmian
**ROR -** Rotacja (obrót) celu w prawo o ile bitów. Ile=1, cl lub wartość 0-31|63
**ROL -** Rotacja (obrót) celu w lewo o ile bitów. Ile=1, cl lub wartość 0-31|63
**RCR -** Rotacja (obrót) przez przeniesienie celu w prawo o ile bitów. Ile=1, cl lub wartość 0-31|63
**RCL -** Rotacja (obrót) przez przeniesienie celu w lewo o ile bitów. Ile=1, cl lub wartość 0-31|63
**RORX -** Rotacja źródła w prawo o ile bitów i zapisanie w celu. Cel jest rejestrem 32|64 bitowym. Ile przyjmuje wartość 0-31|63

**Warunkowe i skoku:**

Warunki dotyczące flag:

| | | |
|---|---|---|
| E/Z | equal/ zero | ZF=1 |
| NE/NZ | not equal/ not zero | ZF=0 |
| C | carry | CF=1 |
| NC | not carry | CF=0 |
| O | overflow | OF=1 |
| NO | not overflow | OF=0 |
| S | sign (negative) | SF=1 |
| NS | not sign (non-negative) | SF=0 |
| P/PE | parity/ parity even | PF=1 |
| NP/PO | not parity/ parity odd | PF=0 |

**CMOVcc** - Jeśli jest spełniony warunek cc, przesyła źródło do miejsca przeznaczenia (rejestr 16, 32 lub 64 bitowy). Instrukcja wprowadzona w procesorach rodziny P6!

**CMOVE/CMOVZ** - Prześlij jeżeli equal/zero
**CMOVNE/CMOVNZ** - Prześlij jeżeli not equal/ not zero
**CMOVA/CMOVNBE** - Prześlij jeżeli above/ not below or equal
**CMOVAE/CMOVNB** - Prześlij jeżeli above or equal/ not below
**CMOVB/CMOVNAE** - Prześlij jeżeli below/ not above or equal
**CMOVBE/CMOVNA** - Prześlij jeżeli below or equal/ not above
**CMOVG/CMOVNLE** - Prześlij jeżeli greater/ not less or equal
**CMOVGE/CMOVNL** - Prześlij jeżeli greater or equal/ not less
**CMOVL/CMOVNGE** - Prześlij jeżeli less/ not greater or equal
**CMOVLE/CMOVNG** - Prześlij jeżeli less or equal/ not greater
**CMOVC** - Prześlij jeżeli carry
**CMOVNC** - Prześlij jeżeli not carry
**CMOVO** - Prześlij jeżeli overflow
**CMOVNO** - Prześlij jeżeli not overflow
**CMOVS** - Prześlij jeżeli sign (negative)
**CMOVNS** - Prześlij jeżeli not sign (non-negative)
**CMOVP/CMOVPE** - Prześlij jeżeli parity/ parity even
**CMOVNP/CMOVPO** - Prześlij jeżeli not parity/ parity odd

**Jcc** – Skoki warunkowe (cc – warunek)

**JE/JZ** - Skocz jeśli equal/zero
**JNE/JNZ** - Skocz jeśli not equal/not zero
**JA/JNBE** - Skocz jeśli above/not below or equal
**JAE/JNB** - Skocz jeśli above or equal/not below
**JB/JNAE** - Skocz jeśli below/not above or equal
**JBE/JNA** - Skocz jeśli below or equal/not above
**JG/JNLE** - Skocz jeśli greater/not less or equal
**JGE/JNL** - Skocz jeśli greater or equal/not less
**JL/JNGE** - Skocz jeśli less/not greater or equal
**JLE/JNG** - Skocz jeśli less or equal/not greater

**JC** - Skocz jeśli carry
**JNC** - Skocz jeśli not carry
**JO** - Skocz jeśli overflow
**JNO** - Skocz jeśli not overflow
**JS** - Skocz jeśli sign (negative)
**JNS** - Skocz jeśli not sign (non-negative)
**JPO/JNP -** Skocz jeśli parity odd/not parity
**JPE**/**JP** - Skocz jeśli parity even/parity

**Sterujące przebiegiem programu:**

**JMP -** Skok bezwarunkowy (etykieta)
**JCXZ/JECXZ/JRCX -** Skok jeśli zero w rejestrze CX/ECX/RCX (do etykiety)
**LOOP** - Pętla z licznikiem CX/ECX/RCX. Zmniejsza CX/ECX/RCX o 1 i jeśli nie uzyskano zera przeskakuje do podanej etykiety.
**LOOPZ/LOOPE** - Pętla z licznikiem CX/ECX/RCX i zero/equal. Zmniejsza CX/ECX/RCX o 1 i jeśli nie uzyskano zera w CX/ECX/RCX i flaga ZF=1 przeskakuje do podanej etykiety.
**LOOPNZ/LOOPNE -** Pętla z licznikiem CX/ECX/RCX i not zero/not equal. Zmniejsza CX/ECX/RCX o 1 i jeśli nie uzyskano zera i flaga ZF=0 przeskakuje do podanej etykiety .
**CALL** - Wywołanie podprogramu. wysyła na stos adres powrotu EIP|RIP lub CS:EIP|RIP. Parametr adres wpisuje do EIP|RIP lub CS:EIP|RIP.
**RET** - Powrót z podprogramu. Jeśli posiada parametr ile, to dodatkowo usuwa ze stosu ile bajtów.
**IRET** - Powrót z podprogramu obsługi przerwania. Instrukcja iret/iretd/iretq wraca z podprogramu obsługi przerwania, pobiera ze stosu adres powrotu do CS:EIP|RIP oraz flagi zachowane przy wywołaniu przerwania.
**INT** - Przerwanie programowe. Instrukcja int działa podobnie do instrukcji call, jednak dodatkowo wysyła na stos flagi i wchodząc do podprogramu część z nich zeruje.
**INTO** - Przerwanie przy przekroczeniu zakresu. Wywołuje przerwanie programowe (4) w przypadku ustawienia flagi OF (nadmiaru).
**BOUND** - Sprawdza, czy indeks tablicy (wartość 16/32 bitowa ze znakiem) zawarty w rejestrze idx nie przekracza jej granic określonych przez strukturę w pamięci gr złożoną z granicy dolnej i górnej. W przypadku przekroczenia granicy generowany jest wyjątek przekroczenia granicy tablicy.
**ENTER** - Tworzy ramę stosu w podprogramie z uwzględnieniem poziomu zagnieżdżenia podprogramów lokalnych (level) i rozmiaru w bajtach zmiennych lokalnych (storage). Na stosie umieszcza wskaźniki ram stosu: podprogramu wywołującego, wszystkich poziomów nadrzędnych i własny.
**LEAVE** - Usuwa ramę stosu (utworzoną instrukcją ENTER) przed wyjściem z podprogramu. Przypisuje do wskaźnika stosu rejestr bazowy, następnie zdejmuje rej. basowy ze stosu.

**Operacje na znacznikach, bitach i bajtach:**

Flagi:

**STC** – Ustawienie flagi CF
**CLC** - Zerowanie flagi CF
**CMC** - Zanegowanie flagi CF
**CLD** - Zerowanie DF – flagi kierunku
**STD** - Ustawienie DF – flagi kierunku
**LAHF** - Przesłanie flag do rejestru AH
**SAHF** - Przesłanie rejestru AH do flag. Bity 1,3,5 są ignorowane.
**PUSHF/PUSHFD/PUSHFQ** - Wysłanie zawartości flag na stos
**POPF/POPFD/POPFQ** – Pobranie zawartości flag ze stosu
**STI** - Ustawienie IF – flagi przerwań. Włącza po następnej instrukcji system przerwań maskowalnych.
**CLI** - Zerowanie IF. Wyłącza system przerwań maskowalnych.

Bity:

**BT** - Testowanie bitu. Wyznacza wartość bitu nr (rejestr lub wartość) w bazie (rejestr lub zmienna) i umieszcza ją w CF.
**BTS** - Testowanie bitu z ustawianiem. Wyznacza wartość bitu nr (rejestr lub wartość) w bazie (rejestr lub zmienna) i umieszcza ją w CF. Następnie ustawia badany bit.
**BTR** - Testowanie bitu z zerowaniem. Wyznacza wartość bitu nr (rejestr lub wartość) w bazie (rejestr lub zmienna) i umieszcza ją w CF. Następnie ustawia badany bit.
**BTC** - Testowanie bitu z negacją. Wyznacza wartość bitu nr (rejestr lub wartość) w bazie (rejestr lub zmienna) i umieszcza ją w CF. Następnie neguje badany bit.
**TEST** - Porównanie logiczne. Wyznacza iloczyn logiczny(bit po bicie) zawartości celu i źródła (rejestr lub wartość), wynik jest pominięty, ustawia flagi.
**BSF** - Przeszukiwanie bitów w przód. Szuka w rejestrze lub zmiennej źródła najmłodszego bitu=1, jego indeks umieszcza w rejestrze celu (ZF = 0).
**BSR** - Przeszukiwanie bitów wstecz. Szuka w rejestrze lub zmiennej źródła najstarszego bitu=1, jego indeks umieszcza w rejestrze celu (ZF = 0).
**LZCNT** - Zlicza zerowe bity od najstarszego
**TZCNT** - Zlicza zerowe bity od najmłodszego
**BEXTR** - Wycina ciąg bitów
**BLSI** - Kopiuje najmłodszy ustawiony bit
**BLSR** - Zeruje najmłodszy ustawiony bit
**BLSMSK** - Tworzy maskę do bitu=0
**BZHI** - Zeruje starsze bity

**SETcc** cel - Jeśli jest spełniony warunek cc, ustawia bajt na 1, w przeciwnym wypadku na 0.

**SETE/SETZ** - Ustaw bajt jeśli equal/ zero
**SETNE/SETNZ** - Ustaw bajt jeśli not equal/ not zero
**SETS** - Ustaw bajt jeśli sign (negative)
**SETNS** - Ustaw bajt jeśli not sign (non-negative)
**SETO** - Ustaw bajt jeśli overflow
**SETNO** - Ustaw bajt jeśli not overflow
**SETPE/SETP** - Ustaw bajt jeśli parity even/ parity
**SETPO/SETNP** - Ustaw bajt jeśli parity odd/ not parity
**SETA/SETNBE** - Ustaw bajt jeśli above/ not below or equal

**SETAE/SETNB/SETNC** - Ustaw bajt jeśli above or equal/ not below/ not carry
**SETB/SETNAE/SETC** - Ustaw bajt jeśli below/ not above or equal/ carry
**SETBE/SETNA** - Ustaw bajt jeśli below or equal/ not above
**SETG/SETNLE** - Ustaw bajt jeśli greater/ not less or equal
**SETGE/SETNL** - Ustaw bajt jeśli greater or equal/ not less
**SETL/SETNGE** - Ustaw bajt jeśli less/ not greater or equal
**SETLE/SETNG** - Ustaw bajt jeśli less or equal/ not greater

**RDPID** - Czyta 32-bitowy identyfikator procesora do rejestru celu.
**RDTSC** - Read Time Stamp Counter. Czyta 64-bitowy licznik do rejestrów EDX: EAX
**RDRAND** - Czyta 16|32|64-bitową liczbę losową (deterministic random bit generator) do rejestru celu wg normy NIST SP 800-90A. Jeśli CF = 1 wartość jest prawidłowa.
**RDSEED** - Czyta 32|64-bitową liczbę losową (non-deterministic random bit generator) do rejestru celu wg norm NIST SP 800-90B i NIST SP800-90C. Jeśli CF = 1 wartość jest prawidłowa.

## Operacje na łańcuchach:

**MOVS/MOVSB/MOVSW/MOVSD/MOVSQ** - Prześlij łańcuch/bajtów/słów/podwójnych słów/poczwórnych słów
**CMPS/CMPSB/CMPSW/CMPSD/CMPSQ** - Porównaj łańcuchy/bajtów/słów/podwójnych słów/poczwórnych słów
**SCAS/SCASB/SCASW/SCASD/SCASQ** - Skanuj łańcuch/bajtów/słów/podwójnych słów/poczwórnych słów
**LODS/LODSB/LODSW/LODSD/LODSQ** - Ładuj łańcuch/bajtów/słów/podwójnych słów/poczwórnych słów
**STOS/STOSB/STOSW/STOSD/STOSQ** - Zapamiętaj łańcuch/bajtów/słów/podwójnych słów/poczwórnych słów
**REP** - Powtarzaj dopóki ECX nie jest zerem
**REPE/REPZ** - Powtarzaj dopóki equal/zero
**REPNE/REPNZ** - Powtarzaj dopóki not equal/not zero

## Operacje na rejestrach segmentowych:

**LDS** - Załadowanie pełnego wskaźnika z użyciem DS. Wczytanie pełnego adresu źródła do pary rejestrów ds:cel(32).
**LES** - Załadowanie pełnego wskaźnika z użyciem ES, anal
**LFS** - Załadowanie pełnego wskaźnika z użyciem FS, anal
**LGS** - Załadowanie pełnego wskaźnika z użyciem GS, anal
**LSS** - Załadowanie pełnego wskaźnika z użyciem SS, anal

## Inne operacje:

**LOCK** - Powoduje niepodzielne wykonanie następnej instrukcji
**LEA** - Ładowanie adresu efektywnego
**NOP** - Nie wykonuje żadnego działania
**UD2** - Instrukcja niezdefiniowana. Generuje wyjątek instrukcja niezdefiniowana, nic nie robi, wprowadzona do testów.
**XLAT/XLATB** - Tłumaczenie w oparciu o tablicę translacji.
**MOVBE** - Przesłanie po zamianie kolejności bajtów. Jeden z argumentów musi być rejestrem (16, 32, 64).
**CPUID** - Identyfikacja procesora
**XGETBV** - Czyta do edx:eax zawartość rozszerzonego rejestru kontrolnego o indeksie ecx.