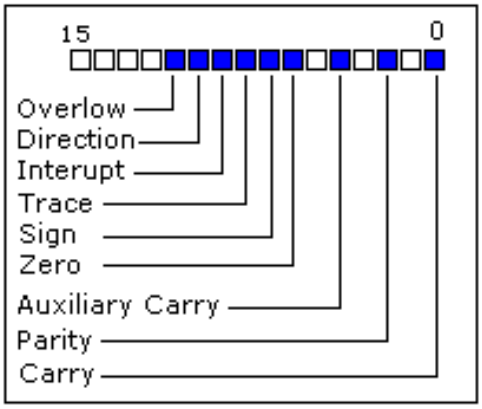
Instrukcje Arytmetyczne i Logiczne

Instrukcje arytmetyczne i logiczne działają na rejestr statusowy (rejestr znaczników –ang. **Flags**)



Rejestr ten ma 16 bitów, każdy nazywany jest znacznikiem (**flag)** i może przyjmować wartości **1** lub **0**.

- **Znacznik przeniesienia - Carry Flag (CF)** – ustawiany jest na **1** kiedy wystąpi przeniesienie dla liczb bez znaku, np. kiedy dla argumentów 8-bitowych dodamy **255 + 1**. Gdy nie ma przeniesienia bit ten jest ustawiony na **0**.

- **Znacznik zera - Zero Flag (ZF)** – ustawiany na **1** kiedy wynik jest **zero**. Dla wartości różnej od zera bit ten ustawiany jest na **0**.

- **Znacznik znaku - Sign Flag (SF)** – ustawiany na **1** gdy wynik jest **ujemny**. Gdy wynik jest **dodatni** – ustawiany jest na **0**. Znacznik ten przyjmuje wartość najbardziej znaczącego bitu wyniku.

- **Znacznik nadmiaru - Overflow Flag (OF)** – ustawiany na **1** kiedy wystąpi przepełnienie dla liczb ze znakiem, np. kiedy dodamy **100 + 50** (wynik nie jest w przedziale -128...127).

- **Znacznik parzystości - Parity Flag (PF)** – ustawiany jest na **1** kiedy wynik posiada parzystą liczbę jedynek, a na **0** kiedy liczba bitów o wartości jeden w wyniku jest nieparzysta. Uwaga: analizowane jest tylko 8 bitów wyniku nawet gdy wynik jest 16-bitowy!

- **Znacznik przeniesienia pomocniczego - Auxiliary Flag (AF)** – ustawiany na **1** kiedy wystąpi przeniesienie z bitu 3 na 4 (nibble-4 bity).

- **Znacznik zezwolenia na przerwania maskowalne - Interrupt enable Flag (IF)** – kiedy ustawiony jest na **1** CPU przyjmuje przerwania od urządzeń zewnętrznych.

- **Znacznik kierunku - Direction Flag (DF)** – znacznik ten wykorzystywany przez instrukcje przetwarzające łańcuchy znaków, kiedy znacznik ustawiony jest na **0** znaki przetwarzane są do przodu, kiedy ma wartość **1** – kierunek przetwarzania jest odwrotny.

Poznamy trzy grupy instrukcji arytmetycznych i logicznych.

Grupa pierwsza: **ADD**, **SUB**, **CMP**, **AND**, **TEST**, **OR**, **XOR**

Wykorzystywane są następujące rodzaje argumentów:

REG, memory
memory, REG
REG, REG
memory, immediate
REG, immediate

REG: AX, BX, CX, DX, AH, AL, BL, BH, CH, CL, DH, DL, DI, SI, BP, SP.
memory: [BX], [BX+SI+7], zmienna, itd...
immediate: 5, -24, 3Fh, 10001101b, itd...

Po wykonaniu operacji wynik przechowywany jest w pierwszym argumencie. Rozkazy **CMP** i **TEST** ustawiają tylko znaczniki nie zapisując wyniku operacji (wykorzystywane są do podejmowania decyzji podczas wykonywania program – skoki warunkowe i wywołanie podprogramów).

Instrukcje te działają tylko na następujące znaczniki: **CF**, **ZF**, **SF**, **OF**, **PF**, **AF**.

- **ADD** – dodaje drugi operand do pierwszego.
- **SUB** - odejmuje drugi operand od pierwszego.
- **CMP** - odejmuje drugi operand do pierwszego lecz **nie zapisuje wyniku odejmowania, ustawia tylko znaczniki**.
- **AND** – iloczyn logiczny pomiędzy parami bitów dwóch operandów. Stosowane są zależności:
  1 AND 1 = 1
  1 AND 0 = 0
  0 AND 1 = 0
  0 AND 0 = 0
  Jak widzimy wynik jest równy **1** tylko wtedy, gdy oba bity są **1**.
- **TEST** – działa tak samo jak **AND** ale **ustawia tylko znaczniki**.
- **OR** - suma logiczna pomiędzy parami bitów dwóch operandów. Stosowane są zależności:
  1 OR 1 = 1
  1 OR 0 = 1
  0 OR 1 = 1
  0 OR 0 = 0
  Jak widzimy wynik jest równy **1** jeśli przynajmniej jeden bit argumentów jest równy **1**.
- **XOR** – Suma rozłączna (eXclusive OR) pomiędzy parami bitów dwóch operandów. Stosowane są zależności:
  1 XOR 1 = 0
  1 XOR 0 = 1
  0 XOR 1 = 1
  0 XOR 0 = 0
  Jak widzimy wynik jest równy **1** kiedy bity argumentów różnią się.

Druga grupa: **MUL**, **IMUL**, **DIV**, **IDIV**

Wykorzystywane są następujące rodzaje argumentów:

REG
memory

REG: AX, BX, CX, DX, AH, AL, BL, BH, CH, CL, DH, DL, DI, SI, BP, SP.

memory: [BX], [BX+SI+7], zmienna, itd.

Instrukcje **MUL** i **IMUL** działają tylko na znaczniki: **CF**, **OF**

Kiedy wynik przekracza zakres operandów znaczniki ustawiane są na **1**, w przeciwnym przypadku na **0**.

Dla instrukcji **DIV** i **IDIV** znaczniki są niezdefiniowane.

- **MUL** – mnożenie liczb bez znaku:

  kiedy operand jest typu **byte**: AX = AL * operand.

  kiedy operand jest typu **word**: (DX AX) = AX * operand.

- **IMUL** – mnożenie liczb ze znakiem:

  kiedy operand jest typu **byte**: AX = AL * operand.

  kiedy operand jest typu **word**: (DX AX) = AX * operand.

- **DIV** – dzielenie liczb bez znaku:

  kiedy operand jest typu **byte**: AL = AX / operand
  AH = reszta z dzielenia (moduł)

  kiedy operand jest typu **word**: AX = (DX AX) / operand
  DX = reszta z dzielenia (moduł)

- **IDIV** – dzielenie liczb ze znakiem:

  kiedy operand jest typu **byte**: AL = AX / operand
  AH = reszta z dzielenia (moduł).

  kiedy operand jest typu **word**: AX = (DX AX) / operand
  DX = reszta z dzielenia (moduł).

Trzecia grupa: **INC**, **DEC**, **NOT**, **NEG**

Wykorzystywane są następujące rodzaje argumentów:

REG
memory

REG: AX, BX, CX, DX, AH, AL, BL, BH, CH, CL, DH, DL, DI, SI, BP, SP.

memory: [BX], [BX+SI+7], zmienna, itd.

INC, **DEC** Instrukcje te działają tylko na następujące znaczniki:

ZF, **SF**, **OF**, **PF**, **AF**.

NOT Instrukcja nie zmienia żadnego znacznika!

NEG Instrukcje ta działa tylko na następujące znaczniki:

CF, **ZF**, **SF**, **OF**, **PF**, **AF**.

- **NOT** – Neguje każdy bit operandu.
- **NEG** – Zamienia liczbę na ujemną (w systemie U2). Zamienia każdy bit argumentu na przeciwny i dodaje do wyniku 1.

Zadania

1. Zapoznać się z działaniem instrukcji arytmetycznych i logicznych dobierając odpowiednie argumenty dla operacji. Sprawdzać wyniki korzystając z kalkulatora Windows.
2. Dobrać takie wartości argumentów aby móc zaobserwować ustawianie poszczególnych znaczników.
3. Sprawdzić możliwość wykonywania obliczeń na liczbach wielokrotnej precyzji (np. 32-bitowych)
4. Zapoznać się z instrukcjami przesunięć i rotacji (shl, shr, sar, sal, rol, ror, rcl, rcr).