# Programowanie niskopoziomowe

# Materiały dodatkowe

Labolatoria 5

„Sterowanie przebiegiem wykonania programu”

**CMP**

- Składnia: CMP *destination*, *source*

**działanie:** porównuje *destination* i *source* ustawiając odpowiednio flagi; działanie instrukcji jest identyczne jak: sub destination,source

z tą różnicą, że wynik nie jest nigdzie zapamiętywany, a mimo to ustawiane są flagi.

*destination***:** rejestr lub adres pamięci.

*source***:** konkretna wartość, rejestr lub adres pamięci (ale tylko jeśli *destination* również nie jest adresem pamięci).

**przykład:** cmp edx, ebx

**TEST**

- TEST destination, source

**działanie:** wykonuje operację AND (koniunkcja) na odpowiadających sobie bitach operandów *destination* i *source*;

działa podobnie jak instrukcja and, jednak nie zapisuje nigdzie wyniku operacji. Ustawia jedynie stosowne flagi, w sposób identyczny w jaki robi to instrukcja and.

*destination***:** rejestr lub adres pamięci.

*source***:** konkretna wartość, rejestr lub adres pamięci (ale tylko jeśli *destination* również nie jest adresem pamięci).

**przykład:** test edx, ebx

**Skok bezwarunkowy**

- Składnia: JMP *etykieta*

**Skoki na podstawie pojedynczej flagi**

| Mnemonic | Description | Flags |
|---|---|---|
| JZ | Jump if zero | ZF = 1 |
| JNZ | Jump if not zero | ZF = 0 |
| JC | Jump if carry | CF = 1 |
| JNC | Jump if not carry | CF = 0 |
| JO | Jump if overflow | OF = 1 |
| JNO | Jump if not overflow | OF = 0 |
| JS | Jump if signed | SF = 1 |
| JNS | Jump if not signed | SF = 0 |
| JP | Jump if parity (even) | PF = 1 |
| JNP | Jump if not parity (odd) | PF = 0 |

**Skoki oparte na równości**

| Mnemonic | Description |
|---|---|
| JE | Jump if equal (*leftOp* = *rightOp*) |
| JNE | Jump if not equal (*leftOp* ≠ *rightOp*) |
| JCXZ | Jump if CX = 0 |
| JECXZ | Jump if ECX = 0 |

**Skoki oparte na porównaniach bez znaku**

| Mnemonic | Description |
|---|---|
| JA | Jump if above (if *leftOp* > *rightOp*) |
| JNBE | Jump if not below or equal (same as JA) |
| JAE | Jump if above or equal (if *leftOp* >= *rightOp*) |
| JNB | Jump if not below (same as JAE) |
| JB | Jump if below (if *leftOp* < *rightOp*) |
| JNAE | Jump if not above or equal (same as JB) |
| JBE | Jump if below or equal (if *leftOp* <= *rightOp*) |
| JNA | Jump if not above (same as JBE) |

**Skoki oparte na porównaniach ze znakiem**

| Mnemonic | Description |
|---|---|
| JG | Jump if greater (if *leftOp* > *rightOp*) |
| JNLE | Jump if not less than or equal (same as JG) |
| JGE | Jump if greater than or equal (if *leftOp* >= *rightOp*) |
| JNL | Jump if not less (same as JGE) |
| JL | Jump if less (if *leftOp* < *rightOp*) |
| JNGE | Jump if not greater than or equal (same as JL) |
| JLE | Jump if less than or equal (if *leftOp* <= *rightOp*) |
| JNG | Jump if not greater (same as JLE) |

**Konstruowanie pętli:**

```
mov ecx, 100

label:

    ;assembler code here
loop label
```

instukcja loop odejmuje od ecx 1, sprawdza czy ecx nie jest zerem , jeżeli nie jest to wykonuje skok do określonej etykiety

Skok musi być skokiem krótkim (do 127 bajtów). Wydajniejszą strukturą jest kod przedstawiony poniżej:

```
    mov edx, 100

        label:
        ; assembler code
        here dec edx
        jnz label
```

MASM udostępnia szereg użytecznych makr, także do sterowanie przebiegiem programu. Odpowiednią pomoc znajdziejsz w edytorze MASM32 Menu Help-> High Level Macro Help.

Uwaga: od zadania 4 można ich używać w swoich rozwiązaniach o ile treść zadania nie stanowi inaczej.

**.IF Przykłady**

```
.if variable == value

        ;do something here
.endif
```

```
.if variable == value
        ; do something
.elseif variable != another_value
        ;do someting else
.else
        ;otherwise
.endif
```

```
.if variable > 50 && variable < 100
        ;number is within range
.else
        ;number is out of range
.endif
```

```
.IF eax > ebx && eax > ecx

        mov edx,1

.ELSE

        mov edx,2
```

**.ENDIF**

**.IF SDWORD PTR eax > ebx**

**mov result,1**

**.ENDIF**

**.REPEAT Przykład**

```
.repeat
        Sub eax,1
.until eax < 1
```

**.WHILE Przykład**

**.WHILE eax < 10**

**inc eax**

**.ENDW**

**Operatory wykorzystywane w wyrażeniach porównujących**

| 1 Operator | Description |
|---|---|
| *expr1* == *expr2* | Returns true when *expression1* is equal to *expr2*. |
| *expr1* != *expr2* | Returns true when *expr1* is not equal to *expr2*. |
| *expr1* > *expr2* | Returns true when *expr1* is greater than *expr2*. |
| *expr1* >= *expr2* | Returns true when *expr1* is greater than or equal to *expr2*. |
| *expr1* < *expr2* | Returns true when *expr1* is less than *expr2*. |
| *expr1* <= *expr2* | Returns true when *expr1* is less than or equal to *expr2*. |
| ! *expr* | Returns true when *expr* is false. |
| *expr1* && *expr2* | Performs logical AND between *expr1* and *expr2*. |
| *expr1* \|\| *expr2* | Performs logical OR between *expr1* and *expr2*. |
| *expr1* & *expr2* | Performs bitwise AND between *expr1* and *expr2*. |
| CARRY? | Returns true if the Carry flag is set. |
| OVERFLOW? | Returns true if the Overflow flag is set. |
| PARITY? | Returns true if the Parity flag is set. |
| SIGN? | Returns true if the Sign flag is set. |
| ZERO? | Returns true if the Zero flag is set. |