Základní řídící konstrukce

ISU cv 7

http://www.fit.vutbr.cz/~isakin/isu

info

- Prohození obsahu 7. a 8. cvičení
  zásobník <-> konstrukce a cykly

Ne/podmíněné skoky

Nejčastější porovnávací instrukce

- Aritmetické porovnání: `cmp op1, op2`
- Logické porovnání: `test op1, op2`

Programový čítač

- Obsahuje adresu instrukce, která má být provedena
- Adresa ukládána do EIP = Extended Instruction Pointer
- Počáteční hodnota - adresa první instrukce v main
- Po provedení instrukce se do EIP automaticky nahraje adresa následující instrukce
- EIP nelze měnit běžnými instrukcemi
- Hodnotu můžeme měnit skokem na návěští

Nepodmíněný skok

- `jmp label` (v C goto)
  - Nepodmíněný skok vpřed (dolů) nám umožní část kódu vynechat.

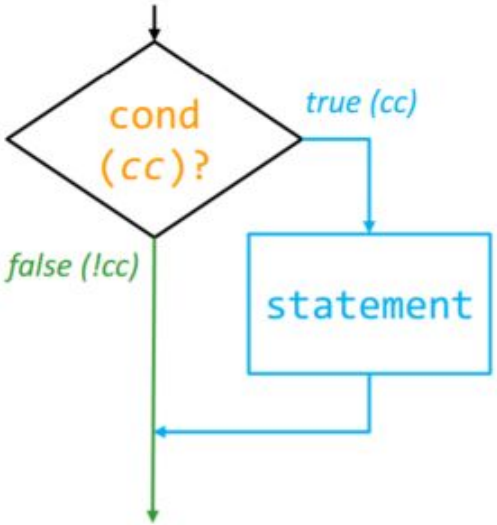
```
1    mov   eax, 10              ;EAX = 10
2    jmp   label                ;skoc na navesti LABEL
3    mov   eax, 20              ;EAX = 20 (bude preskoceno)
4  label:                       ;navesti LABEL
5    call WriteInt32NewLine     ;vypis EAX (hodnota 10)
```

  - Nepodmíněný skok vzad (nahoru) vytvoří nekonenčý cyklus.

```
6    mov   eax, 0               ;EAX = 0
7  label:                       ;navesti LABEL
8    call WriteInt32NewLine     ;vypis EAX
9    inc   eax                  ;EAX = EAX + 1
10   jmp   label                ;skoc na navesti NAVESTI
```

Podmíněné skoky

- Skočí na návěští pouze při splnění podmínky



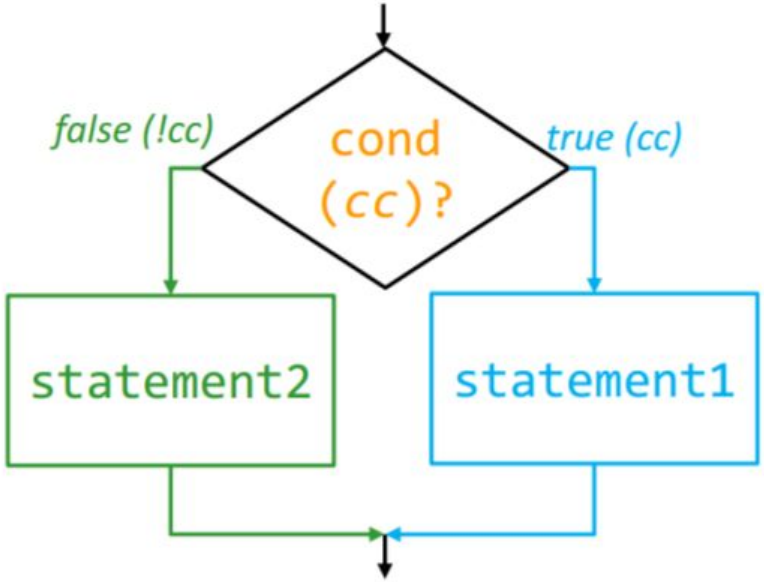

```
    …
    CMP/TEST
    JNcc false
(true:)
    statement
false:
    …
```

```
mov eax, -5
cmp eax, 0
jg .kladne
…
.kladne:
```

Podmíněné skoky

```
    CMP/TEST
    Jcc true
(else:)
    statement2
    JMP endif
true:
    statement1
endif:
    …
```

```
    CMP/TEST
    JNcc else
(true:)
    statement1
    JMP endif
else:
    statement2
endif:
    …
```



Podmíněné skoky

- Skočí na návěští pouze při splnění podmínky

Skok může být podmíněn hodnotou nějakého konkrétního příznaku (JO, JC, JS, JZ):

```
1  jo    label ;skoc pokud OF == 1 (Jump if Overflow)
2  jc    label ;skoc pokud CF == 1 (Jump if Carry)
3  js    label ;skoc pokud SF == 1 (Jump if Sign)
4  jz    label ;skoc pokud ZF == 1 (Jump if Zero)
```

Podmínka skoku může být negovaná (JNO, JNC, JNS, JNZ):

```
5  jno   label ;skoc pokud OF == 0 (Jump if Not Overflow)
6  jnc   label ;skoc pokud CF == 0 (Jump if Not Carry)
7  jns   label ;skoc pokud SF == 0 (Jump if Not Sign)
8  jnz   label ;skoc pokud ZF == 0 (Jump if Not Zero)
```

Podmíněné skoky

JUMPS Unsigned (Cardinal)			
JA	Jump if Above	JA Dest	(≡ JNBE)
JAE	Jump if Above or Equal	JAE Dest	(≡ JNB ≡ JNC)
JB	Jump if Below	JB Dest	(≡ JNAE ≡ JC)
JBE	Jump if Below or Equal	JBE Dest	(≡ JNA)
JNA	Jump if not Above	JNA Dest	(≡ JBE)
JNAE	Jump if not Above or Equal	JNAE Dest	(≡ JB ≡ JC)
JNB	Jump if not Below	JNB Dest	(≡ JAE ≡ JNC)
JNBE	Jump if not Below or Equal	JNBE Dest	(≡ JA)
JC	Jump if Carry	JC Dest	
JNC	Jump if no Carry	JNC Dest	

JUMPS Signed (Integer)			
JG	Jump if Greater	JG Dest	(≡ JNLE)
JGE	Jump if Greater or Equal	JGE Dest	(≡ JNL)
JL	Jump if Less	JL Dest	(≡ JNGE)
JLE	Jump if Less or Equal	JLE Dest	(≡ JNG)
JNG	Jump if not Greater	JNG Dest	(≡ JLE)
JNGE	Jump if not Greater or Equal	JNGE Dest	(≡ JL)
JNL	Jump if not Less	JNL Dest	(≡ JGE)
JNLE	Jump if not Less or Equal	JNLE Dest	(≡ JG)
JO	Jump if Overflow	JO Dest	
JNO	Jump if no Overflow	JNO Dest	
JS	Jump if Sign (= negative)	JS Dest	
JNS	Jump if no Sign (= positive)	JNS Dest	

Switch

- možno zapsat několika způsoby

```
switch (x) {
    case val1:      block1; break;
    case val2:      block2; break;
    . . .
    case valN:      blockN; break;
    default:        blockDefault;
}
```

```
segment .data
cases       DD case_val1
            DD case_val2
            DD case_default
            . . .
            DD case_valN
segment .text
    MOV EAX, x
    JMP dword [cases+4*EAX]
case_default:
    blockDefault
    JMP break
case_val1:
    block1
    JMP break
    . . .
case_valN:
    blockN
break:
```

Ukázka

- cv7_A.asm

Úkoly

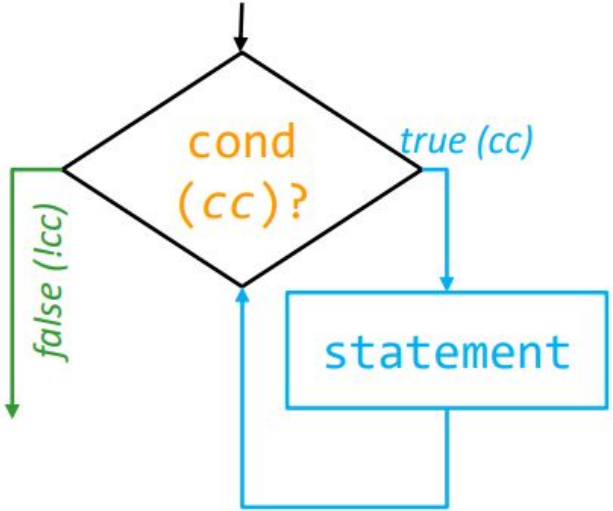
- cv7_1.asm
- cv7_2.asm

Cykly

while

```
    …
while:
    CMP/TEST
    JNcc end
    statement
    JMP while
end:
    …
```

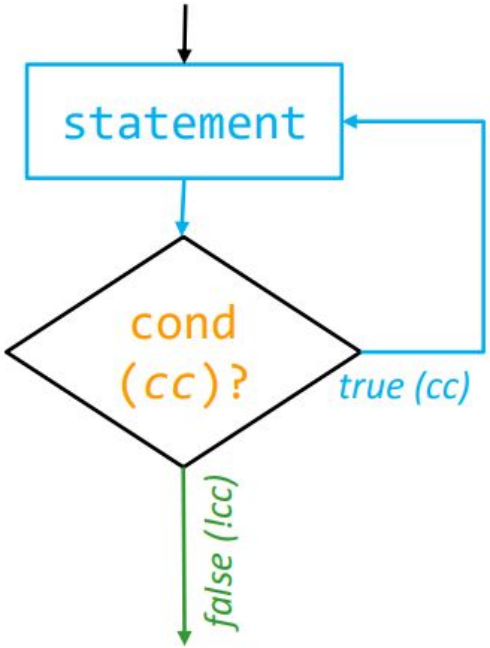
```
    …
while:
    CMP/TEST
    Jcc true
    JMP end
true:
    statement
    JMP while
end:
    …
```



do-while

```
    …
dowhile:
    statement
    CMP/TEST
    Jcc dowhile
    …
```

```
    …
dowhile:
    statement
    CMP/TEST
    JNcc end
    JMP dowhile
end:
    …
```



while, do-while

- Cyklus while se na začátku těla ptá jestli cyklus chceme ukončit.

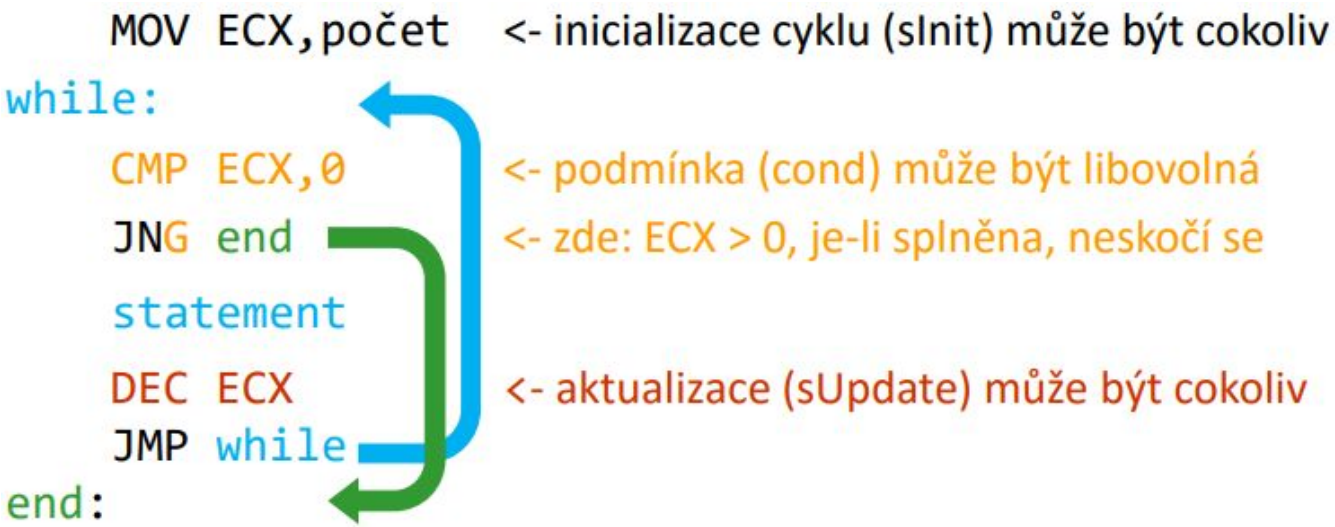
```
1  condition:
2    cmp   eax, 0     ;porovnanim hodnot nastavime priznaky
3    je    end        ;podminkou rozhodujeme o ukonceni cyklu
4  while:
5    ;telo WHILE
6    jmp   condition ;ne-podminenym skokem se vracime na podminku
7  end:
```

- Cyklus do-while se na konci těla ptá jestli cyklus chceme opakovat.

```
8  while:
9    ;telo WHILE
10 condition:
11   cmp   eax, 0     ;porovnanim hodnot nastavime priznaky
12   jne   while      ;podminkou rozhodujeme o opakovani cyklu
13 end:
```

for (známý počet iterací)

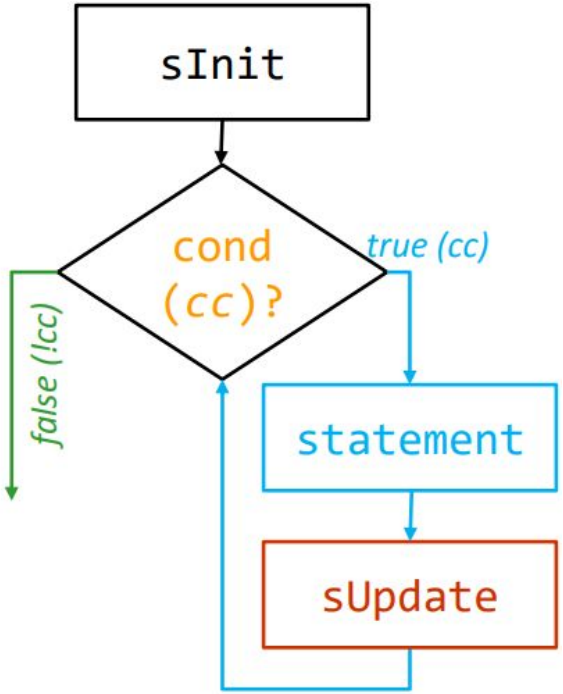
Jako while + INC/DEC registru ECX + vyhodnocení ECX



for (známý počet iterací)

cyklus `for` jazyka C
obecně ~ `while`

```
    …
    sInit
for:
    CMP/TEST
    JNcc end
    statement
    sUpdate
    JMP for
end:
    …
```



loop

- *for* cyklus s automatickou dekrementací ECX
  pokud není rovno 0 (JNZ) provede skok na návěští.

```
    mov ecx, 10
cycle:
    ; some code
    loop cycle
```

- *break* a *continue* ?

Ukázka

- cv7_B.asm

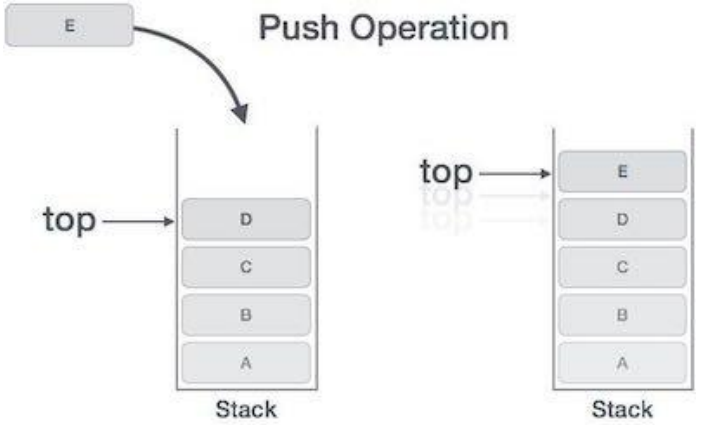
Úkoly

- cv7_3.asm
- cv7_4.asm

Zásobník

Hlavní instrukce zásobníku

- `push` op
- `pop` op



ESP - ukazatel na vrchol zásobníku (stack pointer), vždy na vrcholu, furt se může hýbat

Ukázka

- cv7_C.asm

Úkol

- cv7_5.asm

THE END
... for this week