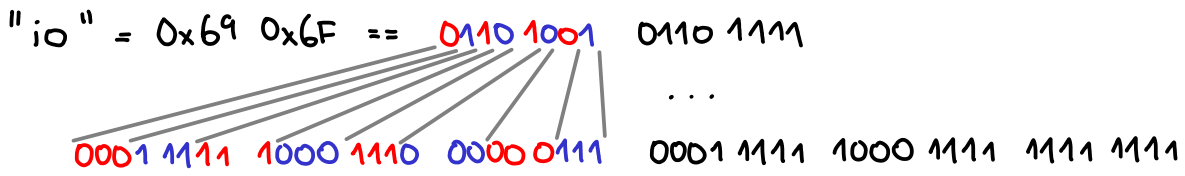
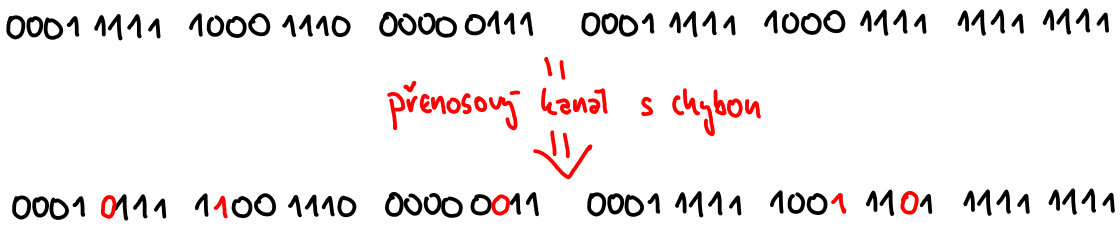
Zadání isu8-cv5-1

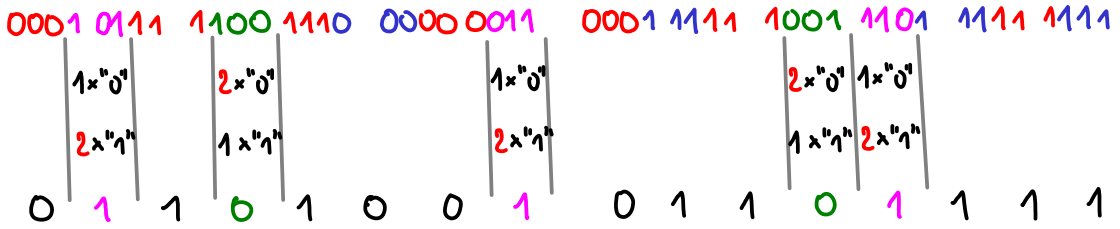
Uvažujte zprávu zakódovanou pomocí "triple bit redundancy" kódu. Tato zpráva je vytvořená ztrojením každého bitu vstupní posloupnosti bajtů:



Při přenosu zprávy může následně dojít k poškození obsahu zprávy.



V případě, že je poškozen maximálně jeden bit z každé trojice, dokážeme zprávu zrekonstruovat:

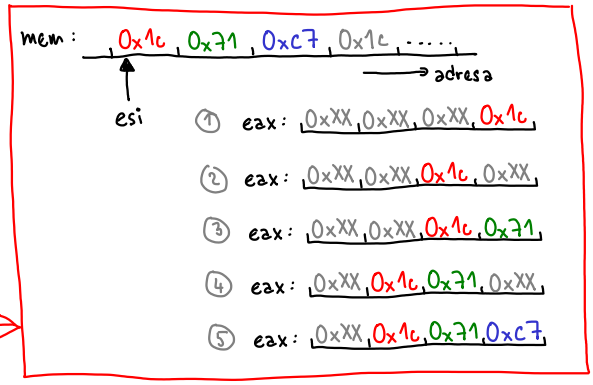


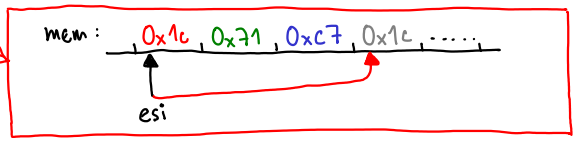
Vaším úkolem je naimplementovat tuto rekonstrukci v jazyce symbolických instrukcí pro architekturu x86.

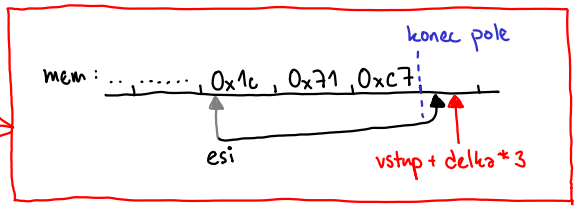
Zprávy zakódované s různou chybou můžete generovat pomocí python skriptu, který je k dispozici ke stažení.

Doporučený postup implementace

```
%include "rw32-2018.inc"

section .data

; Vstupni pole obsahujici retezec "Hello World!" v "triple bit redundancy" kodu (kazdy bit je vlozeny 3x), vstup obsahuje bitove chyby ktere lze opravit
; - pozor: trojice bajtu jsou ulozeny jako "big endian", tzn. prvni bajt z trojice (na nejnizsi adrese) je nejvice vyznamny (MSB)!
vstup db 0x38,0x9c,0x49,0x3b,0x13,0x8e,0x3b,0x1d,0x09,0x3b,0x1d,0x89,0x3b,0x1d,0x06,0x27,0x12,0x49,0x3b,0xe3,0xb6,0x3b,0x1d,0x06,0x3b,0x62,0x71,0x3b,0x1d,0x89,0x3b,0x13,0x89,0x27,0x12,0x4e
; vhodny ladici vstup: 00011100 01110001, 11000111 00011100, 01110001 11000111; vystup 2B: 01010101, 01010101 ("UU")
;vstup db 0x1c,0x71,0xc7,0x1c,0x71,0xc7

; Delka rozkodovane sekvence (delka vstupniho pole deleno 3, protoze kazdemu zakodovanemu bajtu nalezi 3 bajty vstupni sekvence), tzn. delka vystupu
; - jedna sa o konstantu spoctenou za prekladu, neni ulozena v datove sekci
delka equ ($ - vstup) / 3

section .bss

; Misto pro ulozeni dekodovane sekvence, 1 bajt navyse poslouzi k ulozeni ukoncovaci nuly '\0' (C-ckovy styl retezce)
vystup resb (delka + 1)

section .text
CMAIN:
    mov ebp, esp; for correct debugging

    ; Inicializace pred vnejsim cyklem
    ; - pouzijte esi a edi jako ukazatele na vstupni a vystupni retezec bajtu, inicializujte je na "vstup" a "vystup"
    ; - v pripade, ze to bude nutne, muzete pridat dalsi incializace (zalezi na spusobu implementace)


    ; Navesti vnejsiho cyklu
    ; - idea: v kazde iteraci nactete 3 bajty z pole "vstup" a dekodujte je na 1 bajt, ktery ulozte do pole "vystup"
    .bajt_cyklus:
        ; Nacteni 3B ze vstupni posloupnosti do registru eax
        ; - vzhledem k pouziti "big endian" na vstupu nacitejte postupne po bajtech a pouzijte posuv eax pro vytvoreni mista pro dalsi bajt
        ; - cela operace by mela skoncit nasledovne: eax <= 0xXX:[esi+0]:[esi+1]:[esi+2] (obsah nejvyssiho bajtu je nezajimavy)


        ; Posun ukazovatele esi na dalsi vstupni posloupnost 3 bajtu pro dalsi iteraci


        ; Inicializace pred vnitrnim cyklem
        ; - nastavte pocet iteraci nebo vynulujte pocitadlo cyklu (zalezi na spusobu implementace)
        ; - vynulujte registr pro ulozeni dekodovaneho bajtu (zalezi na spusobu implementace)


        ; Navesti vnitrniho cyklu
        ; - idea: nactene 3 bajty v eax postupne po 3 bitoch prekodujte v 8 iteracich na 1 bajt do zvoleneho vystupniho registru
        .symbol_cyklus:
            ; Nacteni 3 bitu a ulozeni zodpovidajiciho 1 bitu do ciloveho registru
            ; - zjistete pocet jednicek v nejnizsich 3 bitech eax
            ; - jestlize je pocet jednicek 2 nebo 3, vlozte bit 1 do ciloveho registru, jinak vlozte 0
            ; - posunte registr eax tak, aby byli v dalsi iteraci spracovane nasledujici 3 bity
            ; - orotujte vystupni registr tak, aby byl v dalsi iteraci vlozen nasledujici bit
            ; (kroky a jejich poradi zavisi na implementaci)


            ; Ukoncovaci podminka vnitrniho cyklu a podmineny skok na ".symbol_cyklus"
            ; - dekrementujte resp. inkrementujte pocitadlo a porovnajte s 0 resp. 8
            ; - podminene skocte na ".symbol_cyklus"
            ; - jestlize pouzivate ecx jako pocitadlo, muzete vyuzit instrukci loop
            ; (kroky a jejich poradi zavisi na implementaci)


            j__ .symbol_cyklus

        ; Zapis vysledneho bajtu do vystupniho pole
        ; - zapiste dekodovany bajt na adresu edi


        ; Posun ukazatele edi na nasledujici bajt vystupu


        ; Ukoncovaci podminka vnejsiho cyklu a podmineny skok na ".bajt_cyklus"
        ; - porovnejte ukazatel na vstup esi s ukazatelem "za" konec pole (vstup + delka * 3)
        ; - podminene skocte na ".bajt_cyklus"


        j__ .bajt_cyklus

    ; Ulozeni nuloveho bajtu za konec dekodovane sekvence
    ; - predpoklada, ze edi ukazuje "za" posledni zapsany bajt (jestli se drzite osnovy, melo by platit bez dalsich instrukci)
    mov [edi], byte 0
    mov esi, vystup
    ; Vypis vysledneho retezce
    call WriteString
    call WriteNewLine

    ; Navratova hodnota z main
    xor eax, eax
    ret
```



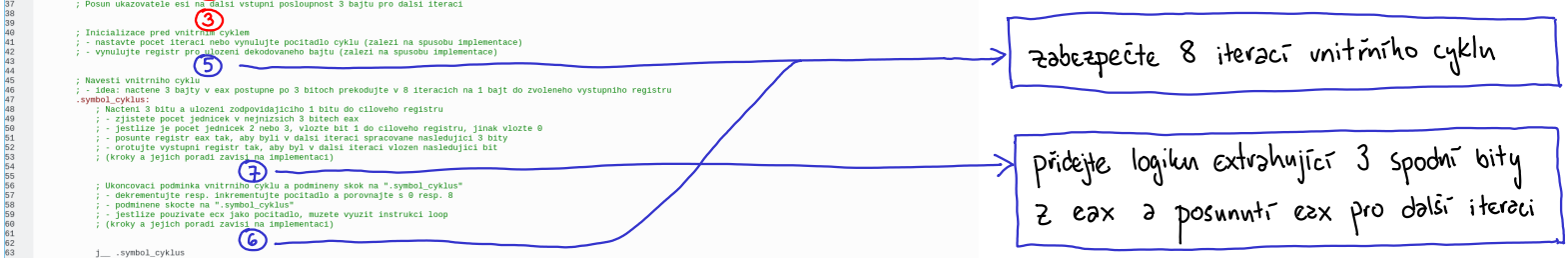




Po dokončení kroků 1-4 zakomentujte skok implementující vnitřní cyklus (j__ .symbol_cyklus) a skontrolujte si implemetaci použitím debuggeru (správné načítaní do eax, ukončení cyklu po přečtení vstupu).

Na výstup by měl být vypsán prázdný řetězec, program nesmí spadnout na zlý přístup do paměti.

```asm
%include "rw32-2018.inc"

section .data

; Vstupni pole obsahujici retezec "Hello World!" v "triple bit redundancy" kodu (kazdy bit je vlozeny 3x), vstup obsahuje bitove chyby ktere lze opravit
; - pozor: trojice bajtu jsou ulozeny jako "big endian", tzn. prvni bajt z trojice (na nejnizsi adrese) je nejvice vyznamny (MSB)!
vstup db 0x38,0x9c,0x49,0x3b,0x13,0x8a,0x3b,0x1d,0x89,0x3b,0x1d,0x89,0x3b,0x1d,0xb6,0x27,0x12,0x49,0x3b,0xe3,0xb6,0x3b,0x1d,0xb6,0x3b,0x62,0x71,0x3b,0x1d,0x89,0x3b,0x13,0x89,0x27,0x12,0x4e
; vhodny ladici vstup: 00011100 01110001, 11000111 00011100, 01110001 11000111; vystup 2B: 01010101, 01010101 ("UU")
;vstup db 0x1c,0x71,0xc7,0x1c,0x71,0xc7

; Delka rozkodovane sekvence (delka vstupniho pole deleno 3, protoze kazdemu zakodovanemu bajtu nalezi 3 bajty vstupni sekvence), tzn. delka vystupu
; - jedna sa o konstantu spoctenou za prekladu, neni ulozena v datove sekci
delka equ ($ - vstup) / 3

section .bss

; Misto pro ulozeni dekodovane sekvence, 1 bajt navyse poslouzi k ulozeni ukoncovaci nuly '\0' (C-ckovy styl retezce)
vystup resb (delka + 1)

section .text
CMAIN:
    mov ebp, esp; for correct debugging

    ; Inicializace pred vnejsim cyklem
    ; - pouzijte esi a edi jako ukazatele na vstupni a vystupni retezec bajtu, inicializujte je na "vstup" a "vystup"
    ; - v pripade, ze to bude nutne, muzete pridat dalsi incializace (zalezi na spusobu implementace)


    ; Navesti vnejsiho cyklu
    ; - idea: v kazde iteraci nactete 3 bajty z pole "vstup" a dekodujte je na 1 bajt, ktery ulozte do pole "vystup"
    .bajt_cyklus:
        ; Nacteni 3B ze vstupni posloupnosti do registru eax
        ; - vzhledem k pouziti "big endian" na vstupu nacitejte postupne po bajtech a pouzijte posuv eax pro vytvoreni mista pro dalsi bajt
        ; - cela operace by mela skoncit nasledovne: eax <= 0xXX:[esi+0]:[esi+1]:[esi+2] (obsah nejvyssiho bajtu je nezajimavy)


        ; Posun ukazovatele esi na dalsi vstupni posloupnost 3 bajtu pro dalsi iteraci


        ; Inicializace pred vnitrnim cyklem
        ; - nastavte pocet iteraci nebo vynulujte pocitadlo cyklu (zalezi na spusobu implementace)
        ; - vynulujte registr pro ulozeni dekodovaneho bajtu (zalezi na spusobu implementace)


        ; Navesti vnitrniho cyklu
        ; - idea: nactene 3 bajty v eax postupne po 3 bitoch prekodujte v 8 iteracich na 1 bajt do zvoleneho vystupniho registru
        .symbol_cyklus:
            ; Nacteni 3 bitu a ulozeni zodpovidajiciho 1 bitu do ciloveho registru
            ; - zjistete pocet jednicek v nejnizsich 3 bitech eax
            ; - jestlize je pocet jednicek 2 nebo 3, vlozte bit 1 do ciloveho registru, jinak vlozte 0
            ; - posunte registr eax tak, aby byli v dalsi iteraci spracovane nasledujici 3 bity
            ; - orotujte vystupni registr tak, aby byl v dalsi iteraci vlozen nasledujici bit
            ; (kroky a jejich poradi zavisi na implementaci)


            ; Ukoncovaci podminka vnitrniho cyklu a podmineny skok na ".symbol_cyklus"
            ; - dekrementujte resp. inkrementujte pocitadlo a porovnajte s 0 resp. 8
            ; - podminene skocte na ".symbol_cyklus"
            ; - jestlize pouzivate ecx jako pocitadlo, muzete vyuzit instrukci loop
            ; (kroky a jejich poradi zavisi na implementaci)

            j__ .symbol_cyklus

        ; Zapis vysledneho bajtu do vystupniho pole
        ; - zapiste dekodovany bajt na adresu edi


        ; Posun ukazatele edi na nasledujici bajt vystupu


        ; Ukoncovaci podminka vnejsiho cyklu a podmineny skok na ".bajt_cyklus"
        ; - porovnejte ukazatel na vstup esi s ukazatelem "za" konec pole (vstup + delka * 3)
        ; - podminene skocte na ".bajt_cyklus"


        j__ .bajt_cyklus

    ; Ulozeni nuloveho bajtu za konec dekodovane sekvence
    ; - predpoklada, ze edi ukazuje "za" posledni zapsany bajt (jestli se drzite osnovy, melo by platit bez dalsich instrukci)
    mov [edi], byte 0
    mov esi, vystup
    ; Vypis vysledneho retezce
    call WriteString
    call WriteNewLine

    ; Navratova hodnota z main
    xor eax, eax
    ret
```



Implementaci kroků 5-7 je možné ověřit pomocí debuggeru, kdy vo vnitřním cyklu v osmi iteracích v místě 7 postupně dostáváte hodnoty tří bitů z eax.

Počítadlo iterací vnitřní smyčky může být ecx, ale jenom v případě, že ho nebudete potřebovat (některé přístupy pro určení výsledného bitu v bodě 9 mohou vyžadovat posun rotaci o nekonstantní počet bitů). Je snadné ecx + případnou instrukci loop vyměnit za jinou později.

```
%include "rw32-2018.inc"

section .data

; Vstupni pole obsahujici retezec "Hello World!" v "triple bit redundancy" kodu (kazdy bit je vlozeny 3x), vstup obsahuje bitove chyby ktere lze opravit
; - pozor: trojice bajtu jsou ulozeny jako "big endian", tzn. prvni bajt z trojice (na nejnizsi adrese) je nejvice vyznamny (MSB)!
vstup db 0x38,0x3c,0x49,0x3b,0x13,0x8a,0x3b,0x1d,0x89,0x3b,0x1d,0x89,0x3b,0x1d,0xb6,0x27,0x12,0x49,0x38,0xe3,0xb6,0x3b,0x1d,0xb6,0x3b,0x62,0x71,0x3b,0x1d,0x89,0x3b,0x13,0x89,0x27,0x12,0x4e
; vhodny ladici vstup: 00011100 01110001, 11000111 00011100, 01110001 11000111; vystup 2B: 01010101, 01010101 ("UU")
;vstup db 0x1c,0x71,0xc7,0x1c,0x71,0xc7

; Delka rozkodovane sekvence (delka vstupniho pole deleno 3, protoze kazdemu zakodovanemu bajtu nalezi 3 bajty vstupni sekvence), tzn. delka vystupu
; - jedna sa o konstantu spoctenou za prekladu, neni ulozena v datove sekci
delka equ ($ - vstup) / 3

section .bss

; Misto pro ulozeni dekodovane sekvence, 1 bajt navyse poslouzi k ulozeni ukoncovaci nuly '\0' (C-ckovy styl retezce)
vystup resb (delka + 1)

section .text
CMAIN:
    mov ebp, esp; for correct debugging

    ; Inicializace pred vnejsim cyklem
    ; - pouzijte esi a edi jako ukazatele na vstupni a vystupni retezec bajtu, inicializujte je na "vstup" a "vystup"
    ; - v pripade, ze to bude nutne, muzete pridat dalsi incializace (zalezi na spusobu implementace)
                                    (1)


    ; Navesti vnejsiho cyklu
    ; - idea: v kazde iteraci nactete 3 bajty z pole "vstup" a dekodujte je na 1 bajt, ktery ulozte do pole "vystup"
    .bajt_cyklus:
        ; Nacteni 3B ze vstupni posloupnosti do registru eax
        ; - vzhledem k pouziti "big endian" na vstupu nacitejte postupne po bajtech a pouzijte posuv eax pro vytvoreni mista pro dalsi bajt
        ; - cela operace by mela skoncit nasledovne: eax <= 0xXX:[esi+0]:[esi+1]:[esi+2] (obsah nejvyssiho bajtu je nezajimavy)
                                    (2)

        ; Posun ukazovatele esi na dalsi vstupni posloupnost 3 bajtu pro dalsi iteraci
                                    (3)

        ; Inicializace pred vnitrnim cyklem
        ; - nastavte pocet iteraci nebo vynulujte pocitadlo cyklu (zalezi na spusobu implementace)
        ; - vynulujte registr pro ulozeni dekodovaneho bajtu (zalezi na spusobu implementace)
                                    (5) (8)

        ; Navesti vnitrniho cyklu
        ; - idea: nactene 3 bajty v eax postupne po 3 bitoch prekodujte v 8 iteracich na 1 bajt do zvoleneho vystupniho registru
        .symbol_cyklus:
            ; Nacteni 3 bitu a ulozeni zodpovidajiciho 1 bitu do ciloveho registru
            ; - zjistete pocet jednicek v nejnizsich 3 bitech eax
            ; - jestlize je pocet jednicek 2 nebo 3, vlozte bit 1 do ciloveho registru, jinak vlozte 0
            ; - posunte registr eax tak, aby byli v dalsi iteraci spracovane nasledujici 3 bity
            ; - orotujte vystupni registr tak, aby byl v dalsi iteraci vlozen nasledujici bit
            ; (kroky a jejich poradi zavisi na implementaci)
                                    (7) (9)

            ; Ukoncovaci podminka vnitrniho cyklu a podmineny skok na ".symbol_cyklus"
            ; - dekrementujte resp. inkrementujte pocitadlo a porovnajte s 0 resp. 8
            ; - podminene skocte na ".symbol_cyklus"
            ; - jestlize pouzivate ecx jako pocitadlo, muzete vyuzit instrukci loop
            ; (kroky a jejich poradi zavisi na implementaci)
                                    (6)

            j__ .symbol_cyklus

        ; Zapis vysledneho bajtu do vystupniho pole
        ; - zapiste dekodovany bajt na adresu edi
                                    (10)

        ; Posun ukazatele edi na nasledujici bajt vystupu
                                    (11)

        ; Ukoncovaci podminka vnejsiho cyklu a podmineny skok na ".bajt_cyklus"
        ; - porovnejte ukazatel na vstup esi s ukazatelem "za" konec pole (vstup + delka * 3)
        ; - podmineny skocte na ".bajt_cyklus"
                                    (4)

        j__ .bajt_cyklus

    ; Ulozeni nuloveho bajtu za konec dekodovane sekvence
    ; - predpokklada, ze edi ukazuje "za" posledni zapsany bajt (jestli se drzite osnovy, melo by platit bez dalsich instrukci)
    mov [edi], byte 0
    mov esi, vystup
    ; Vypis vysledneho retezce
    call WriteString
    call WriteNewLine

    ; Navratova hodnota z main
    xor eax, eax
    ret
```

příprava registru pro výsledný bajt

nastavení bitu v závislosti na přečtených 3 bitů (volba většiny):

000 } 0
001
010
011 } 1
100 } 0
101
110 } 1
111

+ otočení cílového registru (příprava na další bit)

zápis dekódovaného bajtu na adresu edi

posun edi na následující bajt výstupu

Váš kód je po dokončení bodů 8-11 kompletní.

Volbu většiny je možné implementovat různě, např. počítáním jedniček ("pop count" instrukcí) a porovnáním na 2, nebo podmínkou x < 3 || x == 4 a.p. Doporučuji vyhnout se dalším vnořeným cyklům.

Je možné, že Vám budou docházet volné registry. V případě potřeby si jejich obsah můžete zálohovat na zásobník (např. "push eax", následně vymaskovat 3 spodní bity, na konci bodu 9 pak "pop eax" + posun o 3 bity).
Dále je možné ušetřit registry používáním 8-bitových variant, počet iterací vnitřního cyklu, dekódovaný bajt a další hodnoty se změstí do bh, bl, ch, cl, dh, dl -- sdílejte registry.