Příklad č. 1

Vytvořte v paměti RAM pole, umístěte jen v paměti SRAM od začátku, jeho velikost bude dána konstantou `N`. Pole naplňte, přičemž hodnota prvního prvku bude dána konstantou `PRVNI`, a další prvky budou mít hodnoty o 1 větší než předcházející. Registr používaný jako čítač v cyklech si pojmenujte `count`.

```
          .include "m32def.inc"

// Konstanty
.equ      N         = 5       // Počet prvků v poli
.equ      PRVNI     = 3       // První číslo v poli

// Přejmenování registru
.def      temp      = r17     // Pro uložení hodnot pro uložení do pole
.def      count     = r16     // Registr po počítání počtu cyklů pro naplnění celého pole

// Proměnné v SRAM
          .DSEG
pole:     .BYTE     N         // Pole o N prvcích

// Vlastní program
          .CSEG
          .ORG      0x0000
start:
          // Příprava cyklu
          ldi       count, N  // Inicializace řídící proměnné cyklu
          ldi       temp, PRVNI    // První prvek v poli
          // Uložení ukazatele na pole do X
          ldi       XL, LOW(pole)
          ldi       XH, HIGH(pole)
initPole:
          // Plnění pole
          st        X+, r17        // pole[i++] = temp
          inc       temp           // Další prvek v poli bude o 1 větší
          dec       count
          brne      initPole

end:      rjmp      end       // Nekonečná smyčka
```

Příklad č. 2

- Do předchozího příkladu doplňte druhé pole o stejné velikosti, přičemž do něj zkopírujte všechny prvky pole původního s tím, že ke všem prvkům přičtete hodnotu třetího prvku pole prvního, neboli:
  `pole2[i] = pole1[i] + pole1[2], i = 0,1, ... N`

- přičtení hodnoty proveďte v registru, který si pojmenujte temp. Pro uložení prvku pole1[2] si pojmenujte registr např. pole1_2

```
        .include "m32def.inc"

// Konstanty
.equ    N       = 5     // Počet prvků v poli
.equ    PRVNI   = 3     // První číslo v poli

// Přejmenování registru
.def    temp    = r17   // Pro uložení hodnot pro uložení do pole
.def    count   = r16   // Registr po počítání počtu cyklů pro naplnění celého pole
.def    pole_2  = r18

// Proměnné v SRAM
        .DSEG
pole:   .BYTE   N       // Pole o N prvcích
pole2:  .BYTE   N       // Druhé pole se zkopírovanými prvky z pole + pole[2]

// Vlastní program
        .CSEG
        .ORG    0x0000
start:
        // Příprava cyklu
        ldi     count, N  // Inicializace řídící proměnné cyklu
        ldi     temp, PRVNI    // První prvek v poli
        // Uložení ukazatele na pole do X
        ldi     XL, LOW(pole)
        ldi     XH, HIGH(pole)
initPole:
        // Plnění pole
        st      X+, r17       // pole[i++] = temp
        inc     temp          // Další prvek v poli bude o 1 větší
        dec     count
        brne    initPole
        // Konec cyklu

        // Příprava cyklu
        ldi     count, N  // Počet opakování cyklu -> zkopírují se všechny prvky

        ldi     YL, LOW(pole)  // Pro adresaci v pole
        ldi     YH, HIGH(pole)

        ldi     XL, LOW(pole2) // Pro adresaci v pole2
        ldi     XH, HIGH(pole2)

        //Kopie pole[2] -> r1
        ldd     pole_2, Y+2
```

```
kopiePoli:
          ld        r0, Y+    // Aktuální pozice pole1 do r0, Y++
          add       r0, pole_2// Sečtení r0 = r0 + pole_2
          st        X+, r0    // Uložení r0 na aktuální pozici v pole2
          dec       count     // Odečtení 1 od count
          brne      kopiePoli // Je-li count != 0 skok na kopiePoli

end:      rjmp      end       // Nekonečná smyčka
```

Příklad č. 3

- Napište program, ve kterém do paměti ROM uložíte libovolný řetězec, a zkopírujte jej do RAM (tedy do pole). Řetězec umístěte na konec programu (za poslední instrukci)

```
          .include  "m32def.inc"

// Konstanty
.equ      POCET_ZNAKU         = 12

// Přejmenování registrů
.def      count     = r16
.def      temp      = r17

// Proměnné
          .DSEG
poleZnaku:.BYTE     POCET_ZNAKU

          .CSEG
start:
          // Uložení adresy na poleZnaku do X
          ldi       XL, LOW(poleZnaku)
          ldi       XH, HIGH(poleZnaku)
          // Uložení adresy s prvním znakem v ROM (posunutý o 1 doleva, b0 = 0)
          ldi       ZL, LOW(retezec << 1)
          ldi       ZH, HIGH(retezec << 1)
          // Inicializace řídící proměnné
          ldi       count, POCET_ZNAKU
// Cyklus pro uložení znaků z ROM do SRAM
cyklus:
          // Vložení aktuálního byte z ROM do temp, Z se o 1 zvýší
          // b0 v Z bude nejprve 0, po přičtení 1 bude b0 = 1 ostatní bity se nezmění
          // po druhém přičtení 1 k Z, bude b0 = 0 a k ostatním bitům se přište 1
          // Jinak se nejprve načte první byte z ROM, pak druhý, poté se přesune na
          // další řádek v ROM atd. ...
          lpm       temp, Z+
          st        X+, temp  // Uložení temp do X, X se o 1 zvýší
          dec       count     // Snížení count o 1
          brne      cyklus    // Pokud se count != 0 skok na cyklus

end:      rjmp      end       // Nekonečná smyčka

// Uložení řetězce "Pozdrav vsem" do ROM. První byte = 'P', atd. ...
retezec:
          .DB       "Pozdrav vsem"
```

Příklad č. 4

- Modifikujte předchozí program tak, aby byl řetězec umístěn v ROM od adresy 0x100

```
          .include   "m32def.inc"

// Konstanty
.equ      POCET_ZNAKU          = 12

// Přejmenování registrů
.def      count      = r16
.def      temp       = r17

// Proměnné
          .DSEG
poleZnaku: .BYTE     POCET_ZNAKU

          .CSEG
start:
          // Uložení adresy na poleZnaku do X
          ldi        XL, LOW(poleZnaku)
          ldi        XH, HIGH(poleZnaku)
          // Uložení adresy s prvním znakem v ROM (posunutý o 1 doleva, b0 = 0)
          ldi        ZL, LOW(retezec << 1)
          ldi        ZH, HIGH(retezec << 1)
          // Inicializace řídící proměnné
          ldi        count, POCET_ZNAKU
// Cyklus pro uložení znaků z ROM do SRAM
cyklus:
          // Vložení aktuálního byte z ROM do temp, Z se o 1 zvýší
          // b0 v Z bude nejprve 0, po přičtení 1 bude b0 = 1 ostatní bity se nezmění
          // po druhém přičtení 1 k Z, bude b0 = 0 a k ostatním bitům se přište 1
          // Jinak se nejprve načte první byte z ROM, pak druhý, poté se přesune na
          // další řádek v ROM atd. ...
          lpm        temp, Z+
          st         X+, temp   // Uložení temp do X, X se o 1 zvýší
          dec        count      // Snížení count o 1
          brne       cyklus     // Pokud se count != 0 skok na cyklus

end:      rjmp       end        // Nekonečná smyčka

// Uložení řetězce "Pozdrav vsem" do ROM. První byte = 'P', atd. ...
          .ORG       0x100      // Nastavení čítače adresy na 0x100
retezec:
          .DB        "Pozdrav vsem"
```

DCV č. 1

- upravte příklad č. 2 tak, že hodnoty do pole2 budou kopírovány od konce, tedy

  `pole2[N-i] = pole1[i] + pole1[2], i = 0,1, ... N`

```
          .include  "m32def.inc"

// Konstanty
.equ      N         = 10
.equ      PRVNI     = 22

// Přejmenování registrů
.def      count     = r16
.def      temp      = r17
.def      pole1_2   = r18

// Proměnné
          .DSEG
pole1:    .BYTE     N
pole2:    .BYTE     N

          .CSEG
          // Uložení první adresy pole1 do X
          ldi       XL, LOW(pole1)
          ldi       XH, HIGH(pole1)
          // Inicializace řídící proměnné cyklu
          ldi       count, N
          // Inicializace proměnné pro plnění pole
          ldi       temp, PRVNI
// Cyklus pro naplnění pole1 PRVNI, PRVNI+1, ....
cyklus:
          // Vložení temp na aktuální pozici v pole1, X se o jednu zvýší
          st        X+, temp
          inc       temp      // temp se o 1 zvýší
          dec       count     // count se o 1 sníží
          brne      cyklus    // Pokud se count != 0 skok na cyklus

          // Inicializace řídící proměnné cyklu
          ldi       count, N
          // Adresa pole1 do Y
          ldi       YL, LOW(pole1)
          ldi       YH, HIGH(pole1)
          // První adresa za posledním prvkem v pole2 do X
          ldi       XL, LOW(pole2+N)
          ldi       XH, HIGH(pole2+N)
          // Pomocí adresování s offsetem uložíme do pole1_2 = Y[2] (= pole1[2])
          ldd       pole1_2, Y+2

// Cyklus pro kopie do pole2
kopiePole2:
          // Do temp se uloží aktuální pozice v pole1, Y se o 1 zvýší
          ld        temp, Y+
          // Sečtení temp = temp + pole1_2
          add       temp, pole1_2
```

```
            // X se o jednu sníží, poté se na tuto novou pozici uloží temp
            // Před cyklem se do X uložila adresa za pole2, ale protože se X nejprve
            // sníží, pole1[0] se uloží na poslední pozici pole2 (pole2[N-1])
            st          -X, temp
            // count se o 1 sníží
            dec         count
            // Pokud se count != 0 skok na kopiePole2
            brne        kopiePole2

end:        rjmp        end         // Nekonečná smyčka
```

DCV č. 2

- Výpočet v předchozím programu upravte podle následujícího předpisu:

  `pole2[N-i] = pole1[i] + pole2[K], i = 0,1, ... N`

  kde `K` je konstanta známá v době psaní programu (`.equ`)

```
            .include   "m32def.inc"

// Konstanty
.equ        N           = 10
.equ        PRVNI       = 22
.equ        K           = 3

// Nové pojmenování registrů
.def        count       = r16
.def        temp        = r17
.def        pole2_K     = r18

// Proměnné
            .DSEG
pole1:      .BYTE       N
pole2:      .BYTE       N

            .CSEG
            // Uložení ukazatele na pole1 do X
            ldi         XL, LOW(pole1)
            ldi         XH, HIGH(pole1)
            // Naplnění řídící proměnné cyklu
            ldi         count, N
            // Inicialicaze proměnné pro plnění pole
            ldi         temp, PRVNI
// Cyklus pro naplnění pole1 hodnotami PRVNI, PRVNI+1, atd. ...
cyklus:
            // Uložení temp na aktuální pozici v poli1
            st          X+, temp
            // Zvýšení temp o 1
            inc         temp
            // Zmenšení count o 1
            dec         count
            // Není-li temp roven 0 skok na cyklus
            brne        cyklus
```

```
            // Naplnění řídící proměnné cyklu
            ldi         count, N
            // Uložení první adresy pole1 do X
            ldi         XL, LOW(pole1)
            ldi         XH, HIGH(pole1)
            // Uložení první adresy za pole2 do Y
            ldi         YL, LOW(pole2+N)
            ldi         YH, HIGH(pole2+N)
            // Načtení pole2[K] do pole2_K
            // K první adrese pole2 si překladač přičte K, výsledek bude adresa pole2[K]
            lds         pole2_K, pole2+K
// Kopie do pole2
kopiePole1:
            // Načtení aktuálního prvku v pole1 do temp, ukzatel se o 1 zvýší
            ld          temp, X+
            // Sešetní temp = temp + pole2_K
            add         temp, pole2_K
            // Snížení ukazatele ne pole2 o 1 a na tuto novou pozici se uloží temp
            // Před cyklem je do Y vložena adresa o 1 vyšší než se poslední prvek v pole2,
            // protože se Y nejdříve o 1 sníží, tím se dostane na poslední prvek v pole2
            st          -Y, temp
            // Odečtení 1 od count
            dec         count
            // Není-li count roven 0 skok na kopiePole1
            brne        kopiePole1

end:        rjmp        end         // Nekonečná smyčka
```

DCV č. 3

- Do předchozího programu doplňte výpočet součtu všech prvků obou polí dohromady. Výsledek uložte do zvláštní proměnné.

```
            .include   "m32def.inc"

// Konstanty
.equ        N           = 10
.equ        PRVNI       = 1
.equ        K           = 3

// Nové pojmenování registrů
.def        count       = r16
.def        temp        = r17
.def        pole2_K     = r18
.def        soucet      = r19

// Proměnné
            .DSEG
pole1:      .BYTE       N
pole2:      .BYTE       N
soucetProm:             .BYTE       1
```

```
            .CSEG
            // Uložení ukazatele na pole1 do X
            ldi         XL, LOW(pole1)
            ldi         XH, HIGH(pole1)
            // Naplnění řídící proměnné cyklu
            ldi         count, N
            ldi         soucet, 0
            // Inicialicaze proměnné pro plnění pole
            ldi         temp, PRVNI
// Cyklus pro naplnění pole1 hodnotami PRVNI, PRVNI+1, atd. ...
cyklus:
            // Uložení temp na aktuální pozici v poli1
            st          X+, temp
            add         soucet, temp    // Přičtení k soucet temp (soucet += pole1[count])
            inc         temp        // Zvýšení temp o 1
            dec         count       // Zmenšení count o 1
            brne        cyklus      // Není-li temp roven 0 skok na cyklus

            // Naplnění řídící proměnné cyklu
            ldi         count, N
            // Uložení první adresy pole1 do X
            ldi         XL, LOW(pole1)
            ldi         XH, HIGH(pole1)
            // Uložení první adresy za pole2 do Y
            ldi         YL, LOW(pole2+N)
            ldi         YH, HIGH(pole2+N)
            // Načtení pole2[K] do pole2_K
            // K první adrese pole2 si překladač přičte K, výsledek bude adrese pole2[K]
            lds         pole2_K, pole2+K
// Kopie do pole2
kopiePole2:
            // Načtení aktuálního prvku v pole1 do temp, ukzatel se o 1 zvýší
            ld          temp, X+
            add         temp, pole2_K   // Sečtení temp = temp + pole2_K
            // Sečtení soucet s temp (soucet += pole1[count] + pole2[K])
            add         soucet, temp
            // Snížení ukazatele ne pole2 o 1 a na tuto novou pozici se uloží temp
            // Před cyklem je do Y vložena adresa o 1 vyšší než se poslední prvek v pole2,
            // protože se Y nejdříve o 1 sníží, tím se dostane na poslední prvek v pole2
            st          -Y, temp
            // Odečtení 1 od count
            dec         count
            // Není-li count roven 0 skok na kopiePole2
            brne        kopiePole2

            // Uložení výsledného součtu do proměnné soucetProm
            sts         soucetProm, soucet

end:        rjmp        end         // Nekonečná smyčka
```