Digitális technika HF2
Elkészítési segédlet
Gépi szintű programozás

A programozási feladat egy adott probléma 3 féle megoldásának elkészítése. Mindegyik program lehet egyetlen közös forrásfáljlban, a megoldás során aktívan használva a blokkos megjegyzés be- és kikapcsolás lehetőségét. A program neve legyen HF2.s.

A program egyetlen perifériát használ, ez a kimeneti LED kijelző, a követelmények teljesítéséhez elegendő ennek az egyetlen perifériának a regiszterét deklarálni.

```
;*******************************************************************************
;* Digitális technika VIMIAA01 HF2 mintamegoldás                               *
;* Adatvektor aktív egyes bitjeinek megszámolása,                              *
;* eredmény kijelzése a LED perifférián                                        *
;*******************************************************************************

DEF LD    0x80                ; LED adatregiszter          (írható/olvasható)
```

A programok a memóriában tárolt tesztadatsorozat elemeti dolgozzák fel.

A tesztadat sorozat beolvasható lenne adatfájlból is, de a HF2 megoldása során elegendő a forráskódban megadni a tesztadatokat. A tesztadatok az adat szegmensbe kerülnek, az adatszegmens kezdőcíme legyen DIGIT_CODE és ettől címtől kezdve töltsük fel a feladat előírásai szerint átalakított 7 darab adatbájtot, ami a személyes DIGIT kód számjegyeit és a szomszédos számok összegét tartalmazza hexadecimális formátumban, a végén egy nulla értékkel. Tehát ha a minta DIGIT_KÓD 1234567, akkor a generált teszt adatvektor 132537495B6D70 hexadecimális digitsorozat, kétdigites hexa bájtok formájában a memóriába a következőképpen írható:

```
    DATA

;*******************************************************************************
;* Az adatvektor tárolása a memóriában                                         *
;*******************************************************************************
DIGIT_CODE
    [redacted]13, 0x25, 0x37, 0x49, 0x5B, 0x6D, 0x70
```

A feladat kiírás szerint a programverziók hatékonyságvizsgálatába ennek a teszt adatvektornak a memória foglalását nem kell beleszámítani, csak azokat a memóriabiteket/bájtokat, amiket a program az adott algoritmus megvalósítása miatt használ.

A programok keretrendszere:

Mindhárom program azonos feladatot lát el, ezért bizonyos általános jellemzők azonosan kezelhetők. Ez a javítást és a programok értelmezését is megkönnyíti, ezért kérjük, hogy a program bemeneti paramétereinek kezelésekor, a végeredmény megadásakor és a közbenső részeredmények számításakor az alábbi regiszterhasználati előírást tartsuk be:

```
;*************************************************************************
;* A HF2: Adatvektor 1 értékű bitjeinek megszámolása                     *
;* Regiszter használat: r0=Adatpointer r1=Adatelem számláló, r2=Bitszámláló *
;*                      r8 - r15 =Munkaregiszterek                       *
```

A programok elkészítésénél a MiniRISC utasításkészlet bármely utasítása tetszőleges módon használható, azonban minden utasítássor tartalmazzon értelmezhető magyarázó megjegyzést.

HF2_1. Hagyományos algoritmus (HAGY.s)

A hagyományos algoritmus a feladatot a legelemeibb módon, az előírások szerint hajtja végre. Aprogram az induláskor a bemeneti paramétereknek megfelelően inicializálja a program változókat, majd bájtról bájtra beolvassa az adatvektor hexadecimális digitpárjait. Itt kihasználja a hatékony indirekt adatvektor címzési lehetőséget:

```
read:
    mov r8,[illegible]          ; Aktuális  adatelem betöltése memóriából
```

Az adatbájtok 1 értékű bitjeit egy bitszámláló hurokban számolja meg:

```
bit_loop:




    j[illegible]    bit_loop            ; Ismétlés, amig lesz következő bit
```

A szükséges adminisztráció és az aktuális feltételek kiértékelése után az előző műveleteket ismétli az adatbájtok egymás utáni beolvasásásával, majd a végeredményt a LED perifériára írja.

```
    j[illegible]    read                ; Új adatelem olvasása
    mov     LD, r2                      ; Összeg kijelzése
```

Mivel itt az adatmemóriába az algoritmus végrehajtását segítő területfoglalás nem történt a DH=0 értéket használva a programköltség egyszerűen számítható és a címoldal táblázatába írható. A dokumentációba kérjük bemásolni a lefordított program HAGY.lst, listafájl „lényeges részét”, ami pontosan mutatja a program paramétereit. (Itt a lényeges részeket kitakartuk, a feladat érdekességének megőrzése érdekében.)

```
C 00            start1:
C 00    C000        mov     r0, #DIGIT_CODE[00]     ; Az adatvektor mutató betöltése
C 01    C107        mov     r1,[illegible]          ; Adatvektor bájtok száma
C 02    C200        mov     r2,[illegible]          ; A bitszámláló előkészítése

C 03            read:
C 03    [illegible]     mov     r8, [illegible]     ; Aktuális  adatelem betöltése memóriából
C 04    [illegible]
C 05            bit_loop:
C 05    [illegible]
C 06    [illegible]
C 07    [illegible]     j       bit_loop[05]        ; Ismétlés, amig lesz következő bit
C 08    [illegible]
C 09    [illegible]
C 0A    [illegible]
C 0B    [illegible]     j       read[03]            ; Új adatelem olvasása
C 0C    9280        mov     LD[80], r2              ; Összeg kijelzése
```

HF2_2. Táblázatos algoritmus (TABL.s)

Sok algoritmus esetében jelentős gyorsítás érhető el, ha a megoldás során bizonyos részszámításokat előre kiszámított és táblázatban rögzített eredmények alapján építünk be. A megoldás előnye akkor használható, ha a használt utasításkészletben létezik olyan utasítás, amely képes adatfüggő címzéssel kezelni a memóriát. Ebben az esetben egy táblázat bázis cím és egy aktuális ofszet használatával az adott részadathoz tartozó részeredmény egy lépésben megkapható. A módszer két használhatatlan verziója:

1. A részadat mérete 1 bit: A táblázat mérete 2 bájt: Az első bájton tárolt részösszeg 0, a második bájton tárolt részösszeg 1. A Báziscím + ofszet címzéssel tehát 0 értékű bitre 0 részösszeget, 1 értékű bitre 1 részösszeget olvasunk vissza és használunk fel a végösszeg kiszámításakor. Ez egy egyáltalán nem hatékony megoldás.
2. A részadat mérete a teljes adatvektor, a feladat szerint 56 bit. A táblázat mérete 2^{56} bájt lenne, és a közvetlen kiolvasással egy lépésben megkapnánk az eredményt. Ez nyilvánvalóan nem megvalósítható megoldás. (Persze a memóriaméret csökkenthető, mert egyrészt nincs minden bináris kombináció kihasználva, másrészt a tesztadatot a digitkód egyértelműen meghatározza és ezek száma < 10^6, sőt az évfolyam létszám pedig kisebb 10^3, de most nem ez az érdekes.)

A hatatékony megoldáshoz tehát egy kedvező méretű részadatot és ezáltal realizálható méretű táblázatot kell meghatároznunk. Mivel a MiniRISC rendszerben az adatmemória mérete 128 bájt, ezért a részadat mérete biztosan kisebb 7 bit. A lehetséges értékek közül a 4 bit tűnik egy jó választásnak, azért is, mert az előállítása a 8 bites adatelemekből viszonylag egyszerűen megoldható.

Ezek után az program feladata: Az adatmemóriában előkészít egy táblázatot a SUM_LUT címkétől kezdve a szükséges számú bájton, ami tartalmazza az egyes hexadecimális számjegyek bináris formájában található egyes bitek számát. Tehát a táblázat 5. eleme 1, a 12. eleme 3.

```
    DATA

;*****************************************************************************
;* Az adatvektor tárolása a memóriában                                       *
;*****************************************************************************
DIGIT_CODE
    [illegible]3, 0x25, 0x37, 0x49, 0x5B, 0x6D, 0x70

    org 0x10;
SUM_LUT
    [illegible] 0x00, [illegible] 0x01,
    [illegible] 0x02, [illegible] 0x03 [illegible]
```

A táblázat az *org* 0xnn direktívával az általunk kijelölt címre helyezhető. (Ez lesz a SUM_LUT értéke).

A program az induláskor a bemeneti paramétereknek megfelelően inicializálja a program változókat, majd bájtról bájtra beolvassa az adatvektor hexadecimális digitpárjait. Itt kihasználja a hatékony indirekt adatvektor címzési lehetőséget:

```
read:
    mov r8,[illegible]          ; Aktuális  adatelem betöltése memóriából
```

Az adatbájtból először az alsó 4 bites hexa digitet kimaszkolva hozzáadja a SUM_LUT táblázat bázis címéhez, kiolvassa a táblázat adott címhez tartozó értéket és hozzáadja a részösszeget a végösszeget tartalmazó regiszterhez, majd az adatbájtot újraolvasva, digitjeit megcserélve ugyanezt még egyszer végrehajtja. A műveletet addig ismétli, amíg minden adatbájtot feldolgoztunk.

```
read:
    mov     r8, (r0)            ; Aktuális adatelem betöltése
    and     r8, #0x0f           ; Alsó 4 bit maszkolása
                                ; [illegible] kezdőcím betöltlése
                                ; Táblázat [illegible] számítása
                                ; Részösszeg [illegible]
                                ; [illegible] összegzése
    mov     r8, (r0)            ; Aktuális adatelem újra betöltése
[illegible]                     ; Alsó-felső digit csere
    and     r8, #0x0f           ; Alsó 4 bit maszkolása
                                ; Táblázat kezdőcím betöltlése
                                ; Táblázat[illegible] számítása
                                ; Részösszeg [illegible]
                                ; Részösszeg összegzése

    j[illegible]    read        ; Új adatelem olvasása
```

Mivel itt az adatmemóriába helyezett táblázatnak az algoritmus végrehajtása szempontjából jelentős szerepe van, ezért a programköltség szempontjából jogos, hogy figyelembe vesszük. Tehát a SUM_LUT méretét a címlapon lévő táblázatban adjuk meg és a képlet szerint számoljunk vele. A dokumentációba kérjük bemásolni a lefordított program TABL.lst, listafájl „lényeges részét”, ami pontosan mutatja a program paramétereit. (Hasonlóan a korábbi HAGY.lst példához.)

HF2_3. Aritmetikai algoritmus (ARIT.s)

Az aritmetikai algoritmus a bináris összeadás tulajdonságain alapul, hatékony SIMD (Single Intsruction, Multiple Data, Egyetlen utasítás több adaton is egyszerre végrehajtva) jellegű párhuzamosítást használ. A számítás során először az 1 bit méretű bemeneti operandusokat adjuk össze 1 bites teljes összeadókkal, ekkor ezekből 2 bit méretű részeredmények {co,s} képződnek. Ezeket páronként 4 bites összeadókkal összegezzük, a keletkező 3 bites részösszegeket újra 4 biten összegezzük és megkapjuk egyetlen adatbájt 1 értékű bitjeinek számát.

Az algoritmus ötlete a 8 bites ALU keskenyebb, 2 bites/4 bites összeadókra osztásának módszerén alapul, annak kihasználásával, hogy összeadás esetém maximum 1 bit az átvitel, tehát ha legalább egy 0 van két kisebb méretű összeadó között, akkor a két művelet nem zavarja egymást, a feladat elvégezhető.

```
;*******************************************************************************
;* A HF2: Adatvektor 1 értékű bitjeinek megszámolása                           *
;* Regiszter használat: r0=Adatpointer r1=Adatelem számláló, r2=Bitszámláló    *
;*                      r8 - r15 =Munkaregiszterek                             *
;* 3. verzió: Az adatvektor elemeinek párhuzamos összegzésével                 *
;*******************************************************************************
```

A program az induláskor a bemeneti paramétereknek megfelelően inicializálja a program változókat, majd bájtról bájtra beolvassa az adatvektor 8 bites részeit. Itt kihasználja a hatékony indirekt adatvektor címzési lehetőséget, a korábbi megoldásokhoz hasonlóan.

```
read:
    mov r8, [illegible]          ; Aktuális  adatelem betöltése memóriából
```

Mivel az első 1 bit széles összeadásokhoz a páros-páratlan biteket szét kell válogatni, ezért egy munkaregiszterbe másolatot készít. Az eredeti példányt és a másolatot a megfelelő maszk értékekkel úgy maszkolja, hogy egyikben csak a páros, másikban csak a páratlan bitek maradnak érvényesek, a többiek nullázódnak. Ezután a páratlan biteket tartalmazó regisztert jobbra léptetjük és a két részeredményt összeadjuk. Ugyanezen elv alapján maszkolunk 4 db 2 bit széles bitmezőt, az egyiket 2-vel jobbra léptetjük, összeadjuk. Végül az így képzett részösszegeket egy kis pozícionálás után összeadjuk és készen van egy adatelem eredménye.

Az első ütemben tehát ez a következőképpen működik:

```
    mov r9, r8              ; Másolat r9-be
    and r8, #0x55           ; Páros bitek maszkolása
    and r9, #0xAA           ; Páratlan bitek maszkolása
    sr0 r9                  ; és léptetése jobbra
    add r8, r9              ; Páros-páratlan bitek összegzése
```

Eza alapján a további lépések már könnyen kódolhatók. Ezt a ciklust is addig ismételjük, amíg van feldolgozandó adat.

```
    j[illegible] read           ; Új adatelem olvasása
    mov LD, r2              ; Összeg kijelzése
```

A beadandó HF2 dokumentáció tartalmazza a megfeleően kitöltött és sajátkezű aláírás fotóját/bemásolását tartalmazó képpel kiegészített címlapot a munka rövid értékelésével. A feladatkiírás szerint az egyes részfeladatok ASM folymatábráját is kérjük felrajzolni, vagyis az egy bájt bitjeinek megszámolását végző algoritmust. Ez lehet kézi rajz is, de szép, olvasható formában bemásolva. Minden részfeladatnál az xyzv.lst fájlban a programsorok mindegyike tartalmazzon magyarázó megjegyzést.