Mikrokontroller alapú rendszerek - BMEVIAUAC06
Házi feladat dokumentáció

Csizy Ádám
******

Feladat kiírás

Belső memóriában tárolt 16 bites kódszavak (tömb) közötti legkisebb Hamming távolság megkeresése. Bemenet: tömb kezdőcíme (mutató), elemek száma. Kimenet: legkisebb távolság számértéke (regiszterben).

Használt fejlesztőkörnyezet: Silicon Labs Simplicity Studio
Használt fejlesztőkártya: Silicon Labs EFM8BB3 Busy Bee Starter Kit

Alkalmazott programozási logikák

Mivel az alkalmazott architektúra (memória, általános regiszterek) bájt szervezésű, ezért a kezelt 16 bites kódszavakat, továbbá az egyéb, 8 bitet meghaladó méretű adatszerkezeteinket több bájton, folytonosan helyezzük el a memóriában az Intel *little endian* elvét szem előtt tartva (alacsonyabb helyiértékű bájt az alacsonyabb című memória szegmensben kerül elhelyezésre, adatfeldolgozás esetén mindig az alsó bájtot dolgozzuk fel elsőként stb.). Az alkalmazott algoritmusok közül mindig a gyakrabban használt (esetünkben ez a belső ciklus - szubiteráció) számára tartjuk fenn a CPU regisztereit, a használat szempontjából alacsonyabb prioritású ciklusaink változóit az adatmemóriában tároljuk el (további alternatíva a regiszter bank váltás).

Adatszerkezetek

A specifikáció nem nyilatkozik a kódszavak (adatvektor) módosíthatóságának mivoltáról, se a belső memórián belüli elhelyezéséről, ezért memóriatakarékossági megfontolásból a kódszavakat a program memóriában (CODE SEGMENT) konstansokként helyezzük el. A tömbelemekhez való hozzáférést bázisregiszter indexelt címzés használatával valósítjuk meg (MOVC utasítás és DPTR használata).

A programunkban használt változóknak (algoritmusokban használt lokális változók, a kitűzött feladat végeredményének eltárolása stb.) az adatmemóriában (DATA SEGMENT) foglalunk helyet az alsó nem bitcímezhető tartománytól kezdődően (0x30 hexa cím).

Mind az adatmemória, mind a kódmemória esetén az inicializációt és allokációt általános célú szegmensek létrehozása és kiválasztása előzi meg, szem előtt tartva ezzel az esetleges későbbi program és memóriatartalom módosítás során felmerülő memória átszervezési szempontokat.

Algoritmusok

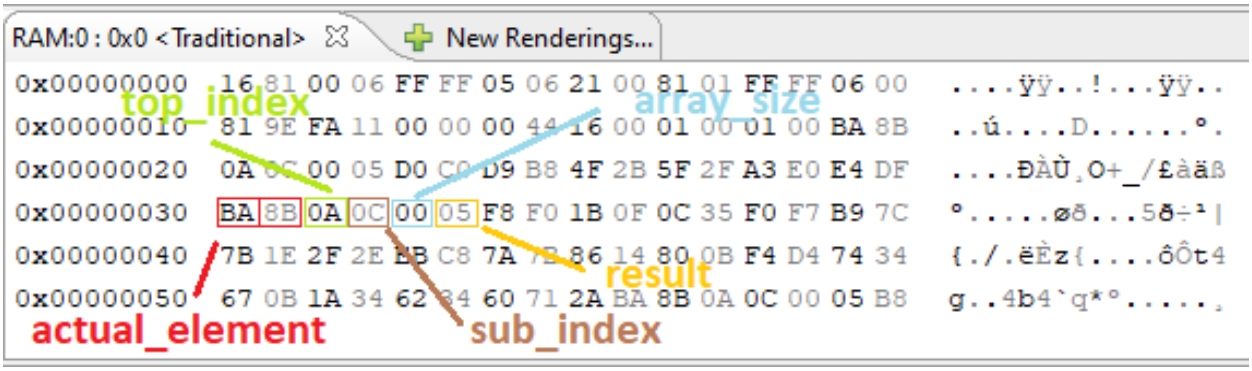
A kitűzött feladat megoldását egy szubrutinba kiszervezett algoritmus valósítja meg, melyet a főprogramból hívunk, majd a kapott végeredményt a főprogramon belül az adatmemóriába mentjük el. Az algoritmus három fő részre bontható: egy külső iterációs ciklusra (topiteráció), egy az előbb említett iteráción belül elhelyezkedő belső iterációs ciklusra (szubiteráció) és a minimális Hamming távolságot

számító logikai magra. A minimális Hamming távolság számítás a kódszavak összehasonlításán alapul. Valamennyi kódszót össze kell hasonlítani az összes többivel a lehető leghatékonyabb módon, redundáns műveletvégzés nélkül. Ennek az összehasonlításnak a szervezését végzi a két egymásba ágyazott iterációs ciklus. A külső iteráció (topiteráció) az adatvektor legelső elemétől indul (egyik összehasonlítandó kódszó), majd egyesével a magasabb kódmemória címek irányába haladva végig lépked a kódszavakon egészen az utolsó előtti kódszóig (erre azért van szükség, hogy az utolsó kódszót ne hasonlítsuk össze önmagával, ekkor ugyanis a minimális Hamming távolság 0-nak adódik, így hamis eredményt kapunk). A belső iteráció (szubiteráció) a külső iteráció (topiteráció) aktuális elemét követő kódszótól (másik összehasonlítandó kódszó) kezdve a magasabb kódmemória címek irányába haladva végig lépked a kódszavakon egészen az utolsó kódszóig. Az iterációkban a bázisregiszter indexelt címzés indexének növelésével valósul meg a léptetés (egy iterációs cikluson belül kétszer léptetjük az indexet – alsó és felső bájt behozatal). A belső ciklus a külső ciklus aktuális indexét veszi át, mint kezdő indexet (valójában közvetlenül átvétel után még megnöveljük eggyel az első ciklus indítása előtt, hogy értékhelyes bájtot címezzünk meg). Az iterációk leállási feltétele a tömb végének (legfelső (szubiteráció) vagy az az előtti elemének (topiteráció)) elérése, melyet a tömb méretének, mint bemeneti adatnak egy lokális változóba történő másolásával, ezen változó értékének ciklusonkénti csökkentésével, majd pedig null értékének vizsgálatával valósíthatunk meg (DJNZ utasítás). Így valamennyi kódszót össze tudjuk hasonlítani anélkül, hogy ugyan azt a két kódszót többször is vizsgálnánk. Magát az összehasonlítást és a Hamming távolság meghatározását a belső iterációba ágyazott logikai mag végzi. A Hamming távolság két kódszó esetén a kódszavak megfelelő bitenként vett eltéréseinek összegét jelenti. Ez a legegyszerűbben a kizáró vagy művelettel (XRL utasítás) végezhető el. Az eredmény az akkumulátorban képződik, a benne lévő egyes biteket kell összegezni. Az összegzést az akkumulátor tartalmának jobbra (vagy balra) forgatásával (esetünkben RRC utasítás) valósíthatjuk meg. A legfelső bit helyére ilyenkor a CY flag értéke kerül, a köztes bitek értéke eggyel jobbra tolódik a legalsó bit értéke pedig a CY flagbe kerül, melynek értékét könnyen vizsgálhatjuk és CY = 1 esetén feltételes ugrás használatával (JC vagy JNC utasítások) egy rögzített 8 ciklusszámú mikrocikluson belül összegezhetjük egy erre a célra használt változóban (R2 regiszter). Az összehasonlítandó kódszavak felső bájtjai esetén is hasonlóan járunk el. Az erre a célra használt változónk tartalmát (aktuális Hamming távolság) összehasonlítjuk egy másik változó tartalmával, melyben az eddig kiszámított legkisebb Hamming távolságot tároljuk (R6 regiszter – alapértelmezett értéke a legnagyobb távolság: 0x10 = 16). Ha az aktuális érték kisebb az eddigi legkisebbnél, akkor frissítjük az értékét. Mivel nem áll rendelkezésre aritmetikai relációt vizsgáló utasítás, ezért egy kivonás segítségével (SUBB utasítás) vizsgáljuk a két távolság különbségét: R2 regiszter és R6 regiszter értékének különbsége. Negatív különbség esetén az akkumulátor legfelső 7. bitjén egy úgynevezett „borrow” átvitel keletkezik, mely a CY flaget 1 értékűre állítja (a CY flaget már előzőleg nulláztuk), melyet könnyen vizsgálhatunk és szükség esetén az eddigi legkisebb Hamming távolság értéke felül írható (R6 regiszter értéke). A szubrutinból való visszatérést követően a főprogramban az adatmemóriába mentjük a rutin kimenetét (R6 regiszter értéke), majd a location counter ($ szimbólum) használatával egy végtelen ciklusban várakozunk.

Tesztelés

Bemeneti teszt adatok:
16 bites kódszavak: 0x14CD, 0x75A3, 0x1004, 0xE54C, 0x8BBA, 0xFA21
Adatvektor mérete: 0x06

A program magja az elvártnak megfelelően, valós időben lefutott, értékhelyes megoldást kaptunk.



A program futtatását követő adatmemória kép

Látható, hogy az adatmemóriában lefoglalt változóink sorrendje a programtörzsbeli deklaráció sorrendjével egyezik meg, elhelyezkedésük a 0x30 címtől folytonos. A *result* változóban a kapott végeredmény (minimális Hamming távolság) található, mely megegyezik az *R6* regiszter tartalmával. Az *array_size* változó tartalma 0x00, mivel a külső ciklus (topiteráció) leállási feltétele az átvett adatvektor iterációként csökkentett méretének zérus értéke. A *sub_index* változó a belső ciklusban (szubiteráció) utoljára vizsgált kódszó bázis + index regiszteres címzés indexének eggyel megnövelt értékét tartalmazza (az alkalmazott programozási logika miatt), mely bájt szervezésű memória és egy 6 elemű 16 bites adatvektor esetén 11 + 1 = 12 = 0x0C értéknek felel meg. A *top_index* változó tartalmával is hasonló a helyzet, azzal a különbséggel, hogy itt az adatvektor utolsó előtti elem indexének eggyel megnövel értékét láthatjuk (szintén az alkalmazott programozási logika miatt), mely az előbb említett adatvektor és architektúra esetén 9 + 1 = 10 = 0x0A értékűnek adódik. Az *actual_element* változó 2 bájton tartalmazza a külső ciklusban (topiteráció) utoljára vizsgált kódszót (0x8BBA) *little endian* bájtsorrendnek megfelelően.

A feladat megoldásához felhasznált források

- A tanszéki weboldalon fellelhető segédanyagok (https://www.aut.bme.hu/Course/VIAUAC06).
- A Keil C51 fordítóprogram-család hivatalos weboldalán közzétett utasításkészlet leírás (http://www.keil.com/support/man/docs/is51/is51_opcodes.htm).

Kijelentem, hogy a kiadott feladatot meg nem engedett segítség nélkül, saját magam oldottam meg, az elkészített forráskód valamint a hozzá tartozó dokumentáció a saját szellemi termékem.