Labda mozgása

Heiszman Henrik
Neptun kód: ENV2R9
Pázmány Péter Katolikus Egyetem, Információs Technológiai és Bionikai Kar
1083 Budapest, Práter utca 50/A
heiszman.henrik@hallgato.ppke.hu

Téma–Labda mozgásának modellezése regiszterek értékének változtatásával, előre meghatározott feltételek mellett.

I. A JEGYZŐKÖNYVBEN HASZNÁLT FOGALMAK

Ciklus: más néven iteráció, az ismétlődő tevékenységek megvalósítására szolgáló eszköz.

Végtelen ciklus: olyan ciklus, melynek futása sohasem zárul le.

Vagy: logikai művelet, amely csak abban az esetben ad hamis értéket, ha a bemeneti értékek rendre nullák/hamisak.

XOR (kizáró vagy): logikai művelet, amely abban különbözik a VAGY művelettől, hogy itt abban az az esetben is hamis értéket kapunk, ha mindegyik bemeneten igaz/egyes érték érkezik. Más szóval, csak akkor kapunk igaz értéket, ha minimum egy érték eltér a többitől. Két változónál: I-H és H-I bemeneti kombinációra kapunk igaz értéket.

JMP: feltétel nélküli ugrás utasítása, akkor ugrik a program a következő szakaszra, ha a futás során elért a JMP utasításhoz, nincs feltétele az ugrásnak.

II. FELADAT LEÍRÁSA

A feladatom az volt, hogy írjak egy programot, amely mozgat egy végtelen ciklusban két előre meghatározott határ között két regiszterben eltárolt változót, ezzel reprezentálva egy labda mozgását. Kikötés volt továbbá, hogy a változás sebessége legyen egy egység cikluson kent mindkét irányban.

III. A MEGVALÓSÍTÁS MEGTERVEZÉSE

Ahhoz, hogy az elkészített program megfeleljen az elvárásainknak, több esetet kell megvizsgálni. Az egyes esetek akkor az ütközés típusától fognak függeni.

Első esetben azt a szcenáriót vettem figyelembe, amikor a „labda” függőleges „fallal” ütközik.

Második eset az, ha a „labda” vízszintes „fallal” ütközik.

Első esetben ütközés után a labda vízszintes irányú sebessége változik, a függőleges irányú pedig változatlan marad.

Második esetben az ütközés során a labda függőleges irányú sebessége változik, a vízszintes irányú pedig változatlan marad. (Pont fordítva, mint az első esetben.)

Esetek vizsgálata során már csak egy végtelen ciklusra van szükség, amelyen belül végzi el a program a mozgatást és annak vizsgálatát, hogy történt-e ütközés, és ha igen, akkor milyen típusú (horizontális vagy vertikális).

IV. A PROGRAM ELKÉSZÍTÉS

A programot az előző részben leírt gondolatmenetem alapján valósítottam meg.

Az elkészült programkód a következő ábrán látható. *(1. ábra)*

```
main:
    // legyen 10x10 a jatekmezo --> x=[0;10] y=[0;10]
    mov #0,R4 // x koordináta
    mov #0,R5 // y koordináta
    mov #1,R6 // x sebessége
    mov #1,R7 // y sebessége
    mov #0,R8
    mov #10,R9

 ujra:

    add R6,R4
    add R7,R5

    //elso eset, ha oldalfallal utkozik (x=0 || x=10)
     //változzon meg az y iranyu sebessege -1-szeresére
    cmp R8,R4
    jnz NoAction
    xor #0xffff,R7
    inc R7

NoAction:
    cmp R9,R4
    jnz Action
    xor #0xffff,R7
    inc R7

Action:

    //második eset, ha fent vagy lent utkozik (y=0 || y=10)
    //változzon meg az x iranyu sebessege -1-szeresére
     cmp R8,R5
    jnz NoAction_
    xor #0xffff,R6
    inc R6
NoAction_:
    cmp R9,R4
    jnz Action_
    xor #0xffff,R6
    inc R6
Action_:

jmp ujra
      ret
```

1. ábra
A megvalósított program kódja

V. A KÓD MAGYARÁZATA

Ebben a részben az elkészített program magyarázatát írom le részekre bontva.

Első rész: változók és konstansok tárolása

```
mov #0, R4
mov #0, R5
mov #1, R6
mov #1, R7
mov #0, R8
mov #10, R9
```

Az R4 és R5 regiszterbe eltároltam a mozgatni kívánt változókat. Mivel a feladat leírása nem határozta meg, hogy mi legyen ezeknek a kezdő értéke, így én mind a kettőnek a 0-t adtam meg. Ezen kívül az R6-ban eltároltam az R4 sebességet és R7-ben pedig eltároltam az R5 sebességét. Ezek kezdeti értékek (R4, R5, R6, R7), mivel ezek a program futása közben folyamatosan változni fognak. Ezen kívül az R8 és R9 regiszterekben eltároltam 0-t és 10-et. Erre azért volt szükség, mert a feladat leírásában nem szerepelt a határ melyen mozgatni kell az értékeket. Én önkényesen úgy döntöttem, hogy az egy 10x10-es játékmező legyen.

Továbbiakban a cikluson belüli tartalmat magyarázom el. A ciklusnak az „ujra” címkét adtam. Ez a ciklus gondoskodik arról, hogy a labda mozogjon a játékmező.

Ciklus első eleme: mozgatás

```
add R6,R4
add R7,R5
```

Az R4 és R5 regiszterekhez egyenként hozzáadtam a megfelelő sebességet összeadás utasítással úgy, hogy az eredmény az R4 és R5 regiszterekbe kerüljön. Gyakorlatilag mozgattam a pontot egy egységgel.

Ciklus második eleme: első eset vizsgálata (ütközés függőleges fallal)

```
cmp R8,R4
jnz NoAction
xor #0xffff,R7
inc R7

NoAction:
cmp R9,R4
jnz Action
xor #0xffff,R7
inc R7

Action:
```

Abban az esetben, ha a labda függőleges fallal ütközik, akkor az x koordinátája (R4 értéke) vagy egyenlő nullával vagy tízzel. Ilyenkor (ha ez a feltétel teljesül) az y irányú sebességét (R7 regiszter értéke) -1-szeresére kell változtatni, így a ciklus a következő körében a labda a pattanásnak megfelelő irányba fog mozogni. A -1-gyel való szorzást két lépésben, a kettes komplemensnél tanultak alapján, oldottam meg: xor műveletet végeztem a 0xffff és az R7 regiszterrel, majd inkrementáltam R7-ot.

Az első egyenlőség vizsgálatakor ha R8 és R4 (x és 0) nem egyenlő, akkor a NoAction részhez ugrik a program, ahol most az R9 és R4 (x és 10) egyenlőséget vizsgálja. Ha ez a feltétel sem igaz, akkor a program az Action részhez ugrik és elkezdi a második eset vizsgálatát, ami azt nézi, hogy történt-e a vízszintes fallal ütközés.

Ciklus harmadik eleme: második eset vizsgálata (ütközés vízszintes fallal)

```
cmp R8,R5
jnz NoAction
xor #0xffff,R6
inc R6

NoAction_:
cmp R9,R4
jnz Action
xor #0xffff,R6
inc R6

Action_:
```

Ennek a résznek a működése teljes mértékben megegyezik az előző ciklusrészével. Csak a megfelelő változókat kellett kicserélni, mert ebben az esetben azt teszteljük, hogy az R5 (y) értéke megegyezik-e nullával vagy tízzel. Egyezés esetén -1-szeresre állítottam az x sebességét (R6). Abban az esetben, ha egyik feltétel (y=0 vagy y=10) sem teljesült, a program az Action_ részhez ugrott. A „cmp” utasítás működése ebben a két esetben megegyezik az előzőekben leírtakkal.

Ciklus negyedik elem: feltétel nélküli ugrás

```
jmp ujra
```

A „jmp” utasítás azt jelenti, hogy a program minden esetben ugrik. Az én programomban úgy használtam fel ezt az utasítás (jmp ujra), hogy az „ujra” részhez ugorjon a programom. Ez azt eredményezi, hogy a program előre kezdi a ciklus és újra elvégzi a benne leírt utasításokat (mozgatás és esetek vizsgálata). A feltétel nélküli ugrásra azért volt szükség, mert a feladatban foglaltak alapján, végtelen ciklust kellett létrehozni.

VI. AZ ELKÉSZÜL PROGRAM ELLENŐRZÉSE

Az elkészült programomat lépésenkénti futtatással ellenőriztem.

Azzal, hogy a program minden körben eggyel növeli (adott esetben csökkenti) mind a két koordináta értékét (R4 és R5) és a játékmező egy 10x10-es négyzet, a labda folyamatosan átlósan fog mozogni. Ebből adódóan a futást demonstráló ábrákon csak az érdekes eseteket szemléltetem. Ilyen eset az indítás (ciklus első köre), egy állapot, amikor megváltozott a pozíció és az ütközés utáni értékek.*(2-5. ábra)*

R4	0x0001
R5	0x0001
R6	0x0001
R7	0x0001
R8	0x0000
R9	0x000A

2. ábra
Első kör után bekerültek a megfelelő értékek a megfelelő regiszterekbe

R4	0x0003
R5	0x0003
R6	0x0001
R7	0x0001
R8	0x0000
R9	0x000A

3. ábra
Az elvártaknak megfelelően folyamatosan változik a labda helye (R4 és R5 eggyel nőtt)

R4	0x000A
R5	0x000A
R6	0x0001
R7	0x0001
R8	0x0000
R9	0x000A

4. ábra
Az ütközés pillanata

R4	0x0009
R5	0x0009
R6	0xFFFF
R7	0xFFFF
R8	0x0000
R9	0x000A

5. ábra
Ütközés után, látszik, hogy R6 és R7 értéke -1-re váltott és R4 és R5 értéke csökkenni kezdett

Az ellenőrzés során mindent rendben találtam és minden kritériumnak eleget tett a programom.

Felhasznált források

Mérési utasítások

Logikai műveletek

Számábrázolás, mikrokontroller előadás