Mérési jegyzőkönyv

Méréstechnika és Információs Rendszerek Tanszék.

Egyszerű kalkulátor tervezése M456

című laboratóriumi gyakorlatról

A mérést végezték: Karsai Áron

Papp Dominik Edvárd

A mérést vezető oktató neve: Benesóczky Zoltán

A jegyzőkönyvet tartalmazó fájl neve: MCS4_Karsai_Papp.pdf

Felhasznált eszközök:

Mérőpanel FPGA mérőpanel	Sorszám:
Mérőpanel FPGA mérőpanel	Sorszám:

Mérési jegyzőkönyvet csak az M456 mérés utáni 1 héten belül kell leadni!

A jegyzőkönyhöz a mérési leírásban talál útmutatót.

A dokumentációt egy zip file-ba becsomagolva kérjük a HF portálra feltölteni **az M456 utolsó mérését követő 1 héten belül**. Ebben legyen benne a jegyzőkönyv pdf file-ként és az összes feladat programjai (lefordítható, működőképes verziók).

Bővebb információ: Benesóczky Zoltán benes@mit.bme.hu

CALC_1 program

Specifikáció

A CALC_1 program egy olyan számológépet valósít meg, melyet a DIP kapcsolók és a BTN0-BTN3 segítségével vezérelhetünk, kijelzője pedig az LD0-LD7 ledek.

A DIP kapcsolók segítségével megadhatunk egy-egy 4 bites *a* (7-4 kapcsolók, LSB a 4. kapcsolón) és *b* (3-0 kapcsolók, LSB a 0. kapcsolón) operandust bináris formában. Az egyes kapcsolók felső állása 1-es bitnek, alsó állása 0-s bitnek felel meg. Ezt követően a nyomógombok segítségével kiválaszthatjuk az elvégzendő műveletet: $a + b$ (BTN0), $a - b$ (BTN1), $a * b$ (BTN2) vagy a XOR b (BTN3). Az elvégzett művelet eredményét az LD7-LD0 ledeken láthatjuk a kívánt művelethez tartozó gomb megnyomását követően bináris formában, ahol LD0 az LSB. Az égő led 1-es bitnek, a sötét led 0-s bitnek felel meg. Az eredmény újabb művelet kiválasztásáig ott marad a ledeken. Mivel az összes feladatban (CALC_1-CALC_5) pozitív számokra szorítkozunk, így ha a *b* operandus nagyobb az *a* operandusnál, az minden esetben matematikailag hibás műveletnek minősül, hisz az eredmény negatív lenne, ami nem értelmezett. A CALC_1 programban ezt a hibát az összes led együttes világítása jelzi.

Felépítés

A program szerevezését tekintve egy végtelen ciklusban lekérdezi a nyomógombok állását, majd megnézi, lenyomásra került-e egy gomb. Amennyiben nem, úgy a végtelen ciklus újrakezdődik, ellenkező esetben a főprogram meghívja a megfelelő műveletet elvégző szubrutint. Az összeadás, kivonás és XOR műveletekre a MiniRISC processzor utasításkészlete tartalmaz megfelelő utasítást, azonban a szorzásért felelős algoritmust magunknak kellett lekódolni, a mérésvezető által a táblánál ismertetett bináris szorzás algoritmusa szerint.

CALC_2 program

Specifikáció

A CALC_2 program a CALC_1-hez hasonló számológépet valósít meg. Vezérélésében, felépítésében, kijelzésében, és funkcionalitásában teljesen megegyezik, egy kivétellel. A CALC_2 program a CALC_1 XOR művelete helyett az *a* / *b* műveletet valósítja meg. Az osztás eredményének kijelzésénél az LD7-LD4 ledek az egészrészt (LD4 az LSB), az LD3-LD0 ledek (LD0 az LSB) a maradékot jelzik. Természetesen, ha a *b* operandus 0, az matematikai hiba (a 0-val való osztás nem értelmezett), és ez igaz lesz az összes további (CALC_3-CLAC_5) programra is. A CALC_2 programban ezt a hibát – a negatív eredményt adó kivonás esetéhez hasonlóan – az összes led világítása jelzi.

Felépítés

A program felépítése annyiban módosult a CALC_1-hez képest, hogy az XOR műveletet megvalósító szubrutint egy bináris osztó szubrutinra kellett cserélni. Ezt az algoritmust is magunknak kellett lekódolni, a mérési útmutatóban megadott melléklet alapján.

A matematikai hibát a CALC_2 és CALC_3 programokban a hibát észlelő szubrutin visszatérése után a Z flag 1-es értékével jelzi.

CALC_3 program

Specifikáció

A CALC_3 program a CALC_2-höz hasonló számológépet valósít meg. Vezérélésében, felépítésében és funkcionalitásában teljesen megegyezik. Az eltérés a kijelzés módjában rejlik. A CALC_3 programban a DIP kapcsolókon beállított *a* operandust a hétszegmens kijelző DIG3 digitjén, a *b* operandust pedig a hétszegmens kijelző DIG2 digitjén jeleníti meg hexadecimális formában.

A kiválasztott művelet elvégzését követően az eredmény a hétszegmens kijelző DIG1-DIG0 digitjein jelenik meg (DIG0 a kisebb helyiérték) szintén hexadecimális formában. Osztás esetén a DIG1 digiten az egészrész, a DIG0 digiten a maradék látható. A kivonásnál és osztásnál felmerülő előzőekben tárgyalt matematikai hibákat most az eredményt jelző digiteken az „EE" betűkombináció jelzi. Egy adott művelet eredménye ezúttal is a következő műveletig ott marad.

Felépítés

A program felépítése az előzőekhez képest annyiban módosult, hogy a kijelzést is külön szubrutin végzi, amelyet a főprogram hív. A kijelzésért felelős *basic_display* szubrutint itt részletezzük, mivel a későbbiekben részletesebben bemutatandó CALC_4 és CALC_5 programok is ezt a szubrutint használják.

A *basic_display* szubrtuin

A szubrutin bemenetként az R7 regiszterben 4-4 biten megkapja a DIG3 és DIG2 digiteken kijelzendő értéket (ebben a sorrendben), valamint az R6 regiszterben ugyanígy a DIG1 és DIG0 digiteken kijelzendő értékeket. Ezen kívül az R8 regiszter felső 4 bitje tartalmazza, hogy üres-e az adott digit (DIG3-DIG0 sorrendben MSB-től kezdve). A bitek 1-es értéke jelzi, hogy az adott digit üres. Ugyanezzel a logikával az R8 regiszter alsó 4 bitje tartalmazza, hogy az adott digiten világít-e a tizedespont. Kimenete nincs, csak a hétszegmens kijelzőt állítja.

A szubrutin működése elég egyszerű, minden egyes digitnél a következőket végzi el:

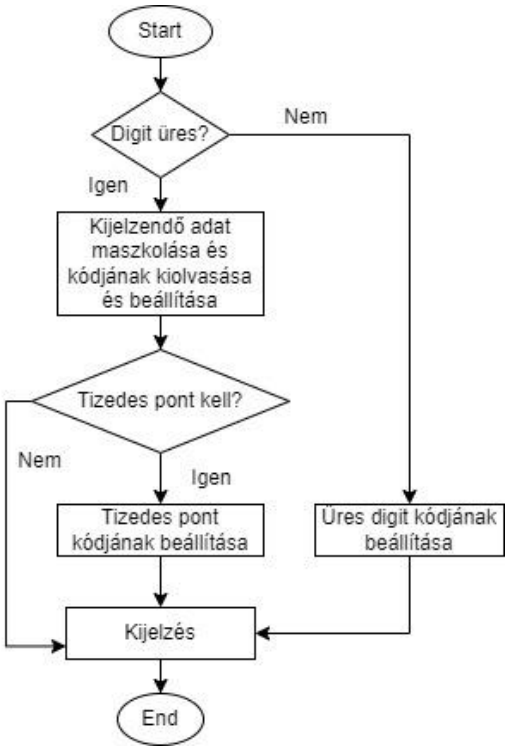
- megnézi üres-e
- ha nem üres, akkor az adott regiszterből a digitet R9-be helyezi, majd ott kimaszkolja a 4 bites számot
- a mérési útmutató példaprogramjának megfelelően az adatmemória elejére definiált szegmenstábla segítségével bekerül az R9-be a kijelzendő digit hétszegmens kódja

- megnézi, hogy be kell-e állítani a tizedespontot
- ha kell, akkor beállítja, egyénként tovább ugrik
- beállítja a megfelelő szegmenseket a kijelzőn
- tovább ugrik a következő digit állátását végző logikára
- ha a digit üres volt, a fentieket átugorva üresre állítja a megfelelő szegmenseket a kijelzőn, majd így megy tovább a következő digit logikájára

Egy digit logikájának kódja:

```
;kijelzi az r7-r6 számokat a 7szegmenses kijelzõn
basic_display:
    ;r7 dig3|dig2  r6 dig1|dig0 r8 blank|dp
    ;DIG0 kiírása
    TST r8, #0x10   ;blank tesztelése
    JNZ DIG0_blank  ;ugrunk, ha üres a digit
    MOV r9, r6      ;dig0 mozgatása
    AND r9, #0x0F   ;maszkolás, megkapjuk a dig0 számot
    ADD r9, #sgtbl  ;szegmens logika
    MOV r9, (r9)
    TST r8, #0x01   ;tizedespont tesztelése
    JZ load_DIG0    ;ugrunk, ha nem kell állítani
    OR r9, #0x80    ;tizedespont beállítása
load_DIG0:
    MOV DIG0, r9    ;szegmensek beállítása
    JMP DIG1_logic
DIG0_blank:
    MOV r9, #0x00   ;üres szegmens
    MOV DIG0, r9    ;szegmensek beállítása
DIG1_logic:
    ;DIG1 kiírása
    TST r8, #0x20   ;blank tesztelése
    JNZ DIG1_blank  ;ugrunk, ha üres a digit
```

Egy digit logikájának folyamatábrája:



CALC_4 program

Specifikáció

A CALC_4 program a CALC_3-hoz hasonló számológépet valósít meg. Vezérlésében megegyezik, de funkcionalitásában, felépítésében és kijelzésében kicsit különbözik.

Funkcionálisan abban különbözik, hogy most már a hexadecimális helyett a felhasználóbarát egy számjegyű BCD számokat képes csak operandusként kezelni, illetve az eredmény is BCD-ben jelenik meg két számjegyen (legnagyobb eredmény a 9*9=81 lehet, így ez biztosan elfér két számjegyen). Ennek megfelelően már nem csak az eredmény, hanem az operandusok is lehetnek hibásak, mégpedig akkor, ha a DIP kapcsolókon egy nem 0-9-ig terjedő számot állított be a felhasználó binárisan.

A funkcionális különbségekből következik a kijelzésbeli különbség. Ha hibás az *a* operandus, helyén az „E” karakter jelenik meg, az összes többi digit pedig inaktív marad. Ha az *a* operandus már nem hibás, de a *b* operandus az, akkor csak az eredmény digitjei lesznek inaktívak. Egyébként helyes operandusokon való műveletvégzésnél az eredmény is megjelenik, és az újabb helyes műveletvégzésig, vagy hibás operandusig ott marad. Ezen kívül a matematikai hibát az eredmény digiteken továbbra is az „EE” karakterkombináció fogja jelezni, azonban most fél másodpercenként ki-be kapcsolva villogva.

Ez utóbbiból következik a felépítésbeli különbség, ugyanis a fél másodperces időzítést a MiniRISC processzor beépített időzítője megszakításának kezelésével kellett megvalósítani.

Felépítés

A főprogram felépítése ezúttal is hasonló az eddigiekhez. Kiegészítés a megszakításkezeléshez szükséges inicializálás. Ezen kívül az operandusokat is ellenőrizni kell beolvasásuk után. Mint a nyomógombok ellenőrzését, ezt is sorban teszi a program (*a* és *b* sorrendben) a hibás operandusok esetén a megfelelő digitek tiltásával. Hibás operandus esetén a főprogram végtelen ciklusa le sem ellenőrzi a gombokat, a műveletvégzés amúgy sem lehetséges ilyenkor. Különbség még ezen kívül, hogy az összeadás és kivonás műveletek szubrutinjai eltűntek (tkp. nem is volt rájuk szükség soha), illetve a Z flag helyett a matematikai hibát az R4 regiszter decimális 1 értéke jelzi a megszakításkezelő *ISR* szubrutin felé. Tovább kisebb változás a *basic_display* szubrutinban, hogy nem is ellenőrzi le az eredmény digiteket a kiírás során, ha az R4 jelzi, hogy hibás az eredmény. Egyrész ez gyorsítja a kijelző szubrutint, másrészt nem akad össze a tárgyalandó *ISR* szubrutinnal.

További külön szubrutin végzi az eredmény BCD-be konvertálását (*bin_2_bcd*), azonban ez konkrétan csak az osztó szubrutint hívja meg, az osztó operandus helyére 10-et rakva.

Az *ISR* szubrutin

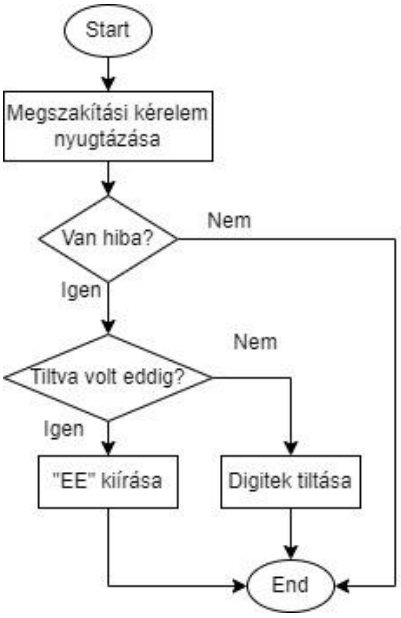
Maga a szubrutin elég egyszerű. A megszakítási kérelem nyugtázása után:

- megnézi, hogy van-e hiba (R4 regiszter 1-es értéke)
- ha van, megnézi, hogy mi van az eredmény digitek egyikén (mivel egyszerre kell villogniuk, és ugyanazt kell kijelezniük, így kezelhetők egyszerre)
- ha nem voltak tiltva, akkor tiltja őket
- ha tiltva voltak, kiírja az „EE” kombinációt
- ha nem volt hiba alapból, egyből visszatér a szubrutin

A szubrutin kódja:

```
;a két digit kezelhetõ egységesen, hisz mindkettõre E-t kell kiírni illetve tiltani kell
ISR:
    MOV r15, TS ;IT törlése
    TST r4, #0x01
    JZ IT_END
    MOV r15, DIG0   ;digit beolvasása
    TST r15, #0xFF  ;teszt, hogy nulla volt-e
    JZ DIG_zero
    MOV r15, #0x00  ;digitek tiltása, ha égett
    MOV DIG0, r15
    MOV DIG1, r15
    RTI
DIG_zero:
    MOV r15, #0x79  ;E kiírása, ha tiltva voltak
    MOV DIG0, r15
    MOV DIG1, r15
IT_END:
    RTI ;visszatérés az IT-bõl
```

A szubrutin folyamatábrája:



CALC_5 program

Specifikáció

A CALC_5 program a CALC_4-hez hasonló számológépet valósít meg. Funkcionalitásában megegyezik, de vezérlésében, felépítésében és kijelzésében kicsit különbözik.

Vezérlésében a különbség, hogy ezúttal a MiniRISC USRT vevőjén keresztül érkező adatokból találja ki, hogy mi a pontos elvégzendő művelet. A program egy decimális számot (*a* operandus), egy műveleti jelet („+”, „-”, „*”, „/”), még egy decimális számot (*b* operandus), végül egy ENTER vagy „=” karaktert vár bemeneti szekvenciaként. Ezen karakterek között bármilyen egyéb érkező karakterekre érzéketlen a program, de ha a bemeneti szekvenciában sorban következő karakterből egy is érkezik, akkor a szekvencia következő adatát fogja várni. A szekvencia végén kezdődik a művelet elvégzése. Ha a szekvencia közben vagy után bármikor ESC karaktert kap, a bemeneti szekvencia elölről kezdődik.

A vezérlésben való különbségekből származnak a kijelzésbeli különbségek. Az egyes digitek az eddigieknek megfelelnek a hétszegmens kijelzőn, azonban most csak akkor jelennek meg a számok, ha a bemeneti szekvenciában ott tart a program (pl. a *b* operandus értelemszerűen csak akkor jelenik meg, ha már jött előtte *a* operandus, illetve az eredmény csak a szekvencia végén látszódik). Ezen kívül matematikai hiba esetén nem kell villognia, elég statikusan megjelennie az „EE” kombinációnak az eredmény helyén.

Szintén a vezérlésben való különbségekből származnak a felépítésbeli különbségek. Most a megszakításkezelő szubrutin az USRT-t fogja kezelni, és a főprogram már nem a gombokat fogja lekérdezni, hanem azt, hogy hol tart a szekvencia, és ha véget ért, akkor milyen műveletet kell elvégeznie.

Felépítés

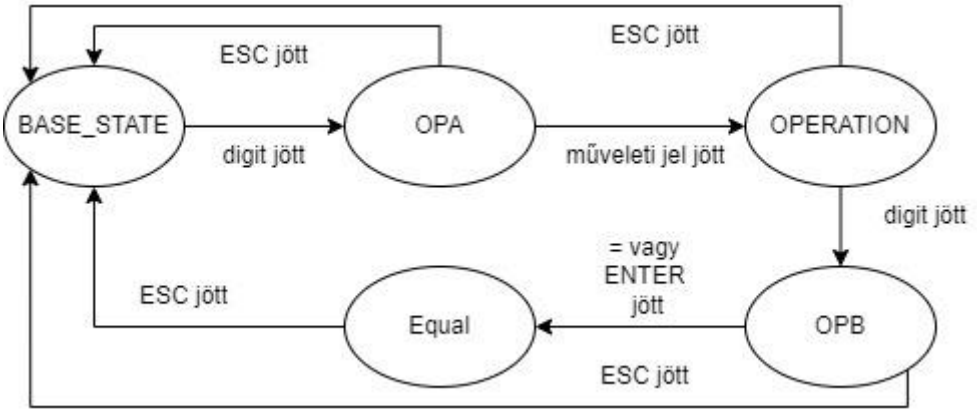
A fent említetteken túl felépítésbeli változás az, hogy ezúttal az USRT megszakításának kezelésére van az inicializáció. Ezen kívül a villogtatás hiányában nem kell alkalmazni a CALC_4-beli kiegészítést a *basic_display* szubrutinhoz. Bekerült még egy ellenőrző szubrutin is, amely eldönti, hogy az érkezett karakter vajon decimális számjegy-e (*check_operand_validity*).

A program lelke a főprogram, amely az *ISR* szubrutin által továbbított adatokat dolgozza fel. Az R5 regiszter 1-es értékén keresztül jelzi a rutin a főprogramnak, hogy új adat jött, és a 0-s értéken keresztül tudja ezt nyugtázni a főprogram. A beérkezett adatot az R15 regiszterben

adja át a rutin, illetve a bemeneti szekvencia állapotát az R12 regiszterben. A főprogram az állapotregisztert ciklikusan kérdezgetve (hasonlóan a nyomógombos megoldáshoz) az adatokat az adott állapotoknak megfelelő különböző regiszterekbe menti (külön regiszter az *a* és *b* operandusoknak, illetve a műveleti jel karakterének), majd nyugtázza az adat feldolgozását az R5 regiszter 0-ba állításával. Ha a szekvencia véget ért (ezt is az állapotból tudja), akkor a megfelelő regiszterébe elmentett műveleti jel karakterének lekérdezésével elvégzi a kijelölt műveletet (szintén hasonlóan az eddigi nyomógombos megoldáshoz).

Az *ISR* szubrutin

Az szubrutin tulajdonképpen az alábbi állapotgépet valósítja meg:



Az ezt megvalósító kód pedig:

```
ISR:
    MOV r15, UD     ;jött adat beolvasása
    MOV r5, #NEW_DATA   ;jelezzük, hogy jött adat
    MOV r14, r12    ;elmentjük az elõzõ state-t
    CMP r15, #ESC   ;ha ESC jött, akkor a STATE alapállapot, 0 és visszatérünk
    JNZ JUMP_STATE
    MOV r12, #BASE_STATE
    JMP RTI_ISR
    ;state reg: r12
JUMP_STATE:
    CMP r12, #OPA   ;elõzõleg OPA volt-e a STATE
    JZ OP_A
    CMP r12, #OPB   ;elõzõleg OPB volt-e a STATE
    JZ OP_B
    CMP r12, #OPERATION ;elõzõleg OPERATION volt-e a STATE
    JZ STATE_OPERATION
    CMP r12, #Equal     ;elõzõleg Equal volt-e a STATE
    JZ EQUAL
START:
    ;ide akkor jutunk, ha alapállapotban vagyunk
    MOV r6, r15     ;megnézzük, hogy a jött adat decimális szám-e
    JSR check_operand_validity
    JZ RTI_ISR
    MOV r12, #OPA   ;ha decimális szám, a STATE OPA lesz
    JMP RTI_ISR
```

```
SET_OPERATION:
    ;ide akkor jutunk, ha az elõzõ adat az A operandus volt és most mûvelet jött
    MOV r12, #OPERATION
    JMP RTI_ISR
STATE_OPERATION:
    ;ide akkor jutunk, ha az elõzõ adat mûvelet volt
    MOV r6, r15     ;ellenõrizzük, hogy decimális szám jött-e
    JSR check_operand_validity
    JZ CHECK_PREV_STATE ;ha nem szám jött, megnézzük változott-e az elõzõ állapothoz képest a STATE
    MOV r12, #OPB
    JMP RTI_ISR
OP_B:
    ;ide akkor jutunk, ha az elõzõ adat a B operandus volt
    CMP r15, #0x3d  ;a jött adat '='?
    JZ SET_EQUAL
    CMP r15, #0x0d  ;a jött adat '\r', azaz enter?
    JZ SET_EQUAL
    JMP RTI_ISR
SET_EQUAL:
    ;ide akkor jutunk, ha az elõzõ adat a B operandus volt és a jelenlegi adat '=' vagy enter
    MOV r12, #Equal
    JMP CHECK_PREV_STATE ;ha nem '=' vagy enter jött, megnézzük változott-e az elõzõ állapothoz képest a STATE
EQUAL:
;ide akkor jutunk, ha az elõzõ adat '=' vagy enter volt
    MOV r6, r15     ;ellenõrizzük, hogy decimális szám jött-e
    JSR check_operand_validity
    JZ CHECK_PREV_STATE  ;ha nem szám jött, megnézzük változott-e az elõzõ állapothoz képest a STATE
    MOV r12, #OPA
CHECK_PREV_STATE:
    ;ha nem változott az elõzõhöz képest a STATE, akkor nem tekintünk rá érvényes adatként
    CMP r12, r14
    JNZ RTI_ISR
    MOV r5, #DATA_PROCESSED
RTI_ISR:
    RTI
```