Széchenyi István Egyetem
Gépészmérnöki, Informatikai és
Villamosmérnöki Kar

SZAKDOLGOZAT

Hován Henrik

[Villamosmérnöki BSc szak]

[2022]

SZÉCHENYI
EGYETEM
UNIVERSITY OF GYŐR
GÉPÉSZMÉRNÖKI, INFORMATIKAI
ÉS VILLAMOSMÉRNÖKI KAR

SZAKDOLGOZAT

Számítógép architektúra tervezés, szimulálás és hardveres megvalósítás

Hován Henrik

Villamosmérnöki BSc szak

Automatizálási szakirány

2022

Feladat-kiíró lap szakdolgozathoz

Hallgató adatai

Név: Hován Henrik

Neptun-kód: FLMCIN

Szak: Villamosmérnök BSc

Specializáció: Automatizálási szakirány

Tagozat: nappali

A szakdolgozat adatai

Kezdő tanév és félév: 2022/23/1

Nyelv: magyar

Típus: nyilvános

Számítógép architektúra tervezés

Feladatok részletezése:

1) Történelmi kitekintő, irodalomkutatás
2) Architektúra elméleti megtervezése
3) Rendszer gyakorlati megtervezése, legyártása
4) Teszt program elkészítése
5) Az elkészült rendszer tesztelése és értékelése

Belső konzulens adatai

Név: Somogyi Miklós

Tanszék: Automatizálási Tanszék

Beosztás: Tanszéki mérnök

Külső konzulens adatai

Név: Dr. Keresztes Péter

Munkahely: Széchenyi István Egyetem

Beosztás: Nyugalmazott egyetemi docens

Győr, 2022. 05. 20.

belső konzulens

külső konzulens

Dr. Ballagi Áron tanszékvezető,
Automatizálási Tanszék

Nyilatkozat

Alulírott, Hován Henrik (FLMCIN), Villamosmérnök BSc szakos hallgató kijelentem, hogy a Számítógép architektúra tervezés, szimulálás és hardveres megvalósítás című szakdolgozat feladat kidolgozása a saját munkám, abban csak a megjelölt forrásokat, és a megjelölt mértékben használtam fel, az idézés szabályainak megfelelően, a hivatkozások pontos megjelölésével.

Eredményeim saját munkán, számításokon, kutatáson, valós méréseken alapulnak, és a legjobb tudásom szerint hitelesek.

Győr, 2022. 12. 15.

Hován Henrik

Kivonat

Számítógép architektúra tervezés, szimulálás és hardveres megvalósítás

A számítástechnika története hosszú időre nyúlik vissza, egészen az 1600-as évekig. A számítástechnika nagyapjának is nevezett Pascal első mechanikus számítógépétől elindulva napjainkig a számítógépek elméleti felépítésükben kevés változáson mentek keresztül. Az elektromosság korában először kisebb teljesítménnyel, jelfogókból, relékből, később vákuumcsövekből készültek az ilyen rendszerek. A modernebb eszközök megjelenésével, a tranzisztor majd az integrált áramkör megalkotása után az igények és az elvárások a számítógépekkel szemben folyamatosan nőttek. Ezáltal a számítógépek fizikai felépítése folyamatosan változott, a korszerű számítógépekben már pár évente lecserélődő architektúra felépítése gyökereiben mégis azonos az első rendszerekével. Dolgozatomban számba veszem a történelem során megalkotott, a számítástechnika szempontjából fontos fordulópontot képviselő találmányokat, azok felépítését, és a tudósokat, akik ezek mögött a találmányok mögött álltak. Ezután az általuk felállított alapelvek alapján és is elkészítem a saját számítógép rendszeremet, amit az ezek által a tudósok által felállított szempontok szerint értékelek és fejlesztek. Végezetül az elméletben felépített rendszert először egy szimulációs környezetben valósítom meg, amit szintén nekem kell megalkotnom, majd, ha a szimulációs során az elméletek bizonyítanak és a rendszer működőképes, akkor azokat gyakorlatban is megvalósítom moduláris rendszerben. Ezzel párhuzamosan elkészítek pár minta programot, amivel tudom tesztelni a rendszer működését, és az ezeknek a programoknak a könnyű fejlesztését elősegítő fordítóprogramot is, amivel egy ember által könnyebben olvasható nyelven lehet dolgozni. A befejezésemben szót ejtek a jövőben lehetőségekről, a projekt további fejlesztési irányairól.

Abstract

Designing and simulation of a computers architecture

The history of computer technology goes back a long time, all the way to the 1600s. Starting from Pascal's first mechanical computer, also known as the grandfather of computer technology, computers have undergone few changes in their theoretical structure until today. In the age of electricity, such systems were first made with lower power, from signal detectors, relays, and later from vacuum tubes. With the appearance of more modern devices, after the creation of the transistor and the integrated circuit, the needs and expectations of computers have continuously increased. As a result, the physical structure of computers is constantly changing, but the structure of the architecture, which is replaced every few years in modern computers, is fundamentally the same as the first systems. In my thesis, I consider the inventions created throughout history that represent an important turning point in terms of computer technology, their structure, and the scientists who were behind these inventions. Then, based on the principles set by them, I create my own computer system, which I evaluate and develop according to the criteria set by these scientists. Finally, I first implement the system built in theory in a simulation environment, which I also have to create, then if the theories prove themselves during the simulation and the system is functional, then I implement them in practice in a modular system. In parallel, I will create a couple of sample programs with which I can test the operation of the system, as well as the translation program that facilitates the easy development of these programs, which can be used to work in a language that is easier for a human to read. In my conclusion, I will write about future possibilities and the further development directions of the project.

Tartalomjegyzék

1. Bevezetés 1

2. Történelmi áttekintés 2

2.1 Korai számítási eszközök, a mechanikus számítógépek 2

2.1.1 Pascal és Leibnitz 2

2.1.2 Charles Babbage 2

2.1.4 Harvard architektúra 3

2.2 Az elektronikus számítógépek 4

2.2.1 A háború nyomása 4

2.2.2 Konrad Ernst Oto Zuse 4

2.2.3 ENIAC 4

2.2.4 A Neumann számítógép 5

2.3 A tranzisztorok kora 6

2.4 Az integrált áramkörök megjelenése 7

2.5 Modern számítógépek felépítése 8

3. Elvárások 8

4. Tápegység és órajel forrás 9

4.1 Tápegység 9

4.2 Órajel 13

4.2.1 Órajel generátor 13

4.2.2 128 Hz 14

4.2.3 32768Hz 14

4.2.4 100000Hz 15

4.3 Órajelek szimulációja 15

4.3.1 128Hz 15

4.3.2 32768Hz 16

4.3.3 10000kHz 16

4.4 A valós áramkör mérései 17

4.4.1 128Hz 17

4.4.2 32768Hz 19

4.4.3.100000Hz 20

5. Regiszterek és ALU 21

5.1 Aritmetikai és logikai egység elméleti felépítése 21

5.1.1 Regiszterek 22

5.1.2 Összeadás 22

5.1.3 Logikai „és”, logikai „vagy” 24
5.1.4 Logikai „nem” 25
5.1.5 Hasonlítás 25
5.2 Aritmetikai és logikai egység gyakorlati megvalósítása 26
5.2.1 Hordozó panel és regiszterek 26
5.2.2 Összeadó 26
5.2.3 Logikai „és”, logikai „vagy”, logikai „nem” 27
6. Memóriák 27
6.1 Memória modul elméleti felépítése 28
6.1.1 RAM 29
6.1.2 Program memória 29
6.2 Memória modul gyakorlati felépítése 30
7. Vezérlő egység 32
7.1 Elméleti felépítés 32
7.1.1 Módosító jelek 33
7.2 Program számláló 34
8. GPIO és DIO 36
8.1 GPIO és DIO elméleti felépítése 36
9. Logikai eltolás 37
10. Komparátor 38
11. Szimulátor 39
11.1 Órajel forrás kalkulátor 39
11.2 Szimulátor 40
11.2.1 Program számláló 40
11.2.2 Memóriák 42
11.2.3 Aritmetikai és logikai egység 43
11.2.4 Vezérlő egység 44
11.2.5 Be és kimenetek 46
11.3 Utasításkészlet generátor 47
12. Utasításkészlet 47
12.1 CISC 48
12.2 RISC 48
12.3 Saját rendszer 48
2. táblázat A rendszer utasításkészlete 49
12.3.1 Logikai műveletek változókkal 50
12.3.2 Aritmetikai műveletek 51

12.3.3 Ugrások 52
12.3.4 Memória műveletek 53
12.3.5 GPIO műveletek 54
12.3.6 Valós idejű óra 55
12.4 Utasítás végrehajtás 55
12.5 Utasításkészlet szerkesztése 57
12.6 Utasításkészlet feltöltése 58
12.7 Egyszerű fordító 58
13. Próba programok 60
13.1 Alap funkcióteszt 60
13.3 GPIO vezérlés és LCD 61
13.3.1 HD44780 vezérlésű kijelzők 61
13.3.2 Számolás teszt 62
13.4 Bemenetek olvasása 62
14. Összegzés 63
Irodalomjegyzék 64
Mellékletek I
1. Melléklet I
2. Melléklet IV

1. Bevezetés

Az elmúlt évszázad egyik legnagyobb vívmánya és egyértelmű mozgatórugója a számítástechnika. Napjainkban már minden eszközünkben megtalálhatóak az egyszerűbb vagy bonyolultabb számítógépek, amik egyszerűbbé teszik a mindennapokat. A legtöbb esetben bele se gondolunk, hogy amikor megnyomunk egy billentyűt vagy megérintünk egy területet az okostelefon kijelzőjén, mi is történik a háttérben. Ennek a kérdésnek a megválaszolását tűztem ki célomul, amihez a legjobb eszköznek azt láttam, ha tervezek egy teljesen saját számítógépet, egészen az alapjaitól.

Egy számítógép alapjának azt az elvet, rendszert nevezhetjük, ami alapján az működik. Ezt a rendszert átfogó névvel csak architektúrának szokták nevezni, és habár ez nem egy hivatalos elnevezés, az IBM mérnökei általi első említés óta ez lett elterjedt.

A történelem során több számítógép architektúra is készült, amik mind-mind a Turing-elvek alapján kívánták megvalósítani a számítógép funkcióit. Dolgozatomban kísérletet teszek egy alapszintű funkcionalitással rendelkező számítógép elméleti és gyakorlati megvalósítására. Ehhez segítségül hívom az Andrew S. Tanenbaum által írt Számítógép-architektúrák könyvet, valamint az interneten elérhető szöveges és videós forrásokat.

A rendszer eredeti elképzelése a BenEater felhasználónevű YouTube felhasználó 8 bites rendszerén alapul, annak elméleti felépítéséből kiindulva. Az ő általa elkészített eszköz több pontos is eltér attól, amire én kísérletet teszek:

- az utasítás 8 bites, első 4 bitje az utasító szó, felső 4 bitje pedig egy esetleges cím,
- a rendszer nem tud logikai eltolást végezni.

A számítógép-architektúrával legrészletesebben az Andrew S. Tanenbaum által írt Számítógép-architektúrák könyv foglalkozik. Ebben hat mérföldkőre osztja szét a számítástechnikai eszközök fejlődését:

- nulladik generáció: a mechanikus számítógépek 1642-1945,
- első generáció: vákuumcsöves számítógépek 1945-1955,
- második generáció: tranzisztoros számítógépek 1955-1965,
- harmadik generáció: integrált áramkör számítógépek 1965-1980,
- negyedik generáció: magas integráltságú áramkör számítógépek 1980-napjainkig,
- ötödik generáció: láthatatlan számítógépek. [1]

2. Történelmi áttekintés

2.1 Korai számítási eszközök, a mechanikus számítógépek

2.1.1 Pascal és Leibnitz

1642-ben alkotta meg az első mechanikus számítógépet Blaise Pascal, aki ezzel az apjának, a franci kormány adószedőjének akart segíteni. [1] Ez a fogaskerekekből álló szerkezet egy karral volt üzemeltethető és az összeadás és kivonás műveletére volt csak képes. Tiszteletére róla nevezték el a Pascal programozási nyelvet. [1] Alig 30 évvel később Gottfried Wilhelm von Leibniz báró, német matematikus megépítette a szintén mechanikus, de már szorozni és osztani is képes gépét. [1][3] Ezután mintegy százötven évig nem történt érdemi változás a számítástechnika történetében.

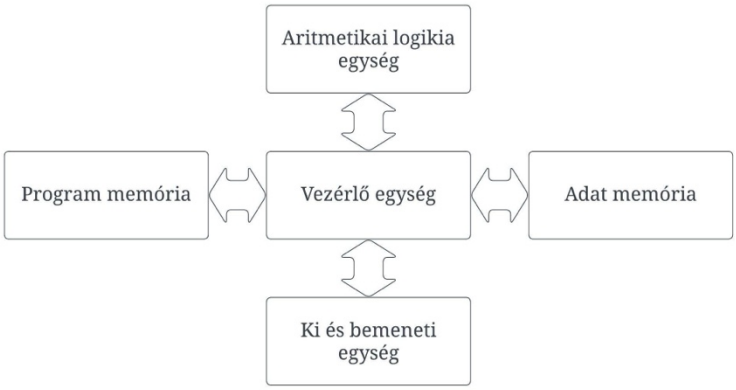
2.1.2 Charles Babbage

A cambridge-i egyetem matematika professzora 1823-ban megalkotta a saját, teljesen mechanikus differenciagépét, ami még, Pascaléhoz hasonlóan csak a két alapműveletre volt képes. Ezt az eszközt hajózási navigációs táblák összeállításához használta, azonban nagyon hamar megunta egy célra alkalmas találmányát, így egy sokkal bonyolultabb és szélesebb körűen használható eszközön kezdett el dolgozni. Ebbe a rendszerbe a teljes családi vagyonát és egy 17000 fontos kormány támogatást ölt bele, hogy megalkothassa az úgynevezett analitikusgépet. Ez az eszköz négy egységből állt:

- tároló (memória),
- malom (számolóegység),
- bemeneti rész (lyukkártya olvasó),
- kimeneti rész.

A rendszer nagy előnye volt, hogy általános célú eszközként, lyukkártyákról olvasott be utasításokat és ezeket hajtotta végre. Sosem láthatta viszont rendszerét teljes tökéletességgel működni, mert a kor technológiája nem tette lehetővé annyira precíz alkatrészek elkészítését, ami ehhez a feladathoz szükséges lett volna.

Az analitikus gép egy egyszerű assembly nyelven volt programozható, amire Babbage Ada Augusta Lovelace-t, Lord Byron híres brit költő lányát bízta meg, és így lett ő a világ első programozója. Tiszteletére elnevezték róla az Ada® programozási nyelvet. [1]



1. ábra *A Harvard-architektúra felépítése*

2.1.4 Harvard architektúra

Howard Aiken a Harvard egyetem Phd kutatásai során hatalmas mennyiségű számítást volt kénytelen elvégezni kézzel, és miután fokozatát megszerezte nem volt száméra kétséges, hogy ezeket a folyamatokat valahogy automatizálnia kell. Elkezdte kutatni az irodalmat és rátalált Babbage analitikus gépére, amit újra alkotott jelfogók segítségével. [1]

Ez, a mechanikus gép elvén készült rendszernek az elméleti felépítése látható a *1. ábrán*, ami egy olyan számítógép felépítési elv, amiben a program és adatsín elkülönül egymástól. Ezt a Harvard Mark I. számítógéphez dolgozták ki 1944-ben és a mai napig a DSP és mikrovezérlők architektúrájának az alapját képezi. Előnye a kettős busz miatt a programkód futás közbeni módosítása, elérése, adatként való felhasználása. Szintén ezen tulajdonsága miatt nehezebb rá kártékony szoftvereket írni. [1]

A Harvard architektúra tulajdonságai:

- a memória szóhosszúsága, időzítése, tervezési technológiája, címzése is különböző lehet,
- a program memória gyakran nagyobb, mint az adat memória,
- az utasítások csak olvasható, míg az adatok írható és olvasható memóriában kapnak helyet,
- a Neumann architektúrával ellentétben képes egyszerre adatot írni vagy olvasni és utasítást beolvasni. [5]

A Harvard architektúra is rendelkezik néhány hátránnyal:

A modernebb SoC (System on Chip – Rendszer a Chipben) alapú rendszereknél a különböző memóriatípusok összehangolása nehezen megoldható feladat, itt általában Neumann elvű a külső memória rendszer. [5]

2.2 Az elektronikus számítógépek

2.2.1 A háború nyomása

Az 1940-es években a háború korai szakaszában a német tengeralattjárók rádiós kommunikáción keresztül kapták az utasításokat, azonban egy egyszerű kódolási rendszer a szövetségesek számra könnyen visszafejthető lett volna, és ezzel a németek tengeralattjárók okozta taktikai előnyeiket elvesztették volna. Ezt a kódolást az ENIGMA nevű rendszerre bízták, aminek gyors és biztos feltörése egy fő feladattá vált a II. világháború idején. A brit kormány annyira fontosan kezelte ezt a feladatot, hogy egy szupertitkos laboratóriumot állítottak fel, ahol elkészítették 1943-ban a COLOSSUS névre keresztelt elektronikus számítógépet, aminek munkálataiban Turing is részt vett. Mivel a brit kormány a rendszer dokumentumait 30 évre titkossá nyilvánította, így nagy hatása nem volt a számítástechnikára, mégis ez az első elektronikus számítógép. [1]

2.2.2 Konrad Ernst Oto Zuse

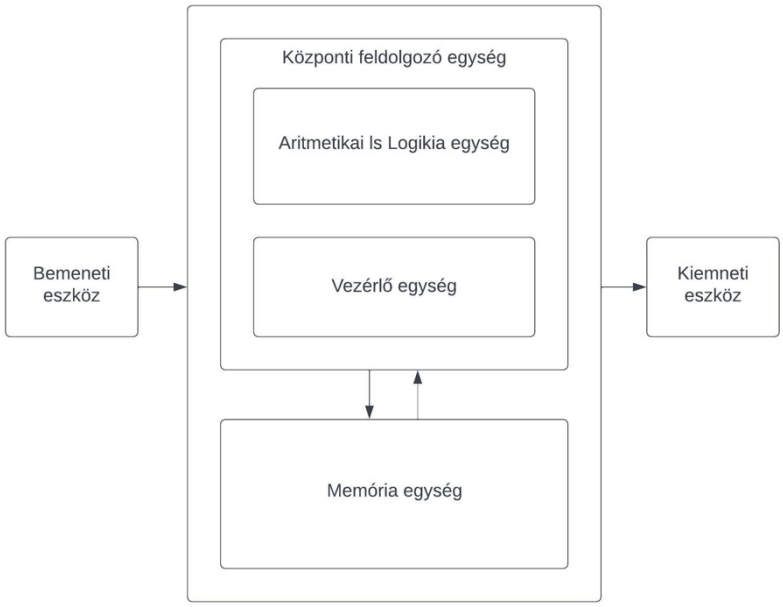
Konrad Zuse a XX. század egyik számítástechnikai úttörője, német mérnök. Első számítógépét még főiskolai tanulmányai alatt kezdte el építeni. Élete során több számítógépet is készített, első ezek közül a Z1 volt. Ez még egy mechanikus eszköz volt, később a Z2 már reléket használt. A Z3 volt az első program vezérelt, kettes számrendszerben dolgozó számítógép. Munkáinak a korszakban nem tanúsítottak nagy jelentőséget, az akkori náci Németországban, konstrukció ettől függetlenül úttörők voltak, a Z3-as számítógépe 3 évvel előzte meg a Mark I-et. Első általa készített számítógépek elpusztultak a háborúban, így nem gyakoroltak nagy hatást a számítástechnika fejlődésére.

Élete utolsó éveiben a számítástechnika elméletével foglalkozott, megalkotta az első magas szintű programozási nyelvet, a Plankalkült 1948-ban. Vele szinte teljesen egy időben a világ több pontján születtek meg az első elektronikus számítógépek. Az egyik legnagyobb hatása a magyar feltalálónak, Neumann Jánosnak volt, akinek alapelvei alapján készülnek napjainkban is a számítógépek. Vele párhuzamosan készült a Princeton egyetemen a Mark I. nevű, eltérő felépítésű, de hasonlóan nagy hatást gyakorló számítógép. [6]

2.2.3 ENIAC

John Mauckley tisztában volt vele, hogy hadsereget érdeklik az automatizált számológépek, és így ötletével támogatásért fordult hozzájuk. Sikerrel is járt, és elkezdhette az ENIAC (Electronic Numerical Integrátor And Computer) megtervezését és megépítését doktorandusz hallgatójával, J. Presper Eckert-el.

A rendszer 18000 vákuumcsőből és 1500 jelfogóból állt, 30 tonnát nyomott áramfogyasztása pedig 140 kilowatt volt. Programozása 6000 kapcsolóval és dugaszoló aljzatok sokaságával volt lehetséges. A rendszer nem készült el 1946-ig, így az eredeti céljára már nem volt alkalmas. Mauckley és Eckert engedélyt kaptak, hogy egy nyári egyetem során tudóstársaiknak bemutassák a gépet. Ez az esemény keltette fel a digitális számítógépek iránti robbanásszerű érdeklődést. A két tudós nem sokkal később elkezdett újabb gépükön, az EDVAC-on dolgozni, ez azonban dugába dőlt mikor elhagyták a Pennsylvaniai Egyetemet és megalakították saját cégüket. [1]



2. ábra *A Neumann-architektúra felépítése*

2.2.4 A Neumann számítógép

Az *2. ábrán* látható elméleti felépítés, melyet társaival Neumann János publikált 1945. június 30-án, amiben a számítógépet öt fontosabb alkotóelemre bontotta szét:

- aritmetikai-logikai egység és annak regiszterei
- vezérlő egység, amiben helyezkedik el a program számláló és az utasításkészlet
- operatív tároló, az adatok és utasítások tárolására
- háttértár
- perifériákhoz tartozó ki- és beviteli mechanizmusok. [2][5]

A Neumann architektúra úgynevezett „De Facto” szabvány, azaz az adat és utasítás-címek a memória ugyanazon címtartományán vannak leképezve. Ilyen típusú rendszerek az:

- EDVAC (Neumann) egyenletmegoldó tárolt-programú gép,

- Eckert, Mauckly: ENIAC, UNIVAC (University of Pennsylvania) numerikus integrátor,
- a mai modern mini-, micro és mainframe számítógépek

A Neumanni számítógép tervezés alapelvei:

- a számítógépet egy a memóriájában tárolt program vezérli (Turing),
- a vezérlést vezérlés-folyam segítségével lehet leírni, amiben fontos az adatút megtervezése,
- a gép belső memóriájában a programkód és a végrehajtásukhoz szükséges adatok együttesen megtalálhatóak. A program felülírhatja önmagát, ami a Neumann architektúra definíciója is egyben
- az aritmetikai és logikai műveletek végrehajtását külön egység végzi
- az adatok és programok beolvasására és az eredmények kiírására be- és kimeneti perifériák szolgálnak
- 2-es számrendszer alkalmazása, bináris számítógép [2]

A korai számítástechnikai eszközök fix programokkal rendelkeztek, a teljes struktúra átalakítása volt szükséges egy új programhoz. Ehhez képest a tárolt programú számítógépek utasításkészlet alapú architektúrával és változtatható programokkal rendelkeztek, ami magas szintű változtathatóságot és alkalmazhatóságot tett lehetővé.

A Neumann architektúrának az előnyei mellett van néhány hátránya is:

- az önmagát változtatni képes program miatt lehetőség van kártékony programok készítésére, és egy hibaforrás is lehet, ha nem szándékosan, csak egy hiba miatt módosítja magát a program,
- Neumann „bottleneck”: a sávszélesség korlátja a CPU és a memória között, ami legfőképpen nagy adatmennyiségek mozgatása során fordulhat elő,
- a cache memóriát nem alkalmazó rendszerekben nem lehet egyszerre adat írás, olvasás és utasítás beolvasás műveletet végrehajtani az egybusz rendszer miatt.

Neumann IAS gépével egy időben készült az M.I.T. gépe is, a Whirlwind I, amit folyamatirányításra terveztek. Ennek a gépnek a munkálatai vezettek Jay Forrester felfedezéséhez, a mágnesgyűrűs memóriához.

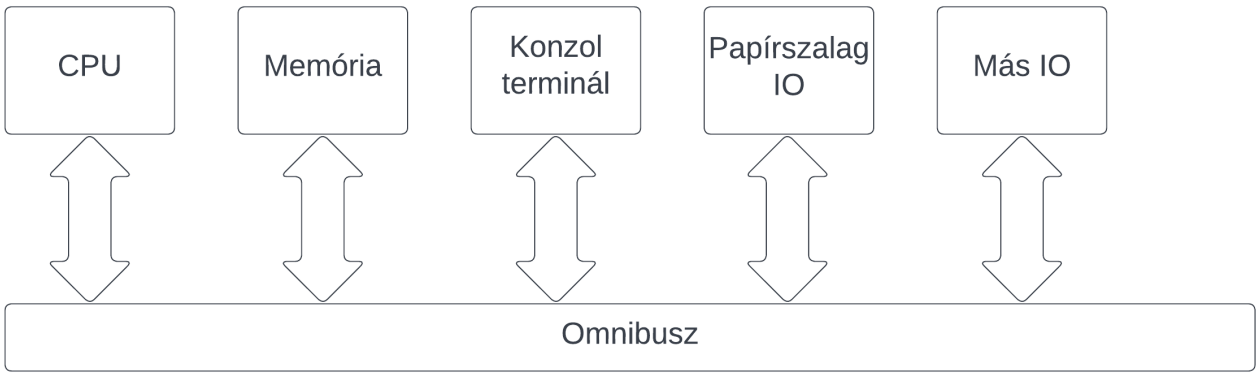
2.3 A tranzisztorok kora

A Bell Labs három munkatársa 1948-ban feltalálta azt az eszközt, ami azóta is meghatározó alkotóeleme az elektronikának. Ez a három tudós: John Bardeen, Walter Brattain és William Shockley, a találmányuk pedig a tranzisztor.

Tíz éven belül a tranzisztorok forradalmasították a számítógép ipart, a vákuumcsövek pedig elavulttá váltak. Az első tranzisztoros számítógép az M.I.T.-n épült meg és a Whirlwind I. mintájára 16 bites volt. A TX-0 (Transistored eXpreimental computer 0) eredetileg a sokkal bonyolultabb TX-

2 tesztelésére készült, de utóbbit nem becsülték sokra a képességei ellenére. Ekkor jelentősen begyorsult a számítógépipar, az IBM a 7090-el megalkotta a kor leggyorsabb számítógépét.

A TX-0 munkálatain dolgozó Kenneth Olsen 1957-ben megalapította saját cégét Digital Equipment Corporation (DEC) néven. Első számítógépük a PDP-1-es 4096 darab 18 bites szót tartalmazott és 200000 utasítás végrehajtására volt képes másodpercenként.



3. ábra *A PDP-8 omnibusz rendszere*

A DEC néhány évvel később megalkotta a PDP-8-at, aminek különlegessége az *3. ábrán* látható omnibuszos rendszer volt. Ez jelentősen eltért az addigi megoldásoktól, de azóta szinte minden kisszámítógépben ezt a megoldást használják. Számítógépemben én is ezt a megoldást követem. [1] Az IBM a 7090-es után bemutatta a 7094-et, ami 2 mikroszekundum ciklusidővel rendelkezett, 32536 szavas, szavanként 36 bites mágnesgyűrűs memóriával.

1964-ben egy CDC nevű kis cég bemutatta a 6600-as számítógépét, ami kétszer gyorsabb volt minden addigi számítógépnél. Ennek az eszköznek a különlegessége a párhuzamosság volt, gondos programozással egyszerre akár 10 utasítást is végre tudott hajtani, amit a különböző kisszámítógépek tettek lehetővé, amik minden számításon kívüli feladatot átvettek a CPU-tól.

2.4 Az integrált áramkörök megjelenése

Robert Noyce 1958-as felfedezése, a szilícium integrált áramkör lehetővé tette, hogy egy kis lapkára több tranzisztort is integráljanak, ennek hatására a korábbi egyszerű tranzisztoros számítógépeknél sokkal bonyolultabb, de kisebb és gyorsabb számítógépeket tudtak alkotni. Az IBM a 7094-el és a 1401-el hatalmas sikereket ért el, azonban ez a két rendszer nem volt semmilyen mértékben kompatibilis egymással, sok gyárnak pedig mind a két rendszerre szüksége volt, ami miatt két külön programozó részleget kellett fenntartaniuk.

Az IBM egy merész lépésre szánta el magát és egy mindenre jó rendszert mutatott be, a System/360-at. A kisebbik, 1401-es kiváltására a 360 Model 30-at, míg a nagyobbik 7094 kiváltására

a 360 Model 75-öt. Ezeket a számítógépeket ugyanazon az assembly nyelven lehetett programozni, a kisebb változaton írt programok gond nélkül futottak a nagyobb változaton, az utóbbira írtak viszont nem biztos, hogy elfértek a másik memóriájában. Ezzel a termékcsaládok gondolata gyorsan elterjedt az iparban. Ugyancsak a 360 volt az első számítógép, ami más számítógépeket tudott emulálni. [1]

2.5 Modern számítógépek felépítése

A modern rendszerekben a Neumann és Harvard architektúra előnyeit igyekeznek egyesíteni, de többségükben utóbbit használják. A módosított Harvard-architektúra esetén napjainkban a CPU felváltva képes mindkét rendszerként viselkedni. Amíg a gyorsítótárban található adatokból dolgozik, addig Harvard, ha a program memóriához közvetlenül fordul az eszköz, akkor Neumann-architektúraként viselkedik. Ilyen elven működnek a mai ARM és Intel X86 processzorok. [1]

Más változatok lehetővé teszik a programmemóriában található szavak kizárólagos olvasását. Ezt a technikát használják a mikrovezérlők, például a Microchip (korábban Atmel) AVR kontrollerek.

3. Elvárások

A saját számítógép rendszerem megtervezésének első, és legfontosabb lépése, annak az eldöntése, hogy milyen funkciókkal kell rendelkeznie. Van néhány olyan egyértelmű tulajdonság, amivel rendelkeznie kell, ahhoz, hogy számítógépnek lehessen nevezni:

- képesnek kell lennie utasításokat végrehajtani, amiket egy tárolóból olvas ki,
- képesnek kell lennie műveletek végrehajtására különböző számokkal,
- képesnek kell lennie az eredményeket ideiglenesen, vagy tartósan tárolni,
- képesnek kell lennie a felhasználó számára értelmezhető módon információt közölni.

A rendszer kialakítása, a modulok megtervezése során, ezeknek a céloknak a megvalósítására próbálok törekedni.

A rendszert 5 modulra osztottam szét:

- tápegység és órajel forrás,
- aritmetikai logikai egység,
- memória egység,
- vezérlő egység,
- felhasználói interfész.

4. Tápegység és órajel forrás

4.1 Tápegység

Ahhoz, hogy a rendszer működőképes legyen, szükség van egy olyan stabil áramforrásra, ami képes kielégíteni az összes igényt, mégis alacsony veszteséggel dolgozik. Erre a feladatra a legmegfelelőbb egy kapcsolóüzemű, úgynevezett *buck* konverter, ami nagy hatékonysággal képes nagyobb feszültségekből kisebbeket előállítani. Habár a buck konverter kialakítható egyedi felépítésű kompenzációs áramkörrel és erősítőkkel, egyszerűbb és megbízhatóbb megoldást tud nyújtani egy erre a célra tervezett integrált áramkör.

A feladatra a Texas Instruments LM25010 típusú buck vezérlő IC-t választottam ki.

Néhány fontos adat a kiválasztás szempontjából

- kevés passzív alkatrészt igényel
- széles tartományban képes a bemeneti feszültség kezelésére, 6-42V
- 1A kimeneti áramot tud stabilan biztosítani
- akár 1MHz kapcsoló frekvenciát is elérhet, ezzel csökkentve a releváns zajt a rendszerben
- belső kapcsoló FET-et tartalmaz

A gyártó az áramkör tervezéséhez több eszközt is biztosít, köztük Excel táblás kalkulátort és online kalkulátort, de az adatlap alapján, kézzel is kiszámíthatóak a szükséges alkatrész értékek.

A tápegység méretezése az alábbi számítás mentén történt: [7]

Kimeneti feszültség

$$\frac{R1}{R2} = \frac{Vout}{2{,}5} - 1 \tag{1}$$

$$Vout = 2{,}5 * \left(\frac{R1}{R2} + 1\right) \tag{2}$$

$$5 = 2{,}5 * \left(\frac{R1}{R2} + 1\right) = 2{,}5 * \left(\frac{1}{1} + 1\right) \tag{3}$$

A számításból kiderül, hogy a feedback áramkör ellenállásait úgy kell megválasztanunk, hogy

azok azonosak legyenek. Az adatlap 1k és 10k közötti értéket javasol, így 1k kerül az áramkörbe.

A számítás következő lépéseként az R_{ON} ellenállás kiszámítását adja meg az adatlap. Ezzel az ellenállással lehet beállítani a kapcsoló frekvenciát.

$$Ron = \frac{V_{OUT}*(V_{nominal}-1{,}4V)}{V_{nominal}* F_{SW}*1{,}18*10^{-10}} - 1.4k \tag{4}$$

$$R_{ON} = \frac{5V*(12V-1{,}4V)}{12V*500kHz*1{,}18*10^{-10}} - 1{,}4k = 74857 \tag{5}$$

Ez alapján a kiválasztott ellenállás a 75kO méretű, amivel 6V feszültségen 433kHz, 42V feszültségen 546kHz a kapcsoló frekvencia.

Következő lépésként meg kell határozni a tekercs értékét, amihez az alábbi képlet ad segítséget:

$$L = \frac{V_{OUT}*\left(V_{IN(MAX)}-V_{OUT}\right)}{I_{OR}*F_{S(MIN)}*V_{IN(MAX)}} \tag{6}$$

Ahol I_{OR} a legkisebb terhelő áramot, $F_{S(MIN)}$ a legkisebb kapcsoló frekvenciát jelenti maximális bemeneti feszültség mellett.

Feltételezve, hogy a legkisebb terhelés 100mA, és a legkisebb frekvencia az adatlap szerint a névleges 75%-a, azaz 409,5kHz, a képlet így áll össze:

$$L = \frac{5V*(42V-5V)}{0{,}1A*409{,}5kHz*42V} = 107{,}5uH \tag{7}$$

Ez a képlet az L tekercs legkisebb ajánlott értékét adja meg, de az adatlap ajánlja, hogy ennél nagyobb kerüljön betervezésre.

Az LM25010 típusú IC megfelelő működéséhez a kimenetén legalább $25mV_{P-P}$ ingadozásnak kell lennie, annak pontos értéket az alábbi képlet határozza meg:

$$V_{RIPPLE} = 25mV_{PP} * \frac{R1+R2}{R2} \tag{8}$$

$$V_{RIPPLE} = 25mV_{PP} * \frac{2k}{1k} = 50mV_{PP} \quad (9)$$

A szükséges feszültség ingadozás a tekercs áramingadozásából jön létre a C2 ESR ellenállására és az R3-ra hatva. Elsőként meg kell határozni a minimális áram ingadozást, ami a bemeneti feszültség legkisebb értéke esetén lép fel:

$$I_{OR(MIN)} = \frac{V_{OUT}*(V_{IN(MIN)}-V_{OUT})}{L_{MAX}*F_{S(MAX)}*V_{IN(MIN)}} \quad (10)$$

Mivel a számítások alapján az L értéke minimum 107,5 uH kell legyen, így itt az $L1_{MAX}$ értékét 120uH-ben határozom meg, az $F_{S(MAX)}$ értéke 546kHz, a $V_{IN(MIN)}$ pedig 6V.

$$I_{OR(MAX)} = \frac{5V*(6V-5V)}{107{,}5uH*546kHz*\ 6V} = 14{,}19mA_{PP} \quad (11)$$

Ez alapján már számolható az ESR minimális értéke a C2 kimeneti kondenzátorhoz:

$$ESR_{MIN} = \frac{V_{RIPPLE}}{I_{OR(MIN)}} = \frac{50mV}{14{,}19mA} = 3{,}52 \quad (12)$$

Ha a kiválasztott kondenzátor nem rendelkezik a megfelelő ESR értékkel, egy vele sorba kötött R3 ellenállás használata szükséges.

Belső és külső EMI jelek befolyásolhatják az IC működését, ezért egy Schottky dióda használata ajánlott az adatlap szerint, aminek a méretezése a következő képlet alapján történik:

$$P_{D1} = V_F *\ I_O * (1 - D) = 2{,}5V * 1{,}5A * \left(1 - \frac{5V}{12V}\right) = 2{,}1875W \quad (13)$$

A képlet alapján egy olyan Schottky diódát kell kiválasztani, ami alkalmas a 2,1875W elnyelésére.

A tápegység bemenetén egy bemeneti kondenzátor limitálja a feszültség ingadozását, aminek az érték meghatározásához először ki kell számolni a kapcsolójel aktív idejét:

$$t_{ON(MAX)} = \left(\frac{1{,}18*10^{-10}*(75k+\ 1{,}4k)}{6V-1{,}4V} + 67ns\right) * 1{,}25 = 2{,}53us \quad (14)$$

A bemenő feszültség nem eshet 5,5V alá, annak érdekében, hogy a táp az alacsony feszültség védelem határa fölött maradjon:

$$C1 = \frac{I_O*t_{ON}}{deltaV} = \frac{1.0A*2{,}53us}{0{,}5V} = 5uF \quad (15)$$

Normális esetben a fent kiszámoltnál alacsonyabb értékű kondenzátor is használható a kapcsolásban, mert a képlet a lehető legrosszabb esetet feltételezi.

Az utolsó kiszámolandó alkatrész érték a lágyindítás időzítéséért felelős kondenzátor. Ennek az alábbi képlettel lehet megadni az értékét:

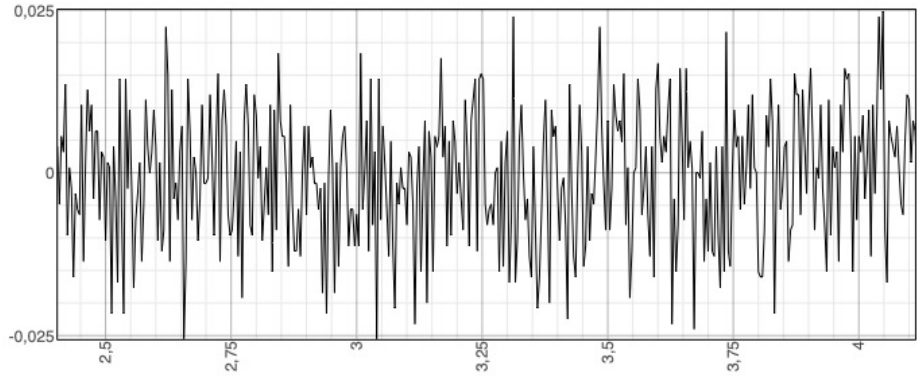
$$C_{SS} = \frac{t_{SS}*11{,}5uA}{2{,}5V} = \frac{5ms*11{,}5uA}{2{,}5V} = 22nF \quad (16)$$

Az így meghatározott alkatrészekkel összeállított áramkör a valóságban az alábbi tulajdonságokkal rendelkezik:

V_{OUT} 4,98V

V_{RIPPLE} 0,05V

A *4. ábrán* látható szintű kimeneti feszültség ingadozás tapasztalható a tápegységen. Ebben az ingadozásban benne van a 3 darab órajel forrás működéséből keletkező zaj is, eszerint a tápegység zaja terheletlenül még kisebb lehet.

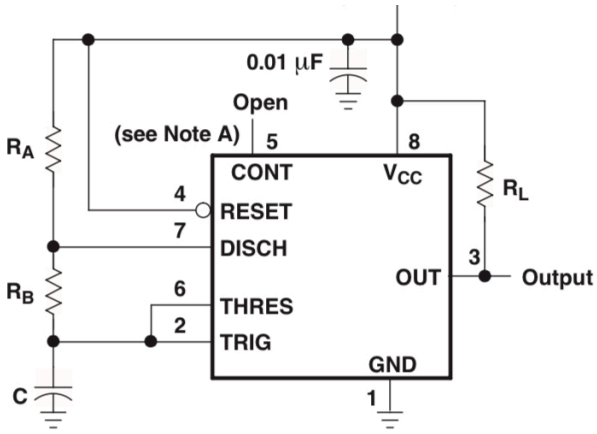


***4. ábra** Tápegység feszültség ingadozása*

4.2 Órajel

4.2.1 Órajel generátor

Ugyanezen a panelen kapott helyet a rendszer órajelének létrehozásáért felelős áramkör is. Itt egy nagyon fontos funkció, a hardveres módosítás nélküli frekvencia állítás is megvalósításra került.



5. ábra *Az NE555 astabil áramköri rajza [4]*

A stabil négyszögjel létrehozásáért az *5. ábrán* látható módon bekötött NE555 integrált áramkör felel. Ennek a kimeneti frekvenciája az adatlap alapján az alábbi képlettel állapítható meg: [4]

$$t_H = 0{,}693(R_A + R_B) * C \tag{17}$$

$$t_L = 0{,}693(R_B) * C \tag{18}$$

$$period = \ t_H + t_L = 0{,}693 * (R_A + 2R_B) * C \tag{19}$$

A rendszerben 3 különböző órajel előállítására van szükség:

- 128Hz
- 32768Hz
- 100000Hz

Ez utolsó érték előállítására az NE555 nem alkalmas, de ekkora frekvenciát az áramkör se tud kezelni, így itt a lehetséges maximum 100kHz frekvenciára tervezek. [4]

4.2.2 128 Hz

Az egyszerűség kedvéért a C értékét változtatom csak, R_A és R_B értékeit állandóként kezelem.

$R_A = 100\Omega$

$R_B = 10k\Omega$

$$\frac{1}{128}s = 0{,}693 * (100\Omega + 20000\Omega) * C \quad (20)$$

$$C[F] = \frac{\frac{1}{128}s}{\frac{20100\Omega}{0{,}693}} = \mathbf{560nF} \quad (21)$$

Kitöltési tényező:

$$t_H = 0{,}693(100\Omega + 10000\Omega) * 560nF = \mathbf{0{,}0039s} \quad (22)$$

$$\frac{3{,}9*10^{-3}s}{\frac{1}{128}s} = \mathbf{50{,}17\%} \quad (23)$$

4.2.3 32768Hz

$$\frac{1}{32768}s = 0{,}693 * (100\Omega + 20000\Omega) * C \quad (24)$$

$$C[F] = \frac{\frac{1}{32768}s}{\frac{20100\Omega}{0{,}693}} = \mathbf{2{,}19nF} \quad (25)$$

Kitöltési tényező:

$$t_H = 0{,}693(100\Omega + 10000\Omega) * 2{,}19nF = \mathbf{0{,}00001525s} \quad (26)$$

$$\frac{1{,}525*10^{-5}s}{\frac{1}{32768}s} = \mathbf{49{,}97\%} \quad (27)$$

4.2.4 100000Hz

$$\frac{1}{100000}s = 0{,}693 * (100\Omega + 20000\Omega) * C \quad (28)$$

$$C[F] = \frac{\frac{1}{100000}s}{\frac{20100\Omega}{0{,}693}} = \mathbf{718pF} \quad (29)$$

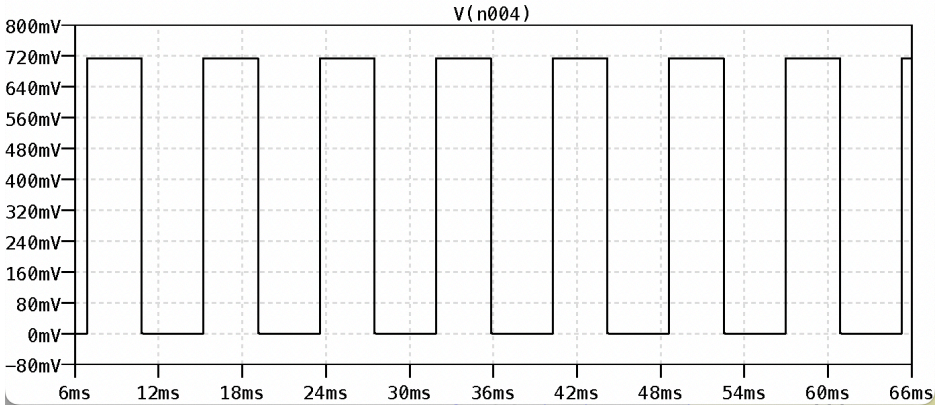
Kitöltési tényező:

$$t_H = 0{,}693(100\Omega + 10000\Omega) * 718pF = \mathbf{0{,}0000050s} \quad (30)$$

$$\frac{5*10^{-6}s}{\frac{1}{100000}s} = \mathbf{50\%} \quad (31)$$

4.3 Órajelek szimulációja

4.3.1 128Hz

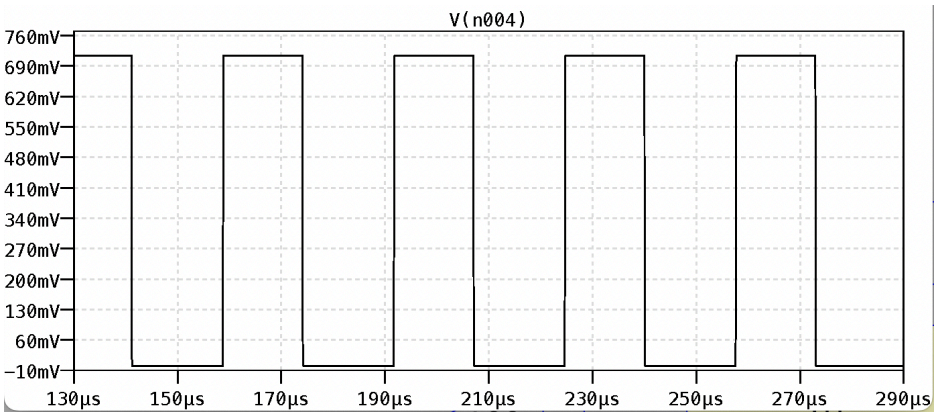


6. ábra *128Hz szimuláció kimenete*

A szimulációkhoz az LTSpice, ingyenesen elérhető szoftvert használtam fel.

Az első áramkör kimeneti jele a *6. ábrán* látható, a kiszámolt értékekkel 119,87Hz frekvencián üzemelt, ami 6,35%-os eltérés a kívánt értékhez képest.

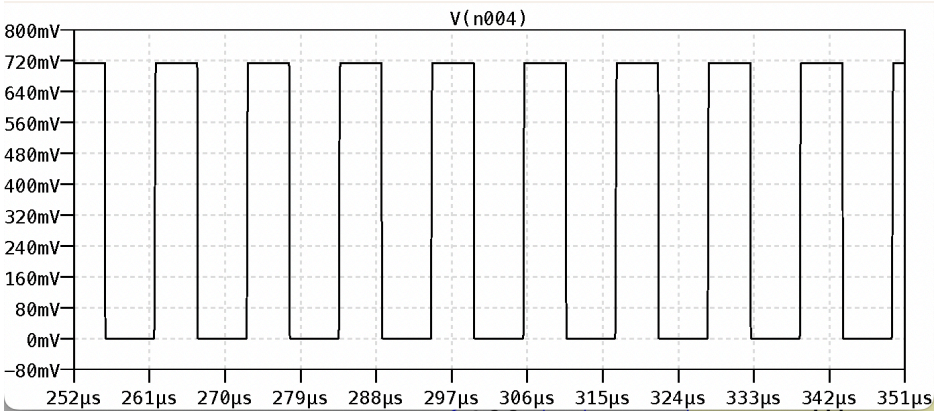
4.3.2 32768Hz



7. ábra *32768Hz szimuláció kimenete*

A második áramkör frekvenciájának szimulációja a *7. ábrán* látható, ami szerint a frekvencia 30,347kHz. A szimulált érték 7,38%-kal kisebb a kívánt értéknél. A jel magas értékű ideje 15,338uS, a kitöltése 46,55%.

4.3.3 10000kHz



8. ábra *A 100000Hz szimuláció kimenete*

A harmadik áramkör szimulációs eredményét a *8. ábra* mutatja, amiről leolvasható, hogy a frekvencia 90,96kHz, ami 9,04%-kal tér el a kívánt értéktől. A jel magas szint ideje 4,99uS, a kitöltése 45,4%.

4.4 A valós áramkör mérései

A korábban kiszámolt értékekkel összeállítottam a három darab generátort a valóságban is, amik az elvégzett mérések alapján jelentős eltérést mutatnak a számítások alapján várt értékekhez képest. Ennek az oka az alkatrészek gyártása során alkalmazott tűrésekre vezethető vissza. Míg az ellenállások, amik beépítésre kerültek 1%-os eltérést engednek meg a feltüntetett értékükhöz képest. Ezek szerint tehát egy 10kΩ értékű ellenállás lehet 9900Ω vagy 10100Ω, és a kettő között bármennyi.

60,24kHz 25kHz 72,99Hz

4.4.1 128Hz

A 128Hz-es áramkör megvalósítása után a kapott kimeneti frekvenciája 72,99Hz, ami 48,76% eltérést jelent a kívánt értékhez képest.

Mivel a kiszámolt 560nF érték nem szabvány, így ezt kettő vagy több más értékű kapacitás kombinálásával kell előállítani. Két darab 1uF kondenzátort sorba kötve 500nF értéket kaptam.

Ha ezek az ellenállások tökéletesek és pontosan megfelelnek a feltüntetett értéknek, akkor a következő képlet alapján a rendszer frekvenciája

$$T_{periódus} = 0{,}693 * (100\Omega + 20000\Omega) * 500nF = 0{,}0069646s \tag{32}$$

$$\frac{1}{0{,}0069646} s = 143{,}58Hz \tag{33}$$

kell legyen.

A nagy frekvencia eltérésen felül a jelalak se tökéletes, nagyon alacsony a kitöltési tényező.

A két párhuzamosan kötött kondenzátor együttesen az alábbi két érték által megadott tartományon belül bármekkora értéket felvehet:

$$\frac{1}{\frac{1}{0{,}000001F*0{,}8}+\frac{1}{0{,}000001F*0{,}8}} = 400nF \tag{34}$$

$$\frac{1}{\frac{1}{0{,}000001F*1{,}2}+\frac{1}{0{,}000001F*1{,}2}} = 600nF \tag{35}$$

Ezen kívül az ellenállások is befolyásolják a frekvenciát kis mértékben.

A lehetséges valós értékekkel számolva az alábbi két érték között kell, hogy beálljon a frekvencia, ha más befolyásoló tényező nincs a rendszerben:

$$\frac{1}{0{,}693*(100\Omega*0{,}99+20000\Omega*0{,}99)*400nF} = 181{,}29Hz \tag{36}$$

$$\frac{1}{0{,}693*(100\Omega*1{,}01+20000\Omega*1{,}01)*600nF} = 118{,}47Hz \tag{37}$$

Mivel az így kapott tartományon is jelentősen kívül esik a valós mérés, egyéb tényezők is befolyásolják a működést. Az NE555 integrált áramkör felépítését áttanulmányozva látható, hogy az 50%-os kitöltési tényező nem megvalósítható stabilan az egyszerű astabil kapcsolással.

Kettő féle megoldás lehetséges a kívánt kitöltés eléréshez:

- a control(5) lábat egy 10kΩ ellenállással a földre húzva,
- a kimeneten keresztül tölteni a kapacitást.

Az első megoldás a jelenlegi áramkör jelentős módosítása nélkül is megoldható, a control lábon már található egy alkatrész, amihez hozzá lehet forrasztani a 10kΩ ellenállást.

Ezzel a megoldással a belső komparátorok referencia értékeit lehet változtatni, ezzel lehetővé téve a kívánt működést. Ezzel a magoldással az alacsony szintű kitöltési tényező:

$$T_{alacsony} = \ln(2) * R_b * C \tag{38}$$

A magas kitöltési tényezője pedig az alábbi módon alakul:

$$T_{magas} = \ln\left(\frac{3}{2}\right) * (R_a + R_b) * C \tag{39}$$

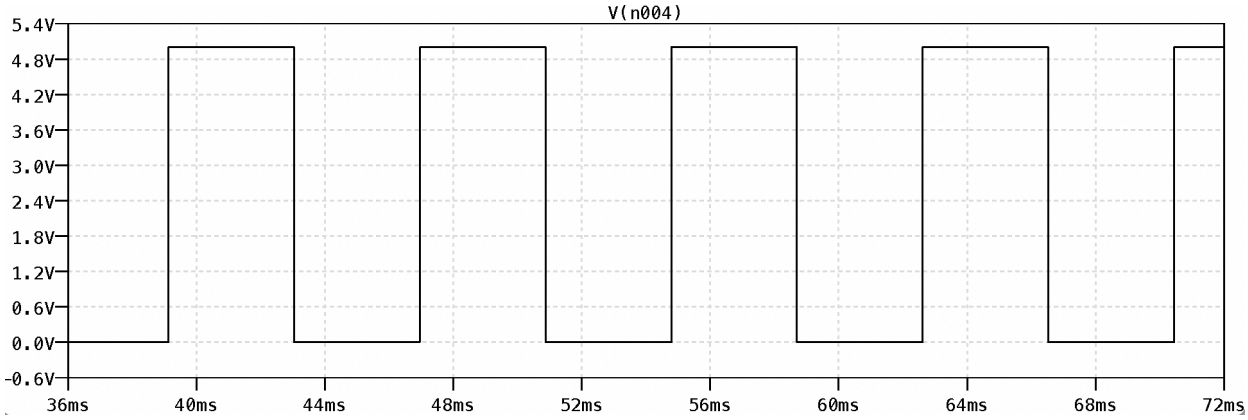
$$T_{periódus} = T_{alacsony} + T_{magas} \tag{40}$$

A 128Hz-es jel előállításához a $T_{alacsony}$ idő 3,9ms, ez a jelenlegi kapacitással az alábbi ellenállás értékkel jön létre:

C = 500nF

$$R_b = \frac{T_{alacsony}}{\ln(2)*C} = \frac{\frac{\frac{1}{128}s}{2}}{\ln(2)*0{,}0000005F} = 11{,}271k\Omega \tag{41}$$

$$R_a = \frac{T_{magas}}{\ln\left(\frac{3}{2}\right)*C} - R_b = \frac{\frac{\frac{1}{128}s}{2}}{\ln\left(\frac{3}{2}\right)*0{,}0000005F} - 11{,}271k\Omega = 7{,}997k\Omega \tag{42}$$



9. ábra *Módosított 128Hz szimulációja*

Az újra számolt értékekkel módosítottam a szimulációs modellt, ami a *9. ábrán* láthatóan a várt eredményt adta vissza, a mért frekvencia 126,88Hz, ez 1,65% pontosságot jelent.

A számítások szerint kapott értékű alkatrészeket nem lehet megvenni, így az azokhoz legközelebbi értékeket kell beépíteni a rendszerbe.

A 11,271kΩ-hoz legközelebbi szabványos érték a 11,3kΩ, A 7,997kΩ-hoz legközelebbi a 8,06kΩ, míg a kívánt kapacitást kettő darab kondenzátor párhuzamos kötésével, 470nF és 27nF értékkel lehet legközelebb elérni. Az így leszimulált áramkör frekvenciája 128,34Hz.

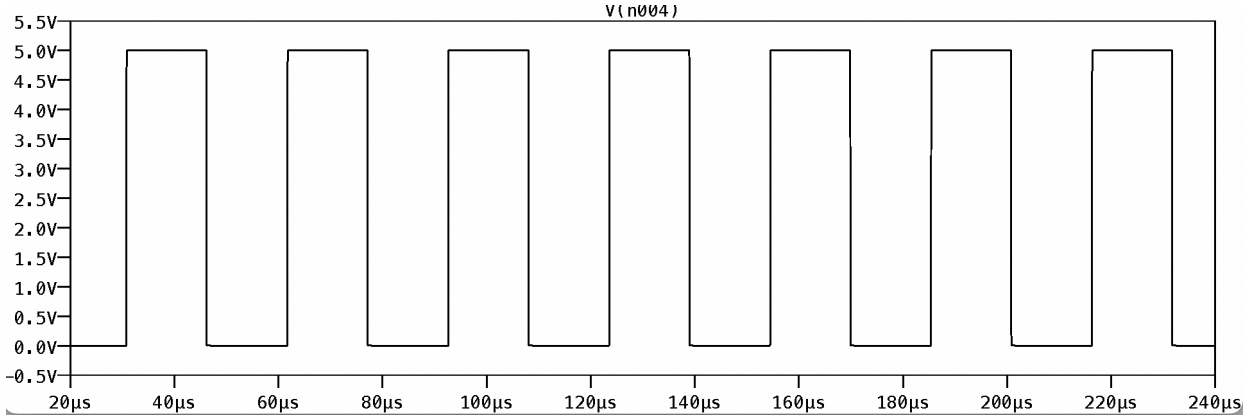
4.4.2 32768Hz

Az előző pontban tárgyaltak alapján, ennek a forrásnak az értékeit a következőképpen kell módosítani:

C = 1nF

$$R_b = \frac{T_{alacsony}}{\ln(2)*C} = \frac{\frac{\frac{1}{32768}s}{2}}{\ln(2)*1nF} = 22{,}014k\Omega \tag{43}$$

$$R_a = \frac{T_{magas}}{\ln\left(\frac{3}{2}\right)*C} - R_b = \frac{\frac{\frac{1}{32768}s}{2}}{\ln\left(\frac{3}{2}\right)*1nF} - 22{,}014k\Omega = 15{,}619k\Omega \tag{44}$$



10. ábra *32768Hz-es módosított 555 áramkör szimulációja*

A *10. ábrán* láthatóan a korábbiakban tapasztaltakkal összhangban, a módosított NE555 áramkör sokkal pontosabban hozza létre az 50%-os kitöltési tényezőt, a szimulált frekvencia 32378Hz, ami 1,2% eltérés a tervezett értékhez képest.

A kiszámolt értékekhez legközelebbi szabványos értékek a 15,4kΩ és a 22,1kΩ, amikkel a szimuláció szerint a frekvencia 32347Hz.

4.4.3.100000Hz

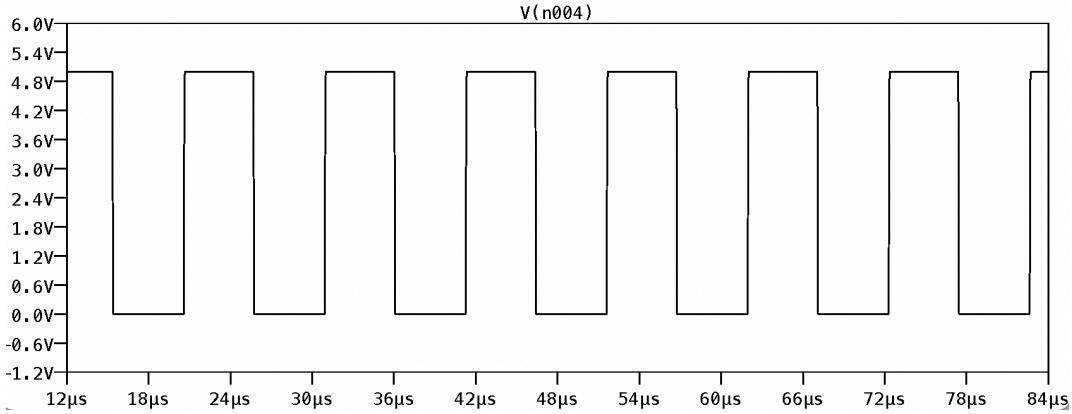
Az előző pontokban tárgyaltak alapján, ennek a forrásnak az értékeit a következőképpen kell módosítani:

C = 1nF

$$R_b = \frac{T_{alacsony}}{\ln(2)*C} = \frac{\frac{\frac{1}{100000}}{2}s}{\ln(2)*1nF} = 7{,}213k\Omega \tag{45}$$

$$R_a = \frac{T_{magas}}{\ln\left(\frac{3}{2}\right)*C} - R_b = \frac{\frac{\frac{1}{10000}}{2}s}{\ln\left(\frac{3}{2}\right)*1nF} - 7{,}213k\Omega = 5{,}118k\Omega \tag{46}$$

Az átalakított áramkör frekvenciája a *11. ábrán* láthatóan 96719Hz, ami 3,28% eltérés a várthoz képest. A legközelebbi szabványos értékek az 5,11kΩ és 7,15kΩ. Ezekkel a szimulált frekvencia 97297Hz.

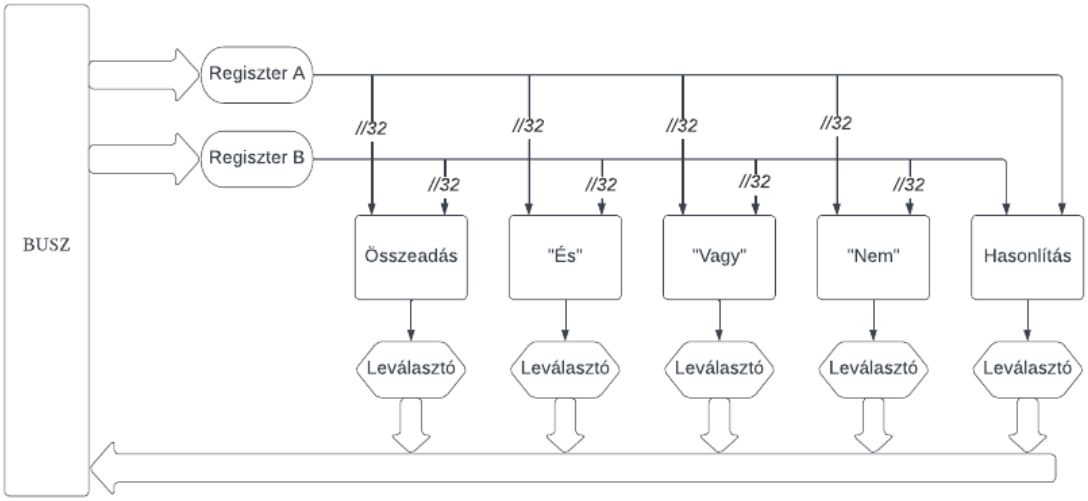


11. ábra *Átalakított 100000Hz szimulációja*

5. Regiszterek és ALU

A számítógép egyik legfontosabb feladata a különböző számokkal való számitások és logikai műveletek elvégzése. Ennek a funkciónak az ellátáshoz az aritmetikai és logikai egységre van szükség.

5.1 Aritmetikai és logikai egység elméleti felépítése



12. ábra *A teljes ALU modul felépítése*

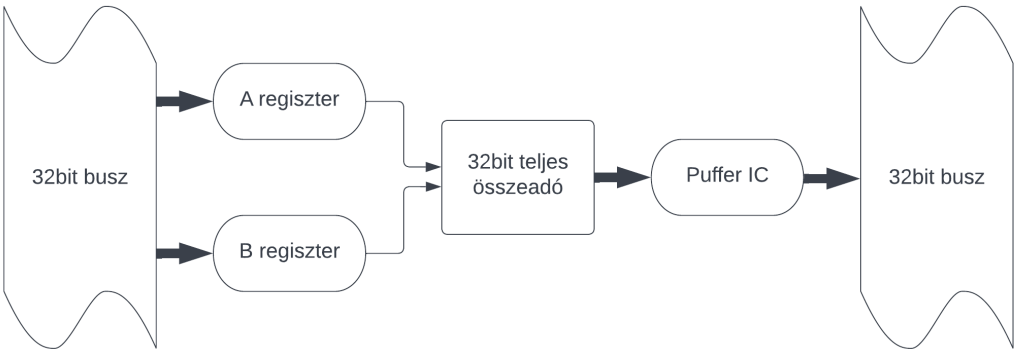
Az aritmetikai és logikai egység (továbbiakban ALU) elméleti felépítése látható a *12. ábrán*. A fő

adatbuszról a művelet elemei két darab egyenként 32 bites regiszterbe kerülnek be. A regiszterek kimenetei folyamatosan aktívak, így az összes műveletet végző egység egyszerre megkapja az értékeket. Az egységek kimenetei is folyamatosan aktívak, így szükség van egy leválasztásra annak érdekében, hogy ne írják felül egymást és adjanak ki hibás eredményt. Erre a feladatra egy busz transzmitter IC-re van szükség.

5.1.1 Regiszterek

Az ALU működése során az értékeket, amikkel műveleteket végez a regiszterében tárolja. Ennek az eszköznek képesnek kell lennie eltárolni 1 byte adatot, azután is, hogy a bemenetén az már nem elérhető. Erre a feladatra egy D típusú tároló egység a megfelelő.

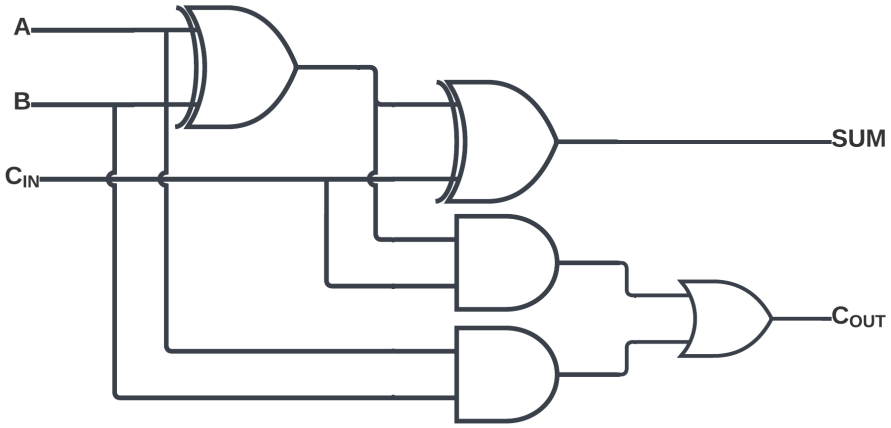
5.1.2 Összeadás



13. ábra *A bináris összeadó logikai felépítése*

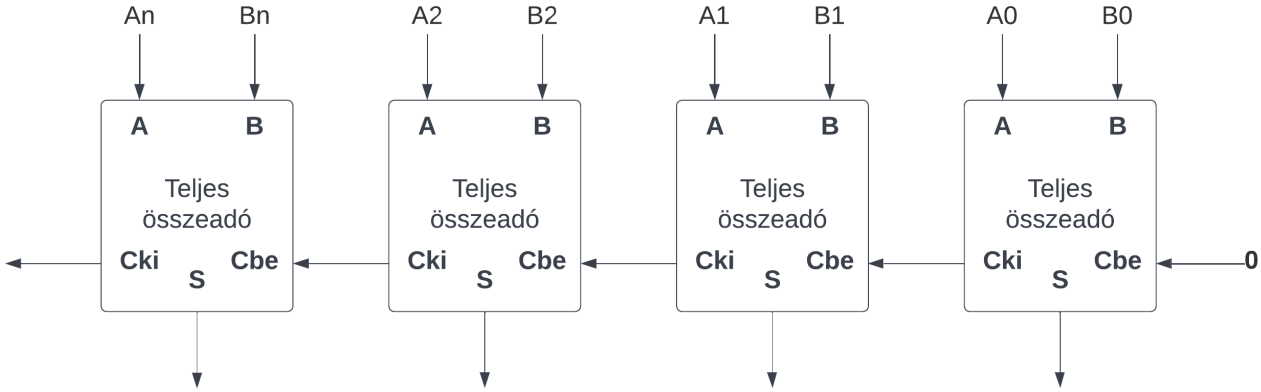
A *13. ábrán* látható egy általános 32 bites összeadó egység elméleti felépítése, ami 3 fő részből áll:

- Regiszterek a változók tárolásához
- Az összeadást végző IC-k csoportja
- Puffer IC a leválasztáshoz



14. ábra *1 bites teljes összeadó [8]*

Az egyik legalapvetőbb művelet az összeadás művelete. Két szám egy-egy bitjének az összeadásához a *14. ábrán* látható összeadó áramkörre van szükség.



15. ábra *Az 1 bites összeadók összekötése kívánt felbontáshoz*

A *15. ábrán* látható, hogy miként kell kaszkádba kötni az 1 bites teljes összeadókat, hogy elméletileg korlátlan szó hosszúsággal tudjon a rendszer összeadásokat elvégezni. Azért csak elméletileg, mert a valóságban az egységek késleltetése összeadódik, így egy bizonyos felbontás után a művelet elvégzése irracionálisan hosszú ideig tartana.

A fent szemléltetett rendszer két darab „n” hosszúságú számból egy darab „n” hosszúságú számot ad ki eredményként és egy vivő jelet. Ez a vivő jel használható a vezérlő egység számára, hogy detektálni tudja, ha túlcsordul az összeadó. Ez akkor történhet meg, ha a két összeadandó szám egyenként belefér a hosszba, viszont összegük már nem. Ilyenkor a szoftver képes lehet ezt a hibát megoldani.

Az S kimeneteken folyamatosan kiadja az eszköz a két szám összegét, így ezeket közvetlen nem lehet az adatbuszra kötni, egy köztes leválasztó IC-re van szükség.

5.1.3 Logikai „és”, logikai „vagy”

A logikai „és”, logikai „vagy” műveletét elvégző egységeket egyben lehet kezelni, mivel ezek fizikai és elméleti felépítése sem tér el egymástól, mindössze a bennük felhasznált logikai kapuk különböznek.

Ezekkel a műveletekkel képes a számítógép a külső adatokat szűrni, azokból csak a lényeges információkat kivenni. A későbbiekben tárgyalásra kerülő be- és kimeneti egység regisztereit nem bitenként, hanem byte méretben olvassa ki a rendszer. Ahhoz, hogy szűrni tudjon egy kívánt bemeneti láb értékére, egy logikai „és” műveletet kell végrehajtania.

Egy példa erre:

A számítógép bemeneti csatlakozóján a kombináció **00101110**.

A rendszer a 6.láb értékére kíváncsi, de ha üresen csak annyit vizsgál, hogy a kiolvasott érték 0 vagy sem, akkor fals értéket is kaphat, hiszen több bemenet értéke is 1, azaz logikai igaz.

Ezért a logikai és műveletével kiszűri a szükséges lábat:

00101110 & 00100000 = **00100000**

Így már egy egyszerű művelettel meg tudja állapítani, hogy a vizsgált láb milyen értéket vett fel.

A logikai „vagy” művelete hasonló módon fut le, annak a kimenet beállításánál van jelentősége:

A kimenet kezdéskori állapota: **00110011**

Azt akarja a rendszer, hogy a 7. láb értéke is 1 legyen.

Ekkor a „vagy” művelettel összevonja a két számot, megkapva a kívánt kimeneti kombinációt.

00110011 | 01000000 = **01110011**

Ez a művelet azonban nem alkalmas arra, hogy egy már 1-es értéken álló kimenetet 0-ba állítson.

Ehhez ismét logikai „és” műveletre van szükség, viszont most egyedül az a bit kap 0 értéket, amelyikhez tartozó kimeneti láb értékét szeretné a rendszer 0-ba állítani:

01110011 & 10111111 = **00110011**

5.1.4 Logikai „nem”

A számítógép az összeadás mellett képes kell legyen két szám különbségét is megadni. Ehhez nincs szükség másra, csak a már meglévő összeadó modulra, valamint egy invertáló egységre, vagyis a logikai „nem” érték létrehozására.

Két szám bináris formában vett szám különbségét úgy kapja meg a rendszer, ha a kivonni kívánt számot invertáltja, hozzá ad 1-et, és a keletkezett érték összegét veszi a másik számmal.

Egy egyszerű példa erre:

A számok: 53 és 19

53 binárisan: 00110101

19 binárisan: 00010011

53 – 19 = 34 00100010

!19 **11101100**

!19+1 **11101101**

53 00110101

(!19+1) 11101101

+ 00100010

Az így keletkezett szám nem tökéletesen 34, mivel a legmagasabb helyiérték 1-es értéket vesz fel, azonban ez a fix felbontású, terv szerint 32 bites rendszerben túlcsordulás bit-ként fog megjelenni az eredmény után, ezzel nem befolyásolva a számítást.

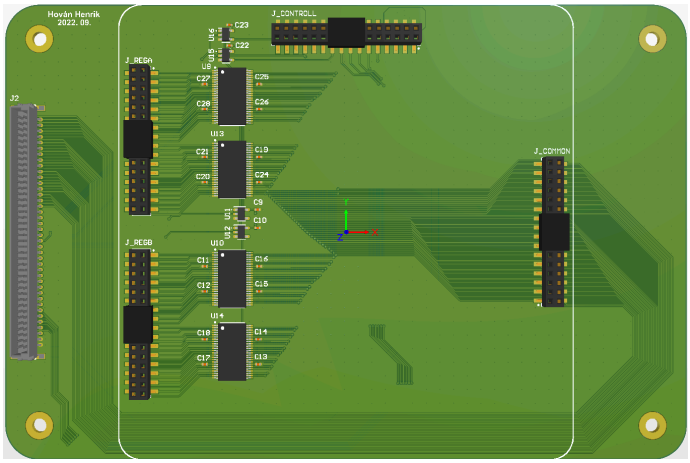
5.1.5 Hasonlítás

Két szám logikai és matematikai összevonásán kívül szükséges lehet bizonyos műveletek végrehajtásához azok egymáshoz képesti viszonyának megállapítása. Leggyakrabban olyan esetekben lehet erre szükség, amikor valaminek csak egy körülmény bekövetkeztekor kell megtörténnie. Ilyen művelet a kondíciós ugrás, amivel megvalósíthatóak a magasabb programozási nyelvek for és while ciklusai.

5.2 Aritmetikai és logikai egység gyakorlati megvalósítása

5.2.1 Hordozó panel és regiszterek

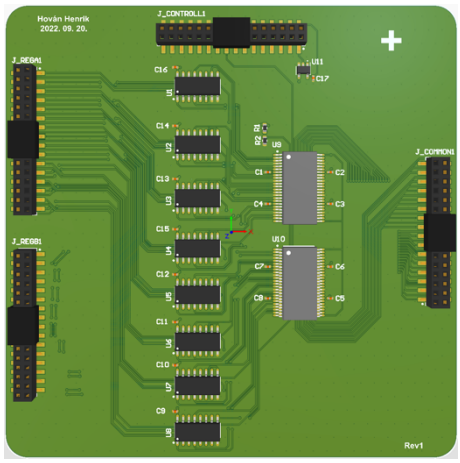
Az aritmetikai és logikai egységet egy rétegesen felépített áramkör csoportként terveztem meg. Ennek a csoportnak az alapját a két darab regisztert tartalmazó hordozó panel adja.



16. ábra *Regiszterek és modul csatlakozók*

A regisztereket két-két darab 16 bites S-R tároló alkotja, a műveleteket elvégző modulokat a *16. ábrán* láthatóan 4 darab tüskesor segítségével lehet csatlakoztatni. A modul jobb oldalán látható az egyes egységeket összekötő sín csatlakozó, amire az összes vezérlő jel és a 32 bites adatsín is ki van vezetve.

5.2.2 Összeadó

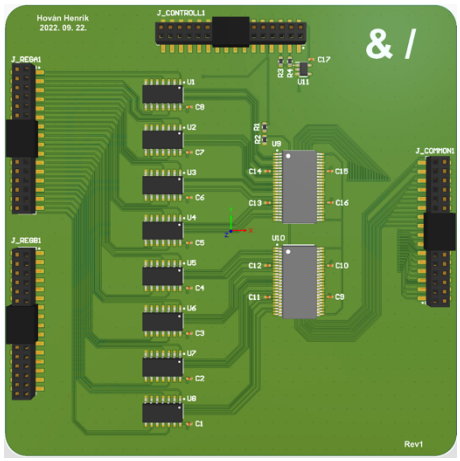


17. ábra *Az elkészült összeadó modellje*

Az összeadás műveletét az elméleti felépítésben tárgyalt integrált áramkörökkel oldottam meg, a *17. ábrán* látható fizikai felépítéssel. Az összes modul alján és tetején is megtalálhatóak a csatlakozók, így azok tetszőleges sorrendben dughatók össze.

5.2.3 Logikai „és”, logikai „vagy”, logikai „nem”

Ezeket a modulokat egyként kezelem a gyakorlati megvalósítás során, aminek modellje a *18. ábrán* látható. Felépítésük teljes mértékben azonos, csak a rajtuk megtalálható integrált áramkörök térnek el. Ezt az teszi lehetővé, hogy a Texas Instruments, akiknek az eszközeit használom, a logikai kapuk esetében teljes lábkompatibilitást tesznek lehetővé az egyes alkatrészek között.



18. ábra *Logikai modulokat alkotó panel modellje*

6. Memóriák

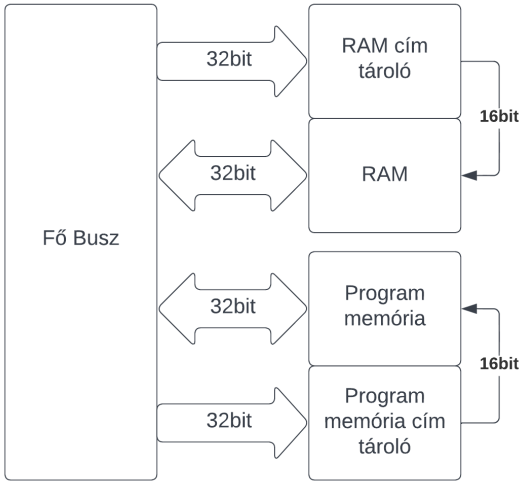
Ahhoz, hogy teljes értékű számítógép lehessen az eszköz, képesnek kell lennie a belső tárolójában található programok végrehajtására és a műveletek eredményeinek eltárolására, utólagos visszahívására. A programok tárolására egy olyan háttértárra van szükség, ami a tápellátás elvétele majd visszakapcsolása után változatlanul tárolja a benne lévő adatokat. Ilyen adattároló egység az EEPROM vagy a FLASH. Az EPROM angolul az Erasable Programmable Read Only Memory azaz törölhető programozható csak olvasható memória rövidítése. Ezeknek a memóriatípusoknak a programozása a megszokott üzemi feszültségnél jelentősen magasabban, jellemzően 12V-on történik. Ahhoz, hogy a bennük tárolt adatot törölni lehessen nagy erősségű, 300nm-nél rövidebb hullámhosszúságú UV fénynek kell kitenni, a rajta található ablakon keresztül. Ekkor a teljes memória törlődik. Az EEPROM az elektronikusan törölhető, programozható, csak olvasható memória

rövidítése. Felépítése hasonló a EPROM-hoz, a bennük lévő lebegő kapus mosfetek szigetelési rétege viszont vékonyabb. A technológiát 1978-ban fejlesztették ki, és egészen a FLASH memória megjelenéséig ez volt a legjobb és legolcsóbb megoldás a nem felejtő memóriák között. A következő technológiai lépés volt a FLASH memóriák megjelenése. az EEPROM-mal ellentétben ezt a memóriatípust blokkonként kell törölni, a korai változatokat pedig csak egyben lehetett.A rendszerben mind a két memória típust felhasználtam különböző feladatokra. A FLASH memória, a bonyolultabb programozási módja miatt az utasítások dekódolásáért felelős tároló részeként, míg az EEPROM a programkódok, állandó értékek mentésére, lehívására került be. Ezeknek a memória típusoknak azonban van egy hatalmas hátrányuk, limitált az írási élettartamuk. Egy bizonyos mennyiségű írás művelet után degradálódnak a sokat használt memóriaterületek. Ezek a memória típusok alkalmatlanok a folyamatos üzem során keletkező változó értékek tárolására, arra valami más típus kell. Ezzel a memóriatípussal szemben nem kell fennálljon az a feltétel, hogy lekapcsolás után is tárolja az adatokat, hiszen arra alkalmasak a korábban tárgyaltak, cserébe viszont képesnek kell lennie a sokszorosát elviselni annak az írási művelet mennyiségnek, amitől az EEPROM tönkremegy. Ez a memória típus a RAM, vagyis a random access memory (közvetlen hozzáférésű memória).

6.1 Memória modul elméleti felépítése

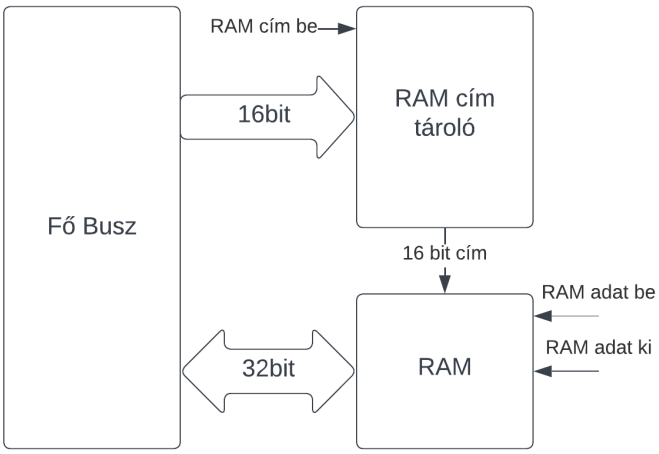
A *19. ábrán* látható a teljes rendszer memória egyszerűsített tömbvázlata a vezérlő jelek nélkül. A RAM és a program memória külön címregisztert kaptak, ezzel lehetővé téve az egy utasításon belüli adatmásolást a két tároló között.

Ha a rendszer teljes 32 bites címméret lehetőségét kihasználná, akkor 4GB RAM és program memória is elhelyezhető lenne a rendszerben, illetve a külső bővítés lehetőségével ez tovább növelhető.



19. ábra *Memória egység tömbvázlata*

6.1.1 RAM



20. ábra *A RAM modul részletes elméleti felépítése*

A RAM egységnek a *20. ábrán* látható módon 3 darab vezérlő jelre van szüksége:

RAM cím be: erre a vezérlő jelre tárolódik a fő adatbuszról a cím tárolóban,

RAM adat be: erre a vezérlő jelre tárolódik a korábban beállított címre az adat a fő adatbuszról,

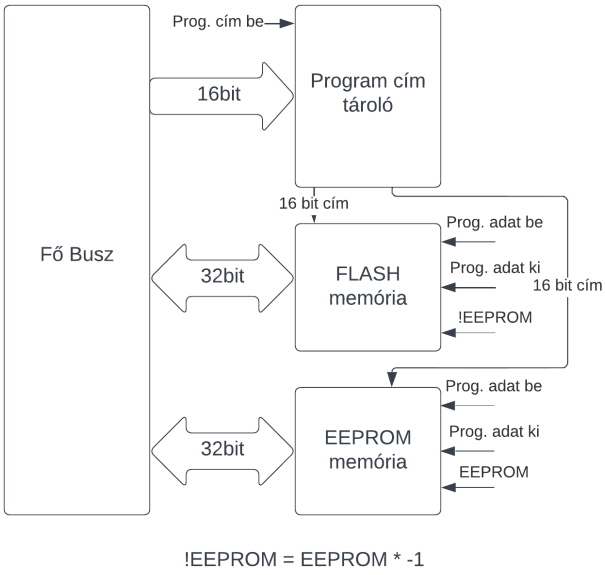
RAM adat ki: erre a vezérlőjelre a memória kimente bekapcsol és a címregiszter által megadott címen tárolt adatot kiadja a fő adatbuszra.

A bemeneti jelek órajel vezérelt jelek, amik csak az órajel felfutásával együtt futnak fel, míg a kimenetet aktiváló jel állandó, az órajel lefutó élével kapcsol fel, és csak a következő órajel lefutással vált állapotot a következő utasítás lépésnek megfelelően.

A kimenet hosszabb távú bekapcsolt állapota lehetővé teszi az egyes modulok közötti adatcserét. Egy modul kimenete az órajel lefutó élével aktiválódik, míg egy másik, célmodul bemenete csak az órajel felfutó élére aktiválódik, amikorra már biztosan beáll a megfelelő állapotra a forrás egység jele.

6.1.2 Program memória

A program memória felépítése a *21. ábrán* láthatóan szinte teljesen azonos a RAM-mal, azzal az eltéréssel, hogy az egyszerűbb változtathatóság miatt egy FLASH és egy EEPROM memória is helyet kapott benne. A FLASH memória írása bonyolultabb feladat, mivel azt csak laponként lehet írni, míg az EEPROM-ot byte-onként.



21. ábra *Program memória elméleti felépítése*

Annak érdekében, hogy a lehető legkevesebb plusz láb igénybevételével tudjon a rendszer választani a FLASH és az EEPROM memória között, csak egy darab kiegészítő jel kerül betervezésre. Ez a jel engedélyezi az EEPROM-ot, és ezzel egyszerre tiltja a FLASH memóriát.

6.2 Memória modul gyakorlati felépítése

A RAM memória megtervezése során a cím tárolóhoz a már korábban, az A és B regiszterek esetén használt 16 bites D típusú latch IC-t használtam fel.

A RAM bemeneti jele órajel vezérelt kell legyen, ezt egy logikai „és” kapuval valósítottam meg. A RAM címtároló bemeneti jele szintén órajel vezérelt kell legyen, a bemenet engedélyező lábhoz hasonlóan, logikai „és” kapuval lett megoldva. A RAM kimenet engedélyezése nem órajel vezérelt, viszont inverz bemenet, emiatt előbb az engedélyező jelet is inverzzé kell alakítani. Ezt a legegyszerűbben egy bit negálóval (logikai „nem” kapu) vagy egy logikai „negált-és” kapuval (NAND) lehet megoldani.

1. táblázat NAND és NOT kapu igazságtáblájának összehasonlítása

NAND			NOT	
A	B	Y	A	Y
0	0	1	0	1
0	1	1	0	1
1	1	0	1	0
1	0	1	1	0

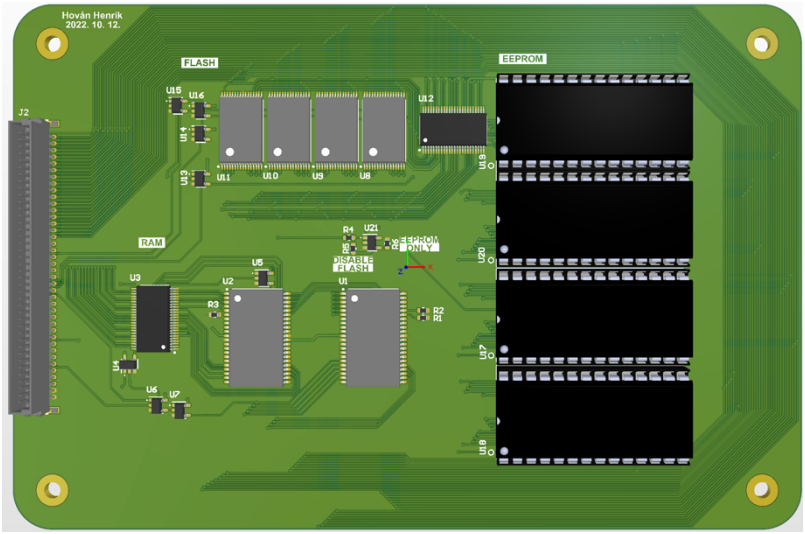
A tervezés során az utóbbi megoldást választottam. Ebben az esetben a kapu két bemeneti lábát össze kell kötni, és így az *1. táblázatban* látható módon azonos lesz a logikai „nem” művelettel.

Ezzel a rendszer alkalmassá vált arra, hogy ha a jövőben olyan memória IC kerül beépítésre, aminek az engedélyező lába nem negált, akkor az ahhoz tartozó hordozó kártya egyszerűen nem kell, hogy tartalmazza a NAND kaput, az utasításkészletet pedig nem kell átalakítani a zavartalan működéshez.

A negálás műveletéhez a 74VHC1G132 típusú, Texas Instruments által gyártott IC-t, az órajel vezérléshez a szintén TI termék, 74AHCT1G08 típust használtam fel. A címeket a SN74ABT16373 típusú 16 bites D típusú tárolóba tárolom.

A RAM-ot két darab, 16x64kbit kapacitású IS61C6416AL-12TLI típusú IC alkotja, míg a Program memóriát négy darab SST39SF040-55-4C-WHE-T FLASH IC és négy darab AT28C64B-15PU EEPROM alkotja.

A rendszer utasításból képes választani, hogy FLASH vagy EEPROM memóriából szeretne adatot betölteni, illetve oda menteni. Ez a megoldás kihagyható és egy ellenállás segítségével valamelyik memóriatípus fixre köthető. Az elméleti felépítés alapján elkészült a memória egység gyakorlati megvalósítása, ami a *22. ábrán* látható.



22. ábra *Az elkészült teljes memória modul*

7. Vezérlő egység

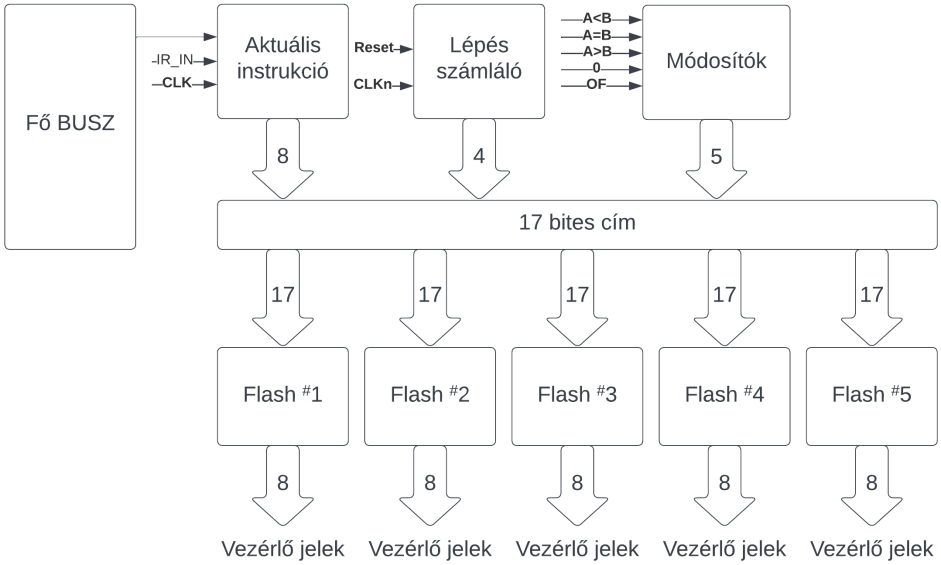
A számítógép különböző moduljainak az együttműködéséhez a szinkront biztosító órajelen kívül egy olyan központi irányításra is szüksége van, ami megadja, hogy mikor milyen egységnek kell feladatot végeznie. Ezt a szerepet a vezérlő egység látja el.

7.1 Elméleti felépítés

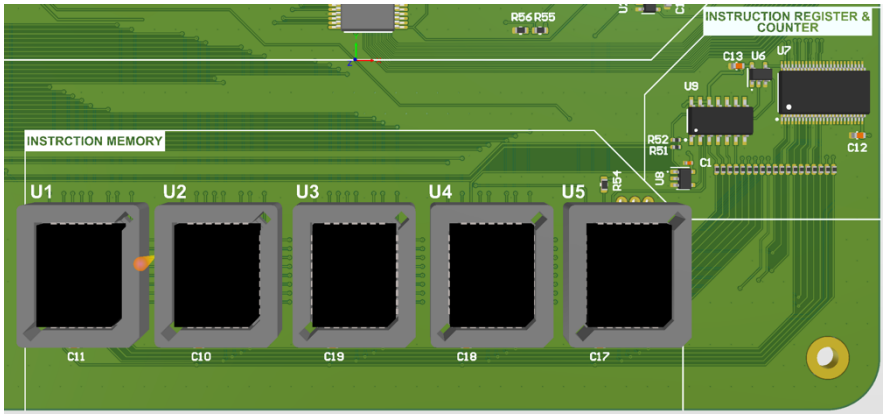
A vezérlő egység elméleti felépítése látható a *23. ábrán*, amin jól látható, hogy három részegységből épül fel:

- aktuális utasítás tároló memória,
- lépés számláló,
- utasítás lépéseit tároló flash memória.

Az aktuálisan végrehajtandó utasítást egy 8 bites D típusú memória tárolja. Egy ilyen utasítás több lépésből is áll, a rendszer felépítése szerint maximum 16-ból. Ahhoz, hogy a rendszer tudja, hogy melyiket kell végrehajtani, a lépés számláló értékét használja fel.



23. ábra *Vezérlő egység elméleti felépítése*



24. ábra *Instrukció memória és lépés számláló*

Az utasítás regiszterben található 8 bites kód, az aktuális lépés azonosítója és a módosító jelek együtt egy címet hoznak létre a memóriában, amin a lépésnek megfelelő kimeneteket felkapcsoló kombináció található. Az elkészült áramkör modelljét a *24. ábra* mutatja.

7.1.1 Módosító jelek

A rendszer egyes moduljai képesek visszajelzést adni a vezérlőegység számára, ezzel módosítva egy utasítás lefutása során a bekövetkező eseményeket. Ilyen módosító jelek a logikai komparátor kimeneti értékei, amik a feltételes ugrás megvalósítása során hasznosak, az összeadó egységből kijövő túlcsordulás jelző, ami egy hiba érzékeléshez, vagy lehetséges hibás eredmény kiszűréséhez

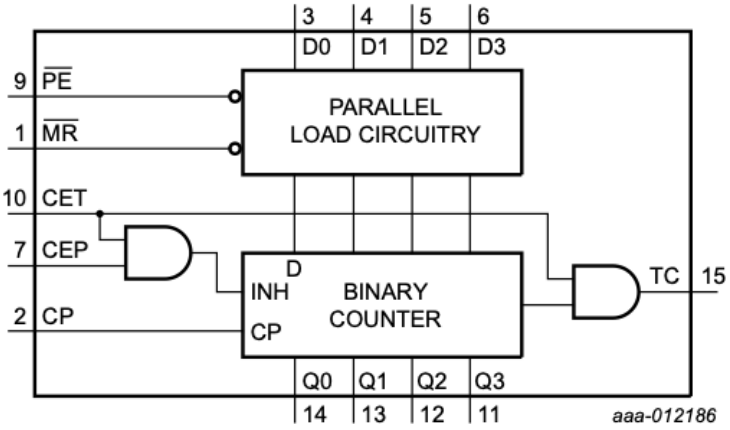
kiváló. Ezen kívül helyet kapott még egy nulla visszajelzés, ami egy újabb feltételes ugrást tesz lehetővé. Ez utóbbit több logikai és kapu összekötésével lehet létrehozni.

Ezen kívül még kettő módosító jel bekötésére van lehetőség, amiket jövőbeni megszakítók implementálásra hagytam meg. A dolgozat írásakor ezekre nem jutott lehetőségem.

7.2 Program számláló

A vezérlő egység részeként terveztem bele a rendszerbe, mégis annak egy sokkal általánosabb eleme a program számláló. Ez az egység adja meg, hogy az utasítás végrehajtása során a program memória melyik pontjáról olvassa ki a következő utasítás azonosítóját vagy az aktuális utasításhoz tartozó adatot és egyéb címet.

A modul felépítése kifejezetten egyszerű, egymás után sorba kötött számláló integrált áramkörökből épül fel, amik órajel és megfelelő engedélyezőjel kombináció hatására végzik a számlálást. A feladat elvégzésére a 74HC163 IC-t használtam fel. Ez az eszköz egy 4 bites előre beállítható bináris számláló. Fontos szempont volt a kiválasztása során, hogy értékét a futás során módosítani lehessen, ezzel elérhető az ugrás parancs implementálása.

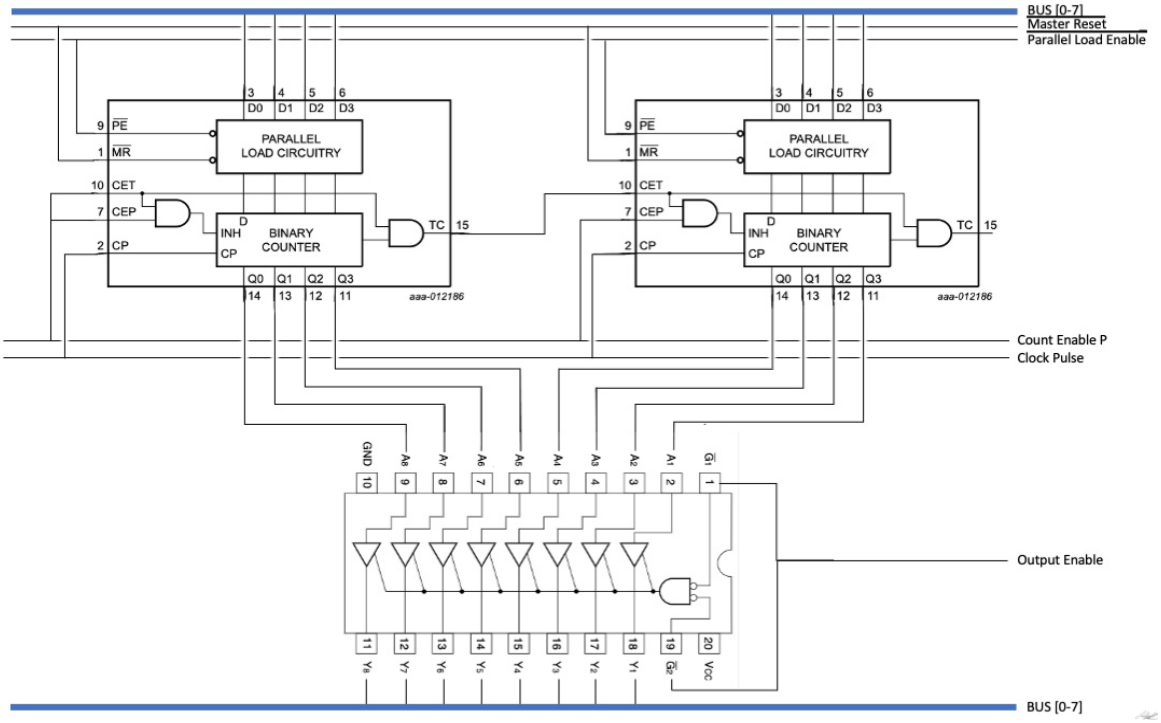


25. ábra *74HC163 adatlap szerinti belső felépítése*

A *25. ábrán* látható logikai felépítéséről a 74HC163 IC-nek kiderül, hogy benne egy 4 bites tároló és egy számláló modul kapott helyet. Mivel ez az eszköz csak 4 bites, így a rendszer szempontjából kevésnek bizonyul. Ennek a problémának a megoldására találhatók meg rajta a CET és TC lábak. [10]

A CET láb közvetlenül össze van kötve egy megelőző ugyanilyen számláló TC lábával. A CET láb logikai 1-es szintje esetén, valamint a CEP logikai 1-es szintje esetén a CP lábon megjelenő felfutó élre az eszköz egyel növeli a számlálója értékét.

Nyolc darab ilyen eszköz sorba kötésével meg is kaptam a kívánt 32 bites számlálási értéket.

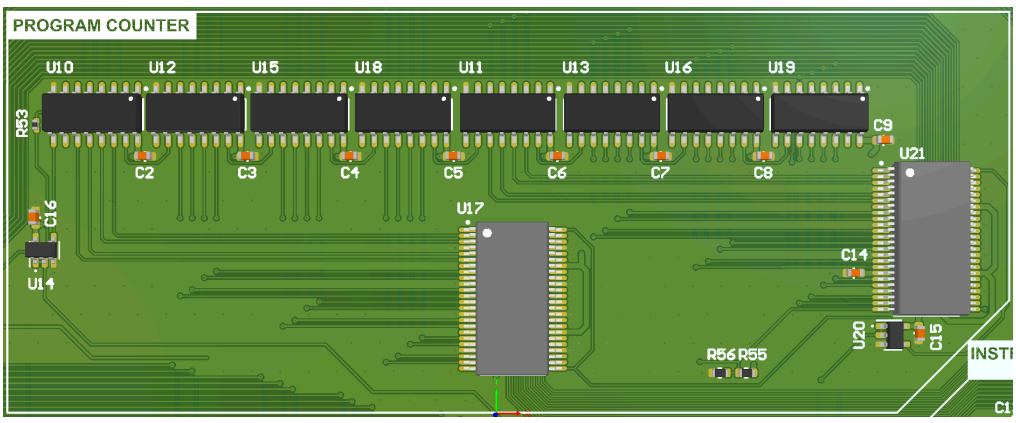


26. ábra *8 bites program számláló egység*

A *26. ábra* mutatja egy 8 bites, 2 darab számlálóból és egy busz meghajtóból felépített áramkör bekötését. Ezen látható, hogy a számlálók bemenete közvetlen a fő buszra van kötve.

Az integrált áramkörön található parallel load enable és master reset lábak inverz aktívak, a hozzájuk kötött folyamatok logikai nulla értékre álláskori lefutó élre futnak le. Ezeket a lábakat a nevük fölött egyenes vonallal szokás jelölni. [10]

Ezt a felépítést tovább gondolva lehet elkészíteni az egység 32 bites változatát, aminek elkészült megvalósítása a *27. ábrán* látható.



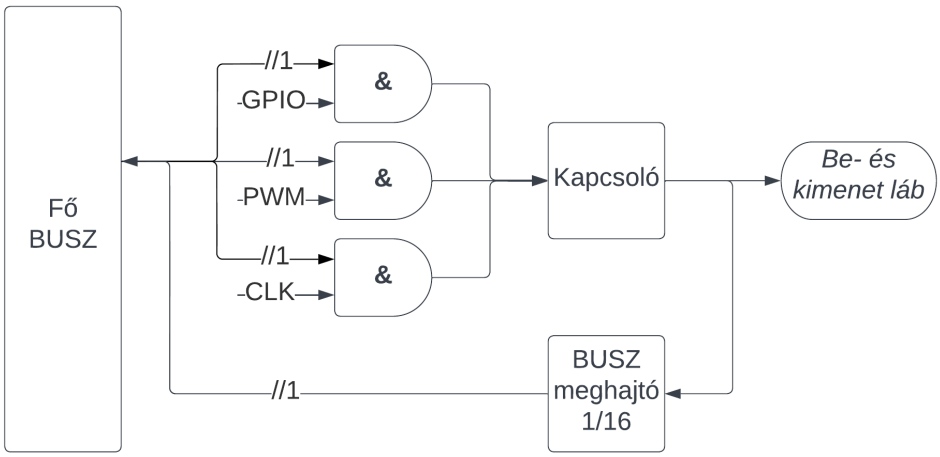
27. ábra *Teljes 32 bites program számláló*

8. GPIO és DIO

Egy modern mikroprocesszor vagy mikrovezérlő esetében akár több száz GPIO (General Porpuse Input Output), azaz általános célú bemenet kimenet is helyet kaphat az eszköz tokozásán. Ezekre a lábakra nem csak feszültséget vagy 0 jelet lehet kapcsolni, hanem az eszközök belső kommunikációs moduljainak és analóg érzékelőinek a kivezetése is rajtuk keresztül megoldott.

A DIO (Digital Input Output) digitális kimenet bemenet ennek egy egyszerűbb változata, ami pusztán csak a logikai 1 és 0 értékek felvételére és érzékelésére alkalmas.

8.1 GPIO és DIO elméleti felépítése

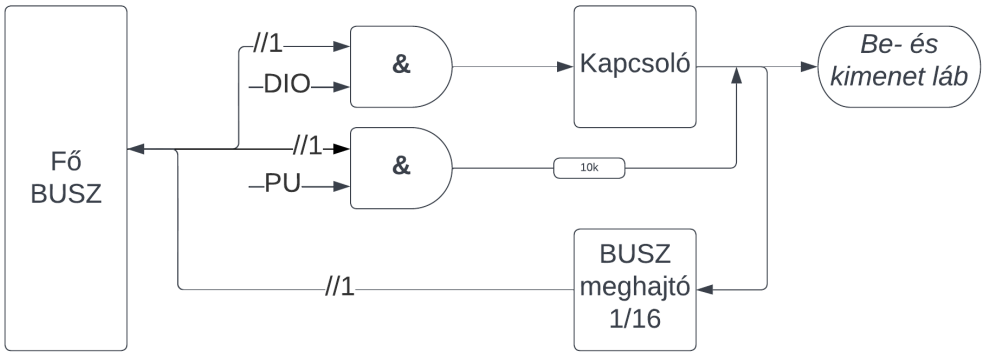


28. ábra *Egy csatornás általános célú be- és kimenet*

A *28. ábrán* látható egy darab általános célú bemenet kimenete logikai felépítése, ami kimenetként három feladatot láthat el:

- egyszerű digitális kimenet,
- egy időzítő kimeneteként PWM kimenet,
- belső órajel kiadása.

A kimeneti multiplexer utáni kapcsoló gondoskodik a leválaszthatóságról, arra az esetre, ha bemenetként szeretném használni a lábat, ekkor a kapcsoló után kötött busz meghajtó egyik bemenete a fő adatbusz hozzá tartozó ágára köti azt. A kész bemeneten helyet kap egy TVS (tranziens feszültség elnyelő) dióda az esetleges kisülések felfogására.

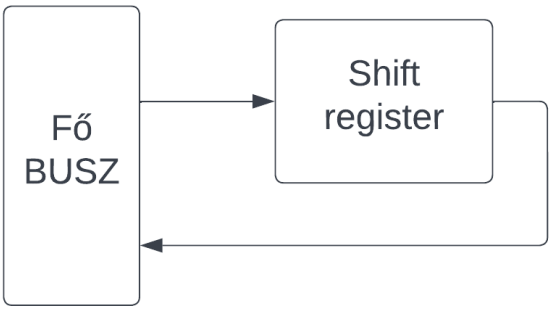


29. ábra *Egy csatornás digitális be- és kimenet elméleti felépítése*

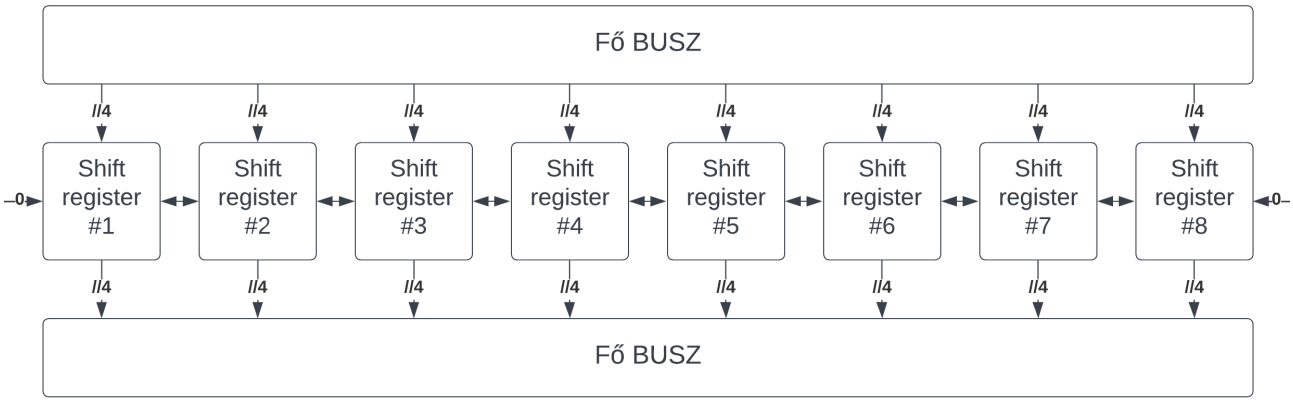
Ezen kívül a *29. ábrán* látható módon néhány GPIO és DIO láb kap úgynevezett felhúzó áramkört, ami egy 10k ellenálláson keresztül magas potenciálra tudja helyezni a lábat úgy, hogy az veszély nélkül földre köthető külsőleg.

9. Logikai eltolás

A bonyolultabb műveletek, mint például az osztás és a szorzás műveletének egyszerűsítésére is használható, de értékek szűrésére és beérkező adatok átalakítására is alkalmas a logikai eltolást végző modul. Ennek az egységnek a felépítését mutatja a *30. ábra*. A logikai eltolás műveletére a 74HC194M típusú integrált áramkört használtam fel. Ez az eszköz egy párhuzamos be- és kimenetű shift regiszter, amiből több darab sorba kötésével lehet elérni a kívánt 32 bites felbontást. Az így kialakított rendszer elméleti felépítését mutatja a *31. ábra*.



30. ábra *Logikai eltolást végző egység egyszerűsített felépítése*



31. ábra A teljes 32 bites logikai eltoló elméleti vázlata.

10. Komparátor

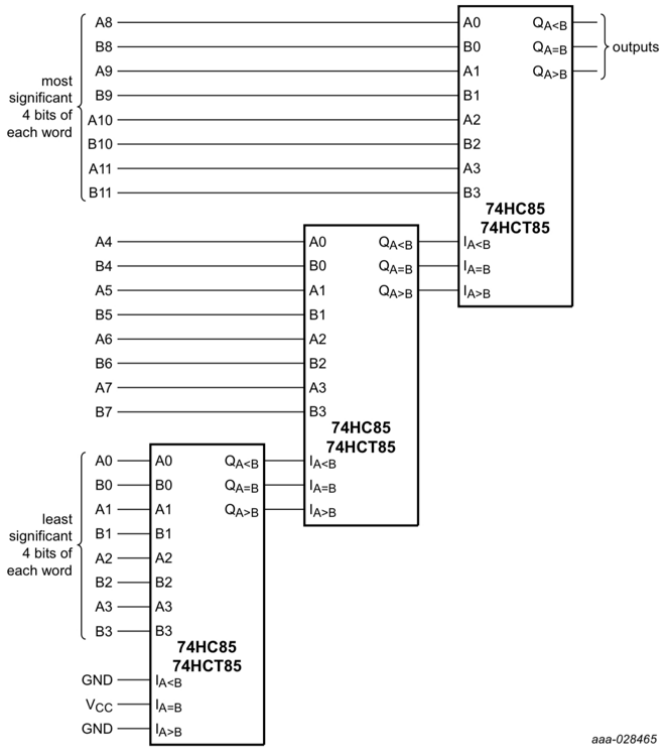
A kompatárot egységnek a feladata az A és B regiszterben található értékek összehasonlítása és az eredmény továbbítása a vezérlő egység felé. Legtöbbször a feltételes ugrás műveletek használják fel ezeket az eredményeket.

A komparátor modul megvalósításához a 74HC85 típusú 4 bites komparátor integrált áramkört használtam fel. Ebből az eszközből 8 darabra van szükség, hogy a kívánt 32 bites számméretet le tudják fedni. A bitek számának növelésével figyelni kell arra, hogy a helyes eredményhez több időre van szüksége az IC- nek mintha csak két 4 bites számot hasonlítana össze.

Egy lehetséges kapacitás növelésre ad példát a *32. ábrán* a Nexpreia által biztosított adatlapban található áramkör. [11]

Ennek a fajta bekötésnek az a legnagyobb hátránya, hogy minden egyes IC bekötésével a várakozási idő is nő, ami szükséges ahhoz, hogy végleges eredményt adjon ki magából a rendszer. Az adatlap által leírt legnagyobb érték, ami a bemenetek megjelenése és a hiteles kimenetek megjelenése között eltelik 22ns. Mivel a második IC csak akkor kezdhet el valós értékekkel dolgozni, ha az első végzett, és így tovább a többi is, ahhoz, hogy két 32 bites szó viszonyát megállapítsuk 176ns időre van szükség.

Ezen felül további tranziens időket is figyelembe kell venni az áramkörben, ilyen például a kimeneti busz illesztő IC tranziens ideje. Összességében 200ns elég az eszköz bemeneti jeleinek beállása és a kimeneti érték leolvashatósága között, ami nagyjából 1MHz sebességű óra jelet tesz lehetővé.



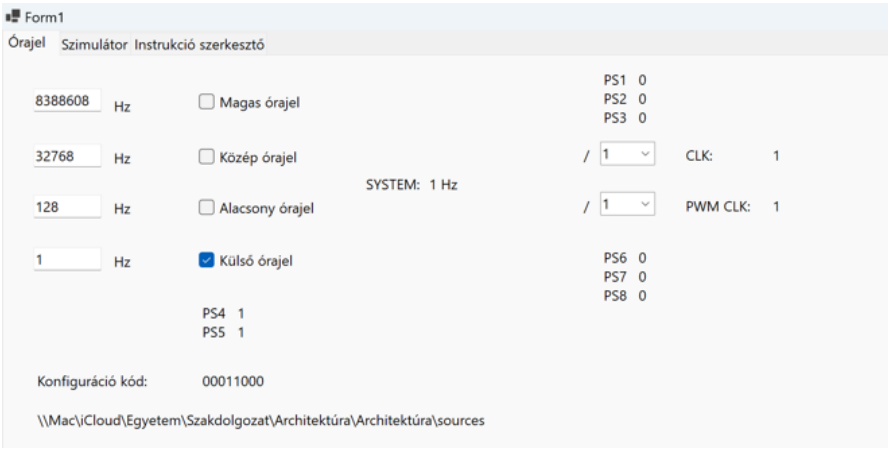
32. ábra *74HC85 komparátor IC-k kaszkád kötése a felbontás növeléséhez*

11. Szimulátor

A tervek szerint a kész rendszer összeállításra kerül a valóságban is, azonban ahhoz, hogy azon a lehető legkevesebbet keljen az olyan hibák miatt álljak, amiket valójában egy hibásan felépített logika okoz az utasításkészletben, szükségem van egy gyorsan módosítható, könnyen átlátható teszt környezetre. Erre a célra Visual Studioban írtam C# nyelven egy egyszerű szimulációs eszközt, amivel csökkentett órajelen (maximum 500Hz) tudok kódokkal kísérletezni.ű

11.1 Órajel forrás kalkulátor

A szimulátor program lapokból épül fel, amik közül mindegyik egy-egy olyan funkciót lát el, amit a tesztelés során hasznosnak vagy fontosnak ítéltem. Az első fülön egy olyan egyszerű grafikus felület kapott helyet, ami legördülő menük segítségével teszi beállíthatóvá a rendszer órajelét. Ez a fül állítja be a szimulátor futási sebességét is.



33. ábra *Órajel kalkulátor*

A *33. ábrán* láthatóan, bal oldalon 4 darab szövegdoboz kapott helyet. Ezekbe a 4 különböző órajel forrás frekvenciáját kell beírni Hz értékkel. A mellettük található rádió gombok segítségével jelölhető ki, hogy melyik forrást használja a rendszer.

Az így létrejövő közös órajelből két darab osztó áramkört szimbolizáló legördülő menü segítségével lehet beállítani a kívánt működési frekvenciát külön a végrehajtó áramkörnek és külön a pwm jeleket előállító áramkörnek.

A beállított értékek alapján a program a 2. mellékletben szereplő kód segítségével létrehozza azt a bináris értéket, amit az órajel panel regiszterébe beírva a kívánt frekvenciájú működést kapja.

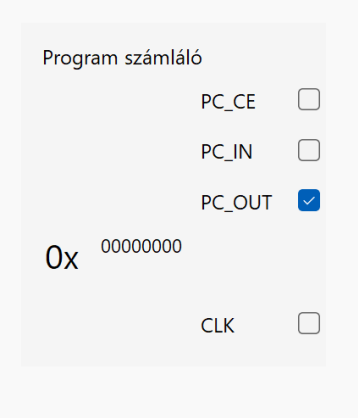
11.2 Szimulátor

A második fülön kapott helyet a teljes szimulátora a rendszernek. Itt az egyes építő modulok grafikusan elkülönített egységekben kaptak helyet, egyedi órajel bemenetekkel, amiket rádiógombok jelképeznek. A kódban hozzá rendeltem egy folyamatot ezekhez a bemenetekhez, amiken belül történik meg minden egyes modul bemeneti értékeinek a változtatása.

11.2.1 Program számláló

Az elsőként elkészült részegység, és az egyik legegyszerűbb is a *34. ábrán* látható program számlálót szimuláló modul. Ennek 4 darab bemenete van:

- PC_IN,
- PC_OUT,
- PC_CE,
- CLK.



34. ábra *Program számláló a szimulátorban*

Ezek a bemenetek rádió gombokkal vannak szimbolizálva. Az egység által tárolt értéket egy label tárolja, amibe beír, vagy amiből kiolvas amikor valamit módosításra van szükség. Ez azonban egy string formátumú adatot tárol.

Ahhoz, hogy ezzel számításokat lehessen végezni, át kell azt alakítani int típusra, amire az alábbi kódrészlettel van lehetőség:

```
Convert.ToInt(label.Text, 16);
```

A C# convert függvénye átalakítja a forrásban megjelölt string formátumú adatsort a kívánt formátumra. Amennyiben a szövegben nem 10-es számrendszerben található számokat kell értelmezni, a forrás megadása után vesszővel elválasztva kell megadni az eredeti számformátumot. Az én esetemben ez 16, mivel az egyszerűségért hexadecimális formátumban jelenítem meg az adatokat.

Miután a szükséges műveleteket elvégezte a program, a felhasználó fele értelmezhető, szövegként kell újra megjelenítenie az eredményt, amit az alábbi kóddal tud végrehajtani:

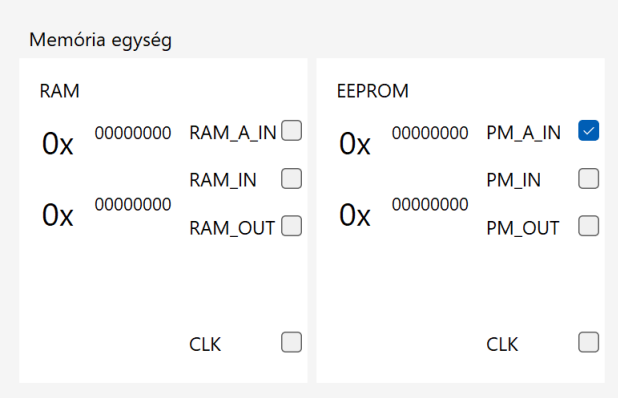
```
label.Text = valtozo.ToString("X8");
```

A számok átalakításáért a ToString függvény felel. Ez bármilyen változó típus értékét képes átalakítani egyszerűen értelmezhető szöveges adattá. Feltételként az “X8” azt jelenti, hogy a kimeneti szöveg 8 számjegyet jelenítsen meg, függetlenül a forrás értékétől.

A CLK in felfutásakor, ha a PC_CE bemenet is aktív állapotban van, a program számláló érteke eggyel növekszik, ha a PC_IN aktív akkor a fő adatbuszt reprezentáló label értéke bekerül a

program számlálóba, ezzel ugrást lehetővé téve, ha pedig a PC_OUT aktív, akkor mindaddig amíg az is marad, a számláló értéke kerül fel az adatbuszra. Amikor az erre utasító jel megszűnik, az adatbusz visszaáll 0 értékre, hacsak másik modul nem vezérli azt.

11.2.2 Memóriák



35. ábra *Memória egység szimulációs modellje*

A *35. ábrán* látható a memória egységet szimbolizáló grafikus modell. A RAM és az EEPROM külön órajel bemenetet kaptak, de ezek egyszerre lépnek működésbe.

A bemeneteik működése és a lezajló folyamatok a két egység esetében teljesen azonosak, így csak az egyiket részletezem, az ott megfogalmazott állítások érvényesek a másik egységre is.

A RAM és az EEPROM egységnek is három-három bemenete van, amik a RAM esetében a következők:

- RAM_A_IN
- RAM_IN
- RAM_OUT

A RAM_A_IN lábon, ha az engedélyező jel aktív, azaz a checkbox be van jelölve, akkor a CLK checkbox bejelölésére a fő adatbuszon lévő érték bekerül a RAM cím regiszterét szimbolizáló, felül található szövegmezőbe. Ezt a program átalakítja int formátumra, a program számlálónál már megismert módon, és egy tömb címeként felhasználja. A címet a következő kód segítségével tölti be és alakítja át megfelelő formátumra:

```
int RAMADR = Convert.ToInt32(BUS.Text, 16);
RAMADR = RAMADR & 0xFF;
RAM_ADR.Text = RAMADR.ToString("X8");
```

A fő adatbuszt jelképező BUS szövegmező tartalmát kiolvassa és egy köztes, RAMADR nevű, integer típusú változóba menti. Ezt ezután átalakítja az &0xFF művelettel, amivel kiszűri az alsó 8 bitjét. Erre azért van szükség, mert a szimulátor esetében csak az első 256 címen található adattal

dolgozik a rendszer, illetve ezzel szimulálva azt, hogy a memória címszélessége kisebb, mint az adatbusz szélessége, és esetleg a rendszer meghívhat olyan címet, ami kívül esik a lehetséges tartományon.

Ezután a RAMADR értékét szöveggé alakítja hexadecimális formátumban, és a cím regisztert jelképező szövegmezőbe írja azt.

Egy háttérben futó időzítő 1 millimásodpercenként behívja azt a függvényt, ami a tömb címéről az értéket kiolvassa, az ott található adatot átalakítja és betölti a modul alsó szövegmezőjébe, hexadecimális formátumban, az alábbi kód segítségével:

```
RAM_PIN.Text = RAM[Convert.ToInt32(RAM_ADR.Text, 16)].ToString("X8");
```

Ebben a RAM_PIN annak a szövegdoboznak az azonosítója, ami a RAM lábait, ezzel a kimenetét szimbolizálja, a Text függvénnyel ennek a tartalmára hivatkozik a kód. A RAM_ADR a címet tartalmazó szövegmezőnek a kódban található azonosítója, aminek a szövegét kiolvassa és számmá alakítja. A RAM[] egy tömb, amiben találhatóak az adatok, amit a RAM tárol.

A RAM_OUT bemenet aktív állapota esetén a RAM[] tömb, RAM_ADR szövegdobozban található szöveg által meghatározott címén található érték felírásra kerül a főbuszt jelképező BUS szövegdobozra. Innen a többi modul le tudja olvasni.

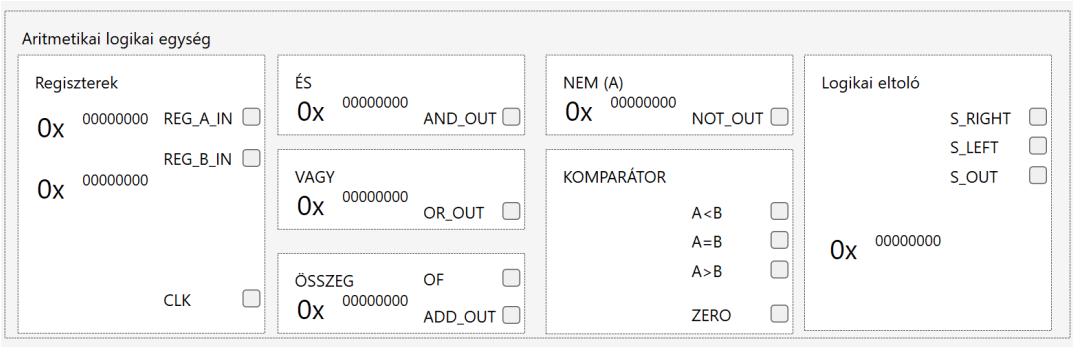
A RAM_IN bemenet aktív állapota esetén, ha a CLK bemenet aktív állapotba lépésekor a BUS szövegdobozban található adatot a lent leírt kódrészlet számmá alakítja, majd a RAM[] tömb RAM_ADR szövegdoboz által meghatározott címére beírja.

```
RAM[Convert.ToInt32(RAM_ADR.Text, 16)] = Convert.ToInt32(BUS.Text, 16);
```

Ezek a lépések teljes egészükben megegyeznek a program memória esetében is, csak a szövegdobozok és változók nevei térnek el.

11.2.3 Aritmetikai és logikai egység

Az aritmetikai és logikai egység szimulációs modellje látható a *36. ábrán*, aminek felépítése megegyezik az elkészült eszközzel. Két darab 32 bites regiszter található rajta, amiben tárolja azokat a számokat, amikkel a műveleteket elvégzi. A műveleteket a szimulátorban is több alegység végzi el, amik ugyanazon két regiszter értékeit olvassák ki.



36. ábra *Az Aritmetikai és logikai egység szimulációs modellje*

Az egyes moduloknak csak kimenet engedélyező lábuk van, ami a program számláló és a memória modulhoz hasonlóan az eredményt tartalmazó szövegdoboz értékét a fő adatbuszt szimbolizáló BUS szövegdoboz értékébe írja be.

A modulok az eredményeket automatikusan hozzák létre, függetlenül a fő órajeltől.

A regiszterek szimbolizáló két darab szövegdoboz tartalmát számmá alakítják, ezután elvégzik a megfelelő műveletet, majd ezt hexadecimális formában ismét szöveggé alakítják, és a kimenetüket jelképező szövegdobozba írják.

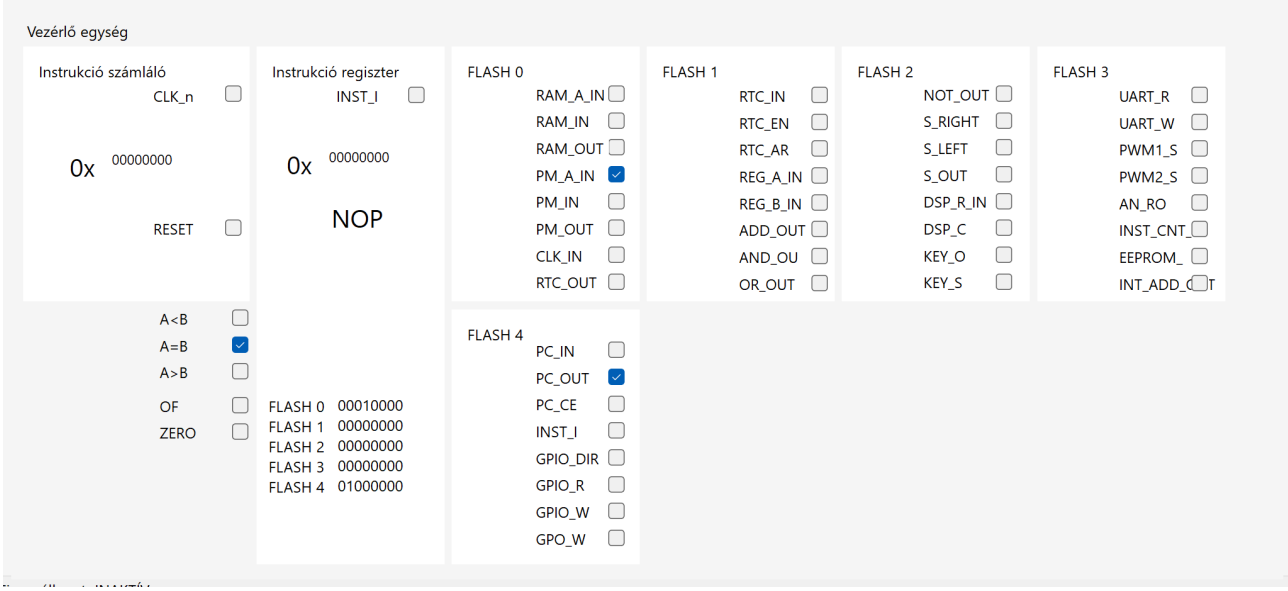
A komparátor egység kimenetei nem az adatbuszra kerülnek fel. Ezek a kimenetek egyenesen a vezérlő egységbe kerülnek, ahol, mint módosítójelek funkcionálnak. Ezen kívül még az összeadó egységnek van egy kimenete közvetlenül a vezérlőre kötve, szintén, mint módosítójel.

A komparátor és a logikai eltolást végző modul külön kártyára kerültek fel a kész terven, viszont azok regiszterei szinkronban működnek az ALU regisztereivel, így nem kezelem őket külön elemként a szimulátorban.

11.2.4 Vezérlő egység

Mint ahogy az elkészült, megépített számítógépen, a szimulátorban is szükség van egy olyan egységre, ami koordinálja a többi elem működését, és megteremti a szinkront. A vezérlő egység a *37. ábrán* látható, az azt alkotó külön számláló egységgel, instrukció regiszterrel, valamint az utasító jeleket tartalmazó FLASH memóriákkal.

Az egész szimulátornak ez a legnagyobb és egyben legbonyolultabb eleme, az összes kiszolgáló egység vezérlő jele helyet kapott rajta. A szimulátor és a valós elektronika egyszerű összehasonlíthatósága érdekében, az itt található kimenetek ugyanabban a sorrendben és csoportosításban vannak, mint az elkészült terveken.



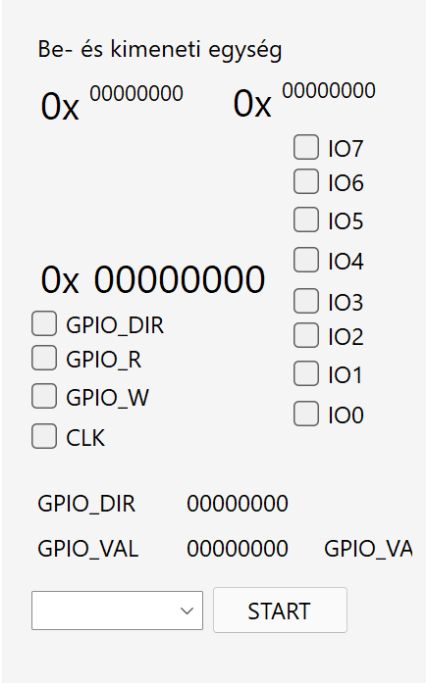
37. ábra *Vezérlő egység szimulációs modellje*

A vezérlő egység egy fél órajellel eltolva, előre dolgozik a rendszer többi eleméhez képest. Erre azért van szükség, hogy egy kimenet már teljesen stabil állapotba állhasson be mikorra egy bemenetnek azt olvasnia kellene.

Az instrukció számláló, az utasítás lépésének a megadására szolgál, folyamatosan aktív, az inverz órajel minden felfutó élére eggyel növekszik az értéke. Ez az egység kapott egy RESET bemenetet is, ami arra szolgál, hogy utasításból is nullázni lehessen ezt a számlálót, így pedig ehetővé téve a változó lefutási idejű utasításokat.

A módosító jelek és az instrukció számláló értékei együttesen létrehoznak egy címet, amire hivatkozik a program. Az utasítás nevét (a képen a NOP azaz nincs utasítás látható) egy külső fájlból olvassa ki, az instrukció regiszter értéke az adott utasítás szövegfájlban lévő sorszámát adja meg. A kiolvasott névvel ellátott txt formátumú fájl tartalmazza azt a bináris kódot, ami a kész rendszerbe is feltöltésre kerül, az ebben kiolvasandó sor számát adja meg a korábban létrehozott hivatkozási cím. Az így kiolvasott szövegsor tagolva van tabulátorokkal, aszerint, hogy hányas sorszámú flash memória kimenetének felel meg.

11.2.5 Be és kimenetek



38. ábra A Be- és kimeneti egység szimulációs modellje

Ahhoz, hogy magasabb funkcionalitású programokat is tesztelhessek a szimulátoromban, szükségem volt egy olyan egységre is, ami képes a valódi eszközhöz hasonlóan a külvilággal és más eszközökkel kommunikálni, így felhasználónak visszajelezni, és attól információkat fogadni.

A *38. ábrán* látható be- és kimeneti egység a többi, egyszerű regiszteralapú szimulátorhoz képest abban különbözik, hogy itt nem minden esetben kell a kimeneten annak az értéknek megjelennie, ami a regiszterében van, hiszen egyes lábak bemenetként vannak konfigurálva. Ezeket a bemeneteket azonban olvasnia kell tudni a rendszernek.

A kívánt teljes kimenet tartalmát a bal felső szövegdoboz jeleníti meg hexadecimális formátumban. Az alatta lévő, nagyobb karakterekkel kiírt hexadecimális kód adja meg, hogy melyik lábak milyen funkciót látnak el. Ahhoz, hogy a tesztelés során a kódban található hibákat javíthassam, ennek a két regiszternek az alsó 8 bitjét külön bináris formában is megjelenítem, hogy az esetleges hibákat kiszűrhessem.

A jelölő dobozok, ha kimenetként vannak konfigurálva nem módosíthatóak a felhasználó által, ha azonban a hozzájuk tartozó láb bemenetként van beállítva, akkor lehetséges a kijelölés változtatása, ezzel a bemeneti érték változásának a szimulálása.

Mivel a rendszernek külső eszközökkel, mint például egy kijelzővel is tudnia kell kommunikálni, meg kellett oldanom, hogy a szimulátor is képes legyen ezt végrehajtani. Az első megközelítésem az volt, hogy a szimulátorba beépítem a kijelző másolatát. Ez habár nem lett volna lehetetlen, a rendelkezésre álló idő, és a feladat kis fontossága miatt inkább úgy döntöttem, egy

Arduino Uno fejlesztői kártya segítségével a szimulátor kimeneteit használhatóvá teszem.

Az egység alján található legördülő menüből ki lehet választani, hogy melyik soros csatlakozón található az Arduino, ha az csatlakoztatva van a számítógéphez. Ezután a start gomb lenyomása után, minden órajel ciklusra kiküldi a kimeneteket szimbolizáló bináris számsort a szimulátor, amit az Arduino-n lévő szoftver dekódol és jelenít meg. Ezzel a megoldással lehetővé vált egy valódi kijelzővel kísérletezni a szimulátor segítségével, így pedig az esetleges hibákat elkerülni. Az Arduino Uno-n mindössze 13 darab kimeneti csatorna szimulálására van lehetőség maximálisan, én ebből 10-et használok fel. Ahhoz, hogy a teljes 32 csatornás rendszert le tudja szimulálni egy Arduino Mega vagy egy egyéb (Nucleo, Raspberry, Teensy) fejlesztői kártyára lenne szükségem.

11.3 Utasításkészlet generátor

A Windows rendszerre elkészült szoftver harmadik füle egy olyan eszközt tartalmaz, aminek a segítségével nem szükséges az utasításokat minden egyes bemeneti kombinációra egyesével megírnom, azt automatizáltan létrehozza nekem. Ez az eszköz fel van készítve arra is, ha egy utasításnak csak egy bizonyos módosítójel kombináció esetén kell lefutnia, mint például a feltételes ugrások. A program ezen ablakában ki lehet választani azt a fájlt, amiből az utasításkészlet összes utasítását ki tudja olvasni nevek szerint. Ezután ebből a fájlból létrehoz egy adatsort, amit a legördülő menühöz rendel. Ebben kiválasztva azt az utasítást, aminek a teljes, 512 soros változatát el szeretném készíteni a Teszt gombot lenyomva elkészíti az összes vezérlő egység flash-hez tartozó bináris adattáblát. Ezeket a megfelelő sorrendben egymás után beleégetve a flash-ekbe a valódi eszközön is ugyanazt a működést kell kapjam, mint a szimulátorban.

12. Utasításkészlet

Egy számítógép képességeit nagyban befolyásolja az utasításkészlete. Napjainkban két féle rendszert különböztetünk meg:

- RISC – Csökkentett utasításszámú számítógép
- CISC – Összetett utasításkészletű számítógép

12.1 CISC

Az összetett utasításkészletű számítógépek több, bonyolult, akár több lépésből álló utasítást tartalmaznak. Ezeknek a rendszereknek a gépi kódjai általában az ember számára könnyebben olvashatóbbak, rövidebbek, viszont futásuk sok esetben lassabb.

12.2 RISC

A csökkentett utasításkészletű számítógépek az 1980-as években jöttek létre, az addigra már nagyon bonyolultra duzzadt utasításkészletű processzorok egyszerűsítésére. Néhány fontos jellemzőjük:

- nem tartalmaz olyan utasítást, ami egyszerre használná a memóriát és az aritmetikát
- alacsony számú, egyszerű címzési módok,
- az utasítások hardveresen vannak megvalósítva, nem pedig mikrokóddal,
- az utasítások azonos hosszúságúak,
- lehetőség szerint az utasítás 1 órajel ciklus alatt fut le
- futószalagos végrehajtás (pipeline technológia)
- nagy számú általános célú regiszter

12.3 Saját rendszer

A fent leírtak alapján a saját rendszeremet a CISC architektúrák közé sorolnám. Az általam implementált utasításkészlet változó hosszúságú, több órajel ciklus alatt fut le egy utasítás. Az olyan technológiák, ami gyorsabbá tennék, mint például a pipeline hiányoznak.

Az utasításkészlet kialakításakor igyekeztem minden olyan folyamatot egy utasításba foglalni, amik többször előfordulhatnak a működés során és ezzel a gépi kódot egyszerűsíteni, ember által könnyen olvashatóvá tenni, ezzel még inkább fordulva a CISC rendszerek felé.

Mivel a memória típus, amit alkalmazok, első alkalommal teljesen üres, 0x00000000 értékekkel van feltöltve, így ahhoz, hogy ne történhessen egy üres rendszer esetén nem várt működés, esetleg két kimenet összenyitása és ezzel a rendszer károsítása, erre az kódra a NOP, nincs utasítás parancsot helyeztem el. Emiatt, ha a rendszerbe teljesen kiürítem a program memóriát, akkor csak lépkedni fog előre a program számlálóval mindaddig amíg az túl nem csordul.

Ezzel az utasítással lehetséges egyfajta késleltetés funkciót is megvalósítani, mert nem csak egy órajel a lefutása, összesen 10 órajelből áll.

Pontos időzítésekhez legjobb megoldás egy olyan időzítő beépítése, ami képes a rendszert a megadott ideig teljes mértékben megállítani, vagy pedig olyan megszakító rendszert kialakítani,

amivel a hosszabb várakozások alatt más folyamatokat is el lehet végeztetni.

Az utasításkészlet kialakításánál fontos szempont volt még, hogy egyes matematikai műveletek egy szóval végrehajthatóak legyenek, ne kelljen például egy kivonás műveletet a programkódban több utasításból összeállítani, emiatt viszont az említett utasítás kialakítása bonyolultabb lett, a hibakeresés a fejlesztés során pedig lelassult.

A dolgozat írásakor az utasításkészlet az alábbi utasításokat tartalmazza:

2. táblázat *A rendszer utasításkészlete*

Név	**Módosítók**	**Mt.**	**Srsz.**	**Leírás**
NOP	XXXXX	1	0	nincs utasítás
JAL	XXXX1	2	1	ugrás, ha A kisebb
JAE	XXX1X	2	2	ugrás, ha A és B egyenlő
JAG	XX1XX	2	3	ugrás, ha A nagyobb
JOF	X1XXX	2	4	ugrás, ha túlcsordul
JZE	1XXXX	2	5	ugrás, ha 0
JMP	XXXXX	2	6	ugrás
APB	XXXXX	4	7	A és B összead (érték a kódban) és ment RAM címre
AAB	XXXXX	4	8	A és B & (érték a kódban) és ment RAM címre
AOB	XXXXX	4	9	A és B | (érték a kódban) és ment RAM címre
ANR	XXXXX	3	10	!A (érték a kódban) és ment RAM címre
ARB	XXXXX	2	11	Összeg eredményét menti RAM címre
RPB	XXXXX	4	12	RAM cím és B összead és ment RAM címre
AMB	XXXXX	4	13	A-ból B kivon és ment RAM címre
RMB	XXXXX	4	14	RAM címből B kivon és ment RAM címbe
FTR	XXXXX	3	15	FLASH címen lévő adatot másol RAM címen lévő helyre
RTF	XXXXX	3	16	RAM címen lévő adatot másol FLASH címen lévő helyre
VTR	XXXXX	3	17	RAM címre étéket ír be
VTF	XXXXX	3	18	FLASH címre értéket ír be
~~*DSD*~~	~~*XXXXX*~~	~~*2*~~	~~*19*~~	~~*Kijelzőre ír ki egy adatot*~~
~~*DSC*~~	~~*XXXXX*~~	~~*2*~~	~~*20*~~	~~*Kijelzőre küld ki egy parancsot*~~
RRT	XXXXX	3	21	RTC kiolvas egy regisztert és RAM címre menti

RTW	XXXXX	3	22	RTC beír egy RAM címről adatot egy regiszterbe
RAB	XXXXX	3	23	A és B regiszterbe beír értéket ramból
PAB	XXXXX	3	24	A és B regiszterbe beír értéket utasításból
GPD	XXXXX	2	25	GPIO irány beállítás utasításból
GWI	XXXXX	2	26	GPIO-ra ír utasításból
GWR	XXXXX	2	27	GPIO-ra ír ram címen lévő érétket
GRR	XXXXX	2	28	GPIO olvas RAM címre
GPS	XXXXX	2	29	GPIO egy adott lábát állítja 1-be
GPR	XXXXX	2	30	GPIO egy adott lábát állítja 0-ba
RTR	XXXXX	2	31	Ram címről ír egy adatot másik RAM címre
AND	XXXXX	2	32	AND értékét menti RAM címre
ORR	XXXXX	2	33	OR értékét menti RAM címre
NOT	XXXXX	2	34	NOT értékét menti RAM címre
SRR	XXXXX	2	35	A regiszter értéket léptet jobbra és RAM címre menti
SLR	XXXXX	2	36	A regiszter értéket léptet balra és RAM címre menti
VRA	XXXXX	2	37	Érétket ír a RAM címről az A regiszterbe
VRB	XXXXX	2	38	Értéket ír a RAM címről a B regiszterbe
ADD	XXXXX	2	39	Összeg mozgat RAM címre

12.3.1 Logikai műveletek változókkal

A különböző csoportokba tartozó utasítások nem egymás után kerültek be a *2. táblázatba*. Ennek az oka, hogy a fejlesztés során megvoltak bizonyos utasítások, amiket mindenképpen implementálni akartam a rendszerbe, de sok olyan művelet volt, amik csak a szimulációs tesztek során váltak fontossá, így azok a sor végére kerültek be, hogy az addigi mikrokódok is működőképesek maradjanak.

A különböző változókkal és utasításban tárolt értékekkel a következő utasítások végeznek logikai műveleteket:

- AAB – A és B logikai „és” műveletét mozgatja egy megadott ram címre,
- AOB – A és B logikai „vagy” műveletét mozgatja egy megadott ram címre,

- ANR – A értékének a logika inverzét mozgatja egy meghatározott ram címre.

Ezekben az utasításokban közös, hogy az A és B regiszterbe beírandó értéket az utasító szó utáni memóriaterületeken az utasítás tartalmazza. Ezek a számok statikusak, nem változnak, ha egy ciklikus működés során a program újra ide fut.

A működés során használt változók logikai módosítása kettő vagy három utasítás végrehajtásával történhet meg. Ilyenkor szükség van a változók regiszterekbe írására, majd a logikai művelet eredményének egy újabb, vagy valamelyik forrással azonos memóriaterületére való mentésére. Ezek az utasítások a következők:

- AND – logikai „és”,
- OR – logikai „vagy”,
- NOT – logikai „nem”.

Ezeket az utasításokat mindig meg kell előzze egy vagy két regiszter feltöltő utasítás:

- RAB – két ram címről bemásolja az értékeket a regiszterekbe,
- PAB – A és B regiszterbe beírja az utasítás szerinti értéket
- VRA – ram címen lévő érétket mozgat az A regiszterbe,
- VRB – ram címen lévő érétket mozgat a B regiszterbe,

Ezekkel az utasításokkal a ram egyes memória területein lévő értékeket lehet a regiszterekbe mozgatni.

Ezek az utasítások azért szerepelnek külön, és nem lettek egybe gyúrva a logikai műveletekkel, mert a logikai eltolás és a hasonlítás műveleténél szükséges lehet a használatuk.

Az említett műveleteket a következő utasításokkal lehet végrehajtani:

- SRR – az A regiszterben tárolt érétket lépteti jobbra és mozgatja a megadott ram címre,
- SLR – az A regiszterben tárolt értéket lépteti balra és mozgatja a megadott ram címre.

Az összehasonlítás nem kapott saját utasítást, mivel arra csak az ugrásoknál van szükség, amit egy későbbi bekezdésben tárgyalok.

12.3.2 Aritmetikai műveletek

Ezeknek az utasításoknak a képességei nagyban befolyásolják a teljes rendszer sebességét és tudását. Minél gyorsabban képes a rendszer egy aritmetikai műveletet végrehajtani, annál gyorsabb program futás érthető el.

Az aritmetikai műveleteket a következő utasítások hajtják végre:

- APB – A és B regiszterbe utasításból beírt számokat összeadja és mozgatja a ram egy címére,

- ARB – A és B regisztereket összeadja és ram címre mozgatja,
- RPB – RAM címen lévő értéket ír A regiszterbe, utasításból feltölti B regisztert, az eredményt a korábbi RAM címre mozgatja,
- AMB – A és B regisztert feltölti utasításból, különbségüket mozgatja ram címre
- RMB – RAM címen lévő érték és utasításban található érték különbségét mozgatja a korábbi RAM címre,

Ezek közül a műveletek közül a legbonyolultabb az „RMB" művelet. Ez az utasítás összesen 4 memóriaterületet foglal el és 13 órajelből épül fel. Az utasítás végrehajtása során a korábban tárgyalt bináris számok kivonási művelet sora fut le. Először veszi a kivonandó számot, negálja és hozzá ad egyet. Ezután azt a B regiszterbe tölti, majd az A regiszterbe betölti a kisebbítendő számot, az összegüket pedig menti egy megadott ram címre. Egy ilyen utasítás felépítése a következő:

RMB kivonandó 1 kisebbítendő címe.

Ebben a kivonandó szám egy utasításban tárolt érték, a kisebbítendő címe a ramban található változóra hivatkozik, az 1-es szám pedig a kivonandó szám negálás utáni 1-gyel történő növeléséhez szükséges.

A bonyolultabb aritmetikai műveleteket a rendszer ebből a két alapműveletből alakítja ki ciklusok segítségével.

12.3.3 Ugrások

Ahhoz, hogy egy műveletet többször egymás után végre lehessen hajtani, a teljes programot újra kezdeni, vagy akár kihagyni részeket, a rendszernek képesnek kell lennie ugrást végrehajtani a kódban. Kétféle utasítás műveletet különböztethetünk meg:

- feltétel nélküli ugrás,
- feltételes ugrás.

A feltétel nélküli ugrás a sima JMP utasítás, ami a program számláló értékét egy az utasításban meghatározott értékre állítja. Ez az utasítás minden alkalommal az eredeti formájában lefut, amikor meghívásra kerül. Ettől eltérő módon működnek a feltételes ugrások:

- JAL – akkor ugrik a megadott címre, ha az A regiszter értéke kisebb, mint a B regiszteré,
- JAE – akkor ugrik a megadott címre, ha az A regiszter és B regiszter érétke egyenlő,
- JAG – akkor ugrik a megadott címre, ha az A regiszter értéke nagyobb, mint a B regiszteré,
- JOF – akkor hajtja végre az ugrást, ha az összeadó modul túlcsordul,
- JZE – akkor hajtja végre az ugrást, ha az A regiszter értéke 0.

Ezek az utasítások a leggyakrabban használtak közé tartoznak, mivel ezekkel lehetséges a

bonyolultabb, több bemenettel és kimenettel dolgozó programok létrehozása, valamint a legtöbb magas szintű nyelvben megtalálható for, while és if függvények végrehajtása.

A feltételes ugrások feltételei az utasítás memória címbemenetein jelennek meg. Mivel a legtöbb utasítás esetében ezeknek a jeleknek semmilyen befolyásolása nem lehet a lefutásra, azokat a táblázat 2. oszlopában látható módon X-szel jelölve, figyelmen kívül kell hagyni. Csak azoknál az kombinációknál szükséges a feltételes ugrás lefutása, ahol a táblázat azt 1-gyel jelöli.

Az utasítás memória bemeneti címeinek felépítése a következő:

MEGSZAKÍTÓK | MÓDOSÍTÓ JELEK | UTASÍTÓ SZÓ | UTASÍTÁS LÉPÉS

Az utasítás lépések értékét a lépés számláló adja meg a vezérlő kártyán. Ez egy 4 bites változó.

Az utasító szót az utasítás regiszter tárolja, ez a szám a táblázat 4. oszlopában található érték.

A módosító jelek, amik a komparátorról és az összeadóról érkeznek, ez egy 5 bites bemenet.

Az megszakítás bemenetek nincsenek használva, memória típustól függően, itt lehetnek érvénytelen címlábak is, a későbbiekben még felmerülhet a használatuk.

Amennyiben az adott feltételes ugrásnak nem megfelelő bemeneti feltétel kód található a módosító jelek bemenetén egy módosított NOP utasítás fut le, ami kétszer növeli a program számlálót, hogy az ugrás címet is átlépje, ezzel elkerülve, hogy azt egy utasító szóként kezelje a rendszer.

A feltételeknek már az utasítás meghívás előtt megfelelőnek kell lenniük, különben az nem kerül végrehajtásra.

12.3.4 Memória műveletek

A különböző memóriaterületek feltöltésére és az azok közti adatmozgatásra is képesnek kell lennie a rendszernek. A memória területek közötti adatmozgatás egy változó létrehozásakor, vagy egy ram-ban tárolt adat végleges mentésekor hasznos, a flash memória kívülről érkező adattal történő írhatósága pedig akkor, ha képes rendszer programkódok betöltésére külső eszközökről.

A memóriaterületek módosítására szolgáló utasítások a következők:

- FTR – a flash megadott címén lévő adatot mozgatja a ram megadott címére,
- RTF – a ram megadott címén lévő adatot mozgatja a flash megadott címére,
- VTR – a ram megadott címére írja az utasításban szereplő értéket,
- VTF – a flash megadott címére írja az utasításban szereplő értéket,
- RTR – a ram egyik címéről mozgatja az adatot a ram másik címére.

A VTR parancs az egyik leggyakrabban használt ezek közül, minden változó létrehozásakor, annak kezdeti értéket adva ez a parancs kerül meghívásra.

12.3.5 GPIO műveletek

A külvilággal való kapcsolattartásra a rendszer eredetileg több módon is képes lett volna, ezek:

- GPIO,
- kijelző,
- billentyűzet.

A fejlesztés során azonban az utóbbi kettő lehetőséget kivettem a tervek közül, és helyette a GPIO kapacitásait növeltem meg, hogy azok miden célt el tudjanak látni.

Ezzel a kész hardver bonyolultságát is tudtam csökkenteni, és a felszabadult vezérlő lábakkal lehetőséget kaptam egy analóg bemenet kialakítására is, amiről később ejtek szót.

A GPIO műveletekhez tartozó utasítások a következők:

- GPD – beállítja, hogy a lábak ki vagy bemenetek,
- GWI – az utasításban lévő értéket kiírja a GPIO regiszterébe,
- GWR – a ram egy címén lévő értéket kiírja a GPIO regiszterébe,
- GRR – a GPIO regisztert kiolvassa és a ram egy megadott címére mozgatja,
- GPS – egy megadott lábat állít magas kimeneti szintre,
- GPR – egy megadott lábat állít alacsony kimeneti szintre.

A fentebb leírt utasításokkal a ki és bemeneteket teljes egészükben le lehet kezelni, valamint vissza ellenőrizni a kimenetek állapotát. Az egyéb beállítási lehetőségekhez, mint például a felhúzás vagy a belső jelek kivezetése egy lábra a dolgozat írása során még nem értem el, a jövőbeli módosítások közé tartoznak. A programozás megkönnyítése érdekében készült el az a két utasítás, ami a lábakat egyesével képes módosítani. Ehhez a művelethez a logikai és illetve a logikai vagy művelete segít.

Egy láb magas állapotba helyezéséért felelős utasítás a következő lépéseket hajtja végre:

1. kiolvassa a GPIO regiszter adatait (kiolvassa a GPIO-kat a RAM-ba),
2. az utasításban találhat, 2 hatványával egyenlő szám határozza meg a módosítandó lábat,

   de itt szerepelhet más szám is, ekkor az összes ahhoz tartozó GPIO magas szintbe fog állni,

   a kiolvasott GPIO regiszter érték és az utasításban található szám és művelete létrehozza az új kimeneti kódot
3. az és művelet eredményét kiírja a GPIO regiszterbe.

A láb nulla helyzetbe állítása már egy ennél bonyolultabb feladat, itt a negálás művelete is szerepet játszik. Ekkor a kiolvasott értéket és az utasításban lévő érték negáltját egyesíti a logikai és műveletével, majd mozgatja a kapott értéket a GPIO regiszterbe.

12.3.6 Valós idejű óra

Az eszköz alap panelján, ahol a táp és az órajel generálás is található, helyet kapott még egy valós idejű óra integrált áramkör is, aminek megvannak a saját dedikált utasításai:

- RRT - kiolvassa az RTC egyik regiszterét és egy megadott ram címre menti,
- RTW – a ram egy magadott címéről kiolvas egy éréket és az RTC egy megadott regiszterébe írja azt.

Ugyan ez is megvalósítható lett volna a GPIO interfészen keresztül, mivel az alsó panel készült el elsőként, és ekkor még voltak eltérések a koncepcióban, így ez az eredeti szerint megmaradt.

12.4 Utasítás végrehajtás

Az utasítások végrehajtása a vezérlő egység legfontosabb feladata. A végrehajtás során az utasítás kódja, a módosító jelek és a lépés számláló által megadott kombináció szerint állítja be a vezérlő lábakat, amikből miden modulhoz tartozik legalább egy. Az utasítás lépésének végrehajtása során nagyon fontos, hogy egyszerre ne aktiválódjon két kimenet, mert az ütközést, adatromlást és esetleges hardveres meghibásodást is okozhat. Erre a gépi kód készítésekor fokozottan ügyelnem kellett. Egy utasítás végrehajtásakor az első lépés annak a betöltése. Minden utasítás első két parancsa teljesen azonos:

1. Programszámláló értékének mozgatása a program memória cím tárolójába
2. Program memória adat mozgatása utasítás regiszterbe, közben programszámláló növelése

Amikor ez a két sor lefut, valójában még az egyel korábbi utasítás folyamata fut, csak a második lépés során lép át a rendszer az aktuális utasításra. Innentől az utasítások nagy része teljesen eltér egymástól, ez alól egyedüli kivétel az ugrás utasítások csoportja. Ezek, ha a bemeneti feltétel megfelelő teljesen egyformák. A kivonás művelete egy bonyolult, sok lépésből álló folyamat, aminek az elkészítésekor fokozottan ügyelnem kellett a memória mozgatások helyességére és a műveletek sorrendjeire, ami miatt az utasítás parancsa bonyolultabb lett a többinél és kevésbé következetes.

A kivonás lépései:

3. program számláló kiolvasása, mozgatása a program memória címtárolójába,
4. program memória kimenet mozgatás A regiszterbe, közben programszámláló növelése,
5. A regiszter negált kimenet mozgatás B regiszterbe,
6. program számláló kiolvasása, mozgatás a program memória címtárolójába,
7. program memória kimenet mozgatása A regiszterbe, közben program számláló növelése,
8. A és B regiszter összegének mozgatása B regiszterbe,
9. program számláló kiolvasása, mozgatása a program memória címtárolójába,

10. program memória kimenet mozgatása ram címtárolójába, közben programszámláló növelése,
11. ram kimenet mozgatása A regiszterbe,
12. A és B regiszter összegének mozgatása ram tárolóba,
13. lépés számláló nullázása.

Ahogy azt a korábbiakban is írtam, az első két lépés minden utasítás esetében azonos, ezért azokat ebből a felsorolásból ki is hagytam.

A harmadik lépés során, a program memóriában, az utasítás részeként, az utasítás második memóriacímén található kivonandó érték címét adja át a program számláló a program memória cím tárolójának.

A negyedik lépés során az ezen a címen található értéket a rendszer az A regiszterbe mozgatja, mindeközben már növeli a program számláló értékét, hogy a következő címen lévő adat is elérhetővé váljon.

Az ötödik lépés során az A regiszterben található szám inverzét mozgatja a rendszer a B regiszterbe, ezzel a bináris kivonás első tényleges lépését végrehajtja.

A hatodik lépés során ismét a program számláló értékét mozgatja a program memória címtárolójába. Ekkor annak, a műveletek végrehajtásához szükséges 1-es számnak a helyét adja meg, ami a kivonandó módosításához szükséges.

A hetedik lépésben a program memória adott címén található érétket, jelen esetben 1-et, mozgatja a B regiszterbe, mindeközben a program számláló értékét növeli.

A nyolcadik lépés során a két szám összegét mozgatja a rendszer a B regiszterbe.

A kilencedik lépésben ismét a program számláló értéke kerül átmozgatásra a program memória címtárolójába.

A tizedik lépésben a korábban meghatározott címen található értéket a program memóriából a ram cím tárolójába mozgatja a rendszer, közben növeli a program számlálót. Mivel az utasítás több memóriaterületet nem foglal a program memóriában, ez az új cím már a következő utasítás nevét tartalmazó helyre mutat.

A tizenegyedik lépés során a ram korábban meghatározott címén található értéket mozgatja a rendszer az A regiszterbe. Ez a szám a kisebbítendő szám.

A tizenkettedik lépés során az A regiszter értékének, ami a kisebbítendő, és a B regiszter étékének, ami a kivonandó az összegét mozgatja a ram korábbi lépésekben megadott címére.

A tizenharmadik lépés során a vezérlő egység nullázza a lépés számlálót, ezzel átlépve a következő utasításra. Ez a lépés nem feltétlenül szükséges, a lépés számláló a 16. lépés után automatikusan újra kezdi a számolást, viszont ezzel gyorsítható a program futása.

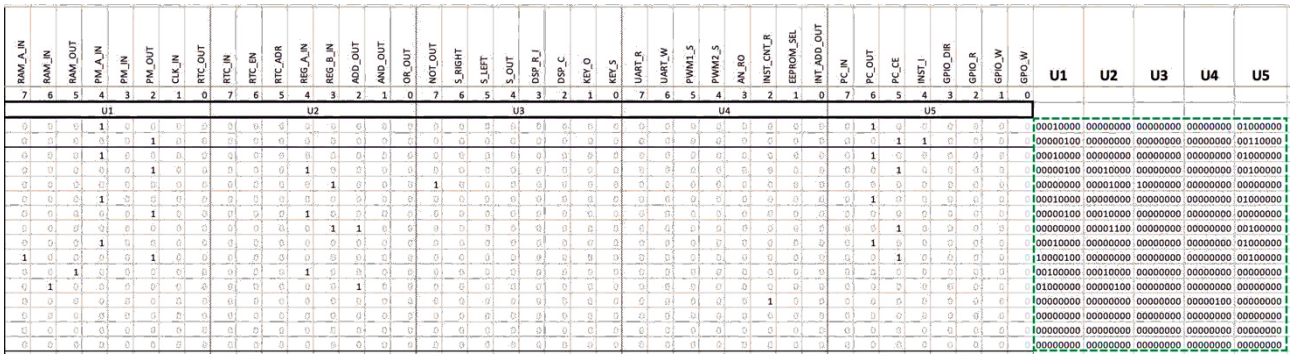
12.5 Utasításkészlet szerkesztése

A rendszer utasításai bonyolultak, és nagyon sok vezérlő jellel kell egyszerre foglalkozni. A korábban említett RMB kivonás utasítás bináris kódja így épül fel:

00010000	00000000	00000000	00000000	01000000
00000100	00000000	00000000	00000000	00110000
00010000	00000000	00000000	00000000	01000000
00000100	00010000	00000000	00000000	00100000
00000000	00001000	10000000	00000000	00000000
00010000	00000000	00000000	00000000	01000000
00000100	00010000	00000000	00000000	00000000
00000000	00001100	00000000	00000000	00100000
00010000	00000000	00000000	00000000	01000000
10000100	00000000	00000000	00000000	00100000
00100000	00010000	00000000	00000000	00000000
01000000	00000100	00000000	00000000	00000000
00000000	00000000	00000000	00000100	00000000
00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000

Minden egyes érték egy sorban megfelel egy vezérlő jelnek és minden 8 értékből álló oszlop megfelel egy hozzá tartozó flash memória kimenetének. Egy ilyen bináris táblázatot szerkeszteni szöveges formában túl nehéz és körülményes lenne, ahhoz, hogy eredményt lehessen vele elérni.

Ehelyett egy, a *39. ábrán* látható Excel táblázatot készítettem, ami tartalmazza az összes vezérlő jel nevét, valamint függvényeket arra, hogy összegyúrja az oszlopokat, és ezt az 5x8 bites sorokból álló adathalmazt kiadja.



39. ábra *Utasítás szerkesztő Excel táblázat*

Ebben a táblázatban, ha egy oszlopba 0 számot írok, azaz a kimenetet abban a pillanatban alacsony szinten szeretném tartani, az értéket halvány szürke színnel írja ki, míg, ha 1-et írok bele, tehát aktiválni szeretném azt a vezérlő jelet, akkor feketére színezi. Ez az átláthatóságot növeli jelentősen. Abban az esetben, ha egy sorban egyszerre több kimenetet is megjelölök, akkor egy hibát ír ki a függvény. Ezzel már egyszerűen létre lehet hozni egyes utasításkat, amiket a korábban leírt Windows programmal lehet a teljes bemeneti kombinációra átalakítani majd összefűzni és feltölteni az utasítás memóriába.

12.6 Utasításkészlet feltöltése

A szimulátor használata közben az utasításkészlet módosítása és tesztelése egyszerű feladat, mivel szöveges formátumban tárolja a gépi kódot, viszont a megvalósult eszközön szükség van arra, hogy valamivel fel legyenek égetve a kódok.

Erre a feladatra egy Arduino Mega fejlesztői panelt használtam fel, valamint egy hozzá elkészített áramköri lapot, amivel egyszerre lehet csatlakoztatni az öt darab programozandó flash memóriát. Ehhez a kapacitáshoz van szükség a Mega lábmennyiségére.

Ugyanezzel az eszközzel lehet a program memóriába beleírni a lefuttatni kívánt programot is.

12.7 Egyszerű fordító

A 11.5 pontban is említett kivonás utasítás szerkesztése is kifejezetten bonyolult az összetett szerkezete miatt, de az alkalmazása sem egyszerűbb. Az első program, amit a rendszerre írtam egy egyszerű ciklus, ami folyamatosan növel egy számot, majd visszalép a kiindulási állapotra. Ennek egy továbbfejlesztett változata az, ami már a kivonást végzi el, ezzel tesztelve a funkciót.

Ezek a programok, az első néhány tesztelés során még ilyen formában lettek elkészítve:

```
17      Érték beírása ram címre
0       ram cím
0       érték

12      Ram cím és érték összeadása, mozgatás ram címre
0       forrás ram cím
1       érték
0       cél ram cím

6       Ugrás
2       ugrás címe
```

Ez a jelenlegi tördeléssel még egészen átlátható is, de hosszabb kódok esetén már kifejezetten bonyolultnak számít, ezért szükséges volt egy, az ember által könnyebben átlátható szerkezetű programkód létrehozására.

Ebben a leírásban a korábbi kód a következő alakot veszi fel:

```
VTR   0     0
RPB   0     1     0
JMP   2
```

Ahhoz, hogy ebben a formában képes legyek programokat szerkeszteni, egy olyan kezdetleges fordítóprogramot kellett készítenem, ami képes átalakítani a második formátumot az elsőre hiba nélkül. Ezen felül kényelmi okokból a változók kezelését is egyszerűbbé akartam tenni, lehetővé téve a definíciókat, és így a változót nem cím, hanem név alapján kezelni.

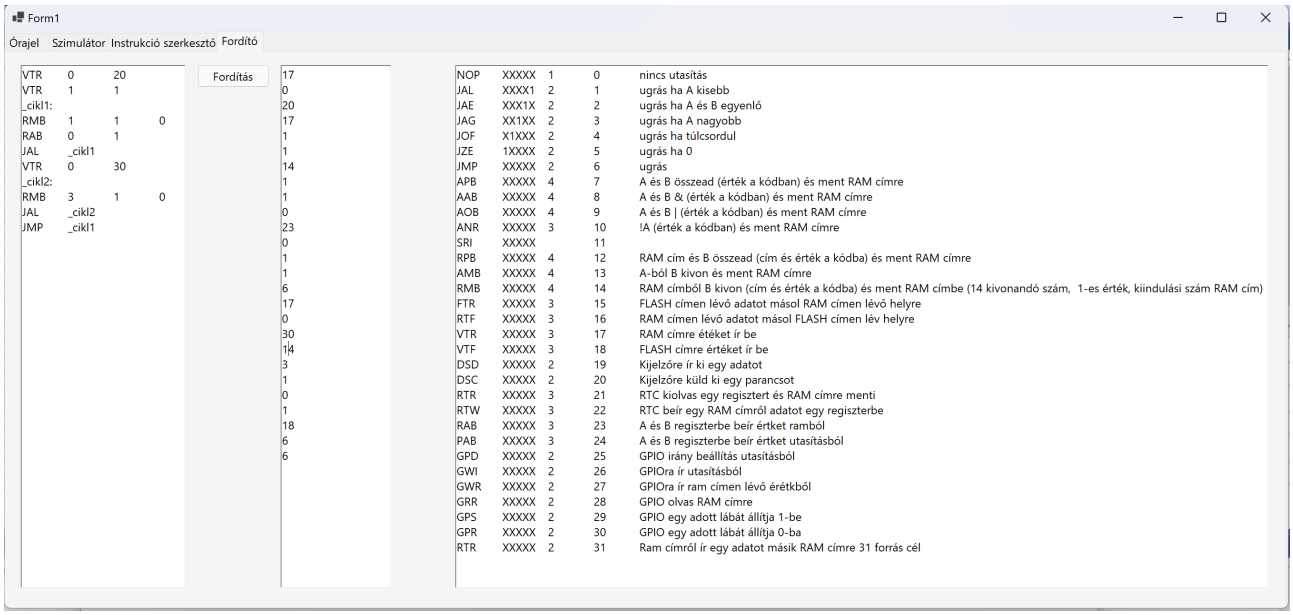
Így az alábbi kód formátum alakult ki:

```
define a    0
VTR    a    0
ciklus:
RPB    a    1    a
JMP    ciklus
```

A define jelöléssel jelzem a programnak, hogy a tabulátor karakter után egy változó neve található, a következő tabulátor után pedig a változó címe. Ezután a szövegben egyszerűen kicseréli az összes megegyező karaktert a címként definiált értékre. Ezt a kódot tartalmazza az I. melléklet.

Ahhoz, hogy az ugrási címekkel se kelljen manuálisan foglalkozni, töréspontokat lehet elhelyezni a szoftverben, amiket el kell nevezni, és a sor végére egy kettőspontot tenni, ebből tudja a fordító, hogy egy olyan soron áll, ami ugrás célcím lehet.

Ezt a sort összevonja a következő sorral, majd a sorszáméra cseréli az összes olyan hivatkozást, ami erre a pontra mutat. Így anélkül lehet a programot módosítani, bővíteni, hogy kézzel újra kellene számolni az összes ugrási pontot.



40. ábra *Fordító program*

A *40. ábrán* látható a fordító program első működőképes változatáról készült képernyőkép. A bal oldali szövegdobozba kell a programot megírni, a középső szövegdobozban jelenik meg a lefordított szöveg, míg a jobb oldali egy segéd szövegdoboz, ami az utasításkészletet tartalmazza.

A bal oldali szövegdobozban minden sor egy utasításnak felel meg, jelentősen átláthatóbb, mint a középső oszlopban található gépi kód.

13. Próba programok

13.1 Alap funkcióteszt

Az első program, ami megírtam a rendszerre valójában a szimulátor tesztelésére inkább szolgált, mint a számítógép tesztelésére. Ebben a kódban mindössze egy ram címen található érték növelése volt a dolga végtelenítve.

A programot magas szintű programozási nyelvben így lehetne leírni:

var *a* = 0;
while(1){
    a++;
}

Ezt viszont így nem lehet beadni a szimulátornak. Először át kell alakítani arra a bináris kódra, amit utána a rendszer értelmezni tud. Erre a feladatra a korábban megírt fordítóprogram tökéletes.

Miután lefordítottam, betöltöttem a szimulátorba és lefuttattam, minden tökéletesen működött, így tovább léphettem a következő funkciótesztre.

13.3 GPIO vezérlés és LCD

A külvilággal való kapcsolattartás és információcsere fontos része egy számítógép működésének. A műveletek eredményeit csak így képes a felhasználó és a szemlélők számára érthetően közölni. A GPIO tesztelésre az Arduino Uno segítségével megvalósított összeköttetést használtam fel. Először egy egyszerű bináris számláló programot tesztelte, ami az előző funkciótesztthez képest annyiban tért el, hogy a szám értékét egyből a GPIO regiszterébe beírta.

A szimulátoros tesztek során minden tökéletesen működött, így egy egyel bonyolultabb GPIO feladat ellátását teszteltem le: a kijelző vezérlést. Mivel a 16x2 karakteres LCD kijelzők vezérlése nem bonyolult, 8 bites módban alap parancsokkal irányíthatóak, így ezzel kezdtem a felhasználói kapcsolattartás megvalósítását.

13.3.1 HD44780 vezérlésű kijelzők

A legtöbb ma használt eszközben a karakteres kijelzők HD44780-as vezérlő egységgel vannak irányítva. Ennek hatalmas előnye a könnyű kezelhetőség és a magas flexibilitás. Két karaktertáblával rendelkezik, ezeken felül pedig akár 8 egyedi karakter is bele programozható, aminek a programozását minden inicializálás alkalmával meg kell tenni. [9]

Ez a vezérlő maximum 80 karakteres kijelző kezelésére képes, amiket változatosan szoktak elrendezni. A legkisebb kijelzők 8 karaktert jelenítenek meg, míg a legnagyobbak kihasználják a teljes memóriaterületet.

Vezérlése kifejezetten egyszerű, 4 és 8 bites üzemmódokkal rendelkezik. A dolgozatban a 8 bites üzemmódban való használatát részletezem, azonban a 4 bites mód is nagyon hasonló, csupán egy utasítást két lépésben kell kiadni.

Az ezzel a vezérlővel felszerelt kijelzőknek 11 vezérlőlába van: 8 az adatoknak és 3 a vezérléshez. A három vezérlőláb a következő:

- EN: a kijelző engedélyező lába, amíg ez alacsony szinten van, az összes többi láb állapota lényegtelen,
- RS: ezzel a lábbal lehet kijelölni a vezérlőnek, hogy egy utasítást vagy adatot szeretne neki küldeni a vezérlő,
- R/W: írás vagy olvasás műveletét indikáló láb, földre húzva írásmódba kerül a kijelző.

A Register Select láb 0 állásában van lehetőség a kijelzőnek parancsot kiadni az adatlábakon keresztül. [9]

A kijelző indítása után egy inicializálási folyamaton kell végig menni ahhoz, hogy az megfelelően működhessen. Első lépésként a kijelzőt kell törölni, ehhez a DB0 lábat magas állapotba, a többi lábat alacsony állapotba kell húzni, majd az EN lábon egy felfutó élre a parancs lefut.

Második lépésként kell a kiinduló konfigurációt elküldeni a kijelzőnek. Ekkor a DB7-DB4 lábakat alacsony állásba kell kapcsolni, a DB3 lábat pedig fixen magas állásba. a DB2 magas állapota állítja a kijelzőt bekapcsolt állapotba, a DB1 kapcsolja be a kurzort, a DB0 pedig a kurzor villogást.

Az első, kijelző tisztító parancshoz hasonlóan azt is az EN láb felfutó éle aktiválja. Ezután már az RS láb magas állásával folyamatosan lehet a kijelző memóriára küldeni a karaktereket, a kijelző automatikusan lépteti a kurzort. [9]

13.3.2 Számolás teszt

A kijelző bekötése, megfelelő bekonfigurálása után az aritmetikát, a GPIO logikát és a teljesítményt is kiválóan lehet tesztelni egy olyan kóddal, ami egy szám adott helyiértékén lévő elemét számolja és jelzi ki. Ennek a hatékony módja a logikai eltolással megvalósított osztás művelete lenne, azonban ezt ebben a pillanatban még nem támogatta sem a szimulátor sem pedig a valós áramkör, így a sokkal teljesítményigényesebb és lassabb összeadáson és hasonlításon alapuló módszert használtam. Ebben a megoldásban a program először a százasokat, majd a tízeseket és végül az egyeseket számolja meg azzal, hogy addig vonja ki belőle ezeket az értékeket amíg a szám nagyobb náluk. Ez egy nagyon lassú, de egyszerű megoldás, ami egyszerre teszteli a kivonás műveletét és a feltételes ugrást is. A tesztek megfelelően lefutottak, nagyobb számoknál viszont már kifejezetten lassú ez a módszer.

13.4 Bemenetek olvasása

A kijelzés után az eszköznek képesnek kell lennie a felhasználó felől adatokat fogadnia és feldolgoznia. Mivel az egy lábat módosító parancsok során a GPIO olvasás művelete már kipróbálásra került, így azt külön nem teszteltem le. A bemenetre építettem egy billentyűzetet, ami a legutóbb leütött billentyűt folyamatosan felhelyezi a bementre amíg nem kerül nullázásra. Ez a működés alacsony órajel mellett kifejezetten lassú és kényelmetlen, de hardveres megaszakítások nélkül, magasabb órajeleken ez a legegyszerűbb megoldás. Az eszközön futó szoftver, ha szükséges az adatokat átalakítja, amiket a billentyűzetről kapott vagy kijelzi azokat a kijelzőn, vagy pedig tárolja egy memóriacímen.

14. Összegzés

A dolgozat elején kitűzött célt, egy olyan számítógép megépítését, ami megfelel az alapfeltételeknek, amiket egy ilyen eszközzel szemben támasztanak, elértem. A rendszer maximális futási sebessége szimulátorban, a Windows adta korlátok miatt 400Hz.

A számítógép képes az utasításokat előre programozott módon végrehajtani, valamint saját programját módosítani, ezzel megfelelve a Neumann számítógép feltételeinek. A mai modern számítógépek ezeket a dolgozatban tárolt egységeket egy burkolat alá integrálva tartalmazzák a nagyobb sebesség elérése érdekében.

A jövőben az elméletet tovább dolgozva tervezem megvalósítani a rendszert FPGA alapokon, amivel már a magasabb szintű integráltság felé léphetek egyet. Ezen felül a funkciók további bővítése érdekében újabb és újabb kiegészítő paneleket szeretnék megvalósítani, köztük olyanokat is amik szabvány videojelek létrehozására képesek, ezzel még egy lépéssel közelebb kerülve a modernebb számítógépek funkcionalitásához.

Irodalomjegyzék

[1] Tanenbaum, A. S.: *Számítógép-architektúrák*

27-41, 2006

[2] Dr. Seebauer M.: *Számítógép architektúra előadás fóliák*

[3] L. Howard Pollard: *Computer Design and Architecture*

3-18

[4] *Texas Instruments: xx555 Precision Timers*

https://www.ti.com/lit/ds/symlink/ne555.pdf 2022.11.10.

[5] Dr. Vörösvári Zs.: *Digitális rendszeren és számítógép architektúrák*

1. előadás: Bevezetés, számítógép generációs. Neumann-Harvard architektúrák 2022.12.02

[6] Wikipédia: Konrad Zuse

https://en.wikipedia.org/wiki/Konrad_Zuse 2022.12.10.

[7] Texas Instruments - *LM25010, LM25010-Q142-V, 1-AStep-DownSwitchingRegulator*

https://www.ti.com/lit/ds/symlink/lm25010.pdf?ts=1670672722287&ref_url=https%253A%252F%252Fwww.google.com%252F 2022.09.20.

[8] Texas Instruments: *High-Speed CMOS Logic 4-Bit Binary Full Adder with Fast Carry*

https://www.ti.com/lit/ds/symlink/cd74hct283.pdf 2022.10.04.

[9] Hitachi: *HD44780U (LCD-II)*

https://www.sparkfun.com/datasheets/LCD/HD44780.pdf 2022.12.05.

[10] Nexperia: *74HC163; 74HCT163*

https://assets.nexperia.com/documents/data-sheet/74HC_HCT163.pdf 2022.11.20.

[11] Nexpreia: 74HC85; 74HCT85

https://assets.nexperia.com/documents/data-sheet/74HC_HCT85.pdf 2022.12.10.

Mellékletek

1. Melléklet

```
int numberofvars = 0;
        string[,] variabl = new string[2000, 2];
        string place = address_files + @"\INSTRCTIONSET.txt";
        var linecnt = File.ReadLines(place).Count();
        richTextBox5.Text = "";
        richTextBox4.Text = "";
        for(int i = 0; i<linecnt; i++)
        {
           richTextBox5.Text += File.ReadLines(place).Skip(i).Take(1).First();
           if (i < linecnt - 1)
           {
              richTextBox5.Text += "\r\n";
           }
        }

        for (int i = 0; i < richTextBox3.Lines.Count(); i++)
        {
           var line = richTextBox3.Lines[i];
           if (line.Contains("define"))
           {
              line = line.Substring(line.IndexOf("\t") + 1);
              variabl[numberofvars, 0] = line.Substring(0, line.IndexOf("\t"));
              variabl[numberofvars, 1] = line.Substring(line.IndexOf("\t") + 1);
              variabl[numberofvars, 1] = variabl[numberofvars, 1].Replace("\t", "");
              numberofvars++;
           }
        }
        for (int i = 0; i < richTextBox3.Lines.Count(); i++)
        {
           var line = richTextBox3.Lines[i];
           if (!line.Contains("define"))
           {
              for (int j = 0; j < numberofvars; j++)
              {
                 if (line.Contains("\t" + variabl[j, 0]))
                 {
                    line = line.Replace("\t" + variabl[j, 0], "\t" + variabl[j, 1]);
                 }
              }
              line = line.Replace("\t", "\r\n");
              if (!line.Contains(":"))
              {
                 richTextBox4.Text += line + "\r\n";
              }
              else
```

```
        {
            richTextBox4.Text += line;
        }
    }
}
int loops = 0;
for(int i = 0; i< richTextBox4.Lines.Count(); i++)
{
    if (richTextBox4.Lines[i].Contains(":"))
    {
        loops++;
    }
}
for (int looping = 0; looping < loops; looping++)
{
    //richTextBox3.Text = richTextBox4.Text;
    for (int i = 0; i < richTextBox4.Lines.Count(); i++)
    {
        var line = richTextBox4.Lines[i];
        if (line.Contains(":"))
        {
            var criteria = line.Substring(0, line.IndexOf(":"));
            for (int j = 0; j < richTextBox4.Lines.Count(); j++)
            {
                if (j != i)
                {
                    var checking = richTextBox4.Lines[j];
                    if (checking.Contains(criteria))
                    {
                        string[] lines = richTextBox4.Lines;
                        lines[j] = richTextBox4.Lines[j].Replace(criteria, i.ToString());
                        for (int k = 0; k < richTextBox5.Lines.Count(); k++)
                        {
                            if (richTextBox5.Lines[k].Substring(0, 3).Contains(lines[j]))
                            {
                                lines[j] = k.ToString();
                            }
                        }
                        richTextBox4.Lines = lines;
                    }
                }
            }
        }
    }
}
for (int j = 0; j < richTextBox4.Lines.Count(); j++)
{
    string[] lines = richTextBox4.Lines;
    for (int k = 0; k < richTextBox5.Lines.Count(); k++)
```

```
    {
        //if (richTextBox5.Lines[k].Substring(0, 3).Contains(lines[j]))
        if (lines[j].Contains(richTextBox5.Lines[k].Substring(0, 3)))
        {
            lines[j] = k.ToString();
        }
    }
    richTextBox4.Lines = lines;
}
```

2. Melléklet

```
public void calculate()
      {
         double clock_in = 0;
         if (HIGH_CLK.Checked)
         {
            clock_in = Double.Parse(textBox1.Text);
         }
         else if (MIDLE_CLK.Checked)
         {
            clock_in = Double.Parse(textBox2.Text);
         } else if (LOW_CLK.Checked)
         {
            clock_in= Double.Parse(textBox4.Text);
         } else if (EXTERNAL_CLK.Checked)
         {
            clock_in = Double.Parse(textBox3.Text);
         }
         label5.Text = clock_in.ToString() + " Hz";

         double mainclk = clock_in / Double.Parse(SYS_PRES.Text);
         double pwmclk = clock_in / Double.Parse(PWM_PRES.Text);

         CLK.Text = mainclk.ToString();
         PWM.Text = pwmclk.ToString();

         CONF_BIN.Text = PS8.Text + PS7.Text + PS6.Text + PS5.Text + PS4.Text + PS3.Text +
PS2.Text + PS1.Text;
}
```