STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 10: Elektrotechnika, elektronika a telekomunikace

8bit počítač na nepájivém poli

Jakub Jurčík
Zlínský kraj

Rožnov pod Radhoštěm, 2023

STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 10: Elektrotechnika, elektronika a telekomunikace

8bit počítač na nepájivém poli

8-bit breadboard computer

Autoři: Jakub Jurčík
Škola: SŠIEŘ Rožnov pod Radhoštěm, Školní 1610, 756 61
Kraj: Zlínský
Konzultant: Ing. Bohumil Federmann

Rožnov pod Radhoštěm, 2023

PROHLÁŠENÍ

Prohlašuji, že jsem svou práci SOČ vypracoval/a samostatně a použil/a jsem pouze prameny a literaturu uvedené v seznamu bibliografických záznamů.

Prohlašuji, že tištěná verze a elektronická verze soutěžní práce SOČ jsou shodné.

Nemám závažný důvod proti zpřístupňování této práce v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších předpisů.

V Rožnově pod Radhoštěm dne 20. 3. 2023 ……

Jakub Jurčík

PODĚKOVÁNÍ

Chtěl bych poděkovat nejprve panu Ing. Lukáši Haplovi, Ph.D., jehož přístup a způsob výuky číslicové techniky zapříčil můj zájem o tento předmět.

Dále patří poděkování americkému youtuberovi Ben Eater. Jeho série „Building an 8-bit breadboard computer!“ byla hlavní inspirací této práce.

Taktéž děkuji panu Ing. Bohumilu Federmannovi za vedení práce.

ANOTACE

Tato práce se zabývá návrhem a realizací 8bitového počítače pomocí integrovaných obvodů. V této dokumentaci bude čtenář seznámen se základními pojmy z teorie číslicové techniky. V praktické části následuje architektura projektu, popis jednotlivých modulů počítače a samotná fyzická realizace.

Záměrem práce nebylo vytvoření co nejvýkonnějšího systému, ale vytvoření funkčního celku, který svou jednoduchostí umožňuje dopodrobna pochopit způsob jeho fungování.

KLÍČOVÁ SLOVA

Počítač, číslicové systémy, architektura počítačů

ANNOTATION

This project consists of the design and implementation of an 8-bit computer using integrated circuits. In this documentation, the reader will be introduced to basic concepts from the theory of digital electronics. The architecture of the project, description of individual modules of the computer, and the physical implementation itself follow in the practical part.

The goal of this project was not to create the most powerful system possible, but to create a functional unit, simple enough to provide a deep understanding of its inner workings.

KEYWORDS

Computer, digital systems, computer architecture

OBSAH

ÚVOD

Počítače se používají prakticky ve všech částech lidského života od práce po zábavu. Většina lidí bere jejich činnost za samozřejmost. Mnoho z nich však nemá tušení, co se odehrává uvnitř. To, že počítač obsahuje základní desku, procesor, grafickou kartu a další, je v dnešní době známo. Jak ale skutečně fungují jednotlivé součásti počítače už ví jen poměrně malá skupina lidí.

Cílem této práce bylo postavit jednoduchý počítač z jednotlivých integrovaných obvodů. Práce má hlavně naučné zaměření, cílem není postavit co nejvýkonnější stroj, nýbrž pochopit princip fungování výpočetních zařízení.

I. TEORETICKÁ ČÁST

1 POČÍTAČ

Počítač je elektronické číslicové zařízení, které slouží ke zpracování dat. Činí tak vykonáváním předem vytvořené posloupnosti instrukcí neboli programu. Počítače pracují ve dvojkové soustavě. Veškerá data, s kterými pracují, tedy musí být zakódována do nul a jedniček. Programy, které počítač vykonává, vycházejí z algoritmů.

1.1 Algoritmus

Algoritmus je předem daný přesný postup. Jedná se o teoretický způsob řešení určitého problému. Aby se určitý postup dal považovat za algoritmus, musí splňovat několik vlastností [1][2][3]:

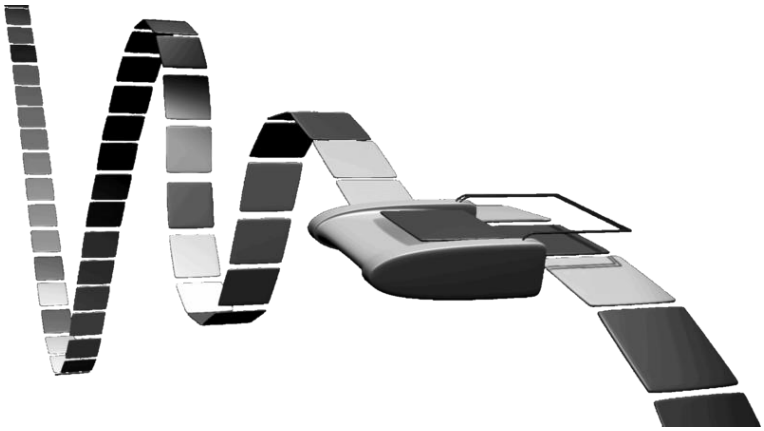
- Konečnost – algoritmus musí být složen z konečného počtu kroků. Počet kroků může být libovolně velký, avšak konečný.
- Elementárnost – algoritmus se skládá z jednoduchých kroků.
- Determinovanost – postup práce algoritmu je přesně dán, a závisí jen na jeho krocích a vstupu. To znamená, že pro stejný vstup vždy vrátí stejný výsledek.
- Determinismus – každý krok algoritmu musí být přesně definován. V každém kroku musí být stanoveno, co se má provést dál.
- Obecnost – algoritmus dokáže řešit více než jeden konkrétní případ, dá se použít pro množinu vstupů.
- Vstup – algoritmus má aspoň jeden vstup.
- Výstup – algoritmus má konečný výstup, odpověď na problém, který pro daný vstup řeší.

1.2 Turingův stroj

Základem výpočetních systémů je teoretický model britského matematika Alana Turinga. Jedná se o Turingův stroj. Obsahuje nekonečně dlouhou pásku složenou z buněk, čtecí a záznamovou hlavu, která se může po pásce pohybovat, a řídící jednotku s konečným počtem stavů.

Řídící jednotka Turingova stroje obsahuje definici tzv. přechodových funkcí. Jsou to funkce, které udávají, kdy a jak se záznamová hlava po pásce posune. Tyto přechodové funkce jsou obdoba programu počítače. [4]

Detailní popis Turingova stroje je na rozsah této práce příliš komplikované téma. Důležité je však zmínit, že Turingův stroj je podle Church-Turingovy teze model dostatečně výkonný na to, aby byl schopen vypočítat jakýkoli algoritmus. Jinými slovy to znamená, že ke každému algoritmu existuje ekvivalentní Turingův stroj a naopak.



Obr. 1.1. Umělecké vyobrazení Turingova stroje [5]

1.2.1 Turingovská úplnost

Důležitý pojem je tzv. Turingovská úplnost. Výpočetní systém je Turingovsky úplný, pokud dokáže simulovat práci Turingova stroje. Aby toho byl schopen, musí obsahovat alespoň jednu konstrukci podmíněného skoku – tj. větvení programu na základě výsledku předchozích operací. [6]

2 BOOLEOVA ALGEBRA

Základem číslicových obvodů je Booleova algebra, pojmenovaná podle anglického matematika George Boolea. Ten v první polovině 19. století ustanovil základní pravidla pro práci s binárními logickými hodnotami. Jsou to takové hodnoty, které mají dva možné stavy. Prvnímu stavu se říká logická jednička, druhému logická nula. V číslicových obvodech se tyto stavy označují jako LOW a HIGH podle napěťových úrovní. [7]

Boole pro logické proměnné definovat tři základní operace:

- Logický součet – aspoň jedna proměnná musí být v log. 1, označuje se symbolem plus (+)
- Logický součin – všechny proměnné musí být v log. 1, označuje se symbolem krát (×)
- Negace – změna proměnné z jednoho stavu do druhého, označuje se symbolem ~ nebo / nebo čarou nad proměnnou

Z těchto operací se pomocí stanovených pravidel dá realizovat jakákoli logická funkce, včetně složitých výpočetních systémů. Ve skutečnosti ale základní operace stačí dvě, protože logický součet se dá sestavit pomocí logického součinu a negace, stejně jako log. součin se dá vytvořit z log. součtu a negace. Standardně se ale jako základní operace uvádějí všechny tři.

2.1 Zákony Booleovy algebry

Aby bylo možné s výrazy v Booleově algebře pracovat, je nutné znát její zákony. Ty pracují s výše zmíněnými základními operacemi. Patří mezi ně následující zákony [7]:

- Zákon komutativní:
  - $A + B = B + A$
  - $A \times B = B \times A$
- Zákon asociativní:
  - $(A + B) + C = A + (B + C)$
  - $(A \times B) \times C = A \times (B \times C)$
- Zákon distributivní:
  - $A \times (B + C) = A \times B + A \times C$
  - $A + (B \times C) = (A + B) \times (A + C)$

- Zákon dvojité negace:
  - ~(~A) = A
- Zákon komplementu:
  - A × ~(A) = 0
  - A + ~(A) = 1
- Zákon idempotence:
  - A + A = A
  - A × A = A
- Zákon neutrálnosti log. hodnot:
  - A × 1 = A
  - A + 0 = A
- Zákon agresivnosti log. hodnot:
  - A × 0 = 0
  - A + 1 = 1
- De Morganovy zákony:
  - ~(A + B) = ~A × ~B
  - ~(A × B) = ~A + ~B

2.2 Logické funkce

Logická funkce je funkce, která pracuje s logickými hodnotami podle zákonů Booleovy algebry. Má jeden a více vstupů a alespoň jeden výstup. Logické funkce je možno minimalizovat. Buďto pomocí algebraické metody, tedy uplatnění zákonů Booleovy algebry, nebo pomocí tzv. Karnaughových map. Logické funkce se kromě algebraického zápisu dají vyjádřit i pravdivostní tabulkou. [7]

2.3 Pravdivostní tabulky

Pravdivostní tabulka je tabulka popisující logickou funkci. Obsahuje sloupce pro vstupy a výstup. Každý řádek je jedna kombinace vstupních proměnných. Z toho vyplývá, že aby tabulka plně popsala funkci, musí mít tolik řádků, kolik je možných kombinací vstupů. Protože logické funkce pracují s binárními proměnnými, je pro *n* vstupů 2^n možných kombinací. Z pravdivostní tabulky se dá vyčíst logická funkce, a naopak z logické funkce se dá sestavit pravdivostní tabulka. [7]

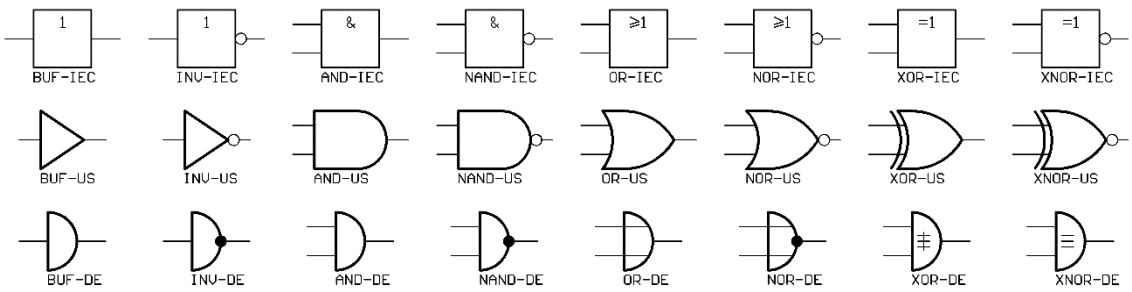
Pokud máme například logickou funkci Y = ~A + B × ~C, vypadala by pravdivostní tabulka následovně:

Tab. 2.1. Pravdivostní tabulka logické funkce

C	B	A	Y
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

2.4 Logická hradla

Logická hradla jsou součástky, které realizují určité logické funkce. Mají jeden a více vstupů, a pouze jeden výstup. Jsou základními prvky číslicových obvodů.



Obr. 2.1. Schématické značky logických hradel podle různých norem [8]

Vyobrazené hradla odpovídají těmto funkcím:

- BUF – opakovač, Y = A
- INV – negace, Y = ~A
- AND – logický součin, Y = A × B
- NAND – negovaný logický součin, Y = ~(A × B)
- OR – logický součet, Y = A + B
- NOR – negovaný logický součet, Y = ~(A + B)
- XOR – exkluzivní disjunkce, Y = A ⊕ B = ~A × B + A × ~B
- XNOR – negace exkluzivní disjunkce, Y = ~(A ⊕ B) = ~A × ~B + A × B

K výše uvedeným logickým hradlům připadají následující pravdivostní tabulky.

Tab. 2.2. Pravdivostní tabulka hradel s jedním vstupem

A	BUF	INV
0	0	1
1	1	0

Tab. 2.3. Pravdivostní tabulka hradel se dvěma vstupy

B	A	AND	NAND	OR	NOR	XOR	XNOR
0	0	0	1	0	1	0	1
0	1	0	1	1	0	1	0
1	0	0	1	1	0	1	0
1	1	1	0	1	0	0	1

3 ČÍSLICOVÉ OBVODY

Číslicové obvody jsou obvody tvořeny pomocí logických hradel. Dělí se na dvě hlavní skupiny.

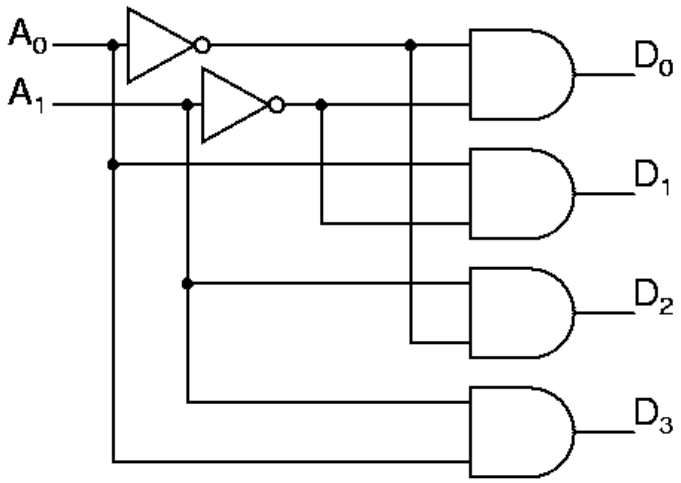
3.1 Kombinační obvody

Kombinační obvody jsou takové obvody, kde výstup přímo závisí na kombinaci jeho vstupů. Změna vstupů se tedy okamžitě projeví na výstupech. Jedna kombinace vstupů odpovídá jedné kombinaci výstupů. Práci kombinačních obvodů lze popsat pravdivostní tabulkou. [7]

Příkladem kombinačního obvodu může být dekodér. Pro *n* vstupů má tento obvod 2^n výstupů. Každý výstup je aktivní pouze při své kombinaci vstupů. Výstupy mohou být aktivní v jedničce nebo v nule.

Tab. 3.1. Pravdivostní tabulka dekodéru 1 ze 4

A_1	A_0	D_3	D_2	D_1	D_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

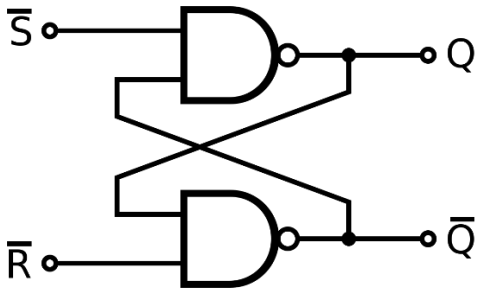


Obr. 3.1. Zapojení dekodéru 1 ze 4 z hradel AND a NOT [9]

3.2 Sekvenční obvody

Sekvenční obvody na rozdíl od kombinačních kromě okamžité kombinace vstupů ovlivňuje i jejich předchozí stav. Znamená to, že takový obvod je schopen zapamatovat si určitý stav i po tom, co vstupní signál zmizí. Kromě pravdivostních tabulek se pro jejich popis používají i tzv. stavové diagramy. [7]

Základním sekvenčním obvodem je klopný obvod RS. Disponuje dvěma vstupy: vstup S od anglického Set nastavuje výstup Q do log. 1. Oproti tomu vstup R od Reset nastavuje výstup Q do log. 0. Realizovat tento obvod je možné pomocí např. dvou hradel NAND nebo dvou hradel NOR.



Obr. 3.2. Zapojení obvodu RS z hradel NAND [10]

Sekvenční obvody se podle přítomnosti hodinového signálu dělí na obvody:

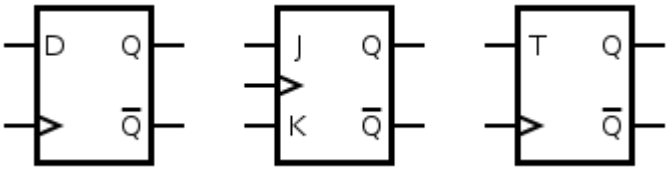
- Asynchronní, hodinový signál zde není použit, obvod mění stav okamžitě při příslušné změně stavů.
- Synchronní, kde se stav obvodu podle vstupů mění až po příchodu hodinového signálu.

Synchronní obvody se podle reakce na hodinový signál dále dělí na:

- Hladinové – tyto obvody jsou aktivní při určité hladině (úrovni) hodinového signálu
- Hranové – tyto obvody reagují pouze na nástupnou nebo sestupnou hranu hodinového signálu

Většina používaných sekvenčních obvodů jsou synchronní hranové. Kvůli tomu, že na rozdíl od hladinových mění stav pouze v jednom bodě v čase, mnohem lépe se synchronizují a je možné z nich stavit složitější systémy.

Do této skupiny patří např. obvod D flip-flop, který při řídící hraně hodinového signálu nastaví výstup Q na hodnotu vstupu D. Pokud připojíme negovaný výstup ~Q na vstup D, při každé řídící hraně hodinového signálu se výstup překlopí. Tímto se dá vytvořit obvod T flip-flop, z anglického Toggle.

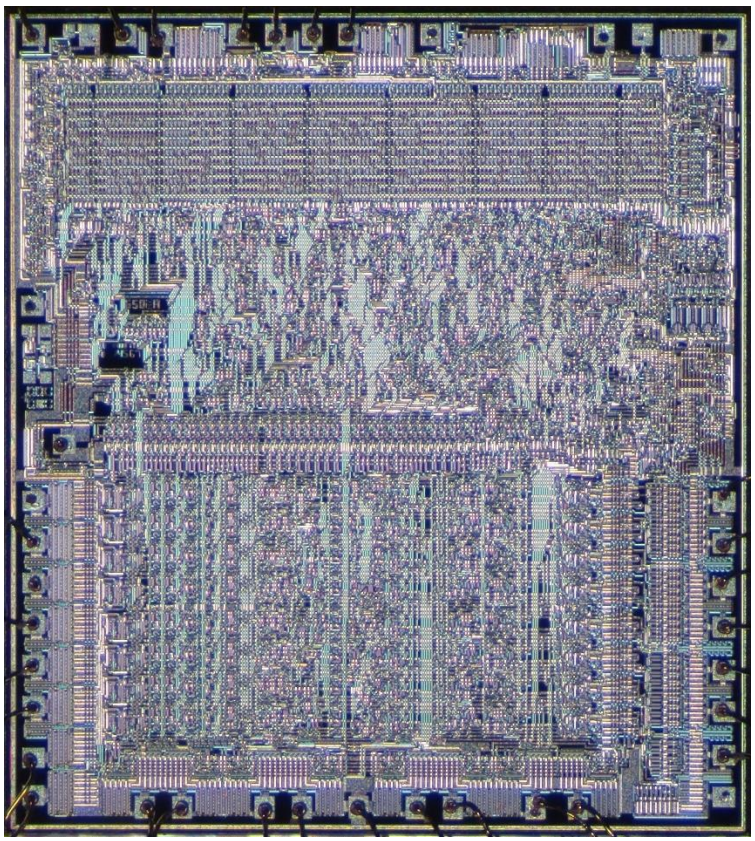


Obr. 3.3. Blokové značky různých typů sekvenčních obvodů [11]

Využití sekvenčních obvodů je rozsáhlé. Obvod RS se dá například použít pro ošetření zákmitů u přepínače. Z obvodů D flip-flop se může vytvořit posuvný nebo paralelní registr. Obvody typu T jde použít pro realizaci čítačů. Různých využití je mnoho.

4 INTEGROVANÉ OBVODY

Integrované obvody (IO) jsou elektronické součástky. Jedná se o obvody vytvořené diskrétními součástkami na polovodičové destičce, většinou z křemíku. Tato destička se nazývá čip. Je opatřena vývody a zapouzdřena.



Obr. 4.1. Křemíkový čip procesoru MOS6502 [12]

Způsobů dělení integrovaných obvodů je celá řada. Hlavní rozdělení je na obvody digitální a analogové. Tato práce se soustředí na digitální neboli číslicové obvody. Další možné rozdělení je podle míry integrace, programovatelnosti, konstrukce apod. [13]

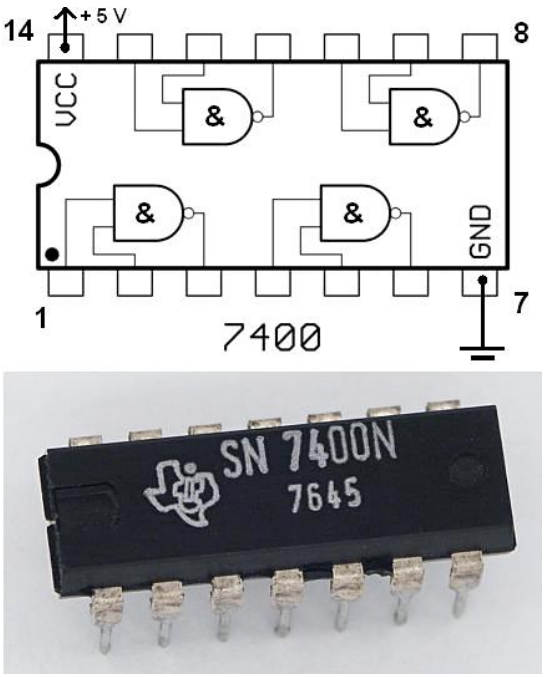
Integrované obvody použity pro realizaci této práce jsou (kromě paměťových) řady 74, a dají se rozdělit do čtyř skupin:

4.1 Hradlové

Hradlové IO jsou obvody, které plní funkci logických hradel. Jeden z nejznámějších IO je obvod 7400, který obsahuje celkem čtyři hradla NAND. Další obvody z této skupiny jsou:

- 7402 – čtyři hradla NOR
- 7404 – šest hradel NOT

- 7408 – čtyři hradla AND
- 7432 – čtyři hradla OR
- 7486 – čtyři hradla XOR

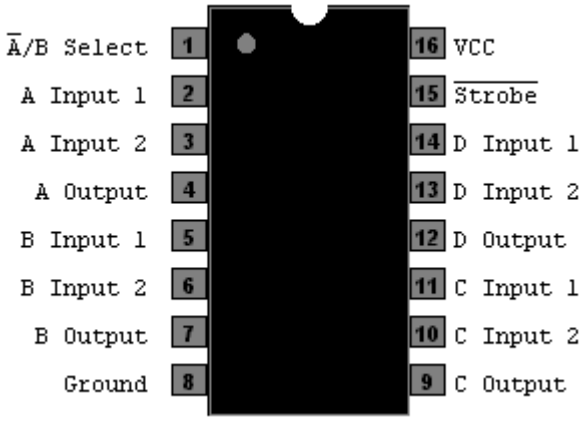


Obr. 4.2. Integrovaný obvod 7400 a zapojení jeho vývodů [14]

4.2 Kombinační

Tyto IO realizují složitější kombinační obvody. Pro tuto práci byly využity následující obvody:

- 74138 – dekodér 1 z 8, výstupy aktivní v nule
- 74157 – čtveřice multiplexorů 2-1

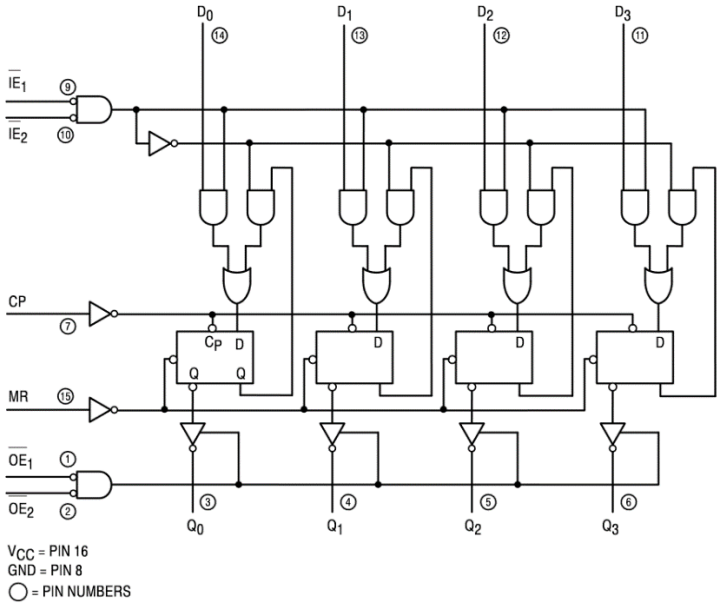


Obr. 4.3. Popis vývodů integrovaného obvodu 74157 [15]

4.3 Sekvenční

Integrované obvody tohoto typu realizují různé sekvenční obvody. V této práci byly použity pro registry a čítače. Patří sem následující IO:

- 74161 – čtyřbitový čítač
- 74164 – osmibitový posuvný registr
- 74169 – čtyřbitový obousměrný čítač
- 74173 – čtyřbitový paralelní registr



Obr. 4.4. Vnitřní zapojení integrovaného obvodu 74173 [16]

4.4 Paměťové

Paměťové integrované obvody jsou z vyjmenovaných skupin nejsložitější. Uchovávají data. Dají se rozdělit na:

- Volatilní – paměť, která po ztrátě napájení ztrácí data
- Nevolatilní – paměť, která po ztrátě napájení data uchovává i nadále

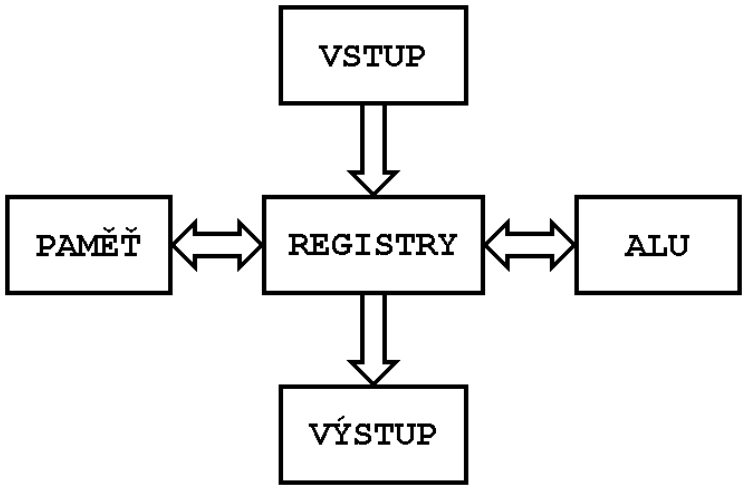
Volatilní paměť je v tomto projektu použita pro operační paměť počítače. Jedná se o statickou paměť RAM. Nevolatilními pamětmi jsou v této práci programovatelné paměti pro čtení typu EEPROM. Použity jsou jako hlavní část řadiče.

II. PRAKTICKÁ ČÁST

5 ARCHITEKTURA POČÍTAČE

5.1 Rozdělení paměti

Obecně se architektury počítačů podle rozdělení paměti dělí na Harvardskou a Von Neumannovu. Harvardská architektura má paměť zvlášť pro program a zvlášť pro data, zatímco Von Neumannova pro program i data používá tutéž paměť. Tento počítač má paměť pro program i data společnou – jedná se tedy o Von Neumannovu architekturu.



Obr. 5.1. Architektura počítače

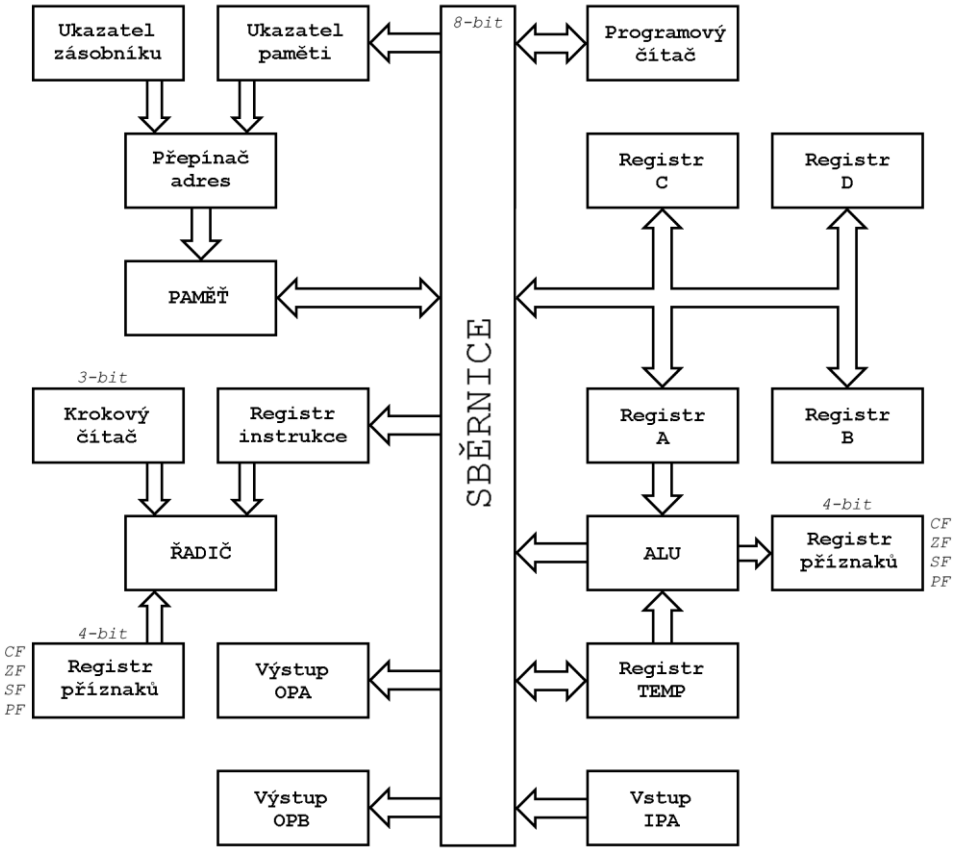
Počítač je také vybaven zásobníkem, který má svůj vlastní paměťový blok. Není tedy možné přímo manipulovat se zásobníkem jako s částí paměti, jde to pouze přes určené instrukce.

5.2 Sběrnice

Zajímavostí zvolené architektury je fakt, že počítač nemá zvlášť datovou a adresovou sběrnici. Data i adresy se mezi moduly přenášejí po stejné sběrnici. Samostatné sběrnice by totiž nepřinesly mnoho užitku. Moduly pracující s adresami by tak i tak musely být připojeny na datovou sběrnici, aby bylo možné provádět skoky, nebo například číst data z adres uložených v registrech. Proto bylo zvoleno mnohem jednodušší řešení společné sběrnice.

5.3 Modularita

Jednotlivé součásti počítače se chovají jako samostatné moduly. S ostatními moduly jsou spojeny sběrnicí a s řadičem jsou spojeny řídícími signály.



Obr. 5.2. Blokové schéma jednotlivých částí počítače

6 REALIZACE PROJEKTU

6.1 Napájecí zdroj

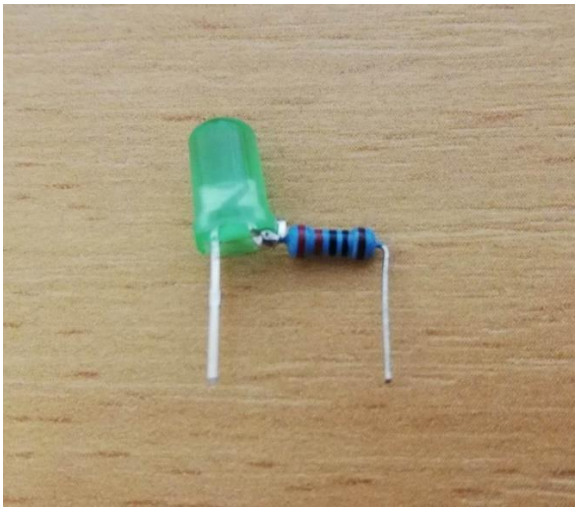
Počítač je zhotoven ze součástek pracujících při 5 V. Jako zdroj napětí je použit zdroj nabíječky mobilního telefonu. Ten je schopen měnit střídavé síťové napětí 230 V na stejnosměrné napětí 5 V. Použitý zdroj má maximální odběr proudu 2 A, počítač při běhu odebírá 1,3 A.

6.2 Realizace

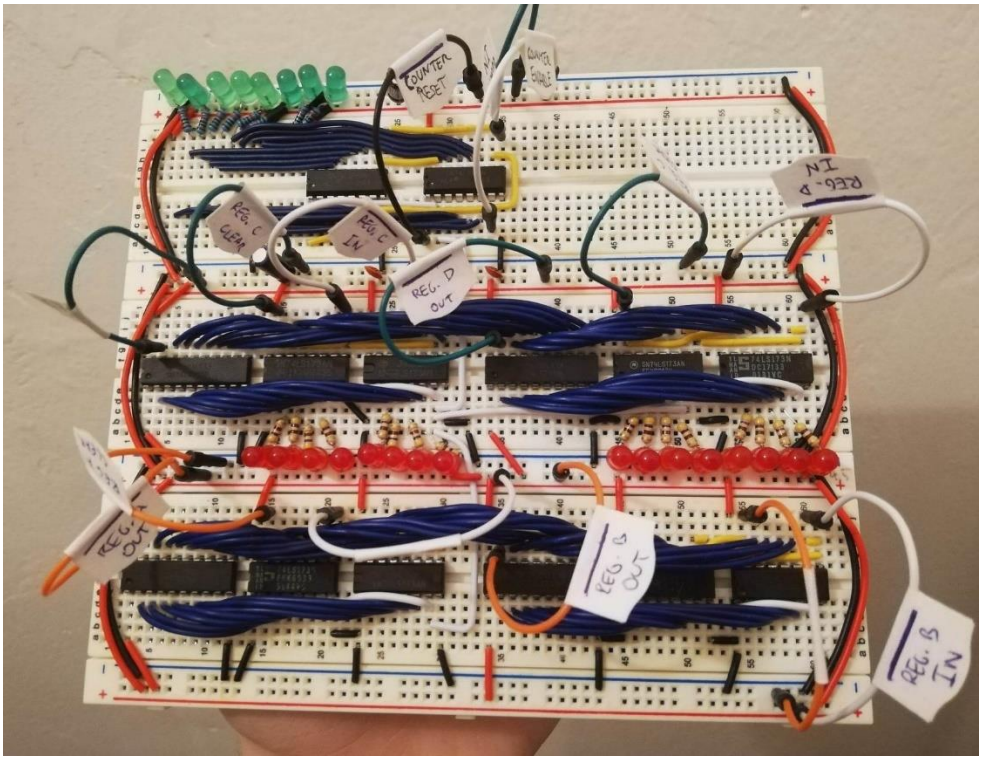
Projekt byl postaven na celkem čtrnácti nepájivých polích. Oproti deskám plošných spojů nebo prototypovým deskám mají nepájivá pole několik výhod. Jedná se zejména o jednoduchost vytváření spojů, a hlavně o snadné odladění chyb. Vytvářet moduly na prototypových deskách by bylo příliš zdlouhavé. To samé platí o desce plošných spojů. U těch je ale větší problém nemožnost opravy případných chyb po výrobě desky.

Na druhou stranu nepájivá pole přináší několik nevýhod. Koneckonců se jedná o pouhou pomůcku pro návrh obvodů. Hlavní nevýhoda je zároveň i hlavní výhoda – tou je skutečnost, že spoje nejsou permanentní. Další nevýhoda je velká parazitní kapacita polí, která znemožňuje stavit obvody pro vysoké frekvence. To u tohoto projektu však takový problém nebyl. Dále je nutno zmínit přechodové odpory. Kvůli nim musely být vytvořeny propojovací vodiče pro spolehlivou distribuci napájení.

Některé spoje nemohly být realizovány v rámci nepájivého pole. Jedná se hlavně o rezistory pro LED. Ty proto byly ve většině případů přímo připájeny k anodě LED:

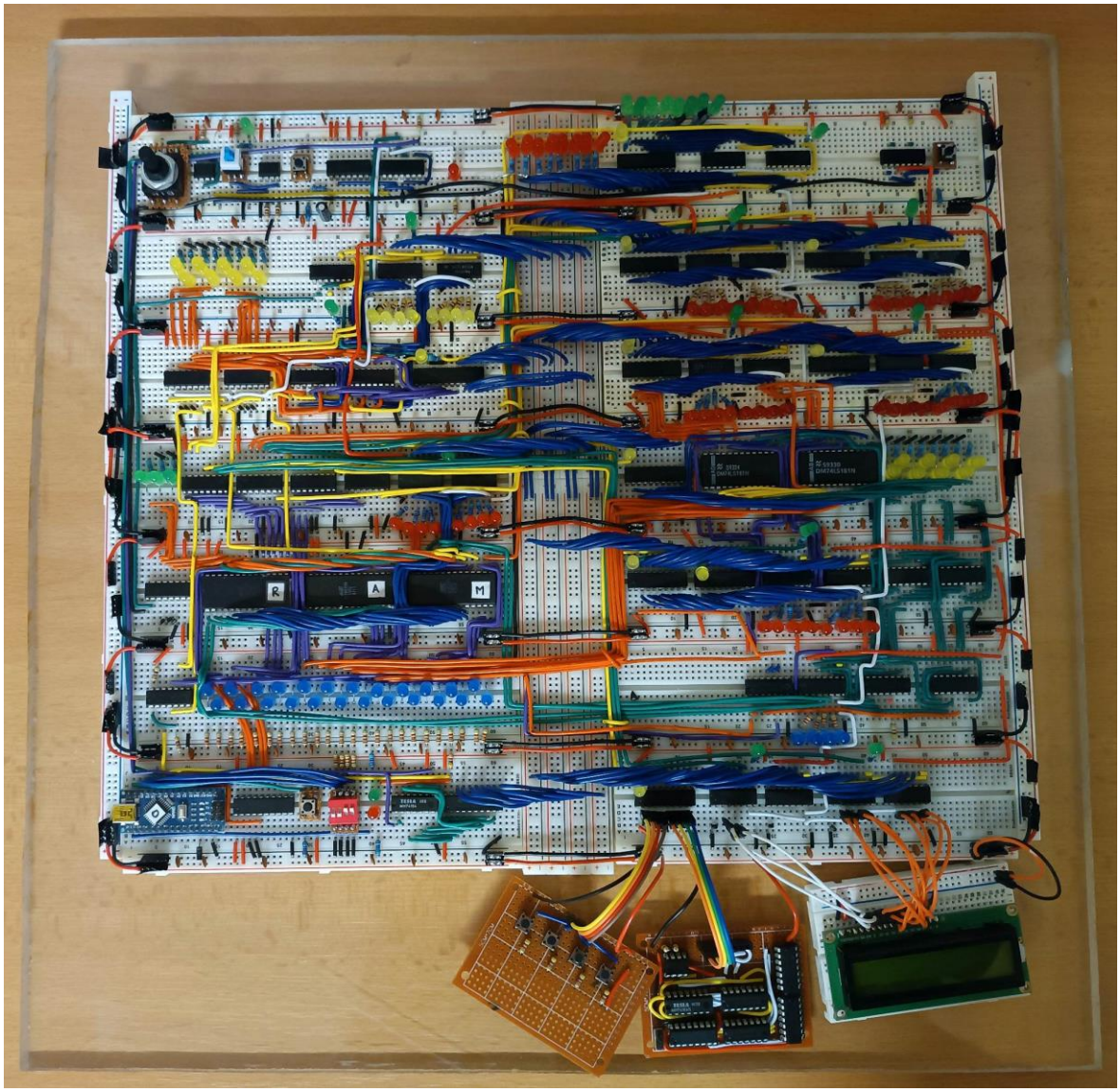


Obr. 6.1. LED s připájeným rezistorem



Obr. 6.2. Projekt v rané fázi realizace

Pro snazší manipulaci byly nepájivé pole přilepeny oboustrannou lepící páskou na spodu polí k čtvercové základně z průhledného plexiskla. Základna má rozměry 45x45x1 cm. Vstupní a výstupní moduly k plexisklu přilepeny nejsou, proto je důležité projekt udržovat ve vodorovné poloze.



Obr. 6.3. Finální verze projektu

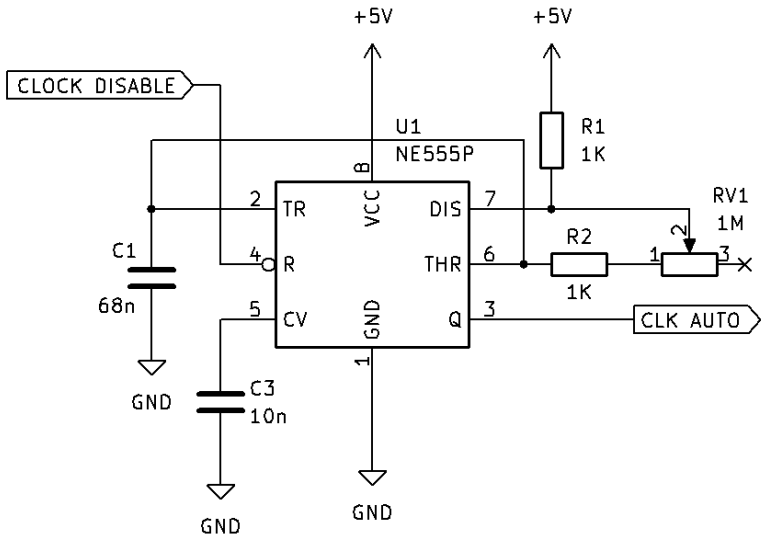
7 NÁVRH INTERNÍCH MODULŮ

7.1 Generátor hodin

Generátor hodin je jedna z nejdůležitějších součástí každého počítače. Vytváří hodinový signál, pomocí kterého jsou synchronizovány veškeré ostatní části. Obecně platí, že vyšší taktovací frekvence je lepší. Znamená to totiž, že výpočetní jednotka je za stejný časový úsek schopna vykonat více instrukcí.

Cíl tohoto projektu nebyl co nejvyšší výkon, ale dosáhnout co největšího porozumění fungování procesoru na skoro nejnižší úrovni. Kvůli tomu je důležité, aby byl generátor hodin poměrně flexibilní. Musí být schopen vytvářet taktovací frekvenci vysokou i nízkou. Kromě toho taky musí nabízet možnost přepnout výstup z automatických hodin na manuální, aby uživatel mohl počítač krokovat.

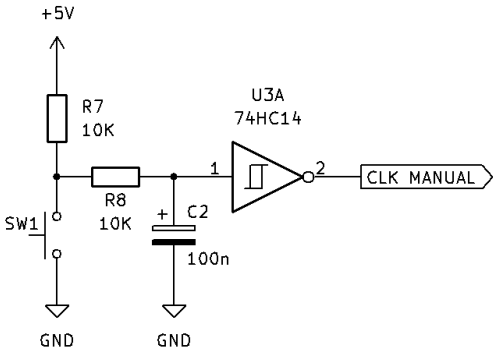
Samotný generátor obdélníkového signálu je realizován časovačem NE555 v astabilní konfiguraci. Hlavním ovládacím prvkem je 1 MΩ potenciometr, pomocí kterého se nastavuje frekvence signálu. Za tímto potenciometrem je ještě 1 kΩ rezistor, aby při maximální rychlosti nedošlo ke zkratu. Pro tento časovač byl zvolen 68 nF kondenzátor. Taktovací frekvence se dá nastavit od 10 Hz po 8 kHz.



Obr. 7.1. Zapojení generátoru obdélníkového signálu

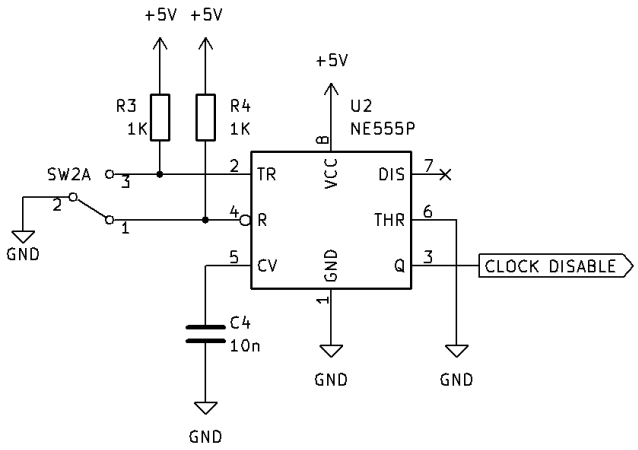
Pro manuální hodiny bylo potřeba ošetřit zákmity tlačítka. Myšlenka manuálního krokování počítače by jinak ztrácela smysl. Zákmity byly ošetřeny jednoduchým obvodem,

kde se při zmáčknutí tlačítka přes rezistory vybíjí kondenzátor. Tento signál pak vyčistí Schmittův klopný obvod v invertoru 74HCT14.



Obr. 7.2. Ošetření zákmitu tlačítka manuálních hodin

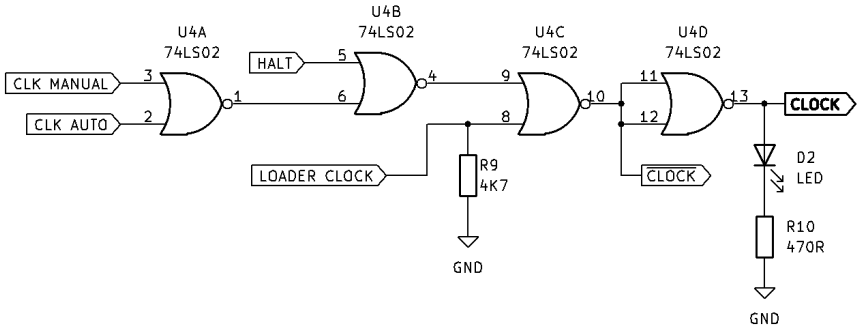
Bylo potřeba ošetřit zákmity i přepínače mezi těmito signály. K tomu byl znovu použit časovač NE555, tentokrát v bistabilní konfiguraci. Výsledkem je prakticky klopný obvod RS v poměrně kompaktním pouzdru. Ošetří se tím neznámý stav, kdy je přepínač v poloze, ve které se nedotýká žádných kontaktů, i případný zákmit po sepnutí.



Obr. 7.3. Ošetření zákmitu přepínače

Než se hodinový signál dostane do potřebných modulů, musí ještě projít jistou logikou. Ta kombinuje automatický a manuální signál do jednoho. Původně byl pro tento účel zamýšlen 2-1 multiplexor, ale nakonec byly tyto signály spojeny hradlem. Nepředpokládá se totiž, že uživatel bude manuálně počítač krokovat, zatímco poběží automatické hodiny. Zároveň logika umožňuje řadiči zastavit počítač pomocí řídícího signálu HALT. Taktéž je potřeba, aby při nahrávání programů do paměti mohl sám pulzovat hodiny i Loader.

K tomu slouží vstup Loader Clock. Obvod byl postaven z hradel NOR. Díky tomu mohl být realizován pouze jedním IO, a to 74LS02.



Obr. 7.4. Výstupní logika generátoru hodin

7.2 Programový čítač

Programový čítač je 8bitový registr, který řídí pořadí vykonávaných instrukcí. Při začátku každé instrukce je v něm uložena adresa právě vykonávané instrukce. Ta se přesune do ukazatele paměti, a programový čítač se inkrementuje. V ten moment ukazuje buď na další instrukci v paměti, nebo na argument právě vykonávané instrukce. Je schopný číst data ze sběrnice – tím se docílí skoků v programu.

Programový čítač je realizován dvojicí synchronních 4bitových čítačů 74LS161 a jednoho budiče sběrnice 74LS245. Ovládán je třemi řídícími signály:

- PC ENABLE – při další nástupné hraně hodinového signálu se hodnota v čítači inkrementuje
- ~ PC IN – při další nástupné hraně hodinového signálu se hodnota v čítači přepíše hodnotou ze sběrnice
- ~ PC OUT – hodnota v čítači se vypíše na sběrnici

7.3 Univerzální registry

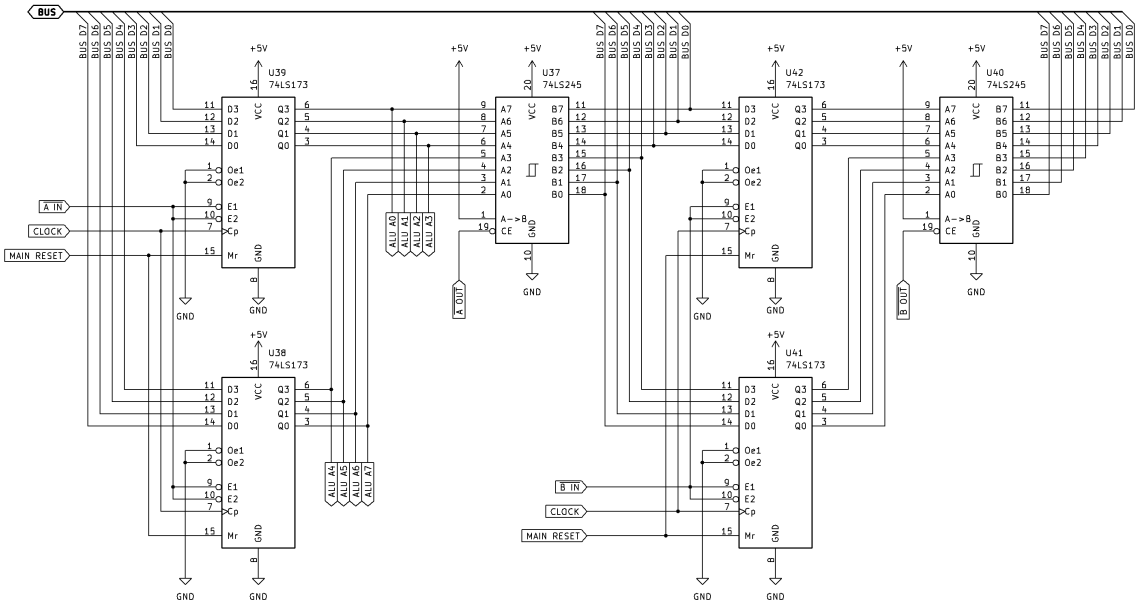
Tento počítač obsahuje čtyři univerzální registry. Jsou to registry A, B, C a D. Registr A také slouží jako první operand aritmeticko-logické jednotky. Všechny registry jsou 8bitové.

Každý registr je realizován dvojicí 4bitových registrů 74LS173 a budičem sběrnice 74LS245. Integrované obvody 74LS173 sice umí pracovat se třístavovou logikou, avšak cíl projektu je do hloubky pochopit způsob práce procesorů (byť jednoduchých). Proto jsou

jejich výstupy vždy aktivní, aby šla vidět hodnota registru na LED indikátorech. Třístavovou logiku a interakci se sběrnicí proto zajišťuje budič sběrnice 74LS245.

Všechny tyto registry jsou ovládány dvěma řídícími signály (místo X lze doplnit A, B, C nebo D):

- ~ X IN – při další nástupné hraně hodinového signálu se hodnota v registru přepíše hodnotou ze sběrnice
- ~ X OUT – hodnota v registru se vypíše na sběrnici



Obr. 7.5. Zapojení registrů A a B

7.4 Paměť

Pro jednoduchost zpracování je adresová šířka paměti 8 bitů. To znamená, že velikost paměti počítače je 256 bajtů. Pro ušetření místa není zásobník část hlavního paměťového prostoru, ale má svůj samostatný 256 bajtů velký blok.

Paměťový modul obsahuje registr pro ukazatel do paměti. Tento registr se jmenuje RP od Ram Pointer, a dokáže pouze číst ze sběrnice. Dále je v modulu registr pro ukazatel do zásobníku – SP od Stack Pointer. Ten je od sběrnice úplně izolován.

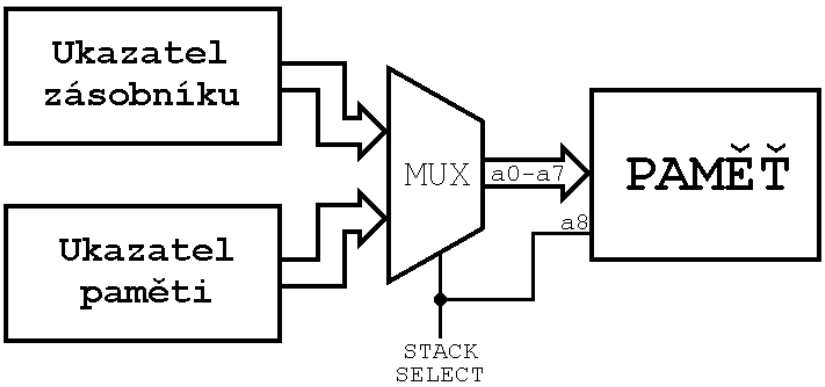
Samotná paměť je realizována integrovaným obvodem CY7C128A, což je statická paměť RAM organizovaná jako 2k x 8. Datové bity jsou přímo připojeny ke sběrnici, jelikož tento IO má vstup i výstup na stejných pinech. To bohužel znamená, že paměť samotná nemá žádné LED indikátory pro data. Hodnotu uloženou v paměti lze tedy okem vidět na

indikátorech sběrnice, a to pouze v moment, co se z paměti čte. Tento IO má 11 adresových bitů, použito je jich 9. Zbylé adresové bity jsou připojeny na 0 V.

Ukazatel do hlavní paměti je realizován pomocí dvou 4bitových registrů 74LS173. Protože se o přepínání mezi hlavní pamětí a zásobníkem stará samostatný blok, výstup z tohoto registru je vždy aktivní. Díky tomu jde pořád fyzicky vidět, na jakou adresu tento registr právě ukazuje.

Ukazatel do zásobníku je zapojen ze dvou 4bitových synchronních obousměrných čítačů 74LS169. Protože není potřeba do tohoto ukazatele nahrávat hodnoty z vnějšku, není připojen ke sběrnici. Všechny datové vstupy jsou připojeny na 0 V. Signál pro čtení, aktivní v nule, je připojen na 5 V. Tento ukazatel vždy ukazuje na další prázdné místo v zásobníku. Pokud se do zásobníku ukládá hodnota, nejdříve se daná hodnota uloží, a poté je SP inkrementován. Pokud se naopak hodnota ze zásobníku čte, nejdříve je nutno SP dekrementovat a až poté hodnotu přečíst. Díky tomu je možné použít vrchol zásobníku jako pomocný registr při vykonávání složitějších instrukcí, aniž by bylo nutné měnit hodnotu ukazatele.

O přepínání mezi ukazatelem do hlavní paměti a zásobníku se stará dvojice 4bitových 2-1 multiplexorů 74LS157. Výstupy jsou připojeny na adresové bity paměťového IO A0-A7. Obvod je vždy aktivní, takže na adresové sběrnici paměti je vždy určitá hodnota. Signál STACK SELECT, který řídí přepínání na multiplexorech také slouží jako devátý bit adresy. Pokud je jeho hodnota 0, vybere se ukazatel do hlavní paměti. Pokud 1, vybere se ukazatel do zásobníku. To znamená, že blok adres 0x000 – 0x0FF je hlavní paměť počítače pro program a data, a blok adres 0x100 – 0x1FF je pro zásobník. Zbylý blok adres 0x200 – 0x7FF je nevyužit.



Obr. 7.6. Blokové schéma ukazatelů paměti

Celý paměťový modul počítače je ovládán těmito řídícími signály:

- ~ RAM IN – při další nástupné hraně hodinového signálu se do paměti uloží hodnota ze sběrnice
- ~ RAM OUT – hodnota v paměti se vypíše na sběrnici
- STACK SELECT – přepínač mezi hlavní pamětí a zásobníkem
- ~ RP IN – při další nástupné hraně hodinového signálu se do ukazatele hlavní paměti uloží hodnota ze sběrnice
- ~ SP ENABLE – při další nástupné hraně hodinového signálu se hodnota v ukazateli do zásobníku inkrementuje nebo dekrementuje
- SP DIR – selektor směru ukazatele do zásobníku, v logické nule ukazatel počítá dolů, v logické jedničce počítá nahoru

7.5 Aritmeticko-logická jednotka

O samotné provádění výpočetních operací se stará aritmeticko-logická jednotka (ALU). Základem jednotky jsou dva 4bitové ALU integrované obvody 74LS181. Pro připojení výstupu na sběrnici je ještě potřeba budič sběrnice 74LS245. Zvolené IO pro ALU vyžadují celkem 6 řídících signálů pro výběr operace. Jsou jimi 4 bity pro samotný výběr operace (ALU SELECT 0-3), 1 bit pro přepínání mezi aritmetikou a logikou (ALU MODE), a 1 bit jakožto přenos z nižšího řádu (ALU CARRY IN). Celkem tento IO umí 48 operací (resp. podporuje 48 různých kombinací řídících signálů), tento počítač jich využívá celkem 18.

Tab. 7.1. Operace používané aritmeticko-logickou jednotkou

M	S_3	S_2	S_1	S_0	C	OPERACE
0	0	0	0	0	0	A + 1
0	0	1	1	0	0	A - B
0	0	1	1	0	1	A - B - 1
0	1	0	0	1	0	A + B + 1
0	1	0	0	1	1	A + B
0	1	1	0	0	0	A + A + 1
0	1	1	0	0	1	A + A
0	1	1	1	1	1	A - 1
1	0	0	0	0	X	~A

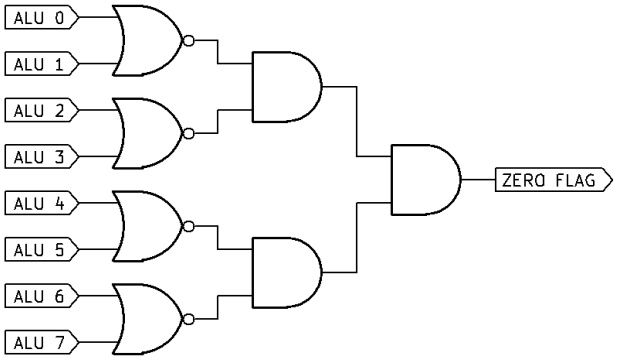
1	0	0	0	1	X	A nor B
1	0	1	0	0	X	A nand B
1	0	1	0	1	X	~B
1	0	1	1	0	X	A xor B
1	1	0	0	1	X	A xnor B
1	1	0	1	0	X	B
1	1	0	1	1	X	A and B
1	1	1	1	0	X	A or B
1	1	1	1	1	X	A

Prvním operandem ALU je registr A, druhý operand je registr TEMP. Výsledek ALU také slouží jako vstup kombinační logiky pro určení příznaků. Počítač pracuje se čtyřmi příznaky:

- CF – Carry Flag, příznak přenosu: aktivní, když operace vyústila v přenos do dalšího řádu nad rámec ALU
- ZF – Zero Flag, příznak nuly: aktivní, když je výsledek operace roven nule
- SF – Sign Flag, příznak znaménka: aktivní, pokud je výsledek operace záporný (dvojkový doplněk)
- PF – Parity Flag, příznak parity: aktivní, když výsledek operace obsahuje lichý počet jedniček

Výstup CARRY OUT z integrovaného obvodu 74LS181 je aktivní v nule, proto je před uložením do příznakového registru potřeba tento signál invertovat. Pokud při sčítání došlo k přetečení do dalšího řádu, je tento výstup aktivní – avšak pokud při odečítání došlo k potřebě vypůjčit z vyššího řádu, je tento výstup neaktivní. Pro jednoduchost realizace bylo zvoleno, aby byl příznak přenosu aktivní v obou případech. Kvůli tomu je potřeba výstup z ALU invertovat při sčítání, a při odčítání ne. Požadavek pro podmíněný invertor splňuje hradlo XOR.

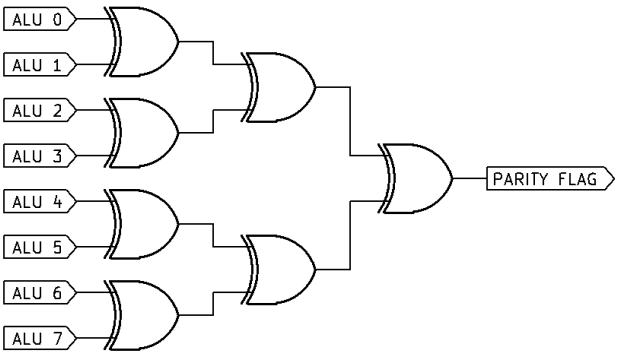
Příznak nuly ZF se zjišťuje pomocí čtyř hradel NOR a tří hradel AND. Výsledek tohoto obvodu je logická jednička pouze tehdy, pokud jsou všechny vstupy hradel NOR v logické nule. Obvod je realizován pomocí IO 74LS02 pro hradla NOR, a IO 74LS08 pro hradla AND. Zapojen je takto:



Obr. 7.7. Zapojení obvodu určujícího příznak nuly

Příznak znaménka SF je přímo připojen na nejvyšší bit výstupu ALU.

Příznak parity PF je určen pomocí sedmi hradel XOR. Jedná se tím pádem o lichou paritu. Logické členy jsou realizovány jsou dvěma IO 74LS86. Dohromady tyto čipy obsahují osm hradel, zbývající hradlo je použito pro podmíněné invertování příznaku přenosu CF. Obvod je zapojen takto:



Obr. 7.8. Zapojení obvodu určujícího příznak parity

Příznaky jsou uloženy v jednom 4bitovém registru realizovaným jedním IO 74LS173. To znamená, že příznaky není možné nastavovat a nulovat samostatně. Pokaždé se nastaví všechny najednou. Toto řešení bylo zvoleno z prostého důvodu – je mnohem jednodušší z hlediska realizace. Výstupy tohoto registru slouží jako nejvyšší adresové bity řídících pamětí EEPROM řadiče.

Některé instrukce nevykonává přímo ALU, ale pouze registr TEMP sám. Jedná se o instrukce Shift Right a Rotate Right. Registr TEMP má pro tento účel dva výstupní budiče sběrnice, oba realizované čipy 74LS245. Jeden budič je na sběrnici připojen přímo, druhý je posunutý o jeden bit vpravo. Pokud se vykonává instrukce Shift Right, do volného nejvyššího bitu se dá logická nula. Když se vykonává instrukce Rotate Right, do

nejvyššího bitu se dá hodnota bitu nejnižšího. O toto přepínání se stará dvojice hradel NAND.

7.6 Vstup a výstup

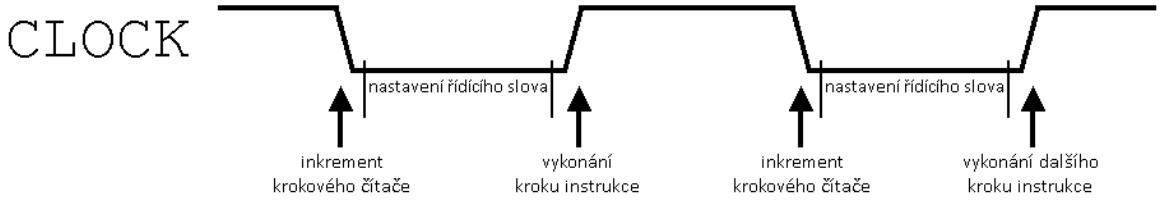
Aby počítač mohl brát vstup od uživatele a zobrazovat výstup, je potřeba vstupní a výstupní rozhraní. Pro tento projekt bylo zvoleno jednoduché řešení. Vstup do počítače zajišťuje jeden 8bitový budič sběrnice 74LS245. Označen je jako IPA od Input Port A. Výstup zajišťují dva 8bitové registry, každý realizovaný dvojicí 4bitových registrů 74LS173. Označeny jsou OPA a OPB od Output Port A a Output Port B. Vstup i výstupy jsou vzhledem k hlavní sběrnici jednosměrné.

7.7 Řadič

Řadič se stará o řízení všech ostatních součástí počítače kromě Loaderu. Podle právě vykonávané instrukce nastavuje správné řídící slovo. Protože se tento řadič chová jako obrovská pravdivostní tabulka, zvolené IO pro realizaci jsou paměti EEPROM.

Každá instrukce se skládá z několika mikroinstrukcí. Každá mikroinstrukce trvá jeden takt. První dva kroky všech instrukcí jsou stejné. První krok je přesunutí hodnoty z programového čítače do ukazatele hlavní paměti. Tím se docílí, že RP ukazuje na další instrukci v pořadí. V druhém kroku se tato hodnota přesune z paměti do registru pro instrukce (IR). Poté se další kroky volí podle instrukce nahrané v IR. Při vykonávání posledního kroku každé instrukce je také nastaven resetovací signál pro krokový čítač. To zajistí, aby každá instrukce zabrala jen tolik času (kroků), kolik potřebuje.

Krokový čítač je realizován jedním 4bitovým synchronním čítačem 74LS161. Pro krokování jsou použity 3 bity, takže každá instrukce může být maximálně 8 kroků dlouhá. Tento IO má vstup pro asynchronní reset. Ten se ale pro ukončení kroku nehodí, vynuloval by čítač před provedením kroku. Čítač 74LS161 ale umožňuje nahrání hodnoty. To znamená, že pokud se všechny vstupy připojí na logickou nulu, stane se signál pro nahrání hodnoty vlastně synchronním resetem. Ten už pro ukončení kroku vhodný je. Krokový čítač není připojen na hodinový signál přímo, ale přes invertor. Díky tomu při sestupné hraně hodinového signálu se tento čítač inkrementuje, a řadič nastaví správné řídící slovo. Při nástupné hraně hodinového signálu se daný krok instrukce vykoná. Čítač se znovu inkrementuje až při následující sestupné hraně.



Obr. 7.9. Průběh hodinového signálu s popisem událostí

Obr. 7.10. Popis událostí při ukončení instrukce

Paměti EEPROM, v kterých jsou uloženy řídící slova pro všechny kroky instrukcí, jsou integrované obvody AT28C256. Tyto paměti jsou organizovány jako 32k x 8. Adresová sběrnice je tedy široká 15 bitů. Adresové bity A0-A7 pocházejí z registru instrukce, a určují, jaká instrukce je právě vykonávána. Další bity A8-A10 určují právě vykonávaný krok instrukce. Poslední čtyři bity adresy A11-A14 pochází z registru příznaků. Pro naprogramování správných dat do těchto pamětí byl postaven open source programátor pamětí na bázi Arduina zvaný TommyPROM.

Některé řídící signály jsou aktivní v nule. Aby ale šlo na LED indikátorech řídícího slova jednoduše vidět, jestli jsou signály aktivní, nebo ne, výstupy z pamětí jsou vždy aktivní v logické jedničce. Kvůli tomu je potřeba některé signály nejdříve invertovat. Jedná se například o signály SP ENABLE, nebo již zmiňovaný reset krokovacího čítače. Pro negaci těchto signálů byl použit IO 74LS04, a IO 74LS00.

Jelikož je vše řízeno výstupy z pamětí EEPROM, nebylo potřeba stavit kódy instrukcí podle toho, co mají vykonávat. Díky tomu může být pro jakoukoli instrukci zvolen jakýkoli kód.

Řídící slovo tohoto počítače je dlouhé 24 bitů. Každý bit má modrou LED připojenou na zem skrz 2.2 kΩ rezistor. Řídící slovo dá se rozdělit do tří částí.

7.7.1 Řízení sběrnice

Pro ovládání čtení ze sběrnice je nutné moci nastavit jeden z 11 signálů, a pro výpis na sběrnici vždy pouze jeden z 10 signálů. Aby se zbytečně neplýtvalo výstupy z paměti, tyto

řídící signály jsou nastavovány pomocí čtveřice dekodérů 74LS138. To přináší několik výhod:

- Pro řízení celkem 21 signálů stačí pouze 8 bitů
- Výstupy zvolených dekodérů jsou aktivní v logické nule, a řídící vstupy skoro všech ovládaných prvků jsou také aktivní v logické nule
- Eliminace rizika zkratu, pokud by dva moduly vypisovaly na sběrnici zároveň

Pro usnadnění realizace a odladění obvodu byly kombinace vstupů voleny tak, aby každá ovládala pouze jeden modul. Pokud například modul umožňuje jen číst ze sběrnice, výstup z dekodérů, který by odpovídal výpisu na sběrnici, není připojen (nc).

Tab. 7.2. Vstupy a výstupy řídících dekodérů

ČTENÍ ZE SBĚRNICE					**VÝPIS NA SBĚRNICI**				
RR_3	**RR_2**	**RR_1**	**RR_0**	**REGISTR**	**RW_3**	**RW_2**	**RW_1**	**RW_0**	**REGISTR**
0	0	0	0	nc	0	0	0	0	nc
0	0	0	1	A	0	0	0	1	A
0	0	1	0	B	0	0	1	0	B
0	0	1	1	C	0	0	1	1	C
0	1	0	0	D	0	1	0	0	D
0	1	0	1	TEMP	0	1	0	1	TEMP
0	1	1	0	nc	0	1	1	0	TEMP SH
0	1	1	1	PC	0	1	1	1	PC
1	0	0	0	RP	1	0	0	0	nc
1	0	0	1	RAM	1	0	0	1	RAM
1	0	1	0	IR	1	0	1	0	nc
1	0	1	1	nc	1	0	1	1	ALU
1	1	0	0	OPA	1	1	0	0	nc
1	1	0	1	OPB	1	1	0	1	nc
1	1	1	0	nc	1	1	1	0	IPA
1	1	1	1	nc	1	1	1	1	nc

7.7.2 Řízení ALU

ALU tohoto počítače vyžaduje celkem 6 bitů pro zvolení správné operace. Zbývající dva bity jsou použity pro určení, zdali je nutné invertovat příznak přenosu, a pro výběr nejvyššího bitu registru TEMP při vykonávání instrukcí Shift Right nebo Rotate Right.

7.7.3 Ostatní signály

Třetí část řídícího slova se stará o nastavování ostatních signálů:

- SP DIR
- ~ SP ENABLE
- STACK SELECT
- PC ENABLE
- ~ UPDATE FLAGS
- ~ RESET MICROSTEP
- HALT

Jeden bit zůstal nevyužit, v budoucnu se použije pro vylepšení funkcí ALU.

7.8 Arduino loader – Hardware

Aby byl počítač schopen vykonávat programy, musí se daný program nejprve nahrát do hlavní paměti. Nahrávat každou hodnotu ručně by ale bylo příliš zdlouhavé a nepraktické. Pro snadné a rychlé nahrání programů slouží Loader. Obsahuje čtyřbitový přepínač pro zvolení programu, a programovací tlačítko. Loader tím pádem podporuje až 16 různých programů. Všechny musí být předtím nahrány na Arduino Nano, skrz které je tento Loader realizován.

Aby mohlo Arduino nahrát hodnoty do paměti, musí být schopné nastavit si samo řídící signály tak, jak potřebuje. Konkrétně to jsou signály pro zápis do ukazatele paměti, a pro zápis do paměti samotné. Těm odpovídá kombinace řídících signálů 1000 a 1001. Arduinu tím pádem stačí určit poslední bit, protože první tři bity jsou pro oba moduly stejné. Aby však nedocházelo ke konfliktům se signály z řídící paměti EEPROM, kombinace z Arduina je nejdříve připojená na tří stavový budič. Arduino pak speciálním signálem určí, zda je Loader aktivní či nikoliv. Tento signál se chová jako ~OE zmiňovaného budiče, ale zároveň je invertován, a tato negovaná verze slouží jako ~OE řídící paměti EEPROM.

Není tím pádem možné, aby nastal zkrat, protože vždy vstupy dekodérů ovládá pouze jedno zařízení. Tří stavový budič je realizován budičem sběrnice 74LS245.

Nastavování správných hodnot na sběrnici umožňuje posuvný registr a budič sběrnice. Osmibitová hodnota se tím pádem může dostat na sběrnici pomocí pouhých dvou výstupů z Arduina. Jeden pro data, druhý pro hodiny. To, zdali je budič sběrnice aktivní, ovládá stejný signál jako pro budič kombinace dekodéru.

Aby Loader mohl nahrávat hodnoty do synchronních modulů, musí ovládat hlavní hodinový signál počítače. Pokud by však hodiny běžely, došlo by k nahrání špatných dat. Kvůli tomu je Loader možné aktivovat pouze tehdy, jsou-li hlavní hodiny zastaveny. Teprve pak může Loader ovládat hodinový signál podle sebe.

Arduino je připojeno k napájení počítače skrz diodu. Zabrání se tak napájení celého počítače skrz malé cesty na desce Arduina, když se do něj nahrávají nové programy nebo software přes USB. Když je počítač zapnut normálním způsobem, Arduino se zapne taky. Opačnému jevu zabrání zmiňovaná dioda.

7.9 Resetovací tlačítko

Jednoduchý reset počítače zajišťuje resetovací tlačítko. Některé moduly očekávají resetovací signál aktivní v jedničce, některé v nule. O nastavení správné hodnoty se stará dvojice invertorů, realizovaných dvěma hradly NAND. Zmáčknutí tlačítka vynuluje hodnoty všech modulů kromě ukazatele zásobníku, který nemá vstup pro asynchronní reset. Pro správný běh počítače ale není potřeba, aby byl při startu tento ukazatel vynulován.

8 NÁVRH EXTERNÍCH MODULŮ

Každý počítač potřebuje nějaké vstupní a výstupní zařízení. Použity jsou vstupní a výstupní moduly. To zajišťuje jistou míru flexibility – díky realizaci projektu na nepájivém poli lze každý modul jednoduše odpojit, a připojit jiný. Uživatel má také možnost vytvořit své vlastní moduly, které pak může připojit na vstupní a výstupní rozhraní.

8.1 Vstupní

8.1.1 Deska s tlačítky

Nejzákladnější vstup jakéhokoli číslicového systému je prosté tlačítko. Pro tento účel byla vytvořena deska se čtyřmi tlačítky, aby modul nezabral všechny dostupné vstupní bity. Tlačítka jsou aktivní v nule, výstup je vždy přes pull up rezistor připojen na 5 V, při zmáčknutí tlačítka se připojí na 0 V.

8.1.2 Generátor pseudonáhodných čísel

Pro jisté programy byl potřeba vymyslet způsob, jak generovat náhodná, nebo pseudonáhodná čísla. Původní nápad byl neustále inkrementovat jeden z registrů, a přečíst hodnotu po stisknutí tlačítka. Náhodnost tedy určovaly intervaly mezi jednotlivými stisky tlačítka. To se ale ukázalo jako nedostačující. Proto byl vytvořen speciální obvod pro generování pseudonáhodných čísel.

Základ tohoto obvodu je posuvný registr s lineární zpětnou vazbou. Vstup tohoto posuvného registru je XNOR jeho posledních dvou bitů. Tento signál je ale negován, pokud jsou všechny bity registru v logické jedničce. Kdyby to nastalo, registr by se zasekl v jednom stavu. Pro dosažení náhodnosti má tento modul vlastní generátor hodin, který generuje frekvenci okolo 10 kHz. To ve výsledku znamená, že je z kódu prakticky nemožné předpovědět, jaké číslo tento obvod vygeneruje.

Generátor hodin je vytvořen pomocí časovače NE555 a posuvný registr je realizován IO 74164. Logiku pro ošetření případného zaseknutí v jedničce zajišťují hradla AND a NAND realizované obvody 74LS08 a 74LS00, lineární zpětná vazba je realizovaná obvodem 74LS86. Modul se dá přepnout mezi módy generování pseudonáhodných 4bitových nebo 8bitových čísel.

8.2 Výstupní

8.2.1 Znakový displej

Pro výstupní modul byl zvolen znakový displej LCD 16x2. Výstupní registr OPA je připojen na řídící signály tohoto obvodu EN, RW, a RS. Registr OPB je připojen na datovou sběrnici D0-D7.

Znakový displej se dá poměrně jednoduše ovládat kódem. Pracuje s ASCII hodnotami, veškerý text, co se do něj pošle, musí být správně zakódován.

Obr. 8.1. Znakový displej 16x2

9 INSTRUKČNÍ SADA

9.1 Přesunové instrukce

Přesunové instrukce přesunují data z jednoho místa do druhého. Dělí se na dvě skupiny.

9.1.1 Paměť – registr

Instrukce paměť-registr přesouvají data mezi pamětí a registry. Patří sem tyto instrukce:

- LDM <addr>, <R> – nahraje hodnotu uloženou v paměti na adrese <addr> do registru <R>
- STM <R>, <addr> – uloží hodnotu z registru <R> do paměti na adresu <addr>
- LDM $<X>, <Y> – nahraje hodnotu uloženou v paměti na adrese v registru <X> do registru <Y>
- STM <X>, $<Y> – uloží hodnotu z registru <X> do paměti na adresu v registru <Y>
- LDR <R>, <value> – nahraje do registru <R> hodnotu <value>
- PUSH <R> – uloží hodnotu z registru <R> na vrchol zásobníku
- POP <R> – nahraje hodnotu z vrcholu zásobníku do registru <R>
- PEEK <R> – nahraje hodnotu z vrcholu zásobníku do registru <R>, ale nemění stav ukazatele zásobníku SP

9.1.2 Registr – registr

Instrukce registr-registr přesouvají data mezi jednotlivými registry. Jedná se o jedny z nejjednodušších operací. Patří sem tyto instrukce:

- MOV <X>, <Y> – přesune hodnotu z registru <X> do registru <Y>
- SWP <X>, <Y> – vymění hodnoty v registrech <X> a <Y>

9.2 Aritmeticko-logické instrukce

Aritmeticko-logické instrukce tvoří skoro polovinu instrukční sady. Provádějí aritmetické nebo logické operace, a jako jediné instrukce nastavují příznaky. Jako argument se pro jednoduchost uvádí vždy jen druhý operand, první operand se automaticky bere jako registr A.

Instrukce, které pracují se dvěma operandy, mohou za druhý operand určit registry B, C a D, hodnotu na dané adrese v paměti, nebo přímou hodnotu. V popisu operací níže jsou tyto možnosti označeny jako <OP>. Výsledek těchto operací se ukládá do registru A. Ten není možné zvolit jako druhý operand, protože ve většině případů to nemá smysl. Například instrukce AND A, která by měla do registru A uložit výsledek operace A AND A, by stav registru A vůbec nezměnila. Proto byly tyto možnosti vypuštěny. Instrukce se dvěma operandy jsou tyto:

- ADD <OP> – přičte hodnotu <OP> k registru A
- SUB <OP> – odečte hodnotu <OP> od registru A
- ADC <OP> – přičte hodnotu <OP> a CF k registru A
- SBC <OP> – odečte hodnotu <OP> a CF od registru A
- AND <OP> – provede logický součin registru A a hodnoty <OP>
- OR <OP> – provede logický součet registru A a hodnoty <OP>
- NAND <OP> – provede negovaný logický součin registru A a hodnoty <OP>
- NOR <OP> – provede negovaný logický součet registru A a hodnoty <OP>
- XOR <OP> – provede exkluzivní disjunkci registru A a hodnoty <OP>
- XNOR <OP> – provede negovanou exkluzivní disjunkci registru A a hodnoty <OP>

Další operace pracují pouze s jedním operandem. Může jím být registr A, B, C i D, nebo hodnota uložená v paměti na dané adrese. Tyto možnosti jsou v následujícím seznamu opět označeny jako <OP>. Instrukce pracující s jedním operandem jsou tyto:

- INC <OP> – inkrementuje hodnotu <OP>
- DEC <OP> – dekrementuje hodnotu <OP>
- SHL <OP> – bitový posuv hodnoty <OP> doleva, do uvolněného bitu se dá 0
- SHR <OP> – bitový posuv hodnoty <OP> doprava, do uvolněného bitu se dá 0
- ROL <OP> – bitový posuv hodnoty <OP> doleva, do uvolněného bitu se dá nejvyšší bit
- ROR <OP> – bitový posuv hodnoty <OP> doprava, do uvolněného bitu se dá nejnižší bit
- INV <OP> – neguje hodnotu <OP>
- TCM <OP> – vytvoří dvojkový doplněk hodnoty <OP> (neguje a přičte 1)

Kromě těchto operací ještě ALU vykonává jednu speciální instrukci. Je jí instrukce CMP od anglického CoMPare. Používá se pro porovnávaní čísel. Určené číslo B se odečte od čísla v registru A, a výsledek se nikam neuloží. Během odčítání se ale nastaví příznaky, ze kterých lze vyčíst tyto informace:

- Pokud je nastaven příznak nuly, porovnávaná čísla se rovnají
- Pokud je nastaven příznak přenosu, číslo v registru A je menší než číslo B
- Pokud tyto příznaky nastaveny nejsou, číslo v registru A je větší než číslo B

Příznaky se pak dají použít pomocí řídících instrukcí.

9.3 Řídící instrukce

Řídící instrukce jsou takové instrukce, které řídí průběh programu. Bez nich by počítač vykonával instrukce pouze tak, jak jdou v paměti za sebou. To může být v jistých případech žádoucí, většinou ale platí, že počítač v paměti různě skáče. Schopnost provádět tyto skoky umožňuje psaní mnohem složitějších programů. Řídící instrukce se dělí podle toho, zda se skok provede vždy, nebo jen za určitých podmínek.

Všechny řídící instrukce pro vlastní provedení skoků pracují s programovým čítačem.

9.3.1 Nepodmíněné

Nepodmíněné řídící instrukce se provedou vždy. Patří sem tři instrukce:

- JUMP <ADDR> – skok
- CALL <ADDR> – zavolání funkce
- RET – návrat z funkce

Nejjednodušší z nich je skok. Jako argument instrukce se udává adresa, na jakou má počítač skočit. Při vykonávání instrukce se tato adresa přesune do programového čítače.

Zavolání funkce je složitější. Stejně jako u klasického skoku se jako argument udává adresa. Nejdříve je ale potřeba uložit návratovou adresu na vrchol zásobníku, a až poté provést skok na určenou adresu.

Samotné provedení funguje následovně: po nahrání instrukce z paměti bude programový čítač ukazovat na adresu, kde je uložena adresa pro skok. Hodnota se tedy z programového čítače přesune do ukazatele hlavní paměti. V ten moment je možné z paměti přečíst cílovou adresu skoku. Ta se přesune do pomocného registru TEMP. Během toho se

inkrementuje programový čítač. To znamená, že teď ukazuje na adresu další instrukce. Tato adresa se uloží na vrchol zásobníku. Až poté je možné přesunout adresu funkce z pomocného registru TEMP do programového čítače pro dokončení skoku.

Třetí instrukcí je návrat z funkce. Tato instrukce je podstatně jednodušší než zavolání funkce. Pro návrat totiž stačí pouze přesunout návratovou adresu z vrcholu zásobníku do programového čítače.

9.3.2 Podmíněné

Každá ze tří nepodmíněných řídících instrukcí má celkem osm podmíněných verzí. Počítač obsahuje celkem čtyři příznaky. Pro každý příznak jsou dvě podmínky, a to, jestli je příznak aktivní, a jestli je příznak neaktivní. Jednotlivé instrukce byly detailněji popsány výše. Pokud jejich podmínka platí, provedou to, co mají. Pokud ne, jsou přeskočeny, a pokračuje se další instrukcí.

Podmíněné řídící instrukce jsou následující:

- JMC <ADDR> - skok, pokud je aktivní příznak přenosu
- JMZ <ADDR> – skok, pokud je aktivní příznak nuly
- JMN <ADDR> – skok, pokud je aktivní příznak znaménka
- JMP <ADDR> – skok, pokud je aktivní příznak parity
- JMNC <ADDR> – skok, pokud není aktivní příznak přenosu
- JMNZ <ADDR> – skok, pokud není aktivní příznak nuly
- JMNN <ADDR> – skok, pokud není aktivní příznak znaménka
- JMNP <ADDR> – skok, pokud není aktivní příznak parity
- CLLC <ADDR> – zavolání funkce, pokud je aktivní příznak přenosu
- CLLZ <ADDR> – zavolání funkce, pokud je aktivní příznak nuly
- CLLN <ADDR> – zavolání funkce, pokud je aktivní příznak znaménka
- CLLP <ADDR> – zavolání funkce, pokud je aktivní příznak parity
- CLLNC <ADDR> – zavolání funkce, pokud není aktivní příznak přenosu
- CLLNZ <ADDR> – zavolání funkce, pokud není aktivní příznak nuly
- CLLNN <ADDR> – zavolání funkce, pokud není aktivní příznak znaménka
- CLLNP <ADDR> – zavolání funkce, pokud není aktivní příznak parity
- RTC – návrat z funkce, pokud je aktivní příznak přenosu
- RTZ – návrat z funkce, pokud je aktivní příznak nuly

- RTN – návrat z funkce, pokud je aktivní příznak znaménka
- RTP – návrat z funkce, pokud je aktivní příznak parity
- RTNC – návrat z funkce, pokud není aktivní příznak přenosu
- RTNZ – návrat z funkce, pokud není aktivní příznak nuly
- RTNN – návrat z funkce, pokud není aktivní příznak znaménka
- RTNP – návrat z funkce, pokud není aktivní příznak parity

9.4 Ostatní instrukce

Zde jsou zařazeny instrukce, které nesplňují požadavky ostatních skupin. Patří sem dvě instrukce:

- NOP – žádná operace, používána pro zdržení nebo pro výplň nevyužitého prostoru v paměti
- HLT – zastavení počítače

10 PROGRAMOVÁNÍ POČÍTAČE

10.1 Jazyk symbolických adres – Assembly

Počítač je schopen vykonávat pouze strojový kód. Psát ho ručně by ale bylo příliš časově náročné a krkolomné. Proto byl vytvořen jazyk symbolických adres. JSA se většinou nepřesně označuje slovem „Assembler“, přesnější je však „Assembly“. Jedná se o reprezentaci instrukcí počítače pomocí mnemonik, tj. mnemotechnických zkratek. Tyto zkratky a jejich příslušné instrukce byly popsány v kapitole Instrukční sada.

Uživatel napíše program v tomto jazyce. Pro jazyk nebyla vytvářena žádná speciální přípona, programy jsou tedy klasické textové soubory. Tento program se dále přeloží na strojový kód pomocí překladače, tzv. assembleru.

10.2 Python assembler

Program pro překládání zdrojového kódu JSA na strojový kód pro procesor se obecně nazývá překladač, anglicky assembler. Pro tento projekt byl napsán jednoduchý překladač v jazyce Python. Jedná se o konzolovou aplikaci. Do té uživatel napíše jméno textového souboru se zdrojovým kódem v JSA. Překladač následně pomocí několika kroků program přeloží do strojového kódu. Kroky jsou to následující:

1. Textový soubor s kódem se rozdělí na řádky do seznamu řádků, prázdné řádky a komentáře jsou vynechány
2. Jednotlivé řádky se dále rozdělí následovně:
    a. Pokud řádek obsahuje instrukci, rozdělí se na mnemoniku a argumenty, dále se podle instrukce určí velikost instrukce v paměti
    b. Pokud řádek obsahuje návěští, rozdělí se řádek na název návěští a velikost 0
    c. Pokud řádek obsahuje definici konstanty, uloží se hodnota a název konstanty do speciálního seznamu a řádek se vynechá
3. Pomocí vzniklého seznamu instrukcí díky hodnoty velikosti instrukce v každé položce se vypočítají skutečné adresy návěští
4. Pokud instrukce obsahuje návěští nebo konstantu, vymění se textový argument instrukce za skutečnou číselnou hodnotu vypočítanou v předchozím kroku
5. Takto upravený seznam instrukcí je pomocí slovníku mnemonik přeložen na strojový kód, který je zapsán do souboru *output.bin*

6. Pokud je program menší než 256 bajtů, zbylé místo ve výstupním souboru je zaplněno nulami
7. Jako poslední je vytvořen textový soubor *output_loader.txt*, který formátuje strojový kód tak, aby se dal zkopírovat do softwaru Loaderu

Při spuštění aplikace *assembler.py* uživatel zadá název souboru (bez .txt). Následně se uživateli podle nastavených hodnot na začátku kódu může zobrazit seznam instrukcí s velikostmi a hodnotami, a přehled výstupu v hexadecimální tabulce. Tyto možnosti se dají vypnout v kódu na řádcích 5 a 6, nastavením proměnných PRINT_INSTR_ARRAY a PRINT_OUTPUT_HEX na True nebo False.

```
00000000: 68 38 DE 22 68 0E DE 22  68 06 DE 22 68 01 DE 22  h8."h.."h.."h.."
00000010: 66 29 1E C8 00 CB 21 47  67 20 67 A0 67 20 9E C9  f)....!Gg g.g ..
00000020: 12 FF 67 00 67 80 67 00  E7 48 65 6C 6C 6F 2C 20  ..g.g.g..Hello,
00000030: 77 6F 72 6C 64 21 00 00  00 00 00 00 00 00 00 00  world!..........
00000040: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000050: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000060: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000070: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000080: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000090: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000000A0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000000B0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000000C0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000000D0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000000E0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000000F0: 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
```

Obr. 10.1. Ukázka strojového kódu počítače

Pro psaní kódu v tomto JSA platí několik pravidel, některá slouží pro interní fungování překladače, některá pro lepší čitelnost:

- Při používání přímých číselných hodnot:
  - `#` značí číslo
  - `$` značí adresu
  - `0x` značí číslo v šestnáctkové soustavě
  - `0b` značí číslo v dvojkové soustavě
- Návěští se neodsazuje, definuje se jako `nazev_navesti:`
- Instrukce se odsazují většinou o dvě mezery, mnemoniky se musí zapsat malými písmeny a názvy registrů velkými:
  - `ldm adresa, B`
- Definice konstant se neodsazuje, značí se názvem, rovnítkem a hodnotou:
  - `konstanta = #0xFF`
- Číselné proměnné se definují návěštím a hodnotou:

    - `ciselna_promenna: #0x01`
- Textové proměnné se definují návěštím, typem textu, a textem ve dvojitých uvozovkách:
    - `textova_promenna: asciiz "text"`
    - ascii = text
    - asciiz = text v paměti ukončený nulou
- Komentář se označuje středníkem, platí pro část řádku za středníkem

Důležité je si uvědomit, že proměnné označené návěštím se samostatně ukládají do paměti počítače, zatímco konstanty označené rovnítkem se do paměti ukládají pouze jako argumenty instrukcí. Dále pokud je návěští použito s instrukcí, která očekává číselnou hodnotu (například instrukce LDR), jako argument instrukce se použije adresa daného návěští. To lze využít např. pro inicializaci ukazatele do paměti, pomocí instrukce LDR se do určeného registru dá nahrát adresa návěští.

Tento překladač byl vytvořen během jedné noci jako rychlá pomůcka pro programování počítače, a tak není dostatečně vyspělý. Jedná se o první verzi s mnoha nedostatky. Překládané programy musí být ve složce *programs*, samotný překladač není zcela odladěn. Také překladač neumí pracovat s výrazy. Byl již vytvořen prototyp kódu zpracovávajícího výrazy, ale ještě nebyl do překladače zakomponován.

10.3 Arduino loader – Software

Výstup z překladače jazyka symbolických adres se poté uloží do programu Loaderu na příslušné místo. Celkem se do něj může uložit až 16 různých programů. Ukládají se do dvourozměrného pole definovaného na začátku kódu. V samotném kódu je pro přehlednost jeden řádek tohoto pole jeden program.

```
/* PROGRAMS AVAILABLE
 * 0000 - HELLO WORLD (LCD)
 * 0001 - JUMP GAME (LCD, BUTTON)
 * 0010 - HAMBURGER (LCD, BUTTON)
 * 0011 - BUTTON GAME (LCD, BUTTON, RNG)
 * 0100 - TEST
 */

const byte PROGRAMS[16][256] PROGMEM = {
  { 0x68, 0x38, 0xde, 0x22, 0x68, 0x0e, 0xde, 0x22, 0x68, 0x06, 0xde, 0x22, 0x68, 0x01, 0xde
  { 0x68, 0x38, 0xde, 0xe0, 0x68, 0x0c, 0xde, 0xe0, 0x68, 0x06, 0xde, 0xe0, 0x68, 0x01, 0xde
  { 0x68, 0x38, 0xde, 0x69, 0x68, 0x0e, 0xde, 0x69, 0x68, 0x06, 0xde, 0x69, 0x68, 0x01, 0xde
  { 0x68, 0x38, 0xde, 0x8f, 0x68, 0x0e, 0xde, 0x8f, 0x68, 0x06, 0xde, 0x8f, 0x68, 0x01, 0xde
  { 0x63, 0x79, 0x07, 0x76, 0x68, 0x38, 0xde, 0x6f, 0x68, 0x0e, 0xde, 0x6f, 0x68, 0x06, 0xde
  { 0x68, 0x38, 0xde, 0x22, 0x68, 0x0e, 0xde, 0x22, 0x68, 0x06, 0xde, 0x22, 0x68, 0x01, 0xde
  { 0x68, 0x38, 0xde, 0x22, 0x68, 0x0e, 0xde, 0x22, 0x68, 0x06, 0xde, 0x22, 0x68, 0x01, 0xde
  { 0x68, 0x38, 0xde, 0x22, 0x68, 0x0e, 0xde, 0x22, 0x68, 0x06, 0xde, 0x22, 0x68, 0x01, 0xde
  { 0x68, 0x38, 0xde, 0x22, 0x68, 0x0e, 0xde, 0x22, 0x68, 0x06, 0xde, 0x22, 0x68, 0x01, 0xde
  { 0x68, 0x38, 0xde, 0x22, 0x68, 0x0e, 0xde, 0x22, 0x68, 0x06, 0xde, 0x22, 0x68, 0x01, 0xde
  { 0x68, 0x38, 0xde, 0x22, 0x68, 0x0e, 0xde, 0x22, 0x68, 0x06, 0xde, 0x22, 0x68, 0x01, 0xde
  { 0x68, 0x38, 0xde, 0x22, 0x68, 0x0e, 0xde, 0x22, 0x68, 0x06, 0xde, 0x22, 0x68, 0x01, 0xde
  { 0x68, 0x38, 0xde, 0x22, 0x68, 0x0e, 0xde, 0x22, 0x68, 0x06, 0xde, 0x22, 0x68, 0x01, 0xde
  { 0x68, 0x38, 0xde, 0x22, 0x68, 0x0e, 0xde, 0x22, 0x68, 0x06, 0xde, 0x22, 0x68, 0x01, 0xde
  { 0x68, 0x38, 0xde, 0x22, 0x68, 0x0e, 0xde, 0x22, 0x68, 0x06, 0xde, 0x22, 0x68, 0x01, 0xde
  { 0x68, 0x38, 0xde, 0x22, 0x68, 0x0e, 0xde, 0x22, 0x68, 0x06, 0xde, 0x22, 0x68, 0x01, 0xde
};
```

Obr. 10.2. Dvourozměrné pole obsahující programy

Samotný program Loaderu obsahuje několik funkcí pro nastavování správných hodnot. Princip samotného programování je relativně jednoduchý. Nejprve se z přepínačů přečte, který z šestnácti programů se nahrává. Dále se pokračuje cyklem od 0 do 255, který v každé iteraci nastaví hodnotu ukazatele paměti na danou adresu, a poté do paměti na tuto adresu uloží položku ze seznamu právě nahrávaného programu. Když cyklus doběhne na konec, byl do paměti počítače nahrán celý program.

Aby toto bylo možné, Loader musí během nahrávání převzít kontrolu nad částí řadiče ovládající sběrnici, a hodinami počítače. Kvůli tomu je možné programy nahrávat pouze, pokud jsou hodiny počítače zastaveny.

Pro nahrání nových programů do Loaderu je potřeba mít nainstalované Arduino IDE. Arduino Nano, kterým je Loader realizován, má starý bootloader mikrokontroleru. Proto je mikrokontroler na desce Arduina označen nálepkou O. To je potřeba při výběru portu a zařízení zvolit. Dále je důležité, že Loader nesmí být připojen pomocí USB kabelu k žádnému zdroji, zatímco je ke svému zdroji připojen počítač. Tento stav vyústí v poškození jednoho IO, a je nutno ho v budoucnu odladit. Vždy tedy musí mít uživatel na paměti, že na desku projektu smí v jeden moment vést pouze jeden kabel napájení.

11 PŘÍKLADY PROGRAMŮ

11.1 Hello, world! LCD

Tento program je v Loaderu uložen pod číslem 0000. Kód vypadá následovně:

```
start:                              ; inicializace displeje
  ldr OPB, #0b00111000              ; 8-bit mode, 2 lines, 5x8 font
  call lcd_instruction              ; zavolání funkce lcd_instruction

  ldr OPB, #0b00001110              ; display on, cursor on, blink off
  call lcd_instruction

  ldr OPB, #0b00000110              ; increment and shift cursor
  call lcd_instruction

  ldr OPB, #0b00000001              ; clear display
  call lcd_instruction

  ldr D, message                    ; adresa návěští message -> reg. D

print_message_loop:
  ldm $D, A                         ; hodnota na adrese v reg. D -> reg. A

  cmp #0                            ; porovnání reg. A s nulou
  jmz end                           ; pokud je nula, program skočí na end

  mov A, OPB                        ; reg. A -> reg. OPB
  ldr OPA, #0b00100000              ; pulz hodin displeje, znak
  ldr OPA, #0b10100000
  ldr OPA, #0b00100000              ; písmeno se zobrazí na displeji

  inc D                             ; reg. D ukazuje na další adresu

  jump print_message_loop           ; skok na začátek cyklu

end:
  hlt                               ; zastavení počítače

lcd_instruction:
  ldr OPA, #0b00000000              ; pulz hodin displeje, instrukce
  ldr OPA, #0b10000000
  ldr OPA, #0b00000000
  ret                               ; návrat z funkce

message: asciiz "Hello, world! "    ; textová proměnná ASCII ukončená nulou
```

ZÁVĚR

Projekt byl úspěšně realizován. Výsledkem je funkční celek, který dokáže vykonávat programy napsané uživatelem. Díky zvolenému způsobu realizace při běhu počítače uživatel fyzicky vidí, jak a kde v daný moment tečou data, jak spolu tyto data pracují, a co znamenají. Tím se může naučit, jak fungují procesory a výpočetní systémy na úrovni číslicových obvodů.

Realizace projektu se neobešla bez řady různorodých komplikací. Jednalo se jak o komplikace softwarové, tak hardwarové. Příkladem může být zprovoznění programátoru pamětí EEPROM, nebo pokles napájecího napětí kvůli přechodovým odporům nepájivých polí. Několik problémů způsobilo špatné zapojení. Všechny problémy a chyby byly ale opraveny.

Práce byla velmi časově náročná, společně s teoretickou přípravou se časová náročnost pohybuje v nižších stovkách hodin. Každý propojovací drát bylo potřeba vyměřit, odizolovat, vytvarovat a otestovat. Takovýchto drátů bylo nutno vytvořit stovky. Kromě drátování do časové náročnosti patří i další aspekty práce – návrhy modulů, příprava řídících slov pro instrukce, návrh instrukční sady, pomocné softwarové práce (assembler, programátor pamětí, testery IO), atp.

Projekt kombinuje znalosti z více oborů informatiky a elektroniky – hardwarové zapojení úzce souvisí s elektronikou a číslicovou technikou. Příprava řídících slov, assembler, nebo automatické testy IO naopak vyžadují obsáhlé znalosti z programování. Pro psaní kódu pro hotový projekt je potřeba znát nízkoúrovňové programování.

Zároveň se nabízí možnost úprav do budoucna. Jedná se například o ošetření hazardních stavů napájení (přepětí, špatná polarita) o které se momentálně stará zdroj, nebo úprava logiky potřebné pro určení příznaku přenosu.

SEZNAM POUŽITÉ LITERATURY

[1] Co je to algoritmus – Základy informatiky pro střední školy. Server Popelka [online] [cit. 24. 02. 2023]. Dostupné z: https://popelka.ms.mff.cuni.cz/~lessner/mw/index.php/U%C4%8Debnice/Algoritmus/Co_je_to_algoritmus

[2] 1. Úvod algoritmus a programovací jazyky – Stránky k výuce informatiky. *Stránky k výuce informatiky – určené nejen pro studenty Gymnázia Vlašim* [online]. Copyright © 2023 [cit. 24. 02. 2023]. Dostupné z: http://www.ivt.mzf.cz/algoritmizace-a-programovani/uvod-do-algoritmu/1-uvod-algoritmus/

[3] Teorie algoritmů. itnetwork.cz - Učíme národ IT [online]. Copyright © 2023 itnetwork.cz. Veškerý obsah webu [cit. 24. 02. 2023]. Dostupné z: https://www.itnetwork.cz/algoritmy/teorie

[4] Turing Machine in TOC - GeeksforGeeks. GeeksforGeeks | A computer science portal for geeks [online] [cit. 24. 02. 2023]. Dostupné z: https://www.geeksforgeeks.org/turing-machine-in-toc/

[5] Wikimedia Commons, Přispěvatelé. „Maquina.png“, Wikimedia Commons. [online] [cit. 24. 02. 2023]. Dostupné z: https://commons.wikimedia.org/wiki/File:Maquina.png

[6] Wikipedie: Otevřená encyklopedie: Turingovská úplnost [online]. c2022 [citováno 24. 02. 2023]. Dostupné z: https://cs.wikipedia.org/w/index.php?title=Turingovsk%C3%A1_%C3%BAplnost

[7] HAPL, L. *Číslicové systémy a DCBLPy pro studenty informatiky*. Ostrava: Ostravská univerzita, 2020. [cit. 24. 02. 2023]

[8] Wikimedia Commons, Přispěvatelé. „Logic-gate-index.png“, Wikimedia Commons. [online] [cit. 24. 02. 2023]. Dostupné z: https://commons.wikimedia.org/wiki/File:Logic-gate-index.png

[9] Design of 2 to 4 Decoder Circuit, ElProCus - Electronic Projects for Engineering Students [online]. Copyright 2013 - 2023 © Elprocus [cit. 24. 02. 2023]. Dostupné z: https://www.elprocus.com/designing-of-2-to-4-line-decoder/

[10] Wikimedia Commons, Přispěvatelé. „SR Flip-flop Diagram.svg“, Wikimedia Commons. [online] [cit. 24. 02. 2023]. Dostupné z: https://commons.wikimedia.org/wiki/File:SR_Flip-flop_Diagram.svg

[11] Wikimedia Commons, Přispěvatelé. Flip flops, Wikimedia Commons. [online] [cit. 24. 02. 2023]. Dostupné z: https://commons.wikimedia.org/wiki/Flip-flops

[12] Wikimedia Commons, Přispěvatelé. „MOS_6502_die.jpg“, Wikimedia Commons. [online] [cit. 24. 02. 2023]. Dostupné z: https://commons.wikimedia.org/wiki/File:MOS_6502_die.jpg

[13] Wikipedie: Otevřená encyklopedie: Integrovaný obvod [online]. c2022 [citováno 24. 02. 2023]. Dostupné z: https://cs.wikipedia.org/w/index.php?title=Integrovan%C3%BD_obvod

[14] Wikimedia Commons, Přispěvatelé. „TexasInstruments_7400_chip,_view_and_element_placement.jpg“, Wikimedia Commons. [online] [cit. 24. 02. 2023]. Dostupné z: https://commons.wikimedia.org/wiki/File:TexasInstruments_7400_chip,_view_and_element_placement.jpg

[15] MikesArcade.com - 74LS157. [online] [cit. 24. 02. 2023]. Dostupné z: https://www.mikesarcade.com/cgi-bin/store.pl?sku=74LS157

[16] 74LS173 Datasheet. [online] [cit 24. 02. 2023]. Dostupné z: https://datasheetspdf.com/pdf-file/375470/Motorola/74LS173/1

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

IO	Integrovaný obvod
ALU	Arithmetic-logic unit, aritmeticko logická jednotka
REG	Registr
CF	Carry flag, příznak přenosu
ZF	Zero flag, příznak nuly
SF	Sign flag, příznak znaménka
PF	Parity flag, příznak parity
IPA	Input port A
OPA	Output port A
OPB	Output port B
TEMP	Temporary, dočasný
PC	Program counter, programový čítač
RP	RAM pointer, ukazatel paměti
SP	Stack pointer, ukazatel zásobníku
IR	Instruction register, registr instrukce
OE	Output enable
EN	Enable
NC	Not connected
RR	Register read
RW	Register write NEBO Read/Write
RS	Register select
JSA	Jazyk symbolických adres

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH