STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 10: Elektrotechnika, elektronika a telekomunikace

Pineapple ONE

Filip Szkandera
Olomoucký kraj

Olomouc 3.4.2021

STŘEDOŠKOLSKÁ ODBORNÁ ČINNOST

Obor č. 10: Elektrotechnika, elektronika a telekomunikace

Pineapple ONE

Pineapple ONE

Autoři: Filip Szkandera
Škola: Vyšší odborná škola a Střední průmyslová škola elektrotechnická, Božetěchova 3, 779 00 Olomouc
Kraj: Olomoucký kraj
Konzultant: Ing. Zuzana Veselá, Ing. Jan Vykydal

Olomouc 3.4.2021

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 Olomouci dne 3.4.2021 ……

Filip Szkandera

Poděkování

Chtěl bych poděkovat Ing. Janu Vykydalovi za cenné rady, odborný dohled a konzultace, díky kterým jsem tuto práci dovedl do zdárného konce.
Dále bych chtěl poděkovat Ing. Zuzaně Veselé, jako vedoucí práce.

ANOTACE

Tato práce se zabývá návrhem, simulací a výrobou RISC-V procesoru z elektronických součástek. Výsledkem práce je makrokontrolér integrující procesor, paměť programu, paměť dat, grafickou kartu a vstupně výstupní porty do věžovité struktury plošných spojů tvořené devíti moduly. Dále práce obsahuje demonstrační aplikaci jednoduchého shellu naprogramovaného v jazyce C, jenž je spustitelná na vytvořeném zařízení.

KLÍČOVÁ SLOVA

RISC, RISC-V, CPU, VGA, RV32I

ANNOTATION

This thesis deals with the design, simulation and making of a RISC-V based processor only out of descrete logic components. The final product is a macrocontroller that integrates a processor, program memory, data memory, graphics card and an input-output ports in a tower structure made of nine circuit boards. This thesis also describes a simple shell application programmed in a C language, that runs natively on this device.

KEYWORDS

RISC, RISC-V, CPU, VGA, RV32I

SZKANDERA, Filip. *PINEAPPLE ONE* . Olomouc, 2021, 70 stran. Středoškolská odborná činnost. Vyšší odborná škola a Střední průmyslová škola elektrotechnická. Vedoucí práce: Ing. Zuzana Veselá

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Procesory jsou nezbytnou součástí moderní elektroniky. Najdeme je nejenom v počítačích a mobilních telefonech, ale stále častěji také v malých elektronických zařízeních.

Funkce těchto procesorů mě vždy fascinovala, a proto jsem chtěl pochopit, jak vlastně fungují. V roce 2019 jsem si sestavil svůj první 8 bitový mikroprocesor, díky kterému jsem pochopil základní principy procesoru.

Cílem tohoto projektu je navázat na moje zkušenosti z prvního procesoru a sestavit nový, rychlejší a kompaktnější procesor s architekturou RISC-V. Dal jsem si také za cíl, že v projektu nesmějí být žádné komerční procesory, hradlová pole ani programovatelná logická pole, ale pouze jednoduchá a volně dostupná logická hradla a paměti, jelikož cílem není emulace procesoru na jiné platformě, ale jeho zhotovení.

V této práci bude čtenář nejprve seznámen s funkcí procesoru, dále budou vysvětleny rozdíly mezi redukovanou a komplexní instrukční sadou. Následuje seznámení s architekturou RISC-V a poté se již práce věnuje samotnému návrhu a realizaci hardwaru procesoru RV32I. Nakonec jsou diskutovány dosažené výsledky.

1 Co je to procesor

Procesor (CPU) je v dnešní době nedílnou součástí každého chytrého zařízení, jako například televizorů, laptopů, chytrých hodinek, mobilních telefonů apod.

Tato kapitola má za cíl vytvořit ve čtenáři, který je v této problematice laikem, představu o tom co procesor obecně je, k čemu je dobrý a jak funguje. Zkusím jej vysvětlit na analogii s kuchařem.

Kuchaře jistě vážený čtenář zná. Se špetkou představivosti si může představit, že vaří podle předepsaného receptu. V této analogii bude kuchař symbolizovat procesor a recept vykonávaný program. Teď jsi můžeme popsat jak takový recept vypadá. Je to seznam dílčích úkonů, které následují chronologicky za sebou v pořadí v jakém mají být vykonány, jejichž provedením kuchař uvaří požadované jídlo.

Program je stejně jako recept tvořen posloupností instrukcí. Instrukce v programu jsou základní operace které procesor vykonává. Kuchař například může v receptu najít instrukci: rozklepnout vejce. V programu určeném pro procesor se setkáme z jednoduššími instrukcemi například: k hodnotě v registru A přičti konstantu X.

Procesor čte program a vykonává jej instrukci po instrukci jako kuchař. Na rozdíl od receptů z kuchařky mohou být programy mnohem rafinovanější. Často obsahují větvení (pokud platí podmínka udělej A, pokud ne, udělej B), či cykly (skoč na adresu Z pokud je v registru A hodnota menší než 42).

Pokud dokáže procesor vykonat libovolnou algoritmizovatelnou úlohu, říkáme o něm že je Turingovsky kompletní.

1.1 Turingův stroj

Alan Mathinson Turing, anglický matematik, fyzik a kryptograf, přišel s teoretickým modelem univerzálního výpočetního zařízení, které dnes nese jeho jméno, odtud Turingův stroj [2]. Jde o stavový automat, který dokáže vykonat libovolný algoritmus.

Turingův stroj si můžeme představit, jako nekonečně dlouhou pásku, tvořenou datovými buňkami. Tato páska prochází hlavou, jenž umožňuje číst a zapisovat data z aktuální buňky. Hlavu je také možné nad páskou posouvat libovolným směrem. Díky těmto možnostem lze na Turingově stroji vykonávat zadaný program.

Publikování tohoto teoretického konceptu bylo důležitým milníkem, který umožnil vyvinout první procesory a na nich založené počítače. Reálné procesory se snaží zachovat si stejné možnosti, jakými disponuje Turingův stroj, avšak dosahují tím odlišným způsobem. Není například možné mít nekonečně dlouhou pásku, ale musíme pracovat s omezenou velikostí polovodičových pamětí.

1.2 Architektury procesorů

Architektura procesoru je souhrn vlastností, které umožňují rozdělovat procesory do skupin s podobnými rysy. Architektura může specifikovat například: počet a strukturu registrů, topologii vnitřních a vnějších sběrnic, nebo strukturu a způsob vykonávání podporovaných instrukcí. V dnešní době se nejčastěji setkáme s dělením na dvě hlavních procesorové architektury a to na RISC a CISC.

Architektura CISC (Complex instruction set computer) česky známá jako komplexní instrukční sada, obsahuje širokou paletu instrukcí, které kromě základních operací, jako: násobení a dělení, mnohdy pokrývají i specifické oblasti jako: práci s řetězci, násobení vektorů a matic, a další speciální instrukce závislé na konkrétním procesoru. Mnohé instrukce jsou v této architektuře redundantní a můžeme je nahradit sekvencí několika jiných instrukcí, dostupných na konkrétním CISC procesoru. Charakteristickým rysem této architektury je, že instrukce mohou mít různou délku, navíc vykonání každé instrukce může mít i různý počet strojových cyklů. Typickým zástupcem této architektury je například i386.

Procesory RISC (Reduced instruction set computer) česky známá jako redukovaná instrukční sada, obsahuje jen nejnutnější instrukce pro to, aby byl Turingovsky kompletní. Klasickým příkladem je například realizace násobení opakovaným přičítáním, či nahrazení instrukce `nop` (No Operation) operací, která nezmění po jejím dokončení stav procesoru. Instrukce mají stejnou délku a vykonání veškerých instrukcí v této architektuře trvá stejný počet strojových cyklů. Toho lze využít zejména pro jednoduchou implementaci zřetězeného zpracování instrukci (Pipelining). Typickým zástupcem této architektury je například architektura ARM Cortex M0.

2 Architektura RISC-V

Architektura RISC-V vznikla v roce 2010 na Kalifornské univerzitě v Berkeley za účelem vytvořit nový standard pro procesory k všeobecnému využití. Tuto architekturu následně zveřejnili s open-source licencí, která umožňuje každému vytvářet vlastní produkty bez nutnosti placení licenčních poplatků [3].

Architektura RISC-V se v dnešní době používá převážně u 32b nebo 64b procesorů, 128b procesory jsou také definované touto architekturou ale nejsou tak časté.

2.1 Instrukční sady

Základní instrukční sadou, kterou musí každý RISC-V procesor podporovat, je sada I (Integer). Další instrukční sady, které architektura RISC-V nabízí, jsou volitelné a závisí také na výsledném účelu konkrétního zařízení. Tyto přídavné sady definují další možné funkce procesoru. Přídavné sady jsou rozděleny podle typu funkcí, které definují.

Značení procesorů pak závisí na přídavných sadách, které podporují. Procesor může být například označen RV32IM, kde RV je zkratkou pro RISC-V, číslovka určuje, o jaký procesor se jedná, v toto případě 32 bitů. Na konci jsou pak vypsány všechny instrukční sady, které jsou podporovány, tedy sada I a sada M [5].

2.1.1 Instrukční sada I

Primárně definuje práci s čísly a rozložení registrové banky.

Tato sada definuje 32 registrů `x0` až `x31`, které jsou umístěny v registrové bance a dokáží uchovávat číselné hodnoty. Registr `x0` je jedinou výjimkou, kde tento registr je pevně připojený na hodnotu 0. Při zápisu do tohoto registru se nic nestane a při čtení vždy vrátí nulu.

Velikost těchto registrů záleží na architektuře konkrétního procesoru, může se tedy jednat o 32, 64 nebo 128 bitový registr. Pineapple ONE je 32 bitový procesor, proto se dále zaměřím pouze na 32 bitové sady.

Všech 32 registrů je pojmenováno podle způsobu jejich využití.

Tab. 2.1: Přehled registrů architektury RISC-V.

Registr	**Název**
`x0`	Zero
`x1`	Return address
`x2`	Stack pointer
`x3`	Ground pointer
`x4`	Thread Pointer
`x5`	Temporary 0
`x6`	Temporary 1
`x7`	Temporary 2
`x8`	Save 0
`x9`	Save 1
`x10`	Function argument / return value 0
`x11`	Function argument / return value 1
`x12`	Function argument / return value 2
`x13`	Function argument / return value 3
`x14`	Function argument / return value 4
`x15`	Function argument / return value 5
`x16`	Function argument / return value 6
`x17`	Function argument / return value 7
`x18`	Save 2
`x19`	Save 3
`x20`	Save 4
`x21`	Save 5
`x22`	Save 6
`x23`	Save 7
`x24`	Save 8
`x25`	Save 9
`x26`	Save 10
`x27`	Save 11
`x28`	Temporary 3
`x29`	Temporary 4
`x30`	Temporary 5
`x31`	Temporary 6

2.1.2 Další instrukční sady

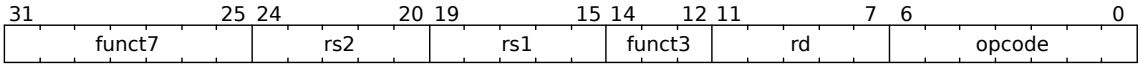
Architektura RISC-V definuje i mnoho dalších, nepovinných instrukčních sad. Mezi tyto sady patří například sada M (multiple) pro násobení a dělení, nebo sada F (floating point) pro umožnění práce s čísly s plovoucí desetinnou čárkou. Instrukční sady pro konkrétní procesor jsou vybírány podle jeho způsobu využití.

2.2 Instrukce

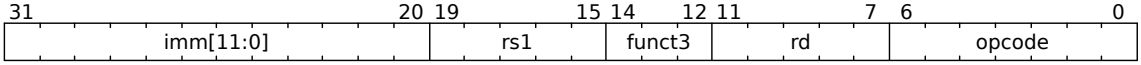
Počet instrukcí i jejich formátů v architektuře RISC-V se stále vyvíjí, ale například instrukční sady I (Integer), M (multiple) nebo F (floating point) jsou již pevně dané. Tato práce se bude věnovat převážně architektuře RV32I, protože je na jejím základě navrhnut procesor Pineapple ONE.

2.2.1 Formát instrukcí

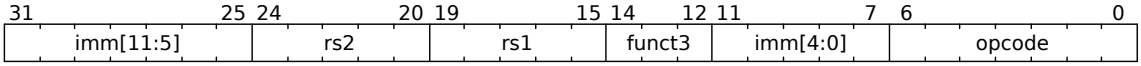
Každá instrukce v architektuře RV32I využívá jeden z 6 pevně daných formátů. Tyto formáty jsou navrženy tak, aby jednotlivé instrukce do sebe mohly zakódovat všechna potřebná data. Díky neměnnosti formátů pro ukládání instrukcí se zjednodušuje návrh a úpravy procesoru.



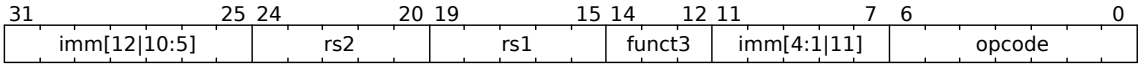
Obr. 2.1: Formát instrukce typu R.



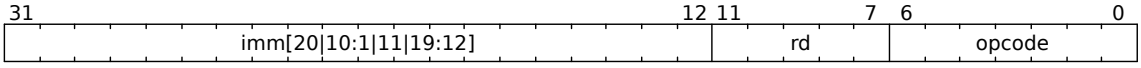
Obr. 2.2: Formát instrukce typu I.



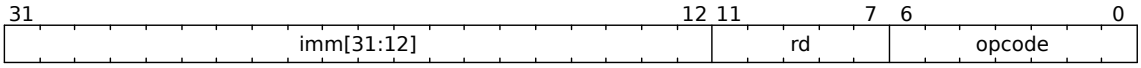
Obr. 2.3: Formát instrukce typu S.



Obr. 2.4: Formát instrukce typu SB.



Obr. 2.5: Formát instrukce typu U.



Obr. 2.6: Formát instrukce typu UJ.

- opcode (operační kód) – Jedná se o 7 bitovou hodnotu, podle které procesor identifikuje typ instrukce a vyhodnotí její následné zpracování.
- rd (destinační registr) – Jedná se o 5 bitovou hodnotu, která nese adresu registru, do které bude zapsán výsledek dané operace
- funct3 (funkce 3) – Upřesňuje instrukci v konkrétním formátu
- rs1 a rs2 (Zdrojový registr 1 a 2) – Adresa zdrojových registrů, se kterými se bude provádět operace
- imm (konstanta) – nese informaci o konstantní hodnotě

2.2.2 Aritmeticko-logické instrukce

Instrukce, které provádějí aritmeticko-logické operace mezi dvěma operandy. Výsledek těchto operací je vždy zapsán do uvedeného destinačního registru. Tyto instrukce jsou vypsány v tabulce 2.2 a využívají formát I, který zobrazuje obrázek 2.2 a R, jenž je na obrázku 2.1.

Tab. 2.2: Seznam aritmeticko-logických isntrukcí.

Název	**Popis**
`add`	add
`addi`	add immediate
`sub`	subtract
`sll`	shift left logical
`slli`	shift left logical immediate
`xor`	xor
`xori`	xor immediate
`srl`	shift right logical
`srli`	shift right logical immeadiate
`sra`	shift right aritmetic
`srai`	shift right aritmetic immediate
`or`	or
`ori`	or immediate
`and`	and
`andi`	and immediate

2.2.3 Instrukce pro práci s operační pamětí

Tyto instrukce načítají data z paměti nebo do ní data zapisují. Ve 32 bitovém procesoru můžeme pracovat s pamětí s délkou slova po 8, 16 nebo 32 bitech. Všechny tyto instrukce spadají do I formátu. Tyto instrukce jsou vypsány v tabulce 2.3 a využívají formát I, který zobrazuje obrázek 2.2 a S, jenž je na obrázku 2.3.

Tab. 2.3: Seznam instrukcí pro práci s operační pamětí.

Název	Popis
`lb`	load byte
`lbu`	load byte immediate
`lh`	load half
`lhu`	load half unsigned
`lw`	load word
`sb`	store byte
`sh`	store half
`sw`	store word

2.2.4 Instrukce pro nepodmíněné skoky

Všechny skoky v RISC-V jsou prováděny na relativní adresu. Instrukce `jal` provádí skok o hodnotu konstanty, kdežto instrukce `jalr` provádí skok o hodnotu registru. Tyto instrukce jsou vypsány v tabulce 2.4 a využívají formát I, který zobrazuje obrázek 2.2 a UJ, jenž je na obrázku 2.6.

Tab. 2.4: Seznam instrukcí nepodmíněných skoků.

Název	Popis
`jal`	jump and link
`jalr`	jump and link register

2.2.5 Instrukce pro podmíněné skoky

RISC-V nemá žádný příznakový registr, podle kterého by mohl vyhodnotit, zda je podmínka pro podmíněný skok splněna nebo ne. Místo toho se porovnávání provádí ve stejné instrukci, kde ihned po porovnání dvou operandů je proveden skok či nikoliv. Tyto instrukce jsou vypsány v tabulce 2.5 a využívají formát SB, který zobrazuje obrázek 2.4.

Tab. 2.5: Seznam instrukcí podmíněných skoků.

Název	Popis
`beq`	branch (if) equal
`bne`	branch (if) not equal
`blt`	branch (if) less than
`bge`	branch (if) greater (or) equal
`bltu`	branch (if) less than unsigned
`bgeu`	branch (if) grater (or) equal unsigned

2.2.6 Instrukce pro načítání vyšších konstant

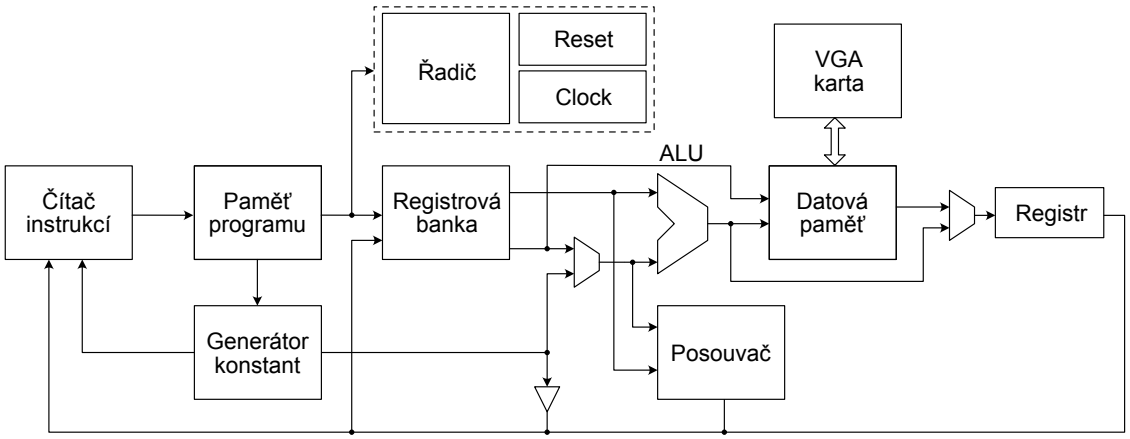
Načtení 32 bitové konstanty není možné, kvůli limitům pevně daného formátu instrukcí. Tento problém lze vyřešit pomocí dvou instrukcí. K načtení nultého až jedenáctého bitu slouží instrukce `addi` a k načtení dvanáctého až třicátého prvního bitu slouží instrukce `lui`. Další možností k načtení velké konstanty je instrukce `auipc`, která provede součet aktuální hodnoty v čítači instrukcí s konstantou bitově posunutou doleva o 12b a výsledek uloží do zvoleného registru. Tyto instrukce jsou vypsány v tabulce 2.6 a využívají formát UJ, který zobrazuje obrázek 2.6.

Tab. 2.6: Seznam instrukcí pro načítání vyšších konstant.

Název	Popis
`lui`	load upper immediate
`auipc`	add upper immediate (to) program counter

3 Procesor Pineapple ONE

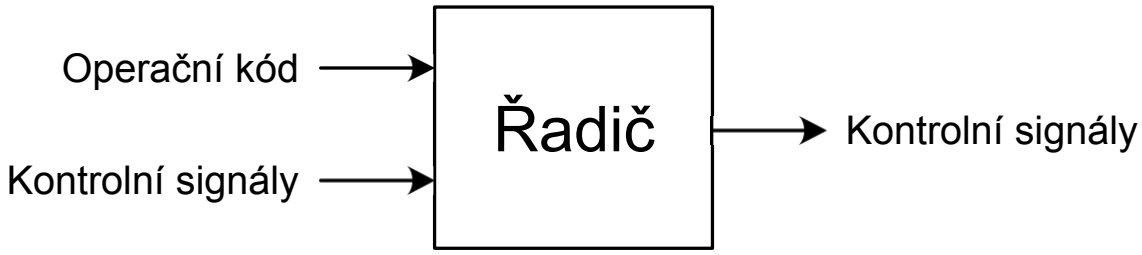
Procesor Pineapple ONE je 32b procesor založený na RISC-V architektuře. Je plně kompatibilní s instrukční sadou I, nese tedy označí RV32I.



Obr. 3.1: Blokový diagram procesoru Pineapple ONE.

3.1 Řadič

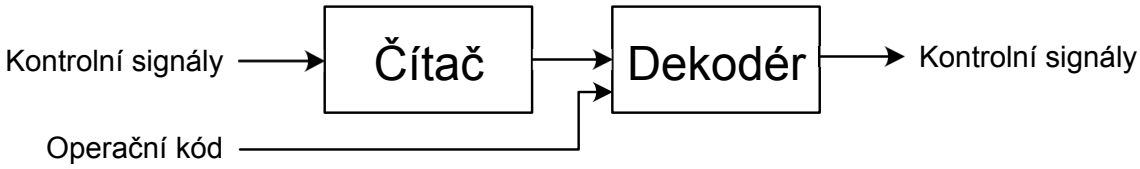
Řadič vykonává mikroprogram, který je složen z mikroinstrukcí, což jsou elementární operace ze kterých se skládají instrukce. Tyto mikroinstrukce řídí pomocí kontrolních signálů ostatní bloky tak, aby se provedla právě vykonávaná instrukce. Operační kód je prvních sedm počátečních bitů v každé instrukci nazývaný opcode, ze kterých je řadič schopen rozpoznat, o jaký mikroprogram se jedná a následně ho vykonat.



Obr. 3.2: Blokové zobrazení jednotky control unit.

Jednotka řadiče je sestavena z čítače a dekodéru. Čítač počítá od nuly do sedmi, kdy inkrementuje svou hodnotu vždy při sestupné hraně taktovacího signálu. Tímto způsobem je rozděleno provádění mikroprogramu na jednotlivé mikroinstrukce. Minimální počet mikroinstrukcí je jedna a maximální je osm. Výstup čítače poté společně s operačním kódem vstupují do dekodéru. Dekodér je sestaven ze tří EEPROM pamětí s označením 39SF010A, které v sobě uchovávají předprogramovanou tabulku s

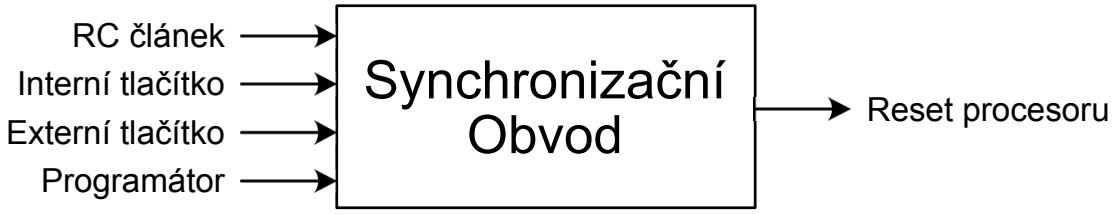
mikroinstrukcemi. Spojení těchto tří pamětí dohromady vznikne 24 řídících signálů, které mohou být využity pro ovládání jednotlivých částí procesoru. Každý signál má svůj vlastní název odvozený od jeho využití.



Obr. 3.3: Vnitřní struktura jednotky control unit.

3.2 Reset

Reset, nebo také nulování se stará o vynulování procesoru. Reset obvod se automaticky spustí při zapnutí procesoru, zaručí tedy správné vykonávání programu po spuštění bez nutnosti zásahu od uživatele. Resetovat celé zařízení je také možné pomocí resetovacího tlačítka, umístěného přímo na jádře procesoru a nebo tlačítkem vyvedeným na přední panel. Připojený programátor má také možnost procesor resetovat.



Obr. 3.4: Vnitřní struktura resetovací jednotky.

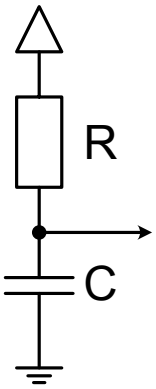
3.2.1 RC článek

RC článek slouží pro detekci připojení napájení do zařízení. Po detekci je celé zařízení resetováno.

Výpočet časové konstanty RC obvodu:

$$\tau = RC \tag{3.1}$$

Resetování RC článkem proběhne pouze po připojení napájení k zařízení, délka resetovacího impulsu proto nemá téměř žádný negativní vliv na použitelnosti celého zařízení. Zvoleny byly běžně dostupné součástky a to odpor R = 18 kΩ a kondenzátor C = 100nF. Výsledná časová konstanta vychází podle vzorce na T = 1.8 ms.



Obr. 3.5: Schématické zapojení RC článku.

3.2.2 Interní a externí tlačítko

Interní tlačítko je vestavěné přímo na samotném jádře procesoru, slouží zejména pro účely testování a není dobře přístupné běžnému uživateli.

Externí tlačítko je tlačítko vyvedené na přední panel celého zařízení a je tedy velmi dobře přístupné běžnému uživateli. Tlačítko je potom spojeno s jádrem procesoru příslušným kabelem. Toto tlačítko není nutné mít zapojeno pro správný chod procesoru.

Po stisknutí jakéhokoliv tlačítka se provede reset celého zařízení a nahraný program se začne vykonávat od začátku. Po stisknutí tlačítka se samotná operace provede v jednotkách milisekund (záleží na vybrané frekvenci taktovacího generátoru), není tedy nutné tlačítko držet po určitou dobu.

3.2.3 Programátor

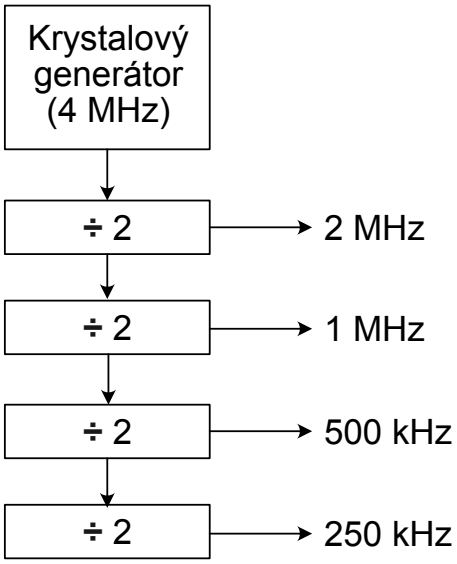
Jedná se o resetovací signál využívaný programátorem při programování zařízení. Po dokončení nahrávání programu je zařízení automaticky programátorem resetováno a není tedy nutný žádný zásah od uživatele. Resetovací signál pro programátor je digitálně ošetřen a programátor nemusí být pro správnou funkci zařízení připojen.

3.2.4 Synchronizační obvod

Tento obvod slučuje všechny vstupní resetovací signály. Sloučený signál je nadále synchronizován s nástupní hranou taktovacího signálu. Výstupní resetovací signál (reset procesoru) je rozveden do jednotlivých bloků procesoru.

3.3 Clock

Taktovací generátor procesoru tvoří 4 MHz krystalový generátor, jehož výstup je napojen na sérii logických dělících členů. Uživatel si tak může zvolit taktovací frekvenci mezi 2 MHz, 1 MHz, 500 kHz a 250 kHz pomocí zkratovací propojky. Doporučená frekvence pro správnou činnost procesoru je 500 kHz nebo 250 kHz.



Obr. 3.6: Blokové zobrazení dělícího modulu taktovacího generátoru.

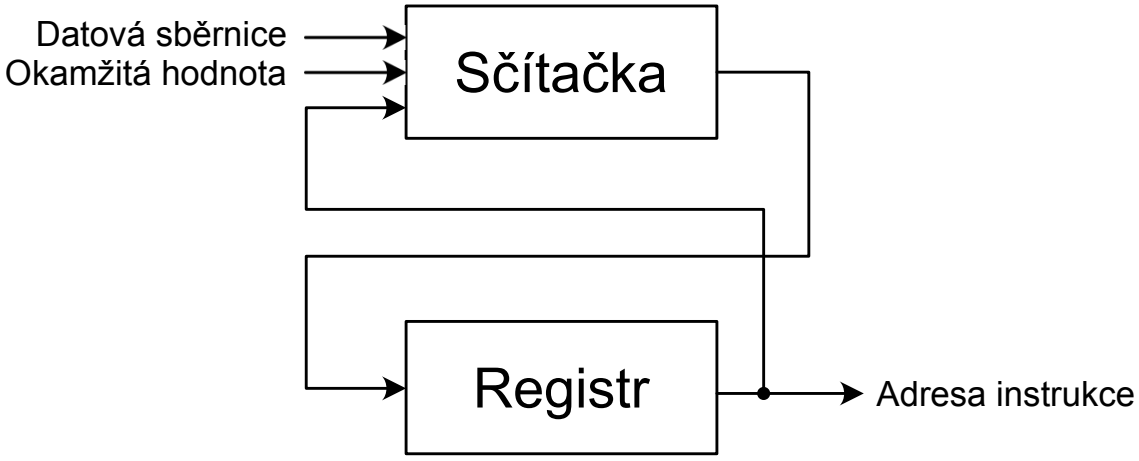
3.4 Čítač instrukcí

Tento modul udržuje adresu právě vykonávané instrukce a vypočítává adresu následující instrukce. V případě, že následující instrukce je instrukce skoková, tak ke své aktuální adrese přičte nebo odečte relativní adresu skoku. Po vykonání ostatních instrukcí přičte ke své stávající adrese číslo 4 pro adresování následující instrukce. Při resetování procesoru se jeho adresa nastaví na nulu.

Obr. 3.7: Blokové zobrazení jednotky control unit.

Čítač instrukcí je složen ze sčítačky a registru. Sčítačka je realizována pěti EEPROM pamětmi s označením 39SF010A. V nich je nahraná předvypočítaná tabulka

veškerých hodnot. Tabulka byla vytvořena pomocí programovacího jazyku Python. Registr je tvořen soustavou čtyř osmi bitových registrových pamětí s označením 74HCT574. Tyto registry udržují adresu právě vykonávané instrukce a při skoku na jinou instrukci uloží hodnotu vypočítanou sčítačkou. Kontrolní signály vstupující do tohoto modulu řídí funkci sčítačky a registru.

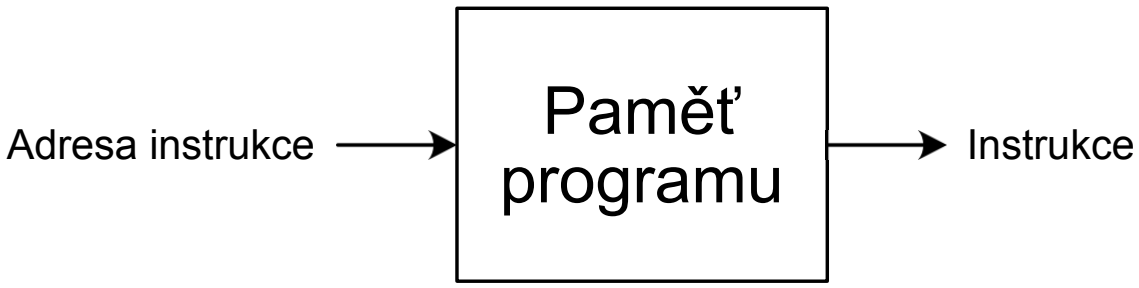


Obr. 3.8: Vnitřní uspořádání modulu čítače instrukcí.

3.5 Paměť programu

V paměti programu je uchovávaný nahraný program. Program se skládá z jednotlivých 32 bitových instrukcí, které jsou v této paměti uloženy a lze je jednotlivě adresovat pomocí adresy instrukce.

Paměť je složena ze čtyřech pamětí typu EEPROM s označením 39SF010A. Kombinace těchto čtyřech pamětí nám dává dohromady 512 kB prostoru pro program. Procesor je kompatibilní i s pamětmi s označením 39SF020A nebo 39SF040A, díky kterým by bylo možné využít až 4 MB prostoru pro program. Paměť EEPROM uchovává uložená data i po vypnutí zařízení, po opětovném spuštění se opět začne vykonávat bez nutnosti přeprogramování.

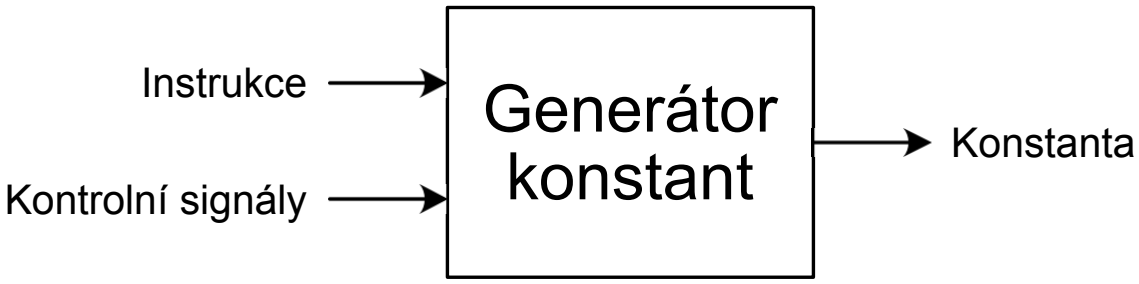


Obr. 3.9: Blokové zobrazení jednotky paměti programu.

Přeprogramování paměti programu je velmi snadný proces pro koncového uživatele. Uživatel může za pomocí připojeného programátoru kdykoliv začít proces přeprogramovávání paměťového bloku. Po dobu programování je činnost procesoru pozastavena a po jejím dokončení je procesor automaticky resetován a nový program je automaticky spuštěn.

3.6 Generátor konstant

Modul, který dekóduje konstantní hodnotu zakódovanou v právě vykonávané instrukci. Dekódovat konstantu je nutné ve formátech instrukcí I, S, SB, U a UJ. Vstup instrukce je přímo napojen na výstup paměti programu a pomocí kontrolních signálů vedoucích z řadiče je vybrán příslušný dekódovací formát.



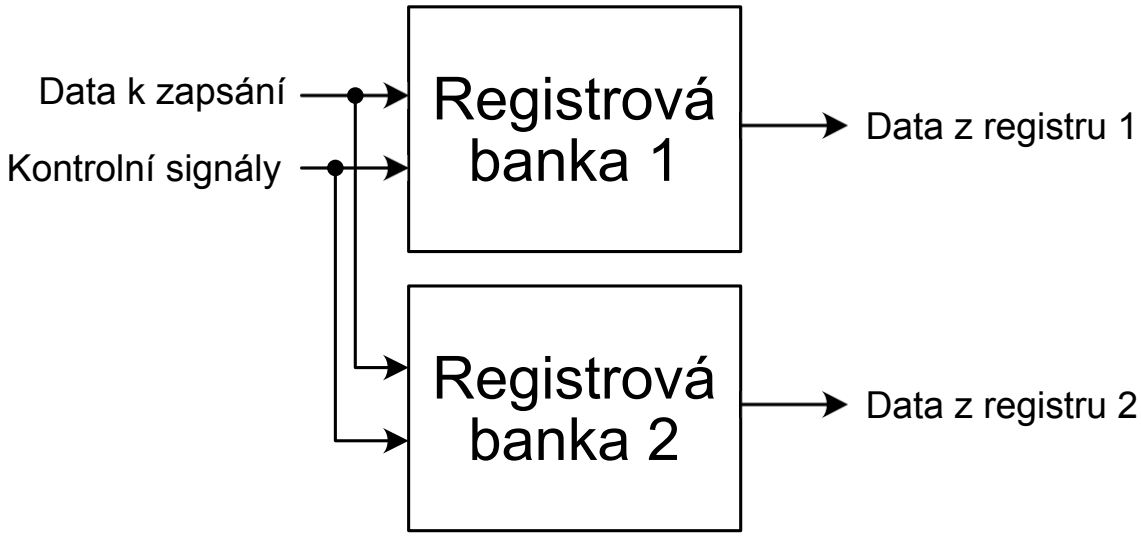
Obr. 3.10: Blokové zobrazení generátoru konstant.

3.7 Registrová banka

Registrová banka je místo, kde je uloženo všech 32 32-bitových registrů, které definuje I-formát. Pro správnou funkci ostatních bloků procesoru je nutná možnost číst až dva registry ve stejnou chvíli, zapisovat je ale možné pouze do jednoho registru v danou chvíli.

Obr. 3.11: Blokové zobrazení jednotky registrové banky.

Tento modul se skládá ze dvou separátních registrových bank, které mají společný přívod pro zapisovaná data. Díky tomuto kroku je možné číst až dva registry v jednu chvíli.

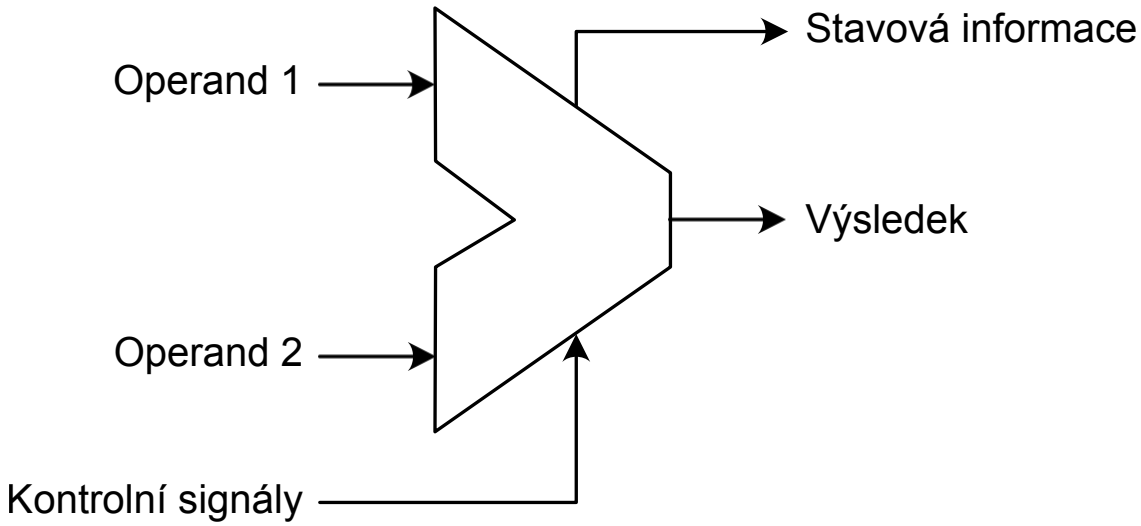


Obr. 3.12: Vnitřní uspořádání modulu registrové banky.

Každá z těchto registrových bank je složena ze dvou 16 bitových SRAM pamětí s označením IS65C1024AL-45T. Tyto paměti mohou v jednu chvíli pouze zapisovat nebo číst data, je tedy nutné mezi těmito funkcemi podle potřeby přepínat. Tuto funkci zajišťují kontrolní signály.

3.8 ALU

Aritmeticko-logická jednotka provádí aritmetické i logické operace s dvěma 32 bitovými operandy. Mimo tyto operace dokáže také při instrukci podmíněného skoku porovnat oba operandy a na tomto základě vyhodnotit, zda se uskuteční podmíněný skok nebo ne.



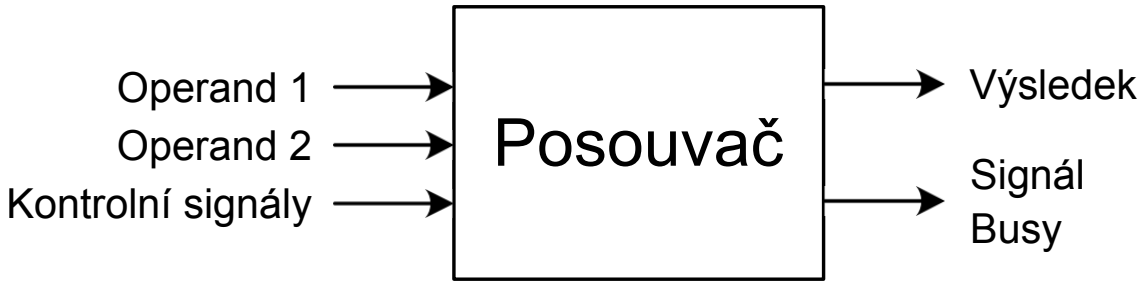
Obr. 3.13: Blokové zobrazení jednotky ALU.

Aritmeticko-logická jednotka je realizována na sedmi pamětech typu EEPROM, které jsou uspořádány jako sekvenční sčítačka. Tyto paměti byly předem naprogramovány tabulkou výsledků, která byla vygenerována pomocí programovacího jazyka Python. Zvolená architektura tohoto procesoru nepodporuje stavový registr pro ukládání informací o výpočtu, proto je tato stavová informace vedena přímo do příslušných modulů jako kontrolní signály. Kontrolní signály vstupující do tohoto modulu specifikují operaci, která se má provést.

3.9 Posouvač

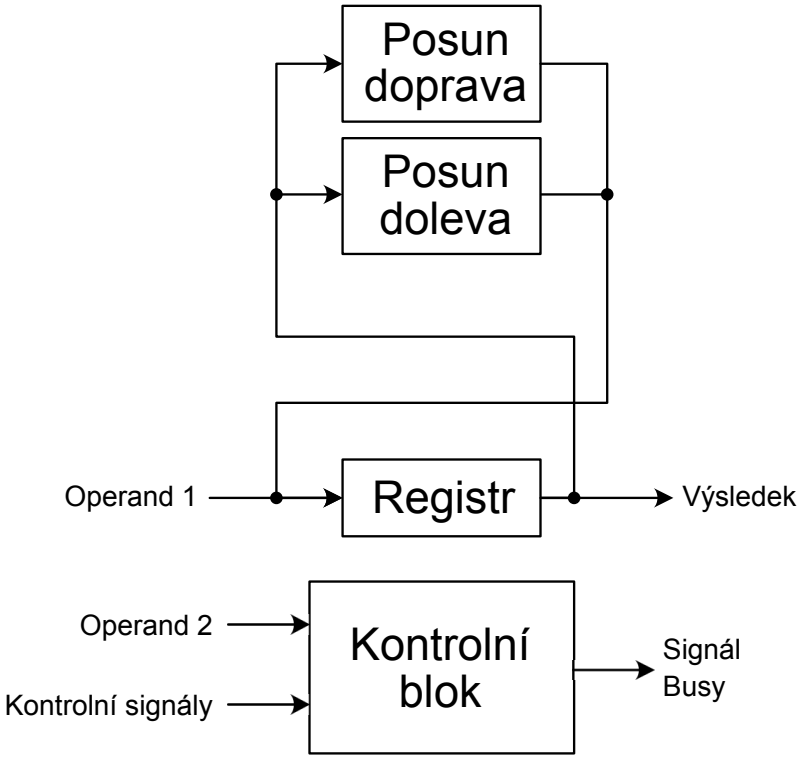
Posouvač slouží jako doplněk aritmeticko-logické jednotky. Účelem této jednotky je posouvat první operand o hodnotu druhého operandu, a to až o 32 pozic. Posuv může být doprava nebo doleva, logicky nebo aritmeticky.

Tento obvod dokáže za jeden takt procesoru posunout danou hodnotu pouze o jedno místo do libovolného směru. Při posuvu o více než jednu pozici tato jednotka pozastaví zbytek procesoru a bude opakovaně provádět stejnou operaci do doby, než bude číslo posunuto o požadovaný počet. Následně vrátí kontrolu hlavní části procesoru.



Obr. 3.14: Blokové zobrazení posouvací jednotky.

Posouvač je složena z kontrolního bloku, registru a posouvacích bloků. Posouvací bloky jsou dva, jeden posouvá hodnotu doleva a druhý doprava. Registr po daném posuvu uloží výslednou hodnotu. Kontrolní signály určují směr posunu, který se bude vykonávat. Kontrolní blok poté pozastaví zbytek procesoru, posune hodnotu daným směrem a uloží ji do registru. Tento cyklus se opakuje do doby, než je hodnota posunuta o požadovaný počet. Signál Busy pozastavuje zbytek procesoru.

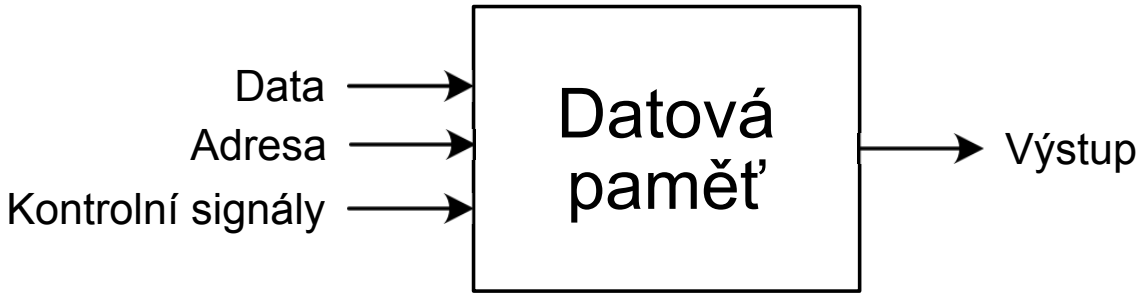


Obr. 3.15: Vnitřní uspořádání posunovací jednotky.

3.10 Datová paměť

Datová paměť umožňuje procesoru ukládat nebo číst data z jakékoliv adresy v daném adresovém rozsahu. Do datové paměti můžeme zapisovat po 32 bitových slovech, 16

bitových půlslovech, nebo osmi bitech, pomocí korespondujících instrukcí `sw`, `sh`, `sb`. Instrukce pro ukládání půlslova (`sh`) a instrukce pro ukládání bytu (`sb`) mají také své varianty, které dané číslo považují vždy za kladné a to jsou instrukce `shu` a `sbu`. Stejně je to u instrukcí načítající dat z paměti `lw`, `lh`, `lb`, `lhu`, `lbu`.



Obr. 3.16: Blokové zobrazení datové paměti.

Datová paměť se skládá ze tří základních částí a to paměť RAM, paměť V-RAM a speciální registry.

Tab. 3.1: Rozložení datové paměti na jednotlivé části společně s jejich adresovým rozsahem.

Adresový rozsah	Typ
0x00000000 ... 0x0001FFFF	RAM
0x00020000 ... 0x3FFFFFFF	Rezervovaný prostor
0x40000000 ... 0x400007FF	V-RAM
0x40000800 ... 0x7FFFFFFF	Rezervovaný prostor
0x80000000 ... 0x8000000F	Speciální Registry
0x80000010 ... 0xFFFFFFFF	Rezervovaný prostor

3.10.1 RAM

RAM (Random access memory), nebo také operační paměť je paměť, kam procesor ukládá různé hodnoty se kterýma bude pracovat. Tato paměť je realizována pomocí SRAM pamětí s označením x. S touto pamětí lze pracovat všemi způsoby zmíněnými výše.

3.10.2 V-RAM

V-RAM (Video random access memory) je část datové paměti, kam se ukládají data, která se zobrazí na displeji. S touto pamětí lze pracovat všemi způsoby zmíněnými výše. Jedná se o dual-port paměť, kde jeden port je přidělený procesoru a druhý VGA kartě.

3.10.3 Speciální registry

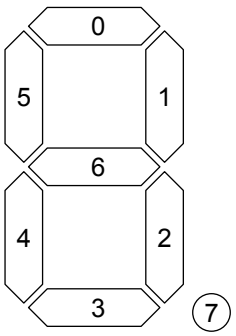
Speciální registry jsou 8 bitové registry, díky kterým může program procesoru řídit různé věci, jako například ovládání periferií, nebo zapínání a vypínání VGA výstupu.

Do registrů označených jako „výstup“ můžeme pouze zapisovat funkcí `sb`, z registrů označených jako „vstup“ můžeme pouze číst pomocí funkce `lb`.

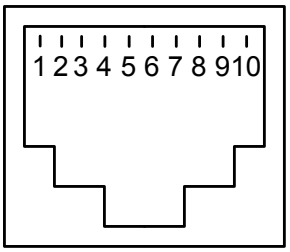
Tab. 3.2: Seznam vestavěných speciálních registrů s jejich adresami.

Adresa [hex]	Vstup/Výstup	Popis
0x80000000	Výstup	Displej
0x80000001	Výstup	VGA_Enable
0x80000004	Vstup	Port A
0x80000005	Vstup	Port B
0x80000006	Výstup	Port C
0x80000007	Výstup	Port D

- Displej – ovládání 7 segmentového displeje, který je umístěný na jádře procesoru
- VGA_Enable – řídí zapínání a vypínání VGA výstupu, při hodnotě `0` je výstup vypnutý, při hodnotě `1` je zapnutý.
- Port A & Port B – vstupní porty, které jsou vyvedeny na hlavní panel
- Port C & Port D – výstupní porty, které jsou vyvedeny na hlavní panel



Obr. 3.17: Zobrazení 7 segmentového displeje, kde číslo na každém segmentu reprezentuje index bitu v „Displej“ registru.



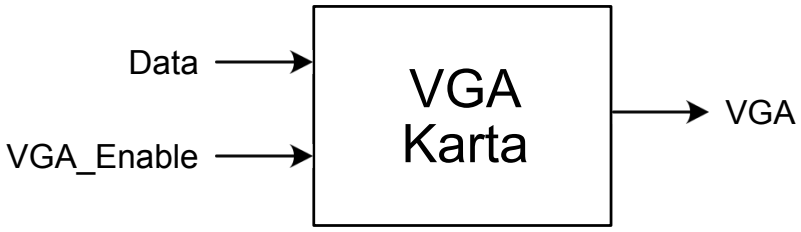
Obr. 3.18: Rozložení pinů v konektoru určeného pro jednotlivé porty. Pohled zepředu na zásuvku RJ50.

Tab. 3.3: Výpis pinů v konektoru RJ50 a jejich funkcí.

Pin	Název
1	5 V (VCC)
2	0 V (GND)
3	DATA BIT 0
4	DATA BIT 1
5	DATA BIT 2
6	DATA BIT 3
7	DATA BIT 4
8	DATA BIT 5
9	DATA BIT 6
10	DATA BIT 7

3.11 Karta VGA

Návrh této karty je inspirován článkem [1]. Tato karta generuje VGA signál, díky kterému je možné zařízení propojit s externím monitorem a zobrazovat na něm například text nebo obrázky. Monitor lze k zařízení propojit standardním VGA kabelem. Monitor po připojení zobrazí standardní rozlišení 800 x 600, to ale kvůli omezené kapacitě paměti V-RAM nebylo možné, proto je tento standard pouze využíván, výsledný obraz je ale pouze v rozlišení 200 x 150 px.



Obr. 3.19: Blokové zobrazení VGA karty.

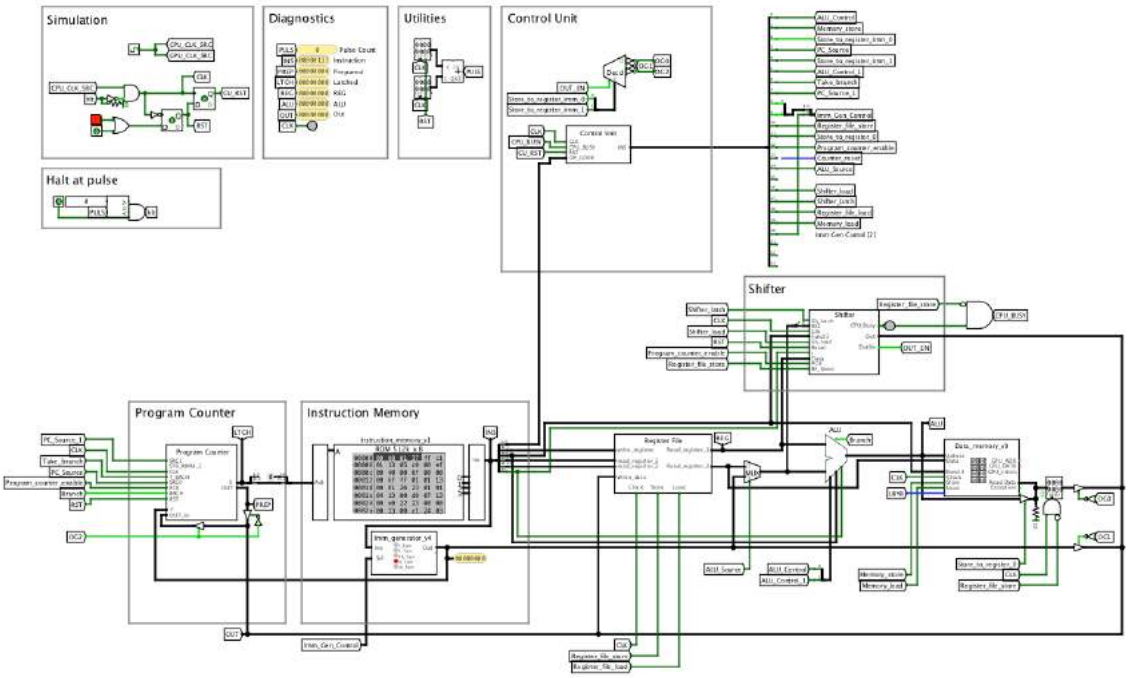
Výstupní signál je monochromatický (černo bílý) a lze jej zapnout nebo vypnout pomocí speciálního registru zvaného „VGA_Enable“. Po zapsání logické `1` do tohoto registru se VGA signál zapne, opačně po zapsání logické `0` se signál vypne.

4 Realizace procesoru Pineapple ONE

V této kapitole se budu věnovat jednotlivým krokům v návrhu a následné realizaci procesoru Pineapple ONE.

4.1 Simulace a návrh obvodů

Nejprve jsem se seznámil s jednotlivými bloky RISC-V v knize [4]. Pro ověření nastudované architektury jsem vytvořil její model, jenž je vyobrazen na obrázku 4.1, v programu Logisim-Evolution. Tento program obsahuje základní logická hradla, registry a paměti, ze kterých se mohou vytvářet logické obvody, ty poté mohu stimulovat s sledovat jak na to reagují výstupy. Pro simulaci analogových částí, jako například RC článku, jsem použil program LTspice.



Obr. 4.1: Simulace procesoru v programu Logisim-Evolution.

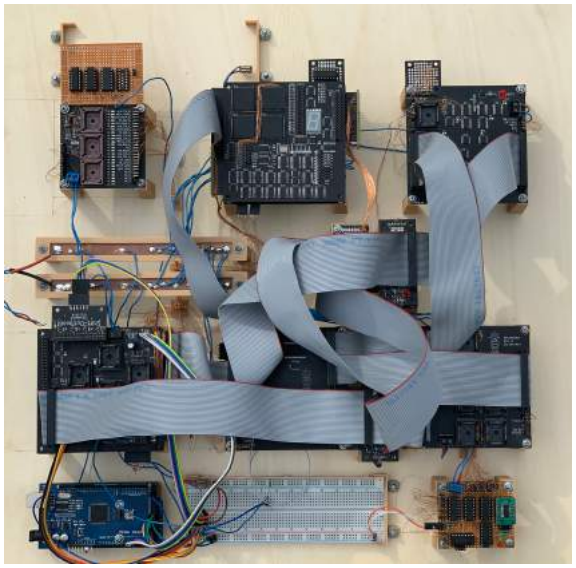
Po dokončení simulace a ověření její správnosti jsem jednotlivé bloky přepracoval tak, aby je bylo možné sestavit z volně dostupných součástek. Cílem tohoto projektu bylo sestavit procesor bez použití mikrokontrolérů, hradlových polí ani jiných programovatelných zařízení, kromě pamětí EEPROM. Celý procesor je proto navržen pouze za pomocí logických hradel, klopných obvodů a pamětí.

4.2 Návrh desek plošných spojů

Schémata a desky plošných spojů (PCB) jsem navrhl v programu Autodesk Eagle. Protože se jedná o mou samostatnou práci, kde všechny náklady na výrobu hradím, tak jsem bral velký ohled na výslednou cenu zařízení a snažil se vybírat co možná nejlevnější řešení bez znatelného kompromisu na kvalitě. Pro výrobu desek plošných spojů jsem zvolil zahraniční firmu JLCPCB, která v době realizace nabízela slevu na dvouvrstvé desky plošných spojů s rozměry do $(10 \cdot 10)$ cm^2 za cenu 2 USD za desku.

Desky plošných spojů kvůli těmto limitacím tedy mohli být maximálně dvouvrstvé a s maximální velikostí $(10 \cdot 10)$ cm^2. Celý procesor by se na jedno PCB o stanovených rozměrech nevešel. Proto jsem jeho části rozdělil na několik desek plošných spojů. Abych co nejvíce využil limitovaného prostoru, využil jsem převážně součástek s typem montáže SMD (Surface mount device).

Po dokončení návrhu PCB a jeho kontrole jsem je nechal vyrobit výše zmíněnou firmou a následně dodané desky osadil. Poté jsem z jednotlivých modulů sestavil prototyp procesoru Pineapple ONE, zvaný Pineapple ZERO.



Obr. 4.2: Obrázek procesoru Pineapple ZERO.

Jednotlivé části procesoru Pineapple ZERO jsem rozmístil na podložku tak, aby bylo co nejjednodušší propojit jeho části propojujícími vodiči. Díky tomuto prototypu jsem byl schopen odladit některé části procesoru a na tomto základě vytvořit finální verzi nazvanou Pineapple ONE. Po odladění byl Pineapple ZERO plně funkční.

Obr. 4.3: Obrázek procesoru Pineapple ONE bez vrchní části skříně

Při vytváření finální verze procesoru jsem použil schémata prototypů, ve kterých jsem opravil chyby. Návrh jsem optimalizoval tak, aby byl procesor tvořen co nejmenším počtem modulů PCB o rozměrech $(10 \cdot 10)$ cm^2, které se budou dát vršit na sebe díky chytrému uspořádání konektorů. Výsledná věž je tvořena devíti moduly a její výhodou je, že k propojení jednotlivých PCB není potřeba žádná dodatečná kabeláž.

4.3 Návrh počítačové skříně

Pro návrh počítačové skříně jsem zvolil program Autodesk Fusion 360. Tvar a velikost skříně je založena na rozměrech jádra procesoru. Pro snadné otevírání skříně je skříň rozdělena na vrchní a spodní část, které jsou spojeny trojicí šroubků a příslušných matek. Šroubky se nacházejí po obvodu ve spodní částí výrobku a po jejich vyjmutí je možné celou vrchní část tahem vzhůru vysunout.

Na přední panel byly umístěny veškeré potřebné konektory, přepínače a indikátory, které jsou propojeny s jádrem procesoru pomocí vodičů s příslušnými konektory. Pro dosažení hladkého povrchu na přední straně panelu byl panel vytištěn touto stranou na podložku, musel jsem ho vytisknout separátně a následně se spodní částí přišroubovat vestavěnými šroubky a příslušnými matkami.

Obr. 4.4: Návrh počítačové skříně v programu Autodedesk Fusion 360.

Počítačová skříň se skládá celkem ze sedmi částí, které jsem navrhl tak, aby je bylo možné snadno vytisknout na 3D tiskárně Prusa MK3. Zhotovené díly jsem následně sestavit do finální podoby.

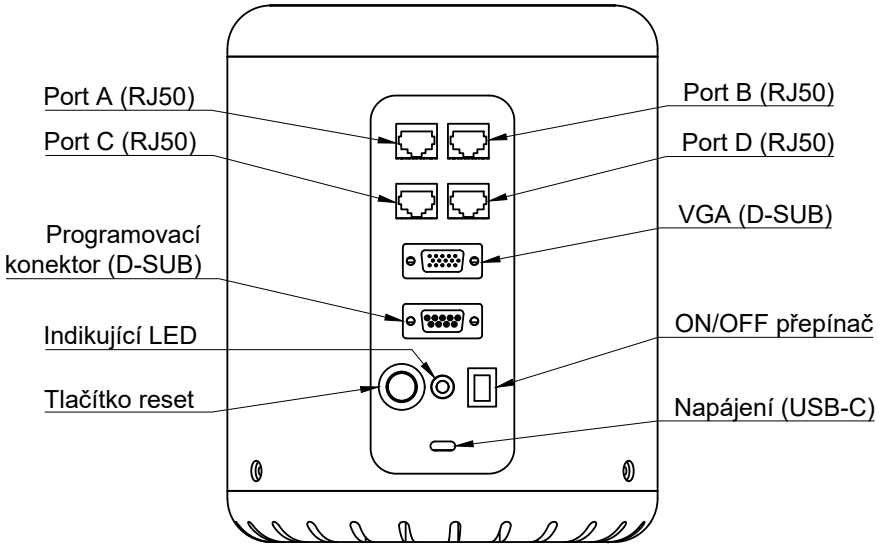
5 Výsledky

V této kapitole se budu věnovat finálnímu výrobku a uvedu ukázku možného využití procesoru. Dále také popíšu vstupně výstupní periferie a představím několik ukázkových externích zařízení.

Obr. 5.1: Finální verze zařízení.

5.1 Periferie

Periferie jsou zařízení, která se připojují k procesoru a tím dále rozšiřují jeho možnosti. Pineapple ONE je možné připojit prostřednictvím konektoru D-SUB k externímu monitoru, nebo lze využít vstupně výstupních portů k přenosu dat.

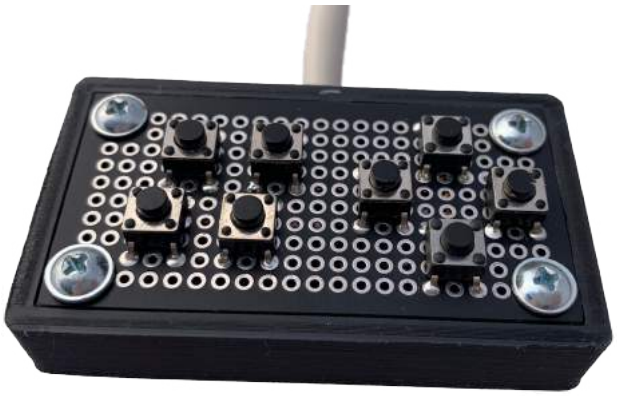


Obr. 5.2: Přední panel zařízení.

Jako ukázku jsem vytvořil několik modulů, které je možné připojit na různé porty procesoru. Patří k nim modul s tlačítky, které lze využít například jako ovladač, modul s LED diodami pro indikaci stavu a nebo modul pro připojení externí klávesnice.

5.1.1 Modul s tlačítky

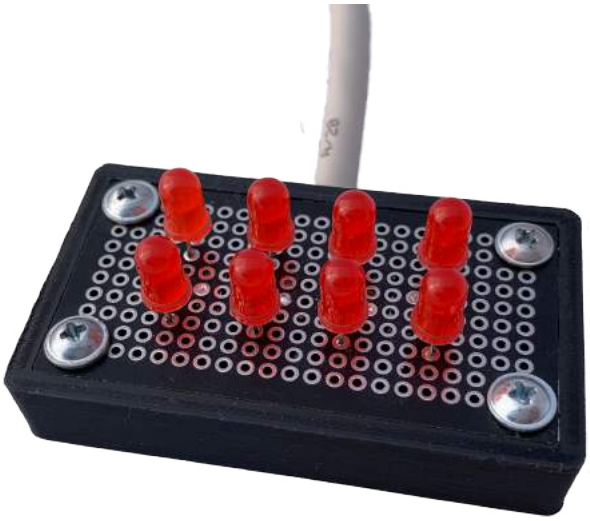
Tento modul slouží jako ukázka možného vstupního zařízení. Je sestaven z osmi spínačů rozmístěných na univerzální destičce, může tedy sloužit jako například ovladač. Toto zařízení je určeno pouze jako vstupní, můžeme tedy využít jeden ze vstupních portů, tedy port A nebo port B.



Obr. 5.3: Tlačítkový modul.

5.1.2 Modul s LED diodami

Tento modul slouží jako ukázka možného výstupního zařízení. Osm LED rozmístěných na desce je možné využít jako například indikátor stavu. Toto zařízení je pouze výstupní, můžeme tedy využít jeden z výstupních portů, tedy port C nebo port D.



Obr. 5.4: Modul s LED diodami.

5.1.3 Modul pro komunikaci s PS/2 klávesnicí

Periferií může být také jiný procesor, hradlové pole, nebo jiný programovatelný kontrolér. Tento modul je založený na koupené destičce převaděče PS/2 protokolu na "scancode".

Obr. 5.5: Modul pro komunikaci s klávesnicí.

5.2 Aplikační programy

Programy pro procesor jsou vytvářeny v jazyce C, který je při kompilaci převedený do strojového kódu a následně je nahrán skrze připojený programátor do procesoru.

5.2.1 Program pro násobení

Ukázka programu, který násobí hodnotu z portu A a portu B a jejich výsledek vypisuje na port C.

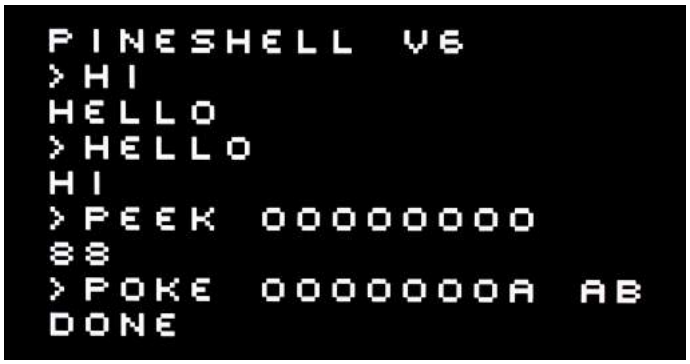
```
#include <stdint.h>

int main(void)
{
    while (1)
    {
        char operand_a = SR->INPUT_A;
        char operand_b = SR->INPUT_B;
        char out = operand_a * operand_b;
        SR->OUTPUT_C = out;
    }
    return 0;
}
```

Ukázkový program provádějící součin.

5.2.2 PineShell

Pro demonstraci využití periferií procesoru jsem se rozhodl vytvořit jednoduchý interaktivní shell. Shell poskytuje textové rozhraní (TUI), které je zobrazováno na monitoru, připojeném přes rozhraní VGA. Je možné jej ovládat klávesnicí připojenou přes rozhraní PS/2.



Obr. 5.6: Ukázka programu Pineshell.

V tuto chvíli může uživatel začít psát potřebné příkazy, které se po stisknutí klávesy „enter“ vykonají.

Seznam všech příkazů:

- `HELLO`
- `HI`
- `PEEK <ADDRESS>`
- `POKE <ADDRESS> <DATA>`
- `SYSTEM INFORMATION`
- `RUN SNAKE`
- `CLEAR`

Příkaz `HELLO` a příkaz `HI` Slouží k demonstraci textového rozhraní, výstupem příkazu `HELLO` bude text „HI“ a obdobně u příkazu `HI` bude výstupem text „HELLO“.

```
1 >HELLO
2 HI
```

Výpis 5.1: Ukázka příkazu HELLO.

```
1 >HI
2 HELLO
```

Výpis 5.2: Ukázka příkazu HI.

Příkaz `PEEK <ADDRESS>` Vrátí hodnotu, která je uložena v datové paměti na adrese <ADDRESS>. Všechny číselné hodnoty jsou v hexadecimální soustavě. Posíláme-li na port A data `0x1F`, můžeme si ověřit jejich správnost následujícím příkazem.

```
1 >PEEK 80000004
2 1F
```

Výpis 5.3: Ukázka příkazu PEEK <ADDRESS>.

Příkaz `POKE <ADDRESS> <DATA>` Přepíše hodnotu v datové paměti na adrese <ADDRESS> hodnotou <DATA>. Všechny číselné hodnoty jsou v hexadecimální soustavě. Pro vykreslení písmene „F“ na vestavěném LED displeji použijeme následující příkaz.

```
1 >POKE 80000000 8E
2 DONE
```

Výpis 5.4: Ukázka příkazu POKE <ADDRESS> <DATA>.

Příkaz `SYSTEM INFORMATION` Vypíše informace o konfiguraci procesoru. Tyto informace budou vypsány v novém okně, stisknutím klávesy „ESCAPE“ se okno skryje.

```
>SYSTEM INFORMATION
```

Výpis 5.5: Ukázka příkazu SYSTEM INFORMATION

Příkaz `RUN SNAKE` Spustí v novém okně hru s názvem „had“. Cílem této hry je nasbírat co nejvíce otazníků po herním poli a tím vytvořit co nejdelšího hada. Had se po herním poli pohybuje automaticky a směr mu udává hráč stisknutím příslušné šipky na klávesnici. Po kontaktu hada s otazníkem se délka hada prodlouží o jedno políčko a vygeneruje se nová pozice otazníku. Pro ukončení hry je třeba stisknout klávesu „ESCAPE“.

```
>RUN SNAKE
```

Výpis 5.6: Ukázka příkazu RUN SNAKE

Příkaz `CLEAR` Vyčistí terminálové okno.

```
>CLEAR
```

Výpis 5.7: Ukázka příkazu CLEAR

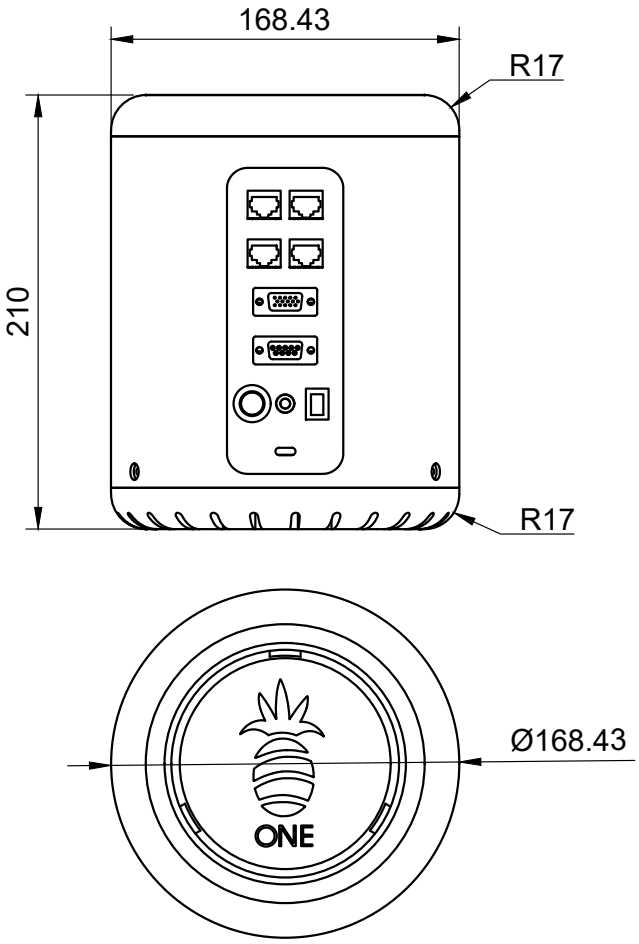
5.3 Parametry zařízení

Po úspěšném otestování celého zařízení jsem vytvořit testovací program, navržený tak, aby využíval všechny části procesoru. Poté jsem zahájil sérii měření.

Průměrná spotřeba Průměrná spotřeba byla stanovena z deseti měření, prováděných s periodou dvou minut. kdy procesor vykonával testovací program, byla 470 mA. Při měření byla frekvence zařízení nastavena na 500 kHz. Měření bylo provedeno digitálním multimetrem FK8550.

Frekvence Frekvence zařízení je uživatelem volitelná mezi 250 kHz, 500 kHz, 1 MHz a 2 MHz pomocí zkratovací propojky. Při měření těchto frekvencí nebyla pozorována žádná výrazná odchylka, ale při frekvenci nad 500 kHz se začaly objevovat nežádoucí artefakty ve vykonávaném programu. Tyto artefakty byly pravděpodobně způsobeny parazitní kapacitou mezi vodiči na deskách plošných spojů.

Fyzické parametry zařízení Rozměry finální verze zařízení jsou zobrazeny na obrázku 5.7. Celková váha činí 850 gramů.



Obr. 5.7: Rozměry zařízení.

Závěr

Cílem této práce bylo sestavit funkční procesor architektury RISC-V, pouze za použití jednotlivých logických hradel a pamětí. Vývoj probíhal v následující etapách:

1. Nejdříve jsem musel nastudovat obecné principy jak procesory fungují, z jakých bloků se skládají a co je jejich účelem. Poté jsem se věnoval analýze architektury RISC-V. Na základě získaných teoretických znalostí jsem vytvořil simulaci procesoru RV32I v programu Logisim-Evolution. Tato etapa mi zabrala přibližně 5 měsíců.
2. Podle schématu vytvořeného v předchozí části jsem sestavil prototyp, abych jej mohl ověřit s reálnými součástkami a odhalit případné problémy. Při testování zařízení bylo odhaleno několik hazardů, které byli ošetřeny RC články. Tato etapa mi zabrala přibližně 8 měsíců.
3. Ve schématech jsem opravil odhalené hardwarové problémy z prototypu. Následně jsem vytvořil finální plošné spoje. Plošné spoje jsem osadil a oživil, také jsem navrl krabičku a tu vytiskl na 3D tiskárně. Tato etapa mi zabrala přibližně 5 měsíců.
4. Po dokončení hardwarové části jsem pro tento procesor napsal jednoduchý shell, kterým jsem testoval procesor a zároveň slouží jako demonstrační aplikace. Tato etapa mi zabrala přibližně 1 měsíc.

Podařilo se mi vytvořit funkční makrokontrolér a demonstrační aplikaci. Zařízení je kompatibilní s procesory RV32I. Celý vývoj mi zabral přibližně 19 měsíců. Náklady na součástky nepřekročili 10 000 Kč. Zařízení pracuje spolehlivě do frekvence 0,5 MHz. Vyžaduje napájecí napětí 5V a při maximální frekvenci odebírá proud 470 mA.

Tato práce zahrnuje první část projektu Pineapple ONE, která se zaměřila na samotnou funkčnost projektu. Ve druhé části optimalizuji návrhy desek plošných spojů a na tomto základě vytvořím pájecí kit pro pokročilé. Ve třetí části vypracuji dokumentaci k jednotlivým obvodům a vytvořím tak výukovou pomůcku pro lepší pochopení fungování procesoru. Kdokoliv si takto bude moct vytvořit vlastní 32 bitový procesor, podívat se na jednotlivé moduly jak spolu komunikují a spolu s dokumentací pochopit princip jeho fungování.

Literatura

[1] Eater, B.: Let's build a video card! [online]. Červenec 2019, [Online; cit. 2021-02-22].
URL https://eater.net/vga

[2] Hordějčuk, V.: Turingův stroj. [Online; cit. 2021-02-13].
URL http://voho.eu/wiki/turinguv-stroj

[3] Nosterský, M.: Vytvoření modelu procesoru RISC-V. 2016, [Online; cit. 2021-02-13].
URL https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=132092

[4] Patterson, D. A.; Hennessy, J. L.: *Computer Organization and Design RISC-V Edition: The Hardware Software Interface*. Oxford, England: Morgan Kaufmann, 2017.

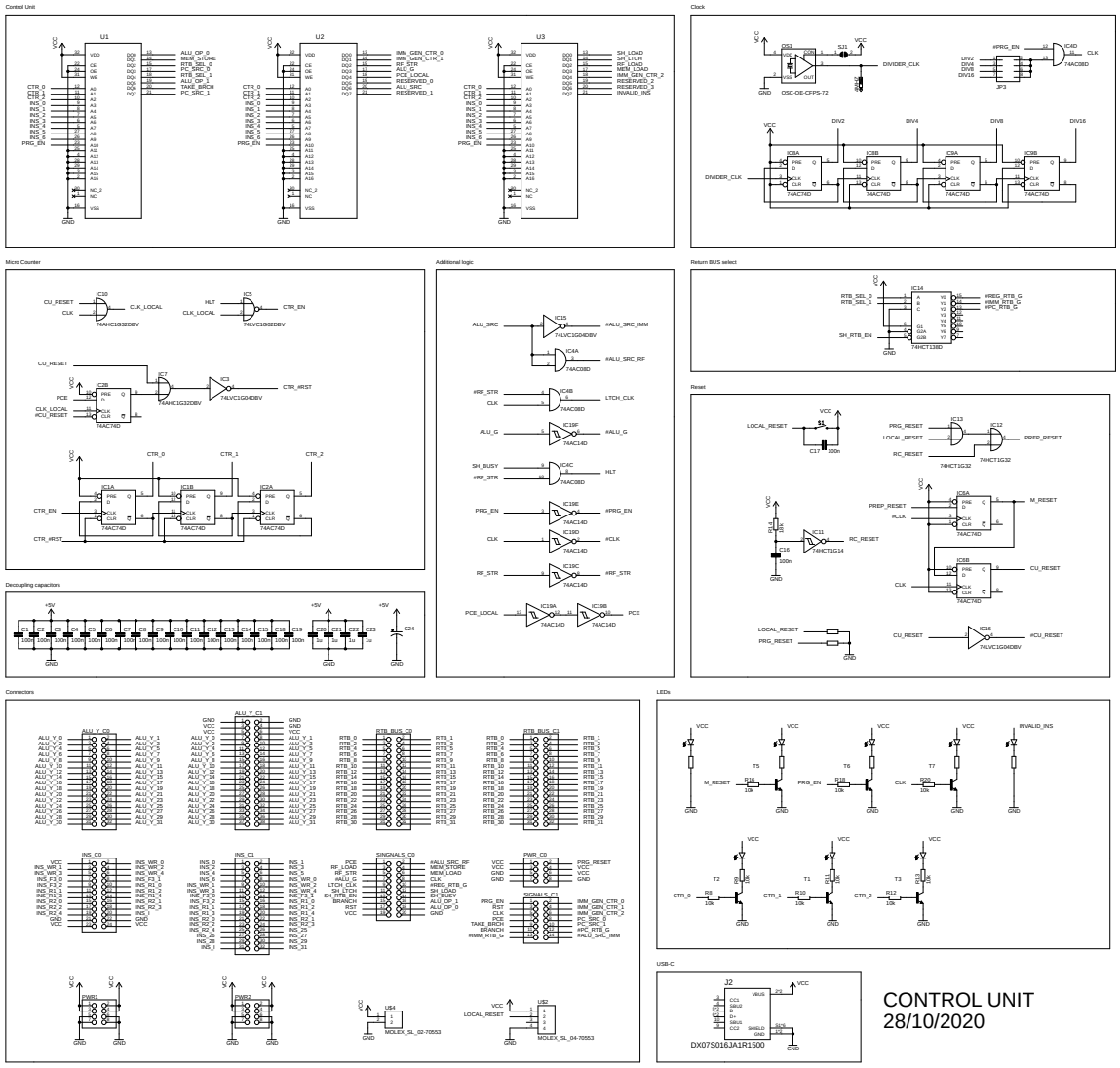
[5] Waterman, A.; Lee, Y.; Patterson, D. A.; aj.: The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Version 2.0. Technická Zpráva UCB/EECS-2014-54, EECS Department, University of California, Berkeley, Květen 2014.
URL http://www2.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-54.html

Seznam symbolů, veličin a zkratek

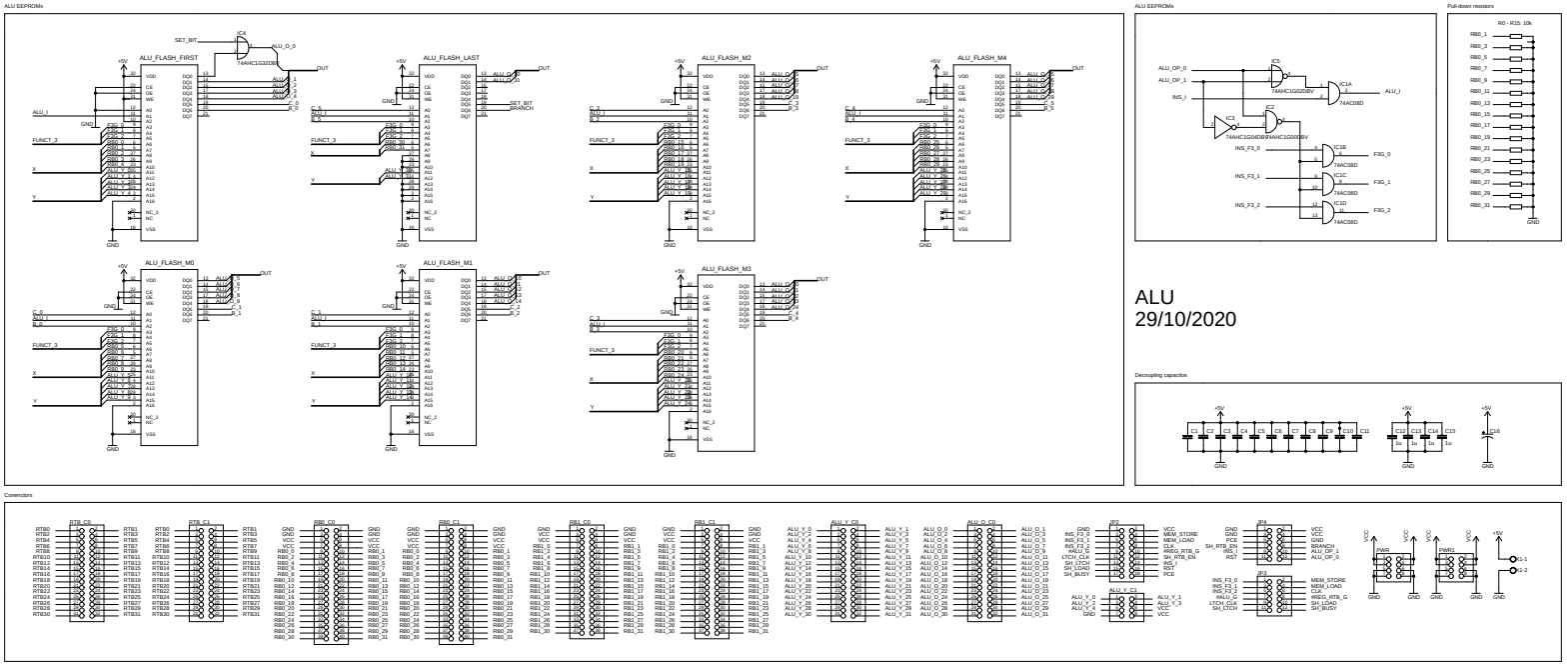
PCB	Printed Circuit Board – Deska plošných spojů
SMD	Surface Mount Device
CPU	Central Processing Unit - Procesor
RISC	Reduced Instruction Set Computer
CISC	Complex Instruction Set Computer
ALU	Arithmetic Logic Unit - Aritmeticko-logická jednotka
RAM	Random Acceess Memory
V-RAM	Video RAM
VGA	Video Graphics Array
TUI	Text User Interface
EEPROM	Electrically Erasable Programmable Read-only Memory
GCC	GNU Compiler Collection

Seznam příloh

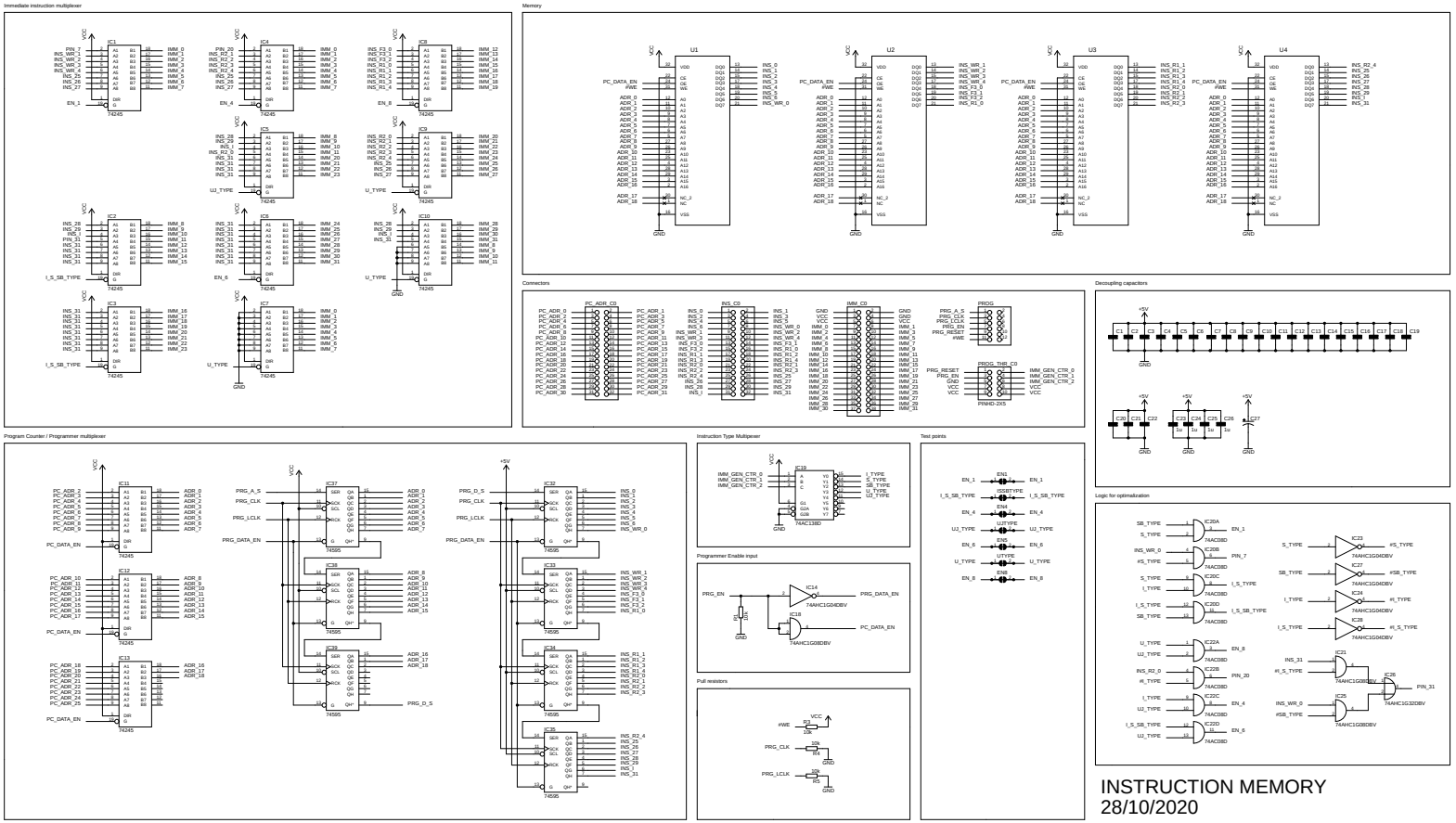
A Přiložená schémata zapojení



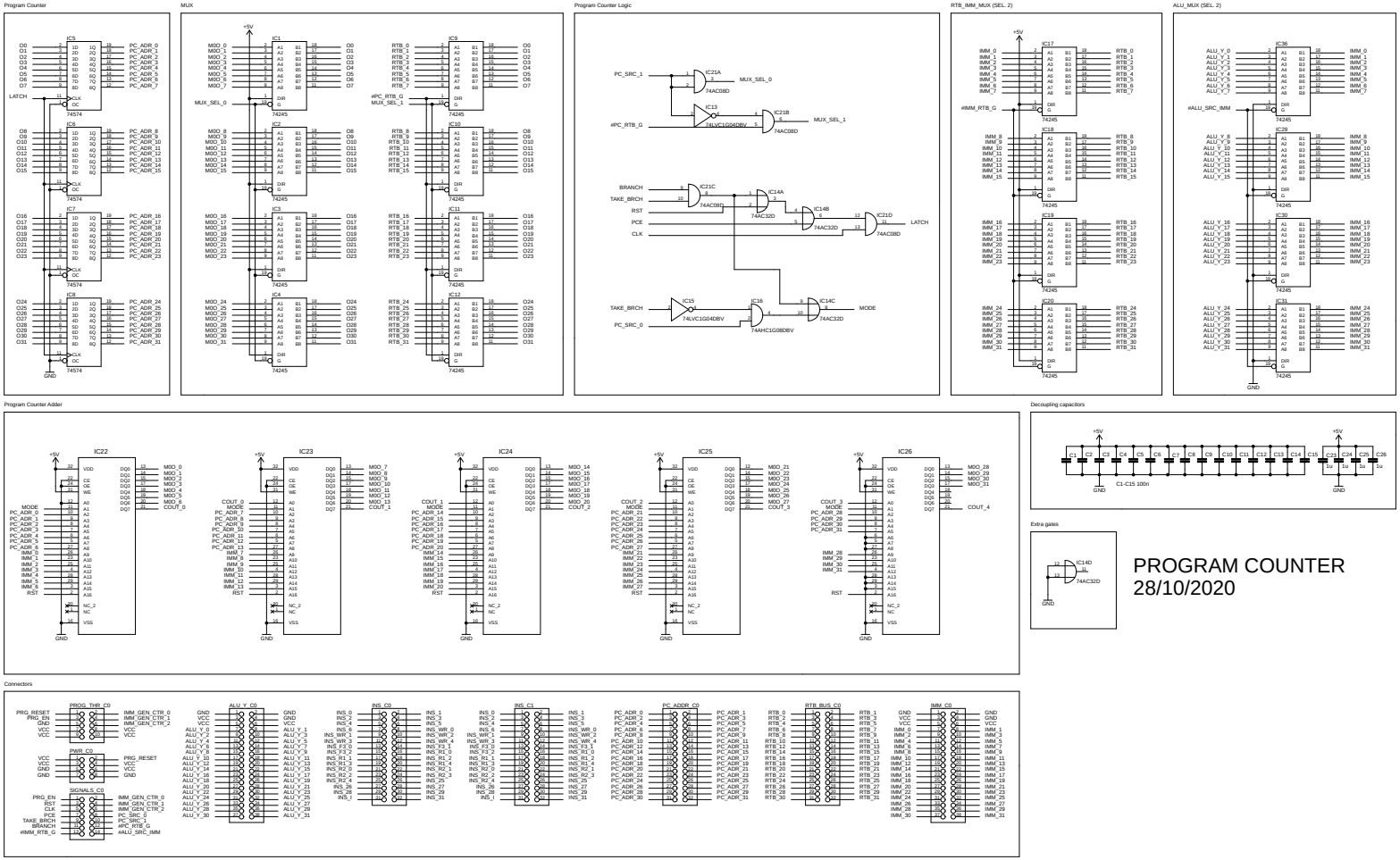
Obr. A.1: Schéma zapojení – Řadič.



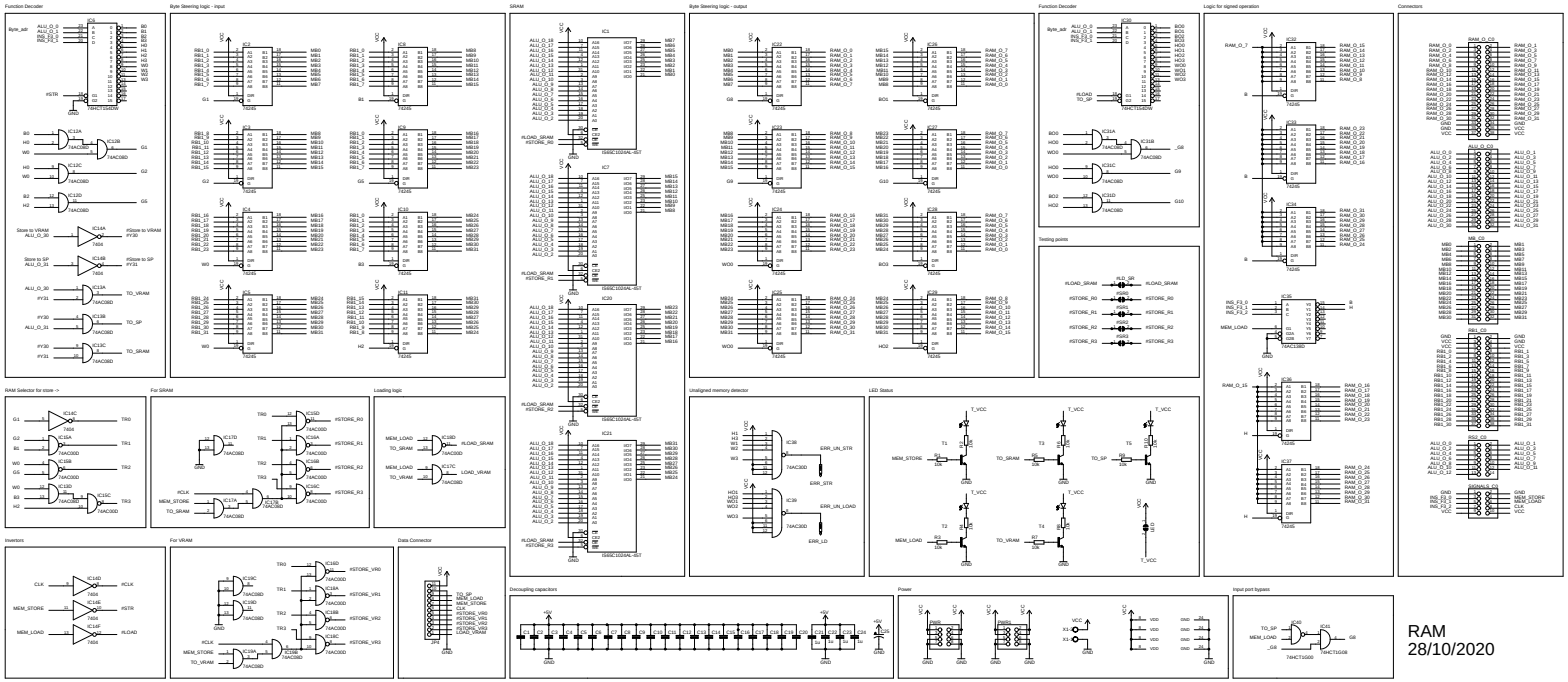
Obr. A.2: Schéma zapojení – ALU.



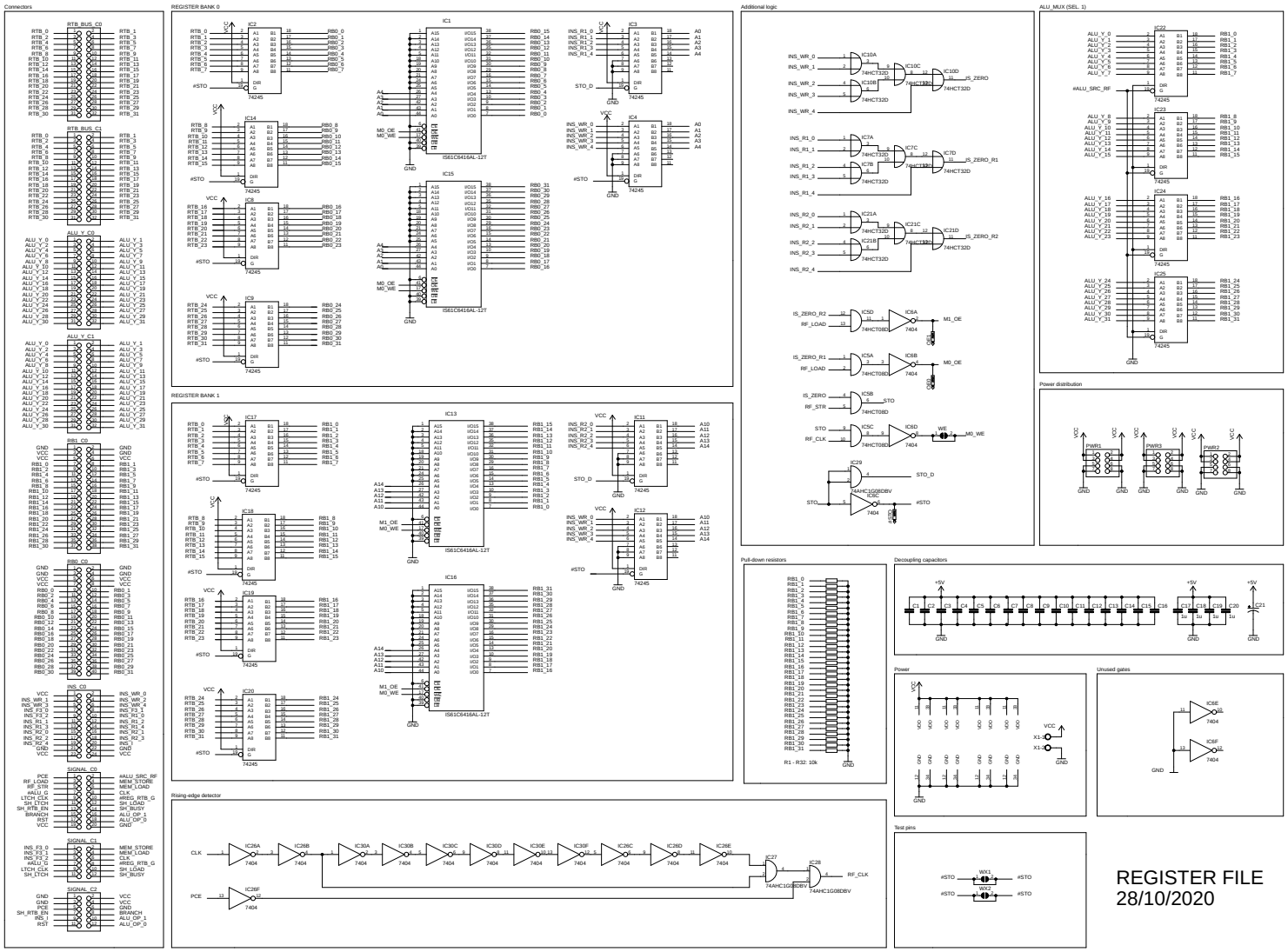
Obr. A.3: Schéma zapojení – Paměť instrukcí.



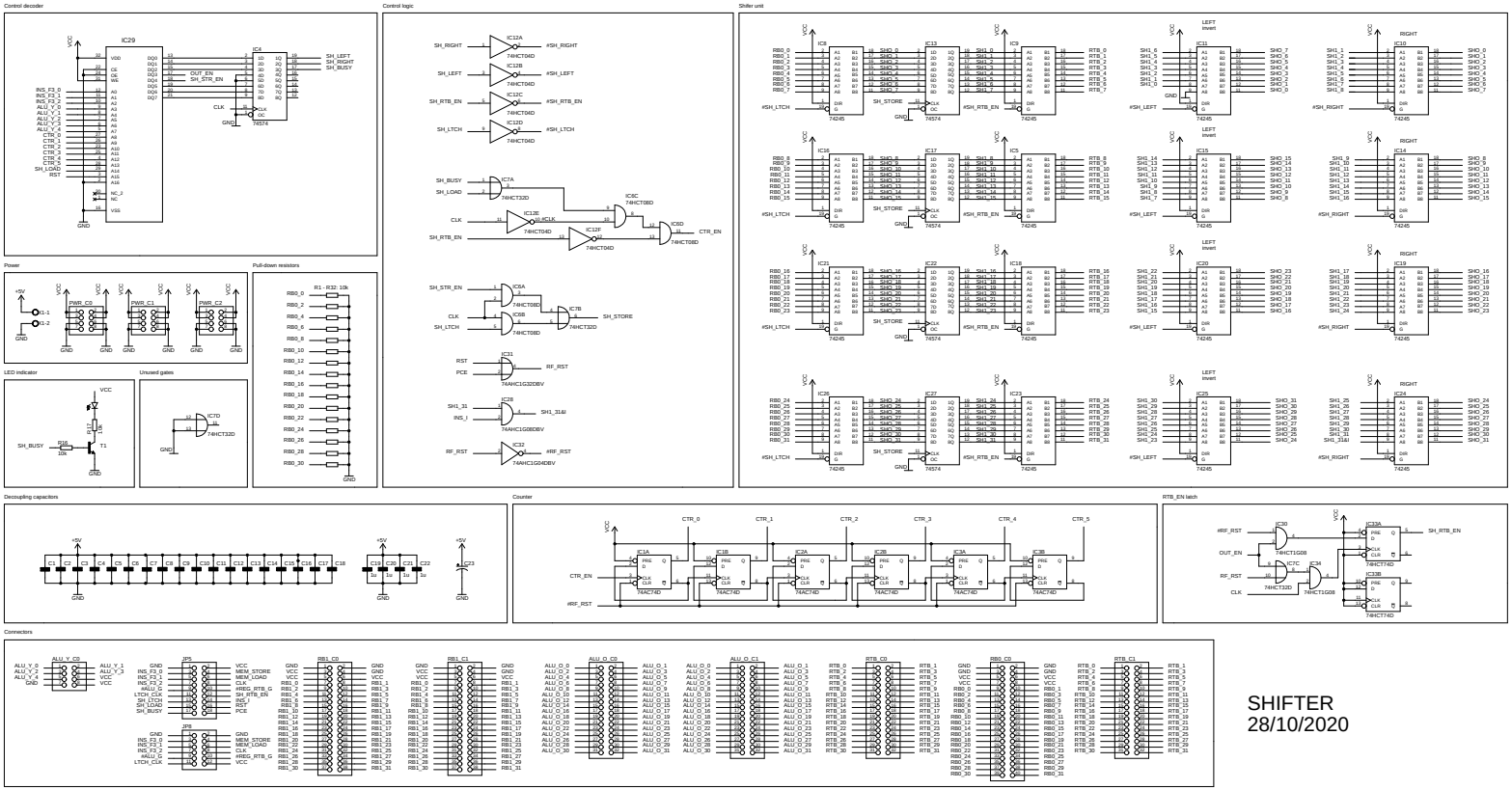
Obr. A.4: Schéma zapojení – Čítač instrukcí.



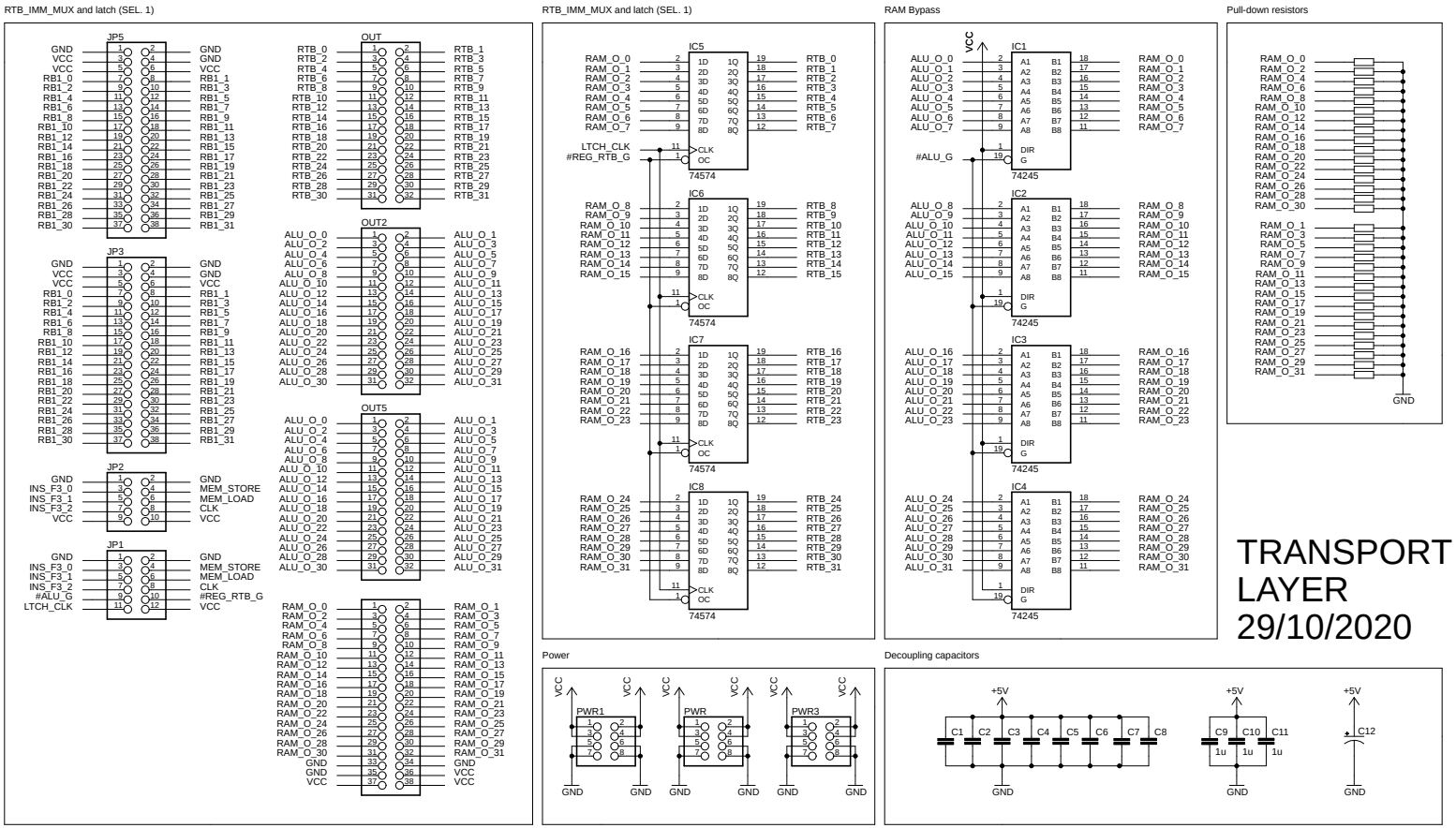
Obr. A.5: Schéma zapojení – Paměť RAM.



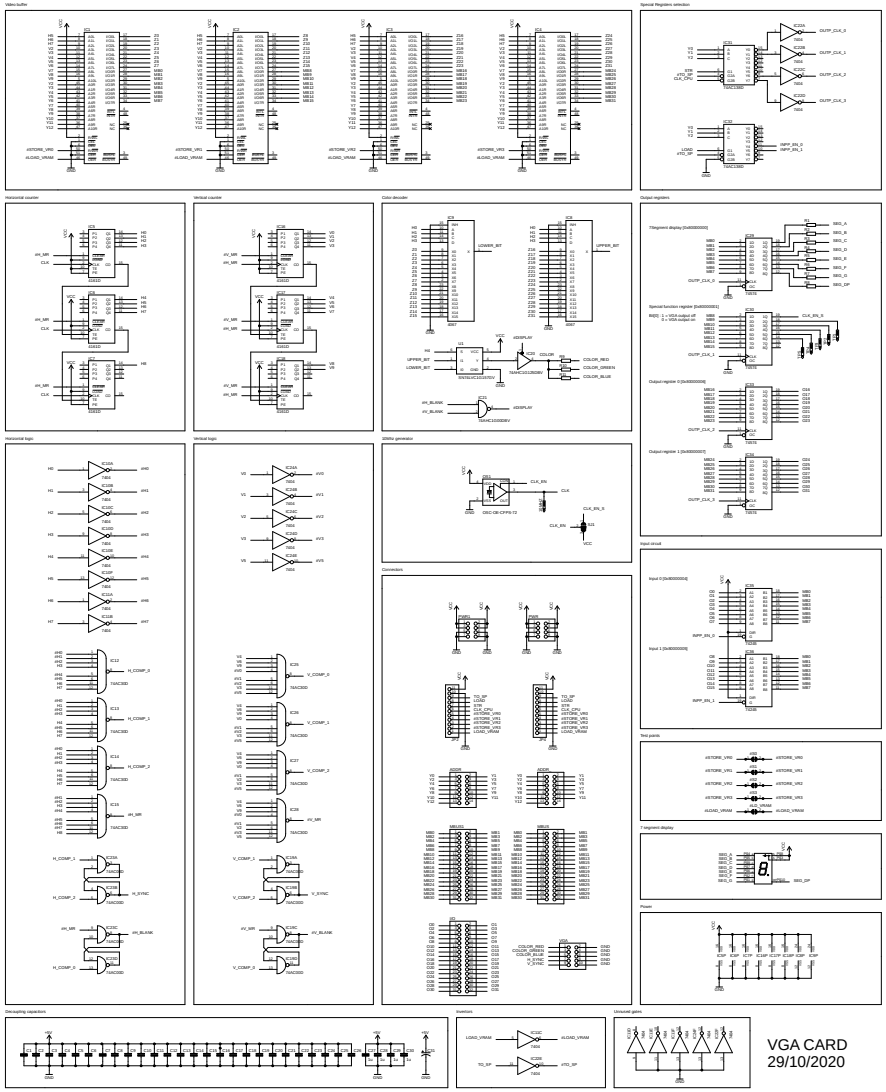
Obr. A.6: Schéma zapojení – Registrová banka.



Obr. A.7: Schéma zapojení – Posouvací jednotka.

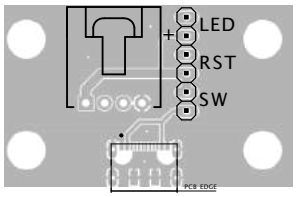


Obr. A.8: Schéma zapojení – Transportní vrstva.

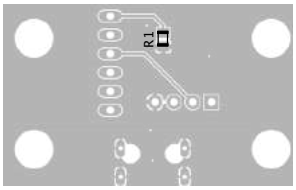


Obr. A.9: Schéma zapojení – Karta VGA.

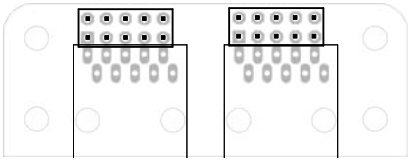
B Osazovací plán plošných spojů



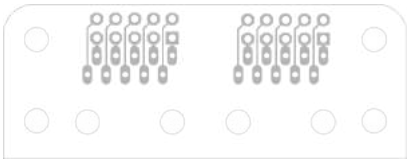
Obr. B.1: Osazovací plán v měřítku 1:1 – USB-C konektor – vrchní strana.



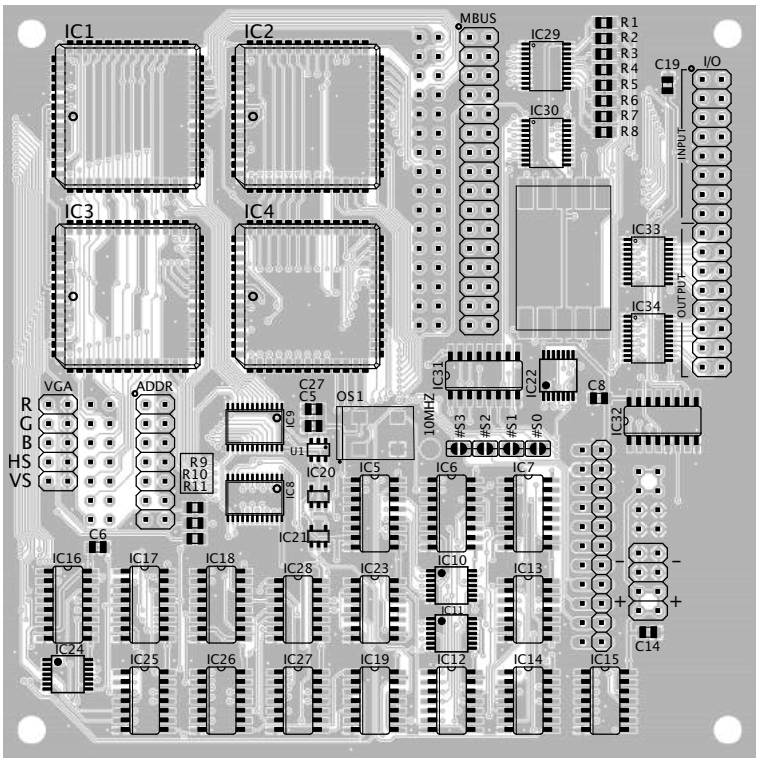
Obr. B.2: Osazovací plán v měřítku 1:1 – USB-C konektor – spodní strana.



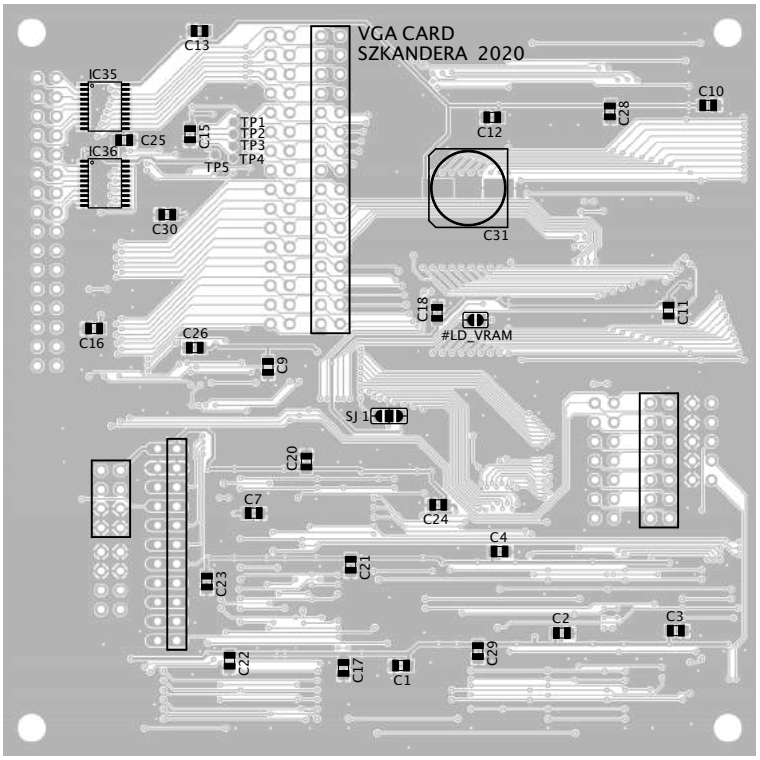
Obr. B.3: Osazovací plán v měřítku 1:1 – RJ50 konektor – vrchní strana.



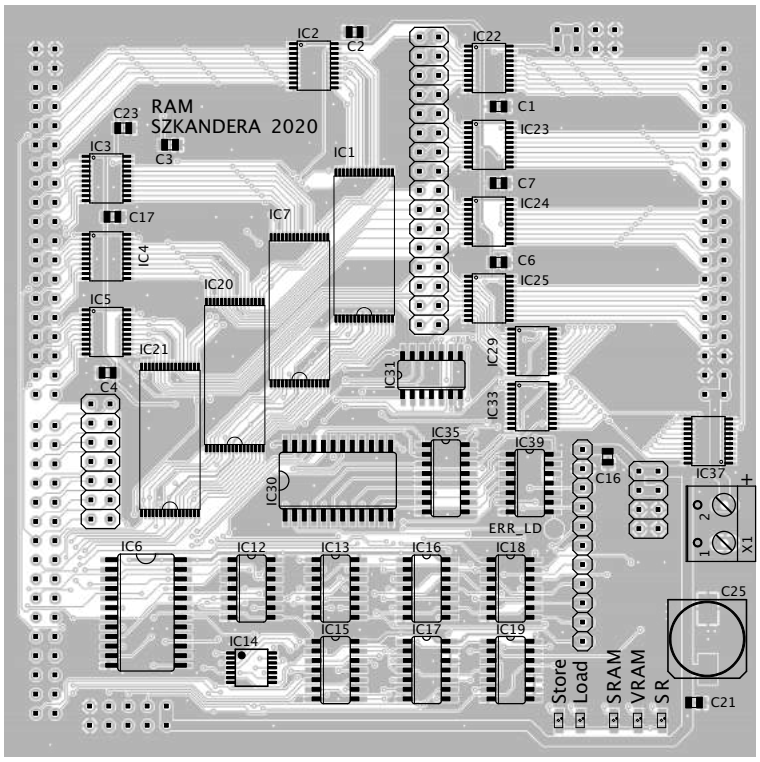
Obr. B.4: Osazovací plán v měřítku 1:1 – RJ50 konektor – spodní strana.



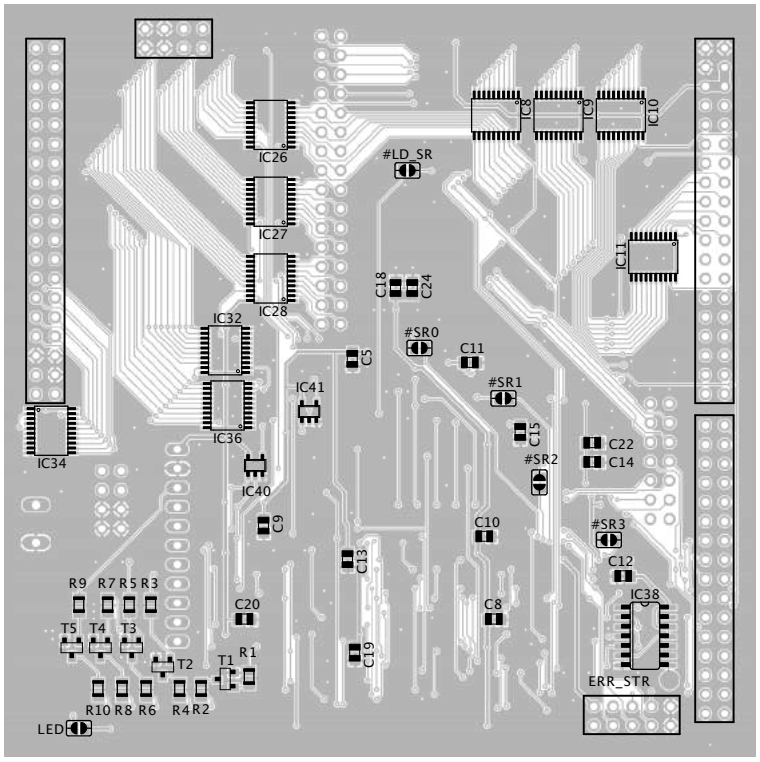
Obr. B.5: Osazovací plán v měřítku 1:1 – VGA karta – vrchní strana.



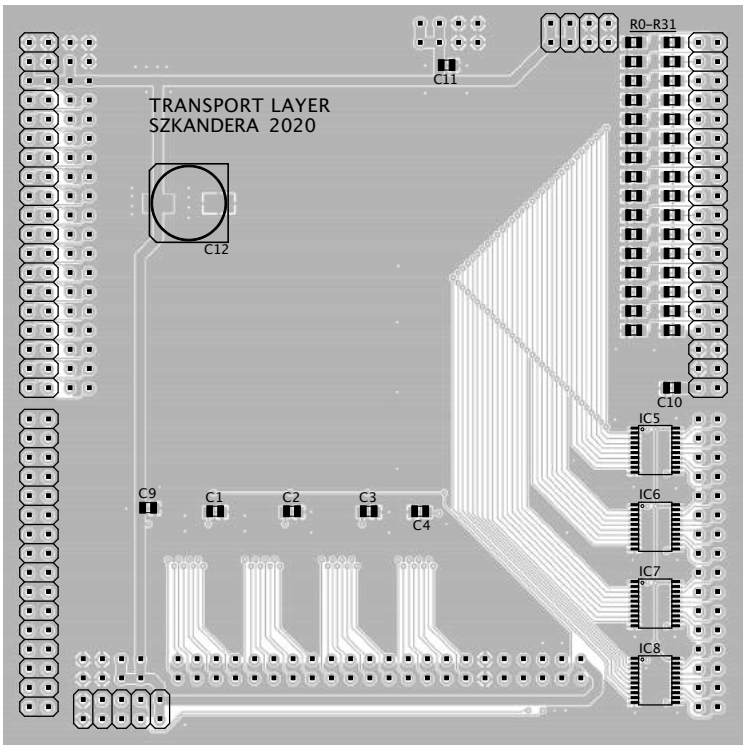
Obr. B.6: Osazovací plán v měřítku 1:1 – VGA karta – spodní strana.



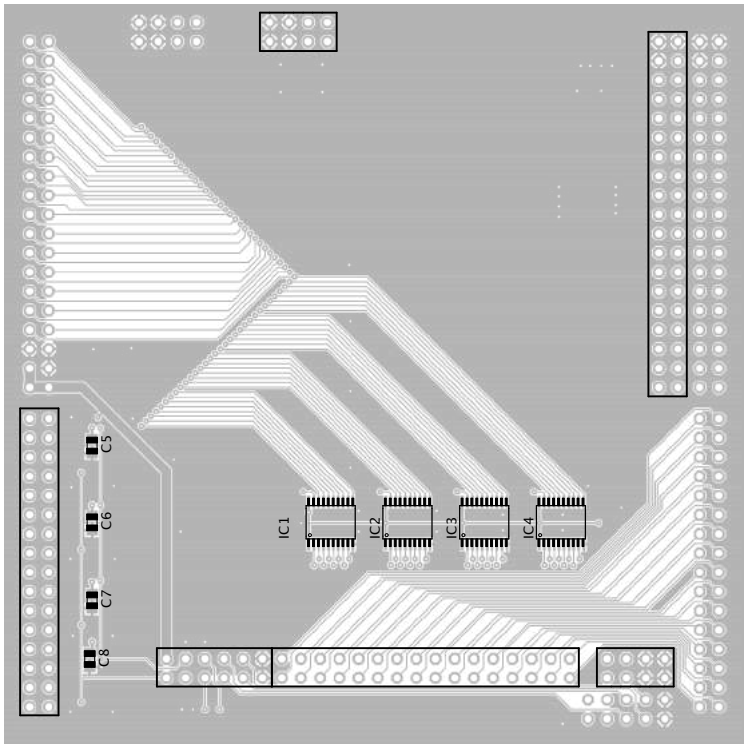
Obr. B.7: Osazovací plán v měřítku 1:1 – RAM – vrchní strana.



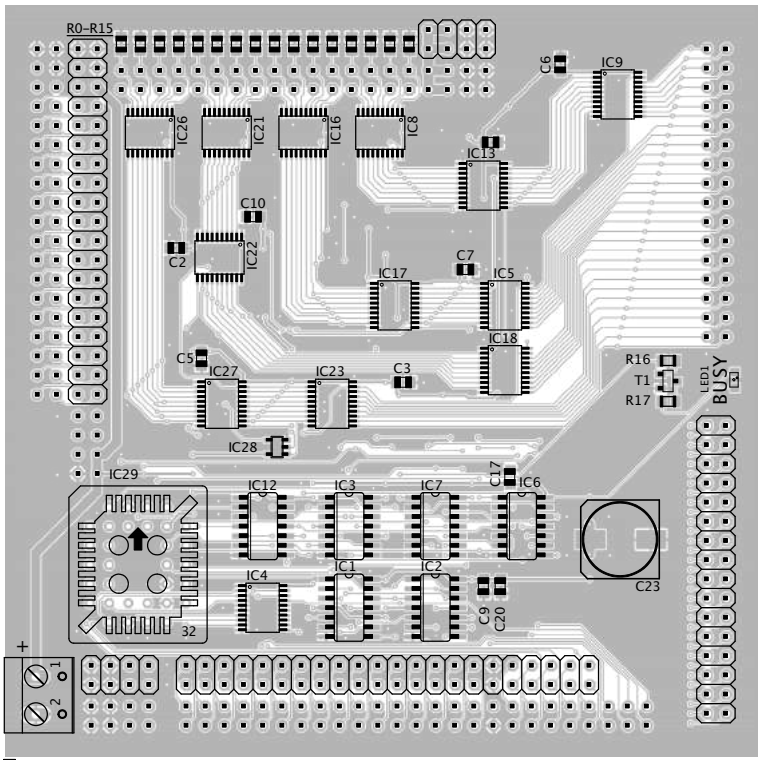
Obr. B.8: Osazovací plán v měřítku 1:1 – RAM – spodní strana.



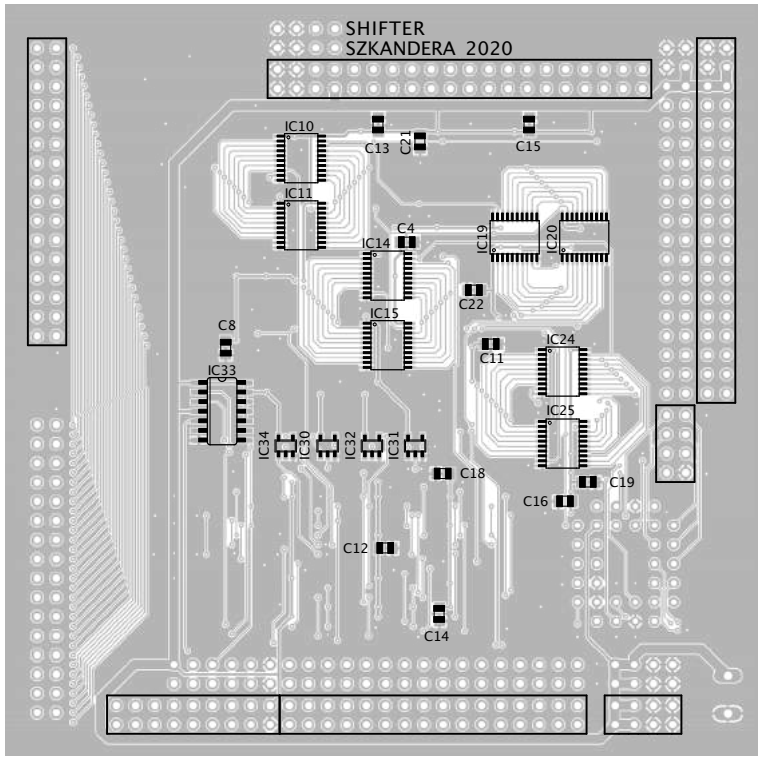
Obr. B.9: Osazovací plán v měřítku 1:1 – Transportní vrstva – vrchní strana.



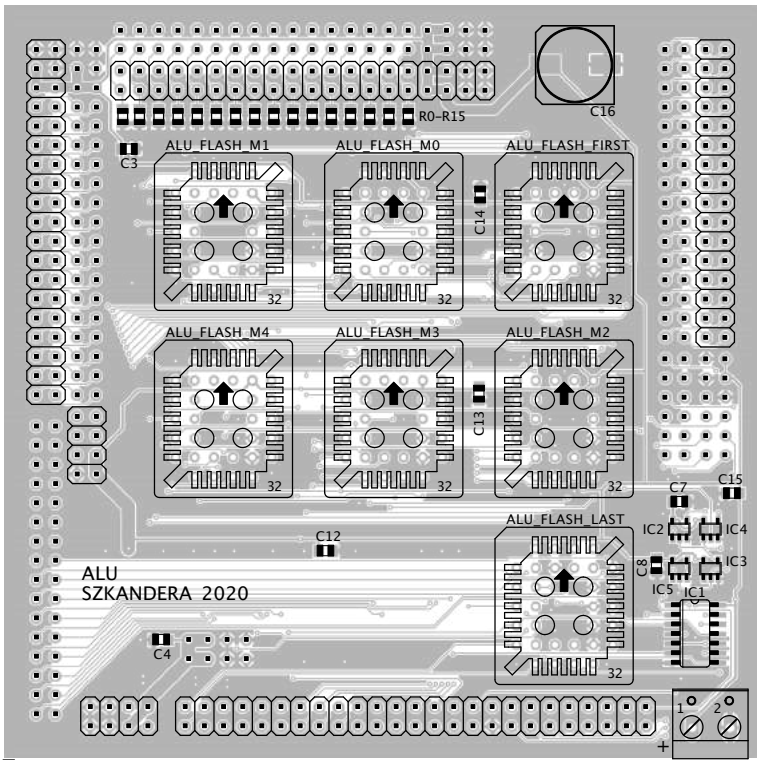
Obr. B.10: Osazovací plán v měřítku 1:1 – Transportní vrstva – spodní strana.



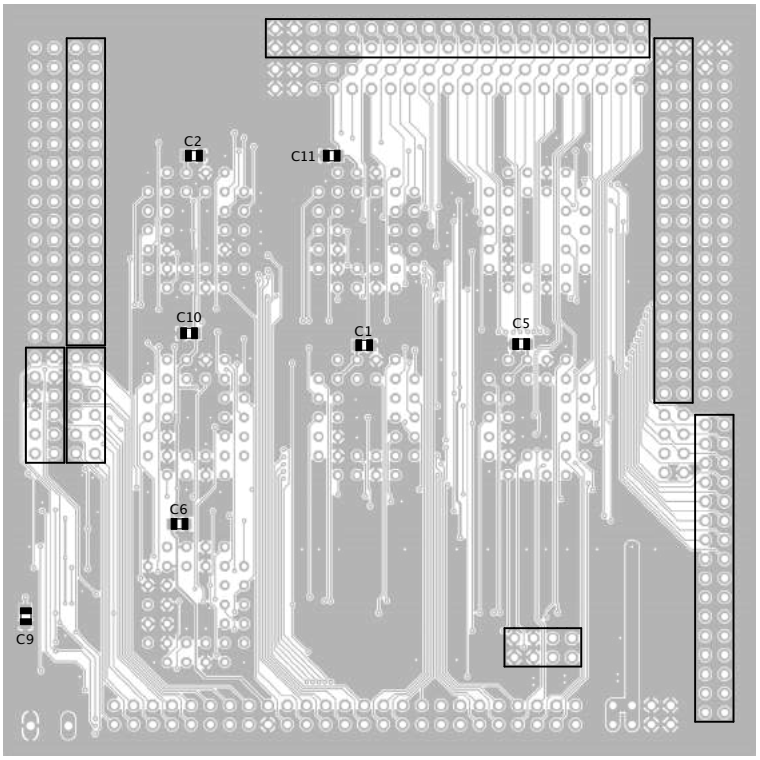
Obr. B.11: Osazovací plán v měřítku 1:1 – Shifter – vrchní strana.



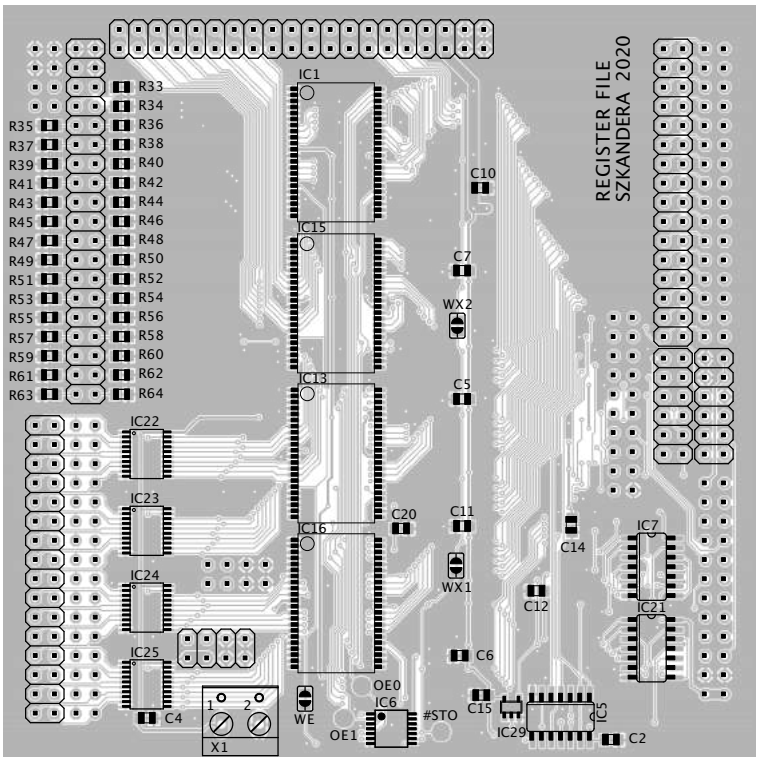
Obr. B.12: Osazovací plán v měřítku 1:1 – Shifter – spodní strana.



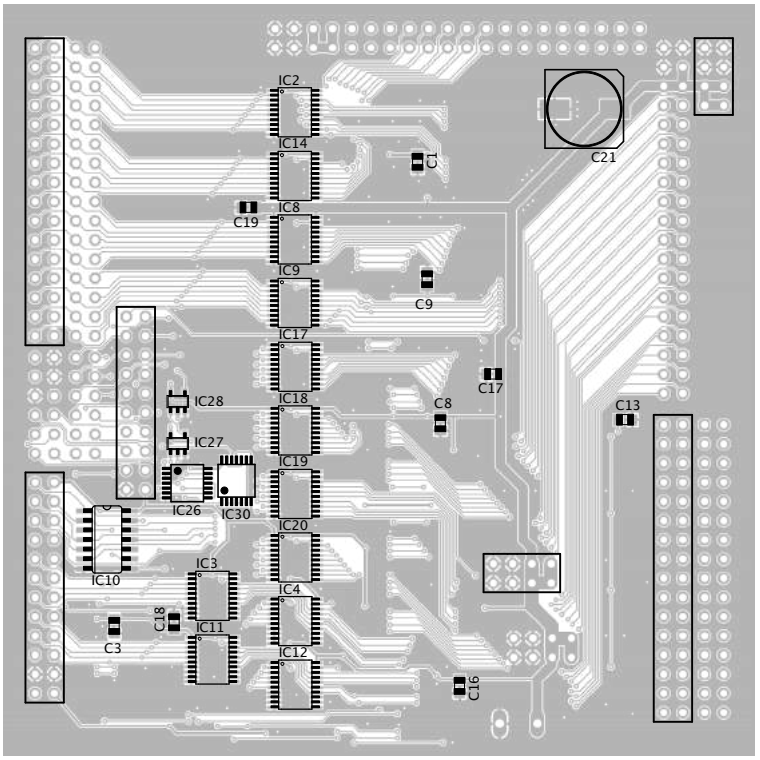
Obr. B.13: Osazovací plán v měřítku 1:1 – ALU – vrchní strana.



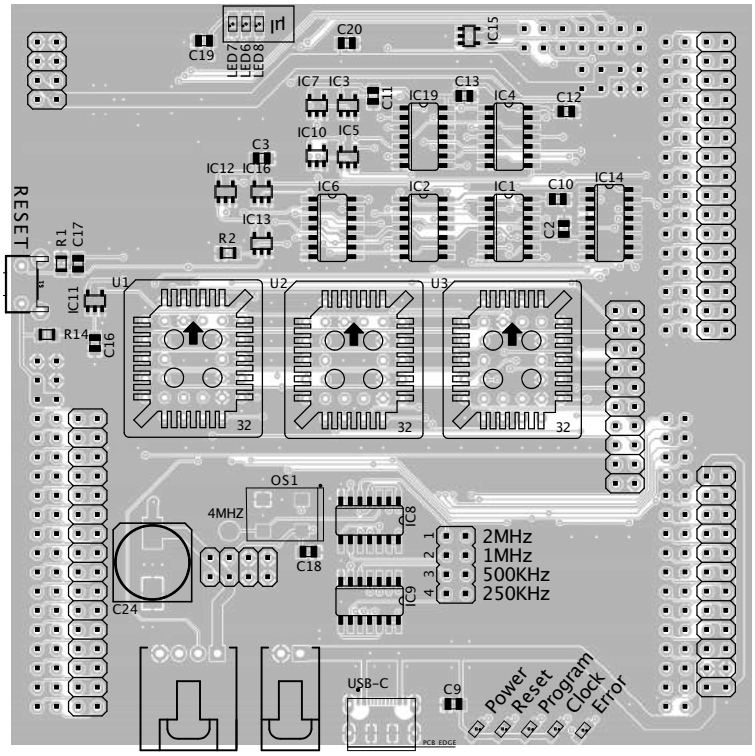
Obr. B.14: Osazovací plán v měřítku 1:1 – ALU – spodní strana.



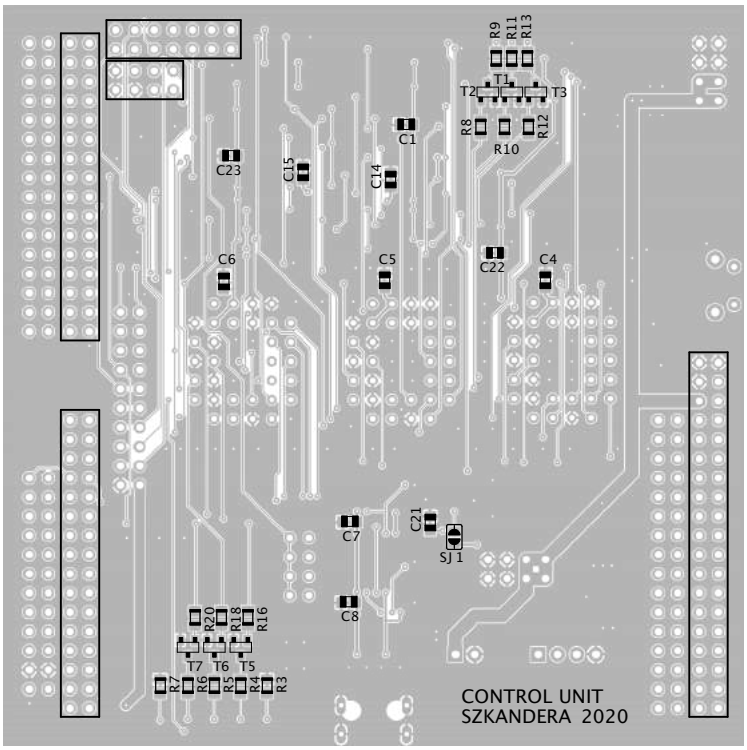
Obr. B.15: Osazovací plán v měřítku 1:1 – Registrová banka – vrchní strana.



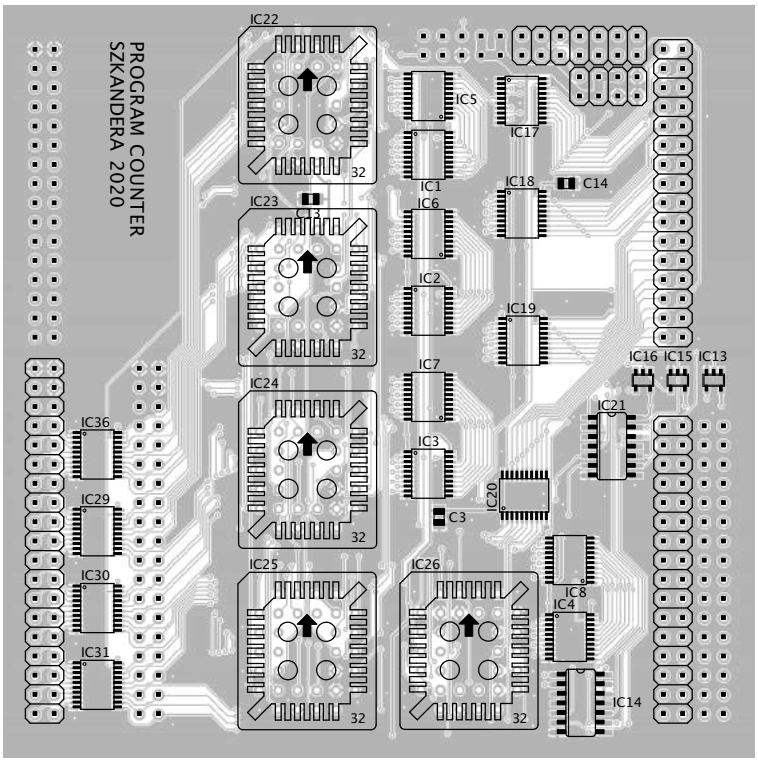
Obr. B.16: Osazovací plán v měřítku 1:1 – Registrová banka – spodní strana.



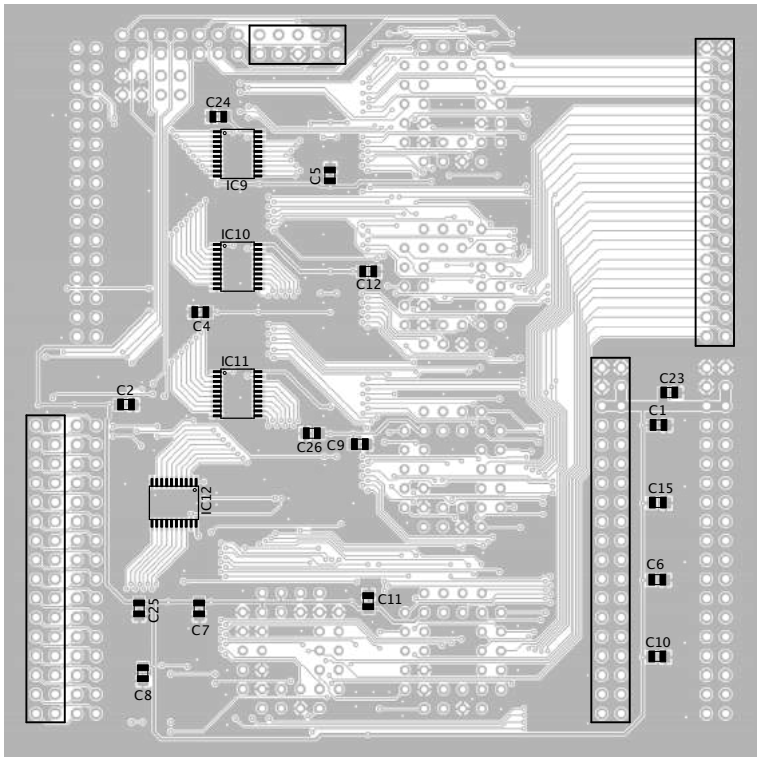
Obr. B.17: Osazovací plán v měřítku 1:1 – Řadič – vrchní strana.



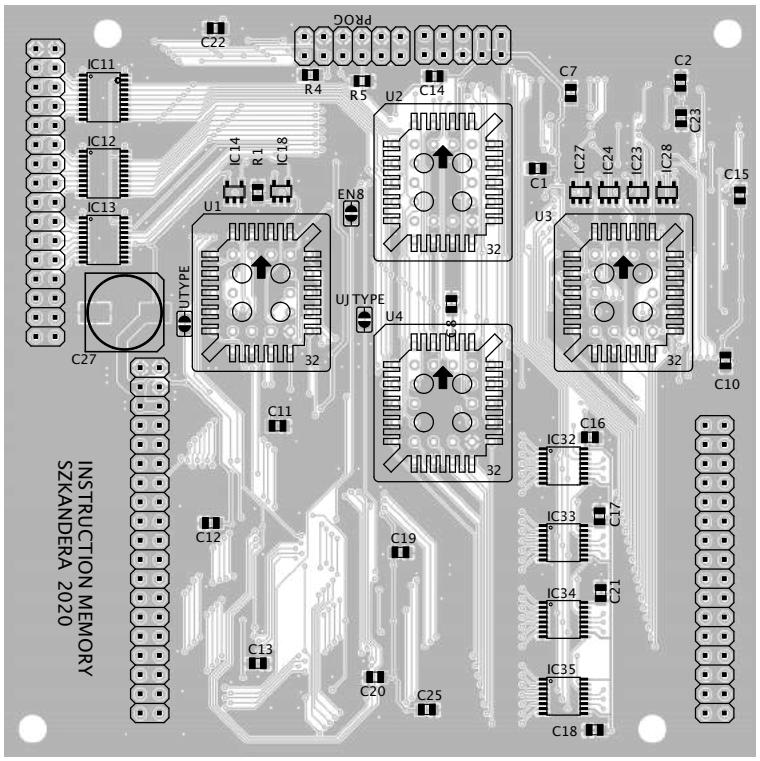
Obr. B.18: Osazovací plán v měřítku 1:1 – Řadič – spodní strana.



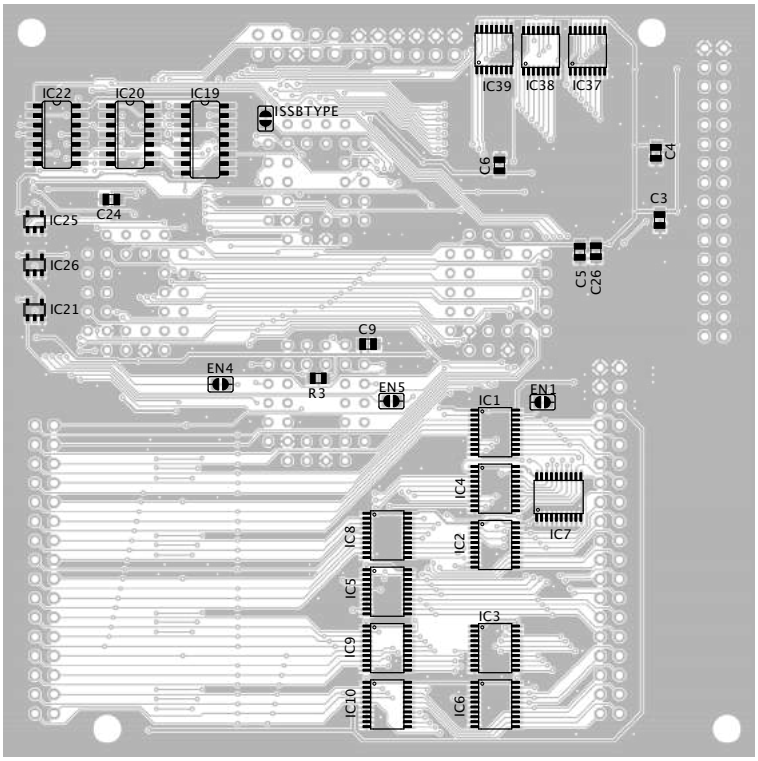
Obr. B.19: Osazovací plán v měřítku 1:1 – Čítač instrukcí – vrchní strana.



Obr. B.20: Osazovací plán v měřítku 1:1 – Čítač instrukcí – spodní strana.



Obr. B.21: Osazovací plán v měřítku 1:1 – Paměť programu – vrchní strana.



Obr. B.22: Osazovací plán v měřítku 1:1 – Paměť programu – spodní strana.