Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Popis architektury RISC-V

Michal Prokš

Bakalářská práce

2022

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2021/2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Michal Prokš**
Osobní číslo:	**I19139**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Téma práce:	**Popis architektury RISC-V**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování

Cílem bakalářské práce je popis architektury RISC-V. V teoretické části práce bude podrobně popsána architektura RISC-V a jednotlivé instrukční sady pro RV32, RV64, včetně jejich modifikací. Praktická část práce bude zaměřena na dostupné GNU nástroje pro vývoj a bude také obsahovat stručný popis dostupných vývojových kitů s touto architekturou.

Rozsah pracovní zprávy: **min 30. stran**
Rozsah grafických prací:
Forma zpracování bakalářské práce: **tištěná**

Seznam doporučené literatury:

PATTERSON, David A. a John L. HENNESSY. Computer organization and design: the hardware/software interface. RISC-V edition. Cambridge, Massachusetts: Morgan Kaufmann Publishers, an imprint of Elsevier, [2018]. ISBN 978-0128122754.
HENNESSY, John L. Computer architecture: a quantitative approach. Sixth edition. Cambridge, MA: Morgan Kaufmann Publishers, [2019]. ISBN 978-0128119051.

Vedoucí bakalářské práce: **Ing. Miroslav Dvořák, Dipl.tech.**
Katedra informačních technologií

Datum zadání bakalářské práce: **17. prosince 2021**
Termín odevzdání bakalářské práce: **13. května 2022**

L.S.

Ing. Zdeněk Němec, Ph.D. v.r.
děkan

Ing. Jan Panuš, Ph.D. v.r.
vedoucí katedry

V Pardubicích dne 28. února 2022

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 15. 12. 2022

Michal Prokš

PODĚKOVÁNÍ

Mé poděkování patří Ing. Miroslavu Dvořákovi, Dipl.tech., za poskytnutí konzultací a zpětné vazby při vypracovávání bakalářské práce.

ANOTACE

Cílem bakalářské práce je popis architektury RISC-V. V teoretické části se věnuje popisu této architektury, jejích instrukčních sad RV32 a RV64 včetně rozšíření. Dále bude architektura RISC-V porovnána s architekturou ARM. V praktické části budou představeny GNU a jiné nástroje pro vývoj aplikací, které budou reprezentovány kompilátory, simulátory a dalším softwarem. Následně budou představeny vývojové kity a mikrokontrolery dostupné na trhu.

KLÍČOVÁ SLOVA

počítačová architektura, RISC-V, RV32, RV64, vývojový kit, vývojové nástroje GNU, mikrokontroler

TITLE

Description of the RISC-V Architecture

ANNOTATION

The aim of the bachelor thesis is to describe the RISC-V architecture. The theoretical part describes this architecture, its RV32 and RV64 instruction sets, including extensions. Furthermore, the RISC-V architecture will be compared with the ARM architecture. In the practical part, GNU and other tools for application development will be introduced, which will be represented by compilers, simulators and other software. Subsequently, development kits and microcontrollers available on the market will be introduced.

KEYWORDS

computer architecture, RISC-V, RV32, RV64, development kit, GNU development tools, microcontroller

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM ZKRATEK

ABI	Application Binary Interface
ADC	Analog-to-Digital Converter
AEE	Application Execution Environment
AES	Advanced Encryption Standard
ALU	Arithmetic Logic Unit
APB	Advanced Peripheral Bus
ARM	Advanced RISC Machine
BSD	Berkeley Software Distribution
CAN	Controller Area Network
CDT	C/C++ Development Tooling
CPU	Central Processing Unit
CSI	Camera Serial Interface
CSR	Control and Status Register
DAC	Digital-to-Analog Converter
DARPA	Defense Advanced Research Projects Agency
DIP	Dual In-line Package
DMA	Direct Memory Access
DSI	Display Serial Interface
ECDSA	Elliptic Curve Digital Signature Algorithm
EEPROM	Electrically Erasable Programmable Read-Only Memory
ELF	Executable and Linkable Format
eMMC	Embedded MultiMediaCard
FPGA	Field Programmable Gate Array
GNU	GNU's Not UNIX
GPIO	General-Purpose Input/Output
GPL	General Public License
HAL	Hardware Abstraction Layer
HBI	Hypervisor Binary Interface
HDMI	High-Definition Multimedia Interface
HEE	Hypervisor Execution Environment
HMAC	Hash-Based Message Authentication Code
I/O	Input/Output

I2C	Inter-Integrated Circuit
I2S	Inter-IC Sound
IDE	Integrated Development Environment
IoT	Internet of Things
ISA	Instruction Set Architecture
JTAG	Joint Test Action Group
LCD	Liquid Crystal Display
LED	Light-Emitting Diode
LPDDR4	Low-Power Double Data Rate 4
LSRAM	Large SRAM
MIPI	Mobile Industry Processor Interface
MIT	Massachusetts Institute of Technology
NAS	Network Attached Storage
OTG	On The Go
PC	Program Counter
PCB	Printed Circuit Board
PCIe	Peripheral Component Interconnect Express
PWM	Pulse Width Modulation
QPI	QuickPath Interconnect
QSPI	Quad Serial Peripheral Interface
RGB	Red-Green-Blue
RHEL	Red Hat Enterprise Linux
RISC	Reduced Instruction Set Computer
RSA	Rivest–Shamir–Adleman
RVWMO	RISC-V Weak Memory Ordering
SAR	Successive-Approximation Register
SBI	Supervisor Binary Interface
SD	Secure Digital
SDRAM	Synchronous Dynamic Random Access Memory
SEE	Supervisor Execution Environment
SERDES	Serializer/Deserializer
SFL	SEGGER’s Friendly License
SHA	Secure Hash Algorithm
SIMD	Single Instruction Multiple Data

SoC	System On a Chip
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
SWD	Serial Wire Debug
TF	TransFlash
TMD	Terasic Mini Digital
UART	Universal Asynchronous Receiver/Transmitter
uPROM	micro–Programmable Read-Only Memory
USART	Universal Synchronous/Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
USBFS	USB Full Speed
uSRAM	micro-SRAM
VIN	Voltage Input

ÚVOD

Bakalářská práce je zaměřená na novou počítačovou architekturu RISC-V. Jedná se o otevřenou architekturu, která umožňuje přizpůsobení konkrétnímu účelu použití pomocí rozšíření základní sady instrukcí. Díky této schopnosti nalezne využití jak v nejjednodušších mikrokontrolerech, tak i v osobních počítačích či serverech.

Jedním z hlavních konkurentů této architektury v současné době je architektura ARM. Její nevýhodou je ale nutnost získání placené komerční licence a možnost potenciálnímu zabránění poskytnutí této licence. RISC-V je naopak šířena jako otevřený standard a je možné ji používat zdarma.

Cílem bakalářské práce je popis architektury RISC-V, jejích základních instrukčních sad pro 32bitovou i 64bitovou variantu a možných standardních rozšíření. Dále bude architektura porovnána s architekturou ARM a budou představeny softwarové nástroje pro vývoj aplikací na tuto architekturu, jako jsou simulátory, kompilátory a integrovaná vývojová prostředí. Nakonec budou popsány příklady implementací RISC-V dostupné na trhu v podobě mikrokontrolerů, jednodeskových počítačů či programovatelných hradlových polí.

1 ARCHITEKTURA RISC-V

RISC-V je nová počítačová architektura (ISA) s redukovanou instrukční sadou. Jejím původním určením byla podpora výzkumu a vzdělávání, ale nyní se začíná prosazovat i jako průmyslový standard. Jedná se o otevřenou architekturu, která je volně dostupná jak pro oblast školství, tak i pro komerční využití. Jedná se o architekturu, která je rozdělena na malé ISA pro základní operace s celými čísly a volitelná standardní rozšíření.

Architektura RISC-V podporuje 32bitové, 64bitové a 128bitové adresové prostory, instrukce a registry pro hardwarové implementace, operační systémy a aplikace. Dále podporuje vysoce paralelizované implementace, včetně heterogenních multiprocesorů. Architekturu je možné úplně virtualizovat, díky čemuž umožňuje jednodušší vývoj hypervizoru. Je možné přidat i instrukce o redukované délce, které zmenší velikost kódu programu a zvýší rychlost výpočtů. RISC-V při své definici neurčuje implementační detaily a měla by reprezentovat softwarové rozhraní pro různé implementace než konkrétní hardwarovou implementaci [1]. Mezi možné oblasti použití patří mikročipy, výukové simulátory, cloudové servery, systémy umělé inteligence, IoT, počítačové systémy v automobilech a další [2].

1.1 Historie RISC-V

Historie architektury začíná v květnu 2010, kdy profesor Krste Asanović spolu s magisterskými studenty (Yunsup Lee, Andrew Waterman) vyvinul RISC-V v rámci laboratoře pro paralelní výpočty (Par Lab) na Kalifornské univerzitě. Projekty v rámci laboratoře včetně RISC-V jsou open-source pod licencí BSD. První financování RISC-V zajistili průmysloví partneři laboratoře. Dalším milníkem bylo v roce 2011 vyvinutí prvního čipu pro RISC-V na 28nm technologii, který byl darován firmou ST Microelectronics [3].

V roce 2015 byl pořádán první RISC-V Workshop [4], jehož cílem bylo informování komunity o různých projektech RISC-V a dohodnutí se na dalším směřování projektů, jakož i možnost představení existující infrastruktury. Dále byla v roce 2015 založena nadace RISC-V Foundation se 36 zakládajícími členy.

Specifikace architektury byly vydány jako volné dílo, text technické zprávy byl ale zařazen pod licenci Creative Commons, aby mohl být později vylepšen externími spoluúčastníky včetně nadace RISC-V Foundation.

Par Lab byl v roce 2013 nahrazen projektem ASPIRE Lab, v jehož rámci byly vyvinuty další RISC-V kompatibilní mikroprocesory. Projekt byl financován různými průmyslovými partnery a dále také agenturou DARPA amerického ministerstva obrany [3].

1.1.1 RISC-V International

Nadace RISC-V Foundation byla založena za účelem vytvoření komunity hardwarových a softwarových inovátorů architektury RISC-V. Je to nezisková společnost, jejímž účelem je určování dalšího vývoje a podpora počátečního přijetí architektury.

V listopadu 2018, RISC-V Foundation zahájila spolupráci s Linux Foundation. V roce 2020 došlo k transformaci na mezinárodní organizaci RISC-V International. Sídlo organizace je ve Švýcarsku [3].

1.2 Hardwarové platformy

Hardwarová platforma založená na RISC-V architektuře může mít jedno nebo více jader kompatibilních s RISC-V společně s jádry, která RISC-V nepodporují. Jádra jsou propojená s dalšími vstupně-výstupními zařízeními, akcelerátory a paměťovými strukturami. Platforma může mít různou velikost a úroveň složitosti, od jednoduchých mikrokontrolerů až po výpočetní cluster.

Komponenta platformy je nazývána jádrem, pokud obsahuje nezávislou jednotku pro načítání instrukcí. Jádra kompatibilní s RISC-V mohou dále podporovat vícevláknové vykonávání. Na architektuře jsou hardwarová vlákna v angličtině označována jako harts.

Jádra kompatibilní s RISC-V mohou mít rozšíření instrukční sady nebo přidaný koprocesor, který může obsahovat další rozšíření instrukční sady a případně je umožněna určitá forma autonomie oproti instrukčnímu toku. Akcelerátory mohou mít podobu autonomních jader se specifickou funkcí nebo jednotek s pevnou funkcí. Příkladem mohou být akcelerátory pro vstupně-výstupní zařízení [1].

1.3 Běhová prostředí a vlákna

Vykonání programu je závislé na běhovém prostředí (EEI). To určuje počet a druh hardwarových vláken a jejich podporované úrovně běhu, chování instrukcí v každém vlákně, počáteční stav programu, způsob obsluhy výjimek a přerušení a dostupnost částí paměti či vstupně-výstupních zařízení.

Běhové prostředí může mít různou implementaci, která může být pouze hardwarová, pouze softwarová nebo může být jejich kombinací. Běhové prostředí vnímá vlákno jako samostatnou jednotku, která čte a zpracovává instrukce. Některá běhové prostředí můžou vytvářet další virtuální vlákna tím, že v čase rozprostřou zpracovávání těchto vláken na méně hardwarových vláken.

Příkladem běhového prostředí je prostředí definované přímo v hardwaru, kde jsou vlákna implementována ve formě fyzických vláken v procesoru. Další možností je operační systém s podporou RISC-V, který umožňuje víceuživatelské běhové prostředí a spravuje přístup do paměti přes virtualizaci. Dále existují emulátory a hypervizory RISC-V. Emulátor umožňuje běh RISC-V vláken na jiné architektuře, např. na systémech s architekturou x86. Hypervizor RISC-V používá více běhových prostředí na úrovni správce pro jednotlivé operační systémy [1].

1.4 Základní instrukční sady

Instrukční sada RISC-V se dělí na ISA na úrovni uživatele, která je definována jako základní instrukční sada pro celočíselné operace a volitelná rozšíření k této základní sadě, a privilegovanou ISA, která slouží pro běh systému při různých úrovních oprávnění a určuje i komplexitu systému. Základní instrukční sada na úrovni uživatele musí být přítomná ve všech možných implementacích architektury RISC-V. Nutnou součástí každé implementace architektury RISC-V je také část privilegované ISA na úrovni stroje a další části jsou volitelné [5]. Následující text bude popisovat ISA na úrovni uživatele. Architekturám s více úrovněmi oprávnění se budou věnovat podkapitoly 1.14 a 1.15.

Standard RISC-V definuje ve skutečnosti čtyři základní instrukční sady, které jsou navzájem rozlišeny délkou a počtem celočíselných registrů a velikostí adresovatelného prostoru. Základní instrukční sady jsou pojmenovány jako I. Před písmeno je připojen prefix RV s počtem bitů architektury. Obsahují výpočetní instrukce pro celá čísla, načítání a ukládání celých čísel a instrukce pro řízení toku.

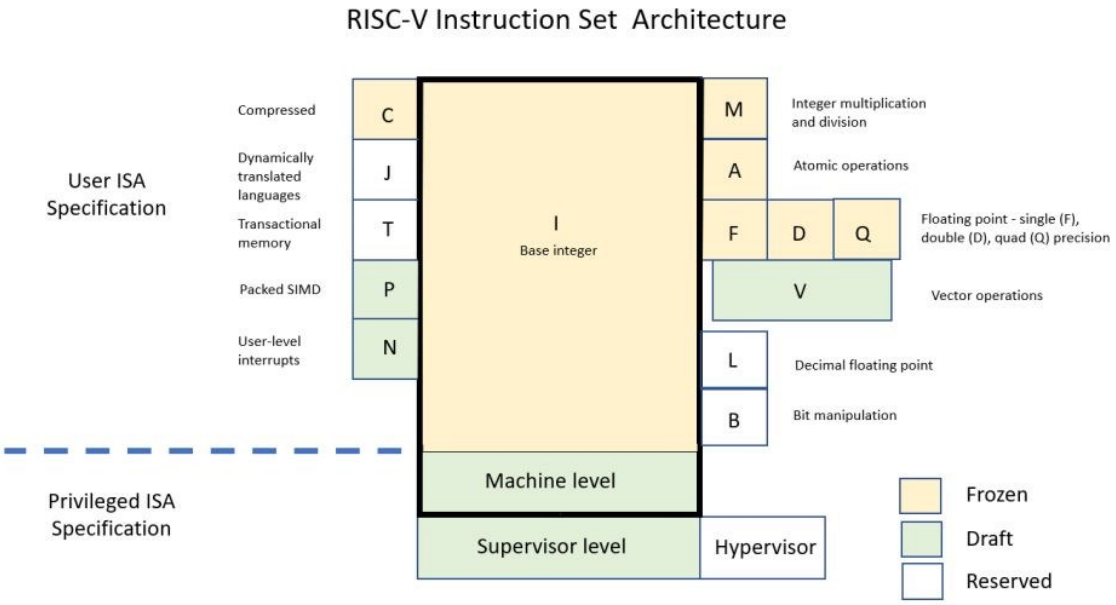
Dvě základní instrukční sady se jmenují RV32I a RV64I a poskytují 32bitovou a 64bitovou architekturu. Další základní instrukční sada je RV32E, která se od RV32I liší poloviční počtem registrů a je určená pro použití v mikrokontrolerech. Poslední varianta je RV128I s podporou 128bitového adresního prostoru.

Každá tato základní sada může být rozšířena jednou nebo více instrukčními sadami. Tyto rozšiřující sady mají svoje kódování, které je rozděleno do tří disjunktních kategorií: standardní, rezervované a vlastní (custom). Standardní kódování jsou definována nadací RISC-V Foundation a neměla by být navzájem v konfliktu. Rezervovaná kódování nejsou v současnosti definovaná, ale jsou připravená pro možné budoucí použití v rámci standardních kódování.

Rozšíření, které není definováno nadací RISC-V Foundation, se nazývá jako nestandardní. Vlastní kódování by nemělo být nikdy použité pro standardní rozšíření a je možné se s ním

setkat u nestandardních rozšíření jednotlivých výrobců. Pokud nestandardní rozšíření používá standardní nebo rezervované kódování, tak je označeno jako nevyhovující.

Standardní instrukční sada a standardní rozšíření RISC-V se v budoucnosti nebudou měnit a přidávání dalších instrukcí bude probíhat přes přidávání dalších volitelných rozšíření [1].



Obrázek 1: Základní instrukční sada a její standardní rozšíření [5]

1.4.1 Sada standardních rozšíření

Standardní rozšíření jsou definována za účelem podpory vývoje softwaru a poskytují například operace pro celočíselné dělení a násobení, atomické operace a aritmetické operace s plovoucí desetinnou čárkou a další možnosti. Rozšíření mohou být rozdělena podle toho, zda již byly jejich specifikace ratifikovány, nebo mohou být v budoucnu změněny (malé změny očekávány u označení frozen, větší u označení draft). Mezi rozšíření, která už mají ratifikovanou specifikaci patří následující: M, A, F, D, Q, C, Zifencei, Zicsr. Podpora jednotlivých rozšíření se přidává jako přípona k názvu architektury, např. RV32IMAFD.

Standardní rozšíření pro násobení a dělení celých čísel má pojmenování M a přidává instrukce pro násobení a dělení hodnot v celočíselných registrech. Další standardní rozšíření se jmenuje A, obsahuje instrukce pro atomické čtení, modifikaci a zápis do paměti pro synchronizaci mezi několika vlákny RISC-V, která používají stejný paměťový prostor. Rozšíření pro čísla s plovoucí desetinnou čárkou v jednoduché přesnosti se značí pomocí F, přidává registry pro čísla s plovoucí desetinnou čárkou a instrukce pro výpočty a pro nahrání

a uložení hodnot do těchto registrů. Rozšíření pro čísla s plovoucí desetinnou čárkou v dvojité přesnosti se označuje D, přidává registry s dvojitou přesností a obsahuje stejné typy instrukcí jako u rozšíření F. Kombinace rozšíření M, A, F a D spolu se základní ISA se značí zkratkou G. Standardní rozšíření C poskytuje instrukce v 16bitové formě. Standardní rozšíření Q poskytuje registry o délce 128 bitů a instrukce, které s nimi pracují.

Poslední standardní rozšíření, která jsou ratifikována, se nazývají Zifencei a Zicsr. Rozšíření Zifencei přidává instrukci FENCE.I, která slouží k explicitní synchronizaci mezi zápisy do instrukční paměti a čtením instrukcí na stejném vlákně. Rozšíření Zicsr přidává podporu instrukcí pro práci s kontrolními a stavovými registry (využití primárně na architektuře s více úrovněmi běhu) [1]. Seznamy instrukcí pro základní instrukční sady a jejich ratifikovaná standardní rozšíření, vytvořené s pomocí technické specifikace [1], je možné nalézt na konci této práce v sekci pro přílohy. Na první straně přílohy jsou ukázky typů formátů instrukcí a jejich příklady pro sadu RV32I převzaté z této specifikace.

1.4.2 Neratifikovaná standardní rozšíření

Mezi standardní rozšíření, jejichž specifikace ještě nebyla ratifikována a je možné, že dojde ke změně, patří např. následující [6]:

- **L:** přidává podporu desetinných čísel o základu deset.
- **B:** přidává podporu operací pro manipulaci s bity.
- **J:** přidává podporu dynamicky překládaných jazyků.
- **T:** přidává podporu operací transakční paměti.
- **P:** přidává podporu instrukcí Packed-SIMD.
- **V:** přidává podporu vektorových operací.
- **N:** přidává podporu přerušení na uživatelské úrovni oprávnění.
- **H:** rozšíření pro hypervizor.

1.5 Kódování instrukcí

Standardní RISC-V kódování je navržené pro instrukce s proměnnou délkou, kde každá instrukce může mít celkovou velikost o libovolném počtu násobků 16 bitů. Základní RISC-V ISA má instrukce o velikosti 32 bitů.

Pro konkrétní implementaci architektury RISC-V jsou dány hodnoty IALING a ILEN. Hodnota IALING označuje omezení na zarovnání instrukcí, které daná implementace vyžaduje, a je možné mít zarovnání po 32 bitech (pro standardní ISA) nebo po 16 bitech (pro některá

rozšíření). Hodnota ILEN určuje maximální délku instrukce, která je podporována na dané implementaci RISC-V a představuje násobek hodnoty IALIGN. Pro základní ISA je také 32 bitů.

Základní RISC-V ISA má paměťový systém buď jako little-endian nebo big-endian. Instrukce jsou v paměti uložené jako sekvence 16bitových little-endian bloků, bez ohledu na endianitu paměťového systému. Bloky jedné instrukce jsou uložené na zvyšujících se adresách po 16 bitech. Blok s nejnižší adresou obsahuje nejnižší bity ve specifikaci instrukce.

Společná pravidla pro kódování instrukcí určitých délek je možné vidět na obrázku č. 2. Všechny 32bitové instrukce základní ISA jsou zarovnány do dvou 16bitových bloků a mají dva nejnižší bity nastaveny na 11. U redukovaných instrukcí o 16 bitech jsou poslední dva bity kódovány na 00, 01 nebo 10. Význam dalších bitů záleží na typu formátu instrukce. Kódování instrukcí s velikostí nad 32 bitů má další pravidla, která jsou v současnosti považována za rezervovaná a nejsou ve stavu frozen (mohou být ještě podstatně změněna). Pro 48bitové instrukce jsou nejnižší bity nastaveny na 011111 a pro 64bitové jsou nastaveny na 0111111. Pro instrukce o délce mezi 80 a 176 bity jsou nejméně významné bity nastaveny na 1111111 a hodnota na dvanáctém až čtrnáctém nejnižším bitu určuje počet 16bitových bloků nad rámec základních pěti. Pokud je tato hodnota nastavena na 111, pak toto kódování označuje instrukce delší než 192 bitů, které mohou být v budoucnu přidány [1].

Obrázek 2: Pravidla pro kódování instrukcí [1]

1.6 Registry

Základní ISA RISC-V dává programátorovi k dispozici 32 univerzálních registrů s délkou danou počtem bitů architektury. Registry x1 až x31 jsou registry pro držení celočíselné hodnoty

a registr x0 obsahuje vždy konstantu nula. RISC-V nemá specificky určený registr pro návratovou adresu subrutiny, ale konvence předpokládá použití registru x1 [1]. Registry x0 až x31, jejich symbolické názvy a popis jsou dány následovně:

Tabulka 1: Seznam základních registrů [1]

Registr	Název v asemblerovém jazyku	Popis	Zodpovědnost za uložení
x0	zero	Vždy vrací nulu	-
x1	ra	Návratová adresa	Volající
x2	sp	Ukazatel zásobníku	Volaný
x3	gp	Globální ukazatel	-
x4	tp	Ukazatel vlákna	-
x5	t0	Dočasný registr / Alternativní návratová adresa	Volající
x6–x7	t1–t2	Dočasný registr	Volající
x8	s0/fp	Registr pro uložení / ukazatel rámce	Volaný
x9	s1	Registr pro uložení	Volaný
x10–x11	a0–a1	Argumenty funkcí / návratové hodnoty	Volající
x12–x17	a2–a7	Argumenty funkcí	Volající
x18–x27	s2–s11	Registry pro uložení	Volaný
x28–x31	t3–t6	Dočasné registry	Volající

Standardní rozšíření F, D a Q pro operace s plovoucí desetinnou čárkou přidávají 32 registrů pro desetinná čísla mající délku 32, 64 nebo 128 bitů [1], resp. Názvy registrů, jejich alternativní názvy v asemblerovém jazyce, popis konvence využití registrů a zodpovědnost za uložení jsou uvedeny v následující tabulce:

Tabulka 2: Seznam registrů pro desetinná čísla [1]

Registr	Název v asemblerovém jazyku	Popis	Zodpovědnost za uložení
f0–f7	ft0–ft7	Dočasné registry	Volající
f8–f9	fs0–fs1	Registry pro uložení	Volaný
f10–f11	fa0–fa1	Argumenty funkcí/ návratové hodnoty	Volající
f12–f17	fa2–fa7	Argumenty	Volající
f18–f27	fs2–fs11	Registry pro uložení	Volaný
f28–f31	ft8–ft11	Dočasné registry	Volající

Přítomný je také registr Program Counter (PC), který obsahuje adresu aktuální instrukce. Architektura RISC-V dále definuje adresovací prostor až pro 4096 kontrolních a řídících registrů (CSR), ale instrukce pro přístup k nim už nejsou součástí základního ISA [1].

1.7 Práce s pamětí

Hardwarové vlákno RISC-V má jeden adresovací prostor o velikosti 2^n, kde n je počet bitů architektury. Slovo je definováno jako 32 bitů, půlslovo (halfword) jako 16 bitů, dvojslovo (doubleword) jako 64 bitů a čtyřslovo (quadword) jako 128 bitů. Paměťový prostor je cyklický, přičemž operace s adresami ignorují přetečení a místo toho používají modulo 2^n.

Mapování hardwarových prostředků do adresního prostoru vlákna je určeno běhovým prostředím. Různé adresové rozsahy v paměti mohou obsahovat buď volná místa (tyto adresy nejsou nikdy dostupné) nebo hlavní paměť, případně jedno nebo více vstupně-výstupních zařízení. Pokud má RISC-V platforma více vláken, adresní prostor libovolných dvou vláken může být sdílený, částečně sdílený či samostatný. Sdílené prostředky mohou být mapovány jak na stejné, tak i na jiné adresy.

Při vykonává instrukcí se provádí přístupy do paměti, a to buď explicitní či implicitní. Pro každou instrukci se provádí implicitní čtení z paměti (instruction fetch). Běhové prostředí může dále určovat další implicitní přístup do paměti, jako například při převodu adres. Instrukce na načtení (např. LW) a uložení (např. SB) představují explicitní přístup do paměti. Běhové prostředí určuje, kterou část paměti je možné zpřístupnit pro daný typ přístupu do paměti. Pokud se instrukce snaží přistoupit do paměti, ke které nemá povolený přístup, pak je vyvolána výjimka.

Implicitní čtení, které nevyvolá výjimku a nemá vedlejší účinky, se může vykonat spekulativně a libovolně brzy, a to i před tím, než je prokázáno, že čtení bude potřeba. Protože je možné, že by se mohlo číst až po zápisu do stejné paměti, a tím by nebyly instrukce řádně seřazené, je nutné toto ošetřit pomocí softwaru nebo kontrolou vyrovnávací paměti (instrukce FENCE.I). Příkladem tohoto chování může být načtení dat najednou, aby poté nebylo potřeba znova přistupovat do paměti při každé instrukci.

Přístupy do paměti vykonané jedním vláknem se můžou pro jiné vlákno zobrazit v jiném pořadí. Přeuspořádání instrukcí je ale omezeno pomocí modelu konzistence paměti. Pro RISC-V je výchozím modelem RVWMO (RISC-V Weak Memory Ordering). Další omezení přeuspořádání instrukcí může být omezeno běhovým prostředím. Stejně jako u implicitního čtení je konzistence zajištěna kontrolou vyrovnávací paměti nebo softwarem [1].

1.8 Instrukční sada RV32I

Instrukcí v instrukční sadě RV32I je 40, mají standardně délku 32 bitů a patří do jednoho z šesti následujících formátů, z nichž dva se liší významem bloků:

- **R-formát:** instrukce používá tři registry, dva jako zdroj dat a jeden cílový – příkladem jsou instrukce pro aritmetické či logické operace (ADD, XOR).
- **I-formát:** instrukce používá dva registry, jeden zdrojový, který se kombinuje se 12bitovou hodnotou zakódovanou jako součást instrukce, a jeden cílový registr, příkladem jsou instrukce pro načtení nebo aritmetické operace (LW, ADDI).
- **S-formát:** instrukce používá dva registry, jeden obsahuje data a druhý se kombinuje se zakódovanou hodnotou pro určení cílové adresy v paměti, instrukce pro ukládání (SW, SB).
- **SB-formát:** struktura stejná jako u S-formátu, ale je určen pro větvící instrukce (BEG, BGE). Označován také jen jako B-formát.
- **U-formát:** instrukce pro načtení větších konstant – velikost 20 bitů (LUI, AUIPC).
- **UJ-formát:** stejná struktura jako u U-formátu, instrukce nepodmíněného skoku (JAL). Označován také jen jako J-formát.

V následujících podkapitolách budou představeny jednotlivé formáty a seznam k nim patřících instrukcí pro základní 32bitovou ISA doplněné o popis jejich funkcí [1], [7].

1.8.1 R-formát instrukcí

Instrukce R-formátu je rozdělena na bloky s následujícími názvy a velikostmi (zleva doprava od 31. do 0. bitu):

Tabulka 3: R-formát instrukcí [7]

funct7	rs2	rs1	funct3	rd	opcode
7 bitů	5 bitů	5 bitů	3 bity	5 bitů	7 bitů

Každé pole má význam neznaménkového celého čísla. Může reprezentovat jakékoliv číslo v daném rozsahu (například 0–31 pro 5bitový blok). Bloky funct7 a funct3 společně s opcode popisují operaci, která se má vykonat. Protože je opcode pro R-formát pevně daný, pak maximální počet operací je určen variacemi funct7 a funct3, tedy může být zakódováno maximálně 1024 instrukcí. Blok rs1 určuje první zdrojový registr, blok rs2 druhý. Blok rd stanovuje registr pro uložení výsledku operace [7].

Seznam instrukcí v R-formátu [1]:

- **ADD:** součet hodnot v registrech rs1 a rs2, který se uloží do cílového registru.
- **SUB:** od hodnoty v rs1 odečte hodnotu rs2, výsledek uloží do cílového registru**.**
- **SLT:** do cílového registru nastaví 1, pokud je rs1 menší než rs2, jinak nastaví 0.
- **SLTU:** jako SLT, ale pro neznaménková čísla.
- **XOR:** do cílového registru uloží exkluzivní disjunkci bitů mezi registry rs1 a rs2.
- **OR:** do cílového registru uloží bitový součet mezi hodnotami v rs1 a rs2.
- **AND:** do cílového registru uloží bitový součin mezi hodnotami rs1 a rs2.
- **SLL:** logický posun hodnoty rs1 doleva o hodnotu uloženou v nejnižších 5 bitech registru rs2.
- **SRL:** logický posun hodnoty rs1 doprava o hodnotu uloženou v nejnižších 5 bitech registru rs2.
- **SRA:** aritmetický posun hodnoty rs1 doprava o hodnotu uloženou v nejnižších 5 bitech registru rs2.

1.8.2 I-formát instrukcí

Formát těchto instrukcí je podobný R-formátu, na rozdíl od něho ale používá pouze dva registry. Instrukce I-formátu je rozdělena na bloky s následujícími názvy a velikostmi (název záleží na typu instrukce):

Tabulka 4: I-formát instrukcí [7]

imm (konstanta)	rs1	funct3	rd	opcode
offset	base	width	dst	LOAD
12 bitů	5 bitů	3 bity	5 bitů	7 bitů

Blok imm je určen pro uložení konstanty – 12bitového čísla. Protože jsou operace prováděné pouze se slovy, je tato konstanta před výpočtem rozšířena na 32 bitů. Blok opcode společně s funct3 specifikují typ operace. Blok rs1 označuje registr použitý jako operand, blok rd registr pro uložení výsledné hodnoty.

Dále je tento formát použit i pro instrukce načtení dat z paměti. Bloky mají stejnou velikost, pouze se změní jejich význam oproti instrukcím pro operace. Offset (celé číslo se znaménkem) je přičten k základní adrese (base) pro vytvoření adresy, ze které se bude číst. Načtená hodnota se uloží do cílového registru (dst). Blok LOAD má pro všechny instrukce načtení hodnotu 0000011. Blok width určuje typ operace načtení [7].

Seznam instrukcí v I-formátu [1]:

- **ADDI:** součet hodnoty v registru rs1 a 12bitové konstanty imm, který se uloží do cílového registru.
- **SLTI:** do cílového registru nastaví 1, pokud je rs1 menší než imm, jinak nastaví 0.
- **SLTIU:** jako SLTI, ale pro neznaménková čísla.
- **XORI:** do cílového registru uloží exkluzivní disjunkci bitů mezi rs1 a konstantou imm.
- **ORI:** do cílového registru uloží bitový součet mezi rs1 a konstantou imm.
- **ANDI:** do cílového registru uloží bitový součin mezi rs1 a konstantou imm.
- **LW:** načte slovo na adrese určené součtem base a offsetu, hodnotu uloží do registru dst.
- **LB:** načte bajt na adrese určené součtem base a offsetu, znaménkově ho rozšíří na délku slova.
- **LH:** načte půlslovo na adrese určené součtem base a offsetu, hodnota je před uložením znaménkově rozšířena na celé slovo.
- **LBU:** jako LB, ale načte neznaménkový bajt, před uložením je hodnota doplněna nulami pro vyplnění 32bitového registru.
- **LHU:** jako LH, ale načte neznaménkové půlslovo, před uložením je hodnota doplněna nulami pro vyplnění 32bitového registru.

- **JALR:** příkaz skoku na adresu určenou součtem hodnoty v registru rs1 a offsetu (nejméně významný bit součtu nastaven na 0), do cílového registru rd se uloží hodnota registru PC + 4.

Instrukce pro bitový posun nepoužívají celou 12bitovou konstantu, ale pouze 5 nejnižších bitů. Tato hodnota se označuje jako shamt [1]:

- **SLLI:** logický posun hodnoty v rs1 doleva o 5bitovou konstantu, uložení do cílového registru.
- **SRLI:** logický posun hodnoty v rs1 doprava o 5bitovou konstantu, uložení do cílového registru.
- **SRAI:** aritmetický posun hodnoty v rs1 doprava o 5bitovou konstantu, uložení do cílového registru.

Do skupiny instrukcí v I-formátu dále náleží systémové instrukce [1]:

- **ECALL:** instrukce se používá k žádosti o službu do běhového prostředí.
- **EBREAK:** instrukce slouží pro návrat řízení do ladícího prostředí.

Jako poslední k I-formátu patří instrukce **FENCE**, která slouží k uspořádání přístupů k paměti a vstupně-výstupním zařízením mezi jednotlivými hardwarovými vlákny či koprocesory. Instrukce rozdělí kód na set předcházejících operací a set následujících operací a zajistí, že pro ostatní vlákna nebo externí zařízení nebudou viditelné operace ze setu následníků před operacemi ze setu předchůdců [1].

1.8.3 S-formát instrukcí

Tento formát se používá pro instrukce uložení. Používají se dva registry, rs1 pro základní adresu a rs2 pro uložení dat. V bloku imm se dále určuje offset. Tento blok byl rozdělen a je na místech funct7 a rd bloků z R-formátu instrukcí. Místo funct7 je dáno nejvyšších 7 bitů offsetu a zbylých 5 bitů je definováno místo rd bloku [7].

Tabulka 5: S-formát instrukcí [7]

imm [11:5]	rs2	rs1	funct3	imm [4:0]	opcode
7 bitů	5 bitů	5 bitů	3 bity	5 bitů	7 bitů

Seznam instrukcí v S-formátu [1]:

- **SW:** hodnota z registru rs2 se uloží na adresu určenou součtem hodnoty v rs1 a imm.
- **SB:** nejnižší bajt z rs2 se uloží na adresu určenou součtem hodnoty v rs1 a imm.

- **SH:** půlslovo na nižší adrese z rs2 se uloží na adresu určenou součtem hodnoty v rs1 a imm.

1.8.4 SB-formát instrukcí

Tento formát určuje instrukce pro větvení. Stanovuje dva registry pro porovnání a offset (imm blok). Výpočet adresy, na které se má pokračovat s vykonáváním instrukcí, se určuje na základě registru Program Counter (PC). Když neskočíme, pak PC = PC + 4 bajty (tedy o jednu instrukci vpřed). V případě skoku je výpočet následující adresy PC změněn na PC + (imm * 4). Blok určuje počet bajtů, o které se máme posunout, a to buď dopředu, v případě kladného čísla, či dozadu, v případě záporného. Protože RISC-V podporuje rozšíření pro 16bitové instrukce, pak musí být přesun dělán přes půlslova místo slov, a to i v případě, že nejsou přítomny žádné 16bitové instrukce či instrukce s volitelnou délkou, které také používají posuny po 16 bitech. Instrukce skoků mohou tedy na 32bitové architektuře dosáhnout pouze na 1024 32bitových adres na obou stranách od hodnoty registru PC. V případě, že je další instrukce dále než 1024 adres, pak musí použít další instrukci skoku jako most.

SB-formát je podobný jako S-formát, definuje dvojici registrů rs1 a rs2 a dále offset. Ten reprezentuje 12bitové číslo. Offset má ve skutečnosti velikost 13 bitů, ale nultý bit je vždy roven 0, a proto není kódován. V první části imm bloku je určen 12. a dále 10. až 5. bit offsetu, v druhé části je určen 4. až 1. bit a následně 11. bit offsetu [7].

Seznam instrukcí v SB-formát [1]:

- **BEQ:** pokud jsou hodnoty v rs1 a rs2 stejné, provede se skok.
- **BNE:** pokud jsou hodnoty v rs1 a rs2 rozdílné, provede se skok.
- **BLT:** pokud je hodnota v rs1 menší než hodnota v rs2, provede se skok.
- **BGE:** pokud je hodnota rs1 stejná nebo větší než hodnota v rs2, provede se skok.
- **BLTU:** jako BLT, ale pro neznaménková čísla.
- **BGEU:** jako BGE, ale po neznaménková čísla.

1.8.5 U-formát instrukcí

Protože I-formát instrukcí umožňuje používat pouze konstanty o velikosti 12 bitů, byl vytvořen tento formát pro nastavení 20 horních bitů konstanty. Formát instrukcí je následující (zleva doprava od 31. bitu do nultého) [7]:

Tabulka 6: U-formát instrukcí [7]

imm	rd	opcode
20 bitů	5 bitů	7 bitů

Blok rd značí cílový registr. U-formát mají dvě instrukce [1]:

- **LUI:** instrukce pro nastavení horních 20 bitů 32bitového čísla v cílovém registru, spodních 12 bitů nastaveno na 0.
- **AUIPC:** instrukce slouží k sestavení adresy relativní k registru PC. Konstanta nastaví horních 20 bitů adresy, spodních 12 je vyplněno nulami. Tato hodnota je přidána k adrese instrukce AUIPC a výsledek je uložen do cílového registru.

1.8.6 UJ-formát instrukcí

V případě skoků v instrukcích větvení se předpokládají kratší skoky a ty jsou vyjádřeny přičtením hodnoty k registru PC. Tyto skoky ale mají omezený dosah. Pro skoky s větším dosahem můžeme použít instrukci **JAL** používající formát UJ, který má stejná pole s formátem U (liší se kódováním konstanty). Je potřeba specifikovat 20bitovou konstantu (offset), opcode a registr pro návratovou adresu. Instrukce JAL uloží PC + 4 do registru rd a následně zvýší hodnotu v registru PC o offset. V konstantě může být zakódován skok až na $\pm 2^{19}$ lokací, které mezi sebou mají vzdálenost dva bajty [7].

1.9 Instrukční sada RV64I

Instrukční sada RV64I obsahuje stejné instrukce jako RV32I a přidává nové instrukce dostupné pouze na 64bitové architektuře. Instrukce známé z RV32I nově pracují s 64bitovými registry, hodnotami a adresovým prostorem. RV64I obsahuje nové instrukce pro manipulaci 32bitových čísel, které ignorují horních 32 bitů a vždy vytváří 32bitové znaménkové hodnoty a výsledek rozšiřují znaménkově na 64 bitů. Tyto nové instrukce mají stejné názvy jako na RV32I a přidávají příponu W:

- ADDIW.
- SLLIW.
- SRLIW.
- SRAIW.
- ADDW.
- SUBW.

- SLLW.
- SRLW.
- SRAW.

Instrukce pro načtení LW, LB a LH čísla nově rozšiřují znaménkově na 64 bitů. Instrukce LHU a LBU je rozšiřují pomocí nul. Dále jsou přítomny nové instrukce pro práci s pamětí:

- **LD:** načtení dvojslova (64 bitů) z paměti na adrese rs1 + offset do registru rd.
- **SD:** uložení dvojslova z registru rs2 do paměti na pozici rs1 + offset.
- **LWU:** načtení neznaménkového slova (32 bitů) z paměti a jeho uložení do registru rd s rozšířením pomocí nul.

Další změnou v instrukcích oproti RV32I jsou operace pro bitový posun (SLLI, SRLI, SRAI), které se zapisují stejně, ale nově se hodnoty posunu (shamt) kódují pomocí šesti nejnižších bitů místo pěti [1].

1.10 Pseudoinstrukce

Pseudoinstrukce jsou instrukce dostupné v zápisu asemblerového jazyka pro RISC-V, ale samotné nemají podporu hardwarové implementace v ISA na RISC-V procesorech. Pouze reprezentují zjednodušený zápis jiných instrukcí a při kompilaci programu se na tyto instrukce přeloží. Po překladu může být pseudoinstrukce reprezentována jednou nebo i více instrukcemi. Dále záleží také na programovém kontextu, ve kterém se pseudoinstrukce nachází, a tedy i stejný zápis pseudoinstrukce může být přeložen na různé instrukce či jejich posloupnosti. Příkladem může být pseudoinstrukce LI pro načtení 32bitového čísla, která se překládá buď na instrukce ADDI, pokud je pro číslo použito 12 bitů a méně, a pokud je větší, tak se překládá na dvojici instrukcí LUI a ADDI.

Seznam vybraných pseudoinstrukcí [8]:

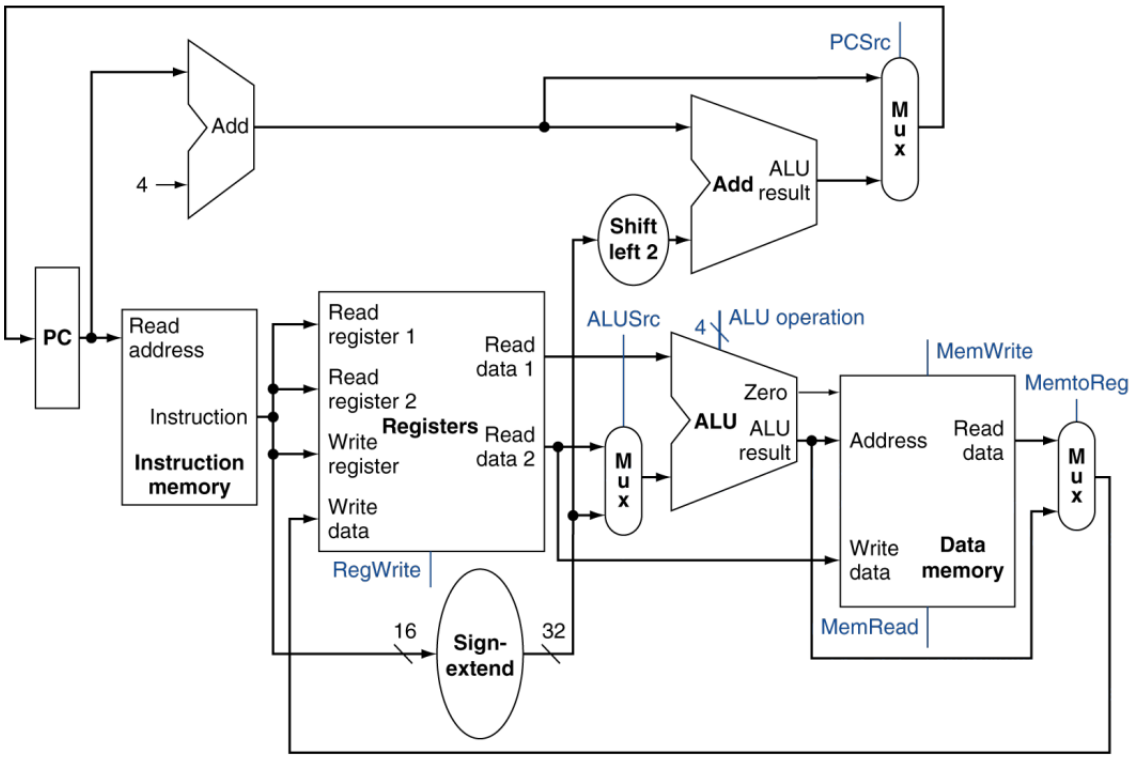
- **LI:** načtení 32bitové konstanty.
- **LA:** načtení 32bitové adresy.
- **MV:** zkopíruje hodnotu v registru sd do registru rd.
- **NOT:** jednotkový doplněk registru.
- **NEG:** negace hodnoty v registru.
- **BGT:** skok, pokud je rs1 větší než rs2.
- **BLE:** skok, pokud je rs1 menší nebo rovný rs2.
- **BGTU:** jako BGT, pro neznaménková čísla.
- **BEQZ:** skok, pokud je rs1 roven nule.

- **BLEZ:** skok, pokud je rs1 menší nebo roven nule.
- **BGTZ:** skok, pokud je rs1 větší než nula.
- **J:** nepodmíněný skok.
- **CALL:** zavolání subrutiny.
- **RET:** návrat ze subrutiny.
- **NOP:** prázdná instrukce.

1.11 Dekódování a zpracování instrukcí

Pro popsání dekódování a zpracování instrukcí je možné zvolit např. jedno-cyklovou implementaci RISC-V architektury (bez pipeline) zobrazenou na obrázku č. 3. Tato implementace se skládá z následujících částí [9]:

- Blok PC označuje Program Counter, který určuje adresu vykonávaných instrukcí, přísluší k němu obvod pro nastavení adresy následující instrukce (blok Add se vstupem 4) a případně se k nové adrese přidá ještě dalšího hodnota, pokud je splněna podmínka v instrukcích skoku.
- Blok pro získání dané instrukce z paměti (Instruction memory), ze kterého jsou dále data posílána do několika bloků, a to do bloku registrů (Registers), případně do bloku pro rozšíření číselných konstant na délku slova a dále také do kontroléru, který spravuje přepínání multiplexorů (označeny modře) a určuje, které operace se mají vykonat.
- Blok registrů umožňuje získání hodnot vstupních registrů instrukce a také zápis výsledku do cílového registru.
- Blok aritmeticko-logické jednotky (ALU), jehož vstupy jsou určeny i multiplexorem. Na vstup mohou přijít buď hodnoty ze dvou registrů (např. instrukce R-formátu) nebo registru a konstanty zapsané v instrukci (např. instrukce I-formátu). ALU poté provede požadovanou operaci se vstupními daty.
- Blok pro čtení a zápis dat do paměti (Data memory), za kterým ještě následuje multiplexor určující, zda se má také provést zápis dat do cílového registru.



Obrázek 3: Jedno-cyklová implementace RISC-V [9]

Vykonávání instrukcí probíhá v pěti fázích: získání instrukce z paměti (fetch), dekódování instrukce, zpracování v ALU, přístup do paměti a zápis do registrů. Vykonávání jednotlivých instrukcí se liší podle jejich typu a některé fáze je možné vynechat.

V první fázi se provede inkrementace adresy pro získání adresy další instrukce a jsou získána data instrukce z aktuální adresy. Ve druhé fázi se provede čtení opcode a určí se typ instrukce a délky jednotlivých polí. Dále se provede čtení dat z potřebných registrů. V případě instrukcí jako ADD, XOR, podmíněných skoků či ukládání do paměti se provádí čtení dvou registrů, v případě práce s konstantou nebo pro čtení z dat z paměti se načtou data pouze z jednoho vstupního registru a v případě instrukce JAL se čtení z registrů neprovádí.

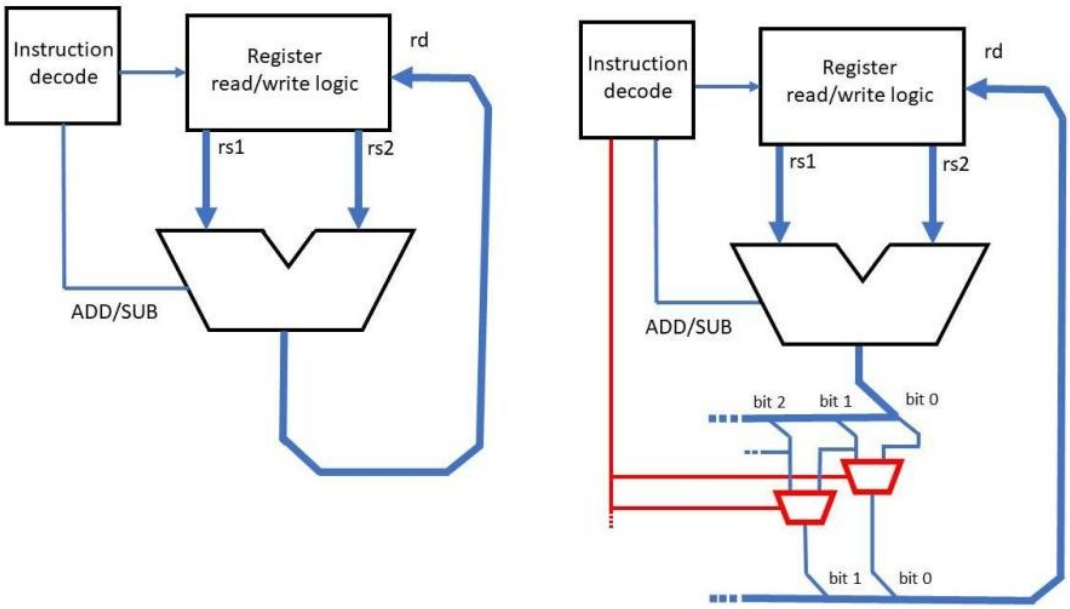
Ve třetí fázi se provedou aritmetické a logické operace, jako sčítání hodnot, bitové posuny, porovnávání hodnot apod. Čtvrtá fáze se provádí pouze pro instrukce čtení/zápisu do paměti. Ostatní instrukce buď nic nevykonávají nebo je tato fáze přeskočena. V poslední fázi se provádí zápis hodnot do cílového registru. Některé instrukce, jako jsou skoky či zápis dat do paměti, nic do registrů nezapisují a stejně jako v předchozí fázi je umožněna nečinnost či vynechání této fáze [9].

1.12 Vytvoření nových instrukcí

Pro jednoduché vytvoření nových instrukcí, které jsou modifikací už existujících, je možné využít znalosti formátů instrukcí podle 1.8. Z nich je patrné, že všechny instrukce obsahují 7bitový opcode na nejnižších bitech a kromě U-formátu a UJ-formátu i pole funct3 od 12. do 14. bitu. U R-formátu je přítomný ještě blok funct7 od 25. bitu do 31. Pro určení cílového registru jsou vždy použity bity od 20. do 24. Díky těmto konvencím pro ukládání kódu konstant a pro umístění ukazatelů na registry je možné vytvořit novou instrukci mapováním na nějaký z těchto formátů, a tím získat základ pro návrh RISC-V, přičemž hardwarová implementace je téměř hotová. Je to díky skutečnosti, že mnohé úkony, jako dekódování instrukcí a přístup k registrům, jsou už vytvořeny ve stávajícím návrhu architektury a je možné jich využít. Takto vzniklé instrukce se také nazývají jako brownfield rozšíření.

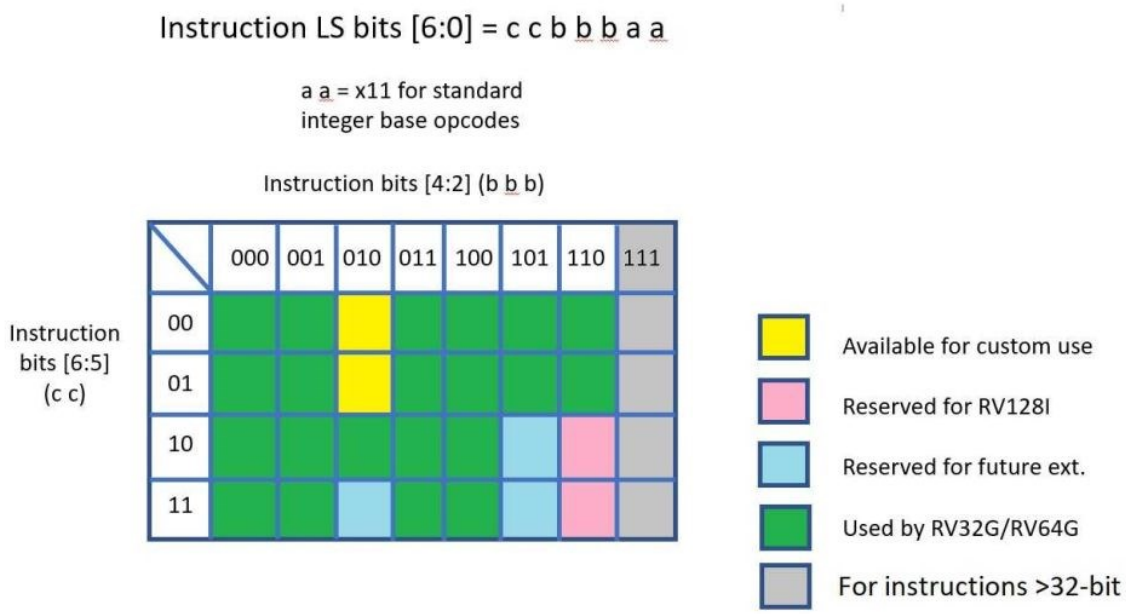
Příkladem takové instrukce může být průměr dvou čísel se zaokrouhlením dolů. Pro implementaci lze využít existujících instrukcí ADD a SUB. Tyto instrukce mají stejné pole opcode a funct3 a liší se pouze u pole funct7 (0100000 pro SUB ,0000000 pro ADD). Díky skutečnosti, že žádná další instrukce nepoužívá stejnou kombinaci opcode a funct3 polí, je možné při využití stejných opcode a funct3 polí zakódovat do pole funct7 až 126 dalších instrukcí, aniž by došlo ke shodě s už existující.

Jedna z možných implementací instrukcí ADD a SUB je ukázána na obrázku č. 4. Hodnoty ze vstupních registrů rs1 a rs2 jsou poslány do aritmeticko-logické jednotky a typ operace se určí podle 30. bitu instrukce. Následně je výsledek zapsán do cílového registru rd. Novou instrukci pro dolů zaokrouhlený průměr je možné implementovat například přes sérii multiplexorů, které budou ovládány libovolně zvoleným bitem v poli funct7 (kromě 30. bitu). Pokud tedy bude tento bit nastaven na 1, tak se kromě standardních operací sčítání a odčítání provede ještě bitový posun doprava o 1 bit (dělení dvěma). Tím získáme nové instrukce pro výpočet (rs1 + rs2) / 2 a také pro (rs1 - rs2) / 2. Díky tomuto přístupu je možné implementovat nové instrukce jen s minimálními změnami v hardware [5].



Obrázek 4: Přidání podpory nové instrukce do návrhu RISC-V procesoru [5]

Možností pro přidání dalších brownfield rozšíření je mnoho, protože základní RV32I ISA využívá pouze 11 ze 128 možných stavů pole opcode i díky využití další polí funct3 a funct7. Ostatní standardní rozšíření a instrukční sady pro 64bitové a 128bitové verze architektury vyžadují pouze omezené množství dalších opcode variací (výjimkou je rozšíření C, protože obsahuje instrukce s rozdílnou délkou). Volné variace druhého až šestého bitu dostupné na RV32, RV64 a RV128 architekturách s rozšířením G jsou znázorněny žlutou barvou na obrázku č. 5 [5].



Obrázek 5: Variace bitů opcode a jejich možnost použití pro kódování nových instrukcí [5]

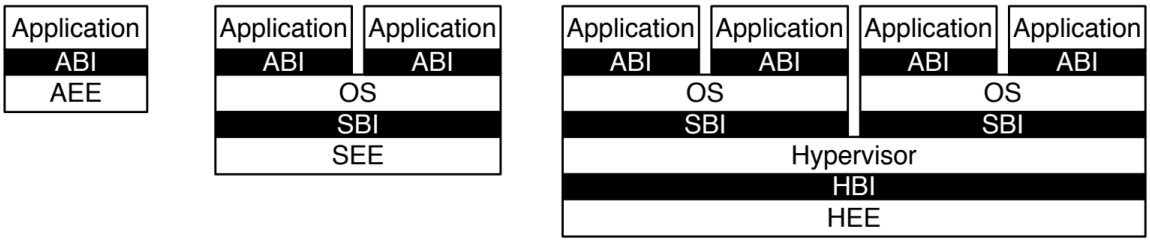
1.13 Výjimky a přerušení

Výjimka označuje vyvolání neobvyklého stavu při výkonu instrukce ve vlákně RISC-V. Přerušení označuje externí asynchronní událost, která může způsobit, že vlákno nečekaně předá řízení. Termín trap označuje předání kontroly na obslužný kód (trap handler), které může být způsobeno buď přerušením nebo výjimkou. Způsob zpracování předání kontroly a jeho viditelnost pro software záleží na běhovém prostředí. Z pohledu softwaru běhového prostředí mohou být čtyři typy předání kontroly.

První je contained trap, která je viditelná pro software v běhovém prostředí a je tímto softwarem i obsloužena. Druhou možností je requested trap. Je to synchronní výjimka, která představuje explicitní volání do běhového prostředí žádající vykonání určité akce jménem softwaru běhového prostředí. Příkladem může být systémové volání. Třetí je invisible trap, která je zpracována transparentně běhovým prostředím a výkon dalších instrukcí pokračuje normálním způsobem po jejím vykonání. Jde například o emulaci chybějící instrukce nebo zpracování přerušení zařízení pro odlišné programy. Při vykonávání si není software běhového prostředí vědom vyvolání invisible trap. Poslední možností je fatal trap, která představuje kritické selhání a způsobí, že běhové prostředí ukončí vykonávání [1].

1.14 RISC-V softwarové zásobníky

Jak bylo uvedeno v podkapitole 1.3, existují různé možnosti implementace architektury RISC-V, kdy běhové prostředí může být na úrovni hardwaru nebo pracuje pomocí supervizoru či hypervizoru. S běhovým prostředím je úzce spojena problematika softwarových zásobníků a privilegované ISA, která slouží k jejich obsluze. Na obrázku č. 6 je možné zhlédnout některé možné typy softwarových zásobníků. Vlevo je příklad jedné aplikace běžící v jednom aplikačním běhovém prostředí (AEE). Aplikace pro běh využívá aplikační binární rozhraní (ABI). ABI obsahuje ISA pro uživatelské úrovně běhu a sadu volání pro interakci s AEE (privilegovaná ISA stroje). ABI umožňuje větší flexibilitu implementace AEE, neboť skrývá její detaily. Stejné ABI může být implementováno pro různé hostitelské operační systémy nebo může být emulací uživatelské úrovně na stroji s jinou ISA [10].



Obrázek 6: Možné softwarové zásobníky podporované na architektuře RISC-V [10]

Prostřední konfigurace je běžný operační systém s podporou běhu více aplikací. Každá aplikace komunikuje skrz své ABI s operačním systémem. Operační systém podobně jako v minulém případě komunikuje s běhovým prostředím, tentokrát s běhovým prostředím supervizoru (SEE). Používá pro to binární rozhraní supervizoru (SBI). SBI obsahuje ISA pro uživatelské a privilegované úrovně běhu a sadu volání přes SBI (privilegovaná ISA supervizoru). Používání jedné SBI na všech SEE umožňuje jedinému obrazu operačního systému běžet na libovolné SEE. SEE může mít různé podoby, od virtuálních strojů na serverech až po zavaděč a vstupně-výstupní systém pro jednoduché hardwarové platformy. Třetím příkladem platformy jsou virtualizované operační systémy spravované hypervizorem, který komunikuje s běhovým prostředím hypervizoru (HEE) skrz binární rozhraní hypervizoru (HBI) [10].

1.15 Úrovně běhu

RISC-V hardwarové vlákno běží na určité úrovni oprávnění, které je zakódováno jako mód v řídících a stavových registrech (CSR). V současné době existují tři úrovně oprávnění popsané v následující tabulce:

Tabulka 7: Úrovně oprávnění v RISC-V [10]

Úroveň	Kódování	Jméno	Zkratka
0	00	Uživatel/Aplikace	U
1	01	Supervizor	S
2	10	Rezervováno	
3	11	Stroj	M

Úrovně oprávnění jsou použity pro zajištění ochrany jednotlivých částí softwarového zásobníku. Pokud v dané úrovni oprávnění je proveden pokus o provedení nepovolené operace, pak je vyvolána výjimka.

Úroveň stroje má nejvyšší oprávnění a je to jediná úroveň oprávnění, která je vyžadována v hardwarové platformě RISC-V. Kód, který běží ve strojovém módu (M-mode), je obvykle považován za důvěryhodný a má nízko-úrovňový přístup k hardwarové implementaci. Dále může být použit pro zabezpečené běhové prostředí. Mód supervizoru (S-mode) je použit pro operační systém a mód uživatele (U-mode) pro běžné aplikace. Každá úroveň oprávnění má základní sadu privilegovaných instrukcí a případně i další rozšíření a varianty. Například M-mode podporuje volitelné standardní rozšíření na ochranu paměti. Počet úrovní oprávnění může být různý – mezi 1 a 3. Výhodou méně oprávnění jsou snížené náklady na implementaci, nevýhodou je ale omezení izolace.

Nejjednodušší implementace RISC-V (běhové prostředí přímo na hardware) poskytuje pouze M-mód, který nemá žádnou ochranu proti škodlivému či chybnému kódu. Další implementace podporují i U-mód, jehož účelem je ochrana systému od uživatelského kódu. S-mód může být použit pro další izolaci mezi operačním systémem a běhovým prostředím supervizoru.

Hardwarové vlákno běžně pracuje v U-módu, dokud není vyvolána výjimka či přerušení, pak je vynuceno přepnutí na obsluhu, která pracuje v režimu s vyšším oprávněním. Vlákno vykoná obslužnou rutinu a poté je obnoveno provádění původního instrukčního toku v U-módu.

Předání řízení, které zvýší úroveň oprávnění, je nazývána vertical trap. Naopak předání, které zůstane na stejné úrovni, se nazývá horizontal trap [10].

Tabulka 8: Podporované kombinace úrovní oprávnění [10]

Počet úrovní	Podporované módy	Typické použití
1	M	Jednoduché vestavné systémy
2	M, U	Zabezpečené vestavné systémy
3	M, S, U	Systémy s OS založenými na Unixu

Implementace může také obsahovat ladící mód (D-mód) s podporou pro ladění mimo čip či testování při výrobě. D-mód má ještě více privilegií než M-mód. D-mód rezervuje pouze několik adres v řídících a stavových registrech, které jsou dostupné pouze v D-módu. Dále může rezervovat i část fyzické paměti platformy [10].

1.16 Porovnání s architekturou ARM

V této podkapitole bude představena konkurenční architektura ARM formou krátkého popisu a následně bude porovnána s architekturou RISC-V.

1.16.1 Architektura ARM

ARM je architektura s redukovanou instrukční sadou, v současnosti vyvíjenou firmou ARM Holdings. Architektura podporuje 32bitové a 64bitové instrukční sady. Využívaná je např. CPU chytrých telefonů, mikropočítačů nebo vestavných systémů [11]. Mezi základní vlastnosti architektury patří podpora multiprocesorových systémů, každá instrukce architektury má pevnou délku a vykonání instrukce trvá jeden cyklus. Architektura dále podporuje pipeline instrukcí, obsahuje sekci pro správu a ochranu paměti a různé typy registrů (celočíselné, desetinné, vektorové) pro zabránění zbytečných přesunů dat [12].

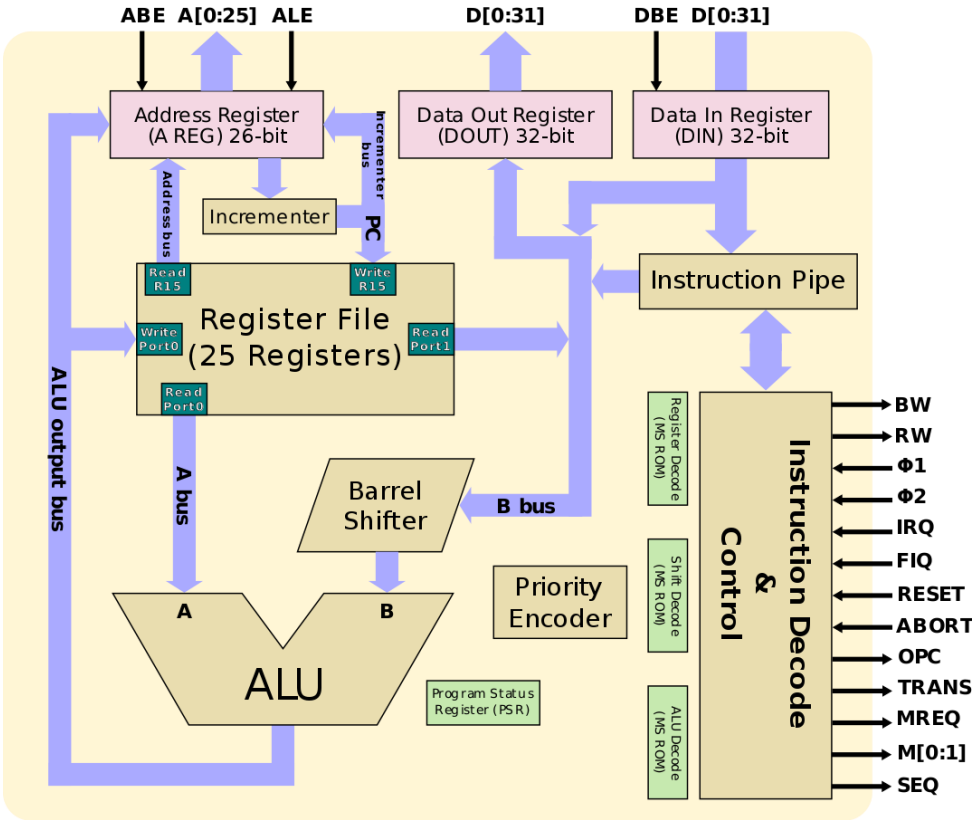
Architektura ARM byla vyvinuta ve firmě Acorn Computers v roce 1985. První počítač s touto architekturou byl Acorn Archimedes, uvedený v roce 1987. V roce 1990 byla založena společnost Advanced RISC Machines Ltd., jako společný projekt společností Acorn Computers, Apple Computer a VLSI Technology. Prvním telefonem využívajícím architekturu ARM se stala Nokia 6110. V roce 1998 společnost vstoupila na londýnskou burzu a změnila jméno na Arm Holdings [13].

Referenční manuál ARM neurčuje, jak přesně má být architektura implementována, místo toho definuje chování na abstraktním stroji. Pokud implementace odpovídá definovanému

chování, je možné takovou implementaci označit za jádra ARM. Jádra ARM jsou organizována do tzv. rodin, které určují danou architekturu pomocí dané verze instrukční sady. První rodina jader se jmenovala Arm1 a využívala architekturu pojmenovanou jako Armv1. Po Arm11 jsou rodiny nově pojmenovány Cortex.

S dalším vývojem přibylo kromě nových verzí i rozšíření Thumb v architektuře Armv4T, které umožňuje použití 16bitových instrukcí. Ve verzi architektury Armv8, která přinesla podporu 64bitové architektury, byly instrukční sady pro 64bitové a 32bitovou architekturu pojmenovány jako A64 a A32. Instrukční sada Thumb byla přejmenována na T32. ArmV8 zavedl exekuční stavy AArch32 a AArch64. V stavu AArch32 jsou používány instrukční sady A32 a T32 a pro stav AArch64 se používá instrukční sada A64 [14].

Architektura je v rodinách Cortex dále rozdělená do tří profilů: A, R a M [14]. Profil A se používá pro výkonné aplikace, jako je běh operačních systémů Windows nebo Linux. Profil R se používá pro aplikace s požadavky práce v reálném čase, jako jsou vestavné systémy nebo síťová zařízení. Profil M se používá pro mikrokontrolery, příkladem využití jsou zařízení internetu věcí [11].



Obrázek 7: Ukázka architektury z rodiny Arm1 [15]

1.16.2 Porovnání architektur

ARM i RISC-V jsou architektury s redukovanou instrukční sadou (RISC) a mají pouze omezené množství instrukcí. Dále to jsou architektury s topologií load-store, kdy procesor není chopen přímých operací nad daty v paměti a požaduje, aby tato data byla před operací přenesena do interních registrů [16]. Obě architektury podporují 32bitovou i 64bitovou instrukční sadu a možnost přidání modulů k procesoru pro zvýšení rychlosti provádění určitých operací [17].

Prvním rozdílem je, že základní sada RISC-V obsahuje pouze základní instrukce a celočíselné registry a základní modul ARM naopak podporuje i složitější operace, jako jsou násobení či práce s desetinnými čísly nebo vektorovými registry. Druhým rozdílem je, že architektura ARM podporuje adresování paměti jak pomocí big endian, tak little endian, a architektura RISC-V podporuje pouze little endian [17].

Dalším rozdílem RISC-V oproti ARM je, že RISC-V je open-source a ARM architektura je proprietární. Integrace ARM procesoru do systému vyžaduje zaplacení poplatku holdingu ARM v podobě licence. ARM je možné licencovat dvěma způsoby. První umožňuje licencovat samotné jádro vyvinuté společností ARM Holdings a druhou možností je licencovat samotnou architekturu ARM a vyvinout vlastní čip, který ale musí být kompatibilní s binárními soubory pro ARM. Příkladem prvního přístupu je čip ARM Cortex M3 a druhý přístup reprezentují např. procesory Apple série A nebo M [17]. Použití RISC-V je naopak bezplatné a nevyžaduje získání licence. Nevýhodou je ale chybějící podpora pro vývoj a integraci, kdy naopak ARM má k dispozici týmy specialistů na vývoj hardwarových systémů, které usnadní návrhářům integraci ARM procesoru. Dalším rozdílem je, že ARM může být blokován vládami, neboť je proprietární. Naopak standard RISC-V je dostupný každému.

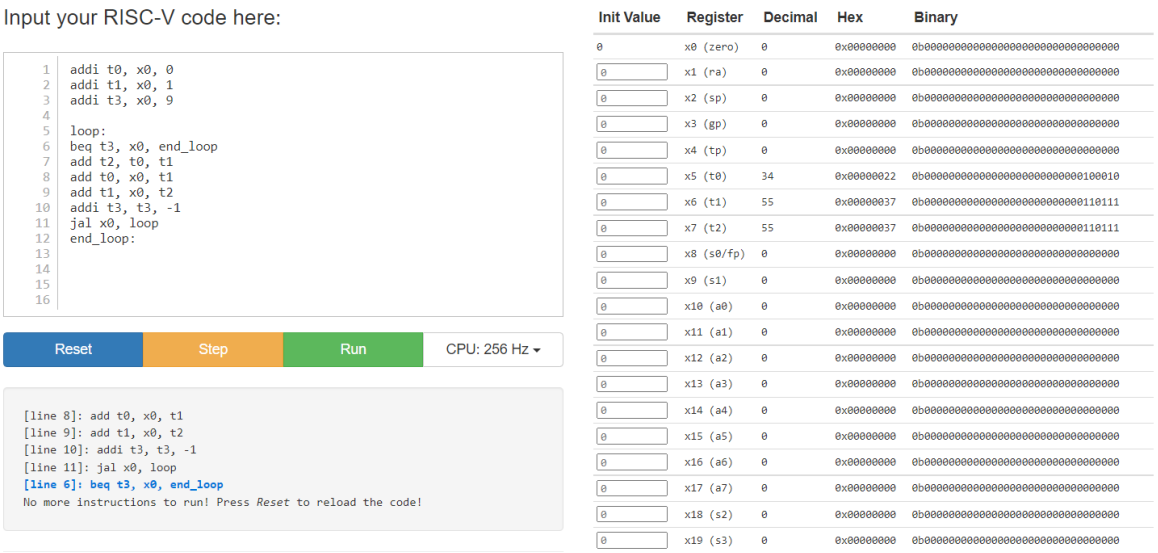
Dále se liší podpora a rozšíření jednotlivých architektur. RISC-V je relativně nová architektura, která má pouze omezenou podporu programovacího prostředí a softwaru. ARM má naopak velkou komunitu vývojářů a množství knihoven s podporou různých cílových platforem, od mikrokontrolerů po servery [16].

2 VÝVOJOVÉ NÁSTROJE RISC-V

V této kapitole budou představeny některé nástroje pro vývoj aplikací na architektuře RISC-V. Nejdříve budou představeny simulátory dostupné přes webovou stránku či ve formě běžné desktopové aplikace. Následně budou uvedeny prostředky pro kompilaci programu a vývoj kódu v integrovaných vývojových prostředích.

2.1 RISC-V Interpreter Cornellovy univerzity

Interpret dostupný na webových stránkách Cornellovy univerzity [18] je jednoduchý simulátor pro provádění kódů RISC-V. Podporuje pouze omezené množství základních instrukcí, jejich seznam je možné nalézt přímo na stránce interpretu. Grafické rozhraní obsahuje textovou oblast pro vložení samotného kódu a dále tlačítka pro restartování běhu, vykonání kódu po jednotlivých instrukcích nebo najednou, přičemž je možné určit takt procesoru (od 1 do 256 Hz) a při spuštění vykonávání je možné je přerušit. Pro možnost ladění při spuštění vykonávání instrukcí najednou je možné vložit do kódu zarážku na určitém řádku. Na pravé straně je možné sledovat stav registrů.



Obrázek 8: Interpret asemblerového jazyka RISC-V z Cornellovy univerzity

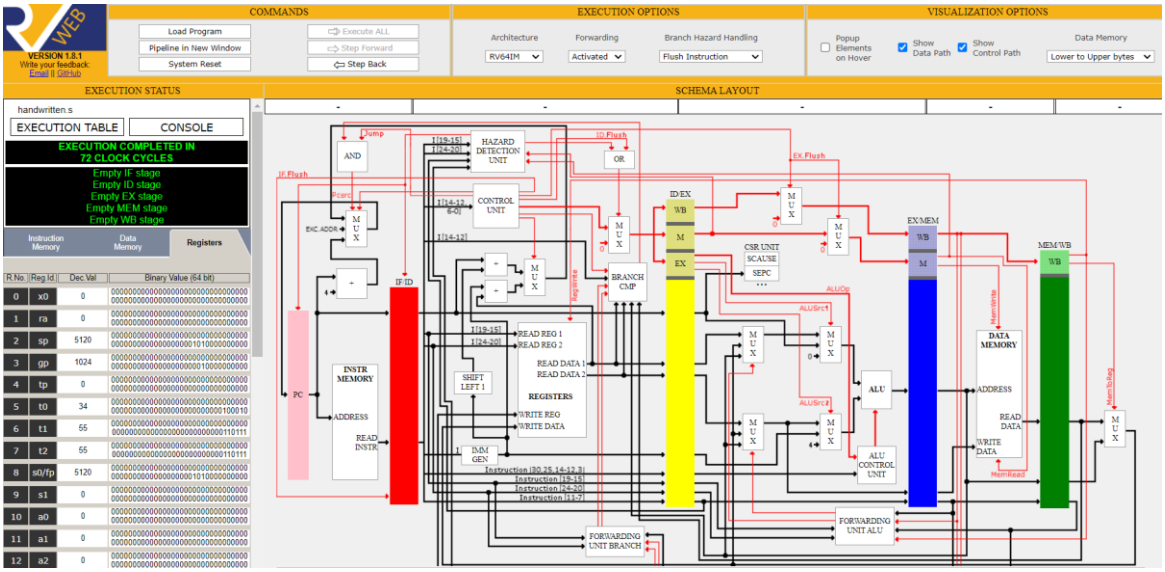
2.2 WebRISC-V

WebRISC [19] je webový simulátor datové cesty instrukcí pro architekturu RISC-V s podporou pipeline. Simulátor je vhodný pro výuku zpracování instrukcí a vykonávání kódu zapsaného v asemblerovém jazyku při zobrazení jednotlivých prvků architektury. Je k dispozici pod licencí BSD 3.

WebRISC-V obsahuje pětistupňovou pipeline s podporou 32bitové i 64bitové architektury v základní sadě I s rozšířením pro násobení M. Dále zobrazuje architektonické prvky a datové a řídící cesty mezi nimi. Simulátor umožňuje zobrazení tabulky pipeline, instrukční paměti, datové paměti, registrů nebo umožňuje interakci pomocí konzole díky implementaci systémových volání.

Dále je možné zvolit způsoby uložení čísel datové paměti a zvolit řešení při výskytu hazardu větvení (vynechání instrukce nebo zařazení zpoždění). Kromě možnosti zápisu vlastního kódu programátora je součástí simulátoru také seznam připravených příkladů, např. výpočet faktoriálu nebo systémového volání. Při vykonávání je možné postupovat buď po jednotlivých krocích vpřed i vzad nebo je možné vykonat celý kód najednou [20], [21]. Na program se ale vztahují následující omezení [19]:

- Nemůže obsahovat více než 256 instrukcí.
- Nemůže vykonat více než 2000 cyklů.
- Maximálně může použít 4 kB paměti.

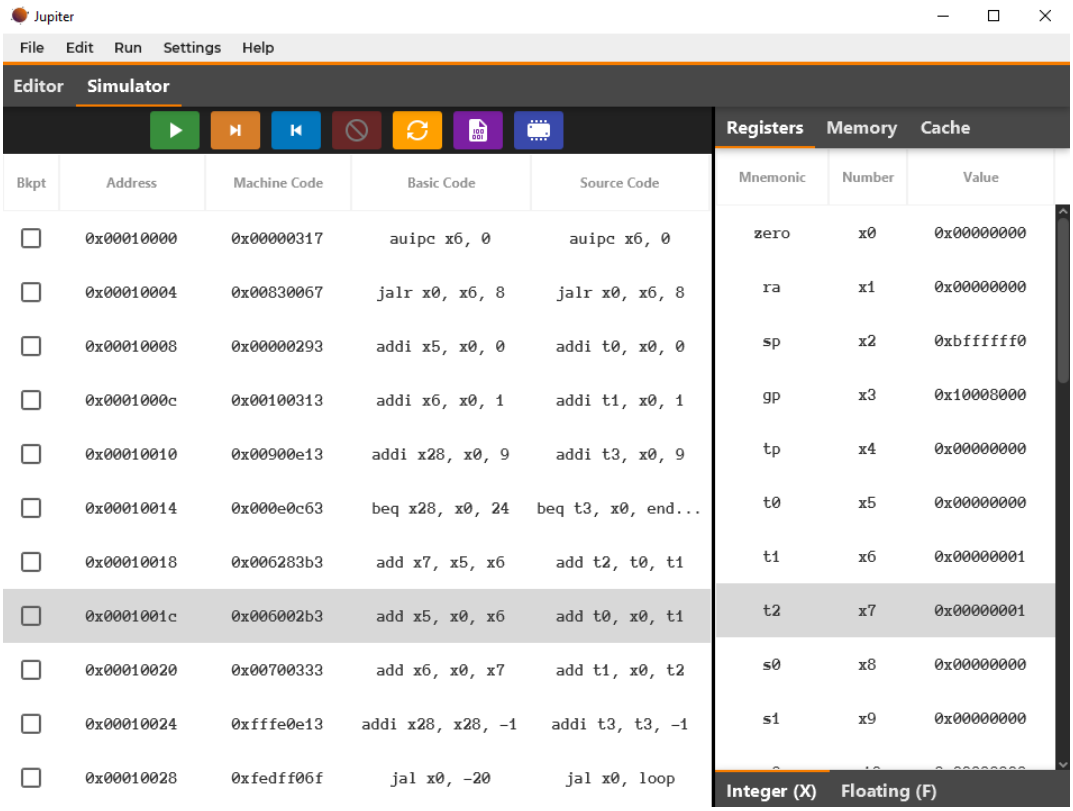


Obrázek 9: Simulátor WebRISC-V

2.3 Jupiter

Jupiter [22] je asembler a běhový simulátor pro platformu RISC-V. Jupiter podporuje architekturu RV32IMF a zároveň i všechny pseudoinstrukce popsané ve specifikaci RISC-V. Mezi hlavní výhody patří jednoduchost použití, neboť byl vyvinut se zaměřením na vzdělávací instituce a začínající programátory. Jupiter nabízí dva způsoby provozování: spuštění v příkazovém řádku nebo v grafickém rozhraní. Dále nabízí možnost sestavení a simulace kódu napsaného v několika souborech. Tento simulátor je dostupný na platformách Linux (Ubuntu), macOS i Windows. Je šířen pod licencí GNU GPL verze 3.

Jupiter se spouští z terminálu, a to buď příkazem *jupiter*, který spustí grafické rozhraní, nebo *jupiter [parametry spuštění] <soubory>* pro spuštění textového módu a vykonání programu. Grafické rozhraní má dvě záložky. Na záložce Editor se mohou spravovat a editovat jednotlivé soubory a pro jejich vykonání se vybere volba Assemble v menu Run. Následně se automaticky přepne na záložku Simulator, kde je k dispozici krokování kódu, jeho vykonání v dávce a možnosti restartu, uložení výpisu binárního kódu či obsahu paměti. Na pravé straně okna je zároveň možné sledovat stav registrů, paměti a nastavit parametry vyrovnávací paměti.

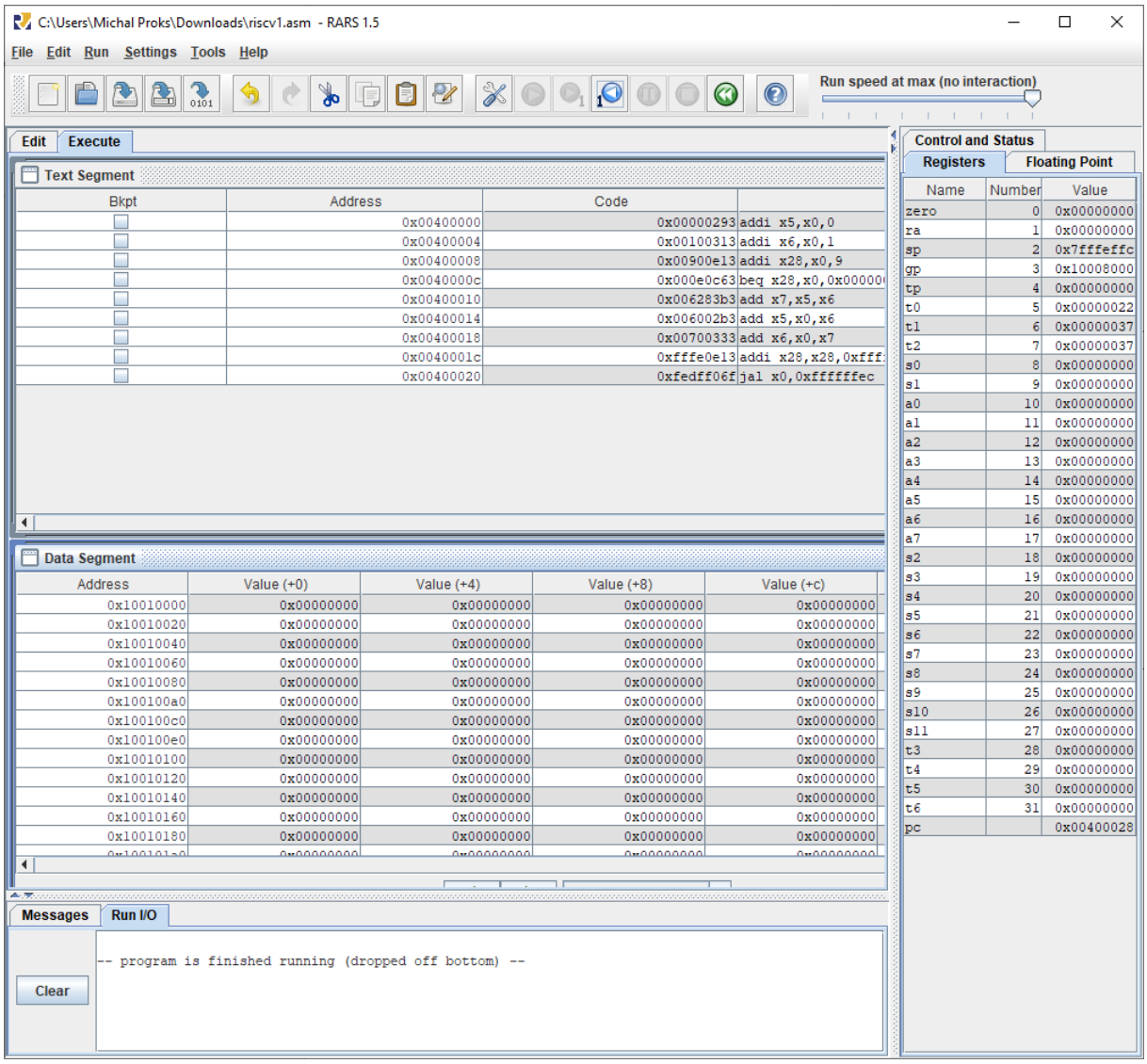


Obrázek 10: Grafické uživatelské rozhraní simulátoru Jupiter

2.4 Simulátor RARS

Simulátor RARS [23] slouží k sestavení a simulaci vykonávání kódu zapsaného v asemblerovém jazyku pro RISC-V. Podporuje 32bitové i 64bitové architektury, a kromě základní sady instrukcí I poskytuje také podporu rozšíření M, F, D a N, jakož i podporu několika typů systémových volání. Podporuje ladění s použitím zarážek a krokování programu vpřed i vzad. Dále nabízí možnost porovnání zapsaných pseudoinstrukcí se strojovým kódem a možnost sestavení z více souborů. V dokumentaci je možné nalézt seznam podporovaných instrukcí, systémových volání a direktiv asemblerového jazyka. Program se šíří pod licencí MIT [24] a je k dispozici ve formě spustitelného souboru JAR. Dále jsou dostupné i zdrojové kódy pro možnost modifikovat kód simulátoru.

Po spuštění je k dispozici hlavní okno se záložkami pro editaci souborů a vykonávání kódu. Nad ním je ovládací panel pro správu souborů, ovládání pro sestavení a běh programu. V pravé části jsou údaje o obsahu základních registrů, registrů pro desetinná čísla a stavové registry procesoru. V dolní části jsou zobrazeny zprávy ze simulátoru a vstupně/výstupní konzola běžícího programu.



Obrázek 11: Simulátor RARS

2.5 RISC-V GNU Compiler Toolchain

RISC-V GNU Compiler Toolchain je sada nástrojů pro vývoj programů v jazycích C a C++ na platformy s architekturou RISC-V a obsahuje programy jako gcc, g++, gdb a další. Podporuje různé linuxové distribuce, např. Ubuntu, Fedora, CentOS, RHEL OS a dále také operační systémy macOS a Microsoft Windows. Je šířen pod licencí GNU GPL verze 2.

Při sestavování toolchainu je možné vybrat ze dvou verzí: generický ELF/Newlib a pokročilejší Linux-ELF/glibc toolchain. Návod na instalaci na linuxových distribucích a macOS je k dispozici na webové stránce projektu na internetovém portálu GitHub. Nejprve je nutné nainstalovat programové prerekvizity, poté stáhnout zdrojové soubory z portálu GitHub a nakonec nakonfigurovat cestu k binárním souborům instalace a provést samotnou instalaci pomocí programu make. V případě varianty Linux-ELF/glibc je možné knihovnu glibc nahradit knihovnou musl libc. Při použití linuxové knihovny je dále defaultně nastavená verze architektury na RV64GC, a to i na 32bitových systémech. Pro vytvoření toolchainu na 32bitové architektury je potřeba při konfiguraci uvést přepínače pro vybrání architektury a typu ABI. Pro vytvoření kompilátoru, který umí kompilovat pro 32bitovou i 64bitovou verzi architektury, je možné použít při konfiguraci přepínač pro povolení více knihoven a při kompilaci používat přepínač *-march* nebo *-mabi* pro konkretizování architektury a rozhraní ABI. Seznam podporovaných architektur a rozhraní ABI je možné také nalézt na stránce projektu na portálu GitHub. Pro instalaci na systém Microsoft Windows je možné použít připravený instalátor verze ELF/Newlib z portálu gnutoolchains.com, který je dostupný ke stažení na záložce RISC-V v levém panelu.

Po nainstalování je možné toolchain začít používat pomocí příkazů, jejichž prefix se mění podle toho, jaká knihovna byla zvolena při instalaci. V případě knihovny Newlib je prefix *riscv64-unknown-elf-*, u knihovny glibc je *riscv64-unknown-linux-gnu-* a u knihovny musl libc je *riscv64-unknown-linux-musl-*. Za tento prefix se připojí jméno nástroje toolchainu a jeho parametry, např. *riscv64-unknown-elf-gcc -o soubor soubor.c* [25].

2.6 Eclipse Embedded CDT

Eclipse Embedded CDT je sada pluginů do integrovaného vývojového prostředí Eclipse, která umožnuje vývoj aplikací C/C++ určených pro platformy ARM a RISC-V bez potřeby vytvářet a spravovat makefily. Dále podporuje ladění aplikací přímo na hardwaru přes rozhraní JTAG/SWD s možností pozorování hodnot v registrech při ladícím procesu. Podporuje širokou škálu sad nástrojů pro 32bitové i 64bitové architektury, přidání mezikroků sestavení

a automatické vyhledání systémových cest a definic maker. Pro jednotlivé toolchainy umožňuje obecnou správu nastavení.

Tuto sadu je možné získat buď zvolením edice IDE Eclipse s názvem Eclipse IDE for Embedded C/C++ Developers nebo přidáním pluginu přímo v Eclipse Marketplace. Tato sada je šířena pod licencí Eclipse Public License 2.0 [26].

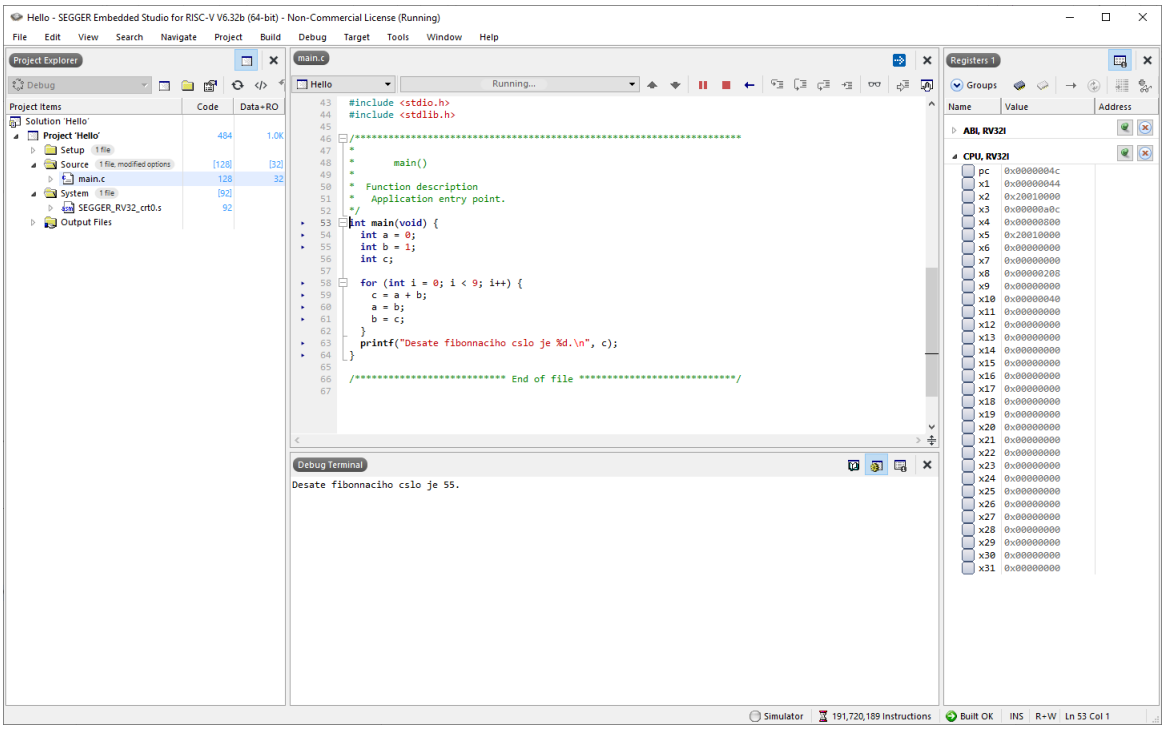
Pro možnost sestavení projektu je potřeba mít nainstalovaný kompilátor, jako např. dříve uvedený RISC-V GNU Compiler Toolchain. Při vytváření nového projektu C/C++ pro RISC-V se na poslední straně průvodce specifikuje typ kompilátoru a cesta k jeho binárním souborům.

2.7 SEGGER Embedded Studio for RISC-V

Embedded Studio for RISC-V od společnosti SEGGER je integrované vývojové prostředí určené pro vývoj aplikací pro platformy postavené na architektuře RISC-V v programovacím jazyku C/C++ nebo asemblerovém jazyku RISC-V. IDE obsahuje ucelený soubor řešení pro vývoj: toolchain, run-time knihovnu, základní verzi simulátoru a nástroje pro hardwarové ladění zařízení RISC-V připojených přes ladící sondy J-Link. Mezi podporované varianty architektury patří: RV32I, RV32IMA, RV32IMAC, RV32IMAF, RV32IMAFC, RV32G, RV32GC, RV32E, RV32EMA, RV32EMAC a 64bitové verze RV64I, RV64E a RV64GC [27].

Vývojové prostředí je k dispozici pod licencí SEGGER's Friendly License (SFL). Pro nekomerční použití nebo vyzkoušení je tento software poskytován bez poplatku. V jiných případech se použití považuje za komerční a je třeba zakoupit licenci v ceně 1,980 eur s technickou podporou a aktualizacemi po dobu jednoho roku [28].

Vzhled uživatelského rozhraní je možné vidět na obrázku č. 12. Hlavní okno je rozděleno mezi panely pro správu souboru projektu, editaci souboru a textový výstup pro činnosti provedené v IDE. V režimu ladění na integrovaném simulátoru jsou dále k dispozici panely s hodnotami registrů, popř. paměti, a dále je také vypisován výstup z programu do konzole.



Obrázek 12: SEGGER Embedded Studio for RISC-V

3 IMPLEMENTACE RISC-V

Architektura RISC-V je v současnosti implementována v řadě produktů jako jsou mikrokontrolery, vývojové kity či programovatelná hradlová pole. V této kapitole budou představeni vybraní zástupci těchto kategorií.

3.1 SiFive HiFive1 Rev B

Vývojový kit HiFive1 Rev B od firmy SiFive je vybaven mikrokontrolerem FE310-G002. Kit váží 22 g a má rozměry 68 mm x 51 mm.

Vývojový kit obsahuje:

- 19 digitálních vstupně-výstupních pinů.
- 9 pinů PWM.
- 2 sběrnice UART.
- Sběrnice I2C.
- Rozhraní Wi-Fi a Bluetooth (mimo čip).
- 19 externích pinů přerušení.
- Externí pin pro probuzení.
- Paměť typu flash o velikost 32 Mb připojenou přes rozhraní SPI.
- Hostitelské rozhraní microUSB.

Rozšiřující vstupně-výstupní konektory a jejich výstupy na pinech jsou odvozeny z platformy Arduino. Mikrokontrolér FE310-G002 je vybaven jádrem SiFive E32 s frekvencí až 320 Mhz a instrukční sadu RV32IMAC. Obsahuje 16 kB instrukční vyrovnávací paměti a 16 kB datové paměti SRAM.

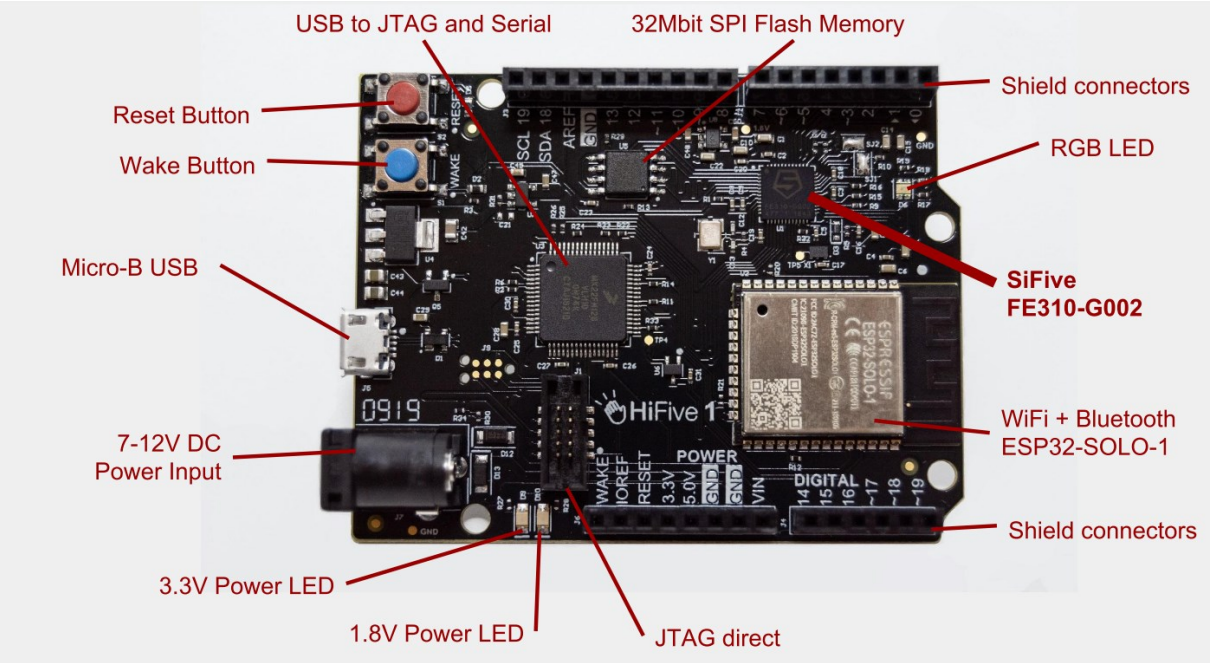
Kit HiFive1 Rev B má integrovaný modul Segger J-Link OB, který přemosťuje USB na JTAG, a dva sériové porty. JTAG se používá pro ladění mikrokontroleru FE310-G002, sériový port 0 se používá pro přístup ke konzoli FE310-G002 a sériový port 1 pro čip Espressif Systems ESP32-SOLO-1.

Tento čip zajišťuje funkce Bluetooth, Bluetooth LE, a Wi-Fi 802.11n. Obsahuje samostatný SoC, paměť flash, přesné diskrétní součástky a anténu PCB. Sériový port 1 se používá na konfiguraci paměti flash, kde je při dodání produktu nahrán firmware esp32-at. Dále je připojen přes rozhraní SPI k mikrokontroleru FE310-G002 a toto rozhraní se používá jako primární datová cesta mezi nimi.

Kit se může napájet třemi způsoby. Prvním je napájení skrz USB napojené na hostitelský systém. Tento způsob zajišťuje i schopnost komunikace s kitem. Druhou možností je napájení

přes napájecí konektor J7, ke kterému se připojí 7–12V zdroj stejnosměrného napětí či baterie. Dále může být kit připojen přes konektor VIN, ke kterému se také připojuje 7–12V zdroj stejnosměrného napětí.

Vývojový kit lze rozšířit o další rozhraní pomocí shieldu. Shield je rozšiřující modul, který je navržen tak, že využívá pro připojení konektor I/O a tvarově odpovídá platformě Arduino. Může být připojen na digitální konektory I/O, sběrnice SPI nebo UART, jejichž účelem je poskytnutí dalších funkcionalit. Při použití těchto rozšíření může být nutné upravení jejich softwarových knihoven. Vývojový kit dále obsahuje konektor 2x5, 0,05" pro použití externího JTAG programátoru. Tento programátor nahradí integrovaný modul a Segger J-Link OB [29], [30], [31].



Obrázek 13: Vývojový kit HiFive1 Rev B [31]

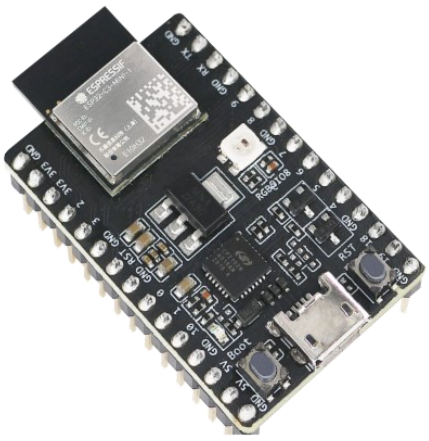
3.2 ESP32-C3

ESP32-C3 je Wi-Fi a Bluetoth 5 SoC postavený na architektuře RISC-V od firmy Espressif Systems. Obsahuje jedno-jádrový 32bitový procesor s maximální frekvencí 160 MHz a čtyřstupňovou pipeline. Dále je vybaven 400 KB interní paměti typu SRAM s 16 KB vyrovnávací pamětí, podporuje mód s nízkou spotřebou a je možné připojit externí paměť flash přes sběrnice SPI, Dual SPI, Quad SPI nebo QPI.

Bezdrátová konektivita je dostupná přes technologii Bluetooth verze 5 a SoC podporuje technologie Bluetooth SIG Mesh and Espressif Wi-Fi Mesh. Wi-Fi subsystém podporuje protokoly IEEE 802.11b/g/n a mezi možné operační módy patří módy SoftAP, Station, SoftAP + Station a promiskuitní mód. Vnější konektivita zahrnuje 22 obecných vstupně-výstupních pinů, dále tři sběrnice SPI, dvě směrnice UART a sběrnice I2C a I2S. Dále obsahuje kontrolér USB/JTAG a univerzální DMA kontrolér s třemi kanály pro příjem a třemi pro vysílání. Mezi analogová rozhraní patří převodník SAR ADC a teplotní senzor.

Podporuje také bezpečností prvky, jako je zabezpečené bootování založené na RSA-3072 a šifrování paměti flash založené na AES-128/256-XTS. Dále jsou přítomny akcelerátory pro SHA a HMAC, generátor náhodných čísel a podpora digitálních podpisů.

Hlavní využití SoC nachází v internetu věcí, průmyslové automatizaci, chytrém zemědělství, spotřebitelské elektronice a dalších oblastech [32].



Obrázek 14: Vývojový kit s čipem ESP32-C3 [33]

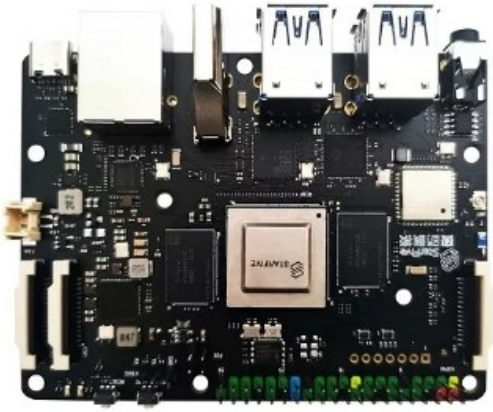
3.3 VisionFive V1

Jednodeskový počítač VisionFive V1 společnosti StarFive je nástupcem počítače BeagleV, který se nedostal do komerční výroby. Zařízení je vybaveno Soc StarFive JH7100, který kromě CPU obsahuje digitální signálový proces Tensilica-VP6 sloužící pro počítačové zpracování obrazu. Dalšími součástmi SoC jsou NVIDIA Deep Learning Accelerator (NVDLA) a engine pro neuronovou síť. CPU je vybaveno dvěma jádry SiFive U74 založenými na architektuře RV64GFC a taktovanými na 1,5 GHz.

Počítač má k dispozici 8 GB paměti LPDDR4 SDRAM ve dvou modulech po 4 GB. Paměť je taktovaná na 2800 MHz. Pro uložení dat je přítomný slot pro karty typu SD. Na zobrazování je možné použít HDMI verze 1.4 nebo port MIPI DSI a pro přenos zvuku buď čtyřpólový konektor jack nebo jde použít rozhraní HDMI. Datové přenosy je možné uskutečnit přes čtyři porty USB verze 3.0 a konektor USB-C. Dále je k dispozici header s 40 univerzálními vstupně-výstupními piny (GPIO). Pro připojení k sítím je možné použít Bluetooth verze 4.2 BLE a technologii Wi-Fi ve specifikaci IEEE 802.11b/g/n. K počítači je možné připojit i kameru přes rozhraní MIPI CSI.

Pro napájení je použito 5V stejnosměrné napětí připojené buď přes konektor USB-C nebo přes header GPIO. Jako operační systém je použit Linux. Počítač je dále kompatibilní se zavaděči GRUB2 a U-Boot.

Díky vysokému výkonu tento počítač najde využití při strojovém učení a aplikacích umělé inteligence. Další příklady použití jsou jako centrální systém chytrého domu, NAS nebo jako multimediální zařízení, díky podpoře dekodéru pro kódování H.264/H.265 a podpoře živého streamování až do rozlišení 4K [34], [35].



Obrázek 15: Jednodeskový počítač StarFive VisionFire V1 [35]

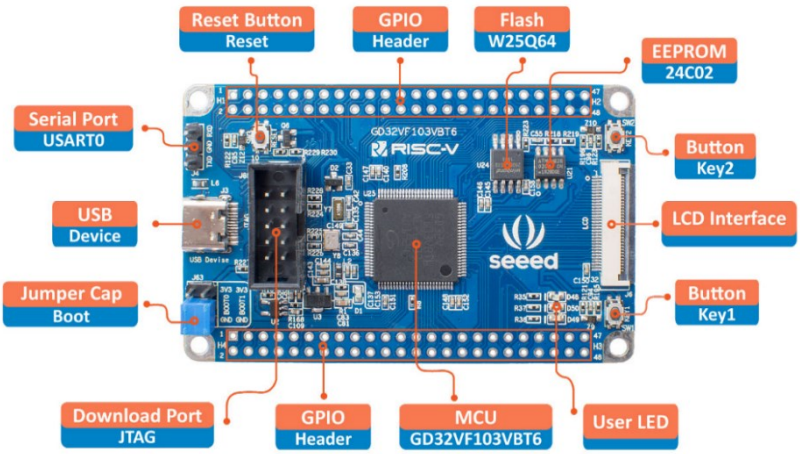
3.4 GD32VF103 RISC-V

GD32VF103 je série mikrokontrolerů se zabudovaným univerzálním 32bitový procesorem RISC-V. Procesor pracuje na frekvenci 108 MHz a má dostupnou paměť SRAM o kapacitě 6 až 32 kB a dále 16 až 128 kB paměti flash lišící se podle konkrétního modelu. Vstupně-výstupní zařízení a příslušenství jsou připojena přes dvě sběrnice APB. K dispozici jsou až dva konvertory ADC (12bitový) a až dva konvertory DAC (12bitový). Dále jsou zabudovány dva obyčejné časovače, čtyři univerzální 16bitové časovače a časovač s technologií PWM. Mikrokontrolery jsou dále vybaveny podporou pro následující sběrnice:

- Až 3x sběrnice SPI, USART.
- Až 2x sběrnice I2C, UART, I2S, CAN.
- USBFS.

Zařízení je napájeno stejnosměrným napětím o velikosti 2,6 V až 3,6 V a má tři módy pro úsporu energie, kterou reprezentují různé preference mezi spotřebou energie a malou latencí při probuzení. Zařízení může pracovat při teplotě od –40 °C do 85 °C. Typickými případy použití jsou monitorování spotřeby, systémy alarmu, displeje LED a další [36].

Příkladem vývojového kitu, který využívá mikrokontroler, této řady je SeeedStudio GD32 RISC-V. Kit obsahuje mikrokontroler GD32VF103VBT6 s 32 kB paměti SRAM a 128 kB paměti flash a podporuje všechny dříve uvedené sběrnice. Na kitu je dále umístěno 80 univerzálních vstupně-výstupních pinů, 8 MB paměti flash a 256 bajtů paměti EEPROM, konektor USB typu C, rozhraní pro připojení displeje LCD, slot pro karty TF, dvě uživatelská tlačítka a jedno tlačítko pro restart, tři uživatelské diody LED a konektor JTAG [37].



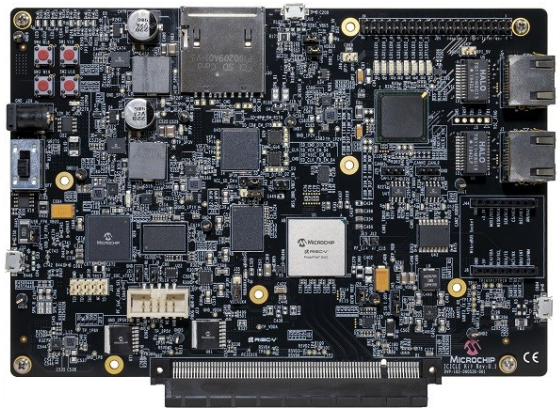
Obrázek 16: Vývojový kit SeeedStudio GD32 RISC-V [37]

3.5 SoC PolarFire

PolarFire je rodina SoC FPGA. Jedná se o SoC FPGA s clusterem jader RISC-V a paměťovým subsystémem L2. CPU jsou postaveny na architektuře RV64IMAC pro monitorovací jádra a RV64GC pro aplikační jádra. Nabízí 25 až 460 tisíc logických prvků, mohou obsahovat 68 až 1420 matematických bloků, 84 až 1460 bloků paměti LSRAM a 204 až 4260 bloků pamětí uSRAM. Dále mají paměť typu uPROM s kapacitou 192 až 553 kb. Vývojové kity mohou mít 4 až 20 serializačních/deserializačních linek, 2 linky pro koncová zařízení nebo kořenové porty rozhraní PCIe druhé generace a obsahují transceivery s kapacitou 12,7 gigabitů/s. Umožňují běh linuxových systémů a aplikací v reálném čase.

Pro zabezpečení je k dispozici hardwarový generátor náhodných čísel a dále podpora šifrování AES-128/192/256 a SHA-1/224/256/384/512. Další přítomné technologie jsou HMAC, RSA, ECDSA, ochrana datového toku, zabezpečené bootování a další. Poskytují také fyzické zabezpečení paměti pro každé procesorové jádro [38].

Příkladem vývojového kitu, který integruje SoC PolarFire, je Icicle firmy Microchip. Toto zařízení je vybaveno pěti-jádrovým procesorem MPFS250T-FCVG484EES, který se skládá z jednoho monitorovacího jádra a čtyř aplikačních jader. Dále má k dispozici 2 GB paměti LPDDR4, 1 Gb paměti SPI flash, 8 GB paměti eMMC flash a slot pro přidání paměťových karet typu SD. Pro programování a ladění je přítomen konektor JTAG a sběrnice UART připojená přes konektor microUSB. Mezi další rozhraní patří čtyři linky 12,7gigabitového rozhraní SERDES, kořenový port PCIe druhé generace, dva porty pro Gigabit Ethernet, OTG USB 2.0 a dále po dvou sběrnicích SPI, I2C a CAN. Vývojový kit je možné dále rozšířit přes 40pinový header kompatibilní s Raspberry Pi nebo soketem mikroBUS [39].



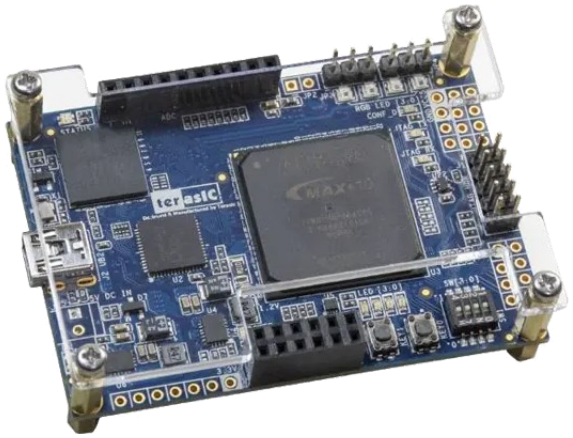
Obrázek 17: Vývojový kit Microchip Icicle [40]

3.6 Nios V

Nios V je rodina soft procesorů pro FPGA značky Intel. Tyto procesory jsou k dispozici v návrhovém softwaru Intel Quartus Prime Pro od verze 21.3. Procesory jsou založené na instrukční sadě RV32IA, disponují pětistupňovou pipeline a rozhraními AXI4. Dále podporují Intel Hardware Abstraction Layer (HAL) a real-teamový operační systém uC/OS-II pro přenos návrhů procesorů rodiny Nios II, oproti kterým Nios V nabízí více než pětinásobný výkon. Mezi další výhody patří bezplatná licence, široké možnosti výběru komunitních i komerčních ladící nástrojů a operačních systémů a podpora JTAG modulu pro ladění přímo na čipu.

K dispozici je také integrované vývojové prostředí RiscFree založené na Eclipse IDE, které umožňuje vytvářet aplikace pro procesory Intel založené jak na architektuře ARM, tak i na architektuře RISC-V. Je dostupné jako součást softwaru Intel Quartus Prime Pro nebo jako samostatná aplikace. Umožňuje provádět ladění v reálném čase a také vizualizovat registry Nios V procesorů [41].

Příkladem vývojového kitu, u kterého je možné použít soft procesory, je T-Core FPGA MAX 10 s integrovaným zařízením MAX 10 10M50DAF484C7G umožňujícím programování soft procesorů. Kit je vybaven 50 tisíci programovatelnými prvky, disponuje 1638 kb paměti M9K a 5888 kb paměti flash. Dále obsahuje dvě tlačítka, čtyři přepínače DIP, čtyři zelené a čtyři RBG diody LED. Součástí je technologie USB-Blaster II pro programování RISC-V a JTAG master pro programování FPGA na dalších vývojových kitech. Pro časování jsou k dispozici dva oscilátory s frekvencí 50 MHz a jeden s frekvencí 10 MHz. Kód RISC-V je možné uložit do 64 Mb paměti QSPI flash. Vývojový kit je možné rozšířit přes headery TMD, ADC a header pro RGB LED. Pro napájení je možné použít mini USB port typu A nebo B nebo externí dvoupinový napájecí header [42].



Obrázek 18: Vývojový kit T-Core FPGA MAX 10 [42]

ZÁVĚR

V současné době představuje architektura RISC-V relativně novou architekturu, která může představovat alternativu k architektuře ARM např. v mikropočítačích, z nichž některé byly představeny. Díky vysoké přizpůsobitelnosti pro konkrétní použití má potenciál se prosadit v širokém spektru použití. Další hlavní konkurenční výhodou je volná dostupnost. Naopak nevýhodou je malá rozšířenost a podpora ve srovnání s komerční konkurencí.

V této bakalářské práci byla popsána architektura RISC-V, její historie, základní instrukční sady ve 32bitové a 64bitové variantě, byla představena dostupná standardní rozšíření, práce s pamětí, úrovně běhu a bylo provedeno porovnání s architekturou ARM. Poté byly představeny dostupné simulátory, kompilátory a integrována vývojová prostředí pro vývoj softwaru pro tuto architekturu. Nakonec byly stručně popsány některé dostupné hardwarové prostředky implementující RISC-V.

POUŽITÁ LITERATURA

[1] WATERMAN, Andrew a Krste ASANOVIĆ. *The RISC-V Instruction Set Manual Volume I: Unprivileged ISA* [online]. Document Version 20191213. San Francisco: RISC-V Foundation, 13. 12. 2019 [cit. 2022-04-12]. Dostupné z: https://github.com/riscv/riscv-isa-manual/releases/download/Ratified-IMAFDQC/riscv-spec-20191213.pdf.

[2] ATWELL, Cabe. What is RISC-V?. *Fierce Electronics* [online]. New York: Questex LLC, 11. 3. 2022 [cit. 2022-8-4]. Dostupné z: https://www.fierceelectronics.com/embedded/what-risc-v.

[3] History of RISC-V. *RISC-V International* [online]. San Francisco: RISC-V International, 2021 [cit. 2022-4-12]. Dostupné z: https://riscv.org/about/history/.

[4] WOOFF, Mark. 1st RISC-V Workshop Proceedings. *RISC-V International* [online]. San Francisco: RISC-V International, 14. 1. 2015 [cit. 2022-4-12]. Dostupné z: https://live-risc-v.pantheonsite.io/proceedings/2015/01/1st-risc-v-workshop-bootcamp/.

[5] QUINNELL, Rich. Creating a Custom Processor with RISC-V. *EE Times Europe* [online]. Cambridge (US): AspenCore Media, 29. 3. 2019 [cit. 2022-8-4]. Dostupné z: https://www.eetimes.eu/creating-a-custom-processor-with-risc-v/.

[6] AUFRANC, Jean-Luc. RISC-V Bases and Extensions Explained. *CNX Software: Embedded Systems News* [online]. Hong Kong: CNX Software Limited, 27. 8. 2019 [cit. 2022-12-4]. Dostupné z: https://www.cnx-software.com/2019/08/27/risc-v-bases-and-extensions-explained/.

[7] HO, Steven. *Great Ideas in Computer Architecture: RISC-V Instruction Formats* [online]. Berkeley: University of California, 27. 6. 2018 [cit. 2022-04-12]. Dostupné z: https://inst.eecs.berkeley.edu/~cs61c/resources/su18_lec/Lecture7.pdf.

[8] ENGHEIM, Erik. RISC-V Instruction-Set Cheatsheet. *ITNEXT* [online]. Netherlands: LINKIT, 15. 5. 2022 [cit. 2022-8-4]. Dostupné z: https://itnext.io/risc-v-instruction-set-cheatsheet-70961b4bbe8.

[9] YAN, Yonghong. *Lecture 07: RISC-V Single-Cycle Implementation* [online]. Columbia: University of South Carolina, 2018 [cit. 2022-8-4]. Dostupné z: https://passlab.github.io/CSCE513/notes/lecture07_RISCV_Impl.pdf.

[10] WATERMAN, Andrew, Krste ASANOVIĆ a John HAUSER. *The RISC-V Instruction Set Manual Volume II: Privileged Architecture* [online]. Document Version 20211203. San Francisco: RISC-V International, 4. 12. 2021 [cit. 2022-04-12]. Dostupné z: https://github.com/riscv/riscv-isa-manual/releases/download/Priv-v1.12/riscv-privileged-20211203.pdf.

[11] PEDAMKAR, Priya. ARM Architecture: Introduction to ARM Architecture. *EDUCBA* [online]. Mumbai: EDUCBA, 2022 [cit. 2022-12-2]. Dostupné z: https://www.educba.com/arm-architecture/.

[12] ARM processor and its Features. *GeeksforGeeks* [online]. Noida, Uttar Pradesh: GeeksforGeeks, 6. 9. 2021 [cit. 2022-12-2]. Dostupné z: https://www.geeksforgeeks.org/arm-processor-and-its-features/.

[13] WALSHE, Ben. A Brief History of Arm: Part 1. *Arm Community* [online]. Cambridge (UK): Arm Limited, 21. 4. 2015 [cit. 2022-12-2]. Dostupné z: https://community.arm.com/arm-community-blogs/b/architectures-and-processors-blog/posts/a-brief-history-of-arm-part-1.

[14] MICHAEL, Stephen St. The Arm Architecture Explained. *ALL ABOUT CIRCUITS* [online]. Boise, Idaho: EETech Media, LLC, 10. 4. 2019 [cit. 2022-12-2]. Dostupné z: https://www.allaboutcircuits.com/technical-articles/arm-architecture-explained/.

[15] ARM1 - Microarchitectures - Acorn. *WikiChip* [online]. New York: WikiChip LLC, 14. 1. 2021 [cit. 2022-12-04]. Dostupné z: https://en.wikichip.org/wiki/acorn/microarchitectures/arm1.

[16] MITCHELL, Robin. ARM vs RISC-V: What Are the Major Differences?. *Electropages* [online]. Dorset: Electropages Media, 31. 3. 2021 [cit. 2022-04-12]. Dostupné z: https://www.electropages.com/blog/2021/03/arm-vs-risc-v.

[17] KAMIR, Erwin. ARM vs RISC-V. *Tech Journeyman* [online]. Tech Journeyman, 25. 2. 2022 [cit. 2022-12-2]. Dostupné z: https://techjourneyman.com/blog/arm-vs-risc-v/.

[18] RISC-V Interpreter. *Cornell University* [online]. Ithaca: Cornell University, 2019 [cit. 2022-8-4]. Dostupné z: https://www.cs.cornell.edu/courses/cs3410/2019sp/riscv/interpreter/.

[19] WebRISC-V - RISC-V PIPELINED DATAPATH SIMULATION ONLINE. *Università di Siena* [online]. Siena: Università di Siena [cit. 2022-11-12]. Dostupné z: https://webriscv.dii.unisi.it/.

[20] GIORGI, Roberto a Gianfranco MARIOTTI. *WebRISC-V: a Web-Based Education-Oriented RISC-V Pipeline Simulation Environment* [online]. Phoenix, AX, (USA): Association for Computing Machinery, June, 2019 [cit. 2022-8-4]. ISBN: 978-1-4503-6842-1/19/06. Dostupné z: http://www.dii.unisi.it/~giorgi/papers/Giorgi19-wcae.pdf.

[21] MARIOTTI, Gianfranco. WebRISC-V. *GitHub: Mariotti94* [online]. San Francisco: GitHub, 4. 4. 2022 [cit. 2022-11-12]. Dostupné z: https://github.com/Mariotti94/WebRISC-V.

[22] CASTELLANOS, Andrés. Jupiter: RISC-V Assembler & Runtime Simulator. *GitHub: andrescv* [online]. San Francisco: GitHub, 27. 9. 2019 [cit. 2022-8-4]. Dostupné z: https://github.com/andrescv/jupiter/blob/main/README.md.

[23] LANDERS, Benjamin. RARS -- RISC-V Assembler and Runtime Simulator. *GitHub: TheThirdOne* [online]. San Francisco: GitHub, 29. 9. 2020 [cit. 2022-8-4]. Dostupné z: https://github.com/TheThirdOne/rars/blob/master/README.md.

[24] LANDERS, Benjamin. Rars/License.txt. *GitHub: TheThirdOne* [online]. San Francisco: GitHub, 28. 8. 2017 [cit. 2022-8-4]. Dostupné z: https://github.com/TheThirdOne/rars/blob/master/License.txt.

[25] RISC-V GNU Compiler Toolchain. *GitHub: riscv-collab* [online]. San Francisco: GitHub, 23. 5. 2022 [cit. 2022-8-4]. Dostupné z: https://github.com/riscv-collab/riscv-gnu-toolchain.

[26] IONESCU, Liviu. Eclipse Embedded CDT: C/C++ Development Tools. *Eclipse Foundation* [online]. Ottawa: Eclipse Foundation, 12. 6. 2022 [cit. 2022-8-4]. Dostupné z: https://projects.eclipse.org/projects/iot.embed-cdt.

[27] Embedded Studio for RISC-V. *SEGGER* [online]. Monheim am Rhein: SEGGER Microcontroller GmbH, 2022 [cit. 2022-8-4]. Dostupné z: https://www.segger.com/products/development-tools/embedded-studio/editions/risc-v/.

[28] Embedded Studio Licensing Conditions. *SEGGER* [online]. Monheim am Rhein: SEGGER Microcontroller GmbH, 2022 [cit. 2022-8-4]. Dostupné z: https://www.segger.com/products/development-tools/embedded-studio/license/licensing-conditions/.

[29] SiFive. *SiFive FE310-G002 Datasheet* [online]. Version v1p2. San Mateo: SiFive, 2021 [cit. 2022-4-12]. Dostupné z: https://sifive.cdn.prismic.io/sifive/4999db8a-432f-45e4-bab2-57007eed0a43_fe310-g002-datasheet-v1p2.pdf.

[30] HiFive1 Rev B. *SiFive* [online]. San Mateo: SiFive, 2022 [cit. 2022-4-12]. Dostupné z: https://www.sifive.com/boards/hifive1-rev-b.

[31] SiveFive. *SiFive HiFive1 Rev B Getting Started Guide* [online]. Version 1.2. San Mateo: SiFive, 2021 [cit. 2022-4-12]. Dostupné z: https://sifive.cdn.prismic.io/sifive/cf239fd0-ae4f-4fd8-a944-fdafb5018153_hifive1b-getting-started-guide_v1.2.pdf.

[32] Espressif Systems. *ESP32 C3 Series: Datasheet* [online]. Version 1.2. Shanghai: Espressif Systems, 2022 [cit. 2022-4-12]. Dostupné z: https://www.espressif.com/sites/default/files/documentation/esp32-c3_datasheet_en.pdf.

[33] ESP32-C3-DevKitM-1. *Espressif Systems* [online]. Shanghai: Espressif Systems, 2022 [cit. 2022-12-3]. Dostupné z: https://docs.espressif.com/projects/esp-idf/en/latest/esp32c3/hw-reference/esp32c3/user-guide-devkitm-1.html.

[34] JADHAV, Abhishek. StarFive Officially Releases the VisionFive V1 SBC. *Embedded COMPUTING DESIGN* [online]. Scottsdale, Arizona: OpenSystems Media, 9. 12. 2021 [cit. 2022-8-4]. Dostupné z: https://embeddedcomputing.com/technology/processing/chips-and-socs/starfive-officially-releases-the-visionfive-v1-sbc.

[35] VÍTEK, Jan. VisionFive V1: RISC-V ve stylu Raspberry Pi přichází. *Svět hardware* [online]. Brno: oXyShop s.r.o., 30. 11. 2021 [cit. 2022-8-4]. Dostupné z: https://www.svethardware.cz/visionfive-v1-risc-v-ve-stylu-raspberry-pi-prichazi/56501.

[36] GD32 RISC-V Microcontrollers. *GigaDevice* [online]. Beijing: GigaDevice, 2022 [cit. 2022-8-4]. Dostupné z: https://www.gigadevice.com/products/microcontrollers/gd32/risc-v/.

[37] SeeedStudio GD32 RISC-V Dev Board. *Seeed studio* [online]. Shenzhen: Seeed Technology Co.,Ltd., 2022 [cit. 2022-11-12]. Dostupné z: https://wiki.seeedstudio.com/SeeedStudio-GD32-RISC-V-Dev-Board/.

[38] PolarFire® SoC FPGAs. *Microchip* [online]. Chandler, Arizona: Microchip Technology Inc., 2022 [cit. 2022-8-4]. Dostupné z: https://www.microchip.com/en-us/products/fpgas-and-plds/system-on-chip-fpgas/polarfire-soc-fpgas.

[39] PolarFire SoC FPGA Icicle Kit. *Microsemi* [online]. Aliso Viejo, California: Microsemi Corp., 2020 [cit. 2022-8-4]. Dostupné z: https://www.microsemi.com/existing-parts/parts/152514.

[40] Microchip PolarFire© SoC FPGA icicle kit. *EBV Elektronik* [online]. Phoenix, Arizona: Avnet, Inc., 2022 [cit. 2022-12-3]. Dostupné z: https://www.avnet.com/wps/portal/ebv/products/new-products/npi/2020/microchip-polarfire-soc/.

[41] Nios® V Processors. *Intel*: *Data Center Solutions, IoT, and PC Innovation* [online]. Santa Clara: Intel Corp., 2. 8. 2022 [cit. 2022-8-4]. Dostupné z: https://www.intel.com/content/www/us/en/products/details/fpga/nios-processor/v.html.

[42] Terasic Technologies T-Core FPGA MAX 10 Development Board. *Mouser* [online]. Brno: Mouser Electronics, 10. 3. 2022 [cit. 2022-8-4]. Dostupné z: https://cz.mouser.com/new/terasic-technologies/terasic-t-core-max-10-development-board/.

PŘÍLOHY

PŘÍLOHA A – FORMÁTY A SEZNAMY INSTRUKCÍ

Typy formátů instrukcí:

31 27	26 25	24 20	19 15	14 12	11 7	6 0	
funct7		rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]			rs1	funct3	rd	opcode	I-type
imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]					rd	opcode	U-type
imm[20|10:1|11|19:12]					rd	opcode	J-type

Formáty instrukcí základní instrukční sady RV32I:

imm[31:12]			rd	0110111	LUI
imm[31:12]			rd	0010111	AUIPC
imm[20|10:1|11|19:12]			rd	1101111	JAL
imm[11:0]	rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND
fm | pred | succ		rs1	000	rd	0001111	FENCE
000000000000		00000	000	00000	1110011	ECALL
000000000001		00000	000	00000	1110011	EBREAK

Seznam instrukcí základní ISA RV32I:

- **ADD:** součet hodnot v registrech rs1 a rs2, který se uloží do cílového registru.
- **SUB:** od hodnoty v rs1 odečte hodnotu rs2, výsledek uloží do cílového registru**.**
- **SLT:** do cílového registru nastaví 1, pokud je rs1 menší než rs2, jinak nastaví 0.
- **SLTU:** jako SLT, ale pro neznaménková čísla.
- **XOR:** do cílového registru uloží exkluzivní disjunkci bitů mezi registry rs1 a rs2.
- **OR:** do cílového registru uloží bitový součet mezi hodnotami v rs1 a rs2.
- **AND:** do cílového registru uloží bitový součin mezi hodnotami rs1 a rs2.
- **SLL:** logický posun hodnoty rs1 doleva o hodnotu uloženou v nejnižších 5 bitech registru rs2.
- **SRL:** logický posun hodnoty rs1 doprava o hodnotu uloženou v nejnižších 5 bitech registru rs2.
- **SRA:** aritmetický posun hodnoty rs1 doprava o hodnotu uloženou v nejnižších 5 bitech registru rs2.
- **ADDI:** součet hodnoty v registru rs1 a 12bitové konstanty imm, který se uloží do cílového registru.
- **SLTI:** do cílového registru nastaví 1, pokud je rs1 menší než imm, jinak nastaví 0.
- **SLTIU:** jako SLTI, ale pro neznaménková čísla.
- **XORI:** do cílového registru uloží exkluzivní disjunkci bitů mezi rs1 a konstantou imm.
- **ORI:** do cílového registru uloží bitový součet mezi rs1 a konstantou imm.
- **ANDI:** do cílového registru uloží bitový součin mezi rs1 a konstantou imm.
- **LW:** načte slovo na adrese určené součtem base a offsetu, hodnotu uloží do registru dst.
- **LB:** načte bajt na adrese určené součtem base a offsetu, znaménkově ho rozšíří na délku slova.
- **LH:** načte půlslovo na adrese určené součtem base a offsetu, hodnota je před uložením znaménkově rozšířena na celé slovo.
- **LBU:** jako LB, ale načte neznaménkový bajt, před uložením je hodnota doplněna nulami pro vyplnění 32bitového registru.
- **LHU:** jako LH, ale načte neznaménkové půlslovo, před uložením je hodnota doplněna nulami pro vyplnění 32bitového registru.

- **JALR:** příkaz skoku na adresu určenou součtem hodnoty v registru rs1 a offsetu (nejméně významný bit součtu nastaven na 0), do cílového registru rd se uloží hodnota registru PC + 4.
- **SLLI:** logický posun hodnoty v rs1 doleva o 5bitovou konstantu, uložení do cílového registru.
- **SRLI:** logický posun hodnoty v rs1 doprava o 5bitovou konstantu, uložení do cílového registru.
- **SRAI:** aritmetický posun hodnoty v rs1 doprava o 5bitovou konstantu, uložení do cílového registru.
- **ECALL:** instrukce se používá k žádosti o službu do běhového prostředí.
- **EBREAK:** instrukce slouží pro návrat řízení do ladícího prostředí.
- **FENCE:** slouží k uspořádání přístupů k paměti a vstupně-výstupním zařízením mezi jednotlivými hardwarovými vlákny či koprocesory.
- **SW:** hodnota z registru rs2 se uloží na adresu určenou součtem hodnoty v rs1 a imm.
- **SB:** nejnižší bajt z rs2 se uloží na adresu určenou součtem hodnoty v rs1 a imm.
- **SH:** půlslovo na nižší adrese z rs2 se uloží na adresu určenou součtem hodnoty v rs1 a imm.
- **BEQ:** pokud jsou hodnoty v rs1 a rs2 stejné, provede se skok.
- **BNE:** pokud jsou hodnoty v rs1 a rs2 rozdílné, provede se skok.
- **BLT:** pokud je hodnota v rs1 menší než hodnota v rs2, provede se skok.
- **BGE:** pokud je hodnota rs1 stejná nebo větší než hodnota v rs2, provede se skok.
- **BLTU:** jako BLT, ale pro neznaménková čísla.
- **BGEU:** jako BGE, ale po neznaménková čísla.
- **LUI:** instrukce pro nastavení horních 20 bitů 32bitového čísla v cílovém registru, spodních 12 bitů nastaveno na 0.
- **AUIPC:** instrukce slouží k sestavení adresy relativní k registru PC. Konstanta nastaví horních 20 bitů adresy, spodních 12 je vyplněno nulami. Tato hodnota je přidána k adrese instrukce AUIPC a výsledek je uložen do cílového registru.
- **JAL:** instrukce pro nepodmíněný skok.

Seznam instrukcí základní ISA RV64I (obsahuje také instrukce z RV32I, které ale nově pracují s 64bitovými čísly a produkují 64bitové výsledky):

- **LD:** načtení dvojslova (64 bitů) z paměti na adrese rs1 + offset do cílového registru.
- **SD:** uložení dvojslova z registru rs2 do paměti na pozici rs1 + offset.
- **LWU:** načtení neznaménkového slova (32 bitů) z paměti a jeho uložení do registru rd s rozšířením pomocí nul.
- **SLLI:** logický posun hodnoty doleva o 6bitovou konstantu.
- **SRLI:** logický posun hodnoty doprava o 6bitovou konstantu.
- **SRAI:** aritmetický posun hodnoty doprava o 6bitovou konstantu.

Ostatní instrukce (ADDIW, SLLIW, SRLIW, SRAIW, ADDW, SUBW, SLLW, SRLW, SRAW) mají stejnou funkčnost jako instrukce stejných názvů (bez přípony W) u instrukční sady RV32I, rozdílem je, že na 64bitové architektuře stále pracují s 32bitovými čísly a výsledek pak rozšíří na 64 bitů.

Seznam instrukcí rozšíření RV32M:

- **MUL:** provede násobení mezi dvěma celočíselnými registry a získá výsledek o délce 2 * velikost registru, do cílového registru uloží dolní polovinu bitů, operace pro znaménkové hodnoty.
- **MULH:** provede násobení mezi dvěma celočíselnými registry a získá výsledek o délce 2 * velikost registru, do cílového registru uloží horní polovinu bitů, operace pro znaménkové hodnoty.
- **MULHU:** funkce stejná jako u MULH, ale pro neznaménkové hodnoty v registrech.
- **MULHSU:** funkce stejná jako u MULH, ale pro znaménkovou hodnotu v prvním registru a neznaménkovou v druhém registru.
- **DIV:** provede dělení mezi dvěma registry a výsledek zaokrouhlí k nule, pro znaménková čísla.
- **DIVU:** funkce stejná jako u DIV, ale pro neznaménková čísla.
- **REM:** zbytek po dělení dvou znaménkových čísel, má stejné znaménko jako výsledek dělení.
- **REMU:** zbytek po dělení dvou neznaménkových čísel.

Seznam instrukcí rozšíření RV64M (obsahuje také instrukce z RV32M, které ale nově pracují s 64bitovými čísli):

- **MULW:** vynásobí dolních 32 bitů zdrojových celočíselných registrů, z výsledku je odděleno dolních 32 bitů a hodnota je znaménkově rozšířena do cílového registru.
- **DIVW:** podíl dolních 32 bitů dvou registrů, podíl znaménkově rozšířen na 64 bitů cílového registru.
- **DIVUW:** jako DIVW, ale pro neznaménková čísla.
- **REMW:** zbytek po dělení dolních 32 bitů dvou registrů, podíl znaménkově rozšířen na 64 bitů cílového registru.
- **REMUW:** jako REMW, ale pro neznaménková čísla.

Seznam instrukcí rozšíření RV32A:

- **LR.W:** načte slovo z paměti, znaménkově ho rozšíří, uloží do cílového registru a zaregistruje rezervační set (set bajtů zahrnující bajty z adresovaného slova).
- **SC.W:** podmíněně zapíše slovo ve druhém zdrojovém registru na adresu v prvním zdrojovém registru, uspěje pouze, pokud je rezervační set nastaven a obsahuje bajty, které se budou zapisovat.
- **AMOSWAP.W:** atomická výměna hodnot.
- **AMOADD.W:** atomický součet celočíselných hodnot.
- **AMOXOR.W:** atomická bitová exkluzivní disjunkce.
- **AMOAND.W:** atomický bitový součin.
- **AMOOR.W:** atomický bitový součet.
- **AMOMIN.W:** atomická znaménková funkce minimum.
- **AMOMAX.W:** atomická znaménková funkce maximum.
- **AMOMINU.W:** atomická neznaménková funkce minimum.
- **AMOMIXU.W:** atomická neznaménková funkce maximum.

Seznam instrukcí rozšíření RV64A:

Obsahuje kromě instrukcí z RV32A také nové instrukce, které pracují na úrovni dvouslov, funkčně jsou stejné jako instrukce pro 32bitovou platformu a mají novou příponu D namísto W.

Seznam instrukcí rozšíření Zicsr:

- **CSRRW:** atomická výměna hodnot v CSR a celočíselných registrech.
- **CSRRS:** přečte hodnotu v CSR, rozšíří ji nulami na délku celočíselných registrů a zapíše do cílového registru. Bity, které mají hodnotu jedna ve zdrojovém registru, budou nastaveny na jedna i v daném CSR, pokud je lze natavit.
- **CSRRC:** přečte hodnotu v CSR, rozšíří ji nulami na délku celočíselných registrů a zapíše do cílového registru. Bity, které mají hodnotu jedna ve zdrojovém registru, budou nastaveny na nulu v daném CSR, pokud je lze natavit.
- **CSRRWI:** jako CSRRW, ale pro nastavení CSR používá konstantu místo hodnoty v registru.
- **CSRRSI:** jako CSRRS, ale pro nastavení CSR používá konstantu místo hodnoty v registru.
- **CSRRCI:** jako CSRRC, ale pro nastavení CSR používá konstantu místo hodnoty v registru.

Seznam instrukcí rozšíření RV32F:

- **FLW:** načtení 32bitového desetinného čísla z paměti do registru.
- **FSW:** uložení 32bitového desetinného čísla z registru do paměti.
- **FMADD.S:** tři zdrojové registry, do cílového registru je uložen výsledek výrazu (rs1 × rs2) + rs3.
- **FMSUB.S:** tři zdrojové registry, do cílového registru je uložen výsledek výrazu (rs1 × rs2) - rs3.
- **FNMADD.S:** tři zdrojové registry, do cílového registru je uložen výsledek výrazu - (rs1 × rs2) - rs3.
- **FNMSUB.S:** tři zdrojové registry, do cílového registru je uložen výsledek výrazu - (rs1 × rs2) + rs3.
- **FADD.S:** součet desetinných 32bitových čísel v registrech.
- **FSUB.S:** rozdíl desetinných 32bitových čísel v registrech.
- **FMUL.S:** součin desetinných 32bitových čísel v registrech.
- **FDIV.S:** podíl desetinných 32bitových čísel v registrech.
- **FSQRT.S:** druhá odmocnina z 32bitového desetinného čísla.
- **FSGNJ.S:** vezme bity z registru rs1, kromě bitu znaménka, a použije znaménko z rs2, výsledek uložen do cílového registru.

- **FSGNJN.S:** vezme bity z registru rs1, kromě bitu znaménka, a použije negaci znaménka z rs2, výsledek uložen do cílového registru.
- **FSGNJX.S:** vezme bity z registru rs1, kromě bitu znaménka, a pro znaménko použije exkluzivní disjunkci bitů znamének z rs1 a rs2, výsledek uložen do cílového registru.
- **FMIN.S:** menší hodnota ze zdrojových registrů je zapsána do cílového registru.
- **FMAX.S:** větší hodnota ze zdrojových registrů je zapsána do cílového registru.
- **FMV.X.W:** instrukce pro přesun bitového vzoru z desetinného registru do celočíselného. Přesun 32 bitů.
- **FMV.W.X:** instrukce pro přesun bitového vzoru z celočíselného registru do desetinného. Přesun 32 bitů.
- **FEQ.S:** pokud je hodnota v rs1 rovna hodnotě v rs2, pak uloží do cílového celočíselného registru 1, jinak uloží 0.
- **FLT.S:** pokud je hodnota v rs1 menší než hodnota v rs2, pak uloží do cílového celočíselného registru 1, jinak uloží 0.
- **FLE.S:** pokud je hodnota v rs1 menší nebo rovna hodnotě v rs2, pak uloží do cílového celočíselného registru 1, jinak uloží 0.
- **FCLASS.S:** prozkoumá desetinné číslo a určí třídu, do které číslo patří, tu zapíše jako bitovou masku do cílového registru.
- **FCVT.W.S:** konvertování 32bitového čísla z desetinného registru na celočíselné znaménkové 32bitové číslo.
- **FCVT.WU.S:** konvertování 32bitového čísla z desetinného registru na celočíselné neznaménkové 32bitové číslo.
- **FCVT.S.W:** konvertování 32bitového znaménkového celého čísla na desetinné 32bitové číslo.
- **FCVT.S.WU:** konvertování 32bitového neznaménkového celého čísla na desetinné 32bitové číslo.

Seznam instrukcí rozšíření RV64F (obsahuje také instrukce ze RV32F):

- **FCVT.L.S:** konvertování 32bitového čísla z desetinného registru na celočíselné znaménkové 64bitové číslo.
- **FCVT.LU.S:** konvertování 32bitového čísla z desetinného registru na celočíselné neznaménkové 64bitové číslo.
- **FCVT.S.L:** konvertování 64bitového znaménkového celého čísla na desetinné 32bitové číslo.
- **FCVT.S.LU:** konvertování 64bitového neznaménkového celého čísla na desetinné 32bitové číslo.

Seznam instrukcí rozšíření RV32D:

- **FLD:** načtení 64bitového desetinného čísla z paměti do registru.
- **FSD:** uložení 64bitového desetinného čísla z registru do paměti.
- **FCVT.S.D:** konverze z 64bitového desetinného čísla na 32bitové desetinné číslo.
- **FCVT.D.S:** konverze z 32bitového desetinného čísla na 64bitové desetinné číslo.
- **FCVT.W.D:** konvertování 64bitového čísla z desetinného registru na celočíselné znaménkové 32bitové číslo.
- **FCVT.WU.D:** konvertování 64bitového čísla z desetinného registru na celočíselné neznaménkové 32bitové číslo.
- **FCVT.D.W:** konvertování 32bitového znaménkového celého čísla na desetinné 64bitové číslo.
- **FCVT.D.WU:** konvertování 32bitového neznaménkového celého čísla na desetinné 64bitové číslo.

Ostatní instrukce (FMADD.D, FMSUB.D, FNMADD.D, FNMSUB.D, FADD.D, FSUB.D, FMUL.D, FDIV.D, FSQRT.D, FSGNJ.D, FSGNJN.D, FSGNJX.D, FMIN.D, FMAX.D, FEQ.D, FLT.D, FLE.D, FCLASS.D) mají v RV32D stejnou funkci jako u rozšíření F, nyní ale používají 64bitové operandy, produkují 64bitové výsledky a přípona S se změnila na D.

Seznam instrukcí rozšíření RV64D (obsahuje také instrukce ze RV32D):

- **FCVT.L.D:** konverze 64bitového desetinného čísla na 64bitové znaménkové celé číslo.
- **FCVT.LU.D:** konverze 64bitového desetinného čísla na 64bitové neznaménkové celé číslo.
- **FCVT.D.L:** konverze 64bitového znaménkového celého čísla na 64bitové desetinné číslo.
- **FCVT.D.LU:** konverze 64bitového neznaménkového celého čísla na 64bitové desetinné číslo.
- **FMV.X.D:** instrukce pro přesun bitového vzoru z desetinného registru do celočíselného. Přesun 64 bitů.
- **FMV.D.X:** instrukce pro přesun bitového vzoru z celočíselného registru do desetinného. Přesun 64 bitů.

Seznam instrukcí rozšíření RV32Q:

- **FLQ:** načtení 128bitového desetinného čísla z paměti do registru.
- **FSQ:** uložení 128bitového desetinného čísla z registru do paměti.
- **FCVT.S.Q:** konverze z 128bitového desetinného čísla na 32bitové desetinné číslo.
- **FCVT.Q.S:** konverze z 32bitového desetinného čísla na 128bitové desetinné číslo.
- **FCVT.D.Q:** konverze z 128bitového desetinného čísla na 64bitové desetinné číslo.
- **FCVT.Q.D:** konverze z 64bitového desetinného čísla na 128bitové desetinné číslo.
- **FCVT.W.Q:** konvertování 128bitového čísla z desetinného registru na celočíselné znaménkové 32bitové číslo.
- **FCVT.WU.Q:** konvertování 128bitového čísla z desetinného registru na celočíselné neznaménkové 32bitové číslo.
- **FCVT.Q.W:** konvertování 32bitového znaménkového celého čísla na desetinné 128bitové číslo.
- **FCVT.Q.WU:** konvertování 32bitového neznaménkového celého čísla na desetinné 128bitové číslo.

Ostatní instrukce (FMADD.Q, FMSUB.Q, FNMADD.Q, FNMSUB.Q, FADD.Q, FSUB.Q, FMUL.Q, FDIV.Q, FSQRT.Q, FSGNJ.Q, FSGNJN.Q, FSGNJX.Q, FMIN.Q, FMAX.Q, FEQ.Q, FLT.Q, FLE.Q, FCLASS.Q) mají v RV32Q stejnou funkci jako u rozšíření F, nyní ale používají 128bitové operandy, produkují 128bitové výsledky a přípona S se změnila na Q.

Seznam instrukcí rozšíření RV64Q (obsahuje také instrukce ze RV32Q):

- **FCVT.L.Q:** konverze 128bitového desetinného čísla na 64bitové znaménkové celé číslo.
- **FCVT.LU.Q:** konverze 128bitového desetinného čísla na 64bitové neznaménkové celé číslo.
- **FCVT.Q.L:** konverze 64bitového znaménkového celého čísla na 128bitové desetinné číslo.
- **FCVT.Q.LU:** konverze 64bitového neznaménkového celého čísla na 128bitové desetinné číslo.

Seznam instrukcí rozšíření C:

- **C.LWSP:** nahraje 32bitovou hodnotu z paměti do cílového registru. Využívá ukazatel na zásobník ABI.
- **C.LDSP:** nahraje 64bitovou hodnotu z paměti do cílového registru. Pouze na RV64C a RV128C. Využívá ukazatel na zásobník ABI.
- **C.LQSP:** nahraje 128bitovou hodnotu z paměti do cílového registru. Pouze na RV128C. Využívá ukazatel na zásobník ABI.
- **C.FLWSP:** nahraje 32bitovou desetinnou hodnotu z paměti do desetinného cílového registru. Pouze na RV32FC. Využívá ukazatel na zásobník ABI.
- **C.FLDSP:** nahraje 64bitové desetinné číslo z paměti do desetinného cílového registru. Pouze na RV32DC a RV64DC. Využívá ukazatel na zásobník ABI.
- **C.SWSP:** nahraje 32bitovou hodnotu z registru do paměti. Využívá ukazatel na zásobník ABI.
- **C.SDSP:** nahraje 64bitovou hodnotu z registru do paměti. Pouze na RV64C a RV128C. Využívá ukazatel na zásobník ABI.
- **C.SQSP:** nahraje 128bitovou hodnotu z registru do paměti. Pouze na RV128C. Využívá ukazatel na zásobník ABI.
- **C.FSWSP:** nahraje 32bitovou desetinnou hodnotu z registru do paměti. Pouze na RV32FC. Využívá ukazatel na zásobník ABI.
- **C.FSDSP:** nahraje 64bitové desetinné číslo z registru do paměti. Pouze na RV32DC a RV64DC. Využívá ukazatel na zásobník ABI.
- **C.LW:** nahraje 32bitovou hodnotu z paměti do cílového registru.

- **C.LD:** nahraje 64bitovou hodnotu z paměti do cílového registru. Pouze na RV64C a RV128C.
- **C.LQ:** nahraje 128bitovou hodnotu z paměti do cílového registru. Pouze na RV128C.
- **C.FLW:** nahraje 32bitovou desetinnou hodnotu z paměti do desetinného cílového registru. Pouze na RV32FC.
- **C.FLD:** nahraje 64bitové desetinné číslo z paměti do desetinného cílového registru. Pouze na RV32DC a RV64DC.
- **C.SW:** nahraje 32bitovou hodnotu z registru do paměti.
- **C.SD:** nahraje 64bitovou hodnotu z registru do paměti. Pouze na RV64C a RV128C.
- **C.SQ:** nahraje 128bitovou hodnotu z registru do paměti. Pouze na RV128C.
- **C.FSW:** nahraje 32bitovou desetinnou hodnotu z registru do paměti. Pouze na RV32FC.
- **C.FSD:** nahraje 64bitové desetinné číslo z registru do paměti. Pouze na RV32DC a RV64DC.
- **C.J:** provede nepodmíněný skok.
- **C.JAL:** provede nepodmíněný skok a zapíše adresu instrukce následující po skoku do registru x1. Pouze na RV32C.
- **C.JR:** provede nepodmíněný skok na adresu ve zdrojovém registru.
- **C.JALR:** provede skok stejně jako C.JR a zapíše adresu instrukce následující po skoku do registru x1.
- **C.BEQZ:** provede podmíněný skok, pokud je zdrojový registr nulový.
- **C.BNEZ:** provede podmíněný skok, pokud je zdrojový registr nenulový.
- **C.LI:** nahraje znaménkově rozšířenou 6bitovou konstantu do cílového registru.
- **C.LUI:** nahraje nenulovou 6bitovou konstantu na 17. až 12. nejméně významný bit, spodních 12 bitů vynuluje a znaménkově rozšíří 17. bit do všech vyšších bitů.
- **C.ADDI:** sečte nenulovou znaménkově rozšířenou 6bitovou konstantu s hodnotou v cílovém registru a výsledek do cílového registru zapíše.
- **C.ADDIW:** jako C.ADDI, ale výsledek je 32bitové číslo, které je znaménkově rozšířeno na 64 bitů.
- **C.ADDI16SP:** sečte nenulovou znaménkově rozšířenou konstantu, násobenou šestnácti, s registrem x2 a výsledek zapíše do cílového registru.
- **C.ADDI4SPN:** sečte nenulovou znaménkově rozšířenou konstantu, násobenou čtyřmi, s registrem x2 a výsledek zapíše do cílového registru.

- **C.SLLI:** provede logický posun vlevo na hodnotě v cílovém registru a výsledek zapíše do cílového registru.
- **C.SRLI:** provede logický posun vpravo na hodnotě v cílovém registru a výsledek zapíše do cílového registru.
- **C.SRAI:** provede aritmetický posun vpravo na hodnotě v cílovém registru a výsledek zapíše do cílového registru.
- **C.ANDI:** provede bitový součin mezi hodnotou v cílovém registru a 6bitovou znaménkově rozšířenou konstantou, výsledek zapíše do cílového registru.
- **C.MV:** zkopíruje hodnotu ve zdrojovém registru do cílového.
- **C.ADD:** sečte hodnoty v zdrojovém a cílovém registru a výsledek zapíše do cílového registru.
- **C.AND:** provede bitový součin mezi hodnotami v zdrojovém a cílovém registru a výsledek zapíše do cílového registru.
- **C.OR:** provede bitový součet mezi hodnotami v zdrojovém a cílovém registru a výsledek zapíše do cílového registru.
- **C.XOR:** provede bitovou exkluzivní disjunkci mezi hodnotami v zdrojovém a cílovém registru a výsledek zapíše do cílového registru.
- **C.SUB:** odečte hodnotu ve zdrojovém registru od hodnoty v cílovém registru a výsledek zapíše do cílového registru.
- **C.ADDW:** sečte hodnoty v zdrojovém a cílovém registru, poté znaménkově rozšíří spodních 32 bitů výsledku a zapíše výsledek do cílového registru. Pouze na RV64C a RV128C.
- **C.SUBW:** odečte hodnotu zdrojového registru od cílového registru, poté znaménkově rozšíří spodních 32 bitů výsledku a zapíše výsledek do cílového registru. Pouze na RV64C a RV128C.
- **C.NOP:** nemění stav, pouze posune registr PC vpřed.
- **C.EBREAK:** způsobí skok do ladícího prostředí.

Rozšíření Zifencei:

- **FENCE.I:** poskytuje explicitní synchronizaci mezi zápisy do instrukční paměti a čtením instrukcí na stejném vlákně.