УДК 004.275.1

РАЗРАБОТКА СОБСТВЕННОЙ RISC-АРХИТЕКТУРЫ И АССЕМБЛЕРА К НЕЙ

Листаров С.С. – студент-магистрант,
Научный руководитель: Мыцко Е.А., к.т.н., доцент
Национальный Исследовательский Томский Политехнический Университет
Россия, г. Томск

В связи с медленными темпами роста производительности современных процессоров, проблема оптимизации программ и приложений является особенно актуальной [1]. Однако за последние десятилетия между разработчиком и процессором возникло большое количество промежуточных слоев, которые, с одной стороны упрощают разработку, с другой стороны, уменьшают эффективность разработанных решений.

Так, производительность инструмента для работы с цифровыми видео FFmpeg увеличилась в 97 раз, после того как разработчики переписали часть ее библиотек с C на ассемблер AVX-512 [2].

Этот пример наглядно демонстрирует превосходство низкоуровневого программирования с точки зрения эффективности.

Однако многие современные разработчики привыкли работать на сравнительно высоких уровнях абстракции, и имеют ограниченное представление о низкоуровневом устройстве компьютера. В виду чего разработанные ими решения, могут быть не самыми эффективными [3].

Таким образом, текущее состояние индустрии располагает к тому, чтобы углубиться в низкоуровневое устройство процессоров и привлечь внимание студентов и специалистов к данной малопопулярной области.

Целью этой работы является детальное описание разработки собственной архитектуры с сокращенным набором команд.

В виду сложности архитектур семейства CISC, было принято решение разработать RISC-архитектуру.

Были изучены несколько представителей этого семейства – открытая архитектура RISC-V, архитектура Супро Гаонкара [5] из его статьи, посвященной разработке 16-битного компьютера и архитектура от автора matbatwings [5] из его видеокурса.

Архитектура RISC-V имеет слишком большое количество инструкций, что будет трудно освоить начинающему низкоуровневому программисту.

Примеры любительских архитектур Супро Гаонкара [4] и matbatwings [5], демонстрируют избыточность такого количества команд, поэтому было принято решение выделить 4 бита на код операции и 4 бита на регистр, чтобы на код операции, регистр назначения, и два исходных регистр уходило ровно два байта. Такое решение позволило активно использовать псевдоинструкции и довести общее число поддерживаемых инструкций до 30.

Для простоты последующих ссылок, архитектуре было присвоено название NERO - **Next-generation Efficient RISC Organization.**

В качестве размера инструкции была выбрана длина в 24 бита, что подходит для записи 16-битного значения в указанный регистр.

Структура инструкций показано на рисунке 1.

	Byte 1								Byte 2								Byte 3							
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
Тип инструкции	Код операции				Операнды																			
Регистр/регистр	OPCODE				REG1				REG2				REG3				X	X	X	X	X	X	X	X
С операндом	OPCODE				REG1				IMMEDIATE16															
С адресом	OPCODE				REG1				ADDRESS16															
Со стэком	OPCODE				REG1				X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Процедура	OPCODE				X	X	X	X	ADDRESS16															
Ветвление	OPCODE				FLAGS				ADDRESS16															

Рисунок 1 – Структура команд архитектуры NERO

Следующим этапом разработки архитектуры был выбор инструкций. Для этой цели были проанализированы наборы команд из перечисленных выше архитектур и найдено их пересечение. Итоговый набор инструкций представлен на рисунке 2. Цвет означает разный тип инструкции.

Мнемоника	Описание	Нотация	Псевдокод
ADD	Сложение	ADD r1 r2 r3	REG1 = REG2 + REG3
SUB	Вычитание	SUB r1 r2 r3	REG1 = REG2 - REG3
MUL	Умножение	MUL r1 r2 r3	REG1 = REG2 * REG3
DIV	Деление	DIV r1 r2 r3	REG1 = REG2 / REG3
OR	Логическое сложение	OR r1 r2 r3	REG1 = REG2 | REG3
AND	Логическое умножение	AND r1 r2 r3	REG1 = REG2 & REG3
XOR	Исключающее ИЛИ	XOR r1 r2 r3	REG1 = REG2 ^ REG3
ADI	Прибавление непосредственного значения	ADI r1 imm16	REG1 = REG1 + IMMEDIATE16
LDI	Запись непосредственного значения	LDI r1 imm16	REG1 = IMMEDIATE16
LOAD	Загрузка значения в регистр	LOAD r1 imm16	REG1 = MEMORY[ADDRESS16]
STORE	Сохранение значения регистра	STORE r1 imm16	MEMORY[ADDRESS16] = REG1
PUSH	Добавление элемента в стэк	PUSH r1	STACK[SP] = REG1; SP += 1
POP	Удаление элемента из стэка	POP r1	REG1 = STACK[SP]; SP -= 1
RET	Возвращение из процедуры	RET	PC = LR; LR = POP
CALL	Переход в процедуру	CALL imm16	PUSH LR; LR = PC; PC = ADDRESS16
BRC	Ветвление	BRC imm16	PC = ADDRESS16 if

Рисунок 2 – Структура команд архитектуры NERO

Данный набор команд уже делает компьютер полным по Тьюрингу [6].

В виду того, что разрабатываемая архитектура создается с целью обучения низкоуровневому программированию школьников и студентов, для большого комфорта программирования требуется добавить еще ряд команд. Однако все 16 кодов операции уже заняты, поэтому будут использоваться псевдоинструкции.

Псевдо-инструкции не усложняют схему АЛУ и не занимают дополнительной площади на кристалле, а поддерживаются исключительно на стороне ассмеблера, что позволяет более эффективно использовать площадь на кристалле.

Набор регистров был выбран на основе книги Цифровая схемотехника и архитектура компьютера [7] и представлен в таблице 1.

Таблица 1 – Карта регистров архитектуры NERO

REGISTER				NAME	DESCRIPTION	ОПИСАНИЕ
0	0	0	0	REG0	Zero Register	Нулевой регистр
0	0	0	1	REG1	General Purpose Register	Регистр общего назначения
0	0	1	0	REG2	General Purpose Register	Регистр общего назначения
0	0	1	1	REG3	General Purpose Register	Регистр общего назначения
0	1	0	0	REG4	General Purpose Register	Регистр общего назначения
0	1	0	1	SP	Stack Pointer	Указатель стэка
0	1	1	0	FR	Flag Register	Регистр состояния
0	1	1	1	PC	Program Counter	Счетчик команд
1	0	0	0	LR	Link Register	Регистр возврата
1	0	0	1	OPCODE	Operation Code Register	Регистр кода операции
1	0	1	0	OP1	Operand Register	Регистр операнда
1	0	1	1	OP2	Operand Register	Регистр операнда
1	1	0	0	OP3	Operand Register	Регистр операнда
1	1	0	1	IMMR	Immediate Register	Регистр 16-битного значения
1	1	1	0	SR	Shift Register	Регистр сдвига
1	1	1	1	FULL	Full Register	Полный регистр

Ассемблер, разработанный на языке программирования Python, позволяет работать лишь с регистрами общего назначения, однако, путем изменения машинного кода программы вручную можно получить доступ ко всем остальным регистрам.

Главной функцией ассемблера является упрощение процесса программирования и поддержка псевдо-инструкций.

В ходе проведенной работы была успешно спроектирована, реализована и верифицирована RISC-архитектура NERO.

Разработанная архитектура была смоделирована в ПО ModelSim и показала корректность своей работы.

Для удобства программирования был написан ассемблер на языке Python, поддерживающий девять псевдоинструкций.

Список литературы:

1. Zhang N. Moore's Law is dead, long live Moore's Law! [Электронный ресурс] // arXiv. – URL: https://arxiv.org/abs/2205.15011 (дата обращения: 15.11.2025).

2. Walton, J. FFmpeg Devs Boast of Up to 94x Performance Boost After Implementing Handwritten AVX-512 Assembly Code [Электронный ресурс] / J. Walton // Tom's Hardware. – 2024. – Mode of access: https://www.tomshardware.com/pc-components/cpus/ffmpeg-devs-boast-of-up-to-94x-performance-boost-after-implementing-handwritten-avx-512-assembly-code (дата обращения: 15.11.2025).

3. Regi G., Kunnappally J. J., Yunus R. Comparative Analysis of Energy Efficiency Between ARM and x86 Architectures in Mobile Devices and Its Implications for Human-Computer Interaction // Kristu Jayanti Journal of Computational Sciences (KJCS). 2025. Vol. 4, № 1. P. 54–67.

4. Design of 16-bit RISC Processor [Электронный ресурс] / S. Gaonkar. // ISROSET. – URL: https://www.isroset.org/pub_paper/IJSRPAS/ISROSET-IJSRPAP-00068.pdf (дата обращения: 15.11.2025).

5. Let’s Make a Redstone Computer [Видеозапись] // YouTube. – URL: https://www.youtube.com/playlist?list=PL5LiOvrbVo8nPTtdXAdSmDWzu85zzdgRT (дата обращения: 15.11.2025).

6. Turing A. On Computable Numbers, with an Application to the Entscheidungsproblem // Proc. London Math. Soc. – 1937. – Vol. s2-42, Iss. 1. – P. 230–265.

7. Харрис Д. М., Харрис С. Л. Цифровая схемотехника и архитектура компьютера. Дополнение по архитектуре ARM / пер. с англ. А. А. Слинкин; науч. ред. Д. А. Косолобов. – М.: ДМК Пресс, 2019. – 356 с.: ил.