Final Project Report

CPR E 581

Tasman Grinnell, Varun Jain, Cory Roth

Abstract

William Wulf's work "Compilers and Computer Architecture" provides a basis for many principles of Reduced Instruction Set Computer (RISC) Instruction Set Architectures (ISAs) that became prevalent in the earlier days of computing at the beginning of the shift from Complex Instruction Set Computers (CISC) to RISC architectures. The proposed project examines the principles presented by Wulf to determine the lasting effects of his work and to evaluate the longevity of the insights presented. The perhaps most dominant ISA in the modern processor market is the CISC ISA x86 that both Intel and AMD implement. Therefore, examining the potentially lasting insights of his proposed ISA, architecture, and principles may continue to be integrated into the modern processor market.

1.0 Introduction

When discussing the "Compilers and Computer Architecture" paper in the lecture, many speculations on the relevance of Wulf's work were presented, with many related to the predictions that Wulf presented regarding the rise of RISC architectures. Therefore, evaluating or researching the principles presented in the paper and their importance in today's ISAs and architectures is an interesting subject to research.

The primary principles examined in Wulf's paper are regularity, composability, and orthogonality. As written directly from the paper:

- "Regularity: If something is done in one way in one place, it ought to be done the same way everywhere. This principle has also been called the "law of least astonishment" in the language design community.
- Orthogonality: It should be possible to divide the machine definition (or language definition) into a set of separate concerns and define each in isolation from the others. For example, it ought to be possible to discuss data types, addressing, and instruction sets independently.
- Composability: If the principles of regularity and orthogonality have been followed, then it should also be possible to compose the orthogonal, regular notions in arbitrary ways. It ought to be possible, for example, to use every addressing mode with every operator and every data type."

In terms of the insights from these principles, regularity intends to simplify the operating principles of ISAs to reduce the complexity and number of cases in which instructions can be written. Orthogonality allows for different sections (instruction set, data type, and addressing mode) of any complex operation to be dealt with separately. Therefore, adding new instructions or addressing modes will not affect the data types the processor can deal with. Following the two prior principles, composability allows a processor to support every possible permutation of the three sections of an instruction. This allows for greater flexibility when making increasingly complex instructions while keeping the processor components fairly standard.

Altogether, Wulf clearly stated principles that were incredibly important and relevant during the shift from CISC to RISC machines, clearly outlining design principles that were reflected in RISC ISAs. By comparing the principles to the modern day, we can determine the longevity of the insights presented by evaluating the most popular ISAs and processor architectures in the current market. Additionally, these insights can assist in determining if modern processors use any design principles similar to his architecture and ISA proposed in "The WM Computer Architecture," evaluating the validity through high-level comparisons.

2.0 Related Work

Our proposed idea is based on the principles declared in Wulf's "Compilers and Computer Architecture." In this paper, Wulf initially describes the principles defined previously. He proposed these ideas to correct the uncooperativeness of ISAs with compilers and high-level languages [1]. Although the paper does not assess the validity of these principles, it demonstrates the shortcomings of CISC ISAs that were prevalent at that time. This paper aims to formally evaluate modern ISAs along with comparing ISAs that do follow them.

Wulf wrote an additional paper, "The WM computer architecture" in which he proposes the WM architecture. The architecture would theoretically run on an ISA he designed that closely follows those 3 principles he stated. This paper helps elaborate his intentions when describing his principles and gives concrete structures to examine instead of simply proposing an ideology. His proposed model would achieve a theoretical performance of 11 RISCy operations per cycle [2]. This is when 4-5 RISCy operations per cycle were common. The project will not be implementing the architecture itself but determine if the implementation of the principles still applies today in regards to modern-day processors.

Another paper related to our proposed idea would be "A Characterization of Processor Performance in the vax-11/780", proposing an architecture called vax-11/780. The vax-11/780 is a processor running an "orthogonal instruction set," defined as 'an instruction set architecture where all instruction types can use all addressing modes [3].' The orthogonal instruction set definition closely follows Wulf's definition of orthogonality. The paper may be useful in examining how the ISA impacted the development of other ISAs. Also, it may provide some insight into how performant processors are that implement orthogonal instruction sets. However, this paper unfortunately does not compare current-day state-of-the-art processors, so the relevance to our idea is limited.

Another paper that might be related is “An Empirical Comparison of the RISC-V and AArch64 Instruction Sets.” The paper does not evaluate RISC-V or AArch64 based on Wulf’s principles but compares modern ISAs. RISC-V and AArch64 are RISC implementations. Wulf’s proposed ISA was also a RISC implementation. Therefore, looking at the metrics of this comparison would be beneficial to our analysis. The results show the instruction sets are relatively closely matched on the metrics the paper evaluated for the benchmarks the paper considered, indicating that neither ISA has a large inherent architectural advantage over the other [4]. This insight can be useful in determining how important an evaluation is. Our proposed paper could do an ISA comparison between these two ISAs to see which one holds more similarities to Wulf's principles and look at their performance results.

Some other broad areas of the project that will be further examined involve designing ISAs. Considering how organizations go about creating and refining an ISA can assist in determining whether current processes are representative of the principles we are evaluating. Another paper titled “How to Design an ISA” [6] that provides a summary of different ISAs, why some are still used, and why some are no longer used. Although the ideas are relatively high-level and merely provide a simple foundation for designing an ISA, a valuable comparison of x86 and RISC-like ISAs is provided. The paper being proposed would do a more in depth analysis of comparisons.

Additionally, challenges in ISA compatibility exist in application specific instruction-set processors (ASIPs), as demonstrated by Zhao et al. [9]. Compatibility issues exist primarily in specialized applications where ISAs must be reduced to fit the specialized processor due to the limited hardware requirements. The issues primarily involve compilers being unable to restrict the compiled code to match the hardware specifications, resulting in essential drivers and firmware forcing to be programmed manually in assembly. Therefore, Zhao et al. propose a static resource model to assist in instruction set design to solve the issues arising from the specialized data flows of ASIPs, avoiding large amounts of low-level programming and the associated labor. Concerning the project, the issues in compiler compatibility and general development are a basis of Wulf’s original paper, with a portion of the project examining differences in compilation per optimization level using GCC. Therefore, challenges of cross-compatibility for ASIPs may provide interesting extensions for the project or background to assist in analyzing compiler performance optimizations.

Further examinations of ASIP ISA design have been explored by Ali et al. [10] by exploring the design process and proposing a profiling tool that assists in evaluating the RISC-V ISA instruction coverage per application to target ASIP development specifically. Through the profiler, efforts are efficiently targeted for System-on-chip (SOC) improvements in ASIP development, resulting in highly efficient chips. Although the applications are limited relative to this work, Ali et al.'s work indirectly motivates an understanding of the adherence to Wulf's Principles and how further ISA optimizations can be targeted.

3.0 Motivation/Main Idea

In William Wulf's "Compilers and Computer Architecture" paper, Wulf proposed principles of orthogonality, regularity, and composability that computer architects should assist in implementing to assist compiler designers in creating highly performant and easing the design process for compilers. The principles seemed to have predicted the rise of Reduced Instruction Set Computer (RISC) Instruction Set Architectures (ISAs) since the paper was published in the years up to the transition from Complex Instruction Set Computers (CISC) to RISC machines. Because of this rapid transition, the question of the relevance of the principles was brought up in discussion, motivating the inspection of current ISAs with the principles presented to determine the lasting relevance or if alternate hardware advancements have rendered the principles useless or ineffective.

Additionally, Wulf followed up later with the proposal of the Wulf Machine, a RISC-like architecture that fuses multiple RISC-like operations together in a single instruction. Wulf claimed that this architecture outperformed other RISC architectures by roughly double the performance (peak 11 RISC operations per cycle compared to 5-6 operations by other architectures [2]). Initial proposals for the project focused on the implementation of the ISA to determine if the theoretical benefit was possible and additionally comparing the performance to a baseline Out-of-Order (OOO) processor to further compare the results to more modern architectures, improving pipelining and execution of instructions with the OOO execution. However, evaluating the WM machine was determined to be impossible since 1. Compiling SPEC2006 Benchmarks in a different ISA that doesn't have working or maintained compilers; 2. Implementing the architecture in the gem5 simulator would be difficult due to the unique architecture, as shown in Figure 1.

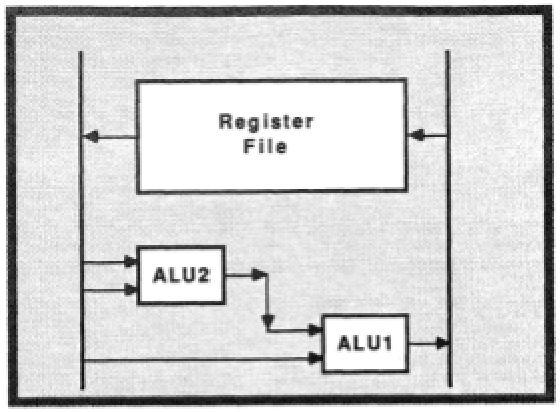

Figure 1: The Wulf Machine's Chained ALU paths, allowing for multiple operations to be executed per cycle.

Due to the aforementioned issues, the focus was shifted to simply evaluating the modern ISAs to the principles, proposing to modify LLVM Compiler backend to reduce/improve adherence to the principles. However, more issues were discovered in the process of attempting to modify the LLVM Backend, those being:

1. Files required to modify the LLVM Backend were locked or inaccessible;
2. Documentation is incredibly limited, resulting in many difficulties attempting to modify the backend.

Therefore, the project focus was shifted again to comparing the performance of SPEC2006 benchmarks in gem5 on the various ISAs for different optimization levels and using a baseline model, closing the gap between Intellectual Property of AMD and Intel that may create performance gaps relative to the RISC processors implemented for ARM and MIPS architectures.

Manually inspecting the ISAs through the reference manual will glean information regarding the adherence to the principles, and using the varying optimization levels can assist in understanding the differences in performance. Additionally, the simulations and binary analyses can further enforce an understanding of compiled instructions that adhere to each principle since compilers may optimize many instructions in the ISA itself.

4.0 Methodology

4.1 Choosing Instruction Set Architectures

In choosing the Instruction Set Architectures to be compared, the primary targets chosen for comparison were ARM and x86 due to their market presence across various platforms. x86 is well known to be highly performant in both server and consumer applications, with x86 processors dominating the performance-hungry gaming market and being used heavily in the workstation sector. Additionally, ARM is used greatly by Apple, an incredibly popular brand for phones, laptops, and PCs with their Macs. Therefore, ARM and x86 were chosen as the primary sets for inspection due to their dominant performance and market share.

Using a RISC instruction set as a baseline is necessary, with MIPS chosen due to varying factors, including the team's personal familiarity and a source of curiosity and interest. x86 is well-known as Complex Instruction Set Computer (CISC) ISA, but comparing the performance of benchmarks is an interesting source of comparison using an identical baseline. Through the baseline, the simulation results can glean additional information about the company's success in creating highly performant processors while still using a CISC ISA. Furthermore, insight into the performance sources being closely related to the microarchitecture or the alternate optimizations is a source of interest.

4.2 ISA Inspection

As a team, the instruction sets determined to be examined were ARM-AA32, MIPS32, and x86 (base instruction set). A subset of each instruction set was determined to be examined based on the core functionality of the instruction set, excluding extensions of the instruction set (e.g., SIMD, Vector), and examined via ISA Reference Manual.

Each instruction was evaluated based on their adherence to Wulf's principles of Orthogonality, Regularity, and Composability. The following definitions of Orthogonality and Composability were used:

- Orthogonality - If instructions do not have reserved registers for use, then they are orthogonal;
- Composability - If instructions are both orthogonal and regular, then they are composable.

The definition of regularity was tailored to each instruction set due to the differences in design for each ISA. The following definitions of regularity were used for each ISA:

- MIPS - If an instruction's format is one of R, I, J, FR, or FI, then it is regular;
- ARM - If an instruction uses the S and cond fields, then it is regular;
- x86 - If an instruction uses all combinations of register and memory size references (8, 16, 32, and 64-bit accesses), then it is regular.

The instructions in MIPS assembly have a fixed 32-bit size and bit fields associated with R, I, J, FR, or FI type instructions, providing a good baseline for regularity. Additionally, the addressing modes of the language are very strict (can only reference registers in instructions), suggesting that adherence to the instruction formats is the baseline of regularity for MIPS. Some notable examples of instructions that do not adhere to Wulf's principles are `mfhi` and `mflo`, non-orthogonal instructions that use the HI and LO registers as sources. Additionally, `cop2` uses a unique instruction format to communicate with a coprocessor.

Similarly, ARM-AA32 uses 32-bit length instructions, but the primary determination point was due to the absence of bit fields in the instruction set manual. Therefore, the S and cond fields were determined to be used as the baseline for regularity due to the appearance in many arithmetic instructions. Some examples of non-adhering instructions are `cdp` and `cdp2` – ARM's coprocessor instructions – which use unique fields, making them non-regular instructions. Additionally, `subs pc,lr` is a special instruction that uses the PC and lr as the source and destination registers to perform specific operations.

Defining the principle of regularity specifically for x86 was difficult due to the complex nature of the addressing modes utilizing many different register and memory addressing methods. For instance, the Add instruction in x86 can address 8, 16, 32, and 64-bit registers or memory locations, while many instructions do not use these modes. Therefore, all evaluated instructions were examined to determine how many instructions use all of the addressing modes versus a limited set. Furthermore, the selection of instructions used only for core functionality accounts for SIMD and Vector type instructions that use a separate set of registers for the instructions. Thus, the instructions are further enforced as core functionality that do not explicitly target a specific optimization. Some principle-defying instructions are `scas`, a non-orthogonal instruction implicitly using the A register (AL, AX, EAX, RAX) to compare to a string in memory. The non-regular `btr` instruction uses almost all register and memory reference sizes, only missing the 8-bit register reference.

Then, instructions from the chosen listing were examined relative to the principles and determined to be regular, orthogonal, or composable. Then, percentages were taken of each principle, inserted into a CSV, and finally graphed using MatPlotLib.

4.3 Compilation and Static Binary Analysis

To identify if these principles are being adhered to and the consequences, disassembled SPEC 2006 benchmarks were examined. The benchmarks provide insight into modern programs, their workloads, and what instructions are used. If compilers are creating programs that use a lot of instructions that do not adhere to these principles, alternative instructions may be more ideal. Another investigated idea was acquiring the instruction mix of these programs, allowing us to find candidates to remove from the LLVM (i.e., if many `mov` instructions are present, then removing or changing the addressing can reduce/improve adherence to the principles).

The benchmarks chosen to compile were:

- Bzip2;
- GCC;
- GoBMK.

The benchmarks were chosen in particular due to their varying sizes. Bzip2 is a relatively small benchmark regarding the number of static instructions, whereas GCC is a large program. GoBMK's size is somewhere in between. Having different-sized benchmarks also allows us to see if the program's size impacts the adherence to the principles. Each of these benchmarks was compiled at optimization levels zero, one, two, and three (using flags -O0, -O1, -O2, and -O3) to determine if different compilation levels affected the binary's compilation.

To determine which instructions were used in these programs, gem5 was used alongside runspec to generate the binaries. As a result, an executable and a folder of .o files were generated from the compiled C code, which was then passed into a Python script to count the instructions. The Capstone library [8] disassembled the binaries and acted as a dictionary for finding the instruction counts in each ISA. The .txt section of each .o file was examined for the instruction mix analysis, and the script output a count of each instruction used for each optimization level for each compiled binary. Another Python script was also used to group instructions into Arithmetic, Memory, Control, or Nop for determining the instruction mix.

4.4 Simulation

Originally, modifying the LLVM compiler backend was the primary interest due to the specially tailored compilation of the binaries that would be interesting to perform. Still, various issues, such as limited documentation and restricted access to files, forced us to find an alternative. Therefore, strictly comparing each ISA's performance on a baseline model was decided to determine differences between ISAs purely based on the instruction set itself rather than differences in hardware differences and implementations.

Each binary will be simulated using a baseline model reflective of an Out-of-Order (OOO) Alpha 21264 CPU. The hardware specifications are as follows:

- 3 GHz Clock Frequency
- 2GB Address Space
- 64KB 2-way Associative L1 DCache
- 32KB 2-way Associative L1 ICache
- 2MB 8-way Associative L2 Cache

Each benchmark will be run for 100 million instructions, a size similar to the provided simpoints previously used in the course.

The baseline OOO model aims to gain a baseline perspective of processor performance per ISA, flattening performance gains acquired from specialized structures in modern commercial processors, such as micro-operations in many x86-based cores produced by Intel or AMD. The Alpha 21264 model is also the primary model that gem5 was modeled off, so using similar hardware structures close to a realistic view of the processor is ideal for simulation.

5.0 Results

5.1 ISA Inspection

Each instruction set had at least 60% of instruction adhering to each principle, with x86 adhering the lowest to composability at 62.95%. MIPS32 and ARM-AA32 are generally very orthogonal at 96.43% and 96.79%, respectively, while x86 is 72.77%. Each ISA was generally not very regular due to the many special instructions (e.g., coprocessor communication, operating system support, etc.), with all instruction sets hovering around 69% (+/- 2%). Figure 2 visually presents the adherence percentages clustered by principle, while Table 1 lists each percentage up to a hundred-thousandths.

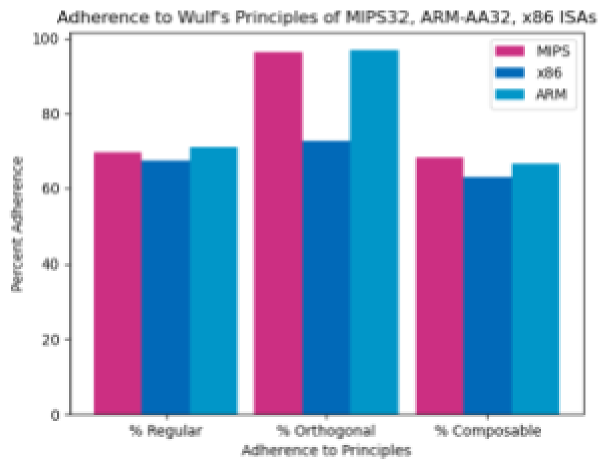

Figure 2: Bar Graph of ISA adherence clustered by principle.

ISA	Percent Regular	Percent Orthogonal	Percent Composable
ARM	71.1009	96.7890	66.5138
MIPS	69.6429	96.4286	68.3424
x86	67.4107	72.7679	62.9464

Table 1: Percents of Regularity, Orthogonality, and Composability of each chosen ISA.

5.2 Compilation and Static Binary Analysis

5.2.1 Principle Adherence per Binary

Bzip2:

Each instruction set had at least 76% of instruction adhering to each principle, with x86 adhering the lowest to composability at 79.03%. Figure 3 visually presents the adherence percentages clustered by principle, while Table 2 lists each percentage up to a hundredth.

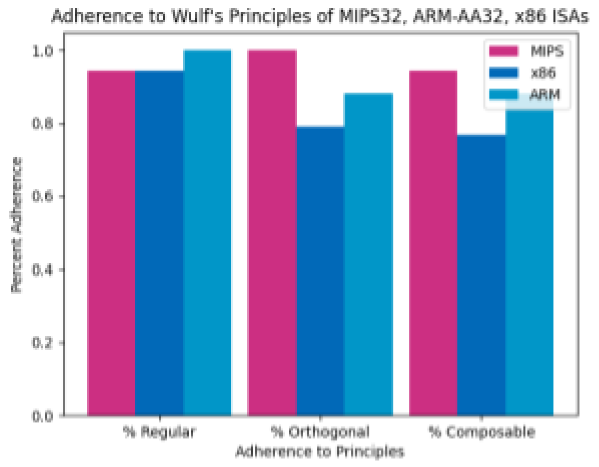

Figure 3: Instruction adherence in the Bzip2 benchmark.

ISA	% Regular	% Orthogonal	% Composable
ARM	99.85	88.06	85.97
MIPS	94.26	99.94	94.26
x86	94.50	79.03	76.68

Table 2: Adherence percentages per ISA for the Bzip2 SPEC2006 benchmark with compile flag -03.

GCC:

Each instruction set had at least 77% of instruction adhering to each principle, with x86 adhering the lowest to Composability at 77.06%. MIPS and ARM had slightly lower percentages of regularity and orthogonality relative to the Bzip2 binaries. Figure 4 visually presents the adherence percentages clustered by principle, while Table 3 lists each percentage up to a hundredth of a percent.

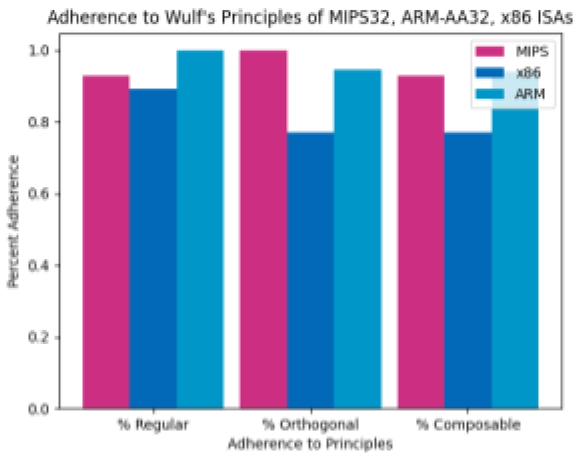

Figure 4: Instruction adherence in the GCC benchmark.

ISA	% Regular	% Orthogonal	% Composable
ARM	99.89	94.55	94.05
MIPS	92.86	99.94	92.91
x86	89.26	77.06	77.06

Table 3: Percents of Regularity, Orthogonality, and Composability per ISA for the GCC SPEC2006 benchmark with compile flag -03.

GoBMK:

Each instruction set had at least 69% of instruction adhering to each principle, with x86 adhering the lowest to Composability at 69.12%. Figure 5 visually presents the adherence percentages clustered by principle, while Table 4 lists each percentage up to a hundredth.

ISA	% Regular	% Orthogonal	% Composable
ARM	99.79	87.83	87.53
MIPS	94.70	99.98	94.70
x86	85.64	69.23	69.12

Table 4: Adherence percentages per ISA for the GoBMK SPEC2006 benchmark with compile flag -03.

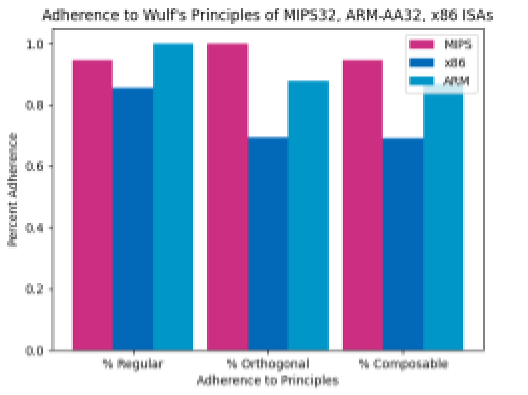

Figure 5: Instruction adherence for the GobMK benchmark.

5.2.2 Instruction Mix

Bzip2:

ISA	Optimization Level	# of Total Instructions	% Arithmetic	% Memory	% Control	% Nop
Arm	-o0	32,377	42.05%	50.15%	7.24%	0.55%
Arm	-o1	14,127	40.18%	46.29%	13.34%	0.20%
Arm	-o2	13,446	37.32%	49.53%	12.85%	0.30%
Arm	-o3	14,787	38.33%	51.78%	9.74%	0.16%
MIPS	-o0	27,324	27.23%	55.03%	8.51%	9.24%
MIPS	-o1	14,258	34.80%	45.50%	14.64%	5.05%
MIPS	-o2	14,027	34.85%	48.85%	13.98%	2.32%
MIPS	-o3	18,058	36.97%	47.81%	13.00%	2.22%
x86	-o0	23,439	23.85%	64.87%	10.64%	0.63%
x86	-o1	11,530	27.89%	53.65%	18.46%	0.00%
x86	-o2	11,788	29.47%	50.26%	16.63%	3.64%
x86	-o3	18,886	29.12%	55.24%	13.32%	2.33%

Table 5: Instruction mix by ISA and optimization level for the Bzip2 benchmark.

Table 5 shows the results of counting all the instructions disassembled and grouped into one of the 4 categories. Figure 6 shows ARM has around a 38% arithmetic instruction mix, whereas MIPS and x86 were 36% and 28%, respectively. ARM only had .3% NOPs, whereas MIPS had 5% and x86 had 3%. ARM had the most instructions at optimization level zero at 32,377, whereas x86 had the most at optimization level three at 18,886.

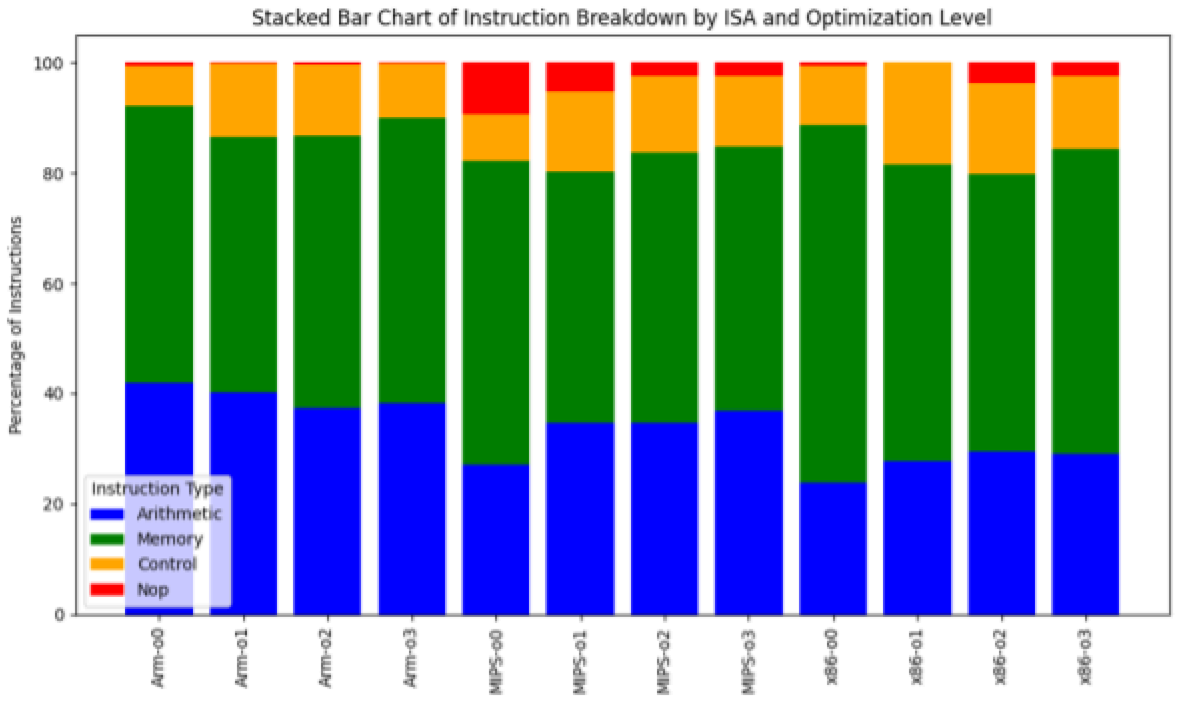

Figure 6: A comparison of grouped instruction mix by ISA and optimization level for Bzip2.

GCC:

ISA	Optimization Level	# of Total Instructions	% Arithmetic	% Memory	% Control	% Nop
Arm	-o0	1,037,734	20.36%	63.34%	15.25%	1.06%
Arm	-o1	327,544	20.79%	58.13%	20.75%	0.33%
Arm	-o2	310,893	20.00%	60.45%	19.03%	0.52%
Arm	-o3	320,068	20.59%	59.45%	19.47%	0.49%
MIPS	-o0	1,327,436	15.68%	55.26%	14.01%	15.05%
MIPS	-o1	848,427	20.65%	51.16%	20.00%	8.19%
MIPS	-o2	637,674	21.17%	55.11%	20.85%	2.87%

MIPS	-o3	861,084	21.83%	54.38%	20.75%	3.03%
x86	-o0	939,301	16.43%	61.01%	21.08%	1.49%
x86	-o1	583,604	20.27%	50.59%	29.14%	0.00%
x86	-o2	601,634	22.67%	46.62%	26.78%	3.93%
x86	-o3	673,504	23.22%	46.42%	26.52%	3.84%

Table 6: Instruction mix by ISA and optimization level for the GCC benchmark.

Table 6 shows the results of counting all the instructions disassembled and grouped into one of the 4 categories. Figure 7 shows that ARM has around a 20% arithmetic instruction mix, whereas MIPS and x86 were 21% and 22%, respectively. ARM only had .3% NOPs, whereas MIPS had 8% and x86 had 3%. MIPS had the most instructions at optimization level zero at 1,327,436, and MIPS had the most at optimization level three at 861,084.

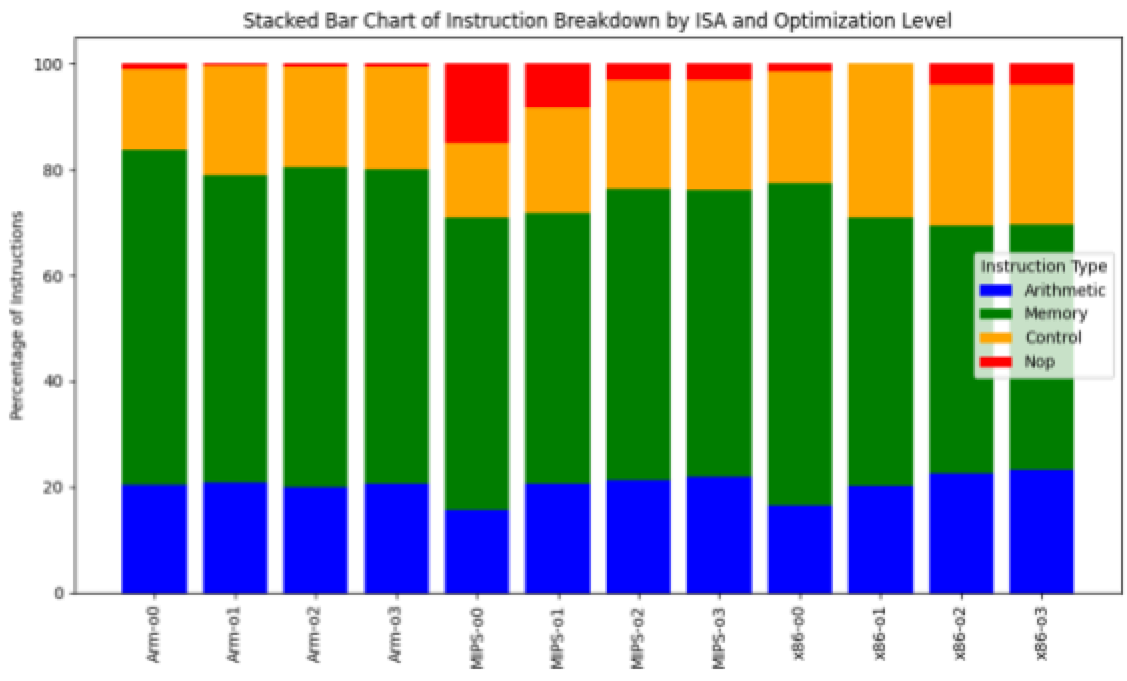

Figure 7: A comparison of grouped instruction mix by ISA and optimization level for GCC.

GoBMK:

ISA	Opt. Level	# of Total Instructions	% Arithmetic	% Memory	% Control	% Nop
Arm	-o0	298,685	32.39%	57.71%	9.17%	0.73%

Arm	-o1	172,486	26.33%	57.97%	15.11%	0.59%
Arm	-o2	171,653	29.14%	55.86%	14.32%	0.68%
Arm	-o3	171,675	28.03%	56.02%	15.30%	0.66%
MIPS	-o0	279,757	26.76%	53.75%	9.59%	9.91%
MIPS	-o1	176,188	32.65%	47.85%	13.95%	5.55%
MIPS	-o2	163,466	33.38%	51.12%	14.14%	1.36%
MIPS	-o3	175,664	33.62%	50.32%	14.54%	1.53%
x86	-o0	257,824	20.50%	64.26%	14.78%	0.46%
x86	-o1	154,714	21.70%	56.56%	21.74%	0.00%
x86	-o2	163,060	24.27%	51.65%	19.96%	4.13%
x86	-o3	186,773	24.73%	51.10%	20.27%	3.90%

Table 7: Instruction mix by ISA and optimization level for the GoBMK benchmark.

Table 7 shows the results of counting all the instructions disassembled and grouped into one of the 4 categories. Figure 8 shows that ARM has around a 28% arithmetic instruction mix, whereas MIPS and x86 have 33% and 22%, respectively. ARM only had .4% NOPs, whereas MIPS had 3% and x86 had 2%. ARM had the most instruction at optimization level zero at 298,685 instructions statically, and x86 had at most at optimization level three at 186,773 instructions.

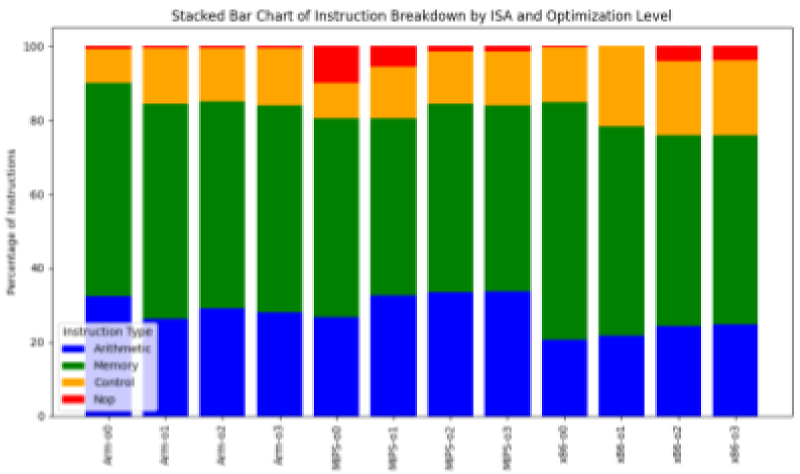

Figure 8: A comparison of grouped instruction mix by ISA and optimization level for the gobmk benchmark.

5.2.3 Compilation Time

During the compilation process for each binary, varying times were noticed to occur, with optimization levels, benchmark, and ISA times being noticeably different. Therefore, the results were gathered in Table 8 and Figures 9-11.

Overall, x86 received the fastest compilation times at the majority of levels, with the only occurrence of a tied or faster compile time held by MIPS and ARM for the Bzip2 binary at optimization levels zero, one, and three. At optimization level zero, MIPS compiled at the same speed as x86, and MIPS and ARM compiled one second faster at levels one and three. However, x86 consistently compiled faster than ARM and MIPS for GCC and GoBMK

ISA	**Optimization Level**	**Bzip2 (seconds)**	**GCC (seconds)**	**GoBMK (seconds)**
Arm	-o0	4	32	12
Arm	-o1	4	64	20
Arm	-o2	5	100	28
Arm	-o3	6	113	32
MIPS	-o0	3	31	12
MIPS	-o1	4	62	20
MIPS	-o2	5	100	29
MIPS	-o3	6	114	32
x86	-o0	3	26	10
x86	-o1	5	52	17
x86	-o2	5	80	23
x86	-o3	7	91	27

Table 8: Compilation time for each binary by ISA and optimization level.

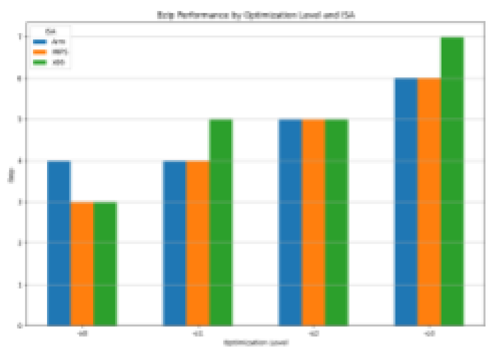
Figure 9: Compilation time per optimization level and ISA for the Bzip2 Benchmark.

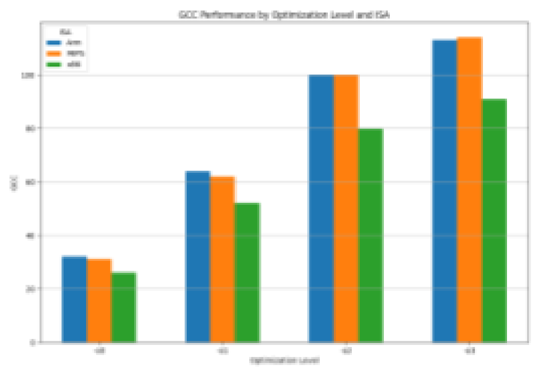
Figure 10: Compilation time per optimization level and ISA for the GCC benchmark.

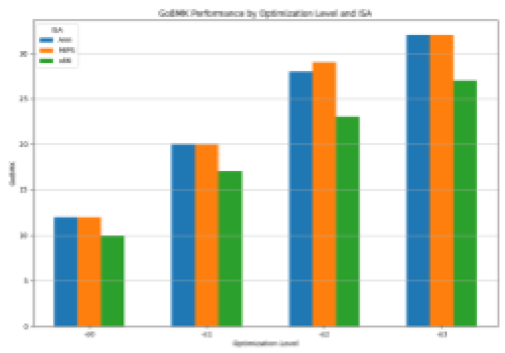
Figure 11: Compilation times per ISA and Optimization level for the GoBMK benchmark.

5.3 Simulation

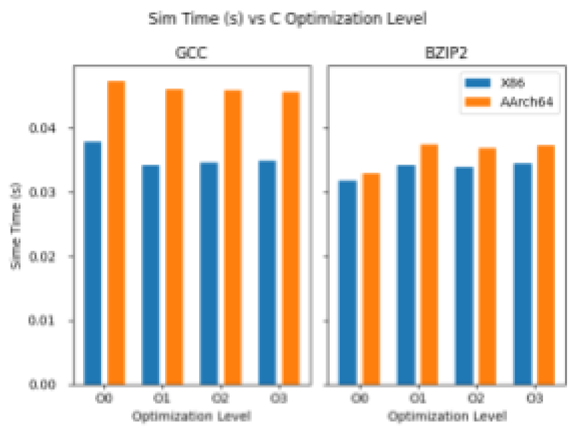

Figure 12: Simulated runtime versus optimization level for x86 and ARM-AA64.

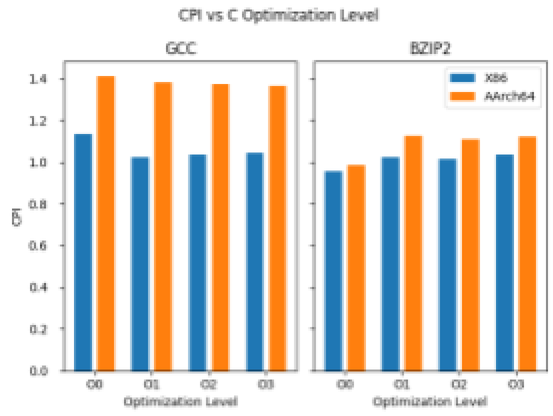

Figure 13: CPI versus optimization level for x86 and ARM-AA64.

Overall, the benchmarks run on the ARM ISA performed with higher CPI and simulated runtime using the baseline specifications. ARM had an average simulated runtime of about 0.046 seconds for GCC and 0.036 seconds for Bzip2, while x86 maintained an average of 0.035 seconds for GCC and 0.034 seconds for Bzip2. Furthermore, ARM's CPI was about 1.38 for GCC and 1.08 for Bzip2, whereas x86's average CPI was 1.06 for GCC and 1.01 for Bzip2.

The MIPS processor simulation failed due to a lack of support for MIPS as a research ISA, resulting in the failure of simulation for MIPS for comparison. Although certain system files were present in the simulator, many portions imperative for simulation were unavailable. Therefore, ARM and x86 were the only ISAs simulated for the project.

In attempting to simulate the ARM-AA32 and MIPS binaries, many issues were encountered due to a lack of support and missing documentation. Although x86 simulations were performed without significant issues, the ARM-AA32 ISA lacks support, resulting in fatal syscall errors caused by implementation details missing from the se_workload.cc file for Linux under ARM in gem5. Therefore, ARM-AA64 was used for the simulation to acquire results for analysis, with configuration files acquired via request of others in the gem5 community who've implemented the syscall files.

6.0 Discussion of Results

6.1 ISA Inspection

Regarding the ISA inspection, the measured adherence was much closer than expected, with x86 being only 2% lower than the next least regular ISA, MIPS. Much of the lack of regularity was due to many instructions supporting multiprocessing applications, such as the `cop2`, `cdp`, and `cdp2` instructions in MIPS and ARM. Similarly, support for operating systems requires unique instructions, similarly hurting adherence to the principles of regularity. Therefore, although the regularity of x86 was much closer to the adherence of MIPS and ARM, the general results were expected.

On the other hand, the MIPS and ARM had vastly higher adherence for orthogonality, about 25% higher than x86. The vastly increased orthogonality was unexpected, as the general expectations were that the results of orthogonality and regularity would be swapped, seeing as x86 is known to be a CISC ISA. However, the variable length instructions were determined to be the baseline for the ISA due to the bias to fixed instruction sizes that would cause the adherence to obviously be zero (0). However, the effect of the variable length instructions doesn't significantly impact overhead for power or performance, as discussed in "Power struggles: Revisiting the RISC vs. CISC debate on contemporary ARM and x86 architectures" [5].

Composability was also generally expected since the definition directly depends on both the regularity and orthogonality of instruction, resulting in the percent being slightly lower than the regularity of each ISA. Notably, ARM and x86 are much less composable (~5% less composable relative to regular), while MIPS is only 1% less composable than regular.

6.2 Compilation and Static Binary Analysis

Regarding the static analysis, there seem to be some noticeable differences regarding adherence to Wulf's principles rather than just from the ISA manuals. Regularity was around 90% for each ISA for each program despite being only around 70% for the manual. This can be attributed to the fact that not every instruction in the manual gets used for each progarm. For x86, instructions like `mov`, `cmp`, `call`, and `jmp` dominate the instruction count. These instructions are regular, so the percentage of regular instructions would be higher than found in the ISA reference manual inspection. An interesting note is that the orthogonality is very similar for MIPS and ARM, but for x86, it is slightly higher but more similar to the low value in the manual. It was 72% for the manual and an average of 75% for the 3 benchmarks.

When looking at the instruction mixes, it can be seen that the high-level instruction mix does not have a large change per compilation level. For x86 GCC, the ratio of arithmetic instructions remained around 20%, consistent across ISAs for a program. An interesting note is that the major difference between optimization levels zero and one compilation for the x86 instruction mix was the removal of all NOPs. However, moving to optimization levels two and three returned some NOPs to the compilation process. Furthermore, the number of instructions generated decreases from optimization levels zero and one but increases when moving to levels two and three. An educated guess for the underlying reasoning is that moving from level zero to one removes unnecessary and redundant instructions, whereas going to levels two and three will try to introduce new optimizations like loop unrolling, improving runtime performance while increasing the number of static instructions in the binary.

Examining the compilation time for the benchmarks, we observe that the time increases as the optimization level increases. However, x86 takes the least time to compile the 3 ISAs. It is hard to tell if it is because the compiler is probably more optimized for x86 or that Wulf’s principles have no meaning on the timing for compilation.

6.3 Simulation

Generally, x86 outperformed ARM on all optimization levels for both benchmarks in simulated runtime, with a 0.011-second lead in GCC and a 0.002-second lead for Bzip2. Also, x86’s CPI was about 0.2 lower for GCC and 0.1 for Bzip2. These results show that on a baseline architecture, x86 outperforms ARM in simulated runtime and CPI.

Therefore, a few points can be drawn from these results, primarily with the behavior of each ISA. It is well known that ARM is a RISC ISA, meaning that the instructions are fixed length, and by the nature of RISC ISAs, the instructions perform fewer tasks per instruction relative to x86, a CISC ISA. Additionally, we can observe that x86 performs relatively better in CPI, resulting in a lower simulated runtime. Therefore, we can say that x86 generally performs better at the same task relative to RISC-based competitors.

7.0 Future Work

Some potential continuations of this project can be to dig deeper into alternate ISAs and extensions to determine optimization sources and the driving force between differences in performance from ISAs and specific implementations (specifically microarchitecture and microcode). Additionally, examining the specific design of ISAs may assist in understanding design choices that have allowed architectures to capitalize on performance bottlenecks that other ISAs cannot uniquely. A specific example is in the microcode associated with Intel and AMD's x86 implementations, which could be incredibly important due to the cutting-edge research allowing the companies to stay on top of the market for performance in various applications. Obviously, the companies' Intellectual Property would likely stifle efforts or render the microcode analysis impossible, but the sources of performance gain would still be very interesting to examine.

Expanding the ISA inspection to include extensions such as SIMD and Vector instructions can reveal implementation information that was obviously missed due to the project's scope. The specific implementations have the potential for increasing performance compared to other ISAs. However, the performance gain is still dependent on the prevalence of those instructions in the final compiled binary.

In terms of analyzing the instruction mixes and manual inspection, giving instructions weights depending on the practical utilization can assist in giving a more realistic view of the ISA adherence due to the large number of instructions that may be unused when executing code. Extending simulation to include benchmarks reflective of alternate applications (e.g., scientific computing) can also expose differences in implementation between ISAs.

Furthermore, continuing efforts to modify the LLVM compiler backend would be much more feasible in the timespan of a research project rather than a course project, with the additional time allowing for our original experiment. Although the WM Machine implementation would also be entirely possible, using the WM Architecture as a comparative point wouldn't be as crucial due to the already outdated nature of the architecture. Therefore, although comparing benchmark statistics from the WM machine to the baseline OOO processor would be interesting, it is not necessary.

For MIPS, a lack of support prevented simulation results from being gained, thus raising the question of continuing implementation or switching to use RISC-V, a very similar ISA that currently has much more support in research. A possible extension could be to fully implement a MIPS core and compare the results to a RISC-V implementation to compare one of the most popular RISC ISAs to MIPS, ARM, and x86. Therefore, remedying these issues and potentially modeling a more complex model as the baseline will yield more relevant information.

8.0 Conclusion

Through the various observations at each project level, we can see that Wulf's Principles of orthogonality, regularity, and composability are still somewhat relevant for ISAs that currently dominate the market. Although x86 is generally non-adherent to Wulf's principles, many of the instructions used in compilation generally adhere to Wulf's principles, primarily with the principle of regularity as opposed to the other two principles. In statically analyzing the binaries, we observed that over 80% of instructions are regular. Still, orthogonality and composability are not as essential in the creation and execution of binaries, with at most 80% of instructions being both composable and orthogonal, further emphasizing the core philosophy of regularity seen in RISC ISAs being essential.

Additionally, the simulation results further enforce the observations due to x86 – a CISC ISA – outperforming one of the most popular RISC ISAs in the market. Previously, we speculated that the lower-level implementation details (micro-operations) and their optimizations may have driven tech giants AMD and Intel. However, we can see that x86 generally outperforms its competitors, only reinforcing the efficacy of x86 as an ISA in the market. Although the principles may still be useful in designing ISAs, they are not an absolute rule.

However, there is still uncertainty with the implementation details of x86 in gem5 due to the inner workings of the simulator itself. There may be complications in the implementation details that have not been addressed. For example, Akram et al. [7] found inconsistencies in the x86 Out-of-Order pipeline modeling the Haswell architecture in gem5 by comparing simulations to hardware for various benchmarks. Further work will be required in examining modern ISAs to determine if simulation results are accurate and if the results are truly reflective of x86 and ARM processors.

References

[1] W. A. Wulf, "Compilers and Computer Architecture," in Computer, vol. 14, no. 7, pp. 41-47, July 1981, doi: 10.1109/C-M.1981.220527.

[2] W. A. Wulf, “The WM computer architecture,” *ACM SIGARCH Computer Architecture*

News, vol. 16, no. 1, pp. 70–84, Mar. 1988. doi:10.1145/44571.44577

[3] J. S. Emer and D. W. Clark, “A Characterization of Processor Performance in the vax-11/780,” *ACM SIGARCH Computer Architecture News*, vol. 12, no. 3, pp. 301–310, Jan. 1984, doi: 10.1145/773453.808199.

[4] D. Weaver, S. McIntosh-Smith, "An Empirical Comparison of the RISC-V and AArch64 Instruction Sets," in *Proceedings of the SC '23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*, 2023, pp. 1557–1565, doi: 10.1145/3624062.3624233.

[5] E. Blem, J. Menon and K. Sankaralingam, "Power struggles: Revisiting the RISC vs. CISC debate on contemporary ARM and x86 architectures," in 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA), Shenzhen, 2013, pp. 1-12, doi: 10.1109/HPCA.2013.6522302.

[6] D. Chisnall. "How to Design an ISA: The popularity of RISC-V has led many to try designing instruction sets," in *Queue*, vol. 21, no. 6, pp. 27–46, 2024, doi: 10.1145/3639445

[7] A. Akram and L. Sawalha, "Validation of the gem5 Simulator for x86 Architectures," 2019 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS), Denver, CO, USA, 2019, pp. 53-58, doi: 10.1109/PMBS49563.2019.00012

[8] Nguyen Anh Quynh and others, "Capstone: The Ultimate Disassembly Framework," [Online]. Available: https://github.com/capstone-engine/capstone. [Accessed: Dec. 16, 2024].

[9] Qin Zhao, B. Mesman and T. Basten, "Practical instruction set design and compiler retargetability using static resource models," Proceedings 2002 Design, Automation and Test in Europe Conference and Exhibition, Paris, France, 2002, pp. 1021-1026, doi: 10.1109/DATE.2002.998425.

[10] M. Ali, E. Aliagha, M. Elnashar and D. Göhringer, "RV-ProViler: Evaluating RISC-V ISA for Application-Specific Requirements," 2024 IEEE Nordic Circuits and Systems Conference (NorCAS), Lund, Sweden, 2024, pp. 1-7, doi: 10.1109/NorCAS64408.2024.10752449.