SAN JOSÉ STATE UNIVERSITY

Department of Electrical Engineering

ADVANCED COMPUTER ARCHITECTURE

PROJECT

CURRENT STATE OF ART: RISC-V

Submitted to –

Prof. Chang Choo
Director of AI FPGA/DSP Hardware Laboratory

Submitted by –

Shivang Singh
MSEE

-- May 14th, 2018 –

TABLE OF CONTENTS

WHAT IS RISC-V?

RISC-V is an open source instruction set architecture (ISA), which is based on reduced instruction set computing (RISC) principles. It is also a free architecture available to everyone in the world. Due to its free availability, it gives everyone a permission and chance to make changes, design, manufacture and then sell RISC-V chips and softwares. It is extremely useful since this is used for and with modern computerized devices, including cloud computers, high-end mobile phones and even with the smallest embedded devices. In other words, we can also state that it is the one who tells the way in which the software talks to the underlying processor, just as in case of x86 ISA for Intel/AMD processors and the ARM Version 8 ISA for the latest and greatest ARM processors [1]. This characterization also leads consideration with respect to its performance and efficiency.

RISC-V project is a relatively new technology, starting with less than a decade ago at University of California, Berkeley in Silicon Valley in the year 2010. It started with 32 bits but now has operation with 32, 64 and 128 bits too [2]. The standards of the RISC-V are maintained by RISC-V foundation, with the foundation members coming from various industries, including software, systems, semiconductors and IP. The focus of the member companies is on making a rich ecosystem like environment for hardware and softwares that surpasses other companies such as ARM. The development team at UC Berkeley uses software tools internally within themselves in the last three years and are still actively developing them ahead. Moreover, many programmers who are outside UC Berkeley are also contributing to the code and fixing bugs as a part of open-source effort for the RISC-V tools.

It is a very important component for the field of computer architecture. Its significance can be easily understood by the fact that its designers consider both performance and power efficiency, since it is highly demanded in large scale cloud computers, high-end mobile phones and in the smallest embedded systems. The instruction set is also extremely valuable along with the supporting software, which avoids any usual flaws in the instructions set within it. It is designed with small, fast, low powered and along with real world implementations. The RISC0V processors are for sure much better and fast processors when compared to x86 or Arm processors, since it depends totally on the quality if the implementation, which includes the micro-architectural design, the circuit design and the process of techniques which is being utilized.

The RISC-V ISA project is being funded by multiple research projects, within and outside UC-Berkeley and around the globe. These sponsors include giant companies like Intel, Nokia, Google, NVIDIA, Oracle, Samsung and many more.

ESSENCE OF RISC-V

RISC-V is an extremely important part of many processors. RISC-V is the main interface in a computer, since it lies between the hardware and the software and performs the task of linking to each other. If a good instruction set was open, available for use by all, it should dramatically reduce the cost of software by permitting far more reuse. It should also increase competition among hardware providers, who can use more resources for design and less for software support [3].

The RISC-V designing engineers focus on that the new principles are becoming rare in instruction set designing, since most successful designs of the last 40 years are nearly similar. Some designs which failed drastically, did so since their sponsoring companies failed commercially, not because the instruction sets were poor technically. Poor commercialization led their failure and not their faults or any bugs or that they were bad.

So, a well-designed open instruction set designed using well-established principles should attract long-term support too [4]. Just like other designs which are optimized only for exposition, the designers state that the RISC-V instruction set is for *practical real computers*.

RISC-V has multiple features which increases the speed of the computer. Some of them are as below:

1. RISC-V has a load and store architecture bit pattern; which simplifies the multiplexers in CPU.

2. Simplifies standard based floating point; which places most significant bits at a fixed location.
3. A Sign Extension is also a critical feature.

These features increase the speed of the computer and reduce the cost and power usage drastically, making RISC-V quite amazing.

The instruction set is variable-width and extensible, making it possible for adding more bits later if required. As of 2016, the 128-bit ISA remains undefined, since there is little practical experience with such large memory systems. However, RISC-V also supports the designers' academic uses. The simplicity of the integer subset permits basic student exercises. The integer subset is a simple ISA enabling software to control research machines. The variable-length ISA enables extensions for both student exercises and research. The separated privileged instruction set permits research in operating system support, without redesigning compilers.

HISTORY

University of California at Berkeley in California is a hub for latest research work. Prof. Krste Asanović, a Professor of Computer Science Division in EECS Department of UC-Berkeley, found many uses for an open-source computer system. There in the year 2010, he decided to develop and publish one "short, three-month project over the summer". He planned was to help both academic and industrial users. Prof David Patterson also helped and contributed to the cause at Berkeley. Patterson originally identified the properties of RISC at Berkeley and RISC-V is one of his long series of cooperative RISC research projects. Early funding was from DARPA.

Multiple organizations supported the RISC-V Foundation. Some of them are as below –

AMD, Andes Technology, BAE Systems, Hewlett Packard Enterprise, Google, IBM, GreenWaves Technology, Microsemi, Huawei, Micron, IIT Madras, Microsoft, NVIDIA, Oracle, Western Digital, SiFive, ICT, NXP, Qualcomm, Rambus Cryptography Research, Mellanox Technology, Cortus and many more !!!!!

The term "RISC" dates from about 1980. Before this, there was some knowledge that simpler computers could be effective, but the design principles were not widely described. Simple, effective computers have always been of academic interest.

Academicians made the RISC instruction set DLX for the first edition of *Computer Architecture: A Quantitative Approach* in the year 1990. David

Patterson was an author, and later assisted RISC-V. However, DLX was only for educational use [5].

ARM CPUs, Version 2, had a public-domain instruction set, and it is still supported by GCC, a popular free software compiler. Three open-source cores exist for this ISA, but they have not been manufactured. Open RISC is an open-source ISA based on DLX, with associated RISC designs. It is fully supported with GCC and Linux implementations. However, it has few commercial implementations.

GOALS OF RISC-V

The goals in significance for developing RISC-V includes:

- To make a totally open-sourced ISA which is freely available for both academic as well as industrial usage.
- To develop an ISA which can be used and is suitable for direct hardware implementation and not only just for using it during simulations and other translations.
- Just as stated above, an ISA which is good for multiple styles and avoids over architecture of microarchitecture or for implementation technologies.
- An ISA which supports huge wide range of extensive user-level extensions and variations.
- An ISA which is divided into a small base integer ISA, which can be used by itself too as a base for other customizable accelerators and other optional standard extensions if desired.
- An ISA which can both work for 32-bit and 64-bit address space variants for applications, OS kernels, and hardware implementations
- The computing community required a completely open ISA, which is freely available to both academia and for industrial purpose.
- Hypervisor has always been the most important part of any system, so making a virtualized ISA, making the work of hypervisor is also desirable.
- To develop support for the revised 2008 IEEE-754 floating-point standard, since it was getting older down the time lane.

SPECIFICATIONS AND DESIGN

RISC-V contains multiple parts within it, for publishing the specification, such as ISA Bases and their extensions, Register Sets, Memory Access, Subroutine calls and jumps and many more. A computer design may add further extensions: Integer multiplication and division ("M"), Atomic instructions ("A") for handling real time concurrency, IEEE Floating Points ("F") with Double-precision ("D") and Quad-precision ("Q") options. Also, there is an optional "compact" ("C") extension to reduce code size. Many RISC-V computers might add this ISA to reduce power consumption, code size, and memory usage.

Future plans are also in view to support hypervisors, virtualization, bit-manipulation ("B"), decimal floating-point ("L"), transactional memory ("T") and many more. Below is an overview with all extensions that are finalized (frozen) and currently in development.

We will be discussing each of them one by one as we move ahead in the study of RISC-V.

1. ISA BASES & EXTENSIONS:

There are small differences in the same extension for different register sizes. If the architecture (32/64 bit) is not important it can be left out and described as "RVI" or "RVIMAFD". The ISA Base and its extensions are developed in a collective effort between industry, research community and educational institutions. Extensions marked as "frozen" are not expected to change in any way except clarifications and improvements in its documentation. We are going to describe the version 2.0 of the RV 321 base instruction set here. In this we have 31 general purpose registers x1-

x31, which hold the integer values. Register x0 is fixed to the constant 0. There is no subroutine return address on a call. Also, we have one additional user-visible register. The structure is like as made below-

xo/Zero
x1
x2
x3
x4
x5
x6
x7
x8
x9
x10
x11
x12
x13
x14
x15
x16
x17
x18
x19
x20
x21

x22
x23
x24
x25
x26
x27
x28
x29
x30
x31
pc for XLEN-1

Fig. 1. RISC-V User-Level Base Instruction Register state

2. BASE INSTRUCTION FORMATS –

In the base ISA, there are 4 formats for instruction, out of which all are 32 bits and must also be aligned in a 4-byte boundary in memory. An address which is misaligned is generated on an unconditional jump if the target address is not 4-byte aligned in the alignment. RISC-V keeps the source and destination registers at the same place in all formats to make the decoding much simpler. The only exception to this is the immediate encoding variants which are 5 bits and are used in the CSR instructions.

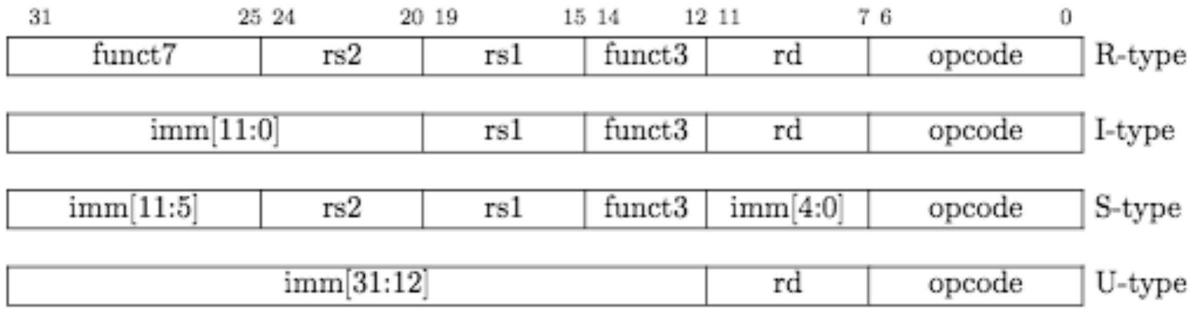


Fig. 2. RISC-V base instruction formats

3. IMMEDIATE ENCODING VARIANTS –

Furthermore, there are two more variants of the instruction formats Band J, based on the handling off the immediates [6]. The only difference between S and B formats is that the 12-bit immediate field is encoded branch offsets in a multiple of 2 in the B format.

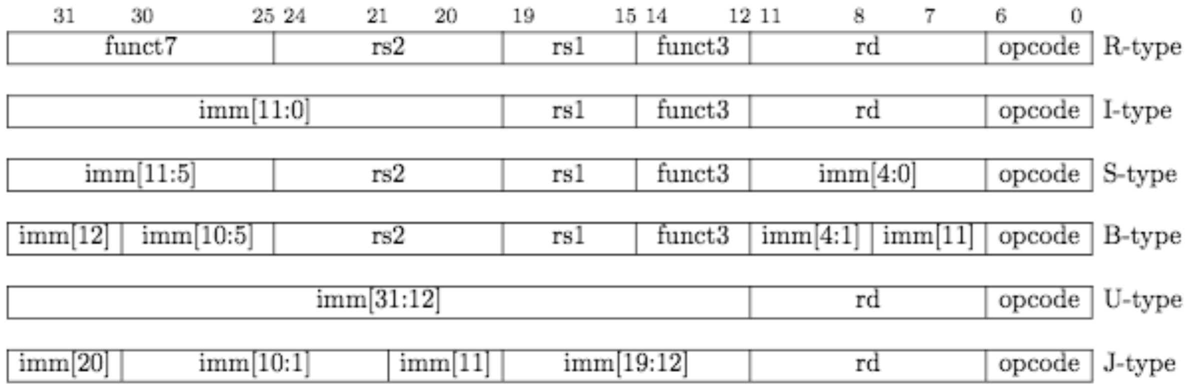
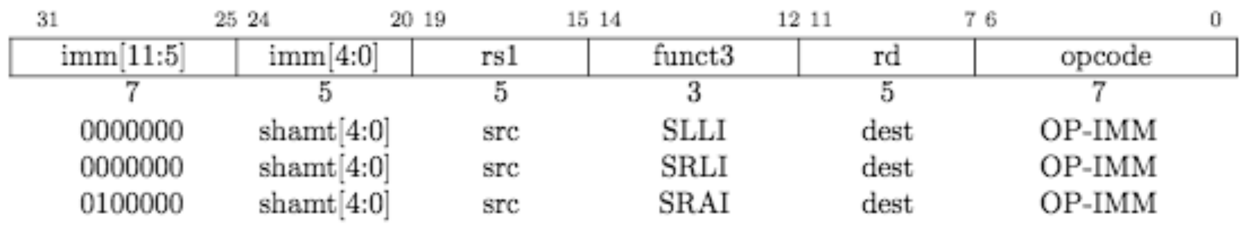


Fig. 3. RISC-V Base Instruction format for immediate variants

4. INTEGER REGISTER IMMEDIATE INSTRUCTIONS –

ADDI instruction adds the sign-extended 12-bit immediate to register r1. The arithmetic overflow is ignored and the result is simple the low XLEN bits of the result. ADDI rd, rs1, 0 is used for implementation of the MV rd, rs1 assembler pseudo-instruction.

SLTI (set less than immediate) puts the value 1 in register rd if register rs1 is less than the sign- extended immediate when both are treated as signed numbers, else 0 is written to rd. SLTIU is similar but compares the values as unsigned numbers.



31 – 25	24 – 20	19 – 15	14 – 12	11 – 7	6 – 0
imm[11:5]	imm[4:0]	rs1	funct3	rd	opcode
7	5	5	3	5	7
0000000	shamt[4:0]	src	SLLI	dest	OP-IMM
0000000	shamt[4:0]	src	SRLI	dest	OP-IMM
0100000	shamt[4:0]	src	SRAI	dest	OP-IMM

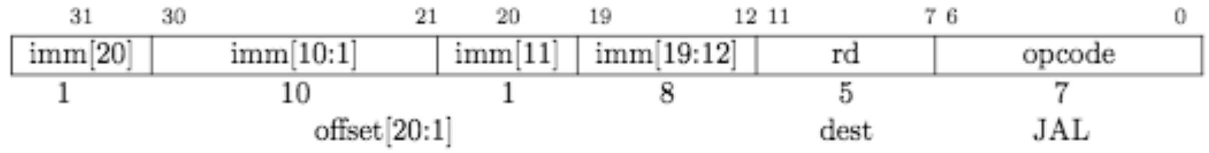
5. CONTROL AND STATUS REGISTER INSTRUCTIONS –

We have two types of the control transfer instructions, which includes –

1. Unconditional Jumps
2. Conditional Branches

We will discuss each of them one by one ahead –

1. UNCONDITIONAL JUMPS: JAL stores the address of the instruction following the jump (pc+4) into register rd. It jumps and link (JAL) instruction uses the J-type format, where the J-immediate encodes a signed offset in multiples of 2 bytes. The offset is sign-extended and added to the pc to form the jump target address. Jumps can therefore target a +/-1 MiB range.



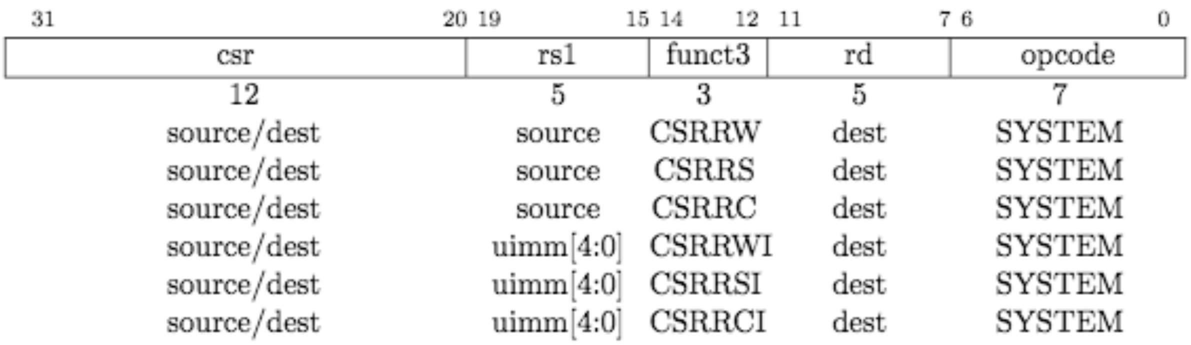
31	30 – 21	20	19 – 12	11 – 7	6 – 0
imm[20]	imm[10:1]	imm[11]	imm[19:12]	rd	opcode
1	10	1	8	5	7
	offset[20:1]			dest	JAL

6. CONTROL AND STATUS REGISTER INSTRUCTIONS –

SYSTEM instructions are used to access the system functionality that need privileged access ad are encoded with I-type instruction format. They can be divided into 2 major classes: one which are atomically read-modify and write control and status registers and all other potential privileged instructions.

A) CSR INSTRUCTIONS –

We can define the full set of CSR instructions in the following table –



31 – 20	19 – 15	14 – 12	11 – 7	6 – 0
csr	rs1	funct3	rd	opcode
12	5	3	5	7
source/dest	source	CSRRW	dest	SYSTEM
source/dest	source	CSRRS	dest	SYSTEM
source/dest	source	CSRRC	dest	SYSTEM
source/dest	uimm[4:0]	CSRRWI	dest	SYSTEM
source/dest	uimm[4:0]	CSRRSI	dest	SYSTEM
source/dest	uimm[4:0]	CSRRCI	dest	SYSTEM

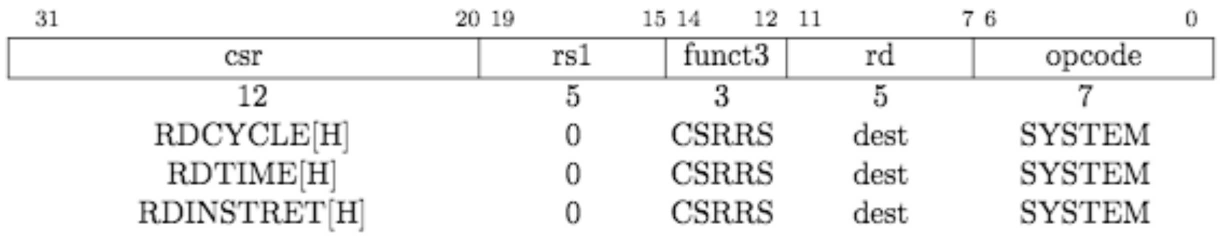
1. ***The CSRRW (Atomic Read/Write CSR)*** instruction atomically swaps values in the CSRs and integer registers. CSRRW reads the old value of the CSR, zero-extends the value to XLEN bits, then writes it to integer register rd.

2. The CSRRS (Atomic Read and Set Bits in CSR) instruction reads the value of the CSR, zero-extends the value to XLEN bits, and writes it to integer register rd.

3. The CSRRC (Atomic Read and Clear Bits in CSR) instruction reads the value of the CSR, zero-extends the value to XLEN bits, and writes it to integer register rd. The initial value in integer register rs1 is treated as a bit mask that species bit positions to be cleared in the CSR.

* Some CSRs, such as the instructions retired counter, may be modified as side-effects of instruction execution. In such cases, if a CSR access instruction reads a CSR, then it reads the value prior to the execution of the instruction. If a CSR access instruction writes a CSR, the update occurs after the execution of the instruction. A value written to ‘instret’ by one instruction will be the value read by the following instruction (i.e., the increment of ‘instret’ caused by the ‘rst’ instruction retiring happens before the write of the new value).

B) TIMERS AND COUNTERS –



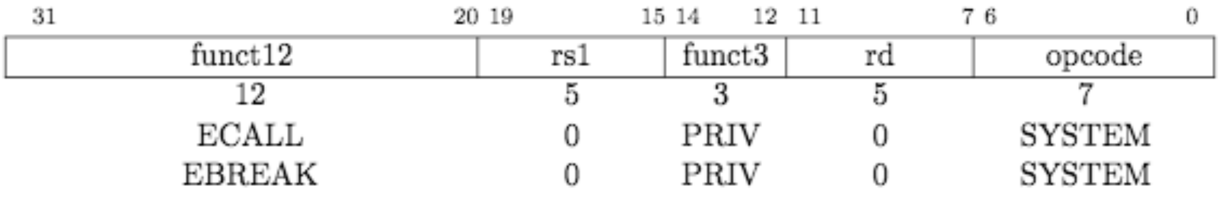
31 20	19 15	14 12	11 7	6 0
csr	rs1	funct3	rd	opcode
12	5	3	5	7
RDCYCLE[H]	0	CSRRS	dest	SYSTEM
RDTIME[H]	0	CSRRS	dest	SYSTEM
RDINSTRET[H]	0	CSRRS	dest	SYSTEM

1. The ***RDCYCLE*** pseudo-instruction reads the low XLEN bits of the cycle CSR which holds a count of the number of clock cycles executed by the processor core on which the hart is running from an arbitrary start time in the past. *RDCYCLE* is an RV32I-only instruction that reads bits 63-32 of the same cycle counter. The underlying 64-bit counter should never overflow in practice. The rate at which the cycle counter advances will depend on the implementation and operating environment.

2. The ***RDTIME*** pseudo-instruction reads the low XLEN bits of the time CSR, which counts wall-clock real time that has passed from an arbitrary start time in the past. *RDTIMEH* is an RV32I-only instruction that reads bits 63. The underlying 64-bit counter should never overflow in practice.

3. The ***RDINSTRET*** pseudo-instruction reads the low XLEN bits, which counts the number of instructions retired by this chart from some arbitrary start point in the past. *RDINSTRETH* is an RV32I-only instruction that reads bits 63-32 of the same instruction counter. The underlying 64-bit counter that should never overflow in practice.

7. ENVIRONMENT CALL AND BREAKPOINTS –



31 20	19 15	14 12	11 7	6 0
funct12	rs1	funct3	rd	opcode
12	5	3	5	7
ECALL	0	PRIV	0	SYSTEM
EBREAK	0	PRIV	0	SYSTEM

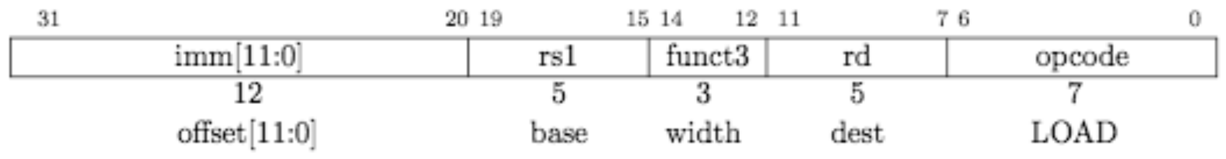
Figure 5.

1. The **ECALL** instruction is used to make a request to the supporting execution environment, which is usually an operating system. The ABI for the system will define how parameters for the environment request are passed, but usually these will be in defined locations in the integer register file.

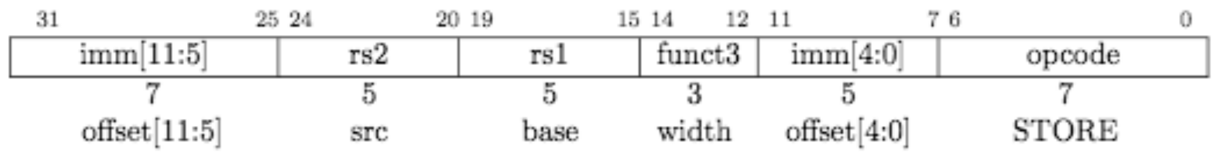
2. The **EBREAK** instruction is used by debuggers to cause control to be transferred back to a debugging environment.

8. LOAD AND STORE INSTRUCTIONS –

Load and store instructions transfer a value between the registers and memory. Loads are encoded in the I-type format and stores are S-type [7]. The effective byte address is obtained by adding register rs1 to the sign-extended 12-bit offset. Loads copy a value from memory to register rd. Stores copy the value in register rs2 to memory.



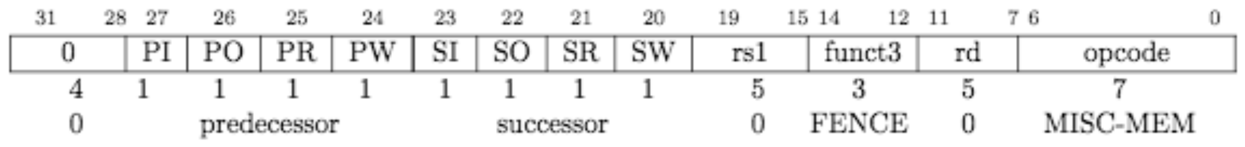
31 20	19 15	14 12	11 7	6 0
imm[11:0]	rs1	funct3	rd	opcode
12	5	3	5	7
offset[11:0]	base	width	dest	LOAD



31 25	24 20	19 15	14 12	11 7	6 0
imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
7	5	5	3	5	7
offset[11:5]	src	base	width	offset[4:0]	STORE

9. MEMORY MODEL –

In the base RISC-V ISA, each RISC-V hart observes its own memory operations as if they executed sequentially in program order. RISC-V has a relaxed memory model between harts, requiring an explicit FENCE instruction to guarantee ordering between memory operations from different RISC- V harts.



31 28	27	26	25	24	23	22	21	20	19 15	14 12	11 7	6 0
0	PI	PO	PR	PW	SI	SO	SR	SW	rs1	funct3	rd	opcode
4	1	1	1	1	1	1	1	1	5	3	5	7
0	predecessor				successor				0	FENCE	0	MISC-MEM

The FENCE instruction is used to order device I/O and memory accesses as viewed by other RISC- V harts and external devices or coprocessors. Any combination of device input (I), device output (O), memory reads (R), and memory writes (W) may be ordered with respect to any combination of the same. The execution environment will define what I/O operations are possible, which load and store instructions might be treated and ordered as device input and device output operations respectively rather than memory reads and writes. For example, memory-mapped I/O devices will typically be accessed with un-cached loads and stores that are ordered using the I and O bits rather than the R and W bits.

CURRENT STATE OF ART OF RISC-V

RISC-V being a high-quality, license free and royalty-free ISA, is experiencing a rapid uptake in the industry as well as academia. It is supported by the growing shared software ecosystem and is working ahead constantly. The qualities making RISC-V so popular these days include its simplicity, clear design, stable system which is designed by the community and for the community only.

Binary compatibility is a blessing and a curse for any successful architecture. It can simultaneously be your biggest strength (by its *stickiness*) and weakness (*cost*). The RISC-V team had the luxury of learning from mistakes made over the last 50 years of computer architecture development and have left all the heavy old baggage behind. Questionable features like delay slots and bloated instruction sets are being replaced by a clean, clear and an extensible architecture.

Benchmarks are disputable but by most methods the RISC-V performance should be good for most use cases. In the prolonged run, good enough and free is going to win forever. For applications, which needs more performance, there are always accelerator options. The configurability of the RISC-V will be a big advantage in IoT applications where energy consumption is the #1 and main concern.

Large companies have adopted for RISC-V for their deeply embedded controllers in their SoCs. NVIDIA is a leader among them, and has stated this in public, while other companies are doing it privately. Several service providers are also aware of the evolving strategies for leveraging the RISC-V values and importance [8]. Smaller propertiary ISA soft core IP companies are also switching to RISC-V standard to access larger marker.

Not only private companies, but also several governments are adopting RISC-V openly. India is among the selected ones. India has adopted RISC-V as their national ISA. US DARPA has also mandated RISC-V in the proposals for security concerns [9]. Israel's national innovation authority has generated a GenPro platform around RISC-V only. Also, multiple countries are also at multiple steps since every country wants to protect security of their information and the infrastructure and their semiconductor industry.

Along with these, several startups are also choosing RISC-V for their new products. Nearly most of them are in the starting setup stages, so effect would be visible in coming years only.

Academic research was one of the most targeted area for RISC-V makers. They made this system for commercial as well as for academic works. So now when they are on the floor, RISC-V is also being used extensively for academic research. Many workshops are being organized around the globe. 1st Workshop on Computer Architecture Research using RISC-V at 50th MICRO was organized in Boston, USA; and was the largest workshop, even bigger than the machine learning tutorial.

APPLICATIONS

RISC-V architecture is used across a wide range of platforms, ranging from cellular telephones and tablet computers to some of the world's fastest supercomputer, such as Sunway TaihuLight.

Most of the low end and mobile systems today depend upon RISC architecture. Some of the examples of the application of the RISC-V architecture are as below –

1. IBM's Power Architecture, which is found in PlayStation 3 and Xbox 360 gaming console contains the same architecture.
2. The MIPS line earlier and now the PlayStation, PlayStation2, Nintendo 64 and many more game consoles.
3. Hitachi's SuperH which is now developed and sold by Renesas as SH4 implements this architecture.
4. MIPS, by Silicon Graphics.
5. SPARC by Oracle and Fujitsu.
6. Hewlett Packard's PA-RISC which is also known as HP-PA.
7.Alpha, used in single board computers, workstations, servers and supercomputers, developed by Digital Equipment Corporation, Compaq and HP.
8. Rocket, a superscalar design, developed by UC-Berkeley.
9. Atmel AVR used in variety of products ranging from XBOX handheld controllers to BMW cars.

COMMERCIALIZATION OF RISC-V

RISC-V architecture is utilized for come across a wide range of platforms, ranging from cellular telephones and tablet computers to some of the world's fastest supercomputer, such as Sunway TaihuLight. It is quite popular in both industry as well as for academia.

Recently, Western Digital has announced to use RISC-V across its product stack, which will use storage and processing for present as well as in coming future stacks too. To develop the RISC-V ecosystem, Western Digital has already started engaging in partnerships and investments with various companies working on RISC-V projects from now only. A very recent example includes the investment of Western Union into Esperanto Technologies, which is a company led by experienced CPU architectures and designers.

Several governments are also adopting RISC-V openly, along with the private companies. India is among those countries. India has adopted RISC-V as their national ISA. US is also not behind. Its DARPA has also mandated RISC-V in the proposals for security concerns.

Along with these government organizations, startups are also choosing RISC-V for their new products. Most of them are in the starting setup stages, so the effect would be visible in coming years only.

POSSIBLE FUTURE SCOPES

For RISC-V we can have multiple future scopes since it is an open source project and new modifications are done every day. Some of the main possible future scopes can be discussed as below-

1. SINGLE STANDARD - Currently, there are dozens of different RISC-V ISAs which are not compatible to each other, so the foundation for RISC-V development should have a single standard, so that it becomes compatible with each and everyone in the market.

2. OPTIMIZATION OF IMPLEMENTED TECHNOLOGY - The optimization of the technology which is utilized for implementation can be done in coming future [10].

REFERENCES

1.https://www.codasip.com/2016/09/22/what-is-risc-vwhy-do-we-care-and-why-you-should-too/

2. https://riscv.org/risc-v-foundation/

3. https://www2.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-146.pdf

4. https://riscv.org/specifications/privileged-isa/

5. https://opencores.org/project/amber

6. https://riscv.org/specifications/

7. https://en.wikipedia.org/wiki/RISC-V14

8.http://www.adapteva.com/andreas-blog/why-i-will-be-using-the-risc-v-in-my-next-chip/

9.https://content.riscv.org/wp-content/uploads/2017/12/Tue0900-StateOfUnion-krste.pdf

10.https://en.wikipedia.org/wiki/Reduced_instruction_set_computer#Use_of_RISC_architectures