

Why RISC-V architecture has 32 registers?

Kunal Ghosh

Hi

Many such architectural questions has been solved in my launched online course on RISC-V ISA. Have you ever thought, why RISC-V instruction sets, or for that matter, most instruction sets today have 16 or 32 general purpose registers? Something to think about

Let's look at an example *load doubleword* instruction below, which loads data into x8 register from memory, whose base address is present in register x23 and offset is '16'. The way a computer sees this instruction is through a 32-bit binary pattern. (Details of the below command is covered in the course)

Applicat Registers	io	n	bi	in	a	y	i	nt	er	fa	ce	(AB	31))	(LE	N	- 64	4bi	t fo	r R	V6-	4							
Let's see a	n	e>	ar	n	ple	10																													
(01110010	00	01	11	11	01	00	10	001	.01	00:	110	00(010	11(000	010	000)11	0	010	11	11	01	10	011	11	11	bin							
				de	stii	nat	tio	n n	egi:	ster	'ro	r																							
load	do	ubl	ew	on	ď					offs	et '	im	m'																						
		1	1	1	ld				x8,	1	6(x	23	3)																						
									sou	irce	re	gis	ter	'rs1																					
	_																																	-	
					_	m	me	dia	ate							rs.	1		1	fun	ct3	3			rd			_		0	co	de			
	31	30	29	2	8	n	26	25	24	23	22	21	20	19	18	17	16	15	5	14 1		12	11	10		8		5	.5						

Now focus on 'rd' and 'rs1' which are called as *destination register* and *source register* respectively. Each of them has been strictly allotted 5-bits. Which means, to represent x8 for 'rd', the field will contain the pattern '01000'

Another example, shown below, adds contents of x8 with data present in x24 and stores results back in x8. Here, we have 2 *source registers* x24 and x8, and one *destination register* x8



Again, focus at 'rd', 'rs1' and 'rs2'. They are all 5 bits, which means, to represent x24 in rs1, that field will contain '11000'

One is good, two is better, three is best...So let's consider third example below

In the example, the contents or 'x8' are stored in memory whose base address is present in register 'rs2' and offset is '8' (

Application bin Registers –	AB	1)					XLE	4 -	641	oit f	for F	RV6	4										
Let's see an exam	ple																						
(011100100001111)	0100100	1010	0011	000	101	100	0010	00	110	001	011	11	011	01	111	11	bin						
store doubleword	ld add sd	x8, x8, x8,	16 x2 8((x23 4, x8 x23))																		
data register	rs2' offset 'i immed	sou imm' liate	irce	regis	ter	'rs1'	rsi			fu	inct3			n	d				op	co	de		
31 30 29	18 27 26 2	5 24	23	22 21	20	19	18 17	16	15	14	13 1	2	11 1	0	5 8	7	6	3	4	3	2	1	0
			rs2				rs1				funct3			'n	rd		Ē		00	ico	de		
fu	nct7		r	\$2			rs]			1.10	ncta								ųμ				
11 30 29	nct7 28 27 26 2	5 24	23 :	2 21 21	20	19	rs] 18 17	16	15	14	13 1	2	11 1	0		Ż	6	\$	4	1	ž:	1	0
101 June 101	nct7 # 27 26 2 hte[11:5]	5 24	23 2 75	s2 22 21 52	20	19	rs] 18 17 rs]	16	15	14 fu	1) 1 inct3	2	11 1 mm	o i	ate[7 4:0]	6	.5	4 op	3	2 de	1	0

Once more, focus on 'rs2' and 'rs1'. They are again 5 bits. Practically, to keep design simple, all registers in a RISC-V architecture is represented by 5-bit binary pattern.



Now the calculation is easy. 5-bits to represent registers, which means total number of registers is $2^5 = 32$ registers

"Simplicity favors regularity and good design demands good compromises" – Just as Prof. David Patterson mentions in his book called "Computer Organization and Design"

The good part has already been per-launched and best part is yet to come.

The reason we launched this course, is because, we were asked to come with a design, which we can take through the entire RTL-synthesis-PNR-STA tool chain and after a lot of thought, we believed that understanding a sophisticated RISC-V CPU core is the best way to learn how to write specs, how to implement and how to PNR...all in one...

Stay tuned, more exciting things yet to come...

Till then...happy learning...