

PQR5 Assembler

Instruction Manual

Jan 2024



Contents

- 1. General Info 3
- 2. Registers..... 4
- 3. Instructions 5
- 4. Binary/Hex file format 10

1. General Info

pqr5asm is an assembler which translates RISC-V assembly to binary/hex code.

Compliance	RV32I (User-Level ISA v2.2) - 37 base instructions + pseudo/custom instructions
Input	Assembly program (sample.s)
Output	Binary/Hex code in ASCII text, .bin formats (sample.bin, sample_bin.txt, sample_hex.txt)
Syntax Rules	<ol style="list-style-type: none"> 1) One instruction per line, semicolon at the end of statement is optional. 2) Base address of the program (address to which instructions should be stored in the instruction memory) can be defined using assembler directive in the first line of program. Binary file generated with this base address for programming. For eg: <code>.ORIGIN 0x400</code> If not provided, overridden to <code>0x00000000</code> 3) Supports <space>, <comma>, and <linebreak> as delimiters for eg: <code>LUI x5 255 <linebreak> or LUI x5, 255 <linebreak></code> 4) Use '#' for inline/newline comments for eg: <code>LUI x5, 255 # This is a sample comment</code> 5) Supports 32-bit signed/unsigned integer, 0x hex literals for immediate. For eg: <code>255, 0xFF, -255</code> Immediate supports parenthesis format for ALU-I instructions: <code>addi x1, x0, 2 <=> addi x1, 2(x0)</code> Immediate gets truncated to 20-bit or 12-bit based on instruction. 6) Register ABI names are case-insensitive. 7) Supports labels for jump/branch instructions: <ul style="list-style-type: none"> ✓ Label is recommended to be of max. 8 ASCII characters ✓ Label should be stand-alone in new line for eg: <code>FIBONACC:</code> <code> mvi x1, 1</code> ✓ Label is case-sensitive. ✓ Pre-processor will assign pc-relative address to label.
Invoking Assembler	<code>pqr5asm.py <assembly source file path></code>

2. Registers

Following registers are supported by the ISA and Assembler ABI.

Register Name	ABI Name	Description
x0	x0	Hard-wired Zero
x1	ra	Return Address
x2	sp	Stack Pointer
x3	gp	Global Pointer
x4	tp	Thread Pointer
x5-x7	t0-t2	Temporary Registers
x8	s0/fp	Saved Register/Frame Pointer
x9	s1	Saved Register
x10-x11	a0-a1	Function Arg/Return Val Registers
x12-x17	a2-a7	Function Arg Registers
x18-x27	s2-s11	Saved Registers
x28-x31	t3-t6	Temporary Registers

Table 2.1: Registers with ABI acronyms

3. Instructions

S No	Instruction	Syntax	Description
1.	LUI	LUI rd, imm	Load Upper Immediate Builds 32-bit constants. Loads 20-bit imm[19:0] into the upper 20-bit of rd. Loads the lower 12-bit of rd with zeroes. eg: LUI x1, 0xFFFF
2.	AUIPC	AUIPC rd, imm	Add Upper Immediate PC Builds PC-relative addresses. Forms 32-bit offset from 20-bit imm[19:0] by loading into the upper 20-bit of rd, and loading the lower 12-bit with zeroes. Adds this offset to the PC, then places the result in rd.
Control Transfer Instructions			
3.	JAL	JAL rd, label OR JAL rd, imm	Jump And Link Unconditional jump. Used to call subroutines. Stores next instruction address, pc+4 in rd for return from subroutine. 20-bit imm[19:0] encodes signed offset in multiples of 2 bytes, and is added to the current pc to get the target address. target address = $pc + 32'(\text{signed}'(\{\text{offset}[20:1], 1'b0\}))$ The unconditional jump range = ± 1 MB.
4.	JALR	JALR rd, rs1, offset	Jump And Link Register Unconditional Indirect jump. Used to call subroutines. Stores next instruction address, pc+4 in rd for return from subroutine. 12-bit imm[11:0] encodes signed offset, and is added to rs1, and clear 0th bit of result to get the target address. target address = $\{(rs1 + 32'(\text{signed}'(\text{offset}))) [31:1], 1'b0\}$ The unconditional jump range = ± 2 kB. (-2048 to +2047)
5.	BEQ	BEQ rs1, rs2, label OR BEQ rs1, rs2, imm	Branch Equal Takes the branch if rs1 == rs2 12-bit imm[11:0] encodes signed offset in multiples of 2 bytes, and is added to the current pc to get the target address. target address =

			$pc + 32'(\text{signed}'(\{\text{offset}[12:1], 1'b0\}))$ The conditional branch range = ± 4 KB.
6.	BNE	BNE rs1, rs2, label OR BNE rs1, rs2, imm	Branch Not Equal Takes the branch if $rs1 \neq rs2$
7.	BLT	BLT rs1, rs2, label OR BLT rs1, rs2, imm	Branch Less Than Takes the branch if $\text{signed}'(rs1) < \text{signed}'(rs2)$
8.	BGE	BGE rs1, rs2, label OR BGE rs1, rs2, imm	Branch Greater Than or Equal Takes the branch if $\text{signed}'(rs1) \geq \text{signed}'(rs2)$
9.	BLTU	BLTU rs1, rs2, label OR BLTU rs1, rs2, imm	Branch Less Than Unsigned Takes the branch if $rs1 < rs2$
10.	BGEU	BGEU rs1, rs2, label OR BGEU rs1, rs2, imm	Branch Greater Than or Equal Unsigned Takes the branch if $rs1 \geq rs2$
Load Store Instructions			
11.	LB	LB rd, rs1, offset	Load Byte Loads 8-bit data from memory, sign-extends to 32-bit, put into rd. load address = $32'rs1 + 32'(\text{signed}'(\text{offset}))$ // expected to be 8-bit aligned
12.	LH	LH rd, rs1, offset	Load Half-word Loads 16-bit data from memory, sign-extends to 32-bit, put into rd.
13.	LW	LW rd, rs1, offset	Load Word Loads 32-bit data from memory, put into rd.
14.	LBU	LBU rd, rs1, offset	Load Byte Unsigned Loads 8-bit data from memory, zero-extends to 32-bit, put into rd.
15.	LHU	LHU rd, rs1, offset	Load Half-word Unsigned Loads 16-bit data from memory, sign-extends to 32-bit, put into rd.
16.	SB	SB rs2, rs1, offset	Store Byte Stores lower 8-bit of rs2 in memory. store address =

			32'rs1 + 32'(signed'(offset)) // expected to be 8-bit aligned
17.	SH	SH rs2, rs1, offset	Store Half-word Stores lower 16-bit of rs2 in memory.
18.	SW	SW rs2, rs1, offset	Store Word Stores rs2 in memory.
Integer Computation Instructions (ALU-I)			
19.	ADDI	ADDI rd, rs1, imm	Add Immediate rd = rs1 + 32'(signed'(imm)) // overflow ignored
20.	SLTI	SLTI rd, rs1, imm	Set Less Than Immediate rd = 1, if signed'(rs1) < 32'(signed'(imm)), else 0
21.	SLTIU	SLTIU rd, rs1, imm	Set Less Than Immediate Unsigned rd = 1, if rs1 < 32'(signed'(imm)), else 0
22.	XORI	XORI rd, rs1, imm	XOR Immediate rd = rs1 XOR 32'(signed'(imm))
23.	ORI	ORI rd, rs1, imm	OR Immediate rd = rs1 OR 32'(signed'(imm))
24.	ANDI	ANDI rd, rs1, imm	AND Immediate rd = rs1 AND 32'(signed'(imm))
25.	SLLI	SLLI rd, rs1, shamnt	Logical Left Shift Immediate rd = rs1 << shamnt[4:0]
26.	SRLI	SRLI rd, rs1, shamnt	Logical Right Shift Immediate rd = rs1 >> shamnt[4:0]
27.	SRAI	SRAI rd, rs1, shmant	Arithmetic Right Shift Immediate rd = signed'(rs1) >>> shamnt[4:0]
Integer Computation Instructions (ALU-R)			
28.	ADD	ADD rd, rs1, rs2	Add rd = rs1 + rs2 // overflow ignored
29.	SUB	SUB rd, rs1, rs2	Subtract rd = rs1 - rs2 // underflow ignored
30.	SLL	SLL rd, rs1, rs2	Logical Left Shift rd = rs1 << rs2[4:0]
31.	SLT	SLT rd, rs1, rs2	Set Less Than rd = 1,

			if signed'(rs1) < signed'(rs2), else 0
32.	SLTU	SLTIU rd, rs1, rs2	Set Less Than Unsigned rd = 1, if rs1 < rs2, else 0
33.	XOR	XOR rd, rs1, rs2	XOR rd = rs1 XOR rs2
34.	SRL	SRL rd, rs1, rs2	Logical Right Shift rd = rs1 >> rs2[4:0]
35.	SRA	SRA rd, rs1, rs2	Arithmetic Right Shift rd = signed'(rs1) >>> rs2[4:0]
36.	OR	OR rd, rs1, rs2	OR rd = rs1 OR rs2
37.	AND	AND rd, rs1, rs2	AND rd = rs1 AND rs2
Pseudo/Custom Instructions			
38.	MV	MV rd, rs1 = ADDI rd, rs1, 0	Move rd = rs1
39.	MVI	MVI rd, imm = ADDI rd, x0, imm	Move Immediate (12-bit immediate) rd = imm
40.	NOP	NOP = ADDI x0, x0, 0	No Operation
41.	J	J label = JAL x0, label	Plain Jump Jump to label
42.	NOT	NOT rd, rs1 = XORI rd, rs1, -1	NOT rd = NOT rs1
43.	INV	INV rd = XORI rd, rd, -1	Invert rd = NOT rd
44.	SEQZ	SEQZ rd, rs1 = SLTIU rd, rs1, 1	Set Equal to Zero rd = 1, if rs1 == 0, else 0
45.	SNEZ	SNEZ rd, rs2 = SLTU rd, x0, rs2	Set Not Equal to Zero rd = 1, if rs1 != 0, else 0
46.	BEQZ	BEQZ rs1, label = BEQ rs1, x0, label	Branch Equal to Zero Jump to label, if rs1 == 0, else 0
47.	BNEZ	BNEZ rs1, label = BNE rs1, x0, label	Branch Not Equal to Zero

			Jump to label, if rs1 != 0, else 0
48.	LI	LI rd, imm ** = LUI rd, U + ADDI rd, L	Load Immediate (32-bit immediate) rd = imm
49.	LA	LA rd, label ** = LUI rd, U + ADDI rd, L	Load Address (32-bit immediate) rd = address(label)
50.	JR	JR rs1 = JALR x0, rs1, 0	Jump Register Address Jump to address = rs1

Table 3.1: Instructions supported by Assembler

** *U* and *L* are Upper 20-bit and Lower 12-bit values derived from *imm*

Conventions used:

rd = Destination register

rs1 = Source register-1

rs1 = Source register-2

imm = 12/20-bit immediate

4. Binary/Hex file format

Input to the assembler is assembly program which is in **.s** file format. On successful compilation, three files are generated:

1. Binary code file in **.bin** format, which may be decoded and directly loaded to CPU for execution.
2. Binary/Hex code files in ASCII **.txt** format. This is human readable.

The **.bin** file contains the instructions to be executed as sequence of bytes in binary format. The following example shows how a **.bin** file is encoded and formatted by the assembler. Big Endian format is used in the binary file.

```
<0xF0><0xF0><0xF0><0xF0>      # Pre-amble marks the beginning
<0x00><0x00><0x00><0x28>      # Program size = (no. of instructions, N x 4) bytes
<baB3><baB2><baB1><baB0>      # Base address of the program byte[3] to [0]
<inB3><inB2><inB1><inB0>      # Instruction-1 byte[3] to [0]
<inB3><inB2><inB1><inB0>      # Instruction-2 byte[3] to [0]
....
....
<inB3><inB2><inB1><inB0>      # Instruction-N byte[3] to [0]
<0xE0><0xE0><0xE0><0xE0>      # Post-amble marks the end
```

PQR5 Assembler

An open-source RISC-V Assembler for RV32I ISA

Developer : Mitu Raj

Vendor : Chipmunk Logic™, chip@chipmunklogic.com

Website : chipmunklogic.com

