

STAM Center
SECURE, TRUSTED, AND ASSURED MICROELECTRONICS

ASU Engineering
1984 A Future School of
Arizona State University

RISC-V Introduction

STAM Center Staff

1

STAM Center
SECURE, TRUSTED, AND ASSURED MICROELECTRONICS

ASU Engineering
1984 A Future School of
Arizona State University

Brief Overview of the RISC-V ISA

- A new, open, free ISA from Berkeley
- Several variants
 - RV32, RV64, RV128 – Different data widths
 - 'I' – Base Integer instructions
 - 'M' – Multiply and Divide
 - 'A' – Atomic memory instructions
 - 'F' and 'D' – Single and Double precision floating point
 - 'V' – Vector extension
 - And many other modular extensions
- We will focus on the RV32I the base 32-bit variant

2

STAM Center
SECURE, TRUSTED, AND ASSURED MICROELECTRONICS

ASU Engineering
1984 A Future School of
Arizona State University

RV32I Register State

- 32 general purpose registers (GPR)
 - x0, x1, ..., x31
 - 32-bit wide integer registers
 - x0 is hard-wired to zero

RV128

RV64

RV32

x0	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
----	----	----	----	----	----	----	----	----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

RV32I

RV128

RV64

RV32

x0	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
----	----	----	----	----	----	----	----	----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

RV32I

3

STAM Center
SECURE, TRUSTED, AND ASSURED MICROELECTRONICS

ASU Engineering
Arizona State University

RV32I Register Conventions

NAME	Register Number	Usage
zero	x0	Hardwired to the constant value 0
ra	x1	Return address for subroutine calls
sp	x2	Stack pointer (stack grows downwards)
gp	x3	Global pointer (e.g. to static data area)
tp	x4	Thread pointer
t0 – t2	x5 – x7	More temporary registers (caller saves)
s0/fp	x8	Frame pointer (to local variables on stack)
s1	x9	Saved register (callee saves)
a0 – a1	x10 – x11	Arguments (parameters) to subroutines / return values
a2 – a7	x12 – x17	Arguments (parameters) to subroutines
s2 – s11	x18 – x27	Saved registers (callee saves)
t3 – t6	x28 – x31	Temporary registers (caller saves)

4

STAM Center
SECURE, TRUSTED, AND ASSURED MICROELECTRONICS

ASU Engineering
Arizona State University

RV32I State

- 32 general purpose registers (GPR)
 - x0, x1, ..., x31
 - 32-bit wide integer registers
 - x0 is hard-wired to zero
- Program counter (PC)
 - 32-bit
- CSR (Control and Status Registers)
 - User-mode
 - cycle (clock cycles) // read only
 - instret (instruction counts) // read only
 - Machine-mode
 - hwid (hardware thread ID) // read only
 - miepic, mcause etc. used for exception handling
 - Custom
 - mtchost (output to host) // write only – custom extension

5

STAM Center
SECURE, TRUSTED, AND ASSURED MICROELECTRONICS

ASU Engineering
Arizona State University

Base Instruction Formats

- The base RISC-V ISA has six instruction formats

31	30	25	24	21	20	19	15	14	12	11	8	7	6	0		
funct7		rs2			rs1			funct3		rd		opcode			R-type	
imm(11:0)				rs1			funct3		rd		opcode			I-type		
imm(11:5)				rs2			rs1			funct3		imm(4:0)		opcode	S-type	
imm(12)		imm(10:5)			rs2			rs1			funct3		imm(4:3)	imm(11)	opcode	B-type
imm(31:12)									rd		opcode			U-type		
imm(20)		imm(10:3)			imm(11)			imm(19:12)		rd		opcode			J-type	

6

Base Instruction Formats

- The base RISC-V ISA has six instruction formats
- The R,I,S & U are most common

31	30	25	24	21	20	19	15	14	12	11	8	7	6	0	
funct7				rs2		rs1		funct3		rd		opcode		R-type	
imm(11:0)				rs1		funct3		rd		opcode		I-type			
imm(11:5)				rs2		rs1		funct3		imm(4:0)		opcode		S-type	
imm(12)		imm(10:5)		rs2		rs1		funct3		imm(4:3)		imm(11)		B-type	
imm(31:12)						rd		opcode		U-type					
imm(20)		imm(10:3)		imm(11)		imm(19:12)		rd		opcode		J-type			

7

RISC-V ISA Features

31	30	25	24	21	20	19	15	14	12	11	8	7	6	0	
funct7				rs2		rs1		funct3		rd		opcode		R-type	
imm(11:0)				rs1		funct3		rd		opcode		I-type			
imm(11:5)				rs2		rs1		funct3		imm(4:0)		opcode		S-type	
imm(12)		imm(10:5)		rs2		rs1		funct3		imm(4:3)		imm(11)		B-type	
imm(31:12)						rd		opcode		U-type					
imm(20)		imm(10:3)		imm(11)		imm(19:12)		rd		opcode		J-type			

8

RISC-V ISA Features

- 7-bit opcode to decode divides instructions into similar types
- Examples: Different opcodes for loads, stores, and R-type instructions

31	30	25	24	21	20	19	15	14	12	11	8	7	6	0	
funct7				rs2		rs1		funct3		rd		opcode		R-type	
imm(11:0)				rs1		funct3		rd		opcode		I-type			
imm(11:5)				rs2		rs1		funct3		imm(4:0)		opcode		S-type	
imm(12)		imm(10:5)		rs2		rs1		funct3		imm(4:3)		imm(11)		B-type	
imm(31:12)						rd		opcode		U-type					
imm(20)		imm(10:3)		imm(11)		imm(19:12)		rd		opcode		J-type			

9

STAM Center Secure, Trusted, and Assured Microelectronics **ASU Engineering** Arizona State University

RISC-V ISA Features

- Funct3 field further decodes type of instruction
- Example: For load instructions, funct3 equals $\log_2(N)$, where N is number of bytes to load

31	30	25	24	21	20	19	14	12	11	8	7	6	0	
funct7				rs2		rs1	funct3		rd	opcode				R-type
imm(11:0)				rs2		rs1	funct3		rd	opcode				I-type
imm(11:5)				rs2		rs1	funct3		imm(4:0)		opcode			S-type
imm(12)		imm(10:5)		rs2		rs1	funct3		imm(4:3)		imm(1:1)		opcode	B-type
imm(31:12)								rd		opcode				U-type
imm(20)		imm(10:3)		imm(1:1)		imm(8:12)		rd		opcode				J-type

10

STAM Center Secure, Trusted, and Assured Microelectronics **ASU Engineering** Arizona State University

RISC-V ISA Features

- Funct7 field further decodes type of instruction
 - Only used in R-Type instructions
- Provides support for large number of arithmetic operations

31	30	25	24	21	20	19	15	14	12	11	8	7	6	0
funct7				rs2		rs1	funct3		rd	opcode				R-type
imm(11:0)				rs2		rs1	funct3		rd	opcode				I-type
imm(11:5)				rs2		rs1	funct3		imm(4:0)		opcode			S-type
imm(12)		imm(10:5)		rs2		rs1	funct3		imm(4:3)		imm(1:1)		opcode	B-type
imm(31:12)								rd		opcode				U-type
imm(20)		imm(10:3)		imm(1:1)		imm(8:12)		rd		opcode				J-type

11

STAM Center Secure, Trusted, and Assured Microelectronics **ASU Engineering** Arizona State University

RISC-V ISA Features

- Register operands in same fields for all instructions
- Supports efficient decoding of instructions

31	30	25	24	21	20	19	15	14	12	11	8	7	6	0
funct7				rs2		rs1	funct3		rd	opcode				R-type
imm(11:0)				rs2		rs1	funct3		rd	opcode				I-type
imm(11:5)				rs2		rs1	funct3		imm(4:0)		opcode			S-type
imm(12)		imm(10:5)		rs2		rs1	funct3		imm(4:3)		imm(1:1)		opcode	B-type
imm(31:12)								rd		opcode				U-type
imm(20)		imm(10:3)		imm(1:1)		imm(8:12)		rd		opcode				J-type

12

STAM Center Secure, Trusted, and Assured Microelectronics **ASU Engineering** Arizona State University

RISC-V ISA Features

- Note the different immediate value encodings in I,S,B,U,J Types
- Different ways to store data in the instruction
 - Loads store address offsets, B & J store address targets

31	30	25	24	21	20	19	15	14	12	11	8	7	6	0																																			
funct7					rs2					rs1					funct3					rd					opcode																								
imm[11:0]															rd					funct3					rd					opcode																			
imm[11:5]															rs2					rs1					funct3					imm[4:0]					opcode														
imm[12]															imm[10:5]					rs2					rs1					funct3					imm[4:3]					imm[11]					opcode				
imm[31:12]															rd					funct3					rd					opcode																			
imm[20]															imm[10:3]					imm[11]					imm[19:12]					rd					opcode														

13

STAM Center Secure, Trusted, and Assured Microelectronics **ASU Engineering** Arizona State University

Computational Instructions

- Register-Register instructions (R-type)
 - Read two register values, perform a computation, and store the result back to the register file

7	5	5	3	5	7
funct7	rs2	rs1	funct3	rd	opcode

- Example Instructions (Not complete)

Instruction	Operation
ADD	$rd = rs1 + rs2$
XOR	$rd = rs1 \wedge rs2$
SLL	$rd = rs1 \ll rs2$
SLT	$rd = rs1 < rs2$

14

STAM Center Secure, Trusted, and Assured Microelectronics **ASU Engineering** Arizona State University

Computational Instructions

- Register-immediate instructions (I-type)
 - Read one register, perform an operation with rs1 data and immediate value, write result back to register file

12	5	3	5	7
imm[11:0]	rs1	funct3	rd	opcode

- Example Instructions (Not complete)

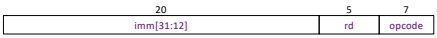
Instruction	Operation
ADDI	$rd = rs1 + imm$
ORI	$rd = rs1 \mid imm$
SRLI	$rd = rs1 \gg (imm \% 32)$
SLTI	$rd = rs1 < imm$

15

STAM Center SECURE, TRUSTED, AND ASSURED MICROELECTRONICS **ASU Engineering** 1984 Future School of Arizona State University

Computational Instructions

- Register-immediate instructions (U-type)
 - Load an immediate value into the register file
 - May or may not perform operation with immediate data before loading



- Example Instructions (Not complete)

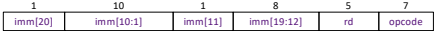
Instruction	Operation
LUI	rd = imm
AUIPC	rd = PC + imm

16

STAM Center SECURE, TRUSTED, AND ASSURED MICROELECTRONICS **ASU Engineering** 1984 Future School of Arizona State University

Control Instructions

- Unconditional jump and link (J-type)
 - JAL: Write PC+4 into the register file, set the PC=PC+imm



- Jump ±1MB range
- Unconditional jump via register and link (I-type)
 - JALR: Write PC+4 into register file, set PC= (rs1 + I-imm) & ~0x01

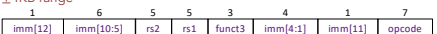


17

STAM Center SECURE, TRUSTED, AND ASSURED MICROELECTRONICS **ASU Engineering** 1984 Future School of Arizona State University

Control Instructions

- Conditional branches (B-type)
 - Read two registers, perform comparison, jump to PC+imm if comparison is true, else execute PC+4 next
 - Jump ±4KB range



- Example Instructions (Not complete)

Instruction	Operation
BLT	next PC = (rs1 < rs2) ? pc + imm : pc + 4
BEQ	next PC = (rs1 == rs2) ? pc + imm : pc + 4

18

Load & Store Instructions

- Load (l-type)

12

5

3

5

7

imm[11:0]

rs1

funct3

rd

opcode

 - opcode = LOAD: $rd \leftarrow mem[rs1 + l-imm]$
 - $l-imm = signExtend(inst[31:20])$
 - funct3 = LW/LB/LBU/LH/LHU
- Store (S-type)

7

5

5

3

5

7

imm[11:5]

rs2

rs1

funct3

imm[4:0]

opcode

 - opcode = STORE: $mem[rs1 + S-imm] \leftarrow rs2$
 - $S-imm = signExtend((inst[31:25], inst[11:7]))$
 - funct3 = SW/SB/SH

19

Assembly Programming

- Function call
 1. Caller places parameters in a place where the procedure can access them
 2. Transfer control to the procedure
 3. Acquire storage resources required by the procedure
 4. Execute the statements in the procedure
 5. Called function places the result in a place where the caller can access it
 6. Return control to the statement next to the procedure call

20

Assembly Programming

- Function call
 - Argument Passing
 - Arguments to a function passed through a0-a7
 - Functions with more than 8 arguments
 - First eight arguments are put in a0-a7
 - Remaining arguments are put on stack by the caller
 - Return Values
 - Return values from a function passed through a0-a1
 - Functions with more than 2 return values

21

Assembly Programming

- Function call
 - Argument Passing
 - Arguments to a function passed through a0-a7
 - Functions with more than 8 arguments
 - Return Values
 - Return values from a function passed through a0-a1
 - Functions with more than 2 return values
 - First two return values put in a0-a1
 - Remaining return values put on stack by the function
 - The remaining return values are popped from the stack by the caller

22

If then Else Assembly

```

if (a0 < 0) then
{
    t0 = 0 - a0;
    t1 = t1 + 1;
}
else
{
    t0 = a0;
    t2 = t2 + 1;
}

```

```

bgez a0, else
# if (a0 is > or = zero) branch to
else
sub t0, zero, a0
# t0 gets the negative of a0
addi t1, t1, 1
# increment t1 by 1
j next
# branch around the else code
else:
ori t0, a0, 0
# t0 gets a copy of a0
addi t2, t2, 1
# increment t2 by 1
next:

```

23

While Do Assembly

```

t0 = 1
while (a1 < a2)
do
{
    t1 = mem[a1];
    t2 = mem[a2];
    if (t1 != t2)
        go to break;
    a1 = a1 + 1;
    a2 = a2 - 1;
}
break: t0 = 0

```

```

li t0, 1 # Load t0 with the value 1
loop:
bgez a1, a2, done
# if (a1 >= a2) Branch to done
lw t1, 0(a1)
# Load a Byte: t1 = mem[a1 + 0]
lw t2, 0(a2)
# Load a Byte: t2 = mem[a2 + 0]
bne t1, t2, break
# if (t1 != t2) Branch to break
addi a1, a1, 1 # a1 = a1 + 1
addi a2, a2, -1 # a2 = a2 - 1
b loop # Branch to loop
break:
li t0, 0 # Load t0 with the value 0
done:

```

24

STAM Center
SECURE, TRUSTED, AND ASSURED MICROELECTRONICS

ASU Engineering
Arizona State University

For loop Assembly

```

a0 = 0;
For ( t0 =10;
      t0 > 0;
      t0 = t0 -1)
do {
  a0 = a0 + t0
}
```

```

li a0, 0    # a0 = 0
li t0, 10   # Initialize loop counter to 10
loop:
  add a0, a0, t0
  addi t0, t0, -1
  # Decrement loop counter
  bgtz t0, loop
  # if (t0 >0) Branch to loop
```

25

STAM Center
SECURE, TRUSTED, AND ASSURED MICROELECTRONICS

ASU Engineering
Arizona State University

Assembly

- Case study
 - Assume that A is an array of 64 words and the compiler has associated registers a1 and a2 with the variables x and y. Also assume that the starting address, or base address is contained in register a0. Determine the RISC-V instructions associated with the following C statement:
 - `x = y + A[4];` // adds 4th element in array A to y and stores result in x

26

STAM Center
SECURE, TRUSTED, AND ASSURED MICROELECTRONICS

ASU Engineering
Arizona State University

Assembly

- Case study
 - Assume that A is an array of 64 words and the compiler has associated registers a1 and a2 with the variables x and y. Also assume that the starting address, or base address is contained in register a0.
 - `x = y + A[4];` // adds 4th element in array A to y and stores result in x
 - Solution:
 - `lw t0, 16(a0)` # a0 contains the base address of array and # 16 is the offset address of the 4th element
 - `add a1, a2, t0` # performs addition

27
