RISC-V ISA

LUI and AUIPC

Sparsh Mittal

LUI and AUIPC are typically not used by human programmers!

They are used by assemblers to implement complex operations

.

Name (Field Size)	Field 7 bits	5 bits	5 bits	3 bits	5 bits	7 bits	Comments
R-type	funct7	rs2	rs1	funct3	rd	opcode	Arithmetic instruction format
I-type	immediate[11:0]		rs1	funct3	rd	opcode	Loads & immediate arithmetic
S-type	immed[11:5]	rs2	rs1	funct3	immed[4:0]	opcode	Stores
SB-type	immed[12,10:5]	rs2	rs1	funct3	immed[4:1,11]	opcode	Conditional branch format
UJ-type	immediate[20,10:1,11,19:12]				rd	opcode	Unconditional jump format
U-type	immediate[31:12]				rd	opcode	Upper immediate format

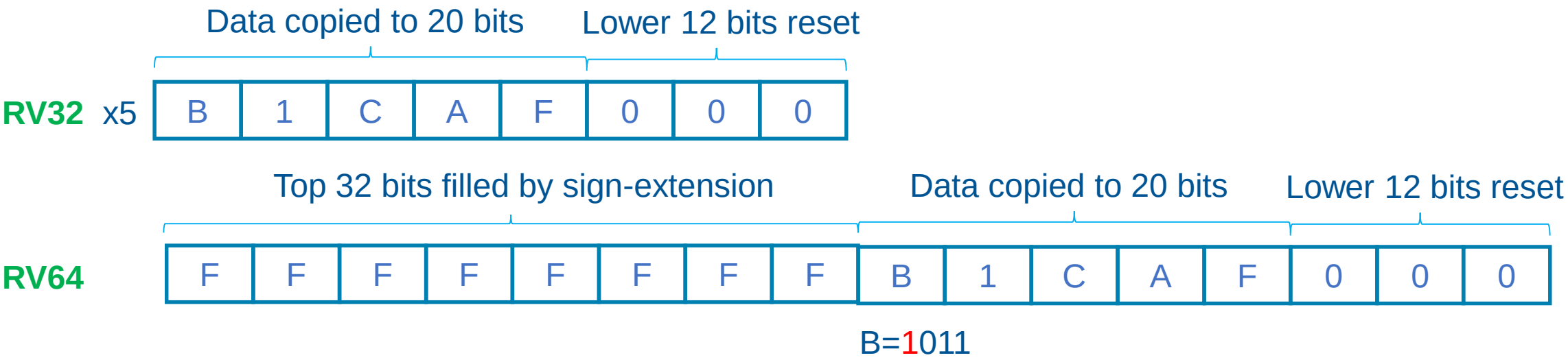
In upper-immediate format, the immediate is 20-bit wide.

LUI: Load upper immediate

In RV32: Copies the immediate value to upper 20 bits of rd. Lower 12 bits are reset to zero.

In RV64: same as above; Bits 32-63 filled by sign-extending 31st bit.

lui x5, 0xB1CAF



LUI: Load upper immediate

lui x5, 0x11000 # x5 will become 0x11000000 in both RV32 and RV64

lui x6, 0x80011 #x6 will become 0x80011000 in RV32 and
0xffffffff80011000 in RV64

Useful for loading relatively large immediate values.

LUI is usually followed by addi (to load lower 12 bits). In this way, we can load all 32 bits

Example of lui

• Write a RISC-V assembly program to add 409932 + 409823.

```
lui t0 , 100              # t0 = 4096 * 100 = 409600
addi t0 , t0 , 332        # t0 = t0 + 332
lui t1 , 100              # t1 = 4096 * 100 = 409600
addi t1 , t1 , 223        # t1 = t1 + 223
add t2 , t0 , t1          # t2 = t0 + t1
```

Alternative code using “li”

```
li t0 , 409932 # t0 = 409932
li t1 , 409823 # t1 = 409823
add t2 , t0 , t1 # t2 = t0 + t1
```

AUIPC: Add Upper Immediate to PC

Creates a 32-bit offset by setting upper 20-bit to the immediate value and resetting lower 12 bits. Then, adds it to program counter (*pc*), and stores the result in rd

In RV64, 31st bit of offset is extended to bits 32 to 63

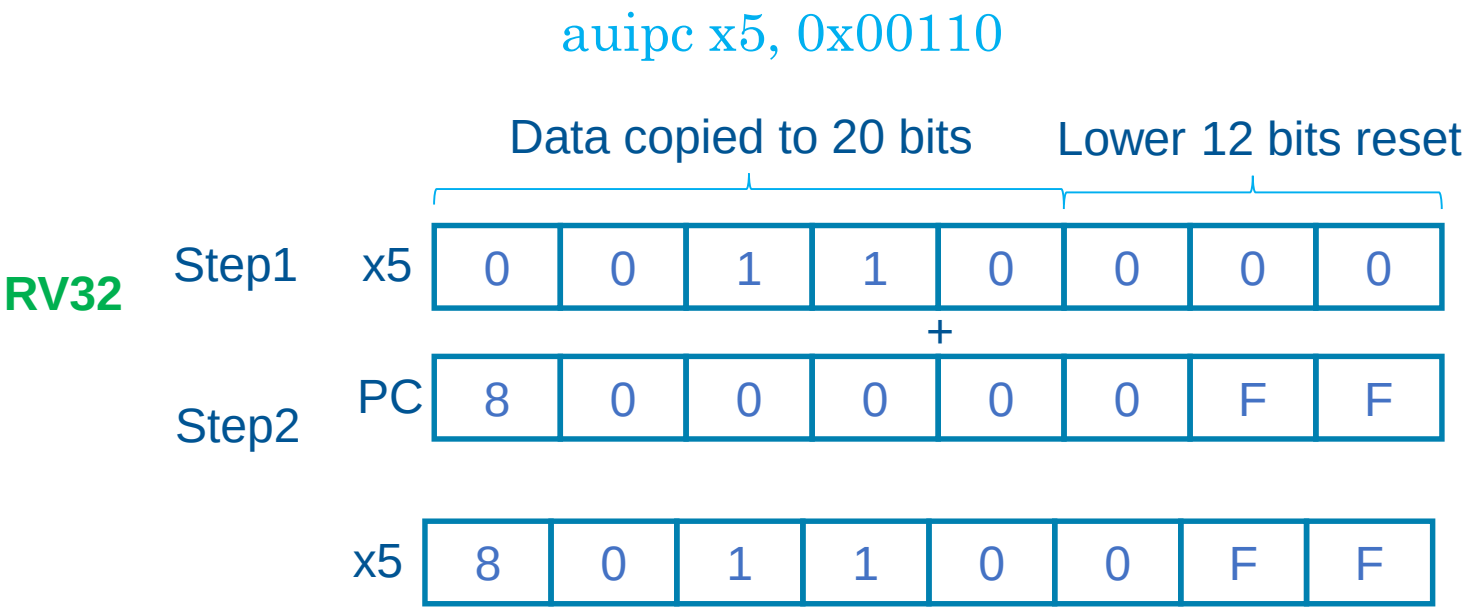
Useful for building PC-relative addresses

Auipc is followed by addi, then jalr to allow long jumps within any 32 bit address

AUIPC: Add Upper Immediate to PC

Example: Assuming *pc* is at 0x800000FF.

auipc x5, 0x00110



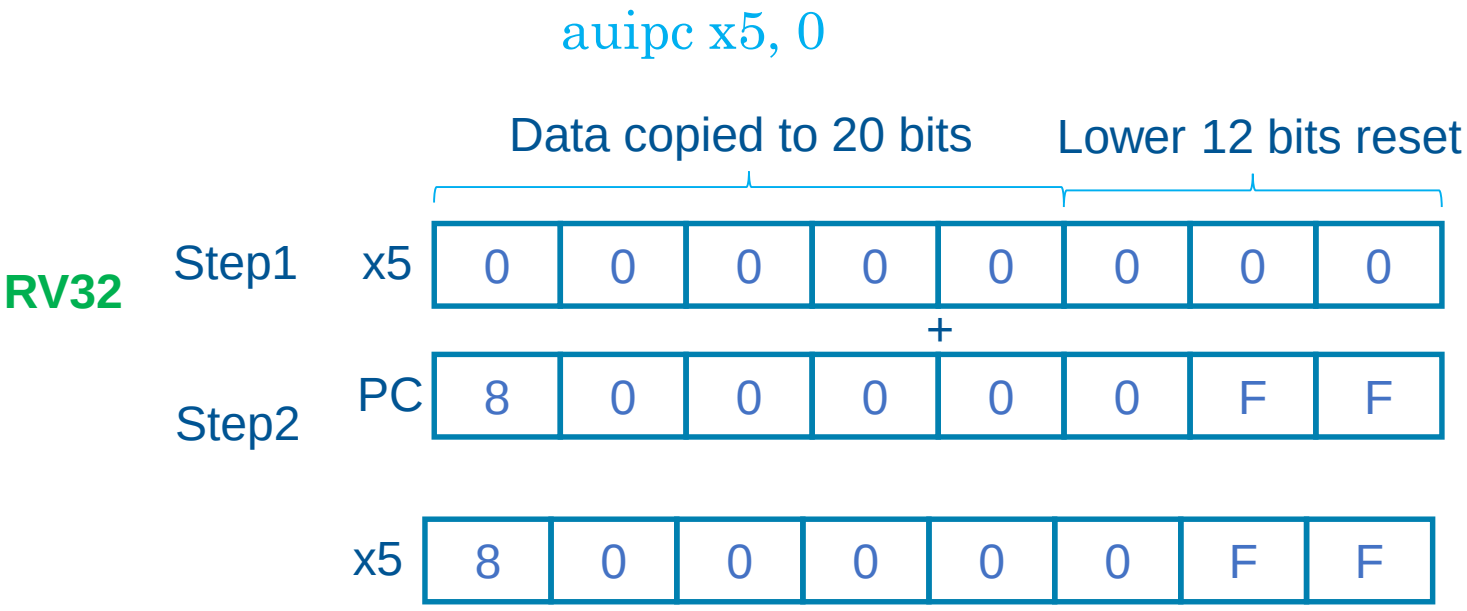
Step1: x5 is updated
Step2: x5+PC is stored in x5.

x5 will become 0x801100FF

Using AUIPC to obtain PC value

Example: Assuming *pc* is at 0x800000FF.

auipc x5, 0



In x86, reading PC value requires 1 store, 2 loads and 2 taken jumps!

x5 will get PC value

Arithmetic Operation

Mnemonic	Instruction	Type	Description
`ADD rd, rs1, rs2`	Add	R	`rd ← rs1 + rs2`
`SUB rd, rs1, rs2`	Subtract	R	`rd ← rs1 - rs2`
`ADDI rd, rs1, imm12`	Add immediate	I	`rd ← rs1 + imm12`
`SLT rd, rs1, rs2`	Set less than	R	`rd ← rs1 < rs2 ? 1 : 0`
`SLTI rd, rs1, imm12`	Set less than immediate	I	`rd ← rs1 < imm12 ? 1 : 0`
`SLTU rd, rs1, rs2`	Set less than unsigned	R	`rd ← rs1 < rs2 ? 1 : 0`
`SLTIU rd, rs1, imm12`	Set less than immediate unsigned	I	`rd ← rs1 < imm12 ? 1 : 0`
`LUI rd, imm20`	Load upper immediate	U	`rd ← imm20 << 12`
`AUIP rd, imm20`	Add upper immediate to PC	U	`rd ← PC + imm20 << 12`