Topic 4

Instruction Encoding

Representing Instructions

- Assembly instructions are translated into binary information
  - Called *machine code*
- RISC-V instructions are
  - Encoded as 32-bit words
  - Stored in 32-bit long memory locations
  - Small number of formats encode operation code (opcode), register numbers, …
  - **Regularity!**

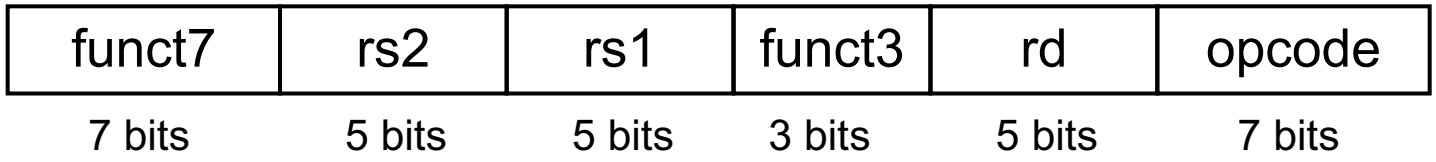
Representing Instructions

- Represent RISC-V instructions with 6 types (format)
  - R-type (Register)
  - I-type (Immediate)
  - S-type (Store)
  - U-type (Load upper immediate)
  - B-type (Branch), a.k.a. SB-type
  - J-type (Jump), a.k.a. UJ-type

Instruction Types

Type	Field					
	7 bits	5 bits	5 bits	3 bits	5 bits	7 bits
R-type	funct7	rs2	rs1	funct3	rd	opcode
I-type	immediate[11:0]		rs1	funct3	rd	opcode
S-type	immed[11:5]	rs2	rs1	funct3	immed[4:0]	opcode
B-type	immed[11,9:4]	rs2	rs1	funct3	immed[3:0,10]	opcode
U-type	immediate[19:0]				rd	opcode
J-type	immediate[19,9:0,10,18:11]				rd	opcode

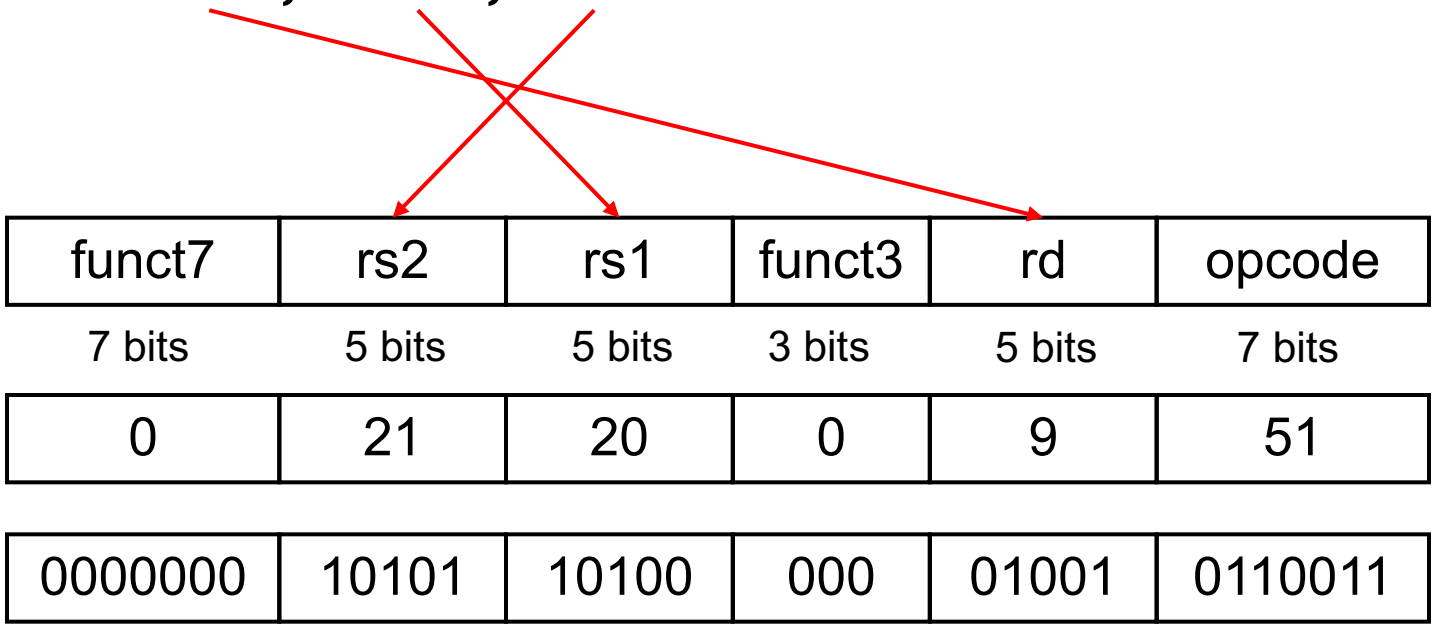
R-type



- Instruction fields
  - opcode: operation code
  - rd: destination register number
  - funct3: 3-bit function code (additional opcode)
  - rs1: the first source register number
  - rs2: the second source register number
  - funct7: 7-bit function code (additional opcode)

R-type Example

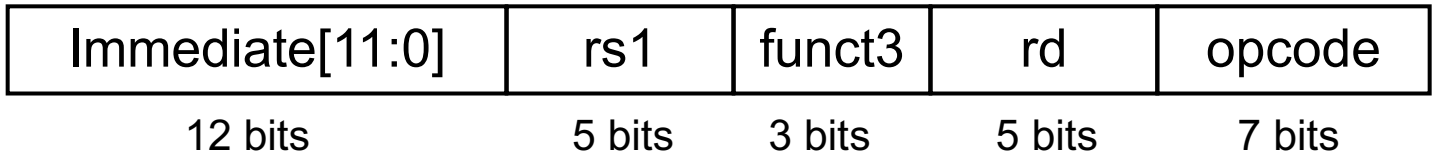
```
add x9,x20,x21
```



funct7	rs2	rs1	funct3	rd	opcode
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits
0	21	20	0	9	51
0000000	10101	10100	000	01001	0110011

$0000\ 0001\ 0101\ 1010\ 0000\ 0100\ 1011\ 0011_2 = 015A04B3_{16}$

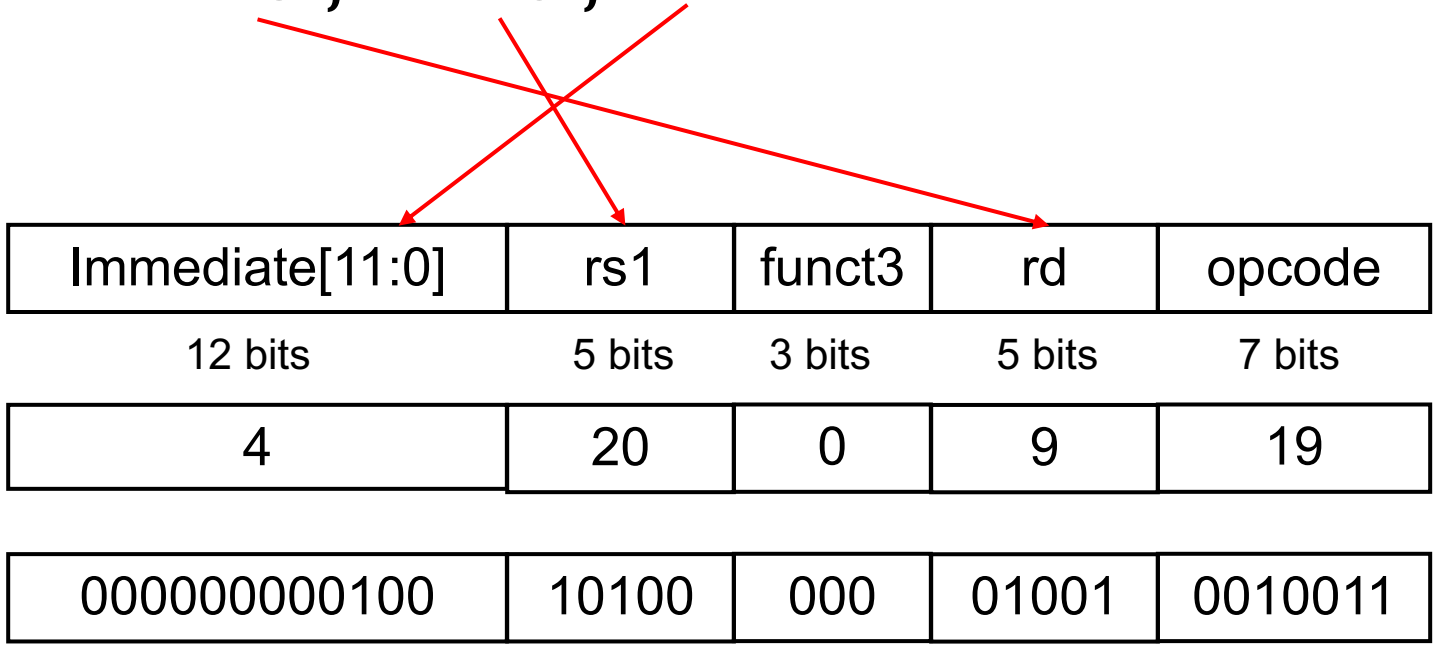
I-type



- Immediate arithmetic instructions
  - rs1: source register number
  - immediate: constant operand
    - 2's complement, sign extended
- load instructions
  - rs1: base address register number
  - immediate: offset added to base address
    - 2s-complement, sign extended
- *Design Principle 4:* Good design demands good compromises
  - Keep formats as similar as possible

I-type Example 1

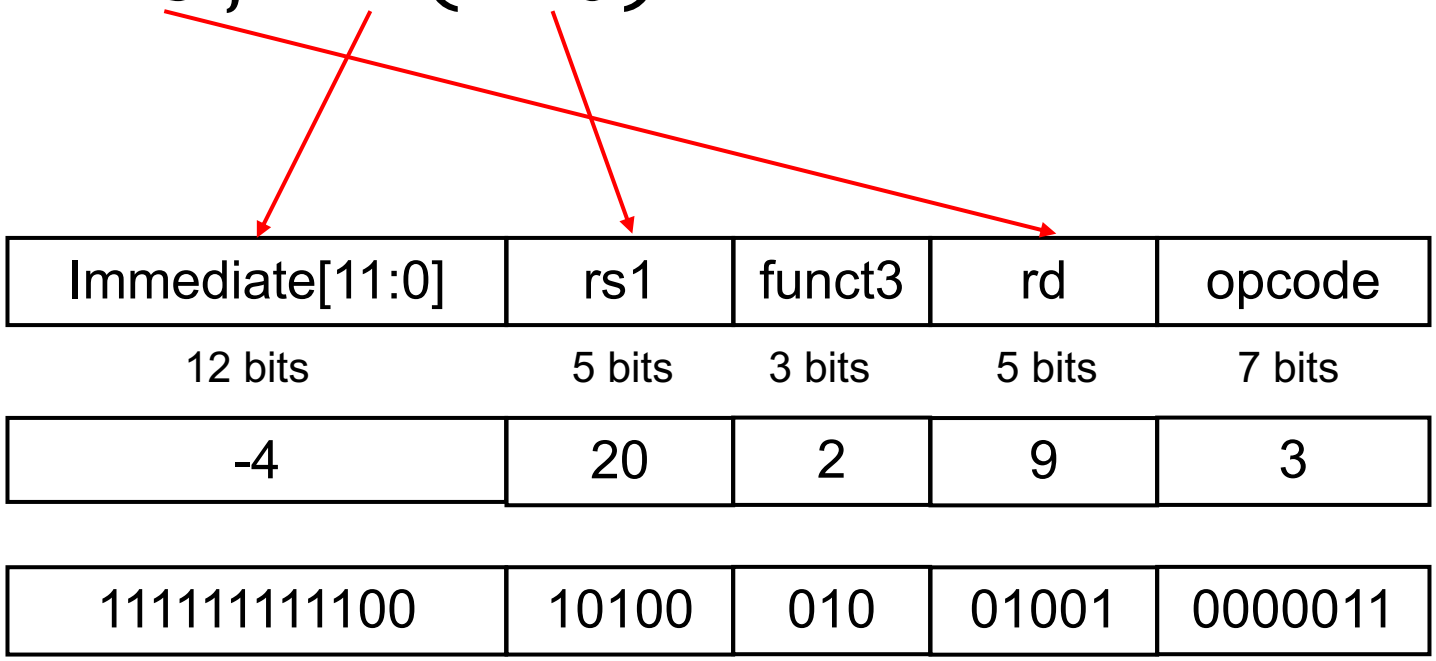
```
addi x9, x20, 4
```



$$000000000100101000000100100100011_2 = 004A0493_{16}$$

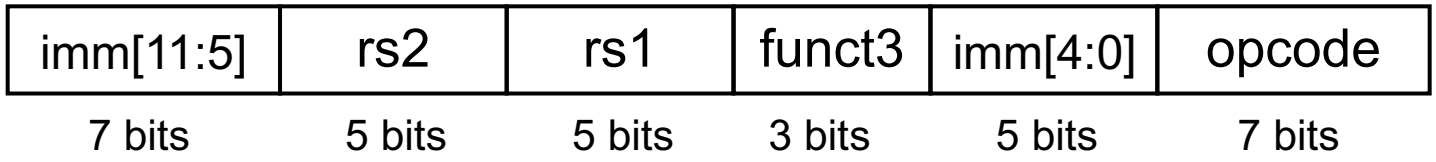
I-type Example 2

```
lw x9, -4(x20)
```



$$111111111100101000100100100000011_2 = FFCA2483_{16}$$

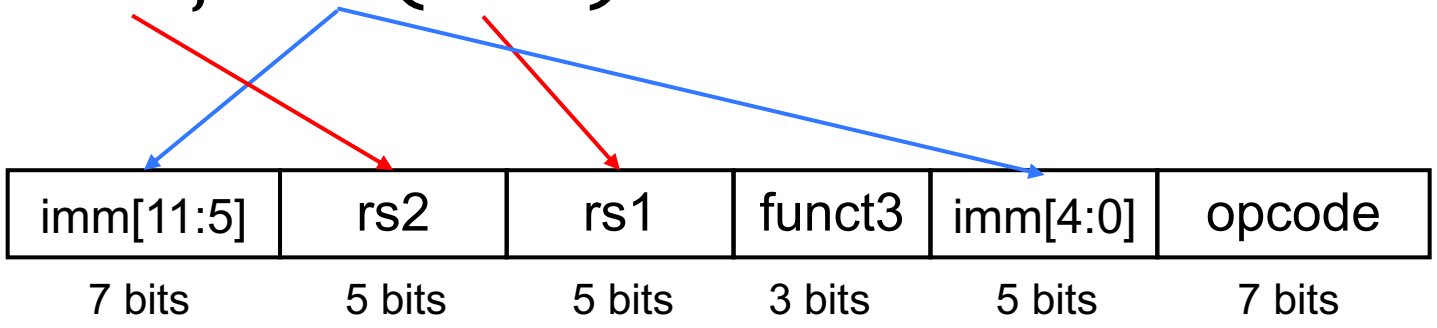
S-type



- For store instructions
- immediate: offset added to base address
- *Design Principle 4:* Good design demands good compromises
  - Keep formats as similar as possible
  - Split the 12-bit immediate so that rs1 and rs2 fields are always in the same positions in an instruction

S-type Example

```
sw x9, -4(x20)
```



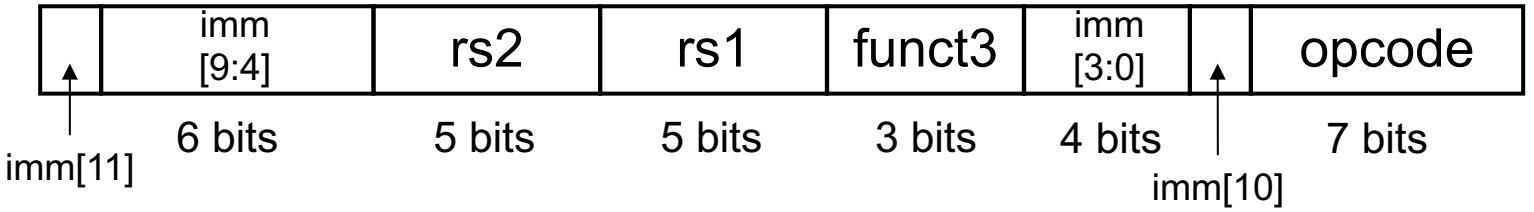
-4 = **1111111_11100**

1111111	01001	10100	010	**11100**	0100011
-1	9	20	2	**-4**	35

$$11111110100110100010111000100011_2 = FE9A2E23_{16}$$

B-type

- beq, bne, blt, bge, bltu, bgeu
- Most branch targets are near branch
  - Forward or backward
  - So 12-bit signed immediate is enough
- B type:



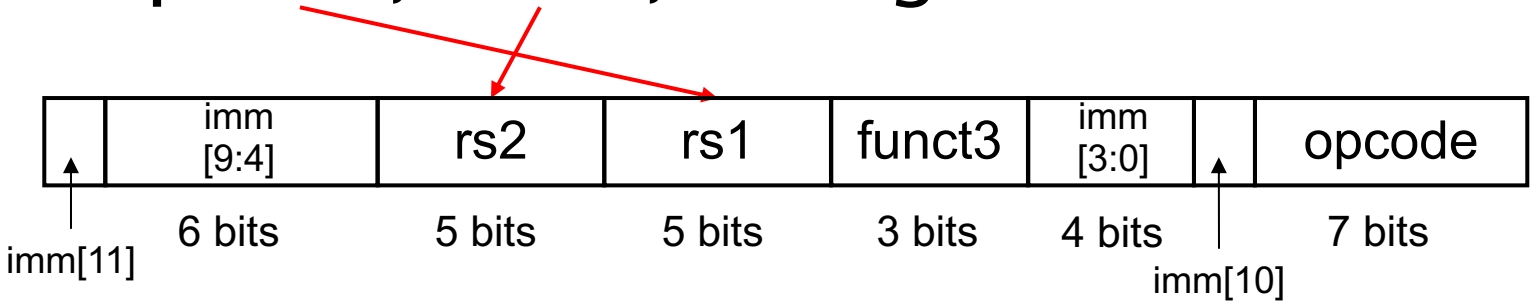
Branch Target address (Target PC)

= Current PC + immediate × 2

Note: This means the Target PC will always be an even number

B-type Example

```
beq x20, x21, Target
```



immediate = (Branch Target – Current PC) >> 1

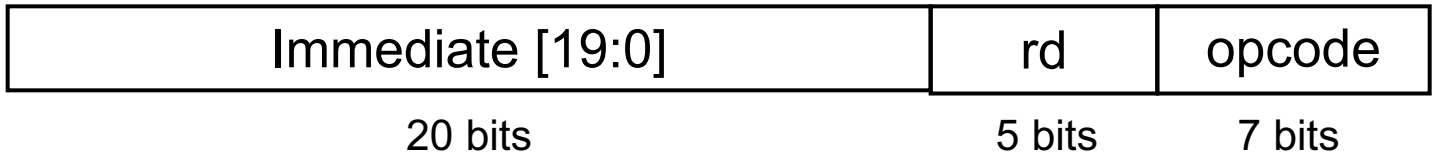
e.g. immediate = -4 = 1_1_111111_1100

1	111111	10101	10100	000	1100	1	1100011

-1	21	20	0	25	99

$$11111111010110100000110011100011_2 = FF5A0CE3_{16}$$

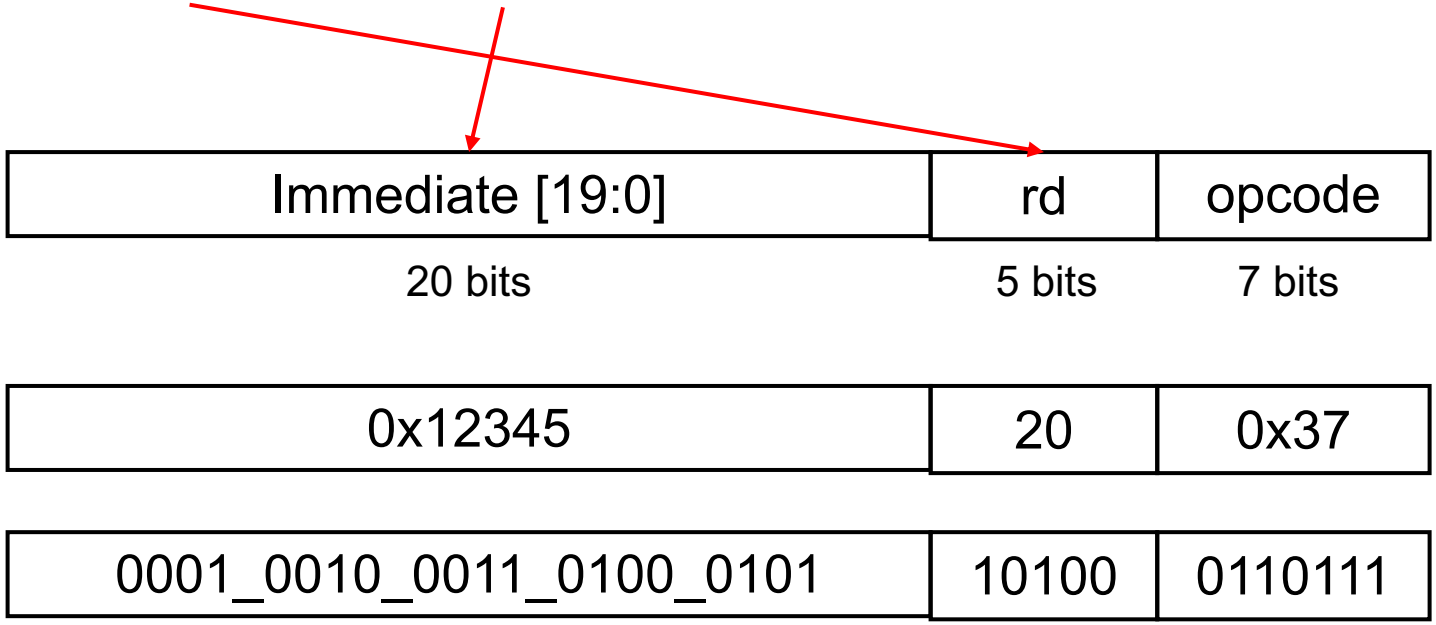
U-type



- For load upper immediate `lui` instruction (and `auipc`)

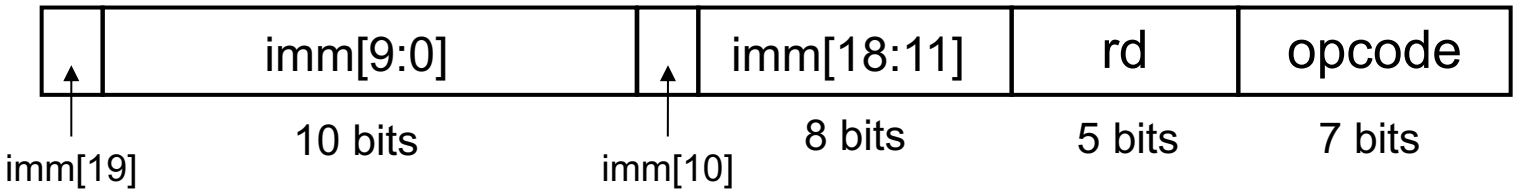
U-type Example

```
lui x20, 0x12345
```



$$00010010001101000101101000110111_2 = 12345A37_{16}$$

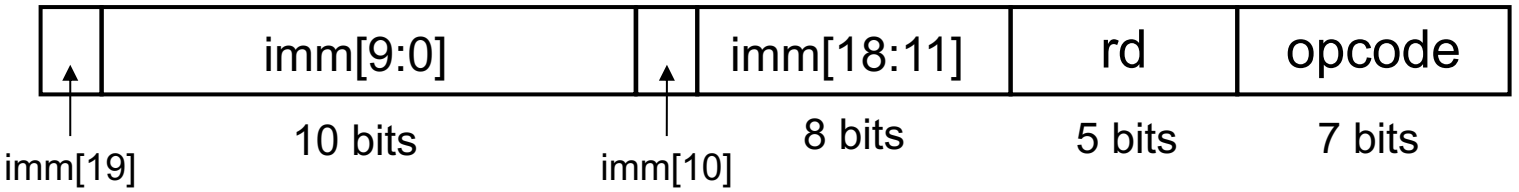
J-type



- For Jump and link (`jal`)
  - x1 <= PC + 4, x1: return address reg.
  - PC <= Target PC
    = Current PC + immediate × 2
- Jump instruction uses 20-bit immediate for larger jumping range

J-type Example

```
jal x1, Target
```



immediate = (Target PC – Current PC) >> 1

e.g. immediate = 14 = **0_00000000_0_0000001110**

0	**0000001110**	**0**	**00000000**	00001	1101111

0x01C00	1	111_{10}

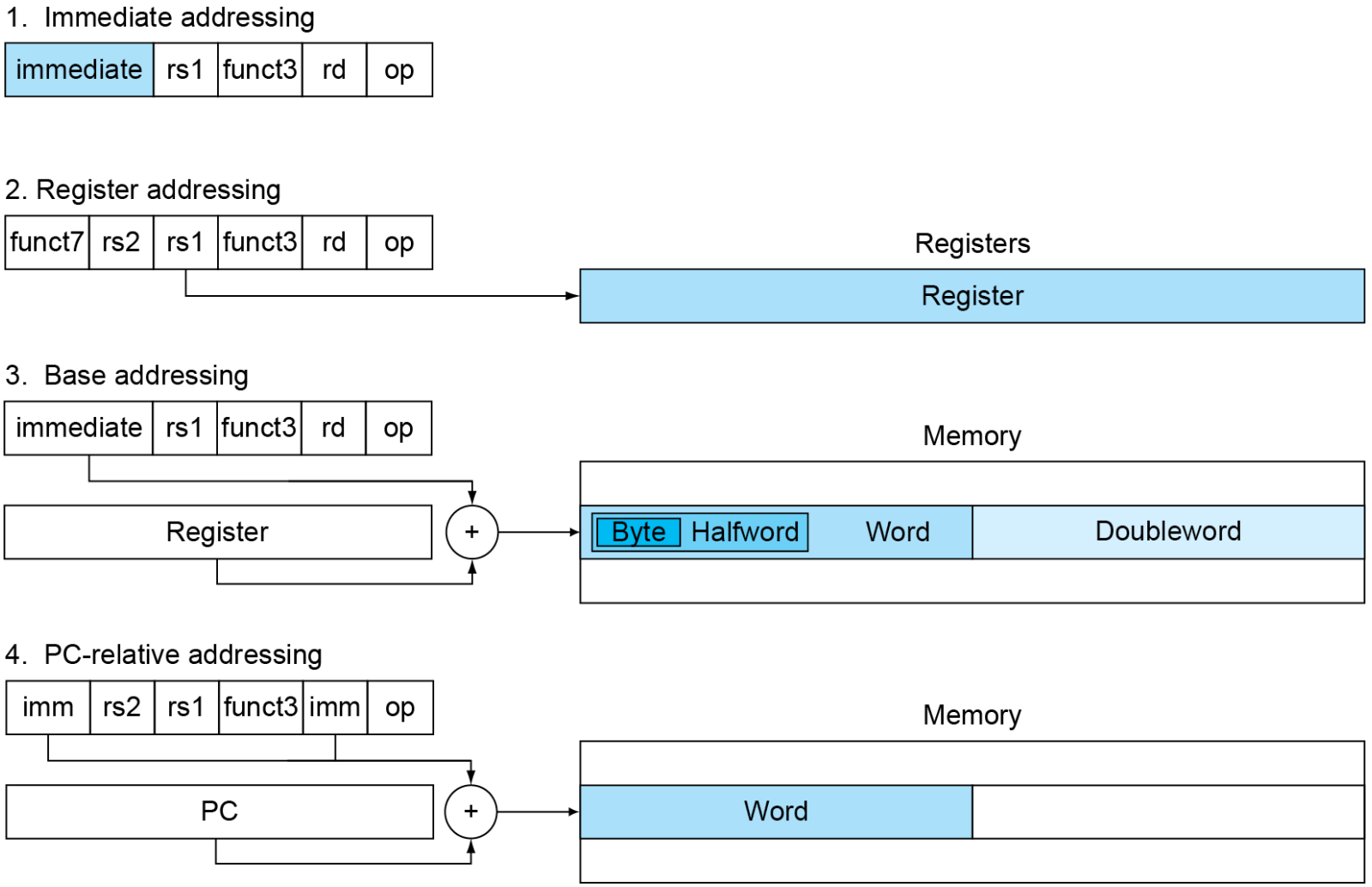
$00000001110000000000000011101111_2 = 01C000EF_{16}$

Performance Considerations

Type	Field					
	7 bits	5 bits	5 bits	3 bits	5 bits	7 bits
R-type	funct7	rs2	rs1	funct3	rd	opcode
I-type	immediate[11:0]		rs1	funct3	rd	opcode
S-type	immed[11:5]	rs2	rs1	funct3	immed[4:0]	opcode
B-type	immed[11,9:4]	rs2	rs1	funct3	immed[3:0,10]	opcode
U-type	immediate[19:0]				rd	opcode
J-type	immediate[19,9:0,10,18:11]				rd	opcode

- In B-type (SB-type) and J-type (UJ-type), immediate bits are swirled around
  - Create difficulty for assemblers
  - But save hardware (muxes) on the critical path

RISC-V Addressing Summary



Big Picture – CPU and Data