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 (`lui` and `auipc`)
    - 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

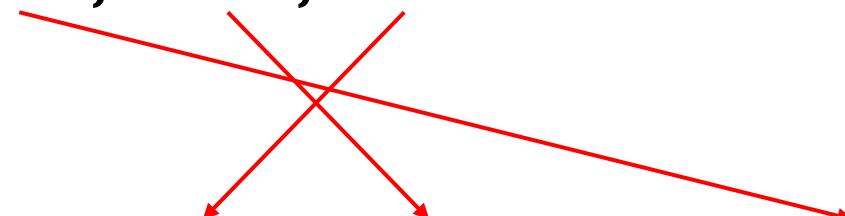| funct7	rs2	rs1	funct3	rd	opcode
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

- Instructions with registers as all operands
- Instruction fields
  - opcode: operation code
  - rd: destination register number
  - funct3: 3-bit function code (additional opcode)
  - rs1: the first source register number (5 bits)
  - rs2: the second source register number
  - funct7: 7-bit function code (additional opcode)
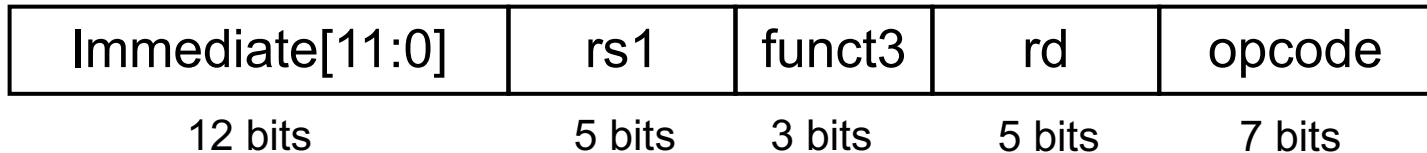
R-type Example

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

- Instructions involving immediate numbers
    - rs1: source register number
    - rd: destination register number
    - immediate: 12-bit constant operand
        - For most of I-type instructions: 2's complement, sign extended
        - For several unsigned instructions (e.g. `sltiu`): unsigned, zero extended
- 12-bit 2's complement number
    - Range: $[-2^{11} \sim 2^{11}-1]$ = [-2048 ~ 2047] = [0x800 ~ 0x7FF]

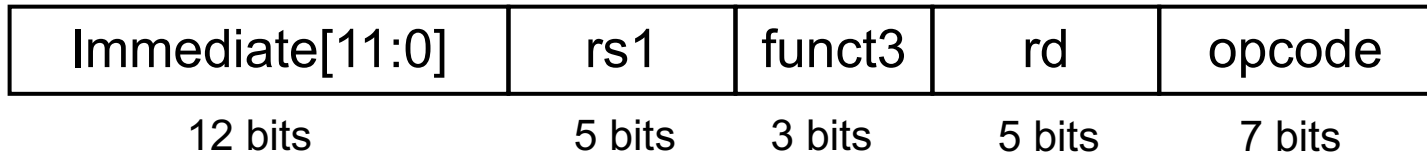# I-type Example 1

`addi x9, x20, 4`

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

4	20	0	9	19

000000000100	10100	000	01001	0010011
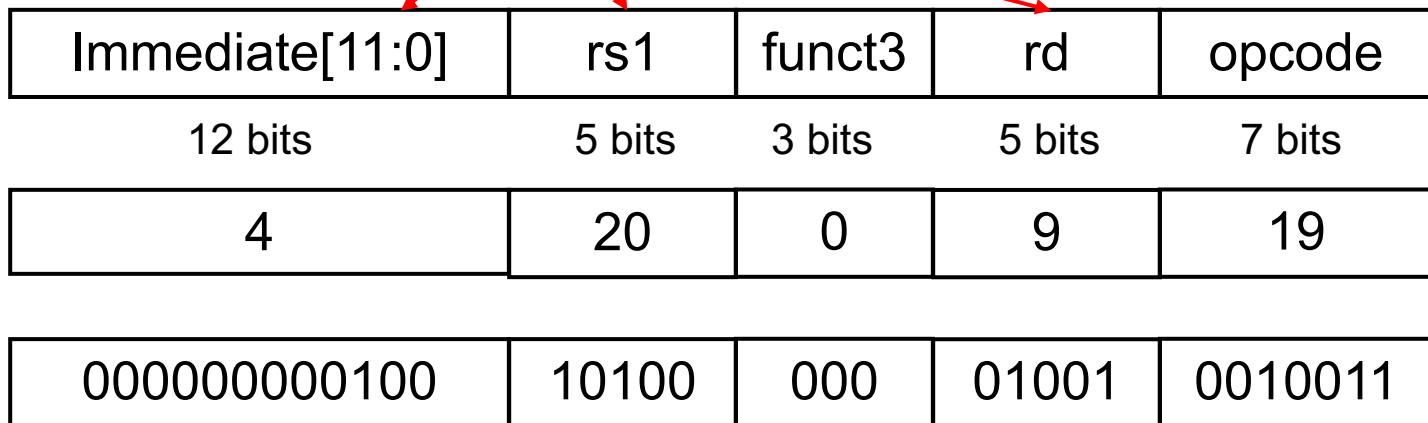
$$00000000010010100000010010010011_2 = 004A0493_{16}$$

I-type

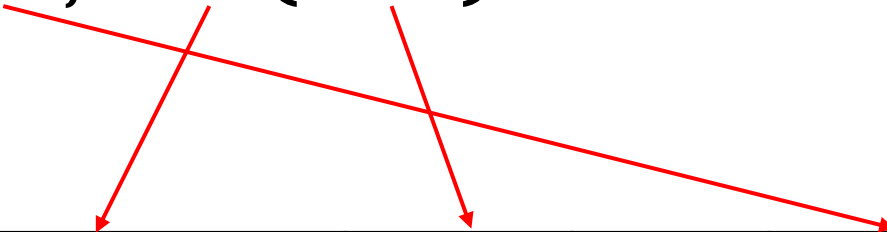| Immediate[11:0]	rs1	funct3	rd	opcode
12 bits	5 bits	3 bits	5 bits	7 bits

- Load instructions are also I-type
  - rs1: base address register number
  - rd: load destination register number
  - immediate: 12-bit offset added to base address
    - 2's-complement, sign extended
    - For `lbu`, `lhu`: immediate number is a signed number, the byte or half-word loaded are unsigned, zero extended in rd
- *Design Principle 4:* Good design demands good compromises
  - Keep formats as similar as possible
  - rs1, funct3, rd, and opcode are all aligned between R-type and I-type

I-type Example 2

`lw x9, -4(x20)`

Immediate[11:0]	rs1	funct3	rd	opcode
12 bits	5 bits	3 bits	5 bits	7 bits
-4	20	2	9	3
111111111100	10100	010	01001	0000011

$111111111100101000100100000011_2 = \text{FFCA2483}_{16}$

S-type

imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

- 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 as in other instructions – more work on Assembler, but less cost in hardware design
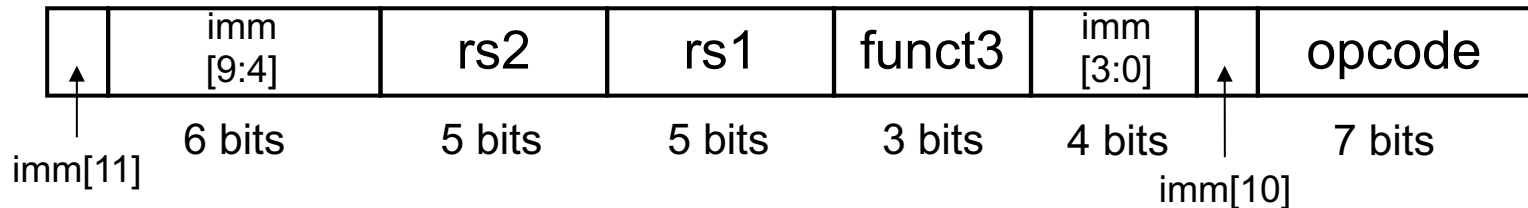
11

S-type Example

`sw x9, -4(x20)`

imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits
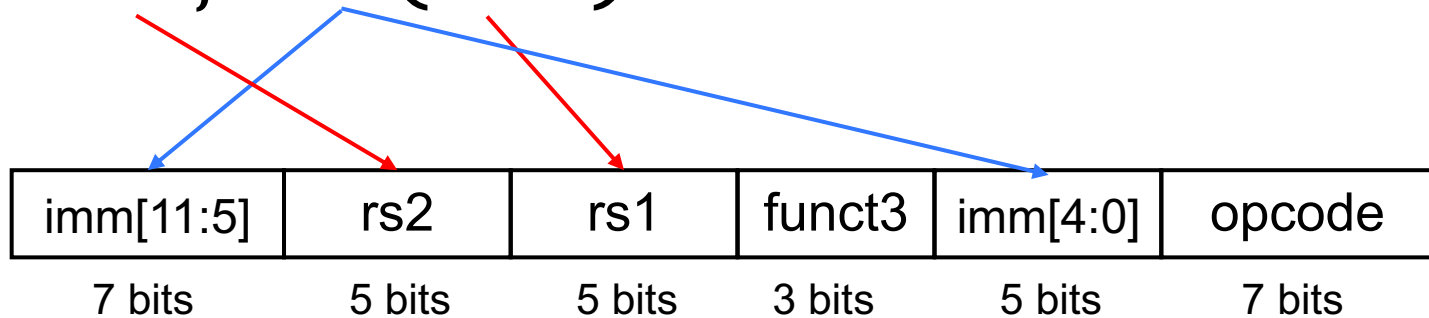
-4 = **1111111**_**11100**

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
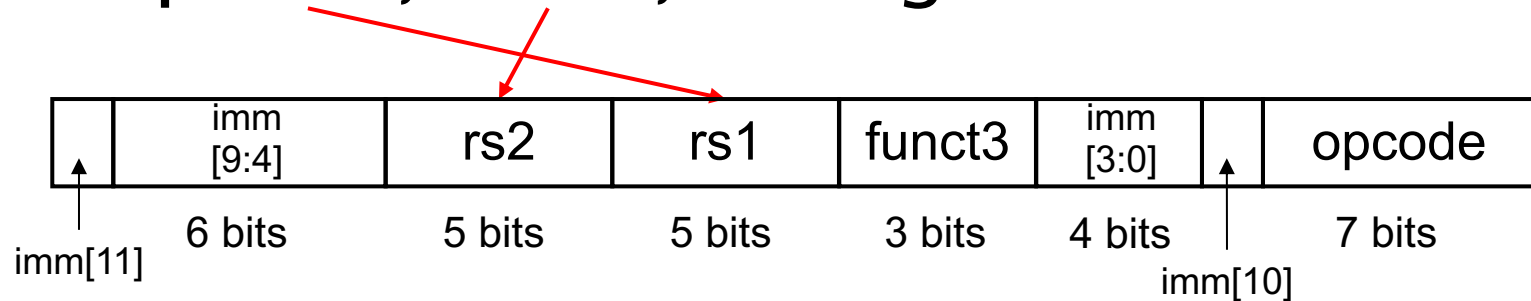 - So 12-bit signed immediate is enough
- B type:

imm [9:4]	rs2	rs1	funct3	imm [3:0]		opcode
6 bits	5 bits	5 bits	3 bits	4 bits		7 bits

imm[11]

imm[10]

Branch Target address (Target PC)

= Current PC + immediate [11:0] × 2

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

B-type Example

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

imm [9:4]	rs2	rs1	funct3	imm [3:0]		opcode
6 bits	5 bits	5 bits	3 bits	4 bits		7 bits

imm[11]

imm[10]

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

e.g.: if immediate = -4 = **1_1_111111_1100**

1	111111	10101	10100	000	1100	1	1100011

-1	21	20	0	25	99

$11111111010110100000110011100011_2 = FF5A0CE3_{16}$

14

U-type

Immediate [19:0]	rd	opcode
20 bits	5 bits	7 bits
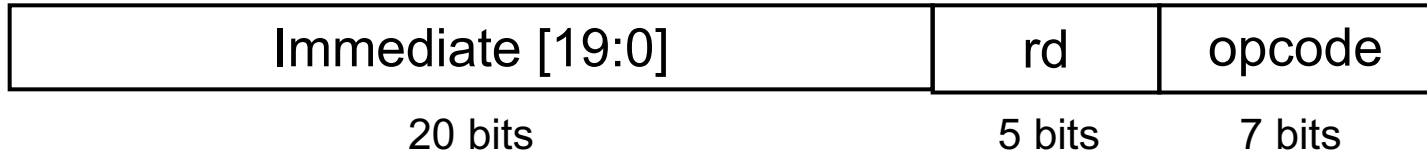
- For load upper immediate (`lui`) instruction and add upper immediate to PC (`auipc`)

U-type Example

`lui x20, 0x12345`

Immediate [19:0]	rd	opcode
20 bits	5 bits	7 bits

0x12345	20	0x37

0001_0010_0011_0100_0101	10100	0110111
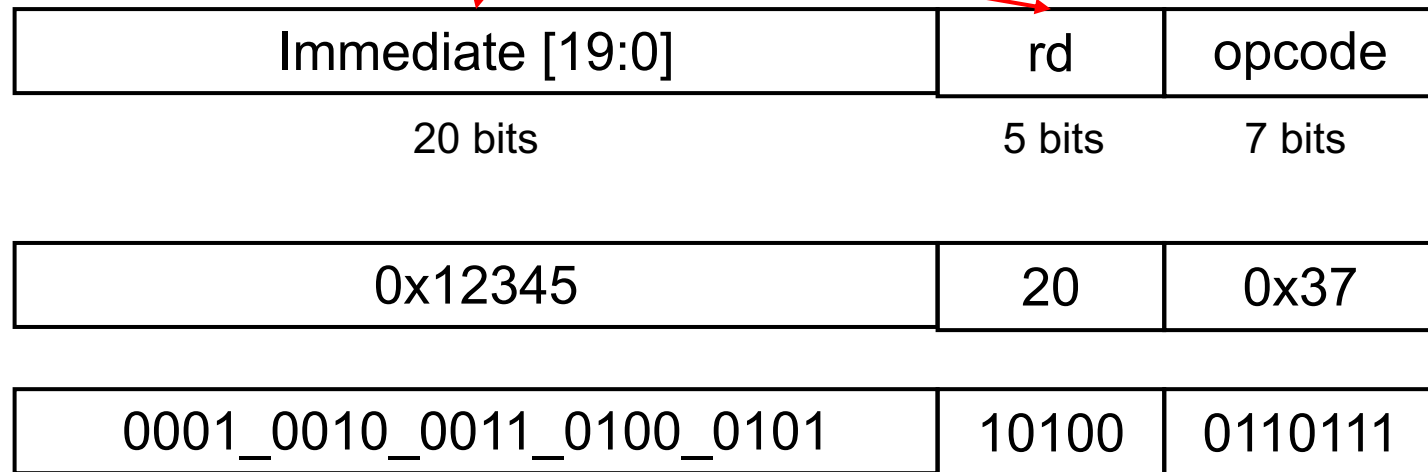
$$00010010001101000101101000110111_2 = 12345A37_{16}$$

J-type

imm[9:0]	imm[18:11]	rd	opcode
10 bits	8 bits	5 bits	7 bits

imm[19]   imm[10]
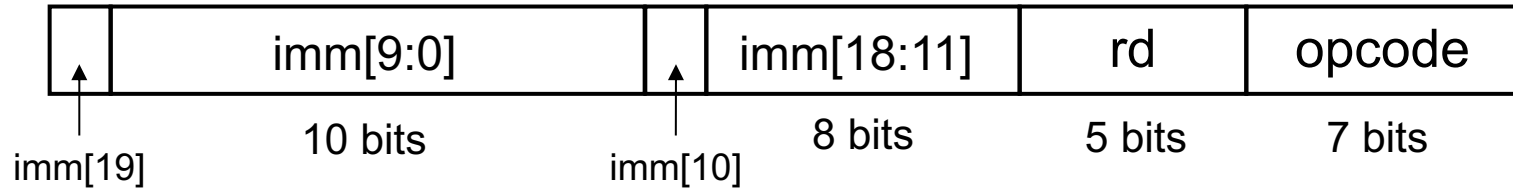
- For Jump and link (`jal`)
    - x1 <= PC + 4, x1: return address reg.
    - PC <= Target PC
            = Current PC + immediate × 2

- Jump instruction uses 20-bit immediate for larger jumping range
- Note: `jalr` is an I-type

J-type Example
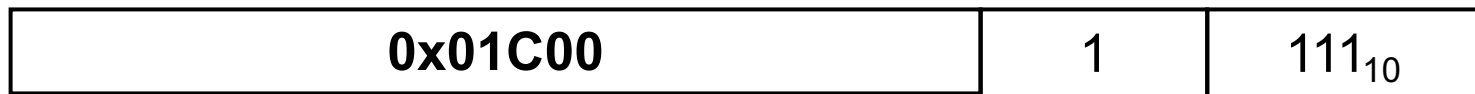
`jal x1, Target`

imm[9:0]	imm[18:11]	rd	opcode
10 bits	8 bits	5 bits	7 bits

imm[19]   imm[10]
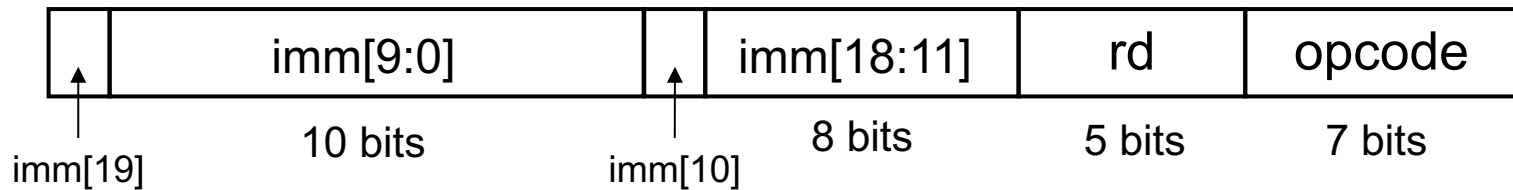
immediate = (Target PC – Current PC) >> 1

e.g.: if immediate = 14 = **0**_**00000000**_**0**_**0000001110**
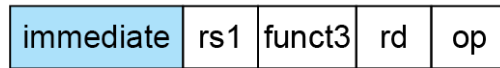
0	0000001110	0	00000000	00001	1101111

0x01C00	1	111_{10}

$00000001110000000000000011101111_2 = 01C000EF_{16}$

Performance Considerations

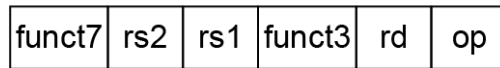| Type	Field					
	7 bits	5 bits	5 bits	3 bits	5 bits	7 bits
R-type	funct7	rs2	rs1	funct3	rd	opcode
I-type	immediate[11:0]		rs1	funct3	rd	opcode
S-type	immed[11:5]	rs2	rs1	funct3	immed[4:0]	opcode
B-type	immed[11,9:4]	rs2	rs1	funct3	immed[3:0,10]	opcode
U-type	immediate[19:0]				rd	opcode
J-type	immediate[19,9:0,10,18:11]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

RISC-V Addressing Summary
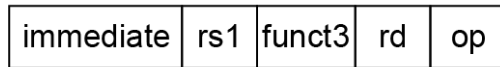
1. Immediate addressing

immediate	rs1	funct3	rd	op

2. Register addressing

funct7	rs2	rs1	funct3	rd	op

Registers

Register

3. Base addressing

immediate	rs1	funct3	rd	op

Register +

Memory

Byte | Halfword | Word | Doubleword

4. PC-relative addressing

imm	rs2	rs1	funct3	imm	op

PC +
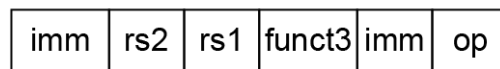
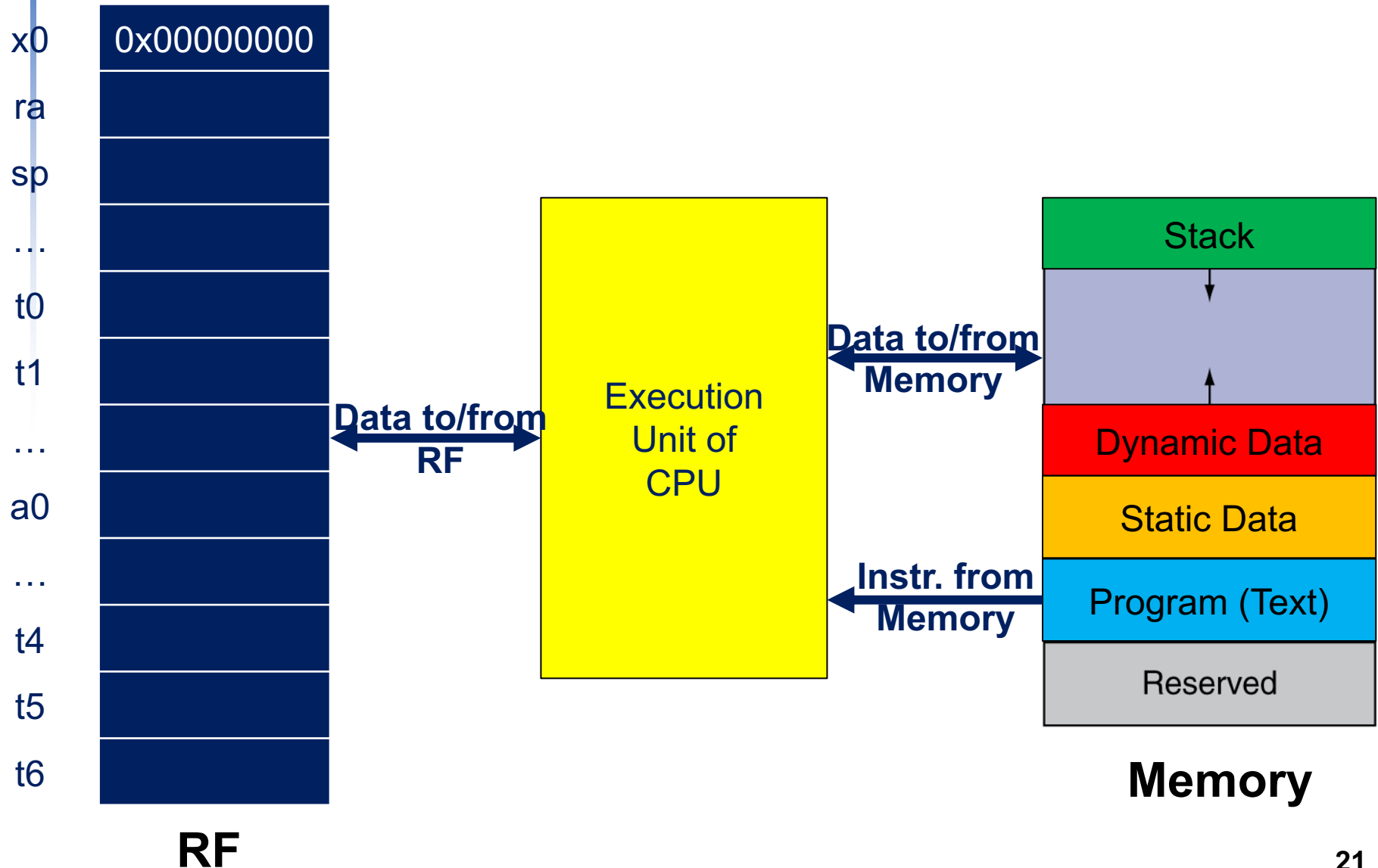Memory

Word

Big Picture – CPU and Data

Exercise

```
Loop: sll  x6, x18, 2      0x00080000
      add  x6, x6, x20      0x00080004
      lw   x5, 0(x6)        0x00080008
      bne  x5, x19, Exit    0x0008000C
      addi x18, x18, 1      0x00080010
      jal x0, Loop          0x00080014
Exit: …                     0x00080018
```