MIPS Coding Continued

Exercise 1

- Suppose we have three arrays, `A, B, C`, all of size 10. Now we want to set `C[i] = min(A[i], B[i])` for all `0<= i <= 9`.

Exercise 1

- Suppose we have three arrays, `A, B, C`, all of size 10. Now we want to set `C[i] = min(A[i], B[i])` for all `0<= i <= 9`.
  - First, we need a loop to walk through the elements (done before)
  - Second, we need to be able to read the elements (done before)
  - Third, we need to be able to compare two numbers (done before)
  - Fourth, we need to write back to the memory (easy)

```
    .data
A:  .word 12, 34, 67, 1, 45, 90, 11, 33, 67, 19
B:  .word 90, 2, 93, 66, 8, 120, 121, 11, 33, 9
C:  .space 40

    .text
    .globl main
main:

done:
    li $v0,10
    syscall
```

```
    .data
A:  .word 12, 34, 67, 1, 45, 90, 11, 33, 67, 19
B:  .word 90, 2, 93, 66, 8, 120, 121, 11, 33, 9
C:  .space 40

    .text
    .globl main
main:
    la $s0, A                       # array A
    la $s1, B                       # array B
    la $s2, C                       # array C
    li $s3, 10                      # length of the arrays
    li $t0, 0                       # using $t0 as I

done:
    li $v0,10
    syscall
```

```
    .data
A:  .word 12, 34, 67, 1, 45, 90, 11, 33, 67, 19
B:  .word 90, 2, 93, 66, 8, 120, 121, 11, 33, 9
C:  .space 40

    .text
    .globl main
main:
    la $s0, A                   # array A
    la $s1, B                   # array B
    la $s2, C                   # array C
    li $s3, 10                  # length of the arrays
    li $t0, 0                   # using $t0 as I

LOOP:

    addi $t0, $t0, 1            # i ++
    bne $t0, $s3, LOOP          # go back if not yet 10 times


done:
    li $v0,10
    syscall
```

```
    .data
A:  .word 12, 34, 67, 1, 45, 90, 11, 33, 67, 19
B:  .word 90, 2, 93, 66, 8, 120, 121, 11, 33, 9
C:  .space 40

    .text
    .globl main
main:
    la $s0, A                     # array A
    la $s1, B                     # array B
    la $s2, C                     # array C
    li $s3, 10                    # length of the arrays
    li $t0, 0                     # using $t0 as I

LOOP:
    sll $t4, $t0, 2               # $t4 = i * 4
    add $t5, $t4,$s0              # $t5 will have the address of A[i]
    lw $t1, 0($t5)                # $t1 has A[i]
    add $t6, $t4,$s1              # $t6 will have the address of B[i]
    lw $t2, 0($t6)                # $t2 has B[i]

    addi $t0, $t0, 1              # i ++
    bne $t0, $s3, LOOP            # go back if not yet 10 times


done:
    li $v0,10
    syscall
```

```
    .data
A:  .word 12, 34, 67, 1, 45, 90, 11, 33, 67, 19
B:  .word 90, 2, 93, 66, 8, 120, 121, 11, 33, 9
C:  .space 40

    .text
    .globl main
main:
    la $s0, A                    # array A
    la $s1, B                    # array B
    la $s2, C                    # array C
    li $s3, 10                   # length of the arrays
    li $t0, 0                    # using $t0 as I

LOOP:
    sll $t4, $t0, 2              # $t4 = i * 4
    add $t5, $t4,$s0             # $t5 will have the address of A[i]
    lw $t1, 0($t5)               # $t1 has A[i]
    add $t6, $t4,$s1             # $t6 will have the address of B[i]
    lw $t2, 0($t6)               # $t2 has B[i]

    add $t6, $t4, $s2            # now $t6 has the address of C[i]
    sw $t8, 0($t6)               # now C[i] has the minimum of A[i] and B[i]
    addi $t0, $t0, 1             # i ++
    bne $t0, $s3, LOOP           # go back if not yet 10 times


done:
    li $v0,10
    syscall
```

```
    .data
A:  .word 12, 34, 67, 1, 45, 90, 11, 33, 67, 19
B:  .word 90, 2, 93, 66, 8, 120, 121, 11, 33, 9
C:  .space 40

    .text
    .globl main
main:
    la $s0, A                   # array A
    la $s1, B                   # array B
    la $s2, C                   # array C
    li $s3, 10                  # length of the arrays
    li $t0, 0                   # using $t0 as I

LOOP:
    sll $t4, $t0, 2             # $t4 = i * 4
    add $t5, $t4,$s0            # $t5 will have the address of A[i]
    lw $t1, 0($t5)              # $t1 has A[i]
    add $t6, $t4,$s1            # $t6 will have the address of B[i]
    lw $t2, 0($t6)              # $t2 has B[i]
    slt $t5, $t1, $t2           # set $t5 to be 1 if A[i] < B[i]
    beq $t5, $0, L1             # if $t5 == 0, goto L1. in this case, A[i] >= B[i]
    ori $t8, $t1, 0             # setting $t8 to be A[i]
    j L2                        # always remember to jump in an if else!
L1:
    ori $t8, $t2, 0             # setting $t8 to be B[i]
L2:
    add $t6, $t4, $s2           # now $t6 has the address of C[i]
    sw $t8, 0($t6)              # now C[i] has the minimum of A[i] and B[i]
    addi $t0, $t0, 1            # i ++
    bne $t0, $s3, LOOP          # go back if not yet 10 times


done:
    li $v0,10
    syscall
```

Representing Instructions in Computers

- Note that computers only have 0's and 1's
- Before we can load MIPS instructions into memory, they need to be translated into machine instructions, which consist of only 0's and 1's
  - In other words, we need to encode or represent instructions
  - The symbolic representation of machine instructions is called assembly language
  - The binary representation of instructions is called machine language
    - A sequence of instructions in binary form is called machine code

Example

```
0x8e700000  lw $16, 0($19)
0x8e680004  lw $8, 4($19)
0x02088020  add $16, $16, $8
0x8e680008  lw $8, 8($19)
0x02088020  add $16, $16, $8
0x8e68000c  lw $8, 12($19)
0x02088020  add $16, $16, $8
0x8e680010  lw $8, 16($19)
0x02088020  add $16, $16, $8
```

MIPS Instruction Encoding

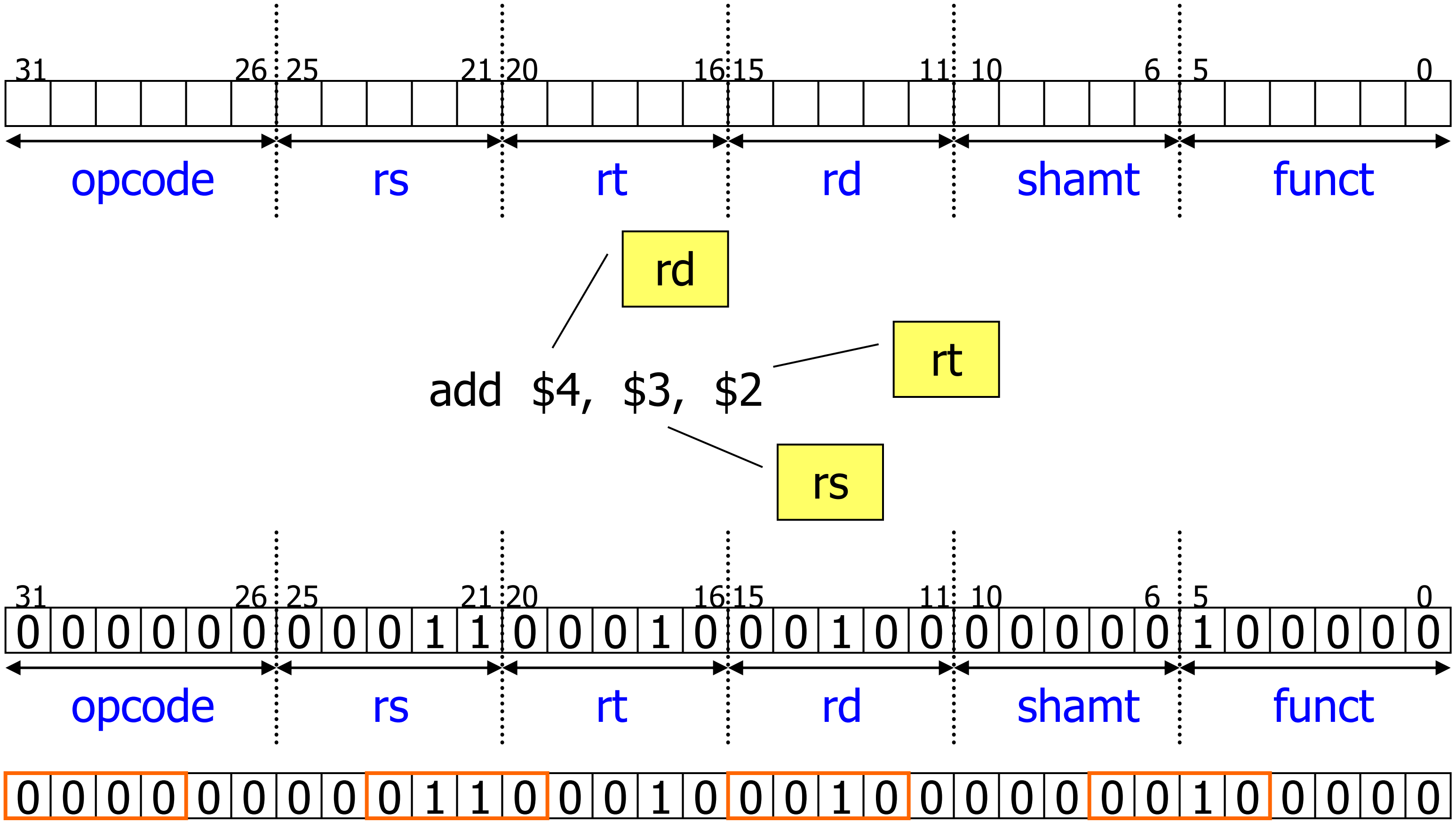
- Each MIPS instruction is exactly 32 bits
  - R-type (register type)
  - I-type (immediate type)
  - J-type (jump type)

op	**rs**	**rt**	**rd**	**shamt**	**funct**

op	**rs**	**rt**	**16 bit address or constant**

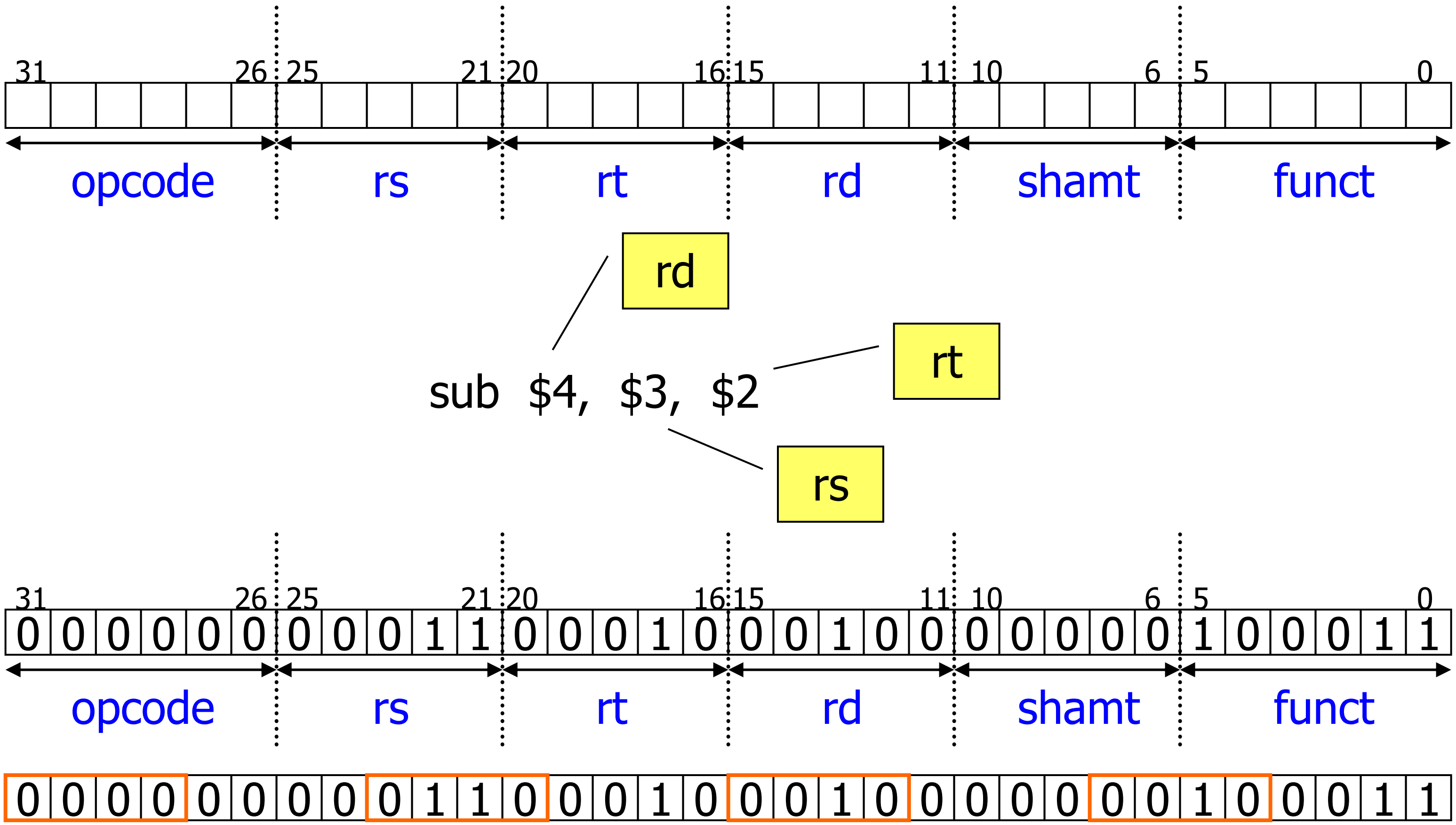
op	**26 bit address**

R-Type Encoding



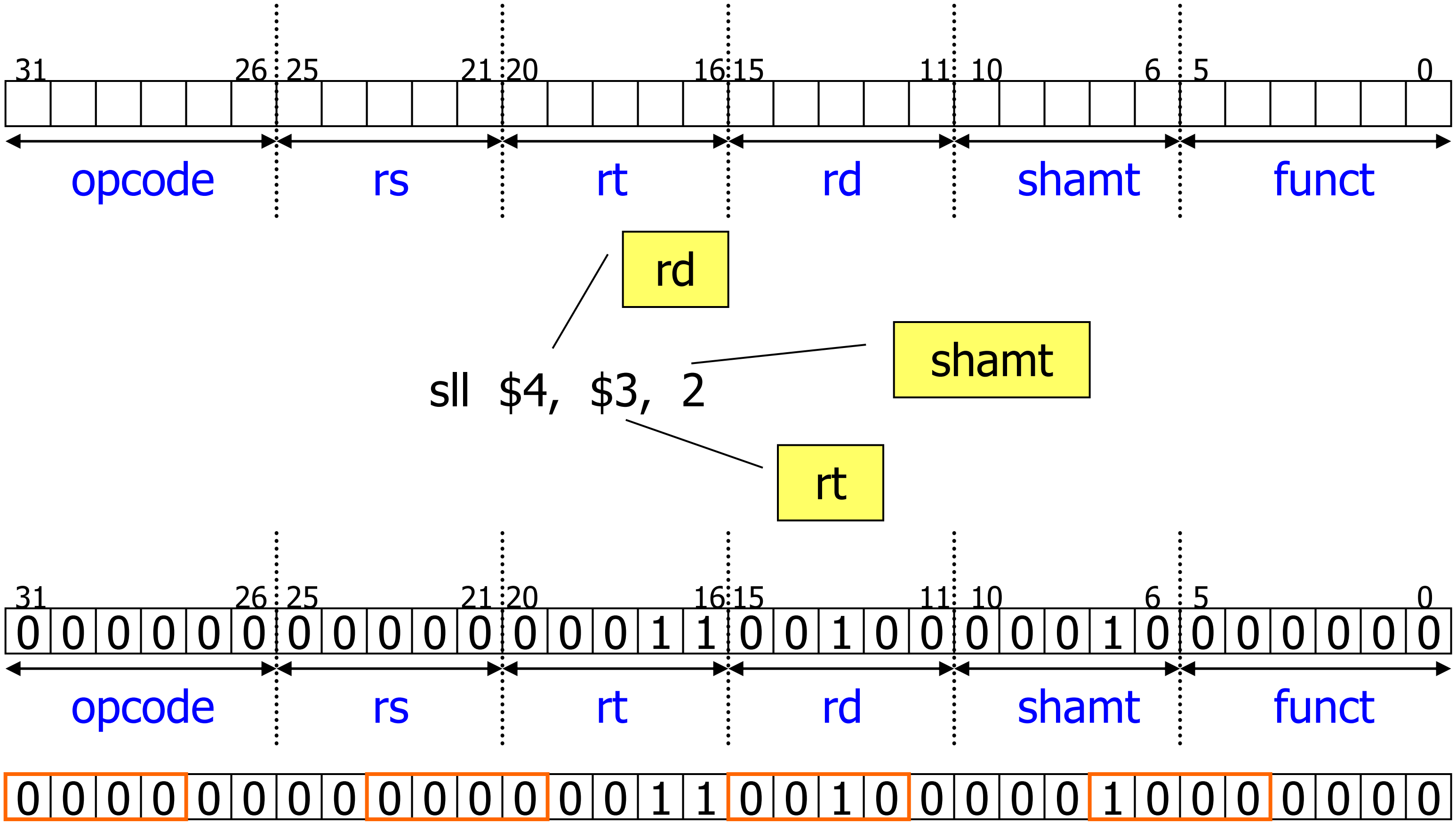
Encoding = 0x00622020

R-Type Encoding



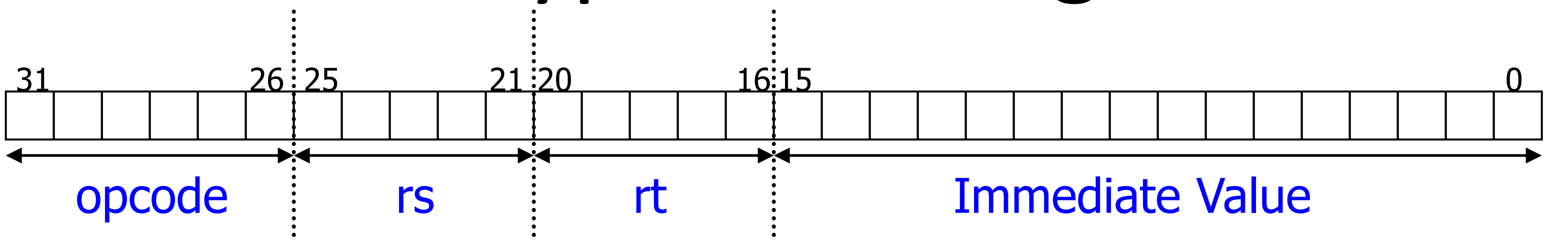
Encoding = 0x00622023

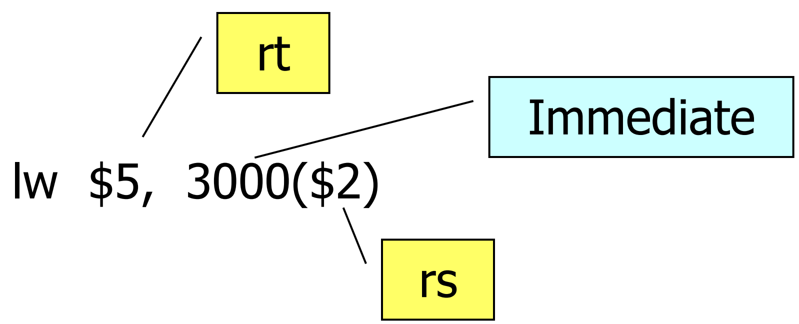
R-Type Encoding

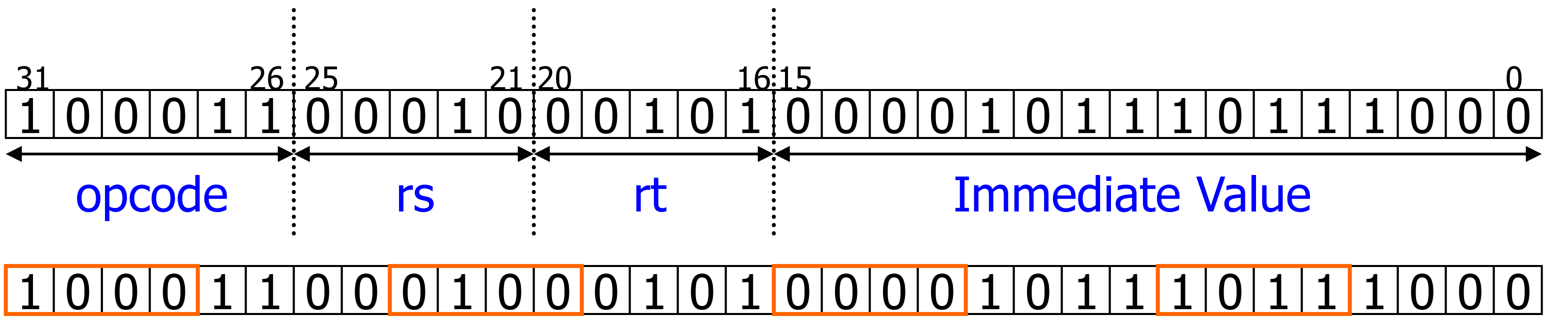


Encoding = 0x00032080

I-type Encoding

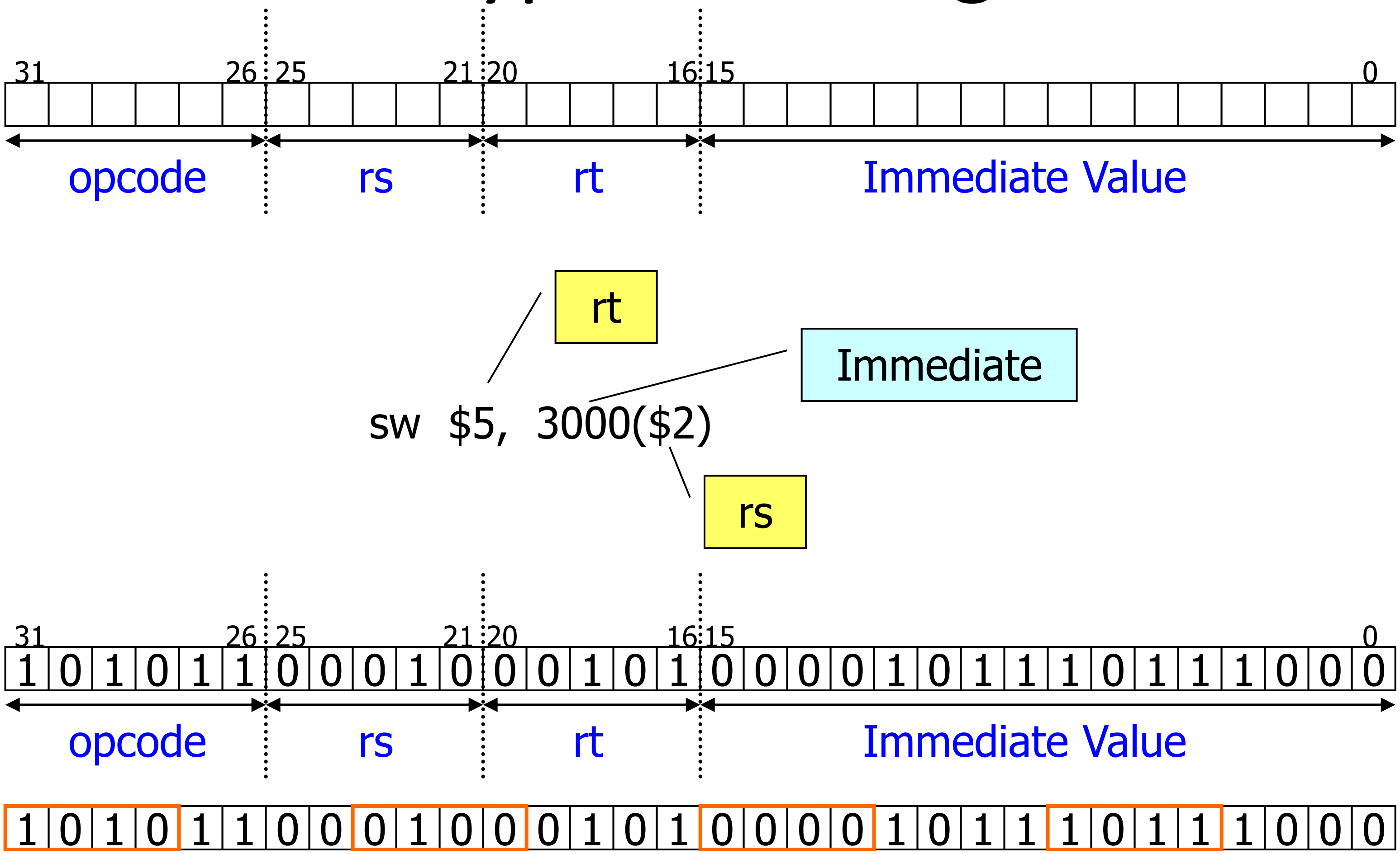






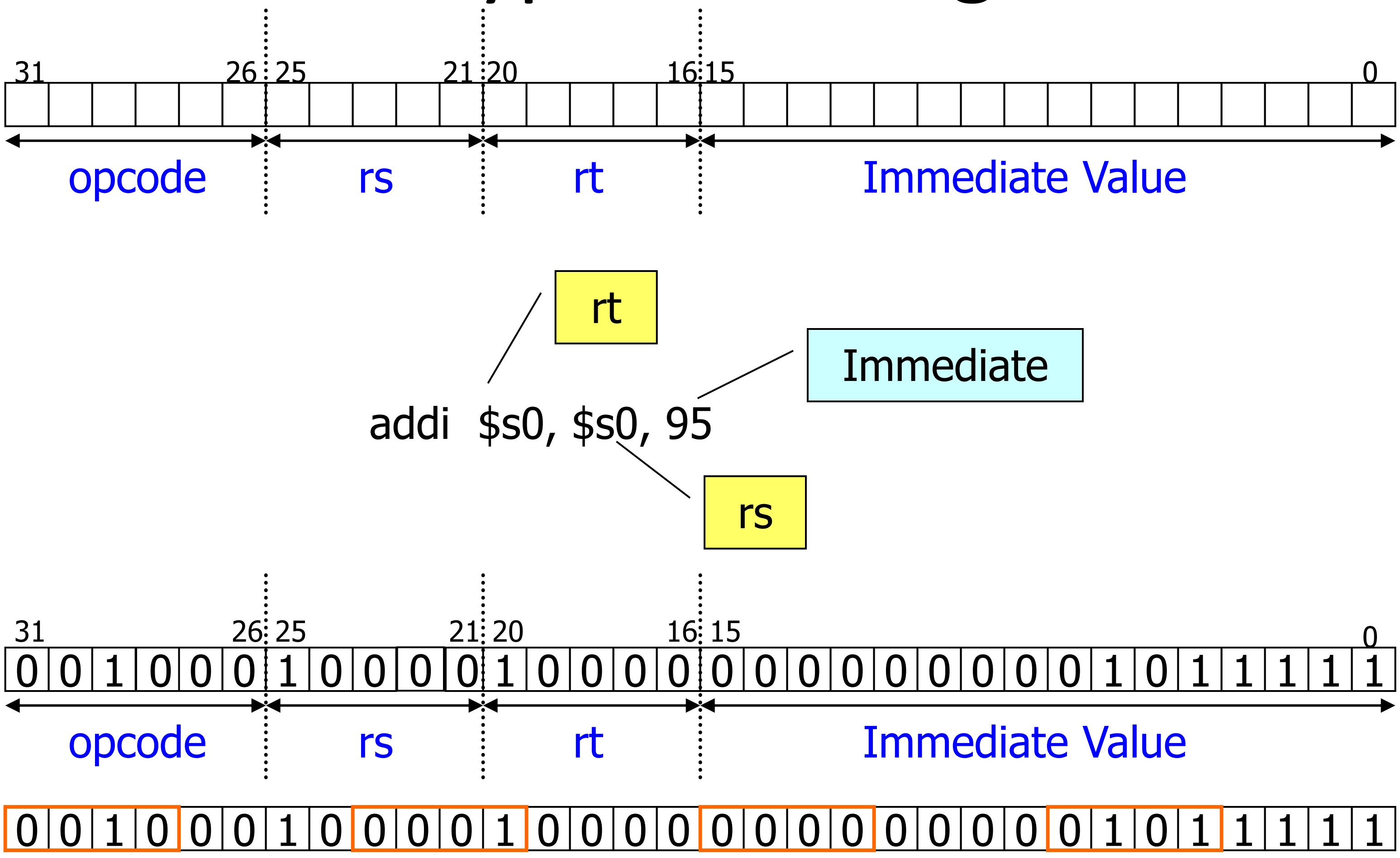
Encoding = 0x8C450BB8

I-type Encoding



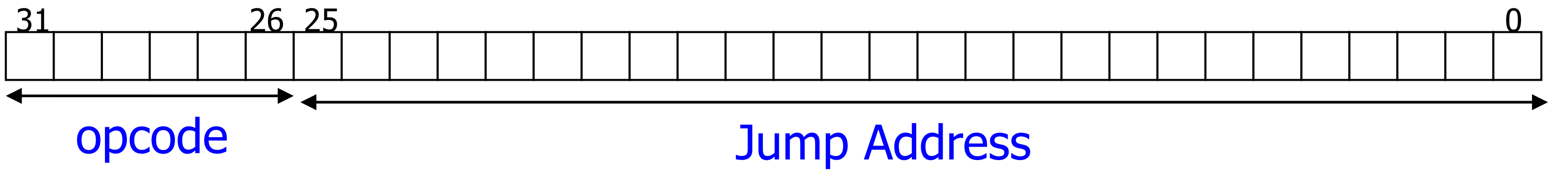
Encoding = 0xAC450BB8

I-type Encoding



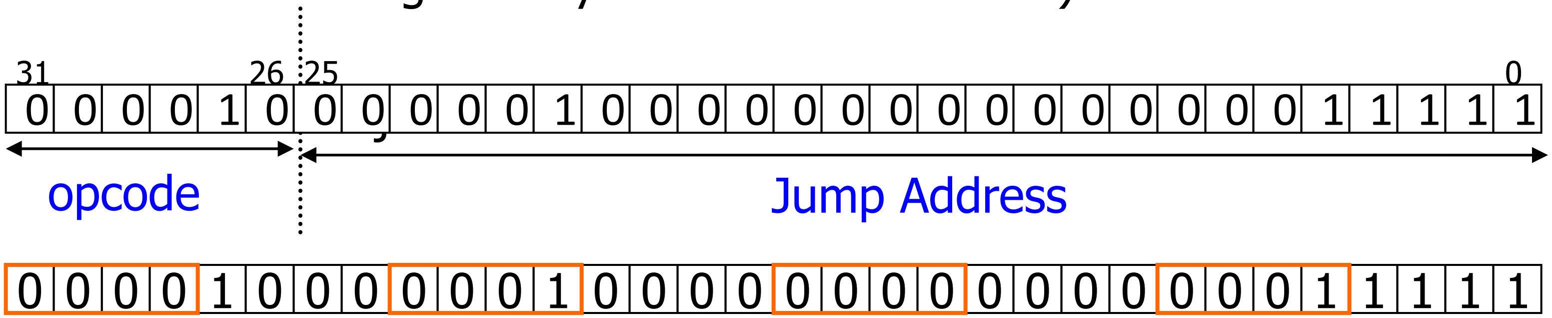
Encoding = 0x2210005F

J-type Encoding



j 0x0040007c — Jump Address

0x0040007c: the address of the instruction to jump to. When encoding it, take the bit 2 to bit 27 (you can also think of it as doing a srl by 2 bits on the address).



Encoding = 0x0810001F

How to Encode Branch Instructions

- To encode these branch instructions, we first need to figure out the value for the associated label
  - This will be done by the assembler
  - Note that the MIPS has the alignment restriction, which means all the labels will be a multiple of 4
  - To increase the range, the address divided by 4 is actually encoded
    - In other words, the address is in terms of words (32 bits), rather than bytes

Encoding Conditional Branch Instructions

- It branches the number of the instructions specified by the offset if register rs equals to register rt

```
beq rs, rt, label
```

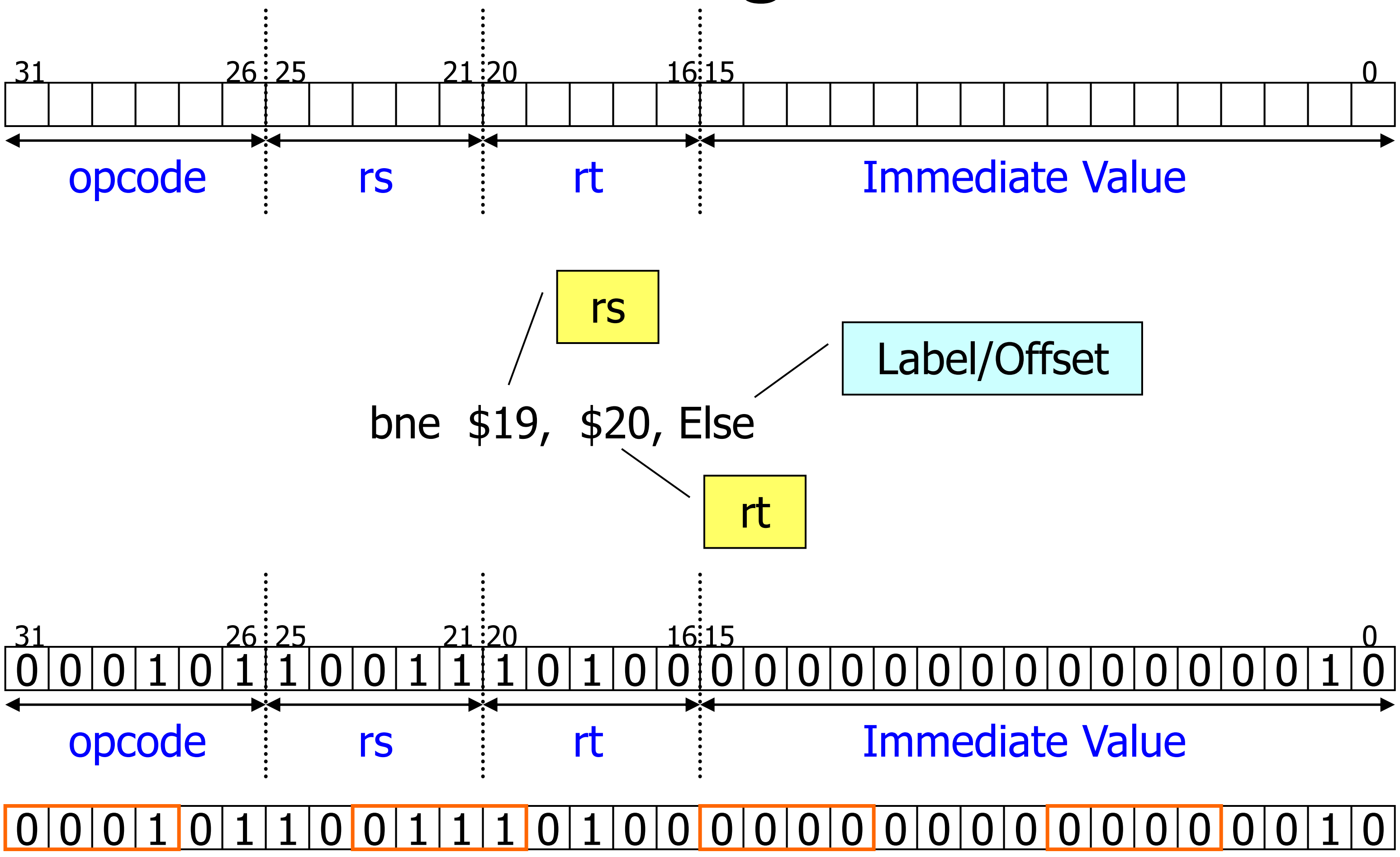
4	rs	rt	Offset
6	5	5	16

  - In the stored-program concept, we implicitly need a register to hold the address of the current instruction being executed
    - Which is called program counter (PC) (should be called instruction address register)
  - What is the value of PC after we finish executing the current instruction?

Encoding Conditional Branch Instructions

- PC-relative addressing
  - The offset of conditional branch instructions is relative to PC + 4
  - Since all MIPS instructions are 4 bytes long, the offset refers to the number of words to the next instruction instead of the number of bytes

Encoding bne



Encoding = 0x16740002