CS202 · Computer Organization

Assignment 2

何泽安 (He Zean)
12011323
Mar. 15, 2022

Problem 1

We first check the 6 most significant bits (*op-code*) and find them are all zeros, which prompts us that this is a *R-type instruction*.

000000	10000	10000	10000	00000	100000
op	rs	rt	rd	shamt	funct

Then check up the funct, $100000_{\text{bin}} = 20_{\text{hex}}$, which is *add*. Also, the number $10000_{\text{bin}} = 16_{\text{dec}}$ of register is $\$s0$. Thus this instruction can be parsed as $\$s0 = \$s0 + \$s0$ or

```
add $s0, $s0, $s0
```

Problem 2

The table prompts us *sw* is an *I-type* function ($2b_{\text{hex}}$), with a definition of M[R[rs]+SignExtImm] = R[rt]. Now that we have $rt = \$t1 = 9_{\text{dec}}$ and $rs = \$t2 = 10_{\text{dec}}$, with an immediate of +32.

$2b_{\text{hex}}$	10_{dec}	9_{dec}	$+32_{\text{dec}}$
101011	01010	01001	0000 0000 0010 0000
op	rs	rt	immediate

Then convert the 32-bit binary instruction to hexadecimal:

$$1010\ 1101\ 0100\ 1001\ 0000\ 0000\ 0010\ 0000_{\text{bin}} = \text{AD490020}_{\text{hex}}$$

Problem 3

```
sll $t0, $s3, 2    # $t0 = (i) << 2, the bias bytes of A[i] to A[0]
add $t0, $t0, $s6  # $t0 now is the addr of A[i]
lw  $t0, 0($t0)    # $t0 = A[i]

sll $t1, $s4, 2    # $t1 = (j) << 2, the bias bytes of A[j] to A[0]
add $t1, $t1, $s6  # $t1 now is the addr of A[j]
lw  $t1, 0($t1)    # $t1 = A[j]

add $t0, $t0, $t1  # $t0 = A[i] + A[j]
sw  $t0, 32($s7)   # save $t0 to B[8]
```

Problem 4

4.1

We first translate the assembly into C, then it's trivial that the loop will repeat 10 times, each loop increases B to B + 2. The initially zero B will be $0 + 2 \cdot 10 = 20$ eventually.

4.2

We can first straightly translate the assembly into C style code:

```
Loop:
    temp = 0 < i;
    if (temp == 0) goto Done;
    i = i - 1;
    B = B + 2;
    goto Loop;
Done:
    // other codes //
```

Then make it more elegant:

```
while (i > 0) {
    i -= 1;
    B += 2;
}
```

Problem 5

One should use instructions that works with immediate number to make the code elegant. We can find relevant instructions *lui* and *ori*.

```
lui $t1, 0x2001        # $t1 = 0010 0000 0000 0001 (0000 0000 0000 0000)
ori $t1, $t1, 0x4924   # $t1 = (0010 0000 0000 0001) 0100 1001 0010 0100
```