Lab Report - 06

PB20111686 Ruixuan Huang

Lab Goal

Use a high-level programming language to implement all the code that has been written before.

Solution List

lab0l (lab1 L version)

```
short lab1l(short a, short b) {
    short beta = b;
    short alpha = a;
    while (--alpha) {
        beta += b;
    }
    return beta;
}
```

lab0p (lab1 P version)

```
short lab1p(short a, short b) {
    short mask = 1;
    short alpha = a;
    short beta = b;
    short ans = 0;
    while (beta) {
        if (alpha & mask) {
            ans += beta;
        }
        beta <<= 1;
        mask <<= 1;
    }
    return ans;
}
```

fib (lab2 fibonacci)

```
short lab2(short n) {
    short r1 = 0;
    short r2 = 1;
    short r3 = 2;
    while (n--) {
        short tmp = r3 + 2 * r1;
        tmp &= (short)2047;
        r1 = r2;
        r2 = r3;
        r3 = tmp;
    }
    return r3 / 2;
}
```

fib-opt (lab3 fibonacci)

```
short lab3(short n) {
    short tab[147] = {
1,2,4,6,10,18,30,50,86,146,246,418,710,178,1014,386,742,722,470,930,326,242,54,706,166,274,6
62,994,518,818,758,770,358,850,342,34,710,370,438,834,550,402,22,98,902,946,118,898,742,978,
726,162,70,498,822,962,934,530,406,226,262,50,502,2,102,82,86,290,454,626,182,66,294,658,790
,354,646,178,886,130,486,210,470,418,838,754,566,194,678,786,150,482,6,306,246,258,870,338,8
54,546,198,882,950,322,38,914,534,610,390,434,630,386,230,466,214,674,582,1010,310,450,422,1
8,918,738,774,562,1014,514,614,594,598,802,966,114,694,578,806,146,278,866,134,690,374,642,9
98,722,982 };
    if(n < 20) return tab[n - 1];
    return tab[20 + (n - 20) & 127 - 1];
}
```

rec (lab4 task1 rec)

In C++ programming, every piece of data has an type. This is different from LC-3 since in the latter a 16-bit vector can be considered as address or number. So this task's C++ programming realization will follow the original program's idea, not its result.

```
void lab4_task1(short r1) {
    if (r1 >= 1)
        lab4_task1(r1 - 1);
    return;
}
```

mod (lab4 task2 mod)

```
short lab4_mod712(short n) {
    short r0 = 0, r1 = n, r2, r3, r4;
    do {
        goto x;
    y:;
        r2 = r1 & 7;
        r1 = r2 + r4;
        r0 = r1 - 7;
    } while (r0 > 0);
    r0 = r1 - 7;
    if (r0 < 0)return r1;
    r1 = r1 - 7;
    return r1;
x:;
    r2 = 1, r3 = 8, r4 = 0;
    do {
        if (r3 & r1) {
            r4 = r2 + r4;
        }
        r2 <<= 1;
        r3 <<= 1;
    } while (r3 != 0);
    goto y;
}
```

prime (lab5 prime)

```
short judge(short n) {
    short r0 = n, r1 = 1, r2 = 2, r3, r4, r5;
    while (1) {
        r3 = 0, r4 = r2;
        do {
            r3 += r2;
            r4--;
        } while (r4);
```

```
            r3 = r0 + ~r3 + 1;
            if (r3 >= 0) {
                r5 = r0;
                do {
                    r5 = r5 - r2;
                } while (r5 > 0);
                if (r5 == 0) {
                    r1 = 0;
                    return r1;
                }
                if (r5 < 0) {
                    r2++;
                    continue;
                }
            }
            else return r1;
        }
    }
```

Disscussion

How to evaluate the performance of your own high-level language programs?

We can use a instruction counter to count how many instructions the program carried. And add one `cnt++;` instruction behind each line of the original program. Here I use the first program in the program list as an example.

```
int cnt = 0;
short lab1l(short a, short b) {
    short beta = b; cnt++;
    short alpha = a; cnt++;
    while (--alpha) {
        cnt++;
        beta += b;
        cnt++;
    }
    return beta;
}
```

Why is a high-level language easier to write than LC3 assembly?

The structure of high-level programming language is closer to human natural language and human thinking mode. And it is more free and flexible. For example, the loop jump mode of LC-3 assembly is sometimes very confusing.

What instructions do you think need to be added to LC3?

I think the right shift operation should be added to LC-3. What should be considered when designing the right shift operation is whether the high bits should be subpplemented by 0 or the symbol. When we use this operation, we often need to obtain the information of each bit. So we need an exit condition. If it is SEXT, it is not convenient to judge. Therefore, here this operation I'm talking about is the high-bits-supply-0 mode. Once added, the division related to the power of 2 will be simplified greatly.

Is there anything you need to learn from LC3 for the high-level language you use?

To finish some specific tasks, the operation with assembly ideas can improve efficiency. For example, take the remainder of some specific numbers, or multiply with assembly ideas. Using assembly to do multiplication may be hundreds of times faster than ordinary multiplication in C language. These extreme optimizations are rarely concerned before, but memory and performance are very important when programming on some special computers (like single chip microcomputer).