优化文档

21373293　王雨帆

由于确实期末时间紧，笔者只在后端做了简单的**窥孔优化**和**乘除法优化**，并写好了寄存器分配的部分接口，但考虑到后续时间不够完善和 debug，并未真正启动寄存器分配。

1 窥孔优化

1.1 优化连续对同一地址的 `sw` `lw`

```
public ArrayList<Assembly> delLwAfterSw(ArrayList<Assembly> textSegment) {
        ArrayList<Assembly> instructions = new ArrayList<>();
        int i;
        int length = textSegment.size();
        for (i = 0; i < length; i++) {
            Assembly cur = textSegment.get(i);
            if (i == length - 1) {
                instructions.add(cur);
                break;
            }
            Assembly next = textSegment.get(i + 1);
            if (cur instanceof MemAsm && next instanceof MemAsm &&
Objects.equals(((MemAsm) cur).getMemAddr(), ((MemAsm) next).getMemAddr())
                    && ((MemAsm) cur).getOp() == MemAsm.Op.SW && ((MemAsm)
next).getOp() == MemAsm.Op.LW) {
                i++;
                instructions.add(cur);
                if (((MemAsm) cur).getReg() != ((MemAsm) next).getReg()) {
                    MoveAsm moveAsm = new MoveAsm(((MemAsm) next).getReg(),((MemAsm)
cur).getReg());
                    instructions.add(moveAsm);
                }
            } else {
                instructions.add(cur);
            }
        }
        return instructions;
    }
```

1.2 优化对同一个寄存器的 `move`

```
public ArrayList<Assembly> moveSameDst(ArrayList<Assembly> textSegment) {
        ArrayList<Assembly> instructions = new ArrayList<>();
        int i;
        int length = textSegment.size();
        for (i = 0; i < length; i++) {
            Assembly cur = textSegment.get(i);
            if (cur instanceof MoveAsm && (((MoveAsm) cur).getDst() == ((MoveAsm)
cur).getSrc())) {
```

```
                //
            }
            else {
                instructions.add(cur);
            }
        }
        return instructions;
    }
```

1.3 优化连续两条像同一个寄存器 `move`

这里需要保证第二条指令的 src 不是第一条的 dst

```
public ArrayList<Assembly> moveOverlap(ArrayList<Assembly> textSegment) {
        ArrayList<Assembly> instructions = new ArrayList<>();
        int i;
        int length = textSegment.size();
        for (i = 0; i < length; i++) {
            Assembly cur = textSegment.get(i);
            if (i == length - 1) {
                instructions.add(cur);
                break;
            }
            Assembly next = textSegment.get(i + 1);
            if (cur instanceof MoveAsm && next instanceof MoveAsm && (((MoveAsm)
cur).getDst() == ((MoveAsm) next).getDst()) && (!(((MoveAsm) next).getSrc() !=
((MoveAsm) cur).getDst()))) {
                //skip cur
            }
            else {
                instructions.add(cur);
            }
        }
        return instructions;
    }
```

1.4 优化加减 0 的 alu 指令

- 加立即数 0

```
if (op == AluAsm.Op.ADDI) {
    if (((AluAsm) cur).getImm() == 0) {
       //如果rs与rd不相等
       if (((AluAsm) cur).getRs() != ((AluAsm) cur).getRd()) {
           MoveAsm moveAsm = new MoveAsm(((AluAsm) cur).getRd(),((AluAsm)
cur).getRs());
           instructions.add(moveAsm);
       }
       //如果相等 skip cur
    }
    else {
        instructions.add(cur);
    }
}
```

- 加减的寄存器在 alu 指令之前一条正好是 `li $r0 0`

```
if (((AluAsm) cur).getRt() == Register.ZERO ||
    (pre instanceof LiAsm && (((LiAsm) pre).getImm() == 0) && ((AluAsm)
cur).getRt() == ((LiAsm) pre).getRd())) {
    if (((AluAsm) cur).getRs() != ((AluAsm) cur).getRd()) {
        MoveAsm moveAsm = new MoveAsm(((AluAsm) cur).getRd(),((AluAsm)
cur).getRs());
        instructions.add(moveAsm);
    }
    // li r2,0也可以去掉
    if (pre instanceof LiAsm && (((LiAsm) pre).getImm() == 0) && ((AluAsm)
cur).getRt() == ((LiAsm) pre).getRd()) {
       instructions.remove(pre);
       }
}
```

2 乘除法优化

2.1 优化乘 2 的幂次

```
//2的幂次
int powerFlag = 0;
int j;
for (j = 1; j < 32; j++) {
    int power = (int) Math.pow(2,j);
    //imm = 2^j
    if (imm == power) {
        powerFlag = 1;
        ShiftAsm shiftAsm = new ShiftAsm(ShiftAsm.Op.SLL, ((MDAsm) cur).getTarget(),
((MDAsm) cur).getRs(),j);
        instructions.add(shiftAsm);
        //skip li
        instructions.remove(pre);
        //skip mflo
```

```
            i++;
            break;
        }
    }
```

2.2 优化乘普通常数

```
//乘其他常数
if (powerFlag == 0) {
    if (imm == 0) {
        MoveAsm moveAsm = new MoveAsm(((MDAsm) cur).getTarget(),Register.ZERO);
        instructions.add(moveAsm);
        //skip mflo
        i++;
    }
    else if (imm == 1) {
        //skip li
        instructions.remove(pre);
        //skip cur & mflo
        i++;
    }
     //imm <= 5优化才有意义
    else if (imm <= 5) {
        int k;
        for (k = 0; k < imm-1; k++) {
            AluAsm aluAsm;
            if (((MDAsm) cur).getTarget() != ((MDAsm) cur).getRs()) {
                if (k == 0) {
                    aluAsm = new AluAsm(AluAsm.Op.ADDU,((MDAsm) cur).getTarget(),
((MDAsm) cur).getRs(),((MDAsm) cur).getRs());
                }
                else {
                    aluAsm = new AluAsm(AluAsm.Op.ADDU,((MDAsm) cur).getTarget(),
((MDAsm) cur).getTarget(),((MDAsm) cur).getRs());
                }
            }
            else {
                 if (k == 0) {
                    aluAsm = new AluAsm(AluAsm.Op.ADDU,Register.T2,((MDAsm)
cur).getRs(),((MDAsm) cur).getRs());
                }
                else if (k < imm - 2){
                    aluAsm = new AluAsm(AluAsm.Op.ADDU,Register.T2,Register.T2,
((MDAsm) cur).getRs());
                }
                else {
                    aluAsm = new AluAsm(AluAsm.Op.ADDU,((MDAsm)
cur).getTarget(),Register.T2,((MDAsm) cur).getRs());
                }
            }
            instructions.add(aluAsm);
```

```
        }
        //skip li
        instructions.remove(pre);
        //skip mflo
        i++;
```

2.3 优化除以 2 的幂次

```
//2的幂次
int powerFlag = 0;
int j;
for (j = 1; j < 32; j++) {
     int power = (int) Math.pow(2,j);
     //imm = 2^j
     if (imm == power) {
         powerFlag = 1;
         ShiftAsm shiftAsm = new ShiftAsm(ShiftAsm.Op.SRL, ((MDAsm)
cur).getTarget(), ((MDAsm) cur).getRs(),j);
         instructions.add(shiftAsm);
         //skip li
         instructions.remove(pre);
         //skip mflo
         i++;
         break;
         }
 }
```