実践情報セキュリティとアルゴリズム 第二回課題

28G23027 川原尚己

実装内容：

eval.rs において実装した部分を図 1 に示す.

```rust
fn eval_arith(ctx: &mut Context, opcode: &ArithOpcode, reg1: &Register, reg2: &Register, reg3: &Register) {
    let val2: u64 = ctx.get_reg(reg2);
    let val3: u64 = ctx.get_reg(reg3);
    match opcode{
        ArithOpcode::Add => {
            ctx.set_reg(r: reg1, val: val2 + val3);
        }
        ArithOpcode::Sub => {
            ctx.set_reg(r: reg1, val: val2 - val3);
        }
        ArithOpcode::Mul => {
            ctx.set_reg(r: reg1, val: val2 * val3);
        }
        ArithOpcode::Div => {
            ctx.set_reg(r: reg1, val: val2 / val3);
        }
    }
}
```

```rust
Op::Arith(opcode: &ArithOpcode, reg1: &Register, reg2: &Register, reg3: &Register) => {
    eval_arith(&mut ctx, opcode, reg1, reg2, reg3)
}
```

図 1 - 1　eval.rs 実装部分（eval_arith）

```rust
fn eval_cmp(ctx: &mut Context, reg1: &Register, reg2: &Register) {
    let val1: u64 = ctx.get_reg(reg1);
    let val2: u64 = ctx.get_reg(reg2);
    if val1 == val2{
        ctx.cond = Condition::Eq
    }
    if val1 < val2{
        ctx.cond = Condition::Lt
    }
    if val1 > val2{
        ctx.cond = Condition::Gt
    }
}
```

図 1 - 2　eval.rs 実装部分（eval_cmp）

eval_arith に関しては，2,3 番目のレジスタに格納されている値を読み込んだ後，各種四則演算を 1 番目のレジスタに書き込んでいる.

eval_cmp に関しても，２つのレジスタに格納されている値を読み込んだ後，それらの大小

関係によって ctx.cond に適切な値を代入している．

example/ex1.S,ex2.S に対する実行結果を図 2 に示す．

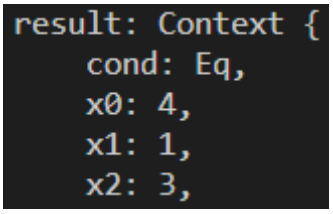

図 2 - 1　ex1.S に対する実行結果

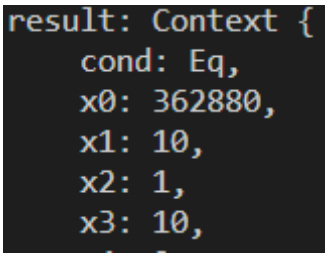

図 2 - 2　ex2.S に対する実行結果

いずれの場合でも想定されたとおりに動作していることがわかる．

また，自身で用意したテストコード ex3.S を図 3 に，その実行結果を図 4 に示す．

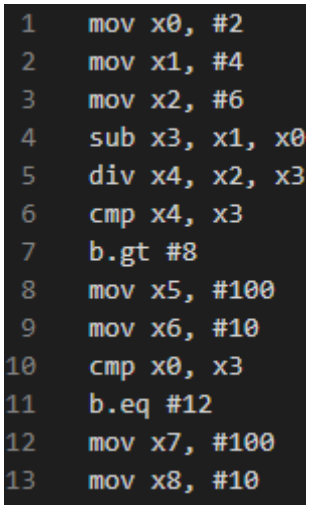

図 3　ex3.S

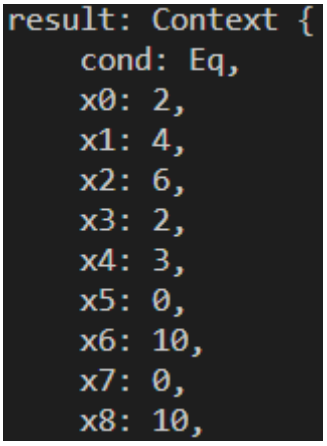

図 4　ex3.S に対する実行結果

ex1.S 及び ex2.S においては，eval_arith，eval_cmp で実装した部分に関することのうち，”add”，”mul”，”b.lt”について動作を確認したため，ex3.S においては残りの”sub”，”div”，”b.gt”，”b.eq”について確認を行った．
実行の結果，いずれのコマンドについても想定の通りに動作していることが確認できた．