Nomes: Matheus Tagliari Becker, Rômulo Pedro Thomsen, João Heitor Zabel da Rocha
Disciplina: Organização de Processadores
Semestre: 4°
Professor: Thiago Felski Pereira
Academia: Universidade do Vale do Itajaí
Cidade: Itajaí

RELATÓRIO CÓDIGO DE INSTRUÇÕES

08/12/2022

O trabalho a ser realizado pelos integrantes das equipes consistem em implementar uma solução que garanta a execução segura de sequências de código assembly de MIPS, evitando a execução de hazard codes (instruções sequenciais que podem atualizar registradores de forma que instruções posteriores possam pegar valores desatualizados em relação aos anteriores no MIPS Pipeline).

O código realizado a seguir feito em PHP, é uma tentativa de solucionar o problema.

```php
$nop = '00000000000000000000000000000000';
$nop_instructions = [];
$linecount = count(file("data.txt"));
$linesread = 0;
```

Nesta etapa acima tem-se a definição de uma instrução NOP (sequência de 32 caracteres nulos “0”), a importação do arquivo que contém o caso teste, contador de linhas percorridas e instruções.

```php
echo $linecount . " instruções.<br>";
if ($file = fopen("data.txt", "r")) {
    while(!feof($file)) {
        $linesread++;
        $line = fgets($file);
        array_push($nop_instructions, $line);
        echo $line . "<br>";
        $type = identify($line);
        if ($linesread == $linecount) {
            break;
        }
        if ($type == "R" or $type == "lw" or $type == "I") {
            array_push($nop_instructions, $nop);
            array_push($nop_instructions, $nop);
        }
        if ($type == "others" or $type == "sw" or $type == "branch") {
            array_push($nop_instructions, $nop);
        }
    }
    fclose($file);
    echo "<br><br>INSTRUÇÕES COM NOPS<br><br>";
}


foreach($nop_instructions as $lin){
    echo $lin . "<br>";
}
```

Nesta etapa o código abre o arquivo e separa todas as instruções por linha, então é vital que o código esteja com cada instrução separada pelas devidas quebras de linha. O código então executa a função de identificação (será explicado futuramente) e tenta estimar quantas instruções *nop* precisam ser adicionadas ao código, o código então mostra a quantia de linhas do código original, o código original em si e o código com os as instruções *nop* na tela.

```
function identify($line) {
    if ($line == "00000000000000000000000000000000") {
        return "nop";
    }
    $opcode = substr($line, 0, 6);
    if ($opcode == "000000"){
        if (substr($line, 26, 6) == "001100") {
            return "syscall";
        }
        else {
            return "R";
        }
    }
    else if ($opcode == "000010") {
        return "lw";
    }
    else if ($opcode == "101011") {
        return "sw";
    }
    else if ($opcode == "000100" or $opcode == "000101") {
        return "branch";
    }
    else if ($opcode == "000010" or $opcode == "000011" or $opcode == "001000") {
        return "J";
    }
    else {
        return "I";
    }
}
```

O código de identificação inicia checando se a linha já é um *nop*, e realiza nada caso seja, então já pula para a checagem de instruções do tipo R, que inicia checando novamente se o *funct* indica ser um *syscall* ao invés de uma devida operação R. Caso esse não seja o caso, o código então checa as demais instruções, se não checar nenhuma colisão a mais, o código retorna que a instrução é do tipo I.

Exemplo de código rodando: