Project2 Report

Name: Chen Huaxun

ID: 120090440

Introduction

This project is a MIPS simulator. The simulator can read data segment from `.asm` file, read machine codes from `.txt` file. It can simulate a memory of 6MB and implement machine codes to give the result in `.out` file.

Usage

Plain Text

```
python simulator.py name.asm name.txt name_checkpts.txt name.in name.out
```

Basic Idea

---

step1 Allocate memory: 6MB in total, 1MB for text segment, start form 0x400000 up to 0xA00000

step2 Create register dictionary to save register

step3 Write functions to save data segment into static memory

step4 Write functions to identify machine code: ***Idea****, no matter what it is, we all try to get rs, rt ,rd, sa, immediate, target*

step5.1 Write functions to implement the instructions

step5.2 and syscall

step6 Write functions to output register.bin & memory.bin file

Memory

Use List data structure to simulate a 6MB memory, 4 byte each for each position.

That is 1572864 position in Decimal

Memory address	Content

0xA00000 -> 0x500000	Stack
0x500000 -> 0xA00000	Data segment
0x400000 -> 0x500000	Text segment

Python

```python
memoryList = [0] * 1572864
static_data = 262144
pc = 0
hi = 0
lo = 0
```

Register

Initial position of `$gp` is 0x508000

Initial position of `$sp` is 0xA00000

Initial position of `$fp` is 0xA00000

Python

```python
regValue = {
    '00000': 0, # $zero
    '00001': 0, # $at
    '00010': 0, # $v0
    '00011': 0, # $v1
    '00100': 0, # $a0
    '00101': 0, # $a1
    '00110': 0, # $a2
    '00111': 0, # $a3
    '01000': 0, # $t0
    '01001': 0, # $t1
    '01010': 0, # $t2
    '01011': 0, # $t3
    '01100': 0, # $t4
    '01101': 0, # $t5
    '01110': 0, # $t6
    '01111': 0, # $t7
    '10000': 0, # $s0
    '10001': 0, # $s1
    '10010': 0, # $s2
    '10011': 0, # $s3
    '10100': 0, # $s4
    '10101': 0, # $s5
    '10110': 0, # $s6
    '10111': 0, # $s7
    '11000': 0, # $t8
    '11001': 0, # $t9
    '11010': 0, # $k0
    '11011': 0, # $k1
    '11100': 5275648, # $gp
    '11101': 10485760, # $sp
    '11110': 10485760, # $fp
    '11111': 0 # $ra
}
```

Static Data

Take out the data segment from `.asm` file, remove comments and labels, save each type of data into the address of static data in **Big Endian** format.

Identify Machine Code

Write `findType(code)` function, which returns the MIPS instruction of that line machine code.

Read `rs, rt, rd, sa, imm, target` from machine code.

Python

```python
def findType(code):
    op = code[0:6]
    if op == '000000':
        func = code[-6:]
        instru = funct[func]
    else:
        if op == '000001':
            op = op + 'x' * 5 + code[11:16]
        instru = opCode[op]
    return instru


cur = memoryList[pc]
pc += 1
rs = cur[6:11]
rt = cur[11:16]
rd = cur[16:21]
sa = cur[21:26]
imm = cur[-16:]
target = cur[-26:] + '00'
imm = int(imm[0], 2) * (-32768) + int(imm[1:], 2)
target = int(target, 2)
instruction = findType(cur)
```

Implment

One function maps to one `if` or `elif`

Python

```python
def add(rd, rs, rt):
    rd = rs + rt
    return rd


if instruction == 'add':
    regValue[rd] = add(regValue[rd], regValue[rs], regValue[rt])
```

Syscall

`Import os` to implement file open, write, read, close

Use a bool type `Exit` to control whether stop or not

Dump

`Write_bin1` to output `register.bin`

`Write_bin2` to output `memory.bin`

Python

```python
def write_bin1(name, pc, lo, hi):
    with open(name, 'wb') as f:
        for value in regValue.values():
            bin_data = int(value).to_bytes(4, 'little')
            f.write(bin_data)
        pc = int(pc) * 4 + 4194304
        f.write(pc.to_bytes(4, 'little'))
        f.write(int(lo).to_bytes(4, 'little'))
        f.write(int(hi).to_bytes(4, 'little'))


def write_bin2(name):
    with open(name, 'wb') as f:
        for value in memoryList:
            if isinstance(value, str):
                bin_data = int(value, 2).to_bytes(4, 'little')
                f.write(bin_data)
            else:
                bin_data = int(value).to_bytes(4, 'little')
                f.write(bin_data)
```