Project 1: Optimizing the Performance of a Pipelined Processor

000, , bugenzhao@sjtu.edu.cn
001, , doctormin@sjtu.edu.cn

May 2, 2020

1 Introduction

Part A
In part A, we write three simple assembly programs to mimic three functions in example.c. Based on ensuring correctnesswe especially focus on the functional equivalence with the example C functions. By selecting and placing labels in the assembly code appropriately, the code is also very readable.
Part B
In part B, we modify the HCL file of the SEQ to add a new instruction — iaddl. The following is the roadmap to finish this part:

- Clarify the computation process of iadd and write it down at the beginning in seq-full.hcl.
- Add any dependence relations of iaddl to all boosigs.
- Design the datapath for iaddl (generate control signals for src and dst)

Part C
We achieve full scores in the benchmark testing **in just 2 hours**, but we **spent 2 more days** researching all the potential methods to optimize the performance even further. The following is our roadmap:

- Change the order of the instruction sequence to avoid data hazard and structure hazards, which leaves $CPI = 12.96$.
- Beyond the changes on instructions order, we use loop unrolling to reduce the number of conditional check and registers updating, which leaves $CPI = 9.83$
- Use a binary search tree to find the precise remaining number of loops after several rounds of unrolling to achieve complete unrolling, which leaves $CPI = 8.95$

- Modify the HCL file to achieve 100% accuracy in branch prediction for certain code pattern, which brings CPI down to 7.78.

Contribution
Ziqi Zhao : Part A (coding) & Part B (coding) & Part C (coding & designing)
Yimin Zhao : Part A (reviewing) & Part B (reviewing) & Part C (designing) & project report

2 Experiments

2.1 Part A

2.1.1 Analysis

In this part, we are asked to implement and simulate three y86 programs. From a macro point of viewthis part is relatively easy. But there are plenty of optimizations worth exploring in terms of code readability and elegance.
Difficult Point

- Be aware of the state of the stack like calling a function will push the ret address first.
- Be careful to protect the callee-save register
- Implement function recursion smartly

[In this part, you should give an overall analysis for the task, like difficult point, core technique and so on.]

2.1.2 Code

sum.ys

```
# 518030910211 ZiqiZhao
# 518030910188 YiminZhao

# Set up stack
        .pos    0
        irmovl  stack,  %esp
        rrmovl  %esp,   %ebp
        pushl   %edx            # save %edx
        irmovl  ele1,   %eax
        pushl   %eax
        call    sum_list
        popl    %edx            # flatten the stack for ele1
        popl    %edx            # restore %edx
        halt
```

```
# Sample linked list
.align 4
ele1:
        .long   0x00a
        .long   ele2
ele2:
        .long   0x0b0
        .long   ele3
ele3:
        .long   0xc00
        .long   0

# sum_list func
sum_list:
        pushl   %ebp            # enter
        pushl   %ecx            # save %ecx
        rrmovl  %esp,   %ebp
        xorl    %eax,   %eax    # clear %eax
        mrmovl  12(%ebp),%edx   # get ls
        jmp     test
loop:
        mrmovl  (%edx), %ecx
        addl    %ecx,   %eax
        mrmovl  4(%edx),%edx
test:
        andl    %edx,   %edx
        jne     loop            # %edx != 0
return:
        rrmovl  %ebp,   %esp    # leave
        popl    %ecx
        popl    %ebp
        ret

# Stack
        .pos    0x400
stack:
```

2.1.3 Evaluation

[In this part, you should place the figures of experiments for your codes, prove the correctness and validate the performance with your own words for each figures explanation.]

2.2 Part B

2.2.1 Analysis

[In this part, you should give an overall analysis for the task, like difficult point, core technique and so on.]

2.2.2 Code

[In this part, you should place your code and make it readable in Latex, please. Writing necessary comments for codes is a good habit.]

2.2.3 Evaluation

[In this part, you should place the figures of experiments for your codes, prove the correctness and validate the performance with your own words for each figures explanation.]

2.3 Part C

2.3.1 Analysis

[In this part, you should give an overall analysis for the task, like difficult point, core technique and so on.]

2.3.2 Code

[In this part, you should place your code and make it readable in Microsoft Word, please. Writing necessary comments for codes is a good habit.]

2.3.3 Evaluation

[In this part, you should place the figures of experiments for your codes, prove the correctness and validate the performance with your own words for each figures explanation.]

3 Conclusion

3.1 Problems

[In this part you can list the obstacles you met during the project, and better add how you overcome them if you have made it.]

3.2 Achievements

[In this part you can list the strength of your project solution, like the performance improvement, coding readability, partner cooperation and so on. You can also write what you have learned if you like.]