Program Analysis and Transformation

Search and Reduce Redundant Patterns in ROOPLPPC

Benjamin Brandt Ohrt, zpn492

June 17, 2019

Abstract

The Pendulum Instruction Set Architecture (PISA) was first introduced in 1995 by Carlin James Vieri and is a reversible assembly language. The assembly language has later been improved and several high-level languages has been build upon it. One of those languages is the extension to the reversible object-oriented programming language (ROOPL) ROOPL++ presented in 2018. One of the issues with ROOPL++ was the amount of produced target code. In this paper we compile source code for a small program counting from 1..100, analyse the possibility of redundant target code and points out where an optimization could be done.

Contents

1 Introduction

1.1 Assembly language

To communicate with the hardware of a computer, one should speak its language. Each Central Processor Unit (CPU) understands its own machine language. This language is known as instructions, stored as bytes in memory. Each instruction has its own unique numeric code called its operation code or opcode for short. A set of instructions is the vocabulary of the language [11, p.76]. An example in MIPS Assembly Language [11] is the instruction that says add $s1(register:17) and $s2(register:18) registers together and store the result into $t0 (register:8), which is encoded by the following hex code:

0232 4020

This not readable for the blunt eye which is why we have a program called an assembler. An assembler is a program that reads the text of an assembly language program and converts the assembly into machine code [6, p.11].

The assembly language origins back to Birkbeck College 1946-1962, where the creation was credited to Kathleen Booth [1]. Kathleen Booth was a PhD whom both visited with John Von Neumann with the famous The Von Neumann Architecture [3], helped with the design of multiple machines and was one of the founders of the Birkbeck department of computer science.

The assembly language is a symbolic programming language, closest to machine code [2]. An assembly language program is stored as plain text. The text consists of a set of instructions which is processed chronological. Each assembly instruction is equal to one machine instruction for example could above addition be shown as:

add $t0, $s1, $s2

1.1.1 Example program

ForITo100.asm

```
J main                          ; for int i = 0; i < 100; i++
                                ;     count += i
loop_i:
ADD $a0, $a0, $zero             ; count += i
ADDI $a1, $a1, 1                ; i++
SLT $t0, $a1, $a2
BEQ $t0, 1, loop_i              ; i < 100
JR $ra                          ; return

main:
ADDI $a0, $zero, 0              ; count = 0
ADDI $a1, $zero, 0              ; i = 0
ADDI $a2, $zero, 100            ; max
JAL loop_i
```

1.2 Reversible computing

A reversible computing system has, at any time, at most a single previous computation state as well as a single next computation state, and thus a reversible computing system [7]. It comes with the promise of reduced energy dissipation, when erasure of information is left out of the program.

The inspiration of reversible computing dates back to 1867 and the study of thermodynamics where James Clark Maxwell made a thought experiment also known as Maxwell's Demon. In the experiment he questions that of the second law of thermodynamics which says:

> [..] It is impossible in a system enclosed in an envelope which permits neither change of volume nor passage of heat, and in which both the temperature and the pressure are everywhere the same, to produce any inequality of temperature or pressure without expenditure of work [8, p.16].

The experiment was later published in 1872 in a book by Maxwell: Theory of Heat [12], while the connection between thermodynamics and computations was first seen in 1949 by John von Neumann. It was in one of his lectures, where he spoke about computations, and the must of a minimum thermodynamic energy dissipation. A minimum which he determines to [5, p.20][8, p.18]:

$k_B T ln N$, k_B as the Boltzmann's constant, T as the temperature and $N = 2$.

Rolf Landauer realize that modern computers with irreversible processes must dissipate that minimum energy as von Neumann described if the erasure of a bit occurs thus resulting in at least the increase of entropy by $k_B ln 2$ [5, p.20][8, p.18]. He also states that reversible operations does not produce the dissipation and irreversible operations can avoid the dissipation by storing information of the computational history. At last Landauer states that the stored history must be erased irreversibly which would just postpone the dissipation of entropy, That last statement is later proven wrong by Charles Bennett [8, p.19].

In 1970 Charles Bennett looks at Landauers last statement and decides to make an experiment. He creates a reversible program consisting of two halves. The first which provides the calculations intended and the second which undid the calculations from the first halves thus ending the program in its starting point. In the experiment he uses a Turing Machine[10] and shows that the information produced by the intended calculation could be used to undo the calculation instead of just be thrown away. With that experiment Bennett had shown that a reversible process would not just postpone the dissipation of entropy it would be able to remove it from the equation [8, p.18].

From Bennetts experiment he came up with a description for an "enzymatic Turing machine" in which reversible logically operations could be executed [8, p.19]. To fullfill the promise of reduced energy dissipation their was a final need of physically logic devices to perform the reversible operations on. Fredkin, Toffoli and Feynmann all aided this need by introducing reversibly logic-gates [8, p.19,22].

1.3 Pendulum microprocessor

Pendulum is a reversible microprocessor, invented by Carlin James Vieri at MIT in 1995. The invention took offset in the existing MIPS R2000 architecture [11][4, p.29]. Vieri's motivation was to create a reversibly processor which would avoid destruction of information and reduce the energy dissipation shown in thermodynamics[1.2] and discussed by Charles Bennet [8]. For this he would focus on memory access, datapath operations on stored values and control flow operations.

The processor has three registers used in the control flow:

1 The program counter (PC) for storing address of the current instruction

2 The branch register (BR) for storing jump offsets

3 The direction bit (DIR) for keeping track of the execution direction.

PC is incremented/decremented by the value stored in DIR, thus changing DIR between 1 & -1, will decide in which direction the instructions will be executed. When the DIR is -1, all instructions is inverted. [4, p.21]

Branch instructions like branch or jump is in normal architectures not reversible, that is because the come-from instruction is not stored anywhere, thus the goto instruction not knowing who called it. This could be handled by storing the PC just before the branch/jump in a special register. In later versions of the Pendulum ISA architecture paired branches is introduced, such that each goto instruction should have branch instruction to the come-from.

If/then statements is in need of an exit condition known as assert.

if e1 then s1 else s2 fi e2

For-loop is in need of an entry, loop and exit condition. Where the e1 should only be true upon entry and e2 only on exit.

from e1 do s1 loop s2 until e2

Subroutines in encased between a top and a bottom, which both contains a branch to each other. It follows that the subroutine is skipped when executing instructions sequentially.

top: BRA bot

<..routine definition goes here..>

bot: BRA top

Memory access is always an exchange. The exchange instruction swaps register value with the value in memory at an address specified by another register [4, p.32].

1.4 Motivation

First of the professor Robert Glück, which gave the inspiration and idea by introducing ROOPL and the two masters thesis of Tue Haulund and Martin Holm Cservenka. Next the motivation for this paper comes from the evaluation in the Masters Thesis of Martin Holm Cservenka [14, p.72], in which he shows the blow up of a compiled ROOPL++ program. From this point I couldn't leave the thought, that maybe the compiler left possible redundant patterns, in which one could reduce or simply remove. With tools like PendVM and the ROOPL compiler at hand the task was straight forward: Write a simple program first directly in the Pendulum Assembly Language and then compare it to a similar compiled ROOPL++ program, find if any unnecessary redundant patterns and reduce/remove them from the program.

1.5 Statement

A separate program which takes the target code of compiled ROOPL++ program, reduce some patterns which is redundant and returns a new file of target code without those patterns.

1.6 Outline

This paper consists of four chapters:

Chapter 1 is the introductory chapter and gives a short introduction to the history of assembly and reversible computing.

Chapter 2 presents two programming languages The Pendulum Assembly Language and ROOPL. A small program is introduced in both languages and compared at the assembly level.

Chapter 3 analyze patterns within the assembly language and search for redundancy.

Chapter 4 evaluates the results from the analysis and presents a conclusion and suggestions for further work.

2 Programming languages

2.1 Pendulum Assembly Language

Pendulum Instruction Set Architecture(PISA) Assembly Language or a shorthand PAL, is a reversible language. It assembles by the Pendulum microprocessor [4, p.48]. In this paper we use the Pendulum virtual machine PendVM [9], which executes programs written in PAL. Pendulum instructions is almost identically to the conventional RISC(Reduced Instruction Set Computing) processor. Thus reversible it introduces some new features.

Branch register is normally zero, if the branch register is not zero, the PC increments by the value in the register [5, p.278]. When this happens the instruction at the destination executes and the branch register is cleared. When implementing subroutines one must use SWAPBR which allows direct access to the branch register, and by that, store the value at the beginning and pop at the end of the routine. It must follow that the subroutine negates the value in the branch register, such that the next SWAPBR will branch back to the location it came from. The branch at the location cancels out the branch register and the PC continues sequentially.

2.1.1 Example program

ForITo100.pal

```
;; main
;;; increment a from [1..n]
subtop:     BRA subbot
main:       SWAPBR $2                ; entry/exit point
            NEG $2                   ; negate offset to return caller
            EXCH $2 $1               ; push return offset to stack
            ADDI $30 100             ; a += n
            BRA swap                 ; set limit ($28) += a, set a = 0
looptop:    BNE $30 $0 loopbot       ; from a = 0 do
            ADDI $30 1               ; a += 1
loopbot:    BNE $30 $28 looptop      ; until a = limit loop body
            SUB $28 $30              ; set limit = 0
            ADDI $30 -100            ; a -= n
            EXCH $2 $1               ; pop return address
subbot:     BRA subtop
```

The entire program is shown in the Appendix 7. We have in section 1.1.1 seen how a similar for-loop is written in the irreversible assembly language MIPS. Now let us take a look on the for-loop in the reversible assembly language PAL.

First off the main method is wrapped within a top and a bot. Next we store the offset, so on the end of the method we are able to return to the caller. Now the looptop is checked once like from a = 0 then proceed to the body. From here on the body is executed once and once again until the criteria in loopbot is true.

2.2 Reversible Object-Oriented Programming

Reversible Object-Oriented Programming (ROOPL) and its successor ROOPL++ which expanded the language with dynamic memory was created by Tue Haulund and Martin Cservenka is build with inspiration from the reversible programming language Janus and compiles to the Pendulum Assembly language. The latest extension of the compiler is accessible at github [13].

simplePrg.rplpp

```
class Program
    int nodeCount
    int limit
    method main()
        limit += 100
        from nodeCount = 0 do
            skip
        loop
            nodeCount += 1
        until nodeCount = limit
        nodeCount -= limit
        limit -= 100
```

This simple program is similar to the two earlier shown assembly programs. Though when compiled the number of instructions explodes. In Appendix 8 the full list of instructions is shown, it sums up to near 400 lines of program.

The program could be divided into parts:

Part	Lines	Description
Static	4	DATA instruction
Malloc	205	Method for memory allocation
Main	134	Main method equal two the other assembly programs
Program structure	54	START .. FINISH

With the above division, its clear that even without the malloc method, the compiled program is still way bigger than the simple self-written PAL program.

A closer look on the main method shows that 26 lines holds a bunch of if-statements with named macros as **cmp** og **f**. The entire entry macro counts 57 lines including the special if-statements. The initialization and clean-up is 20 and 35 lines, where the loop-body is 14 lines.

Part	Lines
1_ main_ 0 to entry	20
entry to test	57
test	14
after exit	35

In the next chapter the focus is on the main method and the program structure. We look into the possibility of redundancy and search for patterns which is unnecessary for the program.

3 Analysis of redundancy

In this is section we analyze the possibility of redundant target code after compilation of the ROOPL++ program introduced in section 2.2. The analysis is split into two parts.

The first part will be a stepwise approach going through the following states:

Discover patterns which is repeated multiple times throughout the target code

Isolate the patterns into subroutines to make the target code more readable

Group subroutines by purpose

Try to reduce the groups, for example by introducing new instructions

The second part describes methods for reducing redundancy by eliminating or replacing redundant patterns found in the first part.

3.1 First part - patterns

This is the first part of the analysis, where we try to identify patterns which is occurring multiple times throughout the target code.

3.1.1 Getting data from memory

Multiple times we'll see a pattern for exchange between a register value and a place in memory. We look into an example starting between entry and the first **cmp**.

The pattern goes like this: Put the address of nodeCount into register 7. Use the address in register 7 to exchange the nodeCount in memory with the zero value in register 8. Register 8 now holds the value of nodeCount. This part is reached several times, once for each time nodeCount is not 100. This exchange is placed 6 times in the code.

We isolate the pattern by introducing a subroutine count. Count can replace the 6 redundant patterns:

```
subtop: BRA     subbot
count:  SWAPBR  $2
        NEG     $2
        EXCH    $2  $1
        ADD     $7  $3
        ADDI    $7  2
        EXCH    $8  $7
        ADDI    $7  −2
        SUB     $7  $3
        EXCH    $2  $1
subbot: BRA     subtop
```

We see that the exchange part takes up 5 lines, where the subroutine takes 10 +1 for the call. Exchanges done more than 2 times could line wise be subroutines. Another benefit by isolating exchanges is the readability in the actual main method.

This pattern is not only seen when retrieving the nodeCount, it is seen every time memory and a register has to communicate. This exchange is the same as changing the register value in $r1 with the value in memory of the stack pointer + some offset. One could formalize a method m with three arguments: m(r1, sp, offset). Why this could by a place to introduce a new instruction.

3.1.2 Special if-statements

Throughout the target code, we seen multiple auto-generated if-statements, with labels like **cmp** and **f**. Lets start by isolating the areas with special if-statements.

```
subtop_if_1:        BRA     subbot_if_1
if_1:               SWAPBR  $2
                    NEG     $2
                    EXCH    $2 $1
cmp_top_15:         BNE     $8 $0 cmp_bot_16
                    XORI    $9 1
cmp_bot_16:         BNE     $8 $0 cmp_top_15
f_top_17:           BEQ     $9 $0 f_bot_18
                    XORI    $10 1
f_bot_18:           BEQ     $9 $0 f_top_17
                    XOR     $6 $10
f_bot_18_i:         BEQ     $9 $0 f_top_17_i
                    XORI    $10 1
f_top_17_i:         BEQ     $9 $0 f_bot_18_i
cmp_bot_16_i:       BNE     $8 $0 cmp_top_15_i
                    XORI    $9 1
cmp_top_15_i:       BNE     $8 $0 cmp_bot_16_i
                    EXCH    $2 $1
subbot_if_1:        BRA     subtop_if_1
```

At first glance we see that the purpose is to set \$6 at one or zero. Simple if-statement could possibly replace the above both complicated and hard-to-read set of instructions. Let's try to figurer out the parts. Let's say if 8 = 0, then XOR \$6 by \$10, where \$10 is the same as 1. This could be written as below.

```
iftop_3:            BNE     $8 $0 ifbot_3
                    XORI    $6 1
ifbot_3:            BNE     $8 $0 iftop_3
```

If this holds, we could, when approaching the part where the special if-statement occurs, take the first line and let it be the condition, find the XOR which is not encapsulated within a label, exchange the XOR with XORI and exchange the second register value with a 1.

3.2 Second part - methods for reduced redundancy

In this section we upgrade PendVM [9] the virtual machine on which we can run PAL programs. We further introduce some new algorithms to reduce the target code of a compiled ROOPL++ program.

Main algorithm of the section is remove_ redundancy which we'll later apply to sample programs for testing.

3.2.1 PendVM - OUT

PendVM does not support methods to save the results of an executed program, which is why i have expanded one of the instructions: OUT.

OUT is now able to write or append a register value to the text-file static_ storage.txt. For now ROOPL has not been expanded which means, that using the OUT functionality requires three steps:

The immediate value in ADDI just after START has to incremented by the value of need OUT instructions. (update stack pointer)

The immediate value in ADDI just before FINISH has to incremented by the value of need OUT instructions. (update stack pointer)

OUT instruction has to be manually inserted into the resulting .pal file of a compiled ROOPL++ program.

The expansion of PendVM exists of the following files:

pendvm.h

```
int i_out(WORD, WORD, WORD);
```

pal_parse.c

```
("OUT", {REG, REG, NIL}, i_out)
```

machine.c

```
int
i_out(WORD r, WORD u1, WORD u2)
  {
  char buf[..];
  sprintf(buf, '%d', m->reg[r1]);
  const char *p = buf;

  FILE *file;
  file = fopen('static_storage.txt', m->reg[r2]);
  fputs(p, file);
  fclose(file);
  return 0;
  }
```

3.2.2 PendVM - EXCHI

Now what we have seen is that the exchange of memory and register values is a 5 line procedure, occurring multiple times within the target code of a ROOPL++ program. Which is why i want to introduce a new instruction called EXCHI, it takes the 3 arguments, two registers and an immediate. First register will get its value swapped with the value in the memory of position of the second register + immediate. The expansion of PendVM exists of the following files:

pendvm.h

```
int i_exchi(WORD, WORD, WORD);
```

pal_parse.c

```
("EXCHI", {REG, REG, IMM}, i_exchi)
```

machine.c

```
int
i_exchi(WORD rd, WORD ra, WORD u1)
  {
  WORD tmp;
  /* ra is the stack pointer
   * u1 is the offset */
  MEMORY *loc = mem_get(m->reg[ra]+u1);

  if (loc->type == MEM_INST) {
    pendvm_error("..");
    return -4; /* "exchange with instruction" error */
  }

  tmp=m->reg[rd];
  m->reg[rd]=loc->value;
  loc->value=tmp;
  return 0;
  }
```

3.2.3 Algorithm - Memory exchange

```
Find the pattern:
----------------------------------------
ADD        $r1  $r2
ADDI       $r1  imm
EXCH       $r3  $r1
ADDI       $r1  -imm
SUB        $r1  $r2
----------------------------------------
Replace it with:
----------------------------------------
EXCHI      $r3  $r2  imm
```

3.2.4 Algorithm - Special if-statements

```
Find the pattern:
----------------------------------------
cmp_top_x    inst_x
cmp_bot_x    inst_x
<...>
XOR          $r1  $r2
<...>
cmp_bot_x_i  inst_x_i
cmp_top_x_i  inst_x_i
----------------------------------------
Replace it with:
----------------------------------------
if_top_x:    inst_x
XORI         $r1  1
if_bot_x:    inst x
```

3.2.5 Algorithm - Remove redundancy

For the task of a successful analysis we need an algorithm in which we can find redundancy, merge and remove instruction and at last update the stack point with the new immediate value. This algorithm can be applied to an existing PAL program, which means, that it is first useful, when a ROOPL++ program has been compiled to target code. Thus the algorithm must take a .PAL file and produce a .PAL file.

This algorithm will open a .pal file copy its content and run the algorithms memory_ exchange and special if-statements. The end result is stored in the file: reduced.pal.

```
int main(int arg, char *args[])
    {
    filehandler.input_file=fopen(args[1], "r");
    if(!filehandler.input_file) {
        printf("%s: Unable to open input file %s.\n","",args[1]);
        exit(1);
    }
    fclose(filehandler.input_file);

    /* memory exchange */
    apply(1, args[1]);
    /* special if-statements */
    apply(2, "reduced.pal");

    return 0;
    }

int apply(int method, const char *file) {
    int action;
    initialize(file);

    if(method == 1) {
        /* loop to load instruction memory */
        while(getinst()) {
            /* check for memory exchange pattern */
            action = memory_exchange();

            if(action == 0)
                fputs(buffer.line, filehandler.output_file);
        }
    } else {
        while(getinst()) {
            /* check for special if-statements */
            action = special_if_statements();

            if(action == 0)
                fputs(buffer.line, filehandler.output_file);
        }
    }
    fclose(filehandler.input_file);
    fclose(filehandler.output_file);
}
```

The program remove_ redundancy is implemented in C and compiles with the GCC 8.2.1 20181127.

GCC remove_ redundancy.c

After compilation it's runable on .pal files.

./a.out <filename>.pal

The program creates two files. First the tmp.pal which is used to hold states within the program, the second is reduced.pal, which is the reduced program of <filename> .pal. The program is divide into two states, first is when we search for patterns of memory exchange, second is when we search for patterns of special if-statements.

The full sourcecode is available in the appendix 10.

4 Evaluation

In this section we evaluate results of the described algorithm remove_ redundancy and discuss possible scenarios where the algorithm won't apply. The produced program p' of the algorithm is tested against the original program p created by the ROOPL++ compiler and a similar handwritten program. The focus is on the instruction count and PendVM steps. To prove that the programs produce similar results, the output 1..100 is stored in a .txt file.

Program	PendVM step	main lines	start lines	Description
p	6572	134	50	Original program produced by the ROOPL++ compiler
p'	2102	50	42	A reduced version of p, where the algorithm remove_ redundancy has been applied
p"	1328	41	5	The example PAL program from the section of Pendulum Assembly Language: ForITo100.pal

On this simple program counting from 1..100, we have reduced the needed steps in PendVM to one third of the original needed steps. Furthermore the number of lines have been reduced which should be reflected on the readability.

4.1 Scenarios which is not supported

The pattern below is seen in Martin Cservenka's program BinaryTree.pal. It's not supported by the algorithm remove_ redundancy.

```
Find the pattern:
-----------------------------------
cmp_top_x    inst_x
cmp_bot_x    inst_x
<...>
EXCH         $rx $ry
<...>
ANDX         $rz $rw
<...>
XOR          $r1 $r2
<...>
cmp_bot_x_i  inst_x_i
cmp_top_x_i  inst_x_i
-----------------------------------
```

We have also seen that DATA instruction as well as jmp macros which refer to a specific line has to be updated after the production of p'. This is also a step which is missing in remove_ redundancy.

5 Conclusion

In this paper we searched for redundancy in the target code of a compiled ROOPL++ program. We have found patterns of redundancy regarding memory exchange and special if-statements. The paper shows an algorithm, which works on simple programs without DATA and jmp instructions. On those programs, the algorithm reduces the redundant patterns resulting in fewer lines of instruction as well as fewer computational steps within the virtual machine PendVM.

While the algorithm works great with smaller programs, running it on Cservenka's BinaryTree.pal or RTM.pal results in broken programs.

6 Further work

Getting closer to a complete algorithm, one should focus on updating line-numbers of DATA instructions and jmp labels, which has changed after patterns have been reduced.

Multiple places in the code, we'll see more than one ADDI instructions of the same register but with different immediate value. Those lines could be merged together.

Finally these reduction algorithms should be implemented within the ROOPL compiler, raising the chance, that future developments would result in updates of the algorithms.

7 Appendix A

ForITo100.pal

```
;;; increment a from [1..n]
subtop:    BRA subbot
main:      SWAPBR $2              ; entry/exit point
           NEG $2                 ; negate offset to return caller
           EXCH $2 $1             ; push return offset to stack
           ADDI $30 100           ; a += n
           BRA swap               ; set limit ($28) += a, set a = 0
looptop:   BNE $30 $0 loopbot     ; from a = 0 do
           ADDI $30 1             ; a += 1
loopbot:   BNE $30 $28 looptop    ; until a = limit loop body
           SUB $28 $30            ; set limit = 0
           ADDI $30 -100          ; a -= n
           EXCH $2 $1             ; pop return address
subbot:    BRA subtop
;; swap(int a, int b)
subtop_3:  BRA subbot_3
swap:      SWAPBR $2
           NEG $2
           EXCH $2 $1
           ADD $28 $30
           SUB $30 $28
           EXCH $2 $1
subbot_3:  BRA subtop_3
```

8 Appendix B

prg.pal

```
;; pendulum pal file
top:                     BRA    start
l_r_nodeCount:           DATA   0
l_r_limit:               DATA   0
l_Program_vt:            DATA   214
l_malloc_top:            BRA    l_malloc_bot
l_malloc:                SWAPBR $2
                         NEG    $2
                         ADDI   $9 2
                         XOR    $8 $0
                         ADDI   $1 1
                         EXCH   $6 $1
```

```
                         ADDI    $1  1
                         EXCH    $7  $1
                         EXCH    $2  $1
                         ADDI    $1  -1
                         BRA     l_malloc1
                         ADDI    $1  1
                         EXCH    $2  $1
                         EXCH    $7  $1
                         ADDI    $1  -1
                         EXCH    $6  $1
                         ADDI    $1  -1
                         XOR     $8  $0
                         ADDI    $9  -2
l_malloc_bot:            BRA     l_malloc_top
l_malloc1_top:           BRA     l_malloc1_bot
                         ADDI    $1  1
                         EXCH    $2  $1
                         SUB     $17  $8
                         XOR     $17  $4
l_malloc1:               SWAPBR  $2
                         NEG     $2
                         EXCH    $2  $1
                         ADDI    $1  -1
                         XOR     $17  $4
                         ADD     $17  $8
                         EXCH    $19  $17
                         XOR     $18  $19
                         EXCH    $19  $17
                         XOR     $13  $9
                         SUB     $13  $7
cmp_top_1:               BGEZ    $13  cmp_bot_2
                         XORI    $14  1
cmp_bot_2:               BGEZ    $13  cmp_top_1
                         XOR     $10  $14
cmp_bot_2_i:             BGEZ    $13  cmp_top_1_i
                         XORI    $14  1
cmp_top_1_i:             BGEZ    $13  cmp_bot_2_i
                         ADD     $13  $7
                         XOR     $13  $9
l_o_test:                BEQ     $10  $0  l_o_test_false
                         XORI    $10  1
                         ADDI    $8  1
                         EXCH    $19  $17
                         XOR     $18  $19
                         EXCH    $19  $17
```

```
RL      $9  1
EXCH    $10  $1
ADDI    $1  −1
EXCH    $11  $1
ADDI    $1  −1
EXCH    $12  $1
ADDI    $1  −1
EXCH    $14  $1
ADDI    $1  −1
EXCH    $16  $1
ADDI    $1  −1
EXCH    $17  $1
ADDI    $1  −1
EXCH    $18  $1
ADDI    $1  −1
EXCH    $20  $1
ADDI    $1  −1
EXCH    $21  $1
ADDI    $1  −1
EXCH    $22  $1
ADDI    $1  −1
EXCH    $23  $1
ADDI    $1  −1
BRA     l_malloc1
ADDI    $1  1
EXCH    $23  $1
ADDI    $1  1
EXCH    $22  $1
ADDI    $1  1
EXCH    $21  $1
ADDI    $1  1
EXCH    $20  $1
ADDI    $1  1
EXCH    $18  $1
ADDI    $1  1
EXCH    $17  $1
ADDI    $1  1
EXCH    $16  $1
ADDI    $1  1
EXCH    $14  $1
ADDI    $1  1
EXCH    $12  $1
ADDI    $1  1
EXCH    $11  $1
ADDI    $1  1
```

```
                        EXCH   $10 $1
                        RR     $9 1
                        ADDI   $8 -1
                        XORI   $10 1
l_o_assert_true:        BRA    l_o_assert
l_o_test_false:         BRA    l_o_test
cmp_top_5:              BEQ    $18 $0 cmp_bot_6
                        XORI   $20 1
cmp_bot_6:              BEQ    $18 $0 cmp_top_5
                        XOR    $11 $20
cmp_bot_6_i:            BEQ    $18 $0 cmp_top_5_i
                        XORI   $20 1
cmp_top_5_i:            BEQ    $18 $0 cmp_bot_6_i
l_i_test:               BEQ    $11 $0 l_i_test_false
                        XORI   $11 1
                        ADD    $6 $18
                        SUB    $18 $6
                        EXCH   $12 $6
                        EXCH   $12 $17
                        XOR    $12 $6
                        XORI   $11 1
l_i_assert_true:        BRA    l_i_assert
l_i_test_false:         BRA    l_i_test
                        ADDI   $8 1
                        RL     $9 1
                        EXCH   $10 $1
                        ADDI   $1 -1
                        EXCH   $11 $1
                        ADDI   $1 -1
                        EXCH   $12 $1
                        ADDI   $1 -1
                        EXCH   $14 $1
                        ADDI   $1 -1
                        EXCH   $16 $1
                        ADDI   $1 -1
                        EXCH   $17 $1
                        ADDI   $1 -1
                        EXCH   $18 $1
                        ADDI   $1 -1
                        EXCH   $20 $1
                        ADDI   $1 -1
                        EXCH   $21 $1
                        ADDI   $1 -1
                        EXCH   $22 $1
                        ADDI   $1 -1
```

```
                EXCH  $23 $1
                ADDI  $1 -1
                BRA   l_malloc1
                ADDI  $1 1
                EXCH  $23 $1
                ADDI  $1 1
                EXCH  $22 $1
                ADDI  $1 1
                EXCH  $21 $1
                ADDI  $1 1
                EXCH  $20 $1
                ADDI  $1 1
                EXCH  $18 $1
                ADDI  $1 1
                EXCH  $17 $1
                ADDI  $1 1
                EXCH  $16 $1
                ADDI  $1 1
                EXCH  $14 $1
                ADDI  $1 1
                EXCH  $12 $1
                ADDI  $1 1
                EXCH  $11 $1
                ADDI  $1 1
                EXCH  $10 $1
                RR    $9 1
                ADDI  $8 -1
                XOR   $12 $6
                EXCH  $12 $17
                ADD   $6 $9
l_i_assert:     BNE   $11 $0 l_i_assert_true
                EXCH  $12 $17
                SUB   $6 $9
cmp_top_7:      BEQ   $6 $12 cmp_bot_8
                XORI  $21 1
cmp_bot_8:      BEQ   $6 $12 cmp_top_7
cmp_top_9:      BNE   $12 $0 cmp_bot_10
                XORI  $22 1
cmp_bot_10:     BNE   $12 $0 cmp_top_9
                ORX   $23 $21 $22
                XOR   $11 $23
                ORX   $23 $21 $22
cmp_bot_10_i:   BNE   $12 $0 cmp_top_9_i
                XORI  $22 1
cmp_top_9_i:    BNE   $12 $0 cmp_bot_10_i
```

```
cmp_bot_8_i:          BEQ     $6  $12  cmp_top_7_i
                      XORI    $21  1
cmp_top_7_i:          BEQ     $6  $12  cmp_bot_8_i
                      ADD     $6  $9
                      EXCH    $12  $17
l_o_assert:           BNE     $10  $0  l_o_assert_true
                      XOR     $15  $9
                      SUB     $15  $7
cmp_top_3:            BGEZ    $15  cmp_bot_4
                      XORI    $16  1
cmp_bot_4:            BGEZ    $15  cmp_top_3
                      XOR     $10  $16
cmp_bot_4_i:          BGEZ    $15  cmp_top_3_i
                      XORI    $16  1
cmp_top_3_i:          BGEZ    $15  cmp_bot_4_i
                      ADD     $15  $7
                      XOR     $15  $9
l_malloc1_bot:        BRA     l_malloc1_top
l_main_0_top:         BRA     l_main_0_bot
                      ADDI    $1  1
                      EXCH    $2  $1
                      EXCH    $3  $1
                      ADDI    $1  −1
l_main_0:             SWAPBR  $2
                      NEG     $2
                      ADDI    $1  1
                      EXCH    $3  $1
                      EXCH    $2  $1
                      ADDI    $1  −1
                      ADD     $6  $3
                      ADDI    $6  3
                      EXCH    $7  $6
                      ADDI    $6  −3
                      SUB     $6  $3
                      XORI    $8  100
                      ADD     $7  $8
                      XORI    $8  100
                      ADD     $6  $3
                      ADDI    $6  3
                      EXCH    $7  $6
                      ADDI    $6  −3
                      SUB     $6  $3
                      XORI    $6  1
entry_11:             BEQ     $6  $0  assert_13
                      ADD     $7  $3
```

```
                    ADDI   $7  2
                    EXCH   $8  $7
                    ADDI   $7  −2
                    SUB    $7  $3
cmp_top_15:         BNE    $8  $0  cmp_bot_16
                    XORI   $9  1
cmp_bot_16:         BNE    $8  $0  cmp_top_15
f_top_17:           BEQ    $9  $0  f_bot_18
                    XORI   $10  1
f_bot_18:           BEQ    $9  $0  f_top_17
                    XOR    $6  $10
f_bot_18_i:         BEQ    $9  $0  f_top_17_i
                    XORI   $10  1
f_top_17_i:         BEQ    $9  $0  f_bot_18_i
cmp_bot_16_i:       BNE    $8  $0  cmp_top_15_i
                    XORI   $9  1
cmp_top_15_i:       BNE    $8  $0  cmp_bot_16_i
                    ADD    $7  $3
                    ADDI   $7  2
                    EXCH   $8  $7
                    ADDI   $7  −2
                    SUB    $7  $3
                    ADD    $7  $3
                    ADDI   $7  2
                    EXCH   $8  $7
                    ADDI   $7  −2
                    SUB    $7  $3
                    ADD    $9  $3
                    ADDI   $9  3
                    EXCH   $10  $9
                    ADDI   $9  −3
                    SUB    $9  $3
cmp_top_19:         BNE    $8  $10  cmp_bot_20
                    XORI   $11  1
cmp_bot_20:         BNE    $8  $10  cmp_top_19
f_top_21:           BEQ    $11  $0  f_bot_22
                    XORI   $12  1
f_bot_22:           BEQ    $11  $0  f_top_21
                    XOR    $6  $12
f_bot_22_i:         BEQ    $11  $0  f_top_21_i
                    XORI   $12  1
f_top_21_i:         BEQ    $11  $0  f_bot_22_i
cmp_bot_20_i:       BNE    $8  $10  cmp_top_19_i
                    XORI   $11  1
cmp_top_19_i:       BNE    $8  $10  cmp_bot_20_i
```

```
                ADD     $9 $3
                ADDI    $9 3
                EXCH    $10 $9
                ADDI    $9 −3
                SUB     $9 $3
                ADD     $7 $3
                ADDI    $7 2
                EXCH    $8 $7
                ADDI    $7 −2
                SUB     $7 $3
test_12:        BNE     $6 $0 exit_14
                ADD     $7 $3
                ADDI    $7 2
                EXCH    $8 $7
                ADDI    $7 −2
                SUB     $7 $3
                XORI    $9 1
                ADD     $8 $9
                XORI    $9 1
                ADD     $7 $3
                ADDI    $7 2
                EXCH    $8 $7
                ADDI    $7 −2
                SUB     $7 $3
assert_13:      BRA     entry_11
exit_14:        BRA     test_12
                XORI    $6 1
                ADD     $6 $3
                ADDI    $6 2
                EXCH    $7 $6
                ADDI    $6 −2
                SUB     $6 $3
                ADD     $8 $3
                ADDI    $8 3
                EXCH    $9 $8
                ADDI    $8 −3
                SUB     $8 $3
                SUB     $7 $9
                ADD     $8 $3
                ADDI    $8 3
                EXCH    $9 $8
                ADDI    $8 −3
                SUB     $8 $3
                ADD     $6 $3
                ADDI    $6 2
```

```
                EXCH   $7 $6
                ADDI   $6 −2
                SUB    $6 $3
                ADD    $6 $3
                ADDI   $6 3
                EXCH   $7 $6
                ADDI   $6 −3
                SUB    $6 $3
                XORI   $8 100
                SUB    $7 $8
                XORI   $8 100
                ADD    $6 $3
                ADDI   $6 3
                EXCH   $7 $6
                ADDI   $6 −3
                SUB    $6 $3
l_main_0_bot:   BRA    l_main_0_top
start:          BRA    top
                START
                ADDI   $4 393
                XOR    $5 $4
                ADDI   $5 10
                XOR    $7 $5
                ADDI   $4 10
                ADDI   $4 −1
                EXCH   $7 $4
                ADDI   $4 1
                ADDI   $4 −10
                XOR    $1 $5
                ADDI   $1 2048
                ADDI   $1 −4
                XOR    $3 $1
                XORI   $6 3
                EXCH   $6 $3
                ADDI   $1 −1
                EXCH   $3 $1
                ADDI   $1 −1
                BRA    l_main_0
                ADDI   $1 1
                EXCH   $3 $1
                ADDI   $3 1
                ADDI   $3 1
                EXCH   $6 $3
                XORI   $7 1
                EXCH   $6 $7
```

```
                XORI    $7  1
                ADDI    $3  -1
                ADDI    $3  -1
                ADDI    $3  1
                ADDI    $3  2
                EXCH    $6  $3
                XORI    $7  2
                EXCH    $6  $7
                XORI    $7  2
                ADDI    $3  -2
                ADDI    $3  -1
                ADDI    $1  1
                EXCH    $6  $3
                XORI    $6  3
                XOR     $3  $1
                ADDI    $1  4
                ADDI    $1  -2048
                XOR     $1  $5
                ADDI    $5  -10
                XOR     $5  $4
                ADDI    $4  -393
finish:         FINISH
```

9 Appendix C

prgReduced.pal

```
;; pendulum pal file
top:                    BRA     start
l_r_nodeCount:          DATA    0
l_r_limit:              DATA    0
l_Program_vt:           DATA    214
l_malloc_top:           BRA     l_malloc_bot
l_malloc:               SWAPBR  $2
                        NEG     $2
                        ADDI    $9 2
                        XOR     $8 $0
                        ADDI    $1 1
                        EXCH    $6 $1
                        ADDI    $1 1
                        EXCH    $7 $1
                        EXCH    $2 $1
                        ADDI    $1 -1
                        BRA     l_malloc1
                        ADDI    $1 1
                        EXCH    $2 $1
                        EXCH    $7 $1
                        ADDI    $1 -1
                        EXCH    $6 $1
                        ADDI    $1 -1
                        XOR     $8 $0
                        ADDI    $9 -2
l_malloc_bot:           BRA     l_malloc_top
l_malloc1_top:          BRA     l_malloc1_bot
                        ADDI    $1 1
                        EXCH    $2 $1
                        SUB     $17 $8
                        XOR     $17 $4
l_malloc1:              SWAPBR  $2
                        NEG     $2
                        EXCH    $2 $1
                        ADDI    $1 -1
                        XOR     $17 $4
                        ADD     $17 $8
                        EXCH    $19 $17
                        XOR     $18 $19
                        EXCH    $19 $17
                        XOR     $13 $9
```

```
                SUB   $13 $7
cmp_top_1:      BGEZ  $13 cmp_bot_2
                XORI  $14 1
cmp_bot_2:      BGEZ  $13 cmp_top_1
                XOR   $10 $14
cmp_bot_2_i:    BGEZ  $13 cmp_top_1_i
                XORI  $14 1
cmp_top_1_i:    BGEZ  $13 cmp_bot_2_i
                ADD   $13 $7
                XOR   $13 $9
l_o_test:       BEQ   $10 $0 l_o_test_false
                XORI  $10 1
                ADDI  $8 1
                EXCH  $19 $17
                XOR   $18 $19
                EXCH  $19 $17
                RL    $9 1
                EXCH  $10 $1
                ADDI  $1 -1
                EXCH  $11 $1
                ADDI  $1 -1
                EXCH  $12 $1
                ADDI  $1 -1
                EXCH  $14 $1
                ADDI  $1 -1
                EXCH  $16 $1
                ADDI  $1 -1
                EXCH  $17 $1
                ADDI  $1 -1
                EXCH  $18 $1
                ADDI  $1 -1
                EXCH  $20 $1
                ADDI  $1 -1
                EXCH  $21 $1
                ADDI  $1 -1
                EXCH  $22 $1
                ADDI  $1 -1
                EXCH  $23 $1
                ADDI  $1 -1
                BRA   l_malloc1
                ADDI  $1 1
                EXCH  $23 $1
                ADDI  $1 1
                EXCH  $22 $1
                ADDI  $1 1
```

```
                        EXCH  $21 $1
                        ADDI  $1 1
                        EXCH  $20 $1
                        ADDI  $1 1
                        EXCH  $18 $1
                        ADDI  $1 1
                        EXCH  $17 $1
                        ADDI  $1 1
                        EXCH  $16 $1
                        ADDI  $1 1
                        EXCH  $14 $1
                        ADDI  $1 1
                        EXCH  $12 $1
                        ADDI  $1 1
                        EXCH  $11 $1
                        ADDI  $1 1
                        EXCH  $10 $1
                        RR    $9 1
                        ADDI  $8 -1
                        XORI  $10 1
l_o_assert_true:        BRA   l_o_assert
l_o_test_false:         BRA   l_o_test
cmp_top_5:              BEQ   $18 $0 cmp_bot_6
                        XORI  $20 1
cmp_bot_6:              BEQ   $18 $0 cmp_top_5
                        XOR   $11 $20
cmp_bot_6_i:            BEQ   $18 $0 cmp_top_5_i
                        XORI  $20 1
cmp_top_5_i:            BEQ   $18 $0 cmp_bot_6_i
l_i_test:               BEQ   $11 $0 l_i_test_false
                        XORI  $11 1
                        ADD   $6 $18
                        SUB   $18 $6
                        EXCH  $12 $6
                        EXCH  $12 $17
                        XOR   $12 $6
                        XORI  $11 1
l_i_assert_true:        BRA   l_i_assert
l_i_test_false:         BRA   l_i_test
                        ADDI  $8 1
                        RL    $9 1
                        EXCH  $10 $1
                        ADDI  $1 -1
                        EXCH  $11 $1
                        ADDI  $1 -1
```

```
EXCH    $12  $1
ADDI    $1  −1
EXCH    $14  $1
ADDI    $1  −1
EXCH    $16  $1
ADDI    $1  −1
EXCH    $17  $1
ADDI    $1  −1
EXCH    $18  $1
ADDI    $1  −1
EXCH    $20  $1
ADDI    $1  −1
EXCH    $21  $1
ADDI    $1  −1
EXCH    $22  $1
ADDI    $1  −1
EXCH    $23  $1
ADDI    $1  −1
BRA     l_malloc1
ADDI    $1  1
EXCH    $23  $1
ADDI    $1  1
EXCH    $22  $1
ADDI    $1  1
EXCH    $21  $1
ADDI    $1  1
EXCH    $20  $1
ADDI    $1  1
EXCH    $18  $1
ADDI    $1  1
EXCH    $17  $1
ADDI    $1  1
EXCH    $16  $1
ADDI    $1  1
EXCH    $14  $1
ADDI    $1  1
EXCH    $12  $1
ADDI    $1  1
EXCH    $11  $1
ADDI    $1  1
EXCH    $10  $1
RR      $9  1
ADDI    $8  −1
XOR     $12  $6
EXCH    $12  $17
```

```
                        ADD     $6 $9
l_i_assert:             BNE     $11 $0 l_i_assert_true
                        EXCH    $12 $17
                        SUB     $6 $9
cmp_top_7:              BEQ     $6 $12 cmp_bot_8
                        XORI    $21 1
cmp_bot_8:              BEQ     $6 $12 cmp_top_7
cmp_top_9:              BNE     $12 $0 cmp_bot_10
                        XORI    $22 1
cmp_bot_10:             BNE     $12 $0 cmp_top_9
                        ORX     $23 $21 $22
                        XOR     $11 $23
                        ORX     $23 $21 $22
cmp_bot_10_i:           BNE     $12 $0 cmp_top_9_i
                        XORI    $22 1
cmp_top_9_i:            BNE     $12 $0 cmp_bot_10_i
cmp_bot_8_i:            BEQ     $6 $12 cmp_top_7_i
                        XORI    $21 1
cmp_top_7_i:            BEQ     $6 $12 cmp_bot_8_i
                        ADD     $6 $9
                        EXCH    $12 $17
l_o_assert:             BNE     $10 $0 l_o_assert_true
                        XOR     $15 $9
                        SUB     $15 $7
cmp_top_3:              BGEZ    $15 cmp_bot_4
                        XORI    $16 1
cmp_bot_4:              BGEZ    $15 cmp_top_3
                        XOR     $10 $16
cmp_bot_4_i:            BGEZ    $15 cmp_top_3_i
                        XORI    $16 1
cmp_top_3_i:            BGEZ    $15 cmp_bot_4_i
                        ADD     $15 $7
                        XOR     $15 $9
l_malloc1_bot:          BRA     l_malloc1_top
l_main_0_top:           BRA     l_main_0_bot
                        ADDI    $1 1
                        EXCH    $2 $1
                        EXCH    $3 $1
                        ADDI    $1 -1
l_main_0:               SWAPBR  $2
                        NEG     $2
                        ADDI    $1 1
                        EXCH    $3 $1
                        EXCH    $2 $1
                        ADDI    $1 -1
```

```
                    EXCHI  $7 $3 3
                    XORI   $8 100
                    ADD    $7 $8
                    XORI   $8 100
                    EXCHI  $7 $3 3
                    XORI   $6 1
entry_11:           BEQ    $6 $0 assert_13
                    EXCHI  $8 $3 2
                    OUT    $8 $31
iftop_3:            BNE    $8 $0 ifbot_3
                    XORI   $6 1
ifbot_3:            BNE    $8 $0 iftop_3
                    EXCHI  $8 $3 2
                    EXCHI  $8 $3 2
                    EXCHI  $10 $3 3
iftop_2:            BNE    $8 $10 ifbot_2
                    XORI   $6 1
ifbot_2:            BNE    $8 $10 iftop_2
                    EXCHI  $10 $3 3
                    EXCHI  $8 $3 2
test_12:            BNE    $6 $0 exit_14
                    EXCHI  $8 $3 2
                    XORI   $9 1
                    ADD    $8 $9
                    XORI   $9 1
                    EXCHI  $8 $3 2
assert_13:          BRA    entry_11
exit_14:            BRA    test_12
                    XORI   $6 1
                    EXCHI  $7 $3 3
                    EXCHI  $9 $3 3
                    SUB    $7 $9
                    EXCHI  $9 $3 3
                    EXCHI  $7 $3 3
                    EXCHI  $7 $3 3
                    XORI   $8 100
                    SUB    $7 $8
                    XORI   $8 100
                    EXCHI  $7 $3 3
l_main_0_bot:       BRA    l_main_0_top
start:              BRA    top
                    START
                    ADDI   $4 314
                    XOR    $5 $4
                    ADDI   $5 10
```

```
XOR     $7  $5
ADDI    $4  10
ADDI    $4  −1
EXCH    $7  $4
ADDI    $4  1
ADDI    $4  −10
XOR     $1  $5
ADDI    $1  2048
ADDI    $1  −4
XOR     $3  $1
XORI    $6  3
EXCH    $6  $3
ADDI    $1  −1
EXCH    $3  $1
ADDI    $1  −1
ADDI    $31  1
ADDI    $30  −1
BRA     l_main_0
ADDI    $31  −1
ADDI    $30  1
ADDI    $1  1
EXCH    $3  $1
ADDI    $3  1
ADDI    $3  1
EXCH    $6  $3
XORI    $7  1
EXCH    $6  $7
XORI    $7  1
ADDI    $3  −1
ADDI    $3  −1
ADDI    $3  1
ADDI    $3  2
EXCH    $6  $3
XORI    $7  2
EXCH    $6  $7
XORI    $7  2
ADDI    $3  −2
ADDI    $3  −1
ADDI    $1  1
EXCH    $6  $3
XORI    $6  3
XOR     $3  $1
ADDI    $1  4
ADDI    $1  −2048
XOR     $1  $5
```

```
                                   ADDI     $5  −10
                                   XOR      $5  $4
                                   ADDI     $4  −314
finish:                            FINISH
```

10 Appendix D

remove_redundancy.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

/** ******* ************************************************ */
/** STRUCTS ************************************************ */
/** ******* ************************************************ */
/* File & flow handling */
struct Controlflow {
    int patternm;
    int pattern;
} controller;

struct Buffer {
    /* The current instruction line */
    char line[256];
    char param_set[5][32];

    /* Some list to split
     * an instruction line
     * into single paramters */
    char param_sets[5][5][32];

    /* Previous instruction lines */
    char previous_lines[5][256];
} buffer;

struct Filehandler {
    FILE *input_file;
    FILE *output_file;
} filehandler;
/** ******* ************************************************ */
/** ******* ************************************************ */

/** ******* ************************************************ */
/* functions */
/** ******* ************************************************ */
int apply();
int memory_exchange();
int special_if_statements();

```

```c
/** ******* ********************************************* */
/* File & flow handling */
/** ******* ********************************************* */
int initialize(const char *file);
int getinst();
char* getparam(int idx);
int saveline(int idx);
int setPreviousLines();

/** ******* ********************************************* */
/** main */
/** ******* ********************************************* */
int main(int arg, char *args[])
    {
    filehandler.input_file=fopen(args[1], "r");
    if(!filehandler.input_file) {
        printf("%s: Unable to open input file %s.\n","",args[1]);
        exit(1);
    }
    fclose(filehandler.input_file);

    /* memory exchange */
    apply(1, args[1]);
    /* special if-statements */
    apply(2, "reduced.pal");

    return 0;
    }

int apply(int method, const char *file) {
    int action;
    initialize(file);

    if(method == 1) {
        /* loop to load instruction memory */
        while(getinst()) {
            /* check for memory exchange pattern */
            action = memory_exchange();

            if(action == 0)
                fputs(buffer.line, filehandler.output_file);
        }
    } else {
        while(getinst()) {
            /* check for special if-statements */
```

```
            action = special_if_statements();

            if(action == 0)
                fputs(buffer.line, filehandler.output_file);
        }
    }
    fclose(filehandler.input_file);
    fclose(filehandler.output_file);
}

int memory_exchange() {
    int action = 0;

    /* reset on an add */
    if( !strcasecmp(getparam(0), "ADD") ) {
        for(int i = 0; i < controller.pattern; i++)
            fputs(buffer.previous_lines[i],
                filehandler.output_file);
        controller.pattern = 0;
    }

     /* memory exchange start
      * save lines */
    if( !strcasecmp(getparam(0), "ADD")
        || controller.pattern > 0) {
        saveline(controller.pattern);
        controller.pattern += 1;
        action = 1;
    }

    /* memory exchange end */
    if( !strcasecmp(getparam(0), "SUB")
        && controller.pattern == 5) {
        char buf[256];

        /* Place previous lines in
         * param_sets[0..4]
         *
         * EXCH within param_sets[2]
         * ADDI within param_sets[1]
         * SUB within param_sets[4]
         */
        setPreviousLines();

        if( !strcasecmp(buffer.param_sets[2][0], "EXCH") ) {
```

```
                strcpy(buffer.param_sets[2][4]
                    , "                         ");
                sprintf(buf,"%s%s  %s %s %s\n",buffer.param_sets[2][4]
                    ,"EXCHI"
                    ,buffer.param_sets[2][1]
                    ,buffer.param_sets[4][2]
                    ,buffer.param_sets[1][2]);
                printf("%s", buf);

                fputs(buf, filehandler.output_file);
                controller.pattern = 0;
                action = 1;
            }
        }

    /* memory exchange final
     * if no pattern was regconized
     * print all five lines */
    if( controller.pattern == 5) {
        for(int i = 0; i < controller.pattern; i++)
            fputs(buffer.previous_lines[i],
            filehandler.output_file);
        controller.pattern = 0;
        action = 1;
    }

    return action;
}

int special_if_statements() {
    int action = 0;

    if( strstr(getparam(0), "cmp") != NULL
        && controller.patternm == 0) {
        saveline(0); /* cmp */
        printf("%s%s\n", "cmp: ", buffer.previous_lines[0]);
        controller.patternm = 1;
    }

    if( !strcasecmp(getparam(0), "XOR")
    && controller.patternm == 1) {
        saveline(1); /* xor */
        printf("%s%s\n", "xor: ", buffer.previous_lines[1]);
    }

```

```c
    if( getparam(0)[strlen(getparam(0))-2]=='i'
        && strstr(getparam(0), "cmp") != NULL
        && controller.patternm == 1) {
        char buf[256];
        char cls[32];
        setPreviousLines();

        strcpy(buffer.param_set[4], "                         ");
        strcpy(buffer.param_set[2], "               ");

        char *pp1 = strndup(buffer.param_sets[0][0],
            strlen(buffer.param_sets[0][0]) -1);
        char *pp2 = strndup(getparam(0), strlen(getparam(0))-3);
        char *pp3 = strndup(buffer.param_sets[0][4],
            strlen(buffer.param_sets[0][4]));
        printf("%s %s\n", "pp1: ", pp1);
        printf("%s %s\n", "pp2: ", pp2);

        if( !strcasecmp(pp1, pp2) ) {
            /* if */
            sprintf(buf,"%s%s%s    %s %s %s\n",
                buffer.param_sets[0][0],
                getparam(2),
                buffer.param_sets[0][1],
                buffer.param_sets[0][2],
                buffer.param_sets[0][3],
                buffer.param_sets[0][4]);
            fputs(buf, filehandler.output_file);

            /* xori */
            sprintf(buf,"%s%s   %s %s\n",getparam(4), "XORI",
                buffer.param_sets[1][1], "1");
            fputs(buf, filehandler.output_file);

            /* fi */
            sprintf(buf,"%s:%s%s    %s %s %s\n", pp3, getparam(2),
                buffer.param_sets[0][1],
                buffer.param_sets[0][2],
                buffer.param_sets[0][3],
                pp1);
            fputs(buf, filehandler.output_file);

            controller.patternm = 0;
        }
        action = 1;
```

```
    }

    return action+controller.patternm;
}

int initialize(const char *file) {
    /* copy input file into
     * tmp file */
    filehandler.input_file=fopen(file, "r");
    filehandler.output_file=fopen("tmp.pal", "w");
    while(getinst()) {
        fputs(buffer.line, filehandler.output_file);
    }
    fclose(filehandler.output_file);
    fclose(filehandler.input_file);

    filehandler.input_file = fopen("tmp.pal", "r");
    filehandler.output_file = fopen("reduced.pal", "w");

    /* make sure file is in valid pendulum format */
    /* get first line */
    fgets(buffer.line,256,filehandler.input_file);
    if( strncmp(buffer.line, ";; pendulum pal file", 20) ) {
        /* compare with known header */
        printf("Input file not in in Pendulum pal format.\n");
        exit(1);
    }
    /* Initialize new file */
    sprintf(buffer.line, ";; pendulum pal file\n");
    fputs(buffer.line, filehandler.output_file);
    return 0;
};

int getinst() {
    int r = fgets(buffer.line,256,filehandler.input_file);

    /*  */
    int fields=sscanf(buffer.line,"%s%s%s%s%s",
        buffer.param_set[0],
        buffer.param_set[1],
        buffer.param_set[2],
        buffer.param_set[3],
        buffer.param_set[4]);

    if( fields==0 || fields==EOF ) return 0;
```

```

    return r;
};

char* getparam(int idx) {
    return buffer.param_set[idx];
}

int saveline(int idx) {
    strcpy(buffer.previous_lines[idx], buffer.line);
    return 0;
};

int setPreviousLines() {
    for(int i = 0; i < 5; i++)
        {
        sscanf(buffer.previous_lines[i],"%s%s%s%s%s",
        buffer.param_sets[i][0],
        buffer.param_sets[i][1],
        buffer.param_sets[i][2],
        buffer.param_sets[i][3],
        buffer.param_sets[i][4]);
        }
    return 0;
}
```

References

[1] http://www.computinghistory.org.uk/det/32489/Kathleen-Booth/

[2] https://www.ibm.com/support/knowledgecenter/SSLTBW_2.1.0/com.ibm.zos.v2r1.asma400/asmr102112.htm

[3] https://en.wikipedia.org/wiki/Von_Neumann_architecture

[4] Vieri, C. J. et al. *Pendulum: A Reversible Computer Architecture*. Master's Thesis. University of California at Berkeley 1993.

[5] Michael P. Frank. *Reversibility for Efficient Computing*. Ph.D Thesis. University of Florida, 1999.

[6] Carter, P. A. *PC Assembly Language*. 2006

[7] Yokoyama, T. and Glück, R. *A reversible programming language and its invertible self-interpreter*. ACM, 2007.

[8] C. H. Bennet. *Notes on the history of reversible computation*. IBM J. Res. Dev., 32(1), 1988

[9] https://github.com/TueHaulund/PendVM

[10] https://plato.stanford.edu/entries/turing-machine/

[11] D. A. Patterson, J. L. Hennessy. *Computer Organization and Design - The Hardware / Software Interface*, 4th. Ed., Elsevier, Inc., 2012

[12] J. C. Maxwell. *Theory of Heat,* 4th Ed., Longmans, Green & Co., London, 1875 (1st Ed. 1871)

[13] M. H. Cservenka. *ROOPLPPC*. https://github.com/cservenka/ROOPLPPC

[14] M. H. Cservenka. *Design and Implementation of Dynamic Memory Management in a Reversible Object-Oriented Programming Language*, Master's Thesis. University of Copenhagen 2018. https://github.com/cservenka/masters-thesis-report