Program Analysis and Transformation Search and Reduce Redundant Patterns in ROOPLPPC

Benjamin Brandt Ohrt, zpn492 June 19, 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	3
	1.1 Assembly language	3
	1.1.1 Example program	3
	1.2 Reversible computing	4
	1.3 Pendulum microprocessor	5
	1.4 Motivation	6
	1.5 Statement	6
	1.6 Outline	6
2	Programming languages	7
	2.1 Pendulum Assembly Language	7
	2.1.1 Example program	7
	2.2 Reversible Object-Oriented Programming	8
3	Analysis of redundancy	9
	3.1 First part - patterns	10
	3.1.1 Getting data from memory	10
	3.1.2 Special if-statements	11
	3.2 Second part - methods for reduced redundancy	12
	3.2.1 PendVM - OUT	12
	3.2.2 PendVM - EXCHI	13
	3.2.3 Algorithm - Memory exchange	14
	3.2.4 Algorithm - Special if-statements	14
	3.2.5 Algorithm - Remove redundancy	14
4	Evaluation	17
	4.1 Scenarios which is not supported	17
	2.2 Scondards winds is not supported to the state of the	
5	Conclusion	18
6	Further work	18
7	Appendix A	19
8	Appendix B	19
9	Appendix C	29
10	Appendix D	37

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

```
1
   J main
                                      for \ int \ i = 0; \ i < 100; \ i++
                                         count \neq i
 3
   loop_i:
   ADD $a0, $a0, $zero
                                       count \neq i
   ADDI $a1, $a1, 1
                                       i++
6
   SLT $t0, $a1, $a2
 7
   BEQ $t0, 1, loop_i
                                       i < 100
8
   JR $ra
                                       return
9
   main:
10
                                      count = 0
   |\mathbf{ADDI} \$ a0, \$ zero, 0
11
12
   ADDI $a1, $zero, 0
                                       i = 0
13
   |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_BTlnN , 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 $k_B ln2$ [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]
3
   subtop:
              BRA subbot
                                      entry/exit point
4
   main:
              SWAPBR $2
              NEG $2
                                      negate offset to return caller
5
6
              EXCH $2 $1
                                     push return offset to stack
7
              ADDI $30 100
              BRA swap
                                     set\ limit\ (\$28)\ +=\ a,\ set\ a=0
8
9
   looptop:
              BNE $30 $0 loopbot
                                   ; from a = 0 do
10
              ADDI $30 1
                                     a \neq 1
              BNE $30 $28 looptop; until \ a = limit \ loop \ body
11
   loopbot:
12
              SUB $28 $30
                                      set limit = 0
13
              ADDI $30 -100
                                      a -= n
                                      pop return address
14
              EXCH $2 $1
15
   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
1
2
       int nodeCount
3
       int limit
4
       method main()
5
            limit += 100
            from nodeCount = 0 do
6
7
               skip
8
            loop
9
               nodeCount += 1
            until nodeCount = limit
10
            nodeCount -= limit
11
12
            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:

1 0		1
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 r1 some offset. One could formalize a method m with three arguments: 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
$f_t o p_1 7$:	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	$88 \ 0 \ \text{cmp_top_15_i}$
	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 iout (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

3.2.4 Algorithm - Special if-statements

```
Find the pattern:
cmp_top_x
             inst_x
cmp_bot_x
             inst_x
<...>
XOR
             $r1 $r2
<\ldots>
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 re-
				move_ redundancy has been
				applied
p"	1328	41	5	The example PAL program
				from the section of Pen-
				dulum 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 $rv
<...>
             $rz $rw
ANDX
<...>
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 removeredundancy.

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

For ITo 100. pal

```
;;; increment a from [1..n]
   subtop:
             BRA subbot
3
             SWAPBR $2
   main:
                                  ; entry/exit point
4
             NEG $2
                                  ; negate offset to return caller
             EXCH $2 $1
5
                                  ; push return offset to stack
6
             ADDI $30 100
                                  ; a += n
7
                                  ; set limit ($28) += a, set a = 0
             BRA swap
8
             BNE $30 $0 loopbot; from a = 0 do
   looptop:
9
             ADDI $30 1
                                  : a += 1
   loopbot:
             BNE $30 $28 looptop; until a = limit loop body
10
11
             SUB $28 $30
                                  ; set limit = 0
12
             ADDI $30 -100
                                   a = n
             EXCH $2 $1
                                  ; pop return address
13
14
   subbot:
             BRA subtop
   ;; swap(int a, int b)
15
   subtop_3: BRA subbot_3
16
   swap:
             SWAPBR $2
17
18
             NEG $2
             EXCH $2 $1
19
20
             ADD $28 $30
21
             SUB $30 $28
22
             EXCH $2 $1
23
   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_bot
   l_malloc_top:
                              BRA
7
   l_malloc:
                              SWAPBR $2
8
                              NEG
                                      $2
9
                              ADDI
                                      $9 2
10
                              XOR
                                      $8 $0
11
                              ADDI
                                      $1 1
                              EXCH
                                      $6 $1
12
```

```
13
                                ADDI
                                        $1 1
14
                                EXCH
                                        $7 $1
15
                                EXCH
                                        $2 $1
                                ADDI
                                        $1 -1
16
                                BRA
                                        l_malloc1
17
18
                                ADDI
                                        $1 1
                                        $2 $1
                                EXCH
19
20
                                EXCH
                                        $7 $1
21
                                ADDI
                                        \$1 -1
22
                                EXCH
                                        $6 $1
                                ADDI
                                        $1 -1
23
24
                                XOR
                                        $8 $0
                                        $9 -2
25
                                ADDI
   l_malloc_bot:
                                BRA
                                        l_malloc_top
26
27
                                        l_malloc1_bot
   l_malloc1_top:
                                BRA
28
                                ADDI
                                        $1 1
29
                                EXCH
                                        $2 $1
30
                                SUB
                                        $17 $8
                                XOR
                                        $17 $4
31
32
   l_malloc1:
                                SWAPBR $2
33
                                NEG
                                        $2
                                        $2 $1
34
                                EXCH
35
                                ADDI
                                        $1 -1
                                XOR
                                        $17 $4
36
                                ADD
37
                                        $17 $8
38
                                EXCH
                                        $19 $17
                                XOR
39
                                        $18 $19
                                EXCH
                                        $19 $17
40
41
                                XOR
                                        $13 $9
42
                                SUB
                                        $13 $7
                                BGEZ
43
   cmp\_top\_1:
                                        $13 \text{ cmp\_bot\_2}
                                XORI
                                        $14 1
44
45
                                BGEZ
                                        $13 cmp_top_1
   cmp\_bot\_2:
46
                                XOR
                                        $10 $14
47
   cmp\_bot\_2\_i:
                                BGEZ
                                        13 \text{ cmp-top-1-i}
48
                                XORI
                                        $14 1
49
   cmp_top_1_i:
                                BGEZ
                                        $13 cmp_bot_2_i
50
                                ADD
                                        $13 $7
                                XOR
51
                                        $13 $9
52
   l_o_test:
                                BEQ
                                        $10 $0 l_o_test_false
53
                                XORI
                                        $10 1
54
                                ADDI
                                        $8 1
55
                                EXCH
                                        $19 $17
56
                                XOR
                                        $18 $19
                                EXCH
                                        $19 $17
57
```

- 0	DI	00.1
58	RL	\$9 1
59	EXCH	\$10 \$1
60	ADDI	\$1 -1
61	EXCH	\$11 \$1
62	ADDI	\$1 -1
63	EXCH	\$12 \$1
64	ADDI	\$1 -1
65	EXCH	\$14 \$1
66	ADDI	\$1 -1
67	EXCH	\$16 \$1
68	ADDI	\$1 -1
69	EXCH	\$17 \$1
70	ADDI	\$1 -1
71	EXCH	\$18 \$1
72	ADDI	\$1 -1
73	EXCH	\$20 \$1
74	ADDI	\$1 -1
75	EXCH	\$21 \$1
76	ADDI	\$1 -1
77	EXCH	\$22 \$1
78	ADDI	\$1 -1
79	EXCH	\$23 \$1
80	ADDI	\$1 -1
81	BRA	$l_malloc1$
82	ADDI	\$1 1
83	EXCH	\$23 \$1
84	ADDI	\$1 1
85	EXCH	\$22 \$1
86	ADDI	\$1 1
87	EXCH	\$21 \$1
88	ADDI	\$1 1
89	EXCH	\$20 \$1
90	ADDI EVCU	\$1 1
91	EXCH	\$18 \$1
92	ADDI	\$1 1
93	EXCH	\$17 \$1
94	ADDI	\$1 1
95	EXCH	\$16 \$1
96	ADDI	\$1 1
97	EXCH	\$14 \$1
98	ADDI	\$1 1
99	EXCH	\$12 \$1
100	ADDI	\$1 1
101	EXCH	\$11 \$1
102	ADDI	\$1 1
1		· ·

```
103
                                 EXCH
                                         $10 $1
104
                                 RR
                                         $9 1
                                         \$8 -1
105
                                 ADDI
106
                                 XORI
                                         $10 1
    l_o_assert_true:
                                 BRA
107
                                         l_oassert
108
    l_o_test_false:
                                 BRA
                                         l_o_test
109
                                         $18 $0 cmp_bot_6
    cmp_top_5:
                                 BEQ
110
                                 XORI
                                         $20 1
111
                                 BEQ
                                         $18 $0 cmp_top_5
    cmp_bot_6:
112
                                 XOR
                                         $11 $20
                                         18 \ cmp\_top\_5_i
113
    cmp\_bot\_6\_i:
                                 BEQ
                                         $20 1
114
                                 XORI
115
                                 BEQ
                                         18 \ cmp_bot_6_i
    cmp\_top\_5\_i:
                                         11 \ 0 \ l_i_test_false
116
    l_i test:
                                 BEQ
117
                                 XORI
                                         $11 1
118
                                 ADD
                                         $6 $18
                                 SUB
                                         $18 $6
119
120
                                 EXCH
                                         $12 $6
                                 EXCH
                                         $12 $17
121
                                         $12 $6
122
                                XOR
123
                                 XORI
                                         $11 1
124
    l_i_assert_true:
                                 BRA
                                         l_i_assert
125
    l_i_test_false:
                                 BRA
                                         l_i test
126
                                 ADDI
                                         $8 1
                                         $9 1
127
                                 RL
128
                                 EXCH
                                         $10 $1
                                 ADDI
                                         $1 -1
129
                                 EXCH
                                         $11 $1
130
131
                                 ADDI
                                         $1 -1
132
                                 EXCH
                                         $12 $1
                                 ADDI
                                         \$1 -1
133
                                         $14 $1
                                 EXCH
134
                                 ADDI
                                         $1 -1
135
                                 EXCH
                                         $16 $1
136
137
                                 ADDI
                                         $1 -1
138
                                EXCH
                                         $17 $1
139
                                 ADDI
                                         $1 -1
                                 EXCH
                                         $18 $1
140
                                 ADDI
141
                                         $1 -1
142
                                 EXCH
                                         $20 $1
                                 ADDI
143
                                         $1 -1
144
                                 EXCH
                                         $21 $1
145
                                 ADDI
                                         $1 -1
                                         $22 $1
146
                                EXCH
                                 ADDI
                                         $1 -1
147
```

```
148
                                 EXCH
                                         $23 $1
149
                                 ADDI
                                         \$1 -1
150
                                 BRA
                                         l_malloc1
                                 ADDI
                                         $1 1
151
                                 EXCH
                                         $23 $1
152
                                 ADDI
                                         $1 1
153
                                 EXCH
                                         $22 $1
154
                                 ADDI
                                         $1 1
155
                                 EXCH
                                         $21 $1
156
157
                                 ADDI
                                         $1 1
158
                                 EXCH
                                         $20 $1
                                 ADDI
159
                                         $1 1
                                 EXCH
                                         $18 $1
160
                                 ADDI
161
                                         $1 1
                                 EXCH
                                         $17 $1
162
163
                                 ADDI
                                         $1 1
                                 EXCH
                                         $16 $1
164
                                 ADDI
                                         $1 1
165
                                 EXCH
                                         $14 $1
166
167
                                 ADDI
                                         $1 1
                                 EXCH
                                         $12 $1
168
169
                                 ADDI
                                         $1 1
170
                                 EXCH
                                         $11 $1
                                 ADDI
171
                                         $1 1
                                 EXCH
                                         $10 $1
172
                                 RR
173
                                         $9 1
174
                                 ADDI
                                         \$8 -1
                                         $12 $6
175
                                 XOR
176
                                 EXCH
                                         $12 $17
177
                                 ADD
                                         $6 $9
178
    l_i_assert:
                                 BNE
                                         $11 $0 l_i_assert_true
                                         $12 $17
179
                                 EXCH
180
                                 SUB
                                         $6 $9
                                         $6 $12 cmp_bot_8
181
    cmp\_top\_7:
                                 BEQ
182
                                 XORI
                                         $21 1
183
                                 BEQ
                                         $6 $12 cmp_top_7
    cmp_bot_8:
184
    cmp_top_9:
                                 BNE
                                         $12 $0 cmp_bot_10
185
                                 XORI
                                         $22 1
186
    cmp\_bot\_10:
                                 BNE
                                         $12 $0 cmp_top_9
187
                                 ORX
                                         $23 $21 $22
188
                                 XOR
                                         $11 $23
189
                                 ORX
                                         $23 $21 $22
190
    cmp\_bot\_10\_i:
                                 BNE
                                         $12 $0 cmp_top_9_i
191
                                         $22 1
                                 XORI
                                 BNE
192
    cmp_top_9_i:
                                         $12 $0 cmp_bot_10_i
```

```
193
    cmp_bot_8_i:
                                  BEQ
                                          $6 $12 cmp_top_7_i
194
                                  XORI
                                          $21 1
195
    cmp\_top\_7\_i:
                                  BEQ
                                          $6 $12 cmp_bot_8_i
196
                                  ADD
                                          $6 $9
197
                                  EXCH
                                          $12 $17
198
                                  BNE
                                          $10 $0 l_o_assert_true
    l_o_assert:
199
                                          $15 $9
                                  XOR
200
                                  SUB
                                          $15 $7
201
                                  BGEZ
                                          15 \text{ cmp\_bot\_4}
    cmp_top_3:
202
                                  XORI
                                          $16 1
203
                                  BGEZ
    cmp\_bot\_4:
                                          $15 \text{ cmp\_top\_3}
204
                                  XOR
                                          $10 $16
205
                                  BGEZ
    cmp_bot_4i:
                                          15 \text{ cmp-top-3-i}
206
                                  XORI
                                          $16 1
207
                                  BGEZ
                                          15 \text{ cmp-bot-}4_{i}
    cmp_top_3_i:
208
                                  ADD
                                          $15 $7
                                          $15 $9
209
                                  XOR
210
    l_malloc1_bot:
                                  BRA
                                          l_malloc1_top
211
    l_main_0_top:
                                  BRA
                                          l_main_0_bot
                                          $1 1
212
                                  ADDI
213
                                  EXCH
                                          $2 $1
                                  EXCH
                                          $3 $1
214
215
                                  ADDI
                                          \$1 -1
216
                                  SWAPBR $2
    l_main_0:
217
                                  NEG
                                          $2
218
                                          $1 1
                                  ADDI
                                  EXCH
                                          $3 $1
219
                                  EXCH
                                          $2 $1
220
221
                                  ADDI
                                          $1 -1
222
                                  ADD
                                          $6 $3
223
                                  ADDI
                                          $6 3
224
                                  EXCH
                                          $7 $6
                                  ADDI
                                          $6 -3
225
226
                                          $6 $3
                                  SUB
227
                                  XORI
                                          $8 100
228
                                  ADD
                                          $7 $8
229
                                  XORI
                                          $8 100
230
                                          $6 $3
                                  ADD
                                          $6 3
231
                                  ADDI
232
                                  EXCH
                                          $7 $6
                                          $6 -3
233
                                  ADDI
234
                                  SUB
                                          $6 $3
235
                                  XORI
                                          $6 1
                                          $6 $0 assert_13
236
    entry_11:
                                  BEQ
237
                                  ADD
                                          $7 $3
```

```
238
                                  ADDI
                                          $7 2
239
                                  EXCH
                                          $8 $7
240
                                  ADDI
                                          \$7 -2
241
                                  SUB
                                          $7 $3
242
    cmp_top_15:
                                  BNE
                                          $8 $0 cmp_bot_16
243
                                  XORI
                                          $9 1
244
    cmp_bot_16:
                                  BNE
                                          $8 $0 cmp_top_15
245
                                          $9 $0 f_bot_18
    f_{-}top_{-}17:
                                  BEQ
246
                                  XORI
                                          $10 1
247
    f_{bot_{1}}:
                                  BEQ
                                          $9 $0 f_top_17
248
                                  XOR
                                          $6 $10
249
                                  BEQ
                                          9 \ f_{top_{1}} - 17_{i}
    f_bot_18_i:
250
                                  XORI
                                          $10 1
                                          $9 $0 f_bot_18_i
251
                                  BEQ
    f_{top_{1}7_{i}}:
252
                                          $8 $0 cmp_top_15_i
    cmp_bot_16_i:
                                  BNE
253
                                  XORI
                                          $9 1
254
                                          88 \ 0 \ \text{cmp\_bot\_16\_i}
                                  BNE
    cmp_top_15_i:
255
                                  ADD
                                          $7 $3
                                          $7 2
256
                                  ADDI
257
                                  EXCH
                                          $8 $7
                                  ADDI
                                          $7 -2
258
                                          $7 $3
259
                                  SUB
260
                                  ADD
                                          $7 $3
                                          $7 2
261
                                  ADDI
                                          $8 $7
262
                                  EXCH
                                          \$7 -2
263
                                  ADDI
                                          $7 $3
264
                                  SUB
                                  ADD
                                          $9 $3
265
                                  ADDI
                                          $9 3
266
                                  EXCH
                                          $10 $9
267
268
                                  ADDI
                                          $9 -3
269
                                  SUB
                                          $9 $3
270
                                  BNE
                                          $8 $10 cmp_bot_20
    cmp\_top\_19:
271
                                          $11 1
                                  XORI
                                          $8 $10 cmp_top_19
272
    cmp_bot_20:
                                  BNE
273
                                  BEQ
                                          $11 $0 f_bot_22
    f_{top_21}:
274
                                  XORI
                                          $12 1
275
                                          11 \ f_{p_2} \ f_{p_3} \ 
    f_bot_22:
                                  BEQ
276
                                  XOR
                                          $6 $12
277
    f_bot_22_i:
                                  BEQ
                                          11 \ f_{top_21_i}
278
                                  XORI
                                          $12 1
279
                                          $11 $0 f_bot_22_i
    f_{top_{2}1_{i}}:
                                  BEQ
280
    cmp_bot_20_i:
                                  BNE
                                          $8 $10 cmp_top_19_i
281
                                  XORI
                                          $11 1
282
    cmp_top_19_i:
                                  BNE
                                          $8 $10 cmp_bot_20_i
```

```
283
                                          $9 $3
                                  ADD
284
                                  ADDI
                                          $9 3
285
                                  EXCH
                                          $10 $9
286
                                  ADDI
                                          $9 -3
287
                                  SUB
                                          $9 $3
                                          $7 $3
288
                                  ADD
                                          $7 2
289
                                  ADDI
290
                                  EXCH
                                          $8 $7
                                  ADDI
291
                                          \$7 -2
292
                                          $7 $3
                                  SUB
                                  BNE
                                          $6 $0 exit_14
293
     test_12:
294
                                  ADD
                                          $7 $3
295
                                  ADDI
                                          $7 2
296
                                  EXCH
                                          $8 $7
297
                                  ADDI
                                          \$7 -2
298
                                  SUB
                                          $7 $3
299
                                  XORI
                                          $9 1
300
                                  ADD
                                          $8 $9
                                  XORI
                                          $9 1
301
                                          $7 $3
302
                                  ADD
303
                                  ADDI
                                          $7 2
304
                                  EXCH
                                          $8 $7
                                  ADDI
305
                                          \$7 -2
306
                                  SUB
                                          $7 $3
                                  BRA
307
    assert_13:
                                          entry_11
308
                                  BRA
                                          test_12
    exit_14:
309
                                  XORI
                                          $6 1
                                  ADD
                                          $6 $3
310
311
                                  ADDI
                                          $6 2
312
                                  EXCH
                                          $7 $6
                                  ADDI
                                          \$6 -2
313
314
                                  SUB
                                          $6 $3
                                  ADD
                                          $8 $3
315
316
                                  ADDI
                                          $8 3
317
                                  EXCH
                                          $9 $8
318
                                  ADDI
                                          \$8 -3
319
                                  SUB
                                          $8 $3
320
                                  SUB
                                          $7 $9
                                  ADD
321
                                          $8 $3
322
                                  ADDI
                                          $8 3
323
                                  EXCH
                                          $9 $8
324
                                  ADDI
                                          \$8 -3
325
                                  SUB
                                          $8 $3
326
                                  ADD
                                          $6 $3
                                          $6 2
327
                                  ADDI
```

```
328
                                  EXCH
                                          $7 $6
329
                                  ADDI
                                          $6 -2
330
                                  SUB
                                          $6 $3
331
                                  ADD
                                          $6 $3
332
                                  ADDI
                                          $6 3
                                  EXCH
                                          $7 $6
333
334
                                  ADDI
                                          $6 -3
335
                                  SUB
                                          $6 $3
336
                                  XORI
                                          $8 100
                                          $7 $8
337
                                  SUB
                                          $8 100
338
                                  XORI
339
                                  ADD
                                          $6 $3
340
                                  ADDI
                                          $6 3
                                  EXCH
                                          $7 $6
341
342
                                  ADDI
                                          $6 -3
343
                                  SUB
                                          $6 $3
    l_main_0_bot:
                                  BRA
                                          l\_main\_0\_top
344
345
                                  BRA
    start:
                                          top
346
                                  START
                                  ADDI
347
                                          $4 393
348
                                  XOR
                                          $5 $4
349
                                  ADDI
                                          $5 10
                                          $7 $5
350
                                  XOR
351
                                  ADDI
                                          $4 10
                                  ADDI
352
                                          $4 -1
353
                                  EXCH
                                          $7 $4
                                  ADDI
                                          $4 1
354
                                  ADDI
355
                                          $4 -10
356
                                  XOR
                                          $1 $5
357
                                  ADDI
                                          $1 2048
                                  ADDI
                                          $1 -4
358
                                  XOR
                                          $3 $1
359
                                  XORI
                                          $6 3
360
                                          $6 $3
                                  EXCH
361
362
                                  ADDI
                                          $1 -1
363
                                  EXCH
                                          $3 $1
364
                                  ADDI
                                          $1 -1
                                  BRA
                                          l_main_0
365
                                  ADDI
                                          $1 1
366
367
                                  EXCH
                                          $3 $1
                                  ADDI
                                          $3 1
368
369
                                  ADDI
                                          $3 1
370
                                  EXCH
                                          $6 $3
                                  XORI
                                          $7 1
371
                                          $6 $7
372
                                  EXCH
```

```
373
                                 XORI
                                         $7 1
374
                                 ADDI
                                         $3 -1
375
                                 ADDI
                                         \$3 -1
376
                                 ADDI
                                         $3 1
377
                                 ADDI
                                         $3 2
378
                                 EXCH
                                         $6 $3
                                 XORI
                                         $7 2
379
380
                                 EXCH
                                         $6 $7
                                 XORI
                                         $7 2
381
382
                                 ADDI
                                         \$3 -2
                                 ADDI
                                         \$3 -1
383
                                 ADDI
384
                                         $1 1
385
                                 EXCH
                                         $6 $3
                                 XORI
386
                                         $6 3
387
                                 XOR
                                         $3 $1
388
                                 ADDI
                                         $1 4
389
                                 ADDI
                                         $1 -2048
390
                                 XOR
                                         $1 $5
                                 ADDI
                                         $5 -10
391
392
                                 XOR
                                         $5 $4
393
                                 ADDI
                                         $4 - 393
394
                                 FINISH
    finish:
```

9 Appendix C

prgReduced.pal

```
;; pendulum pal file
 2
   top:
                               BRA
                                       start
 3
   l_r_nodeCount:
                               DATA
                                       0
   l_r_limit:
                               DATA
                                       0
   l_Program_vt:
                               DATA
                                       214
                                       l_malloc_bot
6
   l_malloc_top:
                               BRA
 7
   l_malloc:
                               SWAPBR $2
8
                               NEG
                                       $2
9
                               ADDI
                                       $9 2
                                       $8 $0
10
                               XOR
11
                               ADDI
                                       $1 1
12
                               EXCH
                                       $6 $1
13
                               ADDI
                                       $1 1
                                       $7 $1
                               EXCH
14
                               EXCH
                                       $2 $1
15
                               ADDI
                                       \$1 -1
16
17
                               BRA
                                       l_malloc1
18
                               ADDI
                                       $1 1
19
                               EXCH
                                       $2 $1
20
                               EXCH
                                       $7 $1
21
                               ADDI
                                       $1 -1
22
                               EXCH
                                       $6 $1
23
                               ADDI
                                       $1 -1
24
                               XOR
                                       $8 $0
                                       $9 -2
25
                               ADDI
26
   l_malloc_bot:
                                       l_malloc_top
                               BRA
   l_malloc1_top:
                               BRA
                                       l_malloc1_bot
27
28
                               ADDI
                                       $1 1
29
                               EXCH
                                       $2 $1
30
                               SUB
                                       $17 $8
31
                               XOR.
                                       $17 $4
32
   l_malloc1:
                               SWAPBR $2
33
                               NEG
                                       $2
                                       $2 $1
34
                               EXCH
                                       \$1 -1
35
                               ADDI
36
                               XOR
                                       $17 $4
37
                               ADD
                                       $17 $8
38
                               EXCH
                                       $19 $17
39
                               XOR
                                       $18 $19
40
                               EXCH
                                       $19 $17
                               XOR
                                       $13 $9
41
```

```
42
                                SUB
                                        $13 $7
                                        13 \text{ cmp\_bot\_2}
43
   cmp_top_1:
                                BGEZ
44
                                XORI
                                        $14 1
45
                                BGEZ
                                        $13 cmp_top_1
   cmp\_bot\_2:
                                XOR
                                        $10 $14
46
47
                                BGEZ
                                        $13 cmp_top_1_i
   cmp\_bot\_2\_i:
48
                                XORI
                                        $14 1
49
                                BGEZ
                                        13 \text{ cmp-bot-}2_i
   cmp_top_1_i:
50
                                ADD
                                        $13 $7
                                XOR
51
                                        $13 $9
52
   l_o_test:
                                BEQ
                                        $10 $0 l_o_test_false
53
                                XORI
                                        $10 1
54
                                ADDI
                                        $8 1
                                EXCH
                                        $19 $17
55
                                XOR
                                        $18 $19
56
57
                                EXCH
                                        $19 $17
                                RL
                                        $9 1
58
59
                                EXCH
                                        $10 $1
                                ADDI
                                        $1 -1
60
                                EXCH
                                        $11 $1
61
62
                                ADDI
                                        $1 -1
                                EXCH
                                        $12 $1
63
                                ADDI
64
                                        $1 -1
                                EXCH
                                        $14 $1
65
                                ADDI
                                        $1 -1
66
                                EXCH
                                        $16 $1
67
                                ADDI
                                        $1 -1
68
                                EXCH
                                        $17 $1
69
70
                                ADDI
                                        \$1 -1
                                        $18 $1
71
                                EXCH
72
                                ADDI
                                        \$1 -1
73
                                EXCH
                                        $20 $1
                                ADDI
                                        $1 -1
74
75
                                EXCH
                                        $21 $1
                                        $1 -1
76
                                ADDI
77
                                EXCH
                                        $22 $1
78
                                ADDI
                                        $1 -1
79
                                EXCH
                                        $23 $1
                                ADDI
80
                                        $1 -1
81
                                BRA
                                        l_malloc1
82
                                ADDI
                                        $1 1
83
                                EXCH
                                        $23 $1
84
                                ADDI
                                        $1 1
                                EXCH
                                        $22 $1
85
86
                                ADDI
                                        $1 1
```

```
87
                                 EXCH
                                         $21 $1
88
                                 ADDI
                                         $1 1
89
                                 EXCH
                                         $20 $1
                                 ADDI
                                         $1 1
90
                                 EXCH
                                         $18 $1
91
92
                                 ADDI
                                         $1 1
                                 EXCH
                                         $17 $1
93
94
                                 ADDI
                                         $1 1
95
                                 EXCH
                                         $16 $1
96
                                 ADDI
                                         $1 1
                                 EXCH
                                         $14 $1
97
                                 ADDI
                                         $1 1
98
99
                                 EXCH
                                         $12 $1
                                 ADDI
                                         $1 1
100
                                 EXCH
                                         $11 $1
101
102
                                 ADDI
                                         $1 1
                                 EXCH
                                         $10 $1
103
104
                                 RR
                                         $9 1
105
                                 ADDI
                                         \$8 -1
                                 XORI
                                         $10 1
106
107
                                 BRA
    l_o_assert_true:
                                         l_oassert
108
    l_o_test_false:
                                 BRA
                                         l_o_test
109
    cmp_top_5:
                                 BEQ
                                         $18 $0 cmp_bot_6
110
                                 XORI
                                         $20 1
111
                                 BEQ
                                         $18 $0 cmp_top_5
    cmp_bot_6:
                                         $11 $20
112
                                 XOR
                                         18 \ cmp_top_5_i
113
    cmp\_bot\_6\_i:
                                 BEQ
                                         $20 1
114
                                 XORI
115
                                 BEQ
                                         $18 $0 cmp_bot_6_i
    cmp_top_5_i:
116
    l_i_{test}:
                                 BEQ
                                         $11 $0 l_i_test_false
                                 XORI
                                         $11 1
117
118
                                 ADD
                                         $6 $18
                                 SUB
                                         $18 $6
119
120
                                 EXCH
                                         $12 $6
121
                                 EXCH
                                         $12 $17
122
                                XOR
                                         $12 $6
123
                                 XORI
                                         $11 1
124
    l_i_assert_true:
                                 BRA
                                         l_i_assert
125
    l_i_test_false:
                                 BRA
                                         l_i_{\text{test}}
126
                                 ADDI
                                         $8 1
127
                                 RL
                                         $9 1
128
                                 EXCH
                                         $10 $1
129
                                 ADDI
                                         $1 -1
                                         $11 $1
130
                                 EXCH
                                 ADDI
                                         $1 -1
131
```

400	THE COLUMN TWO IS NOT	4.2 4.
132	EXCH	\$12 \$1
133	ADDI	\$1 -1
134	EXCH	\$14 \$1
135	ADDI	\$1 -1
136	EXCH	\$16 \$1
137	ADDI	\$1 -1
138	EXCH	\$17 \$1
139	ADDI	\$1 -1
140	EXCH	\$18 \$1
141	ADDI	\$1 -1
142	EXCH	\$20 \$1
143	ADDI	\$1 -1
144	EXCH	\$21 \$1
145	ADDI	\$1 -1
146	EXCH	\$22 \$1
147	ADDI	\$1 -1
148	EXCH	\$23 \$1
149	ADDI	\$1 -1
150	BRA	$l_malloc1$
151	ADDI	\$1 1
152	EXCH	\$23 \$1
153	ADDI	\$1 1
154	EXCH	\$22 \$1
155	ADDI	\$1 1
156	EXCH	\$21 \$1
157	ADDI	\$1 1
158	EXCH	\$20 \$1
159	ADDI	\$1 1
160	EXCH	\$18 \$1
161	ADDI	\$1 1
162	EXCH	\$17 \$1
163	ADDI	\$1 1
164	EXCH	\$16 \$1
165	ADDI	\$1 1
166	EXCH	\$14 \$1
167	ADDI	\$1 1
168	EXCH	\$12 \$1
169	ADDI	\$1 1
170	EXCH	\$11 \$1
171	ADDI	\$1 1
172	EXCH	\$10 \$1
173	RR	\$9 1
174	ADDI	\$8 -1
175	XOR	\$12 \$6
176	EXCH	\$12 \$17
	1	

```
177
                                 ADD
                                         $6 $9
178
    l_i_assert:
                                 BNE
                                          $11 $0 l_i_assert_true
179
                                 EXCH
                                         $12 $17
180
                                 SUB
                                         $6 $9
                                 BEQ
                                          $6 $12 cmp_bot_8
181
    cmp\_top\_7:
182
                                 XORI
                                         $21 1
                                          $6 $12 cmp_top_7
183
    cmp_bot_8:
                                 BEQ
                                 BNE
                                          $12 $0 cmp_bot_10
184
    cmp\_top\_9:
185
                                 XORI
                                         $22 1
186
    cmp_bot_10:
                                 BNE
                                          $12 $0 cmp_top_9
                                          $23 $21 $22
187
                                 ORX
                                          $11 $23
188
                                 XOR
189
                                 ORX
                                         $23 $21 $22
190
                                 BNE
                                          $12 $0 cmp_top_9_i
    cmp_bot_10_i:
191
                                         $22 1
                                 XORI
192
    cmp_top_9_i:
                                 BNE
                                          $12 $0 cmp_bot_10_i
193
    cmp_bot_8_i:
                                 BEQ
                                          $6 $12 cmp_top_7_i
194
                                 XORI
                                         $21 1
                                          $6 $12 cmp_bot_8_i
195
    cmp\_top\_7\_i:
                                 BEQ
                                         $6 $9
196
                                 ADD
197
                                 EXCH
                                         $12 $17
198
                                 BNE
                                          $10 $0 l_o_assert_true
    l_o_assert:
199
                                 XOR
                                          $15 $9
200
                                          $15 $7
                                 SUB
                                         $15 \text{ cmp\_bot\_4}
201
                                 BGEZ
    cmp_top_3:
202
                                 XORI
                                          $16 1
                                 BGEZ
203
    cmp\_bot\_4:
                                          $15 \text{ cmp\_top\_3}
204
                                 XOR
                                          $10 $16
205
                                 BGEZ
                                         15 \text{ cmp-top-3-i}
    cmp_bot_4i:
206
                                 XORI
                                          $16 1
207
    cmp_top_3_i:
                                 BGEZ
                                          15 \text{ cmp-bot-}4_{-i}
208
                                 ADD
                                          $15 $7
209
                                 XOR
                                          $15 $9
    l_malloc1_bot:
210
                                 BRA
                                          l_malloc1_top
211
    l_main_0_top:
                                 BRA
                                         l_main_0_bot
212
                                 ADDI
                                         $1 1
213
                                 EXCH
                                         $2 $1
                                 EXCH
                                          $3 $1
214
215
                                 ADDI
                                          \$1 -1
                                 SWAPBR $2
216
    l_main_0:
217
                                 NEG
                                         $2
218
                                 ADDI
                                         $1 1
219
                                 EXCH
                                          $3 $1
220
                                 EXCH
                                         $2 $1
221
                                 ADDI
                                          $1 -1
```

```
222
                                 EXCHI
                                         $7 $3 3
223
                                 XORI
                                         $8 100
224
                                 ADD
                                         $7 $8
225
                                 XORI
                                         $8 100
226
                                 EXCHI
                                         $7 $3 3
227
                                 XORI
                                         $6 1
228
                                 BEQ
                                         $6 $0 assert_13
    entry_11:
229
                                 EXCHI
                                         $8 $3 2
230
                                 OUT
                                         $8 $31
231
    iftop_3:
                                 BNE
                                         $8 $0 ifbot_3
232
                                 XORI
                                         $6 1
                                         $8 $0 iftop_3
233
                                 BNE
    if bot _{-}3:
234
                                 EXCHI
                                         $8 $3 2
235
                                 EXCHI
                                         $8 $3 2
236
                                 EXCHI
                                         $10 $3 3
237
    iftop_2:
                                 BNE
                                         $8 $10 ifbot_2
238
                                 XORI
                                         $6 1
239
                                 BNE
                                         $8 $10 iftop_2
    if bot _{-}2:
240
                                         $10 $3 3
                                 EXCHI
                                 EXCHI
241
                                         $8 $3 2
242
                                 BNE
                                         $6 $0 exit_14
    test_12:
                                         $8 $3 2
243
                                 EXCHI
244
                                 XORI
                                         $9 1
245
                                 ADD
                                         $8 $9
246
                                 XORI
                                         $9 1
247
                                 EXCHI
                                         $8 $3 2
248
    assert_13:
                                 BRA
                                         entry_11
249
    exit_14:
                                 BRA
                                         test_12
250
                                 XORI
                                         $6 1
251
                                 EXCHI
                                         $7 $3 3
                                 EXCHI
                                         $9 $3 3
252
253
                                 SUB
                                         $7 $9
                                 EXCHI
                                         $9 $3 3
254
255
                                 EXCHI
                                         $7 $3 3
256
                                 EXCHI
                                         $7 $3 3
                                 XORI
257
                                         $8 100
258
                                 SUB
                                         $7 $8
259
                                 XORI
                                         $8 100
260
                                 EXCHI
                                         $7 $3 3
                                 BRA
261
    l_main_0_bot:
                                         l_main_0_top
262
    start:
                                 BRA
                                         top
263
                                 START
264
                                 ADDI
                                         $4 314
265
                                 XOR
                                         $5 $4
                                 ADDI
266
                                         $5 10
```

0.0=	MOD	^- ^-
267	XOR	\$7 \$5
268	ADDI	\$4 10
269	ADDI	\$4 -1
270	EXCH	\$7 \$4
271	ADDI	\$4 1
272	ADDI	\$4 -10
273	XOR	\$1 \$5
274	ADDI	\$1 2048
275	ADDI	\$1 -4
276	XOR	\$3 \$1
277	XORI	\$6 3
278	EXCH	\$6 \$3
279	ADDI	\$1 -1
280	EXCH	\$3 \$1
281	ADDI	\$1 -1
282	ADDI	\$31 1
283	ADDI	\$30 -1
284	BRA	l_main_0
285	ADDI	\$31 -1
286	ADDI	\$30 1
287	ADDI	\$1 1
288	EXCH	\$3 \$1
289	ADDI	\$3 1
290	ADDI	\$3 1
291	EXCH	\$6 \$3
292	XORI	\$7 1
293	EXCH	\$6 \$7
294	XORI	\$7 1
295	ADDI	\$3 -1
296	ADDI	\$3 -1
297	ADDI	\$3 1
298	ADDI	\$3 2
299	EXCH	\$6 \$3
300	XORI	\$7 2
301	EXCH	\$6 \$7
302		
	XORI	\$7 2
303	ADDI	\$3 -2
304	ADDI	\$3 -1
305	ADDI	\$1 1
306	EXCH	\$6 \$3
307	XORI	\$6 3
308	XOR	\$3 \$1
309	ADDI	\$1 4
310	ADDI	\$1 -2048
311	XOR	\$1 \$5
I	~	

312 313			$\begin{array}{cccccccccccccccccccccccccccccccccccc$	
314		ADDI	4 -314	
315	finish:	FINISH		

10 Appendix D

$remove_redundancy.c$

```
1 #include <stdio.h>
 #include <stdlib.h>
 #include <string.h>
 6
 /* File & flow handling */
9
 struct Controlflow {
    int patternm;
10
    int pattern;
11
12
 } controller;
13
14
 struct Buffer {
15
    /* The current instruction line */
    char line [256];
16
17
    char param_set [5][32];
18
19
    /* Some list to split
20
     * an instruction line
21
     * into single paramters */
22
    char param_sets [5][5][32];
23
24
    /* Previous instruction lines */
25
    char previous_lines [5][256];
 } buffer;
26
27
28
 struct Filehandler {
    FILE *input_file;
29
    FILE *output_file;
30
31
 } filehandler;
 32
33
 34
 35
 /* functions */
36
37
 38 | int apply();
 int memory_exchange();
 int special_if_statements();
41
```

```
42
43
  /* File & flow handling */
44
  45
  int initialize(const char *file);
  int getinst();
46
  char* getparam(int idx);
47
  int saveline(int idx);
48
  int setPreviousLines();
49
50
  51
52
  /** main */
  53
  int main(int arg, char *args[])
54
55
      filehandler.input_file=fopen(args[1], "r");
56
57
      if (!filehandler.input_file) {
         printf("%s:_Unable_to_open_input_file_%s.\n","", args[1]);
58
59
         exit (1);
60
      fclose (filehandler.input_file);
61
62
      /* memory exchange */
63
64
      apply (1, args [1]);
      /* special if-statements */
65
      apply (2, "reduced.pal");
66
67
68
      return 0;
69
70
  int apply(int method, const char *file) {
71
72
      int action;
73
      initialize (file);
74
75
      if (method = 1)  {
76
         /* loop to load instruction memory */
         while(getinst()) {
77
78
            /* check for memory exchange pattern */
79
            action = memory_exchange();
80
81
            if(action = 0)
                fputs(buffer.line, filehandler.output_file);
82
83
84
      } else {
         while(getinst()) {
85
            /* check for special if-statements */
86
```

```
87
                 action = special_if_statements();
88
89
                 if(action == 0)
90
                      fputs (buffer.line, filehandler.output_file);
91
             }
92
93
        fclose (filehandler.input_file);
94
        fclose (filehandler.output_file);
95
96
97
    int memory_exchange() {
98
        int action = 0;
99
        /* reset on an add */
100
        if ( !strcasecmp(getparam(0), "ADD") ) {
101
102
             for(int i = 0; i < controller.pattern; i++)
103
                 fputs (buffer.previous_lines[i],
104
                      filehandler.output_file);
105
             controller.pattern = 0;
        }
106
107
         /* memory exchange start
108
109
          * save lines */
110
        if ( !strcasecmp(getparam(0), "ADD")
             | | controller.pattern > 0 | 
111
112
             saveline (controller.pattern);
113
             controller.pattern += 1;
114
             action = 1;
        }
115
116
        /* memory exchange end */
117
        if (!strcasecmp(getparam(0), "SUB")
118
            & controller.pattern = 5) {
119
120
             char buf [256];
121
122
             /* Place previous lines in
123
              * param_sets [0..4]
124
              * EXCH within param_sets[2]
125
              * ADDI within param_sets[1]
126
              * SUB within param_sets[4]
127
              */
128
129
             setPreviousLines();
130
131
             if (!strcasecmp(buffer.param_sets[2][0], "EXCH") ) {
```

```
132
                 strcpy (buffer.param_sets[2][4]
133
                     sprintf(buf, "%s%s_\%s_\%s\\n", buffer.param_sets[2][4]
134
135
                     ,"EXCHI"
                     , buffer . param_sets [2][1]
136
                     , buffer . param_sets [4][2]
137
                     , buffer . param_sets [1][2]);
138
                 printf("%s", buf);
139
140
141
                 fputs (buf, filehandler.output_file);
142
                 controller.pattern = 0;
                 action = 1;
143
            }
144
        }
145
146
147
        /* memory exchange final
148
         * if no pattern was regconized
         * print all five lines */
149
150
        if(controller.pattern = 5) {
            for(int i = 0; i < controller.pattern; i++)
151
152
                 fputs (buffer.previous_lines[i],
                 filehandler.output_file);
153
154
            controller.pattern = 0;
155
            action = 1;
        }
156
157
158
        return action;
159
160
    int special_if_statements() {
161
162
        int action = 0;
163
        if (strstr(getparam(0), "cmp") != NULL
164
165
            && controller.patternm == 0) {
166
            saveline (0); /* cmp */
            printf("%s%s\n", "cmp:", buffer.previous_lines[0]);
167
168
            controller.patternm = 1;
        }
169
170
        if (!strcasecmp(getparam(0), "XOR")
171
        & controller.patternm = 1) {
172
            saveline (1); /* xor */
173
            printf("%s%s\n", "xor:", buffer.previous_lines[1]);
174
175
        }
176
```

```
if (\text{getparam}(0) | \text{strlen}(\text{getparam}(0)) - 2| = 'i'
177
             && strstr(getparam(0), "cmp") != NULL
178
179
             & controller.patternm = 1) {
180
             char buf [256];
             char cls [32];
181
182
             setPreviousLines();
183
             strcpy(buffer.param_set[4], "_____");
184
             strcpy (buffer.param_set [2], "____");
185
186
187
             char *pp1 = strndup(buffer.param_sets[0][0],
                  strlen (buffer . param_sets [0][0]) - 1);
188
             char *pp2 = strndup(getparam(0), strlen(getparam(0)) - 3);
189
             char *pp3 = strndup(buffer.param_sets[0][4],
190
                  strlen (buffer.param_sets[0][4]);
191
             printf("%s \ \n", "pp1: \ \ ", pp1);
printf("%s \ \ \n", "pp2: \ \ ", pp2);
192
193
194
195
             if(!strcasecmp(pp1, pp2)) {
196
                  /* if */
                  sprintf(buf, "%s%s%s \_ \_ \_ _ %s _ %s _ %s \setminus n",
197
                       buffer.param_sets[0][0],
198
                      getparam (2),
199
200
                       buffer.param_sets[0][1],
201
                       buffer.param_sets[0][2],
202
                       buffer.param_sets[0][3],
                       buffer.param_sets[0][4]);
203
204
                  fputs (buf, filehandler.output_file);
205
206
                  /* xori */
                  sprintf(buf, "%s%s = 2%s = %s = n", getparam(4), "XORI",
207
                       buffer.param_sets[1][1], "1");
208
                  fputs (buf, filehandler.output_file);
209
210
211
                  /* fi */
                  sprintf(buf, "%s:\%s\%s \_ \_ \_ \%s \_ \%s \_ ", pp3, getparam(2),
212
213
                       buffer.param_sets[0][1],
214
                       buffer.param_sets[0][2],
215
                       buffer.param_sets[0][3],
216
                  fputs (buf, filehandler.output_file);
217
218
219
                  controller.patternm = 0;
220
             action = 1;
221
```

```
222
        }
223
224
        return action+controller.patternm;
225
226
227
    int initialize(const char *file) {
228
        /* copy input file into
229
         * tmp file */
        filehandler.input_file=fopen(file, "r");
230
231
        filehandler.output_file=fopen("tmp.pal", "w");
232
        while (getinst ()) {
             fputs (buffer.line, filehandler.output_file);
233
234
235
        fclose (filehandler.output_file);
236
        fclose (filehandler.input_file);
237
238
        filehandler.input_file = fopen("tmp.pal", "r");
        filehandler.output_file = fopen("reduced.pal", "w");
239
240
        /* make sure file is in valid pendulum format */
241
242
        /* get first line */
        fgets (buffer.line, 256, filehandler.input_file);
243
        if( strncmp(buffer.line, ";; pendulum_pal_file", 20) ) {
244
             /* compare with known header */
245
246
             printf("Input_file_not_in_in_Pendulum_pal_format.\n");
247
             exit (1);
248
249
        /* Initialize new file */
        sprintf(buffer.line, ";; _pendulum_pal_file \n");
250
251
        fputs(buffer.line, filehandler.output_file);
252
        return 0;
253
    };
254
255
    int getinst() {
256
        int r = fgets (buffer.line, 256, filehandler.input_file);
257
             */
258
259
        int fields=sscanf(buffer.line, "%s%s%s%s%s",
260
             buffer.param_set[0],
             buffer.param_set[1],
261
262
             buffer.param_set [2],
263
             buffer.param_set[3],
264
             buffer.param_set [4]);
265
266
        if ( fields == 0 || fields == EOF ) return 0;
```

```
267
268
        return r;
269
    };
270
271
    char* getparam(int idx) {
272
        return buffer.param_set[idx];
    }
273
274
275
    int saveline(int idx) {
        strcpy(buffer.previous_lines[idx], buffer.line);
276
277
        return 0;
278
    };
279
280
    int setPreviousLines() {
281
        for (int i = 0; i < 5; i++)
282
             sscanf(buffer.previous_lines[i], "%s%s%s%s%s",
283
             buffer.param_sets[i][0],
284
285
             buffer.param_sets[i][1],
286
             buffer.param_sets[i][2],
287
             buffer.param_sets[i][3],
             buffer.param_sets[i][4]);
288
289
290
        return 0;
291
    }
```

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