

# C Compiler for RiSC-16 Microprocessor

(Version 1.50)

[Afshin Sepehri](#), [Bruce Jacob](#)

Department of Electrical and Computer Engineering  
University of Maryland  
{afshin, blj}@eng.umd.edu

## Abstract

*A C compiler for the processor RiSC-16, a simple 16-bit RISC processor, is presented. This compiler covers main features of C programming language and can be used for writing large application for this processor. Such applications can implement intensive-computational programs or even operating systems. Architectural features of the system can be verified using these programs.*

## 1. Introduction

RiSC-16 is a 16-bit Ridiculously Simple Computer instruction set architecture, based on the Little Computer (LC-896) developed by Peter Chen at the University of Michigan. It is an 8-register, 16-bit computer. All addresses are short-word addresses. There are three machine-code instruction formats and a total of 8 instructions. The RiSC-16 is very simple, but it is general enough to solve complex problems [1].

The idea of writing a compiler for a high-level programming language for RiSC-16 returns to courses “Digital Computer Design” and “Micro-architecture” presented by Bruce Jacob in 2001 [2], [3]. In these courses RiSC-16 was used as a teaching ISA for implementing variant computer architectural features of the system. Different versions of the microprocessor supporting instruction decoding, pipelining, data forwarding, branch prediction, cache, virtual memory, user and kernel modes, super-pipelining were implemented using Verilog and C. In all these implementations, it was concluded that the performance of the system is completely benchmark dependent. It required having powerful and CPU-intensive benchmark applications to test the performance, power or other design goals as close to reality as possible. Writing programs in a high-level language such as C and compiling to the assembly language could considerably help this purpose.

The presented C compiler covers main features of C programming language. Meanwhile, it lacks of some of the features and some others are in progress. However, it is comprehensive enough to write large applications to satisfy the goals mentioned earlier.

Section 2 covers a brief review of the supported commands as well as unsupported ones. Section 3 shows an example program compiled by this compiler and its output in assembly.

## 2. C Supported Commands

Since RiSC-16 has a very limited ISA, it is used only for integer operations and as a consequence, “int” is the only data-type, which is supported by the compiler.

However, almost all the operations on this data-type are covered. The list of main operations and commands are followed:

1. Unary and binary arithmetic operations (+, -, \*, /, %)
2. Bit-wise operations (&, |, ~, ^, <<, >>)
3. Logical operations (&&, ||, !)
4. Relational operations (==, !=, <, >, <=, >=)
5. Pre and post increment and decrements (++ , --)
6. Assignment operations (=, +=, -=, \*=, /=, %=, &=, |=, ^=, <<=, >>=)
7. Question mark and comma operators (?, :, ,)
8. Structural flow control commands (if, if-else, for, while, do-while)
9. Unstructured flow control commands (goto, break, continue)
10. Function definition, call and return
11. Parameter passing by value and by reference through pointers
12. Pointer declaration and operations
13. One dimensional static array
14. Different variable scopes (global, local, formal)
15. Nested function call and recursive call
16. Complex and nested operations, function calls and blocks are supported.
17. Full preprocessing supports by having include, define, ifdef, ifndef, ....
18. Macro definition and call

The compiler also checks the syntax and semantic of the program based on the standard regular expression [6] and grammar [7] of C language. In case of finding any error or warning, compiler reports them by writing the exact line number and type of the error. Eighteen different types of errors are supported. Also, a detailed report on the type of generated instructions can be received using verbose switch (-v)

An additional feature in this compiler is capability of communication with user in run time. This is done by using `_CONSOLE` as the variable name, when writing expressions. In case of locating in an input position, the value is received by simulator in run-time. In case of output position, the result of the corresponding calculation is reported on screen. This feature is obviously done by a minor change in simulator code to make it ready to cooperate with compiler.

The output of the compiler is an assembly file of RiSC-16. This file can be compiled by the assembler of RiSC-16 [4]. The result binary code can be simulated by the simple RiSC-16 implementation [5] or any other implementation of this ISA.

Meanwhile, the compiler suffers a few limitations:

1. There is no linker stage in the compiler, so all the functions should be included in the same file or appended using “include” preprocessor as source files.
2. It does not support structures, unions and enumerators. Removing these powerful features can degrade the programming power but do not stop it. Moreover, these are parts of the compiler, which are in progress.
3. “switch” flow control is not supported, but it can be replaced by a couple of if-else blocks.
4. “auto” is the only supported storage class specifier. The other ones, “extern”, “static” and “register” are not supported.

### 3. Example

The following C program shows some of the supported features of the compiler. The compiled code of the example in RiSC-16 assembly is followed:

```

/*****/
/*
 * sample.h
 * written by Afshin Sepehri
 * August 2002
 */
/*****/
#ifndef SAMPLE_H
#define SAMPLE_H
/*****/
int Fact(int no) {
    if(no==0)
        return 1;
    else
        return( Fact(no-1)*no );
}
/*****/
#endif

/*****/
/*
 * sample.c
 * written by Afshin Sepehri
 * August 2002
 *
 * This is just a sample program to show
 * the compiler facilities. The final
 * result of the program is 1000.
 */
/*****/
#include "sample.h"
/*****/
#define MAX_POWER 10
#define MAX_SHIFT 5
/*****/
#define CALCULATE_MODE(a,b) ((a)%(b))
/*****/
int NO = 5;
/*****/
//#define COMPILE_THIS
/*****/
#ifdef COMPILE_THIS
    This part is not compiled!
    It is removed in preprocessing step
#endif
/*****/
int SigmaPower(int max, int power) {
    int m, n;
    int product;
    int result = 0;
    for(m=1;m<max && power<MAX_POWER;m++) {
        n=power-1;
        product = 1;
        do {

```

```

        product *= m;
        n--;
    } while(n>=0);
    result += product;
}
return(result);
}
/*****/
int Unknown() {
    int i=1;
    int result = 5;
    while( (result<=1)<1000 && i<MAX_SHIFT) {
        if(result==100 || i==5) {
            break;
        }
        i++;
    }
    return( (result & 0x100) ? 500 : result );
}
/*****/
void Exchange(int *pa, int *pb) {
    int temp;
    temp = *pa;
    *pa = *pb;
    *pb = temp;
}
/*****/
void InitArray(int *pi, int size) {
    int i;
    for(i=0;i<size;i++) {
        pi[i] = i+1;
    }
}
/*****/
int Summation(int ai[], int size) {
    int i;
    int sum = 0;
    for(i=0;i<size;i++) {
        sum+=ai[i];
    }
    return sum;
}
/*****/
void main() {
    int i=0, result=0, result2;
    int *presult;
    do {
        if(i==0)
            result += Fact(NO);
        else if(i==1) {
            int j;
            for(j=0 , result2=0;j<NO;j++)
                result2+=j;
            result += result2;
        }
        else if(i==2) {
            result += SigmaPower(NO, 3);
        }
        else if(i==3) {
            result += Unknown();
            continue;
        }
    }
}

```

```

    }
    else if(i==4) {
        result+=CALCULATE_MODE(result,100)+
            (result/100)*10;
    }
    else if(i==5) {
        int array[10];
        InitArray(array, 10);
        result += Summation(array, 10);
    }
    else {
        result += 435;
    }
} while(i++<6);
presult = &result;
Exchange(presult, &result2);
_CONSOLE = result2;
}
/*****/

```

Here is the assembly code generated by the compiler.

```

#-----
# RiSC-16 - C Compiler
# Version: 1.50
# Created by Afshin Sepehri
# 2001-02
#-----
                movi   r6, _main
                jalr   r0, r6
#-----
stack:          .space    512
#-----
_Fact:          sw     r6, r7, 0
                lw     r1, r7, -1
                addi  r2, r0, 0
                beq   r1, r2, zero1
                addi  r1, r0, -1
zero1:          addi  r1, r1, 1
                beq   r1, r0,
ifnot2
ifnot2:         beq   r0, r0, then2
                movi  r1, else2
                jalr  r0, r1
then2:          addi  r3, r0, 1
                add   r5, r0, r3
                lw    r6, r7, 0
                jalr  r0, r6
                movi  r4, endif2
                jalr  r0, r4
else2:          lw    r6, r7, -1
                addi  r1, r0, 1
                nand  r1, r1, r1
                addi  r1, r1, 1
                add   r6, r6, r1
                sw    r6, r7, 1
                addi  r7, r7, 2
                movi  r5, _Fact
                jalr  r6, r5
                addi  r7, r7, -2
                lw    r2, r7, -1
                lui   r3, 0x8000
                nand  r3, r3, r2
                nand  r3, r3, r3
                beq   r3, r0, pos3
                add   r3, r0, r0
                nand  r5, r5, r5
                addi  r5, r5, 1
neg3:           add   r3, r3, r5
                addi  r2, r2, 1
                beq   r2, r0,
endmul3
                beq   r0, r0, neg3
pos3:           beq   r2, r0,
endmul3
                add   r3, r3, r5
                addi  r2, r2, -1
                beq   r0, r0, pos3
endmul3:       add   r5, r0, r3
                lw    r6, r7, 0
                jalr  r0, r6
endif2:        lw    r6, r7, 0
                jalr  r0, r6
#-----
_NO:            .fill 5
#-----
_SigmaPower:   sw     r6, r7, 0
                addi  r4, r0, 0
                sw    r4, r7, 4
                addi  r5, r0, 1
                sw    r5, r7, 1
for4:          lw    r6, r7, 1
                lw    r1, r7, -2
                nand  r1, r1, r1
                addi  r1, r1, 1
                add   r6, r6, r1
                lui   r1, 0x8000
                nand  r6, r6, r1

```

```

nand r6, r6, r6
beq r6, r0, pos5
addi r6, r0, -1
pos5: lw r2, r7, -1
addi r3, r0, 10
nand r3, r3, r3
addi r3, r3, 1
add r2, r2, r3
lui r3, 0x8000
nand r2, r2, r3
nand r2, r2, r2
beq r2, r0, pos6
addi r2, r0, -1
pos6: beq r6, r0,
false7
add r6, r0, r2
beq r6, r0,
false7
addi r6, r0, 1
false7: beq r6, r0,
fornot4
beq r0, r0,
forbody4
fornot4: movi r6, endfor4
jalr r0, r6
forbody4: lw r4, r7, -1
addi r5, r0, 1
nand r5, r5, r5
addi r5, r5, 1
add r4, r4, r5
sw r4, r7, 2
addi r6, r0, 1
sw r6, r7, 3
dowhile8: lw r1, r7, 1
lw r2, r7, 3
lui r3, 0x8000
nand r3, r3, r2
nand r3, r3, r3
beq r3, r0, pos9
add r3, r0, r0
nand r1, r1, r1
addi r1, r1, 1
neg9: add r3, r3, r1
addi r2, r2, 1
beq r2, r0,
endmul9
beq r0, r0, neg9
pos9: beq r2, r0,
endmul9
add r3, r3, r1
addi r2, r2, -1
beq r0, r0, pos9
endmul9: sw r3, r7, 3
lw r4, r7, 2
addi r5, r4, -1
sw r5, r7, 2
addi r6, r0, 0
nand r6, r6, r6
addi r6, r6, 1
add r5, r5, r6
lui r6, 0x8000
nand r5, r5, r6
nand r5, r5, r5
beq r5, r0, pos10
addi r5, r0, -1
pos10: addi r5, r5, 1
beq r5, r0,
enddowhile8
movi r5, dowhile8
jalr r0, r5
enddowhile8: lw r1, r7, 4
add r3, r3, r1
sw r3, r7, 4
lw r2, r7, 1
addi r3, r2, 1
sw r3, r7, 1
movi r4, for4
jalr r0, r4
endfor4: lw r5, r7, 4
lw r6, r7, 0
jalr r0, r6
lw r6, r7, 0
jalr r0, r6
#-----
_Unknown: sw r6, r7, 0
addi r6, r0, 1
sw r6, r7, 1
addi r1, r0, 5
sw r1, r7, 2
while11: lw r2, r7, 2
addi r3, r0, 1
shift12: beq r3, r0,
endshift12
add r2, r2, r2
addi r3, r3, -1
beq r0, r0,
shift12
endshift12: sw r2, r7, 2
movi r4, 1000
nand r4, r4, r4
addi r4, r4, 1
add r2, r2, r4
lui r4, 0x8000
nand r2, r2, r4
nand r2, r2, r2
beq r2, r0, pos13
addi r2, r0, -1
pos13: lw r5, r7, 1
addi r6, r0, 5
nand r6, r6, r6
addi r6, r6, 1
add r5, r5, r6
lui r6, 0x8000
nand r5, r5, r6
nand r5, r5, r5
beq r5, r0, pos14
addi r5, r0, -1
pos14: beq r2, r0,
false15
add r2, r0, r5
beq r2, r0,
false15

```

```

false15:      addi  r2, r0, 1
whilenot11:   beq   r2, r0,
whilebody11:  beq   r0, r0,
whilenot11:   movi  r2,
endwhile11:   jalr  r0, r2
whilebody11:  lw    r1, r7, 2
              movi  r2, 100
              beq   r1, r2,
zerol6:       addi  r1, r0, -1
zerol6:       addi  r1, r1, 1
              lw    r3, r7, 1
              addi  r4, r0, 5
              beq   r3, r4,
zerol7:       addi  r3, r0, -1
zerol7:       addi  r3, r3, 1
              beq   r3, r0,
false18:      addi  r1, r0, 1
false18:      beq   r1, r0,
ifnot19:      beq   r0, r0,
then19:       movi  r1, endif19
ifnot19:      jalr  r0, r1
then19:       movi  r5, break20
ifnot19:      jalr  r0, r5
endif19:      lw    r6, r7, 1
              addi  r1, r6, 1
              sw    r1, r7, 1
              movi  r2, while11
              jalr  r0, r2
endwhile11:   nop
break20:      lw    r3, r7, 2
              movi  r4, 256
              nand  r3, r3, r4
              nand  r3, r3, r3
              beq   r3, r0,
ifnot21:      beq   r0, r0,
then21:       movi  r3, else21
ifnot21:      jalr  r0, r3
then21:       movi  r5, 500
              add   r6, r0, r5
              movi  r1, endif21
              jalr  r0, r1
else21:       lw    r2, r7, 2
              add   r6, r0, r2
endif21:      add   r5, r0, r6
              lw    r6, r7, 0
              jalr  r0, r6
              lw    r6, r7, 0
              jalr  r0, r6
#-----
_Exchange:    sw    r6, r7, 0
              lw    r3, r7, -2
              lw    r3, r3, 0
              sw    r3, r7, 1
              lw    r4, r7, -2
              lw    r5, r7, -1
              lw    r5, r5, 0
              sw    r5, r4, 0
              lw    r6, r7, -1
              lw    r1, r7, 1
              sw    r1, r6, 0
              lw    r6, r7, 0
              jalr  r0, r6
#-----
_InitArray:   sw    r6, r7, 0
              addi  r2, r0, 0
              sw    r2, r7, 1
for22:        lw    r3, r7, 1
              lw    r4, r7, -1
              nand  r4, r4, r4
              addi  r4, r4, 1
              add   r3, r3, r4
              lui   r4, 0x8000
              nand  r3, r3, r4
              nand  r3, r3, r3
              beq   r3, r0, pos23
              addi  r3, r0, -1
pos23:        beq   r3, r0,
fornot22:     beq   r0, r0,
forbody22:    movi  r3, endfor22
fornot22:     jalr  r0, r3
forbody22:    lw    r5, r7, -2
              lw    r6, r7, 1
              add   r5, r6, r5
              lw    r1, r7, 1
              addi  r2, r0, 1
              add   r1, r1, r2
              sw    r1, r5, 0
              lw    r3, r7, 1
              addi  r4, r3, 1
              sw    r4, r7, 1
              movi  r5, for22
              jalr  r0, r5
endfor22:     lw    r6, r7, 0
              jalr  r0, r6
#-----
_Summation:   sw    r6, r7, 0
              addi  r6, r0, 0
              sw    r6, r7, 2
              addi  r1, r0, 0
              sw    r1, r7, 1
for24:        lw    r2, r7, 1
              lw    r3, r7, -1
              nand  r3, r3, r3
              addi  r3, r3, 1
              add   r2, r2, r3
              lui   r3, 0x8000
              nand  r2, r2, r3
              nand  r2, r2, r2
              beq   r2, r0, pos25
              addi  r2, r0, -1

```

```

pos25:      beq    r2, r0,
fornot24
forbody24
fornot24:   movi   r2, endfor24
jalr    r0, r2
forbody24:  lw     r4, r7, -2
lw     r5, r7, 1
add    r4, r5, r4
lw     r4, r4, 0
lw     r6, r7, 2
add    r4, r4, r6
sw     r4, r7, 2
lw     r1, r7, 1
addi   r2, r1, 1
sw     r2, r7, 1
movi   r3, for24
jalr   r0, r3
endfor24:  lw     r4, r7, 2
add    r5, r0, r4
lw     r6, r7, 0
jalr   r0, r6
lw     r6, r7, 0
jalr   r0, r6
#-----
_main:     addi   r7, r0, stack
addi   r5, r0, 0
sw     r5, r7, 1
addi   r6, r0, 0
sw     r6, r7, 2
dowhile26: lw    r1, r7, 1
addi   r2, r0, 0
beq    r1, r2,
zero27
addi   r1, r0, -1
zero27:  addi   r1, r1, 1
beq    r1, r0,
ifnot28
beq    r0, r0,
then28
ifnot28:  movi   r1, else28
jalr   r0, r1
then28:  movi   r4, _NO
lw     r4, r4, 0
sw     r4, r7, 5
addi   r7, r7, 6
movi   r3, _Fact
jalr   r6, r3
addi   r7, r7, -6
lw     r6, r7, 2
add    r5, r5, r6
sw     r5, r7, 2
movi   r1, endif28
jalr   r0, r1
else28:  lw     r2, r7, 1
addi   r3, r0, 1
beq    r2, r3,
zero29
addi   r2, r0, -1
zero29:  addi   r2, r2, 1
ifnot30
beq    r2, r0,
beq    r0, r0,
then30
ifnot30:  movi   r2, else30
jalr   r0, r2
then30:  addi   r4, r0, 0
sw     r4, r7, 5
addi   r5, r0, 0
sw     r5, r7, 3
for31:   lw     r6, r7, 5
movi   r1, _NO
lw     r1, r1, 0
nand   r1, r1, r1
addi   r1, r1, 1
add    r6, r6, r1
lui    r1, 0x8000
nand   r6, r6, r1
nand   r6, r6, r6
beq    r6, r0, pos32
addi   r6, r0, -1
pos32:   beq    r6, r0,
fornot31
beq    r0, r0,
forbody31
fornot31:  movi   r6, endfor31
jalr   r0, r6
forbody31: lw    r2, r7, 5
lw     r3, r7, 3
add    r2, r2, r3
sw     r2, r7, 3
lw     r4, r7, 5
addi   r5, r4, 1
sw     r5, r7, 5
movi   r6, for31
jalr   r0, r6
endfor31: lw    r1, r7, 3
lw     r2, r7, 2
add    r1, r1, r2
sw     r1, r7, 2
movi   r3, endif30
jalr   r0, r3
else30:  lw     r4, r7, 1
addi   r5, r0, 2
beq    r4, r5,
zero33
addi   r4, r0, -1
zero33:  addi   r4, r4, 1
beq    r4, r0,
ifnot34
beq    r0, r0,
then34
ifnot34:  movi   r4, else34
jalr   r0, r4
then34:  movi   r1, _NO
lw     r1, r1, 0
sw     r1, r7, 6
addi   r2, r0, 3
sw     r2, r7, 7
addi   r7, r7, 8

```

_SigmaPower	movi r6,		lui r4, 0x4000
	jalr r6, r6	spos40:	nand r3, r3, r3
	addi r7, r7, -8		nand r4, r4, r4
	lw r3, r7, 2		nand r5, r3, r4
	add r5, r5, r3	div40:	addi r3, r0, 0
	sw r5, r7, 2		nand r4, r2, r2
	movi r4, endif34		addi r4, r4, 1
	jalr r0, r4		add r1, r1, r4
else34:	lw r5, r7, 1		lui r4, 0x8000
	addi r6, r0, 3		nand r4, r4, r1
	beq r5, r6,		nand r4, r4, r4
zero35		cont40	beq r4, r0,
	addi r5, r0, -1		beq r0, r0,
zero35:	addi r5, r5, 1	enddiv40	
	beq r5, r0,	cont40:	addi r3, r3, 1
ifnot36			beq r0, r0, div40
	beq r0, r0,	enddiv40:	add r1, r1, r2
then36			lui r3, 0x8000
ifnot36:	movi r5, else36		nand r3, r3, r5
	jalr r0, r5		nand r3, r3, r3
then36:	addi r7, r7, 6	endsign40	beq r3, r0,
	movi r1, _Unknown		nand r1, r1, r1
	jalr r6, r1		addi r1, r1, 1
	addi r7, r7, -6	endsign40:	lw r6, r7, 2
	lw r2, r7, 2		movi r2, 100
	add r5, r5, r2		lui r3, 0x8000
	sw r5, r7, 2		nand r3, r3, r6
	movi r3,		nand r3, r3, r3
continue37		fpos41	beq r3, r0,
	jalr r0, r3		nand r6, r6, r6
	movi r4, endif36		addi r6, r6, 1
	jalr r0, r4	fpos41:	lui r4, 0x8000
else36:	lw r5, r7, 1		nand r4, r4, r2
	addi r6, r0, 4		nand r4, r4, r4
	beq r5, r6,	spos41	beq r4, r0,
zero38			nand r2, r2, r2
	addi r5, r0, -1		addi r2, r2, 1
zero38:	addi r5, r5, 1	spos41:	lui r4, 0x4000
	beq r5, r0,		nand r3, r3, r3
ifnot39			nand r4, r4, r4
	beq r0, r0,	div41:	nand r5, r3, r4
then39			addi r3, r0, 0
ifnot39:	movi r5, else39		nand r4, r2, r2
	jalr r0, r5		addi r4, r4, 1
then39:	lw r1, r7, 2		add r6, r6, r4
	movi r2, 100		lui r4, 0x8000
	lui r3, 0x8000		nand r4, r4, r6
	nand r3, r3, r1		nand r4, r4, r4
	nand r3, r3, r3		beq r4, r0,
	beq r3, r0,	cont41	beq r0, r0,
fpos40			nand r1, r1, r1
	nand r1, r1, r1	enddiv41	addi r3, r3, 1
	addi r1, r1, 1	cont41:	beq r0, r0, div41
fpos40:	lui r4, 0x8000		add r6, r0, r3
	nand r4, r4, r2		lui r3, 0x8000
	nand r4, r4, r4	enddiv41:	nand r3, r3, r5
	beq r4, r0,		
spos40			
	nand r2, r2, r2		
	addi r2, r2, 1		

```

nand r3, r3, r3
lui r4, 0x4000
nand r4, r4, r5
nand r4, r4, r4
beq r4, r0,
divpos41
lui r4, 0x8000
divpos41: nand r5, r3, r3
nand r5, r5, r4
nand r4, r4, r4
nand r3, r3, r4
nand r4, r3, r5
beq r4, r0,
endsign41
nand r6, r6, r6
addi r6, r6, 1
endsign41: addi r2, r0, 10
lui r3, 0x8000
nand r3, r3, r2
nand r3, r3, r3
beq r3, r0, pos42
add r3, r0, r0
nand r6, r6, r6
addi r6, r6, 1
neg42: add r3, r3, r6
addi r2, r2, 1
beq r2, r0,
endmul42
beq r0, r0, neg42
pos42: beq r2, r0,
endmul42
add r3, r3, r6
addi r2, r2, -1
beq r0, r0, pos42
endmul42: add r1, r1, r3
lw r4, r7, 2
add r1, r1, r4
sw r1, r7, 2
movi r5, endif39
jalr r0, r5
else39: lw r6, r7, 1
addi r1, r0, 5
beq r6, r1,
zero43
addi r6, r0, -1
zero43: addi r6, r6, 1
beq r6, r0,
ifnot44
beq r0, r0,
then44
ifnot44: movi r6, else44
jalr r0, r6
then44: addi r3, r7, 6
sw r3, r7, 16
addi r4, r0, 10
sw r4, r7, 17
addi r7, r7, 18

movi r2,
_InitArray
jalr r6, r2
addi r7, r7, -18
addi r6, r7, 6
sw r6, r7, 16
addi r1, r0, 10
sw r1, r7, 17
addi r7, r7, 18
movi r5,
_Summation
jalr r6, r5
addi r7, r7, -18
lw r2, r7, 2
add r5, r5, r2
sw r5, r7, 2
movi r3, endif44
jalr r0, r3
else44: movi r4, 435
lw r5, r7, 2
add r4, r4, r5
sw r4, r7, 2
endif44: nop
endif39: nop
endif36: nop
endif34: nop
endif30: nop
endif28: nop
continue37: lw r6, r7, 1
addi r1, r6, 1
sw r1, r7, 1
addi r2, r0, 6
nand r2, r2, r2
addi r2, r2, 1
add r6, r6, r2
lui r2, 0x8000
nand r6, r6, r2
nand r6, r6, r6
beq r6, r0, pos45
addi r6, r0, -1
pos45: beq r6, r0,
enddowhile26
movi r6, dowhile26
jalr r0, r6
enddowhile26: addi r3, r7, 2
sw r3, r7, 4
lw r5, r7, 4
sw r5, r7, 16
addi r6, r7, 3
sw r6, r7, 17
addi r7, r7, 18
movi r4, _Exchange
jalr r6, r4
addi r7, r7, -18
lw r1, r7, 3
sw r1, r0, 0
halt
#-----

```

## References

1. <http://www.enee.umd.edu/courses/enee646.F2001/RiSC-isa.pdf>
2. <http://www.enee.umd.edu/courses/enee646.F2001/>
3. <http://www.enee.umd.edu/courses/enee759m.S2002/>
4. <http://www.enee.umd.edu/courses/enee759m.S2002/RiSC/a.c>
5. <http://www.enee.umd.edu/courses/enee646.F2001/RiSC.c>
6. <http://www.lysator.liu.se/c/ANSI-C-grammar-l.html>
7. <http://www.lysator.liu.se/c/ANSI-C-grammar-y.html>