

コンパイラの間表現

—中間言語とアセンブリの相互変換を検証する—

野村 直樹

Agenda

◇自己紹介	3
◇はじめに	4
◇コンパイル過程の例	5
◇デコンパイラ	6
◇デコンパイルの難易度とその理由	7
◇本研究の特徴	8
◇中間言語の設計	9
◇C I T（中間言語）の特徴	10
◇Cに対応するC I T（1）	11
◇Cに対応するC I T（2）	12
◇中間言語、目的コード、実行結果の関係	13
◇C I Tのサンプル（ファイル作成）	14
◇C I Tコンパイラの作成	15
◇C I TからR I S C－Vアセンブリが作成可能であるかの検証	16
◇C I Tコンパイルしたアセンブリの例	17
◇Q E M U上での実行例	18
◇R I S C－VアセンブリからC I Tが作成可能であるかの検証	19
◇アセンブリからC I T作成の例	20
◇R I S C－VアセンブリからC I T変換可能なこと	22
◇まとめ	23
◇参考文献、使用ツール、ソフトウェア	24

自己紹介

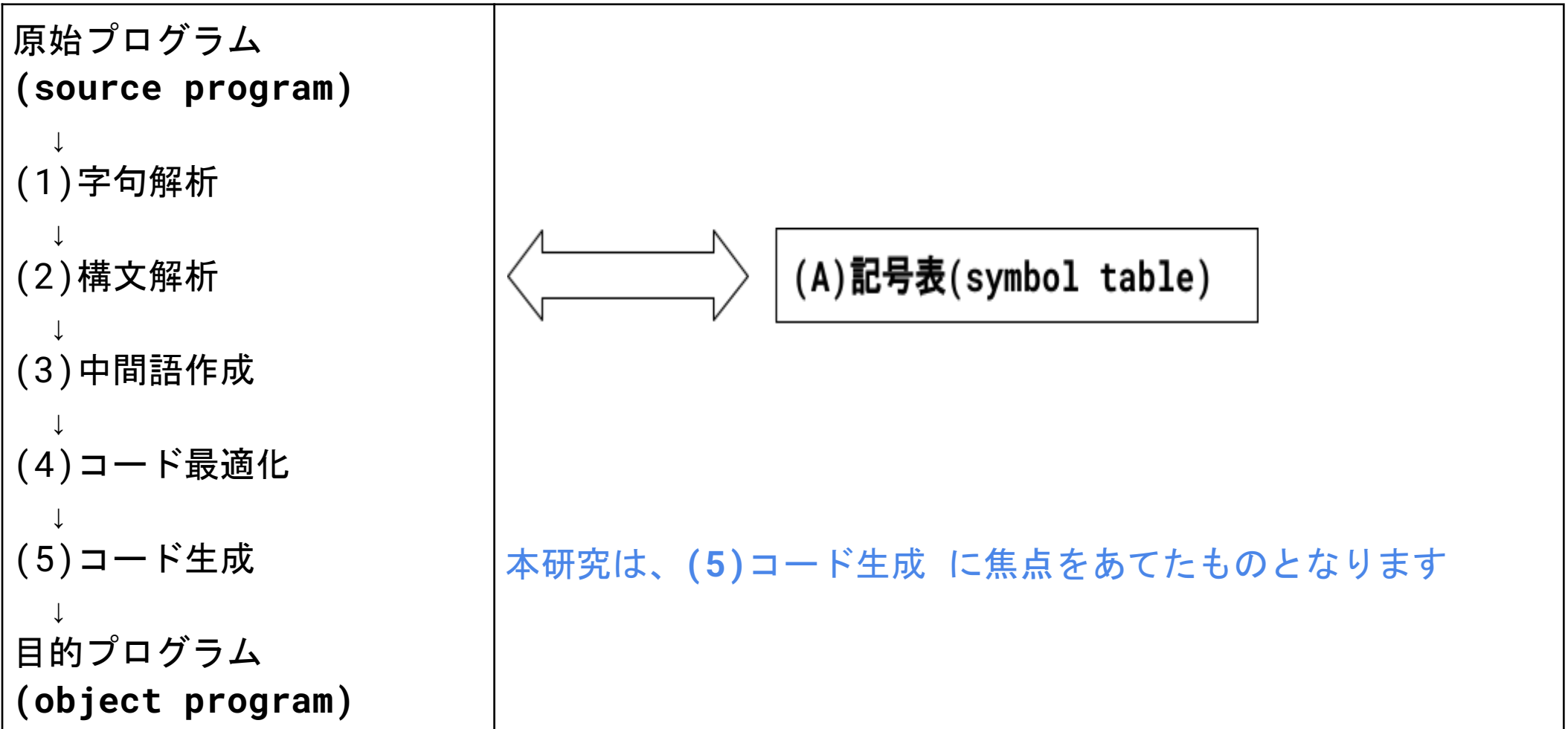
◆名前：	野村直樹
◆所属：	自然と環境コース
◆仕事：	埼玉県久喜市在住のフリーのプログラマー。業務システムからスマホゲームまで対応可。会社設立もしました。新人研修IT講師の経験もあり(3か月間のみ)。参加したスマホゲームプロジェクトの中で一番ヒットしたゲームは、ディバインゲート(配信ガンホー・オンライン・エンターテイメント)。アニメ化もされました。
◆放送大学に再入学し卒業研究の履修に至るまで：	医療ミスで母が亡くなり、医療訴訟に至りました。医療事故調査制度センター調査へ質問中。それに加え、新型コロナウイルスの感染状況と健康上の理由により、しばらく仕事を休むことを選択し、会社を休眠させ、放送大学に再入学。いろいろとモチベーションをなくしていましたが、ゲーム開発、コンパイラ開発の興味はなくなっておらず、今回、履修を考えました。

はじめに

逆コンパイルを理解するには、まずコンパイラをより理解し、扱いやすい中間表現があればと考えました。中間言語は、中間表現が言語の形態をしているものです。

実行形式を元の高レベル言語に戻すというのは難しいとされています。その為、実マシン上での実行ファイルから逆コンパイルする前段階として、より易しいと思われる中間言語とアセンブリの相互変換を検証しました。本研究は、中間言語からのコード生成部分に焦点をあてたものとなります。まず中間言語を設計し、それを対象にRISC-Vアセンブリを出力するコンパイラを作成することを試みました。それに加え、RISC-Vアセンブリから中間言語への変換がどの程度可能であるかを検証しました。

コンパイル過程の例



文献[疋田 88, p3]の図1.3を参考にしたもの

デコンパイラ

- ・ コンパイラの処理とまったく逆のことを行うのがデコンパイラ（逆コンパイラともいう）です。
- ・ 使用される主な領域は、ソフトウェアのメンテナンスとセキュリティなどがあげられます。
- ・ 完全に元どおりのソースコードが得られることは稀であるとされています。
- ・ 対象がアセンブリ言語の場合、逆コンパイラは、データを命令から分離する問題や、変数やサブルーチンの命名の問題に対処する必要がないです。（アセンブリ言語には、データセグメント、データと型の宣言、サブルーチン名、サブルーチン開始位置、サブルーチン終了ステートメントなど、記号テキストの形式で役立つデータ情報があります [Cifuentes94, p17]）

デコンパイルの難易度とその理由

対象	難易度	理由 <small>[No1an10, p23-p60 p127-p132]</small>
Javaクラスファイル	易しい	1) 変数名、メソッド名がクラスファイルに含まれている 2) 命令数が少ない 3) 単純なスタックマシン(レジスタがない) 4) データと命令が分離している
実行ファイル (実マシン) 例 MS-DOS COMファイル UNIX a.outファイル UNIX ELFファイル	難しい	1) 異なるプラットフォームどころか、同じマシン上でもコンパイラが違えば全く違う出力が生成される 2) レジスタがある 3) 実行形式の出力はまったく定義されておらず、データと命令が分離していない 4) リンカも考えなくてはならない

本研究の特徴

- 本研究の目的は、中間言語とアセンブリの相互変換を検証することとなります（実マシン上での実行ファイルから逆コンパイルする前段階として）
- 中間言語からのコード生成部分に焦点をあてたものとなります
- 独自の中間言語のフォーマットを設計。これをCITと命名しました
- C I Tを対象にRISC-Vアセンブリを出力するコンパイラを開発
- サンプルのC I Tファイル作成し、開発したコンパイラでRISC-Vアセンブリを出力
- RISC-VアセンブリからC I Tへの変換がどの程度可能であるかを検証（机上での確認）

中間言語の設計

本研究では、中間言語 C I T を設計します。C I T は、C 言語、CINC (ほぼ C の独自言語) からの変換を想定していて、C と Lisp の言語仕様^{[KR88][小西 88]}を参考にし、いくつかの言語機能を取り去ったものです。以下は、C I T 構文規則 [卒業研究報告書, p12] の一部となります。

<pre><program> →<devers> <defuns> <devers> →{ <struct> <union> <dever> } <defuns> →{ <defun> } <defun> → (DEFUN <funcname> <type> <defgs> <funcblock>) <funcname> →<identifier></pre>	<pre><funcblock> → "{" <devers> <statementlist> "}" <block> → "{" <statementlist> "}" <statementlist> → { <statement> } <statement> → <if> <switch> <while> <dowhile> <for> <expression> <return> <break> <continue> <label> <goto> <expression> → <op></pre>	<pre><struct> → (STRUCT <usertype> {<demember>}1) <union> → (UNION <usertype> {<demember>}1) <usertype> → <identifier> <demember> → (<memname> [<type> <bittype>]) . . 省略 . .</pre>
--	---	---

[URL] https://github.com/Naoiki-Nomura/CIT/blob/main/20231011A/CIT_syntax_rules_20231113A.pdf

C I T (中間言語)の特徴

- C言語の言語機能をなるべく実現(C言語からの変換を想定)
- 構文木をテキストで表現
- 優先順位は、左から右で、かっこ内が優先される。(型指定も左から右となる)
- 型の種類は、int型、char型、float型、double型、構造体型、共用体型、ポインタ型、配列型である。(unsigned型, short型, long型はない)
- グローバル変数、ローカル変数に対応。
- ディレクティブ(プリプロセッサ)は使用できない
- static, extern, volatileは使用できない
- 可変長引数は使用できない

Cに対応するC I T (1)

```
[C]
int  a,b,c;
int  answer1;
int  answer2;
main()
{
    answer1 = 0;
    answer2 = 0;
    a = 0;
    while (a < 2){
        b = 0;
        while (b < 2){
            c = 0;
            while (c < 2){
                answer1 = answer1 + 1;
                c=c+1;
            }
            b=b+1;
        }
        a=a+1;
    }
}
```

```
[CIT]
(DEVER      a  int)
(DEVER      b  int)
(DEVER      c  int)
(DEVER      answer1  int)
(DEVER      answer2  int)
(DEFUN      main void
{
    (T(=)     answer1      0)
    (T(=)     answer2      0)
    (T(=)     a           0)
    (WHILE    (T(<) a       2)
    {
        (T(=)  b           0)
        (WHILE (T(<) b       2)
        {
            (T(=) c           0)
            (WHILE (T(<) c       2)
            {
                (T(=)  answer1  (T(+)  answer1  1))
                (T(=)   c  (T(+) c  1))
            }
        )
        (T(=) b           (T(+) b  1))
    }
    )
    (T(=)  a           (T(+) a  1))
    }
    )
}
```

Cに対応するC I T (2)

```
[C]
//
// 文献[KR88 ,p122]に、記述されているものを、ほぼそのまま記述した
// C言語ソースとなる。文献[KR88 ,p122-126]は、CIT型定義の参考とした。
//
main()
{
    char **argv;           // argv: pointer to pointer to char

    int (*daytab1) [13];    // daytab: pointer to array[13] of int

    int *daytab2[13];       // daytab: array[13] of pointer to int

    void *comp1();          // comp: function returning pointer to void

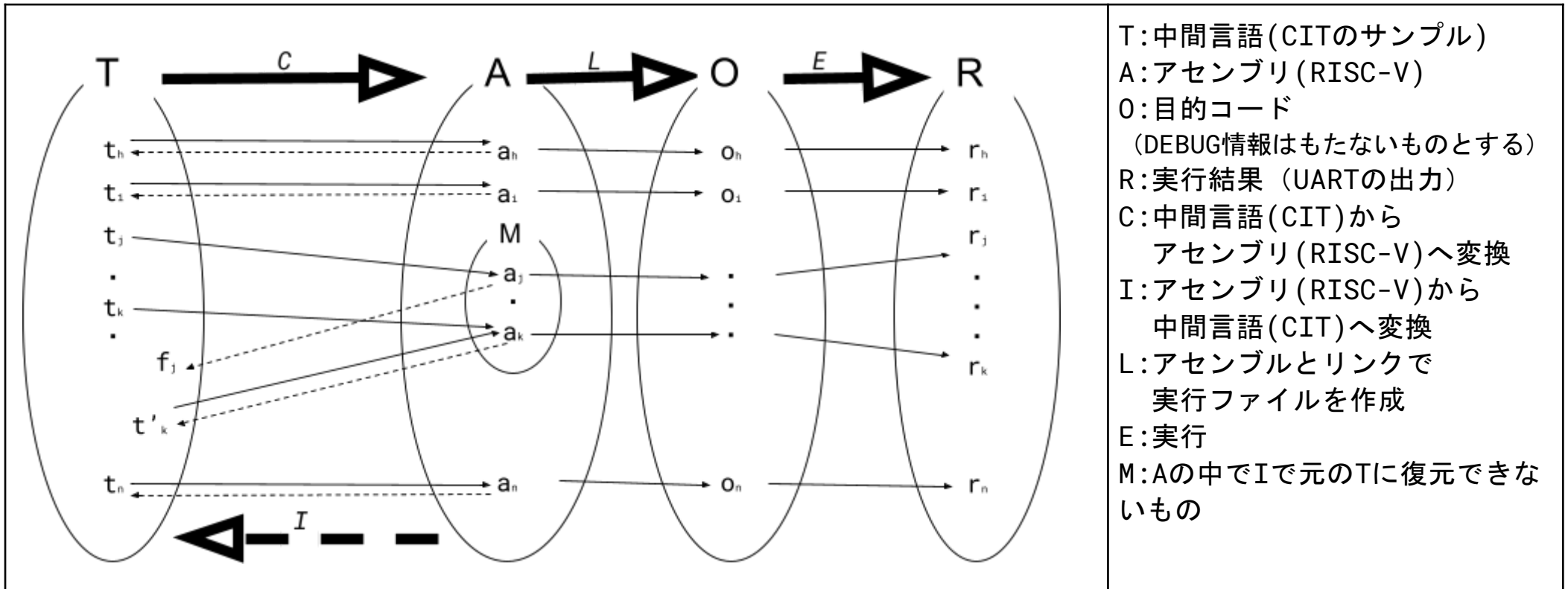
    void (*comp2)();        // comp: pointer to function returning void

    char ((*x1())[1])();    // x: function returning pointer to array[] of
                           // pointer to function returning char

    char ((*x2[3])())[5];   // x: array[3] of pointer to function returning
                           // pointer to array[5] of char
}
```

```
[CIT]
(DEFUN      main void
{
    (DEVER argv      **char)
    (DEVER daytab1    *[13]int)
    (DEVER daytab2    [13]*int)
    (DEVER comp1      ()*void)
    (DEVER comp2      *()*void)
    (DEVER x1         ()*[]*()*char)
    (DEVER x2         [3]*()*[5]char)
})
```

中間言語、目的コード、実行結果の関係



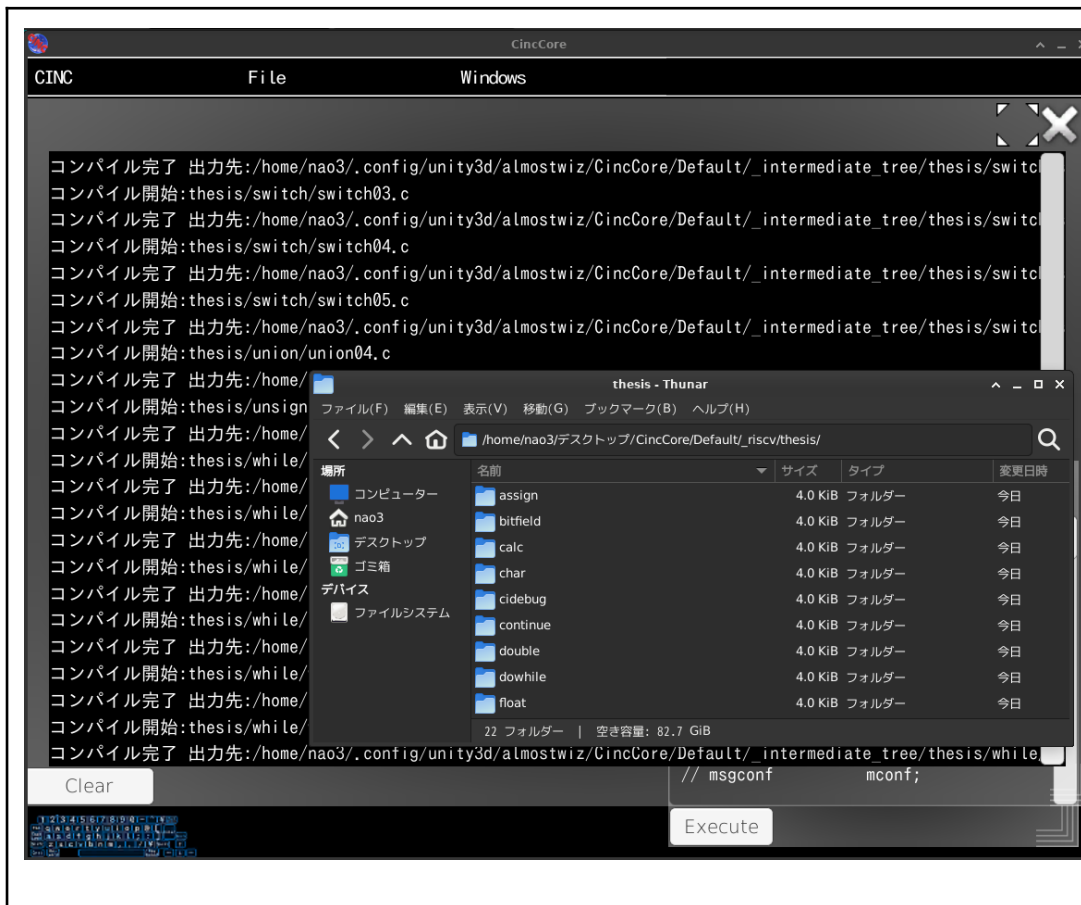
- ・ CによりTからAが生成される。LによりOが生成されOを実行(E)すると実行結果としてRが生成される。Cの逆の機能をIとする。
- ・ Iを実行すると中間言語への復元は、以下のパターンになると予測しました。
 - ①完全に復元可能(a_h は t_h へ変換)。
 - ②完全ではないが復元可能(a_k は t'_k へ変換)。再度、Cを行ってみると、 t'_k から a_k が生成された。
 - ③復元できない(a_j は f_j へ変換)。再度、Cを行ってみると、 f_j から a_j が生成されなかった。
- ・ 仮説検証の方法は、 $t_1 \sim t_n$ にあたるCITを実際に作成し、C、Iが可能であるかの検証を行う。

C I Tのサンプル(ファイル作成)

No	主な検証内容	[別名]ファイル名
1	assign	[t ₁]andassign01.ct [t ₂]assign01.ct [t ₃]divassign01.ct [t ₄]leftassign01.ct [t ₅]minusequal.ct [t ₆]modassign01.ct [t ₇]mulassign01.ct [t ₈]orassign01.ct [t ₉]plusequal.ct [t ₁₀]rightassign01.ct [t ₁₁]xorassign01.ct
2	bitfield	[t ₁₂]bitfield01.ct
3	char	[t ₁₃]char01.ct [t ₁₄]char01.ct [t ₁₅]param01.ct
4	double	[t ₁₆]param01.ct
5	dowhile	[t ₁₇]dowbreak01.ct
6	float	[t ₁₈]float_calculate07.ct [t ₁₉]float00b.ct [t ₂₀]float00c.ct [t ₂₁]float01.ct [t ₂₂]float02.ct [t ₂₃]param01.ct
7	func	[t ₂₄]funcarg01.ct [t ₂₅]funcarg02.ct [t ₂₆]funcorder.ct [t ₂₇]funcorder02.ct
8	goto	[t ₂₈]goto01.ct [t ₂₉]goto02.ct [t ₃₀]goto03.ct
9	if	[t ₃₁]if_and_or01.ct [t ₃₂]if_and01.ct [t ₃₃]if_and02.ct [t ₃₄]if_or_and01.ct [t ₃₅]if_or01.ct [t ₃₆]if_or02.ct [t ₃₇]if01.ct [t ₃₈]if02.ct [t ₃₉]int_if_and_to.ct
10	initialization	[t ₄₀]initialization04c.ct [t ₄₁]initialization04c1.ct [t ₄₂]initialization04c2.ct [t ₄₃]initialization04c3.ct [t ₄₄]initialization04c4.ct
11	pointer	[t ₄₅]pointer_func01.ct [t ₄₆]pointer_func03.ct [t ₄₇]pointer_func06.ct
12	struct	[t ₄₈]vector01.ct [t ₄₉]vector02.ct
13	switch	[t ₅₀]switch01.ct [t ₅₁]switch02.ct [t ₅₂]switch03.ct [t ₅₃]switch04.ct [t ₅₄]switch05.ct
14	while	[t ₅₅]continue01.ct [t ₅₆]continue02.ct [t ₅₇]eachloop01.ct [t ₅₈]eachloop02.ct [t ₅₉]while_break01.ct [t ₆₀]while01.ct

CITコンパイラの作成

CITのサンプルに対応するC言語ソースコードをGCC^[gcc0rg]でコンパイルしたアセンブリとRISC-V原典^[DA18]を参考に、RISC-Vアセンブリを出力するCITコンパイラを作成しました。現時点では、未実装な部分もあり、機能も乏しく完全には実装できていません。以下は、Linux上でCITコンパイラを実行したものです。Cソースを読み込み、中間言語としてCITを出力し、再度CITを読み込み、RISC-Vアセンブリを出力しています。



- ・当初、C言語で新規作成し、CITに変換し、将来的にCITコンパイラ自身もCITコンパイル可能な方向性を考えていましたが、時間がなかった為、自分が以前開発したアプリを流用し、作成することにしました。(ゲームエンジンUnity^[uniive]、C#^[visads]を使用)

- ・コード生成の部分的な例は、[卒業研究報告書, p25-p32]を参考にしてください。

動作状況の動画[URL] <https://www.youtube.com/watch?v=M-nL1nDzCrQ>

CITからRISC-Vアセンブリが作成可能であるかの検証

・CITコンパイラでサンプルのCITファイル(全60ファイル)をコンパイルしアセンブリ出力可能を確認。結果をgithubへアップロードしました。

サンプルCIT	[URL] https://github.com/Naoiki-Nomura/CIT/tree/main/20231011A/_intermediate_tree
アセンブリ出力結果	[URL] https://github.com/Naoiki-Nomura/CIT/tree/main/20231011A/_riscv
コンパイラ動作状況	[URL] https://www.youtube.com/watch?v=M-nLlnDzCrQ

- ・Linux_[jpuoad]上のGCC_[gccOrg]でアセンブル、実行ファイル作成を確認
- ・CITとほぼ同等なC言語ソースをGCC_[gccOrg]でコンパイル、アセンブリ出力した結果と比較し同等であると確認(レジスタ割り当てなどが一部、違うところがあるがほぼ同等と机上で確認)
- ・UARTでデバッグ表示し、実行上も問題ないことを確認(ただし2023年11月)

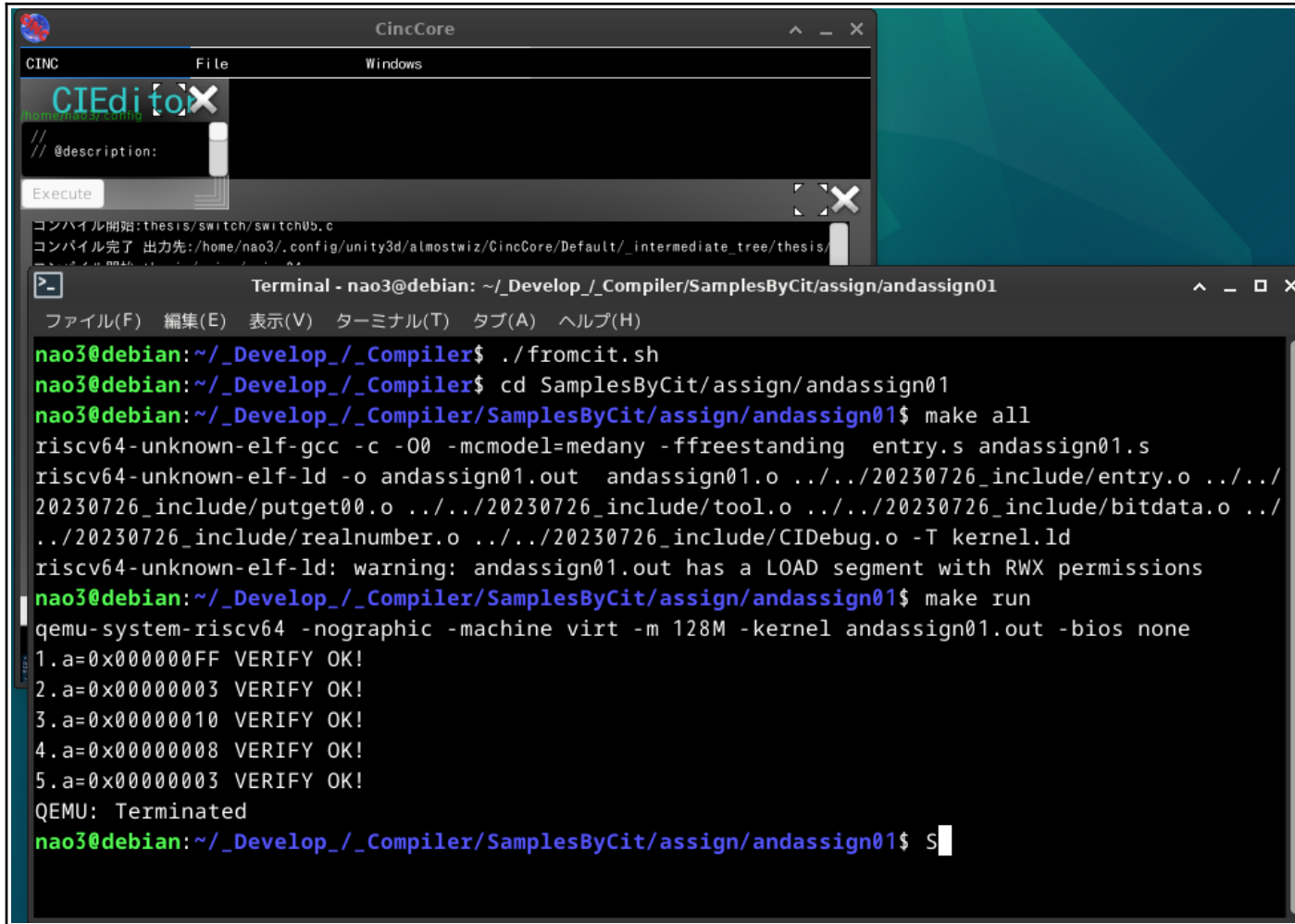
CITコンパイラは、全体的にGCC_[gccOrg]で出力されるアセンブリより読みやすいアセンブリを出力(その反面、無駄なコードがある)するようにしました。また机上で流れを理解しやすくなるように、コメント、ラベルも工夫しています。

C I Tコンパイルしたアセンブリの例

andassign01.ct (T_1)	andassign01.s (a_1)
<pre> (DEFUN main void { (DEVER a int) (DEVER b int) (DEVER c int) (DEVER ap *int) (DEVER bp *int) (T(=) a 0xff) (T(=) b 0x03) (T(=) ap a&) (T(=) bp b&) (T(=) a (T(&) a 0x03)) (T(=) a 0x00ff) (T(=) b 0x1110) (T(=) a (T(&) a b)) (T(=) a 0x000f) (T(=) b 0xffff8) (T(=) ap* (T(&) ap* bp*)) (T(=) a 0x000f) (T(=) b 0xffff0) (T(=) c 0x0003) (T(=) ap* (T(&) ap* (T(+) bp* c))) }) </pre>	<pre> .file "andassign01.ct" .option nopic .attribute arch, "rv64i2p0_m2p0_a2p0_f2p0_d2p0" .attribute unaligned_access, 0 .attribute stack_align, 16 .text .align 2 .globl main .type main, @function main: addi sp, sp, -48 sd s0, 40(sp) addi s0, sp, 48 . 省略 . nop ld s0, 40(sp) addi sp, sp, 48 jr ra .size main, .-main .ident "CIT: () 0.1.0" </pre>

QEMU上での実行例

以下は、CITコンパイラを実行し、出力したアセンブリ([a₁]andassign01.s)をアセンブル、リンク、実行ファイルを生成し、QEMU_[wwworg]上で実行したものととなります。[\[2~3分程度あれば、実演可能です\]](#)



```
CincCore
CINC File Windows
CIEditoX
// @description:
Execute
コンパイル開始:thesis/switch/switch05.c
コンパイル完了 出力先:/home/nao3/.config/unity3d/almostwiz/CincCore/Default/_intermediate_tree/thesis/
Terminal - nao3@debian: ~/_Develop/_Compiler/SamplesByCit/assign/andassign01
ファイル(F) 編集(E) 表示(V) ターミナル(T) タブ(A) ヘルプ(H)
nao3@debian:~/_Develop/_Compiler$ ./fromcit.sh
nao3@debian:~/_Develop/_Compiler$ cd SamplesByCit/assign/andassign01
nao3@debian:~/_Develop/_Compiler/SamplesByCit/assign/andassign01$ make all
riscv64-unknown-elf-gcc -c -O0 -mcmodel=medany -ffreestanding entry.s andassign01.s
riscv64-unknown-elf-ld -o andassign01.out andassign01.o ../../20230726_include/entry.o ../../
20230726_include/putget00.o ../../20230726_include/tool.o ../../20230726_include/bitdata.o ../
../20230726_include/realnumber.o ../../20230726_include/CIDebug.o -T kernel.ld
riscv64-unknown-elf-ld: warning: andassign01.out has a LOAD segment with RWX permissions
nao3@debian:~/_Develop/_Compiler/SamplesByCit/assign/andassign01$ make run
qemu-system-riscv64 -nographic -machine virt -m 128M -kernel andassign01.out -bios none
1.a=0x000000FF VERIFY OK!
2.a=0x00000003 VERIFY OK!
3.a=0x00000010 VERIFY OK!
4.a=0x00000008 VERIFY OK!
5.a=0x00000003 VERIFY OK!
QEMU: Terminated
nao3@debian:~/_Develop/_Compiler/SamplesByCit/assign/andassign01$ S
```

- 1)CITコンパイラを実行でアセンブリを出力
- 2)fromcit.sh でアセンブリを、それぞれの実行用ディレクトリへコピーする
- 3)cd でandassign01の実行ディレクトリに移動
- 3)make all でandassign01.s をアセンブルし、実行ファイルを生成
- 4)make run でデバッグ有り(UART)の実行ファイルをQEMU上で実行
- 5)Ctrl-A X でQEMUを終了

実行手順の動画[URL]

<https://www.youtube.com/watch?v=AH1WWY1GWPC>

RISC-VアセンブリからCITが作成可能であるかの検証

アセンブリ出力されたものを机上でCITに復元可能であるかを確認しました。全60ファイル中、48ファイルは、復元可能(①、②)でした。検証した中では、「③復元できない」は、みつかっていません。ただし、締め切り上の都合などにより、今回、検証できなかったものが12ファイルあります。

主な検証内容	[別名]ファイル名
①完全に復元できた (a_h は t_h へ変換)	[a ₂₀]float00c.s [a ₅₀]switch01.s [a ₅₁]switch02.s [a ₅₂]switch03.s [a ₅₃]switch04.s [a ₅₄]switch05.s [a ₅₅]continue01.s [a ₅₇]eachloop01.s
②完全な復元はできない (a_k は t'_k へ変換)	[a ₁]andassign01.s [a ₂]assign01.s [a ₃]divassign01.s [a ₄]leftassign01.s [a ₅]minusequal.s [a ₆]modassign01.s [a ₇]mulassign01.s [a ₈]orassign01.s [a ₉]plusequal.s [a ₁₀]rightassign01.s [a ₁₁]xorassign01.s [a ₁₃]char01.s [a ₁₄]char01.s [a ₁₇]dowbreak01.s [a ₁₈]float_calculate07.s [a ₁₉]float00b.s [a ₂₁]float01.s [a ₂₂]float02.s [a ₂₄]funcarg01.s [a ₂₅]funcarg02.s [a ₂₆]funcorder.s [a ₂₇]funcorder02.s [a ₂₈]goto01.s [a ₃₇]if01.s [a ₃₈]if02.s [a ₃₉]int_if_and_to.s [a ₄₅]pointer_func01.s [a ₄₆]pointer_func03.s [a ₄₇]pointer_func06.s [a ₄₈]vector01.s [a ₄₉]vector02.s [a ₅₆]continue02.s [a ₅₈]eachloop02.s [a ₅₉]while_break01.s [a ₆₀]while01.s
③復元できない	-
④未検証	[a ₁₂]bitfield01.s [a ₁₅]param01.s [a ₁₆]param01.s [a ₂₃]param01.s [a ₂₉]goto02.s [a ₃₀]goto03.s [a ₃₁]if_and_or01.s [a ₃₂]if_and01.s [a ₃₃]if_and02.s [a ₃₄]if_or_and01.s [a ₃₅]if_or01.s [a ₃₆]if_or02.s

アセンブリからCIT作成の例

CITへの変換: andassign01.s(a₁)

No	andassign01.s	アセンブリコードによりわかること
1.	<pre>addi sp, sp, -48 sd s0, 40(sp) addi s0, sp, 48</pre>	<p>1) 次のコードに、a0レジスタを含むコードがないので、関数パラメータはない [卒業研究報告書, p25, 5.5 <defun>を参照]</p>
2.	<pre>li a5, 255 sw a5, -20(s0) li a5, 3 sw a5, -24(s0)</pre>	<p>1) ローカル変数、int型である。 2) swでローカル変数に代入している</p> <p>(T(=) uk_20 255) (T(=) uk_24 3)</p> <p>となる</p>
3.	<pre>addi a5, s0, -20 sd a5, -36(s0)</pre>	<p>1) addi rd, s0, immediate s0が指定されているので、アドレスとなる</p> <p>2) sdされているのでローカル変数への代入となる</p> <p>(T(=) uk_36 uk_20&) となる</p>
4.	<pre>li a5, 65536 addi a5, a5, -16 sw a5, -24(s0)</pre>	<p>1) liのimmediateが65536で、luiでの表現は4096の倍数となるので、次のaddiを加算した結果が代入される</p> <p>(T(=) uk_24 65520) となる</p>
5.	<pre>ld a5, -36(s0) lw a4, 0(a5) ld a3, -44(s0) lw a2, 0(a3)</pre>	<p>1) uk_36* であることがわかる</p> <p>2) uk_44* であることがわかる</p>

	<pre> lw a1,-28(s0) addw a1,a2,a1 sext.w a1,a1 and a1,a4,a1 sext.w a1,a1 ld a4,-36(s0) sw a1,0(a4) </pre>	<p>3)uk_28 であることがわかる</p> <p>4) (T(+) uk_44* uk_28) ->a1</p> <p>5) (T(& uk_36* a1) ->a1</p> <p>6) (T(=) uk_36* a1)</p> <p>ここで a1は、5)のa1より</p> <p>(T(& uk_36* (T(& uk_36* a1)) となり 4)のa1より</p> <p>(T(& uk_36* (T(& uk_36* (T(+) uk_44* uk_28)))</p> <p>となる</p>
--	--	---

その他の例は、[卒業研究報告書,p56-p69]を参考にしてください

RISC-VアセンブリからCIT変換可能なこと

◇確実に可能なこと

- ・ CITファイル名の復元
- ・ 関数名の復元
- ・ 関数の範囲
- ・ 関数のパラメータ数の復元
- ・ グローバル変数名の復元

◇確定できないこと

- ・ 構造体メンバーの型が全て同じ場合、コード上、配列と区別がつかないことがある
([卒業研究報告書, p21 p32 p41 p44 p61]を参考にしてください)

◇確実に不可能なこと

- ・ ローカル変数名、パラメータ名、コメントは、復元できない
- ・ 構造体メンバー名の復元はできない

まとめ

本研究は、コード生成部分に焦点をあて、新たにCITという中間言語を設計しました。それを対象にRISC-Vアセンブリを出力するコンパイラが作成可能であるかを試しました。今回の範囲においては、結果として可能でした。またRISC-Vアセンブリから中間言語(CIT)への変換がどの程度可能であるかも検証しました。逆コンパイラの検証については、限定的ではあるが検証した範囲においてCITへの変換が可能であることを確認しました。

今後の課題（Future Work）として、現段階のCITコンパイラは、いくつか実装できていないところがあります。認識できているところとして多次元配列、構造体のネストには対応できていません。これを実装できれば、CITコンパイラのプロトタイプが完成となります。

参考文献、使用ツール、ソフトウェア

参考文献

[Cifuentes94] Cristina Cifuentes. Reverse Compilation Techniques. QUEENSLAND UNIVERSITY OF TECHNOLOGY, 1994. https://yurichev.com/mirrors/DCC_decompilation_thesis.pdf

[DA18] David Patterson(著), Andrew Waterman(著), 成田 光彰(翻訳). RISC-V原典. 日経BP, 2018

[KR88] Brian W. Kernighan, Dennis M. Ritchie. The C Programming Language Second Edition, Prentice Hall Software Series, 1988.

[Lattner20] Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, Oleksandr Zinenko. MLIR: A Compiler Infrastructure for the End of Moore's Law, 2020. <https://arxiv.org/pdf/2002.11054.pdf>

[Levine01] John R. Levine (著), 榊原 一矢 (翻訳), ポジティブエッジ (翻訳). Linkers & Loaders. オーム社, 2001

[llvmOrg] Vikram Adve, Chris Lattner. The LLVM Compiler Infrastructure. University of Illinois, <https://llvm.org/>

[Nolan10] Godfrey Nolan (著), 松田 晃一 (翻訳), 小沼 千絵 (翻訳), 湯浅 龍太 (翻訳). デコンパイルング Java. オライリージャパン, 2010

[小西 88] 小西弘一, 清水 剛. CプログラムブックⅢ. アスキー出版局, 1986

[中田 05] 中田 育男. コンパイラ・インフラストラクチャ COINS (COmpiler INfraStructure) の概要. 法政大学情報科学部, 2005. <http://coins-compiler.osdn.jp/050303/COINSAbstract.pdf>

[中田 08] 中田 育男, 渡邊 坦, 佐々 政孝, 森 公一郎, 阿部 正佳. COINS コンパイラ・インフラストラクチャの開発. 日本ソフトウェア科学会, 2008. https://www.jstage.jst.go.jp/article/jsst/25/1/25_1_1_2/_pdf

[中田 99] 中田 育男. コンパイラの構成と最適化. 朝倉書店, 1999

[疋田 88] 疋田 輝雄, 石畑 清. コンパイラの理論と実現. 共立出版, 1988

使用ツール、ソフトウェア

[gccOrg] GCC

<https://gcc.gnu.org/>

[jpuoad] Ubuntu

<https://jp.ubuntu.com/download>

[uniive] UNITY

<https://unity.com/releases/editor/archive>

[visads] Visual Studio

<https://visualstudio.microsoft.com/ja/vs/older-downloads/>

[wwworg] QEMU

<https://www.qemu.org/>

[wwwows] Windows

<https://www.microsoft.com/ja-jp/windows>

> 以上です

> なにか質問がございましたらどうぞ

> ありがとうございました。