組込みソフトウェア開発支援のための命令セットシミュレータ(ISS)作成支援ツールの試作

東園　修平[1)]・片山　徹郎[2)]

A Prototype to Generate an Instruction Set Simulator(ISS) to Support Embedded Software Development

Syuhei HIGASHIZONO, Tetsuro KATAYAMA

Abstract

The embedded systems often develop hardware and software in parallel to reduce their development time. In that case, the problem in which software cannot be tested may happen because hardware does not exit when the software is tested. An ISS(Instruction set simulator) as one solution to this problem sometimes is used, but it takes much time to develop an ISS. This paper develops a prototype to generate an Instruction Set Simulator for improvement of productivity in embedded softwares. The developed prototype generates the source code of the ISS from "ISA Definition File" which is described information on ISA(Instruction Set Architecture). In order to confirm realization and usefulness of the developed prototype, it has generated an ISS of COMETII. As a result, lines of code needed to develop the ISS can be reduced 25.39%.

Keywords: Instruction set simulator(ISS), Instruction set architecture(ISA), Embedded software

1.　はじめに

組込みシステムは、産業用、家庭用と至るところに存在し、現代社会には無くてはならないものとなっている。通常、組込みシステムの開発は、開発期間を短縮するためにハードウェアの開発と並行してソフトウェアの開発を行うことが多い[1)]。

開発したソフトウェアをテストするためには、そのソフトウェアを実行するハードウェアが必要となる。しかし、並行して開発を行っているために、テスト時にハードウェアが存在しない状況が多く、ソフトウェアのテストが実施できないという問題が生じる。

この問題の解決策の 1 つとして、開発しているハードウェアのマイクロプロセッサの動作をシミュレートする命令セットシミュレータを用いることがある。命令セットシミュレータを用いることにより、実機が存在しなくてもソフトウェアのテストを実施できる。このことは、ソフトウェアテストを実施するタイミングを早めることができるので、結果として開発期間の短縮に繋がる。

しかし、命令セットシミュレータを作成するためには、作成対象のマイクロプロセッサの情報である命令セットアーキテクチャの知識が必要であるため、手間がかかる。

そこで本論文では、命令セットシミュレータ作成に伴う手間を削減するために、命令セットアーキテクチャから命令セットシミュレータの作成を支援する命令セットシミュレータ作成支援ツールの試作を行う。

以下、本論文の構成は次の通りである。

第 2 章では、命令セットシミュレータについて説明する。

第 3 章では、今回試作する命令セットシミュレータ作成支援ツールについて説明する。

第 4 章では、今回試作する命令セットシミュレータ作成支援ツールを用いて作成した COMETII の命令セットシミュレータについて検証を行う。

第 5 章では、今回試作する命令セット作成支援ツールについて、考察を行う。

2.　命令セットシミュレータ

命令セットシミュレータ(Instruction set simulator, 以下 ISS と記す)とは、マイクロプロセッサの動作をシミュレートするプログラムのことである。この章では、ISS の作成に必要な命令セットアーキテクチャやその用途について説明する。

2.1　命令セットアーキテクチャ

マイクロプロセッサ上で利用できる命令コードの一覧のことを、命令セットと呼ぶ。命令セットはマイクロプロセッサごとに異なる。命令セットにプログラマ側から見た

1)　情報システム工学専攻大学院生
2)　情報システム工学科准教授

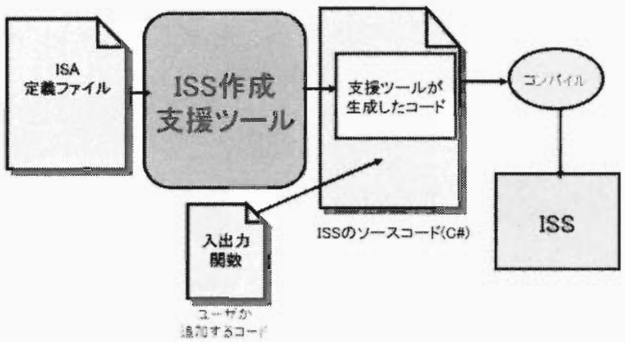

図-1 ISS 作成支援ツールの全体図.

レジスタの構成やアドレッシングモード、メモリアーキテクチャ等を加えたものを命令セットアーキテクチャ(Instruction Set architecture, 以下、ISA と記す)と呼ぶ。

提案する手法では、ISS の作成対象であるマイクロプロセッサの ISA を ISA 定義ファイルとして記述し、ISA 定義ファイルから ISS のソースコードを生成する。

2.2 用途

ISS は、主に以下の用途に用いられる。

- ハードウェアが存在しない状況下での動作検証
  通常、組込みシステムの開発では、開発期間短縮のためハードウェアとソフトウェアの開発が並行して行われる。そのためソフトウェアのテスト時にハードウェアが存在しない状況がある。そのような状況下でテストを実施するための手段の 1 つとして、ISS を用いる。
- 実機とは異なるハードウェア用のプログラムの実行
  ある特定のハードウェア用に作られたプログラムは、一般に他のハードウェア上では動かない。そのため、プログラムの変更を行う必要があるため手間がかかってしまう。この問題の解決策の 1 つとして、ISS を用いることがある。

3. ISS 作成支援ツール

図 1 に今回試作する ISS 作成支援ツールを用いた ISS 作成の流れを示す。ISS 作成支援ツールは ISA 定義ファイルを入力とし、C#のソースコードを出力とする。

以下、ISA 定義ファイル、および ISA 定義ファイルからの C#のソースコード生成手法について述べる。

3.1 ISA 定義ファイル

一般に、ISS の作成は、作成対象のマイクロプロセッサの ISA の情報を元に作成する。そこで本研究では、ISA を入力するためのテンプレートとして、ISA 定義ファイルテンプレートを提案する。

図 2 に ISA 定義ファイルテンプレートを、図 3 にテンプレートの記述例を、それぞれ示す。

以下、このテンプレートに沿って説明する。

```
1  <作成するISSのソースコードの名前>
2  WORD 1語長の長さ
3  MSIZE 主記憶の容量
4  FETCH 最初のフェッチで取ってくる語数
5  OPCODE オペコードの長さ
6  <REGISTER>
7  レジスタ名_サイズ:個数
8  <OPERAND>
9  オペランド名_サイズ:オペランドマスク 値
10 <FORMAT>
11 [命令フォーマット名]:語数 命令の構成
12 <CODE>
13 [命令の名前]命令モード指定番号
14 処理
15 オペコード:命令のフォーマット 引数の設定
16 <END>
```

図-2 ISA 定義ファイルテンプレート.

```
1  <TEST>
2  WORD 8
3  MSIZE 0x100
4  FETCH 1
5  OPCODE 4
6  <REGISTER>
7  GR_8:4
8  SF_1:1
9  <OPERAND>
10 r_2:1 GR0,GR1,GR2,GR3
11 x_2:0 0,GR1,GR2,GR3
12 adr_8:0 IMM
13 <FORMAT>
14 [op_r]:1 op r
15 [op_adr_x]:2 op -- x adr
16 [op_r_adr_x]:2 op r x adr
17 <CODE>
18 [LD]6
19 Rs -> Rd
20 SF = Rr.7
21 0x1:op_r_adr_x Rd: r Rs: ( adr + x )
22 <END>
```

図-3 ISA 定義ファイル例.

3.1.1 仕様記述部

図 2 のテンプレートにおける 1～5 行目までは仕様記述部であり、記述項目とその順序は固定である。まず、1 行目には、<>に挟まれた間にアルファベット大文字 2 文字以上で「作成する ISS の名前」を記述する。2 行目には、「WORD」の後にスペースを入れ、1 語長の長さを 32、16、8 のいずれかで記述する。3 行目には、「MSIZE」の後にスペースを入れ、主記憶の容量を 10 進数または 16 進数(16 進数の場合は 0x を付ける)で記述する。4 行目には、「FETCH」の後にスペースを入れ、最初のフェッチで取ってくる語数を記述する。5 行目には、「OPCODE」の後にスペースを入れ、オペコードの長さをビット数で指定する。

図 3 の記述例では、1 行目の「<TEST>」は、作成する ISS の名前が「TEST」であることを示している。2 行目の「WORD 8」は、1 語長の長さが 8 ビットであること示している。3 行目の「MSIZE 0x100」は、主記憶の容量が 256 語であることを示している。4 行目の「FETCH 1」は、最初のフェッチで 1 語取ってくることを示している。5 行目の「OPCODE 4」は、オペコードの長さが 4 ビットであることを示している。

3.1.2 レジスタ構成記述部

図2のテンプレートにおける6行目以降はレジスタ構成記述部であり、<REGISTER>以下の行に、レジスタの構成

を記述する。レジスタの記述は、レジスタ名はアルファベット大文字2文字以上、レジスタのサイズはビット数で、かつ1語長の長さ以下で記述し、個数は10進数で「レジスタ名_レジスタのサイズ:個数」のように記述する。レジスタのサイズを1ビットと記述した場合、そのレジスタをフラグレジスタとして扱う。

図3の記述例では、6行目の「<REGISTER>」から、レジスタ記述部であることを示し、7行目の「GR_16:8」は、「8ビットで名前がGRのレジスタを4個」、8行目の「SF_1:1」は、「1ビットで名前がSFのレジスタを1個」というレジスタの構成を示している。

3.1.3　オペランド記述部

図2のテンプレートにおける8行目以降はオペランド記述部であり、「<OPERAND>」以下の行に、オペランドを記述する。オペランド記述部は、まずオペランドの名前をアルファベット小文字または数字1文字以上で記述する。この際、先頭は必ずアルファベット小文字でなければならない。オペランドの名前の後に続けて「_」を記述後、オペランドのサイズをビット数で記述する。次に「:」を記述後、オペランドマスクを記述する。オペランドマスクには「0(読み込みのみ可能)」、「1(読み書き可能)」、「2(書き込みのみ可能)」の3種類の値のうちいずれかが記述できる。

図3の記述例では、9行目の「<OPERAND>」からオペランド記述部であることを示し、10行目の「r_2:1 GR0, GR1, GR2, GR3」は、サイズが2ビットで名前が「r」であり、rが0の時GR0、rが1の時GR1、rが2の時GR2、rが3の時GR3であるというオペランドを設定している。11行目の「x_2:0 0, GR1, GR2, GR3」は、サイズが2ビットで名前が「x」であり、xが0の時0、xが1の時GR1、xが2の時GR2、xが3の時GR3であるというオペランドを設定している。12行目の「adr_16:0 IMM」は、サイズが8ビットで名前が「adr」のイミディエイトデータであるオペランドを設定している。

3.1.4　命令フォーマット記述部

図2のテンプレートにおける10行目以降は命令フォーマット記述部であり、<FORMAT>以下の行に、命令のフォーマットを記述する。命令フォーマット記述部は、まず[]に囲まれた区間に命令フォーマットの名前を記述する。命令フォーマットの名前はアルファベット小文字、数字、または「_」を組み合わせた2文字以上で記述する。この際、先頭は必ずアルファベット小文字でなければならない。次に「:」を記述後、語数を記述する。その後、命令フォーマットの構成を記述する。命令フォーマットの構成を記述する際、先頭は必ずオペコードを意味する「op」でなければならない。オペランドとして使用しない箇所には「-」をビット数分記述する。なお、「-」が命令フォーマットの最後尾にくる場合には省略可能である。

図3の例では、13行目の「<FORMAT>」から命令フォーマット記述部であることを示している。14行目の「[op_r]:1 op r」は、命令フォーマットの名前が「op_r」で1語で構成し、仕様記述部でオペコードの長さを4ビットと定義しているため、1語目の先頭4ビットがオペコード「op」であり、次の2ビットがオペランド「r」であることを示している。15行目の「[op_adr_x]:2 op - - x adr」は、命令フォーマットの名前が「op_adr_x」で2語で構成し、1語目の先頭4ビットがオペコード「op」、次の2ビットは何も使用せず、次の2ビットがオペランド「x」、2語目の先頭から8ビットはオペランド「adr」であることを示している。16行目の「[op_r_adr_x]:2 op r x adr」は、命令フォーマットの名前が「op_adr_x」で2語で構成し、1語目の先頭4ビットがオペコード「op」、次の2ビットはオペランド「r」、次の2ビットがオペランド「x」、2語目の先頭から8ビットはオペランド「adr」であることを示している。

3.1.5　コード記述部

図2のテンプレートにおける12行目以降は、「<CODE>」以下の行に、各命令の処理とオペコードを記述する。コード記述部は、まず[]に囲まれた区間に命令の名前をアルファベット大文字2文字以上で記述し、命令モード番号を設定する。命令モード番号は3ビットのマスク値であり、最上位ビットが2番、最下位ビットが0番である。命令モード番号の各ビットの意味を、表1に示す。

表-1 命令モード番号.

ビット番号	意味
0	引数としてRdを持ち、 命令の処理に利用する。
1	引数としてRsを持ち、 命令の処理に利用する。
2	引数としてRdを持ち、 命令の処理の結果をRdに戻す。

命令モード番号の設定後、次の行から命令の処理を記述する。命令の処理に記述できる要素を、以下に示す。

- 引数：引数が存在する場合、引数を表す「Rd」、「Rs」を使用できる。「Rd」はデスティネーションオペランドを示し、「Rs」はソースオペランドを示す。符号付きで使用する場合は「RdA」、「RsA」と記述する。また、命令モードの2ビット目が立っているときはRdの変化後の値を表す「Rr」も使用できる。
- 演算子：11種類の演算子を使用できる。使用できる演算子を、表2に示す
- レジスタ：レジスタを記述する場合は、レジスタ構成記述部で記述したレジスタ名を記述できる。なお、プログラムカウンタを使用する場合は「PC」と記述する。

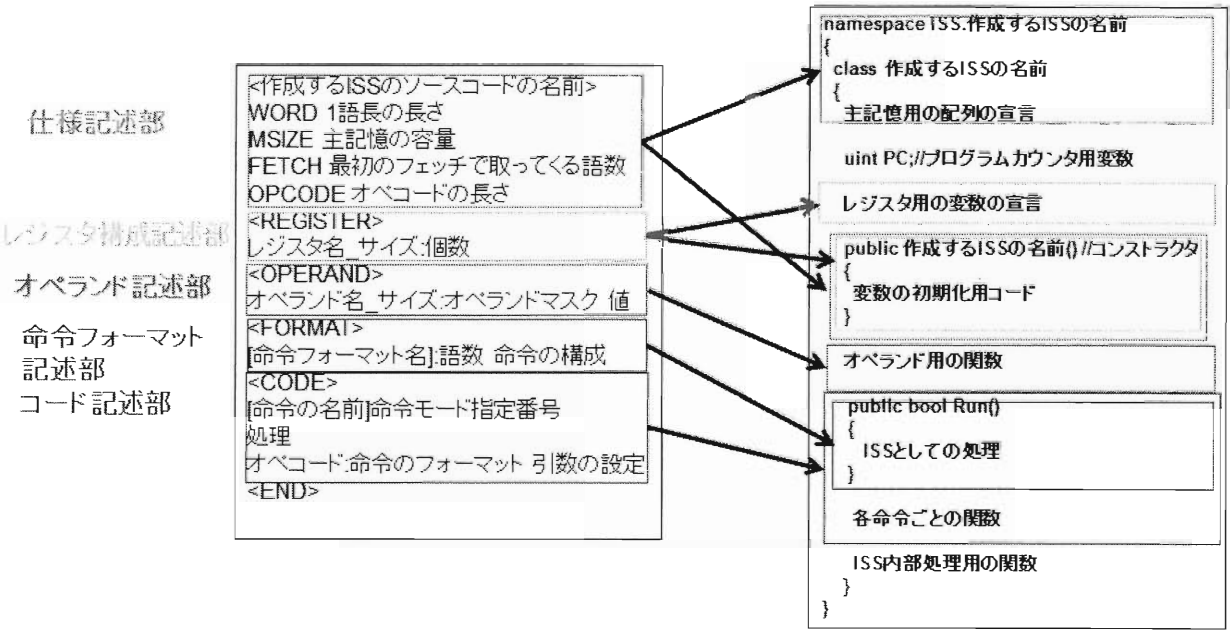

図-4 ソースコードの生成.

表-2 使用できる演算子.

演算子	説明	演算子	説明
+	足し算	-	引き算
<<	右シフト	>>	左シフト
->	代入	&	AND演算
|	OR演算	>	大なり
<=	以上	<	小なり
=>	以下		

- ビットチェック：Rd、Rs、Rrは、各要素の右に「.n(nは十進数の数字)」を付けることによって、要素の右からnビット目が立っているかを確認できる。例として、「Rd.2」ならば、Rdの2ビット目が立っているかどうかを確認できる。また「!」をRd、Rs、Rr前に付けることによって、ビットが立っているかどうかの結果を反転できる。
- フラグレジスタ設定要素：フラグレジスタに値を代入するために、「true」と「false」を使用できる。
- フラグレジスタ用演算子：フラグレジスタ用の演算子として、AND演算子「&&」、OR演算子「||」、フラグに値を設定する「=」を使用できる。

3.2 コード生成

今回試作したISS作成支援ツールは、ISA定義ファイルの情報を読み込み、各記述部の情報を、予め準備したISSのソースコードに埋め込むことによってISSを作成するISAのソースコードを生成する。図4に、ソースコード生成の概念図を示す。今回試作したツールではC#[2]のソースコードを出力する。

3.3 制限

今回試作したISS作成支援ツールには、現状いくつかの制限がある。以下にその制限を記す。

- プログラムカウンタ
  プログラムカウンタは、32ビット固定である。
- 多語長演算の未対応
  仕様記述部「WORD 1語長の長さ」で設定した1語長の長さでしか演算を取り扱えない。
- 予約語
  予約語を以下に示す。予約語を各要素の名前に付けることはできない。
  - 「PC」
  - ISSのソースコードを生成する言語の予約語
- 入出力関数
  生成するISSのソースコードは、レジスタや主記憶用の入出力関数を持たない。そのため、ユーザが追加する必要がある。

4. ISS作成支援ツールの検証

今回試作したISS作成支援ツールの動作検証のために、ISS作成支援ツールを用いて基本情報処理技術者試験のプログラミング能力試験で利用される仮想計算機であるCOMETII[3]のISSを実際に作成する。

ISS作成支援ツールの入力となる、COMETIIのISA定義ファイルを、3.1節で提案したISA定義ファイルテンプ

```
<COMETII>
WORD 16
MSIZE 0x10000
FETCH 1
OPCODE 8
<REGISTER>
GR_16:8
SP_16:1
OF_1:1
ZF_1:1
SF_1:1
<OPERAND>
r_4:1 GR0, GR1, GR2, GR3, GR4, GR5, GR6, GR7
r1_4:1 GR0, GR1, GR2, GR3, GR4, GR5, GR6, GR7
r2_4:1 GR0, GR1, GR2, GR3, GR4, GR5, GR6, GR7
x_4:0 0, GR1, GR2, GR3, GR4, GR5, GR6, GR7
adr_16:0 IMM
<FORMAT>
[op]:1 op
[op_r]:1 op r
[op_r1_r2]:1 op r1 r2
[op_adr_x]:2 op ---- x adr
[op_r_adr_x]:2 op r x adr
<CODE>
[NOP]0
0x00:op

[LD]6
Rs -> Rd
ZF = !Rr.15 && !Rr.14 && !Rr.13 && !Rr.12 && !Rr.11 && !Rr.10 && !Rr.9 && !Rr.8 && !Rr.7
     && !Rr.6 && !Rr.5 && !Rr.4 && !Rr.3 && !Rr.2 && !Rr.1 && !Rr.0
SF = Rr.15
OF = false
0x10:op_r_adr_x Rd: r Rs: ( adr + x )
0x14:op_r1_r2 Rd: r1 Rs: r2
```

図- 5 COMETII の ISA 定義ファイルの一部.

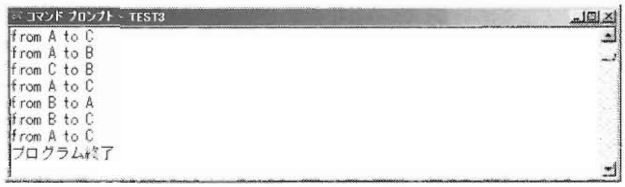

図-6 ISS 作成支援ツールを用いて作成した COMETII の の実行結果.

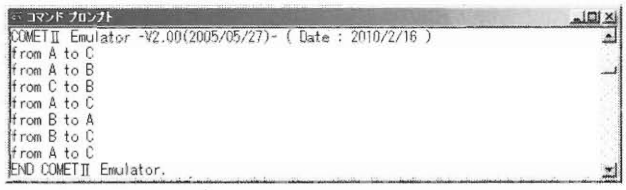

図-7 IPA の COMETII シミュレータの実行結果.

レートに基づいて記述した。記述した ISA 定義ファイルを図 5 に示す。

ISS 作成支援ツールの入力となる COMETII の ISA 定義ファイルは、193 行となった。

今回試作した ISS 作成支援ツールを用いて作成した COMETII の ISS の動作を検証するために、情報処理推進機構が配布している、COMETII のシミュレータ[4]と比較する。

ISS 作成支援ツールが生成する COMETII のソースコードに文字を出力する命令の追加と COMETII のためのアセンブラ言語である CASLII のアセンブラを作成した。比較のために、ハノイの塔を解くプログラムを使用した。

図6と図7にそれぞれの実行結果の出力画面を示す。それぞれの実行結果から、両者が一致していることが確認できる。

よって、今回試作した ISS 作成支援ツールを用いて、実際に ISS を作成できることが確認できた。

5. 考察

5.1 コード行数の削減率

今回試作した ISS 作成支援ツールが、ISS 作成にかかる手間の削減ができることを確認するために、ISS 作成支援ツールを使用せずに ISS を作成した場合との比較を行う。この比較のために、COMETII の ISS を、ISS 作成支援ツールを使用せずに C# で記述した。COMETII の ISS の作成において、今回試作した ISS 作成支援ツールを使用した場合と使用しなかった場合の記述に必要なコード行数を、表2に示す。ここで、ISS 作成支援ツールを使用した場合の行数は、「ISA 定義ファイルの行数」+「追加した動作確認のためのソースコードの行数」である。ISS 作成支援ツールを使用した場合は使用しなかった場合と比較して、COMETII の ISS を作成するために必要な行数を25.39%削減できたことが分かる。

よって、今回試作した ISS 作成支援ツールが、ISS 作成にかかる手間の削減ができたと言える。

表-3 COMETII 作成のための行数と削減率.

ISS 作成支援ツール未使用の場合の行数	382
ISS 作成支援ツール使用の場合の行数	285
削減率（%）	25.39

5.2 関連研究

命令セットアーキテクチャの記述からISSを生成するツールの1つとして、Campinas大学で開発されたArchC[5]がある。ArchCはSystemC[6]に基づくソフトウェア開発環境構築ツールであり、アーキテクチャの記述からアセンブラ、リンカ、シミュレータといったソフトウェアの開発環境を構築することができる。ArchCを利用するためには、SystemCを学ぶ必要がある。これに対して、今回試作したISS作成支援ツールは、命令セットアーキテクチャの仕様のデータをそのまま入力として使うので、SystemCを学ぶ必要がなくその分の手間を削減できる。

また、プロセッサ開発環境の一部として、ISSを生成するASIP Meister[7]がある。ASIP Meisterは特定用途向けプロセッサ開発ツールであり、プロセッサの仕様記述からプロセッサのハードウェア記述言語やSystemC記述、プログラムの開発環境であるアセンブラやデバッガ、ISSを自動生成する。ASIP Meisterはプロセッサの開発環境であるため、命令の動作の記述にパイプライン中のデータ処理を記述する言語を使用する。これに対して、今回試作したISS作成支援ツールは、ISSの作成のみに重点を置いているため、命令の動作の記述にパイプライン中のデータ処理など複雑な記述は必要としない。そのためISSを作成するだけならば、今回試作したISS作成支援ツールの方が容易にISSを作成することができる。

5.3 問題点

以下に、今回提案したISS作成手法の問題点を挙げる。

- 未対応のレジスタ

  今回試作したISS作成支援ツールは、16ビットのレジスタ1個を8ビットのレジスタ2個のように扱うようなレジスタに対応していない。レジスタを上記のように扱えるCPUが多いため、ISS作成支援ツールを改良してこれに対応し、ツールの適用範囲を拡大する必要がある。

- 多語長演算への未対応

  今回提案したISS作成支援ツールは、多語長演算に対応していない。多語長演算が行えるCPUは多いため、提案するISS作成手法でも対応する必要がある。

- 実行時間の計測が不可能

  今回試作したISS作成支援ツールを用いて作成したISSは、命令の実行時間をシミュレートしない。そのため、実機で実行した場合とISSで実行した場合でプログラムの実行時間にずれが生じる。組込みシステムでは時間の概念が重要であるため、実行時間をシミュレートする必要がある。

- ISA定義ファイル記述にかかる時間

  今回試作したISS作成支援ツールの入力として、命令セットアーキテクチャ定義ファイルを提案した。しかし、命令セットアーキテクチャ定義ファイルの記述には時間がかかる。そこで、メーカーが作成している仕様書からISSを作成できるよう入力を再考する必要がある。

6. おわりに

本論文では、命令セットシミュレータの作成にかかる手間の削減を目的とし、命令セットアーキテクチャから命令セットシミュレータのソースコードを生成する命令セットシミュレータ作成支援ツールを試作した。今回試作した、ISS作成支援ツールを使用して、実際にCOMETIIのISSを作成できることを確認した。また、ISS作成支援ツールを使用せずに作成したISSと比較しコード行数の削減率を調べた。

削減率により、今回試作したISS作成支援ツールを用いることによって、ISS作成の手間が削減することを確認した。ISS作成の手間が削減できることは、ソフトウェアテストのタイミングを早めることができ、開発期間の短縮に繋がる。

以下に、今後の課題を挙げる。

- 未対応のレジスタへの対応
- 多語長演算への対応
- 時間の概念の導入
- 入力の再考

参考文献

1) 社団法人日本システムハウス協会エンベデッド技術者育成委員会, 「エンベデッドシステム開発のための組込みソフト技術」, 電波新聞社 (2005).
2) Herbert Schildt(矢嶋 聡 監修/株式会社テック・インデックス 訳), 「独習 C# 第二版」, 翔泳社 (2007).
3) 独立行政法人 情報処理推進機構,「試験で使用する情報技術に関する用語・プログラミング言語 ver 1.0」, http://www.jitec.ipa.go.jp/1_00topic/topic_20081027_hani_yougo.pdf
4) 独立行政法人情報処理推進機構,「CASLII シミュレータ」,http://www.jitec.ipa.go.jp/1_20casl2/casl2dl_002.html
5) The Computer Systems Laboratory (LSC) of the Institute of Computing at the University of Campinas (IC-UNICAMP),「ArchC」, http://www.archc.org/
6) Open SystemC Initiative (OSCI), 「SystemC」, http://www.systemc.org/home/
7) 今井 正治, 武内 良典, 塩見 彰睦, 佐藤 淳, 北嶋 暁「特定用途向きプロセッサ開発システム ASIP Meister」, Technical report of IEICE. DSP 102(399), pp.39-44 (2002).