社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2008－SLDM－133 (30)
2008 / 1 / 17

ソフトウェア開発環境自動構築ツール ArchC の VLIW 拡張

森本 剛徳† 久村 孝寛†† 石浦 菜岐佐† 池川 将夫†† 今井 正治†††

† 関西学院大学 理工学部 〒669-1337 兵庫県三田市学園 2-1
†† NEC システム IP コア研究所 〒211-8666 神奈川県川崎市中原区下沼部 1753
††† 大阪大学 大学院 情報科学研究科 〒565-0871 大阪府吹田市山田丘 1-5
E-mail: †{t_morimoto, ishiura}@ksc.kwansei.ac.jp,
††t-kumura@cq.jp.nec.com, ††ikekawa@bq.jp.nec.com, †††imai@ist.osaka-u.ac.jp

あらまし ArchC は C++/SystemC を基にしたオープンソースのソフトウェア開発環境自動構築ツールであり，比較的簡単な記述からそのプロセッサのソフトウェア開発環境 (Binutils および命令レベルシミュレータとその GDB インタフェース) を自動構築できるが，現在のバージョンは固定長命令の RISC にしか対応していない．本稿では，ArchC を VLIW プロセッサに適用するための拡張法を提案する．VLIW プロセッサの仕様を記述できるよう ArchC のアーキテクチャ記述を拡張し，その記述から VLIW 用のソフトウェア開発環境が自動生成できるよう ArchC を拡張した．RISC プロセッサ BrownieSTD32 を 4 並列化した仮想的な VLIW プロセッサに対し，その ArchC 記述から Binutils, シミュレータを自動構築し，このプロセッサのアセンブリコードを正しくアセンブル，実行できることを確認した．
キーワード ArchC, ソフトウェア開発環境の自動構築, GNU Tool Chain, SystemC, VLIW

VLIW Extension of Software Development Environment Construction Tool ArchC

Takanori MORIMOTO†, Takahiro KUMURA††, Nagisa ISHIURA†,
Masao IKEKAWA††, and Masaharu IMAI†††

† School of Science & Technology, Kwansei Gakuin University, Sanda, 669-1337 Japan
†† System IP Core Research Laboratories, NEC Corporation
1753, Shimonumabe Nakahara-ku Kawasaki, Kanagawa 211-8666
††† School of Informaiton & Technology Graduate School, Osaka University
1-5, Yamadaoka, Suita, Osaka 565-0871
E-mail: †{t_morimoto, ishiura}@ksc.kwansei.ac.jp,
††t-kumura@cq.jp.nec.com, ††ikekawa@bq.jp.nec.com, †††imai@ist.osaka-u.ac.jp

Abstract ArchC is a C++/SystemC-based open-source software, which generates software development environments (consisting of Binutils, an instruction simulator, and its GDB interface) from relatively simple architecture description of target processors. The current version of ArchC, however, can deal with only RISC architectures with fixed instruction word length. This article proposes a method of extending ArchC to handle VLIW processors. The ArchC architecture description language as well as the tool generators is extended so that software development environments for VLIW processors specified by the language are auto-generated. For an example VLIW processor, which is a simple fourfold parallelization of a BrownieSTD32 RISC processor, the extended tool generator successfully generated the Binutils and a simulator, which correctly assembled and executed an assembly code for the processor.
Key words ArchC, automatic construction of software development environment, GNU Tool Chain, SystemC, VLIW

1. はじめに

ASIP (Application Specific Instruction-set Processor) は，命令セットやハードウェア構成を特定の用途専用に設計したプロセッサであり，性能，コスト，消費電力の制約が厳しい組込みシステムを実装するための一つの有力な選択肢である．映像や音響のディジタル信号処理のように，厳しい消費電力の制約下で高い性能が要求される応用は，従来 ASIC (Application Specific Integrated Circuits) で実装されることが多かった．しかし，近年のプロセッサの電力・性能比の向上や，バグ修正や仕様変更への対応の柔軟性を考慮すれば，このような「低電力・高性能」な応用は，今後 ASIP の利用が期待される領域と考えられる．VLIW (Very Long Instruction Word) は，静的スケジューリングによる命令レベル並列実行により，優れた電力性能比を実現できるアーキテクチャであり，このような領域の ASIP の主要な選択肢となる．

ASIP の利用において課題となるのが，ソフトウェア開発環境である．プログラムの規模や性質によっては，コンパイラを用いずアセンブリ言語による開発が行われる場合もあるが，その場合にもアセンブラ，リンカ，命令レベルシミュレータは必須である．これらのツールはプロセッサのアーキテクチャに依存し，その開発 (あるいは既存のツールのリターゲッティング) にはプロセッサとツール双方に関する詳細な知識と多大な労力が必要となる．

この問題を解決する枠組として，ASIP のアーキテクチャ記述からソフトウェア開発環境を自動構築するツールが提案されている．LISA (米 CoWare 社) (注1)，CHESS/CHECKERS (ベルギー TARGET 社) (注2) 等はプロセッサのアーキテクチャ記述から，コンパイラも含めたソフトウェア開発環境を構築することができる．しかし，プロセッサの仕様に基づいた情報を正しく記述することが難しく，環境構築に必要となる記述の量が多い．また，ライセンス制約や利用者数が多くない等の理由から，生成されたツールのチューニングや配布に制限が生じる．CGEN (米 Red Hat 社) (注3) はプロセッサのアーキテクチャ記述から Binutils および GDB のマシン記述を自動生成する．これらのソフトウェアは GPL (注4) に基づきソースコードが公開されており，ユーザーも多いため，ツールのチューニングや配布の点で有利である．しかし，一部のファイルを人手で作成する必要があるので，開発環境を全て自動で構築することはできない．

これに対し，ArchC (ブラジル Campinas 大学) [1] (注5) は比較的簡単な記述から Binutils，GDB インタフェース，シミュレータの構築に必要なファイルを生成し，開発環境を自動構築するツールである．CGEN に比べて対応できるアーキテクチャの範囲は限られるが，環境構築に必要なファイルを全て自動生成することができる．ArchC も GPL の基で公開されており，ソースコードの改変や配布の点で有利である．また既存のプロセッサの記述を流用することができる．

現在，ArchC は基本的に固定長命令のプロセッサに対応している．32 ビットのプロセッサ MIPS，SPARC，PowerPC 等の RISC への対応実績はあるが，32 ビット以外のプロセッサや可変長命令のプロセッサへの対応実績はない．また，VLIW プロセッサに対応する枠組みはまだ存在していない．

そこで，本稿では ArchC の VLIW 拡張の手法を提案する．VLIW プロセッサに対応するために，アセンブラには並列化記号の認識とパッキングビットへの値の書き込みを，シミュレータには命令の並列実行をシミュレーションする機構とレジスタ・メモリのバッファリングの処理を追加する．これらの処理に必要な VLIW プロセッサの情報を記述できるように ArchC 記述を拡張し，その ArchC 記述からソフトウェア開発環境の自動構築を行う処理系の実装を行った．

以下，2 章では GUN Tool Chain，ArchC について述べた後，3 章で ArchC によって生成されるアセンブラ，シミュレータの VLIW 拡張の方法を述べる．4 章では ArchC 記述の VLIW 拡張と ArchC の実装について述べる．

2. ArchC

2.1 GNU Tool Chain

GNU Tool Chain は GNU プロジェクト (注6) によって開発された Binutils，GDB 等からなるソフトウェア開発環境である．Binutils は gas (アセンブラ)，ld (リンカ) をはじめ，表 1 に示すツールを含むバイナリユーティリティ群である．GDB は幅広いオブジェクト形式と，多くのリモートデバッグプロトコルをサポートするデバッガである．GDB は命令セットシミュレータ (run) を含むが，これ以外のシミュレータもインタフェースを用意することにより GDB と接続できる．

GNU Tool Chain は GPL に基づいて公開されており，ソースコードの閲覧，改変を行うことができる．GNU Tool Chain は機械依存部分を独立化した構造になっており，新しいプロセッサへのリターゲッティングはこの機械依存部分を書き換えることにより行う．

表 1 Binutils に含まれる主要なツール

ツール名	用途
addr2line	プログラムアドレスをファイル名と行番号に変換
ar	アーカイブの作成，変更，抽出
as	GNU アセンブラ
c++filt	エンコードされた C++シンボルのデコード
ld	GNU リンカ
gprof	GNU プロファイラ
nlmconv	オブジェクトコードを NLM に変換
nm	オブジェクトファイルからシンボルをリスト表示
objcopy	オブジェクトファイルのフォーマット変換
objdump	オブジェクトファイルの各種情報を表示
ranlib	アーカイブの内容の索引を生成，保存
readelf	ELF フォーマットのオブジェクトファイルの情報を表示
size	オブジェクト，アーカイブファイルの各セクションの容量を表示
strings	ファイルの印字可能な文字列のリスト表示
strip	オブジェクトファイルから指定するシンボルを取り除く

(注1) : http://www.coware.com

(注2) : http://www.retarget.com/index.html

(注3) : http://sources.redhat.com/cgen

(注4) : ソースコードの公開を原則とし，ユーザーに対してソースコードを含めた再配布，改変の自由を認めている．また，再配布，改変の自由を妨げる行為を禁じている．

(注5) : http://www.archc.org

(注6) : http://www.gnu.org

2.2 ArchC

ArchC [1] は Campinas 大学で開発された C++/SystemC に基づくオープンソースのソフトウェア開発環境構築ツールである．ArchC のアーキテクチャ記述 (以下，ArchC 記述と略する) から，ターゲットの Binutils，GDB インタフェース，シミュレータの構築に必要なファイルを生成し，開発環境を自動構築することができる．ArchC の処理の流れを図 1 に示す．ArchC 記述と Binutils マシン記述のテンプレートからターゲットの Binutils マシン記述を生成し，これを用いて Binutils をビルドする．各命令の動作 (ビヘービア) は SystemC で記述され，その記述を基にシミュレータを構築する．GDB インタフェースも ArchC 記述から生成される．

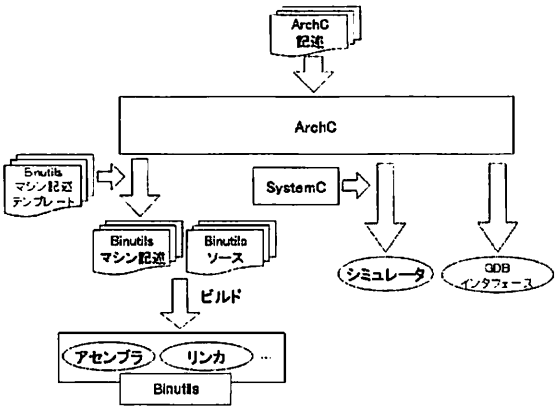


図 1　ArchC の処理の流れ

ソフトウェア開発環境の自動構築に必要な ArchC の主な記述は以下の 3 つである (*machine* はターゲットとなるアーキテクチャ名)．

(1) *machine*.ac

これはプロセッサ全体の仕様記述であり，ワードサイズやメモリーサイズ等を定義する．図 2 に MIPS サブセットの記述例を示す．2 行目はワードサイズが 32 ビットであることを定義している．3 行目は RB という個数が 32 個のレジスタファイルを，4 行目は DM というサイズが 5M バイトのメモリを，5，6 行目は hi，lo，npc という名前のレジスタを宣言している．8 行目はこのモデルで使用する命令セットの記述が mips1_isa.ac にあることを，9 行目はバイトオーダーがビッグエンディアンであることを宣言している．

```
 1:  AC_ARCH(mips1){
 2:      ac_wordsize         32;
 3:      ac_regbank          RB:32;
 4:      ac_mem              DM:5M;
 5:      ac_reg              hi, lo;
 6:      ac_reg              npc;
 7:      ARCH_CTOR(mips1) {
 8:       ac_isa("mips1_isa.ac");
 9:       set_endian("big");
10:      };
11:  };
```

図 2　mips1.ac の記述例

(2) *machine*_isa.ac

これは命令セット記述であり，各命令のフォーマットや命令コード等を定義する．図 3 に MIPS サブセットの記述例を示す．2〜4 行目は各命令タイプのフォーマット (フィールドの名称とビット数)，5〜8 行目は各命令タイプに属する命令のリスト，10〜16 行目は各命令のニーモニック，アセンブリフォーマット，および命令のコードを定義している．

```
 1: AC_ISA(mips1){
 2: ac_format Type_R="%op:6 %rs:5 %rt:5 %rd:5 %shamt:5 %func:6";
 3: ac_format Type_I="%op:6 %rs:5 %rt:5 %imm:16:s";
 4: ac_format Type_J="%op:6 %addr:26";
 5: ac_instr<Type_R> add, addu, sub, subu, slt, sltu;
 6: ac_instr<Type_I> addi, addiu, slti, sltiu, andi, ori, xori,
 7: lui;
 8: ac_instr<Type_J> j, jal;
 9: ISA_CTOR(mips1){
10:   add.set_asm("add %reg, %reg, %reg", rd, rs, rt);
11:   add.set_decoder(op=0x00, func=0x20);
12:   addiu.set_asm("addiu %reg, %reg, %exp", rt, rs, imm);
13:   addiu.set_asm("addiu %reg, %reg, \%lo(%exp)", rt, rs, imm);
14:   addiu.set_asm("addu %reg, %reg, %exp", rt, rs, imm);
15:   addiu.set_decoder(op=0x09);
16:    ...
17:  };
18: };
```

図 3　mips1_isa.ac の記述例

(3) *machine*_isa.cpp

これは命令セットの動作記述であり，各命令の動作を SystemC で記述する．図 4 に MIPS サブセットの記述例を示す．2，4 行目はデバッグ時に表示されるメッセージ，3 行目は命令の動作を定義している．

```
1: void ac_behavior( add ){
2:   dbg_printf("add r%d, r%d, r%d\n", rd, rs, rt);
3:   RB[rd] = RB[rs] + RB[rt];
4:   dbg_printf("Result = %#x\n", RB[rd]);
5:  };
```

図 4　mips1_isa.cpp の記述例

シミュレータと GDB を接続するインタフェースを生成する場合には，その他に *machine*_gdb_funcs.cpp が必要である．

3. アセンブラ，シミュレータの VLIW 拡張

3.1 VLIW プロセッサのモデル

VLIW プロセッサは同一サイクルに複数の命令を実行するプロセッサである．以下，本稿では同一サイクルに実行される複数の命令を「実行パケット」と呼ぶ．VLIW プロセッサには実行パケット内の命令の数が一定のモデルと，実行パケット内の命令の数が可変のモデルがある．以下，本稿では前者を fixed モデル，後者を packed モデルと呼ぶ．

Fixed モデルの VLIW プロセッサは，図 5(a) のように一定数の命令 (この例の場合 4 命令) が並列に実行される．並列実行可能な命令が 4 より少ない場合は図 5(b) の 4，7，8，12 行目のように NOP 命令を挿入する．アセンブリコードは，「先頭

から一定数ずつ命令が並列に実行される」というセマンティクスを与えれば，シンタックスは特に変えなくても済む．しかし，VLIW の way 数が多くなると NOP 命令の割合が多くなってコード密度が低下してしまう．

B	SUB	ADD	NOP
ADD	MPY	NOP	NOP
MPY	LDW	LDW	NOP
...	...	...	...

(a) 実行パケット

```
 1:   B   L1
 2:  SUB  A2,  1, A2
 3:  ADD  A6, A7, A7
 4:  NOP
 5:  ADD  B6, B7, B7
 6:  MPY  A5, B5, A6
 7:  NOP
 8:  NOP
 9:  MPY  A5, B5, B6
10:  LDW  *A3++, B3
11:  LDW  *A5++, B5
12:  NOP
     ...
```

(b) アセンブリコード

図 5　fixed モデル

これに対して，packed モデルの VLIW プロセッサでは，命令中に並列実行の制御を明示することにより，無駄な NOP 命令の挿入を防ぐ．図 6 のように各命令はビットを持ち，これにより並列実行を制御する．以下，本稿では，このようなビットを「パッキングビット」と呼ぶ．Packed モデルには「P タイプ」，「N タイプ」と呼ぶ 2 種類のタイプがある．P タイプは図 6(a) に示すように，ある命令のパッキングビットが 1 ならその命令を直前の命令と並列に実行し，0 なら前の命令とは並列に実行しない．Texas Instruments 社製の TMS320C62x[(注7)] 等がこのタイプに属する．N タイプでは，図 6(b) で示すように，命令のパッキングビットが 1 ならその命令を次の命令と並列に実行し，0 なら次の命令とは並列に実行しない．富士通の FRV[(注8)] 等がこのタイプに属する．

Packed モデルのアセンブリコードでは，命令の並列実行を並列化記号 || で指示する．P タイプでは図 6(c) のように，|| は行の先頭に記述し，その行の命令と前の行の命令を並列に実行することを意味する．N タイプでは，図 6(d) のように，|| は行の末尾に記述し，その命令と次の命令を並列に実行することを意味する[(注9)]．

3.2　アセンブラの拡張

Fixed モデルでは，アセンブラ自体は特に変更する必要がない．Packed モデルでは，アセンブラの構文解析部で並列化記号を認識し，パッキングビットに並列化記号がある場合の値を書き込めばよい (逆アセンブラではその逆の処理を行う)．リンカに関しては特に変更するべき点はない．

B	0	SUB	1	ADD	1
ADD	0	MPY	1		
MPY	0	LDW	1	LDW	1

(a) P タイプの実行パケット

B	1	SUB	1	ADD	0
ADD	1	MPY	0		
MPY	1	LDW	1	LDW	0

(b) N タイプの実行パケット

```
1:     B   L1
2: || SUB  A2,  1, A2
3: || ADD  A6, A7, A7
4:    ADD  B6, B7, B7
5: || MPY  A5, B5, A6
6:    MPY  A5, B5, B6
7: || LDW  *A3++, B3
8: || LDW  *A5++, B5
```

(c) P タイプのアセンブリコード

```
1:  B   L1 ||
2: SUB  A2,  1, A2 ||
3: ADD  A6, A7, A7
4: ADD  B6, B7, B7 ||
5: MPY  A5, B5, A6
6: MPY  A5, B5, B6 ||
7: LDW  *A3++, B3 ||
8: LDW  *A5++, B5
```

(d) N タイプのアセンブリコード

図 6　packed モデル

構文解析や機械語のエンコーディング処理は，Binutils の gas のマシン記述 tc-*machine*.c に定義されているので，これに上記の処理を追加すればよい．逆アセンブラの処理は opcodes のマシン記述 *machine*-dis.c に定義されているので，これに出力処理を追加すればよい．

アセンブラの拡張には，まず VLIW プロセッサが fixed モデルか packed モデルか，また packed モデルの場合には P タイプか N タイプかの情報が必要である．また，パッキングビットの位置，並列化記号がある場合のパッキングビットの値の情報も必要となる．

3.3　シミュレータの拡張

ArchC のシミュレータは単純なスカラプロセッサ向けなので，そのままでは VLIW の並列実行機構をシミュレーションすることができない．例えば，図 7 のように，並列実行される 1 行目と 2 行目の命令の A5 に WAR (Write After Read) 依存がある場合，ADD 命令の A5 の値は MOVE 命令による更新の前の値でなければならないが，ArchC のシミュレータをそのまま用いると，A5 は MOVE 命令による更新後の値になってしまう．VLIW プロセッサのシミュレーションを正しく行うためには，実行パケットを正しく形成し，実行パケット内の全命令の実行が終了した後に各命令の演算結果をレジスタ・メモリの値に書き込む必要がある．

```
1:      MOVE A5 ← #1
2:   || ADD  A6 ← A7, A5
```

図 7　WAR 依存の例

ArchC により生成されるシミュレータでは，1 命令毎に図 8(a) のような処理を行う．プログラムカウンタ ac_pc で命令フェッチを行い，その結果にデコードを行った後，プログラムカウンタの更新を行い，デコード結果に基づき実行を行う．Fixed モデルの場合には，図 8(b) のように，フェッチ，デコードした命令を順次実行パケットに挿入して行き，実行パケット内の命令数が指定の並列度に達した時点で，実行パケット内の命令をまとめて実行する．ただし，命令を実行する前に，フェッチのた

(注7)：http://www.ti.com/

(注8)：http://jp.fujitsu.com/

(注9)：本来，パッキングビットの意味 (VLIW プロセッサが P タイプか N タイプか) とアセンブリ中の並列化記号 || の位置は独立であり，図 6 の (c) のアセンブリコードから (b) の機械語を，(d) のアセンブリコードから (a) の機械語を生成することも可能であるが，本稿では実装の容易性を考慮し，本文中の 2 通りのみを考えるものとする．

めに更新した ac_pc の値を，実行パケット形成前の値に戻す必要がある．レジスタファイル，メモリ，レジスタの値はパケット内の命令を実行している間は更新せず，実行終了後に一斉に更新する (reg，regbank，mem の commit())．Packed モデルの場合には，図 8(c) のように，パッキングビットの値を判定することにより実行パケットを形成する．命令の行に並列化記号が記述されている場合のその命令のパッキングビットの値を Pb とする．パッキングビットの値が Pb に等しいとき，常に命令を実行パケットに挿入する．パッキングビットの値が Pb でないときには，P タイプと N タイプで処理が異なる．P タイプではパケット中の命令を実行した後，命令をパケットに挿入する．N タイプでは命令をパケットに挿入し，パケット中の命令を実行する．実行パケットの形成以外の処理は fixed モデルと同じである．

```
i = fetch(ac_pc);
d = decode(i);
ac_pc+=4;
exec(d);
```

(a) シミュレータの処理の流れ

```
i = fetch(ac_pc);
d = decode(i);
ac_pc+=4;
Packet = Packet ∪ {d};
if(|Packet|==指定の並列度){
  ac_pc-=4*|Packet|;
  for(e ∈ Packet){
    ac_pc+=4;
    exe(e);
  }
  Packet = φ;
  reg.commit();
  regbank.commit();
  mem.commit();
}
```

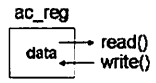
(b) fixed モデルの処理の流れ

```
i = fetch(ac_pc);
d = decode(i);
ac_pc+=4;
if(パッキングビット==Pb){
  Packet = Packet ∪ {d};
}
else{
  ac_pc-=4*|Packet|;
  if(P タイプ){
    for(e ∈ Packet){
      ac_pc+=4;
      exe(e);
    }
    Packet = φ;
    Packet = Packet ∪{d};
  }
  else if(N タイプ){
    Packet = Packet ∪{d};
    for(e ∈ Packet){
      ac_pc+=4;
      exe(e);
    }
    Packet = φ;
  }
  reg.commit();
  regbank.commit();
  mem.commit();
}
```

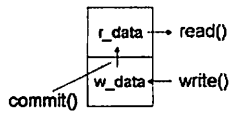
(c) packed モデルの処理の流れ

図 8 各シミュレータの処理の流れ

レジスタ，レジスタファイル，メモリの値は実行パケット内の命令の実行終了まで更新しないようにバッファリングする必要がある．ArchC ではレジスタ，レジスタファイル，メモリは C++のクラスとして定義されており，例えばレジスタは図 9(a) のようにレジスタクラスに定義されている read，write メソッドを利用して data の読み出しと更新を行っている．バッファリングを行うためには，図 9(b) のように，data を read，write 用に 2 重化し，実行パケットの実行終了後に commit メソッドで write 用 data を read 用 data に書き込むようにすればよい．メモリは全データを 2 重化すると効率が悪くなるので，書き込みのアドレスと値の組をリストとして保持し，commit メソッドで全ての書き込みの結果を反映するようにする．



(a)ArchC のレジスタ



(b) バッファリングを行うレジスタ

図 9 レジスタの拡張

ただし，プログラムカウンタ ac_pc もレジスタのインスタンスとして定義されているため，レジスタクラスを図 9 のように置換してしまうと，並列実行終了まで ac_pc が更新されなくなり，命令フェッチができなくなってしまう．そこで，プログラムカウンタだけは，従来のバッファリングを行わないレジスタとして定義する必要がある．

シミュレータの処理は *machine*.cpp に定義されているので，これを図 8(b)(c) のように変更すればよい．値のバッファリングは ArchC のヘッダファイルである ac_reg.H，ac_regbank.H，ac_memport.H を，プログラムカウンタの定義は *machine*_arch.H，*machine*_arch_ref.H を変更すればよい．

シミュレータの拡張に必要な情報はアセンブラの拡張と同じである．

4. ArchC の VLIW 拡張とその実装

アセンブラやシミュレータの VLIW 拡張に必要な情報を記述できるよう，ArchC 記述の拡張を行った．そして，拡張した ArchC 記述から VLIW 用の Binutils とシミュレータのマシン記述を自動生成できるように ArchC の拡張を行った．

4.1 ArchC 記述の VLIW 拡張

VLIW 用情報を追加した *machine*.ac の記述の例を図 10 に示す．ArchC 記述に追加した VLIW 用情報は以下の通りである．

- vliw_type
  VLIW プロセッサが fixed モデルか packed モデルかを指定する．
- vliw_ways
  最大並列度を指定する．
- vliw_pack_with
  Packed モデルの場合に P タイプ (prev) か N タイプ (next) かを指定する．
- vliw_packing_bitpos

パッキングビットの位置を指定する．*machine_isa.ac* 中の命令フォーマット定義でパッキングビットをフィールド (この例では pack_r，pack_i，pack_j) として定義し，そのフィールド名を用いてパッキングビットの位置を指定する．

・vliw_packing_bit

並列化記号がある場合のパッキングビットの値を指定する．

```
 1:  AC_ARCH(mips1) {
 2:    ac_wordsize     32;
 3:    ac_regbank      RB:32;
 4:    ac_mem          DM:5M;
 5:    ac_reg          hi, lo;
 6:    ac_reg          npc;
 7:    vliw_type       packed;
 8:    vliw_ways       4;
 9:    vliw_pack_with next;
10:    vliw_packing_bitpos pack_r, pack_i, pack_j;
11:    vliw_packing_bit 1;
12:    ARCH_CTOR(mips1) {
13:       ac_isa("mips1_isa.ac");
14:       set_endian("big");
15:    };
16: };
```

図 10　VLIW 情報を追加した *machine.ac* 記述

以上の拡張の構文解析が行えるよう，ArchC の文法を定義している arch_grammar.h，arch_grammar.y，arch_lex.l を変更した．

4.2　VLIW 対応マシン記述自動生成

VLIW 対応のソフトウェア開発環境の自動構築のために，アセンブラのマシン記述生成ツール acbingen (ArchC binary utilities generator)，シミュレータのマシン記述生成ツール acsim (ArchC simulator generator) に対して，VLIW 用情報を反映したファイルを生成するように変更した．acbingen のファイルである main.c，gas.c，gas.h，utils.c，utils.h に 3.1 章で述べた処理を追加し，acsim のファイルである acsim.c に 3.2 章で述べた処理を追加した．

4.3　実装と実行結果

ArchC の VLIW 拡張を Debian Linux のカーネル 2.4 上で実装した．利用したツールのバージョンは ArchC-2.0beta3，Binutils-2.16.1，SystemC-2.1.v1 である．

RISC プロセッサ BrownieSTD32 [2] を VLIW 化した仮想的なプロセッサに対し，拡張した ArchC を用いた環境構築の実験を行った．このプロセッサは BrownieSTD32 を単純に 4 並列化したものであり，各スロットで任意の命令を実行できる．fixed モデル，P タイプの packed モデル，N タイプの packed モデルの 3 通りについて，ArchC 記述から Binutils とシミュレータを自動構築することができた．図 11(a) のようなアセンブリコードをスカラ版の BrownieSTD32 のアセンブラ，シミュレータで実行した結果と，並列化記号を手動で記述した図 11(b) のようなコードを，自動生成した VLIW 用アセンブラ，シミュレータで実行した結果が一致する (各実行パケットの実行終了時点でのレジスタファイルとメモリの値がスカラ版と等しい) ことを確認した．

```
 1: .text
 2:   .align 2
 3:   .globl        _main
 4:   .typo _main, @function
 5: _main:
 6:   sw    -4(r6),r3
 7:   sw    -8(r6),r5
 8:   addi  r5,r6,-8
 9:   addi  r6,r6,-8
10:   addi  r7,r0,0
11:   add   r4,r0,r7
12:   lw    r3,4(r5)
13:   addi  r6,r5,8
14:   lw    r5,0(r5)
15:   jpr   r3
16: .size _main, .-_main
```

(a) スカラ版 Brownie32 のアセンブリコード

```
 1: .text
 2:   .align 2
 3:   .globl        _main
 4:   .type _main, @function
 5: _main:
 6:   sw    -4(r6),r3
 7:   sw    -8(r6),r5
 8:   addi  r5,r6,-8
 9: || addi  r6,r6,-8
10: || addi  r7,r0,0
11:   add   r4,r0,r7
12: || lw    r3,4(r5)
13:   addi  r6,r5,8
14: || lw    r5,0(r5)
15: || jpr   r3
16: .size _main, .-_main
```

(b) P タイプの Packed モデルのアセンブリコード

図 11　実験に用いたサンプルコード

5. むすび

本稿では，ArchC の VLIW 拡張手法を提案した．VLIW 用情報を記述できるように ArchC 記述を拡張し，そこからソフトウェア開発環境を自動構築する処理系を実装した．

今後は C62x，FRV，SPXK5 [3] 等の VLIW プロセッサに ArchC を対応させることを目標に，本手法の評価，改良を行っていく予定である．今回拡張を行ったアセンブラでは，実際にプロセッサが並列に実行できない命令の組み合わせがアセンブリコードに記述されても，それを検出してエラーとすることができない．このエラー処理を実現することが 1 つの重要な課題として挙げられる．

謝辞　本研究を進めるにあたり，ご討論頂いた日本電気株式会社の石原希実氏に感謝致します．御助言・御指導頂きました大阪大学の武内良典准教授はじめ大阪大学今井研究室の諸氏に感謝致します．また，支援と助言を頂いた石浦研究室の諸氏に感謝致します．

文　献

[1] R. Azevedo, S. Rigo, M. Bartholomeu, G.Araujo, C. Araujo and E. Barros: "The ArchC Architecture Description Language and Tools," International Journal of Parallel Programming, Vol. 33, No. 5, pp. 453–484 (Oct. 2005).

[2] 岩戸宏文，稗田拓路，田中浩明，佐藤淳，坂主圭史，武内良典，今井正治: "ASIP 短期開発のための高い拡張性を有するベースプロセッサの提案," 情学技報，VLD2007-92 (Nov. 2007).

[3] T. Kumura, M. Ikekawa, M. Yoshida, and I. Kuroda, "VLIW DSP for Mobile Applications," IEEE Signal Processing Magazine, Vol. 19, No. 4, pp. 10–21 (July 2002).