

1C-1

V60アーキテクチャを生かした言語処理系

木村 裕* 小野 洋彦* 堀口 信次**

*日本電気(株) マイコンコンピュータソフトウェア開発本部

**日本電気技術情報システム開発(株)

1. 緒言

32ビット・マイクロプロセッサV60用のソフトウェア開発支援ツール(Cコンパイラ、アセンブラ、リンカ、等)をチップ開発と並行して開発したので報告する。これらのツールはV60のアーキテクチャの様々な特徴を十分に生かすような機能を備えている。たとえば、Cコンパイラでは32本のレジスタの有効利用、パイプラインを考慮した命令列の生成、豊富なアドレッシング・モードの中から最適なモードの選択、等を行なっていること、またアセンブラやリンカではV60の記憶保護機構を考慮した機能を有していること、等が挙げられる。また、仮想記憶情報(アドレス変換テーブル、等)を生成するコンフィギュレータも作成した。

本稿では様々なツール群の中でも特にクロスCコンパイラに絞って、その機能、実現方法、評価について述べる。

2. V60ソフトウェア開発支援システム

図1にこのシステムの概要を示す。本システムはUNIX上で動作するクロスソフトウェア開発支援システムである。

3. V60 Cコンパイラの構成

本コンパイラの構成を図2に示す。本コンパイラの構成上の特徴はコンパイラコード生成ジェネレータCOO[1, 2]を用いてコード生成部を作成した点にある。このジェネレータを使用することにより、UNIXシステムVのCコンパイラと同等の言語機能が得られただけでなく、V30 Cコンパイラなどとソースコード全体の3/4を共通化できた。これにより、コンパイラの品質の安定、保守性の向上、開発期間短縮が実現でき、Vシリーズの新しいアーキテクチャや機能拡張に対しても、迅速な対応が可能となった。特に、本コンパイラ開発はチップ開発と並行して行なわれたが、COOを用いたことで、この間のアーキテクチャ変更や最適命令選択等のコンパイラの性能改善に柔軟に対処できた。

また、中間言語としてUNIX上のCコンパイラと同一形式のものを採用することにより、UNIX上の他の言語もサポートでき、最新UNIXのCコンパイラ仕様

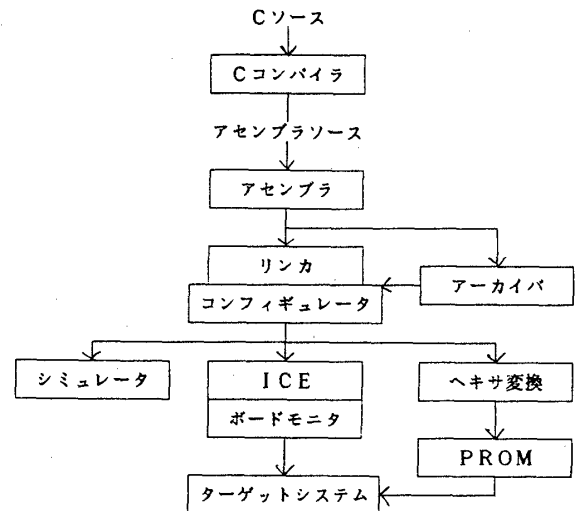
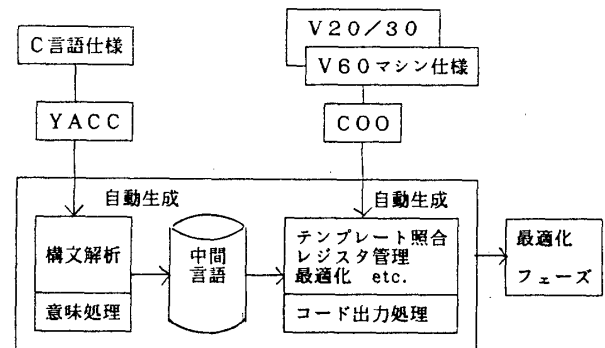


図1 V60ソフトウェア開発支援システム



*COO: COmpiler Object code Generator Generator

図2 Cコンパイラの構成

に素早く追従できる。また、新しく開発する言語[3]もこの中間言語仕様に合わせることで構文意味解析部の作成だけで、V60上で動作するコンパイラをサポートできる。

4. V60アーキテクチャとCコンパイラ

本コンパイラはV60アーキテクチャの特徴である

- (1) パイプライン制御による実行
- (2) 豊富なアドレッシング・モード
- (3) 32本の汎用レジスタ
- (4) その他、高級言語向きアーキテクチャ
(直交性のある命令、スタック・フレーム操作命令、ビット・フィールド操作命令、等)

Compiling for the V60

Yutaka Kimura*, Hirohiko Ono*, Shinji Horiguchi**
 *NEC Corporation Microcomputer Software Development Lab.
 **NEC Scientific Information System Development Co. Ltd.

を生かすオブジェクト・コードを出力する。以下では、上記(1)～(3)について説明する。

4.1 パイプライン制御

パイプラインでは、命令AでレジスタRへの書き込みがあり、次の命令Bの実効アドレス計算にそのレジスタRを使っていると、命令Aの実行終了までオペランド実効アドレス計算が待たされ、パイプの流れが止まってしまう。これをレジスタ・ハザードあるいはアドレス生成インターロックと呼ぶ。これを防ぐ方法として、命令AとBの間に適切な命令をプログラムの他の部分から移動して挿入する方法がある。本コンパイラでは最適化フェーズにおいて、基本ブロック（分岐を含まず、制御の入口／出口が各々1つの命令列）内から移動する命令を捜し挿入する。移動可能条件は大別して(1) プログラムの意味が変わらないこと、(2) レジスタ・ハザードが再び発生しないこと、である。本コンパイラは(2)の状況を回避するために、レジスタの循環割り当てを行なっている(図3)。表1はコンパイラ自身をコンパイルしたときのレジスタ・ハザードの発生件数とレジスタを順次割り当てと循環割り当てと変えてみたときのレジスタ・ハザード回避件数の違いを示したものである。この表から循環割り当ての方が、ハザード発生を減少させ、かつ、ハザード回避回数を増すことが分かる。

```
int i,j,k,l;
int a[10];      mov.w _i,r0
int b[10];      mov.w _a(r0),_k
ary(){          mov.w _j,r0
    k = a[i];    mov.w _b(r0),_l
    l = b[j];
}
```

(a)C言語プログラム

(b)順次割り当て

```
mov.w _i,r0      mov.w _i,r0
mov.w _a(r0),_k  mov.w _j,r1
mov.w _j,r1      mov.w _a(r0),_k
mov.w _b(r1),_l  mov.w _b(r0),_l
```

(c)循環割り当てとコード列再編成によるハザード回避

図3 レジスタ割り当てとハザード回避の例

表1 レジスタ・ハザード回避率

アセンブラ総ステップ数：約26.7kステップ

	順次	循環
発生件数	568	562
回避件数	131	148
回避率	23%	26%

4.2 アドレッシング・モード

V60ではポインタによる間接参照を高速化するために、2重間接アドレッシング・モードをサポートしている。このような強力なアドレッシング・モードを有効利用するために、コード生成部がテンプレートとして保持しているターゲット・マシンのアドレッシング・モードと入力プログラムの式トリーとの比較を行ない、マッチングする候補の中で最大長のものを選択する。たとえば、プログラム

```
struct A *p;
p->A.code = 10;
```

は

```
mov.w #10,4[-10[fp]]
```

と1命令だけ出力する。

また、アドレッシング・モードを利用して加減算を行なっている。たとえば、次のような最適化が可能となる。

```
mov.w 4[ap],r0
dec.w r0      → movea.w -1[4[ap]],-4[fp]
mov.w r0,-4[fp]
```

4.3 32本のレジスタ

変数の領域は通常メモリにとられるが、メモリのアクセス時間はレジスタに比べ、非常に大きいので、実行時のメモリ参照は極力避けたい。従って式の評価結果や変数の値はできる限り、レジスタに保持しておくべきである。レジスタ数の少ないマシンでは、共通式処理機能を利用することでレジスタの退避命令が出てしまうが、V60においてはレジスタの使用数を上げ、共通式の結果をできるだけ長い時間保持することができる。

一方、アプリケーションによっては、32本全部を使用するとかえって効率が下る場合もある。たとえば、頻繁にプロセス・スイッチが生じる場合、32本のレジスタを退避・回復するオーバーヘッドは大きい。そこで、本コンパイラでは、コンパイル・オプションとしてコンパイラの使用するレジスタ数を制限することができる。

5. 結言

以上、V60ソフトウェア開発支援システムのなかでも特に、コンパイラに関して、V60アーキテクチャの特徴を生かした機能について述べた。今後の課題としては、実機上でのオブジェクトの評価結果に基づき性能改善を行なう予定である。さらに、パイプラインを効率よく動作させる最適化（フラグ・ハザード、等）について検討して行く予定である。

参考文献

- [1] 三橋、他 "Code Generator Generator" 情報処理学会研究報告3-2,1985年12月
- [2] 石川、他 "汎用マイクロコンピュータでの最適化実現法" 第33回情報処理学会全国大会
- [3] 森本、他 "Adaプログラムの試作" 第33回情報処理学会全国大会