方式レベル記述言語AIDLの改良

9N−4

森本貴之　永村伊織　中村宏　朴泰祐　中澤喜三郎

筑波大学電子・情報工学系

1.はじめに

新しい命令セットアーキテクチャや処理方式を採用する計算機を効率良く短期間に設計するためには、初期設計の段階の方式レベル設計からシミュレーション等によって各種の性能評価を行なうことが必要である。

ところが、一般的には新しい計算機を設計するたびに、あるいは命令セットアーキテクチャまたは命令制御処理方式を改良するたびにシミュレータそのものを作り替えなければならない。これは方式レベルの設計を、統一してかつ簡潔に記述する言語が存在しないためである。

そこで、方式レベルにおける多種多様な設計を統一してなおかつ簡潔に記述できる方式レベル記述言語 AIDL (Architecture and Implementation Description Language) を設計した[1][2][3]。AIDLは、基本的なパイプライン制御はもとより data forwarding, delayed branch, multi-functional unit, superscalar といった命令実行制御方式を正確にかつ簡潔に記述することを目的としている。

しかし、記述対象としてプロセッサとその制御を中心に考えてきたため、メモリ系のデータ転送の記述が繁雑になる場合があった。この問題を解決するために、新たに中間信号変数と繰り返し構造の導入、フラグ型変数と動作単位であるステージに対する配列の導入を行なった。この改良により、キャッシュメモリと4バンク構成のメモリシステムを持つアーキテクチャが、モジュール的により容易にかつ簡潔に記述できることを確認した。

2.従来の方式レベル記述言語AIDLの特徴

AIDLで取り扱う時間系は離散時間で、複数の動作をまとめる単位としてステージ (stage) があり、パイプライン制御方式のステージと対応付けけられている。各ステージは開始条件を満たす毎に起動され、同時刻に同一ステージを複数起動することはできない。

AIDLにはデータパス中のレジスタに対応するレジスタ型変数と、動作間の制御を行なうフラグ型変数、そして今回導入した2種類の変数、がある。

実際のハードウェアにおけるレジスタ間のデータ転送には時間の経過が伴うため、レジスタ型変数に対する値の転送には時間の経過が必要となっている。

フラグ型変数は動作間の制御を記述するために導入された変数で、TRUE, FALSE の2値を持つ。[1][2][3]

3.追加言語仕様

今回加えられた改良は以下の3点である。

3.1 中間信号変数の導入

中間信号変数はレジスタ型変数とフラグ型変数の論理組み合せからなり、参照されるとその時刻の値を時間の経過無しに返す。値は TRUE, FALSE の2値を持つ。ただし、再帰的な定義はできない。

中間信号変数の導入により、繁雑かつ同じものが多数現れる条件判断が簡潔にまとめて記述できるようになる。

(定義) terminal 〔 変数名 ： 組み合わせ条件； 〕

3.2 繰り返し構造の導入

従来の言語仕様では、動的なキャッシュヒット判定等の記述が繁雑になるので、この問題を解決するために繰り返し構造を二つ導入する。

3.2.1 forall

forall の内部は時間的に並列に実行され、実行動作は時間の経過を必要とする。並列に実行可能な記述をまとめて記述する操作 (operation) でのみ使用できる。

(定義)

forall (カウンタ設定；境界条件；カウンタ再評価)
〔 実行文 〕；

3.2.2 forseq

forseq の内部の意味は逐次処理に基づくが、forseq 部の実行は時間経過を伴わない。値を返すのみで、時間の経過なしに実行される関数 (function) でのみ使用可能である。

(定義)

forseq (カウンタ設定；境界条件；カウンタ再評価)
〔 実行文 〕；

3.3 配列構造の導入

従来は配列構造を持たなかったフラグ型変数とステージに配列構造を導入する。

4.AIDLによる記述例

4.1 記述対象モデル

プロセッサの例として Hewlett Packard の PA-RISC1.1 の命令セットを簡略化した仮想的マシンを記述対象とし、命令実行制御方式としてパイプライン方式を採用したものをとりあげた。さらに、キャッシュメモリ (容量64KB、ブロックサイズ64B、16way-set associative、store-through 方式) とメインメモリシステム (4バンクーインタリーブ方式、PU-SCU 間のデータパスとバンクの管理及び制御を SCU (Storage Control Unit) で行なう (図1)) を持たせた。データ依存、制御依存等の命令実行阻害要因が発生した場合は、その要因が解消されるまで後続の命令はインターロックされることとした。

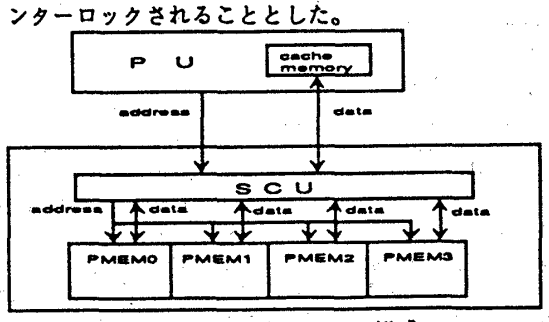

図1 メモリアーキテクチャ構成

Improvement of System Level Description Language AIDL
Takayuki MORIMOTO,Iori NAGAMURA,Hiroshi NAKAMURA,Taisuke BOKU,and Kisaburo NAKAZAWA
Institute of Information Sciences and Electronics, University of Tsukuba

4.2 記述例

以下の記述抜粋1～3は、上記のモデルを今回追加した言語仕様を用いて記述した中の一部を抜粋したものである。

・記述抜粋1　中間信号変数の定義と使用

```
terminal{
 sigE6:destE.D==IRexecute1<6:10>&&destE_valid==TRUE;
 sigM6:destM.D==IRexecute1<6:10>&&destM_valid==TRUE
                                  &&destM.op=='h12;}
                    ⋮
if (sigE6) then
if (sigE6|sigM6) then
```

・記述抜粋2　キャッシュの判定と制御（操作、関数の使用によるモジュール化）

```
% stage memory
if (search(Maddr) != -1) then cache_hit<-TRUE;
   change(search(Maddr),wnum[Maddr<C1:C2>][0:W]<0:WB>);
 else cache_hit<-FALSE; endif;
```

関数 search を呼びだして、キャッシュがヒットしたかどうかを調べる。ヒット（返値が-1でない）ならば操作 change を呼びだして LRU の値の変更を行ない、flag を立てる。変数 wnum は wnum[x][y] の2次元配列を持ち、x 次元方向の配列番号はカラムアドレス（Maddr<C1:C2>）を、y 次元方向の配列番号は LRU の値（0～W）を示す。ここで、C1, C2 は"カラムアドレスの範囲"、WB は"アドレス検索に必要なビット数 - 1"を、W は"way 数 - 1"を示すパラメータである。C1,C2,WB,W は予め宣言され、その宣言を変更することにより異なる構造をもつキャッシュの記述も容易に行なえる。今回の記述対象モデルでは、C1=20,C2=25,WB=19,W=15 である。

```
function search(Sblock<0:31>) register{Sblock<0:31>;}{
  forseq(i:=0;i<=W;i:=i+1){
    if (Sblock<0:WB>==wnum[Maddr<C1:C2>][i]<0:WB>)
      then return(i);}endif;};
   return(-1);}
```

forseq を使用してキャッシュヒットを調べる。検索するアドレス値の入ったレジスタ型変数 Sblock の 0～WB ビットと等しい値が、キャッシュのタグアレイを示す wnum の中のカラムアドレスの等しい0～W 番目までの配列中に存在するか否かを配列0 から順に検索する。存在すればその配列の番号を返し、なければ -1 を返す。

```
operation change(x,num[Maddr<C1:C2>][0:W]<0:WB>)
 register{wnum[Maddr<C1:C2>][0:W]<0:WB>;}{
   if (x != 0) then
    forall (i:=1;i<=x;i:=i+1){
     wnum[Maddr<C1:C2>][i]<-wnum[Maddr<C1:C2>][i-1];}
    wnum[Maddr<C1:C2>][0]<0:WB><-Maddr<0:WB>;endif;}
```

forall を使用してキャッシュヒットの場合の LRU 値の変更を行なう。ただしここでは、ある LRU 値に対するタグを変える。x 次元がキャッシュヒットしたカラムアドレスである wnum の配列において、y 次元方向の配列番号が1からx 番までの構成要素にそれぞれ1小さい配列番号の値を代入し、0番にはヒットしたアドレスを代入する。このとき、forall は

```
wnum[Maddr<C1:C2>][1]<-wnum[Maddr<C1:C2>][0];
wnum[Maddr<C1:C2>][2]<-wnum[Maddr<C1:C2>][1];
                    ⋮
```

の様に展開され、3.2.1で述べたように並列に実行される。従って、新しい wnum[Maddr<C1:C2>][2] には古い wnum[Maddr<C1:C2>][0] の値ははいらない。

・記述抜粋3　SCU によるアクセスバンクの判定

```
% stage scu1
switch(S1Maddr<30:31>)
 case 'h0:pmemstart[0]<-TRUE;
          ⋮
```

アドレスの下位2ビット（S1Maddr<30:31>）を見て0～3番までのどのバンクにアクセスするかを決定して、フラグ型変数 pmemstart[i] を立てる。ただし、i はアクセスするバンクの番号である。

・記述抜粋4　主記憶のバンク構成記述（ステージとフラグ型変数の配列使用）

```
stage pmem[i](pmemstart[i]){
```

pmem[i]：ステージ名
pmemstart[i]：フラグ型変数

各バンクでの実行動作（readやwrite）を記述する。ただし、ステージである pmem[i] は pmem[0:3] であることを予め宣言しておく必要がある。pmemstart[i] が立てられたバンクのみが起動される。各バンクでの制御は同じで、変数とステージの配列番号のみが異なる。

5.結果

中間信号変数の導入により、繁雑かつ同じものが多数現れる条件文の判定条件部分が簡潔になった。

従来の言語仕様では繁雑にならざるを得なかった動的なキャッシュヒットの判定や replacement 等の動作が、繰り返し構造の導入により極めて簡潔に記述できた。（従来はすべての条件についてそれぞれ実行記述が必要）

さらに、フラグ型変数とステージに配列構造を導入したことによりバンクの数だけ同様の記述が必要であったメモリ系の記述を一括にまとめることができた。（各バンクでの制御は同じで、幾つかの変数とステージの名前のみが異なっていた）

今回対象としたモデルにおいて、記述量を行数で比較すると、従来の言語仕様で911行だったものが言語仕様を追加したことで671行（26.3%減）で済むようになった。従来の3/4以下の記述量となっているが、メモリのバンク数が増えたり、命令実行制御方式が更に複雑になれば、より記述量が減少する割合が高くなると思われる。

6.終わりに

今後は、今回追加した言語仕様を汎用方式シミュレータに実装し、より複雑なアーキテクチャと命令実行制御方式の AIDL による記述と評価（superscalar 等）を行なう予定である。

参考文献

[1]中村宏他、方式レベル記述言語AIDLによる命令実行制御方式,DAシンポジウム'91.

[2]伊藤元久他、方式レベル記述言語AIDLの概要,情報処理学会第43回全国大会,4R-1,1991.

[3]H.Nakamura et.al.,"Architecture and Implementation Description Language for Advanced Processor Design" IEEE APCCAS'92.