# 6E−6 汎用方式レベル記述言語AIDLのシミュレータの実装

伊藤元久　位守弘充　中村宏　中澤喜三郎

筑波大学電子・情報工学系

## 1. はじめに

高性能な計算機を短期間に設計するためには、初期設計の段階である方式レベル設計からシミュレーションをして設計評価をおこなうことが必要である。そこで、我々は、方式レベルにおける多種多様な設計を統一して、しかも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]。その後、この汎用方式レベル記述言語AIDLを解釈・実行するシミュレータの実装をおこなった。本稿では、AIDLのシミュレータの実装手法、特に並列動作する制御論理間の時間関係の無矛盾性を保障する方式について詳細に報告する。

## 2. AIDLの特徴

まず、AIDLの特徴を、時間と動作の関係を中心に簡単に述べる[1][2]。

(1) AIDLは離散時間軸上で定義されている。離散時刻の最小間隔は記述対象計算機のマシンサイクルに相当すると考えてよい。この離散時刻の最小間隔をAIDLでは単位時刻と呼ぶ。

(2) 並列に実行される動作と逐次的に実行される動作をAIDLでは明確に区別して記述する。並列に実行されるべく記述された動作は、同時に実行され、その動作の結果は記述順によらない。以下に例を示す。変数A,B,Cの初期値をそれぞれ、0,1,2とすると(a),(b)ともに変数Aの値は1になる。

```
(a)          (b)
A <- B;      B <- C;
B <- C;      A <- B;
```

(3) AIDLでは、変数への値転送には時間の経過が伴う。変数の値は一連の動作の終了時に確定する。

(4) (2)動作の並列性と(3)値の代入に時間の経過が伴うことからAIDLでは、変数はある時刻においてただひとつの値をとる。

## 3. 時間関係の無矛盾性保障の必要性

2. で述べたように、並列に動作するべく明確に記述された動作は並列に実行され、動作結果は記述順によるべきではない。しかし、シミュレータ内部では逐次的に処理される。そこで、逐次処理のシミュレーションと本来の言語上の意味を一致させる必要がある。そのため、シミュレータに時間関係の無矛盾性を保障する機構を設ける必要性が生じる。この機構は、実行結果を一時領域に保存しておき、一連の動作終了時に、各変数に値を転送することで実現している。シミュレータは単位時刻ごとに、動作の終了を評価し、終了時刻ならば変数への代入をおこなう機構を呼びだし処理を委託する。

AIDLの言語仕様では、同一時刻に、同じ変数に複数の値が代入されることは、原則として許されていない。変数への代入をおこなう機構部では、多重代入の有無を検出し、このような多重代入が起こった場合は、シミュレーション実行時エラーとして処理する。

AIDLにおいて同時刻の多重代入が認められる唯一の例外は、flag型変数への代入である。flag型変数はAIDLの変数型のひとつであり、TRUEあるいはFALSEの2値のみを変数値としてとり、動作記述内で主に制御の流れを記述するのに用いる。同時刻に相異なる値が代入されたならば、flag型変数は、優先値をとる。この優先値は、変数の宣言時に宣言されていなければならない。

## 4. 時間関係の無矛盾性保障の機構

無矛盾性保障の機構を、AIDLの代表的な変数型である、flag型変数とregister型変数を例に述べる。両変数型共にtime-stampを用いて最終更新時刻を保持している。そして、このtime-stampを用いて多重代入の有無を判断する。多重代入が生じたときの対応はflag型変数とregister型変数では異なる。

### 4.1 flag型変数の機構

flag型変数に多重代入がおこなわれても、register型変数と違い、シミュレーション実行時エラーにならず、優先値が代入される。

### 4.2 register型変数の機構

register型変数は、ビット指定が可能で1ビットごとの演算も可能である。ビット演算をおこなう場合、同一の変数内の異なるビットを同時刻に変更しても、矛盾は生じないため多重代入とはならない。多重代入となるのは、同一変数の、同一ビットを、同時刻に変更する場合である。

各ビットの最終更新時刻をもれなく保持するためには、time-stampを各ビットごとに設ける必要がある。しかし、この実装方法は無駄が多い。無矛盾性保障の機構の実現のためには、変数全体としての最終更新時刻と各ビットのごとの更新情報の有無の

---

Implementation of AIDL Simulator

Motohisa ITO,Hiromitsu IMORI,Hiroshi NAKAMURA,Kisaburo NAKAZAWA
Institute of Information Sciences and Electronics,University of Tsukuba

みで十分である。そこで、time-stampは各変数につきひとつだけとし、ビットごとの更新の有無はビット表で管理する方法を採用している。
time-stampは、その変数全体としての最終更新時刻を保持し、ビット表にはtime-stampの時刻に更新があったビット位置が記憶されている。

register型変数ではシミュレーション動作中に多重代入が生じると、エラーメッセージを出力しシミュレーションを停止する。

### 4.3. 無矛盾性保障機構の動作例

**4.1,4.2**で述べた機構の動作を例を上げて説明する。図1に示す**AIDL**による動作記述を例とする。

#### 4.3.1 動作記述の説明

図1に示す動作記述のうち、stage fetch {...}で表される部分（fetchステージ）とstage decode {...}で表される部分（decodeステージ）は並行して同時に実行される。

各ステージ内の動作のうち、設定部分内のdecodeS <- TRUE;（動作a）と本体部分内のIR<0:7> <- Mem[IMAR];（動作b）は逐次性を表す記述子'//'で区切られているので、逐次的に実行される。同じく、decodeS <- FALSE;（動作c）とIR<0:7> <- Mem[IMAR+1];（動作d）も逐次的に実行される。（図2参照）

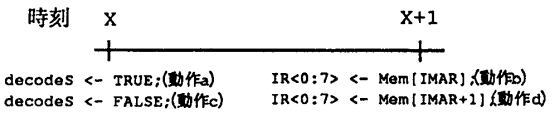


図2 動作と時刻の関係

#### 4.3.2 シミュレータの動作

シミュレータ内部ではfetchステージ、decodeステージの順に実行されるとする。decodeSはflag型変数で優先値はFALSE、IRはregister型変数として宣言してあるとする。

1. まずfetchステージの設定部分（動作a）、次にdecodeステージの設定部分（動作c）が実行され、実行結果が一時領域に保管される。
2. fetchステージ設定部分の実行結果がdecodeSに代入される。decodeSのtime-stampは現在の時刻（時刻X）にセットされる。
3. decodeステージ設定部分の実行結果をdecodeSに代入しようとするが、decodeSのtime-stampが現時刻（時刻X）と等しく、値も異なるため、多重代入であることが判明する。無矛盾性保障機構が作動して、decodeSの値を優先値であるFALSEにする。
4. 次に各ステージの本体部分（動作b,d）が実行され、実行結果が一時領域に保管される。
5. fetchステージ本体部分の実行結果がIR<0:7>に代入される。IRのtime-stampは時刻X+1にセットされ、ビット表はbit0~bit7が真になる。
6. decodeステージ本体部分の実行結果を同じくIR<0:7>へ代入しようとするが、IRのtime-stampが現時刻（時刻X+1）と等しいためビット表と代入ビットの整合性を評価し、多重代入であることが判明する。無矛盾性保障機構が作動しシミュレーション実行時エラーを起こしてシミュレーションを停止する。

```
stage fetch {              fetchステージ
        :
   decodeS <- TRUE;                動作a
        : //
   block(1,TRUE,1) {
        :
      IR<0:7> <- Mem[IMAR];        動作b
        :
   }
}
stage decode {             decodeステージ
        :
   decodeS <- FALSE;               動作c
        : //
   block(1,TRUE,1) {
        :
      IR<0:7> <- Mem[IMAR+1];      動作d
        :
   }
}
```

図1 動作記述例

### 5. おわりに

現在、評価計算機のアーキテクチャは**DLX**[3]とIBM370系アーキテクチャを使用している。そして、評価計算機の動作記述として、

(1) simple pipeline (2) data forwarding
(3) delayed branch (4) (2)+(3)の組み合わせ
(4) multi-functional unit
(5) Tomasuloのアルゴリズム

をとりあげ、シミュレータの完成度を上げると共に、各動作記述の評価をおこなっている。

今後は、評価計算機のアーキテクチャと評価用プログラムの種類を増やすと共に、動作記述として、super-scalar等のより複雑な動作記述をシミュレーションして評価をえたい。

**参考文献**

[1] 中村ほか、方式レベル記述言語**AIDL**による命令実行制御方式記述,DAシンポジウム'91.
[2] 伊藤ほか、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] J.K.Hennessy and D.A.Patterson. Computer Architecture A Quantitative Approach,Morgan Kaufmann Publishers,1900.