計算機アーキテクチャ　127－10
設　計　自　動　化　86－10
（１９９７．　１２．　１１）

連接する変数の生存区間に着目したレジスタ割付けの一方式

田中 旭　佐山 旬子　小谷 謙介　湯川 博司　丸山 幸孝
松下電器産業

近年、機器組み込みソフトウェアを高級言語により開発することが不可欠になっているが、高級言語開発におけるオブジェクトコードサイズの増大が組み込み分野では大きな問題となる。 そこで我々は、コンパイラ技術の観点から組み込みマイコンMN10xシリーズを開発した。 本稿では、MN10xシリーズのような少数のレジスタしか持たないマイコンに対しても、効率の良い大域的レジスタ割付けが可能な方式について報告する。この方式は、変数間の生存区間の継りに着目して、変数に各レジスタを割り付けたときに、どれくらいの転送命令の削減が見込めるかを定量化した割付有効度を求めることにより、冗長な転送命令の発生を抑えることを特長とする。

A Method of Register Allocation Attending to the Series of Variable's Live Range

Akira TANAKA, Junko SAYAMA, Kensuke ODANI, Hiroshi YUKAWA, Yukitaka MARUYAMA
Matsushita Electric Industrial Co.,LTD

Recently, the high-level language is indispensable at the development of the software which is embedded to electrical products. But the increasing code size of program developed by the high-level language is serious in the developing of the embedded program. So, we developed the architecture of micro computer named "MN10x Series" which is designed in consideration of the compiler art. We describe one of the global register assignment method which is able to apply the embedded micro computer which have few register such as "MN10x Series". This method effectively assigns of register to variable by estimating the reducible ratio of the number of transfer instruction, attending to the series of variable's live range.

1．はじめに

近年、マイコン制御の機器に組込まれるソフトウェアのサイズが大規模化しており、高級言語による開発が要望されている。しかし、高級言語による開発では、オブジェクトコードサイズの増大により、コストアップになることが組込み分野では大きな問題となる。

そこで我々は、マイコンの設計方針として、組込みマイコンに要請されるオブジェクトコードの高効率性と、大規模な組込みソフトウェアを十分に処理可能な高速実行性を両立するために、コンパイラを含めたマイコン設計を行っている。

本稿では、この設計方針に基づいた組込みマイコンMN10xシリーズ用のコンパイラにおけるレジスタ割付け方式について報告する。

2．コンパイラの[redacted]

C言語プログラムのオブジェクトコードサイズを圧縮するために、MN10xシリーズではレジスタをデータ用とアドレス用に機能分離して各４本構成とし、機械語命令のレジスタ指定フィールドを２ビット化することにより、コンパイラが生成する頻度の高い基本命令を１バイト化した。（図１）

図１　基本命令のビット割付け

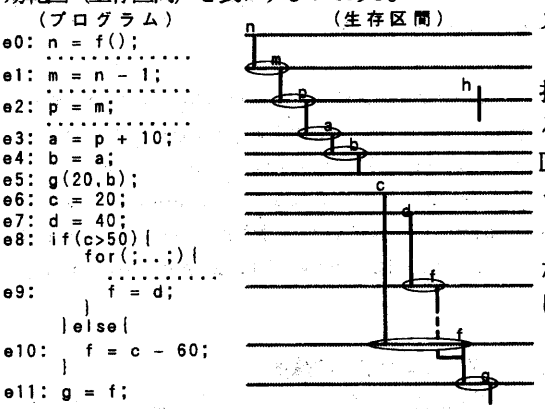
しかし、比較的レジスタ本数が少ないために、コンパイラがレジスタに効率よく変数を割付けないと、かえってレジスタ間やレジスタとメモリ間の転送命令が増加してしまい、基本命令１バイト化の効果が相殺されてしまう。

グラフカラーリング方式[1)~4)]として知られている従来のレジスタ割付け方式は、比較的本数の多い汎用レジスタに対しては有効な割付け方式であるが、本数が少なくデータ用、アドレス用などのように機能差があるレジスタや、関数の引数や戻り値用のように使用用途が指定されているレジスタも含めたレジスタ割付け方式としては不十分である。　そこで我々は、無駄な転送命令の削減度合をヒューリスティクスに定量化した割付有効度を算出して、割付有効度が最大のレジスタを変数に割付ける新規レジスタ割付け方式を考案した。[5)]

本稿では，文献5)で報告した方式を改善した方式と、従来のグラフカラーリング方式との比較結果について述べる。

3．本方式の基本アイディア

図２はプログラムと変数が保持している値の有効範囲（生存区間）を表わすものである。



：終始一致関係

n,h,c が D0、b,d が D1 に割付られている

図２　プログラムと生存区間の例

変数aとbのように生存区間の終わりと始まりが一致するものを「終始一致関係」にあると定義する。ここで変数aとbを同じレジスタに割付けると代入式e4に対応する転送命令は不要となる。またコンパイラが生成する機械命令が２オペランド形式であれば、式e3のような演算式においても、変数pとa に同じレジスタを割付ければ、不要な転送命令は生成されない。

さらに、図２の変数n,m,p,a,b のよう に、終始一致関係により生存区間が連続しているもの同士は、可能な限り同じレジスタに割付ければ転送命令を削減できる。そこで、ある変数Xから終始一致関係により生存区間が連続している変数を、変数Xと「区間連鎖関係」にあると定義する。図２では変数aと区間連鎖関係にある変数はn,m,p,bであり、変数fと区間連鎖関係にある変数はc,dである。

以上を踏まえて、本方式では以下の２点に着目した。

(1) 区間連鎖関係にある変数のうち、位置的に近い変数と同じレジスタに割付ける方が有利。

例えば、図２の変数aに割付けるべきレジスタ選択において、変数n,bに予めレジスタD0,D1が割付けられているとする。このとき、aが存在する位置からnが存在する位置までは、bに比べて距離があるため、変数m,pもnと同じD0に割付けられる可能性が低い。よって、aはbと同じD1に割付ける方が確実に転送命令を削減できる。

(2) 区間連鎖関係にある変数と生存区間が重なる変数に割付けられているレジスタと、異なるレジスタに割付ける方が有利。

例えば、図２の変数aに割付けるべきレジスタ選択において、変数hがレジスタD0に割付けられているならば、変数pにD0は割付け不可能であり、aにD0を割付けると式e3で転送命令が生成されてしまう。

以上の着目点から、変数Vにレジスタrを割付けたときの割付有効度BenefitRate(V,r)を次のように定式化する。

$$BenefitRate(V,r) = \sum_{u} K(V,u,r) / D(V,u)$$

u: rと区間連鎖関係にある変数

K(V,u,r): if Vからuの間に存在するVと区間連鎖関係にある変数tにレジスタが割付けられているとき 0
else if Vからuの間に存在するVと区間連鎖関係にある変数tと生存区間が重なる変数にrが割付けられているとき 0
else if uがrに割付けられているとき 1
else if uがレジスタに割付けられていなくて、かつuがrに割付けられている変数と生存区間が重なるとき −1
else 0

D(V,u): 変数Vから変数uに至るまでの生存区間の長さの和

この割付有効度を簡単に説明すると、変数Vと区間連鎖関係にある変数uを考えるとき、変数uがレジスタrに割付けられていること、または、変数uと生存区間が重なる変数にレジスタrが割付けられていることが、変数Vに割付けるべきレ

[割付有効度を使用したレジスタ割付けアルゴリズム]
入力: レジスタ割り付けする変数の集合
出力: レジスタを割り付けた変数の集合
処理:
s1. 入力の変数Vの全てに対して、次式により割付優先度を計算する
優先度(V) = Σ L(i)/V の生存区間長
i ∈ USE
USE : 変数Vを定義または使用している命令の集合
L(i) :命令iの存在するループのネストの大きさ
s2. for(割付優先度の高い順に変数Vを選択して以下の処理を繰り返す。){
s3. 変数Vに割付け可能なレジスタの集合を求める。
これは変数Vと生存区間が重なっている変数に割付けられているレジスタ以外のレジスタである。
s4. if(割付けるべきレジスタがないとき){ステップs2に進み、次に割付優先度の高い変数を選択する。}
s5. if(割付け可能なレジスタが1本しかないとき){そのレジスタを変数Vに割付けステップs2に進み、次に割付優先度の高い変数を選択する。}
s6. 変数Vについて図4の割付有効度計算(V)を行なう。
s7. if(変数Vに割付け可能なレジスタで、かつ割付有効度計算の結果、有効度が最大のレジスタが複数存在するとき){
s8. 各レジスタ毎に負の割付有効度を格納する処理変数WORKをゼロに初期化する。
s9. for(変数Vと生存区間が重なっていてかつ、まだレジスタが割付けられていない変数V1について繰り返す。){
s10. 変数V1について図4の割付有効度計算(V1)を行なう。
s11. 算出された各レジスタの割付有効度を有効度の最大数で割ることにより正規化する。
s12. 正規化された各レジスタの割付有効度に、変数V1の割付優先度を掛けたものを新たな割付有効度として処理変数WORKに足し込む。
}
s13. s6で求めた割付有効度が最大のレジスタでかつ、処理変数WORKに格納されている負の割付有効度の最小なレジスタを変数Vに割付ける。
} else {
s14. s6で求めた割付有効度が最大のレジスタを変数Vに割付ける。
}
}

図3 レジスタ割付けアルゴリズム

[割付有効度計算アルゴリズム]
入力: レジスタ割り付けする変数
出力: 各レジスタの割付有効度
処理:
s1. 各レジスタの割付有効度を保持する処理変数BRETをゼロに初期化する。さらに追跡対象の集合を保持する処理変数WORK1, 変数の集合を保持するWORK2,WORK3を空に設定する。
s2. 変数Xに対して追跡対象T1を生成、T1の各メンバ値を以下に設定し、割付有効度計算を行なう追跡対象の集合を保持する処理変数WORK1にT1を格納する。
T1.VAR = 変数X; T1.LEN = 1;
T1.RGS = 変数Xに割付け可能なレジスタの集合;
s3. for(処理変数WORK1が空になるまで繰り返す。){
s4. 処理変数WORK1から追跡対象T2を取り出し、処理済みの変数の集合を保持する処理変数WORK2に変数T2.VARを格納する。
s5. if(変数T2.VARにレジスタRが割付けられているとき){
s6. if(レジスタRがレジスタ集合T2.RGSに属しているとき){
s7. 処理変数BRETのレジスタRの割付有効度に 1/T2.LEN を加える。
}
} else {
s8. 変数T2.VARと生存区間が重なりかつレジスタが割付け済みの変数の集合OS1を求める。
s9. OS1に属している変数に割付けているレジスタの集合RS1を求める。
s10. OS1のうちWORK3に格納されていない変数の集合OS2を求め、OS2をWORK3に格納する。
s11. OS2に属している変数に割付けているレジスタの集合RS2を求める。
s12. RS2に属する全てのレジスタR1について、処理変数BRETのレジスタR1の割付有効度から 1/T2.LEN を減ずる。
s13. 変数T2.VARと終始一致関係にあり、かつ、WORK2に格納されていない変数Yの全てに対して追跡対象T3を生成し、T3の各メンバ値を以下に設定し処理変数WORK1に加える。
T3.VAR = 変数Y;
T3.LEN = T2.LEN + 変数T2.VARの生存区間長
T3.RGS = T2.RGS − RS1
}
}
s14. 処理変数BRET内容を結果として返す。

図4 割付有効度算出アルゴリズム

ジスタの選択にどれだけ影響を及ぼすかということを割付有効度は示している。

4. 新規レジスタ割付けアルゴリズム

本節では前節で述べた割付有効度の具体的な計算アルゴリムを図4に、割付有効度を使用したレジスタ割付けアルゴリズムを図3に示す。 図3、4において、レジスタ割付けの対象となる変数と、アルゴリズムの処理の上で導入される変数を区別するために、後者を特に処理変数ということにする。 また、変数名は同じであるが、生存区間が異なる変数は異なるレジスタに割付け可能であるので、別々の変数とする。

図3のレジスタ割付けアルゴリズムの特徴的な点は、ステップs6において算出した割付有効度が最大のレジスタが複数本存在する場合、つまり、割付

けるべきレジスタの候補が複数本存在するときは、その候補のうち生存区間が重なる他の変数に割付けた方が有利である場合の計算をステップs8からs13で行っている点である。特にステップs11,s12では、生存区間が重なる変数の割付優先度に比例して有効度を決定しているので、割付優先度の低くレジスタに割付けられる可能性の少ない変数の影響を小さく見積もるようにしている。

図4では、追跡対象というC言語でいう構造体型のデータを導入している。追跡対象は、変数、生存区間長の和、レジスタ集合 をメンバとし、それぞれ、VAR, LEN, RGSと簡単に表現している。図4のアルゴリズムは、前節で述べた割付有効度を素直にアルゴリズム化しているのではなく、割付有効度を計算する変数を起点に区間連鎖関係にある変数を次々と辿って、処理終了時、全てのレジスタについての割付有効度が求まるようにしている。

図4において、処理変数WORK2は区間連鎖関係が循環して同じ変数に対して処理が繰り返さないようにするために設けている。また処理変数WORK3はステップs12において割付有効度を減ずるレジスタを決めるとき、何度も同じレジスタについて行わないために設けている。これは複数の区間連続関係にある変数と生存区間が重なり、かつレジスタに割付けられている変数が存在するとき、そのレジスタの割付有効度が極端に減少していまうことを防ぐためである。

また、追跡対象のメンバであるRGSは、区間連鎖関係を辿る際、意味のないレジスタに対する割付有効度の計算は行わないようにするために設けている。たとえば、区間連鎖関係にある、ある特定の変数Xと生存区間が重なる変数に、レジスタRが割付けられていたなら、変数Xからさらに辿る区間連鎖関係にある変数にたとえレジスタRが割付けられていても、割付有効度にはなんら影響を与えないからである。

5. アルゴリズム比較

5. 1 評価条件

図5から図8は従来方式（グラフカラーリング方式）と本方式を使用したレジスタ割付けを行った場合について、コンパイラが生成する命令のステップ数とレジスタ充足率を比較したものである。

評価対象のマシンは、命令形式を2アドレス形式とし、レジスタ構成は32bit汎用レジスタ6本から32本構成の場合と、32bitアドレスレジスタ(AR),データレジスタ(DR)各4本計8本構成の場合とし、コンパイラでは次に示すようなレジスタ使用方針とした。（尚以下で割付け用レジスタとは、グラフカラーリング方式で割付けを行うレジスタの数を表わす。またアドレス／データレジスタ構成では、メモリ間接参照演算はアドレスレジスタのみとし、乗除算や論理演算はデータレジスタのみに許されているとしている。）

・汎用レジスタ6本構成の場合
　引数用レジスタ．．　2本
　戻値用レジスタ．．　2本（引数用レジスタと共用）
　割付け用レジスタ．．4本
・汎用レジスタ8本構成の場合
　引数用レジスタ　．．従来方式2本、本方式4本
　戻値用レジスタ　．．2本（引数用レジスタと共用）
　割付け用レジスタ．．6本
・汎用レジスタ12本から32本構成の場合
　引数用レジスタ　．．4本
　戻値用レジスタ　．．2本（引数用レジスタと共用）
　割付け用レジスタ．．残り全てのレジスタ
・アドレス／データレジスタ　8本構成の場合
　引数用レジスタ　．．従来方式 AR/DR各1本計2本、
　　本方式 AR/DR各2本計4本
　戻値用レジスタ　．．2本（引数用レジスタと共用）
　割付け用レジスタ．．6本

また、今回用いたグラフカラーリング方式では文献1)で述べられいるようなノード結合と、ある複数の変数で構成されるノードが一つでもレジスタに割付けられなかったときに、ノードを構成する変数のうち最も生存区間の重なりが多い変数を取り出して、一つのノードを構成し、グラフを再構築するすようなノード分割を行っている。これは、従来方法でも可能な限り区間連鎖関係にある変数を同じレジスタに割付けて無駄な転送命令を出さないようにすることを考慮したためである。さらに、グラフカラーリング方式では引数レジスタなどの特定目的で使用するレジスタはグラフカラーリングアルゴリズムの対象外である。しかし、グラフカラーリングの結果、レジスタに割付けられなかった変数でも、その生存区間の重なりから特

	GR6	GR8	GR12	GR16	GR20	GR24	GR28	GR32	AD8
bubble	93.75	97.44	96.15	96.15	96.15	96.15	96.15	96.15	83.87
fft	91.68	91.94	92.83	91.22	92.48	92.73	93.84	96.00	85.88
intmm	90.91	97.83	100.00	101.11	101.11	101.11	101.11	101.11	91.43
mm	81.25	89.91	95.19	95.19	95.19	95.19	95.19	95.19	80.31
perm	98.33	100.00	100.00	100.00	100.00	100.00	100.00	100.00	85.51
puzzle	99.83	99.82	99.82	99.82	99.82	99.82	99.82	99.82	89.06
queens	93.68	96.79	95.95	97.20	98.57	99.28	99.28	99.28	92.31
quick	94.37	92.42	96.72	100.00	100.00	100.00	100.00	100.00	83.44
towers	95.48	99.33	98.65	98.65	98.65	98.65	98.65	98.65	92.49
tree	92.86	94.55	94.55	94.55	94.55	94.55	94.55	94.55	81.54
AVR	93.21	96.00	96.99	97.39	97.65	97.75	97.86	98.07	86.58

＊グラフカラーリングの結果を100とする。

図5　stanfordベンチマークのステップ数比較

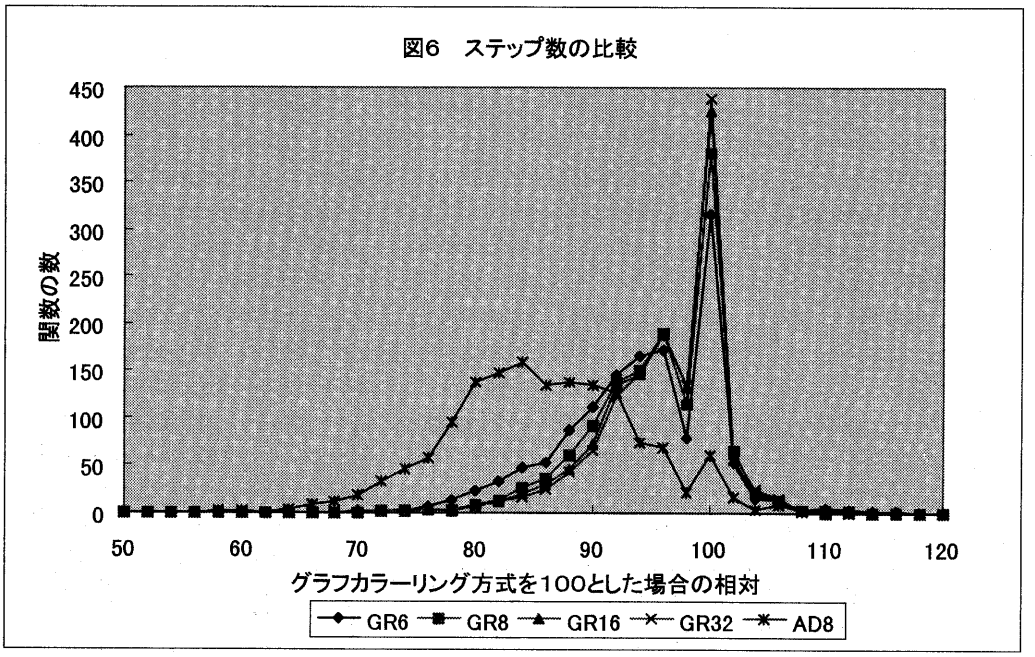


図7　1関数当たりのレジスタ充足率(ｸﾞﾗﾌｶﾗｰﾘﾝｸﾞ)

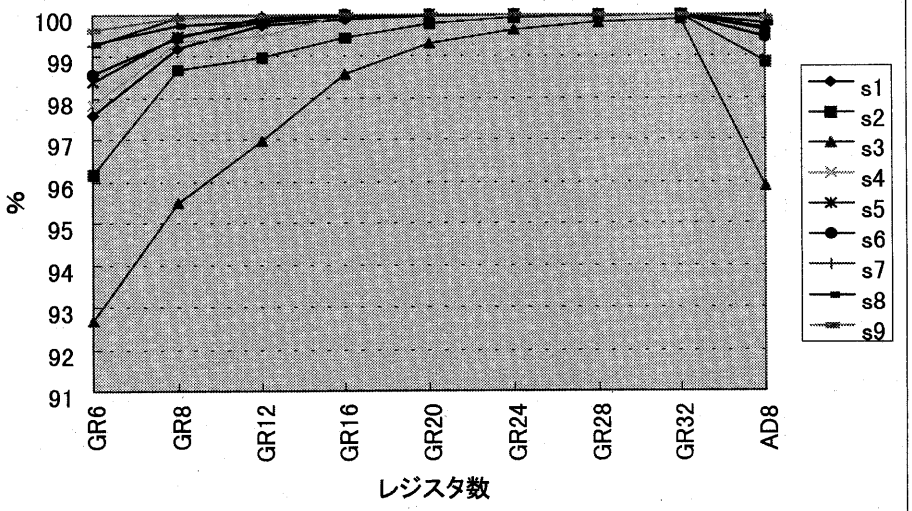
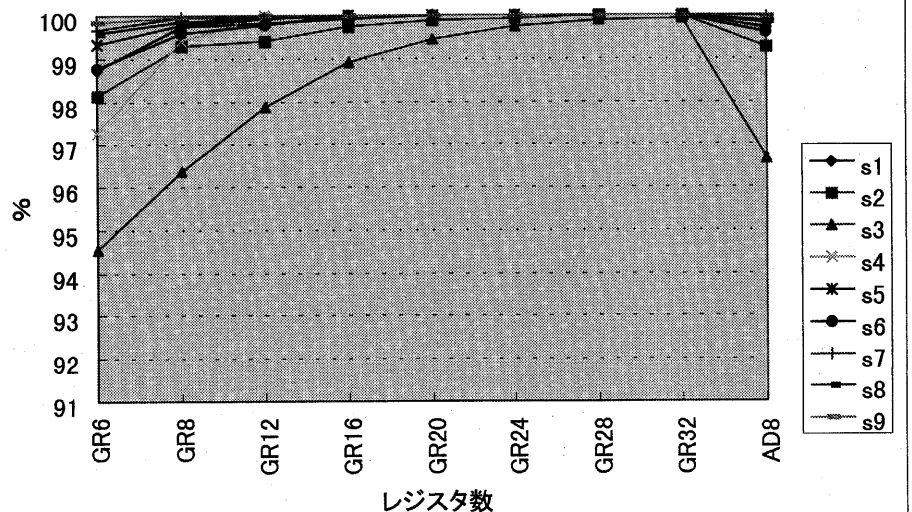


図8　1関数当たりのレジスタ充足率(本方式)



定目的のレジスタに割付け可能であれば割付けるようにしている。また、アドレス／データレジスタ 8本構成では、アドレス／データレジスタの区別なく割付け用レジスタ6本についてグラフカラーリング方式で割付けを行い、演算により必要であれば、アドレス／データレジスタへの転送を行うようにしている。 また、本方式では、変数の型に応じて、アドレスレジスタかデータレジスタかの選択を決定している部分を追加している。 具体的には図4のステップs13においてさらに割付けレジスタ候補が複数あり、その候補の中にアドレスレジスタがあってかつ、変数の型がポインタ型であれば、アドレスレジスタを優先的に割付けるようにしている。

5．2 評価結果

図5はスタンフォードベンチマークについてのステップ数比較であり、従来方法を100とした場合の相対値で示している。100より小さければ本方式のステップ数が少ないことを、大きければステップ数が多かったことを示す。GRは汎用レジスタをADはアドレス／データレジスタ構成を示し、数字はレジスタ本数を示す。 レジスタが少ない場合と、アドレス／データレジスタというようにレジスタに機能制限がある場合に、本方式は有効であることいえる。

図6から図8では、評価サンプルプログラムとして、ベンチマークプログラム、データ処理プログラム、プログラム言語処理、リアルタイムOS、各種組み込みプログラムを用いている。

図6は、サンプルプログラムから本方式で30ステップ以上の関数 約1300個を抜粋して、ステップ数を比較した結果であり、従来方法でのステップ数を100とした相対値で、どれだけの数の関数がそれぞれの相対値に存在するかを示している。例えば、相対値 90のところは、従来方法に比較してステップ数が 90%、つまりステップ数が 10%削減された関数の数を示している。 特に、汎用レジスタ構成8本および16本では、26%減～12%増の範囲で平均4%減、アドレス／データレジスタ構成8本では、40%減～11%増の範囲で平均13%減少した。 ここでも少数で機能制限があるレジスタ構成の場合に、本方式は有効であるといえる。

図7、8はコンパイラが生成する一時変数も含めて、レジスタ割付けの対象となる変数のうち、どれだけの数の変数がレジスタに割付けられかをサンプルプログラムの各関数に対して算出し、その平均をとったものである。 従来と同様に90%以上の充足率を達成している。 なお、サンプルソースのs1からs9までの内訳は以下の通りである。

s1, s2 ... ベンチマークプログラム
s3 ... データ処理
s4 ... リアルタイムOS
s5 ... 言語処理
s6～s7 ... 組み込みプログラム

以上、レジスタ数と構成によって開きがあるが、全体的に本方式が有効であることが示された。

6．結び

本稿では、コンパイラでのレジスタ割付けにおいて、転送命令の削減率をヒューリスティクスに定量化した割付有効度を用いるレジスタ割付け方式を提案し評価を行い、その有効性を示した。

今後、従来方式と比較して本方式が劣っている部分の分析によりさらに改善を図るとともに、本方式を並列化を考慮したレジスタ割付け方式に拡張し、コードサイズと性能面でバランスのとれたレジスタ割付け方式を確立していく予定である。

【参考文献】

1)Chaitin,G.J.,Auslander,M.A.,Chandra,A.KCocke,J.,Hopkins,M. E. and Markstein,P.W.:Register Allocation and Spilling via Coloring , Computer Languages, Vol.6, pp.47-57(1981)

2)F.Chow and J.Hennessy : Register allocation by Priority-base Coloring, In Proceeding of the ACM SIGPLAN '84 Symposium on Computer Construnction, pp.222-232(1984)

3)A.V.Aho, R.Sethi, and J.D.Ullman : Compilers, Principles, Techniques, and Tools, Addison Wesley Publishing Co.,(1986)

4)L.J.Henden et al : A Register Allocation Framework Based on Hierarchical Cyclic Interval Graphs, LNCS 641, pp176-191,October 1992

5)田中,他 : "転送命令削減率の算出によるレジスタ割付け方式"情処全国大会前記, Vol.5, 2N-8(1996)