修 士 論 文

組み込みRTOSでのレジスタセット間高速コンテキスト切り替えに関する研究

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

荻野 雅

2004年3月

修 士 論 文

組み込みRTOSでのレジスタセット間高速コンテキスト切り替えに関する研究

指導教官　田中清史 助教授

審査委員主査　田中清史 助教授
審査委員　日比野靖 教授
審査委員　井口寧 助教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

110027 荻野 雅

提出年月: 2004 年 2 月

概 要

近年，リアルタイム OS は高速な処理を要するシステムに対して適用されているが，ギガビット級の通信機器では，数十μから数μ秒のコンテキスト切り替えによるオーバヘッドが問題となっている．このオーバヘッドは，コンテキスト切り替えのときメモリアクセスすることが原因である．本研究では，複数のレジスタセットを持つ CPU 上で，レジスタセット間でコンテキスト切り替えとタスクの使用レジスタ数の絞り込みを提案し，シミュレーションにより評価を行った．

目 次

図 目 次

表 目 次

第1章　はじめに

1.1　背景，目的

現在，RTOS のコンテキスト切り替えのオーバヘッドに，数 μ 秒から数十 μ 秒の時間を要している．このオーバヘッドは，数 m 秒単位のデッドラインのユーザタスクでは許容することができるが，数 μ 秒単位のデッドラインのユーザタスクでは，このオーバヘッドによりデッドラインオーバーが起こり得る．例えば，ギガビット級の通信機器でのリアルタイムシステムの場合，1 パケット処理は μ 秒単位で行なわれる．この状況下で，μ 秒単位のコンテキスト切り替えでのオーバヘッドは，タスクのデッドラインオーバーの原因となり，それによってパケットの損失を招く．

従来のコンテキスト切り替えは，必ずメモリアクセスを行う方式であった．メモリアクセス時にデータキャッシュミスが発生し，コンテキスト切り替えのオーバヘッドを増大させていた，[1].

ユーザタスクからカーネルタスクへのコンテキスト切り替えの高速化は既に Casablanca[2] にて達成されている．しかし，Casablanca においても，ユーザタスク間でのコンテキスト切り替え時にはメモリアクセスを行わなければならない．

そこで，本研究ではユーザタスク間でのコンテキスト切り替え時のメモリアクセスが問題である点に着目し，コンテキスト切り替えの高速化に向けて，以下の 2 つの手法を提案する．

- レジスタセット間コンテキスト切り替え
- ユーザタスクの使用レジスタ数の絞り込み

提案手法を実現するために，1CPU 内に複数のレジスタセット (8 レジスタセット) を備え，各レジスタセット間でのデータ移動が実現できている CPU である Casablanca を採用する．

提案手法と従来方法との比較，評価を行なうために，Casablanca 準拠の CPU シミュレータを作成した．このシミュレータ上で，提案手法と従来方法とのコンテキスト切り替えルーチンを含んだシステムファイルを実行させ,「デッドラインオーバーしたユーザタスク数」,「ユーザタスクの平均応答時間」,「メモリアクセス回数 (ストアとロードのそれぞれ合わせた回数)」,「キャッシュミス回数」,「コンテキスト切り替え方式」を測定し，2 つの手法についての比較を行い評価する．タスクと時間との関係は，図??に示した通り

とした．ユーザタスクの応答時間は

タスクの終了時刻 – タスクの起動要求時刻

，ユーザタスクの実行時間は,

タスクの終了時刻 – タスクの始動時刻

となる．

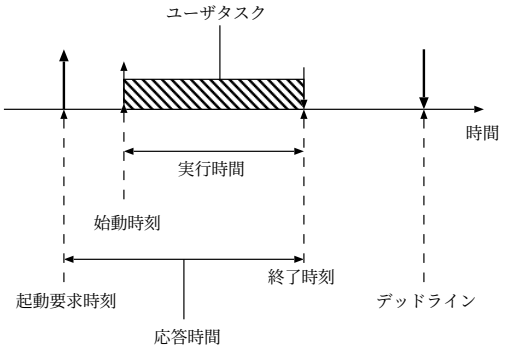

図 1.1: ユーザタスクと時間の関係

1.2 論文構成

本論文の構成を以下に示す．

第 2 章 コンテキスト切り替え

第 3 章 Casablanca

第 4 章 レジスタセット間コンテキスト切り替え

第 5 章 ユーザタスクの使用レジスタ数の絞り込み

第 6 章 評価

第 7 章 関連研究

第 8 章 まとめ

第2章　コンテキスト切り替え

2.1　ユーザタスクの持つコンテキスト

ユーザタスクのコンテキストは以下の情報により構成されている[1].

1. ユーザタスクが使うことができるレジスタ全て (スタックポインタ，フレームポインタ，call 命令後のリターンアドレスを含む)
2. プロセッサ状態レジスタ
3. 乗除算レジスタ
4. プログラムカウンタ
5. ネクストプログラムカウンタ

コンテキスト切り替えの際に，タスクが利用できるレジスタ全てを移動させなければならない (1.)．これは，コンテキスト復帰時に使用していたレジスタのデータが欠けてしまうとタスクが正常に実行することができない．2. は Casablanca の場合を例にあげると，現在のレジスタセットへのポインタと，整数条件コード (Integer Condition Code) をプロセッサ状態レジスタに書き込む．整数条件コードは分岐条件で必要となり，現在のレジスタセットポインタはコンテキスト復帰後，コンテキスト切り替えを行う直前のレジスタセットに戻るために必要な情報である．よって，コンテキストを復帰させた後正常に再開させるにはプロセッサ状態レジスタを移動させる必要がある．3. の乗除算レジスタは乗除算の被除数を保持するために使われる．タスクが乗除算を行っている最中にコンテキスト切り替えが起ることがあるので，コンテキスト復帰後に正常にタスクを再開させるために，移動させる必要がある．4. のプログラムカウンタはコンテキスト切り替え時の命令アドレスが書き込まれるので再開時に中断された命令アドレスへ戻るために必要となるため移動させなければならない，5. のネクストプログラムカウンタは，分岐命令でジャンプする際の遅延スロット命令の段階でコンテキスト切り替えを行う場合，コンテキストを復帰後，タスクを再開させるときに，飛び先アドレスを保持していなければ，ジャンプが行われず，ユーザタスクが正常に実行できなくなる．よって，これら 1. 〜 5. の情報を移動させる必要がある．1.〜5. のデータを退避，復帰させることがコンテキスト切り換えである．

[1] ここでは Casablanca でのコンテキスト切り替えを前提に記述する．ただし，本研究では浮動小数点は扱わず，仮想メモリをしないことを前提にしているので，浮動小数点レジスタ群とメモリ管理ユニットレジスタは除いている．

2.2 コンテキスト切り替えについて

コンテキスト切り替えは以下の条件で起こる.

1. 外部からの割り込み (外部トラップ) やシステムコール (内部トラップ) によりユーザタスクからカーネルへ移行時
2. カーネル処理終了後に実行していたユーザタスクよりも優先度の高いユーザタスクが存在時
3. ユーザタスク終了時
4. 実行条件が揃わず，ユーザタスクが休眠したとき

1. でのコンテキスト切り替えの高速化は第 3 章で示す Casablanca で既に実現されている．3. については，実行していたユーザタスクは完了しているため，ユーザタスクのコンテキストを退避させる必要は無い．4. の場合のコンテキスト切り替えは 1. と同様である．残された 2. の場合のコンテキスト切り替えであるが，Casablanca 上で 2. のコンテキスト切り替えを行うとき，汎用の CPU と同様に従来のメモリアクセスによるコンテキスト切り替えを行わなければならない.

2.3 従来のコンテキスト切り替え

RTOS におけるコンテキスト切り替えは，Linux などの汎用 OS と同様，コンテキストを退避させる際には，タスクが実行の際に使っている 1 レジスタセット全てをメモリに退避させ，復帰させる際には，退避させていた 1 レジスタセット分のコンテキストをメモリから復帰させることで実現している[2](図??に概念図を示す)．この方式では，コンテキスト切り替え時にデータの欠落が無く，安全に移動させる点では十分満たされている．しかし，コンテキストの退避，復帰の際に，メモリアクセスを行うことにより，データキャッシュミスが起こる．データキャッシュのミスペナルティによって，オーバヘッドが増大する [1]．コンテキスト切り替えのオーバヘッド (スケジューリングオーバヘッドも含む) によってユーザタスクがデッドラインオーバを起こす例を図 2.2 に示す.

コンテキスト切り替え時に 1 レジスタセット全てを移動させる理由は，OS 自身がユーザタスクの使用レジスタ数を判別することができなかったためである．これは，ユーザタスクが後から OS に設けられるため，OS の作成の時点ではユーザタスクが実際に使用しているレジスタのみを予測することは出来ないのが原因である．以上の理由により，従来のコンテキスト切り替えの方式では，1 レジスタセットの移動をさせる必要があった．しか

[2]コンテキスト切り替えの際には，%psr(プロセッサ状態レジスタ) など実行中のレジスタセットとは別に存在するレジスタのデータを取り込むために，レジスタセット内のレジスタを確保している．よって実際にユーザタスクが利用できるレジスタ数は 32 を下回る

し，ユーザタスクのコンテキスト内には，レジスタを最大に使わない限り移動させる必要が無いデータが存在し，コンテキスト切り替え時にそれらを移動させなければならない.

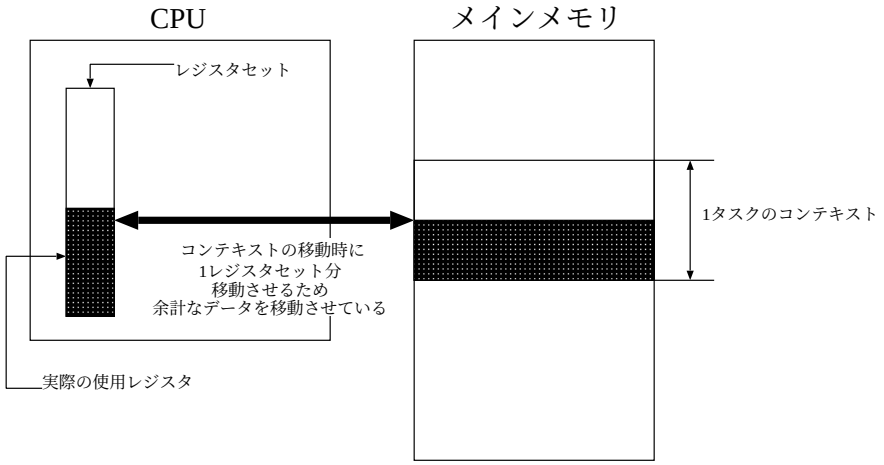

図 2.1: コンテキスト切り替えの概要図

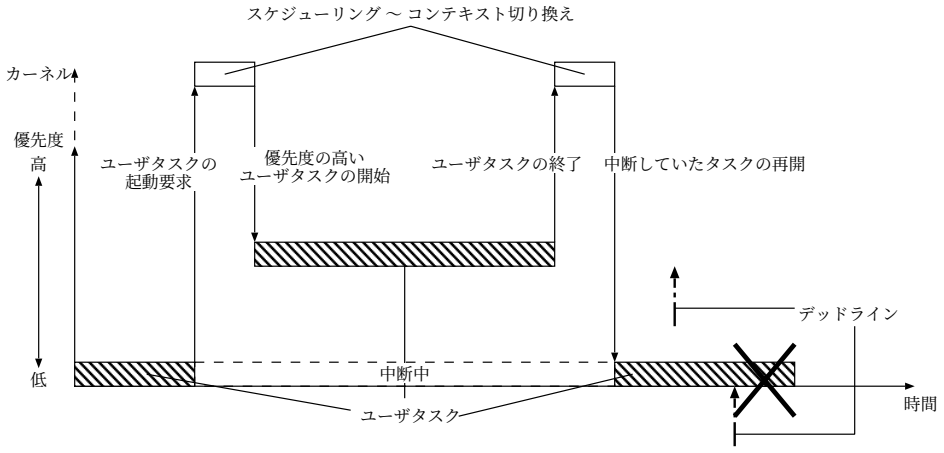

図 2.2: ユーザタスク間でのコンテキスト切り替えによるデッドラインオーバーの例

第3章　Casablanca

3.1　現在のCPU技術

現在，1CPU 内に複数のレジスタセットを持つ CPU[3] が実現し，汎用の CPU としては Intel 社の Xeon[4] や IBM 社の Power4[5] などが普及している，また，組み込み向けの CPU でも，MIPS32 24K[6] が登場により，今後，汎用 CPU のみならず，組み込みシステム向けの CPU でもこのような CPU が普及する傾向がある．そこで，今回コンテキスト切り替えの高速化実現のために，1CPU 内に複数レジスタセット (8 レジスタセット) を備え，各レジスタセット間でのデータ移動が実現されてる Casablanca を採用する．

3.2　Casablancaの特徴

本研究で用いる Casablanca(図 3.2) の機能について説明する．特徴としては

1. SPARC Version 8[7] 互換命令
2. SPARC Version 8 と異なり，レジスタウィンドが無い
3. 8 つのレジスタセットを持つ (1 レジスタセット 32 本レジスタ)
4. SPARC の実装依存命令によりレジスタセット間データ移動命令を実現
5. 高速割り込み応答機能を持つ

が挙げられる．

1. は SPARC Version 8 用の命令コードがほぼそのまま利用できる利点がある[1]．続いて 2. の理由を記す，SPARC Version 8 にある 8 つのレジスタウィンド (図 3.1) は関数移動でのレジスタ退避を行わなくて済む利点があるが，全てのウィンドを使い切る状況が起こる度に，ウィンドオーバーフロー・トラップが起こり，オーバヘッドが発生する [9] [2]．これを避けるためにレジスタウィンドは搭載しない．Casablanca が，Xeon や Power4 など

[1]但し，2. によりレジスタウィンドは備えていないので，GCC[8] でのコンパイル時にレジスタウィンドを使わないコードを生成する”-mflat”のオプションを加える必要がある．

[2]SPARC Version 8 でのユーザタスクのコンテキスト切り替えは，コンテキスト退避時にはウィンドオーバーフローを発生させて，全てのウィンド内のコンテキストをメモリに退避させ，復帰時にはウィンドアンダーフローによってメモリから読み込む．

と同様に CPU 内に複数のレジスタセットを備えている CPU であることを 3. で示している．4. により各レジスタセット間で自由にデータのやりとりが可能である．5. の高速割り込み応答により，ユーザとカーネルとの間でのコンテキスト切り替えの高速化を実現している．ユーザタスク実行中に内部トラップが発生したとき，ユーザタスクのコンテキストを退避させることなく即座に 1 番 (内部トラップ専用レジスタセット) へ移動し，内部トラップトラップルーチンを実行することが可能である (図 3.3 参照)．また，ユーザタスク実行中に外部トラップ (割り込み) が発生したとき，割り込みレベルに応じて 2 から 7 番レジスタセット (外部トラップ専用レジスタセット) へ即座に移動し，割り込みルーチンを実行することが可能である (図 3.4 参照)．

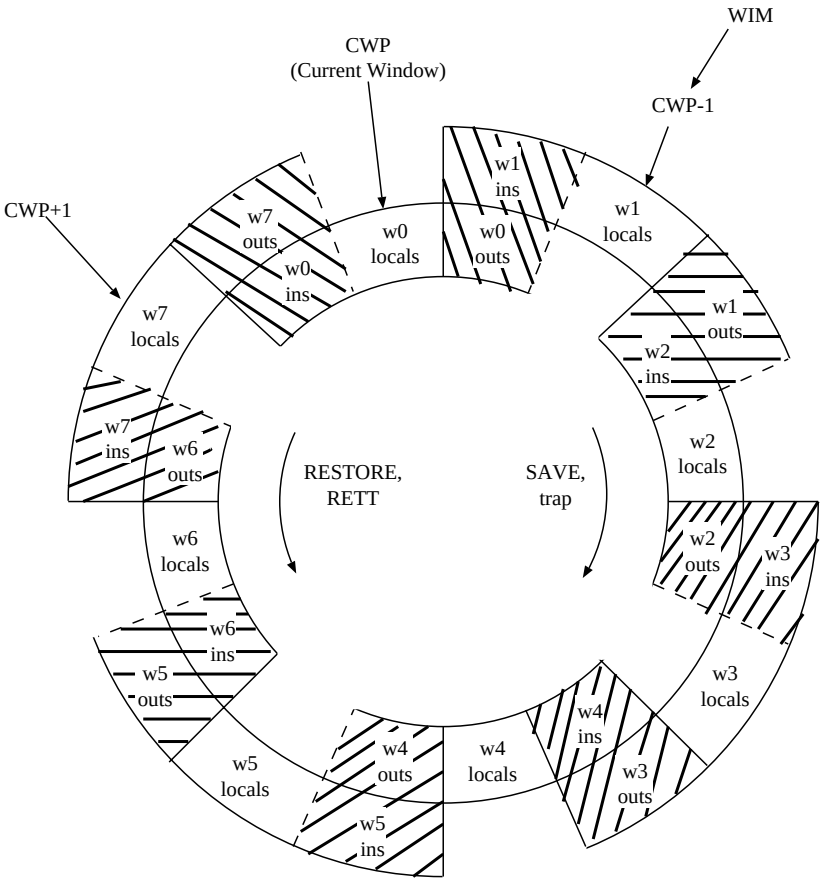

図 3.1: SPARC Version 8 のレジスタウィンド

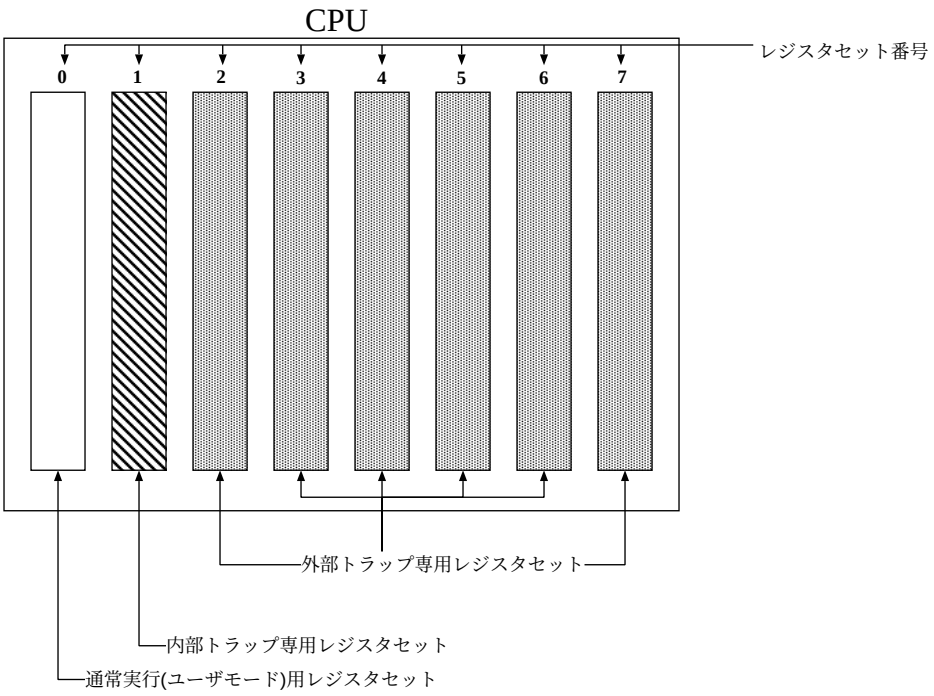

図 3.2: Casablanca の各レジスタセットの用途

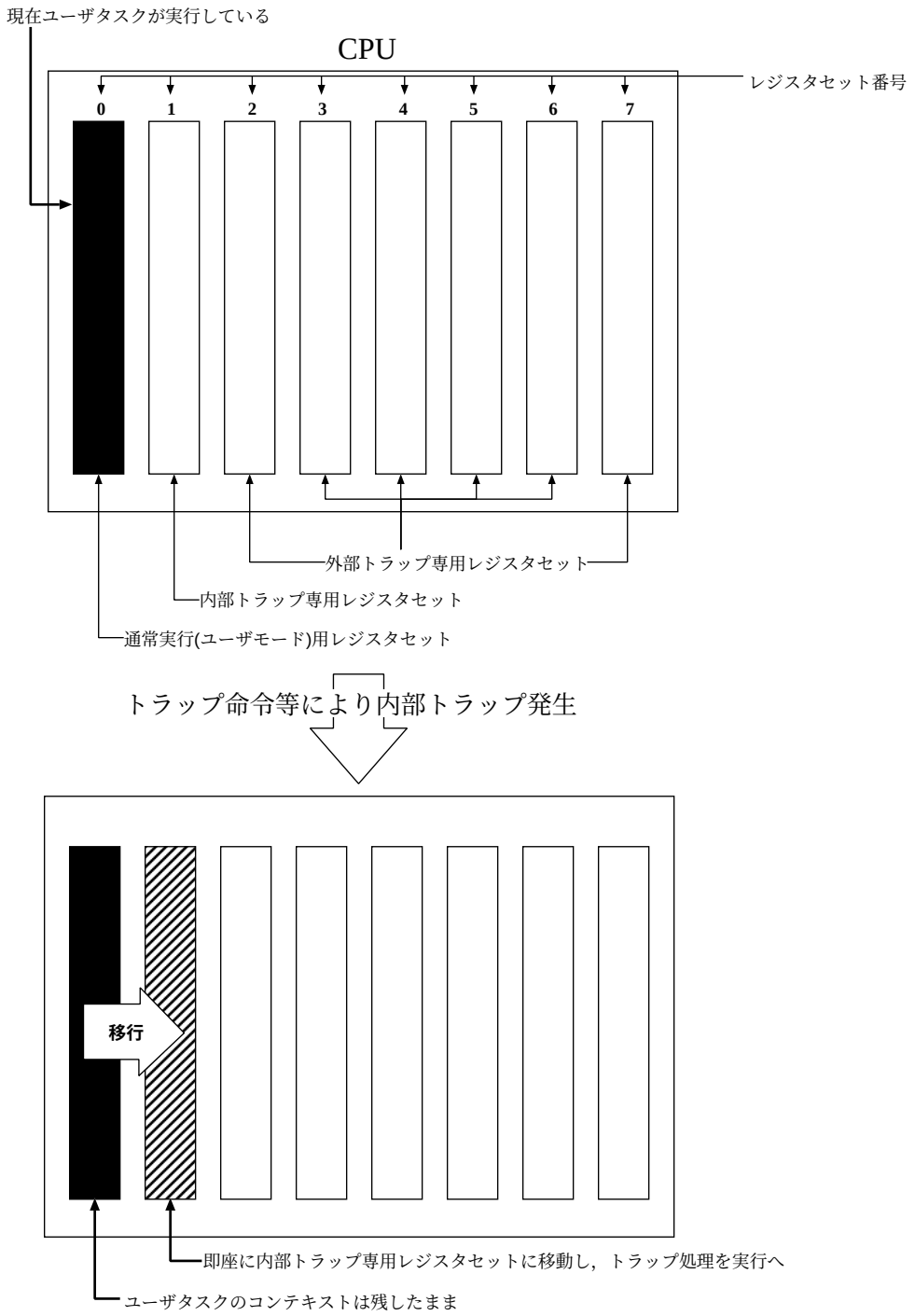

図 3.3: Casablanca 上で内部トラップが発生したときのコンテキスト切り替え

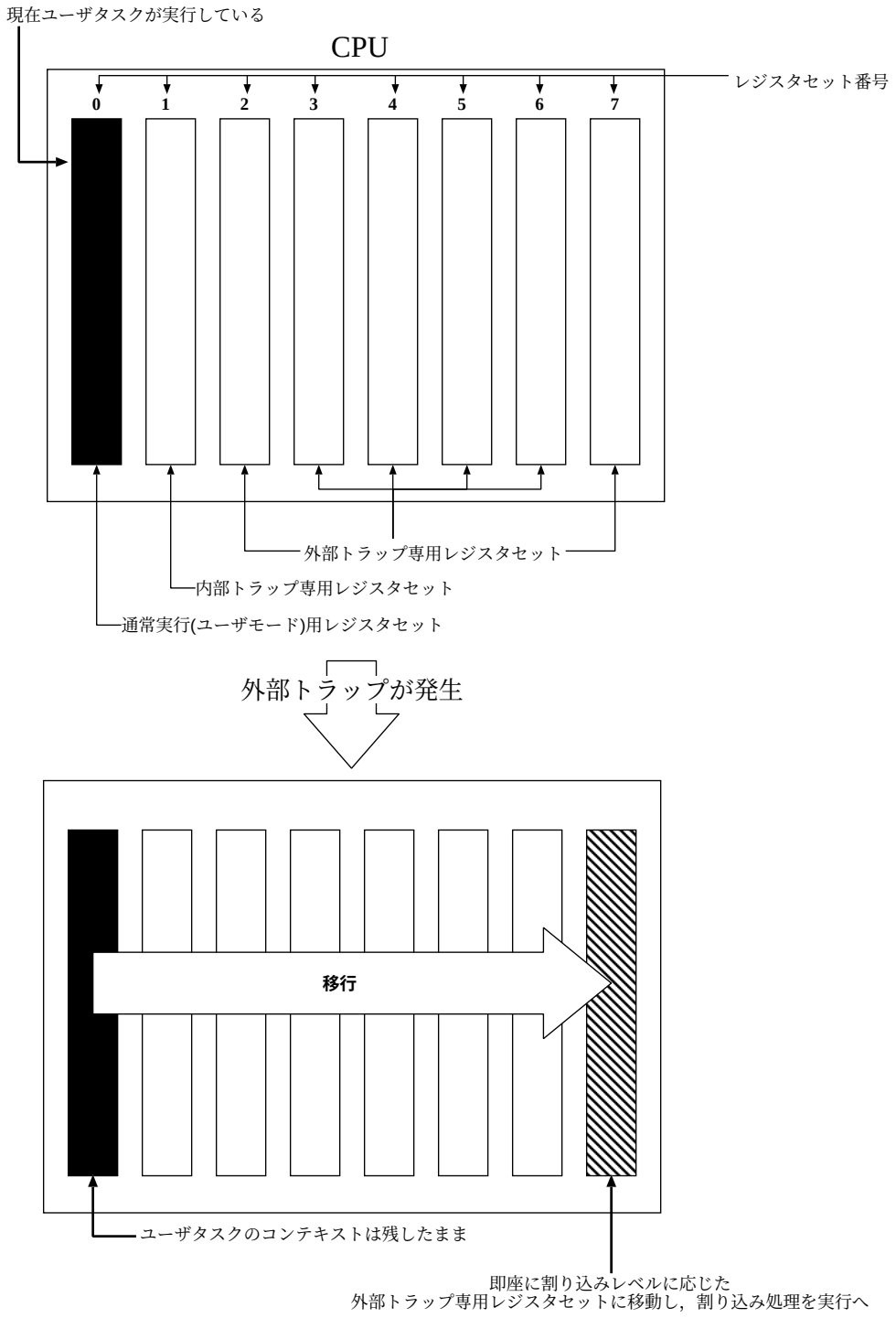

図 3.4: Casablanca 上で外部トラップが発生したときのコンテキスト切り替え

第4章　レジスタセット間コンテキスト切り替え

4.1　前提条件・実現方法

複数のユーザタスクを頻繁に切り替えるシステム上 (例えば，ギガビット級の高速通信用の交換機器) の RTOS では，2 章で述べた従来方式のコンテキスト切り替えを行うとき，コンテキスト切り替えのオーバヘッドを軽減するには，高速な CPU を用いる必要がある．しかし，高速な CPU を使用した場合，消費電力と発熱量が増える．もしくは，データキャッシュを大容量化し，コンテキスト切り替え時のキャッシュミスを軽減することで，オーバヘッドの軽減は可能である．しかし，キャッシュを大容量化する程コストが掛かり (特に CPU に近いキャッシュの容量を大きくする程大量のコストが掛かる)，大容量化によって，キャッシュアクセスが低速化 (通常 1 次キャッシュへのヒット時のアクセスが 1 クロックサイクルで済むものが，1 次キャッシュを大容量化することにより数クロックサイクル掛かる) する．また，キャッシュ容量を大きくすることで消費電力と発熱が増える．

よって，低速でキャッシュ容量の少ない CPU を使う RTOS 上でユーザタスク間のコンテキスト切り替えを高速化するには，メモリアクセスを行わないでタスクのコンテキストを切り替えることが必要である．それには，ユーザタスクのコンテキスト切り換えを CPU 内にあるレジスタセット間で切り替えことで実現できる，この方法により，コンテキスト切り換え時のメモリアクセスによるデータキャッシュミスを軽減される．この手法の実現には以下の 2 条件を満たす必要がある．

- 1CPU 内に複数のレジスタセットの搭載
- レジスタセット間でデータ移動が実装依存命令により実現

1CPU 内に複数レジスタ備える CPU としては，前述の Xeon や Power4 により一般に普及しており，組み込み向けの CPU としても MIPS32 24K が登場した．今回，1CPU 内に 8 レジスタセットを備え，後者のレジスタセット間でのデータ移動が，命令によって実装されている Casablanca をターゲット CPU として採用する．また，ユーザタスクを頻繁に切り替えるシステムをターゲットとした．ただし，例外やパワーオフ，リセットは起こらないとし，外部トラップをユーザタスクの起動要求のみとした．また，起動要求が発生したタスクは，入る度にスケジューリングを行う必要がある．よって，割り込みレベルは 1 つで済むので，全てのユーザタスクの起動要求の割り込みレベルを均一にし，7 番レジスタ

セットで実行させることとした．この条件の元で，Casablanca の 2 から 6 番レジスタセットをコンテキスト保持専用レジスタセットとして使用し，提案手法であるレジスタセット間コンテキスト切替えを実現した．提案手法でのレジスタセットの内訳の概要図を図 4.1 に，コンテキスト切り替えの例 (ユーザタスクのコンテキストを 2 番レジスタセットへの退避，2 番レジスタセットからユーザタスクのコンテキストを復帰) を図 4.2 に示す．

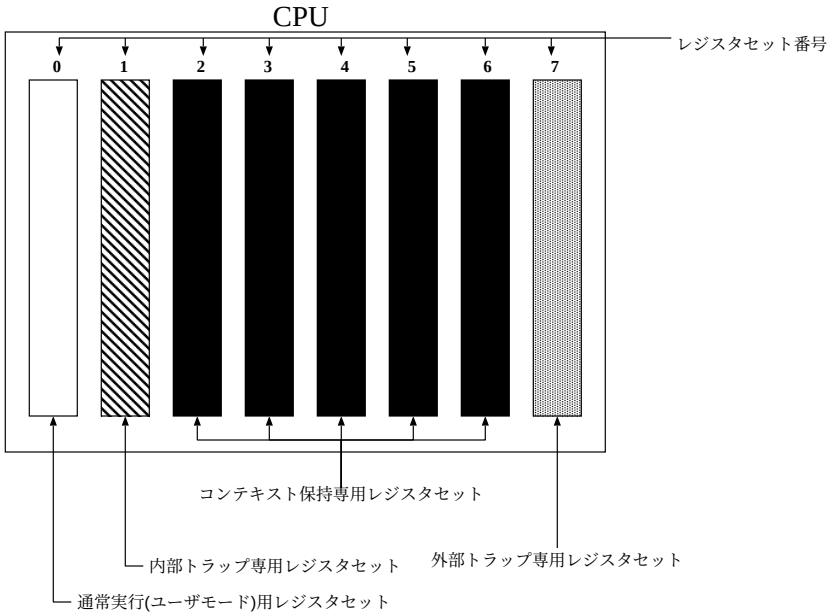

図 4.1: 提案手法での各レジスタセットの用途

4.2 レジスタセット間データ移動命令

今回コンテキスト切り換えに用いるレジスタセット間データ移動命令は SPARC Version 8 の実装依存命令[1]によって実現している．例えば，0 番レジスタセットの 31 番レジスタ (%r31) を 2 番レジスタセットの%r31 へ移動させるには以下に記す命令を使用する．(図 4.3 に概要を示す)

lda　[%r31+%r0] 0xC2, %r31

ユーザタスクのコンテキストである 1 レジスタセットのコンテキスト[2]を 2 番レジスタセットへ退避させるコードを図 4.4 に，1 レジスタセットのコンテキストをメモリへ退避させるときのコードを図 4.5 に記載する．

[1]SPARC Version 8 に実装されている代替空間命令を応用することにより実現している

[2]Casablanca では 1 レジスタセットは 32 レジスタ，全てのレジスタセットの 0 番レジスタ (%r0) はゼロ固定値なので%r0 は移動させる必要がないので除く

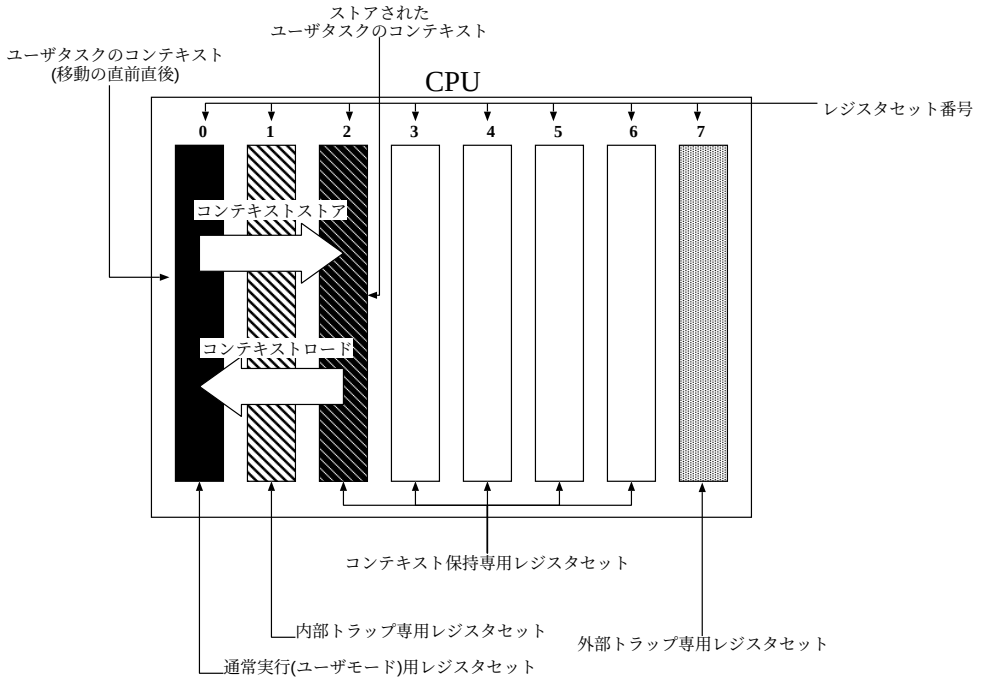

図 4.2: 提案手法でのコンテキスト切り換え例

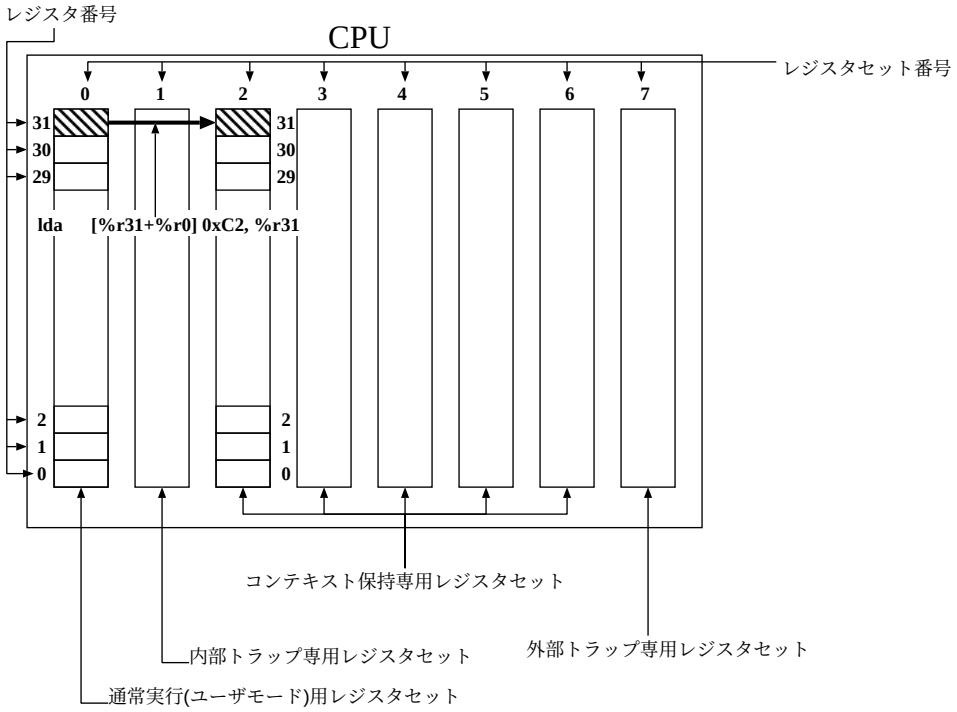

図 4.3: Casablanca での実装依存命令によるレジスタセット間データ移動の例

```
lda   [%r31+%r0] 0xC2, %r31
lda   [%r30+%r0] 0xC2, %r30
lda   [%r29+%r0] 0xC2, %r29
lda   [%r28+%r0] 0xC2, %r28
lda   [%r27+%r0] 0xC2, %r27
lda   [%r26+%r0] 0xC2, %r26
lda   [%r25+%r0] 0xC2, %r25
lda   [%r24+%r0] 0xC2, %r24
lda   [%r23+%r0] 0xC2, %r23
lda   [%r22+%r0] 0xC2, %r22
lda   [%r21+%r0] 0xC2, %r21
lda   [%r20+%r0] 0xC2, %r20
lda   [%r19+%r0] 0xC2, %r19
lda   [%r18+%r0] 0xC2, %r18
lda   [%r17+%r0] 0xC2, %r17
lda   [%r16+%r0] 0xC2, %r16
lda   [%r15+%r0] 0xC2, %r15
lda   [%r14+%r0] 0xC2, %r14
lda   [%r13+%r0] 0xC2, %r13
lda   [%r12+%r0] 0xC2, %r12
lda   [%r11+%r0] 0xC2, %r11
lda   [%r10+%r0] 0xC2, %r10
lda   [%r9+%r0] 0xC2, %r9
lda   [%r8+%r0] 0xC2, %r8
lda   [%r7+%r0] 0xC2, %r7
lda   [%r6+%r0] 0xC2, %r6
lda   [%r5+%r0] 0xC2, %r5
lda   [%r4+%r0] 0xC2, %r4
lda   [%r3+%r0] 0xC2, %r3
lda   [%r2+%r0] 0xC2, %r2
lda   [%r1+%r0] 0xC2, %r1
```

図 4.4: レジスタセット間コンテキスト切り換えのコード (2 番レジスタセットへの退避)

```
sethi %hi(newstack), %r5
or    %r5, %lo(newstack), %r5
st    %r1,[%r5]
std   %r2,[%r5+4]
mov   %r5, %r1
rd    %y, %r5
std   %r4,[%r1+8]
std   %r6,[%r1+16]
std   %r8,[%r1+24]
std   %r10,[%r1+32]
std   %r12,[%r1+40]
std   %r14,[%r1+48]
std   %r16,[%r1+56]
std   %r18,[%r1+64]
std   %r20,[%r1+72]
std   %r22,[%r1+80]
std   %r24,[%r1+88]
std   %r26,[%r1+96]
std   %r28,[%r1+104]
std   %r30,[%r1+112]
```

図 4.5: メモリアクセスによるコンテキスト切り換えのコード (メモリへの退避)

4.3 性能見積り

提案手法 (図 4.4) と従来方式 (図 4.5) のコンテキスト切り替えについて，共にデータキャッシュ，命令キャッシュ共に全くデータが入っていない状態で性能見積りを行った．今回，データキャッシュ，命令キャッシュ共に 2way アソシアティブのブロックサイズ 32bit×8=256bit の 8kByte サイズのキャッシュとし，ミスペナルティは 25 クロックサイクルとする．開始時は，データキャッシュ，命令キャッシュ共に全くデータが入っていないので，最初のアクセスでは Always Miss となる．

表 4.1: コンテキスト切り替えの比較

	データキャッシュミス	命令キャッシュミス	総時間
従来方式	4	3	192
提案手法	0	4	127

表 4.1 より，データキャッシュ，命令キャッシュ共ににデータが全く存在しない状況で実行を開始したとき，コンテキスト切り替えを行ったときに 1.59 倍性能向上した．

4.4 提案手法の優位点

この手法でコンテキスト切り換えを行うことで，以下の点について向上する．

1. ユーザタスクのコンテキスト切り換えの高速化

2. データキャッシュの有効利用

1. については常に 1 レジスタのレジスタセット間移動は 1 クロックサイクルで済み，コンテキスト移動時のデータキャッシュミスが無いことにより，コンテキスト切り換えが高速化され，それによってユーザタスクの応答時間が短縮する．応答時間の短縮化により，ユーザタスクのデッドラインオーバーが軽減される．2. については，ユーザタスクのコンテキストをレジスタセット内に格納することで，タスク実行の際，キャッシュを有効利用できることになり，キャッシュミス数が軽減される．

第5章　ユーザタスクの使用レジスタ数の絞り込み

5.1　前提条件・実現方法

第2章で述べた理由により，今まではコンテキスト切り換えを行う時，1レジスタセット全てを移動させる必要があった．この方法では，コンテキスト切り替え時にタスクのコンテキストを損失無く移動でき，コンテキスト切り換えのルーチンは1通りで済まされるため，小サイズで収まる．しかし，コンテキスト切り換えの際に，ユーザタスクが使用していないレジスタのデータを移動させるときがある．また，第4章で提案したレジスタセット間コンテキスト切り換え手法でコンテキストを切り替える際には従来のコンテキスト切り替えに比べて，命令の数が多くなる[1]．

実際のRTOS上で運用されるユーザタスクの中には，コンパイラがレジスタ数を最大限に割り当てたときでも，1レジスタセットの半分程しか使わない単純かつ小規模なユーザタスクが複数混在している．本研究ではユーザタスク生成時に使用レジスタ数をタスク情報として持たせることで，ユーザタスクの使用レジスタ数の絞り込み手法を実現する．また，レジスタの使用用途(通常実行用レジスタセットでユーザタスクを用いる場合のみ)については，SPARC Version 8とは異なり図5.1で示す方法で行うので，コンパイラの改良を行う必要がある．

使用レジスタ数にあわせてコンテキスト切り替えルーチンを作成する際，レジスタ1本毎についてコンテキスト切替えのルーチンを作成するのでは，ルーチンサイズが肥大化するので，使用レジスタ数で扱う単位を16，32の2通りに限定した．使用レジスタ数＝16の場合,「%r1 〜 %r15」までを使用しているタスクが対象となる(図5.2参照)．ただし，コンパイル時にコンパイラはレジスタ数を16に限定はしない．

ユーザタスクの使用レジスタ数の絞り込み手法の実現により，コンテキスト保持専用レジスタセットに変更を加える．1レジスタセットを16本単位で構成されるフレーム(図5.3)単位で管理を行う．各フレームの管理情報については，メインメモリ上にフレーム管理テーブルを設け，それぞれのフレームについては，フレーム内にタスクを格納しているときにはタスクのIDを，格納していないときは0を書き込む(図5.4の例ではフレーム1

[1]SPARC Version 8ではレジスタとメモリ間でストア，ロードを行う際には合成命令であるstd，ldd命令を使うことにより，1命令で最大2レジスタ分のデータを移動させることが可能である．しかし，レジスタセット間でのデータ移動は，レジスタがアドレッサブルでないことにより，1命令で1レジスタ分のデータ移動しかできないので命令数は増える．

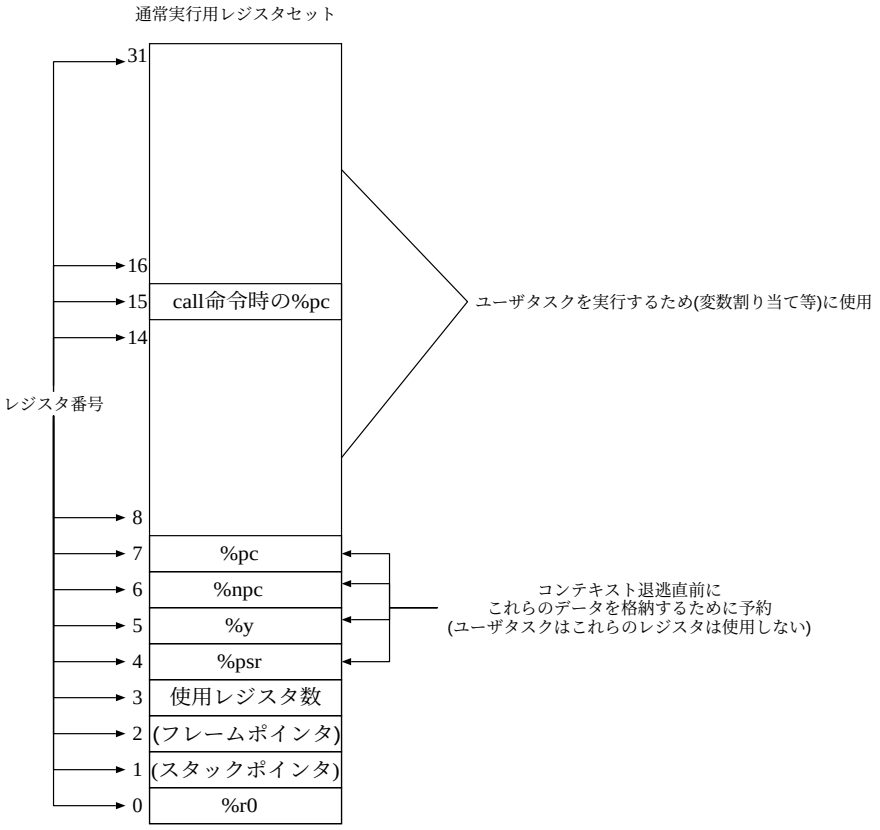

図 5.1: 提案手法での各レジスタ

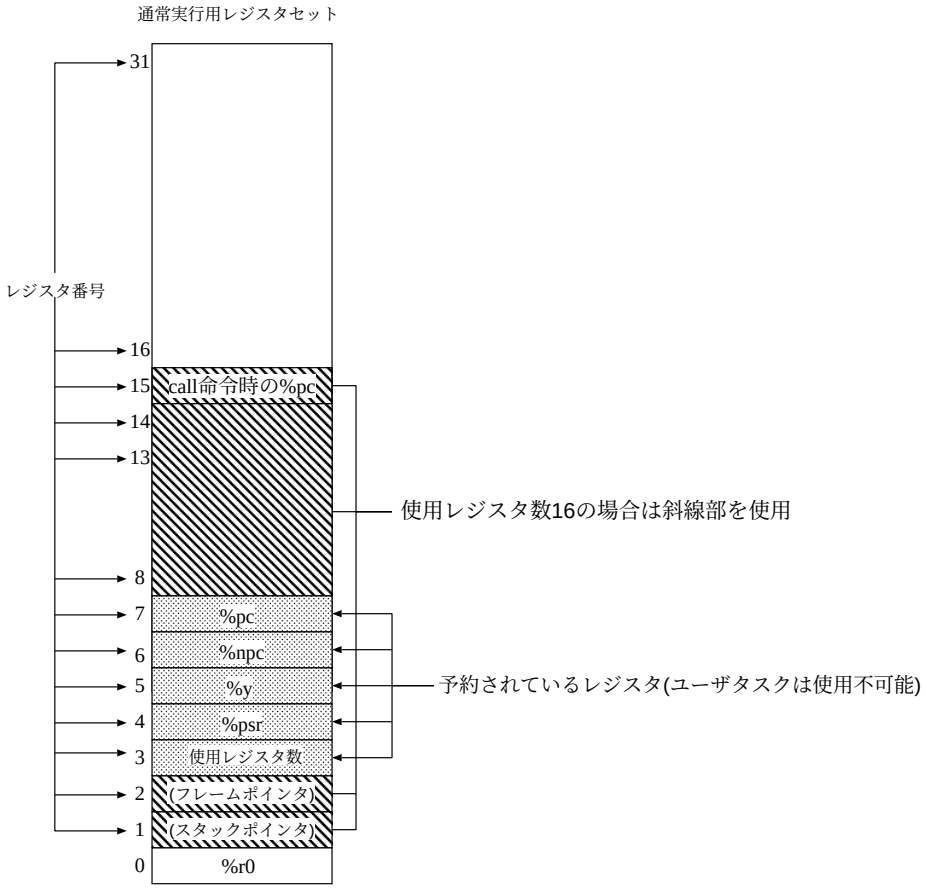

図 5.2: 使用レジスタ数＝ 16 の場合の使用レジスタの概要図

と 2 をタスク ID:1 のコンテキストが使用し，フレーム 3 をタスク ID:2 が使用している)．また，実際 1 フレーム分のユーザタスクのコンテキストをフレーム 1 と通常実行用レジスタセットとの間で切り換えを行うときの概要図を図 5.5 に示す．

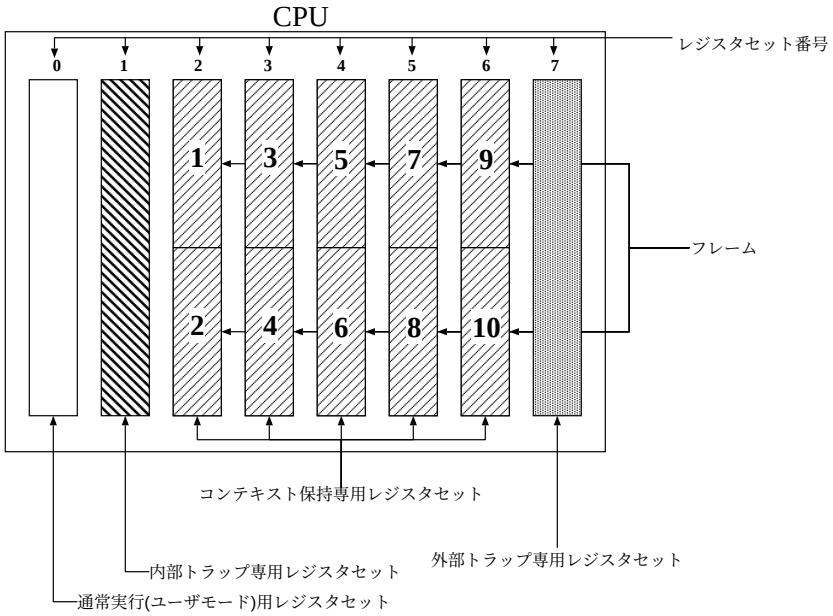

図 5.3: コンテキスト保持専用レジスタセットをフレーム化した概要図

コンテキスト保持専用レジスタセットへのユーザタスクコンテキストの格納は，1 から順に空いたフレームに格納していく．フレームに空きが無いときは，メモリへストアする．1 フレーム分のタスクのコンテキストをフレーム 1 へ退避させる例を図 5.6 に，1 フレーム分のタスクのコンテキストを従来のコンテキスト切り替えでメモリへ退避させる例を図 5.7 に記載する．これにより，コンテキスト切り替えの際に移動させるレジスタ数は少なくなる．

5.2 性能見積り

4.3 と同条件下で 1 フレーム分のコンテキスト切り替えと絞り込みを行わない 2 フレーム分のコンテキスト切り替えについてレジスタセット間でのコンテキスト切り替え (図 5.6，図 4.4) と従来方式によるコンテキスト切り替え (図 5.7，図 4.5) について性能見積りを行った．

表 5.1 より，データキャッシュ，命令キャッシュ内にデータが全存在しない状況でコンテキスト切り替えを行ったときに使用レジスタ数の絞り込みを行ったときレジスタセットコンテキスト切り替えでは 2.02 倍，メモリアクセスでのコンテキスト切り替えでは 1.75 倍性能向上した．

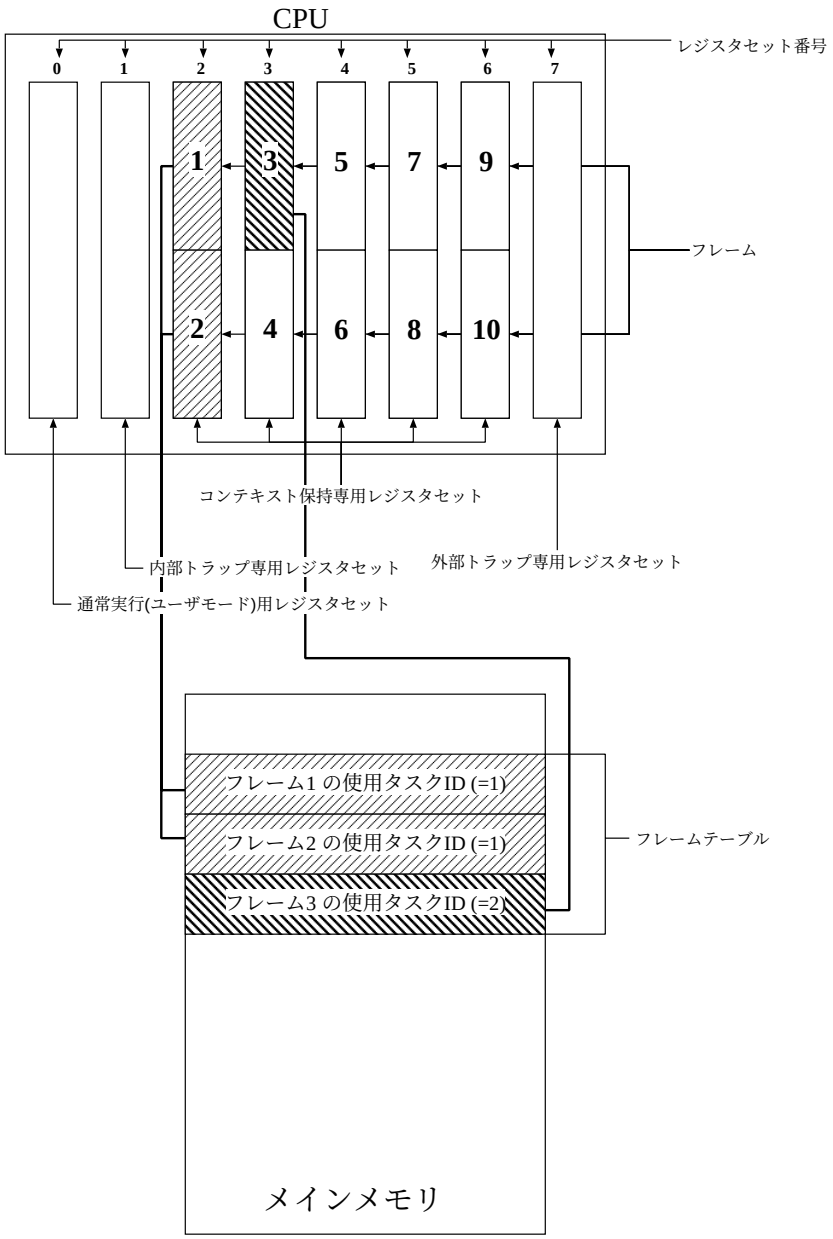

図 5.4: フレームとフレームテーブルとの関係

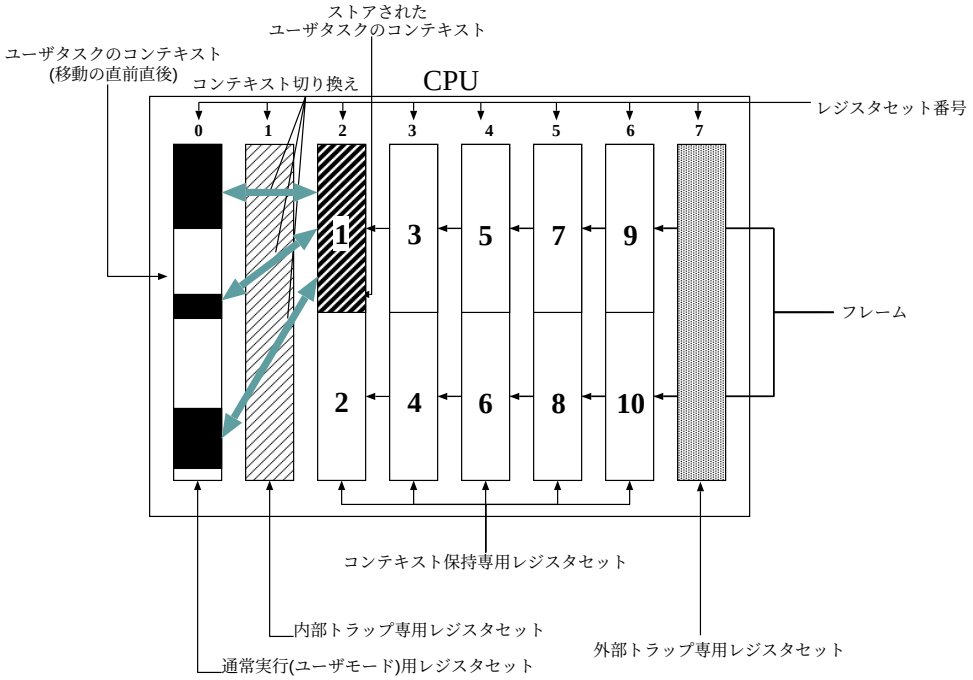

図 5.5: ユーザタスクの使用レジスタ数絞り込みを行ったときのコンテキスト切り換えの概要図

```
lda   [%r15+%r0] 0xC2, %r31
lda   [%r14+%r0] 0xC2, %r30
lda   [%r13+%r0] 0xC2, %r29
lda   [%r12+%r0] 0xC2, %r28
lda   [%r11+%r0] 0xC2, %r27
lda   [%r10+%r0] 0xC2, %r26
lda   [%r9+%r0] 0xC2, %r25
lda   [%r8+%r0] 0xC2, %r24
lda   [%r7+%r0] 0xC2, %r23
lda   [%r6+%r0] 0xC2, %r22
lda   [%r5+%r0] 0xC2, %r21
lda   [%r4+%r0] 0xC2, %r19
lda   [%r3+%r0] 0xC2, %r18
lda   [%r2+%r0] 0xC2, %r17
lda   [%r1+%r0] 0xC2, %r16
```

図 5.6: 1 フレーム分のコンテキストをレジスタセット間で切り換えるコード (フレーム 1 への退避)

```
sethi %hi(newstack), %r5
or    %r5, %lo(newstack), %r5
st   %r1,[%r5]
std   %r2,[%r5+4]
mov   %r5, %r1
rd    %y, %r5
std   %r4,[%r1+8]
std   %r6,[%r1+16]
std   %r8,[%r1+24]
std   %r10,[%r1+32]
std   %r12,[%r1+40]
std   %r14,[%r1+48]
```

図 5.7: 1 フレーム分のコンテキストをメモリアクセスによって切り替えるコード (メモリへの退避)

表 5.1: コンテキスト切り替えの比較

	データキャッシュミス	命令キャッシュミス	総時間
レジスタセット間，絞り込み無し	0	4	127
レジスタセット間，絞り込み有り	0	2	63
メモリアクセス，絞り込み無し	4	3	192
メモリアクセス，絞り込み有り	2	2	110

5.3 提案手法の優位点

本手法により，以下について改善される．

1. コンテキスト切り換え時の命令数の削減
2. コンテキスト保持専用レジスタセット内に多くのコンテキストを格納可能
3. データキャッシュの更なる有効活用

使用レジスタ数が 16 であった場合，従来は 32 本のレジスタ移動させていたのが，16 本のレジスタのみの移動でユーザタスクのコンテキスト切り換えが実現できる．そして，コンテキスト切り替え時の命令数が軽減される．これにより，1. が実現でき，1 つのコンテキスト保持専用レジスタセットにつき最大 2 つのユーザタスクのコンテキストを格納することが可能となり，2. が実現できる．1. と 2. により，コンテキスト切り換え時のメモリアクセス回数が減少し，それに伴い，データキャッシュを書き換える頻度が減って，データキャッシュミスが軽減され，3. が実現できる．

第6章　評価

6.1　シミュレータ

本研究では，Casablanca 準拠の CPU シミュレータを作成した．アセンブリ言語で書かれたシステムファイル (割り込みルーチン，スケジューラルーチン，コンテキスト切り替えルーチン，ユーザタスク) を入力，実行することで評価を行なう．

6.2　シミュレータ仕様

1. 8 レジスタセットをサポート
2. 1 命令，1 クロックサイクル
3. データキャッシュは，サイズ 8kByte，2way，ブロックサイズが 32bit×8=256bit とし，キャッシュミスペナルティは 25 クロックサイクル
4. 命令キャッシュは，サイズ 8kByte，2way，ブロックサイズが 32bit×8=256bit とし，キャッシュミスペナルティは 25 クロックサイクル
5. 2 次キャッシュは無し
6. 仮想メモリは使用しない
7. 外部トラップはユーザタスクの起動要求のみとし，割り込みキューに蓄えてから，割り込み可能状態であれば，キューの先頭から実行していく (図 6.1)．これにより，CPU 内の割り込みコントローラの機能を果たす
8. 割り込みルーチンは全て 7 番レジスタセットのみで実行する (ユーザタスクの起動要求の割り込みレベルを均一化した)(シングルレジスタセット時除く)
9. プログラムカウンタを参照することで，起動要求待ち状態とユーザタスクを判別
10. 9. により，起動要求待ち状態から起動要求処理への移行の際に，起動要求待ち状態のコンテキストを退避させないようにする

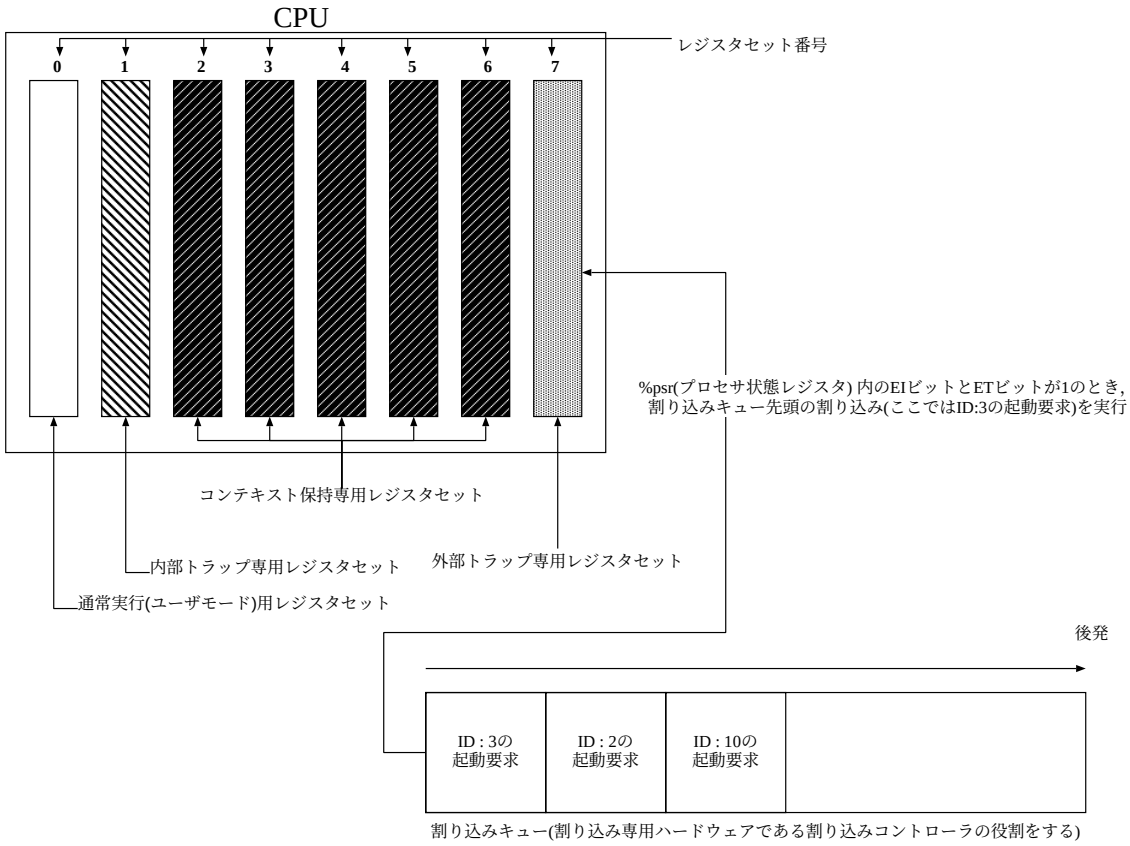

図 6.1: 割り込みキューの概要

6.3 評価方法

評価を行うにあたり，以下の方法を用いる．

- シングルレジスタセット，タスクの使用レジスタ数の絞り込み無し
- シングルレジスタセット，タスクの使用レジスタ数の絞り込み有り
- Casablanca 上でユーザタスクについては従来のコンテキスト切り換えを行い，タスクの使用レジスタ数の絞り込み無し
- Casablanca 上でユーザタスクについては従来のコンテキスト切り換えを行い，タスクの使用レジスタ数の絞り込み有り
- 1 レジスタセット (2 番レジスタセット) をコンテキスト保持専用レジスタセットとして使い，タスクの使用レジスタ数の絞り込み無し
- 1 レジスタセット (2 番レジスタセット) をコンテキスト保持専用レジスタセットとして使い，タスクの使用レジスタ数の絞り込み有り
- 3 レジスタセット (2 番 〜 4 番レジスタセット) をコンテキスト保持専用レジスタセットとして使い，タスクの使用レジスタ数の絞り込み無し
- 3 レジスタセット (2 番 〜 4 番レジスタセット) をコンテキスト保持専用レジスタセットとして使い，タスクの使用レジスタ数の絞り込み有り
- 5 レジスタセット (2 番 〜 6 番レジスタセット) をコンテキスト保持専用レジスタセットとして使い，タスクの使用レジスタ数の絞り込み無し
- 5 レジスタセット (2 番 〜 6 番レジスタセット) をコンテキスト保持専用レジスタセットとして使い，タスクの使用レジスタ数の絞り込み有り

6.4 入力ファイル

入力ファイルは以下のようにする．

1. 先頭にシステム起動準備ルーチン (内部トラップ実行) を置く
2. その直後に起動要求待ちのアイドルタスク (ユーザモード) を置き，起動準備が完了もしくは，レディーキューが空になった場合に飛ぶ
3. 割り込みハンドラを置き，起動要求処理を行う (外部トラップ実行)

4. スケジューラルーチンを設け，その中でコンテキストのストア，ロードルーチンを置く (外部トラップ実行部と内部トラップ実行部の 2 パート構成)

5. コンテキスト切り替えの部分は 6.3 で示した方法に合わせて異なる

6. 最後に ID : 1 〜 50 のユーザタスクの実体を置く

7. 入力ファイルは 6.3 で示したコンテキスト切り替えの手法に合わせ，10 通り用意する

入力ファイルをシングルレジスタセットで実行するときと，8 レジスタセットの Casablanca 準拠シミュレータ上で実行するとき (Casablanca に提案手法を適用するときと適用しないとき) のフローチャートを図 6.2 〜 図 6.12 で示す．図 6.2 では，システム起動から起動要求待ち状態に進み，ユーザタスクの起動要求によってスケジューリングを行い，ユーザタスクを行うまでの流れを示している．この流れは，シングルレジスタセット実行時 (使用レジスタ数の絞り込み有無の双方)，Casablanca にてレジスタセット間コンテキスト切り替えを使用しないときの実行 (使用レジスタ数の絞り込みを行うときと行わないときの双方)，Casablanca にてレジスタセット間コンテキスト切り替えを使用したときの実行 (使用レジスタ数の絞り込みを行うときと行わないときの双方) 全て共通である．図 6.3〜図 6.8 では，ユーザタスク実行中にユーザタスクの起動要求が起るときの流れである．起動要求発生後，スケジューリング行い，スケジューリングの結果，新しいユーザタスク実行もしくは，今まで実行していたユーザタスクの再開に至る．図 6.3 では，シングルレジスタセットにて使用レジスタ数の絞り込みを行わないとき，図 6.4 では，シングルレジスタセットにて使用レジスタ数の絞り込みが行うときである．同様の順で，図 6.5 と図 6.6 では Casablanca にてレジスタセット間コンテキスト切り替えを使用しないとき，図 6.7 と図 6.8 では Casablanca にてレジスタセット間コンテキスト切り替えを使用するときである．図 6.9〜図 6.12 はユーザタスク終了してから，スケジューリングを行い，新規ユーザタスク実行，もしくは，中断していたユーザタスクの再開，レディキューが空なら起動要求待ち状態へ移行するまでの流れを示す．図 6.9 は，シングルレジスタセットにて使用レジスタ数の絞り込みが行わないときと，Casablanca にてレジスタセット間コンテキスト切り替えを使用せず，且つ使用レジスタ数の絞り込みを行わないときである．図 6.10 は，シングルレジスタセットにて使用レジスタ数の絞り込みを行うときと，Casablanca にてレジスタセット間コンテキスト切り替えを使用しないが，使用レジスタ数の絞り込みを行うときである．図 6.11 は Casablanca にてレジスタセット間コンテキスト切り替えを使用するが，使用レジスタ数の絞り込みを行わないときである．図 6.12 は Casablanca にてレジスタセット間コンテキスト切り替えを使用し，使用レジスタ数の絞り込みを行うときである．

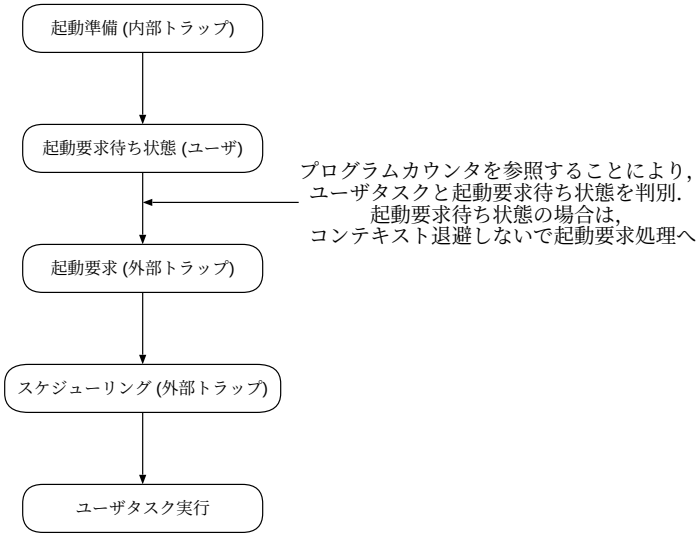

図 6.2: 起動準備〜起動要求待ち状態〜起動要求〜スケジューリング〜ユーザタスク実行まで (シングルレジスタセット (絞り込み有無)，Casablanca(絞り込み有無)，レジスタセット間コンテキスト切り替え (絞り込み有無) 共通)

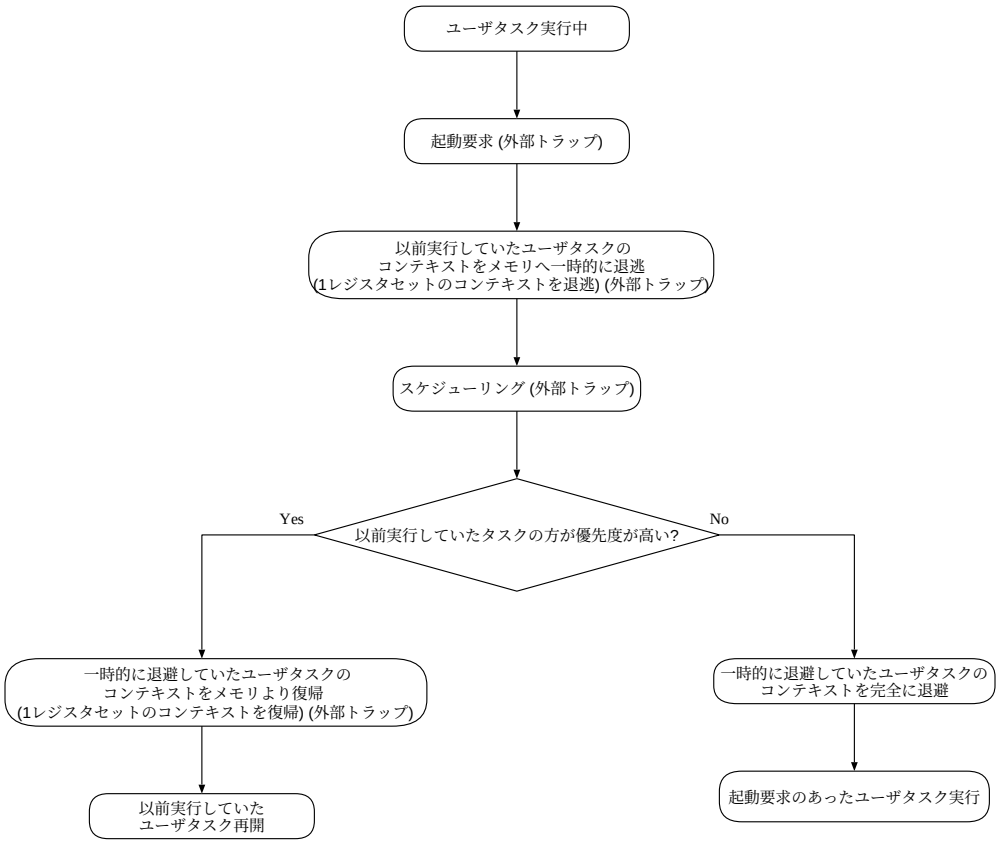

図 6.3: ユーザタスク実行中～起動要求～スケジューリング～ユーザタスク実行まで (シングルレジスタセット，絞り込み無し)

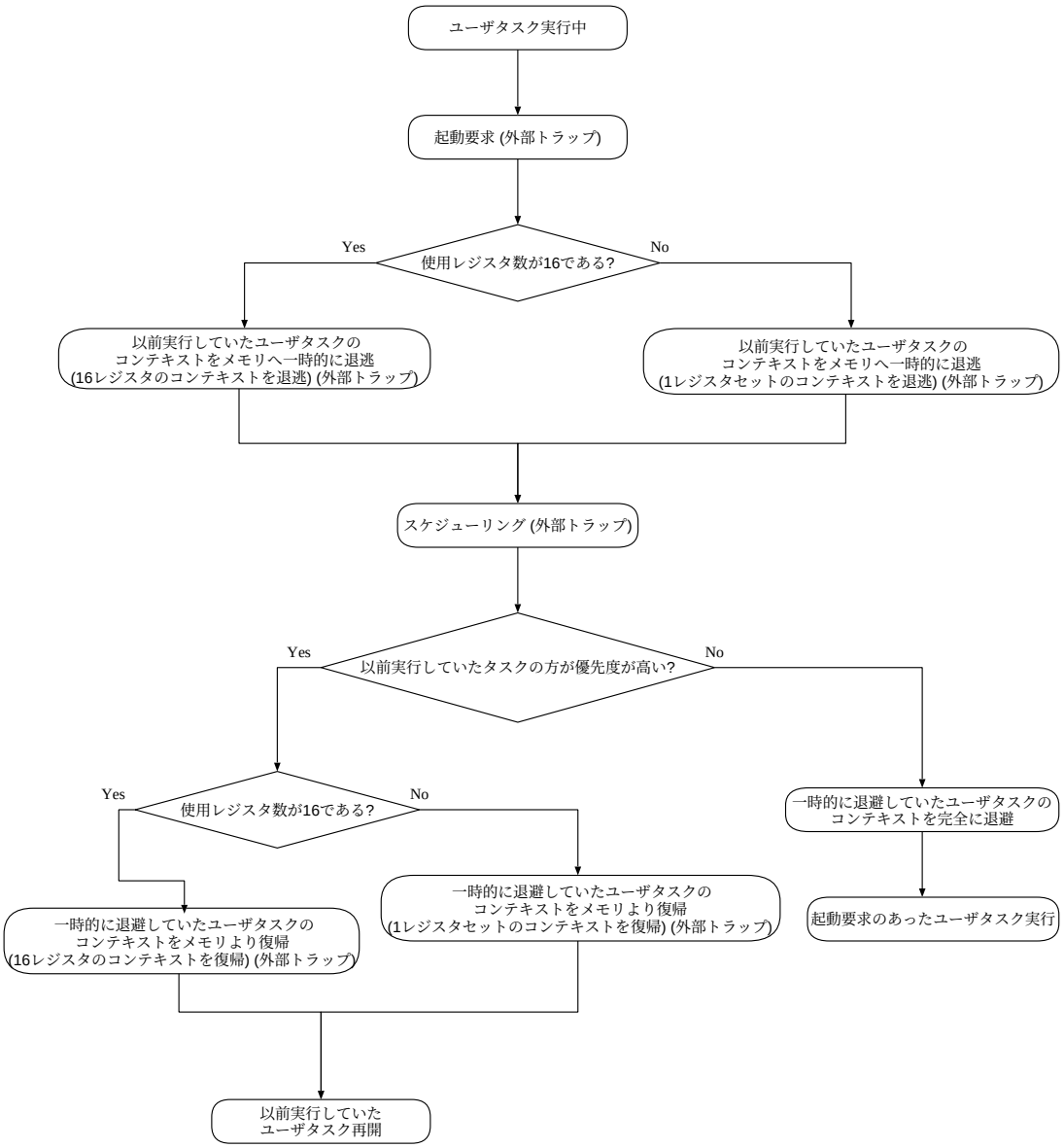

図 6.4: ユーザタスク実行中〜起動要求〜スケジューリング〜ユーザタスク実行まで (シングルレジスタセット，絞り込み有り)

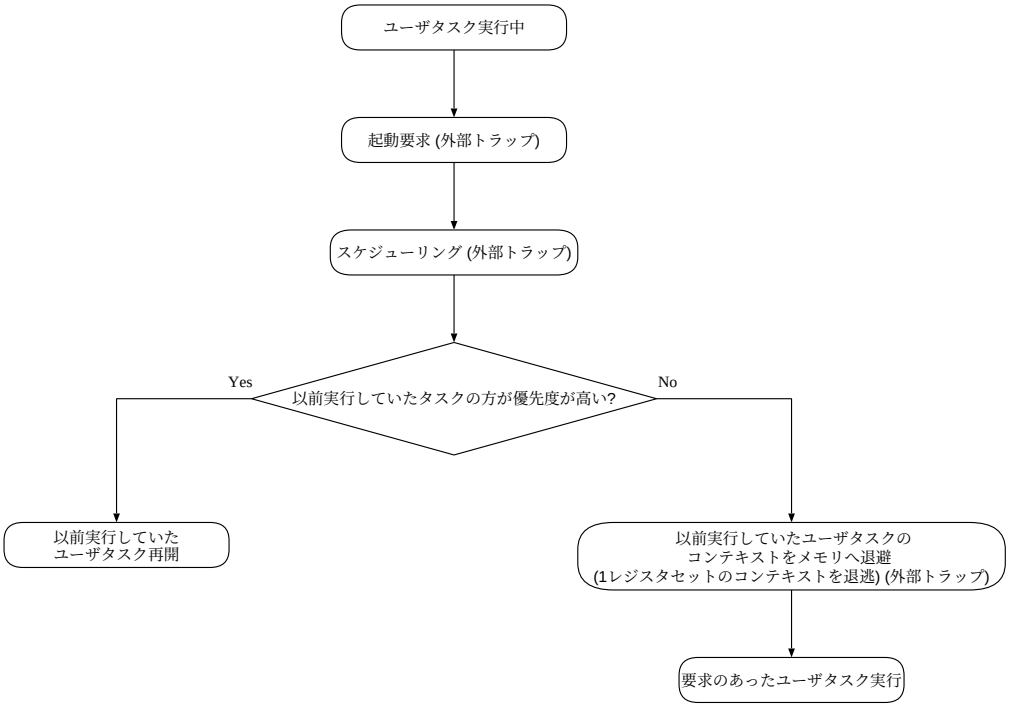

図 6.5: ユーザタスク実行中〜起動要求〜スケジューリング〜ユーザタスク実行まで (Casablanca，絞り込み無し)

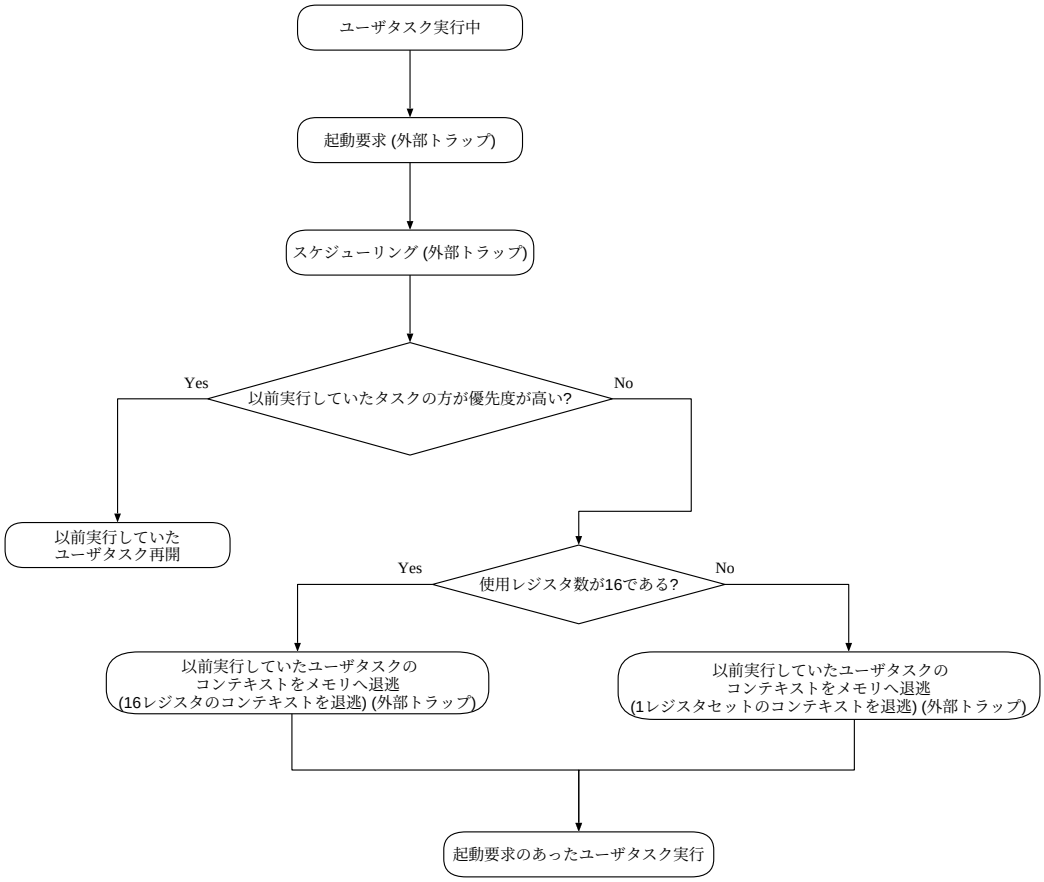

図 6.6: ユーザタスク実行中～起動要求～スケジューリング～ユーザタスク実行まで (Casablanca，絞り込み有り)

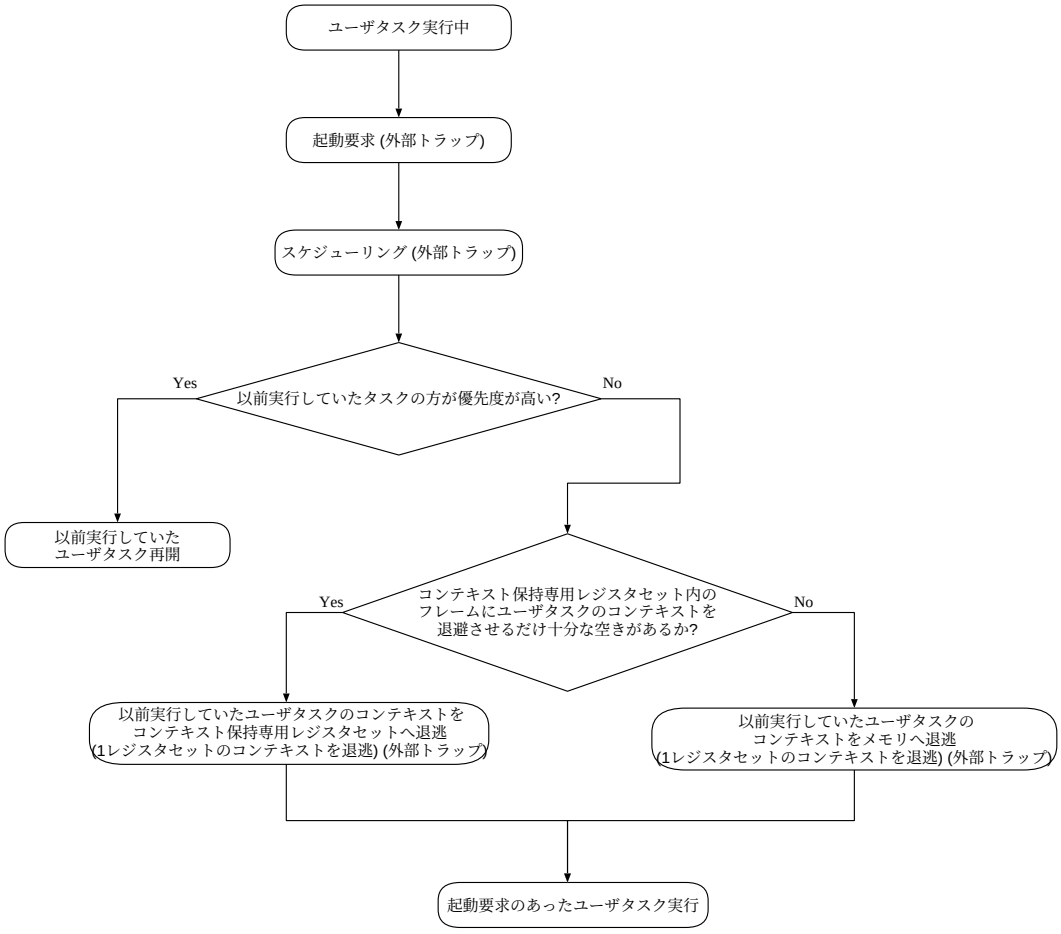

図 6.7: ユーザタスク実行中〜起動要求〜スケジューリング〜ユーザタスク実行まで (レジスタセット間コンテキスト切り替え，絞り込み無し)

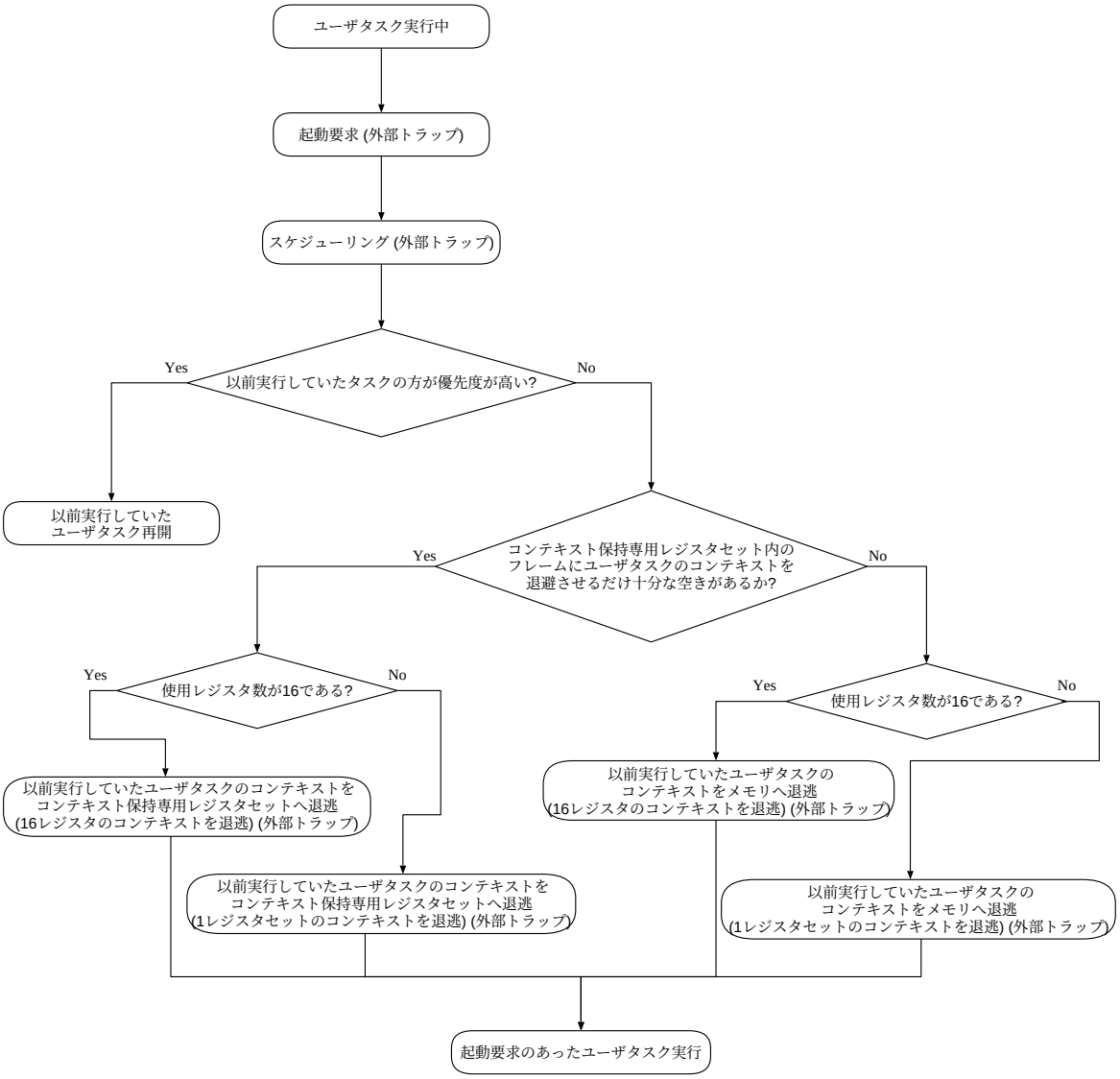

図 6.8: ユーザタスク実行中〜起動要求〜スケジューリング〜ユーザタスク実行まで (レジスタセット間コンテキスト切り替え，絞り込み有り)

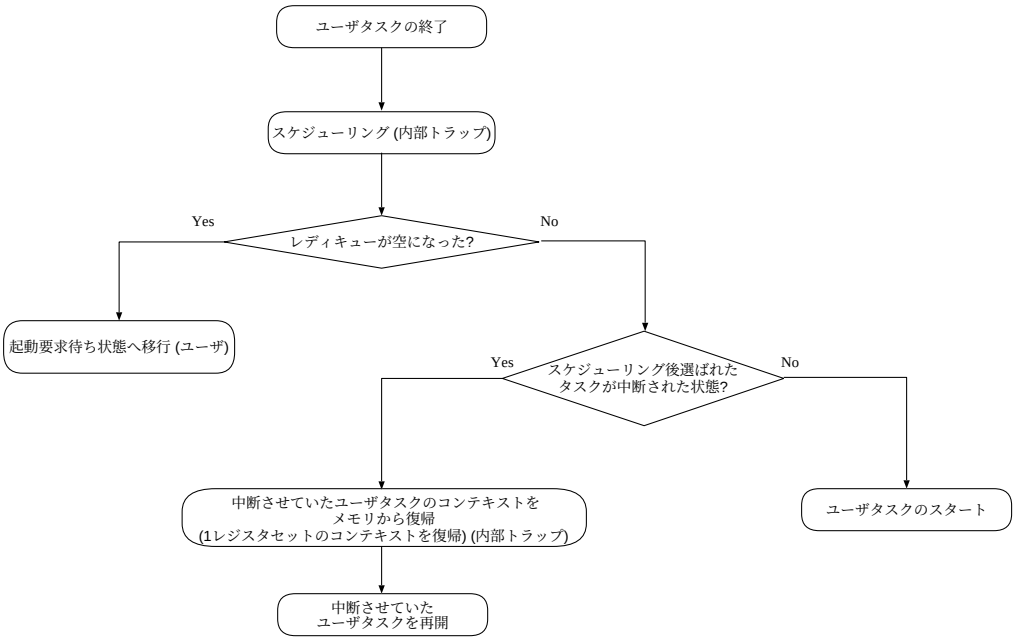

図 6.9: ユーザタスク終了〜スケジューリング〜ユーザタスク実行 or 起動要求待ちまで (シングルレジスタセット，Casablanca 共に絞り込み無し)

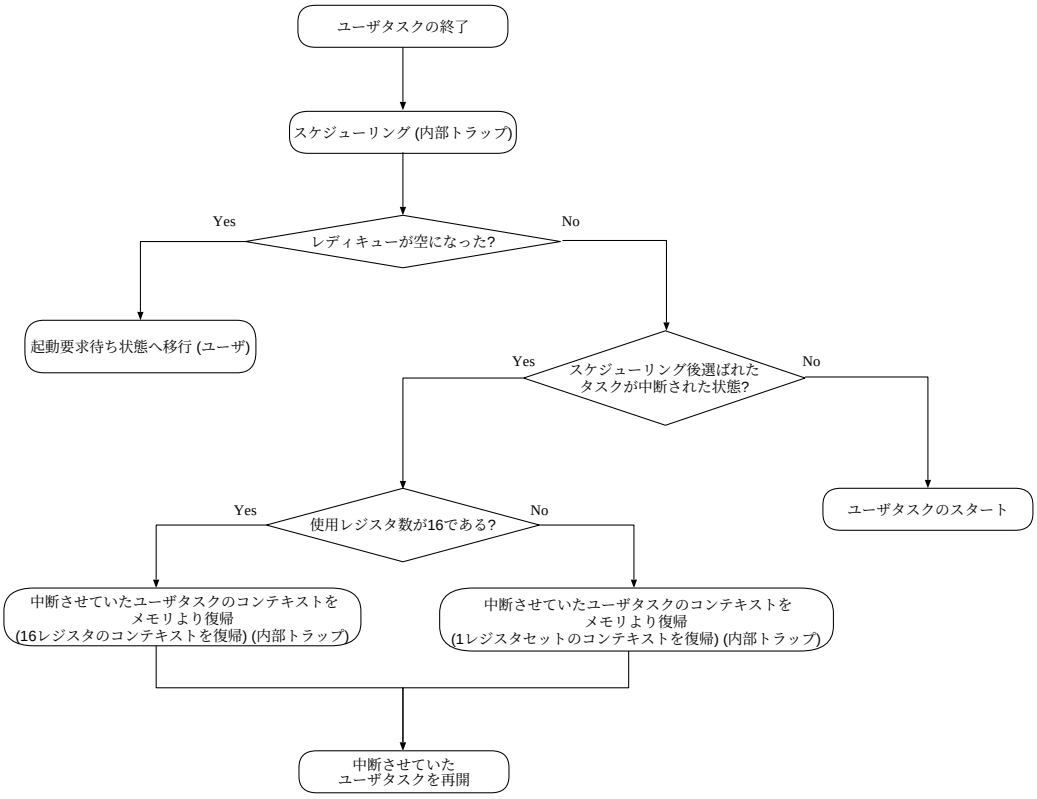

図 6.10: ユーザタスク終了〜スケジューリング〜ユーザタスク実行 or 起動要求待ちまで (シングルレジスタセット，Casablanca 共に絞り込み有り)

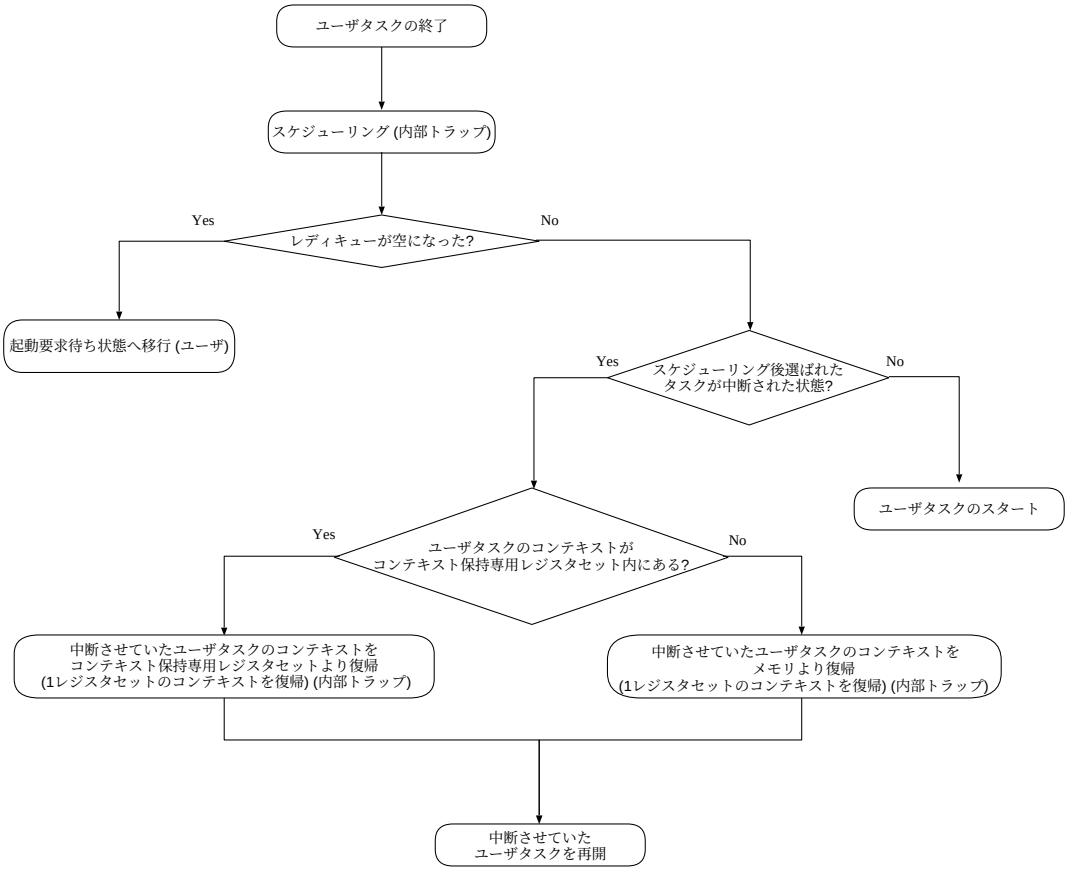

図 6.11: ユーザタスク終了〜スケジューリング〜ユーザタスク実行 or 起動要求待ちまで (レジスタセット間コンテキスト切り替え，絞り込み無し)

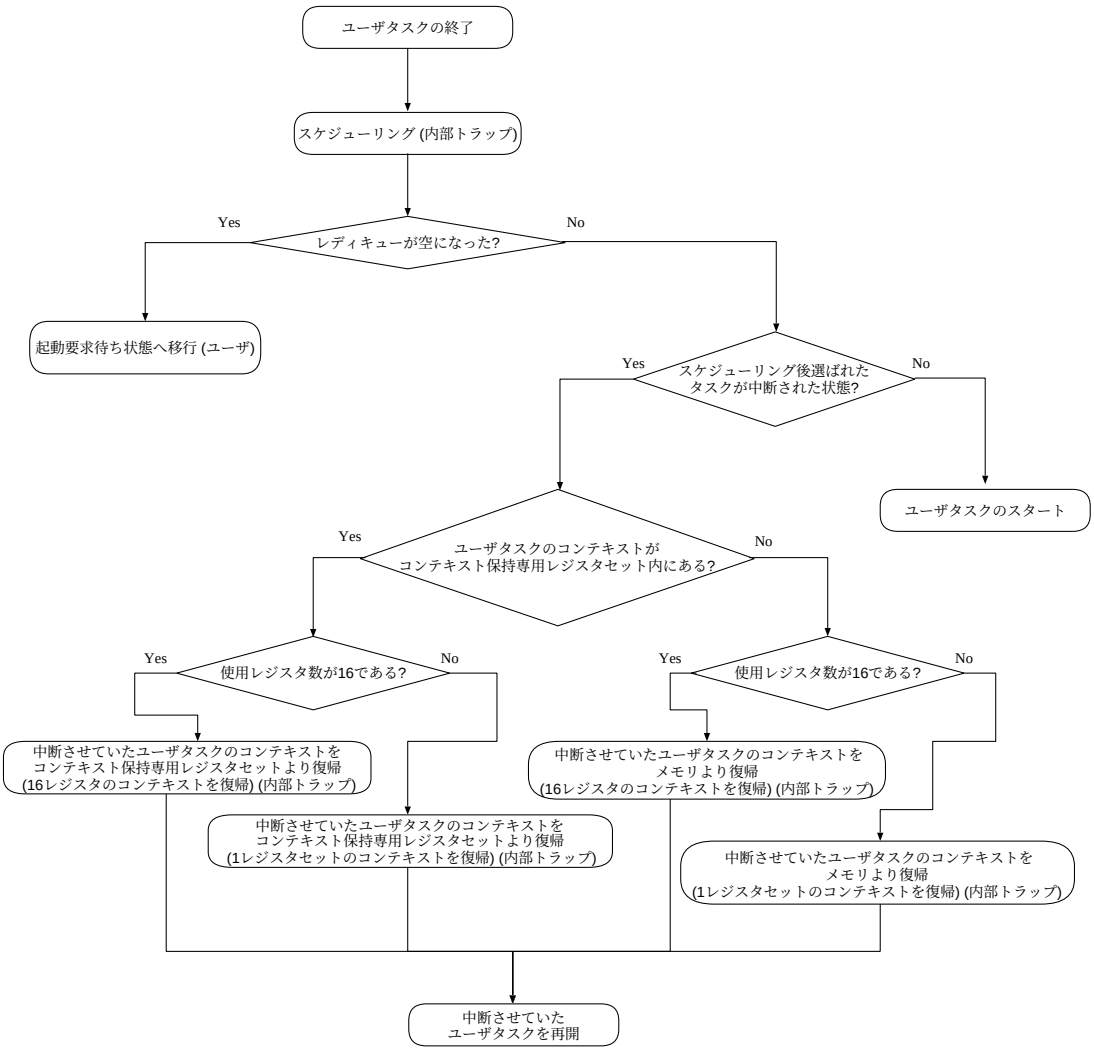

図 6.12: ユーザタスク終了〜スケジューリング〜ユーザタスク実行 or 起動要求待ちまで (レジスタセット間コンテキスト切り替え，絞り込み有り)

6.5 ユーザタスク

ユーザタスクは以下のように用意する.

1. 入力ファイル内に 50 個のユーザタスクを設け、1〜50 の ID を当てる
2. 10 段階の優先度 (1 が最高優先度，10 が最低優先度)
3. その内，周期タスクは優先度 1 〜 5 とし，25 個設ける
4. 非周期タスクは優先度 6 〜 10 とし，25 個設ける
5. タスク毎の優先度の割り当ては,
   ID : 1, 11, 21, 31, 41 = 優先度 : 1,
   ID : 2, 12, 22, 32, 42 = 優先度 : 6,
   ID : 3, 13, 23, 33, 43 = 優先度 : 2,
   ID : 4, 14, 24, 34, 44 = 優先度 : 7,
   ID : 5, 15, 25, 35, 45 = 優先度 : 5,
   ID : 6, 16, 26, 36, 46 = 優先度 : 3,
   ID : 7, 17, 27, 37, 47 = 優先度 : 3,
   ID : 8, 18, 28, 38, 48 = 優先度 : 4,
   ID : 9, 19, 29, 39, 49 = 優先度 : 9,
   ID : 10, 20, 30, 40, 50 = 優先度 : 10
   と定めた.
6. 各ユーザタスクの実行時間 (単位クロックサイクル) は,
   ID : 1 〜 5 = 1000,
   ID : 6 〜 10 = 2500,
   ID : 11 〜 15 = 6500,
   ID : 16 〜 20 = 13000,
   ID : 21 〜 25 = 22000,
   ID : 26 〜 30 = 33500,
   ID : 31 〜 35 = 47500,
   ID : 36 〜 40 = 64000,
   ID : 41 〜 45 = 83000,
   ID : 46 〜 50 = 104500
   を基準として，各タスク生成時に時間調整を行った．時間調整の方法は,
   (a) 5 種類のタスクを C 言語で作成
   (b) 単独でタスクを起動させ実行時間を測定
   (c) ±50 以内の誤差に収まるようにループカウントを増減

(d) 5 種類のタスクを元に実行時間を基準時間に合うように 50 個のタスクを作成

によって実現した

7. 各ユーザの使用レジスタ数は ID : 1, 11, 21, 31, 41 = 使用レジスタ数 : 16,
   ID : 2, 12, 22, 32, 42 = 使用レジスタ数 : 16,
   ID : 3, 13, 23, 33, 43 = 使用レジスタ数 : 32,
   ID : 4, 14, 24, 34, 44 = 使用レジスタ数 : 32,
   ID : 5, 15, 25, 35, 45 = 使用レジスタ数 : 32,
   ID : 6, 16, 26, 36, 46 = 使用レジスタ数 : 16,
   ID : 7, 17, 27, 37, 47 = 使用レジスタ数 : 16,
   ID : 8, 18, 28, 38, 48 = 使用レジスタ数 : 32,
   ID : 9, 19, 29, 39, 49 = 使用レジスタ数 : 32,
   ID : 10, 20, 30, 40, 50 = 使用レジスタ数 : 32
   と定めた．タスク生成時に使用レジスタを確認して，使用レジスタ数が 16，32 となるタスクを生成した．

8. 一度起動要求の入ったユーザタスクはデッドラインオーバーになった場合でも最後まで実行し完了させる

6.6 スケジューラ

スケジューラは μITRON 3.0[10] の優先度別のレディキューを備えたスケジューラとする．レディキューは 10 段階で，優先度が高い順に行っていく．同一優先度内のユーザタスク間では，FCFS(First Come First Service) 方式でスケジューリングを行う．スケジューラがスケジューリングを行う条件は，

- 起動要求発生後の外部トラップ処理で，レディキューにユーザタスクが加わるとき
- ユーザタスクが完了後の内部トラップ処理で，レディキューからユーザタスクが削除されたとき

である．前者では，ユーザタスクが実行されているときにはプリエンプションが発生し，実行していたユーザタスクは実行待ち状態に移行する．図 6.13 で， ID:1 から 8 までを優先度が高い順にレディキューに格納した例を示す．

6.7 メインメモリマップ

組み込み RTOS での評価を行うので，図 6.14 に示す物理メモリに直接レディキュー，タスク情報，フレームテーブル，実行ファイルを書き込む．

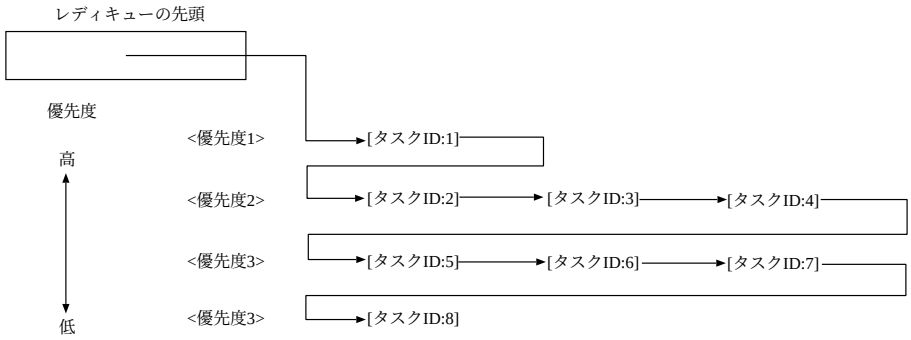

図 6.13: μITRON でのレディキューの例

タスク 1 つ当たりのタスク情報は,「ID」,「優先度」,「ステータス (0 なら実行前，2 なら中断中を示す)」，デッドライン，使用レジスタ数で構成される (図 6.15 参照).

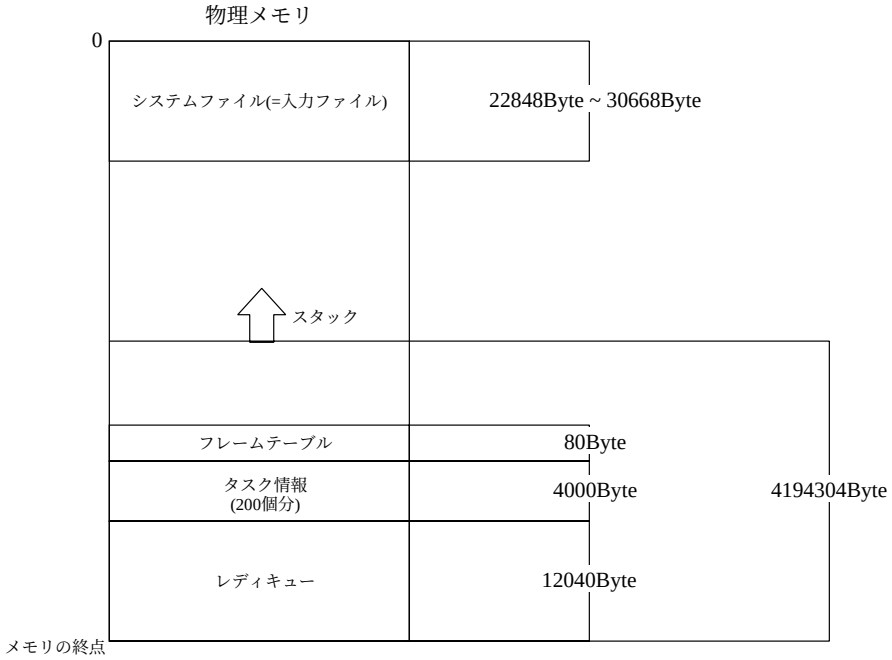

図 6.14: 本シミュレータでのメモリマッピング

6.8 結果

シミュレータ実行前に、実行させるタスクセットを専用のプログラムより決める．それによって，入力ファイル内の 25 個ある周期タスクから 5 個を選抜する．評価は,

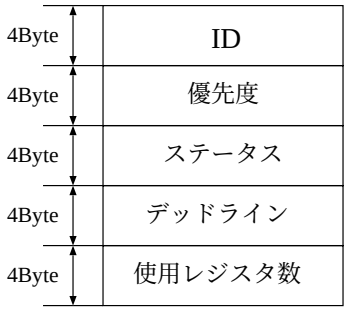

図 6.15: 本シミュレータでのタスク情報 (タスク 1 つ当たり)

- 非周期タスク数が 0，CPU 負荷率が 0.5
- 非周期タスク数が 0，CPU 負荷率が 0.7
- 非周期タスク数が 0，CPU 負荷率が 0.9
- 非周期タスク数が 10，CPU 負荷率が 0.5
- 非周期タスク数が 10，CPU 負荷率が 0.7
- 非周期タスク数が 10，CPU 負荷率が 0.9
- 非周期タスク数が 20，CPU 負荷率が 0.5
- 非周期タスク数が 20，CPU 負荷率が 0.7
- 非周期タスク数が 20，CPU 負荷率が 0.9

の 9 通りからなるタスクセットを用意し，本章の最初で述べた 10 通りの手法で評価を行った．シミュレータの実行クロックサイクルは 10000000 クロックサイクルとした．それぞれ同条件の元で，10 回の異なるタスクセットで実行し，それぞれの実行結果の平均をとった．表 6.1 に周期タスクのみで，CPU 負荷率 $= 0.5$ となるタスクセット実行時に，起動要求が起ったユーザタスク数，起動要求後に実行が完了したユーザタスクタスク数，実行完了したユーザタスクの内デッドラインオーバーしたタスク数について，それぞれ上から順に，シングルレジスタセットで使用レジスタ数の絞り込みが行わないときと行うとき，Casablanca 上で従来のコンテキスト切り替えを行い，使用レジスタ数の絞り込みを行わないときと行うとき，Casablanca 上にコンテキスト保持専用レジスタセットを 1 つ (レジスタセット番号 2) 設け，使用レジスタ数の絞り込みを行わないときと行うとき，Casablanca 上にコンテキスト保持専用レジスタセットを 3 つ (レジスタセット番号 2〜4) 設け，使用レジスタ数の絞り込みを行わないときと行うとき，Casablanca 上にコンテキスト保持専用レジスタセットを 5 つ (レジスタセット番号 2〜6) 設け，使用レジスタ数の

絞り込みを行わないときと行うときの 10 通りについて記載する．続いて，表 6.2 では，コンテキスト切り替えの際にどの手法を用いたかを記載する．左から順に，2 フレーム (32 レジスタ) 分をメモリアクセスによってコンテキスト切り替えを行った回数 (コンテキストを退避させて，その後コンテキストを復帰させるまでを 1 回とする)，1 フレーム (16 レジスタ) 分をメモリアクセスによってコンテキスト切り替えを行った回数，2 フレー ム (32 レジスタ) 分をコンテキスト保持専用レジスタセットとの間でのコンテキスト切り替えを行った回数，1 フレーム (16 レジスタ) 分をコンテキスト保持専用レジスタセットとの間でコンテキスト切り替えを行った回数となる．縦の項目は，表 6.1 と同様である．表 6.3 と 6.4 では 10 回のシミュレーションで用いたタスクセットの平均を記載する．左から順にユーザタスク ID，周期タスクと非周期タスクの分別，タスクの平均負荷と，平均周期，平均デッドライン，ユーザタスクの実行時間，優先度，使用レジスタ数となっており，上から順にタスク ID:1 から ID:50 までについて記載する．周期タスクと非周期タスクの分別は，周期タスクのときには P を，非周期タスクのときには A を書き，それぞれについて判別できるようにした．周期タスクの平均負荷は，1 つのタスクの負荷が，

$$\frac{\text{タスクの実行時間}}{\text{周期}} \cdots\cdots (1)$$

で求められるので，タスクセット 10 個の平均をとり、平均負荷とした．よって，周期タスクの平均負荷は以下の式で求められる．

$$y = \frac{1}{10}\sum_{i=1}^{10} \frac{ID:x \text{ の実行時間}}{\text{入力ファイル } i \text{ での } ID:x \text{ の起動周期}} \cdots\cdots (2)$$

非周期タスクの平均負荷については，非周期タスクが単発起動であることにより，1 つのタスクの負荷が，

$$\frac{\text{タスクの実行時間}}{\text{シミュレータ実行時間}} \cdots\cdots (3)$$

で求められる．よって，非周期タスクの平均負荷は以下の式で求められる．

$$y = \frac{1}{10}\sum_{i=1}^{10} \frac{ID:x \text{ の実行時間}}{10000000} \cdots\cdots (4)$$

平均周期の計算には以下の式を用いた．

$$y = \frac{1}{n}\sum_{i=1}^{n}(ID:x \text{ の起動周期}) \cdots\cdots (5) \ \ (n \text{ は各タスクセットでの使用回数 } (0 \geq n \geq 10))$$

平均デッドラインの計算には以下の式を用いた．

$$y = \frac{1}{n}\sum_{i=1}^{n}(ID:x \text{ の相対デッドライン}) \cdots\cdots (6) \ \ (n \text{ は各タスクセットでの使用回数 } (0 \geq n \geq 10))$$

同様の結果を，CPU 負荷率=0.7 については，表 6.5 から表 6.8 まで，CPU 負荷率=0.9 については，表 6.9 から表 6.12 まで記載する．周期タスクに非周期タスクを 10 個加えた場

合は，CPU 負荷率=0.5 から順に表 6.13 から表 6.24 まで，周期タスクに非周期タスクを 20 個加えた場合は，CPU 負荷率=0.5 から順に表 6.25 から表 6.36 まで記載する．

シミュレーションの結果の内，メモリアクセス回数を図 6.16 に，データキャッシュミス回数を図 6.17 に，命令キャッシュミス回数を図 6.18 に，平均応答時間を図 6.19 に，デッドラインオーバーしたユーザタスク数を図 6.20 に記載する．全てのグラフの X 軸の項目は 9 通りのタスクセットの条件とし，Y 軸の単位は，メモリアクセス回数とデータキャッシュミス回数は回数を，命令キャッシュ平均応答時間はクロックサイクル，デッドラインオーバーしたユーザタスク数はタスク数とする．

表 6.1: 周期タスクのみ，CPU 負荷率 = 0.5 のタスクセット実行時の結果

	起動要求タスク数	実行完了タスク数	Deadline Over
Single RS，絞込無	1244	1232	196
Single RS，絞込有	1244	1232	195
Casablanca, 絞込無	1244	1232	188
Casablanca, 絞込有	1244	1232	187
Context RS = 1, 絞込無	1244	1232	187
Context RS = 1, 絞込有	1244	1232	187
Context RS = 3, 絞込無	1244	1232	187
Context RS = 3, 絞込有	1244	1232	187
Context RS = 5, 絞込無	1244	1232	191
Context RS = 5, 絞込有	1244	1232	190

表 6.1 より，周期タスクのみ，CPU 負荷率 = 0.5 のタスクセットを実行したとき，デッドラインオーバータスク数の起動要求タスク数に対する比率は，最大のとき (シングルレジスタセットで使用レジスタ数の絞り込みを行わないとき，デッドラインオーバータスク数 196) に 15.6%，最小のとき (Casablanca で従来のコンテキスト切り替えを行い，使用レジスタ数の絞り込みを行うとき，Casablanca 上にコンテキスト保持専用レジスタセットを 1 つ設け，使用レジスタ数の絞り込みを行わないときと絞り込み行うとき，Casablanca 上にコンテキスト保持専用レジスタセットを 3 つ設け，使用レジスタ数の絞り込みを行わないときと行うときの計 5 通り，デッドラインオーバータスク数 187) は 15%となった．提案手法により，デッドラインオーバータスク数は最大で 4.6%減少した．ただし，Casablanca 上にコンテキスト保持専用レジスタセットを 5 つ設け，使用レジスタ数の絞り込みを行わないとき (デッドラインオーバータスク数 191)，行うとき (デッドラインオーバータスク数 190) が，Casablanca で従来のコンテキスト切り替えを行い，使用レジスタ数の絞り込みを行わないとき (デッドラインオーバータスク数 188) に比べデッドラインオーバータスク数が多かった．

表 6.2 より，周期タスクのみ，CPU 負荷率 =0.5 のタスクセットを実行したとき，コンテキスト切り替え回数は最大 (シングルレジスタセットで使用レジスタ数の絞り込みを行わないとき) で 506 回，最小 (Casablanca 上にコンテキスト保持専用レジスタセットを 1 つ設け，使用レジスタ数の絞り込みを行わないときと行うとき，Casablanca 上にコンテキスト保持専用レジスタセットを 3 つ設け，使用レジスタ数の絞り込みを行わないときの計 3 通り) で 215 回となった．表 6.1 のデッドラインオーバータスク数とコンテキスト切り替え回数より，デッドラインオーバータスク数が大きく減少 (例えば 196 から 187 へ減少) するとき，コンテキスト切り替えの回数が減少する傾向があった．

表 6.2: 周期タスクのみ，CPU 負荷率= 0.5 のタスクセット実行時のコンテキスト切り替え手法回数

	2Frames Mem.	1Frame Mem.	2Frames RS	1Frame RS
Single RS，絞込無	506	0	0	0
Single RS，絞込有	363	142	0	0
Casablanca, 絞込無	218	0	0	0
Casablanca, 絞込有	199	20	0	0
Context RS = 1, 絞込無	17	0	198	0
Context RS = 1, 絞込有	11	6	184	14
Context RS = 3, 絞込無	0	0	215	0
Context RS = 3, 絞込有	0	0	197	20
Context RS = 5, 絞込無	0	0	218	0
Context RS = 5, 絞込有	0	0	200	21

表 6.3: 周期タスクのみの実行で，CPU 負荷率=0.5 での各タスクの平均分布 (1)

ID	P/A	平均負荷	平均周期	平均デッドライン	実行時間	優先度	使用レジスタ数
1	P	0.017513	5733	1616	1004	1	16
2	-	-	-	-	1002	6	16
3	P	0.038265	8526	2111	1022	2	32
4	-	-	-	-	1007	7	32
5	P	0.026935	12214	6023	1005	5	32
6	P	0.024577	22306	8125	2518	3	16
7	-	-	-	-	2490	8	16
8	P	0.029356	17794	14740	2486	4	32
9	-	-	-	-	2519	9	32
10	-	-	-	-	2485	10	32
11	-	-	-	-	6486	1	16
12	-	-	-	-	6498	6	16
13	P	0.028178	127203	10529	6504	2	32
14	-	-	-	-	6515	7	32
15	P	0.059406	62064	41381	6505	5	32
16	P	0.00432	300925	76440	12999	3	16
17	-	-	-	-	13002	8	16
18	P	0.026482	99569	76180	13001	4	32
19	-	-	-	-	13019	9	32
20	-	-	-	-	13009	10	32
21	P	0.004478	491071	40128	21990	1	16
22	-	-	-	-	22000	6	16
23	-	-	-	-	22009	2	32
24	-	-	-	-	22027	7	32
25	P	0.008235	266990	134992	21987	5	32

表 6.4: 周期タスクのみの実行で，CPU 負荷率=0.5 での各タスクの平均分布 (2)

ID	P/A	平均負荷	平均周期	平均デッドライン	実行時間	優先度	使用レジスタ数
26	P	0.029289	232829	149075	33510	3	16
27	-	-	-	-	33500	8	16
28	P	0.01128	597608	195371	33499	4	32
29	-	-	-	-	33507	9	32
30	-	-	-	-	33513	10	32
31	P	0.009202	1042143	128630	47511	1	16
32	-	-	-	-	47500	6	16
33	P	0.025592	373192	117040	47485	2	32
34	-	-	-	-	47505	7	32
35	P	0.019994	1198126	261186	47485	5	32
36	P	0.027204	1267968	244799	64009	3	16
37	-	-	-	-	64000	8	16
38	P	0.00248	2580645	485888	63997	4	32
39	-	-	-	-	63995	9	32
40	-	-	-	-	64045	10	32
41	P	0.015834	523989	177288	82969	1	16
42	-	-	-	-	83000	6	16
43	P	0.024156	692216	175959	82986	2	32
44	-	-	-	-	82979	7	32
45	P	0.039384	947891	603907	83050	5	32
46	P	0.016481	1456100	317679	104506	3	16
47	-	-	-	-	104500	8	16
48	P	0.007282	1435439	616132	104526	4	32
49	-	-	-	-	104537	9	32
50	-	-	-	-	104544	10	32

表 6.5: 周期タスクのみ，CPU 負荷率= 0.7 のタスクセット実行時の結果

	起動要求タスク数	実行完了タスク数	Deadline Over
Single RS，絞込無	842	840	269
Single RS，絞込有	842	840	267
Casablanca, 絞込無	842	840	258
Casablanca, 絞込有	842	840	257
Context RS = 1, 絞込無	842	840	256
Context RS = 1, 絞込有	842	840	256
Context RS = 3, 絞込無	842	840	256
Context RS = 3, 絞込有	842	840	256
Context RS = 5, 絞込無	842	840	258
Context RS = 5, 絞込有	842	840	256

表 6.5 より，周期タスクのみ，CPU 負荷率 =0.7 のタスクセットを実行したとき，デッドラインオーバータスク数の起動要求タスク数に対する比率は，最大のとき (シングルレジスタセットで使用レジスタ数の絞り込みを行わないとき，デッドラインオーバータスク数 269) に 31.9%，最小のとき (Casablanca 上にコンテキスト保持専用レジスタセットを 1 つ設け，使用レジスタ数の絞り込みを行わないときと絞り込み行うとき，Casablanca 上にコンテキスト保持専用レジスタセットを 3 つ設け，使用レジスタ数の絞り込みを行わないときと行うとき，Casablanca 上にコンテキスト保持専用レジスタセットを 5 つ設け，使用レジスタ数の絞り込みを行うときの計 5 通り，デッドラインオーバータスク数 256) は 30.4%となった．提案手法により，デッドラインオーバータスク数は最大で 4.8%減少した．

表 6.6 より，周期タスクのみ，CPU 負荷率 =0.7 のタスクセットを実行したとき，コンテキスト切り替え回数は最大 (シングルレジスタセットで使用レジスタ数の絞り込みを行わないとき) で 537 回，最小 (Casablanca 上にコンテキスト保持専用レジスタセットを 1 つ設け，使用レジスタ数の絞り込みを行わないときと行うとき，Casablanca 上にコンテキスト保持専用レジスタセットを 3 つ設け，使用レジスタ数の絞り込みを行わないときと絞り込み行うとき，Casablanca 上にコンテキスト保持専用レジスタセットを 5 つ設け，使用レジスタ数の絞り込みを行わないときと絞り込み行うときの計 6 通り) で 215 回となった．表 6.5 のデッドラインオーバータスク数とコンテキスト切り替え回数より，デッドラインオーバータスク数が大きく減少 (例えば 269 から 256 へ減少) するとき，コンテキスト切り替えの回数が減少する傾向があった．

表 6.6: 周期タスクのみ，CPU 負荷率= 0.7 のタスクセット実行時のコンテキスト切り替え手法回数

	2Frames Mem.	1Frame Mem.	2Frames RS	1Frame RS
Single RS，絞込無	537	0	0	0
Single RS，絞込有	293	242	0	0
Casablanca, 絞込無	205	0	0	0
Casablanca, 絞込有	140	65	0	0
Context RS = 1, 絞込無	30	0	174	0
Context RS = 1, 絞込有	17	13	121	53
Context RS = 3, 絞込無	0	0	204	0
Context RS = 3, 絞込有	0	0	139	65
Context RS = 5, 絞込無	0	0	204	0
Context RS = 5, 絞込有	0	0	139	65

表 6.7: 周期タスクのみの実行で，CPU 負荷率=0.7 での各タスクの平均分布 (1)

ID	P/A	平均負荷	平均周期	平均デッドライン	実行時間	優先度	使用レジスタ数
1	P	0.012707	7901	1488	1004	1	16
2	-	-	-	-	1002	6	16
3	-	-	-	-	1022	2	32
4	-	-	-	-	1007	7	32
5	P	0.016323	6157	4071	1005	5	32
6	P	0.021886	11505	12250	2518	3	16
7	-	-	-	-	2490	8	16
8	P	0.043214	26268	13040	2486	4	32
9	-	-	-	-	2519	9	32
10	-	-	-	-	2485	10	32
11	P	0.037999	34148	8267	6486	1	16
12	-	-	-	-	6498	6	16
13	P	0.038104	34157	17302	6504	2	32
14	-	-	-	-	6515	7	32
15	P	0.045732	50581	44252	6505	5	32
16	P	0.018367	153637	34580	12999	3	16
17	-	-	-	-	13002	8	16
18	-	-	-	-	13001	4	32
19	-	-	-	-	13019	9	32
20	-	-	-	-	13009	10	32
21	P	0.009628	520099	33704	21990	1	16
22	-	-	-	-	22000	6	16
23	-	-	-	-	22009	2	32
24	-	-	-	-	22027	7	32
25	P	0.048915	149309	208560	21987	5	32

表 6.8: 周期タスクのみの実行で，CPU 負荷率=0.7 での各タスクの平均分布 (2)

ID	P/A	平均負荷	平均周期	平均デッドライン	実行時間	優先度	使用レジスタ数
26	P	0.009187	364764	117250	33510	3	16
27	-	-	-	-	33500	8	16
28	P	0.028559	564979	89779	33499	4	32
29	-	-	-	-	33507	9	32
30	-	-	-	-	33513	10	32
31	P	0.020837	559539	117040	47511	1	16
32	-	-	-	-	47500	6	16
33	P	0.062812	404249	124336	47485	2	32
34	-	-	-	-	47505	7	32
35	P	0.004367	1087454	305140	47485	5	32
36	P	0.084012	413646	285184	64009	3	16
37	-	-	-	-	64000	8	16
38	P	0.022623	907885	236031	63997	4	32
39	-	-	-	-	63995	9	32
40	-	-	-	-	64045	10	32
41	P	0.067399	496965	141266	82969	1	16
42	-	-	-	-	83000	6	16
43	P	0.01467	565703	207500	82986	2	32
44	-	-	-	-	82979	7	32
45	P	0.053792	545712	517255	83050	5	32
46	P	0.033714	1034991	418696	104506	3	16
47	-	-	-	-	104500	8	16
48	-	-	-	-	104526	4	32
49	-	-	-	-	104537	9	32
50	-	-	-	-	104544	10	32

表 6.9: 周期タスクのみ，CPU 負荷率= 0.9 のタスクセット実行時の結果

	起動要求タスク数	実行完了タスク数	Deadline Over
Single RS，絞込無	1230	1224	369
Single RS，絞込有	1230	1224	367
Casablanca, 絞込無	1230	1225	347
Casablanca, 絞込有	1230	1225	349
Context RS = 1, 絞込無	1230	1225	348
Context RS = 1, 絞込有	1230	1225	345
Context RS = 3, 絞込無	1230	1225	344
Context RS = 3, 絞込有	1230	1225	340
Context RS = 5, 絞込無	1230	1225	351
Context RS = 5, 絞込有	1230	1225	352

表 6.9 より，周期タスクのみ，CPU 負荷率 =0.9 のタスクセットを実行したとき，デッドラインオーバータスク数の起動要求タスク数に対する比率は，最大のとき (シングルレジスタセットで使用レジスタ数の絞り込みを行うとき，デッドラインオーバータスク数 369) に 30%，最小のとき (Casablanca 上にコンテキスト保持専用レジスタセットを 3 つ設け，使用レジスタ数の絞り込みを行うとき，デッドラインオーバータスク数 340) は 27.6%となった．提案手法により，デッドラインオーバータスク数は最大で 7.9%減少した．ただし，従来のコンテキスト切り替えを行い，使用レジスタ数の絞り込みを行うとき (デッドラインオーバータスク数 349)，Casablanca 上にコンテキスト保持専用レジスタセットを 1 つ設け，使用レジスタ数の絞り込みを行わないとき (デッドラインオーバータスク数 348)，Casablanca 上にコンテキスト保持専用レジスタセットを 5 つ設け，使用レジスタ数の絞り込みを行わないとき (デッドラインオーバータスク数 351)，行うとき (デッドラインオーバータスク数 352)，Casablanca で従来のコンテキスト切り替えを行い，使用レジスタ数の絞り込みを行わないとき (デッドラインオーバータスク数 347) に比べデッドラインオーバータスク数が多かった．

表 6.10 より，周期タスクのみ，CPU 負荷率 =0.9 のタスクセットを実行したとき，コンテキスト切り替え回数は最大 (シングルレジスタセットで使用レジスタ数の絞り込みを行うとき) で 1104 回，最小 (Casablanca 上にコンテキスト保持専用レジスタセットを 3 つ設け，使用レジスタ数の絞り込みを行わないとき) で 613 回となった．表 6.9 のデッドラインオーバータスク数とコンテキスト切り替え回数より，デッドラインオーバータスク数が大きく減少 (例えば 369 から 340 へ減少) するとき，コンテキスト切り替えの回数が減少する傾向があった．

表 6.10: 周期タスクのみ，CPU 負荷率= 0.9 のタスクセット実行時のコンテキスト切り替え手法回数

	2Frames Mem.	1Frame Mem.	2Frames RS	1Frame RS
Single RS，絞込無	1103	0	0	0
Single RS，絞込有	734	370	0	0
Casablanca, 絞込無	624	0	0	0
Casablanca, 絞込有	410	218	0	0
Context RS = 1, 絞込無	165	0	456	0
Context RS = 1, 絞込有	71	91	329	127
Context RS = 3, 絞込無	0	0	613	0
Context RS = 3, 絞込有	1	0	397	217
Context RS = 5, 絞込無	0	0	634	0
Context RS = 5, 絞込有	0	0	414	227

表 6.11: 周期タスクのみの実行で，CPU 負荷率=0.9 での各タスクの平均分布 (1)

ID	P/A	平均負荷	平均周期	平均デッドライン	実行時間	優先度	使用レジスタ数
1	P	0.012868	18865	1892	1004	1	16
2	-	-	-	-	1002	6	16
3	P	0.050629	6697	2612	1022	2	32
4	-	-	-	-	1007	7	32
5	-	-	-	-	1005	5	32
6	P	0.03336	15669	8300	2518	3	16
7	-	-	-	-	2490	8	16
8	P	0.034941	17540	11059	2486	4	32
9	-	-	-	-	2519	9	32
10	-	-	-	-	2485	10	32
11	P	0.031612	44828	17238	6486	1	16
12	-	-	-	-	6498	6	16
13	P	0.024207	26868	10868	6504	2	32
14	-	-	-	-	6515	7	32
15	P	0.028534	72332	31771	6505	5	32
16	P	0.046941	61660	26130	12999	3	16
17	-	-	-	-	13002	8	16
18	P	0.057028	76741	65970	13001	4	32
19	-	-	-	-	13019	9	32
20	-	-	-	-	13009	10	32
21	P	0.06693	105244	55968	21990	1	16
22	-	-	-	-	22000	6	16
23	P	0.025355	86805	49456	22009	2	32
24	-	-	-	-	22027	7	32
25	P	0.010794	203703	187792	21987	5	32

表 6.12: 周期タスクのみの実行で，CPU 負荷率=0.9 での各タスクの平均分布 (2)

ID	P/A	平均負荷	平均周期	平均デッドライン	実行時間	優先度	使用レジスタ数
26	P	0.061219	109579	129980	33510	3	16
27	-	-	-	-	33500	8	16
28	-	-	-	-	33499	4	32
29	-	-	-	-	33507	9	32
30	-	-	-	-	33513	10	32
31	P	0.006482	733024	103360	47511	1	16
32	-	-	-	-	47500	6	16
33	P	0.053119	595504	111624	47485	2	32
34	-	-	-	-	47505	7	32
35	P	0.043906	221563	443079	47485	5	32
36	P	0.055736	352233	238080	64009	3	16
37	-	-	-	-	64000	8	16
38	P	0.010943	584795	320000	63997	4	32
39	-	-	-	-	63995	9	32
40	-	-	-	-	64045	10	32
41	P	0.053836	468619	224653	82969	1	16
42	-	-	-	-	83000	6	16
43	P	0.072132	353925	158363	82986	2	32
44	-	-	-	-	82979	7	32
45	P	0.069306	813898	655699	83050	5	32
46	P	0.021169	493669	545490	104506	3	16
47	-	-	-	-	104500	8	16
48	P	0.025062	417065	830148	104526	4	32
49	-	-	-	-	104537	9	32
50	-	-	-	-	104544	10	32

表 6.13: 周期タスク+10 個の非周期タスク，CPU 負荷率= 0.5 のタスクセット実行時の結果

	起動要求タスク数	実行完了タスク数	Deadline Over
Single RS，絞込無	878	876	134
Single RS，絞込有	878	876	130
Casablanca, 絞込無	878	876	126
Casablanca, 絞込有	878	876	124
Context RS = 1, 絞込無	878	876	124
Context RS = 1, 絞込有	878	876	124
Context RS = 3, 絞込無	878	876	121
Context RS = 3, 絞込有	878	876	122
Context RS = 5, 絞込無	878	876	122
Context RS = 5, 絞込有	878	876	123

表 6.13 より，周期タスク+10 個の非周期タスク，CPU 負荷率 =0.5 のタスクセットを実行したとき，デッドラインオーバータスク数の起動要求タスク数に対する比率は，最大のとき (シングルレジスタセットで使用レジスタ数の絞り込みを行わないとき，デッドラインオーバータスク数 134) に 15.3%，最小のとき (Casablanca 上にコンテキスト保持専用レジスタセットを 3 つ設け，使用レジスタ数の絞り込みを行わないとき，デッドラインオーバータスク数 121) は 13.7%となった．提案手法により，デッドラインオーバータスク数は最大で 9.7%減少した．

表 6.14 より，周期タスク+10 個の非周期タスク，CPU 負荷率 =0.5 のタスクセットを実行したとき，コンテキスト切り替え回数は最大 (シングルレジスタセットで使用レジスタ数の絞り込みを行わないとき) で 362 回，最小 (Casablanca 上にコンテキスト保持専用レジスタセットを 3 つ設け，使用レジスタ数の絞り込みを行わないときと絞り込み行うときの計 2 通り) で 186 回となった．表 6.13 のデッドラインオーバータスク数とコンテキスト切り替え回数より，デッドラインオーバータスク数が大きく減少 (例えば 134 から 121 へ減少) するとき，コンテキスト切り替えの回数が減少する傾向があった．

表 6.14: 周期タスク+10 個の非周期タスク，CPU 負荷率= 0.5 のタスクセット実行時のコンテキスト切り替え手法の回数

	2Frames Mem.	1Frame Mem.	2Frames RS	1Frame RS
Single RS，絞込無	362	0	0	0
Single RS，絞込有	160	201	0	0
Casablanca, 絞込無	188	0	0	0
Casablanca, 絞込有	119	68	0	0
Context RS = 1, 絞込無	25	0	162	0
Context RS = 1, 絞込有	13	9	106	60
Context RS = 3, 絞込無	0	0	186	0
Context RS = 3, 絞込有	0	0	118	68
Context RS = 5, 絞込無	0	0	187	0
Context RS = 5, 絞込有	0	0	119	69

表 6.15: 周期タスク+10 個の非周期タスクの実行で，CPU 負荷率=0.5 での各タスクの平均分布 (1)

ID	P/A	平均負荷	平均周期	平均デッドライン	実行時間	優先度	使用レジスタ数
1	P	0.046929	9157	1710	1004	1	16
2	A	0.00005	-	2249	1002	6	16
3	-	-	-	-	1022	2	32
4	A	0.00005	-	2305	1007	7	32
5	P	0.008054	12479	7480	1005	5	32
6	P	0.008058	31247	9150	2518	3	16
7	A	0.000149	-	7360	2490	8	16
8	P	0.021481	67629	14633	2486	4	32
9	A	0.000076	-	9400	2519	9	32
10	A	0.000124	-	11224	2485	10	32
11	P	0.029761	103969	12393	6486	1	16
12	A	0.00026	-	12519	6498	6	16
13	P	0.040255	143629	16140	6504	2	32
14	A	0.000326	-	16093	6515	7	32
15	P	0.013571	47933	73892	6505	5	32
16	P	0.014843	361451	50180	12999	3	16
17	A	0.00052	-	36659	13002	8	16
18	P	0.005361	242527	48360	13001	4	32
19	A	0.000391	-	43246	13019	9	32
20	-	-	-	-	13009	10	32
21	P	0.040674	366404	43340	21990	1	16
22	A	0.0011	-	51638	22000	6	16
23	P	0.013595	161892	71896	22009	2	32
24	A	0.000661	-	49544	22027	7	32
25	P	0.003041	723014	61072	21987	5	32

表 6.16: 周期タスク+10 個の非周期タスクの実行で，CPU 負荷率=0.5 での各タスクの平均分布 (2)

ID	P/A	平均負荷	平均周期	平均デッドライン	実行時間	優先度	使用レジスタ数
26	P	0.028801	405994	146730	33510	3	16
27	A	0.001675	-	88815	33500	8	16
28	P	0.014736	227331	104251	33499	4	32
29	A	0.00134	-	114402	33507	9	32
30	A	0.001005	-	109880	33513	10	32
31	P	0.017958	1008633	120270	47511	1	16
32	A	0.00095	-	115140	47500	6	16
33	P	0.006751	703411	175180	47485	2	32
34	A	0.001425	-	117229	47505	7	32
35	-	-	-	-	47485	5	32
36	P	0.009816	1931450	220160	64009	3	16
37	A	0.00256	-	173568	64000	8	16
38	P	0.008519	1592534	204288	63997	4	32
39	A	0.0032	-	166400	63995	9	32
40	A	0.001921	-	224767	64045	10	32
41	P	0.024109	688561	157700	82969	1	16
42	A	0.00498	-	149178	83000	6	16
43	P	0.025146	1320274	234723	82986	2	32
44	A	0.00166	-	181106	82979	7	32
45	P	0.004407	1884650	899720	83050	5	32
46	P	0.049411	895990	398667	104506	3	16
47	A	0.007315	-	292241	104500	8	16
48	P	0.018722	1116696	468995	104526	4	32
49	A	0.005227	-	382888	104537	9	32
50	A	0.003136	-	373692	104544	10	32

表 6.17: 周期タスク+10 個の非周期タスク，CPU 負荷率= 0.7 のタスクセット実行時の結果

	起動要求タスク数	平均実行完了タスク数	Deadline Over
Single RS，絞込無	1078	1077	141
Single RS，絞込有	1078	1077	137
Casablanca, 絞込無	1078	1077	128
Casablanca, 絞込有	1078	1077	127
Context RS = 1, 絞込無	1078	1077	126
Context RS = 1, 絞込有	1078	1077	126
Context RS = 3, 絞込無	1078	1077	123
Context RS = 3, 絞込有	1078	1077	124
Context RS = 5, 絞込無	1078	1077	127
Context RS = 5, 絞込有	1078	1077	128

表 6.17 より，周期タスク+10 個の非周期タスク，CPU 負荷率 =0.7 のタスクセットを実行したとき，デッドラインオーバータスク数の起動要求タスク数に対する比率は，最大のとき (シングルレジスタセットで使用レジスタ数の絞り込みを行わないとき，デッドラインオーバータスク数 141) に 13.1%，最小のとき (Casablanca 上にコンテキスト保持専用レジスタセットを 3 つ設け，使用レジスタ数の絞り込みを行わないとき，デッドラインオーバータスク数 123) は 11.4%となった．提案手法により，デッドラインオーバータスク数は最大で 9%減少した．

表 6.18 より，周期タスク+10 個の非周期タスク，CPU 負荷率 =0.7 のタスクセットを実行したとき，コンテキスト切り替え回数は最大 (シングルレジスタセットで使用レジスタ数の絞り込みを行わないとき) で 664 回，最小 (Casablanca 上にコンテキスト保持専用レジスタセットを 3 つ設け，使用レジスタ数の絞り込みを行わないとき) で 438 回となった．表 6.17 のデッドラインオーバータスク数とコンテキスト切り替え回数より，デッドラインオーバータスク数が大きく減少 (例えば 141 から 123 へ減少) するとき，コンテキスト切り替えの回数が減少する傾向があった．

表 6.18: 周期タスク+10 個の非周期タスク，CPU 負荷率= 0.7 のタスクセット実行時のコンテキスト切り替え手法の回数

	2Frames Mem.	1Frame Mem.	2Frames RS	1Frame RS
Single RS，絞込無	664	0	0	0
Single RS，絞込有	384	276	0	0
Casablanca, 絞込無	444	0	0	0
Casablanca, 絞込有	290	152	0	0
Context RS = 1, 絞込無	97	0	343	0
Context RS = 1, 絞込有	54	37	235	115
Context RS = 3, 絞込無	0	0	438	0
Context RS = 3, 絞込有	0	0	287	152
Context RS = 5, 絞込無	0	0	446	0
Context RS = 5, 絞込有	0	0	289	157

表 6.19: 周期タスク+10 個の非周期タスクの実行で，CPU 負荷率=0.7 での各タスクの平均分布 (1)

ID	P/A	平均負荷	平均周期	平均デッドライン	実行時間	優先度	使用レジスタ数
1	P	0.035226	13256	1918	1004	1	16
2	A	0.00006	-	2336	1002	6	16
3	P	0.020202	5059	2992	1022	2	32
4	A	0.00007	-	2666	1007	7	32
5	-	-	-	-	1005	5	32
6	P	0.038964	13058	9850	2518	3	16
7	A	0.000124	-	8892	2490	8	16
8	P	0.026057	22068	11899	2486	4	32
9	A	0.000176	-	8542	2519	9	32
10	A	0.000149	-	8129	2485	10	32
11	P	0.035598	39026	11440	6486	1	16
12	A	0.000195	-	12930	6498	6	16
13	P	0.031167	42640	10204	6504	2	32
14	A	0.000261	-	16776	6515	7	32
15	P	0.015267	42608	21787	6505	5	32
16	P	0.059688	137000	47905	12999	3	16
17	A	0.00065	-	36836	13002	8	16
18	-	-	-	-	13001	4	32
19	A	0.000521	-	34125	13019	9	32
20	A	0.00013	-	33591	13009	10	32
21	P	0.020443	107569	57552	21990	1	16
22	A	0.00088	-	48884	22000	6	16
23	P	0.027893	180140	31899	22009	2	32
24	A	0.000661	-	64064	22027	7	32
25	P	0.021068	104363	228976	21987	5	32

表 6.20: 周期タスク+10 個の非周期タスクの実行で，CPU 負荷率=0.7 での各タスクの平均分布 (2)

ID	P/A	平均負荷	平均周期	平均デッドライン	実行時間	優先度	使用レジスタ数
26	P	0.044578	282333	117920	33510	3	16
27	A	0.001005	-	75844	33500	8	16
28	P	0.035768	462731	160710	33499	4	32
29	A	0.001675	-	118188	33507	9	32
30	A	0.001676	-	128211	33513	10	32
31	P	0.009949	1021081	94429	47511	1	16
32	-	-	-	-	47500	6	16
33	P	0.028041	371054	107065	47485	2	32
34	A	0.001425	-	136420	47505	7	32
35	P	0.01537	308955	175180	47485	5	32
36	P	0.013672	468182	305920	64009	3	16
37	A	0.00256	-	151808	64000	8	16
38	P	0.031366	717418	345258	63997	4	32
39	A	0.00192	-	164693	63995	9	32
40	A	0.001281	-	232960	64045	10	32
41	P	0.042772	913262	195216	82969	1	16
42	A	0.00249	-	162458	83000	6	16
43	P	0.012357	1399789	187082	82986	2	32
44	A	0.003319	-	156454	82979	7	32
45	P	0.032148	911632	757624	83050	5	32
46	P	0.028867	1076671	372020	104506	3	16
47	A	0.007315	-	278865	104500	8	16
48	P	0.033559	624306	744875	104526	4	32
49	A	0.004181	-	326562	104537	9	32
50	A	0.002091	-	479446	104544	10	32

表 6.21: 周期タスク+10 個の非周期タスク，CPU 負荷率= 0.9 のタスクセット実行時の結果

	起動要求タスク数	実行完了タスク数	Deadline Over
Single RS，絞込無	1267	1264	454
Single RS，絞込有	1267	1264	438
Casablanca, 絞込無	1267	1264	394
Casablanca, 絞込有	1267	1264	394
Context RS = 1, 絞込無	1267	1264	402
Context RS = 1, 絞込有	1267	1264	395
Context RS = 3, 絞込無	1267	1264	387
Context RS = 3, 絞込有	1267	1265	373
Context RS = 5, 絞込無	1267	1264	398
Context RS = 5, 絞込有	1267	1264	391

表 6.21 より，周期タスク+10 個の非周期タスク，CPU 負荷率 =0.9 のタスクセットを実行したとき，デッドラインオーバータスク数の起動要求タスク数に対する比率は，最大のとき (シングルレジスタセットで使用レジスタ数の絞り込みを行わないとき，デッドラインオーバータスク数 454) に 35.8%，最小のとき (Casablanca 上にコンテキスト保持専用レジスタセットを 3 つ設け，使用レジスタ数の絞り込みを行わないとき，デッドラインオーバータスク数 373) は 29.4%となった．提案手法により，デッドラインオーバータスク数は最大で 17.8%減少した．ただし，Casablanca 上にコンテキスト保持専用レジスタセットを 1 つ設け，使用レジスタ数の絞り込みを行わないとき (デッドラインオーバータスク数 402)，行うとき (デッドラインオーバータスク数 395)，Casablanca 上にコンテキスト保持専用レジスタセットを 5 つ設け，使用レジスタ数の絞り込みを行わないとき (デッドラインオーバータスク数 398)，Casablanca で従来のコンテキスト切り替えを行い，使用レジスタ数の絞り込みを行わないとき (デッドラインオーバータスク数 394) に比べデッドラインオーバータスク数が多かった．

表 6.22 より，周期タスク+10 個の非周期タスク，CPU 負荷率 =0.9 のタスクセットを実行したとき，コンテキスト切り替え回数は最大 (シングルレジスタセットで使用レジスタ数の絞り込みを行わないとき) で 1130 回，最小 (Casablanca 上にコンテキスト保持専用レジスタセットを 3 つ設け，使用レジスタ数の絞り込みを行うとき) で 554 回となった．表 6.21 のデッドラインオーバータスク数とコンテキスト切り替え回数より，デッドラインオーバータスク数が大きく減少 (例えば 454 から 373 へ減少) するとき，コンテキスト切り替えの回数が減少する傾向があった．

表 6.22: 周期タスク+10 個の非周期タスク，CPU 負荷率= 0.9 のタスクセット実行時のコンテキスト切り替え手法の回数

	2Frames Mem.	1Frame Mem.	2Frames RS	1Frame RS
Single RS，絞込無	1130	0	0	0
Single RS，絞込有	773	350	0	0
Casablanca, 絞込無	571	0	0	0
Casablanca, 絞込有	488	88	0	0
Context RS = 1, 絞込無	272	0	303	0
Context RS = 1, 絞込有	220	24	254	64
Context RS = 3, 絞込無	2	0	555	0
Context RS = 3, 絞込有	1	0	467	87
Context RS = 5, 絞込無	0	0	571	0
Context RS = 5, 絞込有	0	0	487	91

表 6.23: 周期タスク+10 個の非周期タスクの実行で，CPU 負荷率=0.9 での各タスクの平均分布 (1)

ID	P/A	平均負荷	平均周期	平均デッドライン	実行時間	優先度	使用レジスタ数
1	P	0.022829	4398	1264	1004	1	16
2	A	0.00002	-	1723	1002	6	16
3	P	0.027116	3769	1503	1022	2	32
4	A	0.00004	-	2457	1007	7	32
5	P	0.01129	8902	11320	1005	5	32
6	P	0.028198	24878	9925	2518	3	16
7	A	0.000124	-	6444	2490	8	16
8	P	0.037386	14275	5419	2486	4	32
9	A	0.00005	-	10225	2519	9	32
10	A	0.000124	-	10443	2485	10	32
11	P	0.015281	42446	13832	6486	1	16
12	A	0.000195	-	10798	6498	6	16
13	P	0.042256	34302	11687	6504	2	32
14	A	0.000261	-	13675	6515	7	32
15	P	0.04812	28530	68900	6505	5	32
16	P	0.127103	51871	38583	12999	3	16
17	A	0.00091	-	36058	13002	8	16
18	P	0.006499	200048	75816	13001	4	32
19	A	0.00026	-	37440	13019	9	32
20	A	0.00039	-	43680	13009	10	32
21	P	0.066004	245829	39212	21990	1	16
22	A	0.0011	-	37839	22000	6	16
23	P	0.057154	172360	53349	22009	2	32
24	A	0.001322	-	54736	22027	7	32
25	P	0.018311	120077	176175	21987	5	32

表 6.24: 周期タスク+10 個の非周期タスクの実行で，CPU 負荷率=0.9 での各タスクの平均分布 (2)

ID	P/A	平均負荷	平均周期	平均デッドライン	実行時間	優先度	使用レジスタ数
26	P	0.026329	127275	105190	33510	3	16
27	A	0.00134	-	97685	33500	8	16
28	P	0.041766	281733	193585	33499	4	32
29	A	0.00067	-	81404	33507	9	32
30	A	0.001341	-	166964	33513	10	32
31	P	0.01696	280136	91960	47511	1	16
32	A	0.001425	-	105513	47500	6	16
33	P	0.022454	664845	94809	47485	2	32
34	A	0.001425	-	117989	47505	7	32
35	P	0.061282	255964	376579	47485	5	32
36	P	0.02525	566064	246400	64009	3	16
37	A	0.0032	-	182169	64000	8	16
38	P	0.018612	899946	312832	63997	4	32
39	A	0.00448	-	227474	63995	9	32
40	A	0.001921	-	224256	64045	10	32
41	-	-	-	-	82969	1	16
42	A	0.00249	-	167992	83000	6	16
43	-	-	-	-	82986	2	32
44	A	0.003319	-	267259	82979	7	32
45	P	0.052005	492650	430935	83050	5	32
46	P	0.019592	1112871	438900	104506	3	16
47	A	0.005225	-	347943	104500	8	16
48	P	0.06231	513930	533646	104526	4	32
49	A	0.005227	-	466488	104537	9	32
50	A	0.004182	-	450603	104544	10	32

表 6.25: 周期タスク+20 個の非周期タスク，CPU 負荷率= 0.5 のタスクセット実行時の結果

	起動要求タスク数	実行完了タスク数	Deadline Over
Single RS，絞込無	569	569	76
Single RS，絞込有	569	569	77
Casablanca, 絞込無	569	569	73
Casablanca, 絞込有	569	569	73
Context RS = 1, 絞込無	569	569	73
Context RS = 1, 絞込有	569	569	73
Context RS = 3, 絞込無	569	569	73
Context RS = 3, 絞込有	569	569	74
Context RS = 5, 絞込無	569	569	73
Context RS = 5, 絞込有	569	569	73

表 6.25 より，周期タスク+20 個の非周期タスク，CPU 負荷率 =0.5 のタスクセットを実行したとき，デッドラインオーバータスク数の起動要求タスク数に対する比率は，最大のとき (シングルレジスタセットで使用レジスタ数の絞り込みを行うとき，デッドラインオーバータスク数 77) に 13.5%，最小のとき (Casablanca で従来のコンテキスト切り替えを行い，使用レジスタ数の絞り込みを行わないときと行うとき，Casablanca 上にコンテキスト保持専用レジスタセットを 1 つ設け，使用レジスタ数の絞り込みを行わないときと絞り込み行うとき，Casablanca 上にコンテキスト保持専用レジスタセットを 3 つ設け，使用レジスタ数の絞り込みを行わないとき，Casablanca 上にコンテキスト保持専用レジスタセットを 5 つ設け，使用レジスタ数の絞り込みを行わないときと絞り込み行うときの計 7 通り，デッドラインオーバータスク数 73) は 12.8%となった．提案手法により，デッドラインオーバータスク数は最大で 5.2%減少した．ただし，Casablanca 上にコンテキスト保持専用レジスタセットを 3 つ設け，使用レジスタ数の絞り込みを行うとき (デッドラインオーバータスク数 74)，行うとき (デッドラインオーバータスク数 190)，Casablanca で従来のコンテキスト切り替えを行い，使用レジスタ数の絞り込みを行わないとき (デッドラインオーバータスク数 73) に比べデッドラインオーバータスク数が多かった．

表 6.26 より，周期タスク+20 個の非周期タスク，CPU 負荷率 =0.5 のタスクセットを実行したとき，コンテキスト切り替え回数は最大 (シングルレジスタセットで使用レジスタ数の絞り込みを行うとき) で 266 回，最小 (Casablanca 上にコンテキスト保持専用レジスタセットを 1 つ設け，使用レジスタ数の絞り込みを行わないときと絞り込み行うとき，Casablanca 上にコンテキスト保持専用レジスタセットを 3 つ設け，使用レジスタ数の絞り込みを行わないときと行うとき,，Casablanca 上にコンテキスト保持専用レジスタセットを 5 つ設け，使用レジスタ数の絞り込みを行わないときの計 5 通り) で 554 回となった．表 6.25 のデッドラインオーバータスク数とコンテキスト切り替え回数より，デッドライ

表 6.26: 周期タスク+20 個の非周期タスク，CPU 負荷率= 0.5 のタスクセット実行時のコンテキスト切り替え手法の回数

	2Frames Mem.	1Frame Mem.	2Frames RS	1Frame RS
Single RS，絞込無	265	0	0	0
Single RS，絞込有	211	55	0	0
Casablanca, 絞込無	162	0	0	0
Casablanca, 絞込有	141	21	0	0
Context RS = 1, 絞込無	24	0	137	0
Context RS = 1, 絞込有	22	2	118	19
Context RS = 3, 絞込無	0	0	161	0
Context RS = 3, 絞込有	0	0	141	20
Context RS = 5, 絞込無	0	0	161	0
Context RS = 5, 絞込有	0	0	141	21

ンオーバータスク数が 3 以上減少するとき，コンテキスト切り替えの回数が減少する傾向があった.

表 6.27: 周期タスク+20 個の非周期タスクの実行で，CPU 負荷率=0.5 での各タスクの平均分布 (1)

ID	P/A	平均負荷	平均周期	平均デッドライン	実行時間	優先度	使用レジスタ数
1	P	0.015177	13664	2440	1004	1	16
2	A	0.00007	-	2086	1002	6	16
3	P	0.010324	9899	2344	1022	2	32
4	A	0.000091	-	2111	1007	7	32
5	P	0.003618	27779	5944	1005	5	32
6	P	0.028423	47995	8125	2518	3	16
7	A	0.000199	-	8319	2490	8	16
8	P	0.010739	23149	10980	2486	4	32
9	A	0.000227	-	6655	2519	9	32
10	A	0.000199	-	10742	2485	10	32
11	P	0.00218	297558	15236	6486	1	16
12	A	0.000585	-	15028	6498	6	16
13	P	0.012725	51110	9230	6504	2	32
14	A	0.000521	-	18307	6515	7	32
15	P	0.009863	65953	24908	6505	5	32
16	P	0.013547	95954	57980	12999	3	16
17	A	0.00091	-	42803	13002	8	16
18	-	-	-	-	13001	4	32
19	A	0.000911	-	47951	13019	9	32
20	A	0.001041	-	55665	13009	10	32
21	P	0.00828	265594	37840	21990	1	16
22	A	0.00198	-	50785	22000	6	16
23	-	-	-	-	22009	2	32
24	A	0.001542	-	56885	22027	7	32
25	P	0.016168	532674	185327	21987	5	32

表 6.28: 周期タスク+20 個の非周期タスクの実行で，CPU 負荷率=0.5 での各タスクの平均分布 (2)

ID	P/A	平均負荷	平均周期	平均デッドライン	実行時間	優先度	使用レジスタ数
26	-	-	-	-	33510	3	16
27	A	0.00335	-	81043	33500	8	16
28	P	0.040326	679229	138376	33499	4	32
29	A	0.003016	-	112336	33507	9	32
30	A	0.002681	-	139225	33513	10	32
31	P	0.004641	1023671	130339	47511	1	16
32	A	0.00285	-	81256	47500	6	16
33	P	0.034377	435740	120269	47485	2	32
34	A	0.003325	-	119156	47505	7	32
35	P	0.018976	1198704	268659	47485	5	32
36	P	0.016528	832807	261760	64009	3	16
37	A	0.00448	-	171812	64000	8	16
38	P	0.025526	514362	321024	63997	4	32
39	A	0.0064	-	274432	63995	9	32
40	A	0.004483	-	229229	64045	10	32
41	P	0.00588	1411124	192560	82969	1	16
42	A	0.00498	-	157146	83000	6	16
43	P	0.023305	713725	211982	82986	2	32
44	A	0.007468	-	198093	82979	7	32
45	P	0.088368	820376	561079	83050	5	32
46	P	0.013701	1633534	321860	104506	3	16
47	A	0.007315	-	333563	104500	8	16
48	P	0.01381	1559632	545908	104526	4	32
49	A	0.009408	-	389668	104537	9	32
50	A	0.009409	-	449768	104544	10	32

表 6.29: 周期タスク+20 個の非周期タスク，CPU 負荷率= 0.7 のタスクセット実行時の結果

	起動要求タスク数	実行完了タスク数	Deadline Over
Single RS，絞込無	1154	1151	237
Single RS，絞込有	1154	1151	234
Casablanca, 絞込無	1154	1151	227
Casablanca, 絞込有	1154	1151	227
Context RS = 1, 絞込無	1154	1152	227
Context RS = 1, 絞込有	1154	1151	227
Context RS = 3, 絞込無	1154	1152	227
Context RS = 3, 絞込有	1154	1152	226
Context RS = 5, 絞込無	1154	1151	229
Context RS = 5, 絞込有	1154	1151	230

表 6.29 より，周期タスク+20 個の非周期タスク，CPU 負荷率 =0.7 のタスクセットを実行したとき，デッドラインオーバータスク数の起動要求タスク数に対する比率は，最大のとき (シングルレジスタセットで使用レジスタ数の絞り込みを行わないとき，デッドラインオーバータスク数 237) に 20.5%，最小のとき (Casablanca 上にコンテキスト保持専用レジスタセットを 3 つ設け，使用レジスタ数の絞り込みを行うとき，デッドラインオーバータスク数 226) は 19.6%となった．提案手法により，デッドラインオーバータスク数は最大で 4.6%減少した．ただし，Casablanca 上にコンテキスト保持専用レジスタセットを 5 つ設け，使用レジスタ数の絞り込みを行わないとき (デッドラインオーバータスク数 229)，行うとき (デッドラインオーバータスク数 230)，Casablanca で従来のコンテキスト切り替えを行い，使用レジスタ数の絞り込みを行わないとき (デッドラインオーバータスク数 227) に比べデッドラインオーバータスク数が多かった．

表 6.30 より，周期タスク+20 個の非周期タスク，CPU 負荷率 =0.7 のタスクセットを実行したとき，コンテキスト切り替え回数は最大 (シングルレジスタセットで使用レジスタ数の絞り込みを行わないとき) で 771 回，最小 (Casablanca 上にコンテキスト保持専用レジスタセットを 3 つ設け，使用レジスタ数の絞り込みを行わないとき) で 427 回となった．表 6.29 のデッドラインオーバータスク数とコンテキスト切り替え回数より，デッドラインオーバータスク数が大きく減少 (例えば 237 から 226 へ減少) するとき，コンテキスト切り替えの回数が減少する傾向があった．

表 6.30: 周期タスク+20 個の非周期タスク，CPU 負荷率=0.7 のタスクセット実行時のコンテキスト切り替え手法の回数

	2Frames Mem.	1Frame Mem.	2Frames RS	1Frame RS
Single RS，絞込無	771	0	0	0
Single RS，絞込有	465	301	0	0
Casablanca, 絞込無	433	0	0	0
Casablanca, 絞込有	263	168	0	0
Context RS = 1, 絞込無	121	0	309	0
Context RS = 1, 絞込有	61	52	202	115
Context RS = 3, 絞込無	3	0	424	0
Context RS = 3, 絞込有	2	1	259	166
Context RS = 5, 絞込無	0	0	432	0
Context RS = 5, 絞込有	0	0	263	173

表 6.31: 周期タスク+20 個の非周期タスクの実行で，CPU 負荷率=0.7 での各タスクの平均分布 (1)

ID	P/A	平均負荷	平均周期	平均デッドライン	実行時間	優先度	使用レジスタ数
1	P	0.020542	9846	2228	1004	1	16
2	A	0.00008	-	2034	1002	6	16
3	P	0.026233	7897	2638	1022	2	32
4	A	0.000081	-	2252	1007	7	32
5	P	0.016341	12459	6423	1005	5	32
6	P	0.044627	17239	10516	2518	3	16
7	A	0.000199	-	7089	2490	8	16
8	P	0.03724	33914	14179	2486	4	32
9	A	0.000202	-	8887	2519	9	32
10	A	0.000249	-	9045	2485	10	32
11	P	0.008529	76044	9204	6486	1	16
12	A	0.000455	-	13126	6498	6	16
13	-	-	-	-	6504	2	32
14	A	0.000586	-	15972	6515	7	32
15	P	0.026331	50452	41756	6505	5	32
16	P	0.025911	152305	35880	12999	3	16
17	A	0.00117	-	43044	13002	8	16
18	P	0.038755	126838	62088	13001	4	32
19	A	0.000911	-	52668	13019	9	32
20	A	0.001171	-	44061	13009	10	32
21	-	-	-	-	21990	1	16
22	A	0.0011	-	43331	22000	6	16
23	P	0.019246	266468	43252	22009	2	32
24	A	0.001542	-	52359	22027	7	32
25	P	0.020727	106077	257488	21987	5	32

表 6.32: 周期タスク+20 個の非周期タスクの実行で，CPU 負荷率=0.7 での各タスクの平均分布 (2)

ID	P/A	平均負荷	平均周期	平均デッドライン	実行時間	優先度	使用レジスタ数
26	P	0.052304	307460	126462	33510	3	16
27	A	0.00268	-	95073	33500	8	16
28	P	0.040359	303265	167142	33499	4	32
29	A	0.002681	-	105692	33507	9	32
30	A	0.002681	-	152893	33513	10	32
31	P	0.007371	1359414	68779	47511	1	16
32	A	0.003325	-	93805	47500	6	16
33	P	0.020988	1019835	129865	47485	2	32
34	A	0.0038	-	122312	47505	7	32
35	P	0.042938	375686	321860	47485	5	32
36	P	0.021616	296122	346880	64009	3	16
37	A	0.00576	-	190975	64000	8	16
38	P	0.035976	355789	468480	63997	4	32
39	A	0.00512	-	263840	63995	9	32
40	A	0.005764	-	288085	64045	10	32
41	P	0.011274	735927	123504	82969	1	16
42	A	0.00664	-	177204	83000	6	16
43	P	0.016202	512203	193556	82986	2	32
44	A	0.006638	-	171892	82979	7	32
45	P	0.015539	1075167	541160	83050	5	32
46	P	0.037639	966233	483486	104506	3	16
47	A	0.00836	-	298870	104500	8	16
48	P	0.031273	669122	468995	104526	4	32
49	A	0.008363	-	339886	104537	9	32
50	A	0.008364	-	425837	104544	10	32

表 6.33: 周期タスク+20 個の非周期タスク，CPU 負荷率= 0.9 のタスクセット実行時の結果

	起動要求タスク数	実行完了タスク数	Deadline Over
Single RS，絞込無	1344	1339	361
Single RS，絞込有	1344	1339	358
Casablanca, 絞込無	1344	1340	341
Casablanca, 絞込有	1344	1340	344
Context RS = 1, 絞込無	1344	1340	343
Context RS = 1, 絞込有	1344	1340	341
Context RS = 3, 絞込無	1344	1340	338
Context RS = 3, 絞込有	1344	1340	341
Context RS = 5, 絞込無	1344	1339	350
Context RS = 5, 絞込有	1344	1338	353

表 6.33 より，周期タスク+20 個の非周期タスク，CPU 負荷率 =0.9 のタスクセットを実行したとき，デッドラインオーバータスク数の起動要求タスク数に対する比率は，最大のとき (シングルレジスタセットで使用レジスタ数の絞り込みを行わないとき，デッドラインオーバータスク数 361) に 26.9%，最小のとき (Casablanca 上にコンテキスト保持専用レジスタセットを 3 つ設け，使用レジスタ数の絞り込みを行わないとき，デッドラインオーバータスク数 338) は 25.1%となった．提案手法により，デッドラインオーバータスク数は最大で 6.37%減少した．ただし，Casablanca で従来のコンテキスト切り替えを行い，使用レジスタ数の絞り込みを行うとき (デッドラインオーバータスク数 344)，Casablanca 上にコンテキスト保持専用レジスタセットを 1 つ設け，使用レジスタ数の絞り込みを行わないとき (デッドラインオーバータスク数 343)，Casablanca 上にコンテキスト保持専用レジスタセットを 5 つ設け，使用レジスタ数の絞り込みを行わないとき (デッドラインオーバータスク数 350)，行うとき (デッドラインオーバータスク数 353)，Casablanca で従来のコンテキスト切り替えを行い，使用レジスタ数の絞り込みを行わないとき (デッドラインオーバータスク数 341) に比べデッドラインオーバータスク数が多かった．

表 6.34 より，周期タスク+20 個の非周期タスク，CPU 負荷率 =0.9 のタスクセットを実行したとき，コンテキスト切り替え回数は最大 (シングルレジスタセットで使用レジスタ数の絞り込みを行わないとき) で 1218 回，最小 (Casablanca 上にコンテキスト保持専用レジスタセットを 3 つ設け，使用レジスタ数の絞り込みを行わないとき) で 693 回となった．表 6.33 のデッドラインオーバータスク数とコンテキスト切り替え回数より，デッドラインオーバータスク数が大きく減少 (例えば 361 から 338 へ減少) するとき，コンテキスト切り替えの回数が減少する傾向があった．

表 6.34: 周期タスク+20 個の非周期タスク，CPU 負荷率= 0.9 のタスクセット実行時のコンテキスト切り替え手法の回数

	2Frames Mem.	1Frame Mem.	2Frames RS	1Frame RS
Single RS，絞込無	1218	0	0	0
Single RS，絞込有	690	522	0	0
Casablanca, 絞込無	713	0	0	0
Casablanca, 絞込有	479	229	0	0
Context RS = 1, 絞込無	441	0	269	0
Context RS = 1, 絞込有	276	138	196	94
Context RS = 3, 絞込無	7	0	686	0
Context RS = 3, 絞込有	80	1	390	232
Context RS = 5, 絞込無	0	0	730	0
Context RS = 5, 絞込有	16	0	469	248

表 6.35: 周期タスク+20 個の非周期タスクの実行で，CPU 負荷率=0.9 での各タスクの平均分布 (1)

ID	P/A	平均負荷	平均周期	平均デッドライン	実行時間	優先度	使用レジスタ数
1	P	0.022829	4398	1264	1004	1	16
2	A	0.00002	-	1723	1002	6	16
3	P	0.027116	3769	1503	1022	2	32
4	A	0.00004	-	2457	1007	7	32
5	P	0.01129	8902	11320	1005	5	32
6	P	0.028198	24878	9925	2518	3	16
7	A	0.000124	-	6444	2490	8	16
8	P	0.037386	14275	5419	2486	4	32
9	A	0.00005	-	10225	2519	9	32
10	A	0.000124	-	10443	2485	10	32
11	P	0.015281	42446	13832	6486	1	16
12	A	0.000195	-	10798	6498	6	16
13	P	0.042256	34302	11687	6504	2	32
14	A	0.000261	-	13675	6515	7	32
15	P	0.04812	28530	68900	6505	5	32
16	P	0.127103	51871	38583	12999	3	16
17	A	0.00091	-	36058	13002	8	16
18	P	0.006499	200048	75816	13001	4	32
19	A	0.00026	-	37440	13019	9	32
20	A	0.00039	-	43680	13009	10	32
21	P	0.066004	245829	39212	21990	1	16
22	A	0.0011	-	37839	22000	6	16
23	P	0.057154	172360	53349	22009	2	32
24	A	0.001322	-	54736	22027	7	32
25	P	0.018311	120077	176175	21987	5	32

表 6.36: 周期タスク+20 個の非周期タスクの実行で，CPU 負荷率=0.9 での各タスクの平均分布 (2)

ID	P/A	平均負荷	平均周期	平均デッドライン	実行時間	優先度	使用レジスタ数
26	P	0.026329	127275	105190	33510	3	16
27	A	0.00134	-	97685	33500	8	16
28	P	0.041766	281733	193585	33499	4	32
29	A	0.00067	-	81404	33507	9	32
30	A	0.001341	-	166964	33513	10	32
31	P	0.01696	280136	91960	47511	1	16
32	A	0.001425	-	105513	47500	6	16
33	P	0.022454	664845	94809	47485	2	32
34	A	0.001425	-	117989	47505	7	32
35	P	0.061282	255964	376579	47485	5	32
36	P	0.02525	566064	246400	64009	3	16
37	A	0.0032	-	182169	64000	8	16
38	P	0.018612	899946	312832	63997	4	32
39	A	0.00448	-	227474	63995	9	32
40	A	0.001921	-	224256	64045	10	32
41	-	-	-	-	82969	1	16
42	A	0.00249	-	167992	83000	6	16
43	-	-	-	-	82986	2	32
44	A	0.003319	-	267259	82979	7	32
45	P	0.052005	492650	430935	83050	5	32
46	P	0.019592	1112871	438900	104506	3	16
47	A	0.005225	-	347943	104500	8	16
48	P	0.06231	513930	533646	104526	4	32
49	A	0.005227	-	466488	104537	9	32
50	A	0.004182	-	450603	104544	10	32

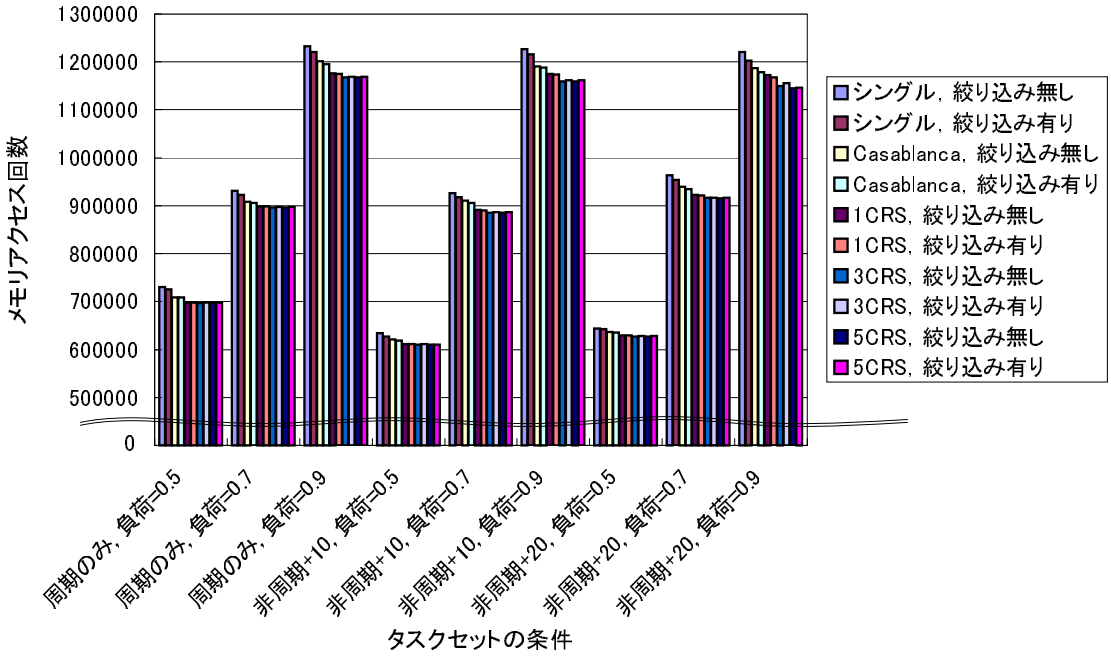

図 6.16: メモリアクセス回数

図 6.16 より，メモリアクセス回数は，全てのタスクセットについて，Casablanca 上で実行したときとシングルレジスタセットで実行したときを比較すると，Casablanca 上で提案手法を使うときと使わないときの減少幅に比べて，Casablanca で実行したときのメモリアクセス回数は，シングルレジスタセットで実行したときのメモリアクセス回数から大きく減少した．これは，Casablanca の高速割り込み応答機能による効果である．また，レジスタセット間コンテキスト切り替えを行うことでメモリアクセス回数は減少し，タスクの使用レジスタ数の絞り込みによりメモリアクセス回数が減少した．ただし，フレーム管理テーブルをメモリに設けたことで，コンテキスト保持専用レジスタセット数が 3，5 の時使用レジスタ数の絞り込みにより，メモリアクセス回数が増えた場合があった．図 6.17 より，データキャッシュミスはメモリアクセス回数 (図 6.16) と同じ傾向になった．これは，メモリアクセス回数が少なければ，データキャッシュアクセスが減り，データキャッシュミスが減ることによる効果である．図 6.18 より，命令キャッシュは提案手法により，増加することが多かった．これは，レジスタセット間コンテキスト切り替えとタスクの使用レジスタ数の絞り込みを行うことで，コンテキスト切り替えルーチンのコードサイズが大きくなり (表 6.37)，入力ファイル全体のコードサイズが大きくなったことの影響である．(表 6.38 参照) 6.19 より，平均応答時間はメモリアクセス回数 (図 6.16) の減少により，減少してたことが分かる．提案手法によるデータキャッシュミス数の減少が大きい (図 6.18) ため，命令キャッシュミス数が大きい場合でも応答時間が短縮できた．ただし，コンテキスト保持専用レジスタセット数が 5 のとき，命令キャッシュミスが Casablanca で従来のコンテキスト切り替えを行い，タスクの使用レジスタ数の絞り込みを行わないときに対し

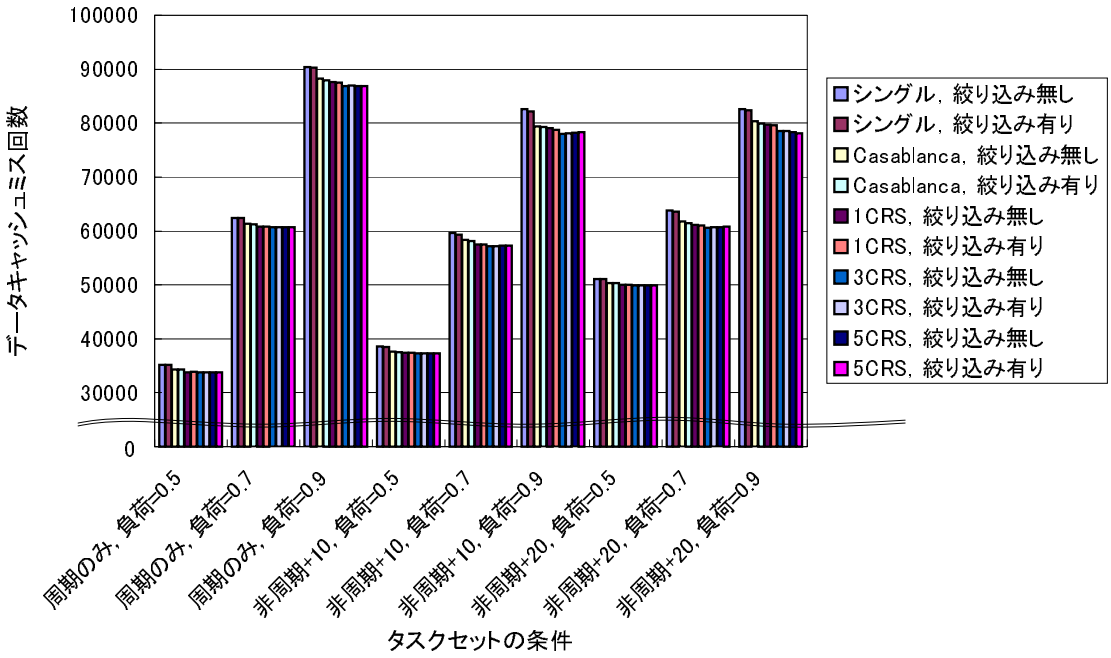

図 6.17: データキャッシュミス回数

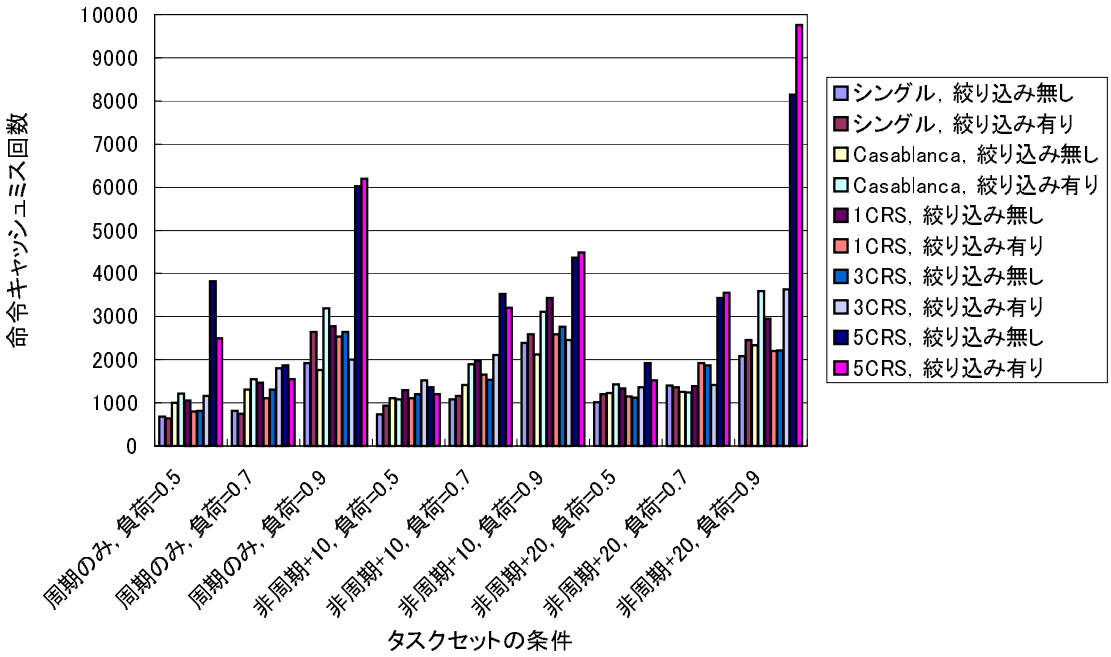

図 6.18: 命令キャッシュミス回数

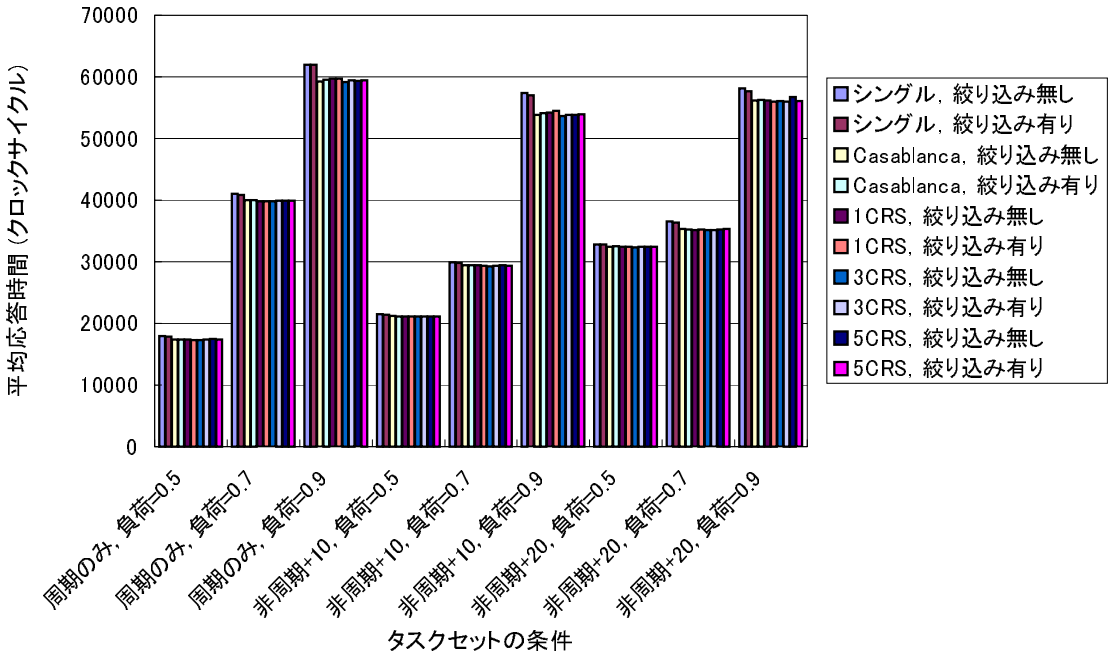

図 6.19: 平均応答時間

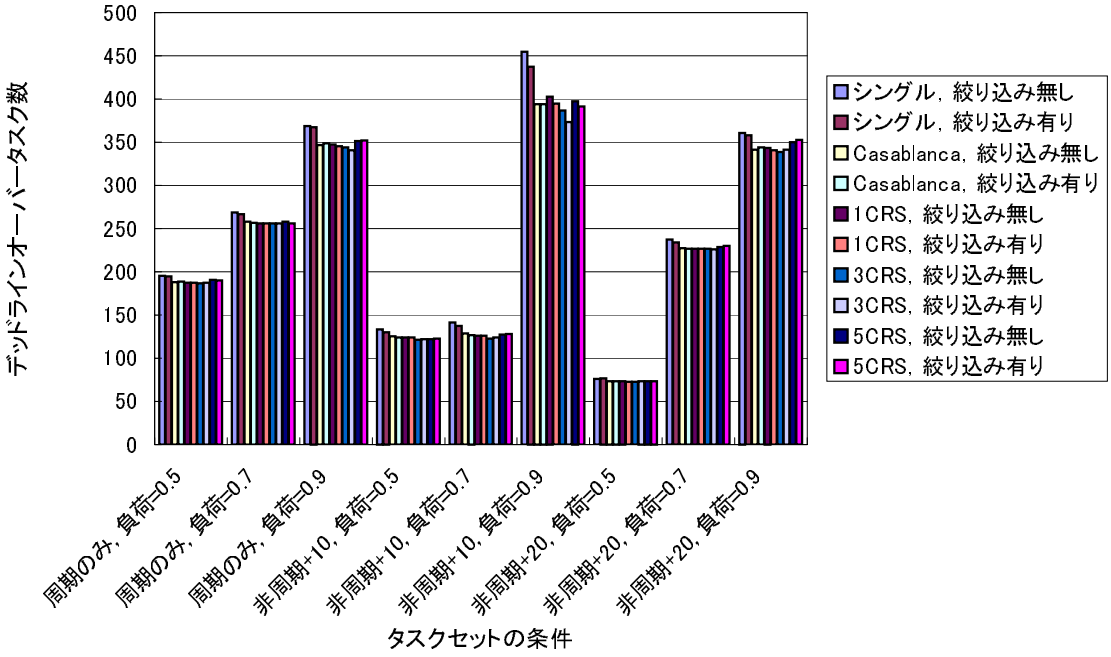

図 6.20: デッドラインオーバーしたユーザタスク数

て，コンテキスト保持専用レジスタセット数が 5 で，タスクの使用レジスタ数の絞り込みを行わないときは最大 3.78 倍，絞り込みを行うときは最大 4.18 倍となる．このため，平均応答時間が Casablanca で従来のコンテキスト切り替えを行い，タスクの使用レジスタ数の絞り込みを行わないときに対して大きくなるときがあった．図 6.20 より，デッドラインオーバーしたユーザタスク数は平均応答時間 (図 6.19) の減少により，減少していく傾向があった．タスクの応答時間が短くなることにより，デッドラインオーバーを回避できるからである．

表 6.37: 各コンテキスト切り替え手法のコードサイズ

	コードサイズ (Byte)	Single RS，絞込無との比率 (%)
Single RS，絞込無	548	100
Single RS，絞込有	704	128
Casablanca, 絞込無	328	60
Casablanca, 絞込有	512	93
Context RS = 1, 絞込無	668	122
Context RS = 1, 絞込有	1288	235
Context RS = 3, 絞込無	1324	242
Context RS = 3, 絞込有	2776	507
Context RS = 5, 絞込無	1980	361
Context RS = 5, 絞込有	4260	777

表 6.38: 各コンテキスト切り替え手法を適用した場合の入力ファイルのコードサイズ

	コードサイズ (Byte)	Single RS，絞込無との比率 (%)
Single RS，絞込無	22848	100
Single RS，絞込有	23004	101
Casablanca, 絞込無	26736	117
Casablanca, 絞込有	26920	118
Context RS = 1, 絞込無	27076	119
Context RS = 1, 絞込有	27696	121
Context RS = 3, 絞込無	27732	121
Context RS = 3, 絞込有	29184	128
Context RS = 5, 絞込無	28838	124
Context RS = 5, 絞込有	30668	134

6.9 考察

6.9.1 優位性

今回提案手法により，改善された点は，

1. メモリアクセス回数が最大 6%減少
2. 1. に伴いデータキャッシュミス数も最大 5%減少
3. 平均応答時間が最大 7.5%減少
4. デッドラインオーバータスク数が最大 18%減少

である．

1. と 2. については，コンテキスト切り替えをレジスタセット間のみでメモリアクセスを行わないとき，メモリアクセス回数が最大 6%軽減された．特に，コンテキスト保持専用レジスタセットを 3 つ以上にしたとき，CPU 負荷率が 0.9 と高い場合でも，コンテキスト切り替え時にメモリアクセスを行う回数が少なかった．3. と 4. については最大値の向上が見られるときには，命令キャッシュミスが少ない．1. と 2. でデータキャッシュミスペナルティと命令数が減少しているため，命令キャッシュのミスが少なければ，ユーザタスクの応答時間が短縮され，デッドラインオーバーするタスクが減少した．

6.9.2 問題点

シミュレーション結果から分かった問題点は，

1. コンテキスト保持専用レジスタセットが 5 つの場合，命令キャッシュミスが増大し，データキャッシュミスの減少による効果を打ち消した
2. 使用レジスタ数の絞り込みの結果，命令キャッシュミスが増えてしまう場合があり，絞り込みを行わないときに比べて，平均応答時間，が増え，デッドラインオーバータスク数が増える場合があった

である．

命令キャッシュミスの増大は以下のことが原因である．

1. ユーザタスク実行時にカーネルが使っていた命令キャッシュのブロックが置換された
2. コンテキスト切り替えの手法が切り替えの度に変わってしまい，ファーストミスが頻発した

1. については，一部の場合に傾向が見られた．1. が起こった場合，命令キャッシュのミス数は大幅に増えていた．しかし，全体を見渡す限り提案手法によって，2. でのミスが多く見られた．特に，コンテキスト保持専用レジスタセットが複数あり，使用レジスタ数の絞り込みを行うことでコンテキスト切り替えルーチンのサイズが大きくなる場合，命令キャッシュミスが多かった (図 6.18 参照)．レジスタはアドレッサブルでないため，各フレームのコンテキスト切り替えに全て網羅しなければならなく，コンテキスト切り替えルーチンのコードサイズが大きくなったことが原因である．命令キャッシュミスの増大により，Casablanca 上での従来のコンテキスト切り替えを行うときより，平均応答時間，実行完了タスク数，デッドラインオーバータスク数が多くなることがあった．(表 6.1，表 6.2，表 6.5，表 6.6，表 6.9，表 6.10，表 6.13，表 6.14，表 6.17，表 6.18，表 6.21，表 6.22，表 6.25，表 6.18，表 6.29，表 6.30，表 6.33，表 6.34，図 6.19，図 6.20 参照)

6.9.3 CPU の負荷率との因果関係

今回，ユーザタスクによる CPU 負荷率を 0.5，0.7，0.9 と変えてシミュレーションを行った．CPU の負荷率が高い程以下の傾向があった．

1. 全体として言えることは，
   (a) 命令キャッシュミス数が増加 (図 6.18 参照)
   (b) 平均応答時間が増加 (図 6.19 参照)
2. シングルレジスタセットでコンテキスト切り替えを行ったときと Casablanca でコンテキスト切り替えを行ったとき，Casablanca で実行させることにより，コンテキスト切り替え回数，デッドラインオーバータスク数，メモリアクセス回数，データキャッシュミス数について，Casablanca 上で提案手法を使うときと使わないときに比べて，大きく減少した．(表 6.1，表 6.2，表 6.5，表 6.6，表 6.9，表 6.10，表 6.13，表 6.14，表 6.17，表 6.18，表 6.21，表 6.22，表 6.25，表 6.18，表 6.29，表 6.30，表 6.33，表 6.34，図 6.16，図 6.17，図 6.20 参照)
3. 負荷率が 0.5 の場合ではコンテキスト保持専用レジスタセットを複数備える状況下ではレジスタセット間でのコンテキスト切り替えだけで済んでいたのが，負荷率が 0.7 より大きくなるとコンテキスト保持専用レジスタセット 3 つでは収まらない状況になり，負荷率 0.9 の場合，5 つあるコンテキスト保持専用レジスタセットが全て埋まり，メモリへ退避させる状況が発生した (表 6.1，表 6.2，表 6.5，表 6.6，表 6.9，表 6.10，表 6.13，表 6.14，表 6.17，表 6.18，表 6.21，表 6.22，表 6.25，表 6.18，表 6.29，表 6.30，表 6.33，表 6.34 参照)

1-(a) は，負荷率が増えれば扱うタスクの数は増えて，コンテキスト切り替えの回数は増え，命令キャッシュが頻繁に置換されるためである．1-(b) は．負荷の増加のために平均

待ちタスク数が大きくなったためである．2. については，Casabalanca の高速割り込み応答機能によるオーバヘッド削減の効果である．外部トラップ処理時にユーザタスクを一時退避させる必要が無いことがオーバヘッド削減を実現させている．更に，命令キャッシュのミスが少なければ，提案手法によりユーザタスクのコンテキスト切り替え時のメモリアクセスが減少し，オーバヘッド削減が可能となる．3. は，優先度の低いタスクがコンテキスト保持専用レジスタセット内に取り残されたままになる状況が起こり，コンテキスト保持専用レジスタセット内の空きフレームが減るからである．

6.9.4 ユーザタスクの実行時間での違い

ユーザタスクのタスクセットの平均実行時間が短い場合，命令キャッシュミスが少なければ，提案手法により大幅にデッドラインオーバータスク数が減少した (例えば，表 6.39, 表 6.40(各項目は，左から順にタスク ID，ユーザタスク ID，周期タスクと非周期タスクの分別，タスクの負荷 (計算式 (1) で計算) と，ユーザタスクの起動周期，相対デッドライン，ユーザタスクの実行時間，優先度，使用レジスタ数となっており，上から順にタスク ID:1 から ID:50 までについて記載する) タスクセットの場合，表 6.41 (見方は表 6.1 と同じ) より，4135 回の起動要求，実行完了タスク数が 4131 で，シングルレジスタセットで従来のコンテキスト切り替えを行った結果デッドラインオーバータスク数が 1112，Casablanca 上で従来のコンテキスト切り替えを行った結果デッドラインオーバータスク数が 807，コンテキスト保持専用レジスタセット数が 3 で使用レジスタ数の絞り込みを行った場合デッドラインオーバータスク数が 615 となった．最大でデッドラインオーバータスク数を 44.7%減少)．逆に平均実行時間が長いタスクセットを実行した場合，デッドラインオーバータスク数が全ての手法で同じになる場合があった (例えば，表 6.42，表 6.43(各項目の見方は表 6.40 と同じ) のタスクセットの場合，表 6.44(各項目の見方は 6.41 と同じ) より，356 回の起動要求，実行完了タスク数が 351 となり，全てのコンテキスト切り替えの方式でデッドラインオーバータスク数が 18 となった)．これは，実行時間の短いタスクセットでの周期タスクは実行回数が多いため，実行時間の長いタスクセットでの周期に比べて周期間隔が短く，デッドラインについても短くなる．そのため，コンテキスト切り替えのオーバヘッド削減によって，デッドラインオーバーを回避したタスクの数が実行時間の長いタスクセットに比べて，実行時間の短いタスクセットでは多くなった．

表 6.39: タスクの実行時間の短いタスクセット実行での各タスクの分布 (1)

ID	P/A	負荷	周期	デッドライン	実行時間	優先度	使用レジスタ数
1	P	0.228286	4398	1264	1004	1	16
2	-	-	-	-	1002	6	16
3	-	-	-	-	1022	2	32
4	P	0.112896	8902	11320	1007	7	32
5	-	-	-	-	1005	5	32
6	A	0.000249	-	8820	2518	3	16
7	-	-	-	-	2490	8	16
8	-	-	-	-	2486	4	32
9	-	-	-	-	2519	9	32
10	-	-	-	-	2485	10	32
11	-	-	-	-	6486	1	16
12	A	0.00065	-	7904	6498	6	16
13	P	0.27895	23316	11336	6504	2	32
14	-	-	-	-	6515	7	32
15	P	0.185544	35059	67652	6505	5	32
16	-	-	-	-	12999	3	16
17	A	0.0013	-	51272	13002	8	16
18	-	-	-	-	13001	4	32
19	-	-	-	-	13019	9	32
20	-	-	-	-	13009	10	32
21	P	0.036522	602103	35024	21990	1	16
22	-	-	-	-	22000	6	16
23	-	-	-	-	22009	2	32
24	-	-	-	-	22027	7	32
25	-	-	-	-	21987	5	32

表 6.40: タスクの実行時間の短いタスクセット実行での各タスクの分布 (2)

ID	P/A	負荷	周期	デッドライン	実行時間	優先度	使用レジスタ数
26	-	-	-	-	33510	3	16
27	A	0.00335	-	94604	33500	8	16
28	-	-	-	-	33499	4	32
29	-	-	-	-	33507	9	32
30	-	-	-	-	33513	10	32
31	-	-	-	-	47511	1	16
32	-	-	-	-	47500	6	16
33	-	-	-	-	47485	2	32
34	A	0.00475	-	155800	47505	7	32
35	-	-	-	-	47485	5	32
36	-	-	-	-	64009	3	16
37	A	0.0064	-	144896	64000	8	16
38	-	-	-	-	63997	4	32
39	-	-	-	-	63995	9	32
40	-	-	-	-	64045	10	32
41	-	-	-	-	82969	1	16
42	A	0.0083	-	189904	83000	6	16
43	-	-	-	-	82986	2	32
44	A	0.008298	-	297140	82979	7	32
45	-	-	-	-	83050	5	32
46	-	-	-	-	104506	3	16
47	A	0.01045	-	271700	104500	8	16
48	-	-	-	-	104526	4	32
49	-	-	-	-	104537	9	32
50	A	0.010454	-	4837207	104544	10	32

表 6.41: 実行時間の短いタスクセット実行時の結果

	起動要求タスク数	実行完了タスク数	Deadline Over
Single RS，絞込無	4135	4131	1112
Single RS，絞込有	4135	4131	979
Casablanca, 絞込無	4135	4131	807
Casablanca, 絞込有	4135	4131	803
Context RS = 1, 絞込無	4135	4131	833
Context RS = 1, 絞込有	4135	4131	807
Context RS = 3, 絞込無	4135	4131	758
Context RS = 3, 絞込有	4135	4131	615
Context RS = 5, 絞込無	4135	4131	772
Context RS = 5, 絞込有	4135	4131	740

表 6.42: タスクの実行時間の長いタスクセット実行での各タスクの分布 (1)

ID	P/A	負荷	周期	デッドライン	実行時間	優先度	使用レジスタ数
1	-	-	-	-	1004	1	16
2	A	0.0001	-	1351	1002	6	16
3	-	-	-	-	1022	2	32
4	-	-	-	-	1007	7	32
5	-	-	-	-	1005	5	32
6	-	-	-	-	2518	3	16
7	A	0.000249	-	4620	2490	8	16
8	-	-	-	-	2486	4	32
9	A	0.000252	-	8650	2519	9	32
10	A	0.000249	-	5139	2485	10	32
11	-	-	-	-	6486	1	16
12	A	0.00065	-	10764	6498	6	16
13	-	-	-	-	6504	2	32
14	A	0.000052	-	8840	6515	7	32
15	-	-	-	-	6505	5	32
16	P	0.258537	-	20280	12999	3	16
17	-	-	-	-	13002	8	16
18	P	0.064989	-	75816	13001	4	32
19	-	-	-	-	13019	9	32
20	-	-	-	-	13009	10	32
21	-	-	-	-	21990	1	16
22	-	-	-	-	22000	6	16
23	-	-	-	-	22009	2	32
24	-	-	-	-	22027	7	32
25	-	-	-	-	21987	5	32

表 6.43: タスクの実行時間の長いタスクセット実行での各タスクの分布 (2)

ID	P/A	負荷	周期	デッドライン	実行時間	優先度	使用レジスタ数
26	-	-	-	-	33510	3	16
27	-	-	-	-	33500	8	16
28	P	0.076043	-	162140	33499	4	32
29	-	-	-	-	33507	9	32
30	-	-	-	-	33513	10	32
31	-	-	-	-	47511	1	16
32	-	-	-	-	47500	6	16
33	-	-	-	-	47485	2	32
34	-	-	-	-	47505	7	32
35	P	0.291646	-	496660	47485	5	32
36	-	-	-	-	64009	3	16
37	-	-	-	-	64000	8	16
38	-	-	-	-	63997	4	32
39	A	0.0064	-	316160	63995	9	32
40	A	0.006405	-	202240	64045	10	32
41	-	-	-	-	82969	1	16
42	-	-	-	-	83000	6	16
43	-	-	-	-	82986	2	32
44	-	-	-	-	82979	7	32
45	-	-	-	-	83050	5	32
46	-	-	-	-	104506	3	16
47	A	0.01045	-	0.01045	104500	8	16
48	P	0.168724	-	228228	104526	4	32
49	A	0.010454	-	570570	104537	9	32
50	-	-	-	-	104544	10	32

表 6.44: 実行時間の長いタスクセット実行時の結果

	起動要求タスク数	実行完了タスク数	Deadline Over
Single RS，絞込無	356	352	18
Single RS，絞込有	356	352	18
Casablanca, 絞込無	356	352	18
Casablanca, 絞込有	356	352	18
Context RS = 1, 絞込無	356	352	18
Context RS = 1, 絞込有	356	352	18
Context RS = 3, 絞込無	356	352	18
Context RS = 3, 絞込有	356	352	18
Context RS = 5, 絞込無	356	352	18
Context RS = 5, 絞込有	356	352	18

6.9.5 非周期タスク数の影響

今回，優先度の関係としては，周期タスク＞非周期タスクとした．このことにより，

1. CPU が高負荷の場合，非周期タスクがコンテキスト保持専用レジスタセット内に常駐する傾向になり，提案手法の優位性を損なう

2. 非周期タスクが取り残されたままの状況が起こるため，デッドラインオーバータスク数が同じ CPU 負荷率の周期タスクのみのセットに比べて多くなる傾向がある

が分かった．

1. により，コンテキスト保持専用レジスタセット内に取り残されたままになるため提案手法を盛りこんだ場合でも，従来のコンテキストせざるを得ない状況が起き，結果に顕著に現れていた．2. については，優先度の低いタスクがコンテキスト保持専用レジスタセット内に取り残されたままになる状況が起こり，コンテキスト保持専用レジスタセット内の空きフレームが減るからである．(図 6.21 に例を示す．この例では，優先度の低いタスクをタスク ID : 6 から 10(タスク ID : 6 の優先度は 6, タスク ID: 7 の優先度は 7, タスク ID : 8 の優先度は 8, タスク ID : 9 の優先度は 9, タスク ID : 10 の優先度は 10) とし, 優先度の高いタスクをタスク ID : 1，2，4，5(タスク ID : 1 の優先度が 1，タスク ID : 2 の優先度が 2，タスク ID : 4 の優先度は 4, タスク ID : 5 の優先度は 5,) とする．タスク ID : 4 〜 10 のコンテキストがフレーム内にあり，全てのフレームを使い切っているとき，実行中のタスク ID : 2 から起動要求のあったタスク ID: 1 に切り替えるとき，空きフレームが無いため，タスク ID : 2 はメモリへ退避させなければならない．)

6.9.6 Casablanca の効果

Casablanca の高速割り込み応答機能は以下の点で優位性を示した．

1. ユーザタスクから外部トラップへの移行時にユーザタスクのコンテキストを一時退避させる必要がない

2. 1. によってコンテキスト切り替えの回数は減少する

3. 2. により，CPU が高負荷になった場合でもシングルレジスタセットに比べて応答時間，デッドラインオーバータスク数，デッドラインオーバー時間が短縮された

4. 2. により，平均実行時間が短いタスクセットでシングルレジスタセットに比べて応答時間，デッドラインオーバータスク数，デッドラインオーバー時間が短縮された

今回，ユーザタスクから外部トラップへの移行時にユーザタスクのコンテキストの一時退避を行わないことで，2.，3. で示したように CPU が高負荷の場合やタスクセットの平均実行時間が短い場合に効果があった．

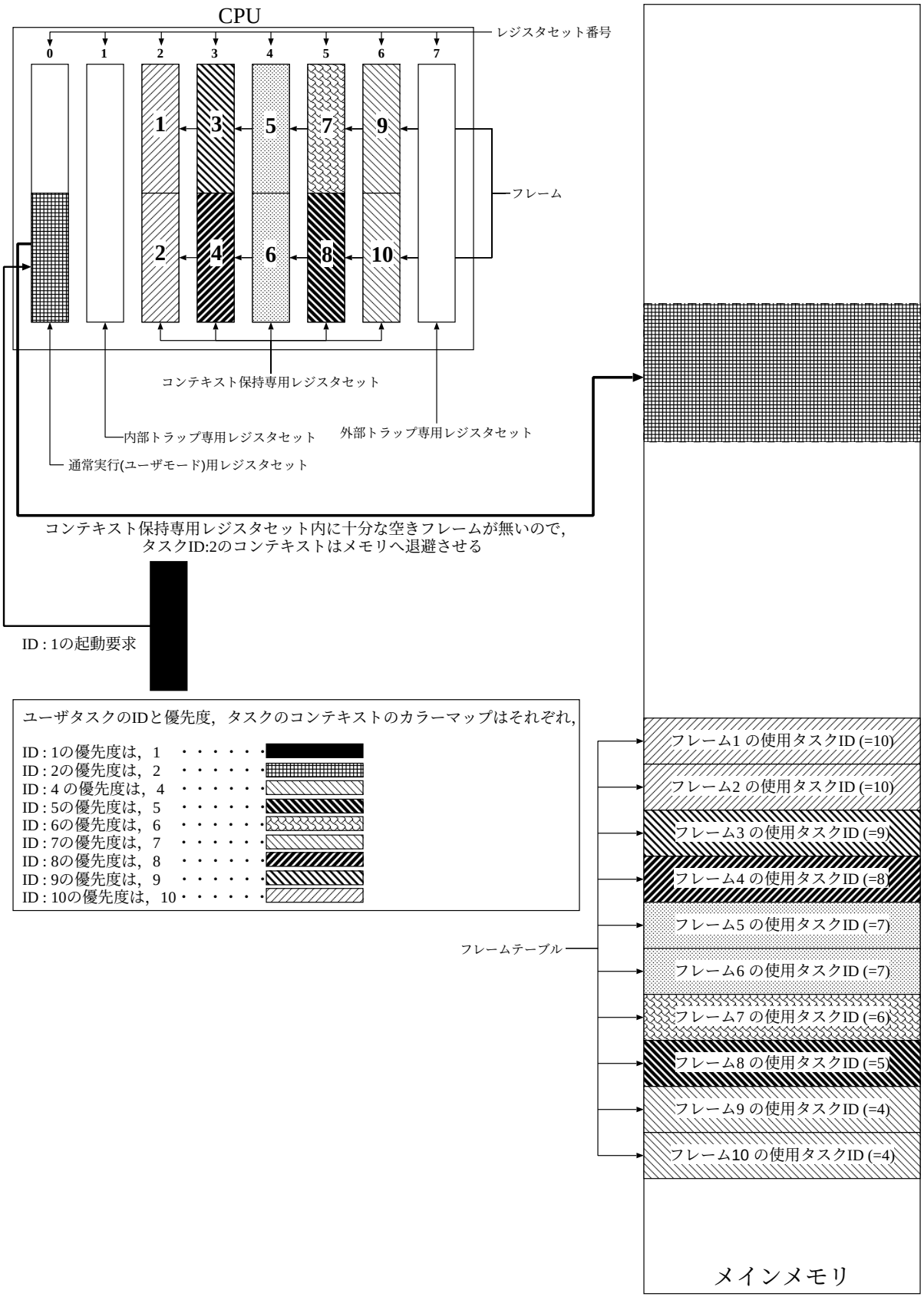

図 6.21: 優先度の低いタスクのコンテキストによってフレームが埋まる例

6.9.7　コンテキスト保持専用レジスタセット数について

今回，コンテキスト保持専用レジスタセット数を 1, 3, 5 としてシミュレーションを行った．その結果，

1. コンテキスト保持専用レジスタセット数が 1 の場合は，フレームが埋まることが多く，従来のコンテキストを行う状況が多かった

2. コンテキスト保持専用レジスタセット数が 5 の場合は，CPU が高負荷になってもフレームが埋まることは殆んど無かったが，コンテキスト切り替えルーチンのサイズが大きくなり，それによる命令キャッシュのミスが増加した．命令キャッシュミスが応答時間を増大させる場合があった

3. コンテキスト保持専用レジスタセット数が 3 の場合は，高負荷でも，全てのフレームが埋まることは少なく，命令キャッシュミスもコンテキスト保持専用レジスタセット数=5 のとき程大きくなかった

となった．今回のシミュレーションでは，コンテキスト保持専用レジスタセット数は 3 のときオーバヘッド削減が大きく提案手法が有効であった．

6.9.8　ユーザタスクの絞り込みの効果

ユーザタスクの絞り込みを行った結果，以下のことが分かった

1. メモリアクセス回数，データキャッシュミス回数の軽減

2. 1. による平均応答時間の短縮化，デッドラインオーバータスク数の減少

3. コンテキスト保持専用レジスタセットのフレーム化することでコンテキスト切り替えルーチンが肥大化し，命令キャッシュミスが増大した

1. はコンテキスト切り替え時に 32 本のレジスタを移動させていたのが，絞り込みにより，16 本の移動で済む場合ができたことの効果によるものである．絞り込み無しと比べて命令キャッシュのミスの増加が少ないときは，データキャッシュミスの軽減により，2. が実現できている．3. については，レジスタがアドレッサブルで無いため，絞り込みを行わないときに比べて，ルーチンサイズを大きくせざるを得なかった．このことより命令キャッシュが増加し，平均応答時間が悪化することもあった．

第7章　関連研究

汎用OSでは，容量の大きい2次キャッシュ，3次キャッシュを設け，キャッシュミスによるミスペナルティを軽減させている．キャッシュサイズを大きくし，2次，3次キャッシュによってミスペナルティが減少し，コンテキスト切り替えが高速化できる [1]．しかし，10MB近い3次キャッシュなど搭載させるにはコストが掛かり過ぎるため，この方法でコンテキスト切り替えを高速化させる手法は向いていない．

現在，組み込みRTOSでのコンテキスト切り換えの高速化手法として，レジスタバンク (レジスタセット) をハードウェアで切り替えるる手法が既に実現されている [11]．この研究では，レジスタバンクの切り替えだけではなく，タスクのコンテキストをコンテキストメモリ (SRAM) に蓄えられることができる．(概要は図 7.1) この手法では，レジスタバンクを専用のハードウェアで切り替えるため，高速にコンテキスト切り替えができるのみならず，レジスタバンク内にタスクのコンテキストが無い場合にはコンテキストメモリから読み出すことにより外部メモリの参照を少なくし，高速化を達成している．(コンテキストメモリがタスクのコンテキスト専用のキャッシュの働きをしている．)

今回は，ソフトウェアだけでのコンテキスト切り替えの高速化を狙って研究を行って来たが，今後の研究では，提案した2つの手法とこれらハードウェアでの高速化手法とを組み合わせて進める必要がある．

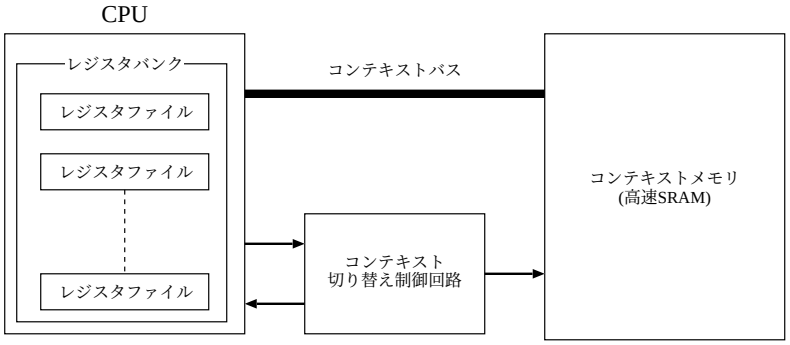

図 7.1: レジスタバンクをハードウェアで切り替える手法の概要図

第8章　まとめ

8.1　総括

本研究ではコンテキスト切り替え高速化の手法として以下の 2 つの手法を提案した.

- レジスタセット間コンテキスト切り替え
- ユーザタスクの使用レジスタ数の絞り込み

これらの手法と従来方式とをシミュレーションにより評価した結果，提案手法により以下について実現できることを確認した.

1. コンテキスト切り替えによるメモリアクセスの軽減
2. 1. に伴い，データキャッシュミスの軽減
3. デッドラインオーバータスク数の軽減
4. ユーザタスクの応答時間の短縮化

シミュレーション結果より，コンテキスト保持専用レジスタセット数は 3 が最も有効であった.
また，今回の提案手法が,

1. CPU が高負荷の場合
2. タスクセットの平均実行時間が短い場合

に有効であることが分かった.
提案手法により向上した点により，今後は以下のことについて実現可能となる.

- コンテキスト切り替えのオーバヘッド軽減で生じた余裕時間により，今までよりも，より多くのユーザタスクが実行可能
- 割り込みによる，設計者にとって予想外のデッドラインオーバーを軽減

8.2 今後の課題

- 組み込み向けの CPU についても 2 次キャッシュを備えた CPU が普及しているので，今後はキャッシュについても 2 次キャッシュを搭載したことを前提で研究を進める必要がある．2 次キャッシュにより，キャッシュミスペナルティについて今までのシミュレーション時と比べて変化する (1 次キャッシュでミスの場合でも，2 次キャッシュでヒットし，キャッシュミスペナルティが軽減されることがある)

- カーネルタスクについて，今回は，タスクの起動要求とスケジューリングの 2 点でシミュレーションを行った．しかし，μITRON3.0 では，例外処理など必要とされる機能が実装されているので，今後は実際の RTOS を意識して，例外などのカーネルタスクを盛りこんだ上で研究を進める必要がある．シミュレーション実行時に，ランダムなタイミングでユーザタスクの起動要求以外のトラップを起こすことで，提案手法と Casablanca のトラップに対する優位性を比較することができる (提案手法では，内部トラップは 1 番レジスタセット，外部トラップは 7 番レジスタセットでのみ実行させていた．提案手法下で，外部トラップ実行中に，実行中の外部トラップより割り込みレベルの高い外部トラップが起きた場合，割り込みレベルの低い割り込みのコンテキストをメモリへ退避させなければならない．同様の場合，Casablanca ではレジスタセットを切り替えるのみで切り替えは済む．)．様々な内部トラップ，外部トラップが起こる状況下で，提案手法が Casablanca に比べてトータルでどれだけの優位性があるか判定することで，実システムへ提案手法を適用させるべきかどうか分かるはずである

- 現在，コンテキスト保持専用レジスタセットへのコンテキストの格納方式はファーストインの方式をとっている．この方式では，優先度が高いタスクを周期タスク，優先度の低いタスクを非周期タスクとしたとき，CPU 負荷率が高いタスクセットになる程，周期タスクの起動要求が頻繁に起こり，非周期タスクは中断されたままの状態が続く．中断された非周期タスクはフレーム内に残れたままになり，これによって，周期タスク間でコンテキスト切り替えを行うときに，コンテキストを退避させるだけの十分な空きフレームが無くなり，結果としてメモリアクセスによる従来のコンテキストを行うことになる．(図 6.21) フレーム内の優先度の低いタスクをメモリに追い出せば，周期タスク間でコンテキスト切り替えを行うときに，従来のコンテキストを行う回数は減少し，コンテキスト切り替えのオーバヘッドが更に軽減される．今後は，優先度の高いタスクのコンテキストをフレーム内に退避させるときに十分な空きフレームが無ければ，全フレーム内にあるタスクのコンテキストの内，優先度の一番低いタスクのコンテキストをメモリへ退避させて空きフレームを作り，優先度の高いタスクのコンテキストをフレーム内に格納する方式に偏光子，提案手法の効果をより一層向上させる (図 8.1 に例を示す)

- コンテキスト保持専用レジスタの管理をメモリ上にフレーム管理テーブルを設けた

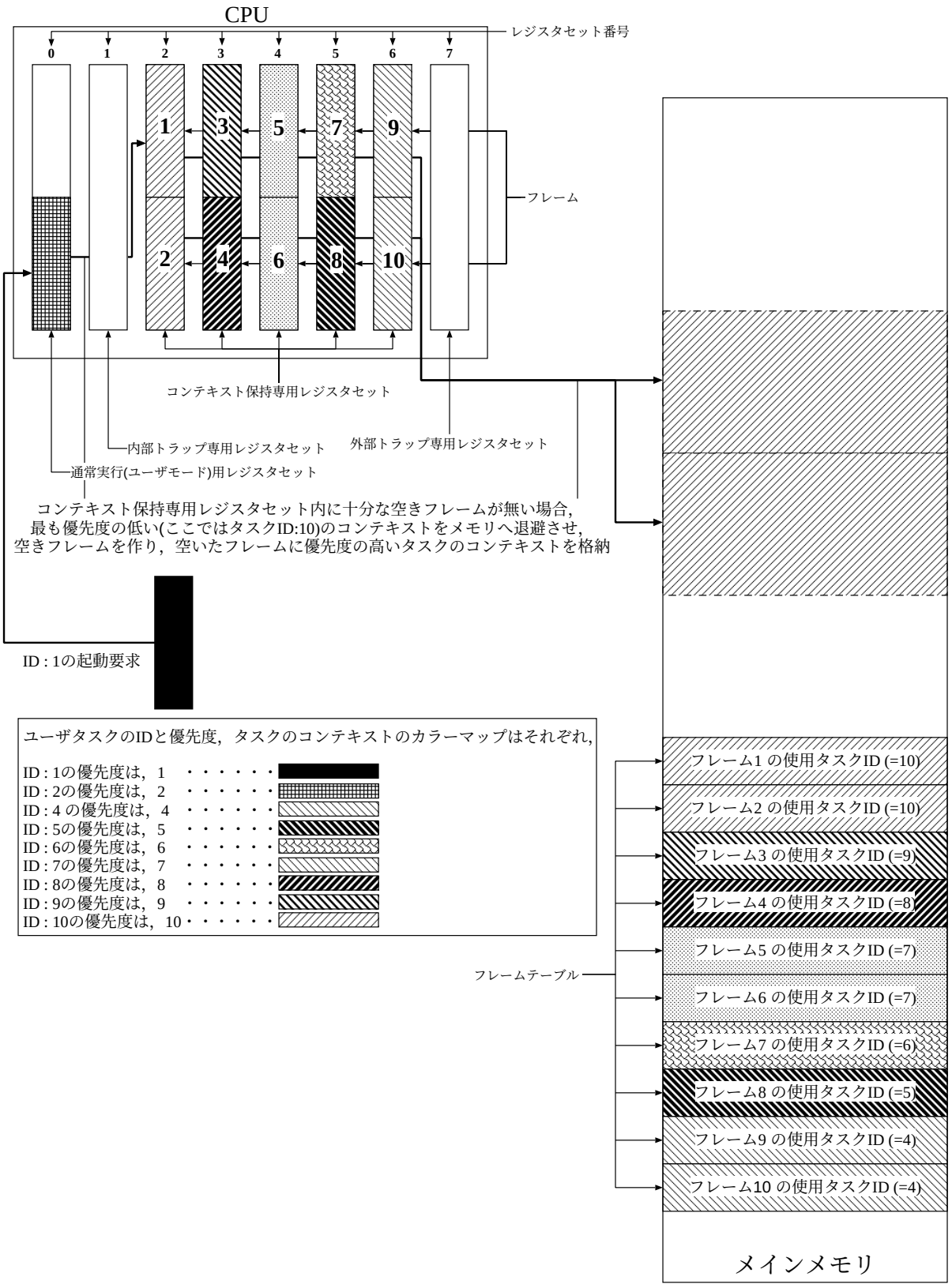

図 8.1: 優先度の低いタスクのコンテキストをフレームからメモリへ退避させて空きフレームを作る例

ことで，フレーム管理の際にメモリアクセスとデータキャッシュミスがあった．これによって，提案手法によるメモリアクセス減少とデータキャッシュミス減少の効果が少なくなった．今後は，フレーム管理専用のレジスタセットやフレーム管理専用レジスタを設ける方法により，フレーム管理時のメモリアクセスによるオーバヘッドを無くす必要がある．実現方法としては，

1. コンテキスト保持専用レジスタセットの 1 つをフレーム管理レジスタセットとして，フレーム管理レジスタセット内にフレーム管理テーブルを設ける (図 8.2)
2. タスクの使用レジスタ数の単位は 16 と 32 のため，1bit の情報で済む (1 のとき 32 本，0 のとき 16 本とする)．これにより，残ったビットは自由に使うことができる．また，全てのコンテキスト保持専用レジスタセットの%r31 には，常にタスクの使用レジスタ数が書き込まれる．%r31 の最上位ビットに使用レジスタ数を書き，残ったビットでフレームを管理する (例えば，2 番レジスタセットの%r31 では，フレーム 1 と 2 を管理)(図 8.3)

がある．

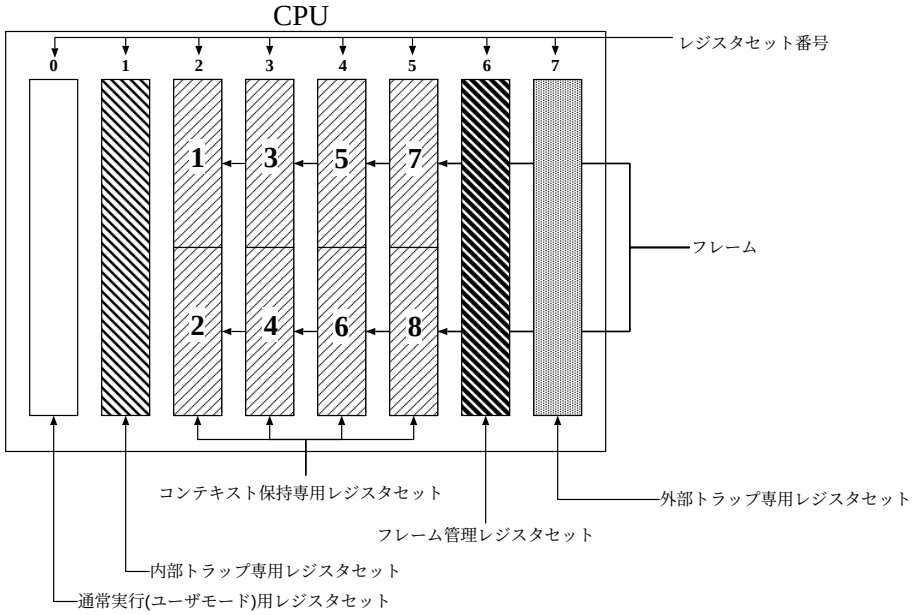

図 8.2: フレーム管理レジスタセットによるフレーム管理

- データキャッシュ，命令キャッシュを，それぞれ無し (Always Miss)，Always Hit にして従来方式のコンテキスト切り替えと提案手法でのコンテキスト切り替えを比較する，これにより，

  1. 命令キャッシュを Always Hit にし，データキャッシュを無しにしたとき，提案手法によるコンテキスト切り替えの向上が明確になって現れる (従来のコンテ

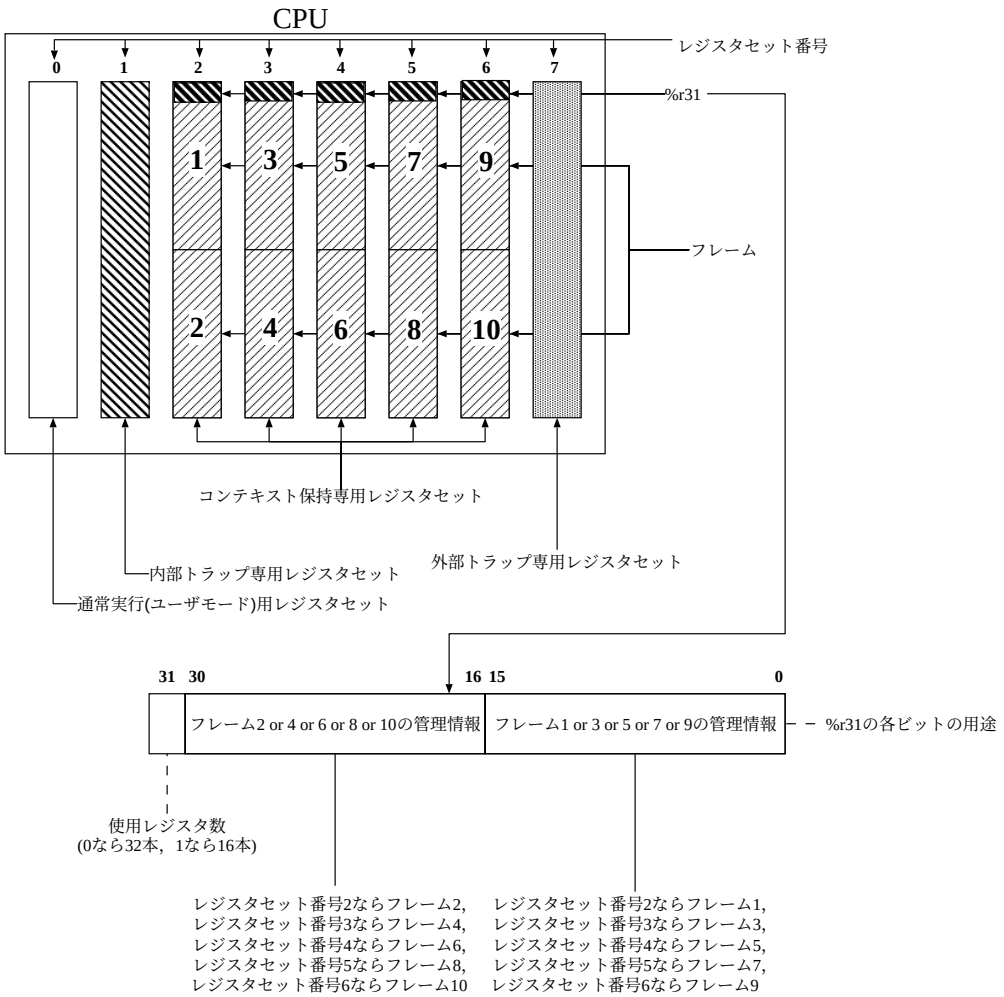

図 8.3: コンテキスト保持専用レジスタセットの%r31 によるフレーム管理

キスト切り替えの方法では，コンテキスト切り替え時にデータキャッシュミスが全て起こる．提案手法では，コンテキスト切り替え時にデータキャッシュミスが起こらない (タスクのコンテキストを格納できるだけの空きフレームがあるとき)．また，使用レジスタ数の絞り込みにより，従来のコンテキスト切り替え方法の場合，絞り込みを行うときと行わないときで，データキャッシュミスの回数が変わる (絞り込みによりコンテキスト切り替え時のデータキャッシュミス回数が半減する)．この方式では，命令キャッシュミスが無いため，コードサイズの増大による命令キャッシュミスの増大 (表 6.37，表 6.38) は無い)

2. 命令キャッシュ，データキャッシュともに Always Hit とする．これにより，従来のコンテキスト切り替えの方法と提案手法との比較で，提案手法によるコンテキスト切り替え時の命令数の減少による効果が分かる (従来のコンテキスト切り替え方法では，命令キャッシュミス，データキャッシュミスが起こらず，コンテキスト切り替えは命令数分の時間を要するだけで済む．提案手法でもコンテキスト切り替えは命令数のみで済むので，それぞれのコンテキスト切り替え方式で実行したとき，どれだけ命令数が減ったかが明らかになる)
3. 命令キャッシュをシミュレータの仕様にし，データキャッシュを Always Hit にしたとき，提案手法でのコードサイズ増加による命令キャッシュミスの増大が分かる (従来のコンテキスト切り替え方法では，データキャッシュミスが起こらず，コンテキスト切り替えは命令数分の時間に，命令キャッシュミスのオーバヘッドを加えた時間だけで済む．提案手法でもコンテキスト切り替えでは命令数分の時間に，命令キャッシュミスのオーバヘッドを加えた時間で済むが，提案手法では従来のコンテキスト切り替えを行うときに比べ，コードサイズが増大しているので，従来のコンテキスト切り替えに比べて命令キャッシュミスが増大する (図 6.18，表 6.37，表 6.38))

が分かる，

今後は，以上のことをシミュレータに盛りこんで，シミュレーションを行い，実システム上での提案手法の優位性を明らかにし，実際のリアルタイムシステムでの提案手法の実現を達成したい．

謝辞

本研究を遂行するにあたり，終始懇切丁寧に御指導を賜わりました田中清史助教授に心から深く感謝するとともに，ここに御礼申し上げます.

貴重な御助言，御討論を頂きました日比野靖教授，井口寧助教授に深く御礼を申し上げます.

また，研究のについての御助言だけではなく，日常の相談にも親身になって御答え下さった田中・日比野両研究室の皆様に厚く御礼を申し上げます.

最後に，今日まで 26 年間私を支えて下さった両親に深く感謝の意を示します.

参考文献

[1] Jeffrey C. Mogul and Anita Borg: The Effect of Context Switches on Cache Performance. ACM ASPLOS, 1991.

[2] 田中 清史，松本 尚，平木 敬: ”Casablanca: 実時間処理 RISC コアの設計と実装．” 情報処理学会研究報告，ARC Vol.99, No.100, pp.51-56，November 1999

[3] Anant Agarwal, Beng-Hong Lim, et.al: APRIL: A Processor Architecture for Multiprocessing. Proceedings of the 17th Annual International Symposium on Computer Architecture, pp.104-114, May 1990.

[4] Intel Corporation : Intel Technology Journal Volume 6 Issue 1 ”Hyper-Threading Technology” February 2002

[5] Joel M. Tendler, Steve Dodson, et.al: Power4 System Microarchitecture. Technical White Paper IBM Server Group, October 2001.

[6] http://www.mips.com/ProductCatalog/P_MIPS3224KFamily/24K.pdf

[7] SPARC International Inc.: The SPARC Architecture Manual Version 8, Prentice-Hall Inc. 1992

[8] http://gcc.gnu.org/

[9] Richard P. Paul : SPARC Architecture, Assmbly Language Programing, and C, Prentice-Hall Inc., 1994

[10] 坂村 健，トロン協会 : μITRON3.0 標準ハンドブック，パーソナルメディア，1993

[11] Jun-ichi Ito, Takumi Nakano, Yoshinori Takeuchi, Masaharu Imai: Effectiveness of a High Speed Context Switching Method Using Register Bank. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, Vol. E81-A, No.12, pp.2661-2667, December 1998.