大阪学院大学　人文自然論叢 ＜論　説＞
第79-80号　2020年３月

組込 CPU における割込・分岐処理の高速化に関する研究

岩　出　秀　平

Study of High-speed Embedded CPU for EIT Processing

Shuhei Iwade

【Abstract】

This study is concerned with the high-speed operation technology of embedded CPU for EIT processing. EIT is the abbreviation of Exception, Interrupt and Trap. The aim of the high-speed technology is the reduction of the number of data transfer instructions which are related to saving or setting data. In order to achieve the purpose, three technologies are developed. First, BPC file which includes plural BPCs is developed. BPC is the return address register and the abbreviation of Backup Program Counter. Secondly, the register file bank is introduced. A register bank consists of plural register files and each register file includes 2^n registers. These hardware can be controlled by a pointer which is generated by the file pointer developed. As a result, data transfer instructions for saving and setting both BPC and registers are reduced to zero. Additionally, the global registers are designed in order to transfer data before and after a branch interaction. The novel CPU which is equipped with these three hardware has been simulated by HDL simulator and can be assured the normal operation. Further the logic synthesis has been carried out with QuartusII for the novel CPU and the normal CPU. As a result, the novel CPU is proved to be the available for embedded application.

研究概要

本研究はリアルタイムシステムにおける割込処理に対応する組込用 CPU の高速化に関するものである．高速化の狙いは割込処理時に必要となるレジスタ等の退避や復帰処理に係る命令の削減である．そのため割込処理プログラムからの戻り先アドレスを保存する BPC（Backup Program Counter）レジスタを複数個保存可能な BPC ファイルを開発し，レジスタの集合体であるレジスタ・ファイルを多バンク化し，これらのハードウェアをポインタで制御可能にした．そして割込や復帰に対応してポインタを発生するファイル・ポインタを開発した．この結果，割込や復帰時の BPC やレジスタの退避・復帰のための命令数を 0 にすることに成功した．また分岐前プログラムと分岐後プログラムとの間でデータをやり取りする機能を付加するため，このポインタを利用したグローバル・レジスタも考案した．以上のハードウェアを搭載した新方式 CPU に対して HDL シミュレーションを実施し，全ての命令の正常動作を確認した．最後に QuartusII で新方式 CPU と旧方式 CPU の論理合成をおこない，論理合成レポートを比較した結果，バンク化によるレジスタ数の増加は想定を下回り且つ遅延時間に変化がないことが判明し，新方式 CPU がリアルタイムシステムに有効であることが分かった．

1．はじめに

CPU における割り込み（以下割込）にはタイマやセンサ等，外部ハードウェアから CPU に割込をかける外部割込とプログラム中の割込命令により自ら割込をかけるソフトウェア割込があり，割込処理後に元のプログラムに復帰する．

サブルーチン処理に使われる分岐命令は，分岐後に元のプログラムに復帰する機能をもつリンク付き分岐命令であり，ハードウェア構成の観点から割込処理と同一である．

割込・分岐命令による割込・分岐処理終了後に元のプログラムに復帰して処理を継続するには以下の条件が必要である．

① 命令メモリの戻り先アドレスが保存されていて，そのアドレスに分岐できること．

② レジスタ・ファイルのデータが，割込や分岐が起きる前の値になっていること．

戻り先アドレスは，割込や分岐を起こした命令の次の命令のアドレスで BPC というレジスタに保存する．レジスタ・ファイルは，２のべき乗個のレジスタから成るレジスタの集合体で一般的には16個，32個または64個等である．上記①②の条件を満たすためには，BPC やレジスタ・ファイル内の全レジスタの値を割込や分岐処理をする前にどこかに退避し，戻るときに復帰させなければならない．

これを実現するために従来の CPU ではデータメモリ内にスタック領域を設けて，割込や分岐が起こったときには BPC とレジスタ・ファイルの値をスタックに退避（ストア）

させ，戻るときにはスタックから復帰（ロード）させている．例えば三菱電機の M32R プロセッサでは，汎用で使われる15番レジスタ R15をスタックポインタに固定し，R15のデクリメントをハードウェアで自動的に行って ST Rsrc1,@-R15命令を繰り返すことにより R15番地のスタックへの退避を，また R15のインクリメントをハードウェアで自動的に行って LD Rdest,@R15+ 命令を繰り返すことにより R15番地のスタックからの復帰を行っている[1]．MIPS Technologies 社の MIPS プロセッサでは，スタックポインタ :sp を汎用レジスタとは別に備え，sw $s0 0（$sp), sw $s1 4（$sp), sw $s2 8（$sp）・・・と固定した SP に相対アドレス（0や4や8・・）を命令のオペランドで与えることにより，それぞれの番地のスタックへの退避を，復帰させる場合は，lw $s0 0（$sp), lw $s1 4（$sp), lw $s2 8（$sp）・・・のように sw と同様に命令のオペランドで与えることにより，それぞれの番地のスタックからの復帰を実現している[2]．スタック技術は1980年頃に提案され現在でもスタックに関する研究は行われている．[3]-[11]

具体的には割込・分岐処理前に，例えば16個のレジスタから成るレジスタ・ファイルの場合，BPC と合わせて17個のデータをスタック領域にストアし，処理が終わると同じく17個のデータをスタックからロードしなければならない．即ち１つの割込または分岐に対して（割込や分岐処理と無関係な）計34個のレジスタとデータメモリ間の転送命令を実行する必要がある．リアルタイム組込 CPU では割込や分岐の回数が多く，次の割込まで前割込処理が終わらず次の割込を取りこぼすことが問題となっているので，スタック方式を適用すると本来の割込処理命令に加えて上記復帰・退避命令のために全体の命令数が増加し，割込処理速度を更に低下させることになる．特にリアルタイム処理に使われる比較的小規模なマイクロプロセッサではコストや消費電力の制約があるため最先端の超高速デバイスが使えず，上記 M32R や MIPS も含めて命令数の増加による処理速度の低下をデバイスで吸収することは難しい．

本研究では，スタック方式の弱点を改善するため退避や復帰時の転送命令を使用せず割込・分岐命令で BPC やレジスタ・ファイルを保存できる新方式のハードウェア・アーキテクチャを考案し，16bit-CPU に実装して動作を確認した．また同規模のスタック方式 CPU も設計して QuartusII で論理合成を行い，論理合成結果の比較を行った．

以下では従来方式であるスタックによる退避・復帰方式の動作について説明した後に新方式ハードウェア・アーキテクチャ，それを実装した新方式 CPU 回路の設計および検証結果について述べる．

2. スタック方式による従来の割込・分岐処理技術

表2.1に従来型 CPU（以下スタック方式 CPU）の命令セットを示す．表2.1において，塗りつぶしで強調されている命令や機能は，新方式 CPU にはないスタック方式 CPU の

命令や機能である．表2.1で強調されていない命令や命令表記は，新方式 CPU と同じ命令または命令表記である．

表2.2にスタック方式 CPU 回路を実行順に５段に分割したパイプラインを示す．表2.1の各命令は，表2.2のパイプラインに従って順に実行される．例えば n 番目の命令が D ステージに入ったとき，n＋1 番目の命令は F ステージに投入される．

表2.1で強調されている命令について説明する．

表2.1　スタック方式 CPU 命令セット

命令分類	命令表記	命令機能
演算命令	ADD Rdest Rsrc	Rdest ← Rdest + Rsrc
	SUB Rdest Rsrc	Rdest ← Rdest - Rsrc
	AND Rdest Rsrc	Rdest ← Rdest & Rsrc
	OR Rdest Rsrc	Rdest ← Rdest | Rsrc
転送命令	LD Rdest @(Rsrc disp4)	Rdest ← M[Rsrc + $(disp4)_{16}$][7:0]]
	LDI Rdest imm9	Rdest ← $(imm8)_{16}$
	POP Rdest	Rdest ← M[SP]; SP++
	LDbpc	BPC ← M[SP]; SP++
	ST Rdest @(Rsrc disp4)	M[Rsrc + $(disp4)_{16}$][7:0]] ← Rdest
	PUSH Rdest	SP--; M[SP]←Rdest
	STbpc	SP--; M[SP]←BPC
分岐命令	BEQ Rsrc1 Rsrc2 pcdisp4	if(Rsrc1==Rsrc2) PC ← PC + $(pcdisp4)_{12}$
	BRA pcdrct12	PC ← pcdrct12
割込命令	TRAP pcdrct3	BPC←PC[TRAP]+1; BRA pcdrct3; 次命令=NOP
	RTE	PC ← BPC
	NOP	No Operation

表2.2　スタック方式 CPU のパイプライン

ステージ名	記号	内容
命令フェッチ	F(instruction Fetch)	命令取出;BPC読出書込
命令デコード	D(instruction Decode)	命令解読;ｱﾄﾞﾚｽ/ﾃﾞｰﾀ準備;次ｱﾄﾞﾚｽ設定
命令実行	E(Execution)	演算実行;スタックポインタ生成
メモリ・アクセス	M(Memory access)	データメモリ読出書込
ライトバック	W(Write back)	レジスタへのデータ書込;BPC←戻り番地

2.1 スタック関連命令の機能

2.1.1 TRAP 命令（ソフトウェア割込）

TRAP 命令機能で，PC は発行した命令のアドレスを保持するレジスタである．PC［TRAP］は TRAP 命令のアドレス，PC［TRAP］+1は TRAP 命令の次命令のアドレスである．BRA pcdrct3は分岐命令の BRA 命令で，PC ← pcdrct3のことであるので次の clk が入ると

pcdrct3番地の命令が実行される．次命令 =NOP は F ステージに読み出された TRAP 命令の次命令を無効にすることである．

TRAP pcdrct3は，表2.2の D ステージで解読され，以下の順に処理が行われる．

① F ステージで TRAP 命令の次命令のアドレス：PC[TRAP]+1を BPC に保存する．

② D ステージで TRAP 処理への分岐命令が格納されている pcdrct3番地へ BRA pcdrct3により無条件分岐させるため，pcdrct3番地を次アドレスとして設定する．

③ TRAP 命令が D ステージに移行したため F ステージに発行された TRAP 命令の次命令を無効化する（NOP 命令にする）．

2.1.2 RTE 命令（割込・分岐からの復帰）

RTE 命令機能で，PC ← BPC は，戻り先アドレスを PC に入力することを示す．

RTE 命令は，表2.2の D ステージで解読され，F ステージに対して戻り先アドレスが格納されている BPC を PC に入力する指示を出し，次の clk で戻り先アドレスの命令（TRAP 命令の次の命令）が発行され割込処理プログラムから元の実行プログラムに戻る．

2.1.3 PUSH 命令（スタックへのレジスタ退避）

PUSH 命令機能で，SP はスタックポインタで，スタック領域のアドレスを保持するレジスタである．SP-- は，SP を 1 だけデクリメントすることを示す．M はデータメモリで，M[SP]は SP 番地のデータである．M[SP]← Rdest は Rdest の値をデータメモリの SP 番地に書込むことを示す．Rdest の R はレジスタで dest はレジスタ番号（0～15）である．

PUSH Rdest は，表2.2の D ステージで解読され，以下の順に処理が行われる．

① E ステージで SP を 1 だけデクリメントする．

② M ステージでデータメモリの SP 番地に Rdest を保存する．

③ 例えば，0≦ dest ≦15の場合，R0～ R15の各々について上記①→②を繰り返す．

2.1.4 STbpc 命令（スタックへの BPC 退避）

STbpc は，PUSH と同じく表2.2の D ステージで解読され，E ステージで SP をデクリメントして，M ステージでデータメモリの SP 番地に BPC を退避させる．

2.1.5 POP 命令（スタックからのレジスタ復帰）

POP 命令機能で，SP++ は，SP を 1 だけインクリメントすることを示す．

POP Rdest は，表2.2の D ステージで解読され，以下の順に処理が行われる．

① M ステージでデータメモリの SP 番地のデータを Rdest に保存する．

② E ステージで SP を 1 だけインクリメントする．

③ 例えば，0≦ dest ≦15の場合，R0～ R15の各々について上記①→②を繰り返す．但

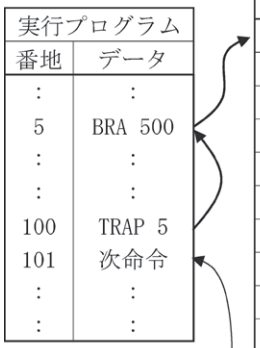
し PUSH 命令が R0から R15までレジスタ番号の昇順に退避させていた場合，POP 命令は R15から R0までレジスタ番号の降順に復帰させることになる．

2.1.6 LDbpc 命令（スタックからの BPC 復帰）

LDbpc は，POP と同じく表2.2の D ステージで解読され，M ステージでデータメモリの SP 番地のデータを BPC に復帰させた後に E ステージで SP をインクリメントする．

2.2 スタック方式による割込処理

スタック方式 CPU の TRAP 命令による割込処理の例を図2.1に示す．図2.1において，実行プログラムの100番地の TRAP 5命令が発行されると 5 番地に分岐し，5 番地の BRA 500命令により500番地に分岐する．500番地の STbpc 命令が BPC をスタック領域に，PUSH Rn（n=0～15）命令が Rn レジスタの内容をスタック領域にそれぞれプッシュした後，命令 A からの割込処理が開始される．命令 Z で割込処理が終了すると，POP R15からレジスタ番号の降順にスタックの内容を Rn レジスタに，LDbpc 命令がスタック領域から BPC にそれぞれポップした後，RTE 命令で割込処理からのリターンが実行されて元の実行プログラムの101番地の次命令に戻る．このようにスタックでは最後に退避したものを最初に復帰させるので，スタックは Last In/First Out の LIFO メモリである．



割込処理プログラム 番地	命令	内容	ジョブ
500	STbpc	BPCをスタック(m-1番地)に書込	プッシュ（退避）
501	PUSH R0	R0 をスタック(m-2番地)に書込	
502	PUSH R1	R1 をスタック(m-3番地)に書込	
:	: :	:	
:	: :	:	
515	PUSH R14	R14をスタック(m-16番地)に書込	
516	PUSH R15	R15をスタック(m-17番地)に書込	
517	命令A	割込処理開始	割込処理
:	:	:	
:	:	:	
:	:	:	
549	命令Z	割込処理終了	
550	POP R15	スタック(m-17番地)からR15に読出	ポップ（復帰）
551	POP R14	スタック(m-16番地)からR14に読出	
:	: :	:	
:	: :	:	
564	POP R1	スタック(m-3番地)からR1に読出	
565	POP R0	スタック(m-2番地)からR0に読出	
576	LDbpc BPC	スタック(m-1番地)からBPCに読出	
577	RTE	割込処理からのリターン	

図2.1　スタック方式 CPU の TRAP 命令による割込処理（SP 初期値 =m の場合）

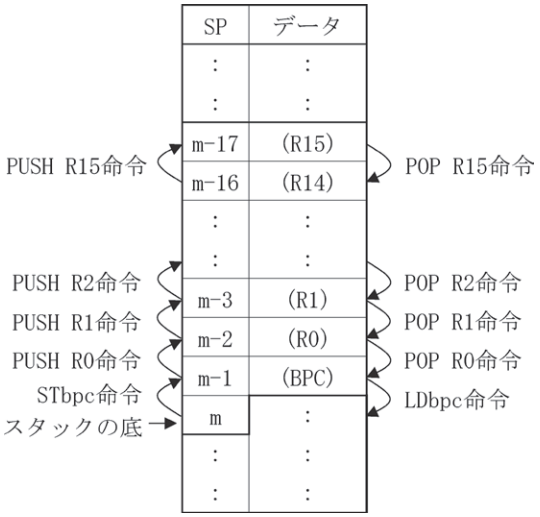


図2.2　図2.1の割込処理におけるスタック領域

図2.2は，図2.1の割込処理においてスタックの底が m 番地のときのデータメモリの状態である．図2.1の STbpc 命令により図2.2において SP=m であった SP が 1 だけデクリメントされ，m-1番地に BPC の内容が退避される．次の PUSH R0命令により SP が更に 1 だけデクリメントされ，m-2番地に R0の内容が退避される．このように BPC と R0～ R15までの全てのレジスタの内容が退避された結果 SP=m-17番地になっている．

割込処理（命令 A ～命令 Z）が終了し，元の実行プログラムに戻るために図2.1における POP R15命令が実行されると，SP=m-17番地のデータが読み出され R15に書き込まれた後，SP は 1 だけインクリメントされ，SP=m-16番地となって次の POP 命令を待つ．このようにして m-1番地にある最後のデータ BPC が復帰されると SP は 1 だけインクリメントされて SP= m 番地に戻る．

3. 従来技術の問題点

前項2. で述べたことをまとめると，従来のスタック方式には以下の欠点がある．

① 割込・分岐が発生する度に戻り先アドレスやレジスタ群の退避・復帰命令を実行しなければならないため，ジョブを実行するための命令数が増加し，それによって処理速度が劣化する．

② 退避や復帰にはスタックポインタの管理が伴うので，通常のロード命令やストア命令では対応できず，スタック専用のストア命令（STbpc, PUSH）とロード命令（LDbpc, POP）を設けなければならない．リアルタイム処理対応の CPU では命令長が16ビットと短い場合が多く，命令の種類を決めるオペコードに割り当てるビッ

ト数が限られていて命令を増やせない場合が多い．従って４命令の追加により他の有用な命令を割愛せざるを得ない場合にはCPUの機能全体に悪影響を与える．

以上の問題を解決するため，スタック専用のストア命令やロード命令を必要としない回路方式を内蔵したCPUを設計した．

4. 新方式ハードウェア・アーキテクチャ

新方式CPUに適用された新方式ハードウェア・アーキテクチャについて述べる．

表4.1に新方式CPUの命令セットを示す．表4.1において，塗りつぶしで強調された命令や機能は，スタック方式CPUにはなく新方式ハードウェア・アーキテクチャを実現する新規命令（BL命令），または命令表記は同じでもスタック方式CPUと異なる新規機能に変更した命令（TRAP命令, RTE命令）である．

表4.1の横ストライプで強調されたLDRR, LDRG, LDGR命令は, 新方式ハードウェア・アーキテクチャ導入によるCPUのプログラミングを容易化するために新規に追加した副次的な命令でレジスタ間コピー命令である．表4.1で強調されていない命令は，スタック方式CPUと同じ命令である．$(imm8)_{16}$は８ビットのimmを16ビットに，$(disp4)_{16}$は４ビットのdispを16ビットに，$(pcdisp4)_{12}$は４ビットのpcdispを12ビットに，それぞれ符号拡張したものである．

以下では新方式ハードウェア・アーキテクチャに特化したパイプライン，新方式ハードウェア・アーキテクチャを実現する命令（表2.1で強調されている命令）の機能，それらの割込処理，そしてレジスタ間コピー命令の機能について述べる．

表4.1　新方式CPU命令セット

命令分類	命令表記	命令機能
演算命令	ADD Rdest Rsrc	Rdest ← Rdest + Rsrc
	SUB Rdest Rsrc	Rdest ← Rdest − Rsrc
	AND Rdest Rsrc	Rdest ← Rdest & Rsrc
	OR Rdest Rsrc	Rdest ← Rdest | Rsrc
転送命令	LD Rdest @(Rsrc disp4)	Rdest ← M[Rsrc + $(disp4)_{16}$][7:0]]
	LDI Rdest imm8	Rdest ← $(imm8)_{16}$
	LDRR Rdest Rsrc	Rdest ← Rsrc
	LDRG Rdest Gsrc	Rdest ← GRsrc
	LDGR Gdest Rsrc	GRdest ← Rsrc
	ST Rdest @(Rsrc disp4)	M[Rsrc + $(disp4)_{16}$][7:0]] ← Rdest
分岐命令	BEQ Rsrc1 Rsrc2 pcdisp4	if(Rsrc1==Rsrc2)　PC ← PC + $(pcdisp4)_{12}$
	BRA pcdrct12	PC ← pcdrct12
	BL pcdrct12	rfp←rfp+1;PC←pcdrct12;BPC[rfp]←PC+1;RF[rfp,n]
割込命令	TRAP pcdrct3	rfp←rfp+1;BRA pcdrct3;BPC[rfp]←PC[TRAP]+1;RF[rfp,n]
	RTE	PC ← BPC[rfp];rfp←rfp-1;RF[rfp,n]
	NOP	No Operation

4.1 新方式ハードウェア・アーキテクチャに特化したパイプライン

表4.2に新方式 CPU 回路を実行順に５段に分割したパイプラインを示す．表4.1の各命令は，表4.2のパイプラインに従って順に実行される．

表2.2に示すスタック方式 CPU のパイプラインでは，2.1.1項で述べたように割込・分岐命令の次に F ステージに投入された命令が，分岐先の命令ではなく，割込・分岐命令の次の命令になるという問題（制御ハザード）があり，F ステージの命令を NOP 化する必要があった．

しかし新方式 CPU において，表4.2に示すように通常 D ステージで行う無条件分岐命令の解読を F ステージで実施することにより制御ハザードを解消している．このことを図4.1と図4.2で説明する．

図4.1に分岐命令BLの解読をDステージで行う場合のパイプラインを示す. 図において，100番地の BL 1000命令は表4.1に示すようにリンク付き分岐命令で，戻り先アドレスである101を退避させて1000番地に分岐する命令である．BL 命令は F ステージで取り出され，D ステージで解読されて次に実行すべき命令のアドレス：1000番地が F ステージに送られる．しかしパイプライン制御では，BL 命令が D ステージに入ったときに自動的に101番地の ADD 命令が F ステージに投入される．このまま実行すると，本来戻ってきてから実行すべき101番地の ADD 命令が，BL 命令の次に実行すべき1000番地の SUB 命令の前に実行されてしまう．従って101番地の ADD 命令を F ステージで無効にする必要がある．この結果，図4.1から分かるように1000番地の SUB 命令発行は１ステージ遅れることになる．

新方式 CPU では上記問題を解決するため，図4.2に示すように分岐条件のない無分岐命

表4.2　新方式 CPU のパイプライン

ステージ名	記号	内容
命令フェッチ	F	命令取出;無条件分岐先ｱﾄﾞﾚｽ設定;rfpﾎﾟｲﾝﾀ設定;BPCnへ保存
命令デコード	D	命令解読;ｱﾄﾞﾚｽ/ﾃﾞｰﾀ準備;無条件分岐以外の次ｱﾄﾞﾚｽ設定
命令実行	E	演算実行
メモリ・アクセス	M	データメモリ読出書込
ライトバック	W	レジスタへのデータ書込

番地	命令	パイプライン						
100	BL 1000	F:BL取出	D:BL解読	E —	M —	W —		
101	ADD R3,R4		F:ADD取出	D —	E —	M —	W —	
1000	SUB R5,R2			F:SUB取出	D:SUB解読	E:SUB実行	M —	W:結果書込

図4.1　分岐命令解読を D ステージで行う場合のパイプライン

番地	命令	パイプライン					
100	BL 1000	F:BL取出/解読	D —	E —	M —	W —	
1000	SUB R5,R2		F:SUB取出	D:SUB解読	E:SUB実行	M —	W:結果書込

図4.2 分岐命令解読をFステージで行う場合のパイプライン

令に対して解読をFステージで行うようにした．図に示すようにBL命令はFステージで取り出されて且つ解読もされるため，1000番地がFステージ内で処理されて次に実行すべき命令のアドレスとなる． 従って101番地がFステージに投入されることはなく，BL命令の次に1000番地のSUB命令を発行することができるため，図4.1のように1ステージ分の遅れは生じない．このように分岐命令のFステージでの命令解読は，割込・分岐処理の高速化をより効果的なものにする．

4.2 新方式ハードウェア・アーキテクチャ関連命令機能

4.2.1 TRAP命令（ソフトウェア割込）

表4.1のTRAP命令機能で，rfpはregister file pointerの略で0～8の値をとり，アドレスとして利用される．BPC[rfp]は，8本のBPCから構成されたファイルでBPCファイルとよぶ．rfpはBPCファイルのアドレスであるので，BPC[rfp]← PC[TRAP]+1は，TRAP命令の次命令のアドレスをBPCファイルのrfp番地に書込むことを示す．RF[rfp,n]（0≦ rfp ≦8, 0≦ n ≦15）は，2次元のレジスタ・ファイル・バンクで，rfpは0～8までのバンクアドレス，nは0～15までのレジスタ番号である．RF[rfp,n]は，rfpバンクのRnレジスタを示す．

TRAP pcdrct3は，表4.2のFステージで解読され，以下の順に処理が行われる．

① Fステージでrfpをインクリメントする．

② Fステージで TRAP処理への分岐命令が格納されている pcdrct 3番地へBRA pcdrct3により無条件分岐させるため，pcdrct3番地を次アドレスとして設定する．

③ FステージでBPCファイルのrfp番地にTRAP命令の次の命令（戻り先命令）の番地：PC[TRAP]+1を保存する．

④ Dステージでレジスタ・ファイル・バンク：RF[バンク番号 :rfp，レジスタ番号 :n]のバンク番号が，上記①で更新されたrfpに自動的に変更される．

以上の操作により，③のBPCの退避と④のレジスタの退避はTRAP命令の中（FステージとDステージ）で実行されるため，STbpcやPUSH等の退避命令は不要になる．

4.2.2 BL命令（リンク付き分岐）

表4.1のBL pcdrct12命令は，サブルーチン分岐に使われる命令で以下の順に処理が行われる．

① TRAP 命令と同じ.

② F ステージで pcdrct12番地から始まるサブルーチンプログラムに制御を移す.

③ TRAP 命令と同じ.

④ TRAP 命令と同じ.

TRAP 命令との違いは②だけで，TRAP 命令では pcdrct3番地を介して間接的に割込処理に分岐するのに対し，BL 命令では分岐処理の先頭番地である pcdrct12番地に直接分岐する.

以上の操作により，③の BPC の退避と④のレジスタの退避は BL 命令の中（F ステージと D ステージ）で実行されるため，STbpc や PUSH 等の退避命令は不要になる.

4.2.3 RTE 命令（割込・分岐処理からの復帰）

表4.1の RTE 命令は，表4.2の F ステージで解読され，以下の順に処理が行われる.

① F ステージで BPC ファイルの rfp 番地に保存されている戻り番地を PC に入力して，次の clk で元の実行プログラム中の TRAP 命令の次命令を発行させる.

② F ステージで rfp をデクリメントする.

③ D ステージで RF[バンク番号 :rfp, レジスタ番号 :n]のバンク番号が②で更新された rfp に自動的に変更される.

以上の操作により，①の BPC の復帰と③のレジスタの復帰は RTE 命令の中（F ステージと D ステージ）で実行されるため，LDbpc や POP 等の復帰命令は不要になる.

4.3 新方式ハードウェア・アーキテクチャによる割込処理

新方式 CPU の TRAP 命令による割込処理の例を図4.3に示す．図4.3において，実行プログラムの100番地の TRAP 5命令が発行されると５番地に分岐し，５番地の BRA 500命令により500番地に分岐する．新方式では BPC やレジスタの退避は必要ないので，500番地から直ぐに割込処理を開始することができる．割込処理が終了すると BPC やレジスタ

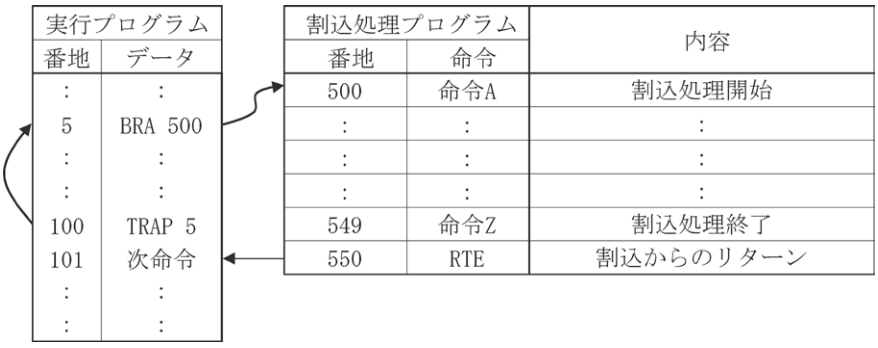


図4.3　新方式 CPU の TRAP 命令による割込処理

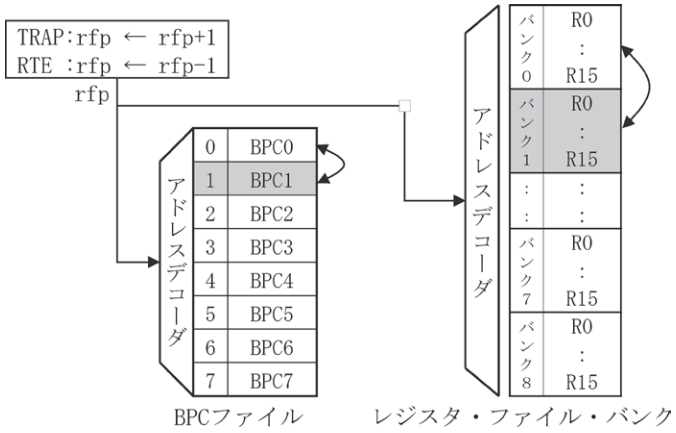


図4.4　新方式 CPU の割込処理における BPC およびレジスタの保護方式

への復帰は不要なので，割込処理が終了次第 RTE 命令で割込処理からのリターンが実行され元の実行プログラムの101番地の次命令に戻ることができる．

上記割込処理を可能とする BPC やレジスタの保護方式を図4.4に示す．

図4.4において，最初 rfp=0であったとする．その場合 BPC ファイルとレジスタ・ファイル・バンクのアドレスデコーダには 0 番地が入るので，BPC0とバンク 0 のレジスタ R0 ～ R15が使用される．

TRAP 命令による割込発生時，4.2.1①で述べたように rfp を 1 だけインクリメントするので，rfp=1となる．その結果，戻り先アドレスを BPC1に書込んで，割込処理プログラムはバンク 1 の R0～ R15を使用することになる．この操作は TRAP 命令として実施されるので，スタック方式で必要であった退避命令（STbpc や PUSH）は不要である．

割込処理が終了し, RTE 命令が実行されると4.2.3①で述べたように rfp=1のままなので，BPC1に保存した戻り番地を PC に送り元の実行プログラムに戻る．次に4.2.3②で述べたように rfp を 1 だけデクリメントするので，rfp=0となり，BPC0と実行プログラムで使用していたバンク 0 のレジスタ R0～ R15が利用可能となる．この操作は RTE 命令として実施されるので，スタック方式で必要であった復帰命令（LDbpc や POP）は不要である．

新方式 CPU に新規導入された BL 命令によるサブルーチン処理の例を図4.5に示す．BL 命令は TRAP 命令と異なり実行されると直接サブルーチンに分岐する. 図4.5において，実行プログラムの100番地の BL 500命令が発行されると直接500番地に分岐し，割込処理が終了すると RTE 命令で割込処理からのリターンが実行され元の実行プログラムの101番地の次命令が実行される．BL 命令における BPC やレジスタの保護方式は図4.4と同じである．

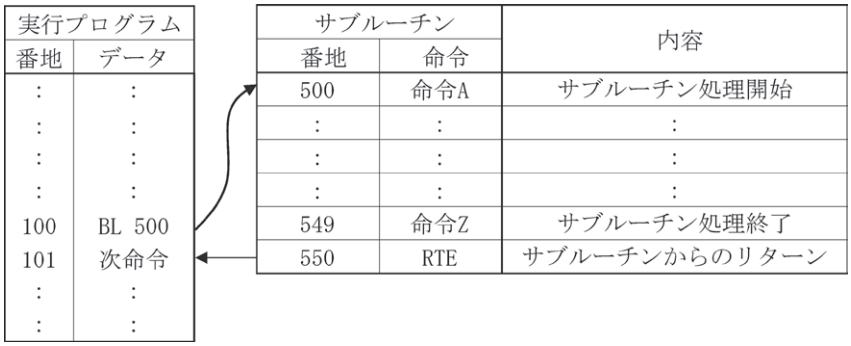


図4.5　新方式 CPU の BL 命令によるサブルーチン処理

4.4 レジスタ間転送命令（レジスタコピー）

図4.4のレジスタ・ファイル・バンク図に示すように，新方式 CPU では，割込・分岐命令によりバンクが変更されレジスタ R0～ R15のセットが強制的に変更されるので，実行プログラムで得たレジスタの内容を割込・分岐処理プログラムで利用するにはデータメモリに移しておく必要がある．（スタック方式の場合，R0～ R15はスタックに退避されるが内容は変更されておらず，割込・分岐処理プログラムも同じレジスタを使うので，自動的にレジスタが引き継がれる．）

そこで新方式 CPU では，図4.4に示すバンク８をグローバル・レジスタとし，バンク０からバンク７との間でレジスタデータのコピーを可能にした．これにより実行プログラムから割込処理やサブルーチンへのデータ引継ぎはバンク８のレジスタに保存することで可能となる．

4.4.1 LDRG Rdest Gsrc

Rdest は実行中のプログラムに割り当てられているバンクに属する dest 番地のレジスタ，GRsrc は，グローバル・レジスタの src 番地のレジスタであり，その機能は，GRsrc の内容を Rdest にコピーすることである．

4.4.2 LDGR Gdest Rsrc

実行中のプログラムに割り当てられているバンクに属する Rsrc の内容をグローバル・レジスタの dest 番地のレジスタにコピーする．

4.4.3 LDRR Rdest Rsrc

実行中のプログラムに割り当てられている同一バンク内でのレジスタ間コピーである．

この命令は，グローバル・レジスタを用いたデータ転送とは関係ないが，LDRG や LDGR 命令をサポートする回路の制御線の値を00にするだけで実現できるため命令セットに加えた．

5. 新方式 CPU 回路の設計

新方式 CPU 回路の設計について述べる．CPU 回路は表4.1の新方式 CPU 命令セットの命令機能に基づいて設計された.

表5.1に新方式 CPU の命令フォーマットを示す．命令長は全て16ビットである．命令の種類を表す opecode はビット15～ビット12までの４ビットであるが，LDRR, LDRG, LDGR に限っては空いているビット１～ビット０の領域を第２の opecode として使用している.

レジスタは R0～ R15の16本としたので，レジスタ番号である dest と src は４ビットである．未使用のビットは０とする.

disp は displacement の略で，アドレス間の距離を表す．imm は immediate の略で演算対象となる即値である．pcdisp は program counter displacement の略で，program counter 値（発行されている命令のアドレス）と分岐先アドレスとのアドレス間距離である．pcdrct は pc direct の略で分岐先の直接アドレスである．また disp4, imm8や pcdrct12の数字はビット数を示している.

設計手順を以下に示す.

手順１：各命令を実現する回路を RTL（Register Transfer Level）で設計.

手順２：回路図が似ている回路同士を統合して回路グループを設計し，最後に全ての回路グループを統合して CPU 全体回路を設計.

表5.1　新方式 CPU の命令フォーマット

bit番号→	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	opecode				dest				src				未使用			
ADD	0	0	0	1									0	0	0	0
SUB	0	0	1	0									0	0	0	0
AND	0	0	1	1									0	0	0	0
OR	0	1	0	0									0	0	0	0
	opecode				dest				src				disp4			
LD	1	0	0	0												
ST	1	0	0	1												
BEQ	1	0	1	0												
	opecode1				dest				src				未使用		opecode2	
LDRR	1	1	0	1									0	0	0	0
LDRG	1	1	0	1									0	0	0	1
LDGR	1	1	0	1									0	0	1	0
	opecode				未使用									pcdrct3		
TRAP	1	1	1	0	0	0	0	0	0	0	0	0	0			
	opecode				dest				imm8							
LDI	0	1	1	1												
	opecode				pcdrct12											
BRA	1	0	1	1												
BL	1	1	0	0												
	opecode				未使用											
RTE	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
NOP	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

手順３：CPU 全体回路を F, D, E, M, W の５段のパイプライン・ステージに分割.

手順４：各 RTL 回路を Verilog HDL で記述

以下では手順１～３で得られた回路について F ステージから順に説明する.

5.1 F ステージ回路

図5.1に新方式 CPU の F ステージ回路を示す．このステージにおける新技術は，通常の回路に bpc_file 回路，file_pointer 回路と fetch_ 解読器を追加し，BPC の退避を TRAP 命令と BL 命令で，BPC の復帰を RTE 命令で実現したことである.

図5.1において，灰色に塗りつぶした回路は，クロックに同期する回路である．楕円の塗りつぶし回路はパイプラインのための D ステージ入口の同期レジスタで楕円内の数字はビット数を示す．また太枠回路は，新型 CPU 回路にのみ存在する回路である.

program counter は，命令メモリのアドレスを保持するレジスタで，その出力 pcout は命令アドレスである.

命令メモリに命令アドレス pcout が与えられると，そのアドレスに存在する命令が inst

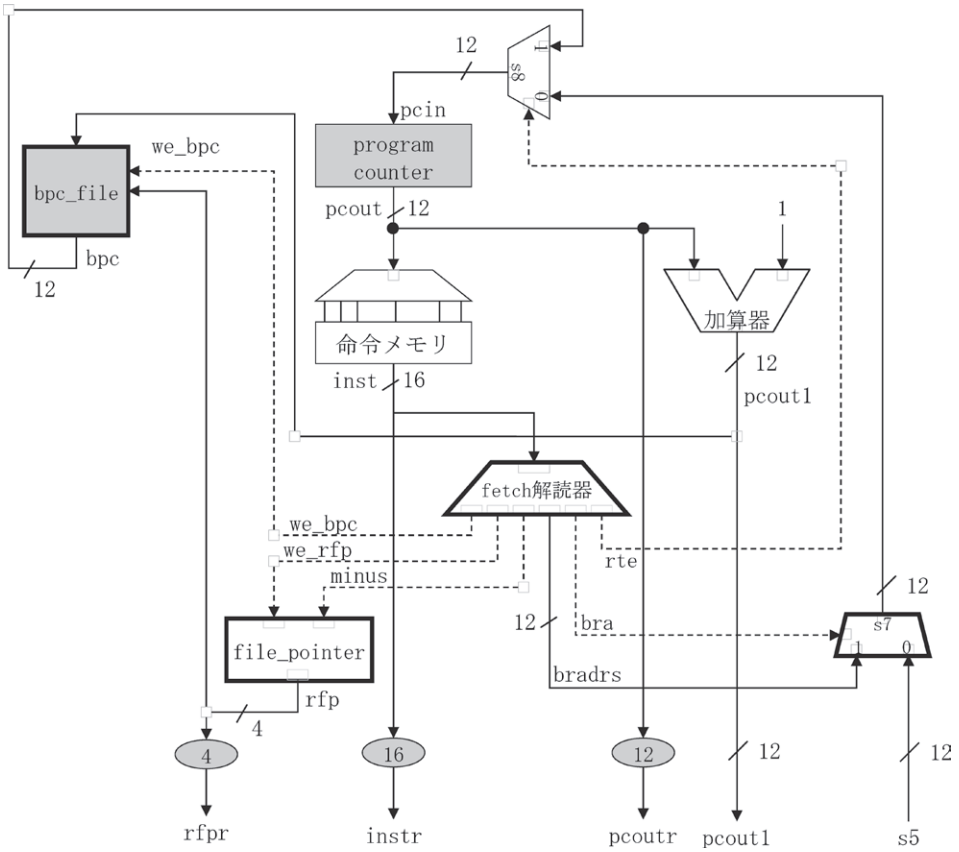


図5.1　新方式 CPU の F ステージ回路

に出力される．

加算器は，発行された命令のアドレス pcout に 1 だけ加算して，次に実行する命令のアドレス pcout1を計算する．

fetch 解読回路は，割込命令である TRAP，無条件分岐命令である BL，BRA，復帰命令である RTE を解読して制御線や分岐先アドレスを出力する回路である．

file_pointer は，we_rfp=1のときに動作する 4 ビットの rfp ポインタを出力する回路で，表4.1の TRAP 命令（割込）や BL 命令（リンク付き分岐）が発行されたときに 1 だけインクリメントされ，RTE 命令が実行されたときに 1 だけデクリメントされる．

bpc_file は，８個の12ビット BPC を退避させることができる同期メモリの一種で，TRAP，BL，RTE 命令のときに we_bpc=1となり動作する．各 BPC のアドレス0000～0111は rfp で与えられ，pcout1がデータとして書き込まれる．理由は，pcout1は発行中の命令の次の命令アドレス（即ち戻り先アドレス）だからである．s7や s8は12ビットのセレクタである．

F ステージ回路の動作について説明する．

表5.1の TRAP pcdrct3命令が inst に出力されると，fetch 解読器でデコードされ，we_bpc=we_rfp=1，minus=0，bra=1，rte=0，bradrs が $(\text{pcdrct3})_{12}$となるので，s7で $(\text{pcdrct3})_{12}$が選択され s8の 0 入力に入り，s8で 0 入力が選択され $(\text{pcdrct3})_{12}$が pcin となり TRAP 命令の次の clk で program counter から出力されて命令メモリに与えられ，$(\text{pcdrct3})_{12}$番地に分岐する．

このとき file_pointer に(we_rfp，minus)=(1,0)が入力されるので，rfp が 1 だけインクリメントされて bpc_file に送られ，we_bpc=1より pcout1がデータとして rfp 番地に書き込まれる．

以上の動作より，TRAP pcdrct3命令を実行すると，$(\text{pcdrct3})_{12}$番地に分岐するだけでなく次の clk で bpc_file において 1 だけインクリメントされたアドレスの BPC に pcout1が退避（保存）され，元の rfp 番地の BPC は消されずに保存される．即ち，スタック方式 CPU での STbpc 命令による BPC の退避が，新方式 CPU では TRAP pcdrct3命令の D ステージで実行されたことになる．

表5.1の BL pcdrct12命令の動作は上記 TRAP pcdrct3命令とほぼ同じである．両命令の違いは，BL 命令のオペランドである pcdrct12が元々12ビットなので TRAP pcdrct3のときのように fetch 解読器内で pcdrct3を12ビットの $(\text{pcdrct3})_{12}$に拡張する必要がないことである．

表5.1の RTE 命令が inst に出力されると，fetch 解読器でデコードされ，we_bpc=0，we_rfp=1，minus=1，bra=1，rte=1，bradrs=0となるので，s8で 1 入力が選択され，bpc_file に最後に保存された BPC が bpc から s8の 1 入力を通って pcin となり RTE 命令の次の clk で program counter から出力されて命令メモリに与えられ分岐前の戻り先アドレスに分岐する．

このとき file_pointer に(we_rfp, minus)=(1,1)が入力されるので，file_pointer 内で rfp が 1 だけデクリメントされるが内部で 1 clk 遅延用のレジスタを通すので直ぐに rfp に出力されることはない．次の clk でデクリメントされた rfp が bpc_file に送られ，RTE 命令実行時に bpc が保持していた戻り先アドレスの前に保存された戻り先アドレスが bpc に出力される．

以上の動作より，RTE 命令を実行すると，戻り先アドレスに分岐するだけでなく，次の clk で file_pointer から 1 だけデクリメントされた rfp が bpc_file に送られ，最後に保存された BPC の一つ前の BPC が非同期で bpc 出力に保持される．即ち，スタック方式 CPU での LDbpc 命令による BPC の復帰が，新方式 CPU では RTE 命令の D ステージで実行されたことになる．

表5.1の BRA pcdrct12が inst に出力されると，fetch 解読器でデコードされ，we_bpc=0，we_rfp=0，minus=0，bra=1，rte=0，bradrs=pcdrct12となるので，s7で pcdrct12が選択され s8の 0 入力に入り，s8で 0 入力が選択され pcdrct12が pcin となり BRA pcdrct12 命令の次の clk で program counter から出力されて命令メモリに与えられ，pcdrct12番地に分岐する．

このとき file_pointer では we_rfp=0より rfp は変化せず，bpc_file でも rfp が不変で we_bpc=0ならば bpc は変化しない．

表5.1の上記以外の全ての命令が inst に出力されると，fetch 解読器の出力が全て 0 になるので，file_pointer や bpc_file は変化せず，s7と s8で 0 入力が選択されるので D ステージからの s5が pcin に転送される．即ち次の命令アドレスは F ステージではなく D ステージで決められる．

5.1.1 BPC ファイル（bpc_file）

図5.2に BPC ファイルの構成図を示す．図において rst はリセット，clk はクロック，we_bpc はデータ書込許可，rfp は4bit のレジスタ・ファイル・ポインタで BPC ファイルのアドレス，pcout1は12bit の BPC 入力，bpc は12bit の BPC 出力である．内部には BPC0 ～ BPC7の 8 個のレジスタが格納されている．各 BPC の bit 幅が12bit である理由は，命令メモリのアドレス幅を12bit としたからである．BPC が 8 個にも関わらずアドレスである rfp の bit 数が4bit である理由は，rfp がレジスタ・ファイルの退避にも使用されるためで，BPC ファイルで有効なポインタビットは，rfp[2:0]の3bit である．

単体の動作について説明する．rst がかかると，BPC0～ BPC7までの全ての BPC レジスタが 0 にリセットされる．書込は clk 同期である．即ち clk= ↑前に we_bpc，rfp，pcout1 が与えられ，clk= ↑になった瞬間に pcout1にある BPC 値が rfp アドレスに書き込まれる．読出は clk 非同期であるので，rfp にアドレスを与えると clk に無関係に BPC 値が bpc に出力される．

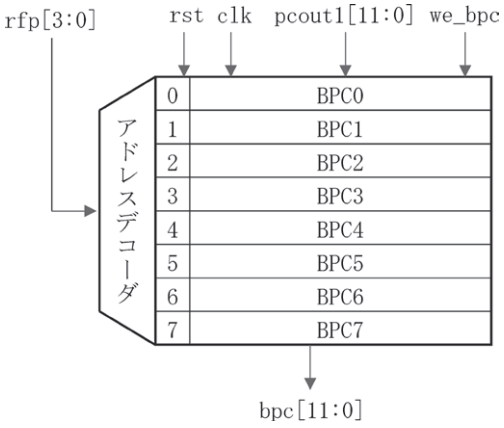


図5.2　BPC ファイルの構成図

次に CPU 中での BPC ファイルの動作について図5.3で説明する．図において BPC ファイルの信号は, rst ～ bpc[11:0]である．pcout は命令メモリから取り出した命令のアドレス, inst はその命令である．we_rfp と minus は file_pointer の出力である．また図中の数字は全て16進数である．

＜リセット時＞

最初に rst=0でリセットされると，rfp および bpc は 0 にリセットされ，pcout と inst より 9 番地の710B（表5.1：LDI R1 B）という命令が発行される．9 番地からスタートする理由は，リセットをかけたときに 9 番地からスタートするように設定したからである．このとき pcout1に与えられる BPC 値は 9 番地＋1より00A となっている．この理由は，A 番地の命令は発行された 9 番地の命令の次に実行される命令だからである．

＜ TRAP 命令実行＞

inst の 3 つ目：00B 番地の E007（表5.1：TRAP 7）命令は，ソフトウェア割込である．

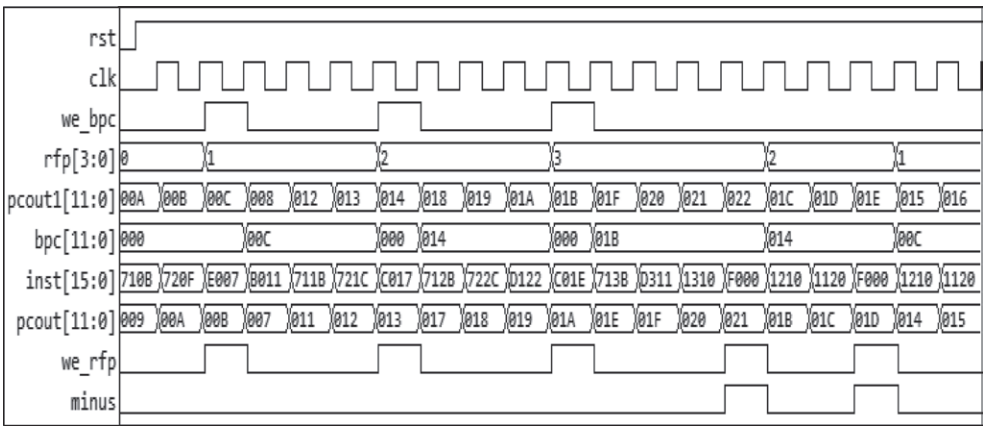


図5.3　BPC ファイルの動作

従って戻り先アドレスである（00B の次のアドレス）00C を BPC ファイルに退避させなくてはならない．E007が発行されたとき，we_bpc=1，pcout1=00C,（file_pointer 入力 we_rfp=1，minus=0より rfp が０→１）rfp=1であるので，clk= ↑となった瞬間に（rfp=）１番地に00C が書き込まれ，書き込まれた値00C は bpc に出力されるはずである．

図に示すように，E007が発行された次の clk= ↑で，bpc=00C となっている．これは BPC ファイルの（rfp=）１番地に戻り先アドレス00C が退避されたことを示している．

このように TRAP 命令は, 割込処理に分岐するだけでなく BPC の退避も実現している．

＜ BL 命令実行＞

TRAP 命令による割込処理（011番地～）に分岐して3clk 後に発行される７番目の命令：（pcout=）013番地の（inst=）C017（表5.1：BL 17）という命令は，リンク付き分岐命令であるので戻り先アドレスである（013の次のアドレス）014を BPC に退避させなければならない．C017が発行されたとき we_bpc=1， pcout1=014,（file_pointer 入力 we_rfp=1，minus=0より rfp が１→２）rfp=2であるので，clk= ↑となった瞬間に（rfp=）２番地に014が書き込まれ，書き込まれた値は bpc に出力されるはずである．

図に示すように，C017が発行された次の clk= ↑で，bpc=014となっている．これは BPC ファイルの（rfp=）２番地に戻り先アドレス014が退避されたことを示している．

このように BL 命令は，割込処理に分岐するだけでなく BPC の退避も実現している．

＜２度目の BL 命令実行＞

pcout=01A 番地の C01E（表5.1：BL 1E）の発行により同様に pcout1=01B が BPC ファイルの rfp=3番地に書き込まれ，C01E の次の clk= ↑で bpc に書き込まれている．

＜ RTE 命令実行＞

（pcout=）01E 番地に分岐後，（pcout=）021番地の（inst=）F000（表5.1：RTE）命令が発行される．

この命令は分岐先からの復帰命令であるので, 戻り先アドレスを BPC から読み出して，そのアドレスに分岐しなければならない．

図より最初の F000が発行されたとき，BPC ファイルの状態は，we_bpc=0（読出），bpc=01B（BPC データ：戻り先アドレス），rfp=3（BPC のアドレス）であるので，clk= ↑となった瞬間に pcout=01B となり01B 番地の1210（表5.1：ADD R2 R1）が発行されている．また（file_pointer 入力 we_rfp=1，minus=1より rfp が３→２）rfp=2に変化するので（rfp=）２番地に格納されている BPC 値014が bpc に出力されている．

このように RTE 命令は，BPC に分岐するだけではなく，BPC の復帰も実現している．

＜２度目の RTE 命令実行＞

（pcout=）01B 番地に分岐後，（pcout=）01D 番地の（inst=）F000（表5.1：RTE）命令が発行される．

図より２度目の F000が発行されたとき，BPC ファイルの状態は，we_bpc=0（読出），

bpc=014（BPC データ：戻り先アドレス），rfp=2（BPC のアドレス）であるので，clk= ↑ となった瞬間に pcout=014となり014番地の1210（表5.1：ADD R2 R1）が発行されている．また（file_pointer 入力 we_rfp=1，minus=1より rfp が 2 → 1 ）rfp=1に変化するので（rfp=）１番地に格納されている BPC 値00C が bpc に出力されている．

5.1.2 ファイル・ポインタ（file_pointer）

図5.4にファイル・ポインタの回路図を示す．図において，count は，rst=0のときに 0 を出力し，we_rfp= ↑のときに s1出力の flpt を読み込んで count に出力する．右の加算器は count を 1 だけインクリメントし，左の加算器は 1 だけデクリメントする．register_4は 4 ビットレジスタで rst=0で 0 を出力し，clk= ↑のとき flpt を読み込んで flptr を出力する．従って flptr は flpt より1clk 過去の値になっている．セレクタ s1と s2は共に minus により制御され，s1はリセット付きセレクタで，minus=0のときインクリメントした値を，minus=1のときデクリメントした値を選択する．s2は minus=0のときインクリメントした直ぐの値を，minus=1のときデクリメントした値の1clk 過去の値を出力する．

次に CPU 中での file_pointer の動作について図5.5で説明する．図において file_pointer の信号は，rst ～ rfp[3:0]である．minus は rfp 出力をインクリメントするかデクリメントするかを指示する入力，we_rfp は file_pointer を動作させる信号，flpt はインクリメント時に rfp に出力する信号，flptr はデクリメント時に rfp に出力する信号である．また図中の数字は全て16進数である．

file_pointer が動作するのは，we_rfp=1のときである．図で we_rfp=1のときの命令 inst を確認すると，順に E007（表5.1：TRAP 7），C017（表5.1：BL 17），C01E（表5.1：BL 1E），F000（表5.1：RTE）である．即ち分岐・復帰命令のときに動作する．

pcout=00B の inst=E007のとき，we_rfp=1，minus=0なので，flpt=0→1になり rfp に出力される．BPC ファイルでは，これを受けて rfp=1番地に pcout1=00C が保存される．次の pcout=013，inst=C017のときも we_rfp=1，minus=0なので，flpt=1→2になり rfp に出力される．BPC ファイルでは，これを受けて rfp=2番地に pcout1=014が保存される．次の pcout=01A，inst=C01E のときも同じく rfp=2→3になる．BPC ファイルでは，これを受けて rfp=3番地に pcout1=01B が保存される．

pcout=021の inst=F000のとき，we_rfp=1，minus=1なので，flpt=3→2になるが，図5.4で述べたように minus=1のとき flptr=3が rfp に出力され BPC ファイルに送られる．BPC ファイルでは，これを受けて rfp=3番地に bpc=01B を出力する．即ち rfp は変化せずに送られ，次の clk= ↑で flptr=3→2，rfp=3→2となりデクリメントされる．

次の pcout=01D の inst=F000のときも F000が発行されたときは rfp=2のままで，次の clk= ↑で rfp=2→1となりデクリメントされる．

以上の file_pointer 動作により BPC ファイルでは，書き込むときは rfp をインクリメン

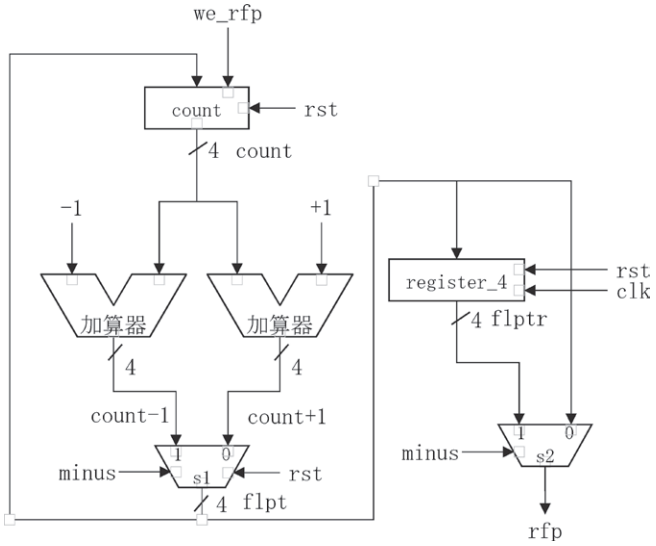


図5.4　ファイル・ポインタ回路図

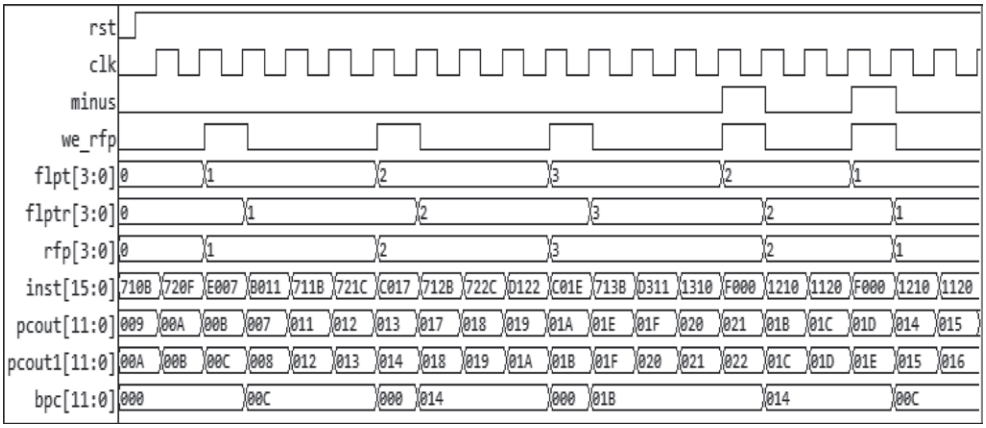


図5.5　file_pointer の動作

トしたアドレスに pcout1を保存していき，読み出すときは rfp をインクリメントしたアドレスの pcout1を bpc に出力した後に rfp をデクリメントしたアドレスの pcout1を bpc に出力するという動作を保証している．

5.2 D ステージ回路

5.2.1 D ステージ回路の概要

図5.6に新型 CPU の D ステージ回路を示す．このステージにおける新技術は，レジスタの退避・復帰命令を削除するため，16個のレジスタから成るレジスタ・ファイルを 1 バンクとして 9 バンクから成るレジスタ・ファイル・バンクを開発したことである．

図5.6において，灰色に塗りつぶした回路は，クロックに同期する回路である．楕円の

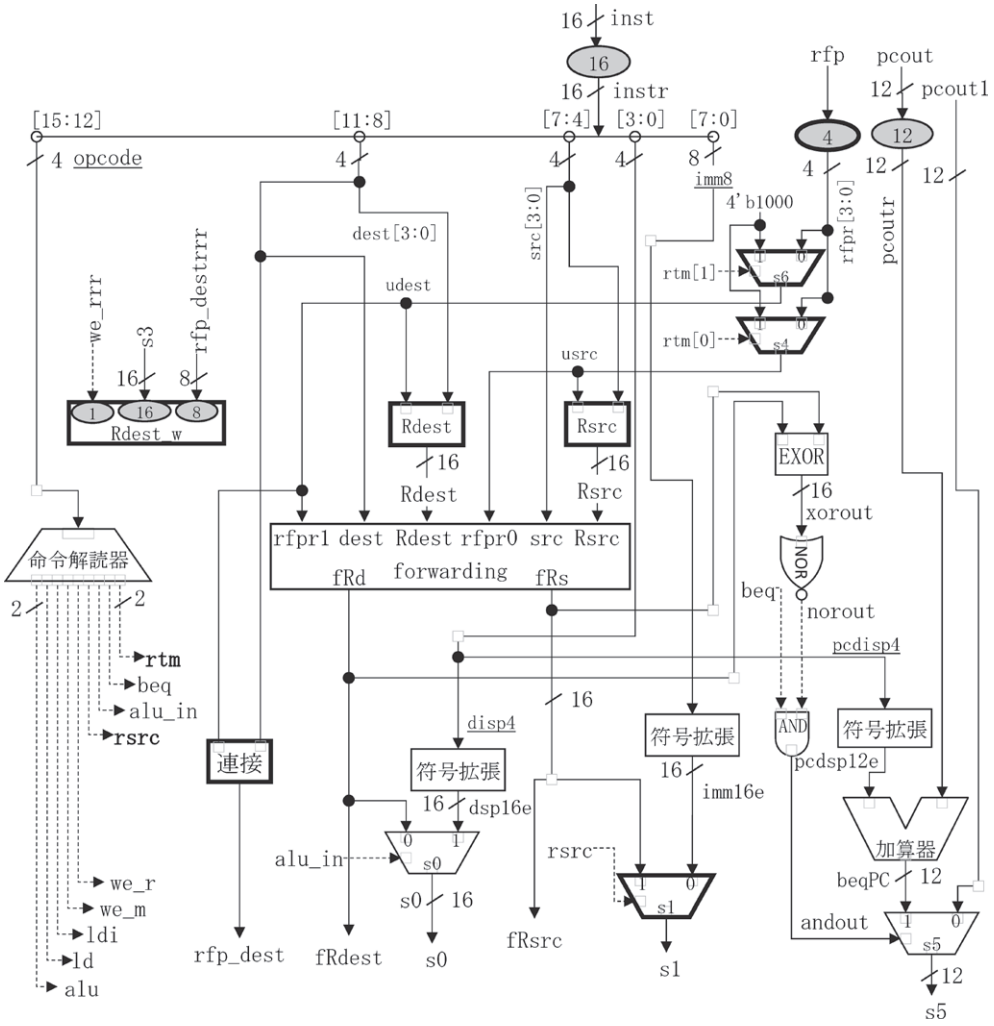


図5.6　新方式 CPU の D ステージ回路図

塗りつぶし回路はパイプラインのための D ステージ同期レジスタで楕円内の数字はビット数を示す．また太枠回路は，新型 CPU 回路にのみ存在する回路である．

命令解読器は F ステージで解読しなかった命令を解読する回路で 9 本の制御線を出力する．

レジスタ・ファイル・バンクは Rdest_w，Rdest，Rsrc の 3 つに分けて描かれているが，機能別に分けて表現しているだけで回路としては 1 つである．Rdest_w はレジスタへの書込回路，Rdest は表5.1の命令フォーマットにおける dest 番地からの読出回路，Rsrc は同じく src 番地からの読出回路である．dest や src は 4 ビットであるが，レジスタ・ファイル・バンクのアドレス入力は 8 ビットである．8 ビットのアドレスは，Rdest の場合は 4 ビットの udest と 4 ビットの dest の連接から，Rsrc の場合は 4 ビットの usrc と 4 ビットの src

の連接からそれぞれ生成される．s6と s4の 0 入力には rfpr[3:0]が，s6と s4の 1 入力には10進数の 8 が入力され，2 ビットの rtm[1:0]信号のうち，s6は rtm[1]により，s4は rtm[0]により選択される．

forwarding 回路は，データ・ハザードを解消する回路で，fRd から fRdest が，fRs より fRsrc が出力される．この回路は本研究内容ではないので説明は省略する．

EXOR → NOR は，EXOR に入力される fRdest と fRsrc が等しいかどうかを判定する回路で，等しい場合に 1 となる．

加算器は，表4.1に示す BEQ 命令での PC+$(\text{pcdisp4})_{12}$の加算に使用される．

5.2.2 レジスタ・ファイル・バンク

図5.7にレジスタ・ファイル・バンクの構成図を示す．rad1[7:0]は読出アドレス 1 で，そのアドレスに存在するレジスタ値が out1[15:0]から出力される．rad2と out2の関係も同じである．wad[7:0]は書込みアドレスで，書込許可 we=1のときに in[15:0]に与えられているレジスタ値を wad アドレスに書込む．

レジスタは144個あるが，R0～ R127の128個について，16個ずつ 8 個の組に分けて，R0～ R15をバンク 0，R16～ R31をバンク 1，・・・・R112～ R127をバンク 7 とした．バンクアドレス（0～7）は rad1，rad2，wad のアドレスの[6:4]の3bit（000～111）で指定する．この上位3bit は，図5.6の D ステージ回路図にある rfpr[3:0]を s4および s6セレクタで

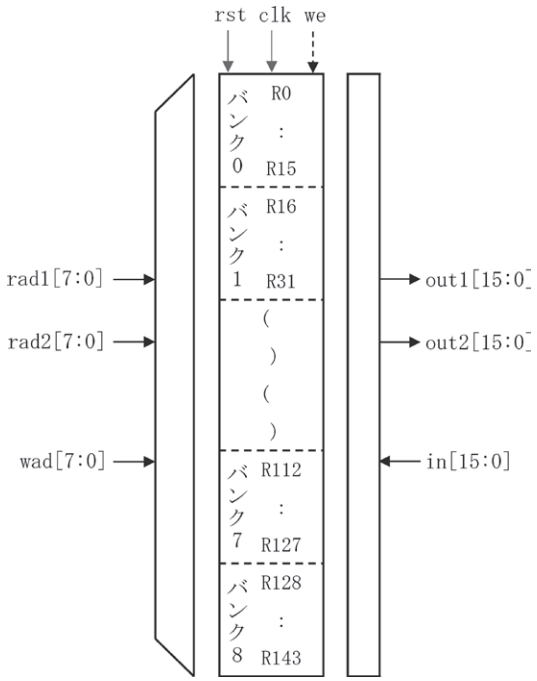


図5.7　レジスタ・ファイル・バンクの構成図

選択した４ビットである．各バンク内での16個のレジスタは図5.6の dest[3:0]と src[3:0]により指定される．まとめるとレジスタ・ファイルの３つのアドレス：rad1, rad2, wad には rfp と dest，rfp と src，rfp と dest' を連接したものが入力され，rad1={rfp,dest}，rad2={rfp,src}，wad={rfp,dest'} となる．上記5.1項で述べたように rfp は割込・分岐命令が入ったときに F ステージで指定されるポインタで，図5.6の s4と s6で異なる rfp を選択することはできないので，同一バンク内の16本のレジスタを使ってプログラムが実行される．

図5.8に CPU の D ステージにおけるレジスタ・ファイル・バンクの割込・分岐・グローバル・レジスタ動作を示す．レジスタ・ファイル・バンクの信号は clk ～ out2までである．図において数字は全て16進数である．

pcoutr=009番地の instr=710B（表5.1：LDI R1 B）命令は，R1←000B を実行する．R1の１は，４ビットの udest と dest の連接であるが，図のように udest=0，dest=1より {udest, dest}=01となっている．次の pcoutr=00A 番地の instr=720F 命令は，R2←000F を実行する．R2の２は同じく図で udest=0，dest=2より {udest, dest}=02となっている．udest=0はバンク０のレジスタ・ファイルである．usrc も０である．

次の pcoutr=00B 番地の instr=E007（表5.1：TRAP 7）命令は割込命令なので，図のように udest=1に増加する．これは次の命令からバンク１のレジスタ・ファイルを使用することを示す．以降の711B（R1←1B），721C（R2←1C）命令での {udest, dest} は，図より =11, 12 となっている．usrc も１に増加している．

次の pcoutr=013番地の instr=C017（表5.1：BL 17）命令は分岐命令なので，図のように

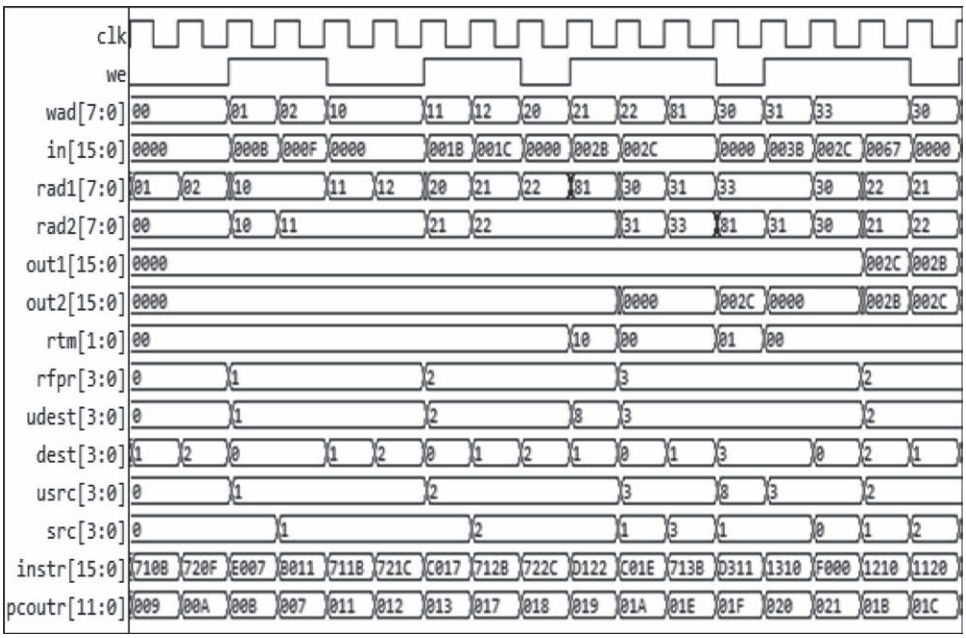


図5.8　レジスタ・ファイル・バンクの動作

udest=2に増加する．これは次の命令からバンク２のレジスタ・ファイルを使用することを示す．以降の712B（R1←2B），722C（R2←2C）命令での {udest, dest} は，図より =21, 22 となっている．usrc も２に増加している．

次の pcoutr=019番地の instr=D112（表5.1：LDGR GR1← R2）命令は R2を GR1にコピーする命令である．Rdest や Rsrc はバンク２にあるので，R2の usrc=2, src=2である．GR1はバンク８のグローバル・レジスタなので，図のように udest=8, dest=1になっている．即ち，レジスタ・ファイル・バンクで，22番地のデータを81番地にコピーするのである．

次の pcoutr=01A 番地の instr=C01E（表5.1：BL 1E）命令は分岐命令なので，図のように udest=3に増加する．これは次の命令からバンク３のレジスタ・ファイルを使用することを示す．以降の713B（R1←3B）命令での {udest, dest} は，図より =31となっている．usrc も３に増加している．

次の pcoutr=01F 番地の instr=D311（表5.1：LDGR R3← GR1）命令は GR1を R3にコピーする命令である．Rdest や Rsrc はバンク３にあるので，R3の udest=3, dest=3である．GR1はバンク８のグローバル・レジスタなので，図のように usrc=8, src=1になっている．即ち，レジスタ・ファイル・バンクで，81番地のデータを33番地にコピーするのである．以降の1310（ADD R3 R1）命令での {udest, dest}，{usrc, src} は，図より33, 31となっている．

以上の動作で rfpr と udest, usrc の関係を見ると，図5.8よりグローバル・レジスタが使用される D122や D311命令以外で rfpr=udest=usrc であることが分かる．rfpr は F ステージの rfp なので，rfp（rfpr）は BPC ファイルとレジスタ・ファイル・バンクの両方に与えられ，割込や分岐が起きたときに BPC ファイルの BPC 保存番地とレジスタ・ファイル・バンクのバンク番号を増加させることで BPC やレジスタの「退避」を不要にしているのである．

他方復帰動作は，図の pcoutr=021番地の instr=F000（表5.1：RTE）命令で実現される．RTE 命令が発行されると前5.1.2項で述べたように，rfp（rfpr）は，発行と同時にデクリメントされるのではなく，次の clk= ↑でデクリメントされる．図では pcoutr=01B 番地に復帰すると，rfpr=udest=usrc=2となり，バンク３からバンク２に移る．01B 番地の instr=1210（ADD R2 R1）の R2と R1の２や１は，{udest,dest} や {usrc,src} なので，図より22, 21となりバンク２に移って計算が行われる．

以上の動作から，RTE による復帰では BPC ファイルの BPC 保存番地とレジスタ・ファイル・バンクのバンク番号を減少させることで BPC やレジスタの「復帰」を不要にしているのである．

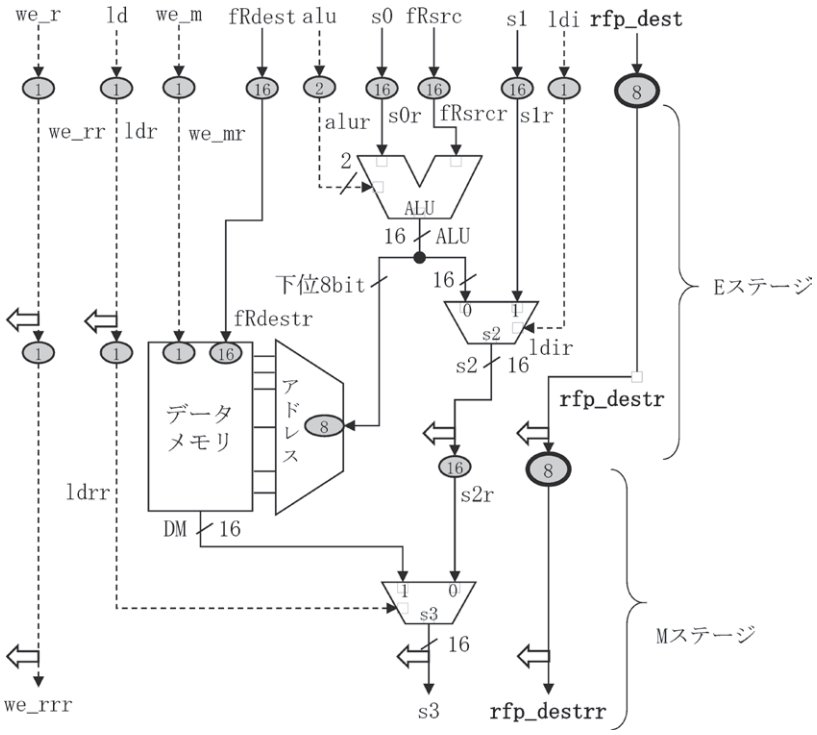


図5.9　新方式 CPU の E ステージと M ステージの回路図

5.3 E ステージと M ステージ回路

図5.9に新型 CPU の E ステージと M ステージの回路図を示す．図において上側が E ステージ，下側が M ステージである．新方式 CPU に関してこれらのステージに新奇性はないが，CPU として評価するため概要を説明する．

5.3.1 E ステージ回路

スタック方式 CPU の E ステージでは PUSH や POP 命令に必要となる SP の計算回路が必要であったが，新方式では不要である．

図5.9において，E ステージの全ての入力にはパイプライン・レジスタが配置されている．表4.1に示したように演算命令は４つだけなので，E ステージの回路は ALU とセレクタ s2だけである．

ALU では表4.1の４つの演算命令を実行する．演算を指示するのは２ビットの alu 信号で，alu=00のとき加算，01のとき減算，10のとき論理積，11のとき論理和を指示する．また，LD 命令や ST 命令におけるデータメモリのアドレス計算（加算，減算）も行う．データメモリのアドレスは８ビットなので，ALU の16ビット出力の下位８ビットだけを M ステージに送る．上位８ビットは放置される．

E ステージの出口にある左横向き矢印は，D ステージのフォワーディング回路へ向かう

信号である．

5.3.2 M ステージ回路

図5.9において，M ステージの全ての入力にもパイプライン・レジスタが配置されている．このステージではデータメモリへの書込みと読出しが行われるので，回路はデータメモリとセレクタ s3だけである．

データメモリに書込む場合, we_mr=1とすると fRdestr が ALU[7:0]番地に書き込まれる．読み出す場合，we_mr=0とすれば ALU[7:0]番地にあるデータが DM から出力される．即ち，書込みか読出かは we_mr が 1 か 0 かで決まる．

ST 命令の場合，書込みが終わると命令が終了する．

LD 命令の場合，読み出したデータ DM を s3で ldrr=1として選択し，s3から D ステージに送ってレジスタ・ファイルに書き込むまで終了しない．レジスタ・ファイルへの書込みアドレスは，D ステージで命令が解読されてから一緒にパイプライン・レジスタを通過してきた rfp_destrr である．これは5.2.2項で述べた {udest, dest} そのもので，D ステージから rfp_dest として出力されていた．即ち, 命令では dest は 4 ビットであったが, レジスタ・ファイル・バンク以降は 8 ビットとして扱っている．書込許可は we_rrr であり, D ステージの命令解読器出力の we_r が一緒にパイプライン・レジスタを通過してきている．

M ステージの出口にある左横向き矢印は，D ステージのフォワーディング回路へ向かう信号であり，且つレジスタ・ファイル・バンクへも転送されて書き込まれる．

6. 新方式 CPU の設計検証

6.1 HDL 記述と論理シミュレーション

新方式 CPU の各 RTL 回路は Verilog HDL により記述されている．表6.1に新方式 CPU の Verilog HDL ファイルの階層構造，モジュール名および HDL 行数を示す．なお行数にコメント文は含まれていない．

表6.1に示すように，0 階層は CPU 全体回路，1 階層は CPU 全体回路にインクルードされる各パイプライン・ステージそして 2 階層は各ステージにインクルードされるモジュールである．2 階層の一部のモジュールは 1 階層に重複してインクルードされるので，重複を削除した 2 階層のモジュールを右側に再掲した．新方式 CPU の HDL 記述行数の総計は566行である．

比較のために表6.2にスタック方式 CPU の Verilog HDL ファイルの階層構造，モジュール名および HDL 行数を示す．スタック方式 CPU の HDL 記述行数の総計は591行である．

表6.1と表6.2の違いは 2 階層である．新方式 CPU に存在するファイルは，bpc_file.v, control_fetch.v, file_pointer.v, reg_file_bank.v であり, スタック方式に存在するファイルは，

表6.1　新方式 CPU の Verilog HDL ファイル：階層別 module 名（行数）

0階層(行数)	1階層(行数)	2階層	重複のない2階層
cpu_novel.v(62)	fetch2.v(19)	adder_12.v **bpc_file.v** **control_fetch.v** **file_pointer.v** PC_12.v rom_cpu.v selector_12.v	adder_12.v(8) alu.v(15) bpc_file.v(13) control.v(33) control_fetch.v(29) EXOR_NOR_8.v(7) extend_3to12.v(11) extend_4to12.v(11) extend_4to16.v(11) extend_8to16.v(11) file_pointer.v(16) forwarding2.v(27) memory.v(13) PC_12.v(10) reg_file_bank.v(18) register_1.v(9) register_12.v(10) register_16.v(10) register_2.v(10) register_4.v(10) register_8.v(10) rom_cpu.v(44) selector_12.v(13) selector_16.v(13) selector_4.v(13)
	decode2.v(55)	adder_12.v control.v EXOR_NOR_8.v extend_3to12.v extend_4to12.v extend_4to16.v extend_8to16.v forwarding2.v **reg_file_bank.v** register_12.v register_16.v register_4.v selector_12.v selector_16.v selector_4.v	
	execution2.v(36)	alu.v register_1.v register_16.v register_2.v register_8.v selector_16.v	
	mem_access2.v(23)	memory.v register_1.v register_16.v register_8.v selector_16.v	
62行	133行	—	371行

bpc_12.v と stackpointer2.v である.

表6.1の 0 階層である cpu_novel.v の HDL に対する論理シミュレーションの結果，前項5.1.1で説明した図5.3および図5.5に示すように BPC ファイルとファイル・ポインタの正常動作を確認し，前項5.2.2で説明した図5.8に示すようにレジスタ・ファイル・バンクの正常動作を確認した.

表6.2　スタック方式 CPU の Verilog HDL ファイル：階層別 module file 名（行数）

0階層(行数)	1階層(行数)	2階層	重複のない2階層
cpu_stack.v(71)	fetch2.v(22)	adder_12.v **bpc_12.v** extend_12to16.v PC_12.v register_1.v rom.v selector_12.v selector_16.v	adder_12.v(8) alu.v(24) bpc_12.v(19) control.v(28) EXOR_NOR_4.v(7) extend_12to16.v(8) extend_3to12.v(8) extend_4to12.v(8) extend_4to16.v(8) extend_8to16.v(8) forwarding2.v(26) memory.v(13) PC_12.v(10) reg_file.v(18) register_1.v(9) register_12.v(10) register_16.v(10) register_2.v(10) register_4.v(10) register_8.v(10) rom.v(93) selector_12.v(13) selector_16.v(13) selector_8.v(13) stackpointer2.v(20)
	decode2.v(45)	adder_12.v control.v EXOR_NOR_4.v extend_3to12.v extend_4to12.v extend_4to16.v extend_8to16.v forwarding2.v reg_file.v register_12.v register_16.v selector_12.v selector_16.v	
	execution2.v(31)	alu.v register_1.v register_16.v register_2.v register_4.v register_8.v selector_16.v selector_8.v **stackpointer2.v**	
	mem_access2.v(18)	memory.v register_1.v register_16.v register_4.v selector_16.v	
71行	116行	—	404行

6.2 FPGA への実装

新方式 CPU を評価するために，２項で述べたスタック方式 CPU も FPGA 向けに QuartusII で論理合成し，ゲート規模や遅延時間等を比較した．

表6.3に FPGA の仕様を示す．FPGA は Altera 社の CycloneII シリーズの EP2C5で，4608個の Logic Elements（LE），26個の4kbits+512parity bits の RAM（計119808bits），26個の組込用9bits x 9bits 乗算器と２個の PLL を搭載している．最大 I/O ピン数は158である．

図6.1に Logic Elements（LE）の回路図[12]を示す．図で組合せ回路は Look-Up Table（LUT）で構成され，レジスタ等は D フリップフロップである Programmable Register で実現される．

表6.3　FPGA 仕様

Family	CycloneII/Altera
Device(TSMC 90nm)	EP2C5 QFP208 8bit
Logic Elements(LE)	4608
RAM (4kbits+512parity bits=4608bits)	26
Total RAM bits (4608bitsx26=119808)	119808
Embedded 18bitsx18bits multipliers	13
Embedded 9bitsx9bits multipliers	26
PLL	2
Maximum user I/O pins	158

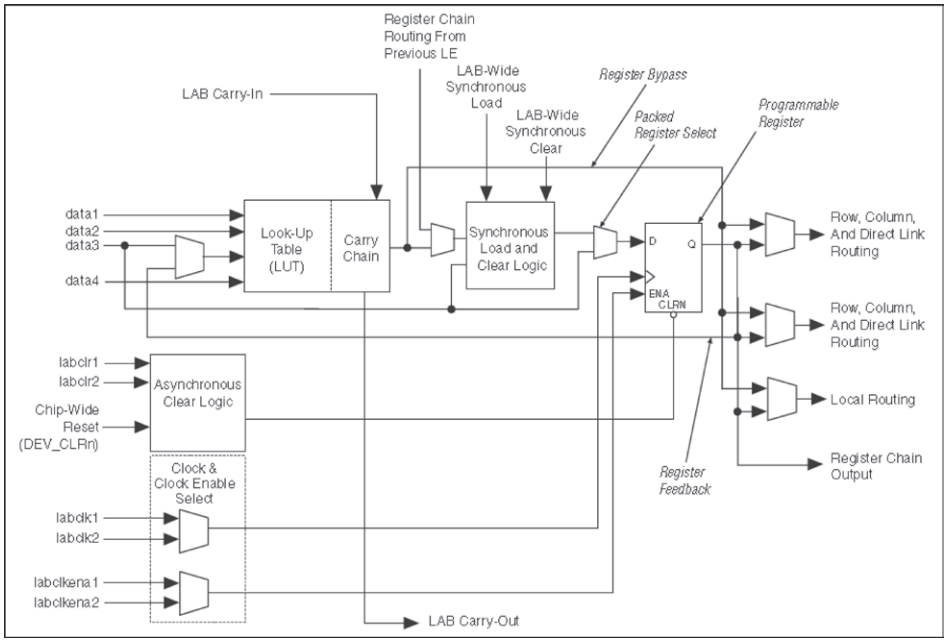


図6.1　QuartusII Logic Element

表6.4に QuartusII による新方式とスタック方式の論理合成結果を示す．QuartusII のデータ参照箇所は，Compiler tool/ Report/ Flow Summary と Compiler tool/ Report/ Timing Analyzer である．

表6.4より Total Logic Elements（LE）は新方式の方が48.8% 増加した．LE のうち組合せ回路である Total Combinational functions では，新方式の方が47.4% 増加し，LE から構成されるフリップフロップである Dedicated Logic Registers は86.7% 増加した．

Total Combinational functions の増加は，新方式での BPC ファイル，ファイル・ポインタ回路とスタック方式でのスタックポインタ回路の差であると思われる．

表6.4　論理合成結果

評価項目	新方式CPU	スタックCPU	増分(%)	QuartusII参照箇所
Total Logic Elements(LE)	1558	1047	48.8	Compiler tool/ Report/Flow Summary
-Total Combinational functions	1359	922	47.4	
-Dedicated Logic Registers	870	466	86.7	
Total Registers	870	466	-	
Total memory bits(RAM bits)	4096	4096	0	
Embedded multipliers	0	0	-	
PLL	0	0	-	
Maximum user I/O pins	22	22	0	
Worst case tco(clock to out) ALU[7:0] → memory → s3	23.313 ns	22.313 ns	4.5	Compiler tool/ Report/Timing Analyzer
参考：Worst case frequency	42.9 MHz	44.8 MHz	-0.04	1/(Worst case tco)

Dedicated Logic Registers の増加は，レジスタ・ファイルをバンク化して９倍にしたためであるが，その割に増えていないのはパイプライン・レジスタの量が多く，元々レジスタ数のベースが多かったためである．逆にいえばバンク化しても２倍以下で収まるならば，バンク化による clk 数の削減効果を考慮すると新方式を採用すべきである．

Total memory bits は新方式，スタック方式ともに4096bits であった．これは，表6.1の cpu_novel.v/ mem_access2.v/ memory.v が両方式で同じで，HDL 記述のメモリセル部を 16bits x 256words（=4096bits）としたためである．memory.v が LE ではなく RAM で生成されたのは，memory.v の記述が表6.3の4kbits の RAM と一致したためと思われる．

使用した I/O ピンは22ピンで，RESET，CLK，８個の LED，８個のディップスィッチそして４個のプッシュスィッチである．

Worst case tco では，新方式 CPU とスタック方式 CPU で，ほぼ同じであった．この遅延パスは，両方式共に図5.9のデータメモリのアドレス入力である ALU からデータメモリの出力 s3までであった．メモリ内はアナログ回路でパイプライン分割できないので，メモリを含むパスは最長遅延になる傾向があるので，この結果は妥当である．従って tco 遅延時間のわずかな差は FPGA 内でのレイアウトの違いから生じていると断定できる．

以上のことから，新方式 CPU はスタック方式 CPU に比べ LE 数が48.8% 増加したが，最長遅延時間は新奇回路ではなくデータメモリで律速されていることが分かった．

む す び

エンジン制御やセンサによる姿勢制御等，割込処理や分岐処理が多く，且つリアルタイム性を要求される16ビットクラスの CPU の高速化を図るため，割込命令や分岐命令に伴

う戻り先アドレスやレジスタの退避，元のプログラムへの復帰命令に伴う戻り先アドレスやレジスタの復帰操作に関して，データメモリ上のスタック領域を使用せず，その結果としてスタックへの退避命令やスタックからの復帰命令も必要としない新方式のハードウェア・アーキテクチャを開発した.

開発した方式では，割込処理や分岐処理プログラムへの分岐を実現する割込命令や分岐命令が戻り先アドレスやレジスタの退避操作も行えるようにし，元のプログラムへの復帰を実現する復帰命令が，戻り先アドレスやレジスタの復帰操作も行って元のプログラムに分岐できるようにした.

このアーキテクチャを実現するために，複数の戻り先アドレスを記憶する BPC ファイルとレジスタ・ファイル 9 個から成るレジスタ・ファイル・バンク，および割込・分岐・復帰に対応して BPC ファイルとレジスタ・ファイル・バンクを制御するファイル・ポインタを考案した.

またプログラム作成の便宜のために分岐前プログラムと分岐後プログラムとの間でデータをやり取りするためグローバル・レジスタも導入した.

HDL シミュレーションの結果，割込命令，分岐命令，復帰命令は，ファイル・ポインタを制御することで，命令発行時または次の clk で戻り先アドレスやレジスタの退避・復帰を実現していることを確認した．これにより新方式 CPU において 1 つの割込（分岐）当たり16レジスタで34clk, 32レジスタで66clk の削減が可能となった．またスタック CPU ではサポートしていなかったグローバル・レジスタの動作も確認した.

新方式およびスタック方式を論理合成し, 実回路で回路規模や遅延時間を比較した結果, バンク化によるレジスタ数の増加は想定を下回り且つ遅延時間は変わらないことが判明し，新方式 CPU が有用であることが実証された.

参考文献

［1］三菱32ビット RISC シングルチップマイクロコンピュータ　M32R ファミリ　CPU 命令セットユーザーズマニュアル：1996-7-31 Ver1.00.

［2］パターソン＆ヘネシー，コンピュータの構成と設計 第 2 版上，日経 BP 社，東京，2002.

［3］黒岩将平，荒堀喜貴，権藤克彦，“実行可能コードを対象とするスケーラブルかつ部分的パス依存なバッファ・オーバフロー静的検知,”情報処理学会論文誌プログラミング（PRO）12(3), 13-13, 2019-07-17.

［4］折笠雄太朗，千葉 滋，“メモリ効率の良いスレッド生成のためのスタック領域のリンクリスト化,”日本ソフトウェア科学会大会論文集 34, 331-336, 2017-09-18.

［5］中嶋健一郎，山田真大，長尾卓哉[他]，山崎二三雄，武井千春，本田晋也，高田広章，“ARMv6アーキテクチャを用いたメモリ保護 RTOS のユーザスタック保護の設計と評

価,”情報処理学会研究報告．EMB，組込みシステム 14, H1-H11, 2009-07-24.
［6］高井康浩，“高速メモリインターフェース：DDR/GDDR-DRAM,”電子情報通信学会技術研究報告 . ICD, 集積回路 110(9), 81-82, 2010-04-15.
［7］井上弘士，“バッファ・オーバフロー検出を目的としたセキュア・キャッシュの性能 / 消費電力解析,”電子情報通信学会技術研究報告．ICD，集積回路 105(475)，43-48, 2005-12-15.
［8］森若和雄，中西恒夫，福田 晃，“スタックのオンチップメモリへの割り当てによるキャッシュ消費電力の削減 ,”情報処理学会研究報告．OS，［システムソフトウェアとオペレーティング・システム］95, 33-40, 2004-02-26.
［9］佐藤弘紹，門田暁人，松本健一，“データの符号化と演算の変換によるプログラムの難読化手法,”電子情報通信学会技術研究報告．ISEC，情報セキュリティ 102(743), 13-18, 2003-03-19.
［10］片山清和，安藤秀樹，島田俊夫，“関数呼び出し時のレジスタの退避 / 復元に着目したメモリリネーミング手法,”情報処理学会研究報告．ARC，計算機アーキテクチャ研究会報告 150, 107-112, 2002-11-27.
［11］上田陽平，山本泰宇，関口龍郎［他］，米澤明憲，“アセンブリ言語レベルでの異種計算機間のヒープとスタックの共有機構,”情報処理学会論文誌プログラミング（PRO）42（SIG03（PRO10））, 27-39, 2001-03-15.
［12］ALTERA　CYCLONE II DEVICE HANDBOOK, VOLUME 1　CycloneII Architecture