卒業論文

マルチ ALU プロセッサのアセンブリプログラミングの動作検証

氏　　名：國吉　裕大
学籍番号：2260070038-7
指導教員：山崎　勝弘　教授
提出日：2012 年 2 月 20 日

立命館大学　理工学部　電子情報デザイン学科

内容梗概

本研究では、ハード/ソフト協調学習システムとデザインパターンを用いて設計されたプロセッサの仕様をもとに設計されたアセンブラとシミュレータを用いてアセンブリプログラミングを行い評価と検証することを目的としている。バイトニックソートのプログラミングを行った。また、プログラムの静的、動的における並列演算、連鎖演算、単一実行数の判別を行いマルチＡＬＵプロセッサＭＡＰの有効性を検証した。

目次

図目次

表目次

1. はじめに

半導体技術は近年急速に発展しており、低消費電力化、LSI の軽量化、高速化の技術が進んでいる。日常生活に関連するものとしては、自動車、パソコン、スマートフォン、テレビ、ゲーム機などが挙げられる。これらの機器はいずれもハードウェアとソフトウェアから構成されており、それらの機器に求められる仕様は大規模になりかつ高速化や小型化が求められている。さらには、製品のライフサイクルは縮小し、開発期間の短縮化も求められている。そのことによって、ハード/ソフト協調設計が開発期間短縮に大きく影響を及ぼす。近年の LSI 開発技術はハードウェアとソフトウェアに密接な関係があり、ハードとソフトの両方の知識を習得するためにも、早期の教育をする必要がある。このように、高集積システム LSI 技術の進化の中、性能は多様化しており、ハードとソフト両方の知識を持ち、さらにプロセッサにおける命令セットとマイクロアーキテクチャの知識が必要不可欠である。

我々は、ハード/ソフト協調学習システムを用いて、複数の ALU による並列処理可能なマルチ ALU プロセッサ（MAP）の設計と開発を進めている。 MAP は、2 つの ALU を有し、並列演算、連鎖演算が有効である。昨年度、マルチ ALU プロセッサ（MAP）用のアセンブラとシミュレータを設計し、開発を行っている。本研究では、マルチ ALU プロセッサ（MAP）のアセンブリプログラムの記述を行い、アセンブラ、シミュレータを通しアセンブリプログラムの検証を行い、FPGA ボード上での動作検証と評価を行うことを目的とする。

本研究では、バイトニックソートのプログラムを MAP のアセンブリプログラムの記述し、アセンブラ、シミュレータを通してプログラムの検証を行い、また FPGA ボード上での動作検証と評価を行う。プログラム内の静的、動的な並列演算と連鎖演算の有効性を検証する。

本論文では、第２章では MAP の構成と命令セットアーキテクチャの詳細を説明する。第３章では MAP で記述したバイトニックソートの検証、評価を行う。

2. マルチ ALU プロセッサ

2.1 MAP の構成

MAP は MONI プロセッサと MIPS プロセッサを参考として構成している。
MAP には以下のような特徴がある。

① ２ALU による並列処理
② １ALU を３２ビットで制御し、２命令をフェッチする
③ ４つの命令形式で全３７命令
④ レジスタファイルは３２個で、全 ALU で共有

図１に MAP のデータパスを示す

MAP は複数 ALU を有する 32 ビットのプロセッサである。院生が設計したプロセッサを、FPGA チップ上で実際に動作させ、大学教育でプロセッサ設計能力を習得することが目標である。MAP のデータパスを図 4 に示す。命令メモリとデータメモリが分離したハーバードアーキテクチャである。MAP の特徴は以下の 5 点に要約できる。

① ALU 数は 2、4、8	…	複数 ALU
② レジスタ共有	…	全 ALU による共有
③ PPU（Parallel Processing Unit）	…	並列連鎖単一処理判定部
④ ハーバードアーキテクチャ	…	命令メモリとデータメモリの分離
⑤ バイトアドレス方式	…	バイト毎のアドレッシング方法

例えば、レジスタ間演算命令では、命令メモリ(IM)から制御ユニット(CU)にオペコードとファンクションコードを送る。また IM から命令分のオペランドのアドレスと、演算結果を格納するアドレスをレジスタファイル(RF)に出力する。ALU は、CU から演算命令の種類を受け取り、演算データの演算を行い、指定したアドレスに演算結果を格納する。PC は加算器によって 8 加算する。

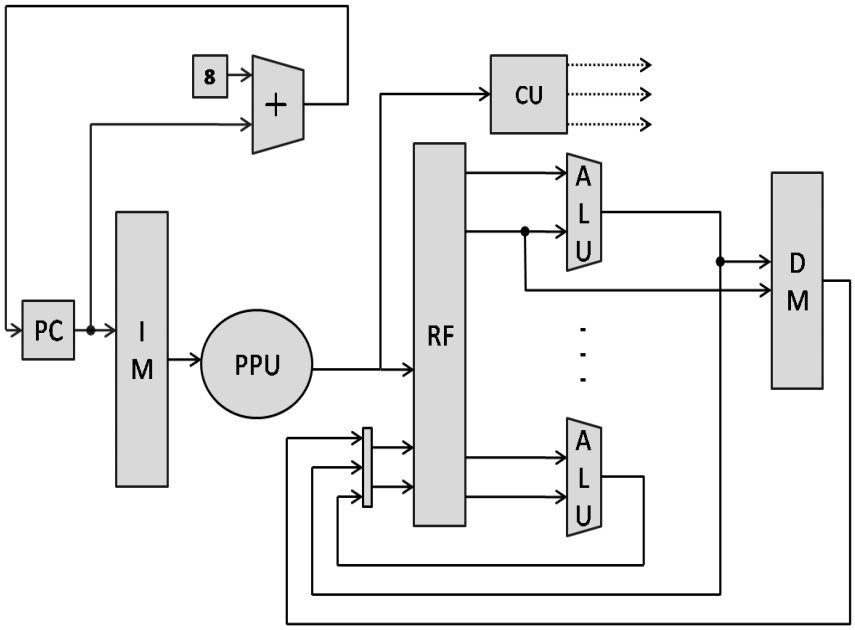

図 1：MAP のデータパス

2.2 MAP の命令セットアーキテクチャ

図 2 に MAP の命令形式を示す。MAP には、Jump 形式（J 形式）、Register 形式（R 形式）、Immediate 形式（I 形式）、Long 形式（L 形式）の 4 つの命令形式がある。J 形式には無条件分岐命令の JUMP、プログラム終了命令となる HALT を定義している。R 形式にはレジスタ間の演算を行う命令を定義している。I 形式にはレジスタ値と即値演算を行う命令を定義している。L 形式には条件分岐命令とメモリ・レジスタへのデータ転送命令を定義している。L 形式と J 形式では DM へのアクセスと PC に JUMP する範囲を多くとることにより、従来のプロセッサにはできなかった大規模な計算を行うことができる。

命令語長		32						命令種類
		6	5	5	5	5	6	
命令形式	R形式	Op	Rs	Rt	Rd	Sha mt	Fn	レジスタ同士による演算
	I形式	Op	Rs	Rd	imm			レジスタと即値による演算
	L形式	Op	Rs	Rd	address/immediate			メモリアクセスや分岐命令
	J形式	Op	address					JUMP、HALT命令

図 2：MAP の命令形式

表１に各命令フィールドの意味を示す。基本的には１ALU を３２ビットで動作させる。Op で命令を R 形式、I 形式、L 形式、J 形式と判断する。また R 形式においては、フィールド Fn で命令を詳細に識別しているため、Op１つに対して 64 種類まで拡張が行える。L 形式と J 形式では、プログラムカウンタやデータメモリにアクセスするフィールドを広く確保するために 64 ビットの長さをもっている。

表 1：命令フィールドの意味

フィールド	bit 幅	用途
Op	6	オペコード、命令を識別
Fn	6	ファンクション、R 形式命令を詳細に識別
Rs	5	ソースレジスタ
Rt	5	ソースレジスタ
Rd	5	演算結果を格納するレジスタ
Shamt	5	シフト量

アドレスの指定方式は、R 形式ではレジスタ直接、I 形式ではレジスタ即値、L 形式でメモリアクセスはベース相対アドレス、分岐命令は絶対アドレス、J 形式は絶対アドレスである。表２に命令セット一覧を示す。

表 2：命令セット一覧

	命令	Op	Rs/Rt/Rd			Shamt	Fn	動作
R	NOP	000000	XXXX				000000	No operetion
	ADD	000001	Rs	Rt	Rd	X	000000	Rd = Rs+Rt
	SUB	000001	Rs	Rt	Rd	X	000001	Rd = Rs-Rt
	AND	000001	Rs	Rt	Rd	X	000010	Rd = Rs&Rt
	OR	000001	Rs	Rt	Rd	X	000011	Rd = Rs|Rt
	XOR	000001	Rs	Rt	Rd	X	000100	Rd = Rs^Rt
	NOT	000001	Rs	X	Rd	X	000101	Rd = !(Rs)
	SLT	000010	Rs	Rt	Rd	X	000000	(if Rs < Rt) Rd = 1
	SGT	000010	Rs	Rt	Rd	X	000001	(if Rs > Rt) Rd = 1
	SLE	000010	Rs	Rt	Rd	X	000010	(if Rs ≦ Rt) Rd = 1
	SGE	000010	Rs	Rt	Rd	X	000011	(if Rs ≧ Rt) Rd = 1
	SEQ	000010	Rs	Rt	Rd	X	000100	(if Rs = Rt) Rd = 1
	SNE	000010	Rs	Rt	Rd	X	000101	(if Rs ≠ Rt) Rd = 1
	SLL	000011	Rs	X	Rd	Shamt	000000	Rd = Rs << Shamt
	SRL	000011	Rs	X	Rd	Shamt	000001	Rd = Rs >> Shamt
	SRA	000011	Rs	X	Rd	Shamt	000010	Rd = Rs >>> Shamt
	JR	000100	Rs	X			000000	PC = Rs
I	ADDI	001000	Rs	Rd	Imm(16bit)			Rd = Rs+imm
	SUBI	001001	Rs	Rd	imm			Rd = Rs-imm
	ANDI	001010	Rs	Rd	imm			Rd = Rs&imm
	ORI	001011	Rs	Rd	imm			Rd = Rs|imm
	XORI	001100	Rs	Rd	imm			Rd = Rs^imm
	SLTI	001101	Rs	Rd	imm			(if Rs < imm) Rd = 1
	SGTI	001110	Rs	Rd	imm			(if Rs > imm) Rd = 1
	SLEI	001111	Rs	Rd	imm			(if Rs ≦ imm) Rd = 1
	SGEI	010000	Rs	Rd	imm			(if Rs ≧ imm) Rd = 1
	SEQI	010001	Rs	Rd	imm			(if Rs = imm) Rd = 1
	SNEI	010010	Rs	Rd	imm			(if Rs ≠ imm) Rd = 1
	LDHI	010011	Rs	Rd	imm			Rd = {imm,Rs[15:0]}
	LDLI	010100	Rs	Rd	imm			Rd = {Rs[31:16],imm}
L	BEQZ	100000	Rs	X	imm			(if Rs = 0) PC = imm
	BNEZ	100001	Rs	X	imm			(if Rs ≠ 0) PC = imm
	LD	100010	Rs	Rd	imm			Rd = DM[Rs+imm]
	ST	100011	Rs	Rd	imm			DM[Rs+imm] = Rd
	JAL	100100	X	Rd	imm			PC = imm:Rd = PC+4
J	JUMP	111110	Imm(26bit)					PC = imm
	HALT	111111	XXXX					Exit

2.3　命令の記述例とメモリアクセス

(1)　擬似命令

擬似命令として COPY と CLEAR の二種類を用意する。

COPY	$1	$2	…	$1 に$2 の内容をコピー。
ADDI	$1	$2	0	
CLEAR	$1		…	$1 の内容を初期化。
SUB	$1	$1	$1	

(2)　命令記述例

表 1 の命令を用いて以下のように記述していく。

例	R 形式命令	加算	ADD	$1	$1	$1
I 形式命令	減算	SUBI	$1	$1	1	
L 形式命令	条件分岐	BNEZ	$1	LOOP		
L 形式命令	LD	LD	$1	0 [$0]		
J 形式命令	HALT	HALT				
L 形式命令	JAL	JAL	$1	WORK		
R 形式命令	JR	JR	$1			

(3)　メモリアクセス

基本的に$0 をベースレジスタとして扱う。

例　　LD　　$1　　8 [$0]

この例では、命令のアドレス部は 8 なので、プログラムの先頭を 0 番地とすると、8 番地のところに求めるデータが格納されていることになる。また、プログラムの先頭アドレス、すなわちベースレジスタ 0 に格納されている値は 0 番地になっています。したがって、求められる有効アドレスは、ベースレジスタ 0 の値＋命令で指定されたアドレス部の値＝ 8 となる。図 3 にデータメモリのアクセスを示す。

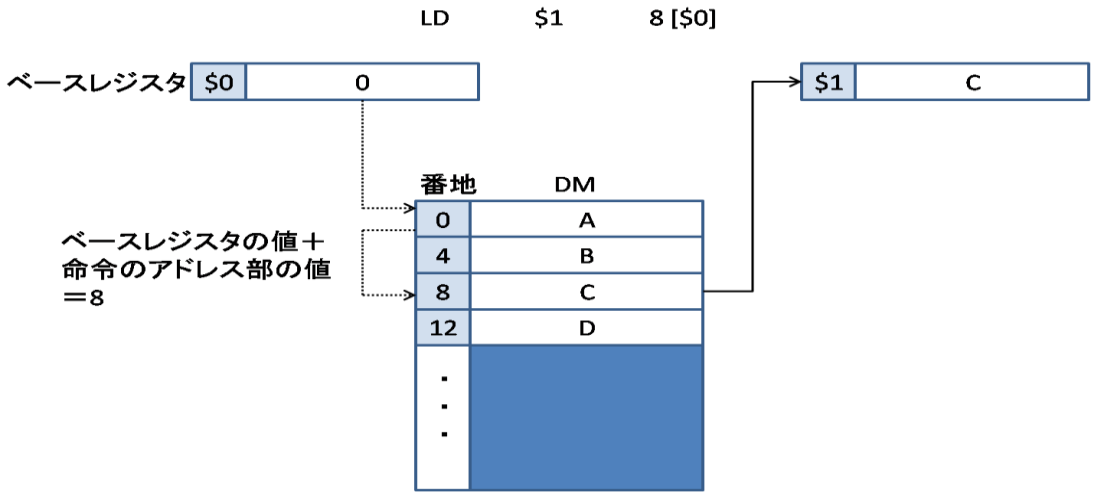

図 3：データメモリのアクセス

2.4 アセンブラとシミュレータ

(1)MAP アセンブラ

MAP アセンブラはアセンブリ言語で記述されたテキストファイルを入力とし、字句解析と構文解析を行い、機械語プログラムを出力する。まず Flex で字句解析を行い、ファイルのアセンブリ命令をトークンの並びに変換する。次に、Bison で構文解析を行い、トークン間の意味を解析していき、それに従って機械語を生成する。

(2)MAP シミュレータ

ユーザが記述したプログラムをアセンブラに通し機械語変換したものを入力とする。入力後、ユーザはプログラムに必要な値を DM に設定する。次にファイルオープンを行い、IM にすべての機械語を格納する。命令実行時は IM から 1 命令分のコードを取り出し、命令の OP を解析する。取り出した命令の命令形式を判別し、それぞれの演算別に分岐させる。次にそれぞれの分岐先で命令を解析し、レジスタ、即値、シフト量、ファンクションを判別し実行する。その後、結果として DM や RF などに値を保存し、それを HALT まで演算を実行し続ける。最終的な演算結果が DM に格納される。

MAP シミュレータの並列・連鎖演算と単一実行判定は、２つの命令を比較し並列・連鎖性を判定する。アドレス番地の値の小さい方を上位命令、大きい方を下位命令とする。まず、上位命令の判別を行う。上位命令のオペコードを解析し、終了命令もしくは分岐命令であるかを判別し、もし終了命令や分岐命令であった場合は単一実行とする。上位と下位にデータ依存がある場合は上位の演算結果を下位でも使用できる連鎖演算とする。上位と下位依存関係がない場合は同時実行とみなし並列演算とする。このようにして並列・連鎖演算と単一実行判定を行う。

3. バイトニックソートのアセンブリプログラミング

3.1 問題の定義

大小関係なくランダムに入力された数字を小さい数字から順番に並べ替えて出力するプログラムの作成。

3.2 バイトニックソートのアルゴリズム

バイトニックコードとは、上昇し下降する列、あるいは並べ替えるとそのようになる列である。バイトニックソートでは、まず各スレーブに値を渡して昇順、降順と交互にソートする。そしてソートできた列を昇順と降順で組み合わせて複数のバイトニック列をつくる。その列を値の大きいもの、小さいものに分け、再びソート、バイトニック列作成を行う。これを全スレーブのデータが組み合わせるまで行い、最後に各スレーブで昇順ソートを行うとバイトニックソートが完了する。以下の手順で要素交換を行うことで、ソートが完了する。図４、図５にバイトニックソートのアルゴリズムを示す。

1. 「昇順＋降順のバイトニック列」にする。
2. 2^n･･･2^0 間を昇順（降順）で比較する。

① バイトニック列にする

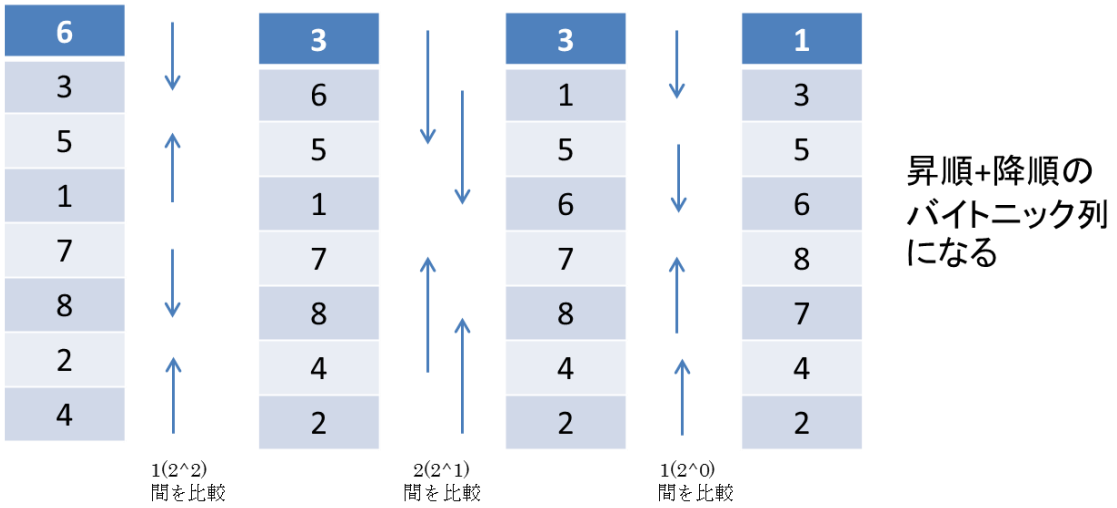

図 4：バイトニック列にする

② 昇順に並べる

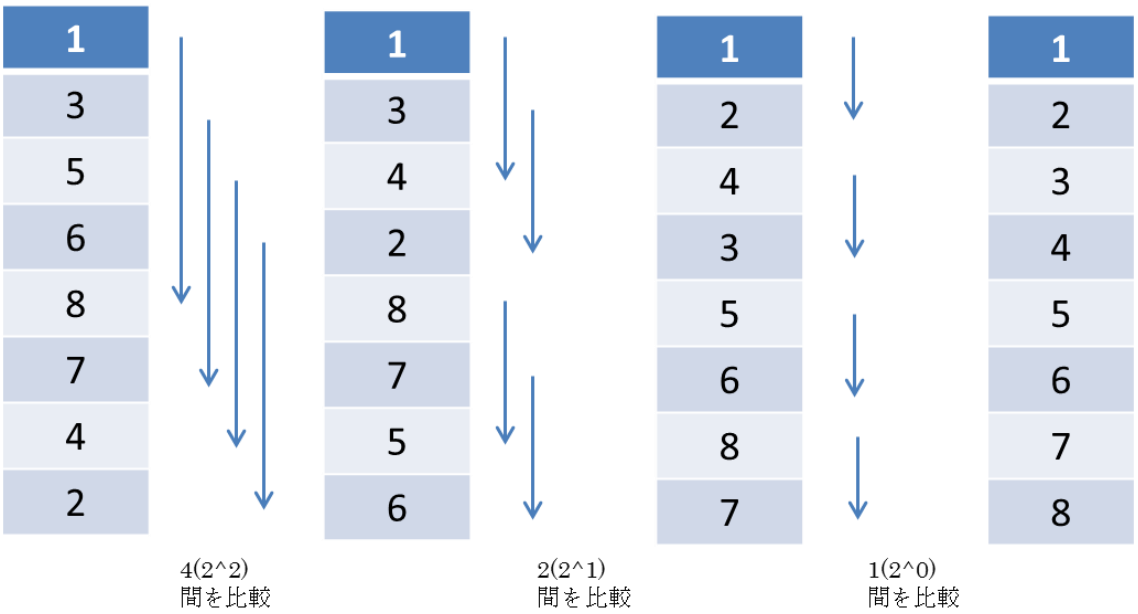

図 5：昇順に並べる

4.　シミュレータを用いた検証

4.1 実験環境

MAP アセンブラと MAP シミュレータを用いてシミュレーションを行う。

4.2 実験結果

今回は 16 個の数をソートするプログラムと 8 個の数をソートするプログラムを作成した。正の数でソートしたものと負の数を交えたものでソートする。結果を以下の表３、表４に示す。これらの表で、並列・連鎖・単一はそれぞれの演算の実行回数が入る。

表 3：16 個のバイトニックソート

	数	マイナスを含む数
ソートする数	15,10,16,6,1,3,9,2,11,8,13,5,12,7,4,14	15、-10、-16、6、1、-3、2、-11、8、-13、5、12、-7、4、14
結果	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16	-16,-13,-11,-10,-7,-3,1,2,4,5,6,8,9,12,14,15
並列	76	80
連鎖	80	80
単一	321	321

表 4：8 個のバイトニックソート

	数	マイナスを含む数
ソートする数	4,7,1,8,5,3,2,6	-4,7,-1,8,5,-3,-2,6
結果	1,2,3,4,5,6,7,8	-4,-3,-2,-1,5,6,7,8
並列	22	22
連鎖	24	24
単一	97	97

4.3 考察

連鎖演算などの並列性があまり高くないのでもっと効率的に計算を行うことができる余地があるように感じた。バイトニックソートのソートのデータ構造はそれぞれのスレーブで昇順、降順に並び替えてソートできた列を昇順と降順で組み合わせて複数のバイトニック列をつくっていくので２の n 乗ごとに無限に数を増やしてソートしていくことができる。

5. FPGA ボードを用いた検証

5.1 実験環境

設計した MAP を FPGA ボード上に実装を行う。実装には HSCS で開発されたプロセッサデバッガ･モニタを用いる。プロセッサデバッガの実装対象として Xillinx 社の Spartan-3 & 3E Starter Kit ボードを使用する。Spartan-3A Starter Kit に実装できるように環境を整える。プロッセッサデバッガとは FPGA ボードに実装する際に、HSCS で開発されたプロセッサデバッガ・モニタを用いる。プロセッサデバッガは、HSCS を利用して設計したプロセッサを、実機上で動作検証するためのハードウェア IP である。本モジュールと接続して論理合成をかけることで様々なプロセッサが FPGA に実装でき、ホスト PC と通信することで実行が可能である。プロセッサデバッガの利点は、FPGA ボードの LED や LCD、スイッチを使用せず、パソコン操作によって実機のデバッグを進められることである。

5.2 実験結果

今回は 16 個の数をソートするプログラムと 8 個の数をソートするプログラムを作成した。正の数でソートしたものと負の数を交えたものでソートする。結果を以下の表 5、表 6 に示す。これらの表で、並列・連鎖・単一はそれぞれの演算の実行回数が入る。

表 5：6 個のバイトニックソート

	数	マイナスを含む数
ソートする数	15,10,16,6,1,3,9,2,11,8,13,5,12,7,4,14	15、-10、-16、6、1、-3、2、-11、8、-13、5、12、-7、4、14
結果	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16	-16,-13,-11,-10,-7,-3,1,2,4,5,6,8,9,12,14,15
並列	76	80
連鎖	80	80
単一	321	321

表 6：8 個のバイトニックソート

	数	マイナスを含む数
ソートする数	4,7,1,8,5,3,2,6	-4,7,-1,8,5,-3,-2,6
結果	1,2,3,4,5,6,7,8	-4,-3,-2,-1,5,6,7,8
並列	22	22
連鎖	24	24
単一	97	97

5.3 考察

実機を使って実験した時には並列演算や連鎖演算などの結果がでないのでこれが可視化できるようにするなどまだ改良の余地はあるように感じた。メモリの範囲が 1024 までしか作成できていないので膨大な量だと実装できない。転送時データ量が多くてメモリ書き込みに時間がかかることがわかったのでプログラムを削減できた方が良い。

演算レベル並列性に関して単一実行が多く連鎖演算、並列演算が少ない。理由としてはメモリアクセスが頻繁に行われ、単一実行が必然と多くなるからである。この問題を解決するには 2 ポート DRAM を用いることによって解消できると考えられる。

6. おわりに

本論文では、本研究室で開発を進めている MAP のアセンブリプログラムの記述を行い、アセンブラ、シミュレータを通してアセンブリプログラムの検証を行った。本研究で、プログラム内の動的な並列、連鎖性を検証し、MAP の有効性を評価した。また、アセンブリ記述において、命令の順序を変えることによって並列演算や連鎖演算の命令数が変わるということも理解した。並列演算や連鎖演算をできるだけ増やすように考えながらプログラミングを行うことでより処理速度の速いプログラムを記述することができる。

本研究を通して､MAP の並列演算の効率性と複数 ALU による演算および ALU によるレジスタの共有とその制御の仕組みについて理解した。今後の課題としては、ハードウェアとソフトウェアの知識をより一層深めることと、アセンブリ記述においてプログラムの簡略化による実行時間の短縮である。

参考文献

[1] David A.Patterson　and　John L.Hennessy　著，成田光彰 訳：コンピュータの構成と設計 第三版 (上)(下)，日経 BP 社，2006.

[2] 境　直樹：デザインパターンを用いたマルチ ALU プロセッサの設計、立命館大学理工学部電子情報デザイン学科卒業論文、2010

[3] 境　直樹：MAP 仕様書、立命館大学理工学部、高性能計算研究室、2011