Small Risc Processor (SRP) 説明書

2019/12/23 ver 1.0 作成

琉球大学工学部情報工学科 和田 知久

[0] はじめに

基本課題はたった9個の命令を実行できるマイクロプロセッサです。ただ、何かのプログラム処理ができないと面白くないので、数値列を大きさの順で並べ替えるソーティング（詳細にはバブルソート）が実行できる程度の命令を含んでいます。したがって、高級プログラム言語の for ループを実現できるように、無条件 JUMP 命令と、条件付き JUMP（BRANCH）命令も含んでいます。

以下に、マイクロプロセッサをよく知らない人でも設計できるように、丁寧にアーキテクチャを説明しますので、マイクロプロセッサという言葉を恐れずに、挑戦して頂きたいと思います。

---

[1] Small RISC Processor SRP アーキテクチャ

図1に Small RISC Processor SRP アーキテクチャを示します。実は今回の SRP アキテクチャはオリジナルなものではなく、ジョン・ヘネシーとデイビッド・パターソンの有名な図書“COMPUTER ORGANIZATION & DESIGN: the hardware / software interface”で登場する RISC コンピュータの命令を9個に削減したものとなっています。また、情報工学科の講義 「計算機アーキテクチャ」とも同じものです。 したがって、かなり多数の方々が見たことのあるポピュラーな構成かと思います。

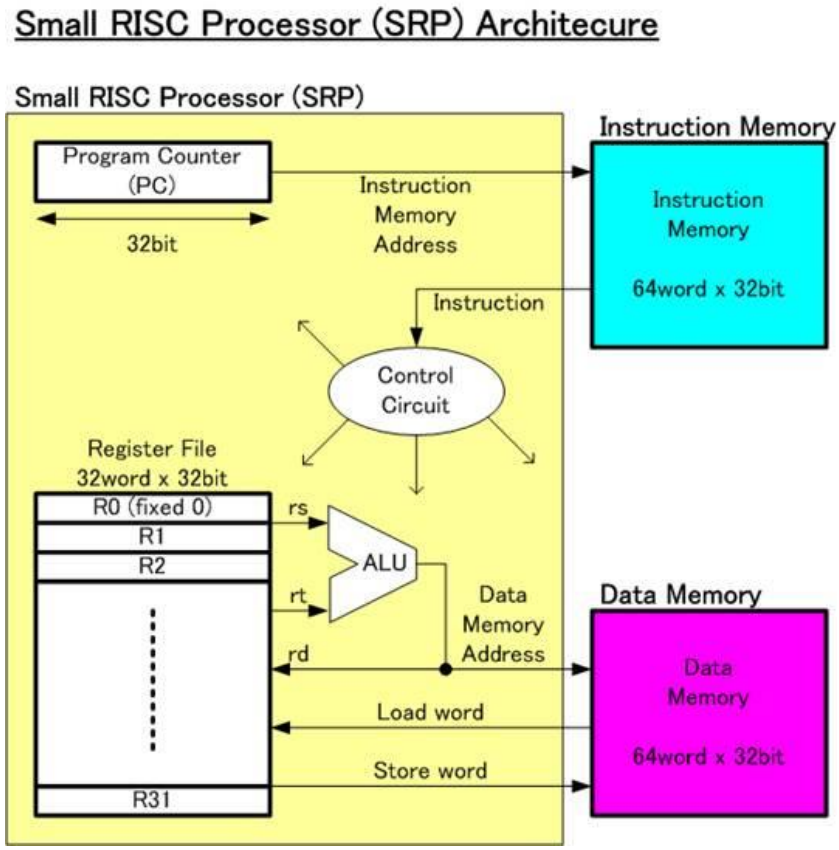

図1： Small RISC Processor SRP アーキテクチャ

図1は SRP 本体と、2つのメモリから構成されています。しかし、コンピュータを動作させるには、命令とデータが必要ですので、ここでは命令を蓄える命令メモリ（Instruction Memory）、データを蓄えるデータメモリ（Data Memory）を2つの独立したメモリで構成しています。その理由は、この方がメモリの使い方が単純化され、プロセッサの設計が単純になるからです。

後ほどのプログラムの説明で、明らかになるのですが、命令メモリには 32 ビット（ここではワードと呼ぶ）の大きさの命令が多数蓄えられています。マイクロプロセッサこの命令メモリに蓄えられた命令をひとつずつ読み出し、その 32 ビット命令が示す演算を実行します。その動作を順に説明します。

① 「命令フェッチ」：マイクロプロセッサにはプログラムカウンター（PC）というカウント機能があります。この PC の値は命令メモリのアドレス信号に対応しており、そのアドレスが示す場所の命令を読みだします。

② 「命令デコード」：上記①で読みだされた命令の内容に従って、必要な動作をするために命令の解読を実施し、必要な制御を行います。図1では control circuit の仕事となります。

③ 「演算とデータメモリアクセス」：上記命令の解読結果に従って、レジスタファイルから必要な値を読みだし、ALU にて演算します。必要あれば、データメモリに対してアクセスを行い、データメモリにデータを書き込み、もしくは読み出しを行います。

④「ライトバック」:最後に、③での演算結果や、データメモリから読みだした値をレジスタファイルに書き込みます。そして、プログラムカウンター(PC)の次のサイクルの値を用意します。

上記①から④の動作でひとつの命令が実行され、繰り返すことで多数の命令を実行することができます。

---

[2] サポート命令

以下表1に9つのサポート命令を示します。算術演算、論理演算、データ転送、条件分岐、無条件分岐の 5 種類に区分されています。

R1,R2 のような記号はレジスタファイルのアドレスを示しています。Rn ではアドレスは n です。SRP では 32 ビットレジスタを 32 個搭載しており、R0 から R31 が存在します。

[注意]SRP での通常の RISC コンピュータに従って、R0 は特別なレジスタでとしており、書き込みはすることができず、常に ALL0 の値を保持しています。ALL0 すなわち、0は、プログラムでよくつかわれる数値であるので、レジスタファイルの 0 番地をその特別な数値として使用しています。

表1　サポート命令

区分	命令	アセンブラ例	例の意味	備考
算術演算	add	add R1,R2,R3	R1 <= R2 + R3	加算
	subtract	sub R1,R2,R3	R1 <= R2 − R3	減算
論理演算	and	and R1,R2,R3	R1 <= R2 and R3	各ビットごとにAND
	or	or R1,R2,R3	R1 <= R2 or R3	各ビットごとにOR
データ転送	load word	lw R1, 100(R2)	R1 <= メモリ[R2+100]	メモリからレジスタへの転送
	store word	sw R1, 100(R2)	メモリ[R2+100] <= R1	レジスタからメモリへの転送
条件分岐	branch on equal	beq R1,R2,25	if (R1=R2) go to PC+4+25*4	等しい時に PC 相対分岐
	set on less than	slt R1,R2,R3	if (R2<R3) R1<=1 else R1<=0	
無条件ジャンプ	jump	j 2500	go to 2500*4	絶対アドレスジャンプ

表1で、R1<=R2+R3 というような表記がありますが、レジスタファイルのアドレス 2 番地に保持されている値とアドレス 3 番地に保持されている値を ALU にて加算を行い、アドレス1番に書き戻すということを意味しています。

Load word 命令はデータメモリ内の 32 ビットのデータをレジスタファイルに転送する命令です。表の例である“lw R1, 100(R2)”は、レジスタファイルのアドレス 2 番に保持されている値と命令中に指定された 100 という整数値の加算を ALU にて行い、その結果をデータメモリのアドレス値としてデータメモリから値を読みだします。そして、その値をレジスタファイルのアドレス1番に書き戻すという意味です。

逆に、“sw R1, 100(R2)”で示される Store word 命令は、レジスタファイルのアドレス1番から値を読みだし、レジスタファイルのアドレス 2 番に保持されている値と命令中に指定された 100 という整数値の加算を ALU にて行い、その結果をデータメモリのアドレス値としてデータメモリに値を書き込みます。命令の違いは最初の sw か lw かだけですが、データの転送方向は逆方向です。

次に、branch on equal ですが、これはレジスタファイル内の2つの値が同じかどうか比較して（減算してその答えが0かどうか）、同じであれば、分岐を行います。

先に説明しましたが、プログラムカウンタ PC が命令メモリ内の実行する命令のアドレス番地を示します。通常分岐がない場合に、メモリ内の次番地にある命令が実行されます。後ほど、図で説明しますが、コンピュータでは一般的に、8 ビット単位（バイト単位）のアドレスが付けられています。ここでは、命令もデータも 32 ビットを想定していますので、4 アドレス番地分でひとつの命令もしくはデータが記憶されています。したがって、次の命令の番地は現在実行中の命令のアドレス＋4となります。

さて、branch on equal の説明に戻りますが、” beq R1,R2,25”では R1 レジスタと R2 レジスタの値を比較します。同じであれば、次の命令のアドレス（現状の PC+4）値を変更します。この例では命令中に25という整数がありますが、これは25先の命令に分岐するという意味です（この数値が負の場合は戻ることになる）。これはアドレスを計算すると、PC+4+25*4 となります。1 命令の大きさが 4 アドレス分ですので、4 倍が必要なわけです。

” slt R1,R2,R3”は set on less than という命令で、R2 の値＜R3 の値であれば、R1 の値を“1”にセットし、そうでなければ“0”にリセットするものです。set on less than と branch on equalをうまく使えば多彩な条件分岐が実現できます。

最後に無条件分岐(jump)命令を説明します。先ほどの条件分岐では、PC+4 の値に対して値を加算（負の数の加算も想定）していましたが、”j 2500”はメモリでの絶対的なアドレスで 0 番地からみて、2500 個目の命令への分岐を示しています。先ほども説明しましたが、メモリのアドレスはバイト単位（バイトアドレシング）を想定していますので、実際のメモリのアドレス値では 4 倍の 2500*4=10000 番地への無条件の JUMP ということになります。

---

[3] メモリアドレッシング

まず図2にコンピュータでは一般的であるバイト単位でのアドレッシングを示します。

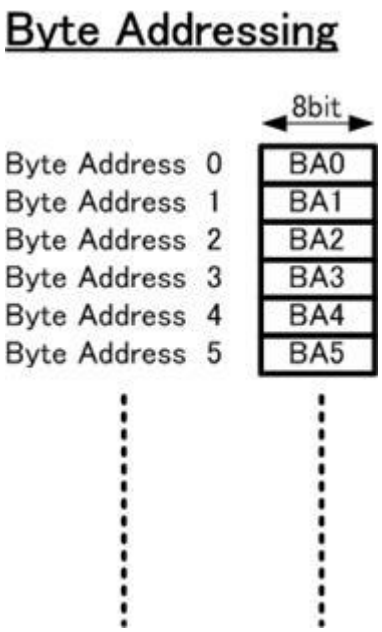

図2　バイトアドレッシング

図の右の四角は 8 ビットの記憶ブロックを意味しており、いわゆるバイトデータに対応します。SRP ではバイトアドレスを用いていますので、1 バイト進むごとに、アドレス値が1上昇します。

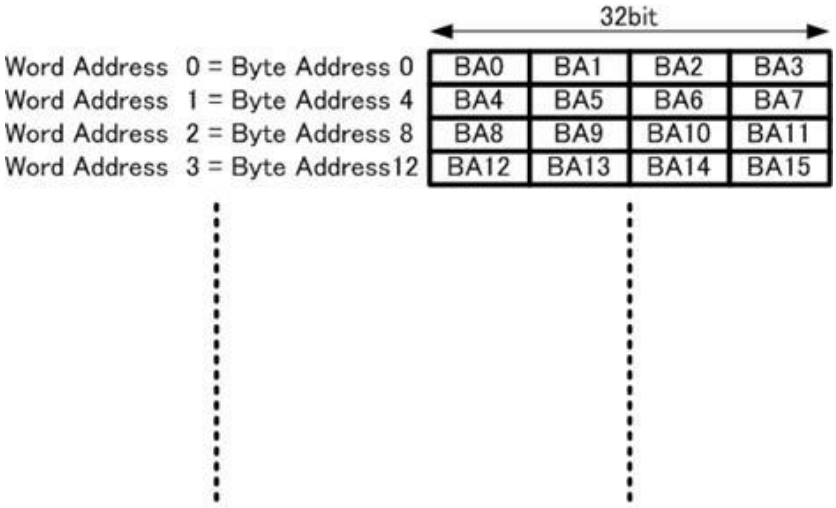

図3　ワードアドレッシング

本課題で扱う命令のサイズは4バイト（32ビット）を想定していますので、図2に示すように横方向の4つのブロックが一つの命令の記憶単位となります。したがって、命令を数えるにはワードアドレシングが便利です。このワードアドレスの値を4倍すると、4バイトブロックの左端バイトのアドレスとなります。

図1に示した SRP アーキテクチャでは2つの外部メモリがありますが、これらのメモリは32ビット単位でアドレスが振られており、メモリをアクセスする場合にはワードアドレスへの変更が必要です。

[4] 命令フォーマット

さて、これまでの話でひとつの命令は32ビットであることは理解済みであると思います。また、命令には算術演算、論理演算、分岐など色々な種類があることも説明しました。またそれだけでなく、命令にはレジスタファイルのアドレスや、メモリのアドレスの一部を示す必要もあります。
通常は上記で述べた多数の意味を示すために、32ビットを分割して使用します。その分割された部分をフィールドと言います。このようなビットでの命令の表現をアセンブラー表現もしくは、機械語表現と言います。以下表2を用いて命令の機械語表現を説明します。

表2　命令フォーマット（各フィールドの値は 10 進数で示してある。）

命令	フォーマット名	例						備考
		6 ビット	5 ビット	5 ビット	5 ビット	5 ビット	6 ビット	
add	R形式	0	2	3	1	0	32	add R1, R2, R3
sub		0	2	3	1	0	34	sub R1, R2, R3
and		0	2	3	1	0	36	and R1, R2, R3
or		0	2	3	1	0	37	or R1, R2, R3
slt		0	2	3	1	0	42	slt R1, R2, R3
lw	I形式	35	2	1	100			lw R1, 100(R2)
sw		43	2	1	100			sw R1, 100(R2)
beq		4	1	2	25			beq R1, R2, 25
j	J形式	2	2500					j 2500

まず、32ビットを表2のように分割しますが、その前に9つの命令を3つの形式に分類しています。R 形式はレジスタアドレスを3つ示す必要のある命令用の形式で、add, sub, and, or, slt がこの形式を用います。レジスタは0番から31番の32種類があるので、5ビットを用いてレジスタのアドレスを示しています。また、R 形式では左端の6ビットがすべて0であり、これが R 形式であることを示しています。add, sub, and, or, slt の区別は右端の6ビットの値で区別されています。また、右から2つめの5ビットフィールドは用いられていません。
次に I 形式ですが、これはレジスタアドレスを2つと、数値（2の補数表現を用いて、正もしくは0もしくは負の数を示す）を示しています。数値はなるべく大きな数値を示すことができる方が良いのですが、トータル32ビットの制限で、数値は16ビットです。2の補数表現を用いていますので、表3が表現可能な数値の範囲となります。
表3　16ビット（2の補数）の表現範囲

	2進数（2の補数表現）	10進数
最大値	0111 1111 1111 1111	32767
0	0000 0000 0000 0000	0
最小値	1000 0000 0000 0000	−32768

この16ビットフィールドは lw, sw ではデータメモリアドレスの計算（バイト単位）に用いられており、beq では命令ワードの分岐数を示しています（ワード単位）。

最後に J 形式ですが、J 形式でレジスタアドレスを示す必要がないので、26ビットの数値が用いられています。この数値を4倍すると、絶対メモリアドレスとなるので、負の数を表現する必要はなく、符号なし数で正または0が表現範囲です。
備考に、表1に対応する命令の例があり、表2に実際のフィールドの値（10進数表示）があり、機械語（表2）でのフィールドの順序が異なることに注意が必要です。

各フィールドには名前があり、それを表4に示す。

表4　フィールド名

フォーマット名	6 ビット	5 ビット	5 ビット	5 ビット	5 ビット	6 ビット	備考
R形式	op	rs	rt	rd	−	func	
I形式	op	rs	rt	offset			
J形式	op	address					

op フィールドと func フィールドが操作を指定している。rs はソースレジスタ、rt も通常はソースレジスタ、rd はデスティネーションレジスタと呼ぶ。

---

[5] 命令 ROM とデータ RAM のサイズに関して

図1で2つのメモリを示したが、今回の課題では与えられたプログラムを実行することを想定し、以下の図4に示すメモリ構成を想定する。
命令 ROM は64ワード X32ビット構成であり、256バイトの大きさを持つ。0番地より255番地のアドレスを命令 ROM に割り当てる。
データ RAM も同じく64ワード X32ビット構成であり、256バイトの大きさを持つ。256番地より511番地のアドレスを割り当てる。
512番地以上のアドレスは不使用とする。

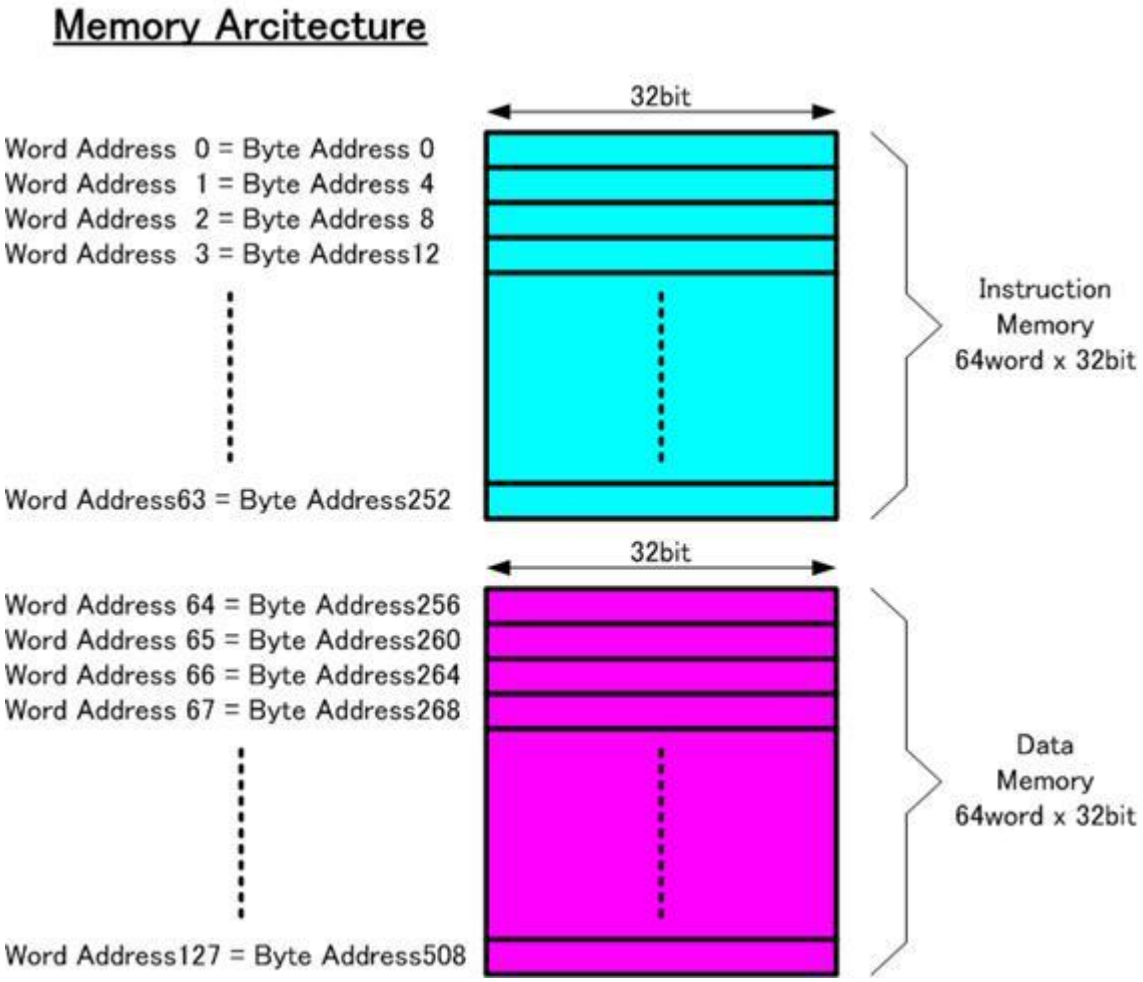

図4　メモリアーキテクチャ

---

[6] サポート VHDL コード

検証として、上記 SRP プロセッサを HDL にて実現し、8ワード（ワード＝4バイト）のソーティング（バブルソート）を実行します。
バブルソートプログラムが入った命令 ROM と初期データが格納されたデータ RAM の VHDL モデルを以下に与えます。
Reset_b 信号が0->1 に解除された後、プログラムカウンタ：PC=0 番地として、実行を開始します。

DRAM のデータをモニターし、以下のようなシミュレーション波形を確認できればバブルソートができています。

使用パッケージ

命令 ROM

データ RAM

MPU

テストベンチ

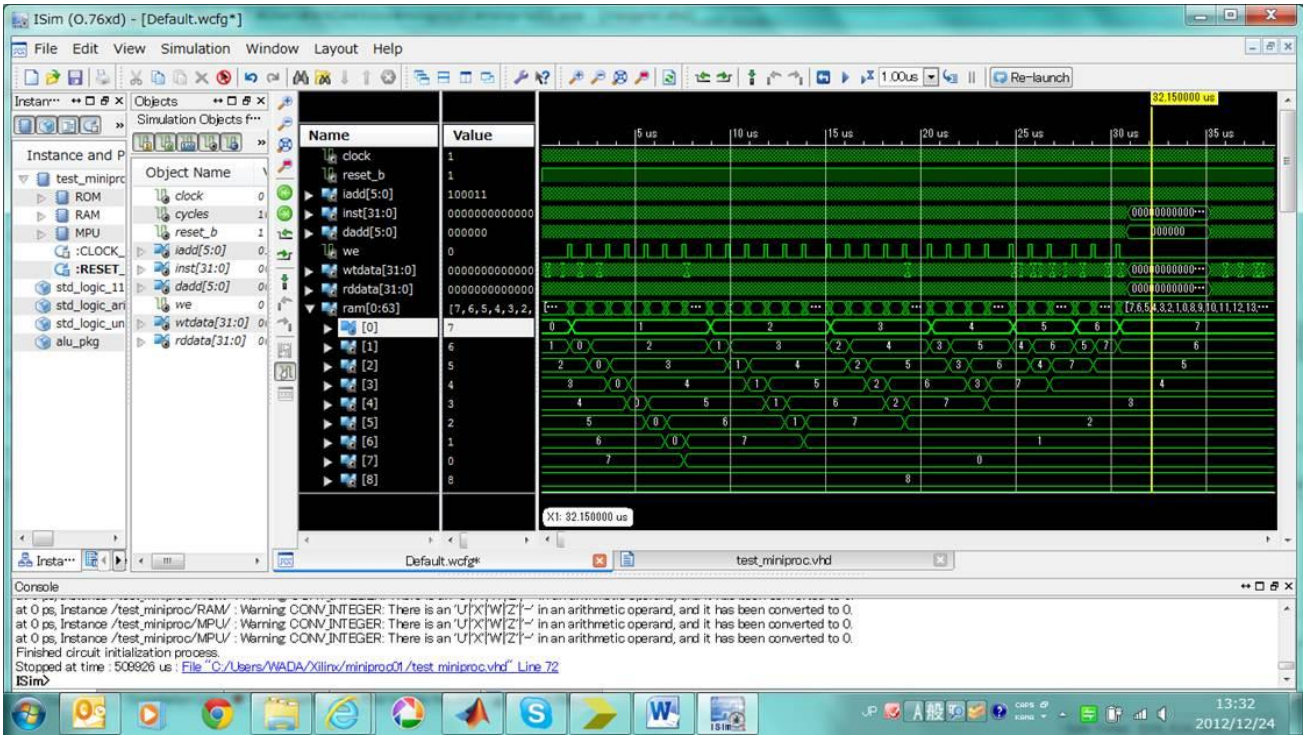

図5　バブルソート実行波形例

---

[7] バブルソートプログラム

バブルソートでは256番地から始まる8ワードのアレイに対して値の大きい数値を下位のアドレスからソートする。アレイの先頭番地は384番地のワードに記憶されている。アレイの大きさはバイトを単位として388番地に記憶されている。ここでは8x4＝32である。ワードの大きさをバイト単位とすれば、4でありこれが392番地に記憶されている。

表5　データ RAM の内容

		データ RAM					ソート後の内容			
バイトアドレス	dram の Add 入力	00	01	10	11		00	01	10	11
256	0	0				->	7			
260	1	1				->	6			
264	2	2				->	5			
268	3	3				->	4			
272	4	4				->	3			
276	5	5				->	2			
280	6	6				->	1			

284	7	7	->	0
		…		
384	32	256		256
388	33	32		32
392	34	4		4
		…		

表6　命令 ROM の内容（バブルソートのプログラム）

		命令 ROM の内容				
バイトアドレス	irom の Add 入力	00	01	10	11	説明
0	0	NOP				
4	1	NOP				
8	2	LW R1, 388(R0)				R1 <= 32
12	3	LW R2, 384(R0)				R2 <= 256, 先頭番地
16	4	LW R3, 392(R0)				R3 <= 4
20	5	ADD R5, R1, R2				R5 <= 288, 最終番地の次
24	6	SUB R5, R5, R3				R5 <= R5 - 4 = 284, 最終番地
28	7	ADD R6, R2, R0				R6 <= R2 = 256　転送、R0 はO固定
32	8	ADD R7, R6, R0				R7 <= R6、比較するデータ1のアドレス
36	9	ADD R8, R7, R3				R8 <= R7 + 4、比較するデータ2のアドレス
40	10	LW R10, 0(R7)				R10 <= 比較するデータ1
44	11	LW R11, 0(R8)				R11 <= 比較するデータ2
48	12	SLT R9, R10, R11				R10 < R11 なら R9 <= 1、違うなら R9 <=0
52	13	BEQ R9, R0, +2				R9 がOならば PC=PC+4+2*4=64 へ分岐
56	14	SW R10, 0(R8)				比較データ1をスワップしてメモリへ
60	15	SW R11, 0(R7)				比較データ2をスワップしてメモリへ
64	16	ADD R7, R7, R3				R7 <= R7 +4, アレイインデックスを進める
68	17	BEQ R7, R5, +1				インデックスが最終アドレスなら LOOP1を抜ける
72	18	J 9				9番地へジャンプ、LOOP1
76	19	SUB R5, R5, R3				R5 <= R5 - 4, 最終番地を下げる。ここに最小値がある。
80	20	BEQ R5, R2, +1				最終番地が先頭番地と一致すると LOOP2 を抜ける。
84	21	J 8				8番地へジャンプ、LOOP2
88	22	NOP				

---

[8] データ RAM の動作

データ RAM はデータの読み出しに関しては、アドレス入力信号6ビットに対して、32ビットデータを出力する組み合せ回路である。
書き込みに対しては、Clock 入力の立ち上りエッジで WE 信号が’1’の時に書き込みが行われる。

詳細の動作波形を以下に示す。

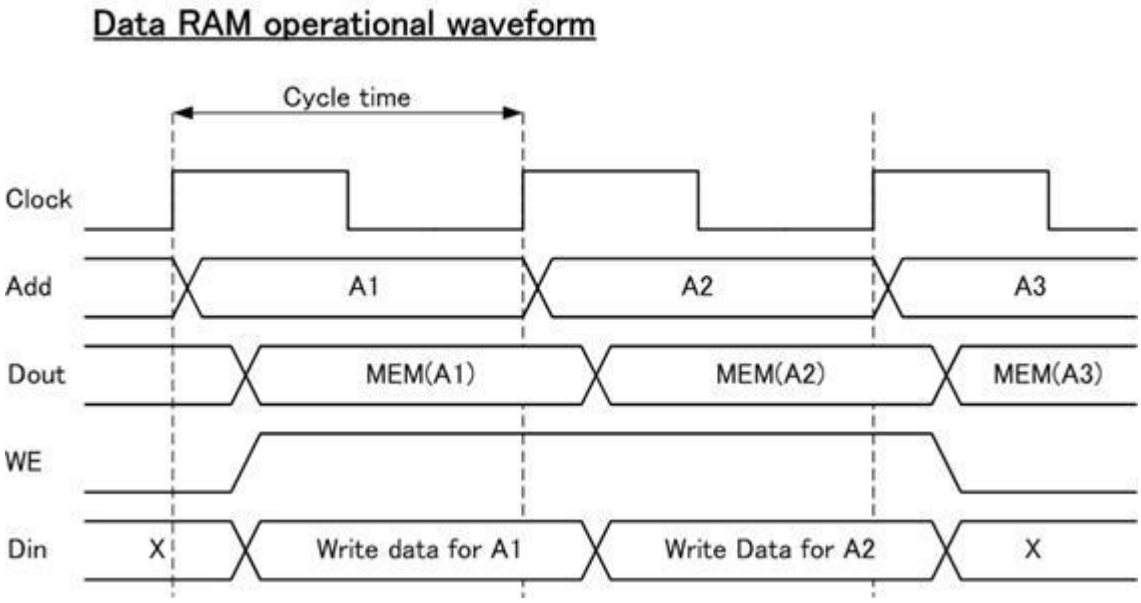

図6　データ RAM 動作波形

---

[9] SRP 詳細構成

SRP の設計アーキテクチャの典型的な例を説明する。1命令を1クロックサイクルごとに実現する例である。
命令 ROM はワードアドレスを入力されると、命令が読みだされる ROM を想定している。SRP 内部のレジスタファイルで ra1, ra2 は2つの読み出し用アドレス信号であり、このアドレスに従って、do1, do2 にレジスタの値がそれぞれ読みだされる。wa は書き込みアドレス信号であり、書き込むデータは din に与えられる。writeRF=1 で、Clock の立ち上がりエッジでデータがレジスタファイルに書き込まれる。
レジスタファイルへのデータの書き込みが必要な命令は add, sub, and, or, lw, slt である。表7にまとめる。
表7　レジスタファイルへのデータの書き込みが必要な命令と読み出しデータ数

区分	命令	レジスタファイルへのデータの書き戻し	レジスタファイルより読みだす値
算術演算	add	1	2
	subtract	1	2
論理演算	and	1	2
	or	1	2
データ転送	load word	1	1
	store word	0	2
条件分岐	branch on equal	0	2
	set on less than	1	2
無条件ジャンプ	jump	0	0

ALU は add, sub, and, or の演算サポートし、その演算結果(result)が0であるかどうかを調べるフラグ(zero)を計算する。slt 命令では、ALU は減算を実行し、zero フラグを調べることで、実装可能である。
データ RAM への書き込みが必要な命令は sw 命令であり、データ RAM の読み出しが必要な命令は lw である。
SRP で△マークは Clock 入力を意味している。クロックの立ち上りエッジで書き込みが発生する順序回路であり、△マークのない他の回路はすべて組合せ回路で構成できる。
”＆”はビット連結、”＋”は加算器、”MUX”はマルチプレクサ、”SE”は符号を保存したビット幅の16 ビットから 32 ビットへの拡張回路である。以下、表 8 に拡張の例を示す。16 ビット表現での MSB ビットを 16 回左側に延ばせば実現できる。

表8　符号を保存したビット幅の 16 ビットから 32 ビットへの拡張の例

	16 ビット（2の補数表現）	32 ビット（2の補数表現）
最大値	0111 1111 1111 1111	0000 0000 0000 0000 0111 1111 1111 1111
0	0000 0000 0000 0000	0000 0000 0000 0000 0000 0000 0000 0000
最小値	1000 0000 0000 0000	1111 1111 1111 1111 1000 0000 0000 0000

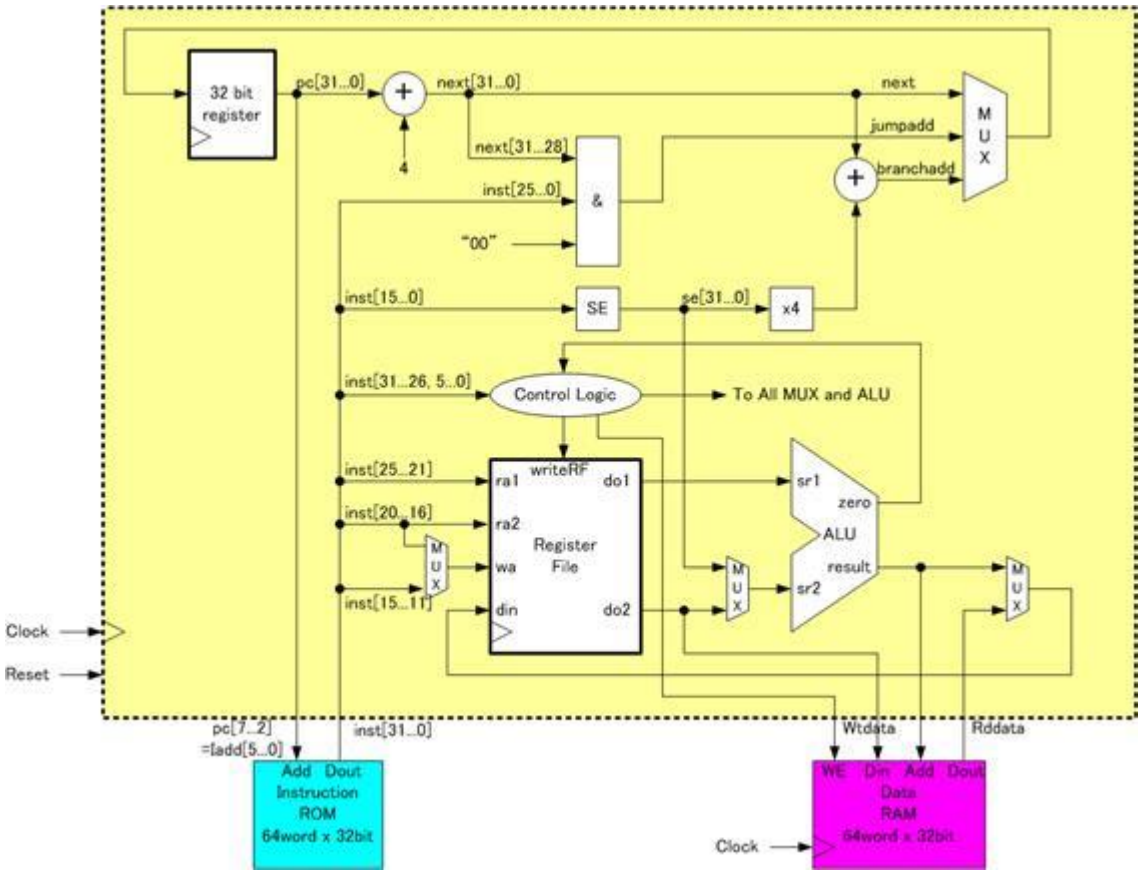

図7　SRP 構成例

図7の右上の3入力 MUX は分岐に関するものである。3入力の一番上が選択される場合は通常の分岐がない場合であり、pc 値は4ずつインクリメントしている。3入力の中央は無条件分岐に対

応している。その前方の“＆”はビットの接続であり、下記に”00”を連接することにより、4倍を実現している。また、上位のビットの不足分4ビットはもとの PC の上位ビットをコピーしている。3入力の一番下は、条件付き分岐が成立した（TAKEN）場合である。PC+4+offset*4 の計算になっている。

図8に、32番地、36番地、40番地を実行中の動作波形を示す。pcの値により、Iadd が生成され、命令 Inst が命令 ROM より取り出されている。命令の各フィールドの値より、ra1, ra2, wa が生成されていることがわかる。

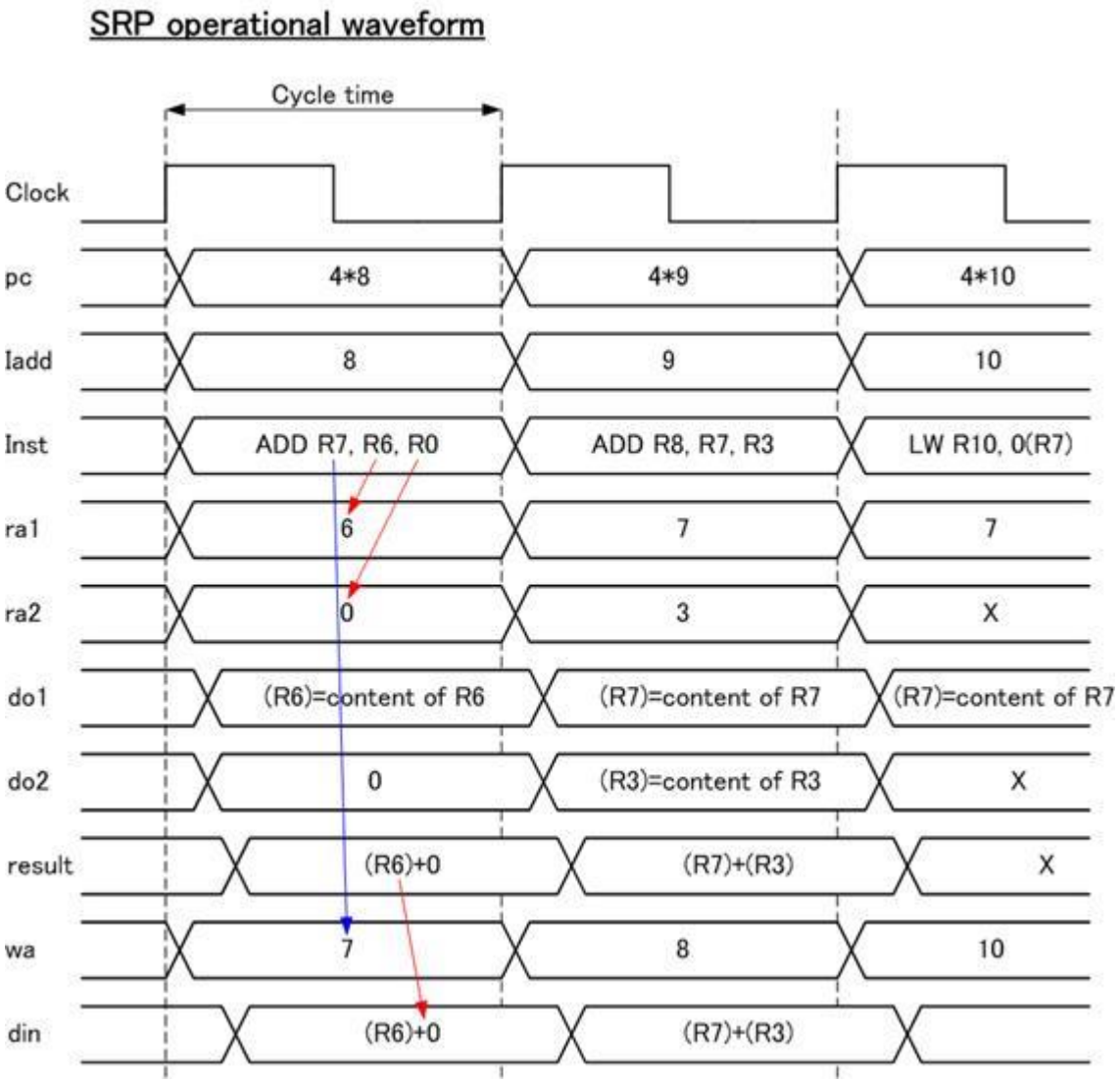

図8　SRP 動作波形図