実験４：マイクロプロセッサの設計と実装

03-120480 島津真人

2012 年 12 月 12 日

1　概要

この実験では、verilogHDL を用いて IA-32 のサブセットである IA-32z の実装を行った。その際作成したソースは以下の URL で git で公開されている。

git://ccv.kuroaka.net/experiment/cpu/cpu

結果としては、IA-32z の命令をすべて実装し、パイプライン化まで行うことができた。esp 以外はデータフォワーディングを行えるように実装したので、データハザードは (push/pop が連続する場合を除いて) 起こらない。制御ハザードについては、分岐しない場合にも命令を一度クリアするためすべての分岐において生じてしまう。動作確認としては、すべての命令を実行した他、フィボナッチ数列を求めること、PUSH/POP を伴う関数へのジャンプなどを行った。実装したフェーズフローチャート、データフローチャートを図 1、図 2 に示す。

	F	R	X	M	W
zLD	MD->IR1 PC+4->PC	RD2->TR	TR+sim8->MA	MD->MDR_R	MDR_R->RD1
zST		RD1->SR1 RD1->MDR_W RD2->TR	TR+sim8->MA MDR_W->MD	-	-
zLIL		-	imm16->DR1	DR1->DR2	DR2->RD2
zMOV		RD1->SR1	SR1->DR1		
reg-reg		RD1->SR1 RD2->TR	TR op SR1->DR1		
reg-imm		RD2->TR	TR op imm->DR1		
zNEG		RD2->TR	op TR->DR1		
zNOT					
zSLL/SLA/ zSRL/SRA		RD2->TR	TR<<sim8->DR1		
zB		PC->TR	TR+sim8->DR1	DR1->ct_PC	ct_PC->PC ct_PC->MA
zBcc					
zJR		-	-	SR2->ct_PC DR1->DR2	ct_PC->PC ct_PC->MA DR2->wesp
zJALR		resp->TR PC->MDR_W RD2->SR1	MDR_W->MD SR1->MA SR1->SR2 TR-4->DR1		
zRET		resp->TR	TR+4->MA TR+4->DR1	MD->ct_PC DR1->DR2	
zPUSH		RD2->MDR_W resp->TR	MDR_W->MD TR-4->MA TR-4->DR1	DR1->DR2	DR2->wesp
zPOP		resp->TR	TR+4->MA TR+4->DR1	MD->MDR_R DR1->DR2	MDR_R->RD1 DR2->wesp
zNOP		-	-	-	-
zHLT		-	-	-	-

図 1　Phase Flow Chart

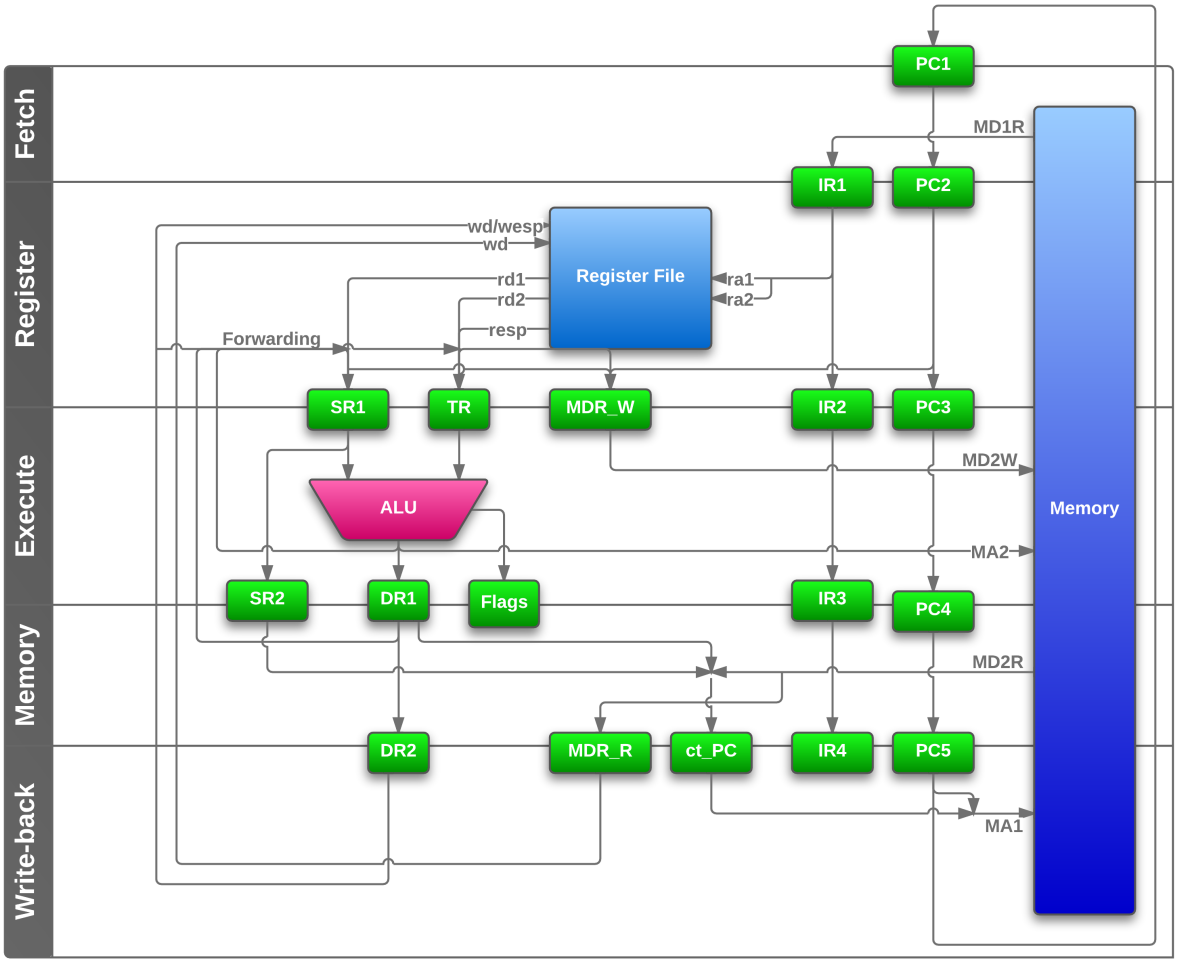

図 2　Data Flow Chart

2　特徴

2.1　トップモジュール

トップモジュールだけではなく全体を通してであるが、function を適切に用いることで組み合わせ回路と順序回路を意識した書き方を行うように意識した。

2.2　ALU

できるだけ演算は ALU で行うように実装した。通常の reg-reg や reg-imm だけではなく、スタックポインタの値や、各種 sim8 の足し算などもここで行なっている。また、フラグもここで出力するようにしている。ここで、キャリー・ボローの検出ために 1 ビット拡張するという工夫を行った。足し算の際には、最上位ビットに 0 を付け足し、足し算してキャリーしたらそこが 1 になることで検出できる。引き算の際には引かれる側に 1 を付け足し、引く側には 0 を付け足すことで、最上位ビットが 1 であればボローなし、0 であればボロー、という判断ができる。ちなみに、シフトや zNEG がやや特殊なフラグの更新を行うが、これも実装されている。

2.3 PC

プログラムカウンタの内部のブロック図を図 3 に示す。セレクタでは、ct_pc の条件である (ct_taken,next_w,next_f) = (1,1,0) が優先的に評価され、その次に next_f==1 が評価され、それ以外なら pc=PC となるようになっている。プログラムカウンタの動作としては、図 2 のデータフローチャートからもわかるように W フェーズで分岐の処理が行われ、F フェーズに移る際には出力が正しくなっている必要がある設計となっている。ゆえに条件分岐においてレジスタが絡んではいけないため、内部で w フェーズでは出力は組み合わせ回路から直接出し、同時に並行して内部の PC のレジスタにも書き込むようになっている。これに関しては組み合わせ回路とレジスタからの出力が切り替わるという点であまり綺麗でないように感じるが、良い解決策が思いつかなかった。

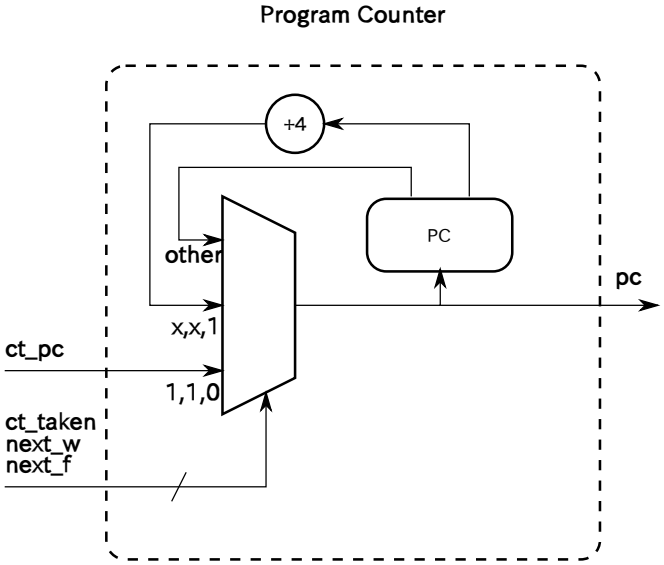

図 3 Program Counter

2.4 レジスタファイル

これは、ほぼ与えられたものを利用した。ただし、スタックポインタの値 (esp) と他の普通のレジスタの書き込みを同時に行う必要があるため、2 入力 (片方が esp 専用)3 出力 (一つは esp 専用) とした。片方を esp 専用とすることで、もう片方が esp を書き込まない限り 2 重で書き込むことはない。その制限を本来ならハードウェアで加えるべきであるが、今回はそこまで実装しなかったためにコンパイラ、もしくはアセンブリを書くさいに気をつける必要がある。

2.5 パイプライン

パイプライン化するにあたって、データハザードがあった場合には適切にデータフローを止める必要がある。それを行うモジュール hazard detector のブロック図を図 4 に示す。「Detect *」と書いてある部分で、そのステージにおいてデータを止める必要があるかを判断し、そのあとで後段部分でのハザードの論理積を取ることでデータフローを適切に処理している。出力はデータが流れる場合は 1, 止める場合は 0 である。その際、論理積部分の入力として後段の出力を回すのではなく、Detect 部分での出力を直接用いることにより遅延が少なくなるようにしてある。なお、ここで止めた場合には後段の IR には NOP を入れるようになっている。また、図 4 の下段にある「Forwarding 1/2」では、次の X ステージの入力に必要となるデータがどのステージにあるかを検出している。この出力結果に応じて、TR/SR に入れるデータを選ぶようになっている。

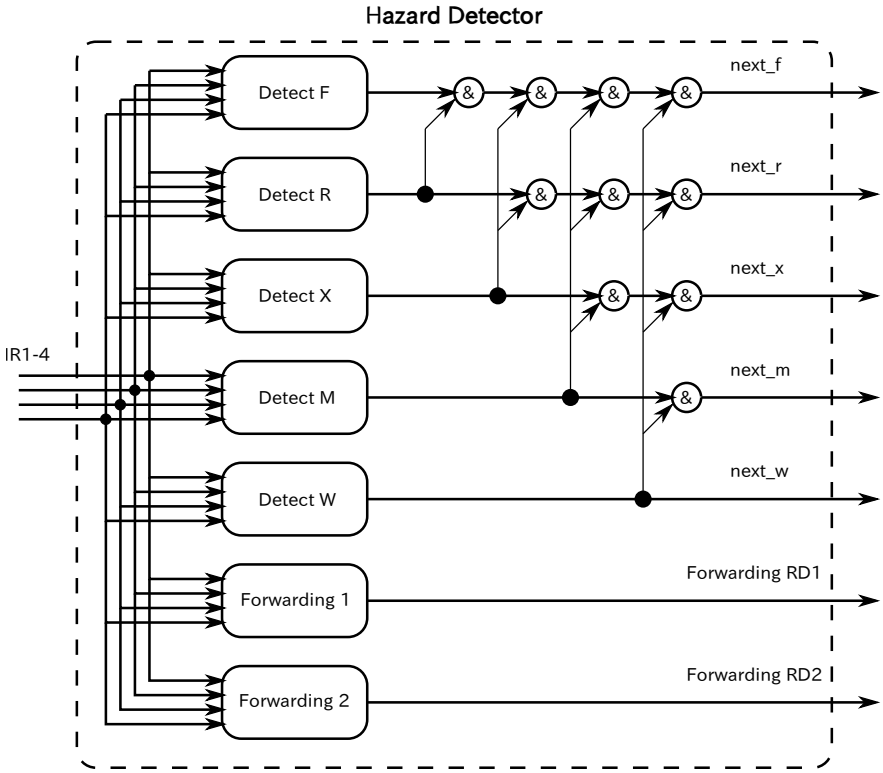

図 4　Hazard Detector