6.823 Computer Architecture	DUE: 4/29/2002

Group 18: Timothy Danford, Kilian Pohl, Bryan Russell

Problem Set 5 — *Instructors: Krste Asanovic, Srinivas Devadas*

Problem 1

(a)

In the following code, R2 is the pointer to the current element, and R5 is the final value of that pointer (so that we can check when the loop has ended).

```
       SLT   R6, R0, R1
       BEQZ  R6, end_loop   ;; ensure that i < N when we loop.
       SLLI  R5, R1, #2
       ADD   R5, R5, R2     ;; setup final value
       LW    R3, 0(R2)
loop:  ADDI  R2, R2, #4     ;; i = i + 1
       SLT   R4, R3, R0     ;; is A[i] <= 0?
       BEQZ  R4, final
       SLT   R6, R2, R5     ;; (i+1) < N?
       SUB   R3, R0, R3     ;; A[i] = -A[i]
       SW    R3, -4(R2)
final: BNEZ  R6, loop
       LW    R3, 0(R2)      ;; get the next A[i]
end_loop:
```

The loop takes 6 or 8 cycles depending on whether the current element is negative, for an average of 7 cycles per element.

(b)

For shortness we've ommitted the setup code here. The meanings of the registers are the same as from part A though: R2 is the address of A[i], and R5 is the final address when the loop is finished. We can do a little modulo arithmetic in the setup code (not part of the loop) to make sure that we only are running the loop on a list with an even number of elements.

```
       LW    R3, 0(R2)    ;; later dup'ed in BNEZ delay slot
loop:  LW    R7, 4(R2)    ;; R3, R7 hold A[i], A[i+1]
       SLT   R4, R3, R0   ;; A[i] <= 0?
       BEQZ  R4, first
       SLT   R8, R7, R0   ;; A[i+1] <= 0?
       SUB   R3, R0, R3   ;; if A[i] <= 0, A[i] = -A[i]
       SW    R3, 0(R2)
```

```
first: BEQZ  R8, scnd
       ADDI  R2, R2, #8    ;; i = i + 2
       SUB   R7, R0, R7    ;; if A[i+1] <= 0, A[i+1] = -A[i+1]
       SW    R7, -4(R2)
scnd:  SLT   R6, R2, R5    ;; i == N?
       BNEZ  R6, loop      ;; if not, then continue...
       LW    R3, 0(R2)
end_loop:
```

The loop has a total length of 13 cycles, but 4 cycles are executed conditionally, for a total of 11 (on average) cycles per loop. Each iteration of the loop handles two elements of the list. Therefore, for long lists, the average number of cycles per element is $5\frac{1}{2}$.

(c)

The same conventions apply as 1A above. This time we've included the (short) setup code again.

```
      SLT    R6, R0, R1
      BEQZ   R6, end_loop
      SLLI   R5, R1, #2
      ADD    R5, R5, R2
loop: ADDI   R2, R2, #4
      LW     R3, -4(R2)
      SLT    R6, R2, R5
      CMPLTZ P0, R3
      SUB    R3, R0, R3
      BNEZ   R6, loop
 (P0) SW     R3, -4(R2)
end_loop:
```

Every iteration of the loop takes exactly 7 cycles, and handles exactly 1 element. Therefore, each element is processed in (on average) 7 cycles.

(d)

The same conventions for register usage apply, as above. Comments are omitted since this is basically just a re-arranged version of the previous problem's code. It's worth noting though, that the 3^{rd} and last lines are the same, analogous to the repeated loads in earlier answers.

	Slot 1	Slot 2
	SLT R6, R0, R1	SLLI R5, R1, #2
	BEQZ R6, end_loop	ADD R6, R5, R2
	LW R3, 0(R2)	ADDI R2, R2, #4
loop:	SLT R6, R2, R5	
	CMPLTZ P0, R3	SUB R3, R0, R3
	BNEZ R6, loop	(P0) SW R3, -4(R2)
	LW R3, 0(R2)	ADDI R2, R2, #4
end_loop:		

The loop now takes 4 cycles to complete, and processes 1 element per iteration, for an average of 4 cycles per element.

(e)

As in 1B, we've omitted the prolog code. Basically, the prolog code sets up R8 as the memory boundary (in previous problems, this was R5) to signal the end of the loop. R2 and R5 are the addresses of the i and $i+1$ elements of the array, respectively. P0 and P1 hold the predicate bits for whether the contents of the i and $i+1$ elements are negative. We also assume that the prolog code handles odd-length arrays (processing the first element singly, if necessary), so that this loop code is only running on even-length lists. This is the reason why, in the 4^{th} line, we only need to check if R5 has run over the boundary R8.

	Slot 1	Slot 2
loop:	LW R3, 0(R2)	LW R6, 0(R5)
	ADDI R2, R2, #8	ADDI R5, R5, #8
	CMPLTZ P0, R3	CMPLTZ P1, R6
	SUB R3, R0, R3	SLT R7, R5, R8
	BNEZ R7, loop	SUB R6, R0, R6
	(P0) SW R3, -8(R2)	(P1) SW R6, -8(R5)

The loop takes 6 iterations to complete every time, but each iteration of the loop handles 2 elements of the array. Therefore, each element of the array is processed in 3 cycles, on average.

(f)

If Ben knows N ahead of time, he can completely avoid the need to use branches, or keep counts of the number of iterations. For each element of the array, at least four operations need to occur: the load, the check if the element is negative, the change from negative to positive, and the store. Assuming that we're still operating in the two operation VLIW instruction realm, with predicated instructions, then the four operations for two successive elements of the array can execute simultaneously. This means that each element will be processed in (on average) 2 cycles.