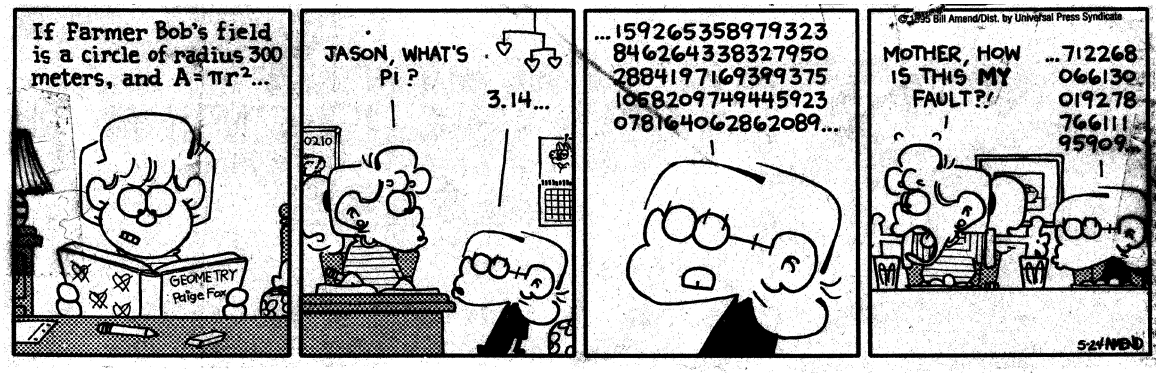
Introduction to Computer Engineering (E85)

Harris Spring 2001

Problem Set 7 **Due: Friday, March 30**



1) Procedure Calls

Ben Bitdiddle is trying to compute f(a,b) = 2*a+2*b (for nonnegative b). He goes overboard in the use of procedure calls and recursion and produces the following code.

```
int f(int a, int b)
{
      int j;

      j = a;
      return j + a + f2(b);
}
```

```
int f2(int x)
{
      int k;

      k = 2;
      if (x == 0) return 0;
      else return k + f2(x-1);
}
```

Ben then translates the two procedures into assembly language, using the following register assignments:

f: $a0 <= a, $a1 <= b, $s0 <= j f2: $a0 <= x; $s0 <= k

```
0x04000000   test:  addi $a0, $0, 5     # a = a0 = 5
0x04000004          addi $a1, $0, 3     # b = a1 = 3
0x04000008          jal f               # call f(5,3)
0x0400000c   loop:  j loop              # and loop forever

0x04000010   f:     addi $sp, $sp, -16  # make room for $s0, $a0, $a1, and $ra
0x04000014   ?      sw $s0, 0($sp)      # save $s0
0x04000018   ?      sw $ra, 4($sp)      # save return address
0x0400001c   ?      sw $a0, 8($sp)      # save a
0x04000020   ?      sw $a1, 12($sp)     # save b
0x04000024          add $s0, $a0, $0    # j = a
0x04000028          add $a0, $a1, $0    # place b as argument for f2
0x0400002c          jal f2              # call f2(b)
0x04000030   ?      lw $a0, 8($sp)      # restore a after procedure call
0x04000034   ?      lw $a1, 12($sp)     # restore b after procedure call
0x04000038          add $v0, $v0, $s0   # j + f2(b)
0x0400003c          add $v0, $v0, $a0   # j + a + f2(b)
0x04000040   ?      lw $ra, 4($sp)      # restore return address
```

```
0x04000044   ?       lw $s0, 0($sp)      # restore $s0
0x04000048           addi $sp, $sp, 16   # restore stack pointer
0x0400004c           jr $ra              # and return

0x04000050   ?f2:    addi $sp, $sp, -12  # make room for $s0, $a0, and $ra
0x04000054   ?       sw $s0, 0($sp)      # save $s0
0x04000058   ?       sw $ra, 4($sp)      # save return address
0x0400005c   ?       sw $a0, 8($sp)      # save x
0x04000060           addi $s0, $0, 2     # k = 2
0x04000064           bne $a0, $0, else   # x = 0?
0x04000068           addi $v0, $0, 0     # yes: return value should be 0
0x0400006c           j done              # and clean up
0x04000070   else:   addi $a0, $a0, -1   # x-1
0x04000074           jal f2              # call f2(x-1)
0x04000078   ?       lw $a0, 8($sp)      # restore x
0x0400007c           add $v0, $v0, $s0   # k + f2(x-1)
0x04000080   ?done:  lw $ra, 4($sp)      # restore return address
0x04000084   ?       lw $s0, 0($sp)      # restore $s0
0x04000088   ?       addi $sp, $sp, 12   # restore stack pointer
0x0400008c           jr $ra              # and return
```

Notice that the code saves and restores a bunch of registers on the stack on the lines annotated with Greek characters. In this problem, we will explore why these saves and restores are necessary for procedure calls and the consequences of omitting them.

a) If we run the code starting at `test`, what value is in $v0 when the program gets to `loop`? Does his program correctly compute 2*a + 2*b?

b) Suppose Ben deletes the lines marked with ?? that save and restore $ra. When we begin f, $ra points to `0x0400000c`, the instruction after `jal f`. When function f does the jump and link to f2, $ra points to `0x04000030`, the instruction after `jal f2`. When f2 returns, we eventually get to the `jr $ra` at the end of f. (`0x0400004c`). Because f2 fails to restore $ra, we jump back to `0x04000030` rather than `0x0400000c`. Thus, we get into an infinite loop from `0x04000030` to `0x0400004c`. On each iteration of the loop, the stack pointer moves up 16 bytes. Eventually, it will point to an illegal memory address and the program will crash.

Now suppose Ben instead deleted the lines marked with ? that saves and restores $a0. Will the program (i) enter an infinite loop but not crash; (ii) crash; (iii) produce an incorrect value in $v0 when the program returns to loop (if so, what value?), or (iv) run correctly despite the deleted lines.

c) Repeat part b for each of deleting the lines marked with ? ?? ?? ?? ?? ?? .

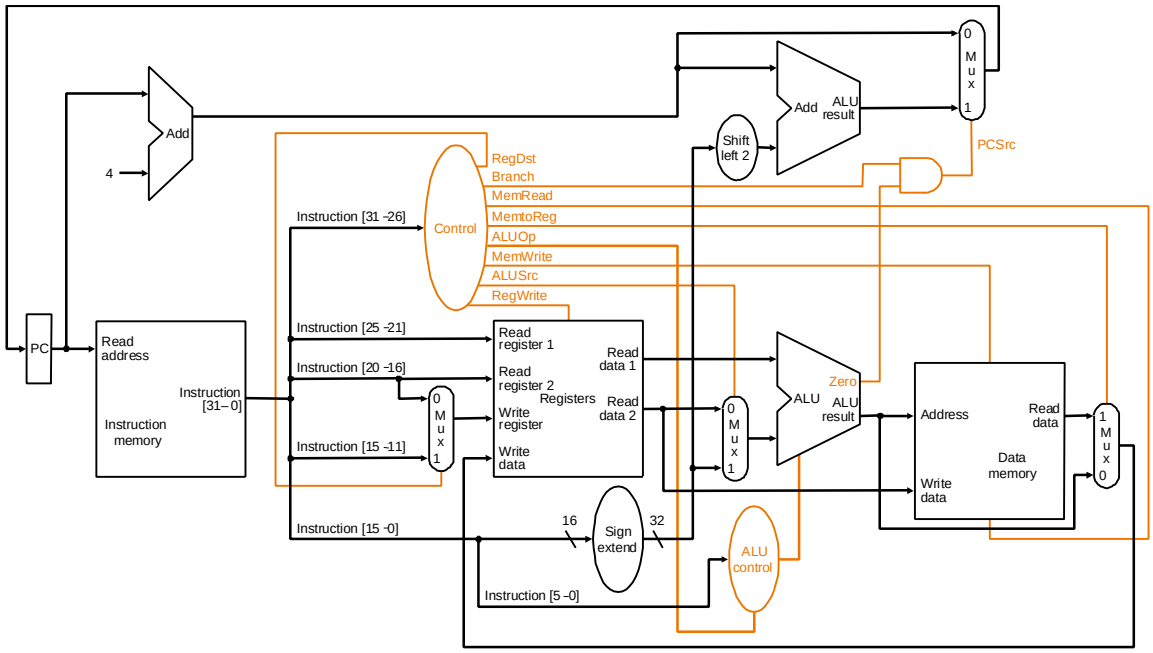
2) Enhancing the MIPS Processor (from E114 Midterm 2, Spring 1999)

The MIPS hardware we have constructed in class does not support `bne`. Modify the single-cycle processor datapath shown on the next page to handle `bne`. Also, complete the truth table for the controller. If you need to add a control signal, do so in the blank column.

3) Time

Please indicate how many hours you spent on this problem set. This will not affect your grade, but will be helpful for calibrating the workload for next semester's class.

Datapath



Controller

Op[5:0]	Instruction	Control Outputs									
		RegDst	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	Jump	*ALUOp[1:0]*	
000000	R-type	1	0	0	1	0	0	0	0	*10*	
100011	lw	0	1	1	1	1	0	0	0	*00*	
101011	sw	X	1	X	0	0	1	0	0	*00*	
000100	beq	X	0	X	0	0	0	1	0	*01*	
000010	j	X	X	X	0	0	0	0	1	*00*	
	bne										