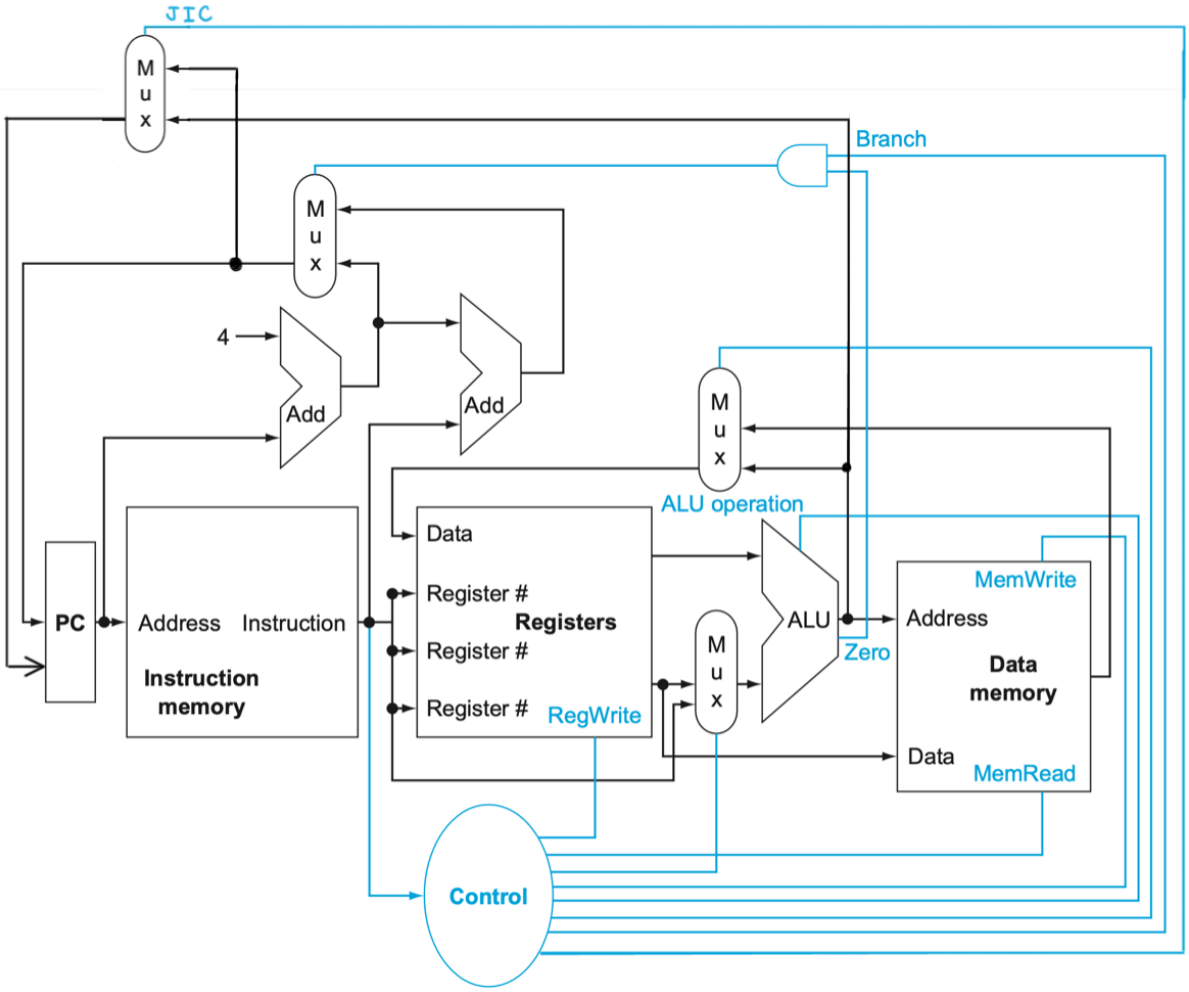
1.

a. Instruction starts are Instruction Memory and then needs the Control Unit to set control signals. Since it uses a register, it would need the Register File to get the value of the register specified in the instruction. Then, the immediate value is added to the register, which requires the Adder, ALU, and ALUSrc Mux. The three components mentioned are needed to select an immediate value addition. It also rewrites the PC value, so the PC is required, a register that holds the current instruction address.

b. A 2-to-1 Mux and a new control signal is needed. The Mux should be placed after the Mux used after the branch component. The first input for this new Mux would be the output of the Mux used after the branch component, and the second would be the ALU output.

c. When the new instruction is called, a new control signal (JIC) should set this new Mux to 1, setting the PC to the value of the register and the immediate. **Below is a diagram from the textbook that has been modified to accommodate the changes needed.**



2. a. Worst-case path:
    i. **Arithmetic R-Format Instruction – 450ps.**
    This calculation includes the I-Mem (225ps), RegRead (110ps), Mux (15ps), and ALU (100ps).
    ii. **lw Instruction – 790ps**
    This calculation includes the I-Mem (225ps), RegRead (110ps), ALU (100ps), D-Mem (340ps), Mux (15ps).
    iii. **Conditional Branch Instruction – 460ps**
    This calculation includes the I-Mem (225ps), RegRead (110ps), ALU (100ps), Sign-Extend (15ps), and the AND gate (10ps).

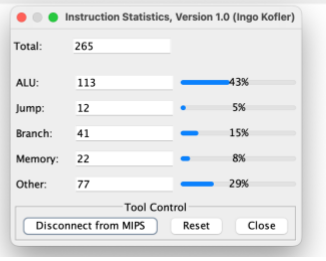
    b. Implementing quicker memory to have quicker DMEM access times is ranked 1st, then creating word addressable IMEM and designing a lower latency control are ranked 2nd. This is because having quicker DMEM access times is the only design approach that genuinely improves the cycle time because it will reduce the DMEM, which, in our case, takes 340ps and is on the worst-case path. Creating word addressable IMEM would not impact it as it is not in any worst-case path. The same is valid for designing a lower latency control unit; since it does not fall on the worst-case path, it would have little to no impact on the cycle time.

3. Program 1 (Instruction Statistics from MIPS):

```
.data
.align 2
str1:  .asciiz "This is the string"
str2:  .asciiz "the"
.text
.globl main

main:
# init
     la $a0, str1          # Load *str address
     la $a1, str2
     addiu $a2, $zero, 20
     addiu $a3, $zero, 3

addiu $t0, $0, 0
j outer_cond
```

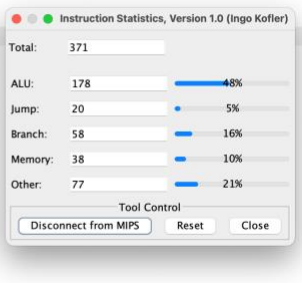


Program 2 (Instruction Statistics from MIPS):

```
.data
.align 2
str:  .asciiz "123456789 123456789 "
delimiter: .byte 0x2C   # example delimiter for ','
.text
.globl main

main:
# init
    addiu $a1, $zero, 20
    la $a2, str
    la $a3, delimiter

outer_cond:
addiu $t0, $t0, 1
slt $t4, $t0, $a1
beq $t4, 1, outer_loop
j exit
outer_loop:
```



a. **Application 1**

Arithmetic, Logical, Shifts and Stores: 113
Jumps: 12
Conditional Branch: 41
Loads: 22
Total = 113 + 12 + 41 + 22 = 188 instructions

Average CPI for Processor A:
(113 * 3) + (12 * 3) + (41 * 3) + (22 * 3) = 564
564 / 188 = 3.00
Average CPI for Processor B:
(113 * 3) + (12 * 2) + (41 * 2) + (22 * 5) = 555
555 / 188 = 2.95

Application 2
Arithmetic, Logical, Shifts and Stores: 178
Jumps: 20
Conditional Branch: 58
Loads: 38
Total = 178 + 20 + 58 + 38 = 294 instructions

Average CPI for Processor A:
(178 * 3) + (20 * 3) + (58 * 3) + (38 * 3) = 882
882 / 294 = 3.00

Average CPI for Processor B:
(178 * 3) + (20 * 2) + (58 * 2) + (38 * 5) = 880
880 / 294 = 2.99

b. To find out which process has better performance, we first need to find the total execution time of each processor.
   Execution Time = Total Cycles / Frequency
   Total Cycles = Total Instructions * Average CPI

   Application 1
   Processor A:
   Total Cycles = 188 * 3.0 = 564
   Execution Time = 564 / Frequency of A

   Processor B:
   Total Cycles = 188 * 2.95 = 555
   Execution Time = 555 / Frequency of B

   Application 2
   Processor A:
   Total Cycles = 294 * 3.0 = 882
   Execution Time = 882 / Frequency of A

   Processor B:
   Total Cycles = 294 * 2.99 = 880
   Execution Time = 880 / Frequency of B

   Total Cycles for Processor A = 564 + 882 = 1446
   Total Cycles for Processor B = 555 + 880 = 1435
   Therefore, since Processor B has fewer Total Cycles, it will be faster than Processor A if the frequencies of both processors are the same.

To find the breakeven frequency between the two processors, we first need to find the ratio between the Frequency of A and B at which the Total Execution Time of the two processors is equal.
Execution Time = Total Cycles / Frequency
Frequency of A = 1446 / Frequency of A
Frequency of B = 1435 / Frequency of B

Since we need to find the ratio, we do:
1446 / Frequency of A = 1435 / Frequency of B

After rearranging, we get:
Frequency of B / Frequency of A = 1435 / 1446 = 0.992

The above shows that for both processors to have identical performance, the frequency of Processor B must be 99.2% of the frequency of Processor A.