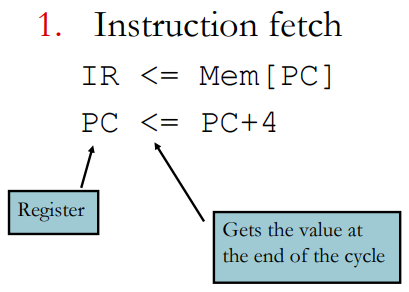
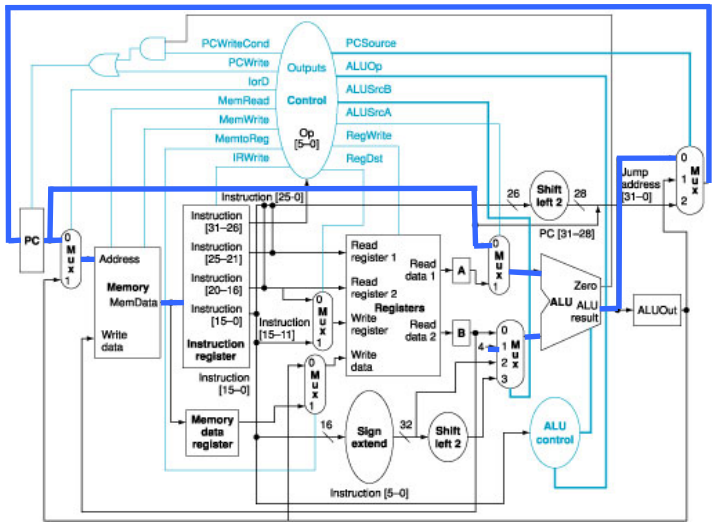
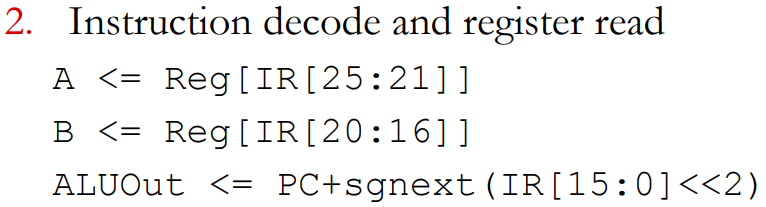
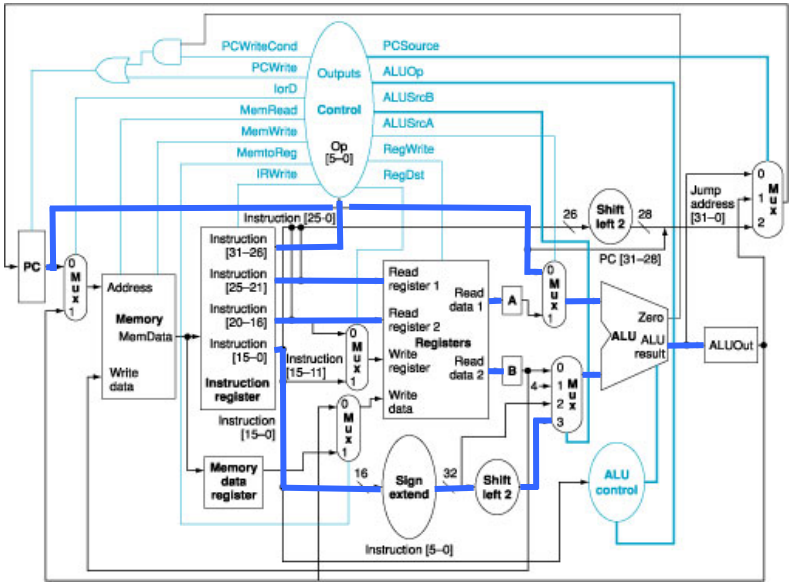
Computer Systems Lecture 14

Cycles 1 And 2

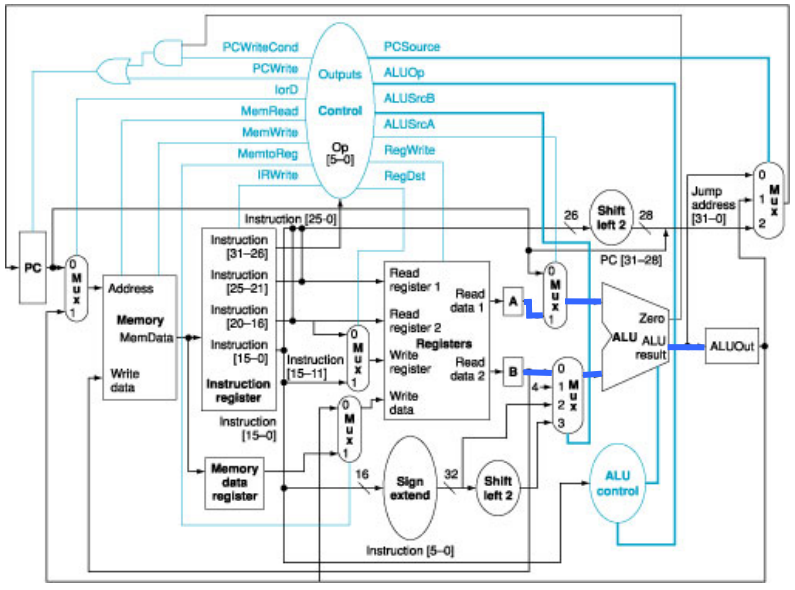
  
In the first cycle we get the instruction

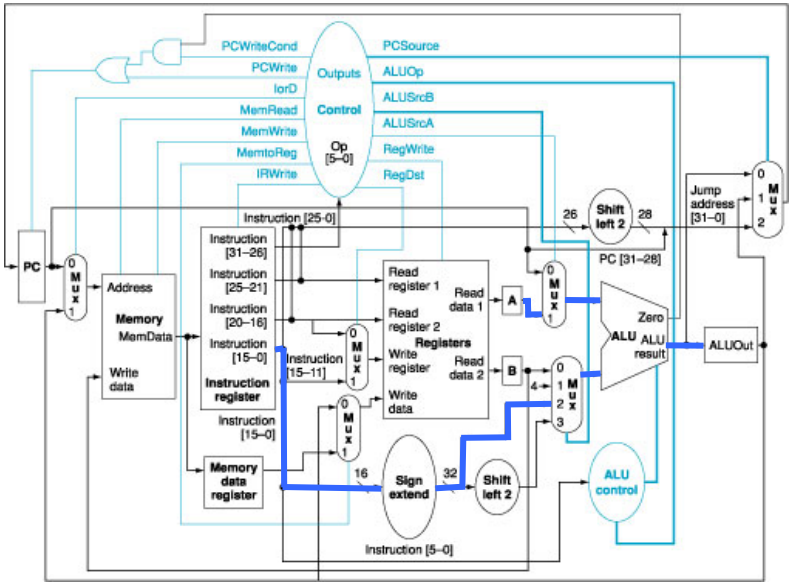
In the second cycle we read two registers from the register block (specified by the instruction) into the A and B registers and then we add the sign extended and left shifted value stored in the last 16 bits of the instruction and store the new value in ALUOut, this is done just incase the instruction is a branch.

Cycle 3

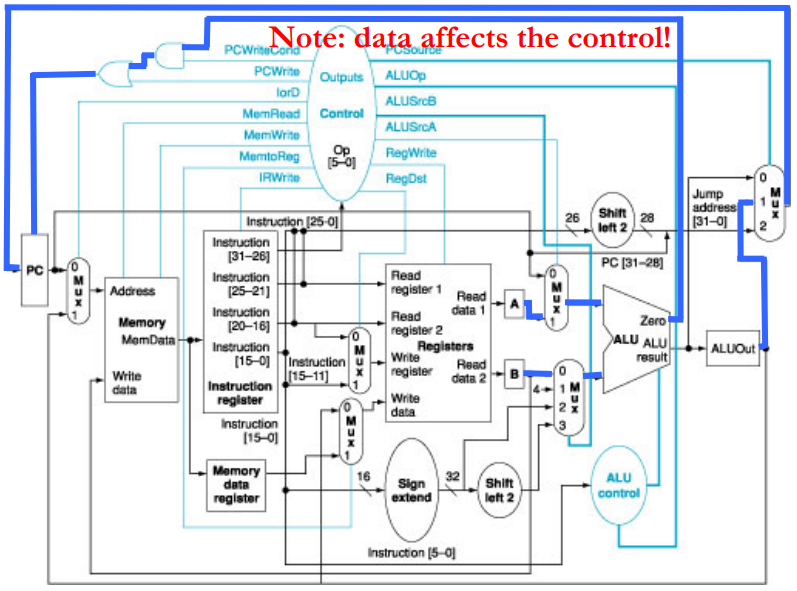
* R-type (arithmetic/logical instruction):
  + ALUOut <= A op B



* Immediate arithmetic, including memory address generation:
  + ALUOUT <= A +sgnext(IR[15:0])

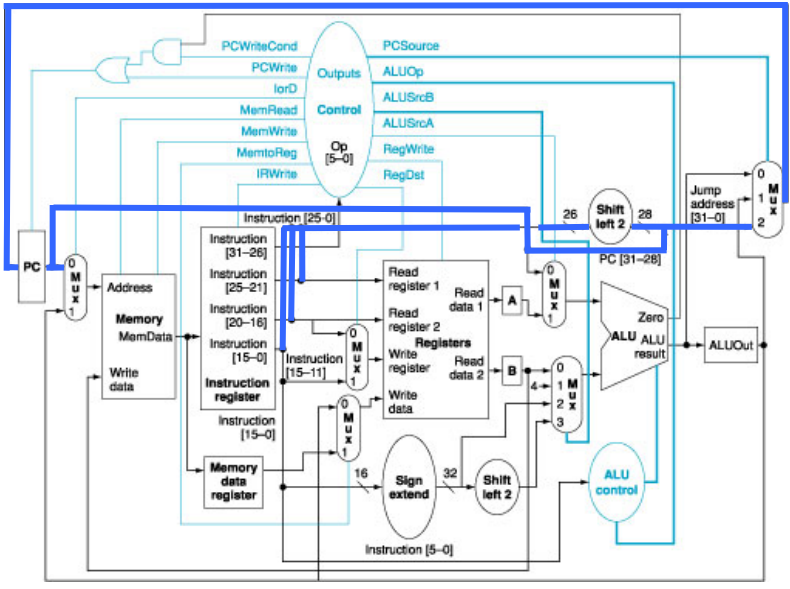


* Branch completion
  + If (A==B) PC <= ALUOut



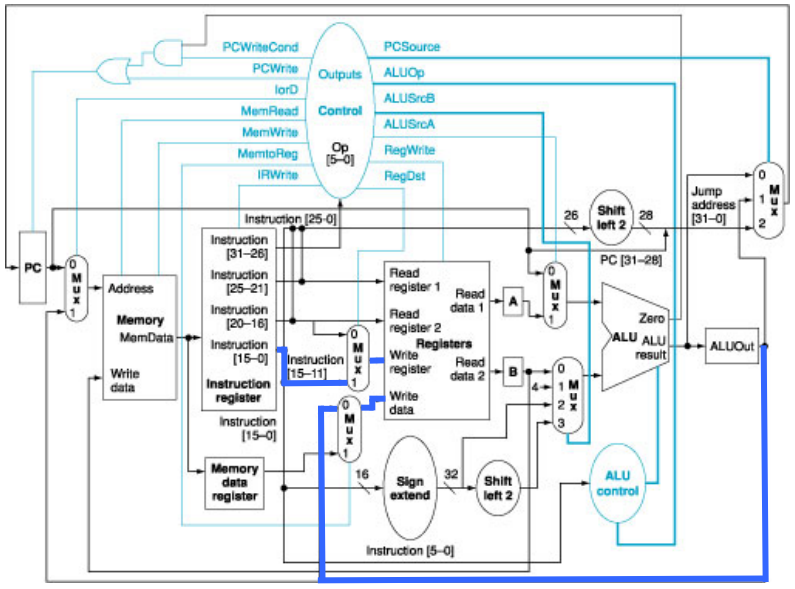
To check if A and B are equal, we subtract them and check if the Zero flag is set.

* Jump completion
  + PC <= {PC[31:28],IR[25:0, 2’b00}

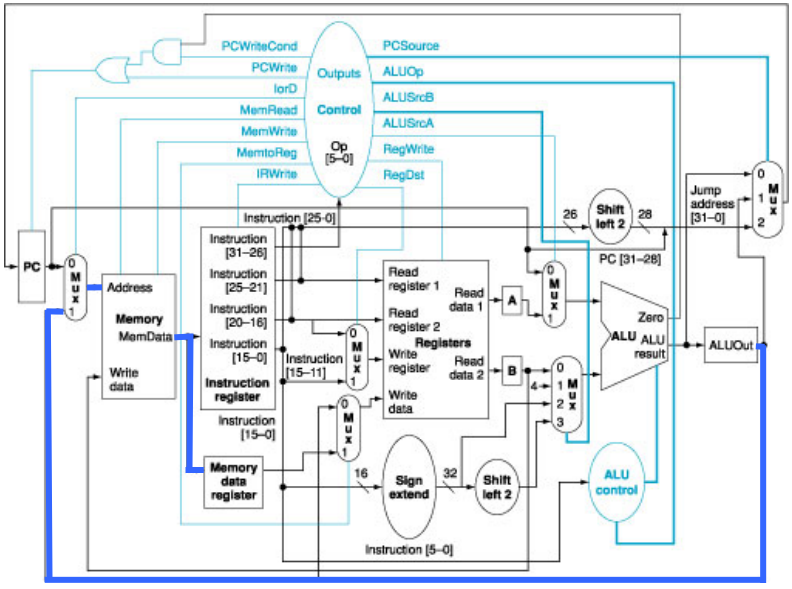


Cycle 4

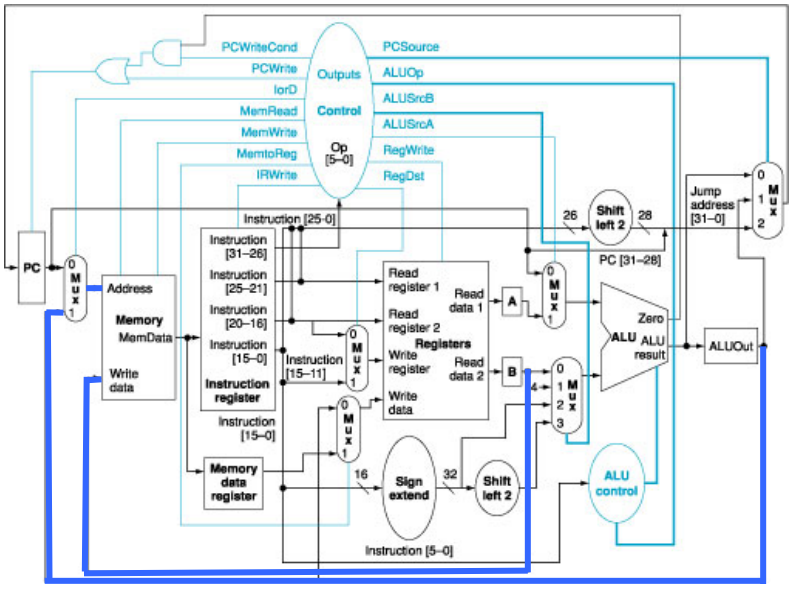
* R-type arith-logical instruction completion
  + Reg[IR[15:11]] <= ALUOut



* Memory access(load)
  + MDR <= Mem[ALUOut]

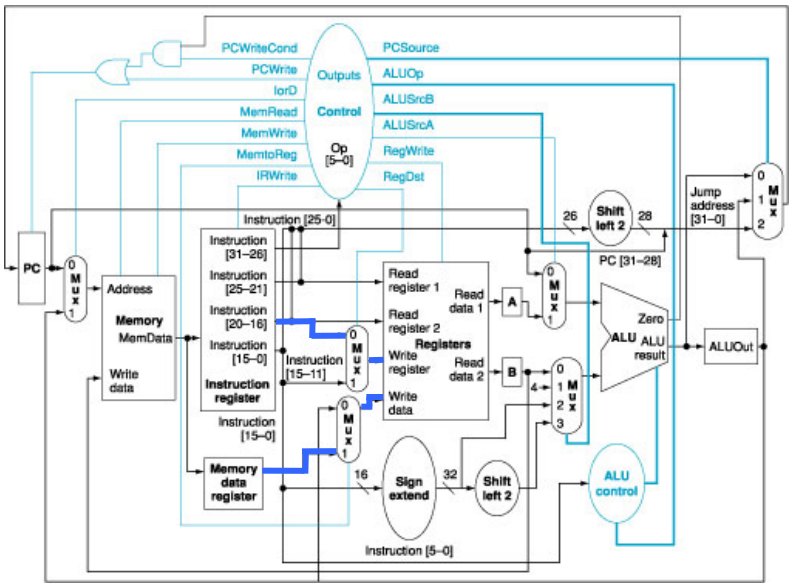


* Memory access(store) & completion
  + Mem[ALUOut] <= B

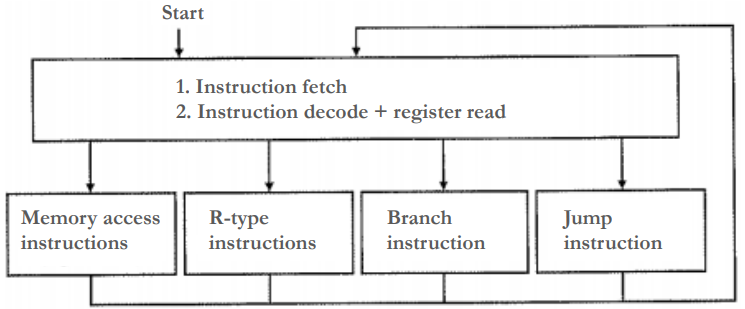


Cycle 5

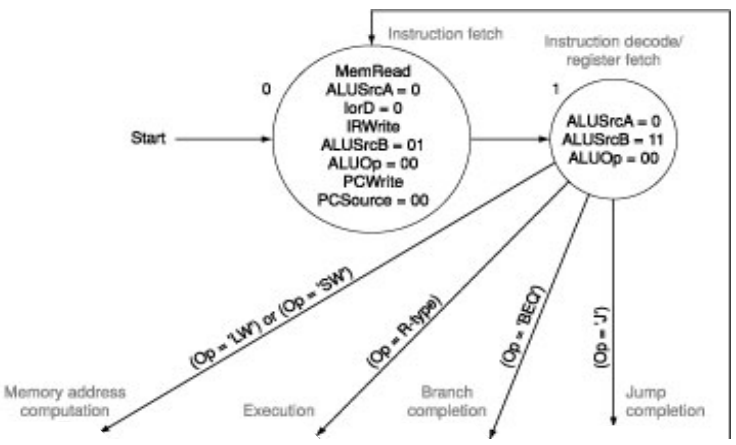
* Load instruction completion
  + Reg[IR[20:16]] <= MDR



Designing The Control Unit



Fetch and Decode States

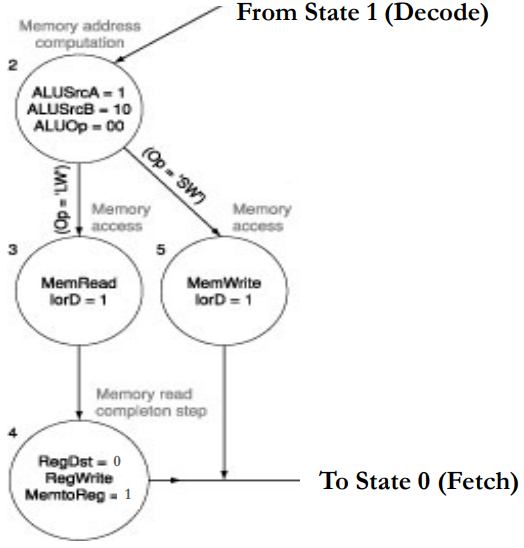


In the first state we set all of the required flags to get the instruction and so, fetch it.

In the second state we set all of the required flags to get the A and B registers set and the op code pulled into the control

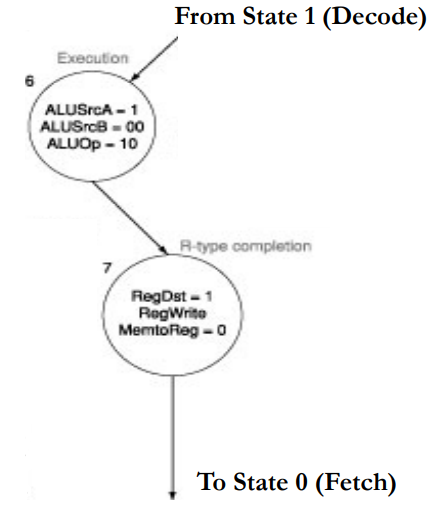
From there, there are four possible paths.

Memory Access States

In the memory access states we start by computing the address and then have two options, save or load.

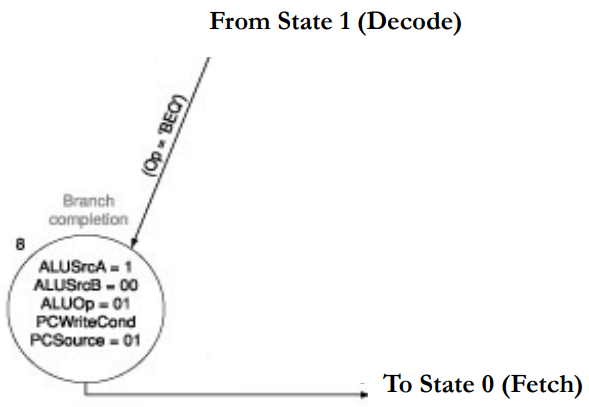
On the save path we write the memory and then go back to the fetch state (finishing in 4 cycles)

On the load path, we read the memory into the memory data register and then next cycle move that value into the destination register and then go back to the fetch state (finishing in 5 cycles).  
R-Type Instruction States

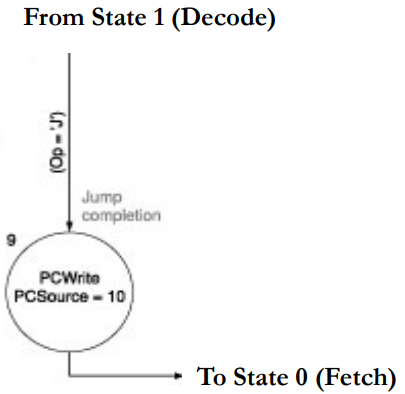
For R instructions we load the A and B registers and execute the operation specified by the ALUOp codes.

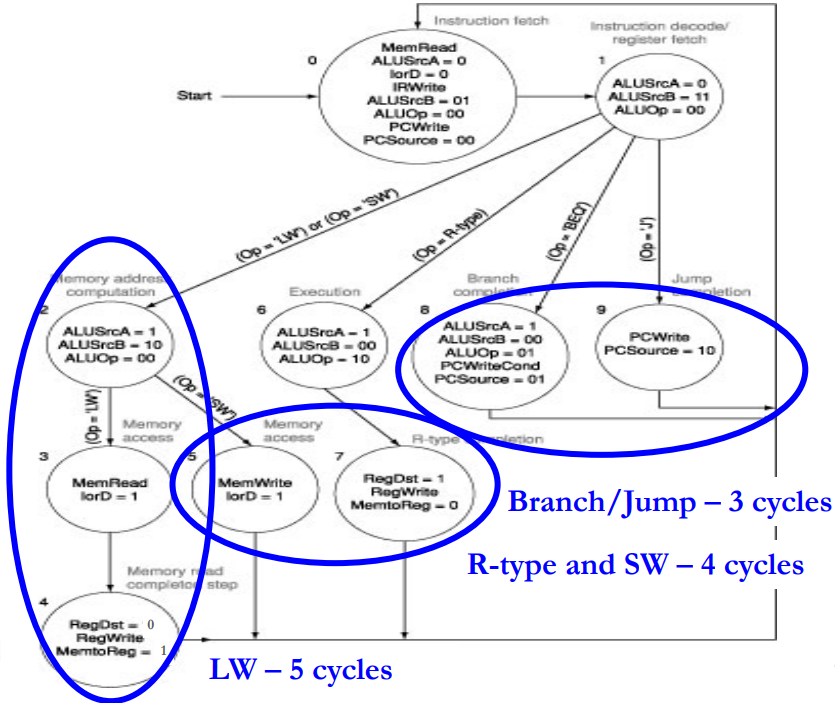
Then we move the value now stored in the ALUOut register into the destination register and go back to the fetch state.

Branch Resolution States

For Branches (in this case branch if equal) the appropriate flags are set to calculate if the branch should be taken and move the pre-calculated address from ALUOut to the PC and then return to the fetch state.

Jump Resolution States

The value of the program counter is set to the appropriate value and then it returns to the fetch state.  
All Together



Implications of Processor Design Choices

Execution time = instruction count \* cycles per instruction \* cycle time.

The instruction count depends on the program no matter what

In a single-cycle processor the number of cycles per instruction is always 1 but the cycle time always has to be as long as the longest possible instruction, which is the load

In a multi-cycle processor the number of cycles changes for each instruction and ranges from 3 to 5 and the cycle time can be much shorter than needed for a single-cycle processor allowing it to run faster, though there are limits to the speed increase.