

## COMPARISON OF SINGLE CYCLE VS MULTI CYCLE CPU ARCHITECTURE...

A single cycle cpu executes each instruction in one cycle. in other words, one cycle is needed to execute any instruction. in other words, our cpi is 1.

Each cycle requires some constant amount of time. This means we will spend the same amount of time to execute every instruction [one cycle], regardless of how complex our instructions may be. to ensure that our processor operates correctly, our slowest instruction must be able to complete execution correctly in one clock tick. This is the big disadvantage of single cycle cpu's: the machine must operate at the speed of the slowest instruction. The big advantage of single cycle cpu's is that they are easy to implement.

As its name implies, the multiple cycle cpu requires multiple cycles to execute a single instruction. This means that our cpi will be greater than 1. The big advantage of the multi-cycle design is that we can use more or less cycles to execute each instruction, depending on the complexity of the instruction. for example, we can take five cycles to execute a load instruction, but we can take just three cycles to execute a branch instruction. The big disadvantage of the multi-cycle design is increased complexity. Control is now a finite state machine - before it was just combinational logic.

Another important difference between the single-cycle design and the multi-cycle design is the cycle time. In the single cycle processor, the cycle time was determined by the slowest instruction. In the multi-cycle design, the cycle time is determined by the slowest functional unit [memory, registers, alu]. This greatly reduces our cycle time.

A single instruction enters the CPU at the Fetch stage and the PC is incremented in one clock cycle. In the next clock cycle, the instruction moves to the Decode stage. In the third clock cycle, the instruction moves to the Access stage and the operands are loaded. In the last two stages, the instruction is executed and the result is stored. In a five stage pipeline a single instruction will take 5 clock cycles to pass through the pipeline. Since the pipeline stages operate independently, a new instruction may enter the Fetch stage as soon as the add instruction has moved to the Decode stage. Under ideal circumstances, a pipelined processor can produce a result on every clock cycle. **Thus, the peak MIPS (Millions of Instructions Per Second) rating of the CPU equals the clock speed in Mhz.** A pipelined CPU achieves maximum throughput only when all stages of the pipeline are filled with instructions which can be processed independently. Performance decreases when gaps or holes appear in the pipeline. A hole is an empty pipeline stage which is not processing an instruction due to hazards in pipeline-ing e.g data hazard , resource hazard or control hazard.

## Single cycle CPU micro-architecture

$$\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} * \frac{\text{Cycles}}{\text{Instruction}} * \frac{\text{Time}}{\text{Cycle}}$$

Let us have to design a CPU whose instruction length is of 16 bits size. Instructions have two register operands/addresses or single memory operand/address. We have reserved 4 bits for opcode. And for register addresses we have 5 bits for each register address. What is needed to execute instruction... e.g. data path and control path (register file and ALU (is just combinational logic) are main parts).

The datapath is the unit that contains all the registers and the functional units. It is where all data computations take place. *The datapath should be constructed in a way such that it is possible to perform all actions necessary to compute the data results of instruction execution.* For example, to implement the add instruction the datapath should provide us with the capability of reading the two registers that we want to add, it should have a functional unit that performs the addition and then allow us to write the result back into the appropriate register.

The control handles the actions that take place in the data path. Eventually the control unit is a finite-state machine or decoder that implements the fundamental execution loop: (1) fetch instruction, (2) decode, (3) read source operands, (4) perform operation, (5) store result, (6) determine which instruction to execute next. It does so by instructing the datapath to perform all appropriate actions.

**How many CPU registers can be accommodated in this architecture???**

As there are 5 bits per register address, 32 registers can be accommodated.....and size of each register should be of same size as of word size of main memory e.g. of 16 bit size...

**How many main memory locations can be addressed??**

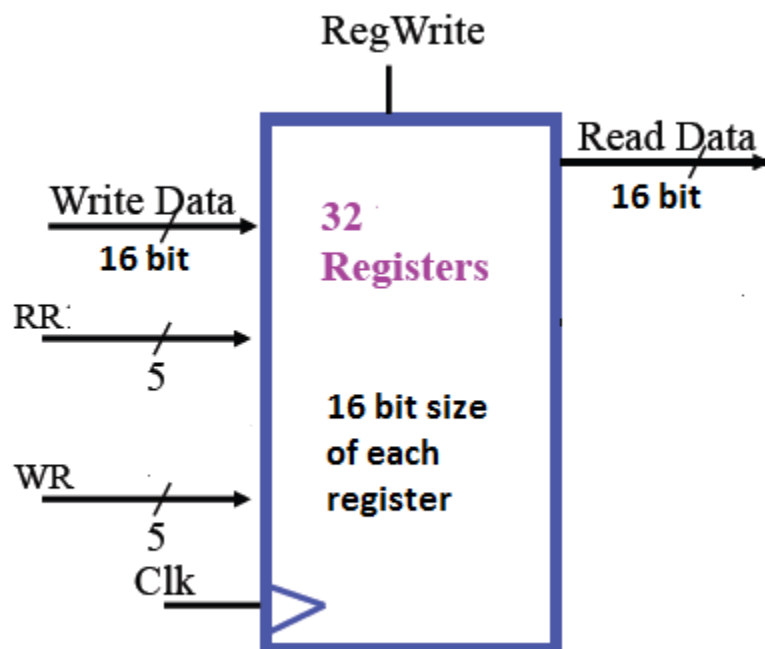
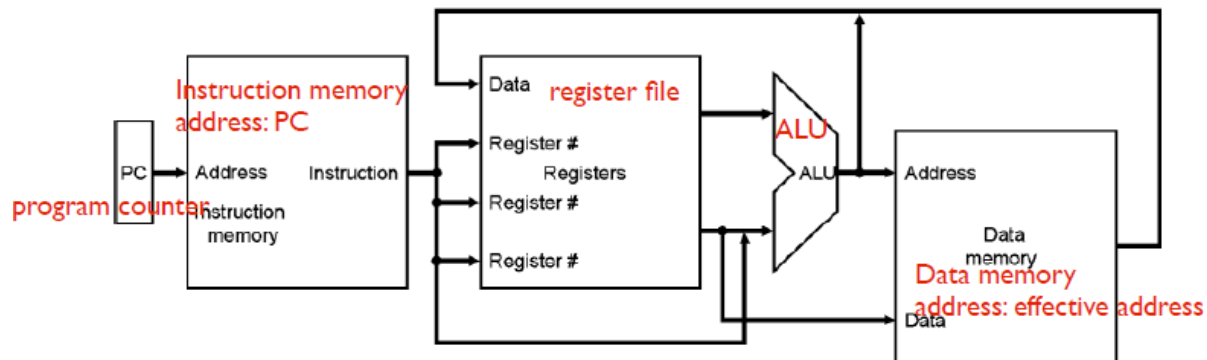
2 power 12 memory locations can be addressed as address part of instruction is of 12 bits when there is only memory operand/address.

Register file and how it is implemented, how CPU hardware works when instruction is executed..e.g. what will be on wires, address wires, control wires....etc.....????????

**Register file (registers and combinational logic or finite state machine)**

Although called a "file", a register file is not related to disk files. A register file is a small set of high-speed storage cells inside the CPU. There are special-purpose registers such as the IR and PC, and also general-purpose registers for storing operands of instructions such as add, sub, mul, etc. **A CPU register can generally be accessed in a single clock cycle, whereas main memory may require dozens of CPU clock cycles to read or write.**

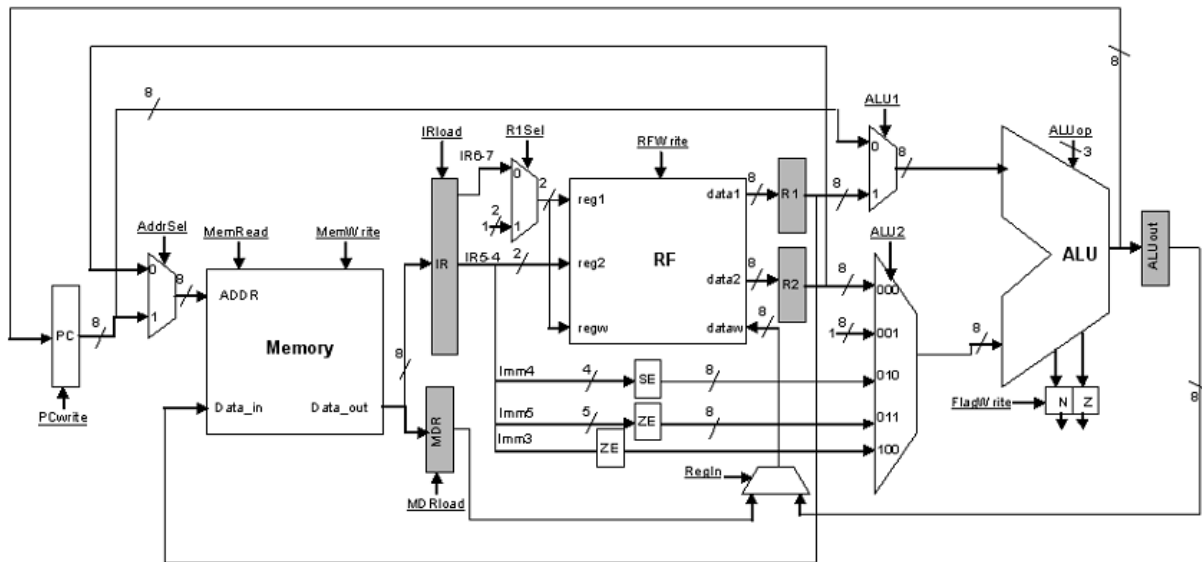
Since there are very few registers compared to memory cells, registers also require far fewer bits to specify which register to use. This in turn allows for smaller instruction codes. For example, the processor we are designing has 32 general-purpose registers, so it takes 5 bits to specify which one to use. In contrast, it has a 2K memory capacity of 16 bit width, so it takes 12 bits to specify which memory array/cell to use (word addressing is used). Below is diagram for single cycle cpu architecture and register file respectively.





## Multicycle CPU Micro-Architecture

As with the single-cycle implementation our processor will consist of two cooperating units the datapath and the control. The key difference here is that the execution of a single instruction will take multiple cycles to complete. Accordingly, the datapath will have to change a bit. The key change will be the introduction of temporary registers to hold the outcomes that are produced at each cycle. This is best understood by looking at the schematic for the datapath. For the time being please ignore the details and focus on the grey boxes. These are the new registers:



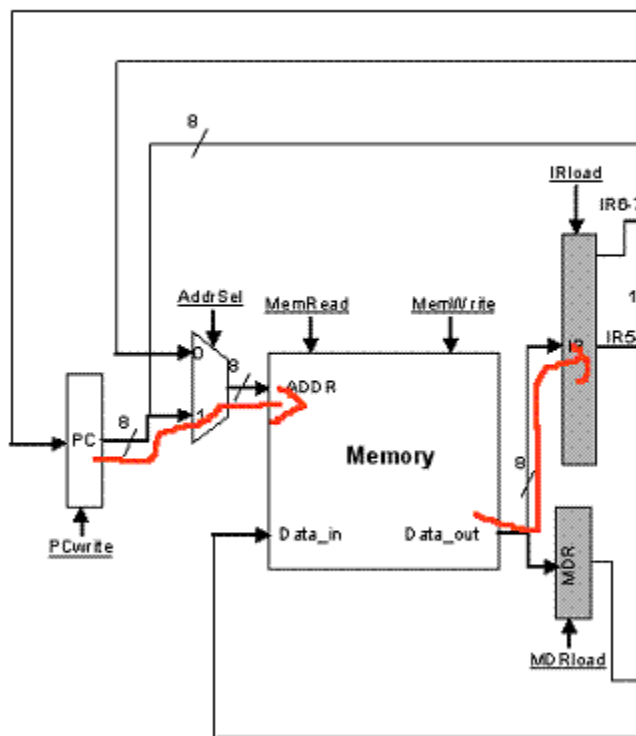
The following temporary registers are introduced:

1. **IR or Instruction Register:** This is used to hold the instruction encoding after it is read from memory. A register is needed because we will use a single memory device both for data and instructions. Accordingly, its output may change during the execution of an instruction (a load will read from memory).
2. **R1 and R2:** These are used to temporarily hold the register values read from the register file.
3. **AluOut:** This is used to temporarily hold the result calculated by the ALU.

4. MDR or Memory Data Register: holds the value returned from memory so that it can later be written into the register file.

Let's see how this datapath was derived. We will explain what happens cycle by cycle. The first two cycles are the same for all instructions since we need to fetch the instruction from memory and then decode it (i.e., the control has to look at the opcode and decide what to do next).

### CYCLE 1: Fetching the Instruction and Incrementing the PC

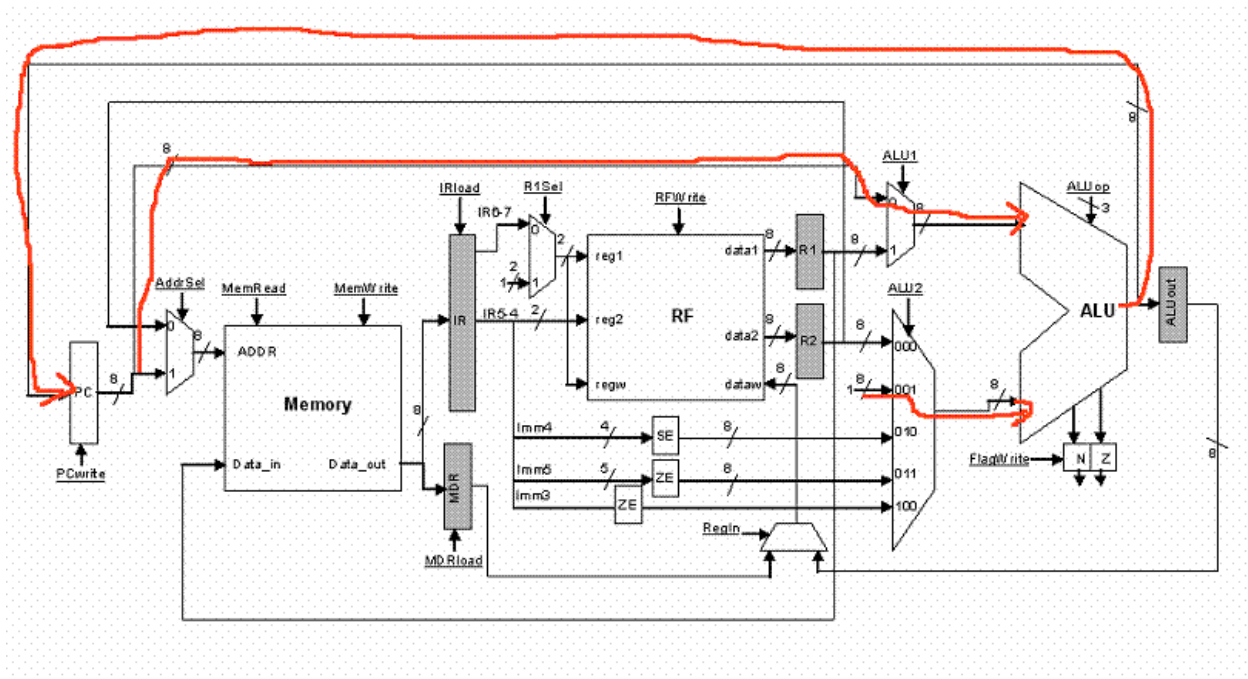


The first step in executing an instruction requires fetching the instruction from memory. For this we have to send the value of the PC register to the address lines of the memory device. Assuming that the memory will respond within this first cycle, we want to store the returned value (this is the encoding of the instruction that we should execute). To do this we need to take the value from the memory's output and write into the IR register.

Because we may access the same memory device to perform a load or a store (read and write respectively) a MUX is needed at the address input so that it is possible to send either the PC or another address. So, during the first cycle we will be reading the instruction encoding from memory. This is probably a good time to also calculate  $PC = PC + 1$  as all instructions

use this (even branches require  $PC + 1$  as part of their target calculation).

In parallel with the memory access, we send the PC value through the ALU1 mux to the ALU. As the second input to the ALU we send the number 1 (input 001 of MUX ALU2). Finally, we set ALUop to 000 (addition). As a result, the ALU will calculate  $PC + 1$ . By setting PCWrite to 1, at the end of the current clock cycle, PC will change and will become  $PC + 1$ .

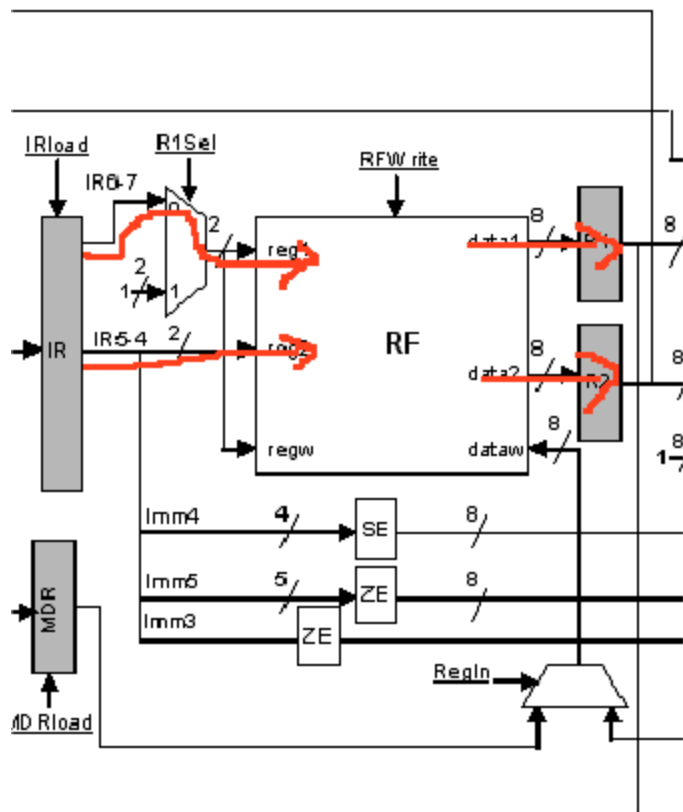


**CYCLE 1 SUMMARY:** In summary the following actions take place during the first cycle. This is often called the FETCH cycle.

$$[IR] = \text{Mem}[PC]$$

$$[PC] = [PC] + 1$$

**CYCLE 2:** Decoding the instruction and reading from the register file. During the second cycle, the control will be taking a look at the instruction opcode in order to decide what should happen during the next cycle. Because many instructions use the registers specified in fields R1 and R2 of the instruction we also read these registers from the register file. Note that some instructions do not use R1 or R2. In this case, we would have read registers that we do not need. While this is extra work we literally had nothing better to do during the second cycle. So, it is OK in hardware to perform actions that may be useful and later ignore the results if they are not needed. This is permissible as long as the extraneous work does not change and machine state in an irreversible way (reads do not change the register values so they are OK).



Thus at the end of the 2nd cycle, registers R1 and R2 are loaded with the values held by the registers identified by the instruction bit fields R1 and R2 respectively.

### CYCLE 2 SUMMARY:

$[R1] = RF[[IR7..6]]$

$[R2] = RF[[IR5..4]]$

### Instruction Decode

### CYCLE 3 and 4

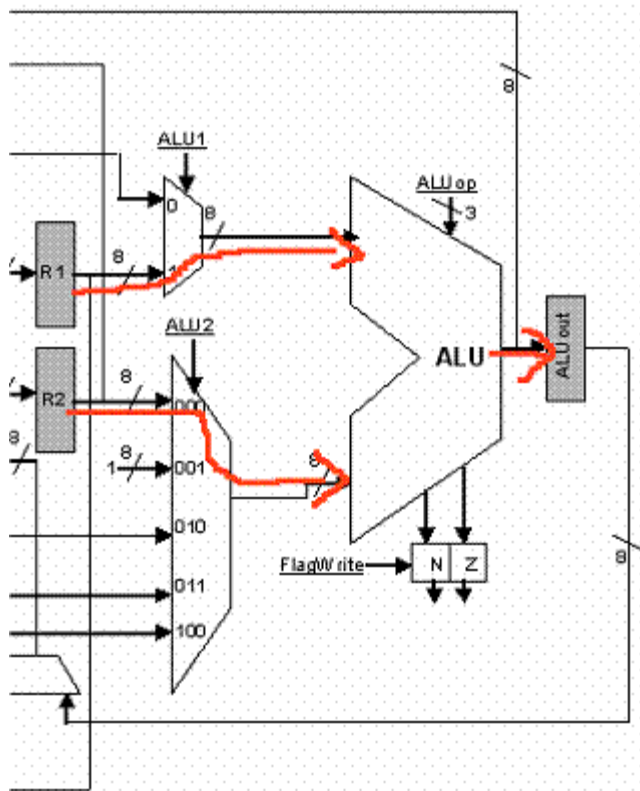
What happens after cycle 2 depends on the actual instruction. Accordingly we will consider each instruction in turn.

\*\*\* **ADD, SUB**

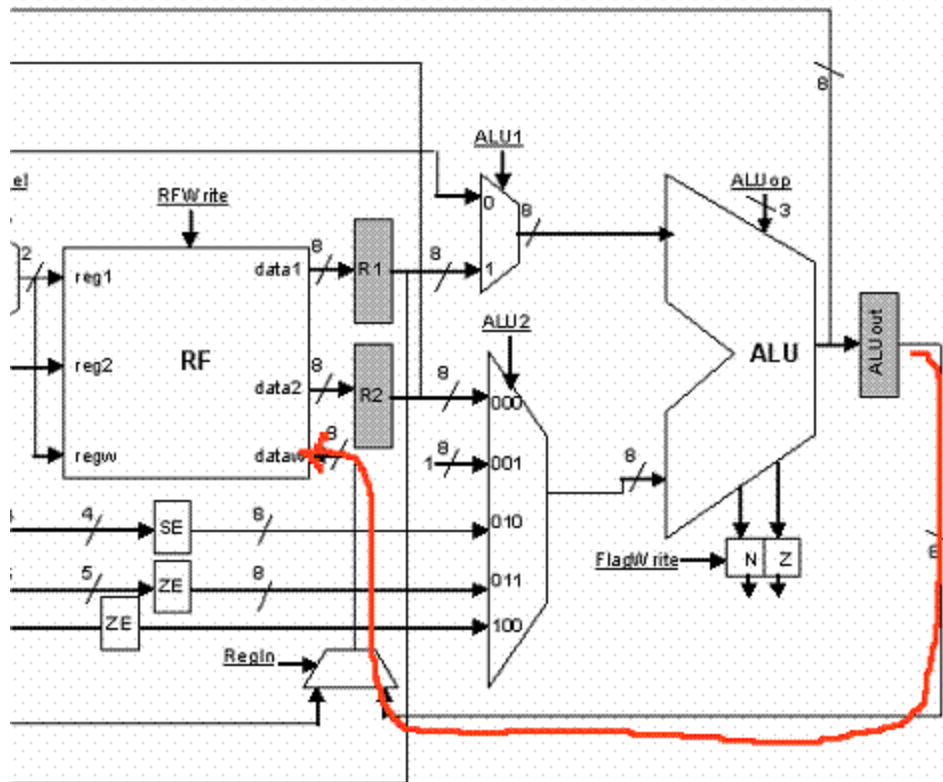


The execution of these three instruction proceeds into additional steps:

In cycle 3 we calculate the operation specified by the instruction and at the end store the result into ALUout. In cycle 4 we write the result into the register file:



Cycle 3



- Cycle 4

### Why a Multiple Cycle CPU?

- The problem => single-cycle cpu has a cycle time long enough to complete the longest instruction in the machine
- The solution => break up execution into smaller tasks, each task taking a cycle, different instructions requiring different numbers of cycles or tasks
- Other advantages => reuse of functional units (e.g., alu, memory)