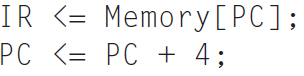
***5 Stages of imlementation***

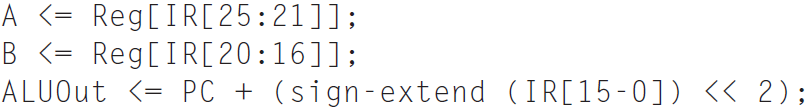
5 stage pipeline implementation:

* *IF - Instruction fetch:*



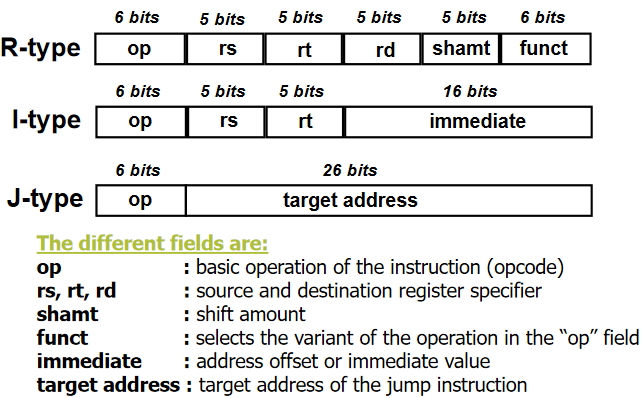
Send the PC to the memory as the address, perform a read, and write the instruction into the Instruction register (IR), where it will be stored. Also, increment the *PC by 4* (each instruction is 4 bytes). We use the symbol “<=” from Verilog; it indicates that all right-hand sides are evaluated and then all assignments are made, which is effectively how the hardware executes during the clock cycle.

* *ID – Instruction decode and register fetch:*



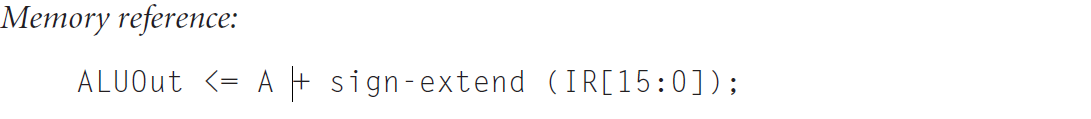
Access the register file to read registers rs and rt and store the results into the registers A and B. Since A and B are overwritten on every cycle, the register file can be read on every cycle with the values stored into A and B. This step also computes the branch target address and stores the address in ALUOut, where it will be used on the next clock cycle if the instruction is a branch. This requires setting ALUSrcA to 0 (so that the PC is sent to the ALU), ALUSrcB to the value 11 (so that the sign-extended and shifted offset field is sent to the ALU), and ALUOp to 00 (so the ALU adds). The register file accesses and computation of branch target occur in parallel.

After this clock cycle, determining the action to take can depend on the instruction contents.

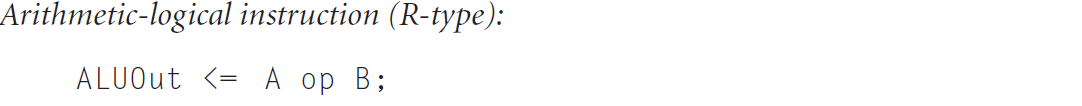


* *EX – Execution/effective address:*

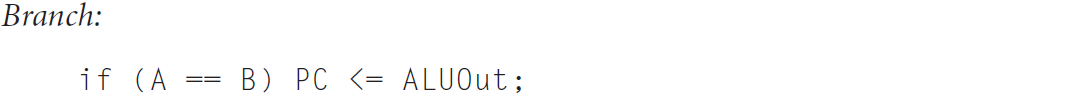
The ALU operates on the operands prepared in the prior cycle, performing one of three functions depending on the instruction type.



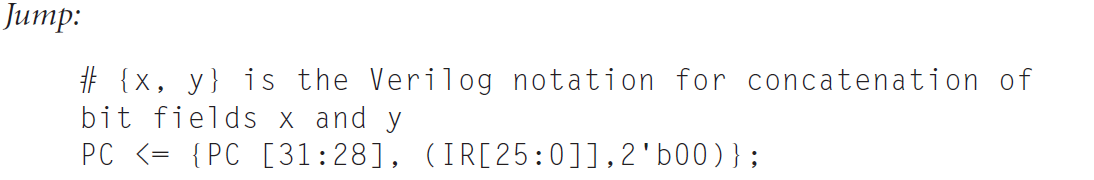
* The ALU is adding the operands to form the memory address. This requires setting ALUSrcA to 1 (so that the first ALU input is register A) and setting ALUSrcB to 10 (so that the output of the sign extension unit is used for the second ALU input). The ALUOp signals will need to be set to 00 (causing the ALU to add).



* The ALU is performing the operation specified by the function code on the two values read from the register file in the previous cycle. This requires setting ALUSrcA = 1 and setting ALUSrcB = 00, which together cause the registers A and B to be used as the ALU inputs. The ALUOp signals will need to be set to 10 (so that the funct field is used to determine the ALU control signal settings).

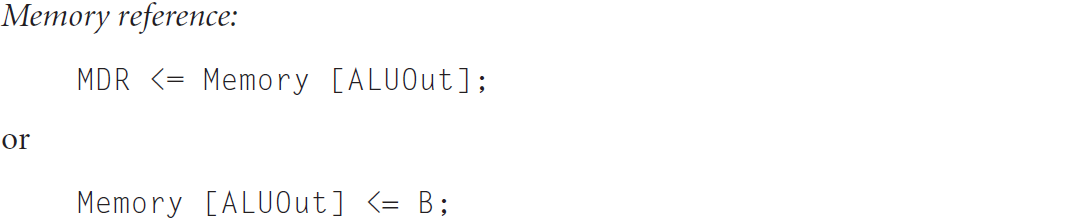


* The ALU is used to do the equal comparison between the two registers read in the previous step. The Zero signal out of the ALU is used to determine whether or not to branch. This requires setting ALUSrcA = 1 and setting ALUSrcB = 00 (so that the register file outputs are the ALU inputs). The ALUOp signals will need to be set to 01 (causing the ALU to subtract) for equality testing. The PCWriteCond signal will need to be asserted to update the PC if the Zero output of the ALU is asserted. By setting PCSource to 01, the value written into the PC will come from ALUOut, which holds the branch target address computed in the previous cycle. For conditional branches that are taken, we actually write the PC twice: once from the output of the ALU (during the Instruction decode/register fetch) and once from ALUOut (during the Branch completion step). The value written into the PC last is the one used for the next instruction fetch.



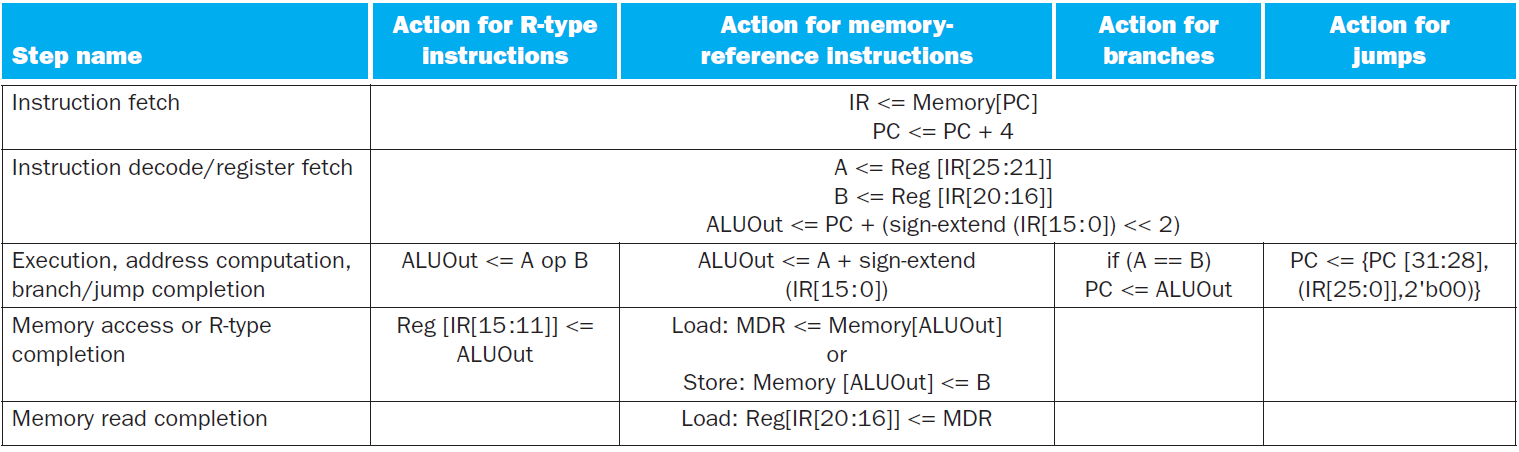
* The PC is replaced by the jump address. PCSource is set to direct the jump address to the PC, and PCWrite is asserted to write the jump address into the PC.
* *MEM – Memory access:*

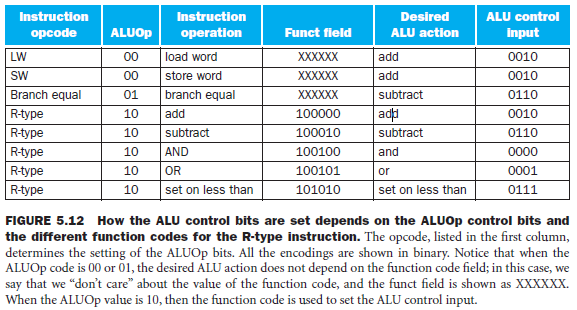
If the instruction is a load, memory does a read using the effective address computed in the previous cycle. If it is a store, then the memory writes the data from the second register read from the register file using the effective address.



* *WB – Write-back:*

Write the result into the register file, whether it comes from the memory system or form the ALU.

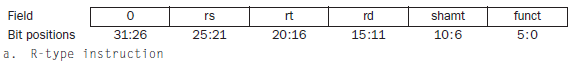


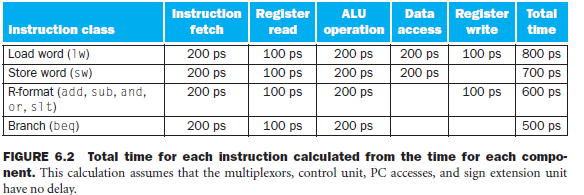


* An R-type instruction, such as **add $t1,$t2,$t3**. Although everything occurs in 1 clock cycle, we can think of four steps to execute the instruction; these steps are ordered by the flow of information:

*Step 1: Fetch*

*Instruction is fetch form instruction memory, PC increment by 4*

**

**