D102
CPU Internals:
Structure and Operation

Peter Rounce - room G06

p.rounce@cs.ucl.ac.uk

Structure and Operation of a simplified MIPS CPU

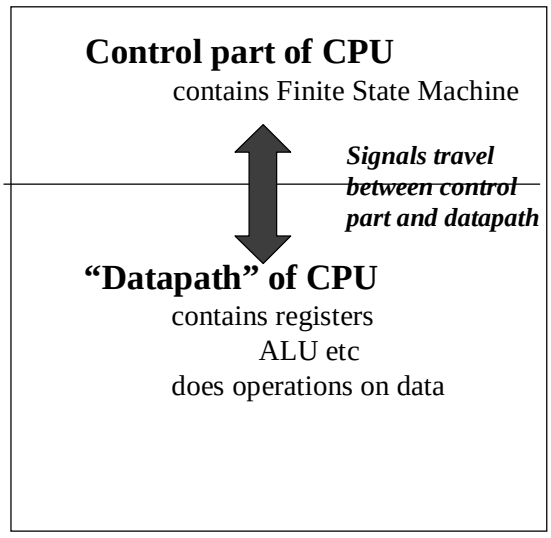
•Components will be just things we have examined during the course
Registers, adders etc, mulitplexors

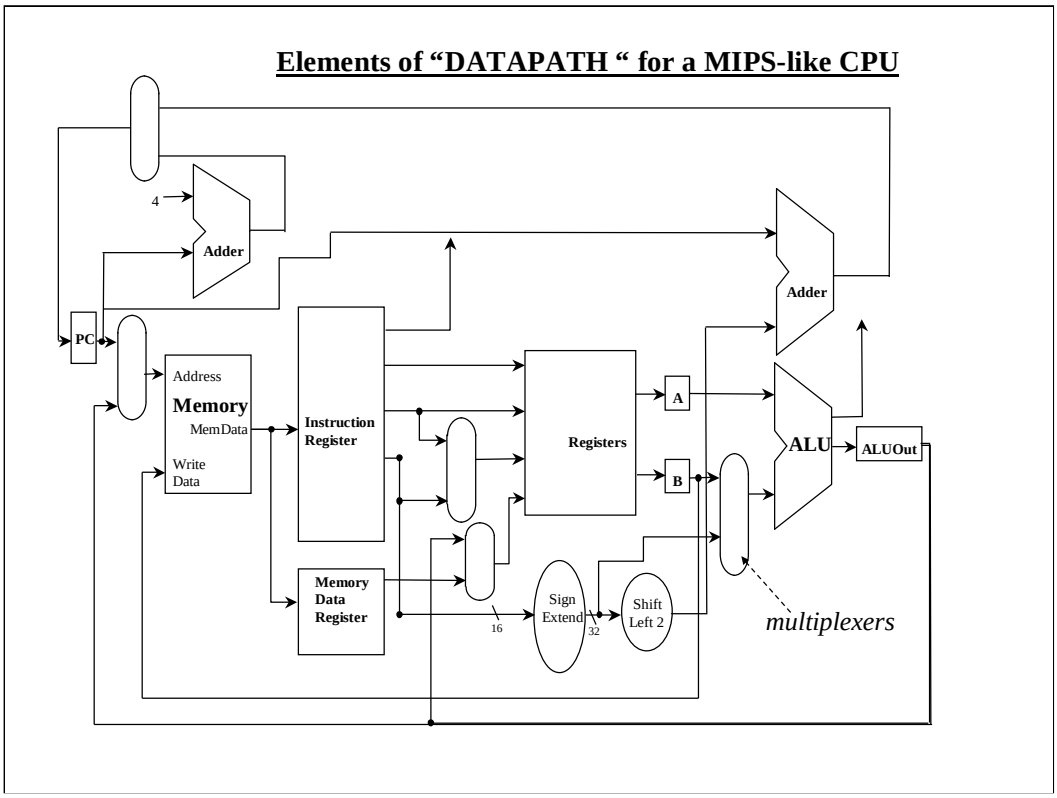
•Examine the operation sequence of arithmetic, branch and memory instructions

• Look first and instruction fetch and then instruction – same for all instructions

• then look first at ALU operation and difference between instructions

• we will be looking at how signals from the control logic determine the actions and finally we look at how these control signals are produced in the right sequence


Control part of CPU
contains Finite State Machine
Signals travel between control part and datapath
"Datapath" of CPU
contains registers
ALU etc
does operations on data



The datapath contains:-

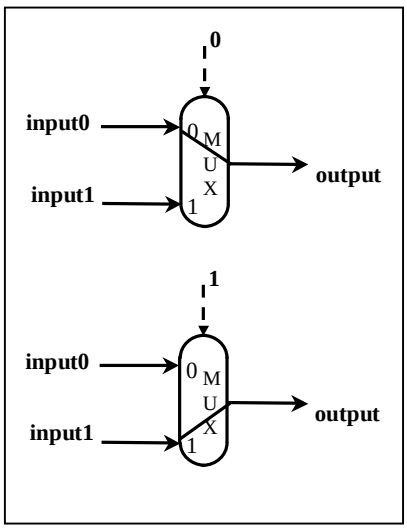
Registers – these are the storage elements and consist of the programmer visible general registers ($0-$31, pc) and registers used for temporary storage by the CPU and invisible to the programmer (instruction register, memory data register, ALUOut, A, B registers)

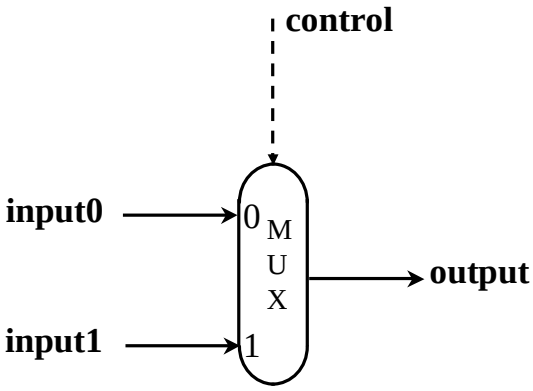
Multiplexers – select from a set of inputs to be output from the multiplexer, allowing just one of the inputs to drive the output. These allow for multiple different pathways to exist within the CPU and for control logic to choose the appropriate pathways to be used for an instruction.

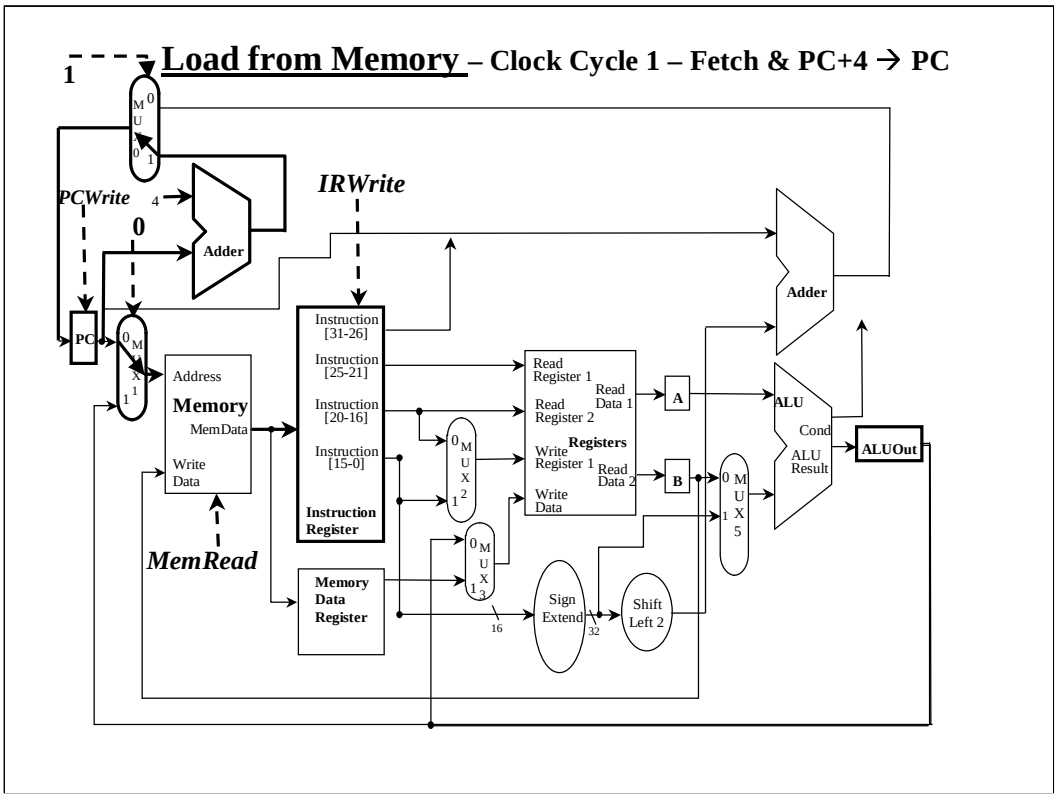
Functional units – unit to process values in some ways, (ALU, adder, shift units)

Multiplexor Operation

Value on control determines input to output connection







An example instruction is fetched and executed in the next 5 slides.

The instruction is a Load Word instruction: the longest instruction in this version of the architecture.

This instruction has five stages:

fetch instruction,

decode instruction & read registers,

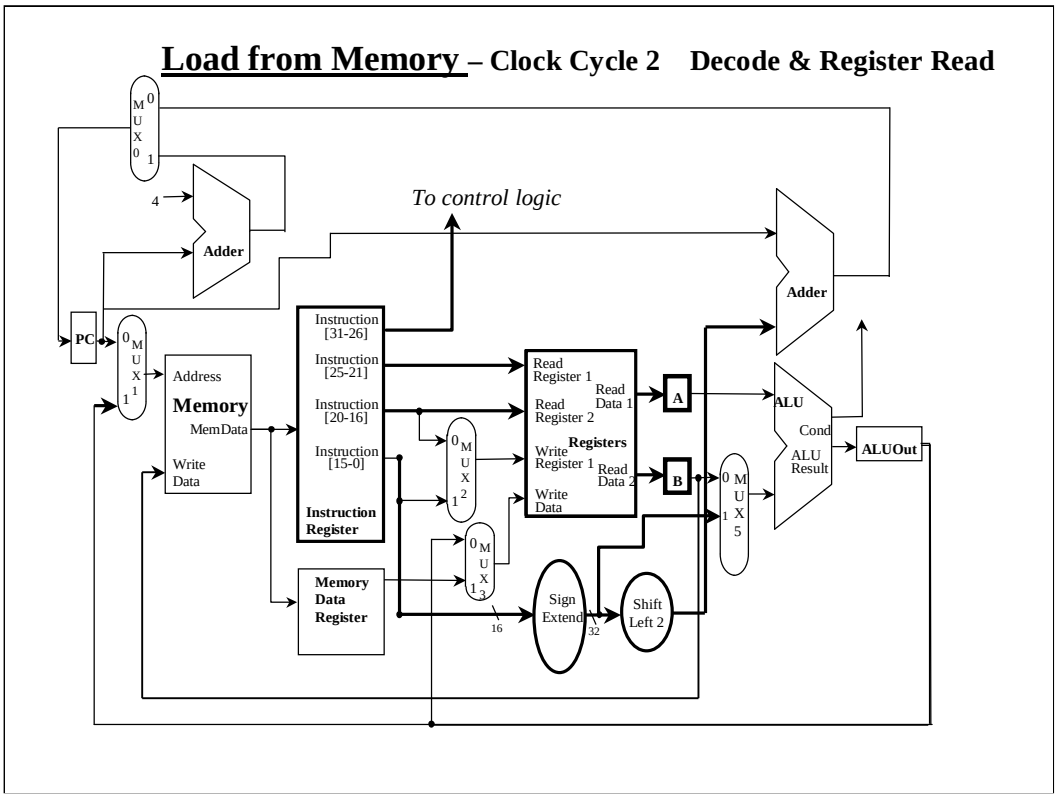
use adder to produce address,

use address to read value from memory,

move value from memory into the target register.

Each stage takes one clock cycle.

The first stage is shown here. The value from the PC is routed to the memory via a multiplexer, and a read is done at the memory. The value from the memory is stored in the instruction register (IRWrite is active in this cycle) at the end of the clock cycle. While the read is going on the value from the PC is also routed through an adder where 4 is added to the value: the output from the adder is stored in the PC at the end of the clock cycle (PCWrite is active in this cycle) .



Clock cycle 2:

The instruction is stored in the instruction register at the and of clock cycle 1.

Components of the instruction are routed to different places:-

As this CPU executes MIPs instructions, the top 6 bits of the instruction (bits 26-31), the opcode bits, are routed to the control logic of the CPU, along with any other necessary bits. These bits determine the operation and from these bits the control logic generates signals for the rest of the instruction to signal multiplexer settings, ALU operation and register write settings.

The source register selectors, that indicate, which registers are or may be used in the instruction are passed to the register block, and 2 registers are read and the data output are stored in registers A and B.

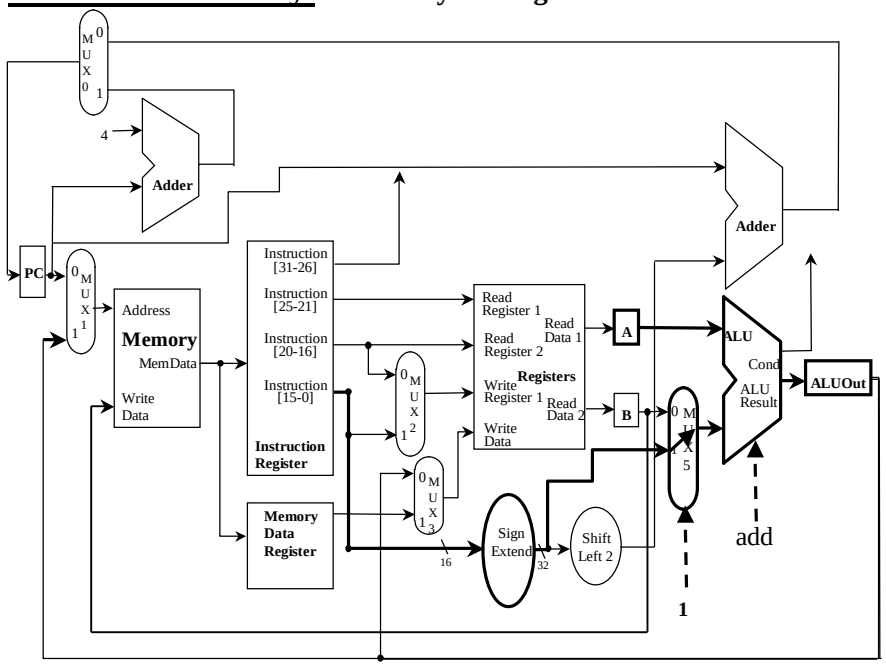
The least significant 16 bits (bits 15-0) are routed in a number of ways to the multiplexer at the front of the ALU:-

Route 1: bits 15-0 are signed extended to 32 bits and passed to the multiplexer input 1 for use by load, store, addiu instructions.

Route 2: bits 15-0 are signed extended to 32 bits, shifted left by 2 and passed to the branch adder to calculate the branch target address in branch instructions.

Route 1: bits 15-0 are zero extended to 32 bits and passed to the multiplexer input 3 for use by andi, ori, lui instructions etc: *route not shown in diagram*.

Load from Memory – Clock Cycle 3 generate address in ALUOut

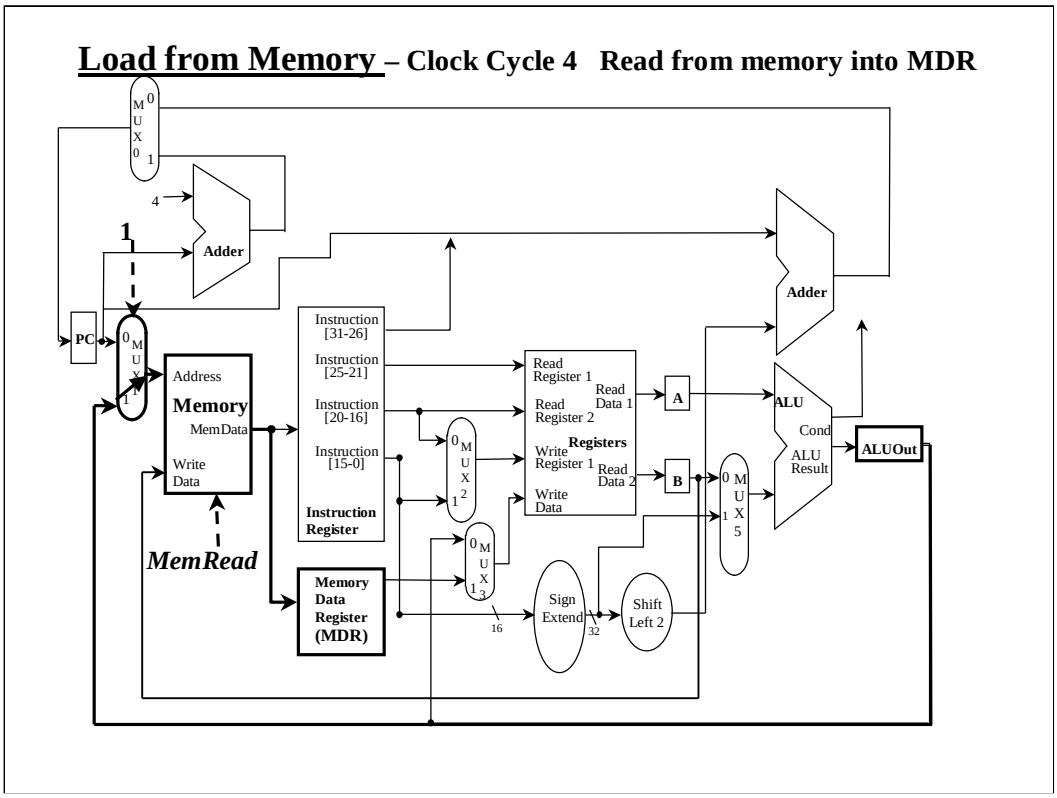


This is the first instruction specific cycle- cycles 1 and 2 are the same for all instructions.

The cycle needs to take the signed extended offset from bits 15-0 of the instruction (available at mulitplexer input 1) and add it to the source register value (available in register A).

Thus the control logic will have sent appropriate values to the muliplexer on front of the ALU to route the required input to the ALU, and also sent appropriate values to the ALU to select the ALU to add its inputs.

The output of the ALU is stored in the ALUOut register at the end of the clock cycle for use in the next clock cycle.

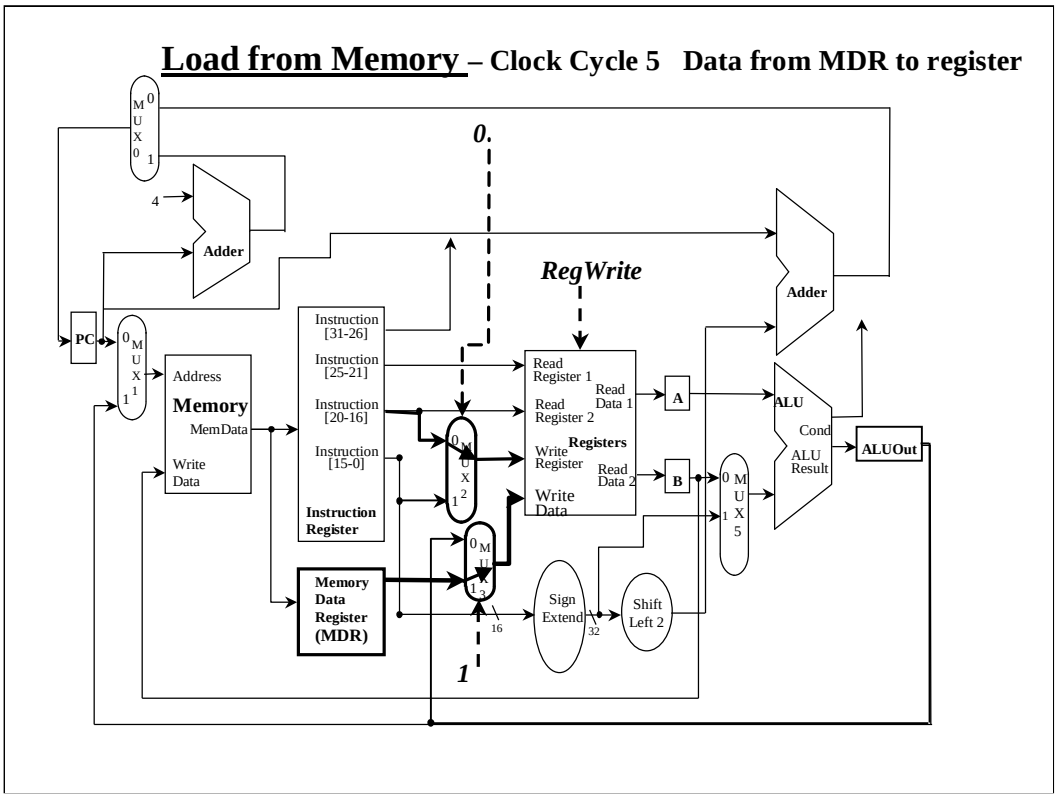


In cycle 4, the Load instruction reads from memory using the address calculated in cycle 3. The address is stored in *ALUout*.

The address is routed via input 1 MUX1 under the control of signal *IorD* to the Address bus, and then to the memory.

The control signal *MemRead* is activated to cause the memory to output the selected memory location contents.

The output data from the memory passses down the data bus and is stored in the ***Memory Data Register*** at the end of Clock 4..



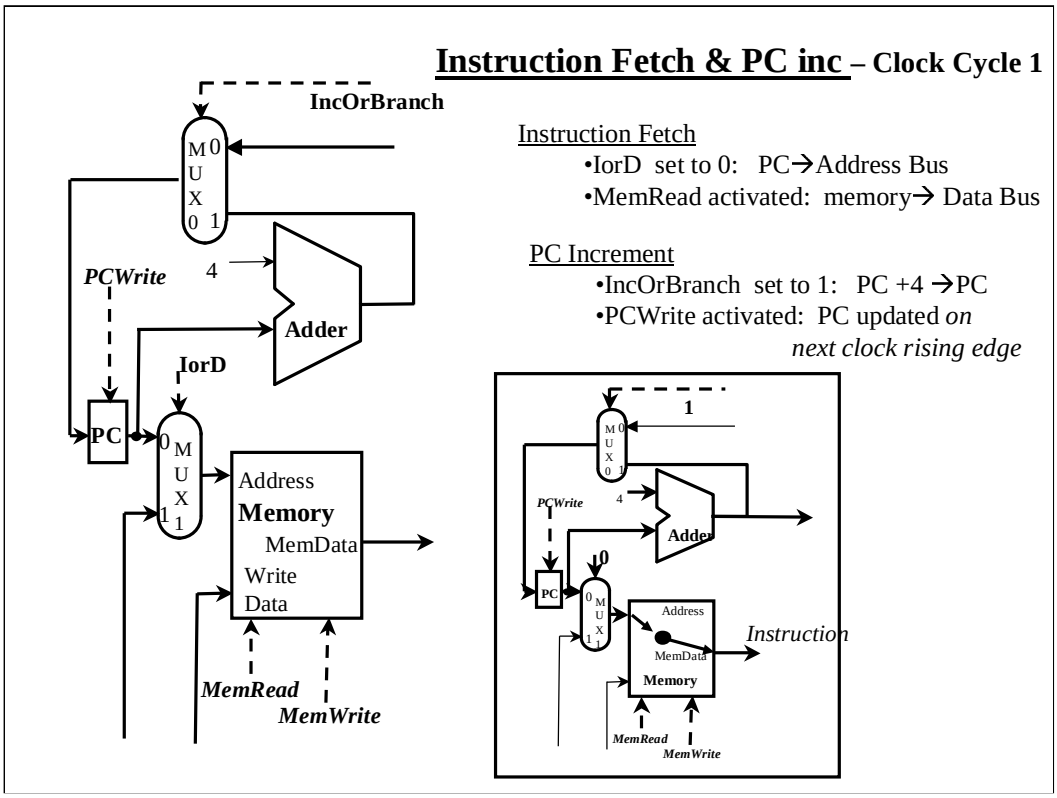
On cycle 5 the Load instruction completes, when the value read from memory in cycle 4 is moved from the Memory Data Register into the destination register.

The destination register specifier on bits 20-16 is routed to the Write Register input of the Register block via input 0 of MUX2.

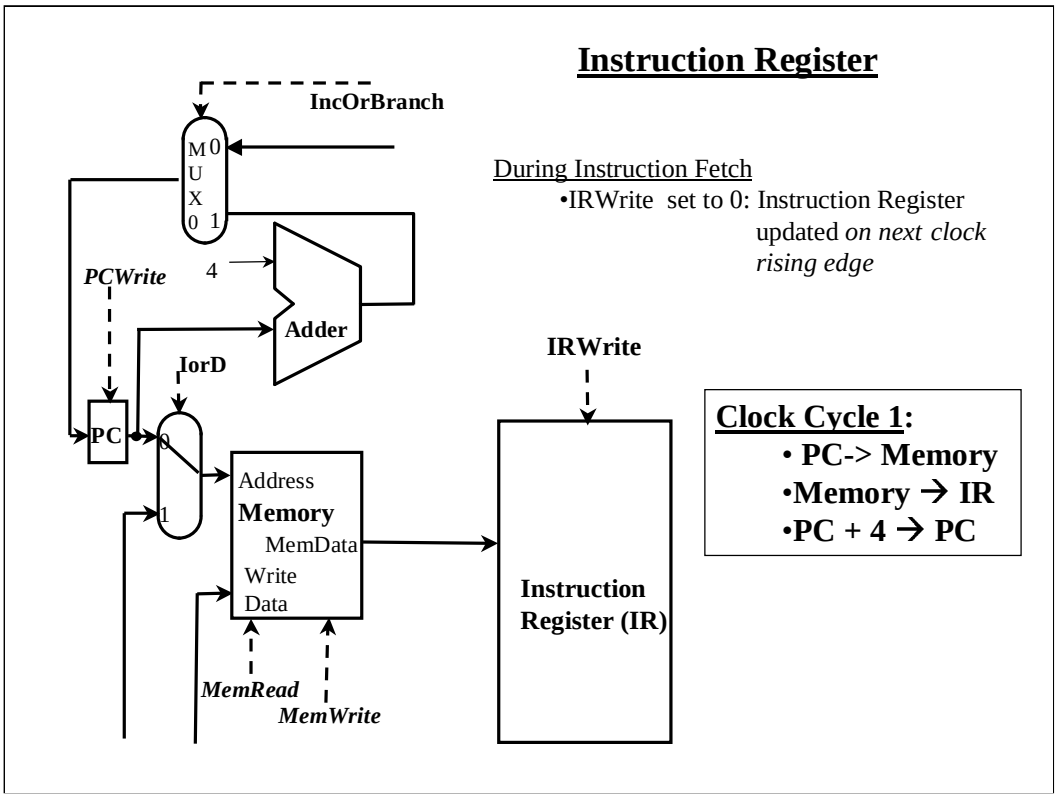
The value in the Memory Data Register (read from memory in cycle 4) is passed via input 1 of MUX3 to the register block data inputs.

RegWrite is set active to allow the selected register to be written.

The value from memory (in the Memory Data Register) is written (loaded into) the destination register.

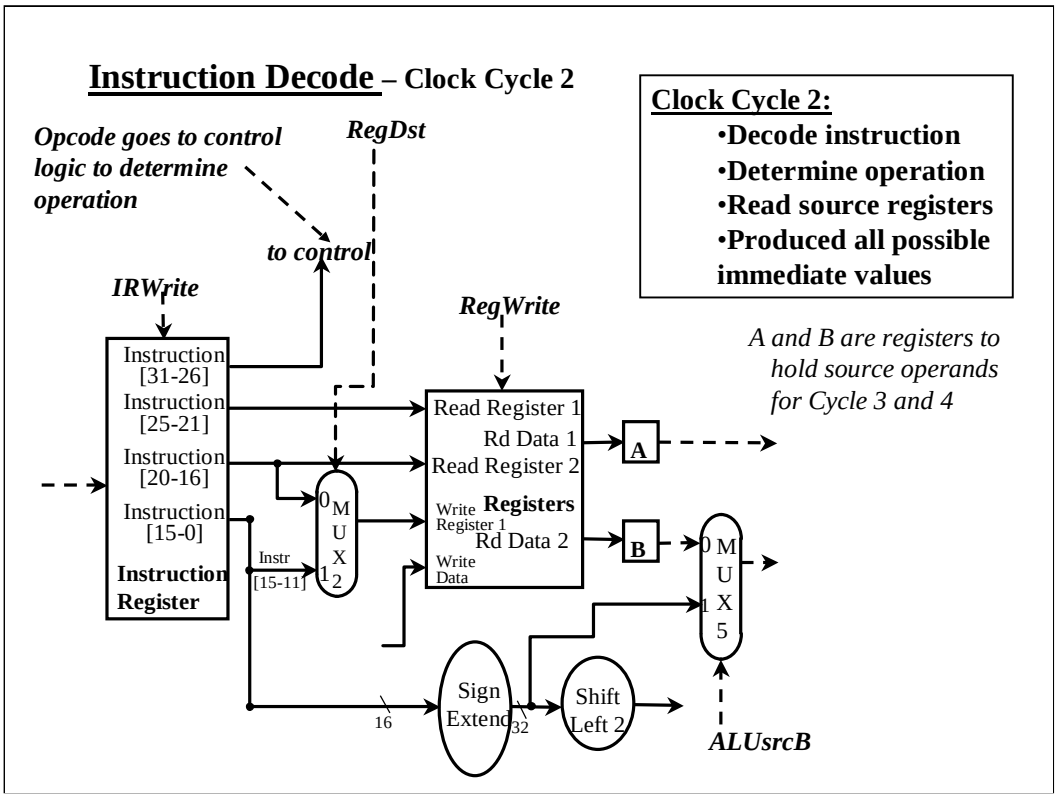


This slide shows in more detail the instruction fetch stage of the CPU: the memory should be considered to be a high speed cache memory.



This slide shows in more detail the instruction fetch stage of the CPU: the Instruction Register is written with the instruction to be executed.

The IRWrite control signal controls this write, and this signal is only active in this first cycle. The instruction is held in IR for the rest of the instruction execution.



All possible fields taken out of instruction

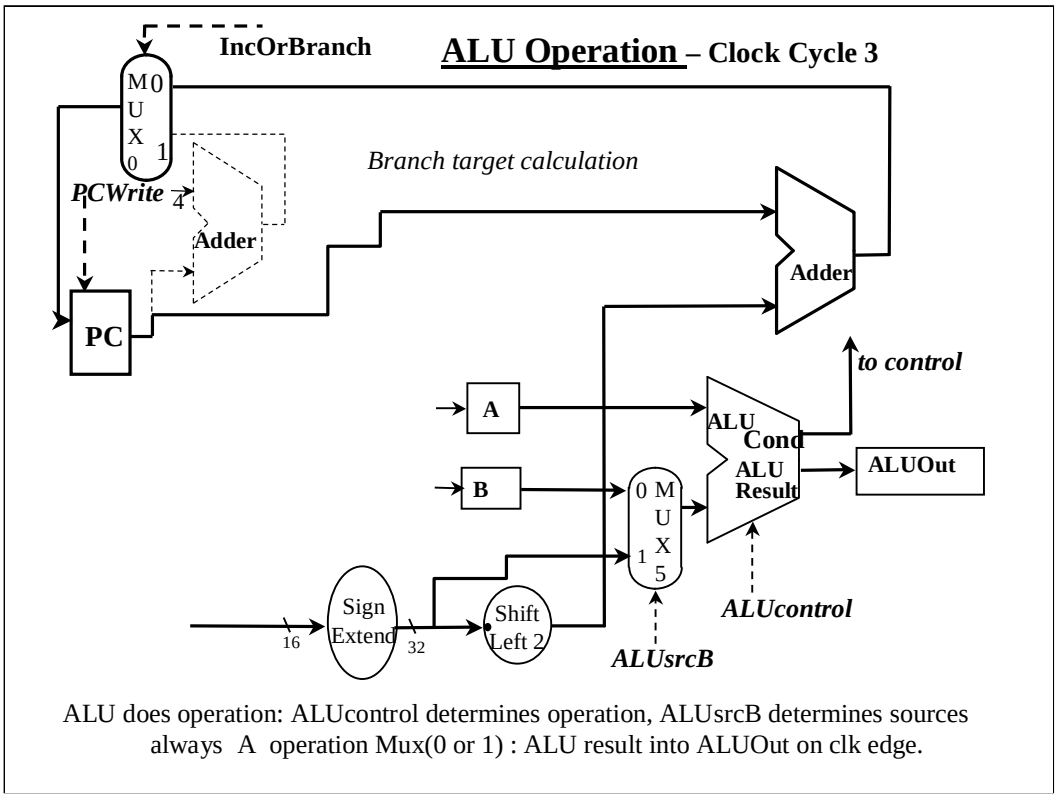
•Opcode (bits 31-26), rs (25-21), rt, rd , offset

Opcode goes to control logic to determine operation

rs and rt fields are passed to register block to read 2 registers.
Register contents are output and stored in A and B by rising clock edge at end of cycle 2.

Offset field (bits 15-0) is sign extended from 16 bits to 32 bits and shifted left by 2.
This provides for branch and addi instructions.

For unsigned operations, there would need to be a zero-extender, but this would clutter diagram.



ALU does operation: ALUcontrol determines operation, ALUsrcB determines sources always A operation Mux(0 or 1) : ALU result into ALUOut on clk edge.

The Alu always operates to produce a result that is stored in ALUOut on the rising edge at the end of the cycle: the result may not always be used.

The contents of register A (the contents of the rs register of the instruction) is one operand to the ALU operation.

Multiplexer 5 determines the other operation under the control of *ALUSrcB* signal:

ALUsrcB = 0: the contents of register B (the contents of the rt register of the instruction) is the 2nd operand to ALU (Rformat instructions).

ALUsrcB = 1: the signed extended offset of the instruction is the 2nd operand to ALU : addi instruction, load and store instructions

At the same time as the ALU is doing the above, the (branch) Adder unit is adding the contents of PC (the updated PC value after cycle 1) to the sign-extended and shifted-left-2 offset to produce the branch target address. This value is passed through mux 0 to the PC.

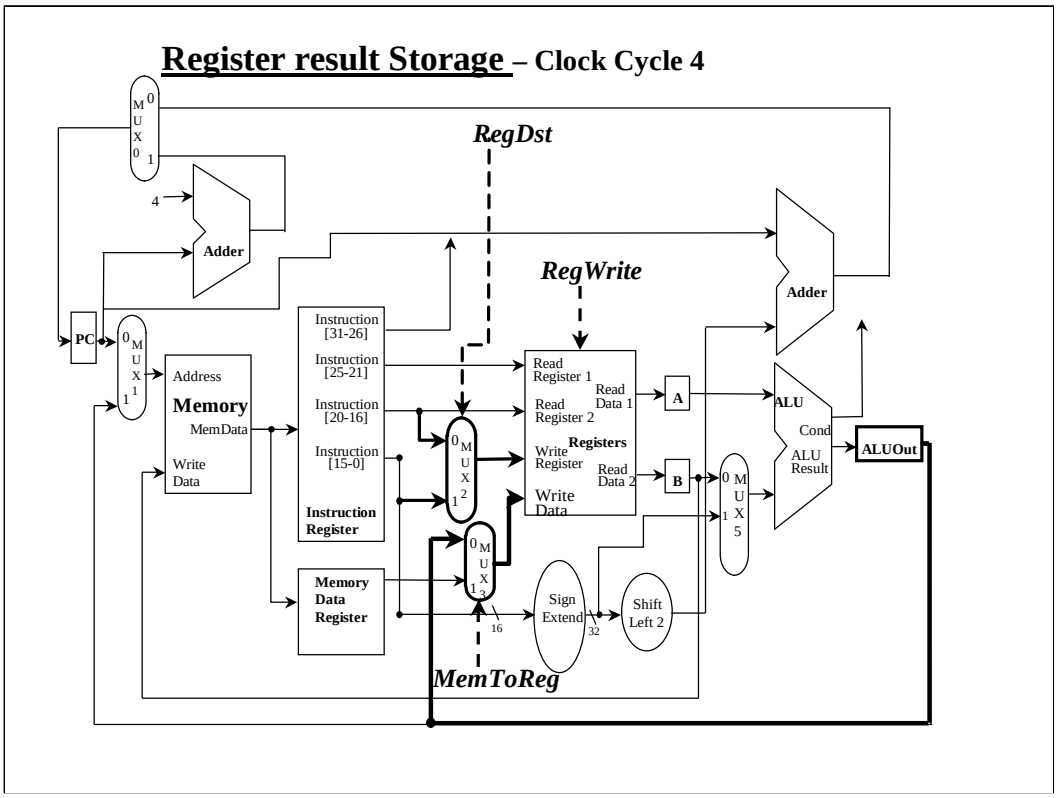
The *PCWrite* control signal will be activated during cycle 3 if the instruction is a branch and the associated branch test comes out true.

If the instruction is a branch, then:-

the ALU will perform a condition test on the operands to the branch

the ALU will set the *Cond* output to true or false depending on the result of the compare

the control logic will use the *Cond* output to activate *PCWrite* control signal if the condition test and the branch is to be taken.



Different operations occur during cycle 4 depending on the instruction.

For arithmetic operations, add, addiu, xor, and, andi, the contents of ALUOut have to be written into the destination, rd, register.

This is shown in the slide.

Control signal *MemToReg* selects the pathway from *ALUout* to the register block data inputs.

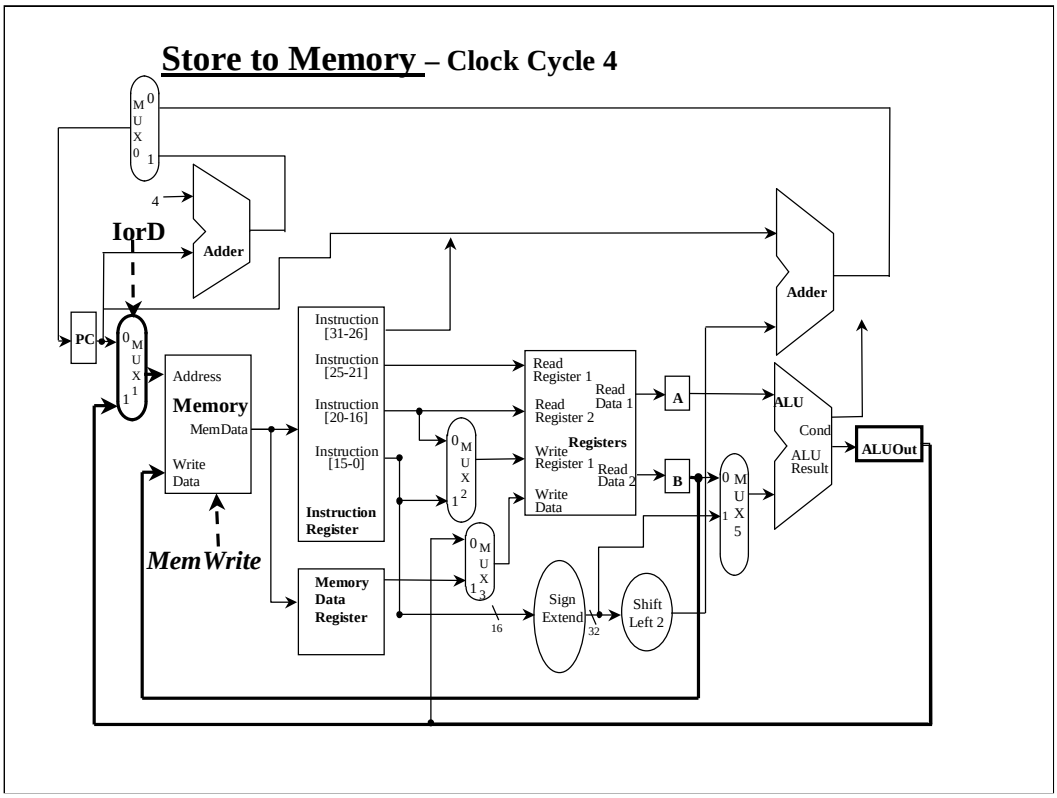
The register written is controlled by the inputs to the Write Register input of the Register block. The value on here is controlled by MUX2.

The control signal *RegDst* determines the routing:-

0 - bits (20-16) are routed to the Write Register inputs: addi, xori, andi (instructions with a 16-bit offset).

1 - bits (15-11) are routed to the Write Register inputs: add, xor, and (instructions with 3 registers).

RegWrite is set active to allow the selected register to be written.



Store instructions complete in Cycle 4.

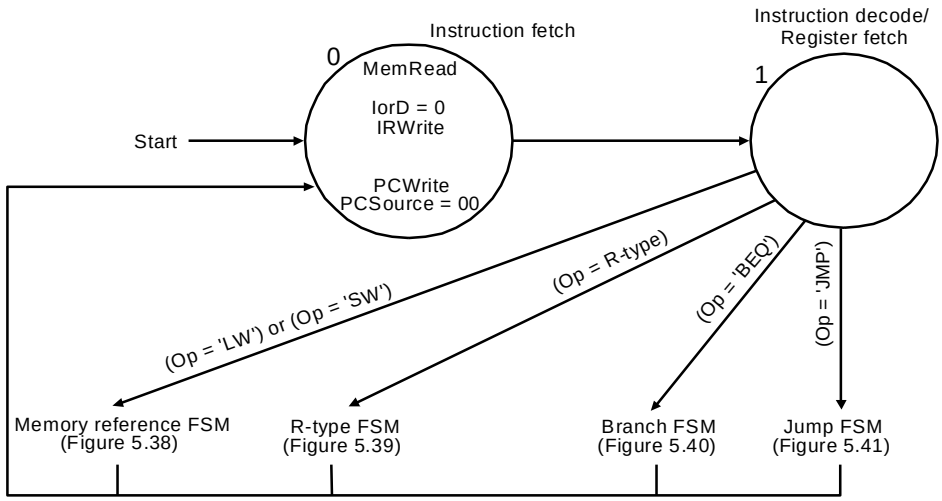
The address to be written is calculated by the ALU in cycle 3 and is stored in *ALUout*.

It is routed via MUX1 under the control of signal *IorD* to the Address bus, and then to the memory.

The data to be written comes from register *rt* and is store in register A, from cycle 2.

The control signal *MemWrite* is activated to write to the Memory.

The first 2 clock cycles of the FSM that defines the control logic



The circles show the first 2 stages of the FSM that is the control logic.

The numbers against the circles are the state assigments: 0 for the fetch stage and 1 for the decode stage.

Inside the circles are shown the important signals and their values.

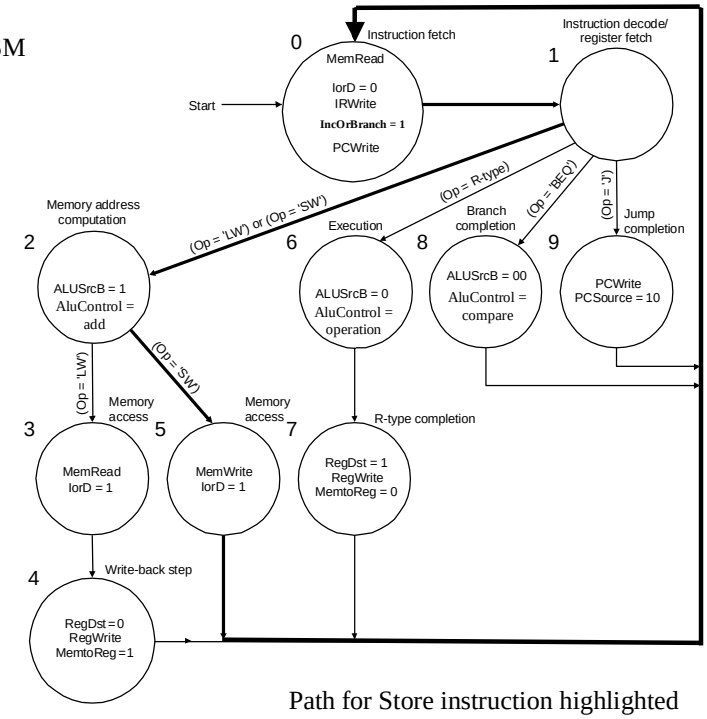
Thus in state 0, the fetch state:

the control signals MemRead, IRWrite, Pcwrite are active to read the memory, enable IR to write the instruction, and PC to write the incremented value at the end of the clock cycle;

the multiplexer controls IorD, PCsource are set to route values through the appropriate multiplexers.

After state 1, the next state is determined by the opcode of the instruction read.

Full FSM



Path for Store instruction highlighted

The states show the output values to be set for each state.

Signals that control multiplexors show signal setting (0 or 1), when these are relevent at other times their values don't matter.

Signals that are active/inactive are only shown when they are active at other times they are inactive.