ICS 233
COMPUTER ARCHITECTURE

MIPS Processor Design

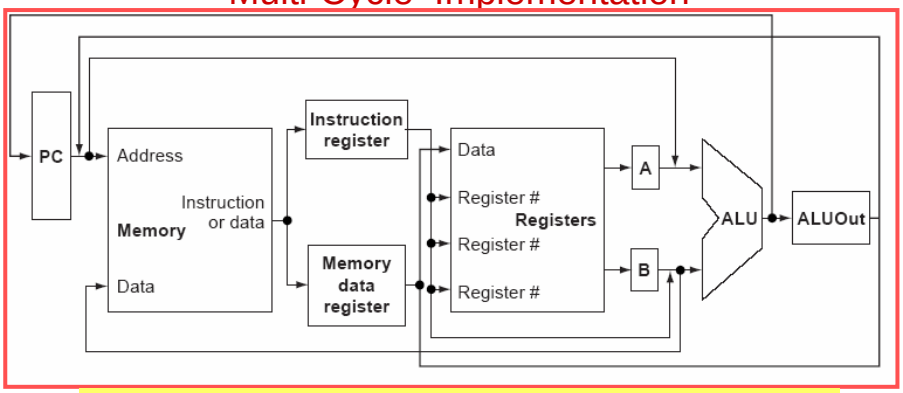
Multicycle Implementation

Lecture 21

Multi-Cycle Implementation

- Each instruction is broken into a series of steps corresponding to the functional unit operations that are needed
- In a multicycle implementation, each step in the execution will take 1 clock cycle.
- The multicycle implementation allows a functional unit to be used more than once per instruction, as long as it is used on different clock cycles.
- This sharing can help to reduce the amount of hardware required
- **Major advantages of a multicycle implementation**
  - ability to allow instructions to take different numbers of clock cycles and
  - ability to share functional units within the execution of a single instruction

Multi-Cycle Implementation



High level view of the multicycle datapath

- A single memory unit is used for both instructions and data
- There is a single ALU, rather than an ALU and two adders
- One or more registers are added after every major functional unit to hold the output of that unit until the value is used in a subsequent clock cycle

Multi-Cycle Implementation

- In the multicycle design, we assume that the clock cycle can accommodate at most one of the following operations:
  - a memory access,
  - a register file access (two reads or one write), or
  - an ALU operation.
- Hence, any data produced by one of these three functional unit (the memory, the register file, or the ALU) must be saved, into a temporary register for use on a later cycle.
- The following temporary registers are added to meet requirements:
- The Instruction registers (IR) and the memory data register (MDR) are added to save the output of the memory for an instruction read and a data read, respectively.
- Two separate registers are used since, both values are needed during the same clock cycle.
- The A and B registers are used to hold the register operand values read from the register file.
- The ALUOut register holds the output of the ALU.

Multi-Cycle Implementation

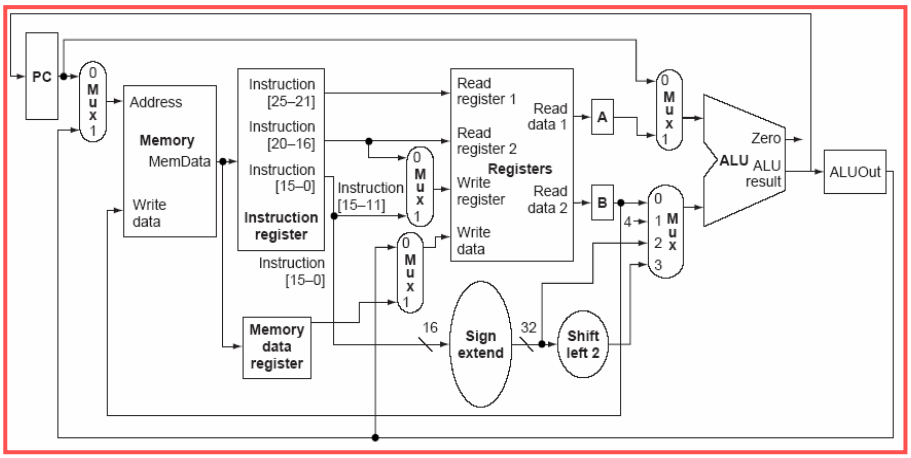
- All the registers except the IR hold data only between a pair of adjacent clock cycles and will thus not need a write control signal.
- The IR needs to hold the instruction until the end of execution of that instruction, and thus will require a write control signal.
- Because several functional units are shared for different purposes, we need both to add multiplexors and to expand existing multiplexors.

Multi-Cycle Implementation

- Since one memory is used for both instructions and data,
  - need a multiplexor to select between the two sources for a memory address, namely, the PC (for instruction access) and ALUOut (for data access)
- Replacing the three ALUs of the single-cycle datapath by a single ALU means that the single ALU must accommodate all the inputs that used to go to the three different ALUs.
- Handling the additional inputs requires two changes to the datapath:
  - An additional multiplexor is added for the first ALU input. The multiplexor chooses between the A register and the PC.
  - The multiplexor on the second ALU input is changed from a two-way to a four-way multplexor.
  - The two additional inputs to the multiplexor are the constant 4 (used to increment the PC) and the sign-extended and shifted offset field (used in the branch address computation).

Multi-Cycle Implementation

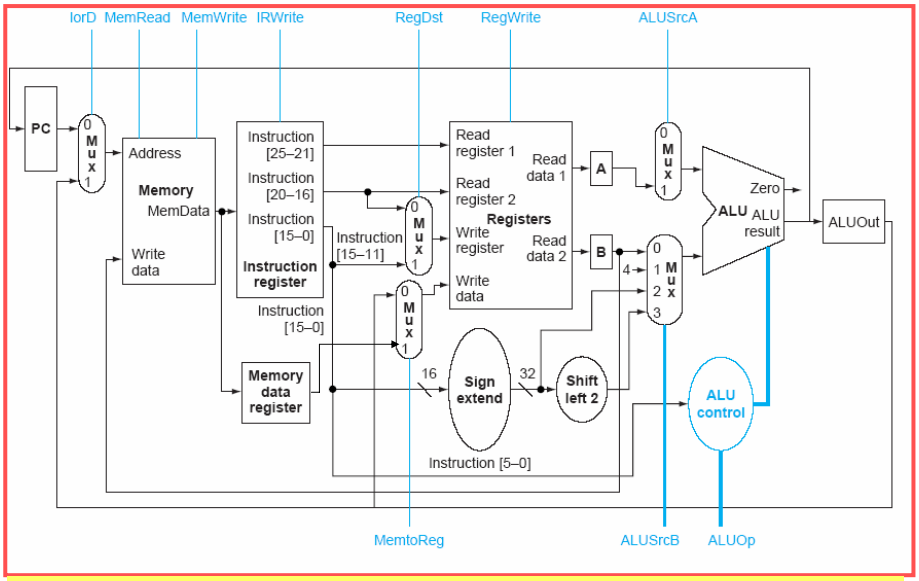
Multicycle datapath for MIPS that handles the basic instructions



Multi-Cycle Implementation

- Because the datapath takes multiple clock cycles per instruction, it will require a different set of control signals.
- The programmer-visible state units (the PC, the memory, and the registers) as well as the IR will need write control signals.
- The memory will also need a read signal.
- Finally, each of the two-input multiplexors requires a single control line, while the four-input mulitplexor requires two control lines.
- The multicycle datapath still requires additions to support branches and jumps; after these additions.

Multi-Cycle Implementation



Multicycle datapath for MIPS with control lines

Multi-Cycle Implementation

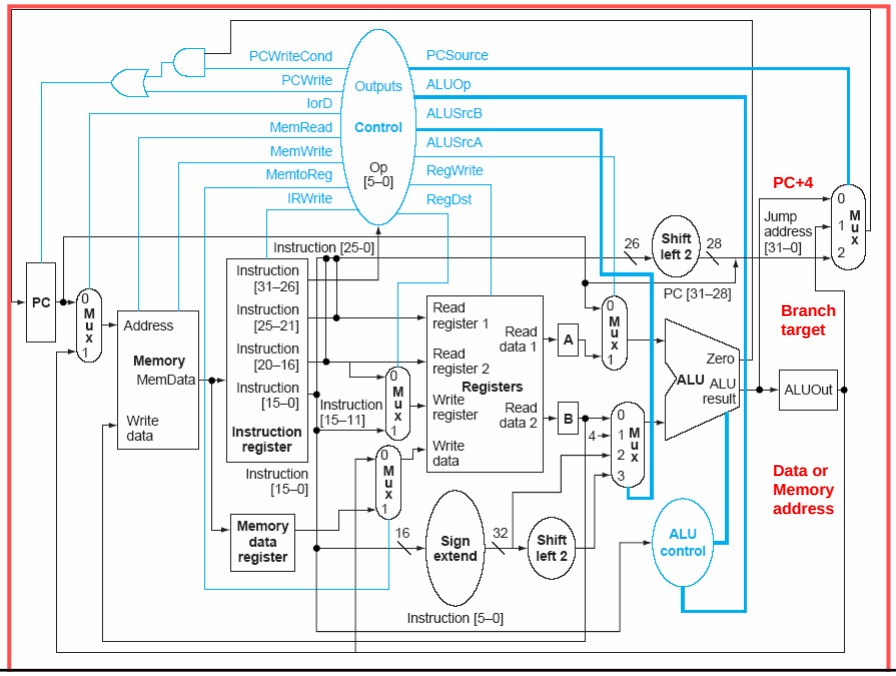
- With the jump instruction and branch instruction, there are three possible sources for the value to be written into the PC:

1. The ouput of the ALU, which is the value PC+4 during instruction fetch. This value should be stored directly into the PC.
2. The register ALUOut, which is where we will store the address of the branch target after it is computed.
3. The lower 26 bits of the Instruction register (IR) shifted left by two and concatenated with the upper 4 bits of the incremented PC, which is the source when the instruction is a jump.

Multi-Cycle Implementation

- During a normal increment and for jumps, the PC is written unconditionally.
- If the instruction is a conditional branch, the incremented PC is replaced with the value in ALUOut only if two designated registers are equal.
- Hence, our implementation uses two separate control signals:
  - **PCWrite**, which causes an unconditional write of the PC, and
  - **PCWriteCond**, which causes a write of the PC if the branch condition is also true.

Complete datapath for Multi-Cycle Implementation with necessary control signals



Multi-Cycle Implementation

Signal name	Effect when deasserted	Effect when asserted
RegDst	The register file destination number for the Write register comes from the rt field.	The register fie destination number for the Write register comes from the rd field.
Regwrite	None	The general-purpose register selected by the Write register number is written with the value of the Write data input.
ALUSrcA	The first ALU operand is the PC	The first ALU operand comes from the A register.
MemRead	None	Content of memory at the location specified by the Address input is put on Memory data output.
MemWrite	None	Memory contents at the location specified by the Address input is replaced by value on Write data input
MemtoReg	The value fed to the register file Write data input comes from ALUOut.	The value fed to the register file Write data input comes from the MDR.
IorD	The PC is used to supply the address to the memory unit.	ALUOut is used to supply the address to the memory unit.
IRWrite	None	The output of the memory is written into the IR.
PCWrite	None	The PC is written; the source is controlled by PCSource.
PCWriteCond	None	The PC is written if the Zero output from the ALU is also active.

Actions of the 1-bit control signals

Multi-Cycle Implementation

Signal name	Value (binary)	Effect
ALUOp	00	The ALU performs an add operation.
	01	The ALU performs a subtract operation.
	10	The funct field of the instruction determines the ALU operation.
ALUSrcB	00	The second input to the ALU comes from the B register.
	01	The second input to the ALU is the constant 4.
	10	The second input to the ALU is the sign-extended, lower 16 bits of the IR.
	11	The second input to the ALU is the sign-extended, lower 16 bits of the IR shifted left 2 bits.
PCSource	00	Output of the ALU (PC + 4) is sent to the PC for writing.
	01	The contents of ALUOut (the branch target address) are sent to the PC for writing
	10	The jump target address (IR[25:0] shifted left 2 bits and concatenated with PC+4[31:28]) is sent to the PC for writing.

Actions of the 2-bit control signals

14