Multi-cycle datapath

Drawbacks of single cycle Datapath

- All instructions must complete in 1 cycle (CPI = 1)
  - different instructions do different amounts of work, for example:
    - **add** uses instruction memory, register file, ALU, register file again
    - **lw** uses instruction memory, register file, ALU, data memory, register file again
- The longest possible path determines the clock cycle

Relative Cycle time :

- What is the longest path (slowest instruction) assuming 4ns for instruction and data memory, 3ns for ALU and adders, and 1ns for register reads or writes? Assume negligible delays for muxes, control unit, sign extend, PC access, shift left by 2,routing, etc

Type	Instruction Memory	Register Read	ALU Operation	Data Memory	Register Write	Total
R-format	4	1	3	0	1	9
lw	4	1	3	4	1	13
sw	4	1	3	4	0	12
beq	4	1	3	0	0	8
j	4	0	0	0	0	4

- In single-cycle, we define a clock cycle to be the length of time needed to execute a single instruction. So, our lower bound on the clock period is the length of the most time-consuming instruction.
- In our previous example, our jump instruction needs only 4ns but our clock period must be 13ns to accommodate the load word instruction!
- Hardware units can only be used once in the cycle
    - some must be replicated (ALU, memory)
    - increased hardware costs

Multi-cycle Datapath

- Instead of viewing the instruction as one big task that needs to be performed, in multi-cycle the instructions are broken up into smaller fundamental steps.
- As a result, we can **shorten the clock period** and **perform the instructions incrementally across multiple cycles.**
- Fewer resources if some can be reused in different cycles

Multi-cycle implementation

- Break up the execution cycle into steps:
  - want each step to contain work that takes about the same amount of time
  - instructions only use the steps they need

R Instruction steps

- An instruction is fetched from **instruction memory** and the PC is incremented.
- Read two source register values from the **register file**.
- Perform the **ALU operation** on the register data operands.
- Write the result of the ALU operation to the **register file**.

Load Instruction steps

- An instruction is fetched from **instruction memory** and the PC is incremented.
- Read a source register value from the **register file** and **sign-extend** the 16 least significant bits of the instruction.
- Perform the **ALU operation** that computes the sum of the value in the register and the sign-extended immediate value from the instruction.
- Access **data memory** at the address given by the result from the ALU.
- Write the result of the memory value to the **register file**.

Store Instruction steps

- An instruction is fetched from **instruction memory** and the PC is incremented.
- Read two source register values from the **register file** and **sign-extend** the 16 least significant bits of the instruction.
- Perform the **ALU operation** that computes the sum of the value in the register and the sign-extended immediate value from the instruction.
- Update **data memory** at the address given by the result from the ALU.

Branch Equal (beq) instruction steps

- An instruction is fetched from **instruction memory** and the PC is incremented.
- Read two source register values from the **register file** and sign-extend the 16 least significant bits of the instruction and then left shifts it by two.
- The **ALU** performs a subtract on the data values read from the register file. The value of PC+4 is added with the sign-extended left-shifted-by-two immediate value from the instruction, which results in the branch target address.
- The Zero result from the ALU is used to decide which adder result should be used to **update the PC**.

Jump Instruction step

- An instruction is fetched from **instruction memory** and the PC is incremented.
- **Concatenate** the four most significant bits of PC+4, the 26 least significant bits of the instruction, and two zero bits. Assign the result to the PC.

General Steps :

- So, generally, we can say we need to perform the following steps:

    1. Instruction fetch.
    2. Instruction decode and register fetch.
    3. Execution, memory address computation, branch completion, or jump completion.
    4. Memory access or R-type instruction completion.
    5. Memory read completion.

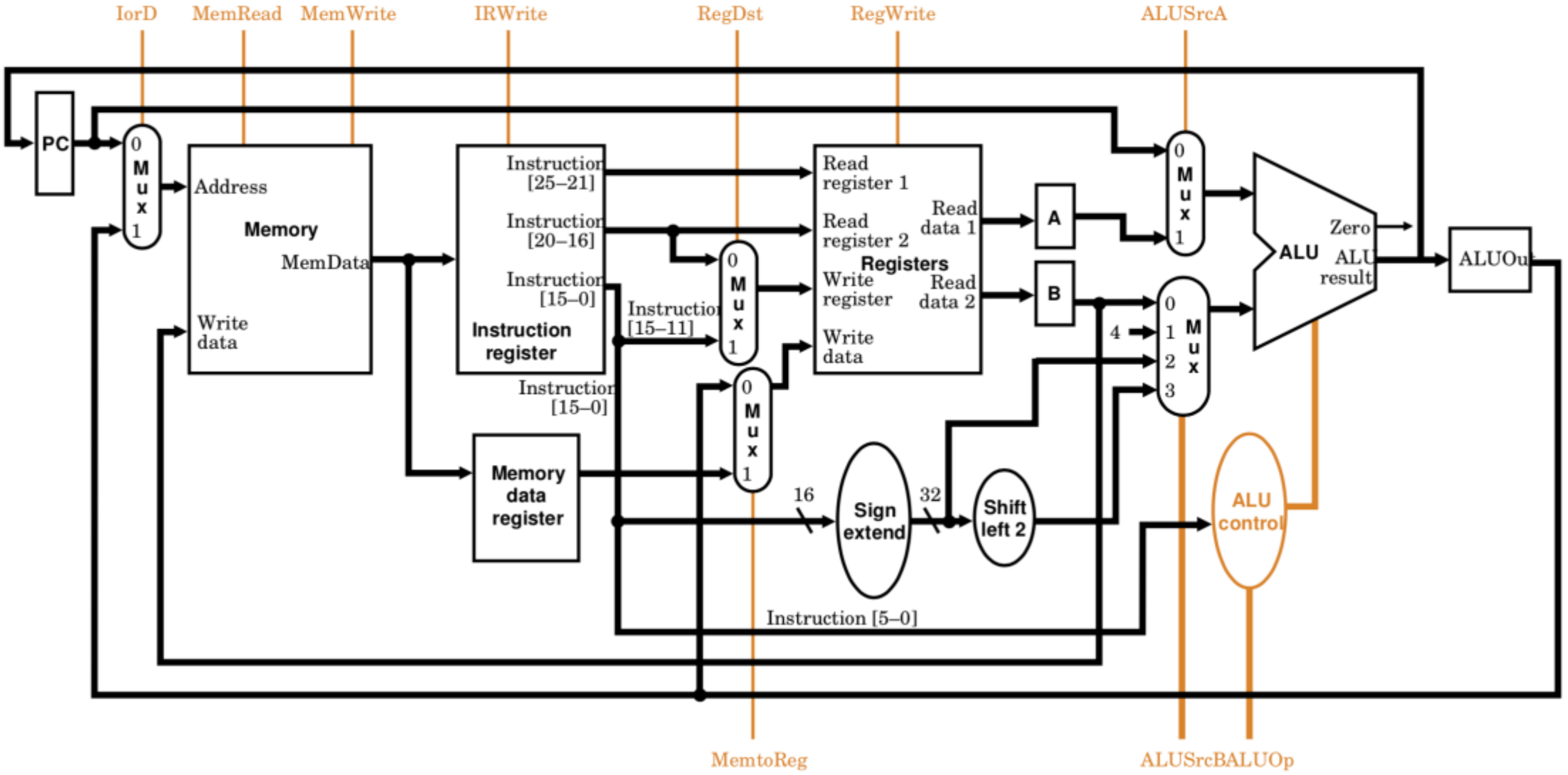
Hardware Changes for Multi-cycle datapath

- Add some temporary registers since some information that is calculated in one cycle is needed in subsequent cycles
    - instruction register (IR)
    - memory data register (MDR)
    - ALU source registers, A and B
    - ALUOut

Data that is calculated in 1 instruction & needed by subsequent instructions is stored in ISA-visible state (PC, registers, memory)

- Larger or more MUXes
    - MUX to memory address
    - MUX to ALU source 1
    - larger MUX to ALU source 2

Multi-cycle datapath

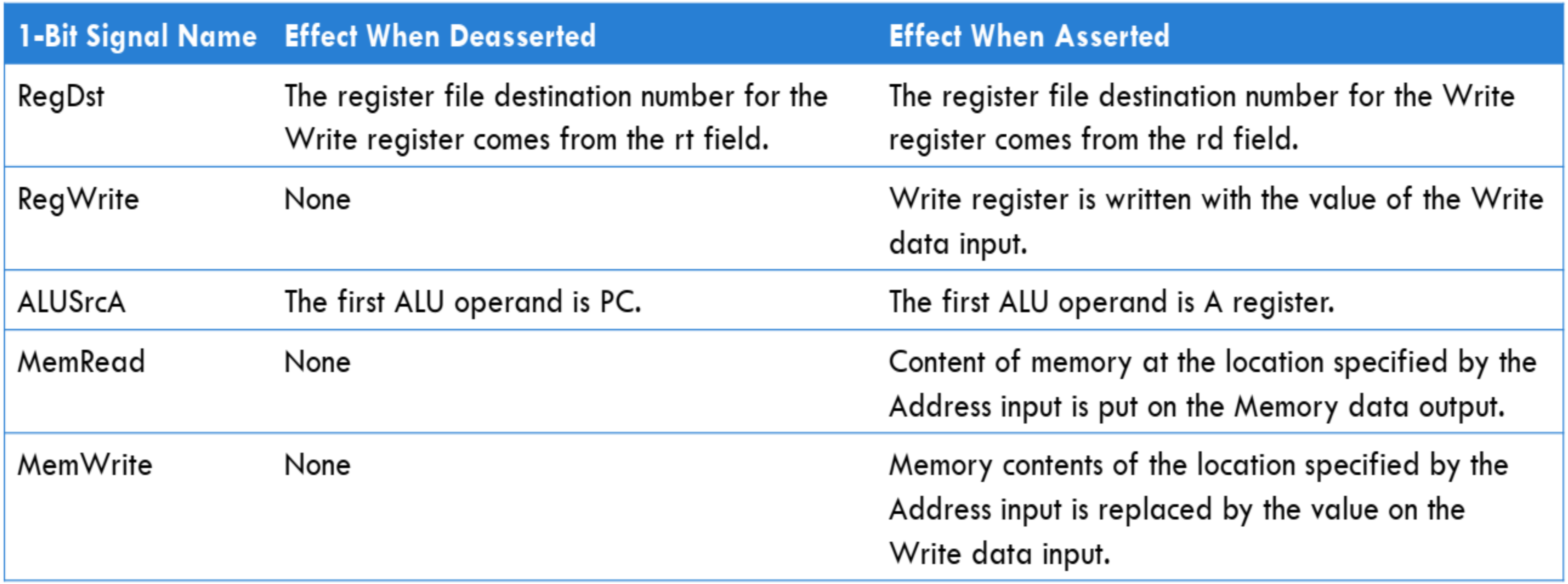


<u>New temporary registers</u>

- **Instruction register (IR)** – holds the instruction after its been pulled from memory.
- **Memory data register (MDR)** – temporarily holds data grabbed from memory until the next cycle.
- **A** – temporarily holds the contents of read register 1 until the next cycle.
- **B** – temporarily holds the contents of read register 2 until the next cycle.
- **ALUout** – temporarily holds the contents of the ALU until the next cycle.

- One of the changes that is made is that we’re using only a single ALU. We have no dedicated adders on the side. To implement this change, we need to add some **multiplexors**.
  - **ALUSrcA** multiplexor chooses between the contents of PC or the contents of temporary register A as the first operand.
  - **ALUSrcB** multiplexor chooses between the contents of temporary register B, the constant 4, the immediate field, or the left-shifted immediate field as the second operand
  - **PCSource** multiplexor chooses between jump target, branch target or normal next instruction address

Multi-cycle Datapath and Control



1-Bit 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 file destination number for the Write register comes from the rd field.
RegWrite	None	Write register is written with the value of the Write data input.
ALUSrcA	The first ALU operand is PC.	The first ALU operand is A register.
MemRead	None	Content of memory at the location specified by the Address input is put on the Memory data output.
MemWrite	None	Memory contents of the location specified by the Address input is replaced by the value on the Write data input.

Asserted – 1; Deasserted - 0

Multi-cycle Datapath and Control

1-Bit Signal Name	Effect When Deasserted	Effect When Asserted
MemToReg	The value fed to the register file input is ALUout.	The value fed to the register file input comes from Memory data register.
lorD	The PC supplies the Address to the Memory element.	ALUOut is used to supply the address to the memory unit.
IRWrite	None	The output of the memory is written into the Instruction Register (IR).
PCWrite	None	The PC is written; the source is controlled by PC-Source.
PCWriteCond	None	The PC is written if the Zero output from the ALU is also active.

Multi-cycle datapath performance

- Assume

  loads: 5 ns,

  stores: 4 ns,

  R-type: 4 ns,

  branches: 3 ns

- percentage in total instructions loads: 22%, stores: 11%, R-type: 50%, branches: 17%
- both implementations have the same number of instructions
- $\text{CPI}_{\text{single}} = 5$
- $\text{CPI}_{\text{multi}} = 5*.22 + 4*.11 + 4*.50 + 3*.17 = 4.05$
- Speedup = 5/4.05 = 1.2

Speedup = Ew/Ee

$$CPI = \frac{CPU\ Clock\ Cycles}{Instruction\ Count}$$

S.No.	Single Cycle Datapath	Multiple Cycle Datapath
1	Instructions are not subdivided.	Instructions are divided into arbitrary number of steps.
2	Clock cycles are long enough for the lowest instruction.	Clock cycles are short but long enough for the lowest instruction.
3	There are only 1 instruction that can be executed at the same time.	There are only 1 instruction that can be executed at the same time.
4	There is 1 cycle per instruction, i, e., CPI = 1.	There is a variable number of clock cycles per instructions.

S.No.	Single Cycle Datapath	Multiple Cycle Datapath
5	Control unit generates signals for the entire instruction.	Control unit generates signals for the instruction’s current step and keeps track of the current step.
6	There is duplicate hardware, because we can use a functional unit for at most one subtask per instruction.	There is no duplicate hardware, because the instructions generally are broken into single FU steps.
7	Extra registers are not required.	Extra registers are required to hold the result of one step for use in the next step.