Outline of Lecture

- **Problems with Single Clock Cycle Datapath**
- **A Multiple CLock Cycle Implementation of a Data-path**

Problems with Single-Cycle Datapath Implementations

- In the datapath that we designed, *every* instruction takes a single clock cycle - thus the CPI = 1.
- The length of the clock cycle is determined by the *longest* possible path in the datapath.
- This longest path is for the `load` instruction. It uses 5 functional units in series: Instruction memory, register file, ALU, data memory, and register file.
- Even though each instruction takes just one clock cycle, this clock cycle is large - thus the datapath would have a *poor* overall performance.

Example

Assume that the operation time for the major functional units of the datapath that we built are:

Memory Unit: 2 nsec.

ALU and adders: 2 nsec.

Register file (read or write): 1 nsec.

Multiplexers, control unit, etc. have no delay

Which of the following implementation would be faster?

1) An implementation in which every instruction operates in one clock cycle of a fixed length

1) An implementation where every instruction executes in one clock cycle of a variable-length clock, which for each instruction is only as long as it needs to be.

Answer

$$CPU\ \ execution\ \ time =$$

$$Instruction\ \ count \times CPI \times Clock\ \ cycle\ \ time$$

$$CPU\ \ execution\ \ time =$$

$$= Instruction\ \ count \times Clock\ \ cycle\ \ time$$

The critical path for the different instruction types is as follows:

Instruction type	Functional units used by the instruction type				
R-format	Instruction fetch	Register access	ALU	Register access	
Load word	Instruction fetch	Register access	ALU	memory access	Register access
Store word	Instruction fetch	Register access	ALU	memory access	
Branch	Instruction fetch	Register access	ALU		
Jump	Instruction fetch				

Length of each Instruction

Using the critical paths, we can compute the required length for each instruction type:

Instruction type	Instruction memory	Register read	ALU operation	Data memory	Register write	Total
R-format operations	2	1	2	0	1	6 ns
Load word	2	1	2	2	1	8 ns
Store word	2	1	2	2		7 ns
Branch	2	1	2			5 ns
Jump	2					2 ns

The clock cycle for the datapath with a single clock for all instructions is 8 nsec.

The datapath with a variable clock has a cycle between 2 nsec and 8 nsec.

Typical programs have: 24% loads, 12% stores, 44% R-format instructions, 18% branches, and 2% jumps - thus the average time per instruction is 6.3 nsec - (*almost 12.7 % better*) than single-clock cycle.

Restrictions

- Unfortunately, it is very difficult to build a variable-length clock cycle.
- Also, the penalty for using a single length clock cycle becomes more severe had we considered other instructions such as *multiplcation* or *floating-point operations* which take a long time to complete.
- As a result, the best solution is to consider a small clock cycle - derived from the basic functional unit delays - and allow each instruction to be executed in a *multiple number of clock cycles*.

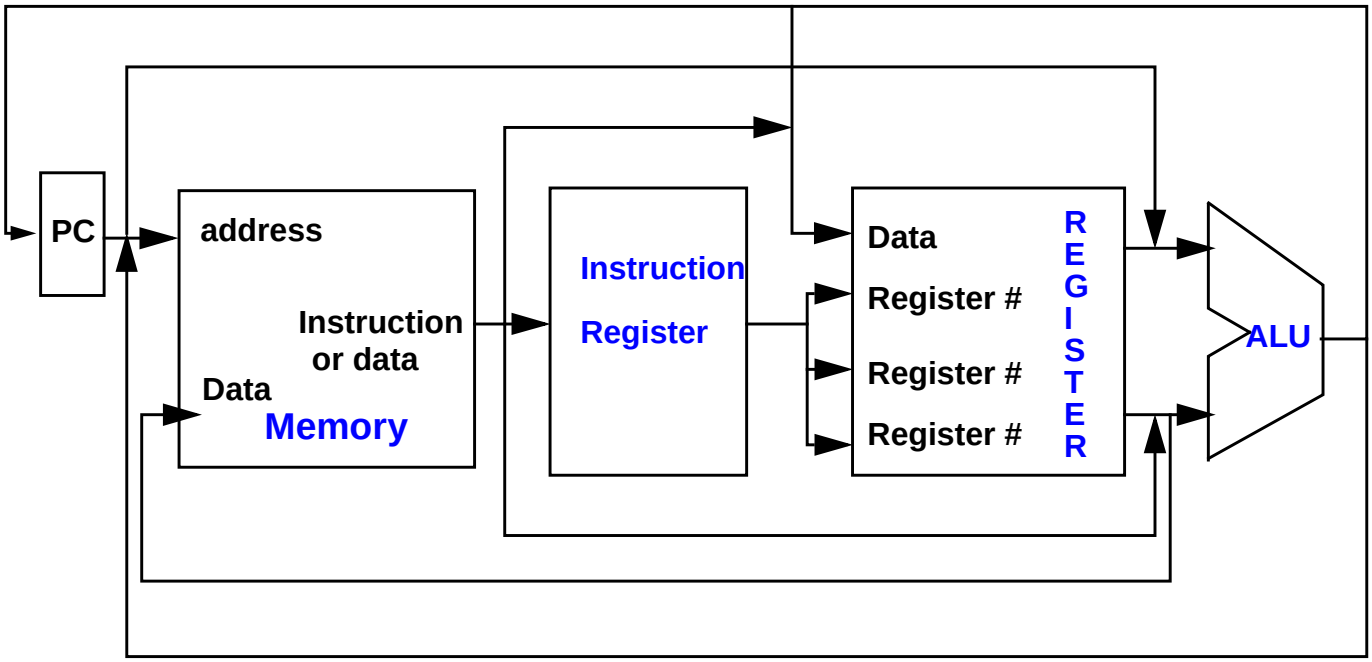
A Multiple Clock Cycle Implementation

- To execute each instruction, the instruction has to go through a number of functional units.

Instruction type	Functional units used by the instruction type				
R-format	Instruction fetch	Register access	ALU	Register access	
Load word	Instruction fetch	Register access	ALU	Memory access	Register access
Store word	Instruction fetch	Register access	ALU	Memory access	
Branch	Instruction fetch	Register access	ALU		
Jump	Instruction fetch				

- We let each functional unit take one clock cycle. • Then, each instruction will take a ***variable*** number of clock cycles.

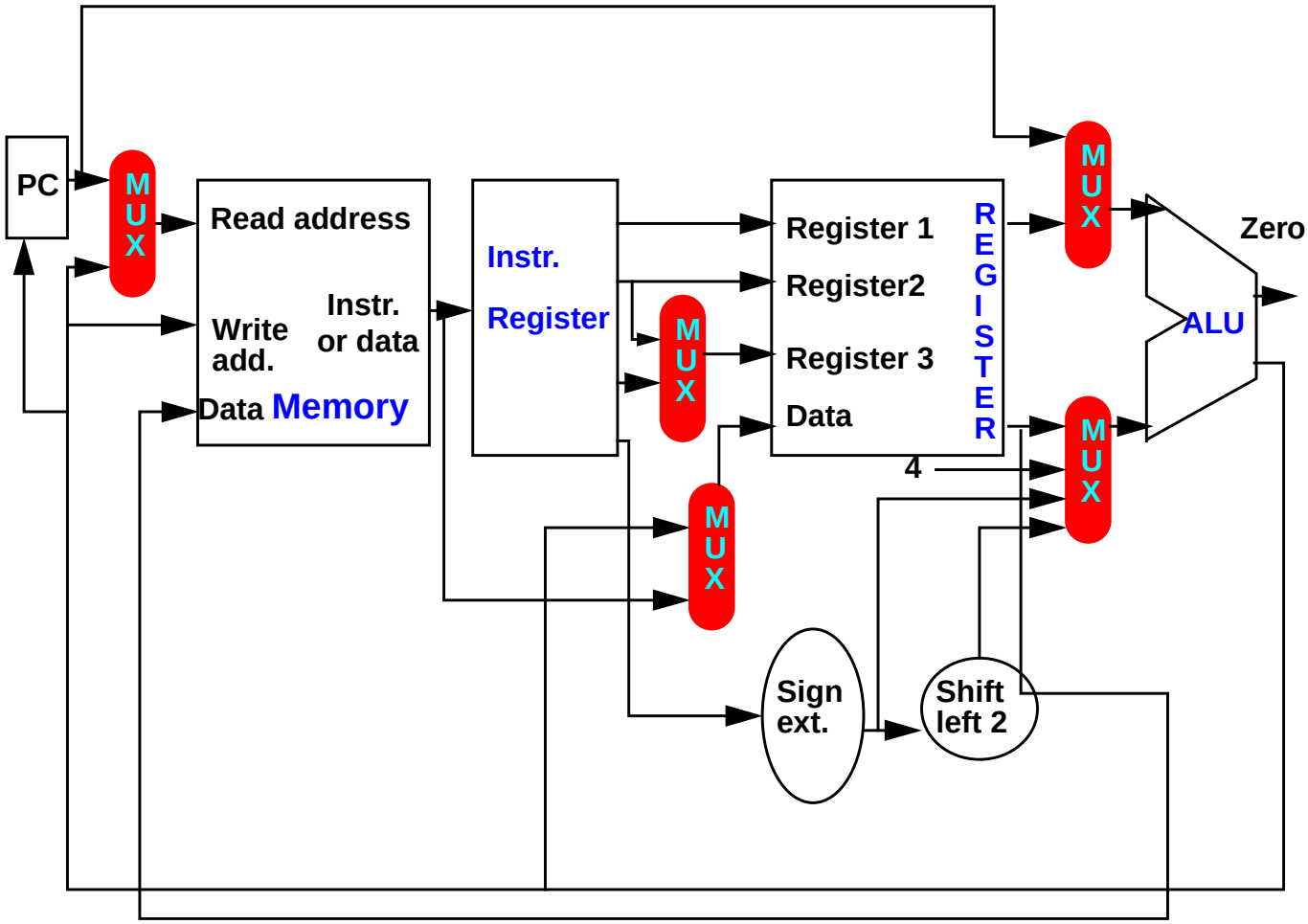
- **Each functional unit can be used by more than instruction as long as it is used in different clock cycles - it can help reduce the amount of hardware needed in the datapath (*better sharing of resources*).**



Differences

- Compared to our single-cycle datapath, these are the differences:
  - ➜ A single memory unit is used for both instructions and data.
  - ➜ A register - *Instruction register* - is used to save the instruction after it is read.
  - ➜ There is a single ALU, rather than an ALU and 2 adders.
- Since there is a lot of sharing of resources (memory, ALU, etc.), we need to add *multiplexers* to accomplish a good sharing of resources - the datapath including the multiplexers is shown next.

- By using multiplexers, we reduce the number of memory units from 2 to 1, and eliminate two adders.



Further Reading

Chapter 5. David A. Patterson and John L. Hennessy. *Computer Organization & Design: The Hardware / Software Interface*. Morgan Kaufman (page 371-382).