



Lecture 09: Datapath and Control

Topics

- Datapath
 - x stages performing computation [$x=5$]
 - y hardware components needed
- Combinational and Sequential logic
 - Logic Gates

SONGS ABOUT COMPUTER SCIENCE

DIGITAL LOGIC

Written by Emmanuel Schanzer

To the tune of: A Spoonful of Sugar

http://www.cs.utexas.edu/users/walter/cs-songbook/digital_logic.html

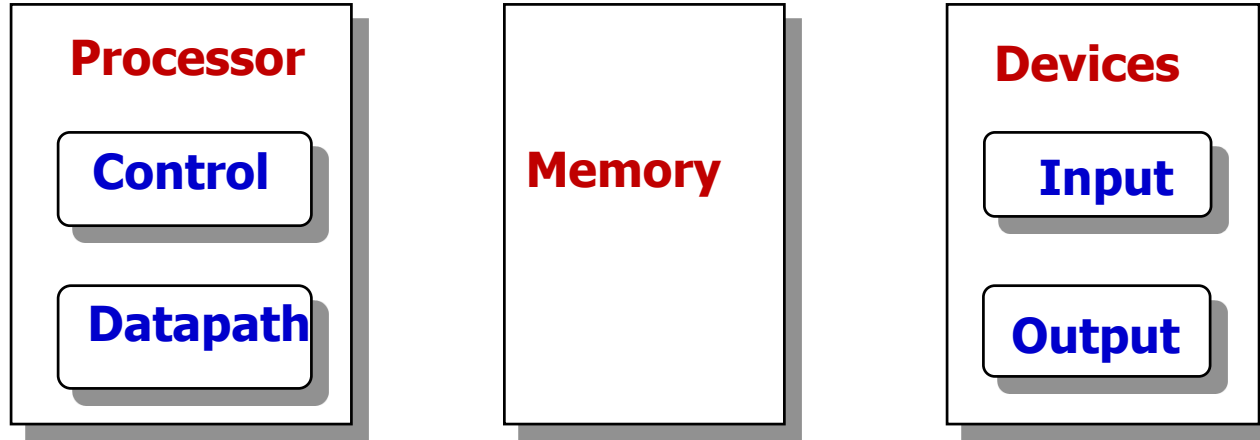
To get all our programs to run
There must be compilation done
Into ones and zer-os:
Seems rather tame.

But how do ones and zeros make
Our hardware bump and shake?
Trans-..is-..tor..gates!
It won't be hard to see.
We Use...

Digital logic: inputs, switches...and grounds.
inputs, switches...and grounds.
inputs, switches...and grounds.
Just by abstracting voltage, into zer-o's and ones.
We can hack all night and day.

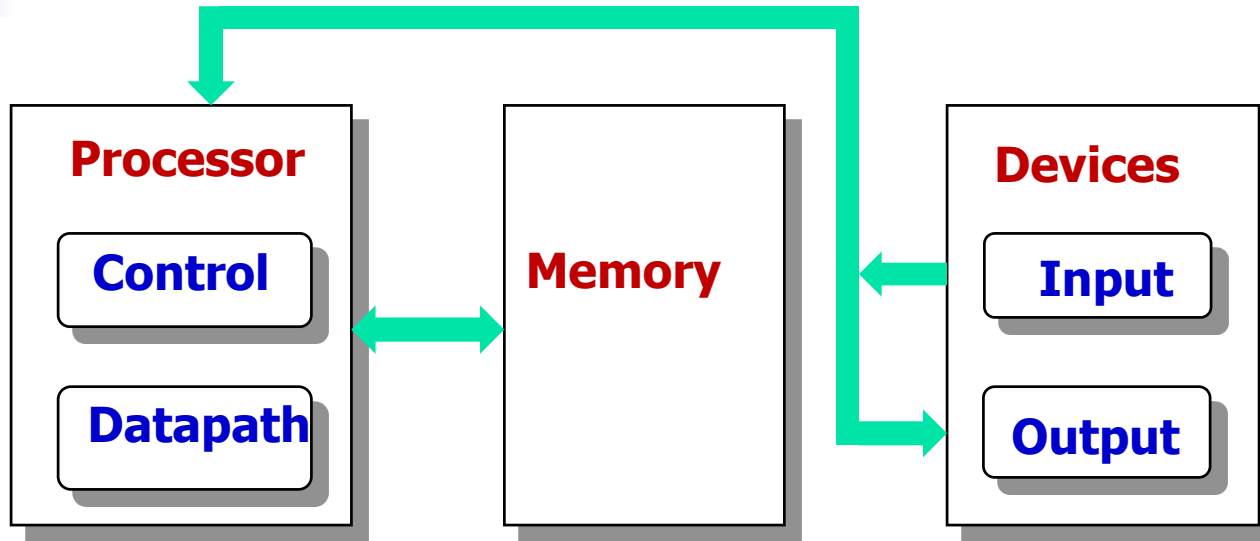
... ..

Five components of a Computer



- **Processor (CPU):**
 - **Datapath:** Elements that process data and addresses in the CPU [e.g. registers, ALUs, ...]
 - **Control:** Determines which computation is performed [e.g. Routes data thru datapath (which regs, which ALU op)]

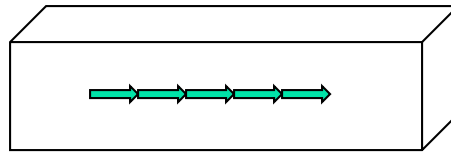
Connection and Communication via Bus



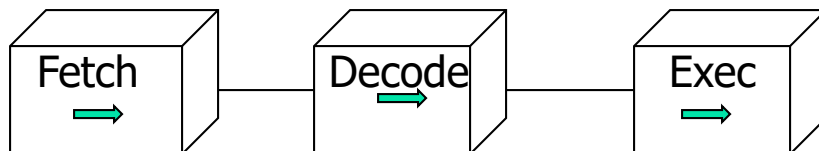
- A Bus is a shared communication link
- Single set of wires used to connect multiple subsystems
- A Bus is also a fundamental tool for composing large, complex systems
 - ... this topic is fully covered in Computer Architecture course...

Datapath concept

- Possible approach: build a single block to “execute an instruction”, which designed to perform all necessary operations starting from fetching the instruction
 - Possible, but too bulky, inefficient and inflexible



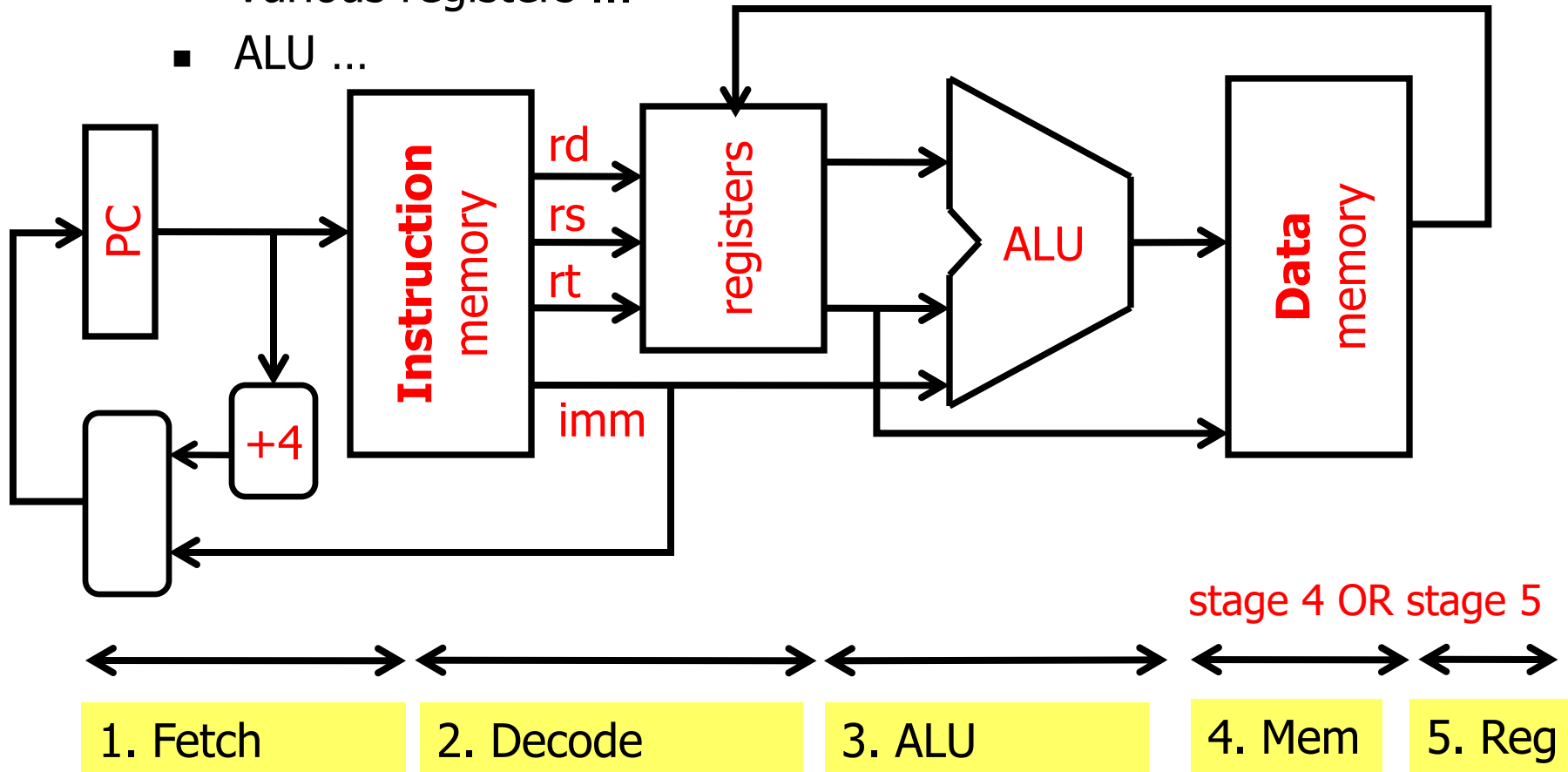
- Modular approach: break up the process of “executing an instruction” into stages, then connect stages... datapath is born!
 - Advantages:
 - smaller, easier to understand and easier design blocks
 - easier to optimise: one block can be changed without affecting the others



Elements that process data and addresses in the CPU

Datapath Stages

- PC (Program Counter)...
- Memory...
- Various registers ...
- ALU ...

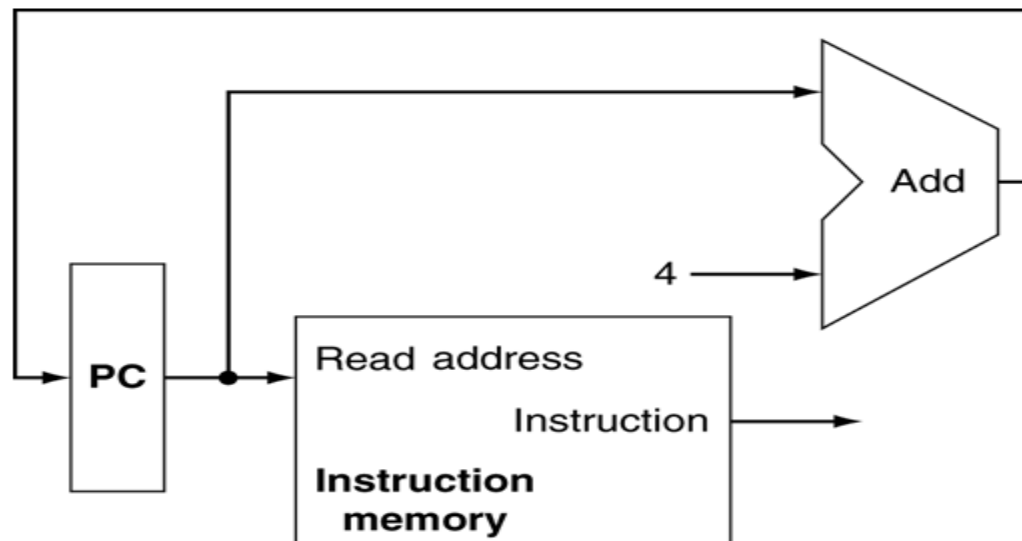
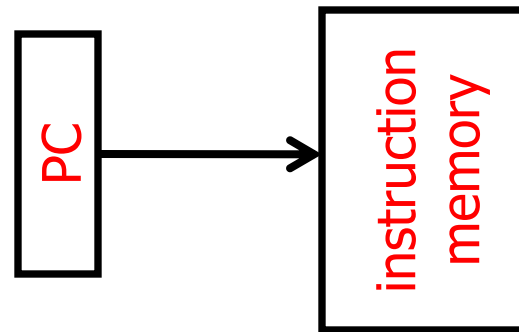


Datapath Stages: 1

- There is a wide variety of MIPS instructions: what general steps do they have in common?

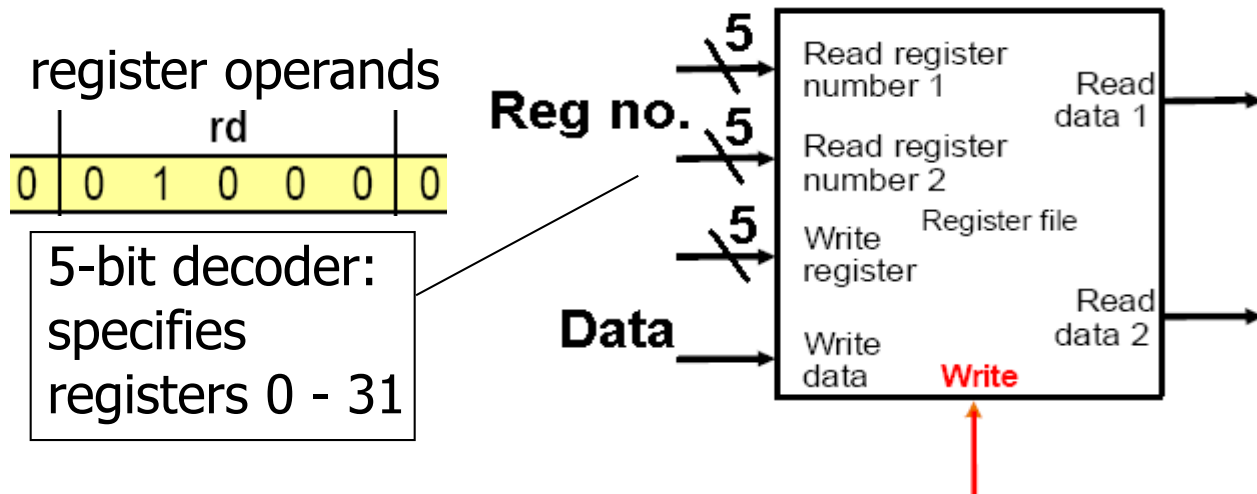
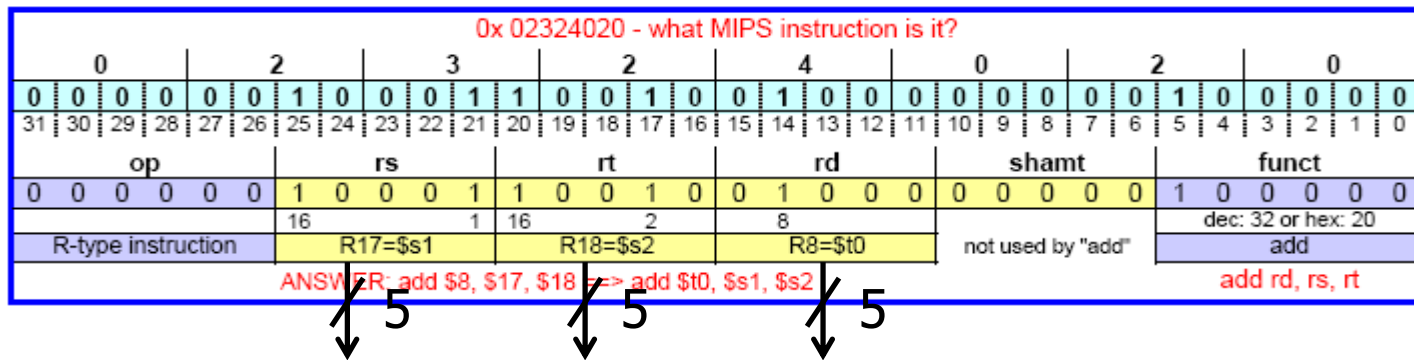
Stage 1: **Instruction Fetch**

$PC = PC + 4$

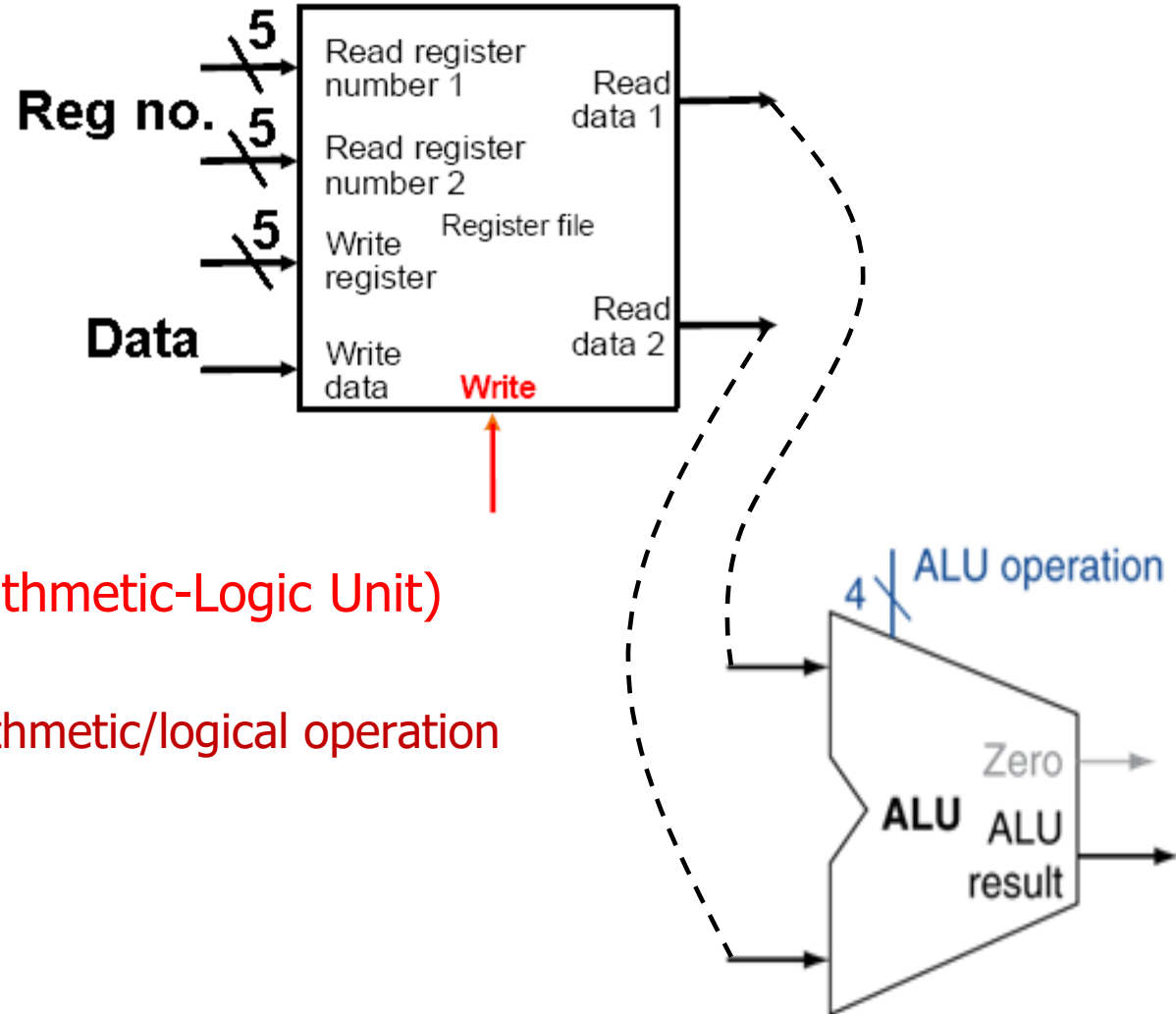


Datapath Stages: 2

Stage 2: Instruction Decode [**0x 02324020** --- refer to 'instruction decoding.pdf' in materials folder]



Datapath Stages: 3



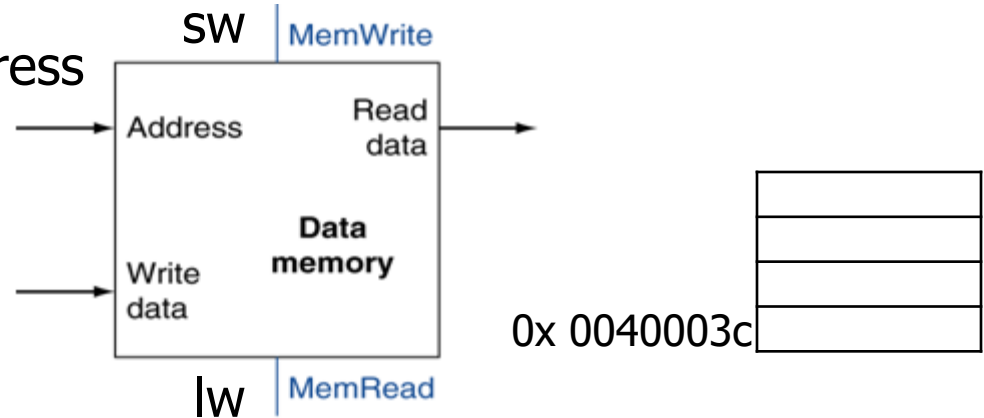
Stage 3: ALU (Arithmetic-Logic Unit)

- ...
- Perform arithmetic/logical operation
- ...

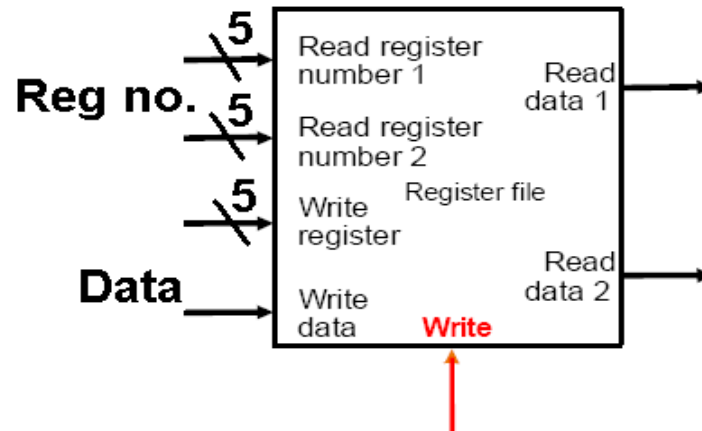
Datapath Stages: 4 and 5

Stage 4: Memory Access (load/store data)

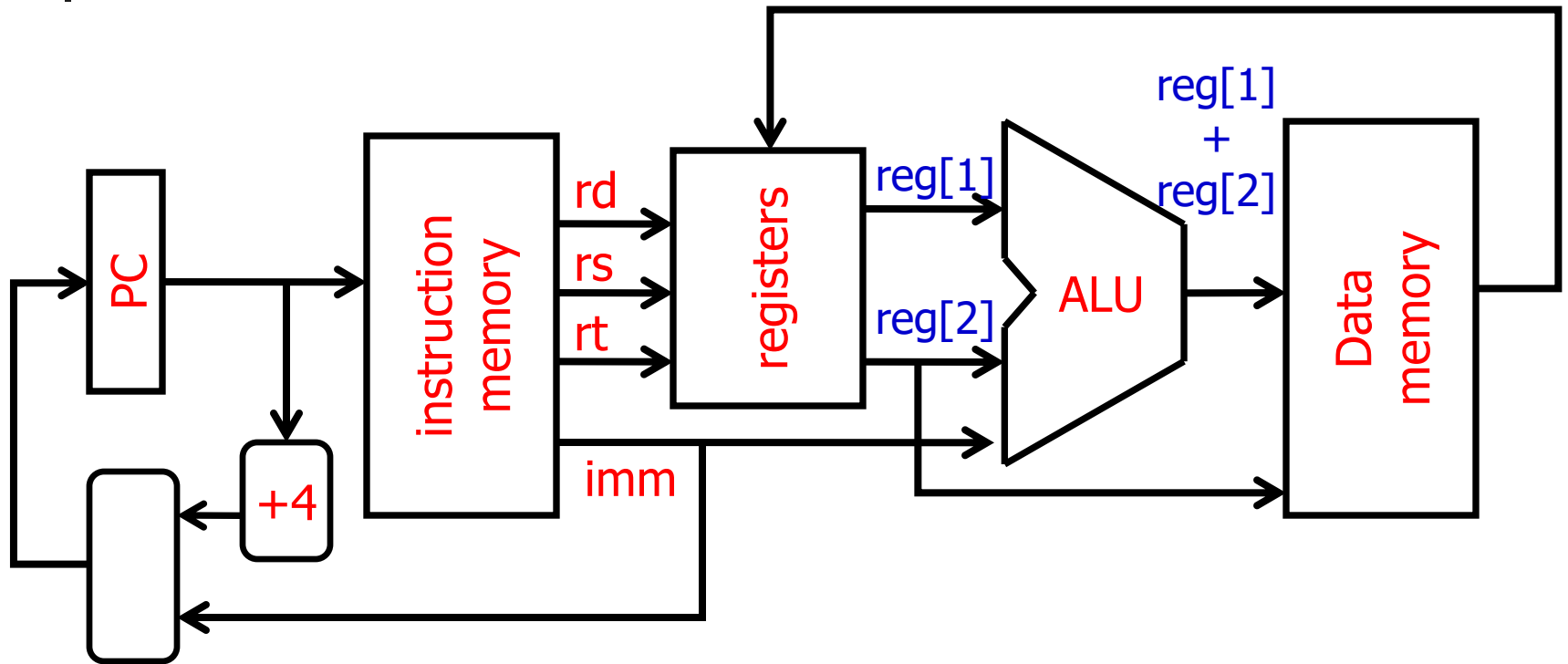
Calculate memory address



Stage 5: Register Write



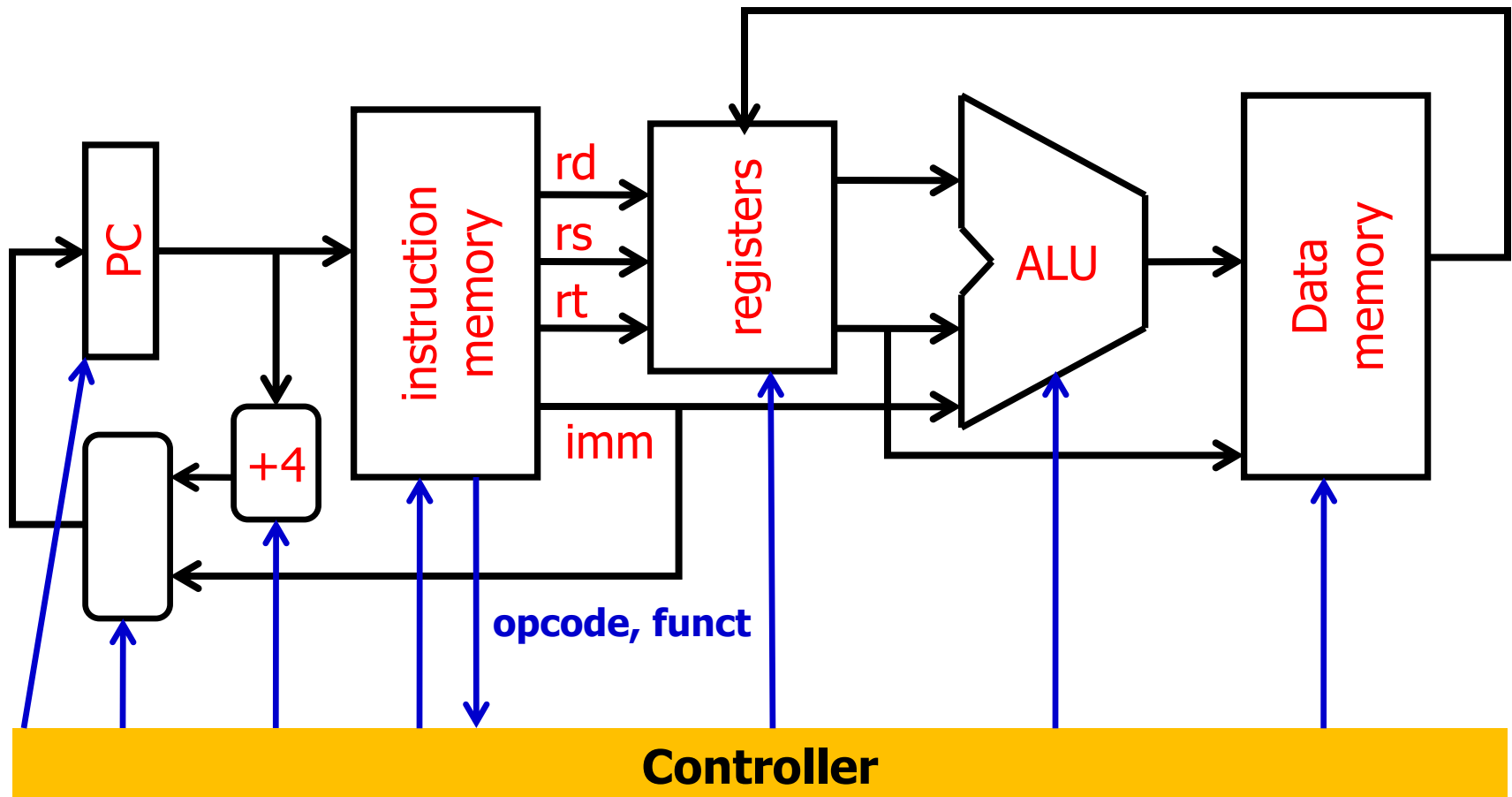
Datapath Instruction Example



look at a sample instruction add: **add \$r3, \$r1, \$r2** # $r3 = r1 + r2$

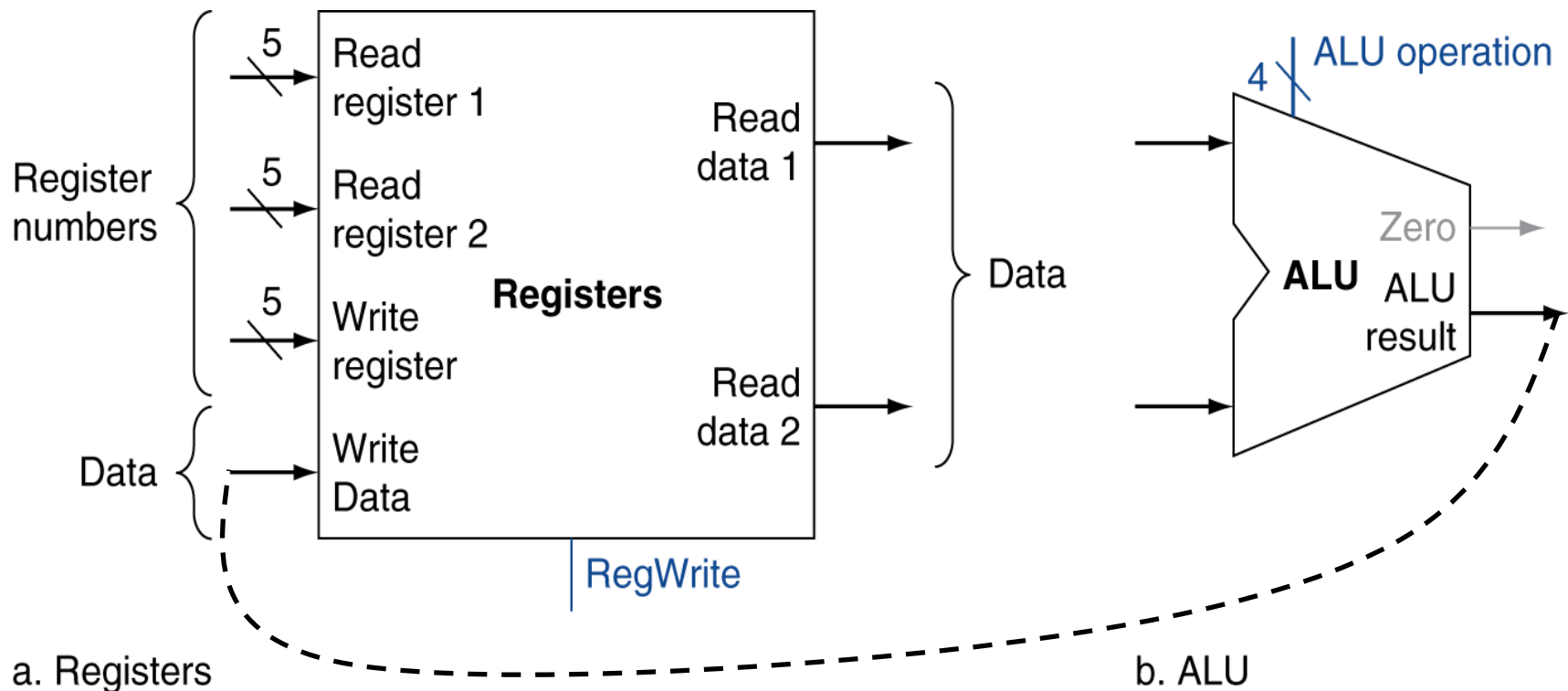
Role of Controller

- Controller causes the right transfers to happen.



R-Format Instructions

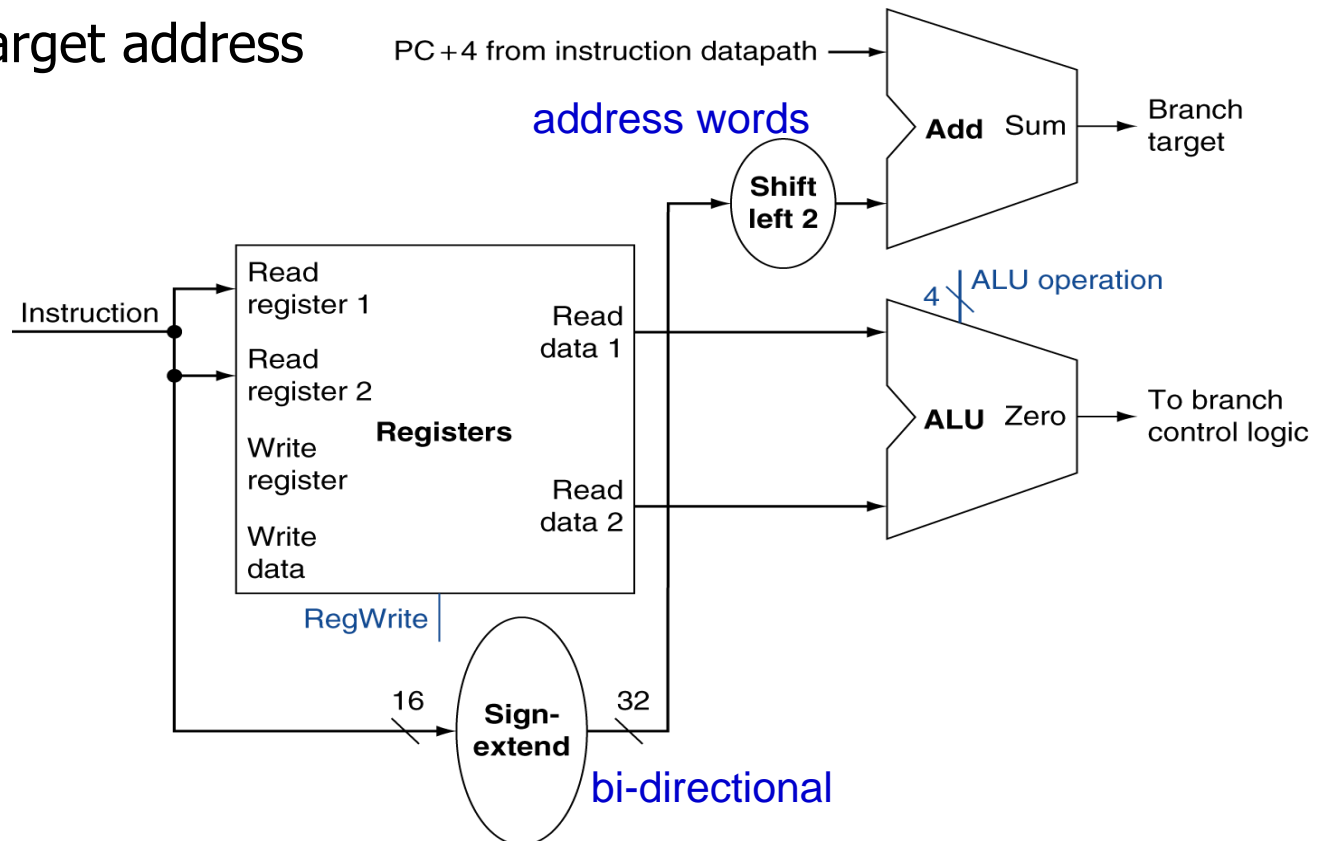
- Read two register operands
- Perform arithmetic/logical operation
- Write register result



Branch Instructions

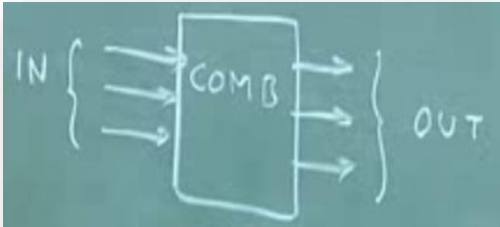
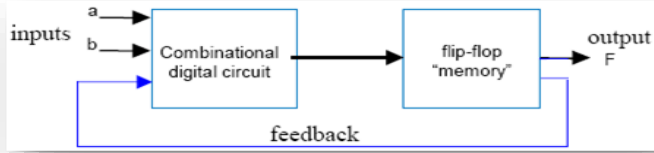
- Read register operands
- Compare operands
 - Use ALU, subtract and check Zero output
- Calculate target address

bne \$s0, \$s1, Exit



Combinational and Sequential Circuits

- A complex functional component is implemented using lower level components - there are two classes of logic circuits:

| COMBINATIONAL circuits | SEQUENTIAL/STATE circuits |
|--|--|
| No memory | Have internal memory (or: elements that contain state) |
| <p>The output depends only on the input</p>  | <ul style="list-style-type: none"> ■ the output depends both on the set of inputs supplied and the value stored in memory, which is called the state of the logic block ■ how to ensure memory element is updated neither too soon, nor too late?  |
| Truth Tables, Boolean Algebra | Finite state machine (Mealy, Moore) |

- ...more on sequential circuits later, first: combinational logic...



Truth Tables, Boolean Algebra

- Combinational logic does not have memory, so it's described by defining the values of the outputs for each possible set of input values, as in **truth tables**.
- Another approach: one can express the logic function with logic equations with the use of **Boolean algebra**.

Main Boolean algebra laws: [“+” is operator OR, and “•” is operator AND]

| | |
|-------------------|---|
| Identity law | $A+0=A$ and $A \bullet 1=A$ |
| Zero and One laws | $A+1=1$ and $A \bullet 0=0$ |
| Inverse laws | $A+\bar{A}=1$ and $A \bullet \bar{A}=0$ |
| Commutative laws | $A+B=B+A$ and $A \bullet B= B \bullet A$ |
| Associative laws | $A+(B+C)=(A+B)+C$ and $A \bullet (B \bullet C)=(A \bullet B) \bullet C$ |
| Distributive laws | $A \bullet (B+C)=(A \bullet B)+ (A \bullet C)$ and $A+(B \bullet C)=(A+B) \bullet (A+C)$ |

Logic Gates, Inverter and Multiplexor (Mux)

1. **AND** gate ($c = a \cdot b$)



| a | b | $c = a \cdot b$ |
|---|---|-----------------|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

2. **OR** gate ($c = a + b$)



| a | b | $c = a + b$ |
|---|---|-------------|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

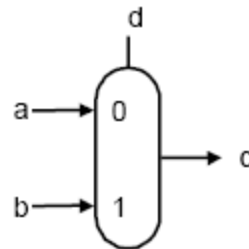
3. **Inverter** ($c = \bar{a}$)



| a | $c = \bar{a}$ |
|---|---------------|
| 0 | 1 |
| 1 | 0 |

4. **Multiplexor (Mux)**

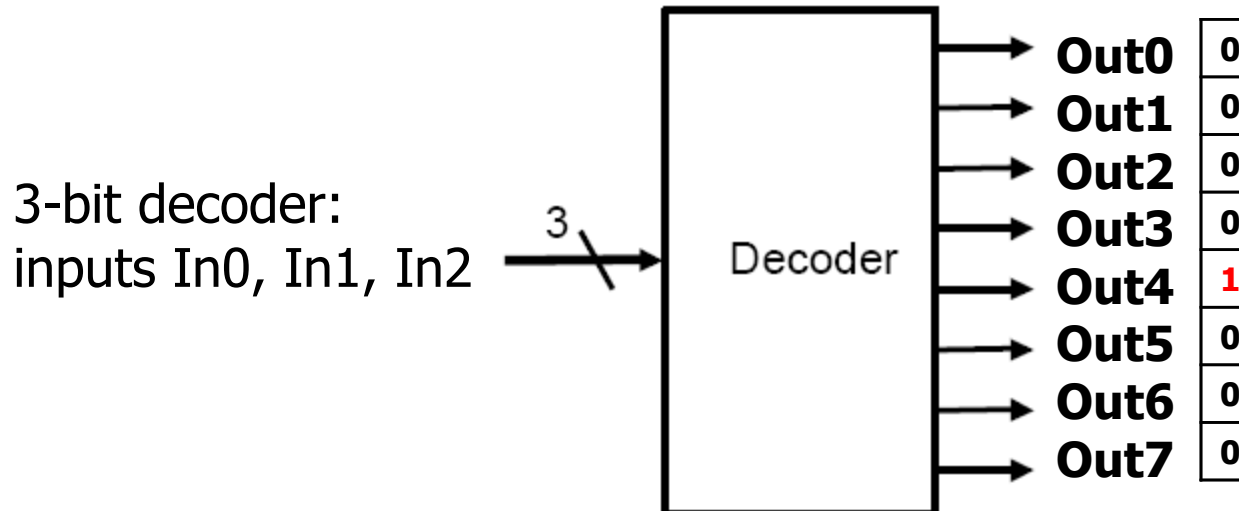
(if $d = 0$, $c = a$;
else $c = b$)



| d | c |
|---|---|
| 0 | a |
| 1 | b |

Selector (Decoder)

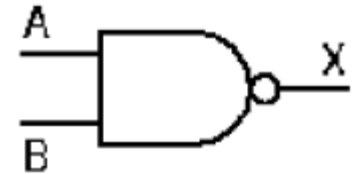
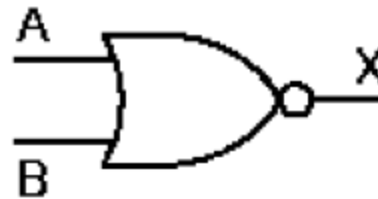
- Only one output is asserted for each input combination
- For example:
if inputs In0, In1, In2 are: **1 0 0** (decimal 4):
Out4 = **1**, all other outputs are **0**



Convert one bit-pattern to another bit-pattern

Universal Gates [Exercise]

- In fact ALL logic functions can be constructed with only a single gate type, if it is inverting.
- Common inverting gates are NOR (inverted OR) and NAND (inverted AND).
- NOR and NAND are called **universal gates**.



Exercise:

prove the above by implementing AND, OR and NOT:

1. using only NOR gates,
2. using only NAND gates.

| A | B | X |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

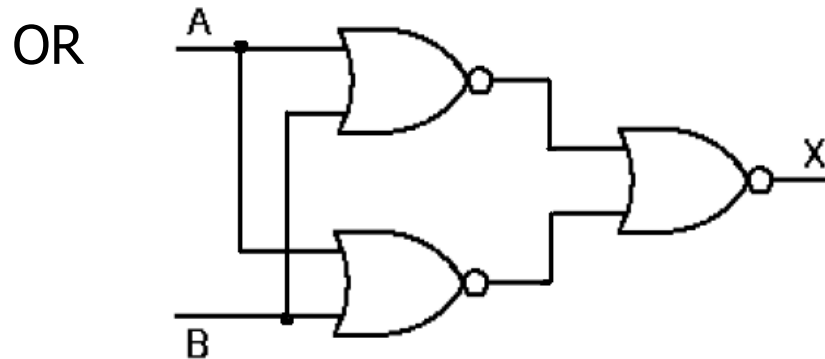
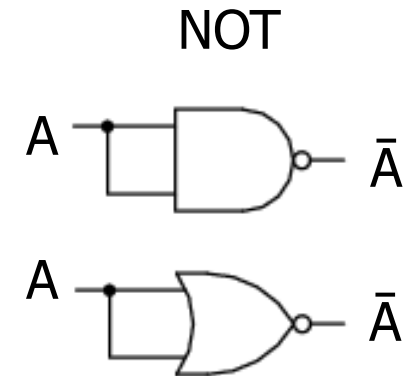
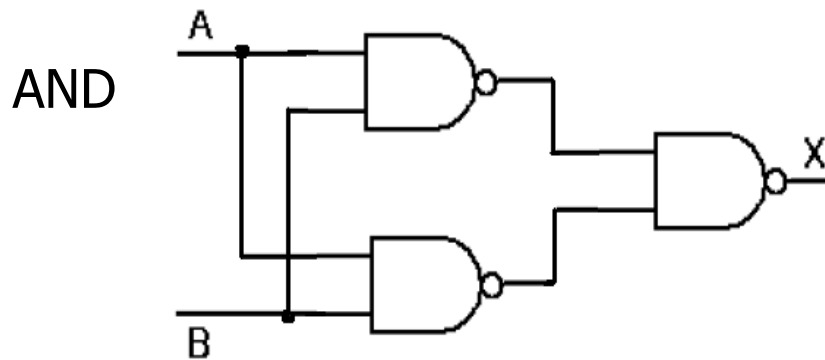
| A | B | X |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Universal Gates [Exercise]

Exercise:

prove the above by implementing AND, OR and NOT:

1. using only NOR gates,
2. using only NAND gates.



XOR implementation [Exercise]

- How to construct eXclusive OR (**XOR**)?

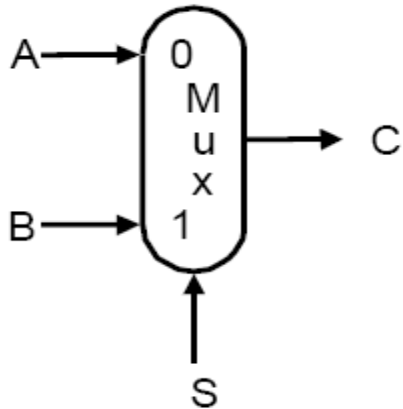


| X | Y | Z |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

| Ordinary Implementation | Universal Implementation |
|--|--------------------------|
| <p>A circuit diagram showing the ordinary implementation of an XOR gate. It consists of two AND gates and one OR gate. The top AND gate has inputs 'X' and 'y'. The bottom AND gate has inputs 'y' and 'x'. The outputs of both AND gates are connected to the inputs of the OR gate, which has an output 'Z'.</p> | ? |

Multiplexor implementation [Exercise]

- How to construct Multiplexor (**Mux**)?

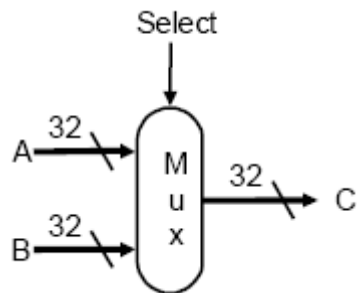


| S | c |
|---|---|
| 0 | a |
| 1 | b |

| Ordinary Implementation | Universal Implementation |
|-------------------------|--------------------------|
| | ? |

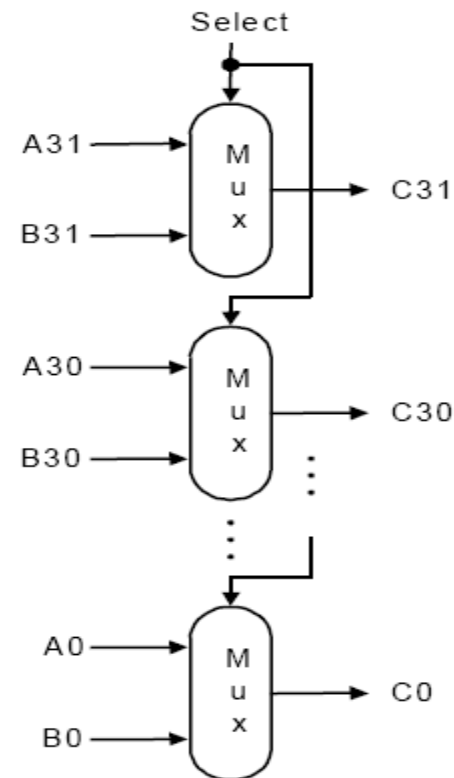
Arrays of logic elements

- Used when combinational operations need to be performed on entire word (32 bits)
 - for example when the result of an instruction that is written into a register can come from one of two sources



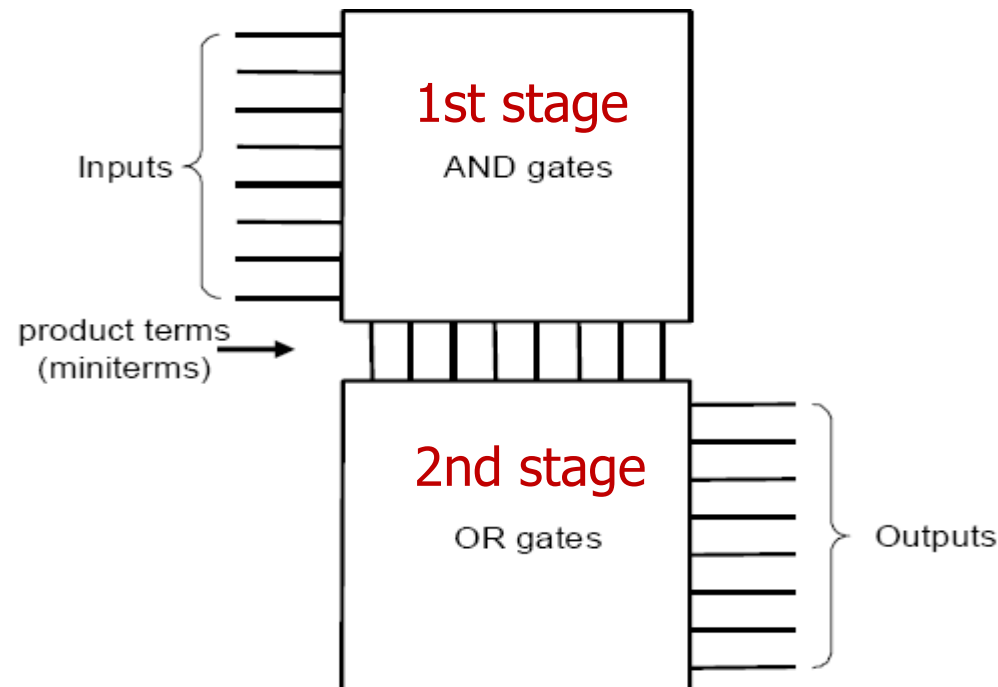
32-bit wide 2-to-1 multiplexor

32-bit wide multiplexor is an array of 32 1-bit multiplexors



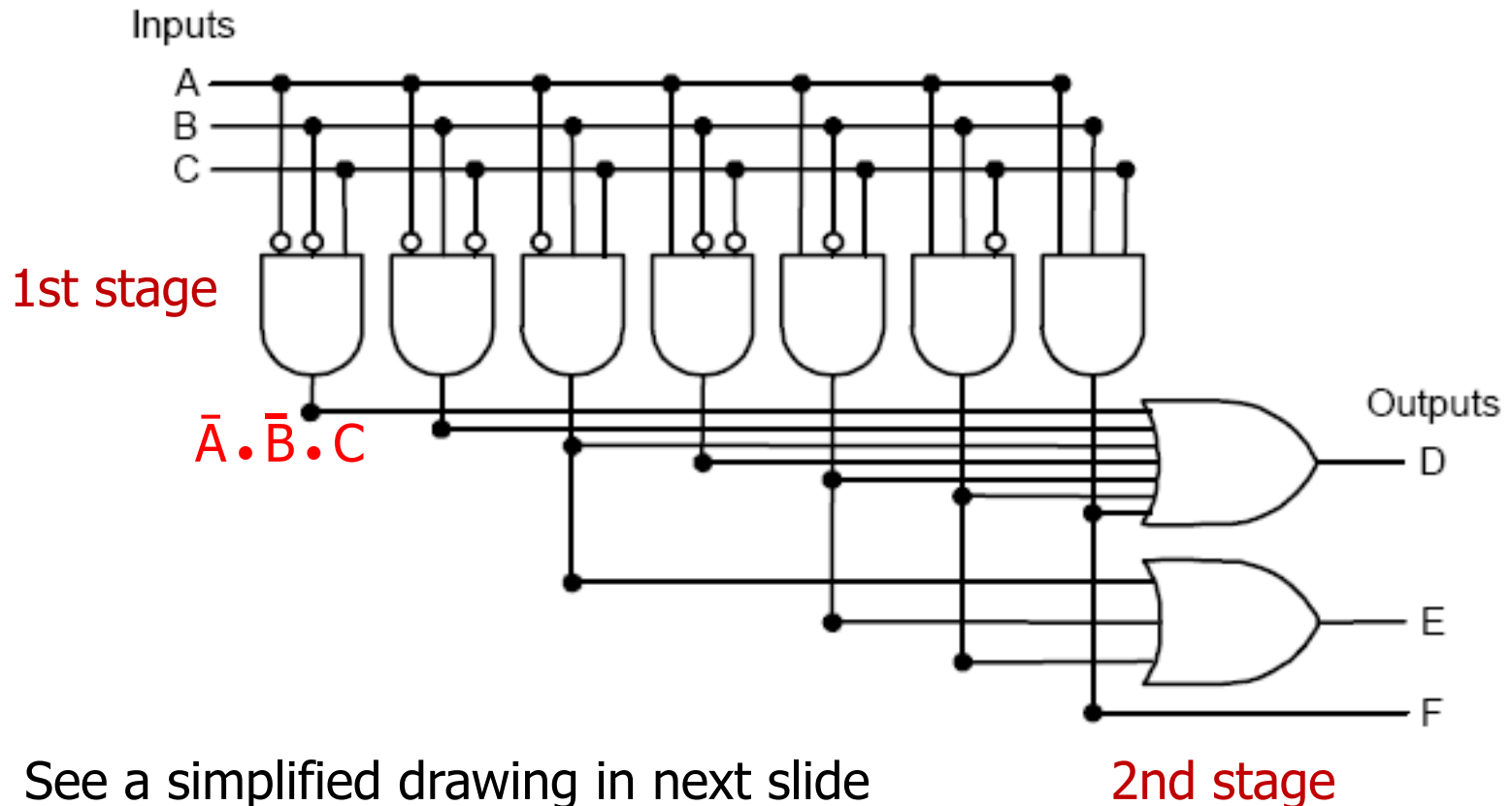
Programmable Logic Array (PLA)

- Two-level sum of products representation
 - the 1st stage: array of **AND** gates that forms a set of product terms (also known as *miniterms*),
 - the 2nd stage: array of **OR** gates each of which forms a logical sum of any number of the product terms.



Sample PLA 1/2

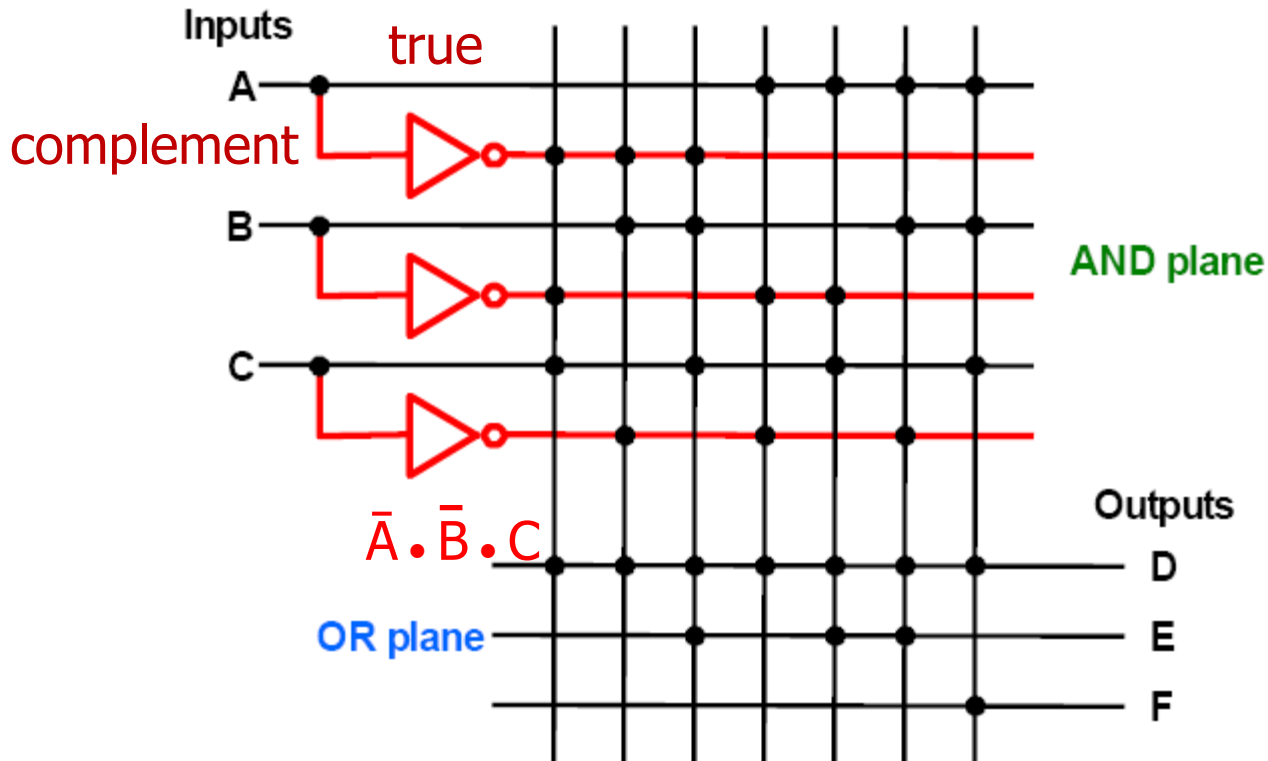
- Implements logic from page C-13 in the textbook (three inputs, three outputs)



See a simplified drawing in next slide

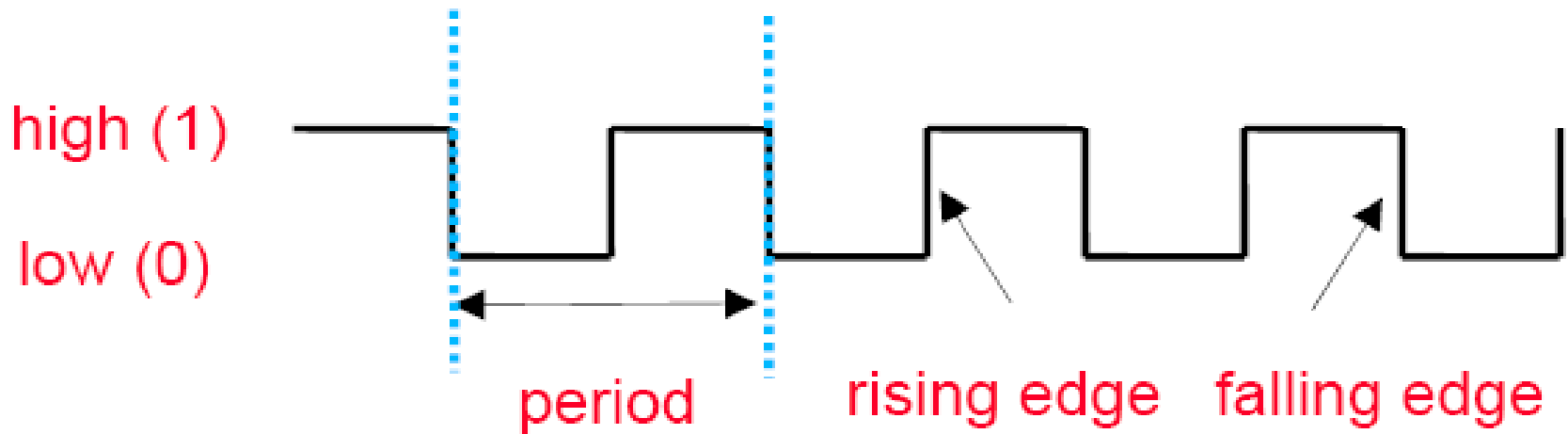
Sample PLA 2/2

- As before, simplified drawing shows AND and OR planes.
 - Note inputs A, B and C run the width of AND plane in both true and complement form.



Clock

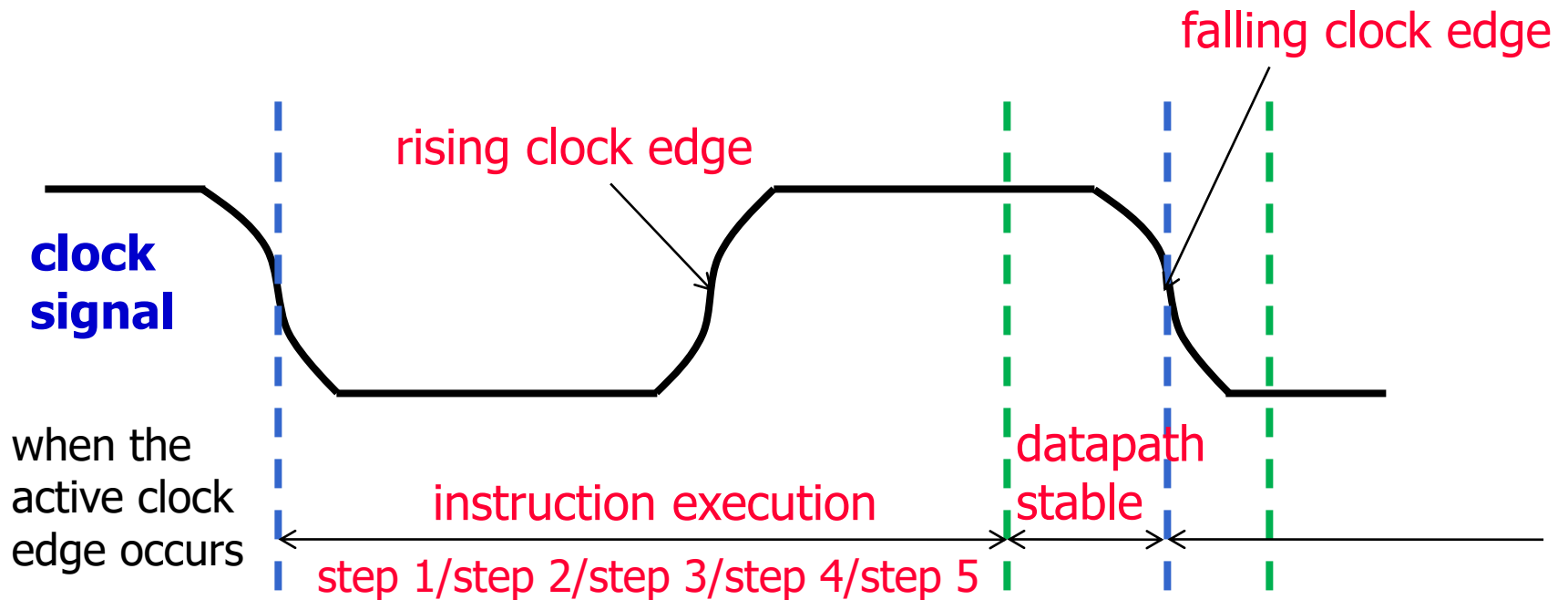
- Clock: free running signal with fixed cycle time (clock period)



- Clock determines when to write memory element
 - level-triggered – act (store) on clock high (or low)
 - edge-triggered – act (store) only on clock edge
- We will consider here only negative (falling) **edge-triggered** clocking methodology

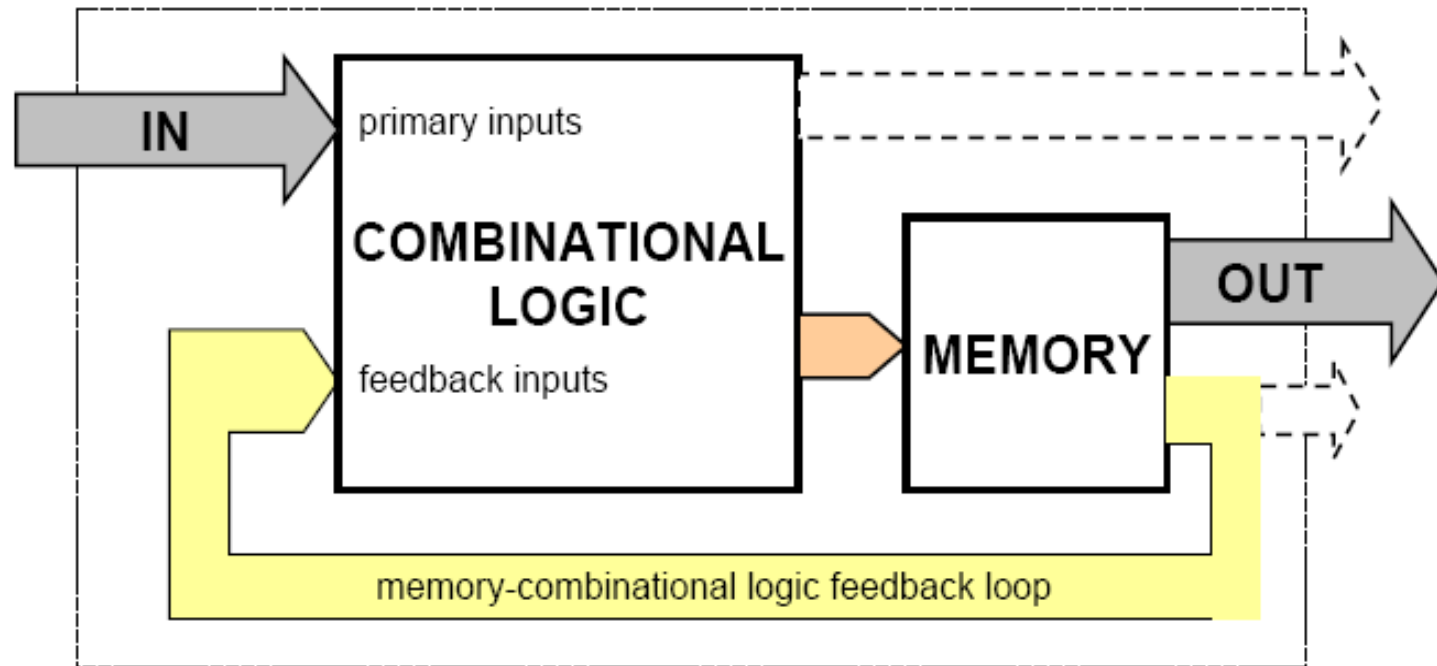
Clock in MIPS (5 steps of datapath)

- Synchronous (or clocked) combinational circuits
- Single-cycle machine: does everything in one clock cycle
 - instruction execution = up to 5 steps
 - must complete 5th step before cycle ends



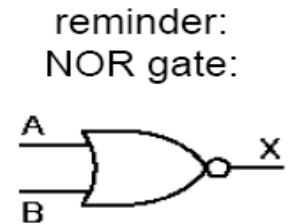
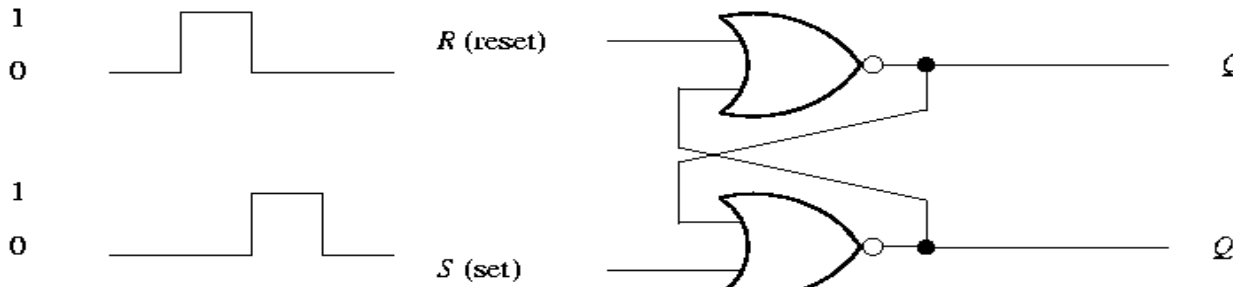
Sequential logic: memory elements

- All memory element store states:
 - The output from any memory element depends BOTH on the inputs and on the value stored inside the memory element.
- The simplest memory elements are unlocked (see S-R latch next).

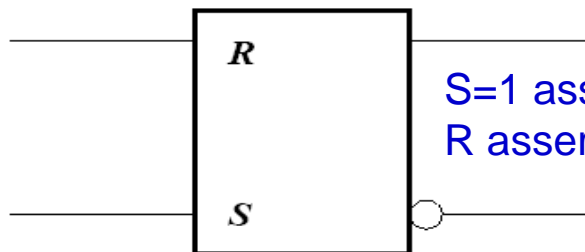


SR (set-reset) Latch, unlocked

- SR-latch implemented with **NOR** Gates
- Output depends on both inputs and values stored (previous state)
 - $S = 1$ and $R = 1$ not allowed
 - $R=S=0$ (removal of the input combination of 0's and 1's), output will not change (NC) -- depends on the values in previous state
 - Otherwise output copying input (set or reset action)



| S | R | Q | Q' |
|---|---|----|----|
| 0 | 0 | NC | NC |
| 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 |

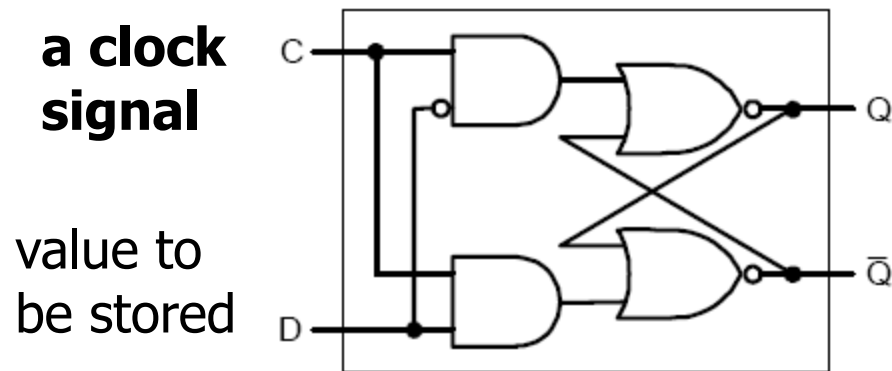


$S=1$ asserted, set $Q=1$ asserted;
 R asserted, Q deasserted.

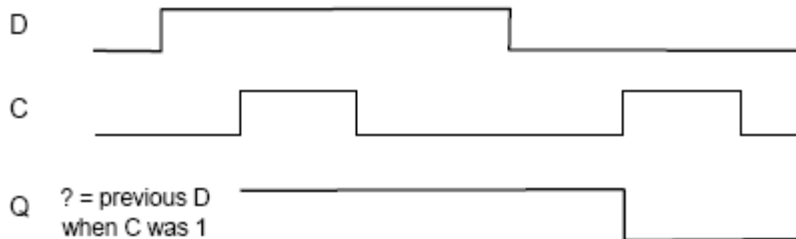
| A | B | X |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

D Latch (clocked), or: Delay Flip-Flop

- When the clock C is asserted, the latch is opened, and the output Q immediately assumes the value of the D input
- Sometimes called a transparent latch (when the latch opened, Q changes as D changes)



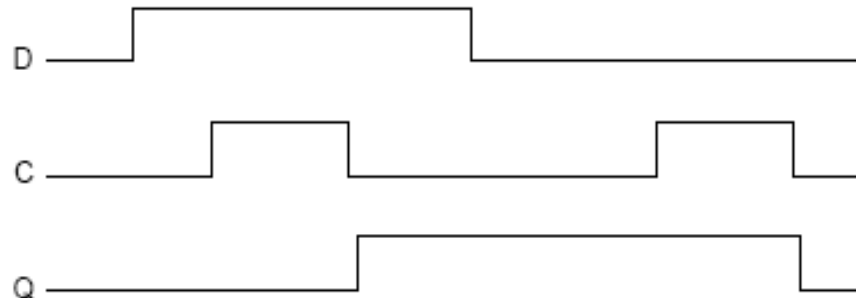
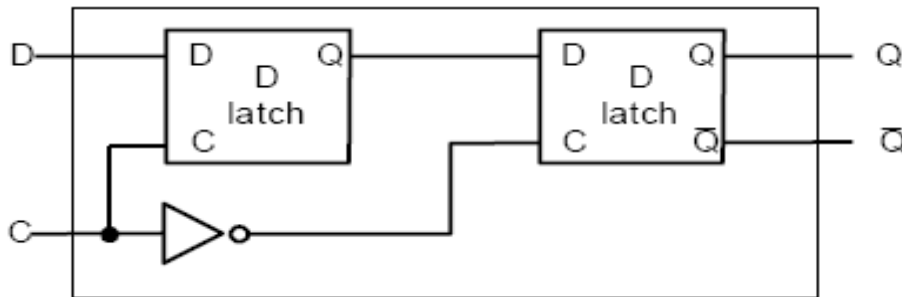
FYI - Electronics Demonstrations:
“Clocked SR Flip-Flop” at
<https://www.falstad.com/circuit/e-clockedsrff.html>



C controls the behavior of **Q**
attempting to learn from **D**

D flip-flop with a falling-edge trigger

- Flip-flop: state changes only on a clock edge
 - Master: the first latch, when the clock C is asserted Q follows D
 - Slave: the second latch, when the clock C falls gets its input from the output of the master latch.



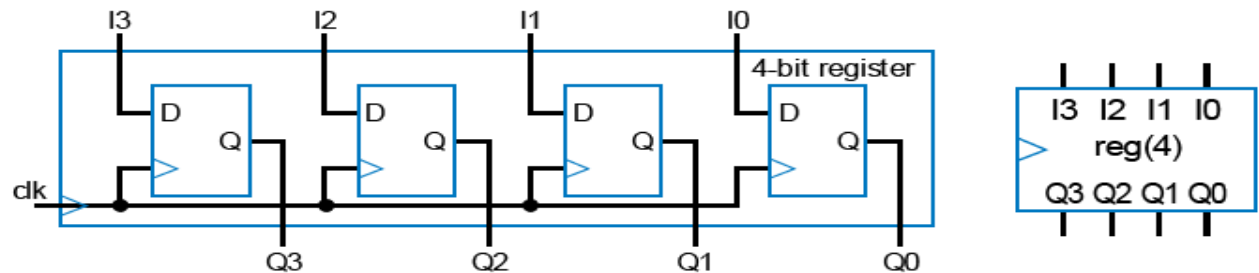
FYI - Electronics Demonstrations:
"Master-Slave Flip-Flop" at
<https://www.falstad.com/circuit/e-masterslaveff.html>

C controls the behavior of **Q**
attempting to learn from **D**

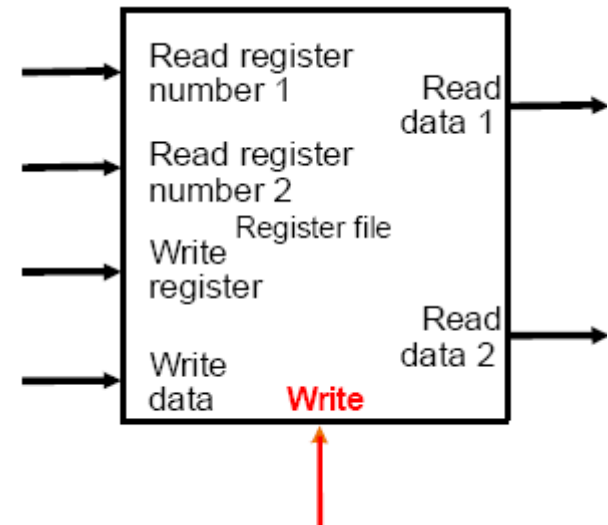
Register File Implementation

Appendix C, C-54

- **Register:** multiple flip-flops forming a single entity with the same clock signal

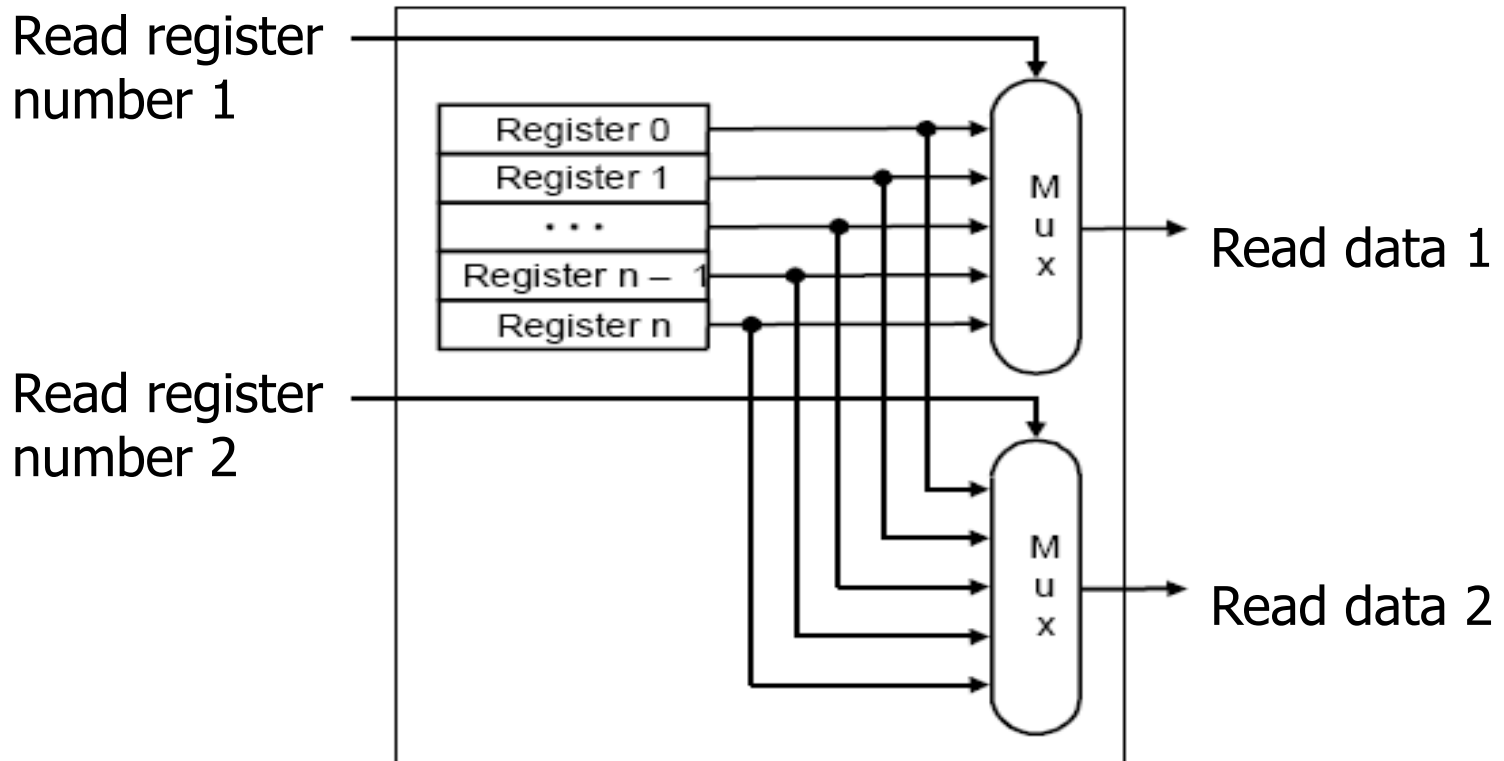


- A register file with two read ports and one write port.
- Can be implemented with a decoder for each read or write port and an array of D flip-flops used as registers.



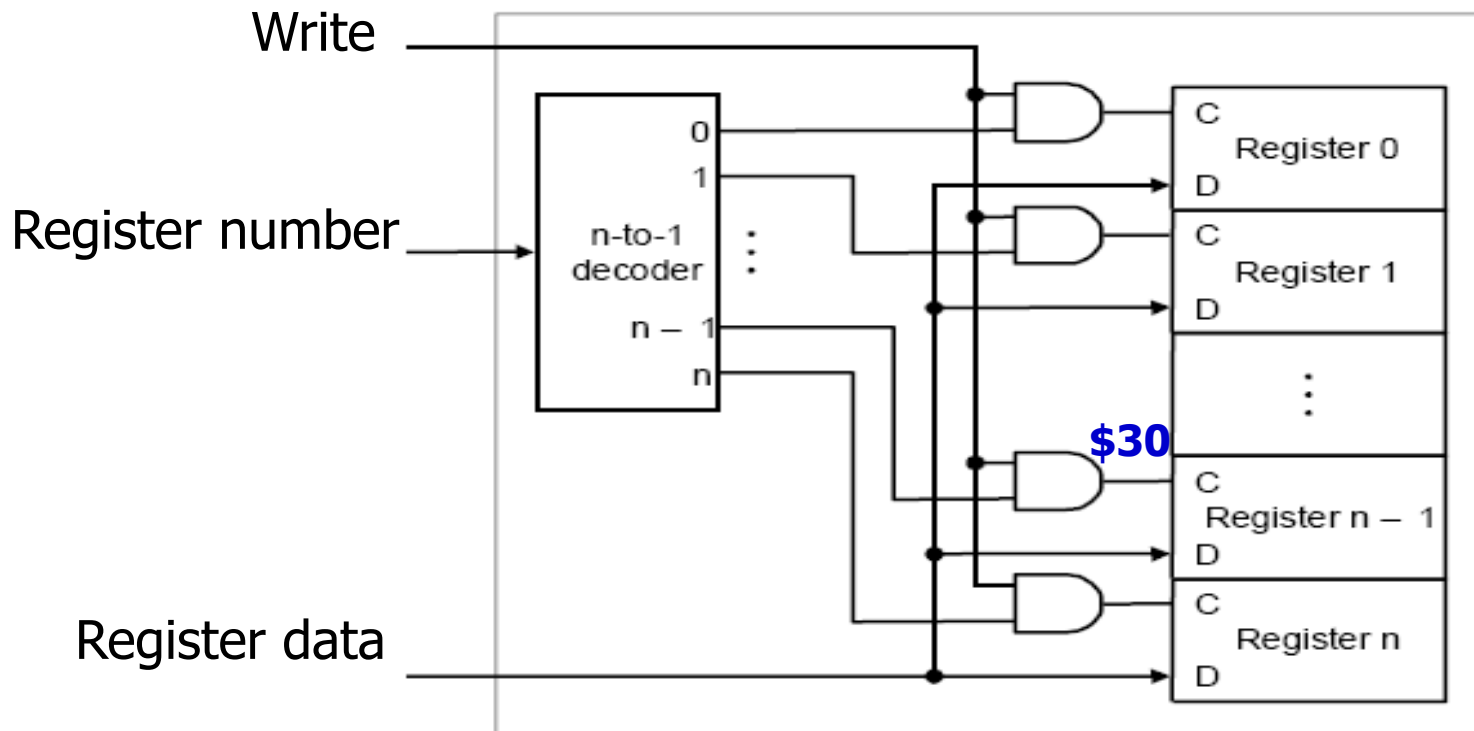
Register File Implementation 1/2 (read)

- Implementation of two register read ports for a 32-bit wide register file
- 'read register' signal used as the multiplexor selector signal.



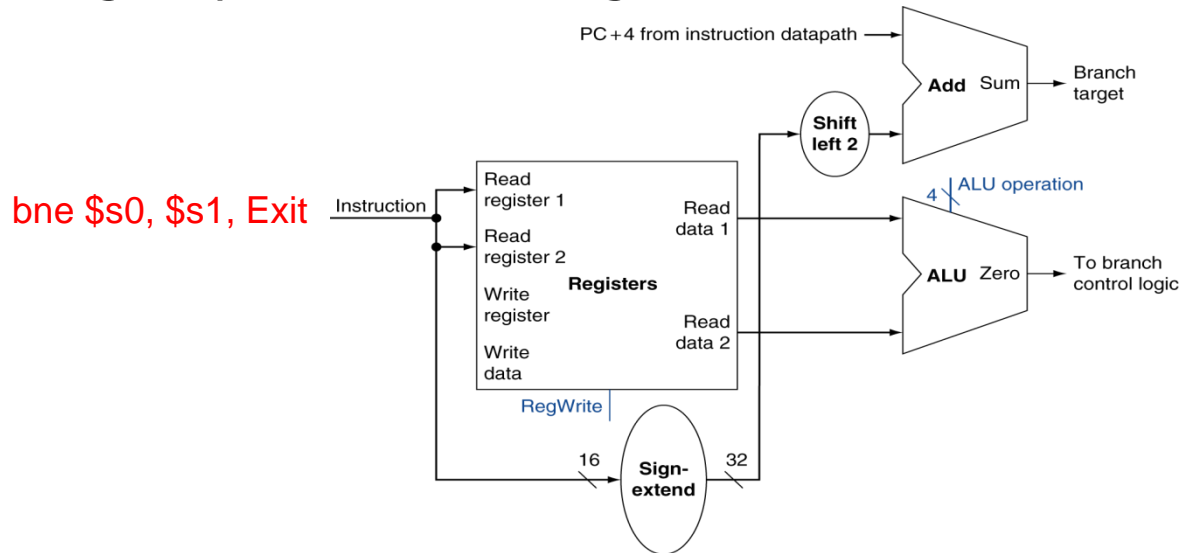
Register File Implementation 2/2 (write)

- Decoder used together with the write signal to determine which register to write.
- All three inputs will have set-up and hold-time constraints.

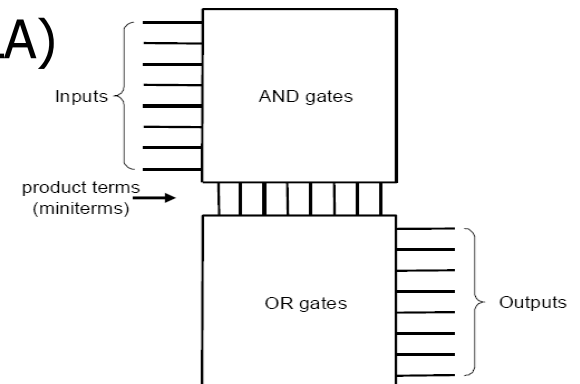


Revision


- The following block-diagram demonstrates the branch instruction processing. Explain what do 'Sign-extend' and 'Shift left 2' perform.



- The two-stage Programmable Logic Array (PLA) models its output as "a sum of products". What does it mean?



Recommended readings

| | |
|------------------------|--|
| General Data | UnitOutline LearningGuide Teaching Schedule Aligning Assessments  |
| Extra Materials | ascii_chart.pdf bias_representation.pdf HP_AppA.pdf instruction_decoding.pdf masking_help.pdf PCSpim.pdf PCSpim Portable Version Library materials |

PH6: Appendix B: The Basics of Logic Design
PH5: Appendix B: The Basics of Logic Design
PH4: Appendix C: The Basics of Logic Design

Text readings are listed in Teaching Schedule and Learning Guide

PH6 (PH5 & PH4 also suitable): check whether eBook available on library site

PH6: companion materials (e.g. online sections for further readings)

<https://www.elsevier.com/books-and-journals/book-companion/9780128201091>

PH5: companion materials (e.g. online sections for further readings)
<http://booksite.elsevier.com/9780124077263/?ISBN=9780124077263>