NEC 304

STLD

Lecture 34

Datapath Analysis

Rajeev Pandey

Department Of ECE

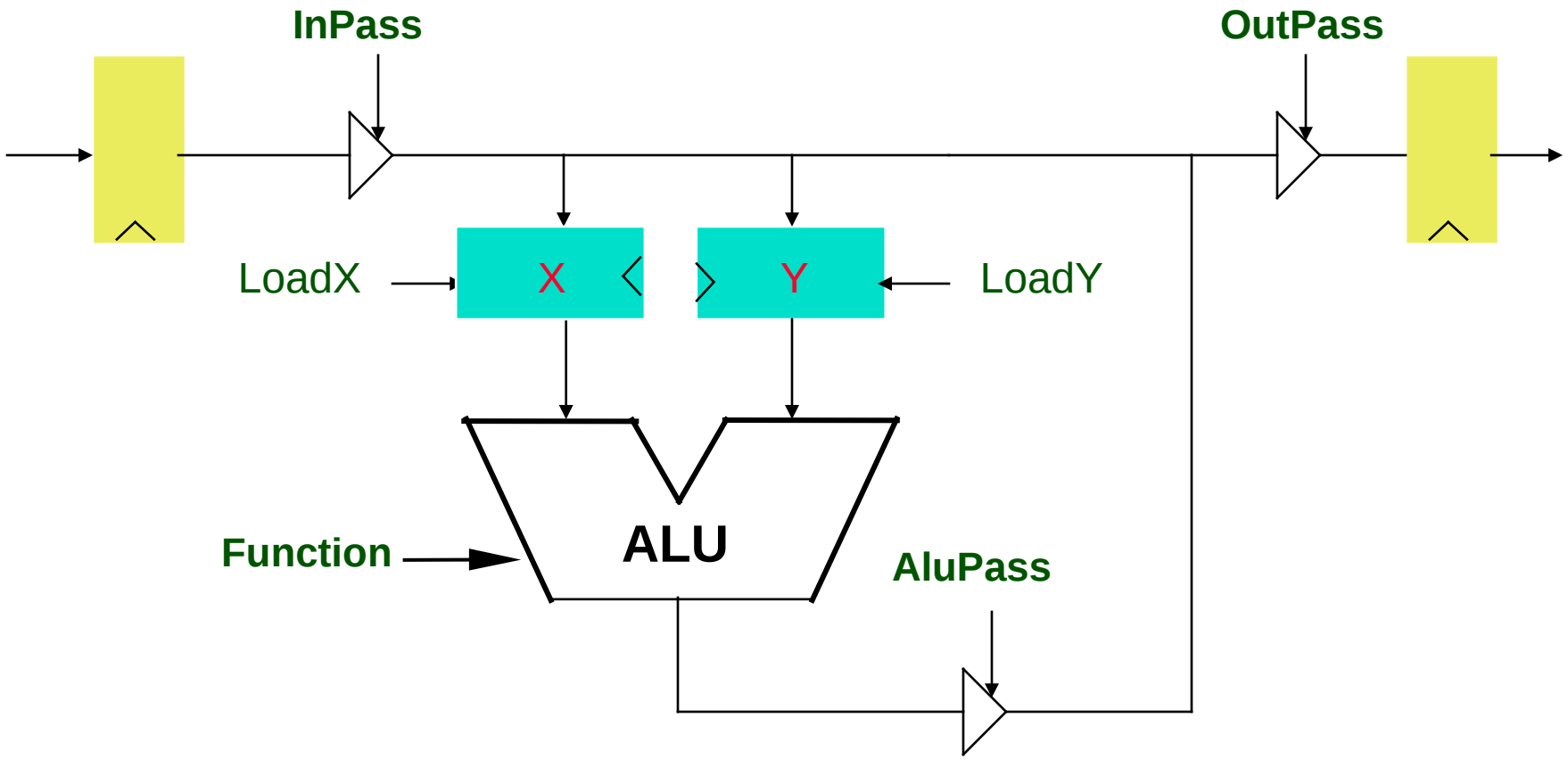
rajeevvce2007@gmail.com

Overview

- Datapaths must deal with input and output data values
  - Implement with tri-state buffers
- Necessary to control external interfaces
  - Perform repetitive operations
- Some datapaths require decision making
  - Control outputs implemented in ROM
- Moving towards software
  - Control implemented as a series of instructions
- Understanding the data and control path

Datapath I/O

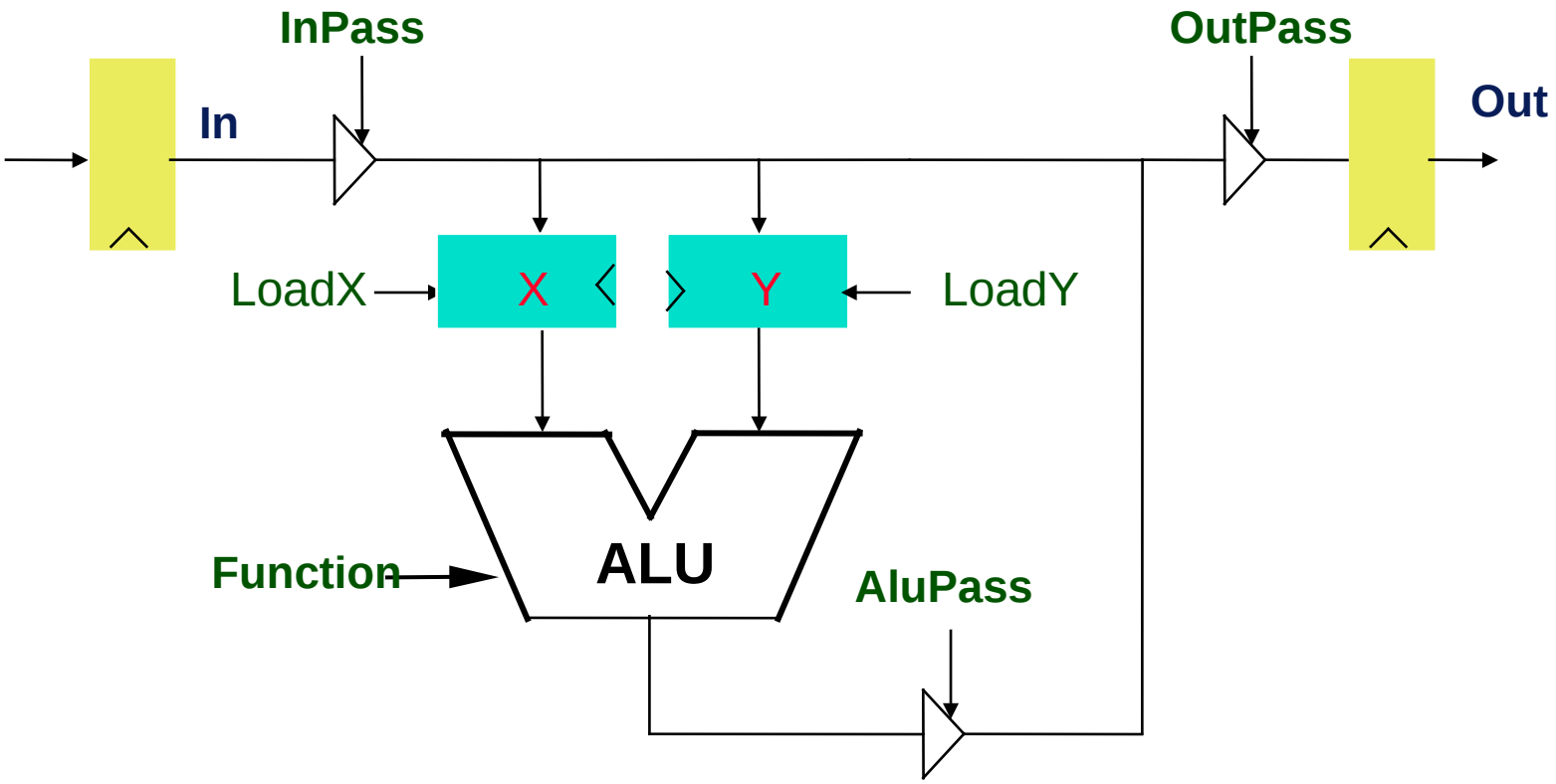
- **A wire can be driven by only one tri-state at a time**
  - **If InPass is active, AluPass must be inactive**
  - **If AluPass is active, InPass must be inactive**



Datapath I/O

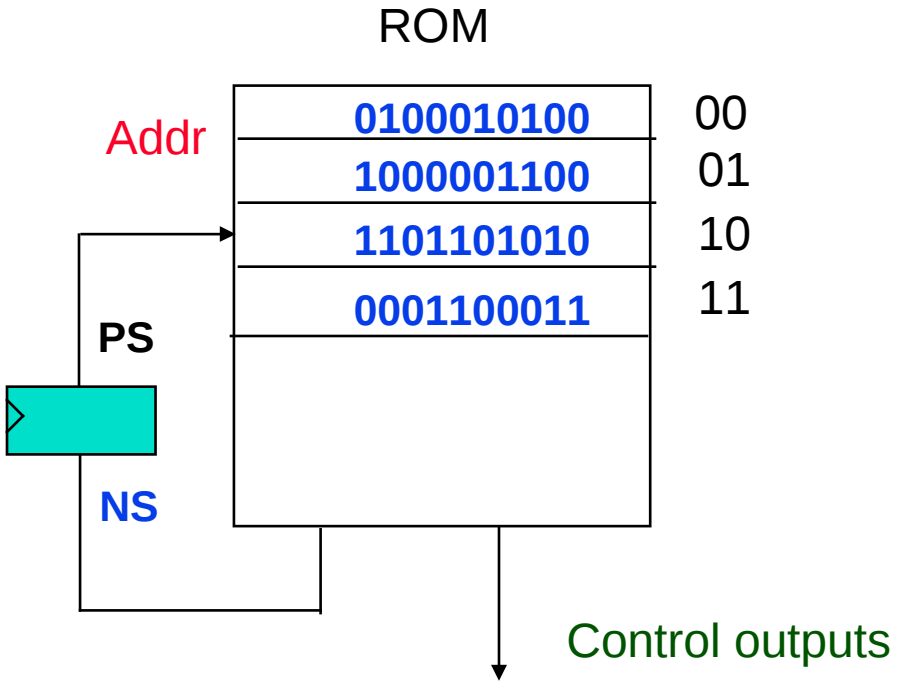
- **Two values enter from the left (A and B)**
  - **Need to perform (A+B)+A**
  - **In -> X (Load A)**
  - **In -> Y (Load B)**
  - **A+B -> Y**
  - **(A+B)+A -> Out**

Four steps and then repeat



Implementing the Control ROM

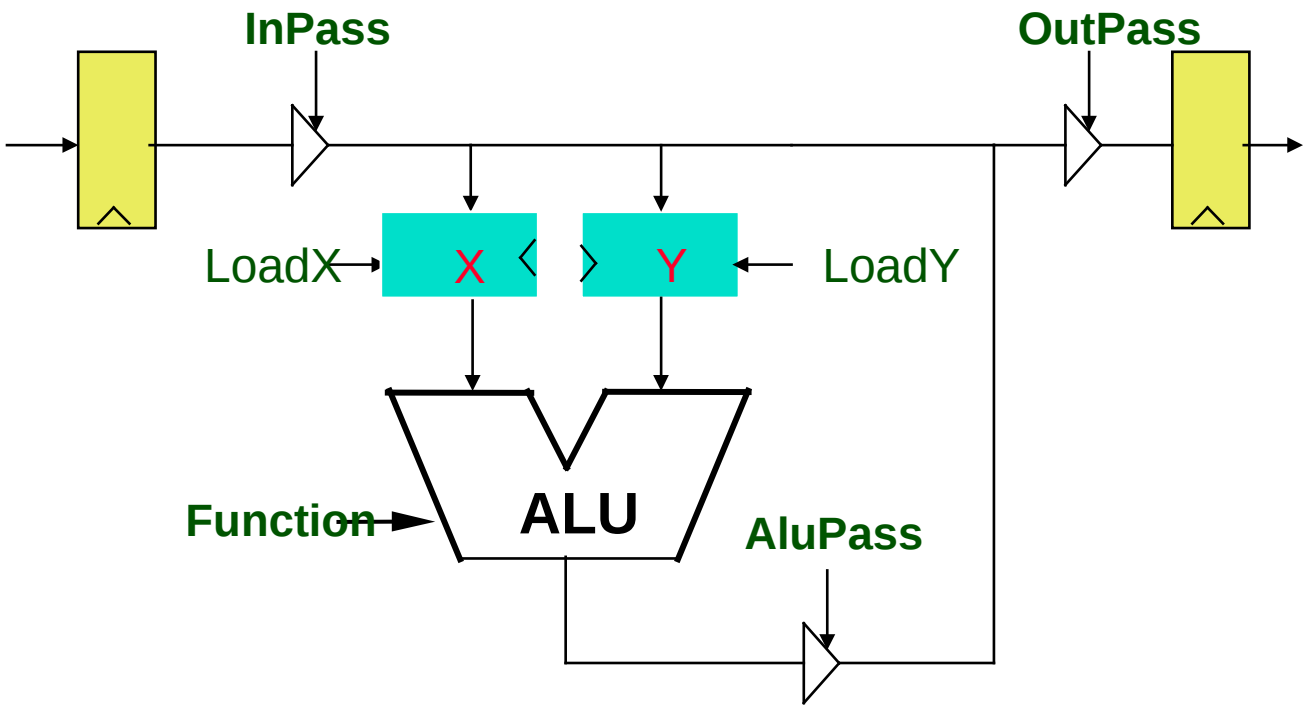
- **Two values enter from the left (A and B)**
  - **Need to perform (A+B)+A**
  - **In -> X (Load A) - State 00**
  - **In -> Y (Load B) - State 01**
  - **A+B -> Y - State 10**
  - **(A+B)+A -> Out - State 11**



PS	NS	Function	LoadX	LoadY	InPass	AluPass	OutPass
00	01	000	1	0	1	0	0
01	10	000	0	1	1	0	0
10	11	011	0	1	0	1	0
11	00	011	0	0	0	1	1

More Complicated Example

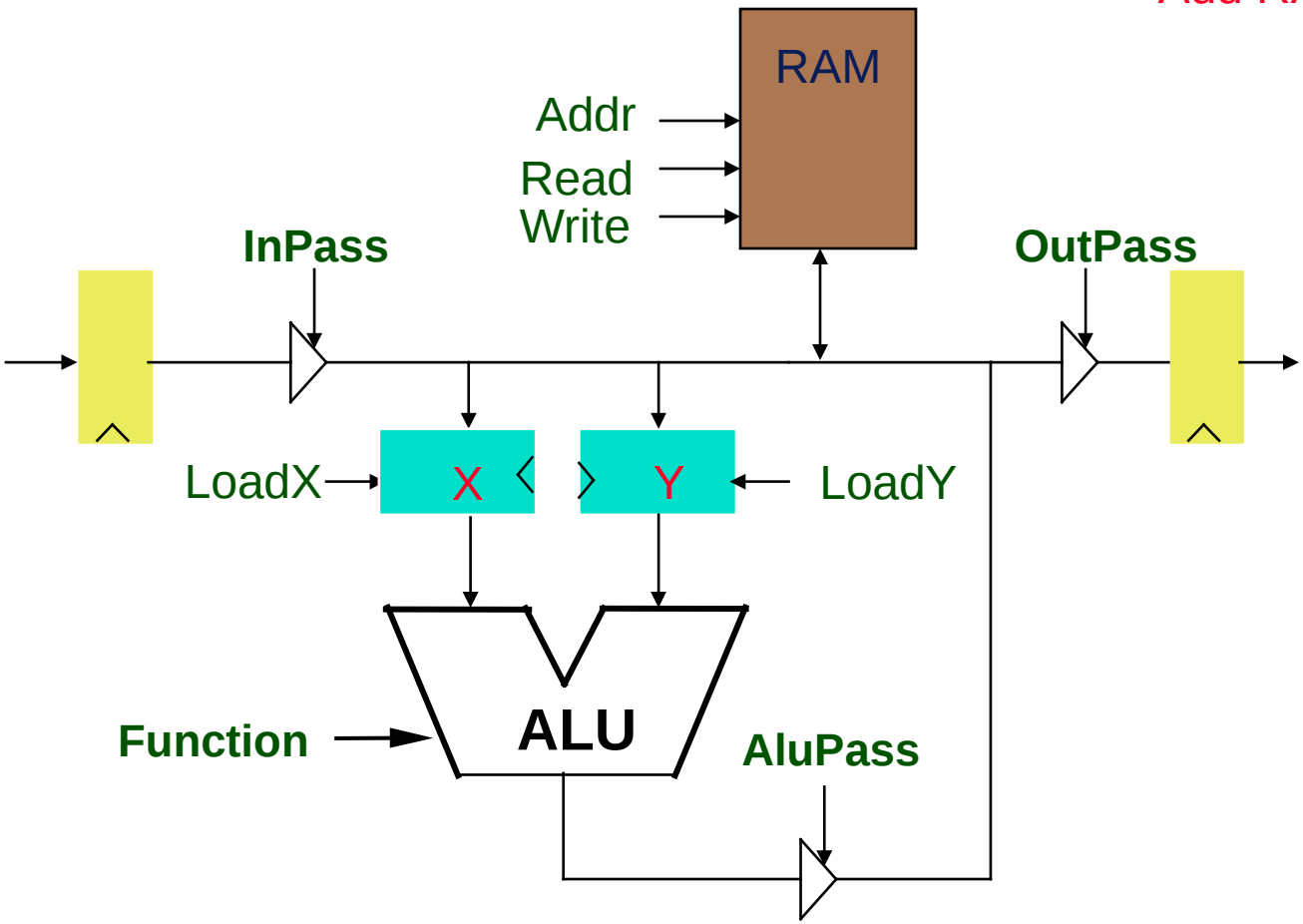
- Can we compute (A+B) . (A-B)?
- Currently, no place for intermediate storage
- Solution: Add RAM to datapath.



More Complicated Example

- **Can we compute (A+B) . (A-B)?**
  - **Need to add intermediate storage.**
- **Typical sizes (1MB – 2GB)**

Add RAM to the Datapath



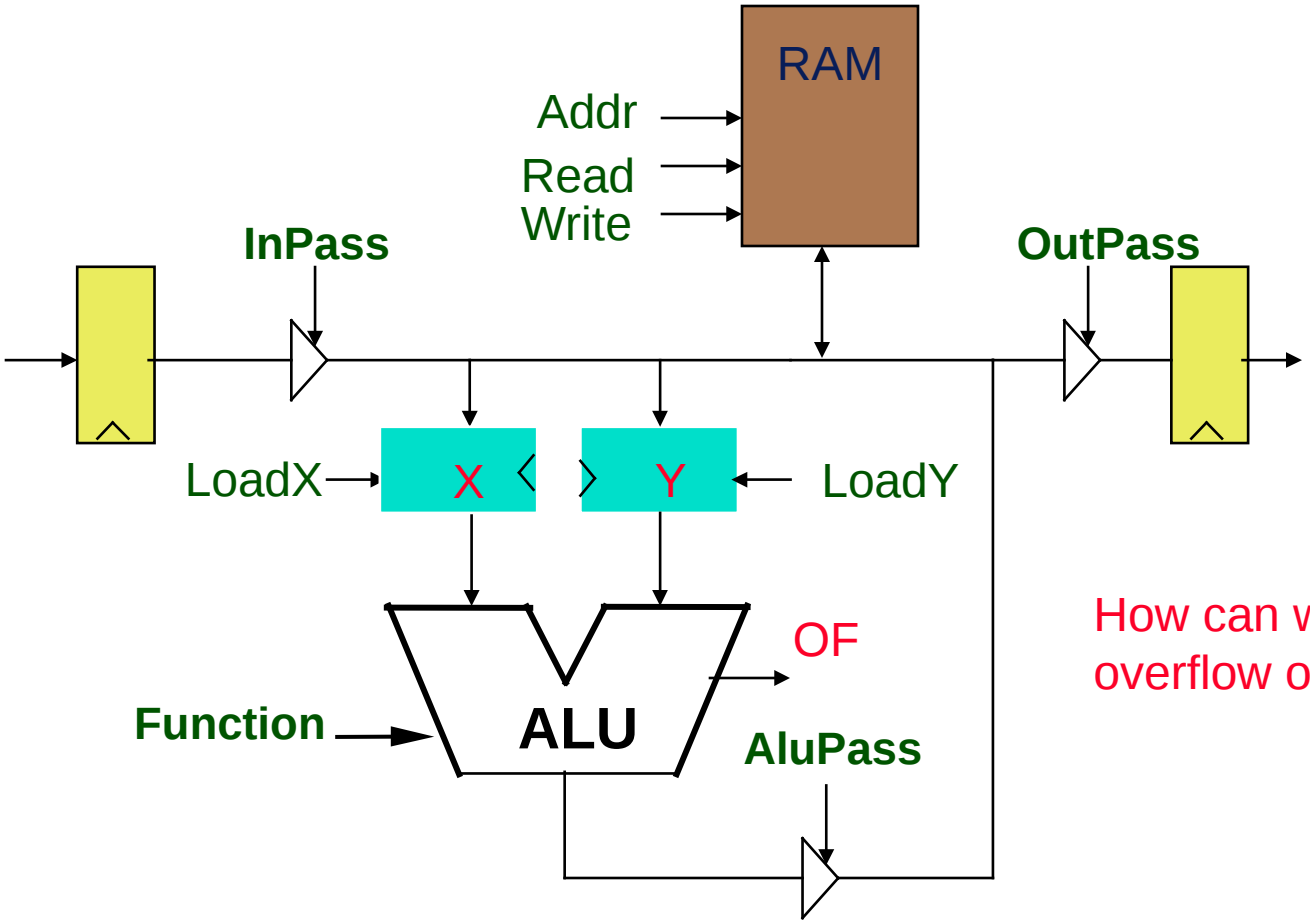
Implementing the Control ROM

- **Two values enter from the left (A and B)**
  - Need to perform (A+B) . (A-B)
  - In -> X (Load A) - State 000
  - In -> Y (Load B) - State 001
  - A+B -> RAM[4] - State 010
  - A-B -> X - State 011
  - RAM[4] ->Y - State 100
  - (A+B) . (A-B) ->Out - State 101

PS	NS	Function	LoadX	LoadY	InPass	AluPass	OutPass	Addr	Read	Write
000	001	000	1	0	1	0	0	000	0	0
001	010	000	0	1	1	0	0	000	0	0
010	011	011	0	0	0	1	0	100	0	1
011	100	010	1	0	0	1	0	000	0	0
100	101	000	0	1	0	0	0	100	1	0
101	000	110	0	0	0	1	1	000	0	0

Does the Value of the Data Matter?

- **Problem: Add A to itself until overflow occurs**
  - **Amount of steps depends on A**



How can we determine if overflow occurred?

Implementing the Control ROM using Conditions

- **One value enters from the left**

Add A to itself until overflow occurs

- **In -> X, Y (Load A, B) - State 0 - Next state 1**
- **X+Y -> Out, X - State 1 - Next state (1 if no overflow, 0 if overflow)**

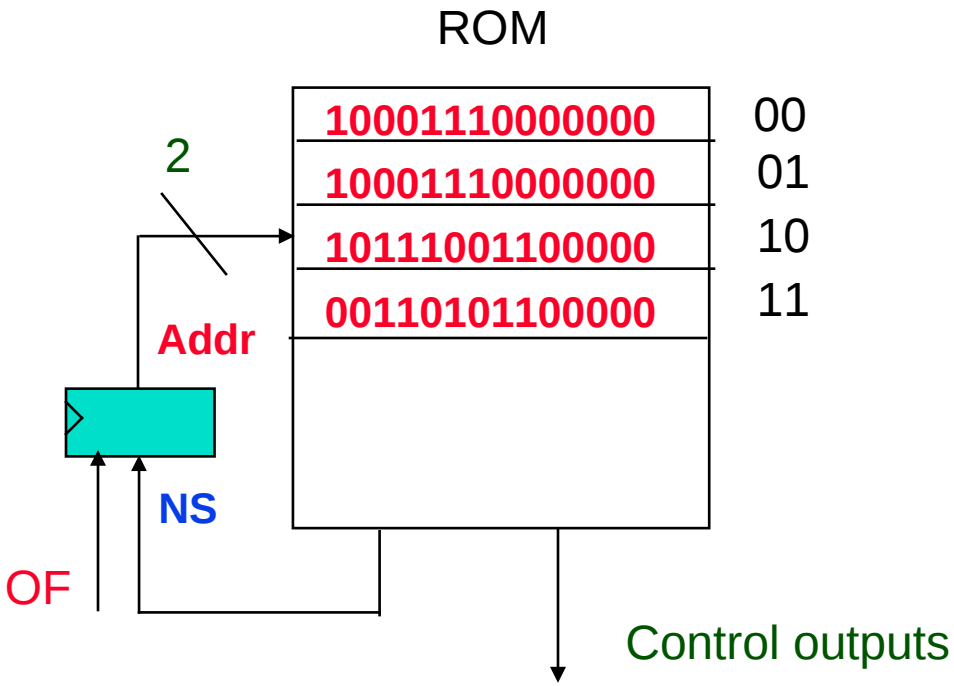
Include overflow (OF) bit as a ROM input
Note that it doubles the size of the ROM

PS	OF	NS	Function	LoadX	LoadY	InPass	AluPass	OutPass	Addr	Read	Write
0	0	1	000	1	1	1	0	0	000	0	0
0	1	1	000	1	1	1	0	0	000	0	0
1	0	1	011	1	0	0	1	1	000	0	0
1	1	0	011	1	0	0	1	1	000	0	0

Bits in the ROM
Each row indicates a ROM word

Implementing the Control ROM with Conditionals

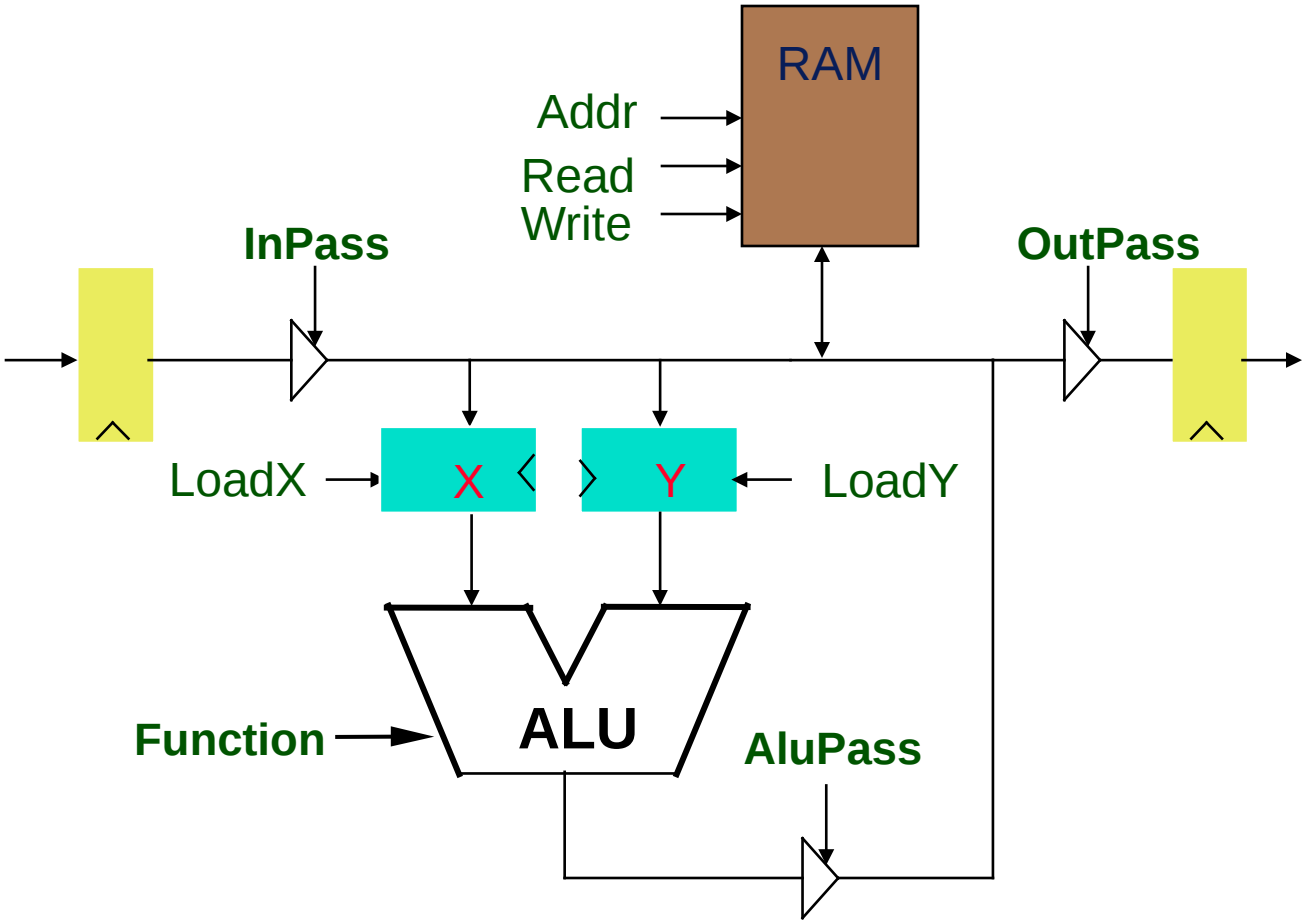
- **Control path may have many inputs**
  - **Overflow, carry out, zero**
- **Used to perform conditional operations**
- **If statements and loops in programming languages**



PS	OF	NS	Function	LoadX	LoadY	InPass	AluPass	OutPass	Addr	Read	Write
0	0	1	000	1	1	1	0	0	000	0	0
0	1	1	000	1	1	1	0	0	000	0	0
1	0	1	011	1	0	0	1	1	000	0	0
1	1	0	011	1	0	0	1	1	000	0	0

One More Example

- **Read two values from RAM (locations 0 and 1) and store to location 2.**
  - **Very common operation for microprocessor**



Implementing the Control ROM

- **Perform memory reads and writes**
  - **RAM[0] -> X - State 00**
  - **RAM[1] -> Y - State 01**
  - **X+Y -> RAM[2] - State 10**

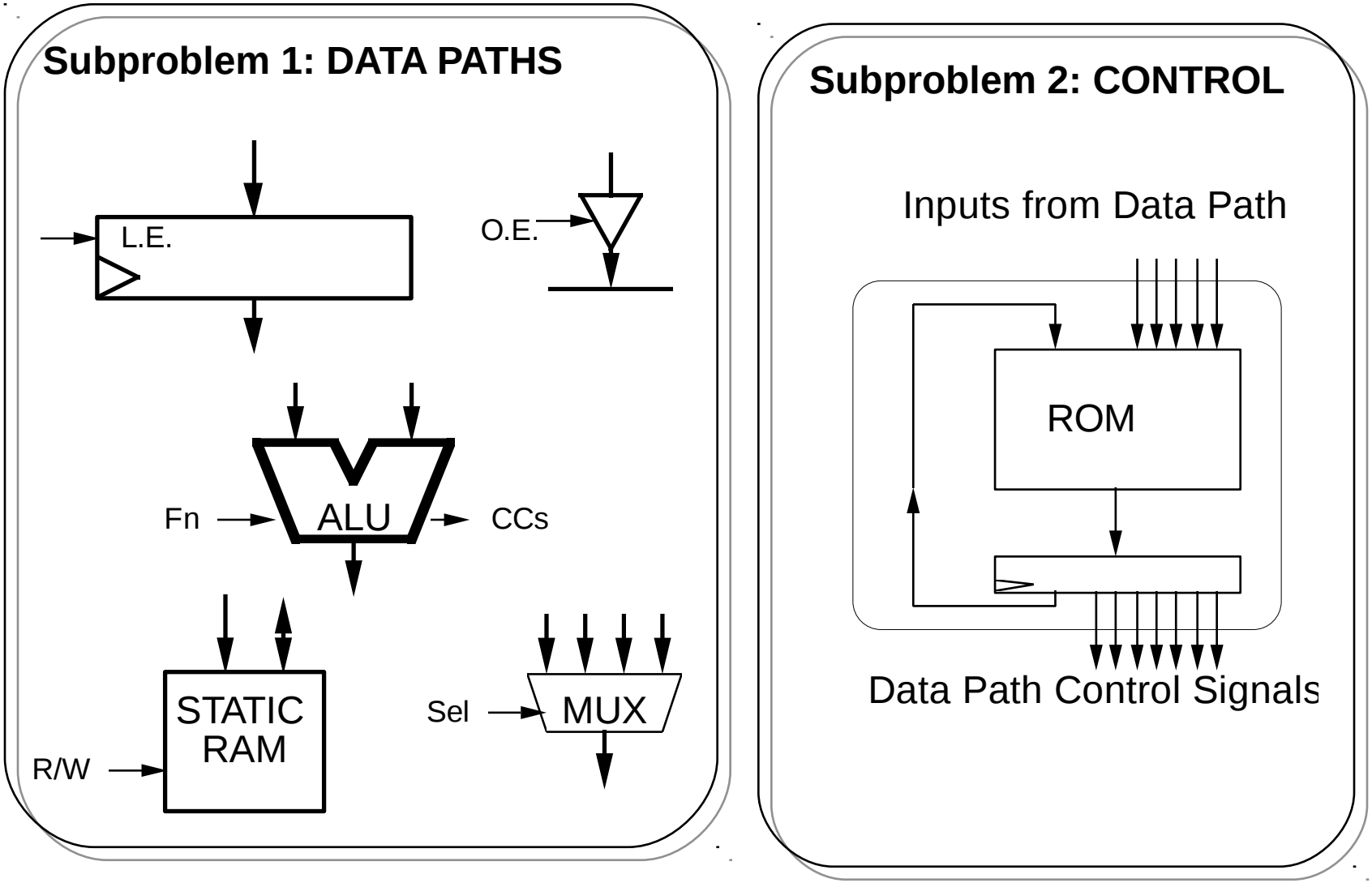
No interaction with outside interfaces (In, Out) is required

Very similar to microprocessor operations

PS	NS	Function	LoadX	LoadY	InPass	AluPass	OutPass	Addr	Read	Write
00	01	000	1	0	0	0	0	000	1	0
01	10	000	0	1	0	0	0	001	1	0
10	00	011	0	0	0	1	0	010	0	1

This slide marks the end of required material for this lecture!

Processor Construction Kit



Processor Compilation

- **Software engineer writes C program**
- **Compiler software converts C to assembly code**
- **Assembler converts assembly code to binary format**

C program

```
main () {
int A, B, C;

C = A + B;
}
```

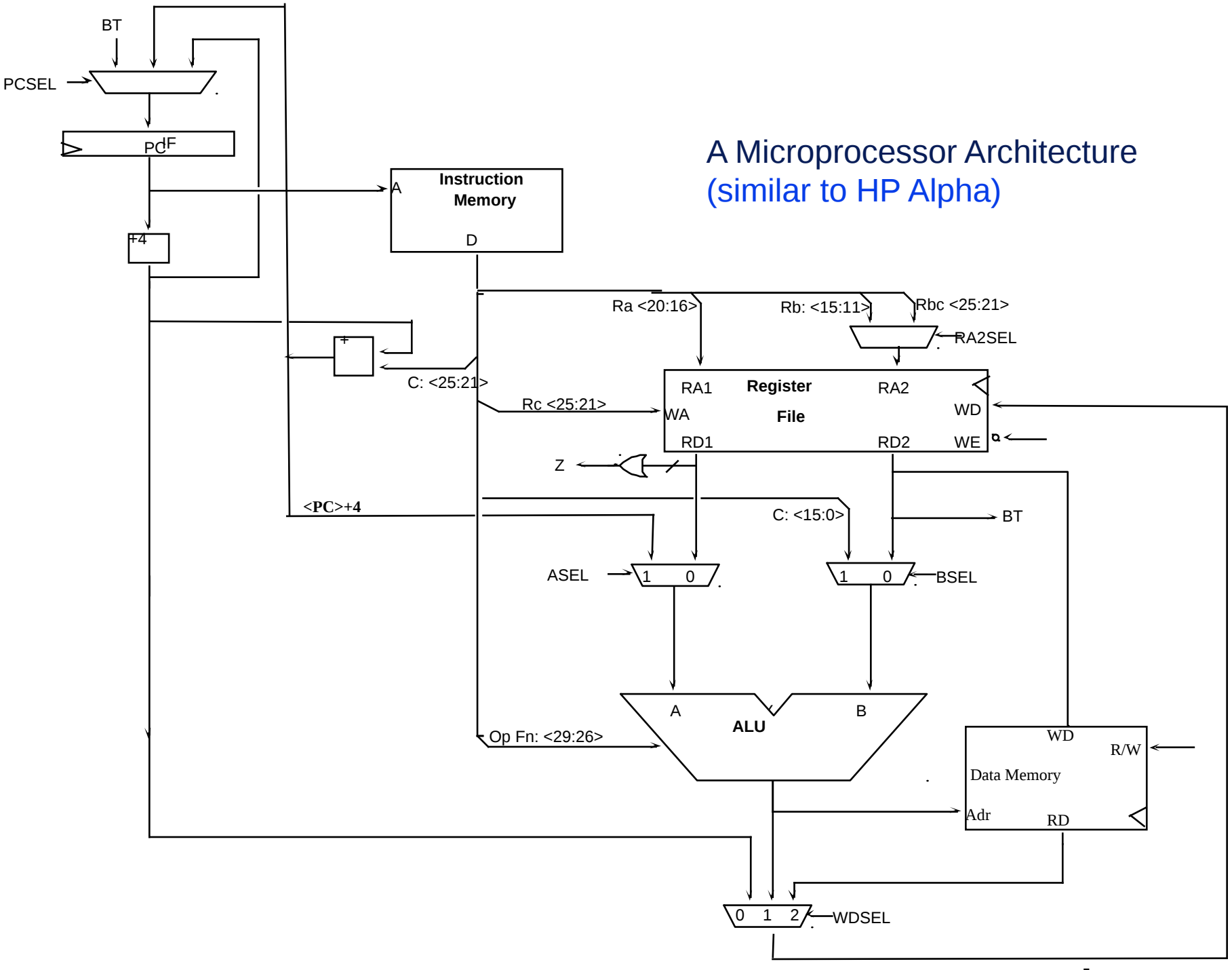
Compile

Assembly program

```
LD  R1, A          ; load A to Reg R1
LD  R2, B          ; load B to Reg R2
ADD R3, R1, R2     ; Add R1, R2 -> R3
ST R3, C           ; Store result in C
```

A, B, and C are storage locations in main memory (DRAM)

A Microprocessor Architecture (similar to HP Alpha)



Summary

- Datapaths are important components of computer systems
- Interaction between control and data path determines execution time
- Each sequence of operations can be represented with a ROM program
  - Each row in the state table corresponds to a word in the ROM
- Multiple rows for each state if the ROM has a control input (e.g. ALU overflow)
- Next time: Notation to represent the data and control paths