Lab 2: Linked List to Array Conversion in RISC-V

Saumya Patel

UNIVERSITY OF ALBERTA
Department of Computing Science

July 31, 2025

Contents

Introduction

In this lab, you will develop a RISC-V assembly subroutine that:

- Converts a singly linked list into an array.
- Locates the index of a specified target element during the conversion.

This exercise strengthens your understanding of data structures, memory management, and assembly programming.

What is a Linked List?

- A linear data structure where each element (node) is dynamically allocated and stored in non-contiguous memory.
- Each node contains:
  - **Data**: The value stored in the node.
  - **Pointer**: The address of the next node.
- The last node uses a sentinel value (e.g., -0_) to indicate the end of the list.

Linked List: Definition and Use Cases

Definition

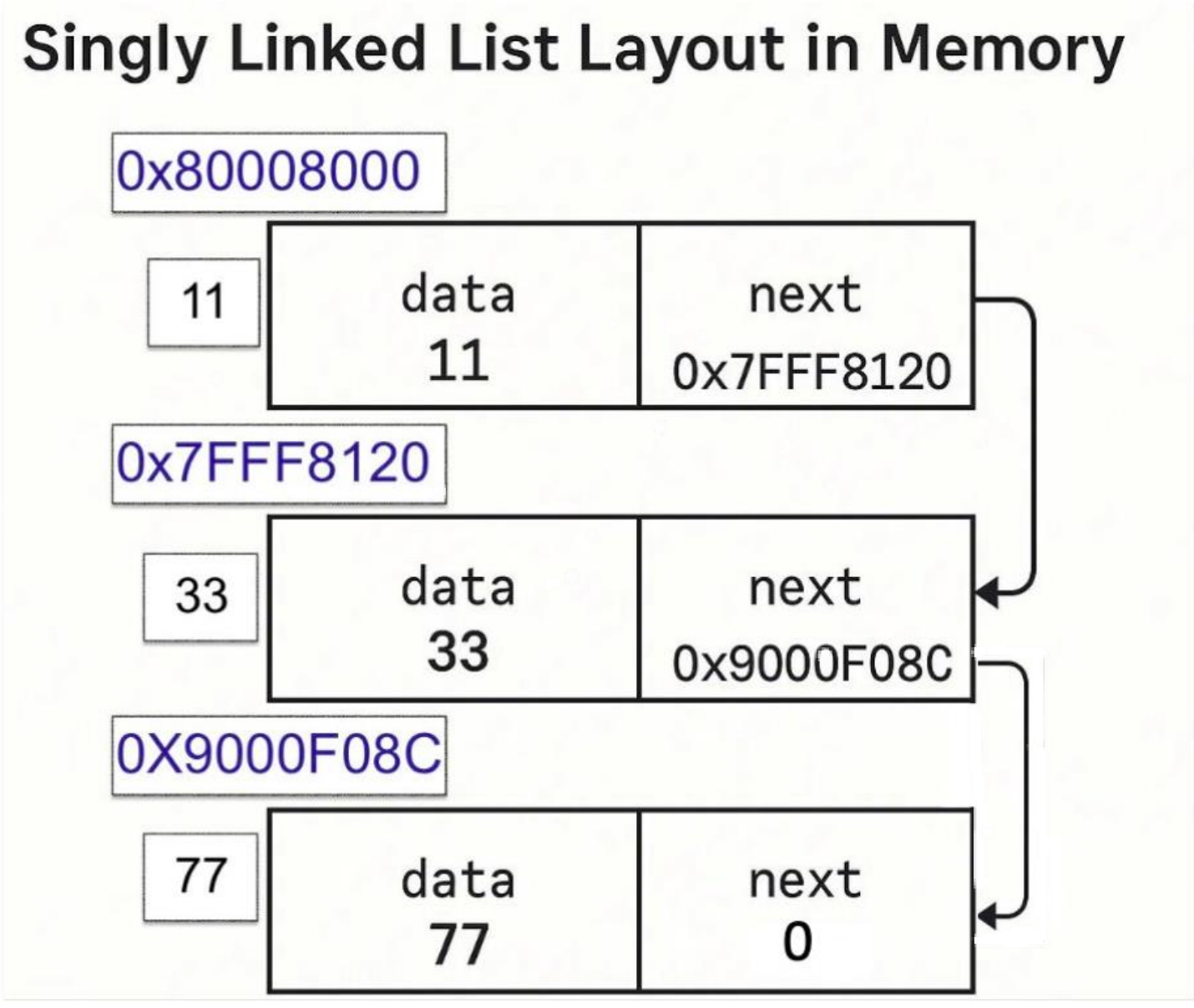
A linked list is a sequence of nodes where each node stores data and a reference to the next node.

- Preferred when:
  - The number of elements is not known in advance.
  - Frequent insertions and deletions are expected, especially in the middle or beginning.
- Arrays provide faster random access, but are less efficient for resizing or inserting elements.

Memory Layout of Linked Lists

- Nodes are **not** stored contiguously in memory.
- Each node is dynamically allocated using `malloc` in risc-v (dont worry about it, its handled in Common.s).
- Structure of a node:
- The final node's pointer is set to a sentinel value

Visual Representation



Lab Task Overview

- Implement a RISC-V subroutine `Convert` that:
  - Traverses a singly linked list.
  - Copies each node's data into an array.
  - Locates the last occurrence of a target value.
  - Stores all node data followed by the sentinel.

Detailed Requirements

- **Traverse** the linked list from the head node.
- **Store** each node's data in consecutive array positions.
- **Handle Sentinel**: When the pointer field equals the sentinel value `(0)`, stop traversal and store the sentinel in the next array element.
- **Find Target**: During traversal, compare each node's data with the target value. If found, record its index.
- **Return**: If the target is not found, return `-1`.

Test Case Format

- Test cases are plain text files ending with `.txt` and must follow the format below:

```
L T
[sequence of integers representing the linked list]
```

- **Where:**
  - **L**: Length of the linked list.
  - **T**: Target value to find.

Example:

```
11 5
5 7 2 3 4 5 1 2 3 4 5
```

Input Guarantees

- The linked list contains between 0 and 50 elements.
- All data values are positive integers.
- The linked list is never empty.
- The target value is a positive integer.

Subroutine Specification

Convert Function

Arguments:

- a0: pointer to output array
- a1: pointer to head of linked list
- a2: pointer to target value

Return Values:

- a0: 1 if target found, otherwise 0
- a1: index of target element, or –1 (–1 if not found)

Effect: Converts the linked list to an array and locates the target element.

Marking Guide

- 20%: Code cleanliness, readability, and comments.
- 15%: Correctly indexing the next node.
- 15%: Correctly inserting elements into the array.
- 10%: Correctly finding the target element.
- 20%: Correct handling of the presence/absence of the target(target found/not found, moving correct values into the registers)
- 20%: Correctly moving values into specified registers.

Submission Instructions

- Submit only `Convert.s` (your subroutine implementation).
- Do **not** any labels that exist in the **Common.s** file
- Do **not** modify `.include "common.s"` or the common.s file.
- Ensure your code works on the lab machines.
- **Make sure to follow the correct program writing style and maintain code comments and cleanliness**

Additional Information

- All instructions for the lab are provided in the assignment.
- For more on screen updates and visualizations, see the lab PDF.
- Focus on correct traversal, data copying, and target search in your subroutine.

Conclusion

- Linked lists are flexible for dynamic data but require sequential access.
- Arrays allow fast random access but are less flexible for resizing.
- This lab builds skills in data structure manipulation and assembly programming.