alex-levesque.com

ELEC 274 — Computer Architecture

WINTER 2026

Based on lectures by N. Manjikian – Queen's University

Notes written by Alex Lévesque

Contents

1 Preface

Grading Scheme:

Textbook:

Comments:

-

2 Assembly Guide

In Nios II, every instruction is 32 bits wide. Since memory addresses are assigned to individual bytes (8 bits), we know that each instruction occupies a 4-byte block. Therefore, in a routine, the starting address of the next instruction must be exactly 4 bytes higher than the previous one.

In register transfer notation, use [...] to denote contents of a location. Use $\leftarrow$ to denote transfer to a destination. Example: $R2 \leftarrow [LOC]$. We can also do $R4 \leftarrow [R2] + [R3]$

2.1 Data Movement & Memory Access

These instructions are used to move data between registers or between registers and memory

`mov` (move register): `mov dest., src.`

- Definition: copies the contents of one register into another register
- Purpose: Used when you need to duplicate a value that is currently in a register to use it elsewhere without modifying the original

`movi` (move immediate): `mov dest., IMM16`

- Definition: Loads a 16-bit constant (immediate) value into a register. The value is sign-extended to 32 bits.
- Purpose: Used to initialize a register with a small number (like 0, 1, or small offsets)

`movia` (move immediate address): `mov dest., label/32-bit address`

- Definition: A macro (pseudo-instruction) that loads a full 32-bit value (typically a memory address) into a register
- Purpose: Since standard instructions can only handle 16-bit numbers at a time, movia is essential for loading the addresses of labels so the program knows where data is located in memory

`ldw` (load word): `ldw dest., byte_offset(base address)`

- Definition: Reads a 32-bit word from a specific memory address and loads it into a destination register
- Purpose: Used to retrieve data stored in RAM so the processor can operate on it

`stw` (store word): `stw source, byte_offset(base address`

- Definition: Writes the 32-bit value currently in a register to a specific memory address
- Purpose: Used to save the results of calculations back into RAM for later use

Byte offsets are used to provide a flexible and efficient way to access specific memory locations relative to a known starting point

2.2 Arithmetic Operations

These instructions perform mathematical calculations.

`add` (add): `add dest., src., src.`

- Definition: adds the contents of two source registers together and stores the result in a destination register
- Purpose: Used for standard addition of variables

`addi` (add immediate): `addi dest., src., IMM16`

- Definition: adds the value of a source and a 16-bit constant, storing the result in the destination
- Purpose: commonly used to increment counters or move points to the next item in a list

`subi` (subtract immediate): `subi dest., src., IMM16`

- Definition: Subtracts a 16-bit constant value from source and stores the result in the destination
- Purpose: used to decrement counters or adjust values downwards by a fixed amount

2.3 Control Flow (Branching)

These instructions change the order in which the code executes (loops and if-statements). Branch instruction causes repetition of body. Many branches are conditional, as seen below.

`br` (branch): `br LABEL`

- Definition: unconditionally jumps to the instruction located at LABEL
- Purpose: used to force a loop to repeat or to skip over a section of code entirely

`beq` (branch if equal): `beq src., src., LABEL`

- Definition: compares registers; if they are equal, the program jumps to LABEL
- Purpose: often used to check loop termination conditions

`bne` (branch if not equal): `bne src., src., LABEL`

`bge`, `ble`, `bgt`, `blt` (branch if $\geq, \leq, >, <$): `opcode src., src., LABEL`

- Definition: compares registers; if they are $\geq, \leq, >, <$, the program jumps to LABEL
- Purpose: often used to check loop termination conditions

2.4 Subroutines (Functions)

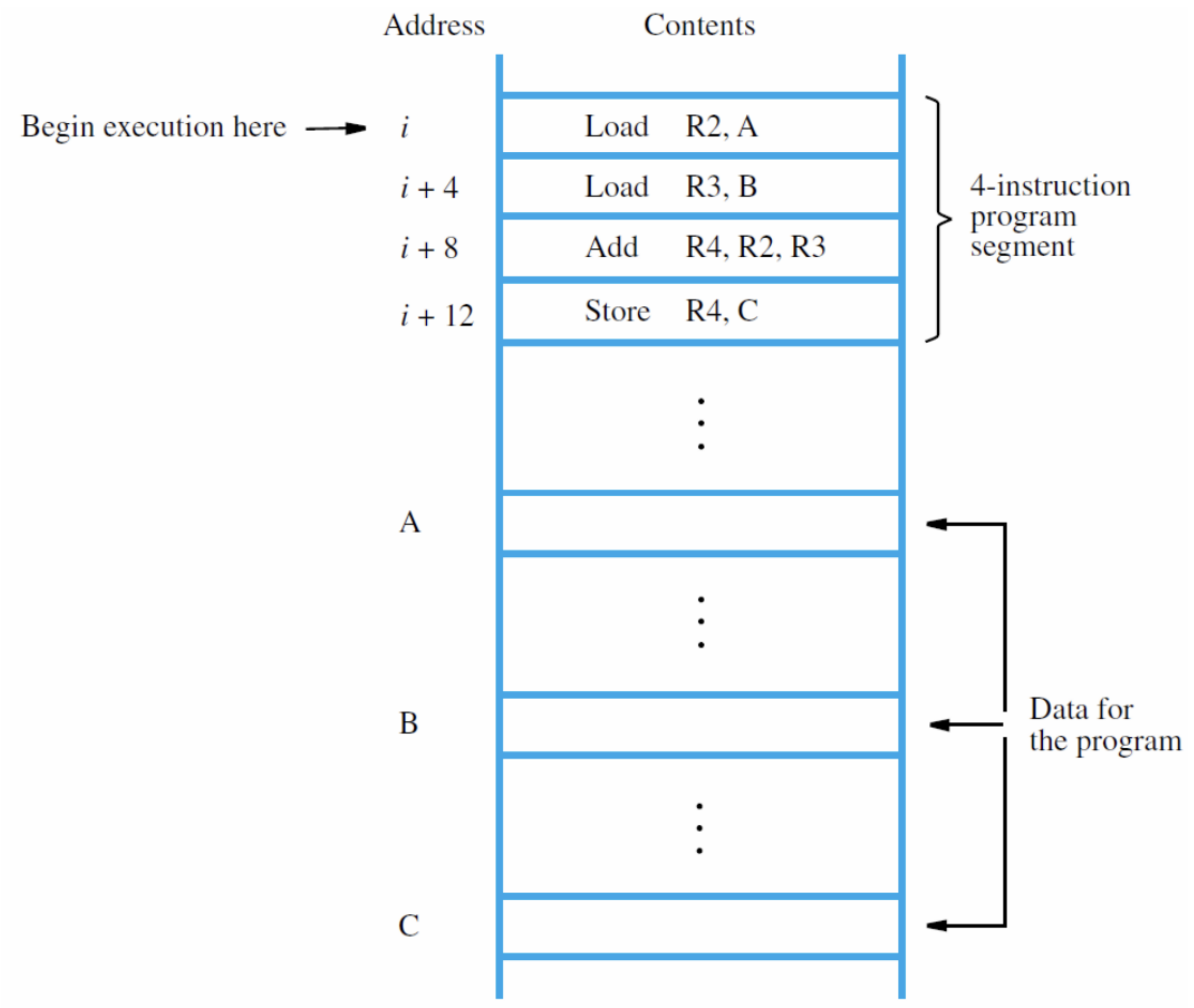
These instructions are used to call and return from functions

`call` (call subroutine): `call LABEL`

- Definition: jumps to LABEL and automatically saves the address of the next instruction into the return address register r31
- Purpose: allows the program to execute a separate block of code and remember where to come back to when it's done

`ret` (return):

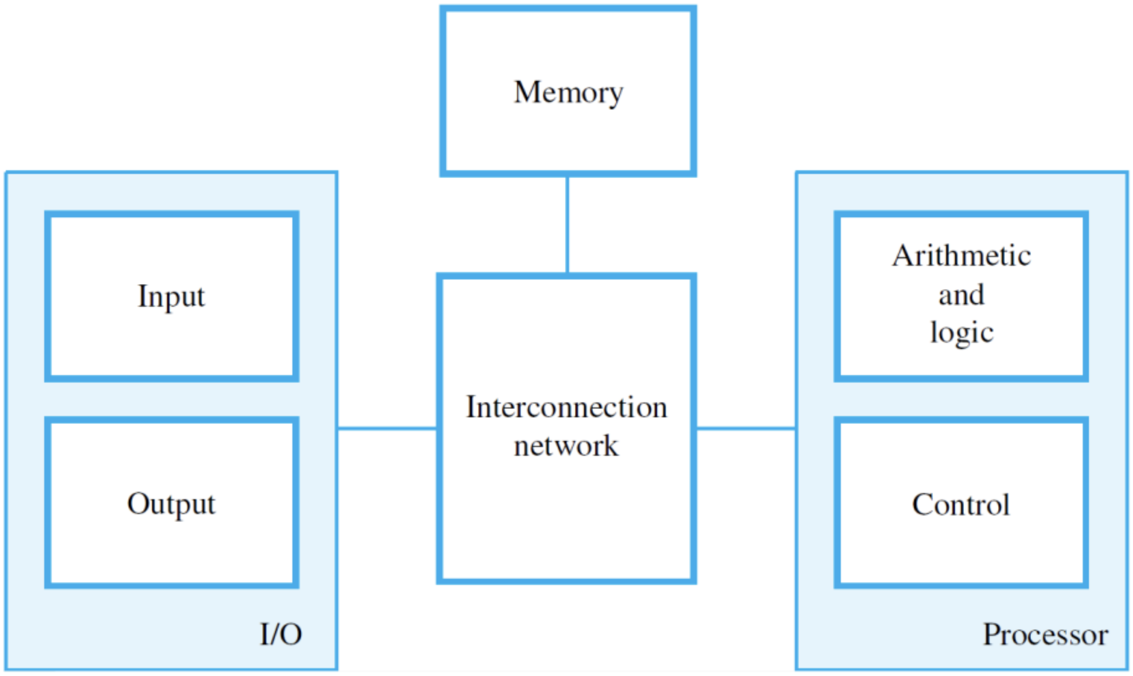
- Definition: jumps to the address stored in the return address register
- Purpose: used at the end of function to send the processor back to the point in the main code immediately following the call instructions


Address
Contents
Begin execution here
i
i + 4
i + 8
i + 12
Load R2, A
Load R3, B
Add R4, R2, R3
Store R4, C
4-instruction program segment
A
B
C
Data for the program

3 Basic Structure of Computers

Definition: Computer architecture is the specification of a set of instructions and behaviour of hardware units

Functional Units: Computers consist of 5 basic units: input, memory, arithmetic and logic, output, and control. The interconnection network supports transfer of information between units. The **processor** includes arithmetic and logic with control. The **I/O System** includes input and output units together.



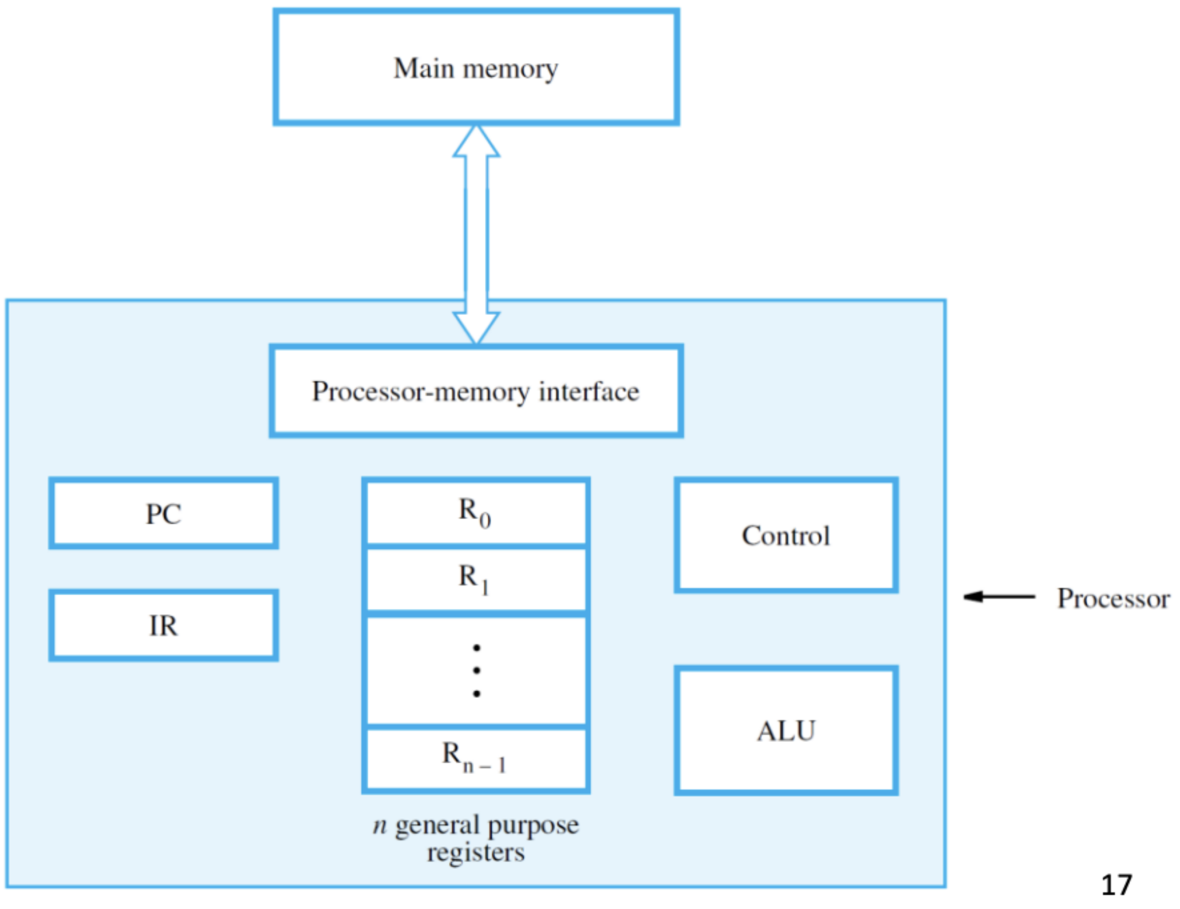
Input: Accepts coded information (in binary) for processing. Ex: mouse, keyboard, etc.

Memory: Stores programs (instruction lists) and data. Ex: RAM, SSD/HDD, cache

ALU: Performs arithmetic $(+, -, \cdot, /)$ and logic $(\wedge, \vee, \neg)$. Ex: high-speed registers for holding operands

Output: Presents processed results. Ex: displays, printers, and audio devices

Control: Coordinates all other units by sending timing and state signals. Ex: control circuits and control lines



How They Interact

1. **Instruction Fetching:** The **Control Unit** uses the **Program Counter (PC)** to find the address of the next instruction in **Memory**. This instruction is moved into the **Instruction Register (IR)** within the processor. During execution of each instruction, PC register is incremented by 4.
2. **Decoding:** The Control Unit interprets (decodes) the instruction in the IR to determine what action is required.
3. **Data Transfer:** Depending on the instruction, data may be moved from an **Input Unit** to Memory, or from Memory to **Processor Registers** (Load instruction).
4. **Processing:** If the instruction involves math or logic, the Control Unit directs the **ALU** to perform the operation using operands stored in registers.
5. **Storing and Outputting:** The result of a calculation is either kept in a register or written back to **Memory** (Store instruction). Finally, the **Output Unit** may transfer these results to a user or external device.

Instructions Running Cycle:

1. Fetch: The CPU fetches the instructions from memory. It uses the PC to know the memory address of the next instruction which is then loaded into the IR
2. Execute: The CPU carries out what the instruction says, could be ALU, jump/branch, moving data, etc.

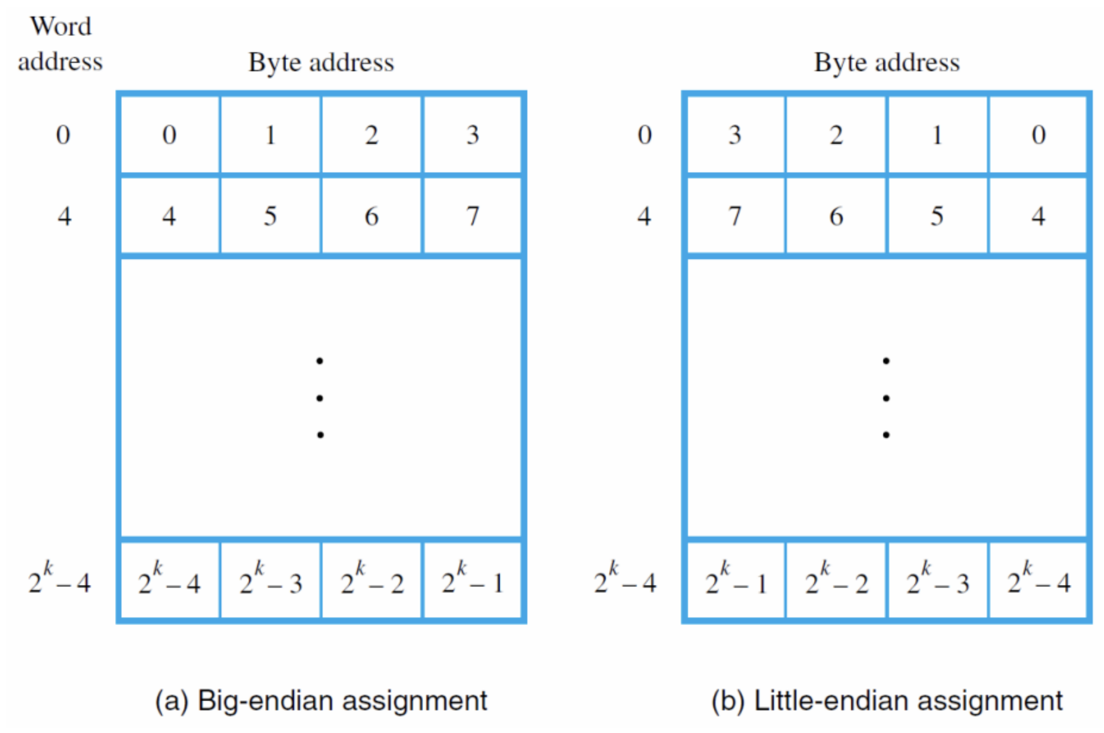
4 Instruction Set Architecture

A common word length is 32 bits. Numbers 0 to $2^k - 1$ are used as addresses for successive locations in the memory. Byte size is always 8 bits, but word length may range from 16 to 64 bits.

Byte locations have addresses $0, 1, 2, \ldots$, and word locations have addresses $0, 4, 8, \ldots$. We provide a byte-addressable memory that assigns an address to each byte. We have two ways to assign byte address across words.

Big-endian: Assigns lower addresses to more significant (leftmost) bytes of word

Little-endian: Uses opposite order.


Word address
Byte address
0
0 1 2 3
4
4 5 6 7
2^k-4
2^k-4 2^k-3 2^k-2 2^k-1
(a) Big-endian assignment
Byte address
0
3 2 1 0
4
7 6 5 4
2^k-4
2^k-1 2^k-2 2^k-3 2^k-4
(b) Little-endian assignment

4.1 Memory Organization

Hierarchy and Storage Types

Memory is organized into different levels to balance speed, capacity, and cost.

Primary Memory: This is fast, electronic memory composed of semiconductor storage cells. It is essential for storing programs and data currently in use.

Cache Memory: A smaller, faster electronic memory located on the same chip as the processor. It holds copies of instructions and data from the main memory that were recently used or are likely to be used soon, significantly speeding up access.

Secondary Storage: This provides large-capacity storage that retains information even when the power if off. While it is less expensive per bit, i tis generally slower than primary memory and has traditionally been based on magnetic or optical devices, though it now includes flash memory.

Physical and Logical Organization

The physical structure of memory dictates how the processor interacts with it.

- Binary Representation: Information is stored in bits
- Words: Bits are grouped into multi-bit "words" (typically 32 bits) to allow the processor to access multiple bits simultaneously for efficiency
- Addressing: Each word location has a unique address, numbered consecutively starting at 0
- Random Access Memory: Memory is organized so that any location can be accessed in a fixed, short amount of time, regardless of where the data is physically located