Bachelor of Science (Honours) in Data Science and Artificial Intelligence

DA 107 – Basic Computer System Architecture

Introduction

Outline

- First generation computers & John von Neumann architecture
- Basic Computing Concepts
- The Register File
- RAM
- A Closer Look At The Code Stream
- Register Vs Immediate

Outline

- Relative Addressing
- Mechanics of Program Execution
- Binary Encoding of Arithmetic Instructions
- Binary Encoding of Arithmetic Instructions With Immediate Value
- Binary Encoding of Memory Access Instructions
- The Store Instruction
- Example Programs

First Generation Computers & Architecture

First generation computers & architecture

- The Electronic Numerical Integrator And Computer (ENIAC) designed & developed by Prof. John Mauchley and his student J. Presper Eckert
- John von Neumann worked on the ENIAC project
- It consisted of 18,000 vacuum tubes and 1500 relays
- It weighted 30 tons and consumed 140 KW of power
- It has 20 registers each capable of holding 10-digit decimal number
- ENIAC was programmed by setting up 6000 multiposition switches and connecting multitude of sockets
- This machine was built around 1946.

First generation computers & architecture

- **Electronic Discrete Variable Automatic Computer (EDVAC) is the successor of ENIAC.**

- **John von Neumann built his own version of EDVAC.**

- **John von Neumann identified that programming computers with huge numbers of switches and cables was slow, tedious, and inflexible**

- **He came to realize that the program could be represented in digital form in the computer's memory, along with the data**

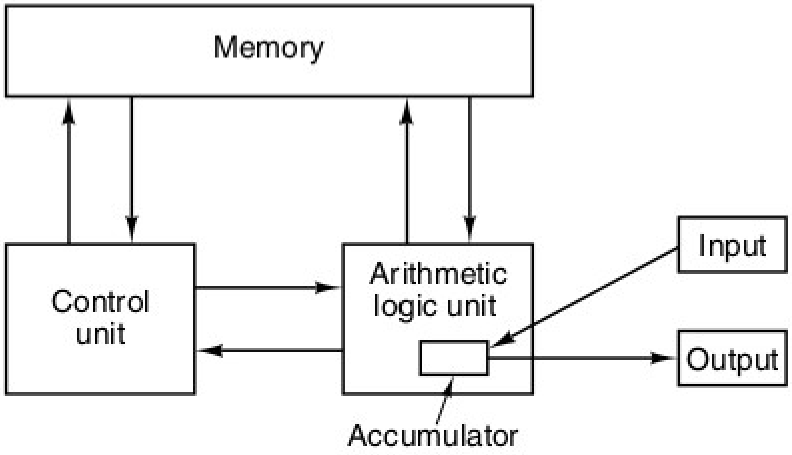
- **He also saw that the clumsy serial decimal arithmetic used by the ENIAC, with each digit represented by 10 vacuum tubes (1 on and 9 off) could be replaced by using parallel binary arithmetic**

First generation computers & architecture

- **The basic design, which he first described, is now known as a von Neumann machine.**

- **It was used in the EDSAC (Electronic Delay Storage Automatic Calculator), the first stored-program computer, and even now, more than half a century later, is still the b**

- **A sketch of the architecture is g**



First generation computers & architecture

- Memory consisted of 4096 words (a word holds 40 bits each 0 or 1)
- Each word held either two 20-bit instructions or a 40-bit signed integer
- The instructions had 8 bits devoted to telling the instruction type and 12 bits for specifying one of the 4096 memory words.
- Together, the arithmetic logic unit and the control unit formed the ‘‘brain’’ of the computer.
- In modern computers they are combined onto a single chip called the CPU (Central Processing Unit).

Basic Computing Concepts

Basic Computing Concepts

- At the heart of the modern computer is the microprocessor

- Commonly called the central processing unit (CPU)

- A computer takes a stream of instructions (code) and stream of data as input.
- Take the instruction +, take two numbers and perform addition
- Take the instruction -, take two numbers and perform subtraction
- Take the instruction *, take two numbers and perform multiplication
- Take the instruction /, take two numbers and perform division

- That is take stream of instructions and stream of data means, continuously perform these instructions on the data

- The results stream then is made up of results of these operations.

Basic Computing Concepts

- **Fundamental functions a computer performs:**
  - **Reading**
  - **Modification**
  - **Writing**

- **To achieve the above three fundamental functions, computer needs:**
  - **Storage**
  - **Arithmetic logic unit**
  - **Bus**

Basic Computing Concepts

Storage

- To say that a computer "read" and "writes" numbers implies that there is at least one number-holding structure that it reads from and writes to.
- All computers have a place to put numbers - a storage area that can be read from and written to

Basic Computing Concepts

Arithmetic logic unit

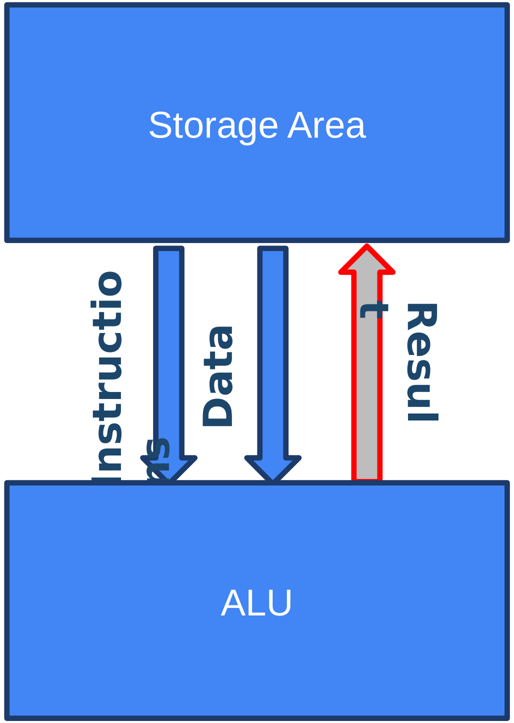
- To say that computer "modifies" numbers implies that the computer contains a device for performing operations on numbers. This device is ALU.
- It's the part of the computer that performs arithmetic operations (+, -, *, /)
- Numbers are read from storage into the ALU's data input port.
- Once inside the ALU, they are modified by means of an arithmetic calculation
- Output is written back to storage via the ALU's output port.

Basic Computing Concepts

Bus

- **To move numbers between the ALU and storage, some means of transmitting numbers is required.**
- **The ALU reads from and writes to the data storage area by means of the data bus**
- **Data bus is a network of transmission lines for shuttling numbers around inside the computers.**
- **Instructions travel into the ALU via the instruction bus**

Basic Computing Concepts



Basic Computing Concepts

- The ALU goes through the following sequence of steps
- Obtain the two numbers to be added from data storage
- Add the numbers
- Place the result back into data storage
- These three steps are carried out billions (10^9) of times per second on a modern CPU.

The Register File

The Register File

- Most computers have a relatively small number of very fast data storage locations attached to the ALU
- These storage locations are called registers
- The first x86 computers only had eight of them to work with
- These registers, which are arrayed in a storage structure called a register file, store the data that the code stream needs.

The Register File

- Building on the previous three-step description of what goes on when a computer's ALU is commanded to add two numbers, we can modify it as follows
- Obtain two numbers to be added from two source registers
- Add the numbers
- Place the result back in a destination register

The Register File - Example

- **Addition on a simple computer with 4 registers named A, B, C and D**
- **Suppose each of these registers contains a number**
- **Task: Add contents of two registers and overwrite the contents of third register with the resulting sum**

Code	Comments
A + B = C	Add the contents of registers A and B. Place the result in C. Overwrite whatever was there in C

The Register File - Example

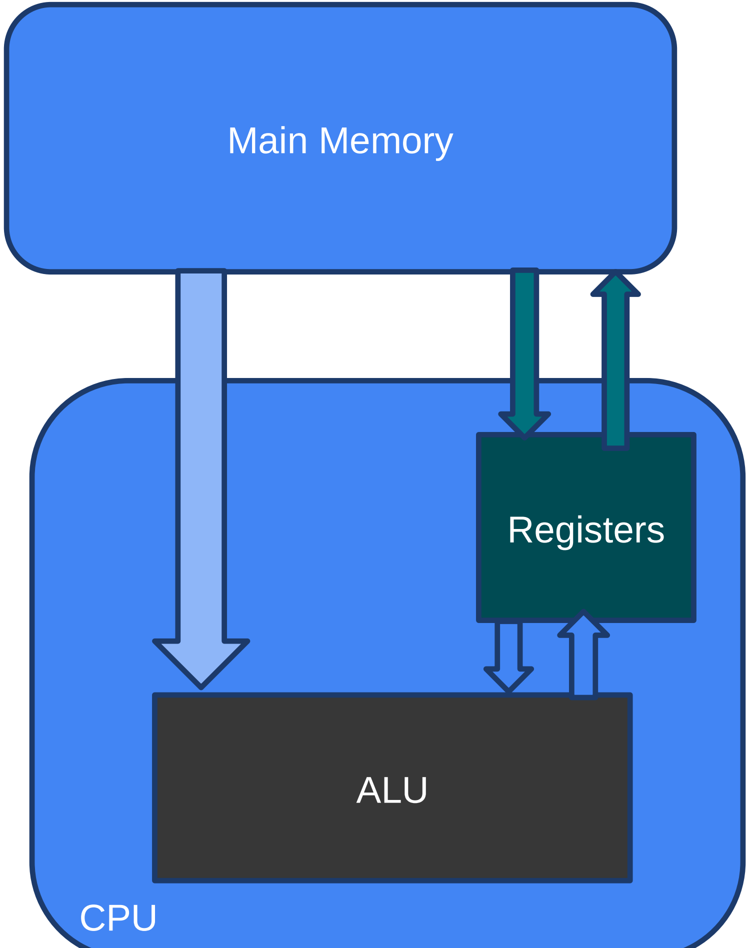
- Three steps are to be performed
- Read the contents of registers A and B
- Add the contents of A and B
- Write the result to register C

RAM

RAM

- Small set of registers are not very useful to perform reasonable computation
- To make a viable computer that does useful work, large storage is required
- This is where computer's main memory comes in
- Main memory, which in modern computers is always some type of random access memory (RAM), stores data on which computer operates
- A small portion of that data set at a time is moved to the registers for easy access from the ALU

RAM



Main memory is situated quite a bit farther away from the ALU

Registers are internal parts of the microprocessor

Main memory is a separate component and is connected to the processor via memory bus

Computation

- **How a computer uses main memory, the registers and the ALU to add two numbers**

- **Load the two operands (numbers) from main memory into two source registers**

- **Add the contents of the source registers and place the result in the destination register using the ALU. To do so, the ALU must perform these steps:**
    - **Read the contents of registers A and B into the ALUs input ports**
    - **Add the contents of A & B in the ALU**
    - **Write the result to the register C via ALU's output port**

- **Store the contents of the destination register in main memory**

Computation

- **The existence of main memory means that the user must manage the flow of information between main memory and the CPU's registers**
- **This means that user must issue instructions to more than just the processor's ALU**
- **He or she must also issue instruction to the parts of the CPU that handle memory traffic.**

A Closer Look At The Code Stream

A Closer Look At The Code Stream

- A code stream is an ordered sequence of instructions.
- Instructions are commands that tell the whole computer (Not just ALU) exactly what actions to perform
- A computer's list of actions encompasses more than simple arithmetic operations
- General instruction types
- If a programmer wants to add two numbers that are in main memory and then store the result back in main memory, he or she must write a list of instructions to tell the computer exactly what to do.

A Closer Look At The Code Stream

- A **load instruction** to move the two numbers from memory into the registers
- An **add instruction** to tell the ALU to add the two numbers
- A **store instruction** to tell the computer to place the result of the addition back into memory, overwriting what was previously there.
- These operations fall into two main categories
- **Arithmetic instructions:** These tell the ALU to perform an arithmetic calculation (**add, sub, mul, div**)
- **Memory-access instructions:** These tell the parts of the processor that deal with main memory to move data from and to main memory (**load** and **store**).

A Closer Look At The Code Stream

- **A detailed example is presented to show how memory access and arithmetic operations work together within the context of the code stream.**

- **These examples are based on a hypothetical computer DLW-1.**

- **Assume DLW-1 has four registers A, B, C and D.**

- **DLW-1 is attached to a bank of main memory that laid out as a line of 256 memory cells numbered #0, #1, #2 …., #255**

DLW-1's Arithmetic Instruction Format

- The DLW-1's arithmetic instructions are in the following instruction format
- Instruction source1, source 2, destination
- There are four parts to this instruction format each of which is called a field
- The instruction field specifies the type of operation being performed (+, -, /, *)
- The two source fields tell the computer which registers hold the two number being operated on
- The destination field tells the computer which register to place the result in.
- Instruction: add A, B, C

DLW-1's Memory Instruction Format

- **To get the processor to move two operands from main memory into source registers, a memory-access instruction load is used.**
- **The load instruction loads the appropriate data from main memory into the appropriate registers.**
- **The store instruction takes data from a register and stores it in a location in main memory.**
- **In DLW-1's this is represented as: instruction source, destination**

An Example DLW-1 Program

- Consider the piece of code in DLW-1

Line	Code	Comment
1	load #12, A	Read the contents of memory cell #12 into register A
2	load #13, B	Read the contents of memory cell #13 into register B
3	add A, B, C	Add the numbers in registers A, B and store result in register C
4	store C, #14	Write the result of addition from register c into memory cell #14

Main Memory	#11	#12	#13	#14
Before add	12	6	2	3
After add	12	6	2	8

Register Vs Immediate

Immediate Values

- Programmer needs to know the exact memory location to load and store
- For small programs this may be feasible. But real computers have billions of possible locations
- Programmer needs a flexible way to access memory
- Modern computers allow the contents of a register to be used as memory address
- The arithmetic instructions required two source registers as input.
- It is possible to replace one or both source registers with explicit numerical value called immediate value

Immediate Values

- Example, increase contents of register A by 2.
- For this no need to store 2 in a register and call add A, B, C
- Instead, write: add, A, 2, A
- Memory addresses are numeric values. They can also be though of immediate values (a number prefixed by #)
- Memory addresses can be stored in registers
- Thus contents of a register D could be constructed as memory address

Immediate Value

- Assume number 12 is stored in register D then the add program is written as

Line	Code	Comment
1	load #D, A	Read the contents of memory cell whose location is in register D
2	load #13, B	Read the contents of memory cell #13 into register B
3	add A, B, C	Add the numbers in registers A, B and store result in register C
4	store C, #14	Write the result of addition from register c into memory cell #14

Main Memory	#11	#12	#13	#14
Before add	12	6	2	3
After add	12	6	2	8

Immediate Value

- **Assume number 14 is stored in register D then the add program is written as**

Line	Code	Comment
1	**load** #11, D	Read the contents of memory cell whose location is in register D
2	**load** #D, A	Read the contents of memory whose location is in register D
3	**load** #13, B	Read the contents of memory cell #13 into register B
4	**add** A, B, C	Add the numbers in registers A, B and store result in register C
5	**store** C, #14	Write the result of addition from register c into memory cell #14

Main Memory	#11	#12	#13	#14
Before add	12	6	2	3
After add	12	6	2	8

Relative Addressing

Relative Addressing

- A data segment is a block of contiguous memory cells
- If the starting address of the data segment is known, one can access all other memory locations in the segment using the formula `base address + offset`
- Where offset is the distance of bytes of the desired memory location from the data segment's base address
- Assume base address is: 11

11	12	13	14	15	16

- To access memory address 12: specify 11 + 1 (`base address + offset`)
- To access memory address 13: specify 11 + 2 (`base address + offset`)
- To access memory address 14: specify 11 + 3 (`base address + offset`)

DLW-1 Program With Use Of Relative Addressing.

- **The processor takes the number in D and add 108**
- **Use the result as load's memory address**
- **Store also works in the same way as load instruction**
- **Load and store units on modern processor contain very fast integer addition hardware.**

Code	Comments
load #(D + 108), A	Read the contents of memory cell at location #(D + 108) into register A
store B, #(D + 108)	Write the contents of register B into the memory cell at location #(D + 108)

Mechanics of Program Execution

Mechanics of Program Execution

- To run a program, all instructions must be represented in binary notation.
- To represent instructions in binary notation, instructions are mapped to strings of binary numbers called opcodes.
- Each opcode designates a different operation

Mechanics of Program Execution

- The 3-bit opcode for the hypothetical DLW-1 microprocessor is given as:

Mnemonic	opcode
add	000
sub	001
load	010
store	011

- Register name to 2-bit binary code mapping is given as:

Register	Binary code
A	00
B	01
C	10
D	11

Mechanics of Program Execution

- The binary values representing both the opcodes and the register codes are arranged in one number of a 16-bit (2-byte) format to get a complete **machine language instruction**.
- This is a binary number which can be stored in RAM and used by the processor

Binary Encoding of Arithmetic Instructions

Binary Encoding of Arithmetic Instructions

- Arithmetic instructions have the simplest machine language instruction formats
- Table below shows the format for the machine language encoding of a register-type arithmetic instruction
- mode bit 0 denotes the instruction is register-type
- mode bit 1 denotes the instruction is immediate type

0	1	2	3	4	5	6	7
mode	opcode			source1		source2	

8	9	10	11	12	13	14	15
destinatio n		0	0	0	0	0	0

Binary Encoding of Arithmetic Instructions – Example

- `add A, B, C`

0	1	2	3	4	5	6	7
0	0 0	0		0	0	0	1
8	**9**	**10**	**11**	**12**	**13**	**14**	**15**
1	0	0	0	0	0	0	0

- The machine instruction is: 0000000110000000

Binary Encoding of Arithmetic Instructions – Example

- add C, D, A

0	1	2	3	4	5	6	7
0	0	0	0	1	0	1	1
8	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	0	0	0	0	0	0	0

- The machine instruction is: 0000101100000000

Binary Encoding of Arithmetic Instructions – Example

- sub C, D, A

0	1	2	3	4	5	6	7
0	0 0 1			1 0		1 1	
8	9	10	11	12	13	14	15
0 0		0	0	0	0	0	0

- The machine instruction is: 0001101100000000

Binary Encoding of Arithmetic Instructions With Immediate Value

Binary Encoding of Arithmetic Instructions With Immediate Value

- Arithmetic instructions with immediate value

0	1	2	3	4	5	6	7
Mode	Opcode			Source1		destination	

8	9	10	11	12	13	14	15
8-bit immediate value							

Binary Encoding of Arithmetic Instructions With Immediate Value

- Example: `add 5, A, C`

0	1	2	3	4	5	6	7
1	0	0	0	0	0	1	0

8	9	10	11	12	13	14	15
0	0	0	0	0	1	0	1

- The machine instruction is: 1000001000000101

Binary Encoding of Arithmetic Instructions With Immediate Value

- Example: add 25, A, C

0	1	2	3	4	5	6	7
1	0	0	0	0	0	1	0

8	9	10	11	12	13	14	15
0	0	0	1	1	0	0	1

- The machine instruction is: 1000000100011001

Binary Encoding of Memory Access Instructions

Binary Encoding of Memory Access Instructions

- Memory access instructions use both register & immediate-type instruction formats

- Load instruction: immediate-type

0	1	2	3	4	5	6	7
Mode	Opcode			0	0	destination	

8	9	10	11	12	13	14	15
8-bit immediate source address							

- As load's source is immediate and not register, bits 4 & 5 takes value 0.

Binary Encoding of Memory Access Instructions

- Example **immediate load instruction**: `load #12, A`
- Load instruction: immediate-type; its machine representation is

0	1	2	3	4	5	6	7
1	0 1 0			0 0		0 0	

8	9	10	11	12	13	14	15
0	0	0	0	1	1	0	0

- As load's source is immediate and not register, bits 4 & 5 takes value 0.

- `The machine instruction is: 1010000000001100`

Binary Encoding of Memory Access Instructions

- **Load instruction: register-type; its machine representation is**

0	1	2	3	4	5	6	7
Mode	Opcode			Source1		0	0

8	9	10	11	12	13	14	15
Destination		0	0	0	0	0	0

- **As load's instruction is register-type, bits 6 & 7 takes value 0.**

Binary Encoding of Memory Access Instructions

- **Load instruction: register-relative addressing; is similar to immediate-type format with base address and the offset stored in the second byte of the instruction**

0	1	2	3	4	5	6	7
Mode	Opcode			Base		destination	

8	9	10	11	12	13	14	15
8-bit immediate offset							

The Store Instruction

The Store Instruction

- The register-type binary format for a store instruction is the same as it is for a load, except that the destination field specifies a register containing a destination memory address

- Source1 field specifies the register containing the data to be stored to memory

- Immediate-type machine language format for a store is shown below

0	1	2	3	4	5	6	7
Mode	Opcode			source		destination	

8	9	10	11	12	13	14	15
8-bit immediate destination address							

Example Programs

Examples

Line	Assembly Language	Machine Language
1	`load #12, A`	10100000 00001100
2	`load #13, B`	10100001 00001101
3	`add A, B, C`	00000001 10000000
4	`store C, #14`	10111000 00001110

Thank You!