Machine Architecture Introduction and LMC Model

1 History of Computers

- Pre 1950s - meant a person in a room doing calculations
- 1871 - Babbage's analytical engine - advanced mechanical calculator that could be programmed in a limited sense. Programmes made by Ada Lovelace
- 1950s-1960s - Curta Calculator - sophisticated mechanical calculator

1.1 Enigma

Evolved from mechanical to electromechanical in the emigma machine.
Rotor wheels at the heart of the device physically set at the start, then rotated as typed. Wires within wheels to increase difficulty to decrypt without the rotors.

The enigma is a **polyalphabetic substitution cipher** based on electromechanical rotors that changed the substitution cypher after each letter.
This would have been unbreakable if better procedures had been followed during use

1.2 The BOMBE

Created by Turing and Colleagues. Tackled the enigma machine. Developed speed, efficiency and reliability.

1.3 Colossus

More sophisticated than the BOMBE, used against the Lorenz cypher. This was programmable, making it easier to input the new settings each day.

1.4 Turing's definition of a computer

"The idea behind digital computers may be explained by saying that these machines are intended to carry out any operations which could not be done by a human computer" - Alan Turing

1.5 A Turing Machine

An abstract mathematical model of computation. Manipulations of 0s and 1s on an infinite strip of paper. Head can make manipulations on the viewed square and move left and right. Shown that all things that could be computed by a more complicated model could be computed by a Turing machine. This is a universal computer

1.6 1948 - Manchester Baby

Winner of the race to build the first **Turing complete** computer. Impractical to use and program

1.7 1952 - IAS Machine

A more practical Turing Complete computer, team behind it headed by von Neumann. Program not hard wired into machine, instead in storage.

1.8 von Neumann Architecture

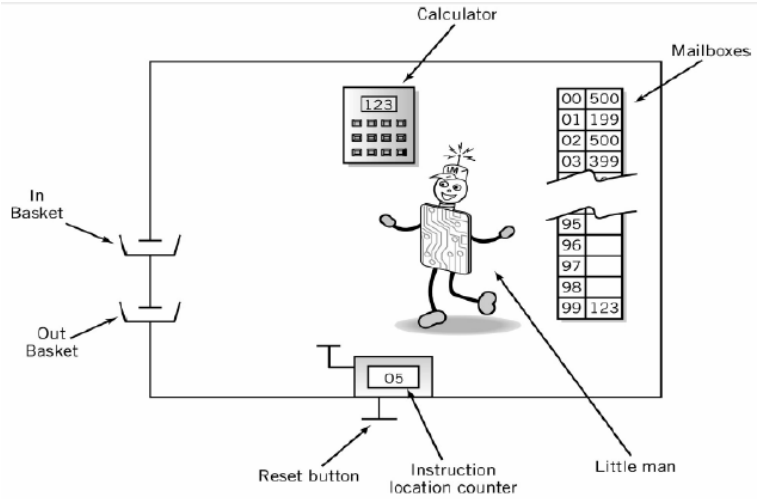
Program made part of data

Developed by the EDVAC design team for the US Army.
The main elements of a computer are virtually unchanged from this:

- Memory holds both programs and data - "Stored Program Concept" - this means machine doesn't need to be rewired for different programs

- Memory is addressed linearly - memory read in order, find what is a program and perform it
- Instructions executed sequentially (unless a branch or reset occurs)

2 LMC Model of Computing



Essential features:

- **Mailboxes** - Have 2 digit addresses so a maximum of 100 mailboxes. Each mailbox contains a slip of paper with 3 digits on it - no more. Store data in these mailboxes
- **Calculator** - Can only display 3 digits, has 0-9,+,- and a flag for negative results. Calculator will loop after 999
- **2 digit counter** - One button to increment the number, one external one to reset it to zero - tells the man which mailbox to look at next. Each time read, man increments it by 1
- **Input and Output Trays**
  - The user can put slips with 3 digits on the **input** tray, to be read when the little man next looks at this tray
  - The little man can write 3 digit notes and put them in the **output** tray, to be read by the user - whenever number read from input tray, put into calculator

The process, the little man:

- Starts by looking at the counter for a mailbox number X
- The man increments the counter by 1
- The man goes to mailbox X and reads what is written on the slip of paper in the mailbox - 3 digits
- The man takes the appropriate action depending on those digits
- The man starts again

2.1 Instruction set

The little man will read the 3 digit message in the mailbox he is sent to, for example 584
The first digit is the instruction (5)
The second and third digits are another mailbox number (84)
The instruction part is also known as an **op code** (operation code)

The power of this architecture is that a number in a mailbox can be treated either as a instruction or data

2.1.1 Example Instructions

op code 5 - LOAD
LOAD, so the man would go to the mailbox address specified, read the 3 digit number at that address, then go to the calculator and enter that number in

op code 3 - STORE
Go to the calculator and note that 3 digit number displayed, then go to the mailbox address specified and enter that number on a new slip of paper

op code 1 - ADD
Go to the mailbox address specified, read the 3 digit number at that address, then go to the calculator and add the number to the number already on the calculator

op code 2 - SUBTRACT
Go to the mailbox address specified, read the 3 digit number at that address, then go to the calculator and subtract the number from the number already on the calculator

op code 9 - INPUT/OUTPUT
Input/Read - op code 9 address 01 - Go to the IN tray, read the 3 digit number there, then go to the calculator and enter the number in
Output/Print - op code 9 address 02 - Go to the calculator, read the 3 digit number there, then go to the OUT tray and leave a slip of paper there with that number on it

op code 0 - BREAK The little man has a rest

2.1.2 Extended Instruction Set

op code 6 - BRANCH
Set the counter to the 2 digits specified in the address, and start fetch of instruction from this new address

op code 7 - BRANCH on ZERO
Go to the counter and read the 3 digit number. If it is zero, set the counter to the 2 digits specified in the address, and start fetch of instruction. Otherwise continue with the next instruction as normal.

op code 8 - BRANCH on POSITIVE
Go to the calculator and read the 3 digit number. If it is positive (including zero), set the counter to the 2 digits specified in the address, and start fetch of instruction, otherwise continue with the next instruction as normal

BRANCH can also be used to form loops by branching backwards

2.1.3 Instruction set overview

Machine Control
op code 0 - BREAK

Arithmetic
op code 1 - ADD
op code 2 - SUBTRACT

Data Movement
op code 3 - STORE
op code 5 - LOAD

Branching
op code 6 - BRANCH
op code 7 - BRANCH on ZERO
op code 8 - BRANCH on POSITIVE

Input/Output op code 901 - INPUT
op code 902 - OUTPUT

2.2 LMC fetch execute cycle

There are two essential phases of the LMC instruction cycle:

- **fetch** - in which the little man finds the instruction to execute
- **execute** - in which the little man performs the work specified in the instruction

2.2.1 Fetch

- The man goes to the counter and reads the address there
- The man goes to the mailbox at that address and reads the 3 digit number in the mailbox

2.2.2 Execute

- The man remembers the target address
- The man goes to the calculator and reads the 3 digits
- The man goes to the mailbox at the remembered address
- The man writes the 3 digits on a slip at that address
- The man goes to the counter and increments it

The man needs to **remember** an address and a 3 digit value

2.3 Potential errors

If data is stored without a halt beforehand, the data will be executed, causing an error
If numbers are added to the calculator, causing it to go over 999 (overflow), the number will loop back round to 000.
If the number goes negative through subtraction, the number given will also be wrong.
Make sure you expect all numbers as inputs, even if you don't expect certain ones to be given.

The negative flag will stay on as soon as any number in the calculator goes negative, and will not turn off, even if the number returns positive.

Usually better to use branch on positive rather than branch on zero, just in case you skip over 0

Branch on positive can be used as an error checker and you can direct the program to halt if the numbers go negative due to an error.

To check not had overflow, subtract addition from original number, error if number is positive. Use branch on positive.

3 Assembly Code

Uses mnemonics, unlike the LMC which only uses numbers.

Hash - comment
Column 1 - label
Column 2 mnemonic for op code
Column 3 - label for address

LDA - Load into accumulator (calculator in LMC)
DAT - Box used for data
STO - Store

BRZ - Branch on zero
HLT - Halt
BR - Branch

Lines of assembly sequentially given mailboxes, will fail to compile if not enough mailboxes

Data should be at the end as otherwise it will be assigned in the middle of the code and so the LMC will try to run it.

4 Harvard Architecture

Separate memory for instructions and data

- Quicker to execute as can access instruction and data at the same time
- Simpler to follow/analyse code
- Avoids the potential for some malware/bugs due to self-modifying code

Most modern processors have a CPU cache which partitions instruction and data, giving a "modified Harvard architecture" giving the best of both worlds