EECS 322 Computer Architecture

The first operational stored-program computer

Instructor: Francis G. Wolff
wolff@eecs.cwru.edu
Case Western Reserve University

This presentation uses powerpoint animation: please viewshow

EDSAC 1949: the first computer

Designed and built at Cambridge University, England, the EDSAC is the first full-scale *operational stored-program computer,* and is therefore the final candidate for the title of "the first computer".

The EDSAC performed its first calculation on May 6, 1949, when a length of perforated paper tape was threaded through the tape reader connected to the machine, and a few seconds later, the computer's printer began clattering out a list of numbers: 1, 4, 9, 16, 25, 36....

EDSAC Simulator: http://www.dcs.warwick.ac.uk/~edsac and Ref: http://hoc.co.umist.ac.uk/storylines/compdev/electronic/edsac.html

EDSAC: subroutines, relocatable, BIOS

• Indeed, EDSAC could access a library of programs called (would-you-believe) *subroutines,*

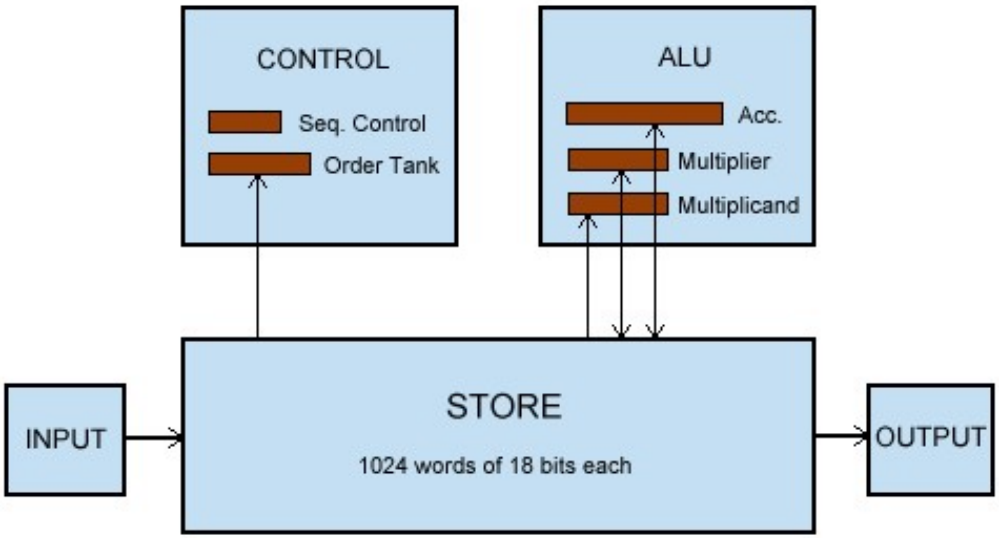
• including what was thought impossible at the time: a subroutine for numerical integration which (by calling an "auxiliary" subroutine) could be written without knowledge of the function to be integrated! (pass the *by address* of another function to a subroutine)

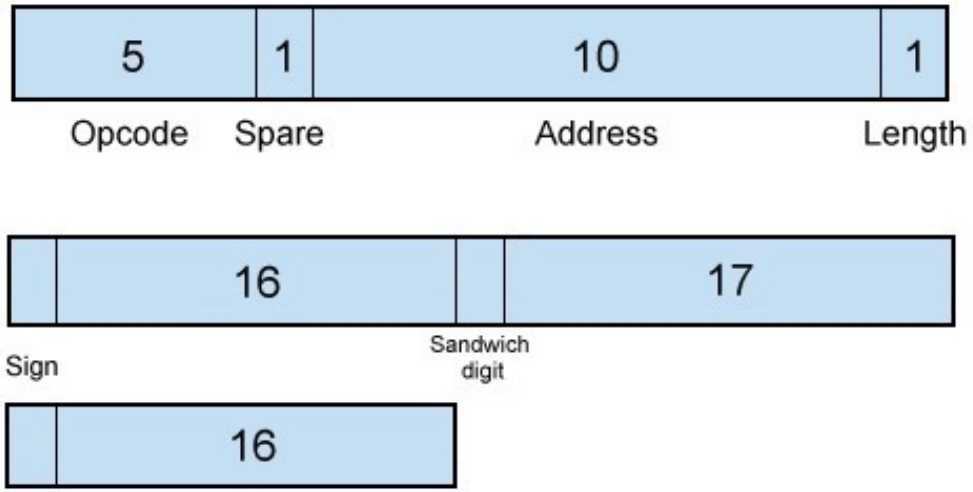
A problem: whenever a tape was read the subroutine may not go to the same memory locations so certain memory addresses had to be changed. This problem was overcome by preceding each piece of code with a set of "coordinating orders", making it *self-relocatable.*

• The next major advance demonstrated by this machine, was a continuation of EDSAC's subroutine idea. The concept of a *bootstrap was invented - a program that is run every time the machine is turned on*. Today, we call that shadow ROM BIOS.

EDSAC Simulator: http://www.dcs.warwick.ac.uk/~edsac and Ref: http://hoc.co.umist.ac.uk/storylines/compdev/electronic/edsac.html

EDSAC architecture





Typical execution times were

1.5 milliseconds for the simple commands = 667 adds/sec

4.5 milliseconds for a multiply = 222 mults/sec

http://www.cl.cam.ac.uk/UoCCL/misc/EDSAC99/simulators/echo/refindex.html

EDSAC memory

Its main memory is of a type that had existed for some years, but had not been used for a computing machine: the "ultrasonic delay line" memory.

It had been invented originally by William Shockley of Bell Labs (also one of the co-inventors of the transistor, in 1948), and Presper Eckert had made an improved version in connection with radar systems.

The "delay storage" referred to an electromechanical delay line: oscillating quartz crystals generated pulses in tubes of mercury and the pulses were recycled to provide memory.

In place of mercury, Turing suggested gin and tonic because the speed of propagation was relatively insensitive to temperature changes!

http://kbs.cs.tu-berlin.de/~jutta/time/msb-chronology-of-dcm.html
http://home.golden.net/~pjponzo/CSH.htm

Memory Store: Mercury Delay Tanks

EDSAC Description

System Clock: 0.5 Mhz

Arithmetic: No overflow or carry bit. Serial +, –, × and &

Registers: A=71 bits, multiplier H=35 bits, PC=10 bits, IR=15bits.

Better than a 32 bit processor!

One Instruction format: $\text{Opcode}_{18..14}\ \text{Spare}_{13}\ \text{Address}_{12..2}\ \text{Length}_{1}$

Input/Ouput Paper tape, Printer, 0-9 telephone dial, 16x36 video

Memory organization: 1024 words (i.e. about 2 kilobytes)

= 32 mercury tanks containing 32 18-bit words

Boot strap loader: Hardwired circuit fills first tank with 31 instructions

Today, we call that shadow ROM BIOS

Short word: Mem[n] $=\text{Mem}[n]_{18..1}$ (Bit 0 is always lost, can only use 17 bits)

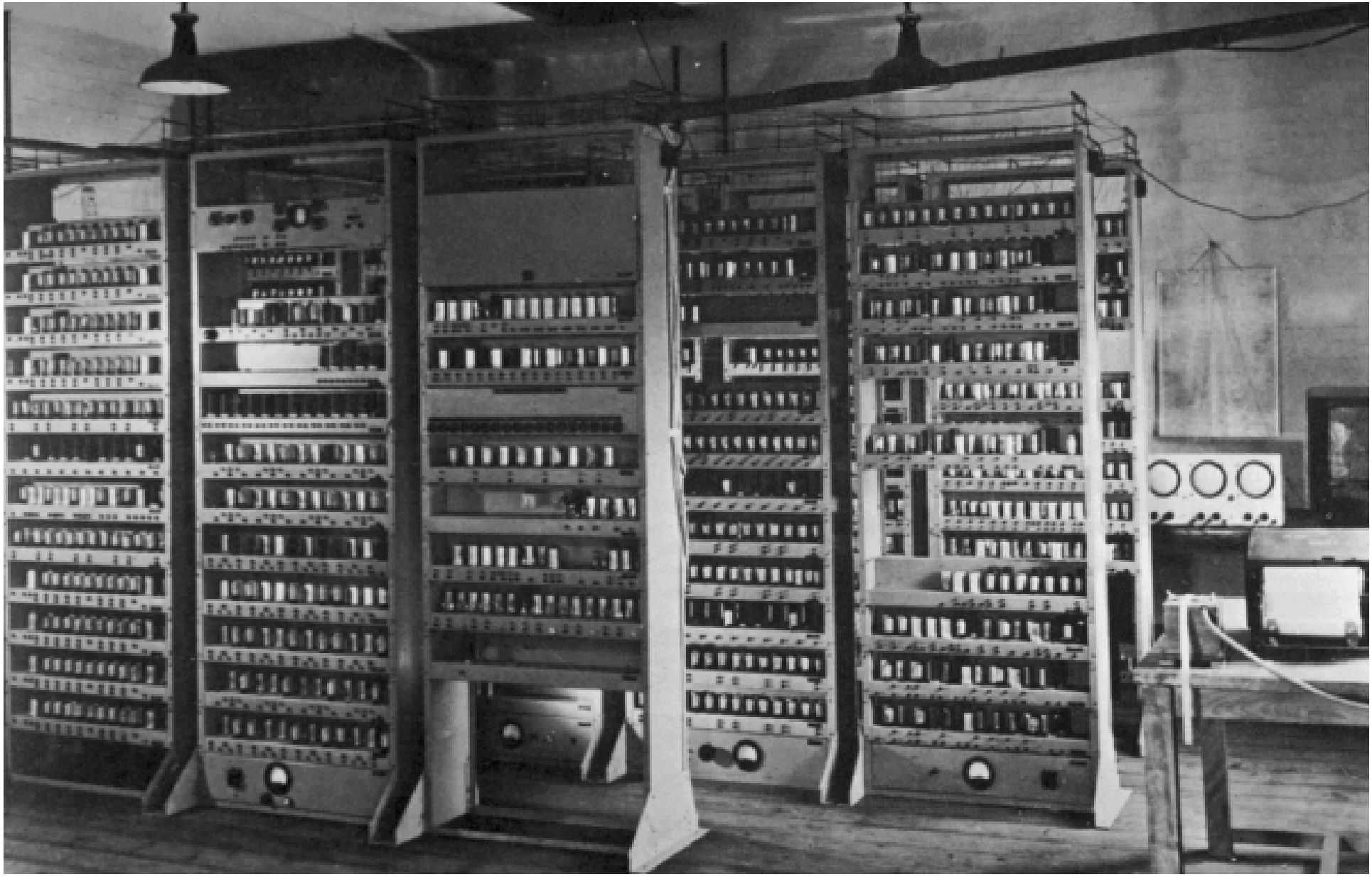
Long word: $\text{Mem}_{35..1}[n+1] = \text{Mem}[n+1]_{18..0} || \text{Mem}[n]_{18..1}$

Serial Memory: can run two adjacent memory location together

Technology: 3500 Tubes

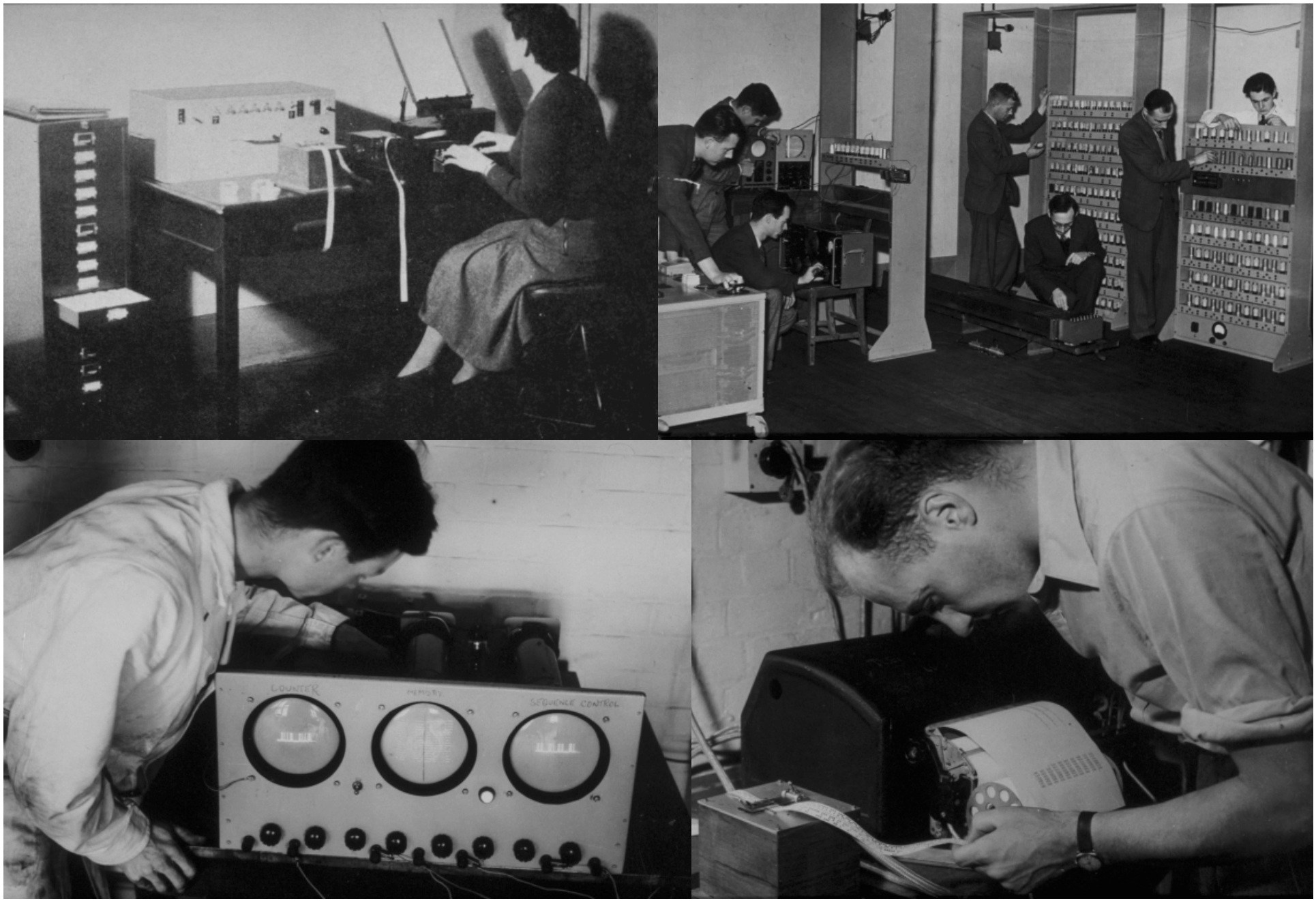
Ref: The Origins of Digital Computers, Brian Randell, 1975, 2nd, Springer-Verlag

EDSAC CPU



Ref: http://www.dcs.warwick.ac.uk/~edsac

EDSAC I/O



The University Mathematical Laboratory. Cambridge
May. 1950.

P.J.Farmer L.Foreman S.A.Barton G.J.Stevens R.Kimpton S.Gill P.Chamberlain
D.W.Willis K.N.Dodd M.Ellison B.P.Vernon H.Fye C.M.Beech R.B.Bonham-Carter A.E.Glennie D.J.Wheeler
E.E.C.McKee J.M.Bennett W.Renwick M.V.Wilkes E.N.Mutch R.A.Brooker C.M.Mumford

(Absent — B.M.Worsley D.G.N.Hunter)

EDSAC Instructions (formally called *orders*)

Instruction	
A n S	$A_{70..0} = A_{70..0} + Mem[n]_{18..1} \| \| 0_{52..0}$
A n L	$A_{70..0} = A_{70..0} + Mem[n+1]_{35..1} \| \| 0_{35..0}$
A n w	$A_{70..0} = A_{70..0} + Mem.w[n]$
S n w	$A_{70..0} = A_{70..0} - Mem.w[n]$
R n S	$A_{70..0} = A_{70..0} >> n$
L n S	$A_{70..0} = A_{70..0} << n$
C n w	$A_{70..0} = A_{70..0}$ & $Mem.w[n]$
H n w	$H_{34..0} = Mem.w[n]$
V n w	$A_{70..0} = A_{70..0} + H_{34..0} * Mem.w[n]$
N n S	$A_{70..0} = A_{70..0} - H_{34..0} * Mem.w[n]$

EDSAC Instructions

Instruction	
T n S	$Mem[n]_{18..1} = A_{70..53}$; $A_{70..0}=0$;
T n L	$Mem[n+1]_{35..1} = A_{70..36}$; $A_{70..0} =0$;
U n S	$Mem[n]_{18..1} = A_{70..53}$
U n L	$Mem[n+1]_{35..1} = A_{70..36}$;
E n S	$PC_{9..0} = (A >= 0)?\ n :\ PC_{9..0}+1$;
G n S	$PC_{9..0} = (A < 0)?\ \ n :\ PC_{9..0}+1$;
Z S	Stop the machine and ring the warning bell
I n S	$Mem[n]_{18..14}$ = Paper Tape Reader
O n S	Printer = $Mem[n]_{18..14}$ (print character in opcode position)
F n S	$Mem[n]_{18..14}$ = Printer character buffer

EDSAC 1952 Tic-Tac-Toe program

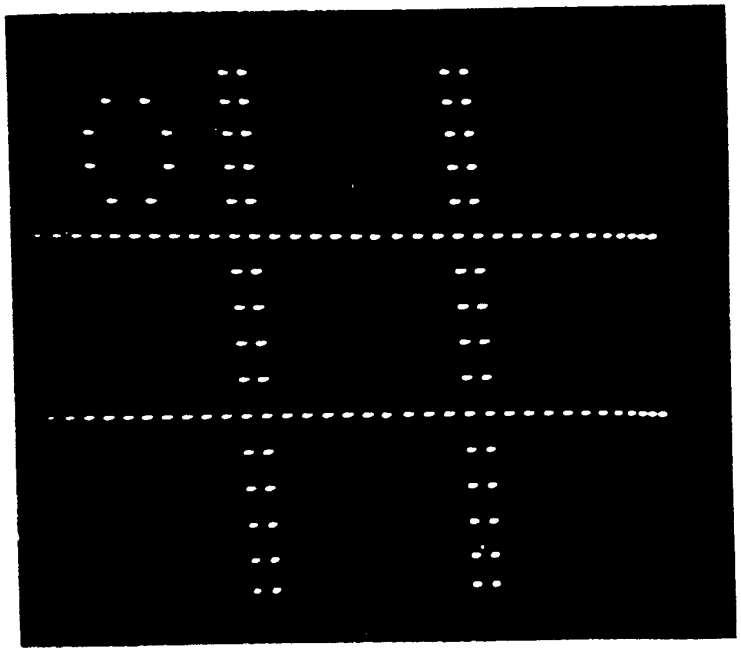
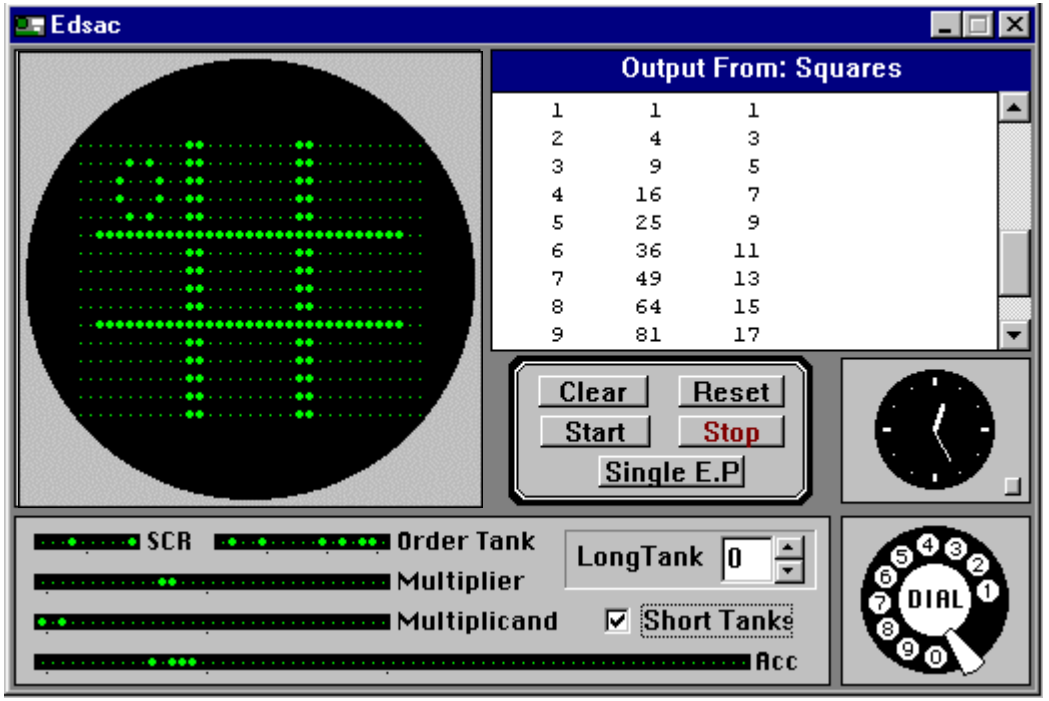


Diagram 1 (b). The board with a nought in position 9.



16 by 36 memory mapped monochrome (1-bit) video

Each memory bit corresponds to a pixel (picture element) on the display

The EDSAC Simulator: http://www.dcs.warwick.ac.uk/~edsac

Ref: http://www.cl.cam.ac.uk/UoCCL/misc/EDSAC99/

EDSAC instruction comparison

Modern computers provide instructions for

call: jal address

return: jr $ra

indexing: lw $rt, $offset($rs)

The EDVAC achieved this through *self modifying code*

At the time, the Von Neuman architecture was view as vital

(i.e. instructions and data are contained in the same memory)

For example: suppose loads on the MIPS *could not add* a base register

How would we do: lw $3,offset($1)

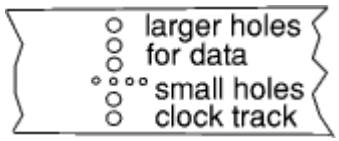
```
32:   addi   $2,$1,offset   #add offset plus base
36:   sh     $2,42($0)      #store within lw instruction
40:   lw     $3,0($0)
```

EDSAC Hello, World

```
31:     T53S    # A=0; last line of code +1 for loader
32:     O41S    # Printer = Mem[41..52]
33:     A32S    # A=A+Mem[32]; get instruction at 32
34:     A39S    # A=A+2; add 1 to address field
35:     U32S    # Mem[32]=A; store new instruction
36:     S40S    # A=A-"O53S"; stop output?
37:     G31S    # if (A<0) then no and goto 31
38:     ZS      # stop machine and ring the bell
39:     P1S     # use instruction to define word =2
40:     O53S    # use instr. to compare last index
```

```
41: *S #letter shift
42: HS
43: ES
44: LS
45: LS
46: OS
47: !S  #blank
48: WS
49: OS
50: RS
51: LS
52: DS
```

Note that the letter code and opcode as the same
Simplifies loader (loader acted as an assembler too!)
11100 = 'A' = Add opcode



Note that the letter code and opcode as the same
Actual paper tape source input (load for initial orders 1)
T53SO41SA32SA39SU32SS40SG31SZSP1SO53S
*SHSESLSLSOS!SWSOSRSLSDS

EDSAC versus the EDVAC: battle of being the first

Before von Neumann, computer programs were stored either mechanically (on cards or even by wires that connected a matrix of points together in a special pattern like ENIAC) or in separate memories from the data used by the program.

Von Neumann introduced the concept of the *stored program*—both the program that specifies what operations are to be carried out and the data used by the program are stored in the same memory.

Although EDVAC is generally regarded as the first stored program computer, Randell states that this is not strictly true [Randell94]. EDVAC did indeed store data and instructions in the same memory, but data and instructions did not have a common format and were not interchangeable.

Sadly, EDVAC was not a great success in practical terms. Its construction was (largely) completed by April 1949, *but it did not run its first applications program until October 1951.* (EDSAC was 1949)

Ref: http://wheelie.tees.ac.uk/users/a.clements/History/History.htm

EDSAC versus the Turing machine

In the 1930's, several mathematicians began to think about what it means to be able to compute a function. As we might phrase their common definition now:

A function is computable if it can be computed by a Turing machine(TM)

The TM model: A formal model for representing algorithms.

Church's Thesis: states that any algorithm can be represented as a TM.

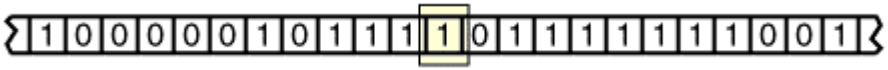
Turing complete: A system that is able to perform the same operations as the TM.

Universal Turing machines: A TM which acts like a modern general purpose computer in that it can "run" other TMs and thus solve any problem which can be solved by TMs.

An *algorithm* is a computational process that takes a problem instance and in a finite amount of time produces a solution.

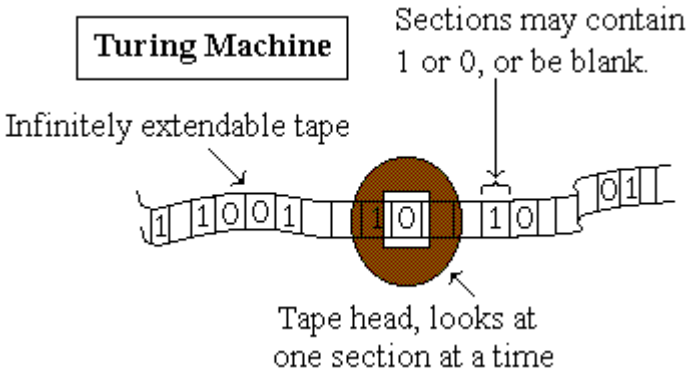
Undecidability: A formal proof that natural, important problems such as the halting problem are "undecidable" or unsolvable.

Turing machine operation



A *Turing machine* (TM) typically works as follows:

1. Read the input symbol from the tape.
2. Choose the next operation found in the state transition table (i.e. FSM), based upon the current state, and the input symbol.
3. Write the output symbol indicated in the matrix cell.
4. Transform into the next state indicated in the matrix cell.
5. Move the tape pointer in the direction indicated in the matrix cell.
6. If the next state is not H, the Halt state, start the instruction loop at the top.



State	Read	Write	Move	Next State
S1	0	0	L	S1
	blank	1	L	S2
	1	blank	R	S1
S2	0	1	R	S2
	blank	0	R	S2
	1	1	L	S1

State Transition Table for a Turing Machine

EDSAC versus the Turing machine

A Turing machine is a very simple machine, but, logically speaking, has all the power of any digital computer. It may be described as follows: A Turing machine processes an infinite tape *whereas a digital computer processes a finite tape.*

The most startling result of Turing's 1936 paper was his assertion that there are well-defined problems that cannot be solved by any computational procedure.

If these problems are formulated as functions, we call such functions *noncomputable*; if formulated as predicates, they are called *undecidable*. Using Turing's concept of the abstract machine, we would say that a function is noncomputable if there exists no Turing machine that could compute it.

EDVAC architecture comparison

EDVAC differs from the modern computers of today:

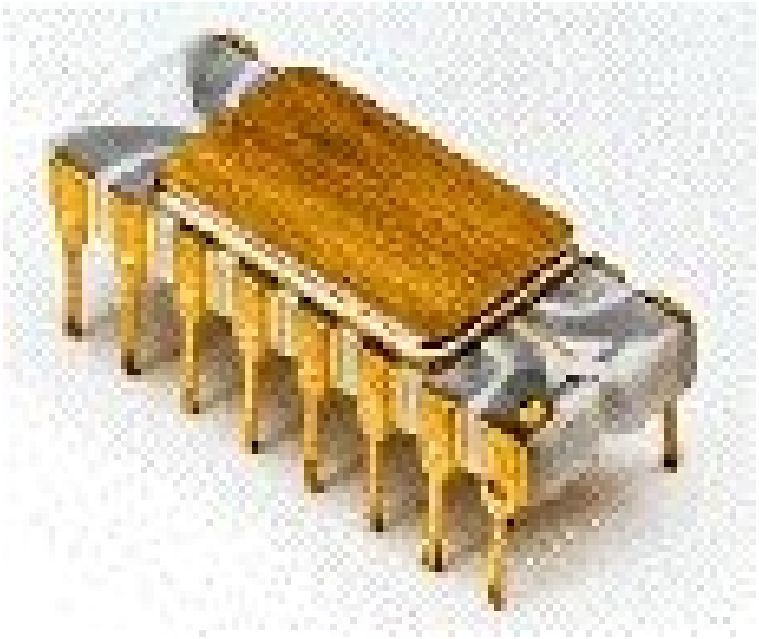
CPU:	Serial ALU to parallel & multiple ALUs and pipelining
Registers:	Serial 71 bit accumulator to 64bit parallel & multiple registers
Memory:	Serial Mercury Delay Tubes to parallel DRAM CMOS
	Single level memory to multilevel: Disk, RAM, L2, L1 cache
Input:	Paper tape to keyboards, mouse, scanners, cdroms, …
Output:	Teletype printer and a bell to 24-bit video, 16-bit sound,

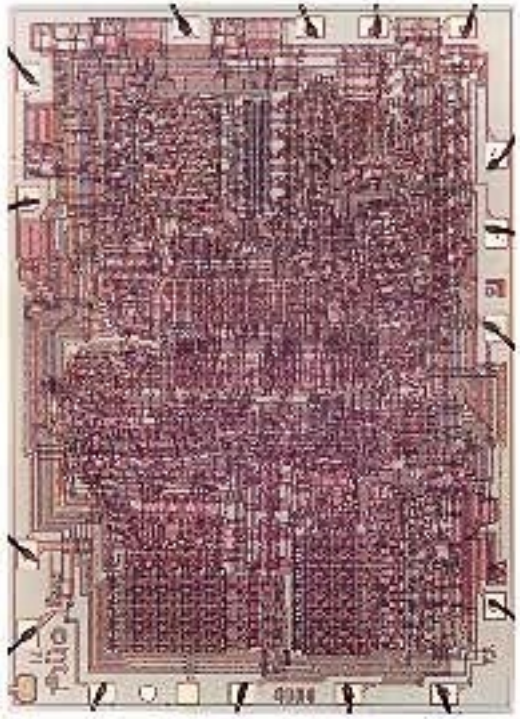
The key design components

parallelism:	achieved though architecture
switching delay:	achieved through technology (silicon)
area:	vacuum tubes to silicon
power:	vacuum tubes to silicon
cost:	mass manufacturing

Intel Microprocessor History: 4004

- **1971 Intel 4004, 4-bit, 0.74 Mhz, 16 pins, 2250 Transistors**

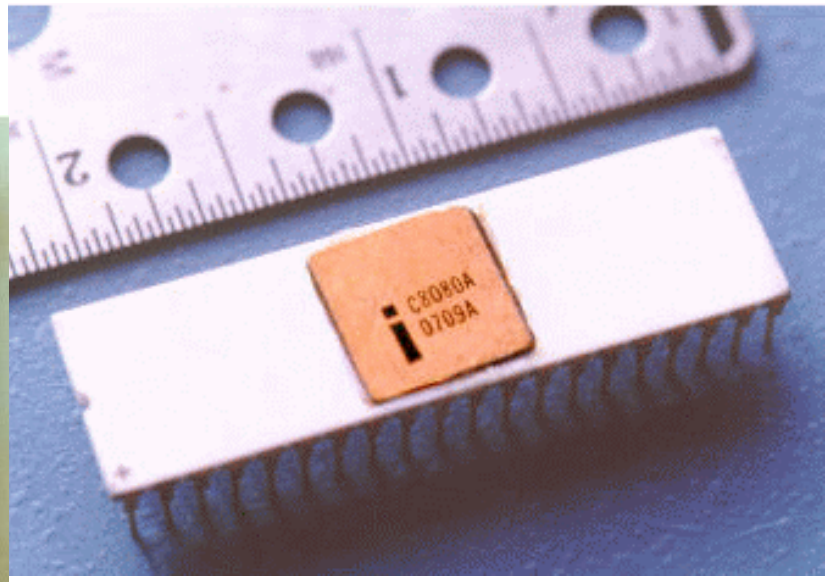


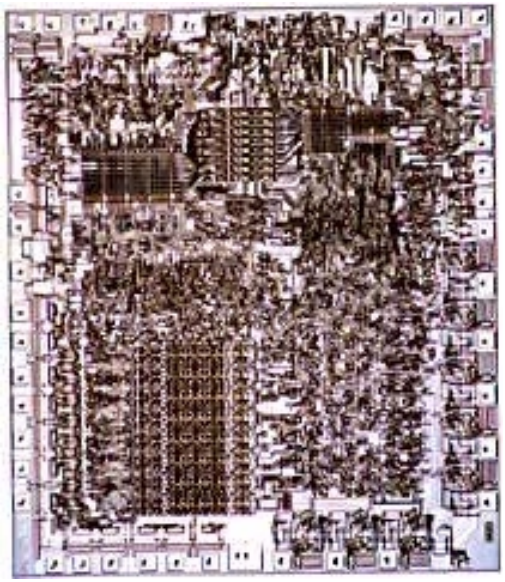


- **Intel publicly introduced the world's first single chip microprocessor: U. S. Patent #3,821,715.**
- **Intel took the integrated circuit one step further, by placing CPU, registers, memory access, I/O on a single chip**

Intel Microprocessor History: 8080

- **1974 Intel 8080, 8-bit, 2 Mhz, 40 pins, 4500 Transistors**



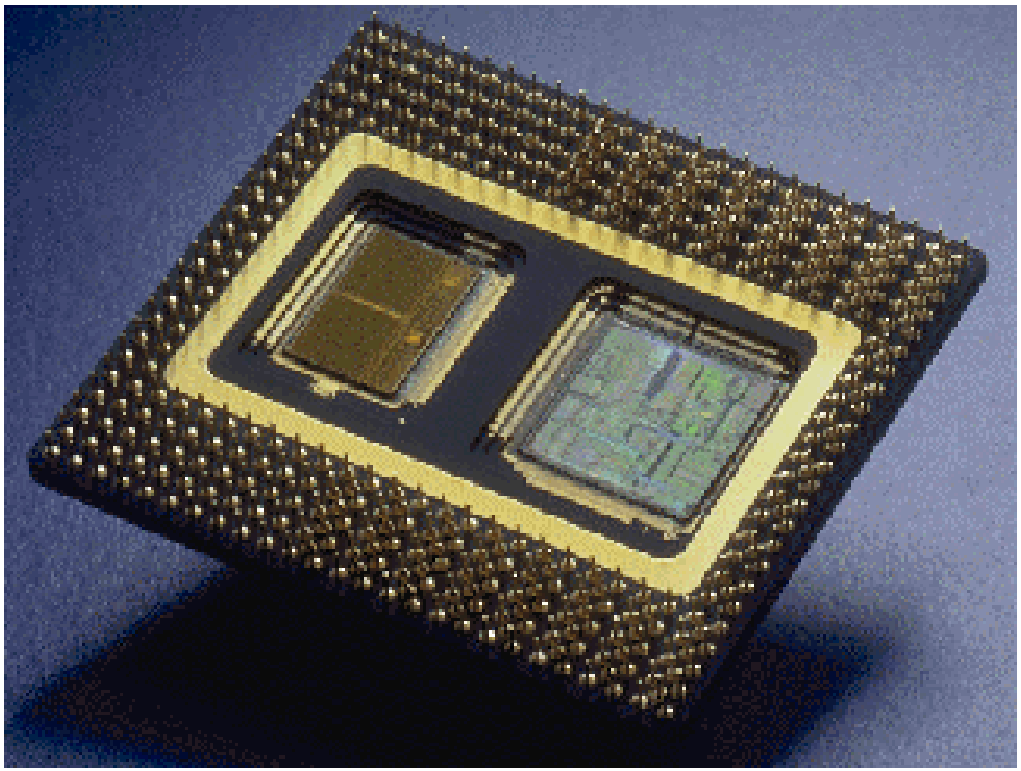


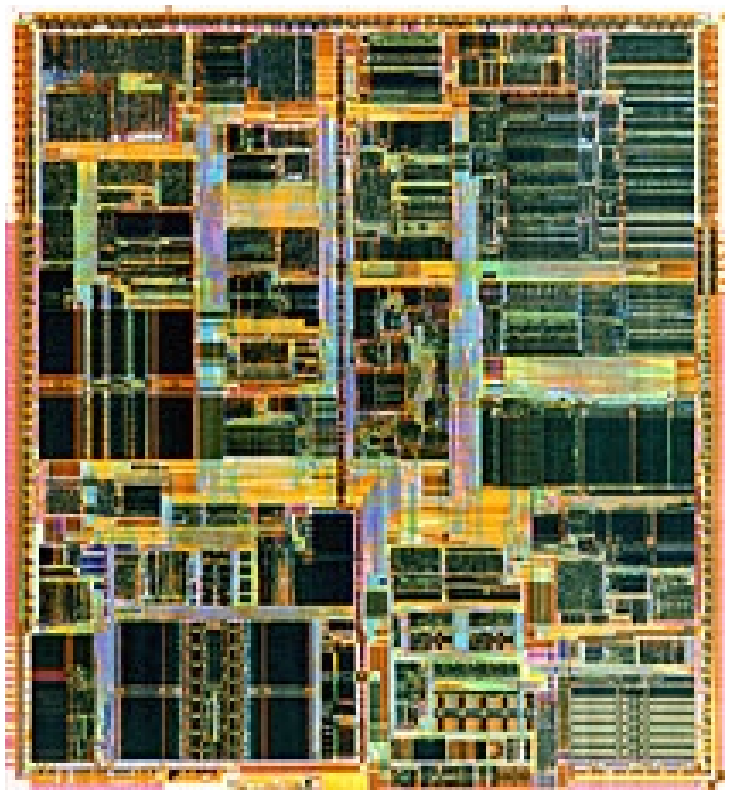
Bill Gates & Paul Allen

write their first Microsoft software product: Basic

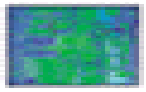
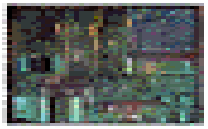
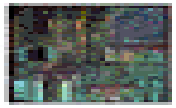
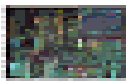
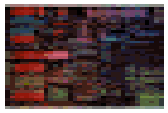
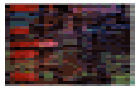
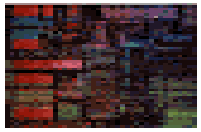
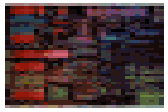
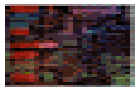
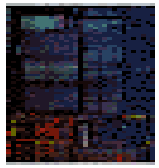
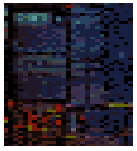
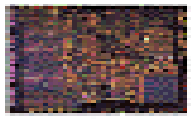
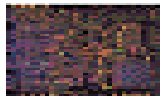
Intel Processor History: Penitum Pro

- **1995 Intel Pentium Pro, 32-bit ,200 Mhz internal clock, 66 Mhz external, Superpipelining, 16Kb L1 cache, 256Kb L2 cache, 387 pins, 5.5 Million Transistors**





Intel's microprocessor evolution

silicon process technology	1.5µ	1.0µ	0.8µ	0.6µ	0.35µ	0.25µ
Intel® Pentium® III processors						
Pentium® II processors						
Pentium® Pro processor						
Pentium® processor						
Intel486™ DX processor						
Intel386™ DX processor						

SoC: System on a chip (beyond Processor)

- **The 2005 prediction: SoC's will be > 100M gates**