13th World Computer Congress 94, Volume 2
K. Brunnstein and E. Raubold (Editors)
Elsevier Science B.V. (North-Holland)

On Basic Concepts of Early Computers in Relation to Contemporary Computer Architectures

Raúl Rojas[a]

[a]Mathematics and Computer Science Department, Freie Universität Berlin, Takustr. 9, 14195 Berlin, Germany

Abstract

In this paper we compare the architecture of early computing devices trying to understand if they contain the minimal structure needed to qualify as *universal computers*. Our intention is to bring some order into the discussion about the inventor or inventors of the computer by adopting a clear cut definition of such devices. We consider the minimal structure and the minimal instruction set computing automata should have in order to achieve universality, showing in this way that the invention of the computer was a truly international endeavor.

Keyword Codes: K.2; C.1.0; F.1.1
Keywords: History of Computing; Processor Architectures; Models of Computation

1. The problem of universal computation

The discussion about who should be called the true inventor of the computer has never been satisfactorily settled. The debate has been additionally obscured, because priority is claimed only over some part of what we understand today under the name 'computer'. For anyone interested in this debate, the first point to be clarified is certainly: what do we understand under the name "computer"? One possible way to arrive at a consensus is to define a computer as a device capable of *universal computation*, that is, capable of implementing all computable functions.

The general recursive functions are those which we call *computable* in the sense of Turing or Church. Any machine capable of universal computation should be able to implement those primitive functions and composition rules. This provides us with a yardstick with which the instruction set of a given machine can be analyzed in order to decide if it is indeed capable of universal computation. The following are the primitive functions and composition rules needed for the definition of the general recursive functions:

zero function	$x \mapsto 0$
successor	$x \mapsto x + 1$
projection	$p_i(x_1, \ldots, x_n) \mapsto x_i$
function composition	$f \circ g(x) = f(g(x))$
primitive recursion	$\phi(0, x_2, \ldots, x_n) = \psi(x_2, \ldots, x_n)$
	$\phi(y', x_2, \ldots, x_n) = \chi(\phi(y, x_2, \ldots, x_n), y, x_2, \ldots, x_n)$
μ operator	$\mu[P(t) = 0]$

The μ operator is the minimization function which for a function P gives the minimal value of t complying with $P(t) = 0$. Primitive recursion is defined in the usual way.

These functions can be implemented in a computing machine with a single accumulator by providing the following primitive instructions and sequencing rules:

zero function	CLR
successor	INC
projection	LOAD address
function composition	inst A
	inst B
primitive recursion	FOR i=0 to n
μ operator	WHILE (condition) DO

The CLR instruction sets the accumulator to zero, INC increments it by one and LOAD deposits the contents of the cell at location 'address' into the accumulator. Since the μ operator works implicitly with a table of function values it is also necessary to provide some kind of indirect addressing in the machine. Otherwise, it is not possible to access numerical tables of variable length. Tuple formation, included implicitly in the definition of functions like $f(x_1, x_2, \ldots, x_n)$ requires an additional STORE instruction to build those tuples (or tables) in memory.

The FOR and WHILE loops can be implemented in a simpler manner by providing some kind of conditional branch in the machine.

General-purpose programming languages implement iterative calculations over tables of numbers by providing indexed arrays. At the machine level some kind of indirect addressing is required to retrieve the elements of an array from memory. There are two possible solutions to this problem. The first is to make the LOAD instruction introduced above retrieve its parameter from a register in the processor. If we want to load the contents of address 100 to the accumulator, we first set the special register to 100 and we then load the address pointed to by this register. This requires additional hardware and a more sophisticated instruction decoding. The second is cheaper and is in fact the one which was used in early computers: Just store the program in memory and let the LOAD instruction work only with a constant argument. Since the program is stored in the memory of the machine, all instructions can be modified by the program being executed. A self-modifying program can store the code for the instruction "LOAD 100" or "LOAD 200" in the required address, and when this address is executed, the effect is the same as if we had provided a register for indirect addressing.

The same strategy can be used in order to define procedure calls. Instead of introducing a CALL instruction, an unconditional branch to the the code of the procedure is executed, but before doing this an unconditional branch back to the main code is inserted at the

end of the procedure's code. In this way the procedure is automatically threaded with the main program. This techniques were used in the University of Manchester's Mark 1 computer, universally acclaimed as the first one to provide memory stored programs [7].

2. Early computing machines

In this section we consider the architecture of several machines and we will be looking for the instructions CLR, INC, LOAD, STORE, conditional branches and indirect addressing (or equivalently self-modifying programs), when trying to decide if a particular implementation can be called a universal computer.

2.1. The architecture of the Z1

The Z1 was developed by the German engineer Konrad Zuse from 1936 to 1938 in Berlin (Figure 1). It was a mechanical computer, but it was not built with rotating parts like many of the calculators used at the time. It used instead sheets of metal capable of controlling binary switches used to store binary digits and to perform logical operations [6]. This is indeed the first interesting characteristic of this machine. Whereas other computers worked with a decimal representation even years after the Z1 was completed, Zuse's choice of the binary system allowed him to keep the the machine under a certain complexity level.

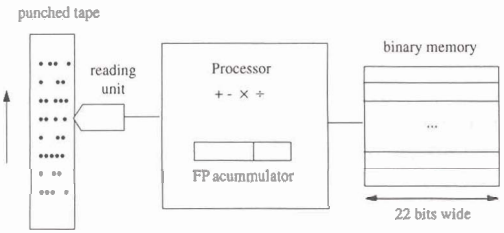

Figure 1. Logical structure of the Z1

As can be seen in Figure 1, the Z1 was capable of computing the four elementary arithmetic operations as well as the square root of numbers. The program was punched on a film tape and was read sequentially by a tape reader. Besides the arithmetic operations, there was an instruction for reading a number from a keyboard and storing it in any of 64 memory cells, as well as an instruction for showing the contents of the accumulator of the machine in a field of electric lamps. Years before John von Neumann explained the advantages of a computer architecture in which the processor has been logically separated from the memory, Zuse had already arrived to the same conclusions. Another interesting feature of the Z1 was its use of a floating-point representation. Each number was stored as a mantissa, an exponent and a sign. The processor computed all arithmetic operations using this coding (called by Zuse *logarithmic representation*) [11]. It would be more than fifteen years before someone built another machine with floating-point registers.

The Z1 lacked conditional or unconditional branches. It could not deal with indexed arrays of numbers and since the program was stored externally no self-modifying code was possible. The Z1 was nevertheless the first *program controlled* calculating device ever built. Its claim to fame lies in this fact.

2.2. Atanasoff's machine

John Atanasoff built his electronic computer at the University of Iowa from 1938 to 1942. His machine stands as a technological feat of simplicity. Whereas all other computing machines developed at the time worked with parallel arithmetic hardware, Atanasoff decided to use a serial approach. Two rotating drums (Figure 2) stored up to thirty fixed-point numbers. Each number was stored in one sector of the drum using 50 capacitors, that is, 50 bits. The machine could thus hold at any time just two different vectors in its $30 + 30$ memory cells. The vectors were read in decimal form from punched cards and were transformed to binary numbers before being stored. Atanasoff designed his machine in order to solve systems of linear equations with up to 29 variables. Since the machine worked with Gauss reduction, one of the vectors was used repetitively to reduce the other. The leading coefficient of the other vector was reduced to zero through a combination of additions, subtractions and shifts. The result was stored in a paper card and the next vector was loaded. Through a combination of manual and automatic operations, large matrices could be reduced to a triangular form.

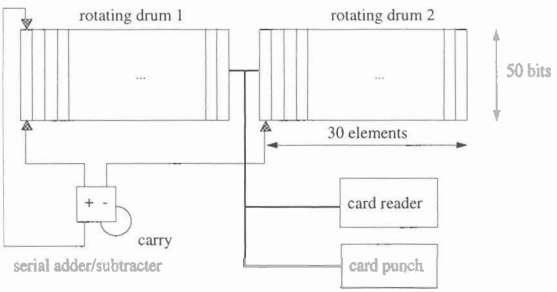

Figure 2. Logical Structure of the ABC

Since the drums rotated once every second, all bits of corresponding vector elements could be read sequentially and could be fed into a serial adder whose result could be written back to one of the drums. The adder could be implemented with a minimum of hardware by recycling the carriage bit. The adder could also subtract one number from another, if this was desired. A logical shift of a binary number was nothing but a read and write operation with a specific delay. There was an adder for each vector element. It would be 1948 before the Manchester team in charge of building a computer would rediscover serial arithmetic and all its benefits regarding the minimization of the hardware needed.

The ABC (Atanasoff-Berry-Computer, as it was called) was a special purpose machine, lacking any kind of external programming feature ("soft programming"). It could only be used to solve systems of linear equations. There was a clear separation between processor and memory, but the processor was not sophisticated enough. The ABC can be called the *first array processor* of the world and as such it would have been of use as an extension or coprocessor to a universal computer, but could not have replaced this one.

2.3. The architecture of Mark I

The Mark I was built by Howard Aiken at Harvard University from 1939 to 1944. Like Atanasoff, Aiken was a physicist anxious to create a machine capable of doing the extensive calculations that he needed. The Mark I was an electromechanical machine, a kind of hybrid between the all-mechanical nature of previous computing devices and the electronics available at the time. Figure 3 shows a diagram of the fundamental building blocks of Aiken's machine.

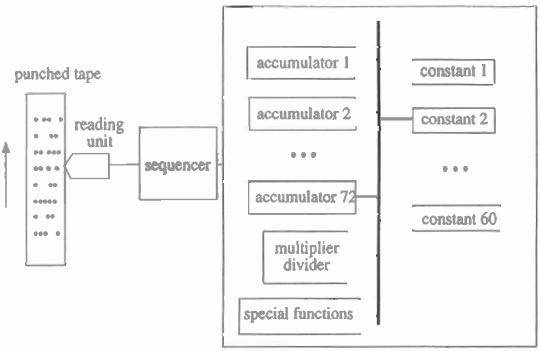

Figure 3. Logical Structure of the Mark I

Aiken used a decimal representation for storage. Rotating gears were used to store and transmit the numbers in the memory. There was no sharp distinction between processor and memory, since each of the 72 memory cells was an accumulator capable of adding or subtracting a given fixed-point number. Each accumulator could transmit its contents through what we would today call a bus to other accumulators. There were also special units which stored tables of numbers and a multiplier. Sixty storage words could be set manually by the operator and constituted the constants in a program. The program itself was stored on punched paper tape, and two reading units made possible the execution in parallel of two programs or of the same one (for checking purposes). The Mark I used what we now call a three-address format for its instructions. All elementary arithmetic operations were possible, as well as sequencing of instructions.

Aiken's Mark I was not a universal computer since it lacked conditional branches. It was possible to repeat a section of code a given number of times, but no conditional execution of loops was possible. Since the program was stored externally, self-modifying code could not be implemented.

2.4. The ENIAC

The ENIAC (acronym for *Electronic Numerical Integrator and Computer*) was built at the Moore School of Electrical Engineering of the University of Pennsylvania from May 1943 to 1945. It solved its first problem in December 1945 and was officially presented in February 1946. The project was conceived originally as an electronic realization of the analog differential analyzers used at the time for the same purpose. The ENIAC has often been hailed in the popular literature as the first computer of the world.

From the viewpoint of computer architecture the ENIAC was a *parallel dataflow ma-*

chine. There was no separation between memory and the computing sections. Decimal fixed-point numbers were stored in any of twenty accumulators, each one capable of transmitting its contents to any of the other accumulators, or of receiving a number and adding it to its stored content. There was a multiplying and a dividing unit. Since the machine operated with a basic clock rate of 100,000 pulses per second, it was felt by the designers that an external program (in punched tape, for example) would not fully exploit the calculating speed of the ENIAC. Programs were implemented by physically connecting the different modules, that is, by effectively hardwiring the program into the machine. Figure 4 shows a diagram representing a program hardwired in the ENIAC.

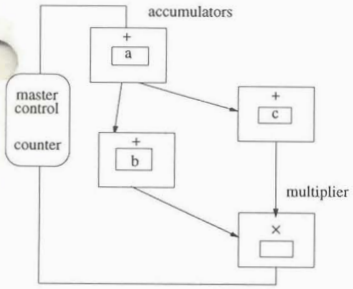

Figure 4. Dataflow in the ENIAC

If it was desired to perform a calculation like $ab + c$ it was necessary to connect two accumulators to the multiplying unit. One of the arguments was received first and the other was transmitted later. The output of the multiplying unit could be added with the contents of the accumulator holding c. Sequencing of arithmetic operations was achieved by connecting the arithmetic units in the desired order, even in parallel. The units worked asynchronously and in a self-clocking manner. A unit which was done with its calculation signaled this with a pulse to the next unit in the computation stream. The ENIAC was thus capable of implementing the CLR operation (since each accumulator could be reset to zero on demand), addition, load and store of numerical data. The ENIAC was also capable of implementing primitive recursion since a special unit, called the master controller, could restart a given thread of computation a fixed number of times. This is equivalent to a FOR loop in a high-level language. More important was the ability of the master controller to stop a given computational thread when a given quantity changed its sign. In this way it was also possible to implement WHILE loops and conditional branches in the machine.

The ENIAC stopped short a being a universal computer, not because the program was hardwired, but because it depended on a single master controller. Conditional loops were only possible at the highest control level and conditional branches consisted solely of transferring control from one loop to the next in a fixed sequence. It should have been possible to arrange for a hierarchical control mechanism consisting of multiple master controllers, but this was never done in practice [3]. Indirect addressing was clearly impossible, since memory and control were intertwined. Self-modifying programs were also ruled out, although some years after its introduction the ENIAC was modified to work with stored programs.

2.5. The Analytical Engine

Finally, we must consider the design of the Analytical Engine conceived by Charles Babbage in the nineteenth century. The architecture of Babbage's machine is surprisingly modern. It includes such features as a clean logical separation of numerical storage and processor, a microprogrammed computing unit and even a pipelined design! [2]. Babbage worked on the Analytical Engine from 1834 until his death in 1871.

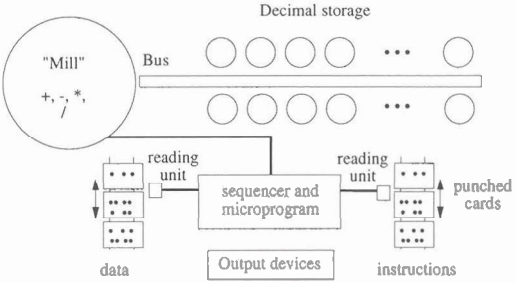

Figure 5. Logical Structure of the Analytical Engine

Figure 5 shows the main building blocks of the machine. Each number was stored in a vertical stack of rotating wheels using a decimal representation. The contents of each storage unit could be transported to the processor of the machine, called by Babbage "the mill". The four basic arithmetic operations were implemented in the mill. The program was read from punched cards, similar to the ones used in the Jacquard looms. Another set of cards contained the data for the program. This separation of code and data was a clever way of eluding the problem of indexed addressing. Since Babbage's machine could implement loops, it was possible to repetitively read the code cards at the same time as new data cards were sequentially fed into the machine. Long tables could be calculated in this way.

In his design Babbage also included the possibility of conditional branching. The reading units should work bidirectionally in order to make this possible. The machine should have also sophisticated output devices, like a machine capable of producing the lead plates needed to print the calculated numerical data or even what we would call today a plotter. Nevertheless the machine lacked any kind of indirect addressing. Babbage was aware of the restrictions imposed by an external program. Because of this, he speculated with the idea of including a card puncher as an output option. The machine could in this way produce its own programs.

3. Architectural comparison

As should be clear from the descriptions given above, no single machine fulfills all the necessary requirements for a universal computer. The Analytical Engine is surprisingly the one which comes closest. Had Babbage indeed built into his machine the possibility of punching his own code-cards, his would have been a real computer. The tables show the main features of the machines we analyzed. We also include the Mark 1 machine built at Manchester from 1946 to 1948, because as far as we know this was the first machine

to fit our definition of a universal computer. The Mark 1 was built under the direction of F.C. Williams and T. Kilburn. This machine stored its program in random-access digital memory implemented with CRT tubes. All necessary instruction primitives were available (in modified form), and although it lacked indirect addressing, self-modifying programs could be written [7].

Machine	memory and CPU separated?	conditional branching?	soft or hard programming	self-modifying programs?	indirect addressing?
Babbage's	√	√	soft	planned	×
Zuse's Z1	√	×	soft	×	×
Atanasoff's	√	×	hard	×	×
H-Mark I	×	×	soft	×	×
ENIAC	×	partially	hard	×	×
M-Mark 1	√	√	soft	√	×

Machine	internal coding	fixed-point or floating-point?	sequential logic?	architecture	technology
Babbage's	decimal	fixed-point	no	pipelined	mechanical
Zuse's Z1	binary	floating	no	sequential	mechanical
Atanasoff's	binary	fixed-point	yes	vectorized	electronic
H-Mark I	decimal	fixed-point	no	parallel	electromechanical
ENIAC	decimal	fixed-point	no	data flow	electronic
M-Mark 1	binary	fixed-point	yes	sequential	electronic

The ENIAC was the fastest machine in its time and could implement loops, but lacked any kind of soft programming. Atanasoff's machine was the first electronic calculating engine. Although the Mark 1 from Manchester was indeed a universal computer, it would be hard to credit its designers with having invented the computer single-handedly, since some of the design ideas were taken from the EDVAC project conceived by the ENIAC creators and John von Neumann. In fairness it should be concluded that the invention of the computer spans a period of more than hundred years, in which scientists from at least three different countries provided the theoretical and practical foundations for this enterprise. If there is a technological achievement in which only international cross-fertilization could have led to success, then the invention of the computer is indeed one.

REFERENCES

1. H. Aiken and G. Hopper, "The Automatic Sequence Controlled Calculator", reprinted in [9], pp. 203-222.
2. A. Bromley, "The Evolution of Babbage's Calculating Engines", *Annals of the History of Computing*, Vol. 9, N. 2, 1987, pp. 113-136.
3. A. W. Burks and A. R. Burks, "The ENIAC: First General-Purpose Electronic Computer", *Annals of the History of Computing*, Vol. 3, N. 4, 1981, pp. 310-399.
4. A. W. Burks and A. R. Burks, *The First Electronic Computer*, The University of Michigan Press, Ann Arbor, 1988.
5. B. Collier, *The Little Engines that Could've*, Garland Publishing, Inc., New York, 1990.
6. K.-H. Czauderna, *Konrad Zuse, der Weg zu seinem Computer Z3*, Oldenbourg Verlag, Munich, 1979.
7. S.H. Lavington, *A history of Manchester computers*, NCC Publications, Manchester, 1975.
8. *Early British Computers*, Digital Press, Manchester, 1980.
9. B. Randell, *The Origins of Digital Computers*, Springer-Verlag, Berlin, 1982.
10. N. Stern, *From ENIAC to UNIVAC*, Digital Press, Bedford, 1981.
11. K. Zuse, *Der Computer – mein Lebenswerk*, Springer-Verlag, Berlin, 1970.