RICE COMPUTER PROJECT: TECHNICAL GUIDE

John Iliffe

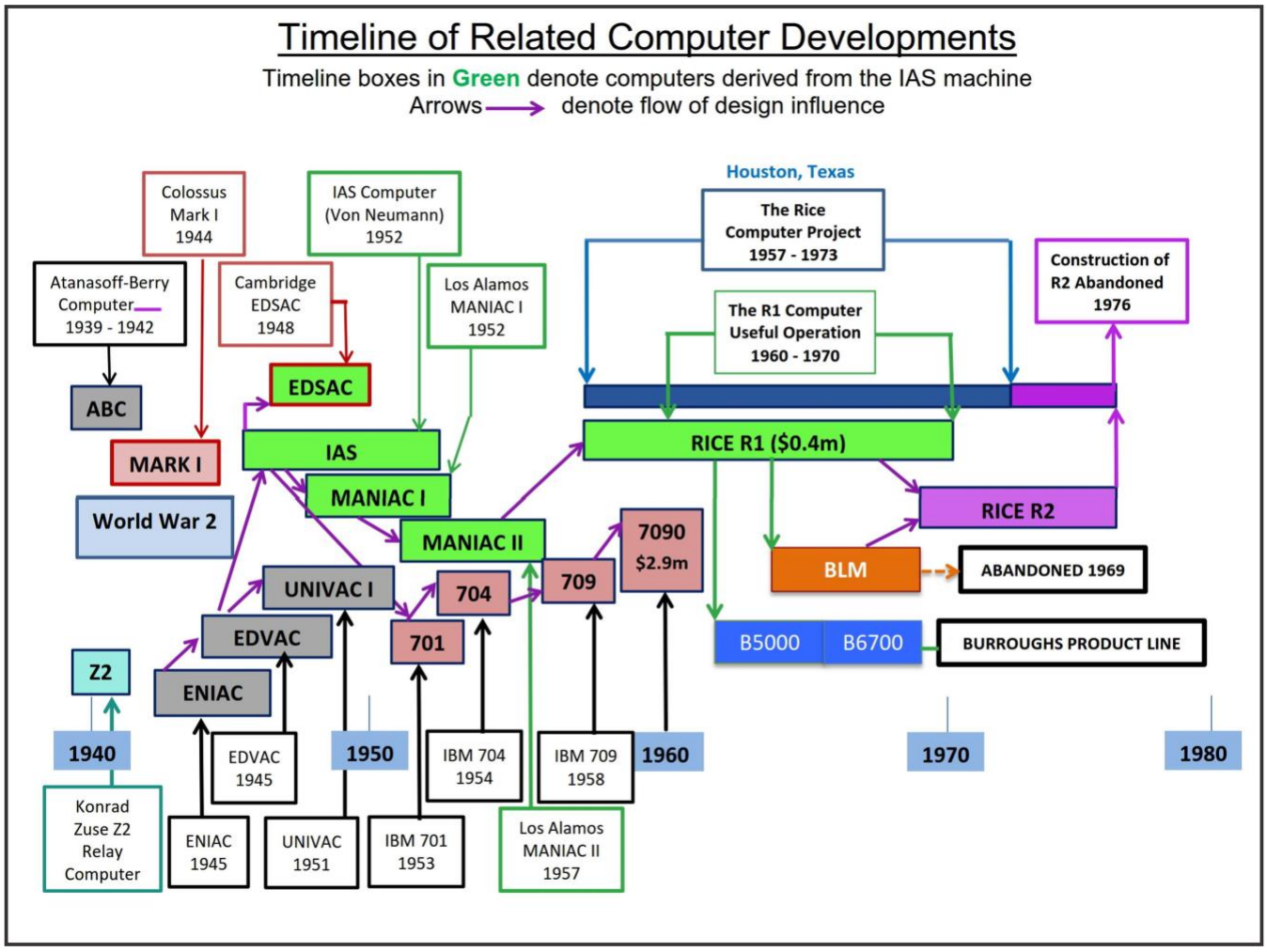
The Rice Computer Project is of historical interest for two reasons. Firstly, it typifies the environment in the late 1950's when many questions of instruction design, arithmetic function, order codes, operating systems and programming languages, which are now submerged under a legacy of software, were debated. And, secondly, it laid the foundation of a method of storage access and management that is still in active use and relevant to contemporary preoccupation with protection and security.

In these notes I will attempt a review of the project as I experienced it, and (with help from former colleagues) describe some of the innovative hardware and software.

(Left to right) Iliffe, Graham, Salsburg, and Kilpatrick in front of R1 in 1960.

Early history of the R1 project

In 1956 Zevi Salsburg, John Kilpatrick (Professors of Chemistry), and Lawrence Biedenharn (Professor of Physics) at the Rice Institute, who used the facilities of the Atomic Energy Commission at Los Alamos, wrote to the AEC asking that a project be established at Rice to build a new computer. The odds were slim that the AEC would act but in fact in 1957 it did. The Project was established as an independent agency at Rice. Shell Development Company in Houston provided additional financial support.



Dr. Martin Graham, who had been working on the design of the MERLIN at Brookhaven National Laboratory and consulting on the MANIAC II project at Los Alamos, was appointed to design and build the machine. He and the three founders of the project decided that the Rice Computer, or "R1", should be an improved version of MANIAC II.

Some of the most important projects in the early history of electronic computers, including those that had any influence on R1 are indicated in the above diagram. The influence of the Los Alamos computers, and the type of calculation carried out on them, will become evident. In summary, it can be said that R1 achieved its objectives, and in

the targeted problem areas it merited comparison with the IBM 709, particularly when higher precision arithmetic was called for.

In 1958 I was invited to join the Project with the aim of developing an operating system and programming languages. Up until then I had been managing the IBM service bureau in London based on the IBM 650 and 704, and welcomed the opportunity to try out new ideas on a machine powerful enough to put them to practical use.

Arriving in September, within a week I found myself presented with the Instruction Manual and discussing the order code with Graham, Kilpatrick, Salsburg and Bob Barton, from Shell. (By this time Biedenharn had left Rice.) It was at that 1958 meeting that John Kilpatrick outlined a new method of indirect addressing that had far-reaching consequences, of which more will be said later.

R1 came into operation in 1960, with 8K words of electrostatic storage. To modern readers this will seem impossibly small, but in comparison with contemporary machines it presented an opportunity to realise practical ambitions that had been limited in the 1950's. In 1966 the memory was extended to 24K words, using magnetic core storage.

YEAR	MACHINE	WORD (bits)	MEMORY (words)
1952	Maniac 1	40	1K
1954	IBM 704	36	4K – 32K
1957	Maniac II	48	12K
1959	IBM 709	36	4K – 32K
1960	R1	56	8K – 24K

R1 was not a time-sharing machine. It was designed for hands-on use, and although it provided very sophisticated methods of program construction and management they were not proof against error propagation. To follow the later history of the Project it will be necessary to step aside to understand the Basic Language Machine, or Burroughs B6700, in which strict controls were imposed, as will be explained later.

Graham left the Project in 1966, to be replaced by Walter Orvedahl, from Maniac III at the University of Chicago. The second phase of the Project, initiated by Orvedahl and Edward Feustel from 1968, was then to design an improved architecture, "R2", which remained faithful to the original concepts R1 while extending its abilities in arithmetic and store management. This is described briefly below, but it will be seen that unreliable hardware and withdrawal of funding in 1976 prevented its completion.

In the later stages of the Project advantage was taken of the inherent flexibility of the R1 microprogram to add to the order code and to attach peripheral devices of particular interest to researchers at Rice.

References

The main source of Project material is the archive donated to the CHM by Ed Feustel. In this note I will add references to the most useful accounts relating to each section, using Catalog numbers from the Draft collection edited by Paul McJones. Where other material is sourced I will aim to give publication particulars.

The Final Technical Report 102726204 (Orvedahl 1970) gives an account of work undertaken and the wider influence of the design. The TV documentary "The Completed Computer" (reference unknown, Fondren Library 1960) illustrates the physical construction and operating procedures. Adam Thornton's "A brief history of the Rice Computer1959-1971" provides a useful summary of the instruction code and system. (https://web.archive.org/web/19961222005446/http://www.princeton.edu/~adam/R1/r1rpt.html). Graham's Oral History (CHM 102746199) supplies historical and technical comments.

Rice Computer Project Team 1959

The R1 architecture

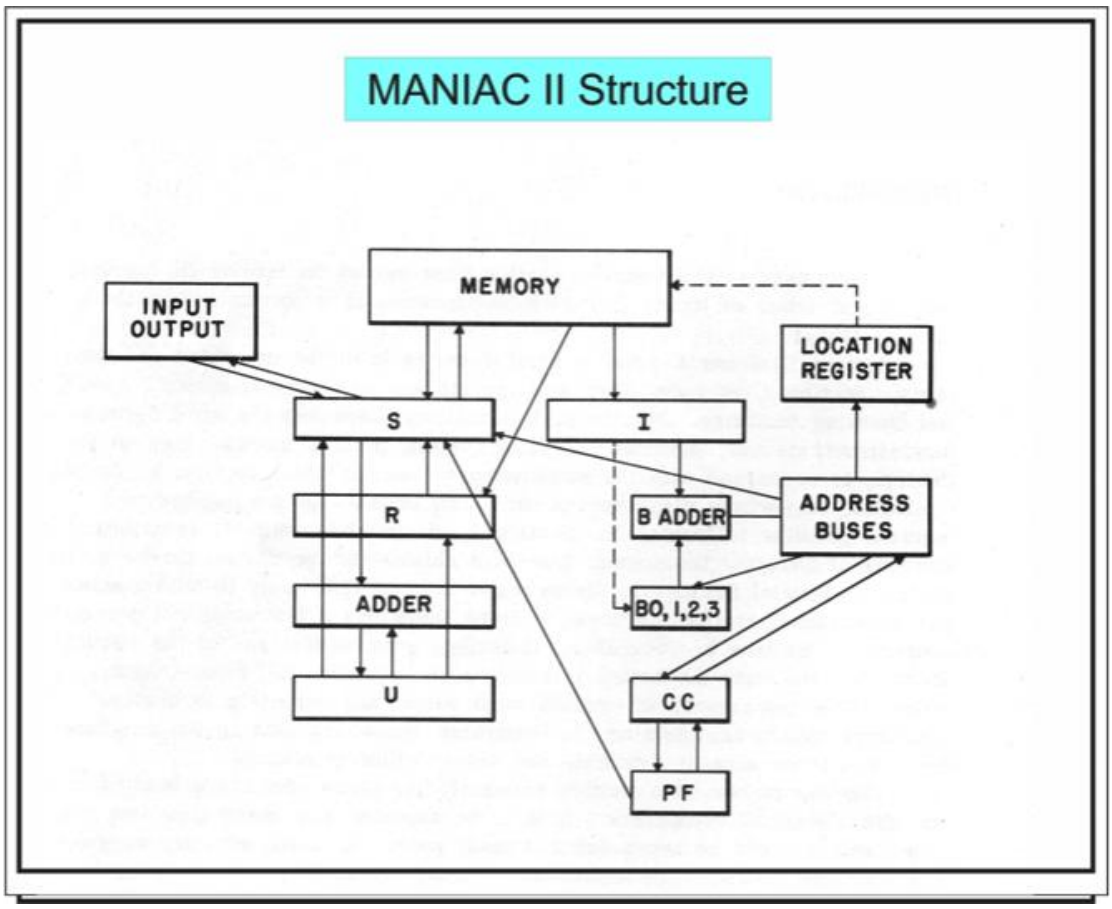
The following two diagrams show the step in design from MANIAC II to R1. In MANIAC II the S register latched data transmitted to and from the memory, while U (the Universal register) and R (Remainder) were used in arithmetic. The I register held the instruction addressed by CC. The B registers indexed the address found in I, using the B ADDER. The "pathfinder register" (PF) was copied from CC on subroutine Calls, and used as Return destination.

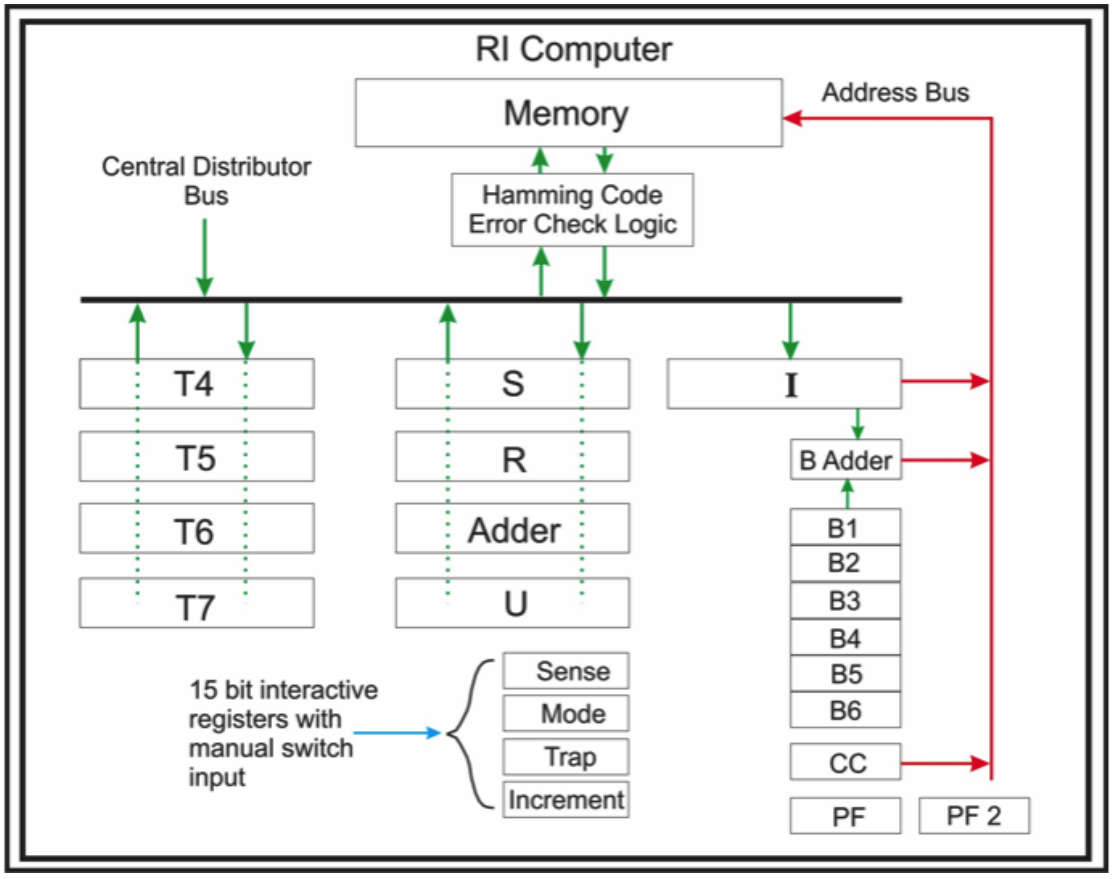
In "improving" MANIAC II a more complex structure emerged. The T registers provided four words of fast access (1μsec) memory. The number of index registers was increased to 8. The T registers have a length of 56 bits including two tag bits; the index registers have 15 bits, as do the interactive registers such as Sense, Mode, Trap, and Increment. Graham was keen to exploit memory tags to detect unexpected conditions in instruction or data, and provided a second pathfinder PF2 to allow entry to interpretive code without affecting the state of the program or the current content of PF.

The memory used RCA Radechon tubes, which were a great improvement over the simple CRTs used in earlier Williams Tube memories. In experiments Graham found that 8192 bits could be stored on each tube with a relatively low error rate, but decided to implement a Hamming Code system by adding seven more bits to each 56-bit word in memory. This allowed the R1 to continue long after aging tubes would make it impossible. As far as is known the R1 was the first (and last) computer to implement error correction in electrostatic memory.

The numerical word (now expanded from 40 to 54 bits) is treated as 1's complement floating point with 48-bit mantissa and 6-bit base-256 exponent. The large base is chosen to extend the numerical range, at the same time reducing the amount of processor time taken in aligning and renormalizing operands. (The resulting numerical range is within $10^{\pm 74}$, with 40-bit accuracy)

The Central Distributor Bus provided a common interface to peripheral devices: Line printer, punched paper tape I/O, magnetic tape and console I/O.

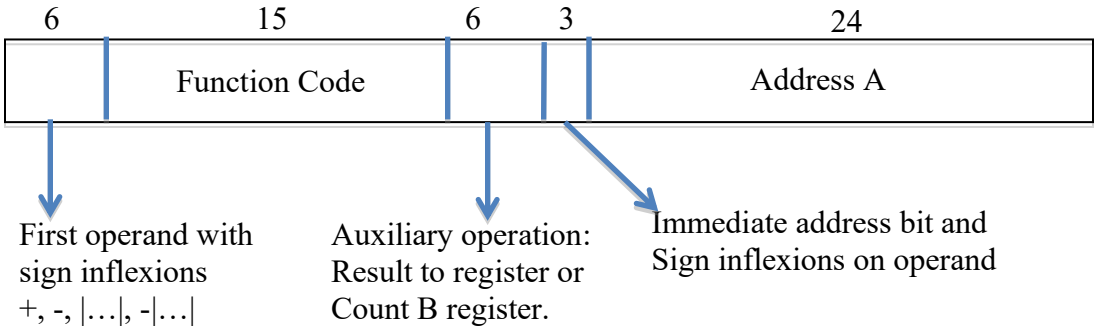

MANIAC II Structure
MEMORY
INPUT OUTPUT
LOCATION REGISTER
S
I
R
B ADDER
ADDRESS BUSES
ADDER
BO, I,2,3
U
CC
PF


RI Computer
Memory
Address Bus
Central Distributor Bus
Hamming Code Error Check Logic
T4
T5
T6
T7
S
R
Adder
U
I
B Adder
B1
B2
B3
B4
B5
B6
CC
PF
PF 2
15 bit interactive registers with manual switch input
Sense
Mode
Trap
Increment

The instruction register (I) contains 54 bits, partitioned into four fields which determine in sequence (1) input to the adder from fast registers; (2) input from memory, or immediate operand; (3) arithmetic or other function; (4) output of result to fast register, or increment/decrement one of the B registers. In each of (1) and (4) the data may be modified in transit.

To achieve numerical accuracy the memory delivered 56-bit words, both to the arithmetic unit and to the I-register. It would have been possible (as on Maniac II) to reduce this to 2 (or 4) simpler instructions, but Graham's decision to use the full instruction word had many advantages: the control counter (CC) and pathfinders (PF and PF2) did not have to address fractional words; rule (2) could be applied without repeated instruction fetches; and the auxiliary address fields could capture many of the register movements involved in expression evaluation (a point of view in stark contrast to advocacy of "top of stack" manipulation). Although complex, the sequence of events could be transparently expressed in assembly language.

The Function Code was decoded as five octal digits, one of which determined the class of operation (arithmetic, shift, jump, etc.) with the remaining four treated as micro-orders. As the logic was asynchronous there was scope to add complex functions for specific research purposes, notably the instructions to assist Salsburg's Monte Carlo calculations and versatile functions for A-D and D-A conversion used by J-C de Braemecker in geophysical analysis and G. Sitton in speech recognition. In this respect R1 was remarkably more flexible than contemporary microcoded systems using read-only memory.



R1 Computer: Instruction format

Of course, the picture would change when instruction caches came into play. As a forerunner of that, R1 had the ability to retain I and repeat the instruction until a prescribed condition was satisfied: common summation or searching operations could thus be initiated without further instruction access.

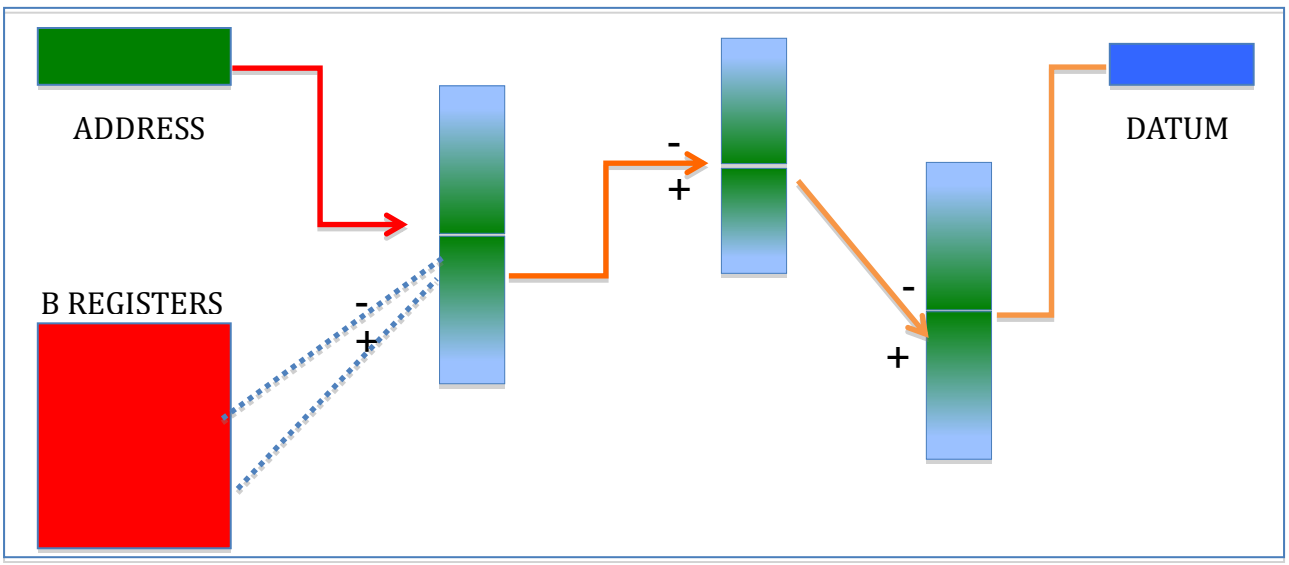
It was possible to preset traps to be sprung when certain tag combinations were recognised, either in processing data or instruction words, at preset points in the decoding cycle, under control of tri-state switches (ON, OFF, or program-controlled). (Somewhat confusingly, tags were later used to identify structural elements. In R1 bit A_1 served that purpose, and a similar method was used on the Burroughs B5000. In R2 it was planned to use an extended tag field to combine both uses.)

The addressing rule (2) is the crux of the scheme outlined by Kilpatrick:

An address A consists of 24 bits, divided into groups A_1, A_2 and A_3 of 1, 8 and 15 bits respectively. A location number is formed from an address in the following manner:
(a) The contents of up to 8 B-registers, selected by A_2, are *added* to the number represented by A_3.
(b) If A_I is zero, the 15-bit number resulting from (a) is the required location number or immediate operand; if A_1 is one, the result of (a) is used to select a word in memory from which a 24-bit address is taken, and the process (a) is repeated, and (b) reapplied.

Thus indirect addressing to any depth, with indexing at each stage, is permissible. The following diagram illustrates how a datum might be located indirectly through several blocks of "addresses".

It can be argued that in a static environment the relative address of DATUM is known at compile time and can be incorporated in code as a simple ADDRESS modifier. Before drawing conclusions it was of interest to measure the supposed "overhead" of the indirect addressing path. After the system and applications had matured special instrumentation was used to measure the incidence of indirect addressing. It was found that, relative to the total demands of instruction and data access, it contributed rather less than 10%, and that a relatively small cache memory would in theory minimize its impact.



The R1 hardware

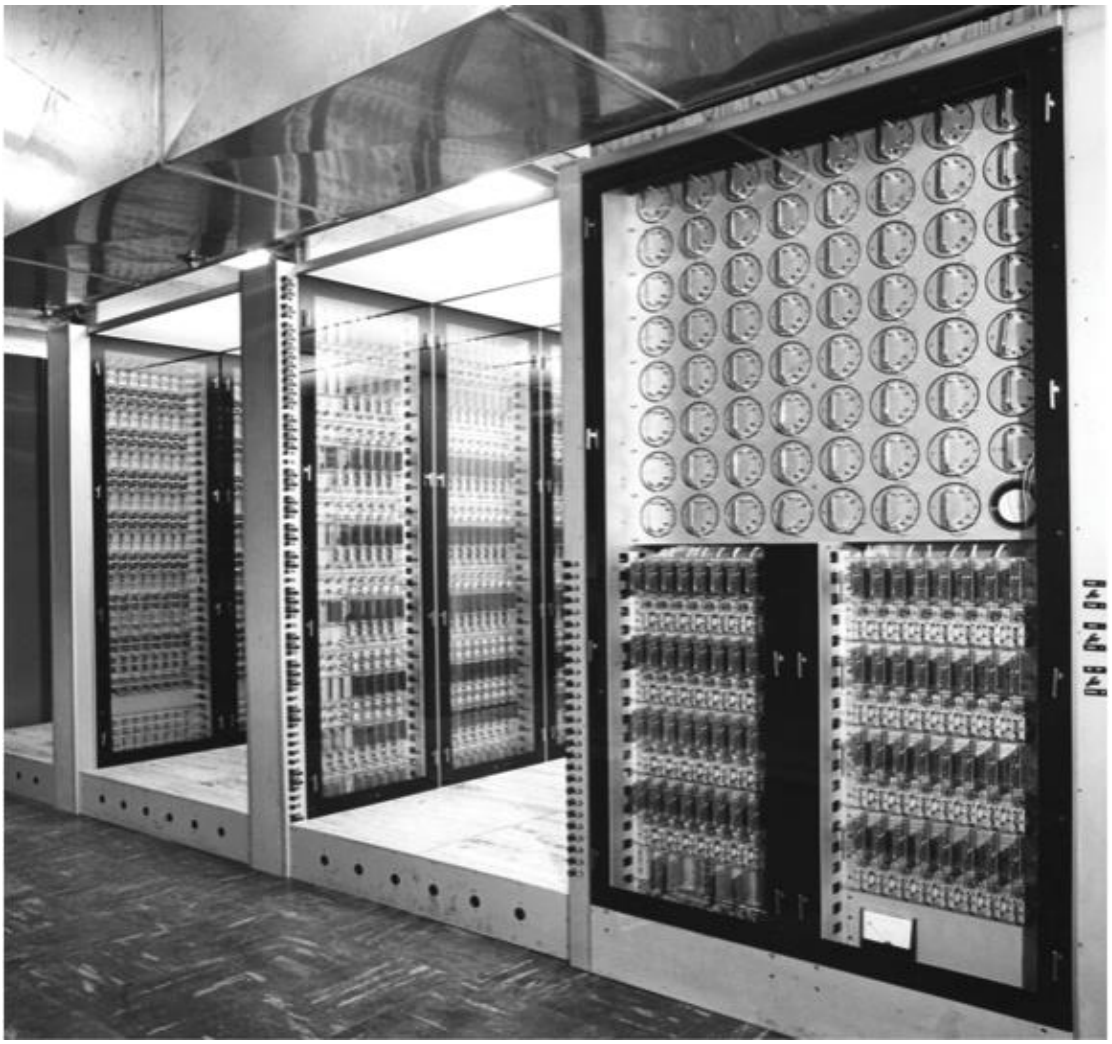
Photographs of R1 show an unusually elegant structure, far from the recycled component racks often seen in machines of that era, which was the result of custom engineering by a local supplier. Cooling ducts carried air under the floor, up the racks, and back across the ceiling to a chiller fan. The following photograph shows R1 in use in 1960 with Jane Jodeit (Griffin) at the console, Martin Graham onlooking.

Personal computing, 1960

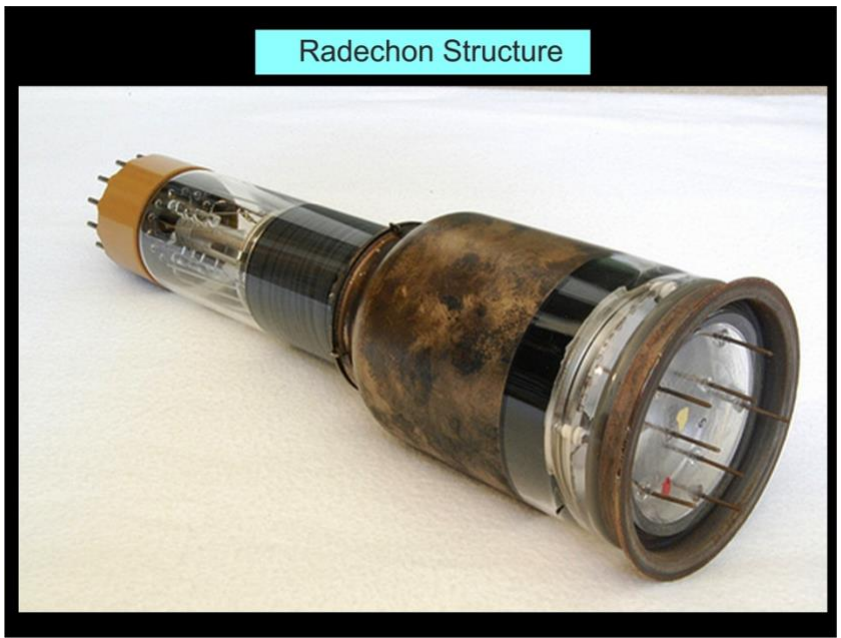
When above photograph was taken the overwhelming majority of program "tests" in major computer centers were submitted on card or tape to a postbox whence machine-operating staff would queue jobs and eventually – perhaps hours later - return the result to the user. The possibility of interacting directly with one's program was, to most, a luxury beyond the dreams of avarice for years to come. It was, however, normal practice on R1.

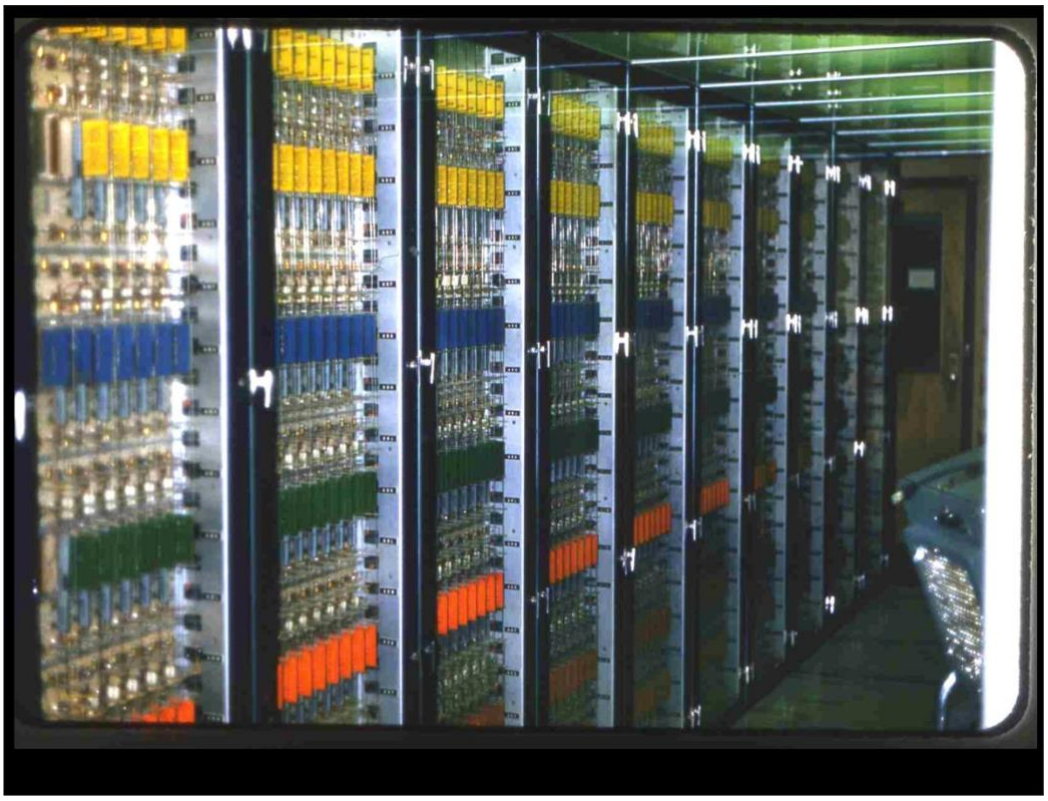
Note the absence of video display. On the left of the console are four sets of control switches that display the state of a program and may be used to vary trapping response or arithmetic style. The console displays as octal triads the content of the U register, and value of CC. An octal keypad could be used to insert a control word into U (see SPIREL).

A rotary handle not unlike an egg whisk could be used to advance the program at manual speed. Regrettably, it had no "reverse" mode.

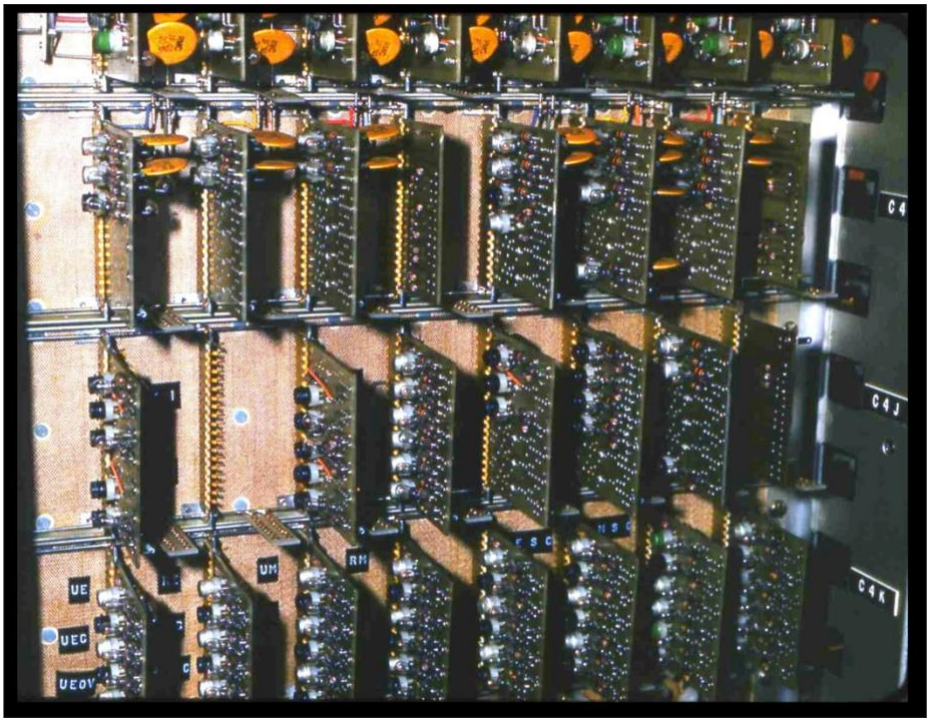


This view of the machine shows one memory bank and parts of the arithmetic and control sections. The 63 memory tubes were placed in accessible positions for ease of maintenance. Just as with modern dynamic RAM semiconductor ICs the charge decays rapidly and the data must be read and rewritten many times per second. Circuitry in the memory cage provided the continuous regeneration cycles and error correction.





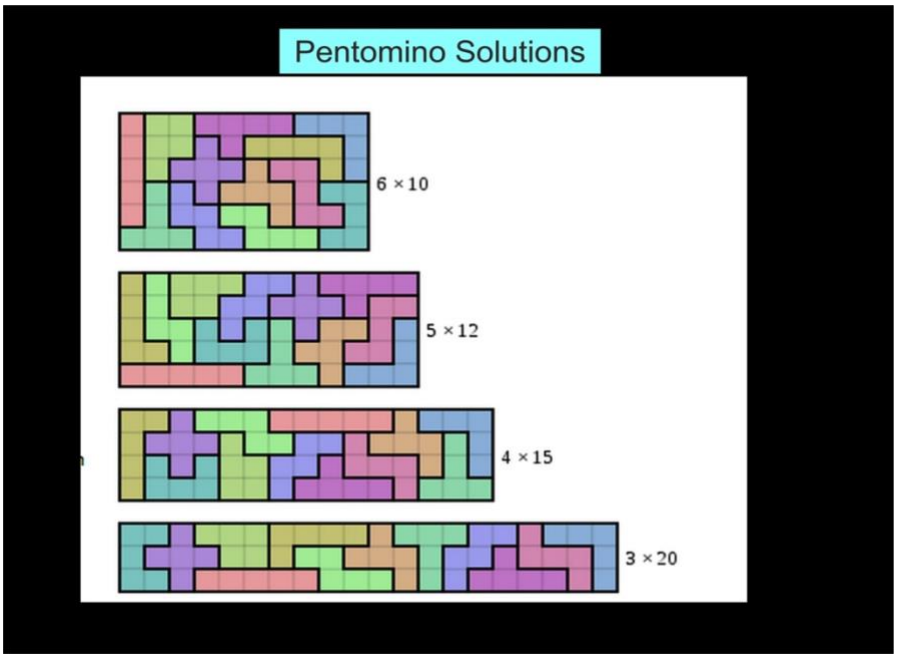
Colored circuit boards holding the diode logic circuits for the registers in the Arithmetic Unit racks.



Cards in the Control rack with transistor logic for decoding instructions.

References

Manual for the Rice Institute Computer (1958) CHM 102726209 contains initial specifications of peripheral devices, arithmetic representation, addressing and instructions. The Appendix gives details of circuit design. *Rice University Computer: Reference Manual* (1962) CHM 102726213 is more up-to-date, but gives no circuit details.

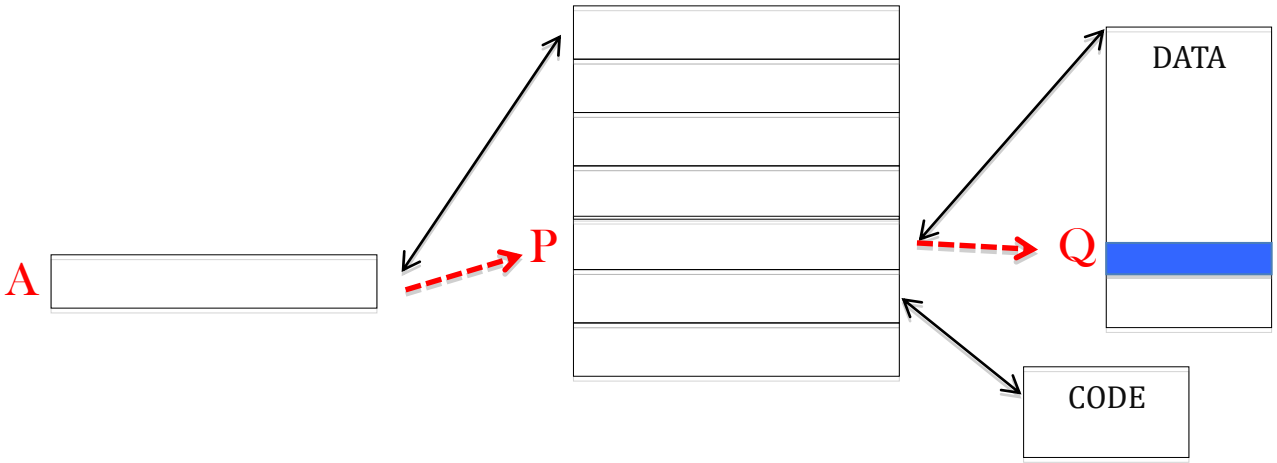


R1 sparked to life at the time when Martin Gardner began his famous mathematical recreation essays in Scientific American.

Kilpatrick programmed the solution to the Pentomino puzzle as the first test application of R1.

Operating system and languages in R1

The indirect addressing mechanism described by Kilpatrick might have been used in many ways, but it was seized upon as a means of representing commonly occurring data structures and gaining access to them efficiently. The key to this was a "mapping" element that allowed the user to access data without direct knowledge of memory location. Barton and I came away in September 1958 with that idea, which evolved into the *Descriptor* used in the Burroughs machines, and the *Codeword* used on R1. Both made use of the fact that the addresses used only half the codewords: the remainder was used to categorize the block referred to (as CODEWORD, DATA, or INSTRUCTION) and, most importantly, to give the LENGTH of the block. Note that each block referred back to its defining codeword.



The above diagram illustrates what might be called the "Standard Model" of access. Here A is a codeword addressing a block of codewords, within which item p addresses a block of data. To obtain the q'th data element one would compute offsets (usually by indexing) from the current starting addresses of each block, ending up with $A_{p,q}$.

In R1 the format for a codeword that labels a block beginning at address A_3 is:

N	i	a	Y	INDIRECT ADDRESS A

where `N` (15 bits) is the length of the block; `i` (12) is the index of the first word (usually zero in R1); `a=1` if block consists of codewords; `Y` (2 bits) determines the print format for data output; and A_3 is the address of the first element in the block, as described above. Thus in R1 the program used to select $A_{p,q}$. might take the form:

```
B1=P; B2=Q; LOAD *A
```

By 1961 the Standard Model was embedded in many programming language implementations (with varying addressing rules).

The importance of R1 was that it *defined* the user address space as a tree from the beginning, rather than a linear address space. This might be described as turning the memory "inside out". No other form of partitioning was needed. Sequences of data, codewords or instructions could be defined *at any stage of computation* and re-sized or re-shaped: the space assigned was automatically recovered when no longer in use. This demonstrated the first practical use of dynamically assigned segmented storage. It was an obvious step to extend the evaluation of codewords to define more "abstract" structures such as Null elements, Delayed definitions, peripheral channels, and so on.

Within the options provided by the tree structure there is a single block of data equating to the store of the IAS architecture, such as DATA in the above diagram. In that sense there is no loss of generality, though in fact instructions and data were usually kept separate in R1.

The almost contemporary use of Descriptors in the Burroughs systems (B5000 and successors) achieved many of the same advantages as codewords on R1. Store blocks were copied to backing store (drum or disk) on the B5000 in order to free space, whilst R1 compacted them in memory. It was a bold move on Barton's part to orient the architecture towards the conventions of Algol 60, and not to provide a flexible set of central registers. At that point the approaches to design of R1 and Burroughs diverged: Barton elected to make the machine fit the language, whereas R1 defined a tree-structured memory, then looked for ways of exploiting it in language design.

That resulted in four main software components, which exploited the natural tendency of the system to deal with instruction, codeword, and data segments as *objects* that could be combined dynamically to form executable programs: the AP1 assembler, the SPIREL program manager, the GENIE compiler, and the STEX store manager. They came into operation in 1961, but development continued until 1967 under the guidance of Jane Jodeit, assisted by Mary Shaw.

It is worth recalling that despite increases in memory size compilation of programs frequently entailed laborious use of intermediate storage, with the result that correcting substantial programs might take immoderate amounts of machine time – not to mention the physical problems encountered in handling large punched card decks or (as in R1) paper tape reels. In addressing such problems FORTRAN 2 introduced the SUBROUTINE and COMMON area. In R1 it was possible to go further and treat an entire program as an assembly of procedures and data in which individual components could be modified and re-tested without needing to recompile the entire source text.

Barton relocated to Burroughs in 1959

ASSEMBLER

AP1 provided symbolic versions of instruction steps that translated into the fields described above. For example:

```
-T4  FMP  A+B1, B1-1
```

negates `T4 to` the accumulator (`U`), multiplies by the content of address `A+B1`, then decrements `B1`. It might be followed by:

```
B1 IF (NZE) TRA LOCX, U➔T5
```

The use of the control counter (CC) as an address modifier enabled program segments to be relocated in memory without alteration.

The analogue of a COMMON region was provided by a Value Table, which was a set of codewords and constants forming the top level of the program tree. Any reference to external objects was directed through VT, which contained the initial BIOS, mathematical library, and compilers.

SPIREL

The SPIREL program manager enabled individual components of an execution run to be loaded via VT initially, or manipulated at intermediate stages, and finally to be output as results or diagnostic information. It was activated by a set of *control words* that could be read from paper tape, or passed internally as a parameter, or from the console keyboard. Although that was about the limit of interactive use it was an advance on other diagnostic aids.

GENIE

There was no pressure on R1 to provide a variant of some "standard" language or operating regime. GENIE was innovative in several respects. By fixing a solenoid to the carriage of the Friden Flexowriter Graham allowed printing of sub- and super-script text that approximated to mathematical notation. The form of imperative commands was elaborated so that the principal assignment could be followed by a number of auxiliary definitions, as in:

```
x = 2a^3 + 3ab - b, a = 2gt, b = |y1 - y3|
```

in which the compiler would reorder the calculation to evaluate `a,b` then `x,` making best use of fast registers. (Note that lexical rules allowed multiplication signs to be omitted in many expressions.) Another innovation was to cater for vectors and matrices in

arithmetic expressions, with single, double and complex components: Genie was probably the first general-purpose language to incorporate such intrinsic array operations.

In any system allowing late binding of program components a method of defining free variables, i.e. those not defined within the individual segments, must be implemented. In R1 the program tree was extended by a symbol table (ST) giving for each an index in the value table (VT): in the coded segment references to any free variables were trapped when first encountered and replaced by indirect references to VT.

STEX

To keep track of store demands in GENIE it was necessary to activate the store management module, STEX. The principle of operation was to compact store assignment as necessary, at the same time adjusting codewords to reflect the change. For that purpose each segment contained a back-reference to its defining codeword in the program tree. In any such system problems arise when intermediate calculations have resulted in addresses (in registers or stack, for example) that also need to be updated. In R1 invocation of STEX was avoided in such situations (by following rules of program writing). The problem did not "go away" until a strict means of marking addresses was proposed.

References

Programming memoranda. Rice Institute Computer Project CHM 102726210 records the evolution of the programming ideas from 1959-1963. *Programming systems. Complete manual set for the Rice R1 computer* CHM 102726215 (1968) is the most comprehensive definition of Assembly languages, SPIREL, GENIE, and library functions. STEX is best described in Jane G. Jodeit *Storage organization in programming systems* CHM 102726222. J.K. Iliffe and Jane G. Jodeit, *A dynamic storage allocation scheme*, Computer Journal 5 (1962) p 200. CHM 102726218

The R2 Architecture

Although the problems of address recognition noted above could be "worked round" it was recognised that stricter rules had to be applied in a multiprogramming environment. At the same time it was clear that the ten-year-old hardware of R1 was in need of upgrading. In 1968 a project to build a new computer was begun. It was decided that the "R2" computer would embody many of the ideas of the Basic Language Machine, which used tag bits to classify and identify different types of data and address in memory.

R2 went beyond the BLM in a number of respects. Using the existing 1 microsecond core memory of R1 each 64-bit word provided 4 tag bits (as required for address and data type identification) and two "software tags" (as used on R1). The 54-bit numeric representation of R1 was retained. A single parity bit replaced the Hamming code. Double precision and complex data tags were recognised by hardware functions.

The address format defined the start and extent of a memory block, but also specified in the codeword the *index* of the first component, which was used in computing the word offset into the array. An address could either prescribe the type of data to which it referred or let the target be self-identifying. Modification and Limitation operators allowed formation of addresses of reduced range, which was the most important innovation in that it allowed secure access to portions of data segments.

Stack structure was controlled so that the hierarchy of procedure calls and task activations could be traced by system and diagnostic routines. In that respect it was similar to the Burroughs B6700, but without reference to particular language conventions.

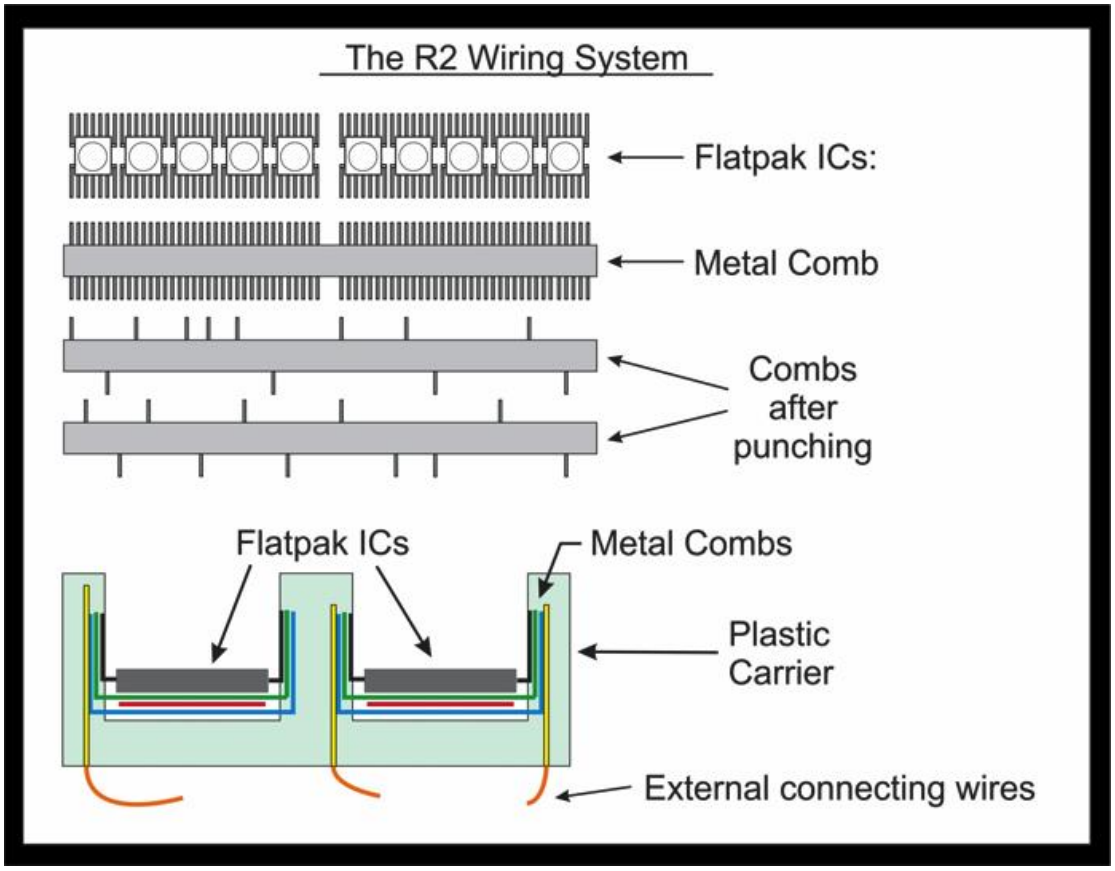
The use of the two "soft" tag bits followed R1: if a tag combination of interest arose (i.e. 01, 10 or 11) and a corresponding Mode switch was set (by program or on the console), a trap to an interpretive routine occurred. That could be used to trace instructions or to handle unusual categories of data. A further data classification was provided by a hard tag (1000), intended to be used to represent "Null" quantities outside the number representations defined for the computer, such as Underflow or Infinity. Here again, a trap would direct control to interpretive software.

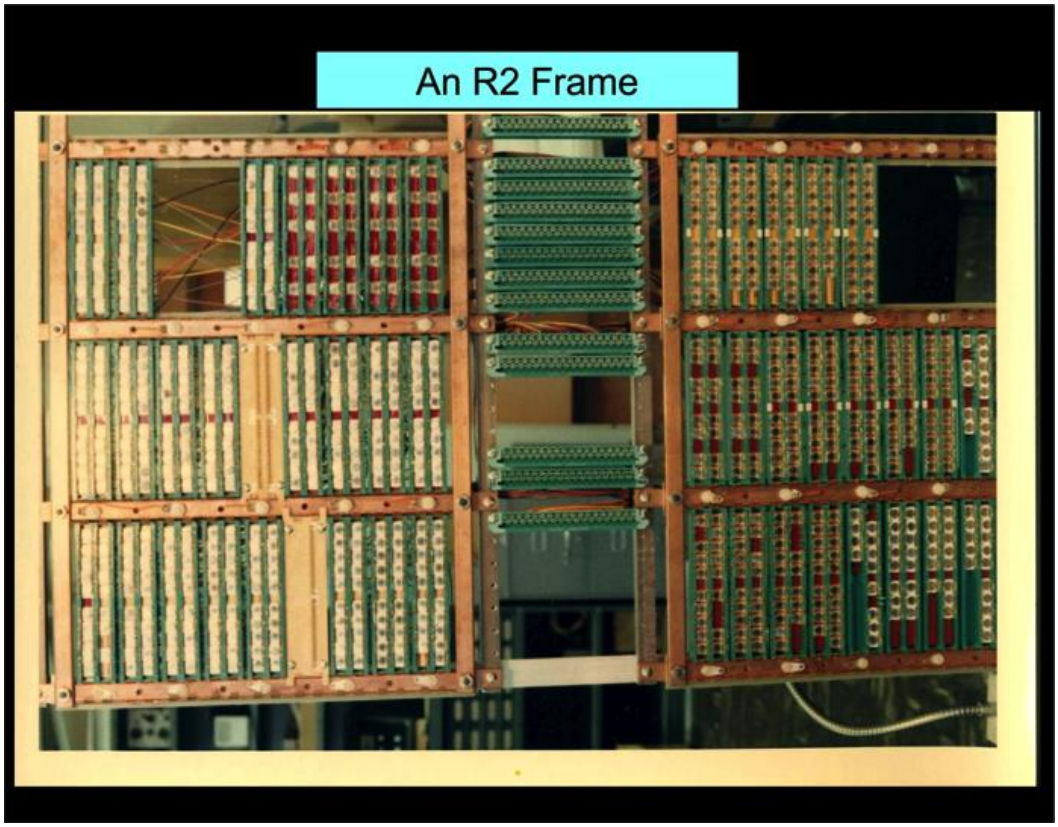
The R2 Hardware

Integrated circuit technology provided only simple gate functions but it was decided to use high-speed RCA ECL gates with 5-nanosecond propagation delay at the cost of a higher power requirement. A floating-point addition time of 50-200 nanoseconds, multiply 3 microseconds, and address calculation 150-200 nanoseconds were targeted.

A system of plastic carrier blocks holding 10 integrated circuit packages in a trough used metal combs to interconnect the 10 ICs. Wires from the back interconnected the carrier blocks. Ernest Sibert, a graduate student (of J. Alan Robinson) in philosophy, wrote a program for R1 that took in IC connection information extracted from the logic drawings as Sigsby Rusk developed them. The program then determined which ICs to group together in each carrier block to maximize the connections by the combs and thus minimize the amount of wiring on the back. Sibert's program then calculated which teeth to remove from the combs in each carrier and drove a punch interfaced to R1 to generate the combs. Each carrier had 3 or 4 layers of combs separated by an insulator to accomplish connections to the ICs.

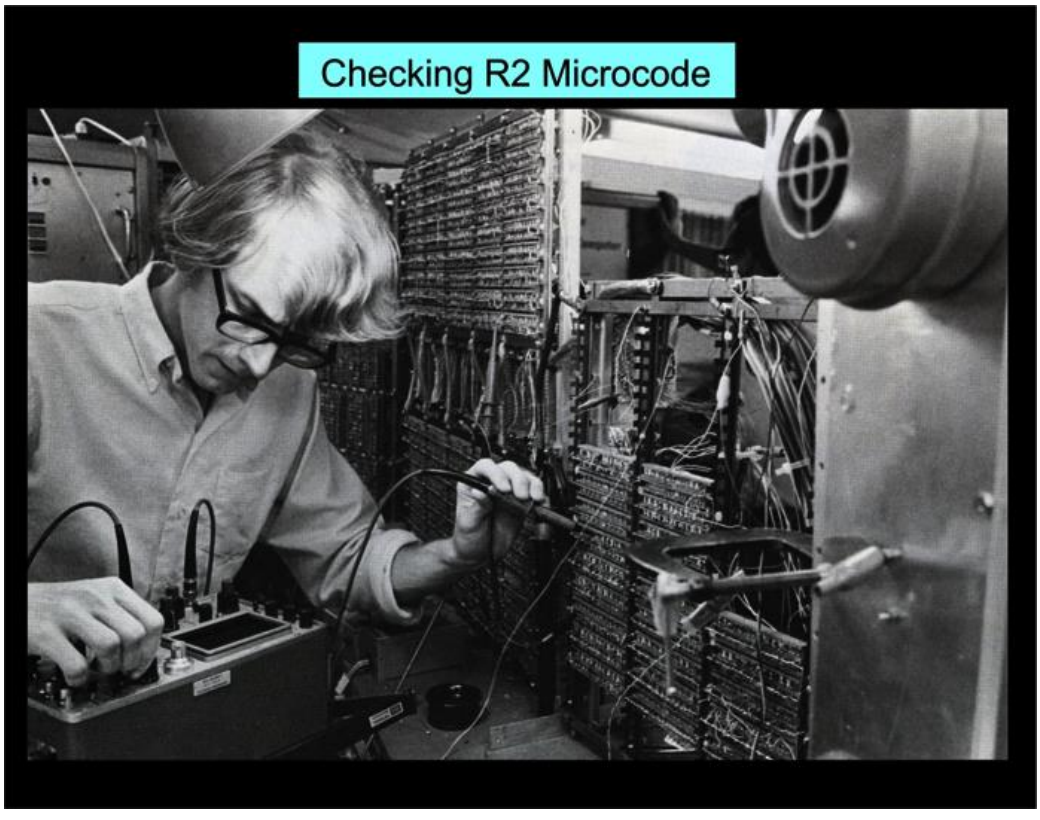
The diagram shows how the selected teeth were left on the comb by the punch then stacked in the carrier blocks with a layer of red insulating film between combs. The IC leads, comb teeth, and inserted wires were then soldered together.





One of the frames of the R2, showing the heavy copper bus bars which distributed 5 volts and ground power to the ICs as well as serving as the structural framework. The column of green sockets in the center was there to accept cables to connect to other parts of the machine.

Graduate student John Doerr checking out the initial microcode circuits, implemented as a wired network of logic gates which were activated in sequence to carry out the desired operations.



Without doubt R2 would have been a worthy addition to the Rice Project, opening up new avenues of research. Sadly, after the considerable ingenuity and effort spent in design it ran into insurmountable mechanical difficulties. Although (as can be seen from the pictures above) hardware was near completion, it became clear that funds were not forthcoming to complete construction, which was abandoned in 1976.

References

Rice Computer-2 general specifications CHM 102726240

Edward A. Feustel. "*The Rice Research Computer (R-2) -- a tagged architecture*", *AFIPS Proceedings*, Vol. 40, May 1972, pages 369-377. PDF at feustel.us

Edward A. Feustel. "*On the advantages of tagged architectures*", *IEEE Transactions on Computers*, Vol. 22, July 1972, pages 644-652. PDF at feustel.us

Basic language Machine: John K. Iliffe. "*Elements of BLM*". November 7, 1968, Catalog number 102726223

The Legacy of the Project

The obvious legacy of R1 was the widespread adoption of the Standard Model in commercial language implementations, mapping into a "flat" storage region. In such schemes the index supplied might be checked against declared dimension, and the codeword type might be examined. If strictly applied a high level of security could be achieved. However, strict application was uncommon, and in early timesharing machines individual programs were confined to regions defined by "Base" and "Limit" registers, under control of a Supervisor program, running with Supervisor privileges. The stumbling block was always the need to resort to assembly code to achieve performance goals. The emergence of C stated the problem in another way, with its use of *pointers.*

By the mid-60's software overruns, particularly in operating systems, were of such concern that efforts to restrain error propagation, which essentially meant controlling the formation and use of addresses, were under way. And if intra-program protection could be achieved inter-program protection followed. In fact the solution to the problem was already in the Standard Model.

Both R1 and the B5000, whose origins could be traced back to the meeting at Rice in September 1958, were susceptible to failure because of inability to discriminate safely between addresses and data in the central processor. (Their exact nature is difficult to pin down at this distance: the reason for introducing explicit tags in the B6700 is not explained in current literature). The problem was solved in similar ways: by adding tag bits to some or all memory words and copying them securely through all data paths. The redesigns, which started in 1963, resulted in the Burroughs B6700 family and Ferranti's (later ICL's) experimental Basic Language Machine (BLM). The two new architectures were demonstrated in 1968 (or thereabouts – it is difficult to be more precise).

Barton and Iliffe at Catalina Island in 1969, following delivery of B6700

In 1969 the BLM was rejected as a product proposal by ICL. Nor was the Burroughs solution followed elsewhere, perhaps because it was focused on Algol, or on stack manipulation, but that is pure speculation. In that sense there is no "legacy" to report.

Although the use of paged addressing had little impact on logical management of store, it was seen that an elaboration of the page tables would assist partitioning on "need-to-know" principles. In the GE 645 (1965) the necessary checks were made on every store access by consulting a *segment* table (Multics) closely resembling a set of codewords. In 1966 Dennis and Van Horn presented a generalised approach to segment management that built on the role of Descriptors and Codewords: their term *Capability* gained general acceptance, although wide variations in usage occurred. By the 1970's the idea of accessing memory though a table defining the context was widespread.

Any processor architecture targeting high security goals must satisfy certain practical aims to be successful: it must be foolproof, cheap, fast, flexible and easy to use. The above points were self-evident from the outset. R1 had all except the first and (by accepting the security risks) enabled the advantages of codeword-based structure to be demonstrated. A number of experimental and commercial designs explored this area, though it would be hard to find any that was cheap, fast and flexible. By 1982 it was reported (Wikipedia): "iAPX432 capability architecture had now started to be regarded more as an implementation overhead than as the simplifying support it was intended to be". Experience seemed to show (incorrectly, as it happened) that however much microprogram or (in later years) semiconductor area was devoted to the problem the practical goals would still be out of reach.

A note on the Basic Language Machine

The most important feature of the BLM was that it recognised *delimited* pointers. A codeword was *evaluated* to give a pointer, but at the same time the *limit* value was copied. Subsequently it could be modified to refer to any proper subsequence of the original segment. Moreover, it was impossible by chance manipulation of the pointer (such as was possible in C) to refer to any other data that happened to be in the same execution environment. This form of controlled address formation was sufficient to establish any required partition of program space - program environment, process, procedure, parameter list, and so on - with far greater precision than seen elsewhere. Apart from the BLM only R2 would have that ability.

The BLM is briefly summarized in J.K. Iliffe, Basic Machine Principles Macdonald-Elsevier 1968, 2nd ed. 1972. A concise outline of BLM structure is given in Elements of BLM (1968) CHM 102726224.

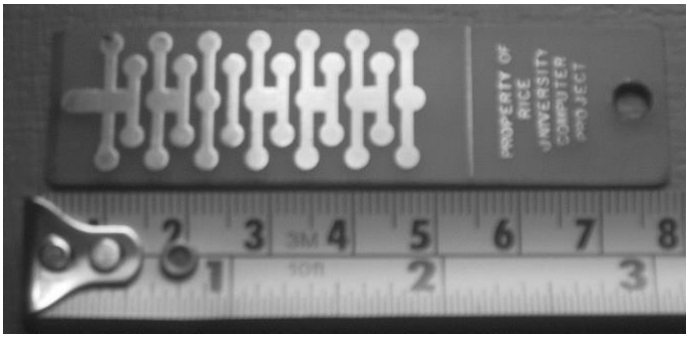
Back to the future

Computer architecture as now understood can be traced to the original formulation of EDVAC at the Moore School, and von Neumann's exposition thereof: although many advances in engineering have been witnessed the basic structure of individual programs has changed very little.

It has been shown that many advantages accrue from the use of tagged memory: compilers, system and library procedures are precisely defined and protected, errors are quickly detected and reported, foreign imports (such as channel controllers) can be strictly confined to "safe" areas, and so on.

In addition, there are indications that, apart from solving issues of protection, other design areas would benefit from tag-dependent operations that are not available in conventional processors. They include object management, language design, compiler technology, and operating characteristics, which are best approached by restarting from first principles.

It might have been hoped that systems would develop to take advantage of the opportunities opened up by this fundamental shift. In fact that has happened only rarely, and never more freely than on the original R1. As Perlis remarked: "Adapting old programs for new machines usually means adapting new machines to behave like old ones". So the opportunity was lost.



A computing antique (1960): a "log-in" key used to gain access to R1

References

The B6700 has been described many times in open literature.

Some capability architectures are described in H. M. Levy, *Capability-Based Computer Architecture* 1984, http://homes.cs.washington.edu/~levy/capabook/. See also J.K. Iliffe, *Advanced Computer Design,* Part 3: Abstraction. Prentice-Hall 1982.