The Myth of the Harvard Architecture[1]

Richard Pawson

Introduction

The term 'Harvard architecture' appears in many college level textbooks on computer architecture. For example, in Computer Architecture: A Quantitative Approach, Hennessy and Patterson say this:

> '*The* [Harvard] *Mark-III and Mark-IV were being built after the first stored-program machines. Because they had separate memories for instructions and data, the machines were regarded as reactionary by the advocates of stored-program computers. The term* ***Harvard architecture*** *was coined to describe this type of machine. Though clearly different from the original sense, this term is used today to apply to machines with a single main memory but with separate instruction and data caches.*' [21]

This is a sound explanation that avoids mistakes made in some other sources, cited later. As it suggests, the term 'Harvard architecture' has more than one meaning. The term itself was not coined until the 1970s in the context of designing the first microcontroller (complete computing device on a single chip) and it was only retrospectively applied to the 'Harvard machines' – designed by or for the Harvard Computing Laboratory (HCL), under the leadership of Howard Aiken. Later, it was applied again to RISC processors that cached instructions and data separately.

Every mainstream computer designed since 1945 stores instructions and data separately at some point – ultimately in different registers within the processor. Within the historical contexts listed above, instructions and data were separated at some additional level, but both the nature and the motivation of the further separation differed in each case. Encompassing these separate developments in a single term encourages misleading generalisations such as this[36] :

> '*the Harvard architecture … allows the CPU to access instruction and data simultaneously*'. [37]

That was true of the two later developments (microcontrollers and RISC processors), but not for any Harvard machine.

In *Essentials of Computer Architecture* [10] , Comer positions the 'Harvard architecture' as an alternative to the 'von Neumann architecture'. While there is much dispute both about the exact scope and definition of the latter, and how much of it is legitimately attributed to John von Neumann, few dispute that the primary source, *The First Draft of a Report on the EDVAC* [40] [40] (the 'First Draft') embodies – in today's terms – an 'architecture'. [2] No definition of the 'Harvard architecture' provides an equivalent basis to the First Draft for designing a computer. The various historical developments labelled as 'Harvard architecture' have each resulted in a single design

[1] This paper appears in: IEEE Annals of the History of Computing Print ISSN: 1058-6180 Online ISSN: 1934-1547 Digital Object Identifier: 10.1109/MAHC.2022.3175612
https://ieeexplore.ieee.org/document/9779481

choice that may be adopted *within* the von Neumann architecture. The Harvard Mark III and IV adopted all the design principles of the latter bar two: representing numbers in binary, and treating memory as a flexible resource for storing all forms of data and instructions. And while most modern machines still owe much to the First Draft, none since the 1950s has followed it in every respect. Only the most recent interpretation of 'Harvard architecture' – storing instructions and data in a common memory but caching them separately within the processor – is applicable to modern general-purpose computing.

This terminology might be sloppy, but what relevance does it have to the formal history of computing? It has encouraged a myth that Aiken invented an architecture superior to that commonly attributed to von Neumann, but that this superiority was not recognised until many decades later. Two individuals who worked with Aiken at the HCL, Peter Calingaert and Grace Hopper, have implied this:

> *Today's prevailing wisdom praises the separate storage of instructions and data as the 'Harvard architecture'.* [4]

> *'Aiken always insisted that the data and the program must be stored independently. We lost that concept for a while when people came along and said "Oh, we want to store the program in the same memory as numbers so that we can alter the program." … In my opinion, that put more bugs in programs than anything else ever did.'* [24]

Others have amplified these assertions, for example:

> *'In hindsight, with knowledge of the proliferation of von Neumann architecture-enabled security threats, there is reason to wonder whether the entire information technology industry would not have been vastly better off had there been early agreement to embrace the Harvard architecture and its complete separation of code and data memory regions, despite the costs involved.'* [29] [29]

In this myth, the Harvard and von Neumann architectures are cast in roles broadly equivalent to Betamax and VHS in the story of video-tape format competition [11] , the populist version of which holds that the earlier-and-superior technology – Betamax – was eclipsed by the later-and-inferior, but better-marketed, technology – VHS.

Howard Aiken is rightly recognised as a pioneer in automated computing: his vision and drive led to the creation of one of the first large-scale automated computers, completed and applied to real work before the end of WWII. He also initiated the first comprehensive postgraduate programme in what we today call 'computer science'. It is not an intent of this article to diminish Aiken's reputation for these achievements one iota, simply to put paid to the persistent myth of Aiken the architectural prophet without honour in his own time. His concern about program immutability was not justified even at the time, and his pattern of storing instructions in non-writable memory would prove unworkable with the later introduction of operating systems.

Since the 'Harvard architecture' has been contrasted to the 'von Neumann architecture', we start by looking at the meaning of the latter term, identifying ten key design principles that it embodies. The Harvard machines are then evaluated against the principles. The article then explores the emergence of the term 'Harvard architecture' during the invention of the microcontroller, and its subsequent re-interpretation in the context of RISC microprocessors, before concluding with the relevance of this discussion to the present day.

The 'von Neumann architecture'

The validity of the term 'von Neumann architecture' has been widely questioned [17] . It is being used here only because the term 'Harvard architecture' is often explicitly contrasted to it. There is also no consensus on the exact meaning of the former, except that it emanates from work initiated by the team that designed the ENIAC plus new contributors including John von Neumann, to design a successor machine that – even before ENIAC was running – would use new technologies and new design principles to overcome what were already perceived as limitations of the ENIAC's design. Although, even before any new machine had been built to use the new ideas, the ENIAC itself was substantially modified to adopt many of them [18] [18]

This article re-uses a framework set out by Haigh, Priestley, and Cope in [17] [17] , which groups design principles drawn from the First Draft under three headings: the 'EDVAC Hardware paradigm', the 'von Neumann architectural paradigm', and the 'Modern Programming Paradigm'. The number and wording of the principles listed under those three headings, below, vary slightly from their version, but principally for brevity, not to favour any argument being advanced herein. In designing a computing machine, each of the principles listed below *could* be adopted independently of the others – indeed the authors of that framework have shown that while some were evident in the first discussions of the EDVAC, others took time to be agreed. For later reference within this article, the principles have been numbered: **#1** to **#10**.

The EDVAC Hardware Paradigm

Large addressable read/write memory (#1). Initially enabled by Eckert's design for a 'delay line store' (derived from the Mercury delay lines used in radar systems) and stretched by von Neumann's vision that the computer should be applied to different kinds of mathematical modelling that were far more data intensive, the planned EDVAC design would advance the requirement from tens of numbers stored in working (writable) memory, to thousands.

Binary number representation (#2). Where the ENIAC and other early computing devices had stored and processed numbers in decimal format, the natural representation for the *users* of the machine, the EDVAC used binary, which, to achieve the same precision, was about 25% more efficient in storage, and made for simpler arithmetical circuits. This gain was felt to outweigh the cost of the additional circuits and/or software needed to convert from/to decimal for input and output.

The von Neuman Architecture Paradigm

Separate organs for storage, arithmetic, and control (#3). The ENIAC had been built around 20 'accumulators', each combining working storage (for one number) with arithmetic and control circuits. Scaling up the memory by at least two orders of magnitude meant that it would not be feasible to repeat this pattern.

Special purpose registers (#4). Inside the processor there would be a few fast storage units, known as registers, each with a dedicated purpose – such as the Accumulator, Instruction Register, and Program Counter – hardwired to different arithmetic and/or control circuits.

Program executed from fast memory (#5). Program instructions should be held in numbered memory locations, randomly accessible at high speed. (Note that this may be adopted independently from the next principle).

Fully interchangeable memory (#6). von Neumann's wording of this principle in the First Draft is surprisingly tentative, stating that it is:

> '…*tempting to treat the entire memory as one organ, and to have its parts even as interchangeable as possible.'* [40]

In the paragraphs preceding this comment he had been talking about the multiple differing needs for data storage in a program run, so he was making a tentative case for treating *all* memory units as interchangeable, not just the storage of data and instructions.

Program loadable from external media (#7). von Neumann foresaw that future computers would not be applied for days at a time to a single problem, but would tackle many different problems within one day. Therefore, it should be possible to read the program rapidly into memory from some external medium. This principle is not explicit in the (incomplete) First Draft, which does not cover Input/Output, but was clearly a part of the thinking.

The Modern Programming Paradigm

Sequential atomic instructions (#8). Programming the ENIAC had involved physically configuring the operation of each accumulator and their interconnection, though plugboards and switches. Thought had been given to allowing this physical configuration to be specified 'in software' (to use a modern term), but by the time of the First Draft, the concept of a program (as we now call it) had changed to mean the specification of separate, atomic, instructions to be processed in a sequence.

Automated jumps (#9). This was probably meant to contrast with paper-tape run machines which had no automated branching.

Instructions operating on variable addresses (#10). The First Draft indicated that it should be possible to vary the address part of an instruction. One need for that was to apply the same code to different data elements successively, which von Neumann demonstrated in his first program [27] [27] It would also support subroutines, which upon completion needed to return execution to the instruction after the one that had called the subroutine [41] These requirements can be realised in different ways: by modifying the address portion of an instruction stored in the program memory (the initial idea); by modifying a copy of the instruction held in a register (von Neumann himself later proposed this); or by defining an instruction that reads its address from one or more specialised registers. The third option – which includes what is now known as indexed or indirect addressing – was implemented by others even before the first machine to adopt the EDVAC blueprint, for example in the Manchester Mark I [28] It is misleading to equate this principle with the idea of 'self-modifying code', not only because of the alternative ways in which it could be implemented, but because even the initial idea did not permit instructions to overwrite other instructions – only the address portion of those instructions.

The Harvard machines

We can now evaluate the design of the Harvard machines against the 10 principles. The results are summarised in Table 1, with more details to follow for each machine. (For comparison, the table also shows the original ENIAC, the modified ENIAC, and the EDSAC – the first machine explicitly designed to follow the proposed EDVAC design.)

Design principle	Harvard Mark I 1944	Harvard Mark II 1948	Harvard Mark III 1950	Harvard Mark IV 1952	Original ENIAC 1945	Modified ENIAC 1948	EDSAC 1949
#1 Large addressable read/write memory	No	No	Yes	Yes	No	Yes	Yes
#2 Binary number representation	No	No	No	No	No	No	Yes
#3 Separate storage, arithmetic, and control	No	Yes	Yes	Yes	No	Yes	Yes
#4 Special purpose registers	No	No	Yes	Yes	No	Yes	Yes
#5 Program executed from fast memory	No	No	Yes	Yes	No	Yes	Yes
#6 Fully interchangeable memory	No	No	No	No	No	*	Yes
#7 Program loadable from external media	No	No	Yes	Yes	No	No	Yes
#8 Sequential atomic instructions	Yes	Yes	Yes	Yes	No	Yes	Yes
#9 Automated jumps	No	No	Yes	Yes	No	Yes	Yes
#10 Instructions operating on variable addresses	No	No	Yes	Yes	No	Yes	Yes

Table 1 The Harvard machines evaluated against 10 design principles evident in the First Draft. Three other machines are shown for comparison. Each is listed with its year of completion.
** Banks of rotary switches – originally intended as function tables – could now store instructions or constant data. So, the read-only memory was interchangeable, but read/write memory was for data only.*

Mark I and Mark II

The design of the Harvard Mark I [1] [14] [5] originally known as the Automatic Sequence Controlled Calculator (ASCC), preceded the conception of the EDVAC and the ENIAC. Nonetheless, the Mark I can be said to have anticipated principle **#8**, because it was programmed by defining a sequence of instructions, captured on 24-channel paper tape. Grace Hopper would later argue in [14] that

> *'…because it was sequentially programmed … Mark I clearly resembled more closely* [than the ENIAC] *what we have today.'*

Instructions were read and executed one at a time in strict sequence; there were no machine-controlled jumps, conditional or unconditional, though by 1946 a limited form of what we would today call a 'conditional expression' had been added. Subroutines involved halting the machine at a defined point and manually repositioning or switching paper tapes.

The core of the machine comprised 72 rotating mechanical counters, each representing a 23-digit signed decimal number, and each capable of performing addition and subtraction. Centralised relay-based logic circuits added multiplication and division, as well as the ability to interpolate between successive values in a 'function table' – either one built into the machine or specified as arbitrary function tables on three 24-channel paper tape readers.

The Mark II [15] [6] , [25] broadly followed the architecture of the Mark I, replacing the mechanical rotating counters with 48 faster-operating registers built from relays, but stripped of their addition/subtraction functionality. The latter was now implemented in a centralised relay-based logic unit. So, whether consciously or not, the Mark II had adopted principle **#3**.

Mark III and Mark IV

In the Mark III [16] [30] [30] all logic was implemented using vacuum tube electronics, and working data was stored on eight magnetic drums: two 'fast' and six 'slow'. Only the fast ones were directly addressable by the instruction logic: data could be bulk transferred between the fast and slow drums. (While this article argues that Aiken's splitting of instruction and data stores was of little lasting significance, he is arguably not given enough credit for pioneering the splitting of the data store into what we now call 'primary' and 'secondary' online storage.)

The later Mark IV replaced the 'fast' drums with what was referred to at the time as 'magnetic delay lines', but could perhaps be more clearly described as 'magnetic core shift registers': a solid-state

electromagnetic form of memory, working somewhat like the later 'core memory', but with serial rather than random access.

In both the Mark III and IV, instructions were also held on magnetic drum memory, thus permitting fast jumps between instructions. However, in both machines, the storage of instructions and data was physically separated – each element of memory storage was permanently dedicated either to data or to instructions. Instruction memory could be loaded from external media before a run commenced but could not be written-to by other instructions.

The Mark III and IV both therefore implemented all the design principles of the EDVAC identified earlier, with the exceptions that they stayed with decimal representation and did not store instructions and data in a common memory space.

Aiken's rationale for keeping the stores separate

Given that the Mark III was designed from scratch, that the design did not start until well after the First Draft had been widely circulated, and that it clearly adopted many principles from the First Draft, it seems likely that the idea of fully interchangeable memory was consciously rejected.

It is well documented that Aiken abhorred the idea of altering code at run time. In addition to the sources cited earlier, Fred Brooks recalls that Aiken was

> *'…so adamant about protecting proven program code that after he had recorded instructions on the drum on the Mark IV he unplugged the write circuits.'* [3]

which was almost certainly pure showmanship by Aiken: none of the Harvard machines had instructions that could write to the instruction store, so unplugging the write circuits after loading the program would have made no difference.

However, we do not have clear evidence that Aiken held these strong views at the time the Mark III was being designed (commencing January 1948). Indeed, it seems unlikely given that at that time no-one yet had practical experience of running programs where the instructions were being modified at run-time. Furthermore, the Mark III would implement the principle of 'Instructions operating on variable addresses' (**#10**) via specialised registers: a 'delta' register for indexed addressing, and a register to store the *previous* value of the 'line number' (program counter) for returning from subroutines. So, there would have been no need – identified at that time – to modify instructions on the fly on the Mark III.

Aiken's fear of the consequences was, arguably, misplaced. For though it is true that the non-writable (at run time) instruction store prevented corruption of code, having variable addressing implemented via registers does not reduce the likelihood of accidental corruption to data, as any programmer who has ever made an 'off by one error' in their indexing will testify.

Another possible argument for the separated stores and access circuits is that it could permit the next instruction to be read while the current one is being decoded and/or executed. However, this was not possible on any Harvard machine – except for instructions such as multiply that had their own dedicated circuits. The Mark III's processing cycle (approx. 4.3 milliseconds) generally commenced with reading the next instruction, before (in most cases) going on to read and write data.

The most likely explanation for the split memories is simply that it allowed the design of the two stores to be optimised to the different characteristics – both static and dynamic – of data and

instructions. Thinking of instructions and data as entirely separate things had begun with the Mark I. Ceruzzi states:

> *'That the* [instruction and data tape] *units were physically identical suggests that the Mark I's designers might have recognized that in some sense numbers and operations are equivalent. Probably they did not.'* [7]

Supporting that argument is the fact that one of the Mark I's three data tape readers was subsequently converted into a second instruction tape reader to facilitate switching between tapes, but there does not appear to have been the idea to make the tape units dynamically interchangeable.

By the time of the Mark III, Aiken would have been aware of the proposal to treat data and instructions at least somewhat interchangeably, but he probably still saw greater advantages in keeping them separate. Even von Neumann's tentative suggestion of interchangeable memory in the First Draft (quoted earlier) had been preceded by the counterargument:

> '*While it appeared that various parts of this memory have to perform functions which differ somewhat in their nature and considerably in their purpose…'*. [40]

The strongest evidence for this claim lies in the Mark III's specifications for the instruction and data drums. While all drums use the same recording technology, every other aspect of the specifications is different (see Table 2).

	Instruction store	**'Fast' data storage**	**'Slow' data storage**
No. of drums	1	2	6
Drum diameter	16 inches	8 inches	8 inches
Rotation speed	1725 RPM	6900 RPM	6900 RPM
Pulse density	20 per inch	10 per inch	10 per inch
Parallel channels	152 x 1-bit	36 x 4-bit in total	400 x 4-bit in total
Bits per second (per channel)	28,776	28,903	28,903
Format	38-bit instruction accessed in parallel across 38 channels.	16-digit, signed decimal format accessed serially from a (4-bit) channel	16-digit, signed decimal format accessed serially from a (4-bit) channel
Capacity	4000 instructions	200 working numbers + 10 configurable constants + 150 permanent constants	4000 numbers

Table 2 *Summary of the drum memory specifications on the Harvard Mark III.*

Aiken's preferred representation of numbers (16-digit decimal, each digit encoded to 4 bits, using an unusual representation [16] , and of instructions (a 38-bit format that facilitated the design of a radical and effective keyboard, closely matching the instruction semantics, to punch the tape) were very different. To implement a single shared store would have required the adoption of a larger address range (pushing up the number of bits needed for each of three addresses in the instruction format), as well as a fixed size 'word' for both data and instructions. Nor would it have been possible to load the whole instruction in parallel, while loading the data serially by decimal digit and parallel within each digit, which suited his decimal processing circuits.

From this start point, it seems likely that the design of the Mark III's two stores proceeded independently. Ultimately, successive instructions had to be recorded 125 rows apart on the drum, to put the next instruction close to where the read head would be by the end of the processing

cycle. And while the 8 data storage drums were all driven from a single motor via a gearbox (the Mark III must surely be the only computer where the operator panel included a 'Gearbox low oil pressure' warning light), the instruction drum was driven by its own motor – a far from optimal arrangement that required sophisticated electronic speed control circuitry to keep the instruction and data drums in synch. Unsurprisingly, the Mark IV – having moved the high-speed data memory to solid-state technology – merged the instruction and slow-speed data stores onto a single drum. However, they were still allocated physically separate channels, accessed via separate circuits.

The end of the Harvard machines

The Mark IV was the last of the Harvard machines. With the emergence of commercially manufactured computers, military funding for expensive one-off computing machines was less available. In 1956 the HCL installed a UNIVAC I [8] , the first commercially-produced electronic computer – a gift from Remington Rand[2]. We can only imagine Aiken's reaction to the fact that this machine relied on modifying instructions *in situ* to implement subroutines [36]

Another factor in the demise may have been, as Calingaert recalls in [4] , that Aiken kept the HCL team largely isolated from developments elsewhere. This isolation cut both ways: in the words of Aiken's biographer, friend, and colleague at Harvard, I. Bernard Cohen, Aiken's machines had little influence on the main line of the rapidly developing design of computers [9]

Had the HCL continued to develop its own range of machines, Aiken's insistence that program code should be immutable would have proven to be a huge liability with the emergence of operating systems. In the era of the Harvard machines, the user had been in complete charge of a machine. If their program contained errors, these could impact only that user's run; the machine would then be reset, and control of the machine passed to the next user. Batch processing changed that, to be followed later by time-sharing [42] [42] Both relied on a 'supervisor' program, responsible for loading, running, and unloading the user program and data, and for intervening to cancel a program that failed to terminate within a time limit or attempted an illegal action. The supervisor program had to be able to write instructions into memory that would then be executed.

With the new need, however, came new forms of protection against corruption – accidental or intentional – by the user programs. A hardware mechanism limited the memory address range accessible to a user program, with the supervisor running in a privileged mode that gave it access to the full memory.

Neither the Harvard machines nor the EDVAC design – at least had it followed von Neumann's idea that only the address portion of an instruction could be over-written – could have made the transition to this new world of supervisory programs or operating systems. The difference is that the EDVAC design could evolve without anyone having to recant a strongly espoused principle of doctrine.

The final irony is that while Aiken might have been gratified to learn that – 50 years after his death – dynamic mutation of code is typically prevented (in user programs), he would surely have been astonished to learn that an increasing number of computer scientists, especially those favouring functional programming languages, now advocate the run-time immutability of *data* [20] [20]

[2] Based on an email to the author from Peter Calingaert, who worked in the HCL at that time.

The coining of 'Harvard architecture' in the first microcontrollers

The term 'Harvard architecture' did not exist in the era of the Harvard machines. Even the word 'architecture' was not applied to the context of computing until the early 1960s [19] . References may be found to the 'Harvard class' (of computers) [23] , or to the 'Aiken architecture' [39] in the 1970s, though the meaning given to these terms is not consistent. The term 'Harvard architecture' does not appear in print until 1982 [26] [26] – and it was given a specific meaning that would not have applied to the Harvard machines.

In 1971, the same year that Intel had announced the first 4-bit microprocessor, Texas Instruments (TI) had developed the first complete computing device on a single chip: microprocessor, memory, and I/O. These devices eventually became known as 'microcontrollers'; they were used in embedded applications such as industrial controllers, domestic appliances, calculators, and electronic toys. Instructions and data were stored in separate memories. This was not a design choice: it was dictated by requirements. For most embedded applications, the program had to be persistent, and the device had to boot up automatically from power-on, so instructions were stored in ROM, while variable data had to be stored in the (typically smaller) RAM. However, this starting point suggested a new possibility for connecting the on-chip components, where the RAM is connected to an address and data bus, but the ROM is hardwired to the program counter (PC) and the instruction register (IR) – see Figure 3.

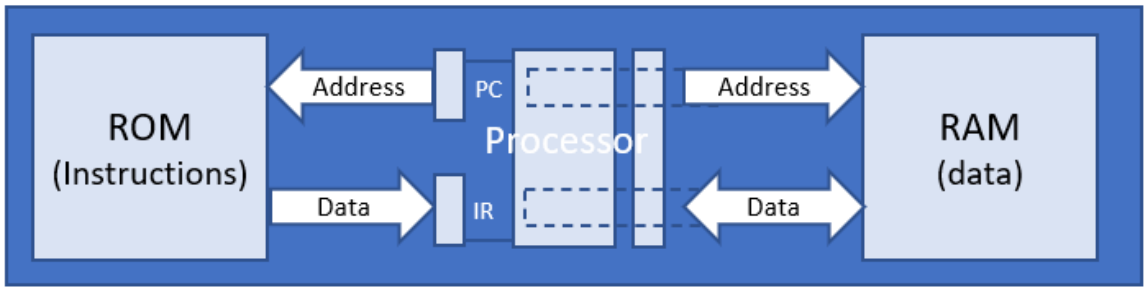

Figure 1 *In the early microcontrollers, the processor's Program Counter (PC) was hard-wired to the address inputs to the ROM, and the data output from the ROM was hard-wired to the Instruction Register (IR). This specific design was the first use of the term 'Harvard architecture'.*

The main motivations for this new arrangement in microcontrollers were simplicity (it needed less multiplexing on the buses), which translated into lower cost, and that the ROM and RAM could have different data widths and different address widths. TI's first microcontroller, for example, had 1024 x 8 bits of ROM and 64 x 4 bits of RAM.

This new configuration also meant that fetching an instruction could overlap with reading or writing data. As soon as the value in the PC is incremented, or overwritten (for a jump), the next instruction automatically appears on the input to the IR – without having to wait for the current instruction to finish using the address/data buses for accessing data. The next clock pulse merely latches the instruction into the IR.

The new arrangement was not designed to prevent writing to the program store: it was dependent on the lack of need for it. And it was suited only to microcontrollers because it depended upon either the ROM or RAM (or both) being on the same chip as the processor: there was not enough space at the edge of a chip to permit two data buses plus two address buses to be exposed as 'tabs'

and hence as pins on the package. (Chip and package sizes have increased over time; but so have address and data bus widths).

This arrangement became known as the 'Harvard architecture'. When the term was coined is unclear[3], but the 1982 publication [26] clearly implies that the term had been in use within the microcontroller design community for some time. The name is widely assumed to be a nod to the Harvard machines but, if so, the connection is tenuous at best. Where the Harvard Mark III/IV had separated the stores by choice, on the microcontrollers it was not a choice. And the only beneficial side effect gained from the split that microcontrollers had in common with the Harvard machines, was the possibility of different address and word sizes for instructions and data. It is quite possible that whoever coined the term wrongly believed that the Mark III/IV designs had also enabled overlapping instruction and data fetches. Possibly they just wanted a name for the new pattern, and 'Harvard architecture' offered a certain cachet.

By 1982 it was already being argued that advances in silicon manufacture made the shared memory address space – as used on most microprocessors – a valid option for microcontrollers [26] . However, the 'Harvard architecture' – as defined for microcontrollers – would continue to be used in some Digital Signal Processor (DSP) chips, a specialised development from the original microcontrollers, where the speed advantage carries a greater significance [31] . It has also been used in several more recent 'retro-computing' projects where the aim is to build a complete computer from scratch using only technology that existed before the first microprocessors[4].

Split caches in RISC microprocessors

The third historical context to which the term 'Harvard architecture' is now commonly attached is the emergence of a new generation of microprocessors arising from three research projects in the early 1980s – the IBM 801 minicomputer [35] , the MIPS project at Stanford [22] , and the RISC project at Berkeley [32] – for which RISC (Reduced Instruction Set Computing), would later become the generic name. 'Harvard architecture' does not appear in any of the original research project reports, but external commentators had started to add this label by the late 1980s [34] and it was being used in product documentation by 1990 [32]

Although various approaches to splitting instructions from data were explored in these projects, the one that stuck was to keep them together in main memory, but cache them separately within the processor (Figure 4). One motivation for that split was that from the perspective of caching, the characteristics of the two things were different, allowing the hardware cache logic to be optimised for each. For example, unlike the data cache, the instruction cache does not have to worry about instructions becoming 'stale', because the instructions do not change dynamically (the latter was enforced by the memory management unit). It would also permit simultaneous instruction and data fetch, provided that the fetch was from the cache.

[3] The author believes that the term was most likely coined by Gary Boone, or by one of his team, at Texas Instruments working on single chip microcomputing devices in the early 1970s. The author managed to locate one surviving member of that original team, Charles Brixey, in April 2021, who recalled that the term 'Harvard architecture' was in use within the team from early on, but did not know its origin. Surendar Magar (see [31]), who later worked closely with that team, gave a similar recollection.

[4] One of the best-known of these is the Gigatron: https://gigatron.io/

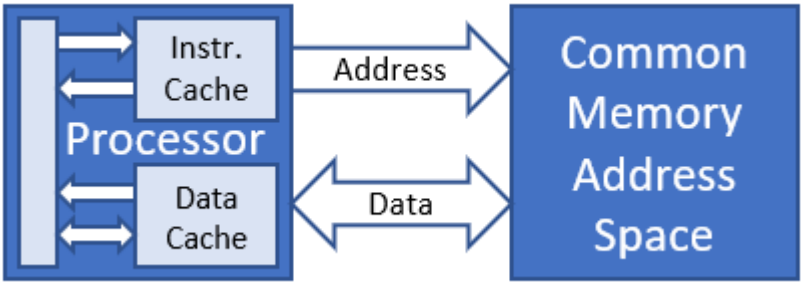

Figure 2 *In the 1990s the term 'Harvard architecture' appeared again, to describe processors that cache both instructions and data, but separately*

The actual performance gain attributable specifically to splitting the caches was quite small compared to that delivered by caching in general. The evolution of the ARM – the family of RISC processors found in most mobile phones – provides a benchmark [12] In the ARM1 there was no separation of instructions and data; 'pipelining' (overlapping of instruction and data fetches) was achieved by a different mechanism. Caching was introduced with ARM3. Simulation had shown that a 'perfect' caching mechanism (one where every item happens to be in the cache when it is needed) would deliver a 1.13x performance improvement if data was cached, a 1.95x improvement if instructions were cached, and a 2.5x improvement if both data and instructions were cached. As implemented in ARM3, a single on-chip cache was used for both data and instructions, and the measured performance improvement came surprisingly close to the perfect ideal. The ratio of cached instructions to data was not fixed: it would vary dynamically during operation.

The StrongARM CPU [12] developed by Digital Equipment Corporation (DEC) and derived from extant ARM designs, was the first to feature separate caches for instructions and data, and this pattern was subsequently adopted in the ARM9 and most subsequent implementations. The ARM9 got close to the *ideal* nett throughput of one instruction per clock cycle, but prior versions of ARM were already delivering better than 80% of this goal by other means. So, the gain from splitting the cache was at most 1.25x, and probably rather less since it was introduced at the same time as multiple other performance-enhancing tweaks.

This split-cache design is, today, the most widely used meaning for the term 'Harvard architecture'. It is sometimes labelled as 'modified Harvard architecture', though that term had been coined back in 1982, with a different meaning[5].

Conclusion

The three historical developments that have been labelled as 'Harvard architecture' are summarised in Table 3. All adopted most of the principles associated with the term 'von Neumann architecture', including the fact that they ultimately stored instructions and data in dedicated registers. All three additionally split instructions and data earlier in the process, but in different ways, to achieve different benefits, and with different limitations.

[5] Surendar Magar had used the term 'modified Harvard architecture' in his 1982 patent application, [31] where it referred to the fact that the processor could load instructions from the data memory for debugging purposes.

Period	Devices	Design	Possible benefits	Limitations	Applicability
Late 1940s to early 1950s	Harvard Mark III/IV	Physically separate instruction and data stores	- Optimisation of each store design to characteristics of instructions or data	- Program loaded manually - Memory allocation fixed	Not viable for modern computing
1970s onwards	Microcontrollers and later DSP chips	Instructions in ROM, hardwired to Program Counter and Instruction Register	- Simpler design - Supports different word/address sizes - Instruction/data fetches in parallel	- Program must be in ROM - ROM or RAM must be small enough to be on-chip	Embedded computing applications only
1990s onwards	Microprocessors with split on-chip caches	Instructions and data held in common RAM, but cached separately on the processor	- Modest gain in performance though cache design optimisation, and by permitting instruction/data fetches in parallel (when from cache)		General purpose computing

Table 3 *Summary of the three distinct meanings of the term 'Harvard architecture' in relation to three historical developments.*

Only the third development is relevant to modern general-purpose computing, because it is the only one that will work with an operating system. The performance advantage (relative to having a unified cache for data and instructions) is modest. And it would be more accurately described as a 'modified von Neumann architecture' than a 'modified Harvard architecture'. In short, it isn't an architecture, and it didn't derive from work at Harvard.

As well as perpetuating the myth that Howard Aiken developed a better architecture, the advantages of which were not recognised until after his passing, the continuing use of the term 'Harvard architecture' reinforces the idea that there is a dichotomy between that and the 'von Neumann architecture'. The author has seen the negative consequences of this in an educational context, where students come away with the impression that modern computers are either 'von Neumann' or 'Harvard'. One question in a high-school Computer Science exam set by a public exam board, asked for advice to be made to a business whose systems were running too slowly. Included in the official Mark Scheme's examples of creditable points that might be made in the short-essay response, alongside such valid points as 'replace HDDs with SSDs' and 'install more RAM', was 'use the Harvard architecture'. The exam board's decision to cite such a suggestion as valid is perhaps understandable when an otherwise respectable textbook implies that Harvard vs. von Neumann is just another buyer's choice, like Linux or Windows:

> '*The chief disadvantage* [of the Harvard architecture] *arises from inflexibility: when purchasing a computer, an owner must choose the size of the instruction memory and the size of data memory. Once the computer has been purchased, an owner cannot use part of the instruction memory to store data nor can he or she use part of the data memory to store programs.*' [10]

Research for this article stopped short of asking for such a machine at the local computer store.

Acknowledgements

Many people provided help to the author in the form of pointers to useful sources, personal recollections, feedback on early ideas or drafts, or patient explanations of complex technical ideas. Notwithstanding which, the author takes full responsibility for any remaining errors or misunderstandings. Nor does inclusion in these acknowledgements imply endorsement for any argument advanced in this article. Thank you specifically to (in alphabetical order): Penny Ahlstrand,

Walter Belgers, Gordon Bell, Charles Brixey, Fred Brooks, Peter Calingaert, Martin Campbell-Kelly, Paul Ceruzzi, Steve Furber, Steve Golson, Tom Haigh, John Hennessy, Andrew Herbert, Peter Higginson, Surendar Magar, Ada Negraru, David Patterson, Simon Peyton Jones, Mark Priestley, Brian Randell, Mark Smotherman, John Stout.

Author Biography

Since starting his career at Commodore in 1977, Richard Pawson has worked in many different roles within the computing industry: software development, robotics, electronic toy design, technology journalism, consulting, and teaching. He currently manages a large open-source software-development framework and creates free resources for teaching computer science. He has a degree in Engineering Science, a PhD in Computer Science, and a PGC in Intellectual Property Law. He lives near Henley-on-Thames, UK and may be contacted as rpawson@metalup.org

Bibliography

[1] H. H. Aiken, "Proposed Automatic Calculating Machine" (Nov. 4th, 1937), reproduced in *Makin' Numbers – Howard Aiken and the Computer*, I.B.Cohen and G.W.Welch, Eds. MIT Press, Cambridge, MA, USA, 1999, pp 9–29

[2] W. Aspray, *John Von Neumann and the Origins of Modern Computing*, MIT Press, Cambridge, MA, USA, 1990, pp 25–48

[3] F. Brooks, Jr, "Aiken and the Harvard 'Comp Lab'" in *Makin' Numbers – Howard Aiken and the Computer*, I.B.Cohen and G.W.Welch, Eds. MIT Press, Cambridge, MA, USA, 1999, p140

[4] P. Calingaert, "Aiken as a Teacher" in *Makin' Numbers – Howard Aiken and the Computer*, I.B.Cohen and G.W.Welch, Eds. MIT Press, Cambridge, MA, USA, 1999, p159

[5] R. Campbell, "Aiken's First Machine: The IBM ASCC/Harvard Mark I" in *Makin' Numbers – Howard Aiken and the Computer*, I.B.Cohen and G.W.Welch, Eds. MIT Press, Cambridge, MA, USA, 1999, pp 31–63

[6] R. Campbell, "Mark II, an Improved Mark I", in *Makin' Numbers – Howard Aiken and the Computer*, I.B.Cohen and G.W.Welch, Eds. MIT Press, Cambridge, MA, USA, 1999, pp 111–127

[7] P. Ceruzzi, "Introduction" to reprint of *A Manual of Operation for the Automatic Sequence Controlled Calculator*, in the Charles Babbage Institute Reprint Series for the History of Computing, MIT Press, Cambridge, MA, USA, 1985, p xxiii

[8] I. B. Cohen, *Howard Aiken – Portrait of a Computer Pioneer*, MIT Press, Cambridge, MA, USA, 2000, p205

[9] I. B. Cohen, "Howard Aiken and the Dawn of the Computer Age" in *The First Computers, History and Architectures*, R. Rojas and U. Hashagen, Eds., MIT Press, Cambridge, MA, USA, 2002, p119

[10] D. Comer, *Essentials of Computer Architecture*, Chapman & Hall, London, U.K. 2017, pp 70–71

[11] M. Cusumano, Y. Mylonadis, and R. S. Rosenbloom, "Strategic Maneuvering and Mass-Market Dynamics: The Triumph of VHS over Beta", *Business History Review 66*, Mar. 1992

[12] S. Furber, *ARM System-on-chip architecture*, second edition, Addison-Wesley, Reading, MA, USA 2000

[13] M. D. Godfrey and D. F. Hendry, "The Computer as von Neumann Planned it", *IEEE Annals of the Hist. of Comput,*, Vol 15, No.1, 1993

[14] Harvard University Computation Laboratory, *A Manual of Operation for the Automatic Sequence Controlled Calculator*, Harvard University Press, Cambridge, MA, USA 1946

[15] Harvard University Computation Laboratory ,"Description of a Relay Calculator", *The Annals of the Computation Laboratory of Harvard university*, Volume XXIV, Harvard University Press, Cambridge, MA, USA, 1949

[16] Harvard University Computation Laboratory, *Description of a Magnetic Drum Calculator*, Harvard University Press, Cambridge, MA, 1952

[17] T. Haigh, M. Priestley, and C. Rope, "Reconsidering the Stored-Program Concept", *IEEE Annals of the Hist. of Comput.*, January-March 2014

[18] T. Haigh, M. Priestley, and C. Rope, *ENIAC in Action*, MIT Press, Cambridge, MA, USA, 2016, pp 153–171

[19] D. Halsted, "The Origins of the Architectural Metaphor in Computing", *Annals of the Hist. of Comput.*, January-March 2018

[20] P. Helland, "Immutability changes everything", *Communications of the ACM* 59(1), 2016

[21] J. L. Hennessy, D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 1st ed. Morgan Kaufmann (Publishers, Inc.), San Mateo, CA, USA, 1990, p25

[22] J. Hennessy, N. Jouppi, S. Przybylski, C. Rowen, T. Gross, F. Baskett, and J. Gill, "MIPS: A Microprocessor Architecture", *ACM SIGMICRO Newslett.* Volume 13 Issue 4 Dec. 1982

[23] R. B. Hibbs, "Features of an advanced front-end CPU", *AFIPS '71 (Spring): Proc. May 18-20, 1971, Spring Joint Comput. Conf.*

[24] G. M. Hopper, "Commander Aiken and My Favorite Computer", in *Makin' Numbers – Howard Aiken and the Computer*, I.B.Cohen and G.W.Welch, Eds. MIT Press, Cambridge, MA, USA, 1999, pp 187–188

[25] G. M. Hopper, *Original Drafts for Mark II Manual*, Grace Murray Hopper Collection, Smithsonian Institution, 1948

[26] B. Huston, "Single-chip microcomputers can be easy to program", in *Proc. June 7-10, 1982, National Computer Conference, New York, NY, USA*, Association for Computing Machinery

[27] D. Knuth, "Von Neumann's First Computer Program", *Computing Surveys*, Vol.2 No.4, December 1970

[28] S. H. Lavington, *Early British Computers*, Manchester University Press, U.K. 1980, p37

[29] J. Ledin, *Modern Computer Architecture and Organization*, Packt Publishing, Birmingham, U.K. 2020, p175

[30] J .A. N. Lee, "Howard Aiken's Third Machine: The Harvard Mark III Calculator or Aiken-Dahlgren Electronic Calculator", *IEEE Annals of the Hist. of Comput.*, January-March 2000

[31] S. Magar, "Microcomputer with ROM test mode of operation", *US Patent 4,507,727*, Mar. 1985.

[32] Motorola Corp. *MC88100 RISC Microprocessor user's manual*, 2nd edition, Prentice-Hall (Inc.), Englewood Cliffs, NJ, USA, 1990, p22

[33] D. A. Patterson, C. H. Sequin, "RISC-I: A Reduced Instruction Set VLSI Computer", *Proc. Eighth Annual Symp. on Comput. Architecture,* Minneapolis, MN, May 1981

[34] T. Perry, "386 vs. 030: the Crowded Fast Lane", *Dr. Dobbs Journal*, 1st Jan 1988, available: https://www.drdobbs.com/386-vs-030-the-crowded-fast-lane/184407891?pgno=4

[35] G. Radin, "The 801 minicomputer", *ACM SIGPLAN Notices*, Volume 17 Issue 4 April 1982

[36] Remington Rand Inc. *Programming for the UNIVAC FAC-TRONIC System*, 1953. p72

[37] S. Shiva, *Advanced Computer Architectures*, Taylor & Francis, New York, NY, USA 2006, p18

[38] T. R. Thompson, "Description of Harvard Mark IV", Colloquium at Cambridge, 1952. Available: http://www.computinghistory.org.uk/det/63078/63078-Description-of-Harvard-Mark-IV-by-A.E.-Oettinger

[39] K. J. Thurber, D. Jensen, L. A. Jack, L. L. Kinney, P. C. Patton, L. C. Anderson, "A systematic approach to the design of digital bussing structures", *AFIPS '72 (Fall, part II): Proc. December 5-7, 1972, Fall Joint Computer Conference, part II*

[40] J. von Neumann, *First Draft of the Report on the EDVAC*, June 1945

[41] M. V. Wilkes, D. Wheeler and S. Gill, *The Preparation of Programs for an Electronic Digital Computer*, Addison-Wesley, Reading, MA, USA 1951, pp 25–50

[42] M. V. Wilkes, *Time-Sharing Computer Systems,* Elsevier (Science Publishing Co.), Amsterdam, The Netherlands; New York, NY, USA 1968, pp 20–35