USING EARLY INSTRUCTION SETS TO INTRODUCE COMPUTER ARCHITECTURE*

David L. Tarnoff
Department of Computer and Information Sciences - 70711
East Tennessee State University
Johnson City, TN 37614
423 439-6404
tarnoff@etsu.edu

ABSTRACT

Motivating computer science students to study computer architecture can be difficult, especially when complex, modern architectures such as the Intel® Core™ i7 are held up as examples. The solution presented here is to introduce computer architecture by first teaching students how to program simpler, historic machines such as Konrad Zuse's Z1, the Manchester Baby, and the Princeton IAS machine. The limitations of these machines can then be used to motivate students to learn the principles of addressing modes, instruction set architectures, and CPU register design. These early architectures can demonstrate how much, and how little, has changed in the area of computer architecture. This paper summarizes the results of research into the machine language instruction sets and architectures of the Zuse Z1, the Manchester Baby, and the Princeton IAS machine. It then presents examples of how these instruction sets can be used to motivate students in a senior-level computer architecture course.

INTRODUCTION

The first week of a course in computer architecture might acquaint students with a brief history of computing. The instruction might go something like, "The IBM Automatic Sequence Controlled Calculator for which the Harvard architecture is named contained 500 miles of wire 3,300 relays, and 1,464 ten-pole switches for the input of decimal constants. It weighed 10,000 pounds and required a five horsepower motor to

---

turn a 50' shaft that drove the calculator's mechanical components." [4] These facts, while accurate, do not encourage a student to go deeper into the topic, nor do they introduce the architecture used by these machines.

The goal of this work is to use very early instruction sets to motivate students to begin thinking about architectural issues. A study of these machines should also show students that concepts such as pipelining, microcode, and parallel processing have been around since Charles Babbage designed his Analytical Engine in 1838 [6].

The next few sections of this paper present a brief overview of the architectures of three early machines. Following these sections is a discussion of how these architectures can be used to support the study of modern computer architecture.

KONRAD ZUSE'S Z1

From 1936 to 1938, Konrad Zuse constructed his Z1, an automaton he had designed in order to perform the time-consuming calculations necessary in his occupation as a civil engineer [5]. It incorporated a number of important features still used in modern computing including a floating-point, binary numbering system, an addressable memory to store data, distinct decimal input and output units, an arithmetic unit with a carry look-ahead adder, and a control unit that used a two-stage execution pipeline. The machine itself was mechanical although Zuse later developed machines with electromechanical relays. His third machine, the Z3, was identical in architecture to the Z1 except that it replaced the unreliable mechanical elements with electromechanical relays and added a square root function [7].

The Z1 used a Harvard-style architecture executing programs directly from a punch tape reader (35mm standard movie film) while it stored data in a separate memory. In the case of the Z1, storing the programs in memory was avoided because of the expense of the memory [5].

Zuse's floating-point data format is remarkably similar to the IEEE Standard for Floating-Point Arithmetic (IEEE 754) in common use today. The most significant bit is a sign bit for the value. The next seven bits contain the base-two exponent coded in two's complement representation. The last fourteen bits are the significant without the "hidden bit", i.e., the most significant bit of the significand, which is always a one for non-zero values. Zero was represented with an exponent of -64 [7].

The Z1 instruction set consisted of eight 8-bit instructions including two for memory transfers and one each for input and output. The remaining instructions performed mathematical operations on the arithmetic unit's two floating-point registers, R1 and R2. Addition, subtraction, multiplication, and division combined R1 and R2 placing the result in R1 and clearing R2 [7].

The memory load instruction's operand field held the address from which data was to be loaded, but did not identify the destination register. This was because the first load would load R1 while subsequent loads loaded R2. The memory store instruction's operand field also did not identify a register. All stores would store R1 to memory and clear R1. A store would reset the loading process to load R1 next.

Table 1 presents the Z1 instruction set along with each instruction's opcode [7].

Table 1: Z1 Instruction Set [7]

Instruction/Description	Opcode	Cycles
Lu - read keyboard Ld - display result	01 110000 01 111000	9 to 41 9 to 41
Pr z - load from address z Ps z - store to address z	11 Z6Z5Z4Z3Z2Z1 10 Z6Z5Z4Z3Z2Z1	1 0 or 1
Lm - multiplication Li - division Ls1 - addition Ls2 - subtraction	01 001000 01 010000 01 100000 01 101000	16 18 3 4 or 5

What the Z1 lacked was program flow control. There were no conditional branches. Loops, therefore, could only be implemented by connecting the paper tape's ends to form a physical loop in the code. There was also no way to implement decision structures [7].

MANCHESTER SMALL-SCALE EXPERIMENTAL MACHINE

In 1948, a proof of concept for an electronic memory called the Williams Tube resulted in the Manchester Small-Scale Experimental Machine (SSEM), the first stored-program computer. The SSEM, nicknamed "Baby," had a 32x32-bit memory, a 32-bit accumulator for intermediate results, and a register, CI, that points to the address of the current instruction [2].

Both instructions and data were stored using 32-bit words. The instruction contained a 13-bit address (bits 0 to 12) and a 3-bit function field (bits 13 to 15). Because the SSEM had a 32-word memory, only the first five bit positions were required for the address.

Figure 1 presents the SSEM instruction layout. The leftmost bit, bit 0, represented the least significant digit. It was common at the time of the SSEM development to view bits as if they were arranged along an X-axis with magnitude increasing from left to right. This was also true for numeric values, which were represented in 32-bit two's complement form [2].

Bit position:	0	1	2	3	4	5..12	13	14	15	16..31
Function:	Memory address					Unused	Function			Unused

Figure 1: SSEM Instruction Layout [2]

The SSEM had a small instruction set, each function of which was identified using three bits. One function accessed the arithmetic unit, which was capable only of subtraction and negation. For flow control there were two unconditional jump functions, a conditional jump function, and a stop function. One function for loading and one function for storing allowed the SSEM to access data in memory. Table 2 presents the SSEM instruction set along with the binary function codes for programming it [2].

Program flow was controlled with the control register CI. Before an instruction was loaded, CI was automatically incremented by one to the next memory address. This meant

that if the SSEM initialized CI to 0 before running a program, the first instruction to be executed would be at location 1, not 0. This also meant that the destination address for jump instructions was given as one less than the desired address.

Table 2: SSEM Instruction Set [2]

Binary Code	Modern Mnemonic	Instruction/Description
000	JMP	Load CI w/address stored in location pointed to by instruction
100	JRP	Retrieve offset stored in location pointed to by instruction and add to CI
010	LDN	Copy the negated contents of the memory location to accumulator
110	STO	Store contents of accumulator to memory location
001	SUB	Subtract the contents of the memory location from the accumulator
101	-	Not defined (If used, SSEM performed a SUB)
011	CMP	Skip the next instruction if the content of the accumulator is negative
111	STOP	Halt the machine

The CMP function also affected CI. If a CMP instruction was encountered and the accumulator contained a negative value, CI was incremented a second time thereby skipping the subsequent instruction [2].

The SSEM's two jump commands are considered special cases in modern computing. The JMP function is now known as an indirect jump. This means that the memory address referenced by the instruction contained the address to be jumped to unconditionally. The JRP function is now known as an indirect relative jump. This means that the value stored at the address pointed to by the instruction represents an offset from the current instruction. Although JRP is not necessary for a machine of SSEM's simplicity, it was added because its cost was trivial, yet it allowed for relocatable code. For example, a loop with an instruction that says, "jump 5 memory locations back" instead of using a hard coded address can be placed anywhere in memory [2].

PRINCETON IAS MACHINE

After having worked on the Electronic Numerical Integrator and Calculator (ENIAC) with John W. Mauchly, a professor at the University of Pennsylvania's Moore School of Electrical Engineering, and J. Presper Eckert, Mauchly's graduate student, John von Neumann, a mathematician at Princeton's Institute for Advanced Study (IAS), developed a new design for a stored program architecture. This work was motivated by the difficulties with programming the ENIAC manually by setting switches and routing cables, an arrangement that was time consuming and did not allow for the storing of programs for later reload and execution. With Mauchly and Eckert's input, von Neumann wrote a report titled, "First Draft of a Report on the EDVAC," which proposed a machine that stored instructions in memory alongside the data [9]

John von Neumann then collaborated with Arthur Burks and Herman Goldstine from Princeton's Institute for Advanced Study (IAS) to develop a new machine based on the EDVAC design, a machine later known as the IAS machine. The IAS machine, described in their report titled, "Preliminary Discussion of the Logical Design of an Electronic Computing Instrument," had the same architecture as that described in the EDVAC report although the later paper went into more detail regarding implementation including a description of the instruction set [1].

As implemented, the Princeton IAS machine had a 1,024 word binary memory, each location of which could store a 40-bit word. When used to store data, the 40-bit location used two's complement representation of a fraction, i.e., a fixed point value between -1 and 1. When used to store an instruction, each 40-bit location stored two 20-bit instructions identified as "left" and "right". The left instruction was executed first [3].

Each 20-bit instruction was partitioned into four parts: a 10-bit memory address, a step digit used to halt or continue operation, an 8-bit opcode, and a spare bit that identified special operations. The 10-bit address allowed the IAS machine to access any location in its 1,024 word memory. The step digit, if set to 0, halted the machine after the execution of an instruction without incrementing the program counter. If the step digit was set to 1, the program counter was incremented and the next instruction was executed. The 8-bit opcode defined the operation to execute. An instruction's last bit was 0 except in the case of a special order instruction [3]. Figure 2 presents the layout of the IAS instruction word.

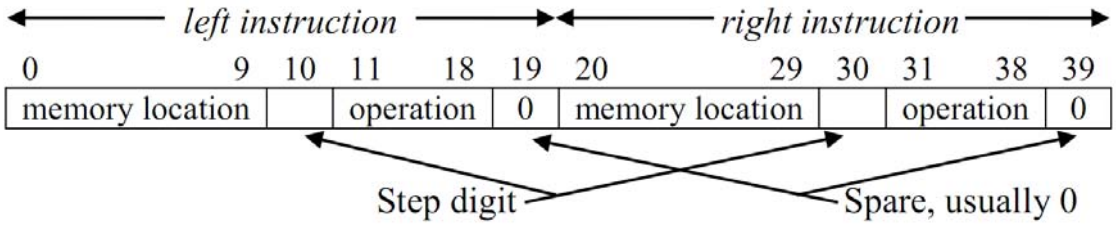

Figure 2: Princeton IAS Machine Instruction Layout

The IAS machine had seven registers for storing binary values. Two 40-bit registers, AC and MQ, in the arithmetic unit maintained the sources for and results of arithmetic operations. Five registers in the control unit maintained the step-by-step sequencing of instructions. These registers were the program counter (PC), the memory buffer register (MBR), the memory address register (MAR), the 40-bit instruction buffer register (IBR), and the register to hold the currently executed instruction (IR).

The final progress report on the Princeton IAS machine as submitted by Goldstine, Pomerene, and Smith also presented a detailed description of the instruction set including opcodes [3]. This description is summarized in Table 3.

In addition to the instructions presented in Table 3, there were six I/O instructions for transferring data between memory and the outside world. Over the course of its life, the IAS machine used I/O including teletypewriters, punch cards, and magnetic wire [10].

Table 3: Princeton IAS Machine Instruction Set [3]

Opcode (binary)	Operation description
11100101	Load AC with value from address in memory
11100100	Add value from address in memory to the AC
11101101	Load AC with negative of value from address in memory
11101100	Subtract value from address in memory from the AC
11110101	Load AC with absolute value of number from memory
11110100	Add absolute value of number from memory to the AC
11111101	Load AC with negative absolute value of number from memory
11111100	Subtract absolute value of number from memory from the AC
11100000	Multiply MQ with value from address in memory; result in AC:MQ
11100010	Multiply MQ with value from address in memory; upper half in AC
11111000	Divide AC by value from address in memory; AC=remainder, MQ=quotient
11100110	Load MQ with value from address in memory
11010100	Store AC to address in memory
11010101	Store AC to address in memory then clear AC
11010000	Unconditional jump to left instruction of destination address (step = 0)
11010000	Unconditional jump to right instruction of destination address (step = 1)
11011000	If AC >= 0, jump to left instruction of destination address (step = 0)
11011000	If AC >=0, jump to right instruction of destination address (step = 1)
10100000	Shift AC right one position with LSB shifting into MQ
10100010	Shift AC right one position discarding LSB
10101000	Shift AC right one position discarding MSB
10100100	Transfer MQ to AC
N/A*	Replace address portion of left instruction at address w/leftmost 10 bits in AC
N/A*	Replace address portion of right instruction at address w/leftmost 10 bits in AC

CLASSROOM APPLICATION

For the past two years, the author has opened his senior-level computer architecture course by presenting the details of these machines along with Charles Babbage's 1838 Analytical Engine design and the architecture and instruction set of the IBM ASCC. The students are familiarized with the machines through assignments such as writing a simulator for the IAS machine or writing code for the Manchester Baby, which can be executed using on-line simulators [8].

The machines are then referred to throughout the semester as an introduction to different architectural topics. For example, addressing modes can be introduced through a classroom discussion on the difficulties in creating constants or pointers using the IAS machine. Each of these machines only supported one numeric representation allowing the students to see the benefit of numbering systems and their effect on a machine's

applications. The Baby's relative jump command can serve as a jumping off point for a discussion on relocatable code. Microcode can be introduced using the description of Charles Babbage's use of studs screwed into music box-style barrels to implement instructions. The binary instructions of the IAS machine were partitioned into fields aiding in instruction set implementation. All but the IAS machine were Harvard architecture due to the expense of memory, so a comparison can be made between the IAS machine and the others to show how Harvard differs from von Neumann. Almost every one of the machines described in this paper implemented a two-stage pre-fetch pipeline. Finally, all of these systems had to be halted in order to perform user I/O, which can be used as a starting point to discuss polled I/O and then interrupts.

Since implementing the changes outlined in this document along with the use of a single-board MIPS processor, the students' satisfaction with the course and their feelings for how worthwhile it was improved dramatically. As measured by the university's student assessment of instruction tool, their rating of how worthwhile the computer architecture course was increased on a scale of 1 to 4 from a 2007 mean of 2.86 to a 2010 mean of 3.63.

CONCLUSION

Many of the fundamental concepts of computer architecture have existed since the first computing automatons were created. The complex implementations of modern computers present a challenging learning curve for even the brightest students. By covering the fundamentals of computer architecture by holding up early architectures as an example, students appear to gain a better understanding of how architecture affects computing and why certain implementations are so important.

BIBLIOGRAPHY

[1] Burks, A. W., Goldstine, H. H. and von Neumann, J., *Preliminary Discussion of the Logical Design of an Electronic Computer Instrument*, Princeton, NJ: Institute for Advanced Study, 1946.

[2] Burton, C. P., The Manchester University Small-Scale Experimental Machine programmer's reference manual, *Computer Conservation Society*, 2, 1997.

[3] Goldstine, H. H., Pomerene, J. H. and Smith, C. V., *Final Progress Report on the Physical Realization of an Electronic Computing Instrument*, Princeton, NJ: The Institute for Advanced Study, 1954.

[4] IBM, IBM's ASCC (a.k.a The Harvard Mark I), www-03.ibm.com/ibm/history/exhibits/markI/markI_intro.html, retrieved December 3, 2010.

[5] Maxfield, C., The life and work of Konrad Zuse, www.techbites.com/20090911498/myblog/articles/z000c-the-life-and-work-of-konrad-zuse-index.html, retrieved December 3, 2010.

[6] Myhrvold, N. & Swade, D., An evening with Nathan Myhrvold and Doron Swade: Discussing Charles Babbage's Difference Engine, 2008, www.computerhistory.org/events/index.php?id=1206647564, retrieved December 3, 2010.

[7] Rojas, R., Konrad Zuse's legacy: The architecture of the Z1 and Z3, *IEEE Annals of the History of Computing*, 19, (2), 1997.

[8] Sharp, D., Manchester Baby simulator, www.davidsharp.com/baby/, retrieved December 3, 2010.

[9] von Neumann, J., First draft of a report on the EDVAC (Original publication in 1945), *IEEE Annals of the History of Computing*, 15, (4), 1993.

[10] Ware, W. H., *The History and Development of the Electronic Computer Project at the Institute for Advanced Study*, Santa Monica, CA: RAND Corporation, 1953.