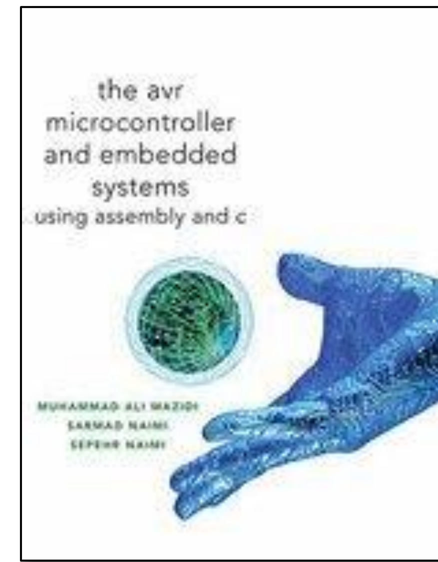
EE 346 Microprocessor Principles and Applications

An Introduction to Microcontrollers, Assembly Language, and Embedded Systems

Reading

The AVR Microcontroller and Embedded Systems using Assembly and C)
by Muhammad Ali Mazidi, Sarmad Naimi, and Sepehr Naimi



Chapter 0: Introduction to Computing

Section 0.3: Semiconductor Memory (except DRAM)

Section 0.4: CPU Architecture

Chapter 1: The AVR Microcontroller: History and Features

Section 1.2: Overview of the AVR Family

Chapter 2: AVR Architecture and Assembly Language Programming

Section 2.1: The General Purpose Registers in the AVR

Section 2.2: The AVR Data Memory

Section 2.8: The Program Counter and Program ROM Space in the AVR

Section 2.9: RISC Architecture in the AVR

Section 2.10: Viewing Registers and Memory with AVR Studio IDE

Chapter 3: Branch Call and Time Delay Loop

Section 3.3: AVR Time Delay and Instruction Pipeline

CONTENTS

What is a Flip-Flop and a Register 4

ATmega328P Block Diagram 5

The AVR Engine 6

Instruction Set Architecture 6

AVR CPU CORE Architecture 8

AVR CPU CORE Architecture 9

AVR CPU Instructions 10

Instruction Fetch and Execute 11

Harvard versus Princeton Memory Model Instruction Fetch Cycle 12

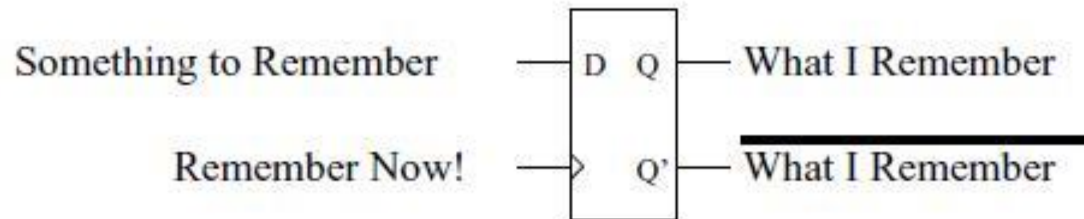
Atmel ATmega328P Memory Model 14

ATmega328P I/O Memory Map 15

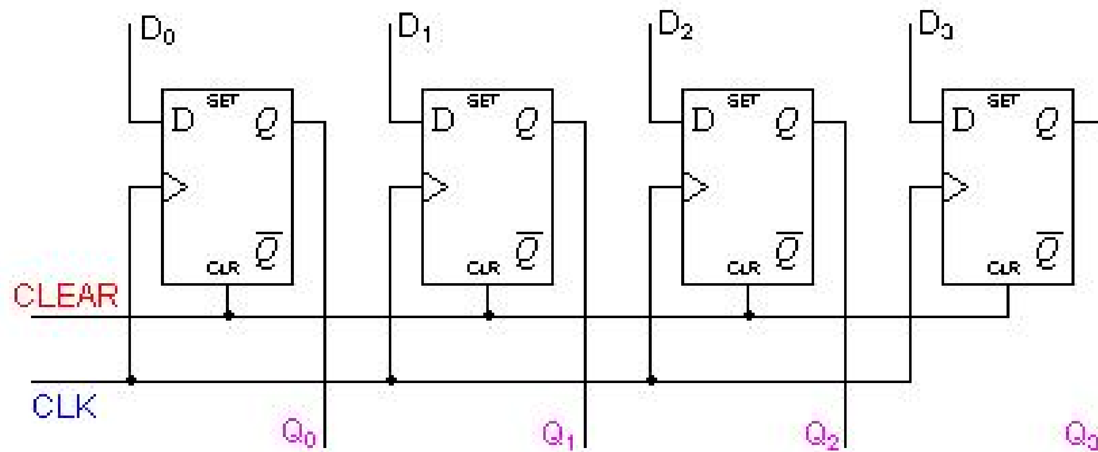
WHAT IS A FLIP-FLOP AND A REGISTER?

You can think of a D flip-flop as a one-bit memory. The *something to remember* on the D input of flip-flop is remembered on the positive edge of the clock input.[1]

D_t	Q_{t+1}
0	0
1	1
X	Q_t



A register is a collection of flip-flops sharing the same clock input.



[1] Source: http://sandbox.mc.edu/~bennet/cs314/slides/ch5me-4.pdf

ATMEGA328P BLOCK DIAGRAM[2]

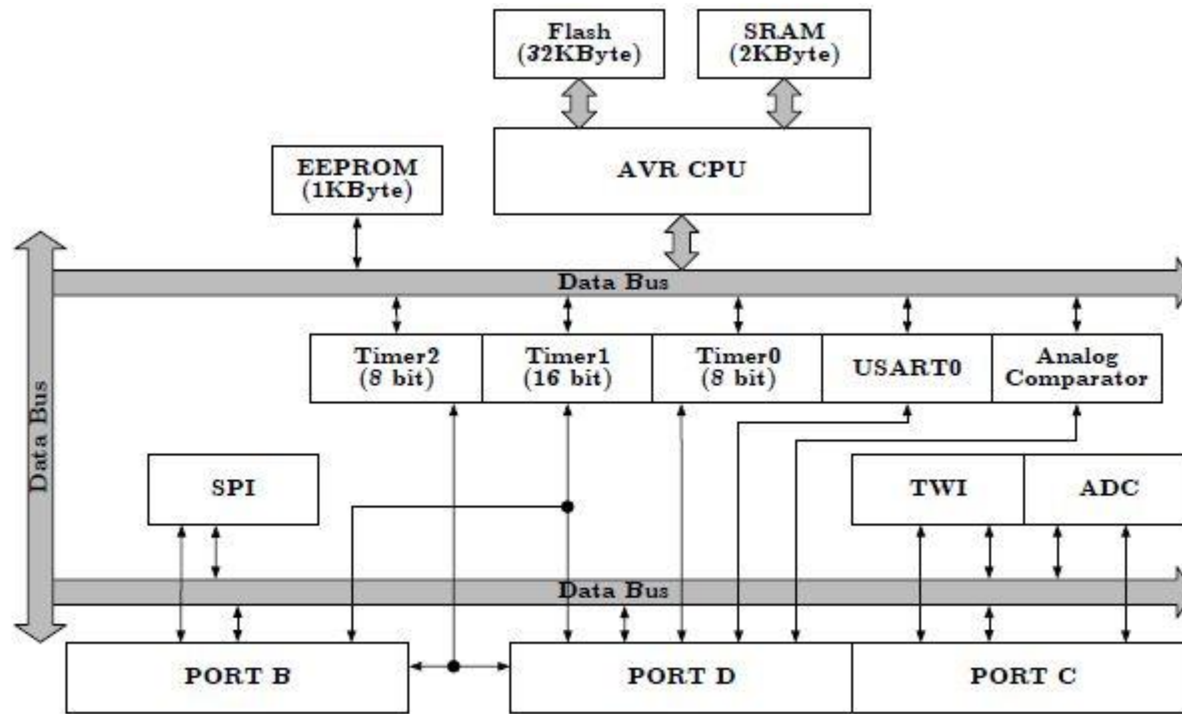


Figure 5.1: ATmega328P block diagram, adapted from ATMEL (2009).

[2] Here is a more accurate block diagram: ATmega328P Data Sheet http://www.atmel.com/dyn/resources/prod_documents/8161S.pdf page 5 Figure 2-1 Block Diagram

The AVR Engine

Let's adopt the analogy used by Charles Babbage when he called his computer an Analytical Engine. For closer look see this [article in Wired]() and [ATmega328 Wikipedia]() page.

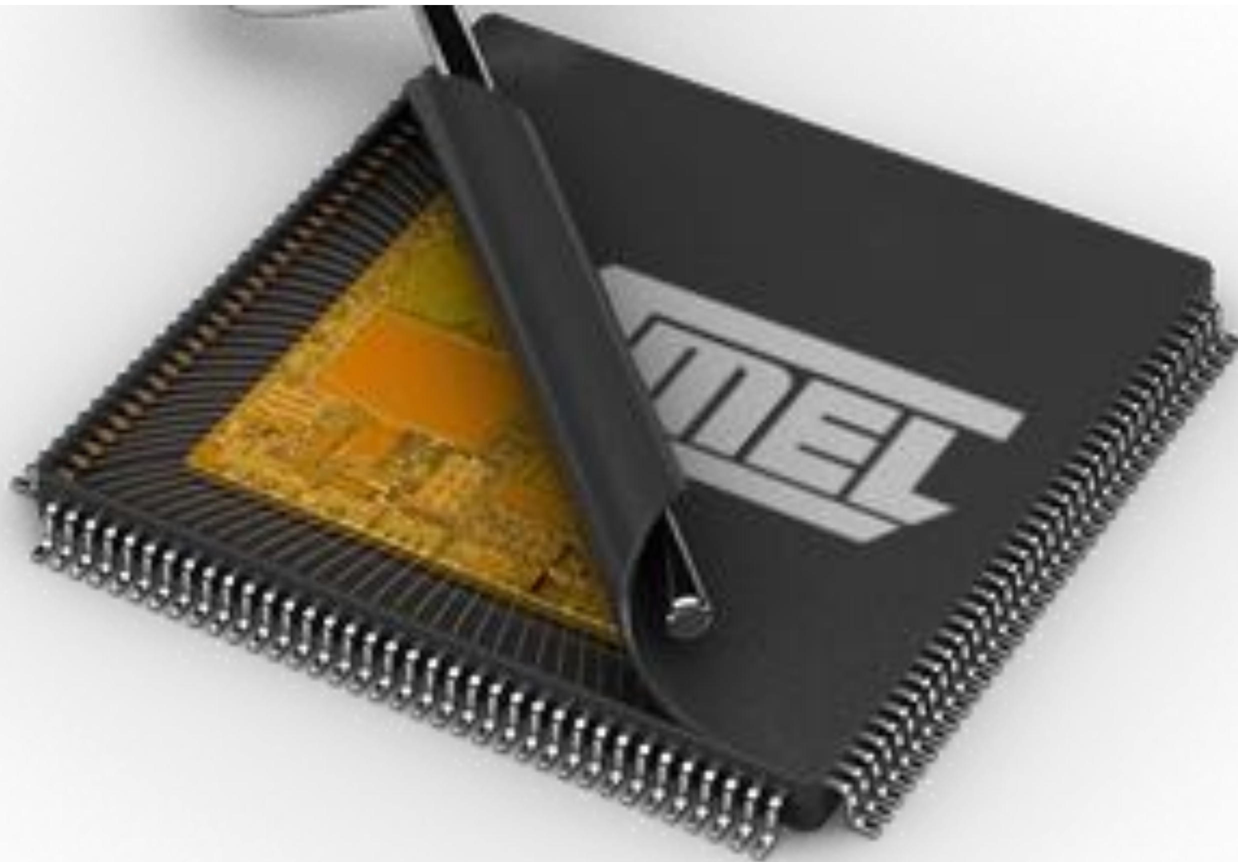


Photo credit Mayank Prasad, maxEmbedded.com

Instruction Set Architecture

"The Parts of the Engine"

- The Instruction Set Architecture (ISA) of a microprocessor includes all the registers that are accessible to the programmer. In other words, registers that can be modified by the instruction set of the processor.
- With respect to the AVR CPU illustrated in Figure 5.2, these ISA registers include the 32 x 8-bit general purpose registers, status register (SREG), the stack pointer (SP), and the program counter (PC).

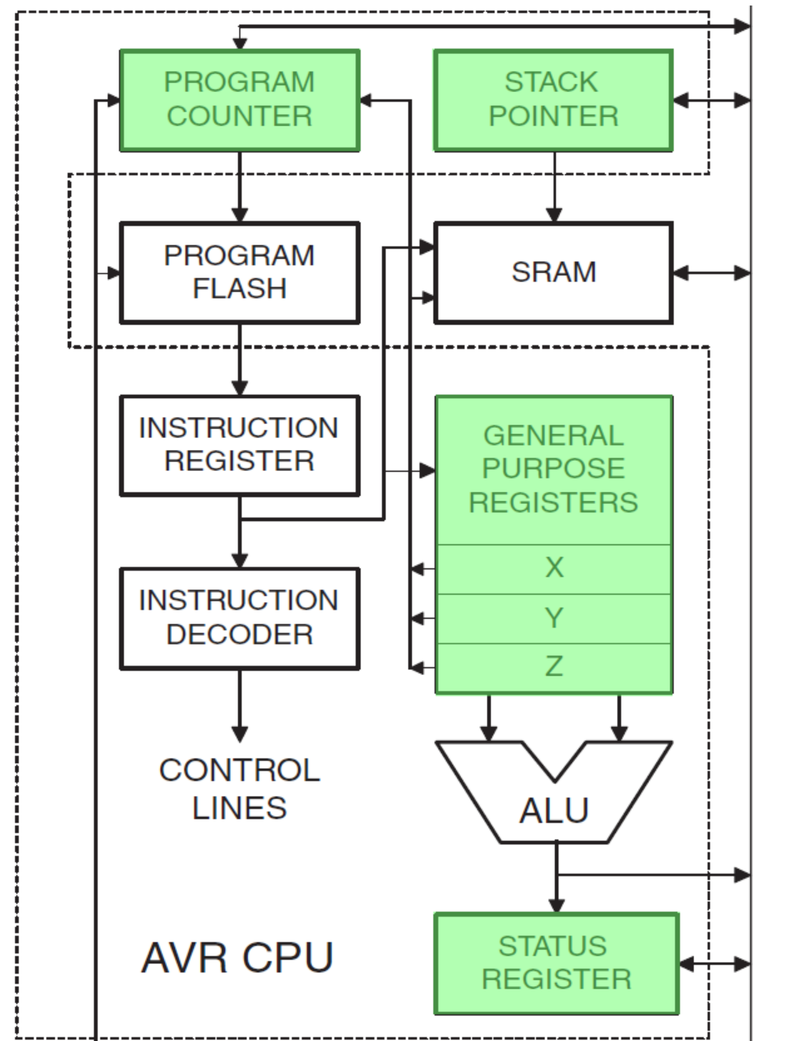


Figure 5-2: AVR Central Processing Unit ISA Registers[3]

[3] Source: ATmega16 Data Sheet http://www.atmel.com/dyn/resources/prod_documents/2466s.pdf page 3

AVR CPU CORE ARCHITECTURE[4]

"Features of the Engine"

Part I

- Reduced Instruction Set Computer (**RISC**): The instruction set of the computer and target compiler(s) are developed in concert allowing the optimization of both. In this way, a relatively high performance processor can be realized by "reducing" the amount of work any single instruction needs to do; leading to a simpler hardware design (smaller, faster, and cheaper).

8051 Microcontroller	**ATmega328P Microcontroller**
`cjne  A, 0x99,next`	`cmp   r16,0x99`
	`brne  next`

- *Mostly* **16-bit fixed-length instructions.** Instructions have from zero to two operands. Many of today's RISC microprocessors have up to three operands.
- **The Register File of the AVR CPU contains 32 x 8 bit** ***mostly*** **Orthogonal** (or identical) **General Purpose Registers** – instructions can use any register; therefore, simplifying compiler design.
- **Load-store memory access**. Before you can do anything to data, you must first *load* it from memory into one of the general-purpose registers. You then use **register-register** instructions to operate on the data. Finally, you *store* your answer back into memory.

[4] **Reading:** Section 5.2 AVR CPU Core

AVR CPU CORE ARCHITECTURE

"Features of the Engine"

Part II

- Modified **Harvard memory model**: A Harvard memory model separates Program and Data memory into separate physical memory systems (Flash and SRAM) that appear in different address spaces. A *Modified* Harvard memory model has the ability to read/write data items from/to program memory using special instructions. A Princeton memory model computer has only a single address space, shared by both the program and data.

- A **Two-stage Instruction Pipeline** (fetch and execute) resulting in *most* instructions being executed in one clock cycle. Consequently, the performance of a 20 MHz processor would approach 20 MIPS (Millions of Instructions Per Second). Compare this with the 8051 Complex Instructions Set Computer (CISC) computer which takes a minimum of 12 clock cycles to execute a single instructions (12 MHz clock = 1 MIPS).

- Simplicity of the computer architecture translates to a faster learning curve and utilization of the machine by the student.

AVR CPU INSTRUCTIONS

"The Language of the Machine"

The Instruction Set of our AVR CPU can be functionally divided (or classified) into:

1. Data Transfer
2. Arithmetic and Logical
3. Bit and Bit-Test
4. Control Transfer (Branch Instructions) — "Load the Program Counter"
5. MCU Control — **nop**, **sleep**, **wdr**, break

- Data Transfer instructions are used to Load and Store data to the General Purpose Registers, also known as the *Register File*.
    - Exceptions are the push and pop instructions which modify the Stack Pointer.
    - By definition these instructions do not modify the status register (SREG).
- Arithmetic and Logic Instructions plus Bit and Bit-Test Instructions use the ALU[5] to operate on the data contained in the general purpose registers[6].
    - Flags contained in the Status Register (SREG) provide important information concerning the results of these operations.
    - For example, if you are adding two signed numbers together, you will want to know if the answer is correct. The state of the overflow flag (OV) bit within SREG gives you the answer to this question (1 = error, 0 no error).
- As the AVR processor fetches and executes instructions it automatically increments the program counter (PC) so it always points at the **next instruction to be executed.** Control Transfer Instructions allow you to change the contents of the PC either conditionally or unconditionally.
    - Continuing our example if an error results from adding two signed numbers together we may want to conditionally (OV = 1) branch to an error handling routine.

[5] Implemented using combinational logic

[6] Implemented using sequential logic

INSTRUCTION FETCH AND EXECUTE

"The Basic Cycles of the Engine"

Once built, our computer lives to **Fetch and Execute** instructions, the bread-and-butter of the computer programmer. For this reason, the programmer views the computer as a vehicle for executing a set of instructions. This perspective is codified by the *Instruction Set Architecture* (ISA) of the computer.

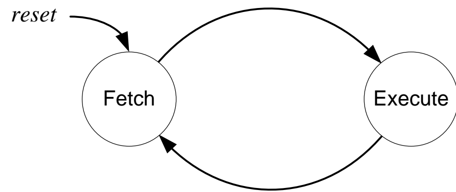


Figure 3: The Two Basic States of all Microprocessor

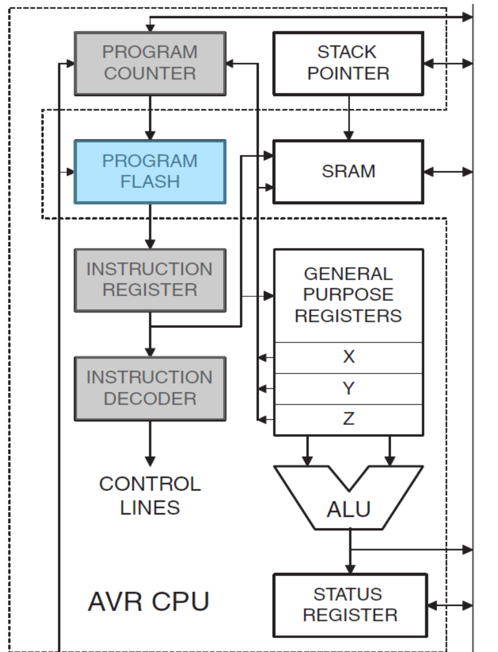


Figure 4: AVR CPU Registers and Logic used to Fetch and Execute an Instruction[7]

[7] Source: ATmega16 Data Sheet http://www.atmel.com/dyn/resources/prod_documents/2466s.pdf page 3

Harvard versus Princeton Memory Model Instruction Fetch Cycle

1. The CPU presents the value of The program counter (PC) on the address bus and sets the read control line.
2. The Flash program memory looks up the address of the instruction and presents the value on the data bus.
3. The value from the data bus is placed into the instruction register and the CPU clears the read control line. The instruction register now holds the instruction to be executed.
4. The program counter is incremented so it points to the next instruction to be executed.
5. The instruction decoder interprets and implements (executes) the instruction.

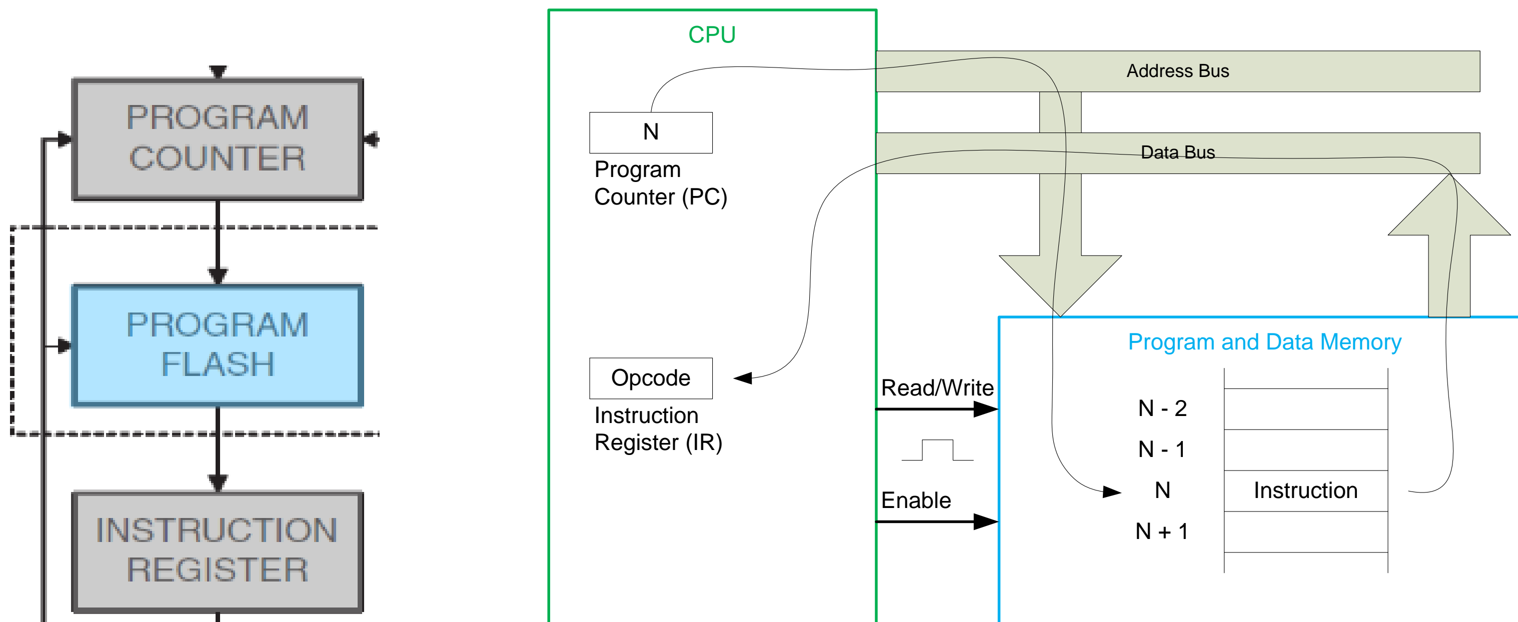


Figure 5: Bus Activity for an Instruction Fetch Cycle for Harvard (left) and Princeton (right) Memory Models

I/O Address Space versus Memory Mapped I/O

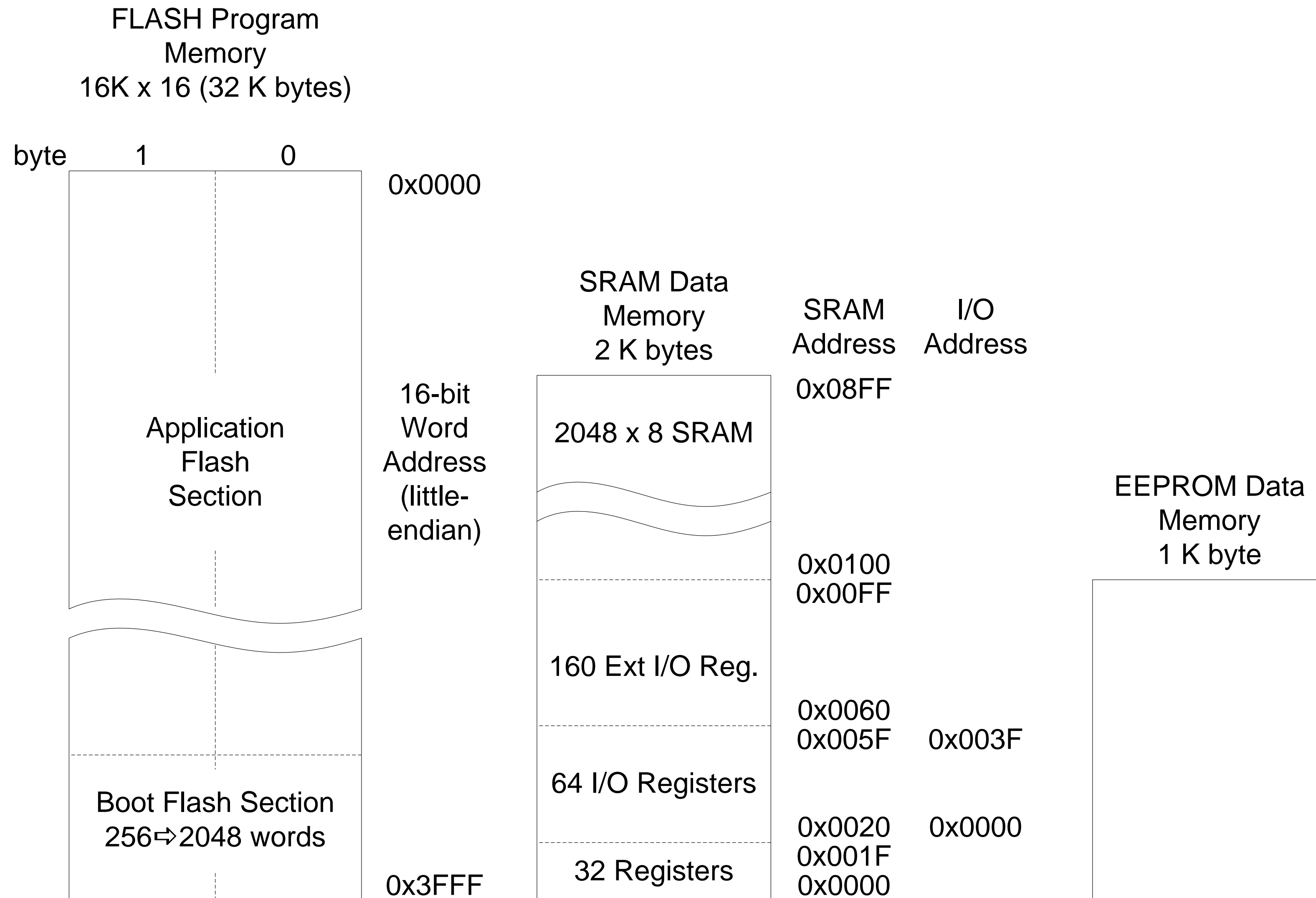
- Input and Output ports have traditionally been treated as separate parts of the computer.
- The AVR includes an `in` instruction to read from an I/O port and an `out` instruction to write to an I/O port.
- The AVR has 64 I/O registers accessible to these two instructions

Problem: The Atmel ATmega line of Microcontrollers needs more than 64 I/O registers (GPIO, Timers, …)

Solution: Instead of looking at computers having 5 basic elements (Input, Output, ALU, CPU, Memory), you can simplify the design to only three (CPU, ALU, and Memory) now allowing the CPU to access 160 "extended" I/O registers using SRAM instructions like `lds` (load from SRAM) and `sts` (store to SRAM).

- This was such a powerful technique that Atmel extended the I/O mapping to include the 32 general purpose registers, the original 64 I/O registers, and the 160 extended I/O registers. The overlaying of the I/O address space with the SRAM address space is shown in the next slide.
- A side benefit of the double mapping is the large number of ways of accessing data within SRAM (addressing modes) versus the limited number of instructions and addressing modes available for accessing the original 64 I/O registers (i.e., `in`, `out`).
- It is very important to realize that I/O registers are not contiguous within the address space (I/O or SRAM). The mapping is simply a convenient way of accessing registers physically located in diverse locations within the Silicon chip.

ATMEL ATMEGA328P MEMORY MODEL[8]



[8] Source: ATmega328P Data Sheet http://www.atmel.com/dyn/resources/prod_documents/8161S.pdf Chapter 7. AVR Memories Figure 2-2

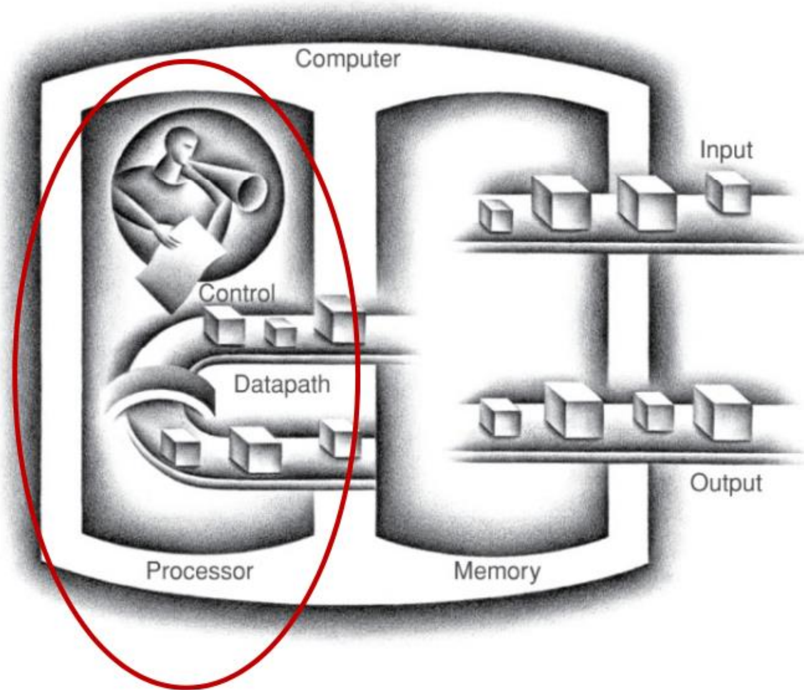
ATmega328P I/O Memory Map[9]

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
0x29 (0x49)	Reserved	–	–	–	–	–	–	–	–	
0x28 (0x48)	OCR0B	Timer/Counter0 Output Compare Register B								
0x27 (0x47)	OCR0A	Timer/Counter0 Output Compare Register A								
0x26 (0x46)	TCNT0	Timer/Counter0 (8-bit)								
0x25 (0x45)	TCCR0B	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	
0x24 (0x44)	TCCR0A	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00	
0x23 (0x43)	GTCCR	TSM	–	–	–	–	–	PSRASY	PSRSYNC	143/165
0x22 (0x42)	EEARH	(EEPROM Address Register High Byte) 5.								21
0x21 (0x41)	EEARL	EEPROM Address Register Low Byte								21
0x20 (0x40)	EEDR	EEPROM Data Register								21
0x1F (0x3F)	EECR	–	–	EEPM1	EEPM0	EERIE	EEMPE	EEPE	EERE	21
0x1E (0x3E)	GPIOR0	General Purpose I/O Register 0								25
0x1D (0x3D)	EIMSK	–	–	–	–	–	–	INT1	INT0	72
0x1C (0x3C)	EIFR	–	–	–	–	–	–	INTF1	INTF0	72
0x1B (0x3B)	PCIFR	–	–	–	–	–	PCIF2	PCIF1	PCIF0	
0x1A (0x3A)	Reserved	–	–	–	–	–	–	–	–	
0x19 (0x39)	Reserved	–	–	–	–	–	–	–	–	
0x18 (0x38)	Reserved	–	–	–	–	–	–	–	–	
0x17 (0x37)	TIFR2	–	–	–	–	–	OCF2B	OCF2A	TOV2	163
0x16 (0x36)	TIFR1	–	–	ICF1	–	–	OCF1B	OCF1A	TOV1	139
0x15 (0x35)	TIFR0	–	–	–	–	–	OCF0B	OCF0A	TOV0	
0x14 (0x34)	Reserved	–	–	–	–	–	–	–	–	
0x13 (0x33)	Reserved	–	–	–	–	–	–	–	–	
0x12 (0x32)	Reserved	–	–	–	–	–	–	–	–	
0x11 (0x31)	Reserved	–	–	–	–	–	–	–	–	
0x10 (0x30)	Reserved	–	–	–	–	–	–	–	–	
0x0F (0x2F)	Reserved	–	–	–	–	–	–	–	–	
0x0E (0x2E)	Reserved	–	–	–	–	–	–	–	–	
0x0D (0x2D)	Reserved	–	–	–	–	–	–	–	–	
0x0C (0x2C)	Reserved	–	–	–	–	–	–	–	–	
0x0B (0x2B)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	93
0x0A (0x2A)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	93
0x09 (0x29)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	93
0x08 (0x28)	PORTC	–	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	92
0x07 (0x27)	DDRC	–	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	92
0x06 (0x26)	PINC	–	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	92
0x05 (0x25)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	92
0x04 (0x24)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	92
0x03 (0x23)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	92
0x02 (0x22)	Reserved	–	–	–	–	–	–	–	–	
0x01 (0x21)	Reserved	–	–	–	–	–	–	–	–	
0x0 (0x20)	Reserved	–	–	–	–	–	–	–	–	

[9] Source: ATmega328P Data Sheet http://www.atmel.com/dyn/resources/prod_documents/8161S.pdf Chapter 30 Register Summary

APPENDIX A PROCESSOR CONTROL AND DATAPATH[10]

Control	Datapath
Component of the processor that commands the datapath, memory, data, I/O devices according to the instructions of the memory	Components of the processor that perform arithmetic operations and holds data



[10] https://www.ida.liu.se/~TDTS10/info/lectures/Lecture3.pdf

APPENDIX B CALCULATING THE LAST ADDRESS

Given a 16K word (2 bytes / word) memory, what is the last address, in hexadecimal?

- The range of memory addresses, like an unsigned number, is from 0 → $2^n - 1$
- We are given the size of our memory in decimal as $16K_{10}$. So the first step is to convert this number to a power of 2.

  $16K_{10} = 2^4 * 2^{10} = 2^{14}$, which in binary would be…

- Which then can directly be expressed as a binary number.

2^{14}	2^{13}	…	2^0	
1	0	…	0	, …1 followed by 14 zeros
			−1	
0	1		1	, or 14 ones

- So the answer is 0x3FFF

- As a short-cut, if you can convert the memory size to a power of 2, the exponent equals the number of 1 in the answer. By dividing the exponent by 4, you have the number of hex digits which are F (1111_2), with the remainder giving you the most significant hex digit. In our example 4 goes into 14, 3 times with a remainder of 2, where 2 ones (0011_2) equal hexadecimal 3_{16}.

Appendix C I/O Address Space versus Memory Mapped I/O

Reading: Your textbook covers memory organization in Section 0.3 “Semiconductor Memory” and I/O Mapping in Section 2.2 “The AVR Data Memory.” The following material covers mapping of the I/O address space in a slightly different way. The material was provided in bullet form earlier in this document.

From Charles Babbage’s Analytical Engine to Dr. Jon Von Neumann’s paper on the EDVAC computer, Input and Output have been treated as separate parts of the computer. Input and Output parts of your PC include the keyboard, mouse, printer, display, etc. To support these “peripheral” devices many microprocessors include a separate I/O address space and instructions for working with the registers contained used to control and access data provided by the peripheral device. For the AVR microcontroller you read an I/O register using an `in` instruction and write using the `out` instruction. When Atmel adopted the AVR architecture, they discovered that the 64 I/O registers accessible to these two instructions was insufficient for all the peripheral devices that they were planning on adding to the ATmega line of Microcontrollers. Specifically, they added 160 “extended” I/O registers. However, the AVR microprocessor was only designed for 64 I/O registers. To solve this problem, Atmel turned to an alternative way of working with I/O devices pioneered by Motorola and the 6800 family of processors (among others). Motorola realized that there was no reason to treat input and output devices any different from memory. Now instead of looking at computers having 5 basic elements (Input, Output, ALU, CPU, Memory), you could simplify the design to only three (CPU, ALU, and Memory). Now accessing the 160 “extended” I/O registers was accomplished using SRAM instruction like lds (load from SRAM) and sts (store to SRAM). This was such a powerful technique that Atmel extended the I/O mapping to include the 32 general purpose registers, the original 64 I/O registers, and the 160 extended I/O registers. The overlaying of the I/O address space with the SRAM address space is shown in the next slide.

A side benefit of the double mapping is the large number of ways of accessing data within SRAM (addressing modes) versus the limited number of instructions and addressing modes available for accessing the original 64 I/O registers.

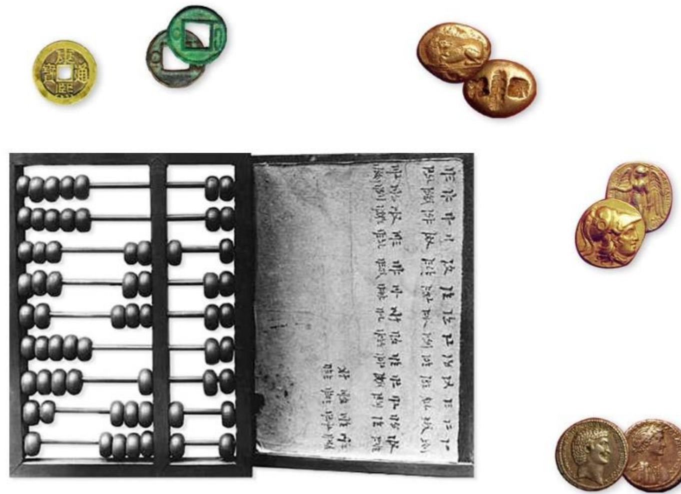
It is very important to realize that I/O registers are not contiguous within the address space (I/O or SRAM). The mapping is simply a convenient way of accessing registers physically located in diverse locations within the Silicon chip.

Appendix D A Brief History of the Computer

4,000 to 3,000 BC Abacus (+, -, *, /)

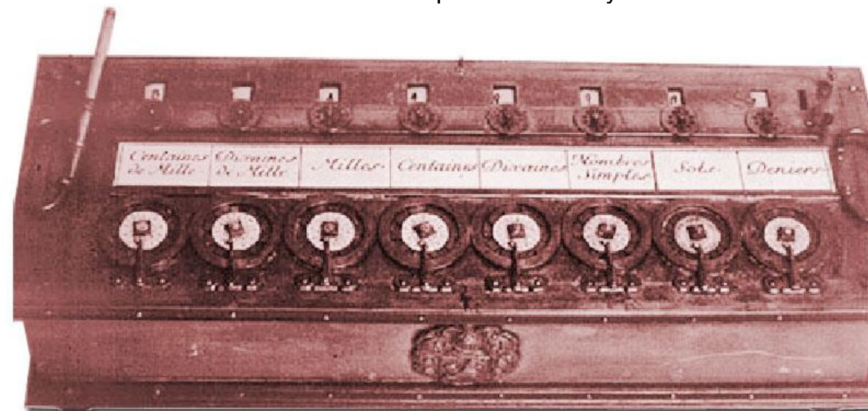
- The abacus is an instrument used to perform arithmetic calculations. The positions of beads on a set of wires determine the value of the digit. Romans called these beads calculi the plural of calculus, meaning pebble. This Latin root gave rise to the word calculate. In one contest the Abacus easily won over a mechanical calculator. The abacus is still used in China, Japan, and Korea.

gastropod (snail)



1642 Blaise Pascal Mechanical Calculator (+, -)

- Designed at the age of 20. Rotating wheel mechanical calculator with automatic carry between digits on addition and subtraction of decimal digits (like the odometer in a car). In 1671 Baron von Leibnitz created a calculator, which could add, subtract, and multiply.
- A Human Computer with a mechanical calculator can execute 500 operations a day

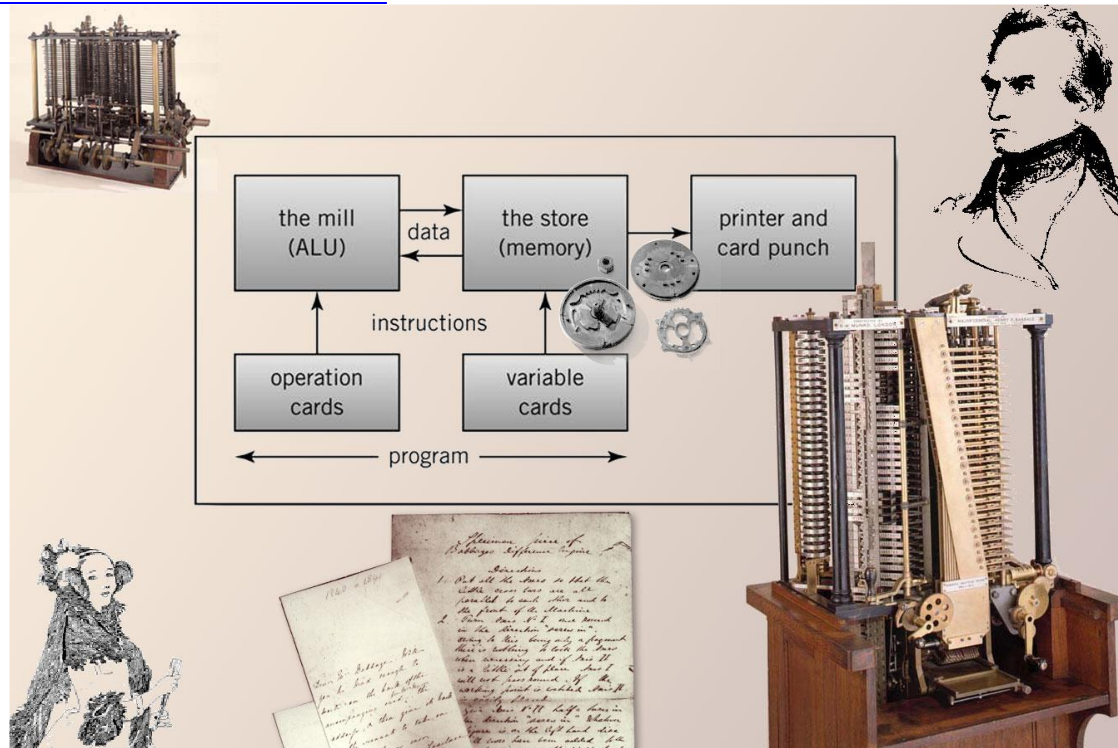


1833 Charles Babbage and the Analytical Engine

- Conceived by Babbage, the engine established the basic principles upon which modern general-purpose digital computers are constructed. This **mechanical** machine performed instructions dictated by punched cards, with the variable values being determined by a second set of cards. The punched cards came from Joseph marie Jacquard's loom, where they controlled the operation of the weaving machines in 1812.
- Neither the Analytical Engine or Difference Engine (1820), a special purpose computer designed to solve polynomial expressions (ex. $N^2 + N + 41$), were ever entirely completed by Babbage known as "the irascible genius." The difference engine has recently been built as shown here.

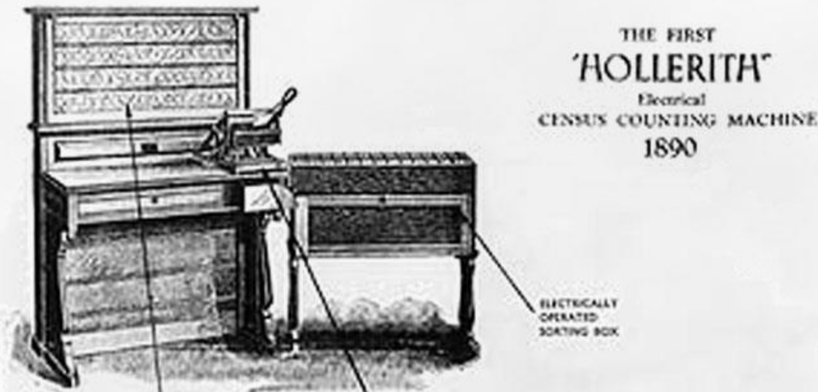
1843 Ada Byron and the First Computer Program

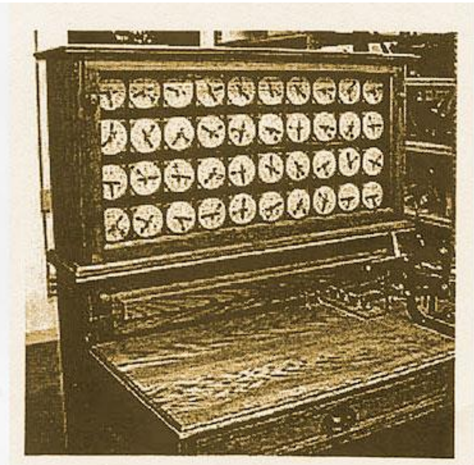
- Ada Byron, Lady Lovelace, was one of the most picturesque characters in computer history. Augusta Ada Byron was born December 10, 1815 the daughter of the illustrious poet, Lord Byron. Ada was brought up to be a mathematician and scientist. It was at a dinner party at Mrs. Somerville's that Ada heard in November 1834, Babbage's ideas for a new calculating engine, the Analytical Engine. Ada, in 1843, married to the Earl of Lovelace and the mother of three children under the age of eight, wrote an article describing Babbage's Analytical Engine. Lady Lovelace's prescient comments included her predictions that such a machine might be used to compose complex music, to produce graphics, and would be used for both practical and scientific use. When inspired Ada could be very focused and a mathematical taskmaster. Ada suggested to Babbage writing a plan for how the engine might calculate Bernoulli numbers. This plan, is now regarded as the first "computer program." Like her father, she died at 36, Ada anticipated by more than a century most of what we think is brand-new computing.
  Source: http://www.scottlan.edu/lriddle/women/love.htm

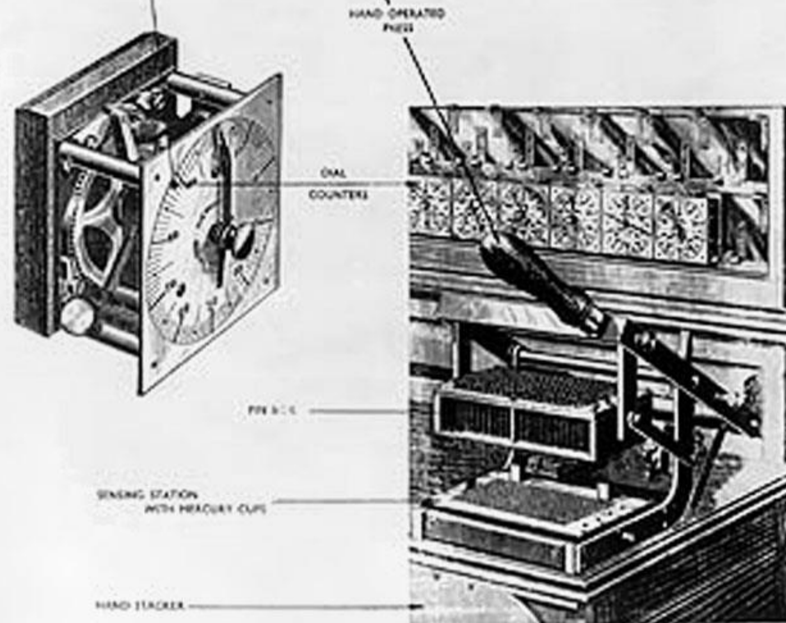


1890 Herman Hollerith and the Census Counting Machine

- Hollerith developed punched cards for tabulating equipment used in the 11th census of the United States. Cards contained 288 locations, size of dollar bill in order to save on tooling. Contact brushes completed electrical circuits allowing the system to do: counting, sensing, punching, and sorting. Started Tabulating Machine Company, which turned into the Computer-Tabulating-Recording Company, which turned into the International Business Machine Corporation (IBM) in 1924.

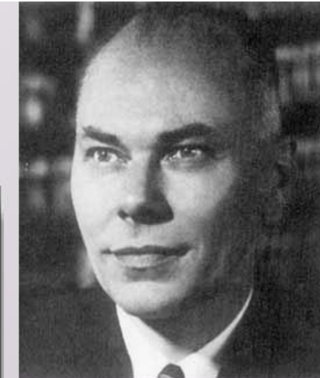


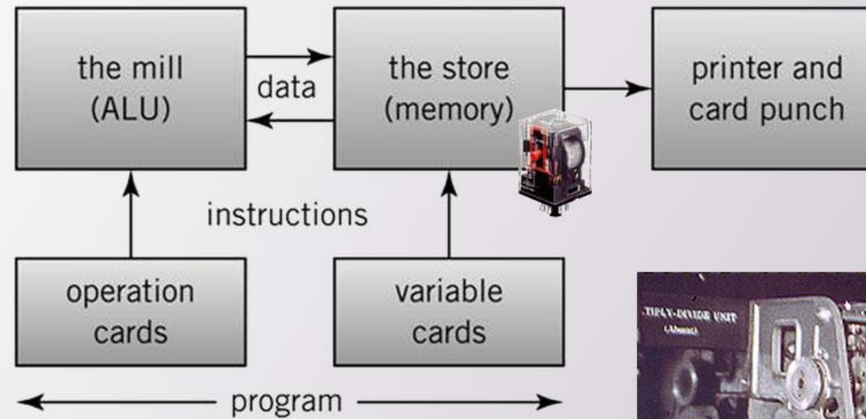


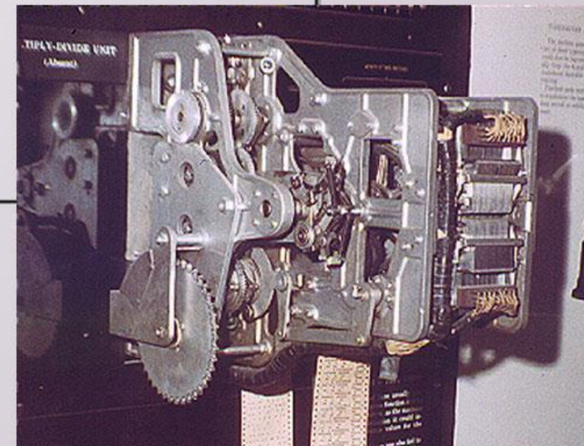


1937 Harvard Mark I

- Howard Hathaway Aiken at **Harvard** proposed to IBM the Mark I or Automatic Sequence Controlled Calculator — this was to be the first large-scale calculator. Very similar to the Analytical engine, the machine used a combination of **electromechanical** devices, including many relays. It went to work in 1944 calculating with numbers of 23 digits and computer products of 46-digit accuracy. It received its instructions from perforated tape, from IBM cards, and from the mechanical setting of 1,440 dial switches. Output was either by IBM cards or by typing columns of figures on a roll of paper. The Mark I could perform one division per minute. The machine was in operation for many years, generating many tables of mathematical functions (particularly Bessel functions), and was used for trajectory calculations in World War II.

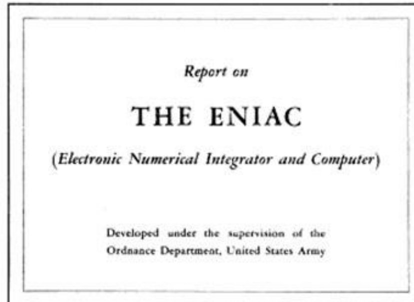






1943 Electronic Numerical Integrator and Computer (ENIAC)

- Engineers J. Presper Eckert and John W. Mauchly created the ENIAC at the Moore School of Engineering of the University of Pennsylvania between 1943-1946. Built in war time secrecy for the army ordnance department, the ENIAC was designed to do Trajectory calculations. Containing 18,000 **vacuum tubes**, each accumulator using 100 vacuum tubes arranged as 10 columns of 10 tubes each, the ENIAC could add two 10-digit numbers (the size of ENIAC's decimal accumulators) in 200 microseconds. Thirty thousand (30,000) times faster than the Mark I. The ENIAC was programmed by patch board and switches. The ENIAC was later moved at a cost of $100,000 to the Ballistic Research Laboratories at the Aberdeen Proving Ground.


Report on

THE ENIAC

(*Electronic Numerical Integrator and Computer*)

Developed under the supervision of the Ordnance Department, United States Army

TECHNICAL REPORT I

Volume I

(Bound in two volumes)

UNIVERSITY OF PENNSYLVANIA

Moore School of Electrical Engineering

PHILADELPHIA, PENNSYLVANIA

June 1, 1946

WAR DEPARTMENT

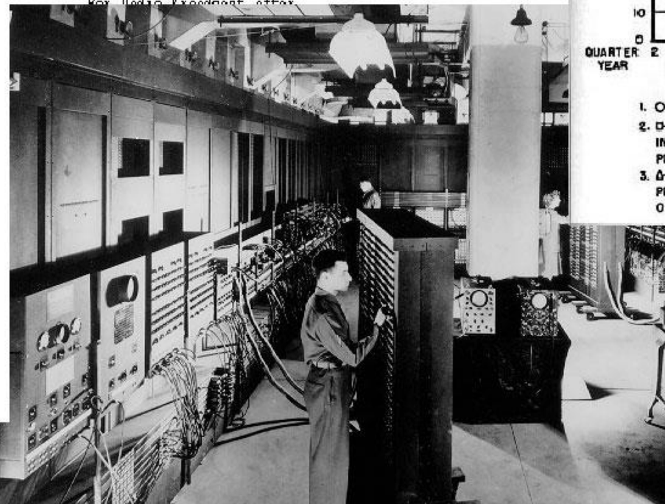
Bureau of Public Relations

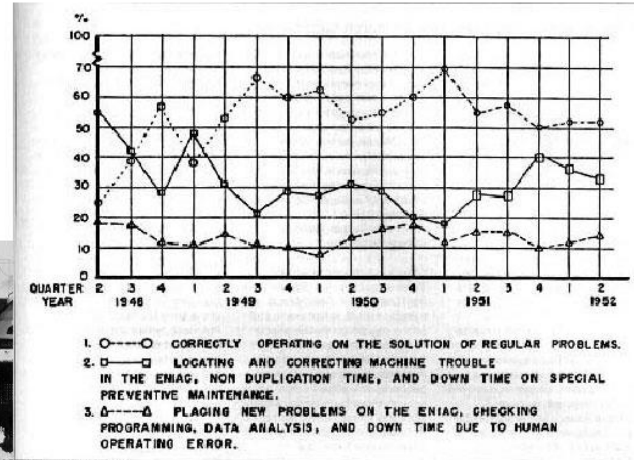
PRESS BRANCH

Tel. - RE 6700

Brs. 3425 and 4860

FOR RELEASE SATURDAY A.M., FEBRUARY 16, 1946

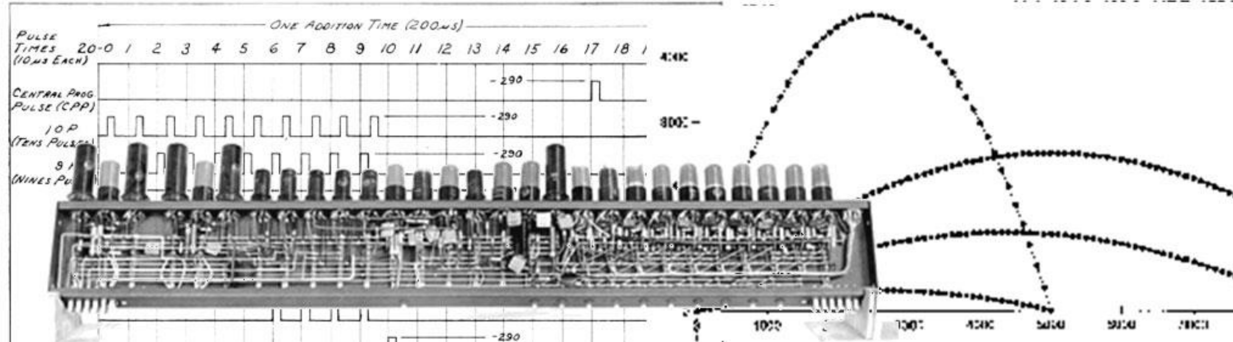




QUADRANT ELEVATION - MIL

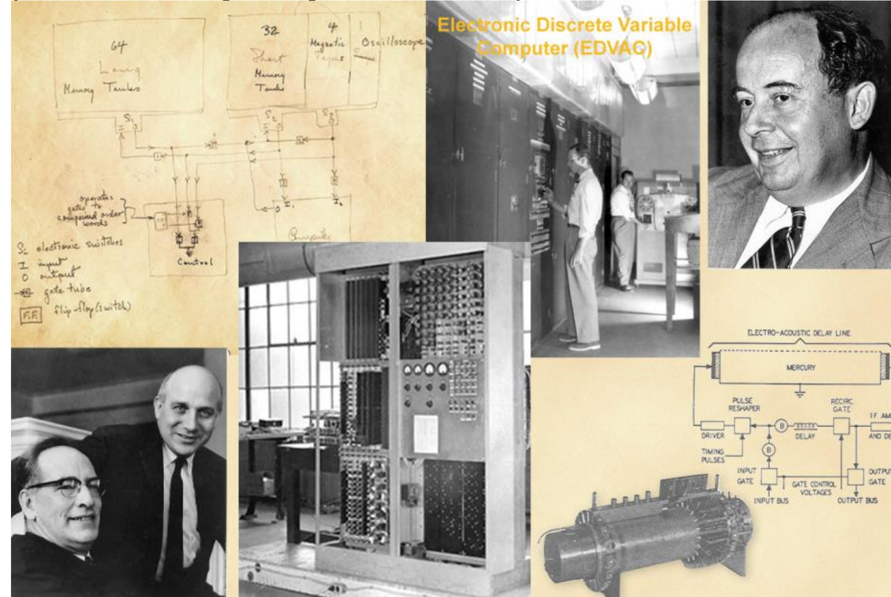
Height of Target - Meters

Range Meters	-400	-300	-200	-100	0	100	200	300	400
10000	318.3	329.9	341.6	353.2	364.9	376.6	388.4	400.1	412.0
10100	323.1	334.6	346.1	357.7	369.4	381.0	392.7	404.5	416.2
10200	327.8	338.3	350.8	362.3	373.9	385.6	387.1	408.8	420.5
10300	332.5	344.0	355.5	366.9	378.4	380.0	401.6	413.2	424.9

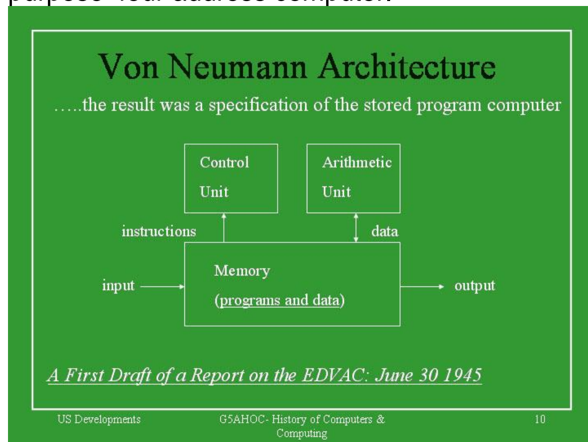


1945 Dr. John Von Neumann and the Electronic Discrete Variable Computer (EDVAC)

- EDVAC was the first general-purpose stored program binary electronic (vacuum tube) computer. Completed in 1950 after the EDSAC thus it was not the first operational stored program computer. The technical work done on the EDVAC was by Eckert and Mauchly, Notable the Ultrasonic (or Supersonic) Delay Line, with the logical organization done by Von Neumann, Burke, and Goldstine.

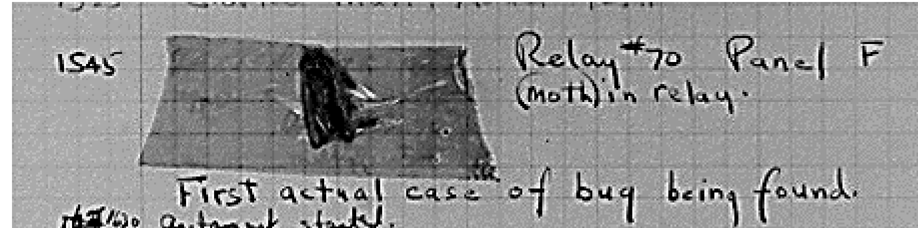


- This computer was the blueprint for most modern day computer systems having in it the 5 principle organs that make up almost all modern day computers. **Input, Output, Arithmetical, Central Control, Memory** (storing both the numerical as well as the instructional information for a given problem), Eckert as well as others left before the EDVAC was ever completed. Architecturally the EDVAC is classified as a general purpose four address computer.



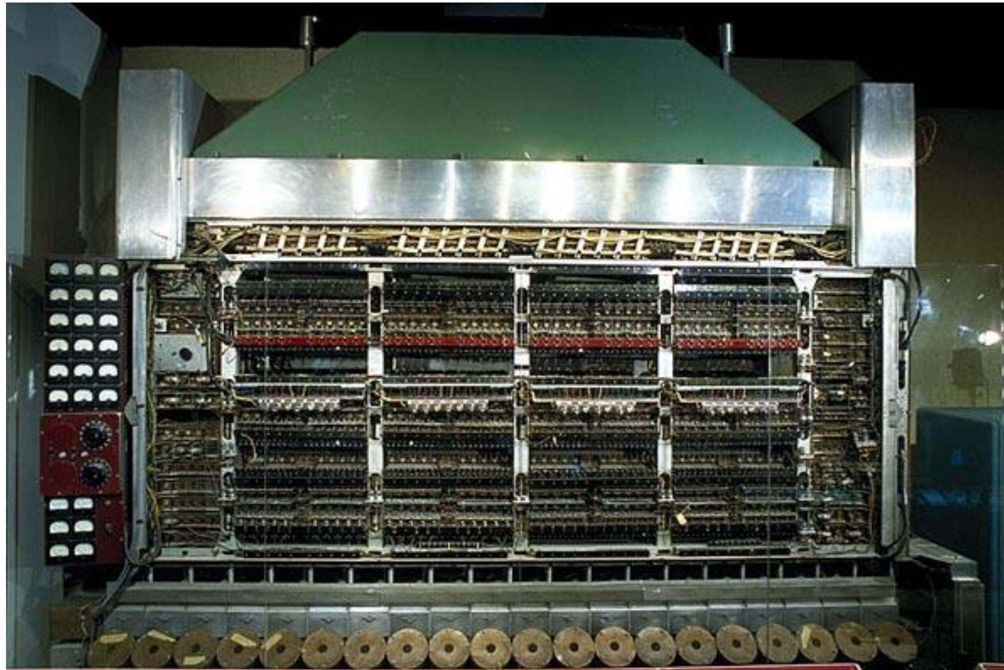
1947 The First Computer Bug

- American engineers have been calling small flaws in machines "bugs" for over a century. Thomas Edison talked about bugs in electrical circuits in the 1870s. When the first computers were built during the early 1940s, people working on them found bugs in both the hardware of the machines and in the programs that ran them.
- In 1947, engineers working on the Mark II computer at Harvard University found a moth stuck in one of the components. They taped the insect in their logbook and labeled it "first actual case of bug being found." The words "bug" and "debug" soon became a standard part of the language of computer programmers.



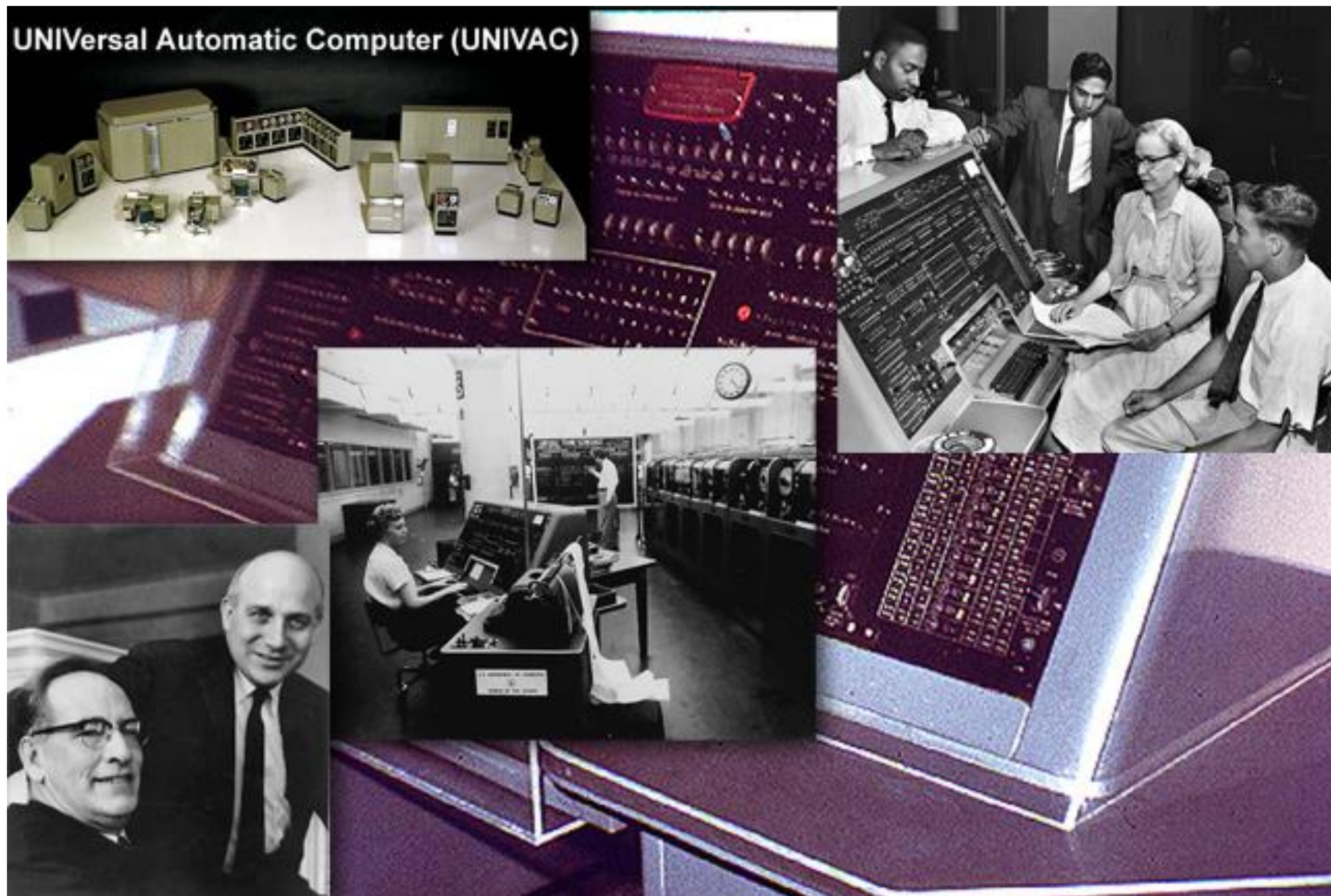
1951 John Von Neumann and Princeton's IAS (Institute for Advance Study) Machine

- Designed to develop a world weather model, the IAS machine incorporated most of the general concepts of parallel binary stored-program computers. That is it used random access memory or parallel memory, CRTs. One address computer.



1951 Eckert and Mauchly and the UNIVAC I

- Soon after the formal dedication of ENIAC computer, J. Presper Eckert and John W. Mauchley's left the University of Pennsylvania to start their own business. Early orders from U.S. government agencies and other potential customers were not enough to keep the young Eckert-Mauchley Computer Corporation alive, and Remington Rand agreed to purchase the firm in 1950. Work on the **UNIVAC I** (Universal Automatic Computer) went forward, and the first commercially available **electronic** (vacuum tube) digital computer was delivered to the Bureau of the Census in early 1951. By 1957, some 46 copies of the machine had been installed at locations ranging from the David Taylor Model Basin of the U.S. Navy Bureau of Ships, to Pacific Mutual Life Insurance Company, to the offices of the Commonwealth of Pennsylvania.
- The UNIVAC, like the ENIAC, had vacuum tube circuit elements. There also were some 18,000 crystal diodes. Central memory was handled in acoustic delay-line tanks, which were used in several early computers. UNIVAC also had an external magnetic tape memory, as well as magnetic tapes used in input and output. Users of UNIVAC played an important role in the development of programming languages. *Source*: Smithsonian Computer History Collection



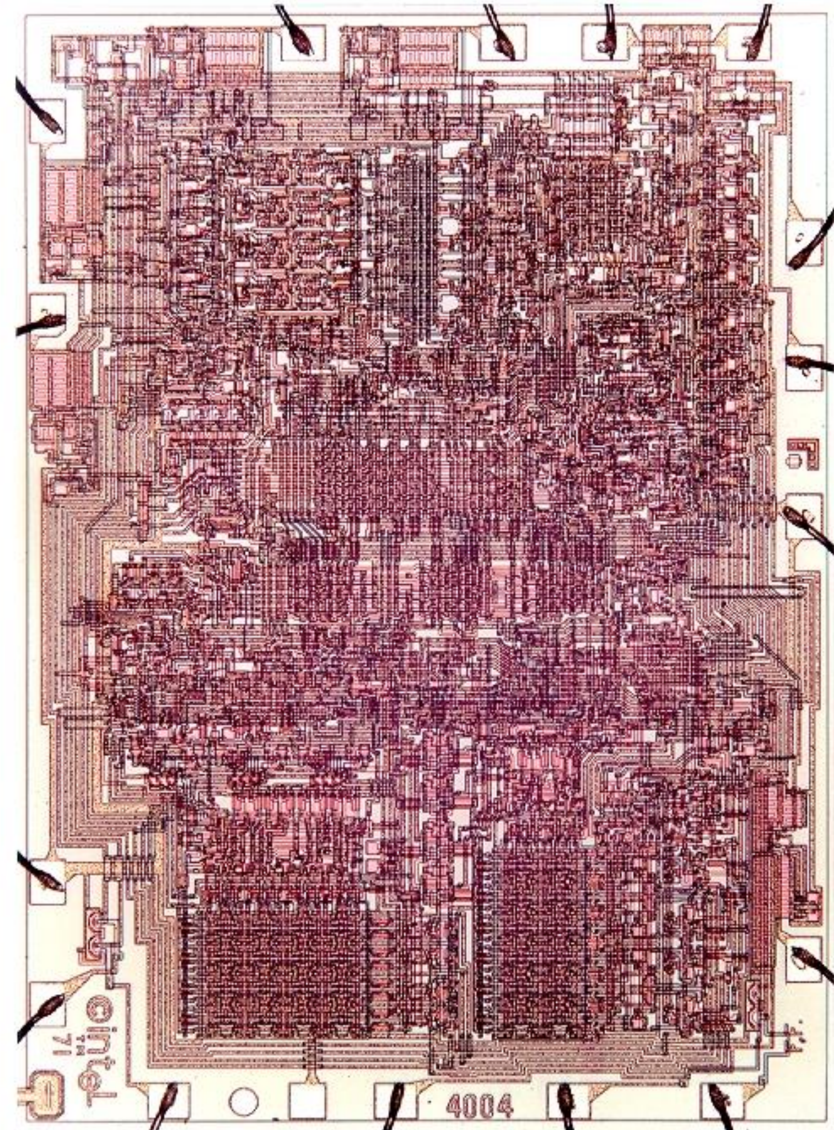
1965 Digital Equipment Corporation (DEC) PDP-8

- Designed using **Integrated Circuits**, DEC sold the first PDP-8 for only $18,000. Later versions of this machine that incorporated improvements in electronics appeared over the next decade. These became steadily smaller and cheaper, triggering a rush of new applications in which the computer was **embedded** into another **system** and sold by a third party (called an Original Equipment Manufacturer, or OEM). Some machines were specifically designed for time sharing and for business applications. Ultimately over 50,000 PDP-8's were sold (excluding those embedded as single chips into other systems) bringing computers into the laboratory and the manufacturing plant's production line, and thus the **minicomputer** industry was born. (read "The Sole of a New Machine").

The x86 isn't all that complex — it just doesn't make a lot of sense

Mike Johnson
Leader of the 80x86 Design at AMD
Microprocessor Report (1994)

June 1969 to April 1971 Ted Hoff and Intel 3-chipset 4004



- Intel, a company founded in 1968, is asked by Busicom of Japan to design a custom LSI calculator chip-set. Intel discovers design will take 11 36-40 pin IC packages and proposes a creative alternative. Ted Hoff, at Intel, had been working with the PDP-8 min-computer and proposed to Busicom that a general purpose LSI chip-set be designed that could be programmed to be a calculator or for other applications. We are so used to using computers, that the genus of this step can escape us. The traditional solution was to design what you wanted using logic gates. What Ted Hoff envisioned was a wholly different approach. You design a simple CPU and taught it using software to do what you want. Today these computers are known a microcontrollers and embedded systems. Publicly announced on November 1971.

Nov 1969 to Jan 1972 Vic Poor and the Intel 8008

- Vic Poor of Datapoint Corporation of San Antonio, Texas (manufacturers of "intelligent terminals" and small computer systems) along with Cogar and Viatron engineers design a very elementary computer, and put under contract Intel and Texas Instruments to implement the design on a single logic chip. Intel succeeded, but their product executed instructions approximately ten (10) times as slowly as Datapoint had specified and way behind schedule (work had been stopped by Intel to complete the Busicom chip-set.); so Datapoint declined to buy it, and built their own product using existing logic components. And thus Intel holding a computer-like logic device (whose development had been paid for) marketed the Intel 8008 and the microcomputer industry was born.

1975 John Cocke and the **IBM 801**

- The first (Reduced Instruction Set Computer) RISC machine was developed as part of the IBM 801 Minicomputer Project. John Cocke contributed many detailed innovations in the 801 processor and associated optimizing compiler, and is considered the "father of RISC architecture."

- “John's concept of the RISC resulted from his detailed study of the trade-offs between high performance machine organization and compiler optimization technology. He recognized that an appropriately defined set of machine instructions, program controls, and programs produced by a compiler -- carefully designed to exploit the instruction set -- could realize a very high performance processor with relatively few circuits. Critical to the success of RISC was the concept of an optimizing compiler able to use the reduced instruction set very efficiently and maximize performance of the machine.”
Source: http://domino.watson.ibm.com/comm/pr.nsf/pages/news.20020717_cocke.html

1976 Intel i8748

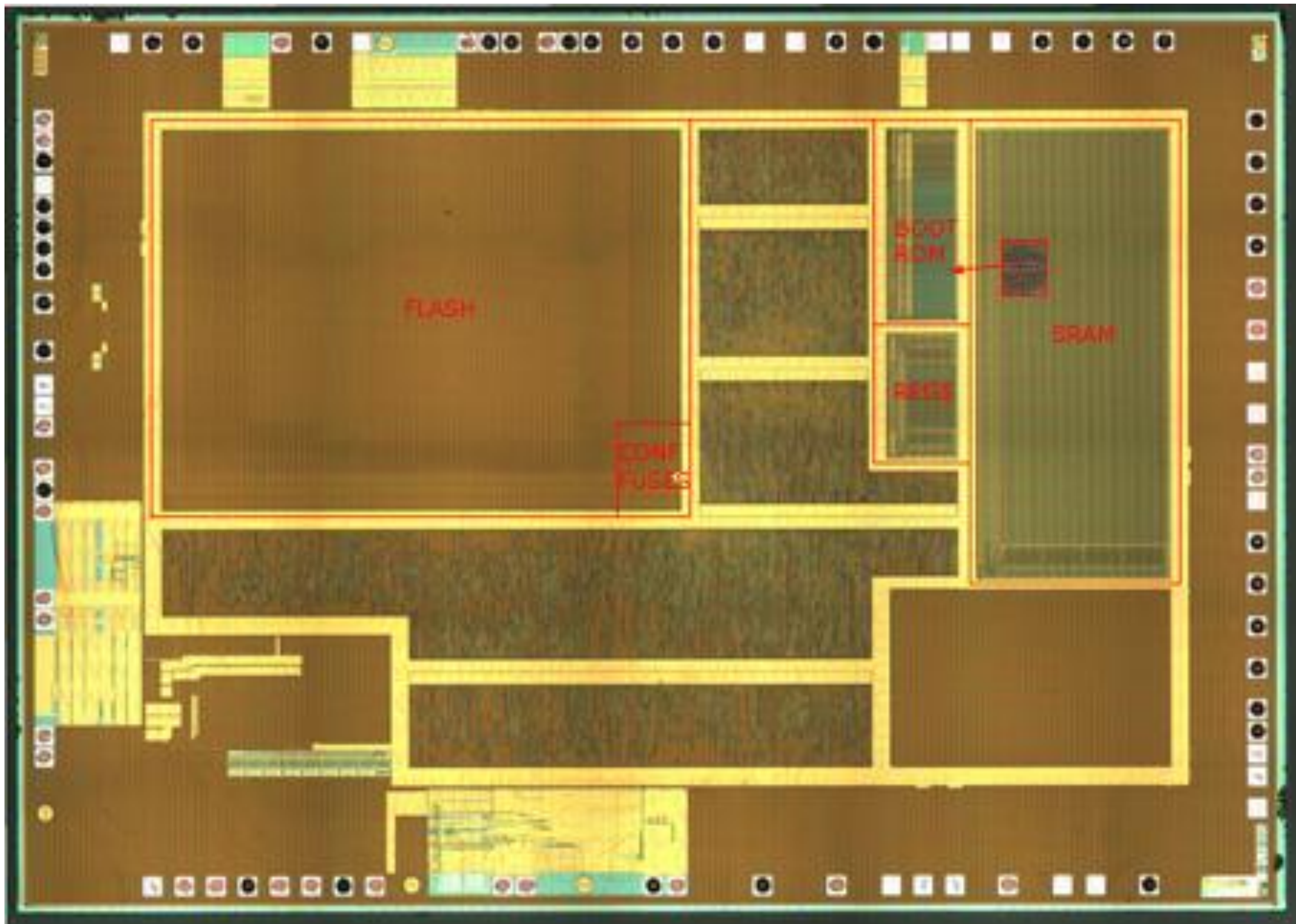
- Prior to 1976 small board computers (SBCs) were designed around microprocessor chips, like the 8080. These SBCs included all the features needed to implement a very simple computer system. These SBCs, of which the D2 by Motorola, KIM-1 by MOS Technology, and SDK-85 by Intel are the most memorable, quickly found their way into design labs at colleges, universities, and electronic companies. By adding peripheral cards these SBCs could read sensors and control actuators. In 1976 Intel put all of the features found on an SBC and parts of the peripheral cards into one chip known as the i8748. With over 17,000 transistors the i8748 was the first device in the MCS-48 family of microcontrollers. This IC, and other MCS-48 devices, quickly became the de facto industrial standard in control-oriented applications. Soon MCS-48 devices were replacing electromechanical components in many modern appliances.

1980 Intel 8051

- With over 60,000 transistors, the power, size, and complexity of microcontrollers moved to the next level with Intel’s introduction of the 8051, the first device in the MCS-51 family of microcontrollers. In a bold move, Intel allowed other manufacturers to make and market code-compatible variants of the 8051. This step led to its general acceptance by the engineering community as the de facto standard in microcontroller architectures.

1996 **Atmel AVR**

- AVR is a moniker for a family of Atmel 8-bit RISC microcontrollers. The AVR is a Modified Harvard architecture machine with program and data stored in separate physical memory systems that appear in different address spaces. The AVR architecture was conceived by Alf-Egil Bogen and Vegard Wollan at the Norwegian Institute of Technology (NTH). When the technology was sold to Atmel, the internal architecture was further developed by Alf and Vegard at Atmel Norway, a subsidiary of Atmel founded by the two architects. The name AVR sounds cool and does not stand for anything. *Source*: http://en.wikipedia.org/wiki/Atmel_AVR

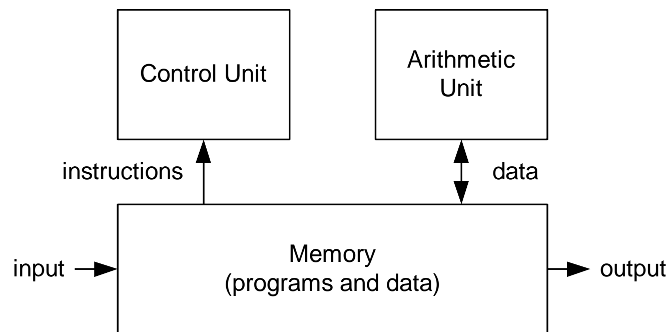

FLASH
BOOT ROM
SRAM
CONF FUSES

Appendix E Classic Computer Architecture

As we discovered in our short history lesson, computers are designed to meet a specific set of requirements. In the early days, these requirements were to meet some military, science, civil, or commercial need. For the military, it was predominately the calculation of ballistic tables; for science to calculate the motion of the planets or the weather. For civil keeping track of people and commercial keeping track of the money. To meet these requirements the computer was conceived and described by its (1) hardware components and (2) the instructions it could execute. The former, for all modern day computers, were codified by Von Neumann in his landmark paper describing the architecture of the EDVAC computer.

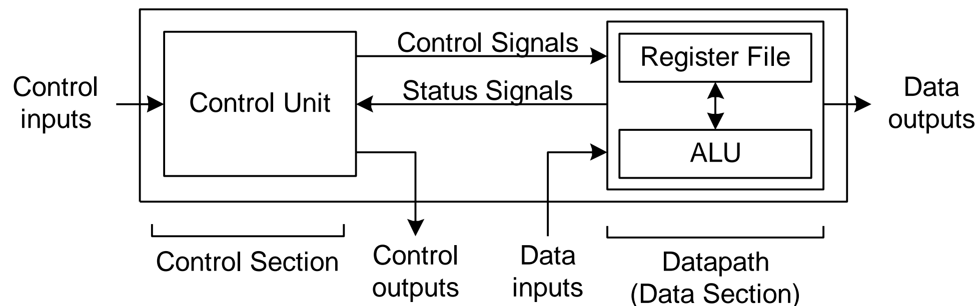
Von Neumann's paper describes a computer architecture having five basic components: Input, Output, Memory, Control, and Arithmetical.

Figure 1-1 A First Draft of a Report on the EDVAC: June 30, 1945



For this class we will ***Reparation*** these elements as discussed in the next section and defined in Figure 1-3. An important component of this new viewpoint is the **central processing unit** (CPU) which will be divided into a Control and a Datapath element as shown in the Figure 1-2. Atmel literature uses the term **microcontroller unit** (MCU) in place of the more generic central processing unit. In this course the two terms are considered synonymous.

Figure 1-2 High-level view of a CPU



Classic Microcontroller Architecture

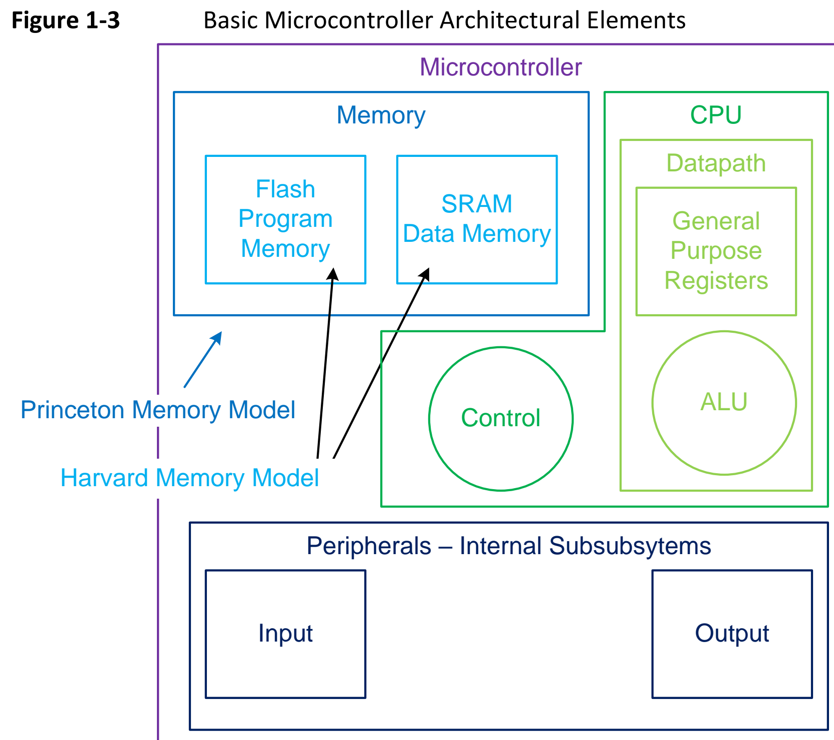
The CPU is divided into a Control and a Datapath element as shown in the Figure 1-2. The Control Unit contains combination logic and translates the instructions held in the instruction register (not shown) into the control signals needed to execute the instruction. The data path contains the **General Purpose Registers** (technically known as the **Register File**) and the **Arithmetic and Logic Unit** (ALU). The Datapath includes a few other registers which we will learn about shortly.

The integration of the program and data memory described by Von Neumann is today known as the **Princeton** memory model. The architecture of our AVR processor separates these two types of memory into **Flash Program Memory** and **Static Random Memory** (SRAM). This separation of program and data memory more resembles the Harvard Mark I computer, than the EDVAC computer, and is therefore known as the **Harvard** memory model.

The input and output functions of Figure 1-1 will be treated together and simply called input/output (I/O). For microcontrollers, the term I/O includes all the **Peripherals** (Parallel I/O, Counter/Timers, etc.) supported by a particular model of microcontroller, in our case the ATmega328P.

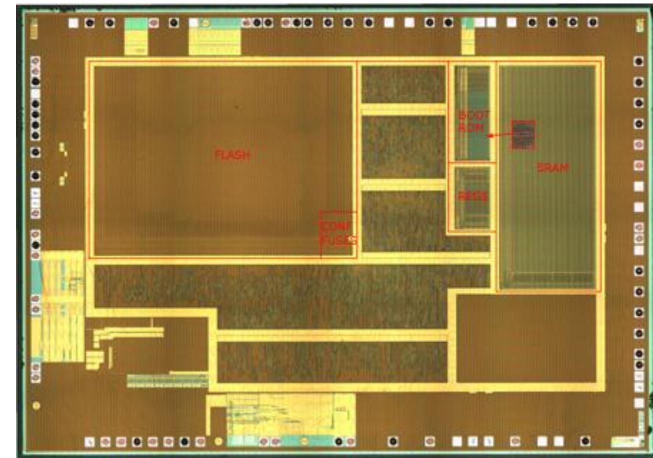
For this class the Von Neumann architecture is thus repartitioned into five basic blocks: Flash Program Memory, SRAM Data Memory, Control Unit, Datapath, and Input-Output.

Figure 1-3 Basic Microcontroller Architectural Elements



Appendix F ATmega328P Architectural Overview

Reading: Section 5.1 Overview plus Atmega8 Block Diagram



Clock

- ATmega Family – Up to 20 MHz
- Arduino Duemilanove – 16 MHz (ATmega328P)
- ALU – On-chip 2-cycle Hardware Multiplier

Memory

- ATmega Family – Up to 256 KBytes Flash, 4K Bytes EEPROM and 8K Bytes SRAM.
- ATmega328P – 32 KBytes Flash, 1K Bytes EEPROM, and 2K Bytes SRAM
- Self-Programming Flash memory with boot block (ICSP header)

Peripheral Subsystems

- Two 8-bit (PORTB, PORTD), plus One 7-bit (PORTC) General Digital I/O Ports
- Programmable Serial USART, Master/Slave SPI Serial Interface.
- Byte-oriented 2-wire Serial Interface (TWI) is Philips I^2C compliant.
- Two 8-bit Timer/Counters with Separate Prescaler and Compare Mode
- One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
- Six PWM Channels
- 8-channel 10-bit A/D converter with up to x200 analog gain stage.
- Programmable Watchdog Timer with Separate On-chip Oscillator
- On-Chip Debug through JTAG or debugWIRE interface.

Other Features

- External and Internal Interrupt Sources with 2 instruction words/vector

Note

- In the following Block Diagram, Power (Vcc), Ground (GND), and the clock input (XTAL) are are present but not shown.

APPENDIX G MICROPROCESSOR VERSUS MICROCONTROLLER

	Typical Microprocessor	Low-Cost Microcontrollers
Cost	Expensive ($100s)	Cheap (<$10)
	Factor of a ten (10)	
Speed	3 GHz (giga - 10^9)	20 MHz (mega – 10^6) *effective* = 20 MIPS
	Factor of a hundred (10^2)	
Cores	**Up to Four**	**One**
Pipeline Stages	**4 to 20**	**0 to 2**
Address Bus	64 bits ($2^{64} = 2^2 2^{60}$)	16 bits ($2^{16} = 2^6 2^{10}$)
	2^{64} = 18,446,744,073,709,551,616 ≈ 10^{19}	2^{16} = 65,536 ≈ 10^4
	Factor of approximately a Quadrillion (10^{15})	
Data Bus	32 bits to 64 bits	8 bits
Instruction Set	Complex	Simple and I/O control oriented
ALU	Floating Point Unit	Simple ALU (+, -, x, plus logic operations)
Program & Data Memory	No	Yes
Peripherals	No	Parallel I/O, Counter/Timers …

Appendix H Two-stage Instruction Pipeline

Pipelining A technique that breaks operations, such as instruction processing or bus transactions, into smaller distinct stages or tenures (respectively) so that a subsequent operation can begin before the previous one has completed.

From the Atmel ATmega328P Data Sheet Chapter 6 *AVR CPU Core*, Section 6.1 *Overview* and with respect to Figure 6-1 Block Diagram of the AVR Architecture

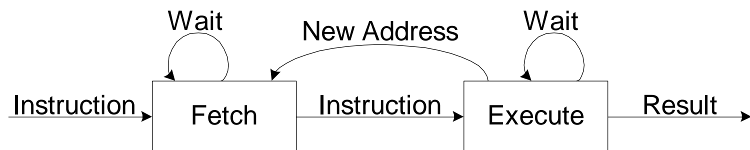
> "In order to maximize performance and parallelism, the AVR uses a Harvard architecture – with separate memories and buses for program and data. Instructions in the program memory are executed with a *single level pipelining*. While one instruction is being executed, the next instruction is pre-fetched from the program memory. This concept enables instructions to be executed in every clock cycle. The program memory is In-System Reprogrammable Flash memory."

A pipeline stage begins and ends with a register; controlled by a clock. Between the register(s) is combinational logic. Although counter-intuitive, Flash program memory can be viewed as combinational logic with an address generating a word of data. With respect to our AVR architecture (Figure 6-1) the two registers of interest are the Program Counter (PC) and the Instruction Register (IR). Without pipelining these two registers in the control unit (PC, IR) would require two clock cycles to complete a basic computer operation cycle. Specifically, an instruction is (1) fetched and then (2) executed.

Figure 10 Fetch and Execute Cycle of the Atmel ATmega Microcontroller

For most instructions, especially one based on a modified Harvard memory model, program memory is not accessed during the execution cycle. This memory down time could be used to fetch the next instruction to be executed, in parallel with the execution cycle of the current instruction. Here then is an opportunity for pipelining! Figure 10.2 illustrates the idea. The pipeline has two independent stages. The first stage fetches an instruction and places it in the Instruction Register (IR), while the second stage is executing the instruction. This two-stage instruction pipeline is also called instruction prefetch can be found in some of the earliest microprocessors including the Intel 8086

Figure 11 Instruction Prefetch of the Intel 8086 Microprocessor



For our RISC architecture *most* instructions are executed in a single cycle (also known as elemental instructions). In this perfect world where all instructions take one cycle to fetch and one cycle to execute, after an initial delay of one cycle to fill the pipeline, known as *latency*, each instruction will take only one cycle to complete.

Figure 12 Program Execution in a AVR RISC two-Stage Instruction Pipelined Architecture

	Time			
	1	2	3	4
Fetch	Instr. 1	Instr. 2	Instr. 3	Instr. 4
Execute		Instr. 1	Instr. 2	Instr. 3

Forgetting for now the circuit delays attendant with implementing the pipeline (for example the latch), and other complicating issues, our performance would be twice that of a non-pipelined design.

APPENDIX I ATMEGA328P INSTRUCTION SET[11]

The ***Instruction Set*** of our AVR processor can be functionally divided (or classified) into the following types:

- Data Transfer Instructions
- Arithmetic and Logic Instructions
- Bit and Bit-Test Instructions
- Branch (Control Transfer) Instructions
- MCU Control Instructions

[11] Source: ATmega328P Data Sheet http://www.atmel.com/dyn/resources/prod_documents/8161S.pdf Chapter 31 Instruction Set Summary

Mnemonics	Operands	Description	Operation	Flags	#Clocks
ARITHMETIC AND LOGIC INSTRUCTIONS					
ADD	Rd, Rr	Add two Registers	Rd ← Rd + Rr	Z,C,N,V,H	1
ADC	Rd, Rr	Add with Carry two Registers	Rd ← Rd + Rr + C	Z,C,N,V,H	1
ADIW	Rdl,K	Add Immediate to Word	Rdh:Rdl ← Rdh:Rdl + K	Z,C,N,V,S	2
SUB	Rd, Rr	Subtract two Registers	Rd ← Rd - Rr	Z,C,N,V,H	1
SUBI	Rd, K	Subtract Constant from Register	Rd ← Rd - K	Z,C,N,V,H	1
SBC	Rd, Rr	Subtract with Carry two Registers	Rd ← Rd - Rr - C	Z,C,N,V,H	1
SBCI	Rd, K	Subtract with Carry Constant from Reg.	Rd ← Rd - K - C	Z,C,N,V,H	1
SBIW	Rdl,K	Subtract Immediate from Word	Rdh:Rdl ← Rdh:Rdl - K	Z,C,N,V,S	2
AND	Rd, Rr	Logical AND Registers	Rd ← Rd • Rr	Z,N,V	1
ANDI	Rd, K	Logical AND Register and Constant	Rd ← Rd • K	Z,N,V	1
OR	Rd, Rr	Logical OR Registers	Rd ← Rd v Rr	Z,N,V	1
ORI	Rd, K	Logical OR Register and Constant	Rd ← Rd v K	Z,N,V	1
EOR	Rd, Rr	Exclusive OR Registers	Rd ← Rd ⊕ Rr	Z,N,V	1
COM	Rd	One's Complement	Rd ← 0xFF – Rd	Z,C,N,V	1
NEG	Rd	Two's Complement	Rd ← 0x00 – Rd	Z,C,N,V,H	1
SBR	Rd,K	Set Bit(s) in Register	Rd ← Rd v K	Z,N,V	1
CBR	Rd,K	Clear Bit(s) in Register	Rd ← Rd • (0xFF - K)	Z,N,V	1
INC	Rd	Increment	Rd ← Rd + 1	Z,N,V	1
DEC	Rd	Decrement	Rd ← Rd – 1	Z,N,V	1
TST	Rd	Test for Zero or Minus	Rd ← Rd • Rd	Z,N,V	1
CLR	Rd	Clear Register	Rd ← Rd ⊕ Rd	Z,N,V	1
SER	Rd	Set Register	Rd ← 0xFF	None	1
MUL	Rd, Rr	Multiply Unsigned	R1:R0 ← Rd x Rr	Z,C	2
MULS	Rd, Rr	Multiply Signed	R1:R0 ← Rd x Rr	Z,C	2
MULSU	Rd, Rr	Multiply Signed with Unsigned	R1:R0 ← Rd x Rr	Z,C	2
FMUL	Rd, Rr	Fractional Multiply Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULS	Rd, Rr	Fractional Multiply Signed	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULSU	Rd, Rr	Fractional Multiply Signed with Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2
BRANCH INSTRUCTIONS					
RJMP	k	Relative Jump	PC ← PC + k + 1	None	2
IJMP		Indirect Jump to (Z)	PC ← Z	None	2
JMP[1]	k	Direct Jump	PC ← k	None	3
RCALL	k	Relative Subroutine Call	PC ← PC + k + 1	None	3
ICALL		Indirect Call to (Z)	PC ← Z	None	3
CALL[1]	k	Direct Subroutine Call	PC ← k	None	4
RET		Subroutine Return	PC ← STACK	None	4
RETI		Interrupt Return	PC ← STACK	I	4
CPSE	Rd,Rr	Compare, Skip if Equal	if (Rd = Rr) PC ← PC + 2 or 3	None	1/2/3
CP	Rd,Rr	Compare	Rd – Rr	Z, N,V,C,H	1
CPC	Rd,Rr	Compare with Carry	Rd – Rr – C	Z, N,V,C,H	1
CPI	Rd,K	Compare Register with Immediate	Rd – K	Z, N,V,C,H	1
SBRC	Rr, b	Skip if Bit in Register Cleared	if (Rr(b)=0) PC ← PC + 2 or 3	None	1/2/3
SBRS	Rr, b	Skip if Bit in Register is Set	if (Rr(b)=1) PC ← PC + 2 or 3	None	1/2/3
SBIC	P, b	Skip if Bit in I/O Register Cleared	if (P(b)=0) PC ← PC + 2 or 3	None	1/2/3
SBIS	P, b	Skip if Bit in I/O Register is Set	if (P(b)=1) PC ← PC + 2 or 3	None	1/2/3
BRBS	s, k	Branch if Status Flag Set	if (SREG(s) = 1) then PC←PC+k + 1	None	1/2
BRBC	s, k	Branch if Status Flag Cleared	if (SREG(s) = 0) then PC←PC+k + 1	None	1/2
BREQ	k	Branch if Equal	if (Z = 1) then PC ← PC + k + 1	None	1/2
BRNE	k	Branch if Not Equal	if (Z = 0) then PC ← PC + k + 1	None	1/2
BRCS	k	Branch if Carry Set	if (C = 1) then PC ← PC + k + 1	None	1/2
BRCC	k	Branch if Carry Cleared	if (C = 0) then PC ← PC + k + 1	None	1/2
BRSH	k	Branch if Same or Higher	if (C = 0) then PC ← PC + k + 1	None	1/2
BRLO	k	Branch if Lower	if (C = 1) then PC ← PC + k + 1	None	1/2
BRMI	k	Branch if Minus	if (N = 1) then PC ← PC + k + 1	None	1/2
BRPL	k	Branch if Plus	if (N = 0) then PC ← PC + k + 1	None	1/2
BRGE	k	Branch if Greater or Equal, Signed	if (N ⊕ V= 0) then PC ← PC + k + 1	None	1/2
BRLT	k	Branch if Less Than Zero, Signed	if (N ⊕ V= 1) then PC ← PC + k + 1	None	1/2
BRHS	k	Branch if Half Carry Flag Set	if (H = 1) then PC ← PC + k + 1	None	1/2
BRHC	k	Branch if Half Carry Flag Cleared	if (H = 0) then PC ← PC + k + 1	None	1/2
BRTS	k	Branch if T Flag Set	if (T = 1) then PC ← PC + k + 1	None	1/2
BRTC	k	Branch if T Flag Cleared	if (T = 0) then PC ← PC + k + 1	None	1/2
BRVS	k	Branch if Overflow Flag is Set	if (V = 1) then PC ← PC + k + 1	None	1/2
BRVC	k	Branch if Overflow Flag is Cleared	if (V = 0) then PC ← PC + k + 1	None	1/2

Mnemonics	Operands	Description	Operation	Flags	#Clocks
BRIE	k	Branch if Interrupt Enabled	if (I = 1) then PC ← PC + k + 1	None	1/2
BRID	k	Branch if Interrupt Disabled	if (I = 0) then PC ← PC + k + 1	None	1/2
BIT AND BIT-TEST INSTRUCTIONS					
SBI	P,b	Set Bit in I/O Register	I/O(P,b) ← 1	None	2
CBI	P,b	Clear Bit in I/O Register	I/O(P,b) ← 0	None	2
LSL	Rd	Logical Shift Left	Rd(n+1) ← Rd(n), Rd(0) ← 0	Z,C,N,V	1
LSR	Rd	Logical Shift Right	Rd(n) ← Rd(n+1), Rd(7) ← 0	Z,C,N,V	1
ROL	Rd	Rotate Left Through Carry	Rd(0)←C,Rd(n+1)← Rd(n),C←Rd(7)	Z,C,N,V	1
ROR	Rd	Rotate Right Through Carry	Rd(7)←C,Rd(n)← Rd(n+1),C←Rd(0)	Z,C,N,V	1
ASR	Rd	Arithmetic Shift Right	Rd(n) ← Rd(n+1), n=0..6	Z,C,N,V	1
SWAP	Rd	Swap Nibbles	Rd(3..0)←Rd(7..4),Rd(7..4)←Rd(3..0)	None	1
BSET	s	Flag Set	SREG(s) ← 1	SREG(s)	1
BCLR	s	Flag Clear	SREG(s) ← 0	SREG(s)	1
BST	Rr, b	Bit Store from Register to T	T ← Rr(b)	T	1
BLD	Rd, b	Bit load from T to Register	Rd(b) ← T	None	1
SEC		Set Carry	C ← 1	C	1
CLC		Clear Carry	C ← 0	C	1
SEN		Set Negative Flag	N ← 1	N	1
CLN		Clear Negative Flag	N ← 0	N	1
SEZ		Set Zero Flag	Z ← 1	Z	1
CLZ		Clear Zero Flag	Z ← 0	Z	1
SEI		Global Interrupt Enable	I ← 1	I	1
CLI		Global Interrupt Disable	I ← 0	I	1
SES		Set Signed Test Flag	S ← 1	S	1
CLS		Clear Signed Test Flag	S ← 0	S	1
SEV		Set Twos Complement Overflow.	V ← 1	V	1
CLV		Clear Twos Complement Overflow	V ← 0	V	1
SET		Set T in SREG	T ← 1	T	1
CLT		Clear T in SREG	T ← 0	T	1
SEH		Set Half Carry Flag in SREG	H ← 1	H	1
CLH		Clear Half Carry Flag in SREG	H ← 0	H	1
DATA TRANSFER INSTRUCTIONS					
MOV	Rd, Rr	Move Between Registers	Rd ← Rr	None	1
MOVW	Rd, Rr	Copy Register Word	Rd+1:Rd ← Rr+1:Rr	None	1
LDI	Rd, K	Load Immediate	Rd ← K	None	1
LD	Rd, X	Load Indirect	Rd ← (X)	None	2
LD	Rd, X+	Load Indirect and Post-Inc.	Rd ← (X), X ← X + 1	None	2
LD	Rd, - X	Load Indirect and Pre-Dec.	X ← X - 1, Rd ← (X)	None	2
LD	Rd, Y	Load Indirect	Rd ← (Y)	None	2
LD	Rd, Y+	Load Indirect and Post-Inc.	Rd ← (Y), Y ← Y + 1	None	2
LD	Rd, - Y	Load Indirect and Pre-Dec.	Y ← Y - 1, Rd ← (Y)	None	2
LDD	Rd,Y+q	Load Indirect with Displacement	Rd ← (Y + q)	None	2
LD	Rd, Z	Load Indirect	Rd ← (Z)	None	2
LD	Rd, Z+	Load Indirect and Post-Inc.	Rd ← (Z), Z ← Z+1	None	2
LD	Rd, -Z	Load Indirect and Pre-Dec.	Z ← Z - 1, Rd ← (Z)	None	2
LDD	Rd, Z+q	Load Indirect with Displacement	Rd ← (Z + q)	None	2
LDS	Rd, k	Load Direct from SRAM	Rd ← (k)	None	2
ST	X, Rr	Store Indirect	(X) ← Rr	None	2
ST	X+, Rr	Store Indirect and Post-Inc.	(X) ← Rr, X ← X + 1	None	2
ST	- X, Rr	Store Indirect and Pre-Dec.	X ← X - 1, (X) ← Rr	None	2
ST	Y, Rr	Store Indirect	(Y) ← Rr	None	2
ST	Y+, Rr	Store Indirect and Post-Inc.	(Y) ← Rr, Y ← Y + 1	None	2
ST	- Y, Rr	Store Indirect and Pre-Dec.	Y ← Y - 1, (Y) ← Rr	None	2
STD	Y+q,Rr	Store Indirect with Displacement	(Y + q) ← Rr	None	2
ST	Z, Rr	Store Indirect	(Z) ← Rr	None	2
ST	Z+, Rr	Store Indirect and Post-Inc.	(Z) ← Rr, Z ← Z + 1	None	2
ST	-Z, Rr	Store Indirect and Pre-Dec.	Z ← Z - 1, (Z) ← Rr	None	2
STD	Z+q,Rr	Store Indirect with Displacement	(Z + q) ← Rr	None	2
STS	k, Rr	Store Direct to SRAM	(k) ← Rr	None	2
LPM		Load Program Memory	R0 ← (Z)	None	3
LPM	Rd, Z	Load Program Memory	Rd ← (Z)	None	3
LPM	Rd, Z+	Load Program Memory and Post-Inc	Rd ← (Z), Z ← Z+1	None	3
SPM		Store Program Memory	(Z) ← R1:R0	None	-
IN	Rd, P	In Port	Rd ← P	None	1
OUT	P, Rr	Out Port	P ← Rr	None	1
PUSH	Rr	Push Register on Stack	STACK ← Rr	None	2

Mnemonics	Operands	Description	Operation	Flags	#Clocks
POP	Rd	Pop Register from Stack	Rd ← STACK	None	2
MCU CONTROL INSTRUCTIONS					
NOP		No Operation		None	1
SLEEP		Sleep	(see specific descr. for Sleep function)	None	1
WDR		Watchdog Reset	(see specific descr. for WDR/timer)	None	1
BREAK		Break	For On-chip Debug Only	None	N/A

Note: 1. These instructions are only available in ATmega168PA and ATmega328P.