

# EECE416 Microcomputer Fundamentals

## Microprocessor Architecture

**Dr. Charles Kim**  
**Howard University**

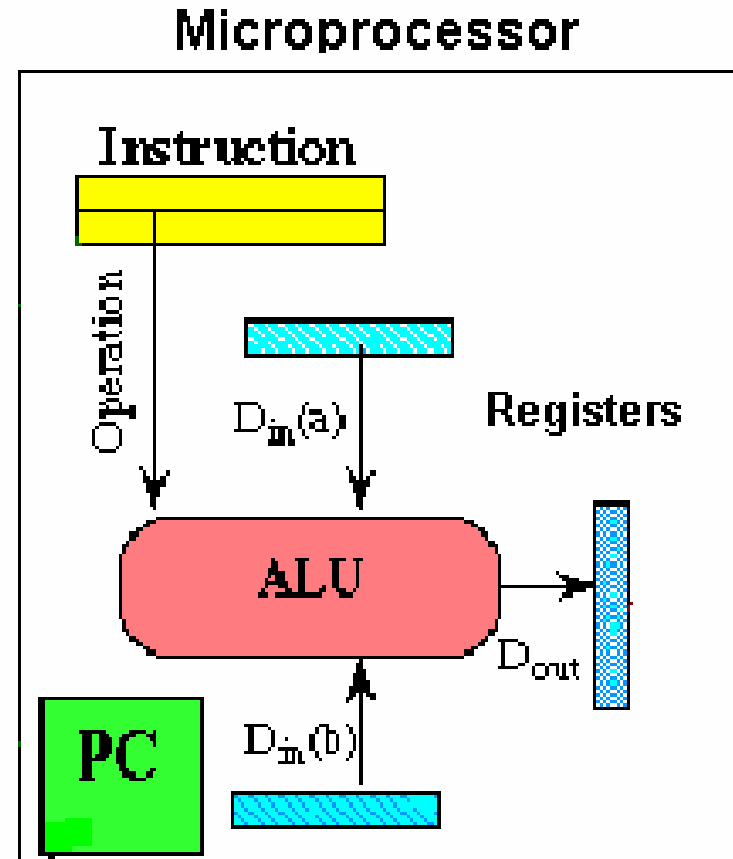
# Computer Architecture

## ⌘ Computer System

- ☒ CPU (with PC, Register, SR) + Memory

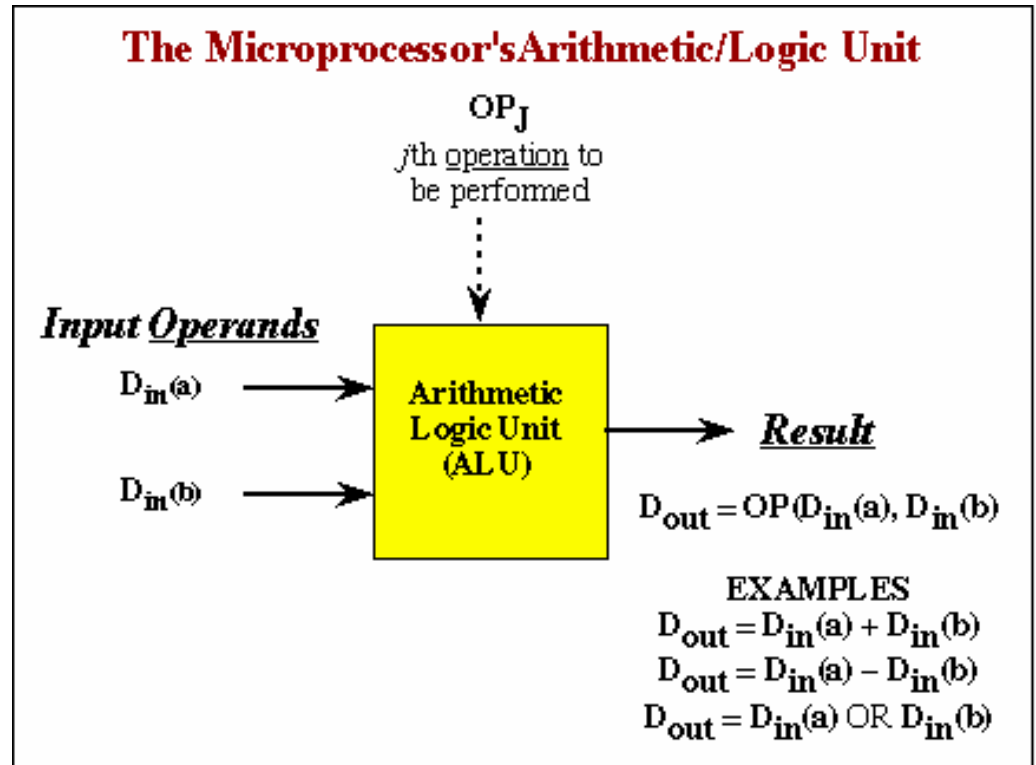
## ⌘ Computer Architecture:

- ☒ "conceptual design and fundamental operational structure of a computer system"
- ☒ "blueprint and functional description of **requirements** and **design implementations** of a computer"
- ☒ focusing on the way the CPU **performs** and **accesses** memory.

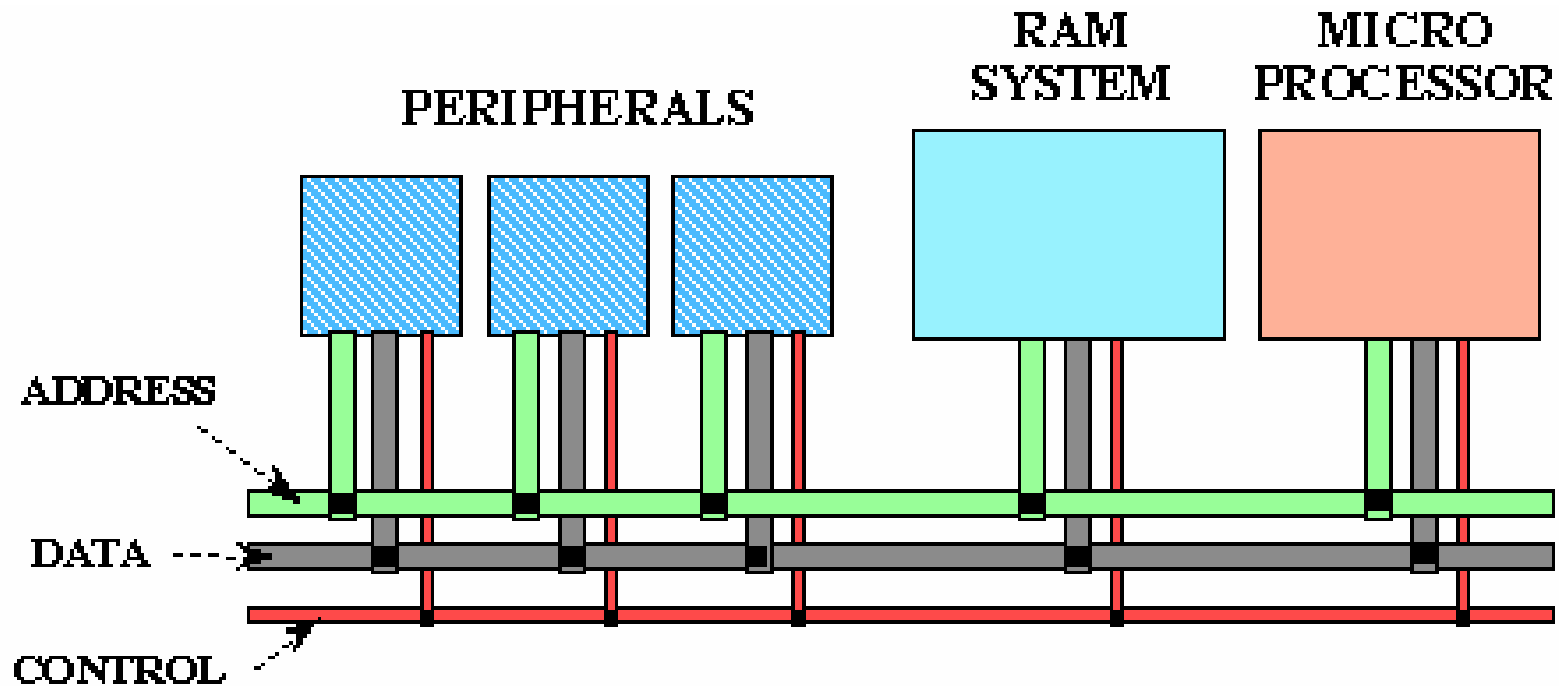


# Computer Architecture

- ALU (Arithmetic Logic Unit)
- Fundamental building block of CPU
- Binary Full Adder



# Microprocessor Bus



# Architecture by CPU+MEM organization

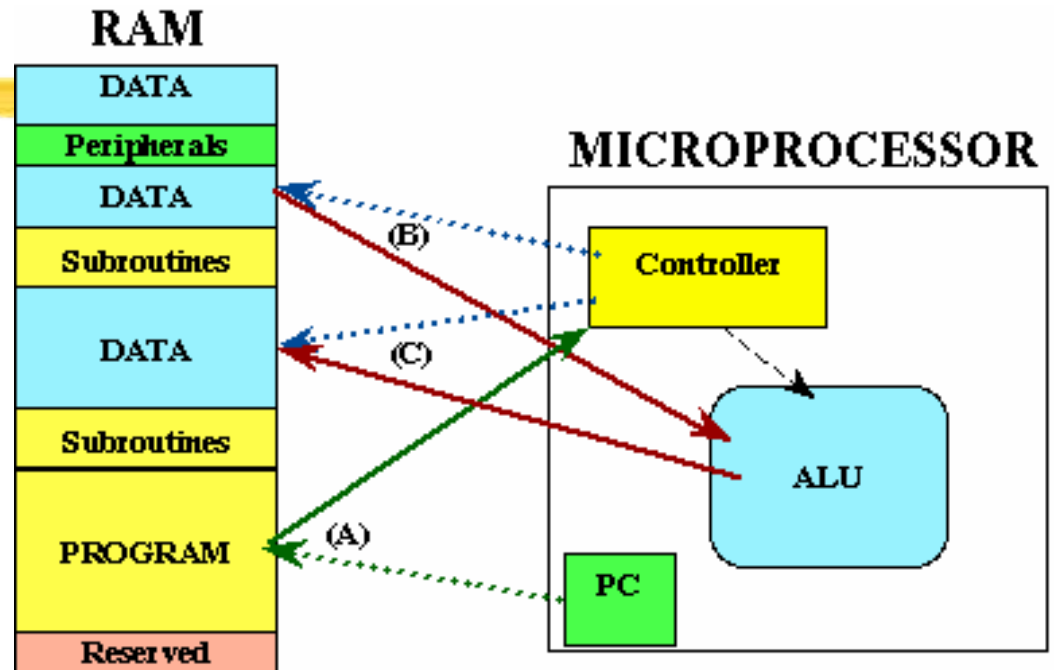
- ⌘ Princeton (or von Neumann) Architecture
  - ☑ MEM contains both Instruction and Data
  - ☑ Von Neumann Bottleneck – CPU  $\leftrightarrow$  Memory
  - ☑ Cache
- ⌘ Harvard Architecture
  - ☑ Data MEM and Instruction MEM
  - ☑ Higher Performance
  - ☑ Better for DSP
  - ☑ Higher MEM Bandwidth

# Princeton Architecture

1. **Step (A):** The address for the instruction to be next executed is applied

(**Step (B):** The controller "decodes" the instruction

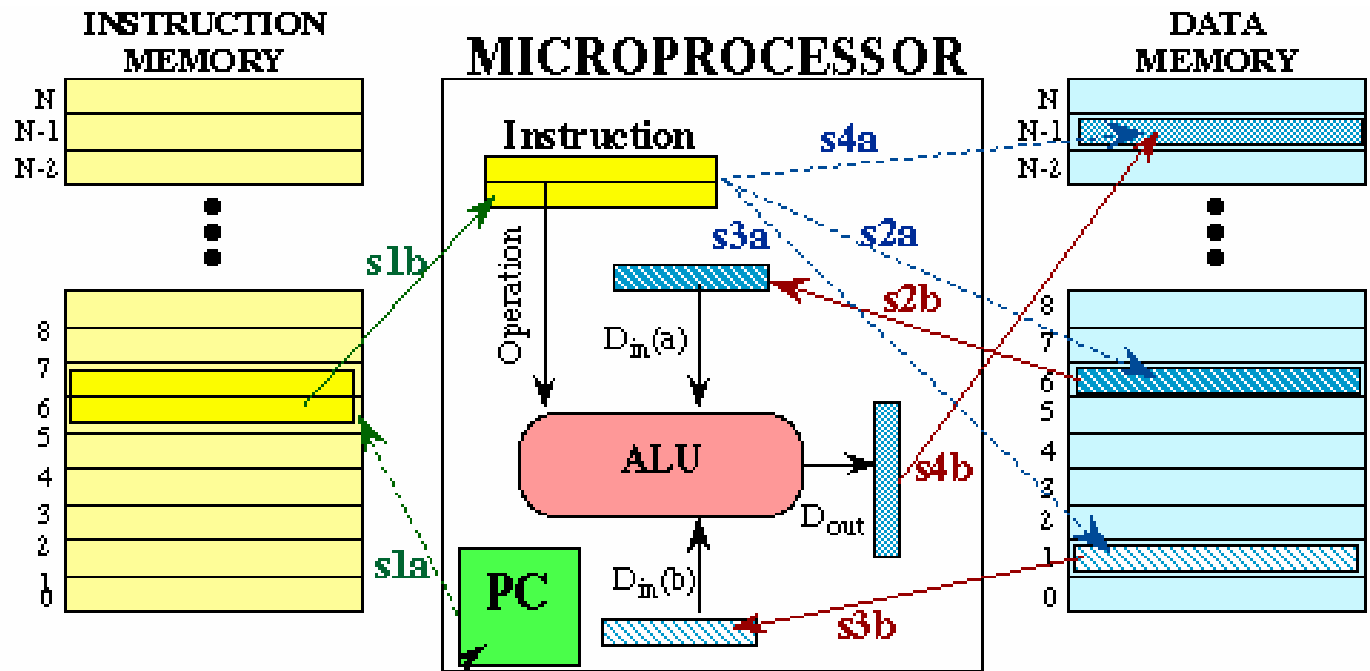
3. **Step (C):** Following completion of the instruction, the controller provides the address, to the memory unit, at which the data result generated by the operation will be stored.



- CPU can be either reading an instruction or reading/writing data from/to the memory.
- Both cannot occur at the same time since the instructions and data use the same bus system

# Harvard Architecture

- ⌘ CPU can both read an instruction and perform a data memory access at the same time.
- ⌘ Faster for a given circuit complexity because instruction fetches and data access do not contend for a single memory pathway.

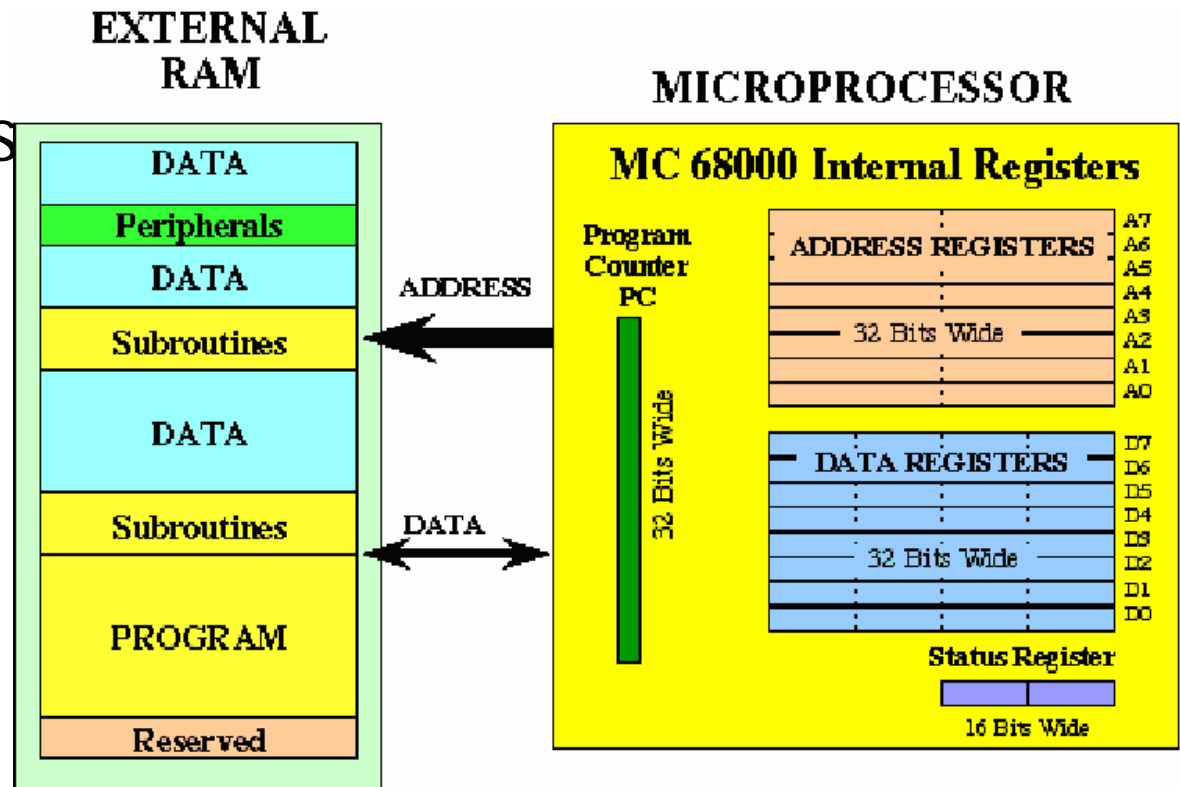


"PC" is the microprocessor's Program Counter

Address	Action
s1a	s1b: get instruction
s2a	s2b: get first data input
s3a	s3b: get second data input
s4a	s4b: store data output

# Internal Memory ("register")

- External memory access is Very slow
- For quicker retrieval and storage
- Internal registers



# Architecture by Instructions and Executions

## ⌘ CISC (Complex Instruction Set Computer)

- ☒ Variety of instructions for complex tasks directly to hardware
- ☒ Easy to translate high-level language to assembly
- ☒ Complex Hardware
- ☒ Instructions of varying length

## ⌘ RISC (Reduced Instruction Set Computer)

- ☒ Fewer and simpler instructions
- ☒ Each instruction takes the same amount of time
- ☒ Less complex hardware
- ☒ High performance microprocessors
- ☒ Pipelined instruction execution (several instructions are executed in parallel)

# CISC

- ⌘ Architecture of prior to mid-1980's
  - ☒ IBM390, Motorola 680x0, Intel80x86
- ⌘ Basic Fetch-Execute sequence to support a large number of complex instructions
- ⌘ Complex decoding procedures
- ⌘ Complex control unit
- ⌘ One instruction achieves a complex task

# Characteristics of RISC

## ⌘ Favorable changes for RISC

- ⊞ **Caches** to speed instruction fetches
- ⊞ Dramatic memory size increases/cost decreases
- ⊞ Better *pipelining*
- ⊞ Advanced optimizing compilers

## ⌘ Characteristics of RISC

- ⊞ Instructions are of a uniform length
- ⊞ Increased number of registers to hold frequently used variables (16 - 64 Registers)
- ⊞ Central to High Performance Computing

# Pipelining

⌘ A method of increasing the number of concurrent operations

⌘ Two types

☐ Arithmetic pipeline

☒ Ex: floating point multiplication

☐ Instruction pipeline

☒ Partitions the fetch-execution into several stages

# Pipeline-based Architectural techniques

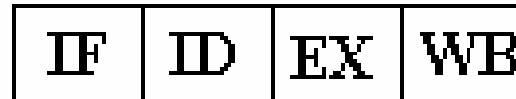
## ⌘ Basic Model: 4 stages

- ⊞ IF (Instruction Fetch)
- ⊞ ID (Instruction Decode)
- ⊞ EX (Execute)
- ⊞ WB (Write Back)

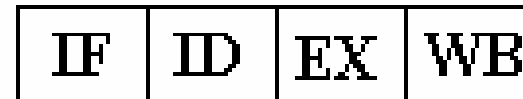
INSTRUCTION

TIME

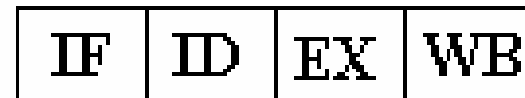
ADD R5, R6, R7



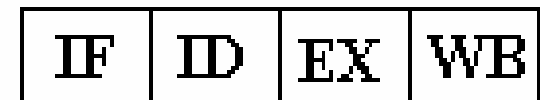
SUB R8, R9, R10



CMP R11, R12



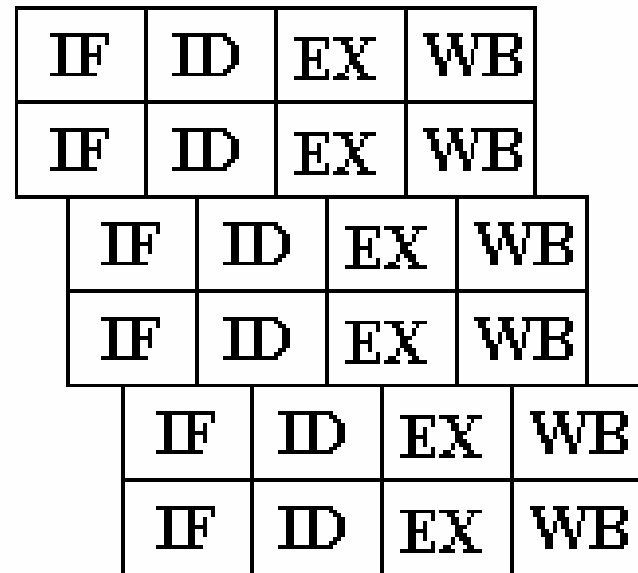
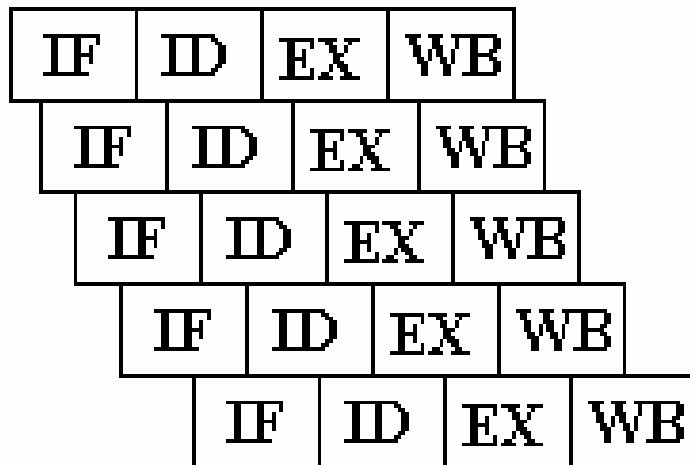
BNE R3



# Pipelining Techniques

## ⌘ Superpipeline architecture :

- ⊞ Fine grained partitioning of the stages



# Pipelining Techniques

## ⌘ Superscalar pipeline architecture:

- ⏏ More than one instruction in one clock cycle

IF	ID	EX	WB
IF	ID	EX	WB

IF	ID	EX	WB		
IF	ID	EX	WB		
	IF	ID	EX	WB	
	IF	ID	EX	WB	
		IF	ID	EX	WB
		IF	ID	EX	WB

# Pipelining Techniques

## ⌘ VLIW(Very Long Instruction Word) architecture:

- ☒ Several instructions are packed into one long instruction word

Memory Ref. Instr. 1	Memory Ref. Instr. 2	Float Pt. Istr 1	Float Pt. Inst 2	Integer Inst
159 ... 128	127 ... 96	95 ... 64	63 ... 32	31 ... 0

## ⌘ Vector pipelines:

- ☒ Vector operations are implemented with single instruction (special H/W)

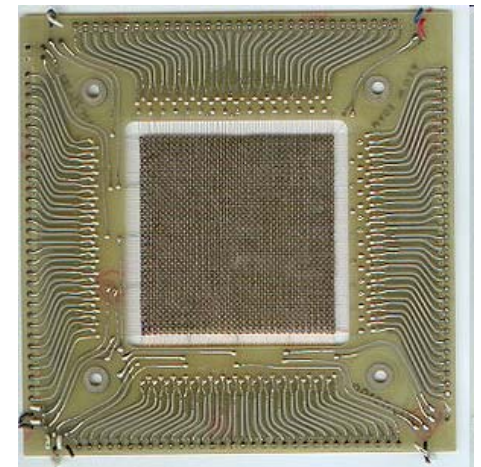
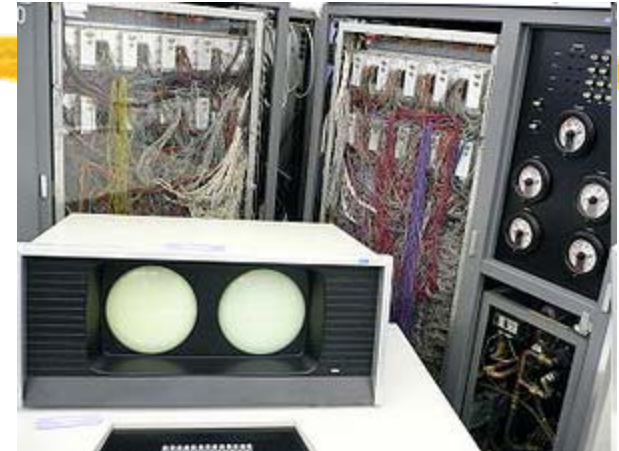
<b>Differences</b>	<b>CISC</b>	<b>RISC</b>
<b>Instruction Type</b>	Many	Few
<b>Addressing Mode</b>	Many	Few
<b>Instruction Format</b>	Variable	Fixed
<b>Pipelining</b>	Difficult	Efficient
<b>Control Method</b>	Micro-programmed control	Hardwired control
<b>Instruction Access</b>	Most access memory	Mostly load-store architecture
<b>Optimization Dependency for high performance</b>	Less on optimizing compiler	Highly on optimizing compiler

# History of RISC

## ⌘ *RISC Roots: CDC 6600 (1965)*

• •

- ☒ Control Data Corporation CDC 6600 'Supercomputer'
- ☒ Designed by Seymour Cray
- ☒ Emphasized a small (74 op codes) load/store and register-register instruction as a means to greater performance.
- ☒ The CDC 6600 itself has roots in the UNIVAC 1100, which many CDC 6600 engineers worked on.



# Load/Store Architecture ?

## Load/Store architecture

1. Memory accesses slow a processor down.
2. Even with cache, that only brings the average access time down for memory accesses.
3. There are still times when the processor is doing nothing, while waiting for memory accesses to complete.
4. In the load/store architecture,
  - 4.1) the addressing mode for every operand is fixed.
  - 4.2) for arithmetic/logical type instructions, the addressing mode for all operands will be register mode.
  - 4.3) make sure that there are enough registers, because everything ends up in registers.
  - 4.4) to get stuff to/from memory and into/out of registers, we have explicit instructions that move data.
5. Load instructions read data from memory and copy it to a register.
6. Store instructions write data from a register to memory.

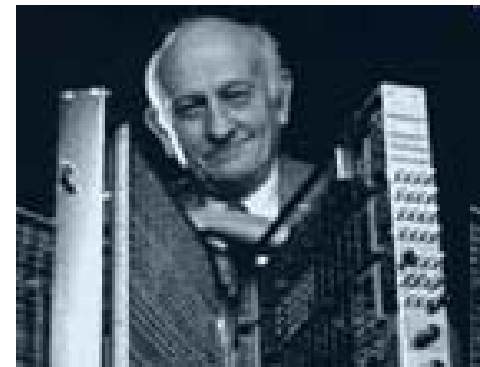
## Load-store architecture

Only **load** and **store** instructions access the memory, all other instructions use registers as operands. What is the motivation? Primary motivation is speedup - registers are faster.

# History of RISC

## ⌘ ***RISC Formalized: IBM 801 (1975)***...

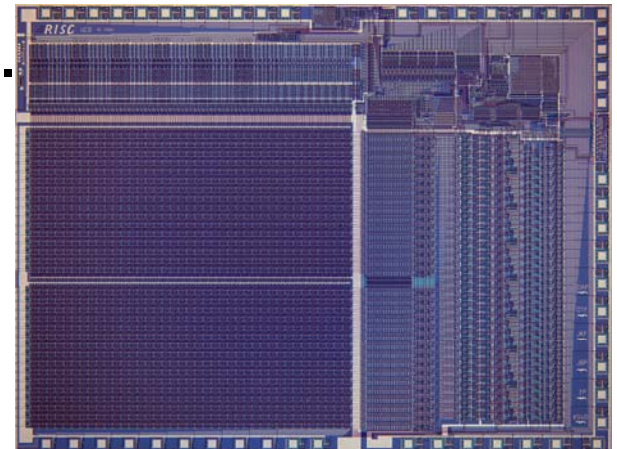
- ⊞ The first system to formalize these principles was the IBM 801 project (1975)
- ⊞ The design goal was to speed up frequently used instructions while discarding complex instructions that slowed the overall implementation.
- ⊞ Like the CDC 6600, memory access was limited to load/store operations
- ⊞ Execution was pipelined, allowing 1 instruction per cycle.



# History of RISC

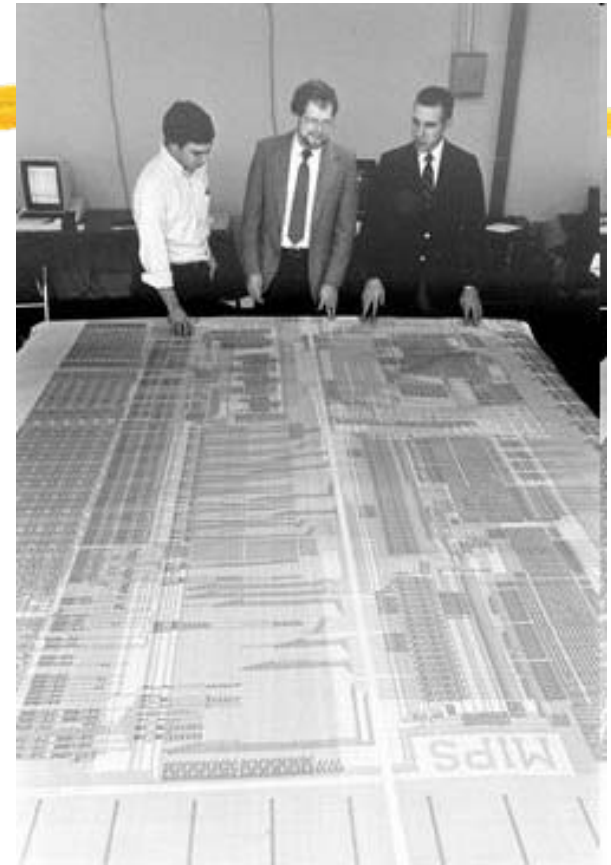
## ⌘ *RISC Refined: Berkeley RISC*

- ☑ Some time after the 801, around 1981, projects at Berkeley (RISC I and II) further developed these concepts.
- ☑ The term RISC came from Berkeley's project, which was the basis for the SPARC processor.
- ☑ Execute in a 3 stage pipeline.



## *RISC Refined: Stanford MIPS . .*

- ☒ Some time after the 801, around 1981, Stanford University further developed these concepts.
- ☒ The Stanford MIPS project was the basis for the **MIPS**.
- ☒ MIPS stands for **Microprocessor without Interlocked Pipeline Stages**, using the compiler to eliminate register conflicts.



# SPARC & ARM



## ⌘ SPARC

- ☑ Scalable (originally Sun) Processor ARChitecture
  - ☑ Scalability - Flexible integration of cache, memory and FPUs
- ☑ Open Architecture
- ☑ Designed by Sun Microsystems for their own.
- ☑ Standard 68000-based CPUs and a standard operating system, Unix.

## ⌘ ARM

- ☑ Advanced RISC Machine, originally Acorn RISC Machine
- ☑ One of the most elegant modern processors in existence.
- ☑ Simple, with a short 3-stage pipeline, and it can operate in big- or little-endian mode.





# INTEL VS. ARM

## ⌘ Next PCs

- ☑ Smart phones (on ARM)
- ☑ Mobile Devices – MP3, Digicam (on ARM)
- ☑ Run on Intel's x86? --- Intel's wish


## ⌘ ARM revisited

- ☑ No chip hardware – license only (powerful and variety of licensees) → cell phones etc
- ☑ SoC device (CPU + I/O + Peripherals+ Memory + etc)

## ⌘ INTEL

- ☑ Does not want to License x86 (Lesson from AMD)
- ☑ New approach for SoC: Atom based X86 SoC

Intel Atom



<b>Produced</b>	2008–present
<b>Common manufacturer(s)</b>	Intel
<b>Max. CPU clock</b>	800 MHz to 2 GHz
<b>FSB speeds</b>	400 MHz to 667 MHz
<b>Min. feature size</b>	45nm
<b>Instruction set</b>	x86, x86-64 (not for the N and Z series)
<b>Cores</b>	1, 2
<b>Package(s)</b>	441-ball µFCBGA
<b>Core name(s)</b>	Silverthorne Diamondville

**Intel Atom** is the [brand](#) name for a line of [x86](#) and [x86-64 CPUs](#) (or [microprocessors](#)) from [Intel](#), designed in [45 nm CMOS](#) and used mainly in [Netbooks](#). The Atom Z series is code-named Silverthorne and the Atom N series is code-named Diamondville. As of June 2009, the most used chips in the Netbook retail market are Z520, Z530, and N270.