Von Neumann Machines

Most modern computers are based on this design
Developed by John von Neumann at Princeton
At the Institute for Advanced Studies in the 1940's.

These machines have 3 major components:

- a CPU
- a main-memory system
- an I/O system

von Neumann Characteristics

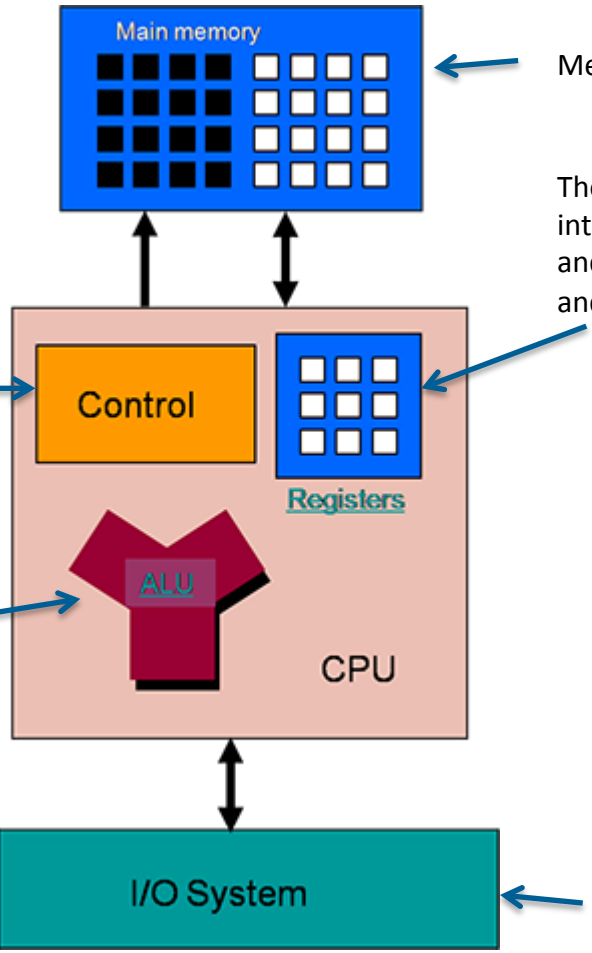
- Both programs and data are stored in a single memory
  program instructions can be manipulated like data

- The program counter is used to fetch instructions
  data operands are fetched during the execute cycle
  based on the operand addressing mode

- Instructions execute sequentially
- flow of control may be altered by a branch type instruction

- CPU accesses memory over a single path
  Which is a potential bottle neck
  Called the "von Neumann bottleneck”

The diagram below depicts a system that adheres to the von Neumann architecture:



Memory holds both instructions and data.

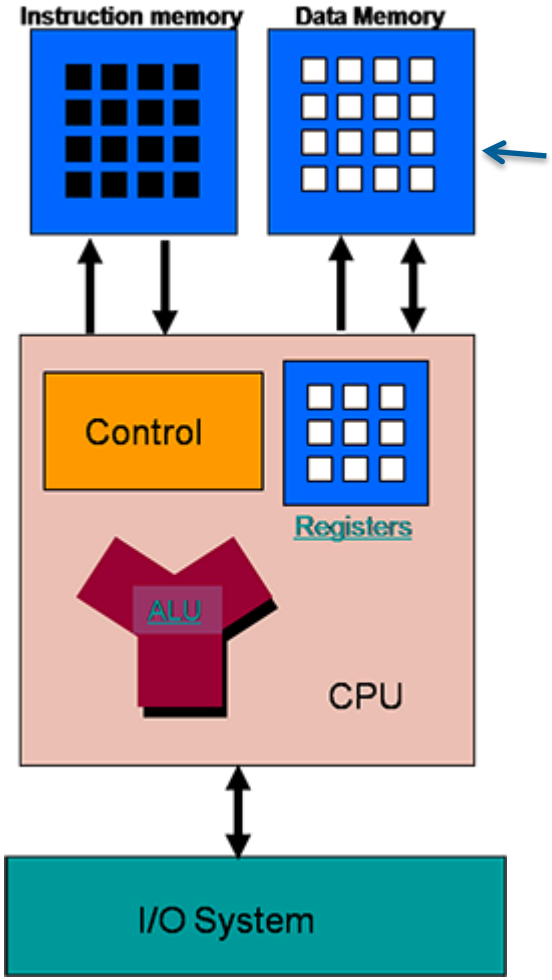
The register set (also called register file) serves as internal storage within the CPU to hold operands and intermediate values during instruction fetch and execution.

The major subsystems within the CPU are: The control unit (CU) which selects and interprets machine instructions and coordinates the various parts of the computer in executing these instructions.

The arithmetic logic unit (ALU) which performs arithmetic, logical, and shift operations on the operands and generates the results.

Allows data to be brought into the system and results to be sent out of the system.

As an alternative, illustrated below is a “Harvard architecture”:

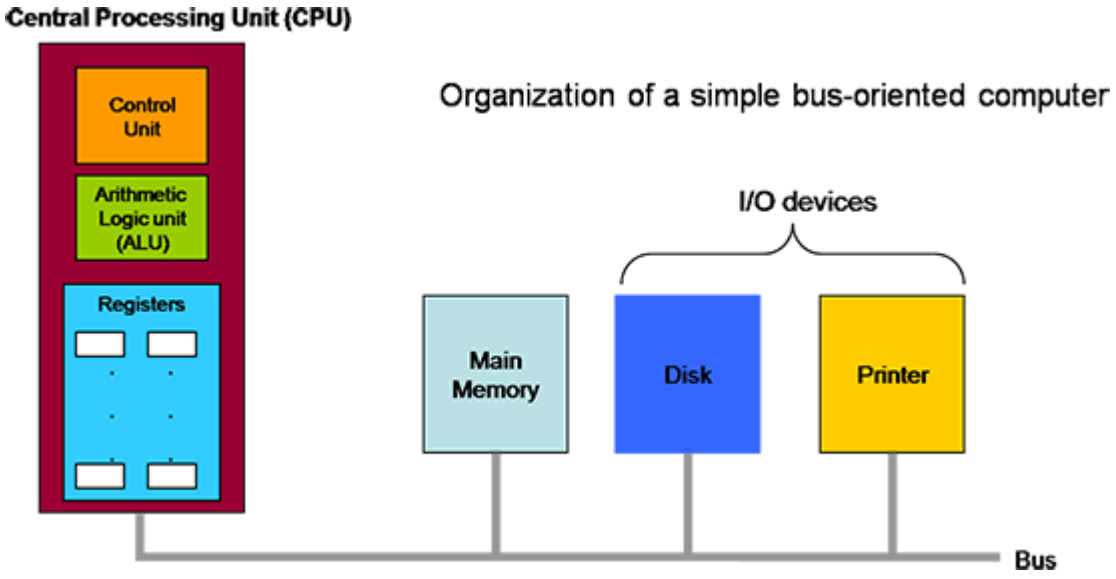


The distinguishing feature is the presence of separate instruction and data memories.

This allows one instruction to be fetched while another stores or reads an operand.

This design was developed at Harvard University by Howard Aiken and others.

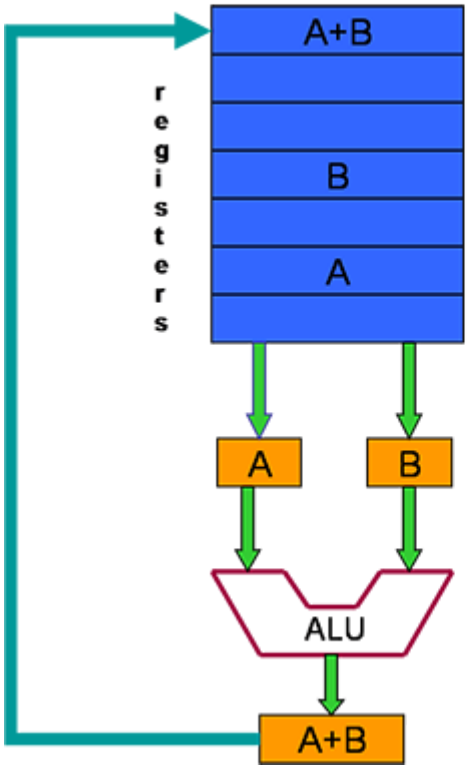
Computer components exchange data and communicate over a “system bus”:



A bus is a collection of parallel wires that carry address, data, and control signals
External buses connect the CPU to memory and I/O devices
Internal buses carry signals between registers, the ALU and the control unit

The Datapath

Different organizations have different datapaths

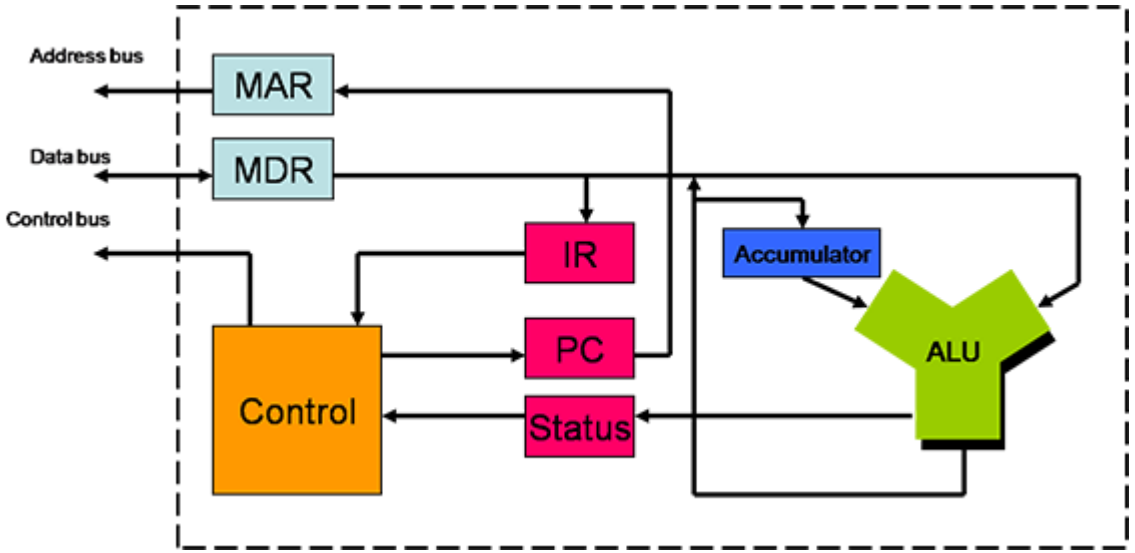


The registers, ALU and interconnecting buses constitute the data path for the machine.

The datapath contains all of the devices and pathways needed to execute instructions.

Typical Data path for executing instructions

Early designs used a special accumulator register
to hold an input operand for the instruction
and to received the instruction's result



Such systems employed one-address instructions
a single memory operand
the second implicit operand was the accumulator register

Example One-address Instructions

LDA	**X**	**# load the variable X from memory into the accumulator**
ADD	Y	# add the memory variable Y to the accumulator
STA	Z	# store the accumulator result into memory variable Z
BGE	P1	# branch to location P1 if result is not negative

Most instructions incur a penalty
- the time required to access the memory operand

The alternative is a load/store architecture
- allows only a few instructions to use memory operands
- all others have to use operands that reside within registers
- registers takes a fraction of the memory access time (better performance)

2 and 3 address instructions

Accumulators limit the number of high cost CPU registers

Other designs use two or more instruction memory operands
This gives rise to two-address and three-address instructions:

```
Mul   x,y   # the variable x is multiplied by y and the product is stored back in x
Add   z,w   # the sum of z plus w is stored in z
```

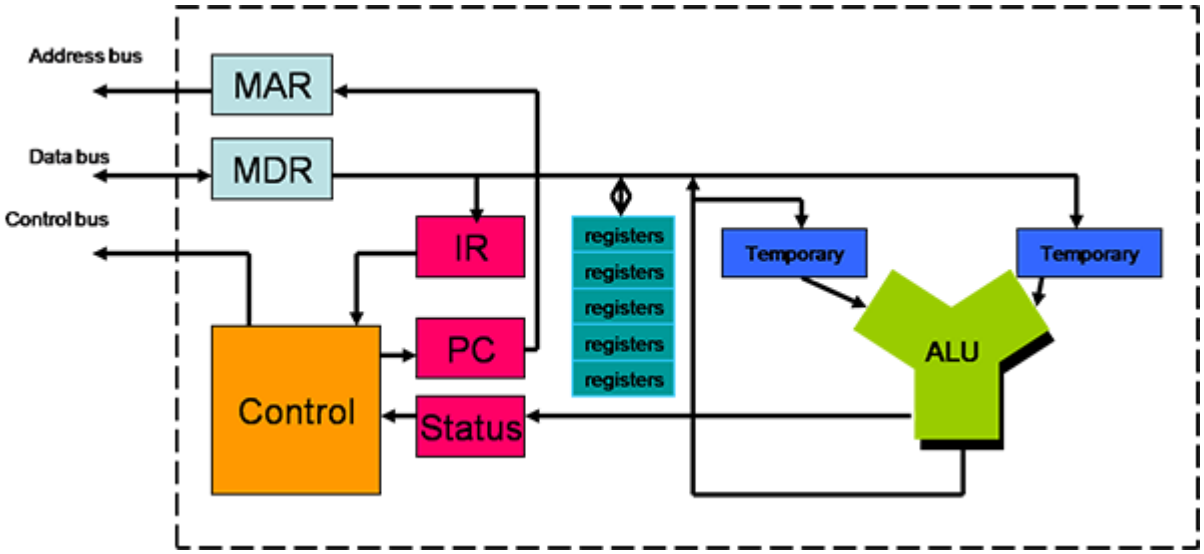
Three-address instructions:

```
Mul   z,x,y   # the product of x times y is stored into z
Add   w,x,y   # the sum of x plus y is stored into w
```

such instructions use even more memory accesses
therefore take longer to execute.

Multi-register Systems

Over time, register cost declined and reliability increased



More on-chip registers allowed fast register-to-register operations

indexed and register indirect addressing could be used

Multi-register Systems

MAR and MDR serve as the CPU to memory interface

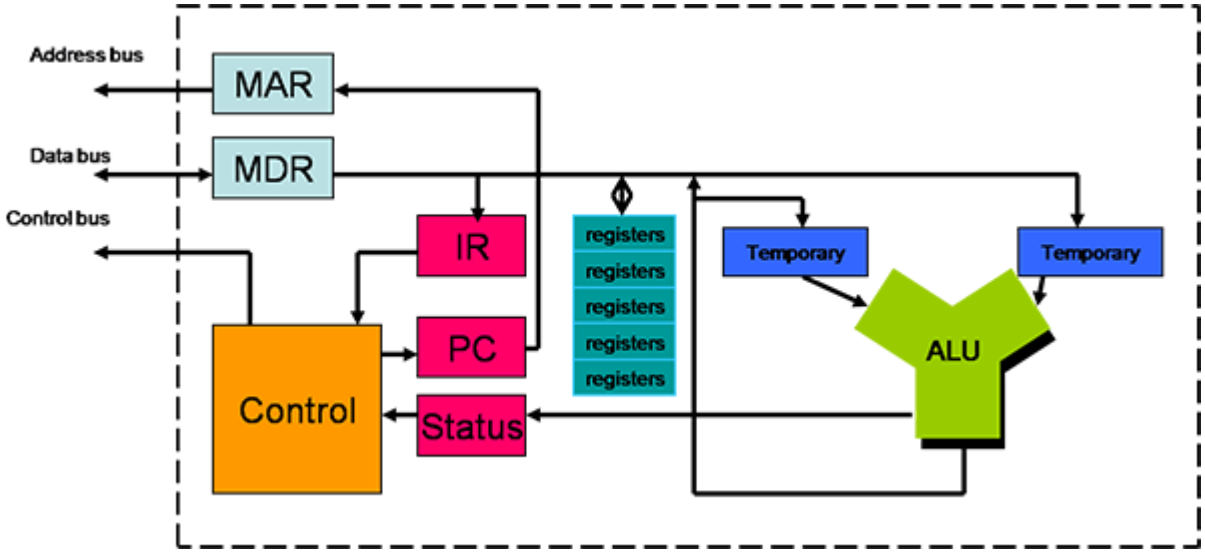
MAR (memory address register)

holds the address of the item in memory to be accessed.

MDR (memory data register)

receives the item read from memory

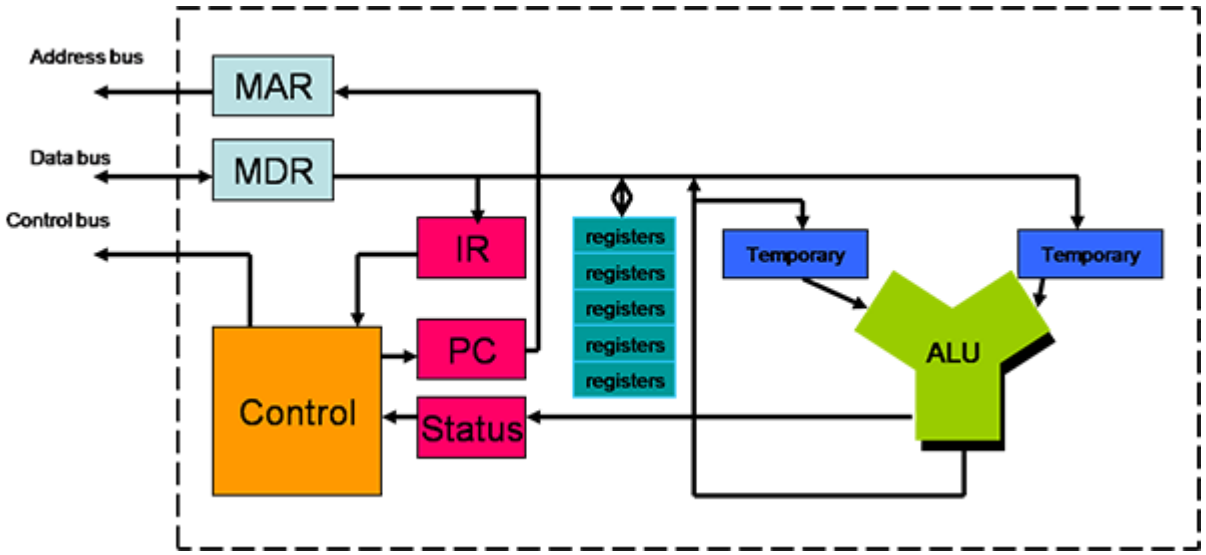
or holds the item to be written into memory



Multi-register Systems

IR is the instruction register
Holds the instruction while the control unit processes it

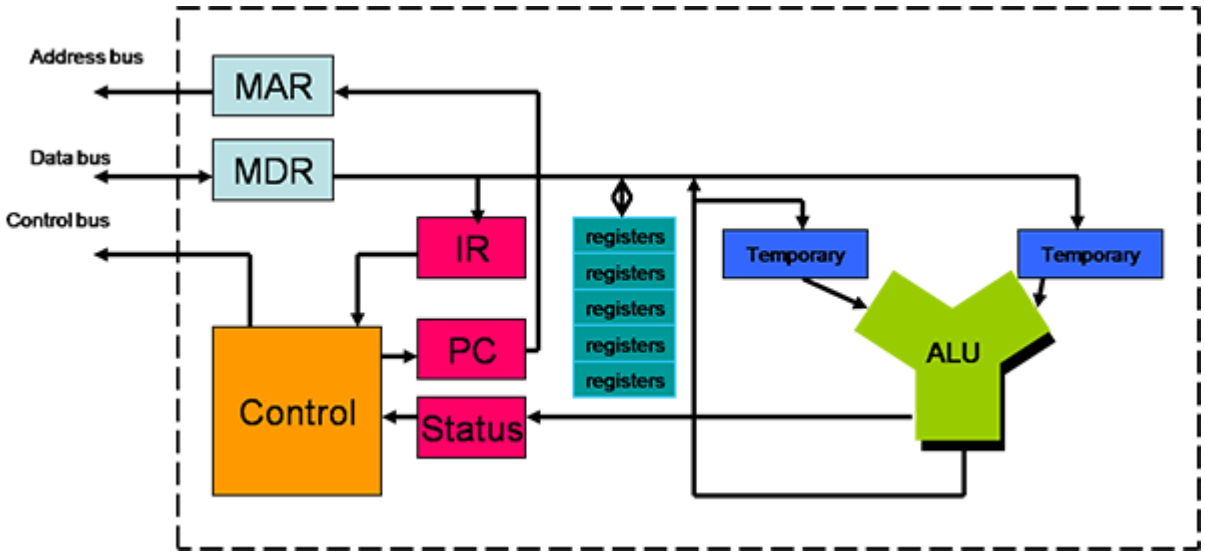
PC is the program counter register
holds the address of the next instruction to fetch from memory



Multi-register Systems

STATUS register holds condition codes
(negative, zero, positive, a carry, etc.)

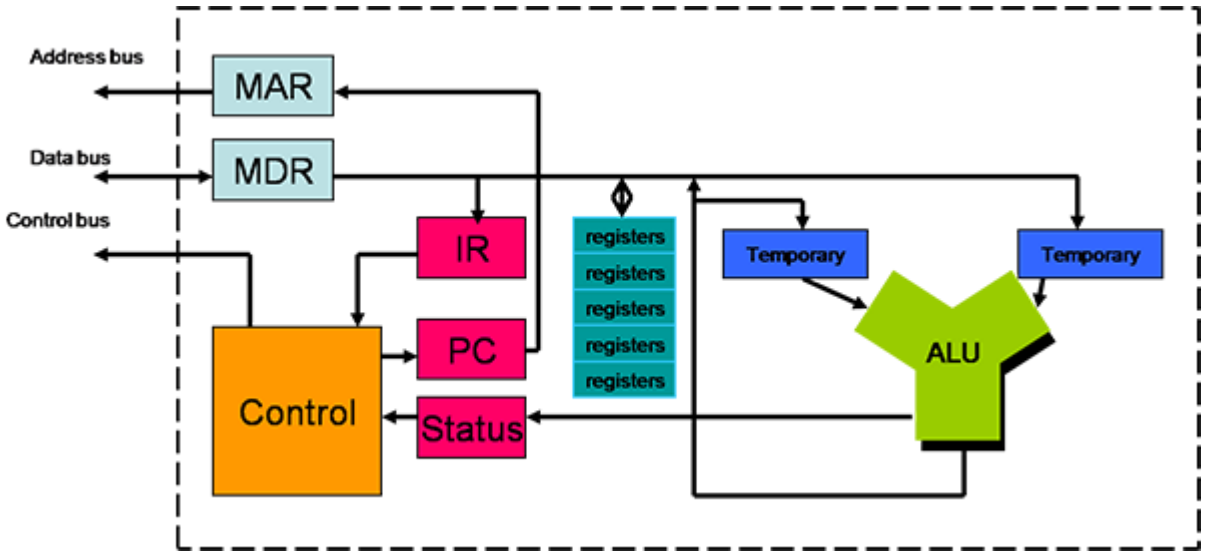
Registers serve as internal storage cells
hold operands and results for the instructions



Multi-register Systems

Control unit generates signals that direct operations
For example, which registers to use, what ALU operation to perform, etc.

The ALU actually performs the operations
(such as addition, multiplication, shifting, etc.)



Multi-register Systems

Multi-register designs allow explicit use of one or more register operands

Some systems have instructions that use 1 memory operand and 1 or more registers: (for example the Intel x86 systems and their clones)

```
add    eax,m        # add the 32-bit variable m to the 32-bit eax register
mov    ebx,data1    # copy the contents of the variable data1 into the ebx register
```

These are sometimes called one and a half address instructions.

Stack Architecture

Some classes of computers use mostly implicit stack operands
These are known as stack-based machines
they use zero-address instructions

the instructions make no explicit reference to operands

Examples of such instructions are:

```
add       # pop the top two stack elements, add them and push the sum onto the stack
dup       # make a copy of the top element (i.e., duplicate it)
sub       # pop the top two elements and push the difference back onto the stack
pop   x   # remove the top element and store it into memory variable x
push  5   # push the constant 5 onto the stack
```

these instructions incur a penalty for each operand access
unless the stack is implemented using registers

Load/Store Architecture

MIPS machines are reduced instruction set computers (RISC)
- require that most instructions use only registers operands
- only the load and store instructions use memory operands
- That is, they employ a load/store architecture

An example of a MIPS instruction that uses 3 register operands:

```
sub    $8,$9,$10    # subtract register $10 from $9 and put the difference in $8
```