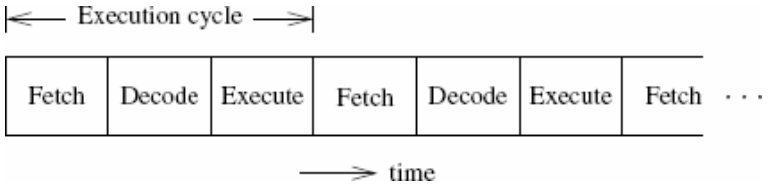
The Instruction Cycle

---

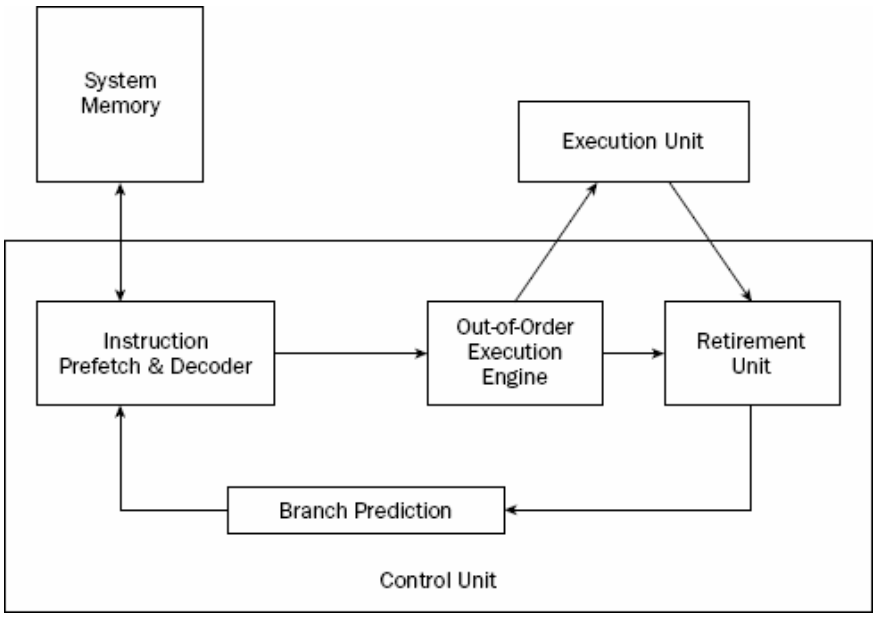
1. The Instruction Cycle

- Instructions are processed under direction of the control unit in step-by-step manner.
- Each step is referred to as a *phase*.
- There are six fundamental phases of the instruction cycle:
    1. fetch instruction (aka pre-fetch)
    2. decode instruction
    3. evaluate address (address generation)
    4. fetch operands (read memory data)
    5. execute (ALU access)
    6. store result (writeback memory data)

Instruction cycle:



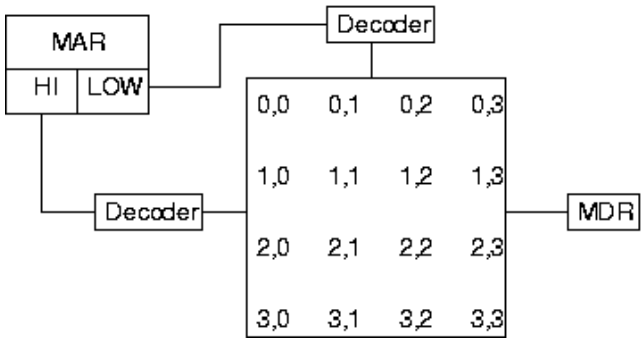
Pentium 4 instruction cycle:



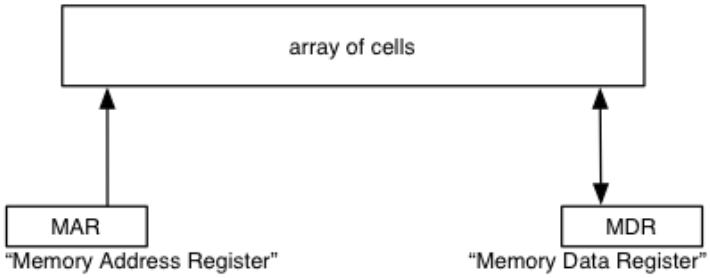
2. Fetch Instruction Phase

- Obtain next instruction from memory.
- Load instruction into instruction register IR.
- **MAR** is loaded with instruction pointer.
- The instruction is loaded through the **MDR**.
- Increment processor counter PC, that is, update instruction pointer address while reading instruction from memory.

Memory Circuitry:



Memory Operations:



fetch (addr):
1. Put *addr* into MAR
2. Tell memory unit to "load"
3. Memory copies data into MDR

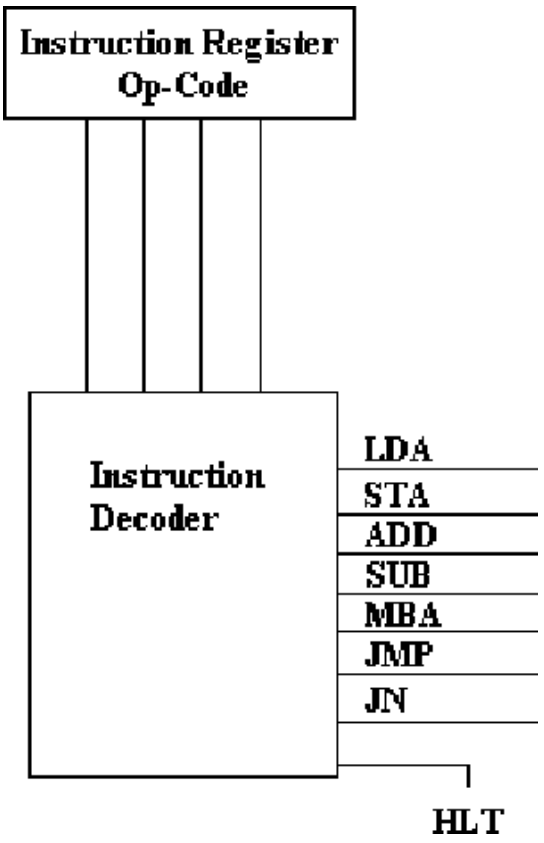
store (addr, new-value):
1. Put *addr* into MAR
2. Put *new-value* into MDR
3. Tell memory unit to "store"
4. Memory stores data from MDR into memory cell.

3. Decode Instruction Phase

- Decoder circuit examines opcode of the instruction.
- Result is selecting unique decoder output line.

Instruction decoder:

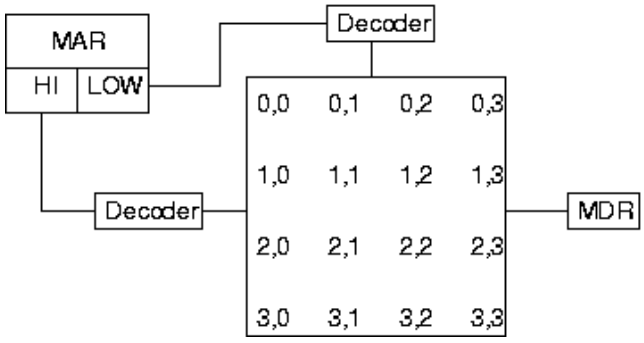
- Output line signals a circuit which implements the corresponding operation.



4. Evaluate Operand Address Phase

Compute address of the memory location of the instruction operand.

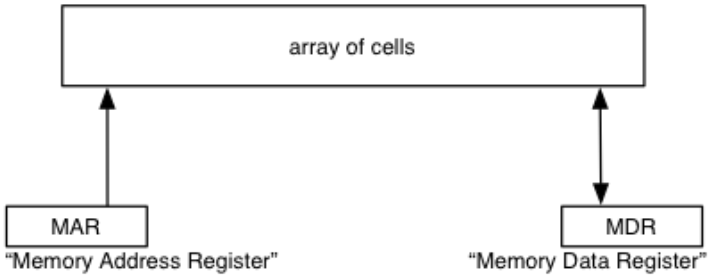
Memory Circuitry:



5. Fetch Operands Phase

- Load **MAR** with address calculated.
- Read memory into **MDR**, making data available as input to the processing unit.

Memory Operations:



fetch (addr):
1. Put *addr* into MAR
2. Tell memory unit to "load"
3. Memory copies data into MDR

store (addr, new-value):
1. Put *addr* into MAR
2. Put *new-value* into MDR
3. Tell memory unit to "store"
4. Memory stores data from MDR into memory cell.

6. Steps in a Typical Read Cycle

---

1. Place the address of the location to be read on the address bus via **MAR**.
2. Activate the memory read control signal on the control bus.
3. Wait for the memory to retrieve the data from the addressed memory location.
4. Read the data from the data bus into **MDR**.
5. Drop the memory read control signal to terminate the read cycle.

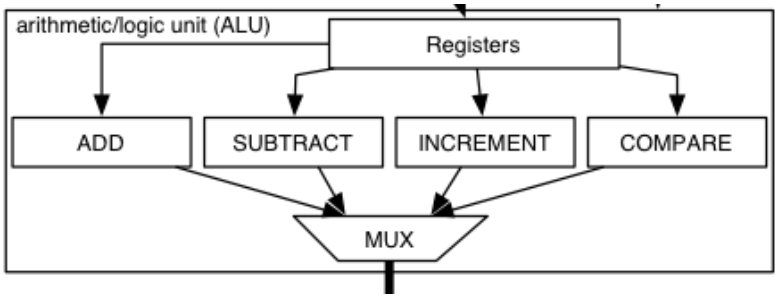
A simple Pentium memory read cycle takes 3 clocks:

- Steps 1-2 and then 4-5 are done in one clock cycle each.
- For slower memories, *wait cycles* will have to be inserted.

7. Execute phase

Microcode for the instruction, selected by the decoder output line, is executed by the ALU.

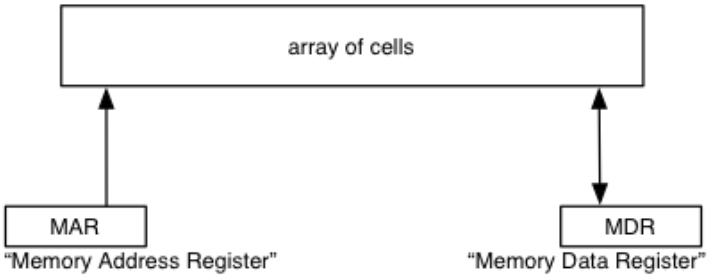
The ALU:



8. Store Result Phase

- Result is written to the designated destination of the instruction operand
- Instruction Cycle begins anew.

Memory Operations:



fetch (addr):
1. Put *addr* into MAR
2. Tell memory unit to "load"
3. Memory copies data into MDR

store (addr, new-value):
1. Put *addr* into MAR
2. Put *new-value* into MDR
3. Tell memory unit to "store"
4. Memory stores data from MDR into memory cell.

9. Steps in a Typical Write Cycle

1. Place the address of the location to be written on the address bus via **MAR**.
2. Place the data to be written on the data bus via **MDR**.
3. Activate the memory write control signal on the control bus.
4. Wait for the memory to store the data at the addressed location.
5. Drop the memory write control signal to terminate the write cycle.

A simple Pentium memory write cycle takes 3 clocks:

- Steps 1-2 and 4-5 are done in one clock cycle each.
- For slower memories, *wait cycles* will have to be inserted.

10. Properties of Memory

- Random access:
  - Accessing any memory location takes the same amount of time.
- Volatility:
  - Volatile memory.
  - Needs power to retain the contents.
- Non-volatile memory:
  - Retains contents even in the absence of power.
- Basic types of memory:
  - Read-only memory, **ROM**.
  - Read/write memory, **RAM**.

11. ROM, Read-only Memory

- ROM characterestics are:
  - Cannot be written into this type of memory.
  - Non-volatile memory
  - Most are factory programmed (i.e., written.)
- Programmable ROMs (PROMs)
  - Can be written once by user
  - A fuse is associated with each bit cell
  - Special equipment is needed to write (to blow the fuse)
- PROMS are useful:
  - During prototype development.
  - If the required quantity is small.

12. EPROM, Erasable PROMs

- EROM characterestics are:
  - Can be written several times.

- Offers further flexibility during system prototyping.
- Can be erased by exposing to ultraviolet light.
- Cannot erase contents of selected locations.
- All content is lost on re-write.

- Electrically erasable PROMs, or **EEPROM**s.
  - Contents are electrically erased.
  - No need to erase all contents
  - Typically a subset of the locations are erased as a group.
  - Most EEPROMs do not provide the capability to individually erase contents of a single location.

13. RAM, Read/Write Memory

- Commonly referred to as random access memory, **RAM**:
  - Volatile memories.
- Two basic RAM types:
  1. Static RAM, **SRAM**:
     - Retains data with no further maintenance.
     - Typically used for CPU registers and cache memory.
  2. Dynamic RAM, **DRAM**:
     - A tiny capacitor is used to store one bit.
     - Due to leakage of charge, DRAMs must be refreshed to retain contents.
     - Read operation is destructive in DRAMs.

14. DRAM Types

- FPM DRAMs
  - FPM = Fast Page Mode
- EDO DRAMs
  - EDO = Extended Data Out
  - Uses pipelining to speedup access
- SDRAMs
  - Use an external clock to synchronize data output
  - Also called SDR SDRAMs (Single Data Rate)
- DDR SDRAMs
  - DDR = Double Data Rate

- Provides data on both falling and rising edges of the clock

- RDRAMs
  - Rambus DRAM

15. Instruction Examples

```
add [eax], edx       ; requires all phases of instruction cycle.

add eax, 5           ; does not require Evaluate Address Phase

mov eax, [myaddress] ; does not require the Execution Phase
```

16. Changing the Sequence of Execution

- Instructions are executed in sequence, second instruction follows the first, and so on.
- Arithmetic, logic, and data movement instructions follow this criteria.
- Another type of instruction, *control instruction*, changes sequence of instruction execution.
- Control instructions change PC, (Instruction Pointer register **EIP** on 32-bit Intel x86 platforms) during the Execute Phase of the Instruction Cycle.
- New value for PC is obtained during Fetch Phase from the instruction operand.
- As soon as new instructions cycle begins, next instruction to fetch will be obtained at the new PC address.

17. JMP Instruction Example

- Let's assume that register **EAX** stores a *memory offset*.
- The memory offset is a 32-bit absolute value, which can be added or subtracted from the current address of program execution found in the instruction pointer EIP.
- Consider x86 JMP instruction:

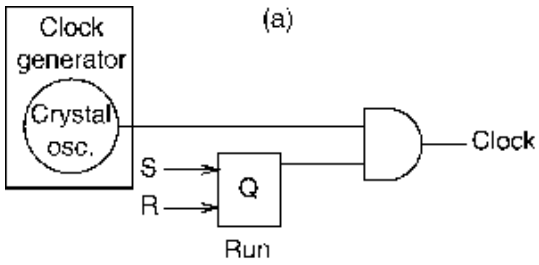
```
jmp eax
```

- The JMP transfers control to the new address by modifying the program counter:

$$EIP_{new} = EIP_{current} + EAX.$$

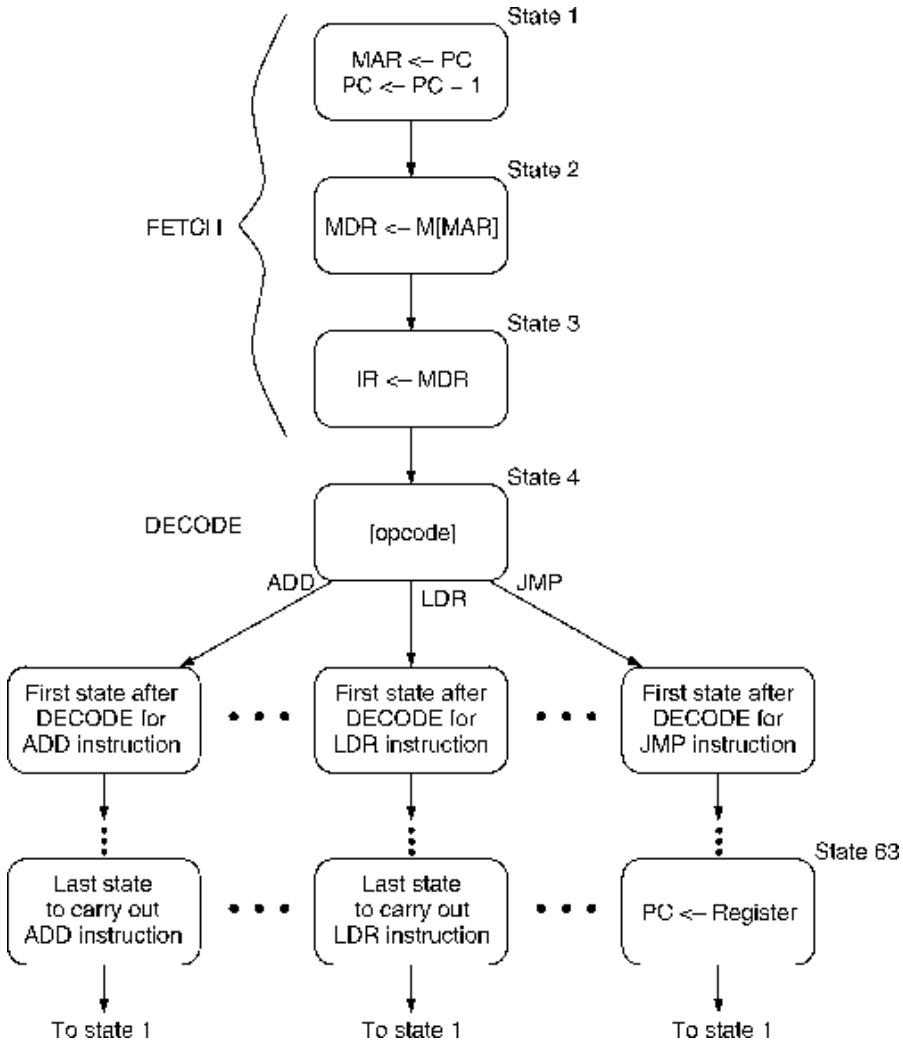
- The result of this operation is *unconditional jump* to a new address in the program.
- Opcode of this x86 instruction is **0FFh**.

18. Instruction Cycle FSM



The diagram on the right shows a very abbreviated instruction cycle FSM of **LC-3**, the *little computer*.

Each state corresponds to one clock cycle.



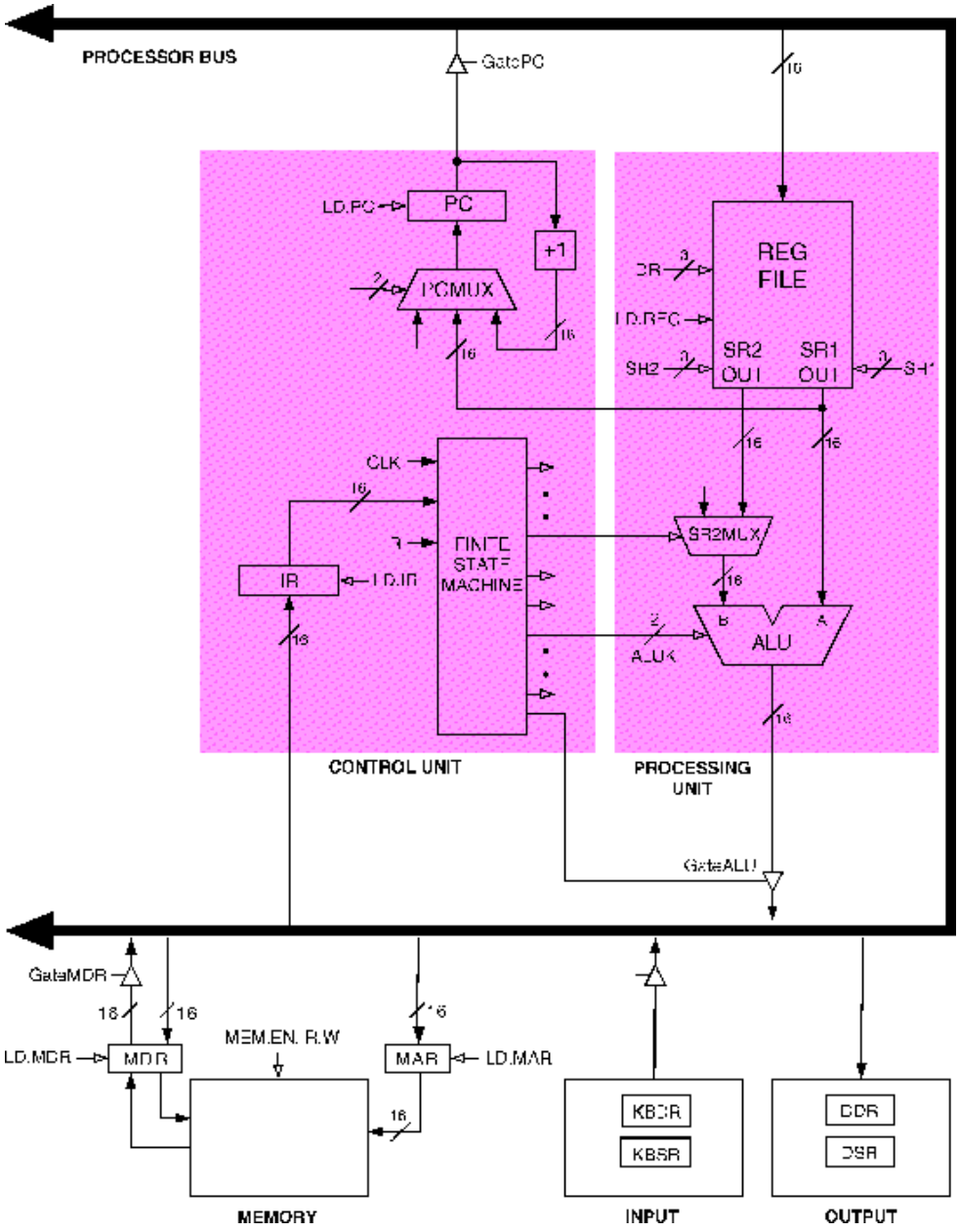
19. Micro-instructions and the Microcode

- A computer program is executed as a series of individual *CPU instructions*.
- CPU instructions constitute a *machine-language level* of abstraction.
- Collections of logic circuits necessary to execute individual instructions are called *microcode*.
- At the microcode level, there is a collection of *programs* that actually

LC-3, the little computer:

implement the instructions.

- The micro-instructions are normally stored in permanent memory of the CPU itself.



20. Microcode Details

- A CPU that uses microcode has a collection of internal scratchpad registers that are not directly accessible outside of the CPU.
- Simple circuits, such as an adder, wired together, are part of the microcode.
- Machine language instructions do have access to the scratchpad registers.
- Although microcode resembles machine language, there are many differences:
    - Micro-instructions typically rely on bits that directly control circuits, such as *master-slave flip-flop*.
    - No program counter is required: each instruction *is hardwired* to the next instruction.
- In general, micro-programming is more complex than assembly language programming.

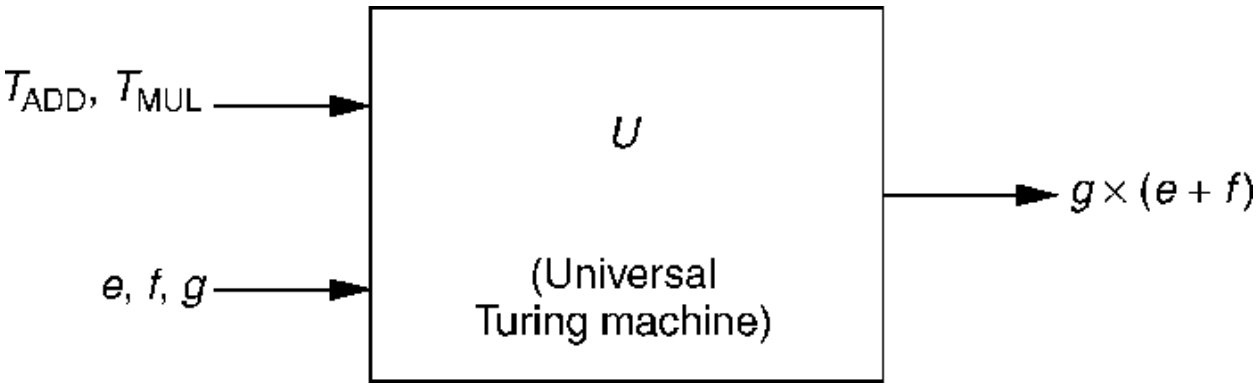
21. Turing Machine

An idea of a *general-purpose programmable computing device*, such as *traffic danger sign* discussed earlier, could be implemented by **universal Turing Machine**:

Turing machine is a mathematical model of a device that...

...computes via a series of discrete steps, and

...is not limited in use by a fixed amount of data storage.



The operations are limited to reading and writing symbols on an *infinitely long tape*.

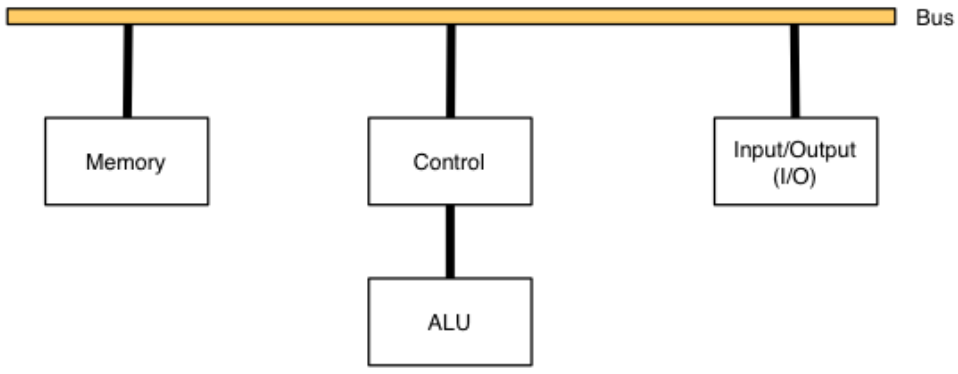
At each step the machine reads the symbol at the current position on the tape.

- This idea still exists in modern computer design with low-level microcode, which directs the reading and decoding of higher level machine code instructions.

22. Von-Neumann Machine Summary

In contrast to a Turing machine, a **von Neumann machine** has a *random-access memory* (RAM) which means that each successive operation can read or write *any memory location*, independent of the location accessed by the previous operation.

The von Neumann Machine:



A von Neumann machine also has a *central processing unit* (CPU) with one or more *registers* that hold data that are being operated on.

23. Von-Neumann Machine Cont.

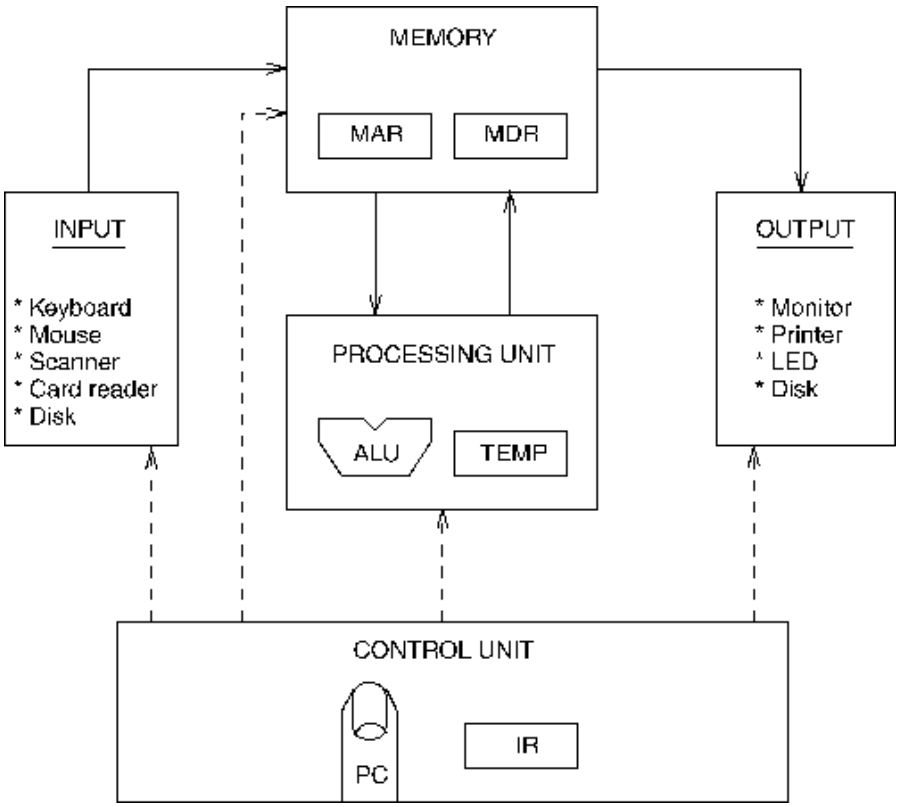
- The CPU has a set of built-in operations (its *instruction set*) that is far richer than with the

The von Neumann Machine:

Turing machine, including the arithmetic, logic, interrupt facilities, and memory access:

- devices: adders , subtractors , comparators , multipliers , dividers , multiplexers , decoders , and so on.
- ability to *branch* to another part of a program, if the binary integer in some register is equal to zero (conditional branch),
- stack operations, etc.

- The CPU can interpret the contents of memory either as *instructions* or as *data* according to the *fetch-execute cycle*.



24. Von-Neumann Architecture Requirements

---

1. *Single stream* of instructions, sequenced by *instruction counter*.
2. Instructions *stored with data* in addressable memory.
3. Instructions encoded as numbers and are *modifiable* by arithmetic operations.
4. Radix 2 (binary) storage and math.
5. Word length is *long enough* for scientific computation.
6. *Single address* instructions, or *single operand* instructions, means that single, well-defined operations use exactly one *memory address* in combination with a *register operand* or *immediate data*.

   See also: Wikipedia article about instruction set (ISA), Von Neumann model , and PC architecture articles on the web.