

## Notes: Ch. 4: von Neumann Model

### A. Why?

The von Neumann model of computer is the kind we use.

### B. Outcomes

By the end of the class you should

- Know that in the von Neumann model, programs are stored in memory as data.
- Know that a Central Processing Unit contains a Processing Unit and Control Unit.
- Know how memory is accessed using memory address and data registers.
- Know what the ALU (Arithmetic Logical Unit) and data registers do
- Know that the Control Unit contains an Instruction Register and Program Counter.
- Know the different parts of the instruction cycle and what happens during them.
- Know the fundamental differences between data movement, calculation, and control instructions.

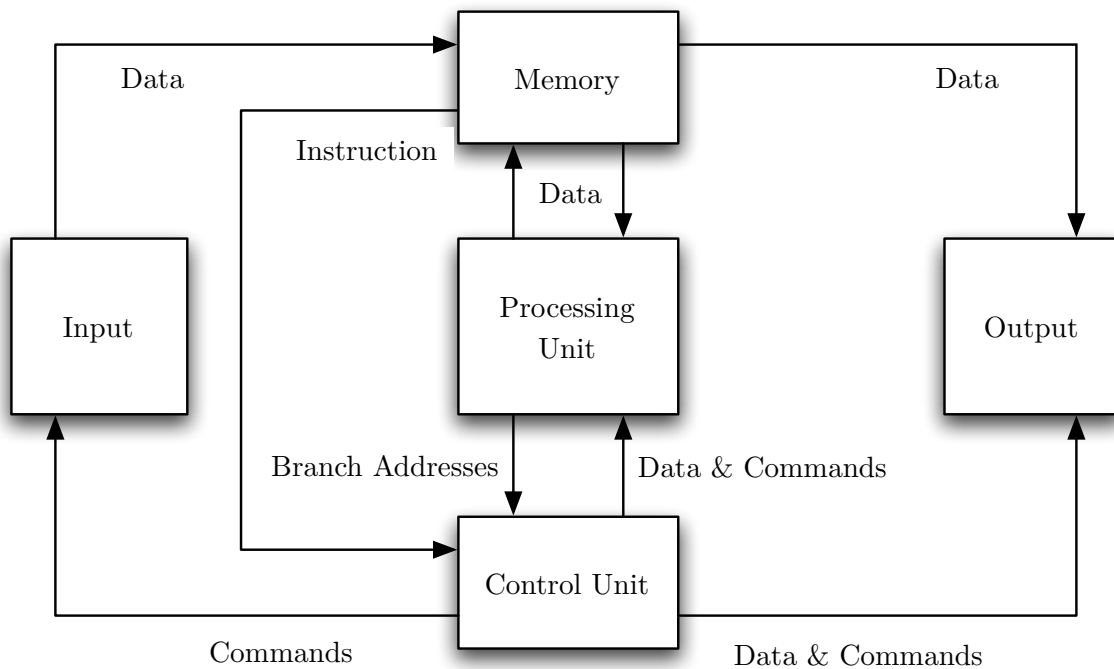
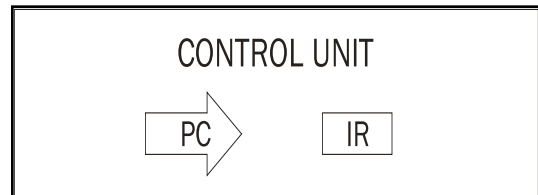
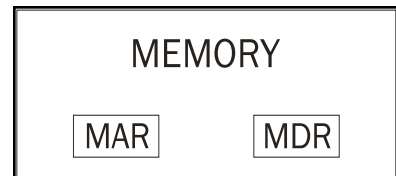
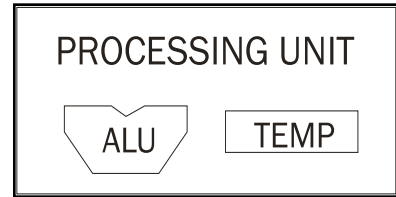
### C. Early Electronic Computation Devices

- Programs not stored in memory:
  - 1939: ABC (Atanasoff-Berry Computer)
    - No CPU, but did math & binary logic.
    - Not programmable (built for one kind of problem).
  - 1943: ENIAC
    - Presper Eckert and John Mauchly -- first general purpose electronic computer.
    - Hard-wired program -- settings of dials and switches.
- Programs stored in memory
  - 1944: Beginnings of EDVAC
    - among other improvements, includes program stored in memory
  - 1945: John von Neumann
    - wrote a report on the stored program concept, known as the First Draft of a Report on EDVAC.

### D. von Neumann machine

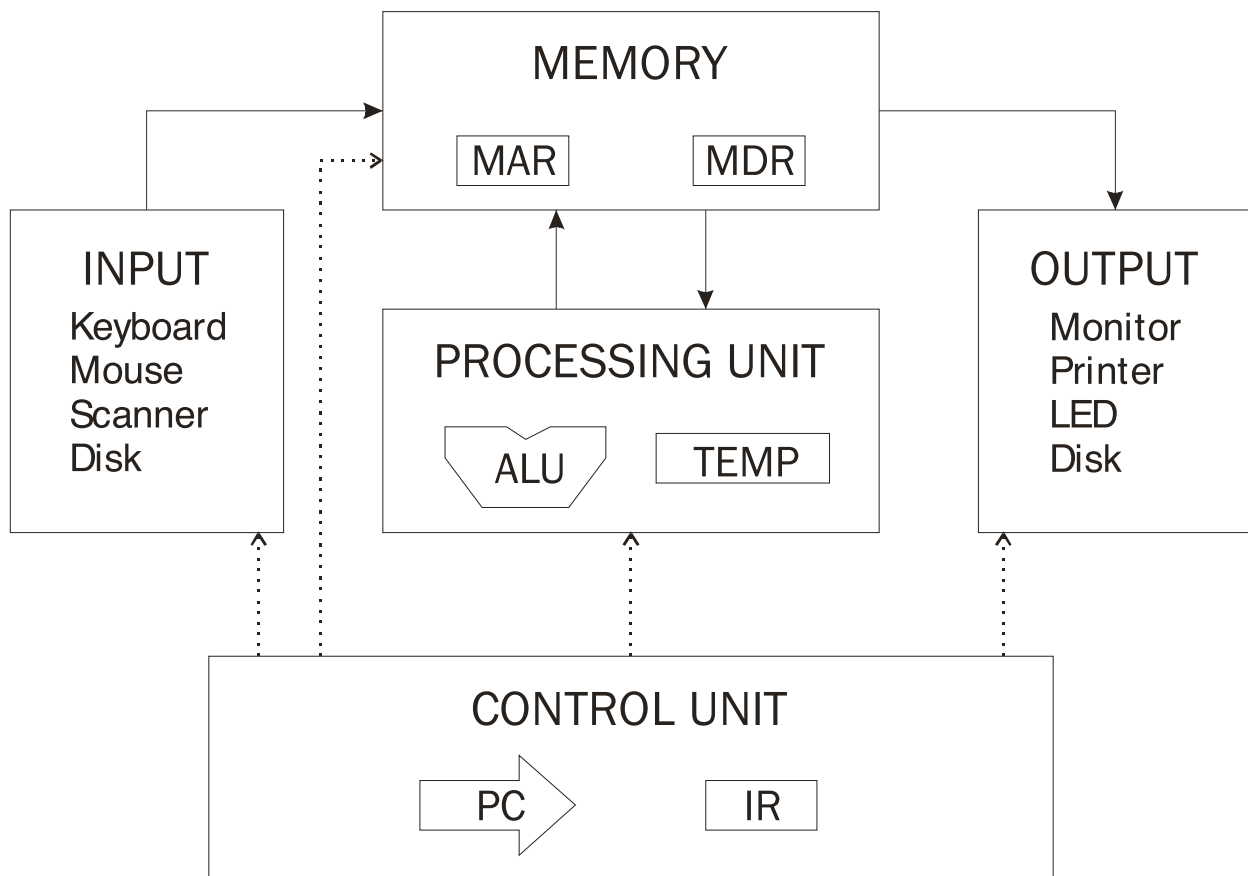
- The basic structure proposed in the draft became known as the “von Neumann machine” (or model).
- A **memory**, containing instructions and data
- **Central Processing Unit**
  - A **processing unit**, for performing arithmetic and logical operations
  - A **control unit**, for interpreting instructions
- Input and Output devices

- Processing Unit
  - **ALU** = Arithmetic and Logic Unit
  - **Registers** - Small, fast temporary storage
  - **Word Size** = width of registers.
  - Gets commands from Control Unit
  - Reads memory, writes memory
- Memory
  - Address:  $K$  bits; each address stores  $M$  bits.
  - **Memory Address Register (MAR)** ( $K$  bits wide): Contains address to read/write
  - **Memory Data Register (MDR)** ( $M$  bits wide): Contains value read/written
- **Control Unit (CU)**
  - Controls execution of the program.
  - **Instruction Register (IR)** contains the current instruction.
  - **Program Counter (PC)** contains the memory address of the next instruction to be executed. (“Instruction Pointer” would probably be a better name, but PC is traditional.)



Typical von Neumann Computer

- Input & Output
  - Keyboard, mouse, disk, video, printer, etc.
  - Receives commands from processing unit (or control unit; depends on design).
  - Sends data from memory/Receives data into memory (or processing unit; depends on design).
  - Sends **interrupt signals** to control unit when done with operation.



Another Typical View of a von Neumann Computer

- Actual computers are more sophisticated than original von Neumann design
  - Many processing units (e.g. floating-point)
  - I/O processing offloaded to special hardware (e.g. video cards)
  - I/O goes directly into/out of memory (avoids bottleneck of processing unit or control unit)
  - Control unit works on > 1 instruction at a time (execute current instruction while decoding next instruction while loading one after that into memory).
  - Can have multiple control unit/processing unit pairs (multi-core CPUs)

## E. Instruction Format, Instruction Set Architecture

- Instruction has an Opcode (Operation code): Binary number for instruction
- Specifies some number  $\geq 0$  of operands (possibly in memory, possibly in registers)
- Specifies some number  $\geq 0$  of results.
- Note: Different instructions can take different amounts of times.
- **Fixed-length instruction** machine: Easier to design, limits instruction designs.
- **Variable-length instruction** machine: More flexible instruction sets, more compact use of memory, hardware more complicated.
- **Instruction Set Architecture (ISA)**
  - The set of instructions for a computer, their formats, the way they specify opcodes and operands and results.

## F. Instruction Cycle

- A computer executes a program by executing a sequence of instructions.
- Get instruction from memory, execute it; get instruction from memory, execute it, etc.
- Instruction cycle is how one instruction is executed.
  - (1) Fetch instruction; (2) Decode instruction; (3) Evaluate address(es) of operands; (4) Fetch operand(s); (5) Execute instruction; (6) Store results of execution.
  - Details vary by instruction; not all instructions have phases 3, 4, 6 (depends on whether/how many operands it uses and whether/how many results it produces).
  - Operands/results might be in memory or in registers.
  - Whether some bits specify an instruction or data depends on how we use it.
  - Look at generic instruction cycle first; study a specific version in a bit.
- **Fetch instruction**
  - Program Counter contains a memory address; get contents of that address, copy them into Instruction Register.
  - Increment Program Counter (so that it points to next instruction in memory).
- **Decode instruction**
  - Which instruction do we have? Use decoder on opcode bits.
  - Depending on instruction, retrieve bits from other parts of instruction.
- **Evaluate Addresses** (= Get effective addresses of operands)
  - If the instruction has operands, then for each operand, figure out which register or memory location the operand is stored at.
- **Fetch operands**
  - If the instruction has operands then retrieve them.
    - Might be in registers, in memory, or encoded within instruction itself.
- **Execute instruction**
  - Depending on instruction type, do something

- **Data movement instruction:** Load value to/Store value from memory.
- **Calculation instruction:** Perform some function on data (add, subtract, etc).
- **Control (Branch/Jump) instruction:** Change the program counter.
  - Program Counter (PC) points to next instruction to execute; changing PC causes next instruction to come from somewhere else. (Good for decision, iterative statements.)
- **Store Results**
  - Take result(s) of load-from-memory instructions and calculation instructions and put them somewhere. (Might be register(s), memory locations.)
  - Some instructions don't typically produce results. (E.g. store-to-memory, branch.)

## G. Simple Decimal Computer

- Here's the instruction set architecture for a very simple computer: Instructions are 4 decimal digits long: NRMM, where N is an opcode (0-9), R is a register number (0-9), and MM is a memory location (00-99).

Opcode	Meaning
0	HALT execution
1	Load register R from memory location MM
2	Store register R into memory location MM
3	Add contents of memory location MM to register R
4	Not: Do bitwise NOT on contents of register R (ignore MM)
5	Load register R with MM (not contents of MM) [An "immediate" operand; we're not using it as an address.]
6	Add MM (not contents of location MM) to register R
7	Branch to location MM (ignore R)
8	Branch to location MM if register R contains zero.
9	Unused (causes runtime error)

When the machine powers up, it sets all registers to zero and starts by executing the instruction at location 00.

- Here's a sequence of code that adds 1 to the contents of memory location 99, using R0 (register 0) as temporary storage:
  - 1099, 6001, 2099
- Here's code that sets R1 to -1 by taking the 2's complement of 1:
  - 5101, 4100, 6101
- Assuming R1 contains -1, this code subtracts 1 from R0 and branches to location 57 if the result is 0. (Hope there's something interesting at location 57 :-). It uses location 98 as temporary storage.
  - 2198, 3098, 8057

## A. The LC-3 Computer

- Text uses the Little Computer version 3
- 16-bit words, 8 data registers (named R0 – R7), 4-bit opcodes (16 instructions)
- Four kinds of operands
  - Immediate (part of instruction)
  - Register (number 000, 001, ..., 110, 111)
  - Memory address
    - Base offset: Requires a register number and a binary number: Add contents of register to binary number; result is a memory address.
    - PC offset: Requires a binary number: Add contents of Program Counter and binary number; result is a memory address.
      - Program counter points to next instruction, not current instruction, so offset=0 specifies next instruction, not current instruction.
- Not every instruction has every kind of operand.