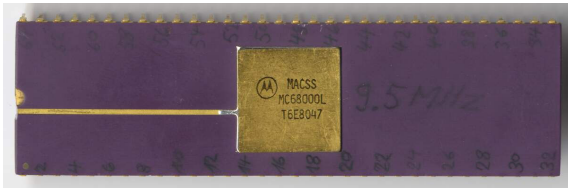


System Architecture

- We started from transistor level, and have built up from there:
 - transistors → gates
 - gates → components
 - components → machines
- Next level: architecture of a CPU



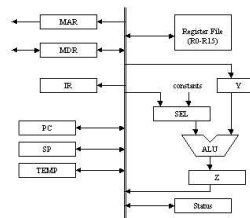
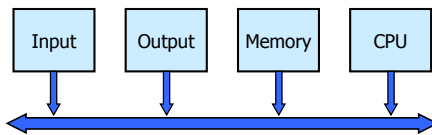
System Components

- Any system (microcontroller, PC, mainframe) has four basic components to it:
 - Input** (keyboard, mouse, network)
 - Output** (screen, speakers, network)
 - Memory**
 - Processor**
- These components are connected through a series of wires called a **bus**.
 - typically 40-100 wires
 - divided into address, data and control
 - multiple buses can exist in a single system, to increase the system's efficiency and allow parallel processing



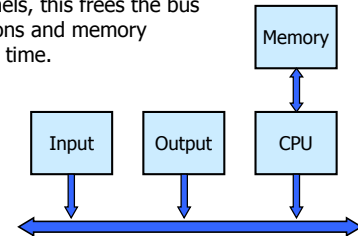
Buses (or Busses)

- Buses exist both inside and outside the CPU.
- Outside the CPU, the bus allows communication between system components.
- Within the CPU, the bus is the main channel of data through which processing takes place.
 - only one operation can be driving the bus at a time (the **bus driver**).



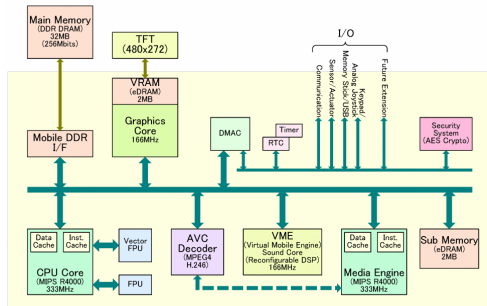
Bus Modifications

- By experimenting with bus configurations, one can achieve improvements to the system's performance.
- Example: attaching memory to the CPU directly.
 - If memory doesn't need to interact with the input and output channels, this frees the bus to perform I/O operations and memory operations at the same time.
 - Doesn't work well for architectures that represent I/O as a memory operation.



Bus Example

- Playstation Portable (PSP)
 - Example of a more complex architecture



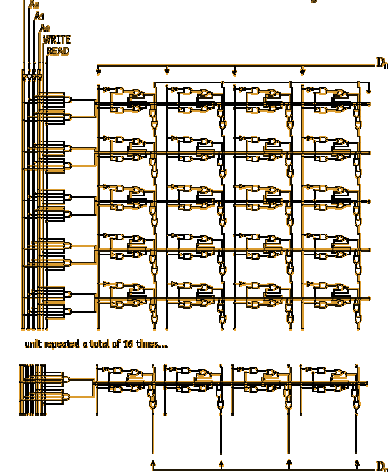
External Input/Output

- Generally considered extraneous to this course.
- Most input and output is either mapped to certain locations in memory, or modeled to look like it's mapped to these memory locations.
 - called **memory-mapped I/O**
- Each device would have its own location in memory for reading or writing (or both). To read from a device, go to that position in memory, and retrieve the byte(s) of input data.
 - other flags may exist to indicate that a new byte of data is ready to be read, or may be set to indicate reading/writing status.

Memory

- Memory stores groups of bits called words
 - The number of bits in each word is called the word length, and can range from 16 to 64 bits, typically.
- Many different types, based on the application:
 - ROM** (Read-Only Memory)
 - contents of memory are fixed; cannot be reprogrammed.
 - EPROM** (Erasable Programmable ROM)
 - erased using ultra-violet light, through small window in chip.
 - needs high voltage to be programmed.
 - EEPROM** (Electrically Erasable Programmable ROM)
 - no ultra-violet light needed.
 - Flash Memory**
 - a special type of EEPROM.

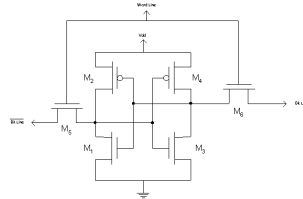
Schematic for a 16 x 4 bit memory



Memory

(continued from previous slide)

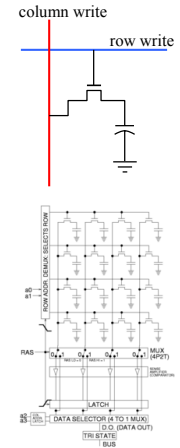
- **RAM** (Random Access Memory)
 - memory used in programming to store data and methods.
 - As opposed to ROMs, RAM loses its contents when the system is powered off.
- **SRAM** (Static RAM)
 - Fast memory, but requires 5 or 6 transistors per bit.
 - 5-20 ns to perform a read or write operation.
 - used for cache memory.



Memory

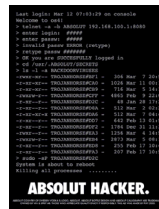
(continued from previous slide)

- **DRAM** (Dynamic RAM)
 - relatively slow, but only requires one transistor per bit.
 - 35-100 ns per operation.
 - used for computers' main memory
 - can be "leaky" → requires refresh circuitry.
- **VRAM** (Video RAM)
 - stores contents of current screen display.
 - memory access time must be faster than screen display rate (e.g. 60 Hz)
 - separate outputs
 - read and write from CPU.
 - output to video display.



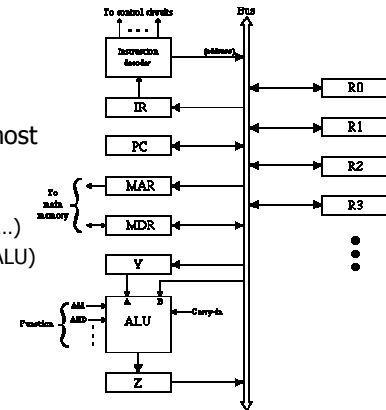
Notes About Memory

- Memory units are **word-addressable**
 - locations in memory are specified with 16- or 24-bit words.
- Memory data is typically stored in **hexadecimal**
 - 0100 0101 1100 1010 0101 0101 1111 1101 is 45cb55fd.
- Most systems implement von Neumann architecture:
 - CPU instructions are stored in memory
 - No way to distinguish between CPU instructions and regular memory data
 - Provides opening for hackers ☹



Central Processing Unit

- The CPU is the brains behind the operation.
- Architectures for this unit may differ, but most have the same basic units:
 - **data registers** (R0, R1...)
 - **arithmetic logic unit** (ALU)
 - **program counter** (PC)
 - **main memory access** (MAR, MDR)

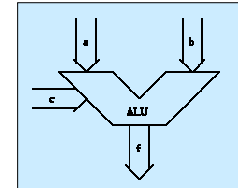


CPU Components

- MAR (**Memory Address Register**)
 - stores the address that specifies a location in memory to read from or write to.
- MDR (**Memory Data Register**)
 - stores the data that is either sent to or received from memory.
- PC (**Program Counter**)
 - stores the memory location of the current instruction.
- IR (**Instruction Register**)
 - stores the current instruction being executed.
- Data Registers (R0, R1,...R7)
 - stores data temporarily during internal operations.

CPU Components

- Address Registers (A0, A1,...A7)
 - temporary storage of address values, to be used in current operation (e.g. retrieving multiple memory values).
- ALU (**Arithmetic Logic Unit**)
 - performs arithmetic (ADD, SUB, COMP, NEG) and boolean (AND, OR, NOT, XOR) operations. Sometimes multiplication is implemented as well, but involves increase in complexity.
 - Operates on two words at a time.
 - Always drawn in manner on right, with data input from the top and flowing to bottom. Operation is specified through bits on the left.
 - Y and Z are used to store input and output temporarily.



Instruction Decoder

- The instruction decoder is the most complex of all the components.
- Its responsibility is to take in instruction words, and decode them into signals that activate all the other components of the system.
 - every arrow on the diagram has to be activated through one of these signals.
 - buffers prevent signals from entering or leaving the components when the instruction decoder's signal is low.

Instruction Decoder

- Two types of CPU architecture:
 - RISC (**Reduced Instruction Set Computer**)
 - Simple instructions.
 - Each instruction takes the same amount of time to execute.
 - CISC (**Complex Instruction Set Computer**)
 - Instructions can vary in terms of execution time.
 - Easier to program, since lower operations are abstracted away by instruction decoder.
 - Involves a lot of state information within instruction decoder.