Basic computer operation and organization

From an engineering viewpoint a computer manipulates coded data and responds to events occurring in the external world. This is called a stored-program or von Neumann machine architecture.

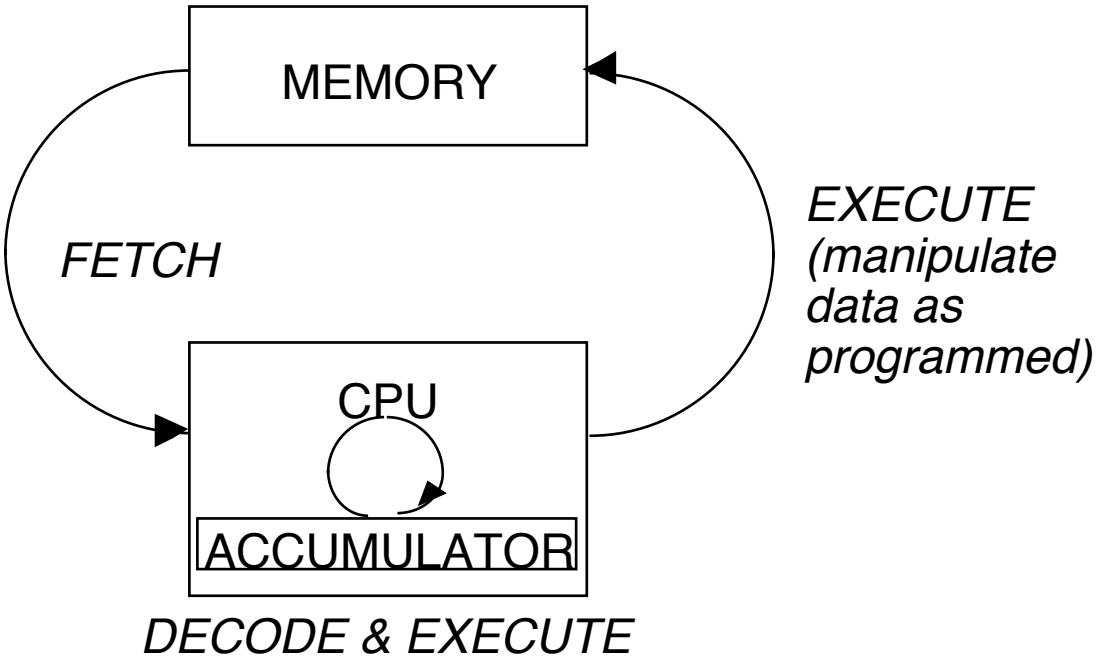
- memory - used to store both programs and data instructions (this is the core of the von Neumann architecture)
  - program instructions are coded data which tells the computer to do something, i.e. adding two numbers together
  - data is simply information to be used by the program, i.e. two numbers to be added together

  $\Rightarrow$We need something to decode the memory and determine what represents instructions and what represents data
- central processing unit gets instructions and/or data from memory, decodes the instructions, and performs a sequence of programmed tasks

Nothing can occur simultaneously or instantaneously in a computer. Important operations are

- fetching instruction(s) from memory
- decoding the instruction(s)
- performing the indicated operations

This is the basic “fetch-execute” cycle



Problems with this diagram:

- what memory is being fetched?
- how does the computer tell program instructions from program data
- what happens when the program needs more than one piece of data?

Answers:

- put program instructions and data in separate areas of memory. These don’t have to be well organized, but do need to be defined and can be intermixed. Usually program instructions are kept together.
- Internal pointers kept in special locations called registers in the CPU keep track of what data and

program instructions are being referenced and/or fetched.
- The CPU has local storage in special locations called registers for temporary storage of data and/or instructions.

Keeping track of what instruction is being executed is so important that a special CPU register called the program counter is used to keep track of the address of the instruction to be executed.

Central Processing Unit

Control unit:
- decodes the program instructions
- program counter which contains the location of the next instruction to be executed
- status register which monitors the execution of instructions and keeps track of overflows, carries, borrows, etc.

RISC reduced instruction set machine
- executes a simple set of instructions very fast

CISC complex instruction set (such as the 68000)

- has a powerful set of instructions which allows many complex operations to be represented as a single instruction

Arithmetic Logic Unit

- carries out the instructions decoded by the control unit

Older microprocessors had special registers called accumulators which had to be used for math calculations. Modern microprocessors such as the 68000 have general-purpose registers and do not have accumulators.

Memory

- ROM read-only memory
  non-volatile $\rightarrow$ all bits can be read, no bits can be changed
- RAM random access memory
  (not really a good name since almost all memory has random access capability)
  volatile $\rightarrow$ all bits can be read and/or written

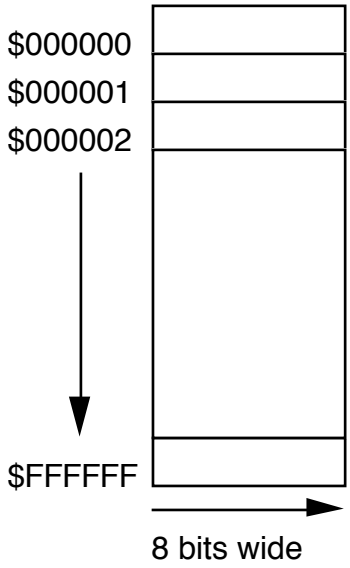
User concern with memory

- large computer usually don't even think about memory organization
- mini computer sequence of RAM, how much RAM, etc.
- micro computer memory constraints on RAM, ROM. Very limited address space.

Computer memory is always organized in a fixed manner:

- 16k x 1 bit
- 4k x 1 bit
- 8k x 8 bits typical of small microcomputers

Vertical grid organization of memory:



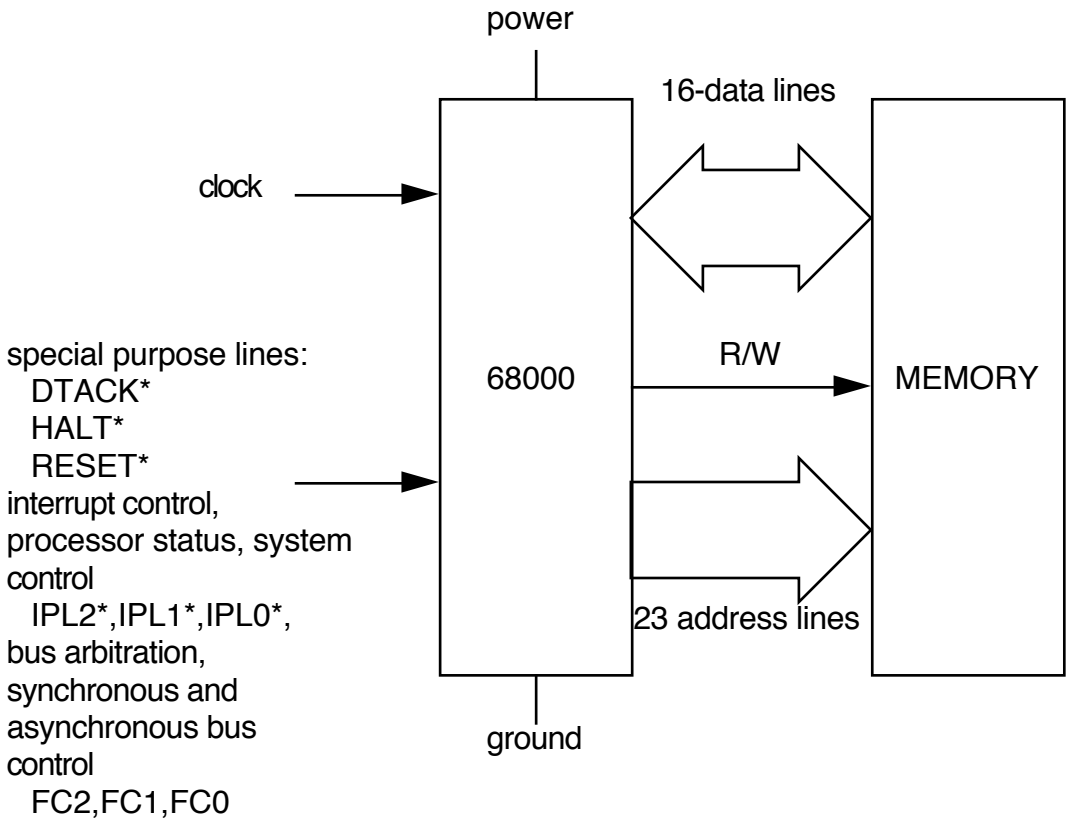
This diagram represents $FFFFFF = 2^{24} bytes of memory

- Any particular processor will have an x-bit address capacity:

64k	6502, Z-80, 8085, etc.
640k	8088
16MB	68000
lots	80386, 68030

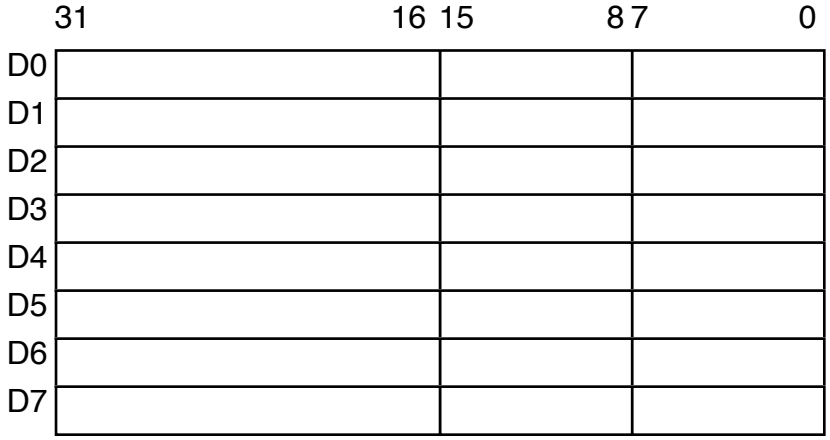
- memory addresses do not need to be contiguous
- ROM and RAM can be intermixed
- memory does not have to start at $000000 or end at $0FFFFF

68000 architecture:

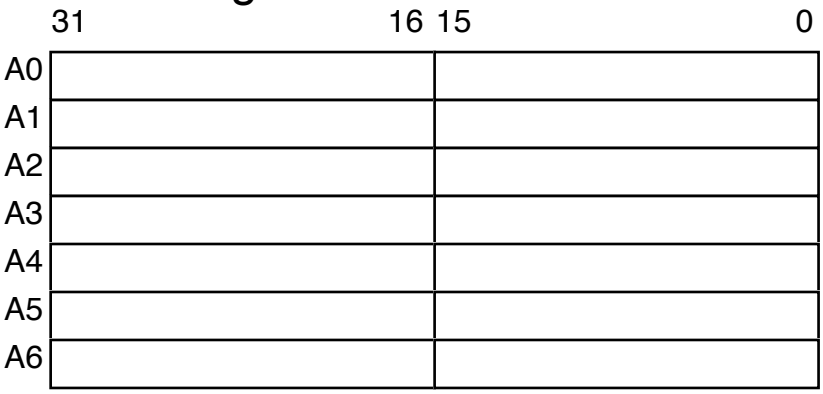


68000's internal register organization:

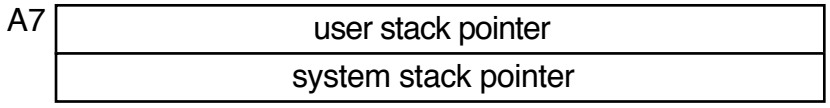
data registers:



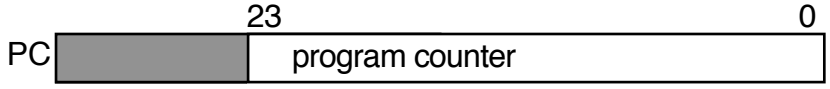
address registers:



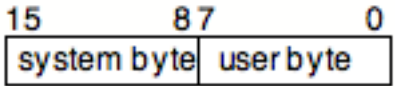
stack pointers:



program counter:

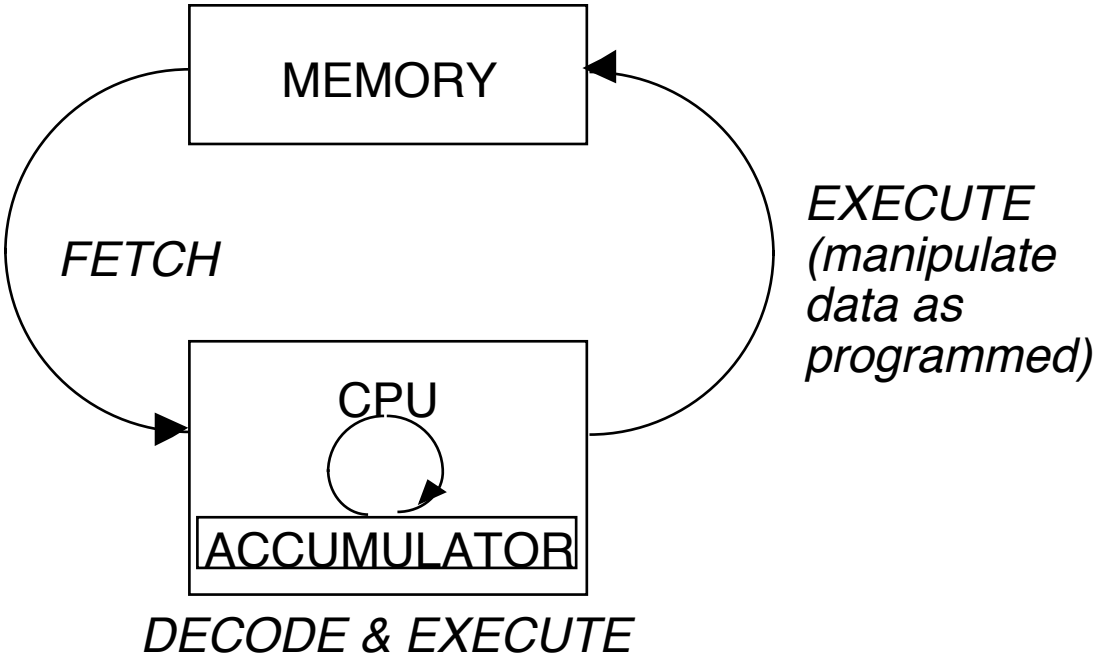


status register:



SOFTWARE ARCHITECTURE:

Fetch-execute revisited:



Basic fetch-execute cycle:

```
loop:   fetch_instruction
execute_instruction
goto loop.
```

Program counter:

Because the computer must keep track of instruction locations it uses the program counter to keep track of the address of the next instruction to be executed:

```
loop:   instruction_register = fetch_instruction(pc_address)
decode_instruction(instruction_register)
execute_instruction(instruction_register)
goto loop.
```

Instruction_register is an internal (to the processor) memory location (register) used to store coded data (instructions).

Instructions are coded data in the following format:

op_code source(s) destination next_instruction

The exact way this information is represented is different for every microcomputer. Present microcomputers typically code instructions in several somewhat simplified formats

op_code source1 source2/destination

or

op_code source/destination

where source, source1 and source2 identify where any needed data is to be obtained and the result (if any) is to be placed in a destination which is also the second source of needed data. For example, x:=x+y is represented in the first format as

ADD Y X

Add processing to decode instructions:

Our original fetch-execute cycle has now become more complex:

```
loop:   instruction_register = fetch_instruction(pc_address)
decode_instruction(instruction_register)
while extension_flag set do
    fetch additional information
    update pc_address
    end while.
execute_instruction(instruction_register)
update pc_address.
update status register.
goto loop.
```

Instructions which require more than one computer word to describe can be indicated by extension_flag and fetched until the instruction is complete.

The exact manner in which memory locations are represented is known as addressing and can directly effect the program execution speed of the computer.