- Here, we study computer organization
  - Structure: how computer components are connected together, how they communicate
  - Function: what these components do
- We start with the most important component, the CPU (central processing unit or processor)
  - This is the brain of the computer, it does all the processing
  - The CPU is in charge of executing the current program
    - each program is stored in memory along with data
    - the CPU is in charge of retrieving the next instruction from memory (fetch), decoding and executing it (execution)
    - execution usually requires the use of one or more circuits in the ALU and temporary storage in registers
    - some instructions cause data movement (memory accesses, input, output) and some instructions change what the *next instruction* is (branches)
  - We divide the CPU into two areas
    - datapath – registers and ALU (the execution unit)
    - control unit – circuits in charge of performing the fetch-execute cycle

Two Kinds of Registers

- User registers
  - These store data and addresses (pointers to data)
    - These are manipulated by your program instructions
      - Example: Add R1, R2, R3
        - » R1 □ R2 + R3
  - Computers will have between
    - 1 and hundreds of registers
    - Possibly divided into data and address registers
  - Registers are usually the size of the computer's word size
    - 32 or 64 bits today, previously it had been 8 or 16 bits
  - Some machines use special-purpose registers
    - each register has an implied usage
  - Others have general-purpose registers
    - use them any way you want to

- Control registers
  - Registers that store information used by the control unit to perform the fetch-execute cycle
    - PC – program counter – the memory location of the next instruction
    - IR – instruction register – the current instruction being executed
    - Status flags – information about the results of the last instruction executed (was there an overflow, was the result positive, zero or negative? Etc)
    - Stack Pointer – location in memory of the top of the *run-time* stack (used for procedure calls and returns)

ALU and Control Unit

- The ALU consists of circuits to perform arithmetic and logic operations
  - Adder
  - Multiplier
  - Shifter
  - Comparitor
    - Etc…
  - Operations in the ALU set status flags (carry, overflow, positive, zero, negative, etc)
  - Also, possibly, temporary registers before moving results back to register or memory

- The control unit is in charge of managing the fetch-execute cycle
  - It sends out control signals to all other devices
  - A control signal indicates that the device should activate or perform it’s function
  - For instance:
    - Instruction fetching requires
      - sending the PC value to main memory
      - signaling memory to read
      - when the datum comes back from memory, move it to the IR
      - increment the PC to point to the next instruction
    - These operations are controlled by the control unit
    - Now the control unit decodes the instruction signals the proper ALU circuit(s) to execute it

The System Clock

- In order to regulate when the control unit issues its control signals, computers use a system clock
  - At each clock pulse, the control unit goes on to the next task
    - Register values are loaded or stored at the beginning of a clock pulse
    - ALU circuits activate at the beginning of a clock pulse
    - Typical clock speeds are based on the amount of time it takes to perform one of these actions (register or ALU)

- Clock performance is based on the number of pulses per second, or its Megahertz (or Gigahertz) rating
  - This is a misleading spec
  - The number of clock pulses (cycles) that it takes to execute one instruction differs from one computer to the next
    - Assume computer A takes 10 clock cycles per instruction but has a 1 Gigahertz clock speed
    - Assume computer B can execute 10 instructions in 11 cycles using a pipeline, but has a 250 Megahertz clock speed
    - Which one is faster? B even though its clock is slower!

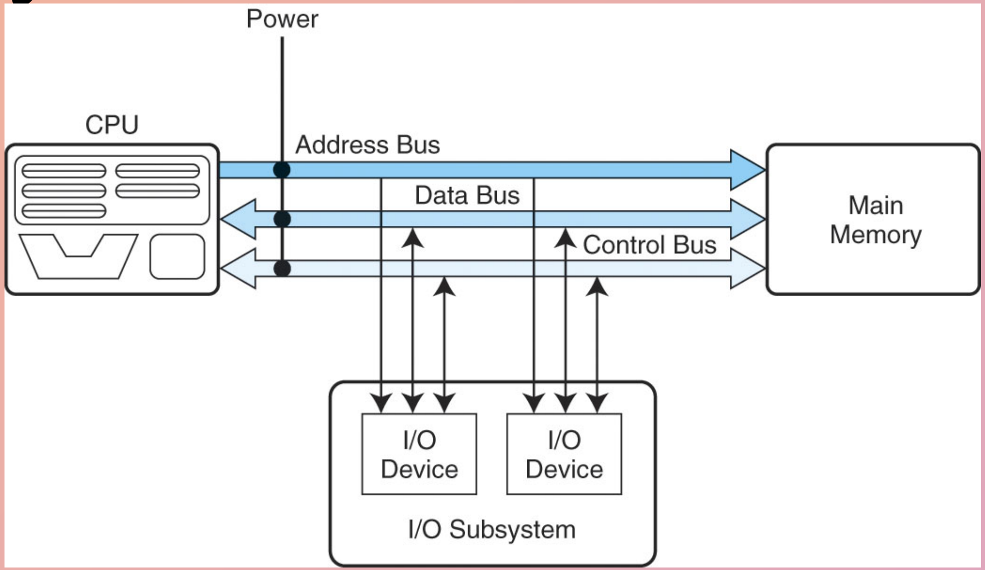
Comparing Clocks

- It is difficult to compute CPU performance just by comparing clock speed
  - You must also consider how many clock cycles it takes to execute 1 instruction
  - How fast memory is
  - How fast the bus is
  - Etc
    - The book offers an example comparing 286 multiply to a pentium multiply
- In addition, there are different clocks in the computer, the Control Unit and the whole CPU are governed by the system clock
  - There is usually a bus clock as well to regulate the usage of the slower buses

The Bus

- System Bus connects the CPU to memory and I/O devices
  - A bus is a collection of wires that carries electrical current
  - The current is the information being passed between components
    - typically the information is a datum or program instruction, but can also include control signals and memory or I/O addresses
  - There are 3 parts to a bus
    - the data bus (for both data and program instructions)
    - the control bus (control signals from the Control Unit to devices, and feedback lines for acknowledging that the devices are ready or for interrupting the CPU)
    - address bus (the address of the memory location or I/O device that is to perform the given data movement operation)
  - Additionally, computers may have multiple buses
    - local bus – connects registers, ALU and control unit together (also an on-chip cache if there is one)
    - system bus – connects CPU to main memory
    - expansion or I/O bus – connects system bus to I/O devices

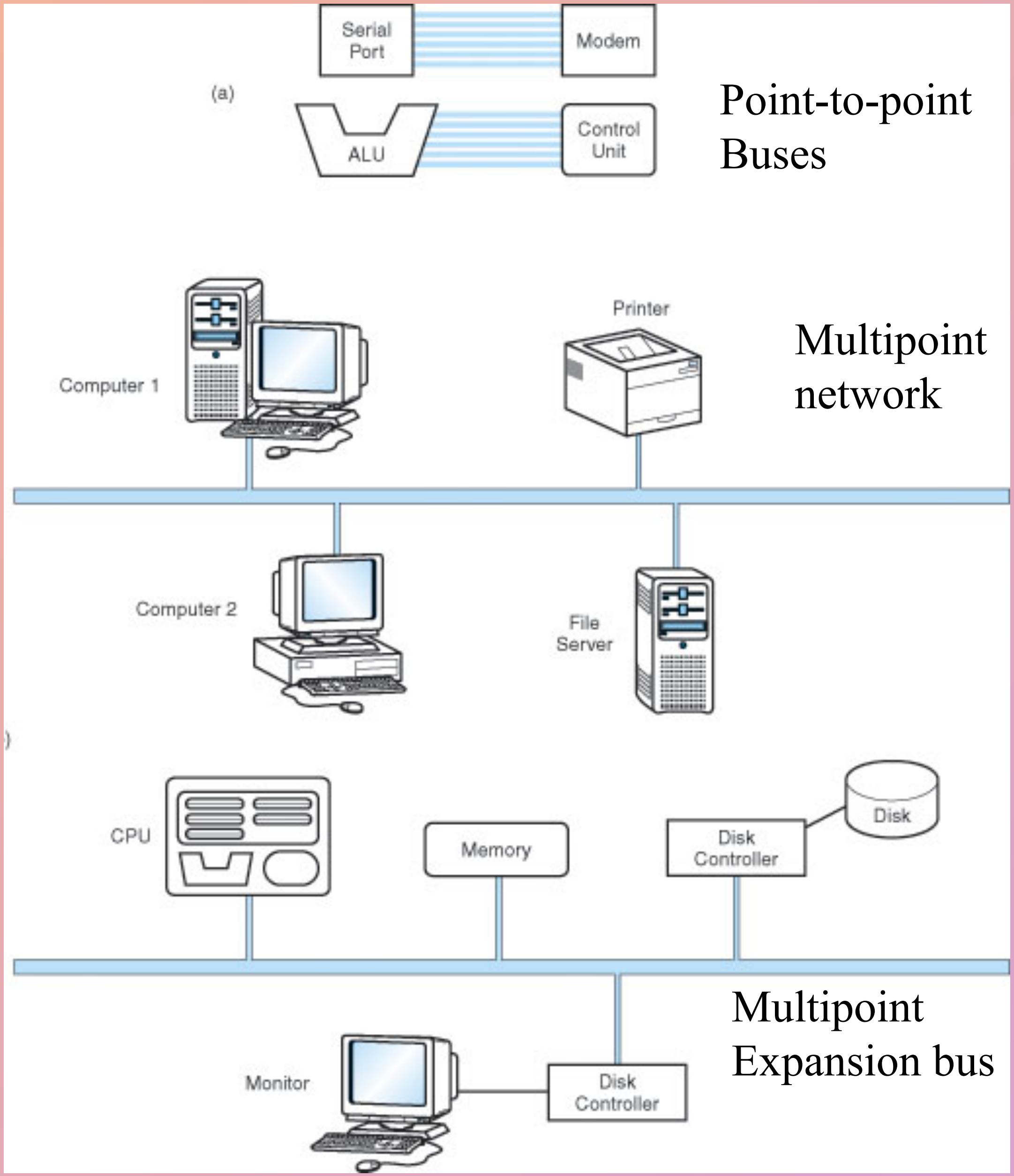
The System Bus



- Here we see the system bus in more detail
- The CPU connects to the bus through pins
- The bus is on the motherboard inside the system unit
- Main memory (a collection of chips) connects to this bus through pins
- The I/O subsystem connects to this bus through the expansion bus
- The bus carries three types of information
  - the address from the CPU of the intended item to be accessed
  - the control information (read versus write, or status information like “are you available?”)
  - the data, either being sent to the device, or from the device to CPU

More on Buses

- Buses connect two types of devices
  - Masters – those devices that can initiate requests (CPU, some I/O devices)
  - Slaves – those devices that only respond to requests from masters (memory, some I/O devices)
- Some buses are dedicated
  - The bus directly connects two devices (point-to-point bus)
- Most buses connect multiple components
  - Common pathway or multipoint



Bus Interactions

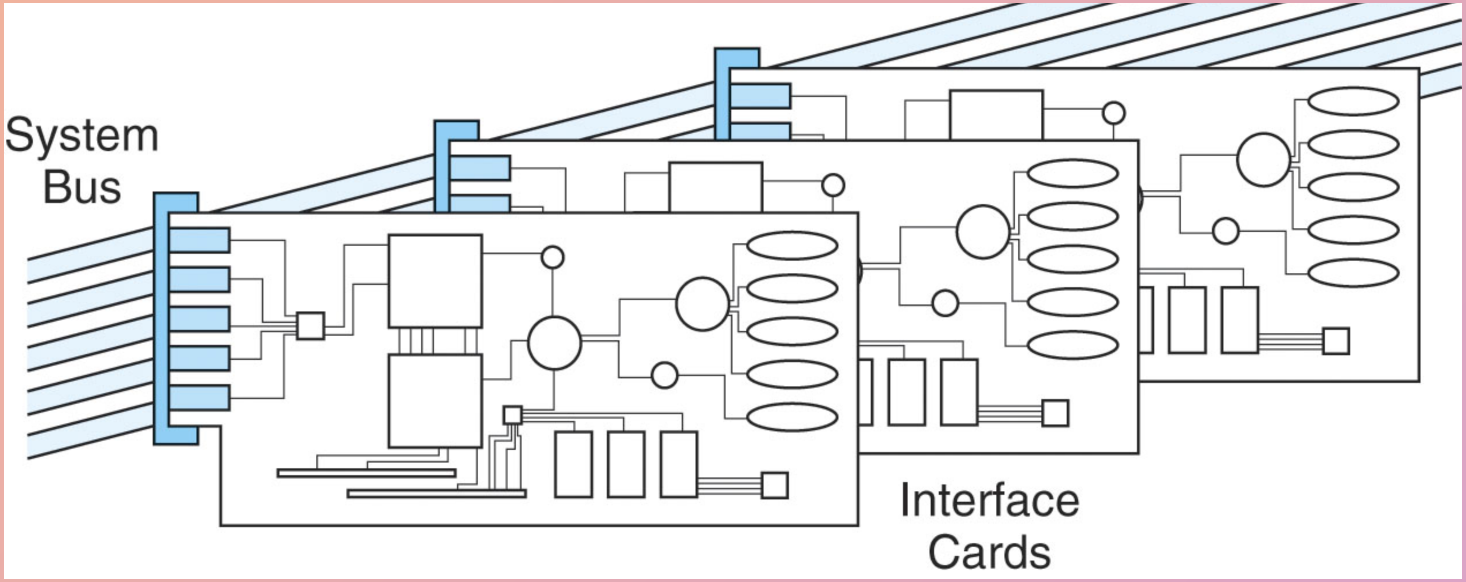
- Except for point-to-point buses, we have to worry about who gets to use the bus
  - Especially the expansion bus where multiple I/O devices may want to communicate between themselves and the CPU or memory at the same time – we need a form of *bus arbitration*
    - Daisy chain arbitration
      - each device has a bus request line on the control bus, when a device wants to use the bus, it places its request and the highest priority device is selected (this is an unfair approach)
    - Centralized parallel arbitration
      - the bus itself contains an arbiter (a processor) that decides – the arbiter might become a bottleneck, and this is also slightly more expensive
    - Distributed arbitration
      - the devices "vote" to determine who gets to use the bus, usually based on a priority scheme, again possibly unfair
    - Distributed arbitration using collision detection
      - it's a free-for-all, but if a device detects that another device is using the bus, this device waits a short amount of time before trying again

I/O Subsystem

- There are many different types of I/O devices, collectively known as the I/O Subsystem
  - Since I/O devices can vary greatly in their speed and usage, the CPU does not directly control these devices
  - Instead, *I/O modules*, or interfaces, take the CPU commands and pass them on to their I/O devices
    - One interface is in charge of one or more similar types of devices
  - To communicate to the right I/O device, the CPU addresses the device through one of two forms
    - Memory-mapped I/O
      - the interface has its own memory which are addressed as if they were part of main memory, so that some memory locations are not used, they are instead registers in the I/O interfaces
    - Isolated I/O
      - the CPU differentiates memory addresses from I/O addresses by having additional control lines indicating whether an address is meant for memory or I/O
        - » we will explore I/O in more detail in chapter 7

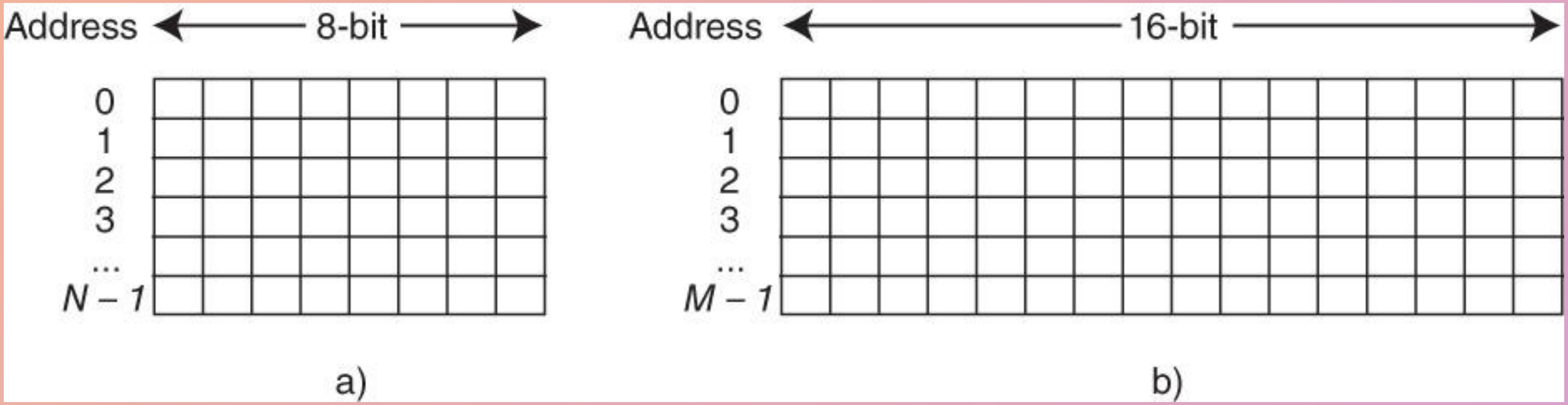
I/O Devices and Bus

- The expansion bus typically is the collection of expansion slots and what gets plugged into them
  - Here we see interface cards (or expansion cards), each with the logic to interface between the CPU and the I/O device (e.g., printer, MODEM, disk drive)
  - Some devices are placed on the card (MODEM), others are plugged into the card through ports



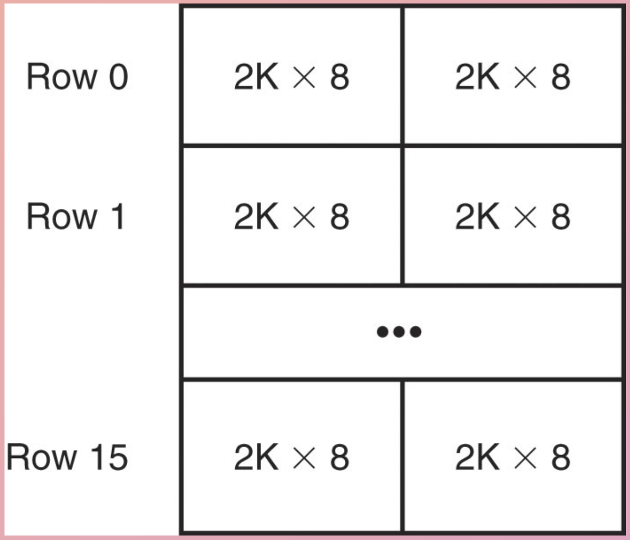
Memory Organization

- Memory is organized into byte or word-sized blocks
  - Each block has a unique address
    - This can be envisioned as an array of cells, see the figure below
  - The CPU accesses memory by sending an address of the intended access and a control command to read or write
  - The memory module then responds to the request appropriately
    - A *decoder* is used to decode the binary address into a specific memory location

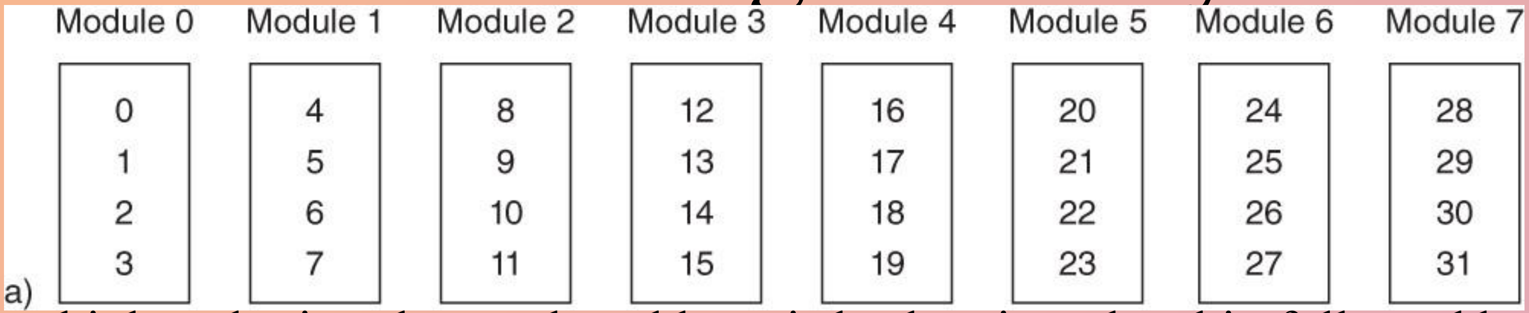


Dividing Memory Across Chips

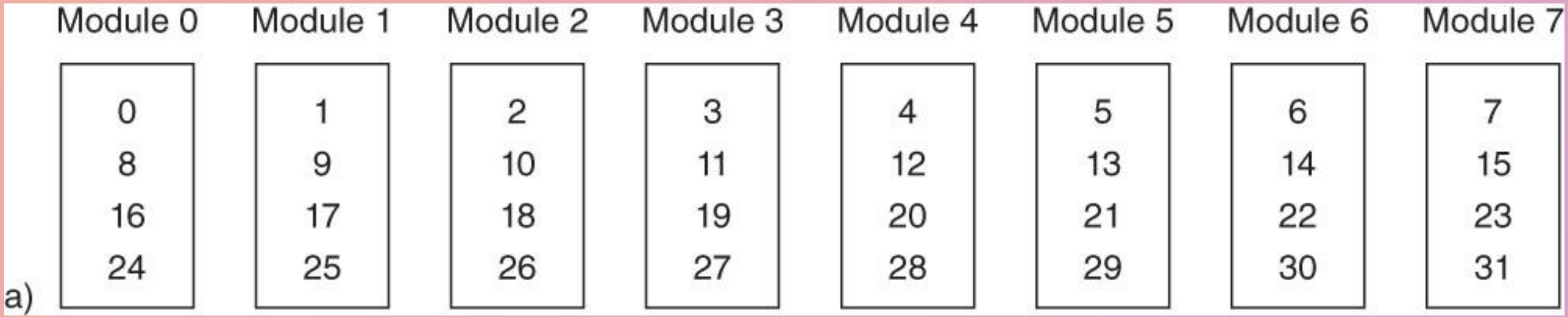
- Each memory chip can store a certain amount of information
  - However, architects decide how memory is spread across these chips
  - For instance, do we want to have an entire byte on a single chip, or spread a byte across 2 or more chips?
  - The figure to the right shows that each chip stores 2 Kbytes and with 16 rows and 2 columns of chips, we can store 64 Kbytes
    - Here, a word (16 bits) is stored in two chips in the same row



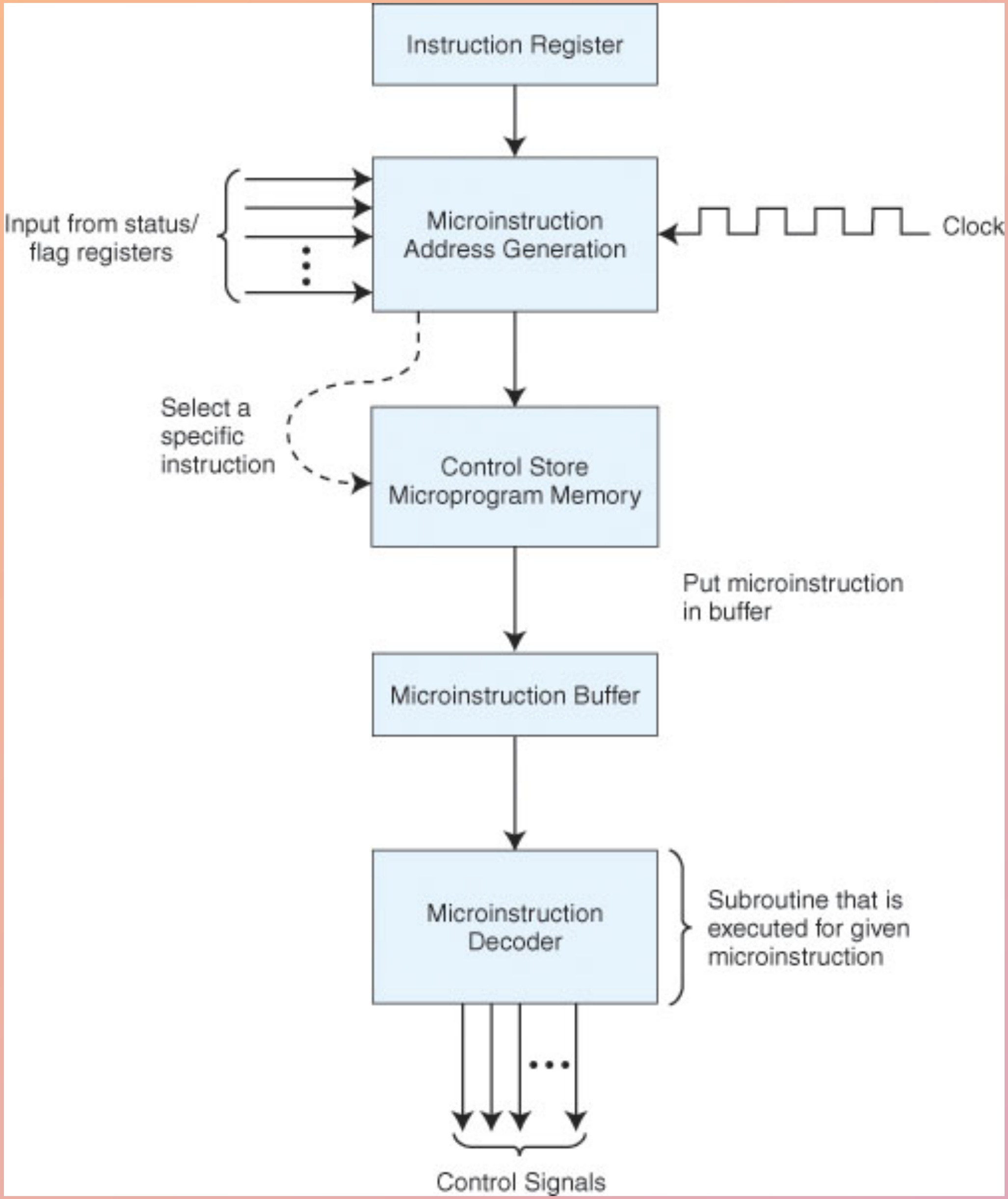
Interleaving Memory



- Using high-order interleave, the address is broken into the chip followed by the location on the chip, giving a layout as shown above
  - The advantage of high-order interleave is that two different devices, working on two different areas of memory, can perform their memory accesses simultaneously
  - e.g., one device accesses address 5 and another accesses 31
- In low-order interleave, the address is broken up by location on the chip first, and the chip number last
  - Consecutive memory locations are on consecutive chips
  - The advantage of lower-order interleave is that several consecutive memory accesses can be performed simultaneously
  - For instance, fetching 4 consecutive instructions at one time



Microprogrammed Control Unit


Instruction Register
Input from status/ flag registers
Microinstruction Address Generation
Clock
Select a specific instruction
Control Store Microprogram Memory
Put microinstruction in buffer
Microinstruction Buffer
Microinstruction Decoder
Subroutine that is executed for given microinstruction
Control Signals

The control store is a ROM that stores all of the microprograms

One microprogram per fetch-execute stage, and one per instruction in the instruction set

Receive an instruction in IR

Start the microprogram, generating the address, fetching the instruction from the ROM and moving it to the microinstruction buffer to be decoded an executed

This process is much more time consuming than the hardwired unit, but is easier to implement and more flexible

CISC vs. RISC

- Complex (CISC)
  - Microprogrammed control unit
  - Large number of instructions (200-500)
  - Instructions can do more than 1 thing (that is, an instruction could carry out 2 or more actions)
  - Many addressing modes
  - Instructions vary in length and format
  - This was the typical form of architecture until the mid 1980s, RISC has become more popular since then although most architectures remain CISC

- Reduced (RISC)
  - Hardwired control unit
  - Instruction set limited (perhaps 80-100 instructions)
  - Instructions rely mostly on registers, memory accessed only on loads and stores
  - Few addressing modes
  - Instruction lengths fixed (usually 32 bits long)
  - Easy to pipeline for great speedup in performance