Org: I/O and Memory

Alex Chi

Update: March 17, 2020

Contents

1 Characteristics of Memory

1.1 Location

- CPU: registers
- Internal: cache, main memory
- External: disk, tape, DVD

1.2 Capacity

- Word size (of disk): natural unit of organization
- Example: 2M 8-bit, 16M 1-bit, same capacity, different organization

1.3 Unit of Transfer

- Internal memory: usually a word (data bus)
- External memory: usually a block (much larger than word)

1.4 Addressable unit

- smallest location being uniquely addressed
- normally a byte for internal memory
- on disks, cluster

1.5 Access Methods

1.5.1 Sequential

- from beginning and read through in order
- access time depends on location

1.5.2 Direct

- individual blocks have unique address
- e.g. disk
- clusters: 4 or more sectors

1.5.3 Random Access

- access time independent of location or previous access
- e.g. ROM, RAM

1.5.4 Associative

- data stored based on portion of its contents other than its address
- access time is independent of location or previous access
- e.g. Cache

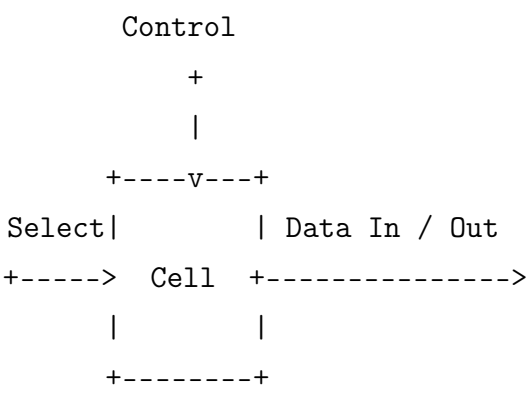
1.6 Performance

- Access time
- Memory cycle time
- Transfer rate

1.7 Physical Types

- Semiconductor (RAM)
- Magnetic (Disk & Tape)
- Optical (CD & DVD)
- Others

1.7.1 Semiconductor Memory



- Read/Write
- Volatile (power supply)

- Temporary storage
- Static or Dynamic (SRAM / DRAM)

1.7.2 DRAM

- Bits stored as **charge** in capacitors
- Need refreshing when powered
- Slower
- Main memory
- [Figure: circuit] in slides

1.7.3 SRAM

- [Figure: circuit] Flip-flop module
- bits stored as on/off switches
- won't leak
- more expensive
- for use of cache
- faster
- complex
- for cache

1.7.4 ROM

- permanent storage
- nonvolatile

1. Applications
   - library subroutines
   - BIOS
   - function tables
2. Types
   - written during manufacturing
   - programmable once (PROM)
   - read "mostly"
     - Erasable Programmable (EPROM) (ultraviolet)
     - Electrically Erasable (EEPROM) (slower to write)
     - Flash Memory

1.8 Memory Hierarchy

- a balance between speed and cost
- register, L1 L2 L3 Cache, internal memory, disk cache, disk
- is it possible to build a computer use only static RAM?
  - speed is fast
  - but expense is high

1.9 Chip Organization

- how to construct chip from memory cells?
- physical arrangement of bits into words ("word" is not system word)

1.9.1 Organization in Detail

- A 16 Mbit chip can be organized as 1M of 16-bit words
- A 16 Mbit chip can be organized as $2048 \times 2048 \times 4$bit array
  - 1-d array: 20 pins for address, 16 pins for data, 1 pin for control (37)
  - 2-d array: 11 pins for row, 11 pins for col, 4 pins for data, 1 pin for control (27)
  - time multiplex: a tick for saving row address, a tick for saving col address
  - control signals: RAS, CAS (row/col address select), WE (write enable, low effective), CE/OE (Chip Enable)
  - refresh row by row
  - an extra pin provides 4x space

1.9.2 Chip Packaging

- refer to figure in slides
- EPROM: no R/W pin
- DRAM: 2-d array, arrow means write direction

1.9.3 Module Organization

- my personal advice is to learn this part with virtual memory.

1. Word Extension
   - higher address is used for selecting which chip to use

        - for example, 3-8 decoder, 2-4 decoder, n - 2^n decoder, which turns a number n into n bool.
    - lower address is used for selecting a particular address on chip
    - output is from one of the chips
    - refer to slides
2. Bit Extension
    - an address is sent to all chips
    - output is combined from all of the chips
    - refer to slides
3. Word and Bit Extension
    - do bit extension first
    - then word extension
    - 256 × 8-bit ⟶ 256 × 32-bit ⟶ 2M × 32-bit
    - 8086 support unaligned access

2 Input/Output Device

2.1 I/O Challenges

- wide variety of peripherals
    - different operation logic
    - speak different language
    - slower than CPU and RAM
- we need I/O modules (ports)

2.2 I/O Module Function

- control & timing
- CPU communication
- device communication
- data buffering
- error detection

2.3 I/O Steps

- check/respond device status

- if device ready, request/get data

2.4 I/O Module

- external device interface logic: operate on device
- data is used by
  - I/O port
  - device
  - therefore, data and status/control register connect to data lines

2.5 I/O Module Design Decision

- hide or reveal device properties
- support multiple or single device
- control device functions or leave for CPU

2.6 I/O Techniques

2.6.1 Programmed I/O

- CPU executes a sequence of I/O operation
  - check status
  - r/w commands
  - transfer data
- CPU waits for I/O module to complete operation
- detail
  - CPU: request I/O
  - I/O: perform operation
  - I/O: set status bits
  - CPU: check status bits periodically
  - CPU: wait, come back later
- I/O commands
  - CPU issue address (identify module)
  - CPU issue commands (control, test, r/w)
- data transfer is very like memory r/w
- pros: simple

- cons: waste of CPU time

2.6.2 Interrupt driven I/O

- overcome CPU waiting
- no repeated CPU checking
- I/O module interrupts when ready
- CPU deals with this event
- details
  - CPU requests I/O operation
  - I/O performs operation
  - I/O interrupts CPU
  - CPU deals with event
- CPU Viewpoint
  - issue read command
  - do other work
  - check for interrupt **at the end of each cycle**
  - if interrupted: save context, process interrupt, recover from saved context
- drawbacks: require CPU involvement

2.6.3 Handling Interrupt

- device issue interrupt
- processor finishes execution of current instruction
- processor signals acknowledgment of interrupt
- push PSW and PC to control stack
- process loads new PC value
- run interrupt handler:
  - save process state (all registers)
  - process interrupt
  - restore process state information
  - restore old PSW and PC
- example: RISC-V PC saves in epc, and you should refresh mie, mstatus

2.6.4 Design Issues

- how can CPU know which module issued interrupt?

- how to locate corresponding handler?
- how to deal with multiple interrupts?

2.6.5 Identifying Interrupting Module

- connect a dedicated line (ARM Cortex), limit number of devices
- software poll
  - in general handler, poll every device
- daisy chain / hardware poll
  - interrupt acknowledge sent down a chain
  - if not received by me, then send to next device
  - priority is fixed
- bus master
  - module claim the bus before interrupt
  - PCI / SCSI
- interrupt controller
  - PLIC on RISC-V
  - 8259

2.6.6 The Process of Interrupt

- INT request
- INT ack
- save PC, save PSW (to memory)
- load PC (jump to interrupt handler)
- in handler, save program context (those used by handler)
- restore program context
- resume user program (hardware)

2.6.7 Handler Program

1. general handler program
   - feasible
   - low performance
   - low flexibility
2. interrupt vectors
   - store anywhere

- flexible
- need to specify IVT (interrupt vector table)
- and need to know index
- process
  - INT req
  - INT ack
  - I/O write INT id to data bus

2.6.8 Handle Multiple Interrupts

- set priorities for interrupts
- nesting of interrupts (low priority can be interrupted)

2.6.9 Direct Memory Access (DMA)

- require additional module on bus
- CPU tells DMA controller
  - R/W
  - device address
  - starting address of memory block
  - amount of data to be transferred
- CPU goes on, DMA does work
- DMA interrupts CPU
- example: virt-io driver in RISC-V, which uses "descriptor" to configure
- in an instruction cycle, the process may be suspended due to DMA operation (cycle stealing)
- single bus, detached DMA
  - use bus twice (I/O to DMA, DMA to memory)
- I/O attached to DMA controller
  - single bus, integrated DMA
  - each transfer uses bus once
- separate I/O bus
  - DMA controller connects to system bus and I/O bus

2.6.10 Which one is better?

- it's up to your usage scenario

2.6.11 Q&A

1. How does a device know which address space it lies in?
   Each device is given a unique identifier.
   A 3-8 decoder-like thing for mux.
2. What's relation between I/O port and USB controller?
   They're a kind of I/O port.