Topic: Cache Memory

Characteristics of Memory "Location wrt Processor"

- Inside CPU – temporary memory or registers
- Inside processor – L1 cache
- Motherboard – main memory and L2 cache
- Main memory – DRAM and L3 cache
- External – peripherals such as disk, tape, and networked memory devices

Characteristics of Memory “Capacity – Word Size”

- The natural data size for a processor.
- A 32-bit processor has a 32-bit word.
- Typically based on processor's data bus width (i.e., the width of an integer or an instruction)
- Varying widths can be obtained by putting memory chips in parallel with same address lines

Characteristics of Memory “Capacity – Addressable Units”

- Varies based on the system's ability to allow addressing at byte level etc.
- Typically smallest location which can be uniquely addressed
- At mother board level, this is the word
- It is a cluster on disks
- Addressable units (N) equals 2 raised to the power of the number of bits in the address bus

Characteristics of Memory “Unit of transfer”

- The number of bits read out of or written into memory at a time.
- Internal – Usually governed by data bus width, i.e., a word
- External – Usually a block which is much larger than a word

Characteristics of Memory “Access method”

- Based on the hardware implementation of the storage device
- Four types
  - Sequential
  - Direct
  - Random
  - Associative

Sequential Access Method

- Start at the beginning and read through in order
- Access time depends on location of data and previous location
- Example: tape

Direct Access Method

- Individual blocks have unique address
- Access is by jumping to vicinity then performing a sequential search
- Access time depends on location of data within "block" and previous location
- Example: hard disk

Random Access Method

- Individual addresses identify locations exactly
- Access time is consistent across all locations and is independent previous access
- Example: RAM

Associative Access Method

- Addressing information must be stored with data in a general data location
- A specific data element is located by a comparing desired address with address portion of stored elements
- Access time is independent of location or previous access
- Example: cache

Performance – Access Time

- Time between "requesting" data and getting it
- RAM
  - Time between putting address on bus and getting data.
  - It's predictable.
- Other types, Sequential, Direct, Associative
  - Time it takes to position the read-write mechanism at the desired location.
  - Not predictable.

Performance – Memory Cycle time

- Primarily a RAM phenomenon
- Adds "recovery" time to cycle allowing for transients to dissipate so that next access is reliable.
- Cycle time is access + recovery

Performance – Transfer Rate

- Rate at which data can be moved
- RAM – Predictable; equals 1/(cycle time)
- Non-RAM – Not predictable; equals

$T_N = T_A + (N/R)$

where

- T_N = Average time to read or write N bits
- T_A = Average access time
- N = Number of bits
- R = Transfer rate in bits per second

Physical Types

- Semiconductor – RAM
- Magnetic – Disk & Tape
- Optical – CD & DVD
- Others
  - Bubble (old) – memory that made a "bubble" of charge in an opposite direction to that of the thin magnetic material that on which it was mounted
  - Hologram (new) – much like the hologram on your credit card, laser beams are used to store computer-generated data in three dimensions. (10 times faster with 12 times the density)

Physical Characteristics

- Decay
  - Power loss
  - Degradation over time
- Volatility – RAM vs. Flash
- Erasable – RAM vs. ROM
- Power consumption – More specific to laptops, PDAs, and embedded systems

Organization

- Physical arrangement of bits into words
- Not always obvious
- Non-sequential arrangements may be due to speed or reliability benefits, e.g. interleaved

Memory Hierarchy

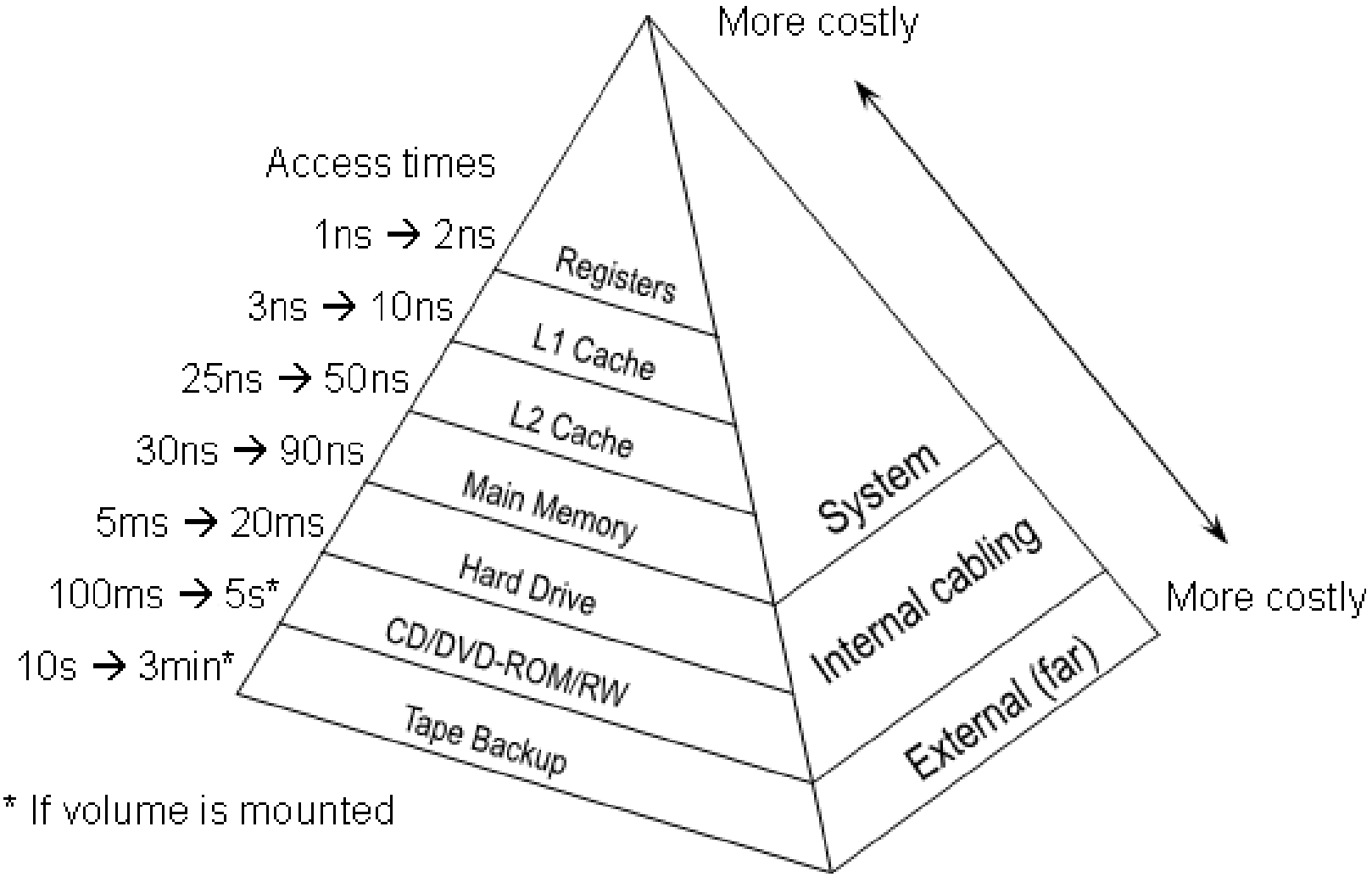
- Trade-offs among three key characteristics
  - Amount – Software will ALWAYS fill available memory
  - Speed – Memory should be able to keep up with the processor
  - Cost – Whatever the market will bear
- Balance these three characteristics with a memory hierarchy
- Analogy –
  Refrigerator & cupboard (fast access – lowest variety)
  freezer & pantry (slower access – better variety)
  grocery store (slowest access – greatest variety)

Memory Hierarch (continued)

Implementation – Going down the hierarchy has the following results:

- Decreasing cost per bit (cheaper)
- Increasing capacity (larger)
- Increasing access time (slower)
- **KEY** – Decreasing frequency of access of the memory by the processor

Memory Hierarch (continued)



Source: Null, Linda and Lobur, Julia (2003). *Computer Organization and Architecture* (p. 236). Sudbury, MA: Jones and Bartlett Publishers.

Mechanics of Technology

- The basic mechanics of creating memory directly affect the first three characteristics of the hierarchy:
  - Decreasing cost per bit
  - Increasing capacity
  - Increasing access time
- The fourth characteristic is met because of a principle known as **locality of reference**

Locality of Reference

Due to the nature of programming, instructions and data tend to cluster together (loops, subroutines, and data structures)

- Over a long period of time, clusters will change
- Over a short period, clusters will tend to be the same

Breaking Memory into Levels

- Assume a hypothetical system has two levels of memory
  - Level 2 should contain all instructions and data
  - Level 1 doesn't have room for everything, so when a new cluster is required, the cluster it replaces must be sent back to the level 2
- These principles can be applied to much more than just two levels
- If performance is based on amount of memory rather than speed, lower levels can be used to simulate larger sizes for higher levels, e.g., virtual memory

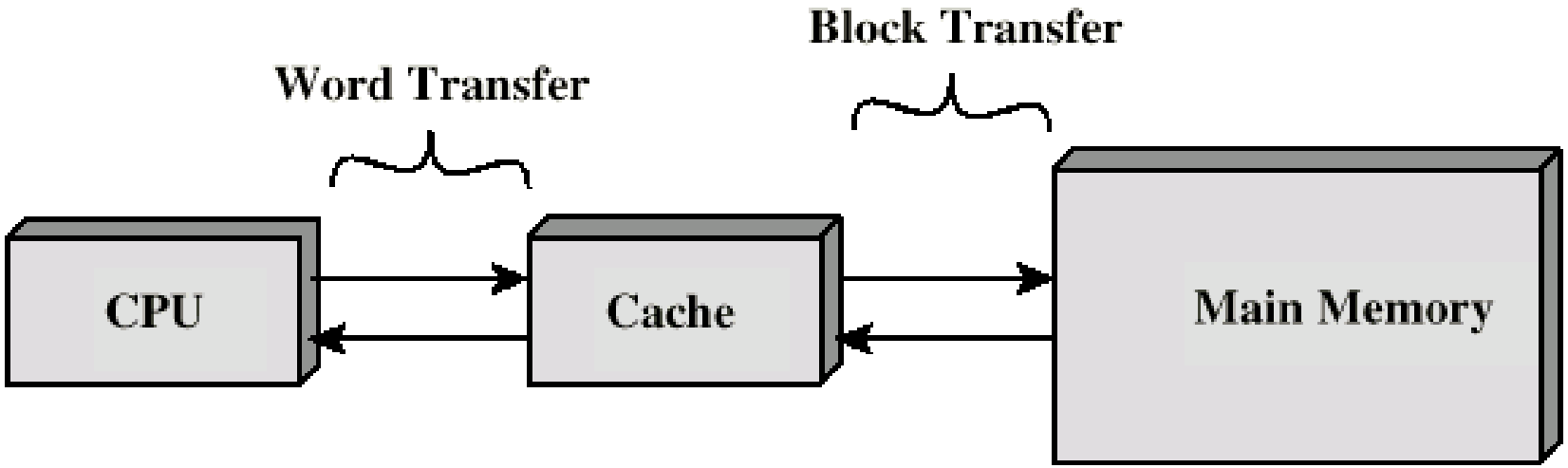
Hierarchy List

- Registers – volatile
- L1 Cache – volatile
- L2 Cache – volatile
- CDRAM (main memory) cache – volatile
- Main memory – volatile
- Disk cache – volatile
- Disk – non-volatile
- Optical – non-volatile
- Tape – non-volatile

Cache

- What is it? A cache is a small amount of fast memory
- What makes small fast?
  - Simpler decoding logic
  - More expensive SRAM technology
  - Close proximity to processor – Cache sits between normal main memory and CPU or it may be located on CPU chip or module

Cache (continued)



Cache operation – overview

- CPU requests contents of memory location
- Check cache for this data
- If present, get from cache (fast)
- If not present, one of two things happens:
  - read required block from main memory to cache then deliver from cache to CPU (cache physically between CPU and bus)
  - read required block from main memory to cache and simultaneously deliver to CPU (CPU and cache both receive data from the same data bus buffer)

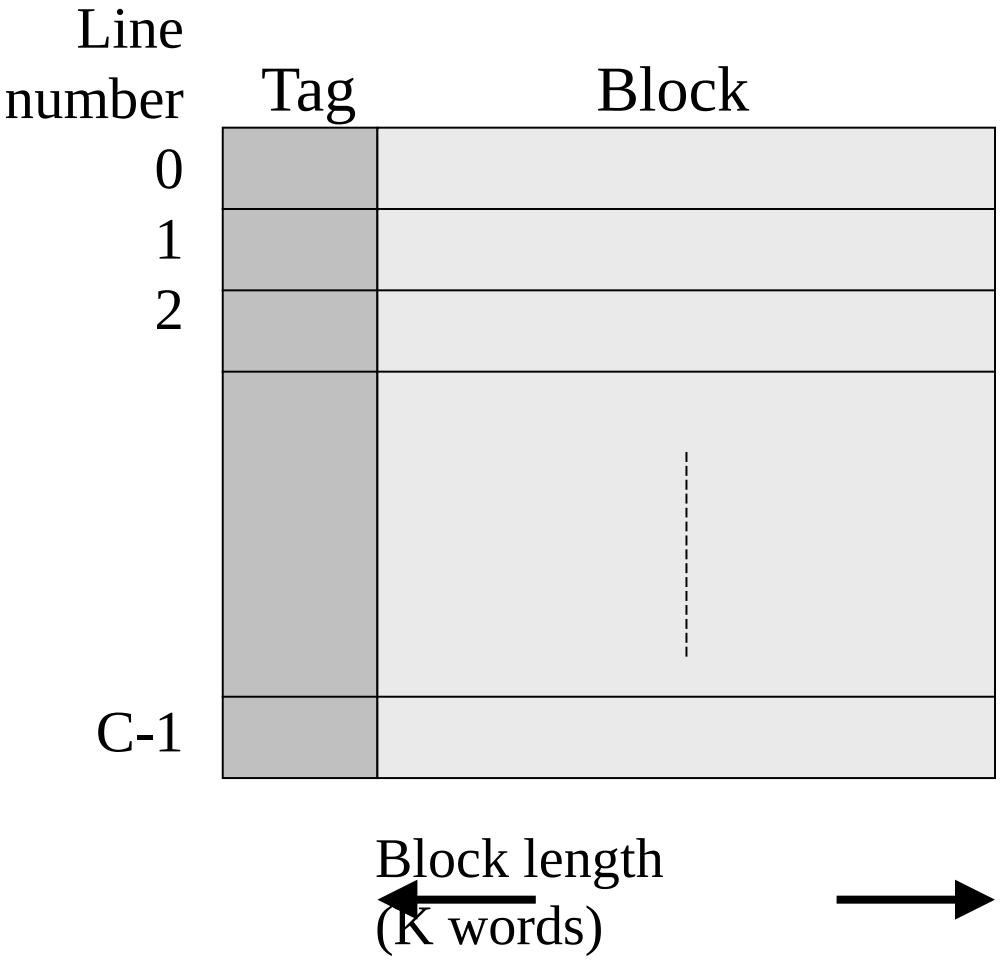
Going Deeper with Principle of Locality

- Cache "misses" are unavoidable, i.e., every piece of data and code thing must be loaded at least once
- What does a processor do during a miss? It waits for the data to be loaded.
- Power consumption varies linearly with clock speed and the square of the voltage.
- Adjusting clock speed and voltage of processor has the potential to produce cubic (cubed root) power reductions (http://www.visc.vt.edu/~mhsiao/papers/pacs00ch.pdf)
- Identify places in in-class exercise where this might happen.

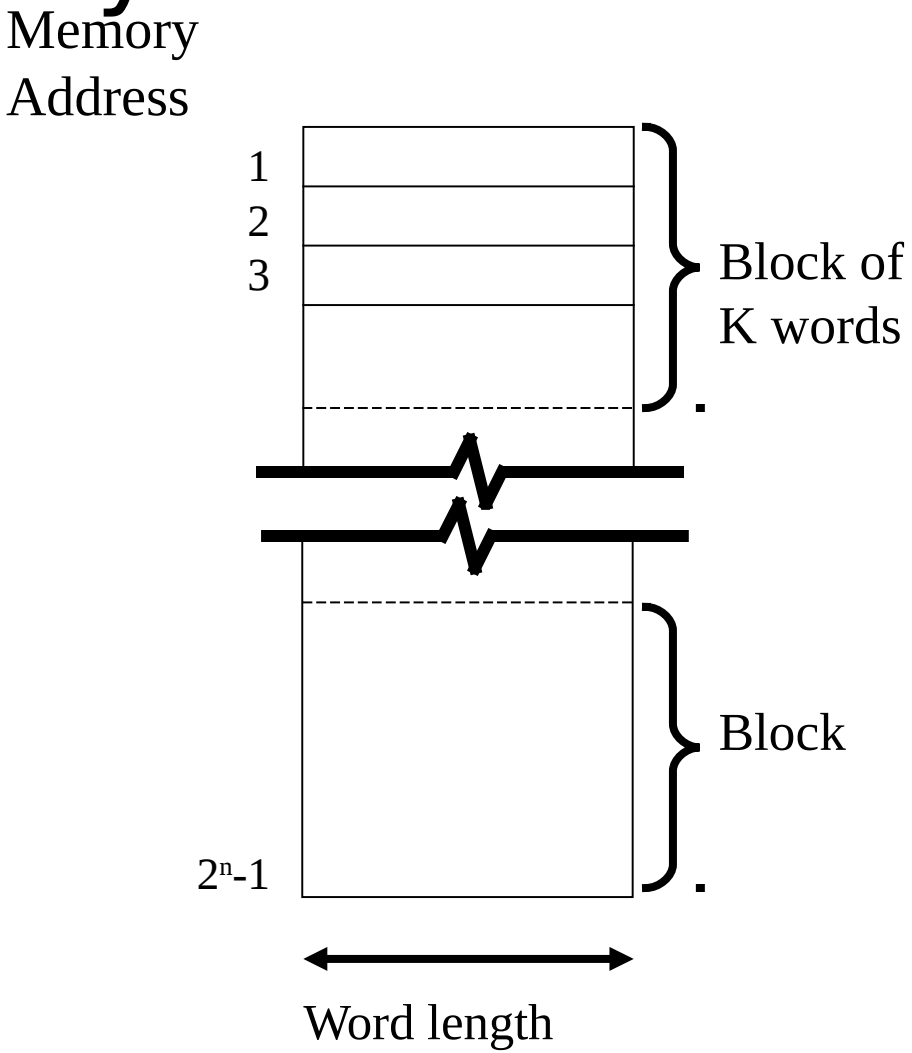
Cache Structure

- Cache includes tags to identify the address of the block of main memory contained in a line of the cache
- Each word in main memory has a unique n-bit address
- There are $M=2^n/K$ block of K words in main memory
- Cache contains C lines of K words each plus a tag uniquely identifying the block of K words

Cache Structure (continued)



Memory Divided into Blocks



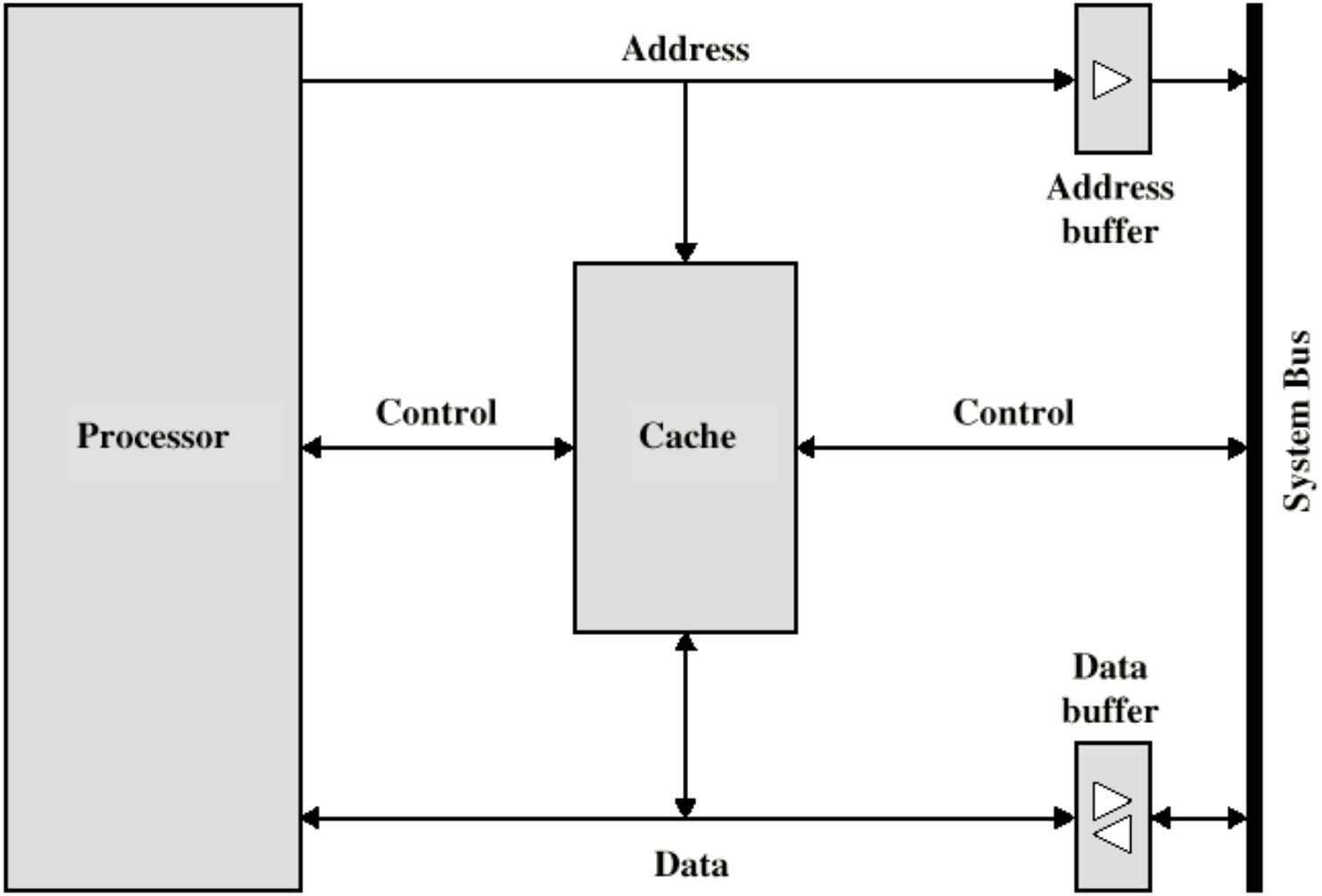
Cache Design

- Size
- Mapping Function
- Replacement Algorithm
- Write Policy
- Block Size
- Number of Caches

Cache size

- Cost – More cache is expensive
- Speed
  - More cache is faster (up to a point)
  - Larger decoding circuits slow up a cache
  - Algorithm is needed for mapping main memory addresses to lines in the cache. This takes more time than just a direct RAM

Typical Cache Organization



Mapping Functions

- A mapping function is the method used to locate a memory address within a cache
- It is used when copying a block from main memory to the cache and it is used again when trying to retrieve data from the cache
- There are three kinds of mapping functions
  - Direct
  - Associative
  - Set Associative

Cache Example

These notes use an example of a cache to illustrate each of the mapping functions. The characteristics of the cache used are:

- Size: 64 kByte
- Block size: 4 bytes – i.e. the cache has 16k (2^{14}) lines of 4 bytes
- Address bus: 24-bit– i.e., 16M bytes main memory divided into 4M 4 byte blocks

Direct Mapping Traits

- Each block of main memory maps to only one cache line – i.e. if a block is in cache, it will always be found in the same place
- Line number is calculated using the following function

$$i = j \text{ modulo } m$$

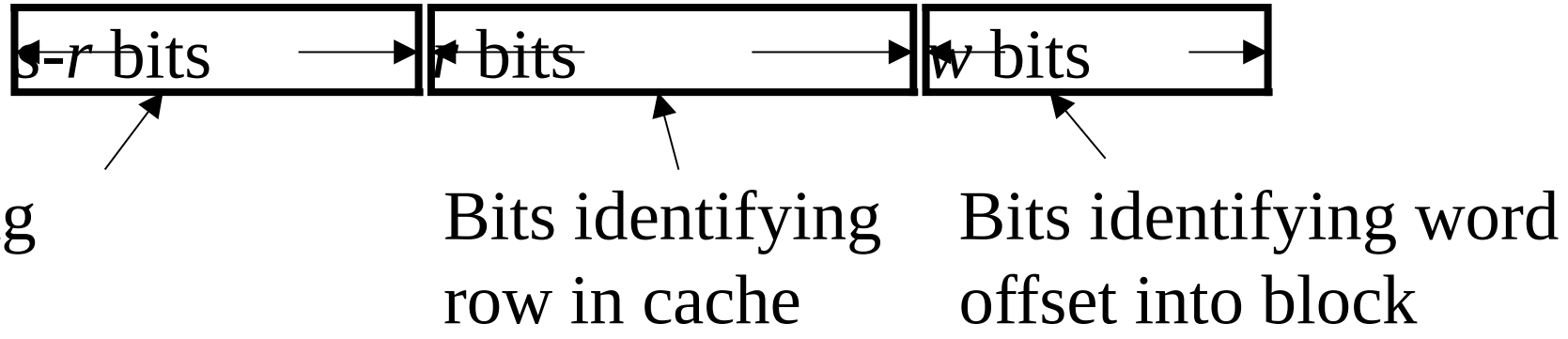
where

i = cache line number
j = main memory block number
m = number of lines in the cache

Direct Mapping Address Structure

Each main memory address can by divided into three fields

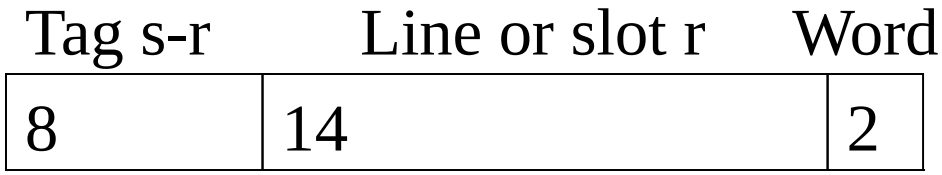
- Least Significant w bits identify unique word within a block
- Remaining bits (s) specify which block in memory. These are divided into two fields
  - Least significant r bits of these s bits identifies which line in the cache
  - Most significant s-r bits uniquely identifies the block within a line of the cache



Direct Mapping Address Structure (continued)

- Why are the r-bits used to identify which line in cache?
- More likely to have unique r bits than s-r bits based on principle of **locality of reference**

Direct Mapping Address Structure Example

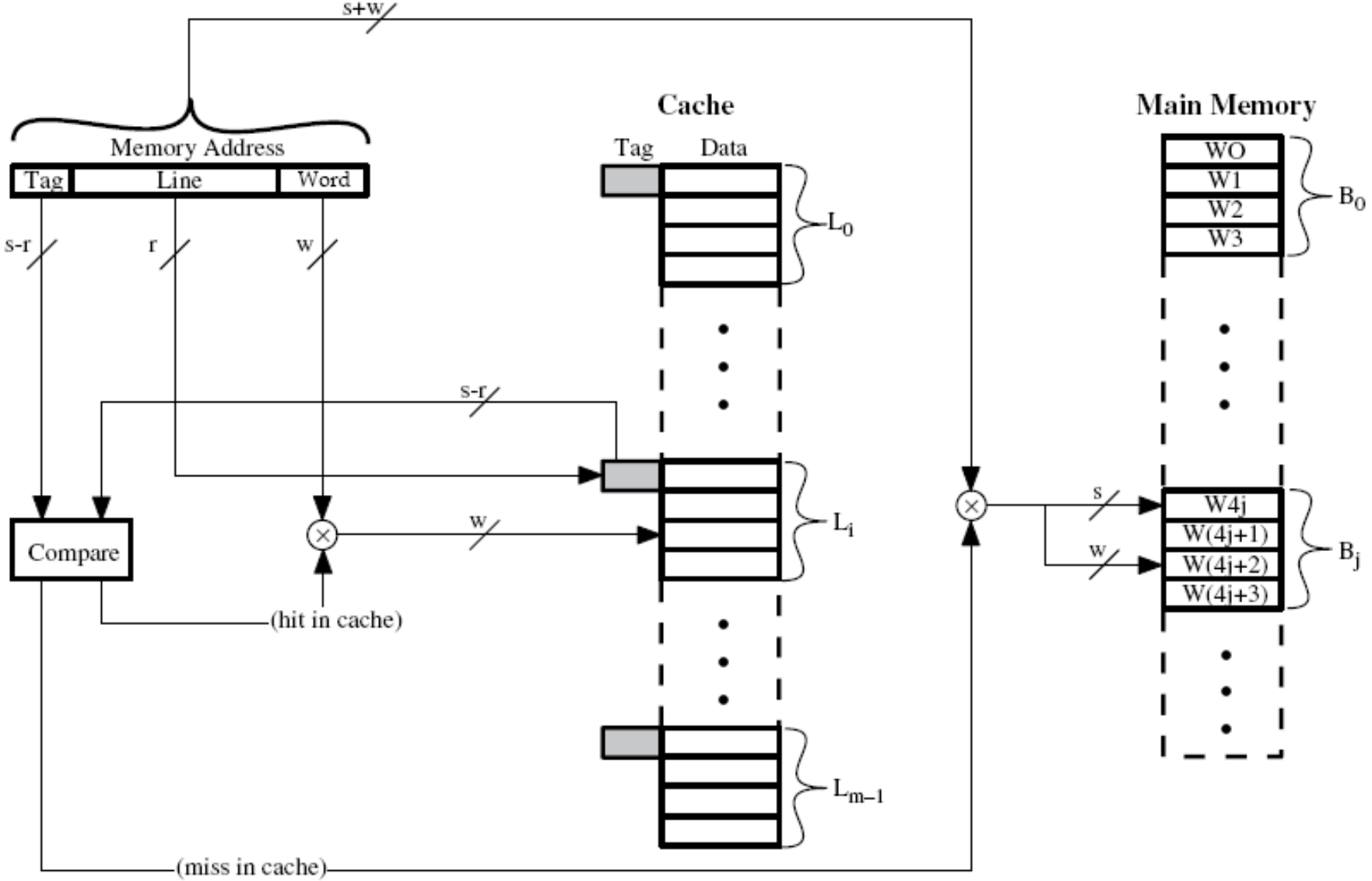


- 24 bit address
- 2 bit word identifier (4 byte block)
- 22 bit block identifier
- 8 bit tag (=22–14)
- 14 bit slot or line
- No two blocks in the same line have the same tag
- Check contents of cache by finding line and comparing tag

Direct Mapping Cache Line Table

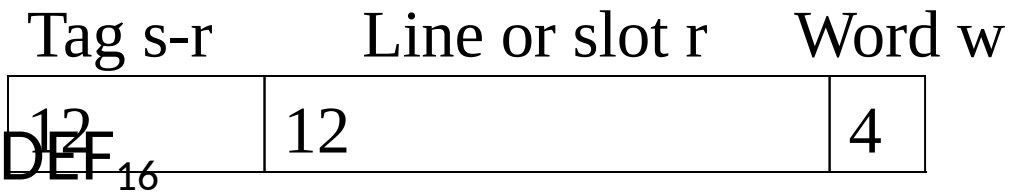
Cache line	Main Memory blocks held
0	0, m, 2m, 3m…2^s–m
1	1, m+1, 2m+1…2^s–m+1
m–1	m–1, 2m–1, 3m–1…2^s–1

Direct Mapping Cache Organization



Direct Mapping Examples

What cache line number will the following addresses be stored to, and what will the minimum address and the maximum address of each block they are in be if we have a cache with 4K lines of 16 words to a block in a 256 Meg memory space (28-bit address)?



a.) $9ABCDEF_{16}$

b.) 1234567_{16}

More Direct Mapping Examples

Assume that a portion of the tags in the cache in our example looks like the table below. Which of the following addresses are contained in the cache?

a.) $438EE8_{16}$ b.) $F18EFF_{16}$ c.) $6B8EF3_{16}$ d.) $AD8EF3_{16}$

Tag (binary)	Line number (binary)	Addresses wi/ block			
		00	01	10	11
0101 0011	1000 1110 1110 10				
1110 1101	1000 1110 1110 11				
1010 1101	1000 1110 1111 00				
0110 1011	1000 1110 1111 01				
1011 0101	1000 1110 1111 10				
1111 0001	1000 1110 1111 11				

Direct Mapping Summary

- Address length = (s + w) bits
- Number of addressable units = 2^{s+w} words or bytes
- Block size = line width = 2^{w} words or bytes
- Number of blocks in main memory = $2^{s+ w}/2^{w} = 2^{s}$
- Number of lines in cache = m = 2^{r}
- Size of tag = (s – r) bits

Direct Mapping pros & cons

- Simple
- Inexpensive
- Fixed location for given block –
  If a program accesses 2 blocks that map to the same line repeatedly, cache misses are very high (thrashing)

Associative Mapping Traits

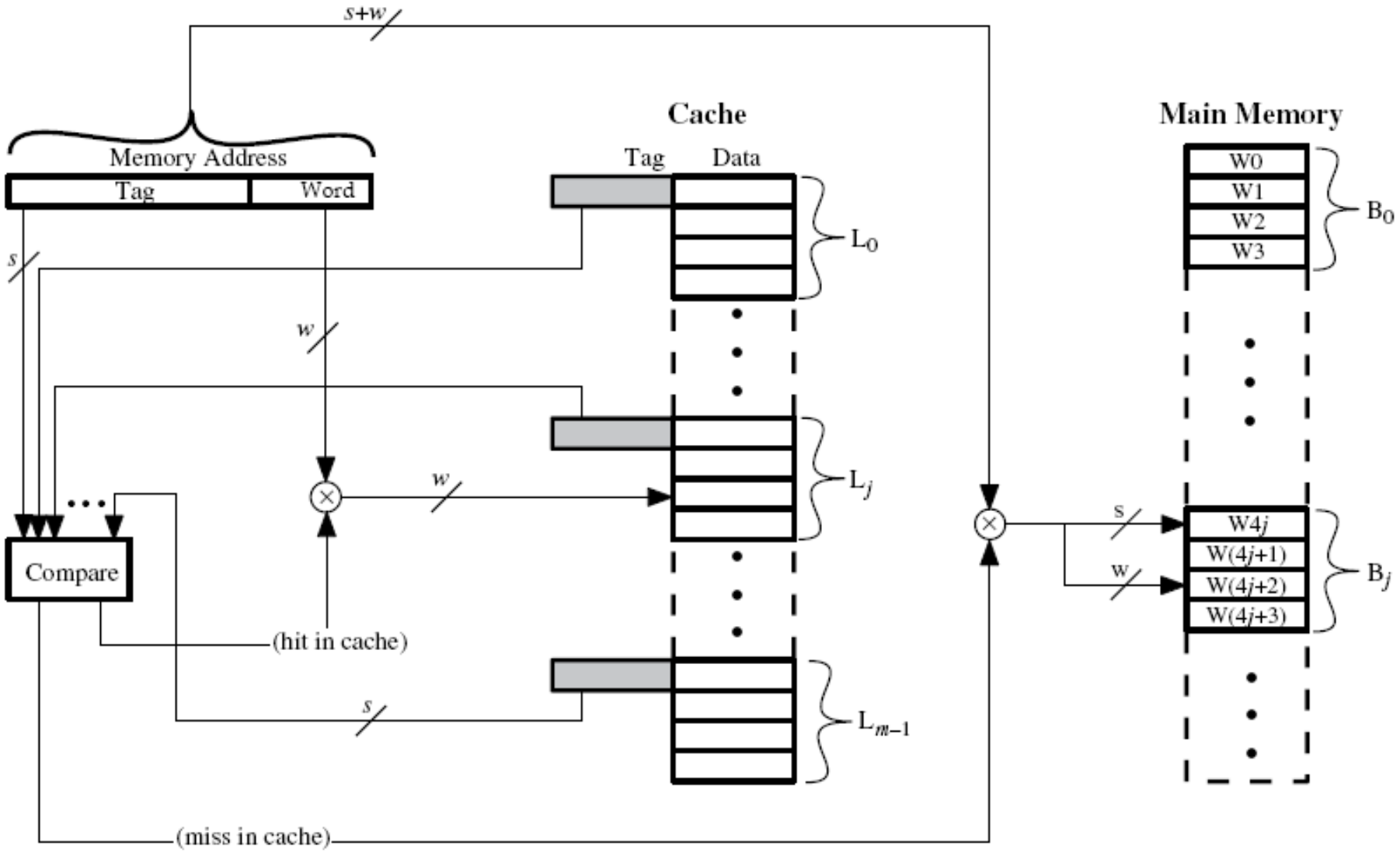
- A main memory block can load into any line of cache
- Memory address is interpreted as:
  - Least significant w bits = word position within block
  - Most significant s bits = tag used to identify which block is stored in a particular line of cache
- Every line's tag must be examined for a match
- Cache searching gets expensive and slower

Associative Mapping Address Structure Example

Tag – s bits (22 in example)	Word – w bits (2 in ex.)

- 22 bit tag stored with each 32 bit block of data
- Compare tag field with tag entry in cache to check for hit
- Least significant 2 bits of address identify which of the four 8 bit words is required from 32 bit data block

Fully Associative Cache Organization



Fully Associative Mapping Example

Assume that a portion of the tags in the cache in our example looks like the table below. Which of the following addresses are contained in the cache?

a.) $438EE8_{16}$ b.) $F18EFF_{16}$ c.) $6B8EF3_{16}$ d.) $AD8EF3_{16}$

Tag (binary)	Addresses wi/ block			
	00	01	10	11
0101 0011 1000 1110 1110 10				
1110 1101 1100 1001 1011 01				
1010 1101 1000 1110 1111 00				
0110 1011 1000 1110 1111 11				
1011 0101 0101 1001 0010 00				
1111 0001 1000 1110 1111 11				

Associative Mapping Summary

- Address length = (s + w) bits
- Number of addressable units = 2^{s+w} words or bytes
- Block size = line size = 2^{w} words or bytes
- Number of blocks in main memory = $2^{s+ w}/2^{w} = 2^{s}$
- Number of lines in cache = undetermined
- Size of tag = s bits

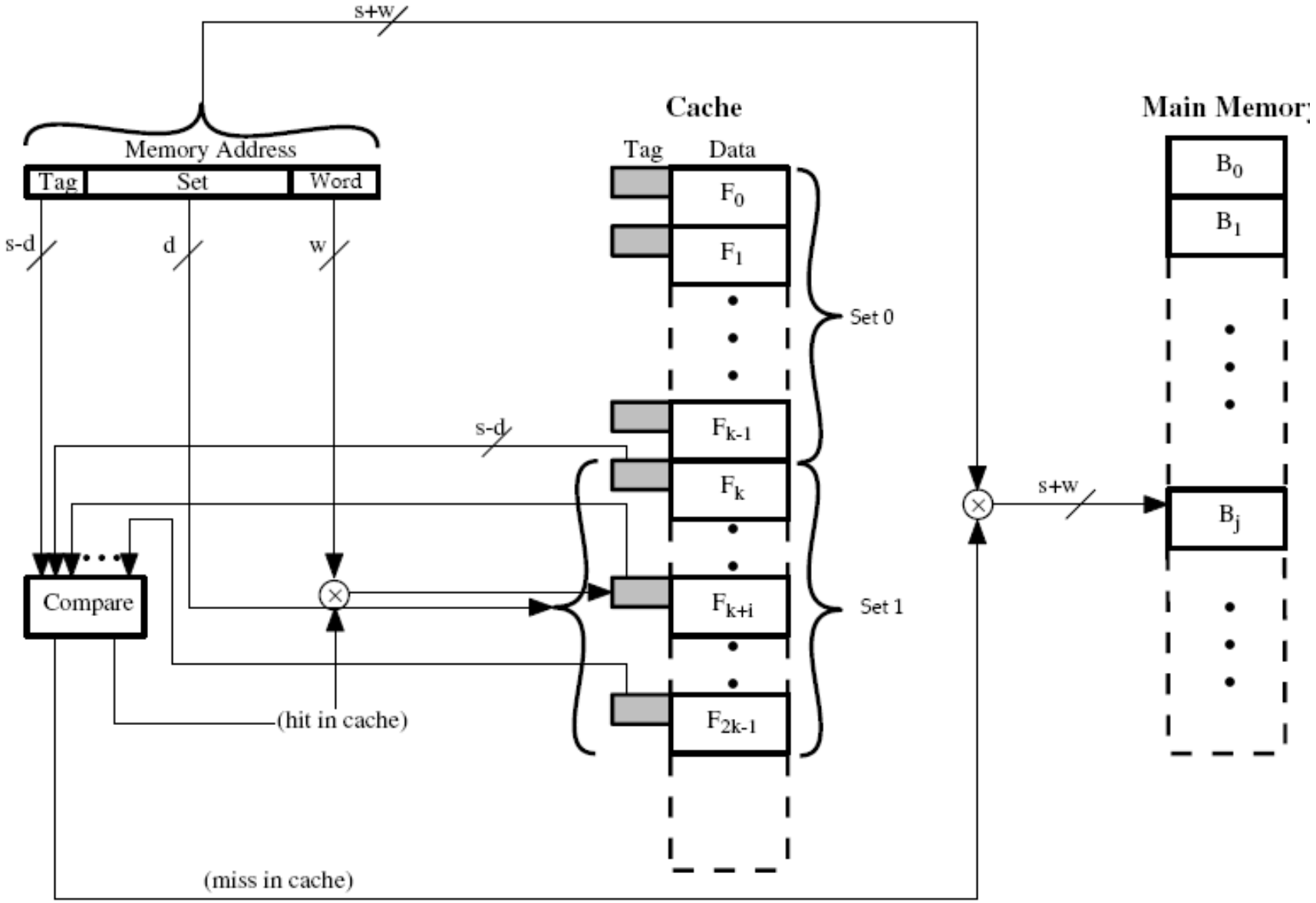
Set Associative Mapping Traits

- Address length is $s + w$ bits
- Cache is divided into a number of sets, $v = 2^d$
- k blocks/lines can be contained within each set
- k lines in a cache is called a k-way set associative mapping
- Number of lines in a cache = $v \bullet k = k \bullet 2^d$
- Size of tag = $(s-d)$ bits

Set Associative Mapping Traits (continued)

- Hybrid of Direct and Associative
  k = 1, this is basically direct mapping
  v = 1, this is associative mapping
- Each set contains a number of lines, basically the number of lines divided by the number of sets
- A given block maps to any line within its specified set – e.g. Block B can be in any line of set i.
- 2 lines per set is the most common organization.
  - Called 2 way associative mapping
  - A given block can be in one of 2 lines in only one specific set
  - Significant improvement over direct mapping

K-Way Set Associative Cache Organization



How does this affect our example?

- Let's go to two-way set associative mapping
- Divides the 16K lines into 8K sets
- This requires a 13 bit set number
- With 2 word bits, this leaves 9 bits for the tag
- Blocks beginning with the addresses 000000_{16}, 008000_{16}, 010000_{16}, 018000_{16}, 020000_{16}, 028000_{16}, etc. map to the same set, Set 0.
- Blocks beginning with the addresses 00000416, 008004_{16}, 010004_{16}, 018004_{16}, 020004_{16}, 028004_{16}, etc. map to the same set, Set 1.

Set Associative Mapping Address Structure

Tag 9 bits	Set 13 bits	Word 2 bits

- Note that there is one more bit in the tag than for this same example using direct mapping.
- Therefore, it is 2-way set associative
- Use set field to determine cache set to look in
- Compare tag field to see if we have a hit

Set Associative Mapping Example

For each of the following addresses, answer the following questions based on a 2-way set associative cache with 4K lines, each line containing 16 words, with the main memory of size 256 Meg memory space (28-bit address):

- What cache set number will the block be stored to?
- What will their tag be?
- What will the minimum address and the maximum address of each block they are in be?

1. $9ABCDEF_{16}$
2. 1234567_{16}

Set Associative Mapping Summary

- Address length = (s + w) bits
- Number of addressable units = 2^{s+w} words or bytes
- Block size = line size = 2^{w} words or bytes
- Number of blocks in main memory = $2^{s+ w}/2^{w} = 2^{s}$
- Number of lines in set = k
- Number of sets = v = 2^{d}
- Number of lines in cache = kv = k * 2^{d}
- Size of tag = (s – d) bits

Replacement Algorithms

- There must be a method for selecting which line in the cache is going to be replaced when there's no room for a new line
- Hardware implemented algorithm (speed)
- Direct mapping
  - There is no need for a replacement algorithm with direct mapping
  - Each block only maps to one line
  - Replace that line

Associative & Set Associative Replacement Algorithms

- Least Recently used (LRU)
  - Replace the block that hasn't been touched in the longest period of time
  - Two way set associative simply uses a USE bit. When one block is referenced, its USE bit is set while its partner in the set is cleared
- First in first out (FIFO) – replace block that has been in cache longest

Associative & Set Associative Replacement Algorithms (continued)

- Least frequently used (LFU) – replace block which has had fewest hits
- Random – only slightly lower performance than use-based algorithms LRU, FIFO, and LFU

Writing to Cache

- Must not overwrite a cache block unless main memory is up to date
- Two main problems:
  - If cache is written to, main memory is invalid or if main memory is written to, cache is invalid – Can occur if I/O can address main memory directly
  - Multiple CPUs may have individual caches; once one cache is written to, all caches are invalid

Write through

- All writes go to main memory as well as cache
- Multiple CPUs can monitor main memory traffic to keep local (to CPU) cache up to date
- Lots of traffic
- Slows down writes

Write back

- Updates initially made in cache only
- Update bit for cache slot is set when update occurs
- If block is to be replaced, write to main memory only if update bit is set
- Other caches get out of sync
- I/O must access main memory through cache
- Research shows that 15% of memory references are writes

Multiple Processors/Multiple Caches

- Even if a write through policy is used, other processors may have invalid data in their caches
- In other words, if a processor updates its cache and updates main memory, a second processor may have been using the same data in its own cache which is now invalid.

Solutions to Prevent Problems with Multiprocessor/cache systems

- **Bus watching with write through** – each cache watches the bus to see if data they contain is being written to the main memory by another processor. All processors must be using the write through policy
- **Hardware transparency** – a "big brother" watches all caches, and upon seeing an update to any processor's cache, it updates main memory AND all of the caches
- **Noncacheable memory** – Any shared memory (identified with a chip select) may not be cached.

Line Size

- There is a relationship between line size (i.e., the number of words in a line in the cache) and hit ratios
- As the line size (block size) goes up, the hit ratio could go up due to more words available to the principle of **locality of reference**
- As block size increases, however, the number of blocks goes down, and the hit ratio will begin to go back down after a while
- Lastly, as the block size increases, the chances of a hit to a word farther from the initially referenced word goes down

Multi-Level Caches

- Increases in transistor densities have allowed for caches to be placed inside processor chip
- Internal caches have very short wires (within the chip itself) and are therefore quite fast, even faster then any zero wait-state memory accesses outside of the chip
- This means that a super fast internal cache (level 1) can be inside of the chip while an external cache (level 2) can provide access faster then to main memory

Unified versus Split Caches

- Split into two caches – one for instructions, one for data
- Disadvantages
  - Questionable as unified cache balances data and instructions merely with hit rate.
  - Hardware is simpler with unified cache
- Advantage
  - What a split cache is really doing is providing one cache for the instruction decoder and one for the execution unit.
  - This supports pipelined architectures.

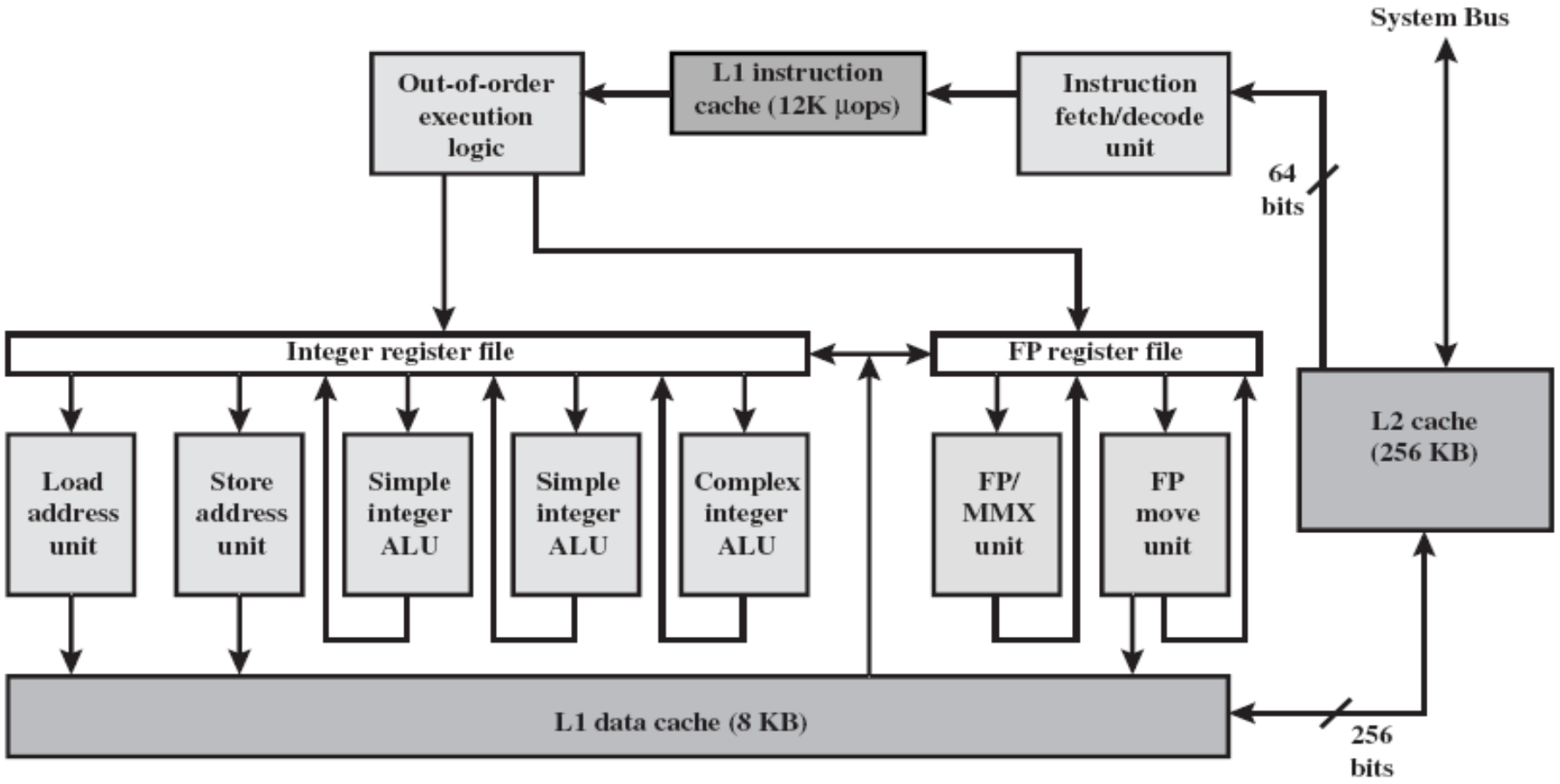
Intel x86 caches

- 80386 – no on chip cache
- 80486 – 8k using 16 byte lines and four-way set associative organization (main memory had 32 address lines – 4 Gig)
- Pentium (all versions)
  - Two on chip L1 caches
  - Data & instructions

Pentium 4 L1 and L2 Caches

- L1 cache
  - 8k bytes
  - 64 byte lines
  - Four way set associative
- L2 cache
  - Feeding both L1 caches
  - 256k
  - 128 byte lines
  - 8 way set associative

Pentium 4 (Figure 4.13)



Pentium 4 Operation – Core Processor

- Fetch/Decode Unit
  - Fetches instructions from L2 cache
  - Decode into micro-ops
  - Store micro-ops in L1 cache

- Out of order execution logic
  - Schedules micro-ops
  - Based on data dependence and resources
  - May speculatively execute

Pentium 4 Operation – Core Processor (continued)

- Execution units
  - Execute micro-ops
  - Data from L1 cache
  - Results in registers

- Memory subsystem – L2 cache and systems bus

Pentium 4 Design Reasoning

- Decodes instructions into RISC like micro-ops before L1 cache
- Micro-ops fixed length – Superscalar pipelining and scheduling
- Pentium instructions long & complex
- Performance improved by separating decoding from scheduling & pipelining – (More later – ch14)

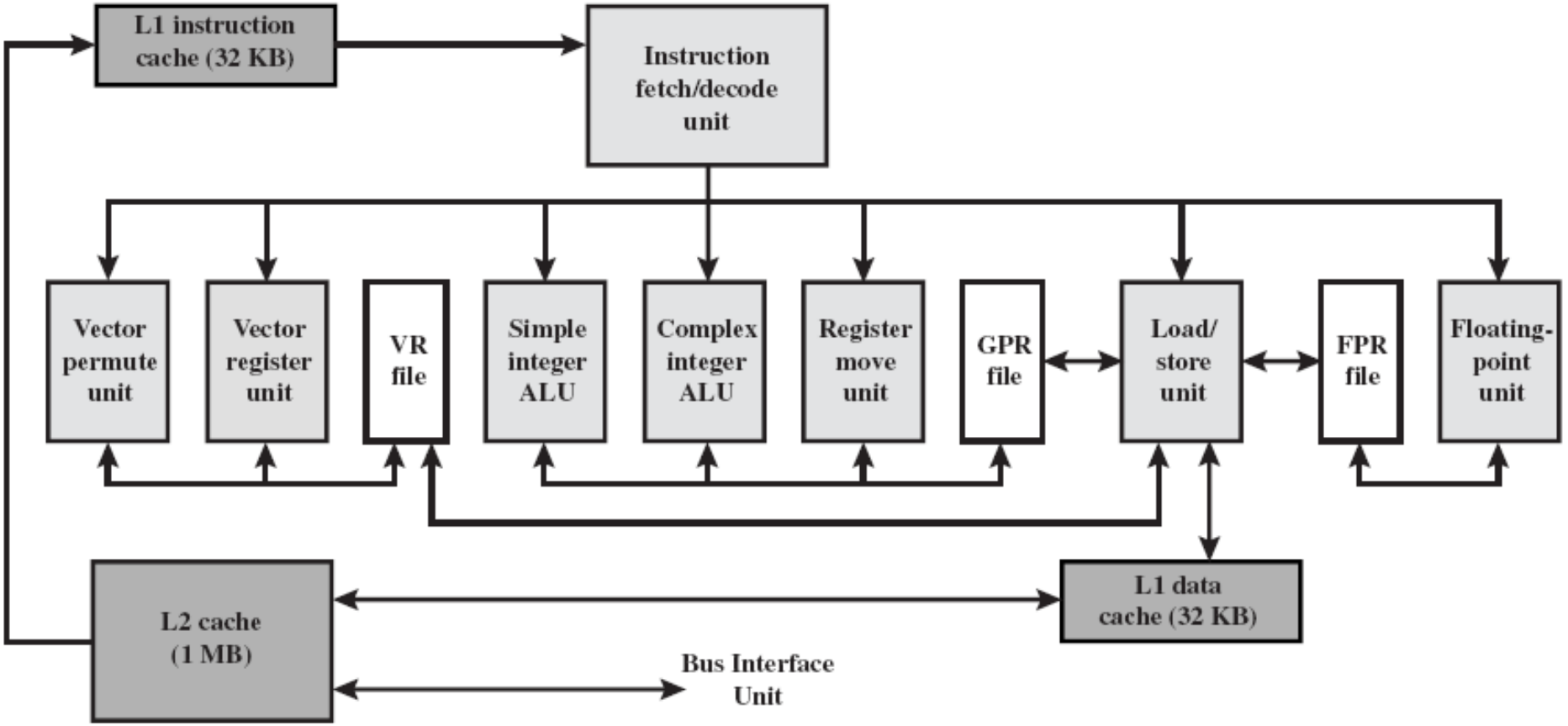
Pentium 4 Design Reasoning (continued)

- Data cache is write back – Can be configured to write through
- L1 cache controlled by 2 bits in register
  - CD = cache disable
  - NW = not write through
  - 2 instructions to invalidate (flush) cache and write back then invalidate

Power PC Cache Organization

- 601 – single 32kb 8 way set associative
- 603 – 16kb (2 x 8kb) two way set associative
- 604 – 32kb
- 610 – 64kb
- G3 & G4
  - 64kb L1 cache – 8 way set associative
  - 256k, 512k or 1M L2 cache – two way set associative

PowerPC G4 (Figure 4.14)



Comparison of Cache Sizes (Table 4.3)

Table 4.3 Cache Sizes of Some Processors

Processor	Type	Year of Introduction	L1 cache[a]	L2 cache	L3 cache
IBM 360/85	Mainframe	1968	16 to 32 KB	—	—
PDP-11/70	Minicomputer	1975	1 KB	—	—
VAX 11/780	Minicomputer	1978	16 KB	—	—
IBM 3033	Mainframe	1978	64 KB	—	—
IBM 3090	Mainframe	1985	128 to 256 KB	—	—
Intel 80486	PC	1989	8 KB	—	—
Pentium	PC	1993	8 KB/8 KB	256 to 512 KB	—
PowerPC 601	PC	1993	32 KB	—	—
PowerPC 620	PC	1996	32 KB/32 KB	—	—
PowerPC G4	PC/server	1999	32 KB/32 KB	256 KB to 1 MB	2 MB
IBM S/390 G4	Mainframe	1997	32 KB	256 KB	2 MB
IBM S/390 G6	Mainframe	1999	256 KB	8 MB	—
Pentium 4	PC/server	2000	8 KB/8 KB	256 KB	—
IBM SP	High-end server/ supercomputer	2000	64 KB/32 KB	8 MB	—
CRAY MTA[b]	Supercomputer	2000	8 KB	2 MB	—
Itanium	PC/server	2001	16 KB/16 KB	96 KB	4 MB
SGI Origin 2001	High-end server	2001	32 KB/32 KB	4 MB	—

[a] Two values seperated by a slash refer to instruction and data caches

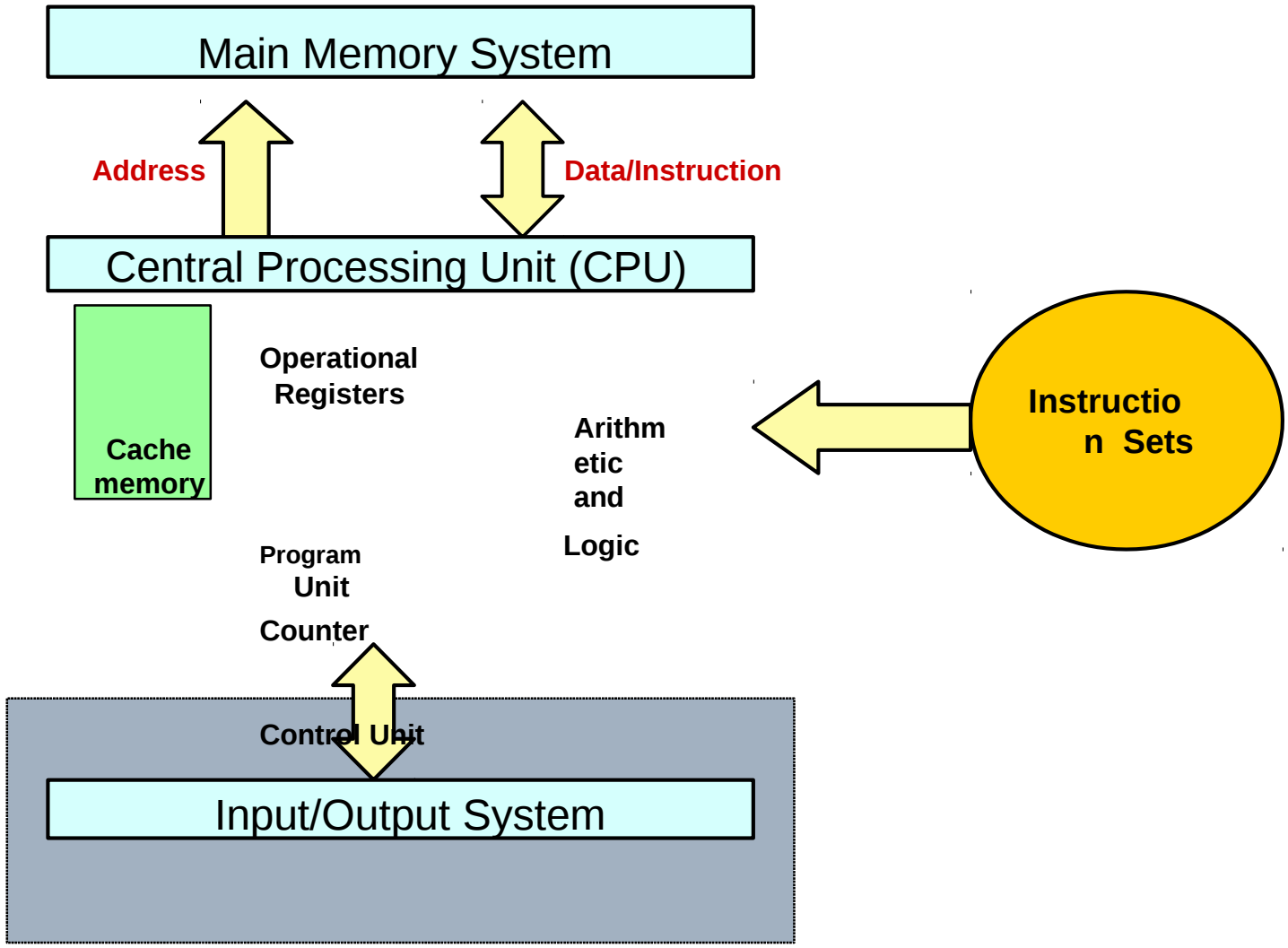
[b] Both caches are instruction only; no data caches

Unit 2
Input/Output Ports & Memory Systems

Outline

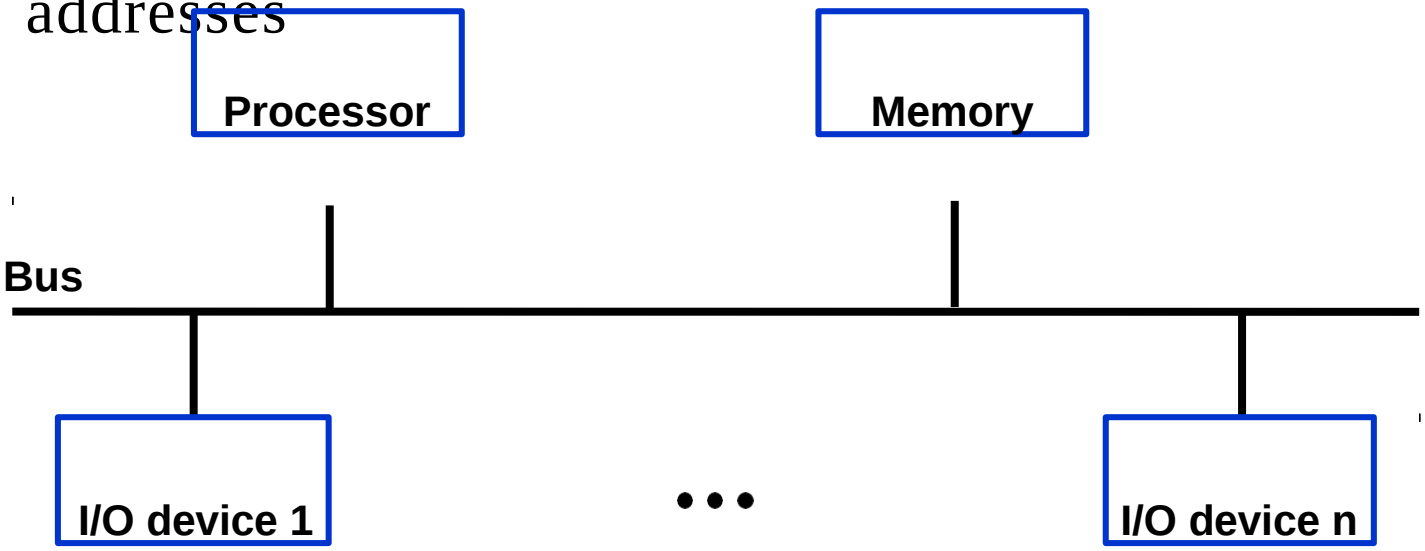
- Accessing I / O Devices
- Interrupts
- Direct Memory Access
- Buses
- Interface Circuits
- Standard I / O Interfaces

Content Coverage



Accessing I/O Devices

- Single-bus structure
  - The bus enables all the devices connected to it to exchange information
  - Typically, the bus consists of three sets of lines used to carry address, data, and control signals
  - Each I / O device is assigned a unique set of addresses



I/O Mapping

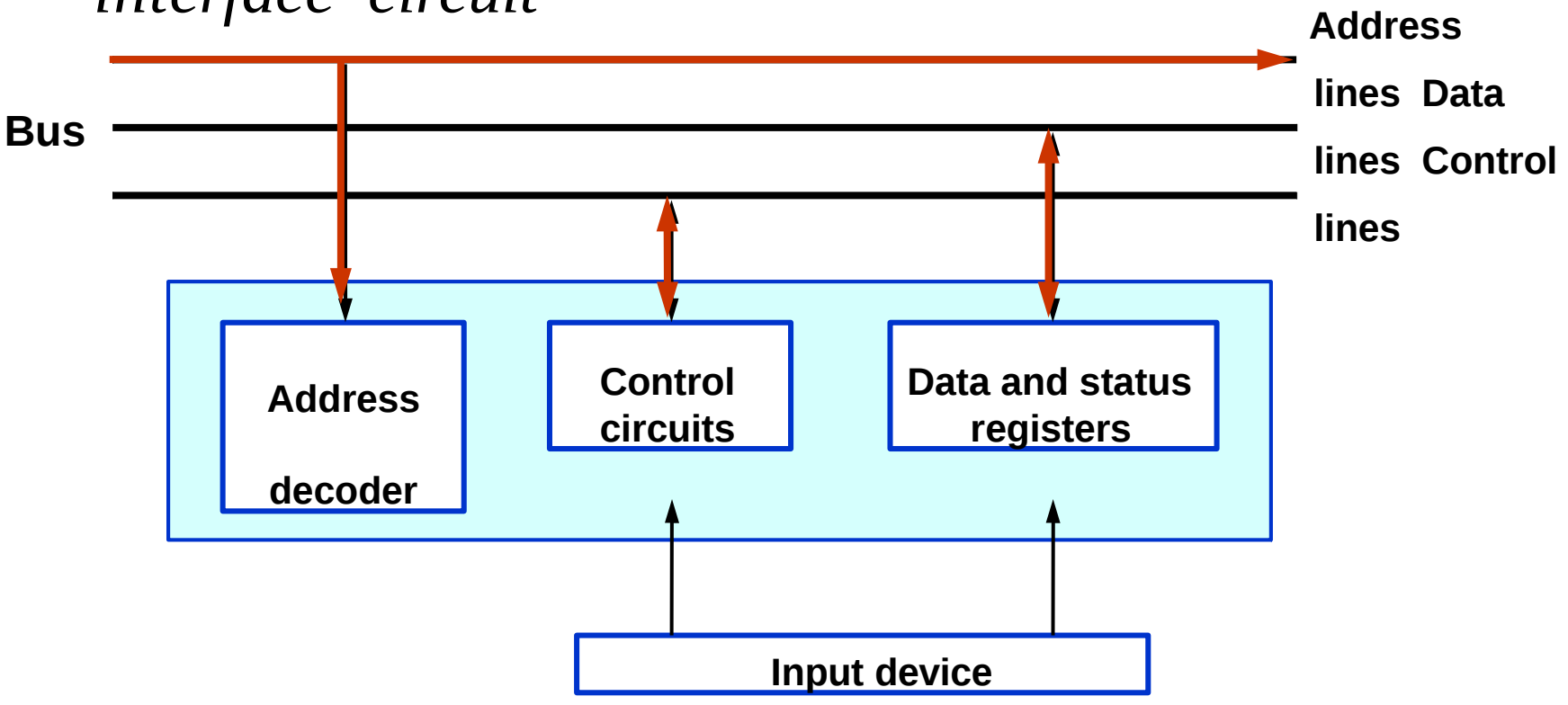
- Memory mapped I / O
  - Devices and memory share an address space
  - I / O looks just like memory read / write
  - No special commands for I/ O
    - Large selection of memory access commands available
- Isolated I / O
  - Separate address spaces
  - Need I / O or memory select lines
  - Special commands for I /O
    - Limited set

Memory-Mapped I/O

- When I / O devices and the memory share the same address space, the arrangement is called memory-mapped I / O
- With memory-mapped I / O, any machine instruction that can access memory can be used to transfer data to or from an I / O device
- Most computer systems use memory-mapped I / O.
- Some processors have special IN and OUT instructions to perform I / O transfers
  - When building a computer system based on these processors, the designer has the option of connecting I / O devices to usethe special I / O address space or simply incorporating them as part of the memory address space

I/O Interface for an Input Device

- The address decoder, the data and status registers, and the control circuitry required to coordinate I / O transfers constitute the device's *interface circuit*

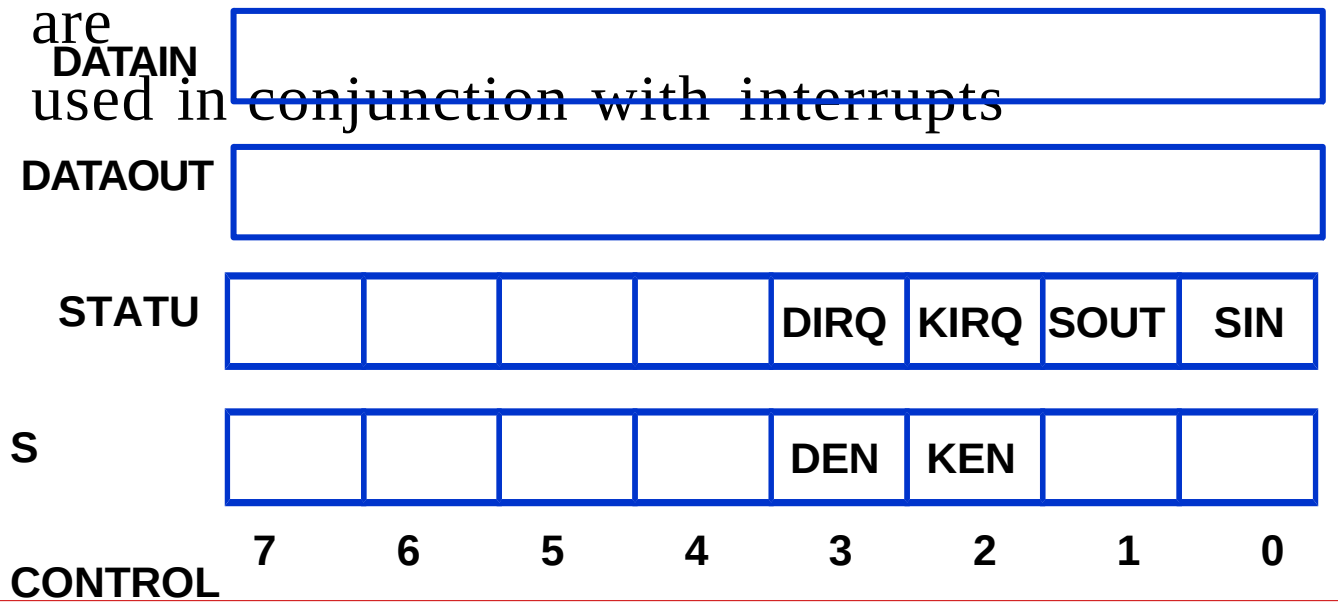


I/O Techniques

- Programmed
- Interrupt driven
- Direct Memory Access (DMA)

Program-Controlled I/O

- Consider a simple example of I / O operations involving a keyboard and a display device in a computer system. The four registers shown below are used in the data transfer operations
  - The two flags KIRQ and DIRQ in STATUS register are used in conjunction with interrupts



An Example

- A program that reads one line from the keyboard, stores it in memory buffer, and echoes it back to the display

```
        Move      #LINE, R0          Initialize memory pointer
WAITK   TestBit   #0,STATUS          Test SIN
        Branch=0  WAITK              Wait for character to be entered
        Move      DATAIN,R1          Read character
WAITD   TestBit   #1,STATUS          Test SOUT
        Branch=0  WAITD              Wait for display to become ready
        Move      R1,DATAOUT         Send character to display
        Move      R1,(R0)+           Store character and advance pointer
        Compare   #$0D,R1            Check if Carriage Return
        Branch≠0  WAITK              If not, get another character
        Move      #$0A,DATAOUT       Otherwise, send Line Feed
        Call      PROCESS            Call a subroutine to process the
                                     input line
```

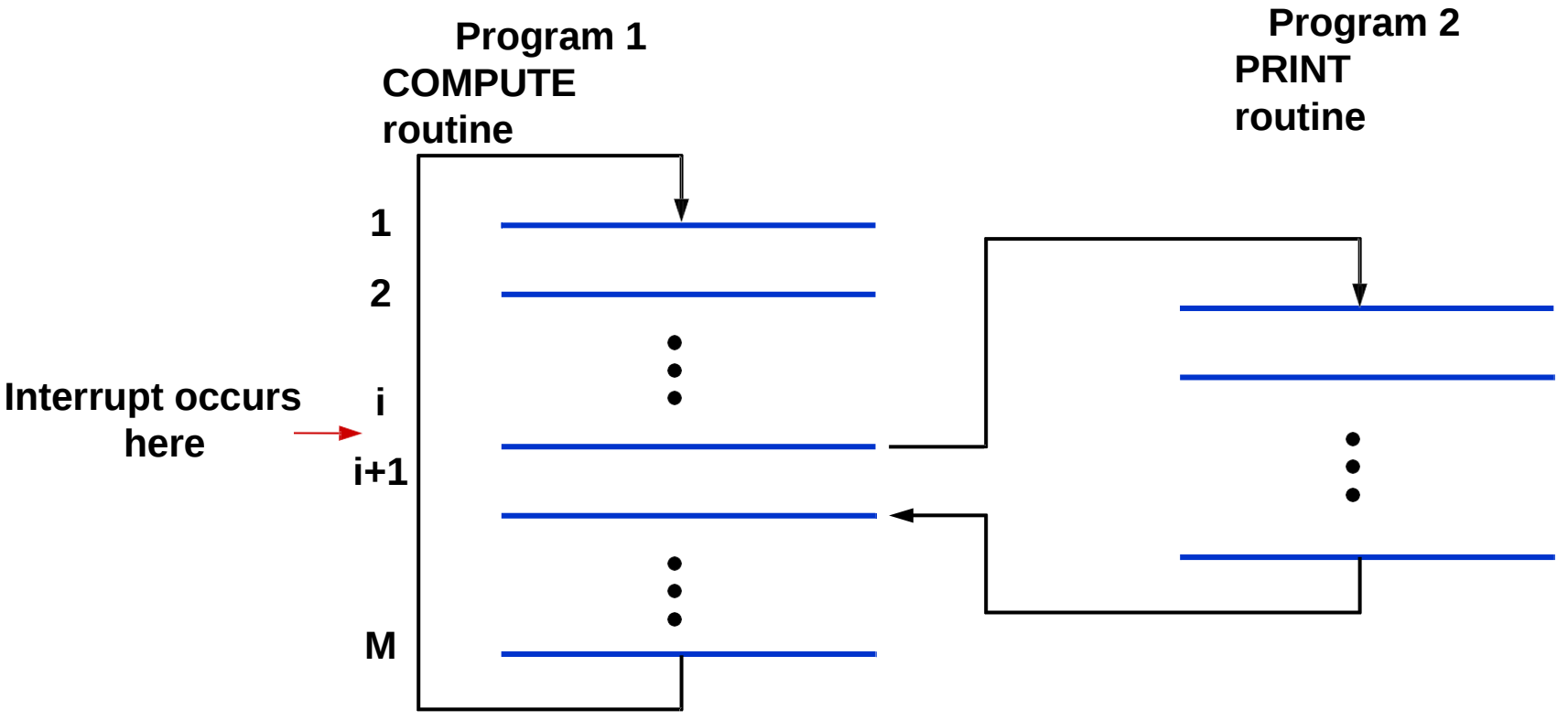
Program-Controlled I/O

- The example described above illustrates program-controlled I / O, in which the processor repeatedly checks a status flag to achieve the required synchronization between the processor and an input or output device. We say that the processor *polls* the devices
- There are two other commonly used mechanisms for implementing I / O operations: *interrupts* and *direct memory access*
  - Interrupts: synchronization is achieved by having the I / O device send a special signal over the bus whenever it is ready for a data transfer operation
  - Direct memory access: it involves having the device interface transfer data directly to or from the memory

Interrupts

- To avoid the processor being not performing any useful computation, a hardware signal called an *interrupt* to the processor can do it. At least one of the bus control lines, called an *interrupt-request* line, is usually dedicated for this purpose
- An *interrupt-service routine* usually is needed and is executed when an interrupt request is issued
- On the other hand, the processor must inform the device that its request has been recognized so that it may remove its interrupt-request signal. An *interrupt-acknowledge* signal serves

Example


Program 1
COMPUTE
routine
Program 2
PRINT
routine
1
2
i
i+1
M
Interrupt occurs
here

Interrupt-Service Routine & Subroutine

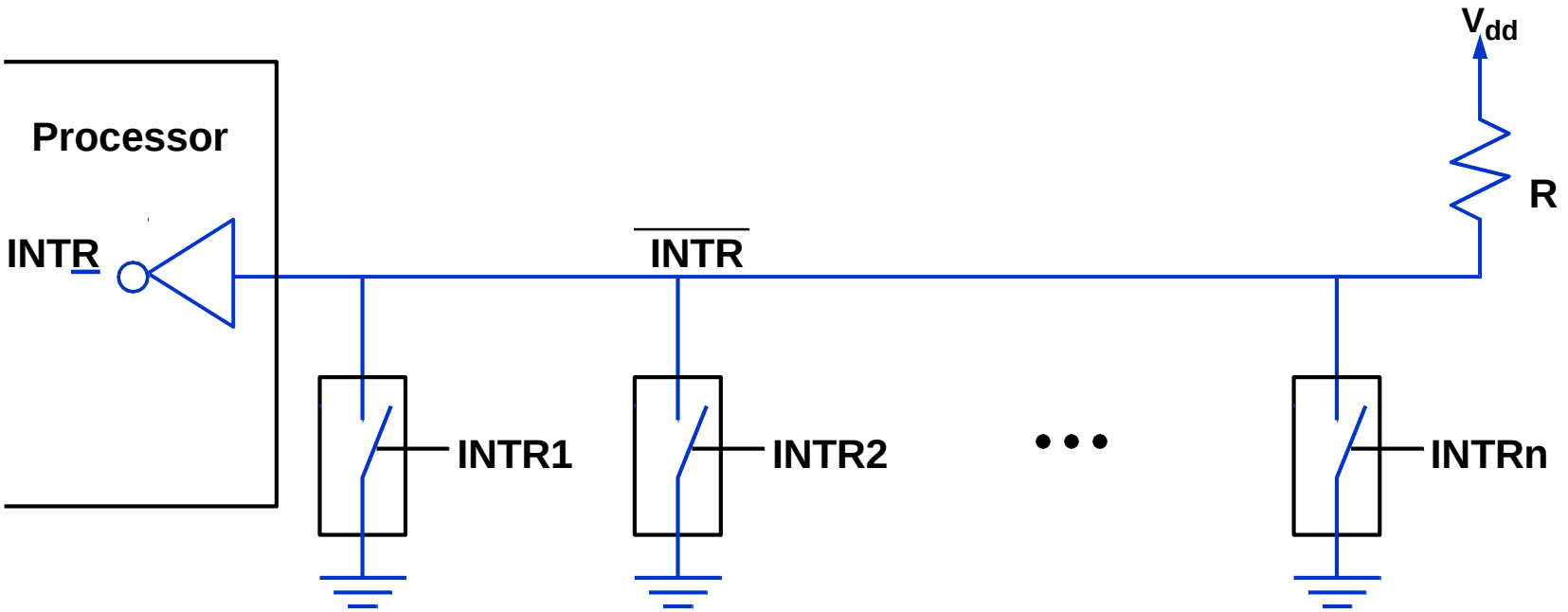
- Treatment of an interrupt-service routine is very similar to that of a subroutine
- An important departure from the similarity should be noted
  - A subroutine performs a function required by the program from which it is called.
  - The interrupt-service routine may not have anything in common with the program being executed at the time the interrupt request is received. In fact, the two programs often belong to different users
- Before executing the interrupt-service routine, any information that may be altered during the execution of that routine must be saved. This information must be restored before the interrupted program is resumed

Interrupt Latency

- The information that needs to be saved and restored typically includes the condition code flags and the contents of any registers used by both the interrupted program and the interrupt-service routine
- Saving registers also increases the delay between the time an interrupt request is received and the start of execution of the interrupt-service routine. The delay is called *interrupt latency*
- Typically, the processor saves only the contents of the program counter and the processor status register. Any additional information that needs to be saved must be saved by program instruction at the beginning of the interrupt-service routine and restored at the end of the routine

Interrupt Hardware

- An equivalent circuit for an open-drain bus used to implement a common interrupt-request line



INTR=INTR1+INTR2+…+INTRn

Handling Multiple Devices

- Handling multiple devices gives rise to a number of questions:
  - How can the processor recogniz e the device requesting an interrupt?
  - Given that different devices are likely to require different interrupt-service routines, how can the processor obtain the starting address of the approp riate routine in each case?
  - Should a device be allowed to interrupt the processor while another interrupt is being serviced?
  - How should two or more simult aneous interrupt request be handled?
- The information needed to determine whether a device is requesting an interrupt is available in its status register

Identify the Interrupting Device

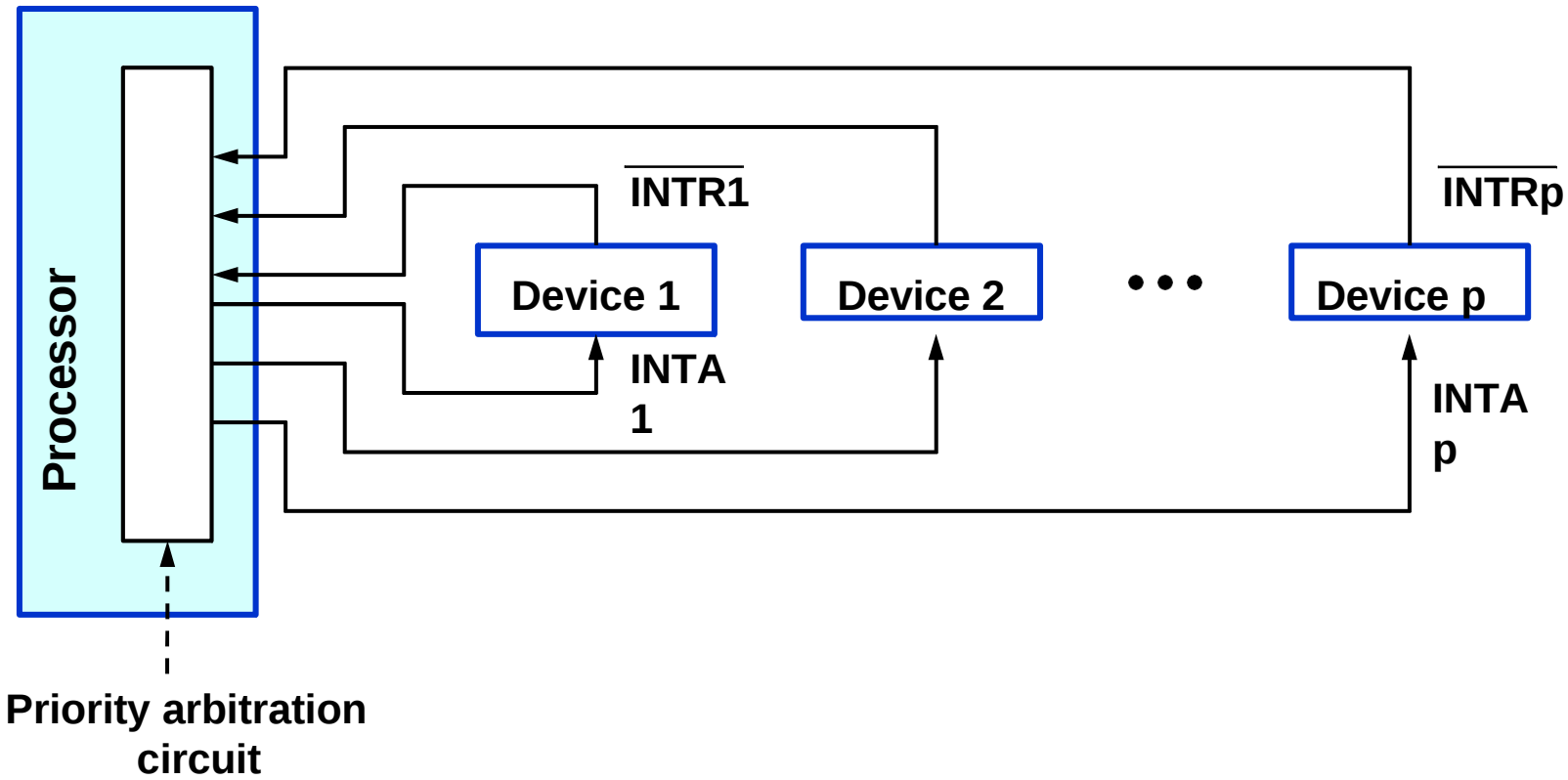
- The simplest way to identify the interrupting device is to have the interrupt-service routine poll all the
  I / O devices connected to the bus
  - The polling scheme is easy to implement. Its main disadvantage is the time spen t interrogating all the devices
- A device requesting an interrupt may identify itself directly to the processor. Then, the processor can immediately start executing the corresponding interrupt-service routine. This is called vectored interrupts
- An interrupt request from a high-priority device should be accepted while the processor is servicing another request from a lower-priority device

Interrupt Priority

- The processor's priority is usually encoded in a few bits of the processor status word. It can be changed by program instructions that write into the program status register (PS). These are privileged instructions, which can be executed only while the processor is running in the supervisor mode
- The processor is in the su pervisor mode only when executing operating system routines. It switches to the user mode before beginning to execute application program
- An attempt to execute a privileged instruction while in the user mode leads to a special type of interrupt called a privilege exception

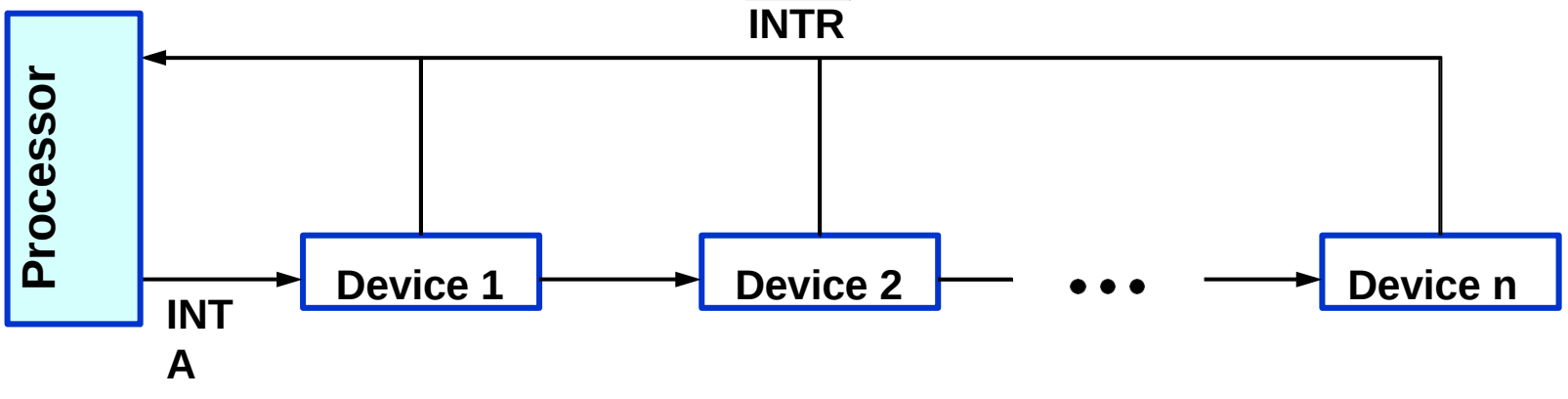
Implementation of Interrupt Priority

- An example of the implementation of a multiple-priority scheme



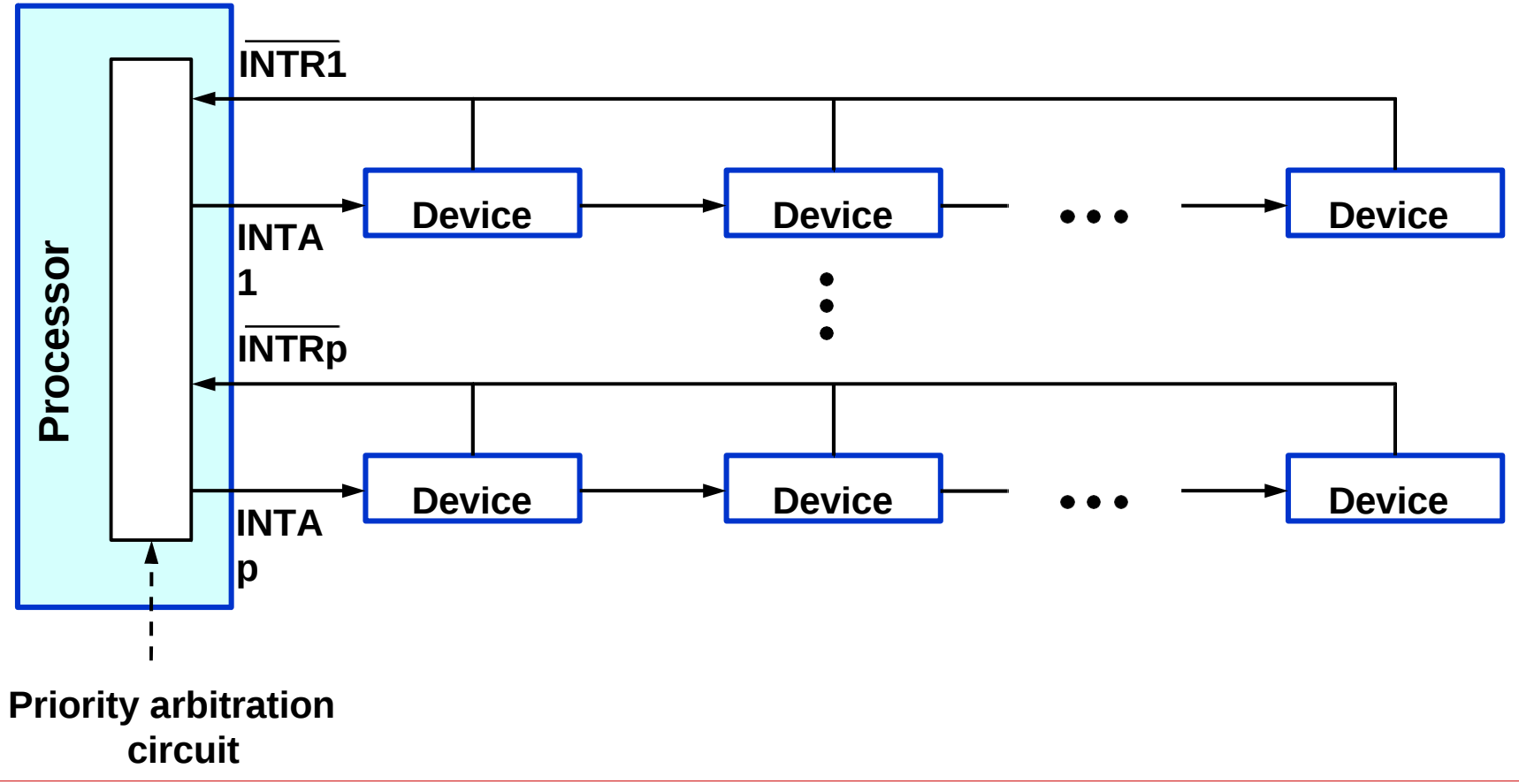
Simultaneous Requests

- Consider the problem of simultaneous arrivals of interrupt requests from two or more devices. The processor must have some means of deciding which request to service first
- Interrupt priority scheme with daisy chain



Priority Group

- Combination of the interrupt priority scheme with daisy chain and with individual interrupt-request and interrupt-acknowledge lines

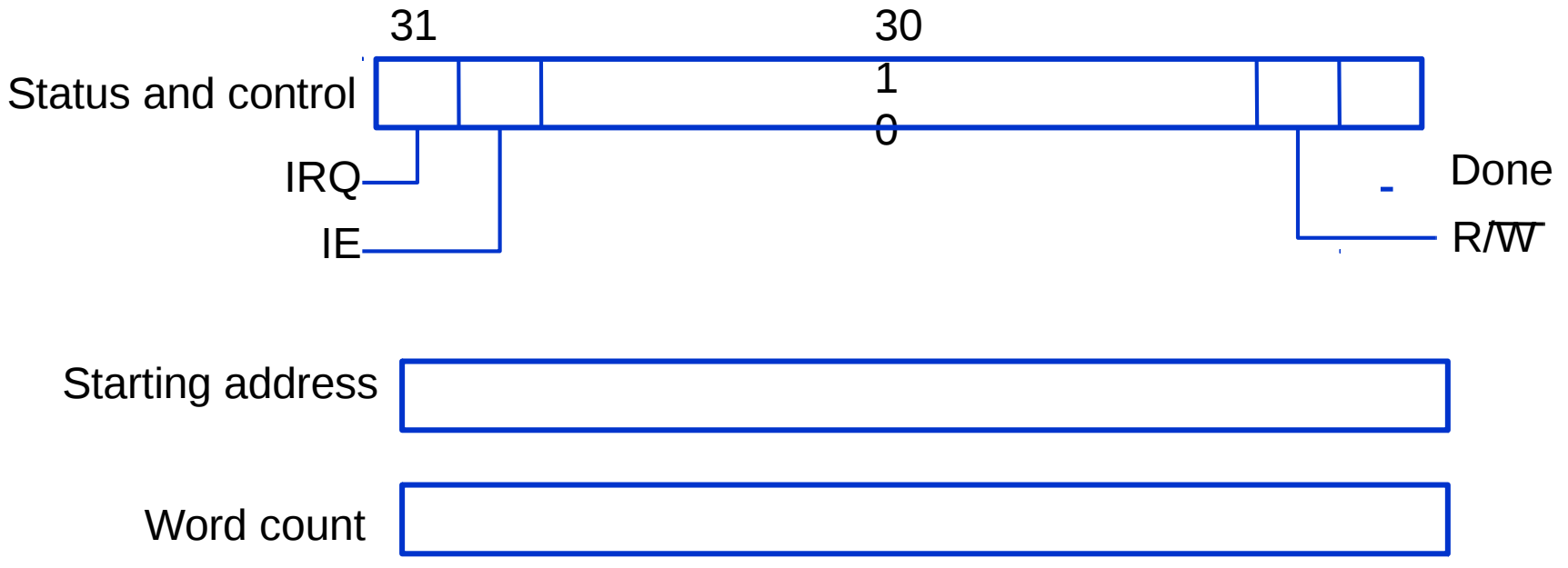


Direct Memory Access

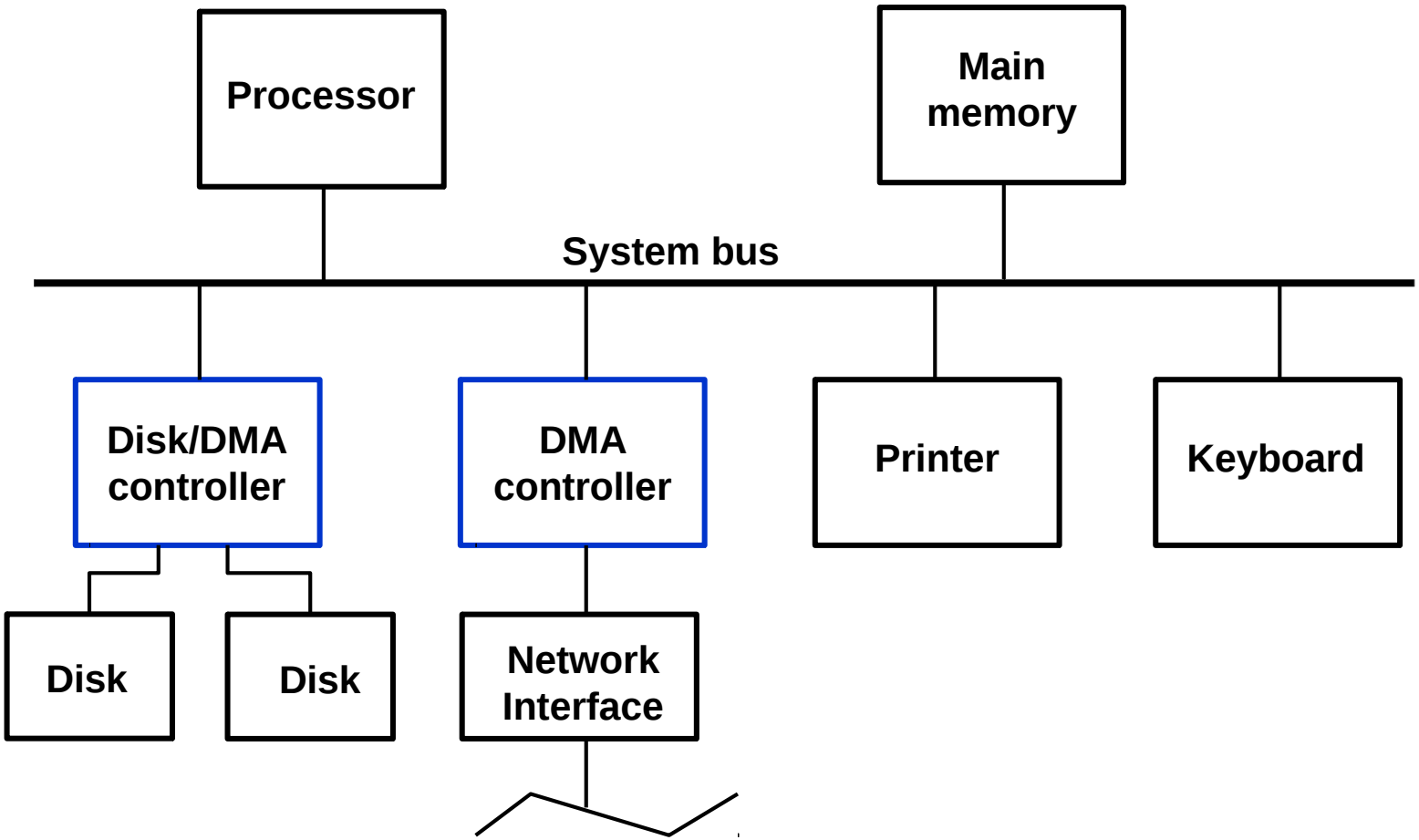
- To transfer large blocks of data at high speed, a special control unit may be provided between an external device and the main memory, without continuous intervention by the processor. This approach is called *direct memory access* (DMA)
- DMA transfers are performed by a control circuit that is part of the I / O device in terface. We refer to this circuit as a DMA controller.
- Since it has to transfer blocks of data, the DMA controller must increment the memory address for successive words and keep track of the number of transfers

DMA Controller

- Although a DMA controller can transfer data without intervention by the processor, its operation must be under the control of a program executed by the processor
- An example



DMA Controller in a Computer System



Memory Access Priority

- Memory accesses by the processor and the DMA controllers are interwoven. Request by DMA devices for using the bus are always given higher priority than processor requests.
- Among different DMA devices, top priority is given to high-speed peripherals such as a disk, a high-speed network interface, etc.
- Since the processor originates most memory access cycles, the DMA controller can be said to "steal" memory cycles from the processor. Hence, this interweaving technique is usually called *cycle stealing*
- The DMA controller may transfer a block of data without interruption. This is called *block/burst* mode

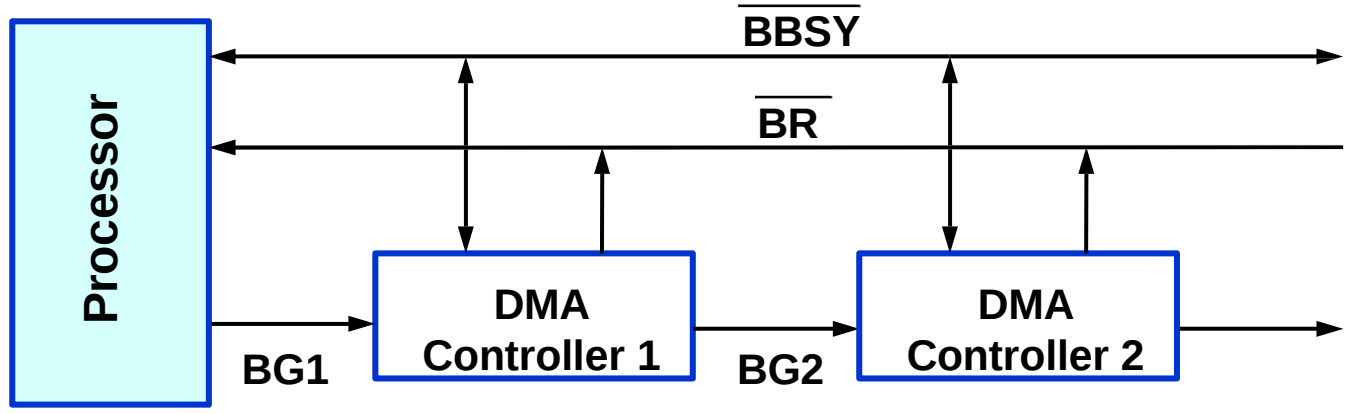
Bus Arbitration

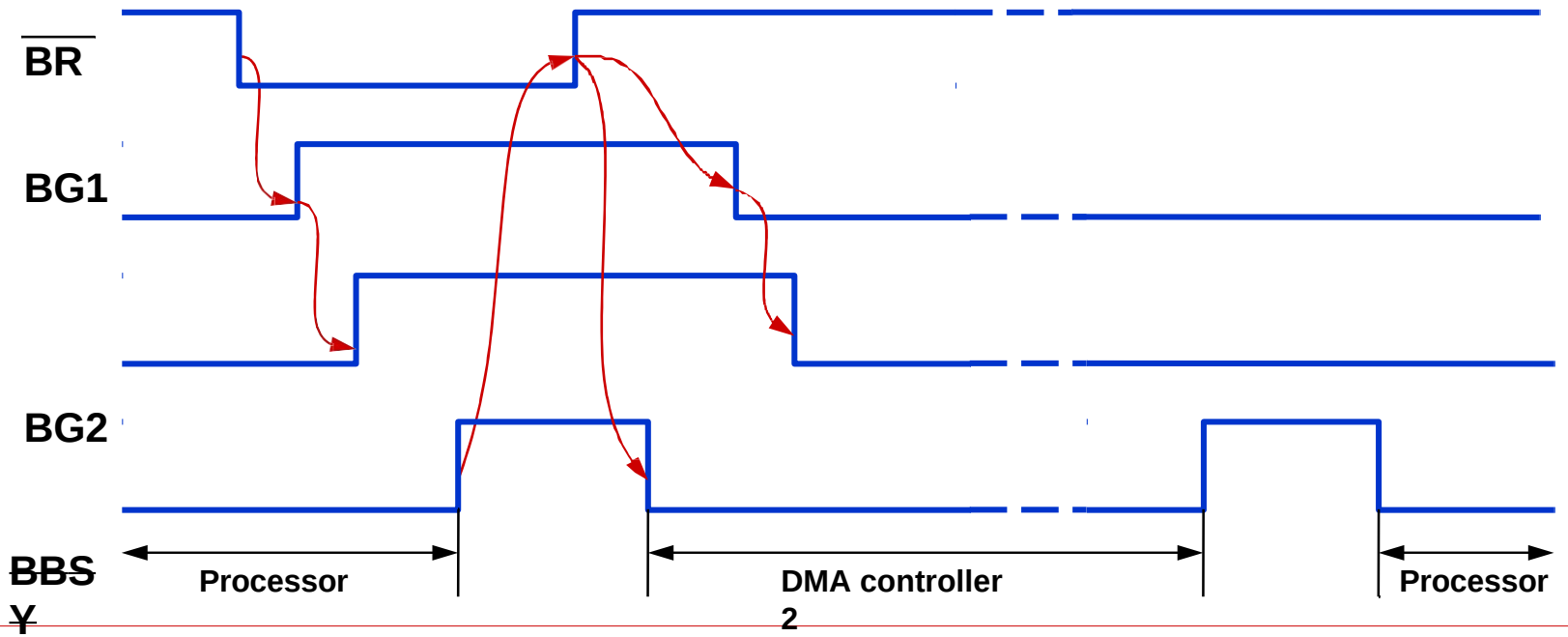
- A conflict may arise if both the processor and a DMA controller or two DMA controllers try to use the bus at the same time to access the main memory. To resolve this problem, an arbitration procedure on bus is needed
- The device that is allowed to initiate data transfer on the bus at any given time is called the bus master. When the current master relinquishes control of the bus, another device can acquire this status
- Bus arbitration is the process by which the next device to become the bus master take into account the needs of various devices by establishing a priority system for gaining access to the bus

Bus Arbitration

- There are two approaches to bus arbitration
  - Centralized and distributed
- In centralized arbitration, a single bus arbiter performs the required arbitration
- In distributed arbitration, all devices participate in the selection of the next bus master

Centralized Arbitration



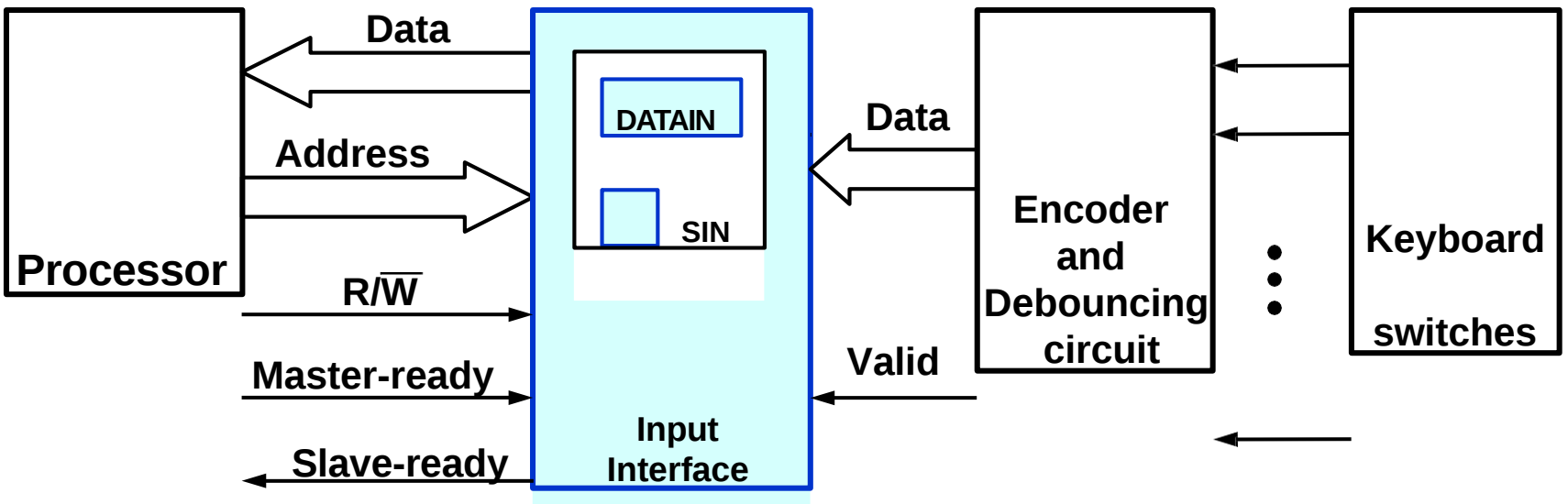


Buses

- A bus protocol is the set of rules that govern the behavior of various devices connected to the bus as to when to place information on the bus, assert control signals, and so on
- In a synchronous bus, all devices derive timing information from a common clock line. Equal spaced pulses on this line define equal time intervals
- In the simplest form of a synchronous bus, each of these intervals constitutes a bus cycle during which one data transfer can take place

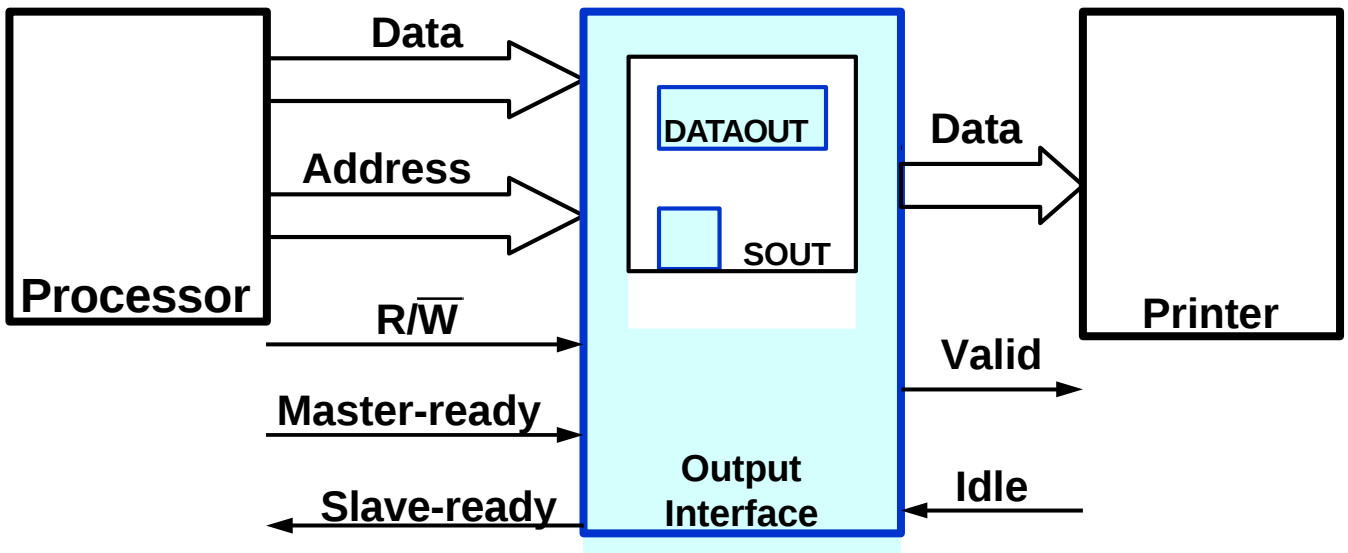
Interface Circuits

- Keyboard to processor connection
  - When a key is pressed, the Valid signal changes from 0 o 1, causing the ASCII code to be loaded into DATAIN and SIN to be set to 1
  - The status flag SIN is cleared to 0 when the processor reads the contents of the DATAIN register



Printer to Processor Connection

- The interface contains a data register, DATAOUT, and a status flag, SOUT
  - The SOUT flag is set to 1 when the printer is ready to accept another character, and it is cleared to 0 when a new character is loaded into DATAOUT by the processor
  - When the printer is ready to acce pt a character, it asserts its idle signal



Serial Port

- A serial port is used to connect the processor to
  I / O devices that require transmission of data one bit at a time
- The key feature of an interface circuit for a serial port is that it is capable of communicating in a bit-serial fashion on the device side and in a bit-parallel fashion on the bus side
- The transformation between the parallel and serial formats is achieved with shift registers that have parallel access capability

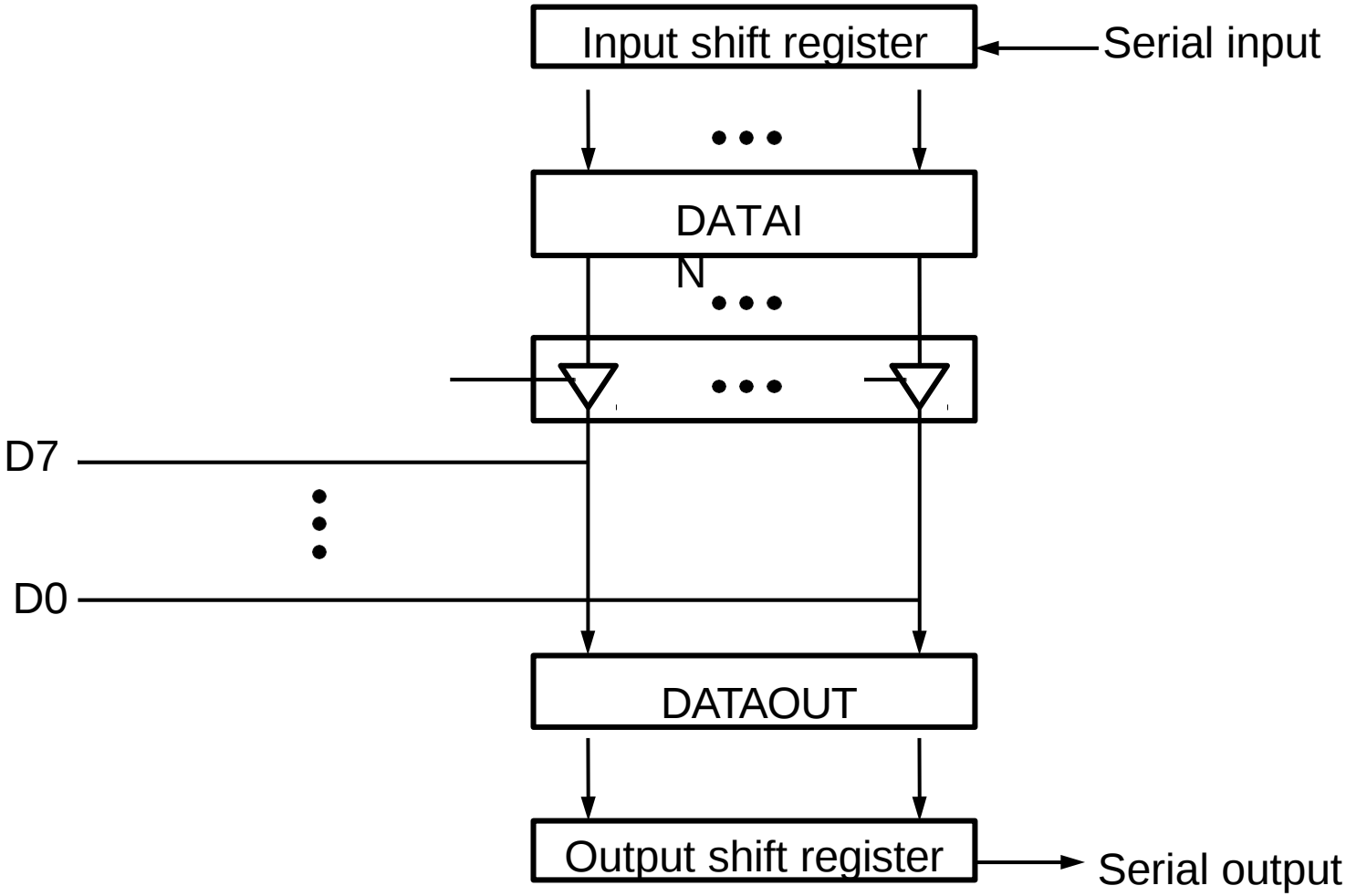
Shift Register

8 bit register == 8 bit data

4 modes

1. Serial In Serial Out (SISO)
2. Serial In Parallel Out (SIPO)
3. Parallel In Serial Out (PISO)
4. Parallel In Parallel Out (PIPO)

A Serial Interface

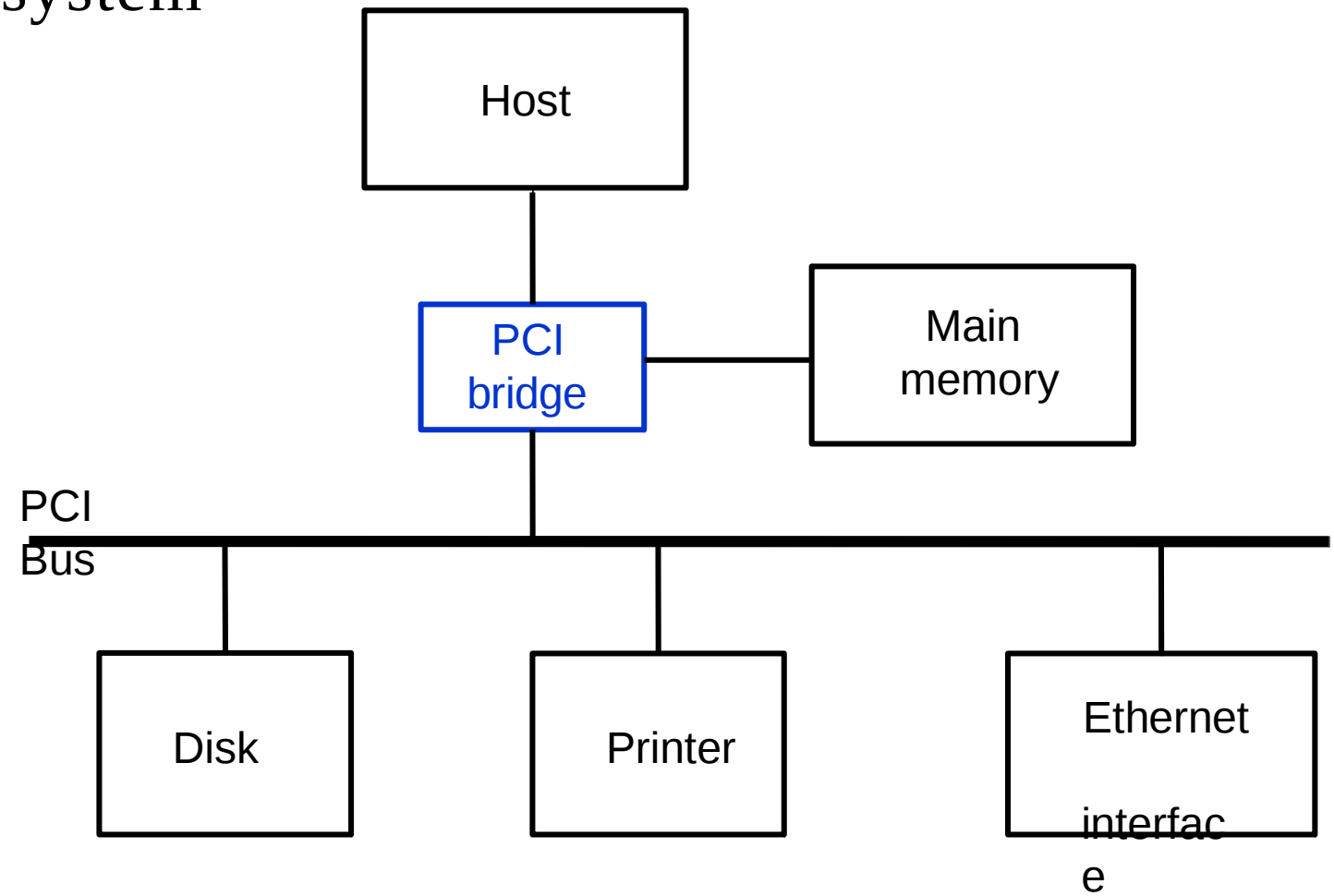


Standard I/O Interfaces

- The processor bus is the bus defined by the signals on the processor chip itself. Devices that require a very high speed connection to the processor, such as the main memory, may be connected directly to this bus
- The motherboard usually provides another bus that can support more devices.
- The two buses are interconnected by a circuit, which we called a bridge, that translates the signals and protocols of one bus into those of the other
- It is impossible to define a uniform standards for the processor bus. The structure of this bus is closely tied to the architecture of the processor
- The expansion bus is not subject to these limitations, and therefore it can use a standardized signaling structure

Peripheral Component Interconnect Bus

- Use of a PCI bus in a computer system



PCI Bus

- The bus support three independent address spaces: memory, I / O, and configuration.
- The I / O address space is intended for use with processors, such Pentium, that have a separate I / O address space.
- However, the system designer may choose to use memory-mapped I / O even when a separate I / O address space is available
- The configuration space is intended to give the PCI its plug-and-play capability.
  - A 4-bit command that accompanies the address identifies which of the three spaces is being used in a given data transfer operation

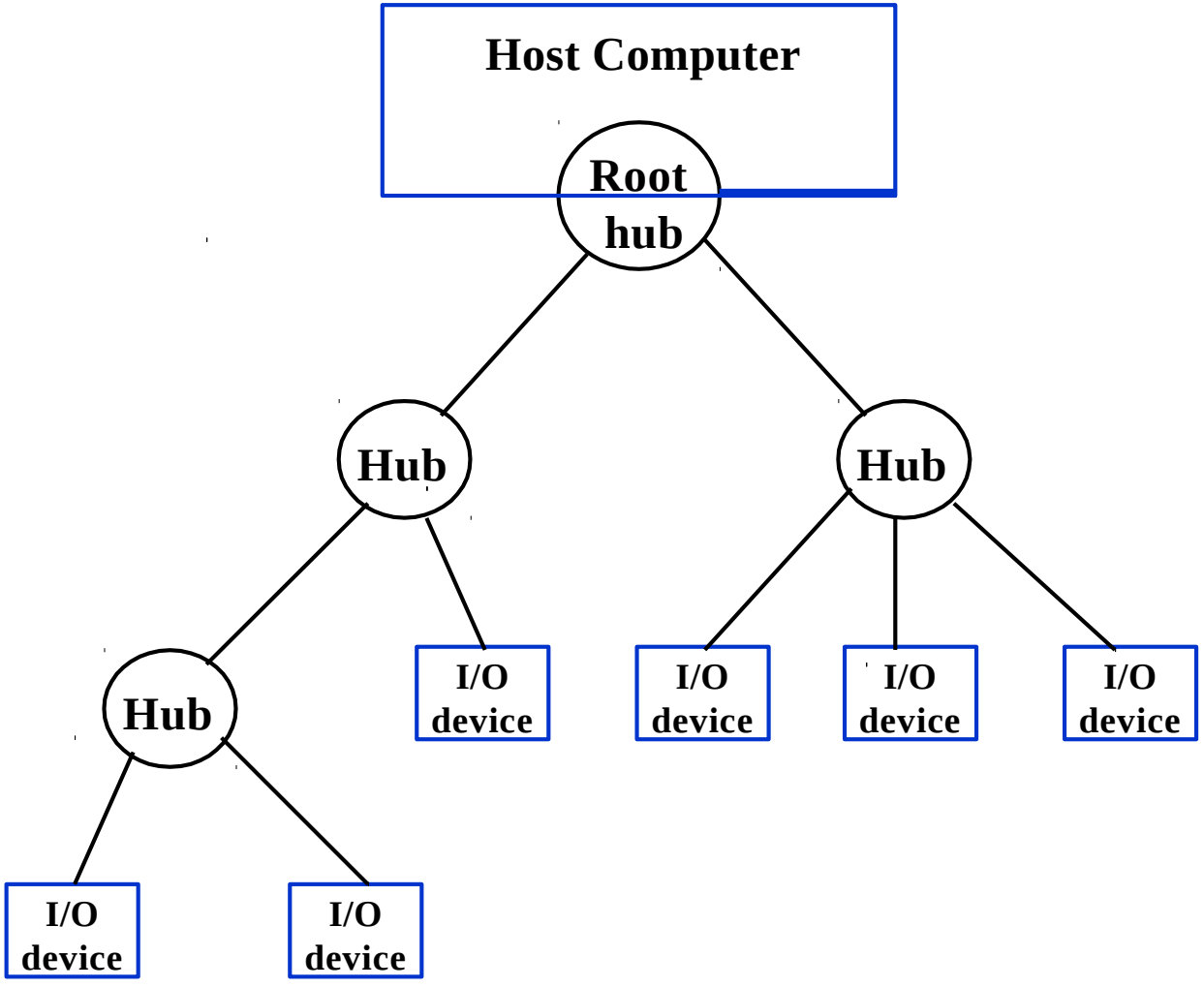
Universal Serial Bus (USB)

- The USB has been designed to meet several key objectives
  - Provide a simple, low-cost, and easy to use interconnection system that overcomes the difficulties due to the limited number of I / O ports available on a computer
  - Accommodate a wide range of data transfer characteristics for I / O devices, including telephone and Internet connections
  - Enhance user convenience through a “plug-and-play” mode of operation

USB Structure

- A serial transmission format has been chosen for the USB because a serial bus satisfies the low-cost and flexibility requirements
- Clock and data information are encoded together and transmitted as a single signal
  - Hence, there are no limitations on clock frequency or distance arising from data skew
- To accommodate a large number of devices that can be added or removed at any time, the USB has the tree structure
  - Each node of the tree has a device called a hub, which acts as an intermediate control point between the host and the I / O device
  - At the root of the tree, a root hub connects the entire tree to the host computer

USB Tree Structure



USB Tree Structure

- The tree structure enables many devices to be connected while using only simple point-to-point serial links
- Each hub has a number of ports where devices may be connected, including other hubs
- In normal operation, a hub copies a message that it receives from its upstream connection to all its downstream ports
  - As a result, a message sent by th e host computer is broadcast to all I / O devices, but only the addresse d device will respond to that message
- A message sent from an I / O de vice is sent only upstream towards the root of the tree and is not seen by other devices
  - Hence, USB enables the host to communicate with the I / O devices, but it does not enable these devices to communicate with each other

USB Protocols

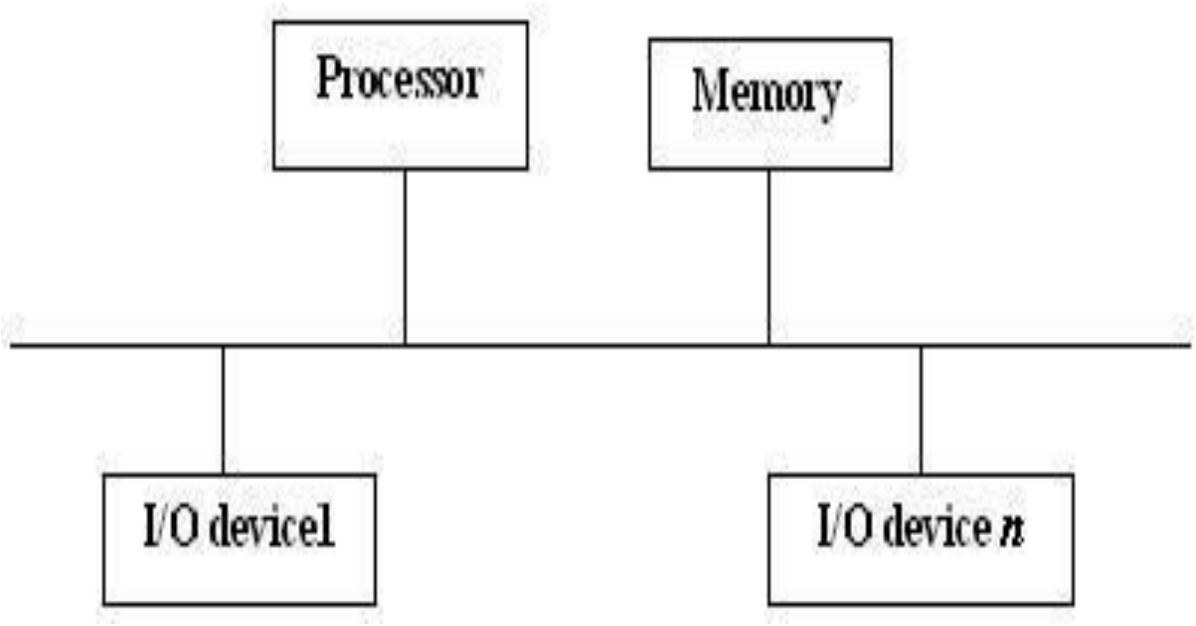
- All information transferred over the USB is organized in packets, where a packet consists of one or more bytes of information
- The information transferred on the USB can be divided into two broad categories: control and data
  - Control packets perform such tasks as addressing a device to initiate data transfer, acknowledging that data have been received correctly, or indicating an error
  - Data packets carry informatio n that is delivered to a device. For example, input and output data are transferred inside data packets

INPUT/OUTPUT ORGANIZATION

Introduction

A general purpose computer should have the ability to exchange information with a wide range of devices in varying environments. Computers can communicate with other computers over the Internet and access information around the globe. They are an integral part of home appliances, manufacturing equipment, transportation systems, banking and point-of-sale terminals. In this chapter, we study the various ways in which I/O operations are performed.

4.1 Accessing I/O Devices



A single-bus structure

A simple arrangement to connect I/O devices to a computer is to use a single bus arrangement, as shown in above figure. Each I/O device is assigned a unique set of address. When the processor places a particular address on the address lines, the device that recognizes this address responds to the commands issued on the control lines. The processor requests either a read or a write operation which is transferred over the data lines. When I/O devices and the memory share the same address space, the arrangement is called memory-mapped I/O.

Consider, for instance, with memory-mapped I/O, if DATAIN is the address of the input buffer of the keyboard

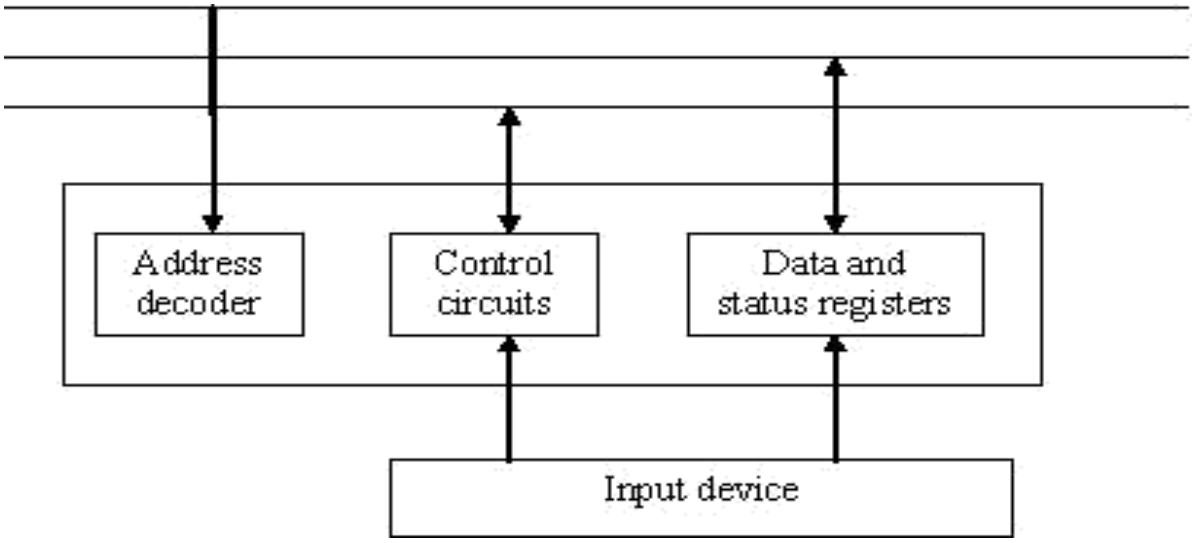
Move DATAIN, R0

And DATAOUT is the address of the output buffer of the display/printer

Move R0, DATAOUT

This sends the contents of register R0 to location DATAOUT, which may be the output data buffer of a display unit or a printer.

Most computer systems use memory-mapped I/O. Some processors have special I/O instructions to perform I/O transfers. The hardware required to connect an I/O device to the bus is shown below:



I/O interface for an input device

The address decoder enables the device to recognize its address when this address appears on the address lines. The data register holds the data. The status register contains information. The address decoder, data and status registers and controls required to coordinate I/O transfers constitutes interface circuit

For eg: Keyboard, an instruction that reads a character from the keyboard should be executed only when a character is available in the input buffer of the keyboard interface. The processor repeatedly checks a status flag to achieve the synchronization between processor and I/O device, which is called as program-controlled I/O.

Two commonly used mechanisms for implementing I/O operations are:

- Interrupts and
- Direct memory access

Interrupts: synchronization is achieved by having the I/O device send a special signal over the bus whenever it is ready for a data transfer operation.

Direct memory access: For high speed I/O devices. The device interface transfer data directly to or from the memory without informing the processor.

4.2 Interrupts

There are many situations where other tasks can be performed while waiting for an I/O device to become ready. A hardware signal called an Interrupt will alert the processor when an I/O device becomes ready. Interrupt-request line is usually dedicated for this purpose.

For example, consider, COMPUTE and PRINT routines. The routine executed in response to an interrupt request is called interrupt-service routine. Transfer of control through the use of interrupts happens. The processor must inform the device that its request has been recognized by sending interrupt-acknowledge signal. One must therefore know the difference between Interrupt Vs Subroutine. Interrupt latency is concerned with saving information in registers will increase the delay between the time an interrupt request is received and the start of execution of the interrupt-service routine.

Interrupt hardware

Most computers have several I/O devices that can request an interrupt. A single interrupt request line may be used to serve n devices.

Enabling and Disabling Interrupts

All computers fundamentally should be able to enable and disable interruptions as desired. Again reconsider the COMPUTE and PRINT example. When a device activates the interrupt-request signal, it keeps this signal activated until it learns that the processor has accepted its request. When interrupts are enabled, the following is a typical scenario:

- The device raises an interrupt request.
- The processor interrupts the program currently being executed.
- Interrupts are disabled by changing the control bits in the processor status register (PS).
- The device is informed that its request has been recognized and deactivates the interrupt request signal.
- The action requested by the interrupt is performed by the interrupt-service routine.
- Interrupts are enabled and execution of the interrupted program is resumed.

Handling multiple devices

While handling multiple devices, the issues concerned are:

- How can the processor recognize the device requesting an interrupt?
- How can the processor obtain the starting address of the appropriate routine?
- Should a device be allowed to interrupt the processor while another interrupt is being serviced?
- How should two or more simultaneous interrupt requests be handled?

Vectored interrupts

A device requesting an interrupt may identify itself (by sending a special code) directly to the processor, so that the processor considers it immediately.

Interrupt nesting

The processor should continue to execute the interrupt-service routine till completion, before it accepts an interrupt request from a second device. Privilege exception means they execute privileged instructions. Individual interrupt-request and acknowledge lines can also be implemented. Implementation of interrupt priority using individual interrupt-request and acknowledge lines has been shown in figure 4.7.

Simultaneous requests

The processor must have some mechanisms to decide which request to service when simultaneous requests arrive. Here, daisy chain and arrangement of priority groups as the interrupt priority schemes are discussed. Priority based simultaneous requests are considered in many organizations.

Controlling device requests

At the device end, an interrupt enable bit determines whether it is allowed to generate an interrupt request. At the processor end, it determines whether a given interrupt request will be accepted.

Exceptions

The term exception is used to refer to any event that causes an interruption. Hence, I/O interrupts are one example of an exception.

- Recovery from errors – These are techniques to ensure that all hardware components are operating properly.
- Debugging – find errors in a program, trace and breakpoints (only at specific points selected by the user).
- Privilege exception – execute privileged instructions to protect OS of a computer.

Use of interrupts in Operating Systems

Operating system is system software which is also termed as resource manager, as it manages all variety of computer peripheral devices efficiently. Different issues addressed by the operating systems are: Assign priorities among jobs, Security and protection features, incorporate interrupt-service routines for all devices and Multitasking, time slice, process, program state, context switch and others.

4.4 Direct Memory Access

As we have seen earlier, the two commonly used mechanisms for implementing I/O operations are:

- Interrupts and
- Direct memory access

Interrupts: synchronization is achieved by having the I/O device send a special signal over the bus whenever it is ready for a data transfer operation

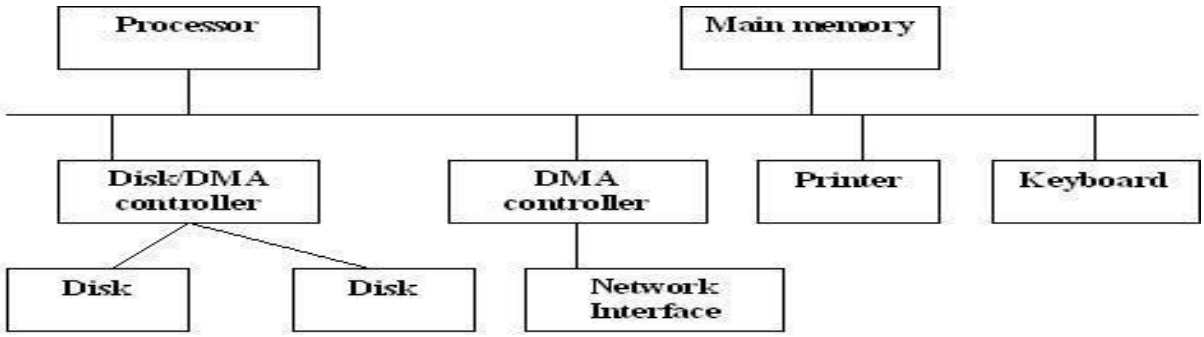
Direct memory access:

Basically for high speed I/O devices, the device interface transfer data directly to or from the memory without informing the processor. When interrupts are used, additional overhead involved with saving and restoring the program counter and other state information. To transfer large blocks of data at high speed, an alternative approach is used. A special control unit will allow transfer of a block of data directly between an external device and the main memory, without continuous intervention by the processor.

DMA controller is a control circuit that performs DMA transfers, is a part of the I/O device interface. It performs functions that normally be carried out by the processor. DMA controller must increment the memory address and keep track of the number of transfers. The operations of DMA controller must be under the control of a program executed by the processor. To initiate the transfer of block of words, the processor sends the starting address, the number of words in the block and the direction of the transfer. On receiving this information, DMA controller transfers the entire block and informs the processor by raising an interrupt signal. While a DMA transfer is taking place, the processor can be used to execute another program. After the DMA transfer is completed, the processor can return to the program that requested the transfer.

Three registers in a DMA interface are:

- Starting address
- Word count
- Status and control flag



Use of DMA controllers in a computer system

A conflict may arise if both the processor and a DMA controller or two DMA controllers try to use the bus at the same time to access the main memory. To resolve this, an arbitration procedure is implemented on the bus to coordinate the activities of all devices requesting memory transfers.

Bus Arbitration

The device that is allowed to initiate data transfers on the bus at any given time is called the bus master. Arbitration is the process by which the next device to become the bus master is selected and bus mastership is transferred to it. The two approaches are centralized and distributed arbitrations.

In centralized, a single bus arbiter performs the required arbitration whereas in distributed, all device participate in the selection of the next bus master. The bus arbiter may be the processor or a separate unit connected to the bus. The processor is normally the bus master unless it grants bus mastership to one of the DMA controllers. A simple arrangement for bus arbitration using daisy chain and a distributed arbitration scheme are discussed in figure 4.20 and 4.22 respectively.

In Centralized arbitration, A simple arrangement for bus arbitration using a daisy chain shows the arbitration solution. A rotating priority scheme may be used to give all devices an equal chance of being serviced (BR1 to BR4). In Distributed arbitration, all devices waiting to use the bus have equal responsibility in carrying out the arbitration process, without using a central arbiter. The drivers are of the open-collector type. Hence, if the input to one driver is equal to 1 and the input to another driver connected to the same bus line is equal to 0 the bus will be in the low-voltage state. This uses ARB0 to ARB3.

4.5 Buses

The Primary function of the bus is to provide a communication path for the transfer of data. It must also look in to,

-When to place information on the bus?

-When to have control signals?

Some bus protocols are set. These involve data, address and control lines. A variety of schemes have been devised for the timing of data transfers over a bus. They are:

Synchronous and Asynchronous schemes

Bus master is an initiator. Usually, processor acts as master. But under DMA setup, any other device can be master. The device addressed by the master is slave or target.

Synchronous bus

All devices derive timing information from a common clock line. Equally spaced pulses on this line define equal time intervals. Each of these intervals constitutes a bus cycle during which one data transfer can take place. Timing of an input/output transfer on a synchronous bus is shown in figure 4.23.

Asynchronous bus

This is a scheme based on the use of a handshake between the master and the slave for controlling data transfers on the bus. The common clock is replaced by two timing control lines, master-ready and slave-ready. The first is asserted by the master to indicate that it is ready for a transaction and the second is a response from the slave. The master places the address and command information on the bus. It indicates to all devices that it has done so by activating the master-ready line. This causes all devices on the bus to decode the address. The selected slave performs the required operation and informs the processor it has done so by activating the slave-ready line. A typical handshake control of data transfer during an input and an output operations are shown in figure 4.26 and 4.27 respectively. The master waits for slave-ready to become asserted before it removes its signals from the bus. The handshake signals are fully interlocked. A change of state in one signal is followed by a change in the other signal. Hence this scheme is known as a full handshake.

4.6 Interface Circuits

An I/O interface consists of the circuitry required to connect an I/O device to a computer bus. On one side of the interface, we have bus signals. On the other side, we have a data path with its associated controls to transfer data between the interface and the I/O device – port. We have two types:

Serial port and
Parallel port

A parallel port transfers data in the form of a number of bits (8 or 16) simultaneously to or from the device. A serial port transmits and receives data one bit at a time. Communication with the bus is the same for both formats. The conversion from the parallel to the serial format, and vice versa, takes place inside the interface circuit. In parallel port, the connection between the device and the computer uses a multiple-pin connector and a cable with as many wires. This arrangement is suitable for devices that are physically close to the computer. In serial port, it is much more convenient and cost-effective where longer cables are needed.

Typically, the functions of an I/O interface are:

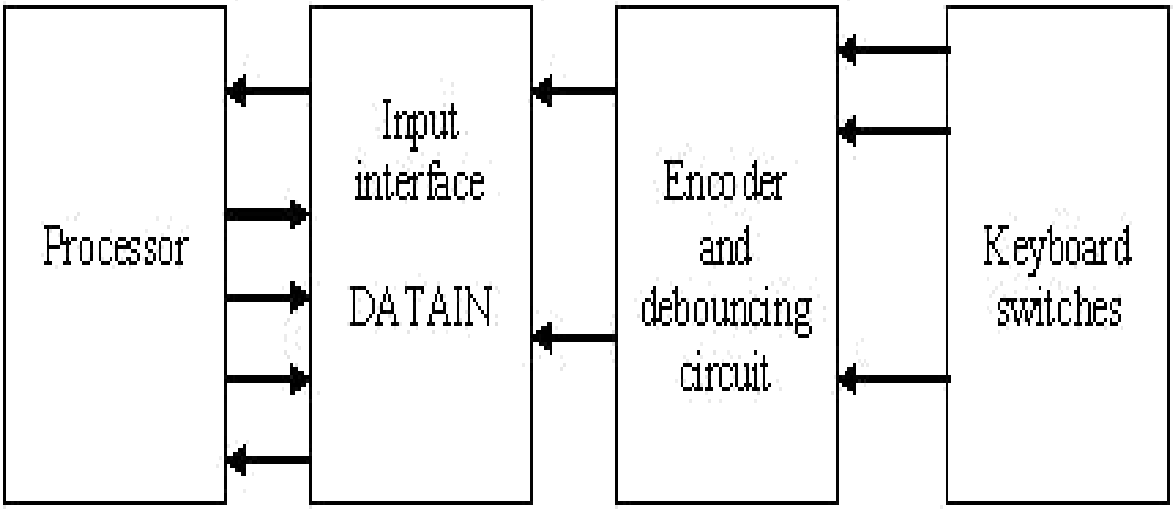
- Provides a storage buffer for at least one word of data
- Contains status flags that can be accessed by the processor to determine whether the buffer is full or empty
- Contains address-decoding circuitry to determine when it is being addressed by the processor
- Generates the appropriate timing signals required by the bus control scheme
- Performs any format conversion that may be necessary to transfer data between the bus and the I/O device, such as parallel-serial conversion in the case of a serial port.

Parallel Port

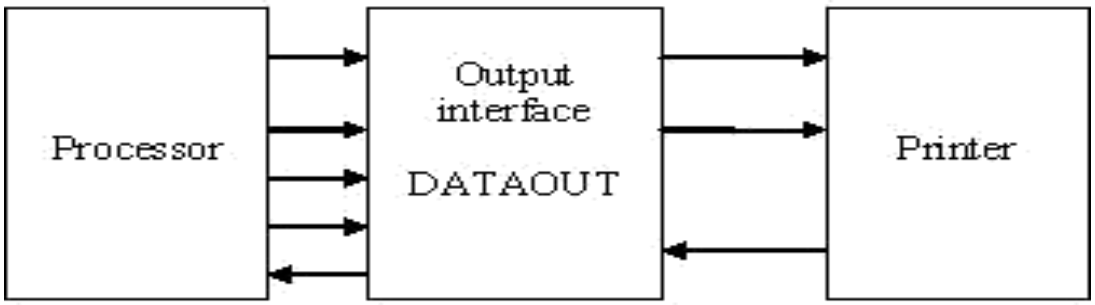
The hardware components needed for connecting a keyboard to a processor Consider the circuit of input interface which encompasses (as shown in below figure):

- –Status flag, SIN
- –R/~W
- –Master-ready
- –Address decoder

A detailed figure showing the input interface circuit is presented in figure 4.29. Now, consider the circuit for the status flag (figure 4.30). An edge-triggered D flip-flop is used along with read-data and master-ready signals



Keyboard to processor connection



Printer to processor connection

The hardware components needed for connecting a printer to a processor are: the circuit of output interface, and

-Slave-ready
-R/~W
-Master-ready
-Address decoder
-Handshake control

The input and output interfaces can be combined into a single interface. The general purpose parallel interface circuit that can be configured in a variety of ways. For increased flexibility, the circuit makes it possible for some lines to serve as inputs and some lines to serve as outputs, under program control.

Serial Port

A serial interface circuit involves – Chip and register select, Status and control, Output shift register, DATAOUT, DATAIN, Input shift register and Serial input/output – as shown in figure 4.37.

4.7 Standard I/O interfaces

Consider a computer system using different interface standards. Let us look in to Processor bus and Peripheral Component Interconnect (PCI) bus. These two buses are interconnected by a circuit called bridge. It is a bridge between processor bus and PCI bus. An example of a computer system using different interface standards is shown in figure 4.38. The three major standard I/O interfaces discussed here are:

-PCI (Peripheral Component Interconnect)
-SCSI (Small Computer System Interface)
-USB (Universal Serial Bus)

Peripheral Component Interconnect (PCI) Bus

The topics discussed under PCI are: Data Transfer, Use of a PCI bus in a computer system, A read operation on the PCI bus, Device configuration and Other electrical characteristics. Use of a PCI bus in a computer system is shown in figure 4.39 as a representation.

Host, main memory and PCI bridge are connected to disk, printer and Ethernet interface through PCI bus. At any given time, one device is the bus master. It has the right to initiate data transfers by issuing read and write commands. A master is called an initiator in PCI terminology. This is either processor or DMA controller. The addressed device that responds to read and write commands is called a target. A complete transfer operation on the bus, involving an address and a burst of data, is called a transaction. Device configuration is also discussed.

- The PCI bus is a good example of a system bus that grew out of the need for standardization.
- It supports the functions found on a processor bus bit in a standardized format that is independent of any particular processor.
- Devices connected to the PCI bus appear to the processor as if they were connected directly to the processor bus. They are assigned addresses in the memory address space of the processor.
- The PCI follows a sequence of bus standards that were used primarily in IBM PCs. Early PCs used the 8-bit XT bus, whose signals closely mimicked those of Intel's 80x86 processors. Later, the 16-bit bus used on the PC At computers became known as the **ISA bus**. Its extended 32-bit version is known as the EISA bus.
- Other buses developed in the eighties with similar capabilities are the Microchannel used in IBM PCs and the NuBus used in Macintosh computers.
- The PCI was developed as a low-cost bus that is truly processor independent. Its design anticipated a rapidly growing demand for bus bandwidth to support high-speed disks and graphic and video devices, as well as the specialized needs of multiprocessor systems. As a result, the PCI is still popular as an industry standard almost a decade after it was first introduced in 1992.
- **An important feature that the PCI pioneered is a plug-and-play capability** for connecting I/O devices. To connect a new device, the user simply connects the device interface board to the bus. The software takes care of the rest.

Data Transfer

- In today's computers, most memory transfers involve a burst of data rather than just one word. The reason is that modern processors include a cache memory. Data are transferred between the cache and the main memory in burst of several words each.
- The words involved in such a transfer are stored at successive memory locations. When the processor (actually the cache controller) specifies an address and requests a read operation from the main memory, the memory responds by sending a sequence of data words starting at that address. Similarly, during a write operation, the processor sends a memory address followed by a sequence of data words, to be written in successive memory locations starting at the address.
- The PCI is designed primarily to support this mode of operation. A read or write operation involving a single word is simply treated as a burst of length one.
- The bus supports three independent address spaces: **memory, I/O, and configuration**. The first two are self-explanatory. The I/O address space is intended for use with processors, such as Pentium, that have a separate I/O address space. However, as noted , the system designer may choose to use memory-mapped I/O even when a separate I/O address space is available.
- In fact, this is the approach recommended by the PCI its plug-and-play capability. A 4-bit command that accompanies the address identifies which of the three spaces is being used in a given data transfer operation.
- The signaling convention on the PCI bus is similar to the one used, we assumed that the master maintains the address information on the bus until data transfer is completed. But, this is not necessary. The address is needed only long enough for the slave to be selected. The slave can store the address in its internal buffer. Thus, the address is needed on the bus for one clock cycle only, freeing the address lines to be used for

sending data in subsequent clock cycles. The result is a significant cost reduction because the number of wires on a bus is an important cost factor. This approach in used in the PCI bus.

- At any given time, one device is the bus master. It has the right to initiate data
- transfers by issuing read and write commands. **A master is called an initiator in PCI terminology**. This is either a processor or a DMA controller.
- The addressed device that responds to read and write commands is called a target.

Device Configuration

- When an I/O device is connected to a computer, several actions are needed to configure both the device and the software that communicates with it.
- The PCI simplifies this process by incorporating in each I/O device interface a small configuration ROM memory that stores information about that device. The configuration ROMs of all devices is accessible in the configuration address space.
- The PCI initialization software reads these ROMs whenever the system is powered up or reset. In each case, it determines whether the device is a printer, a keyboard, an Ethernet interface, or a disk controller. It can further learn bout various device options and characteristics. Devices are assigned addresses during the initialization process. This means that during the bus configuration operation, devices cannot be accessed based on their address, as they have not yet been assigned one. Hence, the configuration address space uses a different mechanism. Each device has an input signal called Initialization Device Select, IDSEL#.
- The PCI bus has gained great popularity in the PC word. It is also used in many other computers, such as SUNs, to benefit from the wide range of I/O devices for which a PCI interface is available.
- In the case of some processors, such as the Compaq Alpha, the PCI-processor bridge circuit is built on the processor chip itself, further simplifying system design and packaging.

SCSI Bus

It is a standard bus defined by the American National Standards Institute (ANSI). A controller connected to a SCSI bus is an initiator or a target. The processor sends a command to the SCSI controller, which causes the following sequence of events to take place:

- The SCSI controller contends for control of the bus (initiator).
- When the initiator wins the arbitration process, it selects the target controller and hands over control of the bus to it.
- The target starts an output operation. The initiator sends a command specifying the required read operation.
- The target sends a message to the initiator indicating that it will temporarily suspends the connection between them. Then it releases the bus.

- The acronym SCSI stands for Small Computer System Interface. It refers to a standard bus defined by the American National Standards Institute (ANSI) under the designation X3.131 .
- In the original specifications of the standard, devices such as disks are connected to a computer via a 50-wire cable, which can be up to 25 meters in length and can transfer data at rates up to **5 megabytes/s**.
- The SCSI bus standard has undergone many revisions, and its data transfer capability has increased very rapidly, almost doubling every two years. SCSI-2 and SCSI-3 have been defined, and each has several options.
- A SCSI bus may have **eight data lines**, in which case it is called a narrow bus and transfers data one byte at a time.
- Alternatively, a wide SCSI bus has 16 data lines and transfers data 16 bits at a time.
- There are also several options for the electrical signaling scheme used.
- **Devices connected to the SCSI bus are not part of the address space of the processor** in the same way as devices connected to the processor bus.
- The SCSI bus is connected to the processor bus through a **SCSI controller**. **This controller uses DMA to transfer data packets** from the main memory to the device, or vice versa.
- A packet may contain a block of data, commands from the processor to the device, or status information about the device.
- Communication with a disk drive differs substantially from communication with the main memory.
- A controller connected to a SCSI bus is one of two types – an initiator or a target.
- An **initiator** has the ability to select a particular target and to send commands specifying the operations to be performed.
- Clearly, the controller on the processor side, such as the SCSI controller, must be able to operate as an initiator.
- The disk controller operates **as a target**. It carries out the commands it receives from the initiator.
- The initiator establishes a logical connection with the intended target. Once this connection has been established, it can be suspended and restored as needed to transfer commands and bursts of data.
- While a particular connection is suspended, other device can use the bus to transfer information.
- **This ability to overlap data transfer requests is one of the key features of the SCSI bus that leads to its high performance.**
- Data transfers on the SCSI bus are always controlled by the **target controller**. To send a command to a target, an initiator requests control of the bus and, after winning arbitration, selects the controller it wants to communicate with and hands control of the bus over to it.

  Then the controller starts a data transfer operation to receive a command from the initiator.

  The processor sends a command to the SCSI controller, which causes the following sequence of event to take place:
- The SCSI controller, acting as an initiator, contends for control of the bus.
- When the initiator wins the arbitration process, it selects the target controller and hands over control of the bus to it.
- The target starts an output operation (from initiator to target); in response to this, the initiator sends a command specifying the required read operation.

- The target, realizing that it first needs to perform a disk seek operation, sends a message to the initiator indicating that it will temporarily suspend the connection between them. Then it releases the bus.
- The target controller sends a command to the disk drive to move the read head to the first sector involved in the requested read operation. Then, it reads the data stored in that sector and stores them in a data buffer. When it is ready to begin transferring data to the initiator, the target requests control of the bus. After it wins arbitration, it reselects the initiator controller, thus restoring the suspended connection.
- The target transfers the contents of the data buffer to the initiator and then suspends the connection again. Data are transferred either 8 or 16 bits in parallel, depending on the width of the bus.
- The target controller sends a command to the disk drive to perform another seek operation. Then, it transfers the contents of the second disk sector to the initiator as before. At the end of this transfers, the logical connection between the two controllers is terminated.
- As the initiator controller receives the data, it stores them into the main memory using the DMA approach.
- The SCSI controller sends as interrupt to the processor to inform it that the requested operation has been completed.
- This scenario show that the messages exchanged over the SCSI bus are at a higher level than those exchanged over the processor bus. In this context, a "higher level" means that the messages refer to operations that may require several steps to complete, depending on the device. Neither the processor nor the SCSI controller need be aware of the details of operation of the particular device involved in a data transfer. In the preceding example, the processor need not be involved in the disk seek operation.

  - The target controller sends a command to the disk drive to move the read head to the first sector involved in the requested read operation.
  - The target transfers the contents of the data buffer to the initiator and then suspends the connection again.
  - The target controller sends a command to the disk drive to perform another seek operation.
  - As the initiator controller receives the data, it stores them into the main memory using the DMA approach.
  - The SCSI controller sends an interrupt to the processor to inform it that the requested operation has been completed.

The bus signals, arbitration, selection, information transfer and reselection are the topics discussed in addition to the above.

Universal Serial Bus (USB)

The USB has been designed to meet several key objectives such as:

- Provide a simple, low-cost and easy to use interconnection system that overcomes the difficulties due to the limited number of I/O ports available on a computer

- Accommodate a wide range of data transfer characteristics for I/O devices, including telephone and Internet connections
- Enhance user convenience through a "plug-and-play" mode of operation

Port Limitation

Here to add new ports, a user must open the computer box to gain access to the internal expansion bus and install a new interface card. The user may also need to know how to configure the device and the software. And also it is to make it possible to add many devices to a computer system at any time, without opening the computer box.

Device Characteristics

The kinds of devices that may be connected to a computer cover a wide range of functionality - speed, volume and timing constraints. A variety of simple devices attached to a computer generate data in different asynchronous mode. A signal must be sampled quickly enough to track its highest-frequency components.

Plug-and-play

Whenever a device is introduced, do not turn the computer off/restart to connect/disconnect a device. The system should detect the existence of this new device automatically, identify the appropriate device-driver software and any other facilities needed to service that device, and establish the appropriate addresses and logical connections to enable them to communicate.

USB architecture

To accommodate a large number of devices that can be added or removed at any time, the USB has the tree structure. Each node has a device called a hub. Root hub, functions, split bus operations – high speed (HS) and Full/Low speed (F/LS).

4.8 Concluding remarks

The three basic approaches of I/O transfers are discussed. The simplest technique is programmed I/O, in which the processor performs all the necessary control functions under direct control of program instructions. The second approach is based on the use of interrupts. The third I/O scheme involves DMA, the DMA controller transfers data between an I/O device and the main memory without continuous processor intervention. Access to memory is shared between the DMAQ controller and the processor.

Three popular interconnection standards – PCI, SCSI, USB are discussed. They represent different approaches that meet the needs of various devices and reflect the increasing importance of plug-and-ply features that increase user convenience.

References:

1. Carl Hamacher, Zvonko Vranesic, Safwat Zaky, Computer Organization, fifth edition, Mc-graw Hill higher education.

2. Computer Architecture and Organization, John P. Hayes, 3rd Edition, McGraw Hill.

3. Computer Organization and Architecture – William Stallings Sixth Edition, Pearson/PHI