B38DF
Computer Architecture and Embedded Systems

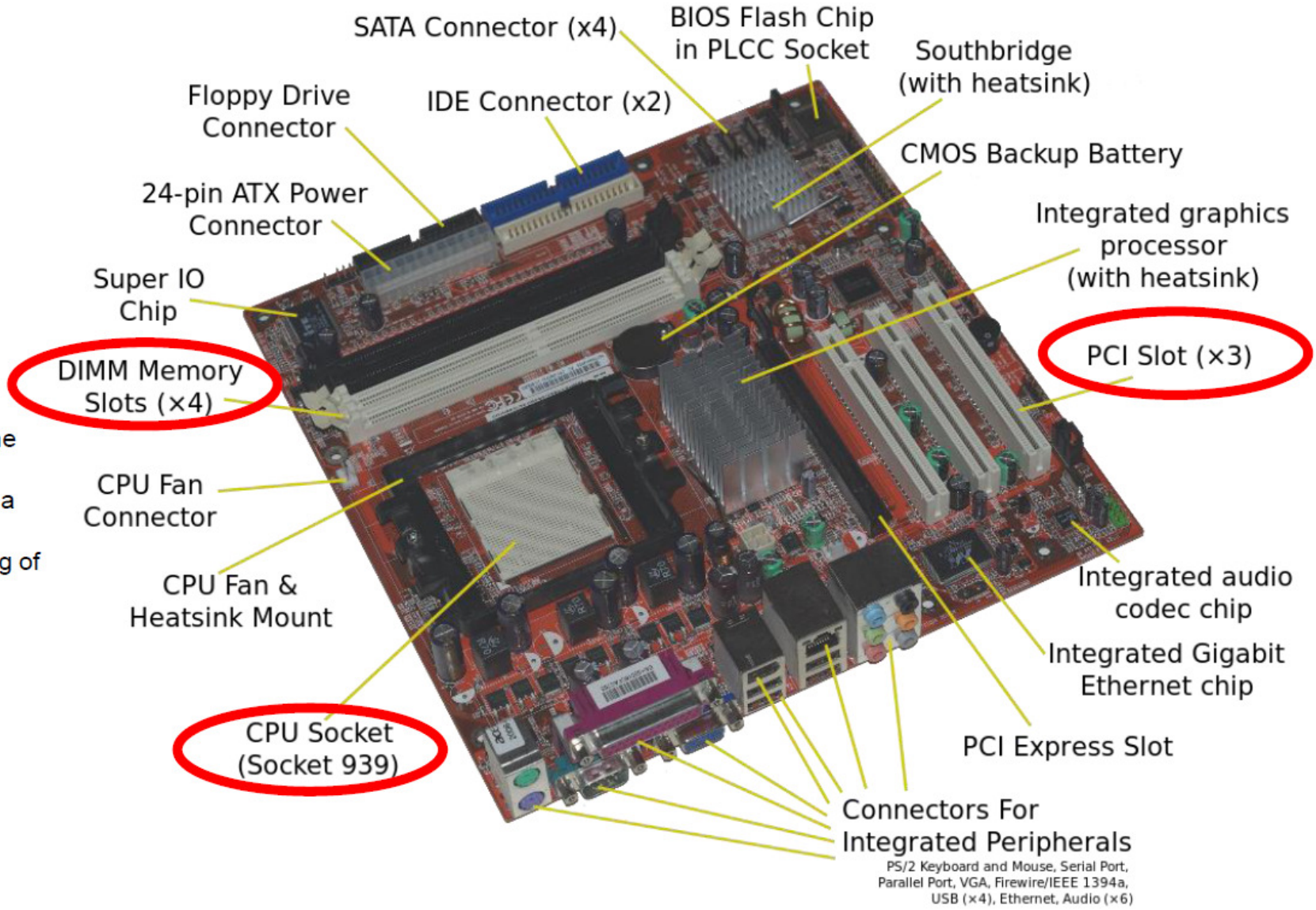
Alexander Belyaev

Heriot-Watt University
School of Engineering & Physical Sciences
Electrical, Electronic and Computer Engineering

E-mail: a.belyaev@hw.ac.uk
Office: EM2.29

Based on the slides provided by Dr. Mustafa Suphi Erden

Motherboard



DIMM: Dual Inline Memory Module; a type of packaging of **RAM**

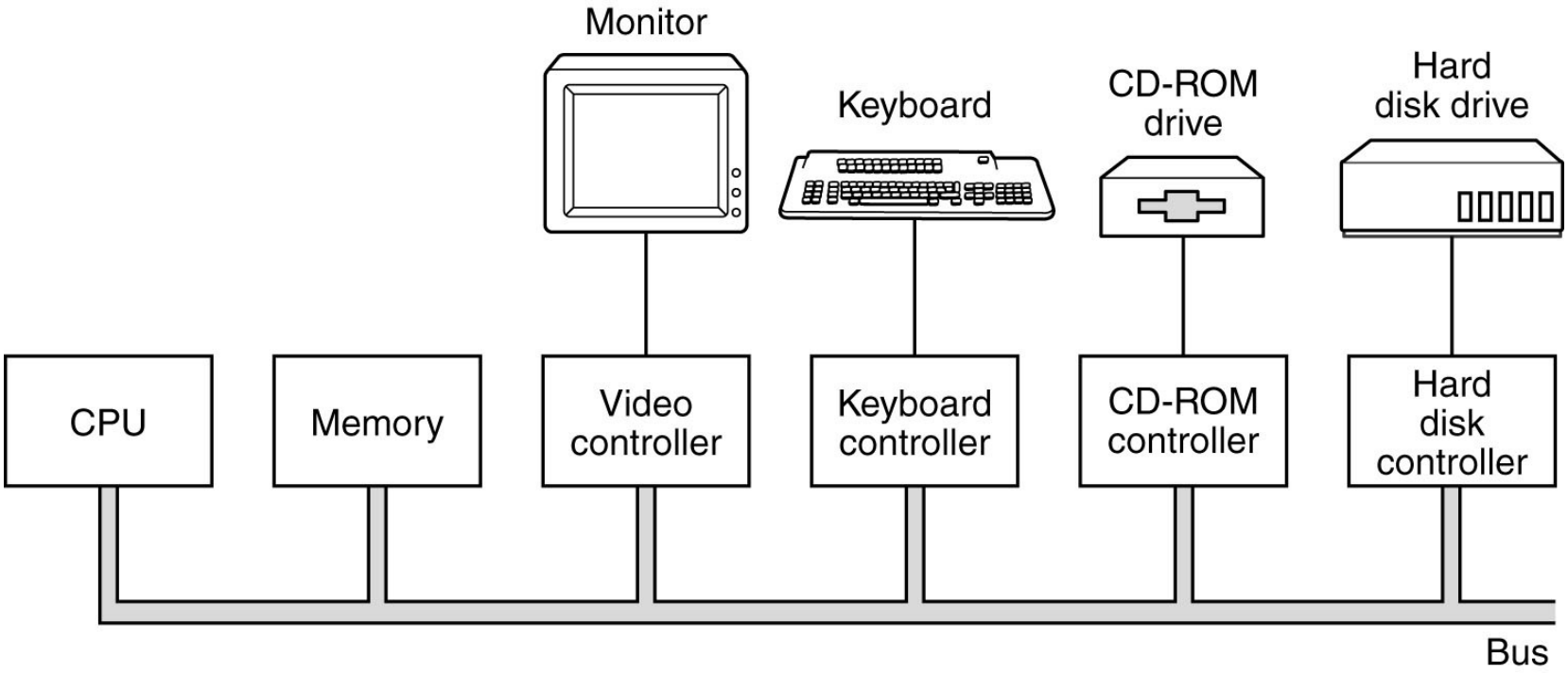
Source: Wikipedia

Motherboard

- A motherboard is the main **printed circuit board (PCB)** found in computers.
- It **holds and allows communication between crucial electronic components**, such as the central processing unit (CPU) and memory, and provides connectors for other peripherals.
- Motherboard specifically refers to a PCB **with expansion capability**
- The components attached to it often include peripherals, interface cards, and daughtercards: **sound cards, video cards, network cards, hard drives, or other forms of persistent storage**; **TV tuner cards, cards providing extra USB or FireWire slots** and a variety of other custom components.
- The motherboard also contains a **bus** etched along its length, and sockets into which the edge connectors of I/O boards can be inserted.

Input/Output (I/O) – Buses and Devices

Logical structure of a simple personal computer.



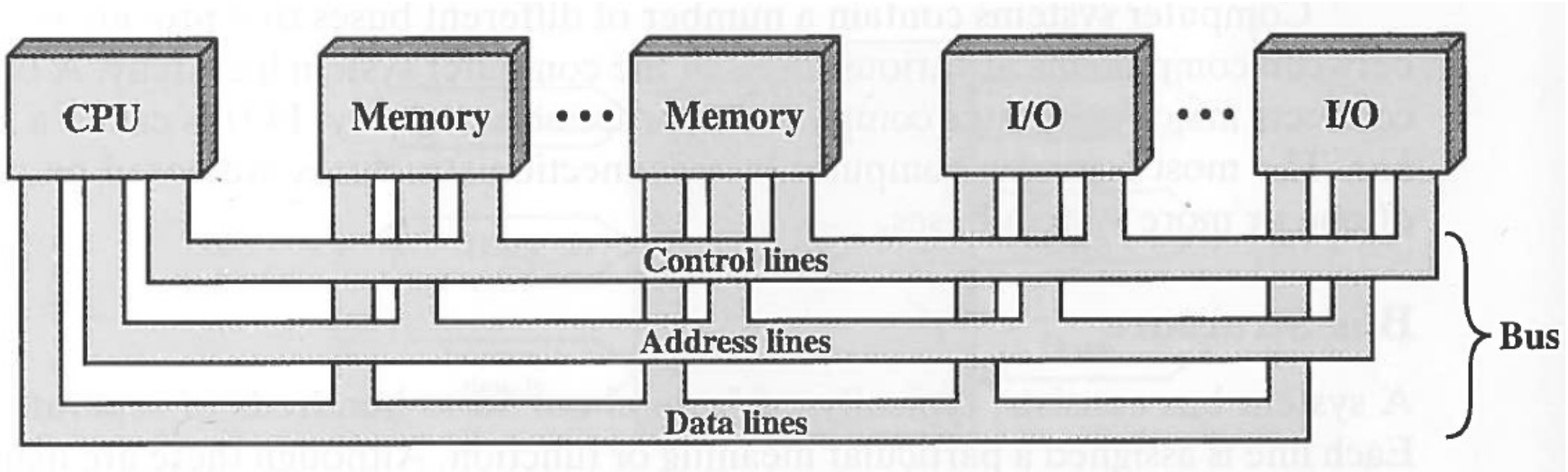
- A single bus system used to connect the CPU, memory and I/O devices.
- Most systems have two or more buses.

Input/Output (I/O) – Buses

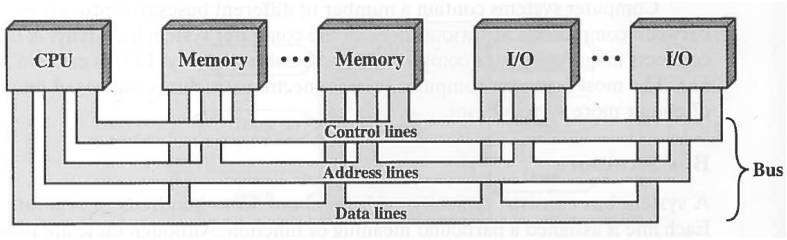
- A bus consists of multiple communication pathways, or **lines**.
- Each line is capable of transmitting signals representing binary 1 and binary 0.
- Several lines of a bus can be used to transmit binary digits simultaneously. E.g. 8-bit unit of data can be transmitted over eight bus lines.
- A system bus consists, typically, of from 50 to hundreds of separate lines.
- Multiple devices connect to the bus, and a signal transmitted by one device is available for reception by all other devices.
- Only one device at a time can successfully transmit through the bus.

Input/Output (I/O) – Bus Lines

- The bus lines can be classified into three functional groups.
- **Data lines:** They provide a path for moving data between system modules.
- Data bus may consist of from 32 to hundreds of separate lines, the number of lines being referred to as the width of the data bus.



Input/Output (I/O) – Bus Lines



- **Address lines:** They are used to designate the source or destination of the data on the data bus.
- The width of the address bus determines the maximum possible memory capacity of the system
- **Control lines**: They are used to control the access to and the use of the data and address lines.
- Typically control lines pass the following control signals:

Memory write: causes data on the bus to be written into the addressed location.

Memory read: causes data from the addressed location to be placed on the bus

I/O write: causes data on the bus to be output to the addressed I/O port

I/O read: causes data from the addressed I/O port to be placed on the bus

Transfer ACK: indicates that data have been accepted from or placed on the bus

Bus request: indicates that a module needs to gain control of the bus

Bus grant: indicates that a requesting module has been granted control of the bus

Interrupt request: indicates that an interrupt is pending

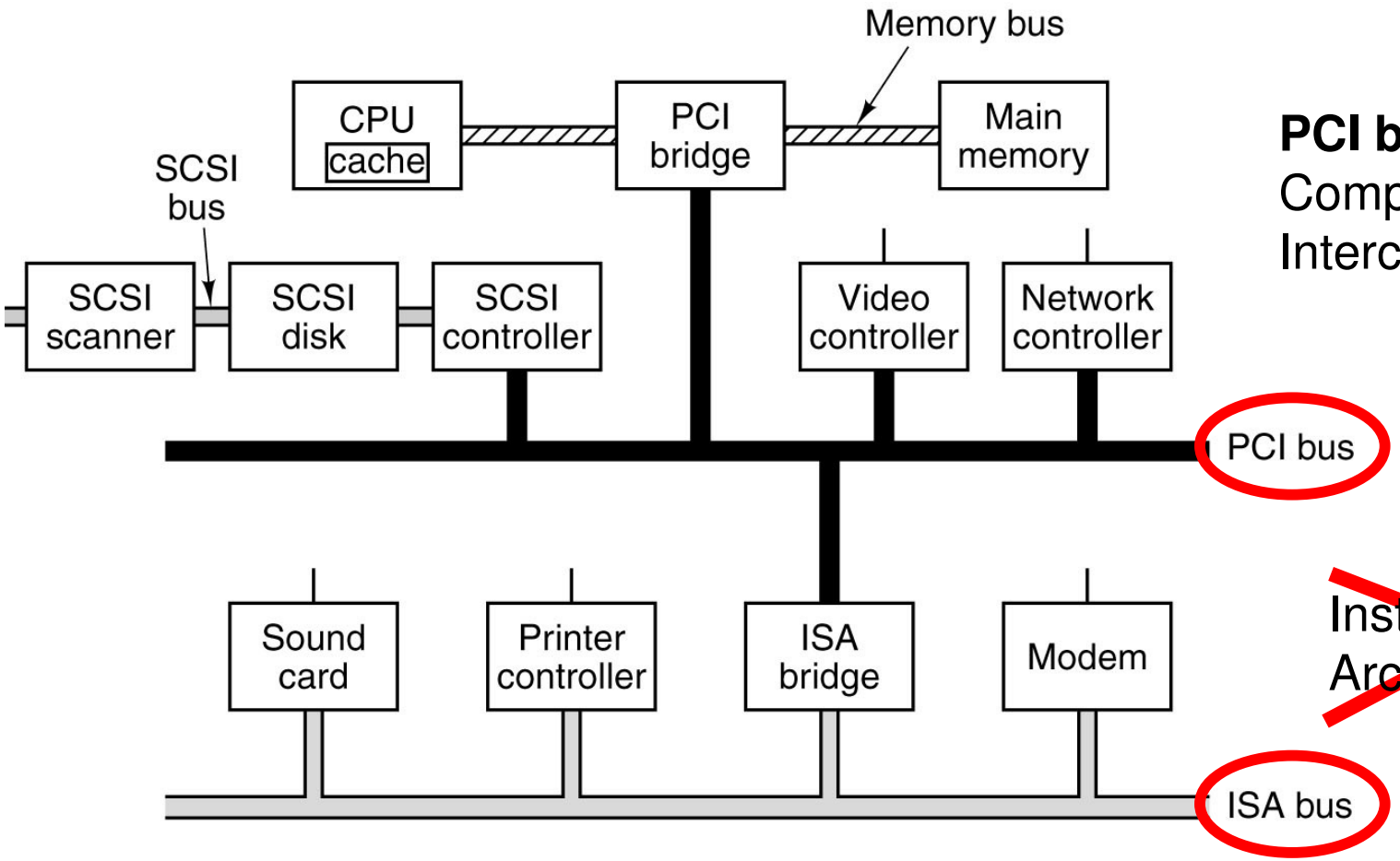
Interrupt ACK: acknowledges that the pending interrupt has been recognized

Clock: is used to synchronize operations

Reset: initializes all modules

ISA and PCI Buses

A typical modern PC with a PCI bus and an ISA bus.

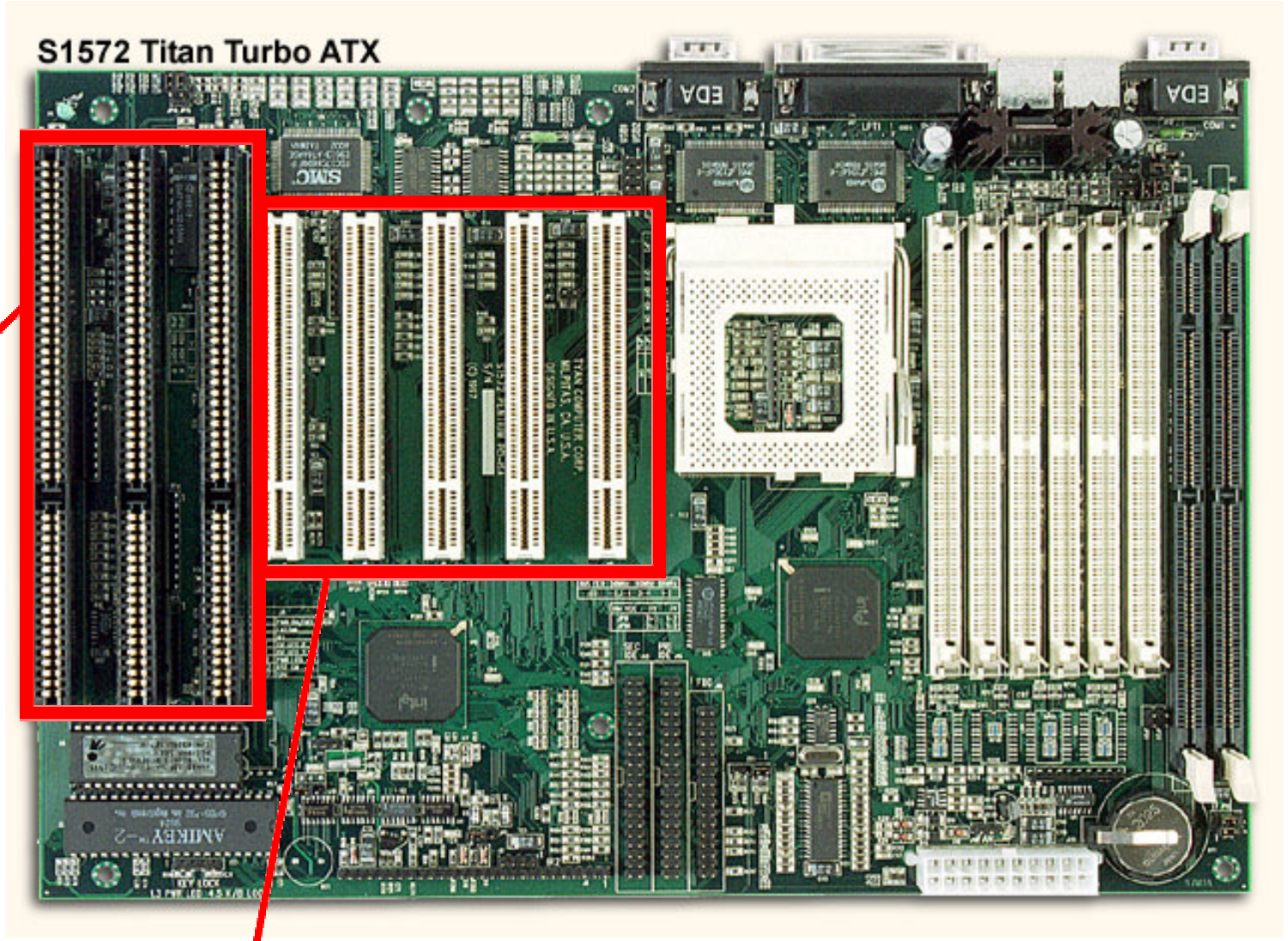


PCI bus: Peripheral Component Interconnect (PCI)

~~Instruction Set Architecture (ISA)~~

ISA bus: Industry Standard Architecture (ISA) is a 16-bit internal bus.

ISA and PCI Buses



ISA

- **ISA bus**: Industry Standard Architecture (ISA) is a 16-bit internal bus.
- The 16-bit ISA bus was also used with 32-bit processors for several years.
- An attempt to extend it to 32 bits, called Extended Industry Standard Architecture (EISA), was not very successful.
- Later buses such as PCI were used instead, often along with ISA slots on the same mainboard.

Industry Standard Architecture

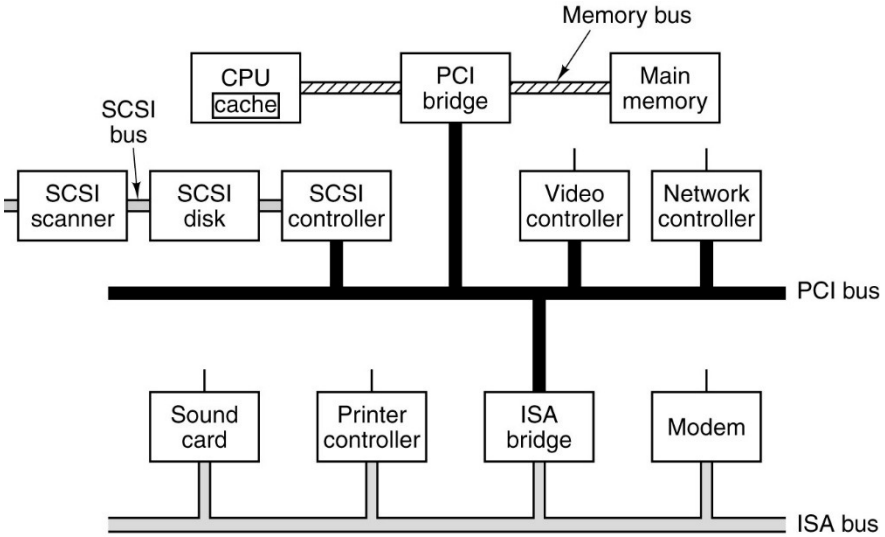
One 8-bit and five 16-bit ISA slots on a motherboard

Year created	1981; 35 years ago
Created by	IBM
Superseded by	PCI (1993)
Width in bits	8 or 16

Source: Wikipedia

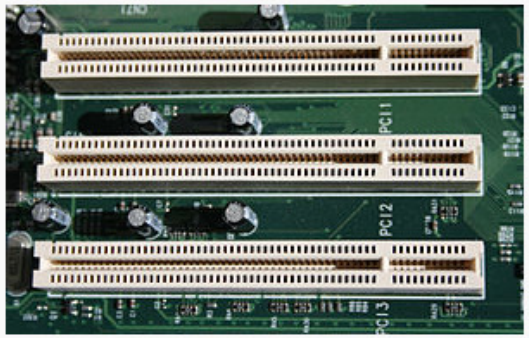
PCI

- **PCI bus**: Peripheral Component Interconnect (PCI)



- CPU talks to a memory controller over a dedicated high-speed connection.
- CPU-memory traffic does not go over the PCI bus.
- PCI bus has a bridge to the ISA bus, so that ISA controller and their devices can still be used.

PCI Local Bus



Three 5-volt 32-bit PCI expansion slots on a motherboard (PC bracket on left side)

Year created	June 22, 1992; 23 years ago[1]
Created by	Intel
Supersedes	ISA, EISA, MCA, VLB
Superseded by	PCI Express (2004)
Width in bits	32 or 64
Speed	133 MB/s (32-bit at 33 MHz – the standard configuration) 266 MB/s (32-bit at 66 MHz or 64-bit at 33 MHz) 533 MB/s (64-bit at 66 MHz)

Source: Wikipedia

Buss Signal Control

- The bus is not used only by **the I/O controllers**, but **also by the CPU** for fetching instructions and data.
- What happens if the CPU and I/O controller want to use the bus at the same time?
- **Bus arbiter**: A chip that decides who uses the bus next and that manages the order of bus usage.
- In general I/O devices are given preference over the CPU, because
  - Disks and other moving devices cannot be stopped, and forcing them to wait would result in loss of data.
- **Cycle Stealing:** When no I/O is in progress, the CPU can have all the bus cycles for itself to reference memory. However, when some I/O device is also running, that device will request and be granted the bus when it needs it. This process is called cycle stealing and it slows down the computer.

Source: Wikipedia

Input/Output – Devices

- Each I/O device consists of two parts:

➢ **Controller**

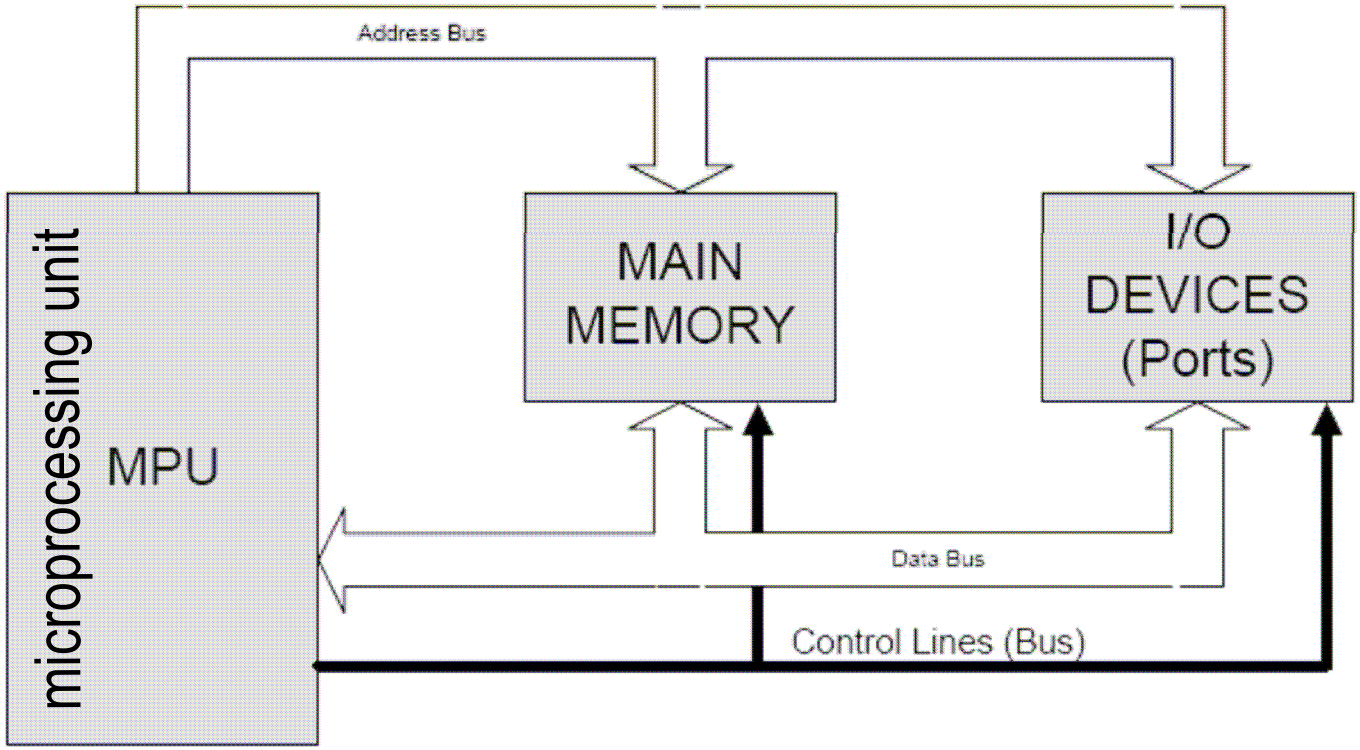
➢ **Device itself**

- **Controller:**

  Controls its I/O device and handles bus access to it.

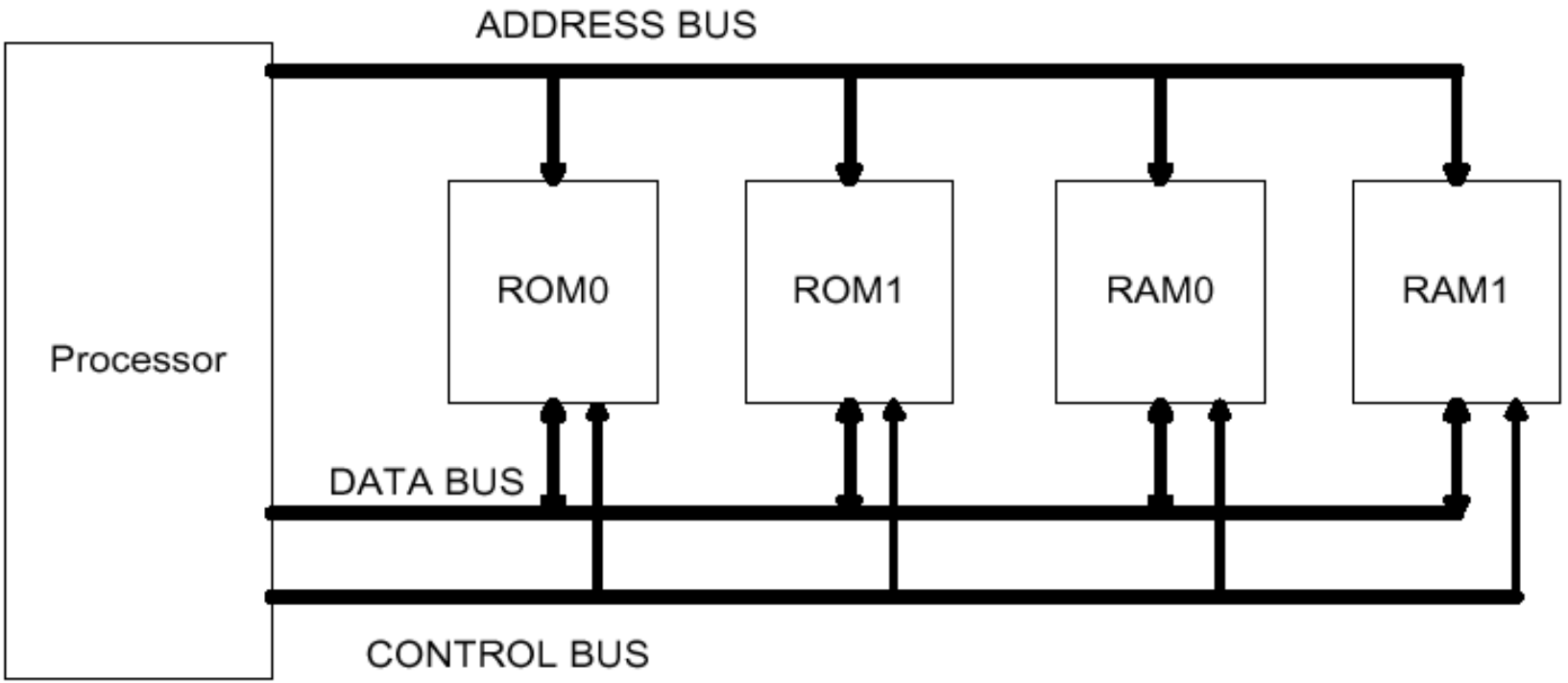
e.g. When a program wants data from the disk, it gives a command to the disk controller, which then issues seeks and other commands to the drive. When the proper track and sector have been located, the drive begins outputting the data as a serial bit stream to the controller. It is the job of the controller to break the bit stream up into units, and write each unit into memory, as it is assembled. A unit is typically one or more words.

Simplified Block Diagram of a Microcomputer



- **Data lines**
- **Address lines**
- **Control lines**

Processor with multiple memory devices

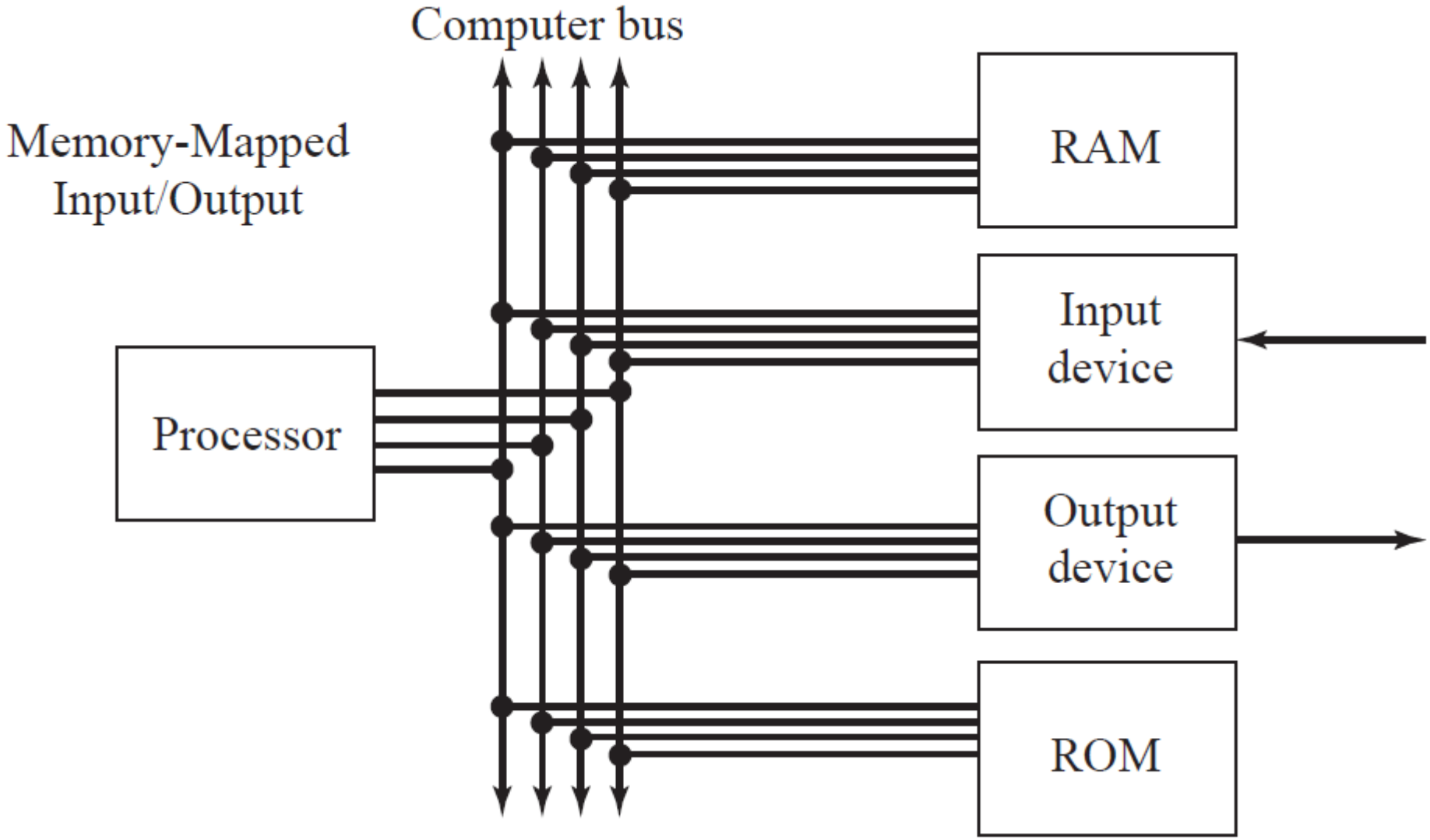


Microprocessor System
with ROM and RAM

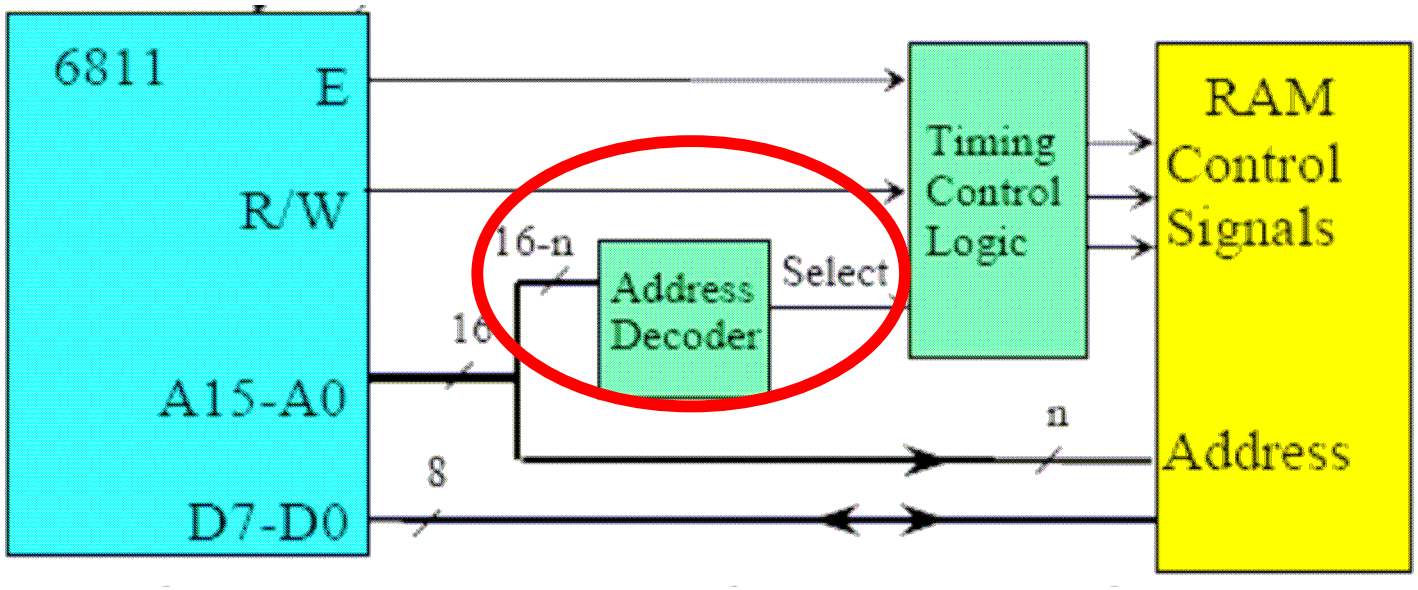
Device Selection and Data Buses

- A PC board may have many memory devices, all attached to the same data bus
- When the processor reads data from the bus, it is essential that only one device drives data onto the bus
- The other memories must be electrically disconnected from the bus while the selected device drives it

Microcomputer Architecture with Memory Mapped I/O



Example Interface

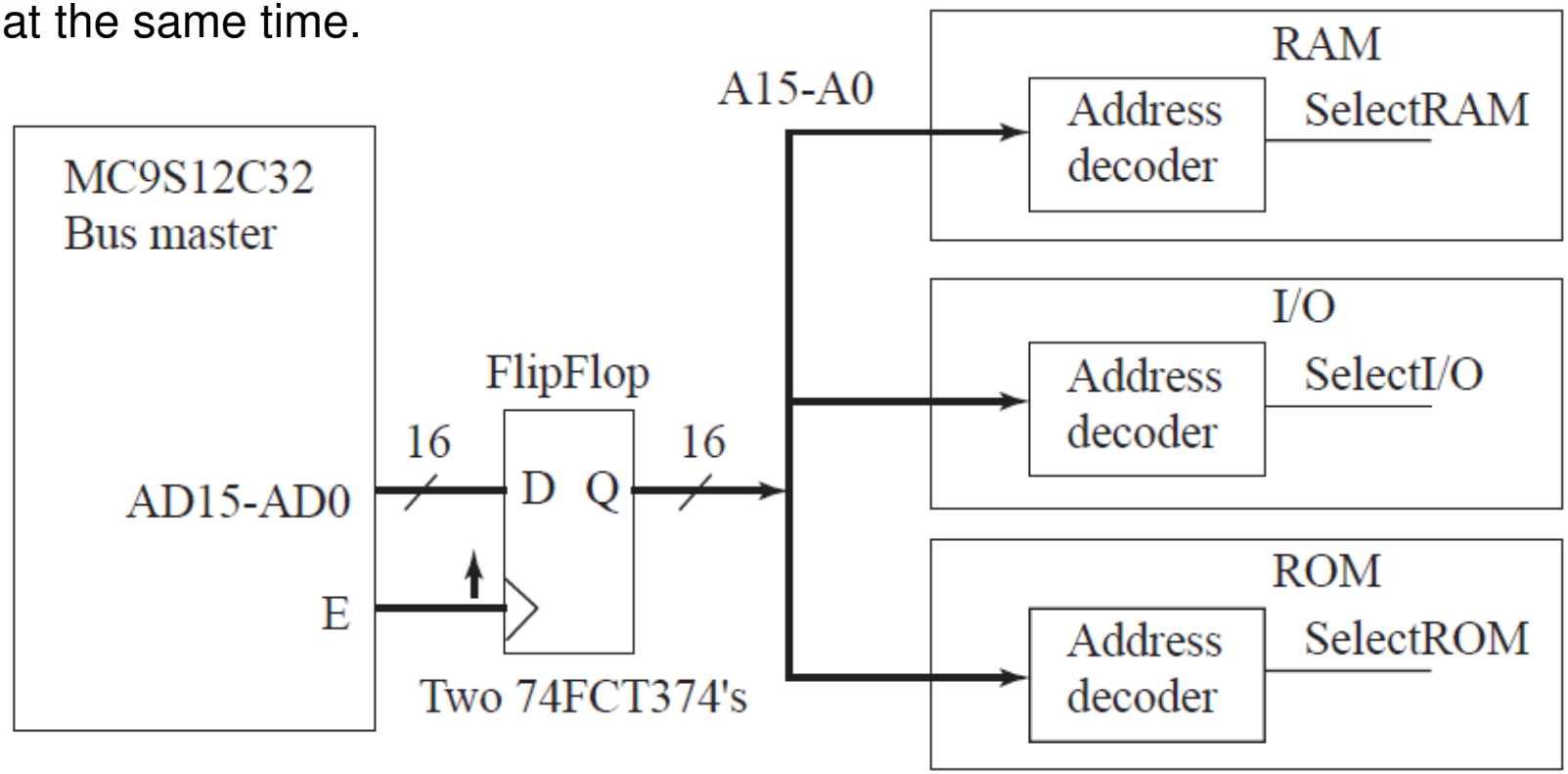


Select is true if address matches the slave address.

Select	R/W	Function	Rationale
0	0	Off	Because the address is incorrect
0	1	Off	Because the address is incorrect
1	0	Write	Data flows from 6811 to slave
1	1	Read	Data flows from slave to the 6811

Address Decoding

The address decoder is usually located in each slave interface. The purpose of the address decoder is to monitor the address lines from the bus master (the processor) and determine whether or not the slave has been selected to communicate in the current cycle. Each slave has its own address decoder, uniquely designed to select the addresses intended for that device. Care must be taken to avoid having two devices driving the data bus at the same time.



If two devices have address decoders with overlapping addresses, then a write cycle will store data at both devices, and during a read cycle the data from the two devices will collide, possibly causing damage to one or both devices.

Full-Address Decoding

Full-address decoding is where the slave is selected if and only if the slave's address appears on the bus. In positive logic,

$$\text{Select} \begin{cases} = 1 & \text{if the slave address appears on the address bus} \\ = 0 & \text{if the slave address does not appear on the address bus} \end{cases}$$

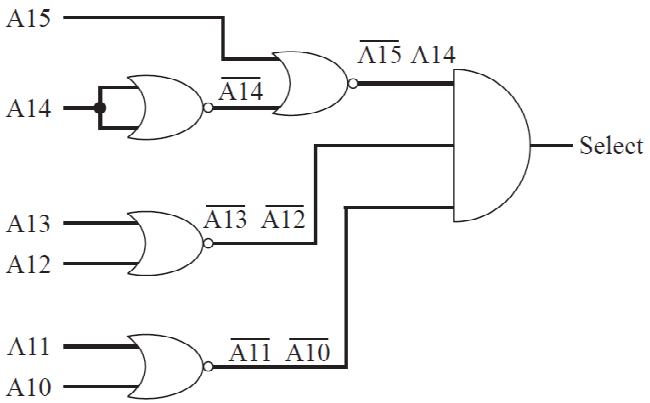
Example. Design a fully decoded positive logic select signal for a 1 KB RAM at $4000–$43FF. ($ = 0x → hexadecimal)

Step 1. Write specified address using **0**,**1**,**X**: **0100,00XX,XXXX,XXXX**

Step 2. Write the equation using all **0**s and **1**s. A **0** translates into the complement of the address bit, and a **1** translates directly into the address bit.

$$\text{Select} = \overline{A15} \cdot A14 \cdot \overline{A13} \cdot \overline{A12} \cdot \overline{A11} \cdot \overline{A10}$$

Step 3. Build circuit:



Minimal-Cost Address Decoding

Minimal-cost address decoding is a mechanism to simplify the decoding logic by introducing don't care states for unspecified addresses.

Address	Select
Matches our device	True
Matches other specified device	False
Unspecified address	Don't care

Example. Design four minimal-cost select signals in positive logic. The 4 KB RAM is at $0000 to $0FFF, the input device is at $5000, the output device is at $5001, and the 16 KB ROM is at $C000 to $FFFF.

Solution

Step 1. Write out the addresses in binary for all devices. Include all specified addresses in the computer, not just the devices that we are designing.

RAM	0000 , XXXX , XXXX , XXXX
Input	0101 , 0000 , 0000 , 0000
Output	0101 , 0000 , 0000 , 0001
ROM	11XX , XXXX , XXXX , XXXX

Minimal-Cost Address Decoding

Step 1. Write out the addresses in binary for all devices. Include all specified addresses in the computer, not just the devices that we are designing.

RAM	0000 , XXXX , XXXX , XXXX
Input	0101 , 0000 , 0000 , 0000
Output	0101 , 0000 , 0000 , 0001
ROM	11XX , XXXX , XXXX , XXXX

Step 2. Choose as many address lines that are required to differentiate between the devices. Consider the different devices in a pairwise fashion. If the decoder for only one device is being built, then only the addresses that differentiate that device from the others are required. In this example, we choose A15, A14, A0. The address bits A15, A12, and A0 also could have been used to differentiate the devices.

Step 3. Draw a Karnaugh map for each device.

a. Put a true for addresses specified by that device

b. Put a false for other devices

c. Put an “X” for unspecified addresses

Step 4. Minimize using Karnaugh maps and determine equations

Step 5. Build circuit

Minimal-Cost Address Decoding

RAM	0000 , XXXX , XXXX , XXXX
Input	0101 , 0000 , 0000 , 0000
Output	0101 , 0000 , 0000 , 0001
ROM	11XX , XXXX , XXXX , XXXX

Step 2. In this example we choose A15, A14, A0.

The address bits A15, A12, and A0 also could have been used to differentiate the devices.

Step 3. Draw a Karnaugh map for each device.

a. Put a true for addresses specified by that device

b. Put a false for other devices

c. Put an "X" for unspecified addresses

RAM

A15	A14 \ A0	0	1
0	0	1	1
0	1	0	0
1	1	0	0
1	0	X	X

ROM

A15	A14 \ A0	0	1
0	0	0	0
0	1	0	0
1	1	1	1
1	0	X	X

Input

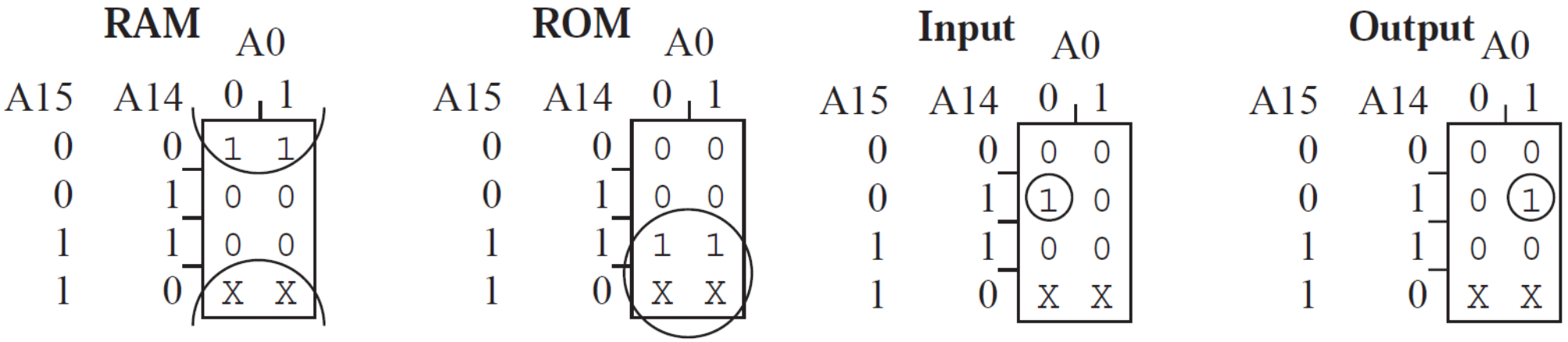
A15	A14 \ A0	0	1
0	0	0	0
0	1	1	0
1	1	0	0
1	0	X	X

Output

A15	A14 \ A0	0	1
0	0	0	0
0	1	0	1
1	1	0	0
1	0	X	X

Minimal-Cost Address Decoding

RAM	0000 , XXXX , XXXX , XXXX
Input	0101 , 0000 , 0000 , 0000
Output	0101 , 0000 , 0000 , 0001
ROM	11XX , XXXX , XXXX , XXXX



Step 4. Minimize using Karnaugh maps and determine equations

$$\text{RAMSelect} = \overline{\text{A14}}$$

$$\text{InputSelect} = \overline{\text{A15}} \cdot \text{A14} \cdot \overline{\text{A0}}$$

$$\text{ROMSelect} = \text{A15}$$

$$\text{OutputSelect} = \overline{\text{A15}} \cdot \text{A14} \cdot \text{A0}$$

Minimal-Cost Address Decoding

RAM	000, XXXX, XXXX, XXXX
Input	0101, 0000, 0000, 0000
Output	0101, 0000, 0000, 0001
ROM	11XX, XXXX, XXXX, XXXX

Step 4. Minimize using Karnaugh maps and determine equations

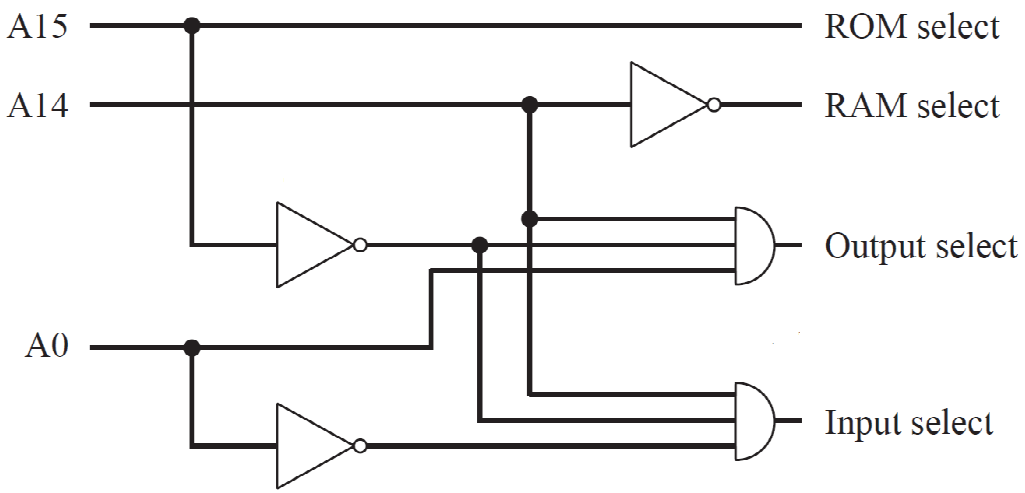
$$\text{RAMSelect} = \overline{A14}$$

$$\text{InputSelect} = \overline{A15} \cdot A14 \cdot \overline{A0}$$

$$\text{ROMSelect} = A15$$

$$\text{OutputSelect} = \overline{A15} \cdot A14 \cdot A0$$

Step 5. Build circuit



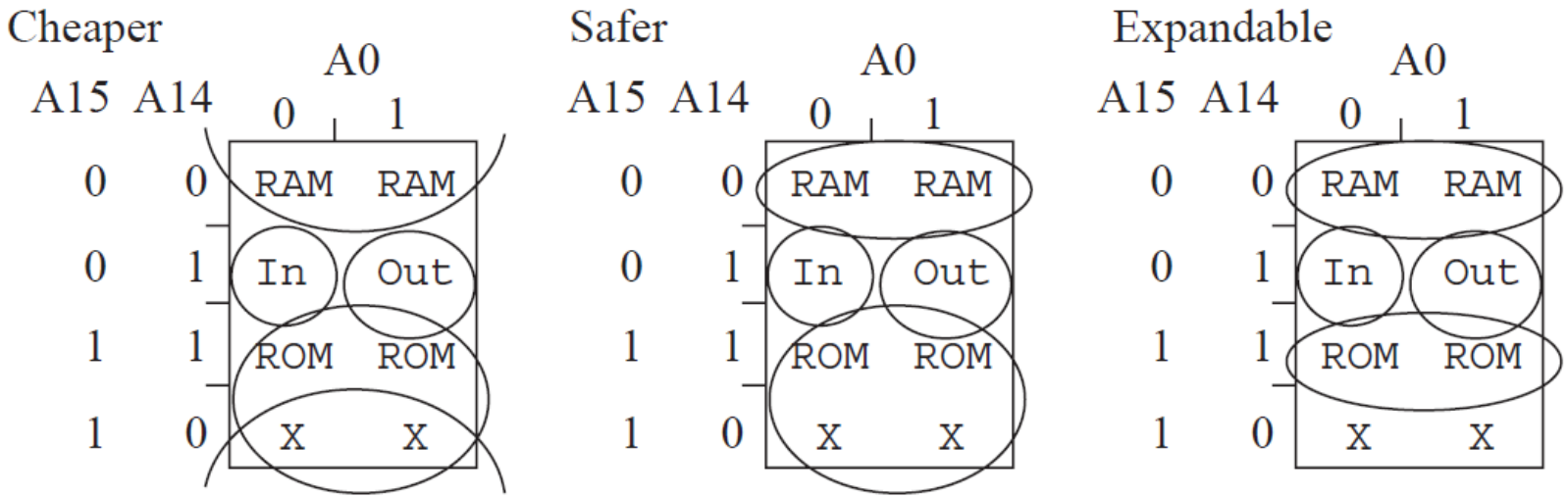
Minimal-Cost Address Decoding

Performance tip: Use minimal-cost decoding on systems where cost and speed are more important than future expansion.

Observation: If there are no unspecified addresses, then minimal-cost decoders will be the same as the full-address decoders.

Common error: If one does not select enough address lines to differentiate between the devices, then some addresses may incorrectly select more than one device.

Observation: If one selects too many address lines, then the Karnaugh map will be harder to draw, but the resulting solution should be the same.



Another example

Design a minimal cost positive-logic address decoder for “Your Device” in the following system.

```
RAM              $6000 - $6FFF,
Your Device      $7000 - $7FFF,
ROM              $E000 - $FFFF
```

Show your design steps, the chip enable Boolean equation and the simple circuit.

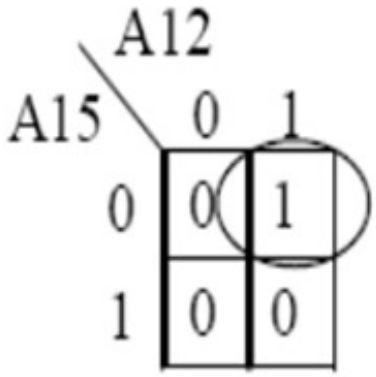
Solution:

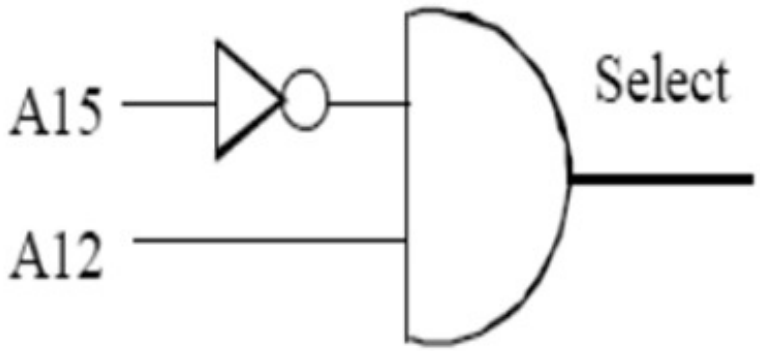
RAM	$6000-$6FFF	0110,XXXX,XXXX,XXXX	needs A12
YourDevice	$7000-$7FFF	0111,XXXX,XXXX,XXXX	
ROM	$E000-$FFFF	111X,XXXX,XXXX,XXXX	needs A15

From the memory address lines,

The Chip Enable for "Your Device" Should be

Chip Enable $_{\text{(Your Device)}}$ **= NOT (A15) * A12**





Full Address Decoding

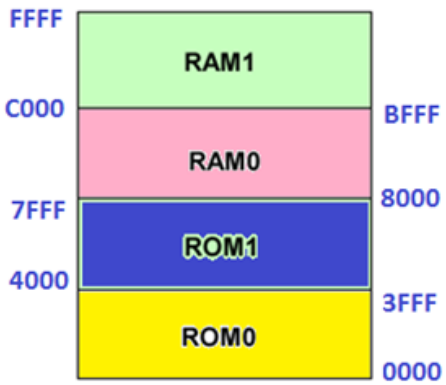
- Need external decoding hardware to ensure that only one device is accessed at any one time
- Simple techniques enable the Chip Enable of just one device, based on the address bus contents

  Implement this system, consisting of 4 x (16K x 8) memories:

  - ROM0 - 0000h - 3FFFh
  - ROM1 - 4000h - 7FFFh
  - RAM0 - 8000h - BFFFh
  - RAM1 - C000h - FFFFh

(…h = 0x… → hexadecimal)

Simple Address Decoding - Example

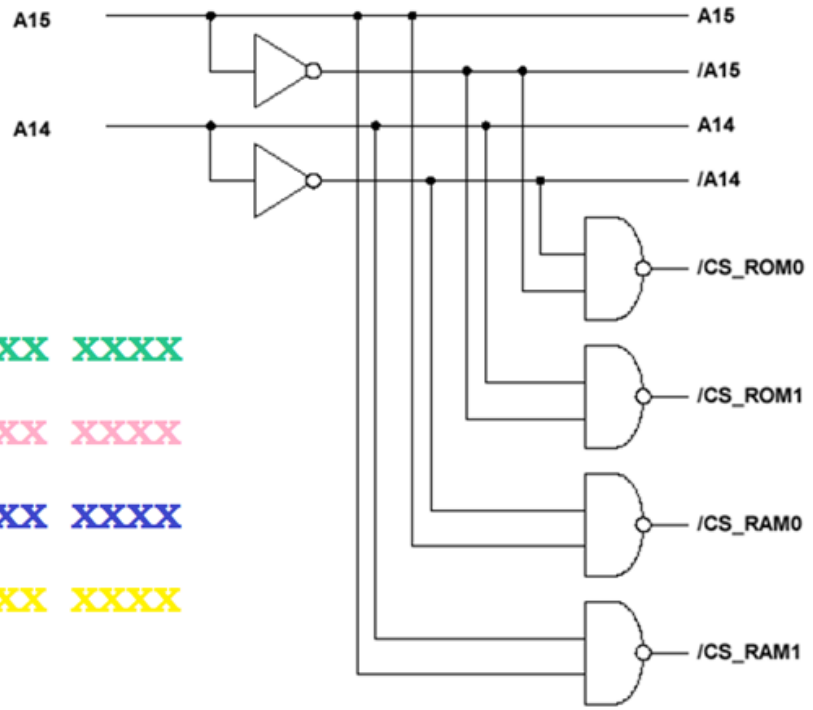


RAM 1 : SELECT = A15 * A14

RAM 0 : SELECT = A15 * $\overline{A14}$

ROM 1 : SELECT = $\overline{A15}$ * A14

ROM 0 : SELECT = $\overline{A15}$ * $\overline{A14}$



RAM 1 :	$C000 - $FFFF	11XX XXXX XXXX XXXX
RAM 0 :	$8000 - $BFFF	10XX XXXX XXXX XXXX
ROM 1 :	$4000 - $7FFF	01XX XXXX XXXX XXXX
ROM 0 :	$0000 - $3FFF	00XX XXXX XXXX XXXX

Problem 1

Design a **fully-decoded** positive-logic address decoder for *Your Device* in the following system. (Design just the decoder for *Your Device*, not all of them). The inputs are A15, A14, …A0 and the output is **Select**. Positive logic means **Select**=1 when *Your Device* should be activated, and **Select**=0 when *Your Device* is deactivated. Memory addresses are specified in hexadecimal numbers.

RAM	**$0000–$0FFF**
Your Device	**$4000–$5FFF**
I/O ports	**$8010–$801F**
ROM	**$C000–$FFFF**

Solution to Problem 1

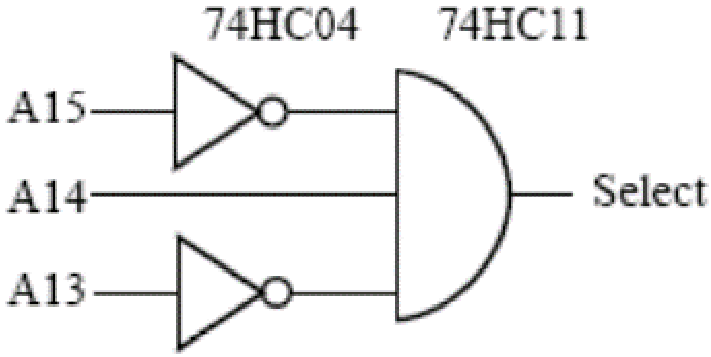
Design a **fully-decoded** positive-logic address decoder for *Your Device* in the following system. (Design just the decoder for *Your Device*, not all of them). The inputs are A15, A14, …A0 and the output is **Select**. Positive logic means **Select**=1 when *Your Device* should be activated, and **Select**=0 when *Your Device* is deactivated. Memory addresses are specified in hexadecimal numbers.

RAM	**$0000–$0FFF**
Your Device	**$4000–$5FFF**
I/O ports	**$8010–$801F**
ROM	**$C000–$FFFF**

Write **$4000–$5FFF** as **010X,XXXX,XXXX,XXXX**

Give logic equation directly using all 0's and 1's,

Select = not (A15) * A14 * not (A13)



Problem 1b

Design a **minimal-cost** positive-logic address decoder for *Your Device* in the following system. (Design just the decoder for *Your Device*, not all of them). The inputs are A15, A14, …A0 and the output is **Select**.

RAM	**$0000–$0FFF**
Your Device	**$4000–$5FFF**
I/O ports	**$8010–$801F**
ROM	**$C000–$FFFF**

Write **$0000–$0FFF** as **0000,XXXX,XXXX,XXXX**
Write **$4000–$5FFF** as **010X,XXXX,XXXX,XXXX**
Write **$8010–$801F** as **1000,0000,0001,XXXX**
Write **$C000–$FFFF** as **11XX,XXXX,XXXX,XXXX**

Select = not (A15) * A14

Problem 2

Using logic gates, design a minimal cost, positive-logic address decoder for a component *Your Device* in a small embedded microprocessor system. You do not have to design all four address decoders, just the one for *Your Device*.

I/O Ports	**$5800–$58FF**
RAM	**$D000–$D3FF**
Your Device	**$D800–$DFFF**
ROM	**$E000–$FFFF**

Solution to Problem 2

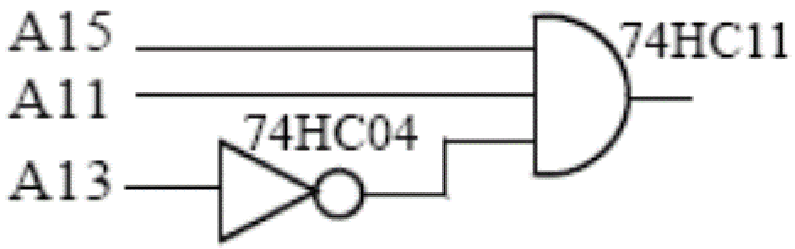
Using logic gates, design a minimal cost, positive-logic address decoder for a component *Your Device* in a small embedded microprocessor system. You do not have to design all four address decoders, just the one for *Your Device*.

```
I/O Ports     $5800–$58FF
RAM           $D000–$D3FF
Your Device   $D800–$DFFF
ROM           $E000–$FFFF

I/O Ports     $5800–$58FF   0101,1000,XXXX,XXXX
RAM           $D000–$D3FF   1101,00XX,XXXX,XXXX
Your Device   $D800–$DFFF   1101,1XXX,XXXX,XXXX
ROM           $E000–$FFFF   111X,XXXX,XXXX,XXXX
```

Needs A15, A13, A11 for decoding each memory devices.

Your Device Select = A15 * not(A13) * A11



Problem 3

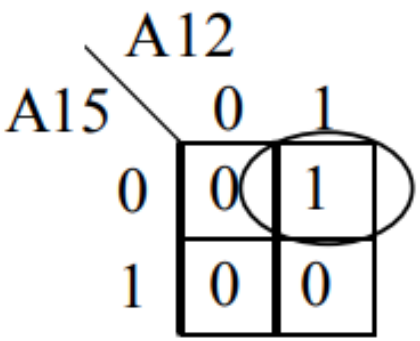
Design a minimal-cost positive-logic address decoder for YourDevice in the following system. RAM $6000-$6FFF, YourDevice $7000-$7FFF, ROM $E000-$FFFF. Show 1) design steps, 2) equation, 3) circuit.

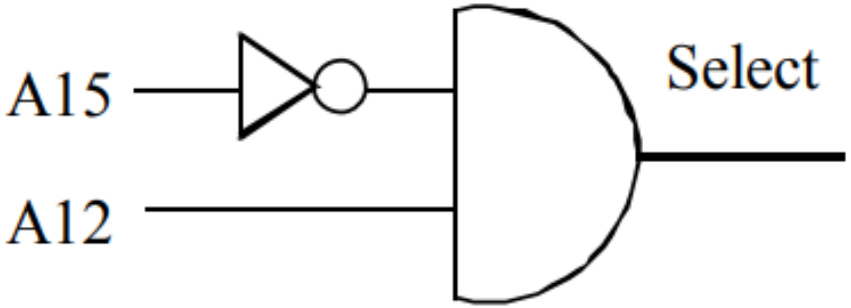
Solution to Problem 3

Design a minimal-cost positive-logic address decoder for YourDevice in the following system. RAM $6000-$6FFF, YourDevice $7000-$7FFF, ROM $E000-$FFFF. Show 1) design steps, 2) equation, 3) circuit.

RAM	$6000-$6FFF	`0110,xxxx,xxxx,xxxx`	needs A12
YourDevice	$7000-$7FFF	`0111,xxxx,xxxx,xxxx`	
ROM	$E000-$FFFF	`111x,xxxx,xxxx,xxxx`	needs A15

Select = not(A15)*A12





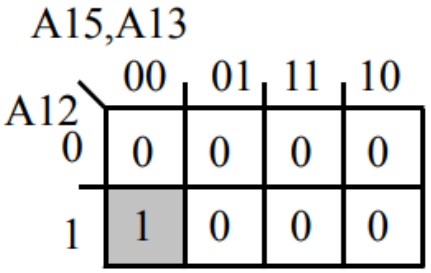
Problem 4

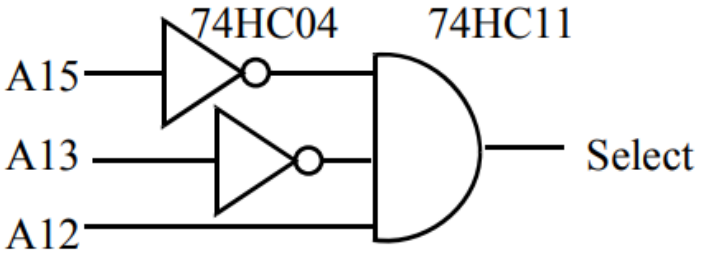
Design a minimal-cost positive-logic address decoder for YourDevice in the following system. The inputs are A15, A14, …A0 and the output is Select. Positive logic means Select=1 when YourDevice should be activated, and Select=0 when YourDevice is deactivated. RAM $4000-$47FF, YourDevice $5000-$57FF, ROM $6000-$7FFF, ROM $C000-$FFFF. Show 1) design steps, 2) logic equation, 3) digital logic circuit.

Solution to Problem 4

Design a minimal-cost positive-logic address decoder for YourDevice in the following system. The inputs are A15, A14, …A0 and the output is Select. Positive logic means Select=1 when YourDevice should be activated, and Select=0 when YourDevice is deactivated. RAM $4000-$47FF, YourDevice $5000-$57FF, ROM $6000-$7FFF, ROM $C000-$FFFF. Show 1) design steps, 2) logic equation, 3) digital logic circuit.

RAM	$4000-$47FF	**0100,0xxx,xxxx,xxxx**	must choose A12
YourDevice	$5000-$57FF	**0101,0xxx,xxxx,xxxx**	
ROM	$6000-$7FFF	**011x,xxxx,xxxx,xxxx**	must choose A13
ROM	$C000-$FFFF	**11xx,xxxx,xxxx,xxxx**	must choose A15





Select = not(A15)*not(A13)*A12