Digital microsystems design

Marius Marcu

2020

Objectives

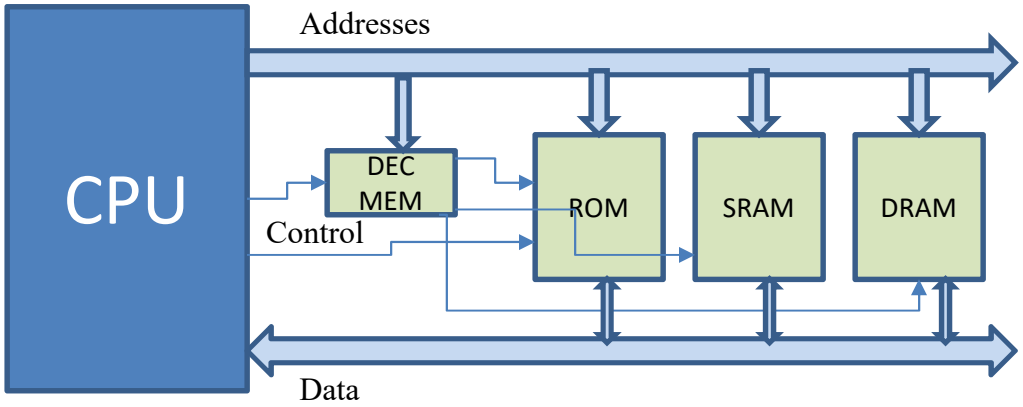
- Specific objectives
  - Internal architecture and external interfaces of x86 processors and their functional behavior
    - Memory connectivity

Outline

- Microprocessor bus
  - Amplifying bus lines
  - Demultiplexing bus lines
  - Decoding address lines
  - Data transfers size
- Memory decoding
- Summary

Buses

- There are several aspects to be addressed when connecting memory and I/O to microprocessor buses:
  - Signal amplifying
  - Signal demultiplexing
  - Slave selection (decoding)
  - Data transfers sizes



Buses

- Amplifying bus lines
  - Fan-out of the processor
  - Example:
    - How many TTL gates the given microprocessor can drive?

Circuit type	I_{OH}	I_{OL}	I_{IH}	I_{IL}
Microprocessor	250 μA	1,8 mA	10 μA	10 μA
SRAM	1 mA	2,1 mA	2 μA	2 μA
Parallel I/O controller	400 μA	2,5 mA	10 μA	10 μA
CMOS gates	8 mA	8 mA	1 μA	1 μA
TTL gates	800 μA	16 mA	40 μA	1,6 mA

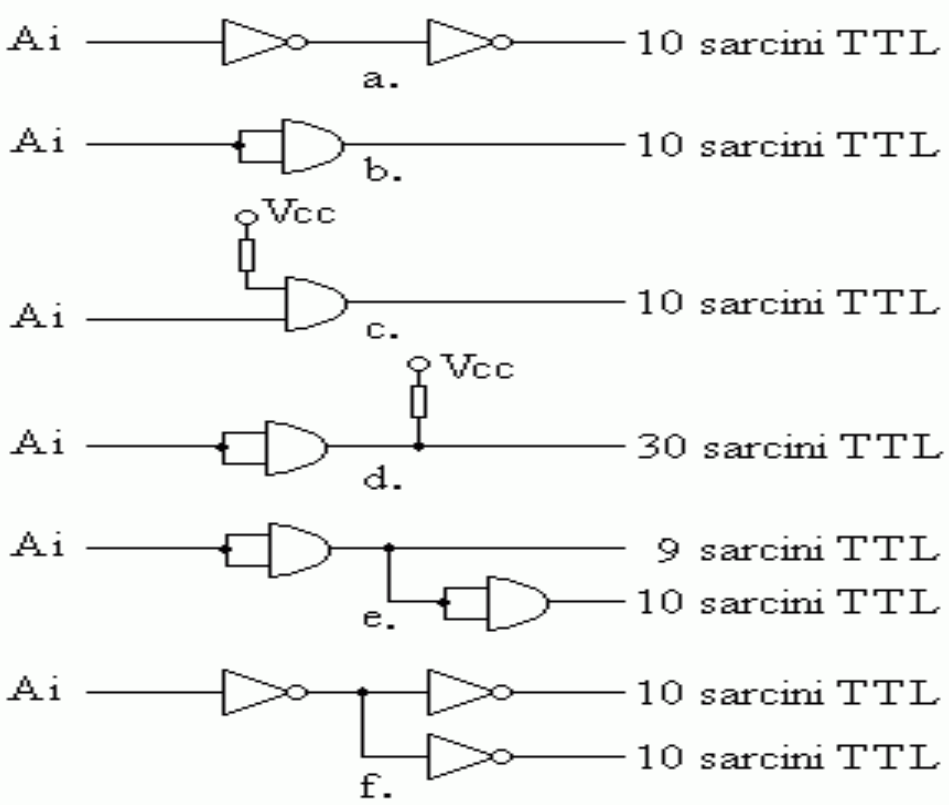
Buses

- Amplifying bus lines
  - Example:
    - 8086 processor can drive maximum one TTL gate
    - How many CMOS gates 8086 processor can drive?

Circuit type	I_{OH}	I_{OL}	I_{IH}	I_{IL}
Microprocessor	250 μA	1,8 mA	10 μA	10 μA
SRAM	1 mA	2,1 mA	2 μA	2 μA
Parallel I/O controller	400 μA	2,5 mA	10 μA	10 μA
CMOS gates	8 mA	8 mA	1 μA	1 μA
TTL gates	800 μA	16 mA	40 μA	1,6 mA

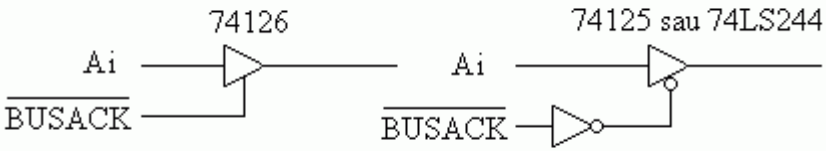
Buses

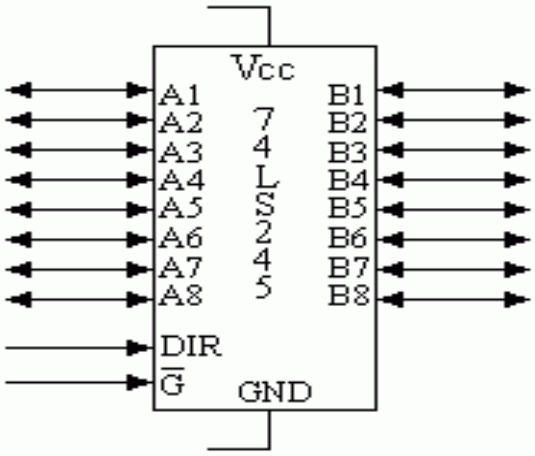
- Amplifying output lines:



Buses

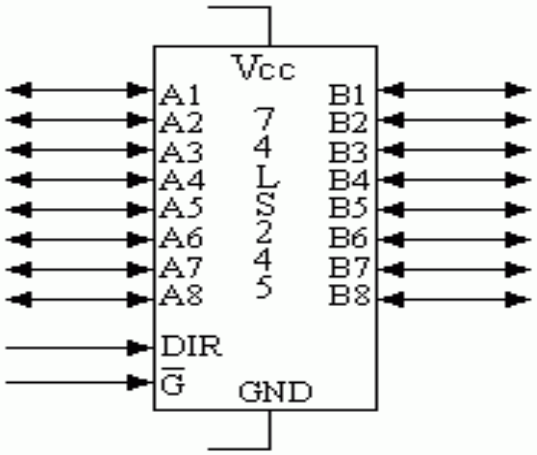
- Amplifying tristate lines

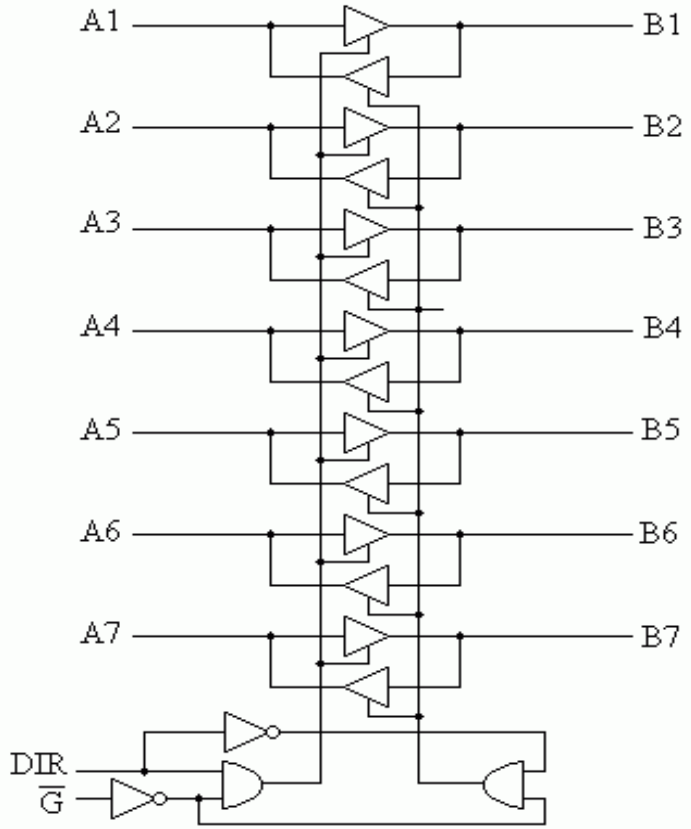




Buses

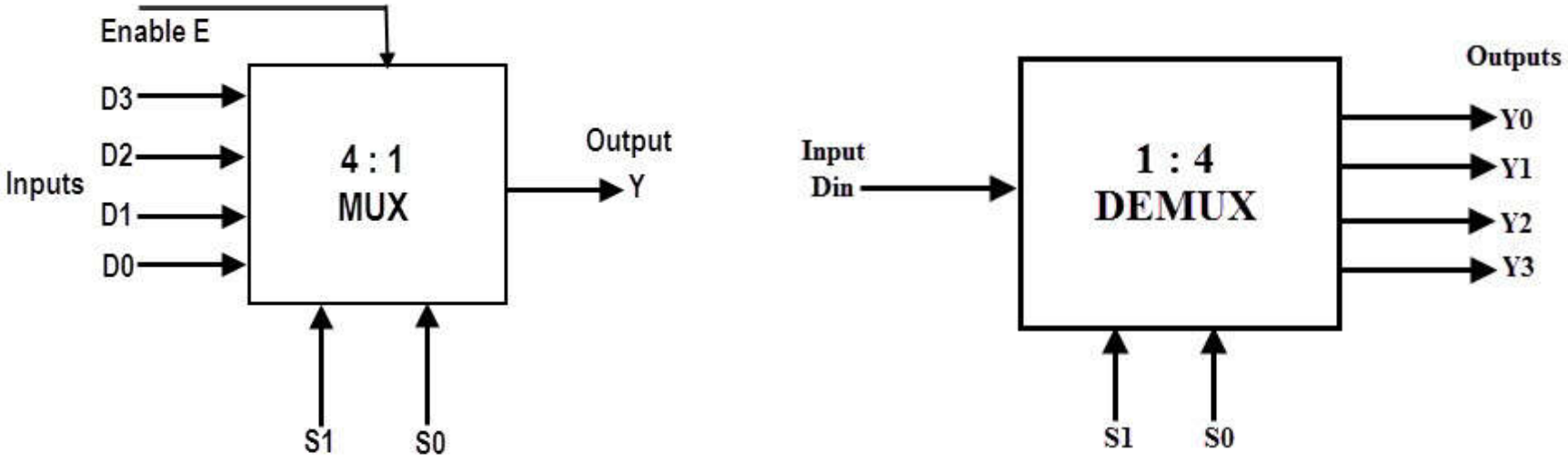
- Amplifying bidirectional lines:
  - 74LS245





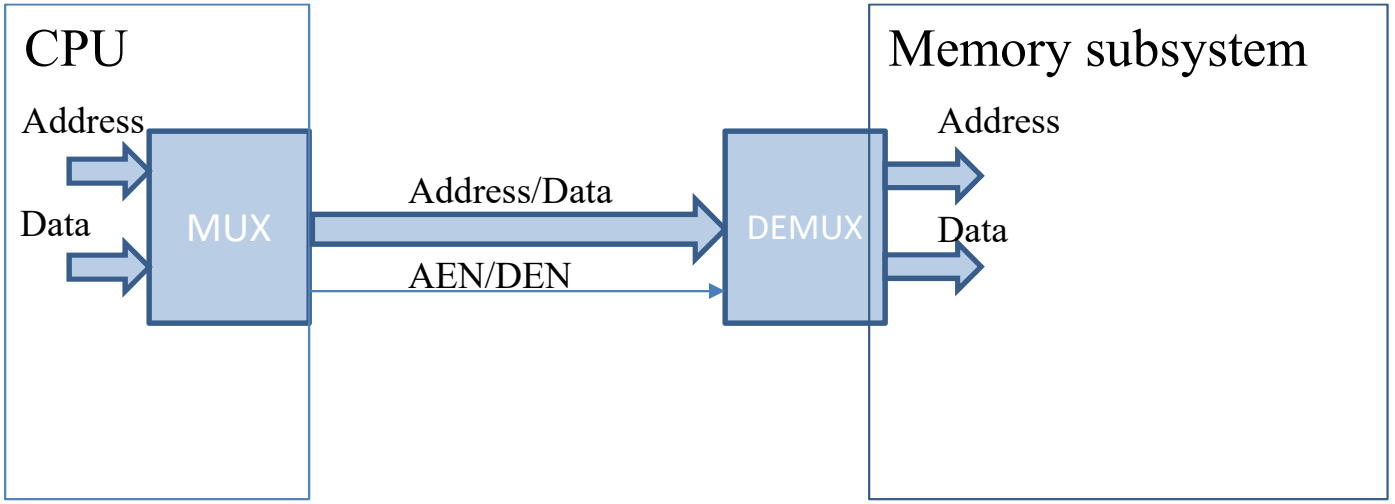
Buses

- Demultiplexing time multiplexed bus lines
  - Time multiplexed signals



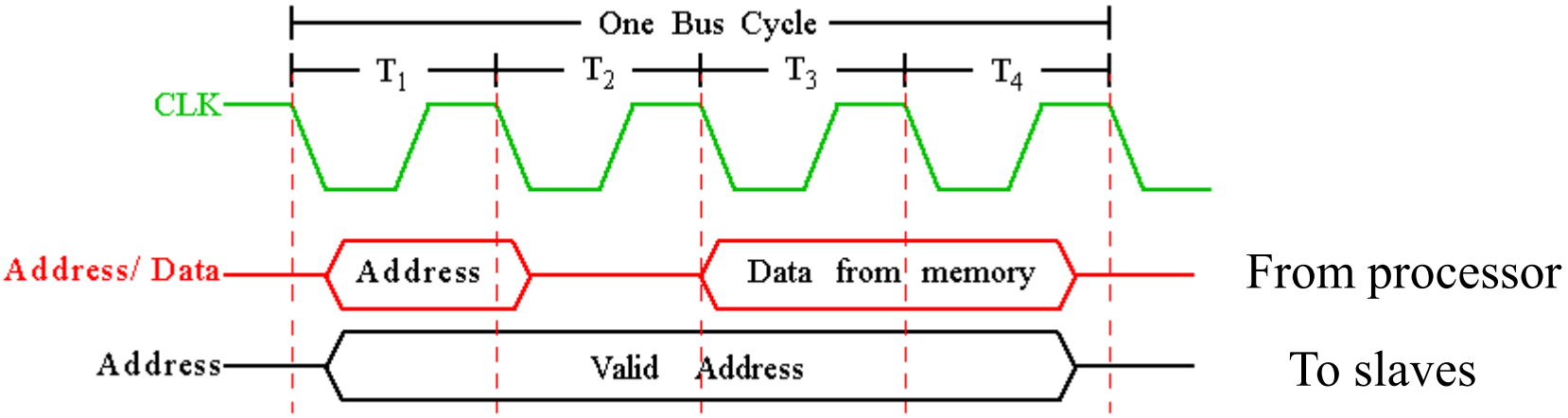
Buses

- Demultiplexing time multiplexed bus lines
  - Time multiplexed address and data values – reduce the number of CPU pins
  - Bus slaves (memories, I/O ports) require address values to be stable during the whole access cycle



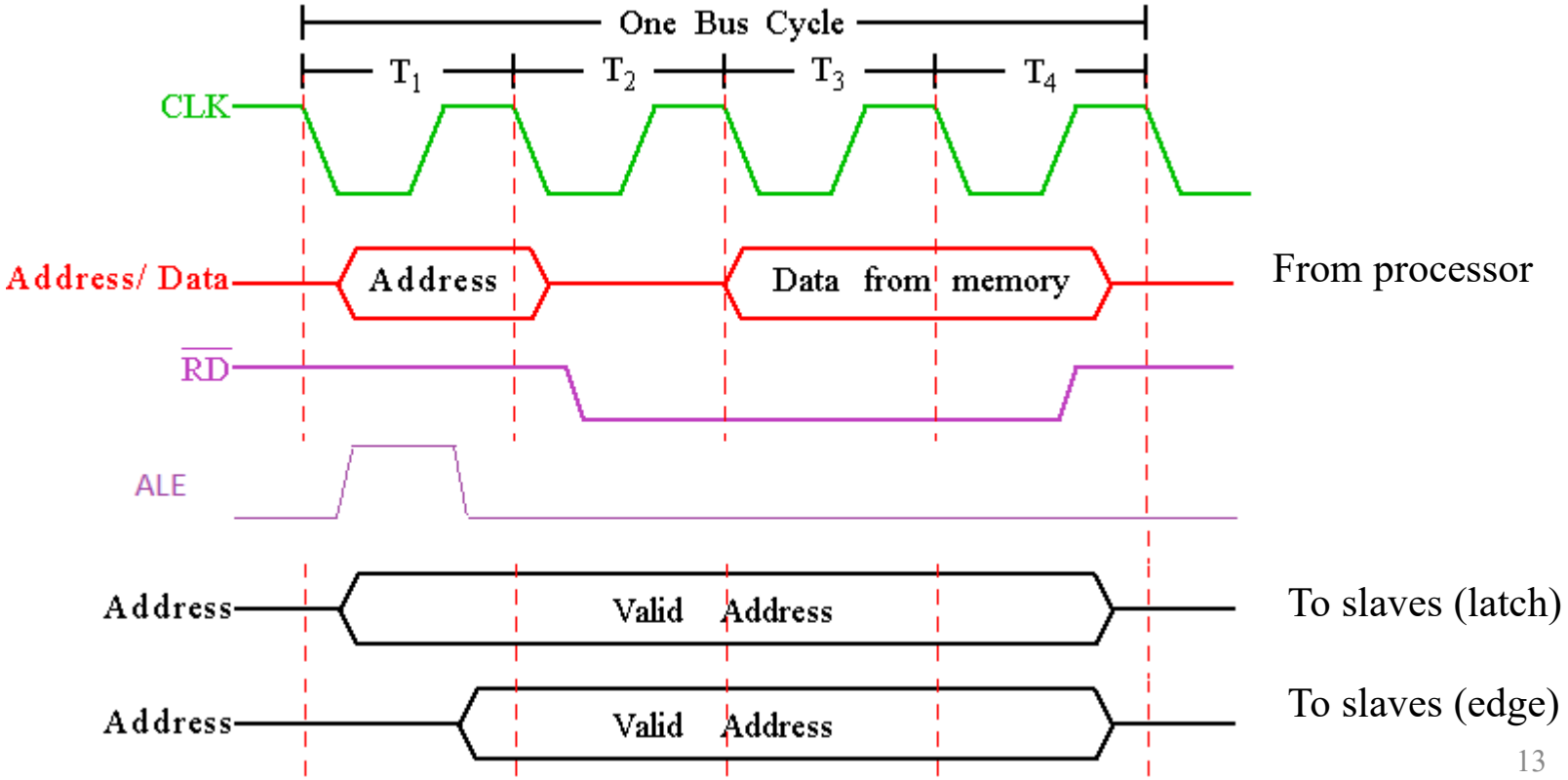
Buses

- Demultiplexing time multiplexed bus lines
  - Some processors multiplex address and data lines
  - The address is sent first followed by data time slot (RD/WR)
  - Memories need the address for the whole cycle
  - How demultiplexing addresses/data lines can be implemented?



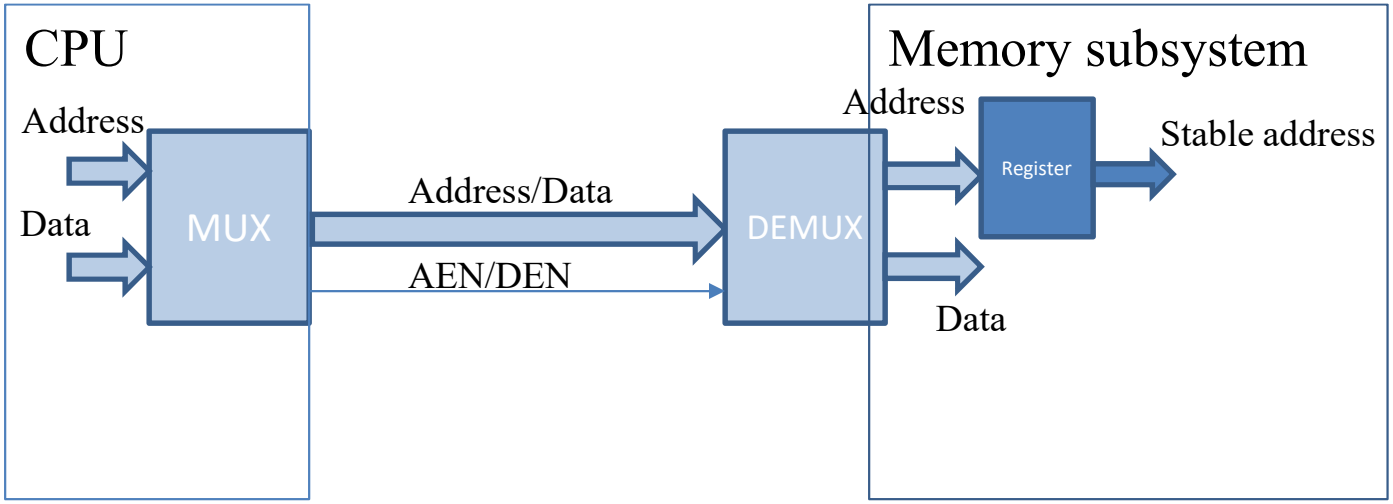
Buses

- Demultiplexing
  - Address latch enable – latches the address into the address buffers for demultiplexing



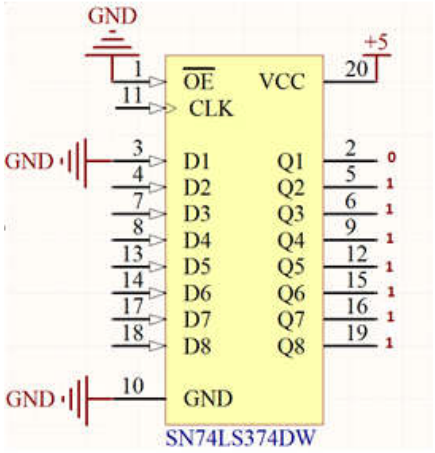
Buses

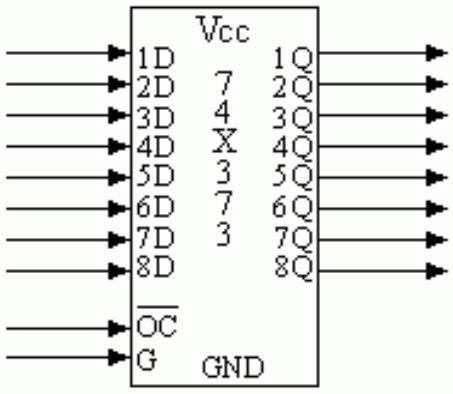
- Demultiplexing



Buses

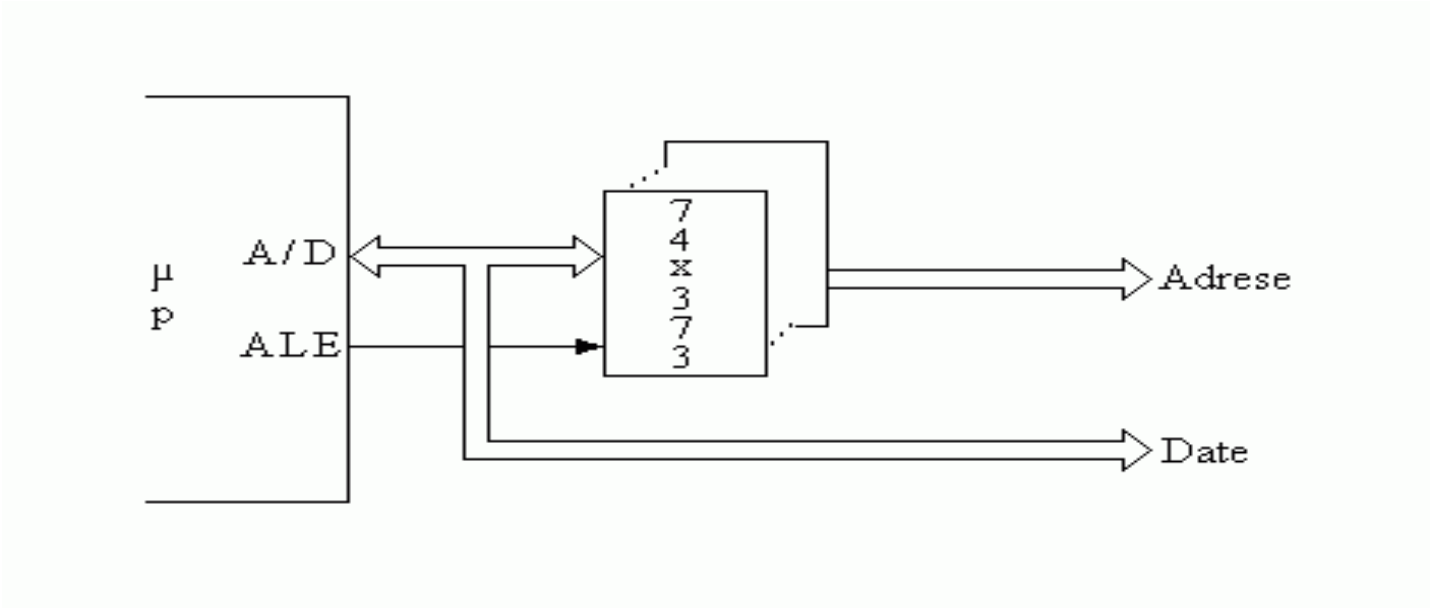
- Demultiplexing address and data values
  - Using latches/registers to store the address while the processor transfers data to/from bus slaves (bus cycles)
    - 74x373 (level), 74x374 (edge)





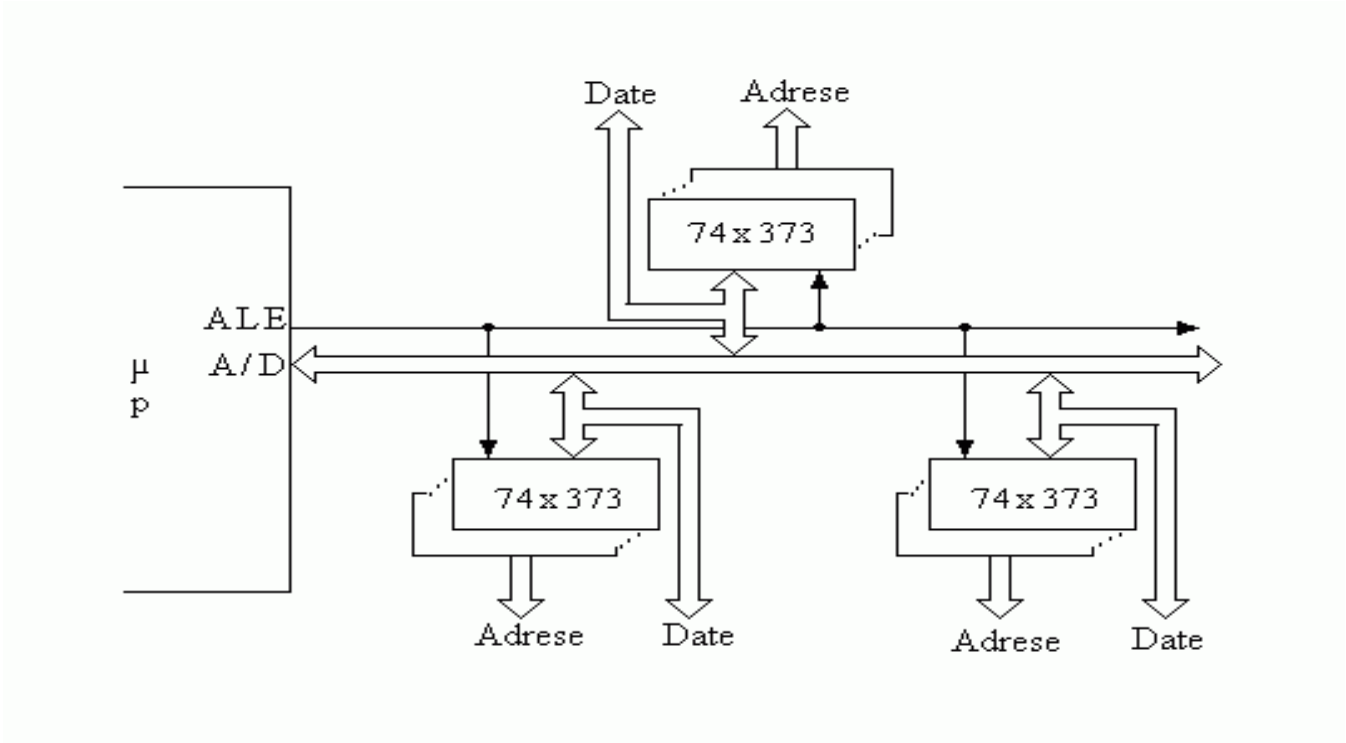
Buses

- Central demultiplexing
  - Centralized



Buses

- Local demultiplexing
  - Distributed

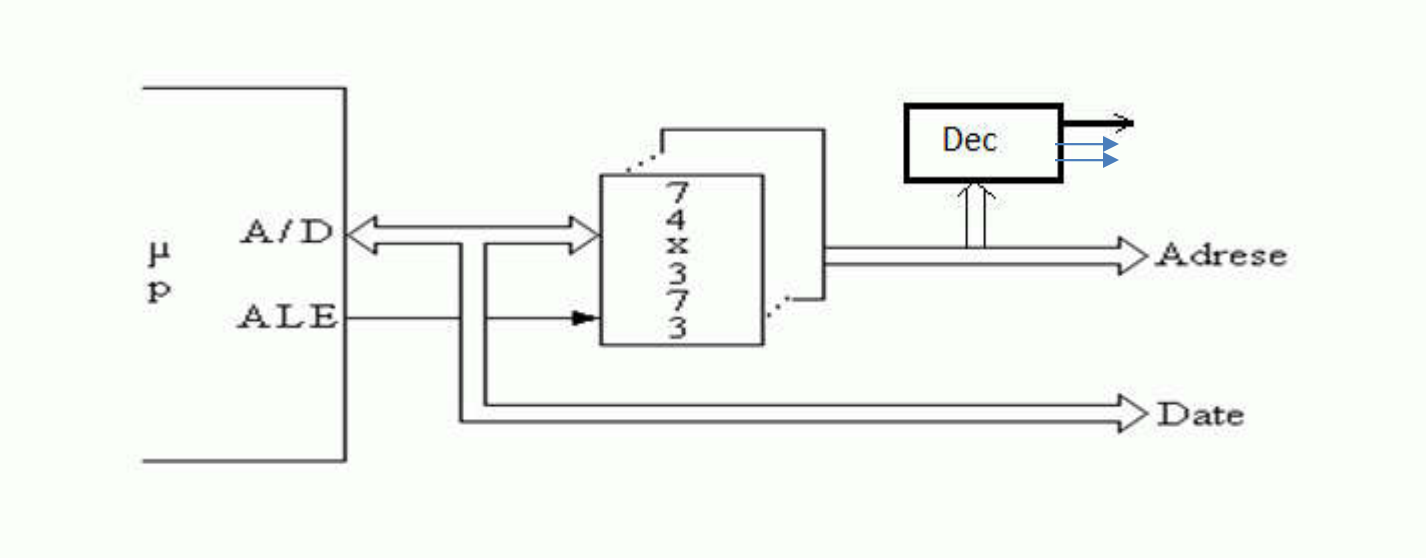


Buses

- Decoding
  - Master – slave transfers
    - One active master and one selected slave
  - Generates selection signals for each slave module in the system (memory and I/O) implementing the memory/IO map
  - Memory address space
    - Memory decoder
  - I/O address space
    - I/O decoder

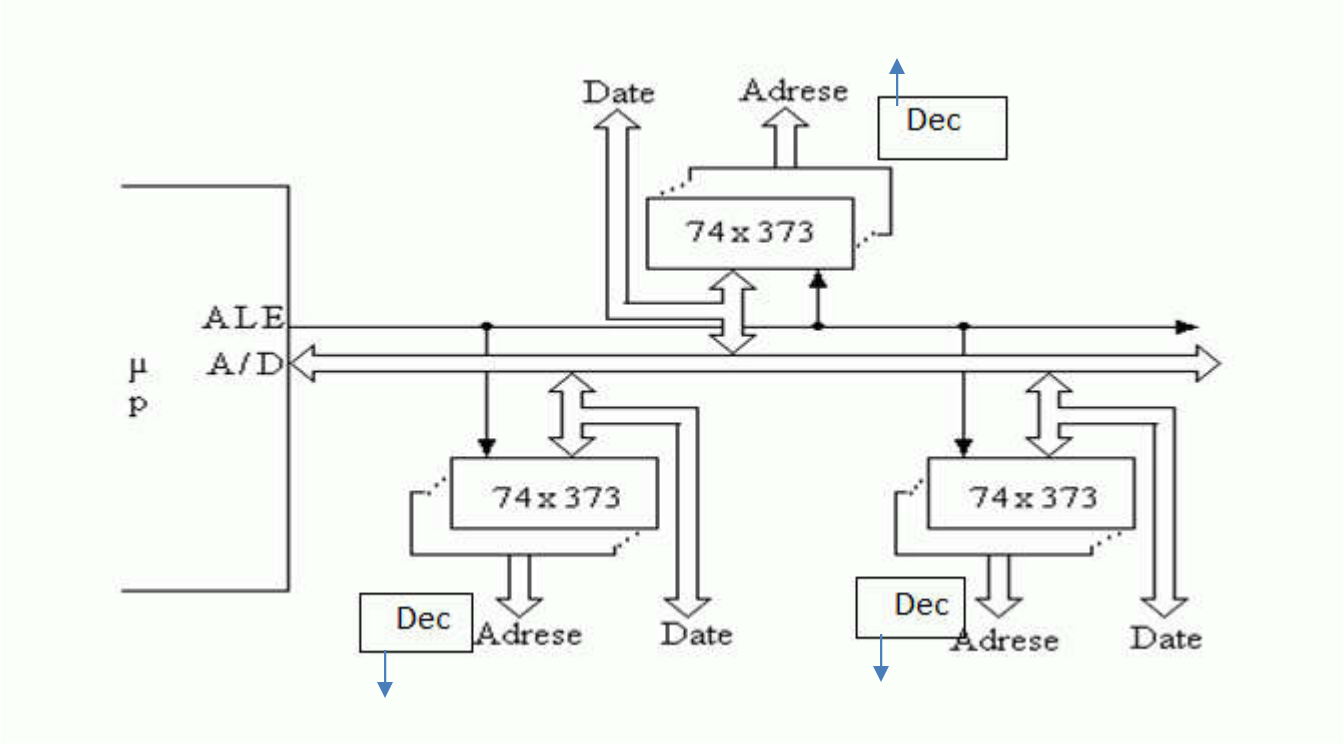
Buses

- Central decoding
  - Centralized



Buses

- Local decoding
  - Distributed

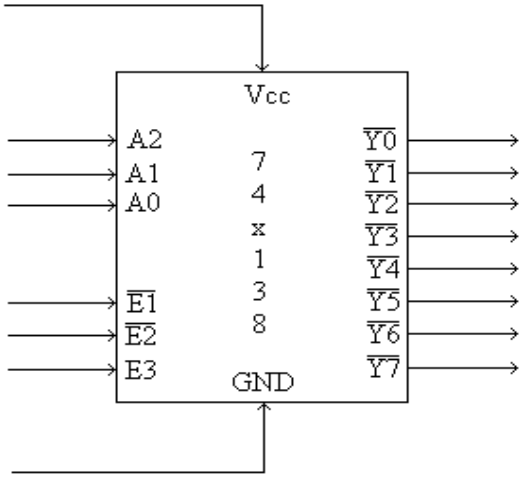


Buses

- Decoding
  - Decoding circuits
    - 74x138: decoder 3 → 8,
    - 74x139: decoder 2 x 2 → 4,
    - 74x42: decoder 4 → 10,
    - 74x154: decoder 4 → 16.

Buses

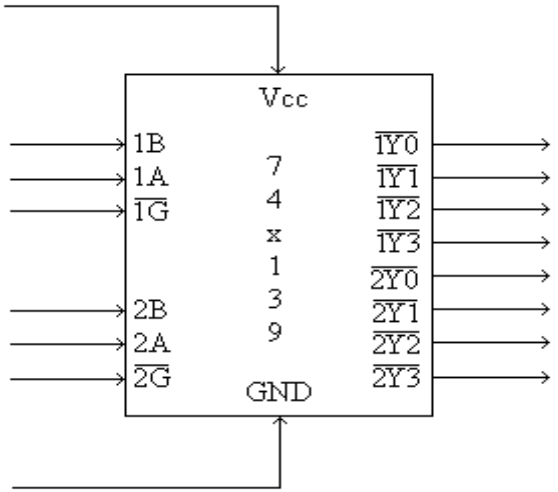
- 74x138



E3	/E2	/E1	A2	A1	A0	/Y7	/Y6	/Y5	/Y4	/Y3	/Y2	/Y1	/Y0
1	0	0	0	0	0	1	1	1	1	1	1	1	0
1	0	0	0	0	1	1	1	1	1	1	1	0	1
1	0	0	0	1	0	1	1	1	1	1	0	1	1
1	0	0	0	1	1	1	1	1	1	0	1	1	1
1	0	0	1	0	0	1	1	1	0	1	1	1	1
1	0	0	1	0	1	1	1	0	1	1	1	1	1
1	0	0	1	1	0	1	0	1	1	1	1	1	1
1	0	0	1	1	1	0	1	1	1	1	1	1	1
0	X	X	X	X	X	1	1	1	1	1	1	1	1
X	1	X	X	X	X	1	1	1	1	1	1	1	1
X	X	1	X	X	X	1	1	1	1	1	1	1	1

Buses

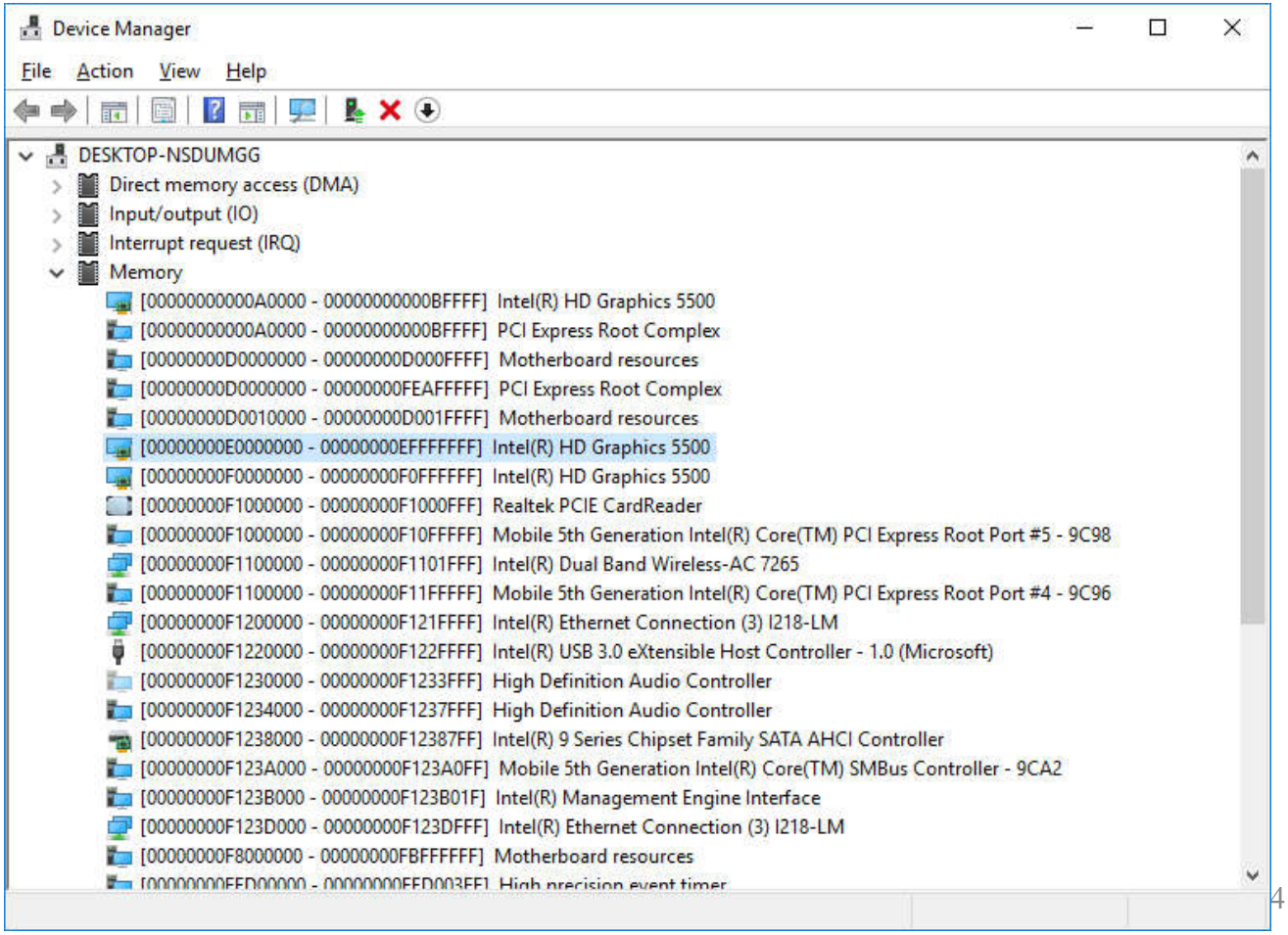
- 74x139



/iG	iB	iA	/iY3	/iY2	/iY1	/iY0
0	0	0	1	1	1	0
0	0	1	1	1	0	1
0	1	0	1	0	1	1
0	1	1	0	1	1	1
1	X	X	1	1	1	1

Memory decoder

- Memory map
  - x86/x64



Memory decoder

- Memory map
  - ARM

Address Map for processor ps7_cortexa9_[0-1]

Cell	Base Addr	High Addr	Slave I/f	Mem/Reg
ps7_intc_dist_0	0xf8f01000	0xf8f01fff		REGISTER
ps7_gpio_0	0xe000a000	0xe000afff		REGISTER
ps7_scutimer_0	0xf8f00600	0xf8f0061f		REGISTER
tree_proc_axi_0	0x40000000	0x400fffff	S_AXI_F	REGISTER
tree_proc_axi_0	0x41000000	0x410fffff	S_AXI_C	REGISTER
ps7_slcr_0	0xf8000000	0xf8000fff		REGISTER
ps7_scuwdt_0	0xf8f00620	0xf8f006ff		REGISTER
ps7_l2cachec_0	0xf8f02000	0xf8f02fff		REGISTER
ps7_scuc_0	0xf8f00000	0xf8f000fc		REGISTER
ps7_qspi_linear_0	0xfc000000	0xfcffffff		FLASH
ps7_pmu_0	0xf8893000	0xf8893fff		REGISTER
ps7_afi_1	0xf8009000	0xf8009fff		REGISTER
ps7_afi_0	0xf8008000	0xf8008fff		REGISTER
ps7_qspi_0	0xe000d000	0xe000dfff		REGISTER
ps7_usb_0	0xe0002000	0xe0002fff		REGISTER
ps7_afi_3	0xf800b000	0xf800bfff		REGISTER
ps7_afi_2	0xf800a000	0xf800afff		REGISTER
ps7_globaltimer_0	0xf8f00200	0xf8f002ff		REGISTER
ps7_dma_s	0xf8003000	0xf8003fff		REGISTER
ps7_iop_bus_config_0	0xe0200000	0xe0200fff		REGISTER
ps7_xadc_0	0xf8007100	0xf8007120		REGISTER
ps7_ddr_0	0x00100000	0x3fffffff		MEMORY
ps7_ddrc_0	0xf8006000	0xf8006fff		REGISTER
ps7_ocmc_0	0xf800c000	0xf800cfff		REGISTER
ps7_pl310_0	0xf8f02000	0xf8f02fff		REGISTER
ps7_uart_1	0xe0001000	0xe0001fff		REGISTER
ps7_coresight_comp_0	0xf8800000	0xf88fffff		REGISTER
ps7_i2c_0	0xe0004000	0xe0004fff		REGISTER
ps7_ttc_0	0xf8001000	0xf8001fff		REGISTER
ps7_scugic_0	0xf8f00100	0xf8f001ff		REGISTER
ps7_ethernet_0	0xe000b000	0xe000bfff		REGISTER
ps7_dev_cfg_0	0xf8007000	0xf80070ff		REGISTER
ps7_dma_ns	0xf8004000	0xf8004fff		REGISTER
ps7_sd_0	0xe0100000	0xe0100fff		REGISTER
ps7_gpv_0	0xf8900000	0xf89fffff		REGISTER
ps7_ram_1	0xffff0000	0xffffffff		MEMORY
ps7_ram_0	0x00000000	0x0002ffff		MEMORY

Memory decoder

- Memory map - example
  - Memory size = ending address - starting address + 1
  - 0xE0000000-0xEFFFFFFF
    - 0x0FFFFFFF + 1 = $2^{28}B = 2^8 \cdot 2^{20}B =$ 256 MB
  - 0x40000000-0x400FFFFF
    - 0x000FFFFF+1 = 1MB
  - 0x00000000-0x0002FFFF
    - 0x2FFFF+1 = 0x30000 = $2^{17} + 2^{16}$=128KB+64KB=192KB
  - 0x00000000-0x3FFFFFFF
    - 0x3FFFFFFF+1=$2^{30}B$=1GB

Memory decoder

- Component of a digital system that generates selection signal for the slave module addressed by in the current bus transaction
- Also known as address decoder
- Inputs: most significant bits of the address bus
- Outputs: selection signals for each memory module
- Design steps
  - Memory requirements and constraints
  - Memory setup
  - Memory map
  - Logic design
  - Optimization (speed, space, complexity)

Memory decoder

- Memory requirements and constraints
  - Processor requirements and constraints:
    - Reset address – covered by ROM memory
    - Interrupts vectors table – RAM memory
    - Address space – max memory size
    - Data bus width / data transfer sizes
  - Memory circuits to use:
    - Memory type, memory size
    - Data pins
  - Operating system requirements
    - OS kernel code and data
  - Compiler and applications
    - Linker setup
    - code, stack, data, heap,…

Memory decoder

- Memory setup
  - Memory circuits parameters:
    - Size (number of memory locations to store)
    - Type (ROM, SRAM, DRAM)
    - Number of address lines (n): size = 2^n
    - Number of data lines
      - number of circuits needed in one block to satisfy uP bus data width requirements
    - Addresses range: 0 - 2^n-1 (hexadecimal)

Memory decoder

- Memory setup
  - Number of circuits
    - Size of required memory / Size of available circuits
    - Take care of measurement units / data width
  - Number of blocks
    - Number of circuits needed in one block to satisfy uP bus data width requirements
    - Number of circuits per block = uP data bus width / memory data bus width
  - Address space requirements for each memory block

Memory decoder

- Memory setup
  - Example
    - Example – connect 1MB of memory using 256 KB memory circuits to a 16 bits data bus processor

    - How many circuits are needed?
    - How are they connected to the processor?
    - How many blocks are obtained?
    - What is the address space required by each block?

Memory decoder

- Memory setup example
  - Number of circuits = 1MB / 256 KB = 4 circuits
  - Number of circuits per block = 16 bits / 8 bits = 2 circuits
  - Number of blocks = 4 / 2 = 2
  - Each block has 2 x 256 KB = 512 KB = 2^{19}B
    - 0x00000000-0x0007FFFF

Memory decoder

- Memory setup
  - Example
    - Example – connect 1MB of memory using 256 Kx16bits memory circuits to a 16 bits data bus processor

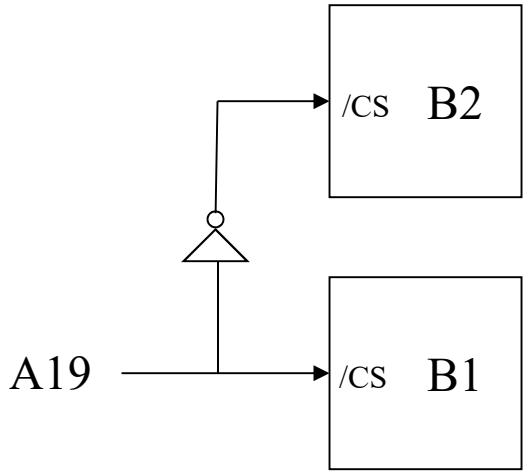
    - How many circuits are needed?
    - How are they connected to the processor?
    - How many blocks are obtained?
    - What is the address space required by each block?

Memory decoder

- Memory setup example
  - Number of circuits = 1MB / 256 Kx16bits = 1 MB / 512KB = 2 circuits
  - Number of circuits per block = 16 bits / 16 bits = 1 circuit
  - Number of blocks = 2 / 1 = 2
  - Each block has 2 x 256 KB = 512 KB = 2^{19}B
    - 0x00000000-0x0007FFFF

Memory decoder

- Logic design
  - Most significant address bits are used to select slave blocks
  - What happens when using 1 bit (MSbit) to select memory blocks?
    - Two blocks to select
  - Example:
    - 1 MB address space
    - 20 address lines
      - A19 memory block selection
      - A18-0 memory location selection
    - B1: 00000 – 7FFFFH (512 kB)
    - B2: 80000H – FFFFFH (512 kB)



Memory decoder

- Linear decoding
  - Ad-hoc logic design
    - 1 bit – 2 blocks of the same size
    - 2 bits – 4 blocks of the same size
    - …
  - Simple design
  - Less circuits
  - Lower resolution
    - Not efficient for small circuits or various sizes

Memory decoder

- Linear decoding
  - Example:
    - 1 MB address space
      - A19-0 address lines
    - 20 address lines
      - A19,A18 memory block selection
      - A17-0 memory location selection
    - Memory map
      - B1: 00000 – 3FFFFH (256 kB)
      - B2: 40000 – 7FFFFH (256 kB)
      - B3: 80000 – BFFFFH (256 kB)
      - B2: C0000H – FFFFFH (256 kB)

A19	A18	Decoder
0	0	B1
0	1	B2
1	0	B3
1	1	B4

Memory decoder

- Logic design and implementation
  - Methods
    - Linear decoding
    - Complete decoding
    - Incomplete decoding
- Optimization
  - Complexity
  - Speed
  - Circuits
  - Address range

Memory decoder

- Memory map – design rules
  - Apply application's requirements and constraints
    - High memory requirements – memories will fill all processor's address space – complete decoding
    - Low memory requirements – partial coverage of processor's address space – incomplete decoding
  - For every memory block an address range within the processor's address space has to be allocated
    - (1) Every memory circuit will receive at least an address range within processor's address space
    - (2) Address ranges allocated to any two different blocks should not overlap each other
    - (3) Starting address allocated to a memory block should be multiple of its size
      - starting_block_address = k x block_size

Memory decoder

- Memory map – design steps
  - Establish the memory setup (based on the requirements and the available circuits)
    - Number of circuits
    - Number of blocks
    - Size of blocks and address space
  - Design the memory map using the design rules
    - Start with the address space requirements
    - Select an address range for each memory block in the uP address space

Memory decoder

- Memory map – design example
  - Design the memory map for the following memory requirements of a 16 bits uP
    - Memory type ROM: 128 KB using 64Kx16 bits circuits
    - Memory type RAM: 128 KB using 32Kx8 bits circuits

Memory decoder

- Memory map – design example
  - ROM:
    - Number of circuits: 1
    - Number of blocks: 1
    - Size of block: 128KB
    - Address space: 0x00000-0x1FFFF
  - RAM:
    - Number of circuits: 4
    - Number of blocks: 2
    - Size of block: 64KB
    - Address space: 0x00000-0x0FFFF

Memory decoder

- Memory map – design example
  - `ROM:  0x00000 – 0x1FFFF`
  - `RAM1: 0x40000 – 0x4FFFF`
  - `RAM2: 0x50000 – 0x5FFFF`

Memory decoder

- Complete decoding
  - All address lines of the processor are used by memory decoder and memory circuits
    - MSbits connected to decoder
    - LSbits connected to memories
    - LSbits selection at block level (A0 – 16 bits)
  - Decoding table
    - Columns: address lines
    - Rows: 2 lines for each memory block
      - starting address and ending address allocated for that block in the memory map

Memory decoder

- Complete decoding
  - Example – design the memory decoder for the following memory map of a 16 bits processor:
    - 00000H – 1FFFFH – memory block B1, 64K x 16 bits,
    - 40000H – 4FFFFH – memory block B2, 2x32K x 8 bits,
    - 50000H – 5FFFFH – memory block B3, 2x32K x 8 bits.

Memory decoder

- Decoding table

A 19	A 18	A 17	A 16	A 15	A 14	A 13	A 12	A 11	A 10	A9	A8	A7	A6	A5	A4	A3	A2	A1	M E M
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	B1
0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	B2
0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	B3
0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	

Memory decoder

- Decoding table

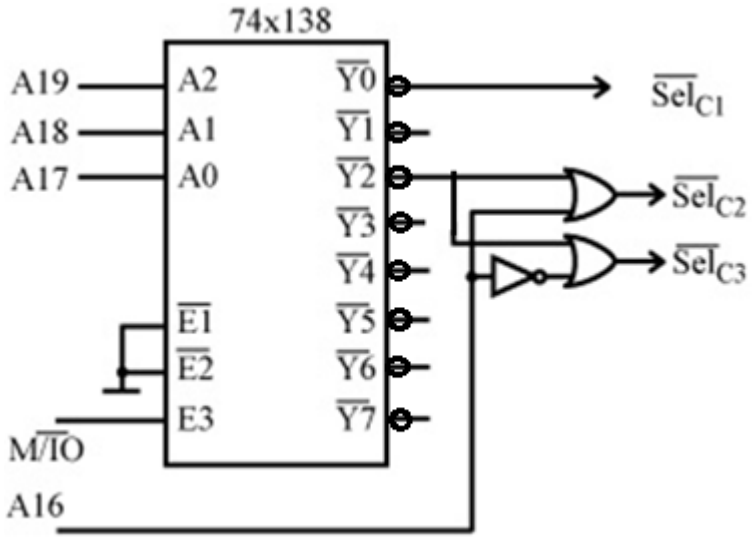
A 19	A 18	A 17	A 16	A 15	A 14	A 13	A 12	A 11	A 10	A9	A8	A7	A6	A5	A4	A3	A2	A1	M E M
0	**0**	**0**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	B1
0	**0**	**0**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
0	**1**	**0**	**0**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	B2
0	**1**	**0**	**0**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
0	**1**	**0**	**1**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	B3
0	**1**	**0**	**1**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	

Memory decoder

- Complete decoding
  - For every memory block, we select columns starting with MSb, having the same logic value of both starting address and ending address
  - The selection function of every memory block can be computed based on the logic values of the most significant bits of address bus
  - The selection function uniquely identifies the memory block it is computed for

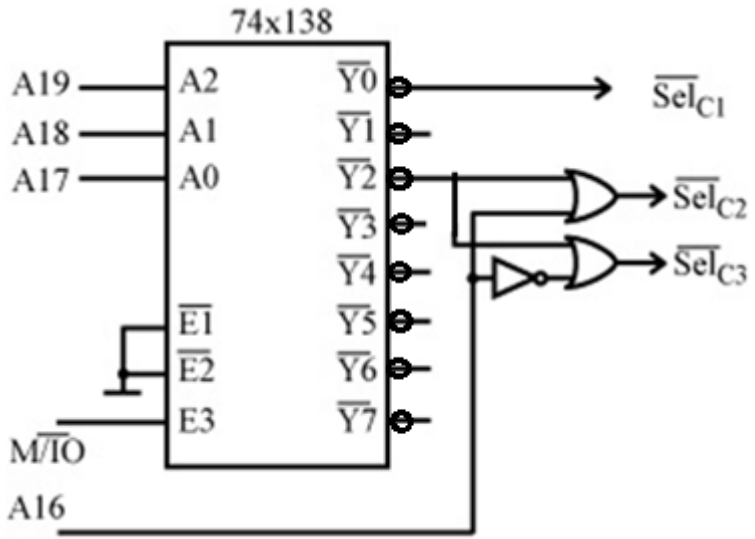
Memory decoder

- Decoding (selection) functions
  - SEL_{C1} = /A19 * /A18 * /A17
  - SEL_{C2} = /A19 * A18 * /A17 * /A16
  - SEL_{C3} = /A19 * A18 * /A17 * A16
- Decoder implementation



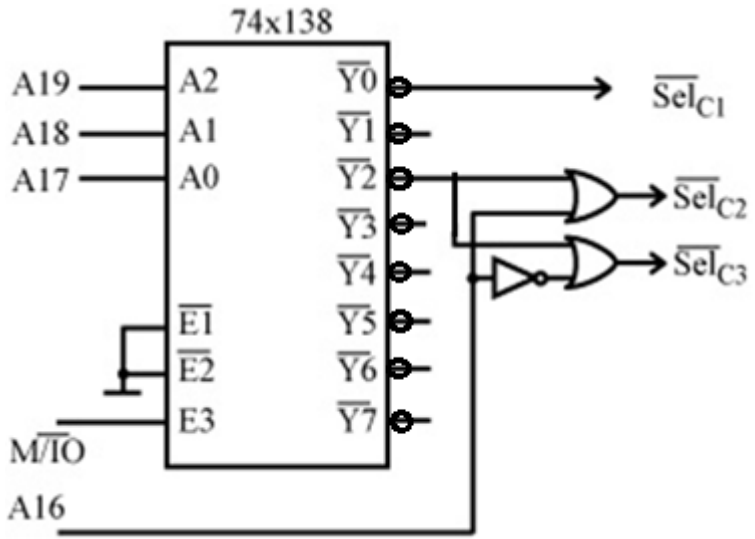
Memory decoder

- Selection signals are active on 0 logic
  - $/SEL_{C1} = \overline{/A19 * /A18 * /A17} = A19+A18+A17$
  - $/SEL_{C2} = \overline{/A19 * A18 * /A17 * /A16}$
  - $/SEL_{C3} = /A19 * A18 * /A17 * A16$



Memory decoder

- Convention to represent inverse logic
  - $/SEL_{C1}$ = /A19 * /A18 * /A17
  - $/SEL_{C2}$ = /A19 * A18 * /A17 * /A16
  - $/SEL_{C3}$ = /A19 * A18 * /A17 * A16



Memory decoder

- Complete decoding
  - Example – design the memory decoder for the following memory map:
    - 00000H – 1FFFFH – memory type ROM, 64K x 16 bits, using memory circuits of 32 KB
    - 40000H – 4FFFFH – memory type SRAM, 32K x 16 bits, using memory circuits of 16 KB
    - 80000H – 9FFFFH – memory type DRAM, 32K x 16 bits, using memory circuits of 32K x 1 bit

Memory decoder

- Incomplete decoding
  - Incomplete decoding is similar to complete decoding, but it does not use all lines of processor's address bus
  - We start with decoding table, trying to identify columns with the same logical value for every memory block
    - These address lines do not impact selection of any memory block
    - These lines can be skipped by decoder in order to simplify its implementation

Memory decoder

- Incomplete decoding
  - The consequence of using less address lines for decoding is an inefficient usage of processor address space
    - If we omit one address line when generating selection function for one memory block, that block will have two address ranges within processor's address space
    - If we omit two address lines, the memory block will occupy four address ranges

Memory decoder

- Incomplete decoding
  - Example – design the memory decoder for the following memory map:
    - 00000H – 0FFFFH – memory circuit C1, 32K x 16 bits,
    - 40000H – 4FFFFH – memory circuit C2, 32K x 16 bits,
    - 80000H – 8FFFFH – memory circuit C3, 32K x 16 bits,
    - C0000H – CFFFFH – memory circuit C4, 32K x 16 bits.

Memory decoder

- Incomplete decoding

A19	A18	A17	A16	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	C
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C1
0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C2
0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C3
1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C4
1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	

Memory decoder

- Incomplete decoding

A 19	A 18	A 17	A 16	A 15	A 14	A 13	A 12	A 11	A 10	A9	A8	A7	A6	A5	A4	A3	A2	A1	C
0	**0**	**0**	**0**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C 1
0	**0**	**0**	**0**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
0	**1**	**0**	**0**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C 2
0	**1**	**0**	**0**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
1	**0**	**0**	**0**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C 3
1	**0**	**0**	**0**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
1	**1**	**0**	**0**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C 4
1	**1**	**0**	**0**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	

Memory decoder

- Incomplete decoding

A 19	A 18	A 17	A 16	A 15	A 14	A 13	A 12	A 11	A 10	A9	A8	A7	A6	A5	A4	A3	A2	A1	C
0	**0**	**0**	**X**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C 1
0	**0**	**0**	**X**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
0	**1**	**0**	**X**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C 2
0	**1**	**0**	**X**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
1	**0**	**0**	**X**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C 3
1	**0**	**0**	**X**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
1	**1**	**0**	**X**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C 4
1	**1**	**0**	**X**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	

Memory decoder

- Complete decoding selection functions
  - $/SEL_{C1}$ = /A19 * /A18 * /A17 * /A16
  - $/SEL_{C2}$ = /A19 * A18 * /A17 * /A16
  - $/SEL_{C3}$ = A19 * /A18 * /A17 * /A16
  - $/SEL_{C4}$ = A19 * A18 * /A17 * /A16
- Incomplete decoding selection functions (removing A16)
  - $/SEL_{C1}$ = /A19 * /A18 * /A17
  - $/SEL_{C2}$ = /A19 * A18 * /A17
  - $/SEL_{C3}$ = A19 * /A18 * /A17
  - $/SEL_{C4}$ = A19 * A18 * /A17

Memory decoder

- Incomplete decoding
  - Each memory circuit will have 2 address ranges within the processor's address space
  - C1
    - 00000H – 0FFFFH,
    - 10000H – 1FFFFH,
  - Identify the incomplete decoding addresses of C2-4

Memory decoder

- Incomplete decoding

A 19	A 18	A 17	A 16	A 15	A 14	A 13	A 12	A 11	A 10	A9	A8	A7	A6	A5	A4	A3	A2	A1	C
0	**0**	**X**	**X**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C 1
0	**0**	**X**	**X**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
0	**1**	**X**	**X**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C 2
0	**1**	**X**	**X**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
1	**0**	**X**	**X**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C 3
1	**0**	**X**	**X**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	
1	**1**	**X**	**X**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	C 4
1	**1**	**X**	**X**	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	

Memory decoder

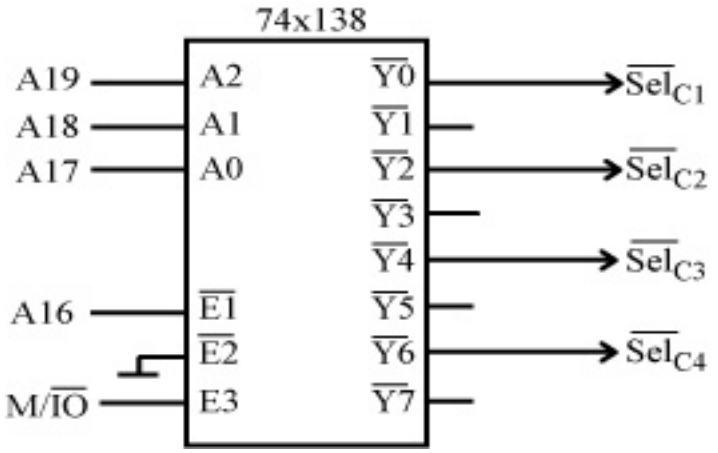
- Complete decoding selection functions
  - $/SEL_{C1}$ = /A19 * /A18 * /A17 * /A16
  - $/SEL_{C2}$ = /A19 * A18 * /A17 * /A16
  - $/SEL_{C3}$ = A19 * /A18 * /A17 * /A16
  - $/SEL_{C4}$ = A19 * A18 * /A17 * /A16
- Incomplete decoding selection functions (removing A17 and A16)
  - $/SEL_{C1}$ = /A19 * /A18
  - $/SEL_{C2}$ = /A19 * A18
  - $/SEL_{C3}$ = A19 * /A18
  - $/SEL_{C4}$ = A19 * A18

Memory decoder

- Incomplete decoding
  - Each memory circuit will have 4 address ranges within the processor's address space
  - C1
    - 00000H – 0FFFFH,
    - 10000H – 1FFFFH,
    - 20000H – 2FFFFH,
    - 30000H – 3FFFFH.
  - Identify the incomplete decoding addresses of C2-4

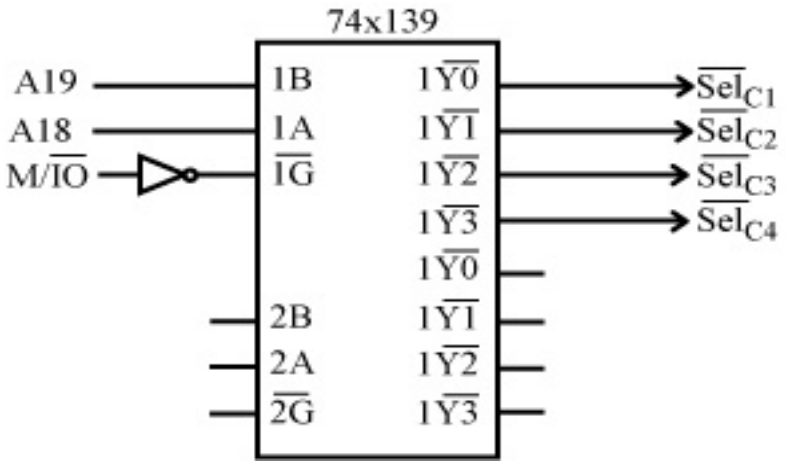
Memory decoder

- Complete decoder implementation
  - $/SEL_{C1}$ = /A19 * /A18 * /A17 * /A16 (0000)
  - $/SEL_{C2}$ = /A19 * A18 * /A17 * /A16 (0100)
  - $/SEL_{C3}$ = A19 * /A18 * /A17 * /A16 (1000)
  - $/SEL_{C4}$ = A19 * A18 * /A17 * /A16 (1100)



Memory decoder

- Incomplete decoder implementation
  - $/SEL_{C1}$ = /A19 * /A18 (00)
  - $/SEL_{C2}$ = /A19 * A18 (01)
  - $/SEL_{C3}$ = A19 * /A18 (10)
  - $/SEL_{C4}$ = A19 * A18 (11)



Memory decoder

- Complete decoding vs. incomplete decoding
  - Complete decoding assumes one and only one address range for a memory block
  - Incomplete decoding allows more than one address range for a memory block
  - Incomplete decoding may simplify the implementation of the decoder
  - Complete decoding allows the efficient usage of the microprocessor address space
  - Incomplete decoding is used for small size memory requirements
  - Incomplete decoding design has low scalability and extensibility

Data transfers

- Memory map - summary
  - Memory circuits parameters
    - Number of circuits
      - Divide the amount of required memory size by the size of the available circuits
        - » 64 kB / 16 kB = 4 circuits
    - Number of blocks
      - Group circuits together in one block so that the memory will provide the maximum data size required by the processor in one transaction (size of the data bus)
      - Divide processor's data width by memory data lines to calculate the number of the circuits needed in one block
        - » 16 / 8 = 2 circuits in one block
        - » 2 blocks of 2 circuits each
        - » 1 block has 32 kB

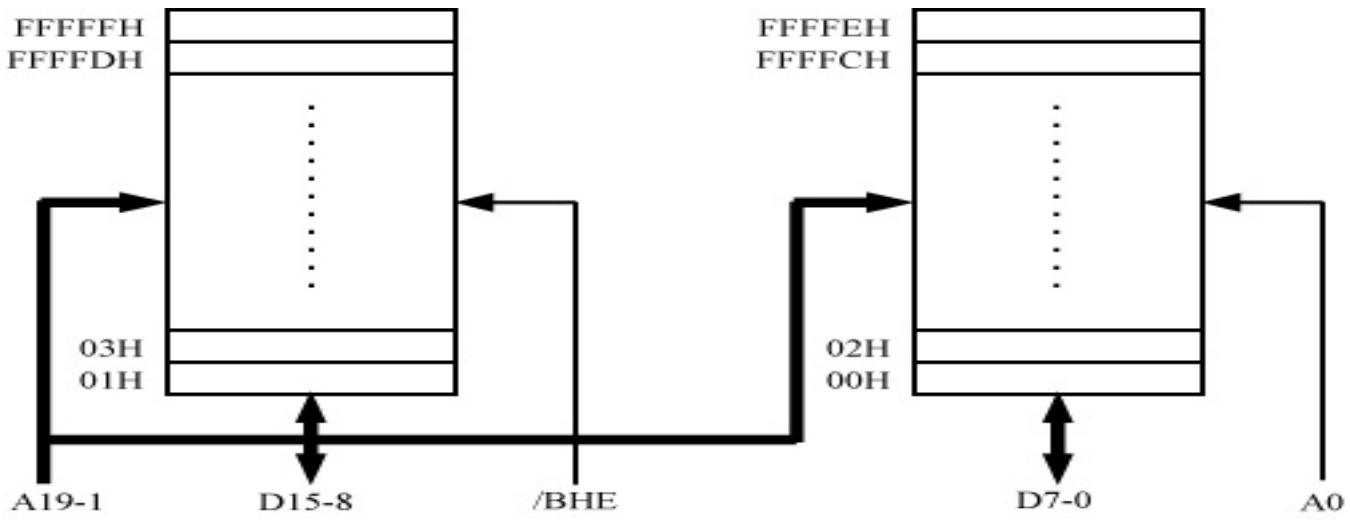
Data transfers

- 16 bits processors have to implement both:
  - 16 bits transfers
  - 8 bits transfers
- 8086: /BHE and A0

$\overline{BHE}$	A_0	Characteristics
0	0	Whole word
0	1	Upper byte from/to odd address
1	0	Lower byte from/to even address
1	1	None

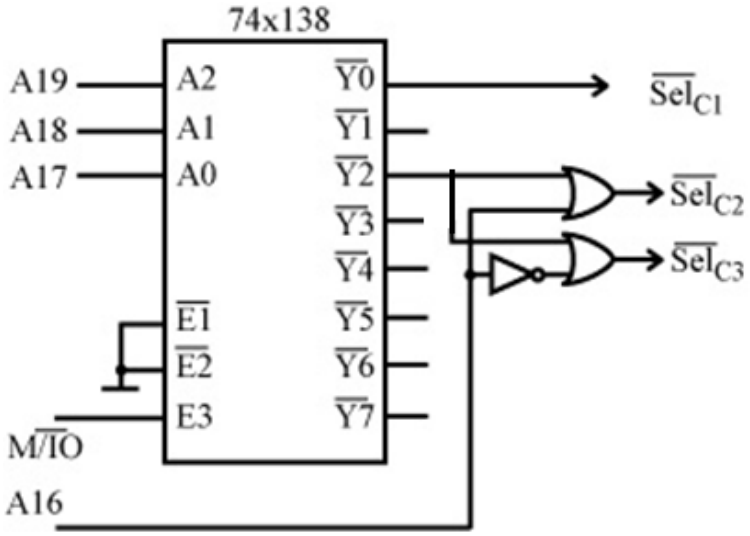
Data transfers

- 16 bits transfers – even address (A0 = 0 , /BHE = 0)
- 8 bits transfers
  - Odd address (A0 = 1, /BHE = 0)
  - Even address (A0 = 0, /BHE = 1)



Summary

- Bus
  - Amplifying bus lines
  - Demultiplexing bus lines
  - Decoding address lines
  - Data transfers


74x138
A19
A18
A17
A2
A1
A0
$\overline{E1}$
$\overline{E2}$
E3
M/$\overline{IO}$
A16
$\overline{Y0}$
$\overline{Y1}$
$\overline{Y2}$
$\overline{Y3}$
$\overline{Y4}$
$\overline{Y5}$
$\overline{Y6}$
$\overline{Y7}$
$\overline{Sel}_{C1}$
$\overline{Sel}_{C2}$
$\overline{Sel}_{C3}$