SYSC3601
Microprocessor Systems

Unit 5:
Memory Structures and Interfacing
1. Memory Types
2. Interfacing 8-bit memory/IO (8088)
3. Interfacing 16-bit memory/IO (8086)
4. Interfacing 32/64-bit memory/IO

Reading: Chapter 10, skim 10-5 & 10-6
Memory Types

Read-Only Memory (ROM)
Non-volatile!

**ROM** Read-Only Memory

programmed during fabrications at factory.

control program in dedicated µP systems is stored in ROM.

**PROM** Programmable Read-Only Memory.

programmed by burning (blowing) tiny Nichrome or silicon-oxide fuses.

Once programmed, it cannot be erased.
Read-Only Memory (ROM) con’t

**EPROM** Erasable Programmable Read Only Memory.

- Memory can be erased by exposure to UV light (up to 20min)
- can be programmed by user, but it is usually removed to be erased.
- Ex: 2716, 2764, 27256.

**EEPROM** Electrically Erasable Programmable Read-Only Memory

- Also called Flash MemoryTM (Intel), or EARM (Electrically Alterable ROM).
- Can be erased and reprogrammed in and by the system.
Random Access Memory (RAM)
2 types:

1) Static (SRAM)
   Retains data for as long as power is applied
   FAST, EXPENSIVE, BIG
   higher gate count – 4 or 6, needs a flip flop
   Used for cache
Memory Types

Random Access Memory (RAM) con’t

2) Dynamic (DRAM)
Retains data for only 2-4ms, then must be refreshed.

Slower but cheaper and can be larger (e.g. 2GB DIMM)

High density (1 transistor plus capacitor)

Usually use a DRAM controller to handle interfacing and refresh.
Memory Types

DRAM devices

- Address pins are *multiplexed*!
- $A_0$-$A_7$ loaded first with $\overline{RAS}$ (row address select).
- $A_8$-$A_{15}$ loaded second with $\overline{CAS}$ (column address select).
- $\overline{CAS}$ also serves as chip select.

Figure 1: TMS 4464
DRAM
Memory Types

DRAM Organization

[Diagram of DRAM Organization]
Memory Types

DRAM types

EDO  Extended Data output DRAM.
  • bits selected by RAS are latched.
  • faster for sequential address - no wait states for sequential accesses.

SDRAM  Synchronous DRAM
  • faster (access times 10nS-8nS).
  • internal state machine tied to system clock controlling operation.
  • Used for “burst-read” 4 64-bit numbers.

DDR  Double data rate. (sequences at twice system clock)

RDRAM  Rambus DRAM.
Memory & I/O Interfacing

General steps for memory and I/O interfacing

Generic memory device:

Memory chip size: $2^N \times M$ bits
Memory & I/O Interfacing

N address lines can address $2^N$ memory locations:

<table>
<thead>
<tr>
<th>Size</th>
<th>Lines</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>1k</td>
<td>1024</td>
<td>000000-003FF</td>
</tr>
<tr>
<td>2k</td>
<td>2048</td>
<td>000000-007FF</td>
</tr>
<tr>
<td>4k</td>
<td>4096</td>
<td>000000-00FFF</td>
</tr>
<tr>
<td>1M</td>
<td>10848576</td>
<td>000000-FFFFF</td>
</tr>
</tbody>
</table>
Memory & I/O Interfacing

Steps to success:

1) Architectural questions:

   How many chips are required?

   How many address lines go to each chip?

   How will chips be organized into banks and which parts of the address bus will be used?

2) Determine address range:

   Typically problem is to place devices within memory map

   Determine START, SIZE, LO (=START), HI (=LO+SIZE-1)

   Determine CONST, SEL, and MEM address lines

3) Generate overall chip select signal (MSEL) from CONST portion of address range and M/IO

4) Generate bank-specific write signals if required

5) Complete interface design! (often using decoders)

   Be sure to connect address bus, data bus, and control bus (RD, WR)
Ex: Design an interface for an 8088 μP to connect a single 2716 (2K x 8) EPROM such that memory starts at address FF800H.

Notes:

The 8088 has 20 address lines and 8 data lines (assuming that it is already fully demultiplexed and buffered)

The 2716 has 1 CS (chip select) pin and one OE (output enable) pin.

Standard logic gates (NAND, NOR, NOT) may be used.
Memory & I/O Interfacing Example 1

Steps to success:

1) Architectural questions:
   How many chips are required? **ONE**
   How many address lines go to each chip?
     The 2716 has \(2k = 21 \times 210 = 211\) locations, so we need **11 address lines** to the 2716 chip.
   How will chips be organized into banks and which parts of the address bus will be used?
     Only 8-bit data bus -> only one bank. No bank-enable signals required.

2) Determine address range:
   START =
   SIZE =
   LO (=START) =
   HI (=LO+SIZE-1)
   Determine CONST, SEL, and MEM address lines

3) Generate overall chip select signal from CONST portion of address range and M/IO

4) Generate bank-specific write signals if required

5) Complete interface design! (often using decoders)
   Be sure to connect address bus, data bus, and control bus (RD, WR)
Memory & I/O Interfacing Example 1

Steps to success:
1) Architectural questions:

2) Determine address range:

   START = FF800h

   SIZE = 800h (2K)

   LO (=START) = FF800h

   HI (=LO+SIZE-1) = FFFFFh (FF800h+800h-1)

Determine CONST, SEL, and MEM address lines:

<table>
<thead>
<tr>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

CNST (decode)  To Memory Device
Memory & I/O Interfacing Example 1

Steps to success:
1) Architectural questions:

2) Determine address range:

3) Generate overall chip select signal from CONST portion of address range and M/IO:

<table>
<thead>
<tr>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

CNST (decode)

=1 only when A19 through A11 are all =1 (IO/M since 8088)
8-bit Memory Interfacing Example 1

4) Generate bank-specific write signals if required \textbf{NOT}

5) Complete interface design! (often using decoders)

Be sure to \textit{connect} address bus, data bus, and control bus (RD, WR):

Don’t forget! (can include in SEL)
# Address Decoding

## Notes on Address Decoding

Address range will look something like this:

<table>
<thead>
<tr>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>M</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constant</th>
<th>Sel</th>
<th>Memory device</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant: to decoder or gate logic to select bank or enable decoders for DDD.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DDD: to decoder to select a memory device.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MMM: to memory devices depending on available address pins.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Address Decoding

Logic decoders:

1. Must also decode IO/\overline{M}.
2. Must use \overline{RD} or \overline{WR}.
Address Decoding

Programmable Decoders

**PLD** Programmable logic device
Arrays of logic elements that are programmable.

3 types:

- **PLA** Programmable logic array
- **PAL** Programmable array logic
- **GAL** Gated array logic

PAL has replaced PROM address decoders in latest memory interfaces.
Typically constructed with AND/OR/NOT logic.
PALs are programmed using software such as PALASM.
Many examples in text use PAL decoders.
In class, we will use logic gates directly, but in practice, PALs can reduce the chip count.
Sample PLD
The 74LS138 3-to-8 decoder

- $Y_n$ goes low whenever:
  1. $G1 = '1'$ and $G2A = G2B = '0'$.
  2. $n = C \times 4 + B \times 2 + A \times 1$

<table>
<thead>
<tr>
<th>C</th>
<th>B</th>
<th>A</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>5</td>
</tr>
</tbody>
</table>

- Typical time delay is 12nS.
Ex: Design a 64k x 8 section of memory for the 8088 using 8k x 8 (2764) EPROMs. The memory start address is A0000H.

Steps to success:
1) Architectural questions:

How many chips are required? **EIGHT**

We need 8 2764s (8k each) for 64k total memory.

How many address lines go to each chip?

The 2764 has 8k = 23 x 210 = 213 locations, so we need **13 address lines** to each 2764 chip (the same 13 lines go to ALL chips).

How will chips be organized into banks and which parts of the address bus will be used?

Only 8-bit data bus -> only one bank. No bank-enable signals required.
Memory & I/O Interfacing Example 2

Steps to success:
1) Architectural questions:
2) Determine address range:

We need a 64k = 26 x 210 = 216 byte memory block.

START = A0000h
SIZE = 10000h (64K = size of ALL chips put together)

LO (=START) = A0000h
HI (=LO+SIZE-1) = AFFFFh (A0000h+10000h-1)

<table>
<thead>
<tr>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**CNST**  **SEL**  **Memory Device**

Will enable decoder  Goes to 74LS138 decoder  Goes to each mem chip
Steps to success:

1) Architectural questions:
2) Determine address range:
3) Generate overall chip select signal from CONST portion of address range and M/IO:

<table>
<thead>
<tr>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

CNST
8-bit Memory Interfacing Example 2

4) Generate bank-specific write signals if required **NOT REQUIRED**

5) Complete interface design! (often using decoders)

Be sure to connect address bus, data bus, and control bus (RD, WR):

![Diagram of memory interfacing example]
EPROM location in 8088/8086 systems

Note: Normally, the 8088 has EPROM located from F8000H to FFFFFFH (upper 32k) since a hardware reset starts execution at FFFFFFFH.

Note: Slower versions of many 2764 EPROMs have memory access times of 450nS.

8088 allows 460nS.

Decoder (’138) delay is 12nS

Must use READY signal to insert wait state using 8284A clock generator. (how many?)
The 8086 has a 16-bit data bus. Memory is arranged in two 8-bit banks:
- low bank: contains all even addresses.
- high bank: contains all odd addresses.
8086 Memory Interface

Aligned/unaligned words

W1 is stored on an even (aligned) address.

It can be accessed in a single read cycle.

W2 is stored at an odd (unaligned) address.

It will require two read cycles (8 T-cycles).

(a) During first read, W2L (odd address) will appear on the high byte of the data bus.

(b) During the second read, W2H (even address) will appear on the low byte of data bus.

During a read operation, both banks may (and often are) activated.
The µP will read 16-bits for read operations, or will only read the correct half of the data bus for byte operations.

Note that AL may receive data from the high half of the data bus when reading a byte from an odd address!
8086 Memory Interface

Write cycles must activate the correct bank(s) based on BHE and BLE (A0). BHE is supplied by µP (multiplexed with S7) A0 is used as BLE i.e. A0=0 for an even address and A0=1 for an odd address (A0 is not even a pin on the 386 and up)

<table>
<thead>
<tr>
<th>BHE</th>
<th>BLE</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Both banks (16 bits)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>High bank (8 bits)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Low bank (8 bits)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>No banks enabled</td>
</tr>
</tbody>
</table>
Memory & I/O Interfacing

Remember the Steps to Success:

1) Architectural questions:
   How many chips are required?
   How many address lines go to each chip?
   How will chips be organized into banks and which parts of the address bus will be used?

2) Determine address range:
   Typically problem is to place devices within memory map
   Determine START, SIZE, LO (START), HI (LO+SIZE-1)
   Determine CONST, SEL, and MEM address lines

3) Generate overall chip select signal (MSEL) from CONST portion of address range and M/IO

4) Generate bank-specific write signals if required

5) Complete interface design! (often using decoders)
   Be sure to connect address bus, data bus, and control bus (RD, WR)
Design a memory interface for the 8086 which will provide 256k bytes of SRAM, organized as 128k x 16bits, starting at address 40000H and using 62256 SRAM chips (32k x 8bit).

Assume that 8086 address, data, status, and control busses are already demultiplexed and buffered.

1. Architectural questions:
   We want 128k x 16 bits
   i.e. 128k x 16bits → 4 chips for both the high and low banks.

   8 chips total

   62256 chips are 32k x 8bit; 32k = 25 x 210 ← 15 address lines

   We will use a 2-to-4 decoder (74LS139) to select one out of four chips from each bank.
2a. Address Range:
   Start address is 40000h

   SIZE: 256k bytes is $28 \times 2^{10}$ → 40000h bytes

   Therefore address range is:

   Low = start = 40000h

   High = (start+size-1) = (40000h+40000h-1) = 7FFFFFh
2b. Address Decoding:

<table>
<thead>
<tr>
<th>19</th>
<th>18</th>
<th>17</th>
<th>16</th>
<th>15</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

| DD | SS | Address Lines | B |

D Decode with \( \overline{M/IO} \) to select memory
S to 2-to-4 decoder.
B to \( \overline{BLE} \)

3. MSEL:

\[ \begin{align*}
& M/IO \\
& A_{18} \\
& A_{19} \\
& \overline{MSEL}
\end{align*} \]
4. Bank-specific write signals:
The 8086 already handles read from high/low/both banks as needed (W bit)
We must select hi/low/both for write control

5. Design memory interface (next slide).
Be sure to connect address bus, data bus, and control bus (RD, WR)
What address range does each chip respond to?
32-bit Wide Memory

Requires 4 banks, each 8-bits wide to generate (up to) 32-bits per read/write
Bank ID is system address ‘mod 4’
Requires 4 bank enable signals for writes:
32-bit Wide Memory

Click to edit Master text styles

Second level

Third level

Fourth level

Fifth level
64-bit Wide Memory