Tutorial 1: Memory Mapping

Prof. John McLeod

ECE3375, 2022 01 14

Problem: Design a memory system for a CPU with $64\,\mathrm{KB}$ memory space and a 8-bit data bus with the following specifications:

- $24\,\mathrm{K}$ memory of ROM, occupying a continuous block at the top of the memory space.
- $8\,\mathrm{K}$ memory of RAM, occupying a continuous block at the bottom of the memory space.
- $8\,\mathrm{K}$ memory of RAM, occupying a continuous block starting at address `0x3000`.

You have access to the following chips:

- Three (3) $64\,\mathrm{Kb}$ ROMs, organized $8\,\mathrm{K} \times 8$.
- Two (2) $32\,\mathrm{Kb}$ RAMs, organized $8\,\mathrm{K} \times 4$.
- Two (2) $32\,\mathrm{Kb}$ RAMs, organized $4\,\mathrm{K} \times 8$.

Draw the memory map, find the chip select logic, and show how the address and data buses are wired to the individual chips.

Solution: A $64\,\mathrm{KB}$ memory space is $2^6 \times 2^{10}\,\mathrm{B}$, so the address bus has 16 bits. The memory addresses can be written as four hex digits $(16 = 4 \times 4)$, therefore the memory address range from `0x0000` to `0xFFFF`.

A $8\,\mathrm{K}$ chip has $2^3 \times 2^{10} = 2^{13}$ addresses, or `0x2000` in hex ($2^{13} = 2 \times 2^4 \times 2^4 \times 2^4$, each 2^4 represents a "16's place" in hex), while a $4\,\mathrm{K}$ chip has 2^{12} addresses, or `0x1000` in hex.

- $24\,\mathrm{K}$ memory of ROM requires all of the three $64\,\mathrm{Kb}$ ROMs (as each ROM possesses $8\,\mathrm{K}$ memory addresses). In hex $24\,\mathrm{K}$ requires $3 \times$ `0x2000` $=$ `0x6000` addresses. If the ROM sits at the top of memory space, this ranges from `0x10000` $-$ `0x6000` $=$ `0xa000` up to `0xFFFF` (inclusive).

- The $8\,\mathrm{K}$ memory of RAM starting at `0x3000` requires `0x2000` addresses, ranging from `0x3000` will range up to `0x4FFF` (inclusive).

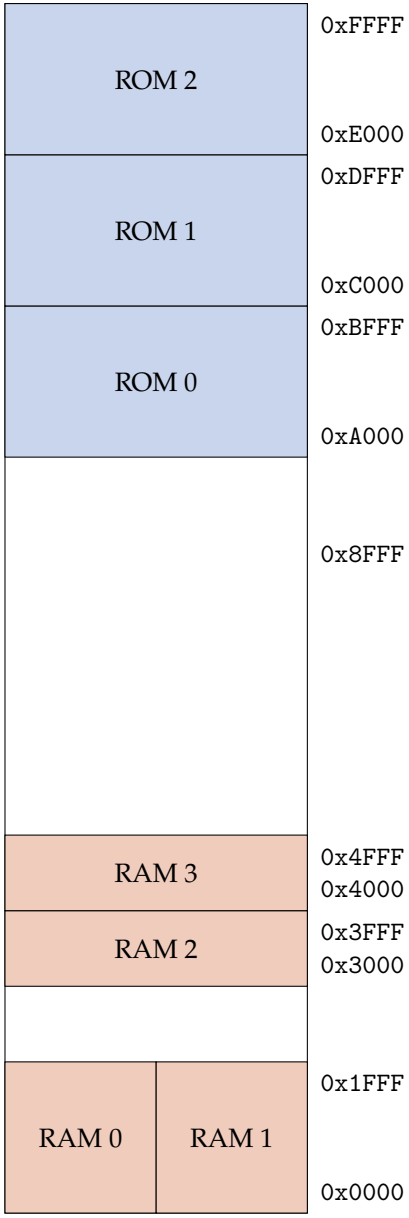
- The $8\,\mathrm{K}$ memory of RAM starting at the bottom of the memory space similarly requires `0x2000` addresses, so it ranges from `0x0000` to `0x1FFF` (inclusive).

Really, the only tricky part of this question is deciding which RAM chips go where, and recognizing that the $8\,\mathrm{K} \times 4$ RAM chips need to sit in *parallel* in this memory space, because they are each only one **nibble** wide.

The important design choice is to **place memory chips at starting addresses that are evenly divisible by the chip size**.

- We could put the $8\,\mathrm{K} \times 4$ RAMs starting at `0x3000`, but the size of those chips (`0x2000`) is not evenly divisible by the starting address (as $3/2 = 1.5$) so this will add difficulty to the problem.
- Instead, it is easier to put the two $8\,\mathrm{K} \times 4$ RAMs in parallel at `0x0000`, and use the two $4\,\mathrm{K} \times 8$ RAMs for the memory starting at `0x3000`.

I will solve this problem the "easy way" first, before returning to the issue of why the other option is the "hard way". With this in mind, the memory map is shown to the right.



Now we need to determine the chip select logic. I will write out the start and end addresses for each chip explicitly in hex and binary for thoroughness. Those of you who passed ECE2277 can probably figure out the chip select logic without such tedious steps.

Chip	Position	Hex Address	Binary Address
RAM 0	Start	`0x0000`	000 0 0000 0000 0000
	End	`0x1FFF`	000 1 1111 1111 1111
RAM 1	Start	`0x0000`	000 0 0000 0000 0000
	End	`0x1FFF`	000 1 1111 1111 1111
RAM 2	Start	`0x3000`	0011 0000 0000 0000
	End	`0x3FFF`	0011 1111 1111 1111
RAM 3	Start	`0x4000`	0100 0000 0000 0000
	End	`0x4FFF`	0100 1111 1111 1111
ROM 0	Start	`0xA000`	101 0 0000 0000 0000
	End	`0xBFFF`	101 1 1111 1111 1111
ROM 1	Start	`0xC000`	110 0 0000 0000 0000
	End	`0xDFFF`	110 1 1111 1111 1111
ROM 2	Start	`0xE000`	111 0 0000 0000 0000
	End	`0xFFFF`	111 1 1111 1111 1111

This bits highlighted in blue are the ones that are consistent over the entire data range for each chip, so they should be used for the chip select logic. Once you have the bit pattern for each chip, this is easily translated into a chip select logic by writing out the n-`AND` of

those n bits, taking the complement of any bits that are zero.

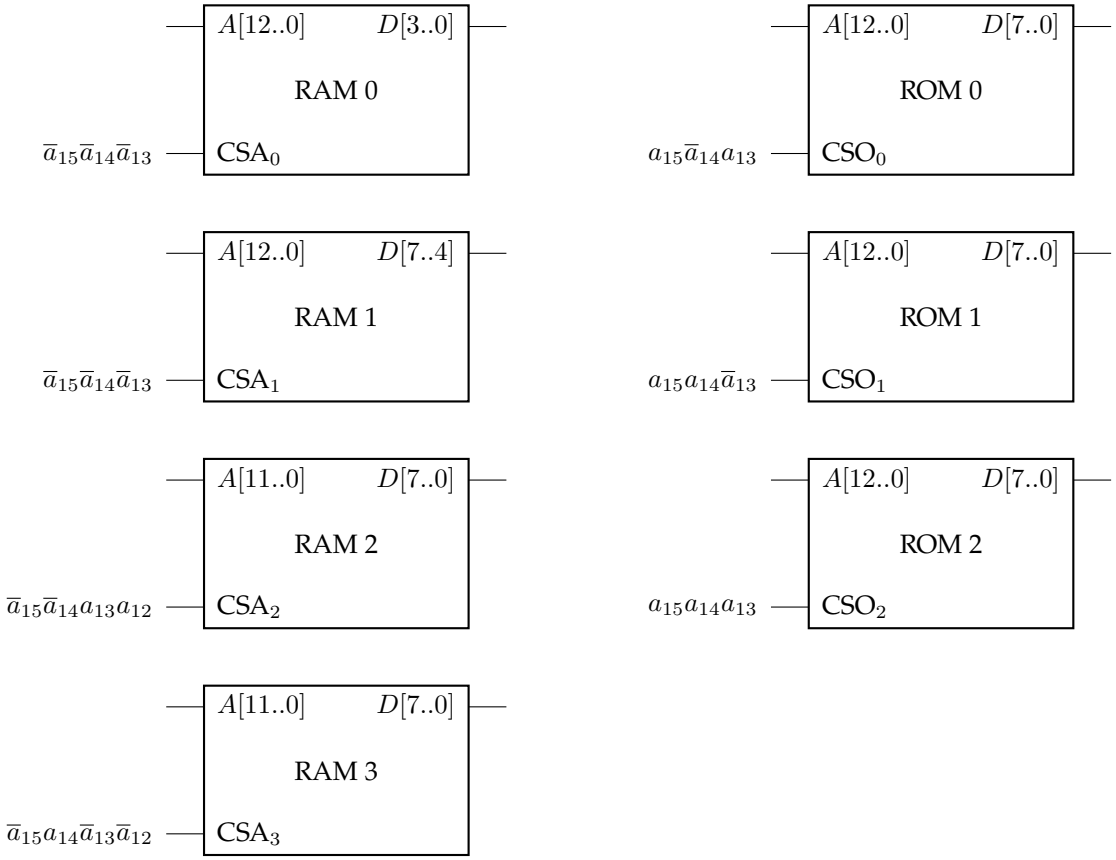
$$\begin{aligned}
\text{CSA}_0 &= \overline{a}_{15}\overline{a}_{14}\overline{a}_{13} \\
\text{CSA}_1 &= \overline{a}_{15}\overline{a}_{14}\overline{a}_{13} \\
\text{CSA}_2 &= \overline{a}_{15}\overline{a}_{14}a_{13}a_{12} \\
\text{CSA}_3 &= \overline{a}_{15}a_{14}\overline{a}_{13}\overline{a}_{12} \\
\text{CSO}_0 &= a_{15}\overline{a}_{14}a_{13} \\
\text{CSO}_1 &= a_{15}a_{14}\overline{a}_{13} \\
\text{CSO}_2 &= a_{15}a_{14}a_{13}
\end{aligned}$$

Each memory chip should be connected to the address and data buses as shown below. Note that this is the important part for the $8\,\text{K} \times 4$ RAM chips.

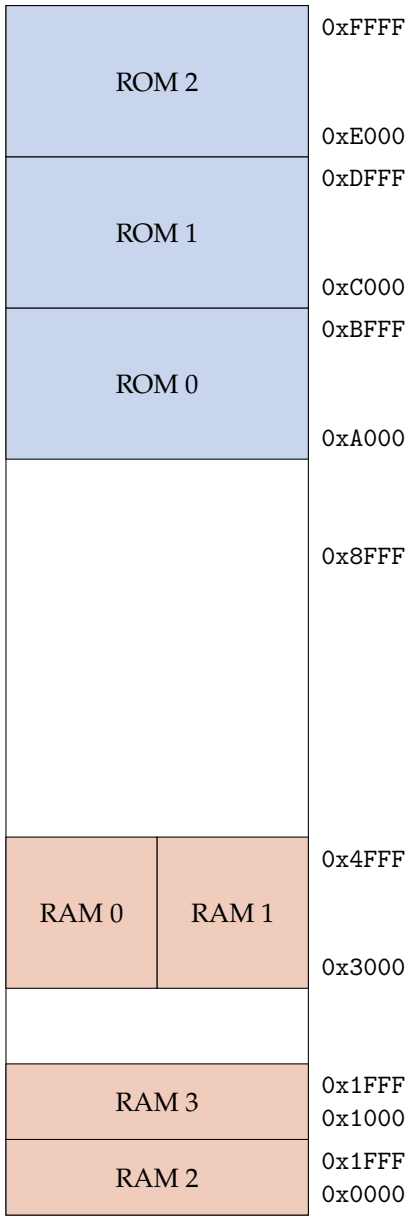
- As noted above, RAM 0 and RAM 1 share the same memory space, because they are each only *half as wide* as the data bus.
- Therefore, RAM 0 and RAM 1 chip should accept a *different half* of the available bits on the data bus.
- It really doesn't matter which bits go to which RAM chip — this is completely transparent to the user as it is impossible to read from a single RAM chip independently. Combining the two **nibbles** of data from each RAM chip to a single **byte** will be automatic.
- It is probably most sensible to give one RAM chip the data lines $D[3..0]$ and the other the data lines $D[7..4]$ as is done here, but really any combination is equally valid. [1]

[1] $D[1, 2, 6, 7]$ for one and $D[0, 3, 4, 5]$ for the other work just as well! Think about it.

As shown, it is fine to just write the logic expression for the chip selection rather than draw an diagram with actual logic gates. Please also pay careful attention to which bits are used for the addresses: the $4\,\mathrm{K} \times 8$ RAM chips only accept 12 address bits ($A[11..0]$) while all other chips accept 13 address bits ($A[12..0]$).



Alternate Solution: A more difficult solution involves using the $4\,\mathrm{K} \times 8$ RAM chips at the bottom of the memory space and the $8\,\mathrm{K} \times 4$ RAM chips in parallel starting at `0x3000`. This memory map is shown to the right. Here I have retained the same name (RAM 0, RAM 1, etc.) of the chips as before.
The ROM chips are all arranged the same was as previous, so they have the same solution for the chip select logic and chip wiring. The logic for RAM 2 and RAM 3 at the bottom of memory space is easy enough:



Chip	Position	Hex Address	Binary Address
RAM 2	Start	`0x0000`	0000 0000 0000 0000
	End	`0x0FFF`	0000 1111 1111 1111
RAM 3	Start	`0x1000`	0001 0000 0000 0000
	End	`0x1FFF`	0001 1111 1111 1111

Again, the bits highlighted in blue are the ones that are consistent over the entire data range, and should be used for the chip select logic.

$$\mathrm{CSA}_2 = \overline{a}_{15}\overline{a}_{14}\overline{a}_{13}\overline{a}_{12}$$
$$\mathrm{CSA}_3 = \overline{a}_{15}\overline{a}_{14}\overline{a}_{13}a_{12}$$

So far, so good. The problem comes when trying to find the logic for RAM 0 and RAM 1:

Chip	Position	Hex Address	Binary Address
RAM 0	Start	`0x3000`	0011 0000 0000 0000
	End	`0x4FFF`	0100 1111 1111 1111
RAM 1	Start	`0x3000`	0011 0000 0000 0000
	End	`0x1FFF`	0100 1111 1111 1111

The problem is that the only consistent bits across the address range is the most significant bit shown in blue.

- Many students in this situation are tempted to follow the pattern from previous and use $\text{CSA}_0 = \overline{a}_{15}$ as the chip select logic, but this **cannot be correct**!
- For one thing, using $\text{CSA}_0 = \overline{a}_{15}$ would cause RAM 0 to be active when trying to write to RAM 2 or RAM 3.
- For another thing, the entire memory range spanned by $A[14..0]$ is **not** available on the RAM chips.

These RAM chips have only 12-bits of address space. The original rule for chip select logic still applies: we need logic that determines when an address is within the range `0x3000` and `0x4FFF`. The way to do this is to break the memory range in two. Basically, instead of asking:

"is the memory address within the range `0x3000` to `0x4FFF`?"

we are asking:

"is the memory address within the range `0x3000` to `0x3FFF` **or** the range `0x4000` to `0x4FFF`?"

By phrasing the problem this way, we can find the proper chip select logic.

Chip	Position	Hex Address	Binary Address
RAM 0, RAM 1	Start	`0x3000`	0011 0000 0000 0000
	Middle-End	`0x3FFF`	0011 1111 1111 1111
	Middle-Start	`0x4000`	0100 0000 0000 0000
	End	`0x4FFF`	0100 1111 1111 1111

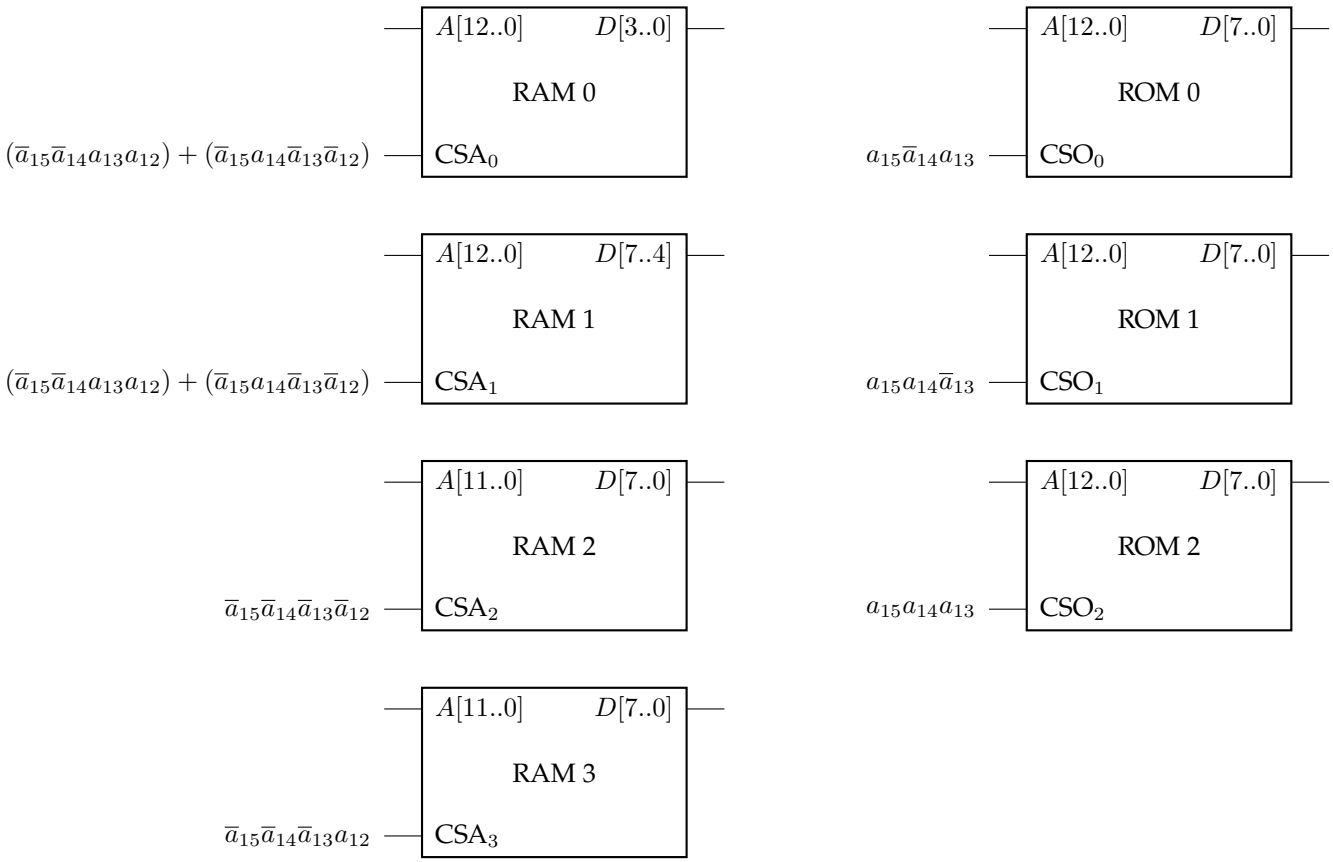
Instead of using a 3-`AND` as the chip select logic, we need to use a 2-`OR`, 4-`AND` expression:

$$\text{CSA}_{0,1} = (\overline{a}_{15}\overline{a}_{14}a_{13}a_{12}) + (\overline{a}_{15}a_{14}\overline{a}_{13}\overline{a}_{12})$$

This logic correctly selects **only** the addresses between `0x3000` and `0x4FFF`.

- Note that the address bit a_{12} must be used **twice**: as both part of the chip select logic, and as part of the internal chip addresses (an $8\,\text{K} \times 4$ chip needs $A[12..0]$ for the address bits.
- Note also that the "bottom addresses" `0x3000` to `0x3FFF` actually correspond to the *top half* of the *internal addresses* in the RAM chips. But this doesn't matter to a programmer and certainly doesn't matter to a user.

The revised chip-wiring diagram is below.


A[12..0]
D[3..0]
RAM 0
$(\overline{a}_{15}\overline{a}_{14}a_{13}a_{12}) + (\overline{a}_{15}a_{14}\overline{a}_{13}\overline{a}_{12})$
CSA$_0$
A[12..0]
D[7..4]
RAM 1
$(\overline{a}_{15}\overline{a}_{14}a_{13}a_{12}) + (\overline{a}_{15}a_{14}\overline{a}_{13}\overline{a}_{12})$
CSA$_1$
A[11..0]
D[7..0]
RAM 2
$\overline{a}_{15}\overline{a}_{14}\overline{a}_{13}\overline{a}_{12}$
CSA$_2$
A[11..0]
D[7..0]
RAM 3
$\overline{a}_{15}\overline{a}_{14}\overline{a}_{13}a_{12}$
CSA$_3$
A[12..0]
D[7..0]
ROM 0
$a_{15}\overline{a}_{14}a_{13}$
CSO$_0$
A[12..0]
D[7..0]
ROM 1
$a_{15}a_{14}\overline{a}_{13}$
CSO$_1$
A[12..0]
D[7..0]
ROM 2
$a_{15}a_{14}a_{13}$
CSO$_2$