# Computer Organización

Iván de Jesús Deras Tábora

March 4, 2019

Period 1 2019

MIPS32SOC Part 3

## Objetives

In this part of the project you'll add support for the following memory access instructions: `lb, lbu, sb, lh, lhu, sh`

## What you will need

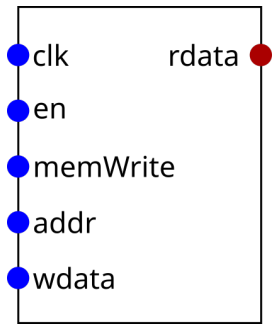
In order to developed this part of the project you'll need the following:

1. The second part of the project completed.
2. The Data Memory module
3. The VGA module
4. The Font ROM for the VGA module
5. The test cases for the processor.

The last 4 items will be provided to you.

### The Data Memory

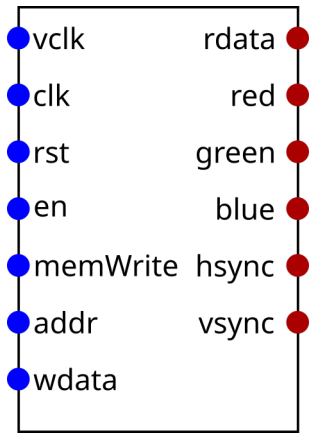
The data memory module has the following schematic:



| Signal | Type | Description | Bits |
|---|---|---|---|
| `clk` | Input | Positive edge clock input. | 1 |
| `en` | Input | Chip enable signal. If this signal is **0** you cannot **read** or **write** to the memory. | 1 |
| `memWrite` | Input | Per byte memory write enable signal. This signal uses one bit per byte, this enable writting of individual bytes in the word. | 4 |
| `addr` | Input | The word address to read of write. | 32 |
| `wdata` | Input | Write data | 32 |
| `rdata` | Output | Read data | 32 |

### The VGA module

The VGA module has the following schematic:




| Signal | Type | Description | Bits |
|---|---|---|---|
| `vclk` | Input | Positive edge VGA clock input (should be 25MHz). | 1 |
| `clk` | Input | Positive edge clock input. This clock input synchronizes the **writes** to the VGA RAM. | 1 |
| `en` | Input | Chip enable signal. If this signal is **0** you cannot **read** or **write** to the memory. | 1 |
| `memWrite` | Input | Per byte memory write enable signal. This signal uses one bit per byte, this enable writting of individual bytes in the word. | 4 |
| `addr` | Input | The word address to read of write. | 32 |
| `wdata` | Input | Write data | 32 |
| `rdata` | Output | Read data | 32 |
| `red` | Output | VGA **red** output. | 3 |
| `green` | Output | VGA **green** output. | 3 |
| `blue` | Output | VGA **blue** output. | 2 |
| `hsync` | Output | VGA **hsync** output. | 1 |
| `vsync` | Output | VGA **vsync** output. | 1 |

Take into account that everything you write to this memory will be displayed in a graphic window during simulation. When running on the FPGA the data will displayed in a VGA monitor.

## Your task

To achieve your goal you'll have to add two modules and modify two others. The modules to add are:

1. Memory Read Data Decoder

2. Memory Write Data Encoder

The modules to modify are:

1. Memory Decoder
2. Control Unit

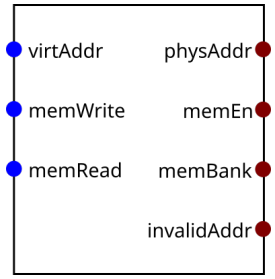
You are free to change the rest of the modules, but this modules are a requirement. Take into account that this modules will be developed in Verilog.

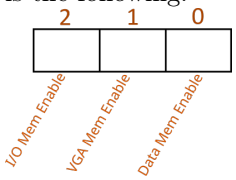
## Memory Decoder

The memory decoder circuit will decide which memory bank to use. The available memory banks are:

- Data Memory
- VGA memory
- I/O memory. Remember this is used to allow memory mapped I/O to the software.

The circuit has the following schematic:




| Signal | Type | Description | Bits |
|---|---|---|---|
| `virtAddr` | Input | The input virtual address. | 32 |
| `memWrite` | Input | If this signal is 1 there is a write request to memory. | 1 |
| `memRead` | Input | If this signal is 1 there is a read request from memory. | 1 |
| `physAddr` | Output | The output physical address. | 13 |
| `memEn` | Output | Memory enable signal. The structure of this signal is the following:<br>  <br><br>In this signal never two bits can be set at the same time. **Note:** If there is no **write** or **read** request this signal should be **0**, that means all memory banks are disabled. | 3 |
| `memBank` | Output | Indicates the memory bank to access. 0 for Data memory, 1 for VGA memory, 2 for I/O memory. | 2 |
| `invalidAddr` | Output | Invalid address. This signal indicate if the input address is invalid. **Note:** If there is not write or read request this signal should be 0. | 1 |

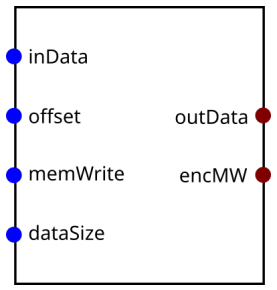
### Virtual addresses

Memory access instructions will use virtual address when accessing the memory. The valid ranges are the following:

- `0x10010000 - 0x10011000` to access global data.
- `0x7FFFEFFC - 0x7FFFFFFC` to access the stack.
- `0xB800 - 0xCACF` to access the VGA memory.
- `0xFFFF0000 - 0xFFFF000C` to access I/O memory. Remember this is to allow memory mapped I/O to the software. Only read operations are allowed, for now any read access to this addresses should return 0.

## Memory Write Data Encoder

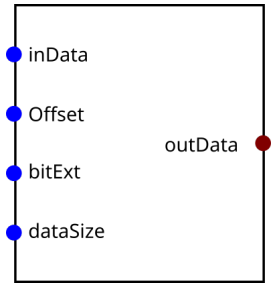
This circuit will prepare the data that needs to be written to memory. The circuit has the following diagram:




| Signal | Type | Description | Bits |
|---|---|---|---|
| `inData` | Input | The input data. | 32 |
| `ofsset` | Input | The byte or half word offset. 0, 1, 2, 3 for byte acces. 0, 2 for half word access. | 2 |
| `memWrite` | Input | The input memory write enable signal. | 1 |
| `dataSize` | Input | The data size to access. 0 for Word, 1 for Half Word and 2 for Byte. | 2 |
| `outData` | Output | The output encoded data. The value will depend on the data size. For example if the data size is `word` the output data will be the same as the input data. But if we want to write for example the first half word into memory, the input data will have the half word into lower 16 bits. In that case the output data will have the lower 16 bits of the input data shifted to right (16 bits). That is we have to move the half word to right position. The same applies to byte. | 32 |
| `encMW` | Output | The encoded memory write enable signal. Every bit indicates the byte we want to write in the memory word. For example if we want to write the second byte in the word, this signal will be $\mathbf{0100_2}$. If we want to write the whole word the signal will be $\mathbf{1111_2}$ in binary. | 4 |

## Memory Read Data Decoder

This circuit will decode the data read from memory. The circuit has the following diagram:



| Signal | Type | Description | Bits |
|---|---|---|---|
| `inData` | Input | The input data. This will be the data read from memory. | 32 |
| `ofsset` | Input | The byte or half word offset. 0, 1, 2, 3 for byte acces. 0, 2 for half word access. | 2 |
| `bitExt` | Input | If this signal is **0**, perform a sign extend, else perform a zero extend. | 1 |
| `dataSize` | Input | The data size to access. 0 for Word, 1 for Half Word and 2 for Byte. | 2 |
| `outData` | Output | The output decoded data. For byte or half word we have to extract the right part (depending on the offset) from the input data. Ex. if `inData` is `0xaabbccdd` `ofsset` is 1 and `bitExt` is 0, then `outData` will be `0x000000bb` | 32 |

## Control Unit

You'll have to add the following signals to the control unit:

| Signal | Description | Bits |
|---|---|---|
| `memDataSize` | Indicates the size of the data to read or write to memory. This is in the case of memory access instructions `lw, sw, lh, lhu, sh, lb, lbu, sb`. 0 is word, 1 is half word and 2 is byte. | 2 |
| `memBitExt` | In the case of a memory read, this signals indicate the type of extension needed. **0** for sign extension and **1** for zero extension. | 1 |

## Testing

For the testing part you'll receive individual test cases for every instruction added to the processor.