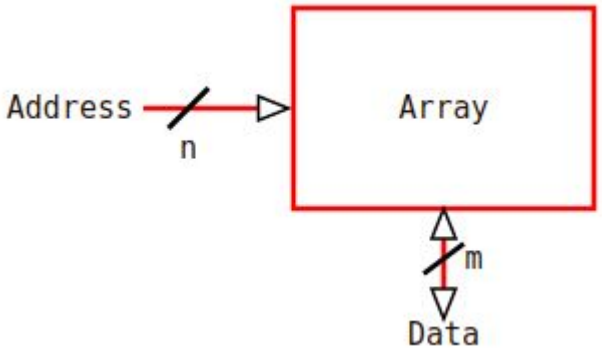
Register Files, RAM, and ROM in Verilog

Instructor: Dr. Vinicius Prado da Fonseca (vpradodafons@online.mun.ca)

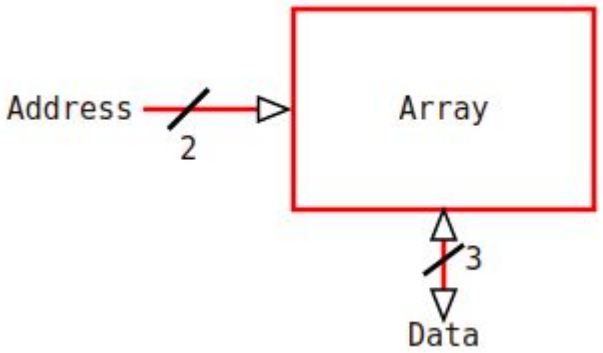
Memory Arrays

- m-bits - word, data size
  - 32-bits
  - 64-bits
- n-bits - address size
- 2^n Addresses/items/words
- Memory size: 2^n x m bits
- Often viewed as an array
  - Sequence of words
  - Address are index of the array
- On a given address
  - Data is read from an address
  - Written to an address



Memory Arrays

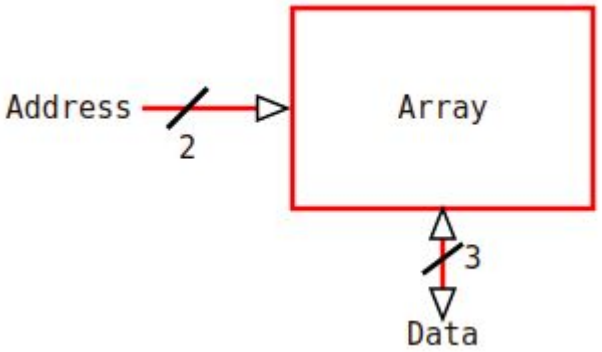
- Most memory is either read or written at one time at a given address
- Memory is implemented with three technologies:
  - ROM - Read Only Memory
  - DRAM - Dynamic Random Access Memory
  - SRAM - Static Random Access Memory
- Historically confusing names:
  - ROM - Non volatile
  - RAM - Volatile
  - Static RAM (no refresh, cache memory)
  - Dynamic RAM (refresh, main memory)
- Random refers to having same access time for any position.
  - not sequential such as tapes.



Memory Arrays

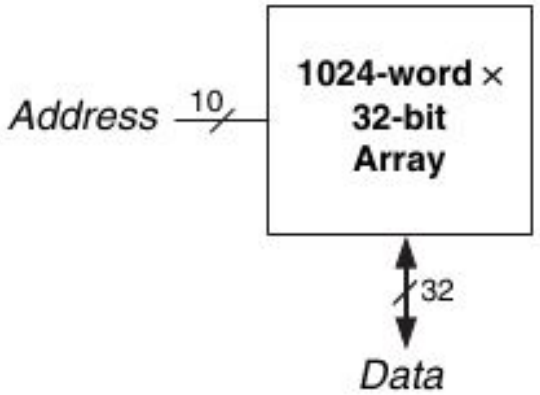
- A memory arranged as 4 words (2 bits for addresses) with 3 bits per word has a total of 12-bits
- 4 addresses = 2^2 = two bits for address
- 2^2x3 = 4x3 = 12 bits

		DATA		
ADDRESS	00			
	01			
	10			
	11			



Memory Arrays

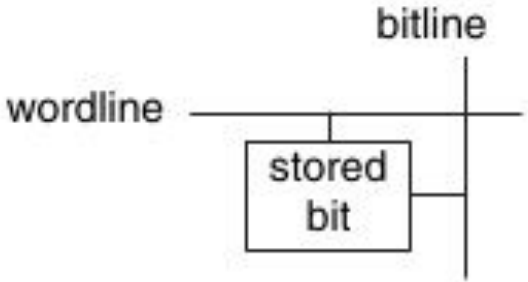
- A memory arranged as 1024 words (10 bits for addresses) with 32 bits per word has 32-kbit.
- 2^{10} x 32 = 1024 * 32 = 32,768 = 32 kbits.



https://www.aqua-calc.com/page/powers-of-two

Bit cells

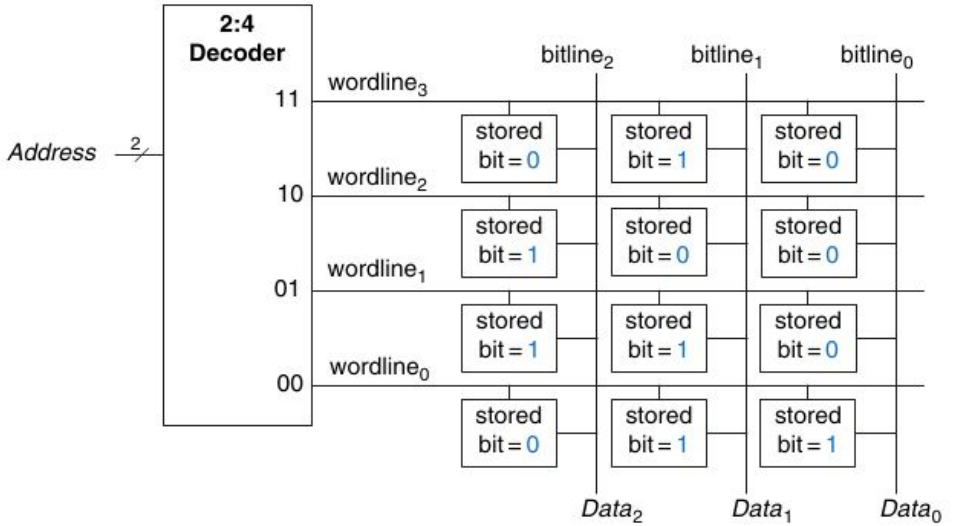
- Read:
  - Bit-line is floating (accepts any values)
  - Word-line is asserted
  - Data output receive stored value
- Write:
  - Bit-line has HIGH or LOW with the data input
  - Then Word-line is asserted to store the data in that address



Organization

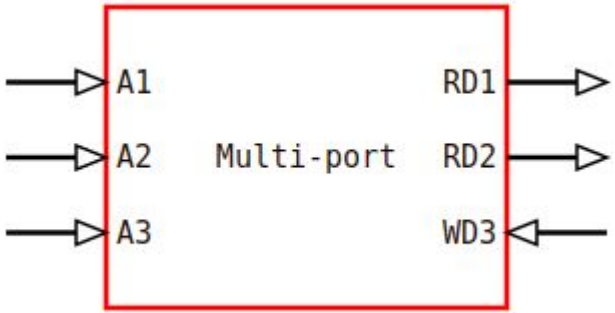
For example:

- Decoder activates only one line
- Read Address 10:
  - Activate word line$_2$
  - Data output = 100
- Write 001 to Address 11
  - bit line$_0$ = 1
  - bit line$_1$ = 0
  - bit line$_2$ = 0
  - activates word line$_3$



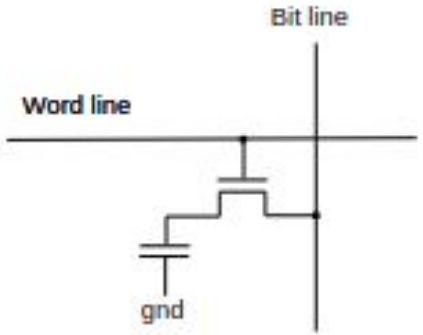
Multi-port Memory

- More ports allow parallel access for reading and writing.
- An example memory that has two read ports and one write port
- Memories in graphics cards (GPU) often allow both reading and writing at the same time.
    - Data is read for display
    - Written to update the contents of the display.



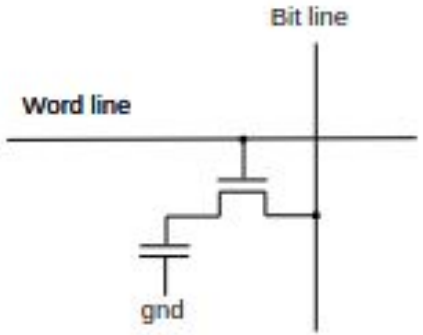
DRAM

- Most common memory for main memories on computers.
- A capacitor (stores a charge) is use to store a bit.
- One transistor is used to access the capacitor
  - Word line is HIGH activates the NMOS
  - Copy the capacitor to the bit line
- DRAM is the slowest technology, but also the densest
- Requires refresh of the capacitor



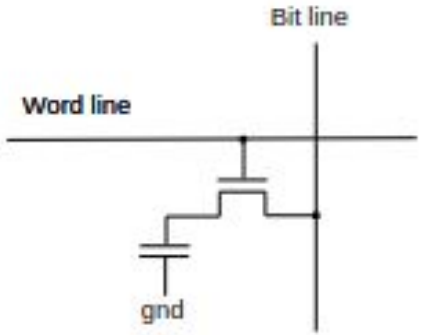
DRAM

- Read:
  - Bit line floating
  - Word line HIGH
  - Transistor closes
  - Capacitor copied to bitline
- Write
  - Bit line with HIGH or LOW
  - Word line HIGH
  - Transistor closes
  - Data copied to capacitor



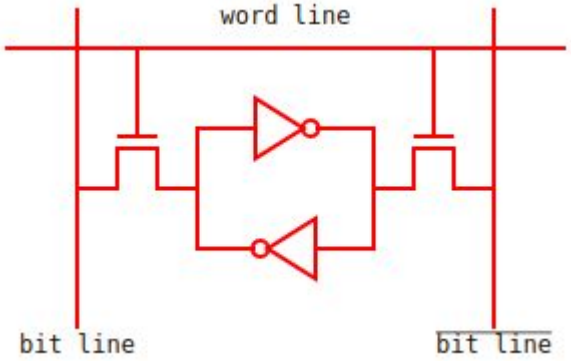
DRAM

- Reading destroys the bit value stored on the capacitor
- Data word must be restored (rewritten) after each read
- Even when DRAM is not read, the contents must be refreshed (read and rewritten)
- Every few milliseconds
- The charge on the capacitor gradually leaks away



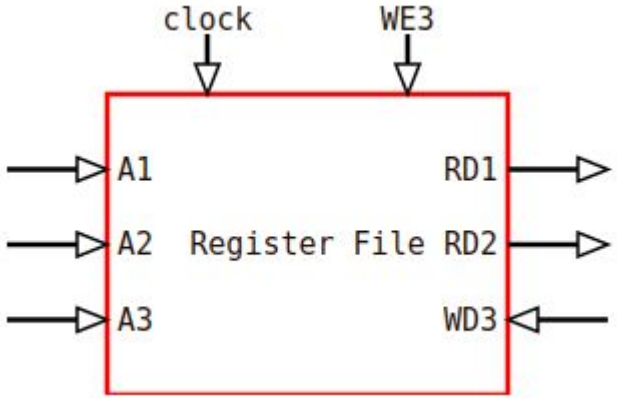
SRAM

- A SRAM bit requires six transistors
- Two transistors to connect bit lines
- Two inverters
  - two transistors each to create a bi-stable circuit.
  - Section 5.5.3
    - Section 3.2
- The word line activates
- The transistors connect the inverters to the bit lines.
- A SRAM cell is faster but less dense
- Do not require refresh



Register Files

- A register file is constructed with set of D flip-flops.
- Each flip-flop requires 20 transistors, but is faster than SRAM or DRAM.
- Most data paths use a register file to hold data.
- A two read ports and one write port register file
    - Read both operands (A+B) at same time



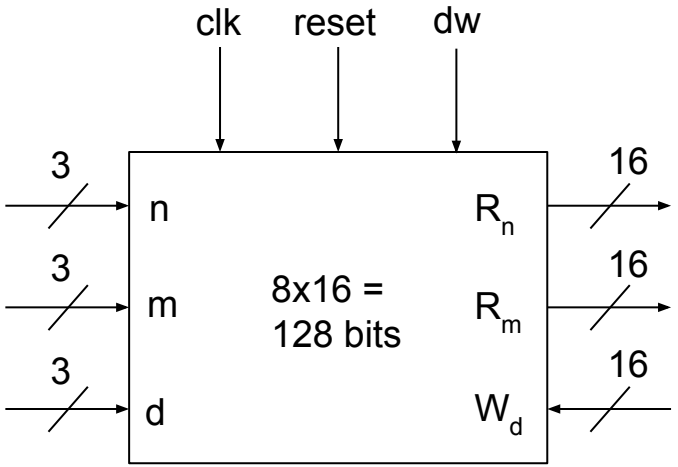
Register Files

- Each word is 16 bits long

```
typedef logic [15:0] reg16_t;
```

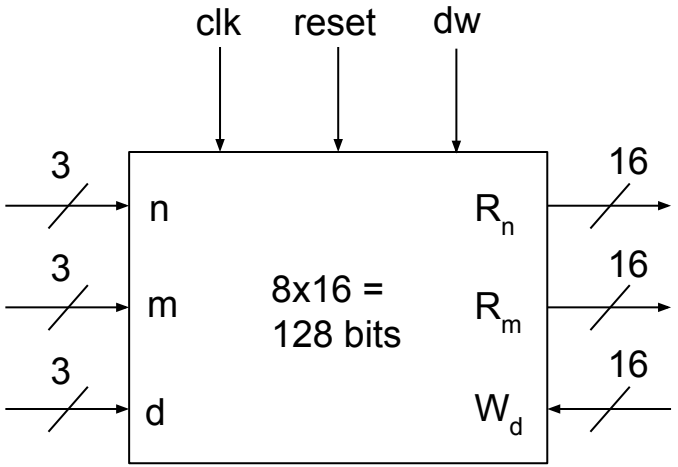
- Selector 3 bits -> 8 addresses

```
typedef logic [2:0] reg_sel_t;
```



Register Files

- output
  - reg16_t rn
  - reg16_t rm
- input
  - reg16_t rd
  - reg_sel_t n
  - reg_sel_t m
  - reg_sel_t d
  - logic dw
  - logic reset
  - logic clk



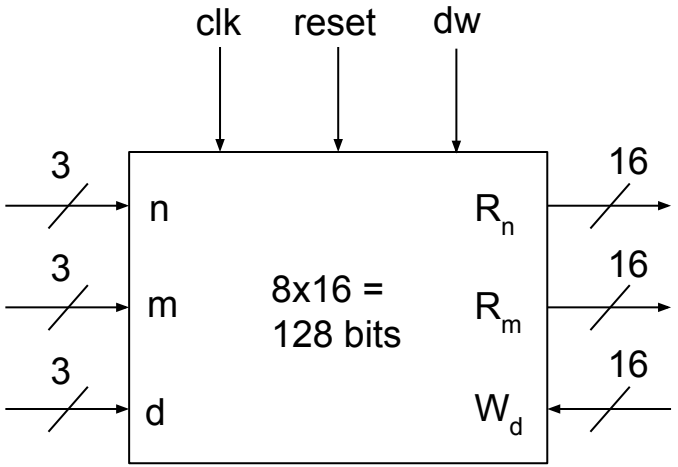
Register Files

- Actual 8x16 register:

```
reg16_t R[7:0];
```

- Two independent read outputs
- Each output connected and indexed:

```
assign rn = R[n];
assign rm = R[m];
```



Register Files

- Actual 8x16 register:

```
reg16_t R[7:0];
```

- Two independent read outputs
- Each output connected and indexed:

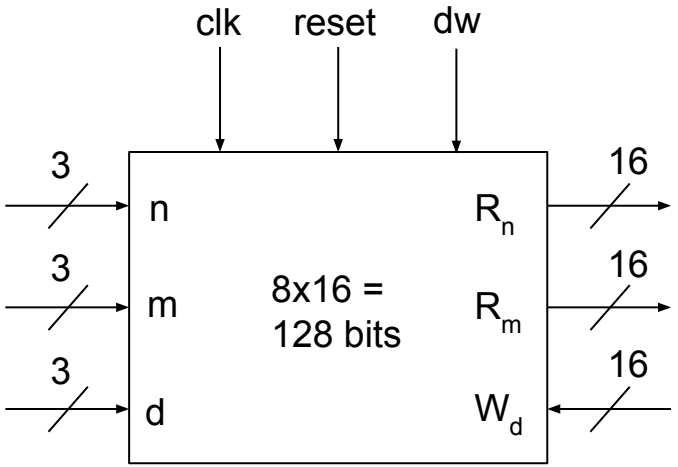
```
assign rn = R[n];
assign rm = R[m];
```

- `adr = n/m`

adr	16-bits word
000	0000000000000000
001	0000000000000000
010	0000000000000000
011	0000000000000000
100	0000000000000000
101	0000000000000000
110	0000000000000000
111	0000000000000000

Register Files

```
always_ff @(posedge clk) begin
    if ( reset ) begin
        for(i=0; i<$size(R); i++) begin
            R[i] <= 0;
        end
    end
    else if ( dw ) R[d] <= rd;
end
```



RAM Verilog

- `module ram #(parameter N=6, M=32)`
- `input`
  - `clk`
  - `we`
  - `[N-1:0] adr`
  - `[M-1:0] din`
- `output`
  - `[M-1:0] dout`

adr (64, 6-bit) 2**6	32-bit word
000000	00000000000000000000000000000000
000001	00000000000000000000000000000000
000010	00000000000000000000000000000000
000011	00000000000000000000000000000000
...	...
111111	00000000000000000000000000000000

RAM Verilog

```
// Actual memory
logic [M-1:0] mem[ 2**N-1:0];

// Positive clock edge
always_ff @(posedge clk)
      if ( we ) mem[adr] <= din;

assign dout = mem[adr];
```

adr (64, 6-bit) 2**6	32-bit word
000000	00000000000000000000000000000000
000001	00000000000000000000000000000000
000010	00000000000000000000000000000000
000011	00000000000000000000000000000000
...	...
111111	00000000000000000000000000000000

RAM Verilog

```
module ram #(parameter N=6, M=32)(input logic clk,
                                  input logic we,
                                  input logic [N-1:0] adr,
                                  input logic [M-1:0] din,
                                  output logic [M-1:0] dout);
    logic [M-1:0] mem[ 2**N-1:0]; // memory
    always_ff @(posedge clk)  // Positive clock edge
         if ( we ) mem[adr] <= din;
    assign dout = mem[adr];
endmodule
```

- Example in the notes

ROM Verilog

```
always_comb
    case (adr)
        0 : dout = 3'b011;
        1 : dout = 3'b110;
        2 : dout = 3'b100;
        3 : dout = 3'b010;
    endcase
```

adr	3-bit word
00	011
01	110
10	100
11	010

- Example in the notes

Questions?

- Next steps
  - Control logic and data paths
  - The design of most computers consist of a data path and control logic