# Final Project

## RAM

wen, waddr, wdata } inputs

raddr input

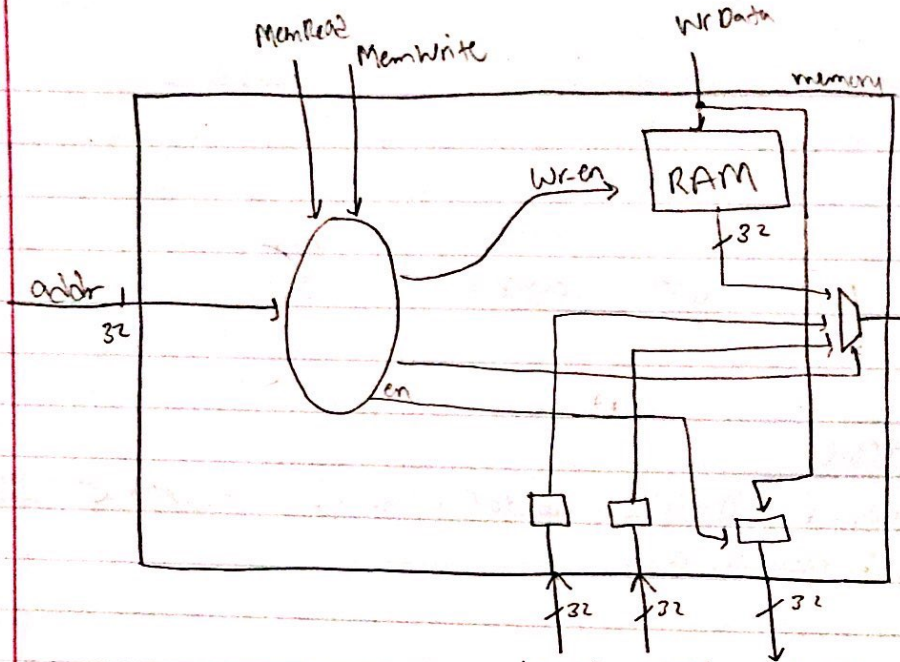
rdata output

### async-Read architecture

```
type memory_array is array (0 to num_words-1) of std_vector (word_width-1 downto 0)
signal memory : memory_array
begin
  process (clk)
    if (rising_edge(clk)) then
      if (wen = '1') then
        memory(to_integer(unsigned(waddr))) <= wdata;
      end if;
  end process;
    rdata <= memory (to_integer(unsigned(raddr)));
```

- Block RAM only supports sync reads.
- use RAM entity posted on website if ever needed

- register file on website is not the one for project just an example.
- in write logic, on if statement, add one statement to write to any register whose address is not 0.




* translate byte address to word address
Shift Right by 2 then pass address into RAM

Should mem address be < 255 or 1024 since we're shifting

- if addr is from 0-255, you want to select RAM
- RAM has one cycle latency when compared to inports, delay inports to match timing.
- RegB from schematic ≙ Wrdata signal from above...

- Controller

IR = RAM[PC]
PC = PC+4
IorD = '0' passes address from PC to memory
MemRead = '1'
wait for some amount of time
IRWrite = '1'
} IR = RAM[PC]

ALUSrcA = '0' so it selects PC
ALUSrcB = '1' to select 4
PCSource = '0'
PCWrite = '1'
} PC = PC+4

Instruction Decode stage just check what kind of instruction it is by checking OP code (top 6 bits)

```
if (31-26 = 0) then
    case bits(5-0)
else
    case bits(31-26)
        case ALUOP
```

ALUOP = bits 31-26

To tell ALU to do an operation (ALU control) do as pure comb. logic

ALUSrcA = '1'

ALUSrcB = '0'

bottom six bits specify what operation to perform (eg. add, sub, etc)

ALUOp specifies what category of operations we want.

configure ALUselect

wait 1 cycle

ALU_LO_HI = '0'

MemtoReg = '0'

rd (is dont 11) specifies what address to write to

RegDst = 1, RegWrite = 1

declare constants for ALU constants so that TA can see name of operation rather than value of select lines

- R-type instructions use two values from two registers, does an operation with the values and stores the result into another register.
- PC+4 happens during each instruction

- I-type Instructions
  - rt: destination register (different field from r-type) (bits 20-16)
  - second value doesn't come from register in register file. It comes directly from instruction register
  - second value specified in bits 15-0

- For branch, TARGET value comes from offset section of mips instruction
- When doing a branch, must get value from ALU-OUT reg so that you can calculate the whether branch is taken while preserving value.
- For two branch instructions w/ same opcode of 1, we must use bits 20-16 to differentiate. Need to add extra control signal reading from those bits.