REGISTERS!

Registers are **quickly** accessible memory locations available to the processor. The size of the register depends on the CPU architecture. Common sizes are 16 bits, 32 bits and 64 bits but they can be larger. The size of registers on a machine is called the **word** size

WHY DO WE NEED REGISTERS?

WHY DO WE NEED REGISTERS?

We need to keep track of the state of the processor and the state of the program running

WHY DO WE NEED REGISTERS?

We need to keep track of the state of the processor and the state of the program running

We need to store "local" copies of the data that the ALU is working on

WHY DO WE NEED REGISTERS?

We need to keep track of the state of the processor and the state of the program running

We need to store "local" copies of the data that the ALU is working on

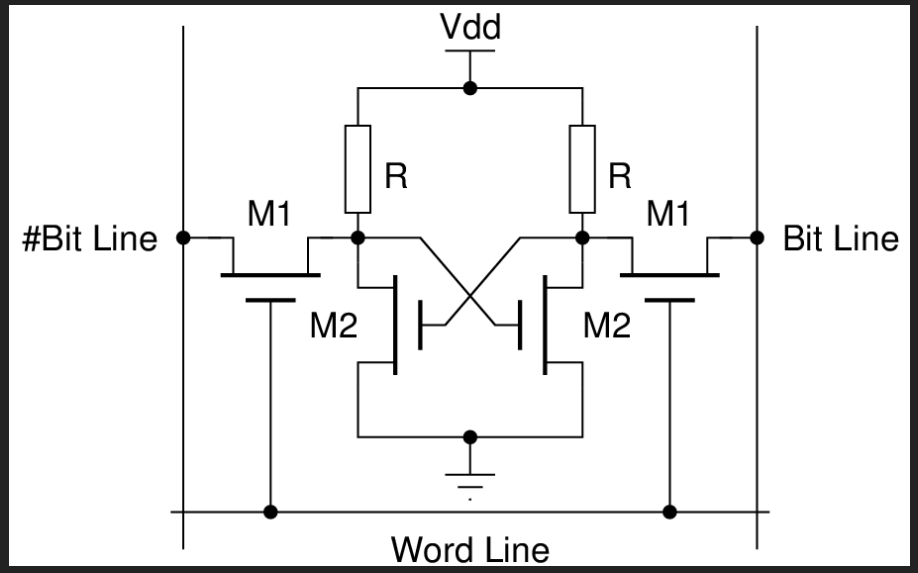
But isn't RAM enough?

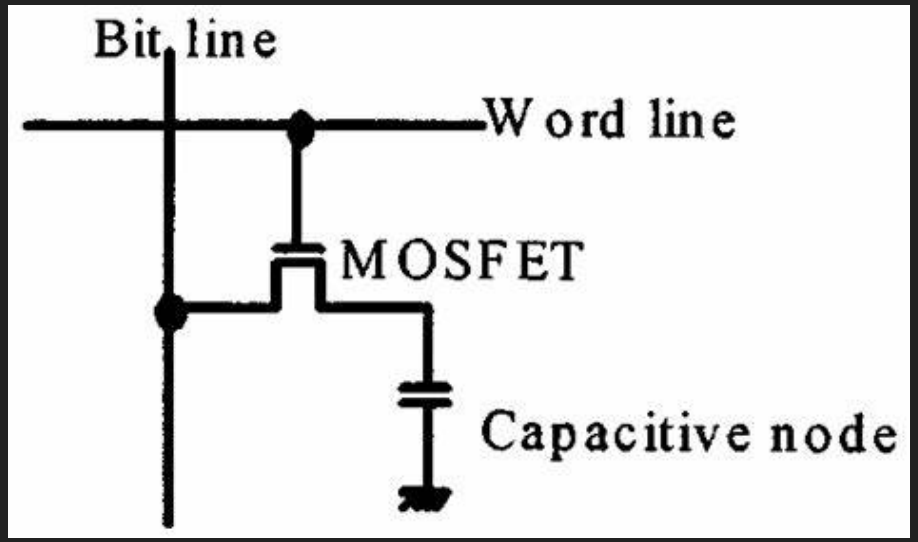
- Relative to the processor's clock speed, reading from and writing to RAM is slow
- Clock speeds are in the **GIGA**hertz, potentially meaning more than 1 billion instructions can be executed in a second!
- Alot of time consuming actions have to be done before RAM can be accessed...

- Relative to the processor's clock speed, reading from and writing to RAM is slow
- Clock speeds are in the **GIGA**hertz, potentially meaning more than 1 billion instructions can be executed in a second!
- Alot of time consuming actions have to be done before RAM can be accessed...

But that's a presentation for another day

Just for your information. Static RAM technology vs Dynamic RAM technology





TYPES OF REGISTERS

The two types of registers we will be looking at are the general purpose registers and the special purpose registers

GENERAL PURPOSE REGISTERS

Essentially just variables for the processor. They can be used to store anything that the ALU is working on and even the results

e.g The accumulator

SPECIAL PURPOSE REGISTERS

These registers are used to aid in the execution of instructions and keep track of the state of the processor

e.g the program counter

THE PROGRAM COUNTER

Simply put the program counter stores the address of the next instruction to be fetched, decoded and executed. Used to facilitate iteration

THE MEMORY DATA REGISTER

This special register stores the data fetched from main memory and the data which will be written to the main memory. Acts as a **buffer**

Example of the program counter using an actual program

Let's look at a simple program

```
Dim i as Integer = 10
Dim j as Integer = 1
j = j + i
```

```
 1 //address of instruction    opcode  operands
 2 0x0000555555555125 <+0>:     push   rbp
 3 0x0000555555555126 <+1>:     mov    rbp,rsp
 4 0x0000555555555129 <+4>:     mov    DWORD PTR [rbp-0x4],0xa
 5 0x0000555555555130 <+11>:    mov    DWORD PTR [rbp-0x8],0x1
 6 0x0000555555555137 <+18>:    mov    eax,DWORD PTR [rbp-0x4]
 7 0x000055555555513a <+21>:    add    DWORD PTR [rbp-0x8],eax
 8 0x000055555555513d <+24>:    mov    eax,DWORD PTR [rbp-0x8]
 9 0x0000555555555140 <+27>:    pop    rbp
10 0x0000555555555141 <+28>:    ret
```

```
 1 //address of instruction    opcode  operands
 2 0x0000555555555125 <+0>:     push   rbp
 3 0x0000555555555126 <+1>:     mov    rbp,rsp
 4 0x0000555555555129 <+4>:     mov    DWORD PTR [rbp-0x4],0xa
 5 0x0000555555555130 <+11>:    mov    DWORD PTR [rbp-0x8],0x1
 6 0x0000555555555137 <+18>:    mov    eax,DWORD PTR [rbp-0x4]
 7 0x000055555555513a <+21>:    add    DWORD PTR [rbp-0x8],eax
 8 0x000055555555513d <+24>:    mov    eax,DWORD PTR [rbp-0x8]
 9 0x0000555555555140 <+27>:    pop    rbp
10 0x0000555555555141 <+28>:    ret
```

```
 1 //address of instruction    opcode  operands
 2 0x0000555555555125 <+0>:     push   rbp
 3 0x0000555555555126 <+1>:     mov    rbp,rsp
 4 0x0000555555555129 <+4>:     mov    DWORD PTR [rbp-0x4],0xa
 5 0x0000555555555130 <+11>:    mov    DWORD PTR [rbp-0x8],0x1
 6 0x0000555555555137 <+18>:    mov    eax,DWORD PTR [rbp-0x4]
 7 0x000055555555513a <+21>:    add    DWORD PTR [rbp-0x8],eax
 8 0x000055555555513d <+24>:    mov    eax,DWORD PTR [rbp-0x8]
 9 0x0000555555555140 <+27>:    pop    rbp
10 0x0000555555555141 <+28>:    ret
```

```
 1 //address of instruction    opcode  operands
 2 0x000055555555125 <+0>:     push    rbp
 3 0x000055555555126 <+1>:     mov     rbp,rsp
 4 0x000055555555129 <+4>:     mov     DWORD PTR [rbp-0x4],0xa
 5 0x000055555555130 <+11>:    mov     DWORD PTR [rbp-0x8],0x1
 6 0x000055555555137 <+18>:    mov     eax,DWORD PTR [rbp-0x4]
 7 0x00005555555513a <+21>:    add     DWORD PTR [rbp-0x8],eax
 8 0x00005555555513d <+24>:    mov     eax,DWORD PTR [rbp-0x8]
 9 0x000055555555140 <+27>:    pop     rbp
10 0x000055555555141 <+28>:    ret
```

```
 1 //address of instruction    opcode  operands
 2 0x0000555555555125 <+0>:     push   rbp
 3 0x0000555555555126 <+1>:     mov    rbp,rsp
 4 0x0000555555555129 <+4>:     mov    DWORD PTR [rbp-0x4],0xa
 5 0x0000555555555130 <+11>:    mov    DWORD PTR [rbp-0x8],0x1
 6 0x0000555555555137 <+18>:    mov    eax,DWORD PTR [rbp-0x4]
 7 0x000055555555513a <+21>:    add    DWORD PTR [rbp-0x8],eax
 8 0x000055555555513d <+24>:    mov    eax,DWORD PTR [rbp-0x8]
 9 0x0000555555555140 <+27>:    pop    rbp
10 0x0000555555555141 <+28>:    ret
```

```
1  //address of instruction    opcode  operands
2  0x000055555555125 <+0>:     push    rbp
3  0x000055555555126 <+1>:     mov     rbp,rsp
4  0x000055555555129 <+4>:     mov     DWORD PTR [rbp-0x4],0xa
5  0x000055555555130 <+11>:    mov     DWORD PTR [rbp-0x8],0x1
6  0x000055555555137 <+18>:    mov     eax,DWORD PTR [rbp-0x4]
7  0x00005555555513a <+21>:    add     DWORD PTR [rbp-0x8],eax
8  0x00005555555513d <+24>:    mov     eax,DWORD PTR [rbp-0x8]
9  0x000055555555140 <+27>:    pop     rbp
10 0x000055555555141 <+28>:    ret
```

```
1  //address of instruction    opcode  operands
2  0x000055555555125 <+0>:     push    rbp
3  0x000055555555126 <+1>:     mov     rbp,rsp
4  0x000055555555129 <+4>:     mov     DWORD PTR [rbp-0x4],0xa
5  0x000055555555130 <+11>:    mov     DWORD PTR [rbp-0x8],0x1
6  0x000055555555137 <+18>:    mov     eax,DWORD PTR [rbp-0x4]
7  0x00005555555513a <+21>:    add     DWORD PTR [rbp-0x8],eax
8  0x00005555555513d <+24>:    mov     eax,DWORD PTR [rbp-0x8]
9  0x000055555555140 <+27>:    pop     rbp
10 0x000055555555141 <+28>:    ret
```

Jumps

```
For i = 0 To 5
    j = j + 5
Next
```

```
 1 0x000055555555125 <+0>:     push   rbp
 2 0x000055555555126 <+1>:     mov    rbp,rsp
 3 0x000055555555129 <+4>:     mov    DWORD PTR [rbp-0x4],0x0
 4 0x000055555555130 <+11>:    mov    DWORD PTR [rbp-0x8],0x0
 5 0x000055555555137 <+18>:    jmp    0x555555555141
 6 0x000055555555139 <+20>:    add    DWORD PTR [rbp-0x8],0x5
 7 0x00005555555513d <+24>:    add    DWORD PTR [rbp-0x4],0x1
 8 0x000055555555141 <+28>:    cmp    DWORD PTR [rbp-0x4],0x4
 9 0x000055555555145 <+32>:    jle    0x555555555139
10 0x000055555555147 <+34>:    mov    eax,DWORD PTR [rbp-0x8]
11 0x00005555555514a <+37>:    pop    rbp
12 0x00005555555514b <+38>:    ret
13
```

```
 1 0x000055555555125 <+0>:     push   rbp
 2 0x000055555555126 <+1>:     mov    rbp,rsp
 3 0x000055555555129 <+4>:     mov    DWORD PTR [rbp-0x4],0x0
 4 0x000055555555130 <+11>:    mov    DWORD PTR [rbp-0x8],0x0
 5 0x000055555555137 <+18>:    jmp    0x555555555141
 6 0x000055555555139 <+20>:    add    DWORD PTR [rbp-0x8],0x5
 7 0x00005555555513d <+24>:    add    DWORD PTR [rbp-0x4],0x1
 8 0x000055555555141 <+28>:    cmp    DWORD PTR [rbp-0x4],0x4
 9 0x000055555555145 <+32>:    jle    0x555555555139
10 0x000055555555147 <+34>:    mov    eax,DWORD PTR [rbp-0x8]
11 0x00005555555514a <+37>:    pop    rbp
12 0x00005555555514b <+38>:    ret
13
```

```
 1 0x000055555555125 <+0>:     push   rbp
 2 0x000055555555126 <+1>:     mov    rbp,rsp
 3 0x000055555555129 <+4>:     mov    DWORD PTR [rbp-0x4],0x0
 4 0x000055555555130 <+11>:    mov    DWORD PTR [rbp-0x8],0x0
 5 0x000055555555137 <+18>:    jmp    0x555555555141
 6 0x000055555555139 <+20>:    add    DWORD PTR [rbp-0x8],0x5
 7 0x00005555555513d <+24>:    add    DWORD PTR [rbp-0x4],0x1
 8 0x000055555555141 <+28>:    cmp    DWORD PTR [rbp-0x4],0x4
 9 0x000055555555145 <+32>:    jle    0x555555555139
10 0x000055555555147 <+34>:    mov    eax,DWORD PTR [rbp-0x8]
11 0x00005555555514a <+37>:    pop    rbp
12 0x00005555555514b <+38>:    ret
13
```

```
 1 0x000055555555125 <+0>:     push   rbp
 2 0x000055555555126 <+1>:     mov    rbp,rsp
 3 0x000055555555129 <+4>:     mov    DWORD PTR [rbp-0x4],0x0
 4 0x000055555555130 <+11>:    mov    DWORD PTR [rbp-0x8],0x0
 5 0x000055555555137 <+18>:    jmp    0x555555555141
 6 0x000055555555139 <+20>:    add    DWORD PTR [rbp-0x8],0x5
 7 0x00005555555513d <+24>:    add    DWORD PTR [rbp-0x4],0x1
 8 0x000055555555141 <+28>:    cmp    DWORD PTR [rbp-0x4],0x4
 9 0x000055555555145 <+32>:    jle    0x555555555139
10 0x000055555555147 <+34>:    mov    eax,DWORD PTR [rbp-0x8]
11 0x00005555555514a <+37>:    pop    rbp
12 0x00005555555514b <+38>:    ret
13
```

```
 1 0x000055555555125 <+0>:     push   rbp
 2 0x000055555555126 <+1>:     mov    rbp,rsp
 3 0x000055555555129 <+4>:     mov    DWORD PTR [rbp-0x4],0x0
 4 0x000055555555130 <+11>:    mov    DWORD PTR [rbp-0x8],0x0
 5 0x000055555555137 <+18>:    jmp    0x555555555141
 6 0x000055555555139 <+20>:    add    DWORD PTR [rbp-0x8],0x5
 7 0x00005555555513d <+24>:    add    DWORD PTR [rbp-0x4],0x1
 8 0x000055555555141 <+28>:    cmp    DWORD PTR [rbp-0x4],0x4
 9 0x000055555555145 <+32>:    jle    0x555555555139
10 0x000055555555147 <+34>:    mov    eax,DWORD PTR [rbp-0x8]
11 0x00005555555514a <+37>:    pop    rbp
12 0x00005555555514b <+38>:    ret
13
```

```
 1 0x000055555555125 <+0>:     push   rbp
 2 0x000055555555126 <+1>:     mov    rbp,rsp
 3 0x000055555555129 <+4>:     mov    DWORD PTR [rbp-0x4],0x0
 4 0x000055555555130 <+11>:    mov    DWORD PTR [rbp-0x8],0x0
 5 0x000055555555137 <+18>:    jmp    0x555555555141
 6 0x000055555555139 <+20>:    add    DWORD PTR [rbp-0x8],0x5
 7 0x00005555555513d <+24>:    add    DWORD PTR [rbp-0x4],0x1
 8 0x000055555555141 <+28>:    cmp    DWORD PTR [rbp-0x4],0x4
 9 0x000055555555145 <+32>:    jle    0x555555555139
10 0x000055555555147 <+34>:    mov    eax,DWORD PTR [rbp-0x8]
11 0x00005555555514a <+37>:    pop    rbp
12 0x00005555555514b <+38>:    ret
13
```

QUESTIONS

1. During the decode and execute stages of the fetch-execute cycle the instruction that is being processed is stored in the CIR[Current Instruction Register]. Explain why the instruction could not be processed directly from the MDR.

2. Are there any advantages of having a small number of registers?

ANSWERS

1. To execute the instruction other data may be fetched which would overwrite the MDR and overwrite the instruction recently fetched

- Instruction encoding - The more registers you have, the more bits you need in your instruction to specify each register. If you had only 4 registers, you would only need 2 bits to uniquely identify each register while 256 would need 8 bits.
- Space - Registers are on the physical board so there's a limited amount of realestate to house them