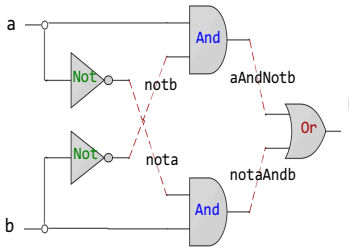
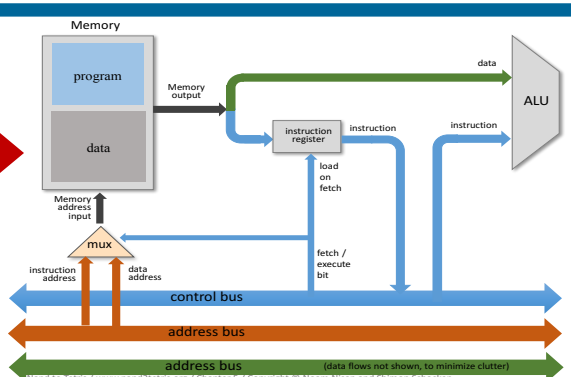
Computer Organization & Assembly Language Programming




```
CHIP Xor {
  IN a, b;
  OUT out;
  PARTS:
  Not(in=a, out=nota);
  Not(in=b, out=notb);
  And(a=nota, b=b, out=w1);
  And(a=a, b=notb, out=w2);
  Or(a=w1, b=w2, out=out);
}
```




```
@R1
D=M
@temp
M=D
```

```
0000000000000001
1111110000010000
0000000000010000
1110001100001000
```

Lecture # 12

Design of Memory Chips

```
#include<stdio.h>
#include<stdlib.h>
int main(){
  printf("Learning is fun with Arif\n");
  exit(0);
}
```

```
global main
SECTION .data
   msg: db "Learning is fun with Arif", 0Ah, 0h
   len_msg: equ $ - msg
SECTION .text
   main:
      mov rax,1
      mov rdi,1
      mov rsi,msg
      mov rdx,len_msg
      syscall
      mov rax,60
      mov rdi,0
      syscall
```

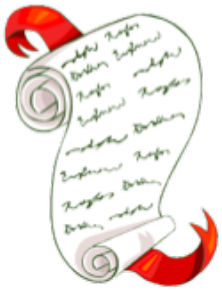
```
0:  b8 01 00 00 00
5:  bf 01 00 00 00
a:  48 be 00 00 00 00 00
11: 00 00 00
14: ba 1b 00 00 00
19: 0f 05
1b: b8 3c 00 00 00
20: bf 00 00 00 00
25: 0f 05
```

Slides of first half of the course are adapted from:
https://www.nand2tetris.org
Download s/w tools required for first half of the course from the following link:
https://drive.google.com/file/d/0B9c0BdDJz6XpZUh3X2dPR1o0MUE/view

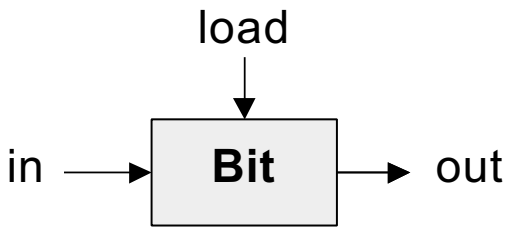
Instructor: Muhammad Arif Butt, Ph.D.

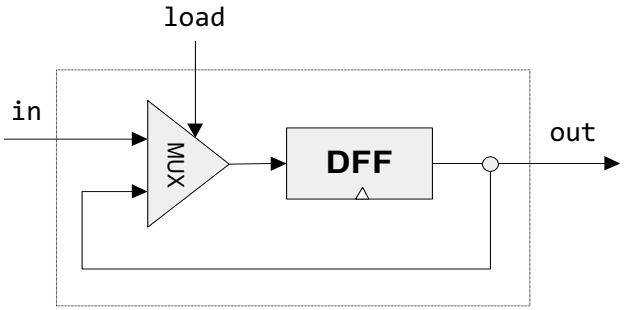
Today's Agenda

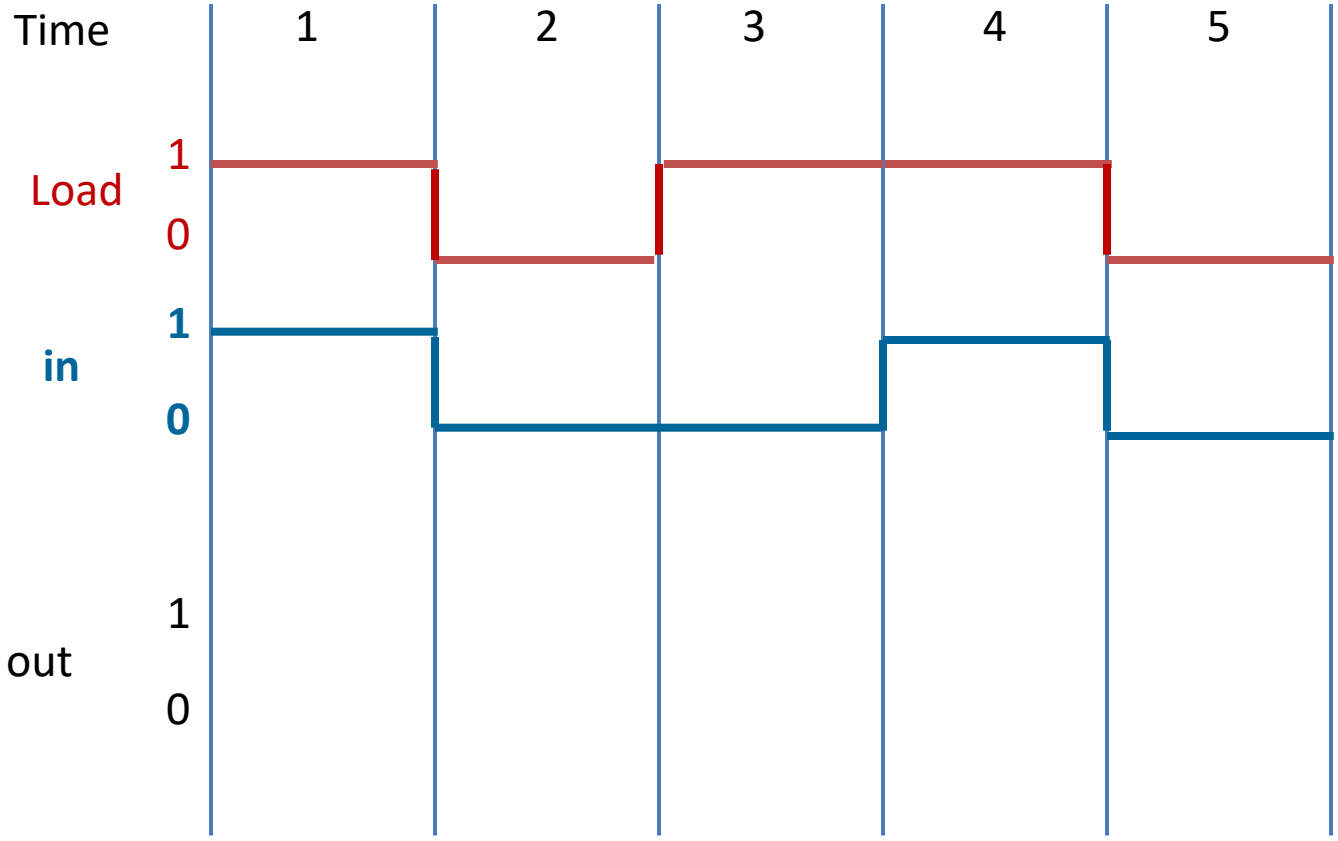
- Class Quiz
- Concept of Memory Hierarchy
- Multi-Byte Read/Write
- Design of Random Access Memory
  - Read/Write Logic of RAM
  - API of a RAM Chip
  - HDL of 8 Words RAM
  - HDL of 64 Words RAM
  - HDL of 512 Words RAM
  - HDL of 4K Words RAM
  - HDL of 16K Words RAM



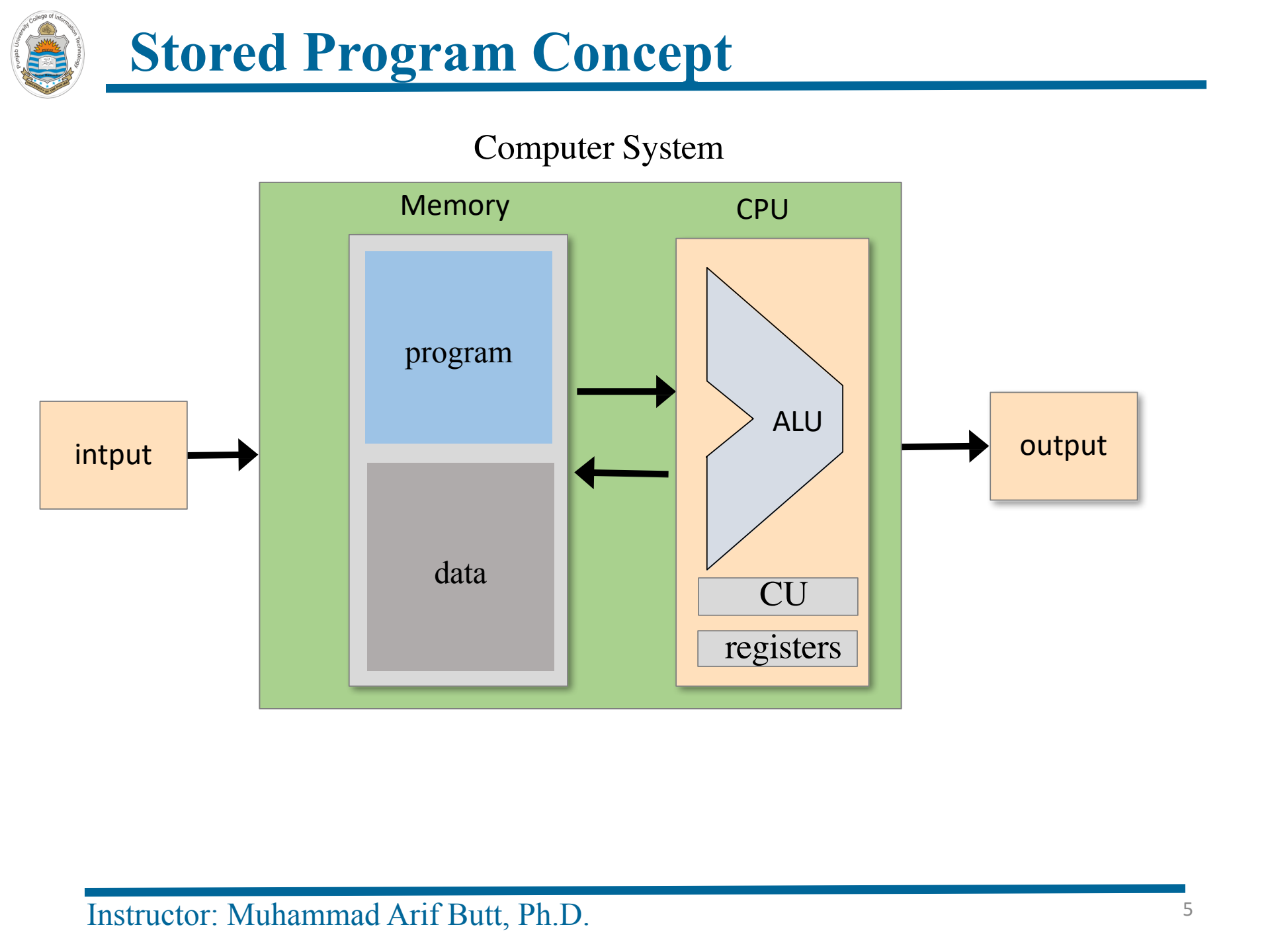
Class Quiz:


load
in
Bit
out


load
in
MUX
DFF
out


Time
1
2
3
4
5
Load
1
0
in
1
0
out
1
0

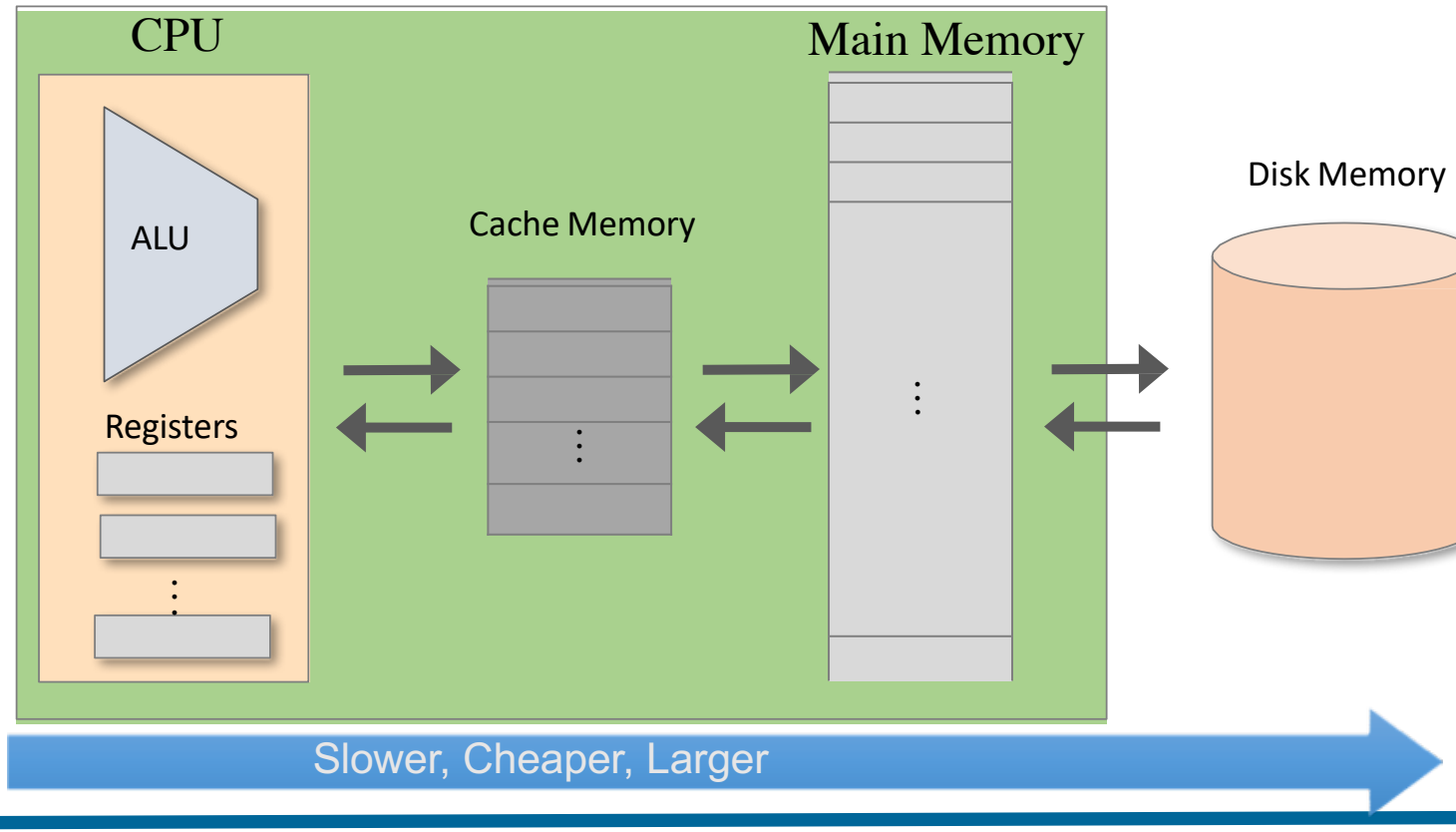
Memory Overview



Stored Program Concept

Memory Hierarchy

- Accessing a memory location is expensive, as we need to supply an address and then read/write the contents of that location of memory. Moreover, moving the memory contents into the CPU and vice versa also takes time
- **Solution:** Memory Hierarchy



Access Times of Memory

Memory Unit	Example Size	Typical Speed
Registers	16, 64-bit registers	1 nanoseconds
Cache memory	4 - 8 Megabytes (L1 and L2)	5-60 nanoseconds
Primary Storage	2 - 32+ Gigabytes	100-150 nanoseconds
Secondary Storage	500 Gigabytes – 4+ Terabytes	3-15 milliseconds

Memory Sizes/Capacity

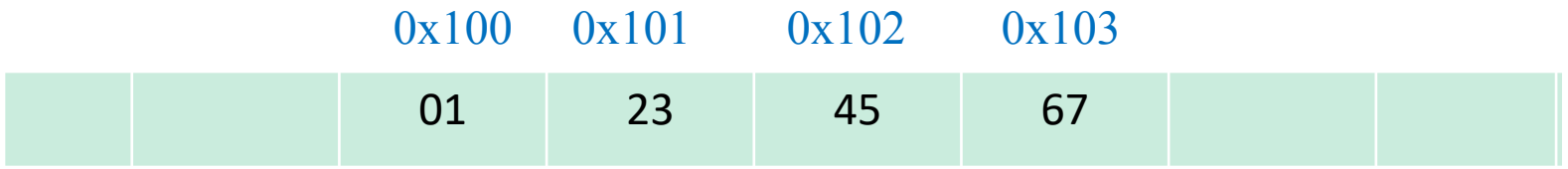
Decimal Term	Abbreviation	Value	Binary Term	Abbreviation	Value	%Larger
kilobyte	KB	10^3	kibibyte	KiB	2^{10}	2%
megabyte	MB	10^6	mebibyte	MiB	2^{20}	5%
Gigabyte	GB	10^9	gibibyte	GiB	2^{30}	7%
terabyte	TB	10^{12}	tebibyte	TiB	2^{40}	10%`
petabyte	PB	10^{15}	pebibyte	PiB	2^{50}	13%
exabyte	EB	10^{18}	exabibyte	EiB	2^{60}	15%
zettabyte	ZB	10^{21}	zebibyte	ZiB	2^{70}	18%
yottabyte	YB	10^{24}	yobibyte	YiB	2^{80}	21%

Random Access Memory (RAM)

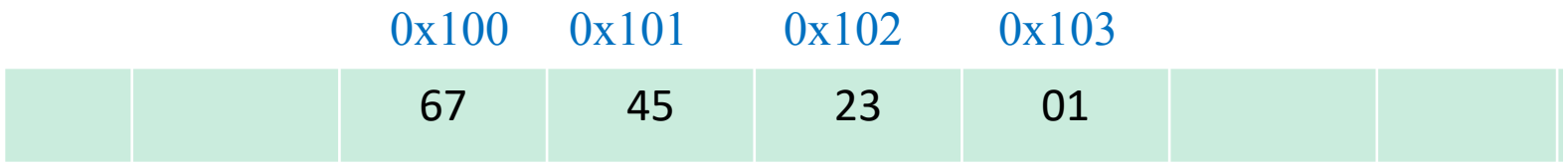
- The computer's main memory is also called the Random Access Memory, because irrespective of the RAM size, every word gets selected instantaneously, at more or less the same time
- It is also known as read/write memory as it allows CPU to read as well as write data and instructions into it
- RAM is a microchip implemented using semiconductors. There are two categories of RAM
  - **Dynamic RAM (DRAM):** It is made up of memory cells where each cell is composed of one capacitor and one transistor. DRAM must be refreshed continually to store information. The refresh operation occurs automatically thousands of times per second. DRAM is slower and less-expensive
  - **Static RAM (SRAM):** It retains the data as long as power is provided to the memory chip. It needs not be refreshed periodically. SRAM uses multiple transistors for each memory cell. It does not use capacitor. SRAM is often used as cache memory due to its high speed. SRAM is more expensive than DRAM

Multi-Byte Ordering

- All 32 bit machines load and store 32 bits of data (word) with each operation. The question is how are the bytes of a multi-byte variable ordered in memory?
- Consider a 32 bit variable having a value of 0x01234567, that needs to be stored at address 0x100
- There are two conventions that the h/w designers can follow:
  - **Big Endian:** Most significant byte is written at the lowest address byte (MSB first). Used by MIPS and Internet



  - **Little Endian:** Least significant byte is written at the lowest address byte (LSB first). Used by x86 and ARM

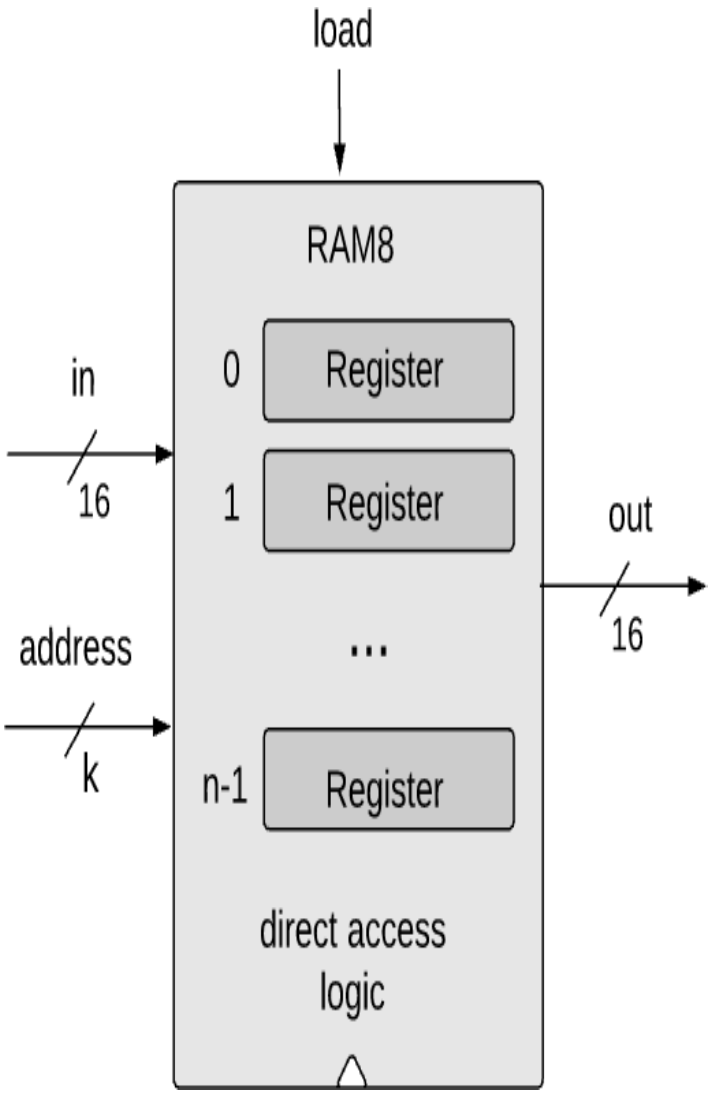


Designing Random Access Memory

Design of RAM

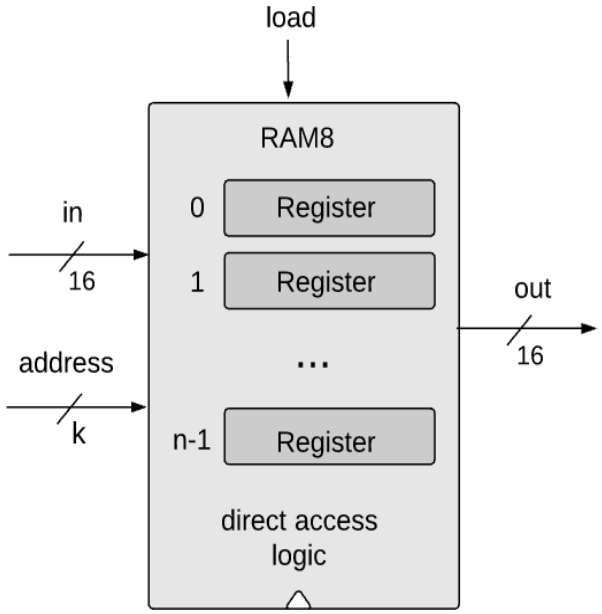
- RAM is an array of ***n w-bit registers***, equipped with direct access circuitry. The number of registers **(n)** and the width of each register **(w)** are called the memory's size and width respectively
- In simple words, you can think of RAM as a sequence of n addressable registers with addresses 0 to n-1
- At any given time only one register in the RAM is selected. It is this register whose value is available on *out* during a read operation. Similarly, it is this register whose contents will be over written during a write operation
- Now to select a register we need its address. Address width varies with the number of registers/words in the RAM, e.g., for RAM8 the address size is 3 bits

$$\mathbf{k} = \text{Address bits} = \log_2 \mathbf{n}$$



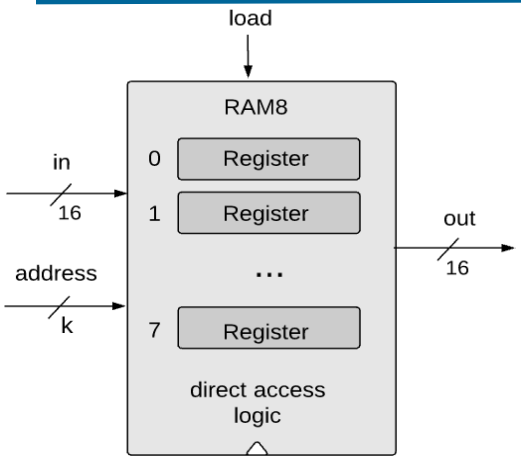
Read/Write Logic of RAM

- At any given point of time: one register in the RAM is selected, all the other registers are irrelevant
- **Read:** To read the contents of register number ***i***,
  - Set address = ***i***
  - Result: The RAM's output pin ***out*** emits the state of the register ***i***. This is a combinational operation, independent of the clock
- **Write:** To write a new data value ***d*** into register number ***i***,
  - Set address = ***i***
  - Set ***in*** = ***d***
  - Set ***load*** = **1**
  - Result: The state of register ***i*** becomes ***d*** and form next clock cycle onwards, ***out*** emits ***d***



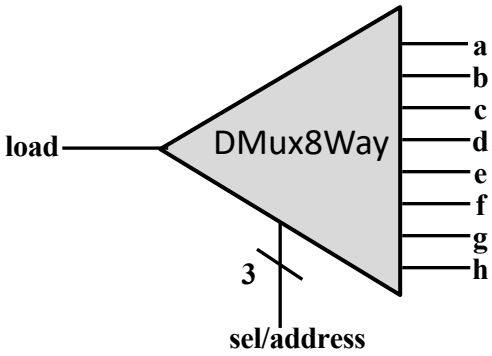
Chip Name	Size (n)	Address bits (k)
RAM8	8	3
RAM64	64	6
RAM512	512	9
RAM4K	4096	12
RAM16K	16384	14

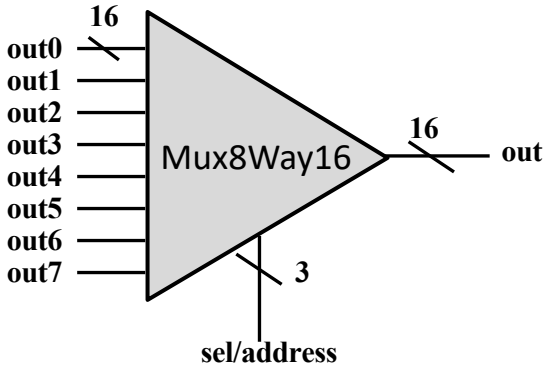
8-Register/words RAM



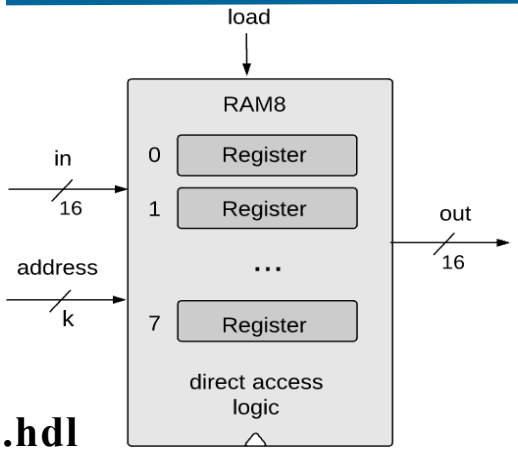
Implementation tips:

- Memory of **8** registers, each 16 bit-wide. ***Out*** holds the value stored at the memory location specified by address. If **load**==1, then the **in** value is loaded into the memory location specified by **address** (the loaded value will be emitted to **out** from the next time step onward)
- Feed the 16 bit ***in*** value to all the registers, simultaneously
- **Write:** Use DMux8Way chip to select one of the eight registers specified by address
- **Read:** Use Mux8Way16 chip to select the 16 bit contents of register specified by address on 16 bit ***out*** output





8-Register/words RAM



Implementation tips:

- Memory of **8** registers, each 16 bit-wide. ***Out*** holds the value stored at the memory location specified by address. If load==1, then the in value is loaded into the memory location specified by address (the loaded value will be emitted to out from the next time step onward)
- Feed the 16 bit ***in*** value to all the registers, simultaneously
- Use DMux8Way chip to select one of the eight registers specified by address
- Use Mux8Way16 chip to select the 16 bit contents of register specified by address on 16 bit ***out*** output

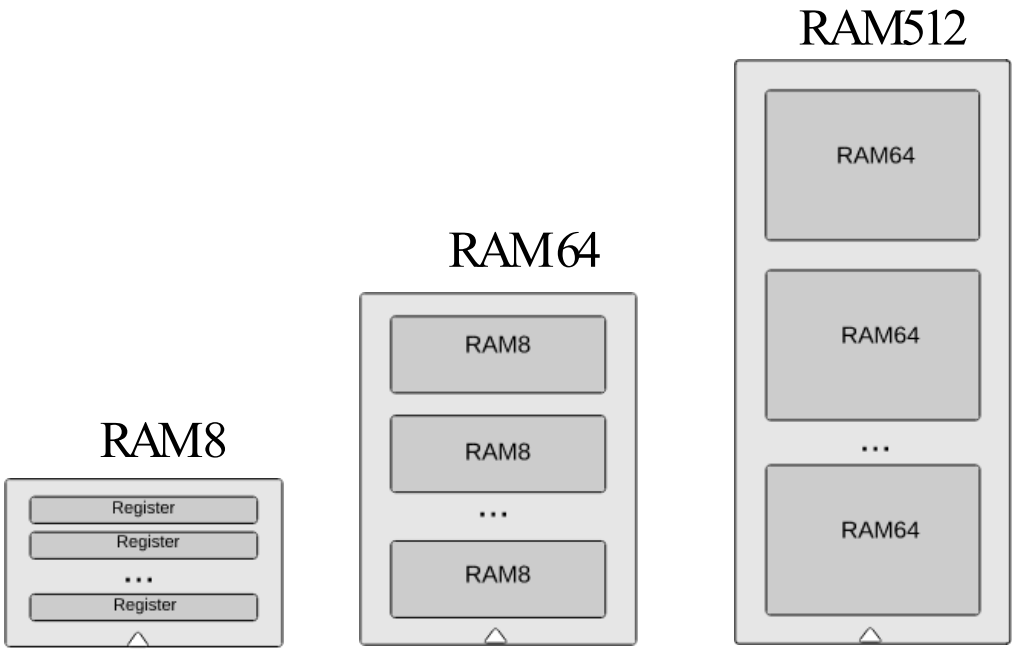
RAM8.hdl

```
CHIP RAM8 {
   IN in[16], load, address[3];
   OUT out[16];
   PARTS:
   DMux8Way(in=load, sel=address, a=load0, b=load1, c=load2, d=load3, e=load4, f=load5, g=load6, h=load7);

   Register(in=in, load=load0, out=out0);
   Register(in=in, load=load1, out=out1);
   Register(in=in, load=load2, out=out2);
   Register(in=in, load=load3, out=out3);
   Register(in=in, load=load4, out=out4);
   Register(in=in, load=load5, out=out5);
   Register(in=in, load=load6, out=out6);
   Register(in=in, load=load7, out=out7);

   Mux8Way16(a=out0, b=out1, c=out2, d=out3, e=out4, f=out5, g=out6, h=out7, sel=address, out=out);
}
```

```
CHIP Register {
     IN in[16], load;
     OUT out[16];
   PARTS:
     Bit(in=in[0], load=load, out=out[0]);
     Bit(in=in[1], load=load, out=out[1]);
     Bit(in=in[2], load=load, out=out[2]);
     Bit(in=in[3], load=load, out=out[3]);
              . . . .
     Bit(in=in[15], load=load, out=out[15]);
}
```

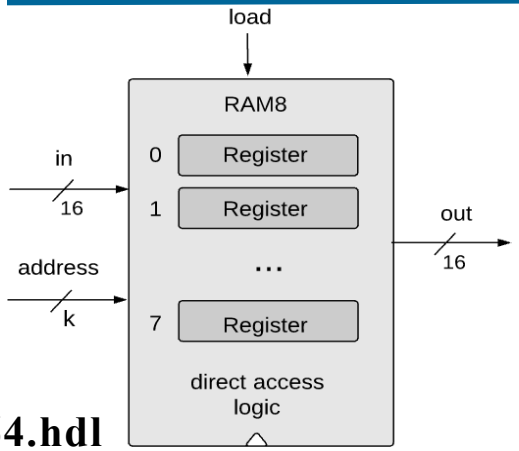
RAM8 Demo

Designing Larger Size RAM Chips



Same technique can be used to implement RAM4K and RAM16K

64-Register/words RAM



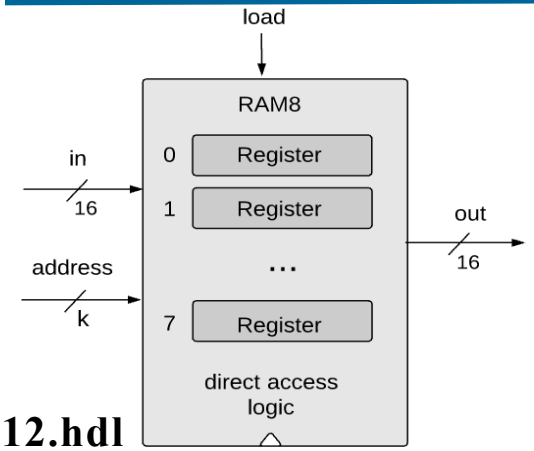
Implementation tips:

- Memory of **64** registers, each 16 bit-wide. ***Out*** holds the value stored at the memory location specified by address. If load==1, then the in value is loaded into the memory location specified by address (the loaded value will be emitted to out from the next time step onward)
- Feed the 16 bit ***in*** value to all the registers, simultaneously
- Use DMux8Way chip to select one of the eight registers specified by address
- Use Mux8Way16 chip to select the 16 bit contents of register specified by address on 16 bit ***out*** output

RAM64.hdl

```
CHIP RAM64 {
    IN in[16], load, address[6];
    OUT out[16];
    PARTS:
    DMux8Way(in=load, sel=address[3..5], a=load0, b=load1, c=load2, d=load3, e=load4, f=load5, g=load6, h=load7);

    RAM8(in=in, load=load0, address=address[0..2], out=out0);
    RAM8(in=in, load=load1, address=address[0..2], out=out1);
    RAM8(in=in, load=load2, address=address[0..2], out=out2);
    RAM8(in=in, load=load3, address=address[0..2], out=out3);
    RAM8(in=in, load=load4, address=address[0..2], out=out4);
    RAM8(in=in, load=load5, address=address[0..2], out=out5);
    RAM8(in=in, load=load6, address=address[0..2], out=out6);
    RAM8(in=in, load=load7, address=address[0..2], out=out7);

    Mux8Way16(a=out0, b=out1, c=out2, d=out3, e=out4, f=out5, g=out6, h=out7, sel=address[3..5], out=out);
}
```

512-Register/words RAM



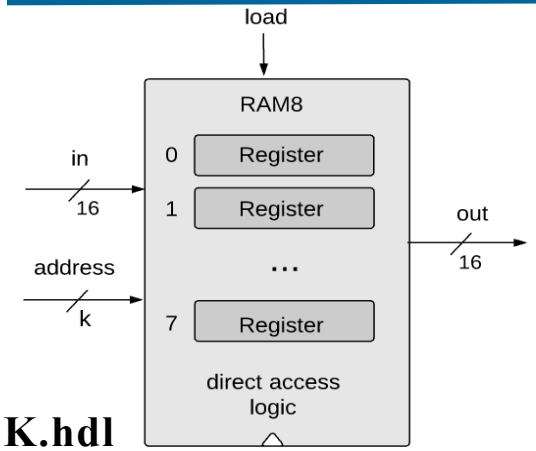
Implementation tips:

- Memory of **512** registers, each 16 bit-wide. ***Out*** holds the value stored at the memory location specified by address. If load==1, then the in value is loaded into the memory location specified by address (the loaded value will be emitted to out from the next time step onward)
- Feed the 16 bit ***in*** value to all the registers, simultaneously
- Use DMux8Way chip to select one of the eight registers specified by address
- Use Mux8Way16 chip to select the 16 bit contents of register specified by address on 16 bit ***out*** output

RAM512.hdl

```
CHIP RAM512 {
   IN in[16], load, address[9];
   OUT out[16];
   PARTS:
   DMux8Way(in=load, sel=address[6..8], a=load0, b=load1, c=load2, d=load3, e=load4, f=load5, g=load6, h=load7);

   RAM64(in=in, load=load0, address=address[0..5], out=out0);
   RAM64(in=in, load=load1, address=address[0..5], out=out1);
   RAM64(in=in, load=load2, address=address[0..5], out=out2);
   RAM64(in=in, load=load3, address=address[0..5], out=out3);
   RAM64(in=in, load=load4, address=address[0..5], out=out4);
   RAM64(in=in, load=load5, address=address[0..5], out=out5);
   RAM64(in=in, load=load6, address=address[0..5], out=out6);
   RAM64(in=in, load=load7, address=address[0..5], out=out7);

   Mux8Way16(a=out0, b=out1, c=out2, d=out3, e=out4, f=out5, g=out6, h=out7, sel=address[6..8], out=out);
}
```

4K-Register/words RAM



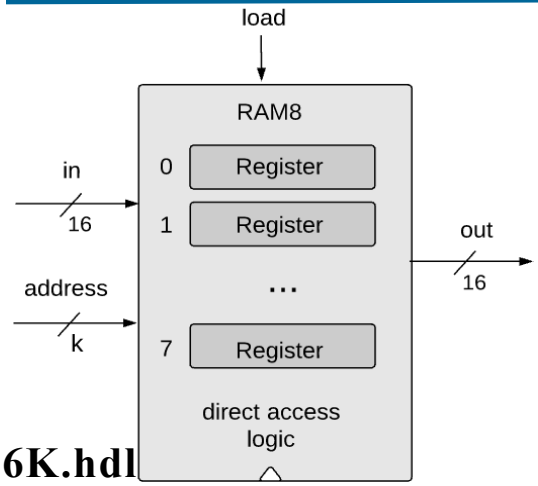
Implementation tips:

- Memory of **4K** registers, each 16 bit-wide. ***Out*** holds the value stored at the memory location specified by address. If load==1, then the in value is loaded into the memory location specified by address (the loaded value will be emitted to out from the next time step onward)
- Feed the 16 bit ***in*** value to all the registers, simultaneously
- Use DMux8Way chip to select one of the eight registers specified by address
- Use Mux8Way16 chip to select the 16 bit contents of register specified by address on 16 bit ***out*** output

RAM4K.hdl

```
CHIP RAM4K {
   IN in[16], load, address[12];
   OUT out[16];
   PARTS:
  DMux8Way(in=load, sel=address[9..11], a=load0, b=load1, c=load2, d=load3, e=load4, f=load5, g=load6, h=load7);

   RAM512(in=in, load=load0, address=address[0..8], out=out0);
   RAM512(in=in, load=load1, address=address[0..8], out=out1);
   RAM512(in=in, load=load2, address=address[0..8], out=out2);
   RAM512(in=in, load=load3, address=address[0..8], out=out3);
   RAM512(in=in, load=load4, address=address[0..8], out=out4);
   RAM512(in=in, load=load5, address=address[0..8], out=out5);
   RAM512(in=in, load=load6, address=address[0..8], out=out6);
   RAM512(in=in, load=load7, address=address[0..8], out=out7);

  Mux8Way16(a=out0, b=out1, c=out2, d=out3, e=out4, f=out5, g=out6, h=out7, sel=address[9..11], out=out);
}
```

16K-Register/words RAM



Implementation tips:

- Memory of **16K** registers, each 16 bit-wide. ***Out*** holds the value stored at the memory location specified by address. If load==1, then the in value is loaded into the memory location specified by address (the loaded value will be emitted to out from the next time step onward)
- Feed the 16 bit ***in*** value to all the registers, simultaneously
- Use DMux8Way chip to select one of the eight registers specified by address
- Use Mux8Way16 chip to select the 16 bit contents of register specified by address on 16 bit ***out*** output

RAM16K.hdl

```
CHIP RAM16K {
   IN in[16], load, address[14];
   OUT out[16];
   PARTS:
   DMux4Way(in=load, sel=address[12..13], a=load0, b=load1, c=load2, d=load3);

   RAM4K(in=in, load=load0, address=address[0..11], out=out0);
   RAM4K(in=in, load=load1, address=address[0..11], out=out1);
   RAM4K(in=in, load=load2, address=address[0..11], out=out2);
   RAM4K(in=in, load=load3, address=address[0..11], out=out3);

   Mux4Way16(a=out0, b=out1, c=out2, d=out3, sel=address[12..13], out=out);
}
```

Things To Do

- Perform testing of the chips designed in today's session on the h/w simulator. You can download the .hdl, .tst and .cmp files of above chips from the course bitbucket repository:

  https://bitbucket.org/arifpucit/coal-repo/

- Interested students should also try to implement RAM chips having a word size other than 16 bits and try running them on the simulator

Coming to office hours does NOT mean you are academically week!