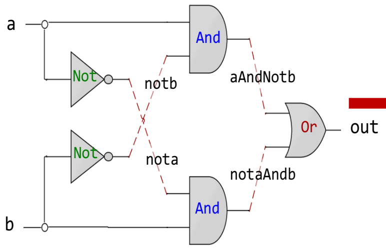
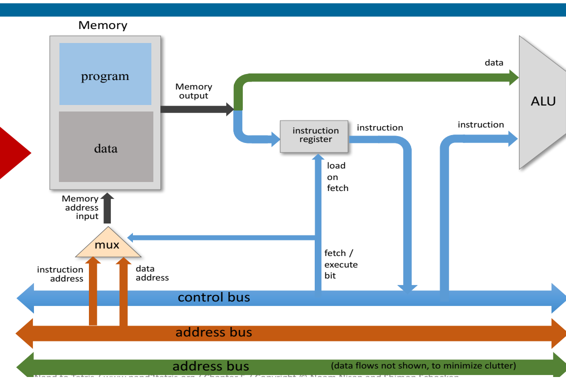
Computer Organization & Assembly Language Programming




```
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 # 13

Design of Counters

```
#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
```

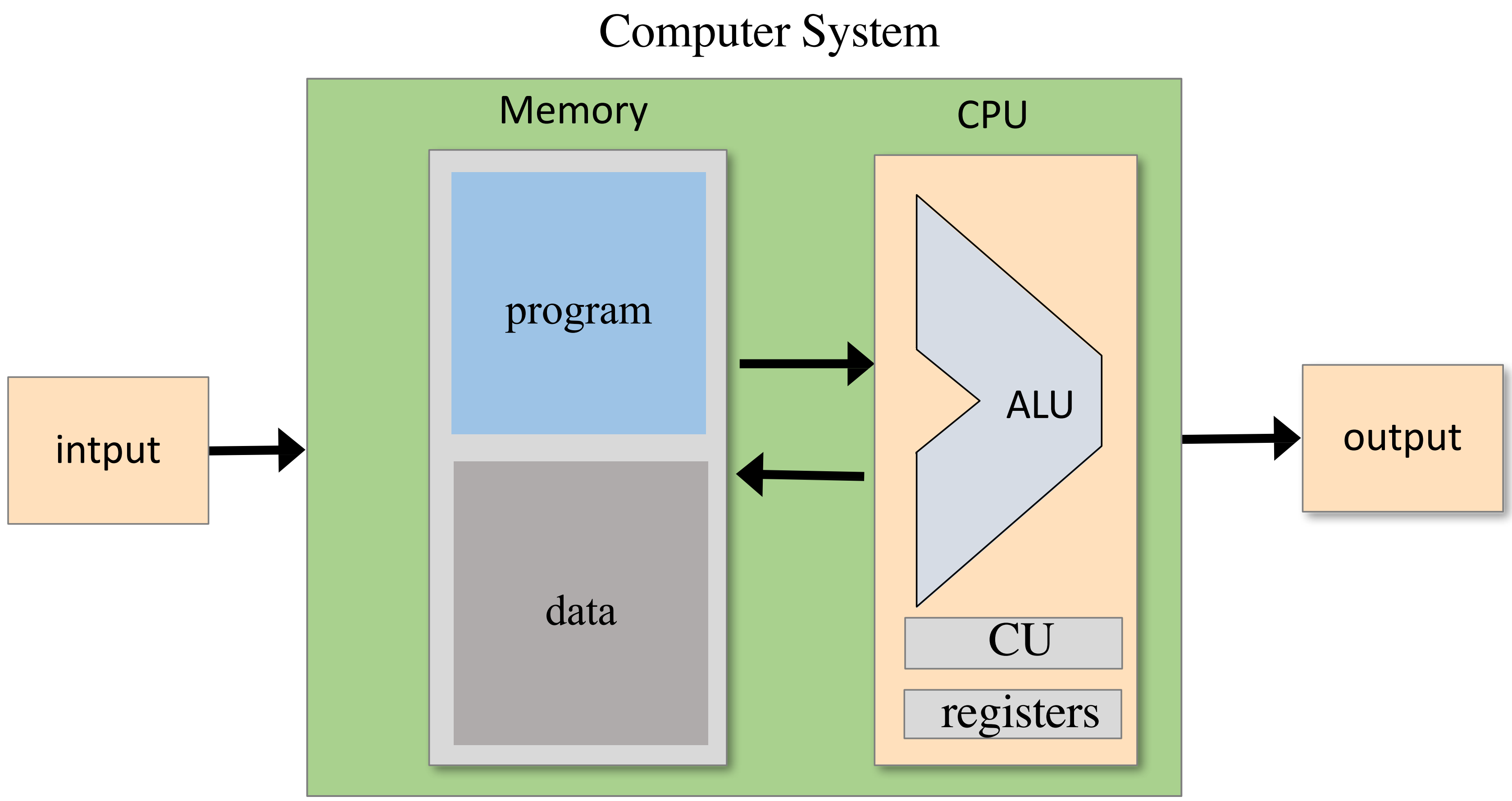
Slides of first half of the course are adapted from:
https://www.nand2tetris.org
Download s/w tools required for first half of the course from the following link:
https://drive.google.com/file/d/0B9c0BdDJz6XpZUh3X2dPR1o0MUE/view

Instructor: Muhammad Arif Butt, Ph.D.

Today's Agenda

- Overview of Hack Computer Components
- Overview of Counters
- Why do we need Counter for our Hack Computer
- Concept of Program Counter
- Counter Simulation
- Design and Implementation of PC for Hack Computer
- Demo on H/W Simulator

Hack Computer System



Counters

Overview of Counters

- A counter is a special type of register that goes through a pre-determined sequence of states upon the application of input pulses
- **Counters are used for:**
  - Counting the number of occurrences of an event
  - Keeping time or calculating amount of time between events
  - Baud rate generation
- **A w-bit counter consists of two main elements:**
  - A w-bit register to store a w-bit value
  - A combinational logic to
    - Compute the next value (according to a specific counting function)
    - Load a new value of user/programmer choice
    - Reset the counter to a default value
- **Examples:**
  - Simple Up/Down Binary Counters
  - BCD Counter(s)
  - Gray Code Counter
  - Ring Counter
  - Johnson Counter

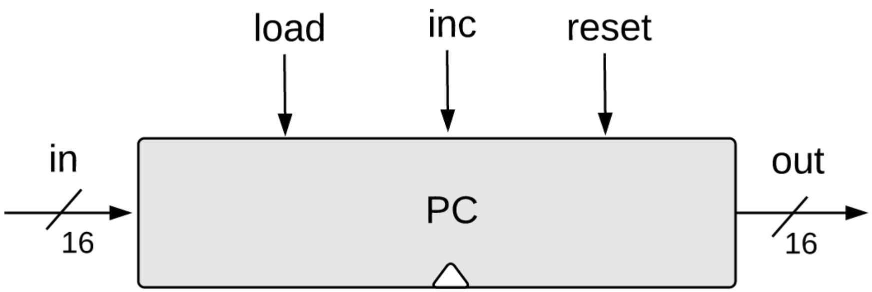
Why we Need Counter Chip for Hack CPU

- Consider a counter chip designed to contain the address of the instruction that the computer should fetch and execute next
- In most cases, the counter has to simply increment itself by 1 in each clock cycle, thus causing the computer to fetch the next instruction in the program
- In other cases, we may want the program to ***jump to an instruction at memory address n***, so the programmer want to set the counter to a value of ***n***, rather than its default counting behavior with ***n+1***, ***n+2***, and so forth
- Finally, the program's execution can be restarted anytime by resetting the counter to 0, assuming that the address of the program's first instruction
- In short, we need a loadable and resettable counter

Program Counter Register

- Every computer has a special register called the Program Counter, normally called the PC, which keeps track of the instruction to be fetched and executed next
- The PC is designed to support three possible control operations:
  - **Reset:** Fetch the first instruction — **PC = 0**
  - **Next:** Fetch the next instruction — **PC++**
  - **Goto:** Fetch instruction at address **n** — **PC = n**

Counter Abstraction



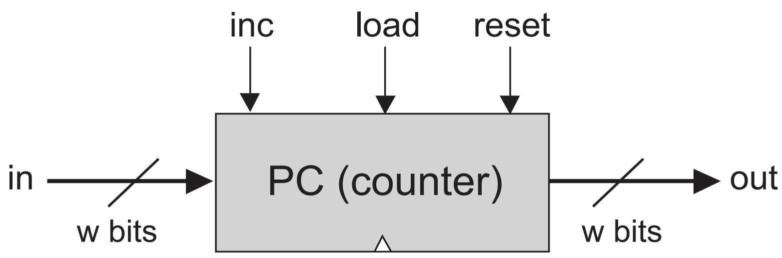

```
if reset[t] = 1 then
      out[t+1] = 0
else if load[t] = 1 then
      out[t+1] = in[t]
else if inc[t] = 1 then
      out[t+1] = out[t] + 1
else out[t+1] = out[t] //do nothing
```

PC = 0

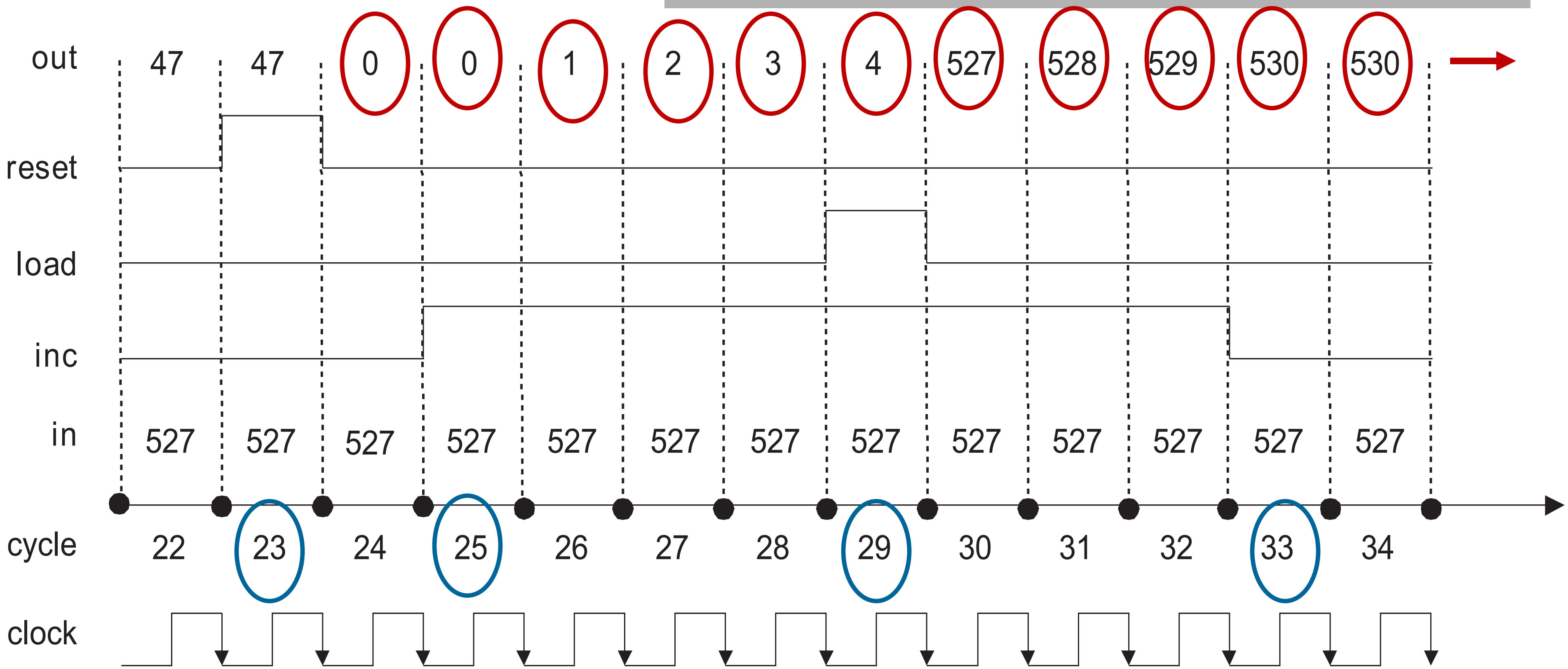
PC = in

PC++

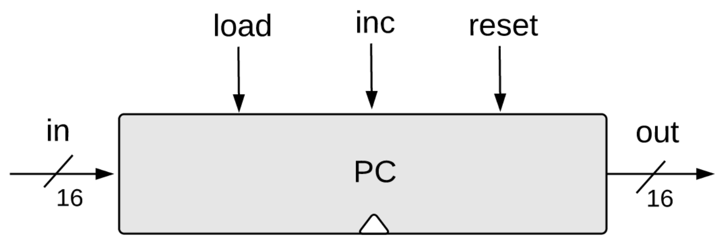
Counter Simulation




```
Chip name: PC  // 16-bit counter
Inputs:    in[16], inc, load, reset
Outputs:   out[16]
Function:  If reset(t-1) then out(t)=0
              else if load(t-1) then out(t)=in(t-1)
                   else if inc(t-1) then out(t)=out(t-1)+1
                        else out(t)=out(t-1)
Comment:   "=" is 16-bit assignment.
           "+" is 16-bit arithmetic addition.
```



16 Bit Program Counter Implementation



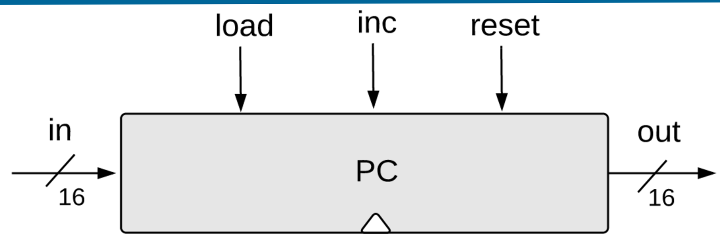

```
CHIP Register {
     IN in[16], load;
     OUT out[16];
   PARTS:
     Bit(in=in[0], load=load, out=out[0]);
     Bit(in=in[1], load=load, out=out[1]);
     Bit(in=in[2], load=load, out=out[2]);
     Bit(in=in[3], load=load, out=out[3]);
           . . . .
     Bit(in=in[15], load=load, out=out[15]);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

```
CHIP Bit {
     IN in, load;
     OUT out;
   PARTS:
     Mux(a=sendBack, b=in, sel=load, out=MuxOut);
     DFF(in=MuxOut, out=sendBack, out=out);
}
```

```
CHIP Inc16 {
  IN in[16];
  OUT out[16];
  PARTS:
   Add16(a=in, b[0]=true, out=out);
}
```

```
CHIP Add16 {
  IN a[16], b[16];
  OUT out[16];
  PARTS:
   HalfAdder(a=a[0], b=b[0], sum=out[0], carry=carry0);
   FullAdder(a=a[1], b=b[1], c=carry0, sum=out[1], carry=carry1);
   FullAdder(a=a[2], b=b[2], c=carry1, sum=out[2], carry=carry2);
   FullAdder(a=a[3], b=b[3], c=carry2, sum=out[3], carry=carry3);
   .........
   FullAdder(a=a[14], b=b[14], c=carry13, sum=out[14], carry=carry14);
   FullAdder(a=a[15], b=b[15], c=carry14, sum=out[15], carry=carry15);
}
```

16 Bit Program Counter Implementation



PC.hdl

```
CHIP PC {
    IN in[16], load, inc, reset;
    OUT out[16];

    PARTS:
        Inc16(in=regContent, out=incremented);
//if (inc == 1)
        Mux16(a=regContent, b=incremented, sel=inc, out=value1);
//else if (load == 1)
        Mux16(a=value1, b=in, sel=load, out=value2);
//else if (reset == 1)
        Mux16(a=value2, b=false, sel=reset, out=value3);
//else
        Register(in=value3, load=true, out=regContent, out=out);
}
```

```
CHIP Mux16 {
  IN a[16], b[16], sel;
  OUT out[16];
  PARTS:
   Mux(a=a[0], b=b[0], sel=sel, out=out[0]);
   Mux(a=a[1], b=b[1], sel=sel, out=out[1]);
             . . . .
   Mux(a=a[1], b=b[15], sel=sel, out=out[15]);
}
```

Program Counter Demo

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 design, implement and simulate binary down counter, cascaded BCD counter, Gray Counter, Ring counter, and Johnson counter

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