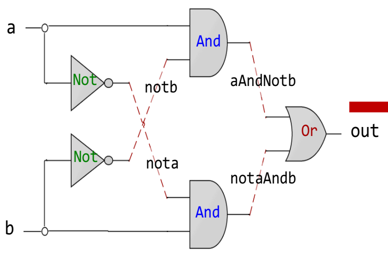
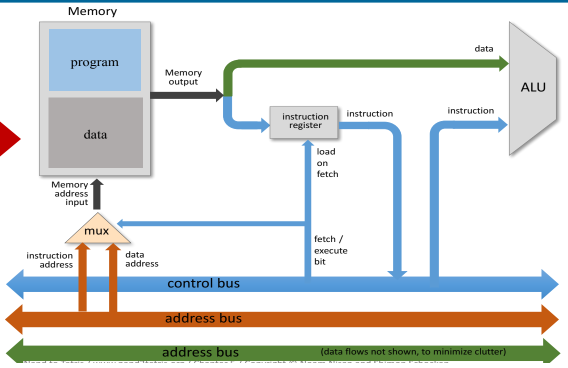
Computer Organization & Assembly Language Programming



```
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 # 05

HDL for Combinational Circuits - IV

```
#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
```

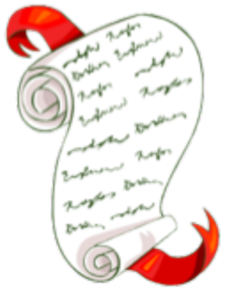
```
0:  b8 01 00 00 00
5:  bf 01 00 00 00
a:  48 be 00 00 00 00 00
11: 00 00 00
14: ba 1b 00 00 00
19: 0f 05
1b: b8 3c 00 00 00
20: bf 00 00 00 00
25: 0f 05
```

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

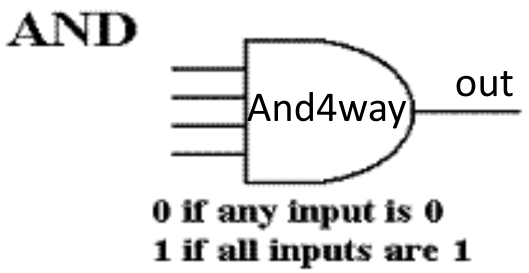
- Multi-bit Gates (more than 2 input gates)
  - And4way
  - Or4way
  - Mux4way
- Array of Bits/Busses (2 input gate with each input of 16 bits)
  - And16
  - Or16
  - Not16
  - Mux16
- Combining Multi-bit gates with Array of Bits
  - And4way16
  - Mux4way 16
  - Mux8way16
- What are built-in Chips?
  - Explicit use of built-in Chips
  - Implicit use of built-in Chips

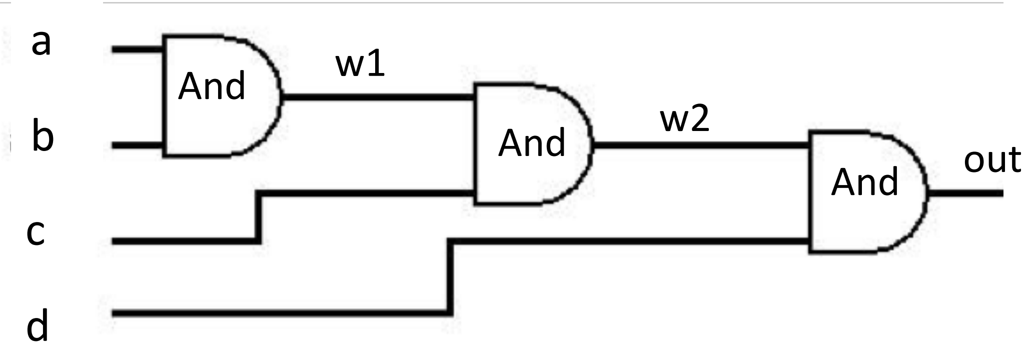


Multi-Bit Gates

And4way: Gate that ANDs 4 bits

- Suppose we want to design an AND gate chip with four inputs
- Although we can design it using the built-in NAND gate, but why to reinvent the wheel.
- Let us design it using the already designed AND gate chips with two inputs



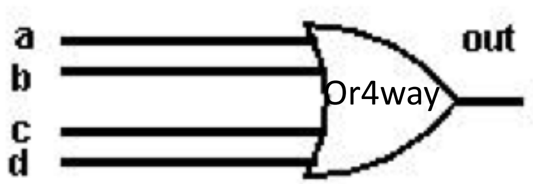


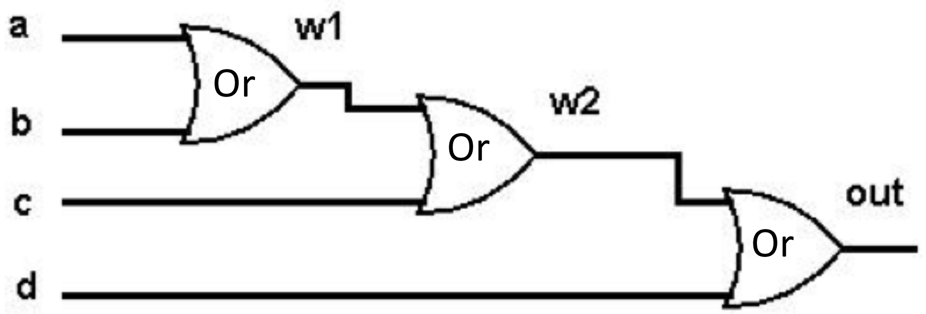
And4way.hdl

```
CHIP And4way{
   IN a,b,c,d;
   OUT out;
  PARTS:
    And(a=a, b=b, out=w1);
    And(a=w1, b=c, out=w2);
    And(a=w2, b=d, out=out);
}
```

Or4way: Gate that ORs 4 bits

- In a similar fashion, we can design an OR gate chip with four inputs using the already designed OR gate chips with two inputs

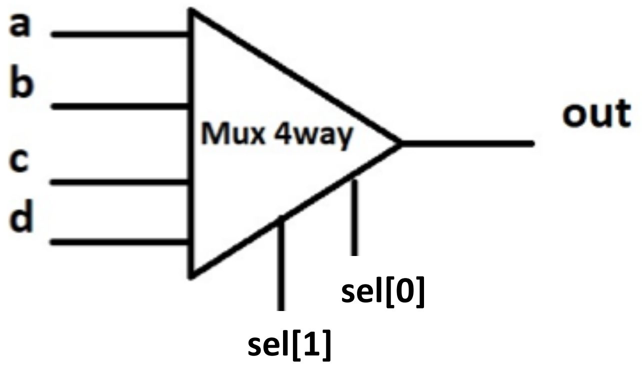


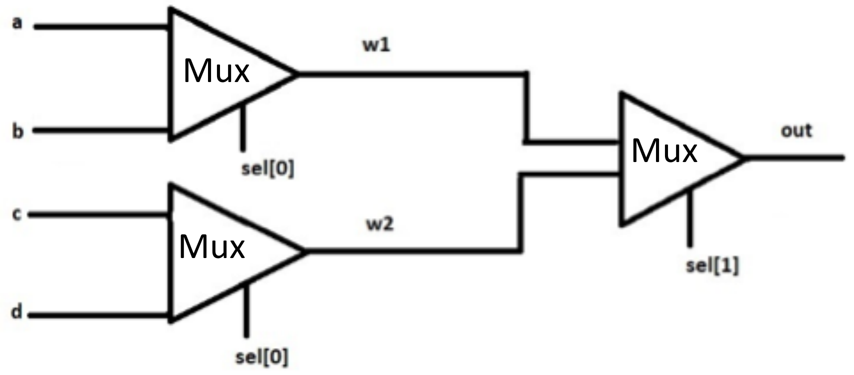


Or4way.hdl

```
CHIP Or4way{
   IN a,b,c,d;
   OUT out;
  PARTS:
  Or(a=a, b=b, out=w1);
  Or(a=w1, b=c, out=w2);
  Or(a=w2, b=d, out=out);
}
```

Mux4way: (using 2x1 Mux)





Sel[1]	Sel[0]	out
0	0	a
0	1	b
1	0	c
1	1	d

Mux4way.hdl

```
CHIP Mux4way{
   IN a,b,c,d, sel[2];
   OUT out;

  PARTS:
  Mux(a=a, b=b, sel=sel[0], out=w1);
  Mux(a=c, b=d, sel=sel[0], out=w2);
  Mux(a=w1, b=w2, sel=sel[1], out=out);
}
```

Multi Bit Gates Demo

Demo

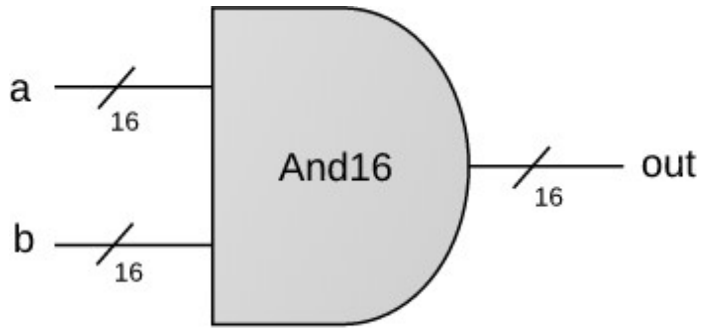
Hardware Simulator

```
05/And4way.hdl
05/Or4way.hdl
05/Mux4way.hdl
```

Array of Bits / Busses

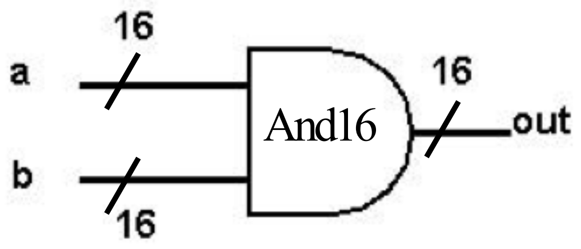
Array Of Bits

- While designing hardware, a lot of times we need to manipulate a bunch of bits together and it is conceptually convenient to think about the bunch of bits that are manipulated together as one entity called busses
- Example: A chip that performs bit-wise AND of two 16 bit numbers. So the chip has two inputs each of 16 bits. The chip also has an output of 16 bits. So in reality, the chip has 32 wires feeding into it, and 16 wires going out of it, but still it's convenient to think about it as two numbers feeding in and one number feeding out

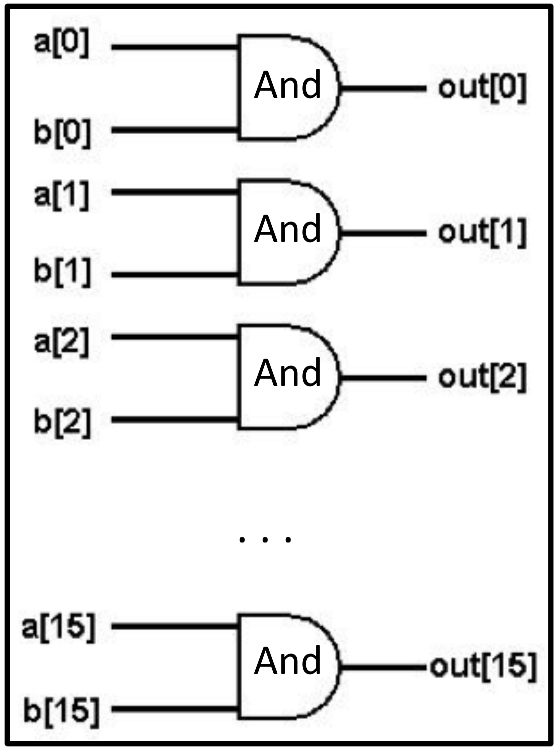


```
CHIP And16 {
    IN a[16], b[16];
    OUT out[16];

    PARTS:
    // Put your code here:
}
```

And16: Gate that And two 16-bit Numbers



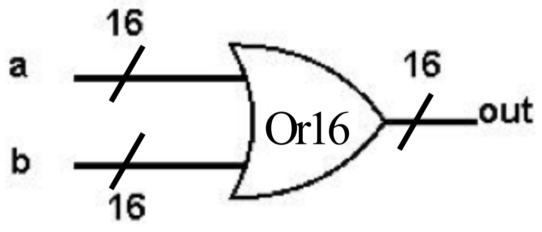
```
a   = 1 0 1 0 1 0 1 1 0 1 0 1 1 1 0 0
b   = 0 0 1 0 1 1 0 1 0 0 1 0 1 0 1 0
out = 0 0 1 0 1 0 0 1 0 0 0 0 1 0 0 0
```



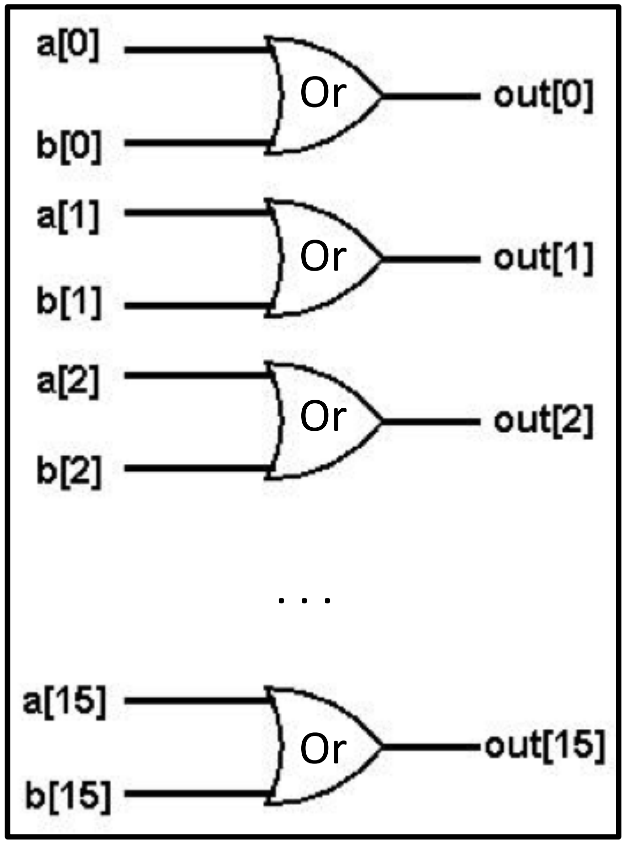
And16.hdl

```
CHIP And16{
  IN a[16], b[16];
  OUT out[16];

  PARTS:
    And(a=a[0], b = b[0], out=out[0]);
    And(a=a[1], b = b[2], out=out[1]);
    And(a=a[2], b = b[3], out=out[2]);
   . . . .
    And(a=a[15], b = b[15], out=out[15]);
}
```

Or16: Gate that Or two 16-bit Numbers



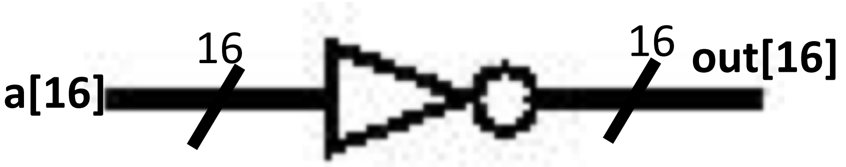
```
a   = 1 0 1 0 1 0 1 1 0 1 0 1 1 1 0 0
b   = 0 0 1 0 1 1 0 1 0 0 1 0 1 0 1 0
out = 1 0 1 0 1 1 1 1 0 1 1 1 1 1 1 0
```



Or16.hdl

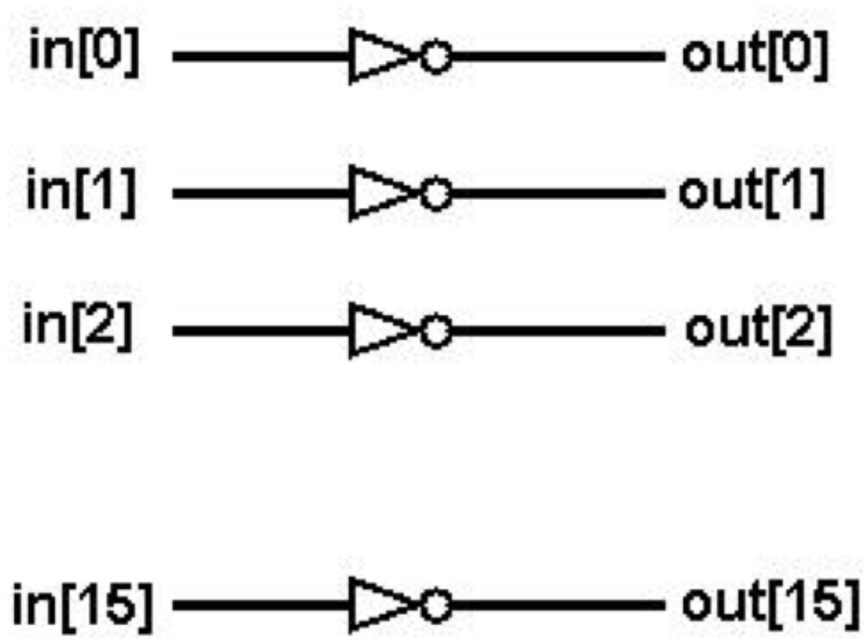
```
CHIP Or16{
   IN a[16], b[16];
   OUT out[16];

  PARTS:
     Or(a=a[0], b = b[0], out=out[0]);
     Or(a=a[1], b = b[2], out=out[1]);
     Or(a=a[2], b = b[3], out=out[2]);
   . . . .
     Or(a=a[15], b = b[15], out=out[15]);
}
```

Not16: Gate that Perform Not of 16-bit Number



a = 0 0 1 0 1 1 0 1 0 0 1 0 1 0 1 0

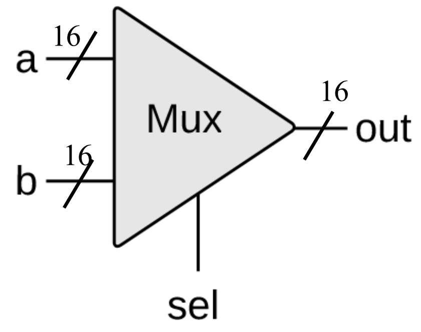
out = 1 1 0 1 0 0 1 0 1 1 0 1 0 1 0 1

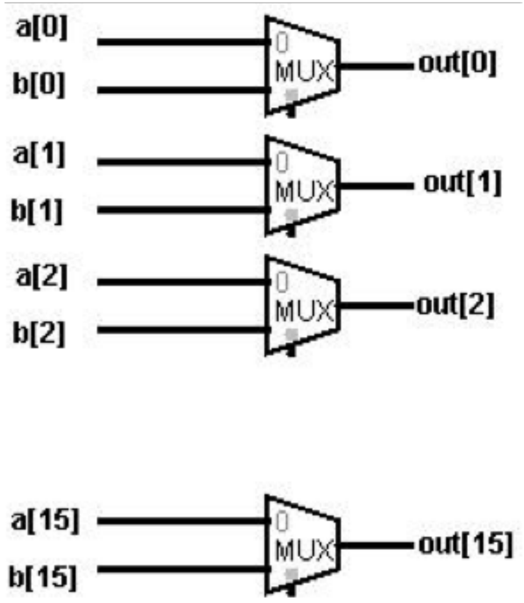


Not16.hdl

```
CHIP Not16{
   IN a[16];
   OUT out[16];

  PARTS:
     Not(in=a[0], out=out[0]);
     Not(in=a[1], out=out[1]);
     Not(in=a[2], out=out[2]);
   . . . .
     Not(in=a[15], out=out[15]);
}
```

Mux16





Mux16.hdl

```
/*
 * 16-bit multiplexor:
 * for i = 0..15 out[i] = a[i] if sel == 0
 *                        b[i] if sel == 1
 */
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[2], b=b[2], sel=sel, out=out[2]);
         …
    Mux(a=a[15], b=b[15], sel=sel, out=out[15]);
}
```

Concept of Sub-Buses

- Buses are indexed right to left: if foo is a 16-bit bus, Then foo[0] is the right-most bit (LSb), and foo[15] is the left-most bit (MSb)
- Buses can be composed from sub-buses, i.e., we can compose a 16 bit bus from two 8 bit buses
- Example: In the code snippet below, we have two 8 bit buses namely lsb and msb. In the first 16 bit value to And16 chip we plug in the 8 bits of lsb and the 8 bits of msb. Note the dotdot notation using which we can mention the sub range of a bus
- Lastly if you want to initialize an entire bus with zeros or ones, you can do so in one command by assigning "true" or "false" to the bus

```
...
IN lsb[8], msb[8], …
...
And16(a[0..7]=lsb, a[8..15]=msb, b=…, out=…);
```

Gates with Buses: Demo

Demo

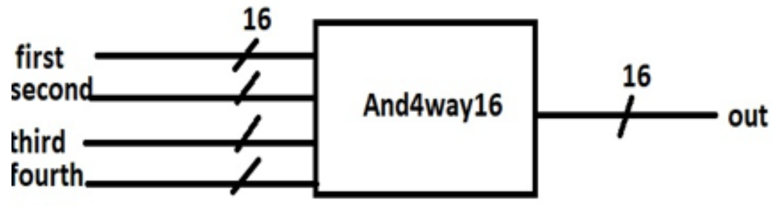
Hardware Simulator

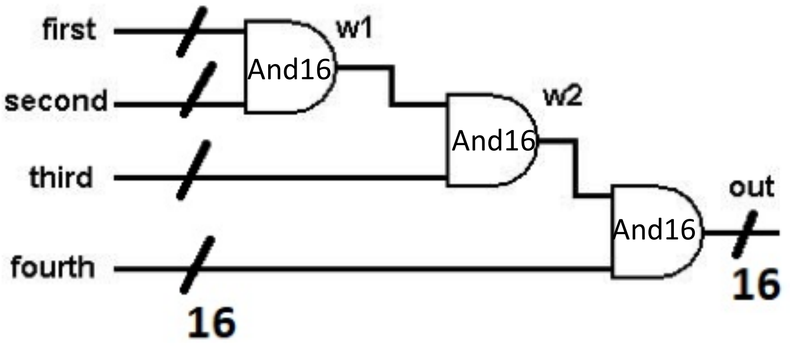
```
05/And16.hdl
05/Or16.hdl
05/Mux16.hdl
```

Combining
Multi-bit Gates and Array of Bits

And4way16

- Suppose now we need to build a chip that bit-wise And four 16 bit numbers. We can design this chip using three And16 chips each capable of Anding two 16 bit numbers
  - ➢ The first And16 chip will bit-wise And two 16 bit numbers and place the result in a variable, w1
  - ➢ The second And16 chip will bit-wise And the third 16 bit number with w1 and place the result in w2
  - ➢ The third Add16 chip will bit-wise And the fourth 16 bit number with w2 and generate the final output

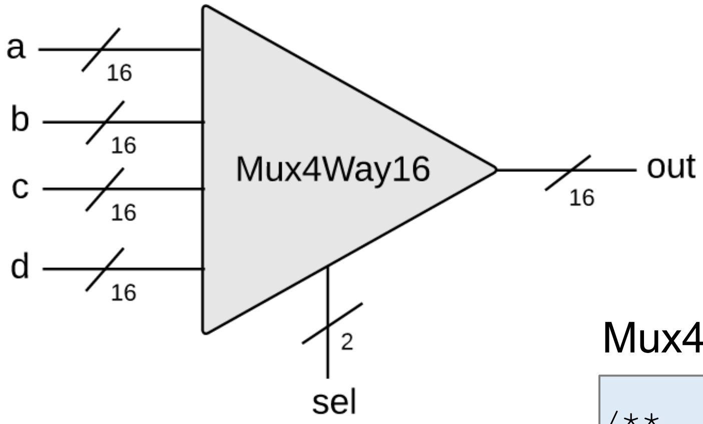


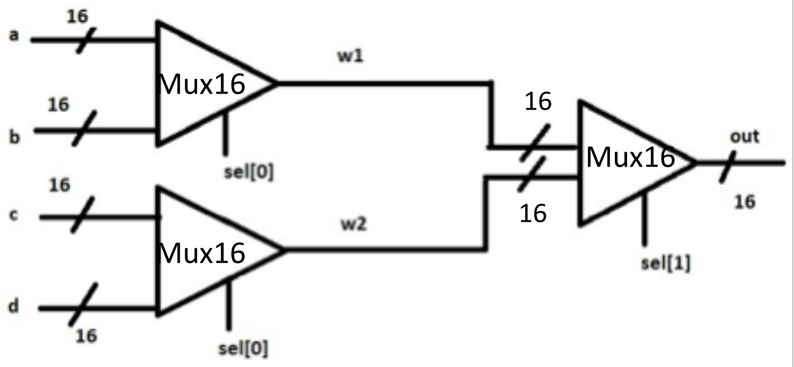


And4way16.hdl

```
CHIP And4way16 {
  IN first[16], second[16],
third[16], fourth[16];
   OUT out[16];


  PARTS:
  And16(a=first, b = second, out=w1);
  And16(a=w1, b = third, out=w2);
  And16(a=w2, b = fourth, out=out);
}
```

Mux4way16



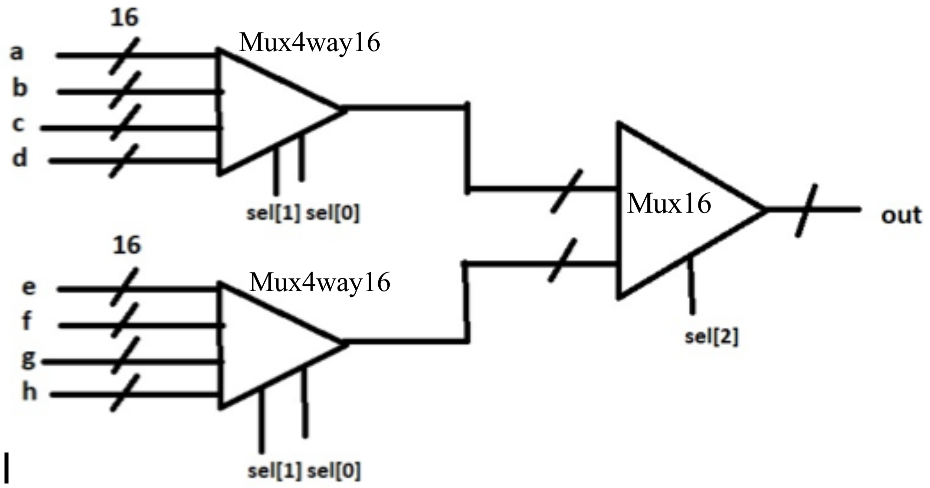


Sel[1]	Sel[0]	out
0	0	a
0	1	b
1	0	c
1	1	d

Mux4Way16.hdl

```
/**
 * 4-way 16-bit multiplexor:
 * out = a if sel == 00
 *       b if sel == 01
 *       c if sel == 10
 *       d if sel == 11
 */
CHIP Mux4Way16 {
    IN a[16], b[16], c[16], d[16], sel[2];
    OUT out[16];

    PARTS:

    Mux16(a=a, b=b, sel=sel[0], out=w1);
    Mux16(a=c, b=d, sel=sel[0], out=w2);
    Mux16(a=w1, b=w2, sel=sel[1], out=out);
}
```

Mux8way16



```
/*   8-way 16-bit mux
 *    out = a if sel == 000
 *    out = b if sel == 001
        ............
 *    out = b if sel == 001
*/
```

Mux8Way16.hdl

```
CHIP Mux8Way16 {
    IN a[16], b[16], c[16], d[16], e[16], f[16], g[16], h[16],
       sel[3];
    OUT out[16];
    PARTS:
    Mux4Way16(a=a, b=b, c=c, d=d, sel=sel[0..1], out=Mux4abcd);
    Mux4Way16(a=e, b=f, c=g, d=h, sel=sel[0..1], out=Mux4efgh);
    Mux16(a=Mux4abcd, b=Mux4efgh, sel=sel[2], out=out);
}
```

Multi-Bit Gates with Buses: Demo

```
05/And4way16.hdl
05/Mux4way16.hdl
05/Mux8way16.hdl
```

What is a Built-in Chip?

Built-in Chips

General

- A built-in chip has an HDL interface and a Java implementation (e.g. here: Mux16.class)
- The name of the Java class is specified following the BUILTIN keyword
- Built-In implementations of all the chips that are supplied in the tools/buitInChips directory

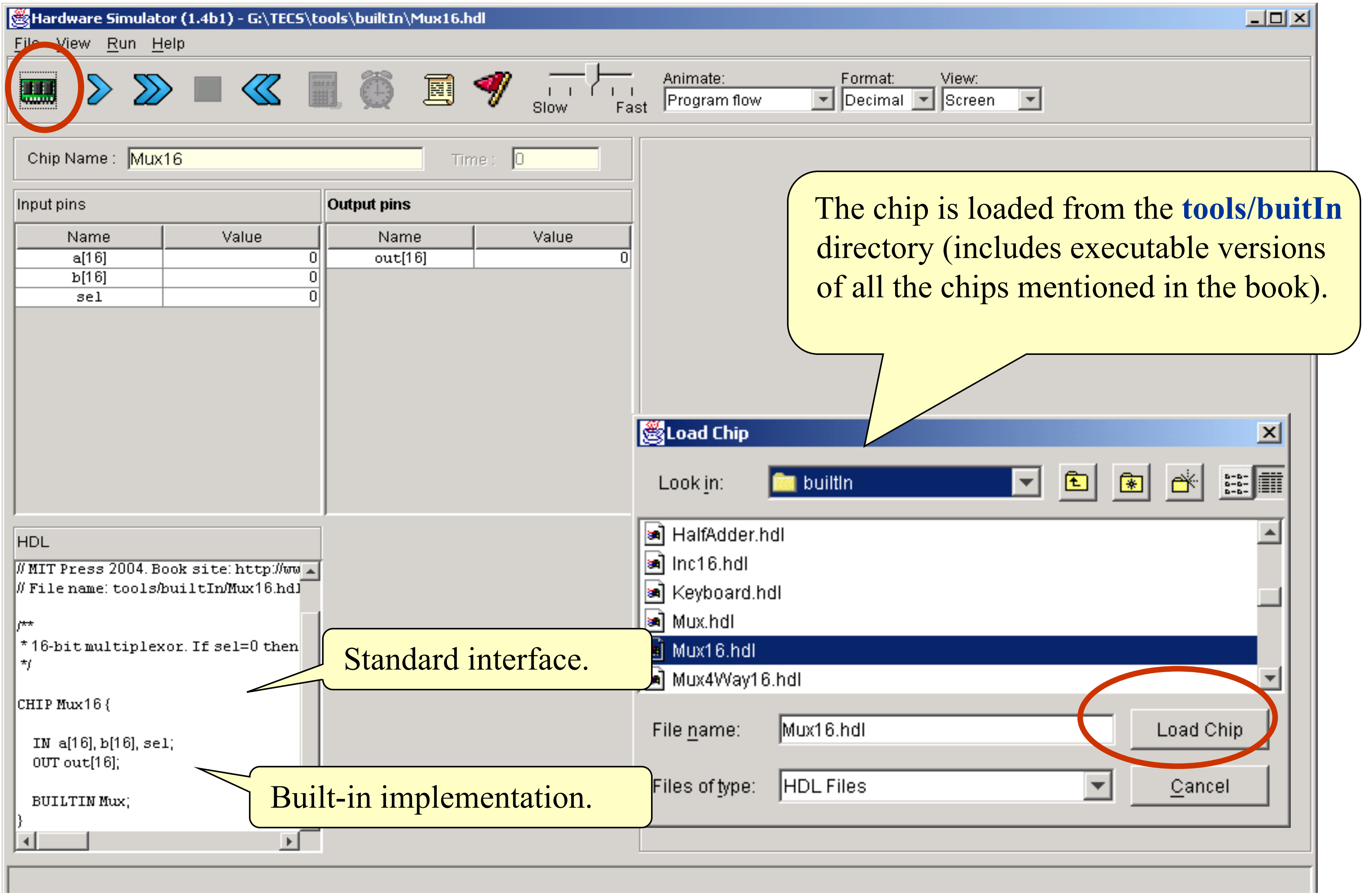
```
// Mux16 gate (example)
CHIP Mux16 {
   IN a[16],b[16],sel;
   OUT out[16];
   BUILTIN Mux16;
}
```

Built-in chips are used to:

- Implement basic primitive gates to build other gates (Nand and DFF)
- Provide the functionality of chips that the user did not implement for some reason
- Improve simulation speed and save memory (when used as parts in complex chips)
- Implement chips that have peripheral side effects (like I/O devices)
- Implement chips that feature a GUI (for debugging)
- Built-in chips can be used either explicitly, or implicitly

Note: The supplied simulator software features built-in chip implementations of all the chips in the Hack chip set. If you don't implement some chips from the Hack chipset, you can still use them as chip-parts of other chips: Just rename their given stub files to, say, Mux16.hdl1. This will cause the simulator to use the built-in chip implementation

Explicit Use Of Built-in Chips



Hardware Simulator (1.4b1) - G:\TECS\tools\builtIn\Mux16.hdl

Animate: Program flow

Format: Decimal

View: Screen

Chip Name : Mux16

Time : 0

Input pins

Name	Value
a[16]	0
b[16]	0
sel	0

Output pins

Name	Value
out[16]	0

HDL

```
// MIT Press 2004. Book site: http://ww
// File name: tools/builtIn/Mux16.hdl

/**
 * 16-bit multiplexor. If sel=0 then
 */

CHIP Mux16 {

  IN  a[16], b[16], sel;
  OUT out[16];

  BUILTIN Mux;
}
```

Load Chip

Look in: builtIn

HalfAdder.hdl
Inc16.hdl
Keyboard.hdl
Mux.hdl
Mux16.hdl
Mux4Way16.hdl

File name: Mux16.hdl

Files of type: HDL Files

The chip is loaded from the **tools/buitIn** directory (includes executable versions of all the chips mentioned in the book).

Standard interface.

Built-in implementation.

Implicit Use Of Built-in Chips

```
/** Exclusive-or gate. out = a xor b */
CHIP Xor {
    IN a, b;
    OUT out;
    PARTS:
    Not(in=a,out=Nota);
    Not(in=b,out=Notb);
    And(a=a,b=Notb,out=aNotb);
    And(a=Nota,b=b,out=bNota);
    Or(a=aNotb,b=bNota,out=out);
}
```

- When any HDL file is loaded, the simulator parses its definition. For each internal chip Xxx(...) mentioned in the PARTS section, the simulator looks for an Xxx.hdl file in the same directory (e.g. Not.hdl, And.hdl, and Or.hdl in this example).
- If Xxx.hdl is found in the current directory (e.g. if it was also written by the user), the simulator uses its HDL logic in the evaluation of the overall chip.
- If Xxx.hdl is not found in the current directory, the simulator attempts to invoke the file tools/builtIn/Xxx.hdl instead.
- And since tools/builtIn includes executable versions of all the chips mentioned in the book, it is possible to build and test any of these chips before first building their lower-level parts.

Summary of Built-in Chips

- If you don’t implement some chips, you can still use them as chip-parts in other chips (the built-in implementations will kick in)
- Remember a chip cannot be used in its own implementation

Things To Do

- Perform interactive 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/

- Before moving ahead, ensure that you have designed all these chips, as we will be needing them to design the Hack Computer

Elementary Logic Gates	Multi-Way Variants	Multi-Bit Variants	Mixed Variants
• Not	• Or4way	• Not16	• Or4way16
• And	• And4way	• And16	• And4way16
• Or	• Mux4way	• Or16	• Mux4way16
• Xor	• DMux4way	• Mux16	• Mux8way16
• 2x4 Decoder		• DMux16	
• 8x3 Encoder			
• 2x1 Mux			
• 4x1 Mux			
• 1x2 Dmux			
• 1x4 Dmux			

- Whenever there is a confusion, please refer to HDL survival guide available on

http://www.arifbutt.me

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