COMPUTER ARCHITECTURE LAB2

FCIS Ainshams University
Spring2021

AGENDA

- N-bit Adder/Subtractor
- Generic Building Blocks of MIPS:
  - Sign extender
  - 32-bit ALU (Hands-on 1)

SIGN EXTENDER

Sign extension is the operation, in computer arithmetic, of increasing the number of bits of a binary number while preserving the number's sign (positive/negative) and value.

In (4)	Out(8)
1010	11111010
0101	00000101

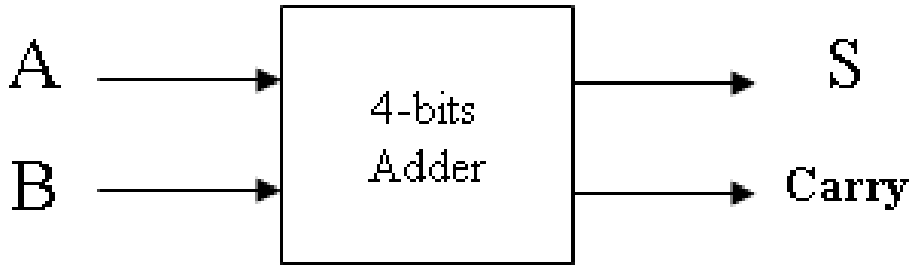
Out = “0000” & IN = X“0” & IN

Out = “1111” & IN = X“f” & IN

SIGN EXTENDER


```
Library IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
ENTITY signext IS — sign extender
PORT ( a : IN STD_LOGIC_VECTOR(15 DOWNTO 0);
        y : OUT STD_LOGIC_VECTOR(31 DOWNTO 0) );
END;
ARCHITECTURE beave OF signext IS
BEGIN
        y <= X"ffff" & a  WHEN a(15) = '1' ELSE
            X"0000" & a;
END beave;
```

4-BIT PARALLEL ADDER (1)



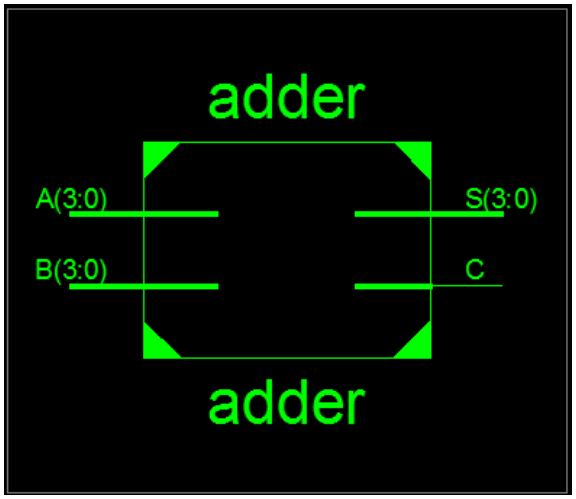
4-BIT PARALLEL ADDER (2)

```
use  IEEE.STD_LOGIC_UNSIGNED.ALL ;
ENTITY adder IS
PORT ( A: IN STD_LOGIC_VECTOR (3 DOWNTO 0);
       B: IN STD_LOGIC_VECTOR (3 DOWNTO 0);
       S: OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
       Carry : OUT STD_LOGIC);
END adder;
                          -- Q2) but what about carry??
ARCHITECTURE Behavioral OF adder IS
 BEGIN
      S <= A + B;
 ENDBehavioral;
```

Q1) Is the synthesizer inferred an 4-bit adder ?

4-BIT PARALLEL ADDER (3)

- ADDER RTL



- Part of Synthesizer report

```
=========================================================================
*                            HDL Synthesis                              *
=========================================================================

Synthesizing Unit <adder>.
    Related source file is "c:/users/amahany/desktop/test/archi2/lab2/adder.vhd".
    Found 4-bit adder for signal <S> created at line 44.
    Summary:
        inferred   1 Adder/Subtractor(s).
Unit <adder> synthesized.

=========================================================================
```

A1) YES it's inferred as Adder

4-BIT PARALLEL ADDER (4) -CARRY

```
ENTITY adder IS  PORT (
    A:   IN        STD_LOGIC_VECTOR      (3  DOWNTO   0);
    B:   IN        STD_LOGIC_VECTOR      (3  DOWNTO   0);
    S:   OUT     STD_LOGIC_VECTOR      (3  DOWNTO   0);
     Cin : IN     STD_LOGIC;
   Carry : OUT STD_LOGIC);
END adder;
ARCHITECTURE Behavioral OF adder IS
   SIGNAL Tmp: STD_LOGIC_VECTOR (4 DOWNTO 0);
    BEGIN
      Tmp <= ('0' & A) + ('0' & B)+Cin;
       S <= Tmp(3 DOWNTO 0);
       Carry <= Tmp(4);
    END Behavioral;
```

4-BIT PARALLEL ADDER (4) -CARRY

```
ENTITY adder IS  PORT (
    A:   IN        STD_LOGIC_
    B:   IN        STD_LOGIC_
    S:   OUT       STD_LOGIC_
    Cin : IN      STD_LOGIC;
   Carry : OUT STD_LOGIC)
END adder;
ARCHITECTURE Behaviora
   SIGNAL Tmp: STD_LOG
    BEGIN
     Tmp <= ('0' & A) + ('0' &
      S <= Tmp(3 DOWNTO
      Carry <= Tmp(4);
   END Behavioral;
```

Q3) What if we want to implement parallel subtractor?

Tmp <= ('0' & A) + ('0' & B')+1;

Q4) What if we want to implement parallel adder/ subtractor?

Tmp <= ('0' & A) + ('0' & BB)+Cin;

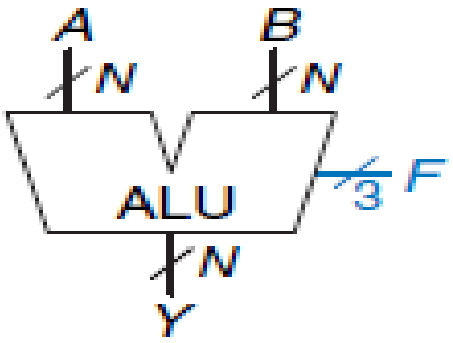
1. Cin is 0 when add, 1 when sub
2. BB = B when Cin=0
   B' when Cin =1

N-BIT ALU

- An Arithmetic/Logical Unit (ALU) combines a variety of mathematical and logical operations.
- ALU might perform addition, subtraction, magnitude comparison, AND, and OR operations.
- The ALU forms the heart of most computer systems.

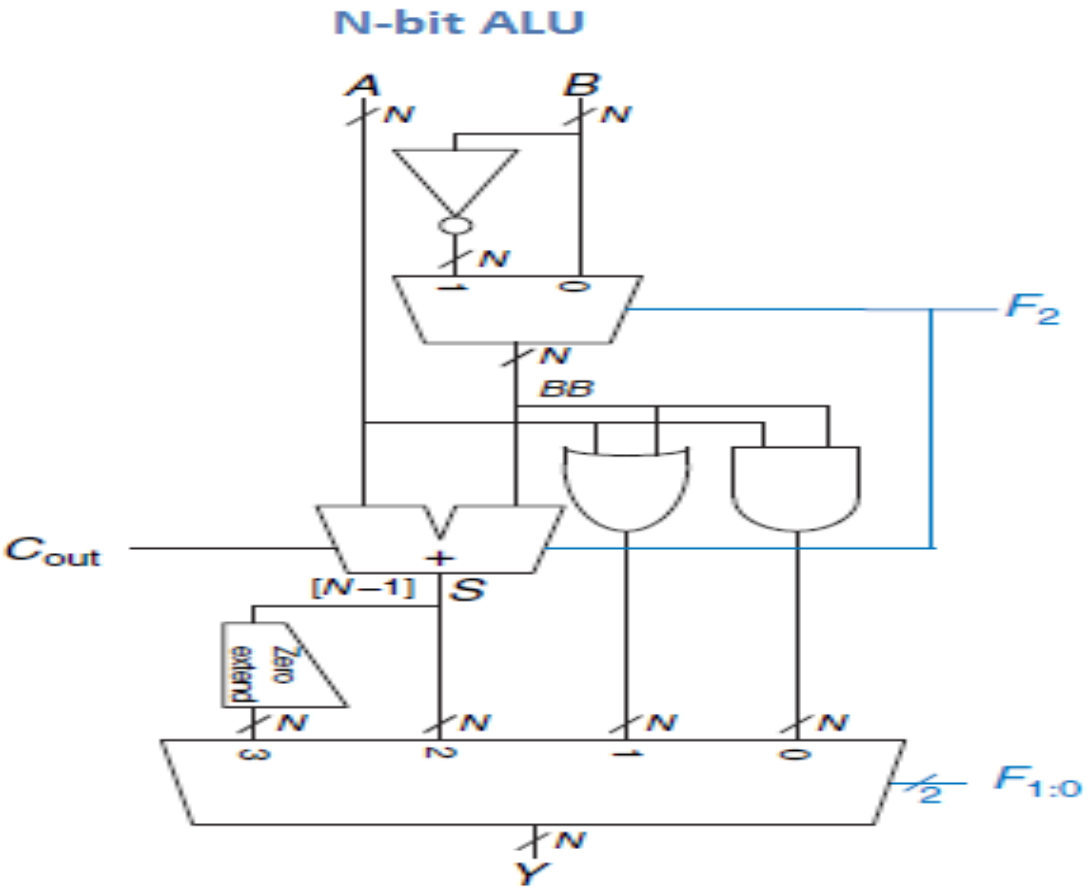
N-BIT ALU

- The ALU receives a control signal F that specifies which function to perform.
- A and B is input Signal.
- Y is output signal

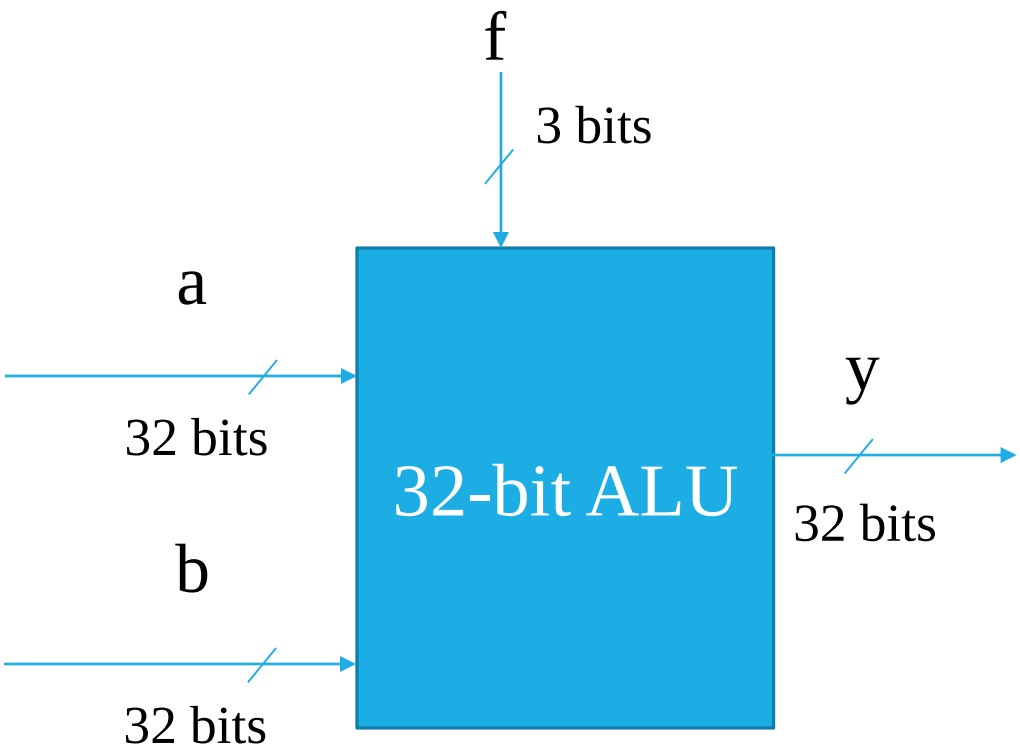


$F_{2:0}$	Function
000	A AND B
001	A OR B
010	A + B
011	not used
100	A AND $\overline{B}$
101	A OR $\overline{B}$
110	A – B
111	SLT

N-BIT ALU IMPLEMENTATION



HANDS-ON 1: 32-BIT ALU



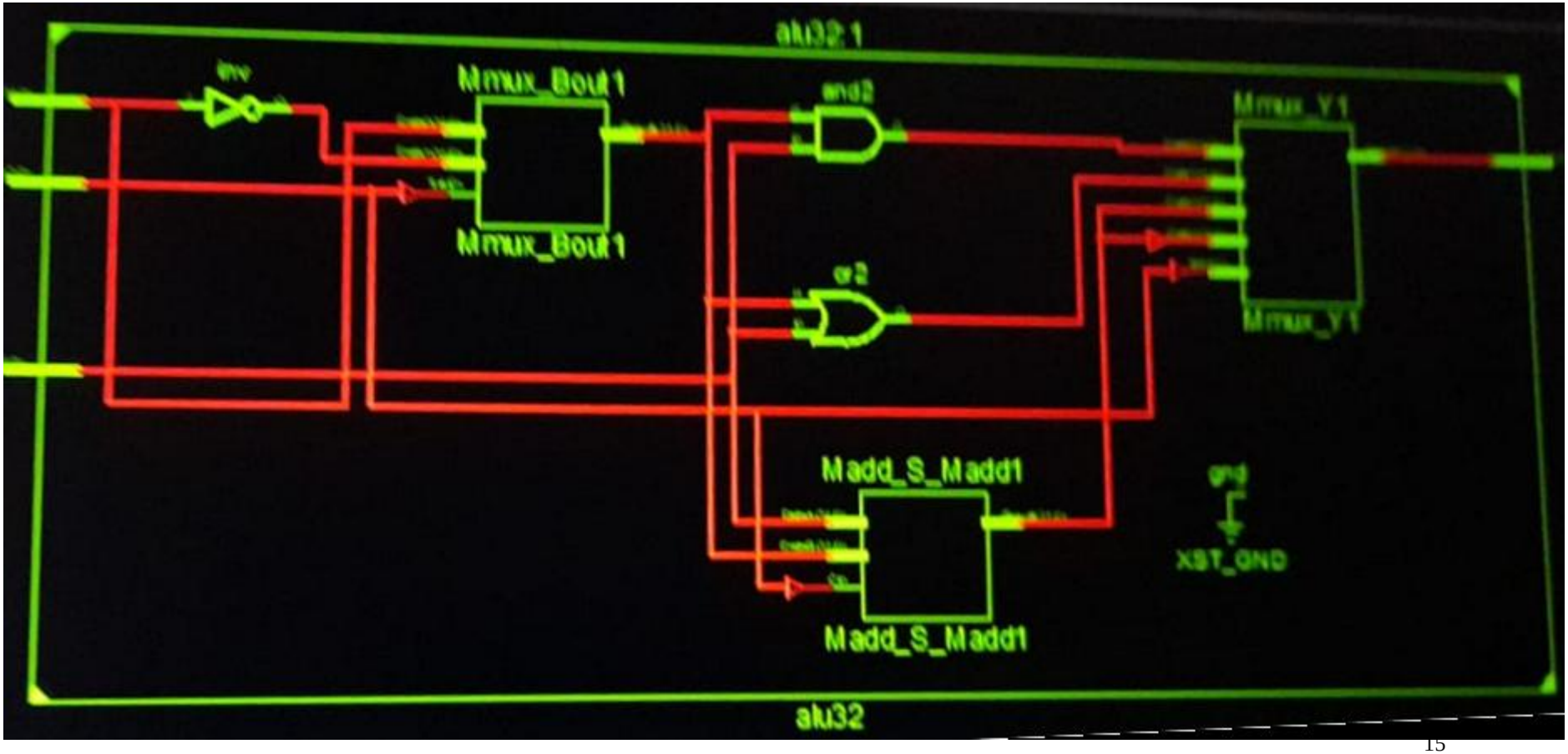
$F_{2:0}$	Function
000	A AND B
001	A OR B
010	A + B
011	not used
100	A AND $\overline{B}$
101	A OR $\overline{B}$
110	A – B
111	SLT

HANDS-ON 1: 32-BIT ALU

```
 entity alu32 is
 port(A, B: in STD_LOGIC_VECTOR(31 downto 0);
 F: in STD_LOGIC_VECTOR(2 downto 0);
 Y: out STD_LOGIC_VECTOR(31 downto 0));
 end;
architecture synth of alu32 is
signal S, BB: STD_LOGIC_VECTOR(31 downto 0);
begin
BB <= (not B) when (F(2) = '1') else B;
S <= A + Bout + F(2);
Y <= A and Bout when F(1 downto 0) =”00” ELSE
       A or Bout when F(1 downto 0) =”01” ELSE
       S when F(1 downto 0) =”10” ELSE
      ("00000000000000000000000000000000" & S(31)) when F(1 downto 0) =”11”
       ELSE X“00000000";
End synth;
```

HANDS-ON 1: 32-BIT ALU

ALU RTL



Thanks