Digital Systems I

Lab05 Slides based on **Chapter 4 - Part II**

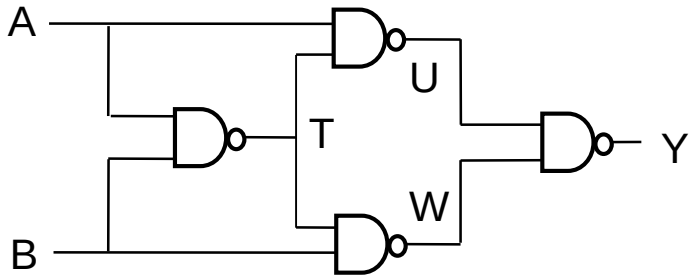
Hierarchical Designs

and

Structural Modeling

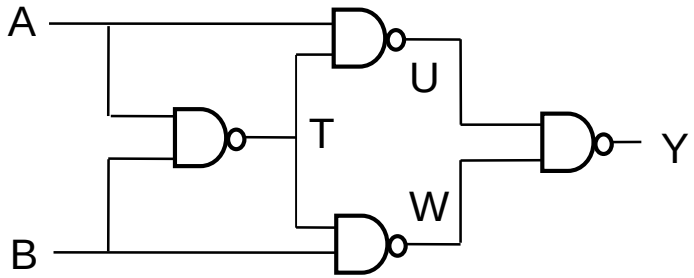
N. Tabrizi Kettering University

Single-bit partial Comparator



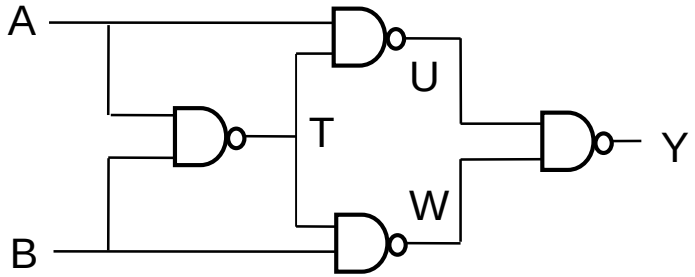
Single-bit partial Comparator

```
ENTITY comp_1 IS
PORT (A, B  : IN    STD_LOGIC;
          Y  : OUT STD_LOGIC
     );
END comp_1;
```

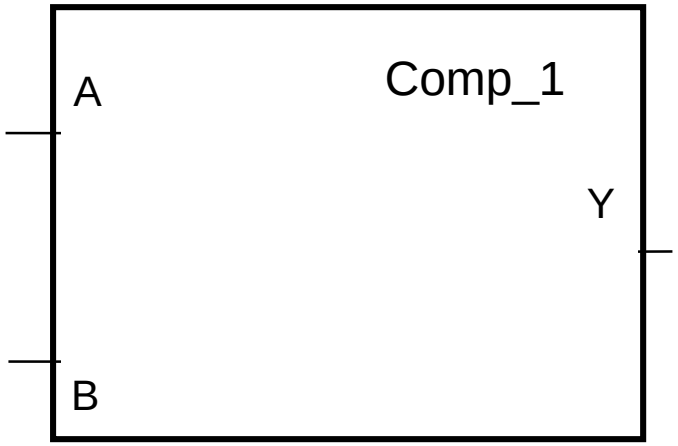


Single-bit partial Comparator

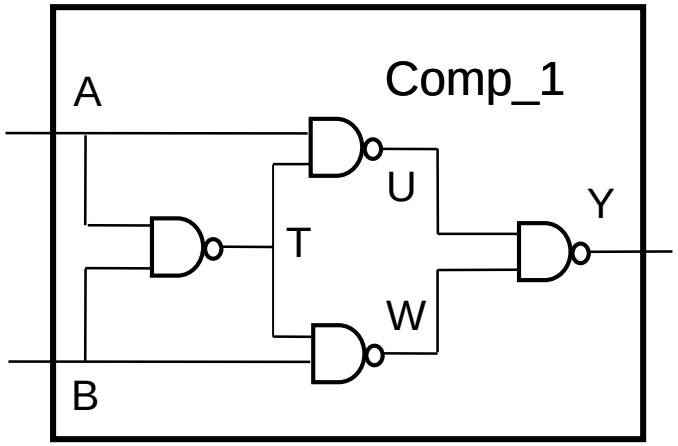
```
ENTITY comp_1 IS
PORT (A, B  : IN    STD_LOGIC;
           Y  : OUT STD_LOGIC
     );
END comp_1;

ARCHITECTURE Algebraic OF comp_1 IS
   SIGNAL T, U, W: STD_LOGIC;
BEGIN
   Y   <= W NAND U;
   W   <= B NAND T;
   U   <= A NAND T;
   T   <= A NAND B;
END Algebraic;
```



Symbol (Graphical Entity)

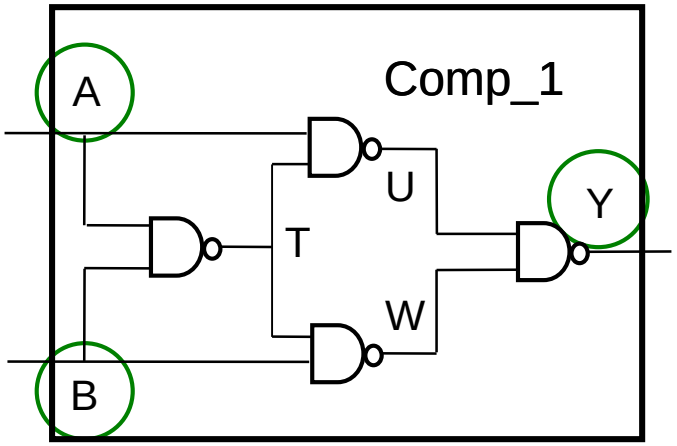


Component



Definitions:

Component



Why do we like Components?

Because they are REUSABLE

- **Formals**: Inputs/Outputs of component
- **Instantiate**: Copy and paste component into new design
- **Instance**: Resulting copy
- **Actuals**: Inputs/Outputs of instance (see next slide)

Structural Modeling

Shows how components are connected together

Structural Modeling

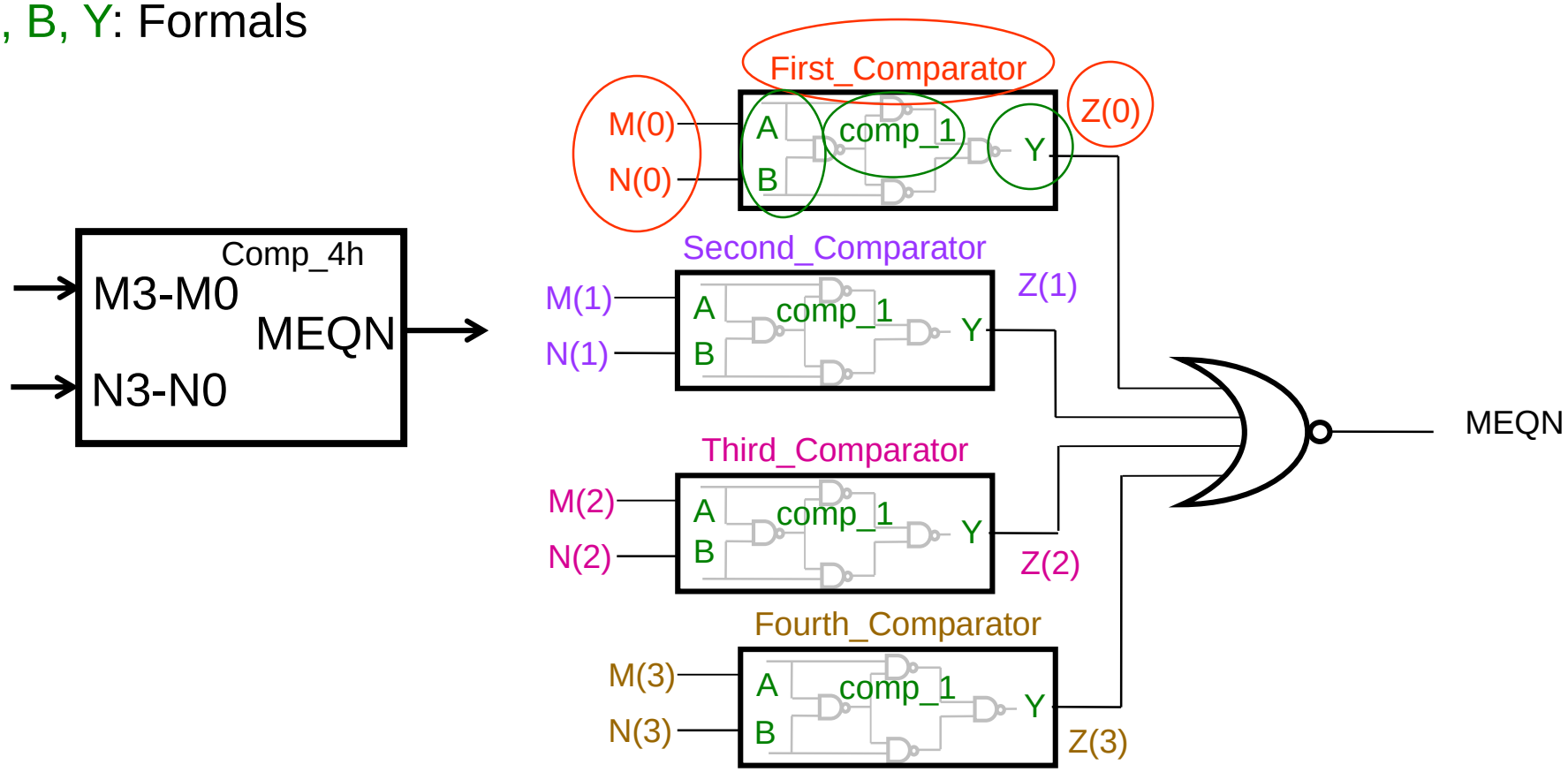
4-bit partial Comparator

First_Comparator: Instance Name

Com_1: Component Name

M(0), N(0), Z(0): Actuals

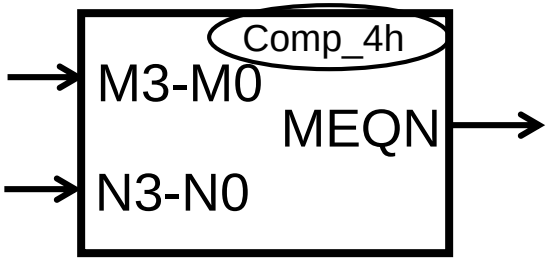
A, B, Y: Formals

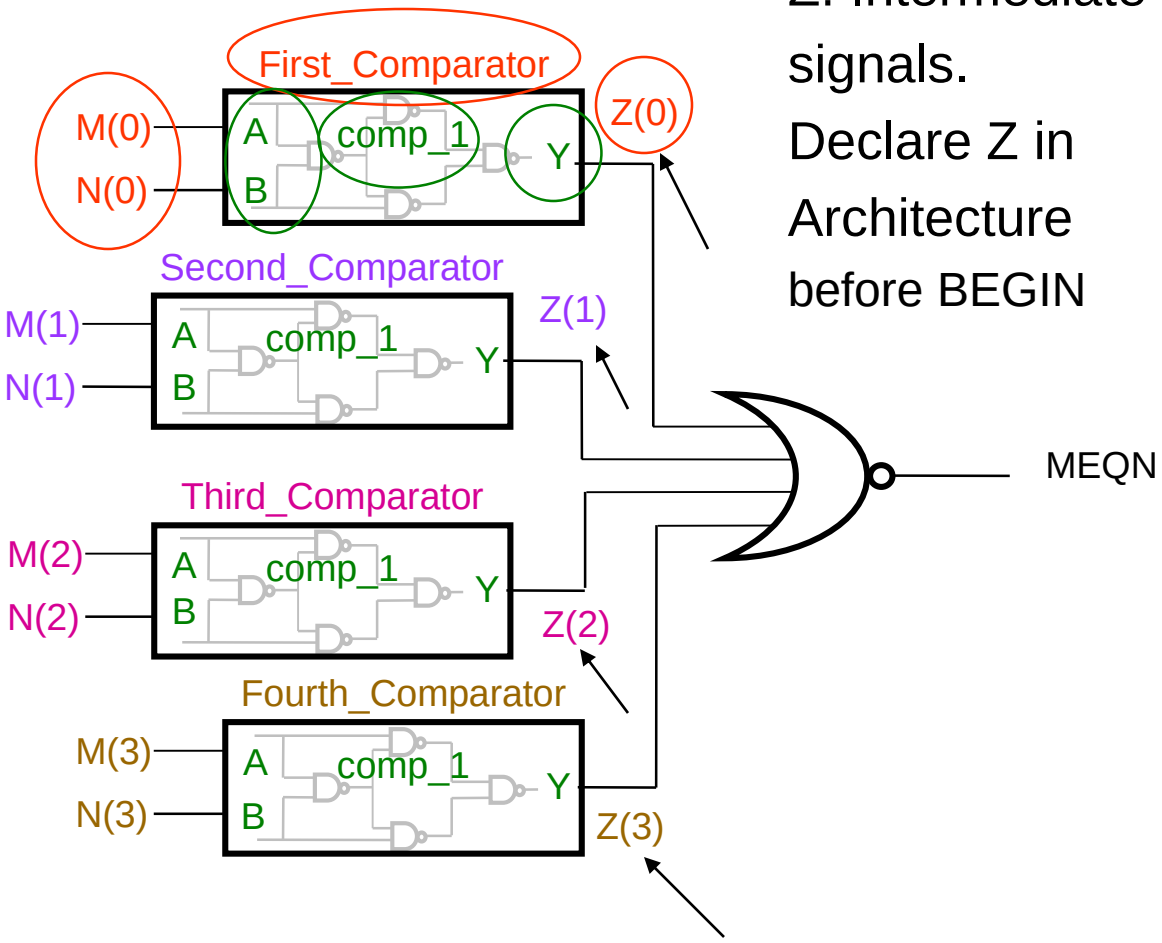


Structural Modeling

4-bit partial Comparator

```
ENTITY comp_4h IS
PORT (M, N   : IN     STD_LOGIC_VECTOR (3 DOWNTO 0);
      MEQN : OUT STD_LOGIC
      );
END comp_4h;
```



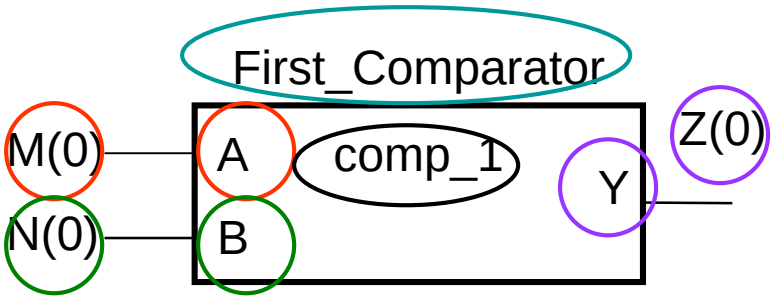


Z: Intermediate signals. Declare Z in Architecture before BEGIN

```
ARCHITECTURE Structure OF comp_4h IS

COMPONENT comp_1
    PORT (A, B: IN      STD_LOGIC;
          Y: OUT   STD_LOGIC
         );
END COMPONENT;

    SIGNAL Z: STD_LOGIC_VECTOR (3 DOWNTO 0);

BEGIN

First_Comparator: comp_1
PORT MAP          (M(0), N(0), Z(0));   -- Actuals

END Structure;
```

NOTE: Add component declarations to ARCHTECTURE (See this page) Or put them in a separate file called a package (Coming up …)

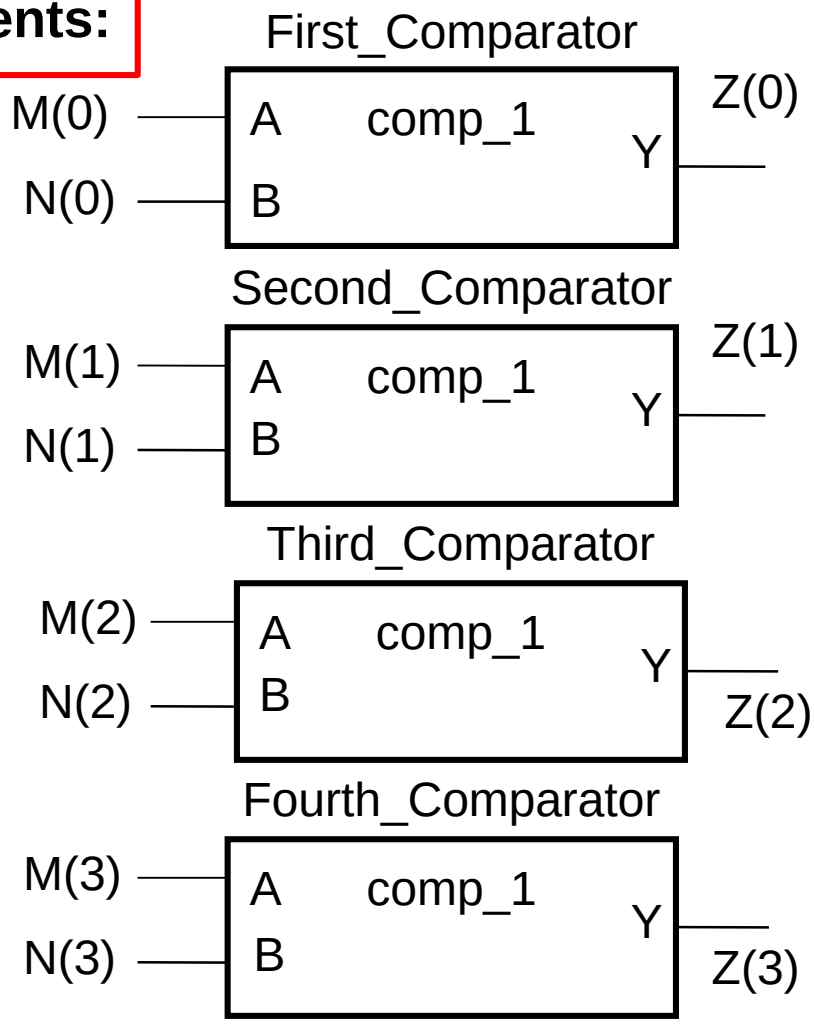
How to instantiate components:



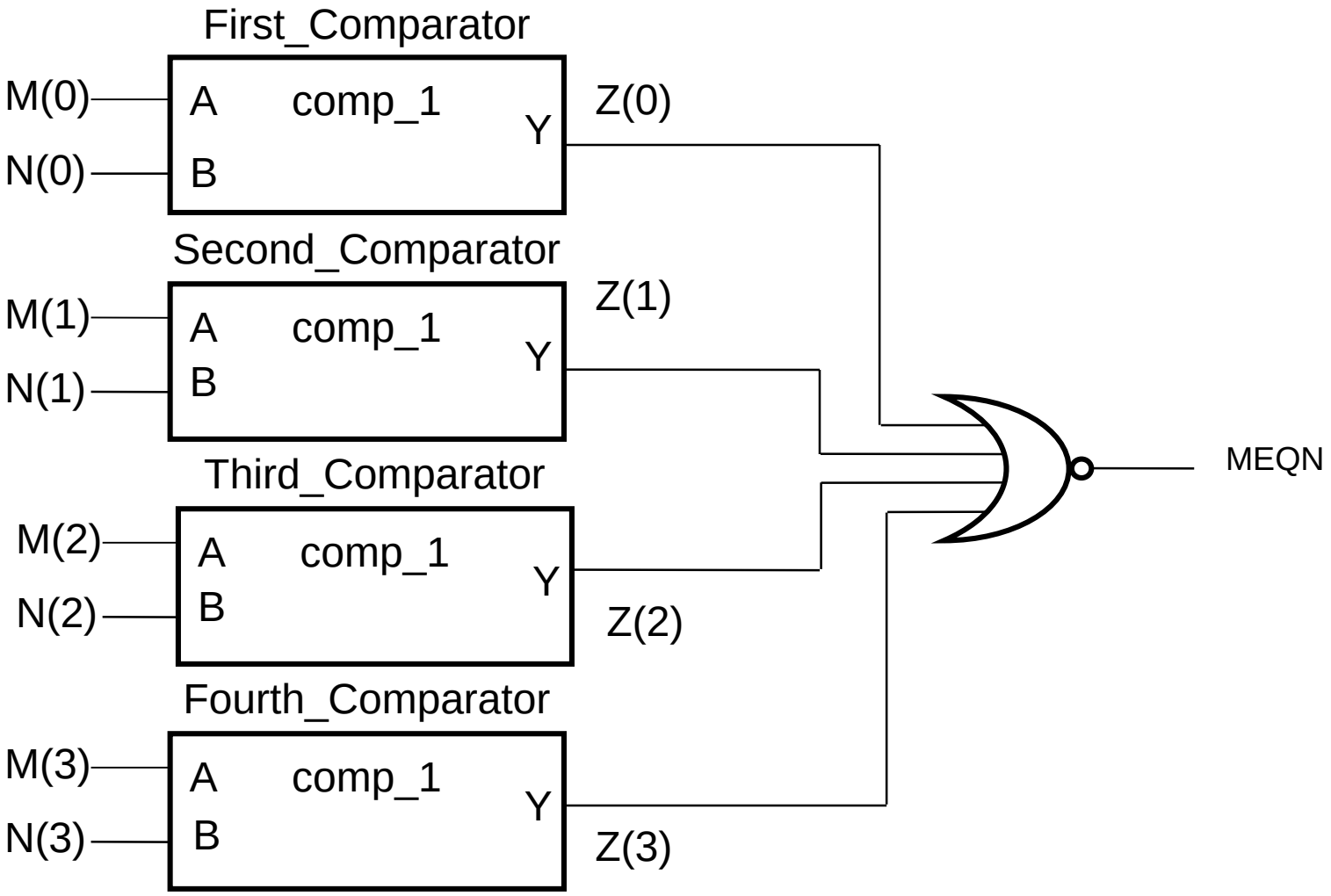
NOTE: Write actuals in the same order that the formals are written.

How to instantiate components:

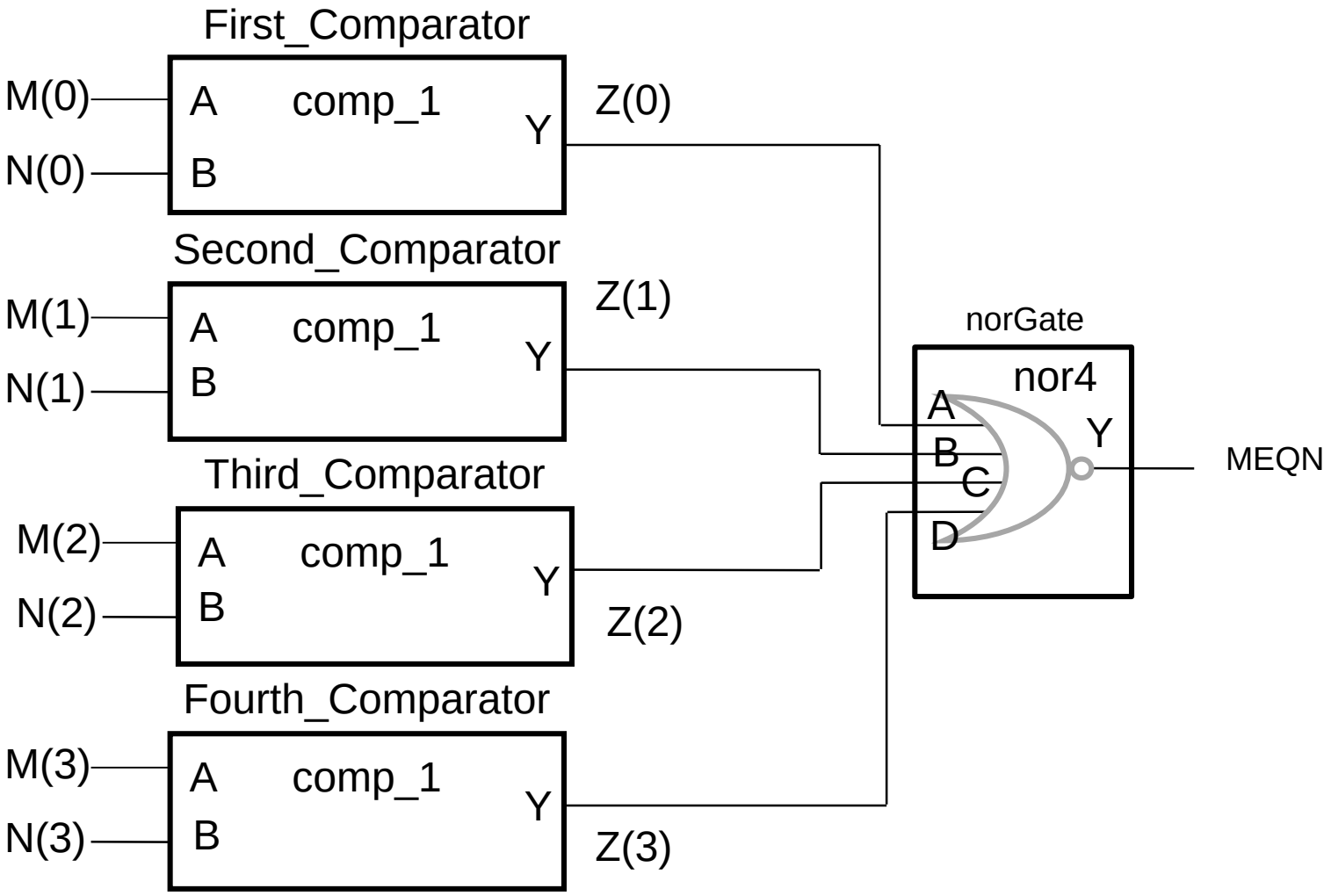
```
BEGIN
First_Comparator: comp_1
PORT MAP   (M(0),  N(0), Z(0));

Second_Comparator: comp_1
PORT MAP   (M(1), N(1), Z(1));

Third_Comparator: comp_1
PORT MAP   (M(2), N(2), Z(2));

Fourth_Comparator: comp_1
PORT MAP   (M(3), N(3), Z(3));

MEQN <= NOT (Z(3) OR Z(2) OR Z(1) OR Z(0));

END Structure;
```



Note: NOR is added algebraically.



Note: NOR is added algebraically.



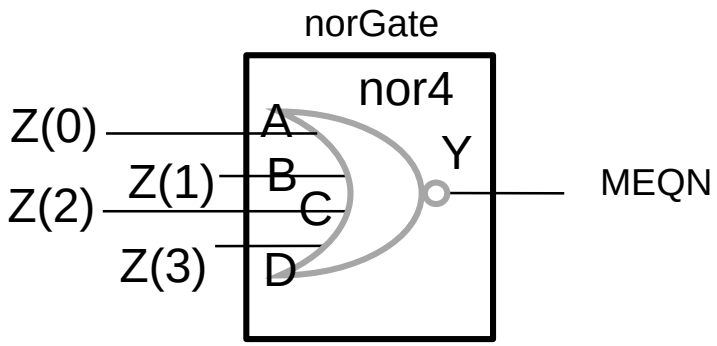
Note: NOR may also be added structurally instead of algebraically.

```
ARCHITECTURE Structure OF comp_4h IS

COMPONENT comp_1
    PORT (A, B  : IN      STD_LOGIC;
              Y  : OUT    STD_LOGIC
          );
END COMPONENT;

COMPONENT nor4
    PORT (A, B, C, D     : IN      STD_LOGIC;
                  Y      : OUT     STD_LOGIC
          );
END COMPONENT;

SIGNAL Z: STD_LOGIC_VECTOR (3 DOWNTO 0);
```

Declare a new component: nor4.

```
BEGIN
First_Comparator: comp_1
PORT MAP   (M(0),  N(0), Z(0));

Second_Comparator: comp_1
PORT MAP   (M(1), N(1), Z(1));

Third_Comparator: comp_1
PORT MAP   (M(2), N(2), Z(2));

Fourth_Comparator: comp_1
PORT MAP   (M(3), N(3), Z(3));
```

Instantiate nor4.




```
norGate: nor4
PORT MAP (Z(0), Z(1), Z(2), Z(3), MEQN);
```

Now NOR is added structurally.

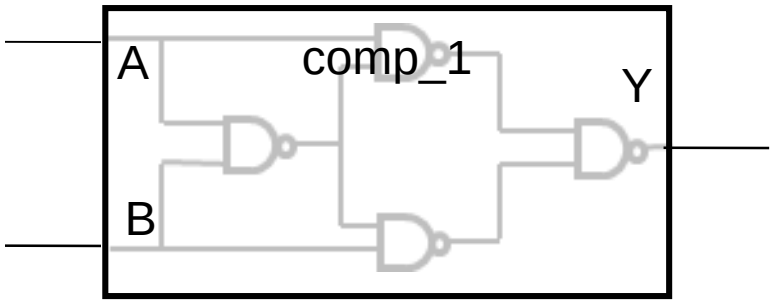
```
END Structure;
```

Describe component nor4

```
ENTITY nor4 IS
PORT (A, B, C, D    : IN     STD_LOGIC;
                Y       : OUT STD_LOGIC
      );
END nor4;

ARCHITECTURE algebraic OF nor4 IS
BEGIN
   Y <= NOT(A OR B OR C OR D):
END algebraic;
```

Lab05 Lecture Exercise

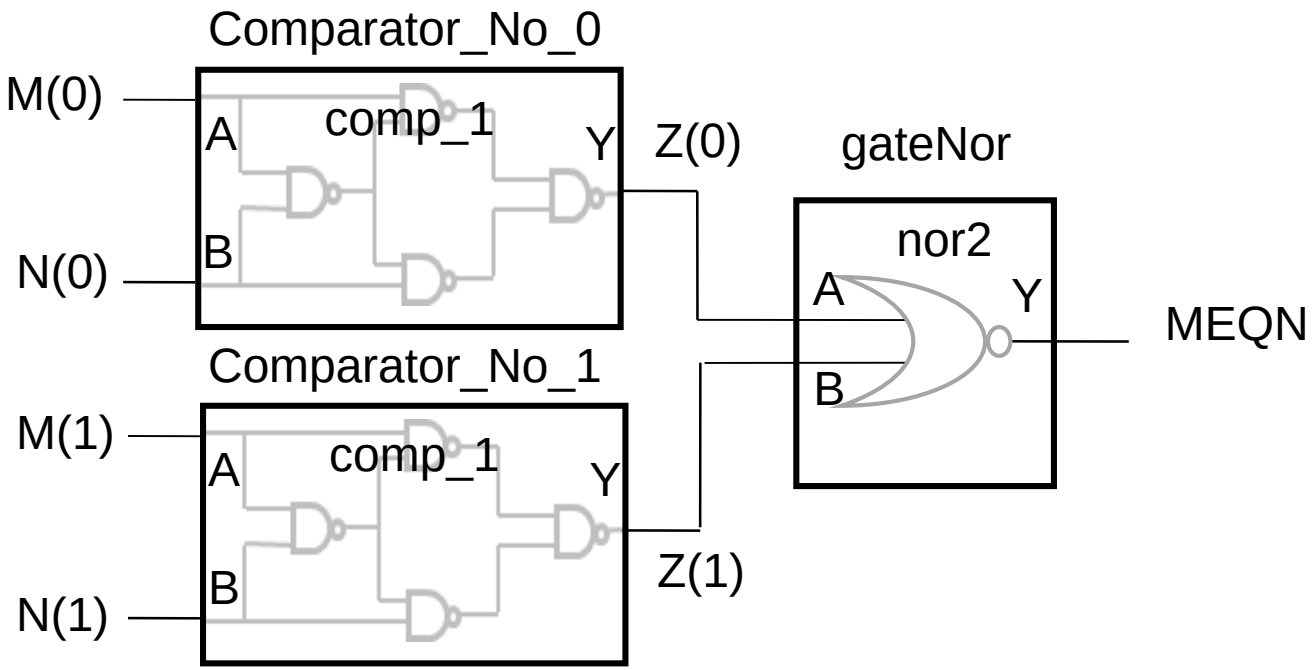
1.



What is the component's name?

What are the formals of the component?

Lab05 Lecture Exercise



What is the name of first instance of comp_1 (the one on top)?

Write the actuals of the second instance of comp_1:

A 4-bit partial comparator is here.

Convert it to a 2-bit comparator:

```
ENTITY comp_4h IS
PORT (  M, N   : IN       STD_LOGIC_VECTOR (3 DOWNTO 0);
          MEQN : OUT    STD_LOGIC     -- Active-high output
        );
END comp_4h;

ARCHITECTURE Structure OF comp_4h IS
-- Intermediate signals
          SIGNAL Z: STD_LOGIC_VECTOR (3 DOWNTO 0);

  COMPONENT comp_1   -- Do NOT retype, copy/paste from ENTITY
  PORT (A,   B  : IN       STD_LOGIC;
                Y   : OUT   STD_LOGIC
          );
  END COMPONENT;
```

```
  COMPONENT nor4
  PORT (A, B, C, D  : IN      STD_LOGIC;
                    Y    : OUT   STD_LOGIC
          );
END COMPONENT;

BEGIN
          First_Comparator                    : comp_1         PORT MAP
(M(0), N(0), Z(0));
   Second_Comparator   : comp_1 PORT MAP  (M(1), N(1), Z(1));
   Third_Comparator      : comp_1 PORT MAP  (M(2), N(2), Z(2));
   Fourth_Comparator    : comp_1 PORT MAP  (M(3), N(3), Z(3));

   gateNor                    : nor4   PORT MAP(Z(3), Z(2), Z(1), Z(0), MEQN);
END Structure;
```

-- NOR gate is added structurally (not algebraically)

-- NOR gate needs its own file (entity and architecture) not shown here.

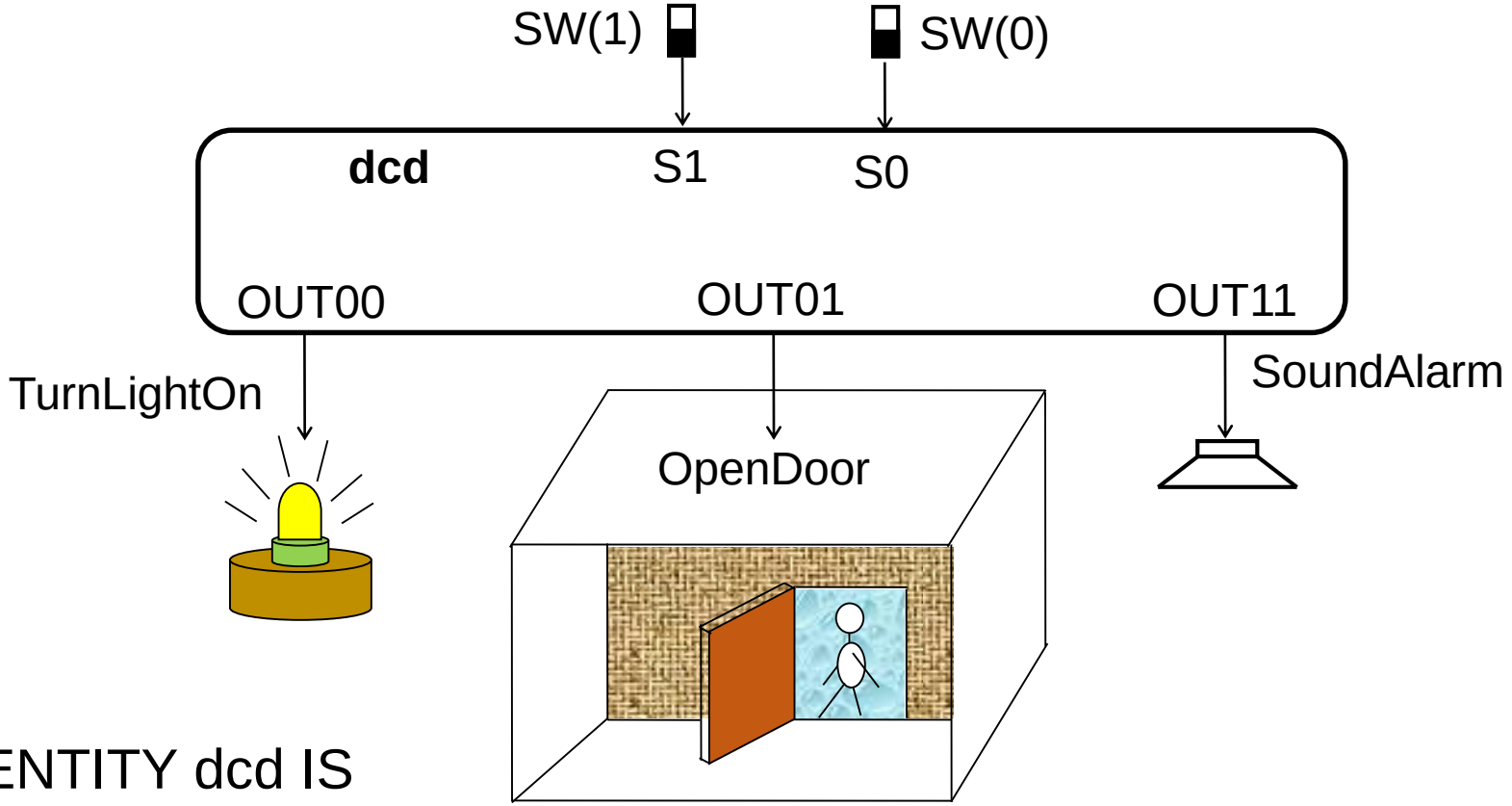
-- You need to compile **ALL files** when doing simulation or synthesis.

2. Scott has built a control system, but when

He wants to open the door, the alarm sounds instead!

He tries to sound the alarm, the light turns on instead!

He wants to turn the light on, the door opens instead!

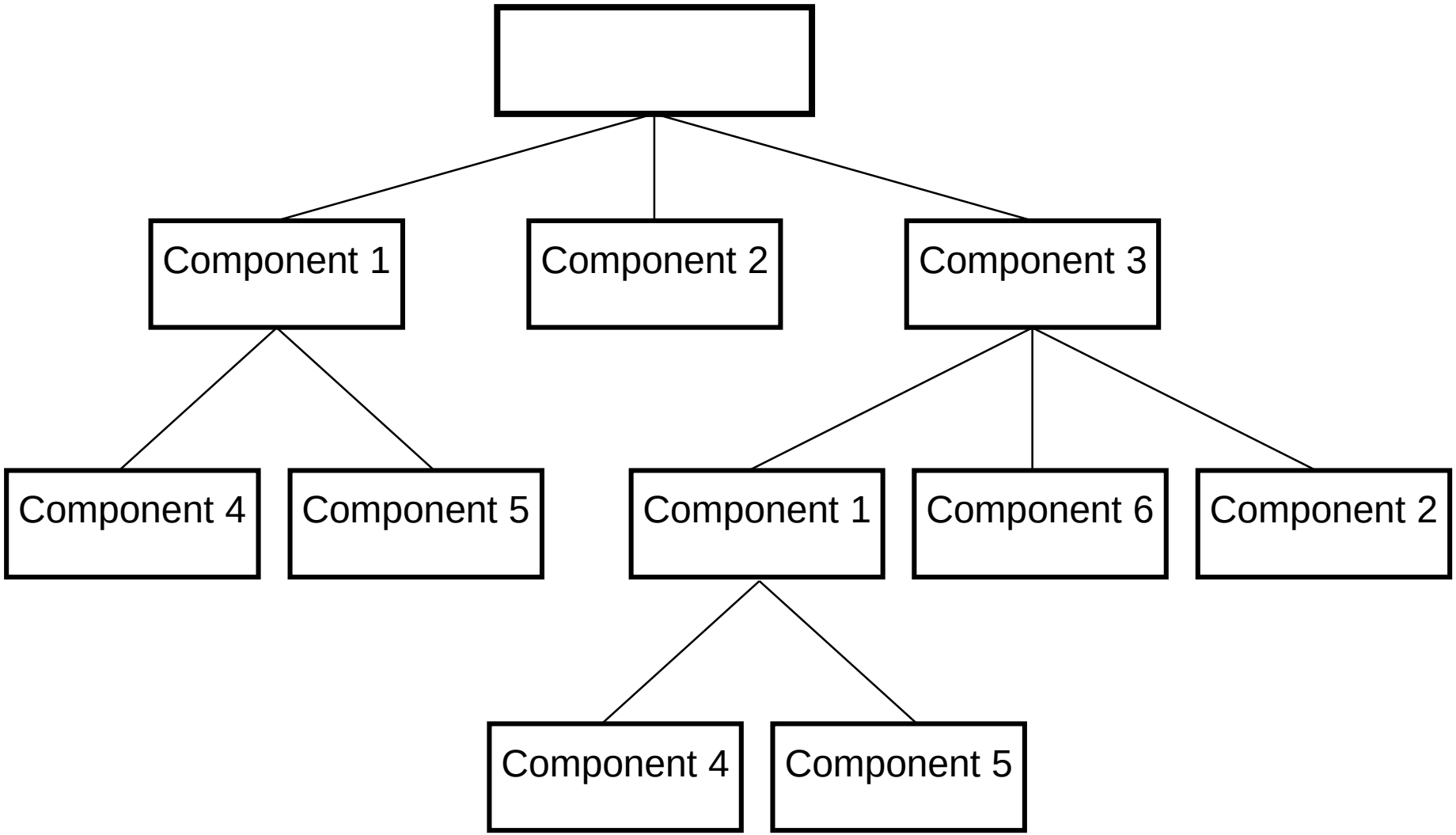



```
ENTITY dcd IS
PORT (          s1, s0                  : IN    STD_LOGIC;
        OUT00, OUT01, OUT11             : OUT   STD_LOGIC );
END dcd;
```

***

```
Controller: dcd     -- Controller is instantiated here
PORT MAP (SW(1), SW(0), OpenDoor, SoundAlarm TurnLightOn);
```

Hierarchical Design

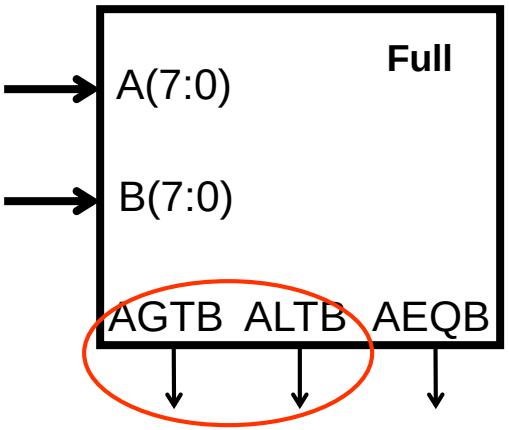


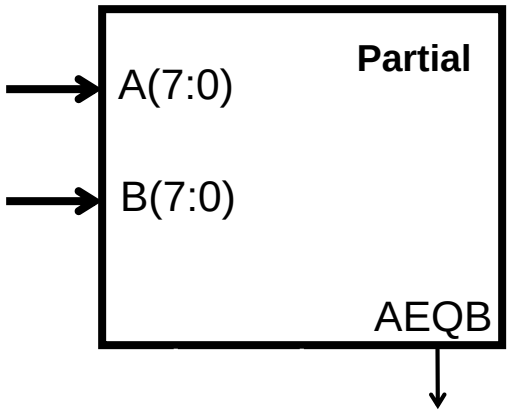
Preparing for lab05 of Digitals Systems I

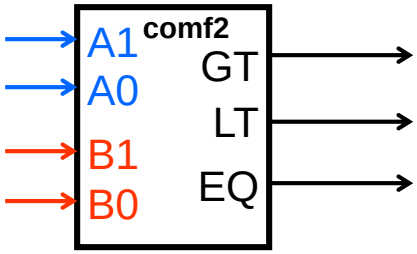
Multi-bit Full Comparator

Preparing for lab 05

Needs Your Close Attention







2-bit **Full** Comparator

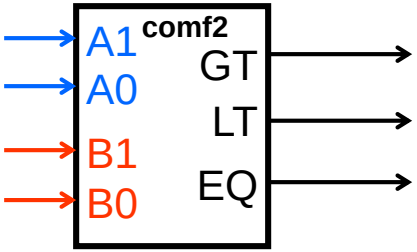
Truth Table in Q17 of HW01

Row	A_1	A_0	B_1	B_0	GT	LT	EQ	Row	A_1	A_0	B_1	B_0	GT	LT	EQ
0	0	0	0	0	0	0	1	8	1	0	0	0	1	0	0
1	0	0	0	1	0	1	0	9	1	0	0	1	1	0	0
2	0	0	1	0	0	1	0	10	1	0	1	0	0	0	1
3	0	0	1	1	0	1	0	11	1	0	1	1	0	1	0
4	0	1	0	0	1	0	0	12	1	1	0	0	1	0	0
5	0	1	0	1	0	0	1	13	1	1	0	1	1	0	0
6	0	1	1	0	0	1	0	14	1	1	1	0	1	0	0
7	0	1	1	1	0	1	0	15	1	1	1	1	0	0	1

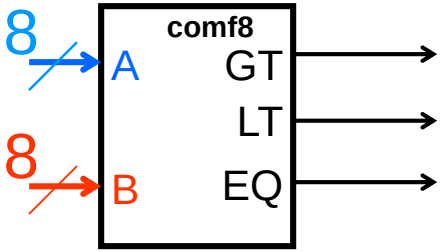
Pre-lab: Draw K-maps. Get minimal SOP & POS

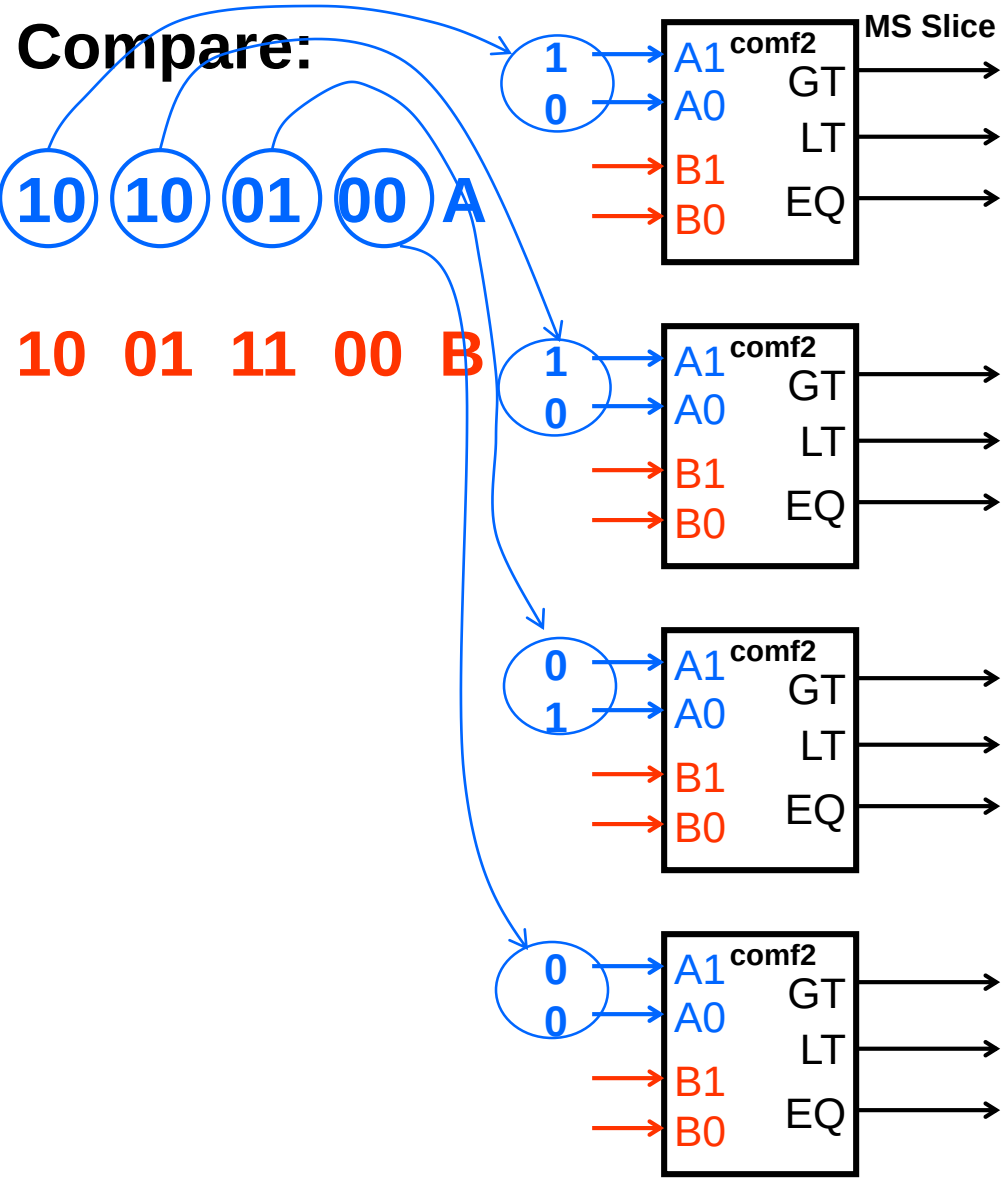
Use **2-bit comparator** slices to get an **8-bit comparator**

What we have

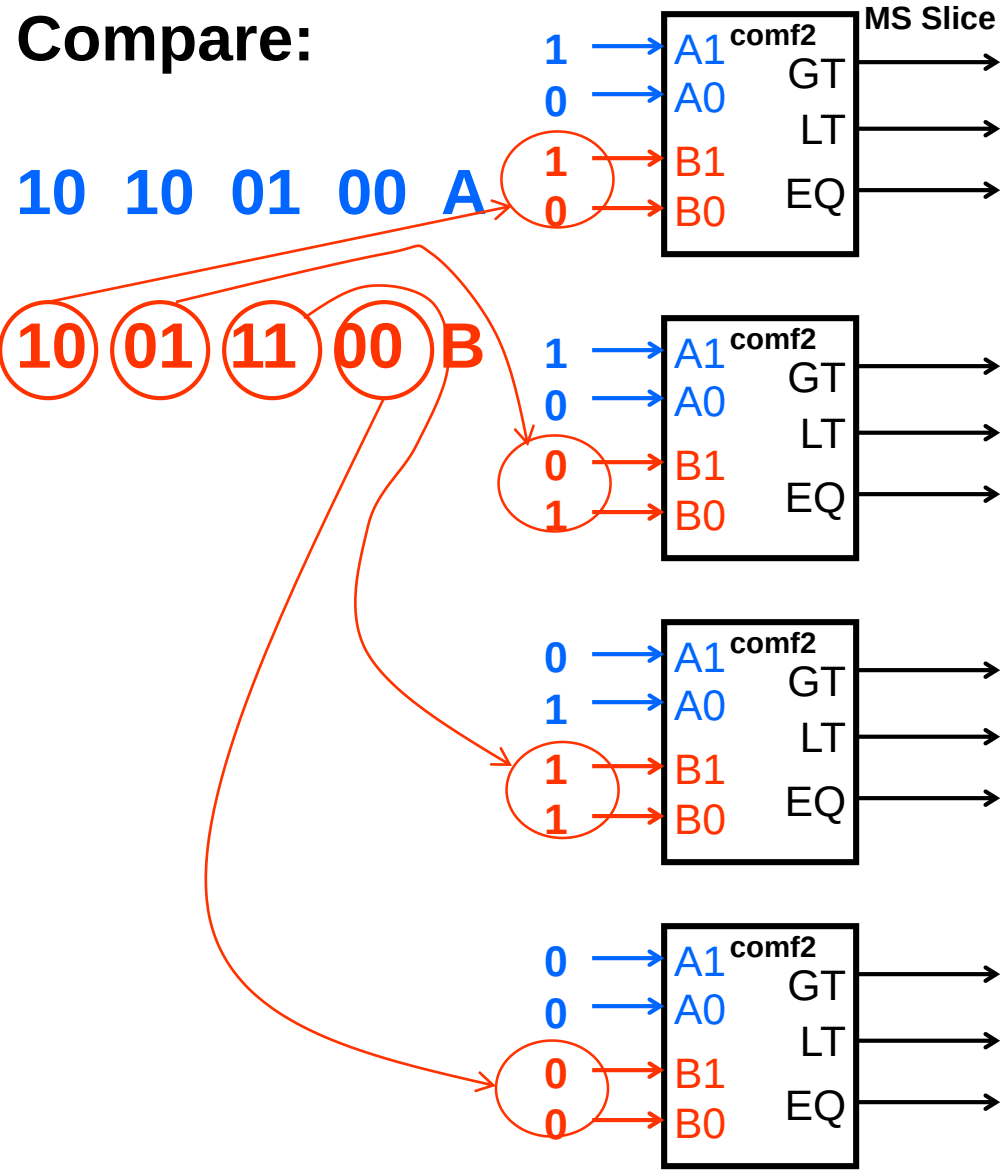


What we need




Compare:
10 10 01 00 A
10 01 11 00 B
MS Slice
comf2
A1
A0
B1
B0
GT
LT
EQ
1
0
0
1
0
0

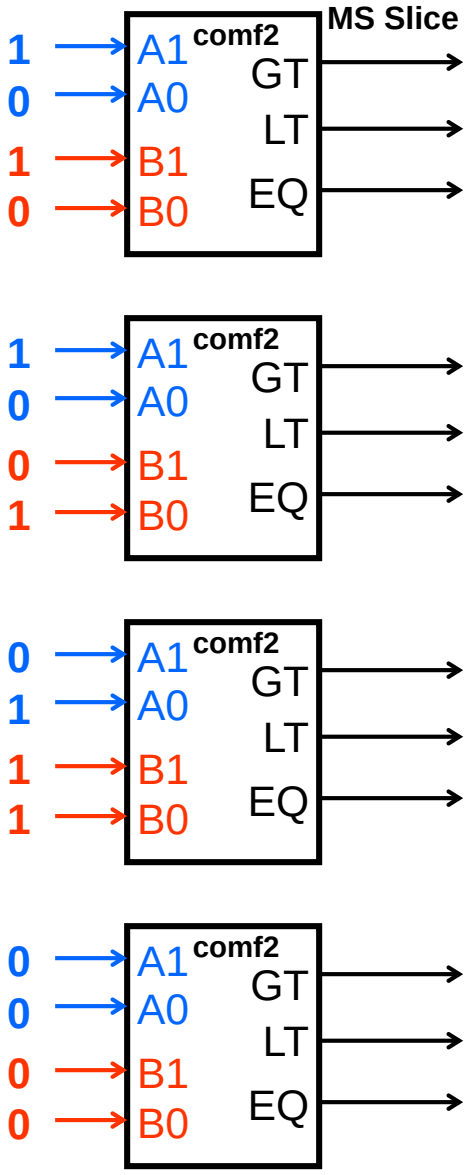
Compare:


10 10 01 00 A
10 01 11 00 B
MS Slice
comf2
A1
A0
B1
B0
GT
LT
EQ
1
0
1
0
1
0
0
1
0
1
1
1
0
0
0
0

Compare:

10 10 01 00 A

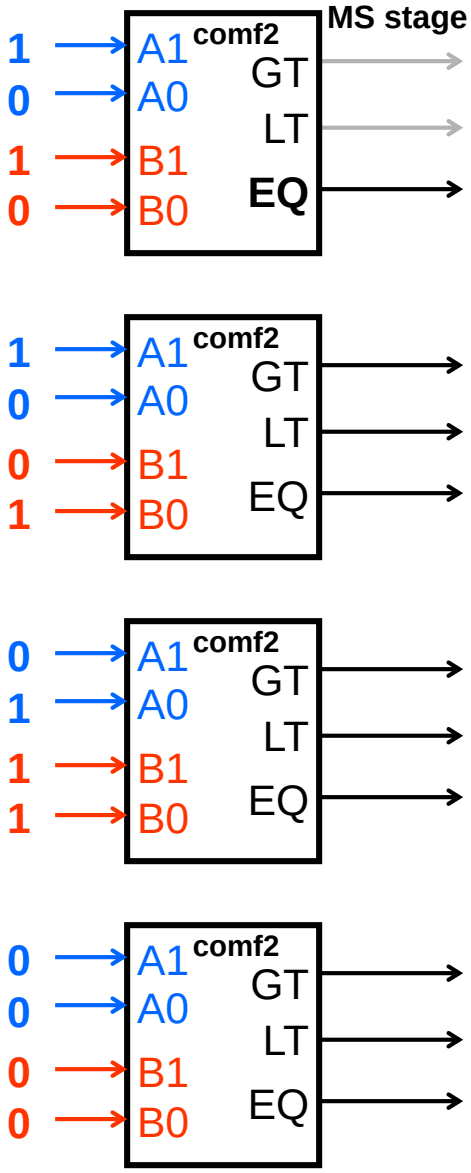
10 01 11 00 B



Compare:

10 10 01 00 A

10 01 11 00 B



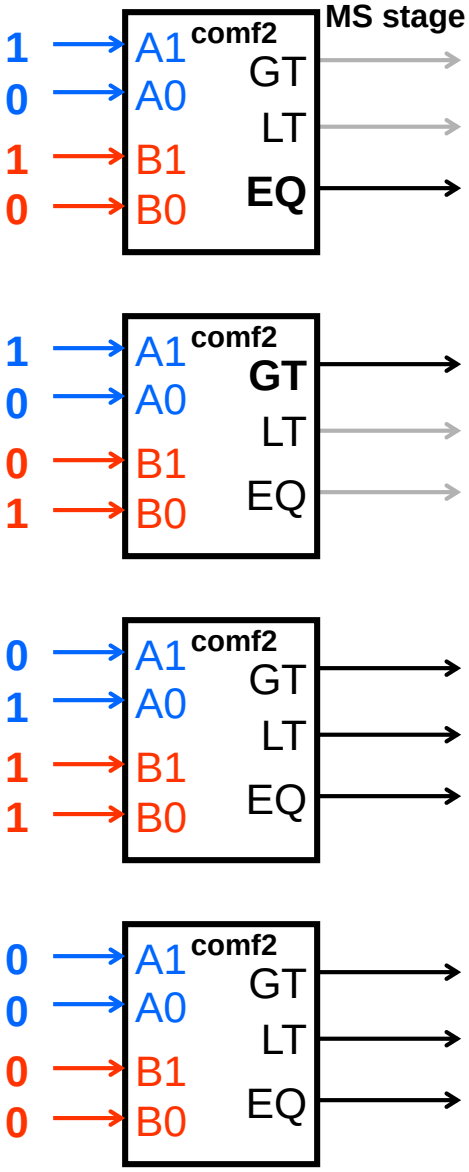
In my stage **A** = **B**

Give this message to lower stage

Compare:

10 10 01 00 A

10 01 11 00 B



In my stage **A = B**

Give this message to lower stage

In my stage **A > B**

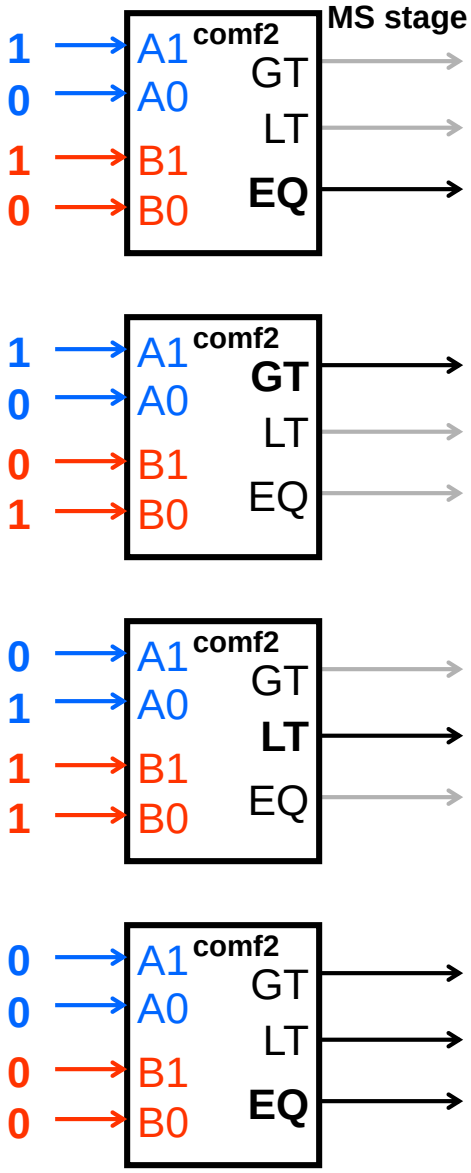
This is final message.

Give it to lower stage.

Compare:

10 10 01 00 A

10 01 11 00 B



In my stage **A = B**

Give this message to lower stage

In my stage **A > B**

This is final message.

Give it to lower stage.

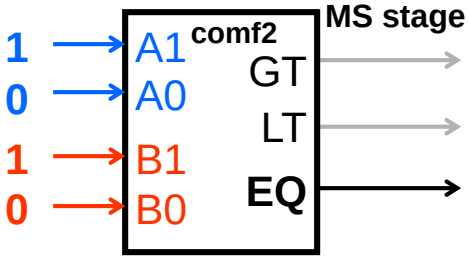
In my stage **A < B**

But tell next stage **A > B**

Compare:

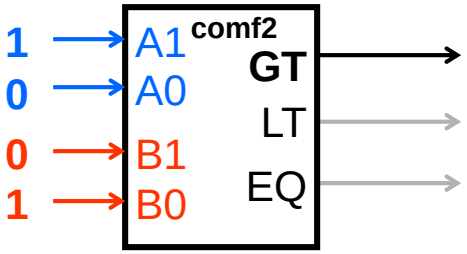
10 10 01 00 A

10 01 11 00 B



In my stage **A = B**

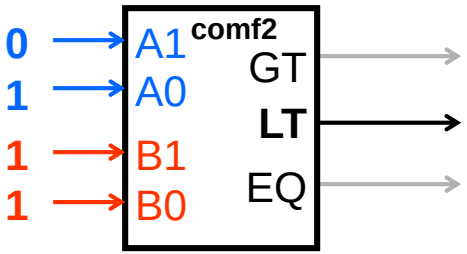
Give this message to lower stage



In my stage **A > B**

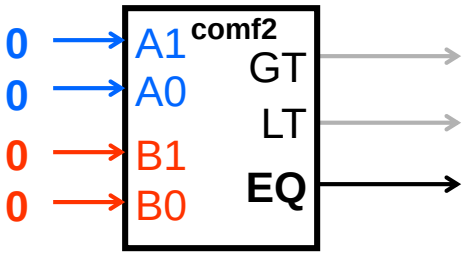
This is final message.

Give it to lower stage.



In my stage **A < B**

But tell next stage **A > B**



In my stage **A = B**

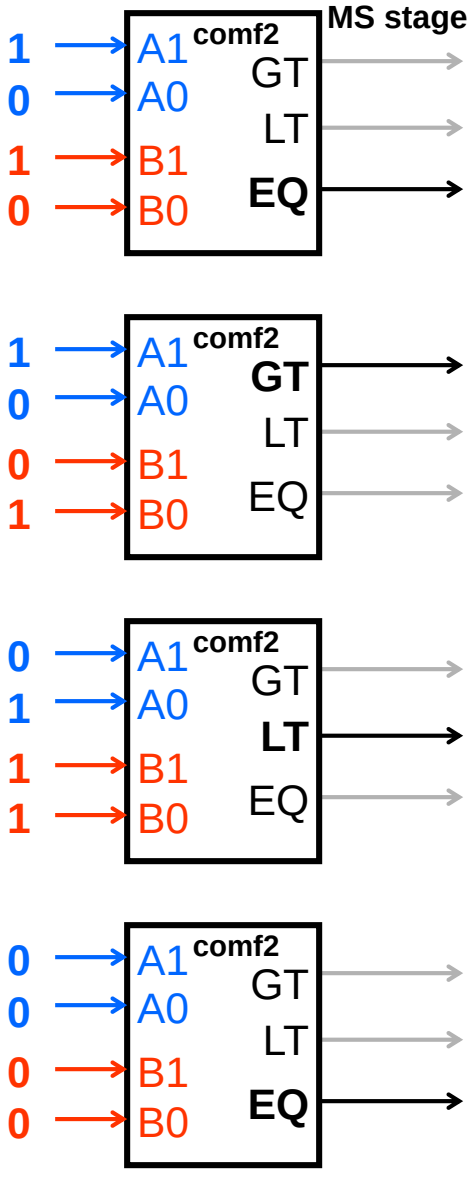
But final message is **A > B**

Compare:

10 10 01 00 A

10 01 11 00 B

What is the **FINAL** Decision?



In my stage **A = B**
Give this message to lower stage

In my stage **A > B**
This is final message.
Give it to lower stage.

In my stage **A < B**
But tell next stage **A > B**

In my stage **A = B**
But final message is **A > B**

Final decision: A > B

Definition:

A **message** tells us whether A > B, A < B or A = B

Conclusion:

Each stage receives 2 messages:

One from its own comparator, one from upper level

And sends one of them to the lower level using these rules:

1. If more significant stage tells you A > B or A < B, listen to it, no matter what your comparator says.
2. If the more significant stage tells you A = B, listen to your comparator.

Replace English Conversation with an Electronic Conversation!

Replace English Conversation with an Electronic Conversation!

New component: ei (**E**xpansion **I**nterface)

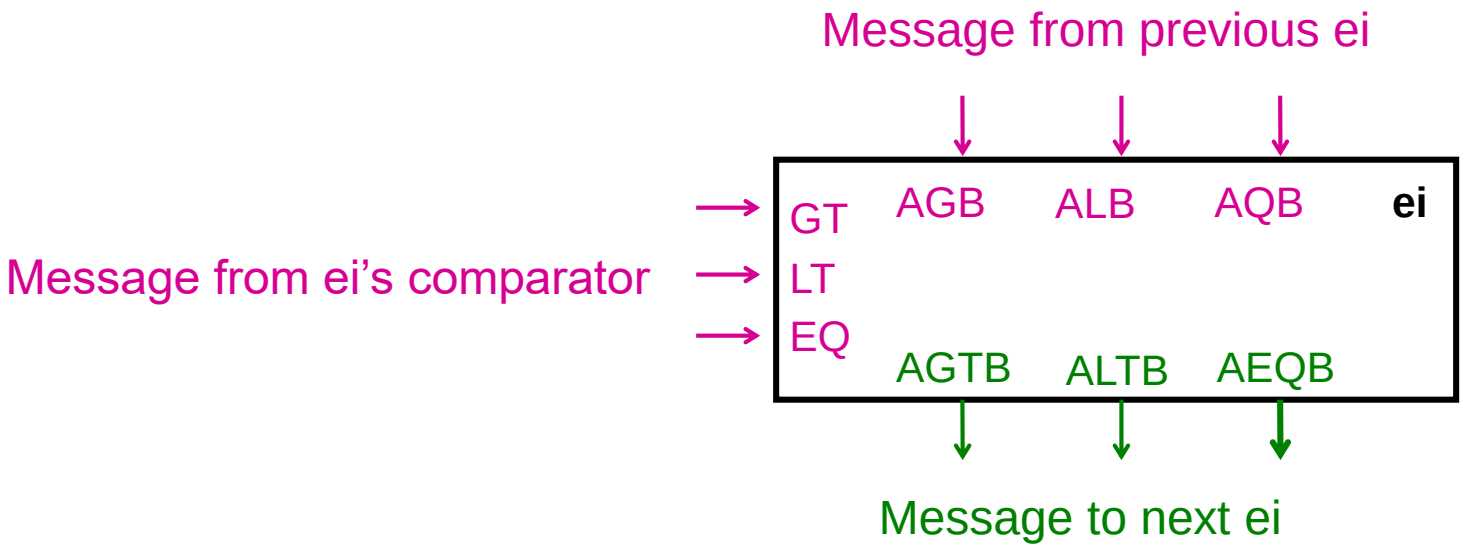
Replace English Conversation with an Electronic Conversation!

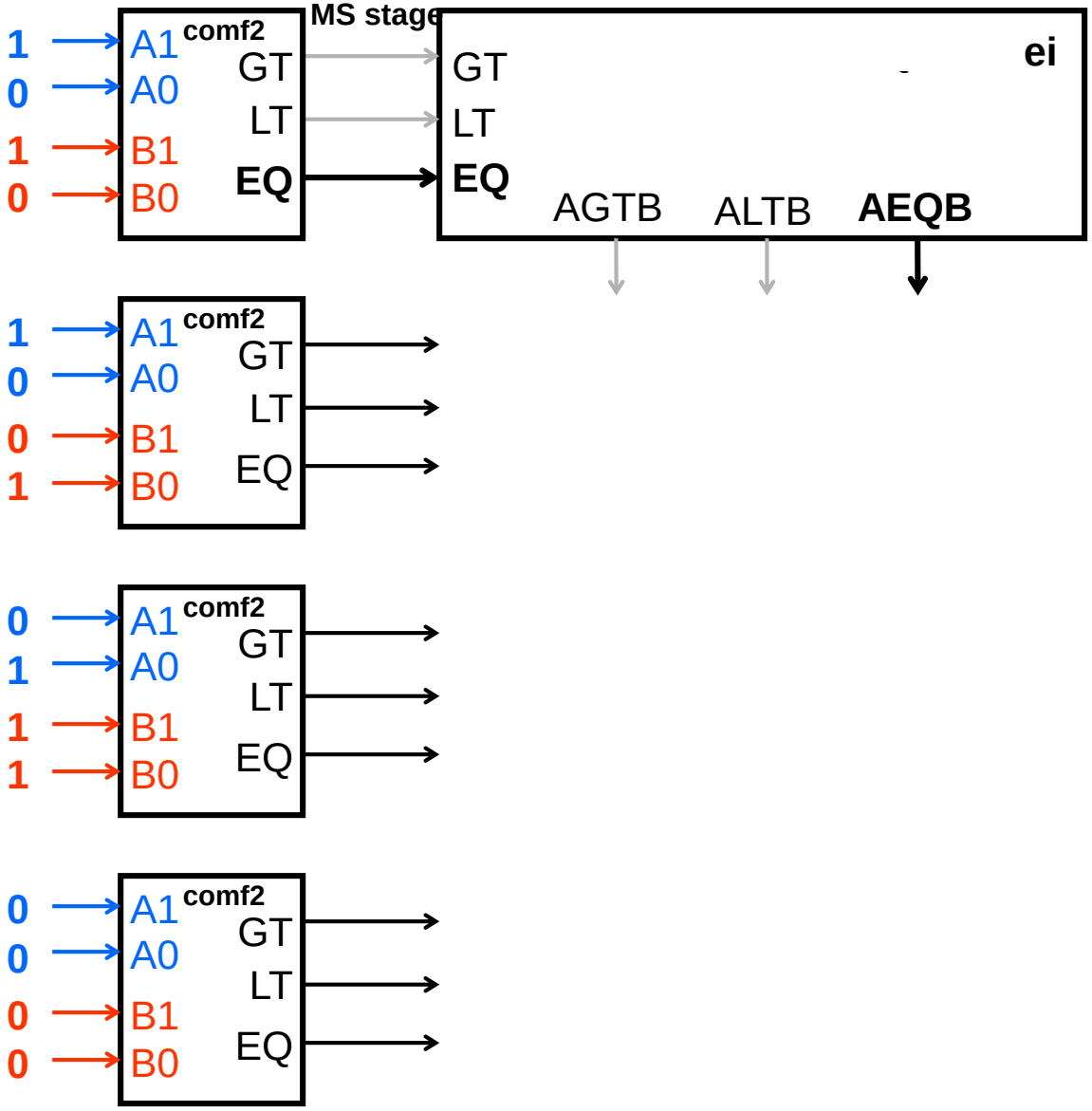
New component: ei (**E**xpansion **I**nterface)

Definition:

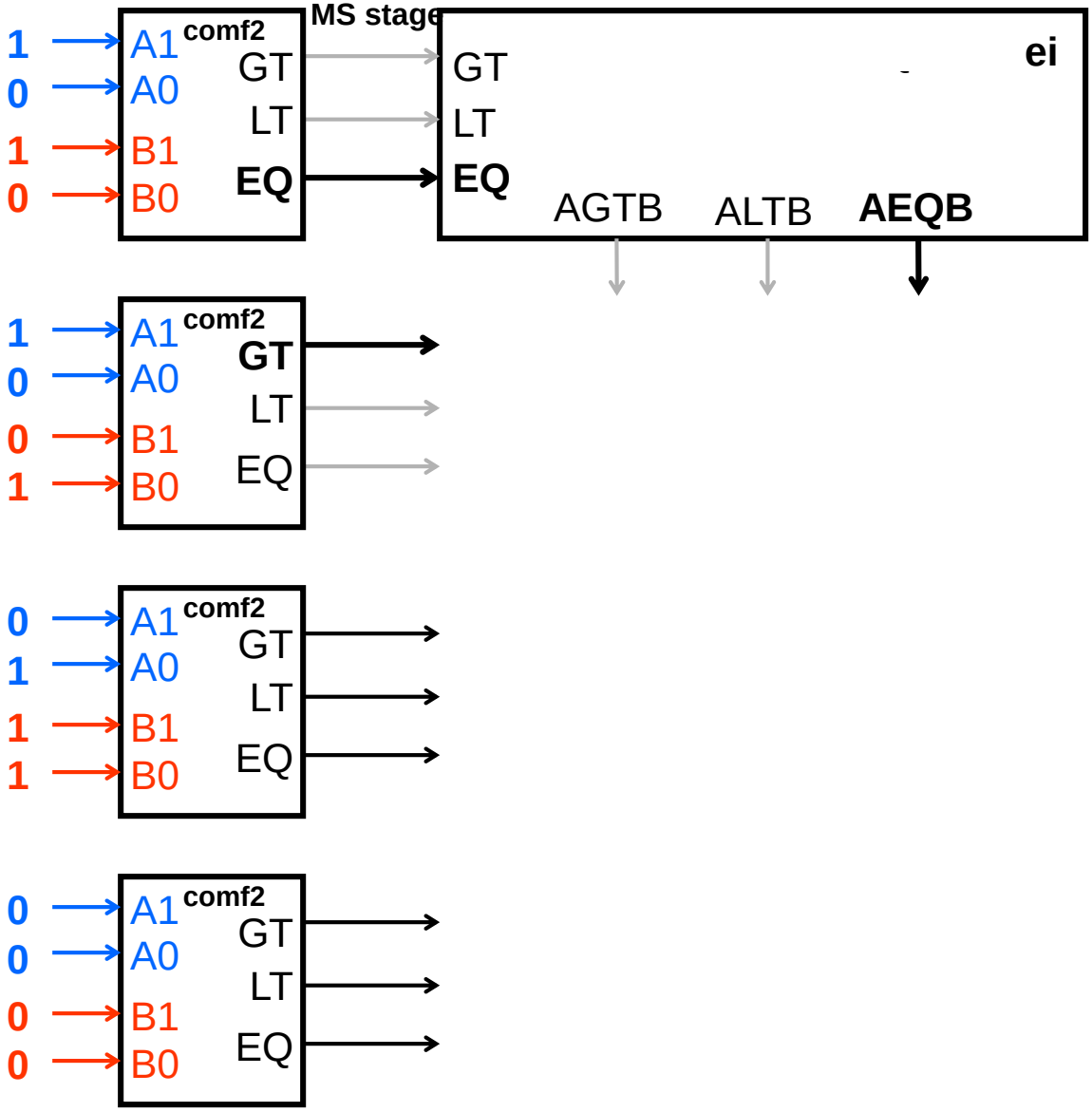
A **message** is a 3-bit Vector: A > B, A < B, A = B

ei receives 2 messages and generates 1 message:

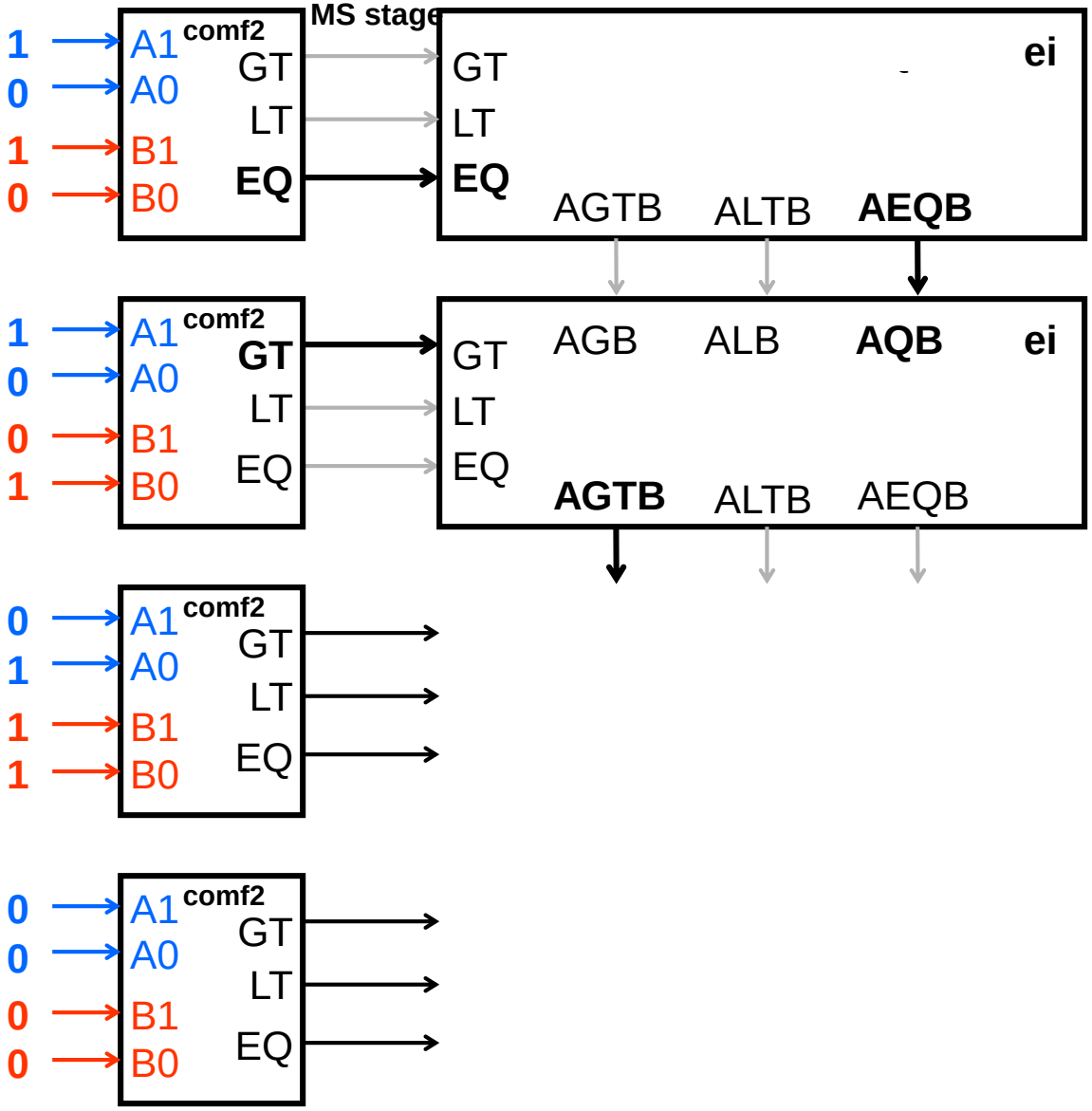


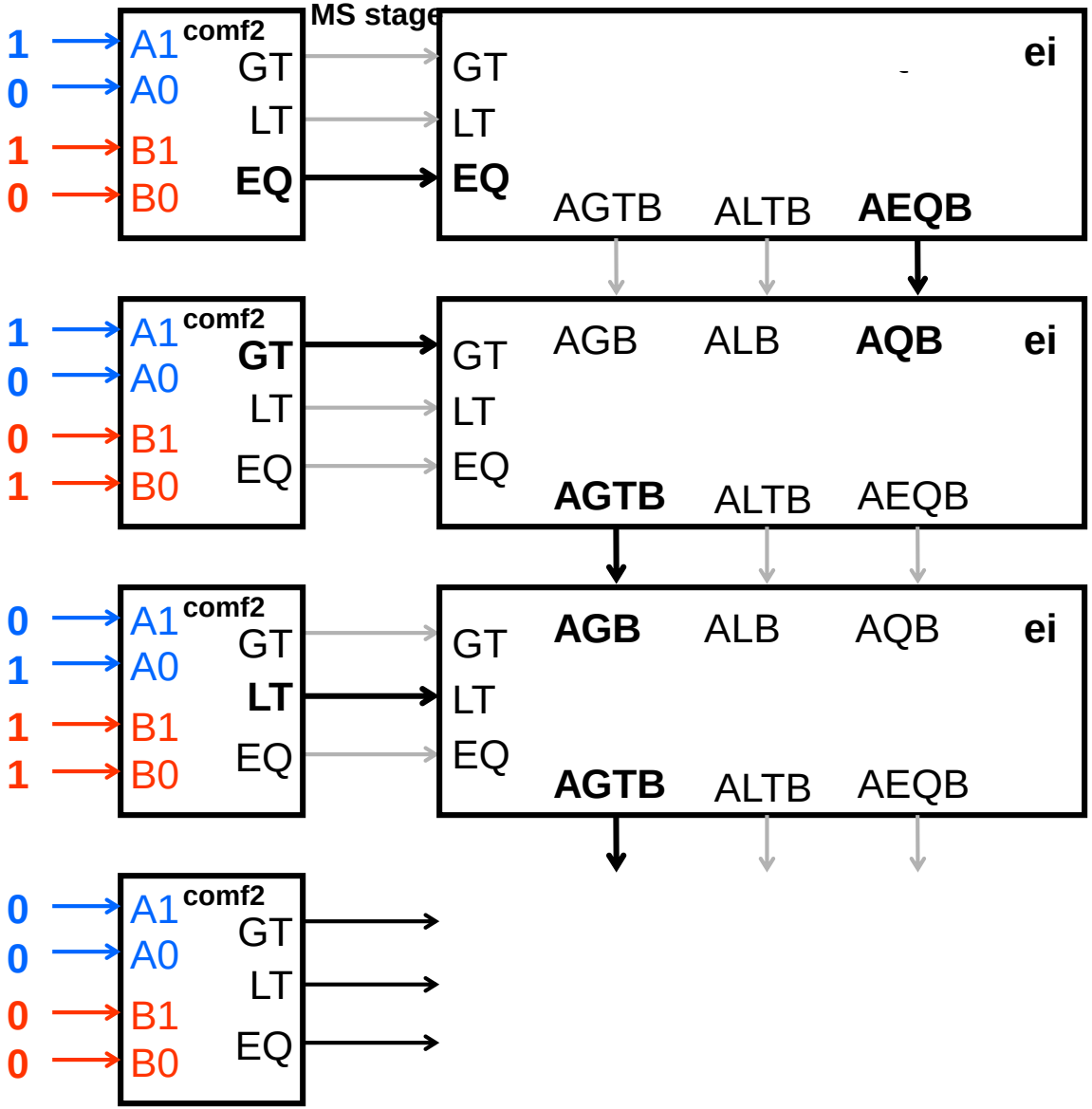

MS stage
comf2
A1
A0
B1
B0
GT
LT
EQ
1
0
1
0
ei
AGTB
ALTB
AEQB
comf2
A1
A0
B1
B0
GT
LT
EQ
1
0
0
1
comf2
A1
A0
B1
B0
GT
LT
EQ
0
1
1
1
comf2
A1
A0
B1
B0
GT
LT
EQ
0
0
0
0

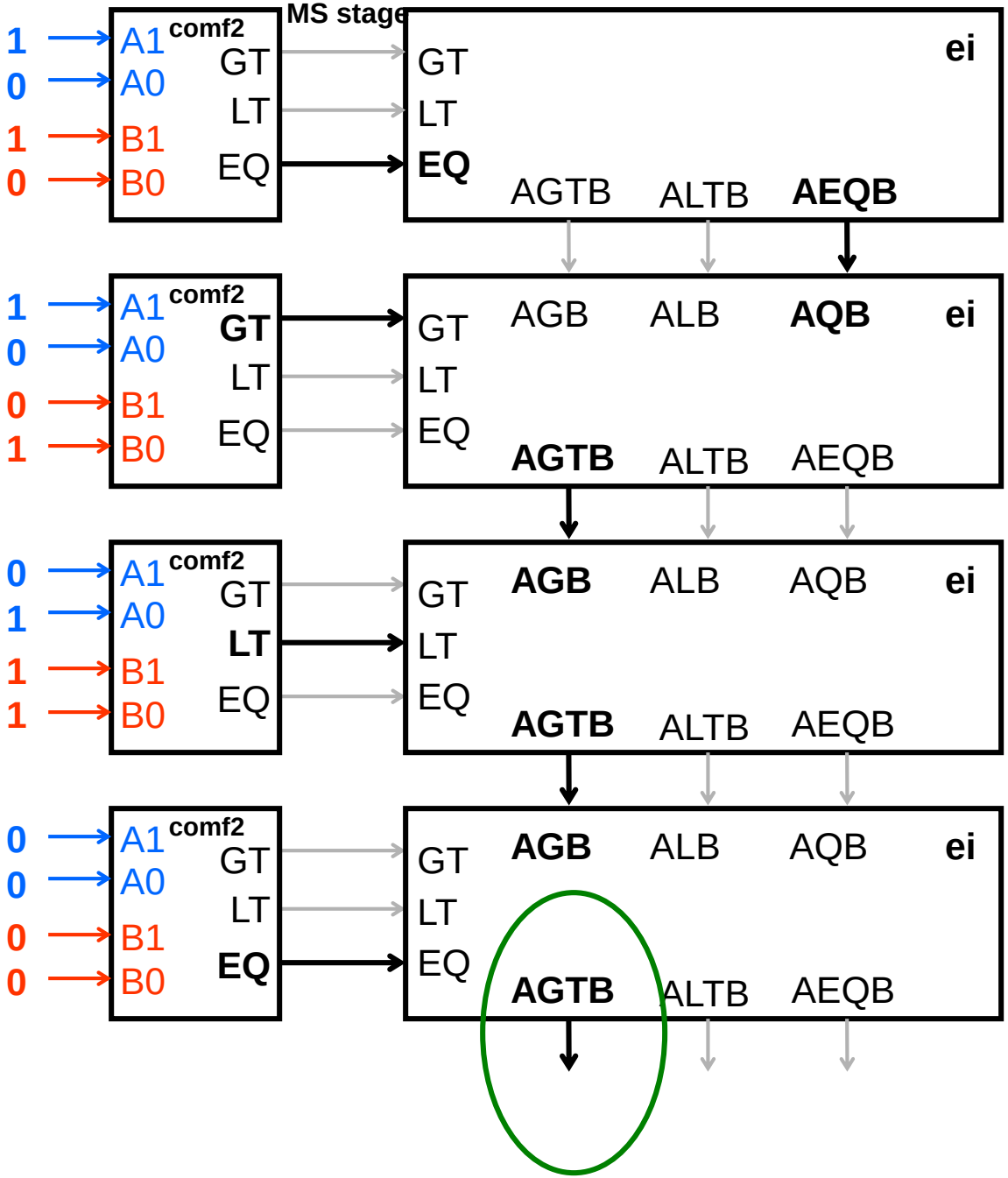
Listen to your comparator

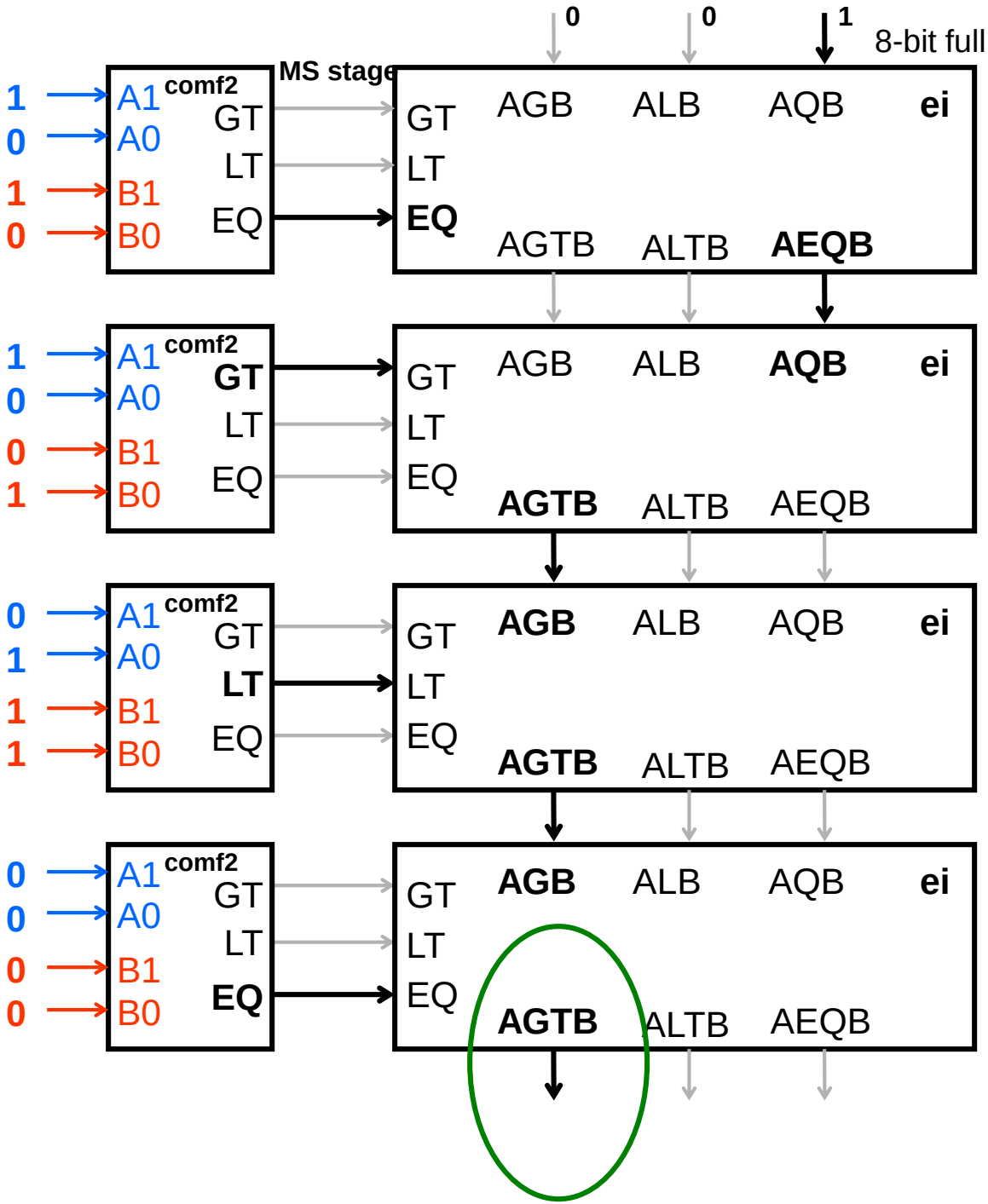


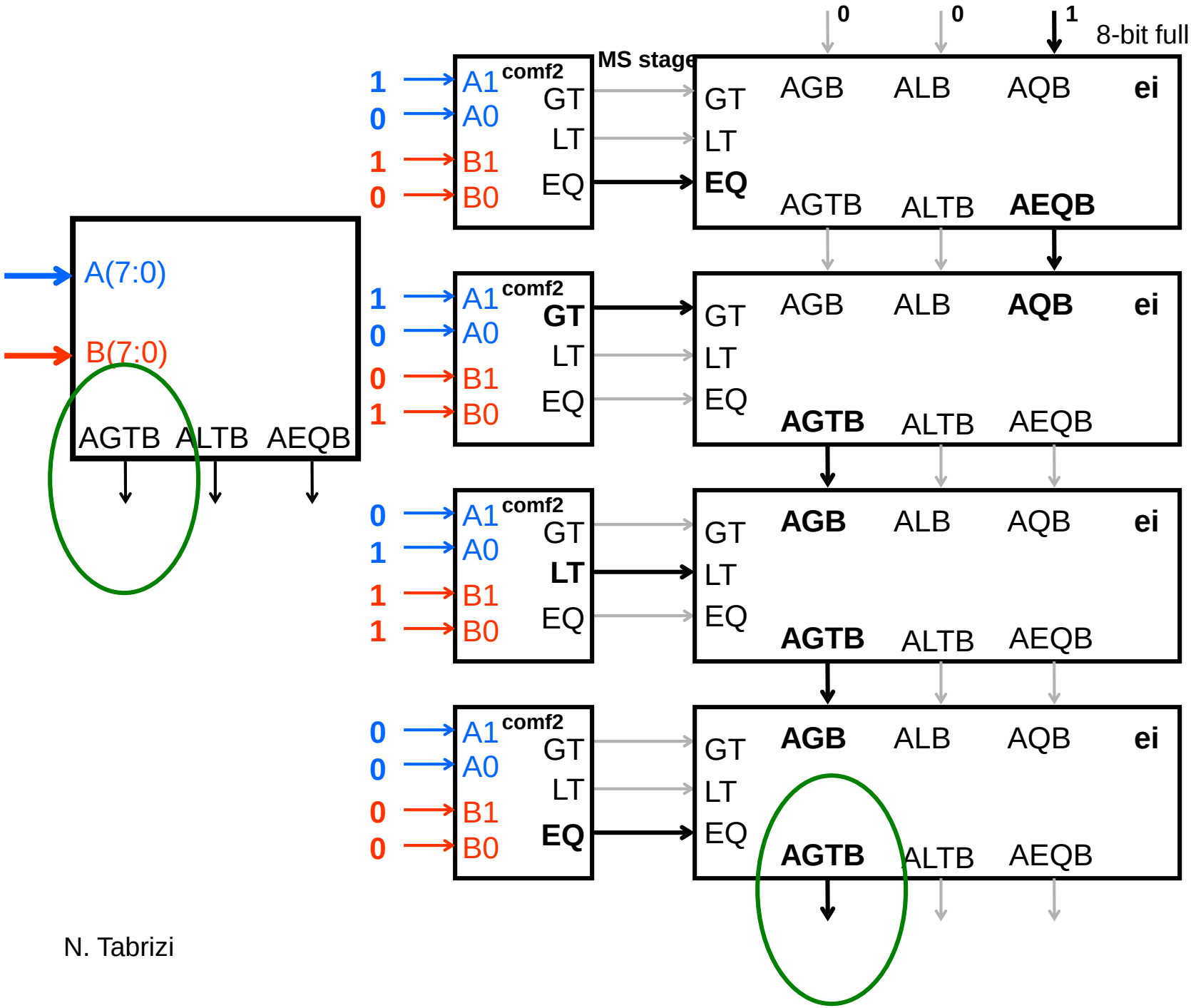
Listen to your comparator



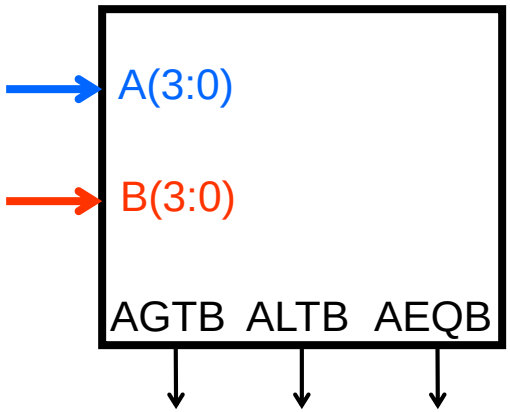

MS stage
comf2
A1
A0
B1
B0
GT
LT
EQ
1
0
1
0
ei
AGTB
ALTB
AEQB
AGB
ALB
AQB
AGTB
ALTB
AEQB
AGB
ALB
AQB
AGTB
ALTB
AEQB
0
0
0
0

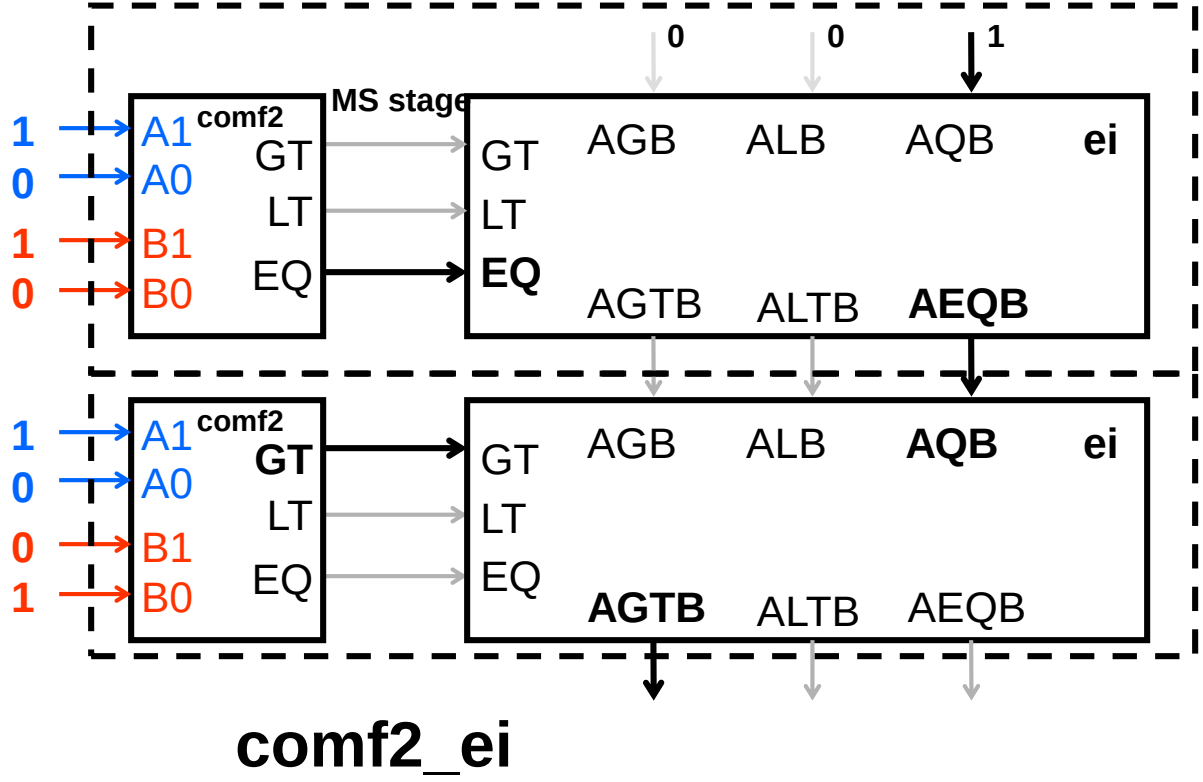

MS stage
comf2
A1
A0
B1
B0
GT
LT
EQ
ei
AGTB
ALTB
AEQB
AGB
ALB
AQB
1
0
1
0
1
0
0
1
0
1
1
1
0
0
0
0


8-bit full comparator
MS stage
comf2
A1
A0
B1
B0
GT
LT
EQ
AGB
ALB
AQB
ei
AGTB
ALTB
AEQB
0
1


8-bit full comparator
MS stage
comf2
A(7:0)
B(7:0)
AGTB ALTB AEQB
A1
A0
B1
B0
GT
LT
EQ
AGB ALB AQB ei
AGTB ALTB AEQB
0
0
1
1
0
1
0
1
0
1
1
1
0
0
0
0

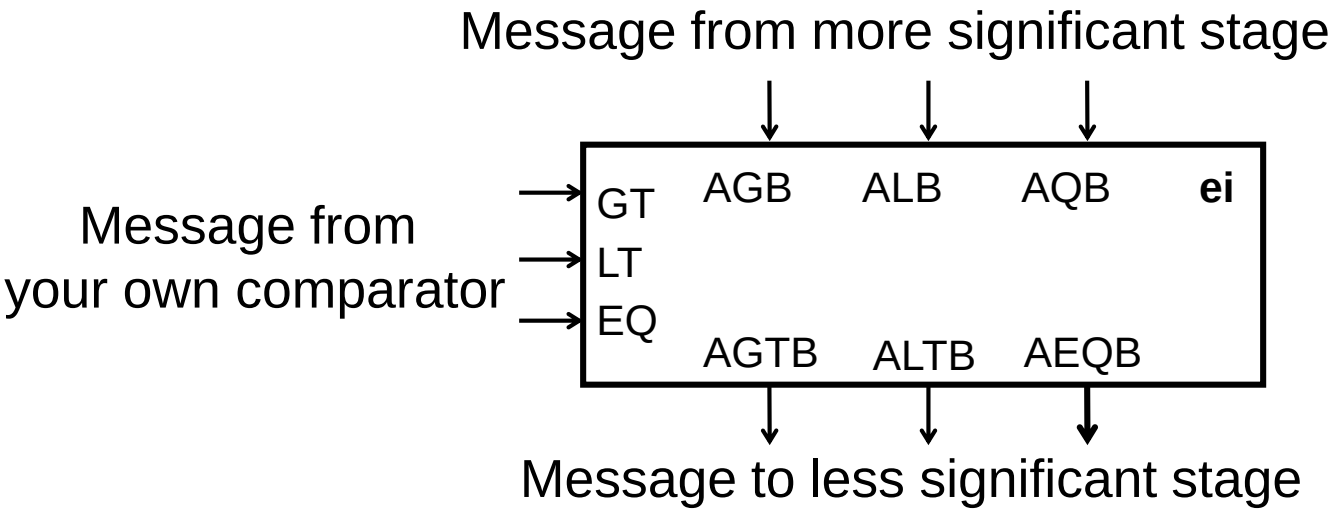
Lab-05





comf2_ei

Design of ei



Translate the **Rule** into 3 functions:

Rule: If the more significant ei says A equals B so far, listen to your own comf2; otherwise listen to the more significant ei.

End of Chapter 4, Part II