Computer Organization

Register Transfer Logic

Department of Computer Science
Missouri University of Science & Technology
hurson@mst.edu

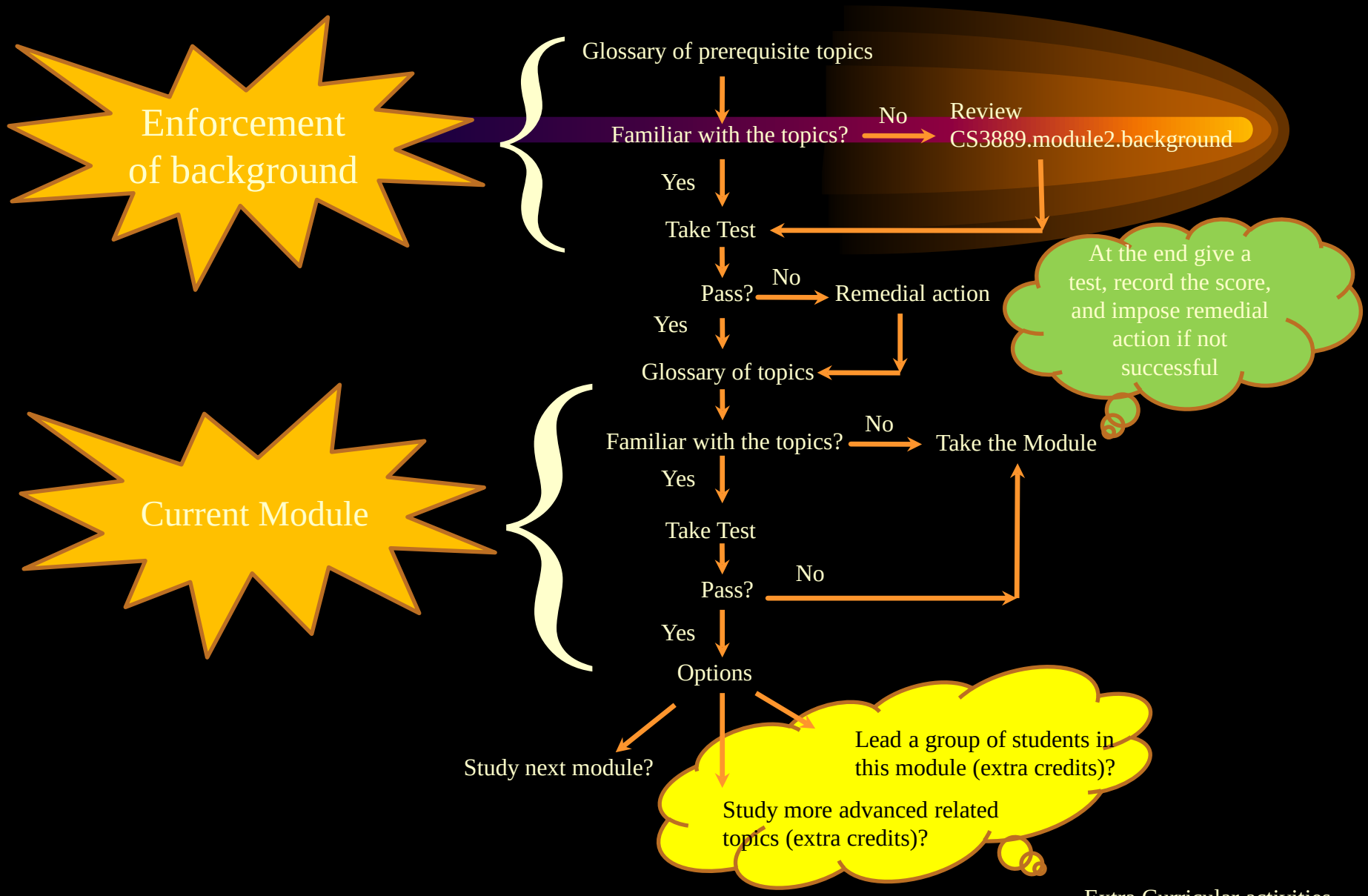
Computer Organization

Note, this unit will be covered in three lectures. In case you finish it earlier, then you have the following options:

1) Take the early test and start CS3889.module3
2) Study the supplement module (supplement CS3889.module2)
3) Act as a helper to help other students in studying CS3889.module2

Note, options 2 and 3 have extra credits as noted in course outline.

Computer Organization



Computer Organization

- **Register Transfer Language** (RTL)
  - Like any other language, the RTL is:
    - A set of symbols and rules to describe:
      - Elementary objects,
      - Syntax, and
      - Semantics of the system.
    - Composed of two parts:
      - Declaration Part
      - Execution Part

Computer Organization

- **Register Transfer Language** (RTL)
  - Declaration Part is used to describe the basic components (elementary objects):
    - A Set of registers and their functions.
    - The binary coded information in the registers.
    - The operations performed on the information stored in the registers.
    - The control functions that initiate the sequence of the operations.

Computer Organization

- **Register Transfer Language** (RTL)
  - Execution part is a collection of statements. Each statement is composed of two parts:
    - **Control Part (label):** is a Boolean function, which determines under what condition the corresponding operation (s) is (are) going to be executed.
    - **Operational Part:** determines the data manipulation actions (μ-operation).

T_1: $A \leftarrow (B)$ $\qquad$ T_2: $A \leftarrow (A)+(B)$

Control Part $\qquad$ Operational part

Computer Organization

- **Register Transfer Language:**
  - There is a one-to-one relationship between the components in RTL and our earlier view at the computer:
    - Set of registers ≡ memory system
    - μ-operations ≡ ALU
    - Control functions ≡ CU

Register Transfer Language:

- Register: It is a group of binary cells suitable for holding binary information. In RTL notation it is represented by Capital letters:

  MAR, A, B, IR, ...

- Bits in a register are numbered - either from left-to-right or right-to-left, starting either from 0 or 1.

Declare	Register	A(8), MBR(12), PC(16), ...
Declare	Sub-Register	PC(L)=PC(1-8), PC(H)=PC(9-16)

Computer Organization

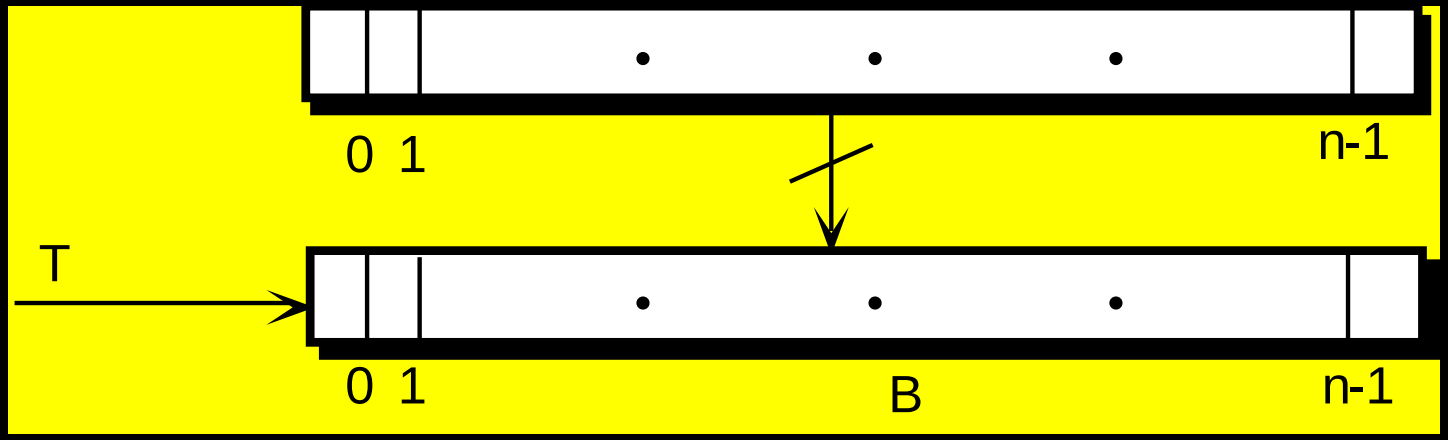
- **Register Transfer Language:**
  - Memory can be viewed as a 2-dimensional storage medium:

    Declare Memory M[0:5, 0:15]

Computer Organization

- **Register Transfer Language: Register**
  - Storage media (registers) can be classified based on various parameters:
  - **Access Time**
    - Fast access storage ("register")
      - General purpose register (R1-R16 in IBM)
      - Special purpose register (PC,IR, ...)
    - Slower access storage (main memory)
  - **Functionality**
    - Storage type (R_1-R_{16} in IBM)
    - Operational type (AC, PC)

- **Register Transfer Language**: Register
  - Graphically a register is represented by a rectangle:

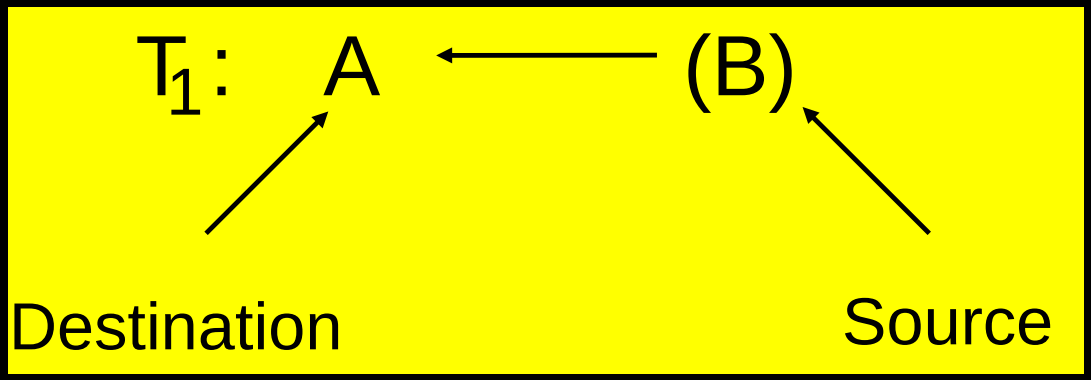


Register Transfer Language: Operations

- A μ-operation is an elementary operation which is executed in one clock pulse period.
- Four classes of μ -operations will be discussed:
  - Inter-register Transfer,
  - Arithmetic,
  - Logic, and
  - Shift

Computer Organization

- **Register Transfer Language: Inter-register Transfer**
  - It is this class of operations which allows data to be transferred from one storage unit to another.



Computer Organization

- **Register Transfer Language**: Inter-register Transfer
  - It can be divided into two groups:
    - Register Transfer and
    - Memory Transfer
  - Inter-register transfer operations can be performed in two fashions:
    - Serial: During each clock pulse one bit of source is transferred to the destination.
    - Parallel: During a clock pulse the whole contents of source is transferred to the destination.

Computer Organization

- **Register Transfer Language**: Inter-register Transfer
  - **Register Transfer:** is the class of operations that determines the data transfer between "registers".
  - **Memory Transfer:** is the class of operations that determines how data stored in memory can be accessed.
  - Note: i) Conventional main memory is random access memory. ii) Any access to main memory is through **M**emory **A**ddress **R**egister (MAR) and **M**emory **B**uffer **R**egister(MBR). Two major operations can be recognized: **READ, WRITE**.

Computer Organization

- **Register Transfer Language**: Memory Transfer
  - Read

    R_1: MAR ← "address"

    R_2: MBR ← (M[MAR])
  - Write

    W_1: MAR ← "address"

    W_2: MBR ← "data"

    W_3: M[MAR] ← (MBR)

Computer Organization

- **Question**
  - What is the minimum number of operations which allows all possible arithmetic operations?
  - What are they?
  - What is the minimum number of operations which allows all possible logic operations?
  - What are they?

Computer Organization

- **Register Transfer Language**: Arithmetic Operations
  - Two basic operations will be studied: ADD and Complement.

T_1: $F \leftarrow (A) + (B)$

T_1: $A \leftarrow (\overline{A})$

T_1: $A \leftarrow (\tilde{A})$ Note: $(\tilde{A}) = (\overline{A}) + 1$

- **Register Transfer Language**: Arithmetic Operations
  - Subtraction, Multiplication, and Division can be performed directly; or,

T_1: $F \leftarrow (A) - (B)$

T_{11}: $F \leftarrow (\overline{B}) + 1$

T_{12}: $F \leftarrow (A) + (F)$

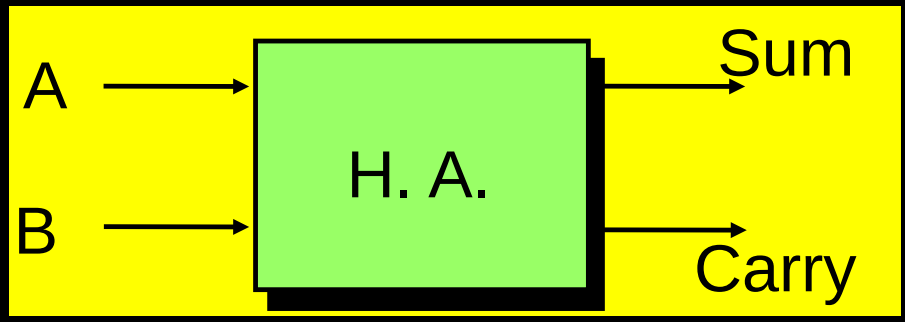
T_1: $F \leftarrow (A) * (B)$ repeated additions

T_1: $F \leftarrow (A) / (B)$ repeated subtractions

- **Register Transfer Language**: Addition
  - Basic building blocks:
    - Half adder (H. A.)

      Sum = A ⊕ B

      Carry = A Λ B

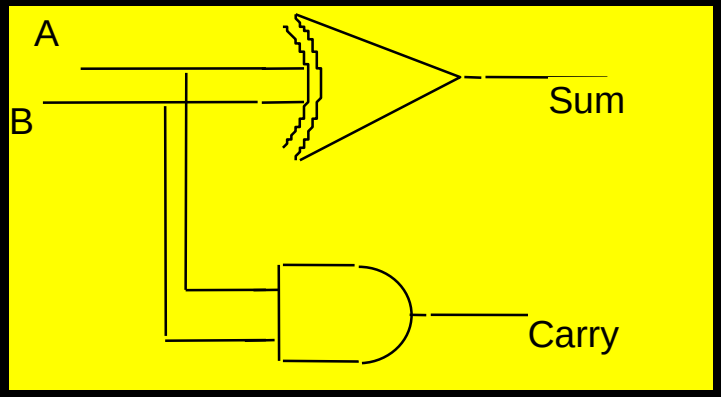


Register Transfer Language: Half Adder

Truth Table

A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Realization

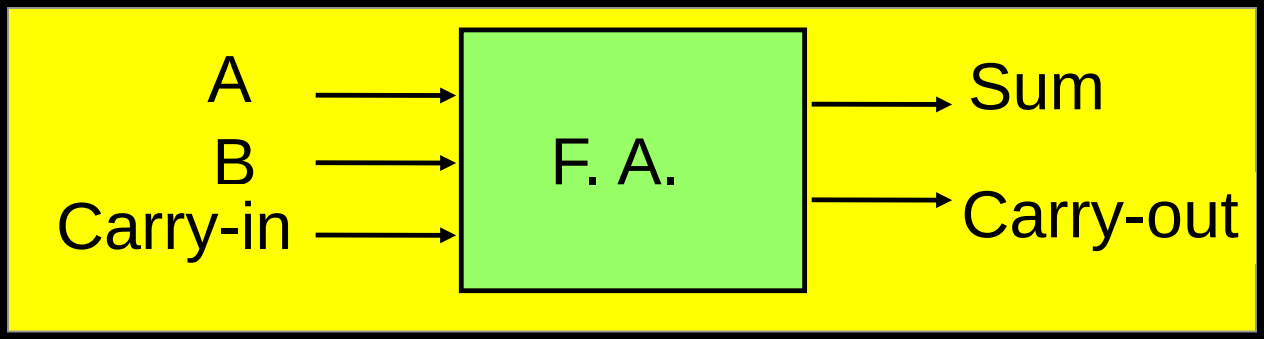


Timing: If Δt is the delay time of a basic logic gate (OR, AND), then the delay of Half Adder = 2 Δt.

- **Register Transfer Language: Addition**
  - Basic building blocks:
    - Full Adder (F.A.)

Sum = A ⊕ B ⊕ Carry-in

Carry-out = A Λ B + (A ⊕ B) Λ Carry-in

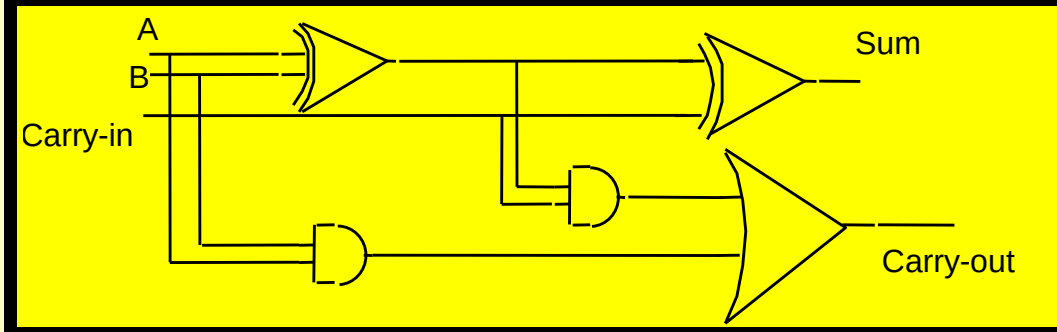


Register Transfer Language: Full Adder

Truth Table

A	B	Carry-in	Sum	Carry-out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

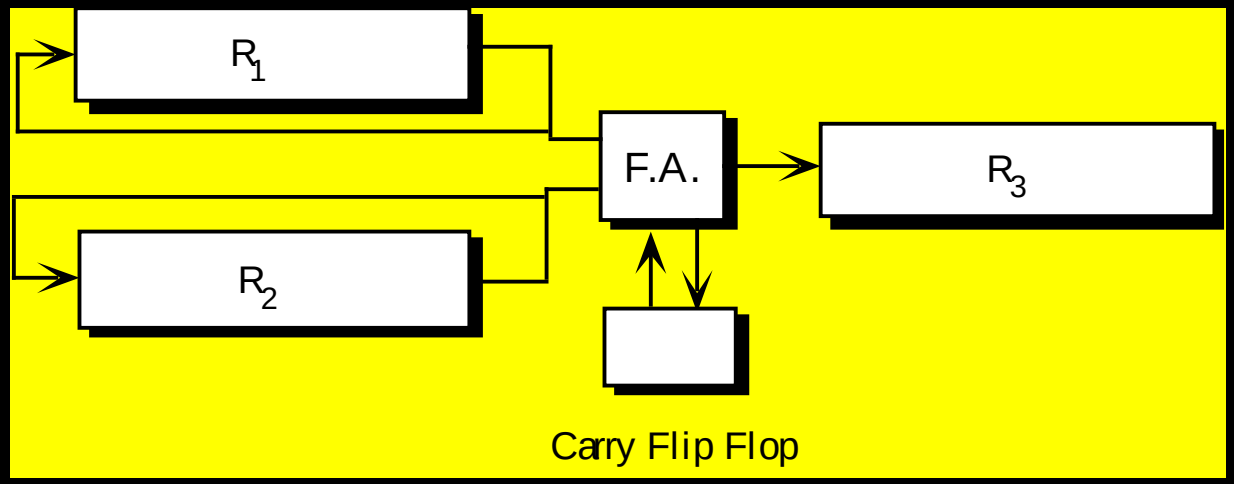
Realization



- Timing: If Δt is the delay time of a basic logic gate (OR, AND), then the delay of Full Adder = $4\Delta t$.

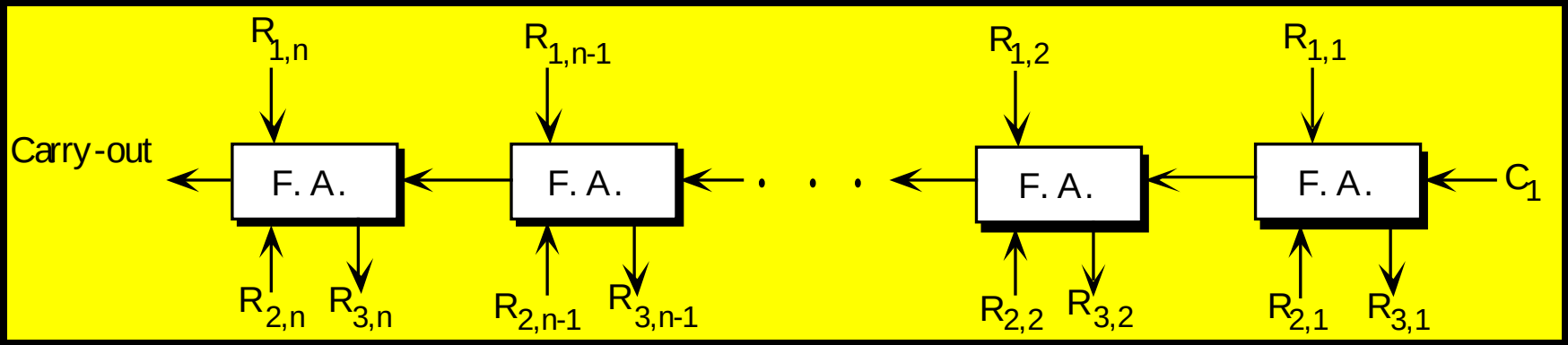
- **Register Transfer Language**: Serial Adder
  - R_1 and R_2 are circular shift registers.
  - R_3 is a shift register.

$$R_3 \leftarrow (R_1) + (R_2)$$



- **Register Transfer Language**: Parallel Adder
  - A cascade of *n* full adders.

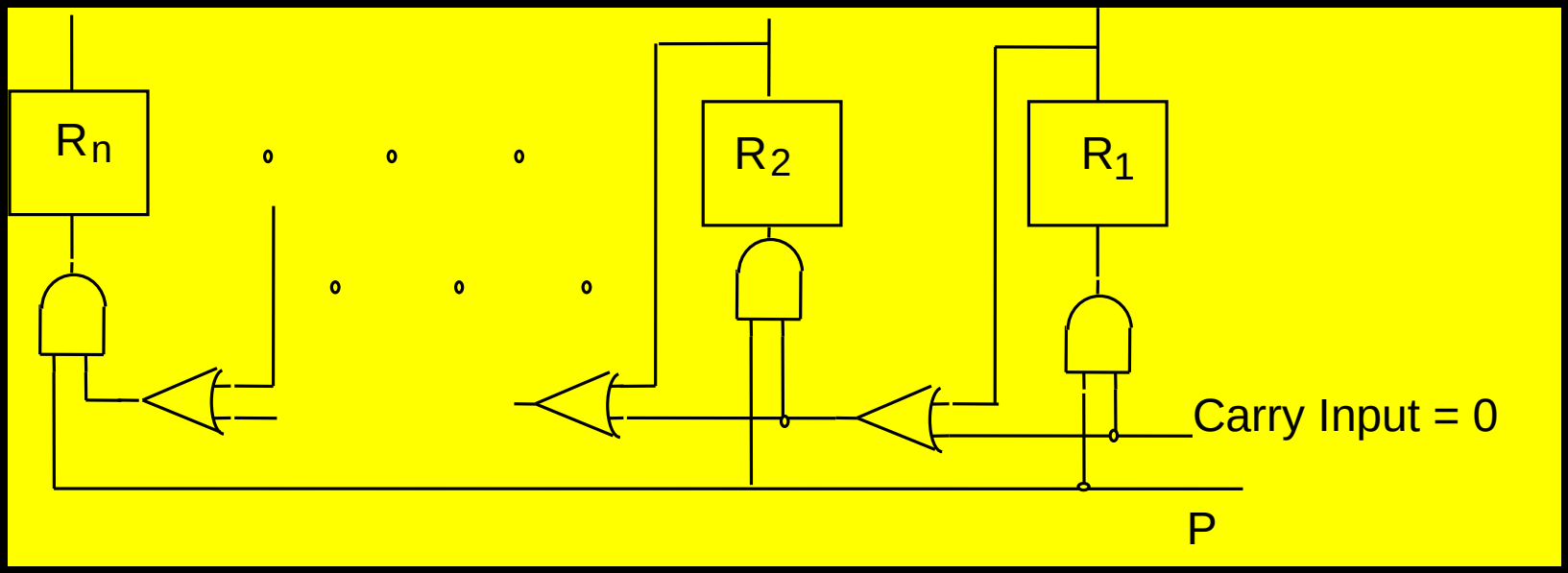
$R_3 \leftarrow (R_1) + (R_2)$



- **Question**
  - Parallel operations are faster and more expensive than serial operations.
  - What are the Delay times of serial and parallel adders?
  - How to improve the performance of parallel adder?

- **Register Transfer Language**: 2^s Complement
  - A simple combinational circuit can be developed to perform 2^s complement operation.
  - Assuming T type Flip Flops, when activation signal (P) is high the following circuit performs 2^s complement operation.

- **Register Transfer Language**: 2^s Complement



- **Question**
  - How is the value of *P* determined?
  - Modify the 2^s complement circuit for 1^s complement operation.

- **Register Transfer Language**: Logic Operations
  - With two binary bits one can have up to sixteen different logic operations.
  - One needs to have either (AND, NOT) or (OR, NOT) combination in order to implement all possible logic operations.

- **Register Transfer Language**: Logic Operations
  - Logic operations can be used in order to implement other functions.
  - Logic operations are useful in string and pattern matching operations.

$$T_1: \quad A \leftarrow (A) \wedge (B)$$

$$T_2: \quad A \leftarrow (A) \oplus (B)$$

- **Register Transfer Language**: Logic Operations
  - **selective set —** Set the bits in register A where there are corresponding 1s in register B.

    A = 1001

    B = 1011 $\quad\quad$ $A \leftarrow A \cup B = 1011$
  - **selective complement —** Complement bits in A where there are corresponding 1s in B.

    A = 1001

    B = 1010 $\quad\quad$ $A \leftarrow A \oplus B = 0011$

- **Register Transfer Language**: Logic Operations
  - **Transfer operation** can be performed by logic operations.

    A = 1001 $\quad$ B $\leftarrow$ 0000

    B = 1100 $\quad$ B $\leftarrow$ A $\cup$ B = 1001

  - Contents of two registers can be compared against each other by logic operation.

    A = 1010 $\quad$ A XOR B = 0000

    B = 1010 $\quad$ A XNOR B = 1111

- **Register Transfer Language**: Shift Operations
  - Shift operations can be used to perform transfer as well as arithmetic operations.
  - To accomplish such an operation, one has to specify two parameters:
    - Direction - i.e., Left or Right
    - Number of positions

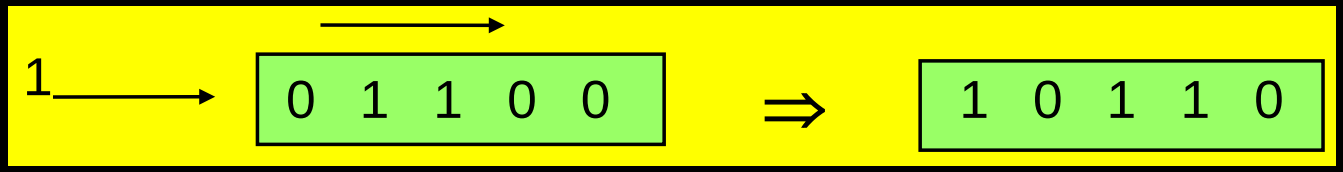
- **Register Transfer Language**: Shift Operations
  - Four types of shift operations can be distinguished:
    - Serial Shift
    - Circular Shift
    - Logical Shift
    - Arithmetic Shift

Computer Organization

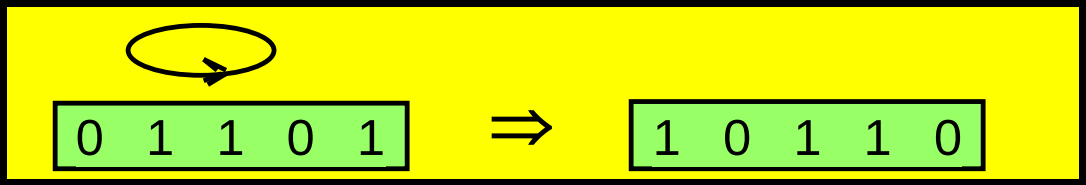
Questions

- Parallel is faster than Serial?
- Parallel is more expensive than Serial?
- How to transfer data among "registers"?
  - Direct Transfer?
  - Busing Technique?
- Compare and contrast **direct transfer** against **busing technique.**
- What is the difference between RAM, DAM, and SAM?
- What determines the length of MAR and MBR?
- How are the lengths of MAR and MBR determined?

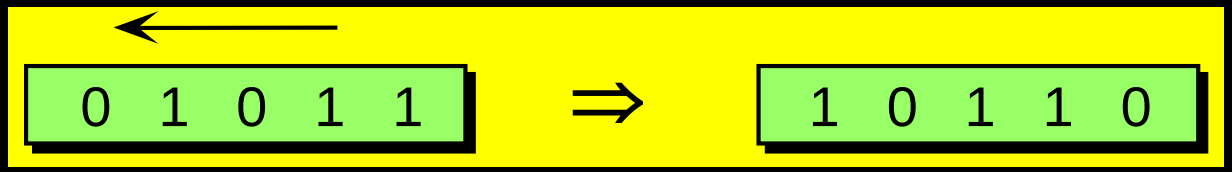
- **Register Transfer Language**: Shift Operations
  - In **serial shift** the extreme bit, left-most or right-most, will receive its contents from outside:



  - In **circular shift** one extreme bit assumes the contents of the other:



- **Register Transfer Language**: Shift Operations
  - In **logical shift** the extreme bit, depending on the direction, always assumes the value 0:



- **Register Transfer Language**: Shift Operations
  - In **arithmetic shift** the contents of the register is assumed to be a signed number.
  - In arithmetic shift:
    - Sign should be preserved
    - Shift right should divide the contents by 2
    - Shift left should multiply the contents by 2
  - To satisfy the above conditions one has to develop different routines based on the representation of the number - i.e., signed magnitude, signed 1^s complement, and signed 2^s complement.

Register Transfer Language: Arithmetic Shift

Data representation	Sign bit	Procedure
Positive	Unchanged	Added bit is zero
Negative Signed Mag.	Unchanged	Added bit is zero
Negative 1^s Complement	Unchanged	Added bit is one
Negative 2^s Complement	Unchanged	Added bit is zero (Shift left)
Negative 2^s Complement	Unchanged	Added bit is one (Shift right)

- **Register Transfer Language**: Arithmetic Shift
  - Signed magnitude number
    - Perform logical shift on the magnitude part.
    - Do not change the sign bit.
  - 1^s complement and 2^s complement number:
    - The whole number should be operated on.
    - For 1^s complement duplicate the sign bit for extreme bit.
    - For 2^s complement shift right duplicate the sign bit.
    - For 2^s complement shift left perform the logic shift.

- **Register Transfer Language**: Arithmetic Shift (Example)
  - Assume

A = 0 0 1 0 1 B = 1 1 0 1 0

  - Signed magnitude
    - AShift A Left

0 1 0 1 0

AShift B Right

1 0 1 0 1

Computer Organization

- **Register Transfer Language**: Arithmetic Shift (Example)
  - 1s Complement
    - AShift A Right
    - AShift B Right
    - AShift B Left

0 0 0 1 0	1 1 1 0 1	1 0 1 0 1

- **Register Transfer Language**: Arithmetic Shift (Example)
  - 2^s complement
    - AShift B Right

      1 1 1 0 1

    - AShift B Left

      1 0 1 0 0

  - Assume C = 1 0 1 1 0 is in 2^s complement.

    AShift C Left => 0 1 1 0 0

    Signed has not been preserved! Why?

Register Transfer Language

- **Full adder**: A basic building block for the design of an adder. It can be designed as a cascade of two half adders.
- **Half adder**: A basic building block — a combinational logic — for the design of an adder. It accepts two binary bits and generates the sum and carry.
- **micro operation**: A basic operation that can be executed during a single clock pulse.

Register Transfer Language

- **Parallel adder**: A way to add two binary numbers — addition will be initiated simultaneously on all of the operand bit pairs.
- **Serial adder**: A way to perform addition. During each clock pulse one pair of the operand bits will participate in the operation.
- **Overflow**: Inability to store a number in its entirety in a register.
- **Interregister transfer operation**: A class of micro operations that allows data transfer between a source and destination registers.

- **Questions**
  - **True or false**: in general, read from memory is faster than writing into the memory!
  - **Assume** that Δt is the delay of a simple logic gate — i.e., an AND or OR gate, what is the execution time of an n-bit parallel adder?
  - **True or false**: parallel operation is faster than serial operation (justify)!

Computer Organization

- **Questions**
  - **True or false**: one needs more hardware support for parallel operations than serial operations!
  - **Write** a simple software program to perform multiplication as a sequence of additions.
  - **Write** a simple software program to perform division as a sequence of subtractions.
  - **How** can we use the same combinational logic for the 2^s complement operation to perform the 1^s complement operation?

- **Questions**
  - **Equality** between two binary bit patterns can be determined by using exclusive OR operation. Do you know any other logic operation to carry out the same operation?
  - **If** Δt is the delay of a simple logic gate — i.e., an AND or OR gate, then what is the execution time of a parallel AND operation, and what is the execution time of a serial AND operation?