İstanbul
Bilgi Üniversitesi

CMPE321 - Computer Architecture

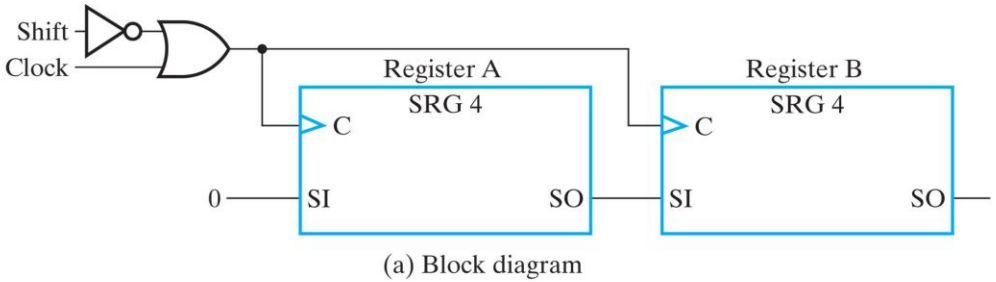
Lecture 4
Serial Transfer and Microoperations

Hakan Ayral, PhD.

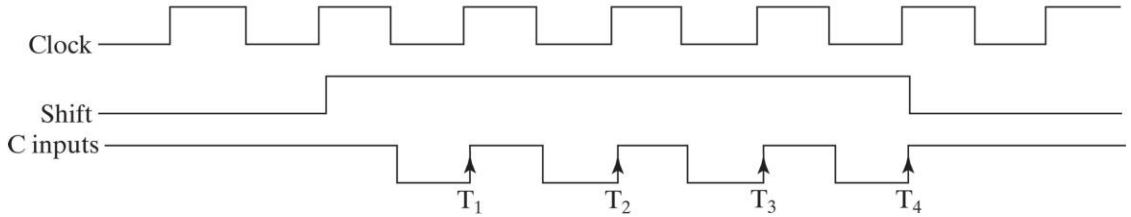
Serial vs Parallel

- A digital system is said to operate in a serial mode when information in the system is transferred or manipulated one bit at a time.
- Information is transferred one bit at a time by shifting the bits out of one register and into a second register.
- This transfer method is in contrast to parallel transfer, in which all the bits of the register are transferred at the same time.

Serial Transfer



(a) Block diagram



(b) Timing diagram

- The serial transfer of information from register *A* to register *B* is done with shift registers, as shown in the block diagram of Figure 21(a).
- The serial output of register *A* is connected to the serial input of register *B*.
- The serial input of register *A* receives 0s while its data is transferred to register *B*.
- It is also possible for register *A* to receive other binary information, or if we want to maintain the data in register *A*, we can connect its serial output to its serial input so that the information is circulated back into the register.
- The initial content of register *B* is shifted out through its serial output and is lost unless it is transferred back into register *A*, to a third shift register, or to other storage.
- The shift control input *Shift* determines when and how many times the registers are shifted. The registers using *Shift* are controlled by means of the logic from Figure (a), which allows the clock pulses to pass to the shift register clock inputs only when *Shift* has the value logic 1.

Serial Transfer

- In previous Figure, each shift register has four stages.
- The logic that supervises the transfer must be designed to enable the shift registers, through the *Shift* signal, for a fixed time of four clock pulses.
- Shift register enabling is shown in the timing diagram for the clock gating logic in Figure 21(b).
- Four pulses find *Shift* in the active state, so that the output of the logic connected to the clock inputs of the registers produces four pulses: *T*1, *T*2, *T*3, and *T*4.
- Each positive transition of these pulses causes a shift in both registers.
- After the fourth pulse, *Shift* changes back to 0 and the shift registers are disabled.
- We note again that, for positive-edge triggering, the pulses on the clock inputs are 0, and the inactive level when no pulses are present is a 1 rather than a 0.

Serial Transfer

- Suppose that content of register *A* is 1011, register *B* is 0010, and *SI* of register *A* is 0. Serial transfer from *A* to *B* occurs in four steps.
- With the first pulse *T*1, the rightmost bit of *A* is shifted into the leftmost bit of *B*, the leftmost bit of *A* receives a 0 from the serial input, and at the same time, all other bits of *A* and *B* are shifted one position to the right.
- The next three pulses perform identical operations, shifting the bits of *A* into *B* one at a time while transferring 0s to *A*.
- After the fourth shift, the logic supervising the transfer changes the *Shift* signal to 0 and the shifts stop.
- Register *B* contains 1011, which is the previous value of *A*. Register *A* contains all 0s.

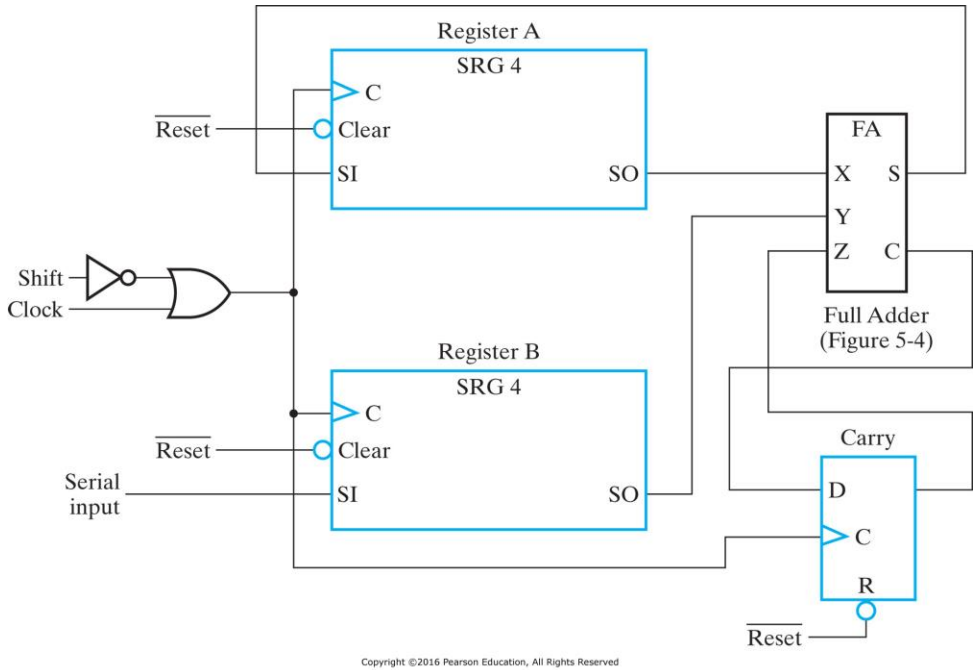
TABLE 6-14
Example of Serial Transfer

Timing Pulse	Shift Register A				Shift Register B			
Initial value	1	0	1	1	0	0	1	0
After T_1	0	1	0	1	1	0	0	1
After T_2	0	0	1	0	1	1	0	0
After T_3	0	0	0	1	0	1	1	0
After T_4	0	0	0	0	1	0	1	1

- In the parallel mode, information is available from all bits of a register, and all bits can be transferred simultaneously during one clock pulse.
- In the serial mode, the registers have a single serial input and a single serial output, and information is transferred one bit at a time.

Serial Addition

- Operations in digital computers are usually done in parallel because of the faster speed attainable.
- Serial operations are slower, but have the advantage of requiring less hardware.
- To demonstrate the serial mode of operation, we will show the operation of a serial adder.
- Also, we compare the serial adder to the parallel counterpart to illustrate the time–space trade-off in design.
- The two binary numbers to be added serially are stored in two shift registers.
- Bits are added, one pair at a time, through a single full-adder (FA) circuit, as shown in Figure.
- The carry out of the full adder is transferred into a *D* flipflop.



- The output of this carry flip-flop is then used as the carry input for the next pair of significant bits.
- The sum bit on the *S* output of the full adder could be transferred into a third shift register, but we have chosen to transfer the sum bits into register *A* as the contents of the register are shifted out.
- The serial input of register *B* can receive a new binary number as its contents are shifted out during the addition.

Serial Addition

- The operation of the serial adder is as follows: Register A holds the first operand, register B holds the second operand, and the carry flip-flop has been reset to 0.
- The serial outputs of A and B provide a pair of significant bits for the full adder at X and Y.
- The output of the carry flip-flop provides the carry input at Z.
- When Shift is set to 1, the OR gate enables the clock for both registers and the flip-flop.
- Each clock pulse shifts both registers once to the right, transfers the sum bit from *S* into the leftmost flip-flop of *A*, and transfers the carry output into the carry flip-flop.
- Shift control logic enables the registers for as many clock pulses as there are bits in the registers (four pulses in this example).
- For each pulse, a new sum bit is transferred to *A*, a new carry is transferred to the flip-flop, and both registers are shifted once to the right.
- This process continues until the shift control logic changes *Shift* to 0.
- Thus, the addition is accomplished by passing each pair of bits and the previous carry through a single full-adder circuit and transferring the sum, one bit at a time, back into register *A*.

Serial Addition

- Initially, we can reset register *A,* register *B*, and the *Carry* flip-flop to 0.
- Then we shift the first number into *B*.
- Next, the first number from *B* is added to the 0 in *A*.
- While *B* is being shifted through the full adder, we can transfer a second number to it through its serial input.
- The second number can be added to the contents of register *A* at the same time that a third number is transferred serially into register *B*.
- Serial addition may be repeated to form the addition of two, three, or more numbers, with their sum accumulated in register *A*.

Serial Addition

- A comparison of the serial adder with the parallel adder provides an example of space–time trade-off.
- The parallel adder has n full adders for n-bit operands, whereas the serial adder requires only one full adder.
- Excluding the registers from both, the parallel adder is a combinational circuit, whereas the serial adder is a sequential circuit because it includes the carry flip-flop.
- The serial circuit also takes n clock cycles to complete an addition.
- Identical circuits, such as the n full adders in the parallel adder, connected together in a chain constitute an example of an iterative logic array.
- If the values on the carries between the full adders are regarded as state variables, then the states from the least significant end to the most significant end are the same as the states appearing in sequence on the flip-flop output in the serial adder.
- Note that in the iterative logic array the states appear in space, but in the sequential circuit the states appear in time.
- By converting from one of these implementations to the other, one can make a space–time trade-off. The parallel adder in space is n times larger than the serial adder (ignoring the area of the carry flip-flop), but it is n times faster.
- The serial adder, although it is n times slower, is n times smaller in space.
- This gives the designer a significant choice in emphasizing speed or area, where more area translates into more cost.

Control of Register Transfers

- Previously, we divided a digital system into two major components, a **datapath** and a **control unit**.
- Likewise, the binary information stored in a digital computer can be classified as either data or control information.
- Data is manipulated in a datapath by using **microoperations** implemented with **register transfers**.
- These operations are implemented with adder–subtractors, shifters, registers, multiplexers, and buses.
- The control unit provides signals that activate the various microoperations within the datapath to perform the specified processing tasks.
- The control unit also determines the sequence in which the various actions are performed.
- This separation of a system into two components and separation of the tasks performed carries over to the design process.
- The datapath and control unit are usually designed separately, but in close coordination with each other.

Control of Register Transfers

- Generally, the timing of all registers in a synchronous digital system is controlled by a **master clock** generator.
- The clock pulses are applied to all flip-flops and registers in the system, including those in the control unit.
- To prevent clock pulses from changing the state of all registers on every clock cycle, some registers have a load control signal that enables and disables the loading of new data into the register.
- The binary variables that control the selection inputs of multiplexers, buses, and processing logic and the load control inputs of registers are generated by the **control unit**.
- The **control unit** that generates the signals for sequencing the microoperations **is a sequential circuit** with states that dictate the control signals for the system.
- At any given time, the state of the sequential circuit activates a prescribed set of microoperations.
- Using status conditions and control inputs, the sequential control unit determines the next state.
- The digital circuit that acts as the control unit provides a sequence of signals for activating the microoperations and also determines its own next state.

Programmable and Non-programmable system

- Based on the overall system design, there are two distinct types of control units used in digital systems, one for a **programmable system** and the other for a **nonprogrammable system**.
  - In a ***programmable system,*** a portion of the input to the processor consists of a sequence of ***instructions***.
  - Each instruction specifies the operation that the system is to perform, which operands to use, where to place the results of the operation, and, in some cases, which instruction to execute next.
  - For programmable systems, the instructions are usually stored in memory, either in RAM or in ROM.
  - To execute the instructions in sequence, it is necessary to provide the memory address of the instruction to be executed.
  - This address comes from a register called the ***program counter*** (***PC***).
  - As the name implies, the *PC* has logic that permits it to count.
  - In addition, in order to change the sequence of operations using decisions based on status information from the datapath, the *PC* needs parallel load capability.
  - So, in the case of a programmable system, the control unit contains a *PC* and associated decision logic, as well as the necessary logic to interpret the instruction.
  - ***Executing*** an instruction means activating the necessary sequence of microoperations in the datapath required to perform the operation specified by the instruction.

Programmable and Non-programmable system

- For a ***nonprogrammable system***, the control unit is not responsible for obtaining instructions from memory, nor is it responsible for sequencing the execution of those instructions.
- There is no *PC* or similar register in such a system.
- Instead, the control unit determines the operations to be performed and the sequence of those operations, based on its inputs and the status bits from the datapath.
- In this section we focus on nonprogrammable system design, which illustrates the use of **state machine diagrams** for control unit design.