Register Transfer Language and Micro-Operations

By
Dr. L.Ranathunga

Lecture 5

Digital System Design

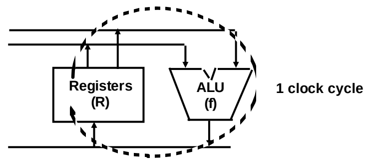
- Uses modular approach
- Modules are constructed from
  - Registers
  - Decoders
  - Arithmetic elements
  - Control logics
- Various modules are interconnected with common data path and control paths to form a digital system
- Register operations within the module could define the module

Micro-Operations

- Operations executed on data stored in registers are called micro-Operations
  - Shift data in registers
  - Count data in registers
  - Clear data in registers
  - Load data in registers
- Micro-operations on registers and control will define the internal h/w organization of a computer system
- To decide the operation within computer system as micro-operation, a descriptive language can use
  - HDL (Hardware Description Language)

MICROOPERATION ...

An elementary operation performed (during one clock pulse), on the information stored in one or more registers



$R \leftarrow f(R, R)$

f: shift, load, clear, increment, add, subtract, complement and, or, xor, ...

ORGANIZATION OF A DIGITAL SYSTEM

- **Definition of the (internal) organization of a computer**
  - **Set of registers and their functions**
  - **Microoperations set**
    - **Set of allowable microoperations provided by the organization of the computer**
  - **Control signals that initiate the sequence of microoperations (to perform the functions)**

REGISTER TRANSFER LEVEL

- Viewing a computer, or any digital system, in this way is called the register transfer level
- This is because we're focusing on
  - The system's registers
  - The data transformations in them, and
  - The data transfers between them.

REGISTER TRANSFER LANGUAGE

- Rather than specifying a digital system in words, a specific notation is used, *register transfer language*
- For any function of the computer, the register transfer language can be used to describe the (sequence of) microoperations
- Register transfer language
  - A symbolic language
  - A convenient tool for describing the internal organization of digital computers
  - Can also be used to facilitate the design process of digital systems.

DESIGNATION OF REGISTERS

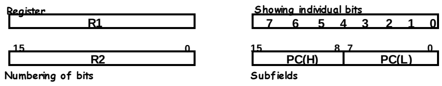
- Registers are designated by capital letters, sometimes followed by numbers (e.g., A, R13, IR)
- Often the names indicate function:
  - MAR - memory address register
  - PC - program counter
  - IR - instruction register

- Registers and their contents can be viewed and represented in *various ways*
  - A register can be viewed as a single entity:
  - Registers may also be represented showing the bits of data they contain

DESIGNATION OF REGISTERS

- **Designation of a register**
  - **a register**
  - **portion of a register**
  - **a bit of a register**

- **Common ways of drawing the block diagram of a register**



Register Transfer language

- Symbolic notations used to describe the micro Operation transfers among register is called a Register Transfer language (RTL)
- Convenient way to describe the internal organization and operation of digital systems.
- Computer registers are designated by capital letters. (Sometimes with numbers)
  - EX: MAR – memory address register
  - PC – Program Counter
  - IR – Instruction register

Register Transfer language

- Provides a language for describing the behavior of computers in terms of step-wise register contents
- Provides a formal means of describing machine structure and function
- Is at the "just right" level for machine descriptions
- Does not replace hardware description languages
- Can be used to describe *what* a machine does (an Abstract RTL) without describing *how* the machine does it
- Can also be used to describe a particular hardware implementation (A Concrete RTL)

RTL

- RTL describes the behavior of computers as stepwise transformations on register contents.
- Describe specific computers at the hardware level
- Variables correspond to the hardware registers
- Operations correspond to the hardware logic
- Verilog is becoming the standard design language in industry

REGISTER TRANSFER

- Copying the contents of one register to another is a register transfer
- A register transfer is indicated as

  R2 ← R1

  In this case the contents of register R2 are copied (loaded) into register R1

  - A simultaneous transfer of all bits from the source R1 to the destination register R2, during one clock pulse
  - Note that this is a non-destructive; i.e. the contents of R1 are not altered by copying (loading) them to R2

REGISTER TRANSFER...

- A register transfer such as

  R3 ← R5

  Implies that the digital system has

  - the data lines from the source register (R5) to the destination register (R3)
  - Parallel load in the destination register (R3)
  - Control lines to perform the action

CONTROL FUNCTIONS

- Often actions need to only occur if a certain condition is true
- This is similar to an "if" statement in a programming language
- In digital systems, this is often done via a *control signal*, called a *control function*
  - If the signal is 1, the action takes place
- This is represented as:

  P: R2 ← R1

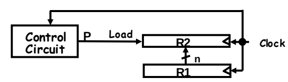
  Which means "if P = 1, then load the contents of register R1 into register R2", i.e., if (P = 1) then (R2 ← R1)

HARDWARE IMPLEMENTATION OF CONTROLLED TRANSFERS

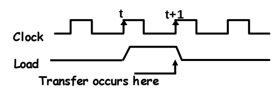
Implementation of controlled transfer

P: R2 ← R1

Block diagram



Timing diagram



- **The same clock controls the circuits that generate the control function and the destination register**
- **Registers are assumed to use *positive-edge-triggered* flip-flops**

SIMULTANEOUS OPERATIONS

- If two or more operations are to occur simultaneously, they are separated with commas

  P: R3 ← R5, MAR ← IR

- Here, if the control function P = 1, load the contents of R5 into R3, and at the same time (clock), load the contents of register IR into register MAR

RTL ....

- Instruction transfer from one register to another
  - $R_2 \leftarrow R_1$
  - Destination ← Source
  - $R_2(H) \leftarrow PC$ (8-15) – Denote a part of register
- Register transfer under conditions
  - If (p=1) then ($R_2 \leftarrow R_1$)
  - When p is control signal
  - P: $R_2 \leftarrow R_1$
- Comma is used to separate operations that execute at the same time
  - P: $R_2 \leftarrow R_1$, $R_3 \leftarrow R_1$ - p is control signal

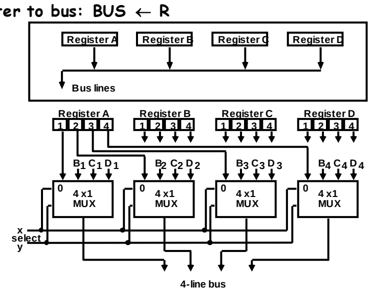
CONNECTING REGISTRS

- In a digital system with many registers, it is impractical to have data and control lines to directly allow each register to be loaded with the contents of every possible other registers
- To completely connect n registers → n(n-1) lines O(n^2) cost
  - This is not a realistic approach to use in a large digital system
- Instead, take a different approach
- Have one centralized set of circuits for data transfer – the bus
- Have control circuits to select which register is the source, and which is the destination

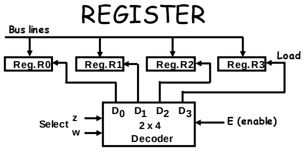
BUS AND BUS TRANSFER

Bus is a path(of a group of wires) over which information is transferred, from any of several sources to any of several destinations.

From a register to bus: BUS ← R

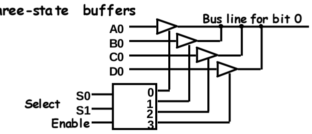


TRANSFER FROM BUS TO A DESTINATION REGISTER



Three-State Bus Buffers

Bus line with three-state buffers



BUS TRANSFER IN RTL

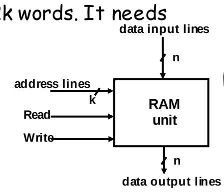
- Depending on whether the bus is to be mentioned explicitly or not, register transfer can be indicated as either
- **R2 ← R1**
- or
- **BUS ← R1, R2 ← BUS**

Memory Transfer

- Memory word is symbolized by letter M
- Address of the memory location denoted by M[AR]
  - AR denotes the address register
  - The data at location given by AR will transfer to DR (Data Register)
    - DR ← M[AR] - Read operation
    - M[AR] ← R1 – Write Operation

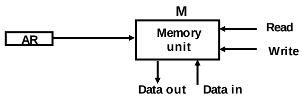
MEMORY (RAM)

- Memory (RAM) can be thought as a sequential circuits containing some number of registers
- These registers hold the *words* of memory
- Each of the r registers is indicated by an *address*
- These addresses range from 0 to r-1
- Each register (word) can hold n bits of data
- Assume the RAM contains r = 2k words. It needs the following
  - n data input lines
  - n data output lines
  - k address lines
  - Read control line
  - Write control line



MEMORY TRANSFER

- Collectively, the memory is viewed at the register level as a device, M.
- Since it contains multiple locations, we must specify which address in memory we will be using
- This is done by indexing memory references
- Memory is usually accessed in computer systems by putting the desired address in a special register, the *Memory Address Register* (*MAR*, or *AR*)
- When memory is accessed, the contents of the MAR get sent to the memory unit's address lines



MEMORY READ

- To read a value from a location in memory and load it into a register, the register transfer language notation looks like this:

R1 ← M[MAR]

- This causes the following to occur
  - The contents of the MAR get sent to the memory address lines
  - A Read (= 1) gets sent to the memory unit
  - The contents of the specified address are put on the memory's output data lines
  - These get sent over the bus to be loaded into register R1

MEMORY WRITE

- To write a value from a register to a location in memory looks like this in register transfer language:

- M[MAR] ← R1

- This causes the following to occur
  - The contents of the MAR get sent to the memory address lines
  - A Write (= 1) gets sent to the memory unit
  - The values in register R1 get sent over the bus to the data input lines of the memory
  - The values get loaded into the specified address in the memory

MICROOPERATIONS TYPES

- ***Computer system microoperations are of four types:***
  - ***Register transfer microoperations***
  - ***Arithmetic microoperations***
  - ***Logic microoperations***
  - ***Shift microoperations***

Arithmetic Operations

- Add operation
  - R3 ← R1 + R2
- Subtract operation
  - R3 ← R1 + R2 + 1 (= R1 – R2 = R3)
  - R2 is in 1's complement form and by adding 1 it become 2's complement form

ARITHMETIC MICROOPERATIONS

- The basic arithmetic microoperations are
  - Addition
  - Subtraction
  - Increment
  - Decrement
- The additional arithmetic microoperations are
  - Add with carry
  - Subtract with borrow
  - Transfer/Load
  - etc. …

Summary of Typical Arithmetic Micro-Operations

R3 ← R1 + R2	Contents of R1 plus R2 transferred to R3
R3 ← R1 - R2	Contents of R1 minus R2 transferred to R3
R2 ← R2'	Complement the contents of R2
R2 ← R2'+ 1	2's complement the contents of R2 (negate)
R3 ← R1 + R2'+ 1	subtraction
R1 ← R1 + 1	Increment
R1 ← R1 - 1	Decrement

BASIC SYMBOLS FOR REGISTER TRANSFERS

Symbols	Description	Examples
Capital letters & numerals	Denotes a register	MAR, R2
Parentheses ()	Denotes a part of a register	R2(0-7), R2(L)
Arrow ←	Denotes transfer of information	R2 ← R1
Colon :	Denotes termination of control function	P:
Comma ,	Separates two micro-operations	A ← B, B ← A

Logic Operations

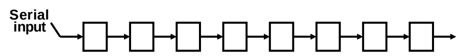
- P: R1 ← R1 ⊕ R2 – Exclusive OR Operation
- P+Q : R1 ← R2 – OR operation is at control signal
- R4 ← R5 U R6 – OR operation on registers
- R4 ← R5 /\ R6 – And operation on registers
- R4 ← R5 U R6 – Nor operation on registers

Shift Micro-Operations

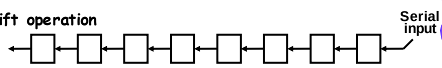
- R ← Shl R – Shift left register
- R ← Shr R – Shift right register

SHIFT MICROOPERATIONS

- There are three types of shifts
  - *Logical shift*
  - *Circular shift*
  - *Arithmetic shift*
- What differentiates them is the information that goes into the serial input
- **A right shift operation**

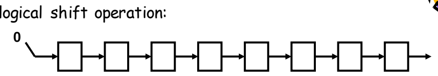


- **A left shift operation**

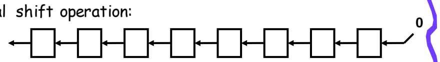


LOGICAL SHIFT

- In a logical shift the serial input to the shift is a 0.
- A right logical shift operation:



- A left logical shift operation:



- In a Register Transfer Language, the following notation is used
  - *shl* for a logical shift left
  - *shr* for a logical shift right
  - Examples:
    - R2 ← *shr* R2
    - R3 ← *shl* R3

Fetch & Execution Cycle

Fetch cycle:

MAR ← PC

INCREMENTER ← PC

PC ← INCREMENTER

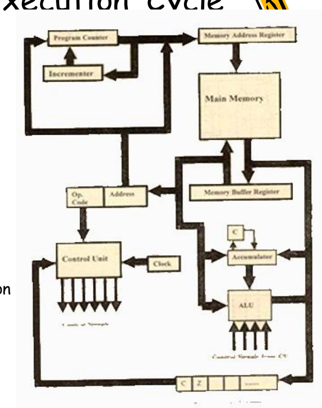
MBR ← M[MAR]

IR ← MBR

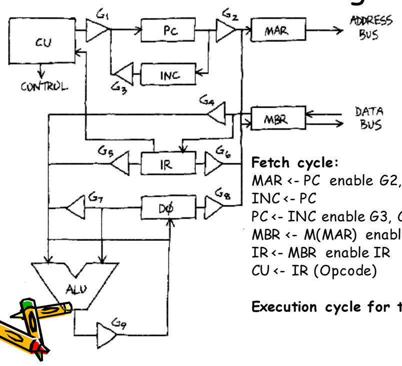
CU ← IR(Opcode)

If it is a arithmetic operation

Execution Cycle?



Fetch & Execution Cycle with control signals



Fetch cycle:
MAR <- PC enable G2, Clock MAR
INC <- PC
PC <- INC enable G3, Clock PC
MBR <- M(MAR) enable MS, Clock MBR
IR <- MBR enable IR
CU <- IR (Opcode)

Execution cycle for transfer data to ALU ?

- Questions?