EE2016 Microprocessor Theory and Lab

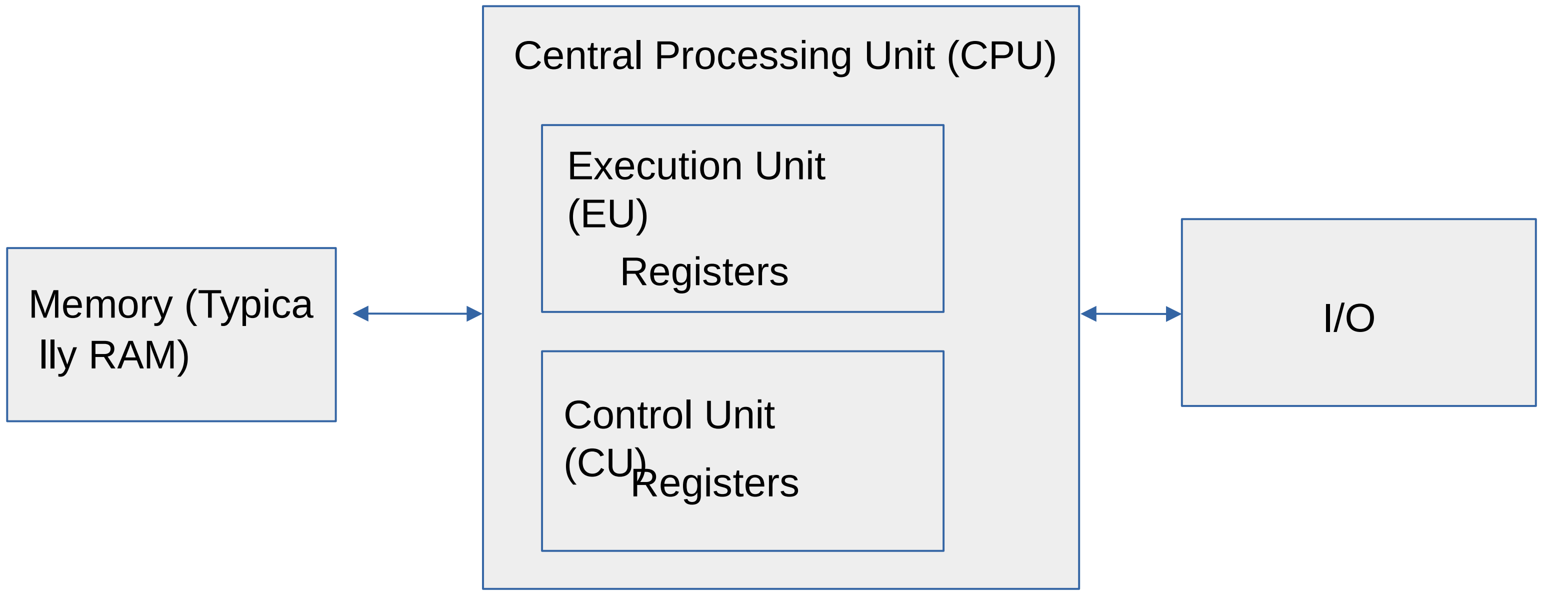
Van Neumann Computer & Memory Block: From Logic Gates to Memory

EE Dept – *IIT Madras Fall, 2024*

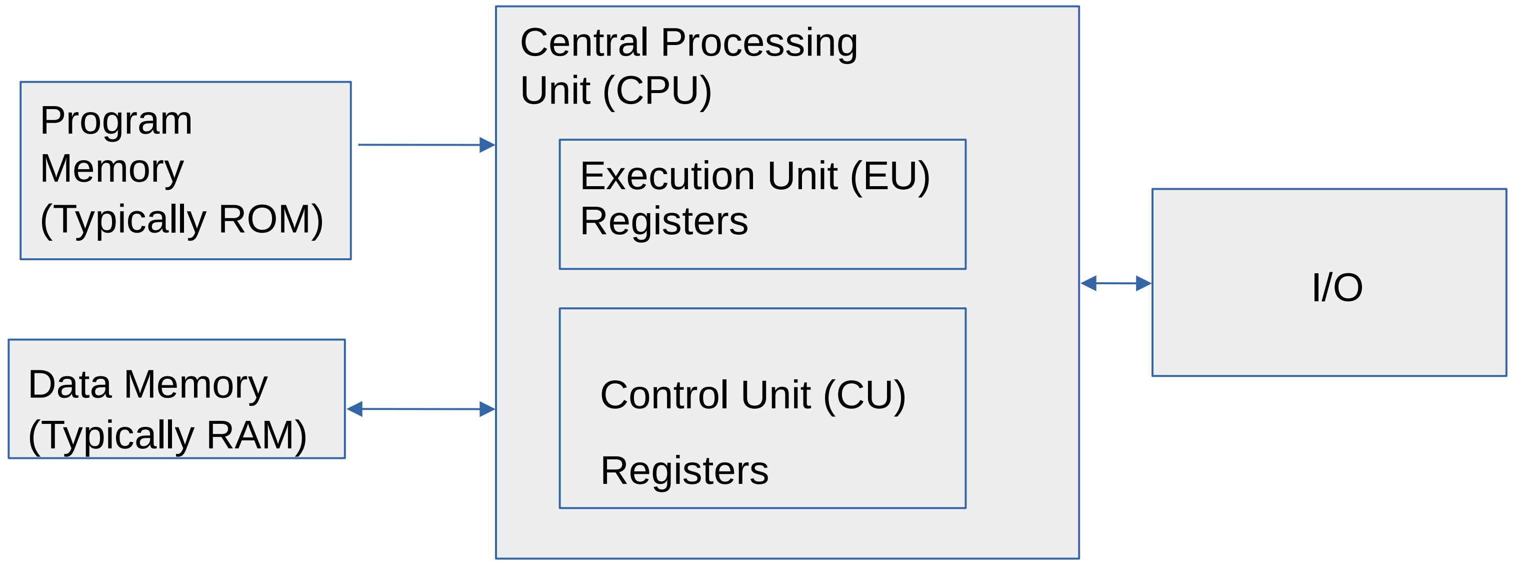
Dr. R. Manivasakan, OSWM Lab, EE Dept, IITM
Email: rmani@ee.iitm.ac.in

1st Week: 4th Class on 2.8.2024

Van Neumann Model



Harvard Model


Program Memory (Typically ROM)
Data Memory (Typically RAM)
Central Processing Unit (CPU)
Execution Unit (EU) Registers
Control Unit (CU)
Registers
I/O

Van Neumann Model

- Van Neumann machine
  - Program (sequence of instructions) & data are stored in the same memory.
  - Advantages – (a) just-in-time compilation (b) self-modifying codes
  - Disadvantages – prone to (a) malaware and (b) software defects (c) data transfer and instruction fetches could not be performed at the same time (needed two clock cycles)

Harvard Model

- Harvard model
  - Program (sequence of instructions) & data are stored in the physically separate storage area
  - Advantages: (a) storage area & signal pathways (buses) are different & hence simultaneous access is possible (b) one instruction per cycle?
  - Disadvantages: (a) embedding the data within instructions are often needed (ex. Self modifying code / ARM debugger break points etc), which is not possible if the memory is different
- A combination of modified Harvard and van Neumann machine is used in modern designs
  - Split-cache version of modified Harvard architecture

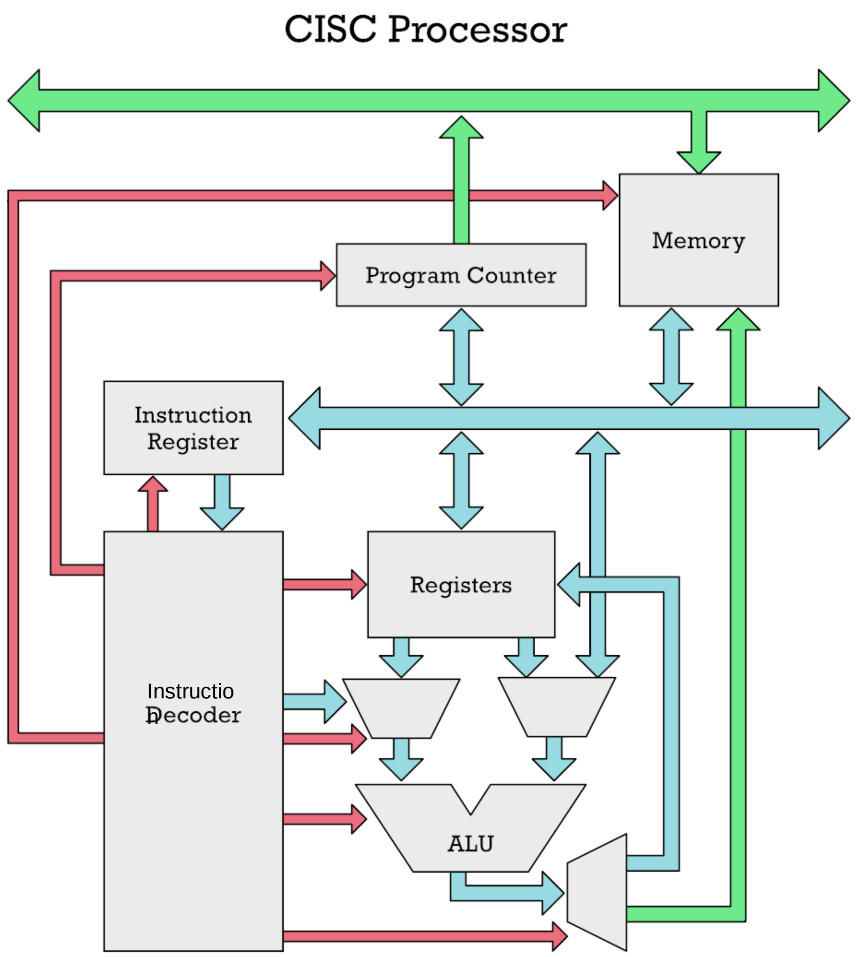
RISC and CISC

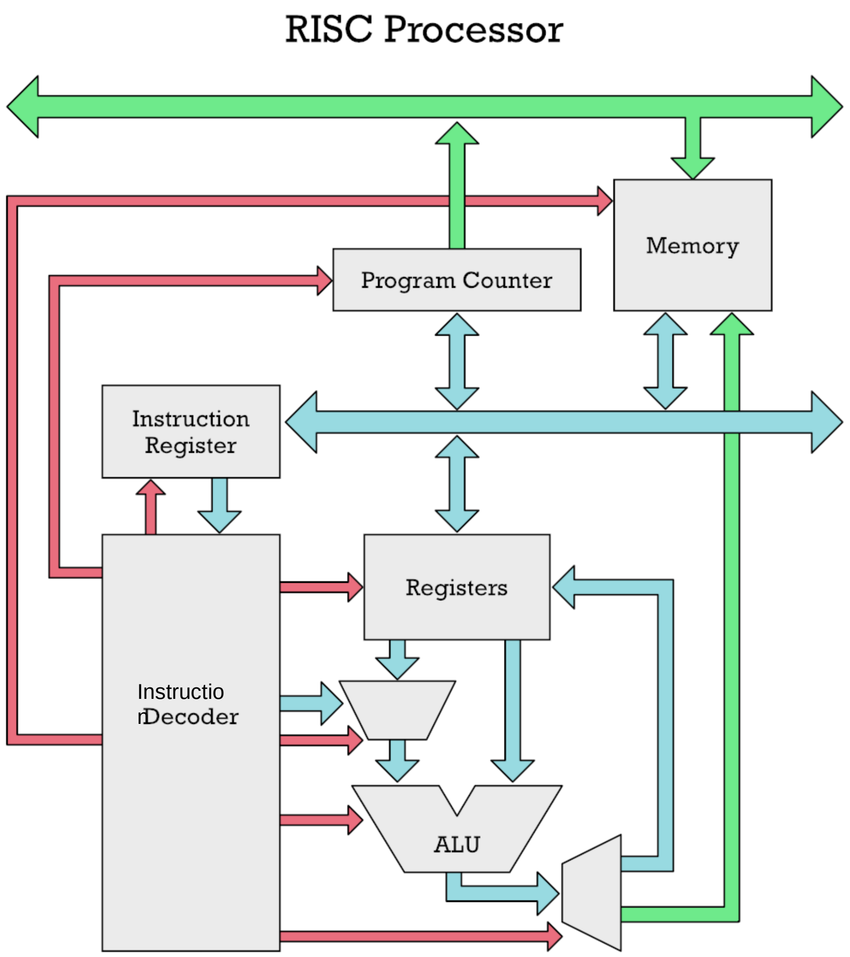
- CISC:
  - Large programs need large memory --> costly
  - To reduce no of instructions / program or task (sacrifying number of cycles/ instruction) --> programs have lesser number of lines --> cheap
  - --> instructions designed with more operations --> complex instructions --> CISC
  - Compound instructions (union of many primitive, unique & basic instructions)
  - Corresponding ISA might be larger (to suit the given task, many flavours of similar instruction)
- RISC
  - Instructions are unique, primitive do an unique operation or minimal no of operations
  - Highly optimized set of instructions --> corresponding cardinality of Instruction Set is minimal, yet could do all possible tasks that a CISC could do
    - No redundancy / overlap among instructions. One instruction cant be replaced by another.
  - Simple addressing modes/ fixed length instructions
  - By and large (?), one cycle execution time

CISC versus RISC

S. No	Issues	RISC	CISC
		Emphasis on software (ISA)	Emphasis on hardware (organization)
		Minimal number of fixed length instructions in ISA (AVR 78, 3 byte?)	Large number of (possibly variable length) instructions in ISA (intel 338 i7 8 byte?)
		Instructions are unique, primitive do an unique operation --> standardised instructions --> given an operation only one unique instruction (out of ISA) can handle it	Complex, often there is an overlap among many instructions. Given an operation, many instructions can do the same.
		Single clock cycle instructions	Instructions can take several clock cycles
		Heavy use of RAM	More efficient use of RAM
		Given a task, large number of instructions --> program memory needed is large	Given the same task, small number of instructions --> program memory needed is small
		Uses, “Load & Store” architecture	Operation directly over data in main memory is also a possibility
		ISA is divided into memory operations & ALU operations	Such division within ISA is not possible, since architecture allows ALU operations directly on data in main memory
		Hardwired unit to decode instructions	Microprogramming to convert instructions to micro instructions & then decode?
		Example of RISC: ARM, PA-RISC, Power Architecture, Alpha, AVR, ARC and the SPARC.	Examples of CISC: VAX, Motorola 68000 family, System/360, AMD and the Intel x86 CPUs

- RISC – V (RISC five) architecture
  - is the V generation open standard RISC architecture, developed at UCBProgram (sequence of instructions) & data are stored in the physically separate storage area
  - Is a load-store architecture & IEEE 754 floating point architecture
  - RISC-V ISA include instruction bit field locations chosen to simplify the use of multiplexers in CPU

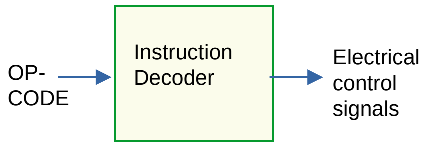


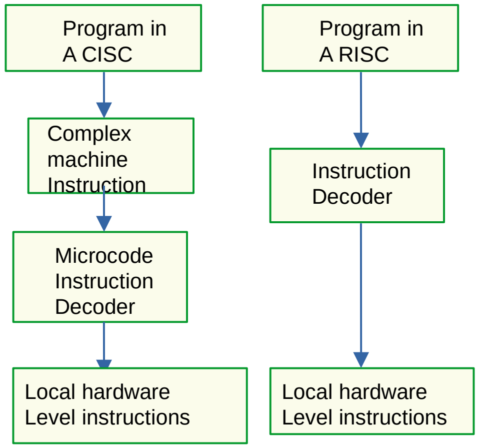


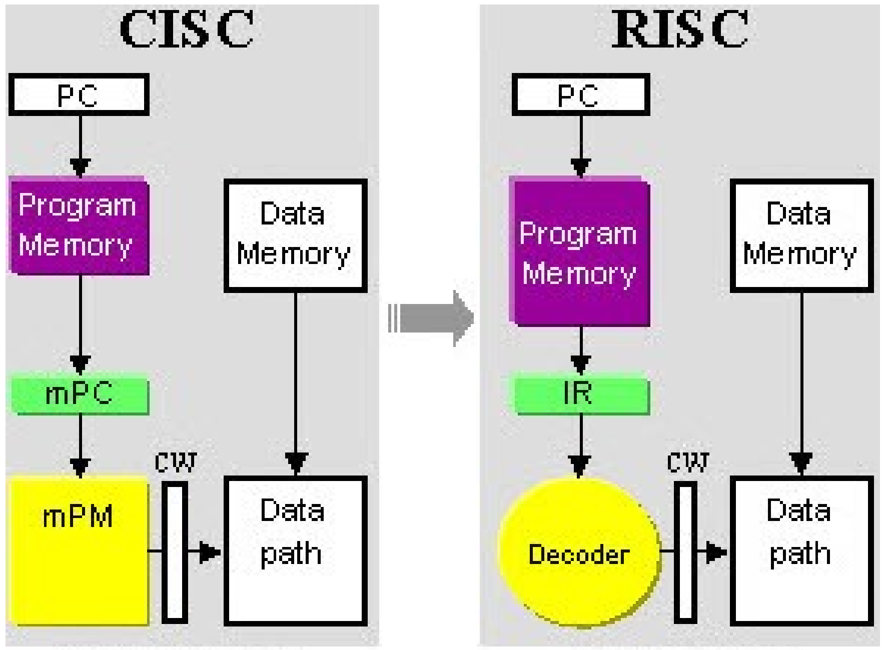
🟦 Data Bus 🟩 Address Bus 🟥 Control Lines 🟨 Data Flow

RISC Vs CISC

- CISC
  - mPM - micro program memory: if faster than the program memory.
  - mPC - micro-program counter
  - CW – hardware command level "Control word"

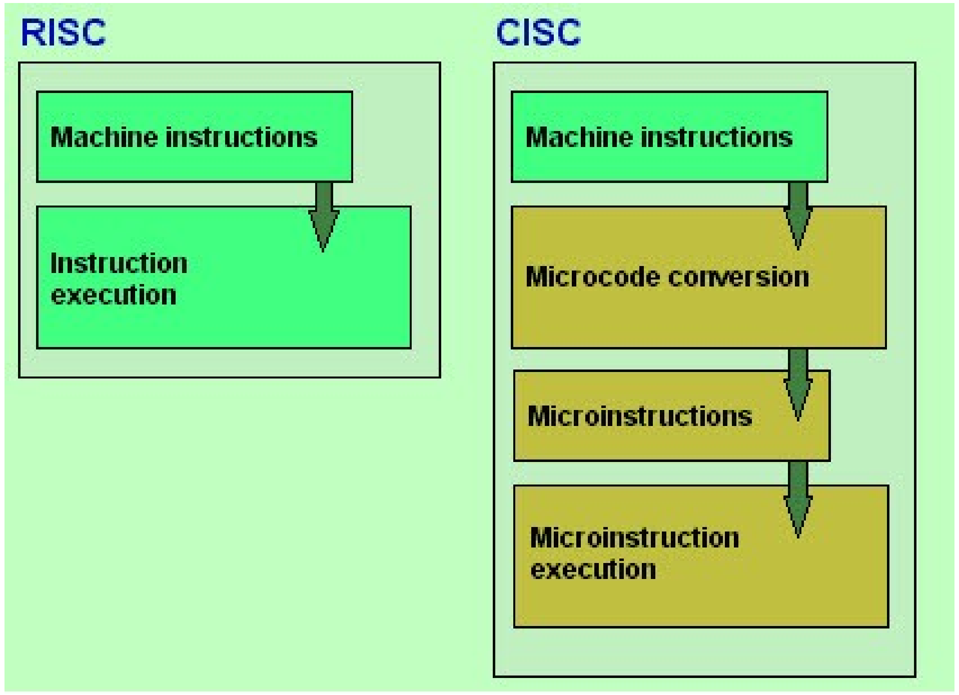



Program in A CISC
Complex machine Instruction
Microcode Instruction Decoder
Local hardware Level instructions
Program in A RISC
Instruction Decoder
Local hardware Level instructions



Complex instructions
One instruction = several CW
PM standard

Simple instructions
One instruction = One CW
PM longer



Some more concepts

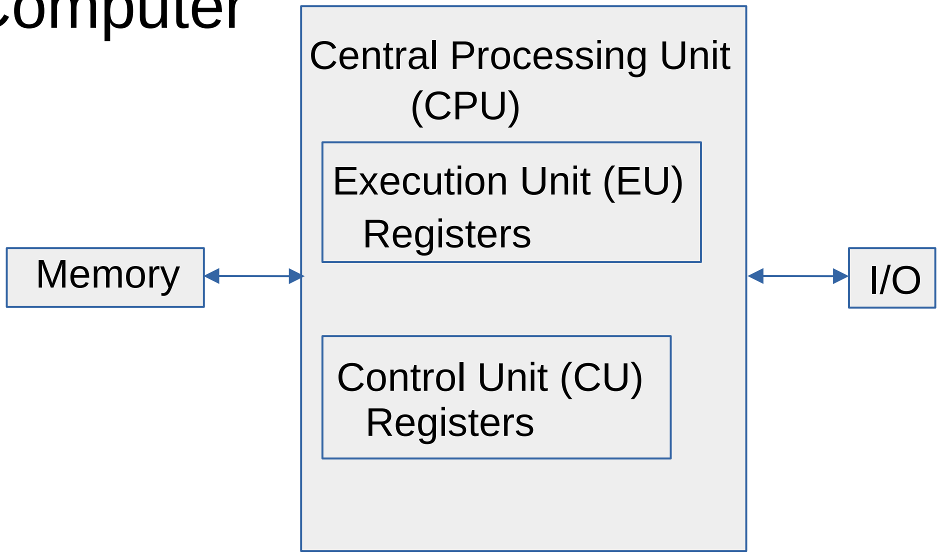
- Floating point format
  - Compromise between precision & storage requirements
- Microcode
  - Microcode is a layer of hardware-level instructions that implement higher-level machine code instructions in terms of CPU circutry level commands.
  - Microcode is used in general-purpose central processing units, although in current desktop CPUs, it is only a fallback path for cases that the faster hardwired control unit cannot handle.
  - Microcode typically resides in special high-speed memory and translates machine instructions, state machine data, or other input into sequences of detailed circuit-level operations.
  - It separates the machine instructions from the underlying electronics so that instructions can be designed and altered more freely.
  - It also facilitates the building of complex multi-step instructions, while reducing the complexity of computer circuits.
- S

Approach Used For Microprocessor Study

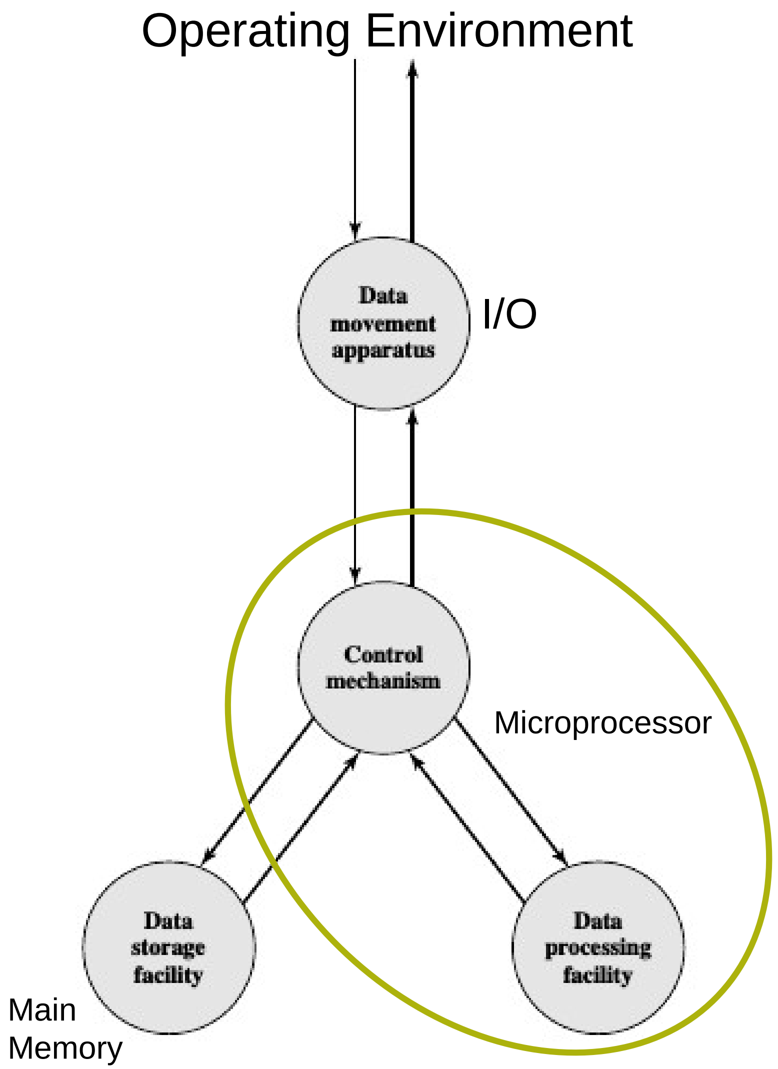
- Bottom top approach
- Top down approach
  - Clearest and most effective
- Structure and Function

Computer Models

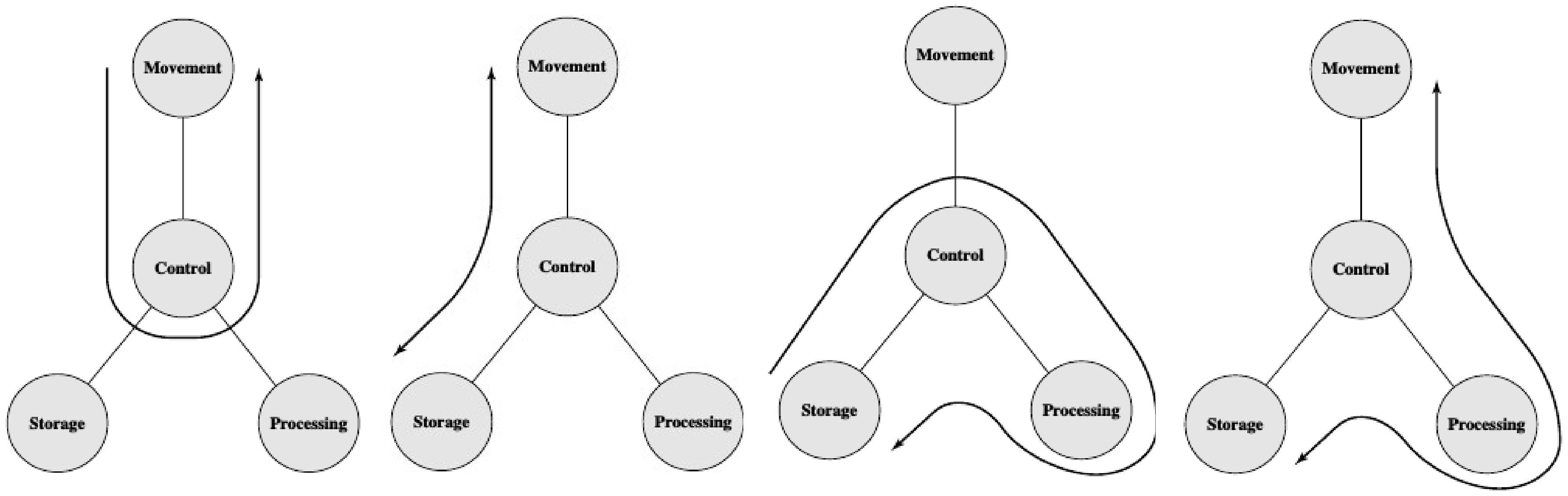
Conventional Model for Computer


Central Processing Unit (CPU)
Execution Unit (EU) Registers
Control Unit (CU) Registers
Memory
I/O

Functional Model for Computer

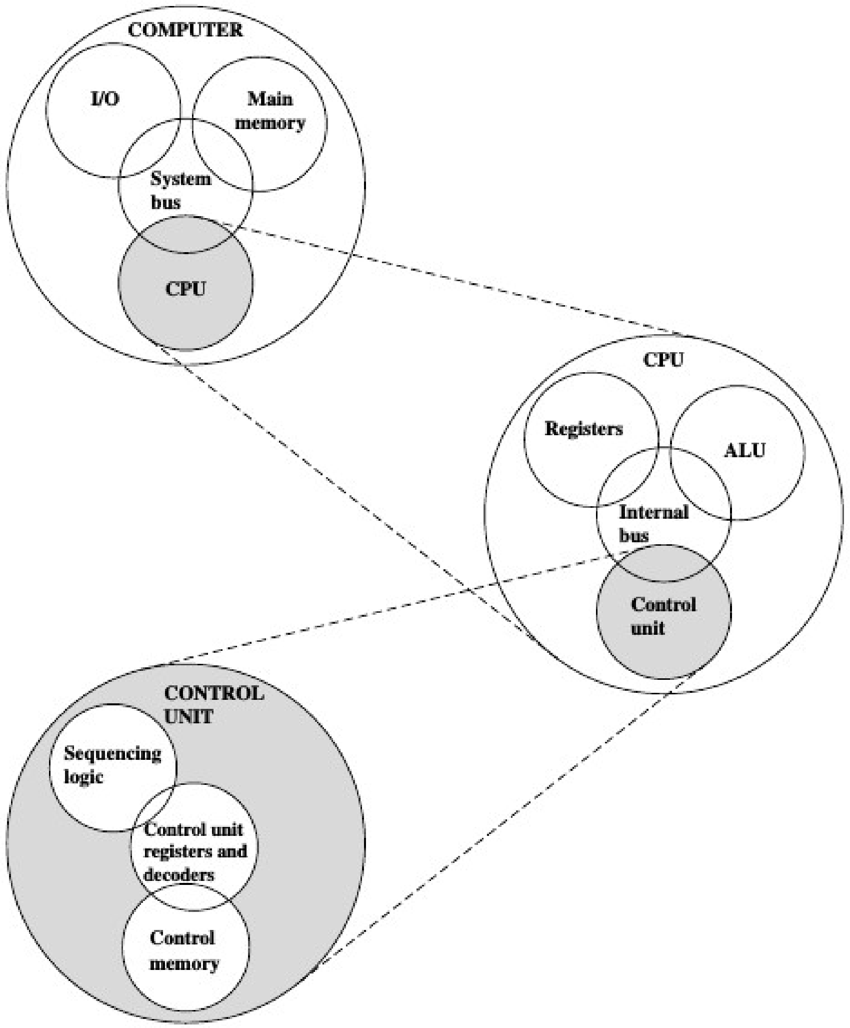


Possible Computer Operational Modes

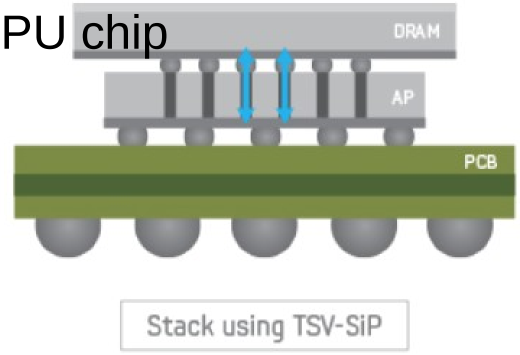


2nd Week: 5th Class on 5.8.2024

Computer: Top Level



RAM cant be integrated into
CPU chip



Birth & Evolution of Computers

- 1939, vaccum tube based computer
  - The Atanasoff–Berry computer, a prototype of which was first demonstrated in 1939, is now credited as the first vacuum-tube computer
- 1941, electromechanical computer Z3 (program controlled computers)
- 1945 Van Neumann’s stored program solid state but built with discrete transistor based computer
- S

●

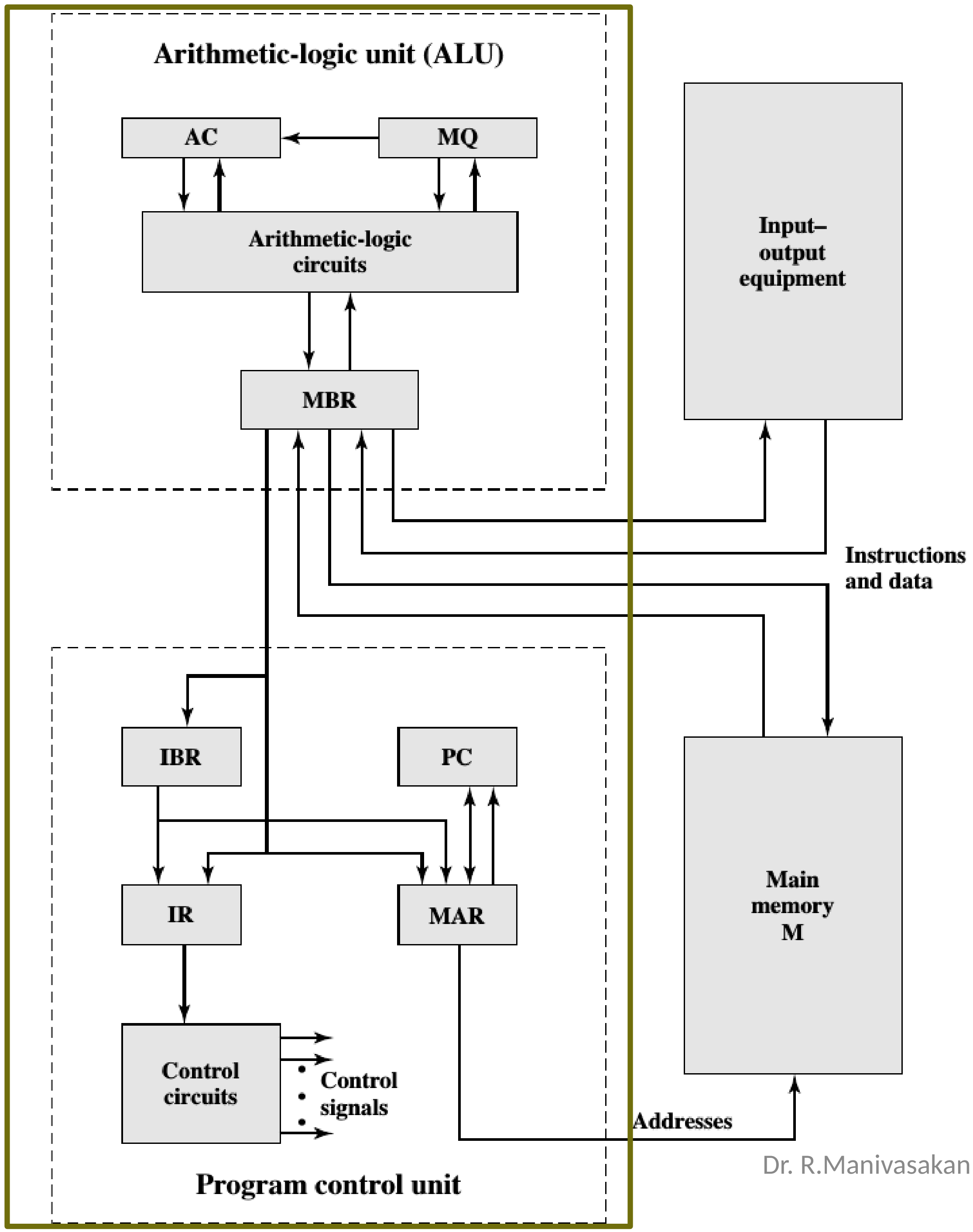
Early electro-mechanical Computer

- Z3, was a German electro - mechanical computer by Konrod Zuse
  - World’s first working programma -ble fully automatic digital computer
  - Started in 1935, completed in 1941
  - 2600 relays, 22-bit word length
  - Clock frequency of 5-10 Hz
  - Program code stored on punched film
  - Z3 lacked conditional branching

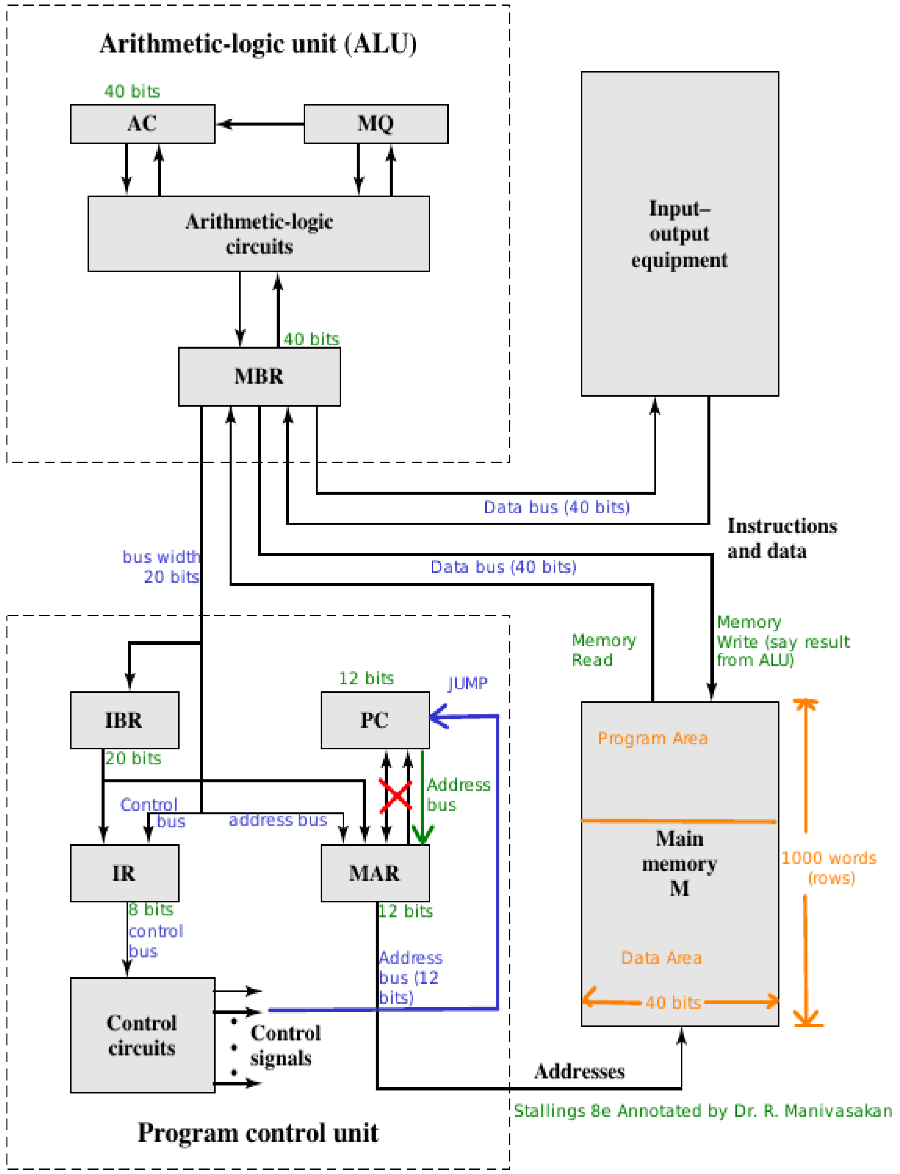
- Von Neumann Computer
  - Electronic stored program computer - keeps both program instructions and data in read–write, random-access memory (RAM)
  - As an aside: program-controlled computers (1940s) - programmed by setting switches and inserting patch cables to route data and control signals between various functional units
  - Most modern computers have common memory for both data and program instructions
    - but have separate caches (small faster memory between the CPU & main memory), for instructions and data, so that most instruction & data fetches use separate buses (split cache architecture).

CPU

Von Neumann model

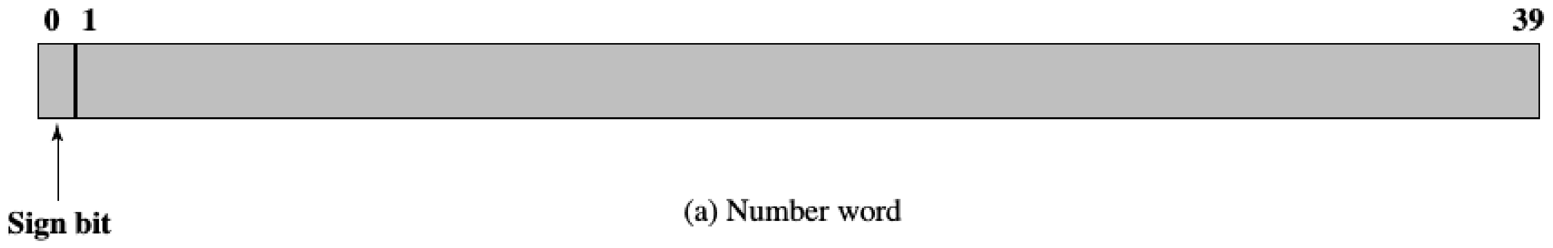


Von Neumann model

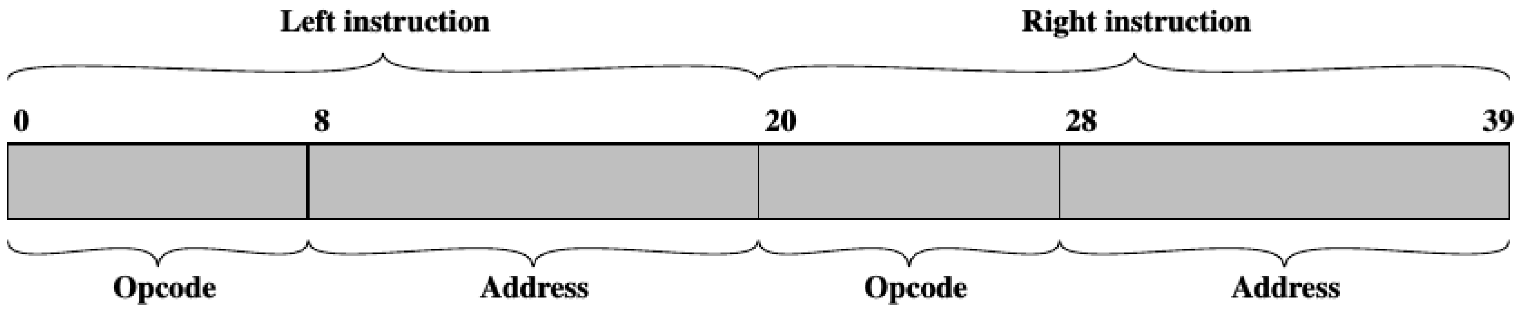


Von Neumann model

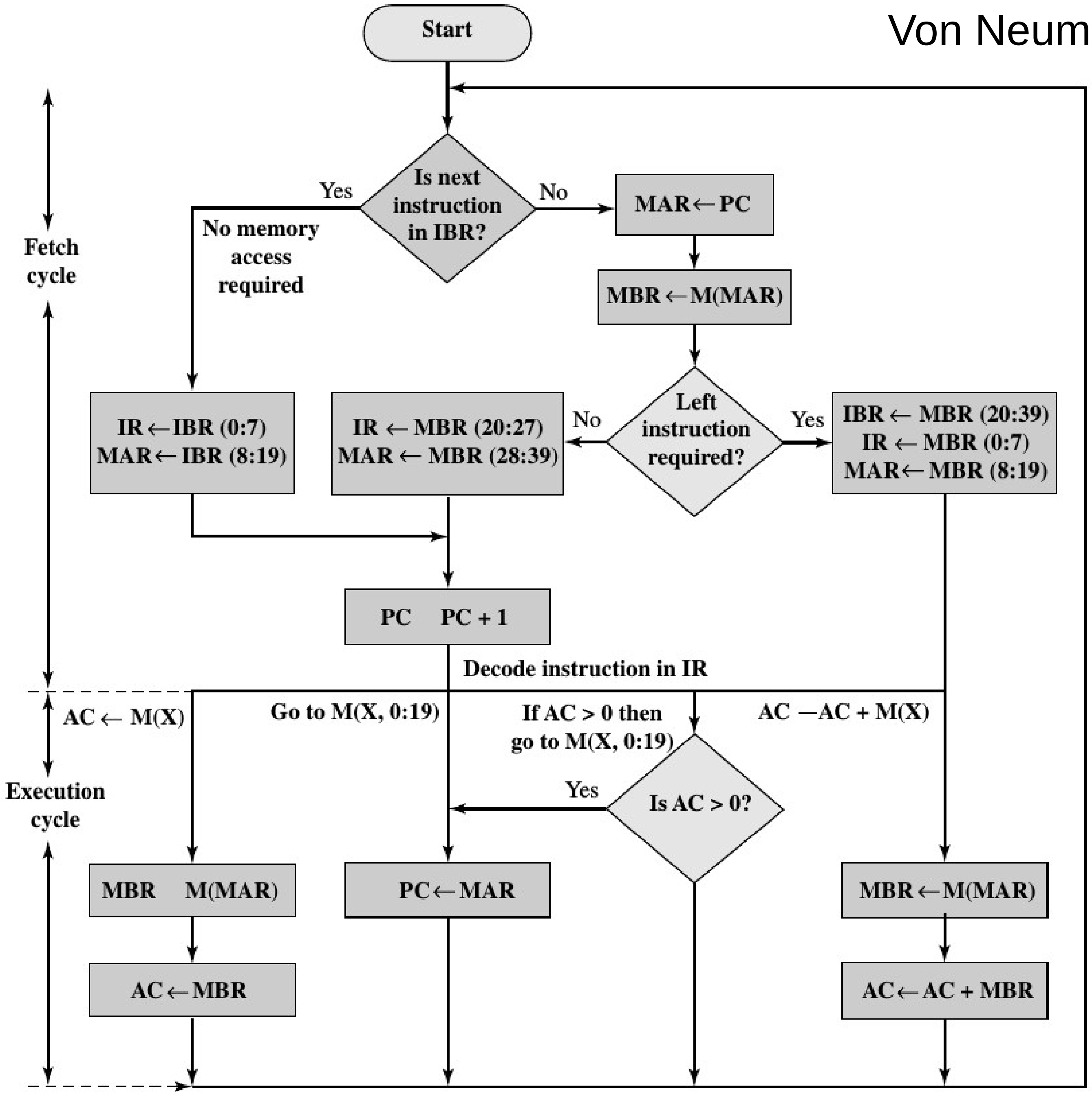
Words in Program Memory in Von Neumann Model



(a) Number word

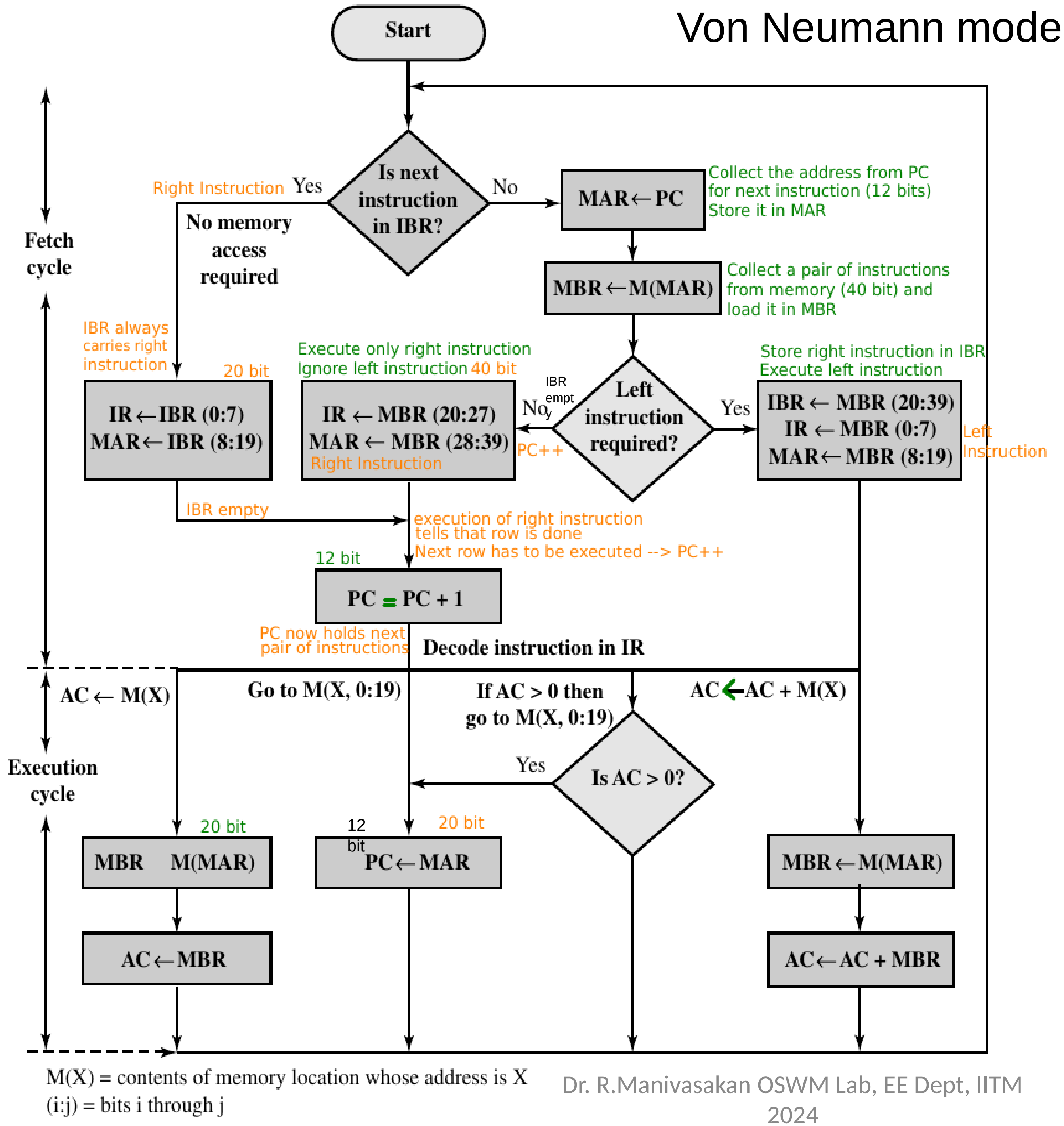


Von Neumann model

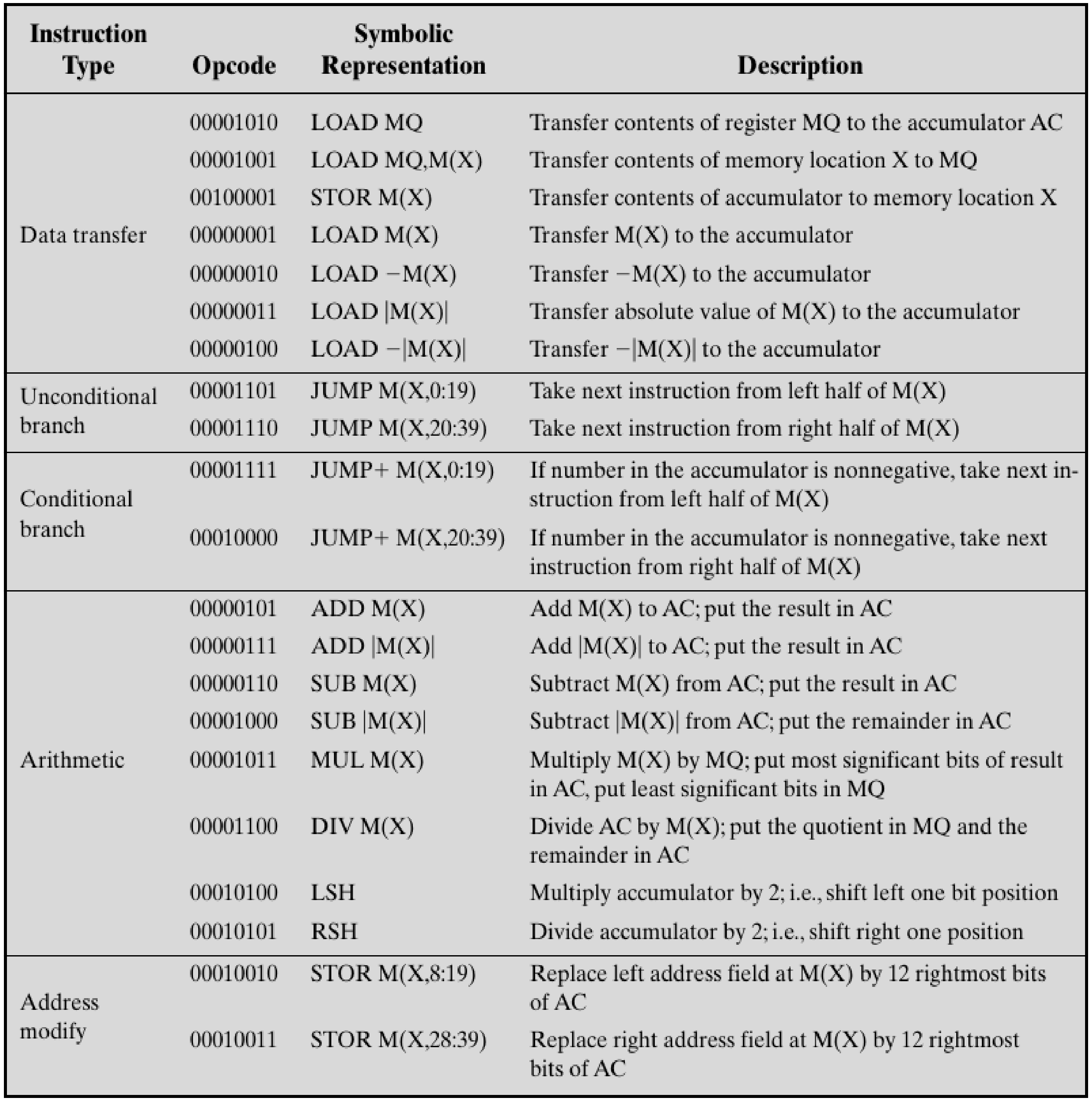


M(X) = contents of memory location whose address is X
(i:j) = bits i through j

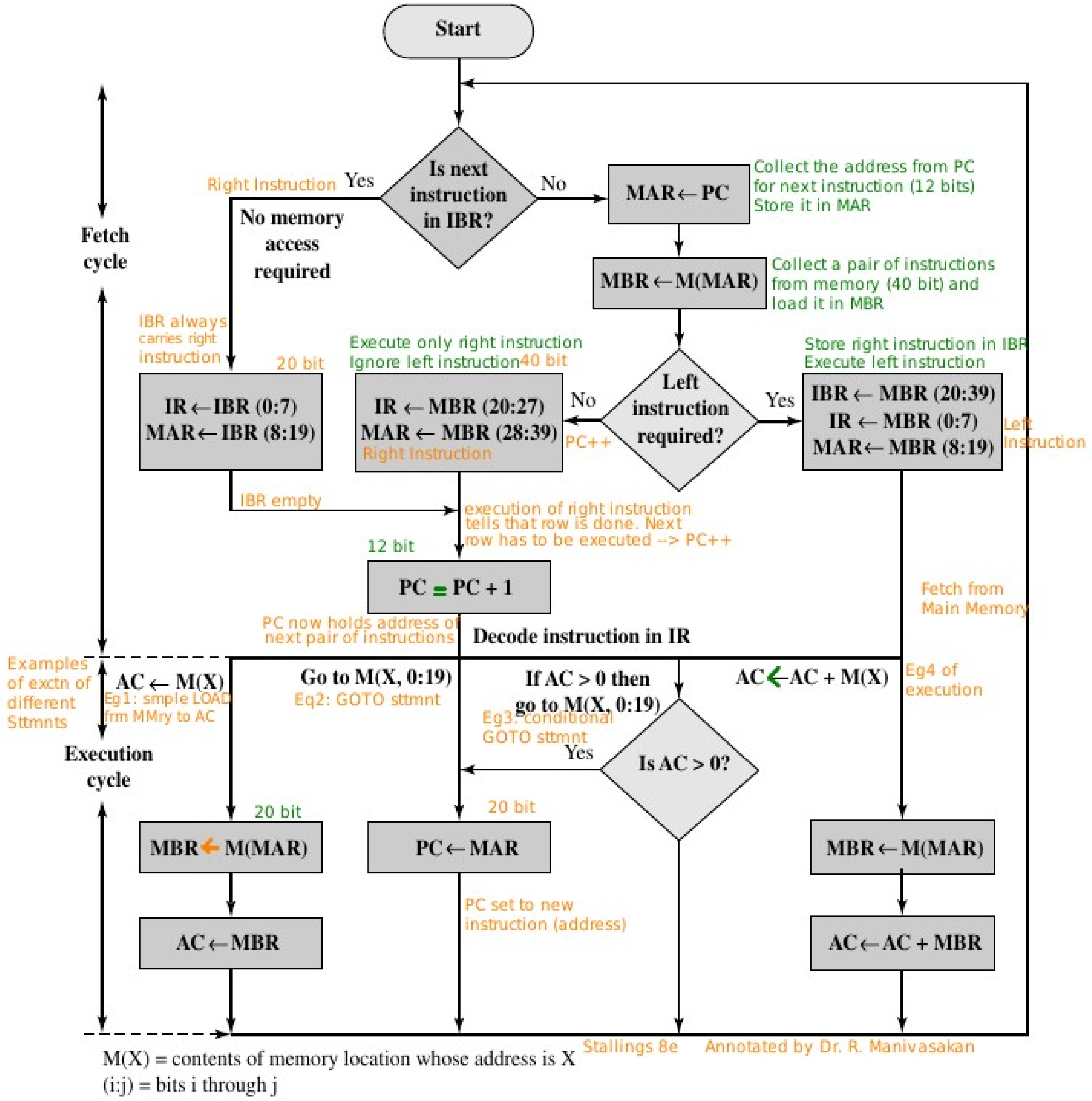
Von Neumann model


Start
Is next instruction in IBR?
Right Instruction
Yes
No
No memory access required
MAR ← PC
Collect the address from PC for next instruction (12 bits) Store it in MAR
MBR ← M(MAR)
Collect a pair of instructions from memory (40 bit) and load it in MBR
Fetch cycle
IBR always carries right instruction
20 bit
Execute only right instruction Ignore left instruction
40 bit
IBR empty
No
Left instruction required?
Yes
Store right instruction in IBR Execute left instruction
IR ← IBR (0:7)
MAR ← IBR (8:19)
IR ← MBR (20:27)
MAR ← MBR (28:39)
Right Instruction
PC++
IBR ← MBR (20:39)
IR ← MBR (0:7)
MAR ← MBR (8:19)
Left Instruction
IBR empty
execution of right instruction tells that row is done
Next row has to be executed --> PC++
12 bit
PC = PC + 1
PC now holds next pair of instructions
Decode instruction in IR
AC ← M(X)
Go to M(X, 0:19)
If AC > 0 then go to M(X, 0:19)
AC ← AC + M(X)
Execution cycle
Yes
Is AC > 0?
20 bit
12 bit
20 bit
MBR M(MAR)
PC ← MAR
MBR ← M(MAR)
AC ← MBR
AC ← AC + MBR
M(X) = contents of memory location whose address is X
(i:j) = bits i through j

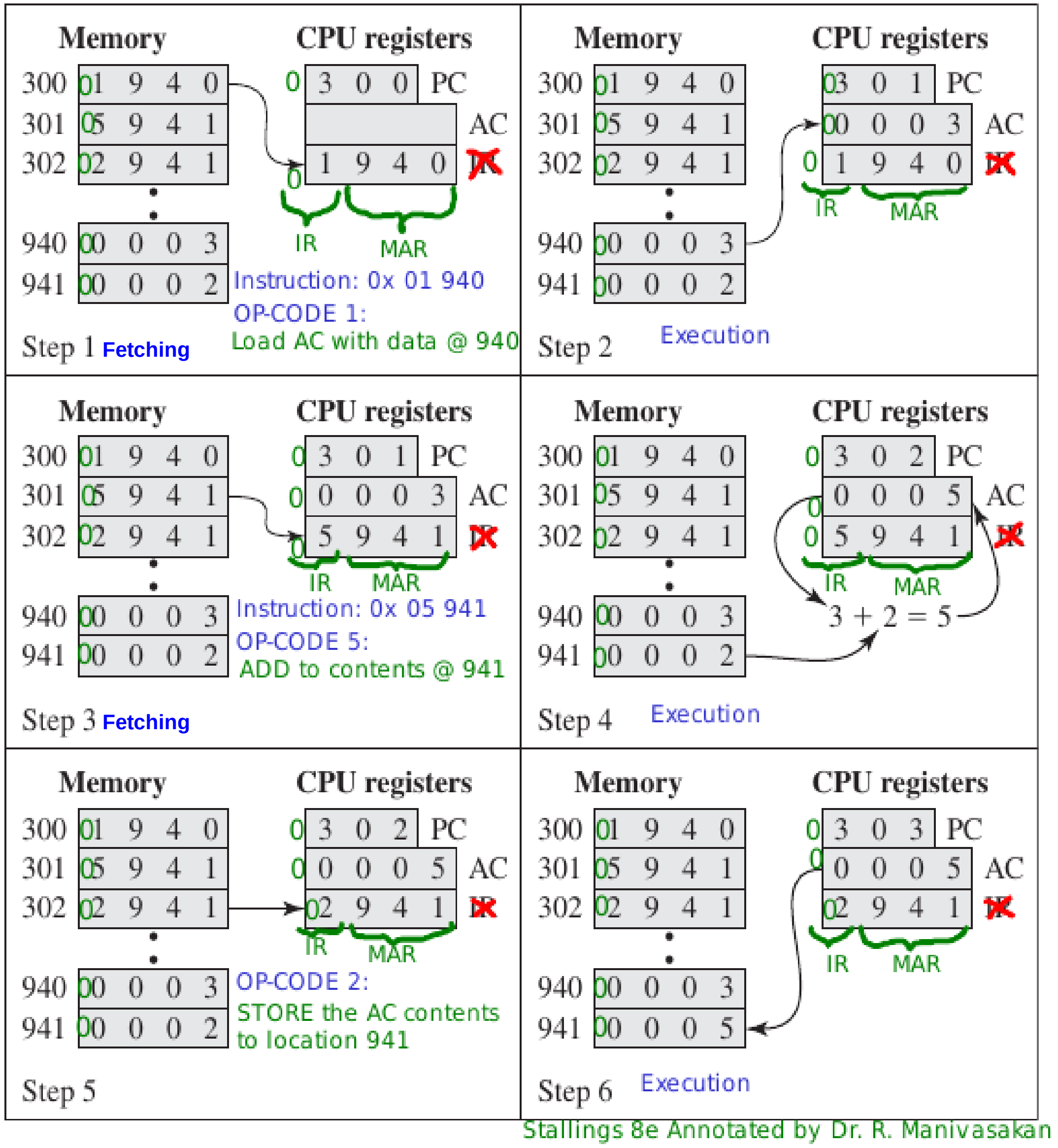
Instruction Set in Von Neumann model



Instruction Type	Opcode	Symbolic Representation	Description
Data transfer	00001010	LOAD MQ	Transfer contents of register MQ to the accumulator AC
	00001001	LOAD MQ,M(X)	Transfer contents of memory location X to MQ
	00100001	STOR M(X)	Transfer contents of accumulator to memory location X
	00000001	LOAD M(X)	Transfer M(X) to the accumulator
	00000010	LOAD −M(X)	Transfer −M(X) to the accumulator
	00000011	LOAD |M(X)|	Transfer absolute value of M(X) to the accumulator
	00000100	LOAD −|M(X)|	Transfer −|M(X)| to the accumulator
Unconditional branch	00001101	JUMP M(X,0:19)	Take next instruction from left half of M(X)
	00001110	JUMP M(X,20:39)	Take next instruction from right half of M(X)
Conditional branch	00001111	JUMP+ M(X,0:19)	If number in the accumulator is nonnegative, take next instruction from left half of M(X)
	00010000	JUMP+ M(X,20:39)	If number in the accumulator is nonnegative, take next instruction from right half of M(X)
Arithmetic	00000101	ADD M(X)	Add M(X) to AC; put the result in AC
	00000111	ADD |M(X)|	Add |M(X)| to AC; put the result in AC
	00000110	SUB M(X)	Subtract M(X) from AC; put the result in AC
	00001000	SUB |M(X)|	Subtract |M(X)| from AC; put the remainder in AC
	00001011	MUL M(X)	Multiply M(X) by MQ; put most significant bits of result in AC, put least significant bits in MQ
	00001100	DIV M(X)	Divide AC by M(X); put the quotient in MQ and the remainder in AC
	00010100	LSH	Multiply accumulator by 2; i.e., shift left one bit position
	00010101	RSH	Divide accumulator by 2; i.e., shift right one position
Address modify	00010010	STOR M(X,8:19)	Replace left address field at M(X) by 12 rightmost bits of AC
	00010011	STOR M(X,28:39)	Replace right address field at M(X) by 12 rightmost bits of AC


Start
Is next instruction in IBR?
Yes
No
Right Instruction
No memory access required
MAR ← PC
Collect the address from PC for next instruction (12 bits) Store it in MAR
MBR ← M(MAR)
Collect a pair of instructions from memory (40 bit) and load it in MBR
Fetch cycle
IBR always carries right instruction
20 bit
Execute only right instruction Ignore left instruction 40 bit
Store right instruction in IBR Execute left instruction
IR ← IBR (0:7)
MAR ← IBR (8:19)
IR ← MBR (20:27)
MAR ← MBR (28:39)
Right Instruction
PC++
Left instruction required?
IBR ← MBR (20:39)
IR ← MBR (0:7)
MAR ← MBR (8:19)
Left Instruction
IBR empty
execution of right instruction tells that row is done. Next row has to be executed --> PC++
12 bit
PC = PC + 1
Fetch from Main Memory
PC now holds address of next pair of instructions
Decode instruction in IR
Examples of exctn of different Sttmnts
AC ← M(X)
Eg1: smple LOAD frm MMry to AC
Go to M(X, 0:19)
Eq2: GOTO sttmnt
If AC > 0 then go to M(X, 0:19)
Eg3: conditional GOTO sttmnt
AC ← AC + M(X)
Eg4 of execution
Execution cycle
Is AC > 0?
20 bit
MBR ← M(MAR)
PC ← MAR
MBR ← M(MAR)
AC ← MBR
PC set to new instruction (address)
AC ← AC + MBR
Stallings 8e
Annotated by Dr. R. Manivasakan
M(X) = contents of memory location whose address is X
(i:j) = bits i through j

Example Execution in Von Neumann's Computer



Load 940;
ADD 941;
ST 941;

Processor Design

- Processor Execution Unit (EU)
    - Arithmetic Logic Unit (ALU)
    - Dedicated computational hardwares (Shift-registers, Booths multipliers, floating point units, address generation unit, load & store unit, etc)
    - Registers native to Execution Unit
    - Status Registers
- Processor Control Unit (CU)
    - Program Counter (Instruction pointer)
    - Stack pointer
    - Instruction Register & Instruction Decoder
    - Program Flow Control Logic
    - Memory Address Register (MAR)
    - Registers native to Control Unit
- Processor Interconnect to External Devices
    - Memory
    - Peripheral Control Unit

Review of EE2001 Digital Systems

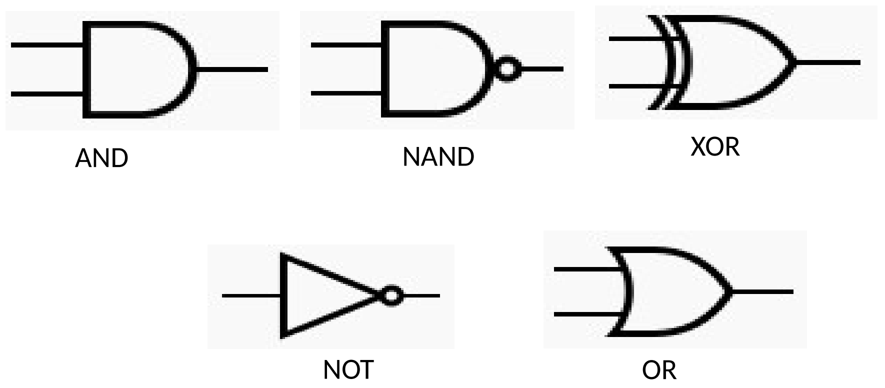
- Recall Digital Systems – Basics
- Universal GATEs
  - Combinational Circuits
  - Sequential Circuits
  - Max-Min Representations
- Flip-Flops
- Counters

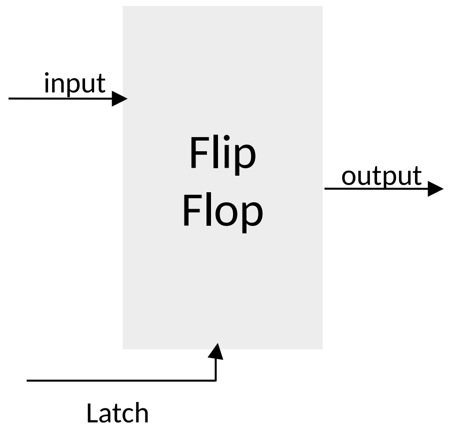
DATA-FLOW In Microprocessors

- 90% plus of all processor instructions consist of data flow
    - Basic Logic Gates / flip-flops
    - Interconnecting Gates, READ / Output Enable, WRITE / Latch
    - DATA Bus, Registers
    - ADDRESS Bus
    - Control Lines
- MOV Instruction, causing data flow or data movement, keeps the processor moving irrespective of the program used
- Arithmetic and Logic function (including Shift / Rotate) contributes to most of the other instruction
    - Together contributing to over 99% of all processor activities

Data Flow components: Logic-gates & flip-flops and their Interconnects

- Logic gates and Flip-flops serve as the building blocks of any digital logic circuits
- All basic operations like memory transfers, I/O or computer arithmetic are performed using logic gates and Flip-flops in synergy



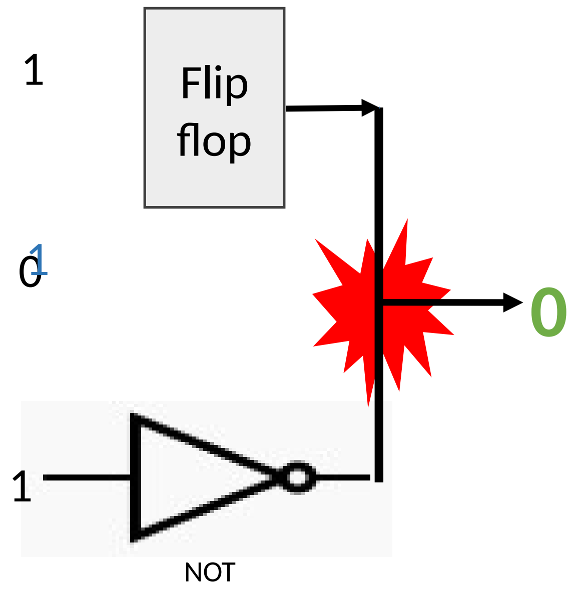


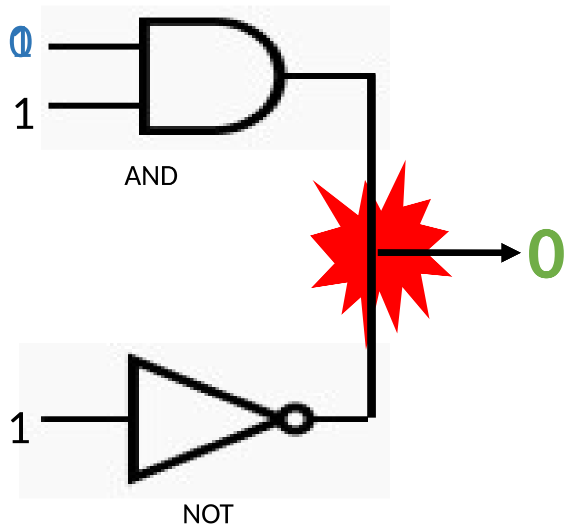
2nd Week: 6th Special Class on 6.8.2024

Interconnecting Gates /Flip-flop outputs

- What will happen if we connect Output of two Gates / flip-flops?

They are likely to have short-circuit or burn

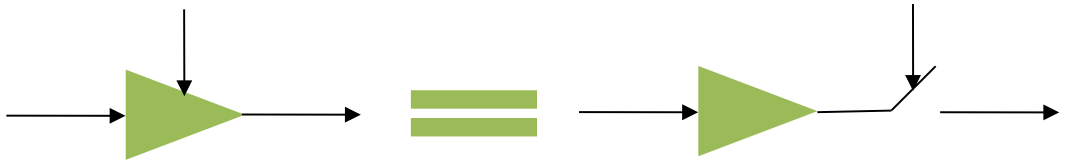




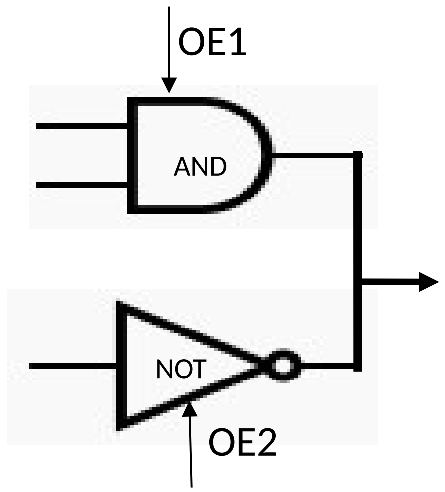
The outputs can not be connected as they have different values, causing short circuit: likely to burn

Tristate Gates to the rescue

- ➢Tristate Gates : A gate
- with a switch at output



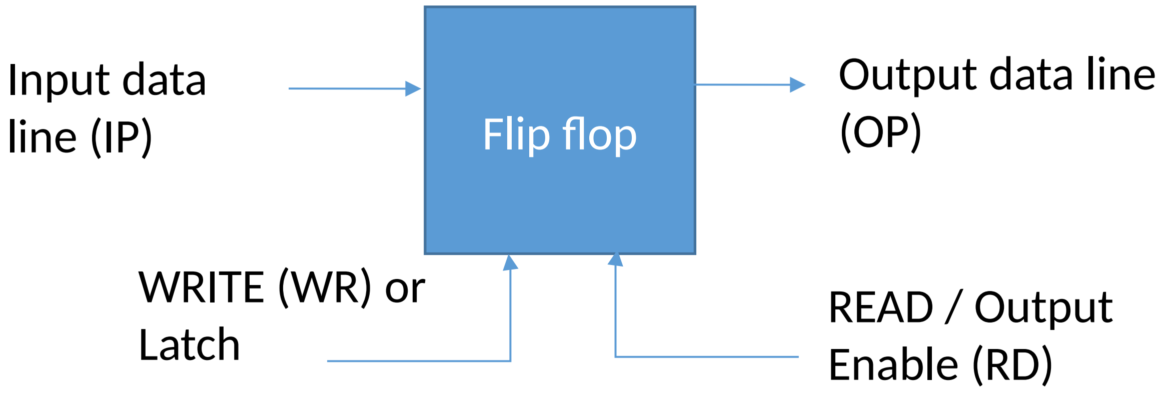
- These gates have an additional input signal called output enable (OE) (also called READ as discussed later)
  - OE allows the output port to assume a additional state: high impedance that effectively disconnects the gate from the output (OUTPUT FLOATS)
  - This allows multiple gates to share the same output line or lines (such as a bus)



Tristate Gates with outputs connected (only one OE enabled at a time)

Flip Flops

- A Flip flop is a sequential circuit that has two stable states and can be used to store state information
- Flip flops are 1 bit memory holders that are equipped with data lines, an output enable (read enable) & a WR enable

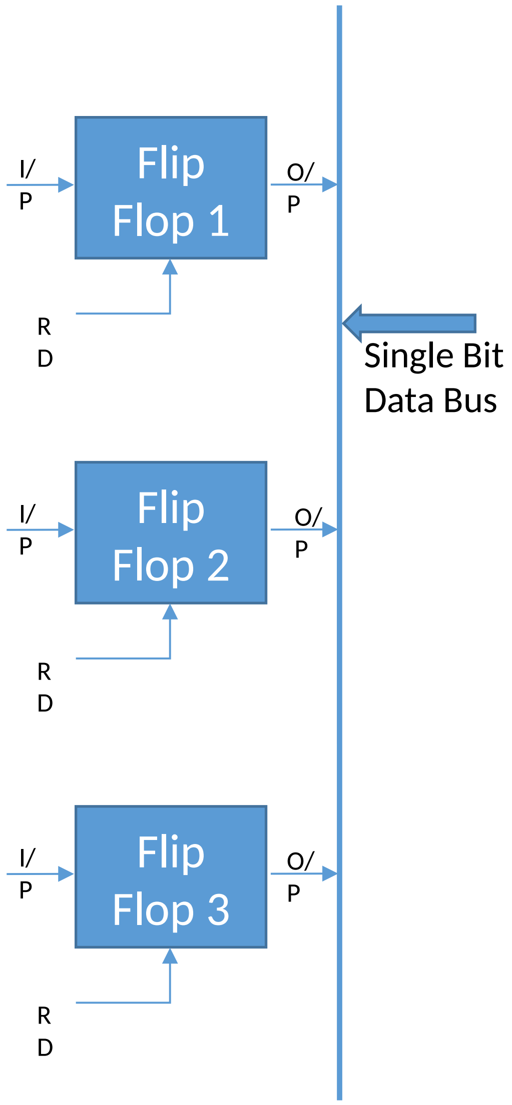


Input data is latched (stored) in flip-flop only, when WR (Latch) is enabled. This is possible in D-flipflop by 'raising edge' of clock pulse. Similarly, output line gives value stored in the flip flop only when RD (Output Enable) is enabled. In D-flipflop output is continuously available. But, to avoid the 'short-circuit' problem of connecting output of multiple GATEs or FFs, O/E switch is installed

Single-bit Data Bus

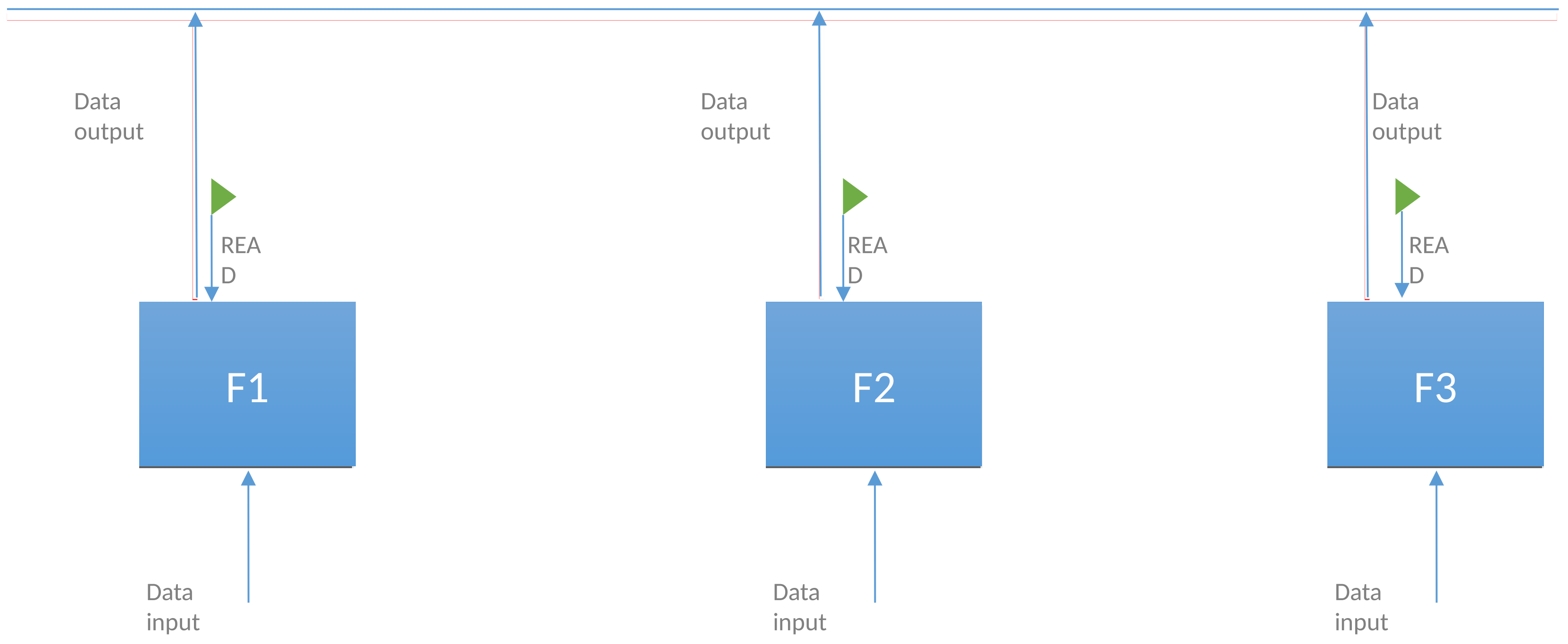
- When OUTPUT of multiple flip flops (single bit storage) is connected, a single bit Data Bus is created
- Only one of flip flops on the bus can have its Output Enable ON at any point of time (others are OFF)
  - The bus has data corresponding to the stored value of o/p enabled flip flop
  - Output enable of a flip flop is also called READ (RD) as enabling it makes data bus read the flip flop data

➢[Intricacy: a single (wire-trace) data bus holds the data o/p from a specific FF (whose O/E 'ON') out of a group of FFs].



Data Movement on Flip flop Output Data Bus

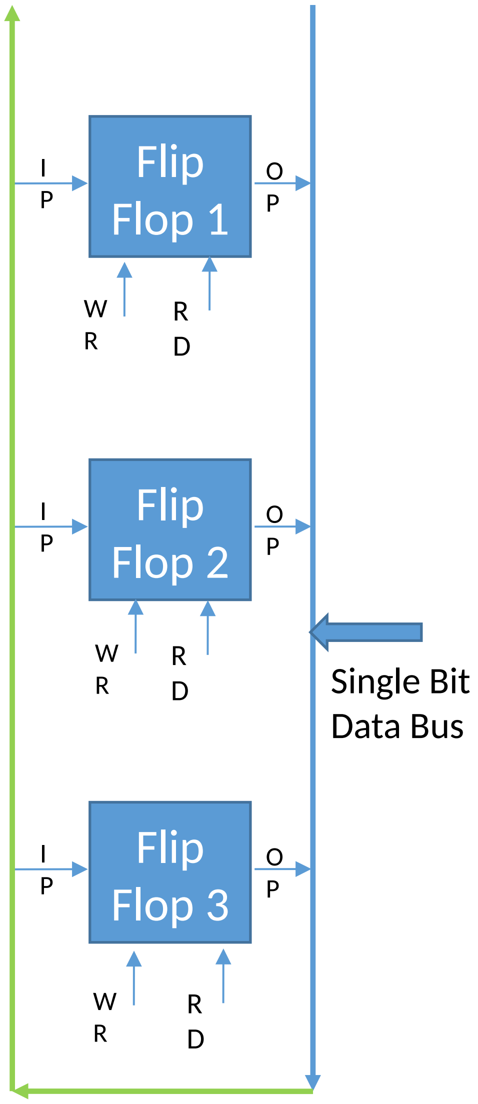
Click on ▶ near each flip flop to read data from that flip flop onto the bus



Reset

Now connecting the Data Bus to INPUTs of the flip flops

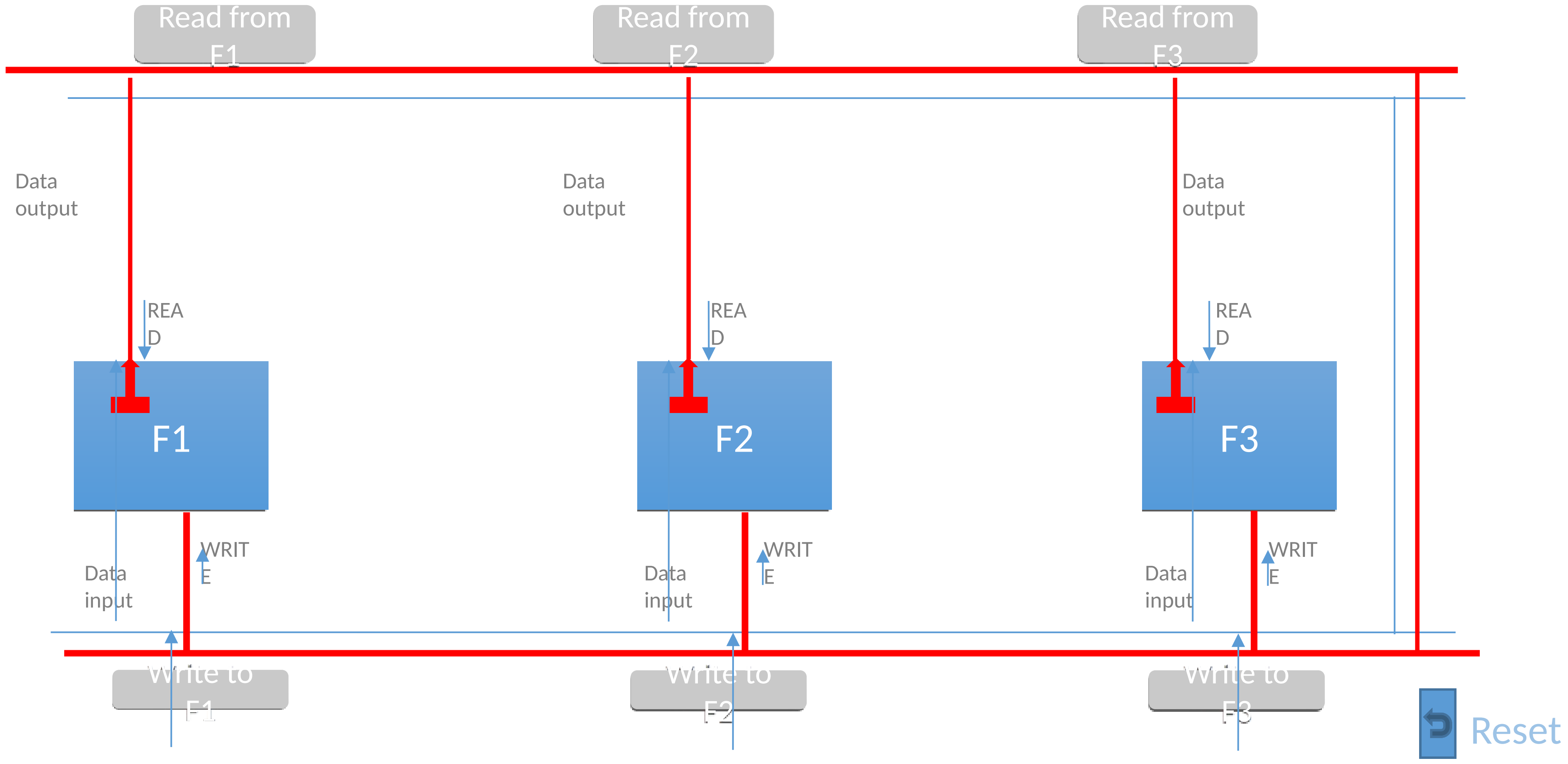
- While RD of a flip flop will enable Data Bus to read the stored value of the flip flop
  - Latch (also called Write or WR as it enables Bus to write to FF) of a flip flop would write (store) the data on Data Bus to this flip flop
- Thus pressing RD of flip-flop 2 and WR of flip-flop 1 will result into transfer of data from flip flop 2 to flip flop 1
  - Two step task involving first flipflop 2 data to be read on the data bus and then written (stored) to flip flop 1
  - MOV FF2-data -> FF1



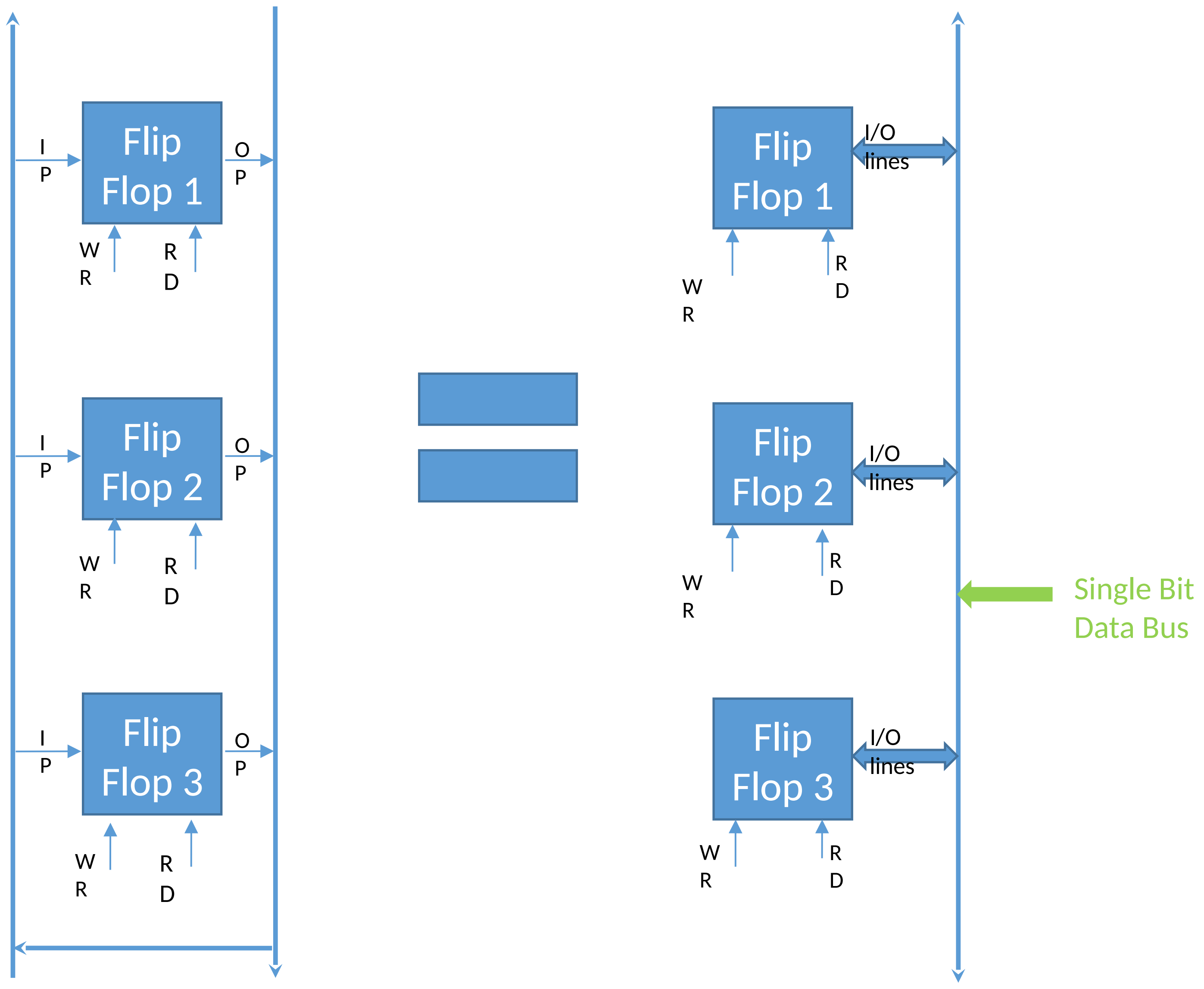
Moving Data between Flip Flops

Using WR/RD input signals to make the data transfer possible

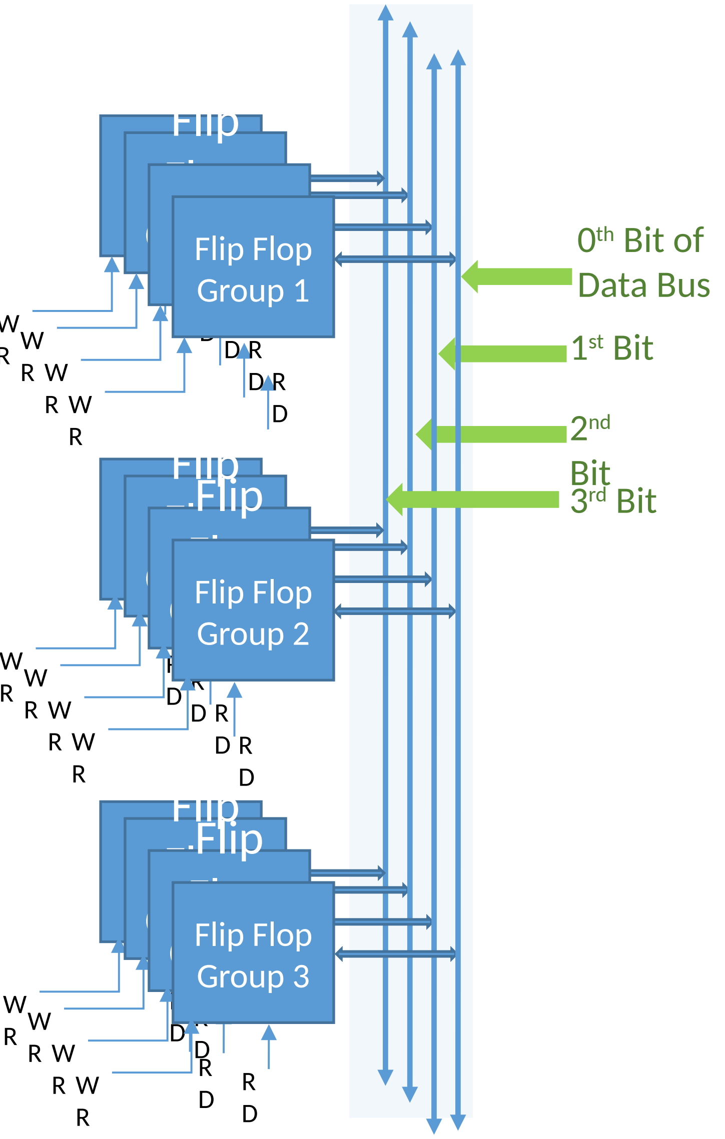
Click first on **Read from buttons** to read data from flip flop onto the bus and then **Write to Buttons** to read the data loaded on the bus into the flip flop



Representing Data bus as bi-directional BUS



Multibit Data Bus

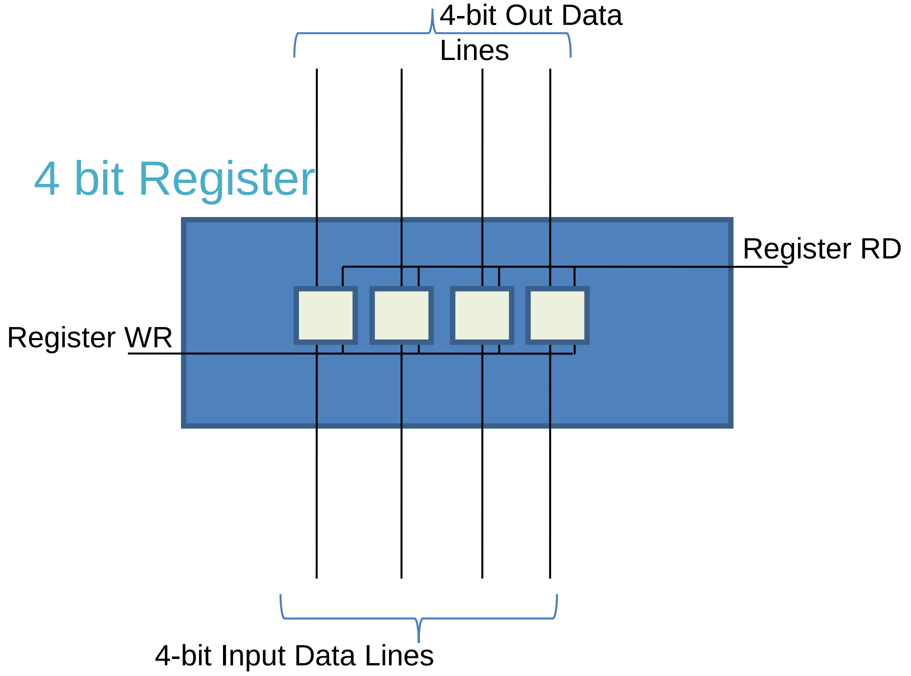


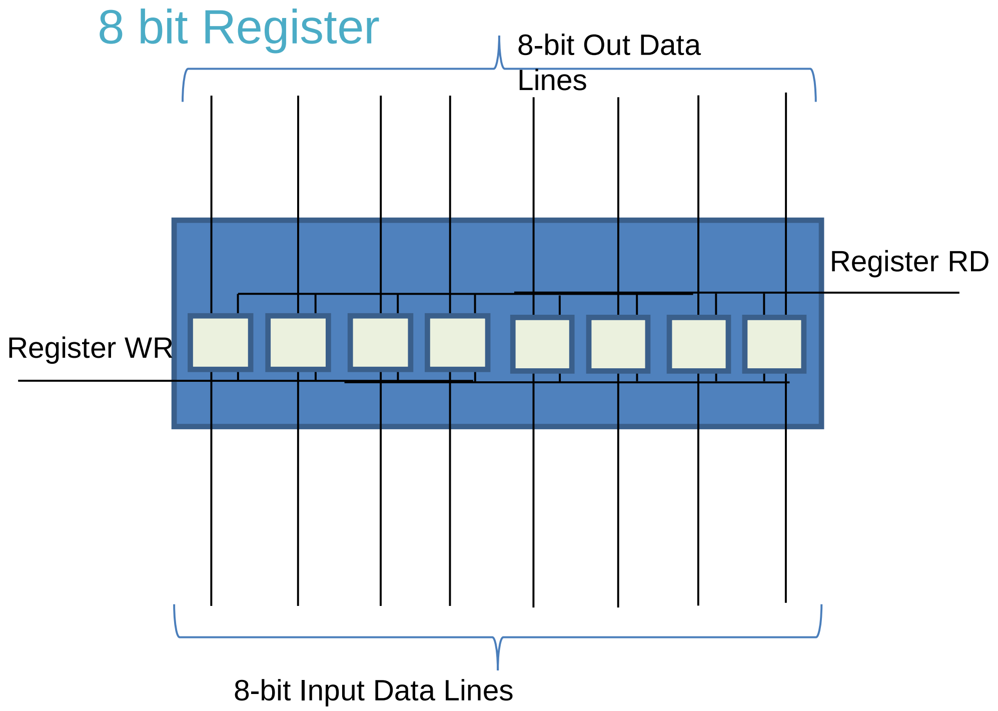
- Flip flops being single bit holders, form a single bit bus when connected to each other enabling single bit data transfer

- What if we want multibit bus to move multiple bits of data in parallel?
  - We can connect flip flops in parallel planes to form a group of flip flops (together they hold a multi-bit word)
  - The data lines of the flip flops contained in the bus can be connected to form a multibit bus

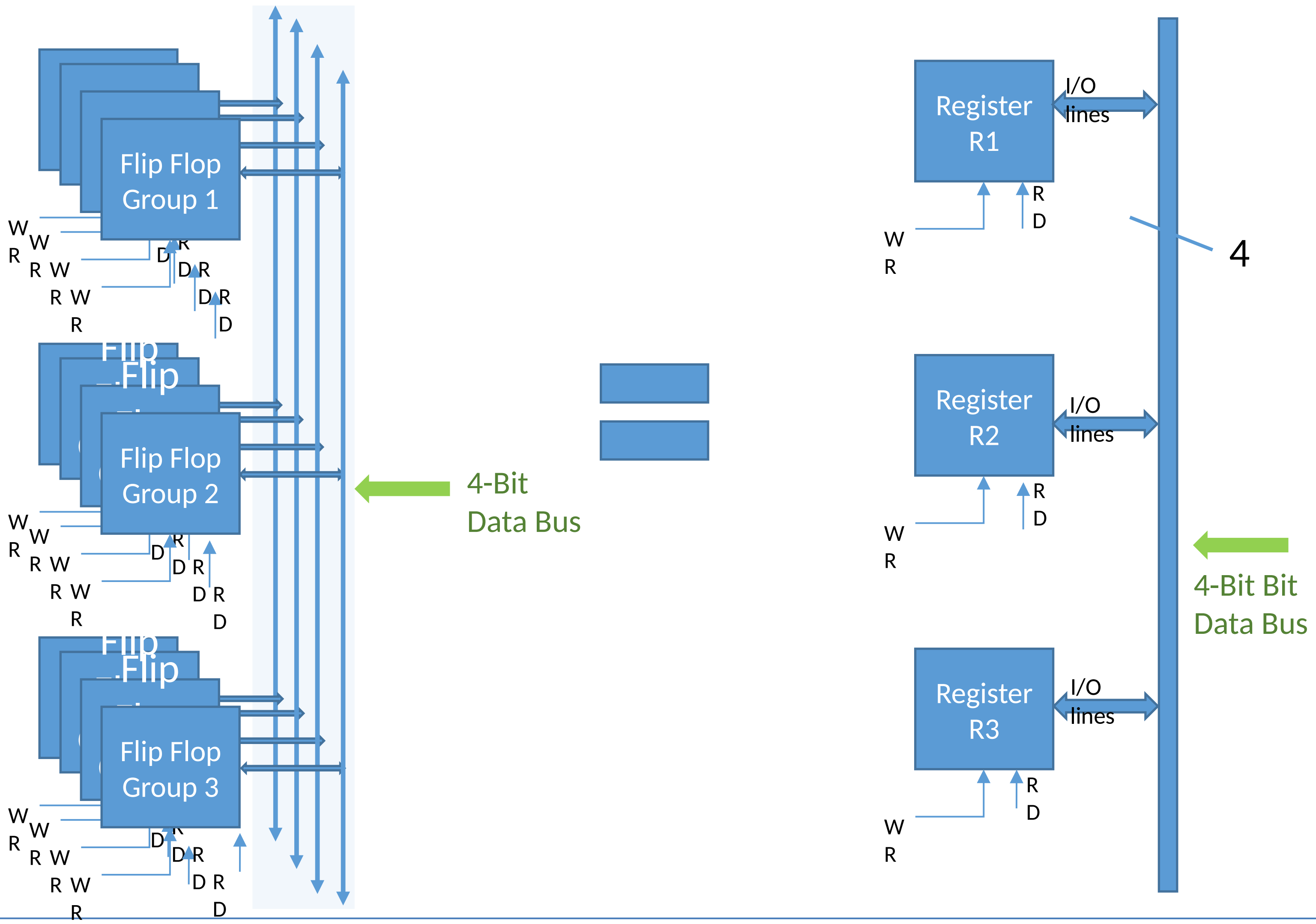
Flip Flops to Registers

- The group of flip flops collectively form a multi bit data (also called data-word) holders: 8/16/32 bit memory are called Registers
- To perform read and write operations from all the flip flops contained in the register, we have a common read enable and a write enable





Multi-bit data bus enabling data transfer between Registers



Your First Assembly Instruction: MOV Instruction

- MOV: instruction in machine / assembly language for data transfer between Registers
  - an operation that forms 80-90% of the operations performed by the processor
- MOV has to be supplied with 2 operands : one for destination and other for source

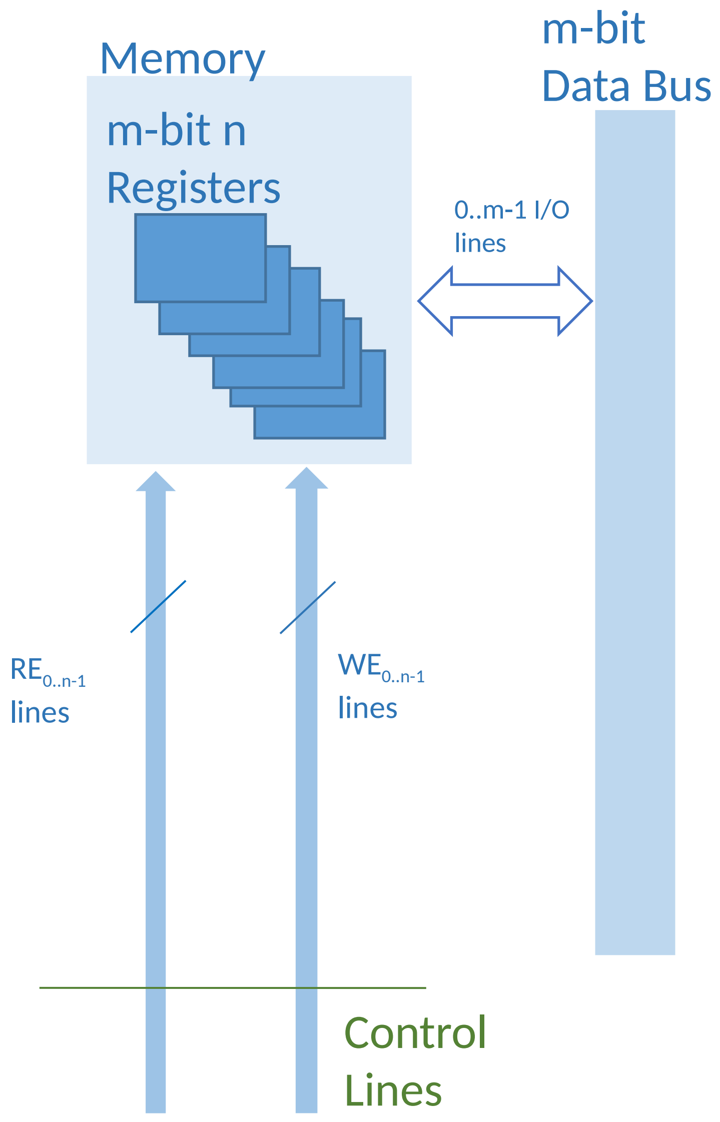
MOV A, B where A is the destination and B is the source

Sample Instruction and its expansion

Instruction	Function	Control
MOV R1, R2	Read from R2	RD2 enable
	Write into R1	WR1 Enable
MOV R20, R15	Read from R15	RD15 Enable
	Write into R20	WR20 Enable

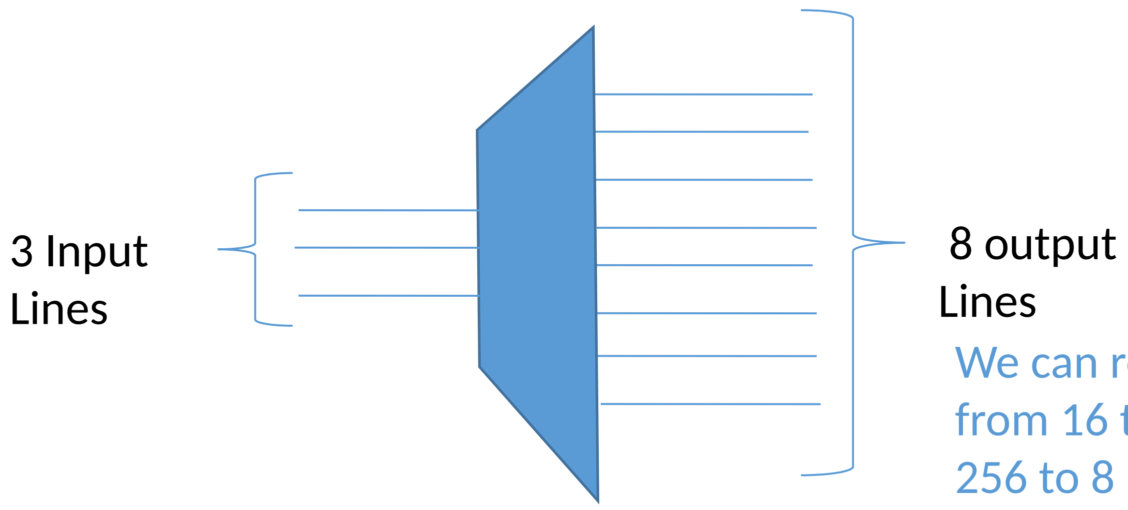
Memory and Data Bus

- Set of registers together form a Memory
- Each of the register in the memory is identified by an index i
  - Registers are named as R0, R1, R2, R3 …… R(n-1)
    - For every register: a WR line and a RD line
  - To enable data transfer each register connected to Data Bus and control the RD and WR lines of these registers to execute the desired transfer
  - Width of the data bus: maximum number of bits a Register can hold or needs to transfer



Using Decoder to Reduce the Address Bus Size

- Can decrease the number of these RD / WR keys by using decoder
  - Example: for 8 lines use a 3:8 decoder and 3 keys in place of 8 keys
-



We can reduce the number of lines from 16 to 4 or from 32 to 5 or from 256 to 8 by using a decoder:
in other words from n lines to log2(n) control

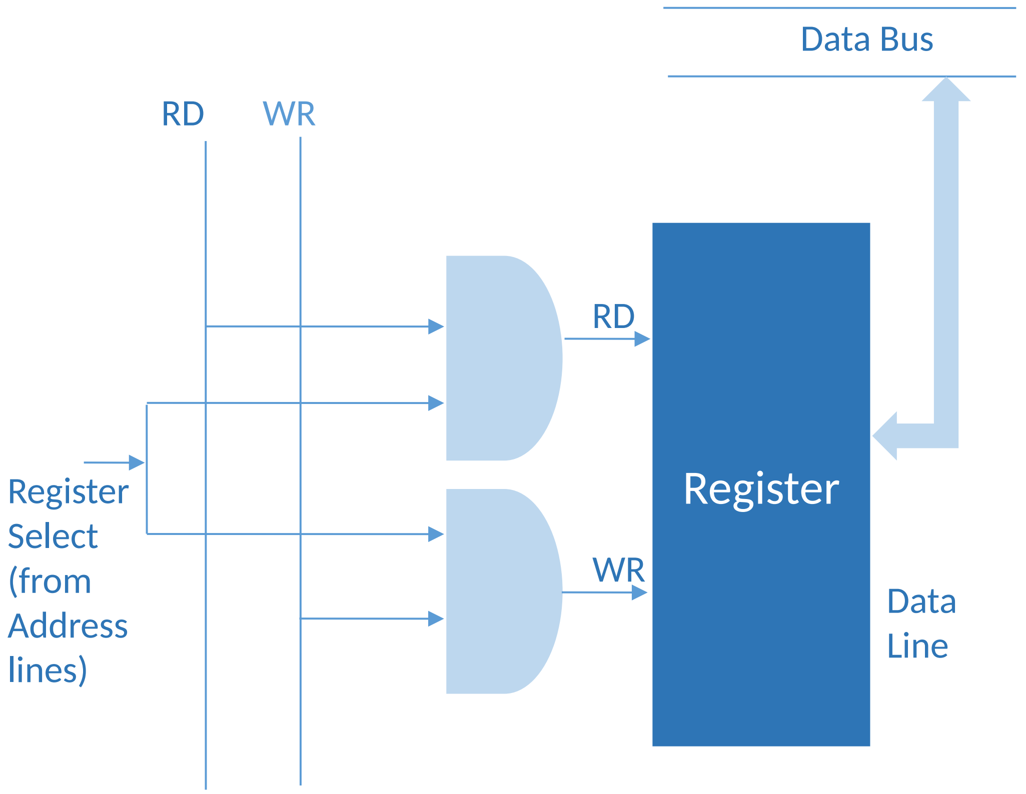
Address Bus

- A specific Register out of n Registers forming a Memory can be selected using log2(n) address lines and an address decoder
    - Address lines together forming **Address Bus**
- Now the function RD or WR for any register can be selected using a common RD and a common WR lines along with Address lines pointing to the register
    - Reduces the number of the control keys
    - Address lines of a Memory enable selection of individual Memory byte/word

Using Address Bus with RD and WR lines

- RD and WR lines: common CONTROL lines shared by all registers
  - A common read line: Enabling this line, puts the Data Bus in Read mode, but with data read from register selected by Address lines
  - A common write line: Use is similar to read line, except it writes the Data from Data Bus into the selected register
- ANDed output of Read line and register select line (obtained from decoding address bus) enables RD of a particular register
  - Similarly, ANDed output of Write line and register select line (obtained from decoding address bus) enables WR of a particular register

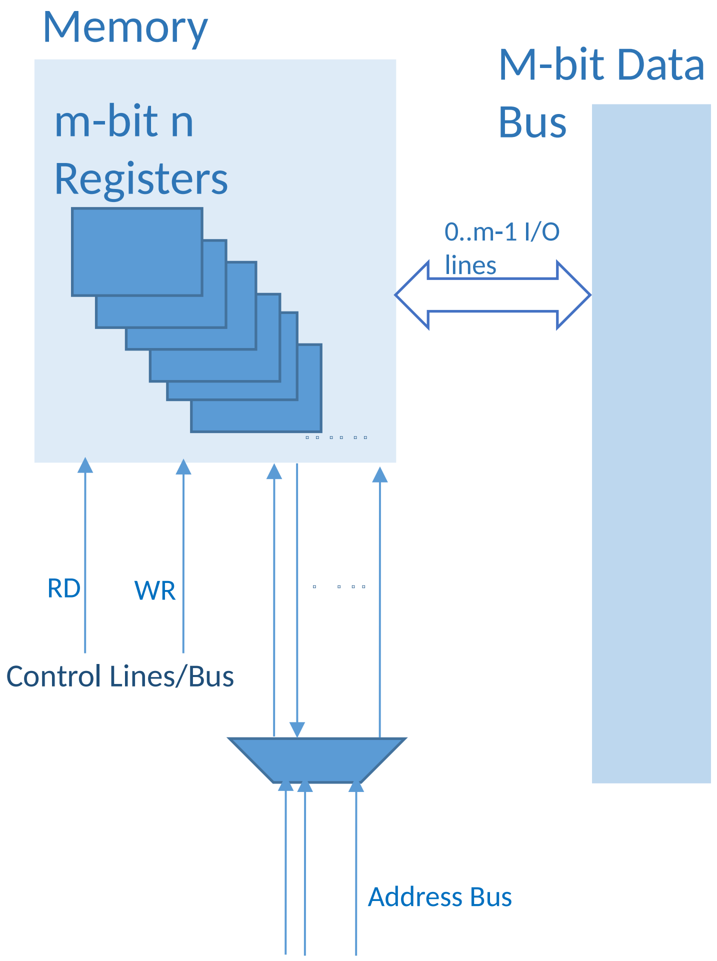
Either RD or WR operation at a time



Register Select	Read Line (RD)	Write Line (WR)	Function
0	X	X	Register virtually dis-connected from the data bus
1	0	1	Data Bus Writing to Register
1	1	0	Data Bus Reading from Register

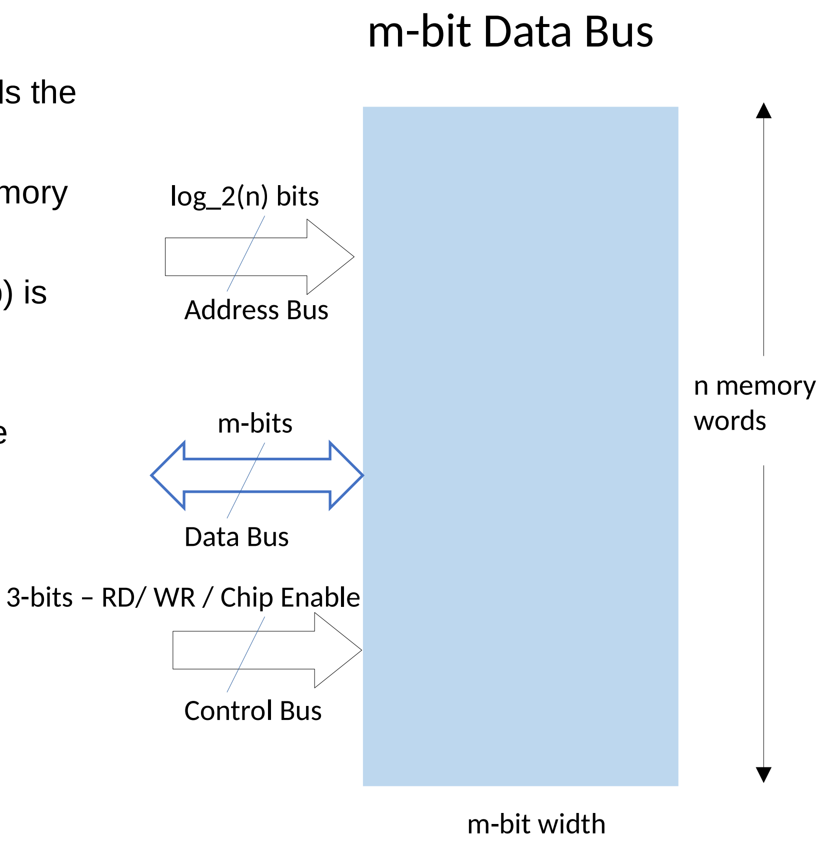
To Sum Up: Memory, Data Bus, Address Bus & Control Bus

- Data Bus
  - m-bit
- Control Bus
  - RD / WR signals
  - Only one operation either RD or WR can happen at a time
- Address bus
  - To select the register (out of n registers) whose content is requested
  - This means we need a n-bit word in which ONLY ONE is ‘1’ and all other bits in that word are ‘0’.
  - The above in turn means, the number of input lines could be further be reduced by using the decoder. Here, for n=8, we need only 3 input address lines
    - Address bus width is 3
    - Internally Address & control signals AND’ed to select that particular register (to write or read)
    -



Model & Convention to Represent Memory: Memory, Data Bus, Address Bus & Control Bus

- Rectangle m X n notation to represent memory block (commercial memory chip)
  - Width denotes the length of the memory word (m-bit, equals the data bus width).
  - Height of the rectangle denotes the total number (n) of memory words stacked together.
  - The size of the memory block (or commercial memory chip) is given by mn bits or (mn)/8 bytes or (mn)/8000 KB or (mn)/8000000 MB & so on.
  - Used to represent primarily the RAM or ROM in this course (secondary HDD are not addressed).
- Data Bus
  - ➢ Width of the rectangle – m-bits
- Control Bus
  - ➢ RD / WR signals
  - ➢ Chip Enable (CE)
  - ➢ Only one operation either RD or WR can happen at a time
- Address bus
  - ➢ Address bus width is log_2 (n)


m-bit Data Bus
log_2(n) bits
Address Bus
m-bits
Data Bus
3-bits – RD/ WR / Chip Enable
Control Bus
n memory words
m-bit width