Computer Architecture:
Register Transfer Language (RTL)

Hossein Asadi (asadi@sharif.edu)

Department of Computer Engineering

Sharif University of Technology

Spring 2024

Copyright Notice

- Some Parts (text & figures) of this Lecture adopted from following:
  - Computer Organization & Design, The Hardware/Software Interface, 3rd Edition, by D. Patterson and J. Hennessey, MK publishing, 2005.
  - "Intro to Computer Architecture" handouts, by Prof. Hoe, CMU, Spring 2009.
  - "Computer Architecture & Engineering" handouts, by Prof. Kubiatowicz, UC Berkeley, Spring 2004.
  - "Intro to Computer Architecture" handouts, by Prof. Hoe, UWisc, Spring 2021.
  - "Computer Arch I" handouts, by Prof. Garzarán, UIUC, Spring 2009.
  - "Intro to Computer Organization" handouts, by Prof. Mahlke & Prof. Narayanasamy, Winter 2008.

Our Lectur Today

Topics Covered Today

- **RTL**

- **Micro-Operation**

- **Register File Design**

Data Movements in Digital Systems

- Digital Systems Consists of
  - Combinational logic
    - AND gate, OR gate, NOT gate, etc
  - Sequential elements
    - FFs, latches

- Question:
  - How we can describe data movements in digital systems?
    - Transistor-level?
    - Gate-level?
      - Too much details and very complicated
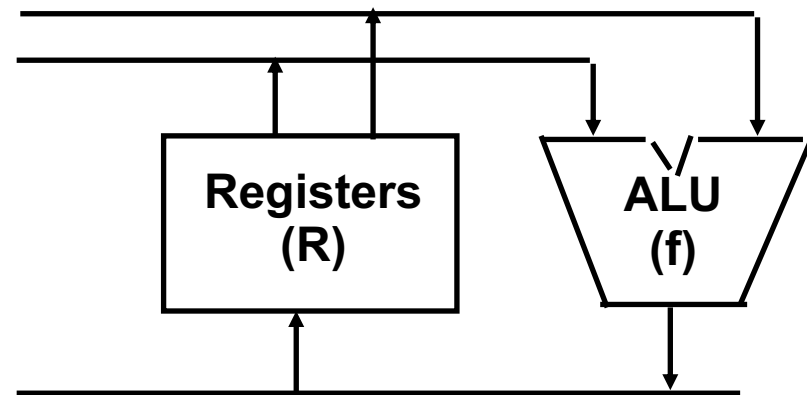
RTL & Micro-Operation

- Data Movement Characterized in terms of:
  - Registers
    - Set of flip-flops/latches
  - Operations done on registers

- Register Transfer Language (RTL)
  - Data movement at register level
  - A language to describe behavior of computers in register flow format

- Micro-operation (Micro-ops or uOps)
  - Functions performed on registers
    - Shift, clear, load, increment, etc.

Micro-Operation

- Definition:

  - An elementary operation performed on information stored in one or more registers

    - During one clock pulse

  - R ⬅ F(R,R)

    - F: shift, load, clear, and, or, xor, etc.

High-Level Description of Computer uArch

- In Terms of RTL & uOps:
  - Set of registers and their functions
  - Microoperations set
    - Set of allowable uops provided by uArch
  - Control signals that initiate sequence of uOps to perform functions

Register Transfer Language

- ## Our Focus in RTL

  - System's registers

  - Data transfers between them

  - Data transformations in them

Designation of Registers

- How Registers Designated?
  - Using capital letters
  - Sometimes followed by numbers
    - e.g., A, R13, IR
- Often names indicate function:
  - PC
    - Program Counter
  - IR
    - Instruction Register
  - MAR
    - Memory Address Register

Designation of Registers (cont.)

- How Registers Represented?
  - Registers and their contents can be viewed and represented in *various ways*
  - A register can be viewed as a single entity:

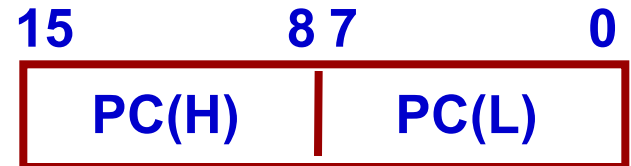 - Registers may also be represented showing bits of data they contain

MAR

Designation of Registers (cont.)

- ## How Registers Represented?
  - Common ways of drawing the block diagram of a register

Showing individual bits

R1

7	6	5	4	3	2	1	0

15   0   15   8 7   0

R2

PC(H)	PC(L)

Numbering of bits   **Subfields**

Data Transfer

- Data Transfer:
  - Copying contents of one reg to another reg

- Example:
  - R2 ← R1
  - Contents of R1 copied (loaded) into R2
  - A simultaneous transfer from R1 to R2
    - All bits transferred at same time on one clock cycle
    - R1: source, R2: destination
    - Data lines from R1 to R2
  - This is a non-destructive
    - Contents of R1 not altered by copying them to R2
  - Control lines to perform action

Data Transfer (cont.)

- ## Controlled (Conditional) Data Transfer
  - Action will occur if a certain condition is true
  - Similar to an "if" statement in SW programming
  - Represented as:

  P: R2 ← R1

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

Controlled Data Transfer
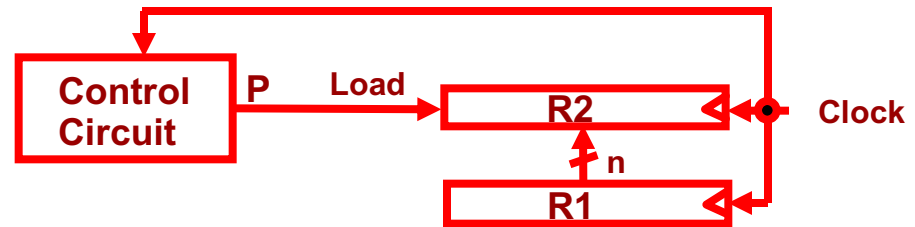
- ## HW Implementation
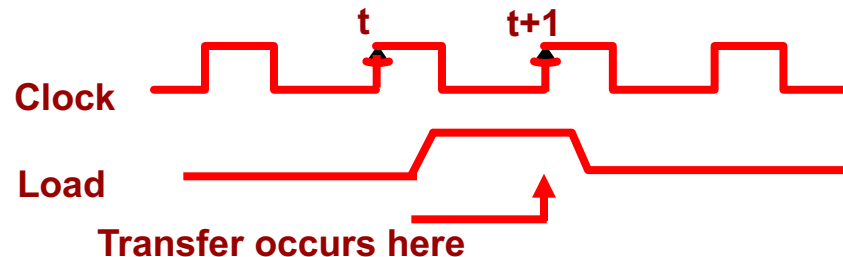
P: R2 ← R1

Block diagram

Control Circuit — **P** — **Load** → R2 ← Clock

R1

n

Timing diagram

Clock

Load

t   t+1

Transfer occurs here

- **Registers assumed to use *positive-edge-triggered* flip-flops**

Data Transfer (cont.)

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

$$P:\ R3 \leftarrow R5,\ MAR \leftarrow IR$$

- ## Here, if control function P = 1 ➔

  - Load contents of R5 into R3
  - At same clock, load contents of IR into MAR

Data Transfer (cont.)

- Practice
  - Draw hardware implementation of following statement
  - P: R1 $\leftarrow$ R2, R2 $\leftarrow$ R1

Blocking and Non-Blocking Statements in RTL

- ## Non-Blocking Statements
  - Statement scheduled and executed together with other non-blocking assignments
  - Example in Verilog
    - A <= B
    - B <= C

- ## Blocking Statements
  - Statement executed sequentially
  - Example in Verilog
    - A = B
    - B = C

Practice

- Code1
  - R1 ← R2, R2 ← R1

- Code2
  - R1 ← R2
  - R2 ← R1

- Code3
  - R1 <= R2
  - R2 <= R1

- Question:
  - New values of R1 and R2?

R1

20

R2

30

Common Micro-Ops

- **Transfer** (R0 ← R1)
  - Transfers data from a reg to another reg
- **Arithmetic** (R0 ← R1 + R2)
  - Performs arithmetic on data in registers
- **Logic/Bit Manipulation**
  - Performs bit (Boolean) operations on data
    R0 ← R1 & R2 ; or R0 ← R1 | R2
- **Shift**
  - Shift data in regs by n bits positions
    R0 ← R1 << 3; or R0 ← R2 >> 2

RTL & MIPS Assembly

MIPS Code	RTL Code
add $t0, $s2, $s4	$t0 ← $s2 + $s4
and r1, r2, r3	r1 ← r2 ^ r3
lw $s1, 100($s2)	$s1 ← Mem[$s2 + 100]
sw $s1, 100($s2)	Mem[$s2 + 100] ← $s1

Summary of Reg Transfer & Micro-Ops

A ← B	Transfer content of reg. B into reg. A
AR ← DR(AD)	Transfer content of AD portion of reg. DR into reg. AR
A ← constant	Transfer a binary constant into reg. A
ABUS ← R1,	Transfer content of R1 into bus A and, at the same time,
R2 ← ABUS	Transfer content of bus A into R2
AR	Address register
DR	Data register
M[R]	Memory word specified by reg. R
M	Equivalent to M[AR]
DR ← M	Memory *read* operation: transfers content of memory word specified by AR into DR
M ← DR	Memory *write* operation: transfers content of DR into memory word specified by AR

Summary of Reg Transfer & Micro-Ops (cont.)

- ## GPR[rs]
  - General Purpose Registers
  - A register from Register File indicated by index rs

- ## RF[rs]
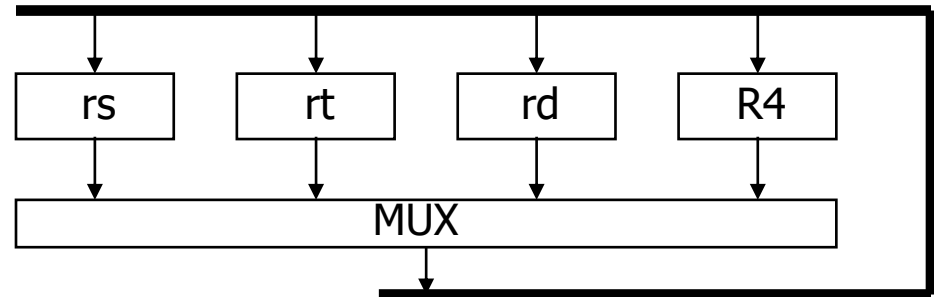 - Register File

Register Transfers: Interconnect

- ## Point-to-Point Connection
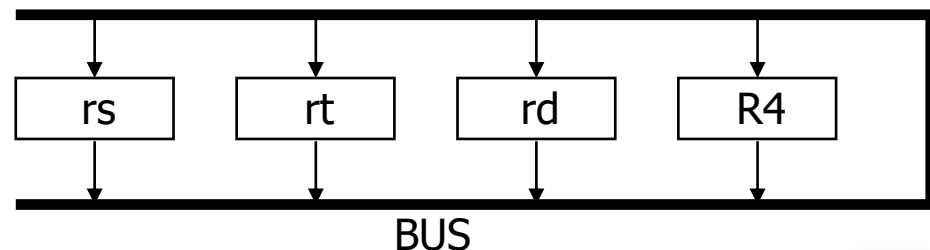 - – Dedicated wires
  - – Muxes on inputs of each register
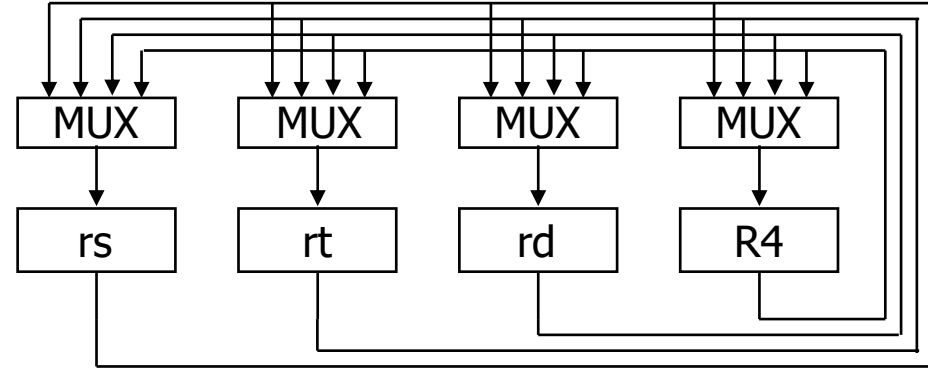
- ## Common Input from mux
  - – Load enables for each register
  - – Control signals for multiplexer

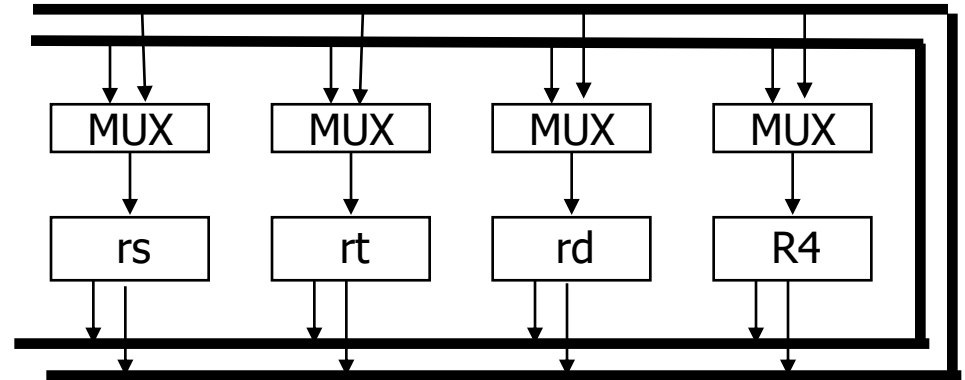- ## Common Bus with Output Enables
  - – Output enables and load enables for each register

Register Transfer: Multiple Busses

- One transfer per bus

- Each set of wires can carry one value

- State Elements
  - Registers
  - Register files
  - Memory

- Combinational Elements
 - Busses
  - ALUs
  - Memory (read)

Connecting Registers

- How to Connect 32 Registers to each other?
  - Impractical to have data and control lines to directly allow each register to be loaded with contents of every possible other registers

- To completely connect n registers
  - $\rightarrow$ n(n-1) lines
  - O(n^2) cost
  - Not a realistic approach to use in a large system

Connecting Registers

- ## Point-to-Point Connection
  - Dedicated wires
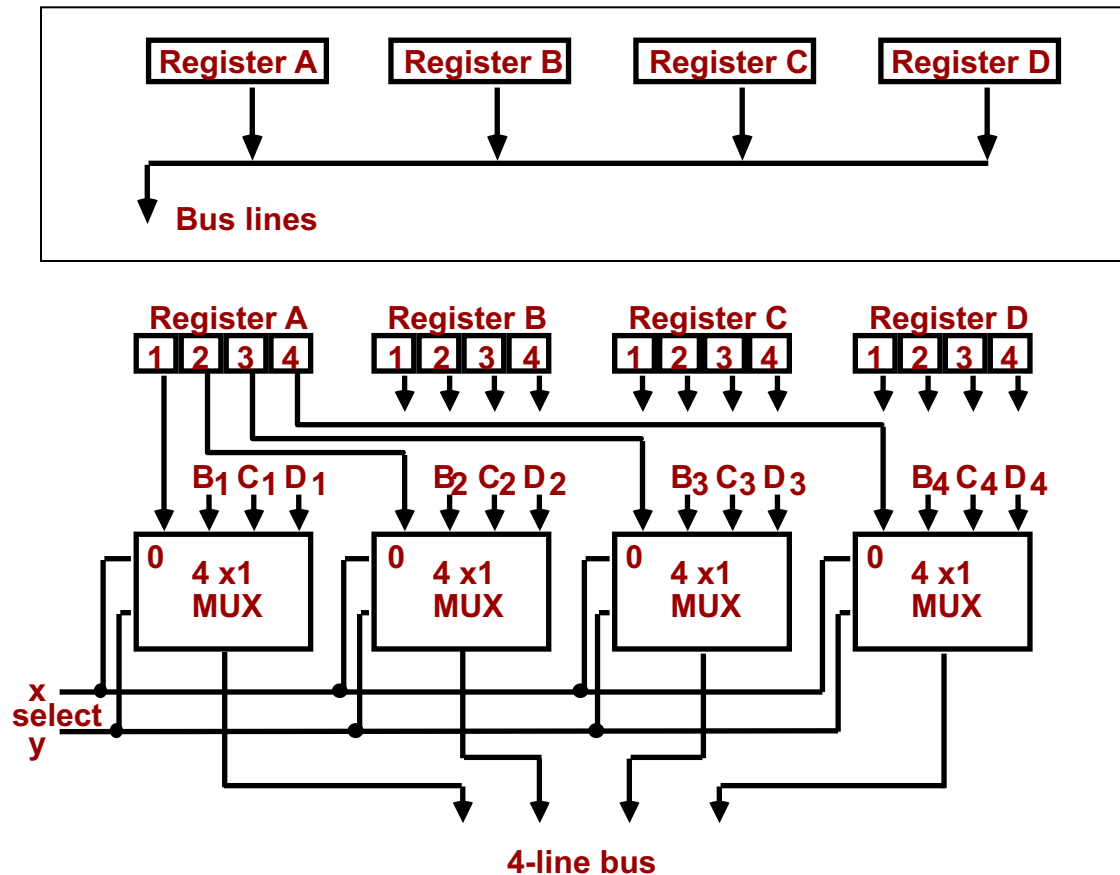 - Muxes on inputs of each register
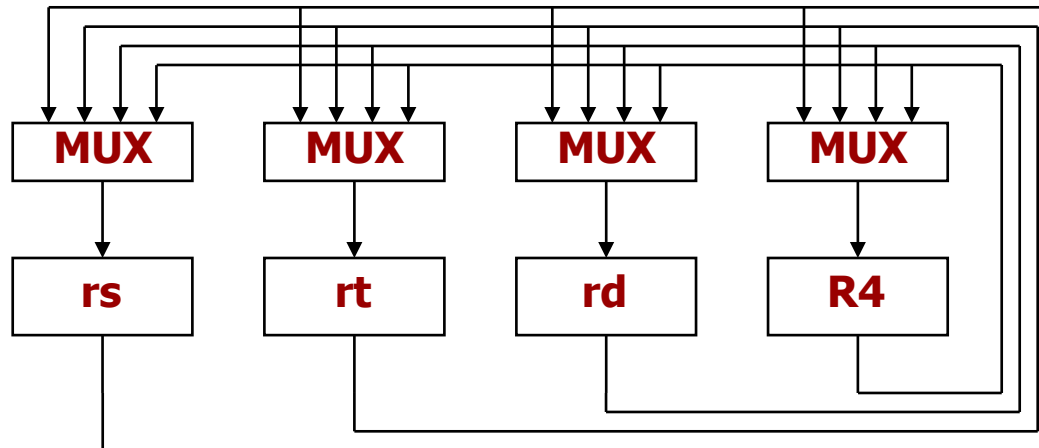
Connecting Registers (cont.)

- **Instead, Take a Different Approach**
  - One centralized set of circuits for data transfer, called bus
  - Have control circuits to select:
    - Which reg as source
    - Which reg as destination
- **Bus Definition:**
  - A path (of a group of wires) over which information is transferred, from any of several sources to any of several destinations
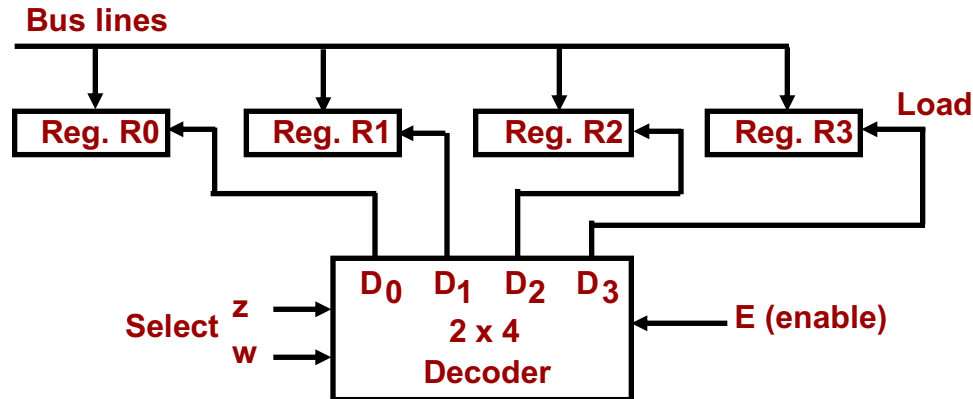 - A group of wires with multiple drivers
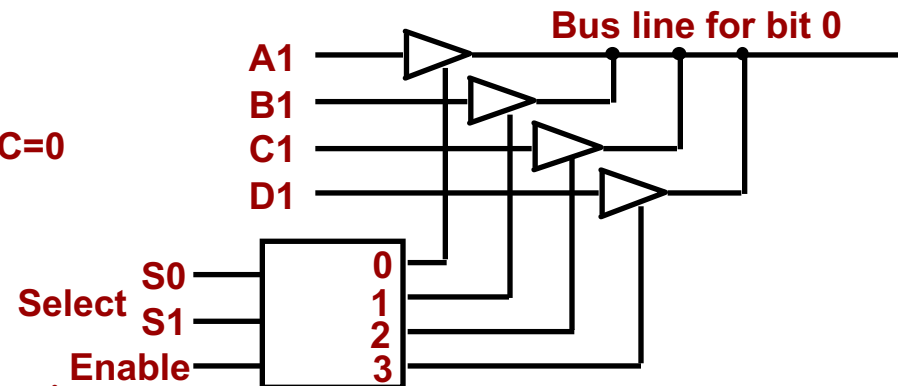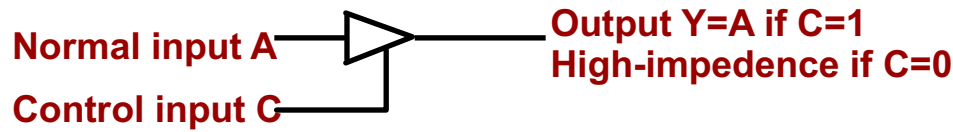
Bus Transfer

- From a register to bus: BUS ← R

Bus Transfer: Bus to Destination Reg

Bus lines

Reg. R0   **Reg. R1**   **Reg. R2**   **Reg. R3**   **Load**

D_0 D_1 D_2 D_3

Select   **z**

w

2 x 4
Decoder

E (enable)

Three-State Bus Buffers

Normal input A

Control input C

Output Y=A if C=1
High-impedence if C=0

Bus line for bit 0

A1
B1
C1
D1

Select **S0**
S1
Enable

0
1
2
3

3-State Buf vs Mux implementation
Pros: area efficient & more scalable
Cons: probably more power consumption (leakage current in high-Z)

Bus vs. Point-To-Point

- Pros (Advantages)
  - Much less routing
  - Less area
  - Easy to scale

- Cons (Disadvantages)
 - Only one data transfer per clock cycle

Memory & Memory Transfer

- ## Memory (RAM)
  - Can be thought as a sequential circuit containing some number of registers

- ## Assume RAM with r = 2^k Words
  - n data input lines
  - n data output lines
  - k address lines
  - A read control line
  - A write control line

Data Input Lines

n

Address Lines

k

Read

Write

RAM unit

n

Data Output Lines
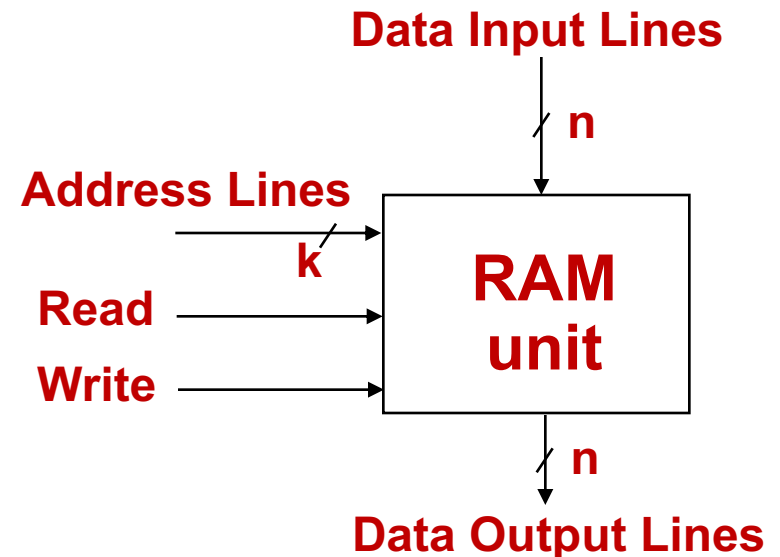
Memory & Memory Transfer (cont.)

- ## Memory Transfer
  - Memory viewed at register level as a device (M)
  - Should specify which address in memory
    - Since it contains multiple locations

- ## Memory Accesses
  - Provide target address in a special register
    - Memory Address Register (MAR, or AR)
  - Contents of MAR sent to memory address lines
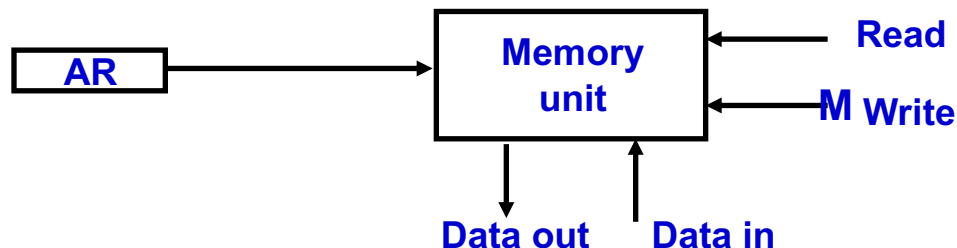
Memory & Memory Transfer (cont.)

- ## Memory Read
  - To read a value from a location in memory and load it into a register
  - RTL notation looks like this:

$$\text{MAR} \leftarrow \text{Address}$$
$$\text{R1} \leftarrow \text{M[MAR]}$$

Memory & Memory Transfer (cont.)

- **Memory Write**
  - To write a value from a reg to a location in memory
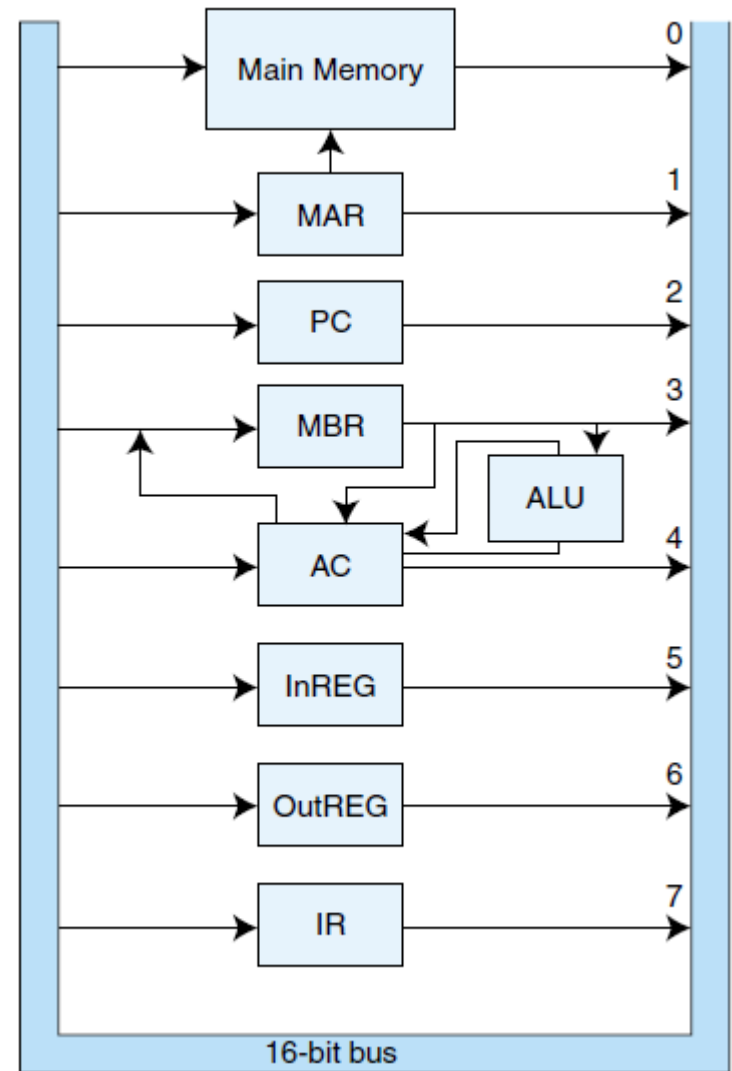 - RTL notation looks like this:

MAR ← address

M[MAR] ← R1

RTL Example

- **Memory Buffer Register**
  - MBR

- **Accumulator**
  - AC

- **Program Counter**
  - PC
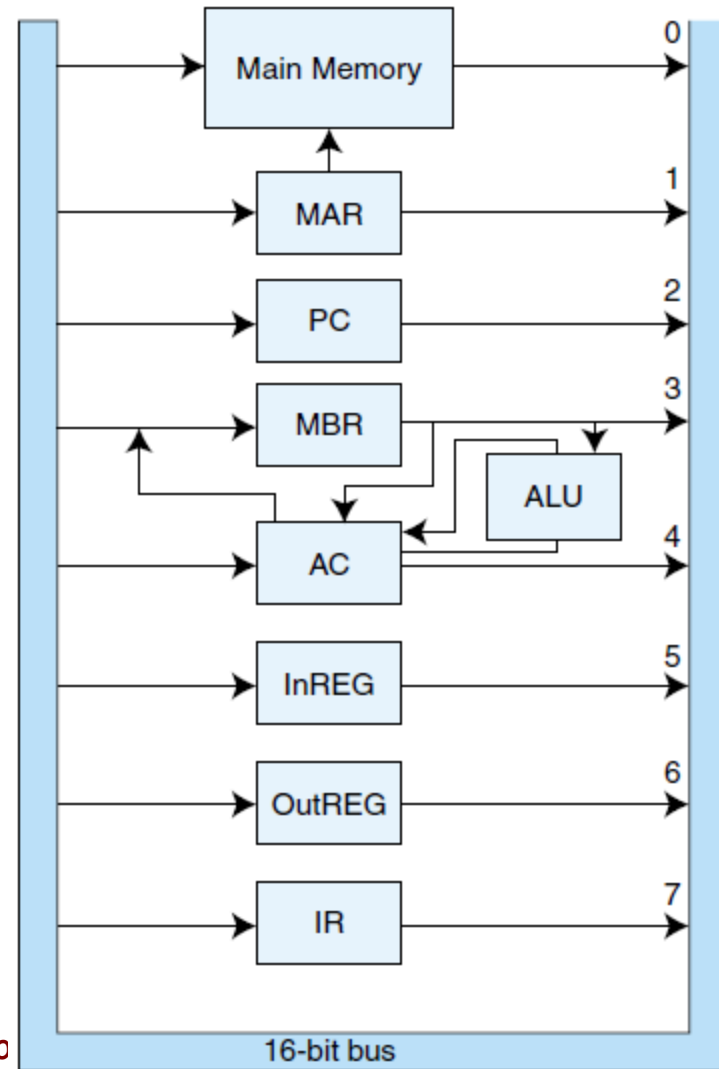
- **Memory Address Register**
  - MAR

- **Instruction Register**
  - IR

RTL Example (cont.)

- ## Store X
  - MAR ← X, MBR ← AC
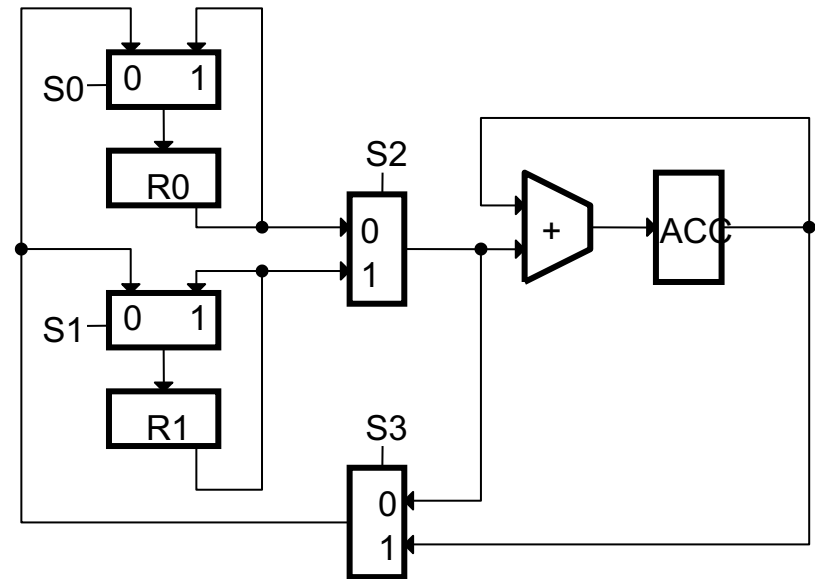 - M[MAR] ← MBR
- ## Add X
  - MAR ← X
  - MBR ← M[MAR]
  - AC ← AC + MBR

Practice

- What is RTL description of this circuit?
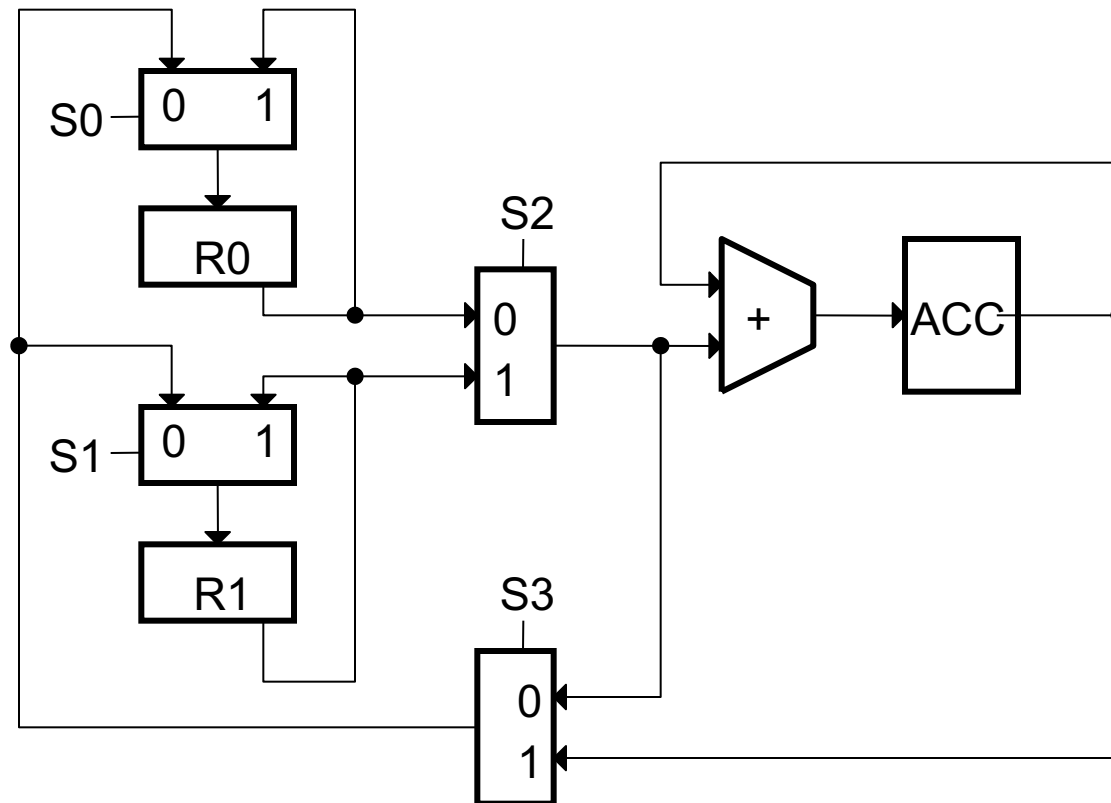
Practice (cont.)

- In Each Clock Cycle:
  - ~S3.~S2.~S1: R1 ← R0
  - ~S3.S2.~S0: R0 ← R1
  - What else?
    - R1 ← ACC
    - ...

- Or use "if-then-else"
  - If (S3=1 and S1=0) then ...

Thanks for Your Attention!