MODULE 4: Computer Organization and MARIE

# Lecture 4.2
# MARIE

Prepared By:

- Scott F. Midkiff, PhD
- Luiz A. DaSilva, PhD
- Kendall E. Giles, PhD

Electrical and Computer Engineering
Virginia Tech

# Lecture 4.2 Objectives

- Describe the components of the MARIE instruction set architecture
- Identify the assembly language instructions supported by MARIE
- Express the micro operations that compose each MARIE instruction using register transfer notation
- Assemble and disassemble a MARIE program

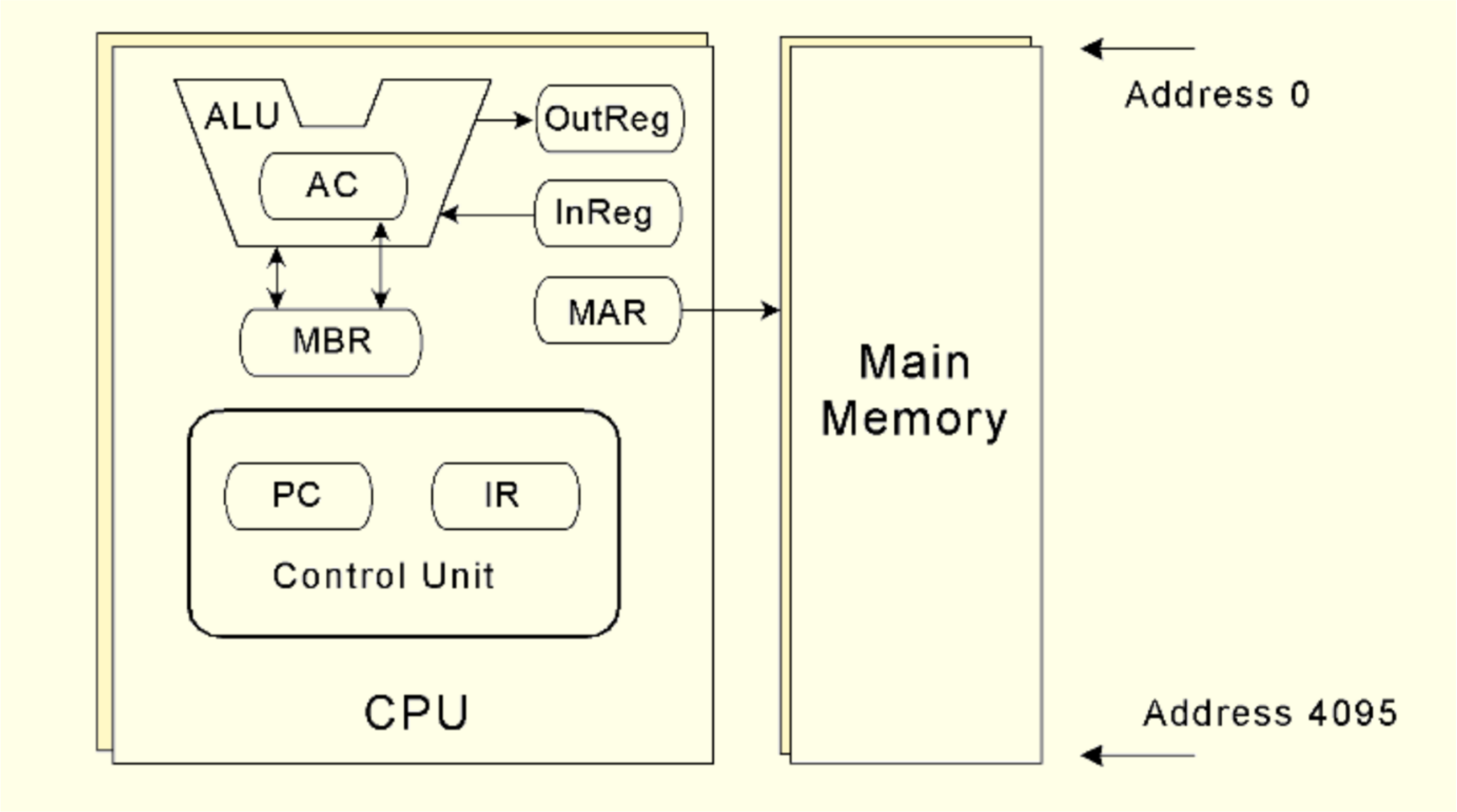
# Instruction Set Architecture

- The instruction set architecture (ISA) defines the view of the processor as seen by the assembly language or machine language programmer
  - Considers registers and memory, instructions, and data
  - The ISA is processor-specific
- The ISA may have different implementations
  - These different implementations are transparent to the assembly language programmer, except perhaps with respect to performance
  - An ISA may be extended across a processor family to provide full compatibility or evolve across a processor family to ensure backward compatibility

# MARIE

- MARIE is a made-up instruction set architecture used for educational purposes
  - Machine Architecture that is Really Intuitive and Easy
- MARIE is a 16-bit processor (it can directly manipulate 16-bit words) with a word-addressable memory space
  - 4K words of memory (12-bit addresses)
- A MARIE simulator is available online
  - Allows us to assemble and execute programs written for MARIE

# MARIE Architecture



# Registers (1)

- Accumulator (AC) – 16 bits
  - General-purpose register to hold data that the CPU needs to process
  - Most real instruction set architectures support multiple general-purpose registers
- Memory Address Register (MAR) – 12 bits
  - Holds memory address of data being referenced
- Memory Buffer Register (MBR) – 16 bits
  - Holds the data just read from memory or the data to be written to memory

# Registers (2)

- Program Counter (PC) – 12 bits
  - Holds the address of the next instruction to be executed
- Instruction Register (IR) – 16 bits
  - Holds the next instruction to be executed
- Input Register (InREG) – 8 bits
  - Holds data from the input device
- Output Register (OutREG) – 8 bits
  - Holds data for the output device
- Status or flag register
  - Holds flags indicating various conditions, such as an overflow in the ALU

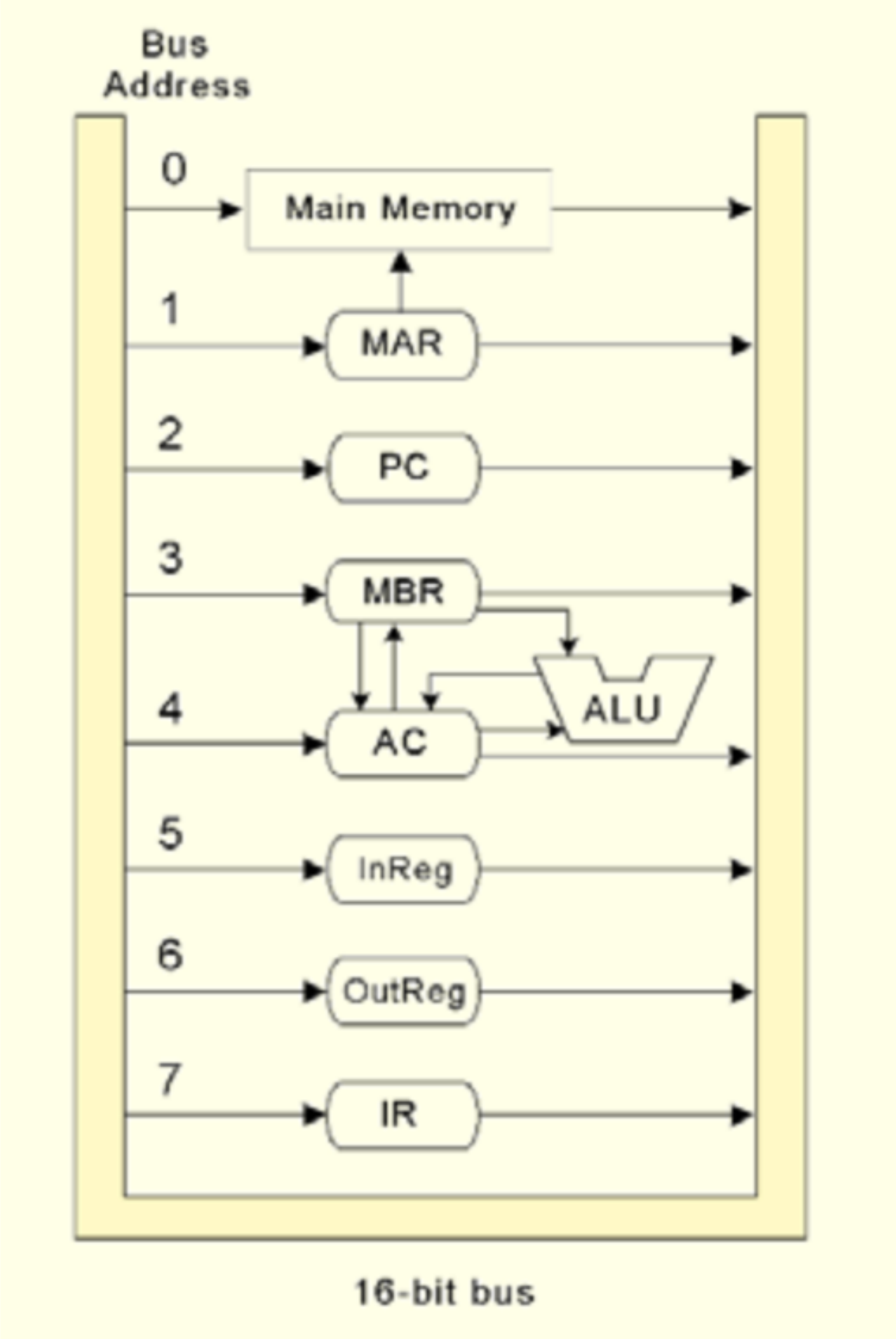
# CHECK POINT

As a checkpoint of your understanding, please pause the video and make sure you can do the following:

- Describe the purpose of the registers in the MARIE architecture shown in the diagram on slide 5.

If you have any difficulties, please review the lecture video before continuing.

# System Bus

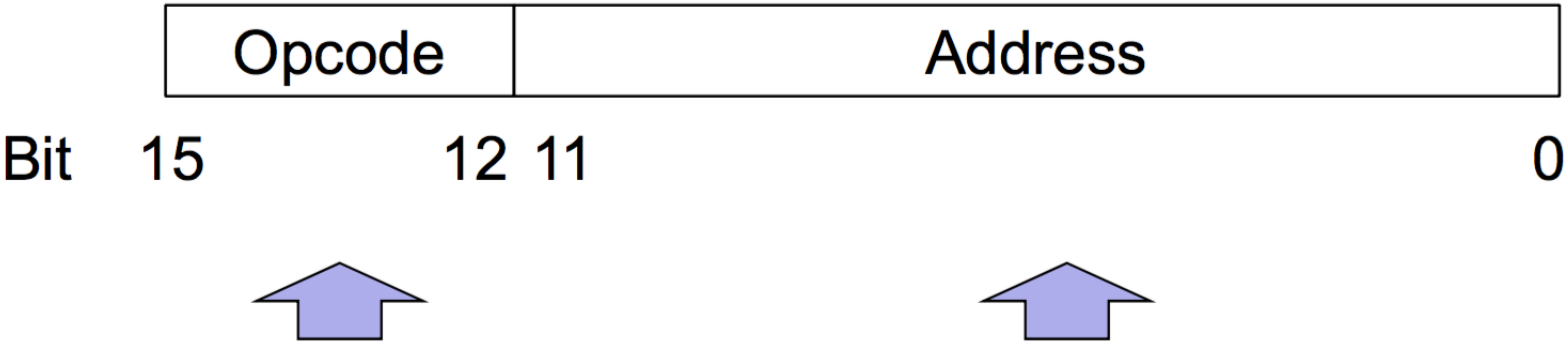



- Each device on the bus is identified by a unique number
  - set on the control lines whenever that device is required to carry out an operation
- Separate connections provided between the AC and the MBR, and between those and the ALU
  - Transfers among those do not require use of the bus

# Assembly Language Instructions

- A MARIE assembly language program is a sequence of assembly language instructions
    - Instructions that are supported by the MARIE ISA, e.g. Load X
- Assembling refers to translating these mnemonic assembly language instructions into their binary equivalents, called machine instructions
- De-assembling refers to interpreting a sequence of machine instructions into assembly language instructions

# Instruction Format




$2^4$ possible instructions

$2^{12}$ possible addresses

# Instructions (1)

- Load X (opcode: 0001)
  - Load the contents of memory address X into the AC (via the MBR)
- Store X (opcode: 0010)
  - Store the contents of the AC into memory address X
- Add X (opcode: 0011)
  - Add the contents of memory address X to the contents of the AC, storing the result in the AC
- Subt X (opcode: 0100)
  - Subtract the contents of memory address X from the contents of the AC, storing the result in the AC

# Instructions (2)

- Input (opcode: 0101)
  - Input a value (in decimal) from the keyboard into the AC
- Output (opcode: 0110)
  - Output the value in the AC to the display (in decimal)
- Halt (opcode: 0111)
  - Terminate the program
- Skipcond (opcode: 1000)
  - Skip the next instruction depending on bit positions 10 and 11
- Jump X (opcode: 1001)
  - Load the value of X into the PC

# CHECK POINT

As a checkpoint of your understanding, please pause the video and make sure you can do the following:

- Describe the function of the MARIE instructions listed on slides 12 and 13.

If you have any difficulties, please review the lecture video before continuing.

# Register Transfer Language

- Each of our instructions actually consists of a sequence of smaller instructions called microoperations
- The exact sequence of microoperations that are carried out by an instruction can be specified using register transfer language (RTL)
- In the MARIE RTL, we use the following notation:
    - M[X] indicates the actual data value stored in memory location X
    - ← indicates the transfer of bytes to a register or memory location

# Load X

| 0 0 0 1 | X |
|---|---|

Bit 15 12 11 0

$$\text{MAR} \leftarrow \text{X}$$
$$\text{MBR} \leftarrow \text{M[MAR]}$$
$$\text{AC} \leftarrow \text{MBR}$$

- Load X into the MAR
- Load the contents of the memory location pointed at by the MAR into the MBR
- Transfer the content of the MBR into the AC

# Store X

| 0 0 1 0 | X |
|---|---|
| Bit 15 … 12 | 11 … 0 |

- Load X into the MAR and the contents of the AC into the MBR
- Store the value in the MBR into the memory location pointed at by the MAR

$$MAR \leftarrow X,\ MBR \leftarrow AC$$
$$M[MAR] \leftarrow MBR$$

# Add X

| 0 0 1 1 | X |
|---|---|
| Bit 15 — 12 | 11 — 0 |

$$\boxed{\begin{array}{l} MAR \leftarrow X \\ MBR \leftarrow M[MAR] \\ AC \leftarrow AC + MBR \end{array}}$$

- Load X into the MAR
- Load the contents of the memory location pointed at by the MAR into the MBR
- Add the value in the MBR to the value in the AC, store the result in the AC

# Subt X

| Bit 15 – 12 | Bit 11 – 0 |
|---|---|
| 0 1 0 0 | X |

MAR ← X
MBR ← M[MAR]
AC ← AC - MBR

- Load X into the MAR
- Load the contents of the memory location pointed at by the MAR into the MBR
- Subtract the value in the MBR from the value in the AC, store the result in the AC

# CHECK POINT

As a checkpoint of your understanding, please pause the video and make sure you can do the following:

- Write the RTL micro operations for the MARIE ADD X instruction.

If you have any difficulties, please review the lecture video before continuing.

# Input

| Bit | 15 … 12 | 11 … 0 |
|---|---|---|
| | 0 1 0 1 | 00…0 |

$AC \leftarrow InREG$

- Value from the input device is stored in the InREG
- Load the value in the InREG into the AC

# Output

| Bit | 15 … 12 | 11 … 0 |
|---|---|---|
| | 0 1 1 0 | 00…0 |

$$\boxed{\text{OutREG} \leftarrow \text{AC}}$$

- Copy the value in the AC onto the OutREG
- Value in the OutREG is sent to the output device

# Halt

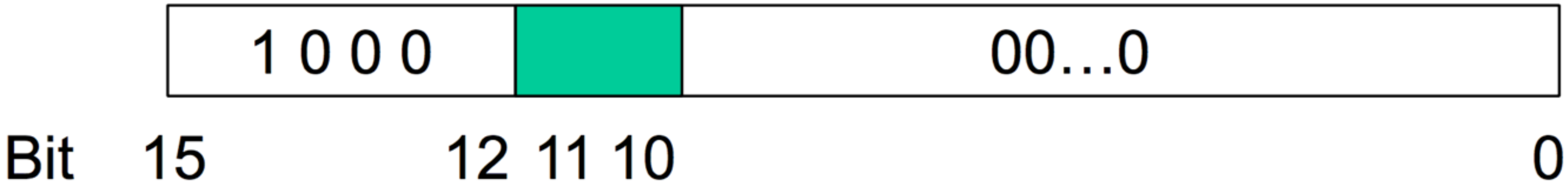
| 0 1 1 1 | 00…0 |
|---|---|

Bit 15 12 11 0

- Machine ceases execution of the program

NO-OP

# Skipcond




```
If IR[11-10] = 00 then
        If AC < 0 then PC ← PC+1
Else If IR[11-10] = 01 then
        If AC = 0 then PC ← PC + 1
Else If IR[11-10] = 10 then
        If AC > 0 then PC ← PC + 1
Else If IR[11-10] = 11 then
        Error condition
```

- Branch if AC is:
  - Negative
    (IR[11-10] = 00)
  - Zero
    (IR[11-10] = 01)
  - Positive
    (IR[11-10] = 10)

# Jump X

| 1001 | X |
|---|---|

Bit 15 12 11 0

$\boxed{PC \leftarrow X}$

- Load the value in the 12 least significant bits in the IR onto the PC
- Causes an unconditional branch

# De-assembling Example

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| Bit | 15 | | | 12 | 11 | | | | | | | | | | | 0 |

Instruction: Add 0xD

# CHECK POINT

As a checkpoint of your understanding, please pause the video and make sure you can do the following:

- By hand, work through the de-assembly example on slide 26.

If you have any difficulties, please review the lecture video before continuing.

# Summary

- The instruction set architecture (ISA) defines the view of the processor as seen by the assembly language or machine language programmer
- MARIE is a (fictitious) 16-bit processor supporting a 4K word memory space, with a very limited instruction set
- Assembly language instructions supported by MARIE include Load X, Store X, Add X, Subtract X, Input, Output, Halt, Skipcond, and Jump X

## MODULE 4: Computer Organization and MARIE

# Lecture 4.2
# MARIE

Prepared By:
- Scott F. Midkiff, PhD
- Luiz A. DaSilva, PhD
- Kendall E. Giles, PhD

Electrical and Computer Engineering
Virginia Tech