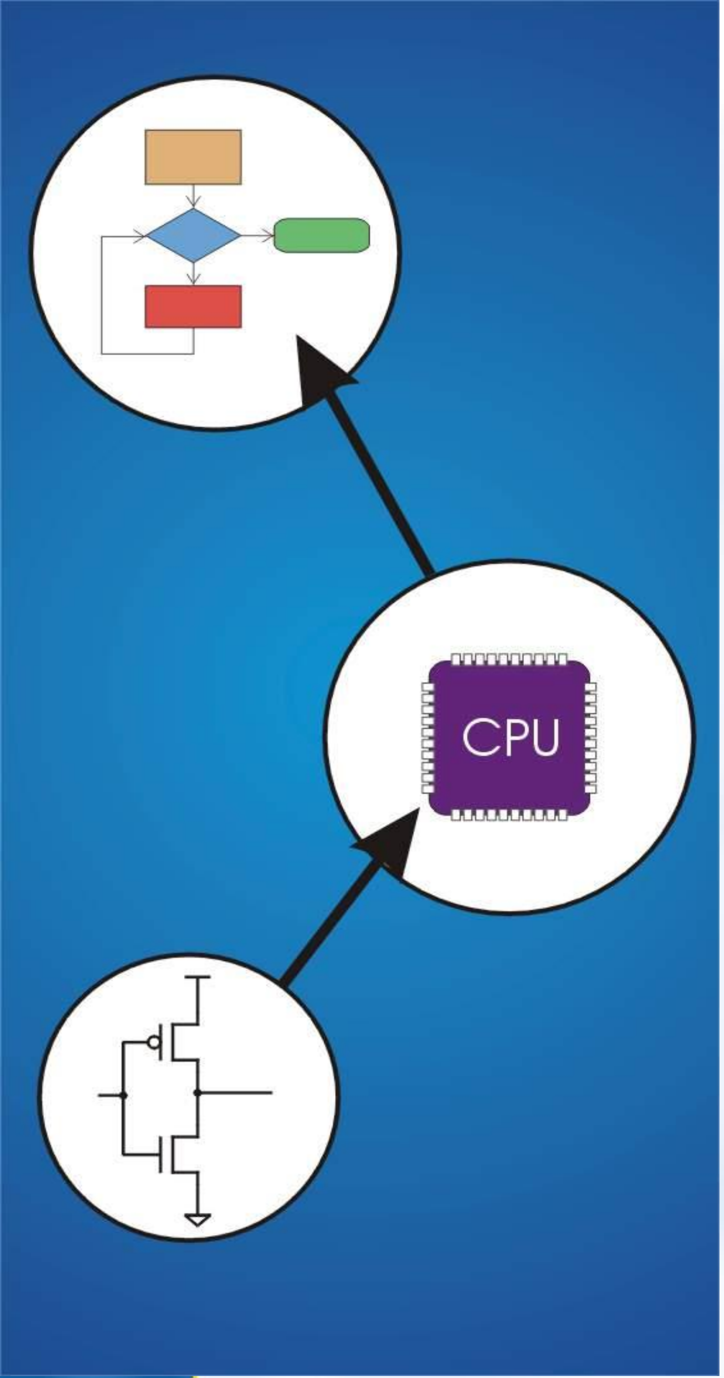


Chapter 4
The Von Neumann Model

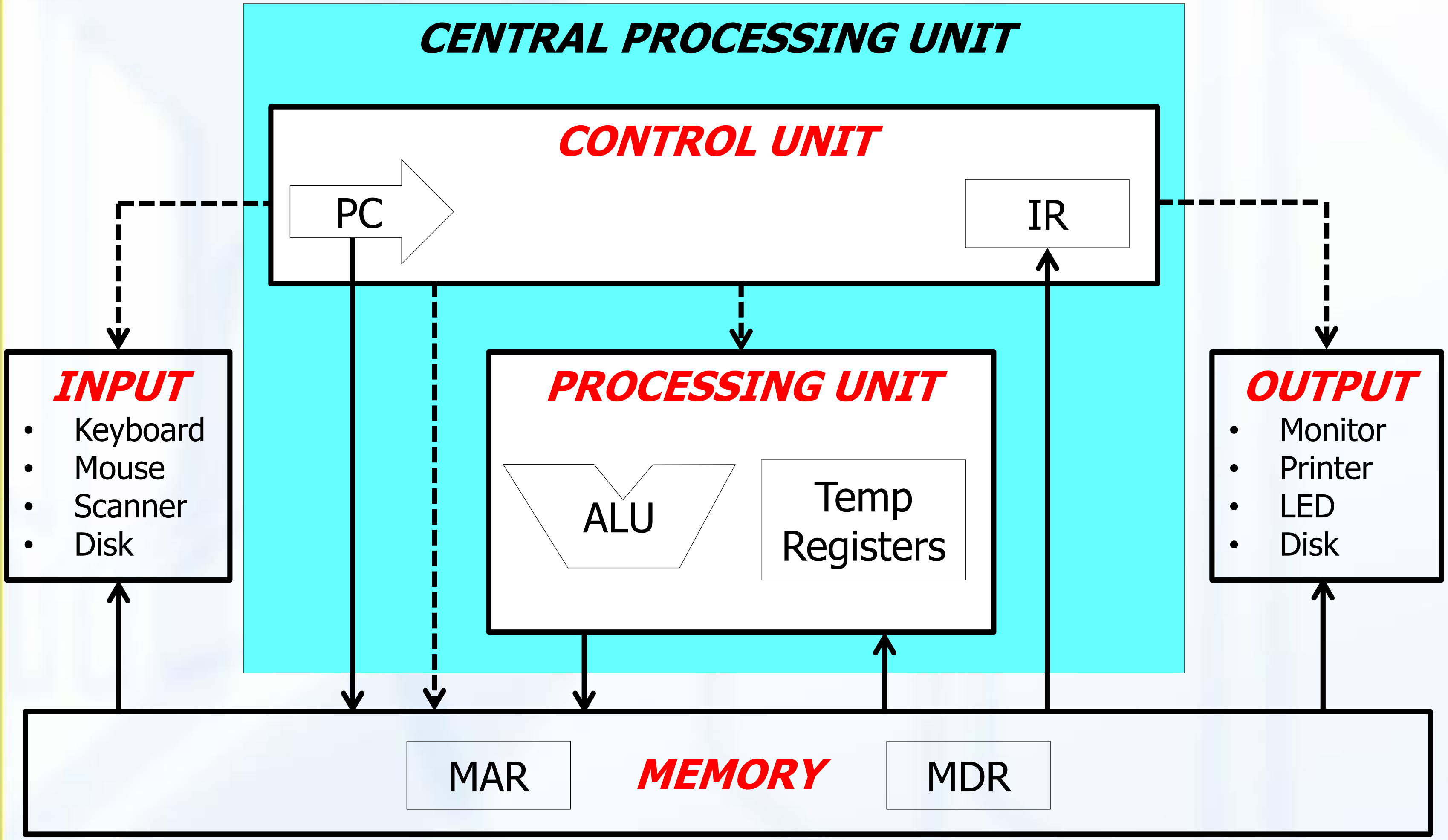
Allan Hancock College
Start here. Go anywhere.

The Stored Program Computer

- **1943: ENIAC**
  - Presper Eckert and John Mauchly -- first general electronic computer.
    (or was it John V. Atanasoff in 1939?)
  - Hard-wired program -- settings of dials and switches.
- **1944: Beginnings of EDVAC**
  - among other improvements, includes program stored in memory
- **1945: John von Neumann**
  - wrote a report on the stored program concept,
    known as the First Draft of a Report on EDVAC
- **The basic structure proposed in the draft became known as the "von Neumann machine" (or model).**
  - a memory, containing instructions and data
  - a processing unit, for performing arithmetic and logical operations
  - a control unit, for interpreting instructions

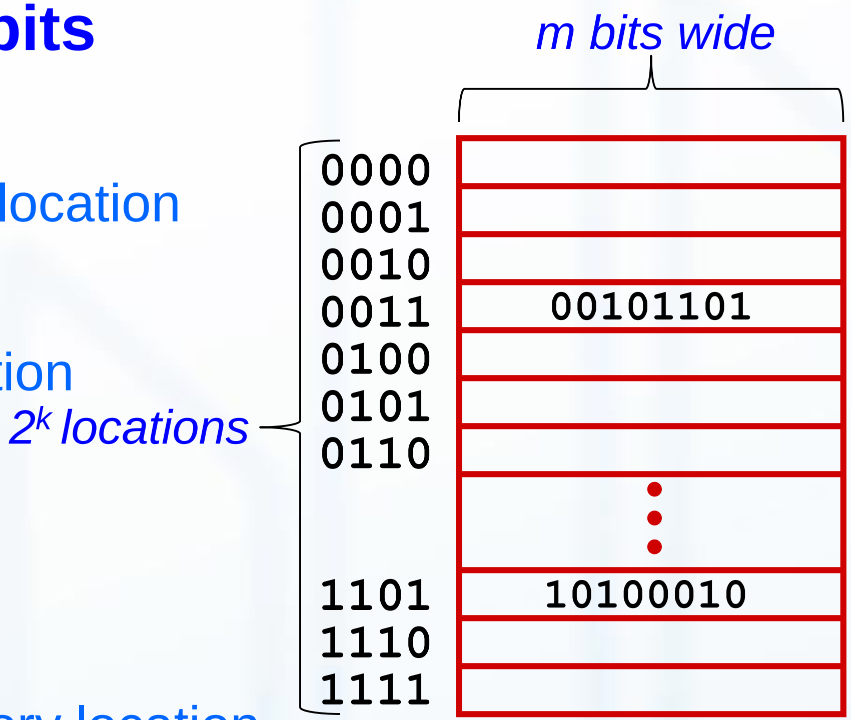
For more history, see https://en.wikipedia.org/wiki/Von_Neumann_architecture

Von Neumann Model: Consists of 5 Parts



Memory

- **2^k x m array of stored bits**
- **Address**
  - unique (k-bit) identifier of location
- **Contents**
  - m-bit value stored in location

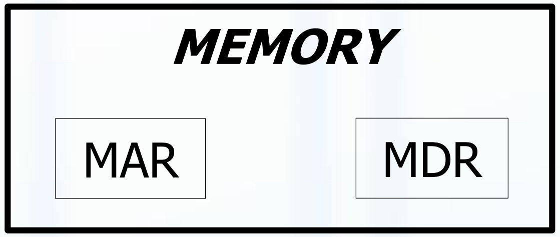


- **Basic Operations:**
- **LOAD**
  - read a value from a memory location
- **STORE**
  - write a value to a memory location

ALLAN HANCOCK COLLEGE Start here. Go anywhere.

Interface to Memory

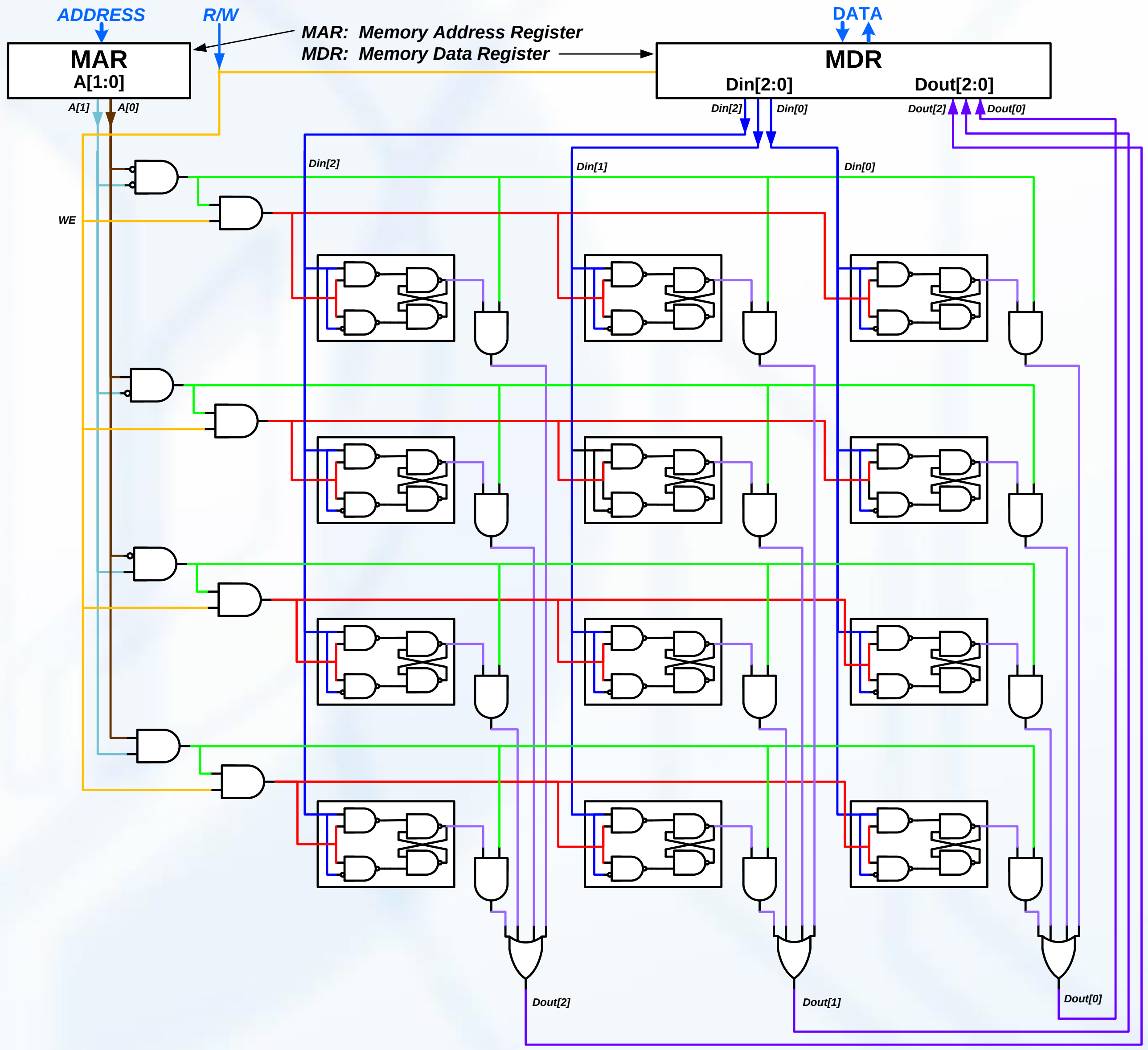
- **How does processing unit get data to/from memory?**
- **MAR: Memory Address Register**
- **MDR: Memory Data Register**



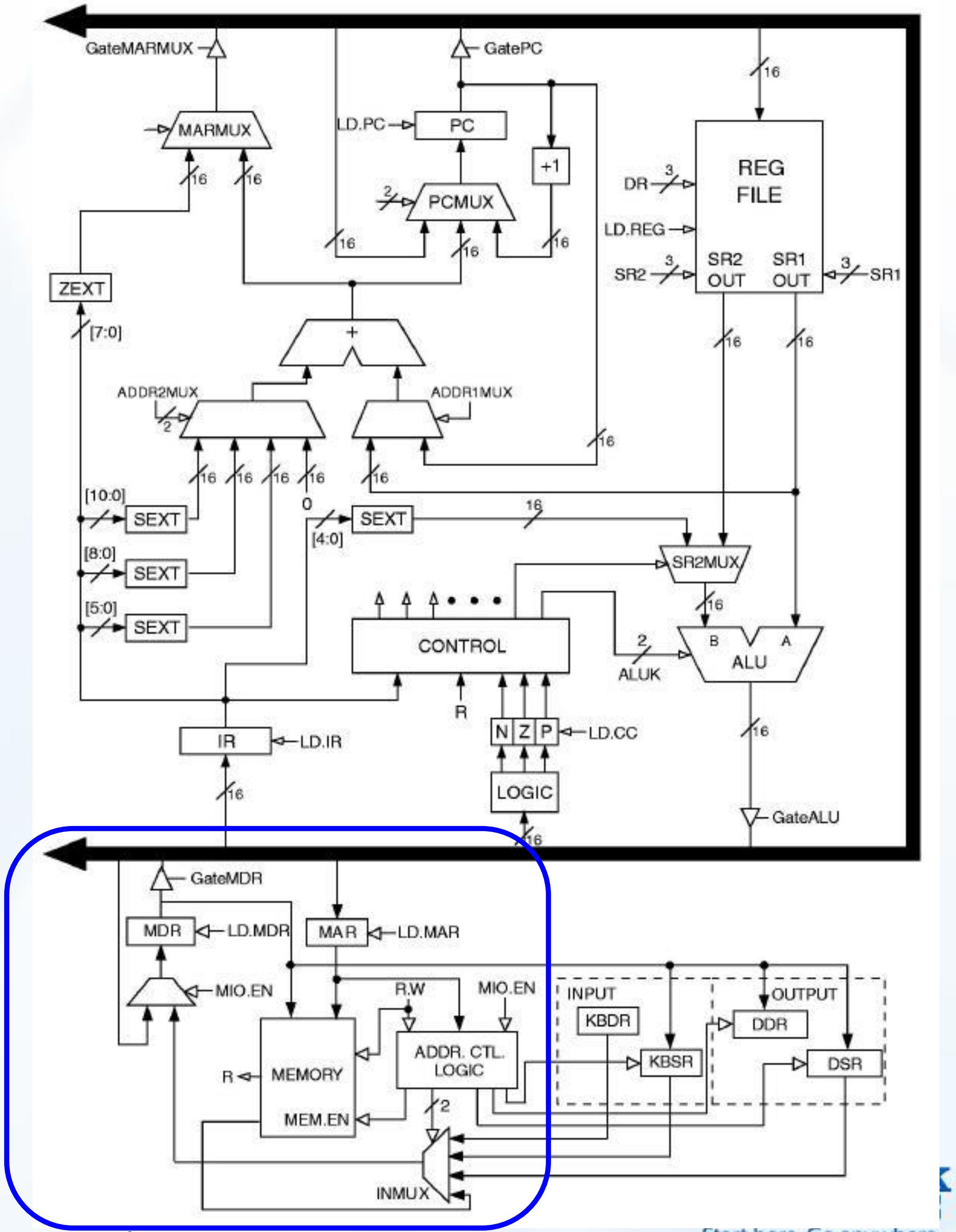
- **To LOAD a location (A):**
  1. Write the address (A) into the MAR.
  2. Send a "read" signal to the memory.
  3. Read the data from MDR.
- **To STORE a value (X) to a location (A):**
  1. Write the address (A) into the MAR.
  2. Write the data (X) to the MDR.
  3. Send a "write" signal to the memory.

ALLAN HANCOCK COLLEGE Start here. Go anywhere.

2^2 x 3-bit Memory Connected to MAR / MDR



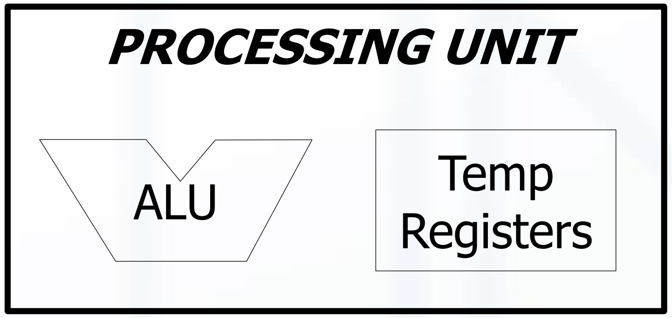
LC-3 Data Path



Memory Subsystem

Start here. Go anywhere.

Processing Unit



- **Functional Units**
  - ALU = Arithmetic and Logic Unit
  - could have many functional units. some of them special-purpose (multiply, square root, …)
  - LC-3 performs ADD, AND, NOT
- **Registers**
  - Small, temporary storage
  - Operands and results of functional units
  - LC-3 has eight registers (R0, …, R7), each 16 bits wide
- **Word Size**
  - number of bits normally processed by ALU in one instruction
  - also width of registers
  - LC-3 is 16 bits

Input and Output

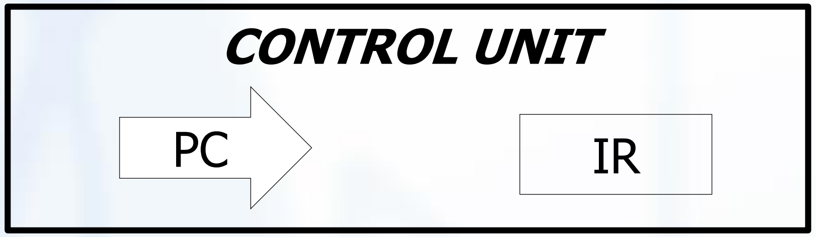
- **Devices for getting data into and out of computer memory**
- **Each device has its own interface, usually a set of registers like the memory's MAR and MDR**
  - LC-3 supports keyboard (input) and monitor/display (output)
  - Keyboard
    - Keyboard Data Register (KBDR)
    - Keyboard Status Register (KBSR)
  - Monitor/display
    - Display Data Register (DDR)
    - Display Status Register (DSR)
- **Some devices provide both input and output**
  - disk, network
- **Program that controls access to a device is usually called a device driver**
  - Driver abstracts details of hardware by providing software Application Programming Interface (API)

INPUT
- Keyboard
- Mouse
- Scanner
- Disk

OUTPUT
- Monitor
- Printer
- LED
- Disk

Control Unit

- **Orchestrates execution of the program**



- **Instruction Register (IR) contains the current instruction.**
- **Program Counter (PC) contains the address of the *next* instruction to be executed.**
- **Control unit:**
  - reads an instruction from memory
    - the instruction's address is in the PC
  - interprets the instruction, generating signals that tell the other components what to do
    - an instruction may take many machine cycles to complete

ALLAN HANCOCK COLLEGE Start here. Go anywhere.

Instruction

- **The instruction is the fundamental unit of work.**
- **Specifies two things:**
  - **opcode**: operation to be performed
  - **operands**: data/locations to be used for operation
- **An instruction is encoded as a sequence of bits. (Just like data!)**
  - Often, but not always, instructions have a fixed length, such as 16 or 32 bits.
  - Control unit interprets instruction
    - Generates sequence of control signals to carry out operation.
  - Operation is either executed completely, or not at all.

Instruction Set Architecture

- **A computer's instructions and their corresponding format are known as the computer's Instruction Set Architecture (ISA)**

- **LC-3 Instruction Overview**
  - IR[15:12]: 4-bit opcode => up to 16 instructions
  - IR[11:0]: remaining 12 bits specify operand(s)

15 14 13 12	11	10	9	8	7	6	5	4	3	2	1	0
Opcode	Operand(s)											

Example: LC-3 ADD Instruction

- **LC-3 has 16-bit instructions.**
  - Each instruction has a four-bit opcode, bits [15:12].
- **LC-3 has eight registers (R0-R7) for temporary storage (requires 3 bits to encode).**
  - Sources and destination of ADD are registers.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADD				Dst			Src1			0	0	0	Src2		

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADD	0	0	0	1	1	1	0	0	1	0	0	0	0	1	1	0

"Add the contents of R2 to the contents of R6, and store the result in R6."

Example of a ***Computational Instruction*** (ADD, AND, …)

Example: LC-3 LDR Instruction

- **Load instruction: reads data from memory**
- **Base + offset mode:**
  - add offset to base register -- result is memory address
  - load from memory address into destination register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LDR				Dst			Base			Offset					

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LDR	0	1	1	0	0	1	0	0	1	1	0	0	0	1	1	0

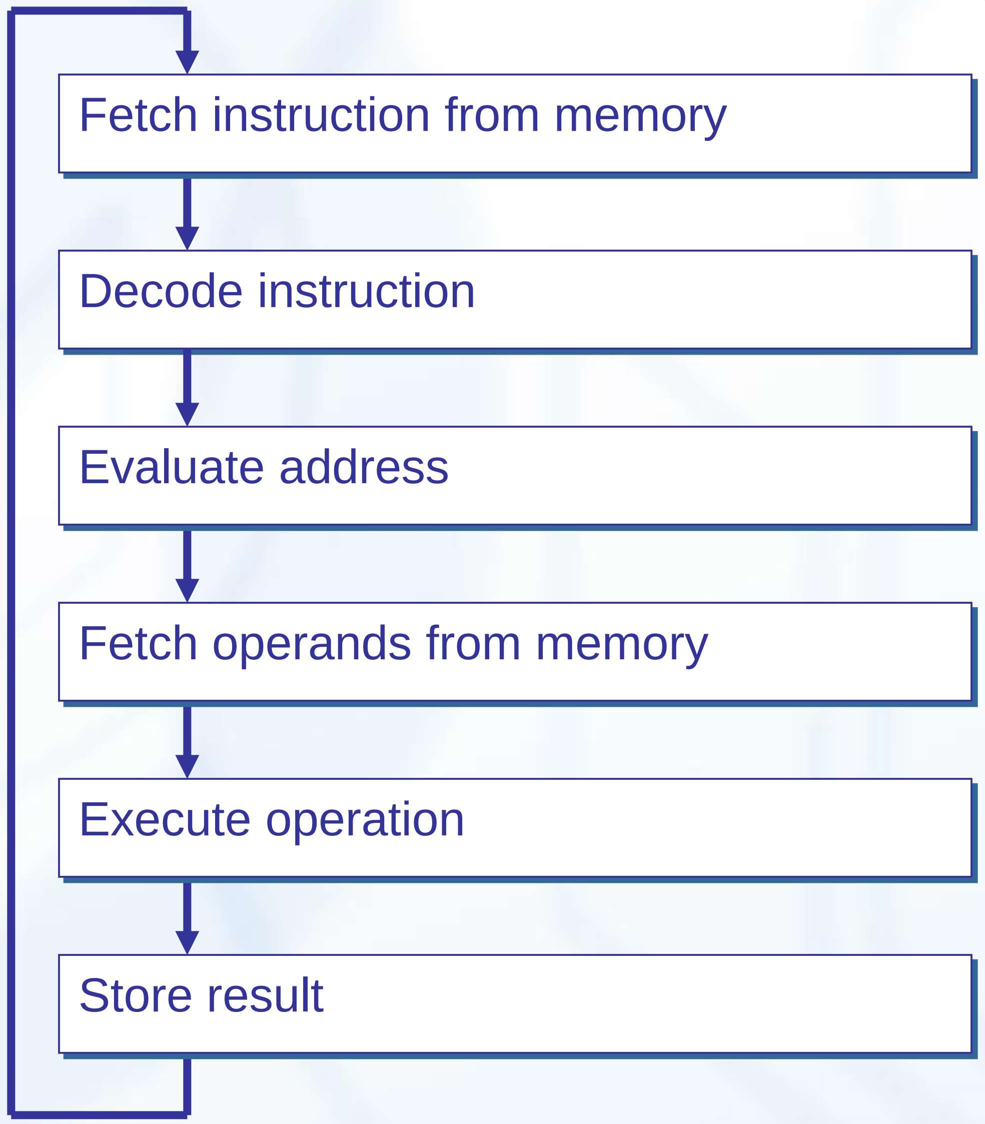
“Add the value 6 to the contents of R3 to form a memory address. Load the contents of that memory location to R2.”

$MAR = R_{BASE} + Offset$
$R_{DST} = MDR$

Example of a ***Data Movement Instruction*** (LDR, STR, …)

Instruction Processing

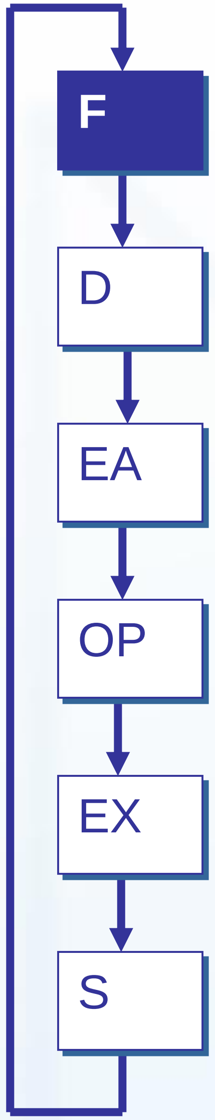
AKA: Instruction Cycle



Instruction Processing: FETCH

- **Load next instruction (at address stored in PC) from memory into Instruction Register (IR).**
  - Copy contents of PC into MAR.
  - Send “read” signal to memory.
  - Copy contents of MDR into IR.

- **Then increment PC, so that it points to the next instruction in sequence.**
  - **PC becomes PC+1.**

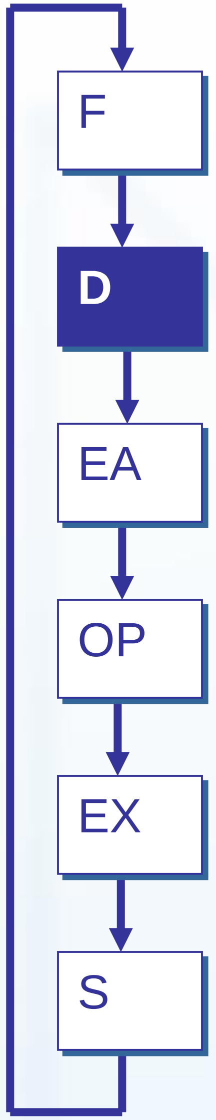


ALLAN HANCOCK COLLEGE Start here. Go anywhere.

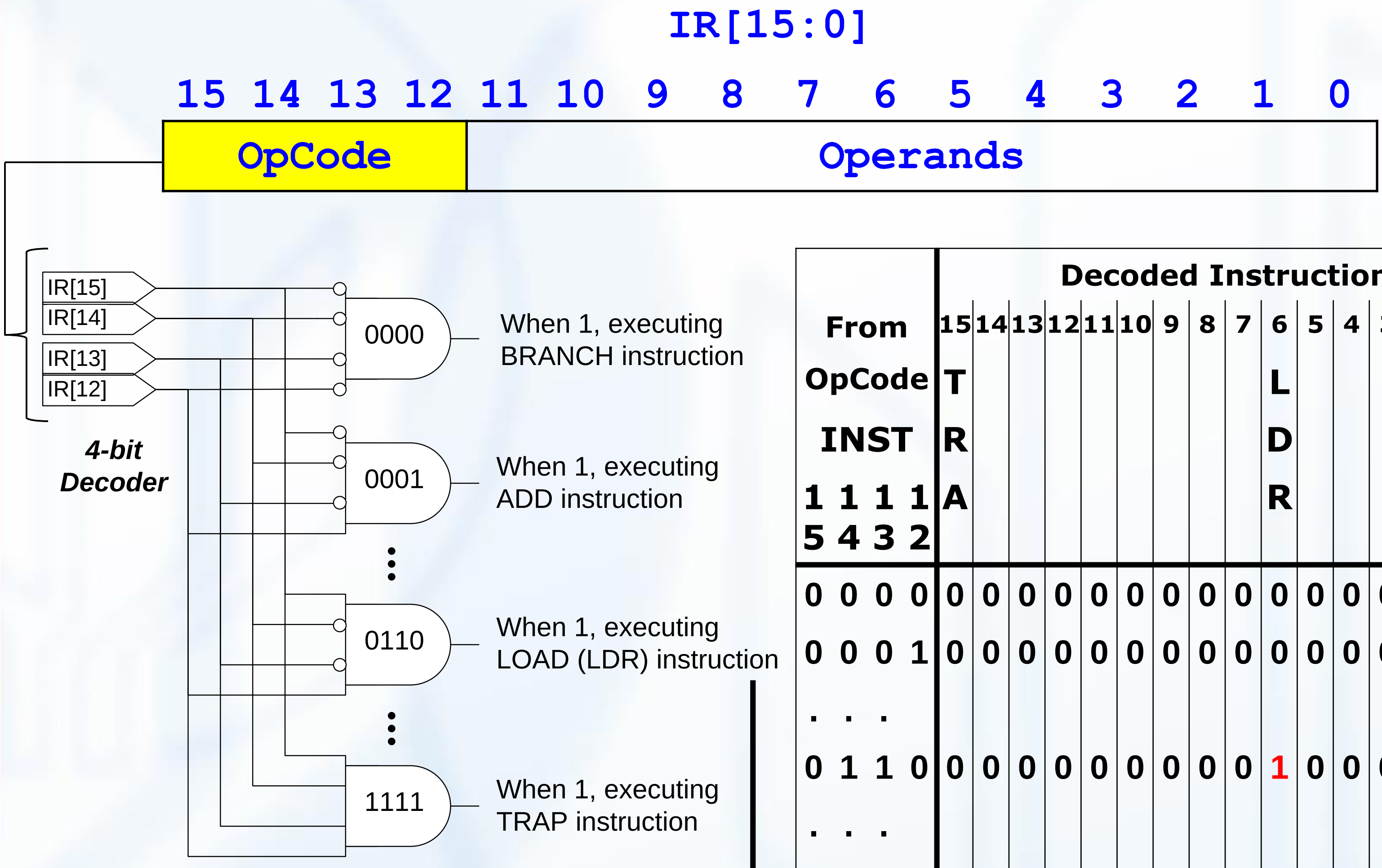
Instruction Processing: DECODE

- **First identify the opcode.**
  - In LC-3, this is always the first four bits of instruction.
  - A 4-to-16 decoder asserts a control line corresponding to the desired opcode.

- **Depending on opcode, identify other operands from the remaining bits.**
  - Example:
    - for LDR, last six bits is offset
    - for ADD, last three bits is source operand #2



LC-3 Instruction Decoder


IR[15:0]
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
OpCode
Operands
IR[15]
IR[14]
IR[13]
IR[12]
4-bit Decoder
0000
When 1, executing BRANCH instruction
0001
When 1, executing ADD instruction
0110
When 1, executing LOAD (LDR) instruction
1111
When 1, executing TRAP instruction

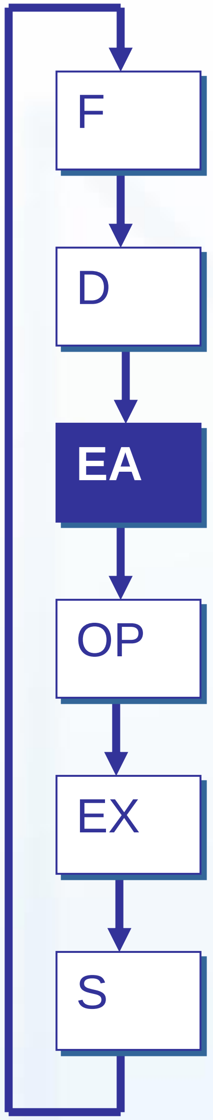
From OpCode INST				Decoded Instruction															
15	14	13	12	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				TRA									LDR					ADD	BRA
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
.	.	.																	
0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
.	.	.																	
1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Instruction Processing: EVALUATE ADDRESS

- **For instructions that require memory access, compute address used for access.**

- **Examples:**
  - add offset to base register (as in LDR)
  - add offset to PC
  - add offset to zero

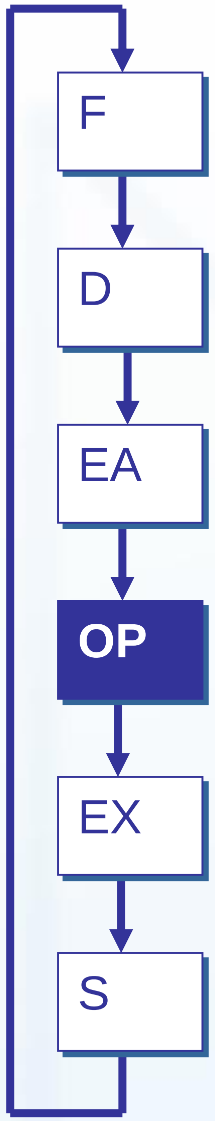
- **Not all instructions require execution of the EVALUATE ADDRESS phase**
  - Ex: ADD R3, R1, R2



Instruction Processing: FETCH OPERANDS

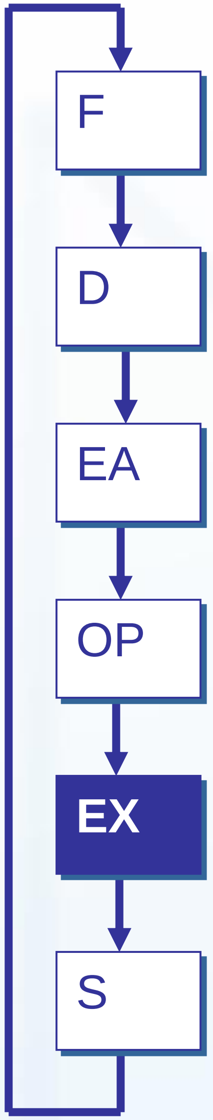
- **Obtain source operands needed to perform operation.**

- **Examples:**
  - load data from memory (LDR)
  - read data from register file (ADD)



Instruction Processing: EXECUTE

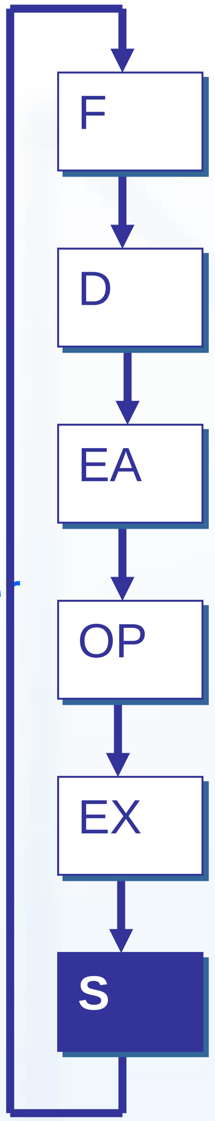
- **Perform the operation, using the source operands.**
- **Examples:**
  - send operands to ALU and assert ADD signal
  - do nothing (e.g., for loads and stores)
- **Not all instructions require execution of the EXECUTE phase**
  - Ex: LDR R1, R0, 3



Instruction Processing: STORE RESULT

- **Write results to destination. (register or memory)**

- **Examples:**
  - result of ADD is placed in destination register
  - result of memory load is placed in destination register
  - for store instruction, data is stored to memory
    - write address to MAR, data to MDR
    - assert WRITE signal to memory



Changing the Sequence of Instructions

- **In the FETCH phase, we increment the Program Counter by 1.**
- **What if we don't want to always execute the instruction that follows this one?**
  - examples: loop, if-then, function call
- **Need special instructions that change the contents of the PC.**
- **These are called control instructions.**
  - jumps are unconditional -- they always change the PC
  - branches are conditional -- they change the PC only if some condition is true (e.g., the result of an ADD is zero)

Example: LC-3 JMP Instruction

- **Set the PC to the value contained in a register. This becomes the address of the next instruction to fetch.**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
JMP				0	0	0	Base			0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
JMP	1	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0

"Load the contents of R3 into the PC."

Example of a ***Control Instruction*** (JMP, BR, …)

ALLAN HANCOCK COLLEGE Start here. Go anywhere.

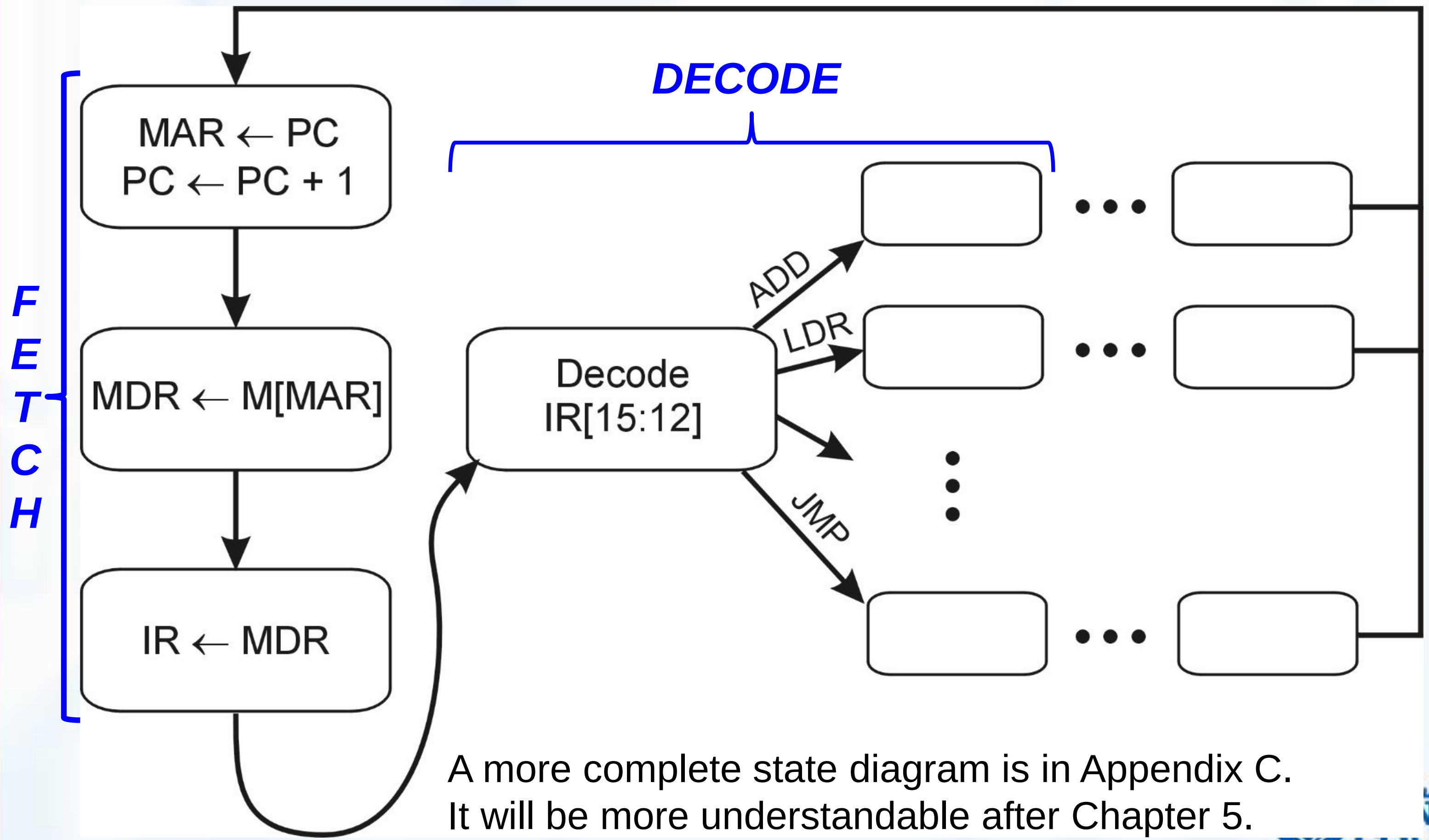
Instruction Processing Summary

- **Instructions look just like data -- it's all interpretation.**
- **Three basic kinds of instructions:**
  - Computational Instructions (ADD, AND, …)
  - Data Movement Instructions (LD, ST, …)
  - Control Instructions (JMP, BRnz, …)
- **Six basic phases of instruction processing:**

  F → D → EA → OP → EX → S
  - not all phases are needed by every instruction
  - phases may take variable number of machine cycles

Control Unit State Diagram

- **The control unit is a state machine. Here is part of a simplified state diagram for the LC-3:**



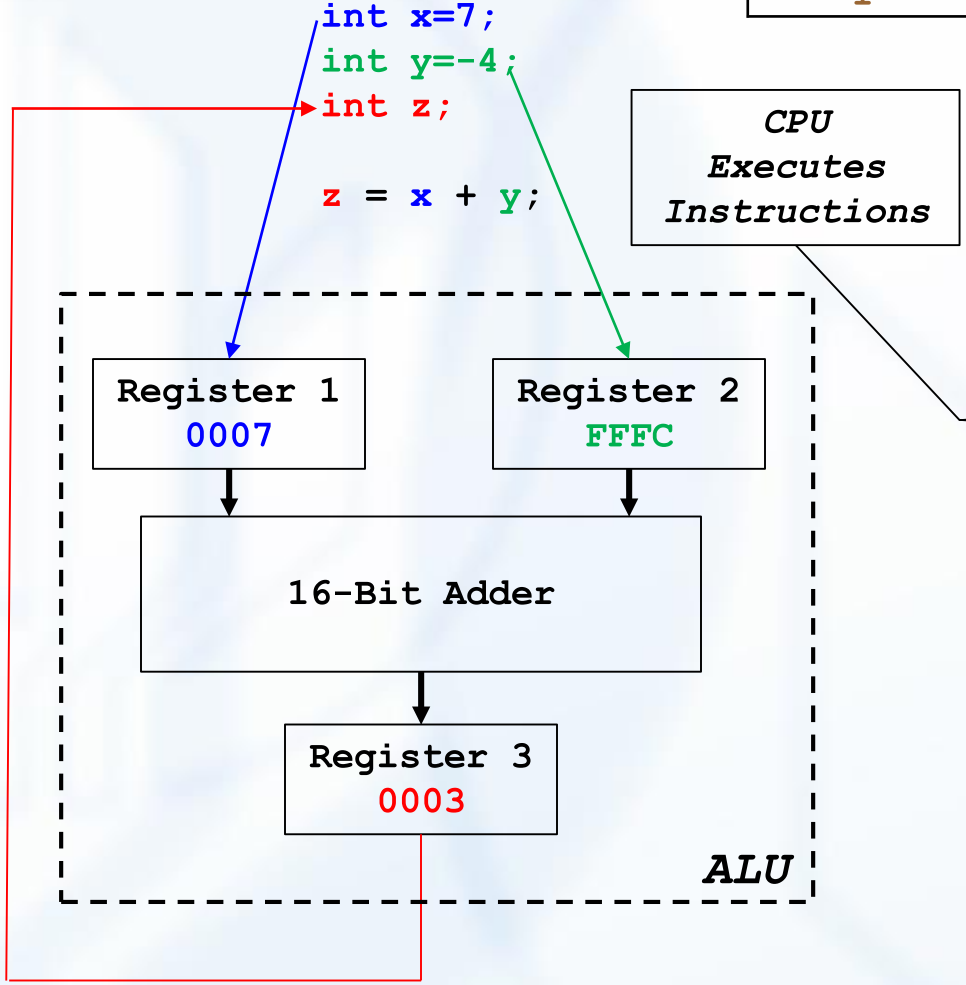
A more complete state diagram is in Appendix C.
It will be more understandable after Chapter 5.

Executing C Code

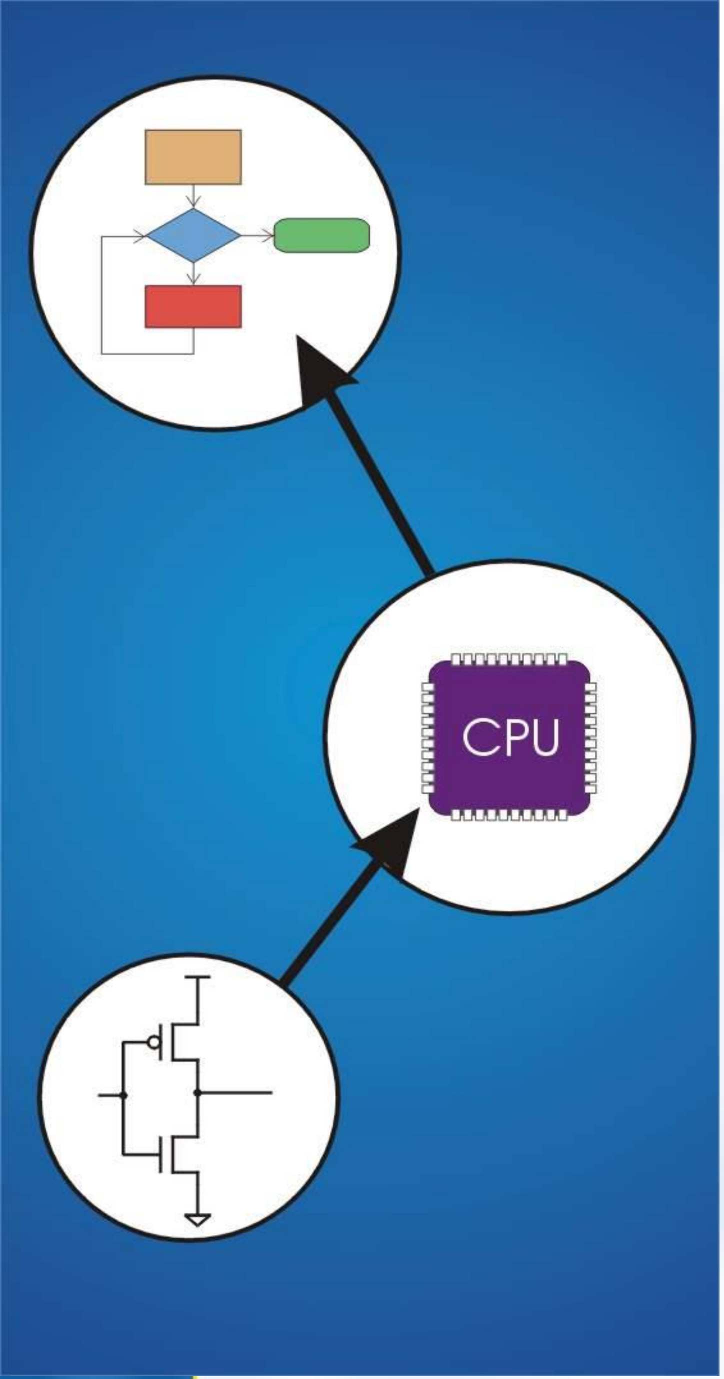
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ADD				Dst			Src1			0	0	0	Src2		
ADD				R3			R1						R2		
0	0	0	1	0	1	1	0	0	1	0	0	0	0	1	0
1				6				4				2			

```
int x=7;
int y=-4;
int z;

z = x + y;
```

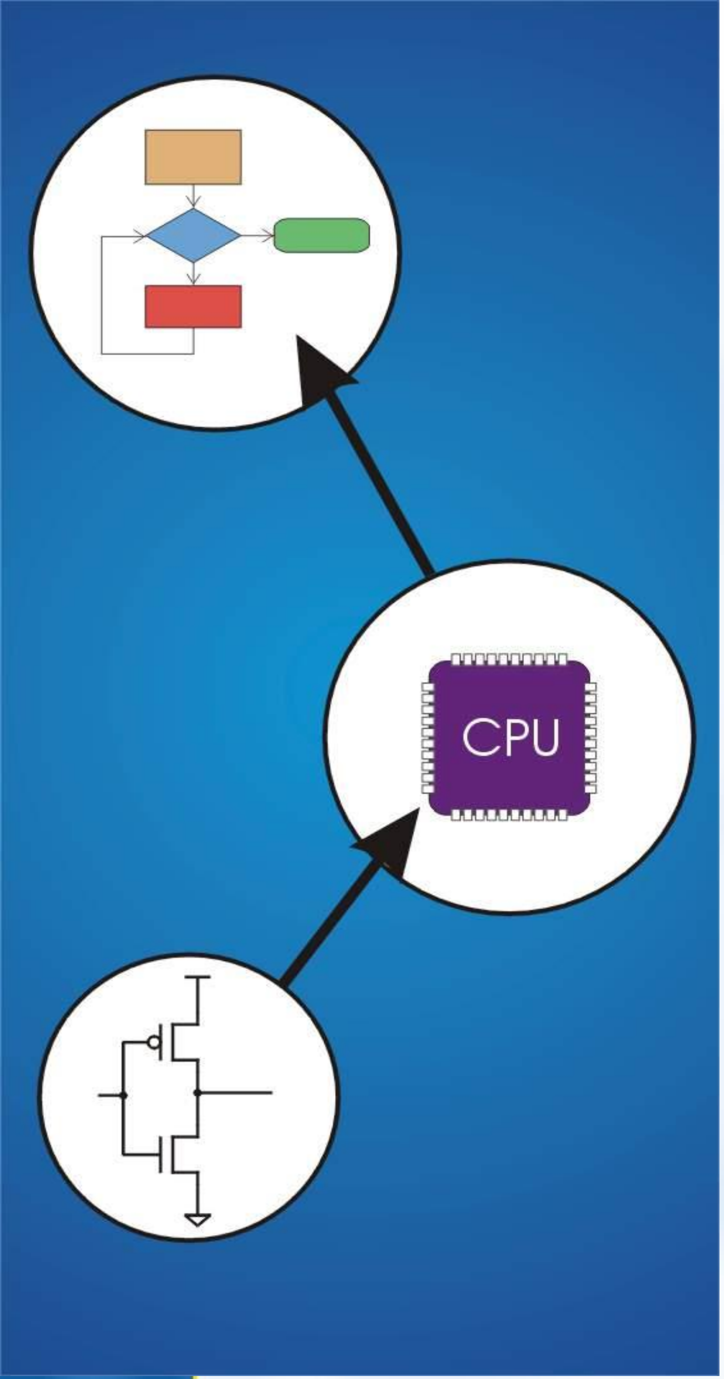


Meaning	Address	Data
	0000	
LOAD x->R1	0900	41D5
LOAD y->R2	0901	4264
ADD R1,R2->R3	0902	1642
STORE R3->z	0903	3379
y	2134	FFFC
z	5AF2	0003
x	B073	0007
	FFFF	



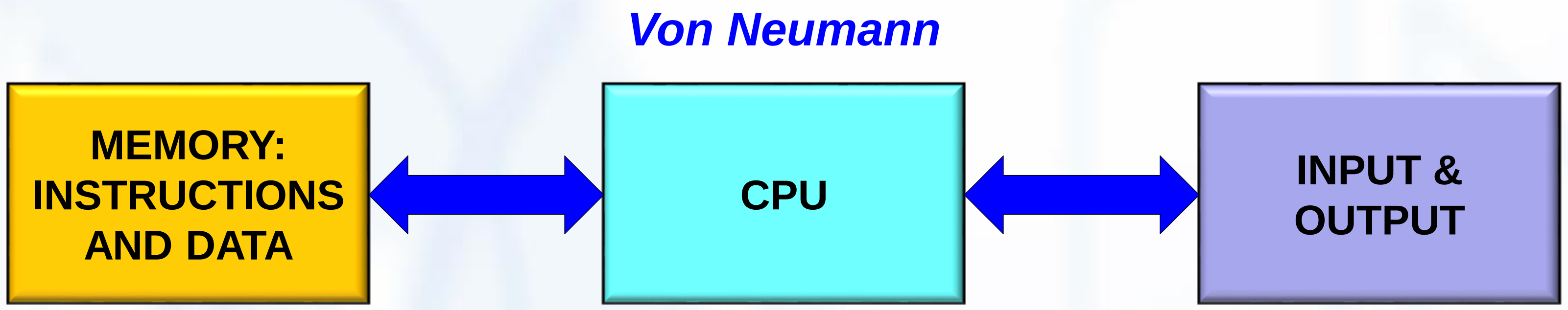
End of Chapter 4 The Von Neumann Model

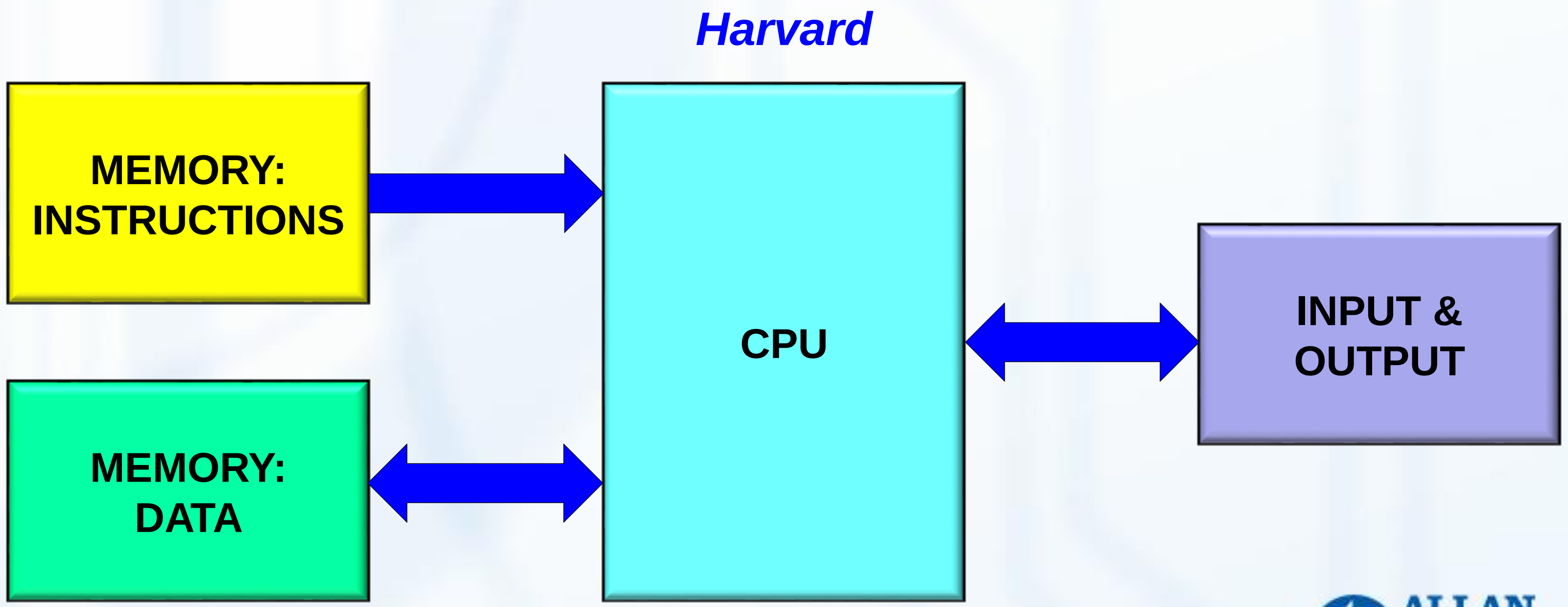
ALLAN HANCOCK COLLEGE Start here. Go anywhere.



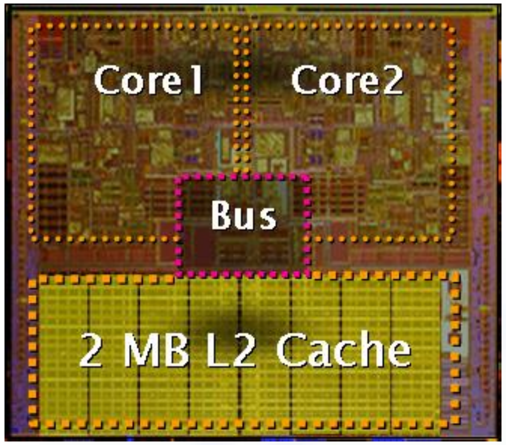
Chapter 4
The Von Neumann Model
Backup Slides

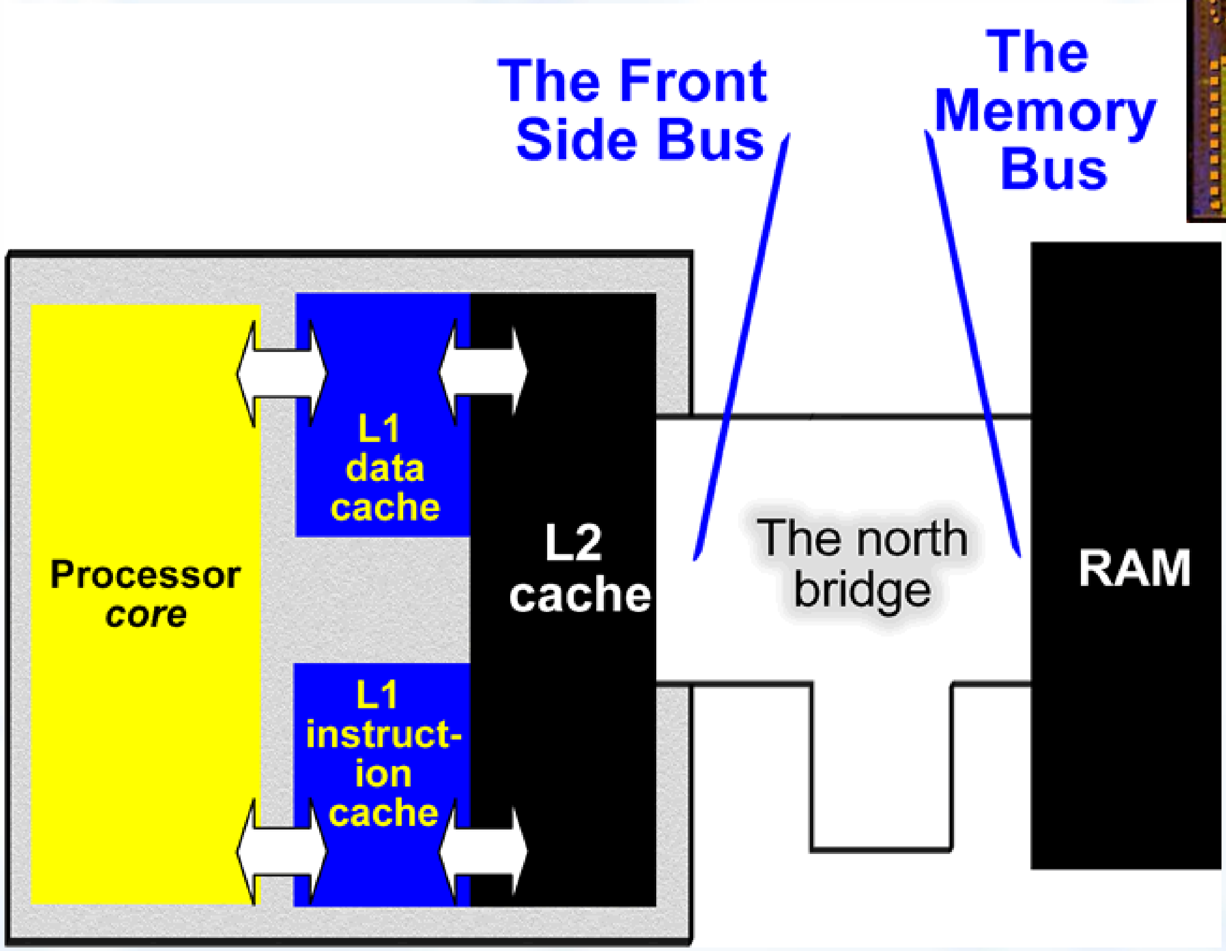
Von Neumann vs Harvard Architecture





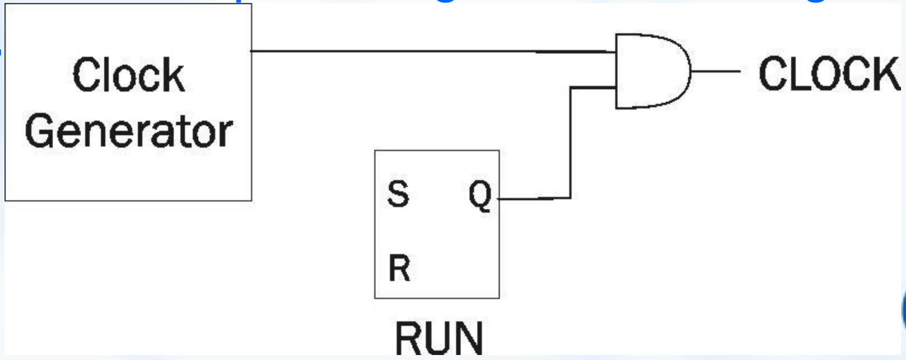
Modified Harvard Architecture





Stopping the Clock

- **Control unit will repeat instruction processing sequence as long as clock is running.**
  - If not processing instructions from your application, then it is processing instructions from the Operating System (OS).
  - The OS is a special program that manages processor and other resources.
- **To stop the computer:**
  - AND the clock generator signal with ZERO
  - When control unit stops seeing the CLOCK signal, it stops processing.


Clock Generator
CLOCK
S
Q
R
RUN