Introduction to Computing Systems
from bits & gates to C & beyond

Chapter 4

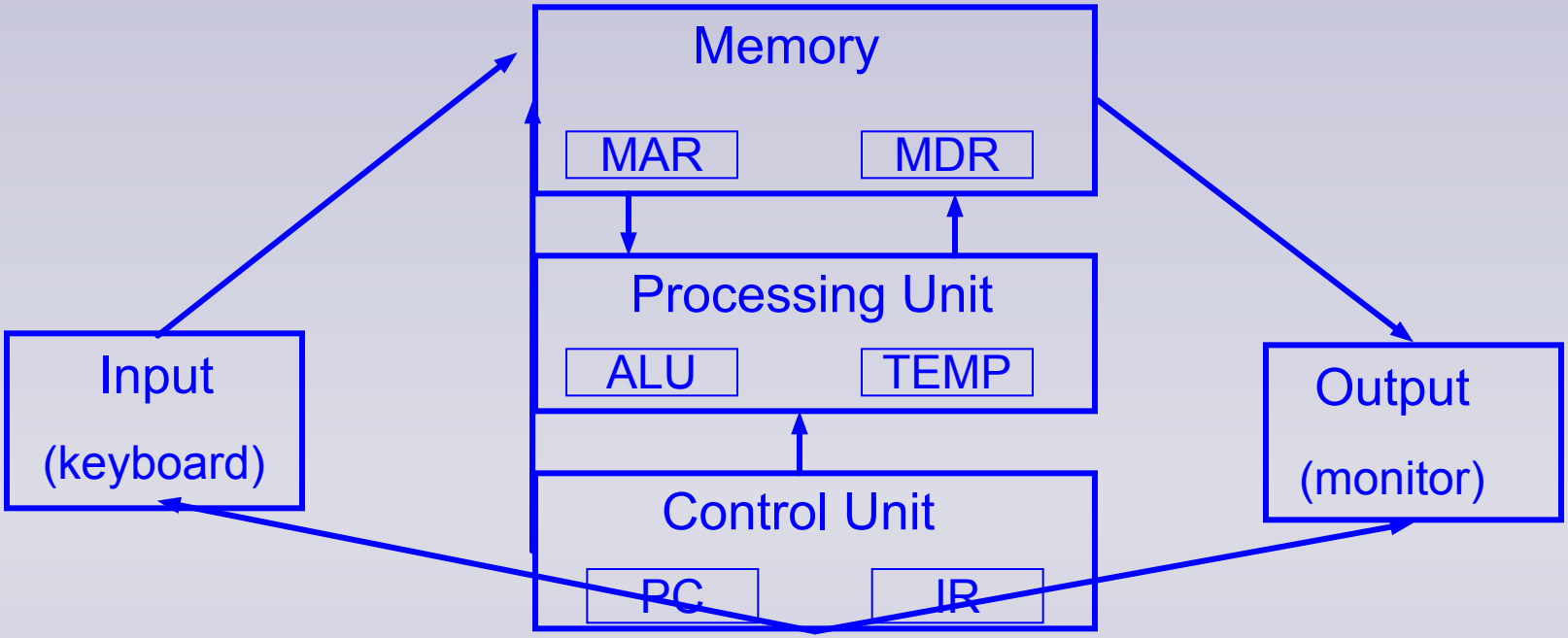
The Von Neumann Model

Basic components
Instruction processing

Putting it all together

- **The goal:**
  - Turn a theoretical device - Turing's Universal Computational Machine - into an actual computer ...
  - … interacting with data and instructions from the outside world, and producing output data.
- **Smart building blocks:**
  - We have at our disposal a powerful collection of combinational and sequential logic devices.
- **Now we need a master plan ...**

The von Neumann Model - 1



- Memory: holds both data and instructions
- Processing Unit: carries out the instructions
- Control Unit: sequences and interprets instructions
- Input: external information into the memory
- Output: produces results for the user

The von Neumann Model - 2

- **Memory**
  - **Each location has an *address* and *contents***
    - ***Address: bit pattern that uniquely identifies a memory location***
    - ***Contents: bit pattern stored at a given address.***
    - ***analogy: p.o. boxes have fixed numbers, but changing contents.***
  - **Address Space:**
    - ***The total number of memory locations ("boxes") available.***
    - ***eg. a 28 bit address provides an address space of 2^{28} locations.***
    - ***The LC-3 has an address space of 2^{16} locations - i.e. it uses a 16 bit address.***

The von Neumann Model - 3

- **Memory (continued)**
  - **Addressability (Byte vs. Word):**
    - ***a word is the basic unit of data used by the processing unit, often multiple bytes; frequently, an instruction must store or retrieve an entire word with a single memory access.***
    - ***Addressability refers to the number of bytes of memory referenced by a given address.***
    - ***Extending the p.o. box analogy (imperfectly!) with an ISA whose word size is 2 bytes:***
      - **if we have to deliver wide envelopes, we could convert pairs of the original single-width boxes into new double-wide boxes.**
      - **we then have the choice of retaining the original numbering scheme, with each of the new boxes keeping both their original addresses (Byte Addressability);**
      - **or we could renumber them all, giving a single address to each of the wider boxes (Word Addressability).**

The von Neumann Model - 4

- **Memory (continued)**
  - **Accessing memory via the MAR & MDR**
    - ***The Memory Address Register sets up the decoder circuitry in the memory.***
    - ***Read: the contents of the specified address will be written to the Memory Data Register.***
    - ***Write: the value to be stored is first written to the Memory Data Register, then the Write Enable is asserted, and the contents of the MDR are written to the specified address.***

The von Neumann Model - 5

- **Processing Unit**
  - **Does the actual work!**
    - *Can consist of many units, each specializing in one complex function.*
    - *At a minimum, has Arithmetic & Logic Unit (ALU) and General Purpose Registers (GPRs).*
    - *The number of bits a basic Processing Unit operation can handle is called the WORD SIZE of the machine.*
  - **ALU**
    - *Performs basic operations: add, subtract, and, not, etc.*
    - *Generally operates on whole words of data.*
    - *Some can also operate on subsets of words (eg. single bits or bytes).*
  - **Registers:**
    - *Fast "on-board" storage for a small number of words.*
    - *Invaluable for intermediate data storage while processing*
    - *Close to the ALU (much faster access than RAM).*

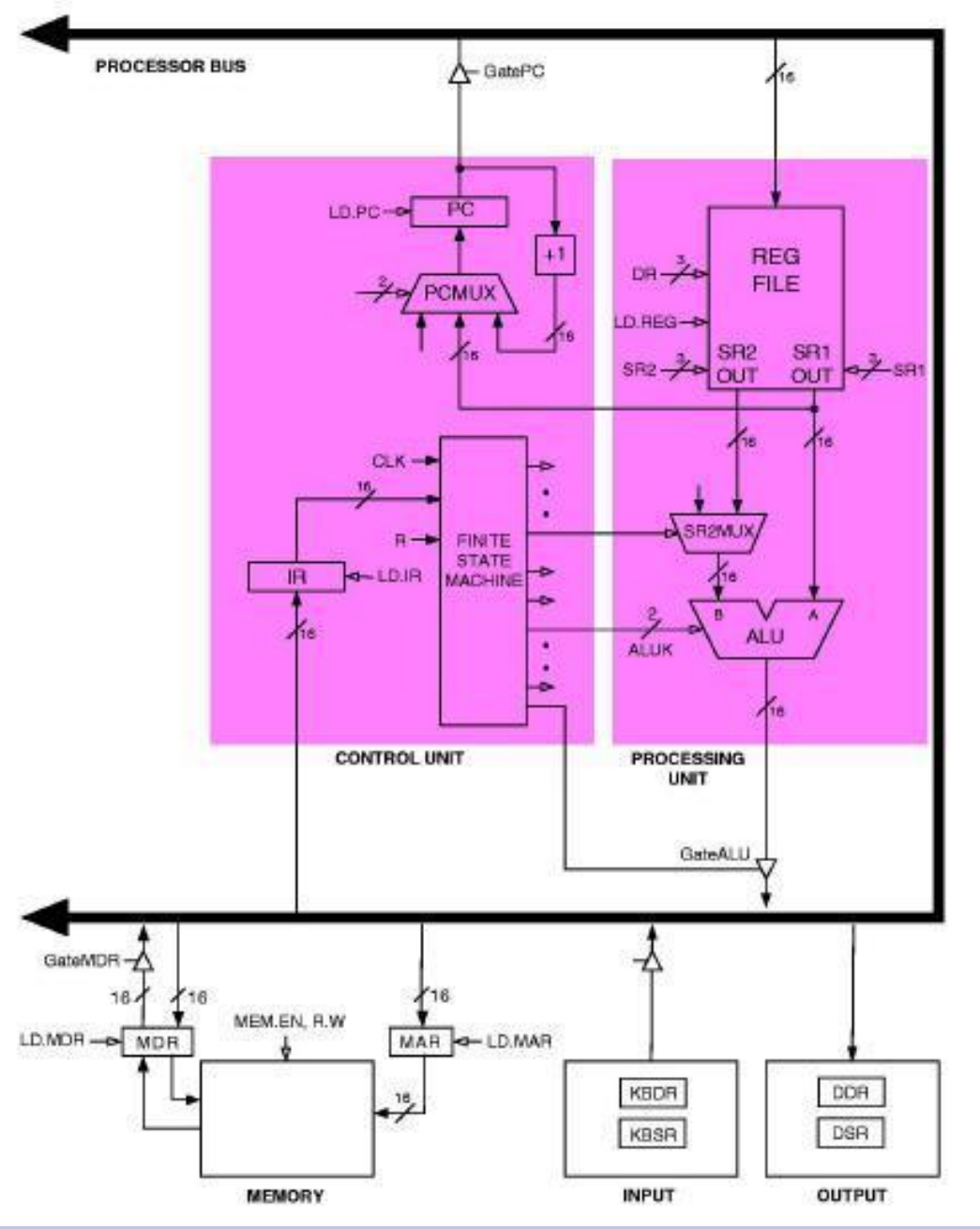
The von Neumann Model - 6

- **Control Unit**
  - The control unit coordinates all actions needed to execute the instruction
    - *It fetches & decodes each instruction, and sets up the appropriate inputs for the Memory, Processing, and I/O units as required.*
    - *It communicates with memory via the Program Counter (PC) and Instruction Register (IR)*
  - PC (aka Instruction Pointer)
    - *Holds the address of the next instruction to be fetched.*
  - IR
    - *Holds the instruction currently being executed.*
    - *This can be a single word, or multiple words.*

The von Neumann Model - 7

- **Input & Output**
  - **Generically known as *peripherals* - not in the sense that they matter less, but because they are external to the CPU.**
  - **This means we will have to develop mechanisms for autonomous devices to communicate with each other - more on this later.**

The LC-3 as a von Neumann machine



Notations

- **Sets of Bits**
  - A[3:0] denotes a set of 4 bits: A_3, A_2, A_1, A_0
  - The content of an n-bit register R is referred to as R[n-1:0]
    - ***R_{n-1} is the most significant bit (MSB), or leftmost bit***
    - ***R_0 is the least significant bit (LSB), or rightmost bit***
    - ***Given R[31:0], R[7:4] refers to the four bits from R_4 to R_7***
- **Bit Assignment**
  - R[5:0] ⇐ I[13:8]
    - ***Means that bits 5 to 0 of register R get assigned the values of bits 13 to 8 of register I.***
- **Contents**
  - ***(Reg1) means "content of Reg1"***
  - ***Mem[loc] means "content of memory location loc"***

Instruction Cycle - overview

- **The Control Unit orchestrates the complete execution of each instruction:**
  - **At its heart is a Finite State Machine that sets up the state of the logic circuits according to each instruction.**
  - **This process is governed by the system clock - the FSM goes through one transition ("machine cycle") for each tick of the clock.**

Instruction Cycle - overview

- **Six phases of the complete Instruction Cycle**
  - **Fetch**: load IR with instruction from memory
  - **Decode**: determine action to take (set up inputs for ALU, RAM, etc.)
  - **Evaluate address**: compute memory address of operands, if any
  - **Fetch operands**: read operands from memory or registers
  - **Execute**: carry out instruction
  - **Store results**: write result to destination (register or memory)

Instruction Cycle - step 1

- **Fetch**
  - The first step is to read an instruction from memory.
  - This actually takes several smaller steps, or "micro-instructions":
    - *MAR ← (PC)* *; use the value in PC to access memory*
    - *PC ← (PC) + 1* *; increment the value of PC*
    - *MDR ← Mem[(MAR)]* *; copy contents of the memory location whose address is stored in MAR to MDR*
    - *IR ← (MDR)* *; copy contents of MDR to IR*
  - Steps 1, 2 & 4 take a single machine cycle each, but step 3 (memory access) can take many machine cycles .

Instruction Cycle - step 2

- **Decode**
  - **The opcode is input to the controller (a Finite State Machine), which sets up the control signals that orchestrate the ensuing sequence of events.**

Instruction Cycle - step 3

- **Evaluate Address**
  - **Computes the address of the operand (if any), or of the memory location to be accessed: e.g. the location from which to obtain a value in a LOAD instruction.**
    - ***This is known as the Effective Address (EA).***

Instruction Cycle - step 4

- **Fetch Operands**
  - Obtains the source operand(s), if required for execution.
  - Operands can come from Registers or RAM, or be embedded in the instruction itself.
    - ***The Effective Address (EA) determined in the previous step is used to obtain an operand from memory.***

Instruction Cycle - step 5

- **Execute**
  - **Now that everything is in place, the instruction is executed.**
    - ***e.g. if the opcode was ADD, the two source operands are added by the ALU.***
    - ***If the opcode was a control instruction, a value is written to the PC***
    - ***Data Movement instructions don't have an execute phase***

Instruction Cycle - step 6

- **Store Result**
  - **If there is a result from the operation it is written to memory (using the EA), or to a register.**

Note: Most instructions don't consist of all 6 steps – e.g.

- **If only using registers, skip Evaluate Address**
- **If only moving data, skip Execute**

Instruction Cycle - step 7

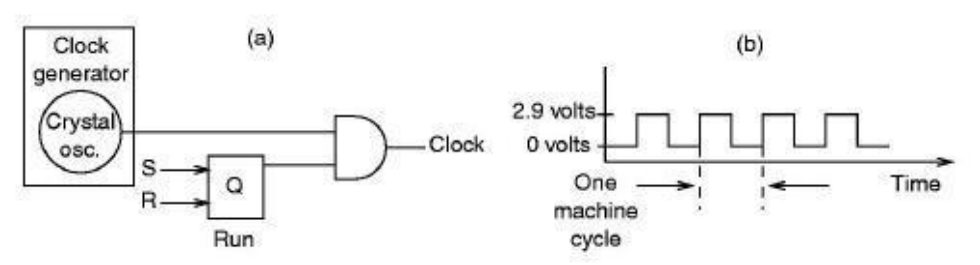
- **Start over …**
  - **The control unit just keeps repeating this whole process: so it now Fetches a new instruction from the address currently stored in the PC.**
    - ***Recall that the PC was incremented in the first step (FETCH), so the instruction retrieved will be the next in the program as stored in memory - unless the instruction just executed changed the contents of the PC.***

Types of Instruction

- ***Operate* Instructions**
  - process data (addition, logical operations, etc.)
- ***Data Movement* Instructions …**
  - move data between memory locations and registers.
- ***Control* Instructions …**
  - change the sequence of execution of instructions in the stored program.
    - ***The default is sequential execution: the PC is incremented by 1 at the start of every Fetch, in preparation for the next one.***
    - ***Control instructions set the PC to a new value during the Execute phase, so the next instruction comes from a different place in the program.***
    - ***This allows us to build control structures such as loops and branches.***

Stopping the computer

- **"User" programs terminate simply by handing control back to the Operating System (OS):**
  - ***The OS then enters a "waiting" loop until a new program is run.***
  - ***The Control Unit is still actively stepping through the instruction cycle.***
- **Left to itself, the control unit would just keep fetching instructions from memory … until we pull the plug!**
  - ***We can cease processing completely by stopping the machine cycle - i.e. by stopping the clock.***



The Instruction Cycle as FSM