ECE 231 – 1a
Intermediate Programming

Dr. Daryl O. Lee
daryllee@ece.unm.edu

THE UNIVERSITY of NEW MEXICO

About this class

- Sitting in a room listening to somebody talk is not the best way to learn (and much research backs this up).
- This course will de-emphasize listening to me talk and emphasize your learning ideas from resources out of class, and discussing problems and questions in class.
- We will try to make our classroom a place of active conversation, rather than passive listening.

We will also use a lot of classroom time as "lab time".

ECE 231

Course Catalog Description – Required course for CompE

- **ECE 231 – Intermediate Programming and Engineering Problem Solving-** **Introduction to elementary data structures, program design and computer-based solution of engineering problems. Topics include use of pointers, stacks, queues, linked lists, trees, graphs, systems and device level programming and software design methodology. Prerequisites: ECE 131.**

Credit Information

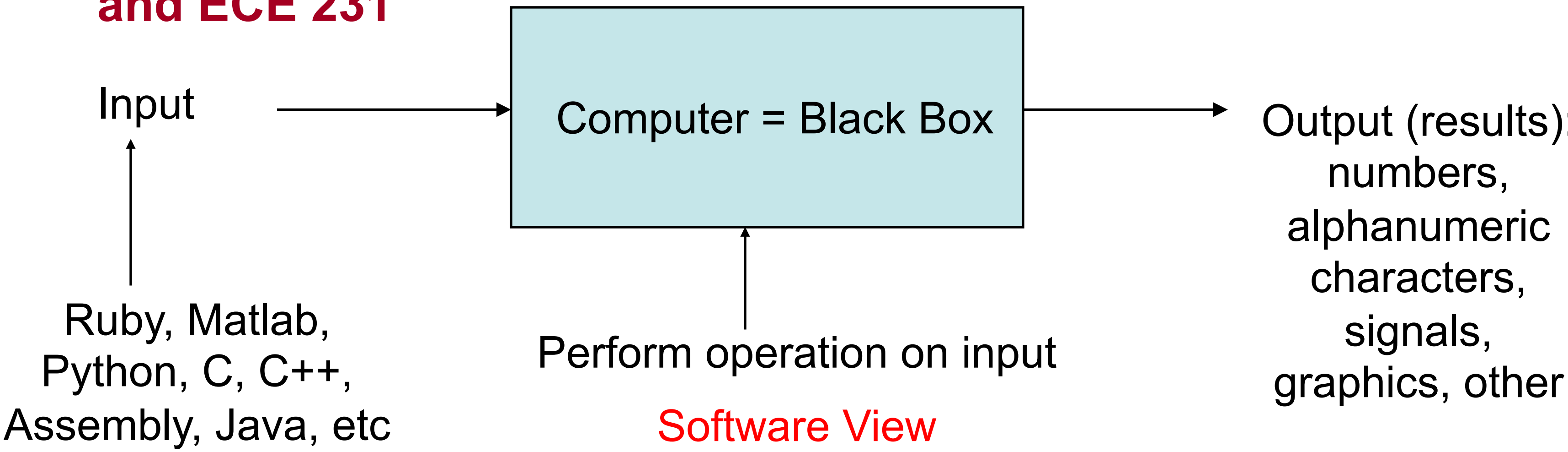
- **We have three credit hours for lecture. We will setup a lab/recitation session to work on assignments. Preferably Tue and Thu from 2 to 4 pm in ECE Room 215.**

Digital System

- **Manipulates discrete elements of information**
- **Best example: digital computer, smartphone, tablets**
- **Capable of greater accuracy and reliability than analog systems**

Two views

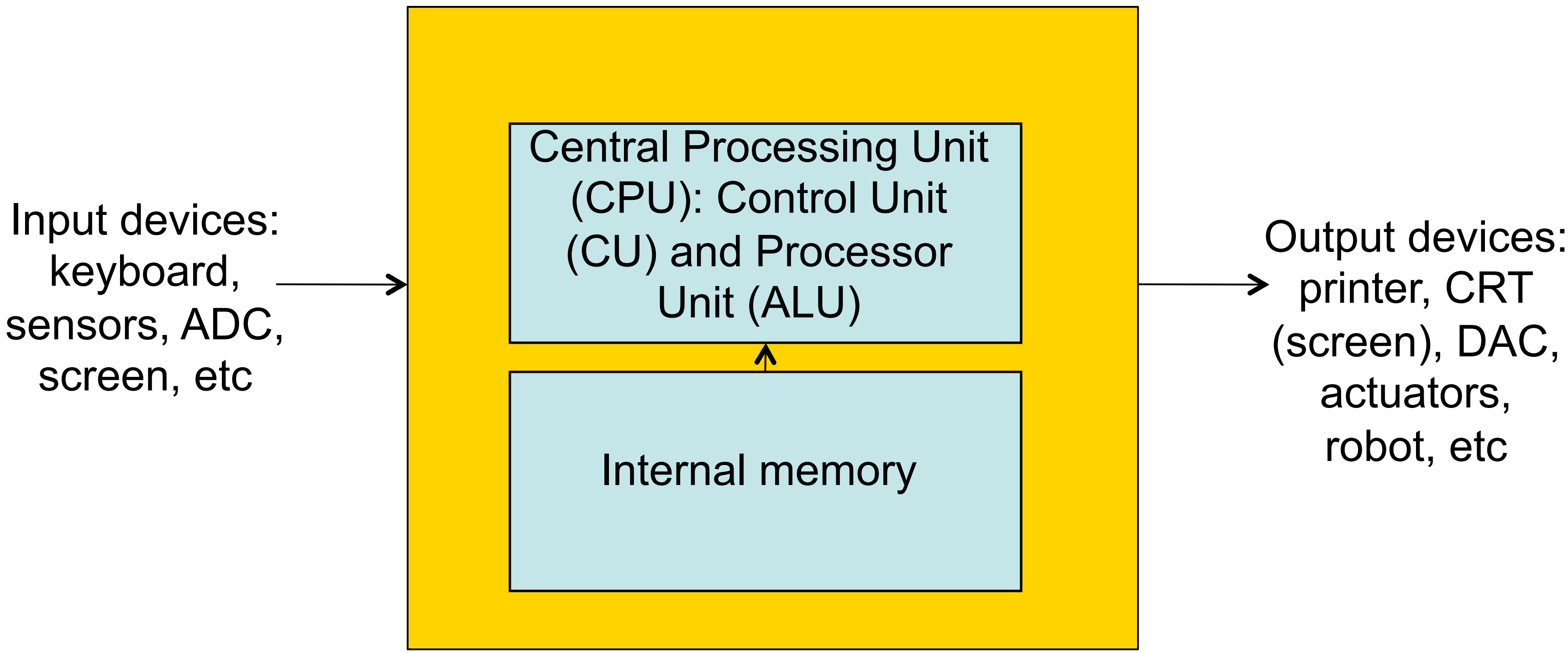
- **From an introductory programming course (software); ECE 131 and ECE 231**



Digital System

Two views

- **From ECE 238L view, hardware**
- **CPU: coordinates activities; timing operations; execution of instructions in sequence, “manager”**

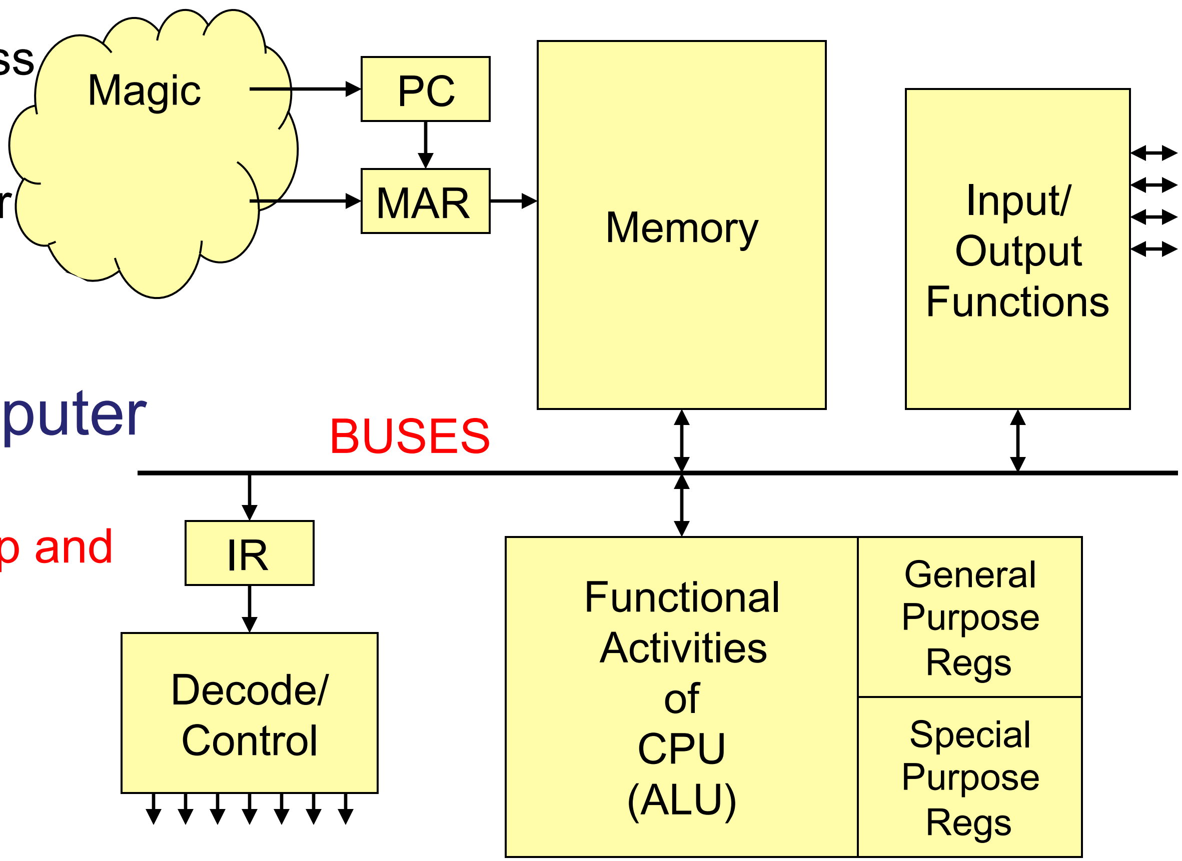


Hardware View

Basic Computer Organization

PC = program counter
MAR = memory address register

IR = instruction register



Inside the Computer

BUSES (inside the chip and outside):

Address Bus

Data Bus

Control Bus

Hardware View

Digital System

Digital Components

- **Central Processing Unit (CPU): Control Unit (CU) (manager) and the Arithmetic Logic Unit (ALU) (data processing)**
- **Memory Unit (MU): program development, holds program and data**
  - **Two types: a) Main (primary or internal) very fast, temporary storage; and b) Secondary or external, slower, long-term storage (disks, CDs, DVDs, optical storage devices, other)**
- **Input/Output (I/O) Devices or Peripherals**
  - **Communication with outside world: Network Interface Card (NIC), Monitor (CRT), plotter, printer, actuators, other**

Digital System

Digital Components

- **All internal and external digital components are connected via BUSES (highways)**
  - **Address Bus**
  - **Data Bus**
  - **Control Bus**

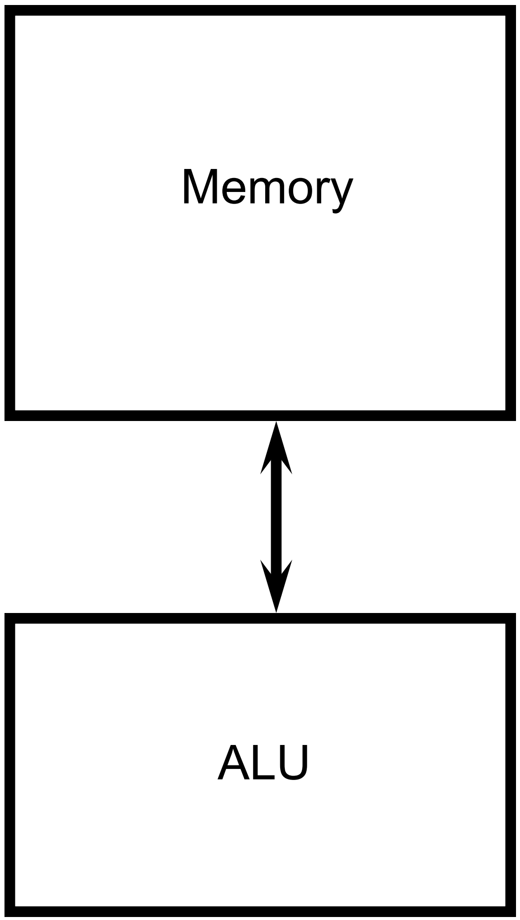
- **Data: physical quantities (signals) can assume only discrete values. Internal signals are either ON or OFF, True or False, 0 or 1, 0v or 5v, other**
  - **Collection of symbols**
  - **Valid only when we give the symbols meaning**

EXAMPLE: MC68000 - A 16/32 Bit Architecture

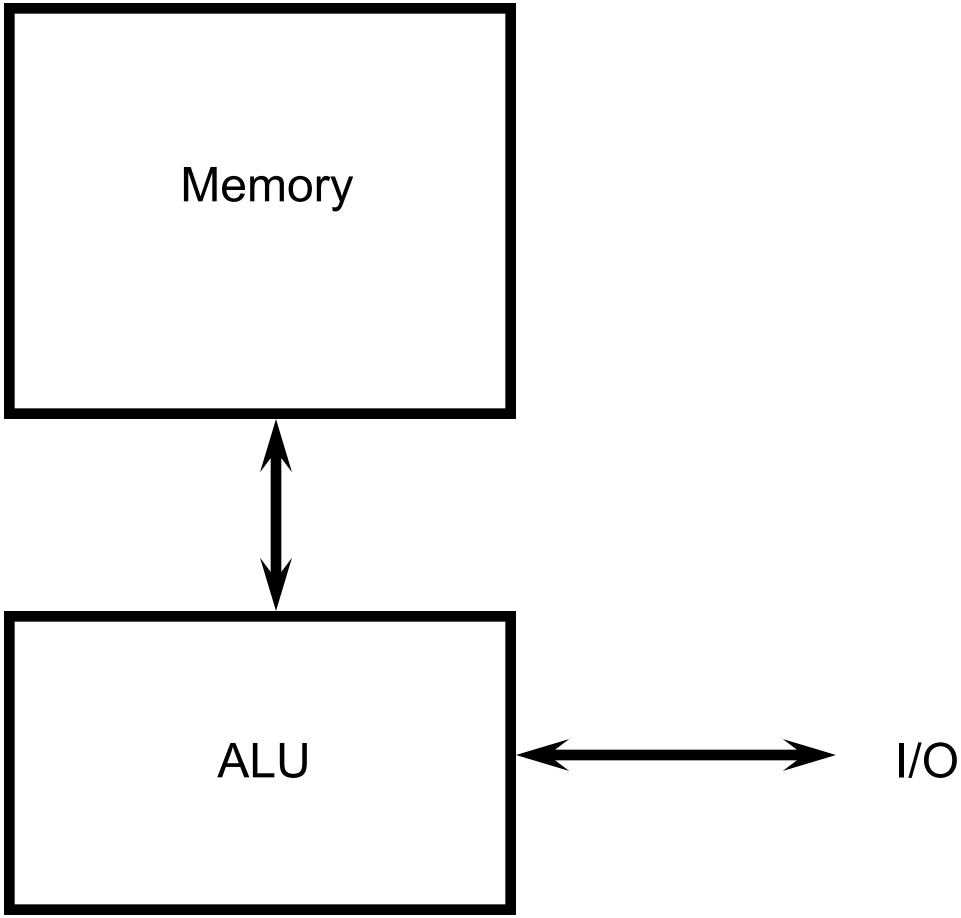
- **17 32 bit data and address registers**
- **16 Mbyte addressing range**
- **56 Instruction types**
- **5 Data types (bit, BCD, byte, word, longword)**
- **Memory mapped I/O**
- **14 Addressing modes**
- **Other family members → more features**

Basic Computer Architecture

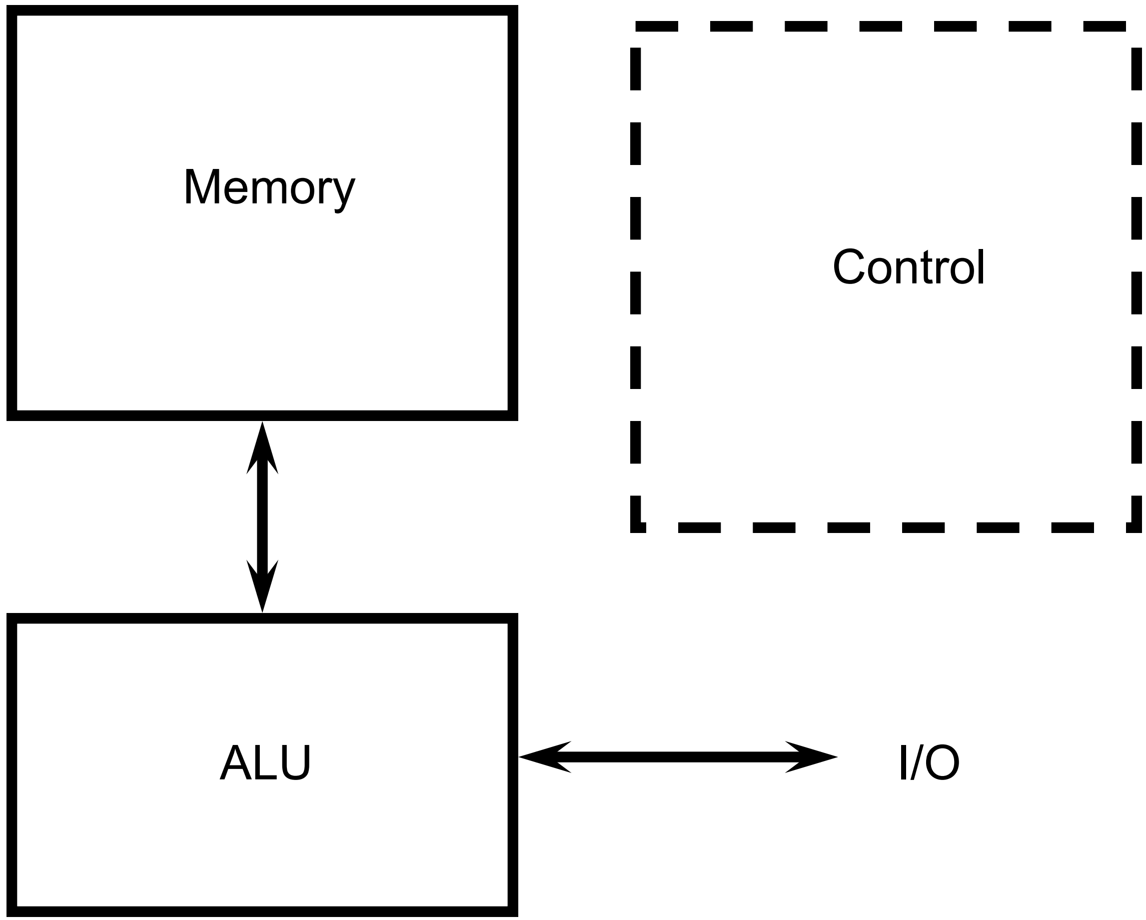
Basic Computer Architecture



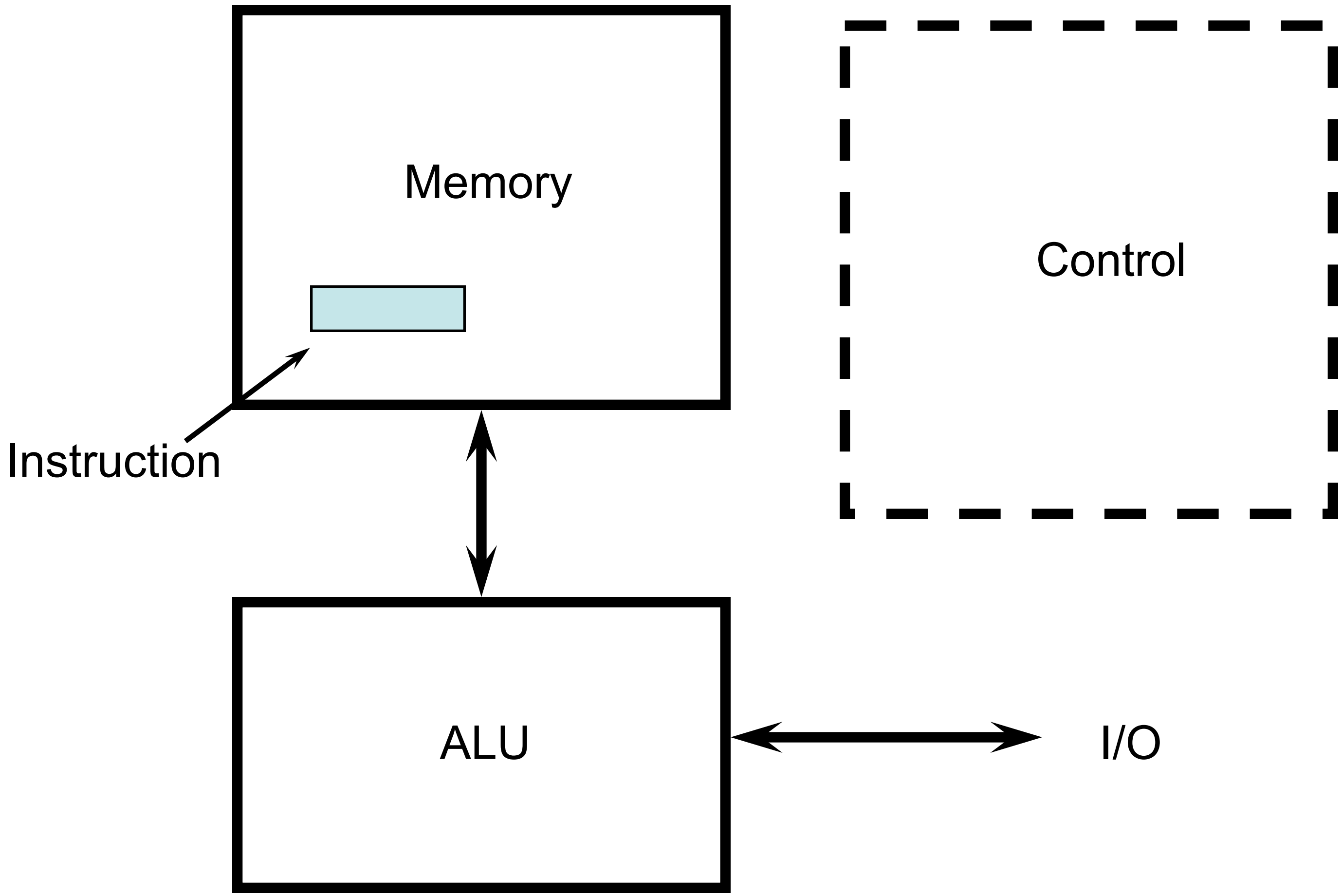
Basic Computer Architecture



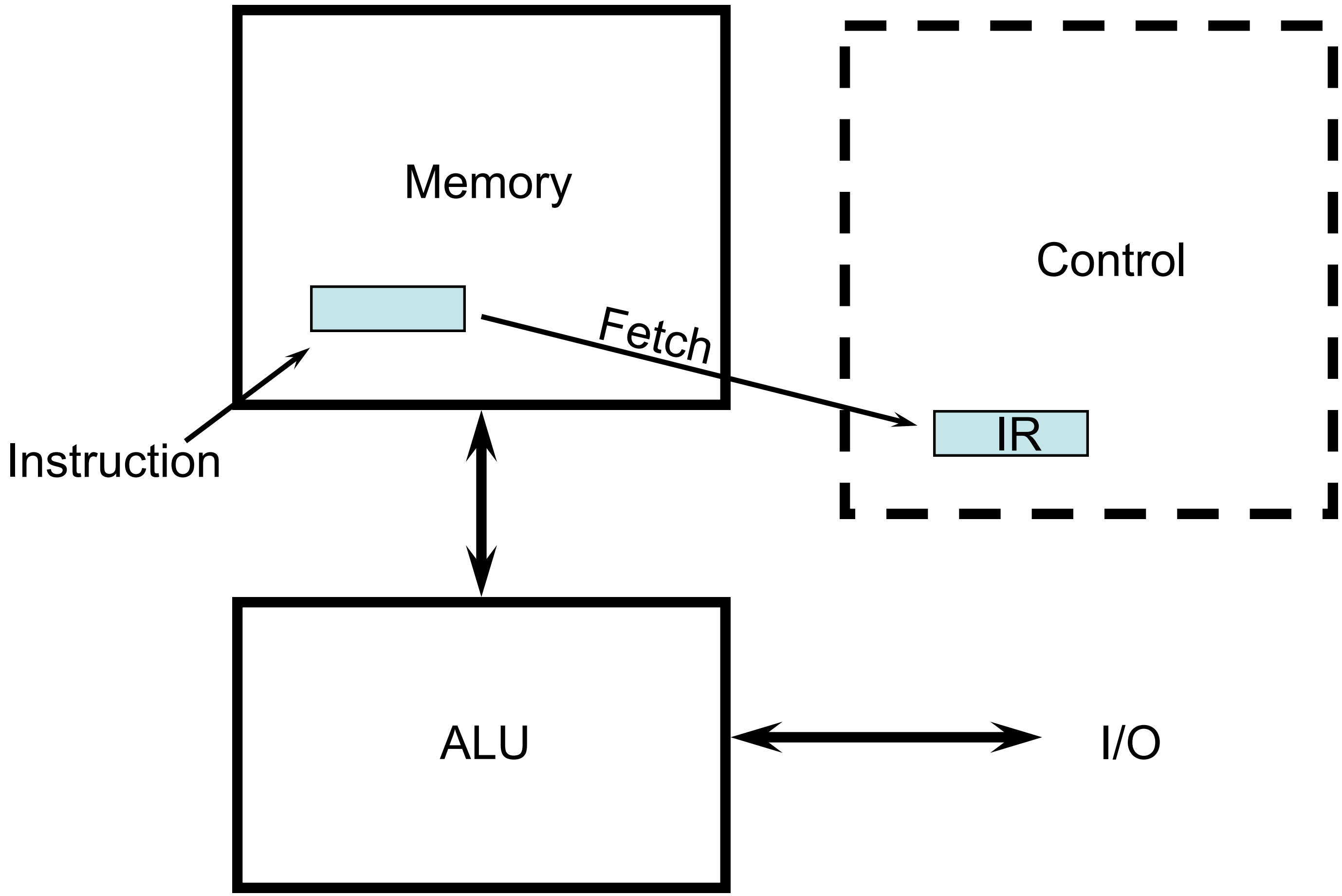
Basic Computer Architecture



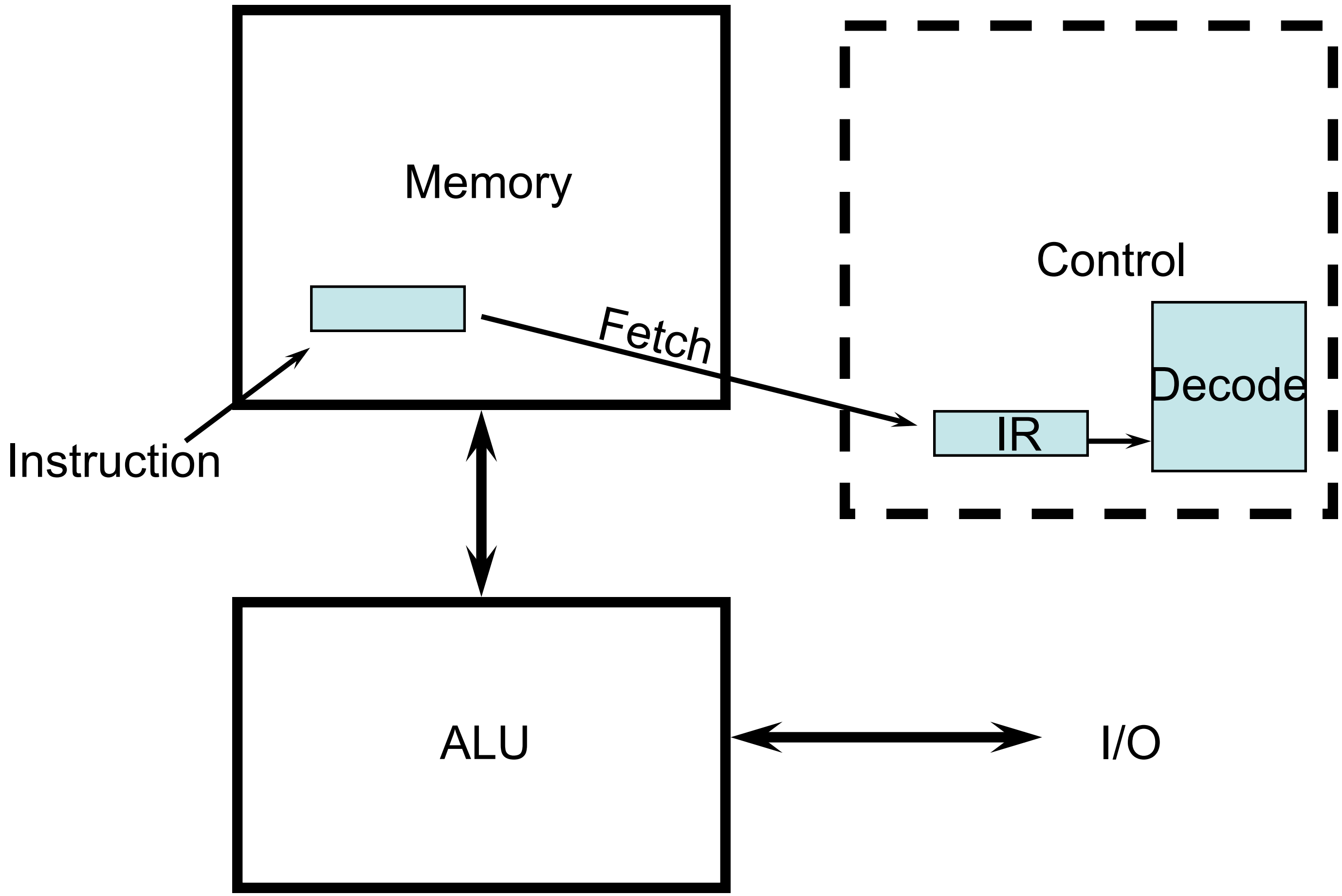
Basic Computer Architecture



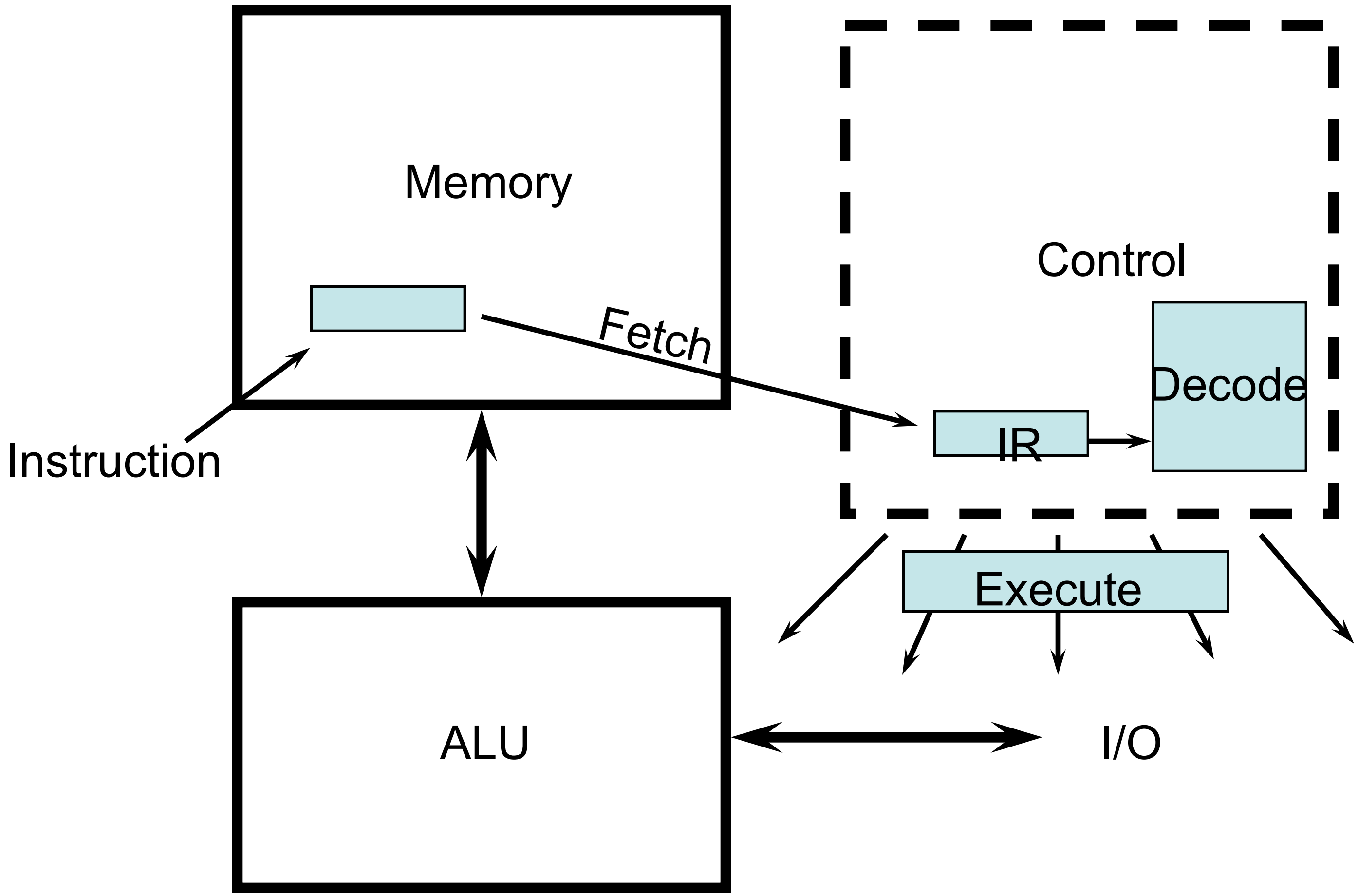
Basic Computer Architecture



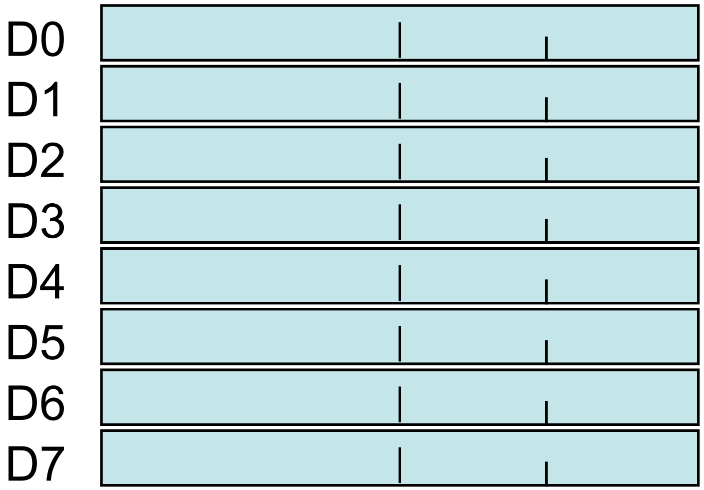
Basic Computer Architecture



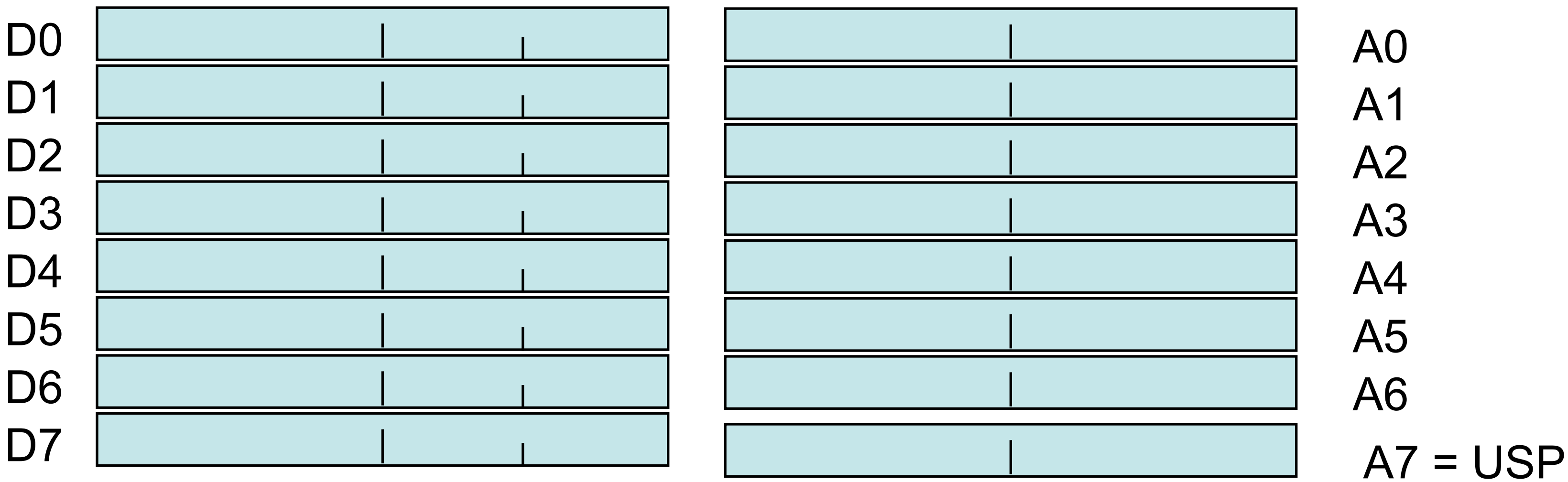
Basic Computer Architecture



ISA of 68000 Processors

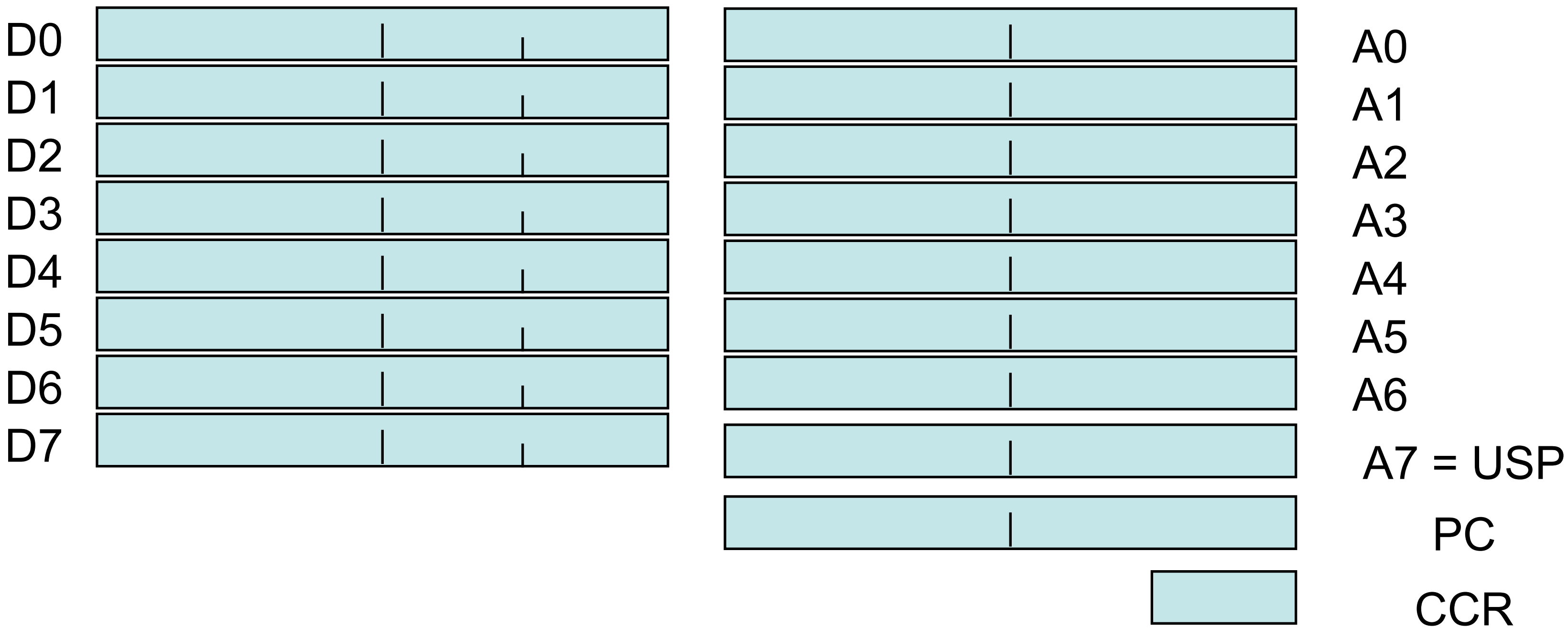


ISA of 68000 Processors

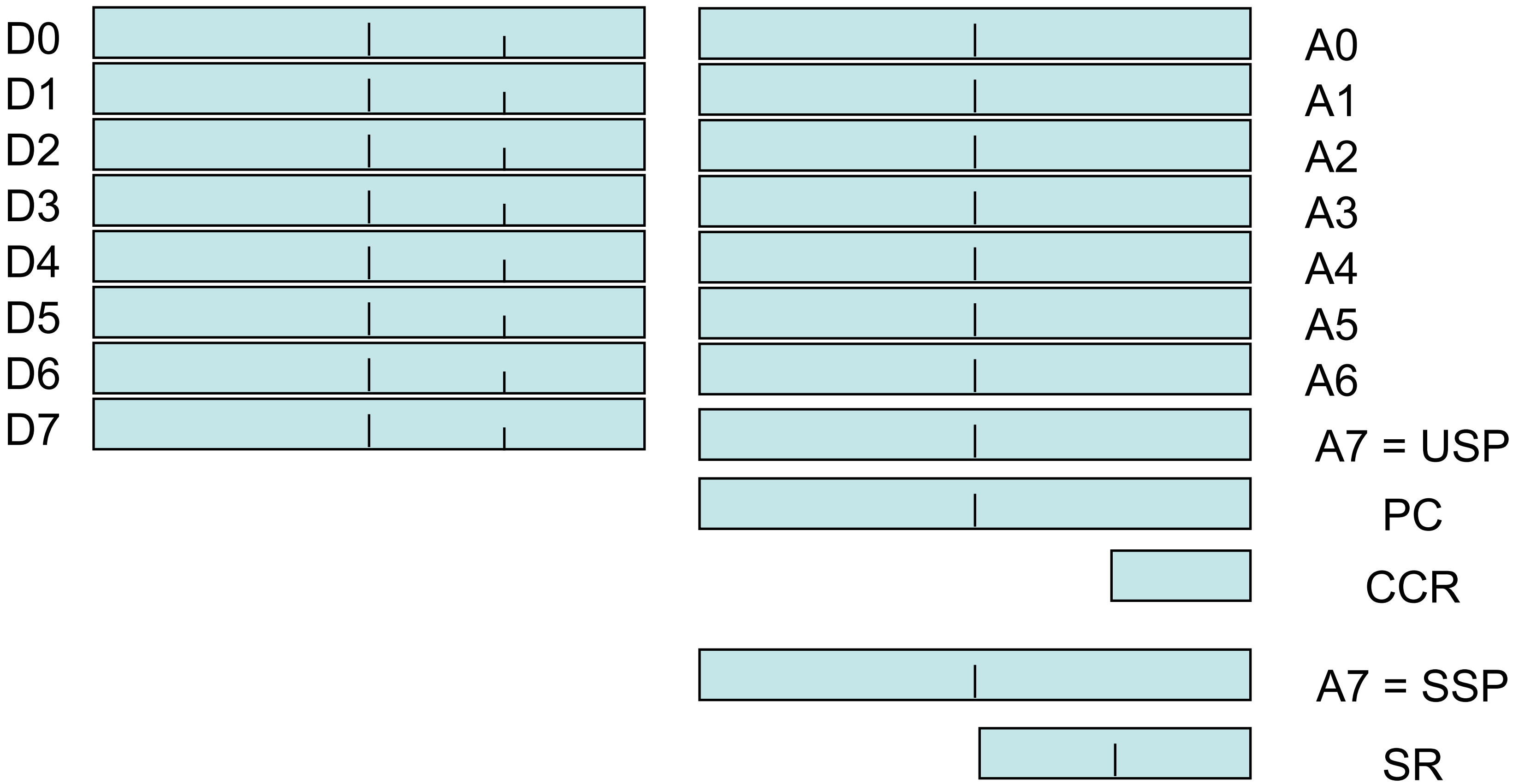


Address registers allow us to perform pointer operations in C and C++. Much faster!!!!!

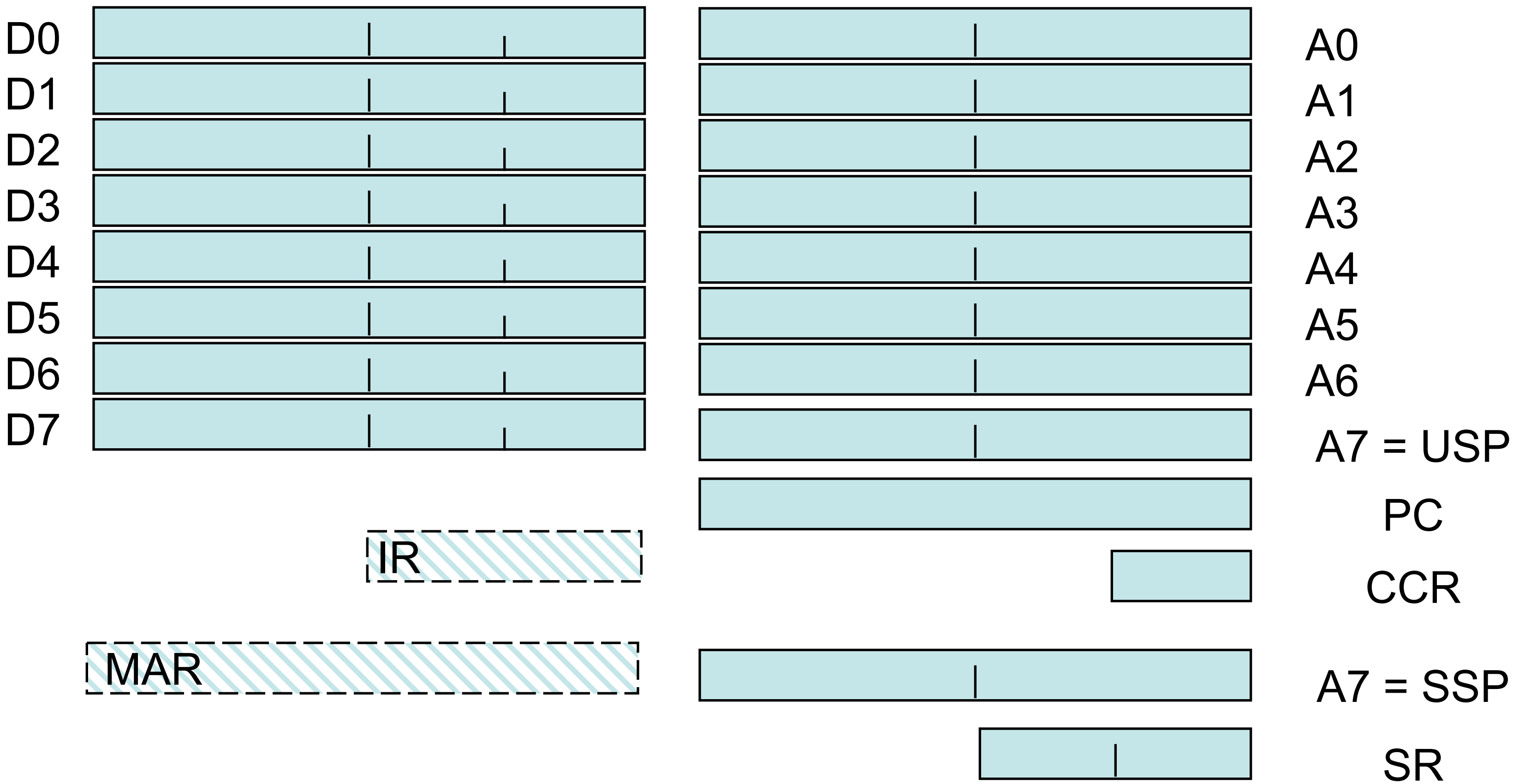
ISA of 68000 Processors



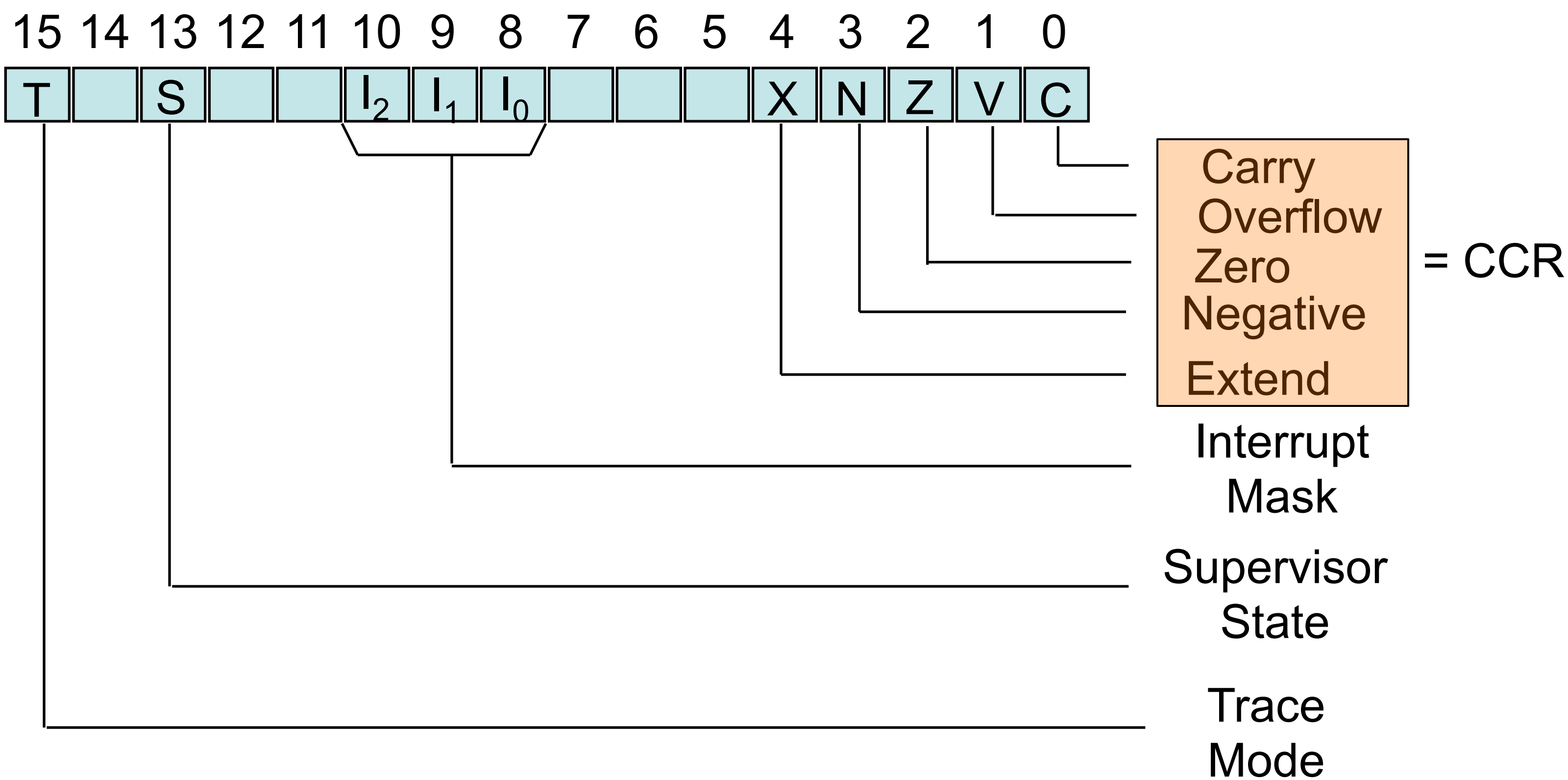
ISA of 68000 Processors



ISA of 68000 Processors



SR of 68000 Processor



SR = status register

CCR= condition code register

RTL for ADD D0, D1

Assembly Language Statement

***fetch*:** **MAR ← PC**

PC ← PC+N

IR ← M[MAR]

***decode*:**

***execute*:** **D1 ← D1 + D0**

ADD/SUB/MUL/DIV/AND/OR/EXOR

Basic Computer Organization

Cycles: Fetch

Decode

Execute

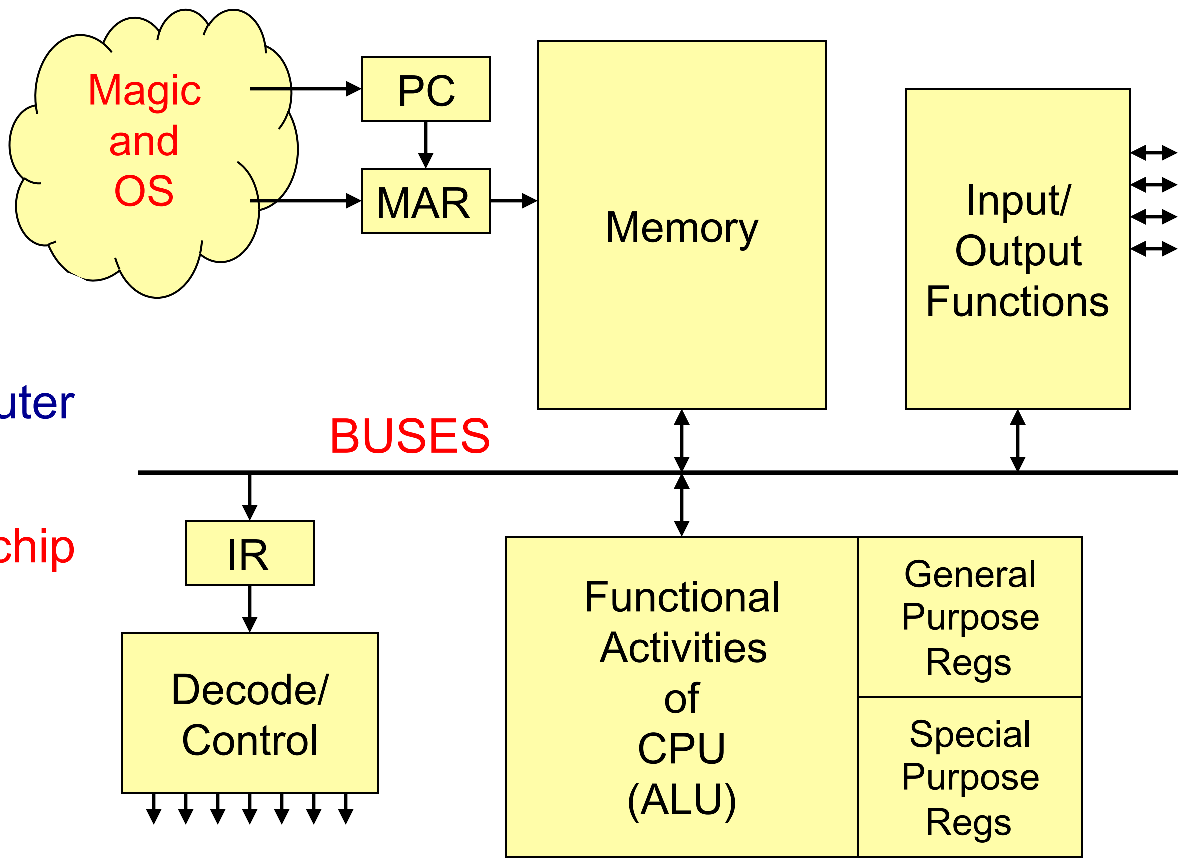
Inside the Computer

BUSES (inside the chip and outside):

Address Bus

Data Bus

Control Bus



Hardware View

Computer

Programming (Magic)

- **Computers are dumb, they do what they are told to do**
  - **The basic operations of a computer system form what is known as the instruction set**
  - **To solve a problem, you must express the solution to the problem in terms of the instructions of the particular computer**
  - **The approach or method that is used to solve a problem is known as the algorithm**
  - **With an algorithm then you translate into a program**
- **High-Level Languages**
  - **At the beginning programming was done in terms of binary numbers (octal, hexadecimal) that correspond directly to the machine instructions and locations in the computer's memory**

Computer

Programming (Magic)

- **High-Level Languages**
  - **Next technological software advance was the development of Assembly languages**
  - **Assembly language (low level language) permits the user to use symbolic names to perform various operations and to refer to the to specific memory locations**
  - **Assembler, a special program that translates the assembly language program from its symbolic format into the specific machine instructions**
  - **One-to-one correspondence between each assembly language statement and a specific machine instruction**
  - **Each microprocessor/microcontroller has its own assembly language; its own instruction set**
  - **Assembly language programs are not portable**

RTL for ADD D0, D1

Assembly Language Statement

fetch: MAR $\leftarrow$ PC

PC $\leftarrow$ PC+N

IR $\leftarrow$ M[MAR]

decode:

execute: D1 $\leftarrow$ D1 + D0

ADD/SUB/MUL/DIV/AND/OR/EXOR

Computer

Programming (Magic)

- **High-Level Languages**
  - **FORTRAN, C, C++, PASCAL, PHP, Python, Ruby, others no longer have to be concerned with the architecture of the particular computer**
  - **Operations performed by these languages are more sophisticated and far removed from the particular instruction set of the particular computer**
  - **One high-level language statement results in many different machine instructions being executed (object code)**
  - **Compiler, translates statements in a high-level language into a form that the computer can understand, that is, into the instruction set of a particular computer (object code)**
  - **Linker, combines the object code produced by the compiler from your C++ programs with the object code of routines (EX. I/O) used by your program (libraries)**

Programming (Magic) Computer

There are two primary ways for a program to be executed:

Compiler - A program that transforms source code into the machine readable code that a CPU can execute.

Ex: Programs written in C, C++, Java, Fortran and Pascal are typically executed using a compiler.

Interpreter - A program that executes source code directly, line-by-line, as each line of source code is encountered.

Ex: Programs written in Matlab, Ruby, Perl and Python are typically executed using an interpreter.

Computer

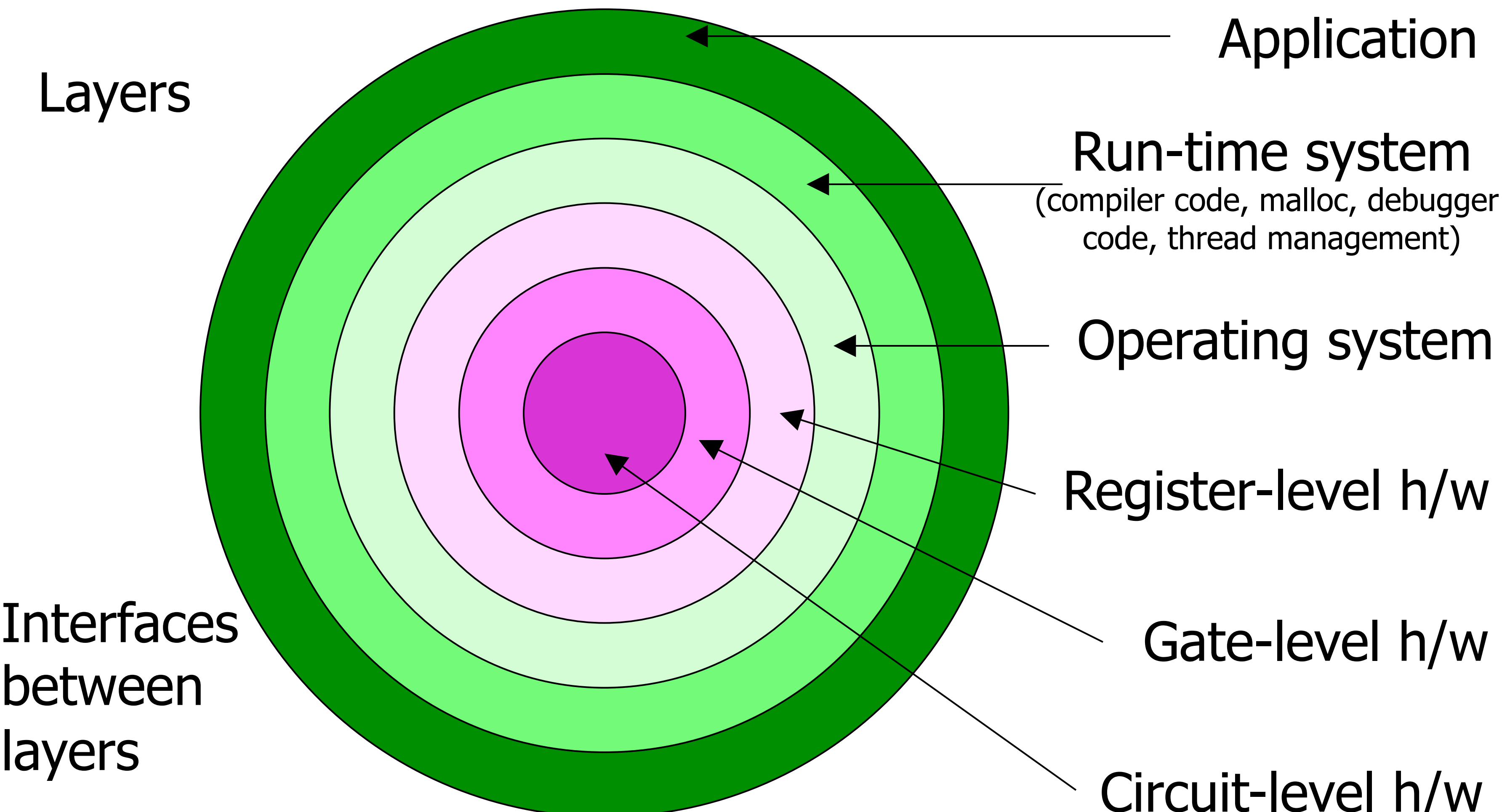
Operating Systems

- **Is a program that controls the entire operation of a computer system – Manager.**
  - **Manages input and output (I/O) operations**
  - **Manages all the resources available**
  - **Manages the execution of programs – YOURS!!!!**
  - **EX. UNIX, Windows, MAC OS, Android, others**

WEB Services

- **The Internet now has become an innovative software platform.**

Software/Hardware Organization: Another View



Types of Languages

- **Programming languages**
  - (e.g., C, C++, assembly language)
- **Scripting languages**
  - Shell scripts (OS) – controls other software programs
  - Emacs, QuakeC
- **Specification languages**
  - Used for system design/analysis
  - UML
- **Machine Code**
  - (the non human-readable form other languages are translated into, sometimes on the fly)
- **Query language**
  - SQL, OQL, XQuery
- **Markup languages**
  - typically used for producing documents
  - HTML, XML, XHTML
- **Transformation languages**
  - transform some input text in a certain formal language into a modified output text that meets some specific goal
  - XQuery
- **Template processing languages**
  - combine one or more templates with a data model to produce one or more result documents
  - Perl, Python
- **Visual programming languages**
  - Labview, Matlab
- **Hardware description languages**
  - Executables for hardware
  - Verilog, SystemC, VHDL
- **Configuration file formats (e.g., INI file)**

ECE 231 - Environments

- UNIX/LINUX: linux.ece.unm.edu OR linux.unm.edu
  - All UNM students have an account in linux.unm.edu
  - To have access to the ECE environment, linux.ece.unm.edu, you need to request an account from the ECE IT personnel
- WINDOWS (Linux like environments for your laptop)
  - CYGWIN or MinGW (my own preference)
  - Bloodshed
  - devc++
  - VisualC++
  - UBUNTU
- MAC
  - Need to install application Xcode to have Unix
  - Application Terminal is a Unix environment

ECE 231 - Environments

- UNIX/LINUX: linux.ece.unm.edu
  - This is the ECE working environment
    - SHELL – interpreter of OS commands
    - .alias – setup your environment to your liking and needs
  - This is the preferred environment where your code will be written, compiled, debugged, executed, and tested
  - Compilers used: gcc for C; g++ for C++
  - Use putty to access the system from a Windows machine; it is an application that runs on top of Windows and emulates a terminal. Through the terminal application you can access the UNIX systems at UNM
  - Use ssh, scp, and sftp to access it from any machine (including Windows/Cygwin or Windows/MinGW). Google(ssh) for usage details.

Setting Up Your ECE Linux Environment

- There are many OS SHELLs
  - I prefer the shell bash, which is the default
    - To change to the C shell execute the command csh
- Setup your environment
  1) Login to your UNIX/LINUX account and create a sub-directory named ece231.
     - You do this by executing the command   mkdir ece231
  2) Go into that sub-directory
     - You do this by executing the command   cd ece231
  3) Create another sub-directory called Ccode
     - You do this by executing the command mkdir Ccode
       - » Download all the C code examples given to you into this sub-directory.
       - » These examples are given to you so you can refresh your C programming skills

Setting Up Your ECE Linux Environment

- Setup your environment

  4) Create another sub-directory called CPPcode
     - You do this by executing the command mkdir CPPcode
       - » In this directory you will develop all the code in C++

  5) .alias file allow you to create short cuts to commands and to setup your UNIX environment

  6) In your home directory create a .alias file that has the aliases below.
     - (Copy/download .alias file from my shared folder if you like. You can add your own aliases and commands by editing this file.
     - The aliases provided below will need to be modified to reflect your UNIX environment, especially the first two aliases.
     - Add the line "source ~/.alias" to your .bashrc file.

  7) After setting up your .alias file, run ". .bashrc".

Setting Up Your ECE Linux Environment

- Sample .alias file

```
# Use these for convenience
alias lss='ls -al'
alias 231='cd ~/ece231'
alias ccode='cd ~/ece231/CCode'
alias cppcode='cd ~/ece231/CPPCode'

# Make compilers more helpful
alias gcc='gcc -std=c99 -Wall -Werror'
alias g++='g++ -Wall -Werror'

# Use these for safety
alias rm='rm -i'
alias cp='cp -i'
alias mv='mv -i'
```

Setting Up Your ECE Linux Environment

- Download from BBLearn the file unixman.txt for a quick reference to UNIX OS commands
- Download from BBLearn the file vim-cheat-sheet.txt for a quick reference to the editor vi commands. (vi is implemented with Vim.)

Q&A