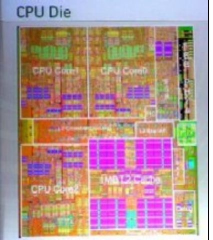



COMP 273
Introduction to Computer Systems

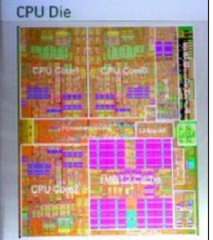
# COMP 273

## Micro Architecture

### Part 1 – Classical

Lectures 8 and 9a

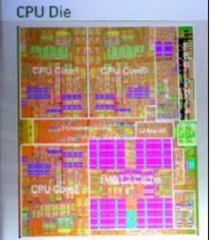
Prof. Joseph Vybihal






# Announcements

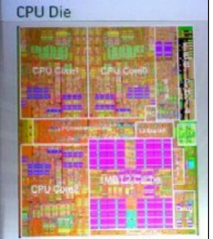
- Assignment #2 out on Friday




COMP 273
Introduction to Computer Systems

# Readings

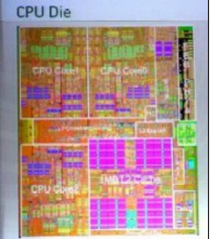
* Our textbook
    - Chapter 4
    - Appendix D
* Web Resources:
    – http://en.wikipedia.org/wiki/Microarchitecture
    – http://www.hardwaresecrets.com/article/313






# At Home

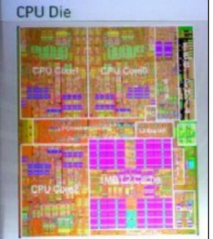
- Study the programs shown in class today and execute them by hand using a simple CPU schematic.
  - A key skill to have as an assembler programmer!!
- Soul Of A New Machine






# Part 1

## The Von Neumann Machine

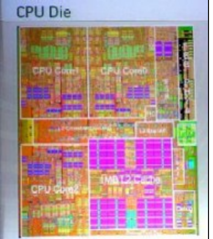





# The Von Neumann Machine

- Traditional computer model proposed by John Von Neumann in 1946

Based on Turing's theoretical ideas.



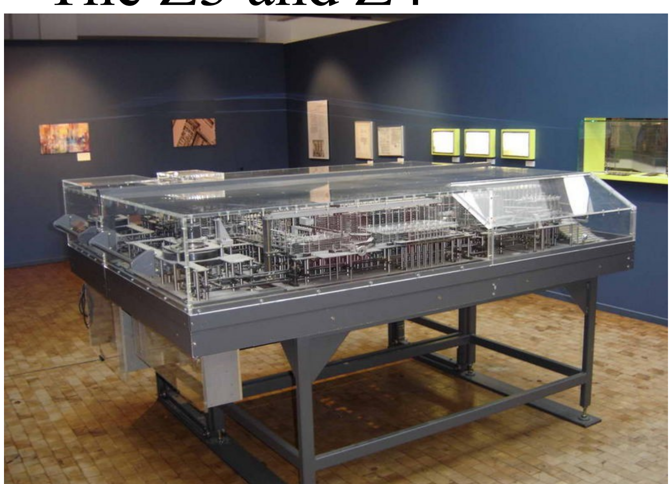


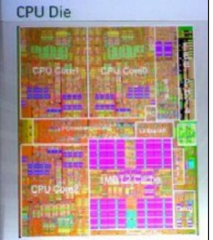

John Atanasoff

# Before Von Neumann

Konrad Zuse
1936 – Germany
The Z3 and Z4

J. Presper Eckert and John Mauchly
ENIAC – 1946 USA
Electronic Numerical Integrator And Computer







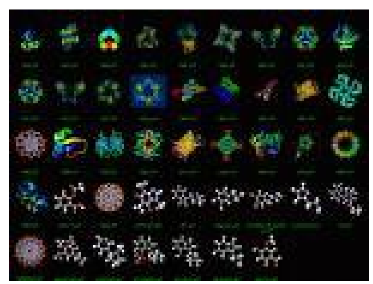

# The Von Neumann Idea

Instructions

Used instructions

Arm stores results here

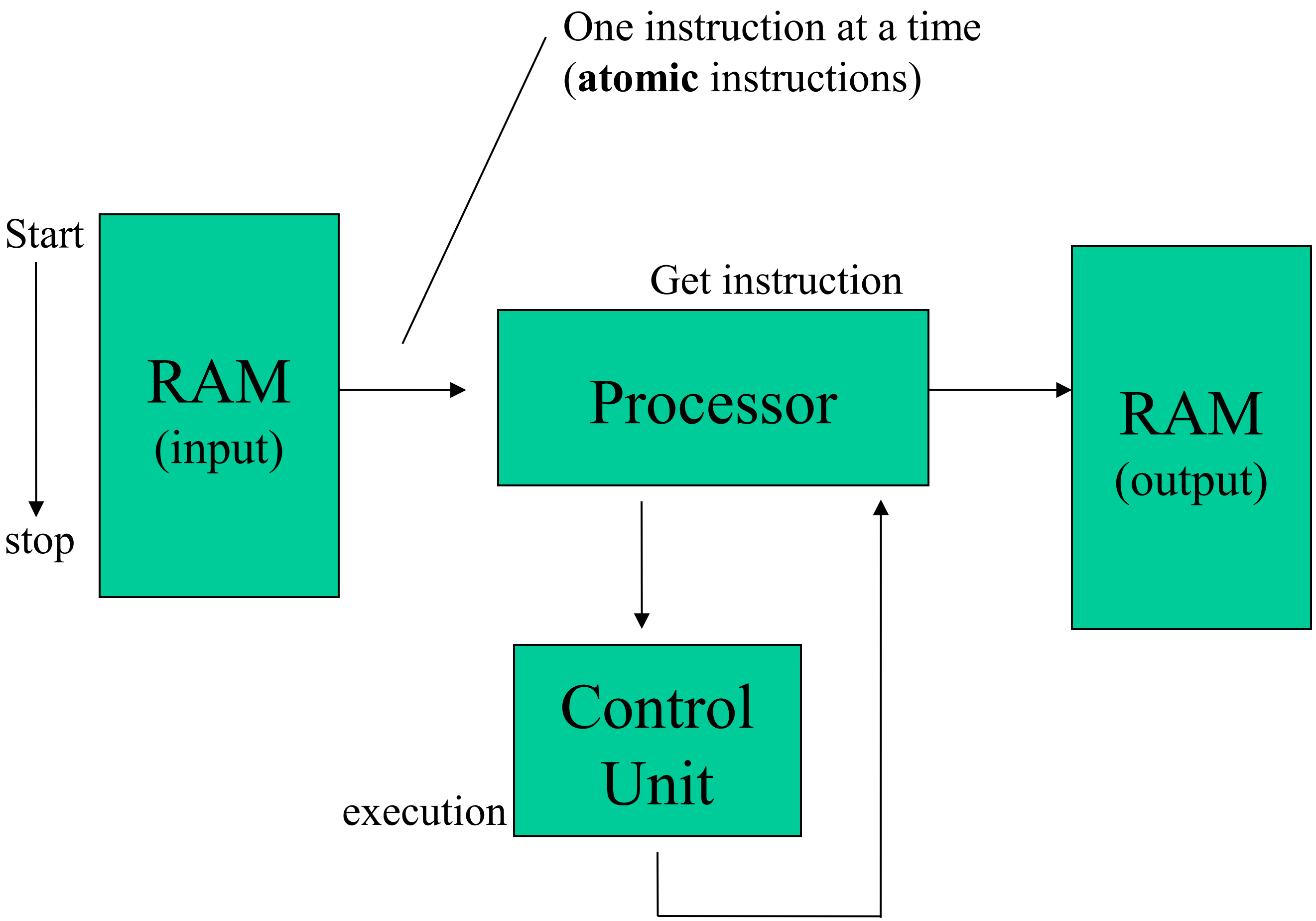
Arm takes 1 **atomic** instruction

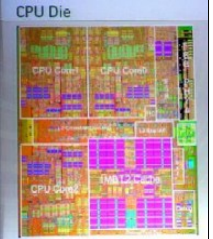


Match instruction with a pattern causes some action to occur.

Random Access Memory

# The Von Neumann Machine

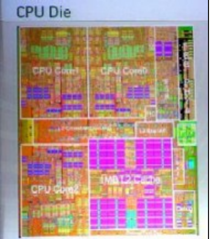







# Part 2

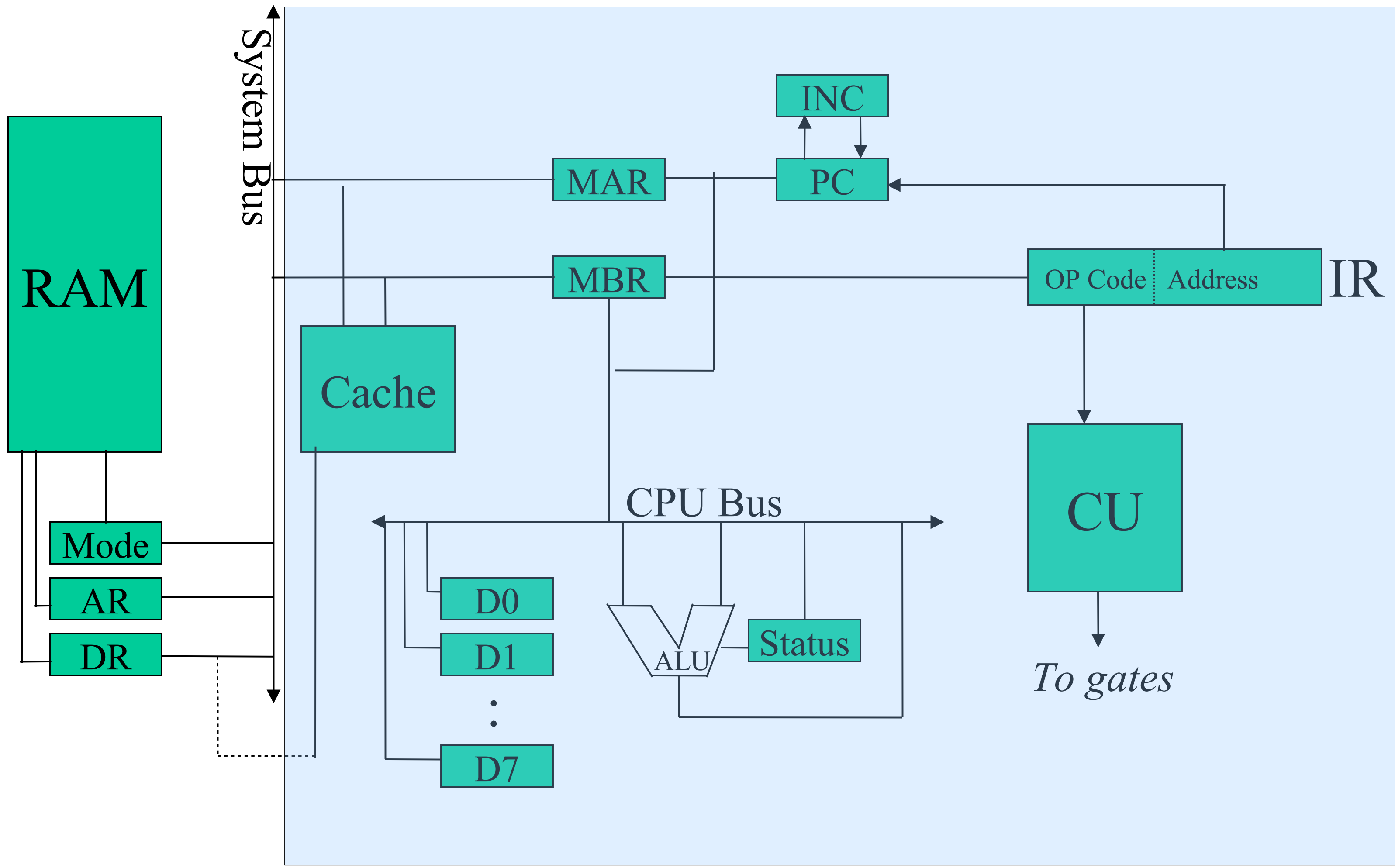
## Classical CPU Architecture & Processing





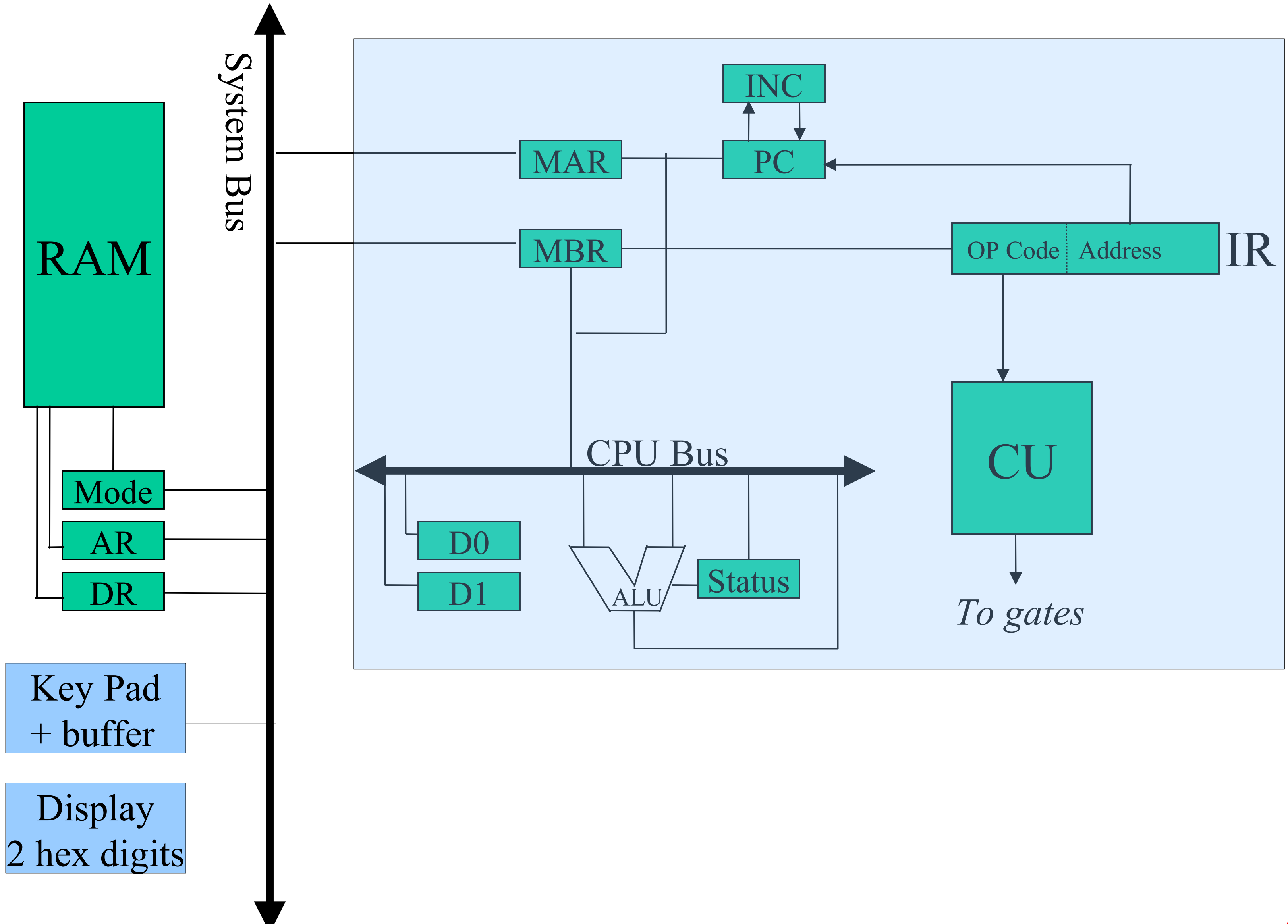


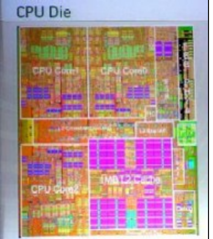

# Classical CPU Design




MAR - memory address register, MBR - memory buffer register, D0-7 Registers, ALU - arithmetic logic unit, INC - incrementer, PC - program counter, IR - instruction register, CU - control unit

# Classical CPU Design

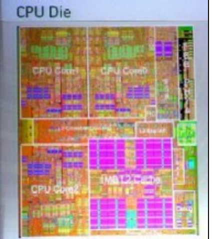







# Example

- Sample classical CPU execution of instructions from RAM.
  - ADD R1,R2,R3
  - GOTO #1000






# CPU Execution Cycle

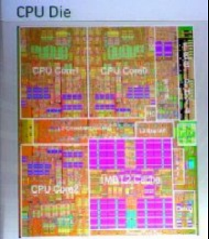
1. **FETCH**
   - 1.1 PC to MAR & PC++
   - 1.2 MAR to RAM AR
   - 1.3 Read signal
   - 1.4 RAM DR to MBR then to IR (or routed to some other register)

2. **DECODE**
   - 2.1 NEED OPERANDS?
     - 2.1.1 FETCH OPERANDS
       - Use address in instruction to do FETCH for each parameter
       - But step 1.4 ends at a CPU register and not the IR
       - And step 1.1 has no PC++ action
   - 2.2 OP-CODE to CU
3. **EXECUTE**
   - 3.1 CU triggers gates to perform the instruction

4. **STORE RESULT**
   - 4.1 Register with address to MAR
   - 4.2 Register with data to MBR
   - 4.3 Write signal sent to RAM AR and DR

# Micro Instruction

- A way to express the operation of a CPU step by step.

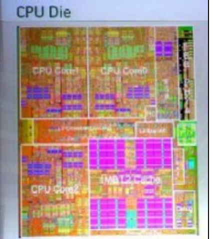





# Micro Instruction

- A way to express the operation of a CPU step by step.

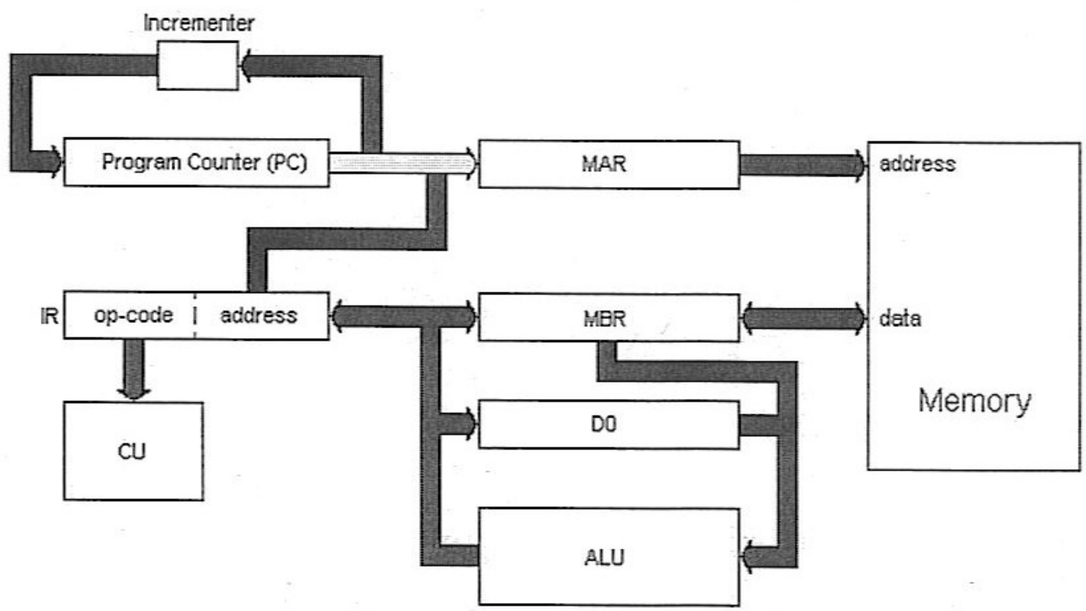
- Syntax:
  - CHANGE_STATE ← OPERATION
    - [MACHINE]
    - [MACHINE(OFFSET)]
    - Constant
    - Operators: +, -, *, /



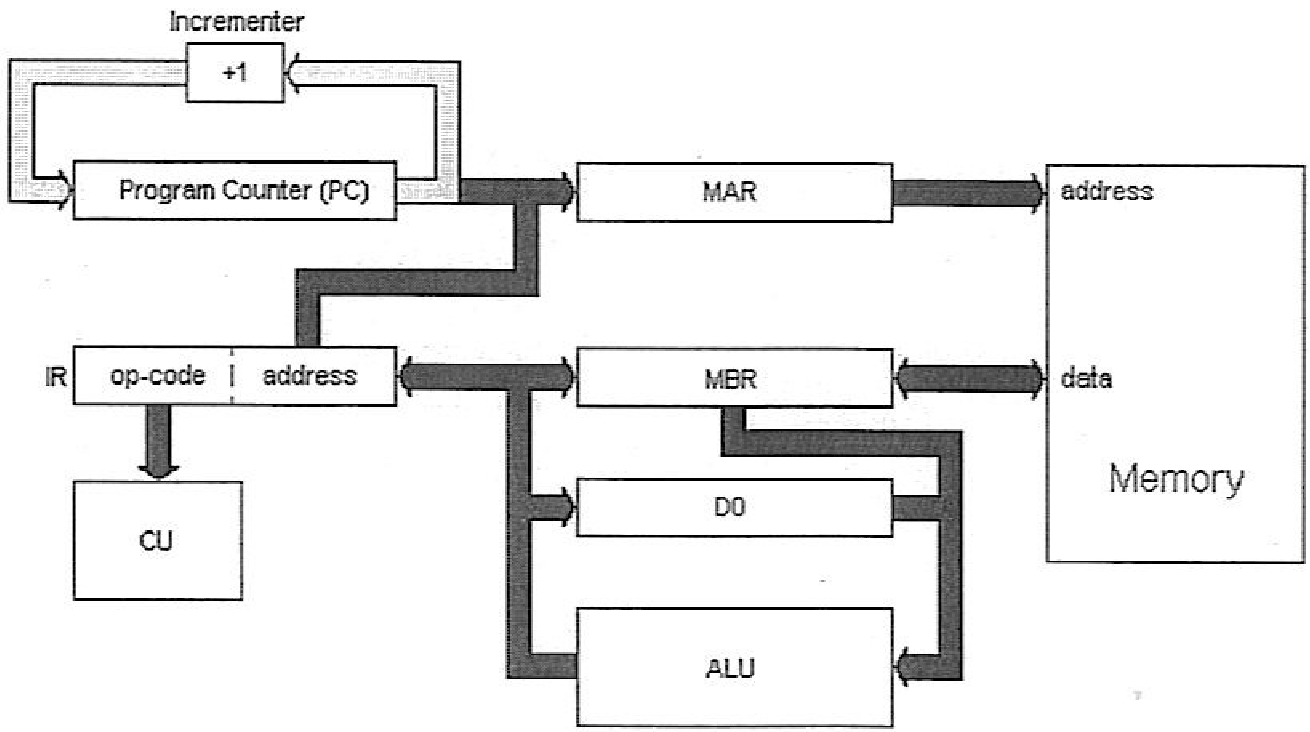

```
FETCH    [MAR] <- [PC]
         [PC] <- [PC] + 1
         [MBR] <- [M([MAR])]
         [IR] <- [MBR]

EXECUTE  [MAR] <- [IR(Address_Field)]
         [MBR] <- [M([MAR])]
         [D0] <- [D0] + [MBR]
```

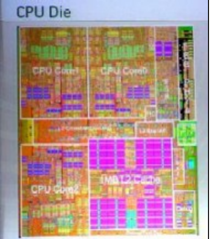
eg
ADD D0,D0,X

```
1)  [MAR] <- [PC]
```



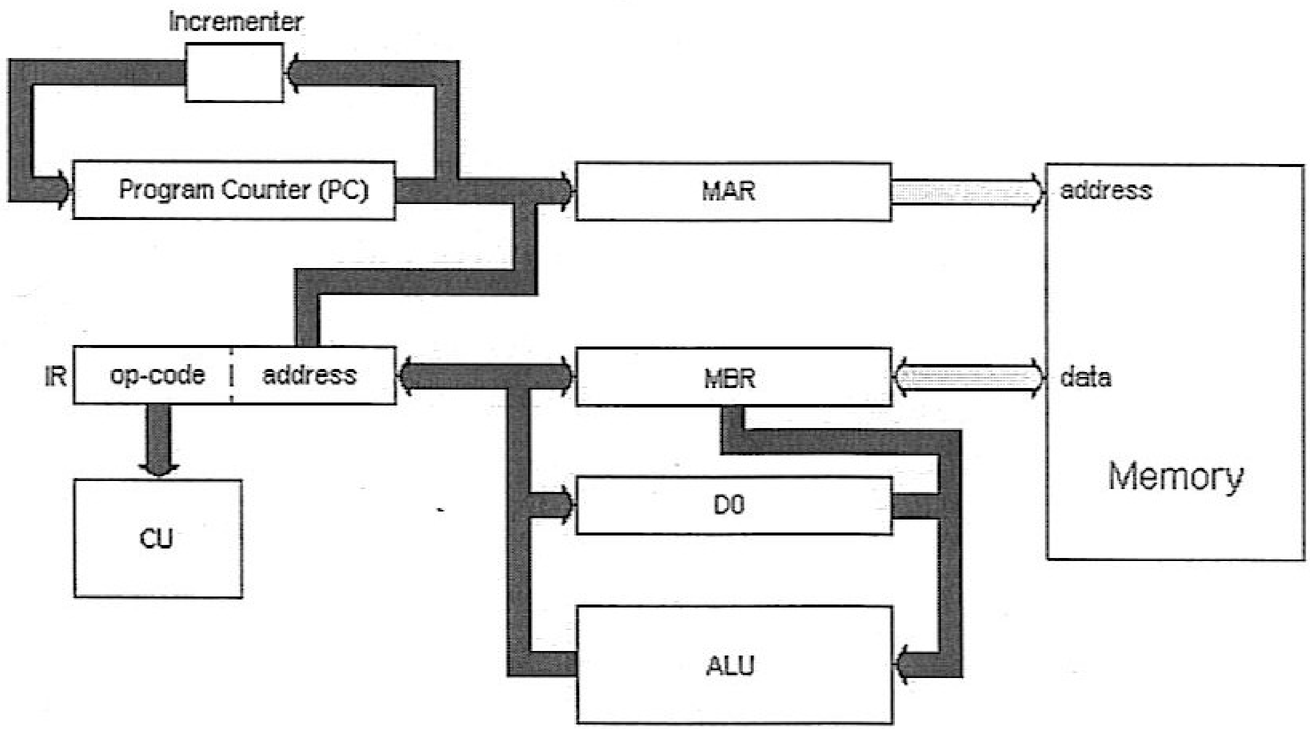
2) [PC] <- [PC] + 1



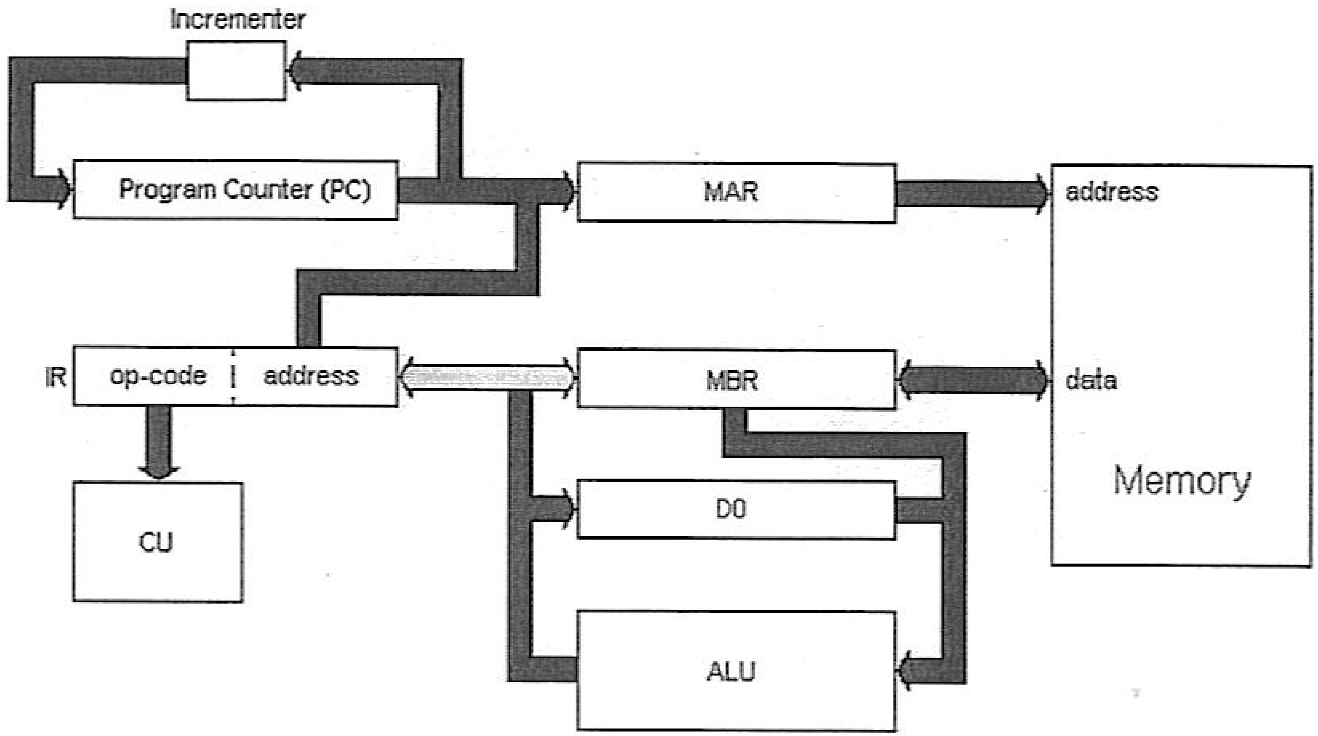




COMP 273
Introduction to Computer Systems

3) [MBR] <- [M([MAR])]


Incrementer
Program Counter (PC)
MAR
address
IR
op-code
address
MBR
data
Memory
D0
CU
ALU

4) [IR] <- [MBR]

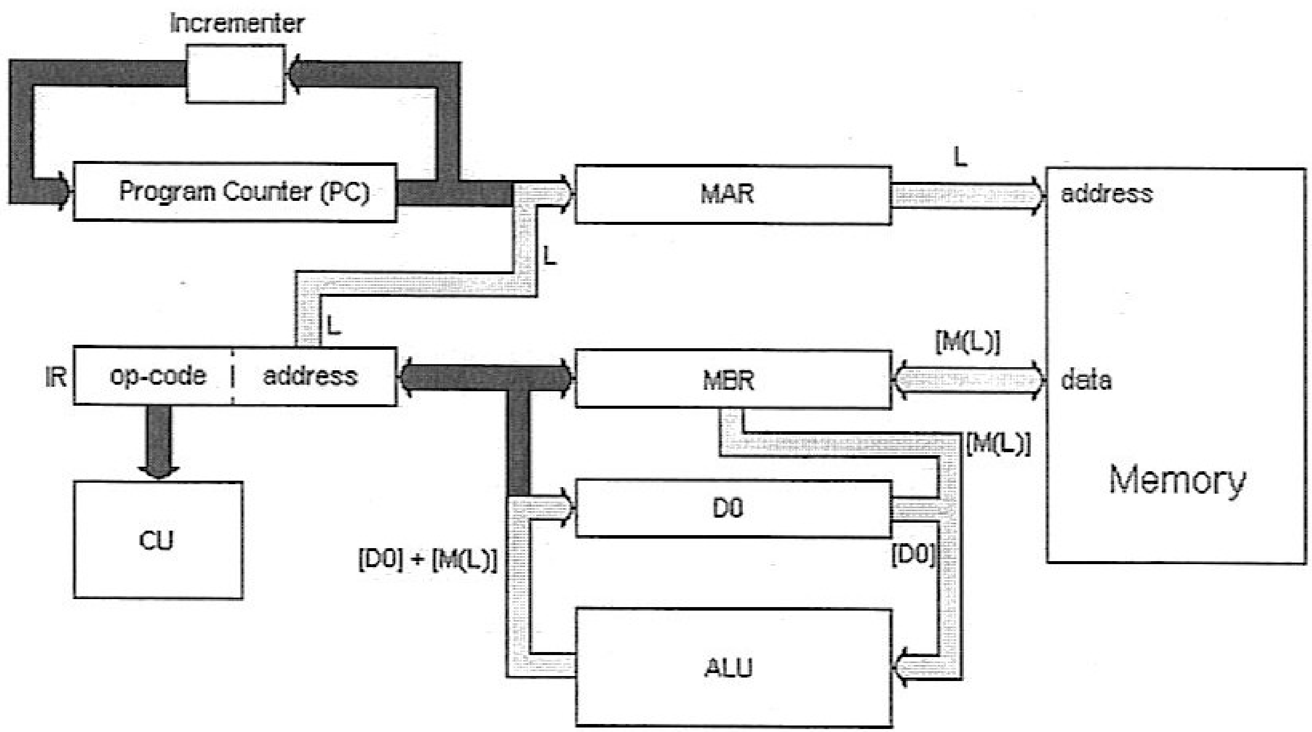


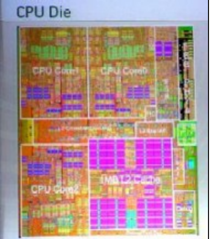



COMP 273
Introduction to Computer Systems

5) The execution phase:

```
[MAR] <- [IR(Address_Field)]
[MBR] <- [M([MAR])]
[D0] <- [D0] + [MBR]
```








# About Modern Memory

- Addressability
  - Does the machine you want to access have an address?
- Accessibility
  - A circuit pathway exists from point A to B
  - The machine is not addressable only accessible.
    - Often via a custom instruction

---

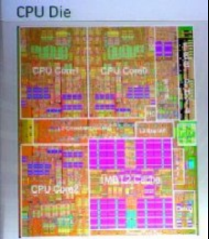
- Byte
  - The smallest addressable space in a computer (8-bits, RAM)
- Word
  - The “standard operational” bit-size for the computer (arbitrary)

---

- Registers
  - In either Byte or Word sizes, or specialty sizes.

---

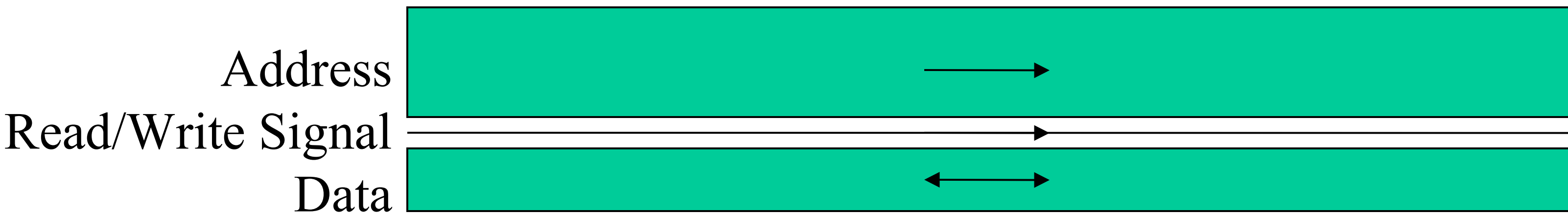
- Ports, slots, and other important registers
  - Often “accessible” but not “addressable”.





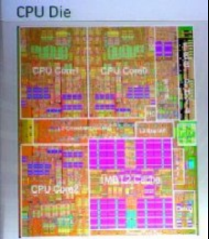

# The Bus

- **BUS** - a conduit for bytes to travel from one location to another location (pathway)



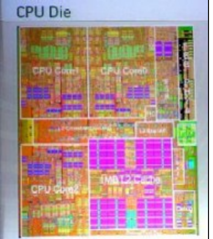

Note:

- 1 bit per wire
- bottle neck, only 1 byte of data per movement
- Duality of operation: byte and word modes
  - The above diagram shows a byte construction




# Modern Memory Types

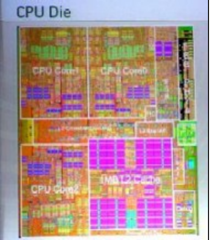
- RAM - Random Access Memory (Primary storage)
  - DRAM: Dynamic (needs to be refreshed)
  - SRAM: Static (no refresh needed)
- ROM - Read Only Memory (advanced wired instructions, PAL/PLA)
  - ROM - hardwired
  - PROM - Programmable once (fuses)
  - EPROM - Erasable / Re-programmable (chemically by heat)
  - EEPROM - Electrically erasable
- Cache - Very fast memory (often on CPU)
  - Used to store frequently accessed info from RAM
  - Bypasses system bus
  - Uni-directional, from Cache1 to CPU / from CPU to Cache2
- Pipeline (processing instructions using an assembly line)
  - Partial parallel execution of instructions
  - A series of instruction sized memory registers in an assembly line






# Part 3

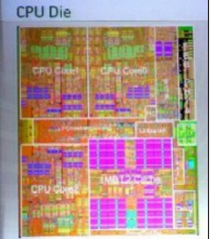
## What is an instruction?






# What is an instruction?

- An atomic command
- A formatted string of binary
- A CPU can only execute one command at a time
- Instructions are stored in RAM
- CPU downloads 1 instruction at a time and executes that instruction
- Programs / algorithms are made out of these instructions

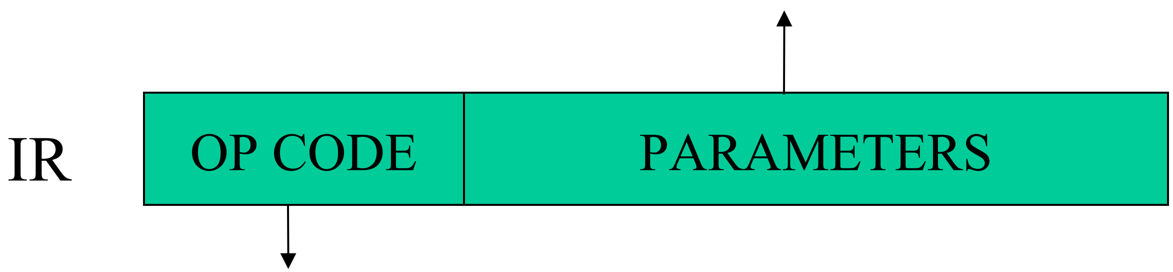




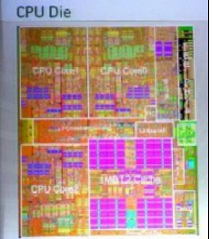

# What is an instruction?

## Instruction Format

<u>Source and destination path codes</u>
containing either:
Register numbers, literals, or addresses




A binary tabulated code
(often represented as a Mnemonic in assembler)






# What is an instruction?
## Example Mnemonic Instructions

IR

| OP CODE | PARAMETERS |
| --- | --- |

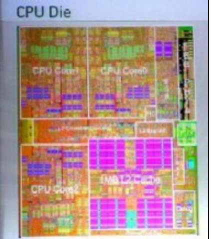
Instruction:

lw $2, ($3)   load a word into register 2 from the address stored in register 3.

mnemonic
(op code)

lw $2, #1A2F   load a word into register 2 from the hex address 1A2F

parameters






# What is an instruction?

## Parameters

IR

| | |
|---|---|
| lw indirect | 2 3 |

IR

| | |
|---|---|
| lw immediate | 2 1A2F |

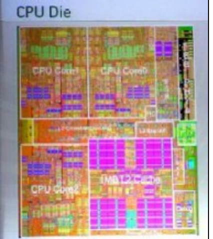
Mnemonics:

lw $2, ($3) — load a word into register 2 from the address stored in register 3.

mnemonic

lw $2, #1A2F — load a word into register 2 from the hex address 1A2F

parameters






# What is an instruction?

## Op code variations

Same Different

| | | |
|---|---|---|
| IR | 1001 000 | 00010 00011 |
| IR | 1001 001 | 00010 1A2F |

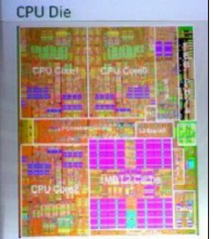
Mnemonics:

mnemonic

lw $2, ($3) load a word into register 2 from the address stored in register 3.

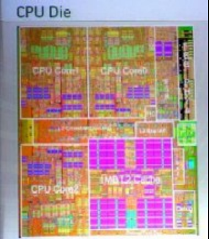
lw $2, #1A2F load a word into register 2 from the hex address 1A2F

parameters





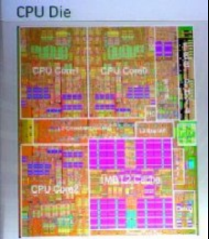

```
##
## hello.a - prints out "hello world"
##
##        a0 - points to the string
##

#################################################
#                                               #
#               text segment                    #
#                                               #
#################################################

        .text
        .globl __start
__start:                # execution starts here
        la $a0,str      # put string address into a0
        li $v0,4        # system call to print
        syscall         # out a string

        li $v0,10
        syscall         # au revoir...


#################################################
#                                               #
#               data segment                    #
#                                               #
#################################################

        .data
str:    .asciiz "hello world\n"

##
## end of file hello.a
```






# Question

- What is the physical representation of the previous program in RAM?
- How does the OS form the connection between the program and the CPU?
    - Jump buffer, deamons, Interrupts
- Execute previous program by hand
    - Assume 10 GP registers
    - Assume IP, IR, ALU
    - Assume memory registers: address, data, status






# Software, Memory & OS

Ways to access OS
- Vector
- Deamon
- Interrupt

CPU

Single core

Slots / ports

OS

Free space

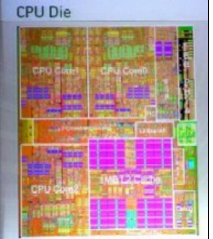
Heap/Stack (dynamic)

.text .data (static)

Free space

Zero page

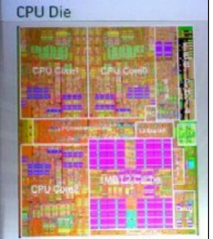
RAM






# Software, Memory & OS

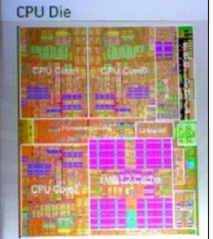
- Operating System
  - Vector: a public array of pointers to private OS functions the programmer can invoke
  - Deamon: a program running in memory that is public to the programmer and has been delegated by the OS some task.
  - Interrupt: a method by which a program can request the OS to perform a task switch
    - Program sleeps, OS does a task on cpu, program wakes up






# Software, Memory & OS

- Your program in RAM
  - Static portion
    - The actual code
    - The .data area, created at compile time and exists for the entire life of the program.
  - Dynamic portion
    - The run-time stack – local variables
    - The heap – malloc and new






## What does this program do?

```
##
## length.a - prints out the length of character
## string "str".
##
##      t0 - holds each byte from string in turn
##      t1 - contains count of characters
##      t2 - points to the string
##

##############################################
#                                            #
#               text segment                 #
#                                            #
##############################################

        .text
        .globl __start
__start:                # execution starts here
        la $t2,str      # t2 points to the string
        li $t1,0        # t1 holds the count
nextCh: lb $t0,($t2)    # get a byte from string
        beqz $t0,strEnd # zero means end of string
        add $t1,$t1,1   # increment count
        add $t2,1       # move pointer one character
        j nextCh        # go round the loop again

strEnd: la $a0,ans      # system call to print
        li $v0,4        # out a message
        syscall

        move $a0,$t1    # system call to print
        li $v0,1        # out the length worked out
        syscall

        la $a0,endl     # system call to print
        li $v0,4        # out a newline
        syscall

        li $v0,10
        syscall         # au revoir...
```

```
##############################################
#                                            #
#               data segment                 #
#                                            #
##############################################

        .data
str:    .asciiz "hello world"
ans:    .asciiz "Length is "
endl:   .asciiz "\n"

##
## end of file length.a
```