C

C

- What is C?

  C is a general-purpose programming language used to write efficient, fast programs. It's especially useful for programming hardware, operating systems, and embedded systems

Why learn to code?

- Automate stuff
- Data analysis
- Running simulations
- Make more MONEY!

Why learn C?

- Used in microcontrollers, Arduino, and system programming
- Helps understand memory, bits, and performance
- It's close to the hardware (low-level access)

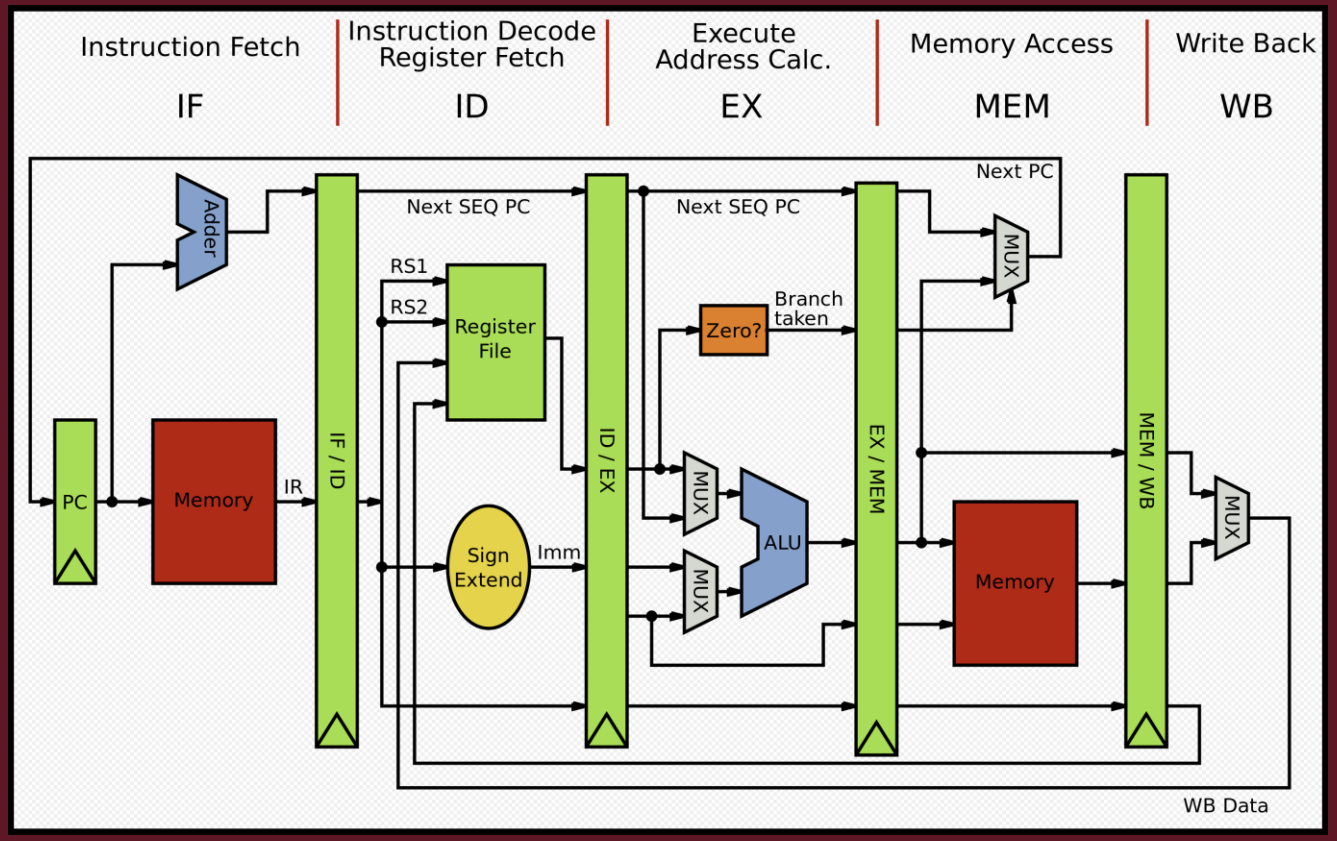
Why C?

- Nobody wants to deal with assembly!

Assembly

```
1  .global _start
2  _start:
3      MOV R0, #0xFFFFFFFF
4      MOV R1, #5
5      MOV R2, #4
6      MOV R3, #3
7
8      ADDS R4,R0,R1
9      ADC R5,R2,R3
```

MIPS Architecture (for Pipelining)



Machine Code

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

```
add $t0, $s1, $s2   (refer to Page 25)
```

special	$s1	$s2	$t0	0	add
0	17	18	8	0	32
000000	10001	10010	01000	00000	100000

MIPS

- Show MIPs reference sheet

C Code

```
#include <stdio.h>

int main() {
    int a = 5;
    int b = 7;
    int c = a + b;
return 0;
}
```

Assembly Code

```
movl    $5, -4(%rbp)    ; store 5 in variable a
movl    $7, -8(%rbp)    ; store 7 in variable b
movl    -4(%rbp), %eax  ; move a into eax
addl    -8(%rbp), %eax  ; add b to eax
movl    %eax, -12(%rbp) ; store result in c
```

Use an online C compiler

- Search “Online C Compiler”

Structure of a C Program

#include <stdio.h> - Includes standard input/output

int main() { - Starting point of every C program

printf("Hello, world!\n"); - Function to display output

return 0; - Ends the program

}

What are data types?

- int – Integer
- char – Character
- float – Number with a decimal

Datatypes example

- int a = 5;
- char c = 'A';
- float pi = 3.14;

  Mention uint16

Printing Datatypes

```
int a = 10;
float pi = 3.14;
char x = 'A';
char abc[] = "Hello";

printf(“a has the value %d\n", a);
printf(“pi has the value %f\n", pi);
printf(“x has the value %c\n", x);
printf("String: %s\n", abc);
```

Input Datatypes (scanf)

```
#include <stdio.h>

int main() {
    char name[50];

    printf("Enter your name: ");
    scanf("%s", name);

    printf("Hello, %s!\n", name);

    return 0;
}
```

Operators

- Arithmatic - +, -, *, /
  a = 5 * c;
- Comparison - ==, !=, <, >
  a = b < 5
- Logical - && (AND), || (OR), ! (NOT)
  if (a > 0 || b > 0)
- Bitwise - & (AND), | (OR), << LEFTSHIFT
  a = b << 2

Flow Control

```
if (a > 0) {
    printf("Positive\n");
} else {
    printf("Zero or negative\n");
}
```

Functions

- What is a function?
  A function is a block of code that performs a specific task.

- You can call a function over and over again with different input to repeat the same task

Example

```
#include <stdio.h>

// Function declaration
int add(int a, int b);

int main() {
    int num1, num2, sum;

    // Input two numbers
    printf("Enter first number: ");
    scanf("%d", &num1);

    printf("Enter second number: ");
    scanf("%d", &num2);

    // Function call
    sum = add(num1, num2);

    // Output result
    printf("Sum = %d\n", sum);

    return 0;
}

// Function definition
int add(int a, int b) {
    return a + b;
}
```

Arrays

- What is an array?
  An array is a collection of variables of the same type stored in contiguous memory

  ```
  int mydata[3] = {10, 20, 30};
  printf("%d", mydata[1]); // Output: 20
  ```

Loops

- Run your code multiple times
- For loop:

```
int i;

for (i = 0; i <= 10; i = i + 2) {
        printf("%d\n", i);
}
```

Loops

- While loop:

```
int i = 0;

while (i < 5) {
    printf("%d\n", i);
    i++;
}
```

Loop with arrays

```
int numbers[] = {10, 20, 30, 40, 50};
int size = sizeof(numbers) /
sizeof(numbers[0]); // Calculate array size


for (int i = 0; i < size; i++) {
printf("Element at index %d: %d\n", i,
numbers[i]);
}
```

Header files

- Stdio.h
- Math.h
- String.h

Quick Python Demo

- name = input("Enter your name: ")
- print("Hello, " + name + "!")

C Code

```
#include <stdio.h>

int main() {
    int a = 5;
    int b = 7;
    int c = a + b;
return 0;
}
```

Assembly Code

```
movl    $5, -4(%rbp)    ; store 5 in variable a
movl    $7, -8(%rbp)    ; store 7 in variable b
movl    -4(%rbp), %eax  ; move a into eax
addl    -8(%rbp), %eax  ; add b to eax
movl    %eax, -12(%rbp) ; store result in c
```

Python Code

```
a = 5
b = 7
c = a + b
```

Python Byte Code

```
1           0 LOAD_CONST               0 (5)
            2 STORE_NAME               0 (a)

 2          4 LOAD_CONST               1 (7)
            6 STORE_NAME               1 (b)

 3          8 LOAD_NAME                0 (a)
           10 LOAD_NAME                1 (b)
           12 BINARY_ADD
           14 STORE_NAME               2 (c)

 4         16 LOAD_NAME                3 (print)
           18 LOAD_NAME                2 (c)
           20 CALL_FUNCTION            1
           22 POP_TOP
           24 LOAD_CONST               2 (None)
           26 RETURN_VALUE
```

Steps

Your Python Code

↓

Python Bytecode

↓

C functions in Python VM (like PyNumber_Add)

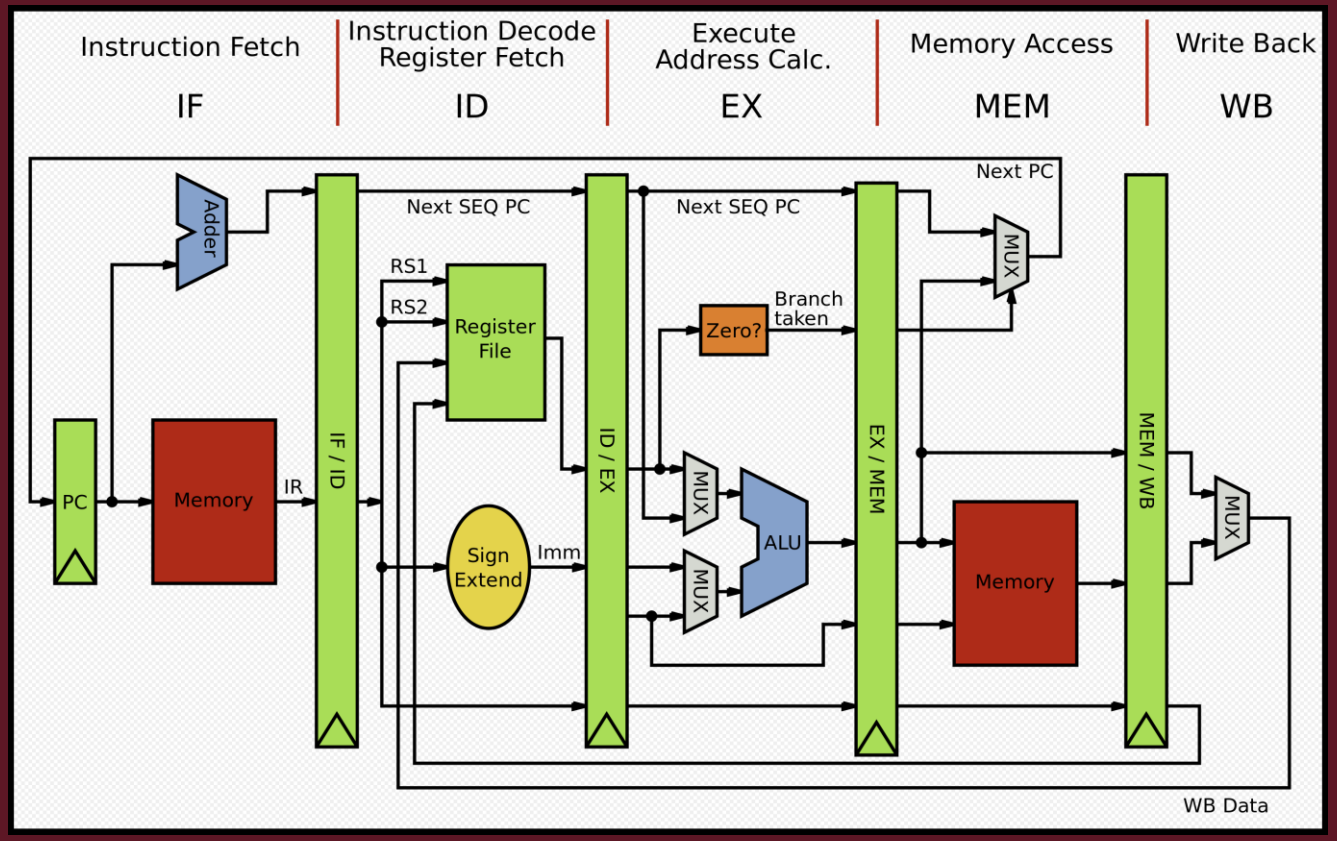
↓

Machine Code (compiled from C)

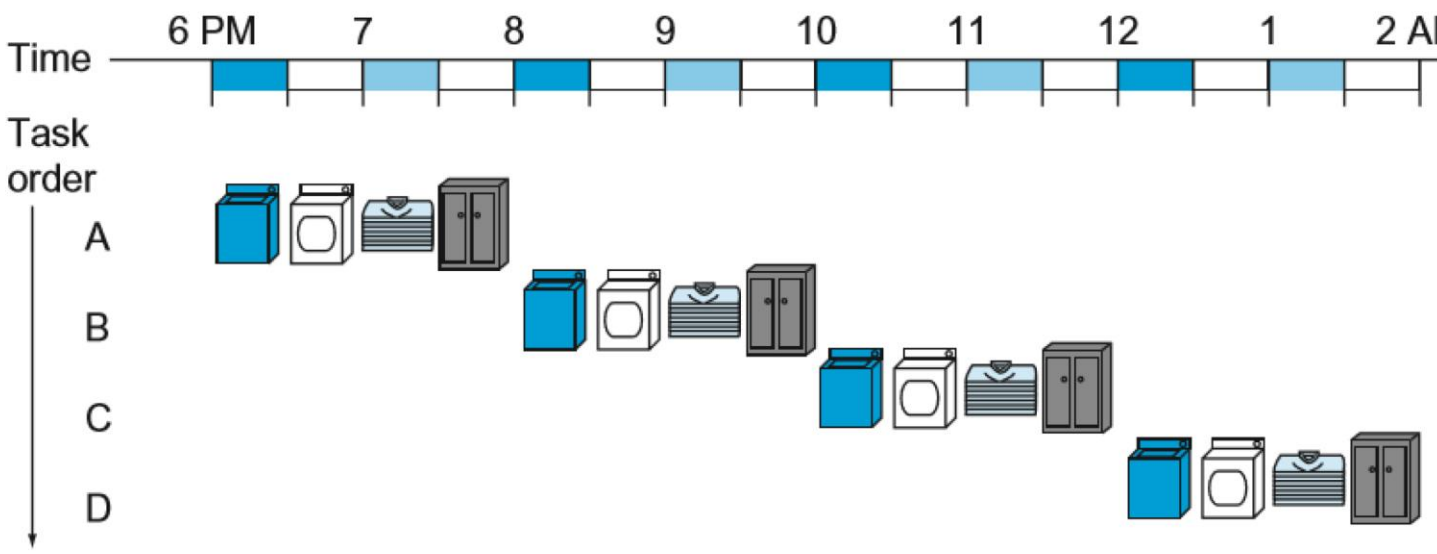
↓

CPU executes instructions (assembly/machine-level)

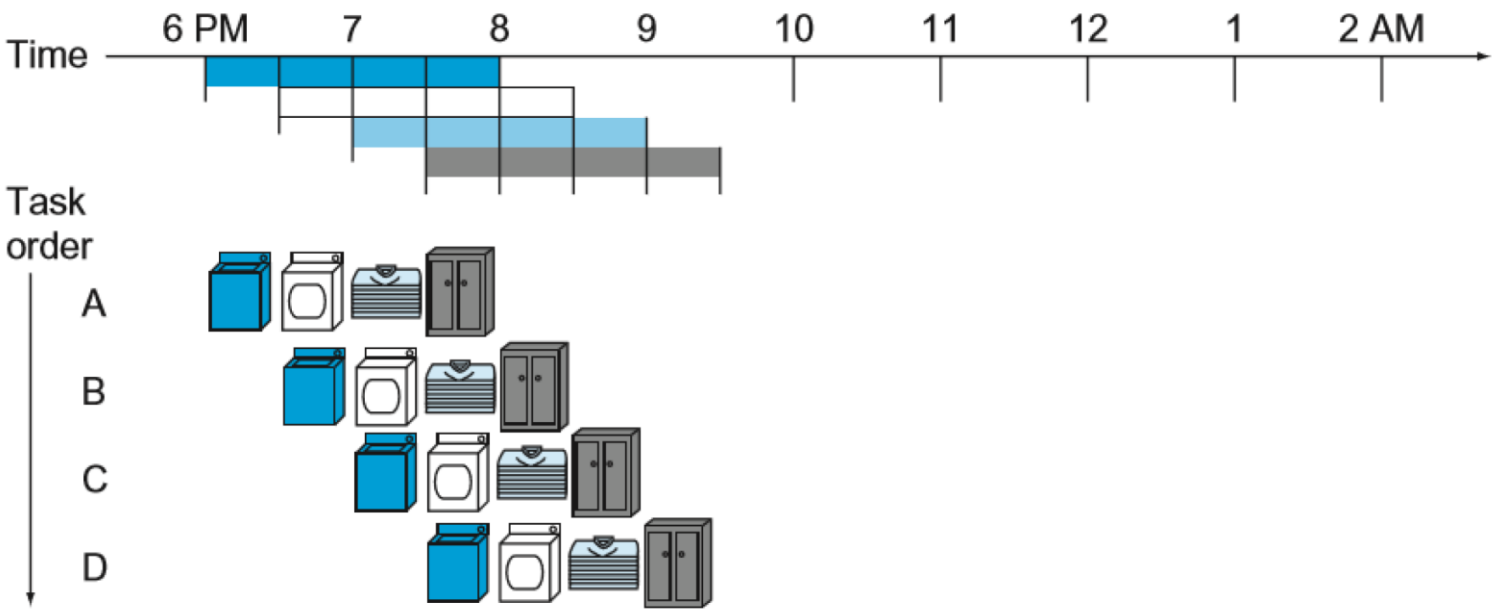
MIPS Architecture (for Pipelining)



No Pipelining



Pipelining



Talk about

- 32 bit, 64 bit architecture
  32 bit RAM limit OS (4GB)
- ARM vs x86 / x64 (Intel / AMD)
- Float v/s Double (IEEE 754)

Summary

- You learned the basics of C
  - Variables
  - Loops
  - Arrays
  - Flow control

- You learned about different architectures