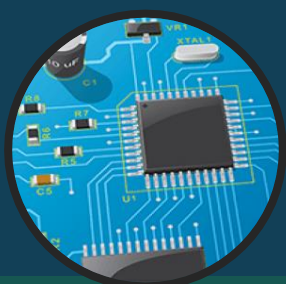
General Concept of Sequential Architecture



Lecture # 02

Fall 2020

Muhammad Imran Abeel

<imran.abeel@seecs.edu.pk>

Outlines

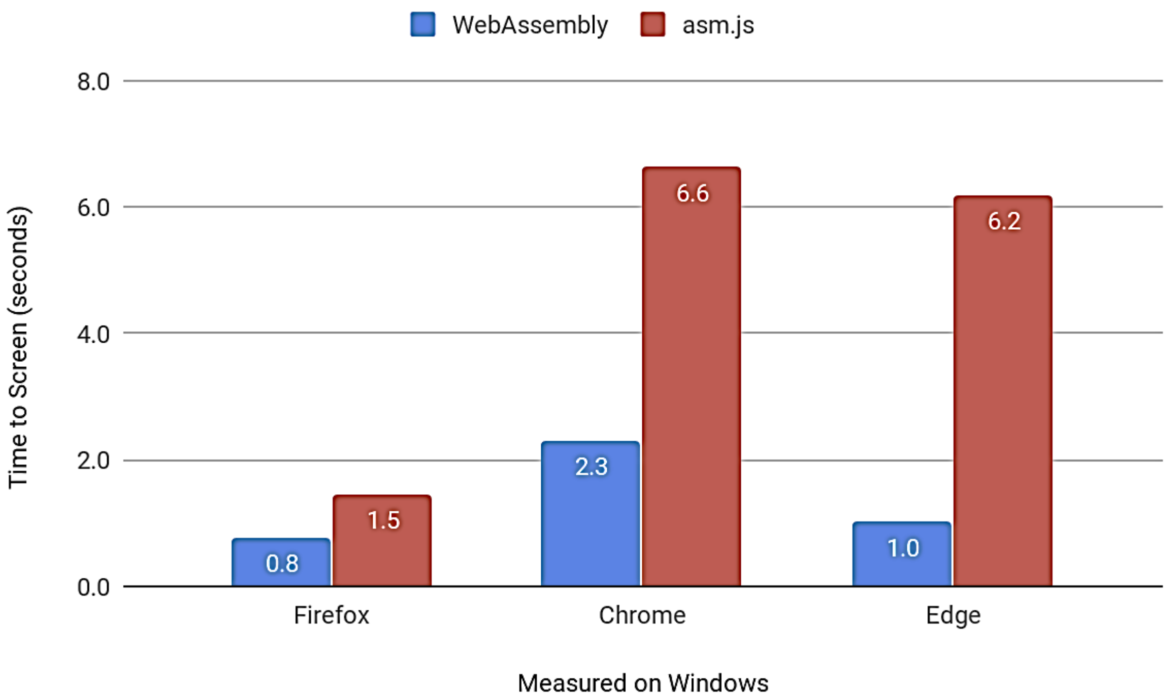
- Introduction to:
  - Assembly Language
  - Virtual Machine
  - Basic Architecture

Why Learn Assembly Language?

- **Short programs stored in a small amount of memory** for single-purpose devices

- Real-time applications with more control
  - Precise Timing & Response
  - Highly Optimized Code
  - Bit Manipulation
  - Better understanding of Computer Hardware
  - Device Drivers like Printers etc

WASM Vs Js

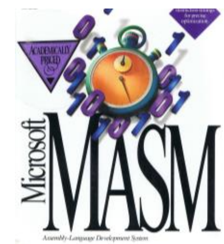


WebAssembly
asm.js
8.0
6.0
4.0
2.0
0.0
Time to Screen (seconds)
0.8
1.5
2.3
6.6
1.0
6.2
Firefox
Chrome
Edge
Measured on Windows

[4]

X86 Execution Environment

- Assembler
  - A program that converts an Assembly Language Source Code Program to Machine Language
  - Popular Assemblers
    - MASM (Microsoft Assembler)
    - TASM (Borland Turbo Assembler)



Is Assembly Language Portable?

Assembly Language Requirements

- Software
  - Editor

    Text Editor for Coding
  - Assembler

    Convert Text into Object File
  - Linker

    Convert Object File into Executable File
  - Debugger

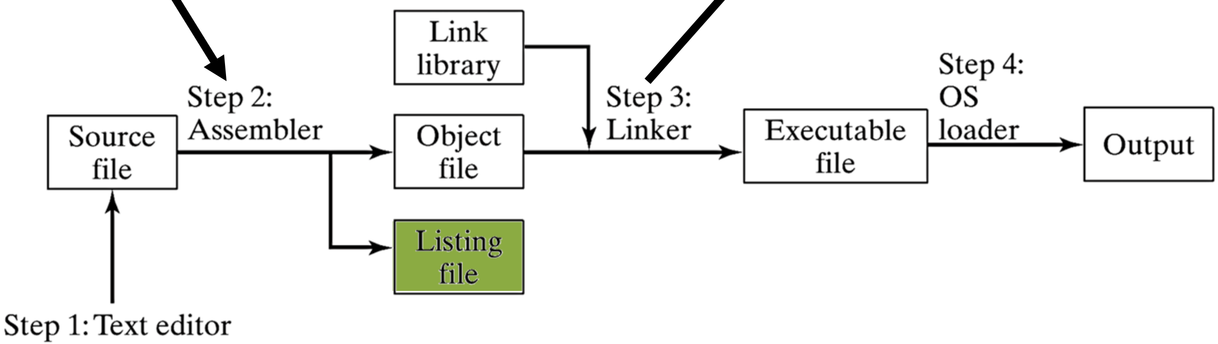
    Display Register Status/ Flag Status

    Error Detection & Correction

Assembly language

Step 2: Read source file and convert into object file (Machine language)

Step 3: The linker copies any required procedures from the link library



- Contains a copy of the Program's Source Code

[7]

Comparison

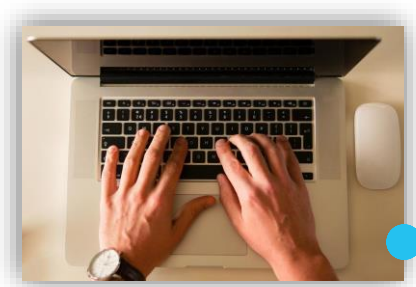
Table 1-1 Comparison of Assembly Language to High-Level Languages.

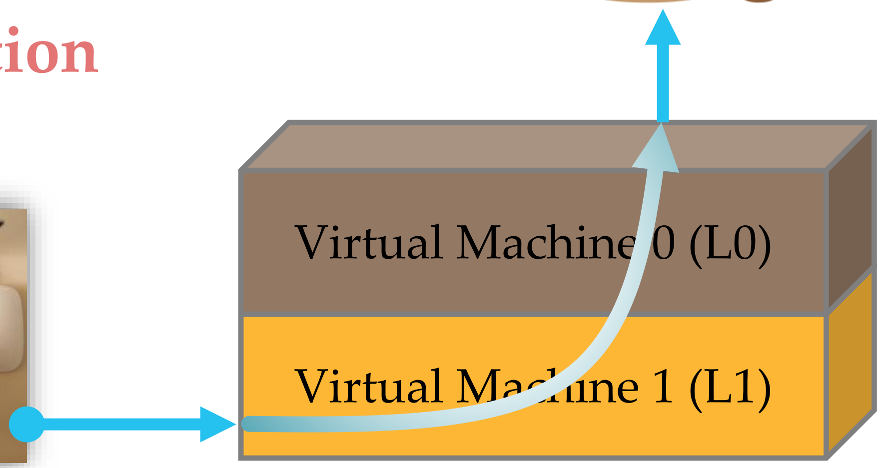
Type of Application	High-Level Languages	Assembly Language
Commercial or scientific application, written for single platform, medium to large size.	Formal structures make it easy to organize and maintain large sections of code.	Minimal formal structure, so one must be imposed by programmers who have varying levels of experience. This leads to difficulties maintaining existing code.
Hardware device driver.	The language may not provide for direct hardware access. Even if it does, awkward coding techniques may be required, resulting in maintenance difficulties.	Hardware access is straightforward and simple. Easy to maintain when programs are short and well documented.
Commercial or scientific application written for multiple platforms (different operating systems).	Usually portable. The source code can be recompiled on each target operating system with minimal changes.	Must be recoded separately for each platform, using an assembler with a different syntax. Difficult to maintain.
Embedded systems and computer games requiring direct hardware access.	May produce large executable files that exceed the memory capacity of the device.	Ideal, because the executable code is small and runs quickly.

[8]

Virtual machine

- Relation between Hardware and Software
  - Interpretation
  - Translation





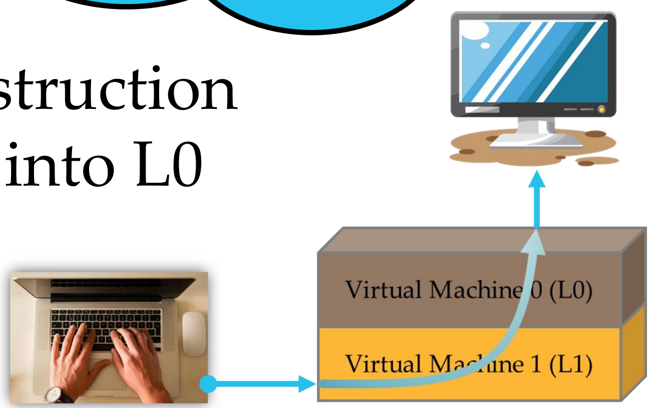
[9]

Virtual Machine

- Interpretation
- Translation

Program starts execution Immediately

Note: But each L1 instruction needs to be decoded into L0 before execution.



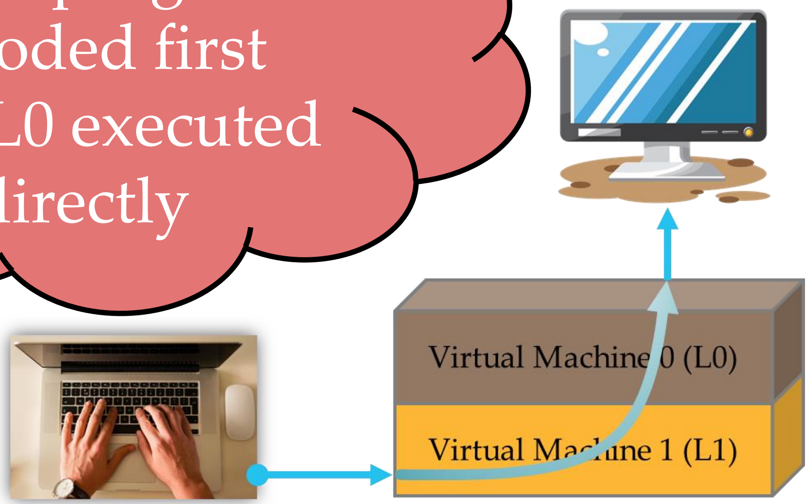
[10]

Virtual Machine

- Interpretation

- Translation

Entire program decoded first than L0 executed directly



[11]

Assembly Language

- Levels of programming language

High Level Language

```
int Y;
int X = (Y + 4) * 3;
```

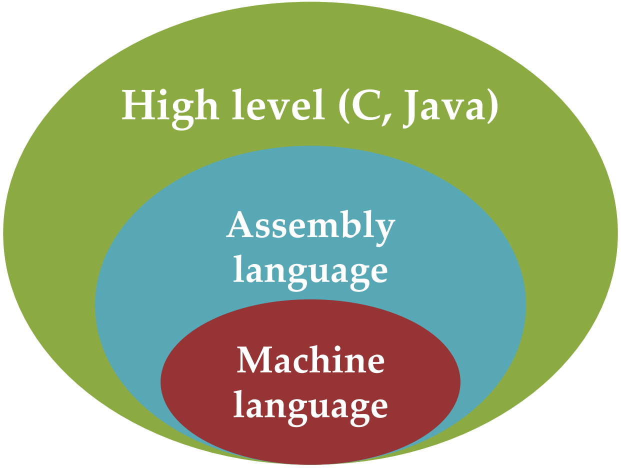
Low Level Language

```
mov eax,Y ;
add eax,4 ;
mov ebx,3 ;
imul ebx ;
mov X,eax ;
```

Executable Machine Code

```
000100101001001001O1
```

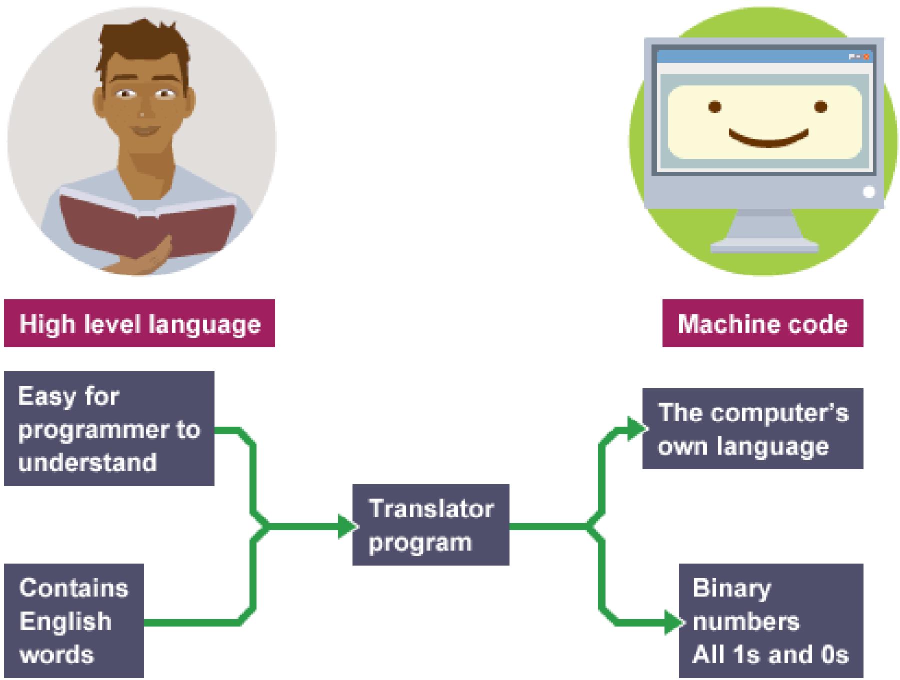
- One-to-many relationship with Assembly Language



[12]

Assembly Language

- Why we need different levels of programming languages?



[13]

Reading Assignment

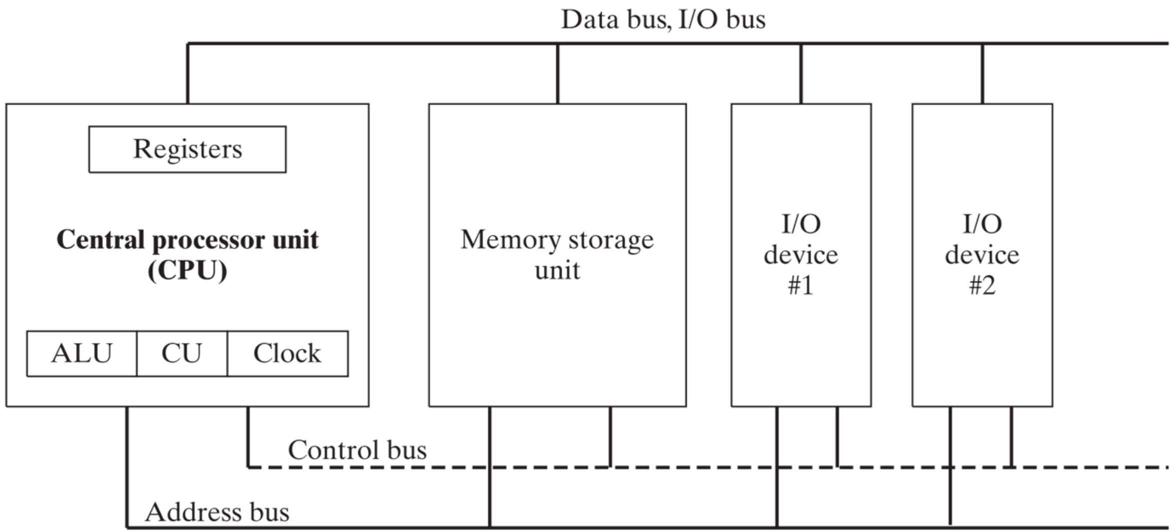
Number System: Binary, Hexadecimal etc

Signed, Unsigned Numbers

1's Compliment, 2's Compliment

Microcomputer Design

FIGURE 2–1 Block diagram of a microcomputer.



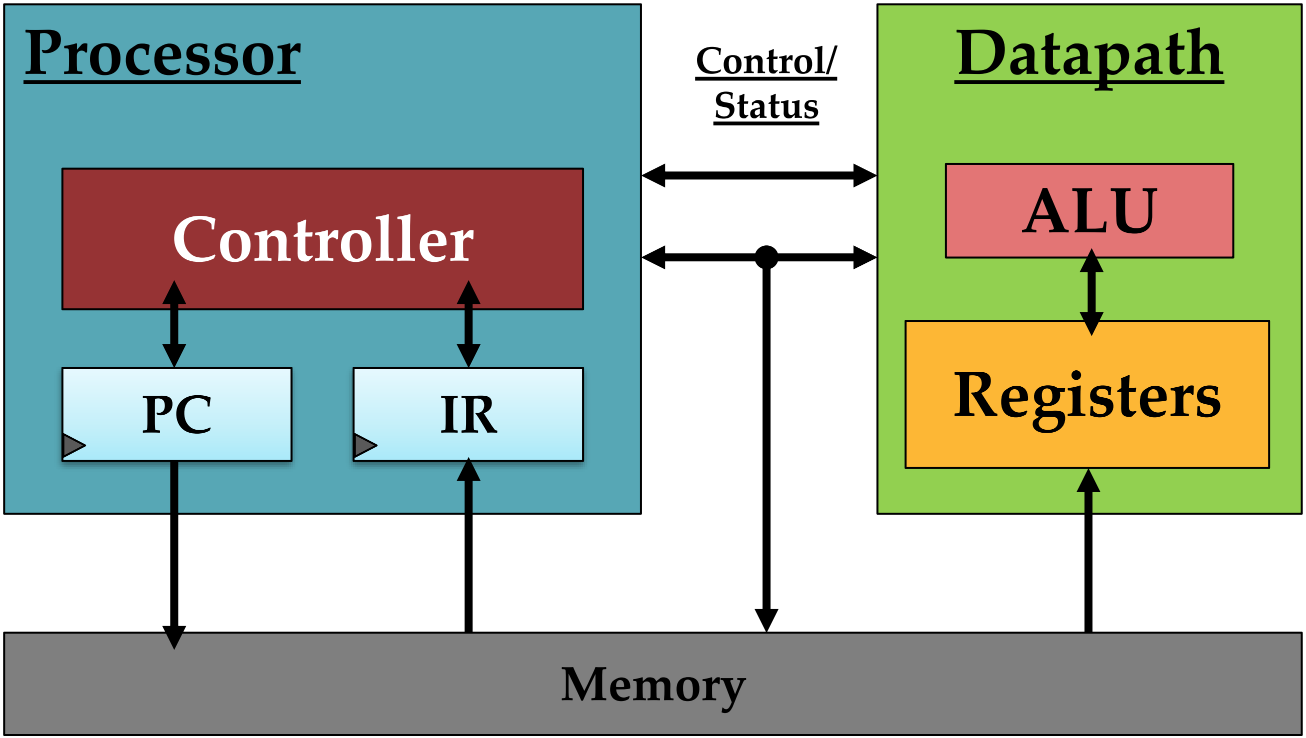
High Frequency Clock

Registers

Central Processing Unit

Arithmetic Logic Unit

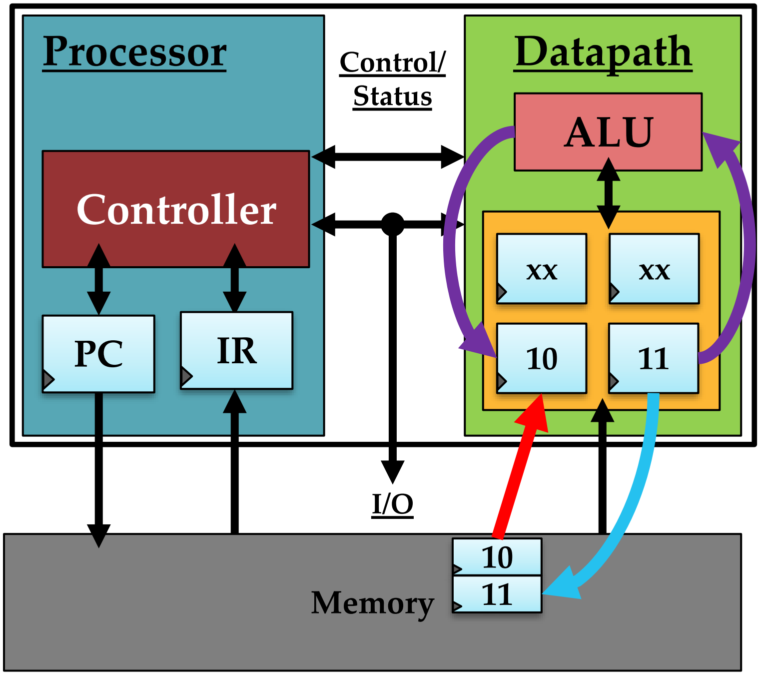
Basic Architecture



PC: Program Counter
IR: Instruction Register

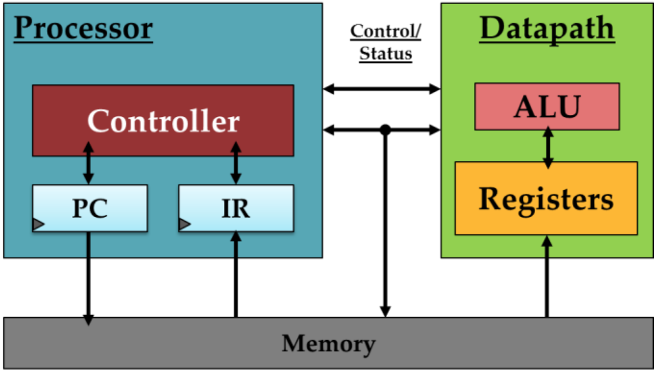
Datapath operations

- Load:
  Memory to Register
- Store:
  Register to Memory
- ALU Op:
  Register to Register



Control Unit

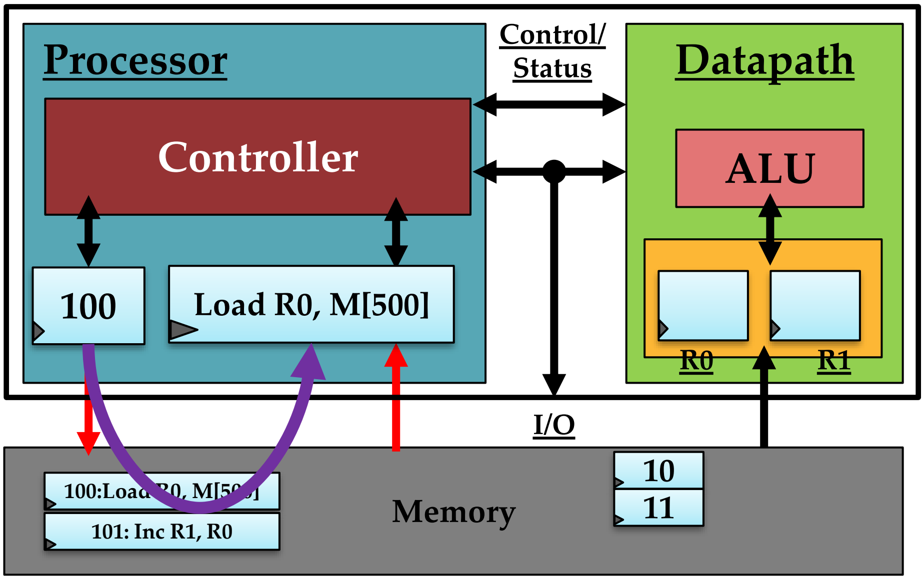
- Generates Control Signals for Datapath Operations
- **Instruction Cycle: Divided into Sub Operations**
  - ***Fetch***
    - get next instruction into IR
  - ***Decode***
    - Determine what instruction means
  - ***Fetch Operand***
    - Load data from memory to register
  - ***Execute***
    - ALU operation
  - ***Store Results***
    - Store operation



[18]

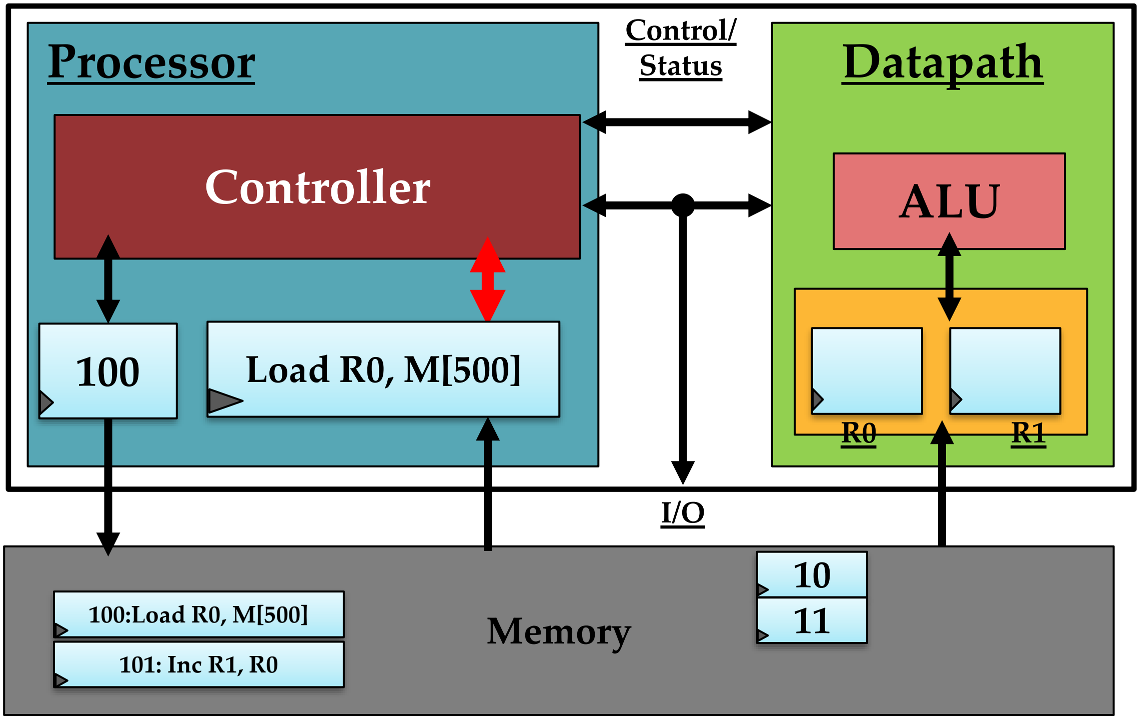
Instruction Cycle: Fetch

- *PC holds position for the next instruction*
- *Get instruction into IR Register*
- *Increment PC*



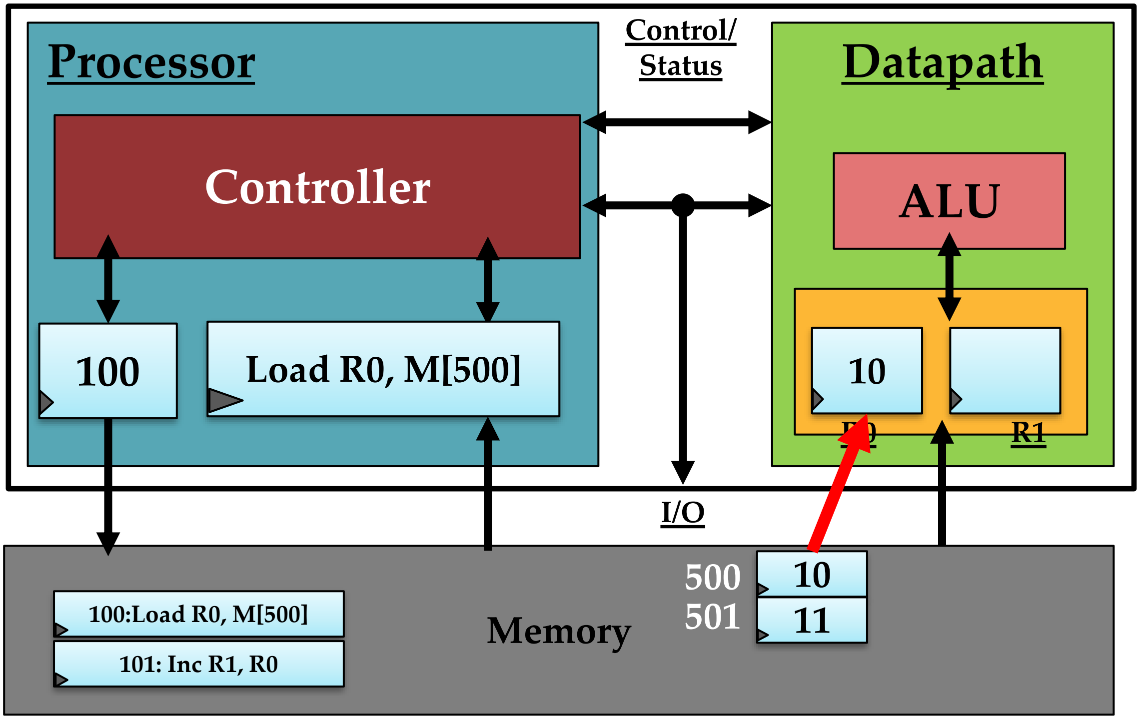
Instruction Cycle: Decode

- *Determine what the Instruction means?*



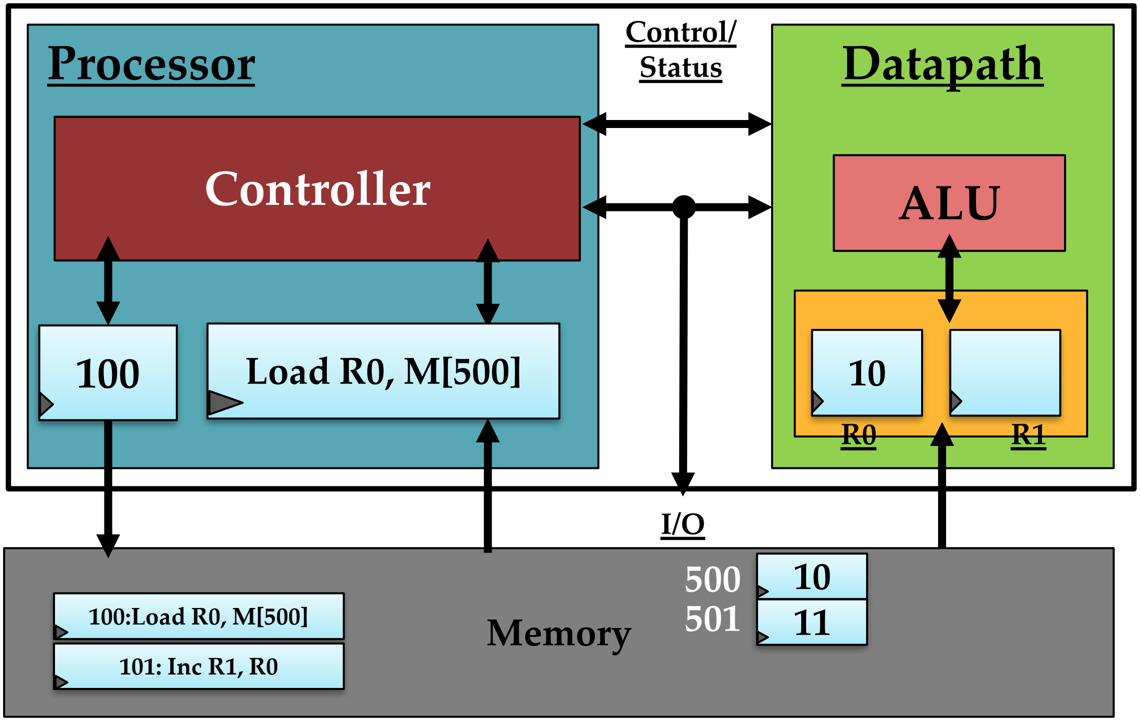
Instruction Cycle: Fetch Data

- *Load data from memory to register (LOAD op.)*



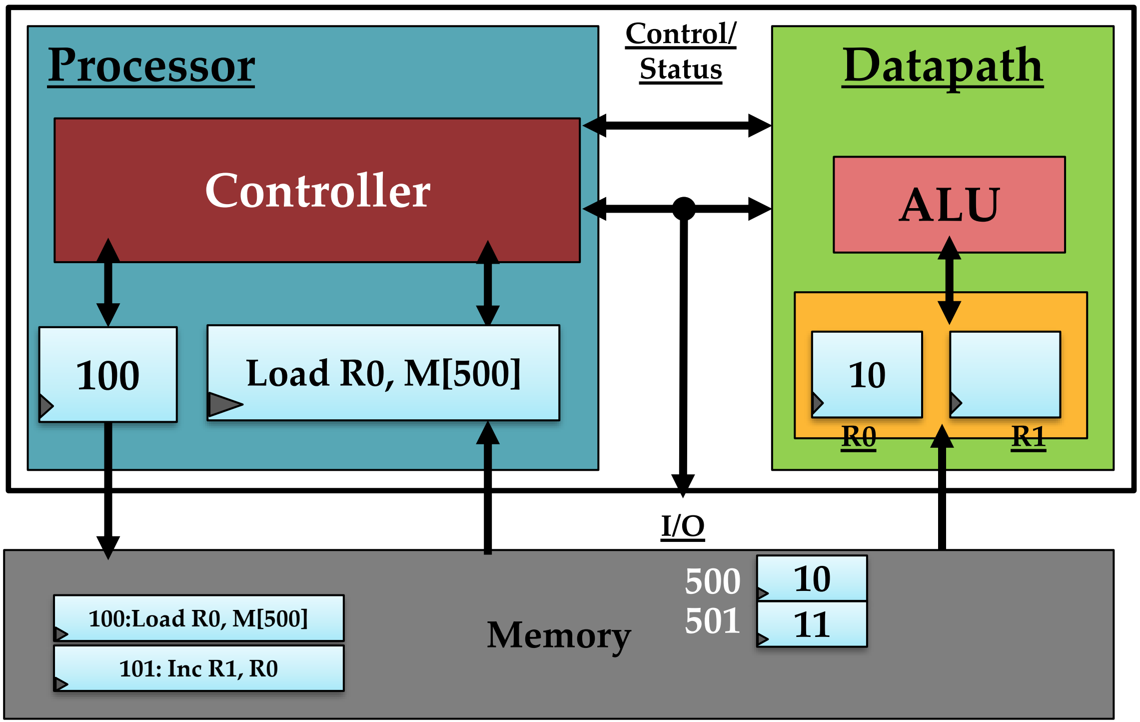
Instruction Cycle: Execute

- *Execute arithmetic operation through ALU (No Op.)*



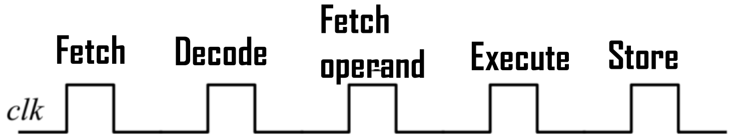
Instruction Cycle: Store

- *Move Data from Register to Memory* *(No Op.)*

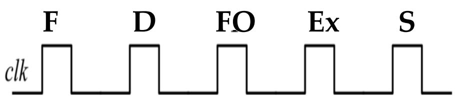


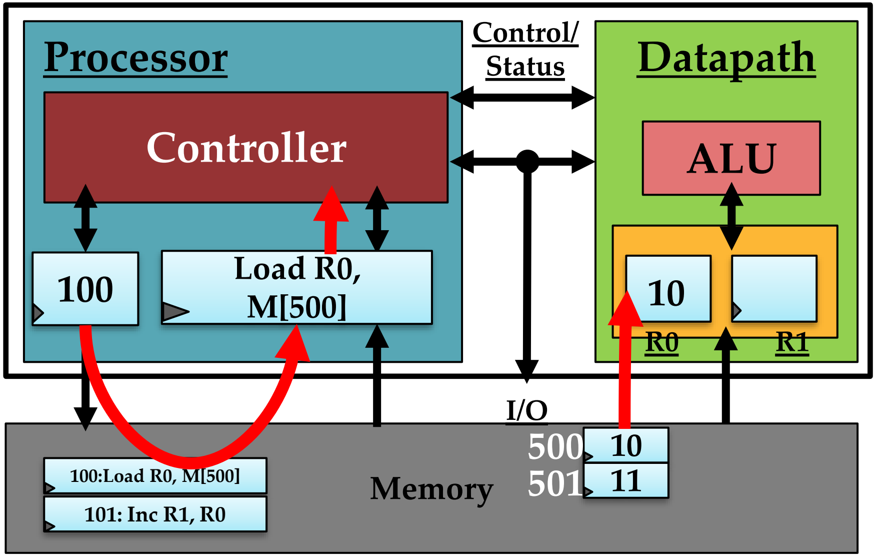
Instruction cycle

- *Move Data from Register to Memory* *(No Op.)*



PC= 100

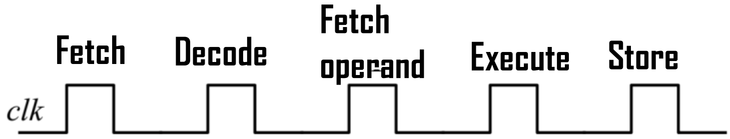




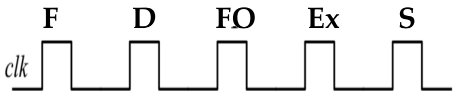
[24]

Instruction cycle

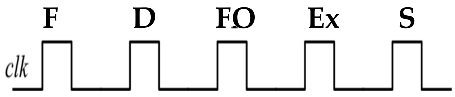
- *Move Data from Register to Memory* *(No Op.)*

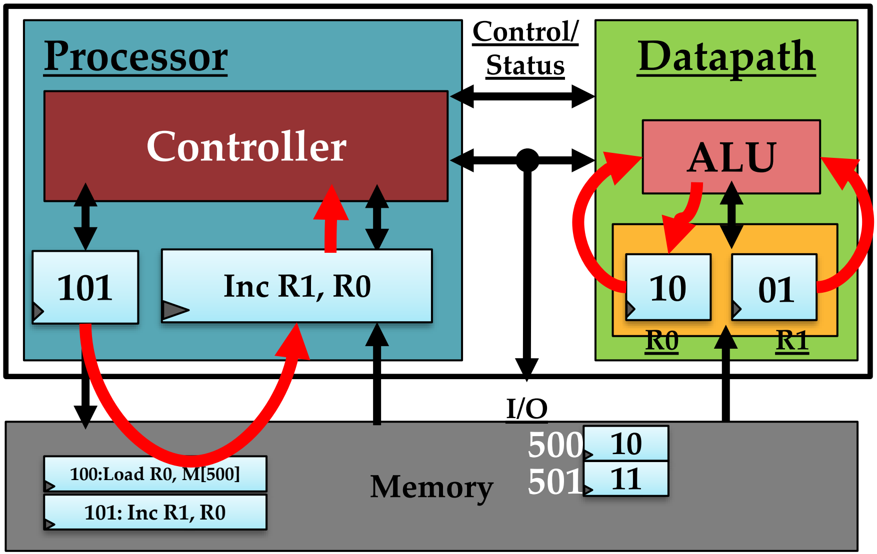


PC= 100



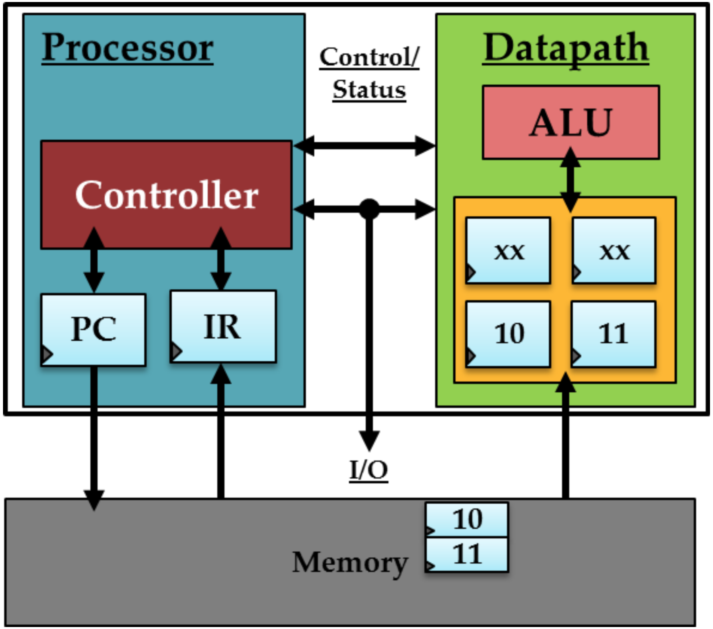
PC= 101





N-bit Processor

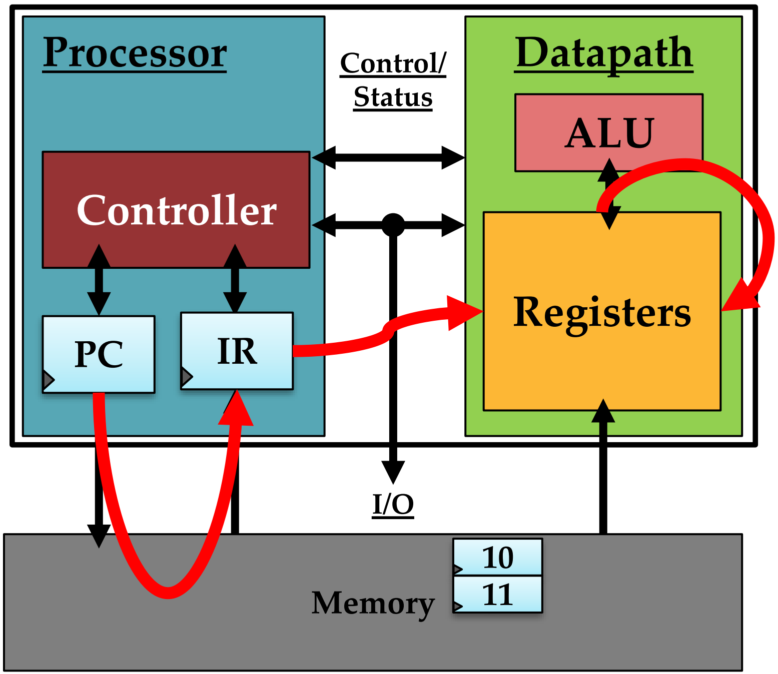
- *N-bit ALU, Registers, Busses, Memory Data Interface*
- *Embedded: 8-bit, 16 bit and 32 bit common*
- *Desktop/Severs: 32-bit & 64 bit*



Program counter (PC) size determine Address Space

Clock Frequency

- <u>*Clock Frequency*</u>
  - *Inverse of clock period ($f = 1/T$)*
  - *f > reg. to reg. delay*
  - *Memory access is often longest*



Questions?

THANK YOU!