

Chapter 3

General Purpose Processors

Introduction

- **General-Purpose Processor**

What would happen if you made a single purpose processor that had the function:

```
While (1) {
```

```
  Read data at memory location,
```

```
  if data is “load location to reg1” then load the given memory location into that register.
```

```
  else if memory location is “add reg1 to reg2” then add the two registers.
```

```
  .
```

```
  .
```

```
  end if;
```

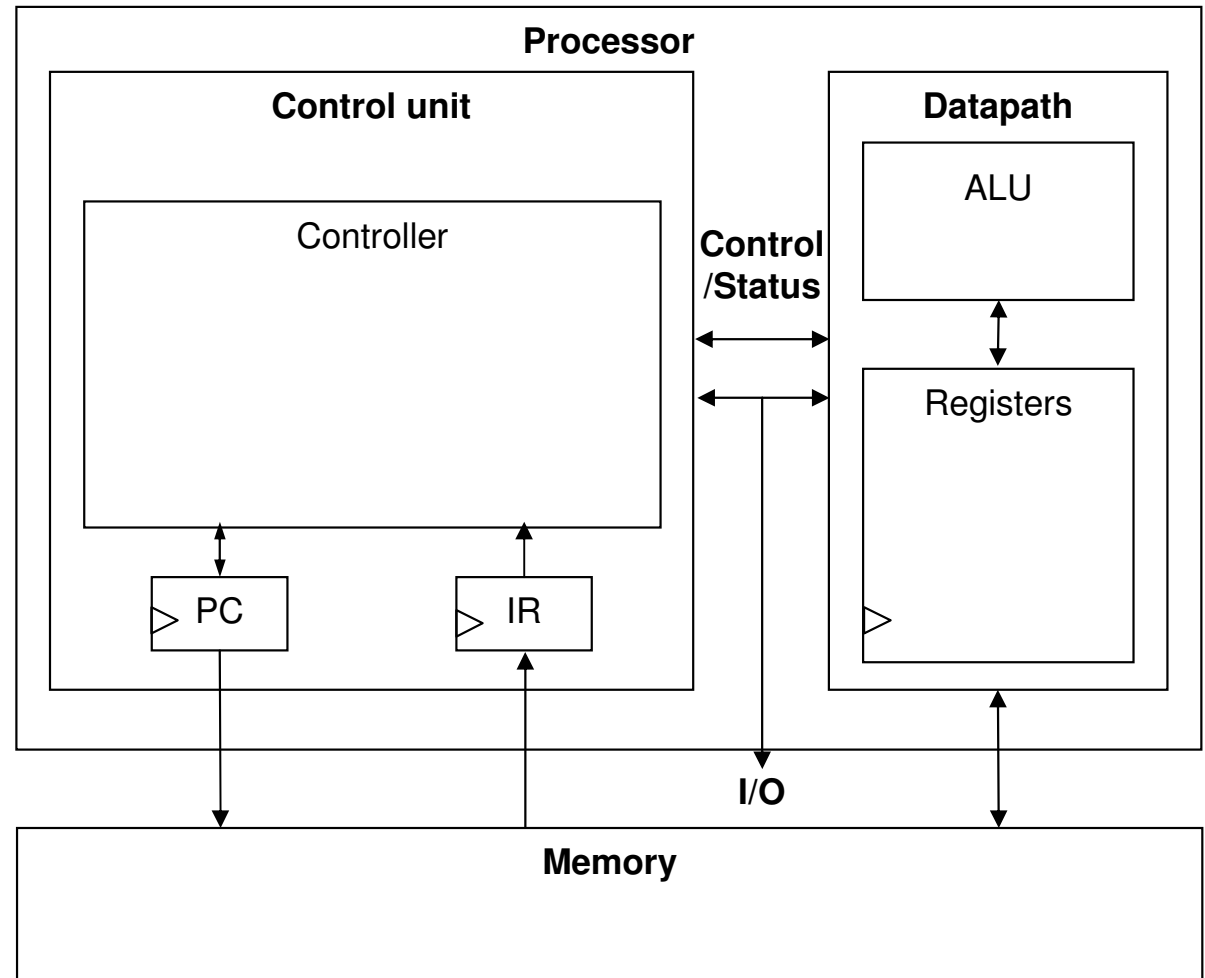
```
}
```

Introduction

- General-Purpose Processor
 - Processor designed for a variety of computation tasks
 - Low unit cost, in part because manufacturer spreads NRE over large numbers of units
 - Motorola sold half a billion 68HC05 microcontrollers *in 1996 alone*
 - Carefully designed since higher NRE is acceptable
 - Can yield good performance, size and power
 - Low NRE cost, short time-to-market/prototype, high flexibility
 - User just writes software; no processor design
 - a.k.a. “microprocessor” – “micro” used when they were implemented on one or a few chips rather than entire rooms

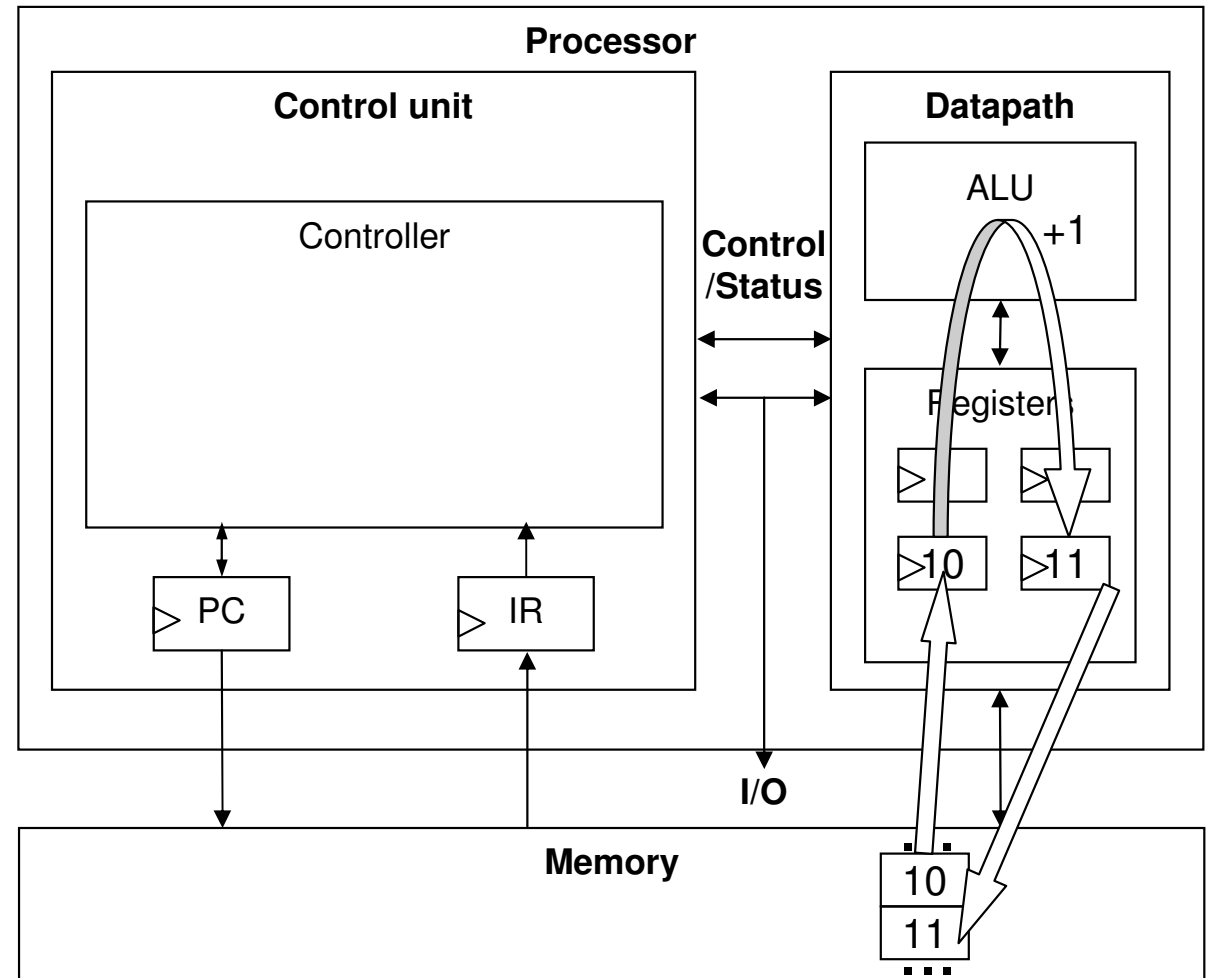
Basic Architecture

- Control unit and datapath
 - Note similarity to single-purpose processor
- Key differences
 - Datapath is general
 - Control unit doesn't store the algorithm – the algorithm is “programmed” into the memory



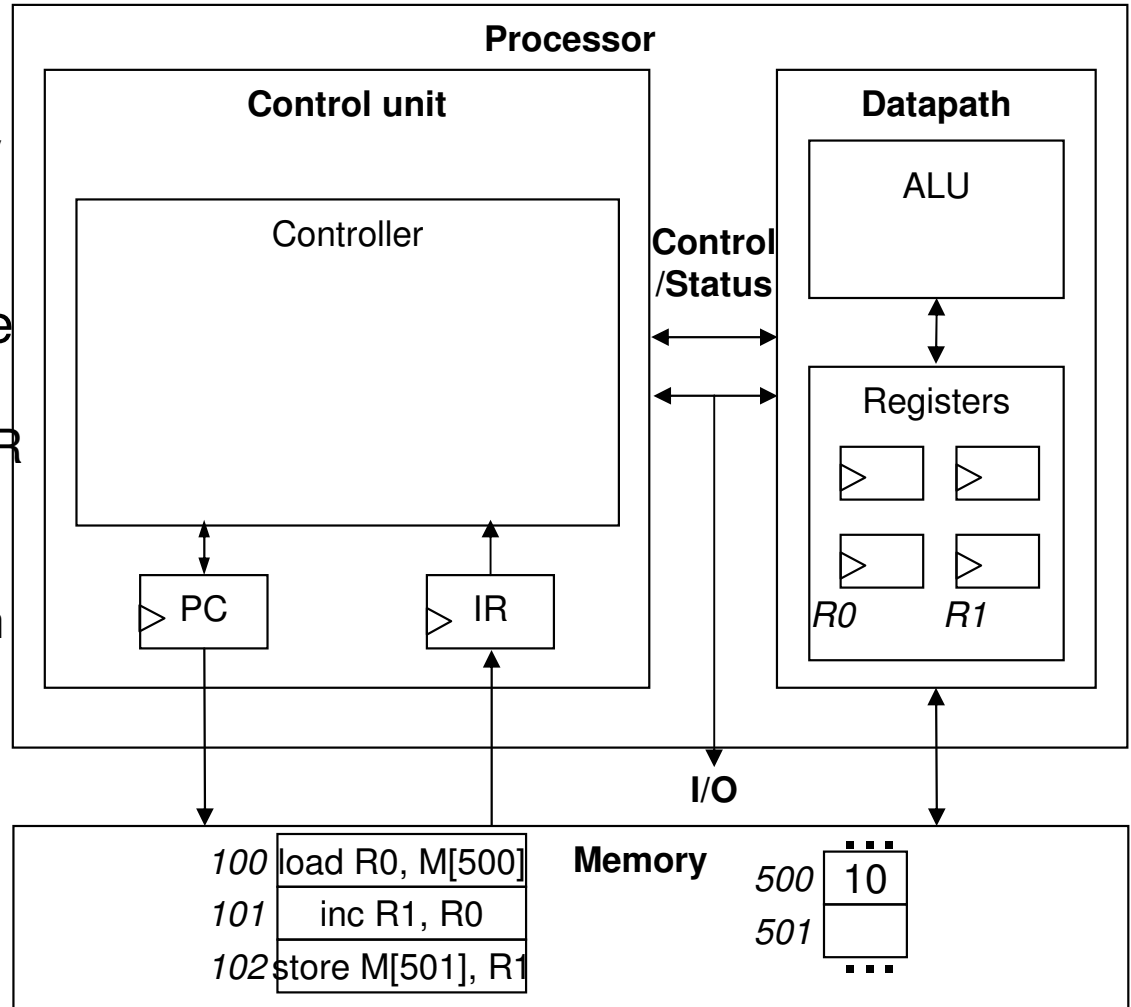
Datapath Operations

- Load
 - Read memory location into register
- ALU operation
 - Input certain registers through ALU, store back in register
- Store
 - Write register to memory location



Control Unit

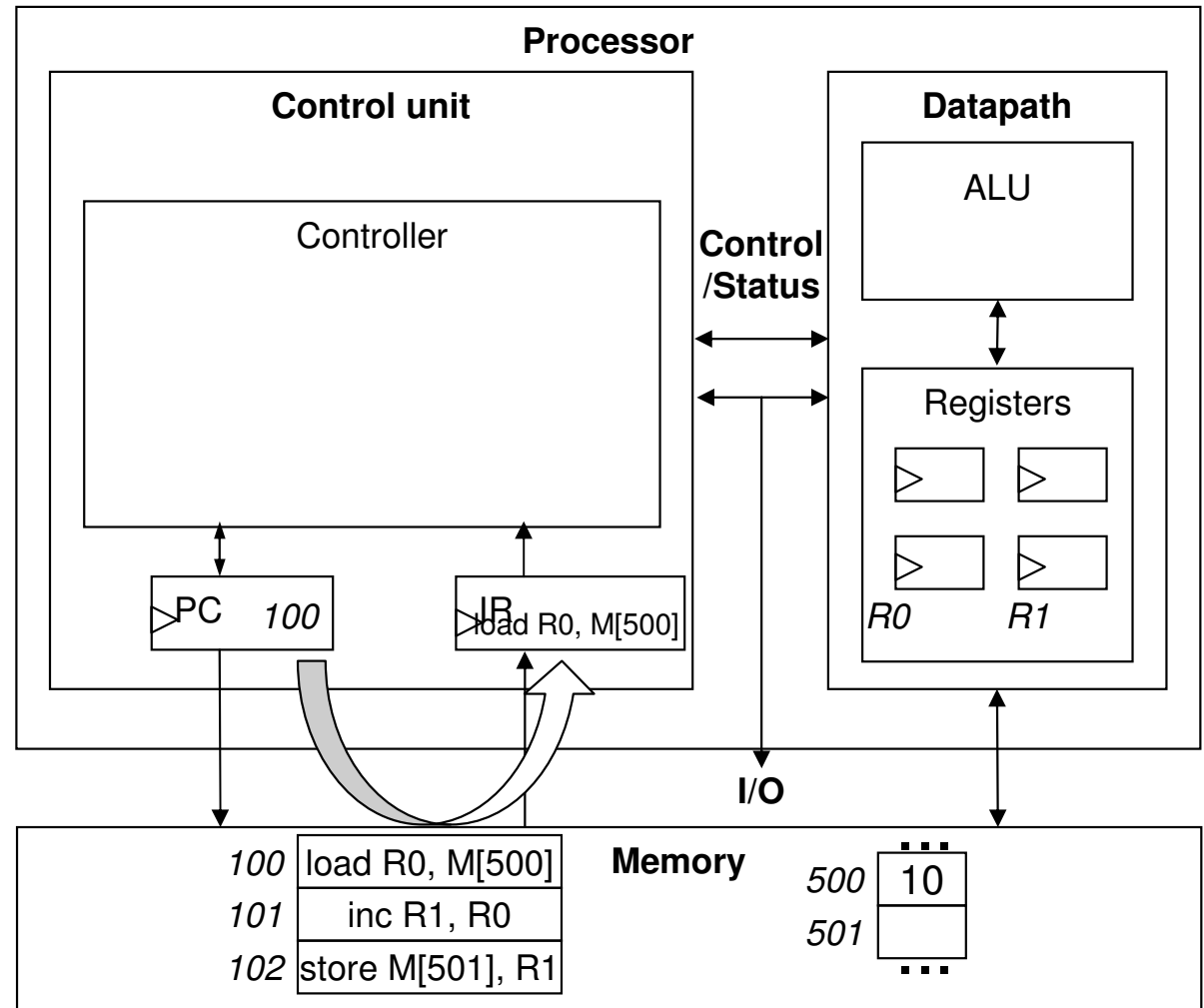
- Control unit: configures the datapath operations
 - Sequence of desired operations (“instructions”) stored in memory
 - “program”
- Instruction cycle – broken into several sub-operations, each one clock cycle, e.g.:
 - Fetch: Get next instruction into IR
 - Decode: Determine what the instruction means
 - Fetch operands: Move data from memory to datapath register
 - Execute: Move data through the ALU
 - Store results: Write data from register to memory



Control Unit Sub-Operations

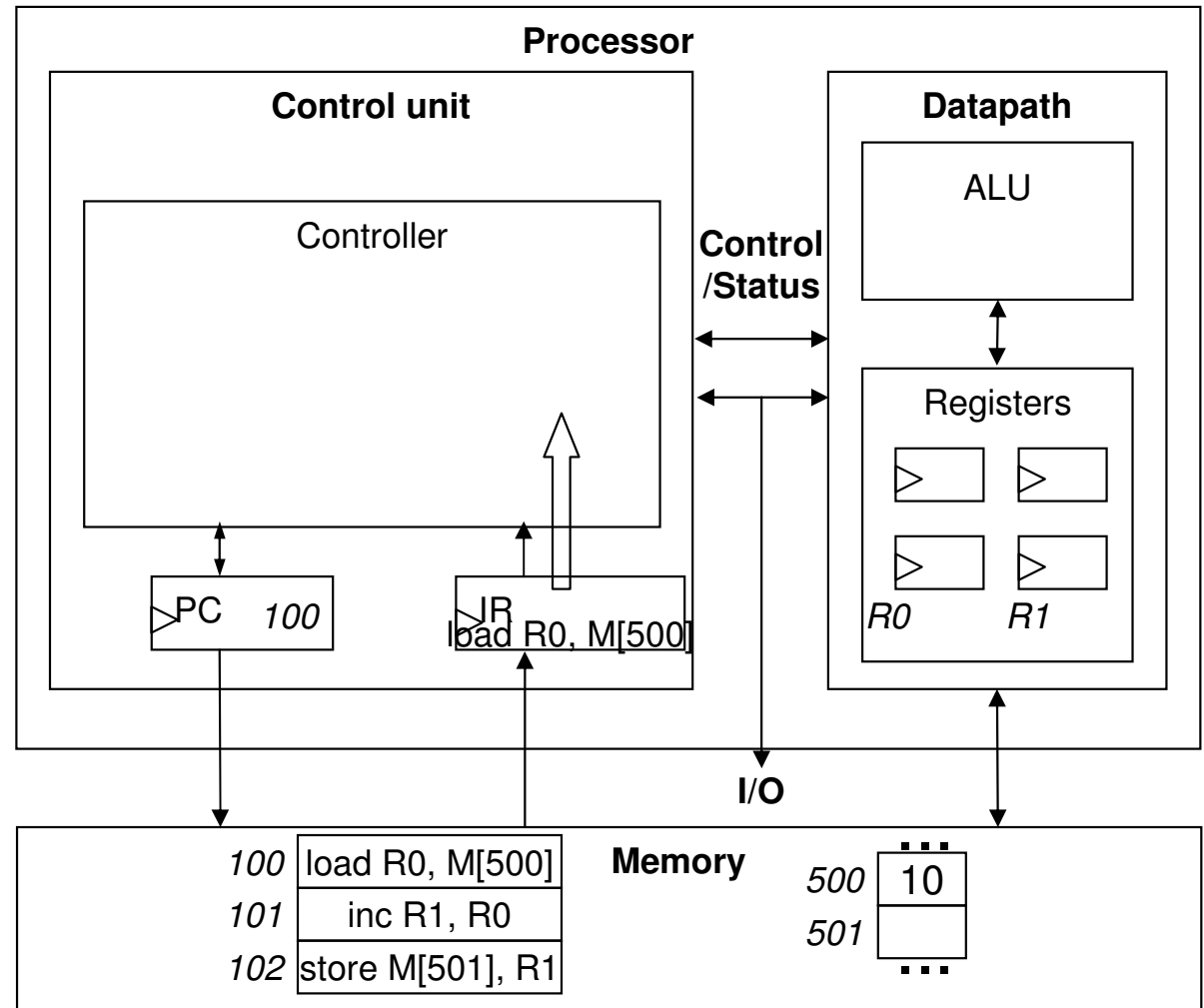
- Fetch

- Get next instruction into IR
- PC: program counter, always points to next instruction
- IR: holds the fetched instruction



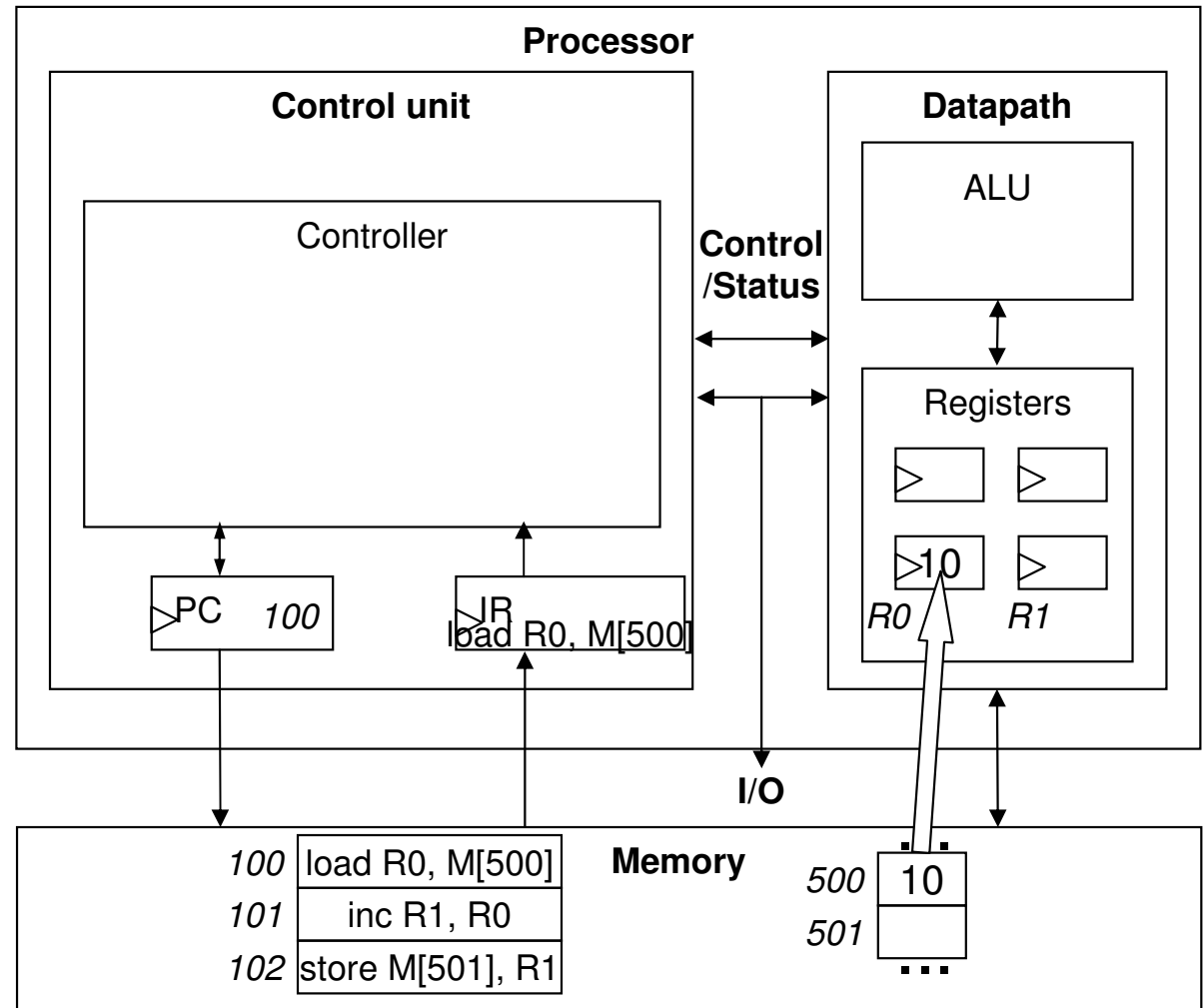
Control Unit Sub-Operations

- Decode
 - Determine what the instruction means



Control Unit Sub-Operations

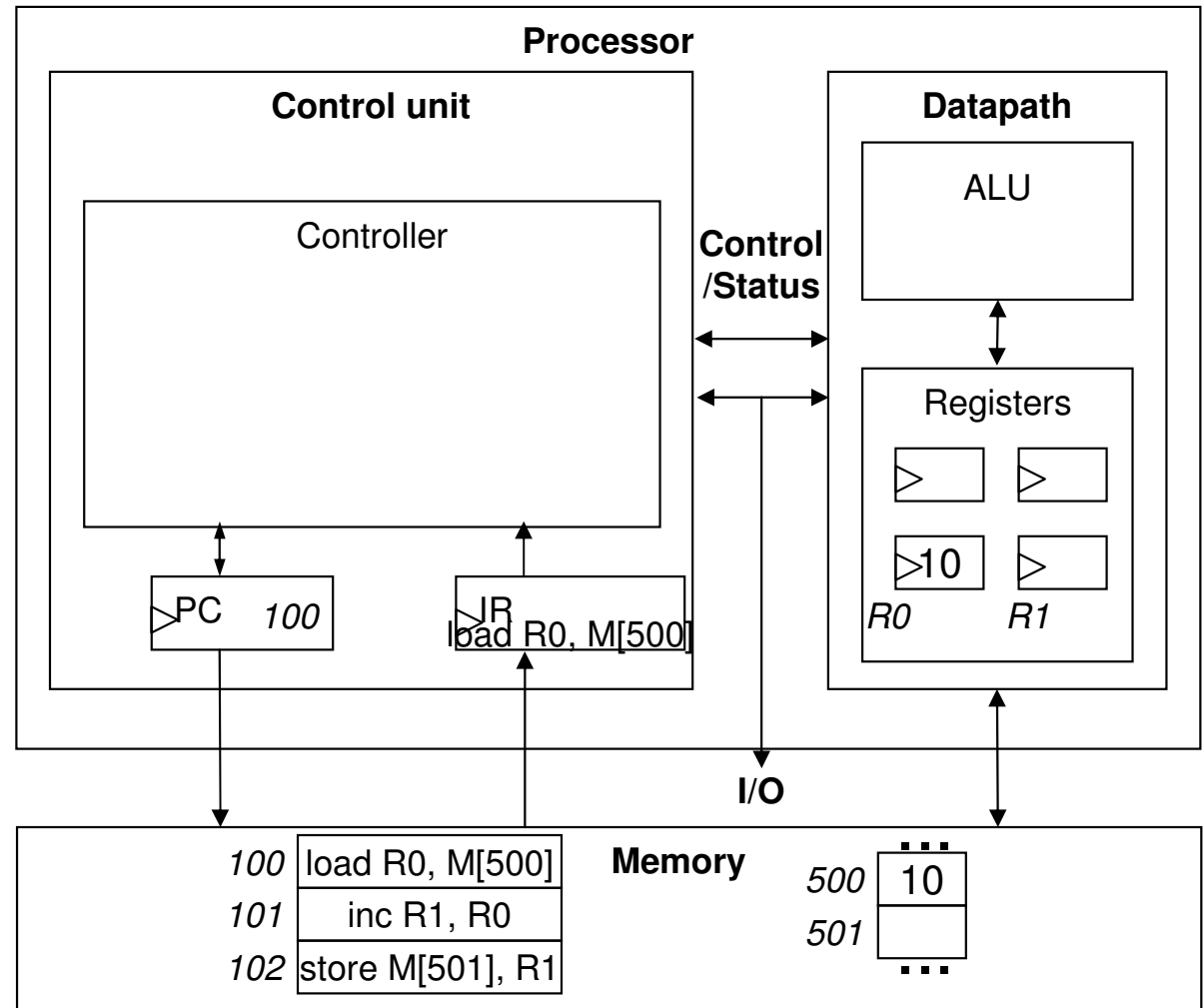
- Fetch operands
 - Move data from memory to datapath register



Control Unit Sub-Operations

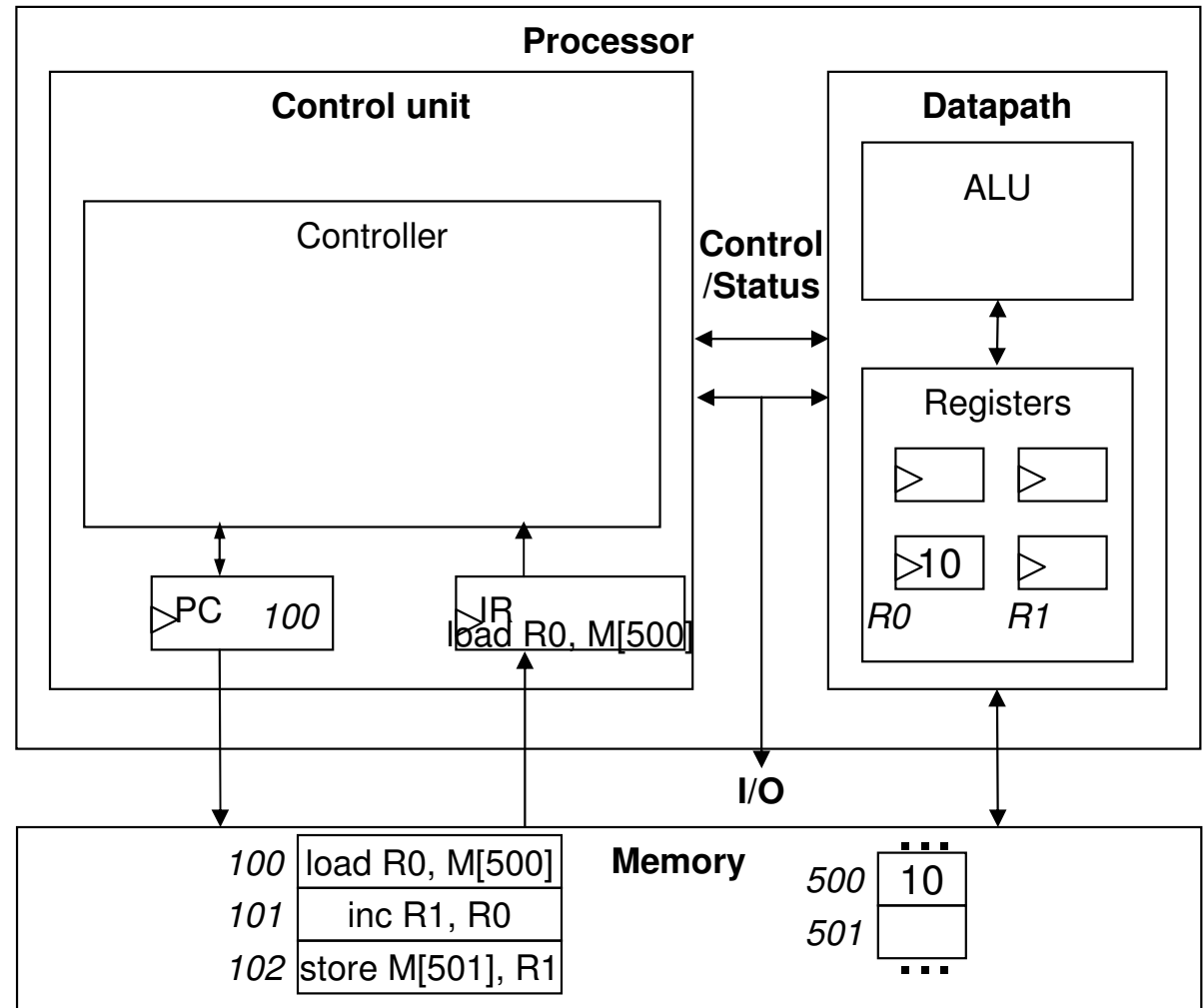
- Execute

- Move data through the ALU
- For the load operation, nothing happens in the ALU, so this clock cycle doesn't accomplish anything



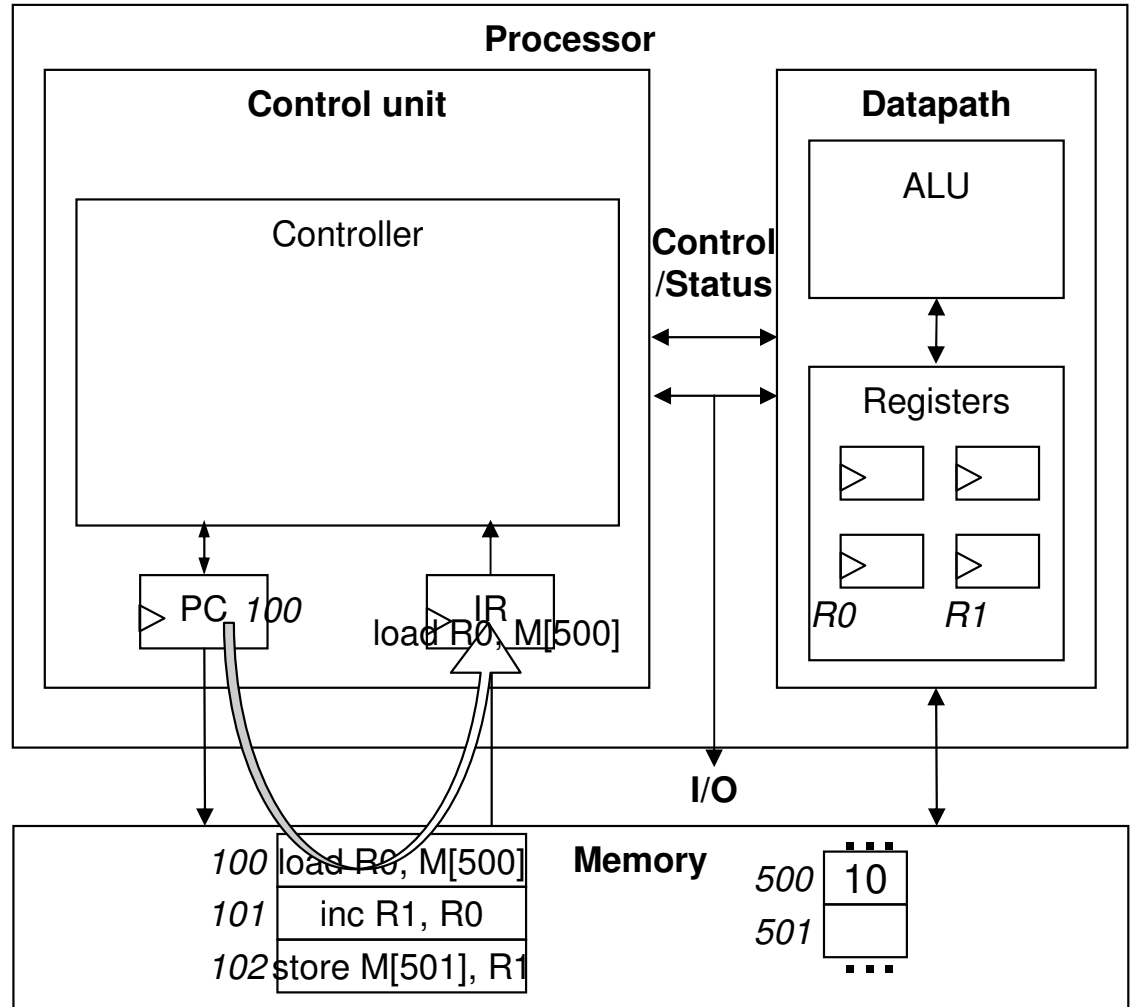
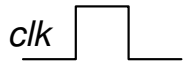
Control Unit Sub-Operations

- Store results
 - Write data from register to memory
 - This particular instruction does nothing during this sub-operation



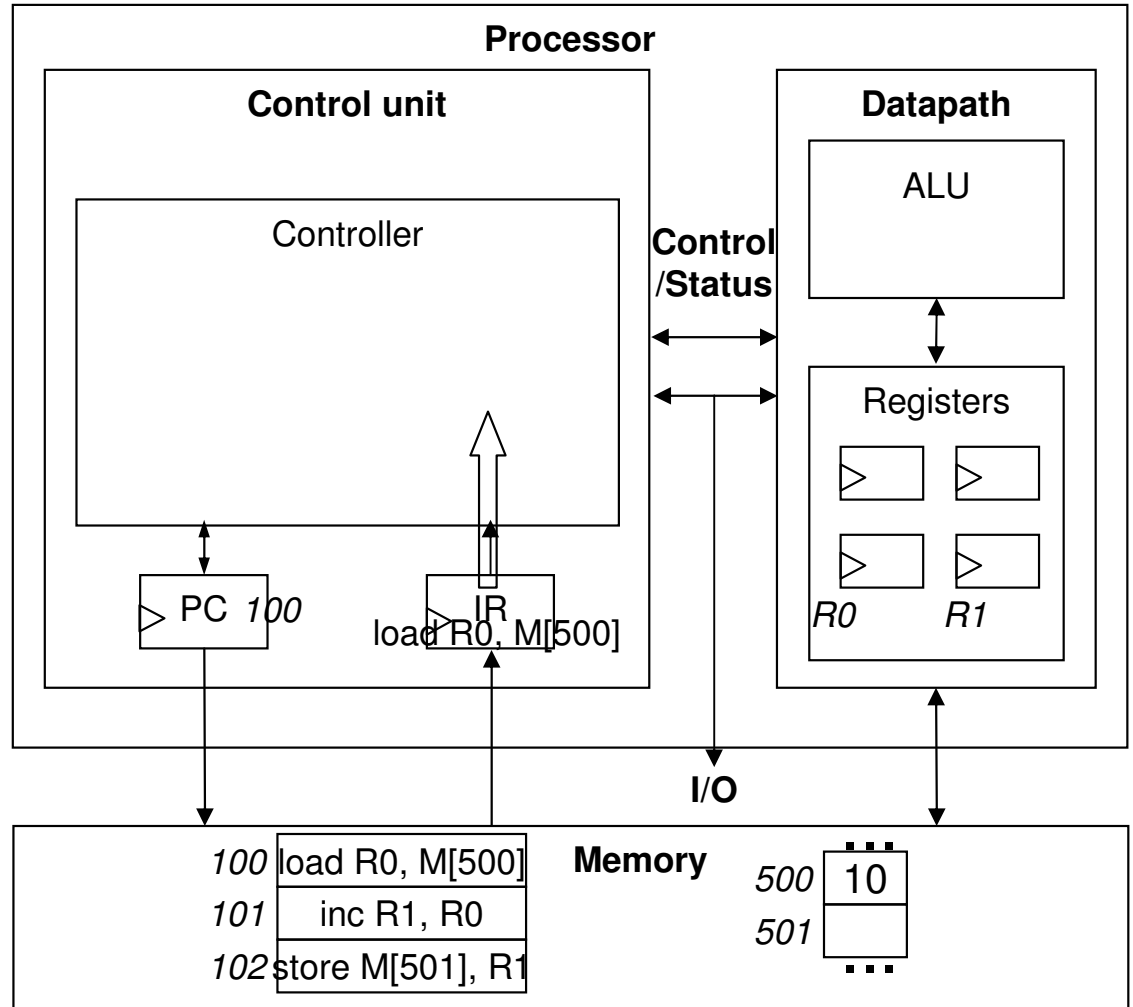
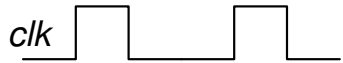
Instruction Cycles

PC=100
Fetch

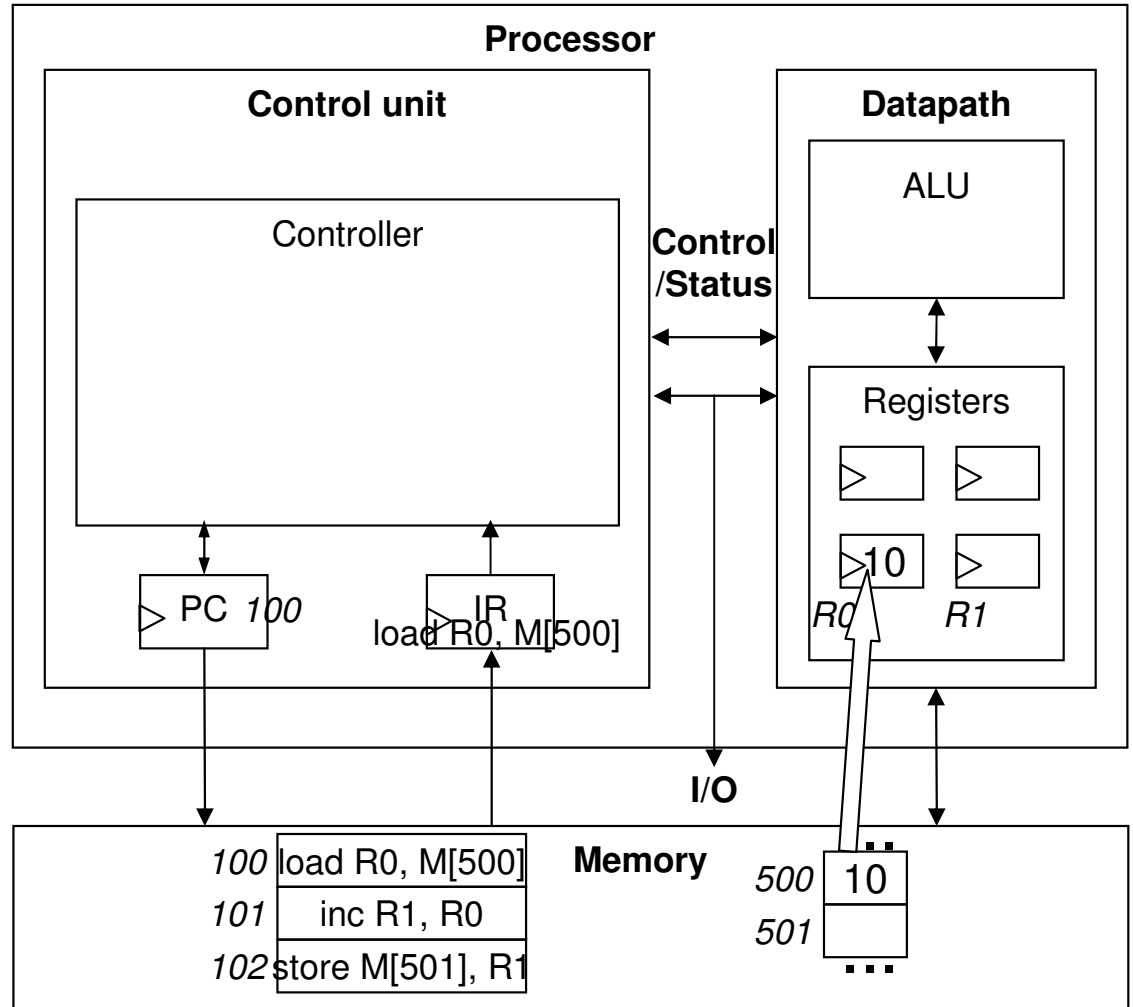
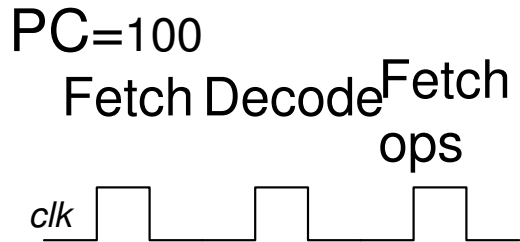


Instruction Cycles

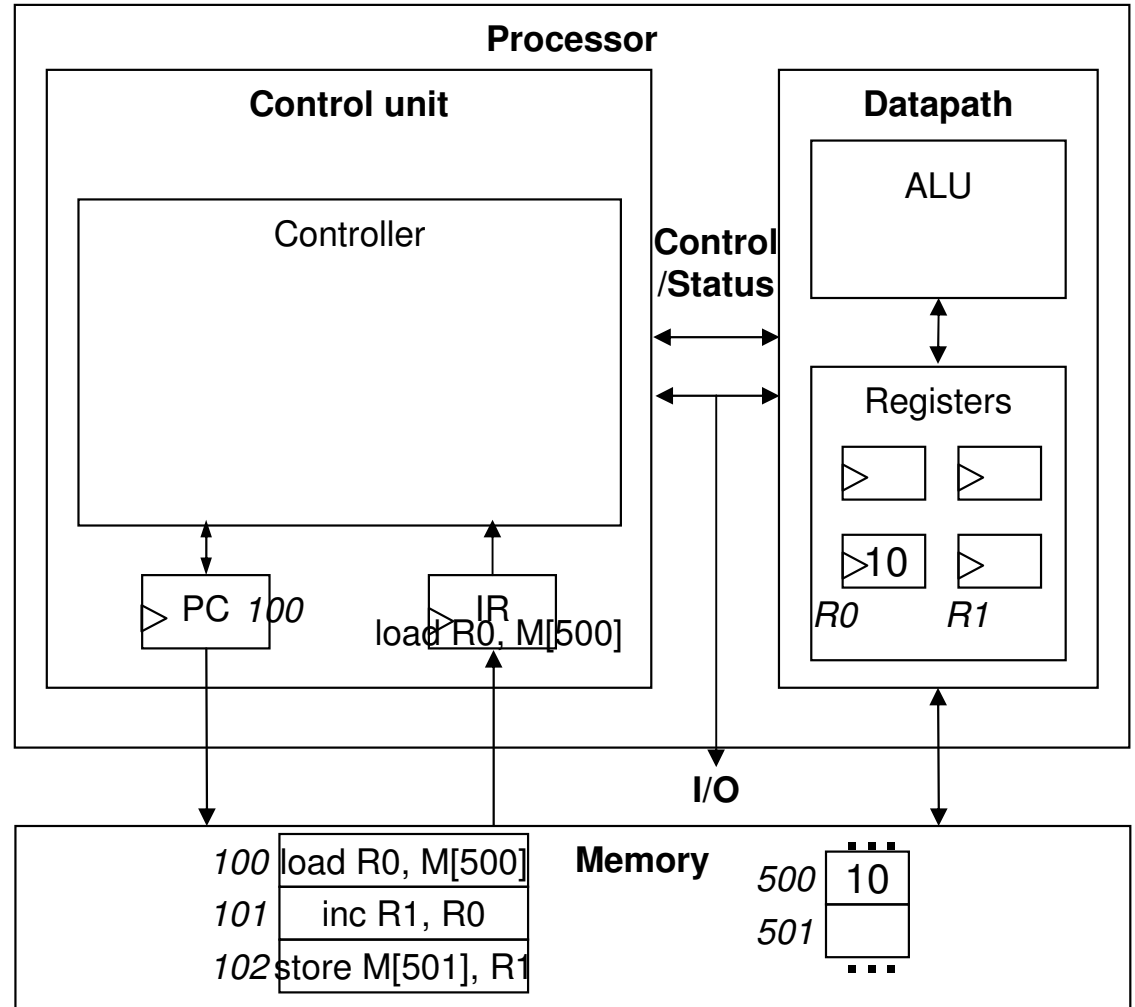
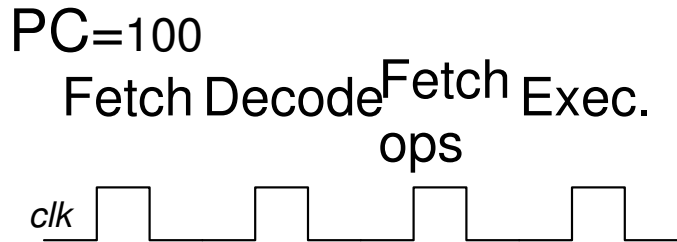
PC=100
Fetch Decode



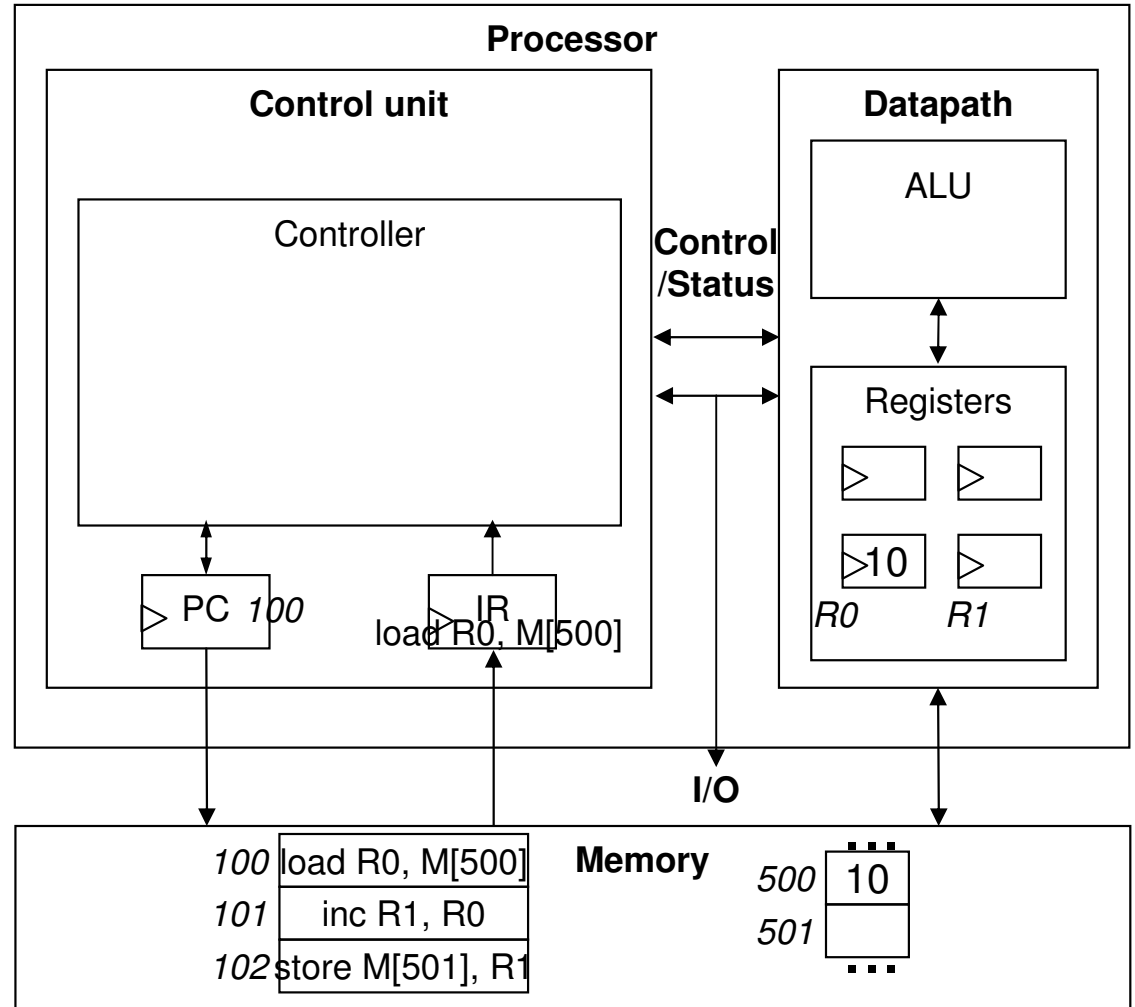
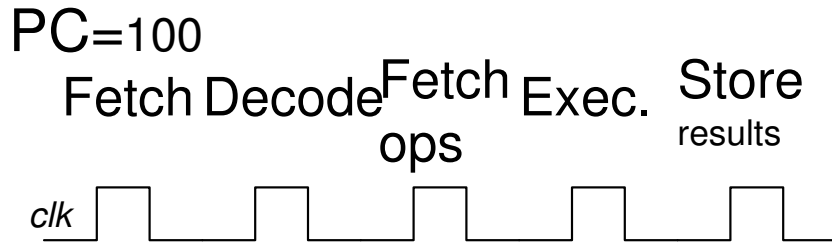
Instruction Cycles



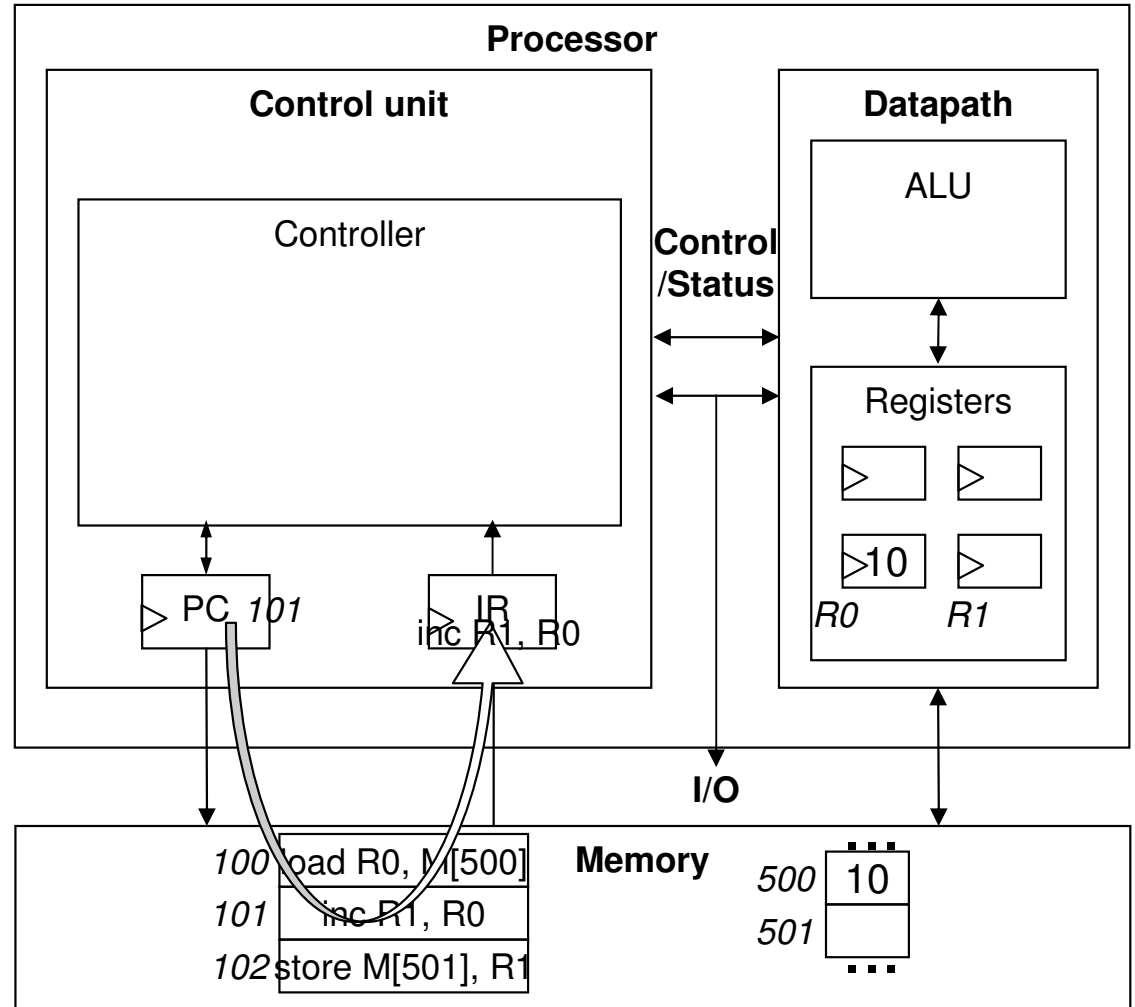
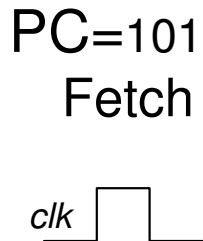
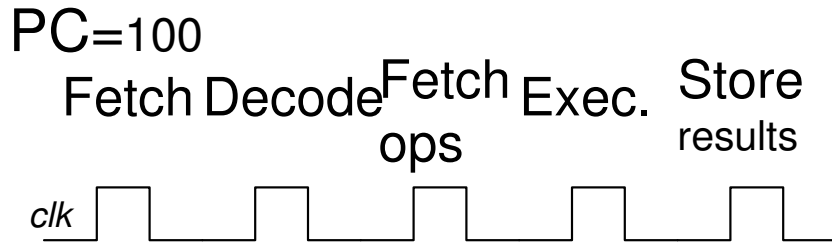
Instruction Cycles



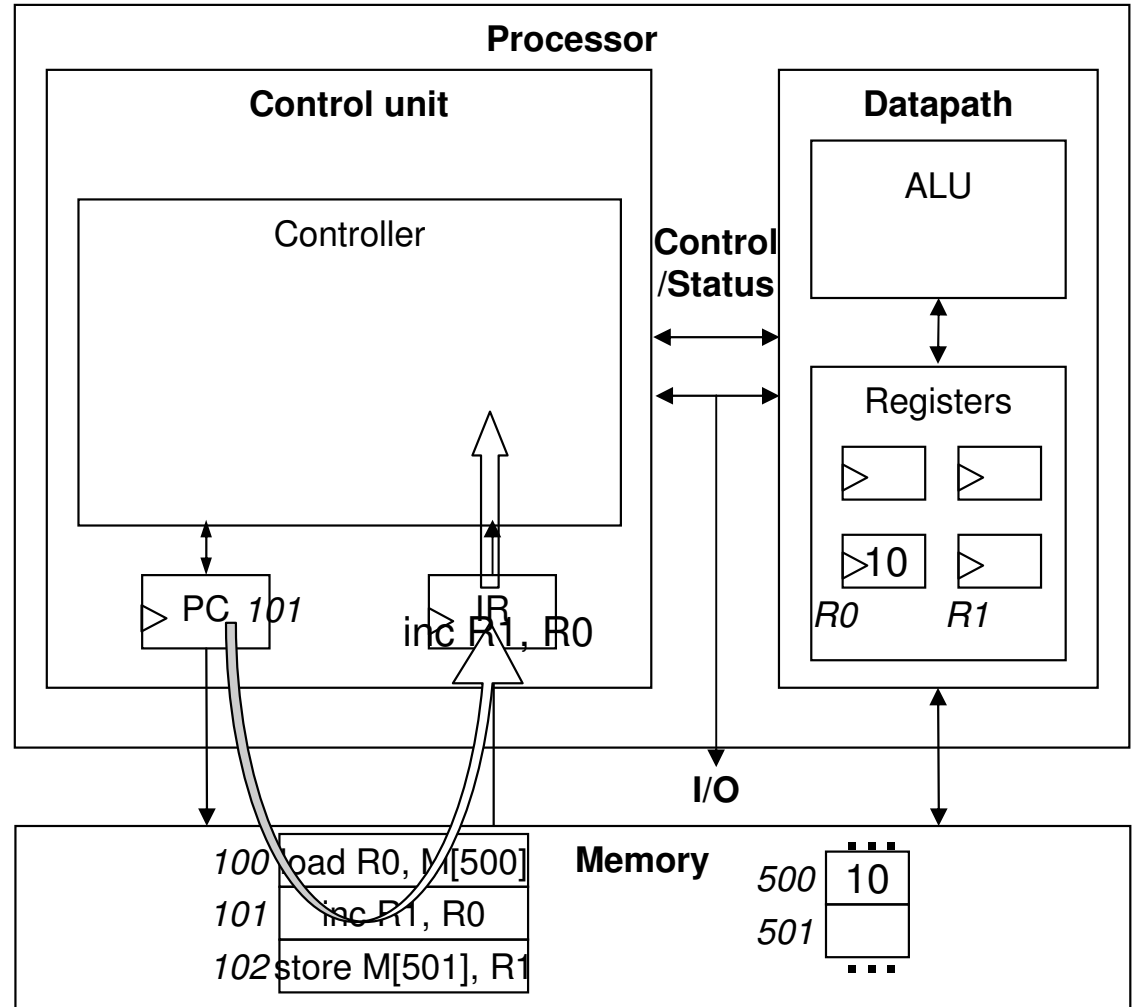
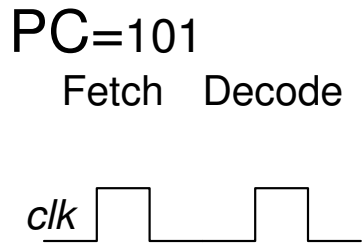
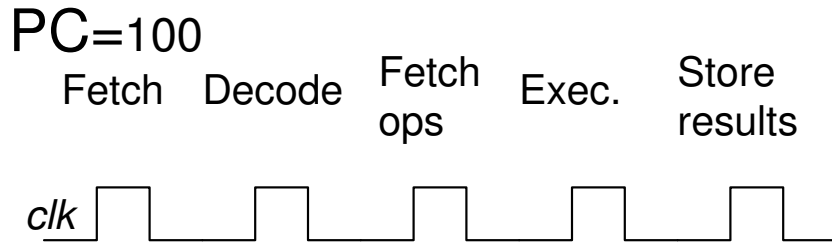
Instruction Cycles



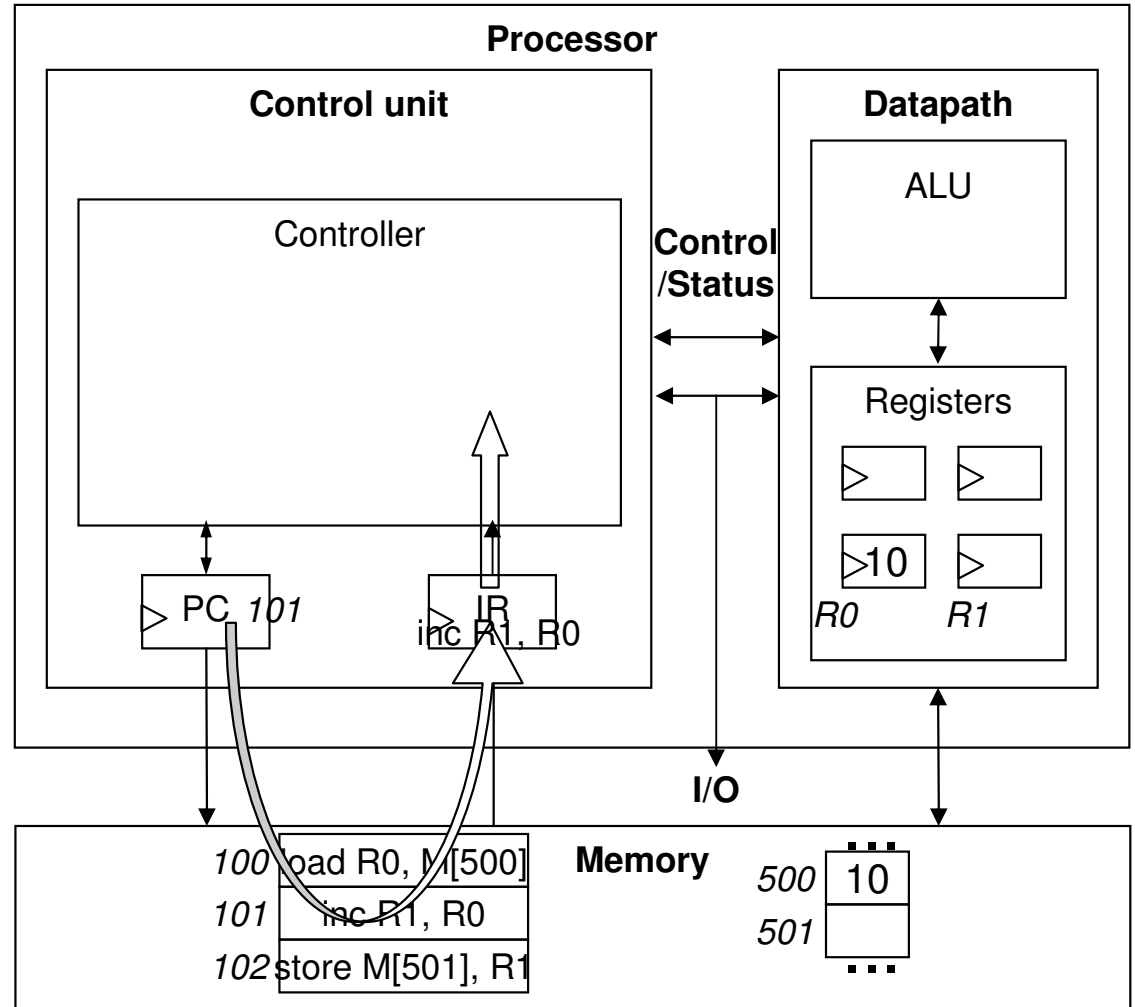
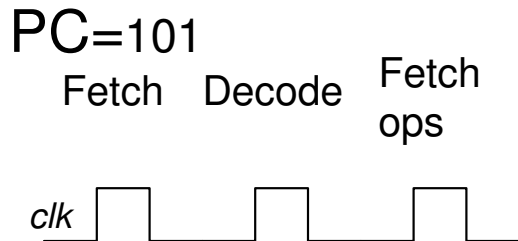
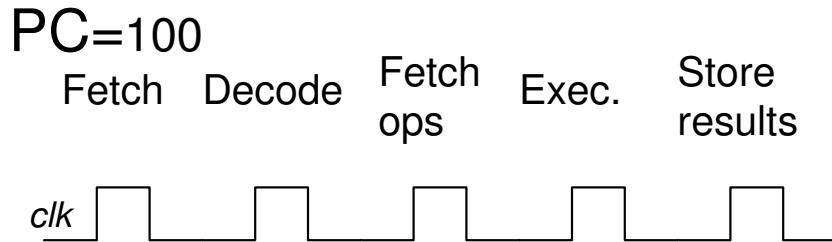
Instruction Cycles



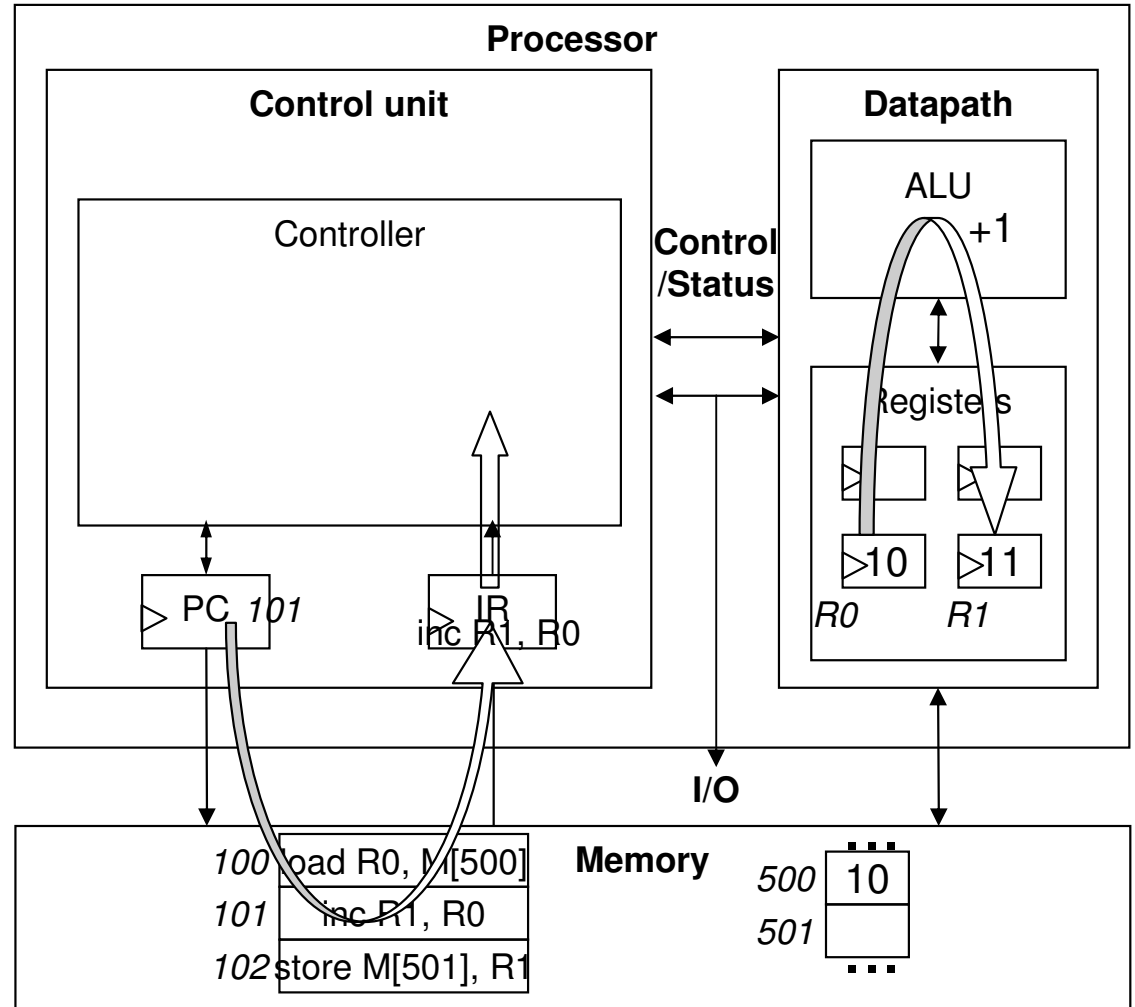
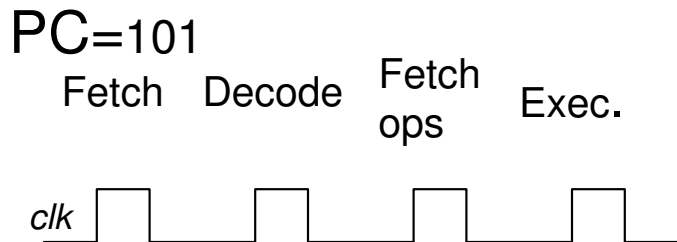
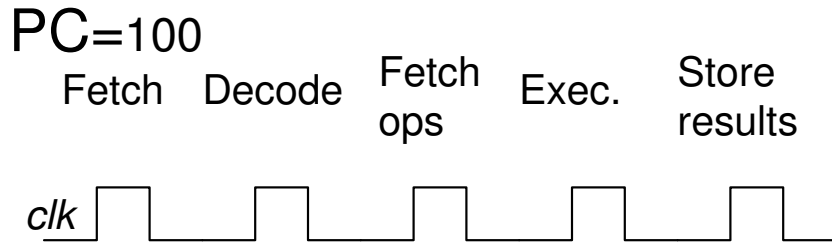
Instruction Cycles



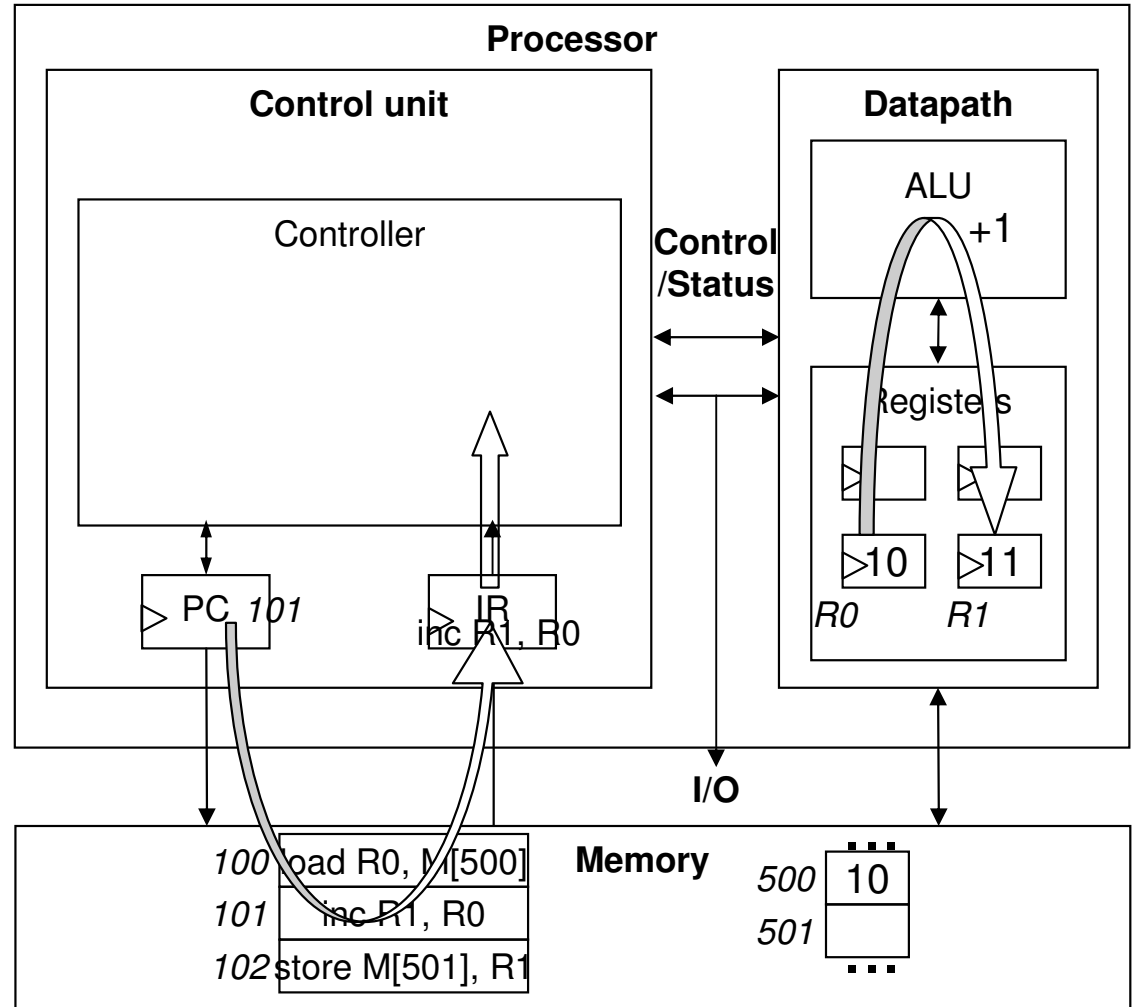
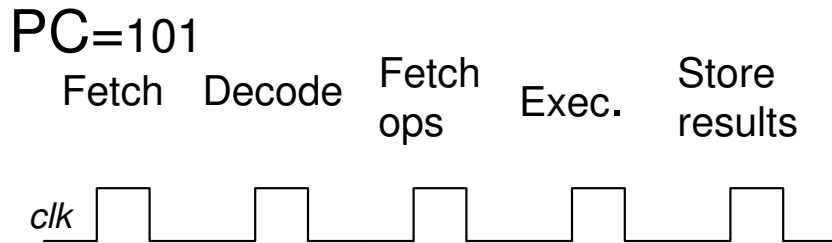
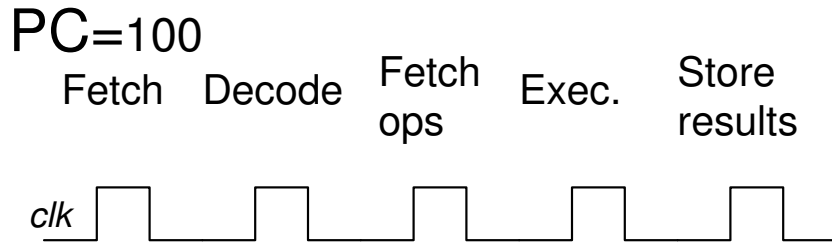
Instruction Cycles



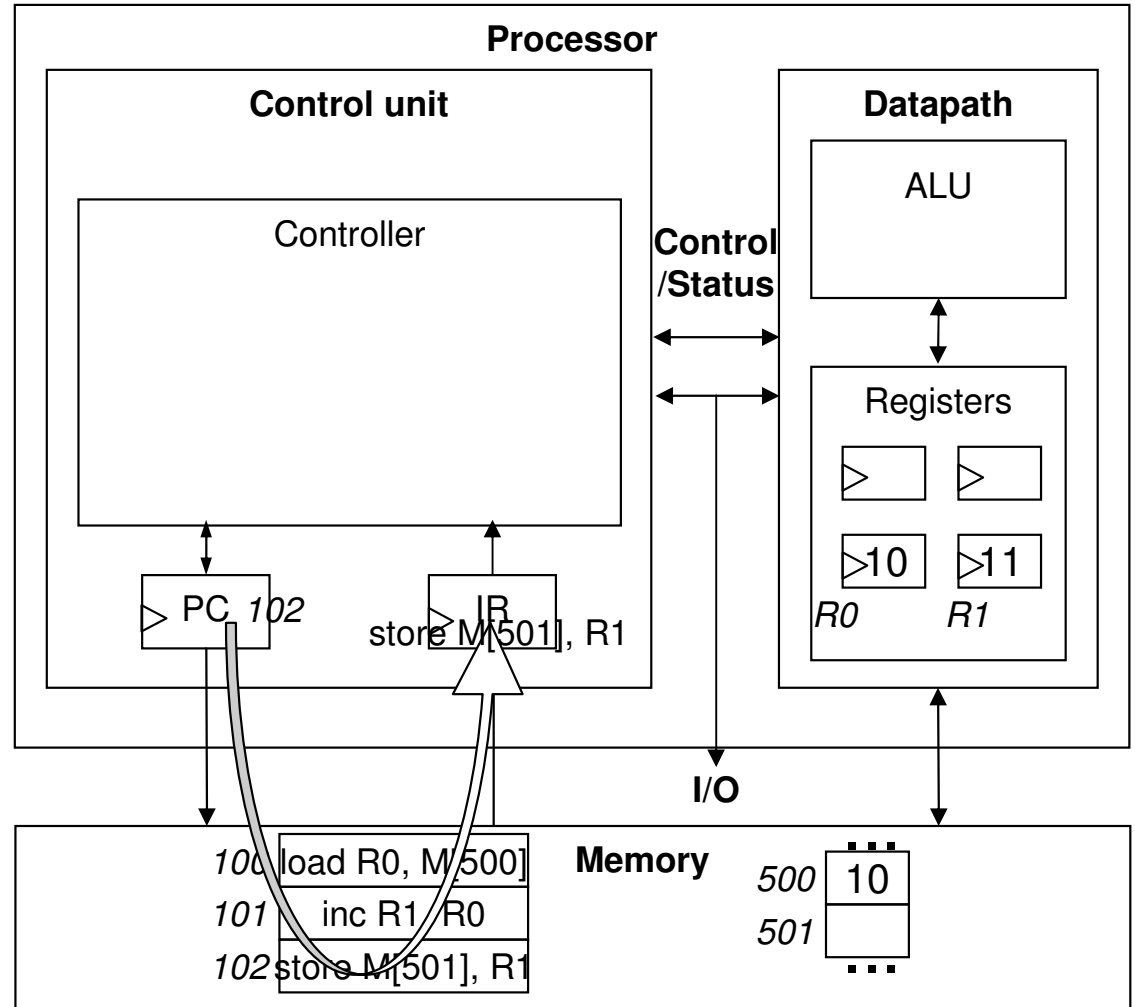
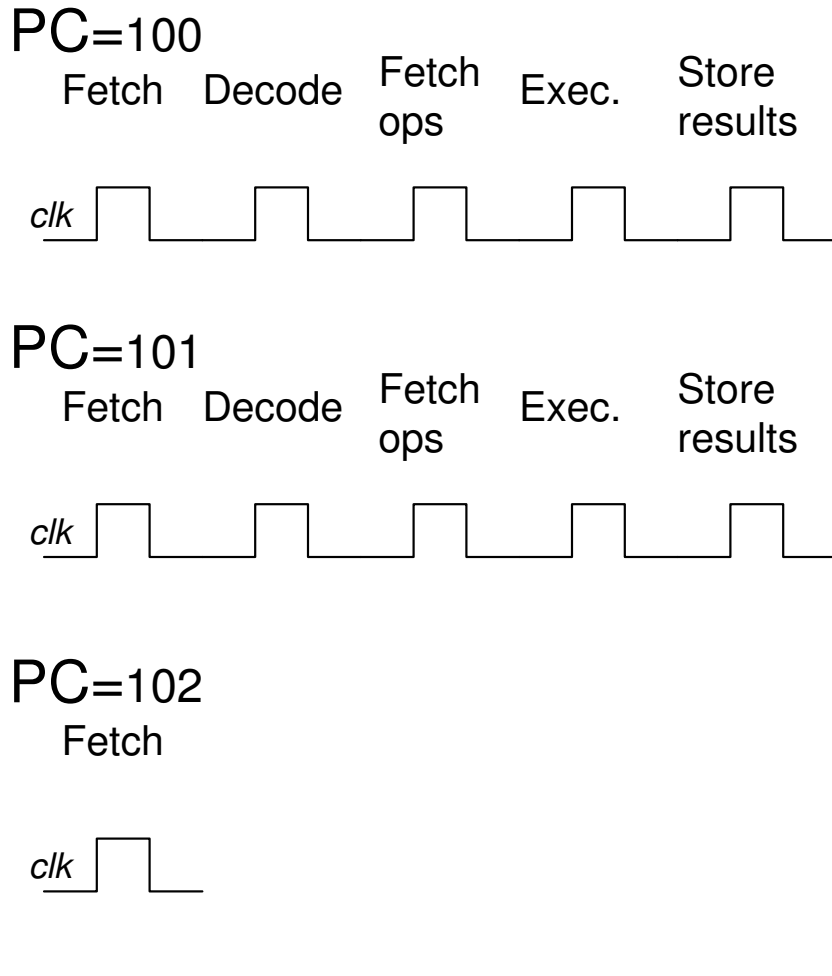
Instruction Cycles



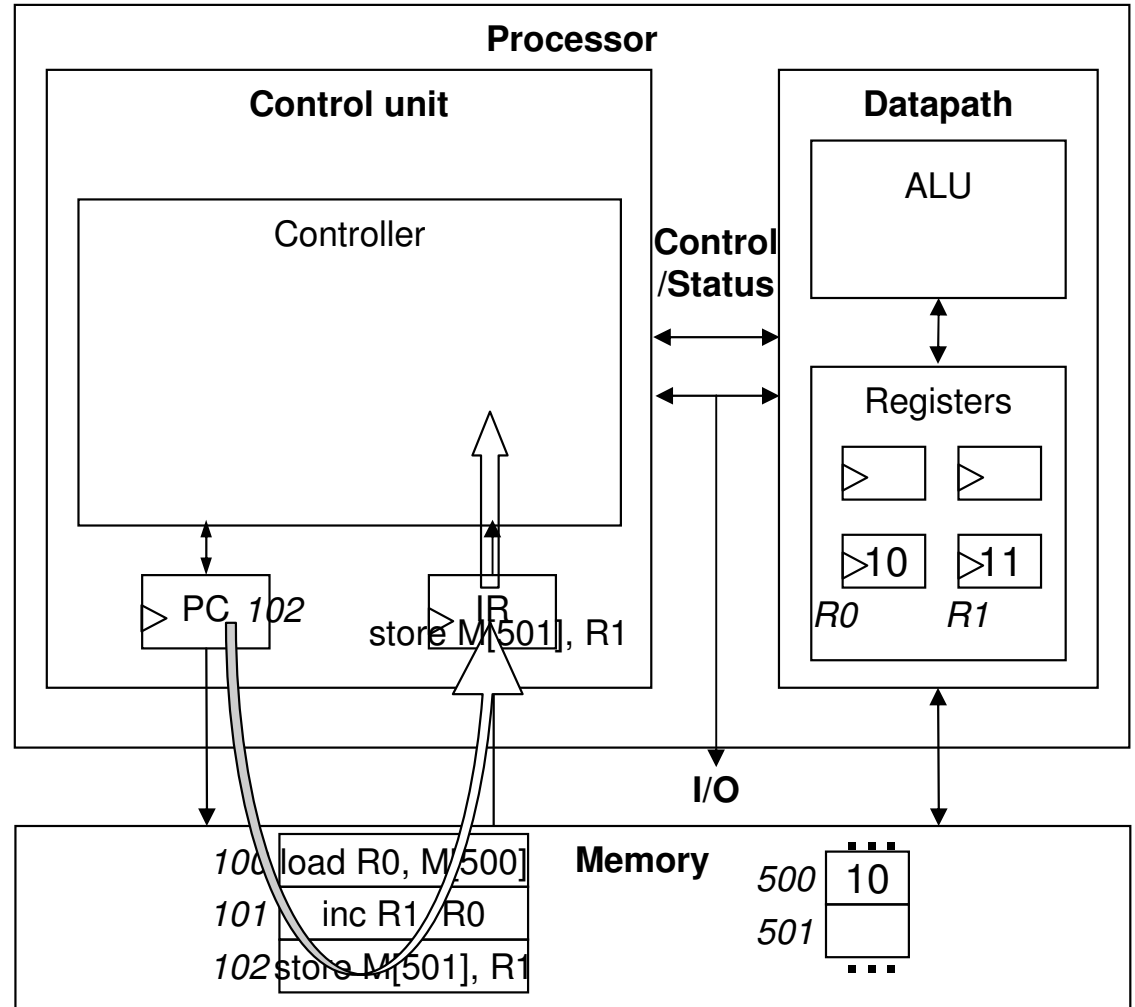
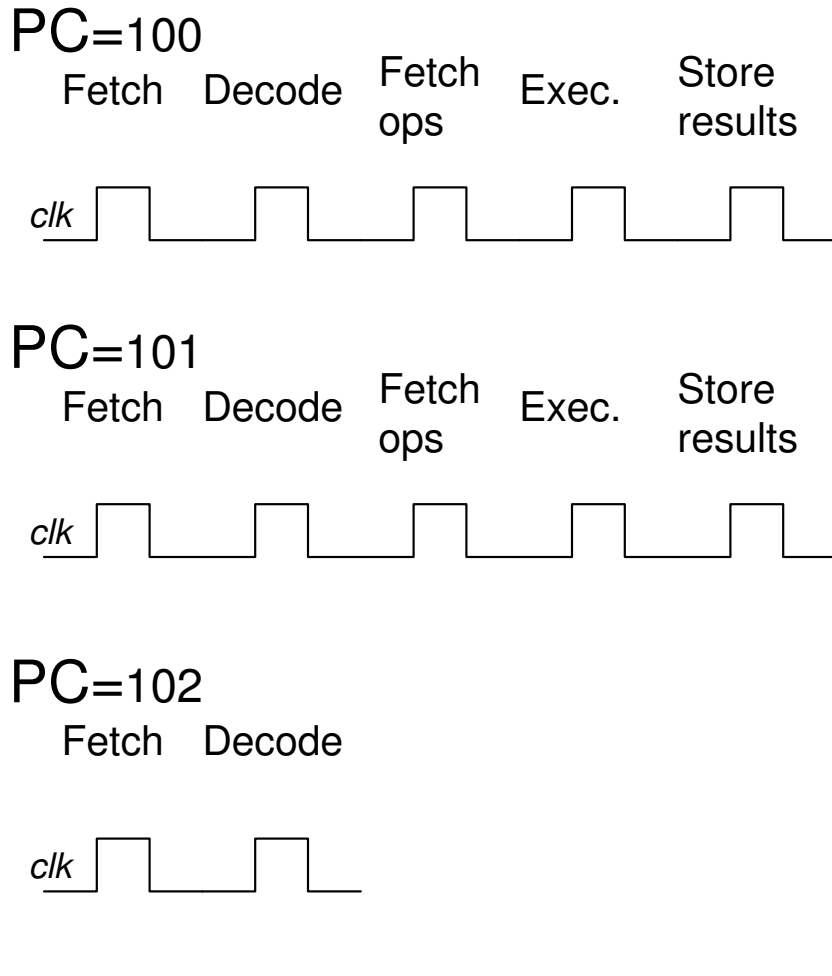
Instruction Cycles



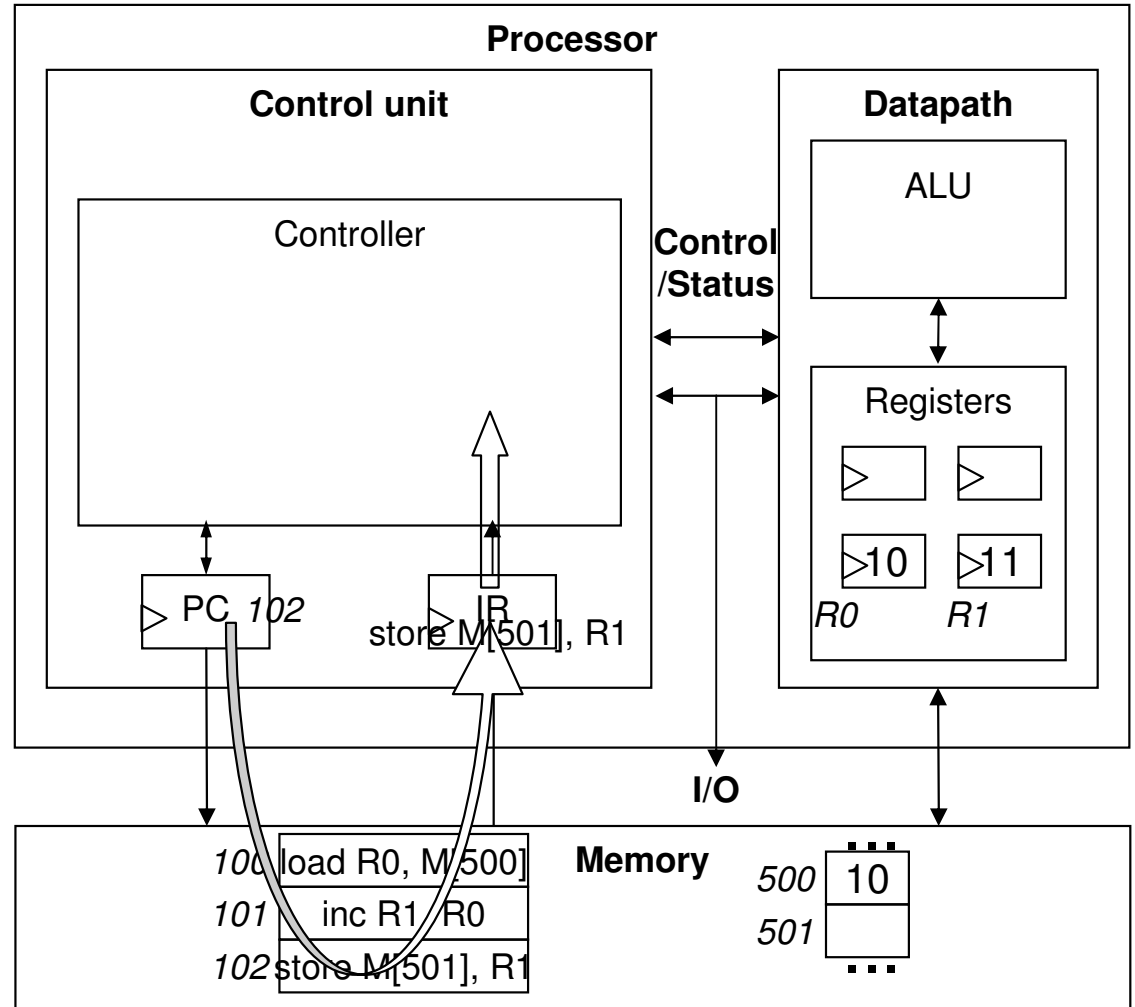
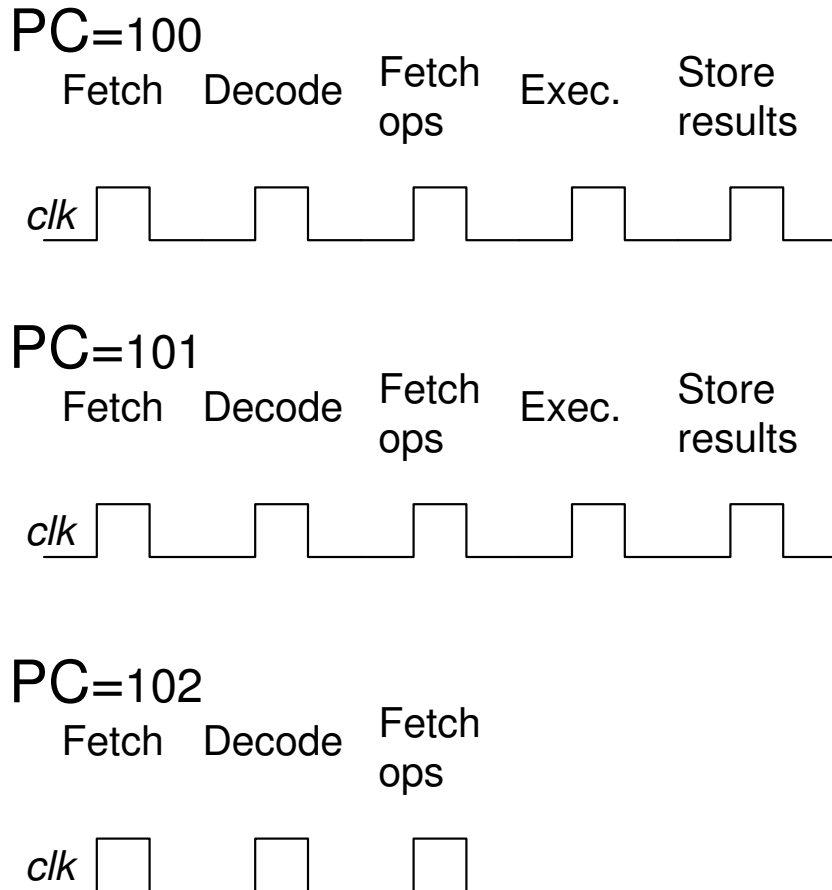
Instruction Cycles



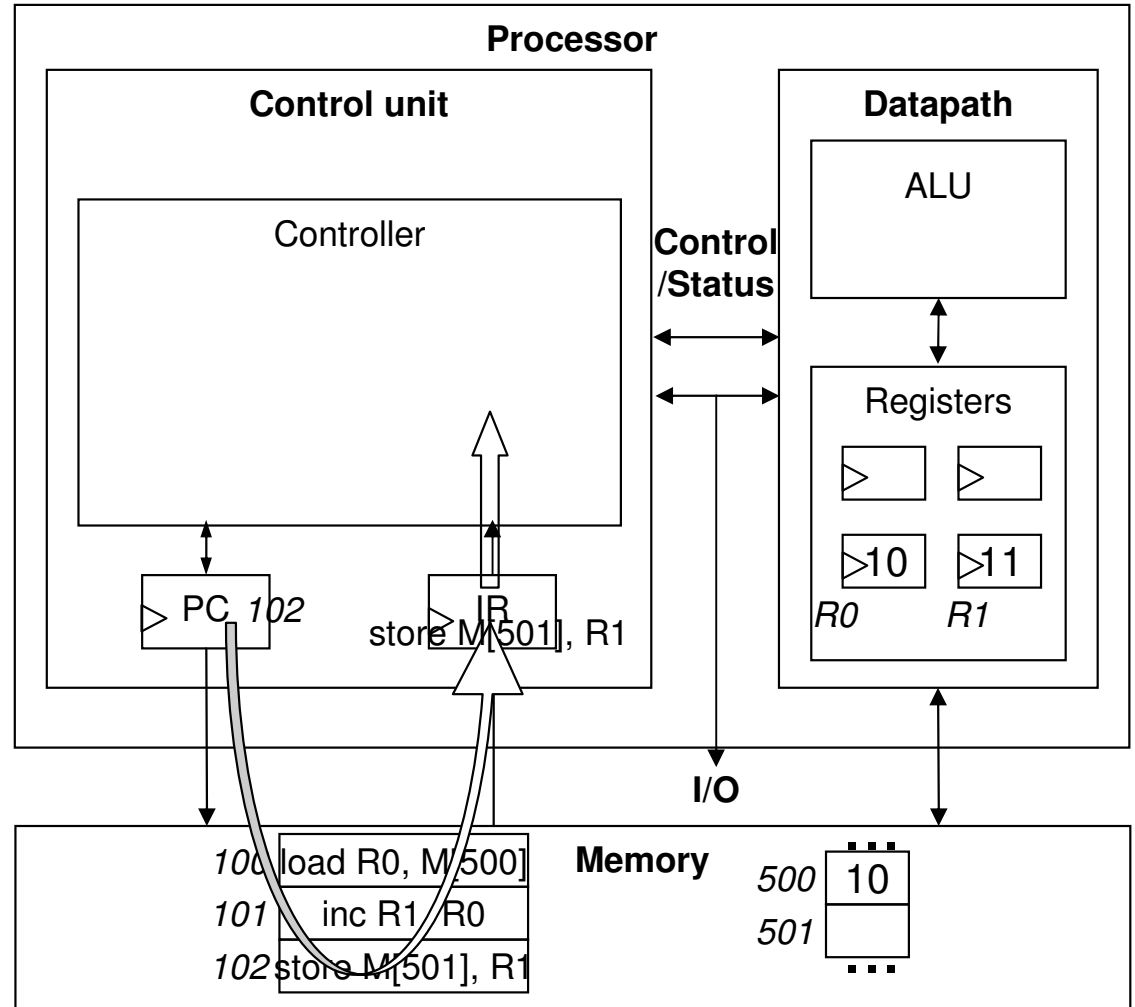
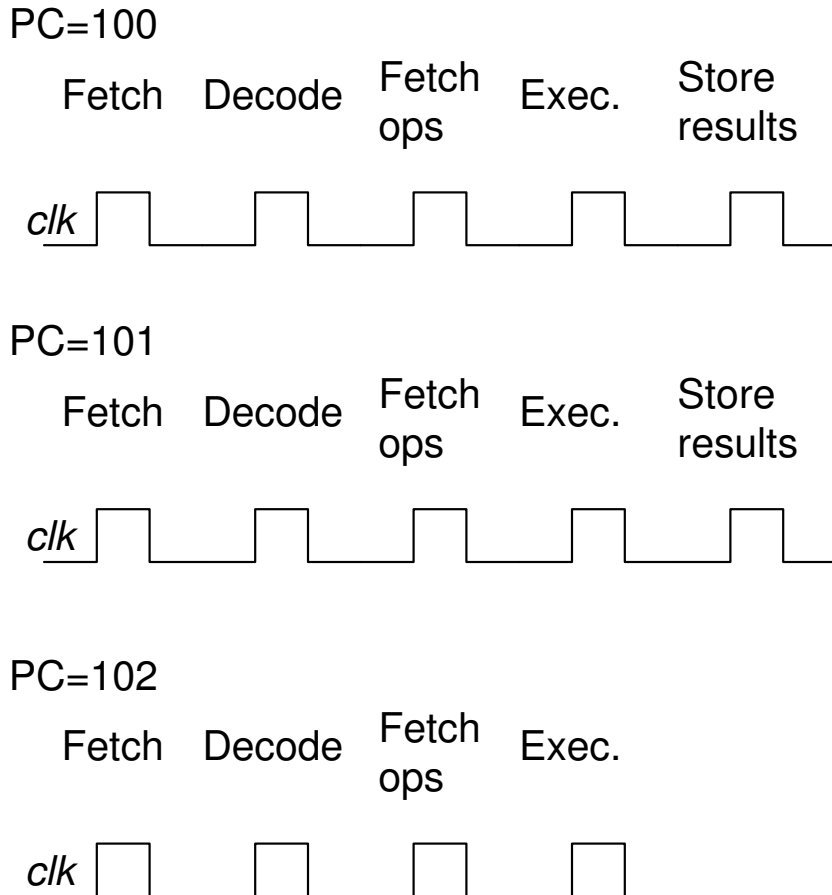
Instruction Cycles



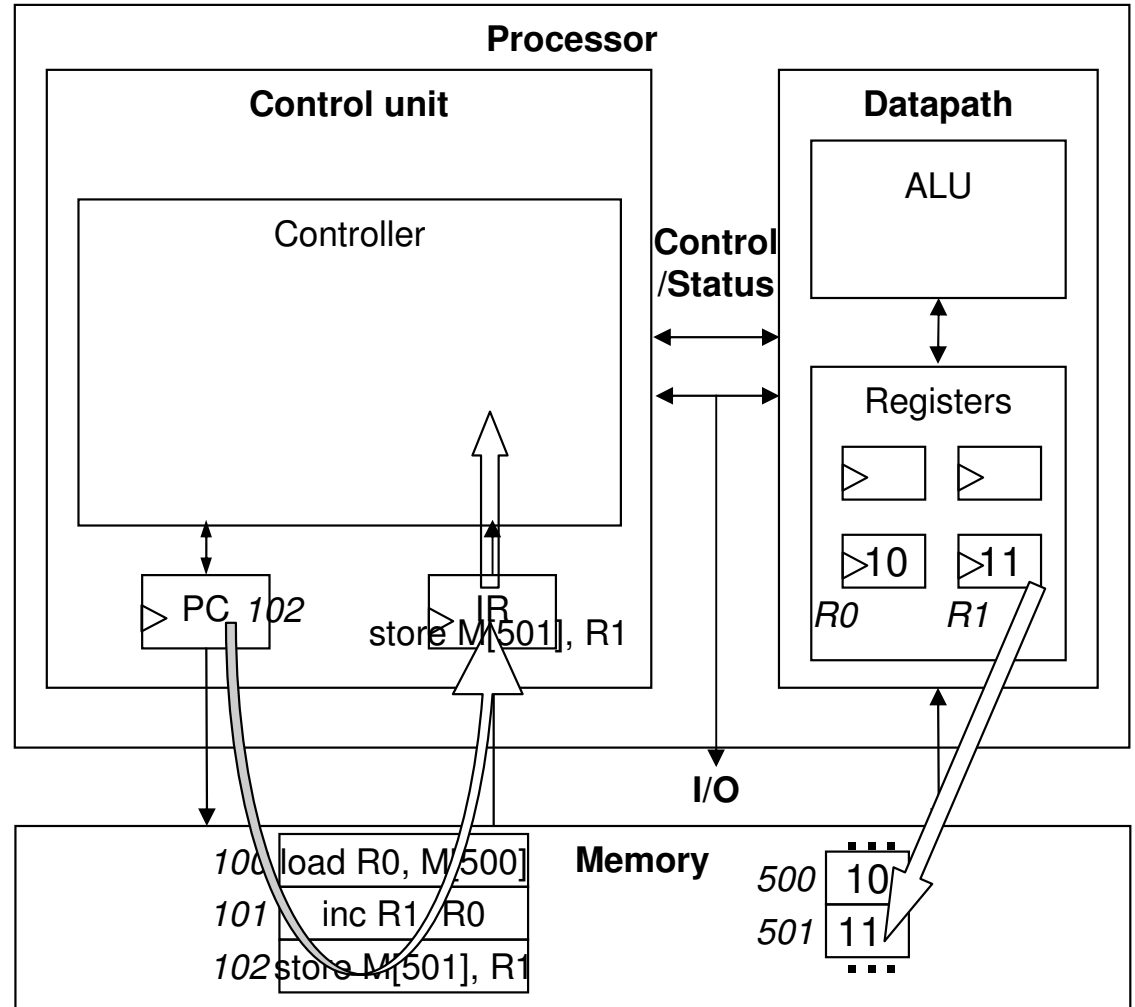
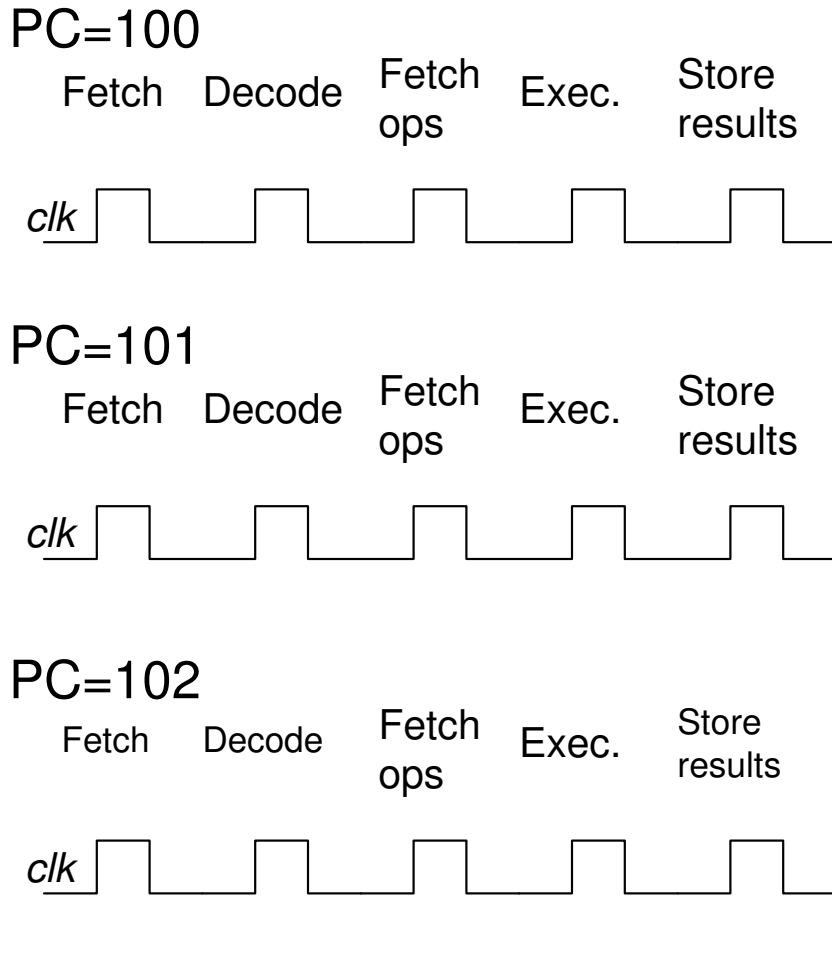
Instruction Cycles



Instruction Cycles



Instruction Cycles



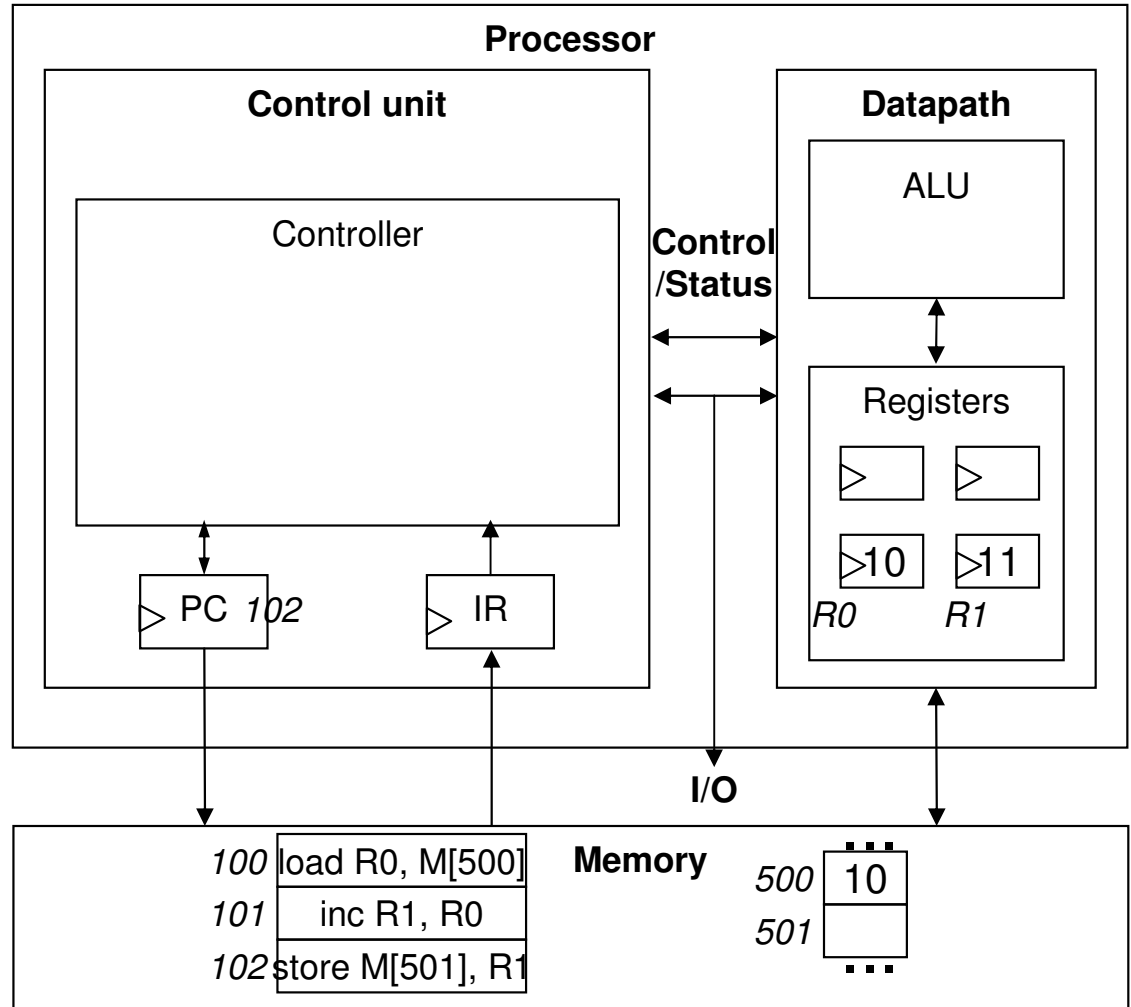
ICE! (but your not actually going to hand it in)

1. What would happen in each of the steps for a command like:

`103 ADD R1, R0`

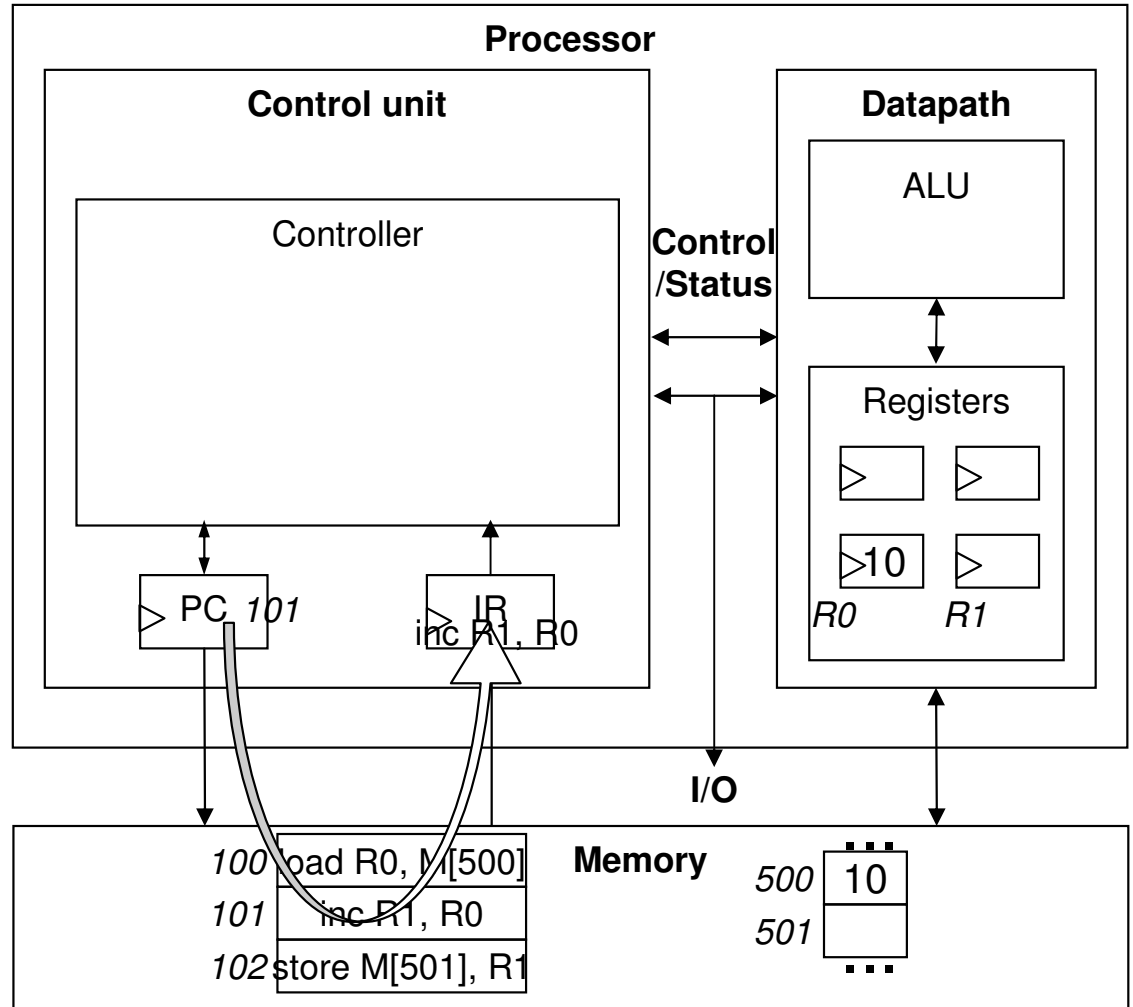
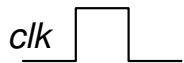
which adds R1 and R0 and stores the result in R1?

2. What functionality are we going to need to add in order to accommodate JUMP commands, which move to different instruction addresses?



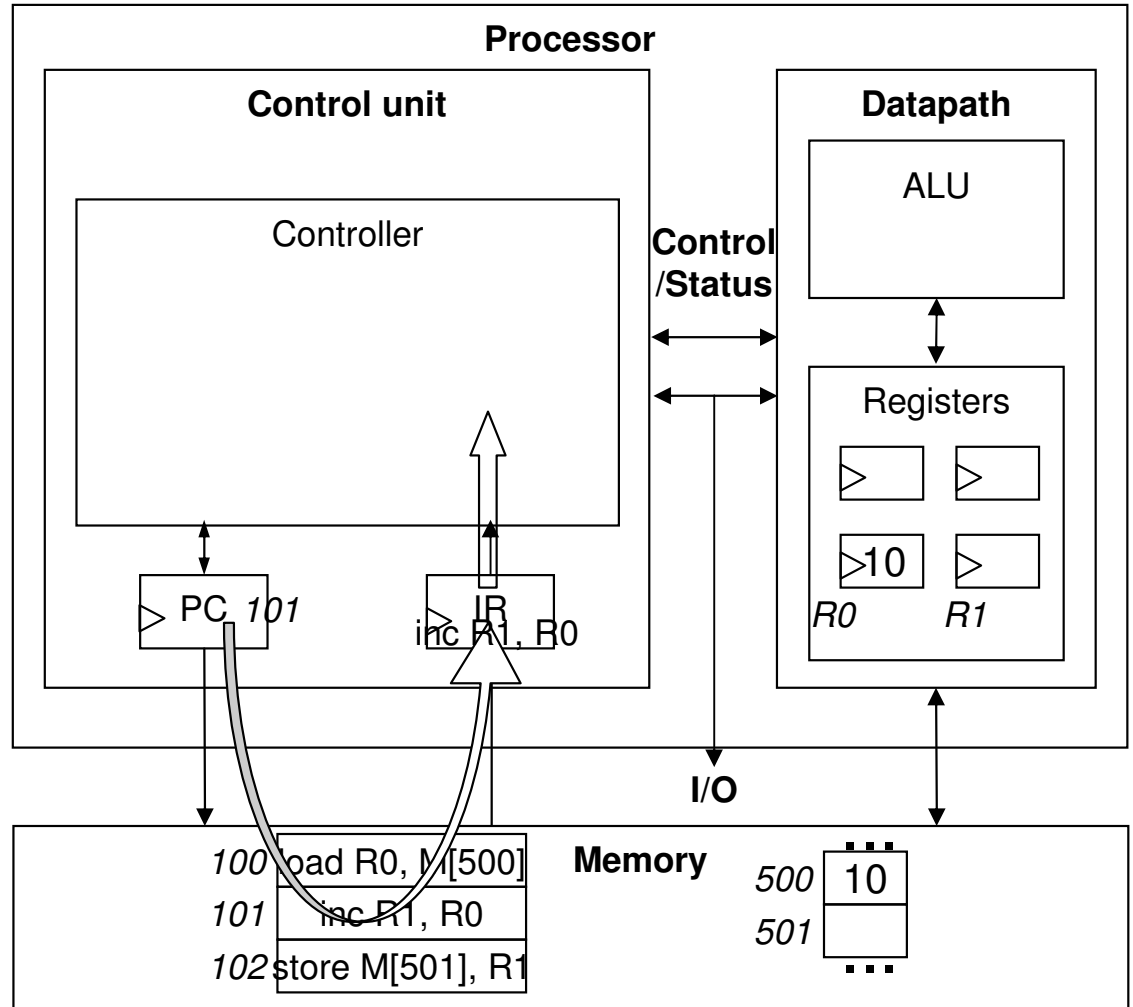
Instruction Cycles

PC=103
Fetch

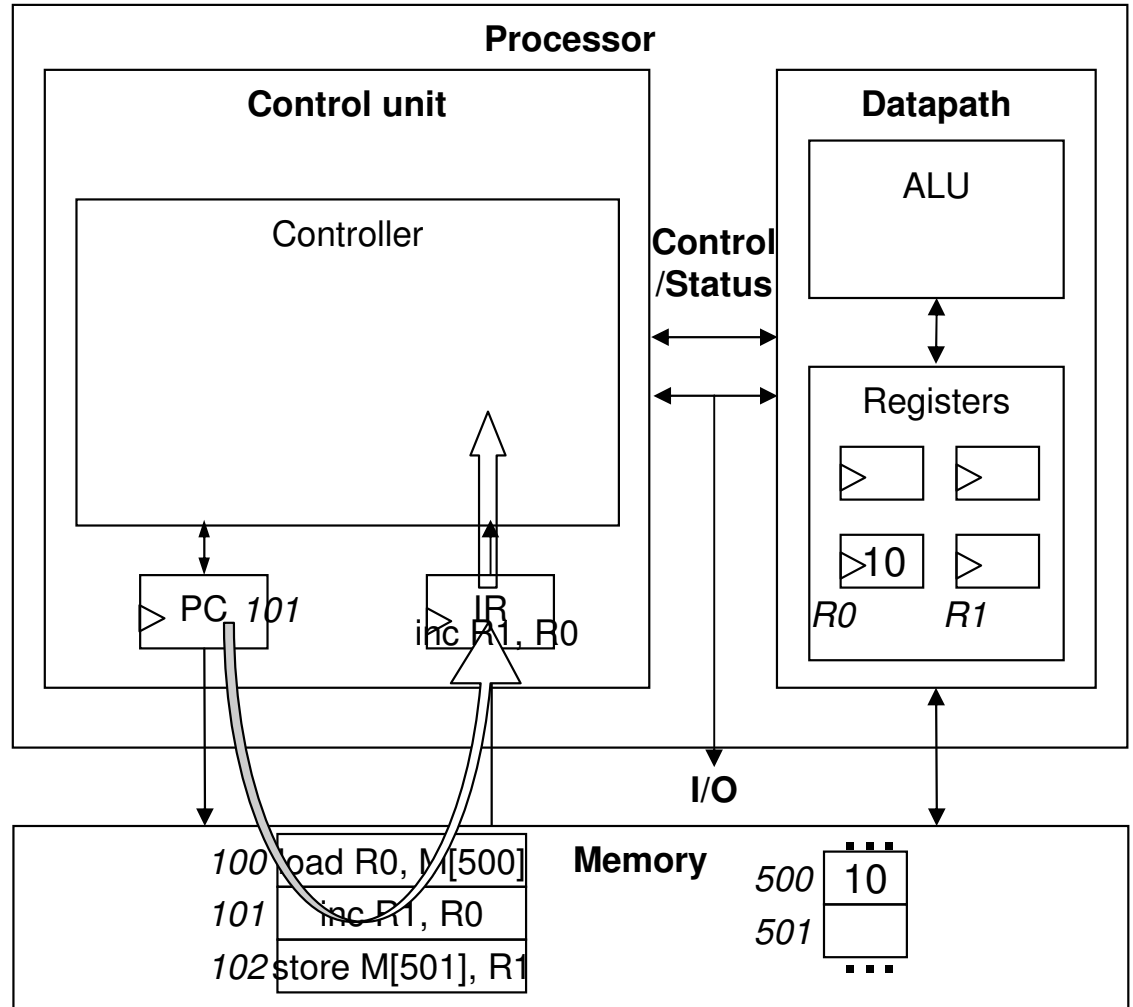
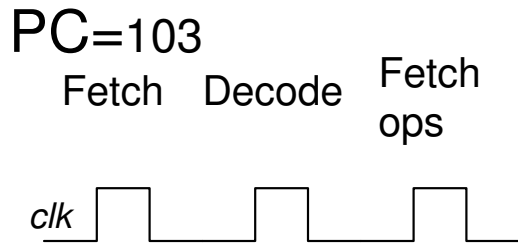


Instruction Cycles

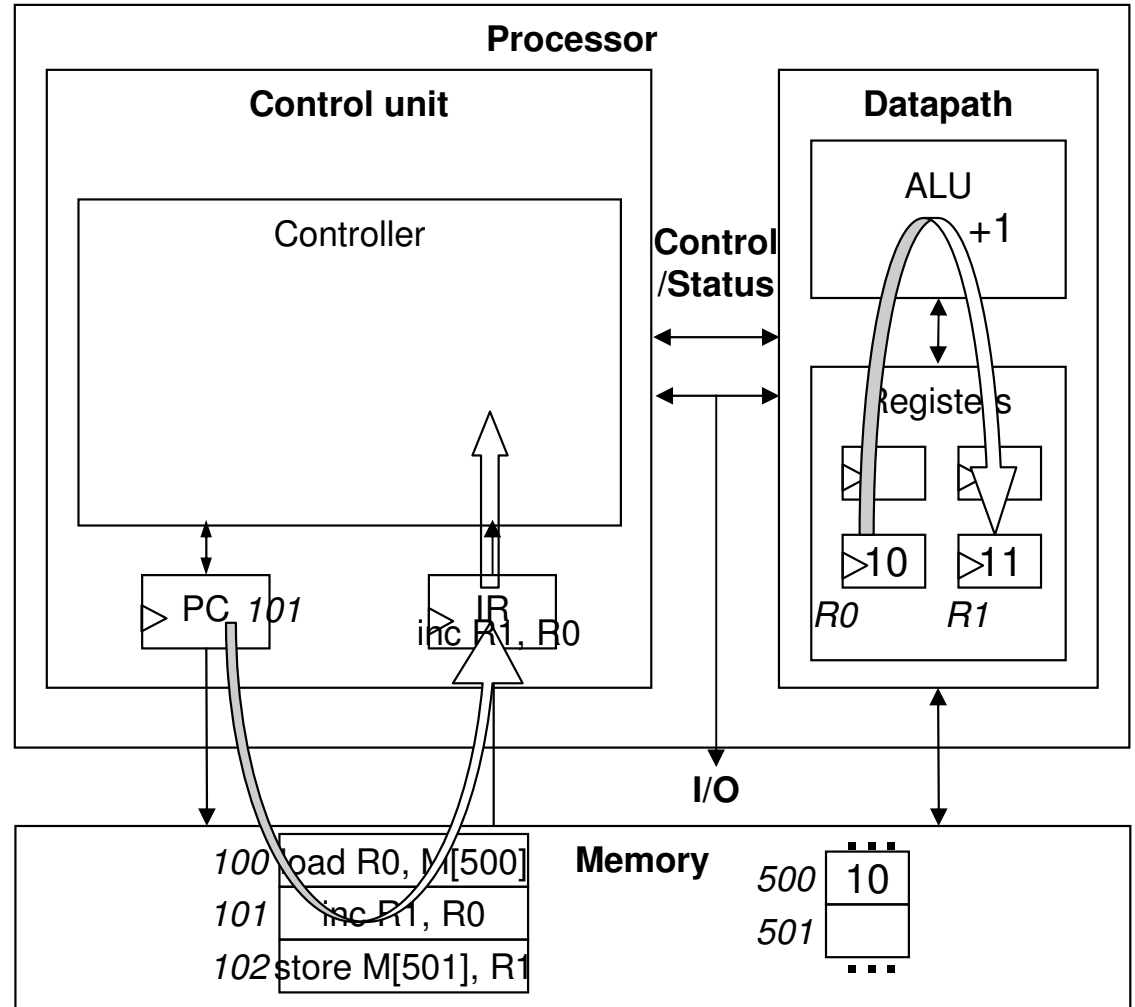
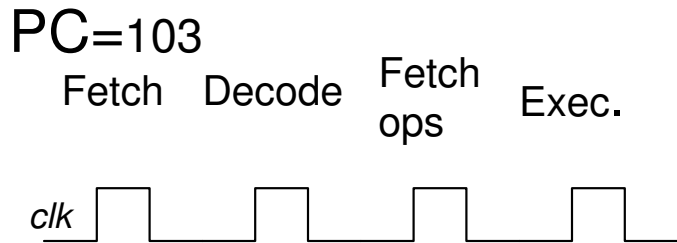
PC=103
Fetch Decode



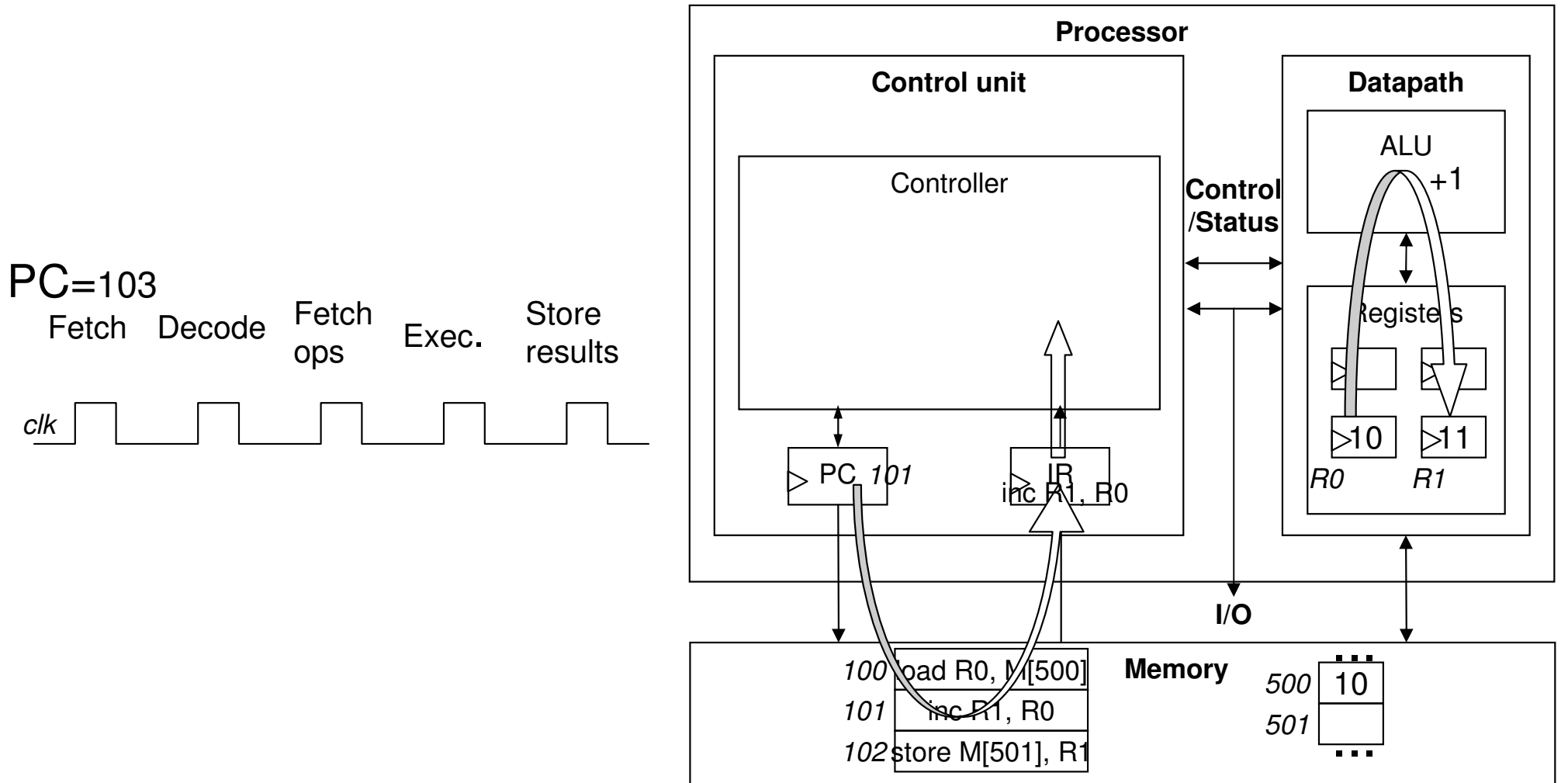
Instruction Cycles



Instruction Cycles

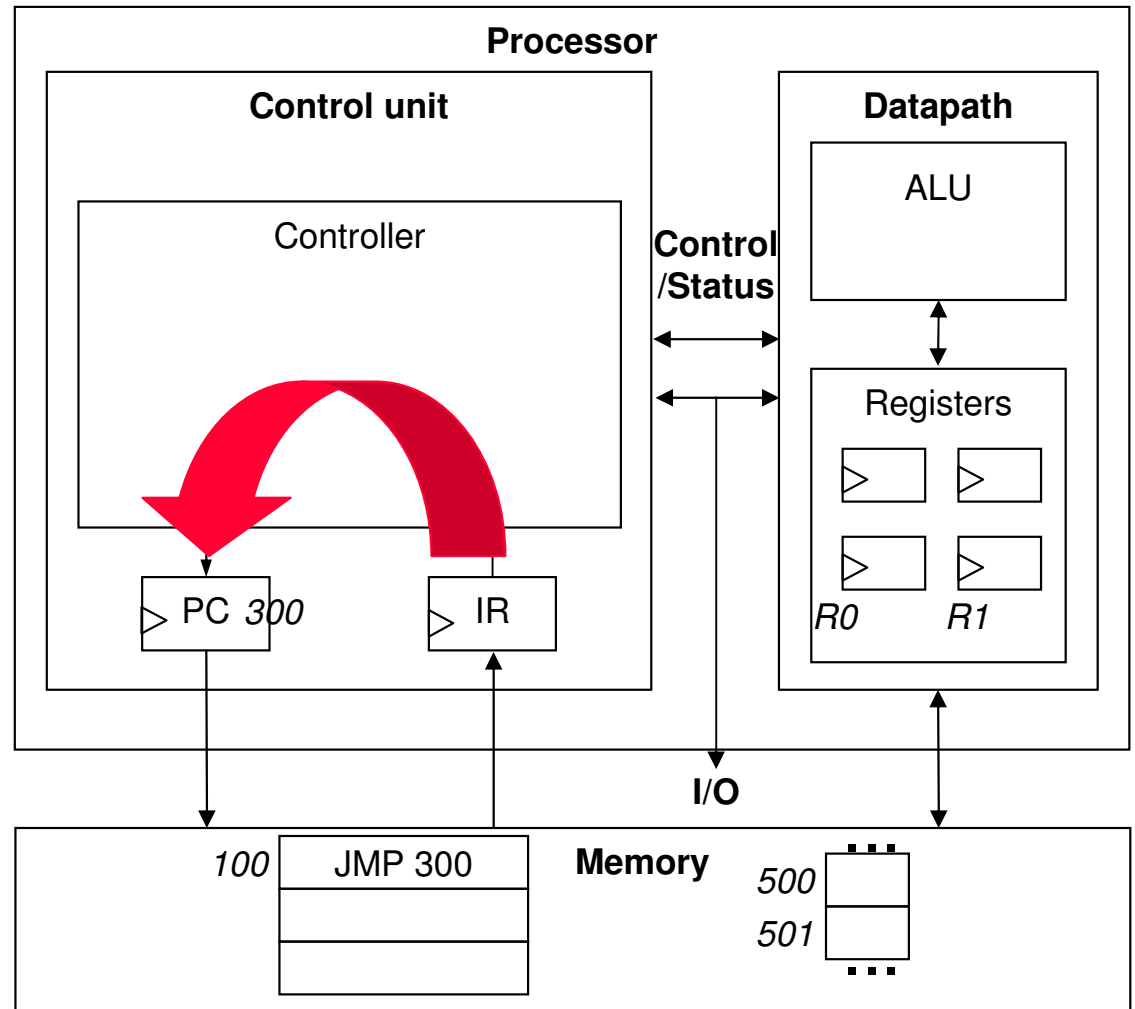


Instruction Cycles



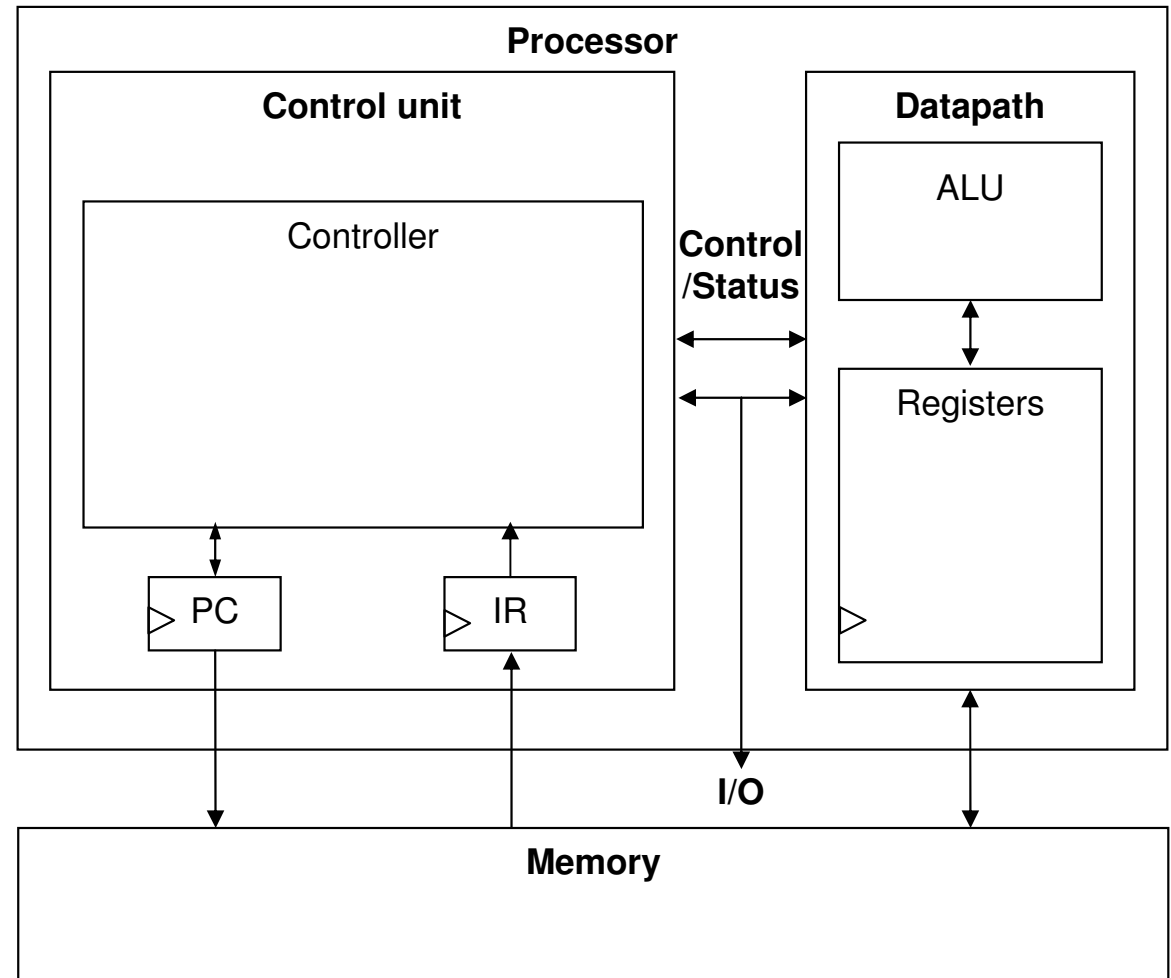
Instruction Cycles

In order to accommodate JUMP commands, you will need some functionality to read a value in the IR and write (a part of) it to the PC.



Architectural Considerations

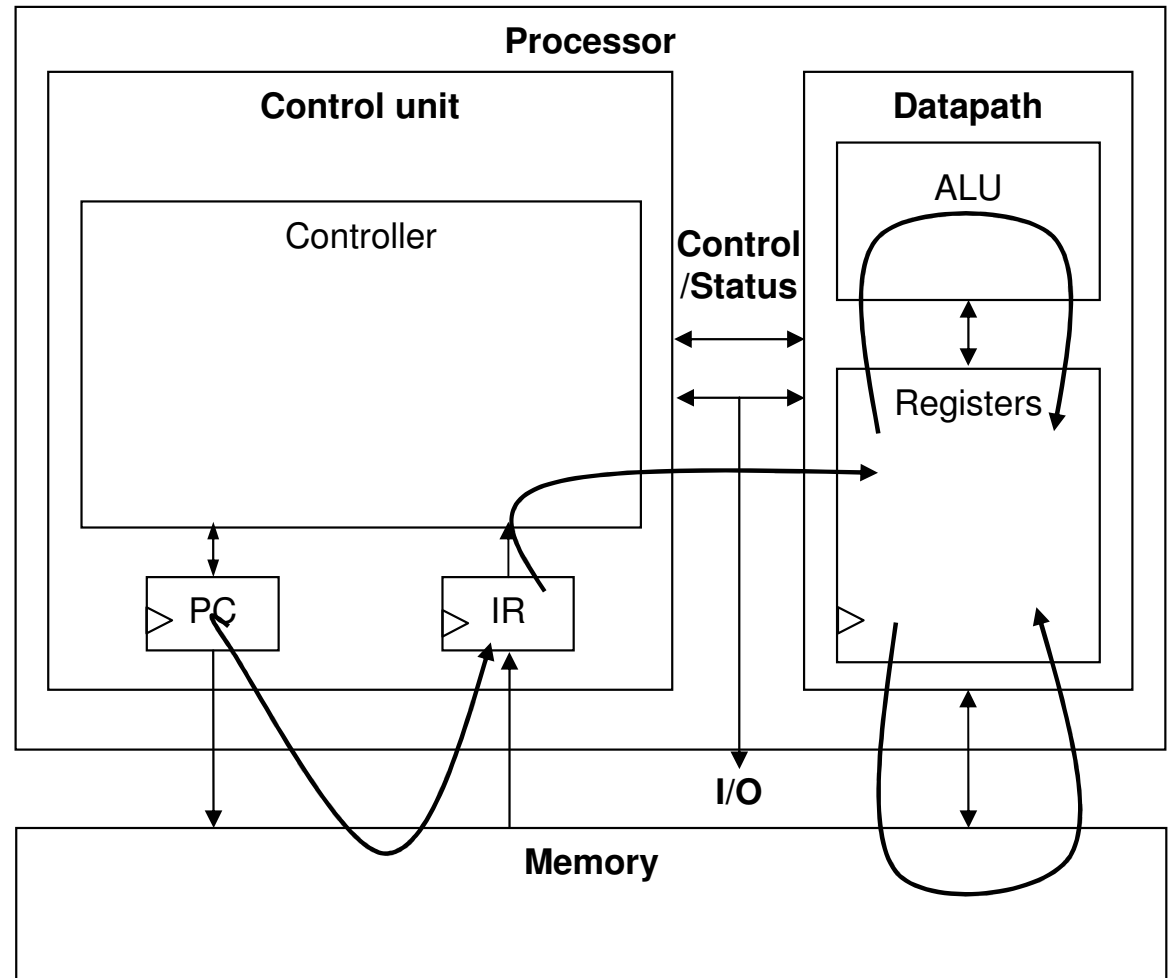
- *N-bit* processor
 - N-bit ALU, registers, buses, memory data interface
 - Embedded: 8-bit, 16-bit, 32-bit common
 - Desktop/servers: 32-bit, even 64
- PC size determines address space



In high performance systems, this is a cache.

Architectural Considerations

- Clock frequency
 - Inverse of clock period
 - Must be longer than longest register to register delay in entire processor



To Do before Tuesday...

- Read page 71 in ESD