Computer Systems

Kasun Gunawardana

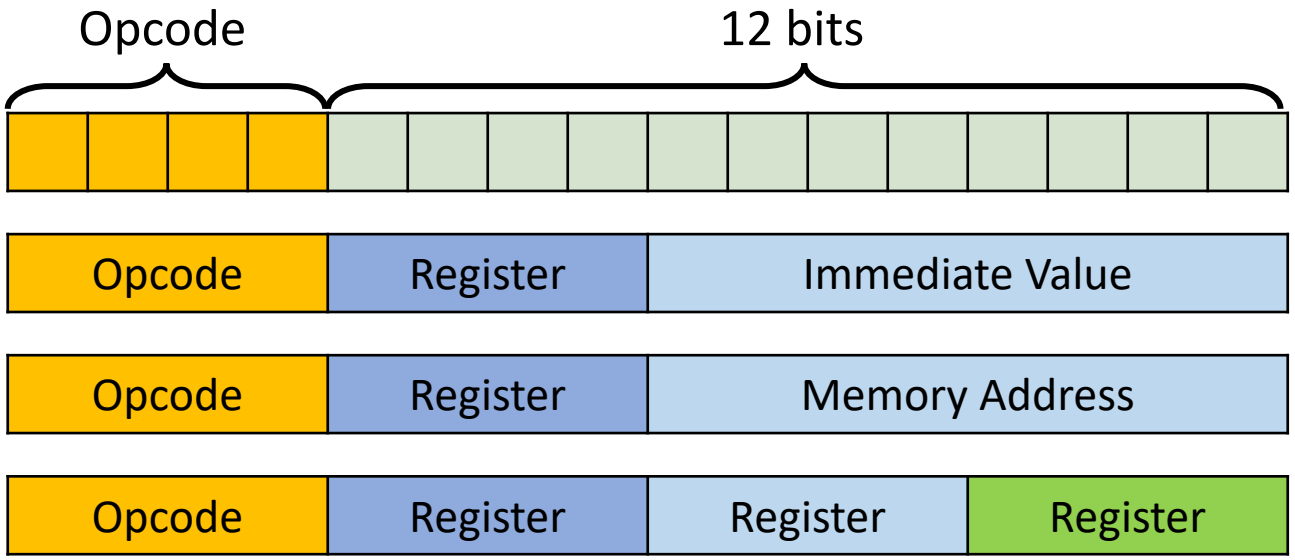
E-mail: kgg

University of Colombo School of Computing

Addressing Modes

Addressing Modes

- Addressing modes are to specify where the instruction operands are located.
- E.g. -



- An addressing mode can specify a constant, a register, or a location in memory.

Common Addressing Modes

- Immediate
- Direct
- Indirect
- Register
- Register indirect
- Displacement
- Stack

Typically, computer architectures provide multiple of these addressing modes.

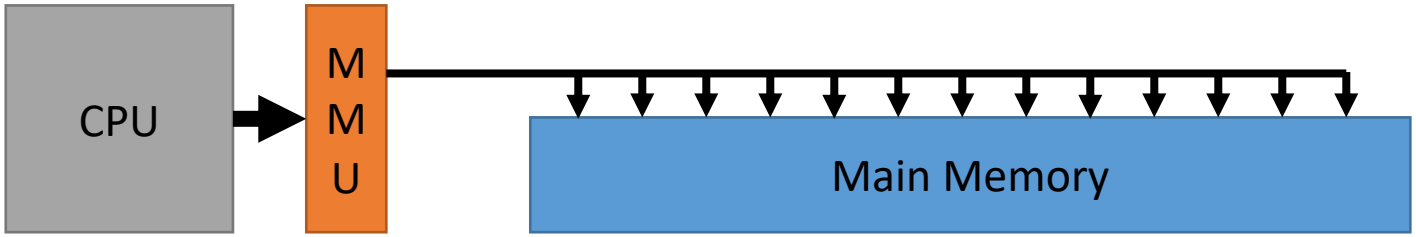
How does the CPU determine the address mode that is used in an instruction ?

Effective Address

- The location of the actual operand is called the effective address of the operand.
- In a computer system without virtual memory,
  - the effective address will be either a main memory address or a register.
- In a computer system with virtual memory,
  - the effective address is a virtual address or a register.

Physical Address

- Finding the actual physical address is a responsibility of the **M**emory **M**anagement **U**nit (MMU).
- MMU is the device that manages and controls the access to memory.
- Because of the MMU, finding this actual physical address is invisible to the programmer.



Common Addressing Modes

- Immediate
- Direct
- Indirect
- Register
- Register indirect
- Displacement
- Stack

Immediate Addressing

- The instruction contains the operand value
  - The operation code is immediately followed by the value to be referenced.
  - The data item to be operated is available as a part of instruction.

`LI R, I`	LOAD the register R with I (I is called an immediate number)

```
LI   R1, 0x0A
```

Immediate Addressing

- Pros
  - Very fast because the value to be loaded is included in the instruction.
- Cons
  - Less flexible due to the value to be loaded is fixed at compile time.
  - The size of the number is restricted to the size of the address field

Direct Addressing

- The instruction's address field contains the effective address of the operand.
  - The effective address of the data item to be operated is available as a part of instruction.

LOAD R, A	LOAD the register R with the content of memory cell A

```
LOAD   R1, 0x80
```

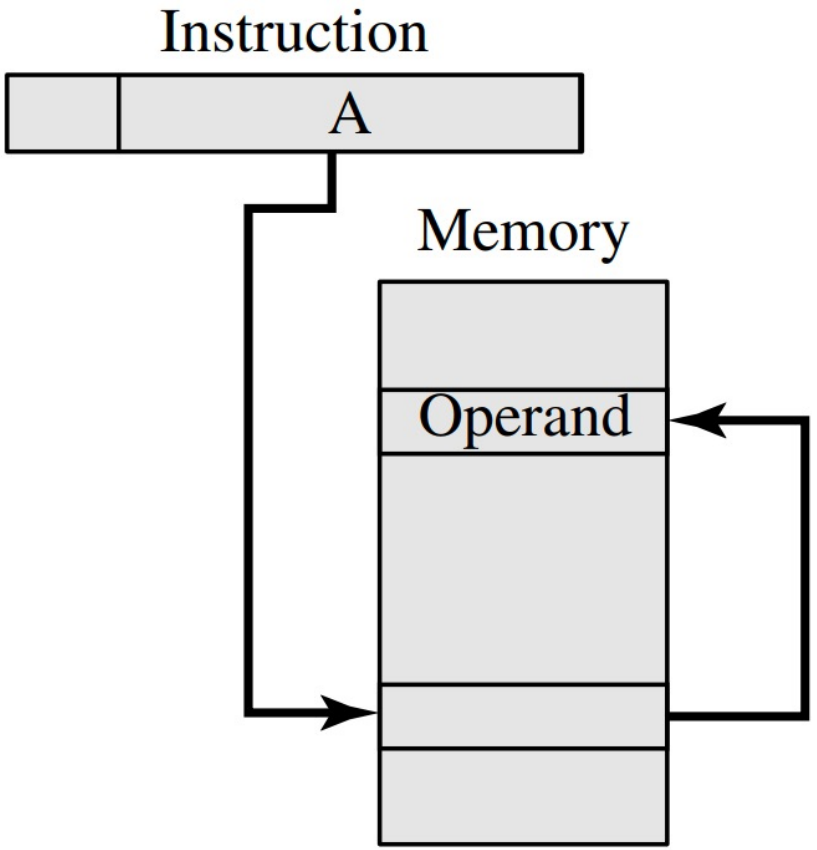
Direct Addressing

- Pros
  - Fast because the value to be loaded can be accessed in a single reference.
  - More flexible as the value is not included in the instruction.

- Cons
  - It provides only a limited address space (just a limitation)

Indirect Addressing

- The instruction's address field contains a pointer to the effective address of the operand.



E.g. – Indirect Addressing

- Let's say we have an array of numbers stored in memory starting at address 0x1000, and we want to access the value at index 2 in the array.
- Let’s assume register R1 holds the base address of the array (0x1000) and R2 holds the index we want to access (2)
- First, we need to calculate the address of the desired element using indexed addressing

```
ADD R3, R1, R2
```

- Now we have calculated the address of the desired element

E.g. – Indirect Addressing (Cont.)

- Next, we need to load the value at the calculated address into a register using indirect addressing.

```
LD R4, (R3)
```

- **(R3)** Use of brackets we indicate the reference to the value pointed by the address enclosed within it.

C Language

- When you declare a pointer in C, such as `int *ptr;`, you're creating a variable (`ptr` in this case) that can store the memory address of an integer.
- When you want to access the value stored at the memory location pointed to by `ptr`, you use the dereference operator (`*`). For example:

```
int x = 42;
int *ptr = &x;  // ptr holds the address of x

int value = *ptr;
// Dereferencing ptr to get the value at the address it points to
// (which is 42 in this case)
```

Indirect Addressing

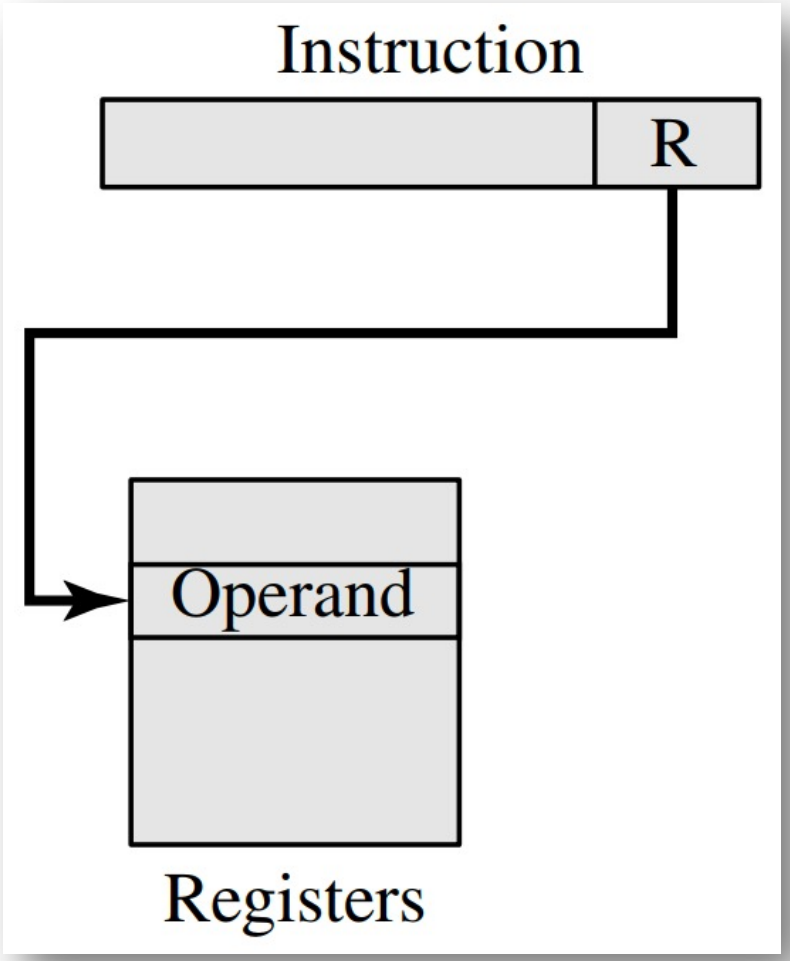
- Pros
  - Availability of larger memory space since entire word can hold an address.

- Cons
  - Instruction execution requires two memory references to fetch the operand: one to get its address and a second to get its value.

Register Addressing (Register Direct)

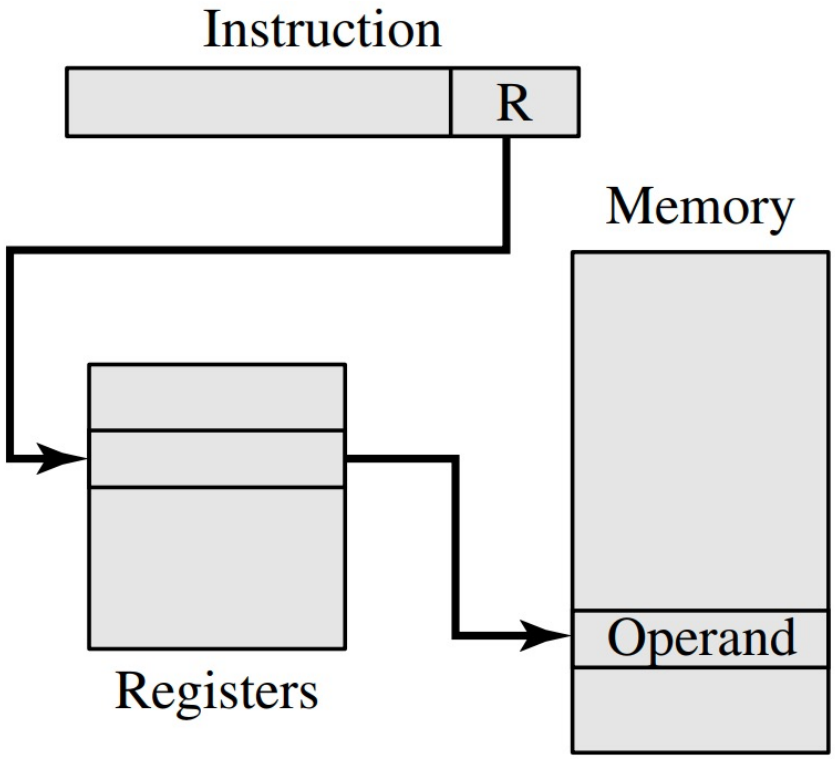
- Similar to direct addressing.
- Instead of memory, a register is used to specify the operand
- E.g. -

```
ADD   R1,   R2,   R3
```



Register Indirect Addressing

- This is a variant of indirect addressing.
- Instruction's operand specifies a register that holds a memory location (effective memory address) of the data item to be used.



Displacement Addressing

- Combination of direct addressing and register indirect addressing.
- The instruction with displacement addressing comes with two address fields.

Activity

0x100	0x210
…	
0x210	0x350
…	
0x350	0x050
…	

```
LOAD   R1,   0x100
```

If the addressing mode for ox100 is as below, what would be the loaded value?

Mode	Loaded Value
Immediate	?
Direct	?
Indirect	?

Instruction Level Pipelining

Instruction Level Pipelining

- Modern CPUs divide the fetch-decode-execute cycle into small steps.
- Each of these steps are made independent, so these smaller steps can be performed in paralell.

The idea is to divide the execution of an instruction into several stages, and then execute each stage of different instructions in parallel.

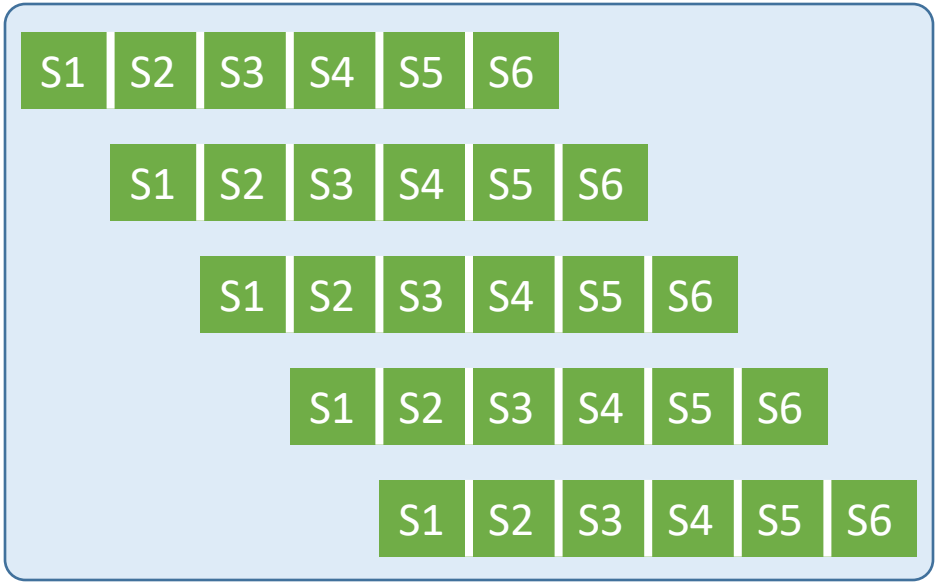
Example Set of Small Steps

1. Fetch instruction
2. Decode opcode
3. Calculate effective address of operands
4. Fetch operands
5. Execute instruction
6. Store result

Instruction Level Pipelining

- Instruction Level Pipelining overlaps these smaller steps of a set of consecutive instructions.

- Need to balance the time taken by each pipeline stage



Pipelining - Efficiency

- Suppose we have a $k - stage$ pipeline.
- Assume the clock cycle time is t_p (it takes t_p time per stage).
- There are n instructions to be processed.
- Instruction 1 (I_1) requires $k \times t_p$ time to complete.
- The remaining $n - 1$ instructions emerge from the pipeline one per cycle, which implies a total time for these instructions of $(n - 1)t_p$.
- Therefore, to complete n instructions using a $k - stage$ pipeline requires:

$$\left(k \times t_p\right) + (n - 1)t_p \quad = \quad (k + n - 1)t_p$$

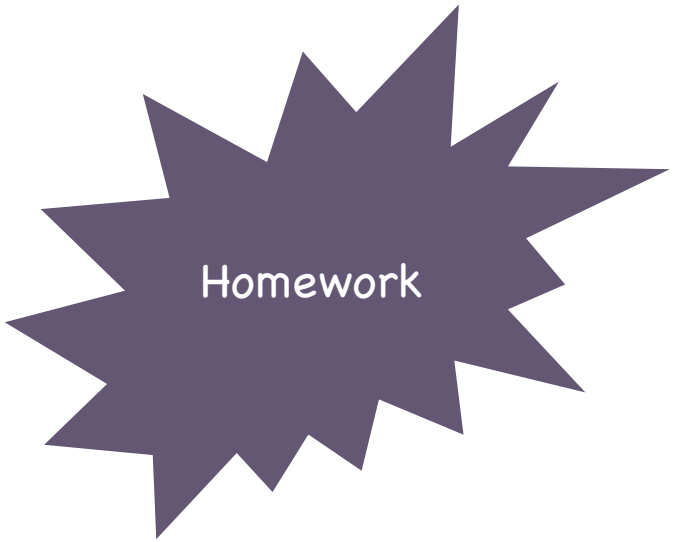
Speedup

- If we calculate the speedup, we gain using a pipeline compared to a CPU without a pipeline

$$Speedup = \frac{(k \times t_p)n}{(k + n - 1)t_p}$$

Instruction Level Pipelining

- Advantages
- Disadvantages
- Limitations
- Issues



Thank You..!