Date.
Page No.

Lec 1

Computer Science and Architecture.

Architecture:
Conceptual design & fundamental operational structure.

- Architecture
  - CPU design
    - Instructions
    - Addressing modes
    - Data Format

Organization:
Implementation of architecture.
Deals with physical devices and their interconnections.
The goal is improving the performance

organization

- I/O organization
- Memory organization
- Performance improvement.

Date.
Page No.

Data format ⇒ representing data in binary

Data Format

- Number
  - Fixed point representation
  - Floating point representation
- characters
  - ASCII
  - EBCDIC

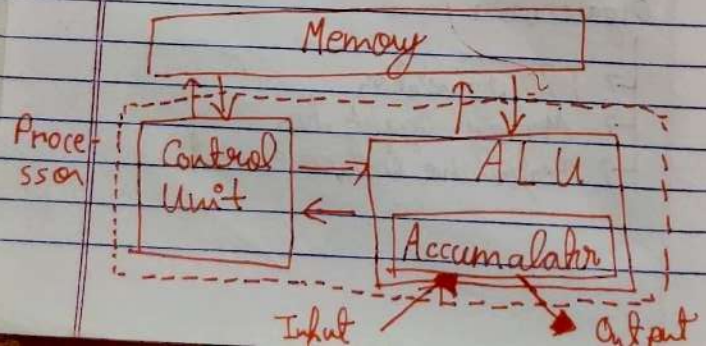
- Unsigned
- signed
  - 1's complement
  - 2's complement
  - signed magnitude

In this course we only cover floating point representation.

Von Neumann's architecture

Also called:
'Shared Program Architecture'

Basic structure is



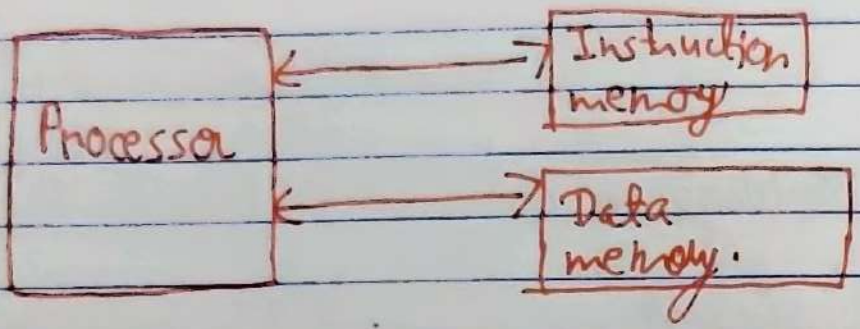
Bottleneck in Von Neumann's architecture:

The instruction & data both cannot be fetched from memory simultaneously.

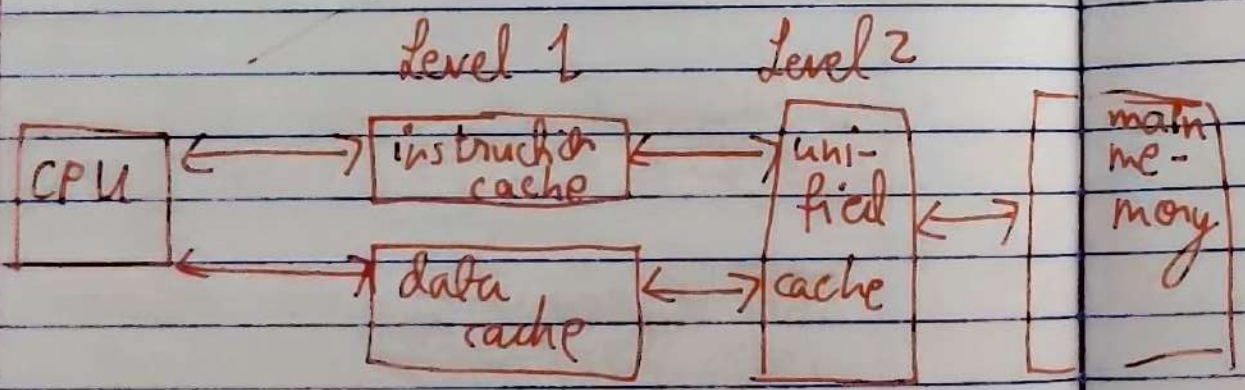
Harvard Architecture

A proposed solution to bottleneck, the memory is split into instruction memory & data memory.

Date.
Page No.



While not implemented completely, it is used to implement cache memory.



Lec 2

Date.
Page No.

Components of Computer

- CPU
  - Control Unit
  - ALU
- Memory
  - Primary / Main memory
  - Secondary / Auxiliary memory
- I/O Devices

Other Component

- System Buses
- CPU registers

System Bus ⇒ Connection line b/w 2 devices within Computer system.

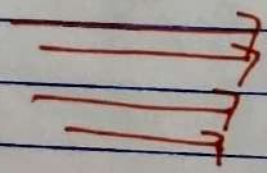
Date.
Page No.

Types of system Buses

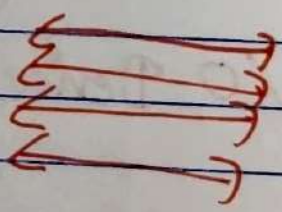
System Buses
- Address Bus
- Data Bus
- Control Bus

* Address Bus is unidirectional
* Data Bus is bidirectional (Duplex)
* Control Bus, every single line is unidirectional.

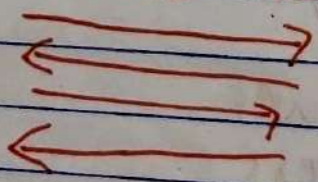
Address Bus:



Data Bus:



Control Bus:



Memory signals for CPU
1) Wait signal
2) Ready signal.

We need these since CPU is faster than memory & memory needs to tell CPU when its busy.

Date. _____
Page No. _____

Memory Cycle Time

Time from accepting a request till memory gets ready again to perform next operation.

CPU Registers

Small memories stored in CPU.

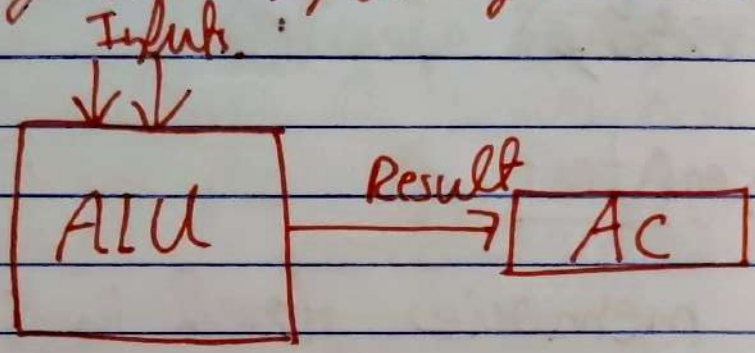
Types

CPU Registers
- General Purpose Registers (GPRs)
- Special Purpose Registers
  - Accumulator (AC)
  - Program Counter (PC)
  - Instruction Register (IR)
  - Stack Pointer (SP)
  - Flag Register / Program Status Word (PSW)
  - Address Register (AR)
  - Data Register (MDR) / MDR Memory

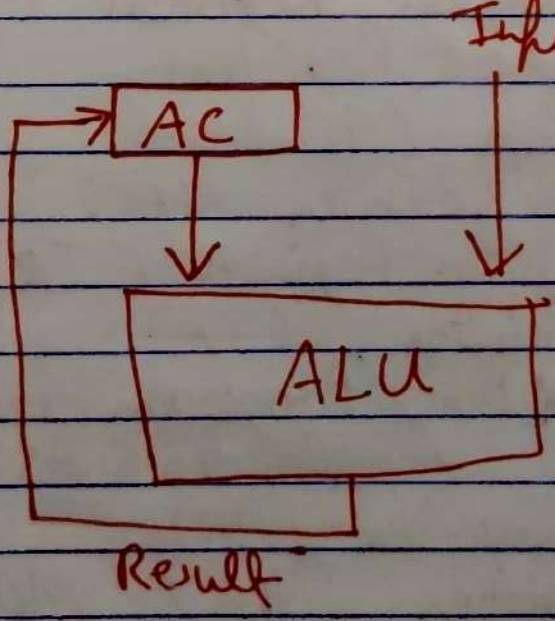
Date.
Page No.

Accumulator :

Used to stored result of ALU & sometimes it store one of the input for ALU also



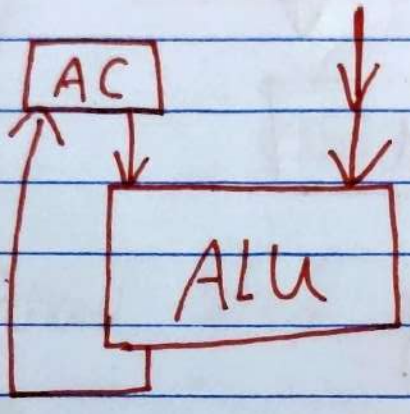
and sometimes,



called Accumulator based architechture.

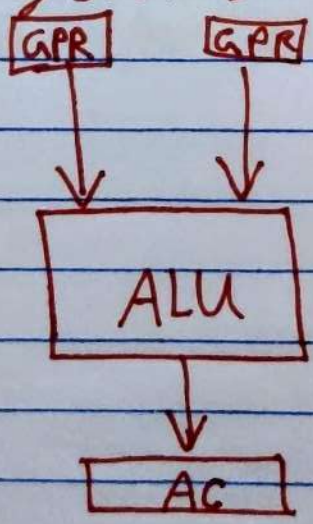
Types of architecture based on input to ALU

i) Accumalator based



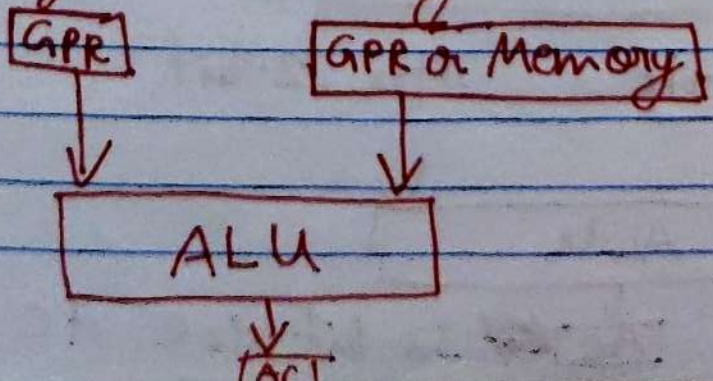
Eg,
AC ← AC + R3

ii) Register based



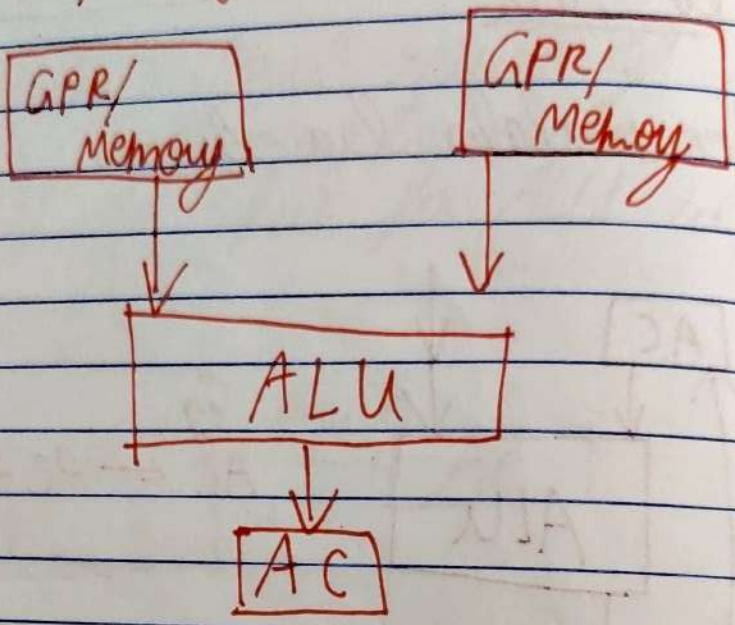
GPR ⇒ General Purpose Register.

iii) Register-memory based

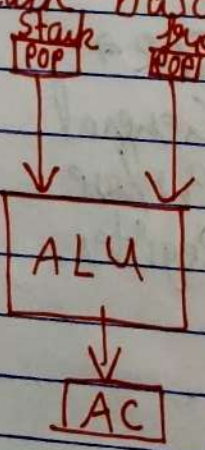


Date.
Page No.

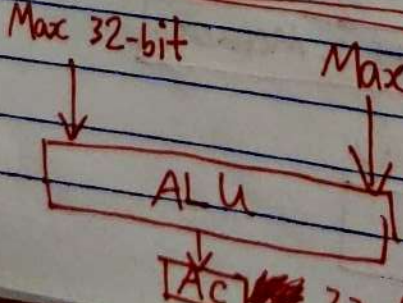
iv) Compiler-system



v) Stack-based



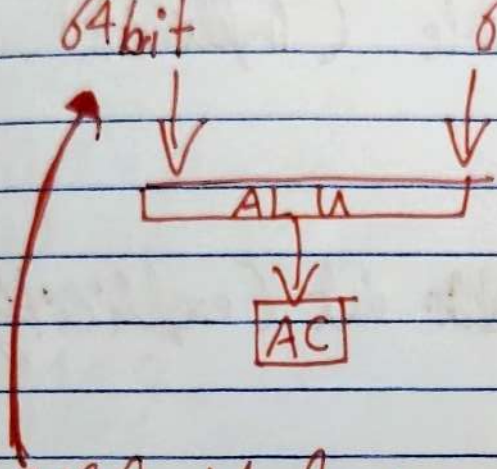
32-bit architecture



32 bit size AC

Note: Accumalators were mainly used in single address computers.
They are now considered obsolete.
ALU now write directly to GPR.

Similarly in 64 bit.



Called Word size

Program Counter

To store address of next instruction

Date.
Page No.

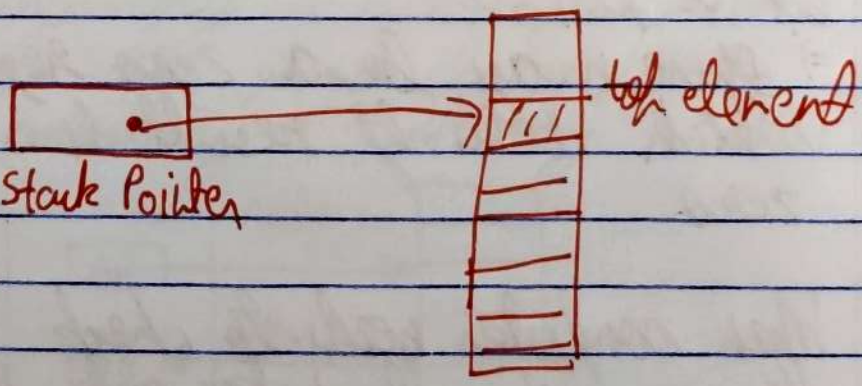
Lec 3

Instruction Register

The fetched instruction is stored in IR. It stores the current instruction which is being executed

Stack Pointer

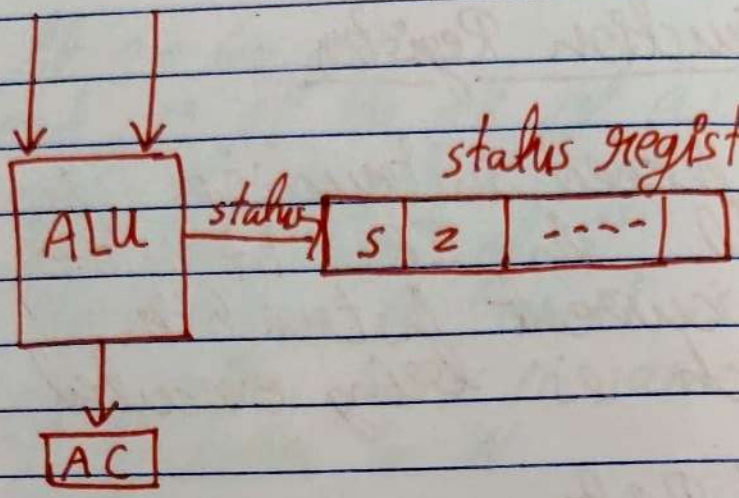
Points at the top of the stack allocated for running the program.



Flag Register / Status Register

ALU generates two outputs
i) Result of operation
ii) Status.

Date.
Page No.



It stores the status of ALU result.
For example,
⇒ there may be a sign register which tells if result is negative or positive
⇒ there may be a zero register which tells if result is zero.

When computer needs to check for conditional statements like a > b or a == b it can use status register flags to check.

Program status Word (PSW)

In some implementations it only refers to the

Date. _____
Page No. _____

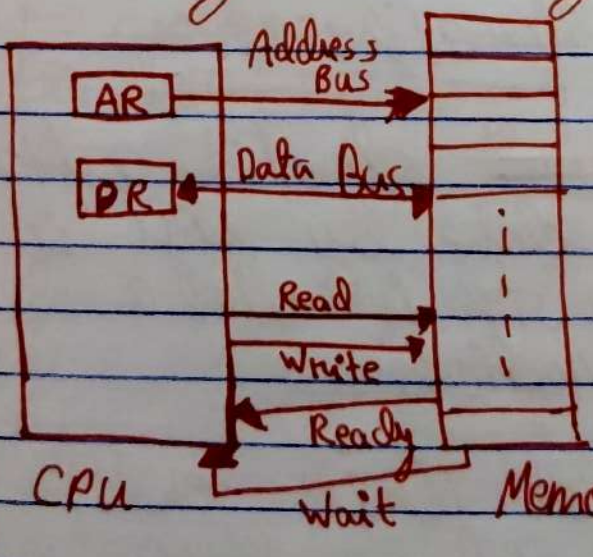
flag registers.
In others it refers to flag register + accumalator.

Address Register

Used to ~~set~~ send address to memory.

Data Register

Used to send data to memory (memory write) & to read data from memory (memory read).



Read / Write / Ready / Wait are components of control bus, connected to control unit.

Date.
Page No.

Memory Read

i) If memory CPU sends address
ii) CPU sends read signal
iii) Data is sent through data lens.

Memory Write

i) CPU sends address
ii) CPU send data through lens
iii) CPU sends write signal
iv) Memory writes the data

Micro Operation

The ~~smallest~~ operation which can be done by CPU is a single step is called a micro operation.

The Control Unit sends the command for these micro operation.

Date.
Page No.

Basically, things like read or write cycles are complex. So they are broken into small steps.
These small steps are micro operation.

Formally,
A micro operation is an elementary operation performed with data stored in registers.

Microoperation
- Register transfer
- Arithematic microoperations
- Logic microoperations
- Shift microoperation

** Operations on CPU registers are microoperations
** RTL (Register Transfer Language) is the symbolic notation of microoperations

Date.
Page No.

In RTL,

1. Register Transfer
   $R_1 \leftarrow R_2$
2. Comma, can be used to separate operations that may be executed simultaneously $R_1 \leftarrow R_2 + R_3$, $PC \leftarrow PC + 1$
3. Memory Transfer
   - Memory Read
     $R1 \leftarrow M[address]$
     Eg, $R1 \leftarrow M[2000]$
     $R2 \leftarrow M[AR]$
   - Memory Write
     $M[address] \leftarrow R1$
     Eg, $M[2100] \leftarrow R1$
     ~~$M[AR] \leftarrow R1$~~
     $M[AR] \leftarrow R1$
     ** Remember AR stores the memory addresses

So M[ ] symbolizes the data in memory

Lec 4

Date. _____
Page No. _____

Arithematic microoperations

- Addition $R_1 \leftarrow R_2 + R_3$
- Subtraction $R_1 \leftarrow R_2 - R_3$
- Complement $R_1 \leftarrow \overline{R_2}$
- 2's Complement $R_1 \leftarrow \overline{R_2} + 1$
- Addition with 2's Complement $R_1 \leftarrow R_2 + \overline{R_3} + 1$
- Increment $R_1 \leftarrow R_1 + 1$
- Decrement $R_1 \leftarrow R_1 - 1$

Note: 2's complement gives the negation, so 2's complement of 5 is -5.

Logic microoperation

- AND $R_1 \leftarrow R_2 \wedge R_3$
- OR $R_1 \leftarrow R_2 \vee R_3$
- X-OR $R_1 \leftarrow R_2 \oplus R_3$
- X-NOR $R_1 \leftarrow R_2 \odot R_3$

Date.
Page No.

Shift Operations

- Logical Shift
- Circular Shift (Rotation)
- Arithematic Shift.

Logical Shift

Shift the bits left / right & add zero at vacant position.

Eg,

1001

After Left shift 0010

After Right shift 0100

Circular shift

The bit at extreme position is not discarded, but added to other end.

Eg,

1001

After Left shift 0011

After right shift 1100

Date.
Page No.

Arithmatic shift

Only applied on signed number.
After the shift sign of the number should not change.

1001
↑
Sign bit
1 ⇒ negative
0 ⇒ Positive

Ex Lets take 1001
Right shift ⇒
1100
↑
1 bit was sign bit & hence was taken from orignal value.

Left shift ⇒
Do the logical left shift
If the sign bit changes, go back to the orignal value else we the logical left shifted value.

Date.
Page No.

Eg,
1101
Left shift 1010
↑
New value

1001
Left shift is 0010
↑
Sign bit changes
So, ~~left shift~~ arithematic left shift not allowed
An error will be generated named Arithematic left shift Overflow.

Word addressable / Byte-addressable

If memory addresses can refer to a single word, memory is word addressable.

In word addressable, a single memory cell has capacity of a single word. (word can be of any size depending on different systems) (usually its 32bit)

Date. _____
Page No. _____

In byte addressable memory, a single memory cell only stores a single byte. (A byte is 8 bits).

Date.
Page No.

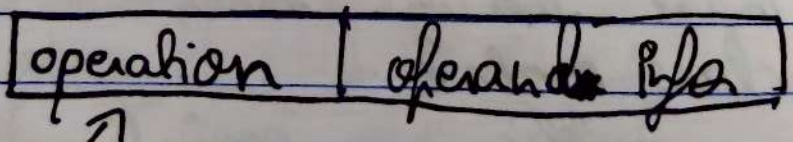
Lec 5

Digital computer takes input in binary form and gives output as binary form.

Program statements are compiled to instructions. These instructions are in binary form.

Instruction ⇒ A group of bits which instruct computer to perform operations.

General instructions have two parts,



operation code/ opcode

Example, if a CPU has 8 bit instructions and first 3 bits are opcode,

Date.
Page No.

There can be 2^3 distinct operations that are supported.

ISA (Instruction set Architecture) is collection of all instructions / operators the CPU can support.

Size of ISA = no of distinct instructions

So in our example, 2^3 is max.

Note: All 2^3 / 8 instructions may not be used in which size of ISA can be lowered.

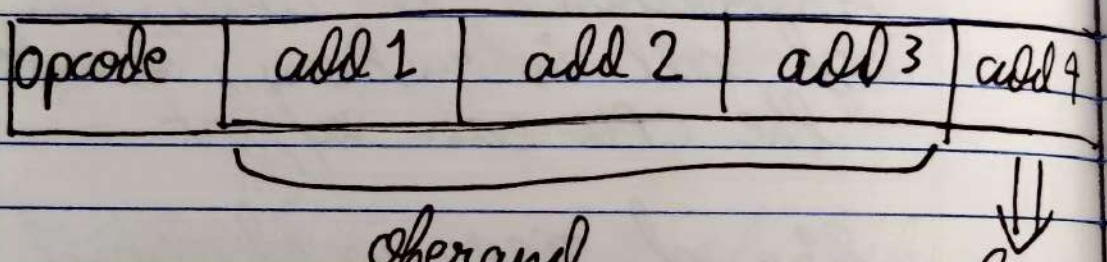
Types of operation
(Based on Operand Information)

- 4-Address Instruction
- 3-Address Instruction
- 2- " "
- 1- " "
- 0- " "

Date.
Page No.

4-address instruction

Maximum 4-addresses can be specified



4-Address instruction computers don't have Program Counter (PC).

Since relocation is hard in 4-address instructions, they are rarely used.

3-address instruction

Maximum 3-address can be specified.

Date.
Page No.

opcode	addr 1	addr 2	addr 3

2-Address Instruction

Max 2-address can be specified within an instruction.

opcode	addr 1	addr 2

Examples;

of 3 address $R_3 \leftarrow R_2 + R_1$

of 2 address $R_2 \leftarrow R_2 + R_1$

1-Address Instruction

Only a single address can be specified

opcode	address

Accumalator is used as second operand implicitly.

Date.
Page No.

Hence, 1 address instruction use accumalator based architecture. (Lec 2).

0-address Instruction

No address are mentioned, so this instruction set is only used for stack based architecture.

The operands are taken from stack.

Note: Higher address instruction CPU can support lower address instruction,

Eg, A 3-address CPU can support
2-address, 1-address and 0-address as well.

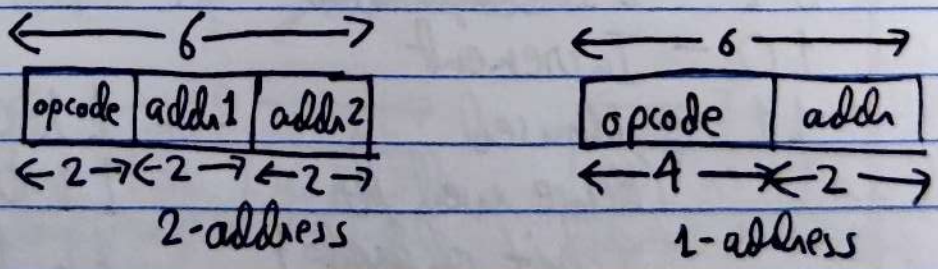
Date. _____
Page No. _____

Lec-6

Multiple Instruction Support

Consider a system which will support both 2-address and 1 address instruction

The system will have 6-bit instructions and 2-bit addresses.
How can we design a way to avoid conflict between 1-add & 2-add. instructions?



We start by looking at the instruction type with smaller opcode. (i.e the 2-address one in our example).

Suppose we want 3 2-address instruc-

Date.
Page No.

So one 2-bit opcode is unused.
This unused opcode is used to identify the ~~operation~~ instruction as a 3-address instruction.

** So there has to be atleast one ~~bits~~ unused opcode from maximum allowed.

Eg, Suppose the 2-bit opcodes in use as,

00 - Add
01 - 2's complement
10 - Increment
11 — Unused (Hence used for 4 bit opcode)

— 1100
1101
1110
1111

Available 4-bit opcodes.

If there are more supported instructions, we will always start

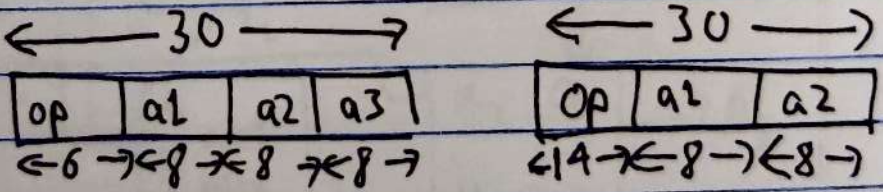
Date.
Page No.

with smallest opcode and use unused to identify higher bit opcodes.

The amount of unused opcodes is adjusted we we want more higher bit opcodes.

Q) A system has 3-address & 2-address instructions.
It has 30-bit instructions and 8-bit addresses.
There are 'x' - 3address instructions how may 2-address instructions can be formulated?

Sol)



As x 3-addr opcodes are in use out of total 2^6

$(2^6 - x)$ are unused
For each unused in 6-bit code, we have 8 bits remaining for 14 bit one.
So,

Date.
Page No.

$8(2^6 - x)$

$8(64 - x)$ is maximum 2-address instruction possible.

Date.
Page No.

Lec 7

Effective Address

→ In a computation type instruction, the effective address is the address of operand in memory.

→ In branch type instruction, the target (place to jump in program) address is effective address.

Instruction Cycle

CPU performs 6 phases to execute a single instruction. These 6 phases form a complete instruction cycle.

i) Instruction fetch
The instruction is fetched using address in PC. The instruction is copied to IR.
PC is incremented by size of instruction.

ii) Instruction decode
The opcode is decoded for

Date.
Page No.

the type of operation / instruction.

iii) Effective address calculation
Based on the type of instruction, ~~calculate~~ ~~decode~~ / read ~~data~~ & ~~to move to~~ the decode the effective address.

iv) Operand fetch
Using the decoded effective address fetch data from memory / register.

v) Execution
Execute the operation using fetched operands in ALU.
The result is stored in Accumulator.

vi) Copy / write back result
Write result from accumulator to the specified location in memory / register.

Date.
Page No.

~~More broadly,~~
More broadly, we have two different cycles, this type of categorization are named Fetch & Execution cycle.

** Sometimes a phase called "Decode instruction"

Instruction Cycle

- Fetch Cycle
  - Instruction Fetch
- Execution Cycle
  - Instruction Decode
  - Effective address calculation
  - Operand fetch
  - Execution
  - Write back result.

* Every instruction does not require all 6 phases.

Addressing Mode

The addressing modes tell CPU how to obtain operand using address field value of instruction.

Date.
Page No.

Implied mode

Opcode defination specifies operand also. So no mode is specified.

Immediate mode

The address field of instruction specifies operand value.
The data in address field is itself used as operand.

Direct mode / Absolute Mode

The address field of instruction specifies effective address, i.e the operand is in memory at the given address

Indirect mode

The address field of instruction specifies the address in memory which has the effective address

Date.
Page No.

The indirect mode is used to implement pointer.
When dereferencing we get data using indirect mode.

Date.
Page No.

Lec 8

Addressing Modes

Register Mode / Register Direct Mode

The address field of instruction specifies a register which holds the operand

Register Indirect Mode

The address field of instruction specifies a register which holds the effective address.

This mode is mainly used to shorten the instruction lenght, because register refrence bits are shorter that memory address size.

Autoincrement / Autodecrement Mode

→ Used to access table of content (array) sequentially.
→ Variant of register indirect mode.
→ Content of register is automatically incremented or decremented.

Date.
Page No.

Autoincrement ~~decrement~~ is post increment ~~decrement~~.

So the operation is done ~~then~~ & then the data in register is changed to the next effective address in sequence.

On the other hand, Autodecrement is predecrement.

Index Register Mode

This mode is used ~~in~~ to access a particular element of an array.

The address field on the instruction contains the base address.
The index value is stored in the index register (hence, name of the mode).

The base address and index value address are added to get effective address to operand.

Date.
Page No.

$A[i] = B + i \times w$

w is size of a single element
$i \times w$ is index value
Base address

This mode hence mostly uses two instructions.

1. Index register $\leftarrow (w) * i$
2. Instruction with this addr. mode.

Implementation

Index Reg. is special purpose

opcode	mode	addr.

Index Reg. is not special

opcode	mode	Index Reg. Identifier	address

If program is relocated in memory, the base address needs to be changed in instruction
* As updating instructions is time costly, the efficiency may be lost.

Date.
Page No.

Index, Base Register Mode

Rather than storing base address in instruction's address field, it is stored in a register called Base register.

Effective addr. = Index reg value + Base Reg Value

If index reg & base reg are not special purpose then instruction of type

opcode	mode	Index Reg Refrence	Base Reg Refrence	addr. (optional)

Lec-9

Date.
Page No.

Index, Base Register + Displacement Mode

Suppose we have a struct in memory (or a record). We made an array of that struct

Using Base + index we can get the ~~first~~ struct.
But ~~if~~ we want to access a member of struct, we will have to use displacement to that member in struct.

In the address field of instruction the displacement is stored.
Base / Index value are stored in their registers.

Effective address = Base Reg Addr. Value + Index Reg Value + (Displacement)

Scaled Mode

This mode is also used to access an element from array.

Date.
Page No.

The address field has the base value and the index number is in register.

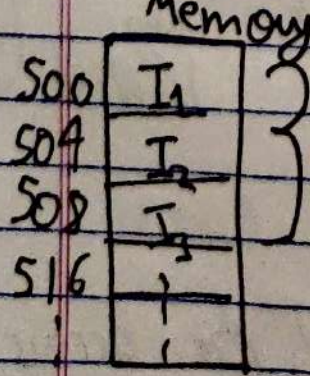
Effective address = Address field value of instruction + index Reg Value × scaling factor

scaling factor → taken implicitly

This mode is more efficient than base register mode.

PC-Relative Mode

This mode is used for branch instruction.
Assume memory as.



Date.
Page No.

Assume CPU is executing instruction I2, so PC = 508.

While decoding CPU detects I2 is Branch instruction, (conditional)

If conditional

True

- Branch taken
- Target instruction is executed (let's assume I6)
- PC is updated, in our example PC = PC + I2
  PC = 520

False

- Branch not taken
- Next instruction in sequence is executed
- No change in PC value.

Opcode	Mode	address

In ~~effected address~~ PC-relative mode, the value is updated as,

PC = PC + addr. field value of instruction

Date.
Page No.

Hence, address field has relative location to skip (this value is sometimes called offset).

No. of instructions skipped = address field value / size of instruct

PC-Relative Mode is also called Position Independent mode.

* This mode is used for Intrasegment jump.

Base Register Mode

This mode can be used for inter segment jumping. (Inter segment means in another segment).

If branch taken, then PC is updated as

PC = Base register value + offset.

Date.
Page No.

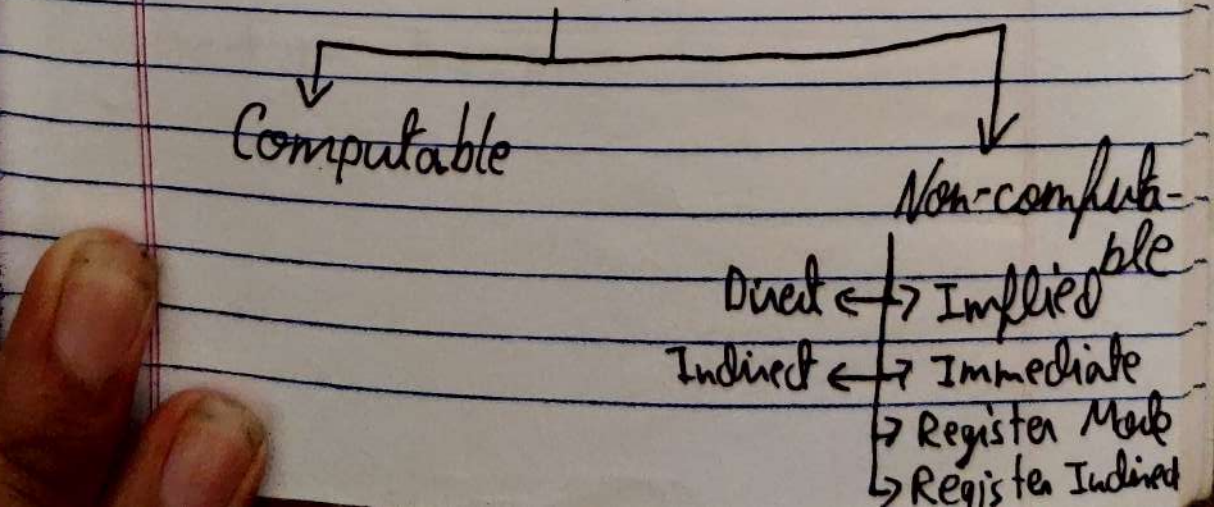
Here, base register will store value to the start of the target segment.

Offset is in address field of instruction.

If segment is relocated only base register is updated, updating instruction is not required.

Note: Base register ~~can also be~~ mode can also be used to store effective address to operand, if opcode is not branching type.

Classification of Mode



Date.
Page No.

Computable

- Auto inc. / Autodecrement
- Indexed Mode
- Index + Base Reg. Mode
- Index + Base Reg + Displacement
- Scaled
- PC-relative / Position independent
- Base Register

* PC-relative & Base register can be used for branching hence sometimes called control modes.

* Remaining all used for data access for operations.

Lec 10 is questions on addressing modes.

classmate
Date
Page

Lec 11

CPU cycle

Time in which CPU performs one micro-operation

$$\text{Clock rate} = \frac{1}{\text{Clock cycle}}$$

CPI (Cycles Per Instruction)

No. of cycles required to execute an instruction.

Execution Time

For 1 instruction the time taken is $CPI_{avg} * 1$ cycle time

So execution time of n instructions is calculated as

$$= n * CPI_{avg} * (1 \text{ cycle time})$$

Suppose a program has k different types of instructions

Then avege CPI is

$$CPI_{avg} = \frac{\sum_{i=1}^{k} n_i \times CPI_i}{\sum_{i=1}^{k} n_i}$$

n_i = no. of instruction of i.
CPI_i = CPI for instruction type i.

Million Instructions per second (MIPS)

It is a way to measure CPU performance.

Suppose it takes t seconds to execute n instructions.

~~no of instructions in 1 second = $\frac{n}{t}$~~

~~Time taken on 1 instruction~~

Instructions per second = $\frac{n}{t}$.

Million Instructions per second = $\frac{n}{t \times 10^6}$.

$$MIPS = \frac{n}{\text{Execution time} \times 10^6}$$

Using formulae of Execution time

$$MIPS = \frac{n}{n \times CPI_{avg} \times (1 \text{ Cycle Time}) \times 10^6}$$

$$MIPS = \frac{1}{CPI_{avg} \times (1 \text{ Cycle Time}) \times 10^6}$$

MIPS varies based on number and complexity of instructions.

So we use FLOPS (Floating Point Operations per second) as a standard metric.

Arithematic and Logic Unit