Instruction Set

Computer Organization

→ Information representation
→ Arithmetic op.
→ Instruction format.

1) Decimal → Binary.
2) Signed Number representation.

$+5 = 0101 \qquad -5 = 1101$

3) 1's Complement.
4) 2's Complement : 1's complement + 1
5) Addition
6) +0 and -0 have different value.
7) 2's complement Conversion to Decimal -ve & +ve
8) Addition of 2's complement Number

range: -2^{n-1} to $2^{n-1} - 1$

9) Subtraction of 2's complement Numbers
10) Overflow in int Arithmetic

$$-2^{n-1} \leq V \leq 2^{n-1} - 1$$

↓ occur when two numbers have same sign and diffent from sum.

Subtraction
2's complement

2's complement of Y + X.

11) Advantage's, Disadvantage. of each

→ Char Representation

1) BCD Binary coded Decimal
2) Hexadecimal Number + Advanges.

→ Memory (store 1 bit number @ cell).

space of another

techniques to achieve

Context switch

- One leaves the Running state

Concurrent Process

→ Two process are concurrent if their execution overlap in time.

→ In a uniprocesor environ, multiprogramming provids concurrency.

→ In a multiprocessor, true parallel execution can occur.

Forms of concurrency

1) Multiprogramming
2) Time sharing : An extension of multiprogramming
3) Multiprocessing

1) Creates logical parallelism by running several processes / threads in a time. The OS keeps several jobs in memory simultaneously. It selects a job from the ready state and starts executing it.

Protection

When multiple processes exist at the same time, and execute concurrently, the OS must protect them from mutual interference.

Processes and threads

→ Traditional processes could only do one thing at a time - they are single-threaded.
→ A thread is an "execution context" or "separately schedulable" entity.
→ Multithreaded processes can do several things at once - they have multiple threads.

Threads

→ several threads can share the address space of a single process, along the resources - such as files.
→ Each thread has its own stack, PC, and TCB (thread control block)
  i) Each thread executes a separate section of the code and has private

Process (Thread) scheduling

Scheduling algorithms:

1) FCFS (First-come, first serve): non-preemptive
2) RR (round robin): preemptive FCFS, based on time slice
   - Time slice = length of time a process can run before being preempted
   - Return to ready state when preempted

Scheduling Goals

→ Optimize turnaround time and/or response time
→ Optimize throughput
→ Avoid starvation (be "fair")

→ Respect priorities
- static
- Dynamic

Interprocess communication (IPC)

→ Processes (or threads) that cooperate to solve problems must exchange information.
→ Two approaches:
1) Shared memory
2) Massage passing (copying information from one process address space to another).
→ Shared memory is more efficient (no copying), but isn't always possible.

Process / thread Synchronization.

→ Concurrent processes are asynchronous: the relative order of events within the two processes cannot ....

Instruction streams

Process A : $A_1, A_2, A_3 \ldots A_m$
Process B : $B_1, B_2, B_3 \ldots B_n$

Sequential :

Semaphore.

→ Process Synchronization - 2 Types.
→ Mutual Exclusion
~~A critical s~~
→ Synchronization.
→ semaphore.
→ Deadlock
→ Deadlock Vs Starvation
→ Causes of Deadlock.
- Mutual exclusion (exclusive access)
- Wait while hold (hold and wait)
- No preemption
- Circular wait

The Structure of compiler

Compiler

Analysis ↙ ↘ Synthesis

— X —

1) Encoded Information
2) Memory
3) Memory Units
4) Addresses Assignment
5) Machine instructions.
6) Data Transfers : Possible Location.
7) " " : Register Transfer Notation
8) " " : Assembly Language Notation
9) MOV instruction
10) Basic instruction type
11) 3 Address Instruction
Advantage : less number of instruction
12) Two~~One~~ Address instruction
13) One - Address Instruction.
14) Processor Registers.
15) Addressing Modes.
16) Program sequencing and control.
17) Specific Machine levels

Why OS?

1) Act as interface b/w software and hardware
2) OS is resource manager

(8) Translating Languages
(9) Assembly Language

1)
2) ═══※═══

3) 3- addresses
4) 2- addresses.
5) 1- addresses
6) 0- addresses
7) Addressing modes
8) Implied addressing
9) Immediate addressing
10) Direct "
11) Indirect "
12) Register "

Register Indirect address.
Pros: large address space
cons: extra memory reference.

Displacement addressing.
Pros: flexible
cons: complex
Stack addressing:
Pros: short and fast fetch
cons: limited by FILO order.

13) index addressing
14) Based Indexed Addressing
15) The instruction cycle: Intermediate

18) Translating Languages
19) Assembly Language

1)
2)

3) 3- addresses
4) 2- addresses.
5) 1- addresses
6) 0- addresses
7) Addressing modes
8) Implied addressing
9) Immediate addressing
10) Direct "
11) Indirect "
12) Register "

Register Indiret adress.
Pros: large adress space
cons: extra memory reference.

Displacement addressing.
Pros: flexible
cons: comple-
Stack addressing:
Pros: short and fast fetch
cons: limited by FILO order.

13) index addressing
14) Based Indexed Addressing
15) The instruction cycle: Interrecte ve

Control Design

1) Data Processing Unit (DPU)
2) Control Unit (CU): issues control signals as instructions to DPU.

*DPU is logically reconfigured by the CU to perform certain set of micro-signals.

Functions of CU

1) Instruction Sequencing: Methods by which instructions are selected for execution or control of the processor is transferred from one instruction to another.
2) Instruction Interaption:

Instruction sequencing

- Each instruction can explicity specify the address of its successor.
- It may increase the instruction length, which in turn increases the cost of memory where it is stored.
- I1 is stored in location A, and I2 is its successor.
- Branch instruction: Specifies implicitly/explicitly an instruction address X.
- Unconditional branch always alters the flow of control by causing PC ← X

Program control transfer

Often it is required to transfer of control from P1 to P2 due to subroutine call or interrupt.

Control stack

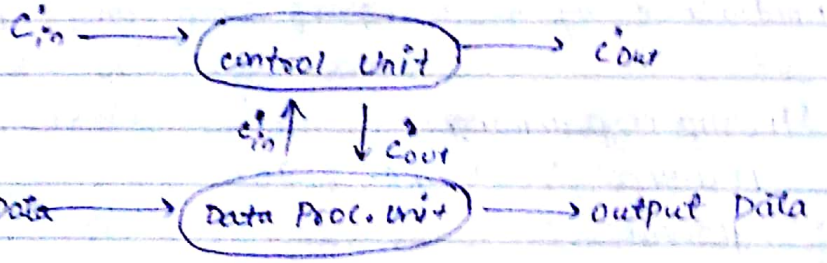
- Stack is used to store return address.
- Also used to store pass variables and variables that are local to P2.

Stack implementation

- Smaller stack : shift register, larger stack : RAM
- Push : memory write, and pop : memory read
- Stack Pointer (SP) : this register contain address of the memory location currently acting as top-of-the-stack.
- Limit register (L) : Contains highest stack address
- Base register (B) : contains lowest stack address
- SP > L (SP < B) : overflow (underflow)

Instruction interpretation

control → signals → via control lines → outside.



- Control and data are relative, rather absolute
- Same physical lines, either data or control at diff. levels.

C'_{out} : Control the operation of D.P. unit (main function of C.U)

C'_{in} : enables the data being processed to influence the CU, allowing data-dependent decision to be made. E.g : errors, overflow.

C_{out} : transmitted to other CU, indication the status such as busy or operation completed.

C_{in} : received from the other CU. e.g supervisory controller - start / stop / timing information.

C_{in} & C_{out} : Synchronize with other CU.

Behaviour of CU :-
Flowchart / description language or both

Implementation methods :

1) Hardwired C.U : sequential logical circuit, which generates specific sequences of control signals.
   - minimizes the no. of ~~compete~~ components
   - maximizes the speed of operation.

   Once constructed, changes in behaviour can be implemened by re-designing and physically ...

2) Microprogramming :
   - Maurice V. Wilkes (1950)
   - flexible & systematic
   - A (micro) instruction stored in a special addressable memory called a control Memory (CM)

Microprogram : sequence of micro instructions (MI) needed to execute a particular operation.
- fetch the MI from C.M. one at a time, and use them to activate the control lines directly.
- Micro-programmed control unit.
- firmware.

- design changes can easily be made by altering the contents of the CM.

Emulation : A micro-programmed CPU can execute programs written in the M/C language of several different computers.

Limitation

(x) Costly than HCU due to C.M. & its access circuitry. slower due to fetch from C.M.

Hardwired control

Trade-Off : Amount of h/w, speed and cost. Design methods in practice are ad hoc, heuristic, cannot easily be formalized (due to large no. of instruction, and interdependency.

Three main methods (suitable for small C.U like RISC)

1) State-table method
2) Delay-Element Method
3) Sequence counter method

} No one can win over other.

1) State-table Method

- like any finite-state sequential m/c

Cin : Input
Cout : output
Row : set of internal states $\{S_i\}$
Internal State : Information stored at discrete point of time
Column : set of external signals to C.U.
Entry in row S_i and column I_j : $S_{i,j}$ / $Z_{i,j}$
$S_{i,j}$ = next state of C.U
$Z_{i,j}$ = set of output signals $Z_{i,j}$ from Cout that are activated by the application of I_j to C.U. when it is in state S_i.

Pros : systematic technique for minimizing no. of gates, flip-flops.

Cons :-

2) Delay element Method

Consider the problem of generating the following sequence of computer signals at time $t_1, t_2, \ldots, t_n$ using a hardwired C.U.

t_1 : Activate $\{C_1, j\}$;
t_2 : Activate $\{C_2, j\}$;
⋮ ⋮ ⋮
t_n = Activate $\{C_n, j\}$;

Cons ~~and Pros~~

3) Sequence counter Method.

—x—

Relationship ~~Module~~ : Seq. counter and Delay element

Microprogrammed Control

→ Every instruction in a CPU is implemented by a sequence of one / more set of concurrent micro operations.
→ Each micro operation is associated
→ Information stored in ROM / RAM, called control memory (CM)
→ Each microinstruction specifies the next

→ A micro programmed computer C1 used to execute programs written in m/c language L2 (other m/c C1) by placing an emulator for L2 in the C.M of C_1
→ C_1 is then said to be capable of emulating C_2.
→ Comparable with assembly language, but requires more detail knowledge of the processor h/w.
→ Symbolic language → micro assembly language
→ Micro assembler.

* Wilke's Design

Microinstruction :

1. Control field : indicates the control lines to be activated,
2. Address field : indicates the address in the CM of the next microinstructions to be executed

K_i, C_i : If $K_i = 1$

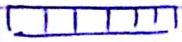
CM : ROW → microinstruction
col → Control / Address lines
CMAR → control memory address register which stores the address of the current instruction

IR : instruction register

- Microinstruction format IBM sys./370 Model 145
- Parallelism in Microinstruction.
  Parity bits : error can be checked
- 21 different control fields: according to purpose.
- Un-encoded format :
- microinstruction Register (μIR)

Horizontal Microinstruction

1) Long formats
2) Ability to express a high degree of parallism
3) Little encoding of the control information



n.

1) Short formats
2) Limited ability to express parallel micro operations
3) Considerable encoding of the control information

$\log(n+1)$

Microinstruction Addressing :

1) Each microinstruction contains the CM address of the next microinstruction to be executed
2) For branching, two possible next address are ~~not~~ included.

Advantage : No time lost in address generation
Limitation : Wasteful of CM space.
Solution : microprogram counter (μPC) for branching.
Analogous to PC at instruction Level

Conditional Branching:

→ The condition to be tested is condition variable or flag generated by the DPU (Data Processing Unit).

→ If several such conditions exist, a condition select subfield is included in the microinstruction.

→ The branch address may be contained in the microinstruction itself, in which case it is loaded into CMAR, when a branch condition is satisfied (to save space CM can load only low-order bits)

Another approach:

Skip on overflow

Micro operation Timing:

→ Each MI (Microinstruction) generates a series of control signal. CS which are active for duration of MI's execution cycle.

→ A single clock pulse synchronizes all control signals. This is called Monophase

→ Polyphase

→ Each control signal is active during one of the phases.

→ Added complexity

Example: Register Transfer operation

Micro operation Timing.

→ Fetch & execute steps can be overlapped.

Residual Control

Storing the cloud field in a register which continues to exercise control unit it is modified by a subsequent MI.

→ Application of Residual control

Control Unit Organization

- A condition select field is used to specify the external condition to be tested in case of conditional branch MI.
- Rest specifies (un/encoded) format of the CS that must be activated to perform the desired micro operation.
- The contents of the addressed word in CM are transferred to the μIR.
- Control fields are decoded (if required) and used to generate CS for DPU, μPC is then incremented.
- For branch instruction in μIC, the contents of the MI address field are loaded in μPC
- MUX activates parallel load control.

S_0	S_1	Meaning
0	0	No branching
0	1	Branch if $V_1 = 1$
1	0	Branch if $V_2 = 1$
1	1	Unconditional branch

NaNoprogrammed Computer

- In some machines, the micro instructions do not directly issue the signals that control the hardware.
- Instead the are used to access 2nd control memory, called nanoControl Memory (nCM), that directly control the hardware

There are 2 level of control memories:
1) High level : Microcontrol memory μCM (MI)
2) lower level : Nano control memory nCM (nano instructions)

Scanned by CamScanner

Advantages

- It can reduce the total size of CMs needed, this translates to smaller chip area.

Disadvantage.

Reduction in speed due to extra memory access for nCM and a more complex CU organization.

One level $S_1 = Hm(N + [\log_2 Hm])$ — (1).

In Two level design, μCM again stores Hm MIs, but N-bit control fields are transferred to nCM.

- In place of latter, each MI in μCM contains $[\log_2 Hn]$ bit address to specify any NI location in nCM.

Hence size becomes,

$S_1 = \{ Hm \cdot (N + [\log_2 Hm]) \}$

$= 650 \left(70 + [\log_2 650] \right)$

$= 52,000$

$S_2 = H_m \left(2 [\log_2 H_m] + [\log_2 r] + \gamma N \right)$

$= 650 \left(2 [\log_2 650] + [\log_2 0.4] + 0.4 \times 70 \right)$

$= 650 \left(2 [10] + [-1] + 28 \right)$

$= 650 \left(20 + 28 - 1 \right)$

$= 650 (47)$

$= 30550$.

Processor Design -

CPU -> executes sequence of instructions stored in a memory which is external to CPU.
- The seq. of operations constitute instruction cycle.
- It has two parts.

i) Fetch cycle :
ii) instruction cycles
- sequence of micro operations, involves a register transfer operation.

CPU cycle time : (tcpu) time required for shortest well-defined CPU micro operation.

clock rate :

CPU designs have following two premises :
- It should be as fast as technology permits; number of components in the CPU must be kept relatively small.
- Main memory (MM) of relatively large capacity is needed to store programs and data.

Memory cycle time (tm) : time elapses between two successive read or write operations.

tm / tcpu : 1-10

- All I/O data transfers are implemented by memory referencing instructions.
- Memory locations and I/O ports share the same address space.

I/O Mapped or port-addressed I/O :-
- have I/O instruction that are distinct from memory-referencing instructions.

Accumulator : CPU register, plays a central role, being used to store an input or output operand in the execution of most instructions.

→ A simple Accumulator based CPU. (Fig.)
→ AR : Memory Address Register : Operation of the CPU.
  i) Fetch cycle
  ii) Execution cycles

3. Special register can be included to facilitate the transfer of control between instructions within a program.
e.g.: status register is used to indicate conditions resulting from the execution of the previous program.

4. Transfer of control b/w the diff. subprogram due to interrupts or subroutine calls and return is also facilitated by special registers
- control is transferred by saving the current PSW (Program status word) in a designed location in MM and loading a new PSW into the CPU
- uses stack
  - Advantage of stack: handles repeated program transfers

5. Facilities can be provided for the simultaneous processing of two or more distinct instructions.
- can be fetched simultaneously by extending the memory addressing circuit & adding sufficient buffer storage to CPU.

→ Typical CPU with general register operation (Fig)

- ALU stores/obtains most of its results/operands into/from this register set
- Special logic is

→ Instruction sets

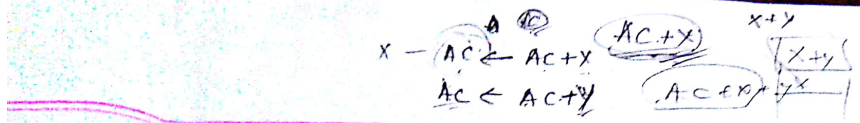
~~The of~~

→ Addressing Modes
- Operand associates with data x

Orthogonality:
The ability to use all relevant addressing modes in a uniform and consistent way all opcodes of an instruction set.
- Simplifies programming both by reducing the number of opcodes needed by simplifying the rules for operand address specification.
- Most computer have little orthogonality.

Absolute Addressing: Simplest mode of (direct) address formation:
- Requires complex operand address to appear in the instruction operand field
- ~~Neo~~-Address is used without further modification to access the desired data item.

Relative Addressing:
- Partial addressing information is included in the instruction.
- Complete address must be constructed by CPU.
- Instruction also identifies storage locations $R_1, R_2, \ldots$
- Effective address A is $f(D, R_1, R_2, \ldots, R_k)$



- R acts as a base register (content is called the address)

3. Facilities the process of indexed data.
   - R is called the INDEX REGISTER.
   - Indexed terms X(0), X(1), ..., X(n) are stored in the consecutive addresses of Main Memory.

→ Auto-Increment and Auto decrement.

Limitation:

- Extra logic ckt and extra processing time required for address computation.
- Indexed items are frequently accessed sequentially.

Number of ADDRESSES:

- Fewer the address, the shorter the instruction.
- Fewer addresses means more primitive inst. and longer program required to perform any given task.

Polish Notation (Polish logician Jan Lukasiewicz)

Postfix : $x * y \Rightarrow xy*$
Prefix : $x * y \Rightarrow *xy$

Advantage : No requirement of parentheses.

$A(B+C) \rightarrow ABC+*$

OPCODE

→ A fixed length field of k bits within each instruction permitting up to 2^k distinct operations to be specified
→ Number of registers are small, so only a few bits are required.

1. Destination Register
2. Source Register address

Example;

ADD R,
i.e A ← A+R, R is any 8 bit register

ADI D, D is a 8 bit word, 2 bytes long

JMP address,
PC ← Address, 3 byte branch instructio

→ Some programs occupy a

Hazard of calling function ??

→ Assign short format to the most frequently used instruction, and the longest format to the least frequently used.
→ The frequency with which specific instruction types occur can be determined.
→ One can therefore attempt to base opcode lengths on these.

Instruction Types :
→ What type of instructions should be included in a general purpose processor's instruction set?
→ A typical machine instruction defines one or two register transfer (micro) operations, and a sequence of such instructions is needed to implement a statement in a high level programming language.
→ Due to complexity of operation, data type and syntax HLPL few successful attempts have been made to construct computers whose machine language directly coresponds to a H.L.P.L (High level Prog. Lang.)
→ Thus there is a semantic gap b/w HLPL and (M.I) (Machine inst.) that implements it, a gap that compiler must bridge.

Requirements :
1) Complete : One should be able to construct a machine langugr program to evaluate any func. that is computable using a reasonable amount of memory space.
2) Efficiency :- Frequently required functions can be performed rapidly using relatively few instructions.
3) Regular :- Instruction set should contain expected opmodes and addressing modes (e.g. left shift or right shift). The instruction set should

also be reasonably orthogonal with respect to the addressing modes.

4) Compatible: To reduce both h/w and s/w design costs, the instructions may be required to be compatible with those of existing machines.

Completeness:

A function f(x) is defined to be computable if it can be evaluated in a finite number of steps by a Turing machine.

But real computer have finite amount of memory, and can be used to any computable function, at least to a reasonable degree of approximation.

- Turing machine has only four basic operations
- while simple instruction sets require simple, therefore inexpensive, logic circuits to implement them, they can lead to excessively complex programs.

Instructions are divided into five major types:

1) Logical Instruction: Include Boolean and other non-numerical operation.
2) Program Control Instructions: Such as branch instructions, which change the sequence in which program executes.
3) Arithmetic Instruction
4) Data tranfer Instruction
5) Input-Output (IO) Instruction

These are not always mutually exclusive.

Pausing. ?!

RISC Vs CISC

- No general agreement about what constitutes the appropriate size of a general purpose instruction set.
- Early computers had small & simple instruction sets, forced by the need to minimize the amount of h/w.
- As h/w became cheaper, instruction set tends to increase both in number and complexity.
- Reduce the semantic gap.

→ multi.

I_F or P_F ?? (slide)

one complex instruction

→ Execution of P_F will generally be slower than that of I_F due to the fact that more time must be spent fetching instruction of P_F.

→ It occupies more memory space than I_F

→ But the disadvantage of I_F is that is adds complexity to the processor.

(slide)

(slide)

Reduced Instructions Set Computer (RISC)

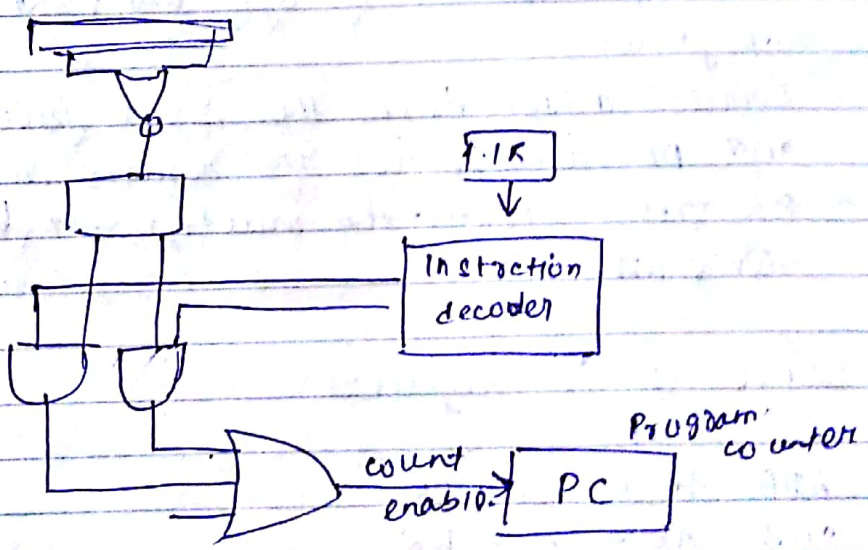
Contains small and relatively simple instruction sets.

1) Few instruction type & addressing modes
2) Fixed and easily decoded instruction
3) Fast - single cycled
4) Hardwired rather than micro programmed control.
5) Memory access limited mainly to load & store options.
6) Use of compilers to optimize object code performance.

Implementation (Design processor).

The design of circuits to implement an instruction set is an exercise in register-level logic design.
- Necessary to devise an algorithm for appropriate register transfers or other micro operations.
- Also should balance ckt cost & execution speed
- Common to choose algo that permit ckts used by instructions to be shared.

* Implementation of two conditional branch instructions
SZA (skip on zero accumulator)
SNA (skip on non-zero accumulator).



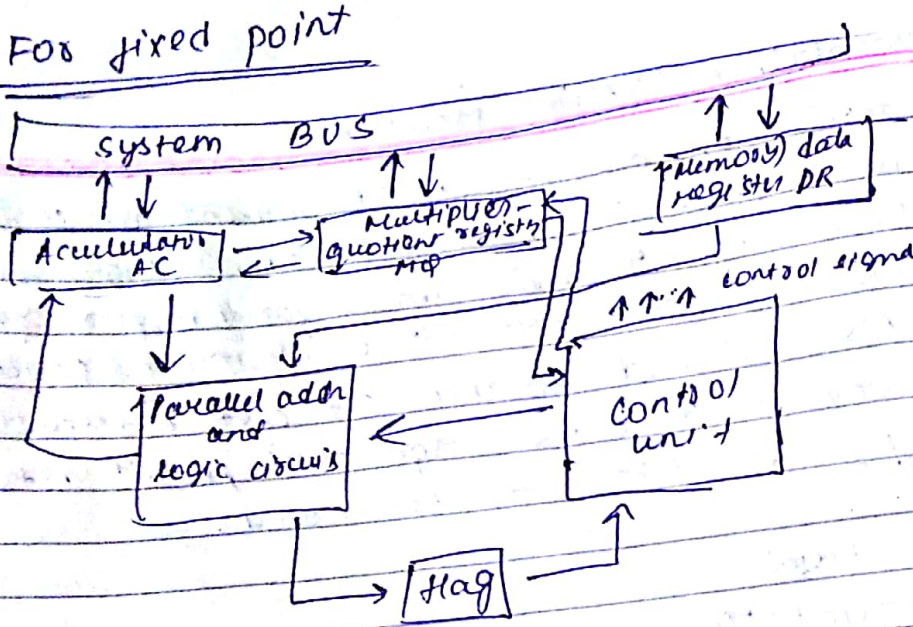
ALU design

Various ckt used in the execution of data processing instructions by a processor are usually merged into a single unit called ALU.
- fixed-point / floating point or both
- can have auxiliary unit called arithmetic co-processor

Structure of a basic fixed point ALU.

Flags = Underflow or overflow activated when

For fixed point



→ Multiplication and Division are done by sequential digit-by-digit shift-and-add/subtract algo.
→ Three one-word registers: AC, MQ and DR
→ MC. MQ : cabable of left and right shiftings
- Parallel adder drives the input from AC and DR and places the results in AC
- DR also stores the multiplicant for derision.

Usage of the registers:

ADD : $AC \leftarrow AC + DR$
SUB : $AC \leftarrow AC - DR$
AND : $AC \leftarrow AC \wedge DR$
X-OR : $AC \leftarrow AC \oplus DR$
MULT : $AC.MQ \leftarrow DR \times MQ$
Div : $A.CMQ \leftarrow MQ \div DR$
OR : $AC \leftarrow AC \vee DR$
NOT : $AC \leftarrow \overline{AC}$

→ Sometimes DR serves as a memory buffer register to store data, addressed by an instruction address field ADR.

Then DR can be replaced by M(ADR), results in a one-address-memory-referencing format.

Bit-Sliced ALU : (4 bit to 16 bit)

- Feasible to design an entire fixed-points ALU on a single IC chip. (4/8 bits)
- can be designed to be expandable in that k copies of the ALU chip can be connected to form a single ALU of k m-bit words directly
- Resulting array-like ckt is called bit-sliced, becoz each component chip processed an independent 'slice' of m-bits for each k m-bit operand.

Advantage.

Any desired word size can be handled efficiently by selecting the appropriate number of components (bit slice) to use.

* 16 bit bit-sliced ALU composed of 4-4-bit slices

- ~~Data buses~~

Floating Point Arithmetic

$X(X_M, X_E)$ i.e $X = X_M * B^{X_E}$

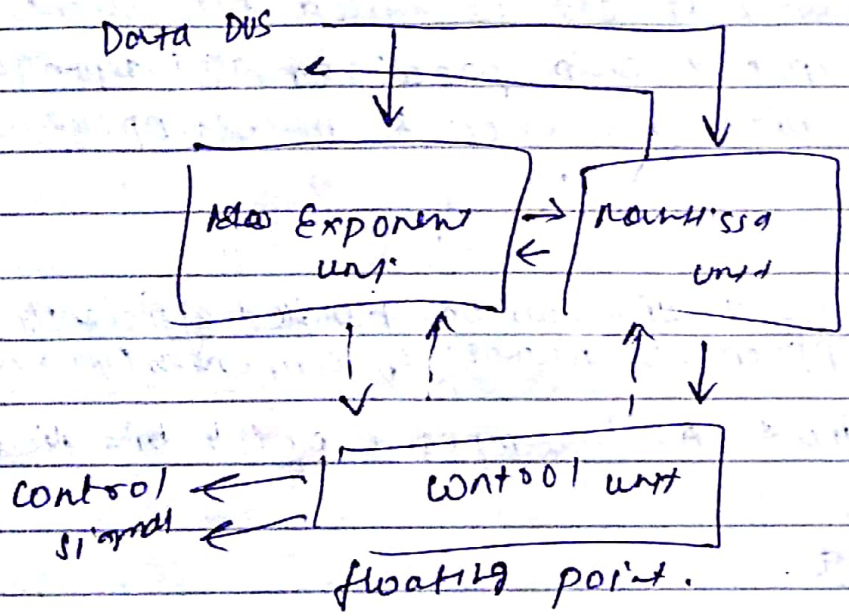
~~$X_M = n_M$ bit~~

$$X * Y = (X_M \times Y_M) \times 2^{X_E + Y_E}$$
$$X / Y = (X_M / Y_M) \times 2^{X_E - Y_E}$$
$$X + Y = (X_M 2^{X_E - Y_E} + Y_M) \times 2^{Y_E}$$
$$X - Y = (X_M 2^{X_E - Y_E} - Y_M) \times 2^{Y_E}$$

Lin - magnitude
2's comp'
1's comp'

→ * and / are relatively simple, since the mantissas and exponents can be processed independently.

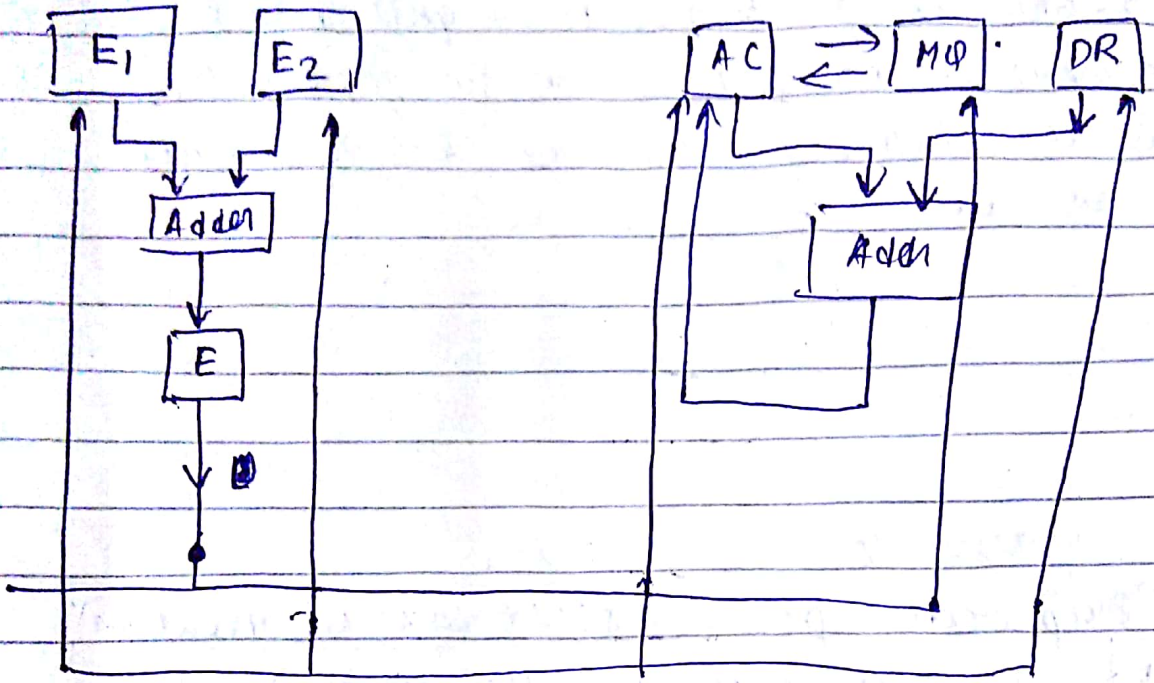
Floati:



steps:

- Compute $(Y_E - X_E)$
- Shift X_M $(Y_E - X_E)$ places to form $X_M 2^{X_E - Y_E}$
- Compute $(X_M 2^{X_E - Y_E} \pm Y_M)$
  (a fixed point + or −)

* Data processing part of a simple floating point arithmetic unit



Drawback: It can be used for n-bit floating-point operations only, even though it has most of the facilities required for n-bit fixed-point operations.

Since all computers with floating-point instruction also have fixed-point instructions, it is often desirable to design a single unit for both types of instructio (a bit difficult ckt).

Arithmetic Processor

- CPU devotes a considerable amount of control and data processing h/w for non-arithmetic functions.
- Such CPU must rely on slower s/w routines
- ∴, h/w costs due to IC count & chip area for implementing complex arithmetic operations often prevent their inclusion in the CPU's instruction set.

- Alternatively, a processor is devoted exclusively to arithmetic functions, so a full range of numerical operations can be implemented in h/w at relatively low cost e.g in a single
- It is an aux

Two ways to implement it.

1. Peripheral processor: It can be treated as a peripheral or I/O device to which the CPU sends data and instructions, & from which it receives results.
2. Co-processor: It is closely coupled to the CPU so that its instructions and register set are extensions to the those of the CPU. The CPU's instruction set has a special subset of opcodes reserved for the auxiliary processors.