INTRODUCTION TO PROCESSORS

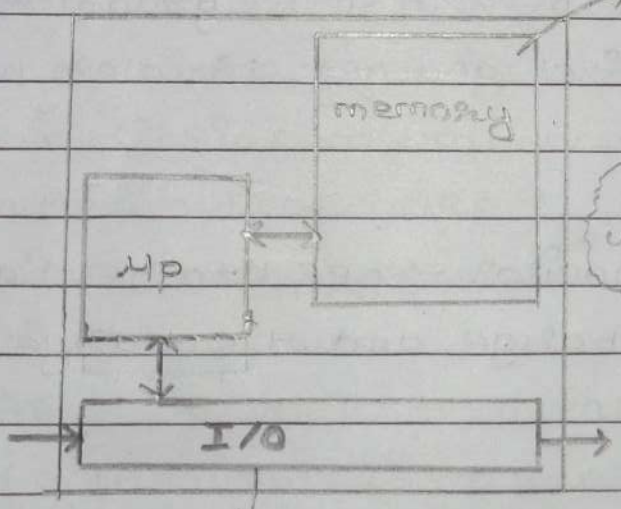
8085

Page No.
Date: / /

* Microprocessor are needed where there is a program to be executed.

* First commercially successful microprocessor was 8085 before it the closest one was 8020 but due to unstable voltage it had to be changed to 8085 with 5 volt stable voltage.
* Then came 8086 — included in IBM PC.
* 80586 first house hold computer use a/c Pentium.1.

* 8085 are not used in modern day computers.
  - Intel donot make them now, it is out lipscenced
  - Microcontrollers have the same grade (not exactly 8085) are used in places like traffic light, remote control etc.



stores either data or program

→ MS-word (program)

files of MS-word (data)

no secondary memory were developed during the time of 8085

takes input displays output

no processing done

fig: computer system.

Memory here means primary memory and more specifically RAM.

RAM

static RAM	dynamic RAM
-cache is SRAM not relevent in 8085 as it was not developed.	slower than SRAM
- stores data in flipflop	stores data in capacitor (refreshing required) charged - 1 discharged - 0
- set-1, reset-0	

Synchronous Dynamic RAM (type of DRAM)

Data (for an example taken Image) is stored in memory in form of 0 and 1, which stores the data for each pixel. more the pixel, more the data (more amount of processing required)

Each pixel has a color stored. If 1-bit for each pixel is separated it can store 2 form 0 and 1 (black or white) If for every pixel we have 2-bit so four option so image will have some kind of shading. With 4-bit you get 16 color (present in C-graphics).

If 16-bit per pixel you get 16-color with 4000 shades per each color.

d-RAM uses capacitor → need to be charged → take their time to charge and discharge → therefore slow.

* For an execution of program first the computer's microprocessor has to get instruction which is stored in memory. The first step therefore is to fetch the instruction from memory.
* After fetching and before execution there comes decoding

a = b + c - HLL → compiler
ADD B, C - ASM → assembler
01101011 - LLL/ML/Obj/Bin

Opcode → this is what is stored in memory.

- Understanding the opcode is called decoding
- Every instruction will have its own unique opcode.
- For decoding there is a m x n decoder in the microprocessor.
- For higher microprocessor there is no hard wired decoder there is a microprogram decoder

concept of binary and Hexadecimal.

* every thing inside microprocessor is done in form of hexadecimal because,

decimal has 10 values.
3 bits = 8 values
4 bits = 16 values — 6 values wasted.
so hexadecimal form was required.
also if 8+2 had to be done;

```
 1000
+0010
-----
 1010   → this value wouldn't have a decimal equivalent.
```

Advantages:

a) Every binary combination has a unique value.
b) A single digit would now have higher value compared to decimal [i.e 16 vs 10]
therefore we can store more data in less space. for consideration in a 4-bit digit decimal can go upto 9999 and hexadecimal upto FFFF whose decimal equivalent is 65535.

* Inside computer everything is in binary form but it is not the binary form of decimal number but the binary form of hexadecimal system.

Hexadecimal	binary equivalent
0H	0000
3H	0011
5H	0101
FH	1111
35H	0011 0101 ← 8 bit Numbers & Nos.
FFH	1111 1111

a/c byte

These are hexadecimal converted to binary and for each hex-digit we use 4-bits.

a/c word

16 bit Nos
FFFF H
3524 H

} 16 bit registers ↓ Registers that hold 16 bit numbers.

{Hexadecimal numbers}

Power of 2:

$2^0 = 1$
$2^1 = 2$
$2^2 = 4$
$2^3 = 8$
$2^4 = 16$
$2^5 = 32$
$2^6 = 64$
$2^7 = 128$
$2^8 = 256$
$2^9 = 512$
$2^{10} = 1024 = 1K$.

$2^{11} = 2^1 \times 2^{10} = 2K$
$2^{12} = 4K$
$2^{20} = 1K \times 1K = 1M$
$2^{30} = 1G$
$2^{40} = 1T$

System Bus

↳ used to transfer data/information.

* How does information travel in a bus?

Bus is a set a line, taken a line. The one end of this line can be connected to Vcc or ground. if connected to Vcc the other end gets logic 1 and 0 if connected to ground. The speed with which it happens is not possible by using a physical switch, therefore a digital switch (transistor) is used which can be biased and changed million times faster.

1 line can transfer 1 bit at a time so 4-bit bus is a bus with 4-lines.

sizes of bus is width of bus. 8-bit bus means 8 lines in a bus.

Address Bus:
A memory has millions of spaces divided, each space has its unique address.
In any operation, first the processor sends the address through address bus, this is done to identify the memory location where operation has to be performed else the sent data may corrupt other important data.

Data bus
Data bus will carry the data to be written or to be read from the memory.

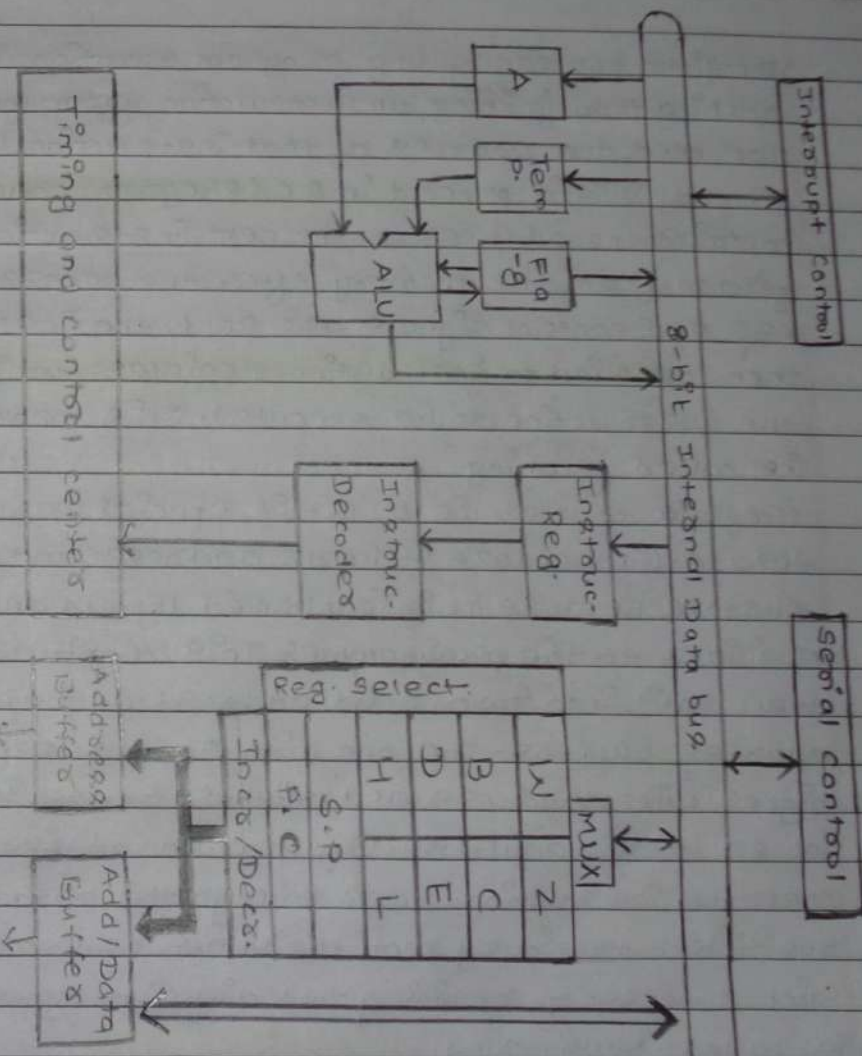
Control bus
Sends the control signals to and from the microprocessor. In this case there are two signal (read and write).

* While writing in the memory, you first select address, then send control signal then write the data which is already sent in data bus.
But while reading, all the process is same but you don't have the data know to transfer through databus as soon as the control signal is sent, so there is a certain delay while reading the data which is called propagation delay.

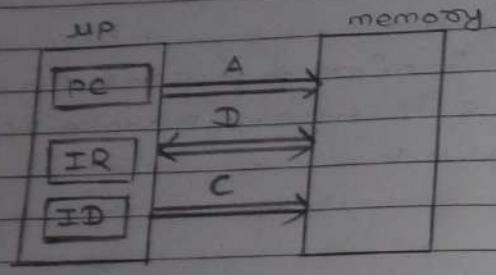
WRITE → address - data - signal
READ → address - signal - data.

8085 Architecture

→ 16 bit Address bus
→ 8 bit Data bus.
→ 8 bit ALU
∴ called 8 bit microprocessor
It has 7, 8 bit General Purpose Registers A, B, C, D, E, H, L.



Program execution



The first process of any program execution is the fetching. For fetching an instruction from the memory we need the address of that instruction in the memory. This is present in P.C (Program counter), which contains the address of the next instruction to be fetched from the memory. After the necessary address and control signals are sent, the instruction is then carried to instruction register which holds the instruction to be executed. This whole process is called fetching.

Program counter is a 16-bit special purpose register, 16-bit because it holds address and special purpose because it is controlled by µp and not available to the programmer. This 16-bit address is then split into two 8-bit address and stored in two address buffers. The one with A_8-A_{15} is called higher order address and another A_0-A_7 is called lower order address. Now when the address is sent then the instruction comes through the Data/Address bus which previously sent the lower order address such process of combining two different types of bus is called Multiplexing.

* since P.C is 16-bit, Incrementer/Decrementer is also 16-bit.

After the process of fetching is done, the instruction are then sent to instruction decoder which are then sent to signal and control center which generate necessary signals for the execution.

During this whole time, after the completion of fetching, the program counter gets incremented by the Incrementer/Decrementer* so, when the next instruction cycle starts the P.C could have the address of the instruction to be fetched.

Therefore, program counter is said to have the address of the NEXT instruction to be executed.

The timing and control center do not itself execute the instructions, rather, it generates the signal which induces the various part of the microprocessor to execute the given instruction. There are 74 possible instructions for 8085 microprocessor, out of which most of them are related with ALU.

Sample execution of 8085 microprocessor.

Suppose we write a program with an instruction to add two numbers in 8085 microprocessor. When the turn of this instruction comes, first the initial operand is fetched from memory to the A register (a/c Accumulator) and the next operand is brought to any of the six general purpose register. After this the Add instruction is executed which tells the ALU to activate the circuit to execute the addition of two numbers. Then the value from the G.P.R is brought to the temp. register, then both of these values are

brought to ALU which performs computation and the final result is stored in A.
While writing a program in any high level language we write

a = b + c

where a is a variable where result is stored whereas b and c contains the operand. In case of 8085 we give such instruction as

ADD B

what this does it takes the data from reg B stores it in temp reg. Add those two value (one from A and another from B) and stores the result on A. Therefore A is the primary register in case of 8085.

* Most of the general purpose registers are placed in a same place except for the accumulator. The up selects the register with the help of Register selector and the value is passed through the MUX. This helps to minimize the hardware requirement and also connecting all the G.P.R to the internal data bus doesn't make sense because the I.D.B is 8-bit and a single G.P.R holds 8-bit data therefore only one register can pass the value at a time anyway.

* As the register A accumulates the value of every operation it is called accumulator.

Flags:
Flag is an 8-bit register which holds the status of CURRENT result. Out of the 8-bits only 5 of them are used as flag while the rest 3 are vacant/unused.

a) zero flag: Tells if the result is zero.
b) sign flag: Tells if the result is negative or positive.
c) Parity flag: used to identify the parity in the result.
d) carry flag: Tells if the result (final) has any carry.
e) Auxillary carry flag: Tells if adding the lower nibble produces any carry.

Registers pairing:
Sometimes while programming, it is not enough to have only 8 bit registers to store the data. Therefore, intel offers programmers to combine two 8-bit registers and use as one. But the registers can't be paired randomly, B and C can be paired, D and E can be paired and H and L can be paired. Among these pairs, the former one will hold the higher bit and the latter will hold the lower bit.

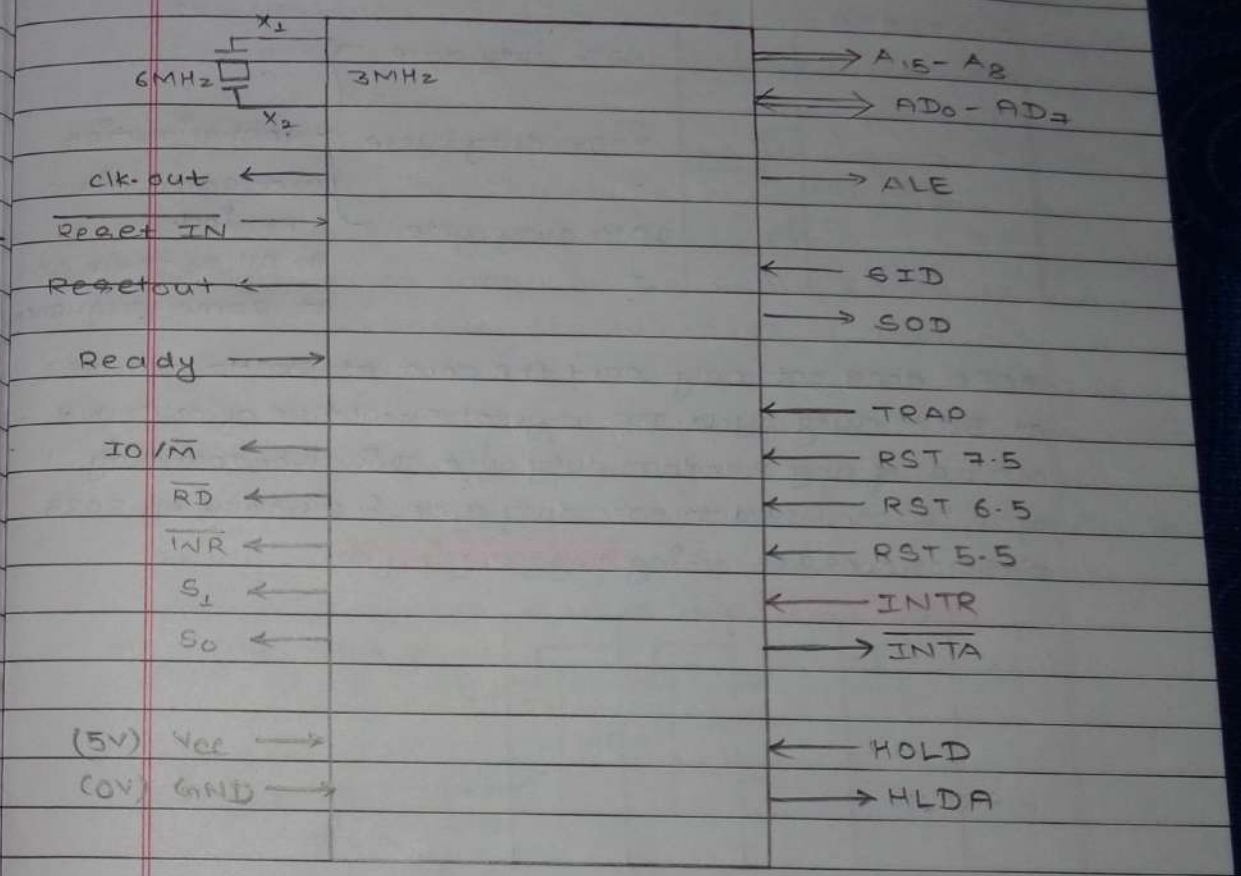
Temporary Register pair:
W and Z are called temporary register pairs, they are the registers for the up to use and not for the programmers. They are used in functions like XCHG.

Stack Pointer

There are three options for a programmer to store the data. The first one is the registers, which should be the goto option as they are located inside the up therefore are faster to access. The rest are in memory (random orders) and in stacks in memory. If the programmer wants to store data in random orders in memory the while passing the data, programmer has to remember the location of each data. On the other hand, stack is the structured format, here the data are stored one above the other and the best part is the address of the top of the stack is known to up through the help of stack pointer. Therefore stack pointer points to the top of stack.

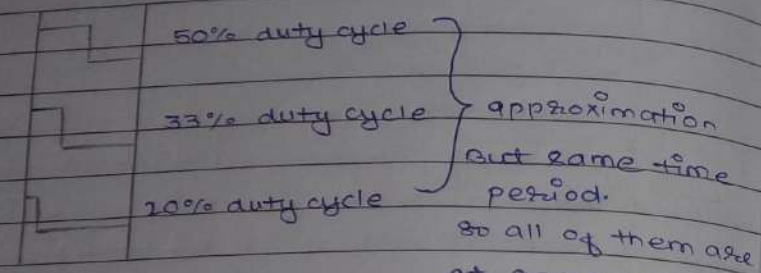
In case of the stack, the data can be accessed in top to bottom format only, therefore the value of the S.P keeps on changing. Whenever data is pushed on to the stack the value of the S.P has to decrease in order to point to that added data and whenever data is popped from the stack, the value of the S.P has to increase. This increment and decrement of S.P is done by incrementer and decrementer. Since, like P.C, SP holds the address memory therefore it is also a 16 bit special purpose register.

PIN DIAGRAM

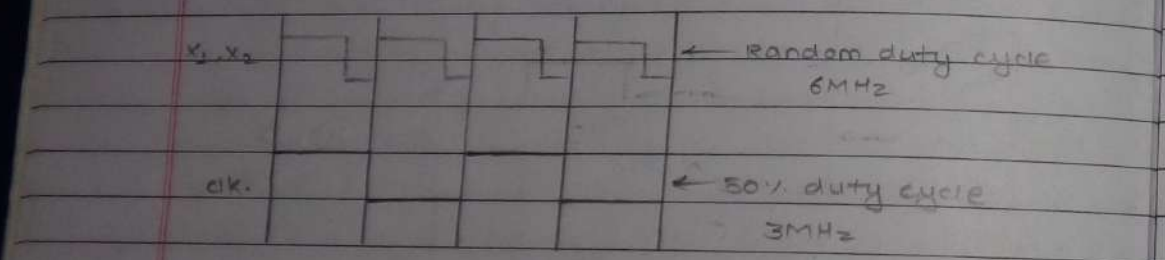


X_1 and X_2 are connected to crystal oscillator, when we charge the plate, the quartz vibrates producing the clock. Clock triggers the microprocessor to perform the next action. Higher the clock frequency higher will the speed of processor. The clock speed of 8085 is 3MHz. However, to produce the clock frequency of 3MHz the frequency of quartz used has to be 6MHz this is because 8085 works at 3MHz at 50% duty cycle.

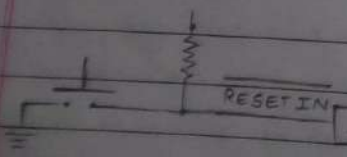
Duty cycle is the time in percentage in a clock cycle upto which the clock is in on state. (high state).



8085 does not only want the clock at 3MHz but also at 50% duty cycle. The crystal should be of 6MHz but can be of any random duty cycle. This random duty cycle is converted to 50% duty cycle internally by 8085 microprocessor using Toggle flip-flop.



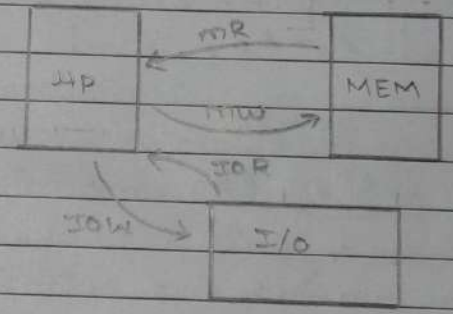
This clock of 3MHz at 50% duty cycle is used by the microprocessor as well as the other chips of the system. This is to make the synchronization among the elements. This clock signal is sent to the elements outside the up through clk out.



Reset IN is an active low signal (triggred when signal is 0) which is used to reset the microprocessor. When the microprocessor is reset it sends reset signal to the other components of the system through Reset out.

Ready signal is used to provide information to the up if other elements of the system are ready to carry out an operation.

Devices do not work at the same speed as the processor. During the transfer of data, peripheral components and up has to be at sync to ensure there is no loss of data. So, the operation is carried out according to the speed of peripheral devices. If the ready signal is 1, it means that device is ready and operation is carried out, else the microprocessor enters the 'wait state'.

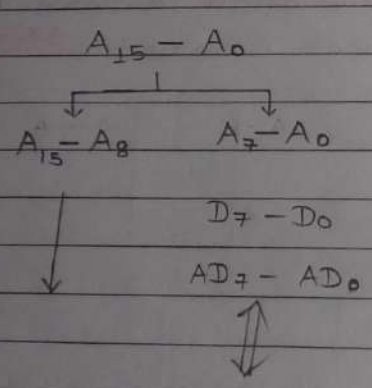


Fundamentally up performs following 4 operations memory read (MR), memory write (MW), I/O read (IOR) and IO write (IOW). For this there are three signals $IO/\overline{M}$, $\overline{RD}$, $\overline{WR}$. $\overline{M}$, $\overline{RD}$ and $\overline{WR}$ are active low signals. $IO/\overline{M}$ is used to define where is operation is taking place, memory or input/output. and $\overline{RD}$ and $\overline{WR}$ denote read and write signal.

S_1 and S_0 are used to denote whether the operation carried out is an input or output operation with respect to the µp.

	IO/M	RD	WR	S_1	S_0
MR	0	0	1	1	0
MW	0	1	0	0	1
IOR	1	0	1	1	0
IOW	1	1	0	0	1

Vcc and GND are for the power supply. Vcc is connected to 5V power supply (hence the name 8085, 8 for 8 bit and 5 for 5V power supply). GND is connected to ground representing 0V (or low signal).



Address in 8085 is of 16-bits carried out by address bus $A_0 - A_{17}$ which is further diveded into two 8-bit buses $A_0 - A_7$ and $A_8 - A_{15}$. In the same way 8085 uses 8-bit data carried by an 8-bit data bus. So in that sense 8085 would need 24 bit bus, 16 for address and 8 for data. But that problem is solved by the technique called multiplexing.

At any given time while performing an operation the microprocessor is either passing the address or transferring the data. µp does not work with address and data simultaneously at any given time. So, taking that into consideration Intel merged $A_0 - A_7$ bus with a data bus and made it $AD_7 - AD_0$ multiplexing bus.

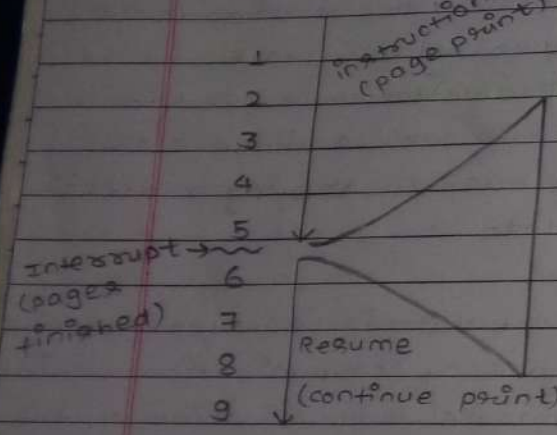
At any given time when we look at the data carried by $A_8 - A_{15}$ it can only be an address but with $AD_{15} = AD_0$ it can either be an address or a data, this is separated using ALE.

ALE (Address Latch Enable) which is used to identify if $AD_7 - AD_0$ bus carries address or data. If ALE is 1, the bus is carrying address if 0, it is carrying data.

On the basis of how the data is transferred there are two different types of communication. Serial communication and parallel communication. If the data is transferred bit by bit then it is called serial communication. If multiple bits are transferred at a time then it is called parallel communication. For parallel communication we need multiple lines (or a bus) therefore it is expensive and used only for short distance communication.

8085 has SID and SOD for serial communication, SID carry data bit by bit into the µp while SOD carry data bit by bit out of the µp. They are called serial input data and serial output data respectively.

Interrupts are the signal sent by the peripherals to the microprocessor to get a service attention. When ever there is a problem in any of the system devices other than the microprocessor, they send an interrupts which triggers the microprocessor to stop whatever it is currently doing and conduct an ISR (Interrupt service Routine) to fix the problem and after that resumes whatever it was previously doing.

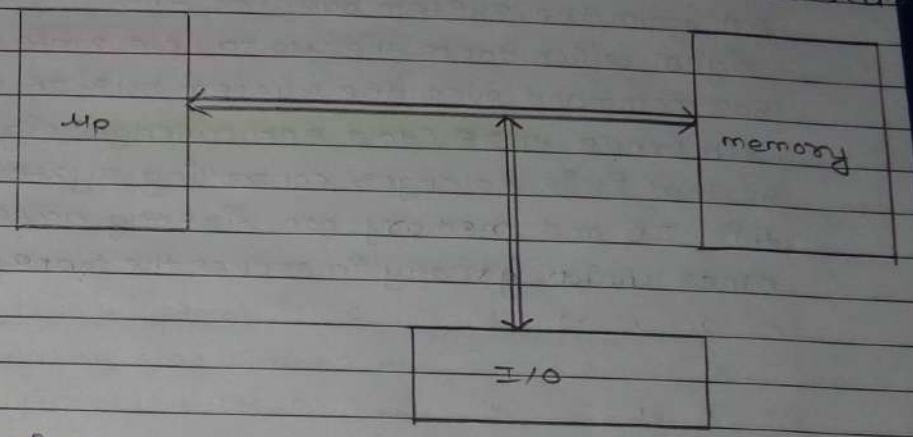


considering the fact that there may be several interrupt signals at a same time, 8085 has 5 pins for interrupt signals. Among these five, there is an order of precedence to make sure that the urgent one is taken care first. These five signal pins are

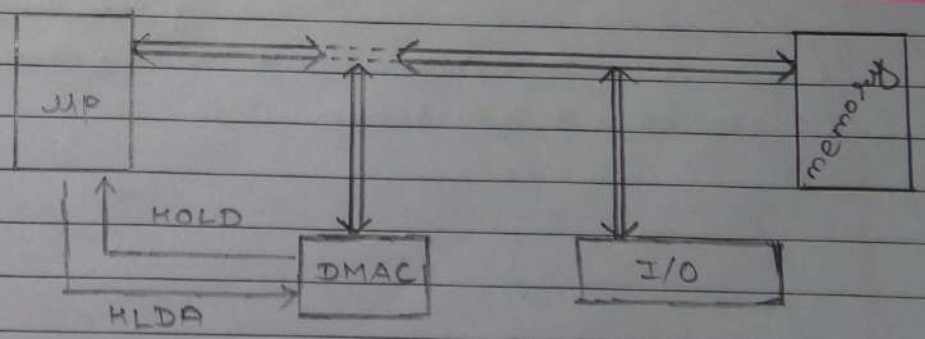
TRAP
RST 7.5
RST 6.5
RST 5.5
INTR

↓ decreasing precedence.

Along with these five pins there is an $\overline{INTA}$ pin which is a/c Interrupt Acknowledge which is used to send a signal to the device facing problem that its problem has been considered.



Technically, memory and I/O can't access each others, meaning that if you want to write something on the memory then that data has to be sent to the up which is then sent to the memory. This might not be a problem for small data files but in case of huge files it takes a hit in speed of data transfer as two cycles are required to transfer a single data chunk. This happens (I/O and memory can't access each other) because they both can't generate any control signal. To solve this a new hardware component has to be added called DMAC.



DMAC (Direct Memory Access control) acts as a bus masters (one that controls the bus) by transmitting the control signal. For controlling the system bus DMAC has to send signal to up to take off its control from the system bus. For this it sends HOLD signal which takes the up to hold state and DMAC has command over the system bus. To response up sends HLDA (Hold Acknowledge) signal to ensure that it is nolonger controlling system bus. After this Io and memory can directly access each other which greatly improves the transfer rate.

8085/8086 Flag Registers

Flag register contains the status of the current result.

status flags of 8085/8086.

SF	ZF	X	AC	X	PF	X	CF

out of these 8 bits, three are don't care bits which value can be 0 or 1 and they donot effect the end outcome.

1) CF (carry Flag)

carry flag stores the information of carry out of the most significant bit (MSB).

eg:

```
  1 1 1 1     1 1 1
  1 1 1 1     1 1 1 1
+ 0 0 0 0     0 0 0 1
(1) 0 0 0 0   0 0 0 0
```

this carry triggers carry flag.

2) Parity flag (PF)

Parity flag tells you the parity of the end result. Parity is known on the basis of numbers of ones. If the result has even parity then parity flag will be 1 else the parity flag will be 0.

eg:

If end result is 1000 0011

number of 1s = 3 (odd)

parity flag = 0

If end result is 0000 0011

parity flag = 1

3> Auxillary carry (AC)
It stores the information of carry from lower nibble to higher nibble.

eg. 1 1 1 1 1 1 1 1 1 1 1 1 This carry triggers AC.
+ 0 0 0 0 0 0 0 1
1 0 0 0 0 0 0 0 0

4> Zero flag (Z)
This flag stores information about if the result is zero. If the result is zero, zero flag is one (1). or activated.

5> Sign flag (S)
Sign flag tells you if the result is positive or negative.
If you take an 8-bit number the MSB of that 8-bit number tells you if the number is positive or negative. This is because unlike in paper where you can put a minus (-) sign infront of the number to denote it is '-ve' in registers you need a binary way of doing it.
If the MSB is '1' it means the number is negative and if it is '0' it means the number is positive.
zero flag copies the MSB of the result. Meaning if zero flag is 1, the result is negative and if 0, the result is positive.

There are two types of numbering system possible.
a) unsigned numbers
considering an 8-bit binary number, in case of an unsigned numbering system all 8 bits are used to represent the magnitude of the number and only +ve numbers are possible in this system. Therefore for 8-bit the range is;
0000 0000 to 1111 1111 or
0 to 255 — total 256 values or
00H to FFH

b) signed numbers
In case of the signed numbers, the most significant bit (MSB) is used to represent the sign of the number. If MSB = 0, then it is positive number if MSB = 1 then it is negative number. Therefore the range of 8-bit number is;
-128 to 127 — total 256 values
-80H to 7F
The unique thing about the negative binary number is that it is stored in the form of 2's complement this is because, it is the only way of getting unique 0 otherwise every other system produces -0 and +0 because of the unique 0, -0 is not taken into consideration and range of -ve numbers elongates to -128 while that of +ve to 127.
∴ 1000 0011 → unsigned 83H
↳ signed -7DH
(2's compliment)
0111 1101

The MSB of signed numbers don't always provide the correct information about the sign of the numbers and such anamolous condition arises in case of overflow.

Additional Flags of 8086:

1) overflow flag (OF)
This flag tells you if your result has overflown meaning it is out of the range of the signed numbers range i.e
a) greater than +127 (7FH) or
b) less than -128 (-80H)
In that case, the sign flag holds the opposite sign i.e 0 for negative and 1 for positive.

examples (all are signed numbers)

```
 42H    0100 0010      OF S Z AC P CY
+23H   +0010 0011       0 0 0  0 1 0
 65H    0110 0101
```

no overflow
positive value
non-zero number
no carry flow lower nibble
even no. of 1's
no end carry

```
 37H    0011 0111      OF S Z AC P CY
+29H   +0010 1001       0 0 0  1 1 0
 60H    0110 0000
```

(1)

```
 42H    0100 0010      OF S Z AC P CY
+43H   +0100 0011       1 1 0  0 0 0
 85H    1000 0101
```

overflow
MSB = 1 but result is +ve.
odd no of 1's.
no end carry

signed flag is 1 because as said earlier sign flag copies the MSB of the result.
Here, the magnitude of the result is correct i.e 85H which can't be demonstrated in 8-bit (7-bit with 1 sign bit) number therefore only the sign is opposite.
Therefore, we must first look at overflow flag and if it is 1, then inverse the result of the sign flag.

All these flags studied till now are called the status flags as they provide the status of the end result and are controlled by ALU.

Control Flags of 8086:
These flags are controlled by the programmers to control different operations.

2) Trap flag (TF)
This flag is used for single stepping the program. In a big program if we face any logical error and want to debug the program then in such condition Trap flag is used. It tells the microprocessor to return the result from each step of instruction. (similar to gdb debugger).
The line after which we want single step programming is used to make the Trap flag 1 and the line where we want to end single step programming is used to make trap flag 0.

3) Interrupt flag (IF)
They are used to enable or disable the interrupts. There may a condition in your program which might have a short time constrain or highly impossible in such case you don't want the microprocessor spending time on any other service program. In such case interrupt flag is disabled meaning no other interrupt signals are processed by microprocessors.

IF = 0 (disabled)

IF = 1 (enabled)

eg: when there is an accident, the program that releases the air bag has to come into action, during that short time period we don't want any interrupt signals to disturb this program so interrupt flag is set to 0.

By default, all the flags, including IF is 0 this is because during booting processor has to perform BIOS operation which has to be done uninterrupted.

4) Direction Flag (DF)
Direction flag is used to determine the direction for string instructions.

ADDRESSING MODES - 8085

The manner in which operand is given in a program is called addressing mode.
The addressing mode available in 8085 are:

a) Immediate
b) Register
c) Direct
d) Indirect
e) Implied.

a) Immediate

Data is given to the instruction itself. up will get the data with the instruction itself therefore it do not have to waste time searching for the data and can perform the operation immediately hence the name immediate.

eg:

MVI B, 25H; B register gets the value 25.
↓
move immediately (B ← 25H)

* Any instruction where there is I there is data else the number is address.

LXI B, 2000H
↓
Load immediately (BC ← 2000H)
B C

* If there is X, there is pair involved.

Advantage - It is simple, you don't need to understand whole program to understand this single line of instruction.

Disadvantage

In 8085, instruction can be of 1 byte, 2 byte or 3 byte.

MOV B, C B=C

Binary equivalent of this instruction is called opcode. There is opcode for entire instruction MOV B, C and every instruction is of 1 byte.

MVI B, 25H B = 25

The opcode for this instruction is for MVI B, not for 25H because 25 is a hardcoded number and if we develop 25H for then we will need for 26H too which would not be efficient, as numbers are infinite.
Here the opcode will have 1 byte and data 25H is of 1 byte so instruction has 2 byte size.

LXI B, 2000H
↓ ↳ no opcode
has opcode 16-bit number
1 byte 2 byte
= 3 byte in total.

Instruction	size (byte)
Push B	1
INR B	1
MOV B, C	1
MVI B, 25H	2
ADI 36H	2
SUI 72H	2
LXI B, 2000	3
STA 3000H	3
SHLD 4000H	3
HLT	1

Immediate addressing mode can't have an instruction of 1 byte as there is data involved. so it can be of either 2 byte or 3 byte.
using more immediate addressing mode increases the memory size of the program.

b) registers
Data will be given by a registers.
MOV B,C;
value will be copied from C to B i:e B=C.
INR B; B=B+1
INX B; BC=BC+1

Advantage: It is smaller to memory size since there is no data in instruction so all instructions will have size of 1 byte.

Disadvantage: It is complex, because to understand what is stored in B after MOV B,C you will first need to know what was stored in C.
(compared to immediate addressing mode)

Initializing three registers to be zero.

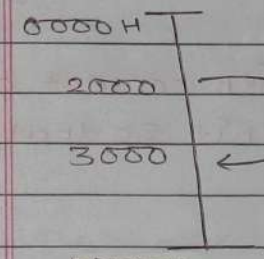
MVI A,00H; 2 byte	MVI A,00H; 2 byte
MVI B,00H; 2 byte	MOV B,A; 1 byte
MVI C,00H; 2 byte	MOV C,A; 1 byte
6 byte	4 byte.

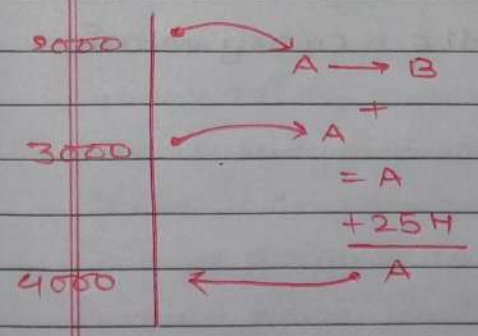
(small program = fast program)

further optimization.
SUB A → A=A-A=0 (1 byte)
MOV B,A 1 byte
MOV C,A 1 byte
3 byte.

c) Direct addressing mode
When the data you want to use is in the memory and you use the address of memory of access it then it is called direct addressing mode.
LDA 2000H;
Load in A the content of memory location 2000H
A ← [2000H]

STA 3000H;
Store the value of A in location 3000H.
A → [3000H]
(a memory location always carries 1 byte)





LDA 2000H
MOV B,A;
LDA 3000H
ADD B;
ADI 25;
STA 4000H

Advantage: It is simple and can easily be understood.

Disadvantage: Each instruction contains a 16 bit address which takes the size of each instruction to 3 byte and makes program larger.

d) Indirect addressing mode:

Most important addressing mode. Here the address will be given by a register apart of that everything is same as in direct mode.

Here, first address is stored in some register* and we use this register to access the memory location hence the name indirect.

eg:

LDAX B;

load A ↓

register pair → because address is 16-bit and each register = 8-bit.

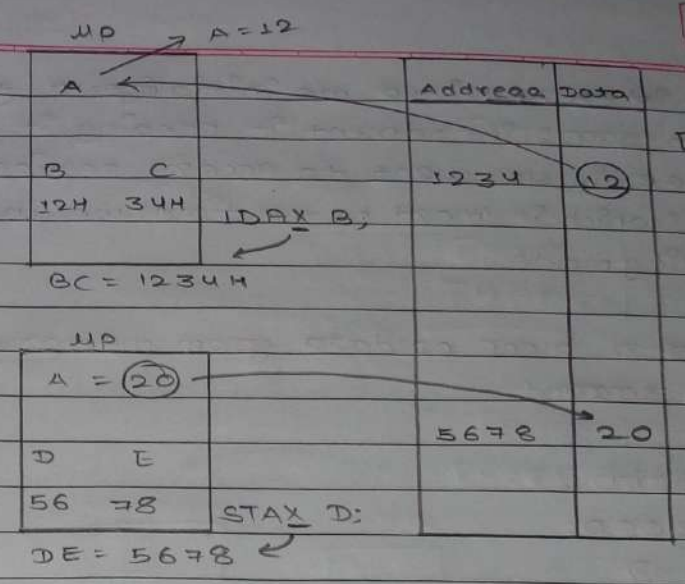
A gets the content of memory location BC.

It's not similar to load A with content of B coz that would be MOV A, B

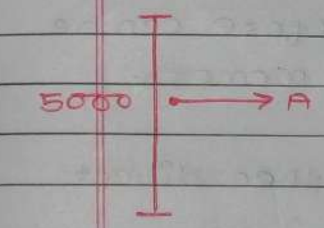
It's not similar to load A with content of BC pair because, BC = 16-bit A = 8-bit so that's not possible in the first place.

STAX D;

store the value of A in the memory location pointed by DE pair.



Why is indirect addressing mode necessary?



Direct

LDA 5000H

Indirect

LXI B, 5000H* [BC = 5000H]

LDAX B

A ← [BC] = A ← [5000H]

Therefore, direct is simpler. If we have to get the value from a single memory address like 5000 in this case we use the direct memory addressing mode. But what if we need to get the data from 100 memory location starting from memory address 5000. Then in such case direct method would be inefficient and we would need indirect method.

If that is required we initialize a register pair to 5000. increment it keeping it on a loop. Therefore, if you want to access series of location (which is most of the time) we use indirect addressing mode.

Transfer a block of data from memory location 2000 to 3000.

```
LXI B, 2000;
LXI D, 3000;
LDAX B;
STAX D;      } put it in a loop.
INX B;
INX D;
```

Advantage: It is flexible and therefore can be kept in a loop to access block of memory.

Disadvantage: It is more complex than direct addressing mode.

e) Implied addressing mode

These are certain instruction which can be performed only on certain operand so operand don't have to be provided they are implied (understood by the microprocessor)

eg: STC; (set the carry flag) It turns carry flag 1.

CMC; (compliment carry flag) if 0 = 1
($\overline{CF}$) if 1 = 0

Advantage: It is simple and small (memory size).

Disadvantage: It is rigid, meaning it works only on some particular operand.

INSTRUCTION SET. (Data transfer)

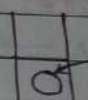
Registers :- B, C, D, E, H, L, A

* M = location pointed by HL pair

(i) MVI B, 25H (can be any register.)
B ← 25H

(ii) LXI B, 2000H
BC ← 2000H

(iii) MVI M*, 25H
M ← 25H HL = 4000
(first HL pair must be initialized)



(iv) MOV B, C (B = C)
B ← C

(v) MOV B, M
B ← M*

(vi) MOV M, C
M* ← C

(vii) LDA 2000H
A ← [2000H]

lower byte lower address higher byte higher address.

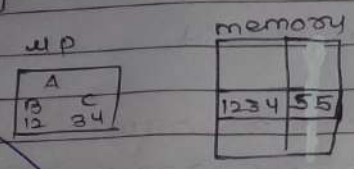
(VIII) STA 3000H.
A → [3000H]

(IX) LHLD 1234H → Load HL pair directly.
L ← [1234H]
H ← [1235H]

(X) SHLD 5000H. → Store HL pair directly.
L → [5000H]
H → [5001H]

(XI) LDAX B
A ← [BC]

(XII) STAX D
A → [DE]



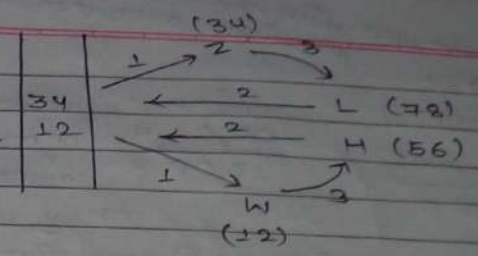
looping possible else
LAD can be used
STA can be used

(XIII) PCHL
PC ← HL

(XIV) SPHL
SP ← HL

(XV) XCHG ← only work on HL pair and DE pair for 8085
HL ↔ DE

(XVI) XTHL ← exchange top with HL
↳ data from top of stack.
L ↔ [SP]
H ↔ [SP+1]



Instruction set (Arithmetic group)

1) ADD B
A = A+B
At most addition can give 9 bit

flag carry
FF
+ FF
(1) FE → A

2) ADD M
A = A+M.

3) ADI 25H → add immediately.
A = A+25H

4) SUB B
A = A-B
If A<B; it needs borrow so to indicate borrow is taken carry flag becomes 1.

5) SUB M
A = A-M

6) SUI 25H
A = A-25H

→ add with the carry of previous operation.

(9) ADC B
A = A+B+CF

```
 12 } FF H
+00 } 01 H
 13   00 H
```

while adding 16-bit numbers we first add lower bit if there is carry we add that carry with addition of higher bits. Here ADC is used.

(9) ADC M
A = A+M+CF

(9) ACI 25H
A = A+25+CF.

→ subtract with borrow

(10) SBB B
A = A-B-CF.

```
  1
  47
-19
 28
```

subtracts the previous borrow.

(11) SBB M
A = A-M-CF

(12) SBI 25H
A = A-25-CF

BC = 12 FFH

(13) INR B
B = B+1

INR C
BC = 13 00 H

INX B
BC = 1300H

(14) INX B
BC = BC+1 *
not done by ALU
done by INCR/DECR.

μp

(15) INR M
M = M+1

B	C
23	FF
HL	
1234	

+1

1234	77 (78)

(16) DCR B
(17) DCR M
(18) DCX B
} decrementors.

→ double addition.

(19) DAD D
HL = HL+DC
so no X required to understand
Normally A is a default register but in this case HL pair acts as default register.
Multiplying 16 bit number 1234H by 2
LXI H 1234H
DAD H

(20) DAA (Decimal Adjust after Addition)
used to perform decimal operation. Works on A register only.

```
 20H      24H      25H
+30H     +34H     +35H
 50H      58H      5AH  expected 60
```

fluke

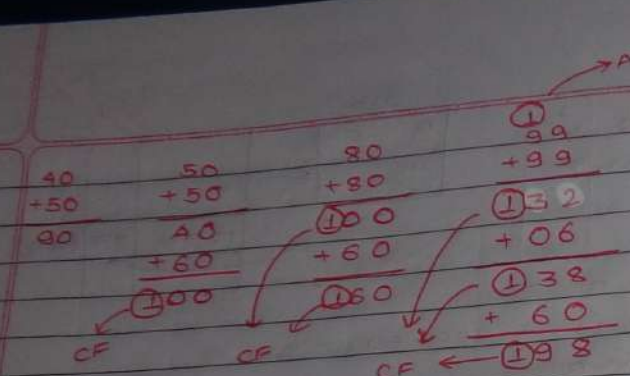
if HN>9 or CF=1 — A — if LN>9 or AC=1

then Add 60 | then add 06.

```
 5A
+06
 60
```

```
 26H
+26H
 4C
+06
 52
```

```
 (1)
 28H
+28H
 50
+06
 56
```



DAA is used to add two decimal numbers so two largest numbers that can be added is 99 and 99.

INSTRUCTION SET (Logic Instructions).

a) AND gate

① ANA B
A ← A^B

ANI F0H → clear lower nibble

A= 0011 0101
1111 0000
0011 0000

② ANA M
A ← A^M

③ ANI 25H
A ← A^25H

XOR
0 0
0

b) OR gate

① ORA A
② ORA M
③ ORI 25H

ORI 0FH → set Lower Nibble

A= 0011 0101
0000 1111
0011 1111

c) XOR gate

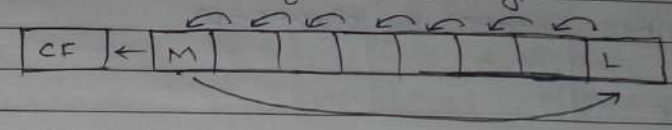
① XRA A
② XRA M
③ XRI 25H

XRI 0FH → complement Lower Nibble

A= 0011 0101
0000 1111
0011 1010

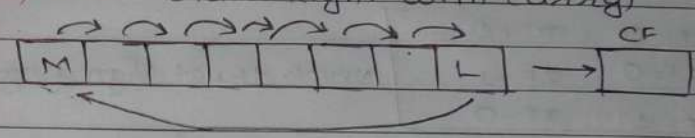
Rotates. ← works on A register only.

1) RLC (Rotate left with carry)



MSB goes to CF as well as to the end.

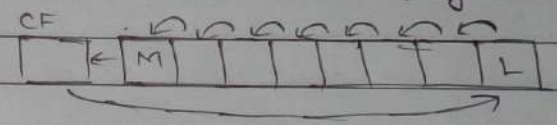
2) RRC (Rotate Right with carry)



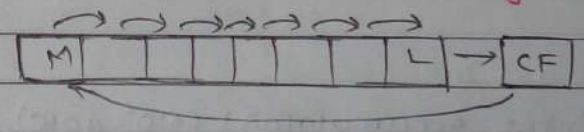
LSB goes to CF as well as to the beginning.

3) RAL (Rotate Arithmetic Left)

Rotates like a 9 bit registers including the register



4) RAR (Rotate Arithmetic Right)



→ to know what this is we rotate it to right x2 the this bit comes to CF, then watch CF, if to know the original no. rotate it left x2.

RLC and RRC will not give original carry afterward RAL and RAR will.

* If LSB is 0 it is even, 1 it is odd.
* If MSB is 1 it is -ve else +ve.
* To count no of 1's rotate and check CF and increase another register everytime CF = 1.
* convert 53 to 35
  rotate 4 times. → ones involve without carry.
* If numbers rotated to left (×2)
  If numbers rotated by right (÷2)

Compare:

1) CMP B
   (A-B)

A>B	CF=0	ZF=0
A=0	CF=0	ZF=1
A<B	CF=1	ZF=0

Never trust sign flag.

2) CMP M (A-M)
3) CPI 25H (A-25)

- STC
  CF = 1
- CMC
  $CF = \overline{CF}$
- CMA (compliment Accumulator) (NOT gate)
  $A = \overline{A}$

INSTRUCTION SET (Stack, IO & machine control)

STACK:

(I) PUSH B. → do not support immediate add. mode
only supports Reg. add. mode
has to be 16 bit Reg.

B	C

1st step: SP ← SP-1
2nd step [SP] ← B
3rd step SP ← SP-1
4th step [SP] ← C

SP-2	C
SP-1	B
SP	XX

* higher bit higher address.

(II) POP B

SP	34
SP+1	12
SP+2	XX

B	C

1: C ← [SP]
2: SP ← SP+1
3: B ← [SP]
4: SP ← SP+1

(III) PUSP PSW (used to push A and F(flag) to Stack)
PSW can only be used with Push & Pop

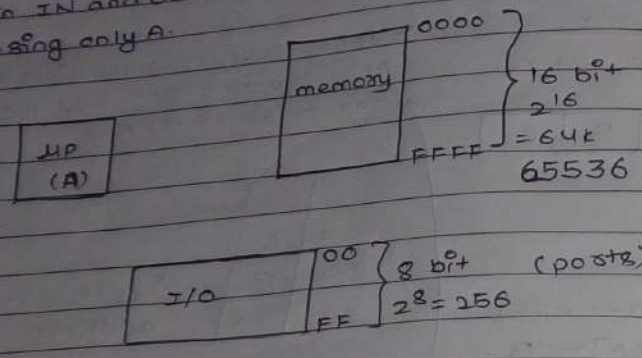
A	F

F	SP-2
A	SP-1
XX	SP.

(IV) POP PSW.

out of 74 instruction only 2 work between I/O and µP.

In IN and OUT instruction data can be transferred using only A.



a) IN I/O address

eg IN 40H

A ← [40H]I/O

b) OUT I/O address

eg: OUT 80H

A → [80H]I/O

Machine control

1) SIM

Set Interrupt Mask

2) RIM

Read Interrupt Mask

3) EI

Enable Interrupts

4) DI

Disable Interrupts

5) NOP

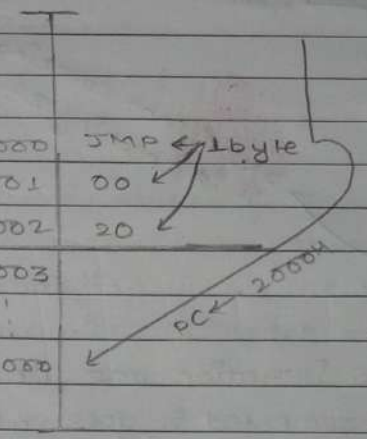
No operation (to create delay)

6) HLT

Halt (stops) µp.

Instruction set (Branch control)

1) JMP addr. ← can also be a label.



00 will be stored in Z
20 will be stored in W

between fetching and execution pc is 1003

Then µp will put

PC ← 2000H (WZ)

conditional Jumps.

a) JC

Jump if carry [CF = 1.]

b) JNC

Jump if no-carry [CF = 0]

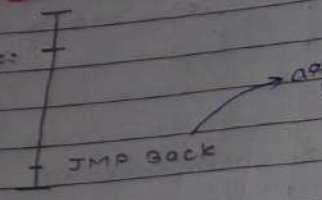
Jump if parity even.

c) JZ	ZF = 1	e) JPE	P = 1	f) JM	SF = 1
d) JNZ	ZF = 0	JPO	PF = 0	g) JP	SF = 0

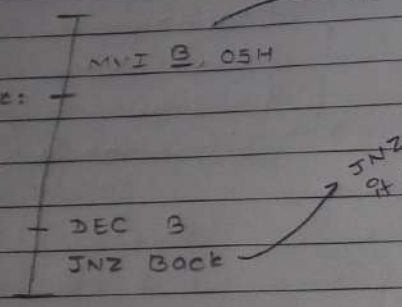
Jump is plus.

Looping

1) go loop:

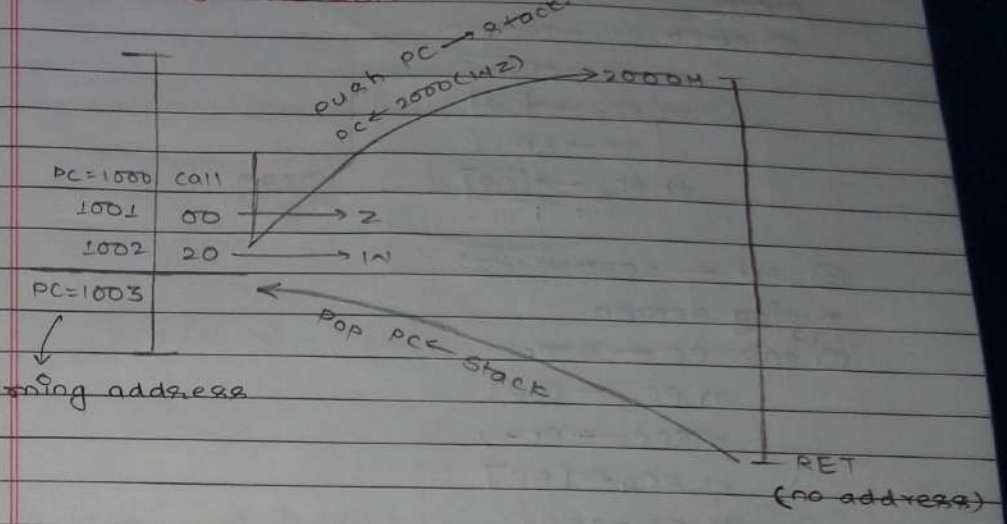


2) Finite loop using JNZ



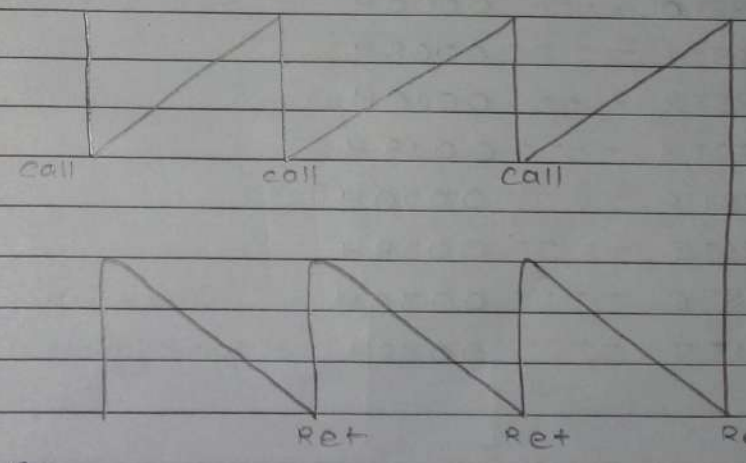
If you want to get 256 iteration use 00H as initializer. when 00 is decremented it goes to FFH which is 1 iteration + 255 operation from FFH to 00H again so in total 256 iterations.

CALL INSTRUCTION:



Since the returning address is required, so up will first push PC to stack before execution of call and changing PC to the value of WZ. So while returning up will pop stack to PC.

Why stack?



{During sub-routine; no. of PUSH has to be equal to POP else address is lost}

Steps:

① Push PC → stack.
   a) SP → SP-1
   b) PC_H → [SP]
   c) SP → SP-1
   d) PC_L → [SP]

② PC ← 2000H (INZ)

During return.

① POP PC ← stack.
   a) PC_L ← [SP]
   b) SP → SP+1
   c) PC_H ← [SP]
   d) SP → SP+1

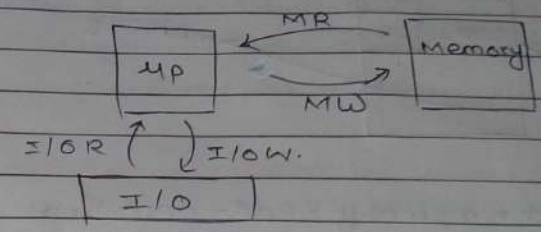
RST n (CALL n x 8)

1 byte call instruction which is used for calling sub routines that are frequently used. It is used for particular locations like

RST 0 ---- 0000H
RST 1 ---- 0008H
RST 2 ---- 0010H
RST 3 ---- 0018H
RST 4 ---- 0020H
RST 5 ---- 0028H
RST 6 ---- 0030H
RST 7 ---- 0038H

TIMING DIAGRAM.

Machine cycle is 1 data transfer in the computer system. The possible data transfer paths are.

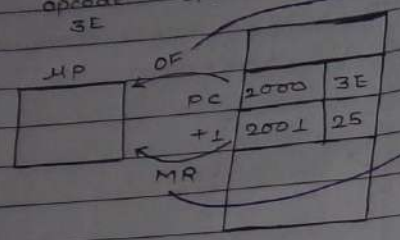


	IO/M	RD	WR	S_1	S_0	T states.
- Opcode fetch	0	0	1	1	1	4T / 6T memory se. + decoding.
- Memory Read	0	0	1	1	0	3T
- Memory write	0	1	0	0	1	3T
- I/O Read	1	0	1	1	0	3T
- I/O write	1	1	0	0	1	3T

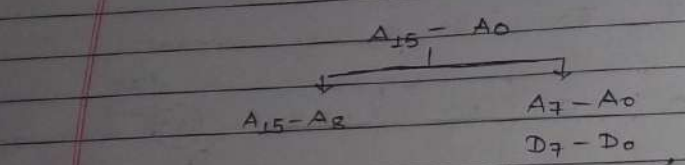
1. send address.
2. send control signal. data travel
3. grab and store data

First machine cycle is always Opcode fetch. It is always memory read operation.

MVI B, 25H
opcode operand
3E

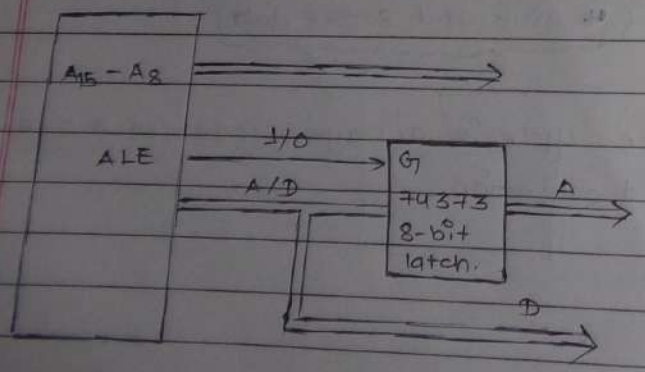


NOTE:
opcode fetch = memory read + decoding.



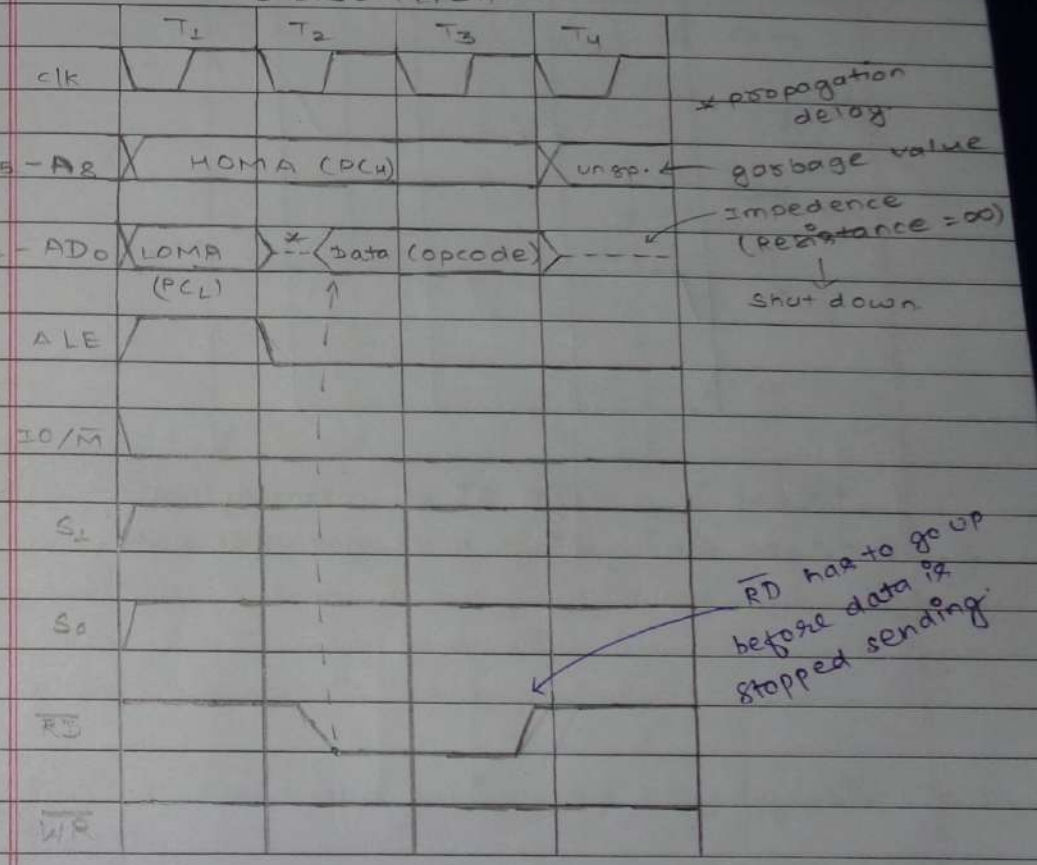
ALE (Address latch Enable) →
1 → AD7 - AD0 is carrying address
0 → AD7 - AD0 is carrying data.

Demultiplexing AD7 - AD0.



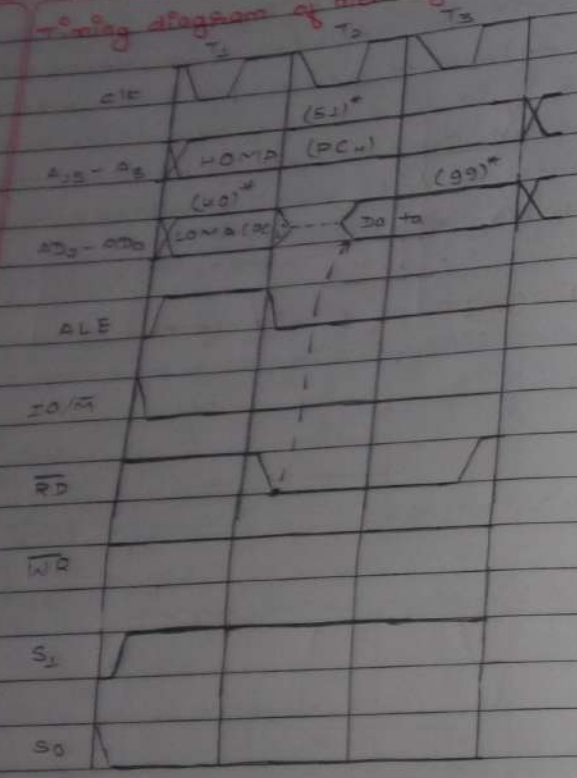
When address is passed ALE is 1 which is passed as gate to 74373, so when address is passed latch allows the address to get through it and also stores it (because it is a set of flipflops). Now when data is passed ALE = 0; gate is closed data goes through another path but the address is still being emitted by Latch.

Timing diagram of opcode fetch.



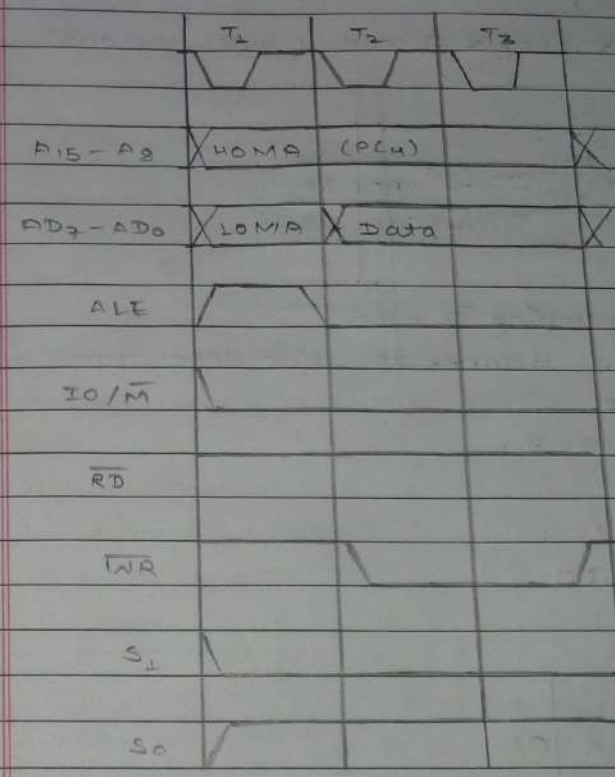
To draw the timing diagram of opcode fetch with 6 T-state just repeat T4 to T5 and T6.

Timing diagram of memory read.



*There is a data 99 at memory location 5140H show the timing diagram to get that data.

Timing diagram of memory write.



Here, there is no propagation delay because address and data is both given by µp so there is no delay where as in read operation µp has to wait for data so there is propagation delay.

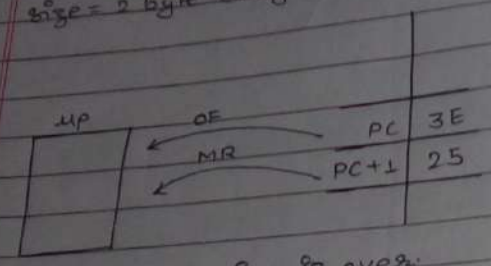
For IO read and write, the only difference is in IO/M.

IO(address) is only 8 bit so same address will duplicate on A15 - A8 and AD7 - AD0.

① MVI B, 25H

B ← 25H

size = 2 byte (1 byte opcode + 1 byte: operand)



After this fetching is over.

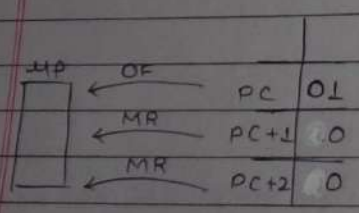
For execution it takes 25 which donot take any machine cycle.

T state = 4+3 = 7.

② LXI B, 2000H

BC ← 2000

size = 3 byte.



After this fetching is over

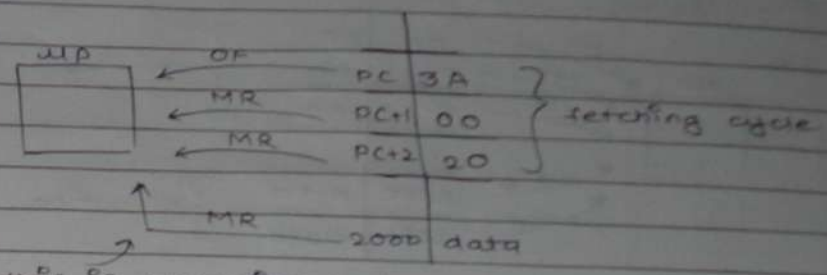
Execution takes immediately since this is internal operation.

T state = 4+3+3 = 10.

③ LDA 2000H

A ← [2000H]

size: 3 byte



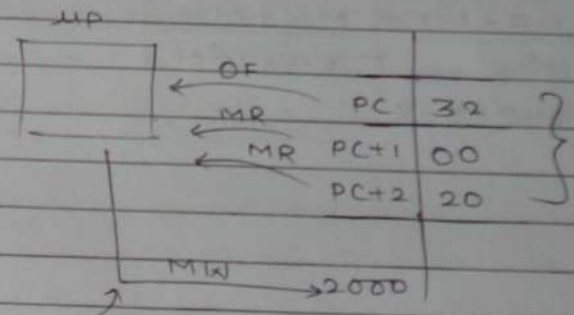
this is execution cycle

T state = 4+3+3+3 = 13.

④ STA 2000H.

A ⟶ [2000H]

size: 3 byte.



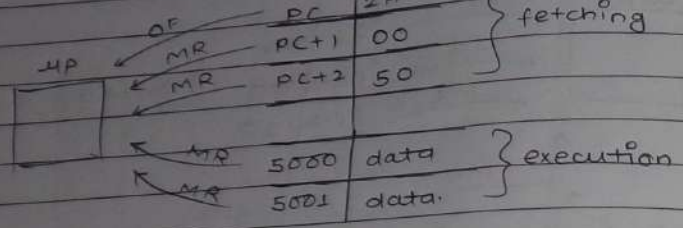
this is execution cycle

T state = 4+3+3+3 = 13

⑤ LHLD 5000H:

L ← [5000H]

H ← [5001H]



T states = 4+3+3+3+3 = 16.

⑥ SHLD 5000H

L ⟶ [5000H]

H ⟶ [5001H]

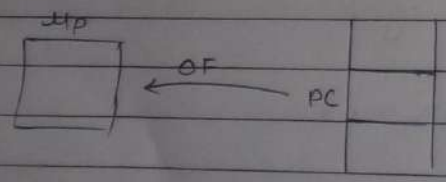
T states = OF + MR + MR + MW + MW
fetching execution

= 4+3+3+3+3

= 16

⑦ ADD B.

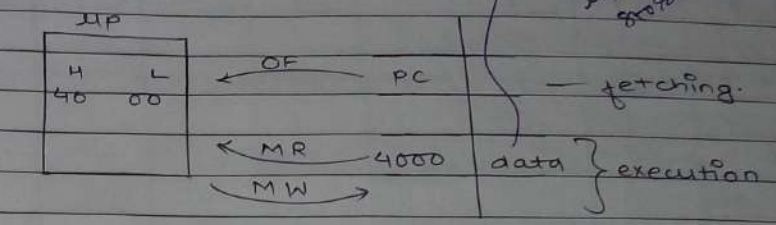
A ⟶ A+B



No machine cycle is required for execution since operation is internal

T state = 4.

⑧ INR M.



T states = 4+3+3 = 10

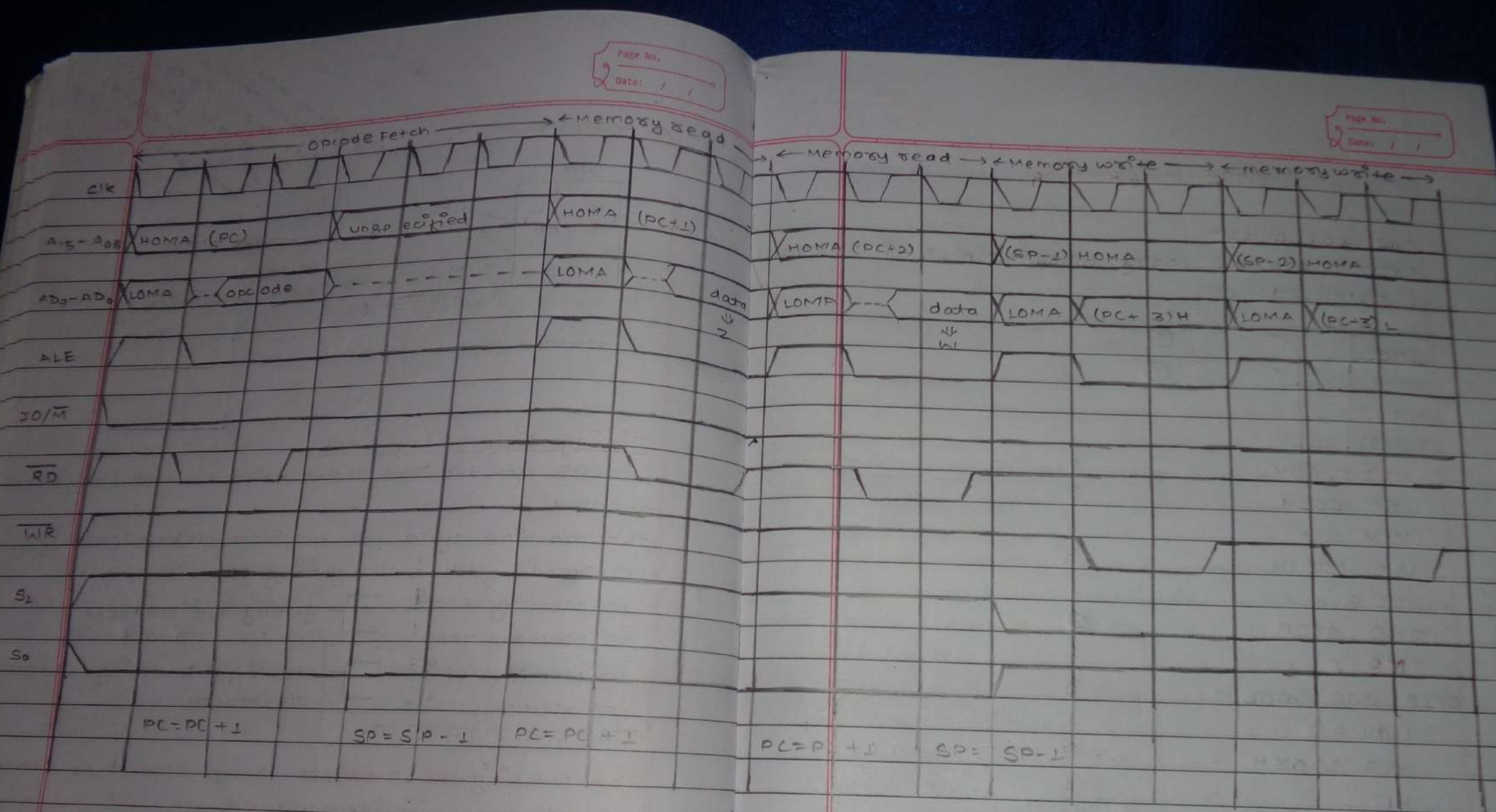
Timing diagram of CALL

STEPS for the call Instruction have already been studied.

The machine cycle for call instructions are.

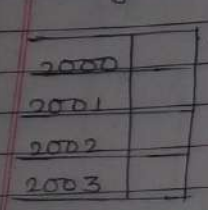
1) Opcode fetch. — getting CALL Instruction
2) Memory read
3) Memory read — getting the address
4) Memory write
5) Memory write — storing returning address to stack.

For pushing the address to the stack, the stack pointer has to be decremented, for 2nd memory write (5th step). SP can be decremented on 4th step since INCR/DECR is free during this period. but for 4th step memory write SP has to be incremented before it, however in 3rd step memory PC has to get incremented therefore INCR/DECR is busy. Hence opcode fetch cycle is extended in order to accomodate for SP decrement and hence is of 6T states.


Opcode Fetch
Memory read
Memory read
Memory write
Memory write
CLK
A15-A08
AD7-AD0
ALE
IO/M
RD
WR
S1
S0
HOMA (PC)
unspecified
HOMA (PC+1)
LOMA
opcode
data
HOMA (PC+2)
(SP-1) HOMA
(SP-2) HOMA
LOMA
data
LOMA
(PC+3)H
LOMA
(PC+3)L
PC=PC+1
SP=SP-1
PC=PC+1
PC=PC+1
SP=SP-1

8085 - Programming

a) Add two numbers between sum and carry; carry is higher bit so stored in higher address.



MVI C, 00H	2
LDA 2000H	3
MOV B, A	1
LDA 2001H	3
ADD B	1
JNC SKIP	3
INR C	1
SKIP: STA 2002 H	3
MOV A, C	1
STA 2003H	3
HLT	1
	22

b) Add two numbers (using M memory pointer)

MVI C, 00H	2
LXI M, 2000H	3
MOV A, M	1
INX H	1
MOV B, M → ADD M	1
ADD B	1 /1
JNC SKIP	1
INR C	3
	1

SKIP: INX H	1
MOV M, A	1
INX H	1
MOV M, C	1
HLT	1
	18/17

- smaller code size.
- no hard coded memory address everywhere.

c) Program to block transfer the data from location 2000H - 2009H to 3000H - 3009H.

```
MVI H, 0AH ; counter
LXI B, 2000H ; source address
LXI D, 3000H ; destination address * 3009H
BACK: LDAX B ; looping
STAX D
INX B
INX D *DCX D
DCR H
JNZ BACK ; looping condition.
HLT
```

* for inverted block transfer.

LDAX H is not allowed
this is done by MOV A, M
STAX H does not exist
MOV M, A

d) Block exchange program.

```
LXI D, 4000H
LXI B, 2000H
MVI H, 0AH
BACK: LDAX B
MOV L, A
LDAX D
STAX B
MOV A, L
STAX D
INX B
INX D
DCR H
JNZ BACK
HLT
```

e) Highest numbers

```
LXI H, 2000H
MVI C, 0AH
MVI A, 00H / SUB A (MVI A, FF) → for lowest number
BACK: CMP M
INC SKIP  (JC SKIP)
MOV A, M
SKIP: INX H
DCR C
JNZ BACK
MOV M, A  /STA <location>
HLT
```

For optimization, you can copy A with the first value at 2000H but after this you have to take the iteration 9 times.

f) Add series of numbers

Initialize A and C to 0, add M to A, if carry increment C else skip at the end A will have lower bit C will have lower bit.

```
MVI D, 0AH
LXI H, 3000H
MVI A, 00H
MVI C, 00H
BACK: ADD M
JNC SKIP
INR C
SKIP: INX H
DCR D
JNZ BACK
MOV M, A
INX H
MOV M, C
HLT
```

g) Adding two 16 bit BCD numbers.

```
MVI C, 00H
LDA 2000H
MOV B, A
LDA 2002H
ADD B ; lower byte addition
DAA ; DAA also considers carry so don't have to follow
STA 2004H
LDA 2001H
MOV B, A
LDA 2003H
ADC B ; adds carry after DAA.
DAA
```

STA 2005H
INC SKIP
INR C
SKIP: MOV B,C
STA 2006H
HLT

b) No of 1's in a number.

MVI B, 00H
MVI C, 08H
LDA 3000H
BACK: RLC ; RAL/RAR/RRC are same in this program.
JNC SKIP
INR B
SKIP: DCR C
JNZ BACK
MOV A, B
STA 3010H
HLT.