

8085 Microprocessor

NovalMS, V.2.0 V.Lobo 2021

INTRODUCTION

Microprocessador 8085

Intel 8080

- First commercial 8 bit microprocessor by Intel
- Successor to the first microprocessor in the world (the 4 bit 4004)

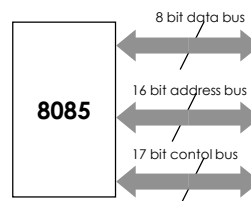
Intel 8085

- Improved version of the 8080
- First microprocessor (μ P) to have commercial success
- Used a lot because of it's simple and "clean" architecture



Basic Architecture

- 8 bit data bus
- 16 bit address bus (64k address space)



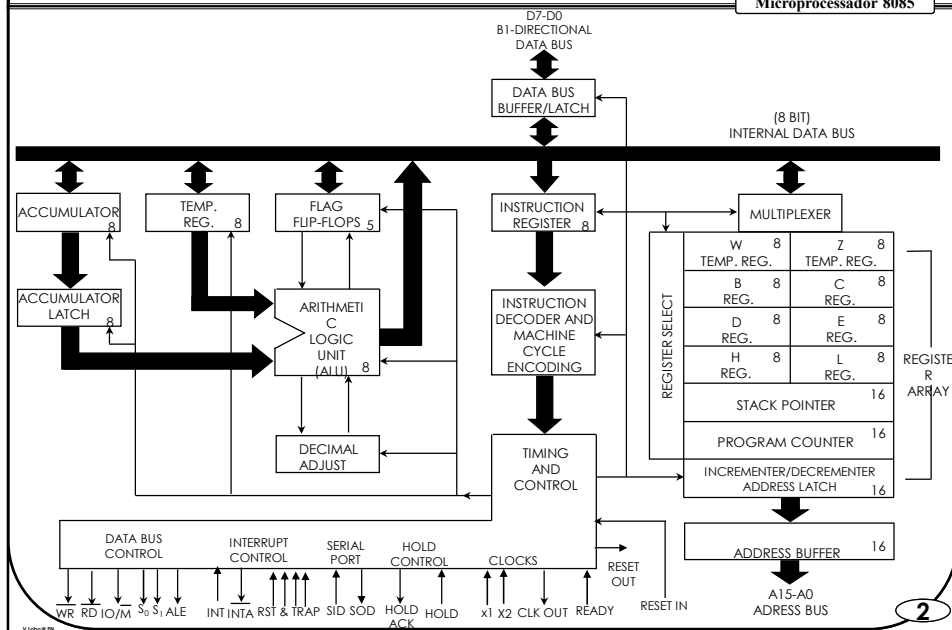
- Precursor to 8086, 80286, 80386, 80486, Pentium,.....,Core i7...

1

1

Internal Architecture

Microprocessador 8085



2

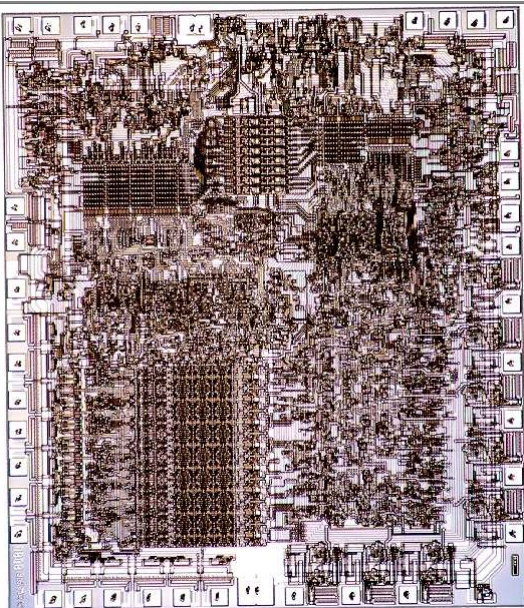
2

8085 Microprocessor

NovalMS, V.2.0 V.Lobo 2021

Silicon implementation

Microprocessador 8085



- Number of transistors: 6000
- Initial manufacturing: April 1974 (8080)
- Manufactures by Intel and various other companies under licences (Siemens, Philips, Texas, etc)
- Different models with varying characteristics:
 - Clock frequency
 - Type of technology (NMOS, CMOS, feature width, etc)
 - Type of encapsulation

3

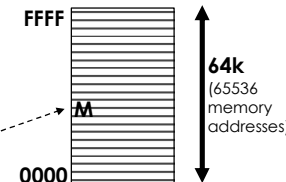
3

REGISTERS

Microprocessador 8085

- **General use registers**
 - There is 1 privileged register, named ACCUMULATOR, that is always used to store:
 - one of the arguments of arithmetic and logic operations
 - The result of arithmetic and logic operations
 - There are 6 general use 8 bit registers, that are sometimes used in pairs to "form 16 bit registers"
 - One the register pairs (H and L, or High & Low) are used to generate addresses. The datum contained in the memory position whose address is stores in HL is sometimes treated as if it were a register (Register M), and is referred to as M or [HL]

A	
B	C
D	E
H	L



Main memory

4

4

8085 Microprocessor

NovalMS, V.2.0 V.Lobo 2021

SPECIAL PURPOSE REGISTERS

Microprocessador 8085

- **Program Counter**
 - Contains the address of the next instructions that will be executed
- **Stack Pointer**
 - Contains the address of the top of the stack (that is managed by the μ P itself)
- **STACK**
 - Used to store data in a LIFO (Last In First Out) structure in the main memory bank
 - Just like a pile of books, where you can add books on top, or remove them from the top
 - You may access the stack with two instructions
 - PUSH - adds a datum (16 bits) to the stack
 - POP - removes a datum (16 bits) from the stack

Instructions:

PUSH BC PUSH DE PUSH HL PUSH PSW
POP BC POP DE POP HL POP PSW
LXI SP, (initial address of the stack)

5

5

FLAGS

Microprocessador 8085

- There is a special register that records events that happened in the ALU (the last operation by the μ P)
- That register has 5 bits, or *FLAGS*
- (almost) ALL arithmetic and logical operations change those flags
- Each bit of that flag may be questioned separately to know if the last operation generated a carry, or was Zero, etc...
- Flags of the 8085 μ P:

Z	S	CY	P	AC
---	---	----	---	----

6

6

8085 Microprocessor

NovalMS, V.2.0 V.Lobo 2021

FLAGS

Microprocessador 8085

- **Meaning of each flag:**
 - **Z Zero**
 - 1 = The result of the last operation was ZERO
 - 0 = The result of the last operation was NOT
 - **S Sign** (same as the MSB, and assumes 2's complement is being used)
 - 1 = The result of the last operation was < 0
 - 0 = The result of the last operation was ≥ 0
 - **CY Carry**
 - 1 = A carry was generated
 - 0 = No carry was generated
 - **P Parity**
 - 1 = Even Parity
 - 0 = Odd Parity
 - **AC AUXILARY CARRY**
 - 1 = A carry was generated in BCD
 - 0 = No carry was generated in BCD

7

7

INSTRUCTION SET

Microprocessador 8085

- **O 8085 has many different types of instructions**
 - It was a CISC at its time (although now even RISC have more instructions)
 - Each instruction is **a number**, or a **machine code**
- **To facilitate reading, each instruction is known by a mnemonic that is related to its function**
 - Example: Instruction number 43_H copies the contents of register E to register B, and its mnemonic is "MOV B,E"
 - A computer program written with mnemonics is said to be written in Assembly Language, ou (incorrectly) Assembler.
- **Assembly instruction can be divided in 4 main tasks:**
 - Move data
 - Perform arithmetic and logic operations
 - Flow Control (deciding which instruction is executed next)
 - Other operations

8

8

8085 Microprocessor

NovalMS, V.2.0 V.Lobo 2021

MOVING DATA

Microprocessador 8085

□ **Addressing Modes**

– There are various ways of specifying which datum should be used:

Addressing mode name	Argument passed to the instruction	Datum used
Immediate	36	36
Direct, by register	A →	36
Direct, by address	2000H →	36
Indirect, by register	[HL] → 2000H →	36
Indirect, by address	[3053H] → 2000H →	36
Indexed	[HL+3] → 1FFDH + 3 →	36

9

9

Moving data

Microprocessador 8085

□ **MOV r1,r2**

– Move data (8 bits) from register r2 to register r1

Examples:

- MOV A,B ; puts in the Accumulator the contents of register B
- MOV H,D ; puts in register H the contents of register D

□ **MOV r1,M / MOV M,r1**

– Moves data (8 bits) from the address POINTED TO by HL to register r1 and vice-versa

MOV M,A ; puts in the memory address pointed to by HL the contents of the accumulator A

□ **LDAX rp / STAX rp (Load Acumulador x...)**

– Moves to the Accumulator the content of the memory address pointed to by rp (or vice-versa)

- STAX BC ; Stores in the memory address pointed to by registers B and C the contents of the Accumulator
- LDAX HL ; equivalent to MOV A,M

10

10

8085 Microprocessor

NovalMS, V.2.0 V.Lobo 2021

Moving data

Microprocessador 8085

- **MVI *r1*, *data*** (8bits) **(move immediate)**
 - Moves to register *r1* the data given.
 - Ex: MVI A,00 ; puts in the accumulator the value 0
 - MVI B,A0H ; puts in B the value 160_{decimal}

- **LXI *rp*, *data*** (16bits) **(load pair immediate)**
 - Moves to register pair *rp* the data given.
 - Ex: LXI BC,0000 ; puts in the register pair BC the value 0
 - LXI HL,3F00H ; puts in the register pair BC the value 3F00

- **XCHG** **(exchange)**
 - Exchanges the contents of HL with the contents of DE

11

V.Lobo 2021

11

Moving data

Microprocessador 8085

- **LDA *addr*** **(Load Accumulator)**
 - Moves data from a given address to the accumulator
 - EX: LDA 1000 ; Loads the accumulator with the number stored in memory address 1000

- **STA *addr*** **(Store Accumulator)**
 - Moves data from the accumulator to a given address (ADDR)
 - EX: STA FFFF ; Stores the content of the accumulator in the memory address FFFF

- **LHLD *addr* / SHLD *addr*** **(Load HL Direct/Store HL Direct))**
 - Moves data (16 bits) from address ADDR to the HL register pair
 - EX: LHLD FFFF ; Loads HL with data from memory address FFFF

12

V.Lobo 2021

12

8085 Microprocessor

NovalMS, V.2.0 V.Lobo 2021

Arithmetic Operations

Microprocessador 8085

- **ADD r / ADC r**
 - Add the contents of register r to the accumulator (with/CARRY)
- **ADD M / ADC M**
 - Add the contents of register the memory address pointed to by HL to the accumulator (with/CARRY)
- **ADI data / ACI data** (*Add Immediate*)
 - Add the number given to the accumulator (with/CARRY)

Example

```
MVI    A, 255
MVI    B, 1
MVI    C, 2
ADD    B
ADI    30
ADC    C
What is the value of A now ?
```

13

13

Arithmetic Operations

Microprocessador 8085

- **SUB r / SBB r**
 - Subtract register r from the accumulator (W/Borrow)
- **SUB M / SBB M**
 - Subtract the contents of memory (pointed by HL) from the accumulator (W/Borrow)
- **SUI data / SBI data**
 - Subtract the given data from the accumulator (W/Borrow)

Example:

```
MVI    A, 10
SUI    03
MVI    B, AFH
MVI    C, 10H
ADD    B
SBB    C
What is the value of A now ?
```

14

14

8085 Microprocessor

NovalMS, V.2.0 V.Lobo 2021

Arithmetic Operations

Microprocessador 8085

- **DAD *rp***
 - Add *rp* to HL, leaving the result in HL

DAD BC

- **DAA**
 - Correct the addition/subtraction in BCD

DAA

- **INR *r***
 - Increment register *r*

INR B

- **INR M**
 - Increment the contents of the memory position point to by HL

INR M

- **INX *rp***
 - Increment the register pair *rp* (*increment eXtended*)
 - Register pairs (BC, DE, ou HL) behave as a single 16 bit number

INX BC

15

V.Lobo@FE

15

Arithmetic Operations

Microprocessador 8085

- **DCR *r***
 - Decrement register *r*

- **DCR M**
 - Decrement the contents of the memory address pointed to by HL

- **DCX *rp***
 - Decrement the contents of register pair *rp*

Exemplos	MVI	H,00H
	MVI	L,FFH
	INX	H
	MVI	L,FF
	INR	L
	DCR	M
DCR	H	
What are the contentes of H and L ?		

16

V.Lobo@FE

16

8085 Microprocessor

NovalMS, V.2.0 V.Lobo 2021

Logic Operations

Microprocessador 8085

- **ANA *r***
 - Logic AND between the accumulator and register *r*

- **ANA M**
 - Logic AND between the accumulator and the memory address pointed to by HL

- **ANI data 8** (and Immediate)
 - Logic AND between the accumulator and datum given

Exemples

```
MVI  A,10H
MVI  B,FFH
MVI  C,10H
CMP  C
ANA  B
CMP  B
```

What are the contents of A,B,C, and the FLAGS ?

17

17

Logic Operations

Microprocessador 8085

- **ORA *r***
 - Logic OR between the accumulator and register *r*

- **ORA M**
 - Logic OR between the accumulator and the memory address pointed to by HL

- **ORI data 8** (or Immediate)
 - Logic OR between the accumulator and datum given

- **XRA *r* / XRA M / XRI I**
 - Logic XOR between the accumulator and register *r* / Memory/datum

```
MVI  A,F8H
ANI  08H
ORI  02
```

What are the contents of A ?

18

18

8085 Microprocessor

NovalMS, V.2.0 V.Lobo 2021

Arithmetic Operations

Microprocessador 8085

□ Comparison operations

- Subtracts the given number from the accumulator, but DOES NOT store the result.
- Changes the FLAGS, that may later be used to take decisions like "IF A=B THEN..."

□ **CMP r**

- Compare the accumulator with register r (compute A-r)

□ **CMP M**

- Compare the accumulator with the memory address pointed to by HL

□ **CPI data₈**

- Compare the accumulator with the given data

19

19

Flow control operations

Microprocessador 8085

□ Jumps

- The next instruction to be executed will not be the next one in the list, but another, stored at a given address

□ **JMP addr 16**

- Jumps to the given address

□ **JNZ addr 16**

- Jumps to the given address if the last operation did NOT result in zero, i.e., if Z=0 (the accumulator ≠ 0)

□ **JZ addr16**

- Jumps to the given address if the last operation resulted in zero, i.e., if Z=1 (the accumulator = 0)

□ **JNC addr 16**

□ **JC addr 16**

- Jumps to the given address if the last operation resulted in a CARRY/ NOT CARRY (CY=0 / CY=1)

20

20

8085 Microprocessor

NovalMS, V.2.0 V.Lobo 2021

Flow control operations

Microprocessador 8085

- **JPE** **addr 16**
 - Jumps if P=1 (parity even)
- **JPO** **addr 16**
 - Jumps if P=0 (parity odd)
- **JP** **addr 16**
 - Jumps if S=0 (positive)
- **JM** **addr 16**
 - Jumps if S=1 (minus)
- **PCHL**
 - Puts in PC the contentes of HL ⇒ Jumps to the address contained in HL.
 - Acts like "JMP M"

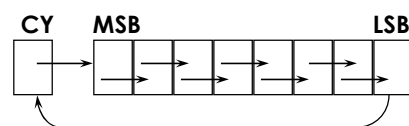
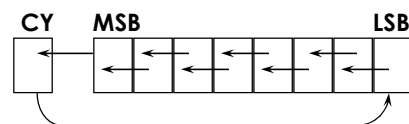
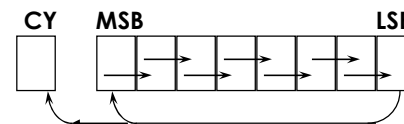
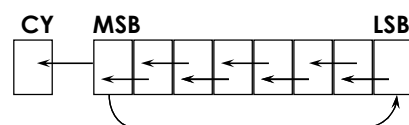
21

21

Rotations

Microprocessador 8085

- **RLC**
 - Rotate (acumulator) left to carry
- **RRC**
 - Rotate right to carry
- **RAL**
 - Rotate left through carry
- **RAR**
 - Rotate right trough carry



22

22

8085 Microprocessor

NovalMS, V.2.0 V.Lobo 2021

Negations and “Set” operations

Microprocessador 8085

- **CMA**
 - Complements the accumulator
- **CMC**
 - Complements the carry flag
- **STC**
 - Set carry flag (puts cy=1)

23

23

Methodology for Assembly Programming

Microprocessador 8085

- **DEFINE OBJECTIVES CLEARLY**
 - Understand and make explicit what operations the program must do. Make explicit what are the **INPUTS** and the **OUTPUTS** that it must produce.
- **DRAW THE BLOCK DIAGRAM or STATE DIAGRAM**
 - Define the basic sub-tasks, how they interact, and how the program progresses through them.
- **DEFINE THE DATA STRUCTURE**
 - Decide **WHAT DATA** is necessary, **WHERE** they are stored, in what **FORMAT**, etc.
 - Define how the μ P registers are used.
 - Write down which variables are stored where (the data lexicon) , and draw a **MEMORY MAP**, showing where data is stored, where the program is kept, etc.

24

24

8085 Microprocessor

NovalMS, V.2.0 V.Lobo 2021

Methodology for Assembly Programming

Microprocessador 8085

□ DRAW A DETAILED FLUXOGRAM

- For each block, draw a separate fluxogram

□ WRITE THE CODE

- Write the code in mnemonics, using a table where you specify **WHERE** the opcode is stored, what the (numeric) machine code is, and how many addresses are used up by each instruction

Mnemonics	Address	Código	N.Bytes	Comments
MVI A,FFH	4000H	XX XX	2	Initialize A (high)
MVI B,02H	4002H	XX XX	2	Initialize B (range)
SUB B	4004H	XX	1	Find the difference
DCR A	4005H	XX	1	Stay within range

EXAMPLE: Compare two bytes that are in consecutive addresses. If they are equal, return A=1, else make A=2. Start the code in 6000H and compare address 2000H with the next.

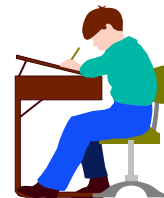
25

25

Examples:

Microprocessador 8085

- Write assembly programs to:
- Compare two bytes that are on consecutive addresses. If they are equal, make A=0, if the first is larger make A=1, and if it is smaller make A=2,. Assume that the first number is at address 3010H
- Sum 2 numbers each one with 24 bits (3 bytes). These numbers are at the addresses stored in register pairs HL and DE. The result should be left at the addresses original pointed to by HL



26

26

8085 Microprocessor

NovalMS, V.2.0 V.Lobo 2021

SUB - ROUTINES

Microprocessador 8085

Objective

- Divide the program into small, modular, and simple tasks
- Create "procedures" that may be "called" from several different places, avoiding duplications

Advantages

- Compact code
 - If you need to do the same thing in different places, you "WRITE ONCE - USE MANY" times.
- Modular code
 - Allows structured programming. Safer and simpler from a software engineering perspective
- Less errors
 - Sub-routines can be tested independently, and "double checked" before being integrated into the system

27

27

CALL/RET INSTRUCTIONS

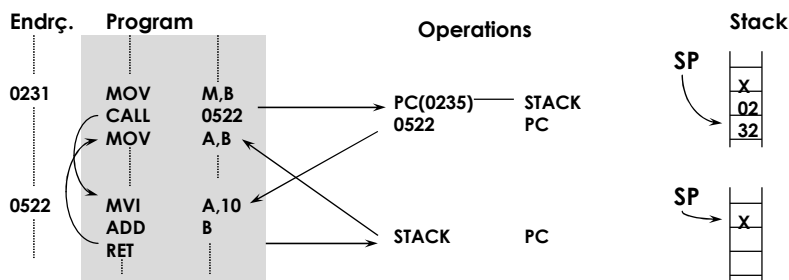
Microprocessador 8085

CALL addr 16

- Calls a sub-routine
- Stores the address contained in PC in the Stack, and jumps to the address given

RET

- Returns from a sub-routine
- Gets an address from the stack (whatever two bytes are there), and jumps to that address



28

28

8085 Microprocessor

NovalMS, V.2.0 V.Lobo 2021

Conditional Calls

Microprocessador 8085

□ Calls are made or not depending on the state of the flags

- CZ / CNZ - Call if Zero / if Not Zero
- CM / CP - Call is Minus / if Positive
- CPE / CPO - Call if Parity Even / if Parity Odd
- CC / CNC - Call if Carry / if Not Carry

□ Exemplo

Convert the data from [2000]
into negative if it is positive

```
LDA 2000
ANA A
CP 2A00
STA 2000
```

Routine that receives a number in the
accumulator, and computes
it's 2's complement

```
CMA ; complements A
INR A ; sums 1
RET
```

29

29

Passing Parameters

Microprocessador 8085

□ 1 – In the REGISTERS

- Limited number of parameters
- Fast and efficient

□ 2 – In FIXED ADDRESSES in main memory

- Forces a permanent occupation of memory addresses
- Slow data passing
- Not relocatable
- Doesn't allow recursion

□ 3 – In the STACK

- Unlimited number of parameters
- Does not interfere with the memory allocation of the rest of the program
- Several conventions for putting/removing data from the stack

30

30

8085 Microprocessor

NovalMS, V.2.0 V.Lobo 2021

Examples of parameter passing

Microprocesador 8085

- Write a routine to sum two bytes, and write a program that calls that routine, using the following methods
 - Passing the parameters in the registers:
 - The routine receives the data in registers B and C, and returns the value in the Accumulator
 - Passing the parameters in fixed addresses
 - The routine receives the data in addresses 20B0H and 20B1H, and leaves the return value in 20B2H
 - Passing the Parameters in the Stack
 - Receiving in the stack the data, and putting the return value also in the stack (using both the “C” and the “Pascal” convention)
 - Receiving in the stack the address of the data and the address where the return value should be stored (and removing those pointers from the stack)

DOCUMENTE ESTAS ROTINAS!

31

31

Passing Parameters to Routines

Microprocesador 8085

- “C” type parameter passing:
 - The routine that puts the data in the Stack is responsible for removing it after returning from the call
 - Allows the number of real parameters to be less than the number of formal parameters
 - Each routine leaves the Stack EXACTLY in the same position where it received it
 - Parameters are put in the Stack “from right to left” (i.e. “backwards” so that the first parameter is the last one to go to the stack)
- “Pascal” type parameter passing:
 - The routine that is called is responsible for removing the data from the Stack
 - There has to be a tight coupling between the routine that calls and the routine that is called to keep the stack balanced
 - Parameters may be put “from left to right”

32

32

8085 Microprocessor

NovalMS, V.2.0 V.Lobo 2021

Documenting Sub-Routines

Microprocessador 8085

- The **INTERFACE** of the routine with the rest of the system must be very well documented
- The documentation must include:
 - What are the parameters ?
 - How are they passed ?
 - Which registers are changed ?
 - What memory addresses are used (if any) ?
- Use (and “abuse” of) comments
- See the examples of the SDK85 monitor routines

33

33

SOFTWARE INTERRUPTS

Microprocessador 8085

- **RST (restart) Instructions**
- They are **CALLs** to pre-defines addresses
- In the 8085 you only have 8: **RST0** to **RST7**
- For each RST you have 8 available addresses before the next RST
 - Generally the interrupt is “vectored” to another address, i.e. the available addresses are used to issue a **JMP** to the location where the routine is actually implemented
- **Jump address of the RST = 8 x number of the RST**
 - RST0 ⇒ CALL 0000
 - RST1 ⇒ CALL 0008
 - RST2 ⇒ CALL 0010

34

34

8085 Microprocessor

NovalMS, V.2.0 V.Lobo 2021

Advantages of using Software Interrupts

Microprocessador 8085

- The calls are independent of the implementation, i.e., you don't have to know where the routines are in memory...
- Interrupts can be easily re-vectored
 - We may change the routines without changing the programs that call them
- The call occupies only 1 byte
- They are usually used for system calls
- Normally, the RST addresses are in ROM, and some are re-vectored to RAM
- RST 0 normally resets the system

35

35

HARDWARE INTERRUPTS

Microprocessador 8085

- Objectives
 - Forces the μP to act in response to an external (that will trigger the execution of a given program)
 - The μP does not waste time verifying the state of the system, because events will make themselves known through the interrupts
 - Urgent events are dealt with minimum delay
- “Perfect” synchronism with external events
 - Wait loops and software based synchronism tend to have large latency.
 - The response to interrupts are (AMOST) immediate.
- “Wakes up” the μP (from “crashes”, wait loops, etc.)
- Two types of interrupts: Direct and Indirect
 - Some are direct to μP pins, other require external circuits

36

36

8085 Microprocessor

NovalMS, V.2.0 V.Lobo 2021

Direct interrupts

Microprocessador 8085

□ **TRAP and RST pins**

- When these pins are actuated, they force the μ P to execute a RST instruction
- A CALL instruction is executed to a given address : $addr = (\text{number of the Rst}) \times 8$
- There 4 such pins:

Pin	Address	Type of actuation
RST 5.5	2CH	by level ("1")
RST 6.5	34H	by level ("1")
RST 7.5	3CH	by flank (0->1) (it has a FF)
TRAP (RST 4.5)	24 H	by maintaining a "1"

37

37

Indirect Interrupts (through INTR/INTA)

Microprocessador 8085

□ **INTR Pin**

- Na external device signals the μ P by actuating the INTR pin
- When it's ready to process the interrupt, the μ P responds by actuating the INTA (interrupt acknowledge) pin.
- When the INTA is generated the μ P will generate a "opfetch cycle" (read an opcode from memory) but without putting an address in the bus
 - Whatever generated the INTR must provide the opcode

Interrupt Request →

Interrupt Acknowledge ←

Something has to send the opcode of the instruction that will be executed

INTR

INTA

8085

D₀-D₇

8

Example:

38

38

8085 Microprocessor

NovalMS, V.2.0 V.Lobo 2021

Simple Example

Microprocessador 8085

- **Increment register B when an external signal is activated**
 - The signal may come from a switch that sends a short impulse when it is flipped

Hardware
(Physical connections)

Interruptor → Rst 5.5
8085

Software

main program

Interrupt routine

address	content
002C	INR B
002D	RET

inst.1
inst.2
inst.3
...
...
...

NOTE: As we shall later see, this routine has some practical problems that would not allow it to work properly

39

39

Interrupt Mask

Microprocessador 8085

- **RSTs may be “masked”, that is, disabled by software**
 - Prevents the μ P from being interrupted when it executes critical code segments
 - Prevents an interrupt routine from interrupting itself
 - Allows the μ P to temporarily ignore a given peripheral
- **The TRAP interrupt is not maskable (NMI)**
 - It is always active
 - It is used many times as a WATCH-DOG to avoid “deep crashes”

8085 Trap ↔ Monostable or Counter

If you don't reset the counter regularly it will RESET the microprocessor

40

40

8085 Microprocessor

NovalMS, V.2.0 V.Lobo 2021

Interrupt Mask

Microprocessador 8085

- You may switch on/of all interrupts at the the same time (except the NMI)
 - DI - Disable Interrupts
 - All interrupt requests are ignored
 - EI - Enable Interrupts
 - Switches the interrupts back on
- You may activate only certain interrupts:
 - There is a register (called INTERRUPT MASK) that may be written/read so as to select which interrupts are active
- To avoid a interrupt interrupting itself, when it is called the mP “automatically” perform a DI. Thus to allow other interrupts to occur a EI must be explicitly performed.
 - EI only produces results 1 instruction later (to allow the RET)

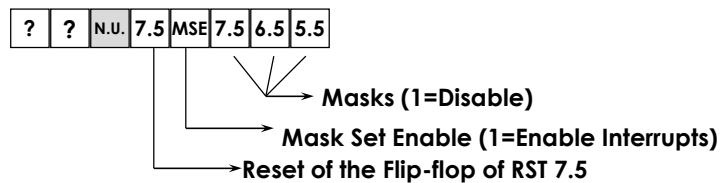
41

41

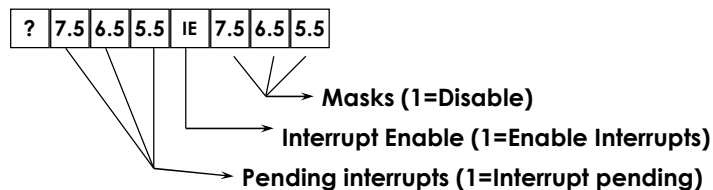
SIM and RIM Instructions

Microprocessador 8085

- SIM - Set Interrupt Mask
 - Puts the contents of the Accumulator in the interrupt mask



- RIM - Read Interrupt Mask
 - Puts the contents of the interrupt mask in the Accumulator



42

42

8085 Microprocessor

NovalMS, V.2.0 V.Lobo 2021

Priorities

Microprocessador 8085

- If more than one interrupt is activated at the same time, there is a priority list amongst them:
- TRAP has the highest priority (and is “non-maskable”)
- RST 7.5 has higher priority than 6.6, that has higher priority than 5.5
- INTR has the lowest priority
- There are more elaborate methods to deal with interrupt prioritization, some supporting “full-nesting”, and a commonly used support chip (8259) to do so.
 - Improved versions of the 8259 are included as modules in most modern support chipsets.

43

43

Interrupt Handlers

Microprocessador 8085

- They are asynchronous routines
 - It is not possible to know at which point in the code they will be called. It is not possible to synchronize these routines with the rest of the program.
 - Thus.. We have to store the context:
 - PC is stored automatically by the “call” instruction
 - FLAGS & ACC have to be stored almost always (with PUSH PSW)
 - Other regs only if they are used
- For convenience they are usually revectorred from their original addresses
 - The interrupt vector does not have space to store all the code
 - The address of the interrupt vector are usually in ROM

44

44

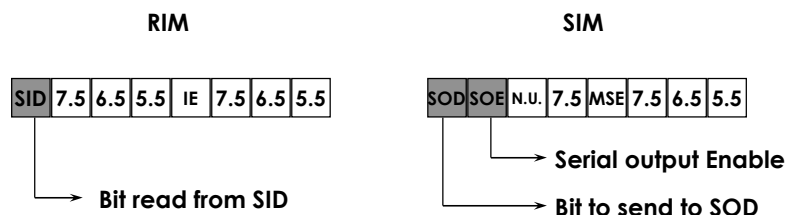
8085 Microprocessor

NovalMS, V.2.0 V.Lobo 2021

Serial I/O with the 8085

Microprocessador 8085

- In the 8085 the **RIM** and **SIM** instructions also control the **serial input/output pins**
 - The pins are named SOD (Serial Output Data) and SID (Serial Input Data)
 - **SIM** - Send data – Sends a bit from the accumulator to SOD
 - **RIM** - Read data – Reads the bit that is in SID to the accumulator



45

45

Parallel I/O with the 8085

Microprocessador 8085

- **Input/Output (I/O) address space**
 - It is an address space similar to the memory, but with only 265 addresses
 - It is generated with addresses of only 8 bits
 - It is used to perform I/O with peripheral devices (or controllers) without wasting memory addresses
- **IN *addr8***
 - Reads the I/O port *addr8* into the Accumulator
- **OUT *addr8***
 - Writes the Accumulator to I/O port *addr8*

MVI A,23H ; puts number 23H in Acc
OUT 20 ; sends it to the I/O port

46

46

8085 Microprocessor

NovalMS, V.2.0 V.Lobo 2021

HLT (HALT) Instruction

Microprocessador 8085

- Put the microprocessor in a “wait” or “idle” state
 - Stops all processing
 - It is only possible to exit this state if there is an interrupt request
 - The response to a interrupt request is extremely fast, because the microprocessor does not have wait for anything (it is idle)

Example of a passive wait:

```
L1:    HLT    ; Waits for an interrupt
      JMP L1 ; Return to the wait
```

47

47

Hardware and external connections

Microprocessador 8085

- There are many different “packaging” options”
- The most common is DIP-40 (Dual In-line Packaging) with TTL levels
- Pinout of the 8085 in DIP-40:

X1	→	1	40	←	Vcc (+5V)
X2	→	2	39	←	Hold
Reset Out	←	3	38	→	Hlda
SOD	←	4	37	→	Clock out
SID	→	5	36	←	~Reset In
TRAP	→	6	35	←	Ready
RST 7.5	→	7	34	→	IO/~M
RST 6.5	→	8	33	→	S1
RST 5.5	→	9	32	→	~RD
INTR	→	10	31	→	~WR
~INTA	←	11	30	→	ALE
AD0	↔	12	29	→	S0
AD1	↔	13	28	→	A15
AD2	↔	14	27	→	A14
AD3	↔	15	26	→	A13
AD4	↔	16	25	→	A12
AD5	↔	17	24	→	A11
AD6	↔	18	23	→	A19
AD7	↔	19	22	→	A9
Vss	→	20	21	→	A8

48

48

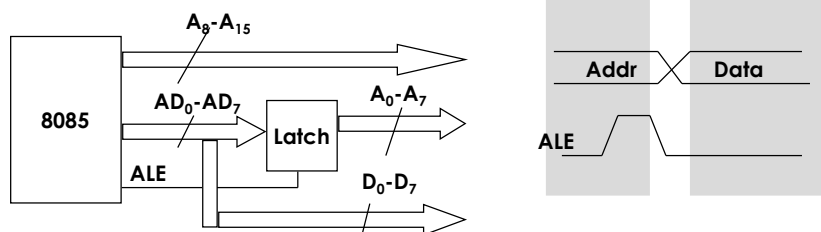
8085 Microprocessor

NovalMS, V.2.0 V.Lobo 2021

Databus Multiplexing

Microprocessador 8085

- To minimize the number of pins in integrated circuit, there are pins that sometimes are used for DATA, and other times are used for ADDRESSES
 - In the beginning of the read/write cycle, they are used to transmit the ADDRESSES
 - At the end of the read/write cycle, they are used for DATA
 - A pulse is generated in the ALE pin (Address Latch Enable) to separate the two types of signals



49

49

Timing/reset/interrupt pins

Microprocessador 8085

- X1, X2
 - Connected to an external crystal to generate (in the 8085 itself) system clock (typical setup)
 - X1 may also be connected to an external Clock Generator
- CLK out
 - System Clock (available for the peripheral devices)
- ~Reset In
 - If kept at "0" during 4 clock cycles forces a re-initialization
- Reset out
 - Used to general a reset signal for the remaining components of the system.
- TRAP, RST x.5, INTR, INTA
 - Interrupt pins

50

50

8085 Microprocessor

NovalMS, V.2.0 V.Lobo 2021

Outros pinos

Microprocessador 8085

- **Vcc e Vss**
 - Power source (5V dc) and ground (0V)
- **SID, SOD**
 - Serial communication pins
- **\sim RD, \sim WR, IO/ \sim M**
 - Read and Write pins with the indication if is to memory or IO devices
- **S0, S1**
 - Status (to see the BUS CYCLE of the microprocessor, in case a Bus Arbitration System is implemented)

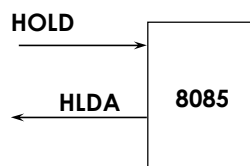
51

51

HOLD and HLDA

Microprocessador 8085

- **Sometimes it is necessary for another device to use the bus that is normally controlled by the μ P, without its interference**
 - Multi-processor system
 - System peripherals / Intelligent sub-systems (for DMA, for example)
- **Operation**
 - When someone wants the bus, it generates a HOLD request to the μ P
 - As soon as the μ P can liberate the bus, it puts all the bus pins in tri-state and activates HLDA (Hold Acknowledge), until the requesting party lowers the request.



52

52

8085 Microprocessor

NovalMS, V.2.0 V.Lobo 2021

Timing and bus cycles

Microprocessador 8085

- All timings are specified in the datasheet
 - “Compatible” integrated circuits: Respect the μP specifications
 - Simplified timing specifications:
 - Consider only what has to happen in each “half-cycle”

Memory Read Cycle (MR)

53

53

Timing and bus cycles

Microprocessador 8085

- “**OPFETCH**” cycle
 - Go get an OPCODE, and execute a “simple” instruction (i.e. a 1 bus cycle instruction, or the first part of a multi-cycle instruction)

54

54

8085 Microprocessor

NovalMS, V.2.0 V.Lobo 2021

Timing and bus cycles

Microprocessador 8085

- **T cycles**
 - Clock cycles
 - Basic operating frequency of the microprocessor
- **M cycles**
 - Machine cycles
 - Groups of clock cycles that do something "with a meaning"
 - They form the "standard cycles" from the BUS point of view
 - There are only 7 machine cycles
 - OPFETCH - Go get an OPCODE (execute it if possible)
 - Memory Read - Get data from memory
 - Memory Write - Store data in memory
 - IO Read - Read an IO port
 - IO Write - Write to an IO port
 - Interrupt Ack - Wait for and OPCODE provided by who requested and Interrupt
 - Bus Idle - Do nothing (the microprocessor is working internally, with no need for using the but, or in WAIT state)
- **Machine Instruction**
 - Has one or more M cycles

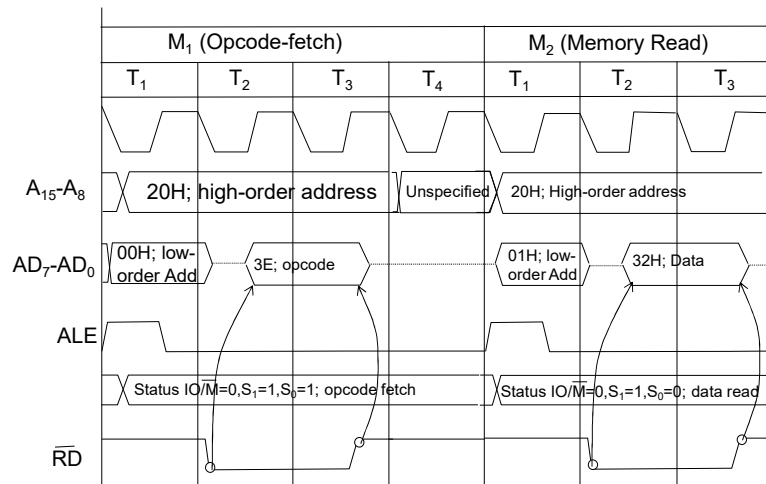
55

55

Timing and bus cycles

Microprocessador 8085

- **MVI A,32H** (executed from address 2000H)

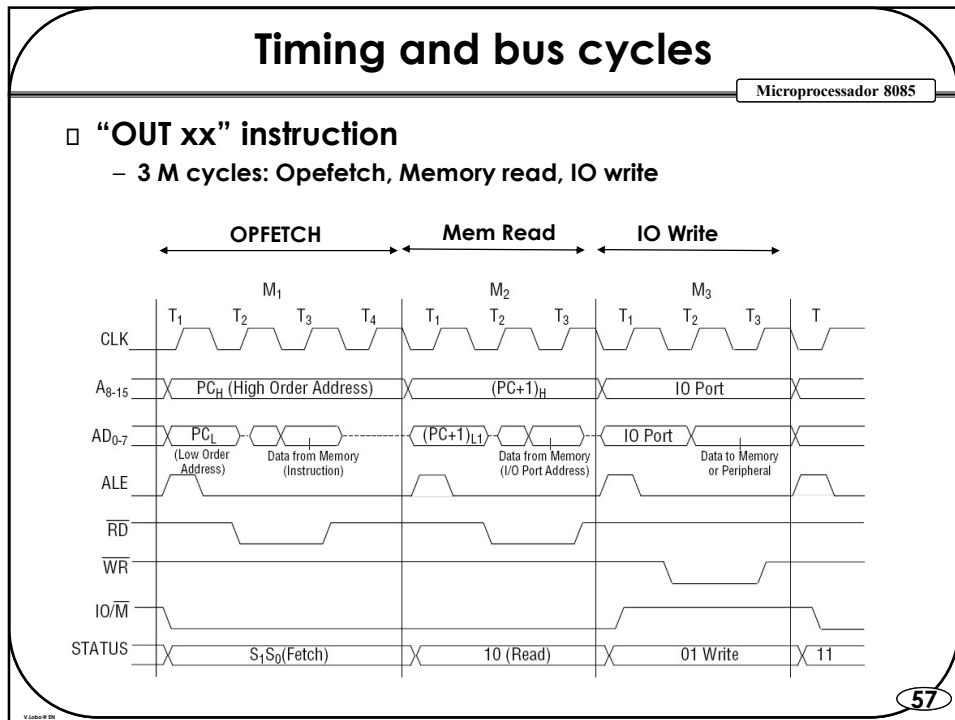


56

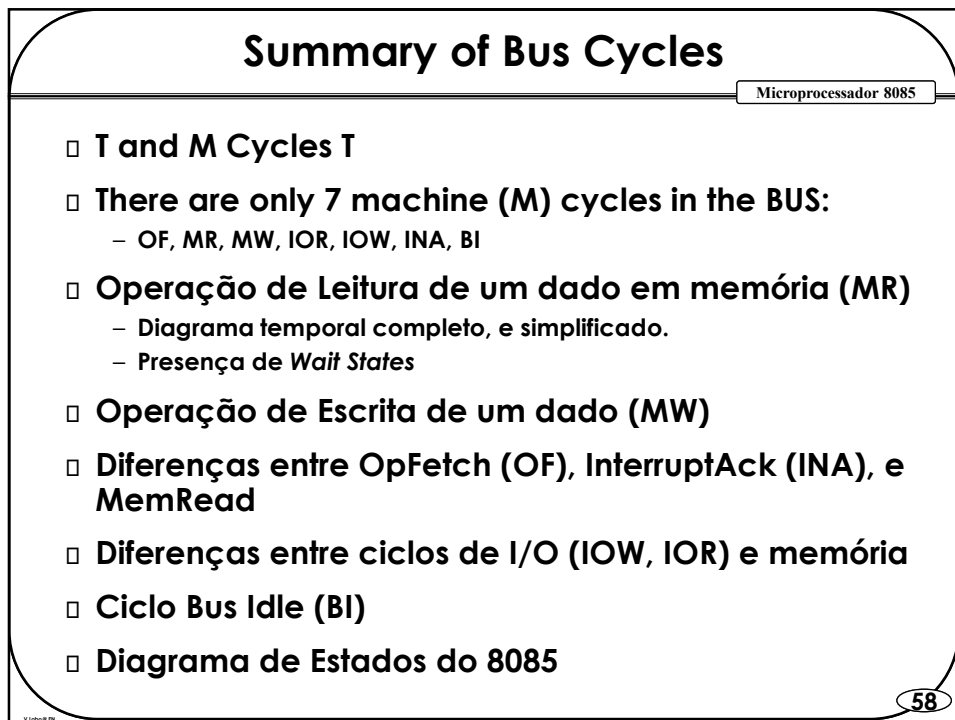
56

8085 Microprocessor

NovalIMS, V.2.0 V.Lobo 2021



57



58

8085 Microprocessor

NovalMS, V.2.0 V.Lobo 2021

