Ministry of Higher Education and Scientific Research
BadjiMokhtar Annaba University
Faculty of Technology

وزارة التعليم العالي و البحث العلمي
جـامعة باجـي مختـار- عنابـة
كليـة الــتكنـولوجيا

Département d'Electronique

Polycopié pédagogique

Titre

Microprossessor and Microcontroller

Cours destiné aux étudiants de 3ème Année Licence Automatique

Année: 2024

Forewords

This course is dedicated to students in the third year of the Bachelor's degree program in Automation and Electronics. It is taught during the first semester for both levels and covers elementary and advanced concepts in architecture, programming, and microprocessors, notably the Motorola 6809, and microcontrollers, particularly the Microchip 16F84.

Throughout the five chapters planned in this course, we progressively master both hardware and software aspects to achieve programming in assembly language for these two components, which lie at the boundary between electronics and computer science. Therefore, this course is necessary to strengthen the skills of both an electronics engineer and a computer scientist and will serve as a foundation for upcoming semesters in both Embedded Systems Electronics and Industrial Automation and Computer Science Master's programs.

Contents

List of Figures

List of Tables

Introduction

Introduction

This course is taught at third year level where we will explore key concepts and essential components that make up the microprocessor system based. A microprocessor-based system is essentially a set of interconnected electronic components, with a central microprocessor as the main element. Microprocessor-based systems are omnipresent in our daily lives, from personal computers to smartphones, household appliances to vehicles. In our exploration, we will delve into the concepts of these systems, with an emphasis on the processor cause it is the central element of a computer system.

The objective of this course is to bridge the gap from hardware to software, thereby transforming a processor into a computer. This transition is based on the programming model of the processor, turning it into a universal programmable machine.

So, since the Microprocessors and Microcontrollers play crucial roles in modern computing and embedded systems, this course provides a solid foundation in the principles, design, programming, and applications of these essential components. By the end of this course, the student will be equipped with the skills and knowledge to tackle diverse challenges in the field of microprocessor and microcontroller systems.

Chapter 1: Microprocessor Architecture

Chapter 1 Microprocessor Architecture

1.1. Introduction to a microprocessor-based system

The introduction to microprocessor-based systems is a fundamental step in understanding the world of computing and electronics. In this introduction, we will explore key concepts and essential components that make up these systems.

A microprocessor-based system is essentially a set of interconnected electronic components, with a central microprocessor as the main element. This microprocessor is a computer component that executes instructions stored in memory to perform various operations. Microprocessor-based systems are ubiquitous in our daily lives, from personal computers to smartphones, household appliances to vehicles.

At the heart of every microprocessor-based system is this central component, the microprocessor, which acts as the brain of the system. It can execute a series of logical and arithmetic instructions to perform complex tasks. Instructions are stored in the system's memory and are read and executed sequentially by the microprocessor.

In addition, a microprocessor-based systems also includes other essential components, such as memory (for data and program storage), input/output devices (for interaction with the user or other devices), and various interfaces to facilitate communication between components.

This chapter will provide you with an overview of the basic structure of a microprocessor-based system. As we delve further into our exploration, we will delve into the concepts and applications of these systems, with an emphasis on their central role in our modern technological world.

1.1.1. Components of a microprocessor-based system

As seen in figure 1, a microprocessor-based system is composed of three main elements:

1. **Central Processing Unit (CPU**): This central unit is the brain of the system. It executes instructions and performs calculations. This is where the actual data processing takes place.

2. **Memory (ROM and RAM):** Memory is used for both temporary storage (RAM - Random Access Memory) and permanent storage (ROM - Read-Only Memory) of data and programs. RAM is volatile, meaning it loses its content when the system is turned off, while ROM retains essential data like the BIOS. In fact, memory serves for the storage of two types of information:

   Data which are information processed by the microprocessor and i**nstructions** which is a set of encoded information that manages the microprocessor's activity.

3. **Input/Output Ports**: These ports enable the system to communicate with external devices, such as input devices (like the keyboard) and output devices (like the screen or printer). They serve as interfaces to interact with the external world.

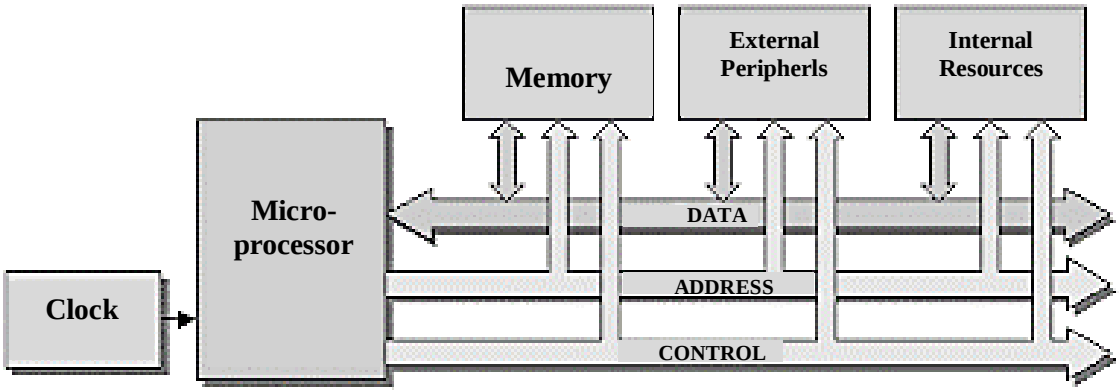


Figure 1: Microprocessor based system Architecture

1.1.2. Connexion of the different modules of a microprocessor based system

A microprocessor must control the functions performed by other modules which are connected with. It must fetch and decode instructions stored in memory, and it must address input/output interfaces to read data from the external world and output the results of its processing [4]. The three essential modules of a microprocessor based system presented in the last section, are interconnected as illustrated in the following figure (1). They are connected via three types of buses :

- **Data Bus**: This consists of a set of electronic tracks that carry data between various components of the system. It facilitates the flow of data within the system and also with external peripherals.

- **Address Bus**: This bus is used to specify the memory location or the address of the data location to be read from or written to. It informs the microprocessor where to find the necessary information.
- **Control and Command Bus**: This bus carries control and command signals that orchestrate the system's operations. It determines which operations need to be performed and when they should be executed.

  These buses play a crucial role in the internal and external communication of the microprocessor, ensuring that data is properly transmitted between the various components of the system and external devices. They form the foundation upon which the operation of a microprocessor-based system relies. In some cases, the data bus and address bus are multiplexed onto a single bus. External logic must then perform the demultiplexing.

- **Input/output interfaces** : enable the microprocessor to communicate with the external world. There are ports exclusively used for input and others exclusively for output. Bidirectional ports also exist. Thus, the microprocessor can read data from input interfaces (e.g., mouse, keyboard, hard disk, etc.) and can output the result of its processing to the external world by addressing output interfaces (such as printers, keyboards, etc.). Therefore, input/output interfaces relieve the microprocessor from direct communication with the external world.

1.2. Internal Architecture of a Microprocessor

A Central processing unit, CPU, consists of the following interconnected functional components: Registers., Arithmetic and Logic Unit ALU, and Control Unit, as shown in figure 2, [4].

1.2.1. Registers

1.2.1.1 Accumulator

This is a general-purpose register that receives operands, intermediate results, or results from the arithmetic and logic unit. It helps avoid frequent memory accesses, thus reducing calculation times. Therefore, most arithmetic and logic operations are performed in the accumulator.

1.2.1.1. Program Counter

The program counter contains the address (offset) of the next instruction in memory that needs to be executed. In other words, it instructs the processor about the next instruction to execute. The program counter register is continually updated after the execution of each instruction to point to the next instruction. Microprocessors in the x86 family, for instance, rely entirely on the program counter register to determine the next instruction.

1.2.1.2. Instruction Register

Every operation that the microprocessor is going to perform is encoded (meaning that for each instruction, a code is assigned that cannot be modified or changed by another code). This code is called the « instruction code » or « operation code. » To execute an instruction, the microprocessor sends the address located in the program counter register to memory. Memory returns the byte at this address (the instruction code), which is then stored in a register called the instruction register (IR). Therefore, the instruction register contains the next instruction to be executed by the processor.

This instruction is transmitted (via a data bus) to the instruction decoder, which is responsible for interpreting it.

1.2.1.3. Instruction Decoder

The instruction decoder interprets the instruction contained in the instruction register (IR). In other words, it determines what operation needs to be performed (e.g., addition, branching, etc.) and how to retrieve the operands required for this operation (e.g., the numbers to be added). The instruction decoder then communicates with the control and command unit, which can trigger the events accordingly.

1.2.1.4. Address Registers

These registers are used to manage memory addressing. The processor can use a single register or a pair of registers to access a memory location, and since registers can be incremented or decremented, it’s possible to access data in memory that is stored in an adjacent manner (such as arrays).

1.2.1.5. Status Register

Also called the FLAGS status register, it is used to hold the state of certain operations performed by the processor. For example, when the result of an operation is too large to fit into the target register (the one meant to hold the result of the operation), a specific bit in the status register (the OF or Overflow Flag) is set to 1 to indicate overflow.

1.2.2. Arithmetic and Logic Unit

As its name suggests, the ALU unit can perform two types of operations :

- **Arithmetic Operations**

These include addition and subtraction, which are basic operations (subtraction is an addition with two's complement), as well as multiplication and division. The data processed is considered in integer representations.

- **Logical Operations**

These operations are performed bit by bit on bits of the same weight in two words, such as AND, OR, NOT, XOR, as well as rotation and shift operations (arithmetic and logical). It receives its operands (the bytes it manipulates) from the data bus. These operands can come from registers or memory. At the end of an operation, the ALU can modify certain bits of the status register (FLAGS). For example, in the case of overflow during an addition (when the result of the addition is too large to fit into a register), the ALU sets the overflow bit in the FLAGS register to 1.

1.2.3. Control Unit

Synchronized by the clock signal, this unit is responsible for initiating events within the processor (it is worth noting that it is connected to all other components of the processor). For instance, when information travels on a bus, it is intended for a specific destination, such as a register. Therefore, the control and command unit « unlocks » the entry to that destination so that the information circulating on the bus can enter it (and not enter elsewhere simultaneously). Essentially, it functions as an automaton that executes different sequences specific to each instruction. This automaton can be implemented in various ways (either hardwired or microprogrammed), and in both cases, the instruction set remains fixed. Most processing units are microprogrammed, thus featuring fixed instruction sets [7].

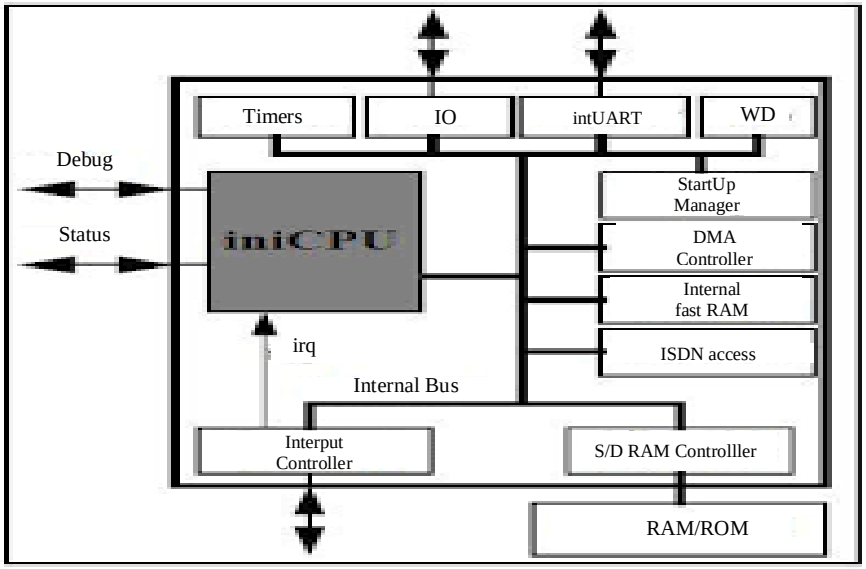


Figure 2: Internal Architecture of a micro processor (6809 de Motorola)

The external architecture of a microprocessor not only means to describe its external shape but also means how the different external devices exchange with the processor. The so-called external architecture of a CPU is defined by the processor's external view. Therefore, it involves understanding instruction data and their processing through the organization of register memory, input and output accesses, etc. In fact, hardware or peripheral devices is allowed to communicate with the processor through Memory and Interruption signals. Of course, these signals will carry different names from one processor to another, but they are almost always present.

1.3.1. Communication with Input/Output

This can involve a flow of information from the outside to the computer (acquisition via the keyboard, a network connection, a hard drive, etc.) or a flow from the computer to the outside (screen, network, disk, etc.). The data exchanged between a device and the processor passes through the interface (or controller) associated with that device. The interface has a buffer memory to store the exchanged data (depending on the type of interface, this buffer memory can range from a single byte to a few megabytes). The interface also stores information to manage communication with the device, as seen in figure 3, [4]:

- Command information to define the operating mode of the interface: data transfer direction (input or output), data transfer mode (polling or interrupt), etc. These command instructions are communicated to the interface during its initialization phase before the start of the transfer.
- Status information that records how the transfer was carried out (transmission errors, receipt of information, etc.). This information is intended for the processor.

During the execution of input/output instructions, the processor sets its IO/M pin to 1 and presents the I/O address on the address bus. The IO/M signal informs the address decoding circuits that it is not a main memory address but the address of an input/output interface.

Communication between the microprocessor and input/output interfaces can be serial (using a single wire, bit by bit) or parallel (using multiple wires), figure 3.

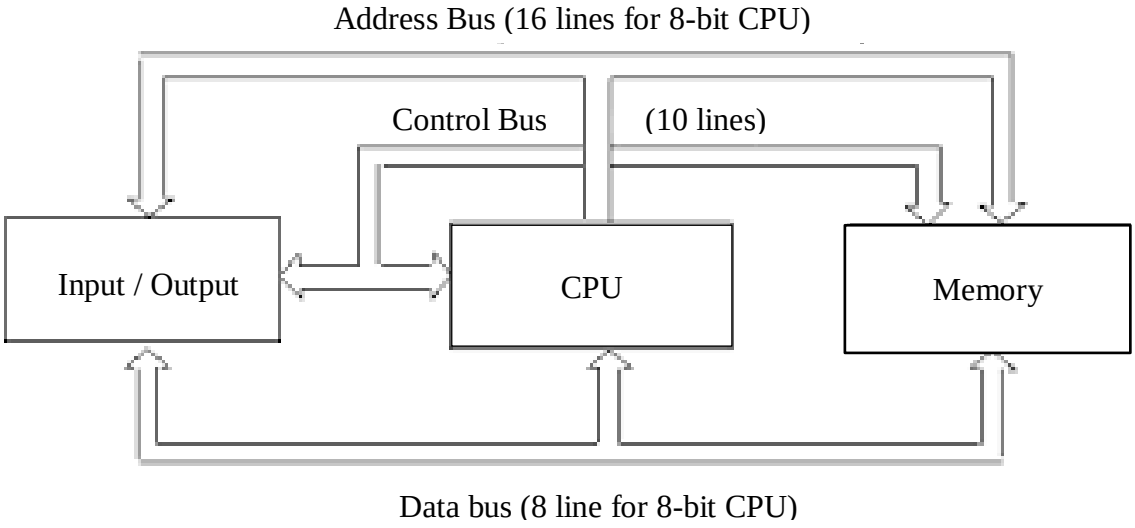


Figure 3: External Architecture of a micro processor

Application Exercise

1. Enumerate the advantages of the ***Harvard*** architecture.
2. What is referred to as the location where the microprocessor attempts to execute the first instruction after Power On Reset (POR)?
3. What components make up a microprocessor-based system?
4. The status register of a processor behaves as follows with the following indicators:
   - C (Carry): Capacity overflow indicator; this bit is inverted in case of borrowing.
   - DC (Decimal Carry): 4-bit overflow indicator.

- Z (Zero): Indicates that the result is zero.

Determine the state of these indicators in the following cases : 8$+2F$; 9F$+61$; 1B$–20

Solution

1. The benefits of the Harvard architecture :

- Simultaneous access to program and data memories: hence faster execution of instructions.
- Limits the issue of accidental program modification.
- Well-suited for implementation on FPGA. However, this architecture also presents drawbacks such as:
- More challenging to implement.
- Increased number of circuit pins.

2. This location is referred to as the "RESET vector."
3. A microprocessor-based system comprises: a microprocessor, RAM and ROM memories, and peripherals.
4. 38$: 00111000 + 2F$: 00101111 9F$: 10011111 + 61$: 01100001 1B$: 00011011 – 20$: 00100000 = 67$: 01100111 = 100$: 100000000 = FB$: 11111011
   C = 0 DC = 1 Z = 0 ; C = 1 DC = 1 Z = 1 ; C = 0 DC = 1 Z = 0

1.4. Conclusion

In conclusion, microprocessors based systems are the cornerstone of the digital age. Their continual advancements are transforming our way of life, work, and interaction with the world. Faced with technical, ethical, and environmental challenges, the industry continues to innovate, promising a future where technology is both powerful and respectful of our planet. The great advantage of these system sis that microprocessors, though small in size, continue to have a vast impact on our society, propelling humanity towards new technological horizons.

Chapter 2 : Introduction To Instruction Set and Interrupts

Chapter 2 Introduction to Instruction Set and Interrupts

2.1 Introduction

The objective of this chapter is to bridge the gap from hardware to software, thereby transforming a processor into a computer. This transition is based on the programming model of the processor, turning it into a universal programmable machine. The programming model serves as the foundation for defining the first level of programming language, known as assembly language. When the processor execute a program, which is a set of instructions stored in the main memory, each instruction is sequentially read from memory, transferred to the Control Unit UC to be decoded, and then executed. To describe the operations executed by a processor in order to approach a programming model using an instruction set, we will closely interrest to the 6809 processor of MOTOROLA.

2.2 Instruction Set

An ***instruction set*** is the **list** of different machine language ***instructions of a computer***. In other terms, the processor is programmable through its instruction set or the set of commands it can execute. These instructions generally correspond to very basic operations, as we have discussed when describing an arithmetic and logic unit, [1], [7].

2.2.1 Program execution

For a program written in a high-level language to become executable, it needs to be translated into the machine language of the microprocessor to be executed. A microprocessor is capable of executing instructions in machine language (microcode), written by the programmer. These instructions stored in numeric form in memory are executed one by one using its internal components: the control and command unit (CCU) and the Arithmetic and Logic Unit (ALU). The execution procedure of a program within the computer is as follows:

1. The CCU extracts an instruction from memory.
2. It analyses the instruction.
3. Searches in memory for the data involved in the instruction.
4. Triggers the appropriate operation on the ALU or I/O.
5. Stores the result in memory if necessary.

2.2.2 CISC and RISC instruction set

There are multiple approaches to the design of a microprocessor and its instruction set. These approaches significantly impact every aspect of microprocessor architecture design. The primary approaches utilised to date are **CISC** and **RISC,** [4].

• **CISC** (Complex Instruction Set Computer) : Complex instruction set with variable instruction length. Due to the potential length and complexity of instructions, few registers are required. • Example: x86 (8086, Pentium)

• **RISC** (Reduced Instruction Set Computer) : Simple instruction set with a fixed length (e.g., 4 bytes). Multiple registers are required to execute complex tasks. • Example: PowerPC, ARM

2.2.3 Instruction Format

An instruction consists of two parts: the operation code that defines a type of operation to be performed, and the parameters that specify the operands on which the operation is carried out, figure 4. In fact, the instruction refers to a command given to the processor enabling it to carry out a basic operation while a machine instruction is a binary string of ***p*** bits, primarily composed of two parts:

- ***The operation code field***

Consisting of **m** bits, it instructs the processor on the type of operation to perform (addition, reading a memory location, etc.). A **m**-bit operation code allows for defining $\mathbf{2^m}$ different operations for the machine. The number of allowed different operations for a machine defines the instruction set of the machine.

- ***The operand field***

Consisting of p – m bits, it indicates the nature of the data on which the operation designated by the operation code should be performed. The method of specifying an operand in an instruction can take various forms; this is referred to as operand addressing mode. Operands are also symbolically represented.

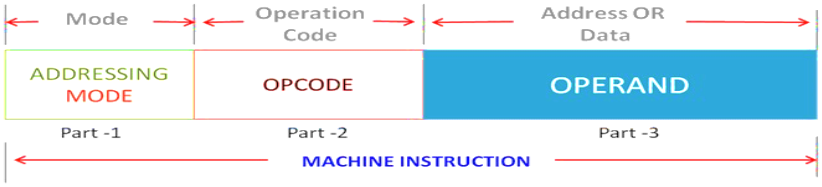


Figure 4: General format of an instruction machine.

2.2.4 Instructions categories

Instructions are divided into groups determined by their characteristics. In the following sections, we focus on the MOTOROLA 6809 processor to discuss the parts covered in this chapter with targeted examples for greater clarity. The instructions of the 6809 can be classified into six main categories: Thus, to be comprehensive an instruction set must include a sufficient number of instructions in each of the following categories, [1]:

• Data transfer instructions
• Data processing instructions
• Data pointer instructions
• Test and branch instructions
• Input-output instructions
• Control instructions

2.2.4.1 Data transfer instruction

They are used for loading from memory, saving to memory, performing transfers from registers to memory, register to register, memory to memory, etc. They Facilitate the movement of data between registers, memory, and peripherals. Data transfer instructions facilitate the transfer of data between these modules, [1]:

• The internal registers of the processor (6809);
• Internal registers and memory;
• Pointers.

2.2.4.1.1 Instructions for transferring to internal registers

The **EXG** and **TFR** instructions allow the transfer of data between internal registers. The transfer can only occur between registers of the same size.

EXG R1, R2 The contents of registers R1 and R2 are exchanged. Example: EXG A, DP or TFR R1, R2

TFR R1,R2 The content of register R1 is transferred to register R2. Example: TFR D, X

2.2.4.1.2 Instructions for transferring to internal registers and memory

The **LD** (LOAD) and **ST** (Store) instructions allow the transfer of data between internal registers and memory, Table 1, [1].

Table 1: Instructions for transferring to internal registers and memory

LD	The data in memory is loaded into the specified register.
ST	The content of the specified internal register in the instruction is transferred to the indicated memory location.

Example: **LDA $1000** Load accumulator A with the content of $1000

STA $2000 Store the content of Acc. A at memory address $200

2.2.4.1.3 Pointer Transfer Instructions

Instructions operating on pointers **U, S, X,** and **Y** allows manipulation of 16-bit data. This data typically represents addresses. Load Effective Address (in register): **LEA**. Operating on pointers **S** and **U** with the **PUSH** (push) and **PULL** (pull) instructions enable the transfer of internal registers into the memory stack, figure 5.

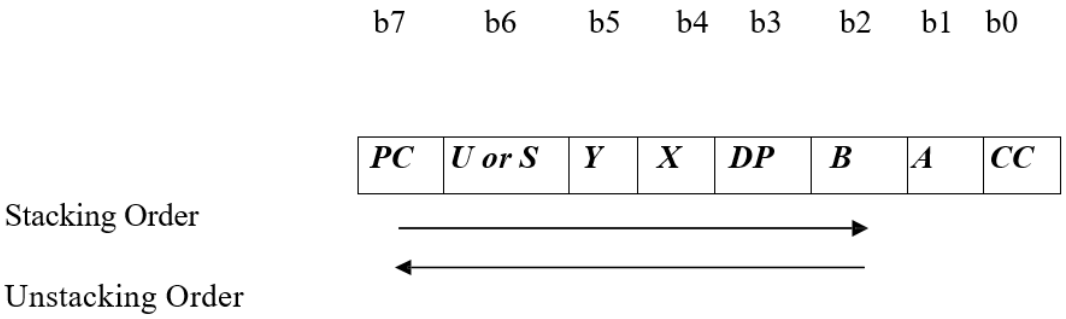


Figure 5: Order of stacking and unstacking of 6809 registers

Definition of a stack

A RAM memory area managed by pointers that allow for rapid data transfer into memory slots according to a well-established protocol. The registers to be pushed/pulled are indicated in the byte (post-byte) immediately following the operation code of the PUSH/PULL instruction. Each bit of the post-byte indicates an internal register, see Figure 2.2.When one of these bits is set to 1, the corresponding register is pushed/pulled. The stack pointer specified in the instruction cannot be pushed/pulled. Each time a byte is placed on the stack, the stack pointer is decremented by 1. For a 16-bit register, the low byte is pushed onto the stack first. Unstacking is identical except that it increments the stack pointer.

Table 2: Definition of a stack

PUL	*Unstacking of (s) register in the stack.*
PSH	*Stacking of (s) register in the stack.*

2.2.4.2 Data Processing Instructions

Data processing instructions can be classified into four categories:

- Arithmetic instructions;
- Logical instructions;
- Shift and rotate instructions;
- Increment/decrement, clear, complement instructions.

2.2.4.3 Arithmetic instructions

These groups of instructions involve the circuits of the ALU. They execute mathematical and logical operations such as addition, subtraction, multiplication, and sometimes division. The arithmetic instructions are listed in Table 3 below:

Table 3: Arithmetic instructions

ADD	Addition of memory content to an accumulator
ADC	Addition of memory content to an accumulator with carry
ABX	Addition of accumulator B to X
DAA	Decimal adjustment of accumulator A
MUL	Multiplication of A by B (unsigned)
SUB	Subtraction of memory content from an accumulator
SBC	Subtraction of memory content from an accumulator with carry
SEX	Sign extension from accumulator B to accumulator A

2.2.4.4 Logical instructions

The AND instruction is useful for setting to 0 or masking one or more bits in a word. The XOR instruction is useful for comparisons. It can also be used to complement a word (EORA #$FF). Logical instructions are listed in Table 4 below:

Table 4: Logical instructions

AND	The AND instruction is useful for setting bits to 0 or masking one or more bits within a word.
EOR	The XOR instruction is useful for comparisons.
OR	It can also be used to complement a word (EORA #$FF).

2.2.4.5 Shift and rotate instructions

Firstly, it is important to distinguish between a shift operation and a rotate operation. In a shift operation, all bits are shifted one position to the right or to the left. The bit that exits the register goes into the carry bit C; the entering bit is a zero. In a rotate operation, the bit entering the register is the one that comes from the carry bit C, Table 5.

Example: **RORA ; ROLB ; LSRA ; LSRB**

Table 5: Shift and rotate instructions

ASR	Right arithmetic shift. Bits are shifted to the right. b0 is transferred to C and b7 remains unchanged.
LSL or ASL	Logical or arithmetic left shift. Bits are shifted to the left. b7 is transferred to C and b0 is set to 0.
LSR	Logical right shift. Bits are shifted to the right. b0 is transferred to C and b7 is set to 0.
ROL	Rotation à gauche Bits undergo a rotation to the left. b7 is transferred to C and its original value is transferred to b0.
ROR	Rotation à droite[2] Bits undergo a rotation to the right. b0 is transferred to C and its original value is transferred to b7

2.2.4.6 Increment/decrement, clear, complement instruction

These instructions are listed in Table 6 below:

Table 6: Increment/decrement, clear, complement instruction

CLR	Resetting memory content or accumulator to 0
DEC	Decrementing memory content or accumulator
INC	Incrementing memory content or accumulator
COM	One's complement of memory content or accumulator
NEG	Two's complement of memory content or accumulator
NOP	No operation. Program counter incremented.

2.2.4.7 Test and Branch Instructions

Test and branch instructions can be classified into three categories:

- Test and comparison instructions;
- Test and unconditionnal branch instructions;
- Test and conditionnal branch instructions.

2.2.4.8 Test and comparison instructions

These instructions are used to perform bit status tests and comparisons to make decisions during program execution based on the value of these indicators. Only the status register is modified; neither the specified register nor the operand in memory is changed. No branching is performed. These instructions are listed in Table 7 below:

Table 7: Test and comparison instructions

BIT	Bits tested: **BITA M ; BITB M** The specified accumulator (A or B) and the operand in memory undergo a logical AND operation.
CMP	Comparison of memory content with an accumulator **CMPA, CMPB**
TST	Test of memory content or an accumulator **TSTA ; TSTB ; TST M**

2.2.4.9 Test and unconditional branch instructions

These instructions cause an unconditional change of execution sequence to a new location. The instructions JSR/RTS use the stack to create reusable routines. A subroutine is a program fragment that lives in user space, performs a well-defined task.

It is invoked by another user program and returns control to the calling program when finished.

Example: CALL and RET

JSR pushes the PC onto the stack and then jumps to the address of the requested routine. RTS automatically performs the return jump by popping the PC, as resumed in Table 8 below :

Table 8: Test and unconditional branch instructions

BRA, BRN	Relative Jump, 1 or 2 bytes
JMP	Absolute Jump, 2 bytes

2.2.4.10 Test and conditional branch instructions

Compare instruction is specifically provided, which is similar to a subtract instruction except the result is not stored anywhere, but flags are set according to the result. A conditional branch instruction is used to examine the values stored in the condition code register to determine whether the specific condition exists and to branch if it does.

These instructions perform tests on 4 indicators of the status register (C, N, Z, V) in order to perform branching during program execution based on the value of these indicators. Two types of branching exist:

- Short branch: displacement between -128 and +127
- Long branch: displacement between -32768 and +32767. Instructions using a long branch have an 'L' preceding their mnemonics. These instructions are listed in Table 9 below:

Table 9: Test and conditionnal branch instructions

BCC ou BHS	Branch if no carry : ***BCC N ; LBCC NN*** ***if C=0, then : PC=PC + N or PC=PC + NN***
BCS ou BLO	Branch if carry : ***BCS N ; LBCS NN***
BEQ	Branch if equal to zero
BNE	Branch if not equal to zero
BGE	Branch if greater than or equal to zero (signed) ***PC= PC + N or PC=PC+ NN***
BLT	Branch if less than (signed
BGT	Branch if greater than (signed)

2.2.4.11 Interrupt and halting Instructions

Interrupt is a mechanism by which an I/O or an instruction can suspend the normal execution of processor and get itself serviced, executed by the instructions in Table 10 below:

Table 10: Interrupt and halting Instructions

NOP	no operation. It cause no change in the processor state other than an advancement of the program counter. It can be used to synchronize timing
HALT	brings the processor to an orderly halt, remaining in an idle state until restarted by interrupt, trace, reset or external action.
RESET	reset the processor. This may include any or all setting registers to an initial value or setting program counter to standard starting location.
INT	It is level triggered and maskable interrupt. It has the lowest priority. It can be disabled by resetting the processor

2.3 Mnemonic Code

2.3.1 Introduction

Developing a program in assembly involves translating an algorithm with verbs that come from the instruction set and whose nouns are the variables or constants mapped in memory or in registers. In fact, every physical machine architecture corresponds to a symbolic form of the machine language associated with the processor; [1].

Each instruction in ***assembly langage*** represents a different ***machine code***, and each microprocessor may have a different assembler. An assembly language instruction is composed of thses three fields:

- A label field: not mandatory, corresponding to the address of the machine instruction.
- An operation code field: corresponding to the binary operation code string of the machine instruction.
- An operand field: which may effectively contain multiple operands separated by commas, corresponding to registers, memory words, or immediate values appearing in machine instructions.

2.3.2 The assembly

The assembly language is linked to the microprocessor, each processor has its own set of instruction. It represents the closest langage to ***machine language*** and is generally composed of rudimentary instructions known as ***mnemonics***.

It is composed of primarily data transfer operations between registers and the external components of the microprocessor (memory or peripherals), as well as arithmetic or logical operations. So, the ***symbolic assembly*** language is an equivalent langage used to avoid dealing with machine language, which is difficult for humans to manipulate. The operation codes of a

machine instruction are represented by abbreviations, indicating the operations. Programming in machine language or assembly language requires a good understanding of the microprocessor architecture.

2.3.3 Advantages of assembly

Assembly language can be considered as s a small compiler.This language relies on the abbreviation of English terms for each instruction. It great advantages is probably thet it minimizes code better control over hardware and is well suited for writing input/output routines.

Examples : The clear function **(CLEAR)** is written : **CLR,** the (**LOAD)** function is **LD,** the (**STORE)** function is **ST**,etc.

Concretely the reasons for choosing a less evolved language are twofold :

Space savings:

The evolved language requires many more machine instructions to perform the same task**.** **Example**: a simple instruction like WRITELN may involve hundreds of instructions. Thus, more memory space is required.

Time savings

More basic instructions imply a slower execution time unless a powerful (fast) microprocessor with a large memory field is available.

2.3.4 Assembly Language Syntax

Each instruction line of the program written in assembly langage has a mnemonic of four parts, where each part is separated by a blank (space).

Illustration of instruction mnemonic

Part 1	**Part 2**	**Part 3**	**Part 4**
Label	**Operation Code**	**Operand Code**	**Comment**

Example

Loop LDA $1500 ; resetting the accumulator content

Part 1 : Label

This field is not mandatory. A label is a symbol of up to 6 alphanumeric characters, starting with a letter of the alphabet. In case the label field is empty, at least one space must precede the next field.

If the 1st character of the line is an *, then the line is considered as a comment. The role of the label is to allow the position of an instruction in the program to be identified.

Part 2 : Operation field

It contains either a mnemonic code of the operation to be performed or an assembly directive.

Part 3 : Operand field

It complements the operation field and contains the "data" necessary for the execution of the instruction. Its syntax is varied and depends on the addressing mode assigned to the instruction.

Part 4 : Comment

Optional, it allows the program to be documented. All writing characters available in the editor can be used.

We can also find labels (names) symbols (register names) numbers expressions: Combination of the above 3 elements. Numbers can be represented in different bases: decimal, octal (@), hexadecimal ($), and binary (%). The base identifier is specified using a suffix or prefix in the case of certain assemblers (H). Without any particular indication, a number is interpreted as decimal.

2.4 Addressing Modes

By addressing mode, we refer to the path that the central processing unit (CPU) must take to access the operand. It specifies how the address field of the instruction is used to determine the operand. The addressing mode is indicated within the instruction in the operation code or in a separate field reserved for this purpose, called the addressing condition. The effective address corresponds to the final address sent to the RAM after transformations of the content of the address portion of the instruction. It should be noted that the power of a microprocessor depends not only on its instruction set but also on its addressing modes, [1], [4].

2.4.1 Addressing Modes of the 6809.

The 6809 microprocessor has 59 basic instructions. Combined with the set of addressing modes (9 in total), they provide 1464 different opcode combinations. (For the 6800 or 6802, there were 72 basic instructions and 193 opcodes.)

The addressing modes are:

- Inherent or Implicit addressing
- Immediate addressing
- Extended addressing
- Indirect Extended addressing
- Direct addressing

- Register addressing
- Indexed Direct addressing
- Indexed Indirect addressing
- Relative addressing

By generating signals on the address bus, the microprocessor can address various memory circuits and interfaces connected to it through buses to access their contents. This access is reflected in an addressing operation. This operation can be performed in several ways thanks to the presence of different addressing modes.

2.4.1.1 Inherent or Implicit Addressing

Inherent addressing is used by instructions that act only on the microprocessor's internal registers. Here, the instruction's opcode contains all the necessary addressing information (source address and/or destination address).

Examples: ABX, ASL, ASR, CLR, INC

2.4.1.2 Register Addressing

The opcode is immediately followed in memory by a byte that defines a register or the register set to be used by the instruction. This byte is called the post-byte. The table below shows the coding of this post-byte:

Example: TFR X, Y (transfer from X to Y). Addr Ad

2.4.1.3 Immediate Addressing

The data is immediately after the opcode of the instruction. The data exists in the form of 1 or 2 bytes. (the opcode is immediately followed in memory by the data on which the operation is performed). This addressing mode concerns all internal registers except the DP.

Examples: LDA #$35, LDY #$1997,

code	Registre
0000	D
0001	X
0010	Y
0011	U
0100	S
0101	PC
1000	A
1001	B
1010	CCR
1011	DP

2.4.1.4 Direct Addressing

The location of the action is expressed by the effective address. The opcode indicates the low part of this address. The high part of the address is provided by the content of the Direct Page Register (DP). The advantage of this mode is that it requires less memory space (1 byte, hence reduced memory size), therefore the instruction execution is faster. This mode is useful in real-time multitasking operating systems, where each task is allocated a page, because with this mode, memory is divided into 256 pages of 256 bytes each.

Examples: LDA $97: Load accumulator A with the content whose address is formed by [DP] and the operand.

LDY $97: Load register Y with the content on 16 bits whose addresses are [DP] and low part and low part+1.

2.4.1.5 Extended (Direct) Addressing

The location of the action is expressed by the effective address. The contents of the two bytes immediately following the opcode specified represent the address (16 bits) of the data. The instruction occupies 3 to 4 bytes.

Examples: LDA $1997: Load accumulator A with the content at address 1997.

LDY $1997: Load register Y with the content at address 1995.

2.4.1.6 Extended Indirect Addressing

Identical to the extended addressing mode but the data is accessed through a specified intermediate address after the opcode. The two bytes following the opcode point to an address whose content represents the address of the desired data. The opcode consists of two bytes - the post-byte is always $9F.

Examples: LDA [$1997]: Load accumulator A with the content whose address is in 1997-1998.

LDY [$1997]: Load register Y with the content whose address (high part) is at $1997.

2.4.1.7 Indexed Addressing

There are two possibilities with this addressing mode: indexed direct and indexed indirect. In this mode, the pointer registers (X, Y, U, S, and PC) are used to calculate the effective address of the desired data. There are 5 types of indexed addressing.

- Indexed addressing with zero displacement
- Indexed addressing with constant displacement (non-zero).
- Indexed addressing with accumulator displacement
- Indexed addressing with auto-increment/decrement
- Indexed addressing relative to the Program Counter (PC)

The byte following the opcode (the post-byte) specifies: the nature of the indexing, the type of addressing (direct or indirect), and the pointer register used. (the attached table shows the format of this post-byte.)

Formation of the post-byte :

b7=1,

b4: indicator if indirect or no,

b4 = 1: indirect mode

b4 = 0: direct mode,

b7=0,

b4 represents the sign,

b5 and b6 represent the registers concerned,

b0, b1, b2, and b3 indicate the addressing mode.

- **Indexed addressing with zero displacement**

The pointer register contains the effective address (E.A.) of the data. A prior loading of the register is imperative.

Example: **LDA 0, X** or **LDA, X**

- ***Indexed addressing with constant displacement***

The effective address of the data is the sum of the displacement (constant in two's complement) and the content of the named register taken as a base. E.A. = [Named Register] + displacement expression. The content of the register is not modified.

Syntax: One opcode + one post-byte + one operand.

Three forms are possible depending on the expression of the value of the constant following the post-byte: the displacement is expressed on 4 bits + one sign bit, the displacement is in the range [-1610 to 1510] or the post-byte is sufficient.

Example: **LDA -2, X** Code A6 1E, i.e., 0 00 1 1110; The displacement is expressed on 7 bits + one sign bit, i.e., it is in the range [-12810 to 12710]. An additional byte after the post-byte is necessary.

LDA 53, X Code A6 88, i.e., 1 00 0 1000; The displacement is expressed on 15 bits + one sign bit. The displacement is in the rang [-3276810 to +327]

LDA $997, X Code A6 89 soit 1 00 0 1001 09 97

- ***Indexed addressing with accumulator displacement***

This mode is similar to the previous mode except that the displacement value (expressed in two's complement) is stored in an accumulator to be added to the content of the named pointer register to form the effective address of the data. The contents of the accumulator and the pointer are not modified by this addition. It's the post-byte that specifies the accumulator used (no additional byte). The benefit is that the displacement value is calculated by the running program, depending on events.

Examples:

LDA B, X: Load register A with the content located at the address expressed by the sum of the contents of registers B and X.

LDX D, Y: Load register X with the content located at the address expressed by the sum of the contents of registers D and X, as well as D+1 and X.

• ***Indexed addressing with auto-increment/decrement***

The concerned register holds the address of the data. In addition to using it in indexed mode with zero displacement, it's possible to modify the content of the register according to the following mechanism:

1. Pre-decrementation:
2. The pointer is decremented by one or two before use.
3. Post-incrementation:
4. The pointer is incremented by one or two after use.

In all cases, the content of the register is modified. The interest of this addressing mode is that an increment/decrement allows managing 8-bit data tables (bytes). Two increments/decrements enable managing 16-bit data tables (words or addresses).

The aspects of pre-decrementation and post-incrementation are very useful for creating additional software stacks whose behavior is identical to that of the S and U stacks (hardware). This allows efficient management of memory blocks organized in 8-bit or 16-bit data.

Examples:

LDA 0,-X ; Opcode: A6 82 Binary: 1 00 0 0010 Load accumulator A with the content located at the address specified by the content of X decremented by 1.

LDY 0,--X ; Opcode: 10 AE 83 Binary: 1 00 0 0011 Load register Y with the content located at the address specified by the content of X decremented by 2.

LDA 0, X+ ; Opcode: A6 80 Binary: 1 00 0 0000 Load accumulator A with the content located at the address specified by the content of X. After loading, the content of X is incremented by 1.

LDD 0, X++ ; Opcode: EC 81 Binary: 1 00 0 0001 Load accumulator D with the content located at the address specified by the content of X. After loading, the content of X is incremented by two.

• ***Indexed Addressing Relative to Program Counter (PC)***

In this addressing mode, the effective address of the data is calculated relative to the program counter (PC). The displacement values are encoded in 8 or 16 bits in two's complement form.

Syntax: Opcode Destination (addr+4), PC.

A program that contains instructions expressed in extended or direct mode, referring to addresses located within its memory implantation area, cannot be executed if its object code is transferred to another memory area. In this case, the program is said to be non-translatable. This inconvenience can be avoided by using indexed addressing mode with relative displacement to the PC. The advantage of the PC-relative indexed mode is that if the entire program is loaded at another address, the same object code still allows the data to be found within the program body and thus achieve correct execution.

Examples:

LDA $FO, PC: Load accumulator A with the memory content at the address PC + $FO.

LDA [$1997, PC]

Application Example

I. Give which kind of addressing mode is concerned by these instructions**:**

LDA [0, X], LDA [$35, X], LDU [D, Y], ADDA [, U++], LDA [$1997, PC].

Solution:

1. Indexed indirect addressing with zero displacement:

- **LDA [0, X]:** Load A with the memory content at [X, X+1].

2. Indexed indirect addressing with constant displacement:

- **LDA [$35, X]:** Load A with the memory content at [X+$35, X +$36].

3. Indexed indirect addressing with accumulator displacement:

- **LDU [D, Y]:** Load register U with the memory content at [Y+D] and [Y+D+1]. The displacement, expressed in two's complement, is contained in A, B, or D.

4. Indexed indirect addressing with double auto-increment/decrement:

- **ADDA [, U++]:** Addition of the content of A and the memory content at [U] and [U+1], followed by two increments.

5. Indexed indirect addressing relative to the program counter:

- **LDA [$1997, PC]:** Load A with the value at the address PC + $1997. Displacements are coded on 8 or 16 bits in two's complement.

II. Upon execution of the following assembly code and considering the initial state of the memory and processor registers, for each line of the assembly routine, identify the addressing modes used. What contains the memory location at address 3000 and the register A ?

Address	Content	Register	Content
$4000	$20	B	$15
$4001	$08	D	$41
$4002	$A1	-	-
$3000	$00	A	$00

Assembly Programm : LOAD D # $4000 ; immediate mode
ADD B $4002, Extentended mode
LOAD A $4000, Extended mode
ADD D $4002, Extended mode
STORE D $3000, Extendedmode

Solution : [3000]= A=$C1 = ; [3001]= B = A9.

2.5 Interrupts

The processing of an interruption follows a well-established protocol that we will detail in this section. As soon as the pin corresponding to the interruption is set to a low level, or when the SWI instruction is encountered, the microprocessor initiates an “implicit procedure”, programmed into the silicon, whose respective flowcharts are represented further below.

It is noted that each flowchart ends with loading the PC with the value contained in the vector of the requested interruption. This causes a jump to the corresponding address.

In a simple and fixed system, where the interruption request can only come from a single source, and where only one interruption program is considered, it is possible to directly implement the beginning of this program at the address contained in the interruption vector. The exit of the implicit procedure will directly lead to the execution of the program [5],[6].

In the general case where multiple “users” can request the interruption and where at least one “user” wants to be able to choose between several different interruption programs, the “implicit procedure” must first lead to an interruption management program.

The interruption management program, for example, IRQ, must ensure the following operations :

- Firstly, identify the requester.
- Enforce a hierarchy among the requesters when there are simultaneous calls. Priority between simultaneous requests from different interruptions is partly achieved by the use of masks.
- Switch the microprocessor (by changing the PC) to the desired interruption program by the requester.
- requester.

2.6 The 6809 Interrupts Mode

To well understand how the processor operates in interrupt mode, we will detail how the microprocessor 6809 of MOTOROLA deals with an interrupt. The processssor procedure that allows the suspension of the execution of one program in favor of another, with the possibility of resuming the execution of the original program where it was suspended. One shoul distinguish two types of interruptions:

2.6.1 Hardware Interrupts

The microprocessor is capable of responding to external requests. Thus, it can process information in “real time”. For this purpose, the microprocessor is “connected” to the external world by lines on which an exchange of messages in the form of logical signals occurs in an indeterminate manner! Based on external requests, the microprocessor must change “state” according to the relative priorities of the ongoing operation and the requested one. Depending on the case, it interrupts or does not interrupt the “normal” execution of the program. If the request is accepted, the microprocessor must be able to quickly process this external request These input interrupts are active at a “low” level. The 6809 microprocessor has four lines, called “hardware interrupt” input lines, which are in order of priority, as shown in Table 11 below.

Table 11: Hardware Interrupts

RESET	Ré-initialisation du microprocesseur
NMI	Non Masquable Interrupt.
IRQ	Interrupt Request
FIRQ	Fast Interrupt Request.

2.6.2 Role of the Condition Code Register (CCR)

The **CCR** register of the 6809 microprocessor is of 8 bits, it is also known as **STATUS** or **FLAG** register.

NMI FIRQ IRQ

E	F		I				

CCR REGISTER

- ***Function of flag I***

Le flag I est un masque relatif à l'interruption IRQ. Lorsque ce flag est mis à 1, le microprocesseur ne prend pas en compte les demandes d'interruption arrivant sur la ligne IRQ, as shown below. With this type of interruption, the total context of the microprocessor is saved on the S stack.

State of the F (FIRQ) flag	
F=1	Inhibited
F=0	Permitted

- ***Function of flag F***

Le flag F est un masque relatif à l'interruption FIRQ. Lorsque ce flag est mis à 1, le microprocesseur ne prend pas en compte les demandes d'interruption arrivant sur la ligne FIRQ, as shown below. With this type of interruption, the total context of the microprocessor is saved on the S stack.

State of the I (IRQ) flag	
I=1	Inhibited
I=0	Permitted

- ***Function of flag E***

This flag is set by the microprocessor to indicate the type of backup that should be performed (partial or complete). It serves somewhat as internal memory. It is systematically used (as a test) by the microprocessor during the RTI instruction to determine what action to take in order to return properly to the suspended program (number of registers to be popped).

Here, the partial context of the microprocessor (3 bytes) is saved on the stack. Only the contents of the PC and CCR registers are involved.

- ***Changing the state of the I and F flags***

I and F flags can be achieved using the following instructions : ANDCC and ORCC.On peut forcer l'état des deux bits à l'aide des instructions suivantes ANDCC d'une part et ORCC d'autre part, Table 12.

Table 12: Changing the state of the I and F flags

Instruction	Flag State
ANDCC #$EF	CCRb4=0
ORCC #$10	CCRb4=1
ANDCC #$BF	CCRb6=0
ORCC #$40	CCRb6=1

2.7 Software Interrupts

Apart from hardware interrupt requests, there are other causes of program interruption. These are "soft" interrupts. They are represented by full-fledged instructions that are placed within programs. They represent "deterministic" events. Context saving is complete. There are three different types of software interrupts, which are: SWI/SWI2/SWI3

2.8 Synchronization Interrupts

Finally, the microprocessor can wait for external events to synchronize its progress with the occurrence of these events. There is a complete "pre-save" of the context. For this purpose, the following instructions are used, Table 13:

Table 13: Synchronization Interrupts

CWAI	Interruption Wait
SYNC	Synchronization Wait
SWI/SWI2/SWI3	Software interrupts
RTI	Return from interrupt subroutine

Important Remarks:

The NMI, FIRQ, IRQ, and RESET lines, as well as the SWI instruction, automatically set the CCRb4 flag to 1 (IRQ masking). The NMI, FIRQ, and RESET lines, as well as the SWI instruction, automatically set the CCRb6 flag to 1 (FIRQ masking). However, there is no obstacle to resetting these flags to 0 if desired.

Application Example

Let's assume we're working on a very simple processor where the CPU has only three specialized registers (MODE register, Ordinal Counter CO, and Stack Pointer SP) and two general-purpose registers (R1 and R2). And let "MEP" denote the processor's status word, encompassing all registers for simplification : **MEP = hCO, MODE, SP, R1, R2i**

1- How does the processor behave in response to these interruptions ?

2-What should the interrupt administrator provide to enable effective processing of these interruptions ?

3- Give the essential steps to do this.

Solution :

1- It proposes an interrupt vector starting at address "vi," for instance. This vector must contain the addresses associated with each interrupt cause.

3- Two routines allowing an administrator to easily save and restore the value of CPU registers: Push the CO and MODE onto the stack, Increment the stack, Switch to master mode and then Branch to the interrupt vector to execute the interruption programm."

Push Programm :	**Pull Programm**
SP := SP - 1;	Load_mep(m:mep)
hm.CO,m.MODEi := mem[SP]	SP := m.SP;
m.SP := SP	mem[SP] := hm.CO,m.MODEi
m.R1 := R1	SP := SP+1
m.R2 := R2	R1 := m.R1, R2 := m.R2
END	RTI

2.9 Conclusion

In this chapter, we have observed that the objective of a computer's execution function is to execute a sequence of instructions on a set of data. A high-performance processor is one that has a well-designed instruction set that should encompass the diverse categories to enable flexible and efficient programming for various tasks. To achieve this, a problem is translated into a sequence of machine instructions characteristic of a processor capable of executing them using a set of electronic circuits. We have pointed out that the executable machine program must reside in main memory (central memory), and the microprocessor executes this program instruction by instruction based on concepts related to microcommands and sequencers. Furthermore, we have provided an overview of a method called pipelining, which enhances the performance of microprocessors.

Chapter 3 The Memories

Chapter 3 The Memories

3.1 Introduction

A memory is composed of a set of identical storage cells or units intended to store information. Each storage unit is identified by a number called its address. For a memory containing N cells, the addresses range from 0 to N-1. Thus, there is a physical system ensuring the correspondence between each storage unit in the memory and its address. The memory control unit (or controller) performs this routing task.

Memory must also be able to communicate with other parts of the computer, especially the central processing unit (CPU) and input/output units. This communication is ensured by connecting components formed by lines (or buses), gates, and registers. The lines are essentially "highways" on which bits of information, forming addresses and data, travel. Gates are used to control the synchronization of processing by allowing electrical impulses to pass or not pass, depending on whether they are open or closed. As for registers, they are actually very small and fast memories used to temporarily store certain information related to the work to be done.

Essentially, the role of memory is to store information and retrieve it when needed. To do so, the information stored in memory must be well identified so that it can be retrieved as needed. There are several different ways to organize memory. The size of the information units it contains can also vary. The choices made by manufacturers depend on what is expected from the computer being built. The characteristics of memories depend, among other things, on the desired performance, the desired capacity, and the budget available, as their cost is generally prohibitive, despite the downward trend.

3.2 Technology of Memory

Until the mid-1970s, main memories were made up of magnetic cores. Since then, semiconductor memories have become dominant, and various semiconductor memory technologies have been developed to build both RAM and ROM. Some research is being done to develop processes using optical perforation or the Jacobson effect, but for now, semiconductor memories do not seem to be truly threatened with disappearance; rather, the latest technological advancements have focused on increasing circuit integration more and more, [2].

3.3 Mechanics

The abacus is a primary form of mechanical digital information storage. Since the 17th century, mechanical calculators have been built with gears that store at least one variable in memory. Historical mass storage media include tapes and punched cards, used since the 18th century. A probe activated the mechanism when it encountered a hole. In telegraphy, the probe opens or closes an electrical circuit. Several probes in parallel allow for the recording of a Baudot code. In the 20th century, optical reading improved the longevity of cards and reading speed. Magnetic and electronic recordings definitively replaced cards and tapes.

3.4 Electromechanics

Electromechanical systems based on relays and rotary selectors were among the first reliable systems designed to store information. Relays record a bit, while rotary selectors record a numeric value, often ranging from 0 to 9, sometimes from 0 to 7 (octal), 11 (time format), 15 (hexadecimal), 23 (time format), or 99. Mechanisms similar to those of mechanical calculators enable the manipulation of information. These systems preside over telephone switching systems.

3.4.1 Magnetic Support

Many information storage systems use magnetic media: hard drives, floppy disks, magnetic tapes, etc. Due to the decreasing prices of electronic systems that form the basis of USB drives and SSDs, magnetic media are now primarily used for storing large volumes of data where access speed is not critical: backups, media files, etc. Bubble memories also existed, which, after raising great hopes, were only briefly commercialized.

3.4.2 Optical Support

Initially employed solely for reading punched cards and perforated tapes, optical media also became used for digital sound in films at the end of the 20th century. As long as only photography allowed the use of light for writing, optical support remained less widespread. The laser read-write disc in the 1990s enabled a more general use of optical support for computing. Since that time, laser-based media have become wide spread: CD-ROMs, DVDs, Blu-rays, and proprietary formats.

3.4.3 Magnetic Core Memories

The magnetic "core" is actually a small ring, made of ferrite, which can take two states: being magnetized in one direction or the other, depending on the current flow applied to it. The cores, with a diameter of a fraction of millimeters, are arranged in rows and columns to form a plane where each core represents a bit. Several planes can be stacked, so that the cores with the same coordinates (row, column) on each plane form words. Each core is traversed by three wires: a row decoder and a column decoder that allow locating a core (and thus a bit), and finally a sensing wire whose function is to detect the direction of magnetization and modify it if necessary. Simply applying a certain current to a core changes the direction of magnetization, and thus its state.

The read operation involves locating the core corresponding to the requested address and using the sensing wire to determine if the bit contains a 0 or a 1. Since this operation is performed by applying a charge to the row and column wires, each read operation results in resetting the content of the read cell to zero. Therefore, the content of the read memory must be read and rewritten for each read operation. To perform a write operation, two steps are also required: first, resetting the cell to zero, and then magnetizing the cell.

3.4.4 Semiconductor Memories

A semiconductor is a material whose electrical conductivity lies between that of insulators and metals. T Semiconductor memories, like integrated circuit technologies, use silicon as their starting material. They represent a privileged domain where advances in high integration are immediately felt. They have been the first applications of complex integrated circuits (LSI - Large Scale Integration, 1,000 gates per integrated circuit) and will continue to be preferred vehicles for demonstrating the feasibility of highly complex integrated circuits and as driving components of technology.he most commonly used are germanium and, especially, silicon.

3.4.5 RAM, ROM

In the last section we have seen that memories can be classified into three main categories according to the technology used:

- Magnetic memory (**hard disk, floppy disk**, etc.): less fast but stores a very large volume of information.

- Optical memory (**DVD, CD-ROM,** etc.), and finally theSemiconductor memory (**main memory, RAM, ROM, PROM**, etc.): very fast but small in size. For now, we will focus in that follows principally on ***main memory***.

As mentioned earlier, several types of memory make up a computer: main memory, registers, and peripheral memories. This leads us to make the distinction between two other types of memory: **Read-Only Memory (ROM)** and **Random Access Memory (RAM)**.

3.4.6 Read-Only Memory (ROM)

A ROM is primarily an integrated circuit. This means that it is a set of interconnected circuits designed to perform a function. In this case, the function is to store a certain amount of information and retrieve it as needed. The ROM package thus contains the various information units (bits) and the circuits necessary to "deliver" the required information when requested and provided with the address of the desired information cell. The inputs to the circuit are therefore the different bits containing the address of the sought information, and the outputs are the different bits containing the data. In addition, there is an input that signals the reading order. The content of a ROM is fixed at construction and cannot be changed. This means that upon construction, we know the desired outputs for each input.

3.4.7 Programmable Read-Only Memory (PROM)

However, as it is costly to build ROM memories for very specific applications, there are variants of ROM memories that can be programmed by the user. These are PROMs (Programmable ROMs), which are essentially "blank" ROMs containing all possible connections and on which a special device, the PROM programmer, allows the destruction of certain internal fuses. This involves eliminating certain connections to retain only the desired ones.

3.4.8 *Electrically Programmable Read-Only Memory (EPROM)*

Also called erasable PROM, it may be advantageous to be able to modify a PROM. However, the fuses destroyed during the programming of a PROM cannot be recreated. This is why EPROMs were developed. Positioned somewhat halfway between RAM and ROM, the EPROM (Erasable PROM) is a device whose content can be erased when subjected to ultraviolet radiation, thus allowing for reprogramming.

3.5 Electrically Erasable Programmable Read-Only Memory (EEPROM)

Also called ***Electrically Erasable PROM,*** this category of ROM memories has the advantage of being electrically erasable, which is simpler than using ultraviolet radiation as is the case for EPROMs, [2].

3.5.1 Random Access Memory (RAM)

Random Access Memory (RAM), also known as main memory or volatile memory, is also comprised of integrated circuits. Since the content of each cell can be read from or written to, it must be able to vary. Unlike ROM, where the output corresponding to a given series of address bits input is fixed upon construction, in RAM, it can change depending on the program being used and the data being supplied to it. The operations differ depending on whether it is a read or a write operation. In the case of a read, the bits constituting the address are "received" by an address decoder which locates the desired cell. Depending on whether this cell contains a 0 or a 1, the data is outputted either on the 0 read/write line or on the 1 read/write line.

3.5.2 Static RAM (SRAM)

Two types of random access memory (RAM) are distinguished: static and dynamic. In static random access memory (SRAM), a bistable circuit composed of two transistors is used to represent a memory element. Without external intervention, the bistable circuit maintains an electrical state representing binary information. In each of the states, one of the two transistors is saturated and the other is blocked, and only the application of an electrical voltage can switch the bistable circuit from one state to another. The fact that a stable state is maintained without external intervention is why it is called static memory.

3.5.3 Dynamic RAM (DRAM)

On the other hand, so-called "dynamic" memories are based on the use of a capacitor that maintains an electrical voltage of 5V or 0V between its electrodes, which correspond to states 1 and 0. However, the capacitor discharges and thus requires periodic refreshing of the memory. This means that reading and rewriting the memory must be done regularly. Therefore, regular external intervention, called refreshing, is required to maintain the state of dynamic memory. Despite this disadvantage, the simplicity of dynamic RAM allows for more of them to be integrated onto a single silicon chip compared to static RAM.

It is precisely this characteristic that contributed to the widespread use of dynamic RAM. For instance, IBM introduced them in its IBM PC microcomputers.

3.5.4 Refresh Techniques

Memory refreshing is a process that involves periodically reading the information from computer memory and immediately rewriting it without any modifications, aiming to prevent the loss of this information.

There are memories that do not require refreshing: SRAM (static random access memory). Memory refreshing is required in DRAM (dynamic random access memory), the most widely used type of random access memory, and refreshing is one of the main features of this memory type. In a DRAM (dynamic random access memory), a data bit is encoded by the presence or absence of an electric charge on a small capacitor. Over time, the electric charges in the capacitors disperse, and without refreshing, the data would be lost. To prevent this loss, a circuit periodically reads each data bit and rewrites it, thus restoring the capacitor charge to its initial level. Each memory refresh operation regenerates a new area of memory, thus traversing all memory zones.

3.5.5 Refresh functioning

Refreshing does not involve the normal read and write operations used to access memory data, but rather ***specialized cycles*** called refresh cycles generated by ***memory circuits*** and interleaved between memory read and write accesses. A refresh cycle is needed and is similar to the read cycle but executes more quickly. As long as the computer is operational, each memory cell must be repeatedly refreshed at intervals specified by the manufacturer, typically on the order of milliseconds. A refresh circuit should perform a refresh cycle on each memory row within a refresh time interval to ensure that no data is lost.

3.5.6 Refresh Circuits

Many types of refresh circuits have been used. In modern systems, refreshing is managed by circuits located in the memory controller, or increasingly, by circuits located on the memory chip itself. The refresh circuit includes a refresh counter that holds the address of the row to be refreshed and an adder that increments the counter to refresh all rows in turn. The counter can be in the memory controller or on the memory chip itself. The fraction of time that the memory spends on refresh operations, known as the refresh overhead, can be calculated as follows:

refresh overhead = time required for refresh (in ms) / maximum interval between refreshes (in ms)

3.6 Memory Characteristics

We will focus on ***main memory*** to present the memory characteristics to these reasons:

- Main memory (MM) represents the workspace of the computer (processor).
- It is the primary storage organ for information used by the processor.
- In a machine (computer/calculator), to execute a program, it must be loaded (copied) into main memory.
- The access time to main memory and its capacity are two factors that influence the execution time of a program (machine performance).

3.7 The capacity of a memory

The capacity (size) of a memory is the number (quantity) of information that can be stored in that memory.

• Capacity can be expressed in:

- Bit: A bit is the fundamental element for representing information.
- Byte: 1 Byte = 8 bits
- Kilobyte (KB): 1 Kilobyte (KB) = 1024 bytes = 210 bytes
- Megabyte (MB): 1 Megabyte (MB) = 1024 KB = 220 bytes
- Gigabyte (GB): 1 Gigabyte (GB) = 1024 MB = 230 bytes
- Terabyte (TB): 1 Terabyte (TB) = 1024 GB = 240 bytes

3.8 Access time

This is the time required to perform a read or write operation. For example, for a read operation, the access time is the time that elapses between the request for reading and the

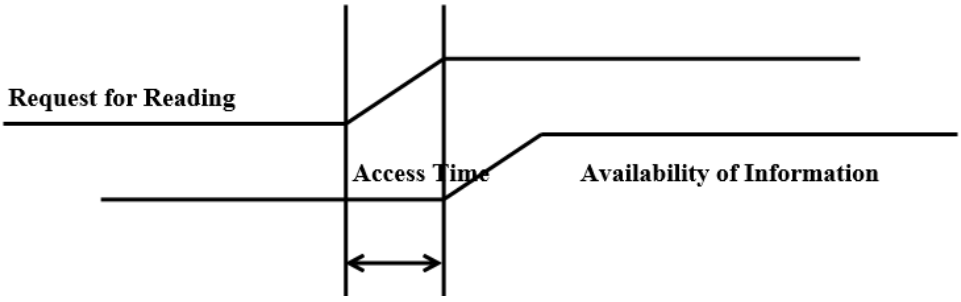


Figure 6 : Memory Acsses Time.

availability of the information. Access time is an important criterion for determining the performance of a memory as well as the performance of a machine, figure 6.

3.9 Addressing Modes

Main memory can be seen as a large vector (array) of words or bytes. A memory word contains multiple memory cells and stores information on n bits. A memory cell stores only one bit while each word has its own address. An address is in fact a unique number that allows access to a memory word. Addresses are sequential (consecutive). The size of the address (the number of bits) depends on the capacity of the memory, shown in figure 7.

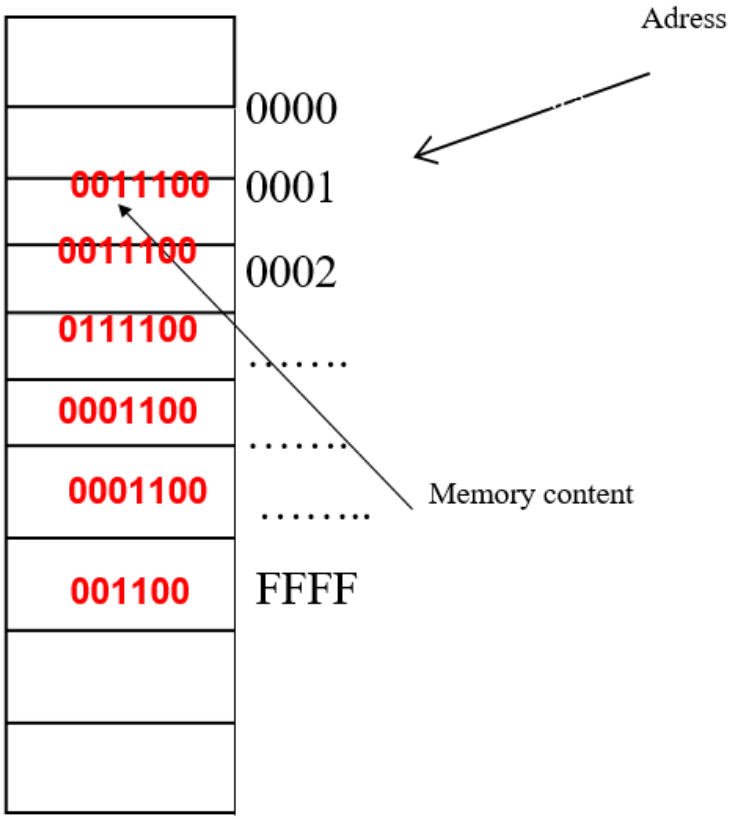


Figure 7 : Memory Logic.

When an address is loaded into the RAM register, the decoder receives the same information as that of the RAM. At the output of the decoder, we will have only one active output. This output will allow us to select a single memory word.

Let k be the size of the address bus (size of the RAM register) and n be the size of the data bus (size of the RIM register or the size of a memory word). The capacity of main memory can be expressed either in the number of memory words or in bits (bytes, kilobytes, ...).

- Capacity = 2^k Memory Words
- Capacity = 2^k * n Bit

Application Example

1. In a memory, the size of the address bus K=14 and the size of the data bus n=4. Calculate the capacity of this memory.

Solution: C = 2^{14} = 16384 words of 4 bits C = 2^{14} * 4 = 65536 bits = 8192 bytes = 8 KB

2.How to read information from main memory?

Solution: the following operations must be performed: show figure 8.

- Load the address of the word to be read into the RAM register.
- Initiate the read command (R/W=1).
- The information becomes available in the RIM register after a certain time (access time).

3. How to write information to main memory?

Solution: the following operations must be performed:

- Load the address of the word where the writing will occur into the RAM register.
- Place the information to be written into the RIM register.
- Initiate the write command to transfer the content of the RIM into memory.

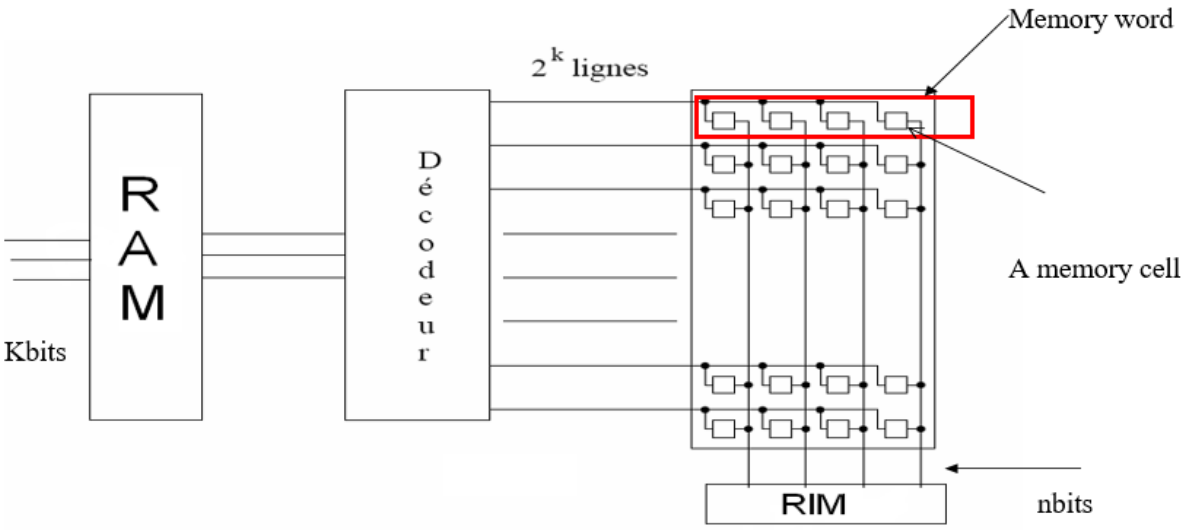


Figure 8: How to select a memory word.

4. Let M be a memory with capacity C, where m is the number of words and n is the size of a word. • Let M' be a module with capacity C', where m' is the number of words and n' is the size of a word. We assume that C > C' (m >= m', n >= n').

What is the number of modules M' needed to realize memory M?. Building a memory of 1KB (where the word size is 8 bits) using modules of size 256 words of 8 bits?

Solution :

To determine the number of necessary modules, we need to calculate the following two factors:

- P = m/m'
- Q = n/n'

P: determines the number of modules, M' required to achieve the number of words in memory, M (row extension), Q: determines the number of modules. M' required to achieve the word size of memory M (word extension or column extension). P.Q gives the total number of modules required to realize memory M. To select the modules, we use the most significant address bits. If P is the row extension factor, then we take k bits such that P=2^k. The remaining address bits are used to select a word within a module.

(m, n) = (1024, 8) → the size of the address bus is 10 bits A9...A0, the size of the data bus is 8 bits D7...D0.

(m', n') = (256, 8) → the size of the address bus is 8 bits (A7'...A0'), the size of the data bus is 8 bits (D7'...D0'). Calculate the two expansion factors, rows and columns:

P = m/m' = 1024/256 = 4 (row extension)

Q = n/n' = 8/8 = 1 (column extension).

The total number of modules is P*Q = 4.

5. We want to create a memory of 1KB (where the word size is 8 bits) using modules of size 256 words of 4 bits.

Solutio :

(m, n) = (1024, 8) → the size of the address bus is 10 bits (A9...A0), the size of the data bus is 8 bits (D7...D0).

(m', n') = (256, 4) → the size of the address bus is 8 bits (A7...A0), the size of the data bus is 4 bits (D3...D0).

• P = 1024/256 = 4 (row extension)

• Q = 8/4 = 2 (column extension)

• The total number of modules is P*Q = 8.

6.Complete the points with the appropriate word.

1. The processor consists of an Arithmetic Logic Unit (ALU) and a **Control** Unit.
2. Unlike a ROM, the content of RAM can be **modified.**
3. Unlike RAM, the content of a hard disk is stored **permanently.**
4. Tape, floppy disk, or hard disk are **magnetic storage** media, while CD, DVD, or Blu-ray are optical storage media.
5. The data bus, address bus, and control bus are collectively called the **Local** Bus.
6. Three memory modules with a capacity of 8 K octets are connected to a 16-bit address bus of an 8-bit processor. How many address lines are needed to address the

3.10 Conclusion

In this chapter, we have seen that due to cost and capacity considerations, memories differ according to their construction technologies and applications. Since the early days of computing, mechanical storage techniques, such as punched tapes, were widely used but then abandoned in favor of more convenient and faster mediums. Various memory technologies have emerged, and continuous improvement in manufacturing techniques has resulted in memories becoming smaller, less costly, consuming less energy, with ever-increasing capacity, and higher speed. The use of CMOS technology has been fundamental in producing components that are much less power-hungry. These components, combined with a tiny battery, have enabled the construction of non-volatile memories, used, for example, in smart cards, [2].

In conclusion, let us discuss the different forms of memories that have existed from the beginning to the present day :

- **Magnetic Tapes**

Used as high-capacity mass storage since 1950. DAT, DLT, or LTO are formats of magnetic tapes. These can only be read in a specific order and are commonly used for making backup copies of hard drives.

- **Floppy Disks**

Mass storage memories consisting of a flexible disk with magnetic surfaces protected by a plastic case. Floppy disks were widely used until the 1990s.

- **Optical Discs**

Optical storage media equipped with a reflective surface. Information is stored as microscopic pits and read by the reflection of a monochromatic beam. The first compact discs were built in 1980 and the first DVDs in 1995. Unlike hard disks, modifications to the stored information on these media are limited or impossible (read-only memory), so they are primarily used for long-term storage of computer data.

- **Integrated Circuits**

Most electronic memories are in the form of integrated circuits or are directly integrated into the integrated circuits of processors or controllers (e.g., assembly in stacked die).

- **RAM Modules**

Memory modules are standardized printed circuit boards, formats such as SIMM or DIMM are widely used in computers. These are typically fast volatile memories but of relatively low capacity.

- **Hard Drives**

Mass storage memories consisting of metal disks enclosed in a shielded casing. Information is stored on the magnetic surfaces of the disks. This is a type of high-capacity mass memory still widely used in 2009 in our computers. Innovative media are emerging (high-capacity flash media, known as Solid State Drives, or holographic media).

- **USB Flash Drive**

A case equipped with flash memory, an electronic component, and a plug compliant with the USB standard. The first USB flash drives were built in the early 2000s. USB drives can be used similarly to magnetic floppy disks as a data transfer medium, long-term storage, or even mass storage.

Chapter 4 The interfaces

Chapter 4 The interfaces

4.1 Introduction

The microprocessor does not only work with its accumulators, registers, and memory. It needs to communicate with the outside world. For this purpose, exchange units are used to facilitate dialogue between the microprocessor and peripherals. These exchanges are conducted through data, control, and address buses.

These exchange units are called ***interfaces*** and occupy a certain number of memory slots in the microprocessor's memory space. They are programmable to adapt to connected peripherals. Several types of interfaces exist on the market. In summary, there are two categories:

Passive interfaces: These units or couplers enable hardware interfacing between various peripherals and the data bus. The management of exchange signals falls upon the microprocessor. In cases where the number of transfers is low, the microprocessor handles this task quite quickly. However, the situation becomes critical when dealing with a complex device. Interfacing with this type of unit involves more extensive hardware and software development (resulting in longer microprocessor time). Once programmed, these units considerably lighten the workload of the microprocessor, allowing it to perform other tasks.

Active interfaces: For truly complex peripherals, intelligent exchange units are used. Here, the microprocessor no longer wastes time managing exchanges. They are entirely handled by the unit, which often includes its own microprocessor. These units typically manage simple or specialized couplers. Some of these intelligent controllers are more specialized than others to perform a specific and very complex task. For example, managing a graphical screen or a co-processor which relieves the microprocessor of all calculation tasks, can be considered as a specialized intelligent interface!

4.2 Serial Interface

The Asynchronous Serial Interface presented in this chapter is the A.C.I.A. 6850 of Motorola. The A.C.I.A (Asynchronous Communication Interface Adapter) is an integrated circuit that constitutes an adapter for asynchronous communication, figure 10. Belonging to the family of UARTs (Universal Asynchronous Receiver Transmitter), this programmable circuit allows for asynchronous serial communication according to the

START-STOP procedure through procedure widely used for low data rates of 50 to 19,200 bits per second, working speed of many devices). A word format is comprises between 5 and 8 bits, all preceded by a start bit and followed by 1 or 1.5 or 2 stop bits, [3]. The start bit is synchronised to a clock but the sequence of characters is asynchronous, figure 9.

4.3 The ACIA components

This integrated circuit includes

- an asynchronous data transmitter,
- an asynchronous data receiver,
- Modem control logic,
- separate clock inputs for transmission and reception.

Transmission and reception can operate simultaneously (full-duplex) and at different speeds.

This circuit includes four internal registers:

- A transmission register
- A reception register
- A control register
- A status register

4.4 Description of the ACIA circuit

As seen in figure 10, the ACIA has several circuit, in fact 24 pins in total. We present this circuit on two sides:

On the microprocessor side:

- **Data bus**: d0 ... d7 Facilitates data exchange between the microprocessor and the ACIA. When the device is not selected, these bidirectional lines are in high impedance
- **Control bus**:

  E line: Exchange activation signal.

  R/W line: Data transfer control line with the microprocessor.

  R/W = 1; only output buffers are activated (possible to read a register).

  R/W = 0; only input buffers are activated (possible to write to a register).

This line is also used as an additional line for addressing internal registers (see table below).Note the absence of the RESET line - This means that initialization is done by software.

- **Address bus**:

CS0, CS1, and /**CS2** (Chip Select): These 3 lines are connected via the decoder to the microprocessor's address bus to select the device, and the selection is validated when the combination is (110).

RS (Register select): This input allows selection of the internal registers (2 bytes of memory). It is used in conjunction with the R/W line so that one can choose a register from the 4 available.

IRQ (Interrupt request): Open-drain output line (no pull-up resistor) active at a low level and connected to the microprocessor's IRQ or NMI inputs or to the PIC (6828: priority interrupt control).

On the external side:

1- **Clock lines**

- **Txclk**: Transmission clock Serves as synchronization (reference) for data transmission on the Txdata line. The transmission shift register is synchronized on a falling edge of this signal.
- **Rxclk**: Reception clock Serves as synchronization for data reception on the Rxdata line. The reception shift register (load and shift), specific to reception, is driven by a rising edge of this signal.

Transmission and reception speeds can vary from 0 to 500 Kbits/s. These two clocks can be divided by 16 or 6

2- **Control lines of a typical modem device**

a- **CTS** (Clear To Send) line

- Input to determine if the modem is ready to receive information. A low level indicates that the modem is ready. A high level indicates that the modem is absent or not ready. Influences the TDRE bit of the SR. Note: If there is no modem (or other device), always set this input to low level.

b- **DCD** (Data Carrier Detect) line Carrier detection or carrier loss.

- Input to determine if the carrier at the modem level is present. Absence of carrier inhibits reception, resulting in a high level on this input. (Fault on the line!) A low level indicates carrier presence at the modem level, which is the normal state. Note: Not used, it should always be kept low!
- This input can generate an IRQ interrupt if CR7 = 1 and if a rising edge occurs on DCD.

c- **RTS** (Request To Send) output Transmission request.

- This output requests the modem (or other device) for transmission by the microprocessor (transmission requested by the microprocessor). The state of this output depends on the word written in the CR. The transmission request is indicated by RTS at low level, figure 9.

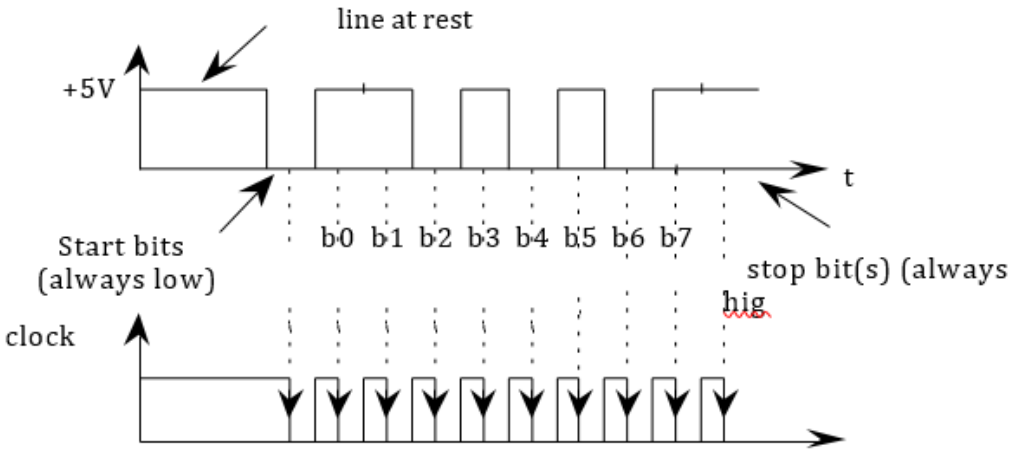


Figure 9: Word Format

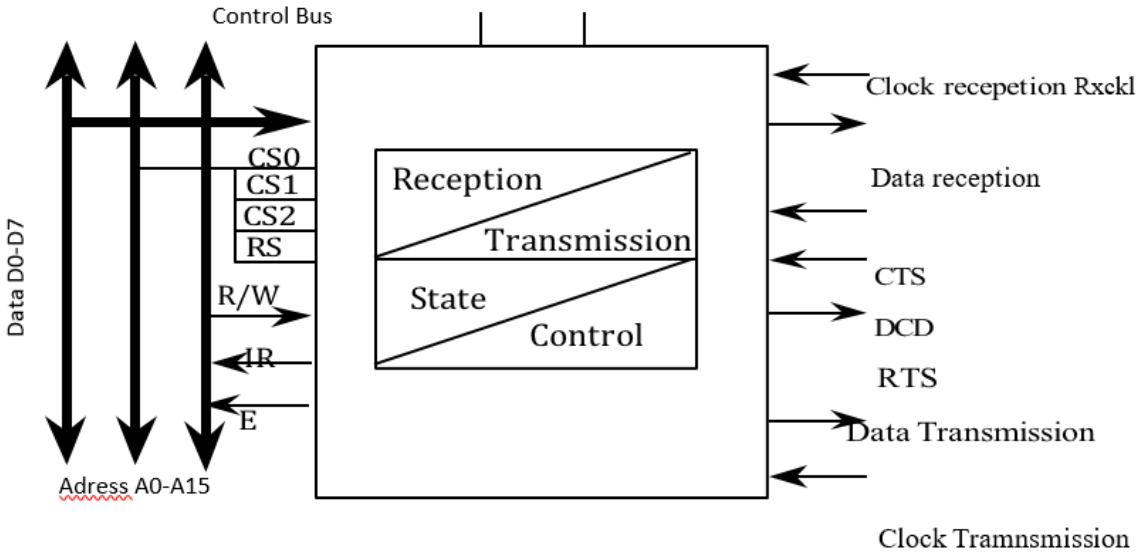


Figure 10 : Externe Artitechture of ACIA

Examples:

a-Modem controlled by an ACIA, figure 11

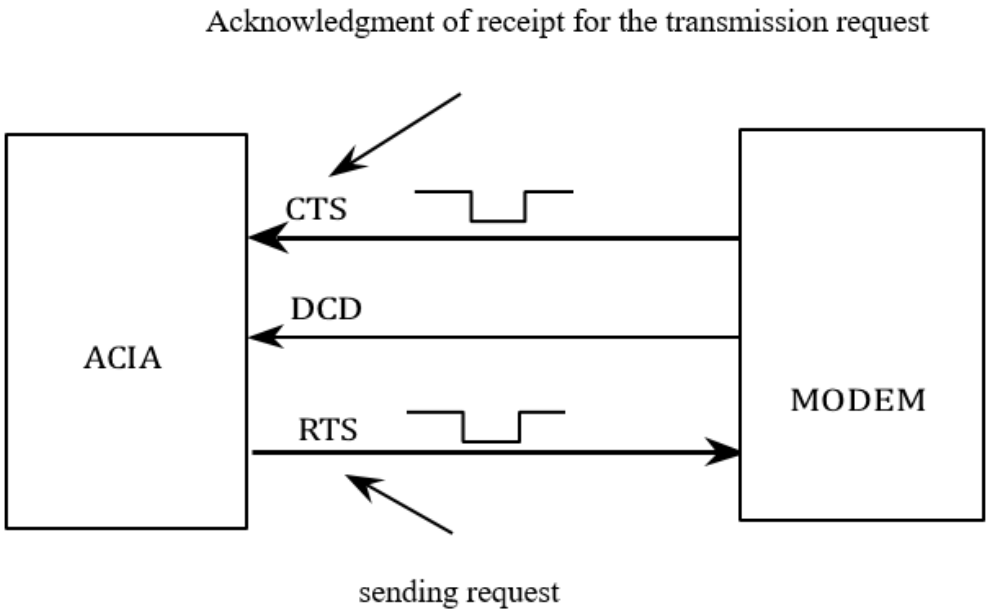


Figure 11: Modem controlled by an ACIA

b- Connection ACIA – ACIA, figure 12

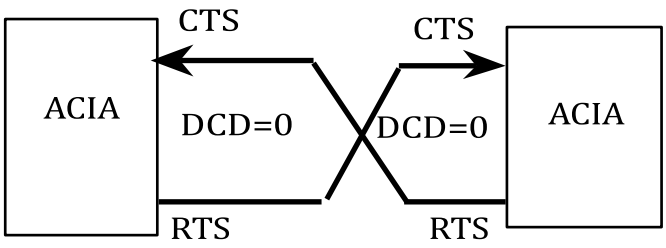


Figure 12 : Connection ACIA - ACIA

4.5 Conclusion

The interface circuits allow the processor to communicate with the outside and exchange with input-output peripherials. In this chapter, we principally focus on the serial interface by illustrating the internal and external architecture of a UART ACIA of Motorla to well understand how it connected with the processor, [3].

Chapter 5 The Microcontroller

Chapter 5 The Microcontroller

5.1 General Overview of the Microcontroller

From microprocessor to microcontroller

As seen in the previous chapters, the processor is the central element of a computer system: it interprets instructions and processes data from a program. It requires certain external elements to function, [8]:

- A clock to pace it (usually quartz or Phase-Locked Loop (PLL));
- Memory to store variables during program execution (RAM) and the program from one power-up to another (ROM). If designing a system dedicated to a specific task (as is often the case with embedded systems), the program is not expected to change. Therefore, it can be stored in read-only memory (ROM).
- Peripherals (to interact with the external world).

These elements are connected by three buses:

- The address bus allows the microprocessor to select the memory location or device it wants to access to read or write information (instruction or data).
- The data bus facilitates the transfer of information between different elements; this information can be instructions or data coming from or going to memory or peripherals.
- The control bus indicates whether the current operation is a read or a write, if a peripheral requests an interrupt to relay information to the processor, etc.

Traditionally, these components are integrated into separate circuits. The development of such a microprocessor-based system is therefore penalized by (non-exhaustive list):

- The need to plan the interconnection of these components (bus, wiring, connection ribbons).
- The physical space occupied by the components and means of interconnection.
- Energy consumption.
- Heat dissipation.
- Financial cost.

A microcontroller (abbreviated as µc, uc, or MCU) is an integrated circuit that combines the essential elements of a computer: processor, memories (ROM and RAM), peripheral units, and input-output interfaces. When all the functions of the computer system are grouped into

Microcontrollers are characterized by :

- a higher degree of integration,
- lower power consumption,
- slower operating speed (ranging from a few megahertz to over one gigahertz),
- and reduced cost compared to general-purpose microprocessors used in personal computers.

Compared to electronic systems based on microprocessors and other separate components, microcontrollers help to :

- reduce the size,
- power consumption, and
- cost of products.
- They have thus contributed to popularizing the use of computing in a wide range of products and processes.
- commonly used in embedded systems, such as automotive engine controllers, remote controls, office equipment, household appliances, toys, mobile phones, etc.

Microcontrollers improve the integration and cost (related to design and implementation) of a microprocessor-based system by bringing together these essential elements in a single integrated circuit. A microcontroller is therefore a standalone component capable of executing the program stored in its ROM as soon as it is powered on. Depending on the models and operating conditions, microcontrollers may require a few external components (quartz, a few capacitors, sometimes a ROM), but this remains very limited. **microcontrollers**.

5.2 Microcontroller Architecture

Microcontroller architecture comprises two main components: ***hardware*** and ***software***. The hardware encompasses all aspects related to the design of the microcontroller and its peripheral units. Software, on the other hand, involves low-level programming using assembly language to access and manipulate the hardware and peripheral units.

We can also present the architecture of microcontroller from another point of view by addressing his ***internal*** and ***external*** aspect.
All microcontrollers operate based on one of two fundamental design models: ***Harvard Architecture*** and ***Von Neumann Architecture***. They represent two different ways of exchanging data between CPU and memory. Microcontrollerss with Harvard architecture are called ***RISC MCs, Reduced Instruction Set Computer***, while microcontrollers with Von-Neumann's architecture are called ***CISC MCs, Complex Instruction Set Computer.***

A ***Harvard architecture*** means that the program and data are stored in separate memory spaces which are accessible ***simultaneously***. Therefore, while one instruction is being executed, the next one can be fetched. This is partly how one execution per clock cycle can be achieved. With other microcontroller architectures, namelly ***Von newman architecture***, there is only one way to access memory, so executions and program instruction access must be done ***alternately.*** We will present a microcontroller from the Microchip family in order to fully detail the architecture of the microcontrollers, [13].

5.2.1 PIC 16F64 Microntroller architecture and peripherals

Microchip introduced powerful microcontrollers called ***Peripheral Interface Controller*** (***PIC)***. They are a generation of 8-bits, 16-bits and 32-bits microcontrollers. These microcontrollers are used for a variety of applications involving limited calculations and some control strategies. They are used for industrial and commercial control applications, appliances control, instrumentation, etc.

In this chapter, our interest turns to the PIC **16F84** because it is suitable for beginners due to its simple architecture and its small components in order to take their first steps in microcontroller programming, [12]. Therefore, we start by presenting the PIC 16F84 through its main features, shown in figure 13 and figure 14.

5.2.2 PIC 16F84 External architecture and pinout

A microcomputer made on a single semiconductor chip is called single-chip microcomputer and single chip microcomputers are generally used in control application. The microcontroller PIC 16F84 is single chip microcomputer marketed in a classic 18 pins case, as shown in figure 13 where each pin is defined below:

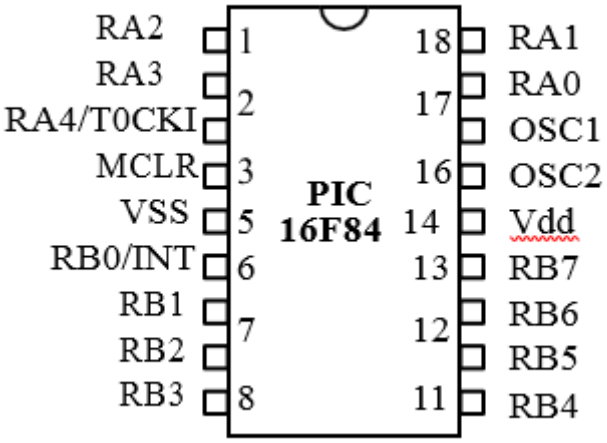


Figure 13:PIC16F84 External architecture

- The circuit is powered by the VDD and VSS pins. These pins supply power to all electronic components of the PIC. To achieve this, connect VSS (pin 5) to ground (0 volts) and VDD (pin 14) to the positive terminal of the power supply, which should provide a DC voltage between 3 and 6 volts.
- The microcontroller is a system that executes instructions sequentially at a speed (frequency) determined by an internal clock within the circuit. This clock needs to be externally stabilized using a quartz crystal connected to the OSC1/CLKIN (pin 16) and OSC2/CLKOUT (pin 15) pins.
- Pin 4 is called MCLR. When the applied voltage is equal to 0V, it allows resetting the microcontroller. This means that if a low level (0 volts) is applied to MCLR, the microcontroller stops, places all its registers in a known state, and redirects itself to the beginning of the program memory to restart the program from the beginning (address in the program memory: 0000). At power-up, since the MCLR pin is at zero, the program starts at address 0000 (MCLR stands for Master Clear Reset).
- Pins RB0 to RB7 and RA0 to RA4 are digital input/output lines. There are a total of 13 of them, and they can be configured as either inputs or outputs. These pins enable the microcontroller to communicate with the external world (peripherals). The group of lines RB0 to RB7 forms port B, while the lines RA0 to RA4 form port A. Some of these pins also have other functions (such as interrupts or timers), [9].

5.2.3 PIC 16F84 internal architecture

The diagram of figure 14 illustrates the general internal architecture of the circuit, comprising the following elements:

1. A power-up initialization system
2. Clock generation system from external quartz
3. Arithmetic and logic unit (ALU)
4. 1k-word flash program memory of 14 bits
5. Program counter (Ordinal Counter) and a stack
6. Specific bus for program (Instruction Bus)
7. Register containing the instruction code to be executed (instruction register)
8. Specific bus for data
9. RAM memory containing:

    Special Function Registers (SFR)

    68 bytes of data
10. EEPROM memory of 64 bytes of data
11. Two input/output ports (Port A and Port B)
12. Watchdog timer

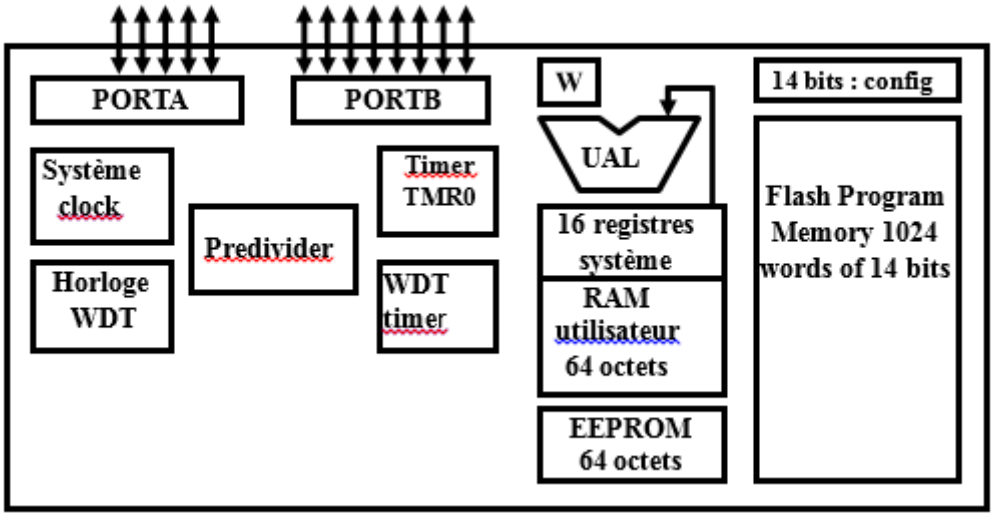


Figure 14: PIC 16F84 Internal Architecture

5.3 Flash Program Memory

This memory of 1024 words stores the program. It is non-volatile and re-programmable as desired. Each 14-bit position contains an instruction. The program location can be anywhere in the memory. However, it should be noted that following a RESET or when switching on, the PIC starts the execution at 0000H. Furthermore, when there is an interruption, the PIC goes to 0004H. It is therefore advisable to place the start of the program after address 0004H and to place a link at the beginning of the programme at 0000H and a link to the end of the interruption routine if there is one at 004H, [11].

5.3.1 RAM Memory

- The Special Function Registers (SFRs) are the operating registers of the PIC. The set of these registers is often referred to as a registry file. We will return to these registers throughout this document.
- GPR registers (General Propose Register) are memory positions that the user can use to store their variables and this data. Regardless of their nature, the RAM positions are always called registers. As seen in Table.14, the RAM consists of two parts.

Table 24: PIC 16F84 RAM Memory and STATUS Register

	bank 0	bank 1	
00	INDF	INDF	80
01	TMR 0	OPTION	81
02	PCL	PCL	82
03	STATUS	STATUS	83
04	FSR	FSR	84
05	PORTA	TRISA	85
06	PORTB	TRISB	86
07			87
08	EEDATA	EECON1	88
09	EEADR	EECON2	89
0A	PCLATH	PCLATH	8A
0B	INTCON	INTCON	8B
0C 4F	Memory USER	Maped in bank0	8C . . . . . CF

STATUS Register	IRP	RP1	RP0	TO	PD	Z	DC	C

The RAM is organized in two banks, to access a register; you must first place yourself in the bank where it is located. This is done by positioning the RP0 bit of the ***STATUS register***.

- RP0 = 0 Bank 0
- RP0 = 1 bank 1

For user memory, the use of the pages (Bank) is not necessary since the Bank 1 is "mapped" with the Bank0. This means that writing a data to 0CH or 8CH is the same.

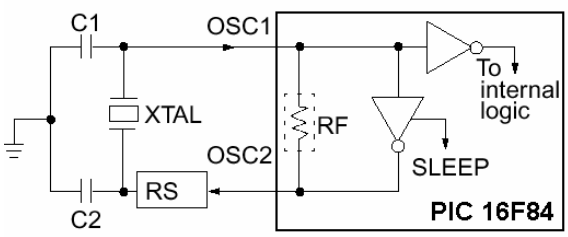
5.3.2 ALU and Register W

It is an 8-bit ALU that performs the arithmetic and logical operations between the accumulator W and any other register 'F' or constant K. The W accumulator is an 8-bit working registry, it does not have an address like the other SFRs. For two-operand instructions, it is always he who contains one of the two operands. For instructions to an operator, it can be either W or any F register. The result of the operation can be placed either in the W work register or in the F work register.

5.3.3 The Clocks

The clock can be either internal or external. The internal clock consists of a quartz oscillator or a RC oscillator, figure 15. With the Quartz oscillator, you can have frequencies up to 4, 10 or 20 MHz depending on the type of MC. With a RC oscillator, the oscillation frequency is set by Vdd, Rext and Cext. It may vary slightly from one circuit to another, where in the pass low filter C2 limits the harmonics due to scratching and reduces the amplitude of the oscillation.In some cases, an external clock to the microcontroller can be used to synchronize the PIC on a particular process.

Regardless of the oscillator used, the system clock is obtained by dividing the Frequency oscillation Fosc by 4, (Fosc/4). So, with a 4 MHz quartz, we get a 1 MHz instruction clock, which is the time to execute an instruction of 1µs.



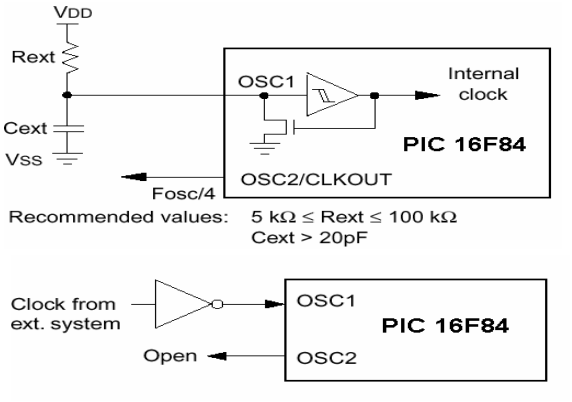


Figure 15: PIC 16F84 system clock

5.3.4 The port of E/S PORTA

The A port designated by PORTA is a 5-bit port (RA0 to RA4). Each E/S is TTL compatible. The direction configuration for each bit of the port is determined with the TRISA registry.

- TRISA bit i = 0 PORTA bit i configured as output
- TRISA bit i = 1 PORTA bit i configured as input

The RA4 pin is multiplexed with the clock input of the TMR0 timer, so it can be used either as normal E/S of port A, or as clock entry for the Timer TMR0, the choice is made using the T0CS bit of the OPTION_REG registry, Table 15.

- T0CS = 0 RA4 is a normal E/S
- T0CS = 1 RA4 = external clock for the timerTMR0

RA4 is an E/S with open drain, that can be used as an output (to turn on a LED for example), by putting an external resistance to Vdd. The diagram above illustrates the principle of an open drain output (or open collector): if RA4 is positioned at 0, the switch is closed, the output is connected to the mass. If RA4 is placed at 1, the switch is open, the output is disconnected and therefore requires the external resistance to bring the current from the power supply to the LED, figure 16, (the value of 1k is given as an indication, you can adjust according to your application)

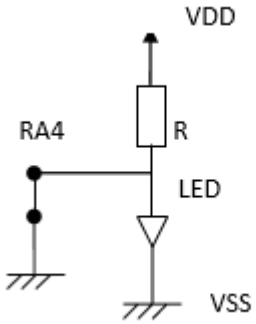


Figure 16: An open drain output

Table 15 : OPTION REGISTER

Register OPTION_REG	RBPU	INTEDG	**TOCS**	TOSE	PSA	PS2	PS1	PS0

5.4 The port of E/S PORTB

The port B designated by PORTB is a two-way port of 8 bits (RB0 to RB7), where all pins are TTL compatible. The management configuration is done using the TRISB register.

In input, the RB0 line also called INT can trigger the external interruption INT. As an input, any of the RB4 to RB7 lines can trigger the RBI interruption, |14].

The TMR0 Timer

The PIC 16F84 TMR0 works following the features below:

- It is continuously enhanced either by the internal Fosc/4 clock (timer mode) or by an external clock applied to the RA4 pin of port A (counter mode). The choice of the clock is made using the T0CS bit of the register OPTION_REG, Table 15.

  TOCS = 0, internal clock

  TOCS = 1, external clock applied to RA4.

- In the case of the external clock, you can choose the front on which the TIMER increases.

  TOSE = 0 increment on rising edge

  TOSE = 1 increment on descending edge

- Whichever clock is chosen, one can pass it in a programable frequency divider (prescaler) whose ratio is fixed by the PS0, PS1 and PS2 bits of the register OPTION_REG, Table 4 below. Whether or not the pre-divider is assigned is done using the PSA bit:

  PSA = 0; the pre-divider is used

  PSA = 1, no pre-divider used (assigned to the watch dog)

- The contents of the TMR0 timer can be accessed through the registry with the same name. It can be read or written at any time. After one writing, increment is inhibited for two instruction cycles.
- TMR0, the T0IF flag is placed at 1. This may trigger T0I interruption if validated.

Table 16 :Mode TMR0 Pre-divider

PS2	PS1	PS0	Div
0	0	0	2
0	0	1	4
0	1	0	8
0	1	1	16
1	0	0	32
1	0	1	64
1	1	0	128
1	1	1	256

5.4 The Watchdog Timer WDT

It is an 8-bit counter that is continuously incremented (even if the MC is in sleep mode) by an integrated RC clock independent of the system clock. When overflowing, (WDT TimeOut), two situations are possible:

- If the µC is in normal operation, the WDT time-out causes a RESET. This prevents staying stuck in the event of the microcontroller being blocked by an uncontrolled undesirable process.
- If the µC is in SLEEP mode, the WDT time-out causes a WAKE-UP, the program's execution continues normally where it stopped before returning to SLEEP mode. This situation is often exploited for timing.

The WDT clock is adjusted so that Time-Out arrives every 18 ms. However, it is possible to extend this time by passing the Time-Out signal into a programmable predictor (shared with the TMR0 timer). the assignment is made using the PSA bit of the OPTION_REG registry

- **PSA = 1** the pre-divider is used
- **PSAs = 0** no pre-divider (affected to TMR0)

The pre-divider ratio is set by the PS0, PS1 and PS2 bits of the OPTION_REG registry, as seen in Table 17. WDT should be used with caution to avoid repeated (unexpected) reset of the program. To avoid a WDT timeOut when running a program, there are two possibilities:

- Inhibit WDT permanently by putting the WDTE bit to 0 in the configuration EEPROM

- Periodically restore the WDT to zero in the program using the CLRWDT instruction to prevent from overflowing.

5.5 The SLEEP mode

The PIC can be placed in low power mode using the SLEEP instruction. In this mode, the system clock is shut down which stops the program running.

Table 17 : Mode WDT Pre-divider

PS2	PS1	PS0	Div
0	0	0	1
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

To exit SLEEP mode, you have to trigger a WAKE-UP, for this there are 3 possibilities:

- External RESET due to PIC initialization by setting the MCLR entry to 0. The PIC resumed the implementation of the programme from the outset.
- Timeout of WDT watch dog if validated. The PIC resumes the program from the instruction following the SLEEP instruction
- Interruption of INT (on RB0) or RBI (on RB4-RB7) or EEI (end of EEPROM data writing). The validation bit of the interruption in question must be validated, however, the WAKE-UP takes place regardless of the global GIE validation Bit position.

There are two situations:

GIE = 0, the PIC resumes the execution of the program from the instruction following the SLEEP instruction, interruption is not taken into account

GIE = 1, the PIC executes the instruction that is located right after the SLEEP instruction and then connects to address 0004 to execute the interruption procedure.

If the instruction following SLEEP is not desired, use the NOP instruction.

5.6 Configuration EEPROM memory

During the deployment phase of a program in the PIC program memory, a configuration EEPROM consisting of 5 14-bit words is also programmed, Table 18, [12]:

- 4 identification words (ID) from the address 0x2000 that may contain any trace that we will not use,
- 1 configuration password (address 0x2007) that allows:

To choose the type of oscillator for the clock

To validate the WDT timer or to prohibit the reading of EEPROM memories.

Table18: Register of Configuration

13	12	11	10	9	8	7	6	5	4	3	2	1	0
CP	CP	CP	CP	CP	CP	CP	CP	CP	CP	PWRTE	WDTE	FOSC1	FOSC0

- Bits 1: 0 FOSC1:FOSC0 Selection of the oscillator type for the clock

  11: RC Oscillator

  10: High speed HS oscillator: high frequency quartz (up to 10 MHz)

  01: Oscillator XT, this is the most used mode, quartz up to 4 MHz

  00: Low power LP oscillator, reduced consumption, up to 200 kHz

- Bit 2 WDTE timer validation

  WDT 1: WDT validated;

  WDT 0; WDT inhibited

- Bit 3 PWRTE validation of a time to turn

  1: Timing inhibited;

  0: validated Timing

- Bit 13:4 Code Protection of the Program

  1: Code reading Protection:
  0: Protection enabled

5.6.1 EEPROM Data Memory

The EEPROM data memory consists of 64 bytes starting at the address 0x2100 that can be read and written from a program.

Access is made using the EEADR and EEDATA registers: any reading and writing in the EEDADA registry is done in the memory position pointed by the EEEDR. In fact, the EEADR contains the relative address relative to the page that starts with 0x2100, i.e., the address goes from 0 to 63.

Two control registers (EECON1 and EECON2) are associated with the EEMROM memory, Table 19. The time of writing of one byte is about 10 ms, the end of each successful writing is announced by the EEIF flag and the reduction to zero of the RW bit of the EECON1 registry. The EEIF flag may trigger the EEI interruption if it has been validated.

Procedure for reading data in the EEPROM

- Place the relative address in EADR
- Set the RD bit of EECON1 to 1
- Read the contents of the EEDATA registry.

Procedure for writing data into the EEPROM

- Writing in the EEPROM shall be permitted: WREN bit = 1
- Place the relative address in the EADR
- Place the data to write in EEDATA
- Place 0x55 in EECON2
- Place 0xAA in EECON2
- Start writing by positioning the WR bit
- Wait for writing to end, (10 ms) (EEIF=1 or WR=0)
- Start again at point 2 if we have other data to write

The WRERR flag is positioned if a typing error occurs EECON2 is not really a Registry. Microchip uses it as a command log. Writing specific values in EECON2 causes a specific command to be executed in the internal electronics of the PIC.

Table 19 : EECON2 REGISTER

EECON2	-	-	-	EEIF	WRERR	WREN	WR	RD

5.6.2 Interruptions

An interruption causes the main program to stop to run an interrupt procedure. At the end of this procedure, the microcontroller resumes the program where it stopped. The PIC16F84 has 4 interruption sources, [11].

Two bits are associated with each interruption: a validation bit and a flag. The first allows you to allow or not to allow the interruption, the second allows the programmer to know what the disruption is. All these bits are in the INTCON register except the EEIF flag of the EEI interruption which is in the EECON1 register.

5.7 Occurrence of an interruption

When the event triggering an interruption occurs, then its flag is positioned at a (raised). If the corresponding interruption has been validated, it is then triggered: the program stops what it is doing and will run the interrupt procedure at address H0004 by performing the following steps, [11], [12]:

- The address contained in the PC (Program Counter) is saved in the stack, then replaced by the value H0004 (interruption routine address).
- The GIE bit is placed "0" to inhibit all interruptions (so that one is not disturbed during the execution of the interruption procedure).
- At the end of the interruption procedure (RETFIE instruction):

  -The GIE bit is reset to the top state (thus allowing another event)

  -The contents of the PC are re-charged from the battery allowing the program to resume where it stopped.

Important remarks:

- The flag remains in high condition even after the interruption has been processed. Therefore, it should always be reset to "0" at the end of the interruption routine, otherwise the disruption will be triggered again right after the RETFI instruction.
- Only the PC is automatically stacked. If necessary, the W and STATUS registers should be saved to RAM and then restored at the end of the routine so that the microcontroller can resume the program in the same conditions as it left it.

5.7.1 INT Interruption (Port B RB0 Entry)

This interruption is caused by a state change on the RB0 input of port B when it is programmed as input. It is managed by the bits:

- INTE: validation bit (1=yes, 0=no)
- INTF: flag
- INTEDG: trigger front, 1=sum, 0=descending (OPTION_REG register)

1. RBI Interruption (RB4 A RB7 du port B)

This interruption is caused by a state change on one of the RB4 to RB7 entries of port B, Front doesn't matter. Associated bits are RBIE (validation) and RBIF (Flag)

2. T0I interruption: TMR0 Timer debugging

This interruption is caused by overload of the TMR0 timer. Associated bits are T0IE (validation) and T0IF (Flag).

3. EEI interruption: End of writing in EEPROM, Table 20

This interruption is triggered at the end of a successful writing in the EEPROM. Associated bits are EEIE (validation) and EEIF (Flag). GIE : This bit allows to validate or prohibit (globally) all interruptions.

Table 20 : EEII Interruption

INTCON	GIE	EEIE	T0IF	INTE	RBIE	T0IF	INTF	RBIF
EECON1	-	-	-	EEIF	WRERR	WREN	WR	RD
OPTION_REG	RBPU	INTEDG	TOCS	TOSE	PSA	PS2	PS1	PS0

4. Indirect addressing

Indirect addressing is done through the FSR and INDF registers. The INDF registry is not a real registry but represents the memory box pointed by the FSR index registry.
To read or write in a memory box using indirect addressing, first place the address in the FSR registry, then read/write in the INDF registry

5. The program counter,

The Program Counter is a 13-bit registry that increases automatically when the program runs, Table 21. However, it can be modified by program to what is called a calculated goto. It is accessible through the PCL and PCLATH registers

Table 21: Counter Program PC

		PCH						PC	L			

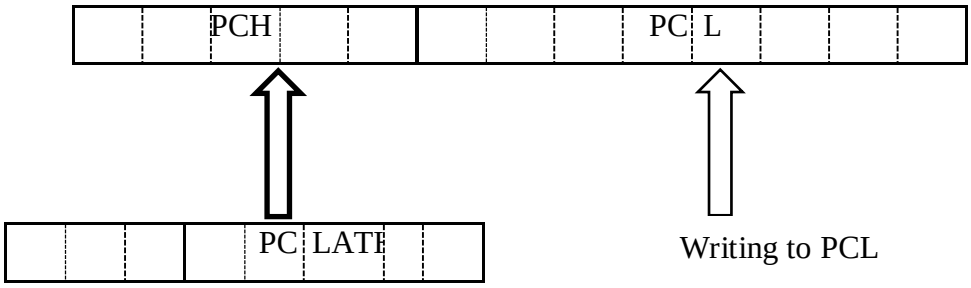
- PCL (8 bits) is the lower part of the PC, it is accessible in reading writing
- PCH (5 bits) is the upper part of the PC, it is not directly accessible.

However, it can be modified indirectly using the PCLATH registry, which is an SFR registry that is read and written and only uses 5 bits.

6. GOTO calculated

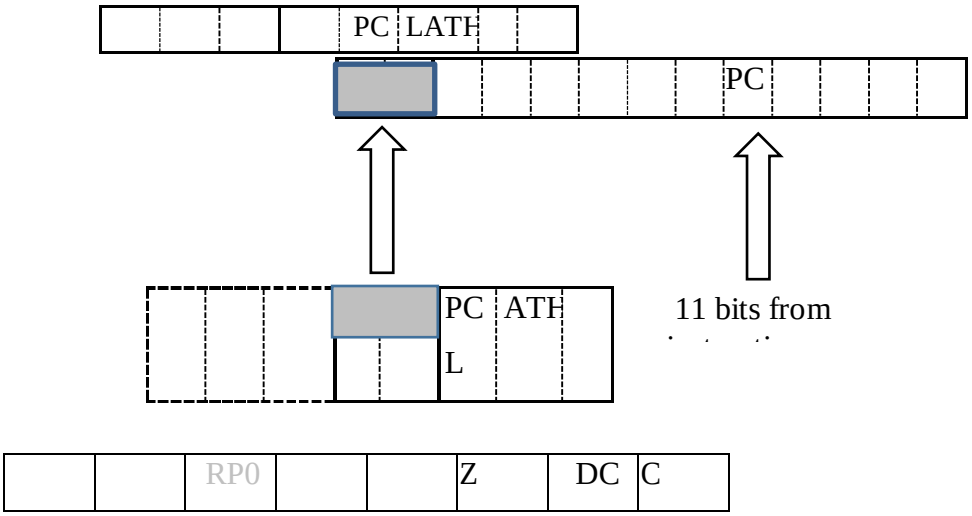
If you want to modify the Program Counter to make a jump, you must first place the top part in the PCLATH registry, then write the bottom part in PCL. When writing to PCL, the contents of PCLATH are automatically re-copied to PCH, Table 22.

Table 22 : PCLATH 1



In the connection instructions, the destination address is encoded on 11 bits. When executing such instructions, the 11 bits are copied into PC, the two missing bits being taken into PCLATH. For the 16F84, we will not need these bits because to address 1024 program lines, only 10 bits of the Counter Program are used, Table 23.

Table 23 : PCLATH 2



Indicators C, DC, and Z are bits that inform us of the outcome of an instruction. They are located in the STATUS register:

STATUS REGISTER

• C (Carry): This bit It moves to "1" when the result of an operation exceeds the value of FF or if the result is negative.

• DC (Digital Carry): This bit moves to "1" when a retention occurs between the bits 3 and 4.

• Z (Zero): This bit is changed to "1", to indicate that the result of the operation is null

5.8 Microcontroller programming

1. The instructions of the 16F84

All Mid-Range PICs have a set of 35 instructions. Each instruction is encoded on a 14-bit word that contains the operation code (OC) and the operand. Except for the jump instructions, all instructions are executed in a clock cycle. Knowing that the clock provided to the PIC is predivided by 4, if we use, for example, a quartz of 4MHz, so we get 1000000 cycles/second, this gives us a power of about 1MIPS (1 Million Instructions Per Second). With a 20MHz clock, you get a more than honorable processing speed.

2. Byte-oriented instructions (direct addressing)

These are instructions that manipulate data in the form of bytes. They are coded as follows:

- 6 bits for instruction: logical, because there are 35 instructions, it takes 6 bites to be able to encode them all.
- 1 bit (d) to indicate whether the result is to be kept in the work register (accumulator) W of the calculation unit (W for Work) or saved in a F register (F: File).
- 7 bits left to encode the address of the operand (128 positions).

Trouble! 7 bits do not give access to total RAM, so here is the explanation of the division of RAM into two banks. To replace the missing bit, the RP0 bit of the STATUS registry is used. Although it is not used on the 16F84, the RP1 bit is also reserved for the change of bank, for example the 16 F876 has 4 banks.

3. "Bit Oriented" Instructions

These are instructions designed to directly manipulate the bits of a registry of a memory box. They are coded as follows:

- 4 bits for instruction - 3 bits to indicate the number of the bit to be manipulated (from 0 to 7)
- 7 bits to indicate the operand.

5.9 Instructions operating on a data (immediate addressing)

These are the instructions that manipulate data that are encoded in the instruction directly. They are coded as follows:

- The instruction is encoded on 6 bits - It is followed by an IMMEDIATE value encoding on 8 bits (thus from 0 to 255).

5.10 Instructions for skipping and appealing procedures

These are the instructions that cause a break in the program sequence. They are coded as follows:

- Instructions are encoded on 3 bits
- The destination is encoded on 11 bits

We can already infer that the jumps only give access to 2K of program memory (211). No problem for the 16F84, which only has 1k of program memory.

1. ***Examples of instruction***

MOVWF F; copies W to address registry F that refers to the address of any SFR or GPR registry. For SFR registers, their names can be used provided that the p16F84.inc file is included in the program

MOVWF	0x2C	; copies W to address memory box $2C_h$
MOVWF	EEDATA	; copies W to the registry EEDATA
MOVF	0x08	; copies W to the registry EEDATA

MOVF F, d; copies the F register either in W or in itself; collecting a register on itself may seem uninteresting, but as instruction places the indicators, it may prove interesting, figure 17.

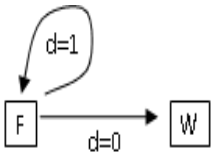


Figure 17: Direction of Loading in W or F

2. *16F84 Instructions Set [14], Table 24.*

Table 24 : 16F84 Instructions Set

INSTRUCTIONS WITH REGISTER (direct)		indicators	Cycles
ADDWF F,d	W+F {W F d}	C,DC,Z	1
ANDWF F,d	W and F {W,F, d}	Z	1
CLRF F	Clear F	Z	1
CLRW	Clear W	Z	1
CLRWDT	Clear Watchdoc timer	TO', PD'	1
COMF F,d	Complementary F {W,F, d}	Z	1
DECF F,d	decrepit F {W,F, d}	Z	1
DECFSZ F,d	decrepit F {W,F, d} skip if 0		1(2)
INCF F,d	Increase F {W,F, d}	Z	1
INCFSZ F,d	Increase F {W,F, d} skip if 0		1(2)
IORWF F,d	W or F {W,F, d}	Z	1
MOVF F,d	F {W,F , d}	Z	1
MOVWF F	W F		1
RLF F,d	rotation left of F through C {W,F ? d}	C	1
RRF F,d	right rotation of F through C , {W,F , d}		1
SUBWF F,d	F, W {W,F , d}	C,DC,Z	1
SWAPF F,d	switches the 2 quartets of F {W,F, d}		1
XORWF F,d	W xor F, {W,F, d}	Z	1
BIT WORKING INSTRUCTIONS			
BCF F,b	RAZ du bit b of the register F		1
BSF F,b	RAU du bit b of the register F		1
BTFSC F,b	test he bit b de F, if 0 skip an instruction		1(2)
BTFSS F,b	test the bit b de F, if 1 skip an instruction		1(2)

INSTRUCTIONS WITH DATA (Immediat)			
ADDLW K	W + K W	C,DC,Z	1
ANDLW K	W and K W	Z	1
IORLW K	W or K W	Z	1

MOVLW K	K W		1
SUBLW K	K – W W	C,DC,Z	1
XORLW K	W xor K W	Z	1

GENERAL INSTRUCTIONS			
CALL L	Linking to a L-label subprogram		2
GOTO L	link to label line L		2
NOP	No operation		1
RETURN	Returns from a subprogram		2
RETFIE	Interruption Return		2
RETLW K	Returns from a subprogram with K in W		2
SLEEP	turns into standby mode	TO', PD'	1

{W, F, d} means that the result goes either in W if d=0 or w, or in F if d = 1 or f

Application Example

Traduce the following algorithm in assembly instructions of PIC16F84. Explain each line of instruction.

Algorithm:

1. Read PORTA: place PORTA into W
2. Transfer W to PORTB
3. Repeat (go to 1)

Solution

1.**Main Programm**

```
movf PORTA, W ; put PORTA into W
movwf PORTB ; put W into PORTB
goto Loop
```

2.**Whole program** (with directives and initialisations)

```
; ------------------------------ Configuration ------------------
-------
list p=16F84A
#include <p16F84A.inc>
__CONFIG _CP_OFF & _WDT_OFF & _PWRTE_ON & _XT_OSC
```

```
; ------------------------------- RESET Startup ------------------
-------
org 0x00 ; start of the program address
goto initialization

; ----------------------------- Ports Initialization -------------
-------
initialization
bsf STATUS, RP0 ; select bank 1
movlw 0xFF ; configure PORTA lines
movwf TRISA ; PORTA as input
movlw 0x00 ; configure PORTB lines
movwf TRISB ; PORTB as output
bcf STATUS, RP0 ; select bank 0

; ---------------------------------- Main Program ---------------
-------
loop
movf PORTA, W ; put PORTA into W
movwf PORTB ; put W into PORTB
goto loop
end
```

5.11 Conclusion

In conclusion, microprocessors based systems are the cornerstone of the digital age. Their continual advancements are transforming our way of life, work, and interaction with the world. Faced with technical, ethical, and environmental challenges, the industry continues to innovate, promising a future where technology is both powerful and respectful of our planet. The great advantage of these system sis that microprocessors, though small in size, continue to have a vast impact on our society, propelling humanity towards new technological horizons.

Indeed, we observed that a microprocessor is able to execute a sequence of instructions on a set of data. A high-performance processor is one that has a well-designed instruction set that should encompass the diverse categories to enable flexible and efficient programming for various tasks. We have pointed out that the executable machine program must reside in main memory (central memory), and the microprocessor executes this program instruction by instruction based on concepts related to micro-commands and sequencers. Furthermore, the performance of microprocessors.

Microcontrollers are at the heart of many modern electronic devices. They integrate a processor, memory, and input/output interfaces, all on a single integrated circuit. This integration makes them ideal for numerous applications, from toys to complex embedded systems.

Bibliography

Bibliography

[1]. A. Richard SOREK « L'assembleur du 6809 et ses périphériques », v4.13 2019

[2]. P.C. Lacaze, J.C Lacroix « Mémoires Electroniques : Concept, matériaux, dispositifs et technologies », collection electronique, edition ISTE.

[3]. J. Y. Haggège, « Microprocesseur et Interfaçages : Support de cours », INSET, 2003.

[4]. Lilen, « Cours fondamental des microprocesseurs », Dunod, 1993

[5]. Alain-Bernard Fontaine, « Le Microprocesseur 16 bits-8086-8088 », 2ième édition, Manuels informatiques», Masson, 1997.

[6]. Michel Aumiaux, « Microprocesseurs 16 bits », 1997.

[7]. J. Crisp,« Introduction to Microprocessors and Microcontrollers», Elsevier, 2nd edit 2004.

[8]. Christian Tavernier, « Microcontrôleurs PIC 10, 12, 16, Description et mise en œuvre », Dunod, 2007.

[9]. Pascal Mayeux, « Apprendre la programmation des PIC Mid-Range par l'expérimentation et la simulation », Dunod, 2010

[10] Micrichip, PIC16F84A Data Sheet 18-pin Enhanced FLASH/EEPROM 8-bit Microcontroller.

[11] Bignoff ; La programmation des PIC, première partie-révision R35 Démarrer les PIC avec le PIC 16F84.

[12] Nebojsa Matic, "PIC Microcontrollers."

[13] Marc Spencer. MEPIC programming for bigginers. The American Radio Relay League, Inc, 2010.

[14] Microchip. PIC Micro MID-Rang MCU Family Reference Manual.