Final project of the assembly language course

Digital Piano in 8086 assembly for DOS

Parra M. Ariel, Batres L. Miguel, Salas P. Diego

4th semester, group A. Center of Basic Sciences, Computer Systems Engineering.

Universidad Autonoma de Aguascalientes

Aguascalientes,Aguascalientes. PC: 20131

{al280862, al350553, al281435}@edu.uaa.mx

**Summary.** This project aims to develop a digital piano program in Assembly language for the intel 8086 processor and DOS 16 bits Operating System, the project aims to generate melodies using the computer's speakers. The program will be using ports 61h to activate and deactivate the speaker and ports 43h and 42h for adjusting the speaker frequency; with four melodies loaded, each at least 30 seconds long, which will be initiated when chosen by the user from a menu; Implementing a piano-like functionality where specific keys trigger corresponding sounds, including flats and sharps. And the program will provide a user-friendly interface for selecting preloaded melodies and enable real-time sound generation akin to playing a piano.

**Key Words:** assembly, intel 8086, DOS.

# Introduction

Assembly language is a CPU-specific low-level language, unlike high-level languages, which abstract away much of the underlying hardware complexity, assembly language provides direct access to a computer's central processing unit (CPU) through manipulation of registers and untyped variables. In this realm, variables are often identified solely by their data size, typically denoted in bytes or words, a convention especially prevalent in architectures similar to the Intel 8086 processor.

In this project we are required to designing a program in assembly language for the Intel 8086 processor that aims to generate melodies through the computer's speaker, using the DOS operating system. Creating such a program requires a deep understanding of the 8086 architecture and its capabilities, as well as mastery of low-level programming concepts. Through careful manipulation of registers, control flow instructions, and direct interfacing with hardware components, in this case, the speaker; The program can produce a wide range of musical tones and rhythms. This project not only showcases the versatility of assembly language but also underscores the intricate relationship between software and hardware in computing systems.

The collaborative nature of this project highlights the significance of teamwork in tackling complex technological challenges. Working together, individuals bring a diverse range of skills and perspectives to the table, enhancing problem-solving capabilities and fostering innovation. Through effective communication and coordination, team members can leverage their collective expertise to overcome obstacles and achieve shared goals. This collaborative effort not only enriches the development process but also reinforces the value of cooperation in advancing the boundaries of technology.

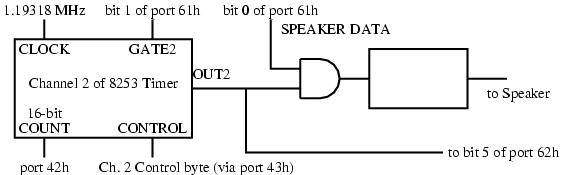
# Theoretical Framework

# Speaker logic gates

Figure 1 is a diagram of the hardware for driving the built-in speaker. OUT2 is the output of Channel 2 of the 8253-5 timer chip, GATE2 (= bit 1 of port 61h) is the enable/trigger control for the Channel 2 counter, and SPEAKER DATA (= bit 0 of port 61h) is a line that may be used independently to modulate the output waveform, e.g., to control the speaker volume.

The count and load modes selected for Channel 2 during BIOS initialization are probably the best to use for tone production. In Mode 3, the counter output is a continuous symmetrical square wave as long as the GATE line of the channel is enabled; the other modes either produce outputs that are too asymmetrical or require retriggering for each count cycle.

The frequency count is loaded into the Channel 2 COUNT register at I/O port 42h. GATE2 (bit 1 of I/O port 61h) must be set to 1 to get an output on OUT2; the SPEAKER DATA line (bit 0 of I/O port 61h) must also be set to 1 to produce a tone. Note that the remaining bits of port 61h must not be changed since they control RAM enable, keyboard clock, etc. To silence the speaker, bits 1 or 0 of port 61h are set to 0 (without disturbing the remaining bits of port 61h).



**Fig. 1.** Speaker

# How to produce a beep sound on the speaker

We can communicate with the speaker controller using IN and OUT instructions. The following lists the steps in generating a beep:

* Send the value 182 to port 43h. This sets up the speaker.
* Send the frequency number to port 42h. Since this is an 8-bit port, we must use two OUT instructions to do this. Send the least significant byte first, then the most significant byte.
* To start the beep, bits 1 and 0 of port 61h must be set to 1. Since the other bits of port 61h have other uses, they must not be modified. Therefore, we must use an IN instruction first to get the value from the port, then do an OR to set the two bits, then use an OUT instruction to send the new value to the port.
* Pause (sleep) for the duration of the beep.
* Turn off the beep by resetting bits 1 and 0 of port 61h to 0. Since the other bits of this port must not be modified, we must read the value, set just bits 1 and 0 to 0, then output the new value.

## Frequency Number used in beep macro

The frequency number is a word value, so it can take values between 0 and 65,535. This means you can generate any frequency between 18.21 Hz (frequency number = 65,535) and 1,193,180 Hz (frequency number = 1).

Knowing this bounds and using the known frequencies in Hertz of musical notes,we created formulas to convert these frequencies into the corresponding frequency numbers. These frequency numbers can then be used as inputs to generate the desired musical tones using the beep macro found in mac.inc

**(1)**

**(2)**

By using this formulas we can get all the values for each note frequency and get the Table 1.

**Table 1.** Note Frequency Numbers using the A= 440Hz scale

|  |  |  |
| --- | --- | --- |
| **Note Frequency Numbers** | | |
| **Note** | **Frequency (Hz)** | **Frequency #** |
| C0 | 16.35 | 72979 (Overflow) |
| C#0/Db0 | 17.32 | 68890 (Overflow) |
| D0 | 18.35 | 65023 |
| D#0/Eb0 | 19.45 | 61346 |
| E0 | 20.6 | 57921 |
| F0 | 21.83 | 54657 |
| F#0/Gb0 | 23.12 | 51608 |
| G0 | 24.5 | 48701 |
| G#0/Ab0 | 25.96 | 45962 |
| A0 | 27.5 | 43388 |
| A#0/Bb0 | 29.14 | 40946 |
| B0 | 30.87 | 38651 |
| C1 | 32.7 | 36488 |
| C#1/Db1 | 34.65 | 34435 |
| D1 | 36.71 | 32502 |
| D#1/Eb1 | 38.89 | 30680 |
| E1 | 41.2 | 28960 |
| F1 | 43.65 | 27335 |
| F#1/Gb1 | 46.25 | 25798 |
| G1 | 49 | 24350 |
| G#1/Ab1 | 51.91 | 22985 |
| A1 | 55 | 21694 |
| A#1/Bb1 | 58.27 | 20476 |
| B1 | 61.74 | 19325 |
| C2 | 65.41 | 18241 |
| C#2/Db2 | 69.3 | 17217 |
| D2 | 73.42 | 16251 |
| D#2/Eb2 | 77.78 | 15340 |
| E2 | 82.41 | 14478 |
| F2 | 87.31 | 13666 |
| F#2/Gb2 | 92.5 | 12899 |
| G2 | 98 | 12175 |
| G#2/Ab2 | 103.83 | 11491 |
| A2 | 110 | 10847 |
| A#2/Bb2 | 116.54 | 10238 |
| B2 | 123.47 | 9663 |
| C3 | 130.81 | 9121 |
| C#3/Db3 | 138.59 | 8609 |
| D3 | 146.83 | 8126 |
| D#3/Eb3 | 155.56 | 7670 |
| E3 | 164.81 | 7239 |
| F3 | 174.61 | 6833 |
| F#3/Gb3 | 185 | 6449 |
| G3 | 196 | 6087 |
| G#3/Ab3 | 207.65 | 5746 |
| A3 | 220 | 5423 |
| A#3/Bb3 | 233.08 | 5119 |
| B3 | 246.94 | 4831 |
| C4 | 261.63 | 4560 |
| C#4/Db4 | 277.18 | 4304 |
| D4 | 293.66 | 4063 |
| D#4/Eb4 | 311.13 | 3834 |
| E4 | 329.63 | 3619 |
| F4 | 349.23 | 3416 |
| F#4/Gb4 | 369.99 | 3224 |
| G4 | 392 | 3043 |
| G#4/Ab4 | 415.3 | 2873 |
| A4 | 440 | 2711 |
| A#4/Bb4 | 466.16 | 2559 |
| B4 | 493.88 | 2415 |
| C5 | 523.25 | 2280 |
| C#5/Db5 | 554.37 | 2152 |
| D5 | 587.33 | 2031 |
| D#5/Eb5 | 622.25 | 1917 |
| E5 | 659.25 | 1809 |
| F5 | 698.46 | 1708 |
| F#5/Gb5 | 739.99 | 1612 |
| G5 | 783.99 | 1521 |
| G#5/Ab5 | 830.61 | 1436 |
| A5 | 880 | 1355 |
| A#5/Bb5 | 932.33 | 1279 |
| B5 | 987.77 | 1207 |
| C6 | 1046.5 | 1140 |
| C#6/Db6 | 1108.73 | 1076 |
| D6 | 1174.66 | 1015 |
| D#6/Eb6 | 1244.51 | 958 |
| E6 | 1318.51 | 904 |
| F6 | 1396.91 | 854 |
| F#6/Gb6 | 1479.98 | 806 |
| G6 | 1567.98 | 760 |
| G#6/Ab6 | 1661.22 | 718 |
| A6 | 1760 | 677 |
| A#6/Bb6 | 1864.66 | 639 |
| B6 | 1975.53 | 603 |
| C7 | 2093 | 570 |
| C#7/Db7 | 2217.46 | 538 |
| D7 | 2349.32 | 507 |
| D#7/Eb7 | 2489.02 | 479 |
| E7 | 2637.02 | 452 |
| F7 | 2793.83 | 427 |
| F#7/Gb7 | 2959.96 | 403 |
| G7 | 3135.96 | 380 |
| G#7/Ab7 | 3322.44 | 359 |
| A7 | 3520 | 338 |
| A#7/Bb7 | 3729.31 | 319 |
| B7 | 3951.07 | 301 |
| C8 | 4186.01 | 285 |
| C#8/Db8 | 4434.92 | 269 |
| D8 | 4698.63 | 253 |
| D#8/Eb8 | 4978.03 | 239 |
| E8 | 5274.04 | 226 |
| F8 | 5587.65 | 213 |
| F#8/Gb8 | 5919.91 | 201 |
| G8 | 6271.93 | 190 |
| G#8/Ab8 | 6644.88 | 179 |
| A8 | 7040 | 169 |
| A#8/Bb8 | 7458.62 | 159 |
| B8 | 7902.13 | 150 |

## Jump tables

## Jump tables, also known as branch tables are an efficient method of transferring program control (branching) to another part of a program. based on specific conditions. Essentially, they consist of an array of addresses that the CPU can jump to, each corresponding to a unique condition or case in the program. For example, in languages like C, a switch statement is often implemented using a jump table, where each entry in the table directs the program flow to a particular case label.

## In assembly language, traditional jump instructions can require explicit condition checking and branching, which may become unwieldy for complex switch-like structures. To address this, alternative approaches can be employed. For instance, one approach involves using ASCII symbols, where the value associated with each symbol can be easily obtained by subtracting a character, offering a more efficient way to determine the target address for a jump.

## Lorem ipsum

# Development

# Code macros

**porDiez macro .**  This macro uses the stack to save the currently used registers, then it does the respective algorithm by doing n<<3 + n<<1 = n\*8 + n\*2 = n\*10 , ‘n’ being the DX register using only 14 Clock Cycles for the algorithm and 38 by the stack, but still less than the 118 – 133 Clock Cycles needed for 16 bit MUL operator.

porDiez MACRO

PUSH AX

PUSH CX

MOV CL,03h;dosbox gives error unless this

MOV AX,DX

SHL DX,CL

SHL AX,1

ADD DX,AX

POP CX

POP AX

ENDM

**sleep macro .**  This macro uses the 15h interruption with AH=86h which sleeps the given microseconds in a double word register logic CX:DX, so i first convert the given miliseconds to microseconds by dividing and multiplying by 10 and putting it in the double word registers.

sleep MACRO int16\_miliseconds

XOR CX,CX

MOV AX,int16\_miliseconds

MOV BX,10

XOR DX,DX

DIV BX

porDiez;DX\*10

MOV CX,DX

XOR DX,DX

DIV BX

ADD CX,DX

MOV DX,CX

porDiez;

porDiez;

porDiez;DX\*1000

MOV CX,AX

XOR AL,AL

MOV AH, 86H

INT 15H

ENDM

**beep macro .**  This macro set ups the speaker, loads the frecuency number to the 42h port, then it turn on the note by setting port 61h and then it sleeps by the given duration producing the beep sound, and finally it closes the the port 61h.

beep MACRO int16\_frequency, int16\_duration

mov al, 182 ; Prepare the speaker

out 43h, al

mov ax, int16\_frequency ; Load frequency number

out 42h, al ; Output low byte (least significant byte)

mov al, ah ; Output high byte (most significant byte)

out 42h, al

in al, 61h ; Turn on note

OR al,00000011b ; Set bits 1 and 0

out 61h, al

sleep int16\_duration

in al, 61h ; Turn off note

and al, 11111100b ; Reset bits 1 and 0

out 61h, al

ENDM

**kbhit macro .**  This aims to work as the kbhit() C function using the DOS keyboard interrupt 16h with AH=01h which will check for a keystroke in keyboard buffer.Then if a keystroke is present, it is not removed from the keyboard buffer.

kbhit MACRO

MOV AH, 01h

INT 16h

ENDM

**getch macro .**  This aims to work as the getch() C function using the DOS keyboard interrupt 16h with AH=00h which will get a keystroke from keyboard (no echo). Then if a keystroke is present, it is removed from the keyboard buffer.

getch MACRO ;return en AL

MOV AH,00h

INT 16h

ENDM

# Code logic

# Piano Procedure, this algorithm works inside of the kbhit and getch loop, when getch returns a value (ASCII character) a set of comparisons

# this function uses the following arrays:

# jumpTable\_numbers DW c\_0,c\_1,c\_2,c\_3,c\_4,c\_5,c\_6,c\_7,c\_8,c\_9,case\_default;

# jumpTable\_letters DW a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z,default;

# This arrays store the label in which the result of getch will jump to, to make a certain key play a piano note.

# The order in which the keys make a note are arranged in with the first part (C3 to B3) from the ‘q’ key to the ‘u’ key using the numbers from ‘1’ to ‘6’ except for ‘3’ for flat notes and the next part (C4 to B4) is from the ‘z’ key to the ‘m’ key using the keys from ‘a’ to ‘h’ except for ‘d’ for flat notes.

# Results

Lorem ipsum dolor sit

# Conclusions

Lo

# Bibliography

1. Akuyou, Keith. (2022). *Lesson P8 - Beeper speaker on MS DOS!.* Retrieved from <https://www.chibialiens.com/8086/platform.php#LessonP8>
2. Andezuthu D, Murugan. (2009). *8086 Assembly Language Program to Play Sound Using PC Speaker*. Retrieved from <http://muruganad.com/8086/8086-assembly-language-program-to-play-sound-using-pc-speaker.htm>
3. Bloom, Margaret. (2017). *Assembly 8086 - DOSBOX - How to produce beep sound?*. Retrieved from <https://stackoverflow.com/questions/43996835/assembly-8086-dosbox-how-to-produce-beep-sound>
4. Caldwell, Urtis. (2004). *Intel 8086 Instruction Timing. Methodist College*. Retrieved from <https://www.oocities.org/mc_introtocomputers/Instruction_Timing.PDF>
5. Cardoso, G. Célio. (2015). *Internal Speaker. Universidade Estadual de Campina*s. Retrieved from <https://www.ic.unicamp.br/~celio/mc404s102/pcspeaker/InternalSpeaker.htm>
6. Craig Stinson, Brad Kingsbury, et al. (2001). *The Assembly Language: database INT 15h, 86h (134) Wait XT-286, AT*. Retrieved from <http://vitaly_filatov.tripod.com/ng/asm/asm_026.13.html>
7. Evans, David. (2022). *x86 Assembly Guide.* University of Virginia. Retrieved from <https://www.cs.virginia.edu/~evans/cs216/guides/x86.html>
8. Hyde, Randall. (1996). *The Art of Assembly Language.* Retrieved from <https://www.randallhyde.com/AssemblyLanguage/www.artofasm.com/DOS/pdf/0_AoAPDF.html>
9. Jones, Nigel. (1999). *"Arrays of Pointers to Functions" Embedded Systems Programming*. Retrieved from https://barrgroup.com/blog/how-create-jump-tables-function-pointer-arrays-c-and-c
10. Preston, Robert. (2023). *What Is Assembly Language? (With Components and Example)*. Indeed. Retrieved from [https://www.indeed.com/career-advice/career-development/what-is-assembly-language#:~:text=An%20assembly%20language%20is%20a,the%20computer%20stores%20and%20reads.](https://www.indeed.com/career-advice/career-development/what-is-assembly-language#:~:text=An assembly language is a,the computer stores and reads.)
11. Richard R, Eckert. (2007). *The Intel 8253/8254 Programmable Interval Timer and Sound on a PC.* Binghamton University. Retrieved from <https://www.cs.binghamton.edu/~reckert/220/8254_timer.html>
12. Ruth, Anderson. (2017). *Assembly Programming IV.* University of Washington. Retrieved from <https://courses.cs.washington.edu/courses/cse351/17sp/lectures/CSE351-L10-asm-IV_17sp-ink.pdf>
13. Sechi, Marco. (n,d)*. Esercizio 6 (MicroChip 8253 -DA CONTROLLARE).* Università degli Studi di Brescia.Retrieved from <https://www.brescianet.com/appunti/sistemi/assembler/Esempi_ASM.htm>
14. Suits H, Bryan. (2022). *Physics of Music - Notes.* Michigan Technological University. Retrieved from <https://web.archive.org/web/20220124125158/https://pages.mtu.edu/~suits/notefreqs.html>
15. William R, Mark. (n.d). *x86 Control Flow*. University of Southern California. Retrieved from <https://ee.usc.edu/~redekopp/cs356/slides/CS356Unit5_x86_Control>