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.

1 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.

2 Theoretical Framework

2.1 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).

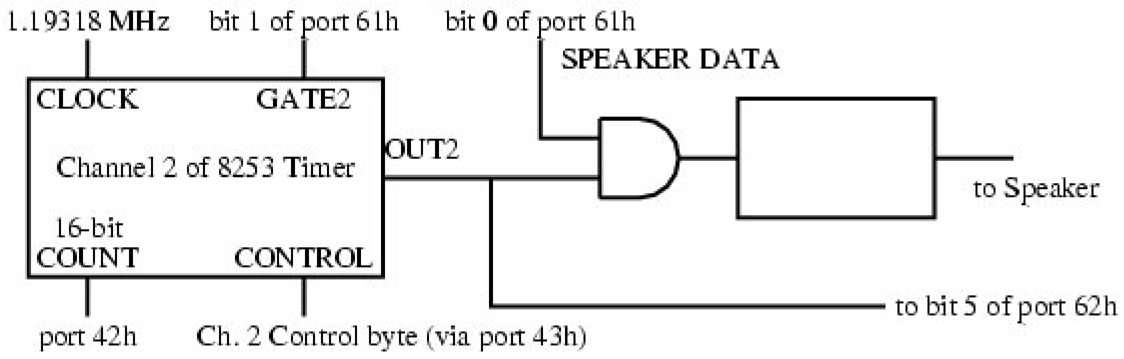


Fig. 1. Speaker

2.2 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.

2.3 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

$$Frequency\ number = \frac{1193180Hz}{Frequency} \quad \textbf{(1)}$$

$$Frequency = \frac{1193180Hz}{Frequency\ number} \quad \textbf{(2)}$$

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

Table 1. All possible notes for each frequency using the A = 440Hz scale

Note Frequency Numbers

Note	Frequency (Hz)	Frequency #
C^0	16.35	**72979 (Overflow)**
$C_\#^0/D_b^0$	17.32	**68890 (Overflow)**

D^0	18.35	65023
$D_\#^0/E_b^0$	19.45	61346
E^0	20.6	57921
F^0	21.83	54657
$F_\#^0/G_b^0$	23.12	51608
G^0	24.5	48701
$G_\#^0/A_b^0$	25.96	45962
A^0	27.5	43388
$A_\#^0/B_b^0$	29.14	40946
B^0	30.87	38651
C^1	32.7	36488
$C_\#^1/D_b^1$	34.65	34435
D^1	36.71	32502
$D_\#^1/E_b^1$	38.89	30680
E^1	41.2	28960
F^1	43.65	27335
$F_\#^1/G_b^1$	46.25	25798
G^1	49	24350
$G_\#^1/A_b^1$	51.91	22985
A^1	55	21694
$A_\#^1/B_b^1$	58.27	20476
B^1	61.74	19325
C^2	65.41	18241
$C_\#^2/D_b^2$	69.3	17217
D^2	73.42	16251
$D_\#^2/E_b^2$	77.78	15340
E^2	82.41	14478
F^2	87.31	13666
$F_\#^2/G_b^2$	92.5	12899
G^2	98	12175
$G_\#^2/A_b^2$	103.83	11491
A^2	110	10847
$A_\#^2/B_b^2$	116.54	10238
B^2	123.47	9663
C^3	130.81	9121
$C_\#^3/D_b^3$	138.59	8609
D^3	146.83	8126
$D_\#^3/E_b^3$	155.56	7670
E^3	164.81	7239
F^3	174.61	6833
$F_\#^3/G_b^3$	185	6449
G^3	196	6087
$G_\#^3/A_b^3$	207.65	5746
A^3	220	5423
$A_\#^3/B_b^3$	233.08	5119
B^3	246.94	4831
C^4	261.63	4560
$C_\#^4/D_b^4$	277.18	4304
D^4	293.66	4063
$D_\#^4/E_b^4$	311.13	3834

E^4	329.63	3619
F^4	349.23	3416
$F_\#^4/G_b^4$	369.99	3224
G^4	392	3043
$G_\#^4/A_b^4$	415.3	2873
A^4	440	2711
$A_\#^4/B_b^4$	466.16	2559
B^4	493.88	2415
C^5	523.25	2280
$C_\#^5/D_b^5$	554.37	2152
D^5	587.33	2031
$D_\#^5/E_b^5$	622.25	1917
E^5	659.25	1809
F^5	698.46	1708
$F_\#^5/G_b^5$	739.99	1612
G^5	783.99	1521
$G_\#^5/A_b^5$	830.61	1436
A^5	880	1355
$A_\#^5/B_b^5$	932.33	1279
B^5	987.77	1207
C^6	1046.5	1140
$C_\#^6/D_b^6$	1108.73	1076
D^6	1174.66	1015
$D_\#^6/E_b^6$	1244.51	958
E^6	1318.51	904
F^6	1396.91	854
$F_\#^6/G_b^6$	1479.98	806
G^6	1567.98	760
$G_\#^6/A_b^6$	1661.22	718
A^6	1760	677
$A_\#^6/B_b^6$	1864.66	639
B^6	1975.53	603
C^7	2093	570
$C_\#^7/D_b^7$	2217.46	538
D^7	2349.32	507
$D_\#^7/E_b^7$	2489.02	479
E^7	2637.02	452
F^7	2793.83	427
$F_\#^7/G_b^7$	2959.96	403
G^7	3135.96	380
$G_\#^7/A_b^7$	3322.44	359
A^7	3520	338
$A_\#^7/B_b^7$	3729.31	319
B^7	3951.07	301
C^8	4186.01	285
$C_\#^8/D_b^8$	4434.92	269
D^8	4698.63	253
$D_\#^8/E_b^8$	4978.03	239
E^8	5274.04	226
F^8	5587.65	213

$F_{\#}^{8}/G_{b}^{8}$	5919.91	201
G^{8}	6271.93	190
$G_{\#}^{8}/A_{b}^{8}$	6644.88	179
A^{8}	7040	169
$A_{\#}^{8}/B_{b}^{8}$	7458.62	159
B^{8}	7902.13	150

2.4 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.

3 Development

3.1 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 milliseconds to microseconds by dividing and multiplying by 10 and putting it in the double word registers.

```
sleep MACRO int16_milliseconds
         XOR CX,CX
         MOV AX,int16_milliseconds
         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 on/off macros. The beep_on macro set ups the speaker, loads the frequency number to the 42h port, then it turn on the note by setting port 61h and it will continue producing the beep sound until thee beep_off macro is called which closes the the port 61h and stops the sound.

```
beep_on MACRO
    MOV     AL, 182         ; Prepare the speaker
    OUT     43h, AL
    MOV     AX, BX          ; Load frequency number from BX
    OUT     42h, AL         ; Output low byte
    MOV     AL, AH          ; Output high byte
    OUT     42h, AL
    IN      AL, 61h         ; Turn on note
    OR      AL,00000011b    ; Set bits 1 and 0
    OUT     61h, AL
ENDM

beep_off MACRO
    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
```

print macro. This moves the OFFSET of the string we want to print to DX, and uses service 09 of INT 21h to print it on the screen.

```
print MACRO str_msg
    MOV AH, 09h
    LEA DX, str_msg
    INT 21h
ENDM
```

PutchC macro. This macro is used to print a character with a specific attribute on the screen. The parameters are the character to print (char_ch), the color attribute (color), the number of times to print the character (n), and the display page (page). The steps are as follows: Move 09h to register AH to prepare for character output with attribute. Move the character to be printed into register AL. Move the display page into register BH. Move the color attribute into register BL. Move the count of how many times to print the character into register CX. Call interrupt 10h to execute the print.

```
putchC MACRO char_ch,color,n,page
    MOV AH,09h
    MOV AL,char_ch
```

```
        MOV BH,page
        MOV BL,color
        MOV CX,n
        INT 10h
    ENDM
```

Gotoxy macro. This macro is used to position the cursor on the screen. The parameters are the x-coordinate (x), the y-coordinate (y), and the display page (page). The steps are as follows: Move the y-coordinate into register DH. Move the x-coordinate into register DL. Move the display page into register BH. Move 02h to register AH to prepare for cursor positioning. Call interrupt 10h to execute the cursor position.

```
    gotoxy MACRO x,y,page
        MOV DH,y
        MOV DL,x
        MOV BH,page
        MOV AH,02h
        INT 10h
    ENDM
```

scrollUp macro. This macro scrolls up a portion of the screen. The parameters are the top row (filSup), the top column (colSup), the bottom row (filInf), the bottom column (colInf), and the color attribute (color). The steps are as follows: Move 06 to register AH to prepare for scrolling. Move 0 to register AL to specify the number of lines to scroll. Move the color attribute into register BH. Move the top row into register CH. Move the top column into register CL. Move the bottom row into register DH. Move the bottom column into register DL. Call interrupt 10h to execute the scroll.

```
    scrollUP MACRO filSup, colSup, filInf, colInf, color
        MOV AH,06
        MOV AL,0
        MOV BH,color
        MOV CH,filSup
        MOV CL,colSup
        MOV DH,filInf
        MOV DL,colInf
        INT 10h
    ENDM
```

3.2 Code logic

Jump tables control logic, to go to the correct label in the program i first store the labels in jump table arrays, and as an array it goes from index 0 to the last index (which is the size of the array minus one), then the input char must be transform into a numeric value. To transform to the corresponding index, the numeric value must be multiply by two because the memory address are data words of sixteen bits, then we compare if the index is bigger than the last index, if it is then it goes to the default case, else to the corresponding index. This is much faster than branching the program with multiple compares.

Main Procedure, this is the main driver code which has a loop and a jump table (jumpTable_menu) to jump in to the corresponding procedure the number one is the piano Procedure, two is the jukebox Procedure, and three is the option to exit the program and return control to the operating system, and any other will return to the loop.

Piano Procedure, this procedure has a loop which has a kbhit and getch, when a key is pressed the kbhit macro ends and calls getch then, getch returns a value (ASCII character) to be used in the following jump table 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 two 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.

Jukebox Procedure, this procedure has a loop and it make you pick between four songs , being: Mario, Zelda, Fur Elise and Despacito respectively, or exiting from this menu, then it uses a jump table (jumpTable_jukebox) which plays the corresponding song using the beep procedure by changing the delay variable (int16_delay).

Beep_proc Procedure, this procedure uses the beep_on macro to turn on the port with the frecuency number in the BX register, and uses the sleep macro to sleep for the duration of the int16_delay variable, then it turns off the sound with the beep_off macro

Menu Procedure, this procedure activates the 10h video mode and prints the strings from the main menu. It uses the gotoxy macro to position the cursor on the screen, the print macro to write the string on the screen, the imprimirColor procedure to print a string with attribute.

PantallaPiano Procedure, this procedure is responsible for printing the piano on the screen. First, it changes the page using service 05 of INT 10h, then using the ScrollUp macro, it paints a white rectangle in the center of the screen. Next, the same macro is called several times to paint black lines on the rectangle to simulate the separation of the keys and the sharp keys. Using the gotoxy macro and the imprimirColor procedure, they position the cursor on each of the keys and prints a string with the name of the note as well as the key to press.

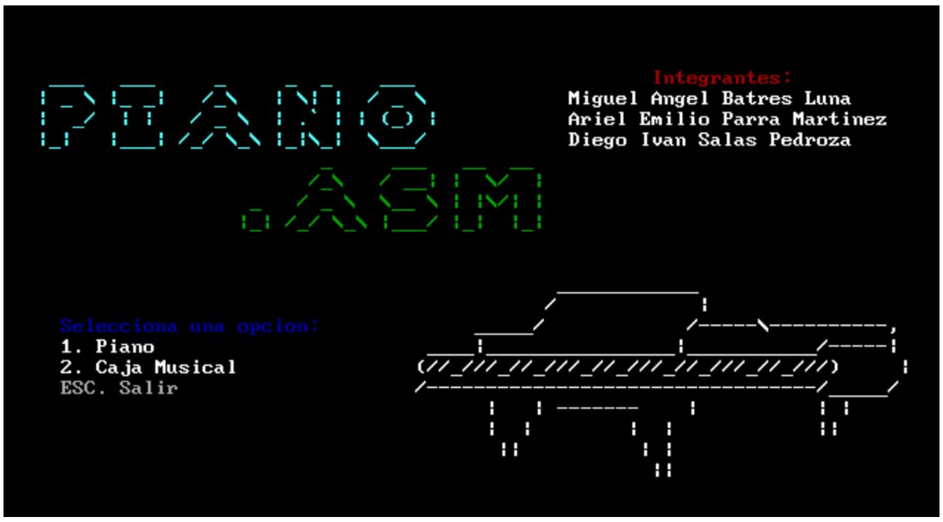
MenuCanciones Procedure, this procedure prints on the screen the repertoire of songs added to the project. . First, it changes the page using service 05 of INT 10. Then, it uses the gotoxy macro to position the cursor on the screen, the putchC macro to print a single character with an attribute on the screen, and the imprimirColor procedure to print the strings on this screen.

ImprimirColor Procedure, this procedure prints a string with an attribute. The value for each attribute is stored in different variables: black, darkBlue, green, lightBlue, red, purple, brown, gray, darkGray, blue, limeGreen, skyBlue, lightRed, pink, yellow, white. Before calling this procedure, we move the color variable we want for the string into BL. Then, we move the OFFSET of the string we want to print into SI using the LEA instruction. Once the procedure is called, the value of the index SI is incremented, as well as the cursor position, to print the string character by character with its atribute.

3.3 Process to make songs

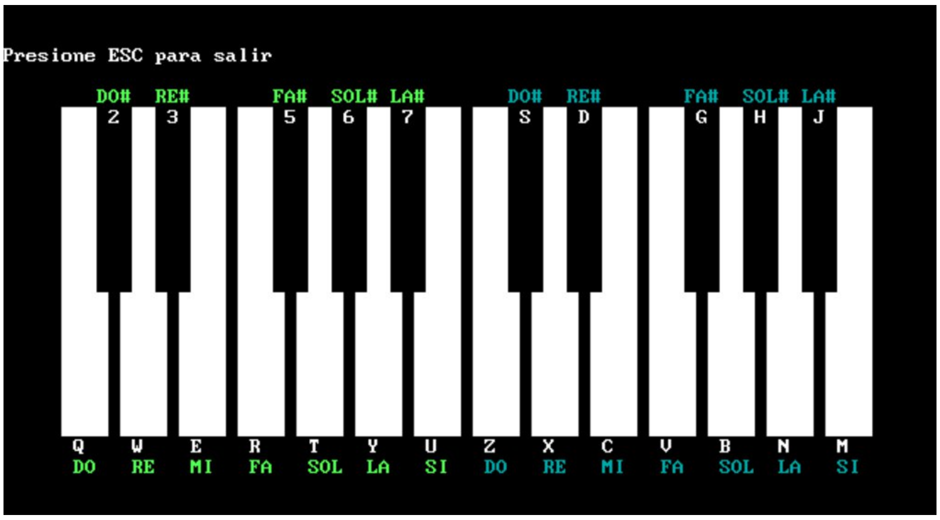
To make each song, we needed each piano note, duration of the note and time delays. We search all around the internet to find some songs which contained notes and durations to then make each song by hand in the song.inc file which contains each song macro (song_1, song_2,song_3,song_4) Mario, Zelda, Fur Elise and Despacito respectively.

4 Results


```
Integrantes:
Miguel Angel Batres Luna
Ariel Emilio Parra Martinez
Diego Ivan Salas Pedroza

Selecciona una opcion:
1. Piano
2. Caja Musical
ESC. Salir
```

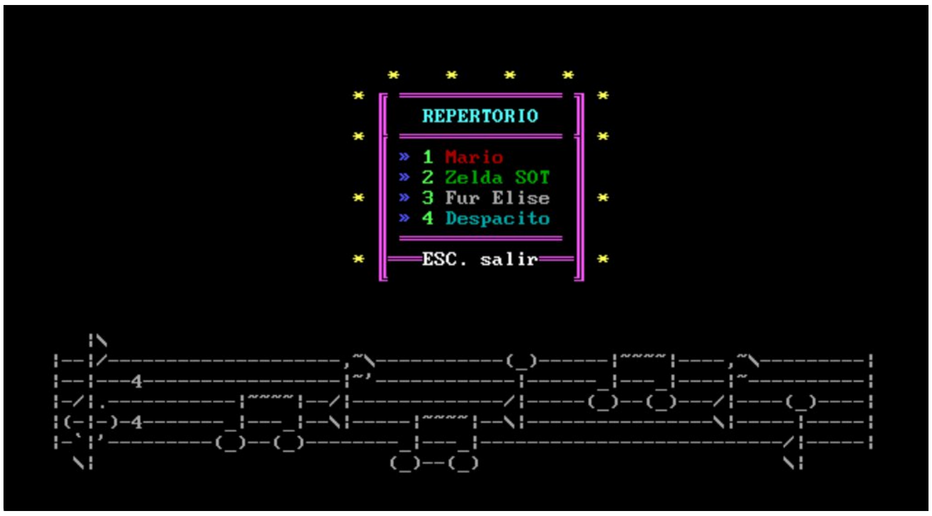
Fig. 2. Main Menu



```
Presione ESC para salir
```

DO# RE# FA# SOL# LA# DO# RE# FA# SOL# LA#
2 3 5 6 7 S D G H J
Q W E R T Y U Z X C V B N M
DO RE MI FA SOL LA SI DO RE MI FA SOL LA SI

Fig. 3. Piano Screen



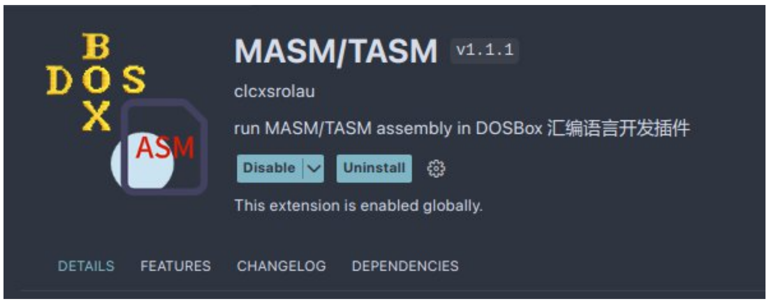
```
REPERTORIO
» 1 Mario
» 2 Zelda SOT
» 3 Fur Elise
» 4 Despacito
ESC. salir
```

Fig. 4. Songlist Menu

4.1 Compiling the project

To compile the project there are two routes, download dosbox and vscode with an extension, or download dosbox and the TASM or MASM compiler, the first rout is download the dosbox and vscode program as seen in Fig. 2 , then change the settings seen in Fig. 3 . For the other route we download dosbox and put the TASM or MASM compiler in the emulated C drive.

For both routes to compile is as simple as opening the dos console and typing “MASM MAIN.ASM” then “LINK MAIN” for the MASM compiler or for the TASM compiler you need to type “TASM MAIN.ASM” and then “TLINK MAIN”, in both cases you will end up with a “MAIN.EXE” file.



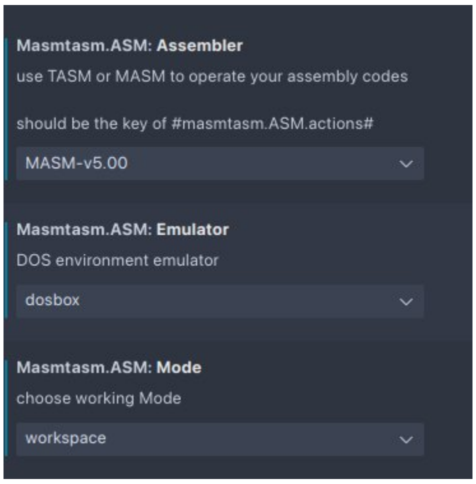
MASM/TASM v1.1.1

clcxsrolau

run MASM/TASM assembly in DOSBox 汇编语言开发插件

This extension is enabled globally.

Fig. 5. Vscode extension



Masmtasm.ASM: Assembler
use TASM or MASM to operate your assembly codes

should be the key of #masmtasm.ASM.actions#

MASM-v5.00

Masmtasm.ASM: Emulator
DOS environment emulator

dosbox

Masmtasm.ASM: Mode
choose working Mode

workspace

Fig. 6 MASM/TASM extension settings

4.2 Execution of the project

To execute the program you need to be at the dos terminal with the project already compiled, then type the relative route to the program in this case is “.\MAIN.EXE” wich can be typed in lower or upper case, then pressing enter or if the MASM/TASM extension is insatlled you just need to right click the main.assembly code in Vscode and click in “un ASM code” which will compile and execute the program. When the program is running you will se the first menu as in **Fig. 2.** , then the first option is the piano in **Fig. 3** .an finally the jukebox (songlist menu) on **Fig. 4.**

5 Conclusions

Batres L. Miguel's Conclusion
The completion of this assembly language project, which involved designing a piano program for the Intel 8086 processor using the DOS operating system, highlights the profound interplay between software and hardware. Through meticulous manipulation of the CPU's registers and direct hardware interfacing, we successfully generated melodies via the computer's speaker. This project underscores the power and versatility of assembly language, showcasing its capability to achieve precise control over hardware components.

Moreover, this endeavor demonstrated the critical importance of teamwork in overcoming complex technical challenges. The collaborative efforts of the team brought together diverse skills and perspectives, enhancing problem-solving strategies and fostering innovation. Effective communication and coordination were key in navigating the intricacies of low-level programming and achieving our project goals. Ultimately, this project not only enhanced our technical expertise in assembly language and 8086 architecture but also reinforced the value of collaboration in driving technological advancements.

Salas P. Diego's Conclusion
Participating in this 8086 assembler project, I was able to acquire more knowledge about the different video modes in this language, as well as the BIOS 10h interrupts. By leveraging BIOS interrupts and custom macros, the project efficiently manages screen output and user interaction, showcasing the versatility and power of assembler language. Key features of this project include the use of macros for precise cursor positioning and character printing, as well as advanced screen manipulation techniques to create dynamic and interactive elements. The consistent use of color and structured design enhances the user experience, making the interface visually appealing and functional.

Parra M. Ariel's Conclusion

The team has shown effective problem-solving skills and innovation in tackling the complex technological challenges. The program’s functionality highlights the flexibility of the assembly language. Over all, this project serves as a testament that to the power not only the power of the assembly language, but the power of teamwork, working on a complex task as this piano. I’ve learned a lot of low level programming like the use of bit masking and bit-wise operators, this knowledge will not only work in assembly but in some higher level languages like C or C++. In conclusion this has been an enriching experience that taught me a lot project and time management to complete the project on time with all the requirements fulfilled.

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. Digital iVision Labs. *(2013). C++ Code For Mario Theme & Intro Song (Interesing C++ Project Code)*. Retrieved from http://cncpp.divilabs.com/2013/12/c-code-for-mario-theme-intro-song.html
8. Evans, David. (2022). *x86 Assembly Guide.* University of Virginia. Retrieved from https://www.cs.virginia.edu/~evans/cs216/guides/x86.html
9. Gluschenko, A. (2018). *Imperial.cpp* . Retrieved from https://gist.github.com/gluschenko/4ff8bb49802dc848626091ff14704112
10. Hyde, Randall. (1996). *The Art of Assembly Language.* Retrieved from https://www.randallhyde.com/AssemblyLanguage/www.artofasm.com/DOS/pdf/0_AoAPDF.html
11. 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
12. Magno, R. (2021). *zeldab*. Retrieved from https://github.com/raymag/zeldab/blob/main/src/zelda.c
13. n3m351d4. (2020.). *Despacito.ino.* Retrieved from https://github.com/n3m351d4/SongsForBeeper/blob/master/Despacito.ino
14. 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.
15. 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
16. 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
17. 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
18. 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
19. William R, Mark. (n.d). *x86 Control Flow*. University of Southern California. Retrieved from https://ee.usc.edu/~redekopp/cs356/slides/CS356Unit5_x86_Control