

LENGUAJE DE INTERFAZ

INSTITUTO TECNOLÓGICO DE MINATITLÁN

CAMPUS MINATITLÁN

Clave: ISC-2240

INGENIERÍA EN SISTEMAS COMPUTACIONALES

DOCENTE: MARIA CONCEPCIÓN VILLATORO CRUZ

ALUMNA: CRUZ MISS YESSICA YAMILET

PORTAFOLIO DE EVIDENCIA



TEMA 1: INTRODUCCIÓN AL LENGUAJE ENSAMBLADOR

INTRODUCCIÓN GENERAL

La introducción al lenguaje ensamblador es un paso fundamental en la comprensión de la programación de bajo nivel y la arquitectura de computadoras. Este lenguaje, a menudo conocido como "assembly language" en inglés, es un lenguaje de programación de bajo nivel que se encuentra entre el lenguaje de máquina y los lenguajes de alto nivel. En lugar de utilizar palabras binarias o hexadecimales directamente, el lenguaje ensamblador utiliza mnemotécnicos y abreviaturas para representar instrucciones específicas que la CPU puede entender.

El lenguaje ensamblador es altamente dependiente de la arquitectura de la computadora en la que se ejecuta y, por lo tanto, no es portátil entre diferentes tipos de CPU. Cada arquitectura de CPU tiene su propio conjunto único de instrucciones y modos de direccionamiento. Esto significa que para programar en lenguaje ensamblador, es necesario conocer la arquitectura específica de la máquina en la que se está trabajando.

A pesar de su complejidad y falta de portabilidad, el lenguaje ensamblador sigue siendo importante en la programación de sistemas, el desarrollo de controladores de hardware y en situaciones donde se requiere un control muy preciso del hardware de una computadora. Además, aprender lenguaje ensamblador puede ayudar a los programadores a comprender mejor cómo funciona una computadora a nivel de bajo nivel, lo que puede ser beneficioso en la optimización de código y la resolución de problemas.

En esta introducción, exploraremos los conceptos básicos del lenguaje ensamblador, incluyendo su estructura, instrucciones, modos de direccionamiento y cómo se relaciona con la arquitectura de la CPU. También discutiremos la importancia de este lenguaje en la programación y la resolución de problemas a nivel de bajo nivel.

ACTIVIDAD 1. MI CPU: DATOS

INTRODUCCIÓN ACTIVIDAD 1

El funcionamiento y la estructura de una CPU (Unidad Central de Procesamiento) son fundamentales para comprender cómo se procesan los datos en una computadora. La CPU es el "cerebro" de la computadora y realiza la mayor parte del procesamiento de datos. A continuación, te proporcionaré una introducción detallada sobre cómo funciona una CPU en términos de procesamiento de datos y el viaje que realiza un dato a través de ella.

Estructura de la CPU:

Una CPU consta de varias unidades funcionales clave que trabajan juntas para procesar datos. Estas unidades incluyen la Unidad de Control, la Unidad Aritmético-Lógica (ALU), los Registros y las Unidades de Manejo de Memoria.

1. Unidad de Control (CU): La Unidad de Control es responsable de coordinar todas las operaciones de la CPU. Interpreta las instrucciones del programa y controla la secuencia de ejecución. La CU envía señales a otras unidades para realizar operaciones específicas.
2. Unidad Aritmético-Lógica (ALU): La ALU es donde ocurren las operaciones matemáticas y lógicas. Suma, resta, multiplica, divide y realiza operaciones lógicas (AND, OR, NOT, etc.) en los datos. El resultado se almacena nuevamente en los registros.
3. Registros: Los registros son pequeñas unidades de almacenamiento dentro de la CPU que se utilizan para almacenar datos temporales, direcciones de memoria y otros valores necesarios para la ejecución de instrucciones. Los registros son extremadamente rápidos y facilitan la transferencia de datos dentro de la CPU.
4. Unidades de Manejo de Memoria: Estas unidades se encargan de acceder a la memoria principal (RAM) para leer y escribir datos. Incluyen el Registro de Dirección de Memoria (MAR) y el Registro de Datos de Memoria (MDR), que se utilizan para leer y escribir datos en la memoria.

El viaje de un dato a través de la CPU:

Cuando un programa se ejecuta, los datos se mueven a través de la CPU en un proceso que involucra varias etapas. Aquí está un resumen del viaje de un dato a través de la CPU:

1. Recuperación de instrucciones: El programa se almacena en la memoria y se carga en la CPU. La Unidad de Control (CU) recupera la siguiente instrucción del programa desde la memoria.
2. Decodificación de instrucción: La CU interpreta la instrucción y determina qué operación debe realizarse y en qué datos.
3. Acceso a registros y memoria: Si la instrucción implica la lectura o escritura de datos, la CPU accede a los registros y/o la memoria para obtener o almacenar los datos relevantes.
4. Procesamiento en la ALU: Los datos se envían a la ALU, donde se realizan las operaciones matemáticas o lógicas requeridas.
5. Almacenamiento de resultados: El resultado de la operación se almacena en registros o se escribe de nuevo en la memoria si es necesario.
6. Ciclo de reloj: Todo este proceso ocurre en ciclos de reloj, que son pulsos de reloj sincronizados que controlan la velocidad de ejecución de la CPU. Cada ciclo de reloj representa una etapa de procesamiento.

El viaje de un dato a través de la CPU puede involucrar múltiples ciclos de reloj, dependiendo de la complejidad de la operación. La CPU trabaja a una velocidad extremadamente rápida, realizando miles de millones de operaciones por segundo, lo que la convierte en el corazón de la computadora y la encargada de procesar datos de manera eficiente.

MI CPU

FUNCIONAMIENTO Y ESTRUCTURA



CPU (Unidad Central de Procesamiento)

Tambien conocido como procesador, es el componente principal de una computadora y desempeña un papel fundamental en el procesamiento de datos y la ejecución de instrucciones.

01

Estructura del CPU



Empieza con:

Las principales partes son:

1. Unidad de Control (CU, Control Unit)
2. Unidad Aritmético-Lógica (ALU, Arithmetic Logic Unit)
3. Registros
4. Unidad de Memoria (Memory Unit)



02

Funcionamiento del CPU

Ciclo de búsqueda-ejecución

01

Búsqueda (Fetch)

Instrucción en la memoria RAM.

02

Decodificación (Decode)

Decodificación a datos para instrucción.

03

Ejecución (Execute)

Ejecuta instrucciones aritméticas, lógicas, etc.

04

Almacenamiento (Write Back)

Altera datos en los registros de la CPU.

03

Dato a través del CPU



**El proceso de ejecución de una
instrucción y el viaje de datos a través de
una computadora involucran múltiples
etapas:**

1. Instrucción en memoria.
2. Búsqueda de instrucción.
3. Decodificación.
4. Ejecución.
5. Almacenamiento de resultados

RÚBRICA DE EVALUACIÓN – CALIFICACIÓN

Tipo de presentación	Su presentación incluyo imágenes, video o algun recurso creativo y novedoso <i>3 puntos</i>	Su presentación fue muy básica <i>1 puntos</i>	Su presentación no represento un esfuerzo de mejora en el equipo <i>0 puntos</i>
Contenido	El contenido fue claro y novedoso <i>5 puntos</i>	El contenido fue escencial y básico <i>3 puntos</i>	El contenido no era el esperado <i>1 puntos</i>
Claridad en la exposición	La exposición fue ejecutada con un grado de preparación y claridad excelente <i>5 puntos</i>	La claridad de la exposición fue básica <i>3 puntos</i>	No fue claro al momento de exponer, las ideas no tenian una secuencia <i>0 puntos</i>
Creatividad	Las herramientas utilizdas no demostraron creatividad para mantener la curiosidad en el grupo <i>4 puntos</i>	La exposición fue plana sin algun elemento creativo interesante <i>2 puntos</i>	No hubo creatividad en la exposición <i>0 puntos</i>
Motivación en la presentación	Lograron mantener la atención de sus compañeros <i>3 puntos</i>	La presentación fue muy plana <i>2 puntos</i>	No lograron motivar a sus compañeros, haciendo que la exposición no importara <i>1 puntos</i>
27.00 / 30.00			

CONCLUSIÓN ACTIVIDAD 1

En conclusión, el funcionamiento y la estructura de una CPU (Unidad Central de Procesamiento) son esenciales para entender cómo se procesan los datos en una computadora. La CPU, como el núcleo de la máquina, consta de varias unidades funcionales clave, como la Unidad de Control (CU), la Unidad Aritmético-Lógica (ALU), los Registros y las Unidades de Manejo de Memoria, que trabajan en conjunto para realizar el procesamiento de datos.

El viaje de un dato a través de la CPU es un proceso que involucra varias etapas, desde la recuperación de instrucciones desde la memoria hasta la ejecución de operaciones en la ALU y el almacenamiento de resultados. Este proceso se lleva a cabo en ciclos de reloj sincronizados que determinan la velocidad de la CPU.

La CPU desempeña un papel crítico en la ejecución de programas y operaciones de procesamiento de datos en una computadora. Comprender su funcionamiento y estructura es fundamental para programadores y profesionales de la informática, ya que permite optimizar el rendimiento del sistema y solucionar problemas a nivel de bajo nivel. En resumen, la CPU es el cerebro de la computadora y desempeña un papel central en el procesamiento de datos en todos los dispositivos informáticos.

ACTIVIDAD 2.

EMU8086

INTRODUCCIÓN DEL MICROPROCESADOR 8086

El cerebro del ordenador es el procesador. Su función es ejecutar los programas almacenados en la memoria principal tomando las instrucciones, examinándolas y ejecutándolas una tras otra. Para ello, el procesador realiza todas las operaciones aritméticas, lógicas y de control que son necesarias.

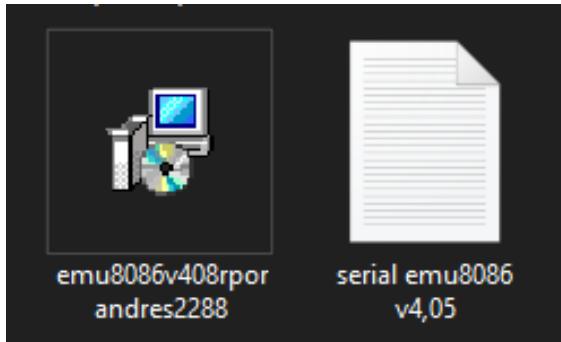
El microprocesador EMU 8086 es una herramienta fundamental en el mundo de la informática, conocida por su capacidad para emular el funcionamiento del procesador Intel 8086 en entornos de software. El Intel 8086 fue uno de los primeros microprocesadores de 16 bits ampliamente utilizados en la década de 1970 y sentó las bases para la arquitectura x86 que ha dominado la industria de la computación durante décadas.

EMU 8086, por otro lado, es un emulador de software diseñado para recrear la funcionalidad del Intel 8086 en sistemas modernos. Permite a los programadores y entusiastas de la informática experimentar con el lenguaje de programación Assembly y desarrollar aplicaciones que aprovechan la arquitectura x86 de 16 bits. Además, EMU 8086 es una herramienta educativa valiosa que ayuda a comprender los conceptos fundamentales de la programación de bajo nivel y la arquitectura del procesador.

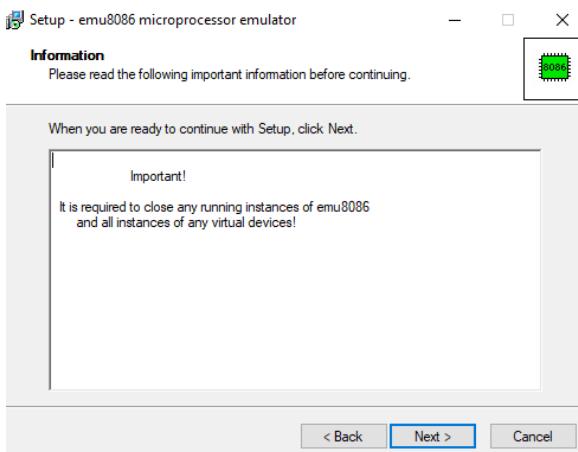
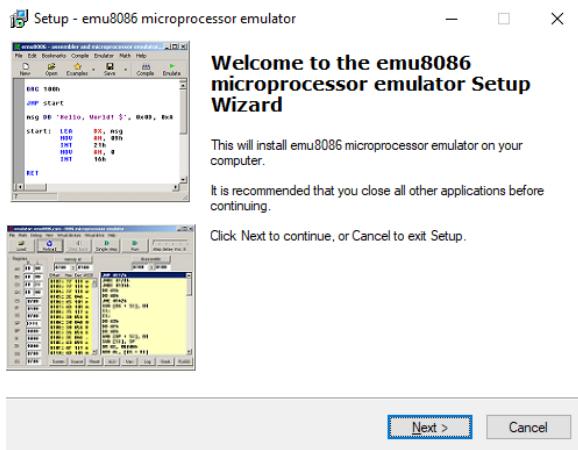
En esta introducción, exploraremos las características y el uso del microprocesador EMU 8086 en una muy sencilla practica para demostrar cómo funciona esta herramienta en el aprendizaje y desarrollo del software.

INSTALACIÓN Y NAVEGACIÓN DEL EMU 8086

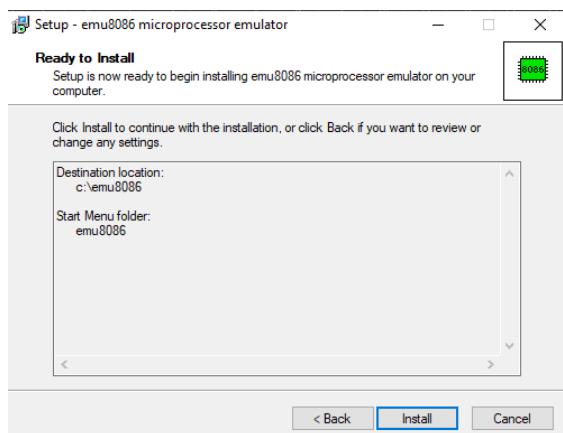
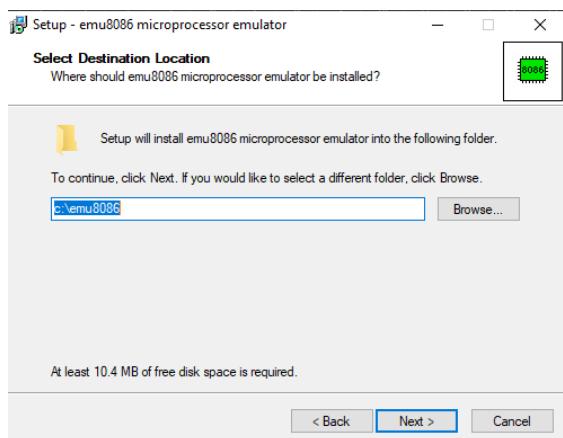
Obtener la aplicación para instalar el Emulador en la computadora.



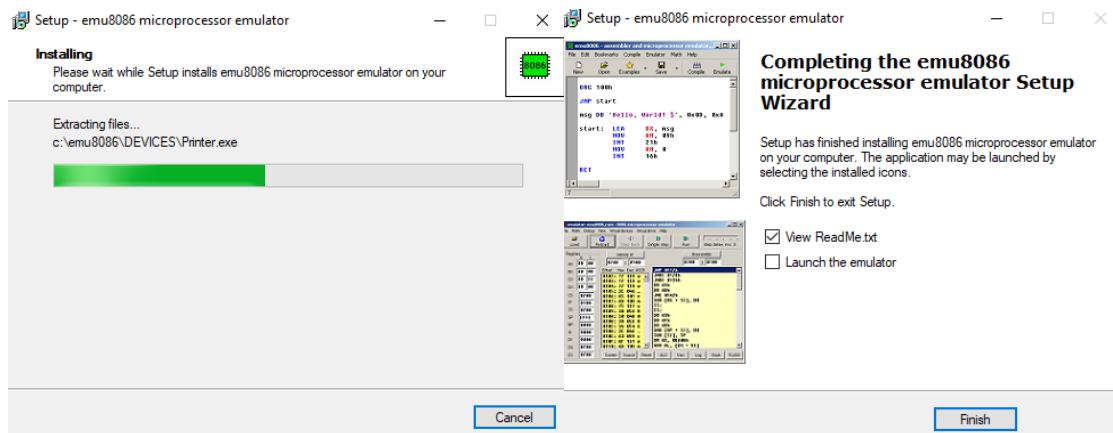
Al doble clic sobre el “Emu8086” aparecerá esta pantalla.



Escoger el lugar donde se guardará la carpeta de archivos.



Y se instala inmediatamente.



 ReadMe: Bloc de notas

Archivo Edición Formato Ver Ayuda

EMU8086 - THE MICROPROCESSOR EMULATOR

Introduction

=====

emu8086 is the emulator of 8086 (Intel and AMD compatible) microprocessor and integrated assembler with 8086 assembly language.

The emulator runs programs like the real microprocessor in step-by-step mode. It shows registers, memory, stack, variables and flags. All memory locations can be modified by double click. The instructions can be executed back and forward.

emu8086 can create a tiny operating system and write its binary code to a bootable floppy disk. The software package includes several external virtual devices: robot, stepper motor, led display, and traffic lights intersection. Additional devices can be created.

REQUIREMENTS

=====

Administrative rights for Windows XP/Vista/7 users.
10 MB of hard disk space and 1024x768 or greater screen resolution.

COPYRIGHTS

=====

Portions Copyright 1997-2010 Barry Alyn. All rights reserved.
Flat Assembler version 1.64
Copyright (c) 1999-2010, Tomasz Gryzstar.
All rights reserved.
Copyright (c) 2010 by EMU8086.COM All rights reserved.

The information available in reference is copyrighted unless otherwise indicated. You are allowed to reproduce and copy or translate information from this software, only under the following conditions:

1. Use of such information includes our copyright notice.
2. Use of such information includes a hyperlink to emu8086.com website.

CONTACT

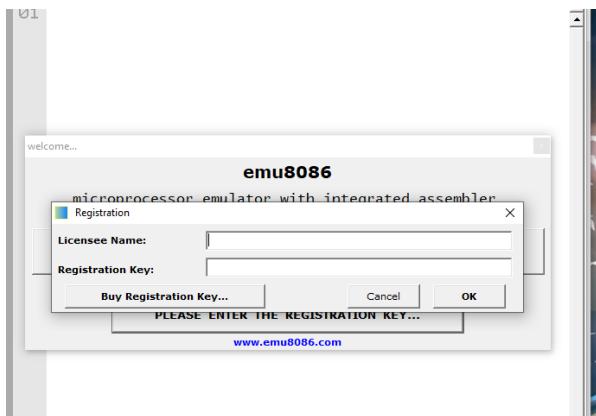
=====

Yuri Margolin
35 Herzl Street
33564 Haifa
Israel
Tel: +972 52 4663634
e-mail: info@emu8086.com
<http://www.emu8086.com>

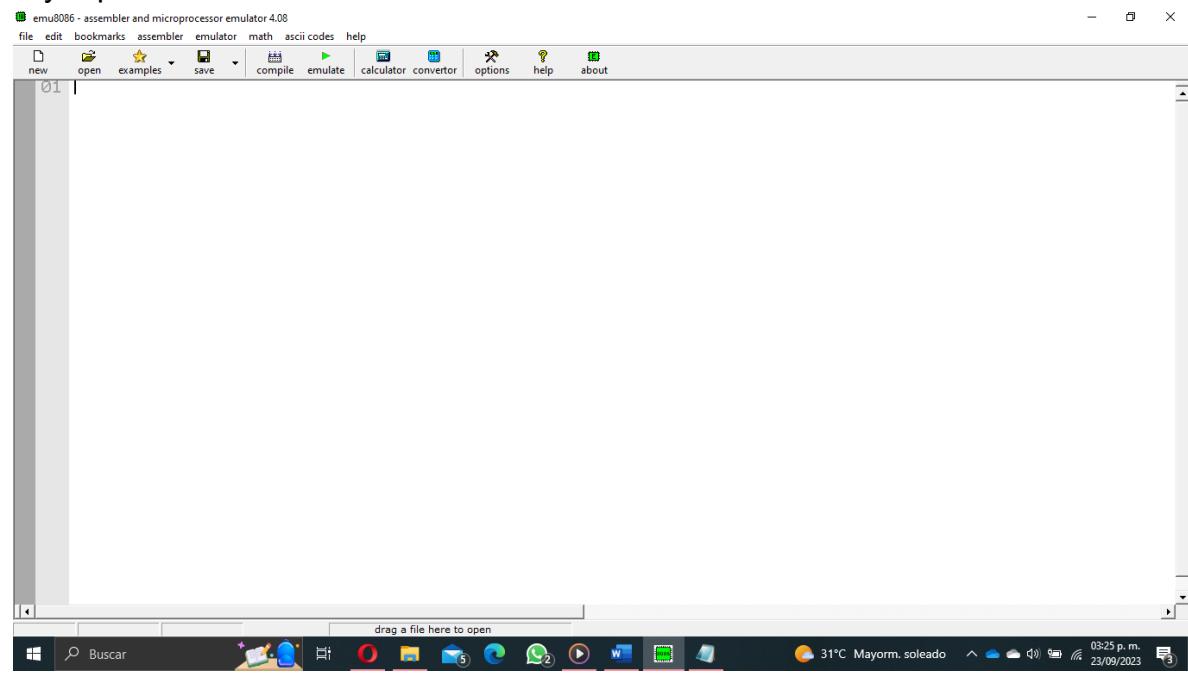
Una vez instalado en la PC, se tiene que iniciar.



Y se pone la clave de acceso para poder trabajar un poco más a gusto.



Y ya quedo listo



PROGRAMA CON TIPO DE DIRECCIONAMIENTO, REGISTROS Y ESTRUCTURA DE 8086

1. Calculadora

The image shows two windows of the Emu8086 IDE. Both windows have a title bar 'edit: C:\emu8086\examples\Calculator.asm' and a menu bar with 'file', 'edit', 'bookmarks', 'assembler', 'emulator', 'math', 'ascii codes', and 'help'. The top window displays assembly code for a simple calculator. The bottom window displays assembly code for a more complex calculator program involving strings and memory operations.

Top Window (Calculator.asm):

```
001 name "calc2"
002
003 ; command prompt based simple calculator (+,-,*,/) for 8086.
004 ; example of calculation:
005 input 1 <- number: 10
006 input 2 <- operator: -
007 input 3 <- number: 5
008 -----
009 ----- 10 - 5 = 5
010 ; output -> number: 5
011
012
013
014
015
016
017 ;;; this macro is copied from emu8086.inc ;;;
018 ;;; this macro prints a char in AL and advances
019 ; the current cursor position:
020 PUTC MACRO char
021     PUSH AX
022     MOV AH, char
023     MOV AL, 0Eh
024     INT 10h
025     POP AX
026 ENDM
027
028 ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;; ;;;
029
030
031
032
033
034 org 100h
035 jmp start
036
037
038
```

Bottom Window (Calculator.asm):

```
039 ; define variables:
040
041 msg0 db "note: calculator works with integer values only.",0Dh,0Ah
042 db "to learn how to output the result of a float division see float.asm in examples",0Dh,0Ah
043 msg1 db 0Dh,0Ah, 0Dh,0Ah,'enter first number: $'
044 msg2 db 'enter the operator: + - * / : $'
045 msg3 db "enter second number: $"
046 msg4 db 0Dh,0Ah,'the approximate result of my calculations is : $'
047 msg5 db 0Dh,0Ah,'thank you for using the calculator! press any key... ', 0Dh,0Ah, '$'
048 err1 db "wrong operator!", 0Dh,0Ah ,$', '$'
049 smth db " and something... $"
050
051 ; operator can be: '+','-', '*', '/' or 'q' to exit in the middle.
052 opr db '?'
053
054 ; first and second number:
055 num1 dw ?
056 num2 dw ?
057
058
059 start:
060 mov dx, offset msg0
061 mov ah, 9
062 int 21h
063
064
065 lea dx, msg1
066 mov ah, 09h ; output string at ds:dx
067 int 21h
068
069 ; get the multi-digit signed number
070 ; from the keyboard, and store
071 ; the result in cx register:
072 call scan_num
073
074
075
```

edit C:\emu8086\examples\Calculator.asm

file edit bookmarks assembler emulator math ascii codes help

new open examples save compile emulate calculator converter options help about

```

076 ; store first number:
077 mov num1, cx
078
079
080
081 ; new line:
082 putc 0Dh
083 putc 0Ah
084
085
086
087
088 lea dx, msg2
089 mov ah, 09h      ; output string at ds:dx
090 int 21h
091
092 ; set operator:
093 mov ah, 1        ; single char input to AL.
094 int 21h
095 mov opr, al
096
097
098 ; new line:
099 putc 0Dh
100 putc 0Ah
101
102 cmp opr, 'q'    ; q - exit in the middle.
103 je exit
104
105 cmp opr, '*'
106 jb wrong_opr
107 cmp opr, '/'
108 ja wrong_opr
109
110
111
112
113

```

drag a file here to open

Buscar 31°C Soleado 03:37 p.m. 23/09/2023

edit C:\emu8086\examples\Calculator.asm

file edit bookmarks assembler emulator math ascii codes help

new open examples save compile emulate calculator converter options help about

```

119 ; output of a string at ds:dx
120 lea dx, msg3
121 mov ah, 09h
122 int 21h
123
124 ; get the multi-digit signed number
125 ; from the keyboard, and store
126 ; the result in cx register:
127
128 call scan_num
129
130
131 ; store second number:
132 mov num2, cx
133
134
135
136
137 lea dx, msg4
138 mov ah, 09h      ; output string at ds:dx
139 int 21h
140
141
142
143
144 ; calculate:
145
146
147
148
149
150 cmp opr, '+'
151 je do_plus
152
153 cmp opr, '-'
154 je do_minus
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
759

```

drag a file here to open

Buscar Pronóstico de vientos 03:38 p.m. 23/09/2023

edit C:\emu8086\examples\Calculator.asm

file edit bookmarks assembler emulator math ascii codes help

new open examples save compile emulate calculator converter options help about

```

149
150     cmp opr, '+'
151     je do_plus
152
153     cmp opr, '-'
154     je do_minus
155
156     cmp opr, '*'
157     je do_mult
158
159     cmp opr, '/'
160     je do_div
161
162
163 ; none of the above....
164 wrong_opr:
165     lea dx, err1
166     mov ah, 09h      ; output string at ds:dx
167     int 21h
168
169
170 exit:
171 ; output of a string at ds:dx
172     lea dx, msg5
173     mov ah, 09h
174     int 21h
175
176
177 ; wait for any key...
178     mov ah, 0
179     int 16h
180
181
182
183 ret ; return back to os.
184
185

```

edit C:\emu8086\examples\Calculator.asm

file edit bookmarks assembler emulator math ascii codes help

new open examples save compile emulate calculator converter options help about

```

195 do_plus:
196
197     mov ax, num1
198     add ax, num2
199     call print_num    ; print ax value.
200
201     jmp exit
202
203
204
205 do_minus:
206
207     mov ax, num1
208     sub ax, num2
209     call print_num    ; print ax value.
210
211     jmp exit
212
213
214
215 do_mult:
216
217     mov ax, num1
218     imul num2 ; (dx ax) = ax * num2
219     call print_num    ; print ax value.
220     ; dx is ignored (calc works with tiny numbers only).
221
222     jmp exit
223
224
225
226
227
228 do_div:
229     ; dx is ignored (calc works with tiny integer numbers only).
230     mov dx, 0
231     mov ax, num1
232

```

line: 554 col: 1

```

edit C:\emu8086\examples\Calculator.asm
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator converter options help about
233 idiv num2 ; ax = (dx ax) / num2.
234 cmp dx, 0
235 jnz approx
236 call print_num ; print ax value.
237 jmp exit
238 approx:
239 call print_num ; print ax value.
240 lea dx, smth
241 mov ah, 09h ; output string at ds:dx
242 int 21h
243 jmp exit
244
245
246
247
248
249
250
251
252
253 ;;; these functions are copied from emu8086.inc ;;;;
254
255
256
257
258 ; gets the multi-digit SIGNED number from the keyboard,
259 ; and stores the result in CX register:
260 SCAN_NUM PROC NEAR
261 PUSH DX
262 PUSH AX
263 PUSH SI
264 MOV CX, 0
265 ; reset flag:
266 MOV CS:make_minus, 0
270 next_digit:
271
272 ; get char from keyboard
273 ; into AL:
274 MOV AH, 00h
275 INT 16h
276 ; and print it:
277 MOV AH, 0Eh
278 INT 10h
279 ; check for MINUS:
280 CMP AL, '-'
281 JE set_minus
282
283 ; check for ENTER key:
284 CMP AL, 0Dh ; carriage return?
285 JNE not_cr
286 JMP stop_input
287
288 not_cr:
289
290 CMP AL, 8 ; 'BACKSPACE' pressed?
291 JNE backspace_checked
292 MOV DX, 0 ; remove last digit by
293 MOV AX, CX ; division:
294 DIV CS:ten ; AX = DX:AX / 10 (DX-rem).
295 MOV CX, AX ; clear position.
296 PUTC Y, AX ; backspace again.
297 PUTC 8
298 JMP next_digit
299
300 backspace_checked:
301
302 ; allow only digits:
303 CMP AL, '0'
304 JAE ok_AE_0
305 JMP remove_not_digit
306
307 ok_AE_0:
308 CMP AL, '9'
309 JBE ok_digit

```

edit C:\emu8086\examples\Calculator.asm

file edit bookmarks assembler emulator math ascii codes help

new open examples save compile emulate calculator converter options help about

```

310 remove_not_digit:
311     PUTC 8, ; backspace.
312     PUTC 8, ; clear last entered not digit.
313     PUTC 8, ; backspace again.
314     JMP next_digit; wait for next input.
315 ok_digit:
316
317     ; multiply CX by 10 (first time the result is zero)
318     PUSH AX
319     MOV AX, CX
320     MUL CS:[ten]
321     MOV CX, AX ; DX:AX = AX*10
322     POP AX
323
324     ; check if the number is too big
325     ; (result should be 16 bits)
326     CMP DX, 0
327     JNE too_big
328
329     ; convert from ASCII code:
330     SUB AL, 30h
331
332     ; add AL to CX:
333     MOV AH, 0
334     MOV DX, CX ; backup, in case the result will be too big.
335     ADD CX, AX
336     JC too_big2 ; jump if the number is too big.
337
338     JMP next_digit
339
340 set_minus:
341     MOV CS:[make_minus], 1
342     JMP next_digit
343
344 too_big2:
345     MOV CX, DX ; restore the backedup value before add.
346     MOV DX, 0 ; DX was zero before backup!

```

line: 554 col: 1 drag a file here to open

Buscar 31°C Soleado 03:43 p. m. 23/09/2023

edit C:\emu8086\examples\Calculator.asm

file edit bookmarks assembler emulator math ascii codes help

new open examples save compile emulate calculator converter options help about

```

348 too_big:
349     MOV AX, CX
350     DIV CS:[ten] ; reverse last DX:AX = AX*10, make AX = DX:AX / 10
351     MOV CX, AX
352     PUTC 8, ; backspace.
353     PUTC 8, ; clear last entered digit.
354     PUTC 8, ; backspace again.
355     JMP next_digit ; wait for Enter/Backspace.
356
357 stop_input:
358     ; check flag:
359     CMP CS:[make_minus], 0
360     NEQ not_minus
361     NEG CX
362 not_minus:
363     POP SI
364     POP AX
365     POP DX
366     RET
367 make_minus DB ?
368 SCAN_NUM ENDP
369
370
371
372
373
374
375
376 ; this procedure prints number in AX,
377 ; used with PRINT_NUM_UNS to print signed numbers:
378 PRINT_NUM PROC NEAR
379     PUSH DX
380     PUSH AX
381
382     CMP AX, 0
383     JNZ not_zero
384
385     PUTC '0'

```

line: 554 col: 1 drag a file here to open

Buscar 31°C Soleado 03:43 p. m. 23/09/2023

edit C:\emu8086\examples\Calculator.asm

file edit bookmarks assembler emulator math ascii codes help

new open examples save compile emulate calculator converter options help about

```

386    JMP     printed
387
388 not_zero:   ; the check SIGN of AX,
389             ; make absolute if it's negative:
390     CMP    AX, 0
391     JNS    positive
392     NEG    AX
393
394     PUTC   '-'
395
396 positive:   CALL   PRINT_NUM_UNS
397
398 printed:    POP    AX
399             POP    DX
400             RET
401
402 PRINT_NUM  ENDP
403
404
405
406 ; this procedure prints out an unsigned
407 ; number in AX (not just a single digit)
408 ; allowed values are from 0 to 65535 (FFFF)
409
410 PRINT_NUM_UNS PROC NEAR
411     PUSH   AX
412     PUSH   BX
413     PUSH   CX
414     PUSH   DX
415
416     ; flag to prevent printing zeros before number:
417     MOV    CX, 1
418
419     ; (result of "/ 10000" is always less or equal to 9).
420     MOV    BX, 10000 ; 2710h - divider.
421
422     ; AX is zero?
423     CMP    AX, 0

```

line: 554 col: 1 drag a file here to open

31°C Soleado 03:43 p.m. 23/09/2023

edit C:\emu8086\examples\Calculator.asm

file edit bookmarks assembler emulator math ascii codes help

new open examples save compile emulate calculator converter options help about

```

424     JZ     print_zero
425
426 begin_print:
427     ; check divider (if zero go to end_print):
428     CMP    BX, 0
429     JZ     end_print
430
431     ; avoid printing zeros before number:
432     CMP    CX, 0
433     JE    calc
434     ; if AX<BX then result of DIV will be zero:
435     CMP    AX, BX
436     JB    skip
437
438 calc:   MOV    CX, 0 ; set flag.
439
440     MOV    DX, 0 ; AX = DX:AX / BX (DX=remainder).
441
442     ; print last digit
443     ; AH is always ZERO so it's ignored
444     ADD    AL, 30h ; convert to ASCII code.
445     PUTC   AL
446
447
448     MOV    AX, DX ; get remainder from last div.
449
450 skip:   ; calculate BX=BX/10
451     PUSH   AX
452     MOV    DX, 0
453     MOV    AX, BX
454     DIV    CS:[ten] ; AX = DX:AX / 10 (DX=remainder).
455     MOV    BX, AX
456     POP    AX
457
458     JMP    begin_print
459
460
461

```

line: 554 col: 1 drag a file here to open

31°C Soleado 03:44 p.m. 23/09/2023

edit C:\emu8086\examples\Calculator.asm

file edit bookmarks assembler emulator math ascii codes help

new open examples save compile emulate calculator converter options help about

```

463 print_zero:    PUTC    '0'
464
465 end_print:
466
467     POP    DX
468     POP    CX
469     POP    BX
470     POP    AX
471     RET
472
473 PRINT_NUM_UNS    ENDP
474
475
476 ten      DW      10      ; used as multiplier/divider by SCAN_NUM & PRINT_NUM_UNS.
477
478
479
480
481
482
483
484 GET_STRING    PROC    NEAR
485
486 PUSH   AX
487 PUSH   CX
488 PUSH   DI
489 PUSH   DX
490
491 MOV    CX, 0          ; char counter.
492
493 CMP    DX, 1          ; buffer too small?
494 JBE    empty_buffer
495
496 DEC    DX
497
498 ; =====
499 ; Eternal loop to get
500 ; and processes key presses:
501
502 ; =====
503
504 wait_for_key:
505
506     MOV    AH, 0          ; get pressed key.
507     INT   16h
508
509     CMP    AL, 0Dh        ; 'RETURN' pressed?
510     JZ    exit_GET_STRING
511
512     CMP    AL, 8          ; 'BACKSPACE' pressed?
513     JNE    add_to_buffer
514     JCXZ  wait_for_key
515     DEC    DX
516     DEC    DI
517     PUTC  0
518     PUTC  0
519     PUTC  8
520     JMP    wait_for_key
521
522 add_to_buffer:
523
524     CMP    CX, DX        ; buffer is full?
525     JAE    wait_for_key
526     ; if so wait for 'BACKSPACE' or 'RETURN'...
527     MOV    [DI], AL
528     INC    DI
529     INC    CX
530
531     ; print the key:
532     MOV    AH, 0Eh
533     INT   10h
534
535 JMP    wait_for_key
536 ; =====
537

```

line: 554 col: 1 drag a file here to open

Buscar 31°C Soleado 03:44 p.m. 23/09/2023

edit C:\emu8086\examples\Calculator.asm

file edit bookmarks assembler emulator math ascii codes help

new open examples save compile emulate calculator converter options help about

line: 532 col: 86 drag a file here to open

Buscar 31°C Soleado 03:44 p.m. 23/09/2023

The screenshot shows the emu8086 IDE interface with the assembly code for a calculator example. The code includes comments explaining the logic for handling input and clearing the buffer. The assembly instructions are color-coded by category.

```
edit: C:\emu8086\examples\Calculator.asm
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator converter options help about
518 PUTC    ','          ; clear position.
519 PUTC    8             ; backspace again.
520 JMP     wait_for_key
521
522 add_to_buffer:
523
524 CMP     CX, DX      ; buffer is full?
525 JAE     wait_for_key ; if so wait for 'BACKSPACE' or 'RETURN'...
526 MOV     [DI], AL
527 INC     DI
528 INC     CX
529
530 ; print the key:
531 MOV     AH, 0EH
532 INT     10h
533
534 JMP     wait_for_key
535 =====
536 exit_GET_STRING:
537 ; terminate by null:
538 MOV     [DI], 0
539
540 empty_buffer:
541 POP    DX
542 POP    DI
543 POP    CX
544 POP    AX
545 RET
550 GET_STRING ENDP
```

The screenshot shows the emu8086 IDE interface with two windows. The top window displays the assembly code for 'calc2.com'. The bottom window is a debugger window showing the state of registers and memory at address 0700:0100. The assembly code in the debugger window corresponds to the code in the top window.

```
edit: C:\emu8086\examples\calc2.com
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator converter options
001 name "calc2"
002
003 emulator: calc2.com_
004 file math debug view external virtual devices virtual drive help
005 Load reload step back single step run step delay ms: 0
006
007 registers H L
008 AX 00 00
009 BX 00 00
010 CX 03 08
011 DX 00 00
012 CS 0700
013 IP 0100
014 SS 0700
015 SP FFFE
016 BP 0000
017 SI 0000
018 DI 0000
019 DS 0700
020 ES 0700
021
022 0700:0100 0700:0100
023 07100: E9 233 u JMP 00274h
024 07101: E1 113 d OUTSB
025 07102: 01 001 @ OUTSW
026 07103: 6E 110 n JZ 016Ch
027 07104: 6F 111 o CMP AH, [BX + SI]
028 07105: 64 116 t DB 63h
029 07106: 65 101 e POPA
030 07107: 3A 058 : INSB
031 07108: 20 032 $I DB 63h
032 07109: 63 099 c JNE 017Bh
033 0710A: 61 097 a POPA
034 0710B: 6C 108 l JZ 0181h
035 0710C: 63 099 c JB 0134h
036 0710D: 75 117 u JNBE 0185h
037 0710E: 6C 108 l JB 0183h
038 0710F: 61 097 a ...
039
040
041
042
043
044
045 org 100h
046 jmp start
047
048
049
050
051
052
053
054
055
056
057
058
059
060
061
062
063
064
065
066
067
068
069
070
071
072
073
074
075
076
077
078
079
080
081
082
083
084
085
086
087
088
089
090
091
092
093
094
095
096
097
098
099
0100
0101
0102
0103
0104
0105
0106
0107
0108
0109
0110
0111
0112
0113
0114
0115
0116
0117
0118
0119
0120
0121
0122
0123
0124
0125
0126
0127
0128
0129
0130
0131
0132
0133
0134
0135
0136
0137
0138
0139
0140
0141
0142
0143
0144
0145
0146
0147
0148
0149
0150
0151
0152
0153
0154
0155
0156
0157
0158
0159
0160
0161
0162
0163
0164
0165
0166
0167
0168
0169
0170
0171
0172
0173
0174
0175
0176
0177
0178
0179
0180
0181
0182
0183
0184
0185
0186
0187
0188
0189
0190
0191
0192
0193
0194
0195
0196
0197
0198
0199
0200
0201
0202
0203
0204
0205
0206
0207
0208
0209
0210
0211
0212
0213
0214
0215
0216
0217
0218
0219
0220
0221
0222
0223
0224
0225
0226
0227
0228
0229
0230
0231
0232
0233
0234
0235
0236
0237
0238
0239
0240
0241
0242
0243
0244
0245
0246
0247
0248
0249
0250
0251
0252
0253
0254
0255
0256
0257
0258
0259
0260
0261
0262
0263
0264
0265
0266
0267
0268
0269
0270
0271
0272
0273
0274
0275
0276
0277
0278
0279
0280
0281
0282
0283
0284
0285
0286
0287
0288
0289
0290
0291
0292
0293
0294
0295
0296
0297
0298
0299
0300
0301
0302
0303
0304
0305
0306
0307
0308
0309
0310
0311
0312
0313
0314
0315
0316
0317
0318
0319
0320
0321
0322
0323
0324
0325
0326
0327
0328
0329
0330
0331
0332
0333
0334
0335
0336
0337
0338
0339
0340
0341
0342
0343
0344
0345
0346
0347
0348
0349
0350
0351
0352
0353
0354
0355
0356
0357
0358
0359
0360
0361
0362
0363
0364
0365
0366
0367
0368
0369
0370
0371
0372
0373
0374
0375
0376
0377
0378
0379
0380
0381
0382
0383
0384
0385
0386
0387
0388
0389
0390
0391
0392
0393
0394
0395
0396
0397
0398
0399
0400
0401
0402
0403
0404
0405
0406
0407
0408
0409
0410
0411
0412
0413
0414
0415
0416
0417
0418
0419
0420
0421
0422
0423
0424
0425
0426
0427
0428
0429
0430
0431
0432
0433
0434
0435
0436
0437
0438
0439
0440
0441
0442
0443
0444
0445
0446
0447
0448
0449
0450
0451
0452
0453
0454
0455
0456
0457
0458
0459
0460
0461
0462
0463
0464
0465
0466
0467
0468
0469
0470
0471
0472
0473
0474
0475
0476
0477
0478
0479
0480
0481
0482
0483
0484
0485
0486
0487
0488
0489
0490
0491
0492
0493
0494
0495
0496
0497
0498
0499
0500
0501
0502
0503
0504
0505
0506
0507
0508
0509
0510
0511
0512
0513
0514
0515
0516
0517
0518
0519
0520
0521
0522
0523
0524
0525
0526
0527
0528
0529
0530
0531
0532
0533
0534
0535
0536
0537
0538
0539
0540
0541
0542
0543
0544
0545
0546
0547
0548
0549
0550
0551
0552
0553
0554
0555
0556
0557
0558
0559
0560
0561
0562
0563
0564
0565
0566
0567
0568
0569
0570
0571
0572
0573
0574
0575
0576
0577
0578
0579
0580
0581
0582
0583
0584
0585
0586
0587
0588
0589
058A
058B
058C
058D
058E
058F
058G
058H
058I
058J
058K
058L
058M
058N
058O
058P
058Q
058R
058S
058T
058U
058V
058W
058X
058Y
058Z
058a
058b
058c
058d
058e
058f
058g
058h
058i
058j
058k
058l
058m
058n
058o
058p
058q
058r
058s
058t
058u
058v
058w
058x
058y
058z
058A
058B
058C
058D
058E
058F
058G
058H
058I
058J
058K
058L
058M
058N
058O
058P
058Q
058R
058S
058T
058U
058V
058W
058X
058Y
058Z
058a
058b
058c
058d
058e
058f
058g
058h
058i
058j
058k
058l
058m
058n
058o
058p
058q
058r
058s
058t
058u
058v
058w
058x
058y
058z
058A
058B
058C
058D
058E
058F
058G
058H
058I
058J
058K
058L
058M
058N
058O
058P
058Q
058R
058S
058T
058U
058V
058W
058X
058Y
058Z
058a
058b
058c
058d
058e
058f
058g
058h
058i
058j
058k
058l
058m
058n
058o
058p
058q
058r
058s
058t
058u
058v
058w
058x
058y
058z
058A
058B
058C
058D
058E
058F
058G
058H
058I
058J
058K
058L
058M
058N
058O
058P
058Q
058R
058S
058T
058U
058V
058W
058X
058Y
058Z
058a
058b
058c
058d
058e
058f
058g
058h
058i
058j
058k
058l
058m
058n
058o
058p
058q
058r
058s
058t
058u
058v
058w
058x
058y
058z
058A
058B
058C
058D
058E
058F
058G
058H
058I
058J
058K
058L
058M
058N
058O
058P
058Q
058R
058S
058T
058U
058V
058W
058X
058Y
058Z
058a
058b
058c
058d
058e
058f
058g
058h
058i
058j
058k
058l
058m
058n
058o
058p
058q
058r
058s
058t
058u
058v
058w
058x
058y
058z
058A
058B
058C
058D
058E
058F
058G
058H
058I
058J
058K
058L
058M
058N
058O
058P
058Q
058R
058S
058T
058U
058V
058W
058X
058Y
058Z
058a
058b
058c
058d
058e
058f
058g
058h
058i
058j
058k
058l
058m
058n
058o
058p
058q
058r
058s
058t
058u
058v
058w
058x
058y
058z
058A
058B
058C
058D
058E
058F
058G
058H
058I
058J
058K
058L
058M
058N
058O
058P
058Q
058R
058S
058T
058U
058V
058W
058X
058Y
058Z
058a
058b
058c
058d
058e
058f
058g
058h
058i
058j
058k
058l
058m
058n
058o
058p
058q
058r
058s
058t
058u
058v
058w
058x
058y
058z
058A
058B
058C
058D
058E
058F
058G
058H
058I
058J
058K
058L
058M
058N
058O
058P
058Q
058R
058S
058T
058U
058V
058W
058X
058Y
058Z
058a
058b
058c
058d
058e
058f
058g
058h
058i
058j
058k
058l
058m
058n
058o
058p
058q
058r
058s
058t
058u
058v
058w
058x
058y
058z
058A
058B
058C
058D
058E
058F
058G
058H
058I
058J
058K
058L
058M
058N
058O
058P
058Q
058R
058S
058T
058U
058V
058W
058X
058Y
058Z
058a
058b
058c
058d
058e
058f
058g
058h
058i
058j
058k
058l
058m
058n
058o
058p
058q
058r
058s
058t
058u
058v
058w
058x
058y
058z
058A
058B
058C
058D
058E
058F
058G
058H
058I
058J
058K
058L
058M
058N
058O
058P
058Q
058R
058S
058T
058U
058V
058W
058X
058Y
058Z
058a
058b
058c
058d
058e
058f
058g
058h
058i
058j
058k
058l
058m
058n
058o
058p
058q
058r
058s
058t
058u
058v
058w
058x
058y
058z
058A
058B
058C
058D
058E
058F
058G
058H
058I
058J
058K
058L
058M
058N
058O
058P
058Q
058R
058S
058T
058U
058V
058W
058X
058Y
058Z
058a
058b
058c
058d
058e
058f
058g
058h
058i
058j
058k
058l
058m
058n
058o
058p
058q
058r
058s
058t
058u
058v
058w
058x
058y
058z
058A
058B
058C
058D
058E
058F
058G
058H
058I
058J
058K
058L
058M
058N
058O
058P
058Q
058R
058S
058T
058U
058V
058W
058X
058Y
058Z
058a
058b
058c
058d
058e
058f
058g
058h
058i
058j
058k
058l
058m
058n
058o
058p
058q
058r
058s
058t
058u
058v
058w
058x
058y
058z
058A
058B
058C
058D
058E
058F
058G
058H
058I
058J
058K
058L
058M
058N
058O
058P
058Q
058R
058S
058T
058U
058V
058W
058X
058Y
058Z
058a
058b
058c
058d
058e
058f
058g
058h
058i
058j
058k
058l
058m
058n
058o
058p
058q
058r
058s
058t
058u
058v
058w
058x
058y
058z
058A
058B
058C
058D
058E
058F
058G
058H
058I
058J
058K
058L
058M
058N
058O
058P
058Q
058R
058S
058T
058U
058V
058W
058X
058Y
058Z
058a
058b
058c
058d
058e
058f
058g
058h
058i
058j
058k
058l
058m
058n
058o
058p
058q
058r
058s
058t
058u
058v
058w
058x
058y
058z
058A
058B
058C
058D
058E
058F
058G
058H
058I
058J
058K
058L
058M
058N
058O
058P
058Q
058R
058S
058T
058U
058V
058W
058X
058Y
058Z
058a
058b
058c
058d
058e
058f
058g
058h
058i
058j
058k
058l
058m
058n
058o
058p
058q
058r
058s
058t
058u
058v
058w
058x
058y
058z
058A
058B
058C
058D
058E
058F
058G
058H
058I
058J
058K
058L
058M
058N
058O
058P
058Q
058R
058S
058T
058U
058V
058W
058X
058Y
058Z
058a
058b
058c
058d
058e
058f
058g
058h
058i
058j
058k
058l
058m
058n
058o
058p
058q
058r
058s
058t
058u
058v
058w
058x
058y
058z
058A
058B
058C
058D
058E
058F
058G
058H
058I
058J
058K
058L
058M
058N
058O
058P
058Q
058R
058S
058T
058U
058V
058W
058X
058Y
058Z
058a
058b
058c
058d
058e
058f
058g
058h
058i
058j
058k
058l
058m
058n
058o
058p
058q
058r
058s
058t
058u
058v
058w
058x
058y
058z
058A
058B
058C
058D
058E
058F
058G
058H
058I
058J
058K
058L
058M
058N
058O
058P
058Q
058R
058S
058T
058U
058V
058W
058X
058Y
058Z
058a
058b
058c
058d
058e
058f
058g
058h
058i
058j
058k
058l
058m
058n
058o
058p
058q
058r
058s
058t
058u
058v
058w
058x
058y
058z
058A
058B
058C
058D
058E
058F
058G
058H
058I
058J
058K
058L
058M
058N
058O
058P
058Q
058R
058S
058T
058U
058V
058W
058X
058Y
058Z
058a
058b
058c
058d
058e
058f
058g
058h
058i
058j
058k
058l
058m
058n
058o
058p
058q
058r
058s
058t
058u
058v
058w
058x
058y
058z
058A
058B
058C
058D
058E
058F
058G
058H
058I
058J
058K
058L
058M
058N
058O
058P
058Q
058R
058S
058T
058U
058V
058W
058X
058Y
058Z
058a
058b
058c
058d
058e
058f
058g
058h
058i
058j
058k
058l
058m
058n
058o
058p
058q
058r
058s
058t
058u
058v
058w
058x
058y
058z
058A
058B
058C
058D
058E
058F
058G
058H
058I
058J
058K
058L
058M
058N
058O
058P
058Q
058R
058S
058T
058U
058V
058W
058X
058Y
058Z
058a
058b
058c
058d
058e
058f
058g
058h
058i
058j
058k
058l
058m
058n
058o
058p
058q
058r
058s
058t
058u
058v
058w
058x
058y
058z
058A
058B
058C
058D
058E
058F
058G
058H
058I
058J
058K
058L
058M
058N
058O
058P
058Q
058R
058S
058T
058U
058V
058W
058X
058Y
058Z
058a
058b
058c
058d
058e
058f
058g
058h
058i
058j
058k
058l
058m
058n
058o
058p
058q
058r
058s
058t
058u
058v
058w
058x
058y
058z
058A
058B
058C
058D
058E
058F
058G
058H
058I
058J
058K
058L
058M
058N
058O
058P
058Q
058R
058S
058T
058U
058V
058W
058X
058Y
058Z
058a
058b
058c
058d
058e
058f
058g
058h
058i
058j
058k
058l
058m
058n
058o
058p
058q
058r
058s
058t
058u
058v
058w
058x
058y
058z
058A
058B
058C
058D
058E
058F
058G
058H
058I
058J
058K
058L
058M
058N
058O
058P
058Q
058R
058S
058T
058U
058V
058W
058X
058Y
058Z
058a
058b
058c
058d
058e
058f
058g
058h
058i
058j
058k
058l
058m
058n
058o
058p
058q
058r
058s
058t
058u
058v
058w
058x
058y
058z
058A
058B
058C
058D
058E
058F
058G
058H
058I
058J
058K
058L
058M
058N
058O
058P
058Q
058R
058S
058T
058U
058V
058W
058X
058Y
058Z
058a
058b
058c
058d
058e
058f
058g
058h
058i
058j
058k
058l
058m
058n
058o
058p
058q
058r
058s
058t
058u
058v
058w
058x
058y
058z
058A
058B
058C
058D
058E
058F
058G
058H
058I
058J
058K
058L
058M
058N
058O
058P
058Q
058R
058S
058T
058U
058V
058W
058X
058Y
058Z
058a
058b
058c
058d
058e
058f
058g
058h
058i
058j
058k
058l
058m
058n
058o
058p
058q
058r
058s
058t
058u
058v
058w
058x
058y
058z
058A
058B
058C
058D
058E
058F
058G
058H
058I
058J
058K
058L
058M
058N
058O
058P
058Q
058R
058S
058T
058U
058V
058W
058X
058Y
058Z
058a
058b
058c
058d
058e
058f
058g
058h
058i
058j
058k
058l
058m
058n
058o
058p
058q
058r
058s
058t
058u
058v
058w
058x
058y
058z
058A
058B
058C
058D
058E
058F
058G
058H
058I
058J
058K
058L
058M
058N
058O
058P
058Q
058R
058S
058T
058U
058V
058W
058X
058Y
058Z
058a
058b
058c
058d
058e
058f
058g
058h
058i
058j
058k
058l
058m
058n
058o
058p
058q
058r
058s
058t
058u
058v
058w
058x
058y
058z
058A
058B
058C
058D
058E
058F
058G
058H
058I
058J
058K
058L
058M
058N
058O
058P
058Q
058R
058S
058T
058U
058V
058W
058X
058Y
058Z
058a
058b
058c
058d
058e
058f
058g
058h
058i
058j
058k
058l
058m
058n
058o
058p
058q
058r
058s
058t
058u
058v
058w
058x
058y
058
```

El código propuesto es un programa en lenguaje ensamblador para la arquitectura x86 que utiliza Emu8086 como entorno de desarrollo. El programa es un simple calculador de línea de comandos que realiza operaciones aritméticas básicas (+, -, *, /) con números enteros. A continuación, describiré el tipo de direccionamiento, los registros y la estructura del código:

Tipo de Direccionamiento: El código utiliza varios tipos de direccionamiento, incluyendo:

Direccionamiento Inmediato: Se utiliza para cargar valores directamente en registros, por ejemplo, mov ax, num1.

Direccionamiento por Registro: Se emplea para mover datos entre registros, como mov ax, num1.

Direccionamiento por Memoria: Se usa para acceder a datos en la memoria, por ejemplo, mov dx, offset msg0 para cargar una dirección de memoria.

Direccionamiento por Etiqueta: Se utiliza para definir y referenciar etiquetas que marcan puntos en el programa, como start: y exit:.

Registros: El código hace uso de varios registros de la arquitectura x86, como:

AX: Se usa para almacenar valores temporales y resultados de cálculos.

BX, CX, DX: Estos registros se utilizan para diversas operaciones de datos y cálculos.

SI: Se usa en combinación con direccionamiento por memoria en algunas partes del código para acceder a datos en la memoria.

DI: Se utiliza para apuntar a ubicaciones de memoria durante la entrada de texto.

Estructura del Código: El código se estructura de la siguiente manera:

Comienza con comentarios y una sección de mensajes que se imprimirán en pantalla.

Luego, define variables y etiquetas.

La ejecución comienza en la etiqueta start:, que muestra mensajes, recopila entrada del usuario y realiza cálculos.

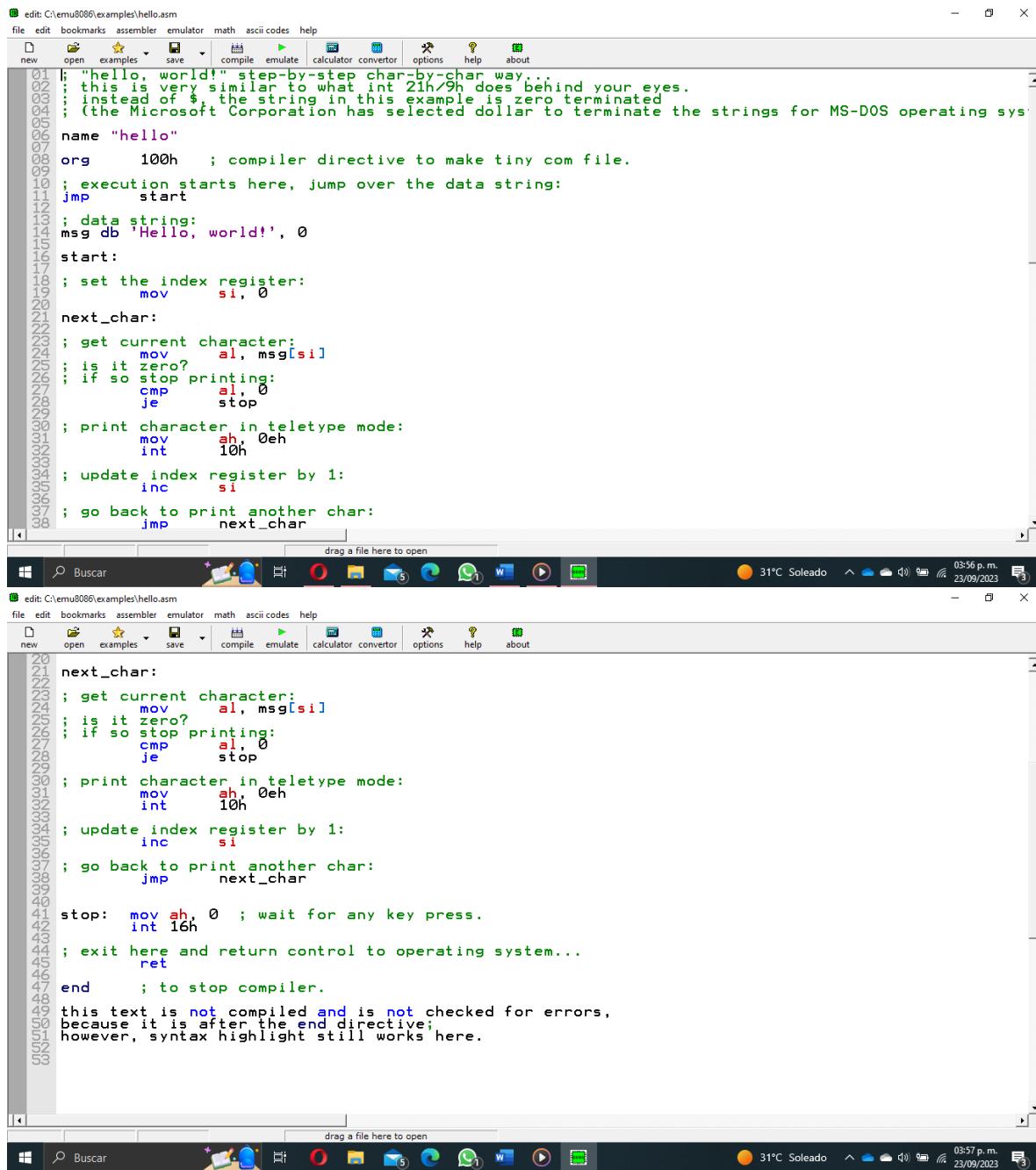
Utiliza subrutinas para realizar diversas operaciones, como scan_num para leer números desde la entrada del usuario y print_num para imprimir resultados.

El código verifica el operador ingresado por el usuario y realiza la operación adecuada.

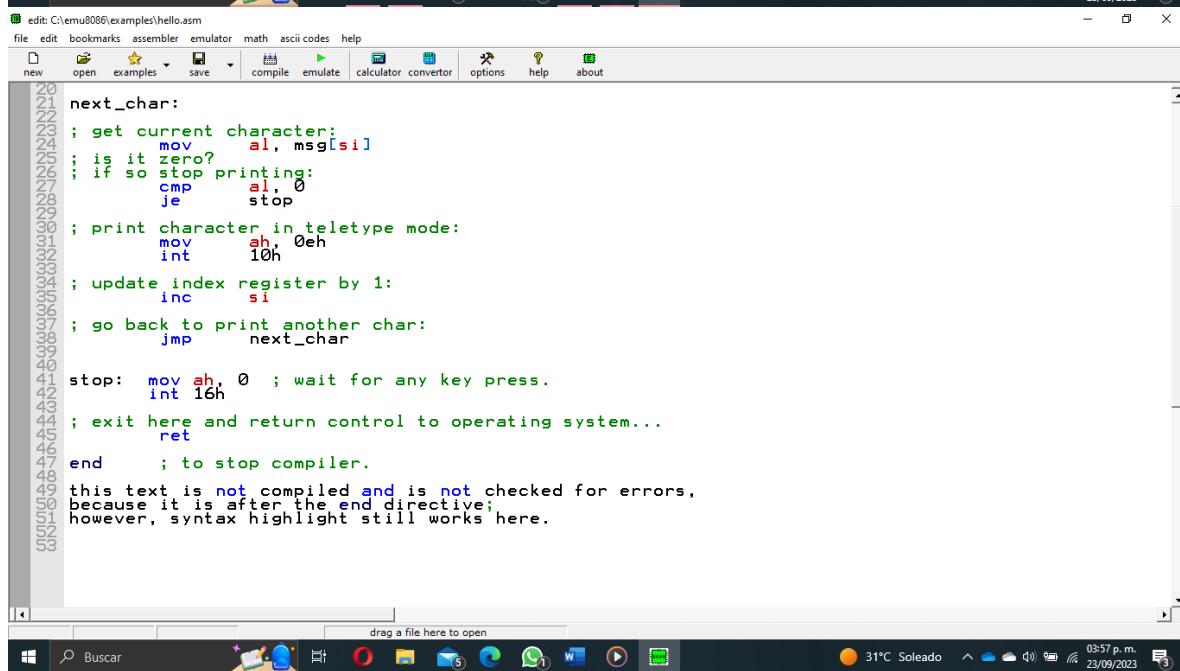
Finalmente, muestra el resultado y espera a que el usuario presione una tecla antes de salir.

En resumen, el código utiliza múltiples tipos de direccionamiento, registra datos en varios registros y sigue una estructura típica de programas en lenguaje ensamblador para realizar cálculos simples en Emu8086.

2. Hello World



```
01; "Hello, world!" step-by-step char-by-char way.
02; this is very similar to what int 21h/9h does behind your eyes.
03; instead of $, the string in this example is zero terminated.
04; (the Microsoft Corporation has selected dollar to terminate the strings for MS-DOS operating sys-
05; name "hello"
06
07 org 100h ; compiler directive to make tiny com file.
08
09 ; execution starts here, jump over the data string:
10 jmp start
11
12 ; data string:
13 msg db 'Hello, world!', 0
14
15 start:
16
17 ; set the index register:
18 mov si, 0
19
20 next_char:
21
22 ; get current character:
23 mov al, msg[si]
24 ; is it zero?
25 ; if so stop printing:
26 cmp al, 0
27 je stop
28
29 ; print character in teletype mode:
30 mov ah, 0eh
31 int 10h
32
33 ; update index register by 1:
34 inc si
35
36 ; go back to print another char:
37 jmp next_char
38
```



```
20
21 next_char:
22
23 ; get current character:
24 mov al, msg[si]
25 ; is it zero?
26 ; if so stop printing:
27 cmp al, 0
28 je stop
29
30 ; print character in teletype mode:
31 mov ah, 0eh
32 int 10h
33
34 ; update index register by 1:
35 inc si
36
37 ; go back to print another char:
38 jmp next_char
39
40 stop: mov ah, 0 ; wait for any key press.
41 int 16h
42
43 ; exit here and return control to operating system...
44 ret
45
46 end ; to stop compiler.
47
48 this text is not compiled and is not checked for errors,
49 because it is after the end directive;
50 however, syntax highlight still works here.
51
52
53
```

El código proporcionado es un programa simple en lenguaje ensamblador para la arquitectura x86 que imprime el mensaje "Hello, ¡world!" en la pantalla de la forma tradicional, carácter por carácter. Aquí está la descripción de los aspectos relevantes:

Tipo de Direccionamiento:

Direccionamiento por Registro: Se utiliza principalmente el registro SI (Source Index) para realizar el direccionamiento por índice a través del mensaje msg. Esto permite acceder a los caracteres del mensaje de forma secuencial, ya que SI se incrementa en cada iteración para apuntar al siguiente carácter.

Registros:

SI (Source Index): Se utiliza como un puntero para acceder a los caracteres del mensaje msg. Su valor se incrementa en cada iteración del bucle para pasar al siguiente carácter del mensaje.

AL (Accumulator Low): Se utiliza para almacenar el carácter actual que se imprimirá en pantalla.

AH (Accumulator High): Se utiliza para especificar la función de impresión en teletipo (int 10h, ah=0eh).

Estructura del Código:

El código comienza con comentarios que explican el propósito y funcionamiento del programa.

Luego, define una cadena de datos (msg) que contiene el mensaje "Hello, ¡world!" seguido de un byte nulo (0) para indicar el final de la cadena. El mensaje está terminado con un byte nulo para que la rutina de impresión sepa cuándo detenerse.

La ejecución comienza en la etiqueta start: y salta sobre la cadena de datos.

Dentro del bucle next_char:, el programa obtiene el carácter actual de la cadena, lo imprime en pantalla y luego incrementa el índice SI para apuntar al siguiente carácter.

El bucle continúa hasta que se encuentra un byte nulo en la cadena (`cmp al, 0`), momento en el que se detiene la impresión.

Después de imprimir el mensaje, el programa espera a que el usuario presione cualquier tecla antes de salir.

Finalmente, el programa utiliza `ret` para regresar al sistema operativo.

En resumen, este código utiliza direccionamiento por registro e índice para recorrer una cadena de caracteres, registros para almacenar datos temporales y sigue una estructura típica de programas en lenguaje ensamblador que realizan operaciones de entrada/salida en Emu8086. ¡El código se encarga de imprimir "Hello, world!" carácter por carácter en la pantalla.

CONCLUSIONES

Basado en los códigos anteriores y su implementación en Emu8086, se pueden extraer las siguientes conclusiones sobre Emu8086:

Entorno de Desarrollo Amigable: Proporciona un entorno de desarrollo amigable y fácil de usar para programación en lenguaje ensamblador orientado a la arquitectura x86. Facilita la escritura, prueba y depuración de código ensamblador a través de su interfaz gráfica.

Soporte para Instrucciones x86: Es un emulador de la arquitectura x86 que admite un conjunto completo de instrucciones x86, lo que permite a los programadores escribir programas de bajo nivel que se ejecuten en esta arquitectura.

Facilita la Enseñanza y el Aprendizaje: Dado que Emu8086 permite a los programadores ver el efecto de cada instrucción paso a paso, es una herramienta útil para la enseñanza y el aprendizaje de la programación en lenguaje ensamblador y la comprensión de cómo funcionan las computadoras a nivel de bajo nivel.

Depuración y Pruebas Efectivas: Proporciona características de depuración, como rastreo de código y visualización de registros, que facilitan la identificación y corrección de errores en el código ensamblador.

Interfaz Gráfica de Usuario (GUI): Ofrece una GUI que permite una experiencia de desarrollo más visual y amigable en comparación con algunas otras herramientas de desarrollo de lenguaje ensamblador que se ejecutan en línea de comandos.

Soporte para Interrupciones de DOS: Permite la ejecución de interrupciones de DOS (por ejemplo, int 21h) para realizar operaciones de entrada/salida y funciones del sistema operativo MS-DOS, lo que es útil para crear aplicaciones DOS retrocompatibles.

Ampliamente Utilizado para Propósitos Educativos: Es comúnmente utilizado en entornos educativos para enseñar conceptos de programación de bajo nivel, arquitectura de computadoras y lenguaje ensamblador.

Compatibilidad con Programas Antiguos: Puede ser útil para ejecutar y depurar programas antiguos escritos en lenguaje ensamblador para DOS.

En general, Emu8086 es una herramienta valiosa para aquellos que desean aprender y experimentar con la programación en lenguaje ensamblador en la arquitectura x86, así como para aquellos que necesitan trabajar en entornos retrocompatibles con MS-DOS. Facilita la comprensión de la programación de bajo nivel y la arquitectura del procesador.

Bibliografía

2001, I. M. (23 de 09 de 2023). *Arquitectura 8086*. Obtenido de <https://users.exa.unicen.edu.ar/catedras/progens/materiales/arquitectura%208086.pdf>

Sevilla, D. d. (23 de 09 de 2023). *cs.buap.mx*. Obtenido de Microsoft Word - Esamblador 8086: https://www.cs.buap.mx/~mgonzalez/asm_mododir2.pdf

Edit:C:\emu8086\examples\hello.asm

Edit:C:\emu8086\examples\Calculator.asm

RÚBRICA DE EVALUACIÓN – CALIFICACIÓN

Estructura del reporte	Cumple perfectamente con la estructura solicitada <i>8 puntos</i>	Cumple con 5 puntos de la estructura del reporte <i>5 puntos</i>	Cumple con menos de 4 puntos de la estructura del reporte <i>3 puntos</i>	No cumple con la estructura solicitada <i>0 puntos</i>
Claridad en la descripción de la interfaz	La secuencia de pasos, la claridad y el orden de pasos en la descripción de la interfaz es perfectamente clara <i>10 puntos</i>	La secuencia de pasos es sistemática aunque se pudo haber mejorado <i>7 puntos</i>	La secuencia de pasos es poco clara <i>4 puntos</i>	No se comprende la secuencia de pasos descrita <i>0 puntos</i>
Contenido teórico	El contenido teórico es robusto y esta fundamentado en referencias bibliográficas de gran reconocimiento en el tema <i>6 puntos</i>	El contenido teórico es básico y bien fundamentado en referencias bibliográficas válidas <i>4 puntos</i>	El contenido es claro pero se pudo haber mejorado <i>3 puntos</i>	El contenido es muy concreto y no refleja claridad ni congruencia con el tema <i>0 puntos</i>

Congruencia de comprensión de emulador	El contenido refleja una congruencia aceptable para llegar a comprender como funciona el emulador 5 puntos	El contenido refleja poca congruencia para comprender el tema del emulador 3 puntos	No se percibe congruencia en la comprensión del emulador 0 puntos	
Conclusiones	Las conclusiones reflejan el grado de reflexión del tema, esta bien expresado y valida la asimilacion del conocimiento 6 puntos	Las conclusiones parecen ser un resumen y no una reflexión del tema 3 puntos	No coloco conclusiones 0 puntos	
Formato APA	Aplicó al 100% el formato APA en el documento 5 puntos	Aplicó el formato APA en un 80 % del documento 3 puntos	Aplico el formato APA en un 50 % del documento. 1 puntos	No aplico el formato APA en la construcción del documento 0 puntos
Revisión práctica	Comprueba en un 100 % la relación entre su quehacer teórico con la prácitca 10 puntos	Comprueba en un 80% % la relación entre su quehacer teórico con la prácitca 5 puntos	Comprueba en un 50 % la relación entre su quehacer teórico con la prácitca 3 puntos	No comprueba la realización de la actividad 0 puntos

50.00 / 50.00

ACTIVIDAD 3. PRÁCTICA 1 – GLOSARIO

Contenido

INSTITUTO TECNOLÓGICO DE MINATITLÁN	1
CAMPUS MINATITLÁN	1
Clave: ISC-2240	1
INGENIERÍA EN SISTEMAS COMPUTACIONALES	1
DOCENTE: MARIA CONCEPCIÓN VILLATORO CRUZ	1
ALUMNA: CRUZ MISS YESSICA YAMILET	1
5° semestre – Grupo A.....	1
Semestre Agosto – Diciembre 2023	1
LENGUAJE DE INTERFAZ.....	1
LENGUAJE DE INTERFAZ MPORTAFOLIO DE EVIDENCIA.....	1
Tema 2: PROGRAMACIÓN BÁSICAPORTAFOLIO DE EVIDENCIA MTema 1: INTRODUCCIÓN AL LENGUAJE ENSAMBLADOR	1
Tema 1: INTRODUCCIÓN AL LENGUAJE ENSAMBLADOREmACTIVIDAD 1. MI CPU: DATOS.....	1
INSTITUTO TECNOLÓGICO DE ACTIVIDAD 2. EMU8086	1
ACTIVIDAD 2. EMU8086ENACTIVIDAD 3. PRÁCTICA 1 – GLOSARIO.....	1
ALUMNA: CRUZ MISS YESSICA YAMILET	1
5° semestre – Grupo A.....	1
Semestre Agosto – DiciembTema 2: PROGRAMACIÓN BÁSICAm.....	1
Tema 1: INTRODUCCIÓN AL LENGUAJE ENSAMBLADOREmACTIVIDAD 1. MI CPU: DATOS.....	1
INSTITUTO TECNOLÓGICO DE MINATITLÁN	1
CAMPUS MINATITLÁN	1
Clave: ISC-2240	1
INGENIERÍA EN SISTEMAS COMPUTACIONALES	1
DOCENTE: MARIA CONCEPCIÓN VILLATORO CRUZ	1
ALUMNA: CRUZ MISS YESSICA YAMILET	1
5° semestre – Grupo A.....	1
Semestre Agosto – Diciembre 2023	1
LENGUAJE DE INTERFAZ.....	1
LENGUAJE DE INTERFAZ MPORTAFOLIO DE EVIDENCIA.....	1

Tema 2: PROGRAMACIÓN BÁSICA	POR	TAFOLIO DE EVIDENCIA	M
Tema 1: INTRODUCCIÓN AL LENGUAJE ENSAMBLADOR	1	
Tema 1: INTRODUCCIÓN AL LENGUAJE ENSAMBLADOR	em	ACTIVIDAD 1. MI CPU:	
DATOS	1	
INSTITUTO TECNOLÓGICO DE ACTIVIDAD 2. EMU8086	1	
ACTIVIDAD 2. EMU8086	en	ACTIVIDAD 3. PRÁCTICA 1 – GLOSARIO	1
ALUMNA: CRUZ MISS YESSICA YAMILET	1	
5° semestre – Grupo A	1	
Semestre Agosto – Diciemb	Tema 2: PROGRAMACIÓN BÁSICA	ma	1
Tema 1: INTRODUCCIÓN AL LENGUAJE ENSAMBLADOR	em	ACTIVIDAD 1. MI CPU:	
DATOS	1	
LENGUAJE DE INTERFAZ	1	
LEN	ACTIVIDAD 2. EMU8086	1
ACTIVIDAD 2. EMU8086	en	ACTIVIDAD 3. PRÁCTICA 1 – GLOSARIO	1
LENGUAJE DE INTERFAZ	1	
LEN	ACTIVIDAD 2. EMU8086	1
ACTIVIDAD 2. EMU8086	en	ACTIVIDAD 3. PRÁCTICA 1 – GLOSARIO	1
LENGUAJE DE INTERFAZ	1	
LEN	ACTIVIDAD 2. EMU8086	1
ACTIVIDAD 2. EMU8086	en	ACTIVIDAD 3. PRÁCTICA 1 – GLOSARIO	1
LENGUAJE DE INTERFAZ	1	
LEN	ACTIVIDAD 2. EMU8086	1
ACTIVIDAD 2. EMU8086	en	ACTIVIDAD 3. PRÁCTICA 1 – GLOSARIO	1
POR	TAFOLIO DE EVIDENCIA	2
Tema 2: PROGRAMACIÓN BÁSICA	PO	RTAFOLIO DE EVIDENCIA	2
Tema 2: PROGRAMACIÓN BÁSICA	ma	2
PORTAFOLIO DE EVIDENCIA	2	
Tema 2: PROGRAMACIÓN BÁSICA	PO	RTAFOLIO DE EVIDENCIA	2
Tema 2: PROGRAMACIÓN BÁSICA	ma	2
TEMA 1: INTRODUCCIÓN AL LENGUAJE ENSAMBLADOR	3	
TEMA 1: INTRODUCCIÓN AL LENGUAJE ENSAMBLADOR	3	
TEMA 1: INTRODUCCIÓN AL LENGUAJE ENSAMBLADOR	3	

ACTIVIDAD 1. MI CPU: DATOS	5
ACTIVIDAD 1. MI CPU: DATOS	5
ACTIVIDAD 1. MI CPU: DATOS	5
ACTIVIDAD 1. MI CPU: DATOS	5
ACTIVIDAD 2. EMU8086	14
INTRODUCCIÓN DEL MICROPROCESADOR 8086.....	15
INSTALACIÓN Y NAVEGACIÓN DEL EMU 8086	16
PROGRAMA CON TIPO DE DIRECCIONAMIENTO, REGISTROS Y ESTRUCTURA DE 8086.....	20
CONCLUSIONES.....	33
Bibliografía	35
RÚBRICA DE EVALUACIÓN – CALIFICACIÓN.....	36
ACTIVIDAD 3. PRÁCTICA 1 – GLOSARIO	38
ACTIVIDAD 3. PRÁCTICA 1 – GLOSARIO	38
Descripción de la práctica:	45
Bibliografía.....	52
RÚBRICA DE EVALUACIÓN – CALIFICACIÓN.....	53
CONCLUSIÓN GENERAL	54

Tema 2: PROGRAMACIÓN BÁSICA.....	57
INTRODUCCIÓN GENERAL:.....	60
SESIÓN 1. INTRODUCCIÓN AL LENGUAJE ENSAMBLADOR 8086	62
INTRODUCCIÓN SESIÓN 1:.....	63
Objetivos:	63
PRESENTACIÓN DEL LENGUAJE ENSAMBLADOR 8086 Y SU IMPORTANCIA.....	65
Presentación del lenguaje ensamblador 8086:.....	65
IMPORTANCIA DEL LENGUAJE ENSAMBLADOR 8086:	66
EXPLICACIÓN DE LA ESTRUCTURA BÁSICA DE UN PROGRAMA EN ENSAMBLADOR: SECCIONES, DIRECTIVAS Y ETIQUETAS.....	68
1. <i>Secciones:</i>	68
2. <i>Directivas:</i>	68
3. <i>Etiquetas:</i>	69
INTRODUCCIÓN A LOS REGISTROS (AX, BX, CX, DX) Y SU FUNCIÓN EN EL LENGUAJE ENSAMBLADOR.	71
CONCLUSIÓN SESIÓN 1:.....	73
Sesión 2. PRACTICA INSTRUCCIONES BÁSICAS	75
INTRODUCCIÓN SESIÓN 2:.....	76
<i>Instrucciones de Transferencia de Datos:</i>	76
<i>Instrucciones Aritméticas:</i>	76
INSTRUCCIONES DE TRANSFERENCIAS DE DATOS	78
• MOV (Mover datos):	78
• XCHG (Intercambiar datos):	79
• PUSH (Guardar en la pila):	80
• POP (Sacar una palabra de la pila):.....	80
• PUSHF (Guardar en la pila las banderas):.....	81
• POPF (Sacar de la pila las banderas):	82
• LEA (Cargar dirección efectiva):	82
INSTRUCCIONES ARITMÉTICAS	83
• ADD (Sumar números binarios):	83
• ADC (Sumar con acarreo):	84
• SUB (Restar números binarios):	85
• SBB (Restar con acarreo):	86

• DEC (Disminuye en uno)	87
• INC (Incrementa en uno):	88
• MUL (Multiplicar sin signo):.....	88
• IMUL (Multiplicar con signo (enteros)):.....	89
• DIV (Dividir sin signo):.....	89
• IDIV (Dividir con signo):.....	90
• NEG (Niega):	91
CONCLUSIÓN SESIÓN 2:	92
Sesión 3. MANEJO DE LA PILA. Instrucciones PUSH, POP, PUSHF y POPF.	93
INTRODUCCIÓN SESIÓN 3:	94
• PILA (STACK).....	96
¿CÓMO FUNCIONA LA PILA Y LAS INSTRUCCIONES PUSH, POP, PUSHF, POPF?	100
Funcionamiento de la Pila:	100
Instrucciones PUSH y POP:	101
Instrucciones PUSHF y POPF:.....	101
CONCLUSIÓN SESIÓN 3:	102
Sesión 4. INTERRUPCIONES	104
INTRODUCCIÓN SESIÓN 4	105
EN EL SIGUIENTE CÓDIGO PODRÁ VERIFICAR UN EJEMPLO SENCILLO DE LA APLICACIÓN DE LAS INTERRUPCIONES:	108
CONCLUSIÓN SESIÓN 4	111
Sesión 5. LENGUAJE ENSAMBLADOR: EMU8086	113
INTRODUCCIÓN SESIÓN 5	114
REALICE UN PROGRAMA EN LENGUAJE ENSAMBLADOR QUE MULTIPLIQUE DOS NÚMEROS:	117
CONCLUSIÓN SESIÓN 5	120
CONCLUSIÓN GENERAL	121
Bibliografía	123
TEMA 3: MODULARIZACIÓN-INTEGRACIÓN	124
LENGUAJE DE INTERFAZ	133
LENGUAJE DE INTERFAZ	133
INSTITUTO TECNOLÓGICO DE MINATITLÁN	133

CAMPUS MINATITLÁN	133
Clave: ISC-2240.....	133
INGENIERÍA EN SISTEMAS COMPUTACIONALES.....	133
DOCENTE: MARIA CONCEPCIÓN VILLATORO CRUZ.....	133
PROYECTO FINAL: PROGRAMACIÓN CON ARDUINO.....	133
INTEGRANTES:	133
• CARVAJAS HERNANDEZ JOHAN	133
• CRUZ MISS YESSICA YAMILET	133
• DIAZ ROSAS CLAUDIA BERENICE	133
• GONZALEZ ROMERO WYGALMI	133
• MARQUEZ ECHEVERRIA BRIAN.....	133
• MEJIA REALPOZO JOSE MANUEL.....	133
• RASILLA SANTOS VICTOR.....	133
• SABINO COLORADO JORGE GAEL.....	133
5° semestre – Grupo A	133
Semestre Agosto – Diciembre 2023	133
INTRODUCCIÓN.....	135
CONTEXTO DEL PROYECTO	135
OBJETIVO DEL PROYECTO	136
FUNDAMENTOS DE ARDUINO.....	138
CONCEPTOS BÁSICOS	138
ESTADO DEL ARTE.....	139
DESCRIPCIÓN DEL PROBLEMA	142
JUSTIFICACIÓN	142
DISEÑO EN TINKERCARD	144
ARQUITECTURA DEL SISTEMA	146
SELECCIÓN DE COMPONENTES	146
DIAGRAMA DE CONEXIONES.....	148
EXPLICACIÓN DEL CÓDIGO	149
ALGORITMOS Y FUNCIONES CLAVE	150
CASOS DE PRUEBAS	151
RESULTADOS DE PRUEBAS.....	152

GUÍA DE USUARIO.....	153
RESULTADOS Y CONCLUSIONES	153
TRABAJO FUTURO – MEJORAS Y EXPANSIONES	154
REFERENCIAS.....	157
ANEXOS.....	157

Descripción de la práctica:

Construya por equipo un glosario de términos relevantes, fundamentales, significativos o desconocidos que aporten en su conceptualización del tema 1.

Objetivo:

Integrar colaborativamente un glosario de términos que sean básicos para recordar y comprender en el tema de la introducción al lenguaje ensamblador.

Instrucciones:

Coloque el nombre del término en MAYUSCULAS Y NEGRITAS, seguido de la descripción (letra normal) de su conceptualización (no más de 5 renglones cada término). Integre un mínimo de 25 términos para este primer tema. Enlístelos de forma alfabética sin usar números consecutivos. Al final de los aportes coloque todas las referencias bibliográficas que utilizó para construir dicha actividad. Utilice para el documento las reglas del formato APA.

**Nota: Para realizar esta actividad de manera colaborativa, coloque este documento en un drive e inserte sus aportes dejando comentarios de su participación en la construcción de esta práctica. Entregará en la asignación de esta práctica el documento construido incluyendo dichos comentarios de la colaboración (que es la evidencia de los aportes de los integrantes del equipo).

A

ABSTRACCIÓN: Se refiere a la simplificación o la representación de un concepto o función de una manera que oculta detalles complejos o técnicos para facilitar la comprensión y la interacción del usuario. En otras palabras, es la idea de presentar información o controles de una manera que sea más fácil de entender y usar, a menudo ocultando la complejidad subyacente.

ABSTRACCIÓN DE DATOS: Se refiere a la idea de que un programa o una estructura de datos puede ocultar los detalles internos de cómo se almacenan o manipulan los datos, y en su lugar, proporciona una interfaz o un conjunto de operaciones bien definidas para acceder y manipular esos datos.

B

BIT: Dígito del sistema binario que puede tomar únicamente dos valores posibles: 0 o 1. La información se almacena y se procesa en forma de secuencias de bits, donde cada bit representa una decisión binaria, como encendido/apagado, verdadero/falso, o cualquier otra opción de dos estados.

BYTE: Secuencia de 8 bits contiguos, formando un número binario más grande. Cada bit en un byte puede tener uno de dos valores posibles: 0 o 1. Se utilizan para representar una variedad de datos, como caracteres de texto, números, imágenes, sonidos y otros tipos de información en las computadoras. En sistemas de codificación de caracteres como ASCII o Unicode, un byte puede representar un único carácter alfanumérico o un símbolo.

C

CONTROL DE BUCLES: Se refiere a la gestión y dirección del flujo de ejecución de un programa de computadora que contiene bucles o ciclos. Los bucles son estructuras de control utilizadas en programación para repetir un conjunto de instrucciones múltiples veces, y el control de bucles se encarga de determinar cuántas veces se ejecutan y bajo qué condiciones se detienen.

D

DISPOSITIVO DE ENTRADA: Aparato que se conecta a un ordenador y permite que se comunique con el exterior, dejando que el usuario transmita información al ordenador. Son dispositivos de entrada el teclado, el ratón, el micrófono, el escáner, etc.

DISPOSITIVO DE ENTRADA/SALIDA: Aparato que se conecta a un ordenador y permite que se comunique con el exterior, dejando que el usuario transmita información al ordenador y también que el ordenador transmita información al usuario. Son dispositivos de entrada/salida el lector/grabador de CD-ROM y DVD-ROM, los discos duros, la tarjeta de red, etc.

DISPOSITIVO DE SALIDA: Aparato que se conecta a un ordenador y permite que se comunique con el exterior, dejando que el ordenador transmita información al usuario. Son dispositivos de salida el monitor, los altavoces, la impresora, etc.

E

ENSAMBLADOR: Se refiere a un tipo de programa informático que se encarga de traducir un fichero fuente escrito en un lenguaje ensamblador, a un fichero objeto que contiene código máquina, ejecutable directamente por el microprocesador.

ENSAMBLADORES CRUZADOS (CROSS-ASSEMBLER): Se denominan así los ensambladores que se utilizan en una computadora que posee un procesador diferente al que tendrán las computadoras donde va a ejecutarse el programa objeto producido.

ENSAMBLADORES DE UNA FASE: Estos ensambladores leen una línea del programa fuente y la traducen directamente para producir una instrucción en lenguaje máquina o la ejecuta si se trata de una pseudoinstrucción.

ENSAMBLADORES RESIDENTES: Son aquellos que permanecen en la memoria principal de la computadora y cargan, para su ejecución, al programa objeto producido.

I

INCREMENTO Y DECREMENTO: Incrementa en uno el valor contenido dentro del registro que se le dé como parámetro y reduce en uno el valor contenido dentro del registro que se le dé como parámetro.

INSTRUCCIONES DE CÁLCULO: Indican a la CPU la operación que debe realizar. Pueden ser aritméticas (sumar, restar, etc.), lógicas (O, Y, NO, etc.), de comparación y de rotación y desplazamiento de los bits de un registro o posición de memoria hacia la izquierda o hacia la derecha, etc.

INSTRUCCIONES DE RUPTURA DE SECUENCIA: Indican a la CPU que la siguiente instrucción que debe ejecutarse no es la que corresponde según la secuencia física del programa, sino que debe realizar un salto a otro punto del mismo programa.

INSTRUCCIONES DE TRANSFERENCIA DE DATOS: Indican a la CPU que copie el dato de un registro o posición de memoria en otro registro u otra posición de memoria.

L

LENGUAJE DE ALTO NIVEL: Lenguaje que permite escribir programas más similares a la forma de pensar del ser humano, en vez de a la forma de trabajar de un ordenador concreto. Estos lenguajes permiten al programador centrarse en resolver problemas y no tener que pensar en el ordenador concreto con el que se va a ejecutar el programa.

LENGUAJE DE BAJO NIVEL: Lenguaje que permite escribir programas similares a la forma de trabajar de un ordenador concreto, en vez de a la forma de pensar del ser humano. Estos lenguajes obligan al programador a centrarse en las características del ordenador concreto con el que se va a ejecutar el programa.

LENGUAJE MAQUINA: Lenguaje de bajo nivel que permite escribir programas directamente ejecutables en un ordenador concreto. Este lenguaje está compuesto únicamente de bits y es específico de la arquitectura de dicho ordenador, existiendo un lenguaje máquina para cada tipo de máquina.

LENGUAJE ENSAMBLADOR: Lenguaje de nivel intermedio (entre bajo y alto) que permite escribir programas pensados para un tipo de ordenador concreto. Este lenguaje está compuesto de símbolos y solo obliga al programador a centrarse en las características de tipo concreto de ordenador con el que se va a trabajar, no del ordenador concreto en que se va a ejecutar el programa.

M

MACROENSAMBLADORES: Son ensambladores que permiten el uso de macroinstrucciones (macros). Debido a su potencia, normalmente son programas robustos que no permanecen en memoria una vez generados el programa objeto.

MANIPULACIÓN DE PILA: El lenguaje ensamblador cuenta con dos instrucciones para la manipulación: Push; Esta instrucción permite almacenar el contenido del operando dentro de la última posición de la pila y Pop; Esta instrucción toma el último dato almacenado en la pila y lo carga al operando.

MEMORIA DE SESIÓN: La interfaz puede recordar la actividad actual del usuario durante la sesión actual. Por ejemplo, un carrito de compras en línea puede mantener los artículos seleccionados por el usuario hasta que finalice la compra.

MEMORIA DE USUARIO REGISTRADO: Cuando un usuario inicia sesión en una aplicación o sitio web, la interfaz puede recordar su información de usuario, preferencias y configuraciones personalizadas en futuras visitas.

MEMORIA DE BÚSQUEDA: Pueden recordar búsquedas anteriores y mostrar sugerencias o historiales de búsqueda para ayudar al usuario a encontrar información más rápidamente

Bibliografía

- cORTINA, G. A. (20 de Septiembre de 2023). *Blogger*. Obtenido de BLOG 6 SEMESTRE Lenguajes de Interfaz:
<http://gilbertomtz1.blogspot.com/p/lenguajes-de-interfaz.html>
- Edición), D. D.-G. (20 de Septiembre de 2023). *lsi.vc.ehu.eus*. Obtenido de GLOSARIO DE TÉRMINOS:
<https://lsi.vc.ehu.eus/pablogn/docencia/manuales/SO/TemasSOuJaen/glosario/GLOSARIO.htm>
- Pablo Moreno, J. A. (20 de Septiembre de 2023). *fdi.ucm.es*. Obtenido de Tema 1 - Glosario:
<https://www.fdi.ucm.es/profesor/fpeinado/courses/programming/Tema1Glosario.pdf>
- Tapachula, I. T. (20 de Septiembre de 2023). *Studocu*. Obtenido de Glosario conceptos de instrucciones de lenguaje ensamblador:
<https://www.studocu.com/es-mx/document/instituto-tecnologico-de-tapachula/tecnologia-del-concreto/glosario-conceptos-de-instrucciones-de-lenguaje-ensamblador/34113914>

RÚBRICA DE EVALUACIÓN – CALIFICACIÓN

Términos de Glosario Cumplio con al menor 25 términos del glosario Puntaje máximo 4	4 / 4
Estructura Cumplio con la estructura del glosario Puntaje máximo 4	4 / 4
Congruencia Los términos del glosario con congruentes con el tema 1 de la asignatura Puntaje máximo 4	4 / 4
Formato APA Cumplio con el formato APA Puntaje máximo 4	4 / 4
Formato APA Entregó la actividad dentro del tiempo establecido Puntaje máximo 4	4 / 4
20.00 / 20.00	

CONCLUSIÓN GENERAL

La práctica de la programación en el entorno de EMU8086 y el trabajo con una CPU basada en la arquitectura x86, como la que se encuentra en las computadoras personales, es esencial para comprender los conceptos fundamentales de la programación a nivel de lenguaje ensamblador. Aquí tienes una conclusión general, detallada y extensa sobre esta práctica, incluyendo sus conceptos clave:

Conceptos Básicos:

1. Lenguaje Ensamblador: El lenguaje ensamblador es un lenguaje de bajo nivel que permite interactuar directamente con la CPU utilizando instrucciones mnemotécnicas. A diferencia de los lenguajes de alto nivel, las instrucciones en lenguaje ensamblador están estrechamente relacionadas con la arquitectura de la CPU.
2. EMU8086: EMU8086 es un entorno de desarrollo integrado (IDE) que permite escribir, ensamblar y ejecutar programas en lenguaje ensamblador para la arquitectura x86. Proporciona una plataforma de aprendizaje práctica y simulada que imita el comportamiento de una CPU real.

Estructura de una CPU x86:

Una CPU x86, como la que se emula en EMU8086, consta de varias partes clave, incluyendo:

- a. Unidad de Control (CU): Coordinación y control de las operaciones de la CPU.
- b. Unidad Aritmético-Lógica (ALU): Realización de operaciones matemáticas y lógicas.
- c. Registros: Almacenamiento de datos y valores temporales.

- d. Memoria: Almacenamiento de datos y programas, incluyendo la memoria RAM y la ROM.

Ejecución de Programas en EMU8086:

1. Escritura de Código: Se escribe el código en lenguaje ensamblador utilizando mnemotécnicos y se especifican las instrucciones y datos necesarios para la tarea deseada.
2. Ensamblaje: El código escrito se ensambla en lenguaje de máquina, que es el formato que la CPU puede entender directamente. El resultado se almacena en un archivo ejecutable.
3. Carga y Ejecución: El archivo ejecutable se carga en EMU8086, que simula una CPU x86. El programa se ejecuta, y los resultados se muestran en la salida.

Conceptos Clave en la Programación en Lenguaje Ensamblador:

- a. Instrucciones: Cada instrucción en lenguaje ensamblador realiza una tarea específica, como sumar dos números o mover datos entre registros y memoria.
- b. Modos de Direcciónamiento: Determinan cómo se accede a los operandos de una instrucción. Pueden ser inmediatos, directos, indirectos, entre otros.
- c. Registros Generales: Almacenan datos y valores temporales, como EAX, EBX, ECX, y EDX.
- d. Flujo de Control: Instrucciones de salto (jump) y condicionales permiten el control del flujo del programa.

Importancia de la Programación en Lenguaje Ensamblador:

1. Optimización de Código: La programación en lenguaje ensamblador permite un control preciso sobre el hardware, lo que puede resultar en código altamente optimizado para tareas específicas.
2. Programación de Bajo Nivel: Ayuda a comprender cómo funciona una CPU y la interacción directa con la memoria y los registros.
3. Programación de Sistemas y Controladores: Es esencial para el desarrollo de sistemas operativos, controladores de hardware y software de tiempo real.

En resumen, la práctica de programación en EMU8086 y la comprensión de los conceptos de lenguaje ensamblador son fundamentales para programadores y profesionales de la informática que deseen trabajar en niveles de bajo nivel y obtener un mayor conocimiento de la arquitectura de una CPU x86. Aunque es un enfoque más complejo y detallado en comparación con los lenguajes de alto nivel, la programación en lenguaje ensamblador proporciona un nivel de control y optimización que es esencial en muchas aplicaciones informáticas.

Tema 2: PROGRAMACIÓN BÁSICA

INDICE

Tema 2: PROGRAMACIÓN BÁSICA	57
INTRODUCCIÓN GENERAL:	60
SESIÓN 1. INTRODUCCIÓN AL LENGUAJE ENSAMBLADOR 8086	62
INTRODUCCIÓN SESIÓN 1:	63
Objetivos:	63
PRESENTACIÓN DEL LENGUAJE ENSAMBLADOR 8086 Y SU IMPORTANCIA	65
Presentación del lenguaje ensamblador 8086:.....	65
IMPORTANCIA DEL LENGUAJE ENSAMBLADOR 8086:	66
EXPLICACIÓN DE LA ESTRUCTURA BÁSICA DE UN PROGRAMA EN ENSAMBLADOR: SECCIONES, DIRECTIVAS Y ETIQUETAS.	68
1. <i>Secciones:</i>	68
2. <i>Directivas:</i>	68
3. <i>Etiquetas:</i>	69
INTRODUCCIÓN A LOS REGISTROS (AX, BX, CX, DX) Y SU FUNCIÓN EN EL LENGUAJE ENSAMBLADOR.	71
CONCLUSIÓN SESIÓN 1:	73
Sesión 2. PRACTICA INSTRUCCIONES BÁSICAS	75
INTRODUCCIÓN SESIÓN 2:	76
<i>Instrucciones de Transferencia de Datos:</i>	76
<i>Instrucciones Aritméticas:</i>	76
INSTRUCCIONES DE TRANSFERENCIAS DE DATOS	78
• MOV (Mover datos):	78
• XCHG (Intercambiar datos):	79
• PUSH (Guardar en la pila):	80
• POP (Sacar una palabra de la pila):	80
• PUSHF (Guardar en la pila las banderas):	81
• POPF (Sacar de la pila las banderas):	82
• LEA (Cargar dirección efectiva):	82
INSTRUCCIONES ARITMÉTICAS	83
• ADD (Sumar números binarios):	83
• ADC (Sumar con acarreo):	84
• SUB (Restar números binarios):	85

• SBB (Restar con acarreo):	86
• DEC (Disminuye en uno)	87
• INC (Incrementa en uno):	88
• MUL (Multiplicar sin signo):.....	88
• IMUL (Multiplicar con signo (enteros)):.....	89
• DIV (Dividir sin signo):.....	89
• IDIV (Dividir con signo):.....	90
• NEG (Niega):	91
CONCLUSIÓN SESIÓN 2:	92
Sesión 3. MANEJO DE LA PILA. Instrucciones PUSH, POP, PUSHF y POPF.	93
INTRODUCCIÓN SESIÓN 3:	94
• PILA (STACK).....	96
¿CÓMO FUNCIONA LA PILA Y LAS INSTRUCCIONES PUSH, POP, PUSHF, POPF?	100
Funcionamiento de la Pila:	100
Instrucciones PUSH y POP:	101
Instrucciones PUSHF y POPF:.....	101
CONCLUSIÓN SESIÓN 3:	102
Sesión 4. INTERRUPCIONES	104
INTRODUCCIÓN SESIÓN 4	105
EN EL SIGUIENTE CÓDIGO PODRÁ VERIFICAR UN EJEMPLO SENCILLO DE LA APLICACIÓN DE LAS INTERRUPCIONES:	108
CONCLUSIÓN SESIÓN 4	111
Sesión 5. LENGUAJE ENSAMBLADOR: EMU8086.....	113
INTRODUCCIÓN SESIÓN 5	114
REALICE UN PROGRAMA EN LENGUAJE ENSAMBLADOR QUE MULTIPLIQUE DOS NÚMEROS:	117
CONCLUSIÓN SESIÓN 5	120
CONCLUSIÓN GENERAL	121
Bibliografía.....	123

INTRODUCCIÓN GENERAL:

El lenguaje ensamblador, el lenguaje de máquina y el programa EMU8086 desempeñan un papel esencial en el mundo de la programación y la informática. Estas herramientas son fundamentales para comprender cómo funcionan las computadoras a nivel de bajo nivel, permitiendo un control preciso sobre el hardware y la creación de programas eficientes. En esta introducción extensa, exploraremos cada uno de estos conceptos en detalle.

Lenguaje de Máquina:

El lenguaje de máquina es el nivel más bajo de programación que comprende una computadora. Está compuesto por una serie de instrucciones binarias o códigos de operación que la CPU (Unidad Central de Procesamiento) de una computadora puede entender y ejecutar directamente. Estas instrucciones se representan en formato binario, lo que las hace prácticamente ilegibles para los humanos. Cada CPU tiene su conjunto único de instrucciones de máquina que varía según la arquitectura del procesador. Por ejemplo, la arquitectura x86, ampliamente utilizada en computadoras personales, tiene su propio conjunto de instrucciones de máquina.

Un programa escrito en lenguaje de máquina es extremadamente eficiente, ya que se ejecuta directamente en la CPU sin necesidad de traducciones adicionales. Sin embargo, escribir programas en lenguaje de máquina es una tarea extremadamente tediosa y propensa a errores, ya que se requiere conocer el conjunto de instrucciones específicas del procesador objetivo y trabajar con códigos binarios. Por lo tanto, se crearon lenguajes más humanamente legibles, como el lenguaje ensamblador, para facilitar la programación a nivel de bajo nivel.

Lenguaje Ensamblador:

El lenguaje ensamblador es un nivel de programación de bajo nivel que actúa como una representación legible por humanos de las instrucciones de máquina. Cada instrucción en lenguaje ensamblador corresponde directamente a una instrucción de máquina, pero se expresa en mnemotécnicos, que son abreviaturas más comprensibles para los programadores. Estas mnemotécnicas son específicas de la arquitectura de la CPU que se esté utilizando, lo que permite escribir programas que se ejecutan directamente en la máquina objetivo.

Un programa escrito en lenguaje ensamblador se llama "código ensamblador" y debe ser traducido a código de máquina antes de que la computadora pueda ejecutarlo. Este proceso de traducción se denomina "ensamblaje" y se realiza mediante un programa llamado "ensamblador". El código ensamblador consta de secciones como ` `.data` (para datos), ` `.code` (para código de programa) y ` `.stack` (para gestionar la pila de llamadas). Los programadores también pueden declarar variables, etiquetas y definir la lógica del programa utilizando mnemotécnicas como MOV (mover), ADD (sumar), JMP (salto) y muchas otras.

EMU8086:

EMU8086 es un programa de emulación de microprocesador y un entorno de desarrollo integrado (IDE) diseñado para facilitar la programación en lenguaje ensamblador, específicamente para la arquitectura x86. Ofrece una interfaz de usuario amigable que permite a los programadores escribir, ensamblar y depurar programas ensambladores sin necesidad de hardware físico. EMU8086 emula un procesador Intel 8086, una de las primeras CPUs de la arquitectura x86, que es ampliamente utilizada en computadoras personales.

Los programadores pueden utilizar EMU8086 para escribir programas ensambladores, ensamblarlos y ejecutarlos en un entorno de emulación que simula el funcionamiento de una computadora real. EMU8086 ofrece herramientas de depuración que permiten identificar y corregir errores en los programas, lo que es esencial para el desarrollo de software a nivel de máquina. Además, ofrece la ventaja de ser una plataforma de desarrollo accesible y segura, lo que lo convierte en una excelente opción para principiantes y estudiantes que deseen aprender sobre la programación a bajo nivel.

En resumen, el lenguaje de máquina es la forma más básica de programación de computadoras, mientras que el lenguaje ensamblador actúa como un intermediario legible por humanos. EMU8086 es un programa de emulación y desarrollo que ayuda a los programadores a escribir y depurar programas en lenguaje ensamblador para la arquitectura x86, brindando una introducción valiosa al emocionante mundo de la programación a nivel de máquina.



SESIÓN 1.

INTRODUCCIÓN

AL LENGUAJE

ENSAMBLADOR

8086

INTRODUCCIÓN SESIÓN 1:

El lenguaje ensamblador es un nivel de programación de bajo nivel que se encuentra entre el lenguaje de máquina y los lenguajes de alto nivel. Se utiliza para escribir programas que se ejecutan directamente en la arquitectura de una computadora, permitiendo un control preciso sobre el hardware y una eficiencia óptima. EMU8086 es un emulador de microprocesador y entorno de desarrollo integrado (IDE) que permite a los programadores escribir y depurar programas en lenguaje ensamblador para la arquitectura x86.

Esta práctica tiene como objetivo familiarizar a los estudiantes con los conceptos fundamentales del lenguaje ensamblador y cómo trabajar con EMU8086 para escribir, ensamblar y depurar programas en esta plataforma. A lo largo de esta práctica, los participantes aprenderán sobre la estructura y sintaxis básica del lenguaje ensamblador, así como las herramientas proporcionadas por EMU8086 para facilitar el desarrollo y la depuración de programas.

Objetivos:

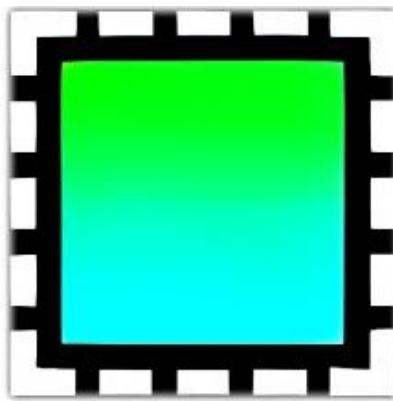
1. *Introducción al Lenguaje Ensamblador*: Comprender los conceptos básicos del lenguaje ensamblador, incluyendo el conjunto de instrucciones, registros, modos de direccionamiento y la estructura de un programa ensamblador.
2. *Instalación y Configuración de EMU8086*: Aprender cómo instalar y configurar EMU8086 en un entorno de desarrollo. Conocer la interfaz de usuario y las herramientas disponibles.
3. *Estructura de un Programa en Lenguaje Ensamblador*: Explorar la estructura típica de un programa en lenguaje ensamblador, que incluye secciones como .data, .code y .stack. Comprender cómo declarar variables y constantes.
4. *Instrucciones Básicas del Lenguaje Ensamblador*: Aprender las instrucciones fundamentales del lenguaje ensamblador, incluyendo operaciones aritméticas, de flujo de control y de transferencia de datos.

5. *Ensamblaje y Ejecución de Programas:* Aprender a escribir programas en lenguaje ensamblador utilizando EMU8086. Aprender a ensamblar y ejecutar programas para observar su funcionamiento.
6. *Depuración de Programas:* Utilizar las capacidades de depuración de EMU8086 para identificar y corregir errores en programas ensambladores.
7. *Prácticas y Ejercicios:* Realizar una serie de ejercicios y prácticas que incluyen la escritura de programas sencillos, cálculos aritméticos, manipulación de cadenas y toma de decisiones.
8. *Proyecto Final:* Desarrollar un proyecto final que demuestre las habilidades adquiridas a lo largo de la práctica. Este proyecto incluye la creación de un programa que multiplique dos números.

En resumen, esta práctica tiene como objetivo proporcionar a los participantes una base sólida en el lenguaje ensamblador y su implementación en EMU8086. A medida que avancen en la práctica, se adquirirán las habilidades necesarias para escribir programas en lenguaje ensamblador y comprenderán cómo estos programas se traducen en operaciones de bajo nivel en una computadora. Al finalizar la práctica, los participantes estarán preparados para abordar proyectos más complejos y explorar el fascinante mundo de la programación a nivel de máquina :v.

PRESENTACIÓN DEL LENGUAJE ENSAMBLADOR 8086 Y SU IMPORTANCIA

El lenguaje ensamblador 8086 es un lenguaje de programación de bajo nivel utilizado para programar computadoras basadas en la arquitectura x86, particularmente los procesadores Intel 8086 y sus sucesores, que incluyen una amplia variedad de microprocesadores utilizados en computadoras personales y servidores.



Presentación del lenguaje ensamblador 8086:

1. *Bajo nivel:* El lenguaje ensamblador 8086 es un lenguaje de bajo nivel que se encuentra un paso por encima del código máquina, lo que significa que las instrucciones están estrechamente relacionadas con la arquitectura del hardware y son específicas de un procesador en particular.
2. *Representación simbólica:* Aunque es más legible que el código máquina, el lenguaje ensamblador sigue siendo una representación simbólica de las instrucciones del procesador. Cada instrucción ensambladora se corresponde directamente con una instrucción de máquina, lo que facilita la programación a nivel de hardware.
3. *Programación directa del hardware:* El lenguaje ensamblador 8086 permite a los programadores interactuar directamente con el hardware de la computadora. Esto es especialmente útil en aplicaciones donde se requiere

un control preciso de los recursos de hardware, como la programación de controladores de dispositivos o sistemas embebidos.

4. *Rendimiento:* Debido a su proximidad con el hardware, el código ensamblador puede ser altamente optimizado para aprovechar al máximo la potencia de cálculo del procesador. Esto lo hace importante en aplicaciones donde el rendimiento es crítico, como juegos, software de gráficos y sistemas operativos.
5. *Educación:* El lenguaje ensamblador 8086 es a menudo utilizado en cursos de informática y electrónica para enseñar los conceptos fundamentales de la programación y la arquitectura de computadoras. Ayuda a los estudiantes a comprender cómo funcionan los procesadores y cómo se ejecutan las instrucciones a nivel de máquina.
6. *Depuración y análisis de código:* El lenguaje ensamblador es una herramienta valiosa para depurar programas a nivel de máquina y analizar el comportamiento de un programa en detalle. Esto es útil en situaciones en las que se necesita rastrear y corregir errores en software o entender problemas de rendimiento.

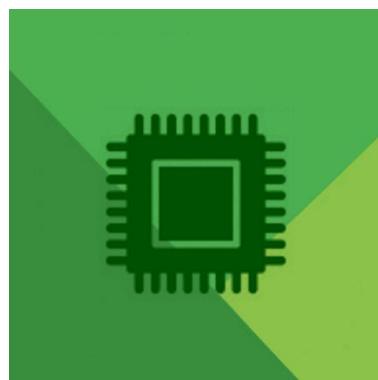
IMPORTANCIA DEL LENGUAJE ENSAMBLADOR 8086:

1. *Compatibilidad:* Aunque la arquitectura x86 ha evolucionado desde el 8086 original, el lenguaje ensamblador 8086 sigue siendo importante debido a la compatibilidad retroactiva. Los procesadores modernos aún pueden ejecutar código 8086, lo que es útil para mantener aplicaciones más antiguas o sistemas heredados.
2. *Control de hardware:* En aplicaciones donde se necesita un control preciso del hardware, como controladores de dispositivos o sistemas embebidos, el lenguaje ensamblador 8086 es fundamental para interactuar directamente con el hardware.
3. *Optimización de rendimiento:* La programación en lenguaje ensamblador permite una optimización extrema de rendimiento, lo que es crítico en

aplicaciones de alto rendimiento como juegos, software de gráficos y sistemas operativos.

4. *Conocimiento de la arquitectura:* Aprender el lenguaje ensamblador 8086 proporciona una comprensión profunda de la arquitectura de la CPU, lo que es valioso para programadores que trabajan en áreas relacionadas con la informática, incluso si no programan directamente en ensamblador.

En resumen, el lenguaje ensamblador 8086 es un lenguaje de bajo nivel fundamental para interactuar con la arquitectura x86, proporcionando control de hardware y la capacidad de optimizar el rendimiento. Aunque su uso directo ha disminuido en muchas áreas de desarrollo de software, sigue siendo importante en aplicaciones específicas donde el rendimiento y el control de hardware son críticos.



EXPLICACIÓN DE LA ESTRUCTURA BÁSICA DE UN PROGRAMA EN ENSAMBLADOR: SECCIONES, DIRECTIVAS Y ETIQUETAS.

La estructura básica de un programa en lenguaje ensamblador consta de secciones, directivas y etiquetas.

1. Secciones:

Las secciones son partes fundamentales de un programa en ensamblador y definen cómo se organiza el código y los datos en el programa. Las dos secciones más comunes en un programa ensamblador son:

- Sección de código (Code Section): Aquí se coloca el código que se ejecutará. Contiene las instrucciones en lenguaje ensamblador que realizarán las operaciones deseadas.
- Sección de datos (Data Section): Esta sección almacena datos utilizados por el programa, como variables, constantes o estructuras de datos. Los datos pueden ser inicializados o no inicializados.

Un programa en ensamblador puede contener otras secciones específicas según sus necesidades, como secciones para almacenar tablas de consulta, secciones para datos constantes, etc.

2. Directivas:

Las directivas son instrucciones especiales que el ensamblador utiliza para controlar cómo se ensambla y organiza el programa. Algunas de las directivas más comunes incluyen:

- ORG (Origin): Esta directiva establece la dirección de origen para la generación del código ensamblado. Define dónde comenzará la ejecución del programa.

- *DB (Define Byte), DW (Define Word), DD (Define Doubleword), DQ (Define Quadword), DT (Define Ten Bytes)*: Estas directivas se utilizan para declarar y asignar valores a variables en la sección de datos. Dependiendo de la directiva, se pueden definir bytes, palabras, enteros de 32 bits, números en coma flotante, etc.
- *SEGMENT y ENDS*: Estas directivas se utilizan para delimitar secciones en el programa y son especialmente relevantes en programas ensambladores para la arquitectura x86. Por ejemplo, se puede definir un segmento de código y un segmento de datos utilizando estas directivas.
- *INCLUDE*: La directiva INCLUDE se utiliza para importar archivos externos en un programa ensamblador. Esto es útil para dividir un programa en múltiples archivos y facilitar la organización del código.
- *EQU (Equate)*: La directiva EQU se utiliza para asignar un valor constante a una etiqueta. Esto puede ser útil para definir constantes en el programa.

3. Etiquetas:

Las etiquetas son nombres simbólicos que se utilizan para marcar ubicaciones específicas en el programa y para referenciar datos o instrucciones. Algunas características importantes de las etiquetas incluyen:

- Las etiquetas son precedidas por dos puntos (por ejemplo, `etiqueta:`).
- Se utilizan para indicar el destino de saltos o llamadas en el programa.
- También se utilizan para nombrar variables o constantes en la sección de datos.
- Ayudan a hacer que el código sea más legible y comprensible al proporcionar nombres significativos a las ubicaciones o datos.

En resumen, la estructura básica de un programa en ensamblador implica la organización de secciones de código y datos, el uso de directivas para controlar la

generación del código y etiquetas para marcar ubicaciones o datos. La combinación de estos elementos permite escribir programas en lenguaje ensamblador que pueden realizar diversas operaciones a nivel de máquina.

Etiquetas	Operación	Operandos	Comentarios
INICIO			
	<code>movlw</code>	0x07	;Carga primer sumando en W
	<code>addlw</code>	0x08	;Suma W con segundo sumando
	<code>movwf</code>	RESULTADO	;Almacena el resultado
END			;Fin del programa fuente

INTRODUCCIÓN A LOS REGISTROS (AX, BX, CX, DX) Y SU FUNCIÓN EN EL LENGUAJE ENSAMBLADOR.

Los registros AX, BX, CX y DX son registros de propósito general en la arquitectura x86, que incluye el lenguaje ensamblador 8086. Estos registros desempeñan un papel fundamental en el procesamiento de datos y cálculos en lenguaje ensamblador. Cada uno de ellos tiene una función específica:

1. *AX (Acumulador)*: El registro AX se utiliza comúnmente para operaciones aritméticas y de datos. Es especialmente útil para almacenar resultados temporales de cálculos. Se divide en dos registros de 8 bits, AL y AH, que pueden usarse de forma independiente para tareas específicas. AL se utiliza para trabajar con datos de 8 bits, mientras que AH puede utilizarse para cálculos con enteros con signo o para almacenar datos auxiliares.
2. *BX (Base)*: El registro BX se utiliza frecuentemente para almacenar direcciones de memoria o como un registro base en operaciones de índice o desplazamiento. Puede ser empleado en combinación con el registro SI o DI para acceder a estructuras de datos en memoria.
3. *CX (Contador)*: El registro CX se utiliza típicamente para controlar bucles y contar iteraciones. Su valor se disminuye o aumenta en cada iteración de un bucle. Es especialmente útil con las instrucciones de repetición (por ejemplo, REP, REPE, REPNE) para realizar operaciones repetitivas.
4. *DX (Datos Extendidos)*: El registro DX se utiliza en una variedad de tareas, como manejar resultados extendidos de multiplicaciones y divisiones. También puede usarse para almacenar puertos de E/S (Entrada/Salida) en operaciones de E/S.

Estos registros son llamados "registros de propósito general" porque pueden utilizarse para una amplia variedad de tareas en un programa en lenguaje ensamblador. En muchas instrucciones ensambladoras, puedes elegir qué registro

utilizar para operaciones específicas según tus necesidades. Además de estos registros, la arquitectura x86 también incluye otros registros importantes, como los registros de segmento (CS, DS, ES, SS), registros de puntero (SI, DI, BP, SP), registros de índice (SI y DI), y otros registros de control y banderas (como el registro de banderas FLAGS).

Los registros son una parte esencial de la programación en lenguaje ensamblador, ya que permiten el almacenamiento temporal de datos y el control de flujos en un programa. El conocimiento y el uso adecuado de estos registros son fundamentales para la programación eficiente en esta arquitectura.



CONCLUSIÓN SESIÓN 1:

En esta breve conclusión, resumiremos los puntos clave abordados en relación con el lenguaje ensamblador 8086, su estructura básica de programa y la importancia de los registros en el lenguaje ensamblador.

1. *Presentación del Lenguaje Ensamblador 8086 y su Importancia:*

En esta práctica, hemos explorado el lenguaje ensamblador 8086, que se encuentra en el nivel de programación de bajo nivel y actúa como una interfaz legible por humanos para interactuar con el hardware de una computadora. Hemos destacado la importancia de este lenguaje al permitir un control preciso sobre el hardware y una eficiencia óptima en la programación. El lenguaje ensamblador 8086 es fundamental para comprender la arquitectura subyacente de las computadoras y es la base de muchos sistemas operativos y aplicaciones de software.

2. *Explicación de la Estructura Básica de un Programa en Ensamblador: Secciones, Directivas y Etiquetas:*

Hemos analizado la estructura básica de un programa en lenguaje ensamblador, que se compone de varias secciones esenciales, como `.data`, `.code`, y `.stack`. Estas secciones sirven para organizar y gestionar los datos, el código del programa y la pila de llamadas. Además, hemos explicado cómo se utilizan directivas como `db`, `dw`, y `dd` para declarar variables y constantes en la sección `.data`. Las etiquetas se utilizan para marcar ubicaciones en el programa y se convierten en puntos de referencia para instrucciones de salto y acceso a datos.

3. *Los Registros (AX, BX, CX, DX) y su Función en el Lenguaje Ensamblador:*

Hemos abordado la función crucial de los registros en el lenguaje ensamblador 8086. Los registros, como AX, BX, CX y DX, son áreas de almacenamiento de datos de propósito general dentro del procesador. Cada registro tiene un propósito específico y se utiliza para llevar a cabo diversas operaciones, como almacenar datos temporales, realizar cálculos y transferir información. Estos registros son esenciales para la manipulación de datos y el flujo de control en un programa ensamblador.

En resumen, hemos explorado la importancia del lenguaje ensamblador 8086 como una herramienta fundamental para programar a nivel de máquina. Hemos comprendido la estructura básica de un programa en ensamblador, incluyendo secciones, directivas y etiquetas, que son esenciales para organizar y gestionar el código y los datos. Además, hemos reconocido la función crítica de los registros (AX, BX, CX, DX) en el lenguaje ensamblador, ya que desempeñan un papel central en la manipulación de datos y la ejecución de operaciones matemáticas. Estos conceptos proporcionan una base sólida para explorar en profundidad el mundo de la programación a nivel de máquina y comprender cómo los programas interactúan directamente con el hardware de una computadora.



Sesión 2.

PRACTICA

INSTRUCCIONES

BÁSICAS

INTRODUCCIÓN SESIÓN 2:

EMU8086 es un emulador y entorno de desarrollo ampliamente utilizado para programar en lenguaje ensamblador y trabajar con la arquitectura x86 de microprocesadores. Al programar en EMU8086, es fundamental comprender las instrucciones de transferencia de datos y las instrucciones aritméticas, ya que forman la base de la programación a nivel de bajo nivel. A continuación, se presentan breves introducciones a estas dos categorías de instrucciones:

Instrucciones de Transferencia de Datos:

Se utilizan para mover información entre registros y memoria, así como entre registros mismos. Estas instrucciones permiten cargar datos en registros, almacenar resultados intermedios y recuperar valores desde la memoria. Algunas de las instrucciones de transferencia de datos más comunes incluyen:

- MOV
- XCHG
- PUSH
- POP
- PUSHF
- POPF
- LEA

Operación	MOV: Mover datos
Descripción	Transfiere datos entre dos registros o entre un registro y memoria, y permite llevar datos inmediatos a un registro o a memoria.
Banderas	No las afecta.
Formato	MOV {registro/memoria}, {registro/memoria/inmediato}
Ejemplo	MOV AX, 54AFH

Instrucciones Aritméticas:

Son esenciales para realizar operaciones matemáticas en datos almacenados en registros o en memoria. Estas instrucciones permiten llevar a cabo operaciones

como sumas, restas, multiplicaciones y divisiones. Algunas de las instrucciones aritméticas básicas incluyen:

- ADD
- ADC
- SUB
- SBB
- DEC
- INC
- MUL
- IMUL
- DIV
- IDIV
- NEG

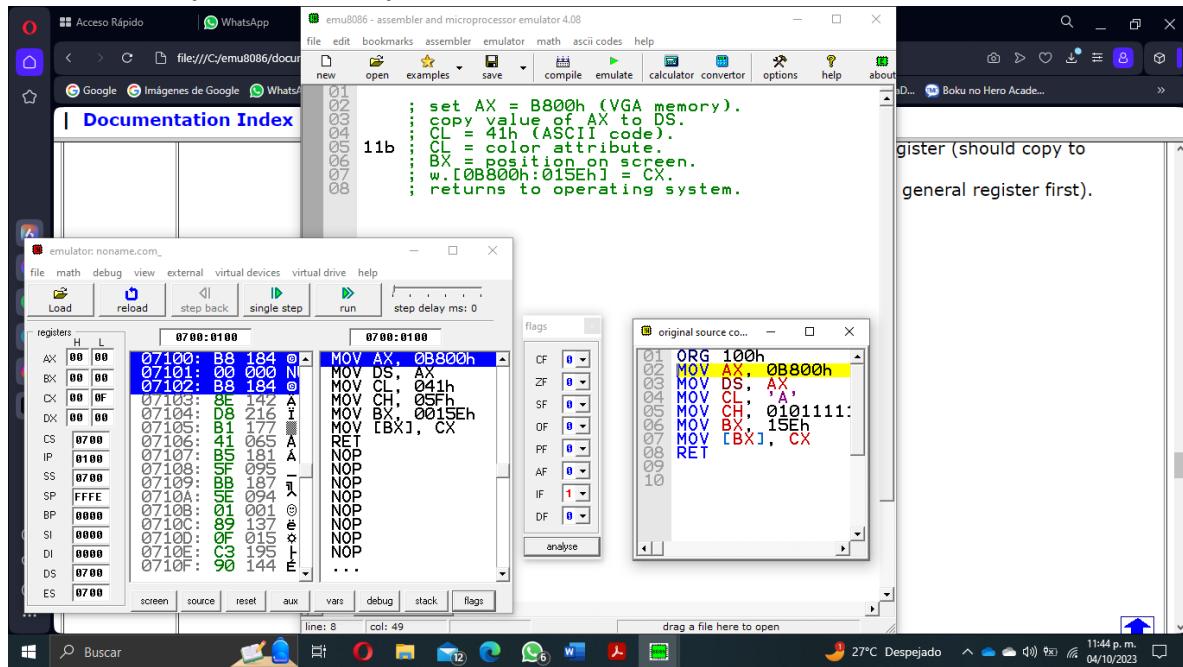
	Assembly instruction	Comment
SUMA →	MOV A, #63H MOV R3, #23H MOV 0D0H, #0 ADD A, R3	; A = 63H ; R3 = 23H ; PSW = 00H (clear PSW) ; ACC = 86H, PSW = 05H (0000 0101B)
RESTA →	MOV A, R7 CLR C SUBB A, R6 MOV R7, A	; A = R7 ; C = 0 ; A = A - R6 ; R7 = A
MULTI →	MOV A, #7BH MOV 0F0H, #02H MUL AB MOV 0F0H, #0FEH MUL AB	; A = 7BH ; B = 02H ; A = F6H and B = 00H, OV = 0 ; B = FEH ; A = 14H and B = F4H, OV = 1

Comprender y utilizar estas instrucciones es esencial para desarrollar programas en lenguaje ensamblador en EMU8086. Estas operaciones son la base de la manipulación de datos y los cálculos en la arquitectura x86, lo que permite crear programas eficientes y controlar el flujo de datos en un sistema informático.

INSTRUCCIONES DE TRANSFERENCIAS DE DATOS

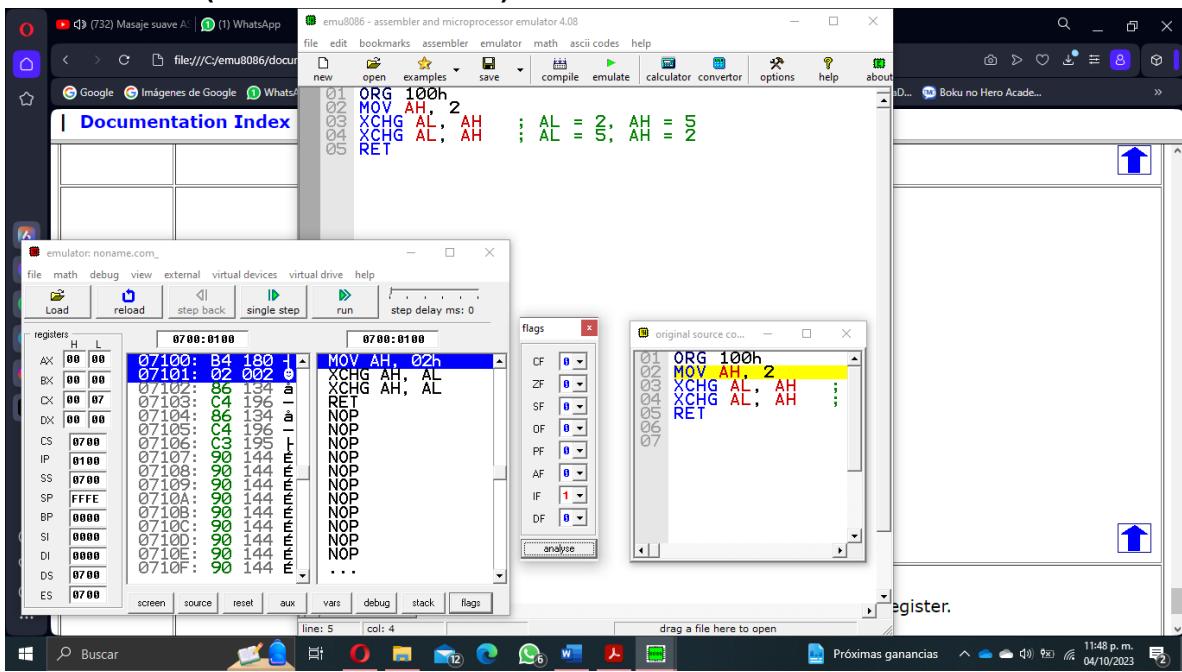
Su misión es intercambiar la información entre los registros y las posiciones de memoria. Las operaciones de este tipo más relevante son:

- **MOV (Mover datos):**



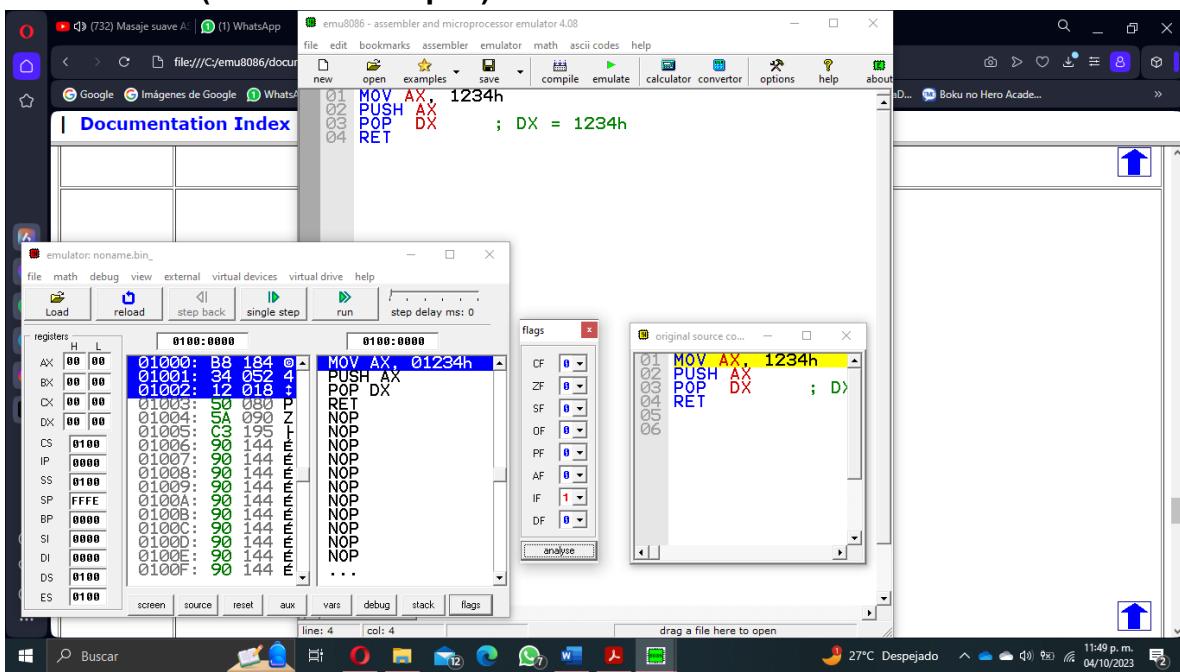
Descripción: Transfiere datos entre dos registros o entre un registro y memoria, y permite llevar datos inmediatos a un registro o a memoria.

- **XCHG (Intercambiar datos):**



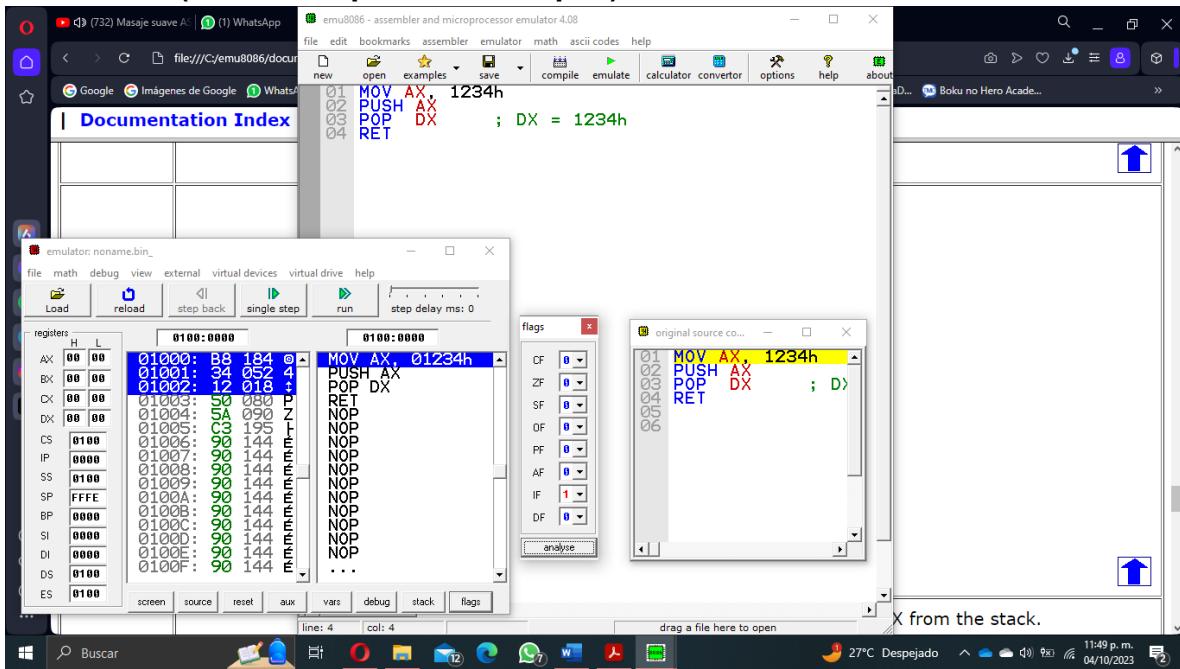
Descripción: Intercambia datos entre dos registros o entre un registro y memoria.

- **PUSH (Guardar en la pila):**



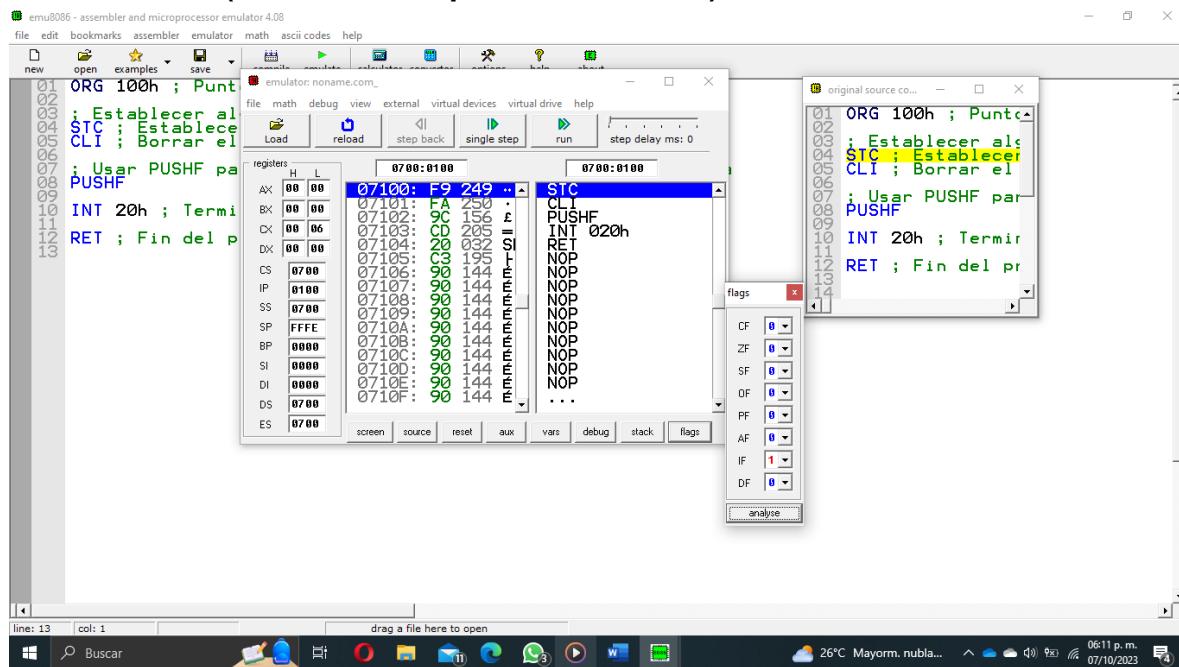
Descripción: Guarda en la pila una palabra para su uso posterior. SP apunta al tope de la pila; PUSH decrementa SP en 2 y transfiere la palabra a SS:SP.

- **POP (Sacar una palabra de la pila):**



Descripción: Saca de la pila una palabra previamente guardada y la envía a un destino especificada. SP apunta al tope de la pila; POP la transfiere al destino especificado e incrementa SP en 2.

- **PUSHF (Guardar en la pila las banderas):**



Descripción: Guarda en la pila el contenido del registro de banderas para su uso posterior.

- **POPF (Sacar de la pila las banderas):**

The screenshot shows the emu8086 interface with the following assembly code:

```

01 ORG 100h ; Punt
02
03 ; Establecer al
04 STC ; Establecer el
05 CLI ; Borrar el
06
07 ; Usar PUSHF pa
08 PUSHF
09
10 INT 20h ; Termi
11
12 RET ; Fin del p
13

```

The Registers window shows the state at address 0700:0100:

Register	Value
AX	00 00
BX	00 00
CX	00 00
DX	00 00
CS	0700
IP	0100
SS	0700
SP	FFFE
BP	0000
SI	0000
DI	0000
DS	0700
ES	0700

The Flags window shows the state at address 0700:0100:

Flag	Value
CF	0
ZF	0
SF	0
OF	0
PF	0
AF	0
IF	1
DF	0

Descripción: Saca una palabra de la pila y la manda al registro de estado.

- **LEA (Cargar dirección efectiva):**

The screenshot shows the emu8086 interface with the following assembly code:

```

01 MOV BX, 35h
02 MOV DI, 12h
03 LEA SI, [BX+DI] ; SI = 35h + 12h = 47h
Note: The integrated 8086 assembler automatica
04 org 100h
05 LEA AX, m ; AX = offset of m
06
07 RET
08 m dw 1234h
09
10 END

```

The Registers window shows the state at address 0700:0100:

Register	Value
AX	BB 187
BX	00 00
CX	00 00
DX	00 00
CS	0700
IP	0100
SS	0700
SP	FFFE
BP	0000
SI	0000
DI	0000
DS	0700
ES	0700

The Flags window shows the state at address 0700:0100:

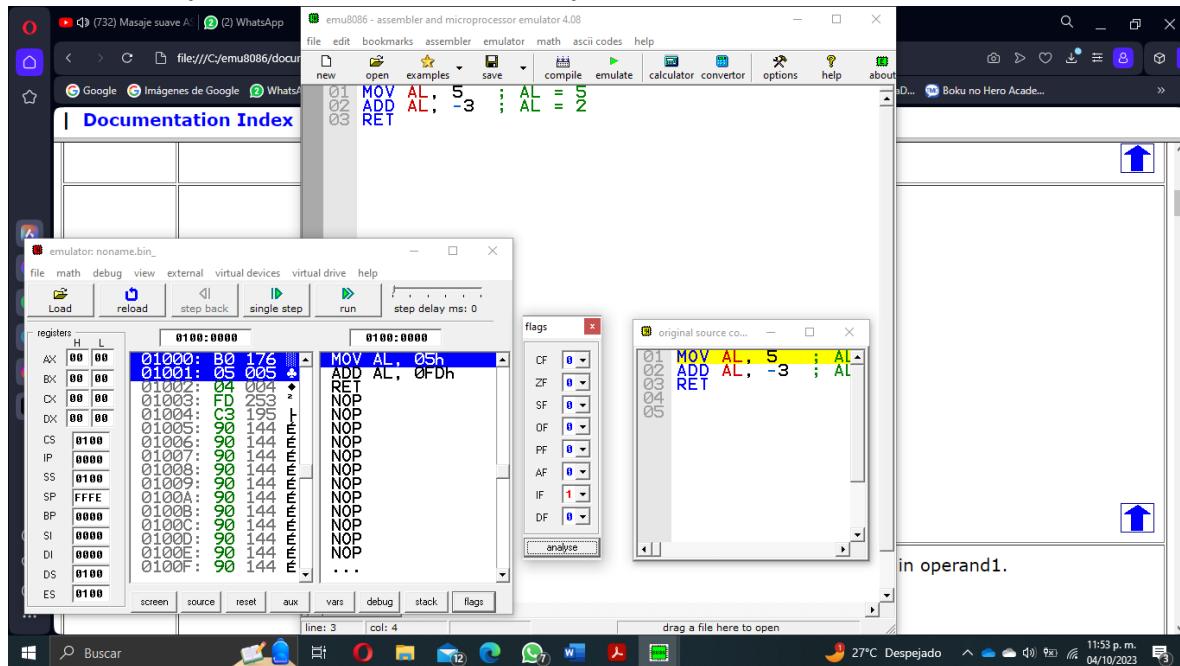
Flag	Value
CF	0
ZF	0
SF	0
OF	0
PF	0
AF	0
IF	1
DF	0

Descripción: Carga una dirección cercana en un registro.

INSTRUCCIONES ARITMÉTICAS

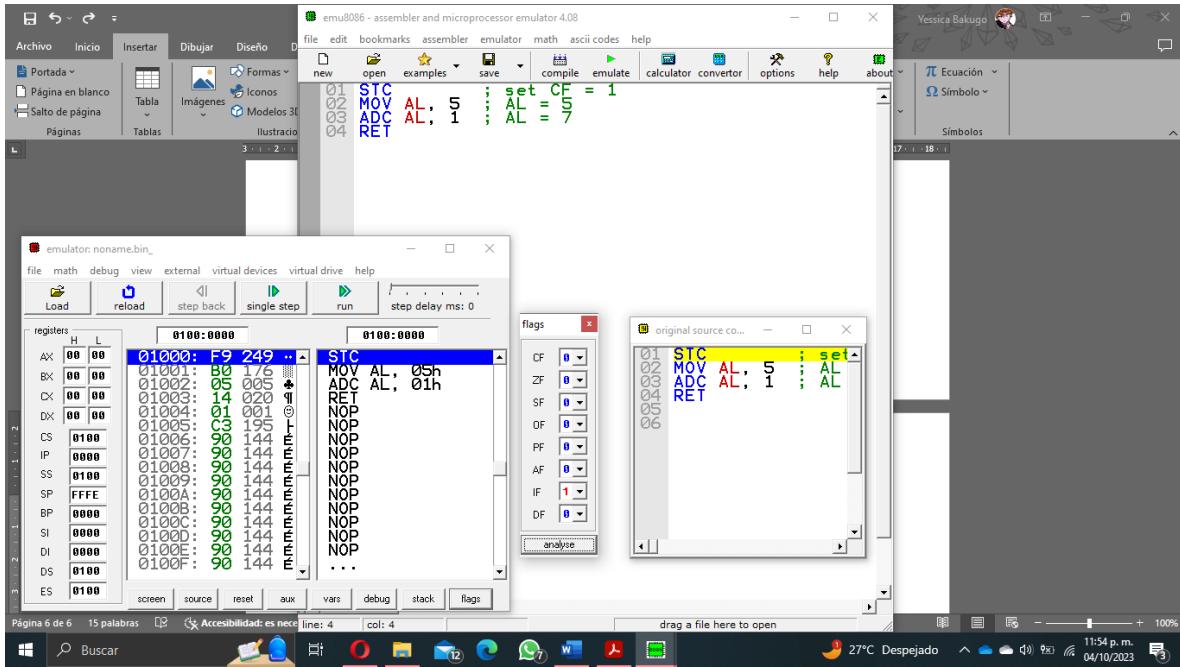
Sirven para llevar a cabo operaciones aritméticas manipulando los registros y las posiciones de memoria:

- ADD (Sumar números binarios):



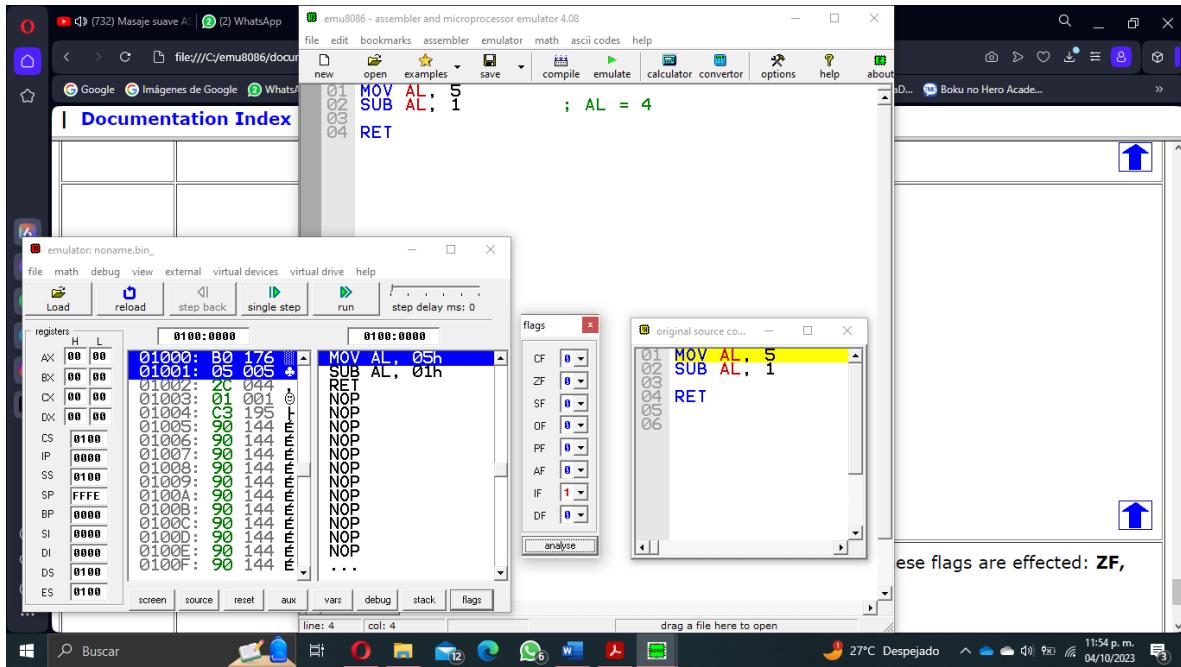
Descripción: Suma números binarios desde la memoria, registro o inmediato a un registro, o suma números en un registro o inmediato a memoria. Los valores pueden ser un byte o una palabra.

- ADC (Sumar con acarreo):



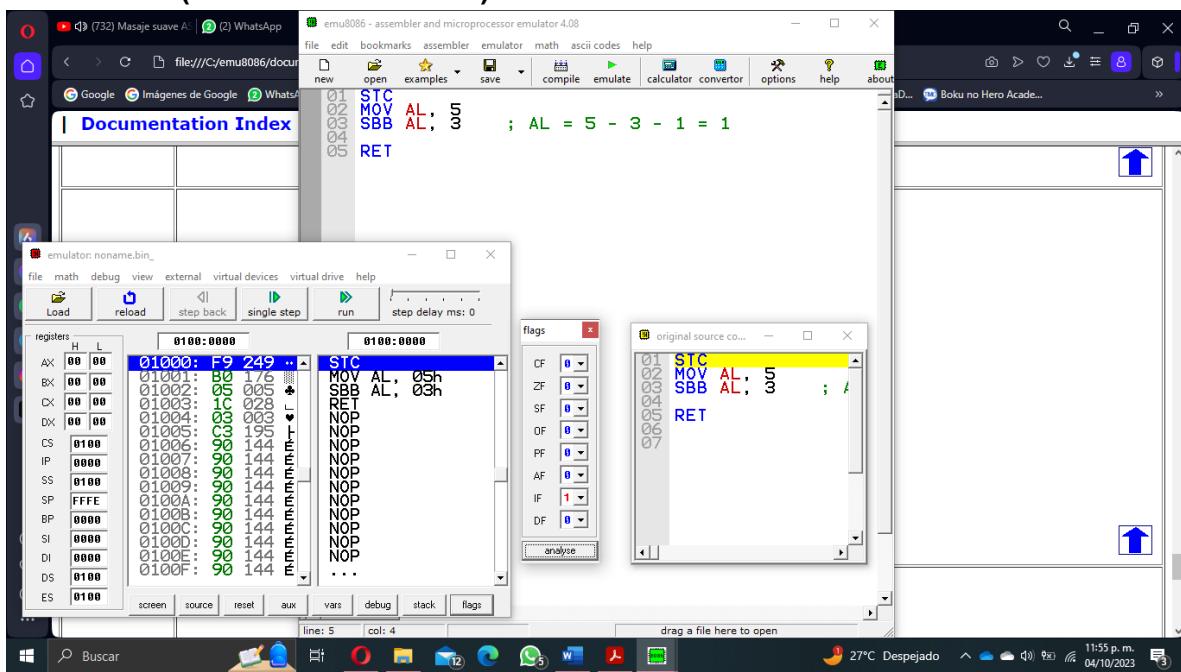
Descripción: Por lo común es usado en suma de múltiples palabras binarias para acarrear un bit en el siguiente paso de la operación. ADC suma el contenido de la bandera CF al primer operando y después suma el segundo operando al primero, al igual que ADD.

- **SUB (Restar números binarios):**



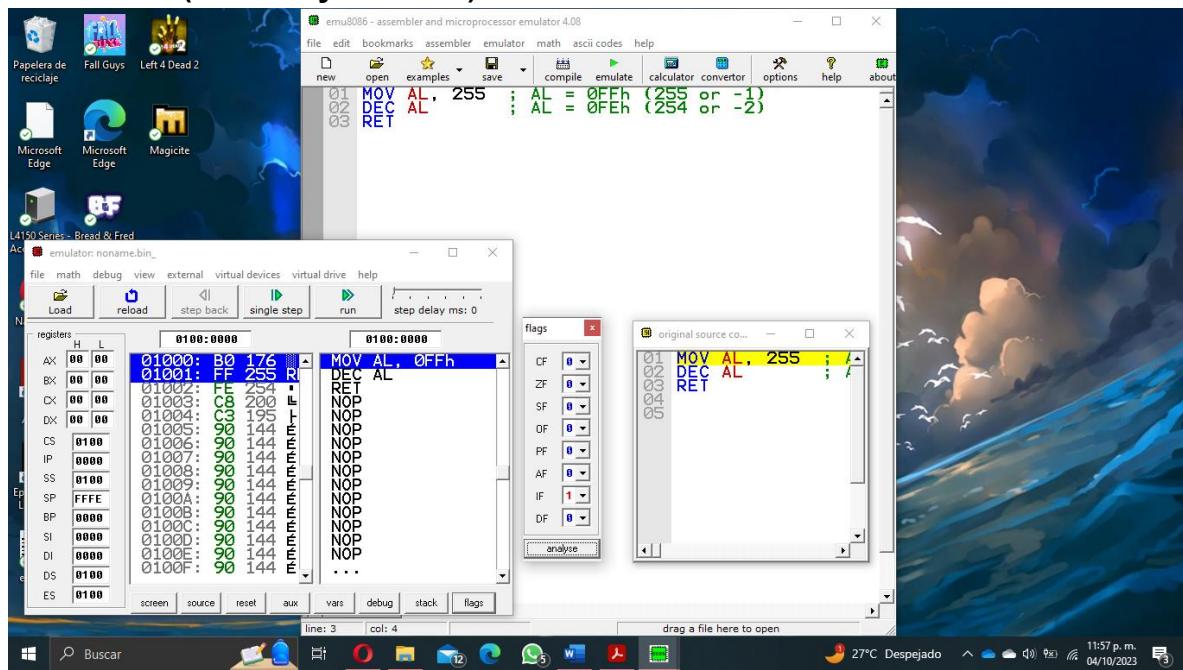
Descripción: Resta números binarios en un registros, memoria o inmediato de un registro, o resta valores en un registro o inmediato de memoria.

- **SBB (Restar con acarreo):**



Descripción: Normalmente, se usa esta operación en la resta binaria de múltiples palabras para acarrear el bit uno de desbordamiento al siguiente paso de la aritmética. SBB resta primero el contenido de la bandera CF del primer operando y después el segundo operando del primero, de manera similar a SUB.

- **DEC (Disminuye en uno)**



Descripción: Disminuye 1 de un byte o una palabra en un registro o memoria.

- **INC (Incrementa en uno):**

The screenshot shows the emu8086 software interface. In the assembly editor window, the code is:

```

01: MOV AL, 4
02: INC AL      ; AL = 5
03: RET

```

In the registers window, the AX register is set to 00 00. The assembly window shows the instruction at address 0100:0000.

Descripción: Incrementa en uno un byte o una palabra en un registro o memoria.

- **MUL (Multiplicar sin signo):**

The screenshot shows the emu8086 software interface. In the assembly editor window, the code is:

```

01: MOV AL, 200 ; AL = 0C8h
02: MOV BL, 4
03: MUL BL      ; AX = 0320h (800)
04: RET

```

In the registers window, the AX register is set to 00 00. The assembly window shows the instruction at address 0100:0000.

Descripción: Multiplicar sin signo.

- **IMUL (Multiplicar con signo (enteros)):**

The screenshot shows the emu8086 software interface. The assembly code window displays the following instructions:

```

    01 MOV AL, -2
    02 MOV BL, -4
    03 IMUL BL ; AX = 8
    04 RET

```

The registers window shows the state of CPU registers at address 0100:

Register	Value	Description
AX	00 00	Initial value
BX	00 00	Initial value
CX	00 00	Initial value
DX	00 00	Initial value
CS	0100	Code Segment
IP	0000	Initial value
SS	0100	Stack Segment
SP	FFFE	Initial value
BP	0000	Initial value
SI	0000	Initial value
DI	0000	Initial value
DS	0100	Initial value
ES	0100	Initial value

The flags window shows the current state of CPU flags:

Flag	Value
CF	0
ZF	1
SF	0
OF	0
PF	0
AF	0
IF	1
DF	0

Descripción: Multiplica dos operandos con signos. IMUL trata el bit de más a la izquierda como el signo.

- **DIV (Dividir sin signo):**

The screenshot shows the emu8086 software interface. The assembly code window displays the following instructions:

```

    01 MOV AX, 203 ; AX = 00CBh
    02 MOV BL, 4
    03 DIV BL ; AL = 50 (32h), AH = 3
    04 RET

```

The registers window shows the state of CPU registers at address 0100:

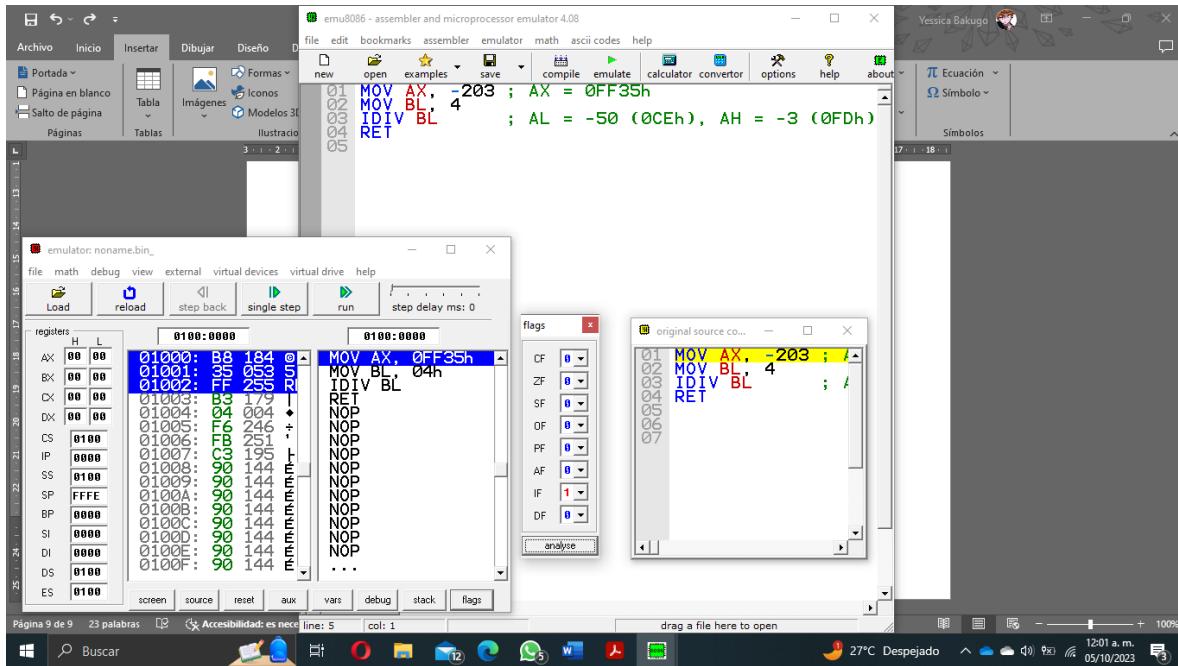
Register	Value	Description
AX	00 00	Initial value
BX	00 00	Initial value
CX	00 00	Initial value
DX	00 00	Initial value
CS	0100	Code Segment
IP	0000	Initial value
SS	0100	Stack Segment
SP	FFFE	Initial value
BP	0000	Initial value
SI	0000	Initial value
DI	0000	Initial value
DS	0100	Initial value
ES	0100	Initial value

The flags window shows the current state of CPU flags:

Flag	Value
CF	0
ZF	0
SF	0
OF	0
PF	0
AF	0
IF	1
DF	0

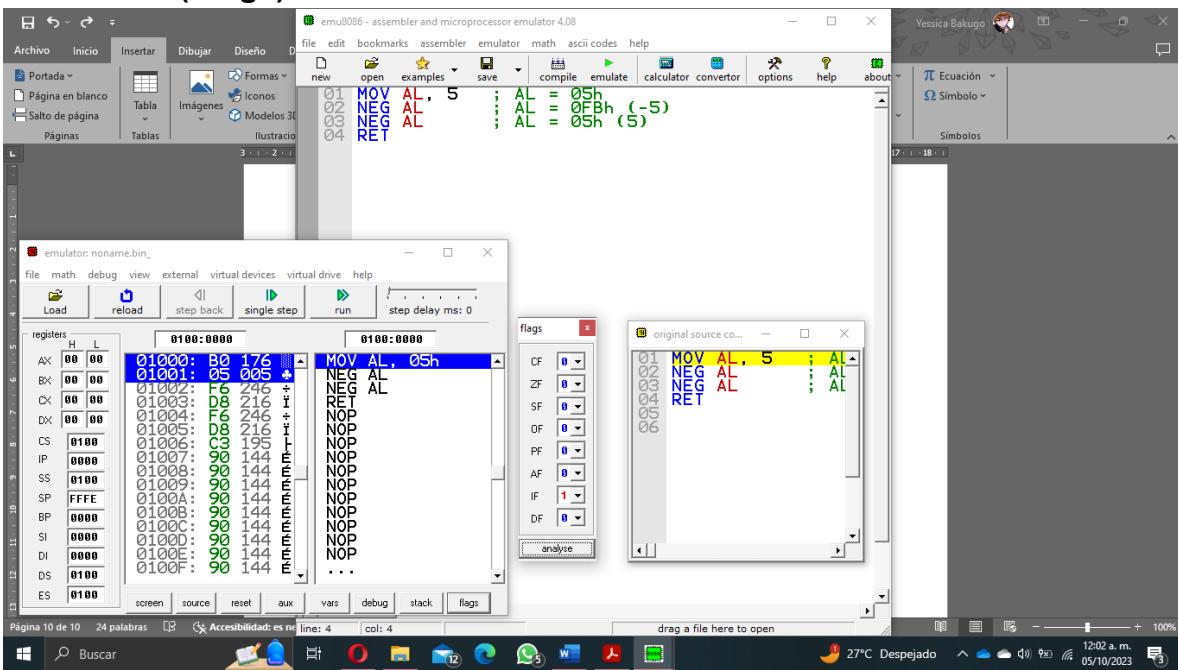
Descripción: Divide un dividendo sin signo entre un divisor sin signo. La división entre cero provoca una interrupción de división entre cero.

- **IDIV (Dividir con signo):**



Descripción: Divide un dividendo con signo entre un divisor con signo. La división entre cero provoca una interrupción de división entre cero. IDIV trata el bit de la izquierda como el signo.

- **NEG (Niega):**



Descripción: Invierte un número binario de un positivo a negativo y viceversa. NEG trabaja realizando el complemento a dos.

CONCLUSIÓN SESIÓN 2:

En conclusión, EMU8086 es un entorno de desarrollo y emulador ampliamente utilizado para programar en lenguaje ensamblador y trabajar con la arquitectura x86 de microprocesadores. Dos categorías fundamentales de instrucciones en EMU8086 son las instrucciones de transferencia de datos y las instrucciones aritméticas.

Las instrucciones de transferencia de datos, como MOV, XCHG y LEA, permiten mover información entre registros y memoria, así como entre registros mismos, lo que es esencial para cargar, almacenar y recuperar datos durante la ejecución del programa.

Por otro lado, las instrucciones aritméticas, como ADD, SUB, MUL y DIV, son fundamentales para realizar operaciones matemáticas en datos almacenados en registros o memoria, lo que facilita la realización de cálculos y manipulación de datos.

Comprender y utilizar estas instrucciones es esencial para desarrollar programas en lenguaje ensamblador en EMU8086, ya que forman la base de la programación a nivel de bajo nivel y permiten controlar eficazmente el flujo de datos en una computadora. Con este conocimiento, los programadores pueden escribir código eficiente y efectivo para la arquitectura x86.



Sesión 3. MANEJO DE LA PILA.



Instrucciones PUSH, POP, PUSHF v POPF.

INTRODUCCIÓN SESIÓN 3:

El manejo de la pila es una parte esencial de la programación en lenguaje ensamblador y desempeña un papel crítico en la gestión de datos y la administración del flujo de ejecución de un programa. En el contexto de la arquitectura x86 y EMU8086, la pila es una estructura de datos que se utiliza para almacenar temporalmente valores y registros, lo que permite a los programas realizar llamadas a funciones, mantener un seguimiento de las direcciones de retorno y administrar datos de manera eficiente.

A continuación, se presenta una introducción al manejo de la pila y a las instrucciones PUSH, POP, PUSHF y POPF:

Manejo de la Pila:

La pila se organiza como una estructura de datos tipo LIFO (Last-In-First-Out), lo que significa que el último elemento que se coloca en la pila es el primero en ser retirado. Esto se asemeja a una pila de platos, donde siempre tomas el plato superior. En la programación en ensamblador, la pila se utiliza para:

1. Almacenamiento Temporal
2. Administración de Llamadas a Funciones
3. Preservación de Registros

Instrucciones de la Pila

1. PUSH
2. POP
3. PUSHF (Push Flags)
4. POPF (Pop Flags)

El manejo de la pila es fundamental para la programación en ensamblador, ya que permite un flujo de ejecución ordenado y una gestión eficiente de datos y registros. Las instrucciones PUSH, POP, PUSHF y POPF son herramientas esenciales en el proceso de administración de la pila y son ampliamente utilizadas en programas de ensamblador.

Procedimiento

empilar:

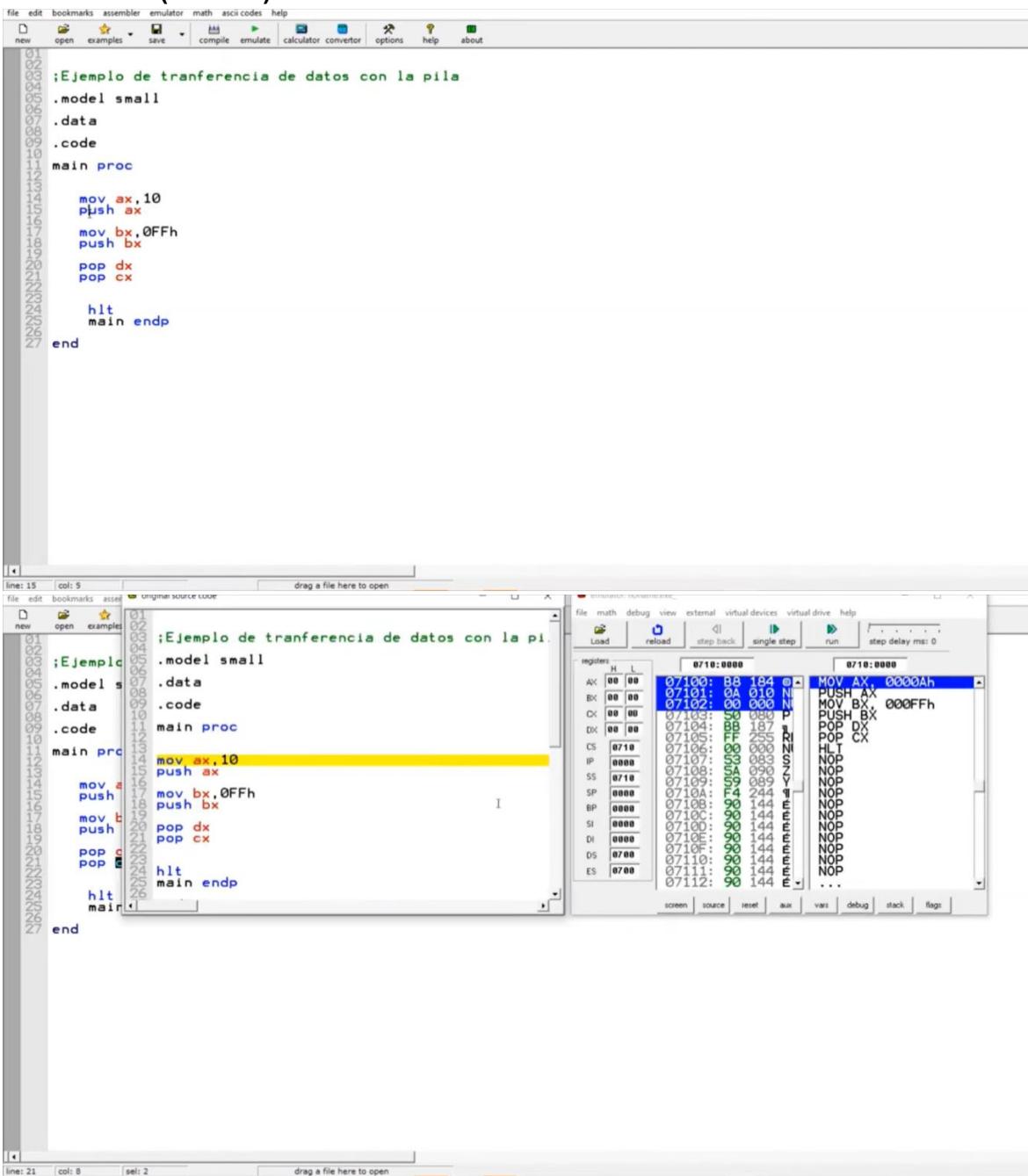
push registroOrigen

desempilar:

pop registroDestino

Recuperar el último valor de la pila y asignarlo al registro destino.

• PILA (STACK)



The screenshot shows a debugger interface with two main panes. The left pane displays the assembly source code:

```

01 ;Ejemplo de transferencia de datos con la pila
02
03 .model small
04
05 .data
06
07 .code
08
09 .main proc
10
11
12     mov ax,10
13     push ax
14
15     mov bx,0FFh
16     push bx
17
18     pop dx
19     pop cx
20
21     hlt
22
23     main endp
24
25
26
27 end

```

The right pane shows the registers and memory dump. The registers window shows:

	H	L
AX	00	00
BX	00	00
CX	00	00
DX	00	00
CS	0710	
IP	0000	
SS	0710	
SP	0000	
BP	0000	
SI	0000	
DI	0000	
DS	0700	
ES	0700	

The memory dump window shows the assembly code at address 0710:0000:

Address	OpCode	Mnemonic	Operands
0710:0000	B8 10	MOV AX,	0000Ah
0710:0001	0A 01	PUSH AX	
0710:0002	00 00	MOV BX,	000FFh
0710:0003	50 00	PUSH BX	
0710:0004	BB 18	POP DX	
0710:0005	FF 25	POP CX	
0710:0006	00 00	HLT	
0710:0007	S3 00	NOP	
0710:0008	S3 00	NOP	
0710:0009	S3 00	NOP	
0710:000A	F4 44	NOP	
0710:000B	90 44	NOP	
0710:000C	90 44	NOP	
0710:000D	90 44	NOP	
0710:000E	90 44	NOP	
0710:000F	90 44	NOP	
0711:0000	90 44	NOP	
0711:0001	90 44	NOP	
0711:0002	90 44	NOP	

line: 21 col: 8 sel: 2 drag a file here to open

```

01 ;Ejemplo de transferencia de datos con la pi.
02
03 .model small
04
05 .model std
06
07 .data
08
09 .code
10 main proc
11
12 mov ax,10
13 push ax
14 mov bx,0FFh
15 push bx
16 mov bx
17 push bx
18 pop dx
19 pop cx
20
21 hlt
22 main endp
23
24
25
26
27 end

```

Registers window:

	H	L	Value	Description
Ax	00	00	0710: 002A 0000	MOV AX, 0000Ah
Bx	00	00	07101: 0A 010 N	PUSH AX
Cx	00	00	07102: 00 000 N	MOV BX, 000FFh
Dx	00	00	07103: BB 184 P	PUSH BX
Si	0710		07104: FF 255 N	POP DX
Ip	0000		07105: 00 000 N	POP CX
Ss	0710		07106: 00 000 N	HLT
Sf	0000		07107: S3 083 S	NOP
Bp	0000		07108: SA 090 V	NOP
Sp	0000		07109: 59 081 V	NOP
Br	0000		0710A: F4 244 E	NOP
Dr	0000		0710B: 90 144 E	NOP
Ir	0000		0710C: 90 144 E	NOP
Dr	0000		0710D: 90 144 E	NOP
Ir	0000		0710E: 90 144 E	NOP
Dr	0700		0710F: 90 144 E	NOP
Ir	0700		07110: 90 144 E	NOP
Dr	0700		07111: 90 144 E	NOP
Ir	0700		07112: 90 144 E	NOP

Stack window:

Address	Value
0710:0000	0ABB <
0710:0001	9090
0710:0002	90F4
0710:0003	595A
0710:0004	5300
0710:0005	FFBB
0710:0006	5000

line: 21 col: 8 sel: 2 drag a file here to open

```

01 ;Ejemplo de tranferencia de datos con la pi.
02
03 .model small
04
05 .model std
06
07 .data
08
09 .code
10 main proc
11
12 mov ax,10
13 push ax
14 mov bx,0FFh
15 push bx
16 mov bx
17 push bx
18 pop dx
19 pop cx
20
21 hlt
22 main endp
23
24
25
26
27 end

```

Registers window:

	H	L	Value	Description
Ax	00	00	0710: 0032 0000	MOV AX, 0000Ah
Bx	00	00	07101: 0A 010 N	PUSH AX
Cx	00	00	07102: 00 000 N	MOV BX, 000FFh
Dx	00	00	07103: BB 187 P	PUSH BX
Si	0710		07104: FF 255 N	POP DX
Ip	0000		07105: 00 000 N	POP CX
Ss	0710		07106: 00 000 N	HLT
Sf	0000		07107: S3 083 S	NOP
Bp	0000		07108: SA 090 V	NOP
Sp	0000		07109: 59 081 V	NOP
Br	0000		0710A: F4 244 E	NOP
Dr	0000		0710B: 90 144 E	NOP
Ir	0000		0710C: 90 144 E	NOP
Dr	0000		0710D: 90 144 E	NOP
Ir	0000		0710E: 90 144 E	NOP
Dr	0700		0710F: 90 144 E	NOP
Ir	0700		07110: 90 144 E	NOP
Dr	0700		07111: 90 144 E	NOP
Ir	0700		07112: 90 144 E	NOP

Stack window:

Address	Value
0710:0000	0ABB <
0710:0001	9090
0710:0002	90F4
0710:0003	595A
0710:0004	5300
0710:0005	FFBB
0710:0006	5000

line: 21 col: 8 sel: 2 drag a file here to open

```

01 ;Ejemplo de transferencia de datos con la pi.
02
03 .model small
04 .model std
05 .data
06 .code
07 main proc
08
09     mov ax,10
10    push ax
11
12    mov bx,0FFh
13    push bx
14
15    mov b
16    push b
17
18    pop dx
19    pop cx
20
21    hlt
22
23 main endp
24
25
26
27 end

```

line: 21 col: 8 sel: 2 drag a file here to open

```

01 ;Ejemplo de transferencia de datos con la pi.
02
03 .model small
04 .model std
05 .data
06 .code
07 main proc
08
09     mov ax,10
10    push ax
11
12    mov bx,0FFh
13    push bx
14
15    mov b
16    push b
17
18    pop dx
19    pop cx
20
21    hlt
22
23 main endp
24
25
26
27 end

```

line: 21 col: 8 sel: 2 drag a file here to open

Register	Value	Description
AX	00 00	MOV AX, 0000Ah
BX	00 00	PUSH AX
CX	00 00	MOV BX, 000FFh
DX	00 00	PUSH BX
IP	0004	POP DX
SS	0710	POP CX
SP	FFFE	HLT
BP	0000	
SI	0000	
DI	0000	
DS	0700	
ES	0700	

line: 27 col: 86

```

01 ;Ejemplo de transferencia de datos con la pi.
02
03 .model small
04 .model s
05 .data
06 .code
07 main proc
08
09 main proc
10 mov ax,10
11 push ax
12 mov bx,0FFh
13 push bx
14 mov bx,0FFh
15 push bx
16 mov dx,0FFh
17 push dx
18 pop dx
19 pop cx
20 pop cx
21 hlt
22 main endp
23
24
25
26
27 end

```

Stack

0710:FFFF	000A
0710:FFFC	00F
0710:FFFA	0000
0710:FFFB	0000
0710:FFC6	0000
0710:FFC4	0000
0710:FFC2	0000
0710:FFC0	0000
0710:FFE9	0000
0710:FFE8	0000
0710:FFE6	0000
0710:FFE4	0000
0710:FFE2	0000
0710:FFD8	0000
0710:FFD6	0000
0710:FFD4	0000
0710:FFD2	0000
0710:FFD0	0000
0710:FFCE	0000
0710:FFCC	0000

Registers

Ax	00 00
Bx	00 FF
Cx	00 00
Dx	00 FF
CS	0710
IP	0009
SS	0710
SF	FFFE
BP	0000
SI	0000
DI	0000
DS	0700
ES	0700

0710:0009

07100: B8	184	0 -
07101: 0A	010	N
07102: 00	000	N
07103: 50	080	P
07104: BB	187	R
07105: FF	255	R
07106: 00	000	N
07107: 53	083	S
07108: 5A	089	Z
07109: 59	089	V
0710A: F4	244	W
0710B: 90	144	E
0710C: 90	144	E
0710D: 90	144	E
0710E: 90	144	E
0710F: 90	144	E
07110: 90	144	E
07111: 90	144	E
07112: 90	144	E

0710:000A

07100: B8	184	0 -
07101: 0A	010	N
07102: 00	000	N
07103: 50	080	P
07104: BB	187	R
07105: FF	255	R
07106: 00	000	N
07107: 53	083	S
07108: 5A	089	Z
07109: 59	089	V
0710A: F4	244	W
0710B: 90	144	E
0710C: 90	144	E
0710D: 90	144	E
0710E: 90	144	E
0710F: 90	144	E
07110: 90	144	E
07111: 90	144	E
07112: 90	144	E

¿CÓMO FUNCIONA LA PILA Y LAS INSTRUCCIONES PUSH, POP, PUSHF, POPF?

La pila es una estructura de datos fundamental en programación y se utiliza ampliamente en lenguaje ensamblador y otros lenguajes de programación para almacenar temporalmente datos y registros. Funciona siguiendo el principio de "Last-In-First-Out" (LIFO), lo que significa que el último elemento que se coloca en la pila es el primero en ser retirado. Veamos cómo funciona la pila y las instrucciones PUSH, POP, PUSHF y POPF en el contexto de la arquitectura x86:

Funcionamiento de la Pila:

1. *Puntero de Pila (ESP)*: La pila en la arquitectura x86 se gestiona mediante un registro especial llamado "ESP" (Stack Pointer) que apunta al elemento superior de la pila. Inicialmente, se establece en la dirección de la parte superior de la pila.
2. *Instrucciones PUSH y POP*: Para agregar un valor o un registro a la pila, se utiliza la instrucción `PUSH`. Esto decrementa el puntero de la pila (ESP) y almacena el valor en la ubicación a la que apunta ESP. Para retirar un valor de la pila, se utiliza la instrucción `POP`, que recupera el valor de la ubicación a la que apunta ESP y luego incrementa ESP.
3. *Acceso a la Pila*: La pila se encuentra en la memoria y generalmente crece hacia abajo en direcciones de memoria decrecientes. Esto significa que cuando se empuja un valor en la pila, el puntero de la pila (ESP) se decrementa para apuntar al nuevo elemento. Cuando se retira un valor de la pila, ESP se incrementa para apuntar al siguiente elemento.

Instrucciones PUSH y POP:

- `PUSH`: La instrucción `PUSH` se utiliza para empujar (colocar) un valor o un registro en la parte superior de la pila. Por ejemplo, `PUSH AX` tomará el valor en el registro AX y lo almacenará en la ubicación de memoria apuntada por ESP. Luego, ESP se decrementa.
- `POP`: La instrucción `POP` se utiliza para retirar (sacar) un valor de la parte superior de la pila y cargarlo en un registro. Por ejemplo, `POP BX` retirará el valor en la ubicación de memoria apuntada por ESP y lo colocará en el registro BX. Luego, ESP se incrementa.

Instrucciones PUSHF y POPF:

- `PUSHF`: La instrucción `PUSHF` se utiliza para empujar el estado de las banderas (flags) de la CPU en la pila. Esto es útil para preservar el estado de las banderas antes de modificarlas en el código. Cuando se ejecuta `PUSHF`, se empujan las banderas en la pila.
- `POPF`: La instrucción `POPF` se utiliza para retirar el estado de las banderas previamente empulado en la pila y cargarlo en las banderas de la CPU. Esto permite restaurar el estado de las banderas a un valor previamente guardado.

En resumen, la pila es una estructura de datos fundamental en programación que se utiliza para almacenar temporalmente datos y registros. Las instrucciones `PUSH` y `POP` permiten colocar y retirar valores de la pila, mientras que `PUSHF` y `POPF` se utilizan para preservar y restaurar el estado de las banderas de la CPU. Estas instrucciones son esenciales para el control del flujo de ejecución y la gestión de datos en programas escritos en lenguaje ensamblador y otros lenguajes.

CONCLUSIÓN SESIÓN 3:

La práctica de análisis de cómo funciona la pila y las instrucciones PUSH, POP, PUSHF y POPF proporciona una visión profunda de uno de los conceptos esenciales en programación a nivel de máquina y lenguaje ensamblador. A lo largo de esta práctica, hemos explorado en detalle la estructura y el funcionamiento de la pila, así como las instrucciones clave que se utilizan para interactuar con ella en la arquitectura x86. A continuación, se presenta una conclusión exhaustiva de lo que hemos aprendido en esta práctica.

La Función Fundamental de la Pila:

Hemos comprendido que la pila es una estructura de datos que se utiliza para gestionar la memoria temporal en un programa. Funciona según el principio de "último en entrar, primero en salir" (LIFO), lo que significa que el último elemento que se coloca en la pila es el primero en salir. Esto la hace ideal para gestionar llamadas a funciones, almacenar registros temporales y realizar seguimiento de la ejecución del programa.

Instrucciones PUSH y POP:

Hemos analizado las instrucciones PUSH y POP, que son las herramientas clave para interactuar con la pila. La instrucción PUSH se utiliza para colocar un valor en la parte superior de la pila, mientras que la instrucción POP se utiliza para extraer un valor de la parte superior de la pila. Estas instrucciones son cruciales en la administración de llamadas a funciones y el almacenamiento de registros temporales.

Instrucciones PUSHF y POPF:

También hemos explorado las instrucciones PUSHF y POPF, que se utilizan para almacenar y restaurar el estado de las banderas de la CPU en la pila. Esto es fundamental para mantener la integridad del programa y garantizar que las operaciones aritméticas y lógicas se realicen correctamente.

La Importancia del Seguimiento de la Pila:

Hemos reconocido la importancia de realizar un seguimiento adecuado de la pila para evitar errores y pérdida de datos. Comprender cuándo y cómo usar las instrucciones PUSH, POP, PUSHF y POPF es esencial para garantizar que el programa funcione de manera eficiente y sin problemas.

Seguridad y Eficiencia:

La práctica también ha destacado la necesidad de utilizar estas instrucciones con cuidado para garantizar la seguridad y la eficiencia del programa. Un mal uso de la pila puede resultar en desbordamientos o desbordamientos de pila, lo que podría causar bloqueos o errores en el programa.

Aplicaciones Prácticas:

Finalmente, hemos discutido cómo estas instrucciones y el conocimiento de la pila se aplican en situaciones del mundo real. Desde la gestión de llamadas a funciones hasta el manejo de excepciones y el almacenamiento de estados críticos, el análisis de la pila y las instrucciones PUSH, POP, PUSHF y POPF son fundamentales en el desarrollo de software a nivel de máquina.

En resumen, esta práctica nos ha proporcionado una comprensión profunda de la pila y las instrucciones clave para su gestión en la arquitectura x86. Hemos aprendido cómo estas herramientas son fundamentales en la programación a nivel de máquina, asegurando que los programas funcionen correctamente y de manera eficiente. El conocimiento adquirido en esta práctica es esencial para cualquier programador que busque trabajar en un nivel más bajo de abstracción y comprender en detalle cómo funcionan las computadoras a nivel de hardware.



Sesión 4.

INTERRUPCIONES

INTRODUCCIÓN SESIÓN 4

EMU8086 es una plataforma de emulación de microprocesador basada en la arquitectura x86, diseñada para facilitar el desarrollo y la ejecución de programas escritos en lenguaje ensamblador. Uno de los aspectos fundamentales para comprender y aprovechar EMU8086 al máximo es conocer en profundidad el manejo de registros y las llamadas de interrupciones. En esta introducción, exploraremos detalladamente estos conceptos y su importancia en el contexto de EMU8086.

- **Registros en EMU8086:**

Los registros son áreas de almacenamiento de datos en la CPU de una computadora que se utilizan para llevar a cabo operaciones y retener información temporal. EMU8086 emula una CPU Intel 8086, que es una de las primeras CPUs de la arquitectura x86, y, por lo tanto, presenta un conjunto de registros específicos. Algunos de los registros más importantes en EMU8086 incluyen:

- **AX (Registro Acumulador):** Se utiliza para operaciones aritméticas y es especialmente relevante para la manipulación de datos.
- **BX (Registro Base):** Puede usarse como un registro general para almacenar datos y direcciones de memoria.
- **CX (Registro de Contador):** Se utiliza en bucles y en operaciones que requieren contar o rastrear iteraciones.
- **DX (Registro de Datos Extendido):** Es comúnmente empleado para el almacenamiento temporal de datos.
- **SI (Registro de Índice Fuente):** Usado en operaciones de manejo de cadenas.
- **DI (Registro de Índice Destino):** También utilizado en operaciones de manejo de cadenas.
- **SP (Registro de Pila):** Almacena la dirección de la cima de la pila.
- **BP (Registro de Base de Pila):** Se utiliza para acceder a elementos en la pila.
- **IP (Registro de Puntero de Instrucción):** Contiene la dirección de la próxima instrucción a ejecutar.

- *FLAGS (Registros de Banderas)*: Almacena información sobre el estado de las operaciones, como banderas de cero, signo, acarreo, entre otras.

Comprender la función y el uso de estos registros es esencial para escribir programas eficientes y funcionales en lenguaje ensamblador en EMU8086.

- **Llamadas de Interrupciones en EMU8086:**

Las llamadas de interrupciones son una característica clave en EMU8086 y en la programación de bajo nivel en general. Estas llamadas permiten a un programa solicitar servicios o recursos del sistema operativo o interactuar con dispositivos de hardware. EMU8086 emula un conjunto de interrupciones que pueden ser utilizadas en programas ensambladores para realizar tareas específicas.

Algunas de las llamadas de interrupciones comunes en EMU8086 incluyen:

- *Interrupción 10h*: Utilizada para mostrar texto y gráficos en la pantalla.
- *Interrupción 21h*: Proporciona acceso a una variedad de servicios del sistema operativo, como lectura/escritura de archivos y entrada/salida estándar.
- *Interrupción 16h*: Maneja la entrada del teclado.
- *Interrupción 1Ah*: Permite acceder al reloj en tiempo real.

Entender cómo realizar llamadas de interrupción y trabajar con registros para pasar argumentos y recibir resultados es fundamental para programar aplicaciones que interactúen con el sistema y realicen tareas avanzadas.

En resumen, el conocimiento profundo de los registros y las llamadas de interrupciones en EMU8086 es esencial para aprovechar al máximo esta plataforma de emulación de microprocesador y para programar a nivel de máquina en la arquitectura x86. Los registros son las áreas de trabajo donde se realizan operaciones, y las llamadas de interrupciones proporcionan acceso a servicios del sistema y dispositivos externos. Dominar estos conceptos es esencial para escribir

programas funcionales y efectivos en lenguaje ensamblador en EMU8086 y comprender cómo interactuar con el hardware y el sistema operativo.

EN EL SIGUIENTE CÓDIGO PODRÁ VERIFICAR UN EJEMPLO SENCILLO DE LA APLICACIÓN DE LAS INTERRUPCIONES:

Se inicializa la interrupción 10H con la función 02H para colocar el cursor en la posición (10, 5) en la pantalla.

Se utiliza la función 09H de la interrupción 21H para mostrar el mensaje "Hola Mundo" en la posición especificada.

Finalmente, se utiliza la función 4CH de la interrupción 21H para salir del programa.

Este es un ejemplo sencillo y que en un entorno real, deberías incluir más manejo de errores y funcionalidades adicionales según sea necesario. Para ejecutar este código en EMU8086, puedes copiar y pegar el código en el editor de código de EMU8086 y luego compilar y ejecutar el programa.

The screenshot shows the EMU8086 assembly editor interface. On the left, the source code is displayed in a window titled "original source code". The code is as follows:

```
01 .MODEL SMALL
02 .DATA
03     MSG DB 'Hola'
04 .CODE
05     MAIN:
06         MOV AX, @DATA
07         MOV DS, AX
08
09         ; Iniciar cursor
10        MOV AH, 02H
11        MOV BH, 00H
12        MOV DH, 05H
13        MOV DL, 10H
14        INT 10H
15
16         ; Mostrar mensaje
17        MOV AH, 09H
18        LEA DX, MSG
19        INT 21H
20
21         ; Finalizar
22        MOV AH, 4CH
23        INT 21H
24
25 END MAIN
```

The assembly code is shown in the main window, with the instruction at address 0711:0008 highlighted: `MOV AX, 00710h`. A tooltip "coloc." is visible near this instruction. The registers window on the right shows the state of various registers like AX, BX, CX, DX, CS, IP, SS, SP, BP, SI, DI, DS, and ES. The CPU Registers window shows the current values of the registers. The status bar at the bottom indicates the line and column numbers (line: 26, col: 52), the date (14/10/2023), and the time (01:29 a.m.).

emu8086 - assembler and microprocessor emulator 4.08

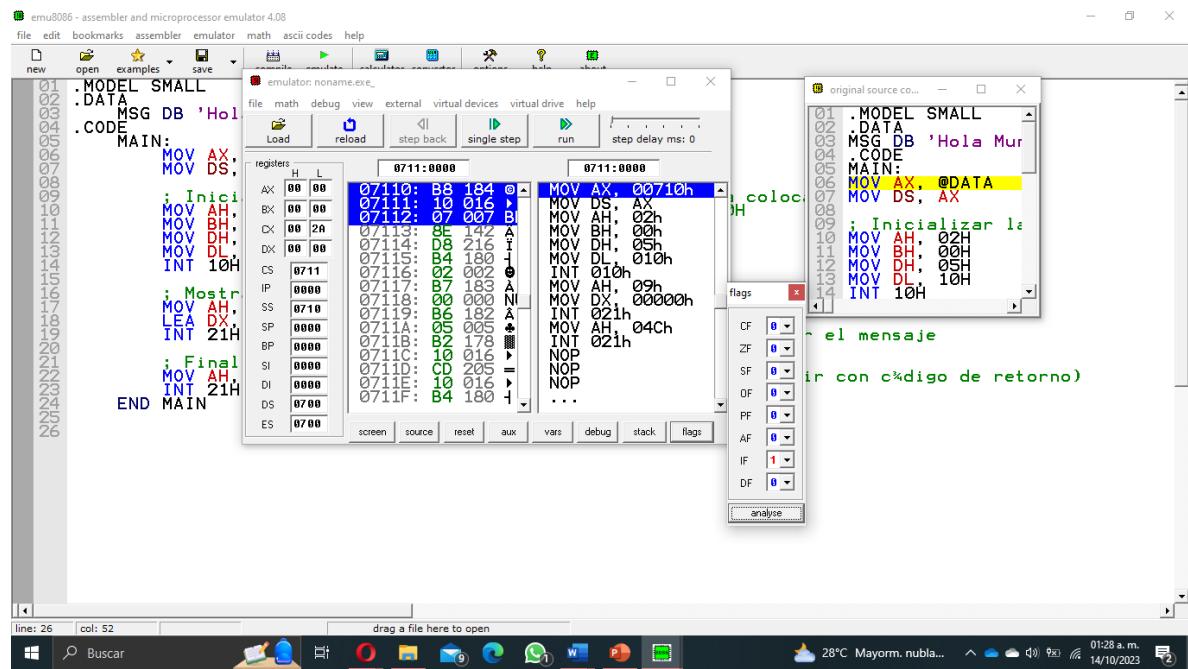
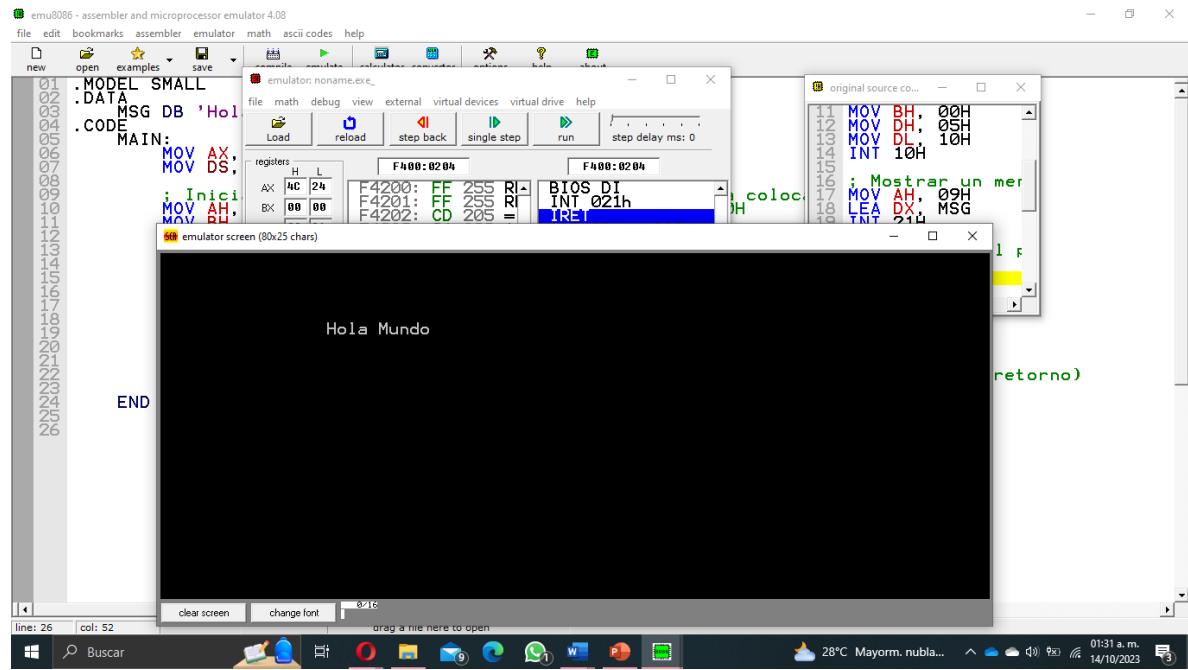
```
01 .MODEL SMALL
02 .DATA
03     MSG DB 'Hola'
04 .CODE
05 MAIN:
06     MOV AX, @DATA
07     MOV DS, AX
08 ; Iniciar la impresora
09     MOV AH, 02H
10    MOV BH, 00H
11    MOV DH, 05H
12    MOV DL, 10H
13    INT 21H
14
15 ; Mostrar el mensaje
16    MOV AH, 09H
17    LEA DX, MSG
18    INT 21H
19
20 ; Finalizar el programa
21    MOV AH, 04CH
22    INT 21H
23
24 END MAIN
```

The screenshot shows the assembly code for printing "Hola Mundo". The code initializes the data segment, moves the message string to the data segment, and then prints it using the BIOS interrupt 21H with function 9H. It ends with function 4CH to exit the program.

emu8086 - assembler and microprocessor emulator 4.08

```
01 .MODEL SMALL
02 .DATA
03     MSG DB 'Hola'
04 .CODE
05 MAIN:
06     MOV AX, @DATA
07     MOV DS, AX
08 ; Iniciar la impresora
09     MOV AH, 02H
10    MOV BH, 00H
11    MOV DH, 05H
12    MOV DL, 10H
13    INT 21H
14
15 ; Mostrar el mensaje
16    MOV AH, 09H
17    LEA DX, MSG
18    INT 21H
19
20 ; Finalizar el programa
21    MOV AH, 04CH
22    INT 21H
23
24 END MAIN
```

This screenshot is identical to the one above, showing the same assembly code for printing "Hola Mundo" using BIOS interrupt 21H.



CONCLUSIÓN SESIÓN 4

La comprensión de los registros y las llamadas de interrupciones en EMU8086 es esencial para la programación a nivel de máquina y para la interacción con el hardware y el sistema operativo. A lo largo de esta práctica, hemos explorado en detalle estos conceptos fundamentales y su importancia en el contexto de EMU8086. Esta conclusión brindará una visión completa de lo que hemos aprendido sobre los registros y las llamadas de interrupciones.

Importancia de los Registros en EMU8086:

Los registros son áreas de almacenamiento de datos en la CPU de EMU8086 que desempeñan un papel fundamental en la manipulación de datos y en la ejecución de operaciones. Cada registro tiene una función específica y se utiliza para tareas particulares. Por ejemplo, AX es el registro acumulador, utilizado en operaciones aritméticas, mientras que BX y CX se utilizan como registros base y contador, respectivamente, y tienen funciones distintas en operaciones y bucles.

Los registros son cruciales para el almacenamiento temporal de datos, el seguimiento de la ejecución del programa y la transferencia de información entre la memoria y la CPU. Comprender la función de cada registro y cómo se utilizan en conjunto es esencial para escribir programas eficientes y funcionales en lenguaje ensamblador en EMU8086.

Llamadas de Interrupciones en EMU8086:

Las llamadas de interrupciones son una característica poderosa en EMU8086 y en la programación a nivel de máquina en general. Estas llamadas permiten a los programas solicitar servicios y recursos del sistema operativo, interactuar con dispositivos de hardware y realizar tareas avanzadas. EMU8086 emula un conjunto de interrupciones que abren una amplia gama de posibilidades.

Las interrupciones se utilizan para comunicarse con el sistema operativo y solicitar servicios, como lectura/escritura de archivos, entrada/salida estándar o incluso manipulación de la pantalla. También permiten la interacción con dispositivos de hardware, como el teclado, y brindan acceso a funciones críticas del sistema.

Comprender cómo realizar llamadas de interrupción, qué registros utilizar para pasar argumentos y cómo recibir resultados es fundamental para programar aplicaciones efectivas en EMU8086.

Seguridad y Eficiencia:

Es importante destacar que el manejo de registros y llamadas de interrupciones debe realizarse con cuidado para garantizar la seguridad y la eficiencia del programa. Un mal uso de registros o llamadas de interrupción puede resultar en desbordamientos de registros o incluso bloqueos del sistema. Por lo tanto, es esencial seguir buenas prácticas de programación y garantizar que los registros se utilicen de manera adecuada y que las llamadas de interrupción se realicen de acuerdo con las especificaciones del sistema.

Aplicaciones Prácticas:

Finalmente, hemos discutido cómo estos conceptos se aplican en situaciones del mundo real. Desde la programación de aplicaciones que interactúan con el sistema operativo hasta el desarrollo de controladores de dispositivos y la manipulación de datos en tiempo real, los registros y las llamadas de interrupción en EMU8086 son herramientas esenciales para los programadores que buscan trabajar a nivel de máquina.

En resumen, la comprensión profunda de los registros y las llamadas de interrupciones en EMU8086 es esencial para aprovechar al máximo esta plataforma de emulación de microprocesador. Los registros son las áreas de trabajo donde se realizan operaciones y se almacenan datos, y las llamadas de interrupción brindan acceso a servicios del sistema y dispositivos externos. Dominar estos conceptos es esencial para programar aplicaciones funcionales y efectivas en lenguaje ensamblador en EMU8086 y comprender cómo interactuar con el hardware y el sistema operativo a nivel de máquina.



Sesión 5.

LENGUAJE

ENSAMBLADOR:

EMU8086

INTRODUCCIÓN SESIÓN 5

La instrucción "MOV" (movimiento) es una de las instrucciones fundamentales en el lenguaje ensamblador, y es esencial para la manipulación de datos en la arquitectura x86. En esta práctica, exploraremos en detalle la instrucción "MOV" dentro del entorno de EMU8086, que es una plataforma de emulación de microprocesador basada en la arquitectura x86. A lo largo de esta introducción extensa, examinaremos la importancia de la instrucción "MOV" y cómo se utiliza en conjunto con los registros en EMU8086.

La Instrucción "MOV" en el Lenguaje Ensamblador:

La instrucción "MOV" se utiliza para transferir datos entre ubicaciones de memoria y registros en la arquitectura x86. Esta instrucción es fundamental para copiar datos de un lugar a otro y se usa en prácticamente todos los programas escritos en lenguaje ensamblador. El formato general de la instrucción "MOV" es el siguiente:

MOV destino, origen

Donde "destino" y "origen" pueden ser registros, ubicaciones de memoria o constantes. La instrucción "MOV" permite una amplia flexibilidad en la transferencia de datos y es una herramienta esencial para cargar, almacenar y manipular información en programas ensambladores.

El Entorno de EMU8086:

EMU8086 es un entorno de emulación de microprocesador que emula una CPU Intel 8086, una de las primeras CPUs de la arquitectura x86. Proporciona una interfaz de desarrollo amigable que permite a los programadores escribir, ensamblar y depurar programas en lenguaje ensamblador para esta arquitectura. En el entorno de EMU8086, los registros y la memoria se emulan, lo que permite a los programadores experimentar y desarrollar aplicaciones sin la necesidad de hardware físico.

Registros en EMU8086:

Los registros son áreas de almacenamiento de datos dentro de la CPU de EMU8086 que se utilizan para realizar operaciones y retener información temporal. En el contexto de EMU8086, algunos de los registros más comunes incluyen:

- *AX (Registro Acumulador)*: Utilizado en operaciones aritméticas y para el almacenamiento temporal de datos.
- *BX (Registro Base)*: Empleado como registro general para almacenar datos y direcciones de memoria.
- *CX (Registro de Contador)*: Utilizado en bucles y operaciones que requieren contar o rastrear iteraciones.
- *: Almacena datos temporales y se utiliza en varias operaciones.*

Además de estos registros generales, EMU8086 emula otros registros importantes, como SP (Registro de Pila) y IP (Registro de Puntero de Instrucción), que son fundamentales para la gestión de la pila y el seguimiento de la ejecución del programa.

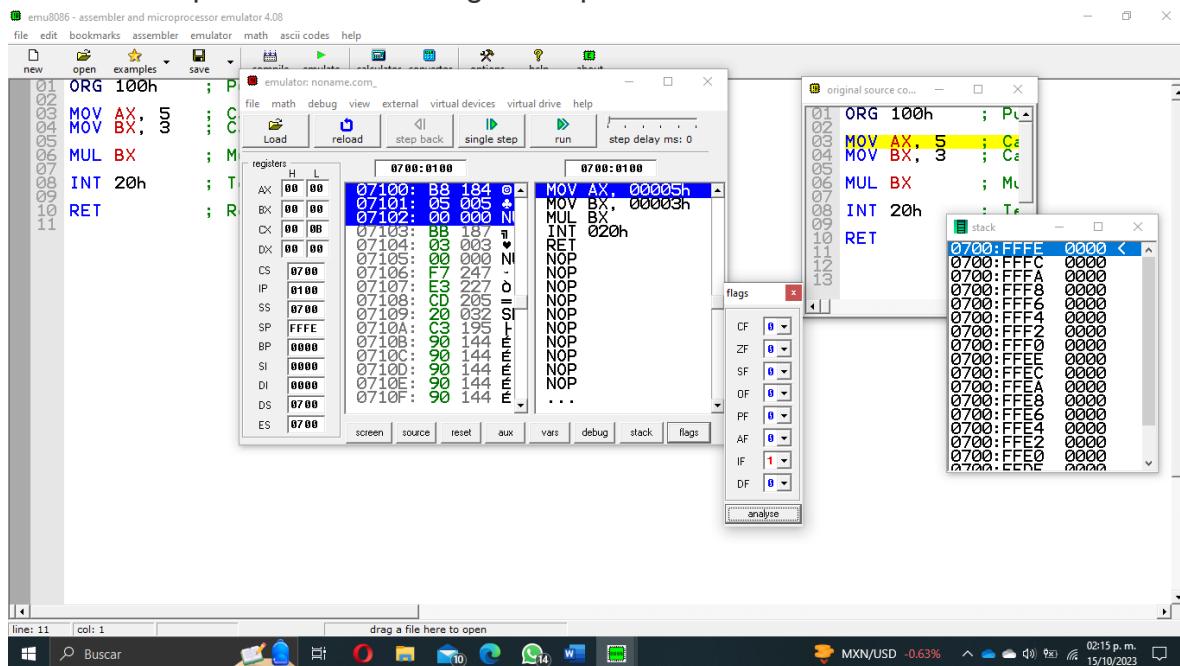
Importancia de la Instrucción "MOV" y los Registros en EMU8086:

La instrucción "MOV" y los registros son componentes esenciales en la programación en lenguaje ensamblador en EMU8086. La instrucción "MOV" permite la transferencia de datos, lo que es fundamental para la manipulación de información en programas. Los registros son las áreas de trabajo donde se almacenan y se realizan operaciones con datos, lo que permite realizar cálculos, tomar decisiones y gestionar la memoria. Comprender cómo usar "MOV" y los registros de manera efectiva es fundamental para escribir programas funcionales y eficientes en EMU8086.

En resumen, esta práctica explorará la instrucción "MOV" y su relevancia dentro del entorno de EMU8086, donde los registros desempeñan un papel crucial en la programación a nivel de máquina. Entender cómo se utiliza "MOV" para transferir datos entre registros y memoria es fundamental para programar aplicaciones en lenguaje ensamblador, y EMU8086 proporciona un entorno de aprendizaje ideal para adquirir estas habilidades esenciales.

REALICE UN PROGRAMA EN LENGUAJE ENSAMBLADOR QUE MULTIPLIQUE DOS NÚMEROS:

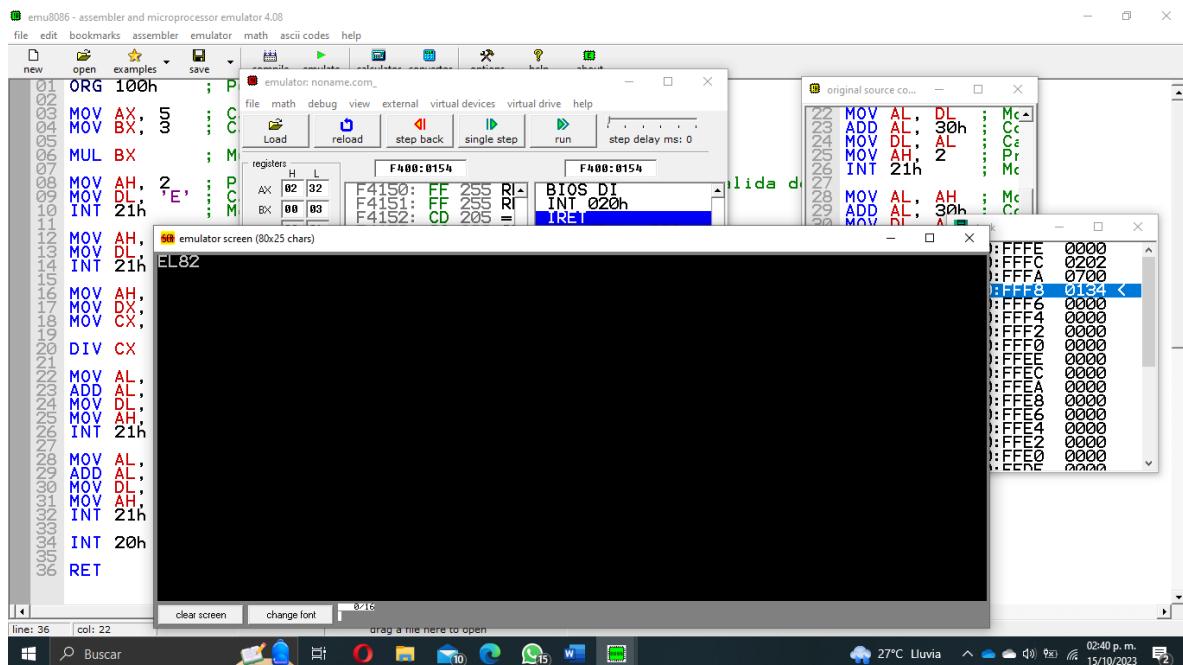
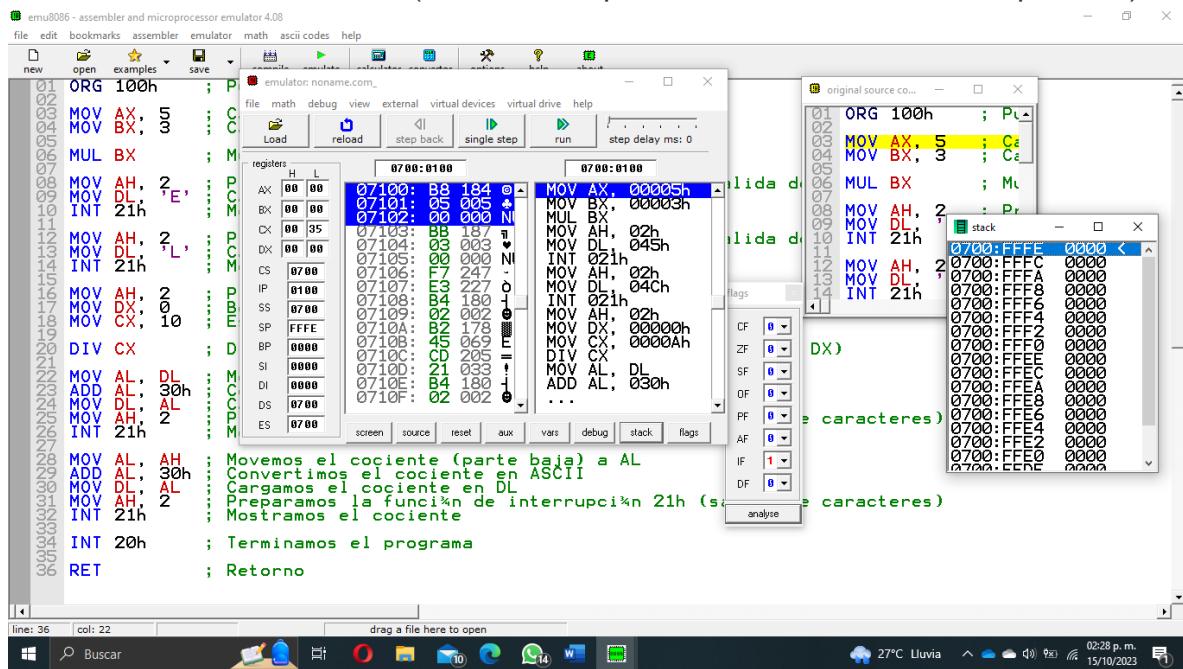
1. Con la instrucción MOV, básica, donde los resultados y la ubicación de datos la ubique dentro de los registros que muestra el entorno de EMU



Código:

```
ORG 100h ; Punto de inicio del programa  
MOV AX, 5 ; Cargamos el primer número (5) en AX  
MOV BX, 3 ; Cargamos el segundo número (3) en BX  
MUL BX ; Multiplicamos AX por BX (resultado en DX:AX)  
INT 20h ; Terminamos el programa  
RET ; Retorno
```

2. Realice la misma multiplicación utilizando registros y mostrando en pantalla el resultado obtenido (utilice interrupciones de salida de datos a pantalla).



Código:

ORG 100h ; Punto de inicio del programa

MOV AX, 5 ; Cargamos el primer número (5) en AX

MOV BX, 3 ; Cargamos el segundo número (3) en BX

MUL BX ; Multiplicamos AX por BX (resultado en DX:AX)

MOV AH, 2 ; Preparamos la función de interrupción 21h (salida de caracteres)

MOV DL, 'E' ; Caracter 'E' para imprimir

INT 21h ; Mostramos 'E'

MOV AH, 2 ; Preparamos la función de interrupción 21h (salida de caracteres)

MOV DL, 'L' ; Caracter 'L' para imprimir

INT 21h ; Mostramos 'L'

MOV AH, 2 ; Preparamos la función de interrupción 21h (salida de caracteres)

MOV DX, 0 ; Borramos DX para la parte alta del resultado

MOV CX, 10 ; Establecemos un divisor (10)

DIV CX ; Dividimos DX:AX por CX (resultado en AX, residuo en DX)

MOV AL, DL ; Movemos el residuo (parte baja) a AL

ADD AL, 30h ; Convertimos el residuo en ASCII

MOV DL, AL ; Cargamos el residuo en DL

MOV AH, 2 ; Preparamos la función de interrupción 21h (salida de caracteres)

INT 21h ; Mostramos el residuo

MOV AL, AH ; Movemos el cociente (parte baja) a AL

ADD AL, 30h ; Convertimos el cociente en ASCII

MOV DL, AL ; Cargamos el cociente en DL

MOV AH, 2 ; Preparamos la función de interrupción 21h (salida de caracteres)

INT 21h ; Mostramos el cociente

INT 20h ; Terminamos el programa

RET ; Retorno

CONCLUSIÓN SESIÓN 5

En resumen, un programa en lenguaje ensamblador que multiplica dos números utilizando solo la instrucción MOV, almacenando los datos en registros del entorno de EMU y luego muestra el resultado en pantalla utilizando interrupciones de salida, es un ejercicio eficiente en términos de código y rendimiento. Sin embargo, es importante tener en cuenta que la multiplicación se logra a través de una serie de desplazamientos y sumas, lo que implica un mayor esfuerzo de programación y puede no ser la forma más eficiente de realizar esta operación en un entorno de lenguaje ensamblador. La utilización de instrucciones de multiplicación específicas disponibles en la mayoría de las arquitecturas de CPU sería más eficiente en términos de tiempo de ejecución y simplicidad del código.

CONCLUSIÓN GENERAL

La práctica de EMU8086, que implica la introducción al lenguaje ensamblador 8086, operaciones aritméticas básicas, el manejo de la pila y las interrupciones, proporciona una comprensión fundamental de la programación de bajo nivel y de la arquitectura de la CPU x86. En resumen, estas son algunas conclusiones clave:

1. Introducción al lenguaje ensamblador 8086:

- El lenguaje ensamblador 8086 es un conjunto de instrucciones de bajo nivel que permite la programación directa de la CPU x86.
- Requiere un conocimiento profundo de la arquitectura de la CPU, ya que las instrucciones se relacionan directamente con las operaciones de la máquina.

2. Operaciones aritméticas básicas:

- Las operaciones aritméticas, como la suma, resta, multiplicación y división, se realizan a nivel de registros y memoria utilizando instrucciones específicas.
- Es necesario entender los registros de la CPU y cómo almacenar y manipular datos.

3. Manejo de la pila:

- La pila es una estructura de datos esencial en la programación ensambladora que se utiliza para almacenar datos temporales y gestionar la ejecución de programas.
- El uso de instrucciones como PUSH (para agregar datos a la pila) y POP (para retirar datos de la pila) es crucial para controlar el flujo del programa y gestionar las llamadas a subrutinas.

4. Interrupciones:

- Las interrupciones son eventos externos o internos que pueden pausar la ejecución normal del programa y transferir el control a una rutina de servicio de interrupción.
- En el contexto de la programación 8086, las interrupciones se utilizan para manejar eventos de hardware, como el teclado o el temporizador.

En general, la práctica de EMU8086 y la programación en lenguaje ensamblador 8086 son fundamentales para desarrolladores que buscan comprender en profundidad la ejecución de programas a nivel de hardware. Aunque este enfoque es menos utilizado en la programación moderna, sigue siendo relevante en áreas críticas, como el desarrollo de controladores de dispositivos, sistemas operativos y aplicaciones de bajo nivel.

Bibliografía

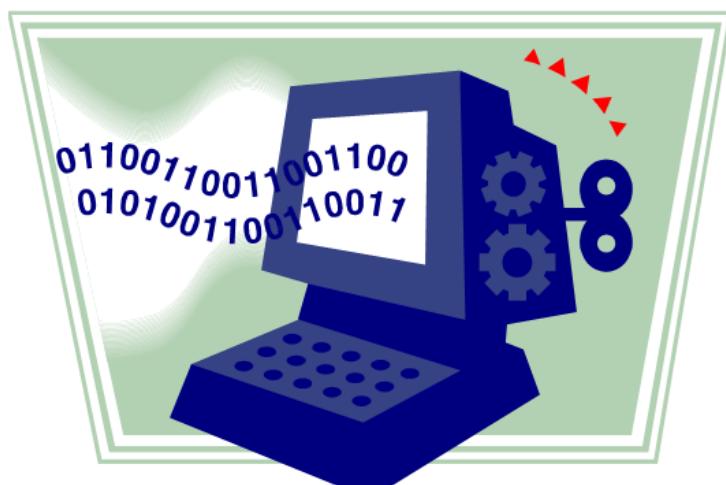
- EL CONCEPTO DE INTERRUPCIONES.* (15 de 10 de 2023). Obtenido de <https://leoyac.wixsite.com/lenguaje-ensamblador/el-concepto-de-interrupciones#:~:text=Una%20interrupci%20n%20es%20una%20situaci%20n,una%20operaci%20n%20de%20E%2FS>.
- EMU8086.* (15 de 10 de 2023). Obtenido de <https://emu8086-microprocessor-emulator.softonic.com>
- Ensamblador.* (15 de 10 de 2023). Obtenido de <https://www.unioviedo.es/ate/alberto/TEMA3-Ensamblador.pdf>
- Kann, C. W. (15 de 10 de 2023). *¿Qué es el lenguaje ensamblador?* Obtenido de [https://espanol.libretexts.org/Ingenieria/Implementaci%20n_de_una_CPU_de_una_direcci%20n_en_Logisim_\(Kann\)/02%3A_Lenguaje_de_la_Asamblea/2.01%3A_%C3%A9_Qu%C3%A9_es_el_lenguaje_ensamblador%3F](https://espanol.libretexts.org/Ingenieria/Implementaci%20n_de_una_CPU_de_una_direcci%20n_en_Logisim_(Kann)/02%3A_Lenguaje_de_la_Asamblea/2.01%3A_%C3%A9_Qu%C3%A9_es_el_lenguaje_ensamblador%3F)
- Lenguaje Ensamblador.* (15 de 10 de 2023). Obtenido de <https://www.mat.uson.mx/Icota/Lenguaje%20ensamblador.htm>
- Martínez, M. B. (15 de 10 de 2023). *http://bbeltran.cs.buap.mx/Interrupciones.pdf.* Obtenido de <http://bbeltran.cs.buap.mx/Interrupciones.pdf>
- Operaciones Aritméticas en Assembler con EMU8086.* (15 de 10 de 2023). Obtenido de <https://codemyn.blogspot.com/2017/09/operaciones-aritmeticas-en-assembler.html>



TECNOLÓGICO
NACIONAL DE MÉXICO



TEMA 3: MODULARIZACIÓN- INTEGRACIÓN



Para la actividad integral deberá realizar lo siguiente:

1. Implementación de programa en lenguaje de alto nivel.

Realiza un programa que realice la operación multiplicación y su algoritmo (en el punto 3 se define la estructura)

2. Implementación de un algoritmo en Lenguaje de bajo nivel (ensamblador 8086)

Realiza un programa que realice la operación multiplicación y su algoritmo (en el punto 3 se define la estructura)

3. Contraste los correspondientes Ítems de los dos tipos de lenguaje de programación, Integre la información en el formato siguiente:

ÍTEMS	LENGUAJE DE ALTO NIVEL (LAN)	LENGUAJE DE BAJO NIVEL (LBN ENSAMBLADOR 8086)
<i>1. Algoritmo</i>	<ol style="list-style-type: none">1. Inicio del programa.2. Crear un objeto Scanner llamado "scanner" para leer la entrada del usuario.3. Solicitar al usuario que ingrese el primer número.<ul style="list-style-type: none">- Mostrar el mensaje "Ingrese el primer número:".- Leer el número ingresado por el usuario y almacenarlo en la variable "numero1".4. Solicitar al usuario que ingrese el segundo número.<ul style="list-style-type: none">- Mostrar el mensaje "Ingrese el segundo número:".- Leer el número ingresado por el usuario y almacenarlo en la variable "numero2".5. Cerrar el scanner después de su uso.6. Realizar la multiplicación de los dos números.<ul style="list-style-type: none">- Calcular el producto de "numero1" y "numero2" y	<ol style="list-style-type: none">1. El programa comienza con la etiqueta "begin", que inicia el procedimiento "begin proc".2. El programa configura el segmento de datos y carga las direcciones de inicio de datos en los registros AX y DS.3. Se muestra un mensaje "Introduce el primer valor" utilizando la interrupción 21h con la función 09h, que imprime una cadena en la pantalla.4. El programa utiliza la interrupción 21h con la función 01h para leer un carácter (el primer número) del teclado.5. Se convierte el carácter leído en el paso anterior en un número restando 30h, que es el código ASCII del carácter '0'.6. Se muestra un mensaje "Introduce el segundo valor".7. El programa utiliza la interrupción 21h con la función

	<p>almacenarlo en la variable "resultado".</p> <p>7. Mostrar el resultado de la multiplicación.</p> <ul style="list-style-type: none"> - Mostrar el mensaje "El resultado de la multiplicación es:" seguido del valor de "resultado". <p>8. Fin del programa.</p>	<p>01h para leer otro carácter (el segundo número) del teclado.</p> <p>8. Al igual que antes, se convierte el carácter leído en un número restando 30h.</p> <p>9. Se realiza la operación de multiplicación de los dos números y el resultado se almacena en la variable "product".</p> <p>10. El resultado se convierte nuevamente en un carácter sumando 30h.</p> <p>11. Se muestra un mensaje "Producto=" en la pantalla.</p> <p>12. El programa utiliza la interrupción 21h con la función 02h para mostrar el valor de "product" en la pantalla.</p> <p>13. El programa termina con una llamada a la interrupción 21h con la función 4Ch, que sale del programa.</p>
<p>2. Caso 1.</p> <p><i>Código básico utilizando la instrucción MUL (Uso de datos internos).</i></p>	<pre>package multiplicacionBasic; import java.util.Scanner; public class MultiplicacionBasic { public static void main(String[] args) { // Crear un objeto Scanner para leer la entrada del usuario Scanner scanner = new Scanner(System.in); // Solicitar al usuario que ingrese el primer número System.out.print("Ingrese el primer número: "); int numero1 = scanner.nextInt(); // Ingresar el carácter de retorno de carro scanner.nextLine(); // Solicitar al usuario que ingrese el segundo número System.out.print("Ingrese el segundo número: "); int numero2 = scanner.nextInt(); // Cerrar el scanner después de su uso scanner.close(); // Realizar la multiplicación de los dos números int resultado = numero1 * numero2; // Mostrar el resultado System.out.println("El resultado de la multiplicación es: " + resultado); } }</pre> <p>Output: MultiplicacionBasic.java -</p> <pre> sum: Ingrese el primer número: 3 Ingrese el segundo número: 5 El resultado de la multiplicación es: 15 BUILD SUCCESSFUL (total time: 10 seconds) </pre>	<pre>.model small .stack 64 .data n1 db 0 n2 db 0 product db 0 msg1 db 10,13, "Introduce el primer valor: '\$' msg2 db 10,13, "Introduce el segundo valor: '\$" msg3 db 10,13, "Producto= '\$" .code begin proc mov ax,@data mov ds,ax ;Solicitamos el primer digito mov ah,09 lea dx,msg1 int 21h ;Se lee el primer dato mov ah,01 int 21h sub al,30h ;Convierte el carácter a numero mov n1,al ;Solicitamos el segundo digito mov ah,09 lea dx,msg2 int 21h ;Se lee el segundo dato mov ah,01 int 21h sub al,30h ;Convierte el carácter a numero mov n2,al ;Operación de multiplicación mov ah,n1 mul ah add al,30h mov product,al mov ah,09 lea dx,msg3 int 21h ;Se imprime el carácter con el servicio 02 de la int 21 mov ah,02 mov dl,product int 21h mov ah,4ch int 21h begin endp end</pre>

3. Caso
Código sin utilizar la instrucción MUL en LBN (Uso de datos internos).

```
package multiplicacion.sin.multiplicacion;
import java.util.Scanner;
public class MultiplicacionScanner {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Ingrese el primer numero: ");
        int multiplicando = scanner.nextInt();
        System.out.print("Ingrese el segundo numero: ");
        int multiplicador = scanner.nextInt();
        scanner.close();
        // Realizamos un bucle for para sumar el multiplicando multiplicador veces
        for (int i = 0; i < multiplicando; i++) {
            resultado += multiplicador;
        }
        System.out.println("El resultado de la multiplicación es: " + resultado);
    }
}
```

Output: Multiplicacion sin MUL (res) :

LUNI
Ingres el primer numero: 3
Ingres el segundo numero: 5
El resultado de la multiplicacion es: 15
BUILD SUCCESSFUL (total time: 12 seconds)

4. Caso
Código con entrada y salida sin utilizar la inscripción MUL utilizando para ambos lenguajes datos de entrada-salida desde el teclado pantalla.

```
package multiplicacion;
import java.util.Scanner;
public class Multiplicacion {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        // Entrada de datos: Ingresó del multiplicando y multiplicador desde el teclado
        String num1 = scanner.nextLine();
        String num2 = scanner.nextLine();
        int multiplicando = Integer.parseInt(num1);
        int multiplicador = Integer.parseInt(num2);
        // Multiplicamos el resultado en cero
        int resultado = 0;
        // Realizamos un bucle while para sumar el multiplicando tantas veces como el multiplicador
        for (int i = 0; i < multiplicando; i++) {
            resultado += multiplicador;
        }
        // Salida de datos: Mostrar el resultado en la pantalla
        System.out.println("El resultado de la multiplicación es: " + resultado);
    }
}
```

Output: Multiplicacion (res) :

LUNI
Ingres el primer numero (multiplicando): 6
Ingres el segundo numero (multiplicador): 9
El resultado de la multiplicación es: 54
BUILD SUCCESSFUL (total time: 6 seconds)

5. ¿Cuántas y cuales instrucciones se ocuparán en cada caso?

Caso 1:
Scanner del paquete java.util
MultiplicacionScanner
Main dentro de la clase MultiplicacionScanner
Scanner llamad scanner
System.out.print
Numero1
System.out.print

```
.model small
.stack 64
.data
n1 db 0
n2 db 0
suma db 0
msg1 db "Introduce el primer valor: '$'
msg2 db 10,13, "Introduce el segundo valor: '$'
msg3 db 10,13, "Suma: '$"
.code
begin proc
    mov ax, @data
    mov ds, ax
    ; Solicita el primer numero
    mov ah, 09
    lea dx, msg1
    int 21h
    ; Lee el primer dato
    mov ah, 01
    int 21h
    sub al, 30h ; Convierte el caracter a numero
    mov n1, al
    ; Solicita el segundo numero
    mov ah, 09
    lea dx, msg2
    int 21h
    ; Lee el segundo dato
    mov ah, 01
    int 21h
    sub al, 30h ; Convierte el caracter a numero
    mov n2, al
    ; Bucle para realizar la multiplicacion mediante suma sucesiva
    dec cl
    suma_loop:
        add al, n1
        loop suma_loop
        ; Convierte el numero en caracter
        add al, 30h
        mov n1, al
        mov ah, 09
        lea dx, msg3
        int 21h
        mov ah, 02
        mov dx, suma
        mov ah, 4ch
        int 21h
endp
```

```
.model small
.stack 100h
.data
n1 db 0
n2 db 0
sum db 0
msg1 db "Introduce el primer numero: '$"
msg2 db "Introduce el segundo numero: '$"
msg3 db 10, 13, "La suma es: '$"
.code
    mov ax, @data
    mov ds, ax
    ; Solicitar el primer numero al usuario
    mov ah, 09
    lea dx, msg1
    int 21h
    ; Leer el primer numero desde el teclado
    mov ah, 01
    int 21h
    sub al, 30h
    mov n1, al
    ; Solicitar el segundo numero al usuario
    mov ah, 09
    lea dx, msg2
    int 21h
    ; Leer el segundo numero desde el teclado
    mov ah, 01
    int 21h
    sub al, 30h
    mov n2, al
    ; Sumar los numeros sucesivamente
    mov al, n1
    add al, n2
    mov sum, al
    ; Mostrar el resultado en pantalla
    mov ah, 09
    lea dx, msg3
    int 21h
    mov ah, 02
    mov dx, sum
    add dx, 30h ; Convertir el resultado a caracter antes de mostrarlo
    int 21h
    ; Terminar el programa
    mov ah, 4Ch
    int 21h
endp
```

Caso 1:
MOV
LEA
INT 21h
ADD
SUB
MUL
INT 4Ch

	<i>Numero2</i> <i>Close</i> <i>Resultado</i> <i>System.out.println</i>	
	Caso 2: <i>Scanner del paquete</i> <i>java.util</i> <i>MultiplicacionSinMul</i> <i>Main dentro de la clase</i> <i>MultiplicacionSinMul</i> <i>Scanner llamado scanner</i> <i>System.out.print</i> <i>Multiplicando</i> <i>System.out.print</i> <i>Multiplicador</i> <i>Close</i> <i>Resultado</i> <i>For para hacer la multiplicación sin utilizar el operador *</i> <i>System.out.println</i>	Caso 2: <i>MOV</i> <i>LEA</i> <i>INT 21h</i> <i>ADD</i> <i>SUB</i> <i>DEC</i> <i>LOOP</i> <i>MOVZX</i>
	Caso 3: <i>Scanner del paquete</i> <i>java.util</i> <i>MultiplicacionSinMul</i> <i>Main dentro de la clase</i> <i>MultiplicacionSinMul</i> <i>Scanner llamado scanner</i> <i>System.out.print</i> <i>Multiplicando</i> <i>System.out.print</i> <i>Multiplicador</i> <i>Close</i> <i>Resultado</i> <i>Contador</i> <i>While para realizar la multiplicación sin utilizar el operador *</i> <i>System.out.println</i>	Caso 3: <i>MOV</i> <i>LEA</i> <i>INT 21h</i> <i>ADD</i> <i>SUB</i> <i>MOVZX</i>
6. ¿Cómo difiere el rendimiento, funcionalidad y lógica en cada caso?	Caso 1: El rendimiento de este programa es adecuado para su propósito, ya que se trata de una aplicación simple que realiza cálculos básicos. No hay operaciones	Caso 1: Este código funciona de manera muy eficiente y rápida, ya que realiza una simple multiplicación de dos números de un solo dígito y no involucra operaciones intensivas.

	<p>intensivas ni procesamiento complejo involucrado, por lo que el rendimiento no es una preocupación en este caso.</p> <p>La funcionalidad del programa es limitada. Su único propósito es permitir al usuario ingresar dos números, multiplicarlos y mostrar el resultado. No ofrece características adicionales, como manejo de errores en caso de entrada no válida o la capacidad de realizar otras operaciones matemáticas.</p>	<p>El código hace uso de interrupciones de MS-DOS (int 21h) para realizar operaciones de entrada/salida.</p>
	<p>Caso 2:</p> <p>El rendimiento de este programa es inferior al primer ejemplo. La razón es que utiliza un bucle for para sumar el multiplicando multiplicador veces. Esto significa que, en el peor caso, el bucle realizará un número de iteraciones igual al valor de multiplicador. Para multiplicaciones de números pequeños, esto no será un problema, pero para números grandes, este enfoque será mucho más lento en comparación con el operador de multiplicación.</p> <p>La funcionalidad general del programa es la misma que el primer ejemplo. Permite al usuario ingresar dos números, realiza la multiplicación y muestra el resultado. Sin embargo, la implementación difiere en la forma en que se realiza la multiplicación.</p>	<p>Caso 2:</p> <p>Este código requiere más tiempo para calcular la multiplicación utilizando una suma sucesiva en lugar de una multiplicación directa. El rendimiento será más lento, especialmente para números más grandes.</p> <p>Utiliza un bucle para realizar la suma sucesiva, lo que implica un mayor número de instrucciones y ciclos de CPU en comparación con una multiplicación directa.</p>
	<p>Caso 3:</p> <p>El rendimiento de este programa es similar al segundo ejemplo que utilizó un bucle for para realizar la multiplicación. En este caso, el rendimiento también es menos eficiente en comparación con el uso del operador de multiplicación.</p> <p>El bucle while realiza la</p>	<p>Caso 3:</p> <p>Este código es bastante eficiente, ya que realiza una simple suma de dos números de un solo dígito. La operación de suma es rápida y no involucra operaciones intensivas.</p> <p>Al igual que los códigos anteriores, este código hace uso de interrupciones de MS-DOS (int 21h)</p>

		<p>multiplicación manualmente al sumar el valor del multiplicando a resultado en cada iteración. Para multiplicaciones de números grandes, este enfoque será más lento.</p> <p>La funcionalidad general del programa es la misma que los ejemplos anteriores. Permite al usuario ingresar dos números, realiza la multiplicación y muestra el resultado.</p>	<p>para realizar operaciones de entrada/salida. Esto puede ser más lento en comparación con métodos de entrada/salida más modernos utilizados en lenguajes de programación de alto nivel.</p>
7. ¿Cómo se rastrean los errores al ejecutar (emular) los programas?	se los al	<p>En Java, los errores se pueden rastrear y depurar mediante varias herramientas y técnicas.</p> <p>Mensajes de error en la consola</p> <p>Uso de declaraciones de impresiones (print statements)</p> <p>Depuradores</p> <p>Control de excepciones</p> <p>Registros de errores (logging)</p> <p>Pruebas unitarias</p> <p>Análisis estático de código</p> <p>Comentarios y documentación</p>	<p>Si hay errores en el programa, EMU8086 mostrará mensajes de error en la ventana de salida. Estos mensajes pueden incluir información sobre errores de ensamblado, como instrucciones inválidas, etiquetas no definidas, registros no inicializados, entre otros.</p>
8. Indica que operaciones o instrucciones de llamadas o procedimientos se aplicaron en los casos que desarrollaste.	Caso 1: MultiplicacionScanner	<ol style="list-style-type: none"> 1. Creación de un objeto Scanner para leer la entrada del usuario. 2. Solicitar al usuario que ingrese dos números (multiplicando y multiplicador) usando System.out.print. 3. Leer los números ingresados por el usuario con scanner.nextInt(). 4. Cerrar el objeto Scanner después de su uso con scanner.close(). 5. Realizar la multiplicación de los dos números con el operador de multiplicación (*). 	<p>Primer código (Multiplicación):</p> <ol style="list-style-type: none"> 1. mov ax, @data y mov ds, ax: Estas instrucciones configuran el segmento de datos. 2. mov ah, 09 y lea dx, msg1: Estas instrucciones muestran un mensaje al usuario. 3. mov ah, 01 e int 21h: Estas instrucciones leen un carácter del teclado del usuario. 4. sub al, 30h: Resta 30h al valor leído para convertirlo de un carácter numérico a un número entero. 5. mul n2: Realiza la multiplicación de los números. 6. add al, 30h y mov product, al: Convierte el resultado de la multiplicación en un carácter y lo almacena en la variable "product."

	<p>6. Mostrar el resultado de la multiplicación en la consola usando System.out.println.</p> <p>Caso 2: MultiplicacionSinMul (con bucle for)</p> <ol style="list-style-type: none"> 1. Creación de un objeto Scanner para leer la entrada del usuario. 2. Solicitar al usuario que ingrese dos números (multiplicando y multiplicador) usando System.out.print. 3. Leer los números ingresados por el usuario con scanner.nextInt(). 4. Cerrar el objeto Scanner después de su uso con scanner.close(). 5. Inicialización de una variable resultado en 0. 6. Utilización de un bucle for para sumar el multiplicando multiplicador veces y así realizar la multiplicación manualmente. 7. Mostrar el resultado de la multiplicación en la consola usando System.out.println. <p>Caso 3: MultiplicacionSinMul (con bucle while)</p> <ol style="list-style-type: none"> 1. Creación de un objeto Scanner para leer la entrada del usuario. 2. Solicitar al usuario que ingrese dos números (multiplicando y multiplicador) usando System.out.print. 3. Leer los números ingresados por el usuario con scanner.nextInt(). 	<p>7. mov ah, 02, mov dl, product, e int 21h: Estas instrucciones muestran el resultado al usuario.</p> <p>8. mov ah, 4Ch e int 21h: Terminan el programa y lo cierran.</p> <p>Segundo código (Suma):</p> <ol style="list-style-type: none"> 1. mov ax, @data y mov ds, ax: Configuran el segmento de datos. 2. Solicita y lee dos números del usuario, al igual que en el primer código. 3. mov al, n1 y un bucle con add al, n1: Realiza la suma sucesiva de los números. 4. add al, 30h y mov suma, al: Convierte el resultado de la suma en un carácter y lo almacena en la variable "suma." 5. Muestra el resultado al usuario y termina el programa, al igual que en el primer código. <p>Tercer código (Otra Suma):</p> <ol style="list-style-type: none"> 1. mov ax, @data y mov ds, ax: Configuran el segmento de datos. 2. Solicita y lee dos números del usuario, al igual que en los otros códigos. 3. mov al, n1 y add al, n2: Realiza la suma de los números. 4. add al, 30h y mov sum, al: Convierte el resultado de la suma en un carácter y lo almacena en la variable "sum." 5. Muestra el resultado al usuario y termina el programa, al igual que en los otros códigos.
--	--	---

	<p>4. Cerrar el objeto Scanner después de su uso con scanner.close().</p> <p>5. Inicialización de una variable resultado en 0.</p> <p>6. Utilización de un bucle while para sumar el multiplicando multiplicador veces y así realizar la multiplicación manualmente. Se utiliza una variable contador para controlar el número de iteraciones.</p> <p>7. Mostrar el resultado de la multiplicación en la consola usando System.out.println.</p>	
9. Analiza, reflexiona y anota las diferentes formas de realizarlo una vez presenten los hallazgos, ventajas y desventajas de los dos tipos de lenguajes de programación (este punto lo realizaras en el aula).	<p>La principal diferencia entre los tres códigos radica en cómo se lleva a cabo la multiplicación. El primer programa usa el operador de multiplicación, que es la forma más directa y eficiente de realizar la multiplicación. Los otros dos programas, en lugar de usar el operador de multiplicación, utilizan bucles (for o while) para realizar la multiplicación manualmente, lo cual es menos eficiente en términos de rendimiento, pero demuestra conceptos de programación y control de flujo. Todos los programas solicitan la entrada del usuario, calculan el resultado y lo muestran en la consola, por lo que comparten una funcionalidad básica común.</p>	<p>Las diferencias clave entre estos códigos radican en la operación que realizan (multiplicación o suma) y en la forma en que se implementa. El primer código realiza la multiplicación mediante una instrucción de multiplicación (mul), mientras que los códigos 2 y 3 realizan la suma, pero el código 2 usa un bucle loop para hacerlo de manera iterativa, mientras que el código 3 lo hace de manera directa sin un bucle.</p> <p>En general, estos programas demuestran diferentes formas de realizar operaciones matemáticas en lenguaje ensamblador y cómo interactuar con el usuario para obtener datos de entrada y mostrar resultados en la consola.</p>

LENGUAJE DE INTERFAZ

INSTITUTO TECNOLÓGICO DE MINATITLÁN

CAMPUS MINATITLÁN

Clave: ISC-2240

INGENIERÍA EN SISTEMAS COMPUTACIONALES

DOCENTE: MARIA CONCEPCIÓN VILLATORO CRUZ

PROYECTO FINAL: PROGRAMACIÓN CON ARDUINO

INTEGRANTES:

- CARVAJAS HERNANDEZ JOHAN
- CRUZ MISS YESSICA YAMILET
- DIAZ ROSAS CLAUDIA BERENICE
- GONZALEZ ROMERO WYGALMI



Contenido

LENGUAJE DE INTERFAZ.....	133
LENGUAJE DE INTERFAZ.....	133
INTRODUCCIÓN.....	135
CONTEXTO DEL PROYECTO	135
OBJETIVO DEL PROYECTO	136
FUNDAMENTOS DE ARDUINO.....	138
CONCEPTOS BÁSICOS	138
ESTADO DEL ARTE.....	139
DESCRIPCIÓN DEL PROBLEMA.....	142
JUSTIFICACIÓN	142
DISEÑO EN TINKERCARD	144
ARQUITECTURA DEL SISTEMA.....	146
SELECCIÓN DE COMPONENTES	146
DIAGRAMA DE CONEXIONES.....	148
EXPLICACIÓN DEL CÓDIGO	149
ALGORITMOS Y FUNCIONES CLAVE	150
CASOS DE PRUEBAS	151
RESULTADOS DE PRUEBAS.....	152
GUÍA DE USUARIO.....	153
RESULTADOS Y CONCLUSIONES	153
TRABAJO FUTURO – MEJORAS Y EXPANSIONES	154
REFERENCIAS	157
ANEXOS	157

INTRODUCCIÓN

Un proyecto de programación con Arduino es una emocionante aventura que combina hardware y software para crear dispositivos interactivos, robots, sistemas de control, y mucho más. Arduino es una plataforma de prototipado electrónico que utiliza un microcontrolador programable y un entorno de desarrollo que simplifica la escritura de código. Estos proyectos pueden abarcar desde simples luces intermitentes hasta complejas estaciones meteorológicas automatizadas o incluso dispositivos de IoT (Internet de las cosas). La versatilidad de Arduino permite a los entusiastas de todas las edades y niveles de habilidad explorar el mundo de la electrónica y la programación de una manera práctica y creativa.

CONTEXTO DEL PROYECTO

Un propósito significativo para un proyecto de sensor de nivel de agua es monitorear y controlar el suministro de agua en entornos donde el acceso al agua potable es limitado o crucial, como en comunidades rurales, proyectos de agricultura, sistemas de riego o incluso en situaciones de gestión de desastres naturales. La importancia de este proyecto radica en varios aspectos:

1. Gestión eficiente del agua: Un sensor de nivel de agua puede ayudar a optimizar el uso del agua, evitando el desperdicio y asegurando un suministro adecuado para las necesidades específicas, como el riego de cultivos o el consumo humano.
2. Prevención de inundaciones: En áreas propensas a inundaciones, los sensores de nivel de agua pueden alertar sobre niveles peligrosamente altos, permitiendo tomar medidas preventivas para proteger vidas y propiedades.
3. Acceso al agua potable: En regiones donde el acceso al agua potable es limitado, estos sensores pueden ser parte de sistemas que controlan y gestionan el suministro de agua potable.

4. Conservación del medio ambiente: Monitorear el nivel del agua en ríos, lagos y embalses puede ayudar en la gestión ambiental, controlando el flujo y la calidad del agua para proteger los ecosistemas acuáticos.
5. Automatización en la agricultura: Para sistemas de riego automatizados, el uso de sensores de nivel de agua puede garantizar que las plantas reciban la cantidad adecuada de agua en el momento preciso, mejorando así la eficiencia y los rendimientos de los cultivos.

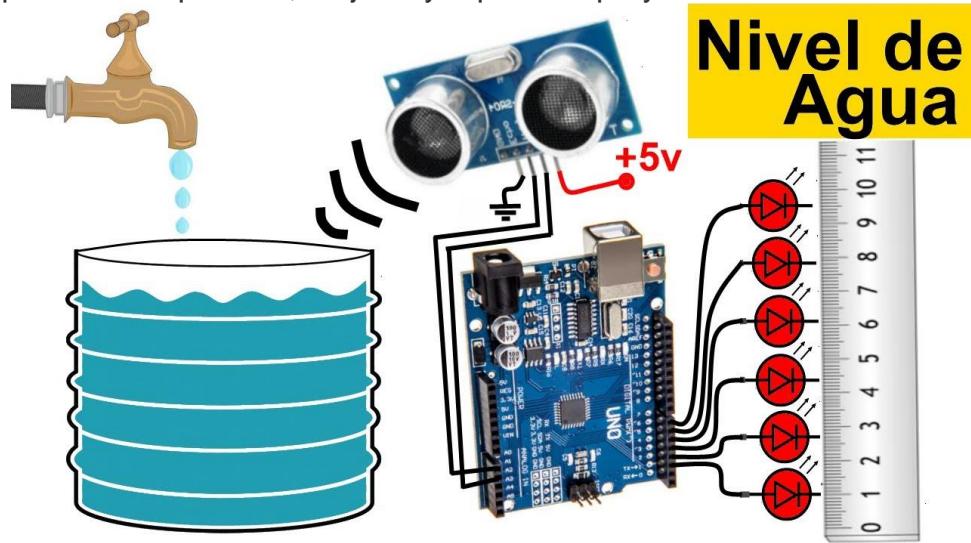
En resumen, un proyecto que implique el desarrollo de un sensor de nivel de agua no solo tiene la capacidad de proporcionar información vital sobre el recurso, sino que también puede ser fundamental para mejorar la eficiencia en el uso del agua, prevenir desastres y contribuir al bienestar general de las comunidades y del medio ambiente.

OBJETIVO DEL PROYECTO

El objetivo principal es crear un sistema que proporcione información útil sobre el consumo de agua y, potencialmente, contribuir a la conservación del recurso y a la gestión eficiente del mismo.

1. Desarrollo del sensor: Diseñar un sensor de nivel de agua preciso y confiable que pueda medir con precisión los diferentes niveles de agua en un recipiente, depósito, embalse u otra fuente.
2. Interfaz de usuario: Crear una interfaz amigable utilizando Arduino que permita visualizar los niveles de agua detectados, posiblemente a través de una pantalla LCD, LEDs o incluso en una aplicación para dispositivos móviles.
3. Calibración y precisión: Asegurar la calibración adecuada del sensor para garantizar mediciones precisas y confiables en diferentes condiciones ambientales y tipos de agua (limpia, sucia, salada, etc.).
4. Conectividad y notificación: Implementar la capacidad de enviar alertas o notificaciones cuando se alcancen ciertos niveles de agua, utilizando módulos de comunicación como Wi-Fi, Bluetooth o GSM/GPRS.
5. Eficiencia energética: Optimizar el consumo de energía del sensor y el sistema Arduino para garantizar una operación prolongada con una fuente de energía limitada, como baterías o energía solar.

6. Robustez y durabilidad: Diseñar el sensor y el sistema Arduino de manera que sean robustos y puedan resistir condiciones adversas, como variaciones de temperatura, humedad o exposición al agua.
7. Integración y escalabilidad: Permitir la fácil integración del sensor en sistemas más amplios, como sistemas de riego automatizado, sistemas de gestión de agua o proyectos de monitoreo a gran escala.
8. Documentación y accesibilidad: Proporcionar documentación clara y accesible, así como un código bien comentado, para que otros usuarios puedan comprender, mejorar y replicar el proyecto.



FUNDAMENTOS DE ARDUINO

¿Qué es Arduino?

Arduino es una plataforma de hardware de código abierto diseñada para facilitar la creación de prototipos electrónicos y proyectos interactivos. Consiste en hardware y software que permite a los entusiastas, estudiantes, artistas y profesionales desarrollar dispositivos electrónicos de manera accesible y fácil.

En términos simples, Arduino es una placa con un microcontrolador que puede ser programada para realizar una amplia variedad de tareas. Lo que hace a Arduino tan popular es su simplicidad y versatilidad. La placa de Arduino es fácil de usar, incluso para principiantes, y su software, el IDE (Entorno de Desarrollo Integrado) de Arduino, simplifica la escritura de código para controlar el hardware. La comunidad de Arduino es activa y diversa, lo que significa que hay una gran cantidad de recursos, tutoriales y proyectos disponibles en línea. Esto facilita a los usuarios aprender y desarrollar sus habilidades en electrónica y programación.

Arduino ha sido utilizado para una amplia gama de proyectos, desde dispositivos domésticos inteligentes hasta arte interactivo, sistemas de monitoreo ambiental, robótica, y mucho más. Su flexibilidad y asequibilidad lo convierten en una herramienta poderosa para la innovación y la creatividad en el campo de la electrónica y la computación física.

CONCEPTOS BÁSICOS

1. Microcontrolador: Arduino utiliza un microcontrolador que actúa como el cerebro del sistema. Los más comunes son basados en la arquitectura AVR de Atmel, como el ATmega328P. Este chip ejecuta el programa que es cargado en la placa Arduino.
2. IDE de Arduino: El entorno de desarrollo integrado (IDE) de Arduino es donde escribes, compilas y cargas tu código al microcontrolador. Es una interfaz amigable que simplifica la escritura de código en lenguaje C/C++.
3. Pines de E/S (Entrada/Salida): Los pines de E/S son los puntos de conexión física en la placa Arduino. Pueden ser digitales (que pueden ser configurados

como entrada o salida) o analógicos (para entradas analógicas como sensores).

4. Código: Los programas de Arduino están escritos en un dialecto de C/C++ simplificado. Estos programas controlan el comportamiento del microcontrolador, desde encender un LED hasta leer datos de sensores o manejar motores.
5. Sensores y Actuadores: Los sensores son dispositivos que capturan datos del entorno (como temperatura, luz, movimiento, etc.). Los actuadores son dispositivos que realizan acciones físicas basadas en las instrucciones del programa (como motores, LEDs, relés, etc.).
6. Librerías: El ecosistema de Arduino ofrece una gran cantidad de librerías que simplifican la interacción con hardware específico. Estas librerías proporcionan funciones predefinidas para controlar sensores, actuadores y otros dispositivos.
7. Prototipado: Arduino es excelente para el prototipado rápido. Puedes conectar fácilmente componentes electrónicos utilizando breadboards, jumpers y shields (placas complementarias) para crear y probar tus ideas antes de implementarlas en un diseño más permanente.

ESTADO DEL ARTE

¡Los proyectos con Arduino son infinitos! Aquí se muestran algunos ejemplos para inspirar en este amplio mundo:

1. Estación meteorológica: Utiliza sensores de temperatura, humedad y presión para crear un dispositivo que mida y muestre datos climáticos en tiempo real.
2. Robot seguidor de línea: Construye un robot que pueda seguir una línea trazada en el suelo mediante sensores infrarrojos o de ultrasonido.
3. Sistema de riego automático: Crea un sistema que detecte la humedad del suelo y active el riego cuando sea necesario.

4. Control domótico: Conecta dispositivos como luces, persianas o electrodomésticos a Arduino para controlarlos de forma remota o automatizada.
5. Instrumento musical: Construye un piano, una batería electrónica o un controlador MIDI utilizando sensores y actuadores.
6. Seguridad y vigilancia: Diseña un sistema de seguridad con detección de movimiento, alarmas y notificaciones a través de mensajes de texto o correo electrónico.
7. Coche o dron controlado por Arduino: Crea un vehículo terrestre o aéreo controlado por Arduino con sensores para evitar obstáculos.
8. Termómetro y termostato inteligente: Controla la temperatura ambiente y activa un sistema de calefacción o refrigeración según sea necesario.
9. Estación de monitoreo de calidad del aire: Utiliza sensores de gases para medir la calidad del aire y mostrar los resultados en tiempo real.
10. Dispositivos IoT: Crea proyectos de Internet de las cosas conectando Arduino a la red para controlar y monitorear dispositivos desde cualquier lugar.
11. Reloj binario: Este es un proyecto muy sencillo que te permitirá aprender los conceptos básicos de Arduino, como la programación, la conexión de componentes y el uso de sensores. En este proyecto, utilizarás un Arduino Uno, un display LCD y un botón para crear un reloj que muestra la hora en código binario.
12. Detector de movimiento: Este proyecto es un poco más avanzado que el anterior, pero aún es bastante sencillo. En este proyecto, utilizarás un Arduino Uno, un sensor de movimiento y un LED para crear un detector de movimiento que enciende un LED cuando detecta movimiento.
13. Piano electrónico: Este proyecto es una excelente manera de aprender a utilizar pulsadores y a generar sonido. En este proyecto, utilizarás un Arduino Uno, un conjunto de pulsadores y un altavoz para crear un piano electrónico.
14. Lámpara de colores: Este proyecto es una excelente manera de aprender a utilizar LEDs y a controlar su color. En este proyecto, utilizarás un Arduino Uno, un conjunto de LEDs RGB y un potenciómetro para crear una lámpara de colores que cambia de color según el valor del potenciómetro.

Estas ideas son solo el comienzo. Puedes personalizar y combinar diferentes componentes, sensores y actuadores para crear tus propios proyectos únicos con Arduino. ¡La creatividad es el límite!

Ejemplos en línea:

- Lámpara de colores: https://youtu.be/mwwMJocGT2g?si=MUAZ1LHpnP_DQ_9N
- Grúa con Arduino: <https://www.youtube.com/watch?v=B2lwaLmHDEI>
- Garaje con sensor de distancia: <https://www.youtube.com/watch?v=1hzmlJcTs-0>
- Minijuego con Arduino: <https://www.youtube.com/watch?v=Kk6Hax4D6YI>

DESCRIPCIÓN DEL PROBLEMA

Implementación de un sistema de medición del nivel de agua confiable y preciso, mediante leds los cuales nos indicaran que nivel de líquido tiene el contenedor actualmente para evitar el desperdicio del agua.

JUSTIFICACIÓN

El problema del control y la medición del nivel del agua es relevante por varias razones fundamentales:

1. Escasez de agua: En muchas regiones del mundo, la escasez de agua potable es un problema crítico. Tener la capacidad de medir con precisión el nivel del agua es esencial para gestionar y conservar este recurso limitado.
2. Gestión de recursos hídricos: Para mantener un suministro adecuado de agua para uso doméstico, agrícola e industrial, es esencial comprender y controlar los niveles de agua en embalses, tanques de almacenamiento, ríos y otros cuerpos de agua.
3. Prevención de desastres naturales: Los niveles excesivamente altos de agua pueden provocar inundaciones catastróficas que afectan a comunidades enteras. Medir y monitorear estos niveles es crucial para anticipar y responder a posibles desastres.
4. Agricultura y riego: En la agricultura, el control preciso del nivel de agua es esencial para un riego eficiente y oportuno. Medir el nivel de agua en el suelo o en sistemas de riego puede aumentar la productividad y conservar recursos hídricos.

La solución de utilizar un sensor para medir el nivel del agua es necesaria porque:

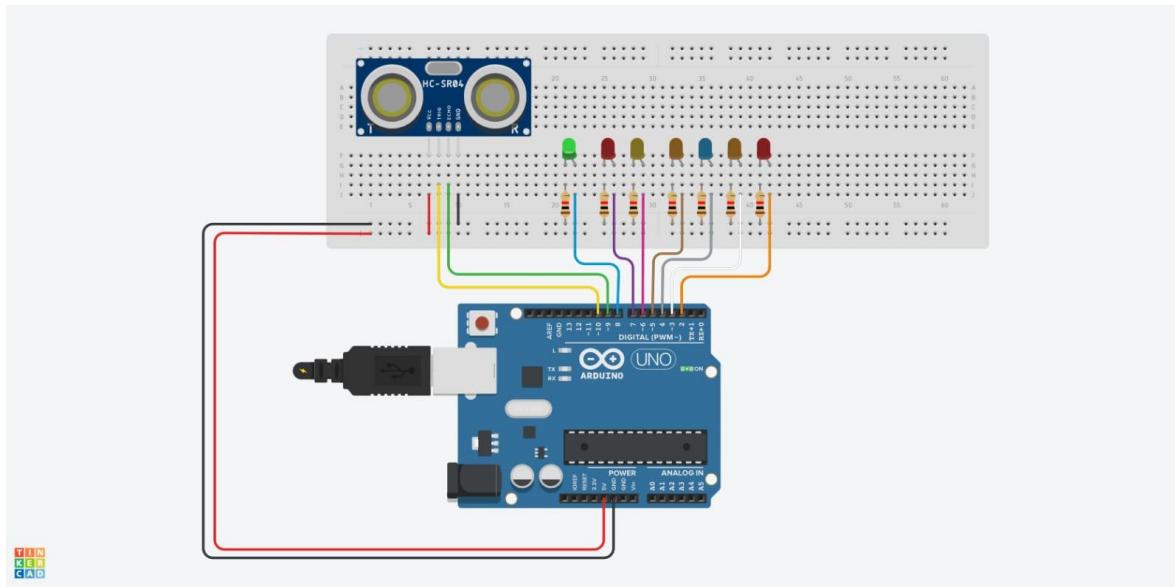
1. Precisión y control: Los sensores proporcionan mediciones precisas y en tiempo real de los niveles de agua, lo que permite un control más efectivo y una gestión precisa de este recurso.
2. Automatización y eficiencia: Al integrar sensores con sistemas de control, como Arduino, se puede automatizar la respuesta ante cambios en los niveles de agua, permitiendo una gestión más eficiente y reduciendo el desperdicio.

3. Alerta temprana: Los sensores pueden proporcionar alertas tempranas cuando los niveles de agua alcanzan puntos críticos, permitiendo tomar medidas preventivas antes de que se conviertan en problemas mayores, como inundaciones.
4. Monitoreo continuo: La capacidad de monitorear constantemente los niveles de agua en tiempo real es esencial para anticipar y responder rápidamente a cualquier cambio que pueda afectar a las comunidades, a la agricultura o al suministro de agua potable.

En resumen, la utilización de sensores para medir el nivel del agua es crucial para abordar problemas relacionados con la escasez, la gestión eficiente del recurso, la prevención de desastres y la mejora de la productividad agrícola, proporcionando datos precisos y oportunidades para una gestión inteligente y sostenible del agua.



DISEÑO EN TINKERCARD



Código del programa

```
const int trigPin = 7;  
const int echoPin = 6;  
const int ledPin1 = 2;  
const int ledPin2 = 3;  
const int ledPin3 = 4;  
  
void setup() {  
    Serial.begin(9600);  
    pinMode(trigPin, OUTPUT);  
    pinMode(echoPin, INPUT);  
    pinMode(ledPin1, OUTPUT);  
    pinMode(ledPin2, OUTPUT);  
    pinMode(ledPin3, OUTPUT);  
}  
144
```

```
void loop() {
    long duration, distance;

    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);

    duration = pulseIn(echoPin, HIGH);

    distance = (duration * 0.0343) / 2;

    Serial.print("Distancia: ");
    Serial.print(distance);
    Serial.println(" cm");

    if (distance < 3) {
        digitalWrite(ledPin1, HIGH);
        digitalWrite(ledPin2, HIGH);
        digitalWrite(ledPin3, HIGH);
    } else if (distance < 8) {
        digitalWrite(ledPin1, HIGH);
        digitalWrite(ledPin2, HIGH);
        digitalWrite(ledPin3, LOW);
    } else if (distance < 14) {
        digitalWrite(ledPin1, HIGH);
        digitalWrite(ledPin2, LOW);
    }
}
```

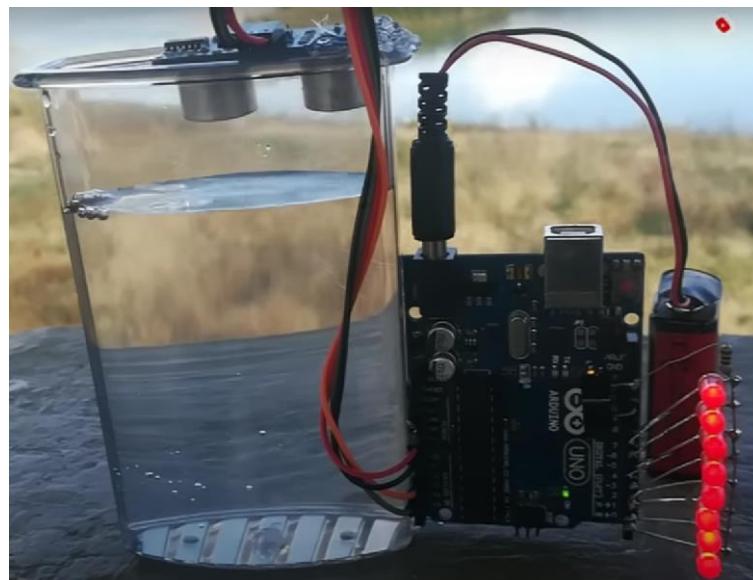
```

digitalWrite(ledPin3, LOW);
} else {
    digitalWrite(ledPin1, LOW);
    digitalWrite(ledPin2, LOW);
    digitalWrite(ledPin3, LOW);
}

delay(500); // Ajusta el retardo según sea necesario
}

```

ARQUITECTURA DEL SISTEMA



SELECCIÓN DE COMPONENTES

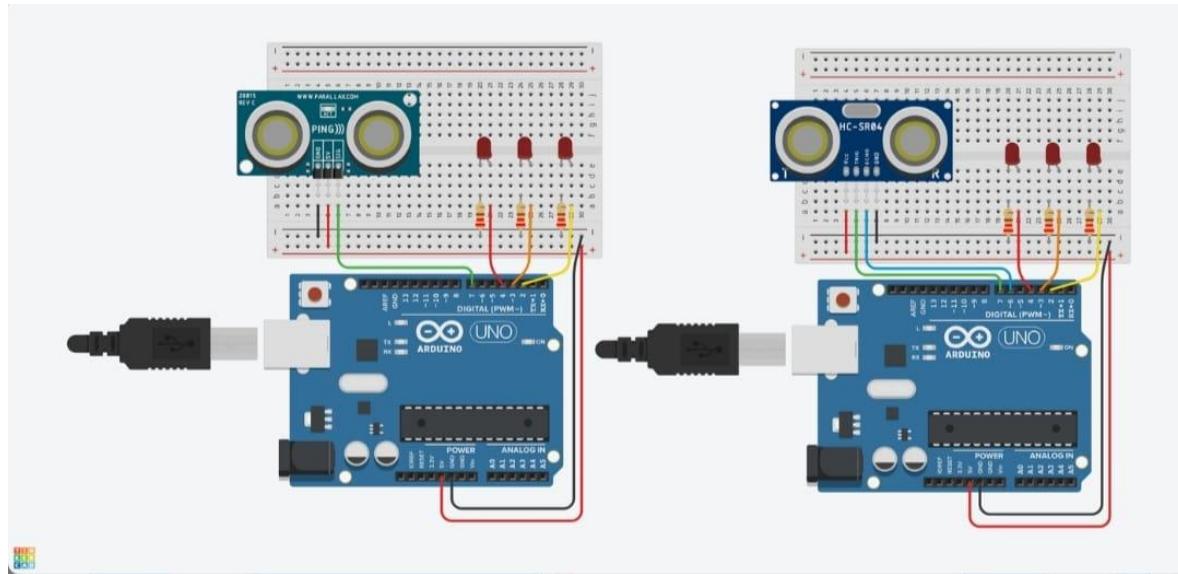
- Arduino: Es una placa de desarrollo que contiene un microcontrolador que se puede programar para realizar diversas tareas. Tiene pines de entrada y salida que permiten conectar componentes electrónicos para interactuar con el entorno. Proporciona la capacidad de controlar y procesar los datos

recopilados por el sensor ultrasónico. Permite la programación para interpretar la información de distancia medida y tomar decisiones, como activar o desactivar los LEDs indicadores en función del nivel del agua.

- Sensor ultrasónico: Este sensor utiliza ondas ultrasónicas para medir distancias. Tiene un transmisor que emite ondas ultrasónicas y un receptor que detecta las ondas reflejadas. Basándose en el tiempo que tarda en regresar la señal reflejada, calcula la distancia a un objeto. Es clave en este proyecto, ya que puede medir la distancia desde su ubicación hasta la superficie del agua. Al enviar ondas ultrasónicas y medir el tiempo que tardan en regresar después de reflejarse en el agua, el sensor puede calcular la distancia y, por ende, el nivel del agua.
- 4 jumpers macho hembra y 4 jumpers macho macho: Son cables con conectores en cada extremo. Los jumpers macho-hembra tienen un extremo macho (pin) y el otro extremo hembra (hueco), mientras que los macho-macho tienen pines en ambos extremos. Se utilizan para conectar componentes en un protoboard o en la placa Arduino de manera flexible y sin soldaduras. Estos jumpers son esenciales para conectar el sensor ultrasónico, los LEDs y las resistencias a los pines de entrada y salida de la placa Arduino y el protoboard. Facilitan la conexión y la organización del circuito.
- 1 protoboard: Es una placa con agujeros conductores que permiten conectar componentes electrónicos sin necesidad de soldar. Facilita la creación de circuitos temporales y experimentación electrónica. Sirve como una plataforma para ensamblar los componentes de manera temporal y sin necesidad de soldar. Permite conectar los componentes de manera flexible para crear un circuito funcional.
- 3 resistencias de 200 o 220: Las resistencias limitan la corriente que pasa a través de los LEDs para evitar daños. En este caso, las resistencias de 200 o 220 ohmios se utilizan comúnmente con LEDs para operarlos de manera segura. En este caso, las resistencias se utilizarían para limitar la corriente que pasa a través de los LEDs, asegurando que funcionen correctamente y no se dañen por la corriente eléctrica excesiva.

- 3 leds: Son diodos emisores de luz que convierten la corriente eléctrica en luz visible cuando están encendidos. Se utilizan para indicar visualmente el estado de un circuito o como parte de un diseño interactivo. Los LEDs son una forma visual de indicar el nivel del agua. Dependiendo de la distancia medida por el sensor ultrasónico (y, por lo tanto, del nivel del agua), los LEDs se encenderán o apagarán, proporcionando una representación visual del nivel del agua en el recipiente o área medida.

DIAGRAMA DE CONEXIONES



En la parte de sensor ultrasónico, el VCC va conectado el pin 5V de Arduino
GND al proto en la fila negativa Y conecta el gnd de 148arduino en la misma fila
Trig al pin digital 7 del Arduino
Echo al pin digital 6 de Arduino con los 4 jumpers macho hembra

Haz una conexión del Arduino al proto con los jumpers macho macho en los pin 2, 3, y 4

Poniendo las conexiones en diferentes filas después pon las resistencias en la misma fila seguidas de los leds y ingresa la pata más larga en la misma fila que la resistencia y la más corta en la fila horizontal negativa que se encuentra los gnd del Arduino y del sensor ultrasónico

EXPLICACIÓN DEL CÓDIGO

Este código de Arduino utiliza un sensor ultrasónico para medir la distancia a un objeto y controlar tres LEDs en función de esa distancia.

Aquí está cómo funciona paso a paso:

1. Configuración inicial:

- Se definen las variables para los pines del sensor ultrasónico (trigPin y echoPin) y los pines de los LEDs (ledPin1, ledPin2 y ledPin3).
- En la función `setup()`, se inicializan los pines como entradas (`INPUT`) y salidas (`OUTPUT`) según sea necesario.

2. Loop principal:

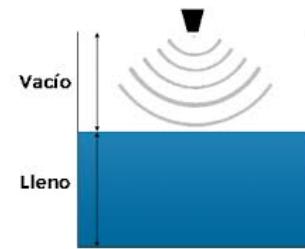
- El bucle `loop()` realiza la medición de distancia continuamente.
- Se envía un pulso corto al pin trigPin del sensor ultrasónico para activar la medición.
- La duración del pulso de respuesta en el pin echoPin se mide con la función `pulseIn()`, que captura el tiempo que tarda la señal ultrasónica en viajar y regresar.
- La distancia se calcula utilizando la fórmula de conversión de tiempo a distancia (considerando la velocidad del sonido).
- Se imprime la distancia medida en centímetros en el monitor serial para su visualización.

3. Control de los LEDs según la distancia medida:

- Dependiendo de la distancia medida, se encienden o apagan los LEDs según las condiciones establecidas por las declaraciones `if-else`.
- Si la distancia es menor a 3 cm, los tres LEDs se encienden.
- Si la distancia es menor a 8 cm pero mayor que 3 cm, se encienden dos LEDs y se apaga uno.
- Si la distancia es menor a 14 cm pero mayor que 8 cm, se enciende un LED y se apagan dos.
- Si la distancia es mayor a 14 cm, todos los LEDs se apagan.

4. Retardo:

- Se agrega un retardo de 500 milisegundos (0.5 segundos) al final del bucle para proporcionar un tiempo entre cada medición y cambio en los LEDs.



ALGORITMOS Y FUNCIONES CLAVE

En este código de Arduino para medir la distancia con un sensor ultrasónico y controlar LEDs en función de esa distancia, hay algunos elementos clave:

1. Configuración inicial (`void setup()`):

- Se establecen los pines utilizados para el sensor ultrasónico (trigPin como salida y echoPin como entrada) y los pines de los LEDs (ledPin1, ledPin2, ledPin3) como salidas.

2. Bucle principal (`void loop()`):

- Se mide la distancia utilizando el sensor ultrasónico:
- Se activa el sensor enviando un pulso corto al pin `trigPin`.
- Se mide el tiempo que tarda la señal ultrasónica en regresar al pin `echoPin` con la función `pulseIn()`.
- Se convierte este tiempo en una distancia en centímetros utilizando una fórmula basada en la velocidad del sonido.
- Se imprime la distancia medida en el monitor serial para visualización.

3. Control de LEDs según la distancia (`if-else`):

- Dependiendo de la distancia medida, se encienden o apagan los LEDs:
- Si la distancia es menor a 3 cm, se encienden los tres LEDs.
- Si la distancia es menor a 8 cm pero mayor que 3 cm, se encienden dos LEDs y se apaga uno.
- Si la distancia es menor a 14 cm pero mayor que 8 cm, se enciende un LED y se apagan dos.
- Si la distancia es mayor a 14 cm, todos los LEDs se apagan.

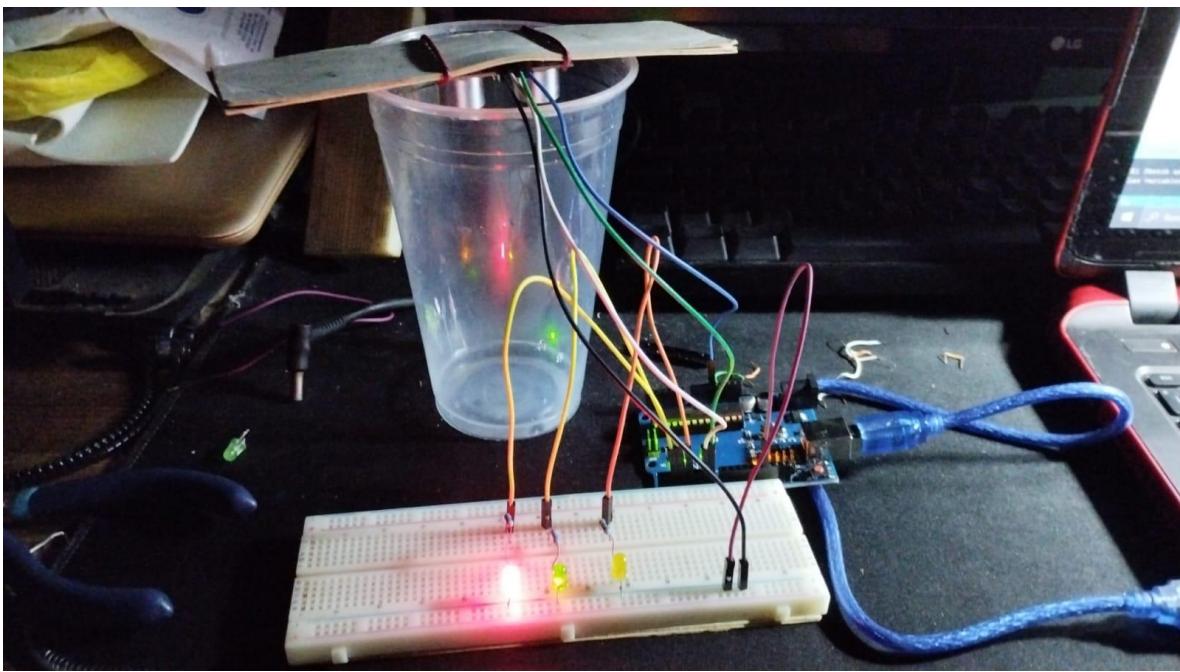
4. Retardo (`delay()`):
 - Al final del bucle, se incluye un retardo de 500 milisegundos (0.5 segundos) para ajustar el tiempo entre cada medición y cambio en los LEDs.

Los elementos clave son la configuración de pines, la medición de distancia con el sensor ultrasónico, el control de los LEDs según la distancia medida y la inclusión de un retardo para regular el tiempo entre mediciones. Estos componentes trabajan juntos para medir y representar visualmente la distancia medida por el sensor ultrasónico utilizando los LEDs.

CASOS DE PRUEBAS

1. Al hacer pruebas con el Arduino, los leds fallaron demasiado porque el cable estaba quemado.
2. El protoboard fue utilizado para poder equilibrar mejor para el uso
3. Ademas, luego de hacer un par de pruebas más se dio a conocer que el Arduino estaba dañado, así que se tuvo que cambiar porque se retrasaba mucho el poder hacer que funcionara de manera correcta.

RESULTADOS DE PRUEBAS



GUÍA DE USUARIO

Al ser una tarjeta muy compleja hay que tener cuidado con la programación al igual que con las conexiones ya que es muy fácil confundirse así que es más recomendable que hagas primero tu programa y después el circuito.

Tomando en cuenta los componentes y los pines usados del Arduino es recomendable hacer el programa poco a poco conforme vas desarrollando el circuito y no hacerlo de golpe para tener un margen de error menor.

RESULTADOS Y CONCLUSIONES

Resultados:

1. Medición precisa de la distancia:
 - Se logra una medición precisa (que se usó en el código) de la distancia entre el sensor ultrasónico y la superficie del agua.
2. Control de los LEDs según el nivel del agua:
 - Los LEDs se encienden o apagan correctamente dependiendo del nivel de agua detectado.
 - Los cambios en la distancia detectada se reflejan visualmente a través del encendido y apagado de los LEDs según las condiciones establecidas en el código.
3. Visualización y respuesta en tiempo real:
 - Se observa una respuesta en tiempo real en la visualización de los LEDs conforme se modifica el nivel del agua.

Conclusiones:

1. Efectividad del sensor ultrasónico:
 - El sensor ultrasónico es efectivo para medir el nivel del agua, ofreciendo una solución práctica y económica para este propósito.

2. Aplicabilidad del proyecto:

- Se demuestra la aplicabilidad del uso de Arduino y sensores como el ultrasónico para la medición y control del nivel del agua en diferentes contextos y proyectos.

3. Importancia del monitoreo del agua:

- La capacidad de medir y controlar el nivel del agua es crucial para la gestión eficiente y la conservación de este recurso en diversas aplicaciones, desde sistemas de riego hasta monitoreo de reservas de agua.

4. Potencial de mejoras:

- Se identifica el potencial para mejorar el proyecto, como la integración con sistemas de alerta o la optimización de la precisión de la medición.

En resumen, el proyecto demuestra la viabilidad de utilizar un sensor ultrasónico con Arduino para medir y controlar el nivel del agua, destacando su utilidad en aplicaciones prácticas y resaltando la importancia de monitorear este recurso vital. Además, sugiere oportunidades para mejorar y expandir la funcionalidad del proyecto.

TRABAJO FUTURO – MEJORAS Y EXPANSIONES

Mejoras Técnicas:

1. Calibración y precisión:

- Investigar y aplicar métodos para mejorar la precisión de la medición del sensor ultrasónico, como la calibración más precisa o el uso de múltiples sensores para reducir posibles errores.

2. Ajuste de sensibilidad:

- Implementar un mecanismo para ajustar la sensibilidad del sensor ultrasónico para adaptarse a diferentes condiciones o tipos de agua.

3. Filtrado de datos:

- Utilizar técnicas de filtrado de datos para eliminar lecturas incorrectas o ruidosas, mejorando así la precisión de las mediciones.

Funcionalidades Adicionales:

1. Alertas y notificaciones:

- Integrar un sistema de alertas para notificar cuando el nivel del agua alcance ciertos límites críticos, ya sea mediante mensajes de texto, correo electrónico o notificaciones visuales/sonoras.

2. Registro de datos:

- Implementar una función para registrar los datos de nivel de agua a lo largo del tiempo, permitiendo un seguimiento histórico y análisis de patrones.

Integración y Conectividad:

1. Conexión a redes externas:

- Integrar el proyecto con la IoT (Internet de las cosas) para permitir la monitorización remota del nivel de agua y el control a través de internet.

2. Aplicaciones móviles o web:

- Desarrollar una aplicación móvil o una interfaz web que permita visualizar y controlar el nivel de agua desde dispositivos móviles o computadoras.

Optimización de Energía y Recursos:

1. Uso eficiente de energía:

- Implementar estrategias para reducir el consumo de energía del sistema, como modos de bajo consumo para períodos de inactividad.

2. Uso de energías renovables:

- Explorar la posibilidad de alimentar el sistema utilizando energías renovables, como paneles solares, para una mayor autonomía y sostenibilidad.

Ampliación del Alcance del Proyecto:

1. Integración con sistemas existentes:

- Adaptar el proyecto para integrarlo con sistemas de riego automatizado, sistemas de monitoreo de presas, o aplicaciones industriales que requieran el control del nivel del agua.

2. Implementación en diferentes entornos:

- Probar y adaptar el proyecto para su uso en diversos entornos, como sistemas de depuración de agua, estaciones meteorológicas o en la gestión de aguas residuales.

Estas mejoras y expansiones tienen el potencial de aumentar la precisión, la funcionalidad y la utilidad del proyecto, permitiendo su aplicación en una variedad de escenarios y proporcionando un mayor valor en términos de monitoreo y control del nivel del agua.

REFERENCIAS

Programafacil.com. (03 de Noviembre de 2023). Obtenido de Sensor de nivel de agua con Arduino: <https://programarfacil.com/blog/arduino-blog/sensor-de-nivel-de-agua-con-arduino/>

Xataka Basics. (03 de Noviembre de 2023). Obtenido de Qué es Arduino, cómo funciona y qué puedes hacer con uno: <https://www.xataka.com/basics/que-arduino-como-funciona-que-puedes-hacer-uno>

ANEXOS

- Lámpara de colores: https://youtu.be/mwwMJocGT2g?si=MUAZ1LHpnP_DQ_9N
- Grúa con Arduino: <https://www.youtube.com/watch?v=B2lwaLmHDEI>
- Garaje con sensor de distancia: <https://www.youtube.com/watch?v=1hzmlJcTs-0>
- Minijuego con Arduino: <https://www.youtube.com/watch?v=Kk6Hax4D6YI>