**16-bit registers A and B hold 8-bit values x and y respectively. Write a program that moves these values so that AH= x and AL = y**

MOV A, 73

MOV B, 35

MOVB AH, AL

MOVB AL, BL

HLT

**Do the same thing by using only 16-bit instructions and OR operation.**

MOV A, 73

MOV A, 0x0049

MOV B, 0x0023

OR A, 0x4900

AND A, 0xFF00

OR A, B

HLT

MOV B, 35

OR A, 18688

AND A, 0xFF00

OR A, B

HLT

**Define variables x and y with the initial values of your choice. Write a program that switches the values of x and y.**

JMP main

x: DW 0x0100

y: DW 0x0111

main:

MOV A, [x]

MOV B, [y]

MOV [x], B

MOV [y], A

HLT

**Write a program that computes z = 16 · (2x − y). Use bit shifting for multiplication.**

JMP main

x: DW 0x0100

y: DW 0x0111

z: DW 0

main:

MOV A, [x]

SHL A, 1

MOV B, [y]

SUB A, B

SHL A, 4

MOV [z], A

HLT

**Write a program that computes z = x2 – y2. Use instruction MUL to square a value.**

JMP main

x: DW 0x0100

y: DW 0x0111

z: DW 0

main:

MOV A, [x]

MUL [x]

MOV B, A

MOV A, [y]

MUL [y]

SUB B, A

MOV [z], B

HLT

**Implement the following conditions in assembly language:**

1. **if (x == 0 || x == 2 || x > y) {**

**x = x - y; }**

**else {**

**if (x < y)**

**x = x + y; }**

JMP main

x: DW 3

y: DW 4

main:

MOV A, [x]

CMP A, 0

JE subtract

CMP A, 2

JE subtract

CMP A, [y]

JA subtract

JB add

HLT

subtract:

SUB A, [y]

MOV [x], A

HLT

add:

ADD A, [y]

MOV [x], A

HLT

1. **if (x + 1 > y)**
2. **if (x % 2 == 0)**

**x = x / 2;**

JMP main

x: DW 4

main:

MOV A, [x]

DIV 2

MOV B, A

MOV A, [x]

DEC A

DIV 2

CMP A, B

JE nothing

MOV [x], B

HLT

nothing:

HLT

**x = x – 1;**

JMP main

x: DW 4

y: DW 4

main:

MOV A, [x]

ADD A, 1

CMP A, [y]

JA decrease

HLT

decrease:

MOV B, [x]

DEC B

MOV [x], B

HLT

**Write a program that determines if a given integer is divisible by 3. The idea is to subtract the constant 3 from the given number until that number is reduced below 3. If the result is 0, the number is divisible by 3.**

JMP main

n: DW 6

main:

MOV A, [n]

MOV B, 3

loop:

SUB A, B

CMP A, 0

JE divisable

CMP A, 3

JB break

JMP loop

divisable:

MOV C, [n]

HLT

break:

HLT

**Use the MUL instruction to simplify**

**the assembly program that**

**computes the factorial**

JMP main

n: DW 5

f: DW 1

main:

MOV B, [f]

MOV A, 2

MOV C, 2

loop:

CMP C, [n]

JA endloop

MUL B

MOV B, A

INC C

MOV A, C

JMP loop

HLT

endloop:

MOV [f], B

HLT

**Write a program that counts how many bits in**

JMP main

x: DW 0xF003

z: DW 0

main:

MOV A, [x]

MOV B, 16

MOV C, 0

forloop:

CMP B, 0

JBE end

DEC B

SHL A, 1

JC increament

JMP forloop

increament:

INC C

JMP forloop

end:

MOV [z], C

HLT

**register A are set to 1. This can be done by**

**shifting left the register 16 times while checking**

**the C flag for the value of the discarded bit.**

MOV A, 0x1F07

MOV B, 0

while:

CMP A, 0x0000

JE out

SHL A, 1

JC do

JMP while

do:

INC B

JMP while

out:

HLT

**Write a program that determines how much memory is currently free. The idea is to read every byte of memory between the addresses 0x0000 and 0x0FFF, and count the number of zero bytes after the last non-zero byte.**

MOV C, 0

MOV B, 0

loop:

CMP C, 0x0FFF

JE break

MOVB AL, [C]

CMP A, 0

JE increase

MOV B, 0

INC C

JMP loop

HLT

increase:

INC B

INC C

JMP loop

HLT

break:

HLT

JMP main

z: DW 0

main:

MOV A, 0

MOV C, 0

while:

CMP A, 0x0FFF

JA endwhile

MOVB BL, [A]

CMPB BL, 0

JE increasecnt

MOV C, 0

INC A

JMP while

increasecnt:

INC C

INC A

JMP while

endwhile:

MOV [z], C

HLT

**Write a function add(x1, x2, x3, ...) that can receive any number of summands x1, x2, . . . , xn and returns their sum in register A. Let the number of summands n be received in register A and all the summands on stack.**

JMP main

n: DW 5

z: DW 0

add:

POP D

POP C

ADD A, C

PUSH A

PUSH D

RET

main:

MOV SP, 0x0FFF

MOV B, [n]

loop:

CMP B, 0

JA function

HLT

function:

PUSH B

CALL add

POP A

MOV [z], A

DEC B

JMP loop

HLT

JMP main

add:

POP C

loop:

CMP A, 1

JB return

POP B

ADD D, B

MOV B, 0

DEC A

JMP loop

return:

PUSH C

RET

main:

MOV D, 0

MOV SP, 0x0FFF

PUSH 5

PUSH 2

PUSH 1

PUSH 3

PUSH 7

MOV A, 5

CALL add

MOV A, D

HLT

**Write a function abs(x) that returns the absolute value of the given 16-bit signed integer x. Suppose that x is in the two’s complement format.**

JMP main

x: DW 0x00FF

s: DW 0

abs:

POP D

POP A

MOV C, 1

loop:

CMP C, 17

JE break

AND A, 0x8000

CMP A, 0x8000

JE do

SHL B, 1

OR A, 0x8000

SHR A, 15

ADD B, A

MOV A, [x]

SHL A, C

INC C

JMP loop

HLT

**Write a function sign(x) that returns the sign of**

**the given 16-bit signed integer x, i.e. 1 if the**

**number is positive, −1 if it is negative, and 0**

**if it is 0. Suppose that x is in the two’s**

do:

SHL B, 1

AND A, 0

SHR A, 15

ADD B, A

MOV A, [x]

SHL A, C

INC C

JMP loop

break:

ADD B, 1

PUSH B

PUSH D

RET

main:

MOV SP, 0x0FFF

MOV A, [x]

PUSH A

CALL abs

POP B

MOV [s], B

HLT

**complement format.**

JMP main

sign:

POP C

CMP A, 0

JE zero

AND A, 0x8000

CMP A, 0

JE positive

CMP A, 0x8000

JE negative

zero:

MOV B, 0

JMP return

positive:

MOV B, 1

JMP return

negative:

MOV B, 0xFFFF

return:

PUSH C

RET

main:

MOV SP, 0x0FFF

MOV A, 0x0003

CALL sign

HLT

**Write a function sum\_arrays(a1, a2, len) that receives**

**Write a function str\_empty(s) that receives a string and returns 1 if the string is empty, and 0 otherwise.**

JMP main:

s: DB "HELLO WORLD"

DB 0

str\_empty:

MOV C, A

MOVB BL, [C]

CMPB BL, 0

JE do

MOV A, 1

JMP return

do:

MOV A, 0

JMP return

return:

RET

main:

MOV SP, 0x0FFF

MOV A, s

CALL str\_empty

HLT

**(by reference) arrays a1 and a2 of 16-bit integers, both**

**of the same size len, and adds a1 ← a1 + a2 element-wise.**

**The function returns no value.**

sum\_arrays:

MOV C, 0

loop:

MOV D, 0

CMP C, 5

JE end

MOV B, a2

MOV A, C

MUL 2

ADD B, A

MOV D, [B]

MOV B, a1

MOV A, C

MUL 2

ADD B, A

ADD D, [B]

MOV [B], D

INC C

JMP loop

end:

HLT

JMP sum\_arrays

a1: DW 1

DW 2

DW 3

DW 4

DW 5

a2: DW 6

DW 7

DW 8

DW 9

DW 10

**Write a function count\_words(s) that receives a string and returns the number of words contained in the string. Words are separated by one or more spaces.**

JMP main

nw: DW 0

s: DB "Program for counting words"

DB 0

count\_words:

MOV C, 0

MOV D, 1

loop:

MOVB BL, [A]

CMPB BL, 0

JE return

CMPB BL, 32

JE do

CMP D, 1

JE word

INC A

JMP loop

word:

INC C

MOV D, 0

INC A

JMP loop

do:

MOV D, 1

INC A

JMP loop

return:

RET

main:

MOV SP, 0x0FFF

MOV A, s

CALL count\_words

MOV [nw], C

HLT

**Write a function str\_cmp(s1, s2) that receives two**

**strings of any size and returns 1 if the strings are**

**equal, and 0 otherwise.**

main:

MOV SP, 0x0FFF

MOV A, s1

MOV B, s2

CALL str\_cmp

HLT

JMP main

s1: DB "Hel"

DB 0

s2: DB "Hello"

DB 0

str\_cmp:

loop:

MOVB CL, [A]

CMPB CL, 0

JE do

MOVB DL, [B]

CMPB DL, 0

JE notsame

CMPB CL, DL

JNE notsame

INC A

INC B

JMP loop

do:

MOVB DL, [B]

CMPB DL, 0

JE same

JMP notsame

notsame:

MOV A, 0

JMP return

same:

MOV A, 1

JMP return

return:

RET

**Write a program that enables you to type a text on the text display.**

MOV C, 0x1000

read:

CMP C, 0x100F

JA end

IN 5

CMP A, 0

JE read

MOV B, A

IN 6

AND B, 1

CMP B, 1

JE display

JMP read

display:

MOVB [C], AL

INC C

JMP read

end:

HLT

**The maximum length of the input text is 16 characters. When the**

**display is full, let the program terminate. Be careful to add new**

**characters only after the keydown events.**

**Write a program that can be aware of two keys being held down at the same time. Suppose we use four keys: ‘8’ - for up, ‘2’ - for down, ‘4’ - for left, and ‘6’ - for right. Let the words “UP”, “DOWN”, “LEFT”, or “RIGHT” appear on the text display while one of these four keys is being held down, and let the display be empty, if no key is being held. Now enable the user to hold down two keys for a diagonal direction, e.g. ‘8’ and ‘4’ for “LEFT UP”. This way the user could control eight directions of motion. This is achieved by processing both, the keydown and keyup keyboard events while adding (keydown) and subtracting (keydown) the directions. Certain key combinations, e.g. ‘8’ and ‘2’, should cancel each other out.**

main:

MOV SP, 0x0FFF ; Initialize the stack.

MOV A, 1 ; Enable keyboard interupts.

OUT 0

STI ; Enable interupts globaly.

loop:

; Wait keypress event.

MOV A, [keypress]

CMP A, 0

JE loop

MOV [keypress], 0 ; We received the keypress event.

CALL clear ; Clear text.

MOV [cursor], 0x1000 ; Reset the cursor position.

MOV A, [dirx] ; Check the left - right direction.

CMP A, 0xFFFF ; Is it left (-1)?

JNE checkright ; If not, check if right.

MOV A, str\_left ; Print out "left".

CALL print

JMP ydirection ; Check ydir

checkright:

CMP A, 1 ; Is it right (1)?

JNE ydirection ; If not, check ydir.

MOV A, str\_right ; Print out "left".

CALL print

ydirection:

MOV A, [diry] ; Check the up - down direction.

CMP A, 0xFFFF ; Is it up (-1)?

JNE checkdown ; If not, check if down.

MOV A, str\_up ; Print out "up".

CALL print

checkdown:

CMP A, 1 ; Is it down (1)?

JNE done ; If not, we are done.

MOV A, str\_down ; Print out "down".

CALL print

done:

JMP loop

HLT

JMP isr\_done

check6:

CMPB AL, '6' ; Was it '6'?

JNE isr\_done ; If not, do nothing.

MOV A, [dirx] ; Change the dirx variable.

ADD A, B ; right left.

MOV [dirx], A ; Store the new value.

isr\_done:

MOV A, 1 ; Remove the IRQ.

OUT 2

MOV [keypress], 1 ; Signal the keypress event.

POP B

POP A

IRET

cursor: DW 0x1000

print:

MOV D, [cursor] ; The current cursor position.

print\_loop:

MOVB BL, [A] ; Get the character.

CMPB BL, 0 ; Is it a terminal value?

JE print\_return ; If yes, we are done.

MOVB [D], BL ; Otherwise copy the character to the display cell.

INC A ; Compute the address of the next character.

INC D ; Compute the address of the next display cell.

JMP print\_loop ; Repeat for the next character.

print\_return:

MOV [cursor], D ; Remember the last cursor position.

RET

clear:

MOV A, 0x1000

clear\_loop:

CMP A, 0x1010

JE clear\_return

MOV [A], 0

INC A

JMP clear\_loop

clear\_return:

RET

str\_left: DB "LEFT \x00"

str\_right: DB "RIGHT \x00"

str\_up: DB "UP\x00"

str\_down: DB "DOWN\x00"

JMP main

JMP isr

dirx: DW 0

diry: DW 0

keypress: DW 0

isr:

PUSH A

PUSH B

IN 5 ; Read KBDSTATUS.

CMP A, 1 ; Is it keydown?

JNE checkkeyup ; If not, check if keyup.

MOV B, 1 ; Add a direction.

JMP setkey

checkkeyup:

CMP A, 2 ; Is it keyup?

JE keyup ; Yes, process the keyup event.

IN 6 ; Read key code to clear it.

MOV [dirx], 0

MOV [diry], 0

JMP isr\_done

keyup:

MOV B, 0xFFFF ; Subtract a direction.

setkey:

IN 6 ; Which key was pressed?

CMPB AL, '8' ; Was it '8'?

JNE check2 ; If not, check key '2'.

MOV A, [diry] ; Change the diry variable.

SUB A, B ; Move up.

MOV [diry], A ; Store the new value.

JMP isr\_done

check2:

CMPB AL, '2' ; Was it '8'?

JNE check4 ; If not, check key '4'.

MOV A, [diry] ; Change the diry variable.

ADD A, B ; Move down.

MOV [diry], A ; Store the new value.

JMP isr\_done

check4:

CMPB AL, '4' ; Was it '4'?

JNE check6 ; If not, check key '6'.

MOV A, [dirx] ; Change the dirx variable.

SUB A, B ; Move left.

MOV [dirx], A ; Store the new value.

**Write a program that endlessly scrolls a long string of at least 20 characters in the first line of the text display.**

done:

MOV A, 2 ; Timer interrupt mask

OUT 2 ; Timer serviced

POP A ; Restore register

IRET ; Return from isr

main:

MOV SP, 0x0FFF ; Initialize the stack pointer

MOV B, s ; Pointer to the string

length:

MOVB CL, [B] ; Get the char

MOV A, [len] ; Get the length var

CMPB CL, 0 ; Check if we read the whole string

JE continue ; If so, continue

INC B ; Update the pointer

INC A ; Increase the length

MOV [len], A ; Save the length

JMP length

continue:

MOV B, s ; Set the pointer back to the beginning of the str

MOV D, 0x1000 ; Set the first address

MOV A, 1250 ; 1250 clock cycles = 0.25 sec

OUT 3 ; Set the reload time

MOV A, 2 ; Timer interrupt mask

OUT 0 ; Set the mask

STI ; Enable interrupts

HLT

; Function to clear the display

clear:

MOVB [D], 0

CMP D, 0x100F

JA ret

INC D

JMP clear

ret:

MOV D, 0x1000

RET

JMP main

JMP isr

s: DB "Hello sunshine"

DB 0

ptr: DW 1

len: DW 0

isr:

PUSH A ; isr will use reg A

do:

MOVB CL, [B] ; Get the char

CMPB CL, 0 ; Check if it is the end of the string

JE set ; If so, set the pointer back to the beginning

INC B ; Increase the pointer of the string

MOVB [D], CL ; Display the char on the address of D

INC D ; Increase the address

CMP D, 0x1010 ; Check if we came to the end of the display

JNE done ; If not, end of interrupt

MOV D, 0x1000 ; Otherwise, set the address at the beginning

JMP set ; Reset the printing of the string

JMP done

set:

MOV B, s ; Pointer to the first address of the string

ADD B, [ptr] ; Add the new pointer to the address

MOV C, [ptr] ; Move the pointer in C

INC C ; Increasse the pointer

MOV [ptr], C ; Save the pointer back

MOV D, 0x1000 ; Call the function clear(D)

CALL clear ; Clear the display

CMP C, [len] ; Check if the pointer reached the length of the string

JE reset\_ptr ; Set the pointer to its initial value

JMP do ; Get back to the display

reset\_ptr:

MOV B, 1 ; The initial value of the pointer

MOV [ptr], B ; Update it in the var

MOV B, s ; Set the pointer at the beginning of the string

JMP do ; Continue displaying

**Animate a progress bar that starts with an empty text display and adds a star symbol ‘\*’ every 0.25 seconds, until the first line of the text display is full. The program then terminates.**

main:

MOV SP, 0x0FFF ; Initialize stack pointer

MOV D, 0x1000 ; Get the address

MOVB [D], '\*' ; Display '\*'

MOV A, 1250 ; Timer value 1250 clock cycles

OUT 3 ; Through TMRPRELOAD

MOV A, 2 ; Timer mask interrupt

OUT 0 ; Set the mask

STI ; Enable interrupts

loop:

MOV A, [q] ; Get global variable quit

CMP A, 1 ; Check if quit = true

JNE loop

CLI ; Disable interrupts

HLT

JMP main

JMP isr

q: DW 0 ; quit = false

isr:

PUSH A ; The isr is using register A

MOVB AL, [D] ; Get the char from the address

INC D ; Increase the address

MOVB [D], AL ; Display the char in the next address

CMP D, 0x100F ; Has the address reached the end?

JB done

MOV [q], 1 ; Set the quit variable to true

done:

MOV A, 2 ; Set the timer interrupt

OUT 2 ; End of timer interrupt

POP A ; Restore the register

IRET ; Return form isr

**Write a function set\_pixel(x, y, color) that receives 8-bit values x and y through registers AL and AH, 8-bit value color through register CL, and draws a pixel at the screen location (x, y) in the given color color.**

JMP main

main:

MOV SP, 0x0FFF ; initialize the stack pointer

MOVB AL, 0x13 ; value for x

MOVB AH, 0x55 ; value for y

MOVB CL, 252 ; set the color

MOV B, A ; store the address

CALL set\_pixel ; call the function

HLT

set\_pixel:

MOV A, 2 ; set the bitmap mode

OUT 7 ; through the red VIDMODE

MOV A, 3 ; clear the video

OUT 7 ; through VIDMODE

MOV A, B ; get the address

OUT 8 ; set through VIDADDR

MOVB AL, CL ; get the color

OUT 9 ; set through VIDDATA

RET

**Write a function rectangle(x0, y0, x1, y1, color) that receives through register A the location (x0, y0) of the upper left corner, through register B the location (x1, y1) of the lower right corner, through register CL the color color, and draws a filled rectangle with the given properties.**

**Write a program that draws the entire 3-3-2 palette to the first line of the screen. Then extend this program so that the palette is repeated in each line.**

stop:

HLT

turn:

CMP D, 0xFFFF ; check if we reached the rightmost address

JE stop

MOV B, 0 ; new line

JMP loop

MOV A, 2 ; set the bitmap mode

OUT 7 ; through VIDMODE

MOV A, 3 ; clear the VRAM

OUT 7

MOV D, 0 ; the first address

MOV B, 0 ; color counter

loop:

MOV A, D ; get the address

OUT 8 ; set the address in VIDADDR

MOV A, B ; get the color

OUT 9 ; set the color in VIDDATA

INC D ; increase the address

INC B ; increase the color counter

CMP B, 255 ; have we gone through every color

JA turn ; if yes next row

JMP loop

JMP main

valueA: DW 0x1234

len: DW 0

turnVal: DW 0

rectangle:

MOV A, 2

OUT 7

MOV A, 3

OUT 7

loop:

MOV A, [valueA] ; get the address

CMP A, D ; compare it with top right corner

JAE doneLen ; we computed the width

OUT 8 ; otherwise set the address

INC A ; increase the address

MOV [valueA], A ; save address value

MOVB AL, CL ; set color

OUT 9 ; through VIDDATA

MOV A, [len] ; get the length var

INC A ; increase the length

MOV [len], A ; save it back

JMP loop

turn:

MOV A, [valueA] ; get the current address

ADD A, [turnVal] ; add the value to obtain the next row address

MOV [valueA], A ; save the value

MOV D, 0 ; set the counter to 0

JMP loop2 ; start printing again

done:

RET

main:

MOV SP, 0x0FFF ; initialize the stack pointer

MOV A, 0x1234 ; (x0, y0)

MOV B, 0x9876 ; (x1, y1)

MOVB CL, 28 ; set the color

; Get the upper right corner in D

MOVB DH, AH ; x0

MOVB DL, BL ; y1

CALL rectangle

HLT

doneLen:

MOV A, 256 ; the whole bitmap

SUB A, [len] ; subtract the length to get the turn value

MOV [turnVal], A ; save the turn value for next row

MOV A, [valueA] ; get the address

ADD A, [turnVal] ; set the address for the next row

MOV [valueA], A ; save the address

MOV D, 0 ; set the counter to 0

; Loop to paint the whole rectangle

loop2:

CMP D, [len] ; have we reached the full length

JAE turn ; turn to the next row

INC D ; increase the counter

MOV A, [valueA] ; get the address

CMP A, B ; compare it with the bottom right corner (last address)

JAE done ; if it's equal, we finish

OUT 8 ; pass the address through VIDDADDR

INC A ; increase the address

MOV [valueA], A ; save the value

MOVB AL, CL ; get the color

OUT 9 ; set the color through VIDDATA

JMP loop2

**Write a function convert\_color(red, green, blue) that receives 8-bit RGB color components through stack and returns in register CL the closest color approximation within the 3-3-2 color palette. Hint: Consider only the most significant bits of the given RGB color components.**

JMP main

ret:

PUSH D

RET

main:

MOV SP, 0x0FFF

PUSHB 0x30 ; blue

PUSHB 0x22 ; green

PUSHB 0xFA ; red

CALL convert\_color

HLT

convert\_color:

POP D ; return value

POPB CL ; get value for red

ANDB CL, 0xD0 ; mask the most significand bits

POPB BL ; get value for green

ANDB CL, 0xD0 ; mask the 3 most significand bits

SHRB BL, 3 ; move 3 bits down

ORB CL, BL ; store them in same register with or

POPB AL ; get value for blue

ANDB AL, 0xC0 ; mask 2 most significand bits

SHRB AL, 6 ; move 6 bits down

ORB CL, AL ; store them in same register with or

**Extend the “Hello world!” program in the following way. Define an additional array colors that holds 12 color indices. Then print out the “Hello world!” string with each character being of the corresponding color from the colors array.**

OUT 9 ; through VIDDATA

INC B ; increase pointer of the string

ADD D, 2 ; Update screen cell

MOV A, [index] ; get the index value

INC A ; increase the index

MOV [index], A ; save the value of the index

JMP loop

return:

RET

main:

MOV SP, 0x0FFF ; initialize the SP

MOV B, string ; Pointer to the string

MOV C, colors ; pointer to the array

CALL print

HLT

print:

MOV A, 1 ; set the text mode

OUT 7 ; through VIDMODE reg

MOV D, 0 ; screen cell

loop:

MOV A, D ; activate screen cell

OUT 8 ; through VIDADDR

MOV A, [index] ; get the index value

MUL 2 ; multiply the index

MOV C, colors ; pointer to the first address of the array

ADD C, A ; add the index to get the i-th element

MOV A, [C] ; get the color

MOVB AH, [B] ; get the char

CMPB AH, 0 ; have we read the whole string

JE return

JMP main

string: DB "Hello world!"

DB 0

colors: DW 253

DW 2

DW 53

DW 25

DW 7

DW 255

DW 213

DW 123

DW 212

DW 234

DW 255

DW 255

index: DW 0

**Write a function print\_rainbow(s) that prints out the string s, so that each letter is of a random color.**

JMP main

s: DB "Good morning!"

DB 0

**Write a function print(s, color) that prints**

**out the string s in the given color color.**

print\_rainbow:

MOV A, 1 ; set the text mode

OUT 7 ; through VIDMODE

MOV D, 0 ; screen cell

loop:

MOV A, D ; get the screen cell

OUT 8 ; set through VIDADDR

MOVB CH, [B] ; set the char

CMPB CH, 0 ; check if we read the whole string

JE ret

IN 10 ; read random number

AND A, 0x00FF ; mask it to get 8 bits

MOVB AH, CH ; prepare the register A

OUT 9 ; set color and char through VIDDATA

INC B ; increase the pointer to the string

ADD D, 2 ; increase the screen cell

JMP loop

ret:

RET

main:

MOV SP, 0x0FFF ; initialize the stack pointer

MOV B, s ; pointer to the string

CALL print\_rainbow

HLT

JMP main

print:

MOV A, 1 ; value of text mode

OUT 7 ; through VIDMODE

MOV D, 0 ; starting address

loop:

MOV A, D ; get the address

OUT 8 ; VIDADDR

MOVB AH, [C] ; get the first char of the string

CMPB AH, 0 ; check if the string is written

JE ret ; if yes return

MOVB AL, BL ; otherwise set the color

OUT 9 ; VIDDATA

INC C ; update the pointer for the next char

ADD D, 2 ; update the 2B address space for a char

JMP loop

ret:

RET

main:

MOV SP, 0x0FFF ; move the stack pointer

MOV C, s ; pointer to the string

MOVB BL, [color] ; value of the color

CALL print

HLT

s: DB "Sonce"

DB 0

color: DB 194

**Print the transposed version of the ASCII matrix, so that the characters are printed from top to bottom, and new column is started when the bottom edge of the screen is reached.**

JE break ; if yes break the loop

JMP loop

do:

MOV D, [i] ; get the index(screen cell)

ADD D, 2 ; add 2 to get the next column

MOV [i], D ; store back

MOV C, 0 ; reset the counter

JMP loop

break:

HLT

i: DW 0

OUT 8 ; through VIDADDR

MOV A, 0

MOVB AH, BL ; set the first caharcter

MOVB AL, 255 ; set the color to white

OUT 9 ; through VIDDATA

ADD D, 256 ; get the next row

INC B ; increase the pointer

INC C ; increase the counter

CMPB BL, 0 ; have we cycled through the whole ASCII

MOV A, 1 ; set the text mode

OUT 7 ; through VIDMODE reg

MOV D, 0 ; screen cell

MOV C, 0 ; counter

MOVB BL, 0 ; characters

loop:

CMP C, 16 ; if we reached the bottom

JE do

MOV A, D ; activate the cell

**Write a function writeln(s) that prints out the given string in the new line, while it preserves all the text printed out with the previous calls of this function. If the newly printed line is below the bottom screen border, it automatically scrolls down one line to make the newly printed line visible**.

OUT 9

RET

main:

MOV SP, 0x0FFF ; initialize the stack pointer

MOV A, 1 ; set the text mode

OUT 7 ; through VIDMODE

MOV D, 0 ; screen cell

MOV C, [ctr] ; counter for calling the function

while:

CMP C, 15 ; for 15 lines

JE exit

CALL writeln ; call the function

MOV C, [ctr] ; get the value of the counter

INC C ; increase

MOV [ctr], C ; store back

JMP while

exit:

HLT

ADD D, 2 ; update the screen cell

INC B ; increase the pointer

JMP loop

ret:

MOV D, [new\_line] ; get the value for the new line

ADD D, 256 ; add 256=new line

MOV [new\_line], D ; store back

MOV A, [counter] ; then get the counter for the lines

INC A ; increase

MOV [counter], A ; store back

CMP A, 15 ; if we reached the bottom

JE scroll ; if yes scroll

RET

scroll:

MOV A, 0xA304 ; vertical scroll information

OUT 8 ; through VIDADDR

MOV A, 16 ; 16 pixels = 2 new lines

JMP main

new\_line: DW 0

ctr: DW 0

counter: DW 0

s: DB "FAMNIT"

DB 0

writeln:

MOV B, s ; pointer to the string

loop:

MOVB CL, [B] ; get the char

CMPB CL, 0 ; if we reached the end of the string

JE ret

MOV A, D ; activate the cell

OUT 8 ; through VIDADDR

MOVB AL, 255 ; set the color to white

MOVB AH, CL ; set the char

OUT 9 ; through VIDDATA

**Display the character with ASCII code 255 (the ring shape) at position (10, 0) on the screen. Let the character fall down one line every 0.5 seconds (25 frames). When it reaches the bottom line, reset its position back to (10, 0). Let the program delay the animation by 2 seconds, after the character has been initially drawn. Extend the solution of Task 9.8, so that there are two objects, which are moving in opposite directions, i.e. one moving up and one moving down.**

MOV A, D ; the current object position

OUT 8 ; activate the cell

MOV A, 0 ; overwrite the cell with 0

OUT 9 ; to erase the obeject 1

MOV A, B ; the current object position

OUT 8 ; activate the cell

MOV A, 0 ; overwrite the cell with 0

OUT 9 ; to erase the object 2

ADD D, 256 ; move one line down

SUB B, 256 ; move one line up

CMPB DH, 16 ; is the object in the 16-th line

JB noreset ; if not, do not reset its position

MOV B, 0x0F14 ; otherwise put it back to cell

MOV D, 0x000A ; otherwise put it back to cell

noreset:

MOV A, D ; the new object position

OUT 8 ; activate the cell

MOVB AH, 255 ; the ring shape

MOVB AL, 255 ; white color

OUT 9 ; print the character

MOV A, B ; second object

OUT 8 ; activate the cell

MOVB AH, 255 ; the ring shape

MOVB AL, 255 ; white color

OUT 9 ; print the character

MOV C, 25 ; wait for 25 frames

CALL wait\_frames ; call wait\_frames (25)

JMP loop

HLT

MOV A, 1 ; set the graphics card to TEXT mode

OUT 7 ; through register VIDMODE

; initialize the object position

MOV A, 0xA300 ; background information

OUT 8

MOV A, 92 ; background color

OUT 9

MOV D, 0x000A ; address of the first shape: cell is (10, 0)

MOV A, D ; activate the cell

OUT 8 ; through VIDADDR

MOVB AH, 255 ; the ring shape

MOVB AL, 255 ; white color

OUT 9 ; print the character

MOV B, 0x0F14 ; address of the second shape: cell is (20, 15)

MOV A, B ; set the address of the second shape

OUT 8 ; through VIDADDR

MOVB AH, 255 ; the ring shape

MOVB AL, 255 ; white color

OUT 9 ; print the character

; enable graphics card interrupt

MOV A, 4 ; mask the graphics card interrupt

OUT 0 ; enable the graphics card interrupt

STI ; enable interrupts globally

; wait for two seconds before the animation

MOV C, 100

CALL wait\_frames ; Call wait\_frames (100) .

; endter the animation loop

loop:

JMP main

JMP isr

; the vertical sync signal

vsync: DW 0

; the ISR to serve the graphics card interrupt request

isr:

PUSH A ; ISR will use register A.

MOV [vsync], 1 ; Set the vsync flag .

MOV A, 4 ; Clear the graphics card interrupt request

OUT 2 ; through the I/O register IRQEOI .

POP A ; Restore the original value of register A.

IRET ; Return from interrupt .

; function wait\_frames (count) waits for count frames

; parameter count is given through register C

wait\_frames:

wait\_frames\_loop:

MOV A, [vsync] ; Check the current vsync value .

CMP A, 0 ; If still 0 ,

JE wait\_frames\_loop ; check it again .

MOV [vsync] , 0 ; The vsync signal received . Reset it.

DEC C ; Count the received frame .

CMP C, 0 ; When enough frames passed ,

JE wait\_frames\_return ; return from the function .

JMP wait\_frames\_loop ; Otherwise , wait another frame .

wait\_frames\_return:

RET

main:

MOV SP, 0x0FFF ; initialize the stack pointer

; set the graphics card

**Place the Pacman and the four ghosts on the screen at random positions. The fours ghosts are Blinky (RGB: 255, 0, 0), Pinky (RGB: 255, 184, 255), Inky (RGB: 0, 255, 255) and Clyde (RGB: 255, 184, 82).**

JMP loop\_gos

; redefining colors

change\_colors:

; Blinky

MOV A, 0xA2F4

OUT 8

MOV A, 0xFF00

OUT 9

MOV A, 0xA2F5

OUT 8

MOV A, 0x0000

OUT 9

; Pinky

MOV A, 0xA027

OUT 8

MOV A, 0xFFB8

OUT 9

MOV A, 0xA028

OUT 8

MOV A, 0xB8FF

OUT 9

; Inky

MOV A, 0xA240

OUT 8

MOV A, 0x00FF

OUT 9

MOV A, 0xA241

OUT 8

MOV A, 0xFFFF

OUT 9

;Clyde

MOV A, 0xA114

OUT 8

MOV A, 0xFFB8

OUT 9

MOV A, 0xA115

OUT 8

MOV A, 0xB852

OUT 9

HLT

OUT 9

INC D ; pointer to the next char

IN 10

AND A, 0x0F1E

OUT 8

MOVB AH, [D]

MOVB AL, 92

OUT 9

MOV B, 0x8820 ; save address of char 'A' in B

MOV C, 16 ; set the counter

MOV D, pac ; pointer to the figure

loop\_pac:

CMP C, 0 ; check if we changed all the data

JE go\_gos ; go to the next figure

DEC C ; decrease the counter

MOV A, B ; set the address of char A

OUT 8 ; VIDADDR

MOV A, [D] ; get the current word (16-bit)

OUT 9 ; VIDDATA

ADD D, 2 ; next word in ram

ADD B, 2 ; next VRAM address

JMP loop\_pac

go\_gos:

MOV B, 0x8840

MOV C, 16

MOV D, gos

loop\_gos:

CMP C, 0

JE change\_colors

DEC C

MOV A,B

OUT 8

MOV A, [D]

OUT 9

ADD D, 2

ADD B, 2

JMP main

pac: DB "\x00\x00\x03\xE0\x0F\xF8\x1F\xFC\x1F\xFC\x3F\xF0\x3F\x80\x3C\x00"

DB "\x3F\x80\x3F\xF0\x1F\xFC\x1F\xFC\x0F\xF8\x03\xE0\x00\x00\x00\x00"

gos: DB "\x00\x00\x03\xC0\x0F\xF0\x1F\xF8\x33\xCC\x21\x84\x2D\xB4\x6D\xB6"

DB "\x73\xCE\x7F\xFE\x7F\xFE\x7F\xFE\x7F\xFE\x7B\xDE\x31\x8C\x00\x00"

pac\_and\_gos: DB "ABBBB\x00"

main:

MOV A, 1 ; text mode

OUT 7 ; VIDMODE

IN 10 ; get random number

AND A, 0x0F1E ; mask it to a valid address

OUT 8 ; VIDADDR

MOV D, pac\_and\_gos ; pointer to the string

MOVB AH, [D] ; set the char

MOVB AL, 222 ; set the color

OUT 9 ; VIDDATA

INC D ; pointer to the next char

IN 10

AND A, 0x0F1E

OUT 8

MOVB AH, [D]

MOVB AL, 252

OUT 9

INC D ; pointer to the next char

IN 10

AND A, 0x0F1E

OUT 8

MOVB AH, [D]

MOVB AL, 13

OUT 9

INC D ; pointer to the next char

IN 10

AND A, 0x0F1E

OUT 8

MOVB AH, [D]

MOVB AL, 192

**Display two ghosts, one magenta and one yellow colored. Make the yellow ghost slowly disappear. Let the animation of disappearing be synchronized with the VSync signal.**

DEC C ; decrease the counter

MOV A, B ; set the address of char A

OUT 8 ; VIDADDR

MOV A, [D] ; get the current word (16-bit)

OUT 9 ; VIDDATA

ADD D, 2 ; next word in ram

ADD B, 2 ; next VRAM address

JMP loop

;changing the shape B

change\_b:

MOV B, 0x8840 ; save address of char 'A' in B

MOV C, 16 ; set the counter

MOV D, gos ; pointer to the figure

loop\_b:

CMP C, 0 ; check if we changed all the data

JE dissapear ; go to the next figure

DEC C ; decrease the counter

MOV A, B ; set the address of char A

OUT 8 ; VIDADDR

MOV A, [D] ; get the current word (16-bit)

OUT 9 ; VIDDATA

ADD D, 2 ; next word in ram

ADD B, 2 ; next VRAM address

JMP loop\_b

dissapear:

MOV A, 4 ; mask the graphics card iterrupts

OUT 0 ; enable the graphicks card interrupts

STI ; enable interrupts globally

MOV B, 0x8820 ; access the definition of char 'A'

MOV C, 16 ; set the counter

loop\_d:

CMP C, 0 ; check if we chnaged all the data

JE done ; if yes then the figure has dissapeared

MOV A, B ; get the address

OUT 8 ; VIDADDR

MOV A, 0 ; value to change current definition

OUT 9

ADD B, 2 ; update the address

DEC C ; decrease the counter

CALL wait\_next\_frame ; call the wait function

JMP loop\_d

done:

HLT

JMP main

JMP isr

gos: DB "\x00\x00\x03\xC0\x0F\xF0\x1F\xF8\x33\xCC\x21\x84\x2D\xB4\x6D\xB6"

DB "\x73\xCE\x7F\xFE\x7F\xFE\x7F\xFE\x7F\xFE\x7B\xDE\x31\x8C\x00\x00"

vsync: DW 0

isr:

PUSH A ; ISR will use register A.

MOV [vsync], 1 ; Set the vsync flag .

MOV A, 4 ; Clear the graphics card interrupt request

OUT 2 ; through the I/O register IRQEOI .

POP A ; Restore the original value of register A.

IRET ; Return from interrupt .

wait\_next\_frame:

PUSH A ; the function will use register A

wait\_next\_frame\_loop:

MOV A, [vsync] ; check the current vsync value

CMP A, 0 ; if still 0

JE wait\_next\_frame\_loop ; check it again

MOV [vsync], 0 ; the vsync signal received. Reset it

POP A ; restore the original value of register A

RET

main:

MOV SP, 0x0FFF ; initialize the stack pointer

MOV A, 1 ; text mode

OUT 7 ; VIDMODE

MOV A, 0x0706 ; address for the first figure

OUT 8 ; VIDADDR

MOV A, 0x41FE ; char 'A' in color FE

OUT 9 ; VIDDATA

MOV A, 0x0A08 ; address for the second figure

OUT 8 ; VIDADDR

MOV A, 0x4234 ; char 'B' in color 34

OUT 9 ; VIDDATA

;changing the shape A

MOV B, 0x8820 ; save address of char 'A' in B

MOV C, 16 ; set the counter

MOV D, gos ; pointer to the figure

loop:

CMP C, 0 ; check if we changed all the data

JE change\_b ; go to the next figure

**Revisit the solution of Task 9.8. Recall that we have animated an object, which was falling down, one text cell at a time. Change the program in such a way, that the object will fall continuously, 1 pixel per frame. Add some text, so that the object falls through it. a) Add another sprite to the solution of Task 9.13 that moves up with a different speed than the first sprite. b) Change the shape of the falling object in Task 9.13 to resemble the ghost from Figure 4.3. c) In Task 9.13 also animate the text by scrolling the window 1 pixel per frame.**

; Set the initial sprite position .

MOV A , 0xA308 ; VRAM address of sprite 1 screen position .

OUT 8 ; Activate the address .

MOV A , 0x7800 ; Let the initial position be (120px, 0px).

OUT 9 ; Set the sprite position .

; chnage definition

MOV D, 0x9FE0 ; change the definition of the ring

MOV C, gos ; pointer to the string

MOV B, 16 ; counter

change:

CMP B, 0 ; if the change is done

JE done

MOV A, D ; if not set the address

OUT 8 ; through VIDADDR

MOV A, [C] ; get the char

OUT 9 ; VIDDATA

ADD D, 2 ; update the address

ADD C, 2 ; update the pointer

DEC B

JMP change

done:

;enable the graphics card interrupts

MOV A, 4 ; mask the graphics card interrupts

OUT 0 ; enable the graphics card interrupts

STI ; enable interrupts globally

MOV B, 0xA302 ; the VRAM address of the HScroll information

MOV D, 0 ; let the offset be initially 0

; Enter the animation loop .

loop:

MOV A , 0xA308 ; VRAM address of sprite 1 screen position .

OUT 8 ; Activate the address .

IN 9 ; Get the current sprite 1 screen position .

INCB AL ; Increase the y position by 1 pixel .

OUT 9 ; Set the new sprite 1 screen position .

CALL wait\_frames ; Call wait\_frames (1) .

MOV A, B ; set the HScroll info

OUT 8 ; VIDADDR

MOV A, D ; set the offset

OUT 9 ; VIDDATA

INC D ; prepare the new offset value

CMP D, 400 ; if the offset has reached 400 pixels

JB offset\_continue

MOV D, 0 ; reset the offset back to 0

offset\_continue:

CALL wait\_frames

JMP loop ; Repeat for the next frame .

HLT

JMP main

JMP isr

gos: DB "\x00\x00\x03\xC0\x0F\xF0\x1F\xF8\x33\xCC\x21\x84\x2D\xB4\x6D\xB6"

DB "\x73\xCE\x7F\xFE\x7F\xFE\x7F\xFE\x7F\xFE\x7B\xDE\x31\x8C\x00\x00"

vsync: DW 0

; The ISR to serve the graphics card interrupt requests .

isr:

PUSH A ; ISR will use register A .

MOV [vsync] , 1 ; Set the vsync flag .

MOV A , 4 ; Clear the graphics card interrupt request

OUT 2 ; through the I / O register IRQEOI .

POP A ; Restore the original value of register A .

IRET ; Return from interrupt .

wait\_frames:

PUSH A

wait\_frames\_loop:

MOV A , [vsync] ; Check the current vsync value .

CMP A , 0 ; If still 0 ,

JE wait\_frames\_loop ; check it again .

MOV [vsync] , 0 ; The vsync signal received . Reset it .

POP A ; restore the original value of register A

RET

; Some text to appear behind the falling object .

str: DB "Systems 1"

DB 0

main:

MOV SP , 0x0FFF ; Initialize the stack pointer .

; Set up the graphics screen .

MOV A , 1 ; Set the graphics card to TEXT mode

OUT 7 ; through register VIDMODE .

; Print the string .

MOV C , str ; Pointer to string .

MOV D , 0x0720 ; Pointer to text cell .

str\_loop:

MOV A , D ; Activate the current text cell

OUT 8 ; through I / O register VIDADDR .

MOVB AH , [C] ; Get the current character .

CMPB AH , 0 ; If at the end of the string ,

JE str\_break ; break the loop .

MOVB AL , 252 ; Set the yellow color .

OUT 9 ; Print the character .

INC C ; Next character .

ADD D , 2 ; Next text cell .

JMP str\_loop

str\_break:

; Display the object as sprite 1.

MOV A , 0xA306 ; VRAM address of sprite 1.

OUT 8 ; Activate the address .

MOV A , 0xFFFF ; White ring shape .

OUT 9 ; Set the sprite shape / color .